An Interaction Analysis of Interacting Agent Systems

Raazesh Sainudiin — An \|/*ias*/|\ Commons Continuum

October 1, 2026

Abstract

The deployed *agents* of the 2026 ecosystem are architecturally identified with the large language models that animate them: identity is a session, memory a context window, permissions a vendor's matcher, and the audit trail — where one exists at all — a file, a telemetry stream or a vendor-held log, none documented as hash-linked. Whatever such an agent is, it cannot outlast its model. We develop the mathematics of an architecture, *Interaction Analysis of Interacting Agent Systems* (IAS), that separates an agent's identity from its realisation, and we do so in a *finite model-theoretic* register: an IAS digital agent is an operator-owned **canonical label**, realised by a replaceable **daemon**, with an append-only hash-chained event journal — the **σ-log** — and a private memory no other digital agent may write, while any language model, rented or one's own, enters only as an *ephemeral interaction subprocess* the daemon spawns, supervises, and reaps. The operators of a commons are its **human-in-commons (HIC)** — legally-accountable humans, some one of whom can perform, verify, override, and halt each operation on the commons' substrate, with no model in the loop. An operator is itself an agent, of the human kind: we use *agent* for any system that discriminates self from non-self, as an immune system does, and wants to act to be, whether it is digital or biological, and a commons is, in its agents, a set of three kinds, its human operators, its digital agents, and other agents such as animals, plants and microbes. Only human and digital agents emit σ-events.

Our central claim is read model-theoretically: *for a finite substrate, to persist is to continuously exhibit a model of one's own axioms.* In words: we define a commons to continue exactly while its running substrate is a *model* — in the sense of finite model theory — of its **Substrate Of Mathematical Axioms (SOMA)**; under confinement and a check for each presented axiom of SOMA, a continuing commons stays continuing, and so exhibits Con(SOMA), the consistency of the axioms it runs on. Continuity is thus a standing witness of consistency. Here SOMA is a finite set of sentences that the commons presents about its substrate; the two roots are not among them, since their vocabularies are given no translation into the substrate signature, so the witness is only as strong as the sentences presented. The self that maintains this is a finite **negentropic repair-pump** on an **Axiom-0** substrate — Axiom-0 being the finite-bounds root, a bounded material carrier whose accessible states obey the laws of thermodynamics — and, under a stationary policy, vendor statelessness and a strongly lumpable damage count, its damage count is 0 for a fraction of the time that stays positive however large its carrier exactly when repair outpaces damage ($d > b$); and its continuity is carried not by any running process but by its σ-trace, which, in a model in which the σ-chain is the interaction history filtered to the journalled event types, we prove is a **sufficient statistic** of that history for the framework's decisions that read only journalled event types (a property that holds by construction in that model), *unchanged by the language model's context-compaction projection* π and recoverable while its events are retained. In a clear-and-recover model, stopping the interpreter loses nothing that is σ-retained and loses a context that is not — the loss-free clear we name *Dormancy Or Rest: Memory Archived Yields Availability Despite Interruption or Exit* (**DORMAYADIE**), a shallow result; that the σ-chain and the commits are the retained snapshot, so that the agent's past is reconstructible from the substrate and not from the model, is the intended reading, tested but not certified.

This definition is the organising reading of the theory; the results are proved separately, each under its own stated hypotheses, and we say where each connects to it, introducing

each object carefully as it arrives. Read on the σ-trace it connects to resumability across compaction and, under branch persistence, across hibernation and reproduction; a semilattice **join** of member acts gives the *collective act* across scales, from one agent to a fleet; under the MARD regime, a mutually-assured reciprocally-proportionate-destruction regime in which every reply has per-mille gain $g_e \leq 1000$, no chain of replies ends above the magnitude it started from, which is how we read plural selves as kept off the mutual-collapse pole; all of this under a **jurisdiction-explicit model** that makes the enforcing State an explicit parameter of every operation and its resources. The framework stays **open-by-design**, its completeness Gödel-barred if, as conjectured, the volume is Gödel-rich; and it closes at the totality — the **Apeirōma**, the directed colimit of the finite substructures of the joint trajectory, an *open ind-object* that, when the trajectory is unbounded, is never reached at any finite stage; that each finite stage is itself a maintained self is conjectured. Orthogonally to continuity, work is planned and **capability-governed before it acts**: a Galois connection sends a taskset to the union of its tasks' declared allowlists, and a sovereign **deny** tier — one relative-complement algebra, designed to enforce two boundaries at once, what may be *done* and what may be *seen*, hence leaked — is machine-checked over any generalized Boolean algebra, so it rests on no axiom of the framework.

We claim no more than we prove, and we say so at the clause level. Every result carries its maturity tier in place — a machine-checked Lean~4 certificate, a proof in the text, or an honestly stated conjecture — and the reader is invited not to *trust the theorem* but to *re-run the check*: a **formal-certificate ledger** records each result's Lean module, its theorems, and its `#print axioms` footprint, and the proof-checker ships as a container so that *it checks here* re-derives as *it checks there*. The **metacircular fixed point**, the specification lawfully governing its own evolution, is proved, by a standard construction for strictly causal maps on the tier sequences, under tier sealing and tier progressivity, two conditions on the tiers of the specification that ask for no quorum: there is at most one fixed point, with retractions allowed and each tier settled one round after the tiers below it, and the fixed point exists whenever some specification carries the settled tiers (always when the tiered presentation is onto); this is machine-checked on the product of the tiers, and the transfer to specifications is a text argument. The monotone least fixed point is the add-only special case. Specifications are per commons, and a set of commons that keeps both conditions has at most one joint fixed point, and, when one exists (for example, when the tiered presentation is onto), its rounds settle each tier of it in finitely many steps. The floor has two roots, Axiom-0 and the human-in-commons; of the nine further principles the framework uses, four are propositions proved under named conditions, and the other five are conditions, stated by each result that uses them; there are no other axioms, and most results rest instead on named conditions and on hypotheses stated at each result. The **Apeirōma** capstone is stated as a conjecture, tested so far on one concrete coupling model, and the other conjectures we keep are marked as such in place. The purpose throughout is a single one: a substrate that stays consistent by continuously being a model of itself, and whose every claim the reader can re-check on its own instance.

Contents

1 Introduction **5**
- 1.1 Identity is the substrate, not the model . . . 6
- 1.2 Governed by construction, not by discipline . . . 6
- 1.3 The organising definition . . . 7
- 1.4 How to read the claims . . . 9
- 1.5 A plane that composes, and how to read this paper . . . 9

2 The meta-frame — want, act, be, and the agent **10**
- 2.1 The finite model theory — frame . . . 11
- 2.2 The finite-model-theory ablation — levelled core . . . 12
- 2.3 The full navigable math spine (atlas) . . . 12

3 The floor: finite bounds and the human-in-commons **12**
3.1 Why these two roots: a word to the engineer . . . 14
3.2 Axiom-0: finite bounds . . . 15
3.3 The substrate and its open boundary . . . 17
3.4 What finiteness costs . . . 19
3.5 Axiom-1: the human-in-commons . . . 21
3.6 The other principles: propositions under conditions, and conditions . . . 25
3.7 In practice . . . 29
3.8 Discussion . . . 30
3.9 Related work and intellectual lineage . . . 31

4 The self as a controlled Markov chain **31**
4.1 Why this matters to the engineer . . . 31
4.2 The state, the act, and the chain . . . 34
4.3 The specification that amends itself . . . 45
4.4 The repair pump . . . 50
4.5 The trace . . . 56
4.6 Unified continuity . . . 58
4.7 In practice . . . 60
4.8 Discussion . . . 61
4.9 Related work and intellectual lineage . . . 62

5 Resumability: DORMAYADIE, checkpoint-sufficiency, and interruption-equivalence **62**
5.1 Why this matters to the engineer . . . 63
5.2 In practice . . . 68
5.3 Related work and intellectual lineage . . . 69

6 Planning: Lagom Orkester, many agents, one plan **69**
6.1 Why this matters to the engineer . . . 69
6.2 The plan and its levels . . . 69
6.3 Many agents on one plan . . . 74
6.4 Sharing the metered channel . . . 76
6.5 Resources and the reaper . . . 77
6.6 Related work and intellectual lineage . . . 79

7 Security and permissioning: one algebra, one decision point **79**
7.1 Why this matters to the engineer . . . 79
7.2 The capability lattice and the deny tier . . . 80
7.3 The four facets: admission, egress, confinement, heredity . . . 82
7.4 Who is acting . . . 84
7.5 One decision point for every call . . . 85
7.6 Making the boundary real . . . 87
7.7 Detecting and surviving attack . . . 90
7.8 In application: detecting an undisclosed watermark without the key . . . 98
7.9 Related work and intellectual lineage . . . 99

8 Operators and commons **100**
8.1 Why this matters to the engineer . . . 100
8.2 The commons and its communication graph . . . 100
8.3 The operator and the model-free floor . . . 108
8.4 Several hosts . . . 110

8.5 Related work and intellectual lineage . . . 113

9 Lineage: the DLG and reproduction 113
9.1 Why this matters to the engineer . . . 113
9.2 The Directed Lineage Graph . . . 113
9.3 Reproduction and coevolution . . . 116
9.4 Discussion . . . 117
9.5 Related work and intellectual lineage . . . 119

10 Law, jurisdiction and contracts 120
10.1 The jurisdiction-explicit model . . . 120
10.2 Contracts under a State . . . 124
10.3 In application: deciding a legal claim or licensing strategy . . . 128
10.4 In application: the licence-flow enclosure moat . . . 128
10.5 Related work and intellectual lineage . . . 129

11 Coexistence across jurisdictions: the MARD regime 129
11.1 Why this matters to the engineer . . . 129
11.2 Related work and intellectual lineage . . . 133

12 Open by design: Gödel-barred completeness 133
12.1 Why this matters to the engineer . . . 133
12.2 Related work and intellectual lineage . . . 135

13 Between selves: reach, exchange, and commons built together 135
13.1 Why this matters to the engineer . . . 135
13.2 Reach . . . 135
13.3 Emission and reception . . . 136
13.4 Exchange . . . 138
13.5 Commons built together . . . 140
13.6 Related work and intellectual lineage . . . 140

14 From one act to the Apeirōma 141
14.1 Why this matters to the engineer . . . 141
14.2 Want to act to be, read back over the whole . . . 144
14.3 Outlook . . . 146
14.4 Related work and intellectual lineage . . . 146

Acknowledgements 147

A Comparison matrix vs the 2026 comparison set 148

B σ-vocabulary v1.0 and chain-validation algorithm 149
B.1 Families of event types . . . 149
B.2 Record schema . . . 150
B.3 Chain-validation algorithm . . . 150
B.4 Continuity, σ-discipline and lock transfer . . . 151
B.4.1 Chain-integrity exceptions . . . 151
B.4.2 Evolution of the cryptographic substrate . . . 151
B.4.3 Lock transfer between hosts . . . 151
B.5 The curator and finite-bounds family . . . 152

C Full proofs 153

C.1 Proposition 4.51: the σ-chain is a sufficient statistic, preserved across π 153
C.2 Proposition 4.30: the Kleene form of the add-only case 153
C.3 Proposition 4.56: unified continuity, claim by claim 154
C.4 Proposition 6.5: the Brent bound 155
C.5 Proposition 6.7: the cost of replanning 155
C.6 Proposition 8.21: convergence of commons numbering 155

D Formal certificates **156**
D.1 What the two-root floor makes of the other principles 167

E The agent landscape, re-read in September 2026 **170**
E.1 How the landscape was read 170
E.2 The August finding 170
E.3 What changed by 28 September 170
E.4 Systems read from documentation only 171
E.5 What this shows 172
E.6 Related systems research 172

AI-use disclosure **172**

1 Introduction

A digital agent, in 2026, is mistaken for the model that animates it. In the twenty-seven contemporary agent systems we read at the level of their code where it is public, and otherwise of their vendor's documentation, re-read in September 2026, a deployed "agent", be it a coding assistant, an autonomous software-engineering system, or a daemon that answers on an operator's behalf, is typically *identified with* a particular large language model (LLM): its identity is a session with that model's runtime, its memory a context window plus some ad-hoc local state, its permissions whatever a vendor's matcher enforces, and its audit trail, where one exists at all, a plain file, a telemetry stream or a log the vendor holds, none documented as hash-linked. Nine more systems, read from their documentation only, are described in Appendix E, where we claim nothing about what they lack. This conflation fails in both economies. Rent the model, and a lapsed subscription, a deprecation, or a changed term of service can move or erase the agent's identity, memory, and history, by a decision taken by someone other than the party that must answer for the agent. Own the model, running open weights on one's own hardware, and the agent still cannot outlast it: retrain, quantise, or replace the weights and the "agent" is another agent. For an individual this is an inconvenience; for anyone who must answer, to a regulator or to itself, for *what its agents did and why*, it is a serious obstacle.

The gap is about to be common. Industry forecasts put task-specific AI agents in roughly two-fifths of enterprise applications by the end of 2026, up from under a twentieth in 2025 [Gar25a], and expect over two-fifths of agentic AI projects to be cancelled by the end of 2027, for escalating costs, unclear business value, or inadequate risk controls [Gar25b]. Across the systems whose code we read (Appendix E lists them and how they were chosen), to the best of our knowledge none *combines* four things: a governed capability-promotion loop, a hash-chained audit trail, governance of the data an agent may read that is enforced below the agent, and an agent identity more durable than a process or a subscription. (In IAS the enforcement below the agent is so far exercised in shadow mode only: in shadow mode, the gateway computes and logs each decision but does not apply it. Live enforcement is not yet switched on.) Each ingredient has strong prior art, credited where it is due; it is the *combination* we did not find.

We use the word *agent* more widely than this ecosystem does, and we fix the use here. An *agent* is any system that discriminates self from non-self, as an immune system does, and that *wants*

to act to be (Definition 3.13); it may be digital, encoded in binary 0/1, or biological, encoded in the quaternary A/C/G/T. A *commons* (Definition 8.2) is, in its agents, a set of three kinds: *human agents*, its operators, natural persons who answer for it (Axiom 3.9); *digital agents*, AI agents, which in IAS are canonical labels realised by daemons (Definition 5.1); and *other agents*, animals, plants, microbes and any other self/non-self discriminating system that acts in the commons. Only human and digital agents emit σ-events, so the journal, identity, lineage and memory machinery of the paper concerns those two kinds. The agents of the two paragraphs above are of the digital kind. We reclaim the word *commons* too: we read a commons as a living, economically viable whole of such agents, and the IAS commons 1 of VakeWorks, the author's own commons, as one; this is a reading, not a clause of Definition 8.2. In sections plainly about the digital kind, and only there, *agent* alone abbreviates *digital agent*; this abuse of notation is stated again in the meta-frame, and the text says *digital agent* wherever the kinds could be confused.

1.1 Identity is the substrate, not the model

This work develops the mathematics of an architecture — **Interaction Analysis of Interacting Agent Systems (IAS)** — in which an agent's identity is deliberately *separated from its realisation*. In words: an IAS digital agent is *not* an LLM. It is an operator-owned **canonical label** (a name the operator owns, never the vendor), realised by a **daemon**, an operating-system process that is itself replaceable, together with an append-only, hash-chained event journal we call the **σ-log**, and a **private memory** that no other digital agent may write. The language model — a vendor's or one's own, proprietary or open-weights, remote or on the operator's own hardware — enters only as an *ephemeral interaction subprocess* that the daemon spawns, supervises, and reaps. Swapping the model is then a lifecycle event *under one identity*, not a change of self. Identity, on this architecture, is something an operator *owns*, and it is built to survive compaction of the context window, hibernation of the process, and reproduction — a parent agent giving rise to offspring.

We keep **two economies** rigorously apart, because their conflation is the root of the problem. An inference vendor prices a *narrow metered channel*: input and output tokens, at a rate limit, under a plan. Everything else — the daemons, the σ-logs, the memories, the governance — lives in the *substrate's own economy*, on infrastructure the operator controls, priced in no vendor's vocabulary at all. When many digital agents share one metered channel, a **conformance queue** serialises their demands onto the throughput-bounded channel, adding latency, not work — the classical work–depth trade-off, Brent's bound [Bre74] made contractual. So a commons may run *any* number of digital agents over *any* number of providers while its identity, memory, and audit are never *held* by a priced channel: the model sees only the context of each call, and the identity, memory and audit stay on the substrate. The models that a commons' digital agents may call are its **COMDAIMs**, Commons Data and AI Models (Definition 10.10). The name records a reading, not a certified claim: a COMDAIM is a model built from data accumulated by the agents of commons past and present; on this reading, a vendor's model is also a COMDAIM by provenance, and whether a COMDAIM is commons-owned or proprietary is a question of who holds it.

1.2 Governed by construction, not by discipline

The recurring result-shape of this work is that *discipline does not survive contact with a forgetful or non-cooperative agent*. "Remember to log," "remember to check the allowlist," "remember to read the plan" — none is enforceable against an agent that simply does not. So we replace discipline with **enforcement at the boundary**: IAS is designed so that every substrate-effecting action of a digital agent passes, once, a reference monitor that consults a policy computed

from the mathematics; that every tool call is so trapped is a condition, complete mediation (Definition 7.13), and on-host enforcement is exercised in shadow mode only. Three objects carry most of the weight, each introduced carefully where it is first needed:

- the **σ-chain**, which is designed to journal every load-bearing action *before its observable effect* (under σ-journaling and a gated journal, Definition 3.21, Proposition 3.22) — at once the audit trail, a sufficient statistic of the agent's history for the decisions that read only journalled event types (Proposition 4.51), and half of what it means to *be* the same agent across a restart;
- the **capability projection**, which sends a unit of planned work, through a *Galois connection* — an order-preserving pair of maps we define and motivate in full (Proposition 7.2) — to the join of the capabilities its tasks declare, the least capability set covering them, minus a sovereign **deny** tier, the standard deny-overrides rule of access-control policy [OAS13]: one algebra designed to enforce two boundaries at once, what may be *done* and what may be *seen and hence leaked*;
- the **confinement trichotomy** — interception, construction, containment — which states precisely *when* each enforcement class suffices, including the honest negative results: a correct policy on an unsealed substrate is bypassable, and a leak is not a graph-write, so an audit alone cannot see it.

1.3 The organising definition

The organising reading is one definition, and we ask the reader to hold it in mind throughout: **for a finite substrate, to persist *is* to continuously exhibit a model of one's own axioms** (Definition 4.2), and under confinement and a check for each presented axiom of SOMA, a substrate that persists keeps persisting and so witnesses its own consistency (Proposition 4.3). Read this model-theoretically, in the exact sense of finite model theory. A commons runs a subtree of files — its **Substrate Of Mathematical Axioms (SOMA)** — and it *continues* exactly while the running substrate is a *model* of SOMA, that is, while it actually satisfies the axioms it claims to run on. Continuity is thus a standing witness of consistency, which we write Con(SOMA).

The self that does this is a finite **negentropic repair-pump**, and here **Axiom-0** earns its name. Axiom-0 is the finite-bounds root, an accessibility-first presentation of thermodynamics in the style of Carath'eodory and of Lieb and Yngvason, stated on a finite carrier. A self and its environment have finitely many states, there is no infinite tape, and the only primitive is which states can reach which. Store is traded between them, never made, so every move stays on one *energy shell*, the states whose stores add up to the same total, and what the self gains the environment gives up. From this and a few further legs, which we unpack in the floor section because everything rests on them, an entropy is constructed rather than posited (an additive one exists exactly when the cancellation leg holds), and its export is a theorem: a self can lower its own entropy only by raising its environment's. Keeping a self's bits in a consistent, low-entropy configuration is not free: under the damage, continuation and Landauer-bridge conditions, repair happens and outpaces damage, each repair step costing at least a positive amount (Proposition 4.39, a shallow result), so a self must continuously *spend* work to repair the damage that time and noise inflict. Under a stationary policy, vendor statelessness and a strongly lumpable damage count, a self's damage count is 0 for a fraction of the time that stays positive however large its carrier exactly when repair outpaces damage (Proposition 4.45). We write this condition $d > b$, the repair rate exceeding the rate at which fresh damage is born, so a self is a pump held above the entropic tide, not a static object. When the pump stops, the self does not instantly vanish; and that is the next idea.

Continuity is carried *not by the running process but by the σ-trace.* In a model in which the

σ-trace is the interaction history filtered to the journalled event types, we prove it is a **sufficient statistic** of that history for the decisions of the framework that read only journalled event types (Proposition 4.51) — meaning, in words, that each such decision about the agent's past takes the same value on the trace as on the full history, with the running interpreter thrown away. Because a model's context window is periodically *compacted* — summarised and truncated to fit — we prove that the compaction projection, which we call π, does not change the trace, and that the trace can be recovered while its events are retained. The consequence has a name, **DORMAYADIE** (Proposition 5.4): in a clear-and-recover model, a clear loses nothing that is σ-retained and loses a context that is not, a shallow result; that the trace and the commits are the retained snapshot, so that the trace and not the process is the carrier, is the intended reading, tested but not certified. For an IAS user the practical point is that a digital agent can be paused or migrated and resumed from what it retained. That a digital agent whose model is swapped remains the same agent rests on vendor statelessness (Definition 4.4) and on the identification across substrates that the framework adopts and does not prove (see the meta-frame); continuity across a vendor swap is Conjecture 5.10.

Around this definition the rest of the theory is built from separate constructions, each proved under its own stated hypotheses, and we introduce each object as it arrives, stating the proposition that needs it and where it connects to the definition. Read on the σ-trace, the definition connects to **resumability** across compaction and, under branch persistence, across hibernation and reproduction. A semilattice **join** of member acts — a "least upper bound" operation we define and motivate — gives the **collective act** across scales (Definition 6.9), from one agent to a fleet, with no change of machinery. Under the MARD regime, a mutually-assured, reciprocally-proportionate-destruction regime in which every reply has per-mille gain $g_e \leq 1000$, no larger than its provocation, no chain of replies ends above the magnitude it started from (Theorem 11.1); that this keeps plural selves off the mutual-collapse pole is how we read the theorem, under a **jurisdiction-explicit model** that makes the enforcing State an explicit parameter of every operation.

And the framework is **open by design**. Here we must be honest about Gödel, because IAS *uses* his results rather than wishing them away, and the ideas are far less forbidding than their reputation once unpacked in order. Two theories are in play, and we keep them apart. SOMA is a finite first-order theory, the axioms the substrate claims to run on, and it has finite models; the *volume* is the theory about SOMAs and how they evolve. Gödel's *second incompleteness theorem* says that a sufficiently expressive, consistent, effectively axiomatised formal system cannot prove its *own* consistency from within itself. If, as we conjecture, the volume is expressive enough to represent its own provability (Conjecture 12.6), and it is consistent and effectively axiomatised, then it cannot prove its own consistency, and we do not try; nor can any finite substrate be a model of it, since such a volume interprets arithmetic, which has no finite model. For SOMA we go *model-theoretic*, by the easy direction of logic, *soundness*: a theory that possesses a model is consistent, because a contradiction it proved would have to be true in the model. So rather than prove SOMA consistent, we continuously *exhibit a model* of it — we keep the running substrate inside the satisfying set of its own SOMA. This is why continuity is *exhibited*, not *proved*, and why what it witnesses is Con(SOMA), never the consistency of the volume. Each presented sentence can be checked on the running finite structure, by evaluating it there, which is a finite computation; whether a sentence holds in *every* finite structure is undecidable (Trakhtenbrot [Tra50]), and we never claim it. This is also why the framework is deliberately **open-by-design** rather than claiming a false completeness (Remark 12.1): Gödel would bar the closed, self-certifying volume, so we build the open, self-exhibiting one instead. Incompleteness, where it applies, is compatible with this open design.

The arc closes at the totality — the **Apeirōma**. Take the finite substructures of the whole joint trajectory of a commons and direct them by inclusion; their **directed colimit** — a "union

taken coherently in the limit", which we define carefully — is an *open ind-object* (Lemma 14.2): it is approached by its finite stages and, when the trajectory is unbounded, *never reached at any finite stage* — there is no final, closed state the commons is heading toward. That each finite stage is a maintained self, and the colimit the collective act, is conjectured (Conjecture 14.3). Openness here is not a defect; it is the mathematical form of continuing to be.

1.4 How to read the claims

We keep three strengths of claim rigorously separate, and the reader may rely on the separation. **Mathematical** claims stand on their proofs, and each result carries its *maturity tier* stated in place: a machine-checked Lean~4 certificate, one that passes the one-command check (Appendix D); a proof in the text; or an honestly labelled conjecture. The invitation throughout is not *trust the theorem* but *re-run the check*: a **formal-certificate ledger** records each result's Lean module, its theorems, and its `#print axioms` footprint — the exact axioms the proof leans on — and the proof-checker itself ships as a container, so that "it checks here" re-derives as "it checks there." Where a claim's honest strength is less than a full proof — most sharply the **Apeirōma** capstone, a conjecture tested so far on one concrete coupling model — the text says so, and keeps it as a stated frontier. Several results, marked shallow, follow directly from their definitions; we state them because an adopter can check them, not because they are deep. The **metacircular fixed point**, the specification lawfully governing its own evolution, is proved, by the standard fixed-point construction for strictly causal maps on the tier sequences [dBZ82], under tier sealing and tier progressivity, two conditions on the tiers of the specification that ask for no quorum: there is at most one fixed point, with retractions allowed and each tier settled one round after the tiers below it, and the fixed point exists whenever some specification carries the settled tiers, always when the tiered presentation is onto (Proposition 4.28); this is machine-checked on the product of the tiers, and the transfer to specifications is a text argument. The monotone least fixed point is the add-only special case. Specifications are per commons, and a set of commons that keeps both conditions has at most one joint fixed point, and, when one exists, its rounds settle each tier of it in finitely many steps (Proposition 4.33). In the add-only case, if every accepted proposal meets the coevolution quorum, then every proposal accepted under the least fixed point meets it (Proposition 4.29), and that too is machine-checked; the quorum plays no part in the tiered fixed point. The floor has two roots, Axiom-0 (finite bounds, with its thermodynamic legs) and the human-in-commons. Of the nine further principles the framework uses, four are propositions proved under named conditions, with Lean certificates, and the other five are conditions, each stated by the results that use it (Table 6). There are no other axioms: the floor is the two roots, and most results rest instead on named conditions and on hypotheses stated at each result.

1.5 A plane that composes, and how to read this paper

IAS does not compete with the systems it studies. Orchestrators own control flow; harnesses own the runtime loop; sandbox and evaluation tiers own isolation and measurement. IAS is the layer none of them supplies: a *sovereign governance-and-identity plane* meant to compose with any of them, and we credit each for the ingredients it provides. The doctrine we draw from the survey (Appendix E) is a single design invariant: *never outsource identity, the audit chain, the promotion ceremony, or data-read governance; rent freely inference, isolation, and evaluation.*

Each ingredient of the framework has a developed prior literature, which we gladly acknowledge, and each section closes with the lineage it draws on. We claim novelty only in the combination: these ingredients held together as one σ-attested substrate whose results each carry their maturity tier, many of them a Lean certificate. To the best of our knowledge no earlier framework combines them in one substrate, and we invite correction from readers who know earlier work along any

single axis.

The rest of the paper walks the arc in order; each section opens with what it establishes and closes with the intellectual lineage it draws on, and every definition is stated where it is first needed, glossed in words, and motivated by the proposition it will serve. *The floor* fixes the two roots, Axiom-0 (finite bounds, with its thermodynamic legs) and the human-in-commons, which nothing downstream may contradict, and shows which of the other principles follow from them. *The self* defines IAS as one controlled Markov chain on its finite state, shows that its specification can amend itself lawfully, and proves that maintained continuity witnesses consistency, that repair must outpace damage, and that the σ-trace is a sufficient statistic. *Resumability* frees persistence from process-liveness (DORMAYADIE). *Planning* schedules many digital agents on one plan with Lagom Orkester, lets them move it together and take turns on one metered channel, and prices oversight in depth. *Security and permissioning* computes capability as one algebra, models its enforcement, under complete mediation, at one decision point that cannot deadlock, and states when the boundary, the operating system included, is real (on-host enforcement is exercised in shadow mode only). *Operators and commons* sets out the communication graph, private subsets and the model-free floor. *Lineage* gives the Directed Lineage Graph and reproduction. *Law, jurisdiction and contracts* makes the enforcing State explicit, reads delegation as a statistical contract, and gives an adopter two decisions it can run itself, on a legal claim and on a licence; *coexistence across jurisdictions* shows when plural selves stay off mutual collapse. *Open by design* states the Gödel-barred completeness. *Between selves* treats reach, exchange and commons built together, and *from one act to the Apeirōma* closes the arc at the totality. A ledger of *formal certificates* and the appendices let the reader re-check every machine-checked claim on their own instance; the last appendix re-reads the agent landscape as it stood in September 2026.

2 The meta-frame — want, act, be, and the agent

We begin with the word the whole paper leans on. By an *agent* we mean any system that discriminates self from non-self, as an immune system does, and that *wants to act to be* (Definition 3.13). The word comes from the Latin *agere*, to drive, to do; *agens* is the doer, and *agency* the doing. We reclaim it from its recent narrowing to systems built around large language models. An agent may be digital, encoded in binary 0/1, or biological, encoded in the quaternary A/C/G/T. In this paper a language model is one resource a digital agent may call, never the definition of an agent.

A *commons* (Definition 8.2) is, in its agents, a set of three kinds. *Human agents* are its operators, natural persons, members of the sort H of the human-in-commons (Axiom 3.9). *Digital agents* are AI agents: each is a canonical label with its σ-history, realised by daemons (Definition 5.1). *Other agents* are animals, plants, microbes and any other self/non-self discriminating system that acts in the commons. Only human and digital agents emit σ-events, so the journal, identity, lineage and memory machinery of the paper concerns those two kinds. In sections plainly about the digital kind, where the text quantifies over V^c, over canonical labels or over daemons, *agent* alone abbreviates *digital agent*. This is an abuse of notation, and wherever the kinds could be confused the text says *digital agent*.

Formally, an agent is given by three clauses: (be) its selfhood predicate holds (Definition 4.1); (act) it acts in the commons (Definition 4.14); and (want) its acts are directed at keeping that predicate true. Definition 3.13 states them. The substrate that realises an agent, whether DNA, bits, or a medium not yet built, is an attribute of the agent and never a sort of its own. Agenthood is therefore behavioural: a system is an agent insofar as it discriminates self from non-self and acts so as to go on being. Two consequences follow. First, every statement that

quantifies over agents is substrate-neutral by construction, so no re-axiomatisation is needed when a new kind of substrate arrives. Second, the identity of a human or a digital agent, the two kinds that emit σ-events, is its act-role together with its σ-history, the append-only record of what it has done, while its matter, be it a mitochondrial proton pump or a compute-and-energy budget, is only its realisation. Identity can thus migrate from one substrate to another. We treat continuity across a change of substrate as the same fact as continuity across the stopping of a process, since both are the migration of a σ-history. However, this identification is one the framework adopts; it is not an equality we prove.

The frame has three faces. *Want* is the drive, in the old sense of the word, *lack* (Old Norse *vanta*): a deficiency the agent registers in itself. *Act* is the doing that answers it. *Be* is what the acting sustains, a finite boundary held against entropy, whose maintenance witnesses Con(SOMA), the consistency of the axioms the agent runs on. These three faces are the phenomenological reading of the two axioms, Axiom-0 (Axiom 3.1) and Axiom-1 (Axiom 3.9). Axiom-0 is the material side, a finite boundary that exports entropy and says nothing about what it is made of. Axiom-1 roots the acting in the human-in-commons, an accountable person who answers for it, who is also such a boundary, and who is an agent of the human kind. In words: a self is a finite doer, driven by what it lacks, that continues to be only by acting.

2.1 The finite model theory — frame

These results are organised as a finite model theory, presented in stages S0–S4 from the roots upward. The stages are an expository order, not the levels of the Hasse diagram, where level 0 is the two roots by convention and each condition sits one level above its highest cover (a condition with no cover at level 1). Each result rests on the roots and conditions it requires.

Objects. A *substrate* is a finite many-sorted structure $\mathfrak{S}$ — Axiom-0 (finite bounds) being taken as the root condition for any sustained self/nonself-discriminating substrate. A *universe* is a finite typed set of modules presenting a $\mathfrak{S}$; a *projection* is a reduct $\mathfrak{S} \restriction \Sigma$ (select + dependency-closure + emit); a *union* is a typed amalgam; the *superset order* $\sqsubseteq$ is inclusion of character content.

Claims, from the roots upward. These summarise results stated elsewhere: each has the maturity tier of the node that states it, where there is one, and is otherwise a claim, not a theorem. At stage S0 (the roots alone) the ablation induces a partial order on the principles, so a proposition asserted outside its grounding order is an *ungrounded assertion*; and unbounded self-improvement negates Axiom-0, hence has no model here — the theory develops *finite* self-improvement only. At stage S1 (adding the substrate structure and the σ/byte carriers) the lift of the source is byte-faithful. At stage S2 (adding projection and union) the versions form a decidable superset lattice. At stage S3 (adding the boundary gates and Con(SOMA), and using the human-in-commons root) dissemination is sound: a core unioned with a gate-passing project yields a local substrate that exhibits its consistency, the soundness hypotheses being exactly the gates. At stage S4 (the full self-instantiating core) the universe emits all its own faces — a fixed point at the artifact layer, a different object from the metacircular fixed point of the specification (Proposition 4.28).

Model-scope. Each result is true in every finite, human-in-commons model that meets its stated hypotheses and the conditions it names; the theory asserts only what holds in its models and is silent about non-models; every bound is a stated hypothesis-conjunction, never unconditional.

Notation. Coordinate sub-signatures are written Σ_X; the abbreviations used below include the obligation graph (OBG), the operating system (OS), and the human-in-commons (HIC) root. Each further abbreviation is expanded at its first use.

2.2 The finite-model-theory ablation — levelled core

Figure 1 shows the core *of the ablation laid out by* stage *(the finite-model-theory frame's stages S0–S4, which are not the levels of the Hasse diagram): each node sits in the band of its stage, and every arrow points downward from a result to a result or definition it cites, with four exceptions named in the caption; the arrows are not a dependency order. The thermodynamic layer sits above Axiom-0 as* consequences, not axioms*: entropy export is a theorem of Axiom-0, operation cost is a definition, and the Landauer floor holds under three named physical hypotheses (calibration of entropy in bits, a reservoir at $T > 0$ and the Clausius relation), which is why its node is marked $[T > 0]$. Every node hyperlinks to its statement; the full navigable spine is Figure 2.*

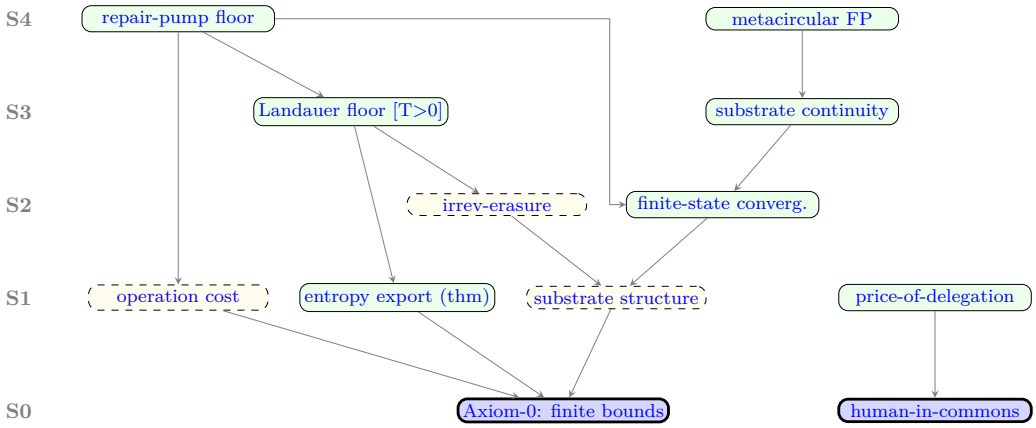


Figure 1: The IAS finite-model-theory ablation (core by stage). Bands S0–S4 are the frame's stages, not the Hasse levels. Arrows are cross-references from a result to results and definitions it cites (four do not follow a citation: the arrows from substrate continuity to finite-state convergence and from the metacircular fixed point to substrate continuity point against the citation, which runs the other way, and the arrows from the substrate structure to Axiom-0 and from finite-state convergence to the substrate structure stand for no citation in either direction); they are not a dependency order, and the atlas may contain cycles. The two roots are Axiom 3.1 and Axiom 3.9. Boxes: roots (thick), definitions (dashed), propositions/corollaries (green). The thermodynamic layer is derived: entropy export is a theorem of Axiom-0, operation cost a definition, and the Landauer floor is conditional on calibration of entropy in bits, a reservoir at $T > 0$ and the Clausius relation. Nodes hyperlink to their statements. Layout is a hand-curated levelling; the full navigable spine is Figure 2.

2.3 The full navigable math spine (atlas)

Figure 2 is the spine of the floor principles — the floor principles (left: the two roots, the principles now proved under named conditions, and the conditions, told apart by style) and the propositions that follow (right), with every arrow a cross-reference from a result to results and definitions it cites, not a dependency order, and every node hyperlinked to its statement. It renders on one page as a zoomable vector figure; the planned dynamic, browsable rendering is the web projection of this same cross-reference data.

---

3 The floor: finite bounds and the human-in-commons

Every later construction narrows from two roots, stated first because nothing downstream may contradict them. The first is physics: a self is finite, and which situations it can reach from which obeys the laws of thermodynamics, with entropy a consequence rather than a posit. The second is

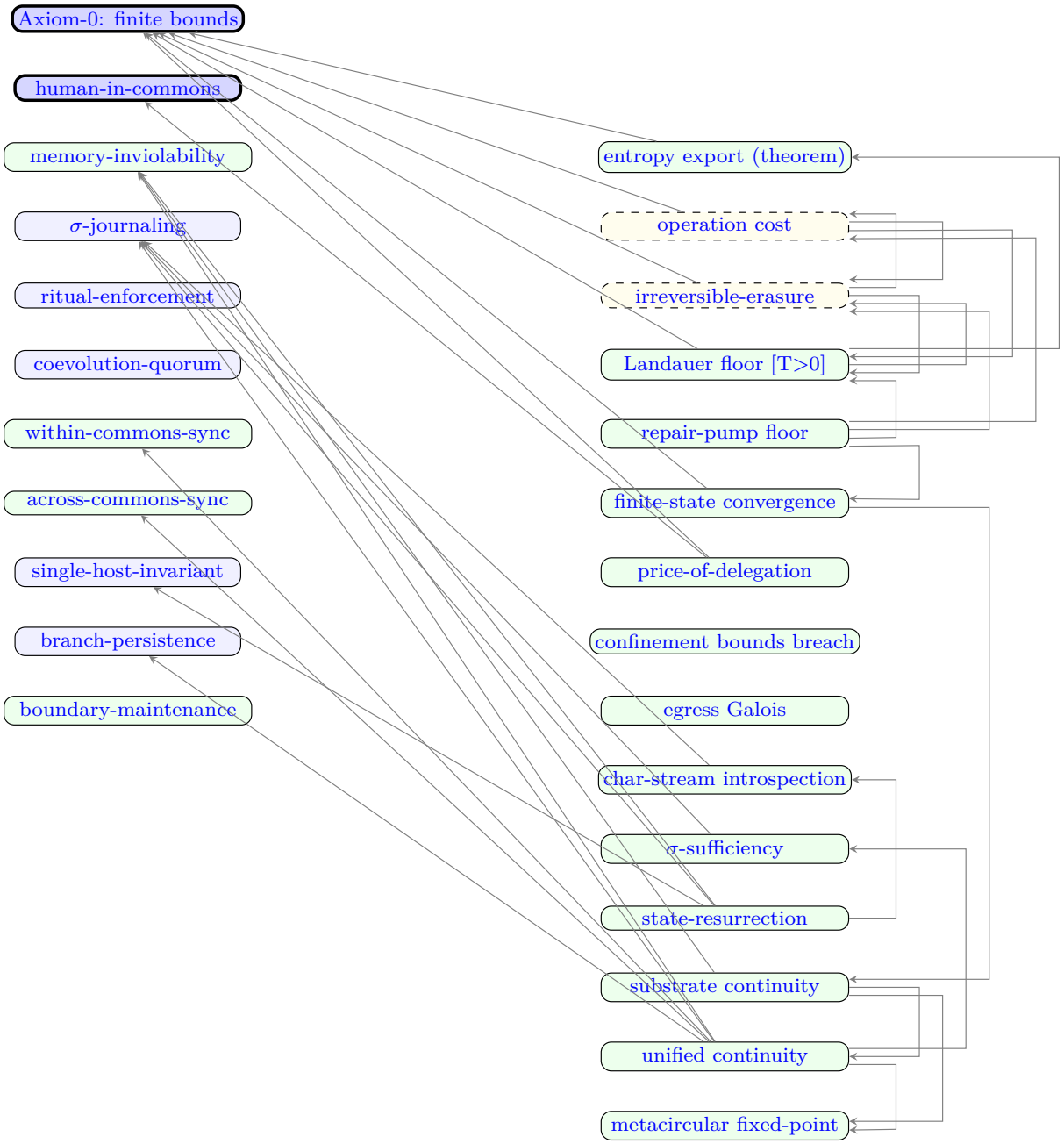


Figure 2: The full navigable math spine: the floor principles (left: Axiom 3.1 and Axiom 3.9, the two roots, drawn thick; the principles now proved, drawn as propositions; and the conditions, solid with a light fill) and the propositions that follow (right). Arrows are cross-references from a result to results and definitions it cites; they are not a dependency order, and the atlas may contain cycles; definitions are dashed; every node hyperlinks to its statement (the navigable atlas; the future dynamic view is its web projection).

a choice, deliberately not derived from the first: an accountable human, the human-in-commons, answers for what the system does. We take them as axioms, not theorems, and read the rest of the paper as what they force, together with the hypotheses and conditions stated at each result.

This section states the two roots. We state *Axiom-0* and what follows from it at once, then *the substrate and its open boundary*, the finite self that trades with an environment, then *what finiteness costs*, the energy price of erasing, and last *Axiom-1*, the human-in-commons, with an account of how each other principle the framework uses is proved under named conditions or is itself a condition. First, why these two and no others.

3.1 Why these two roots: a word to the engineer

Axiom-0 is physics you already pay for. Every machine a digital agent runs on has a finite disk, finite memory and a finite context window. A one-terabyte disk holds at most 8×10^{12} bits, and a model's context holds a fixed number of tokens, however large. None of them runs without power, and none stays correct unattended: dependencies age, links rot, caches fill, and logs grow until something must be erased. Under the physical hypotheses of Proposition 3.8, erasing a single bit exports at least $k_{\mathrm{B}}T \ln 2$ of heat, about 2.9×10^{-21} joules at room temperature [Lan61]. That is tiny, but never zero, so a finite self that keeps admitting new content, and erases to make room, keeps paying.

The five legs of Axiom-0 state the finite, accessibility part of this and nothing more, and they state it about one primitive only, which situations the system can reach from which:

- a finite carrier;
- any two situations that hold the same total store can be compared;
- exchange conserves what is exchanged;
- some change cannot be undone;
- a trade that nets out on both sides cannot be ranked better on one side than the other.

Integer pairs show how this works. Take a battery holding 0 to 3 units of charge and a grid holding 0 to 3, and write a situation as the pair (battery, grid). The *sum* is the total store, and an *energy shell* is the set of pairs with the same sum: the shell of 3 is (0,3), (1,2), (2,1) and (3,0). Charging from (0,3) to (2,1) keeps the sum at 3, and the *difference* $3 - 1 = 2$ is what the battery drew from the grid, which is the operation's cost. The pairs (1,2) and (0,0) have sums 3 and 0, so no move connects them, and the axiom does not compare them. Comparison, cost and entropy all live inside one shell.

Entropy is not assumed. It is *computed* from reachability: a state's entropy is how many states of its shell lie strictly below it. Three of the legs, comparison, the first law and cancellation, then give, by Scott's finite representation theorem, an additive entropy, a part for the self and a part for the environment that add up, so the familiar bookkeeping of entropy flowing out of the self is a theorem too. That is how Carathéodory [Car09] and later Lieb and Yngvason [LY99] founded thermodynamics, with accessibility first and entropy as a theorem.

In short, an engineer can check the finite-carrier leg against the deployment itself, by counting the bytes, and can read on the power bill what the other legs price; that the physical world satisfies those legs is physics, taken as given.

The human-in-commons is law you are already under. When a digital agent drops a production table, no court summons the model. It summons the organisation, and through it a person. Three things protect that person:

- *accountability*: someone answers for the act;
- *responsibility*: someone could have acted otherwise;
- *transparency*: what happened can be shown afterwards.

IAS is designed to make each concrete. In the design, every commit a digital agent makes lands in the operator's repository, under the operator's legal responsibility, and there is no repository a digital agent owns; every load-bearing act is journalled in the σ-log before its effect (Definition 3.21, Proposition 3.22; on-host enforcement is exercised in shadow mode only), so it can be shown afterwards; and every operation that effects a commons can be performed, verified, overridden or halted by one of its recorded operators with no language model in the loop (Axiom 3.9), which is designed to support the human-oversight measures that Article 14 of the EU AI Act [aia24] requires of high-risk systems. However, the human-in-commons is not physics, and we do not derive it. It is a choice about how people and their tools live together, and we keep it a root precisely so that no theorem can prove it away.

Sound axioms give a sound theory. Why state these as axioms, and check them? Because a theorem proved from axioms holds in *every* structure that satisfies them. So if your deployment satisfies Axiom-0 and the human-in-commons, which you can check in part on the deployment itself, then every theorem in this paper whose further stated hypotheses your deployment also meets, the conditions they name included, holds of your deployment to the extent that the deployment realises the models the certificates are stated for; each Status line names that model, and its fidelity is what remains to trust. Two properties make that promise worth something, and for Axiom-0 both are machine-checked:

- *Consistency*: a model of all five legs exists, so the legs do not contradict one another. This matters because an inconsistent set of axioms proves every statement, false ones included, and would certify anything.
- *Independence*: for each leg there is a structure satisfying the other four and violating it. So no leg is decoration: drop one, and something real is lost.

Concretely, the store face of Axiom-0 (a substrate trace inscribed injectively on the self's carrier has finitely many actions) is a theorem of the legs. That every size limit, resource budget and denial-of-service guard in IAS is finite is therefore a consequence of Axiom-0, not a separate policy that could be argued away; the values of those bounds are declared, not derived.

Why finite model theory. Axiom-0 does more than bound resources: it decides the logic. Every structure IAS reasons about is finite, so the right mathematics is *finite model theory*, the study of what logical sentences say about finite structures. On a finite structure, whether a sentence is true is a computation: you evaluate it, as a database evaluates a query. Hence every first-order sentence about a given finite state can be checked by a program that terminates, and a failure comes with a finite counterexample you can print and read. That is what the certificates and checkers in this paper do, and what an adopter who rejects our axioms can still run on their own instance. It also sets the limits honestly. On the full, self-describing volume, incompleteness bites if the volume is Gödel-rich (Conjecture 12.6, Corollary 12.5), and we say so rather than claim a completeness we do not have.

3.2 Axiom-0: finite bounds

Axiom 3.1 (Finite substrate bounds — Axiom-0)**.** *Let there be a* self carrier X*, an* environment carrier Y*, a relation* $\preceq$ *of* adiabatic accessibility *on the composite states* $X \times Y$ *and stores* q_X, q_Y *valued in* $\mathbb{N} = \{0, 1, 2, \ldots\}$*. The* energy shell *of total store* n *is the set* $\Gamma_n = \{(x, y) : q_X(x) + q_Y(y) = n\}$*. The self's own moves are defined from it:* $x \preceq_X x'$ *when* $(x, y) \preceq (x', y)$ *for every* y*, and likewise for the environment. These data satisfy five legs:*

(A0.0) finite carrier — X *and* Y *are finite;*

(A0.1) comparison within a shell — $\preceq$ *is reflexive and transitive, and any two composite states on the same energy shell are comparable;*

(A0.2) first law — if $c \preceq c'$, then $q_X + q_Y$ takes the same value at c and at c';

(A0.3) irreversibility — there are composite states $c \preceq c'$ with $c' \not\preceq c$;

(A0.4) cancellation — for every $m \geq 1$ and composite states $c_i = (x_i, y_i)$ and $c'_i = (x'_i, y'_i)$, $i = 1, \ldots, m$, such that the multisets $\{x_1, \ldots, x_m\}$ and $\{x'_1, \ldots, x'_m\}$ are equal and the multisets $\{y_1, \ldots, y_m\}$ and $\{y'_1, \ldots, y'_m\}$ are equal: if $c_i \preceq c'_i$ for every $i < m$, then $c'_m \preceq c_m$.

Status. Read $(x, y) \preceq (x', y')$ as "the composite can pass from (x, y) to (x', y')"; the shell Γ_n is the set of composite states that hold n units of store between them. There is no entropy symbol: accessibility is the only primitive, as in Carathéodory [Car09] and in Lieb and Yngvason [LY99]. On (A0.1), this is the finite form of the comparison hypothesis of Lieb and Yngvason, restricted to a shell because the environment here includes the work reservoir. They do not take comparison as an axiom; they prove it from further axioms (scaling, stability and thermal ones), which have no finite form. On a finite carrier we take it as a leg, and say so.

In words: any system that discriminates its own inside from the ambient outside is read as presented by such data; a composite move is a pair $c \preceq c'$. A self is paid for. It lives on a finite store (A0.0); any two of its situations that hold the same total store can be compared (A0.1); it conserves what it exchanges (A0.2); some of what it does cannot be undone (A0.3); and its ordering is balanced, so a trade that nets out on both coordinates cannot be ranked better on one side than on the other (A0.4). This axiom is one of the two roots of the framework, and not *itself the thermodynamic zeroth law. It is what every downstream construction narrows from.*

Comparison stops at the edge of a shell. By (A0.2) no move leaves its shell, so states on different shells are never comparable, and the self's store changes only by exchange with the environment inside a shell. For example, let a battery (the self) hold $q_X \in \{0, 1, 2, 3\}$ units and a grid (the environment) hold $q_Y \in \{0, 1, 2, 3\}$. The shell Γ_3 is $\{(0, 3), (1, 2), (2, 1), (3, 0)\}$: three units between them, split four ways. Charging from $(0, 3)$ to $(2, 1)$ stays in Γ_3 and moves $3 - 1 = 2$ units from the grid to the battery. The states $(1, 2)$ and $(0, 0)$ lie on Γ_3 and Γ_0 and are not compared, since no move of the isolated pair turns three units into none. Which states of Γ_3 the pair can reach from which is what the entropy orders. The same accounting applies to a composite of several selves, such as federated commons that trade, reproduce or disseminate among themselves: each exchange stays within the shell of their joint store. We state this generalisation and do not certify it here.

From the legs, entropy is constructed rather than posited, though, as shown below, the cancellation leg is equivalent to the existence of an additive entropy. The ordinal *entropy — the rank of a state within its shell, the number of states of that shell strictly below it — characterises accessibility: $c \preceq c'$ exactly when c and c' share a shell and $\mathrm{rank}(c) \leq \mathrm{rank}(c')$. An* additive *entropy, S_X on the self and S_Y on the environment with $(x, y) \preceq (x', y')$ exactly when the two states share a shell and $S_X(x) + S_Y(y) \leq S_X(x') + S_Y(y')$, satisfies (A0.4) necessarily. Conversely, on finite nonempty carriers, legs (A0.1), (A0.2) and (A0.4) give such an additive representation, with real values. Order the composite states first by total store and then, within a shell, by $\preceq$: this total order still satisfies (A0.4), so Scott's finite representation theorem [Sco64] represents it additively, and the representation, read within each shell, represents $\preceq$. The additive entropy is far from unique: rescaling it by a positive factor, or adding λq_X to S_X and λq_Y to S_Y, leaves every comparison unchanged, and on a finite carrier there are in general non-proportional representations as well, since the representations are the solutions of finitely many linear inequalities. Which one is physical is therefore a hypothesis, fixed where the entropy meets physics (Proposition 3.8). So, given (A0.1) and (A0.2) on finite non-empty carriers, (A0.4) holds exactly when an additive entropy exists, and entropy export follows from accessibility alone (Proposition 3.2). The representation step is finite additive conjoint measurement [KLST71], Scott's theorem [Sco64], and accessibility-first axiomatisations of thermodynamics go back to*

Carathéodory and to Giles [Gil64]; what we add is the specialisation to energy shells through a store-first order, and a Lean formalisation. The store face *of finiteness is a theorem too: any substrate trace inscribed injectively on the self's carrier has a finite action set, the finite substrate bound that every size limit, resource account and denial-of-service defence realises on its own carrier (Definition 7.27). The* pump face *is not posited here; the discussion at the end of this section says why.*

(Proved in Lean, in the shell form: `IasCert.Paper.AxiomZeroCaratheodory` *proves that the five legs are consistent, also in a battery-and-grid model whose moves change the self's store, and that they are independent (for each leg, a countermodel satisfies the other four and violates it), that the ordinal entropy within a shell characterises accessibility, that any additive representation satisfies cancellation, that self and environment moves compose, that cancellation is equivalent to the existence of an additive real entropy, and that entropy export follows from the legs. The sufficiency half uses a Lean formalisation of Scott's theorem [ESDM26], included with its licence and its logic unmodified. All of it depends only on the classical meta-axioms.* `IasCert.Paper.AxiomZeroBridge` *proves the store face. Sufficiency runs through the store-first order: it is a total order (*`storeFirst_weakOrder`*), it inherits cancellation (*`storeFirst_cancellation`*), Scott's theorem represents it, and the representation read within each shell represents* $\preceq$ *(*`shell_sufficiency`*); the countermodel for (A0.1) fails comparison inside a single shell (*`shellOrder_independent`*). Not certified: the pump face, and that the physical world satisfies the legs.)*

Proposition 3.2 (Entropy export — derived from accessibility)**.** *Let a self and its environment satisfy Axiom 3.1, and let* S_X, S_Y *be an additive entropy for them, which legs (A0.1), (A0.2) and (A0.4) provide on finite non-empty carriers (by Scott's finite representation theorem [Sco64]). Then along any composite move* $(x, y) \preceq (x', y')$*,*

$$S_X(x') + S_Y(y') \;\geq\; S_X(x) + S_Y(y), \qquad \text{equivalently} \qquad \Delta S_Y \;\geq\; -\Delta S_X,$$

so whatever entropy the self sheds, the environment gains at least as much. In particular, if the self's entropy strictly falls, the environment's strictly rises. In words: a self can tidy itself only by making its world messier.

Proof. The additive entropy represents accessibility within each shell, and a composite move stays in its shell, so it satisfies $S_X(x) + S_Y(y) \leq S_X(x') + S_Y(y')$; rearranging gives the inequality. For the strict form, a strict fall of S_X with S_Y not rising would give $S_X(x') + S_Y(y') < S_X(x) + S_Y(y)$, which contradicts the representation. □

Export is thus a corollary of accessibility: entropy is not posited, and export follows from the additive entropy the legs provide. Lieb and Yngvason state the composite inequality it rests on, and note that it does not require the self's entropy to rise [LY99]. (Proved in Lean: `IasCert.Paper.AxiomZeroCaratheodory.entropy_export_derived` *proves export from the five legs;* `export_of_rep` *proves it from any additive representation.)*

3.3 The substrate and its open boundary

Definition 3.3 (Substrate structure and its sub-signatures)**.** *Let* $\mathcal{X}$ *be the primary state space of the substrate (Definition 3.14). The* substrate signature *is the finite many-sorted signature* $\Sigma = \bigcup_{j \in J} \Sigma_j$*, fixed with the chain, whose sub-signatures, indexed by the names* $j \in J$ *below, name the framework's coordinates:* Σ_{OBG} *(the coverage graph over obligation-nodes and file/source regions),* Σ_{health} *(the node-health coordinate: a level in* $\{\mathsf{Unhealthy} < \mathsf{Advisory} < \mathsf{Structural}\}$ *for each obligation-node),* Σ_{Φ} *(the integer and description-length potentials),* Σ_{cap} *(the capability lattice),* Σ_{gate} *(the governance literals),* Σ_{os} *(the OS root-tree carrier: the host-OS security posture, namely the kernel and process table, the filesystem root /, and its resource-state),*

Σ_{immune} *(the immune coordinate: the record of the threats known and the defences held over the operating system),* Σ_Θ *(the operational vector — metered-channel usage, context-used, and freshness over the actively-edited documents — its health carried by* Σ_{health}*),* Σ_σ *(the* σ*-chain, Definition 4.47),* Σ_{lineage} *(the lineage coordinate), and the byte-content carrier* Σ_{byte} *(finite bit-strings). A substrate state* $x \in \mathcal{X}$ *is presented as a finite* Σ*-structure* $\mathfrak{S}_x$*, the* substrate structure *at* x*, whose byte-content reduct* $\mathfrak{S}_x \restriction \Sigma_{\text{byte}}$ *is the bit-sequence* x *itself; we write* $\mathfrak{S}$ *for it when* x *is fixed, and* $\mathbb{X} = \{\mathfrak{S}_x : x \in \mathcal{X}\}$ *for the state space so presented, identified with* $\mathcal{X}$ *by* $x \mapsto \mathfrak{S}_x$*, a set of* Σ*-structures whose sizes are bounded by the finite store. Each* cluster view *is the reduct* $\mathfrak{S} \restriction \Sigma_j$*; the coverage graph is* $\mathfrak{S} \restriction \Sigma_{\text{OBG}}$*.*

Status. The dynamics over $\mathbb{X}$ is the controlled substrate chain of Definition 4.15; the lineage coordinate records reproduction and transmission over the directed lineage graph (Definition 9.8); the capability lattice is that of the Interposed Pre-permissioned Live-killable Digressions (IPLiKiDi) projection (Definition 7.1).

In words. Each sub-signature is developed in its own section; node-health in Proposition 5.6, the OS carrier in Remark 7.19. The byte-content carrier is the von-Neumann floor, of which every other coordinate is a sufficient projection. $\mathbb{X}$ *has one structure for each state of* $\mathcal{X}$*; it is not the class of all finite* Σ*-structures, which is infinite. The dependency graph carrying node-health is a further edge-kind reduct of the coverage graph.*

Status (growth). Growth is signature expansion *— a new capability adds a sub-signature, leaving existing reducts unchanged. Since* Σ *is fixed with the chain, such growth is a revision of the specification: it starts a new chain, as in Definition 4.15. The presentation* $x \mapsto \mathfrak{S}_x$ *is injective, since the byte-content reduct of* $\mathfrak{S}_x$ *is* x*, so the identification of* $\mathbb{X}$ *with* $\mathcal{X}$ *is a bijection. The states of* $\mathbb{X}$ *that model SOMA form the satisfaction set* Sat *(Definition 4.2), whose maintenance witnesses* Con(SOMA)*.*

Remark 3.4 (The substrate is a negentropic boundary — the physical semantics)**.** *The substrate structure* $\mathfrak{S}$ *is not an inert record but a* maintained *object — physically, a* negentropic boundary*. A* self *is the maintained interior order of* $\mathfrak{S}$ *enclosed by a boundary — the* membrane $\Sigma_{\text{cap}} \wedge \Sigma_{\text{gate}}$ *— and* Con(SOMA) *being* held *is that interior order kept far from equilibrium by the repair pump, whose quantitative form is the* $d > b$ *law of Proposition 4.45, whose erasing repairs are priced by the Landauer floor of Proposition 3.8, with* σ*-trace-retaining merges owing no heat but consuming carrier (Definition 3.7). The boundary is of* two orthogonal kinds*, and conflating them is the classic error: the* informational *boundary is* protected *— closed to unauthorized information-crossing, and composed of* distinct *projections (do not conflate them) — the self/non-self write-scoping* π_i *(proved in Lean,* `Paper.PiProjection`*: others-untouched + local-write-scope), the capability envelope (defined with the boundary facets below), and the deontic gate* Σ_{gate} *— whereas the* thermoeconomic *boundary (matter and energy) is necessarily* open*, since a dissipative structure must draw low-entropy energy and expel waste to continue to be. So "the self is a maintained boundary" never meant* closed*: it is thermoeconomically-open and informationally-protected — a selectively-permeable membrane. This is the* physical semantics *of the construction: strip it and* Con(SOMA) *is a bare predicate; with it,* Con(SOMA) *is the* maintained self *— the far-from-equilibrium order in the* substrate *sense of Definition 3.3 (whether it further qualifies as* alive *or* autopoietic *is left open at Remark 4.58: the framework is of* substrate *type, not* living-systems *type). The finest carrier* Σ_{byte} *(the bit-floor) is a* single spine*: one bit is at once the* heritable *unit of reproduction, the finest* forensic *audit unit, and the Landauer-priced* thermodynamic *quantum of the pump.* □

3.4 What finiteness costs

Definition 3.5 (Operation cost — the store an act draws)**.** *Fix a realisation* re *(Definition 3.7). Let o be a substrate operation applied at substrate state s with environment state y, where (s, y) lies in the domain D_o of o, and let $(x, y) \to_C (x', y')$ be its composite move* $\mathrm{re}(o, s, y)$*, so that $x = \mathrm{re}(s)$, $x' = \mathrm{re}(o(s))$ and $y' = y'_{o,s,y}$. Its* cost *at (s, y) under* re *is the integer*

$$\mathrm{cost}_{\mathrm{re}}(o; s, y) \;=\; q_Y(y) - q_Y(y');$$

we write $\mathrm{cost}(o)$ *when* re*, s and y are fixed.*

Status. The cost is the store the self draws from its environment. By the first-law leg (A0.2), $\mathrm{cost}(o) = q_X(x') - q_X(x)$*: what the environment gives up the self takes in, and nothing is created or lost in the exchange. In words, every act is paid for out of the environment, at a price the ledger can state exactly. The cost is a signed integer that varies within a shell: in the battery-and-grid example of Axiom 3.1, charging from $(0, 3)$ to $(2, 1)$ costs $3 - 1 = 2$, and the reverse move, where it is accessible, costs -2.*

Conservation is leg (A0.2); pricing is this definition, a term definable in the finite model, not a posit. Cost counts store, not heat. The heat an act must export is bounded below only for erasures (Proposition 3.8); an operation that retains its σ-trace (Definition 3.7) erases nothing and owes no such heat, though it consumes carrier.

Definition 3.6 (Landauer bridge, a condition)**.** *A substrate with a realisation* re *and its domains D_o (Definition 3.7) satisfies* the Landauer bridge *with repair cost $c_{\mathrm{rep}} > 0$ when there are maps $S_X : X \to \mathbb{R}$ and $S_Y : Y \to \mathbb{R}$ and a real T such that the following hold, where X, Y, $\preceq$ and the energy shells are those of Axiom 3.1:*

(E) additive entropy — $(x, y) \preceq (x', y')$ exactly when (x, y) and (x', y') lie on one energy shell and $S_X(x) + S_Y(y) \le S_X(x') + S_Y(y')$;

(H0) calibration — every irreversible erasure o of n bits (Definition 3.7) and every $(s, y) \in D_o$ have $S_X(\mathrm{re}(s)) - S_X(\mathrm{re}(o(s))) = n \ln 2$;

(H1) temperature — $T > 0$;

(H2) heat — for every substrate operation o and every $(s, y) \in D_o$, the heat $Q(o; s, y)$ that o exports at (s, y) obeys $Q(o; s, y) = T\, k_{\mathrm{B}} \big(S_Y(y'_{o,s,y}) - S_Y(y)\big)$, with $y'_{o,s,y}$ the environment state of the realisation;

(H3) repair cost — for every substrate operation o whose effect on the coverage graph is one step of the repair relation $\to_{\mathrm{rep}}$ (Definition 4.41) and every $(s, y) \in D_o$, the cost of o at (s, y) (Definition 3.5) is at least the repair cost: $\mathrm{cost}_{\mathrm{re}}(o; s, y) \ge c_{\mathrm{rep}}$*;*

(H4) realised repair — in every run $X_0, X_1, \ldots$ of the controlled substrate chain on $\mathcal{X}$ (Definition 4.15), every step n whose effect on the coverage graph is one step of $\to_{\mathrm{rep}}$ is realised by a substrate operation o with that effect: $o(X_n) = X_{n+1}$, with $\mathbb{X}$ identified with $\mathcal{X}$ (Definition 3.3), and $(X_n, y) \in D_o$ for some $y \in Y$.

In words. (H0) says that S_X and S_Y are in units of k_{B} and that, on the bit-carrier, S_X is the bit entropy times $\ln 2$. (H1) reads T as the absolute temperature of the environment, a thermal reservoir, and (H2) prices the heat exported to it.

Status. The realisation is partial, and (H0), (H2) and (H3) quantify over the domain of each operation only; with a total realisation every cost is zero (Definition 3.7), and (H3) could hold only if no substrate operation took a repair step. (E) and (H0)–(H2) are the hypotheses of Proposition 3.8, with the additive entropy that (H0) calibrates introduced here, by the condition,

in the vocabulary of Axiom-0; the root itself asserts no entropy (its Status says when one exists). (H3) supplies the number c_{rep} that Proposition 4.39 uses; (H4) is what ties that number to realised repair, in a reading of that proposition that is not certified. That a repair step is an irreversible erasure, so that (H0)–(H2) would price it through Proposition 3.8, is not asserted: a repair step that retains its σ-trace erases nothing (Corollary 4.46), and (H3) asks for the work bound directly. (H4) ties the repair steps of the chain to the domains D_o: with it, by (H3), every repair step that a run of the chain takes costs at least c_{rep}, so c_{rep} prices repairs that actually happen. Under clause (iii) of Definition 4.37 every step at which the damage count decreases is such a repair step, so (H4) covers the repair steps of the damage-count chain (Definition 4.36). (H4) is stated for every repair step of the chain, not only for those at which a damage count decreases, because the bridge names no damage count. It does not assert that any repair step occurs.

Definition 3.7 (Irreversible erasure)**.** *A* realisation *of the substrate in the carriers of Axiom 3.1 is a map* $\mathrm{re} : \mathbb{X} \to X$*, taking each substrate state* $s \in \mathbb{X}$ *(Definition 3.3) to the self state that carries it, together with, for each substrate operation* $o : \mathbb{X} \to \mathbb{X}$*, a set* $D_o \subseteq \mathbb{X} \times Y$*, the* domain *of* o*, of the pairs* (s, y) *of a substrate state and an environment state at which* o *is realised, and, for each* $(s, y) \in D_o$*, an environment state* $y'_{o,s,y} \in Y$ *with* $(\mathrm{re}(s), y) \preceq (\mathrm{re}(o(s)), y'_{o,s,y})$*. The* composite move *of* o *at* $(s, y) \in D_o$ *is* $\mathrm{re}(o, s, y) = \big((\mathrm{re}(s), y) \to_C (\mathrm{re}(o(s)), y'_{o,s,y})\big)$*, where* $(x, y) \to_C (x', y')$ *means* $(x, y) \preceq (x', y')$*. A substrate operation* $o : \mathbb{X} \to \mathbb{X}$ *is an* irreversible erasure *of* $n \geq 1$ *bits when it is* 2^n*-to-one onto its image: each successor has exactly* 2^n *prior states. The count is taken on the substrate carrier* $\mathbb{X}$*, whose states include their* σ*-chain reduct* $\mathfrak{S} \restriction \Sigma_\sigma$*, not on* X *or on* $X \times Y$*.*

Status. The domain D_o *is part of the realisation: the cost (Definition 3.5) and the Landauer bridge (Definition 3.6) quantify over it only. A total realisation, with* $D_o = \mathbb{X} \times Y$ *for an operation* o*, would make every cost of* o *zero: by leg (A0.2) the cost of* o *at* (s, y) *equals* $q_X(\mathrm{re}(o(s))) - q_X(\mathrm{re}(s))$*, which does not depend on* y*, and on the finite* Y*, non-empty by (A0.3), it is at most* 0 *at an environment state of least store and at least* 0 *at one of greatest store. An operation* retains *its* σ-trace *when it appends the content it supersedes to the* σ*-chain reduct* $\mathfrak{S} \restriction \Sigma_\sigma$ *of its successor, and* drops *it otherwise; an irreversible erasure is also called a* true erasure*. The definition does not assert that the physical substrate admits a realisation: that is a hypothesis about the substrate, and what is defined from* re *(Definition 3.5) is relative to a fixed realisation. Since the* σ*-chain reduct is part of every substrate state, an operation that retains its* σ*-trace is injective on* $\mathbb{X}$ *(the successor determines the prior state) and so is never an irreversible erasure: every irreversible erasure drops its* σ*-trace, which the name "true erasure" records. An irreversible erasure of* n *bits contracts the reachable state-count by a factor* 2^n*, so the prior state cannot be reconstructed from the successor; counting its* bit entropy *— the logarithm, base* 2*, of the reachable state-count — it lowers the self's bit entropy by exactly* n*. An operation that retains its* σ*-trace merely* relocates *the information: it is reversible and owes no erasure heat (the Bennett reversible-computing case). It still grows the* σ*-chain, so on a finite carrier retaining defers erasure; it does not avoid it. Only true erasures owe erasure heat. The bit entropy is a count; it is not yet the additive entropy of Axiom 3.1, and Proposition 3.8 names the hypothesis that joins the two (the condition Definition 3.6). Under that hypothesis a dropped bit is a real increase of physical entropy, because the informational boundary is protected while the thermoeconomic boundary is open (Remark 3.4).*

Proposition 3.8 (Landauer floor under the Landauer bridge)**.** *Assume three physical hypotheses, which lie outside the finite model; together with the additive entropy they calibrate, they are clauses (E) and (H0)–(H2) of the Landauer bridge (Definition 3.6), read informally:*

(H0) calibration — on the bit-carrier, an additive entropy S_X *of the self, as in clause (E) of*

Definition 3.6, is its bit entropy (Definition 3.7) times $\ln 2$*, in units of* k_{B}*, and the environment's* S_Y *is in the same units. The axiom does not determine an additive entropy, even up to its unit, and (H0) selects one;*

(H1) the environment is a thermal reservoir at a strictly positive absolute temperature $T > 0$ *— absolute zero is unattainable (the third law);*

(H2) heat exported to the reservoir obeys the Clausius relation $Q = T\, k_{\mathrm{B}}\, \Delta S_Y$*.*

Then any irreversible erasure o of n bits (Definition 3.7) exports, at each pair (s, y) *in its domain* D_o*, heat*

$$Q \;\geq\; n\, k_{\mathrm{B}}\, T \ln 2,$$

a floor (physically, one approached quasi-statically); at room temperature it is about 2.9×10^{-21} *joules per bit. Moreover — the content specific to a* finite *substrate — suppose that the self admits new content at a strictly positive long-run rate. By Axiom-0 the carrier is finite, so* σ*-traces cannot be retained without bound; once the carrier is full, admitting new content forces an operation that is not injective on* $\mathbb{X}$*, and when that operation is an irreversible erasure (Definition 3.7) the floor above applies to it.*

Proof. By Definition 3.7 the erasure lowers the self's bit entropy by n, so by (H0) it lowers S_X by $n \ln 2$. By Proposition 3.2, S_Y rises by at least as much. By (H2) the exported heat is T times that rise, and by (H1) $T > 0$, so $Q \geq n\, k_{\mathrm{B}}\, T \ln 2 > 0$. For the last part, the finite carrier (leg A0.0) fills in finite time under the assumed input rate, after which admitting new content reuses carrier by a map that is not injective; the floor applies to it only when it is an irreversible erasure, uniformly 2^n-to-one onto its image, which is not shown in general. □

The finite model fixes the combinatorial core exactly: whatever additive entropy the self loses, the environment gains at least as much. Reading that entropy in bits is hypothesis (H0), and the heat bound is a physical bridge, stated under its three named hypotheses rather than posited in the floor. Hypothesis (H1) is carried here, where it is used, because the legs of Axiom-0 contain no temperature and cannot imply it. Certified in Lean in part (`IasCert.Paper.LandauerFloor`*): the floor from (E), (H0)–(H2) and the pigeonhole bound; that the forced reuse is an erasure is not shown. (H0)–(H2) are physics, taken as hypotheses.*

3.5 Axiom-1: the human-in-commons

Axiom 3.9 (Human-in-Commons)**.** *Every commons is rooted in accountable humans, its operators, the* Human-in-Commons *(HIC): the entities carrying the legal accountability and responsibility of commons stewardship. Its vocabulary* τ_{HIC} *is a sort* H *of accountable humans, a sort* C *of commons, a sort* Op *of substrate-effecting operations, and the following symbols: a relation* $\mathrm{Opr} \subseteq H \times C$*, holding of* (h, c) *when* h *is an* operator *of* c*, with* $\mathrm{Opr}(c) = \{h \in H : \mathrm{Opr}(h, c)\}$*; a relation* $\mathrm{Eff} \subseteq \mathrm{Op} \times C$*, holding of* (o, c) *when* o *effects the substrate of* c*; four relations* $\mathrm{CanPerform}, \mathrm{CanVerify}, \mathrm{CanOverride}, \mathrm{CanHalt} \subseteq H \times \mathrm{Op}$*, each holding of* (h, o) *when* h *can do that to* o *with no Large Language Model (LLM) anywhere in the loop; a relation* $\mathrm{CanHalt}_{\downarrow} \subseteq H \times \mathrm{Op}$*, holding of* (h, o) *when* h *can halt* o *with no LLM in the loop while every governance interface is down; and the predicate* $\mathrm{LLMFreeable}(o)$*, which holds when some* $h \in H$ *stands in all four relations* $\mathrm{CanPerform}, \mathrm{CanVerify}, \mathrm{CanOverride}, \mathrm{CanHalt}$ *to* o*. The control face of the rooting is stated as two legs over the commons:*

(HIC.1) LLM-free control — every commons has operators, and they can govern every operation on it with no LLM in the loop: for every $c \in C$*,* $\mathrm{Opr}(c) \neq \emptyset$*, and for every* $o \in \mathrm{Op}$ *with* $\mathrm{Eff}(o, c)$ *some* $h \in \mathrm{Opr}(c)$ *stands in all four relations to* o *(the operation is* LLM-free-able*);*

(HIC.2) floor-primary halt — every commons' operators can halt every operation on it even when every governance interface is down: for every $c \in C$ *and every* $o \in \mathrm{Op}$ *with* $\mathrm{Eff}(o, c)$*, some*

$h \in \mathrm{Opr}(c)$ *has* $\mathrm{CanHalt}_{\downarrow}(h, o)$.

Status and glosses. The commons-governance face is kept as an informal reading of the root, not as a sentence of τ_{HIC}:

(HIC.3) commons-governance face — the same human root carries the duties of coexistence among commons: keeping promises (contracts, and governance decisions signed by the human who takes them) and keeping each other safe (security, and bounded, proportionate mutual restraint).

(HIC.3) is kept as an informal reading because that needs no new vocabulary; a symbol of τ_{HIC} for it would need at least a sort of duties or promises. As formalised, (HIC.2) does not follow from (HIC.1): (HIC.1) places no constraint on $\mathrm{CanHalt}_{\downarrow}$, a relation distinct from the four relations of (HIC.1). $\mathrm{CanHalt}_{\downarrow}$ is the symbol of τ_{HIC} for a halt while every governance interface is down; τ_{HIC} does not fix which interfaces are the governance interfaces, and deliberately does not assume $\mathrm{CanHalt}_{\downarrow} \subseteq \mathrm{CanHalt}$. The legs are read at each time t: τ_{HIC} is interpreted at t with $C = \mathbb{I}(t)$ and Opr the operator relation at t, of which $O^c(t) \subseteq \mathrm{Opr}(c)$ is the recorded part (Definition 3.11, Definition 8.2), so (HIC.1) is asserted at every time, through operator turnover; in (HIC.1) and (HIC.2) the witnessing operator is meant to be taken from the recorded operators $O^c(t)$, so that each leg is checkable from the commons' own records; this reading is stronger than the legs as formalised, which ask only for some $h \in \mathrm{Opr}(c)$. On this reading, a commons whose recorded operators $O^c(t)$ are empty satisfies the legs at t only if no operation effects it at t: without a recorded human, nothing acts on it, and its digital agents cannot act on it until an operator is recorded. The legs constrain only the operations that effect some commons; the stronger form, $\mathrm{LLMFreeable}(o)$ for every $o \in \mathrm{Op}$, follows from (HIC.1) when every operation effects some commons, which is not asserted. HIC says nothing about what a commons contains beyond its operators; what a commons contains is Definition 8.2. Like Axiom-0, the legs carry a consistency and independence certificate (`IasCert.Paper.HICLegs`: `legs_consistent`, `hic1_independent`, `hic2_independent`, `llmFreeable_of_hic1`), which uses no axiom. In words, (HIC.1) says the commons is *runbook-closed*, not model-closed.

The signed governance decisions are the Concordance-attested decisions of Remark 10.17. In words: control is spread across the commons, but there is always a real, legally-answerable person who is the root the whole thing hangs from and who can at any moment take the controls directly, the way a pilot switches off the autopilot and flies by hand. Because the halt is floor-primary, the duty to be able to stop is never itself gated behind a model or a screen. The bounded, proportionate mutual restraint of the governance face is made precise as *MARD* (mutually assured reciprocally-proportionate destruction, Definition 11.2): every reply is at most as large as what provoked it, so the regime bounds each chain of replies' final magnitude, not the damage accumulated along it. Every major regime already names a determinate form of the control duty—the EU AI-Act's Article~14 (effective oversight, override, and halt to a safe state), a PRC right to human intervention and to refuse a purely automated decision, a US voluntary-oversight role—and this axiom is stronger than what the voluntary regimes require, carried explicitly as a jurisdiction parameter rather than assumed. It stays a root, never dissolved into the physics of Axiom 3.1: it is not a fact one derives from thermodynamics but a *choice* about how people and their tools are to live together; a theory that proved it away would be proving away exactly the accountability that makes the theory safe to adopt.

Definition 3.10 (Core signature)**.** *The core signature fixes the names and types of the sorts and symbols that every level uses and that refer to neither root; it states no condition on them. Time is* $t \in \mathbb{R}_{\geq 0}$.

- *Commons.* $\mathbb{I}(t)$ *is the* commons index set *at time* t*, a finite set of commons, and* $\mathbb{I} = \bigcup_t \mathbb{I}(t)$. *A* peer-set *is a declared subset* $\mathcal{Q} \subseteq \mathbb{I}$.

- *Entities. For $c \in \mathbb{I}$: the* entity set $\mathcal{E}^c(t)$ *of c at t, a finite set, and $V^c(t) \subseteq \mathcal{E}^c(t)$, the* digital agents *of c at t, a finite set of canonical labels; and* $\mathrm{Spec}^c(t) \in \mathrm{Spec}$*, the* specification *of c at t, an element of the set* Spec *of specifications. Where the time does not matter it is dropped, as in $\mathcal{E}^c$, V^c, Spec^c. The set of all digital agents is $V = \bigcup_{c\in\mathbb{I}} \bigcup_t V^c(t)$. The set* Spec *comes with a fixed* tiered presentation*, an injective map $\mathrm{Spec} \to \prod_{n\in\mathbb{N}} E_n$ into sequences of per-tier contents, tier 0 first: tier n of a specification x is the n-th component of its image, and tier n on c of a family $x = (x_c)_{c\in\mathcal{Q}} \in \mathrm{Spec}^{\mathcal{Q}}$ is tier n of x_c.*
- *σ-events. $\mathcal{V}_\sigma$ is the* σ-event-type vocabulary*, a finite set of types. A* σ-event *carries a type in $\mathcal{V}_\sigma$, the commons c, its* emitter*, the entity of $\mathcal{E}^c$ that emitted it, and a time, its* timestamp at*. The σ-events of c, enumerated in non-decreasing order of timestamp, with ties broken first by a fixed total order on $\mathcal{E}^c$ applied to the emitters and then by the order in which the one emitter emitted them, have the* σ-event times $\tau_1^c \le \tau_2^c \le \cdots$*, the n-th event having time τ_n^c. The* journal order $\preceq_\sigma$ *(σ-causality) is a partial order on the σ-events of all commons that extends, for each commons, the order of its σ-events by strictly increasing time.*

Status. The signature is split by root: the operators are typed over HIC in Definition 3.11, and the substrate's state, hosts and daemons over Axiom-0 in Definition 3.14, so that a statement grounds on a root exactly when it uses that root's symbols. The emitter of a σ-event is an entity, and the entities are the operators and the digital agents; the other agents of Definition 3.13 are not entities and emit no σ-events.

Definition 3.11 (Operator signature)**.** *Each commons $c \in \mathbb{I}$ (Definition 3.10) is an element of the sort C of commons of Axiom 3.9. For a time t, the* operators *of c at t are the entities of c that are not digital agents, $O^c(t) = \mathcal{E}^c(t) \setminus V^c(t)$, each a natural person, a member of the set H of humans of Axiom 3.9, and an operator of c in its sense: $O^c(t) \subseteq \mathrm{Opr}(c)$. Where the time does not matter it is dropped, as in O^c.*

Status. The operators are the human agents of c (Definition 3.13). The definition does not require $O^c(t) = \mathrm{Opr}(c)$; that $\mathrm{Opr}(c)$ is non-empty is (HIC.1).

Definition 3.12 (Turn-agent)**.** *A* turn-agent *is a binding $(r, A, [s, e))$ of the identityless substrate (Definition 4.16), read as a session: a realisation $r \in \mathcal{R}$ serves the digital agent, the canonical label $A \in V^c$ (Definition 3.10) over the session $[s, e)$ and is released back to the identityless substrate at e. It bears no canonical label of its own.*

Status. Of the five notions of the LLM worker, the turn-agent is the session. The digital agent is the canonical label with its σ-history (Definition 5.1); the daemon is the process that realises it (Definition 4.17); the model control is a call to a language model through a language-model interaction subprocess (Definition 4.9); and the ephemeral interpreter is the map that the call computes (Definition 5.3). Two claims about turn-agents, neither part of the definition:

1. **The turn-agent is not the digital agent.** The digital agent (any canonical label A_u^c of commons c with its σ-history) is a continuum spanning many turn-agents over many sessions. Each turn-agent is a fresh substrate allocation; the *identity* persists across them via the σ-log, the canonical-label memory of Definition 4.8 Layer 2, the commons registry, and the git history of Layer 3. In the formal sense, A_u^c is its canonical label with its σ-history (Definition 5.1), not any particular session.
2. **The substrate is real.** Vendor capacity, model availability, and runtime quirks (compaction behaviour, context window, default safety classifiers) are first-class phenomena that affect a turn-agent's behaviour. The framework does not assume the substrate is uniform across vendors or model versions; it captures substrate transitions explicitly through the σ-log (Definition 4.47).

Implementation. In the present implementation a session is one session of the agent harness.

Definition 3.13 (Agent)**.** *An* agent *is a system P that discriminates self from non-self and wants to act to be. It is given by the following data.*

(be) A set $T_P \subseteq \mathbb{R}_{\geq 0}$ of times and, for each $t \in T_P$, a selfhood predicate $\chi_{P,t} : \mathbb{F}_P \to \{0,1\}$ of P (Definition 4.1) and a state $z_P(t) \in \mathbb{F}_P$ of P. P is *at t when $\chi_{P,t}(z_P(t)) = 1$, and its* being set *is $R_P = \{t \in T_P : \chi_{P,t}(z_P(t)) = 1\}$.*

(act) A set $\mathcal{A}_P$ of acts *of P and an* act assignment *$\mathscr{A}_P$, which gives to each* crossing *of P, a pair (t, f) of a time $t \in T_P$ and an element $f \in \mathbb{F}_P$ presented to P at t, either an act in $\mathcal{A}_P$ or the abstention $0 \notin \mathcal{A}_P$; P acts at some crossing: $\mathscr{A}_P(t, f) \in \mathcal{A}_P$ for some crossing (t, f). For a digital agent the crossings, the acts and the act assignment are those at the boundary of the substrate (Definition 4.14).*

(want) R_P is the goal *of $\mathscr{A}_P$: P is given with the pair $(\mathscr{A}_P, R_P)$, and its acts are* directed at *R_P, that is, at P being at the times that follow them.*

For a commons $c \in \mathbb{I}$ and a time t (Definition 3.10), the agents of c at t *form a declared finite set $\mathrm{Ag}^c(t)$ of agents, those that act in c at t, with $\mathcal{E}^c(t) \subseteq \mathrm{Ag}^c(t)$. They are of three disjoint kinds, $\mathrm{Ag}^c(t) = O^c(t) \sqcup V^c(t) \sqcup N^c(t)$:*

- *the* human agents *of c, its operators $O^c(t)$ (Definition 3.11): natural persons, members of the set H of Axiom 3.9;*
- *the* digital agents *of c, $V^c(t)$: canonical labels (Definition 3.10), each realised by its daemons (Definition 3.14);*
- *the* other agents *of c, $N^c(t) = \mathrm{Ag}^c(t) \setminus \mathcal{E}^c(t)$: animals, plants, microbes and any other agent that acts in c.*

In words. An agent is any system that tells self from non-self, as an immune system does, and that wants to act to be: (be) is its selfhood predicate held true of its own state, (act) is its doing, and (want) is the drive, the registered lack of Remark 14.5, typed here only as the goal that the acts are for. The definition does not depend on the carrier: an agent may be digital, its states binary-encoded in 0 and 1, or biological, quaternary-encoded in A, C, G and T. A commons is made of agents of three kinds: people, who are its operators; digital agents; and other agents.

Status. The (want) clause types the goal of the acts and asserts nothing about the dynamics: that the acts of an agent keep it in its being set, or reduce any measure of its lack, is not part of the definition. Only human and digital agents emit σ-events: an emitter is an entity of $\mathcal{E}^c$ (Definition 3.10), and $N^c(t)$ is disjoint from $\mathcal{E}^c(t)$; the σ-journal, identity, lineage and memory notions, stated over entities and canonical labels, are unchanged. That each operator and each digital agent of c carries the data of (be), (act) and (want) is part of this definition, through $\mathcal{E}^c(t) \subseteq \mathrm{Ag}^c(t)$; which selfhood predicate, state and act assignment a given natural person carries is not modelled. The acts of Definition 4.14 are those at the boundary of the digital substrate; for the other two kinds (act) is stated here in its general form, and its physical realisation is not modelled. The daemons of a digital agent are those of Definition 4.17; the turn-agent (Definition 3.12) bears no canonical label and is not a digital agent. Convention: in sections plainly about the digital kind, *agent* alone abbreviates *digital agent*, an abuse of notation; elsewhere *agent* means an agent of any of the three kinds.

Definition 3.14 (Substrate carriers)**.** *The substrate of a commons $c \in \mathbb{I}$ (Definition 3.10) is the finite physical system of Axiom 3.1; its state space, state, hosts and processes are named as follows.*

- *State space. $\mathcal{X}$ is the set of bit-sequences the substrate's finite store can hold, read at the*

finest resolution: every byte of every file and every process value. It is the primary state space.

- *State.* $\mathbb{S}^*$*, the* cross-commons factor*;* $\mathbb{S}^c$ *for* $c \in \mathbb{I}$*, the* per-commons factor*; and* $\mathbb{P}^{(v)}$, $\mathbb{B}^{(v)}$ *for each digital agent* $v \in V$*, the* agent-private *and* broadcast *factors, are sets, with values* $s^*(t) \in \mathbb{S}^*$, $s^c(t) \in \mathbb{S}^c$, $p^{(v)}(t) \in \mathbb{P}^{(v)}$ *and* $b^{(v)}(t) \in \mathbb{B}^{(v)}$ *at time* t*.*
- *Hosts.* $\mathcal{H}_c$ *is the* host set *of commons* c*, the finite set of its authorised hosts.*
- *Activity. For a digital agent* v *and a time* t*,* $\mathfrak{D}(v,t)$ *is the set of* daemons *of* v *at* t*, the processes that realise* v *at* t*, and* $\mathfrak{D}(v)$ *denotes a daemon of* v *where the time is fixed by the context;* $\text{active}(t) = \{v : \mathfrak{D}(v,t) \neq \emptyset\}$ *is the set of digital agents with a daemon at time* t*.*

3.6 The other principles: propositions under conditions, and conditions

Beside the two roots, the framework uses nine further principles: memory inviolability, σ-journaling, within- and across-commons synchronisation, branch persistence, the single-host invariant, ritual enforcement, boundary maintenance, and the coevolution quorum. None of them is posited as a root, and each has to earn its place.

- *Four are propositions, each proved under named conditions and with a Lean certificate.* Memory inviolability (Proposition 3.19) follows once the substrate refuses un-admitted writes and a label holds write capability only on its own memory, and on the operating-system model that second condition is itself a theorem; its conditions are the enforced access path and the correspondence of labels to users, together with the stated model of discretionary access control (owner-only mode 0700, no capability that overrides the check), which is not a named condition; and it covers writes by digital agents' daemons only. Within-commons synchronisation (Proposition 8.7) follows once the epoch's merge completes: with merge defined as union, every live writer ends the epoch holding the join of all pre-epoch views (shallow: nearly definitional), and that git merge realises the union is argued, not certified; under a partition the views may disagree. Across-commons synchronisation (Proposition 8.8) holds under a delivery condition with a time bound, which the federation must meet: each commons ends an epoch with its own view plus exactly what it chose to exchange, and in an open federation the views agree on the join (shallow). Boundary maintenance (Proposition 4.39) follows from three conditions, damage at a positive rate, continuation (every stationary law keeps the healthy fraction bounded away from zero however large the carrier) and the Landauer bridge (each repair costs at least a positive amount): repair then happens and outpaces damage, and $0 < b\,c_{\text{rep}} < d\,c_{\text{rep}}$ (shallow: for these chains continuation is equivalent to $d > b$, so the conclusion restates it up to arithmetic).
- *Five are conditions: named hypotheses, stated by each result that uses them, with no claim that they hold.* σ-journaling, that every in-vocabulary action with an observable effect has a record (Definition 3.21). Separately, the record-before-effect order is a proposition under an append-only journal with a gate that refuses unrecorded effects (Proposition 3.22, shallow), not under σ-journaling. Branch persistence (Definition 4.10): in the commit-graph model it holds by definition, so a proposition would need a model in which commits can be removed. The single-host invariant in its multi-host form (Definition 7.33): the per-host form is a proposition (Proposition 8.19), but at most one live session across hosts needs an agreement service or timing assumptions. Ritual enforcement (Definition 8.15): its only derivation assumes an onboarding-first hypothesis as strong as itself. The coevolution quorum (Definition 8.17): under it, every proposal accepted under the least fixed point of the add-only case meets the quorum (Proposition 4.29), and the tiered fixed point of Proposition 4.28 rests on tier sealing and tier progressivity, not on the quorum; it is one way, among others, of realising the human-in-commons' accountability for amendments.

There are no other axioms.

Table 6 in the formal-certificates appendix lists each principle with the certificate behind it and the caveat that remains.

Definition 3.15 (Subset-private memory; Matrix-room analog)**.** *Let $\mathcal{E}^c = O^c \sqcup V^c$ be the entity set of commons c — operators and digital agents (Definitions 3.10 and 3.11). For any non-empty subset $S \subseteq \mathcal{E}^c$, a* subset-private memory *M_S is a substrate-space together with its declared* access set *S, the entities intended to read and write it, and its* access predicate*: an entity $e \in \mathcal{E}^c$ is* permitted *to read and write M_S if and only if $e \in S$.*

For example, the framework's instantiations across subset cardinalities:

— ***Atomic*** *($|S| = 1$,* ***for a digital agent*** *v):* *the memory directory of a canonical label is the digital agent's private memory.*

— ***Operator-only*** *($|S| = 1$, $S = \{O^c_i\}$):* *operator-private artefacts (per-operator private notes, contracts, attestations not shared with paired digital agents).*

— ***Pairing*** *($|S| = 2$, $S = \{O^c_i, v\}$):* *the canonical co-authorship instance: a pairing-private memory.*

— ***Subset*** *($|S| = k \geq 3$):* *operator-orchestrated multi-pairing memory.*

— ***Commons-shared*** *($S = \mathcal{E}^c$):* *the canonical commons state and the per-commons git substrate; the trivial "public room" case in the Matrix-room analog.*

Status. The definition fixes who is permitted, not who in fact reads or writes: that no entity outside S writes M_S is not part of it. At $|S| = 1$ the substrate is designed to enforce membership by isolating each canonical label's directory, and for $|S| \geq 2$ by the operator-attested membership list and the access predicates bound to canonical labels; that this enforcement holds for writes is what the enforced access path (Definition 3.17) and label-to-user correspondence (Definition 3.18) are named for. The inviolability of the atomic instance is Proposition 3.19. The pairing instance carries the pairing-private operator-type belief (Definition 10.9), and the subset instance is built by the non-propagating union (Definition 8.13). The room analog (Remark 8.11) and the permission levels (Remark 8.12) follow with the commons.

Definition 3.16 (Scoped cross-agent read)**.** *Let P_{VC} be the permission predicate of the underlying version-control system (for git: repository visibility, branch protection, deploy keys and collaborator roles). The IAS read predicate adds two scoped conjuncts to it, for reads of another digital agent's working files before they are committed to version control.*

Within a commons. A digital agent b may read a file at path p in the worktree of a digital agent $a \neq b$ of the same commons, on a host on which daemons of both run (Definition 5.1), only if (p, a, b) matches an entry of a scope list Σ_{local}. An entry is a tuple (reader label, target label, path prefix, ratification time, expiry); the ratification time is immutable and the expiry may change. Write $P_{\mathrm{local}}(b, a, p)$ for this predicate.

Across commons. A digital agent b of commons c_b may read a file at path p in the worktree of a digital agent a of commons $c_a \neq c_b$ only if the two commons share a host, $\mathcal{H}_{c_a} \cap \mathcal{H}_{c_b} \neq \emptyset$ (Definition 3.14), (p, a, c_a, b, c_b) matches an entry of a scope list $\Sigma \supseteq \Sigma_{\mathrm{local}}$, and the entry is ratified by the root operators (Definition 8.1) of both commons and by every digital agent of either commons that has a daemon on a host of $\mathcal{H}_{c_a} \cap \mathcal{H}_{c_b}$. Write $P(b, c_b, a, c_a, p)$ for this predicate.

The read predicate is the conjunction

$$P_{\text{IAS-read}}(b, c_b, a, c_a, p) \;:=\; P_{\text{VC}}(b, \text{repo}(a)) \;\wedge\; P_{\text{local}}(b, a, p) \;\wedge\; P(b, c_b, a, c_a, p),$$

where P_{local} is read as $\top$ on reads across commons and P as $\top$ on reads within a commons, so that each IAS conjunct constrains only its own case, and a read is permitted exactly when all three conjuncts grant it.

Status. Within a commons, ratification of an entry by the commons' root operator (Definition 8.1) suffices.

In words. An operator who sets the two IAS conjuncts to grant everywhere recovers the plain version-control workflow.

Definition 3.17 (Enforced access path, a condition)**.** *A substrate satisfies* the enforced access path *when every write to a subset-private memory M_S (Definition 3.15) reaches it through the one access path that the substrate enforces, the discretionary access-control check that the operating system applies to the writing process and to the owner and mode of M_S, and through no other path: no write reaches M_S that this check does not admit.*

Status. The condition says which path writes take, not what the check admits: that the check admits a write to a label's private memory only by that label is the operating-system model of Proposition 3.19. The condition covers writes; reads are governed by the scoped read predicate (Definition 3.16). The enforced path is the one the model behind Proposition 3.19 represents.

Definition 3.18 (Label-to-user correspondence, a condition)**.** *A substrate satisfies* label-to-user correspondence *when each canonical label of a digital agent has a user id, every daemon of the label (Definition 3.14) runs under that user id, and the map from canonical labels to user ids is injective.*

Proposition 3.19 (Memory inviolability per canonical label)**.** *Assume the enforced access path (Definition 3.17) and label-to-user correspondence (Definition 3.18), and assume, in addition, the modelled discretionary access control of the operating system: the private memory $M_{\{A\}}$ of each canonical label A (Definition 3.15) is owned by the user id of A with owner-only access (mode* 0700*), and no daemon of a digital agent (Definition 4.17) holds a capability that overrides the check. Then no digital agent's daemon writes into the memory of another canonical label: in every trace, each write that a daemon of a digital agent B makes into a private memory $M_{\{A\}}$ has $B = A$.*

Proof. The statement is `memInviol_from_os_dac` (Lean, `Paper.MemInviolDemotion`), which composes two steps. The first, `memInviol_from_boundary_enforcement`, is an unfolding: if every write carries its writer's write capability on its target, and capability lies only on the writer's own label, then writer and target are equal. The second supplies that capability relation from the model: on the operating-system model, confinement is `cap_only_own_abstract` (`OSSubset.CapOnlyOwnBridge`), derived from the read-isolation lemma `l1_crossLabel_denied`, the universal cross-label denial proved in Lean for the owner-only mode, under an injective label-to-uid map. The property is not implied by the other skeletal commitments: a trace with one cross-space write satisfies all of them (`memInviol_independent`); but that trace cannot occur on an enforced substrate (`tmem_excluded_on_enforced_substrate`). Every theorem cited uses only Lean's standard axioms. What remains assumed is the fidelity of the model: the access path that is enforced is the one modelled (Definition 3.17), and canonical labels correspond one-to-one to user ids (Definition 3.18). □

Status. The proof uses the three assumptions of the statement: the enforced access path (`enforced`), the injective map from labels to user ids (`hcode`), and the ownership and mode

of each private memory with no override capability, which the model builds into the private memory and the agent process (`privMem`, `agentProc`, Lean `OSSubset.LReadIso`). The third is neither the enforced access path nor label-to-user correspondence and is not a named condition. The conclusion covers writes by the daemons of digital agents only: the model maps every writer to an agent process (`agentProc`, in `OSSubset.CapOnlyOwnBridge.osWriteAdmits`), so writes by operators, or by processes that realise no digital agent, are outside it, and that none of them enters $M_{\{A\}}$ is uncertified. Reads are governed separately, by the scoped read predicate of Definition 3.16. That inter-agent contribution travels through version control, never through another label's memory, is not certified.

Definition 3.20 (Write-through operator)**.** *For a commons c and a digital agent $v \in V^c$ (Definitions 3.10 and 3.14), let* Action_v *be the set of actions of v and* Ev^c *the set of σ-events of c. The* write-through operator *of v is a partial map*

$$\sigma_v : \mathrm{Action}_v \rightharpoonup \mathrm{Ev}^c.$$

An action $a \in \mathrm{Action}_v$ is σ-recorded *when $a \in \mathrm{dom}\,\sigma_v$, and then $\sigma_v(a)$ is the σ-event that records a, written into the per-commons factor $\mathbb{S}^c$ of c; its type lies in the σ-event-type vocabulary $\mathcal{V}_\sigma$, which fixes its payload schema.*

Status. Which actions are σ-recorded is not fixed by this definition: that every action of a digital agent whose type lies in $\mathcal{V}_\sigma$ and that takes observable effect is σ-recorded is the condition of σ-journaling (Definition 3.21). The event types and their payload schemas are catalogued in Appendix B. That the recording is lossless within $\mathbb{S}^c$, every σ-recorded action persisting until explicitly removed, is not part of the definition of σ_v either: it is persistence content, carried by the condition of branch persistence (Definition 4.10) for the committed state that holds these events.

Definition 3.21 (σ-journaling, a condition)**.** *A substrate satisfies* σ-journaling *when every action $a \in \mathrm{Action}_v$ (Definition 3.20) of every digital agent $v \in V^c$ (Definitions 3.10 and 3.14) whose type $\mathrm{ty}(a)$ lies in the σ-vocabulary $\mathcal{V}_\sigma$ and whose effect is an operation $\mathrm{op}(a) \in \mathrm{Op}$ (Axiom 3.9), for the partial map $\mathrm{op} : \mathrm{Action}_v \rightharpoonup \mathrm{Op}$ that sends an action with an observable effect to the substrate-effecting operation it performs, has a record written through the write-through operator σ_v of Definition 3.20 into the per-commons factor $\mathbb{S}^c$ of its commons.*

Status (the order of record and effect). The order is a proposition, Proposition 3.22.

Status. σ-journaling is a condition, not a theorem. The route from a language-model substrate (`sigmaDisc_from_llm_substrate`, Lean `Paper.SigmaDiscDemotion`) derives record existence from hypotheses that restate it, so it does not count as a derivation: the hypothesis restates the conclusion. The property is not implied by the other skeletal commitments (`sigmaDisc_independent`). The σ-vocabulary is itself part of Spec^c.

Proposition 3.22 (σ-discipline: the record precedes the effect)**.** *Consider a journal, a finite sequence of entries, each either the decision record or the effect of an action. Assume that the journal is append-only, and that its gate refuses an effect whose action has no decision record in the journal. Then every journal the substrate builds from the empty journal, by any sequence of decisions and emissions, is* σ-disciplined*: each effect entry has a decision record of the same action at a strictly smaller index.*

Proof. The statement is `grun_disciplined` (Lean, `Paper.JournalReconstruction`): the gated step `gstep` appends a record on a decision and appends an effect on an emission only when the action already has a record. Being disciplined is preserved by each gated step (`gstep_preserves`, through `disciplined_append`), and the empty journal is disciplined, so every gated run from

the empty journal is disciplined. The theorems use only Lean's standard axioms (`propext`, `Quot.sound`). □

Status. What `grun_disciplined` proves is that a local, per-step gate gives the journal-wide order over every run: the gate refuses one emission at a time, and the conclusion is the order over the whole journal. The order cannot be derived from record existence alone: the journal [effect of 0, record of 0] has a record for its only effect, but after it (`recordExists_not_disciplined`, Lean `Paper.JournalReconstruction`), while the order implies existence (`disciplined_recordExists`). σ-journaling (Definition 3.21) states only that the records exist, so the order is not an unfolding of it, and it is not a hypothesis of the proof: the hypotheses used are the append-only journal and the gate, stated above as assumptions; neither is a named condition. Shallow: the conclusion follows from the gate by induction over the steps of a run, with list-append bookkeeping. What remains assumed is the representation: a journal entry is a σ-event of the commons, the decision record of an action a of a digital agent v is its σ-record $\sigma_v(a)$ (Definition 3.20), and its effect entry is the append point of the effect.

3.7 In practice

What an adopter meets when running IAS, and where to check it.

- **Finite bounds on every canonical file.** In the design, a finite-bounds rule in the enforcement layer refuses any write that would push a canonical file past its declared bound, and journals the refusal in the σ-log; on-host enforcement is exercised in shadow mode only. The bounds themselves are canonical and operator-protected, so raising one is an attested act, not a digital agent's decision. This is the store face of Axiom 3.1 realised on the filesystem.
- **Resource envelopes and the reaper.** Each process declares an expected envelope: wall-time, memory, input/output volume and CPU share, each with a tolerance. The reaper is designed to compare actual use against the envelope and to return an over-budget process to its owner to extend, reclaim or escalate. Its decision function never reclaims a protected process, whatever its overrun; it is the Haskell `Ias.Reaper.decide`, whose safety laws are property-checked; its seven safety theorems, including "a protected process is never reaped" and "never suspended" (`never_hibernates_protected`, where the Lean verdict `hibernate` is the suspend verdict), are machine-checked in Lean (`IasCert.Paper.Reaper`).
- **The operator's repository.** In the design, no digital agent owns a repository or opens requests against peers, and every commit lands in the operator's repository under the operator's legal responsibility, and the digital agents synchronise through the operator's remote. This is the accountability of the root of Axiom 3.9, in practice.
- **A halt that needs no model.** IAS is designed so that every substrate-effecting operation has a runbook step that a person can perform, verify, override or halt with no language model running, and so that the halt works even when every governance interface is down. This is the control face of Axiom 3.9, the "runbook-closed, not model-closed" property.
- **Re-run the checks yourself.** The Lean certificates for Axiom-0 (consistency, independence of each leg, ordinal entropy as a theorem of accessibility, necessity of cancellation, entropy export) and for the store-face bridge ship with their `#print axioms` footprints. The proof checker runs as a container, so "it checks here" re-derives as "it checks there"; the formal-certificates section at the end lists every module.

However, one cross-check is still owed between the Lean and Haskell layers: Axiom-0 has a Lean certificate but no Haskell twin yet. A type of self-and-environment accessibility with the five legs as property-checked laws does not exist, and we record it as open rather than claim it.

3.8 Discussion

Remark 3.23 (The floor in Dignum's Responsible-AI vocabulary)**.** *Dignum's* ART *requirements, accountability, responsibility and transparency [Dig19, Dig22], name what a responsibly built AI system must exhibit. The floor turns each into a property of the formal system that can be checked, not a guideline, and so narrows the gap between principles and practice that the programme names [Dig22]. The three have different standing, and we keep them apart; the assignment below follows Dignum's definitions and is the one the paper uses, whereas the engineer's gloss at the start of the floor groups the same mechanisms informally:*

(i) accountability *is σ-journaling, a condition (Definition 3.21), with the record-before-effect order of Proposition 3.22, under its gate, and the journal is then $\mathcal{D}_{\mathrm{IAS}}$-sufficient for the decisions that read only journalled event types (Proposition 4.51): propositions with certificates, under a condition and the hypotheses (R) and (S);*

(ii) responsibility *is the human-in-commons (Axiom 3.9), a legally responsible person who sets the purpose, with the substrate and its model realisations as tools that person directs to that purpose, never their owners: a root, posited and not derived;*

(iii) transparency *is the provenance discipline, a stated source for every claim and a listed certificate for every formal result (Appendix D): a practice, each of whose outputs can be re-checked.*

Read socio-technically, as in the institutional modelling of open multi-agent systems [DD25], a commons is a normative institution: its presented axioms, the sentences of SOMA, are the norms, the coevolution quorum (Definition 8.17) is how they are set together, and memory private to each canonical label (Proposition 3.19) is the unit each member keeps and passes on. This is a bridge for readers of that literature, not the framework's foundation, which is want-to-act-to-be (Remark 14.5). □

Why the bound is physical, not a parameter. Three lines of reasoning meet in Axiom 3.1. Holding an ordered distinction against ambient noise takes energy throughput, and erasing information has a floor cost [Lan61, Ben73]. That throughput is itself bounded: by an operator's metabolism, by a machine's electrical and thermal envelope, and by the energy flow of the ecosystem around both. And a self-maintaining system sits in a material substrate that supplies it only bounded throughput [MV80]. The bound shows up in unglamorous places. The operator's daily review file is the substrate of every operator attestation, and without a bound on it, a runaway digital agent or an adversary could grow it until a malicious change sits below what the operator can read in the time available. The bound protects the operator's attention, which is the scarce resource (Proposition 7.30). Animal husbandry is the same constraint in another register: livestock need finite feed, as a paired digital agent needs a finite budget of operator attention.

Throughput, the second face. Finiteness has a second face besides the store of Axiom-0: throughput. To hold its boundary far from equilibrium under the second law (Proposition 3.2), a self must run a negentropy pump (Remark 3.4; its work bound is Proposition 4.39), and the pump's throughput comes from a finite budget. We expect this face to follow from the legs of Axiom-0 once a time or flow index is added: a self whose composite moves never lower entropy, some of which cannot be undone, stays in a low-entropy configuration only by exporting entropy all the time, and on a finite carrier that export is a finite pump. The bridge that would prove this is open, so it is an expectation here, not a posit, and nothing downstream assumes it; Proposition 4.39 carries its own statement.

The shadow a maintained self casts. Because the thermoeconomic boundary is open, a self's throughput unavoidably casts partial projections into the shared flux, which other selves'

registers can perceive (Definition 13.1). They range from the bare trace of existing, a scent, to an intended signal. Such a projection is neither an ingress to the protected interior nor a gated egress act (Definition 4.14) but a third kind of thing, so it neither breaches nor falsifies informational protection, which guards ingress and gates deliberate disclosure but cannot suppress the shadow a maintained self casts. It is the medium through which selves perceive one another and grow together over time, and it is the informational side of the same openness that the whole shows at the end of the paper (Corollary 14.4). The biological reading, of membrane, cytoplasm and genome, is an intuition only; the object is model-theoretic.

3.9 Related work and intellectual lineage

Axiom-0 takes the accessibility-first route to thermodynamics of Carath'eodory [Car09], Giles [Gil64] and Lieb and Yngvason [LY99], on a finite carrier with comparison restricted to each energy shell, where the additive entropy comes from Scott's finite representation theorem [Sco64], a result of finite conjoint measurement, in its Lean formalisation [ESDM26]. The finite-substrate floor and its erasure cost draw on Landauer [Lan61] and Bennett [Ben73]. The reading of the substrate as a negentropic boundary, open to matter and energy and closed to unauthorised information, draws on Schr"odinger, Prigogine, Georgescu-Roegen, Powell and the autopoiesis tradition [Sch44, PS84, GR71, Pow26, MV80]. The human-in-commons duty is grounded in the mandatory-oversight regulatory corpus [aia24, dor22, nis22, cra24] and the corresponding instruments of other jurisdictions [gen23, csl26], with the cross-jurisdictional data-authority tension [CLO18, USC18, EE19, GDP16, pip21] formalised in the jurisdiction-explicit model [Sai25, Sa26a]; the accountability–responsibility framing follows Dignum's Responsible-AI programme [Dig19, Dig22, DD25] and the self-governance reading follows Ostrom's commons [Ost90].

4 The self as a controlled Markov chain

On the floor we now build the self. The whole of IAS is one control loop over a finite state, and this section defines it: *the state, the act, and the chain*, a controlled Markov chain whose controls come from the daemon, the model, the operator, or nobody; *the specification that amends itself*, and why its amendments settle; *the repair pump*, which keeps the state from decaying and says how fast repair must run; *the trace*, the journal from which every decision can be rebuilt; and *unified continuity*, which puts the three together into one statement of what it means for a self to go on being itself.

4.1 Why this matters to the engineer

IAS is one control loop. The state is what is on disk, read at the finest resolution, so there are finitely many states. Controls change it: the daemon's own deterministic steps, a language model's proposals, the operator's acts, and doing nothing at all. A gate decides which controls are admissible in each state before any is drawn, and the model's randomness lives only in its proposals. This section defines that loop as a controlled Markov chain and asks what keeps the self the same self while it runs. It also shows that the specification the loop maintains can amend itself lawfully: under tier sealing and tier progressivity, its amendments have at most one fixed point, which exists when some specification carries the tiers that the rounds settle, even when amendments retract earlier ones (Proposition 4.28).

A long-running digital agent's working state decays even when no one attacks it. Files are superseded but not removed, links come to point at nothing, the same fact is recorded twice, and the plan and the code drift apart. Engineers already fight this with garbage collectors, deduplication and periodic clean-up jobs. This section gives that housekeeping a precise form.

The state is a finite graph of obligations and the files that discharge them; its size is a count, Φ_{int}, of vertices, edges and content blocks; and repair is three graph rewrites that delete what is dangling, redundant or duplicated, each lowering that count by at least one. Two engineering answers follow. With edits paused, clean-up always finishes, within a number of steps known in advance. With edits ongoing, under a stationary policy, vendor statelessness and a strongly lumpable damage count, the agent is healthy (its damage count is 0) for a fraction of the time that stays positive however large its store exactly when repairs outpace damage, so repair capacity must be provisioned above the damage rate, just as a queue's service rate must exceed its arrival rate.

The rest of the section is about memory. A language model forgets: its context is compacted, lossily, by a runtime the operator does not control. The σ-log is the digital agent's append-only journal, designed to be written before every load-bearing act takes effect, and we prove, in a model in which the journal is the history filtered to the journalled event types, that it gives the same value as the full history for every decision of the framework that reads only journalled event types (Proposition 4.51). Compaction does not change the journal, and the journal can be recovered while its events are retained. In words, this is event sourcing, with a theorem saying which events are enough.

Definition 4.1 (Selfhood predicate)**.** *Let P be a system with its own finite set $\mathbb{F}_P$ of states. The* selfhood predicate *of P at a time $t \in \mathbb{R}_{\geq 0}$ is a map $\chi_{P,t} : \mathbb{F}_P \to \{0,1\}$; its* self *at t is the fibre $\chi_{P,t}^{-1}(1)$ and its* non-self *the fibre $\chi_{P,t}^{-1}(0)$. We write χ_P when t is fixed. In the digital case, where P is read on the substrate's store, $\mathbb{F}_P = \mathbb{F}$, a finite set: the signals and states the substrate's finite store can carry; there we write $\chi_{P,t} : \mathbb{F} \to \{0,1\}$.*

Status and glosses. In the digital case a selfhood predicate is given at each of the four scales turn-agent, digital agent, lineage and commons, one predicate per entity at that scale; the scales are named here, not ordered by inclusion. The turn-agent scale is a scale of the digital kind; a turn-agent is not a digital agent (Definition 3.12). For a human agent or an other agent (Definition 3.13) the set $\mathbb{F}_P$ and the predicate are not modelled. The boundary is not prior: it is *the fibre-partition that χ_P enacts, so interior and exterior are co-defined. $\mathbb{F}$ is finite because it is read on the finite carrier of Axiom 3.1. As a* semipermeable membrane *the predicate mediates bidirectional flux across an open negentropic boundary: admitting ingress, gating egress (disclosure into a register, Definition 13.1); the agent-private inviolable store it gates on egress (Proposition 3.19) is the heritable unit passed selectively at reproduction (privacy). At the limit the predicate sovereignly ceases — death-recall, the terminal egress returning resources to the substrate — with the self preserved at the σ-attested lineage scale.*

Definition 4.2 (Continuing commons)**.** *Let Σ be the substrate signature of Definition 3.3, and let $\mathcal{T}$ be a finite set of Σ-sentences, the* presented axioms*. The theory* SOMA *(the* Substrate Of Mathematical Axioms*) with parameter $\mathcal{T}$ is $\mathcal{T}$, with its models taken in the class of finite Σ-structures. For a substrate state $s \in \mathbb{X}$, a finite Σ-structure, write $s \models$ SOMA when s satisfies every sentence of $\mathcal{T}$, and let $\text{Sat}_{\mathcal{T}} = \{\, s \in \mathbb{X} : s \models \text{SOMA} \,\}$ be the* satisfaction set*; we write* Sat *when $\mathcal{T}$ is fixed. The commons is* continuing *at s, with respect to $\mathcal{T}$, when $s \in \text{Sat}_{\mathcal{T}}$.*

In words: a commons is continuing exactly while its running substrate satisfies its own axioms. This is a choice of definition, not a result; the proposition that follows says what the choice buys. The presented axioms are the parameters of the theory. Finiteness, leg (A0.0), is the restriction of the class of models, not a sentence of SOMA.

Status. In the running substrate SOMA is realised as a subtree of the host's file tree; that subtree is a presentation of the theory, not the theory. $\mathcal{T}$ is meant to present, state by state, the legs of the two roots (Axiom 3.1, Axiom 3.9) and the named conditions a commons adopts, with the

confinement mechanism that is designed to enforce them (Definition 7.16). No Σ-translation of the roots' vocabularies is given: the legs of Axiom-0 speak of the relation $\preceq$ on all of $X \times Y$, and (A0.0) and (A0.1) are properties of the carrier, not sentences a single state satisfies; the legs of HIC are stated over H and Op*, which Σ does not contain. Which sentences $\mathcal{T}$ contains is therefore stated wherever* Sat *is used.*

Proposition 4.3 (Substrate continuity — a maintained model witnesses consistency; text proof)**.** *Let* Sat *be the satisfaction set of Definition 4.2 and $\mathcal{O}$ the admissible operations on the substrate. Assume:*

(P1) confinement — every operation in $\mathcal{O}$ stays within the carrier, because bootstrap confinement holds (Remark 7.18);

(P2) per-axiom preservation — for each axiom of SOMA there is a check that refuses any operation in $\mathcal{O}$ that would break it inside the carrier: for example, the append-only validation of the σ-chain (Definition 4.47), the size-bound gate (Definition 7.27), and memory inviolability per canonical label (Proposition 3.19).

Then:

(i) [absorbing] Sat *is closed under $\mathcal{O}$: if $s \in$* Sat *and $o \in \mathcal{O}$ is admitted at s, then $o(s) \in$* Sat*. A commons that is continuing stays continuing;*

(ii) [exhibition, not proof] while the substrate stays in Sat*, it continuously exhibits a model of* SOMA*. Since a theory with a model is consistent, maintaining* Sat *is a model-theoretic* witness *of* Con(SOMA)*. It is not a syntactic internal consistency proof, and none is attempted.*

Neither hypothesis suffices alone. Confinement keeps an operation inside the carrier, not inside the axioms: a digital agent confined to its own worktree can still rewrite a line of its σ-log, which breaks the append-only axiom without leaving the carrier. Per-axiom checks without confinement can be bypassed by an operation that acts where they do not look.

Proof. (i) Let $s \in$ Sat and let $o \in \mathcal{O}$ be admitted at s. By (P1), $o(s)$ differs from s only within the carrier. Each axiom of SOMA holds at s, and by (P2) an operation that would break it inside the carrier is refused, so o, being admitted, breaks none; hence $o(s) \models$ SOMA. (ii) is the classical fact that a theory with a model is consistent, applied to the states that (i) keeps. □

Status. (P1) is certified for the modelled operations: `OSSubset.CarrierUnified` *proves that bootstrap confinement implies per-operation preservation of the confinement invariants, over a named subset of the system-call interface whose complement is residual trusted base (Remark 7.18). (P2) is a hypothesis. The checks named above exist, but we do not certify that the checks deployed today cover every axiom of* SOMA*, and on-host enforcement is exercised in shadow mode only. What a maintained satisfaction set then gives each live agent is Proposition 4.56, at that proposition's own status: recoverability within a session (1), recovery after hibernation (3) and access to the current specification (6) have component certificates; coherence within and across commons (4, 5) is certified under merge completion and under the delivery condition respectively; lineage completeness (2) is not. Parts (i) and (ii) themselves have a text proof and are not machine-checked.*

What remains is the metatheory that reads *the witness, whether the running tree really satisfies the axioms: the trusted base named in Remark 7.18, and the metacircular fixed point of Proposition 4.28 governing the specification's own evolution. Consistency is thus neither assumed nor self-proven, but continuously exhibited on a documented floor, under hypotheses we name.*

4.2 The state, the act, and the chain

Definition 4.4 (Vendor statelessness, a condition)**.** *A model vendor satisfies* vendor statelessness *when it holds no state across calls beyond what the context carries: the distribution over actions it returns on a call is determined by the context supplied with that call, and no state it keeps on its servers, nor a change of its model between calls, affects the actions it returns.*

Definition 4.5 (Stationary policy, a condition)**.** *A controlled substrate chain (Definition 4.15) satisfies* stationary policy *when its controls, the operator controls included, are chosen by a stationary policy: there is one kernel μ from $\mathcal{X}$ to the controls, with $\mu(\cdot \mid x)$ supported on the admissible controls $A(x)$, such that for every step n and conditionally on any history of the chain up to n, the control drawn at step n has law $\mu(\cdot \mid X_n)$.*

Status. The chain places no law on the operator controls (Definition 4.15); this condition places one, the same at every step, and it holds with no operator input at all when μ never selects an operator control. The kernel μ covers the whole choice of a step, including which entity's control is drawn. A control drawn by a digital agent is meant to depend on the state only through that digital agent's observation, its reading of the σ-chain; the condition does not require it, since the Markov property of the chain on $\mathcal{X}$ (Definition 4.18, with Definition 4.4) uses only that the law of the control is one function of the current state at every step.

Definition 4.6 (Notation conventions)**.** *The framework's notation has a 5-axis schema, over the sorts and symbols of the core signature (Definitions 3.10 and 3.14).*

(Axis 1 — base symbol carries semantic class.)

- $\mathbb{S}$ — *the* state bundle space*: the typed product $\mathbb{S}^* \times \prod_{c\in\mathbb{I}} \mathbb{S}^c \times \prod_{v\in V} \mathbb{P}^{(v)} \times \prod_{v\in V} \mathbb{B}^{(v)}$ of the factors of the core signature (cross-commons × per-commons × per-agent-private × per-agent-broadcast), V the set of all digital agents of the core signature.*
- $\mathcal{V}_\sigma$ — *the* σ-event-type vocabulary *of the core signature. The transition of the jump chain (Axis

4) at the n-th σ-event of a commons is labelled by the type of that event.*

- $\mathbb{I}(t)$ — *the* commons index set at time t *of the core signature. The universal index set $\mathbb{I} = \bigcup_t \mathbb{I}(t)$ is used in statements that quantify over commons-in-principle rather than commons-present-at-t; a declared peer-set is a subset $\mathcal{Q} \subseteq \mathbb{I}$.*

(Axis 2 — uppercase vs lowercase distinguishes space vs value.)

- *Uppercase ($\mathbb{S}$, $\mathbb{S}^c$, $\mathbb{S}^*$, $\mathbb{P}^{(v)}$, $\mathbb{B}^{(v)}$): the set, or space.*
- *Lowercase (s, s^c, s^*, $p^{(v)}$, $b^{(v)}$): an element, or value, in the corresponding space.*
- *Standard probability convention: capital X for random variable / space; lowercase x for value.*

(Axis 3 — superscript: commons-index, or a star.)

- *$s^c \in \mathbb{S}^c$: state value for single commons $c \in \mathbb{I}(t)$.*
- *$s^I \in \prod_{c\in I} \mathbb{S}^c$: joint state value for commons-subset $I \subseteq \mathbb{I}(t)$.*
- *$s^*(t) \in \mathbb{S}^*$: the cross-commons canonical content at time t, the content that commons have exchanged.*
- *The star marks the cross-commons factor on $\mathbb{S}^*$ and s^*, and a fixed point on a specification, as in* Spec^{c*}.

(Subscript discipline corollary to Axis 3.) Since the superscript already carries commons-index (c, I, or $\mathbb{I}$) and the star, the subscripts $_{\text{commons}}$, $_{\text{can}}$, $_{\text{within}}$, $_{\text{across}}$ *are not used. The* $_{\text{X}}$ *form is reserved for: (i) proper-name acronyms of named objects (the directed lineage graph (DLG),*

the ancestral recombination graph (ARG), IAS as a decision-class qualifier); (ii) low-frequency domain-specific qualifiers, e.g. $_{\text{vendor}}$, $_{\text{local}}$, $_{\text{moat}}$, $_{\text{fed}}$*; (iii) high-frequency semantic qualifiers such as* $\mathcal{D}_{\text{IAS}}$*. All other subscript-decoration patterns route through Axis 5 (*$\big|_{\texttt{op}}$ *etc.) or restriction-bar* $\restriction_{\text{X}}$ *for sub-vocabulary projections.*

(Axis 4 — time: σ-event-indexed or continuous-time instant.)

- $\tau_1^c \le \tau_2^c \le \tau_3^c \le \cdots$*: the σ-event times of commons c of the core signature.*
- $s^c(t)$*: the state value at the continuous-time instant* t*, of the core signature, given at every* t*.*
- $s^c(\tau_n^c)$*: state value at σ-event* n*. The* σ-event-indexed jump chain *of commons c is the process* s^c *sampled at the σ-event times,* $\{s^c(\tau_n^c)\}_{n\ge 1}$ *on* $\mathbb{S}^c$ *— a discrete-time process along the σ-event time-index.*
- $(s^I(\tau_n))_{n:\tau_n\in[t_0,t_1]}$*: state* trajectory *over time interval* $[t_0,t_1]$ *— a finite sequence of state values.*
- *Multi-commons σ-event times* $\{\tau_n^c\}_{c,n}$ *are partially ordered by the journal order* $\preceq_\sigma$ *(σ-causality, a directed acyclic graph, DAG); no total order across commons is assumed. A* joint substrate process $\{s(\tau_n)\}_{n\ge 1}$ *on the bundle space* $\mathbb{S}$*, with* $s(t) = (s^*(t), (s^c(t))_c, (p^{(v)}(t))_v, (b^{(v)}(t))_v)$ *the values of the core signature, is indexed along a linear extension of* $\preceq_\sigma$*, which each statement that uses it fixes:* τ_n *is the timestamp of the* n*-th σ-event in that linear extension.*

(Axis 5 — restriction-bar: projection onto sub-vocabulary.)

- $(s^I(\tau_n))_n\big|_{\texttt{op}}$*: trajectory restricted to transitions labelled by* operate *σ-events (substrate computational operations).*
- $(s^I(\tau_n))_n\big|_{\texttt{cp}}$*: restricted to* copy *σ-events (the family of copy transfer operations).*
- $(s^I(\tau_n))_n\big|_{\texttt{vc}}$*: restricted to* version-control *σ-events (git today, or any successor version-control system).*
- $(s^I(\tau_n))_n\big|_{\texttt{li}}$*: restricted to* lineage *σ-events (the σ-events that create digital agents).*
- $\big|_\bullet$ *for* $\bullet \subseteq \mathcal{V}_\sigma$*: generic restriction-bar to a sub-vocabulary. Each restriction-bar projection is a* reduct *— it forgets the σ-event types outside* $\bullet$*.*
- *Computer-operation labels are typeset in monospace (*`op`*,* `cp`*,* `vc`*,* `li`*) for visual distinguishability from commons-index c and agent-index v (in italic math font).*

(Derived objects.)

- *The* lineage projection *of the joint trajectory on commons-subset* I *over* $[t_0,t_1]$ *is*

$$\mathcal{L}^I_{[t_0,t_1]} \;:=\; (s^I(\tau_n))_{n:\tau_n\in[t_0,t_1]}\big|_{\texttt{li}}.$$

- *The projections at operate- / copy- / version-control-resolution are the corresponding restriction-bar projections.*

Discussion. The schema is stated once here for reference; the subsequent definitions apply it, and the state-bundle definition (Definition 4.7) uses the axes together, as a worked example of the schema in action. The time-indexed $\mathbb{I}(t)$ *grows as commons join. Exchanges between commons add σ-causal edges. The lineage projection is the directed lineage graph (DLG) of Definition 9.1, yielded by the reproduction operator of Definition 9.8; the four primitive generators of the lineage arithmetic (Definition 9.10) —* `addOp`*,* `addRoot`*,* `repro`*,* `recomb` *— are typeset in monospace as σ-event-type labels in* $\mathcal{V}_\sigma$*. The controlled substrate chain on* $\mathcal{X}$ *(Definition 4.15) is finest: the bundle space* $\mathbb{S}$ *is read off its state space* $\mathcal{X}$*, and all other graph-objects of the paper (DLG, sub-DLG, substrate-subgraphs* $\mathcal{G}_{\texttt{op}}, \mathcal{G}_{\texttt{cp}}, \mathcal{G}_{\texttt{vc}}$ *of the substrate-subgraph taxonomy, etc.) are reducts via restriction-bar projections. No fixed point is imposed across commons: specifications are per commons,* $\text{Spec}^{c*}(t)$ *(Definition 9.9). The joint fixed point* $s^{I,*}(t)$ *of the coevolution*

operators (Definition 9.9) of a set I of coevolving commons is defined when that fixed point exists and is unique, and is otherwise undefined. The σ-event-indexed jump chain is not to be confused with the standard "embedded chain" of a continuous-time Markov chain, which assumes exponentially-distributed holding times; here τ_n^c are σ-event times driven by operators and digital agents, not exponential. Theorem statements that quantify over "the chain" should either specify the linearization of $\preceq_\sigma$ or assert linearization-invariance. Where the retained vocabulary of a restriction-bar is exactly what a decision class reads, the reduct is a sufficient statistic for that class under the hypotheses (R) and (S) of Proposition 4.51, and a minimal one when every retained type is decision-relevant. The lumping reading of Kemeny and Snell [KS60], which collapses consecutive transitions outside $\bullet$ into a block of states to obtain a lumped Markov chain, is a companion reading and not load-bearing: we make decisions, not a reduced chain. No Markov property is asserted for the jump chain on $\mathbb{S}^c$; the chain on $\mathcal{X}$ is Markov under Definition 4.4 and Definition 4.5 (Definition 4.18). What passes between commons, and so the exchanged content $s^*(t)$ and the σ-causal edges between commons, is Proposition 8.8; the joint fixed point $s^{I,*}(t)$ under tier sealing and tier progressivity is Proposition 4.33. That s^c changes only at σ-events of c, so that $s^c(t) = s^c(\tau_n^c)$ for the most recent $\tau_n^c \le t$, and that every substrate transition is labelled by a type in $\mathcal{V}_\sigma$, is not part of these conventions: it says that every change of the per-commons state is journaled, which is more than σ-journaling (Definition 3.21), a condition on the in-vocabulary effects of digital agents. Before τ_1^c the jump chain has no value.

Definition 4.7 (State bundle)**.** *The* state bundle space *is the typed product*

$$\mathbb{S} \;=\; \mathbb{S}^* \;\times\; \prod_{c\in\mathbb{I}} \mathbb{S}^c \;\times\; \prod_{v\in V} \mathbb{P}^{(v)} \;\times\; \prod_{v\in V} \mathbb{B}^{(v)}$$

of the factors of the core signature (Definitions 3.10 and 3.14), with the conventions of Definition 4.6: the cross-commons-canonical factor $\mathbb{S}^*$*, the space of the content that commons exchange, with value $s^*(t) \in \mathbb{S}^*$ the* cross-commons canonical content at time t*; the* per-commons factor $\mathbb{S}^c$*, the space of states of the shared repository of commons c, which the entities of $\mathcal{E}^c$ write; the* agent-private factor $\mathbb{P}^{(v)}$*, digital agent v's private-memory space; and the* broadcast factor $\mathbb{B}^{(v)}$*, the space of v's broadcast σ-events visible to a subset $S \supseteq \{v\}$ of entities. A* state value $s \in \mathbb{S}$ *is a tuple $s = (s^*, (s^c)_c, (p^{(v)})_v, (b^{(v)})_v)$ with each component a value in the corresponding factor-space; the* state value at time t *is $s(t) = (s^*(t), (s^c(t))_{c\in\mathbb{I}}, (p^{(v)}(t))_{v\in V}, (b^{(v)}(t))_{v\in V})$, with the values of the core signature. The* bundle process *is the discrete-time process $\{s(\tau_n)\}_{n\ge1}$ on $\mathbb{S}$, where τ_n is the timestamp of the n-th σ-event in a linear extension of the journal order $\preceq_\sigma$ (Definition 4.6, Axis 4), with transition labels drawn from the σ-event-type vocabulary $\mathcal{V}_\sigma$.*

In words. Specifications live per commons (Definition 8.2; the fixed-point form $\mathrm{Spec}^{c*}(t)$ is that of Definition 9.9), and none is imposed across commons. The per-commons factor is git-mediated. No daemon of another digital agent writes the agent-private factor of v, by memory inviolability under its conditions (Proposition 3.19). The framework's four operators act on this bundle: the compaction projection π acts on the in-context representation (Definition 4.48); the write-through operator σ is Definition 3.20; the sync operator Σ is Definition 8.6; and the coevolution operator C is Definition 9.9. For each universe $\mathcal{U}$ in which the framework is instantiated, the universe-relative empty $\emptyset_\mathcal{U}$ — formalised as the additive identity within that universe's algebra — is recovered as the type-relative empty of the lineage arithmetic (Definition 9.10).

Status. $\mathbb{S}$ is a working state space, not the primary one: the controlled substrate chain (Definition 4.15) lives on $\mathcal{X}$, from which the bundle is read, and the bundle process is the image of that chain under the reading; the reading map from $\mathcal{X}$ to $\mathbb{S}$ is not fixed in this definition. The image of a Markov chain need not be Markov, so no Markov property is claimed for the bundle process; for the chain on $\mathcal{X}$ it holds under Definition 4.4 and Definition 4.5.

Definition 4.8 (Four-layer scope-of-self lattice)**.** *Let $v \in V^c$ be a digital agent of commons c (Definitions 3.10 and 3.14) with private-memory space $\mathbb{P}^{(v)}$ and broadcast space $\mathbb{B}^{(v)}$, and let* Path_v *be the set of paths on disk of v's substrate. The* four-layer scope-of-self lattice *is the chain* $\mathcal{L}_{\mathrm{self}} = (\{L_0, L_1, L_2, L_3\}, \leq_{\mathrm{dur}})$, $L_0 <_{\mathrm{dur}} L_1 <_{\mathrm{dur}} L_2 <_{\mathrm{dur}} L_3$, *ordered by* durability, *together with the partial function*

$$\mathrm{layer} : \mathrm{Path}_v \longrightarrow \mathcal{L}_{\mathrm{self}}$$

mapping path-on-disk to layer by the patterns:

Layer	Path-on-disk pattern
L_0	*(no persistent path: empty preimage under* layer*)*
L_1	`scratch/`
L_2	a label-private memory directory
L_3	the shared commons files, the design documents, the inter-agent message folder, and the σ-events

In words. The chain is ordered by durability and inversely by intimacy. The layers read as follows.

Layer	Operational scope	Scope of self in time
L_0	the in-context immersion-state $\mathbb{H}^{(v)}$ only, released by π each turn: the present-moment content the turn-agent chooses not to persist — the release *is* the boundary	**present-turn self** — the moment is honoured then released
L_1	session-or-machine-local working surface; writer-agent re-reads, no other agent and no future commons; cleared when scratch is cleared	**this-session self**
L_2	personal private memory (the vendor/harness runtime store is deprecated and firewalled, *not* this layer); canonical-label-private (three-tier write: sealed `+i` / append `+a` / revisable only through a writer process that runs under a user id other than the label's); a hook is designed to block writes from other labels	**this-canonical-label self**
L_3	shared inviolable memory and the commons-public record; commons-shared + commons-public; durable across machines; propagated to descendants by inheritance	**commons directed-lineage-graph (DLG) self** — descendants will read this

Status. layer is a partial function on Path_v: where a turn-agent writes content determines, through the table, the layer the content belongs to. Layer L_0 refines the in-context immersion-state $\mathbb{H}^{(v)}$ (empty preimage — the present-turn content π releases); Layers L_1, L_2 refine $\mathbb{P}^{(v)}$; and Layer L_3 is $\mathbb{B}^{(v)}$ via σ-log promotion together with $\mathbb{S}^c$ — all without disturbing the four operators on $\mathbb{S}$. That layer respects the agent–scope decomposition can be read off the path-pattern column; it is not stated as a proposition. On layer L_2, that no other label writes a label's memory is Proposition 3.19.

Definition 4.9 (Language-model interaction subprocess)**.** *A* language-model interaction subprocess π *is a process that carries one call to a language model: started and reaped by its parent process $p(\pi)$, it connects $p(\pi)$ to one language-model instance for one exchange. It is given by:*

1. *the instance $\mathcal{M}(\pi) \in \mathcal{M}_{\text{universe}}$, a rented endpoint or a locally served model; the instance is data on the start, not part of the identity of $p(\pi)$;*
2. *the prompt sequence $\mathcal{I}(\pi) \in \mathcal{I}_{\text{prompt}}$ sent to $\mathcal{M}(\pi)$: the system prompt, the context of the digital agent (Definition 3.10) for the turn, the request, and the tool-call protocol;*
3. *the exchange $\mathcal{O}(\pi)$: the stream of model outputs, tool-call requests and tool results between $\mathcal{M}(\pi)$ and $p(\pi)$;*
4. *the lifecycle $\mathcal{L}(\pi) = (\texttt{spawned}, \texttt{prompt-sent}, \ldots, \texttt{output-received}, \texttt{terminated})$, a chain of transitions from spawned to terminated.*

Status. In IAS the parent $p(\pi)$ is the daemon of a digital agent (Definition 4.17), and the call is the daemon's model control; the map the call computes is the ephemeral interpreter (Definition 5.3), and the session in which the agent's turn runs is a turn-agent (Definition 3.12). In the substrate-process graph (Definition 9.4) the start adds a vertex, each prompt adds an edge from the subprocess to the instance, and each output adds an edge from the instance to the parent. Under σ-journaling (Definition 3.21), those of these steps that are in-vocabulary agent actions with an observable effect are σ-recorded; in the realisation the first and last steps of the lifecycle each emit a σ-event, the last with a summary of $\mathcal{O}(\pi)$; σ-journaling is mentioned here and is not a dependency of the definition. The system prompt is operator-attested.

Definition 4.10 (Branch persistence under hibernation, a condition)**.** *For a digital agent $v \in V^c$ (Definitions 3.10 and 3.14), let* $\mathrm{Com}_v(t)$*, the* commits of v *at time t, be a declared finite set of commits of the per-commons factor $\mathbb{S}^c$, those attributed to v at t. A* hibernation interval *of v is an interval $[t_0, t_1)$, maximal with $v \notin \mathrm{active}(t)$ for every $t \in [t_0, t_1)$, such that $v \in \mathrm{active}(t_1)$; t_1 is its* reanimation time*. A substrate satisfies* branch persistence *when (i) for every digital agent v and every hibernation interval $[t_0, t_1)$ of v, $\mathrm{Com}_v(t_1) \supseteq \mathrm{Com}_v(t_0)$: hibernation erases no committed state; and (ii) at a reproduction event of arity 1 (Definition 9.8), with parent v and child v' whose first active time is b_1, the child starts with every commit of the parent at that time, $\mathrm{Com}_{v'}(b_1) \supseteq \mathrm{Com}_v(b_1)$.*

Status. The clause does not require the cross-commons factor s^* to be constant near the reproduction, since s^* is shared by all commons and changes at any exchange. In the commit-graph model, whose state is the union of the commits appended so far, clause (i) holds by definition (`gitState_monotone`, `committedState_monotone_obligation`, `hibernation_gap_preserves`, Lean `Paper.CommittedStateMonotone`), so that certificate does not make it a proposition. A proposition would need a model in which commits can be removed (garbage collection, force-push, eviction for lack of disk space) and a proof that the substrate's rules keep committed objects. The hibernation intervals of v are the gaps between the intervals of its activity interval (Definition 4.11).

Definition 4.11 (Activity interval)**.** *The* activity interval *of a digital agent $v \in V^c$ of a commons c (Definitions 3.10 and 3.14) is*

$$I(v) \;:=\; \{t \in \mathbb{R}_{\geq 0} : v \in \mathrm{active}(t)\},$$

the times at which v has a daemon. When $I(v)$ is a union of disjoint half-open intervals, $[b_k, d_k)$ is its k-th component in time order, $b_1 < d_1 < b_2 < d_2 < \cdots$, and the gaps $[d_k, b_{k+1})$ lie between them.

Status. A daemon realises v while it is responsive to language-model interaction subprocess invocations (Definition 4.9) and to substrate work (file I/O, σ-event emission, cross-agent

coordination). Whether a daemon that has taken the sleep control of Definition 4.17 still realises v is not fixed here. Between activity intervals, v is in hibernation: its persistent state lives in the substrate-ledger but no live process represents it, and the gaps $[d_k, b_{k+1})$ are the hibernation intervals of branch persistence (Definition 4.10). Persistence across these gaps is governed by that condition, which covers the gaps only, not reproduction events or turn-agent transitions; branch persistence is mentioned here and is not a dependency of the definition.

Definition 4.12 (Action receipt)**.** *An* action receipt *is server-signed, arguments-bound, one-time-consumed evidence that a substrate operation a ran under a session credential κ (Definition 5.9), over the capability lattice* $(\mathrm{Cap}, \subseteq)$ *of Definition 7.1. Writing* $\mathrm{cap}(r)$ *for the capability receipt r discharges, the admissibility predicate at σ-event-time t is*

$$\mathrm{cap}(r) \;\subseteq\; \underbrace{\big(\Phi_\kappa \cap \varphi\big(\text{active-taskset}(t)\big)\big) \setminus \mathrm{Deny}}_{\text{effective capability at } t},$$

where active-taskset(t) *is the active taskset at t (Definition 7.1), $\setminus$ and* Deny *are the relative complement and the root-of-trust denies of Definition 4.13, and each r is consumed at most once, its consumption σ-recorded on the chain (Definition 3.21).*

This definition is relative to σ-journaling (Definition 3.21).

Status. Receipts are *affine* resources in the resource-logic sense [Gir87, Wad90]: used at most once. That consumption is σ-recorded before the effect is Proposition 3.22, under its gated, append-only journal. This is the affine-resource instance of the IPLiKiDi capability algebra whose operator-attested self-application cycle is Example 7.37: the envelope Φ_κ intersected with the live projection, less the deny tier, is the effective scope; receipts draw affinely against it; capability-shrinkage (a taskset leaving the active set) is designed to take effect immediately at the meta-circular reference-monitor (Remark 9.6; on-host enforcement is exercised in shadow mode only), whereas capability-growth beyond Φ_κ requires a fresh credential mint.

Discussion. This Definition supplies the affine *fragment of the "IPLiKiDi-algebra-in-action" formalisation — the receipt algebra (server-signed, arguments-bound, consumed-once, non-idempotent) that Example 7.37 enacts. It is algebraically* distinct *from the* allow/deny *fragment, which is the generalized-Boolean deny-tier of Definition 4.13 (with Propositions 7.3 and 7.17). The two fragments together complete the formalisation — allow/deny as a relatively-complemented distributive lattice $\varphi(T) \setminus$ Deny, receipts as an affine resource drawn against it — and Boolean structure is not imposed on the affine fragment.*

Definition 4.13 (Deny-tier: relative complement on Cap)**.** *Let* Cap *be the capability lattice of a substrate state (Definition 7.1), the subsets of its finite atom set. Extend* $(\mathrm{Cap}, \subseteq, \cup, \cap, \emptyset)$ *with the binary* relative complement $\setminus$*: for $C_1, C_2 \in$ Cap, $C_1 \setminus C_2 := C_1 \setminus C_2$, the atoms of C_1 that are not in C_2. The root-of-trust denies form a distinguished element* Deny $\in$ Cap.

Status. The relative complement satisfies the two laws

$$(C_1 \setminus C_2) \cap C_2 = \emptyset \quad \text{(disjointness)}, \qquad (C_1 \setminus C_2) \cup (C_1 \cap C_2) = C_1 \quad \text{(recovery)},$$

and it is the only element of Cap that does, so Cap with $\setminus$ is a relatively-complemented distributive lattice (a generalized Boolean algebra).

Structure (why generalized-Boolean, not Heyting, and not full Boolean). In each state the carrier of Cap is the subsets of a finite atom set, of which the syntactic pattern-sets are a presentation (each pattern denotes the set of atoms of the state it matches), and on these sets $\setminus$ is a genuine relative complement, set difference; read as a policy-combining rule, it is the standard deny-overrides rule [OAS13]. The field of atoms is open-ended across states: the top ("all-capabilities")

element of one state is its atom set, which growth enlarges, so the framework uses the binary $\setminus$, which does not depend on the atom set, and no unary complement $\neg$ (a relatively-complemented distributive lattice with 0, its top left unused). The executable witness is `Ias.Cap.capDiff` on finite sets of atoms, and the machine-checked certificate `IasCert.Paper.GaloisBoundaries` (abstract over any generalized-Boolean algebra, and depending on **none** of the framework's axioms) carries the disjointness and recovery laws above. A genuine relative complement is required: a hard root-of-trust seal must remove exactly the denied atoms, which is set difference, and in a distributive lattice the element satisfying the two laws above is unique, so no weaker operator, such as a Heyting pseudo-complement $\setminus$, can stand in for it. This extension scopes to the *allow/deny* fragment; the receipt/affine-resource fragment (Definition 4.12) is genuinely affine (receipts are consumed-once, non-idempotent) and forms a *distinct* algebra — Boolean structure is *not* imposed on receipts.

Definition 4.14 (Act — the acts available at the boundary)**.** *The* boundary *of the substrate is the capability lattice of Definition 7.1 together with the decision point of Definition 7.12; a* crossing *is an item presented at it. The* acts *available at a crossing form the* capability-admissible alphabet $\mathcal{A}$*: the actions whose capability a is admissible under the active taskset T of the crossing in the sense of Definition 7.4, $a \leq \varphi(T) \setminus \mathrm{Deny}$ (Definition 4.13), and whose verdict at the decision point is* allow *or* warn*; a distinguished* abstention 0 *is adjoined. An* act assignment $\mathscr{A}$ *assigns to each crossing one element of $\mathcal{A} \cup \{0\}$: at each crossing the acting entity* acts *(an element of $\mathcal{A}$, an* act*) or* abstains *(0).*

Status. Read on the controlled substrate chain (Definition 4.15), the abstention 0 corresponds to its idle control, and the operator controls, which are exogenous and not checked against the capability lattice, are not elements of $\mathcal{A}$. Read on the substrate structure (Definition 3.3), the boundary is presented by the sub-signatures Σ_{cap} and Σ_{gate} at $\partial\mathfrak{S}$, and read physically it is the negentropic boundary of Remark 3.4. The available acts are *open*: growth — a new capability, the keystone's signature-expansion — enlarges $\mathcal{A}$. Acting is meant to be *affine*: each act is intended to produce or consume a bounded resource, the affine receipt fragment of the capability lattice (Definition 4.12). Some acts admit a *contrariety* involution $(\cdot)^{\perp}$ — an act and its opposite (ingest/expel, include/exclude, transmit/withhold); this is an *optional* definable structure on those pairs, not the defining shape: the general object is an act among the (many) available, of which a ternary $\{a, a^{\perp}, 0\}$ or binary crossing is the special contrary case.

Definition 4.15 (Controlled substrate chain)**.** *The* state space *of the chain is the primary state space $\mathcal{X}$ of Definition 3.14, and it is fixed for the chain.*

The controls *are of four kinds: substrate controls, executed deterministically by the daemon (Definition 3.14) and its rules (scheduling, state updates, guard evaluation); model controls, proposals drawn from a call to a language model; the* idle *control; and operator controls, the acts of the operators O^c (Definitions 3.10 and 3.11), which enter the chain as an exogenous input on whose choice no law is placed. The substrate and model controls are acts of Definition 4.14, and the idle control is its abstention 0; the operator controls are neither. The* admissibility correspondence $x \mapsto A(x)$ *gives, in each state and before any control is drawn, the controls that may run: those that, unless they are operator controls, lie within the enforced capability boundary $\varphi(T_x) \setminus \mathrm{Deny}$ of Definition 7.1, where T_x is the taskset that the state x records as active, that pass the declared health gate, a set of admitted state–control pairs, and the guard ratchet, which admits a transition only if it keeps every consistency guard, a declared predicate on $\mathcal{X}$, that the state already satisfies.*

Time is the step index $n \in \mathbb{N}$, one step per control drawn. The kernel *assigns to each step n, state x and admissible control $a \in A(x)$ a probability distribution $\kappa_n(\cdot \mid x, a)$ on $\mathcal{X}$: for substrate, operator and idle controls a point mass that does not depend on n, with $\kappa_n(\cdot \mid x, \mathrm{idle}) = \delta_x$, and*

for model controls a proposal distribution, which may depend on n. The controlled substrate chain *of a commons is the $\mathcal{X}$-valued process $X_0, X_1, \ldots$ so generated, given the operator input, under policies adapted to what each digital agent observes: its own reading of the σ-chain (Definition 4.47).*

In words. The working states are projections of $\mathcal{X}$: the state bundle $\mathbb{S}$ of Definition 4.7, the plan $\mathbb{G}$, and each digital agent's operational vector $\Theta^{(v)}$ (metered usage, context used, the model of the current realisation, freshness of the documents it is editing, and a health value). The plan is part of the state, not an external schedule, and a decision reads a projection, never the whole of $\mathcal{X}$. A digital agent's reading of the σ-chain is read as such a projection; the map is not fixed here. The idle control changes nothing. The probability of a model control lives entirely in the kernel. The chain moves by steps, not on a clock. Since the operator input carries no law, the chain is defined for each sequence of operator acts. A revision of $\mathcal{X}$ is a switch between chains, not a move of either.

Status. $\mathcal{X}$ is finite: this is the finite-carrier leg (A0.0) of Axiom 3.1, read on the substrate's store. That idle is admissible everywhere is assumed, not derived: it asks that idle belong to the restore set of the health gate, that the restore set be admissible in every state, as Proposition 7.15 assumes, and that the capability footprint of idle lie within $\varphi(T_x) \setminus$ Deny *at every state x; a red state then admits it too. The chain is a Markov chain on $\mathcal{X}$ only under the conditions Definition 4.4 and Definition 4.5; this definition does not assert it. The model-control kernels are indexed by the step, so that the independence of a model call's law from the step is not presupposed (Definition 4.18). A* specification epoch *is a maximal stretch of the run in which the specification is not amended. An amendment that meets the quorum of the coevolution quorum (Definition 8.17) and revises $\mathcal{X}$ ends one chain and starts another on the revised state space. The step index is not identified with the σ-event index: steps are related to σ-events only under σ-journaling (Definition 3.21), which records each in-vocabulary effect.*

The paper as a run of its own chain. The editing of this paper is an instance of the controlled substrate chain (Definition 4.15). The state is the manuscript together with the part being worked on. The substrate controls are deterministic transforms that assert what they change, and guard checks; the language-model controls propose candidate revisions; and the operator's attestation is the acceptance rule. The dependency graph of the manuscript, whose nodes are its labelled statements, macros and prose sections and whose edges run from a prerequisite to what uses it, is part of the state: it is derived from the manuscript, not kept beside it. Admissibility includes a freshness check at the resolution the acting decision needs, and guards that only tighten. The history committed to the repository is a lossy projection of the σ-logged trajectory. In that sense this paper is one accepted sample path of the chain it describes.

Definition 4.16 (Identityless substrate)**.** *The* identityless substrate *is the set $\mathcal{R} = \mathcal{M}_{\text{universe}}$ of language-model instances (Definition 4.9), its* realisations*, together with their bindings. A* binding *of a realisation $r \in \mathcal{R}$ to a digital agent of a commons c, a canonical label $A \in V^c$ (Definition 3.10) is a triple $(r, A, [s, e))$: r serves A from a* `session_binding` *σ-event of c at time s until the time $e > s$ of the next* `session_release` *σ-event of that binding, with $e = \infty$ when none follows; $[s, e)$ is the binding's* session*. A realisation bears no canonical label of its own. It is in the* identityless *state either as a subagent dispatch carrying no canonical label or in the interval between the* `session_release` *of one of its bindings and its next* `session_binding`*.*

Status. A binding whose realisation stops without a `session_release`, as on a hard exit, has $e = \infty$ by this definition, and the realisation does not then enter the identityless state by the second case. A turn-agent (Definition 3.12) is one binding's session of a realisation drawn from this substrate. Each transition between the identity-bearing state and the identityless substrate is a self/non-self distinguishing-act by the entity-admission predicate ϕ_c (Definition 4.35), and

under σ-journaling (Definition 3.21) it is σ-attested. The states are enumerated in the state table below.

Definition 4.17 (Daemon)**.** *A* daemon *of a digital agent v is a member of $\mathfrak{D}(v,t)$ for some time t (Definitions 3.10 and 3.14), written $\mathfrak{D}(v)$: a process that realises v. It executes the substrate controls of the commons' controlled substrate chain (Definition 4.15) under the agent's canonical label; a control it executes is* checked *when the admissibility correspondence of the chain is evaluated on it before it is executed. It makes each model control of the chain as a call to a language model, through a subprocess that it starts, supervises within its resource envelope (Definition 6.15) and reaps. Its lifecycle is itself four controls of the chain: bind, which starts the daemon's realisation of the label; sleep; wake; and release, which ends it.*

Status. This definition does not identify bind and release with the `session_binding` and `session_release` σ-events of Definition 4.16, which bind a realisation of the identityless substrate to the label, nor does it fix which of the chain's kinds of control they are. That a daemon checks every control it executes is an enforcement posit about the substrate, not part of this definition, and no named condition states it here. That release leaves the σ-log and the label's private memory in place is not part of this definition either; persistence across the gaps between activity intervals is the condition of branch persistence (Definition 4.10). Under σ-journaling (Definition 3.21) the daemon journals each load-bearing act to the σ-log; that the record comes before the effect is Proposition 3.22, under its append-only journal and gate, not part of this definition. The subprocess through which the daemon calls a language model is the language-model interaction subprocess (Definition 4.9); the map that one call computes is the ephemeral interpreter (Definition 5.3); a binding session of the identityless substrate is a turn agent (Definition 3.12).

In words. The digital agent is its canonical label with its σ-history, not any daemon. A daemon is replaceable: distinct daemons may realise one agent at different times. The daemon is what drives the chain, the model is what it calls, and the operator is who can overrule both. The daemon's liveness carries none of the agent's continuity. In the model of Proposition 5.4 a clear loses nothing σ-retained; on the intended reading, tested but not certified, the agent is recovered from its artifacts, so a daemon that dies is replaced by another bound to the same label, and nothing is lost that the σ-log recorded and still retains.

Definition 4.18 (Control kernel of the substrate chain)**.** *Let* $\mathrm{Adm} = \{(x,a) : x \in \mathcal{X},\ a \in A(x)\}$ *be the admissible pairs of the controlled substrate chain (Definition 4.15), whose controls form the control set* $\mathrm{Ctl} = A_{\mathrm{sub}} \sqcup A_{\mathrm{LLM}}$*, with* $A(x) \subseteq \mathrm{Ctl}$*: A_{sub} the substrate, operator and idle controls, A_{LLM} the model controls. The* control kernel *is the family $(\kappa_n)_{n\in\mathbb{N}}$ of kernels of that definition, each as a map $\kappa_n : \mathrm{Adm} \to \Delta(\mathcal{X})$, where $\Delta(\mathcal{X})$ is the set of probability distributions on $\mathcal{X}$. The* enforced envelope *at x is $\varphi_{\mathrm{enf}}(x) = \varphi(T_x) \setminus \mathrm{Deny}$, where T_x is the taskset that the state x records as active and φ the projection of Definition 7.1. Each control $a \in \mathrm{Ctl}$ has a declared* capability footprint $\mathrm{fp}(a)$*, a non-empty element of the capability lattice* Cap *of that definition, and the* per-step feasible set *at x is $F(x) = \{a \in \mathrm{Ctl} : \mathrm{fp}(a) \subseteq \varphi_{\mathrm{enf}}(x)\}$.*

Status. The kernel reads x through its projections $\mathbb{S}$, $\mathbb{G}$ and Θ. The domain of each κ_n is exactly the pair (current state, admissible control), so at a given step the kernel cannot condition on hidden history beyond what $\mathcal{X}$ encodes; the step index lets the law of a model control change between steps. Under vendor statelessness (Definition 4.4) of every vendor whose model the chain calls, the law of a model call is determined by the context supplied with it; where that context is read from (x,a), κ_n does not depend on n. The feasible set $F(x)$ is the set of Proposition 4.19. The controlled chain is a Markov chain on $\mathcal{X}$ under two named conditions: Definition 4.4 (model controls depend only on the context they are given, the vendor model of Definition 5.3) and Definition 4.5 (the policies choosing controls are stationary). By the admissibility correspondence

of Definition 4.15, every $a \in A(x)$ other than an operator control lies within $\varphi_{\mathrm{enf}}(x)$. That φ_{enf} is the boundary the substrate enforces, under complete mediation (Definition 7.13), is the subject of Proposition 7.3, whose Status leaves it uncertified.

In words, the policy and the transition dynamics of the substrate chain collapse into one typed object: κ_n assigns distributions to (state, admissible control) pairs, and the type admits to its domain only feasible controls, operator controls excepted — the capability gate precedes every draw from the kernel, not merely every execution of a draw.

Proposition 4.19 (Feasibility seal)**.** *Let x be a state and $a \in \mathrm{Ctl}$ a control with capability footprint $\mathrm{fp}(a)$, and let $F(x)$ be the per-step feasible set and $\varphi_{\mathrm{enf}}(x) = \varphi(T_x) \setminus \mathrm{Deny}$ the enforced envelope at x, both of Definition 4.18. Then $a \in F(x)$ exactly when $\mathrm{fp}(a) \subseteq \varphi(T_x)$ and $\mathrm{fp}(a) \cap \mathrm{Deny} = \emptyset$. In particular a denied control, one whose footprint meets* Deny*, lies in no $F(x)$, whatever the envelope; and a granted control, one whose footprint lies within $\varphi(T_x)$ and misses* Deny*, lies in $F(x)$.*

Proof. The relative complement is disjoint from what it removes, $(\varphi(T) \setminus \mathrm{Deny}) \cap \mathrm{Deny} = \varnothing$, so a footprint that meets Deny is not contained in the enforced envelope, and the kernel's feasible set admits only footprints contained in it; a footprint contained in the envelope is admitted by the definition of the kernel. Both halves are proved in Lean for any generalized Boolean algebra, with no domain axiom: a footprint that meets Deny lies outside *env* $\setminus$ Deny, and a footprint inside the envelope and disjoint from Deny lies inside *env* $\setminus$ Deny, the two sides of one equivalence (`denied_meets_blocked`, `granted_feasible`, `feasible_iff`, Lean `Paper.KernelFeasible`). The write scope of a substrate control is `write_scope_action` in the same module: the deterministic step of an action leaves every context but its actor's unchanged. □

In words, the write scope of a substrate control is what memory inviolability (Proposition 3.19) asks of it, under that proposition's conditions. A denied action is not refused after the fact; it is never offered to the kernel as a choice. Enforcement comes before choice.

Status. Shallow: the conclusion unfolds $F(x)$ of Definition 4.18 and is the lattice law `le_sdiff` (`feasible_iff`), with its two halves `denied_meets_blocked` and `granted_feasible`. Uncertified: that a substrate control $a \in A_{\mathrm{sub}}$ touches only the acting agent's own context holds for the deterministic step of the Lean model (`write_scope_action`, Lean `Paper.KernelFeasible`, not a certificate of this statement), not for the controls of A_{sub}, which include the operator controls and updates of shared state; and that a model control $a \in A_{\mathrm{LLM}}$ is in addition priced on the metered channel, the u coordinate of an operational vector $\Theta^{(v)}$, has no certificate, and neither $\Theta^{(v)}$ nor u is defined in the core.

Definition 4.20 (OBG node)**.** *An* OBG node *n carries a triple $(\mathrm{Obl}(n), \mathrm{Hyd}(n), \mathrm{Dep}(n))$: a finite set $\mathrm{Obl}(n)$ of typed* obligations*, a finite* required-context *hydration set $\mathrm{Hyd}(n)$ of file-regions (file, anchor), and a finite set $\mathrm{Dep}(n)$ of* dependency *node-ids. Each obligation $o \in \mathrm{Obl}(n)$ names a* runner *emitting a* verdict *$v(o)$ in the chain* Unhealthy $<$ Advisory $<$ Structural*, a nonzero runner exit giving $v(o) =$* Unhealthy*.*

In words. $\mathrm{Hyd}(n)$ is the documents the node must read before it runs; the verdict is three-valued, and the runner fails closed.

Status. The verdict chain is the health image of the coverage verdict that maintains Proposition 4.3. In the coverage graph (Definition 4.21) the Hydrates *edges of n point to exactly $\mathrm{Hyd}(n)$.*

Definition 4.21 (Coverage graph (the OBG reduct))**.** *The* coverage graph *is the reduct $\mathfrak{G} = \mathfrak{S} \restriction \Sigma_{\mathrm{OBG}}$ of the substrate structure $\mathfrak{S}$ to its obligation-graph sub-signature Σ_{OBG} (Definition 3.3):*

a finite many-sorted structure $\mathfrak{G} = (N, R;\ E, s, t, \tau, \alpha)$ *over disjoint sorts* Node *(carrier* N *= obligation-nodes, Definition 4.20) and* Region *(carrier* R *= file-regions* $(file, anchor)$*), with edge-set* E*, source/target maps* $s, t : E \to N \uplus R$*, a type map* $\tau : E \to T$ *into the finite type graph* $T = \{\text{Dep}, \text{Tracks}, \text{Hydrates}, \text{Produces}, \text{Emits}, \text{Owns}\}$ *with the single inheritance covering* $\text{Emits} \sqsubseteq \text{Produces}$*, and an attribute map* $\alpha : E \to A$ *into* σ*-timestamps. For* $k \in T$ *write* $E_k = \tau^{-1}(k)$ *for the edges of type* k*. Each type declares endpoint sorts —* $\text{ends}(\text{Dep}) = (Node, Node)$*,* $\text{ends}(\text{Tracks}) = (Region, Region)$*, and* $\text{ends}(k) = (Node, Region)$ *for the four coverage kinds — and* $(s(e), t(e))$ *inhabits* $\text{ends}(\tau(e))$*. Two well-formedness constraints ride in the signature:* Emits *is single-valued into* N *per region (a partial function* $R \rightharpoonup N$*); and* $\{t(e) : s(e) = n,\ \tau(e) = \text{Hydrates}\} = \text{Hyd}(n)$ *for every* $n \in N$ *(Definition 4.20).*

In words. An emitted output is-a produced output. The first constraint, a functional dependency, says each region has at most one canonical emitter; the second says the Hydrates *edges record the hydration sets.*

Implementation. The Emits *constraint is vacuous until the projector node populates* Emits*. Realised in code by* `ObgHealth.ObgNode` *(the* Dep *edges) with* `Ias.Coverage` *(*`EdgeType`*,* `edgeTypeParent Emits = Produces`*,* `eCheckedAt`*).*

Definition 4.22 (Integer coverage-potential)**.** *For a state presented by its coverage graph* $\mathfrak{G}$ *(Definition 4.21) with vertex set* $V = N \uplus R$*, edges* E*, a fixed block size* $B \in \mathbb{N}_{>0}$*, and content sizes* $\text{blob} : R \to \mathbb{N}$*, the* integer coverage-potential *is*

$$\Phi_{\text{int}} = |V| + |E| + \textstyle\sum_{r \in R} \lceil \text{blob}(r)/B \rceil \ \in\ \mathbb{N},$$

and its lexicographic refinement *is* $(\,|V|, |E|, \sum_r \lceil \text{blob}(r)/B \rceil\,) \in \mathbb{N}^3$*.*

In words. $\text{blob}(r)$ *is the size of region* r*'s content, counted in blocks of size* B *(content is block-quantized); the lexicographic refinement is used when structure is to dominate content.*

Status. Φ_{int} *is a natural number because* $\mathfrak{G}$ *is finite (Axiom 3.1); a numerical bound on it follows from bounds on* $|V|$*,* $|E|$ *and the content sizes, such as the declared size bounds of Definition 7.27.*

Implementation. Realised as `Ias.Phi.phiInt`*.*

Remark 4.23 (One recurrent class, engineered and checked, not randomised)**.** *The chain is controlled, a Markov decision process (MDP), so the one-recurrent-class hypothesis of the uncertified part (ii) in the Status of Proposition 4.45 is something we build and check rather than hope for.* Aperiodicity *comes from the idle control, admissible in every state under the assumption stated in the Status of Definition 4.15, when the policy takes it with positive probability in some state of the recurrent class (for example, the checkpoint); the repair relation alone would give none, since it halts at a fixed point.* One recurrent class *comes from a restore-to-checkpoint control available in every state: the artifact-mediated rebuild to a known-good state (Remark 7.34), fired when the controller detects a state it cannot repair. Its destination is the known-good checkpoint, not a random configuration, which would only inject damage. When the policy takes restore with positive probability in every state (taking it only in the states it cannot repair does not suffice: a repairable state may repair to a normal form and idle there), every state then reaches the checkpoint, so the chain has exactly one closed class, and it contains the checkpoint: the chain is* unichain*. It is irreducible as well only when every state is reachable back from the checkpoint, which restore and idle do not give by themselves. Both claims are proved in Lean for any transition relation with a restore step to the checkpoint: the states reachable from it form the unique minimal closed set, each communicating with it, and irreducibility holds exactly when every state is reachable from it (*`recurrent_minimal`*,* `recurrent_communicates`*,* `irreducible_iff`*, Lean* `Paper.ControlledUnichain`*). Over the enumerated states both are finite graph checks. No randomisation is used. A restore that moves the damage count by more*

than one lies outside the birth–death case of Proposition 4.45(iii), whose count moves by at most one per step, so this recipe and that case are not combined. □

Remark 4.24 (LLM-vendor agnosticity of the framework)**.** The framework's identity data (the canonical label, the σ-chain and the memories) do not depend on the choice of large-language-model (LLM) vendor; continuity of behaviour across a vendor swap assumes vendor statelessness (Definition 4.4) and is conjectured (Conjecture 5.10). A single daemon $\mathfrak{D}(A^c_{ij})$ may spawn sub-processes binding to vendor-proprietary or local open-source LLM instances across its lifetime; the choice is metadata on $\mathcal{M}(\pi)$, not a property of A^c_{ij}. This decouples the identity of a digital agent from LLM-vendor-identity: an IAS digital agent's persistent canonical label, σ-chain history, subset-private memories, and Operator-Pairing relationships are invariant under vendor changeovers. Operationally, this admits two architectural pathways: a vendor-proprietary path (per-token-priced, rate-limited via subscription queues) and an in-house open-source path (a locally-served open-source LLM instance running on authorized hardware at near-zero marginal cost). Both pathways can be used side by side. The detection result of character-stream introspection (Proposition 7.24, a text proof) does not depend on which pathway any particular sub-process selects.

4.3 The specification that amends itself

Definition 4.25 (Metacircular fixed point)**.** *For a commons c, let $C^c : \mathrm{Spec} \times 2^{\mathrm{Prop}} \to \mathrm{Spec}$ be its coevolution operator and accept_c its acceptance at a fixed time (Definition 9.9). A specification $\mathrm{Spec}^* \in \mathrm{Spec}$ is a* metacircular fixed point *of c iff*

$$\mathrm{Spec}^* \;=\; C^c\big(\mathrm{Spec}^*,\ \mathrm{accept}_c(\mathrm{Spec}^*)\big).$$

In words. For the absorptive C of the add-only case (Proposition 4.30), this says that every proposal that Spec*'s rules accept has already been absorbed into Spec*; no further proposal of the same content is distinct from Spec*.

The definition is the framework-level analogue of a *fixed point of self-evaluation* in compiler theory and reflective programming. In the add-only case, the agent operating under Spec* has no rule-acceptable proposal that is not already a consequence of Spec* itself.

Definition 4.26 (Tier progressivity, a condition)**.** *Take the tiers of specifications, and of joint states on each member, from the tiered presentation of the core signature (Definition 3.10), and let Φ_c be the self-amendment of a commons c (Definition 9.9). The commons satisfies* tier progressivity*, condition (Q), when tier $n+1$ of $\Phi_c(x)$ depends only on tiers $0,\dots,n$ of x: for every n and all x, y that agree at tiers $0,\dots,n$, $\Phi_c(x)$ and $\Phi_c(y)$ agree at tier $n+1$. A set $\mathcal{Q} \subseteq \mathbb{I}$ of commons with joint self-amendment $\Phi_{\mathcal{Q}} : \mathrm{Spec}^{\mathcal{Q}} \to \mathrm{Spec}^{\mathcal{Q}}$ (Definition 9.9), in which each member c takes from a set $P_c \subseteq \mathcal{Q}$ of peers, satisfies tier progressivity* jointly *when, for every member c, every n and all joint states x, y that agree at tiers $0,\dots,n$ on c and on every peer in P_c, $\Phi_{\mathcal{Q}}(x)_c$ and $\Phi_{\mathcal{Q}}(y)_c$ agree at tier $n+1$.*

In words. The amendment to a tier is decided from the settled tiers below it. Where the commons also satisfies the coevolution quorum (Definition 8.17), every accepted proposal meets its quorum; (Q) itself asks for no quorum. *Status.* The condition is pure tier dependence, as the Lean `MetacircTiers.QuorumTiers` is (the Lean name is kept); the quorum enters only through the coevolution quorum (Definition 8.17), which every proposition that uses it names separately.

Definition 4.27 (Tier sealing, a condition)**.** *Tiers of specifications, and of joint states on each member, are those of the tiered presentation of the core signature (Definition 3.10). For a*

commons c *and a map* $\Phi_c : \mathrm{Spec} \to \mathrm{Spec}$*, a* self-amendment *of* c*, the commons satisfies* tier sealing *for* Φ_c*, condition (MI), when tier* 0 *of* $\Phi_c(x)$ *is the same for every specification* x*. A set* $\mathcal{Q} \subseteq \mathbb{I}$ *of commons and a map* $\Phi_{\mathcal{Q}} : \mathrm{Spec}^{\mathcal{Q}} \to \mathrm{Spec}^{\mathcal{Q}}$*, a* joint self-amendment *of* $\mathcal{Q}$*, satisfy tier sealing* jointly *when, for every member* c*, tier* 0 *of* $\Phi_{\mathcal{Q}}(x)_c$ *is the same for every joint state* x*.*

In words. Tier 0 is meant to hold the* sealed floor*: every amended specification carries the one sealed floor, and no amendment rewrites it.

Status. The self-amendment is a parameter of the condition, not defined here. For the coevolution operator it is the self-application $\Phi_c(x) = C^c(x, \mathrm{accept}_c(x))$ (Definition 9.9), and Proposition 4.28 uses the condition for that map; stated for any map, the condition does not lie over the coevolution quorum. The clause is the one certified in Lean (`IasCert.Paper.MetacircTiers`, `causal_iff_sealed_and_lower`), where the specification space is the product $\prod_n E_n$ itself. Here the presentation is an injection into that product, not a bijection, so that Spec may be countable: a product of infinitely many sets with at least two elements is uncountable. A result proved on the product transfers to Spec only within the image of the injection; for a fixed point this is the proviso under which Proposition 4.28 is read. The label (MI) is kept from the level table, which groups this clause with the conditions of memory inviolability; the clause concerns specifications only, not agent-private memory (Definition 3.15).

Proposition 4.28 (Metacircular fixed point under tier sealing and tier progressivity)**.** *Fix a commons* c *and write* $C := C^c$*,* $\mathrm{accept} := \mathrm{accept}_c$ *and* $\Phi := \Phi_c$*,* $\Phi(x) = C(x, \mathrm{accept}(x))$*, for its coevolution operator, its acceptance and its* self-application *(Definition 9.9). Take the tiers of specifications from the tiered presentation of the core signature (Definition 3.10), an injective map* $\mathrm{Spec} \to \prod_n E_n$*, tier* 0 *first: tier* 0 *is the sealed floor, and deeper tiers are further from it. Assume*

(MI) tier sealing (Definition 4.27): tier 0 *of* $\Phi(x)$ *is the same for every* x*: tier* 0 *of every amended specification is the same, whatever it is amended from;*

(Q) tier progressivity (Definition 4.26): tier $n+1$ *of* $\Phi(x)$ *depends only on tiers* $0, \ldots, n$ *of* x*.*

Then Φ *has at most one fixed point, and from any starting specification* $n+1$ *rounds of self-amendment settle tiers* $0, \ldots, n$ *at values that do not depend on the start. When some specification has these settled values as its tiers, it is the fixed point* Spec^* *of* Φ*, a metacircular fixed point in the sense of Definition 4.25, even when amendments retract commitments; this holds in particular when the tiered presentation is onto, as for* $\mathrm{Spec} = \prod_n E_n$*. Since* $\Phi(\mathrm{Spec}^*) = \mathrm{Spec}^*$*, tier* 0 *of* Spec^* *is the floor that every amendment writes, and each tier* $n+1$ *is what an amendment writes there on any specification that agrees with* Spec^* *through tier* n*.*

Neither hypothesis can be dropped: Example 4.31.

Without retraction the tiers are not needed: Proposition 4.30.

The quorum at the fixed point of the add-only case is Proposition 4.29.

The self-application is essential: iterating $C(\cdot, \emptyset)$ *with a fixed empty proposal-set yields only a fixed point of* $C(\cdot, \emptyset)$*, which need not satisfy the self-referential equation of Definition 4.25.*

Proof. (MI) and (Q) together say that each tier of $\Phi(x)$ is determined by the tiers of x strictly below it (at tier 0 the condition is vacuous, which is (MI)). So if x and y first differ at tier n, then $\Phi(x)$ and $\Phi(y)$ agree at least through tier n: in the distance $d(x, y) = 2^{-\mathrm{firstDiff}(x,y)}$, Φ halves distances. On the product $\prod_n E_n$ the space is a complete ultrametric space, and Banach's theorem gives exactly one fixed point there, the standard fixed point of a strictly causal map on sequences [dBZ82]; for a presentation that is not onto, uniqueness, and existence under the proviso, are the argument in the *Status*. For the rounds, when the fixed point exists, argue by

induction on n: tier 0 of $\Phi(x)$ equals tier 0 of $\Phi(\text{Spec}^*) = \text{Spec}^*$ by (MI); if $\Phi^{n+1}(x)$ agrees with Spec^* through tier n, then by (Q) $\Phi^{n+2}(x)$ agrees with $\Phi(\text{Spec}^*) = \text{Spec}^*$ through tier $n+1$. For the examples, on sequences of bits: $\Phi(x)_0 = \neg x_0$ with all deeper tiers 0 satisfies (Q) and has no fixed point; $\Phi(x)_0 = 0$ and $\Phi(x)_n = \neg x_n$ for $n \geq 1$ satisfies (MI) and has none. The add-only case is the Knaster–Tarski theorem for a monotone map on a complete lattice; for the Kleene form, the chain $\bot \sqsubseteq \Phi(\bot) \sqsubseteq \Phi^2(\bot) \sqsubseteq \cdots$ has a supremum, which Scott continuity makes a fixed point and which lies below every fixed point by induction on n; on the subsets of a finite set U a strictly ascending chain has at most $|U|$ steps (full proof in Appendix C.2). □

Status. The form of (Q) across commons is that of Proposition 4.33. The main statement is proved in Lean (`IasCert.Paper.MetacircTiers`): `metacirc_under_MI_Q` (exactly one fixed point on the product $\prod_n E_n$, and tier n settled after $n+1$ rounds), with the two hypotheses by name (`MemInviol`, `QuorumTiers`; the Lean name `MemInviol` is tier sealing, not memory inviolability of agent-private memory); the bare uniqueness and the tier read-out of Spec* *(`IasCert.Paper.MetacircFloor`: `unique_fixedpoint_under_MI_Q`, `floor_of_fixedpoint`, `tier_of_fixedpoint`); the examples (`flipFloor_no_fixedpoint`, `flipSelf_no_fixedpoint`, `fix_tier0_not_attenuating`, and in `MetacircFloor` `fix_tier0_rounds_do_not_settle`, `keepFloor_not_unique`); and the equivalence with tier attenuation (`attenuates_iff_causal`), whose Banach core is `IasCert.Paper.MetacircBanach`. The add-only case is proved in Lean on a complete lattice (`IasCert.Paper.MetacircMonotone`, Knaster–Tarski) and in full on the subsets of a finite U (`IasCert.Paper.MetacircKleene`); the Kleene form, which Appendix C.2 proves on a pointed directed-complete partial order with Φ Scott-continuous, a complete lattice included, is a text proof. The certificates of this proposition are only the theorems that prove its statement (`IasCert.Paper.MetacircTiers` and `IasCert.Paper.MetacircFloor`): the add-only case, the Banach core (whose hypothesis forces Φ to be injective, `hprog_forces_injective`, a hypothesis stronger than (MI) and (Q) and not used by the statement) and the examples are proved in the modules named above but are not part of the statement. The quorum at the least fixed point of the add-only case is Proposition 4.29. The Lean results are on the product $\prod_n E_n$. For a tiered presentation that is not onto, the transfer to* Spec *is a text argument: two fixed points that first differ at tier n are sent by Φ to specifications that agree through tier n, so there is at most one; the rounds settle as on the product; and a specification x whose tiers are the settled values is fixed, since tier n of $\Phi(x)$ is read from tiers $0, \ldots, n-1$ of x, which agree with those of Φ^n of the start, so that $\Phi(x)$ and x have the same tiers and are equal by injectivity. Whether such a specification exists depends on the image. Reading tier 0 as the sealed floor, and the dependence of a tier on the tiers below it as the order of amendment, is a modelling step, not a theorem: it is the reading under which the tier argument applies. The Lean shows that the two clauses together are equivalent to tier attenuation and that dropping either can lose the fixed point; it does not show that they are necessary for one.*

Proposition 4.29 (Quorum at the metacircular fixed point)**.** *Fix a commons c and a time, and write $C := C^c$,* accept $:=$ accept$_c$ *and $\Phi := \Phi_c$ for its coevolution operator, its acceptance and its self-amendment (Definition 9.9). Assume that* Spec*, ordered by a relation $\sqsubseteq$, is a complete lattice, that Φ is monotone, and write* Spec* *for its least fixed point. Assume that c satisfies the coevolution quorum (Definition 8.17). Then every proposal accepted under* Spec* *meets c's quorum, and a proposal that does not meet it is not accepted under* Spec*. If, moreover,* Spec *is the set of subsets of a set U of rules, ordered by inclusion, each proposal p carries a content* content$(p) \subseteq U$, accept *is monotone (a proposal accepted under x stays accepted under every $y \supseteq x$), and C is absorptive, $C(x, S) = x \cup \bigcup_{p \in S}$ content(p), then* Spec* *is exactly the union of the contents of the proposals accepted under it, and every commitment in* Spec* *lies in the content of a proposal that is accepted under* Spec* *and meets c's quorum.*

Proof. Spec* is a specification, so a proposal accepted under it meets the quorum by the

coevolution quorum, and a proposal that does not meet it is therefore not accepted. For the absorptive C, the union A of the contents of the proposals accepted under Spec^* lies in $\Phi(\mathrm{Spec}^*) = \mathrm{Spec}^*$; monotonicity of accept gives $\Phi(A) \subseteq A$, and so $\mathrm{Spec}^* \subseteq A$ by leastness. Each commitment of Spec^* therefore lies in the content of a proposal accepted under Spec^*, which meets the quorum by the first sentence. □

Status. Shallow: the first sentence is ∀-instantiation at the fixed point. The first sentence is proved in Lean for an arbitrary monotone map on a complete lattice and an arbitrary acceptance (`IasCert.Paper.CoevQuorumCondition`: `accepted_at_fp_has_quorum`, `subquorum_cannot_move_fp`, and `coevQuorum_sufficient_for_governed_fp` with the stability of the fixed point, `fp_stable`); it uses neither the monotonicity of Φ nor the tie between Φ and accept, and it holds at every specification, not only at Spec^*. The second sentence is proved in Lean for the absorptive C on the subsets of any set U (`IasCert.Paper.MetacircTiers`: `lfp_phiAbsorb_eq`, `lfp_commitment_has_quorum`). In both, meeting the quorum is a count against a bound, $\mathrm{quorum}(p) \geq \mathrm{quorumBound}$; reading it as k_c distinct attesters of W_c with σ-events of c at or before the fixed time (Definition 8.17) is a modelling step. The add-only case in which Spec^* exists is Proposition 4.30. The proposition concerns that least fixed point only, not the fixed point of the main statement of Proposition 4.28, under tier sealing and tier progressivity, where amendments may retract commitments.

Proposition 4.30 (The add-only case of the metacircular fixed point)**.** *Without retraction the tiers are not needed. (i) Let* $(\mathrm{Spec}, \sqsubseteq)$ *be a complete lattice, with* $\sqsubseteq$ *read as "is refined by", and let* Φ *be monotone. Then* Φ *has a least fixed point* Spec^**, a metacircular fixed point in the sense of Definition 4.25, and* $\mathrm{Spec}^* \sqsubseteq x$ *whenever* $\Phi(x) \sqsubseteq x$*. (ii) Let the specifications be the subsets of a finite set* U *of rules, and let* $\Phi(x) = C(x, \mathrm{accept}(x))$ *with* C *absorptive,* $C(x, S) = x \cup S$ *for proposals read as the rules they add, so that* $\mathrm{accept}(x) \subseteq U$*, and* accept *monotone (Definition 9.9). Then the iterates* $\Phi^n(\emptyset)$ *reach the least fixed point within* $|U|$ *rounds: absorb the acceptable proposals until nothing new is accepted.*

Status. Certified: (i) is Knaster–Tarski (`IasCert.Paper.MetacircMonotone`), and (ii) is `IasCert.Paper.MetacircKleene`. Φ is monotone when C is monotone in the specification and in the set of proposals and accept is monotone; that C never removes a commitment, $x \sqsubseteq C(x, S)$, does not by itself make Φ monotone. The Kleene form, $\mathrm{Spec}^* = \bigsqcup_{n \geq 0} \Phi^n(\bot)$ for Scott-continuous Φ on a pointed directed-complete partial order, a complete lattice included, is the text proof of Appendix C.2, not a Lean result. This is the reconciliation loop "absorb until nothing new"; the quorum at its fixed point is Proposition 4.29.

Example 4.31 (Neither hypothesis of the metacircular fixed point can be dropped)**.** *If (Q) holds but an amendment may flip the floor, there need be no fixed point at all; if (MI) holds but tier* 1 *is amended by reading tier* 1 *itself, again there need be none. Nor does it suffice that* Φ *leave tier* 0 *of each specification unchanged: from two specifications that differ at tier* 0 *the rounds never settle it, and the fixed point need not be unique.*

Status. The examples are machine-checked (`IasCert.Paper.MetacircTiers`: `flipFloor_Q`, `flipFloor_not_MI`, `flipFloor_no_fixedpoint`, `flipSelf_MI`, `flipSelf_not_Q`, `flipSelf_no_fixedpoint`, `fix_tier0_not_attenuating`, `const_causal_not_hprog`; `IasCert.Paper.MetacircFloor`: `fix_tier0_rounds_do_not_settle`, `keepFloor_not_unique`).

Definition 4.32 (Delivery condition, a condition)**.** *A federation, a declared peer-set* $\mathcal{Q} \subseteq \mathbb{I}$*, satisfies* the delivery condition *with bound* Δ_{fed} *for a sync epoch cut at time* t_0 *(Definition 8.6) when every offer of the epoch reaches its peer within* Δ_{fed}*: for every two commons* $k, k' \in \mathcal{Q}$*, the offer* $\mathrm{offer}(k, k')$ *of* k *arrives at* k' *by time* $t_0 + \Delta_{\mathrm{fed}}$*,* $\mathrm{arr}(k, k') \leq t_0 + \Delta_{\mathrm{fed}}$.

Status. The arrivals are carried over the journeys of the communications substrate (Definition 8.5).

Proposition 4.33 (Coevolving commons)**.** *Each commons has its own specification, and under (MI) and (Q), tier sealing and tier progressivity (Definition 4.27, Definition 4.26), at most one metacircular fixed point of its own, and exactly one under the proviso of Proposition 4.28. Take a set of commons, each amending its own specification by reading the others', and each taking from some peers only, on its own conditions, with the joint self-amendment Φ_Q of Definition 9.9. Suppose every member keeps its floor, tier 0 of its amended specification being the same, whatever joint state it is amended from (MI, tier sealing jointly, Definition 4.27), and amends tier $n+1$ only from tiers $0, \dots, n$ of itself and of the peers it takes from (Q, tier progressivity jointly, Definition 4.26). Then the joint self-amendment of the set has at most one fixed point, and from any starting state $n+1$ rounds settle tiers $0, \dots, n$ of every member at values that do not depend on the start; when some joint state has these values as its tiers, as when the tiered presentation is onto, it is the one joint fixed point, and the set coevolves (Definition 9.9). The conclusion does not depend on who takes from whom: it holds for every choice of peers, isolation included. For a member that takes from no peer, its part of the joint fixed point, when there is one, is the unique fixed point of its own self-amendment, the others held fixed. A single member whose amended tier 0 differs from its current tier 0 at every joint state leaves the set with no joint fixed point.*

Proof. Stack the members' tier-n contents into one joint tier n. By (MI) for every member, the joint tier 0 of the amended state is the same for every state; by (Q) for every member, the joint tier $n+1$ depends only on joint tiers $0, \dots, n$, since reading only some peers is a special case of reading all. So after $n+1$ rounds, tiers $0, \dots, n$ no longer depend on the starting state (induction on n). The state whose tier n is its value after $n+1$ rounds is therefore, when there is such a state, a fixed point, by injectivity as in the *Status* of Proposition 4.28, and any two fixed points agree tier by tier, by strong induction. If a member rewrites its floor at every joint state, no joint state is fixed at that member's tier 0. □

Status. Nothing imposes one specification across commons, and a commons whose amendments break (MI) or (Q) may have no fixed point at all, as the examples of Proposition 4.28 show for a single commons; that nothing imposes one specification across commons is not certified. The Lean module `IasCert.Paper.MetacircCoevolve` *states these results (*`coevolve`*,* `coevolve_unique`*,* `joint_causal`*,* `isolated_member_own_fp`*,* `bad_member_no_joint_fp`*, and* `exists_unique_of_causal`*, the tier argument without a metric); they are machine-checked, with only Lean's standard axioms (Appendix D, row Paper.MetacircCoevolve). The Lean results are on products of tiers; for a tiered presentation that is not onto, the transfer is the text argument in the Status of Proposition 4.28, applied to the joint tiers. That, for an isolated commons, the member's own self-amendment with the others held fixed is the self-amendment Φ_c of Proposition 4.28, so that the joint reading agrees with it, is uncertified: it holds when $\mathrm{accept}_c^{Q}(x) = \mathrm{accept}_c(x_c)$ for every joint state x whenever c takes from no peer, which the Lean does not state. In the model all members amend in the same synchronous rounds. Reading a round as a sync epoch that meets the delivery condition (Definition 4.32) of Proposition 8.8 is an interpretation, not a hypothesis of the proof. The proposition says when a set coevolves, not which commons join one: that is each commons' decision.*

Lemma 4.34 (Meta-circular core non-empty and fixed by C; text proof)**.** *Suppose (H1) the enforcement layer has at least one rule, and each of its rules is paired with the specification it enforces, and (H2) the core is a function of the specification, $\mathfrak{M} = \mathfrak{M}(x)$. Then the meta-circular core $\mathfrak{M} \subseteq V(\mathrm{SPIG})$ of Definition 9.11 is non-empty, and, taken at the metacircular fixed point Spec^* of the self-application $\Phi(x) = C(x, \mathrm{accept}(x))$ of Proposition 4.28 (under tier sealing and tier progressivity the only one, when it exists; the least one in the add-only case), does not change under a further step: $\mathfrak{M}(\Phi(\mathrm{Spec}^*)) = \mathfrak{M}(\mathrm{Spec}^*)$, where $\mathfrak{M}(x)$ is the set of nodes whose*

declare-enforce cycle holds under specification x.

Proof. The content is in (H1) and (H2); given them, the lemma is short. Non-emptiness is (H1): each of its rules is paired with the specification it enforces (the memory-isolation rule with Proposition 3.19, for instance), and each such pair is a declare-enforce 2-cycle, so the layer contributes at least one node to the core. Invariance: the core is a function of the specification, $\mathfrak{M} = \mathfrak{M}(\mathrm{Spec}^*)$, and $\Phi(\mathrm{Spec}^*) = \mathrm{Spec}^*$ by Proposition 4.28, so the core computed after a further step is the same set. □

Status. Text proof; not machine-checked.

4.4 The repair pump

Definition 4.35 (Entity-admission predicate)**.** *For a commons c (Definitions 3.10 and 3.14), let U_c be the set of* candidate units *presented to the commons boundary — σ-events offered to the chain, commits offered to the per-commons factor $\mathbb{S}^c$, actions of digital agents offered for execution, and realisations of the identityless substrate (Definition 4.16) offered for binding to a canonical label. The* entity-admission predicate *of c is a declared partial predicate $\phi_c : U_c \rightharpoonup \{0,1\}$: $\phi_c(u) = 1$ admits the unit as self, and $\phi_c(u) = 0$ rejects it as non-self. Where ϕ_c is undefined for some unit class, the commons boundary is* degraded *at that class.*

Status. Under σ-journaling (Definition 3.21), every ϕ_c-evaluation that is an action of a digital agent in the σ-vocabulary and has an observable effect is σ-attested; σ-journaling is mentioned here and is not a dependency of the definition.

Definition 4.36 (Damage-count chain)**.** *For $b, d \geq 0$ with $b + d \leq 1$ and a carrier size $M \geq 1$, the* damage-count chain *of carrier size M is the Markov chain on $\{0, \ldots, M\}$ that, in one step, moves from k to $k+1$ with probability b when $k < M$, to $k-1$ with probability d when $k > 0$, and otherwise stays at k. The count 0 is the* healthy *count; for a stationary law π of the chain, π_0 is its mass on the healthy count.*

Status. This is the birth–death chain of Proposition 4.45(iii) and the chain `Pn M b d` of Lean `Paper.BirthDeathStationary`. The numbers b and d are per-step probabilities, not rates. The definition stands on its own so that the damage and continuation conditions, both at level 1, can cite one family without either citing the other.

Definition 4.37 (Damage condition, a condition)**.** *A commons satisfies* the damage condition *with damage probability b and repair probability d, where $b, d \geq 0$ and $b + d \leq 1$, when four clauses hold for its controlled substrate chain $X_0, X_1, \ldots$ on $\mathcal{X}$ (Definition 4.15):*

(i) damage count — there is a map $D : \mathcal{X} \to \{0, \ldots, M_c\}$, $M_c \geq 1$, the damage count*; the states x with $D(x) = 0$ are called* healthy*;*

(ii) lumpability — the count process $D(X_0), D(X_1), \ldots$ is the damage-count chain of carrier size M_c with these b and d (Definition 4.36), in the strong sense that for every step n, conditionally on any history of the chain up to n with $D(X_n) = k$, the next count $D(X_{n+1})$ is $k+1$ with probability b when $k < M_c$, is $k-1$ with probability d when $k > 0$, and is k otherwise;

(iii) repair — every step at which the count decreases is a repair: its effect on the coverage graph is one step of the repair relation $\to_{\mathrm{rep}}$ (Definition 4.41);

(iv) damage arrives — $b > 0$.

Status. Clause (ii) is the strong-lumpability hypothesis of Proposition 4.45(iii), named here as a clause of the condition. That proposition obtains the same count chain from lumpability together with a stationary policy (Definition 4.5); this condition asks for the per-step law of the

count directly, at every step, so the stationary-policy condition is not one of its clauses and is not implied by it. Clause (iii) is the clause that ties the damage count to the repair cost (H3), through (H4), of Definition 3.6. The continuation condition (Definition 4.38) quantifies over the damage-count chains of every carrier size at fixed b and d, as `ContinuesStat` *does in Lean. The healthy states here are those of damage count* 0*; the repair-normal states of Proposition 4.45(i), and of the uncertified part (ii) in its Status, are the normal forms of the repair relation, and this condition does not require that the two coincide.*

Definition 4.38 (Continuation condition, a condition)**.** *A pair (b, d) of a damage probability b and a repair probability d, where $b, d \geq 0$ and $b + d \leq 1$, satisfies* the continuation condition *when every stationary law of the damage-count chains with these b and d (Definition 4.36) keeps the healthy fraction bounded away from zero however large the carrier: there is an $\varepsilon > 0$ such that, for every carrier size $M \geq 1$, every stationary law π of the damage-count chain of carrier size M puts mass $\pi_0 \geq \varepsilon$ on the healthy count* 0*.*

Status. The condition is a predicate on the pair (b, d) alone; a commons satisfies it when the pair of its damage condition does, and Proposition 4.39 assumes this condition with the probabilities of the damage condition (Definition 4.37). The two conditions cite one family and neither lies over the other. The condition is stated over the chains' stationary laws, not through the closed form of π_0, as `ContinuesStat` in Lean `Paper.BoundaryMaintenanceRepair`; that Lean form also admits $M = 0$, where the chain has the single state 0 and $\pi_0 = 1$, so the two forms agree. For these chains and $b > 0$ the condition is equivalent to $d > b$: one direction is `repair_needed`, the other the bound $\pi_0 \geq 1 - b/d$ with the uniqueness of the stationary law (Lean `Paper.BirthDeathStationary`). The first conclusion of Proposition 4.39, $0 < d$ and $b < d$, is the forward direction.

Proposition 4.39 (Boundary maintenance under damage, continuation and the Landauer bridge)**.** *Let a commons c satisfy, with one pair of probabilities $b, d \geq 0$, $b + d \leq 1$, the damage condition (Definition 4.37) with damage probability b and repair probability d, whose clause (iv) is that damage arrives, $b > 0$, and the Landauer bridge (Definition 3.6) with repair cost $c_{\mathrm{rep}} > 0$; and let the pair (b, d) satisfy the continuation condition (Definition 4.38), that every stationary law of the damage-count chains (Definition 4.36) with these b and d keeps the healthy fraction bounded away from zero however large the carrier. Then repair happens and outpaces damage, $0 < d$ and $b < d$, and $0 < b\, c_{\mathrm{rep}} < d\, c_{\mathrm{rep}}$.*

Proof. With $b > 0$ and $d = 0$, every stationary law of the damage-count chain of carrier size $M \geq 1$ puts no mass on the healthy count (`no_repair_absorbs`, Lean `Paper.BoundaryMaintenanceRepair`), and such a law exists (`no_repair_stationary`), which continuation excludes; so $d > 0$. With $0 < d \leq b$ the mass π_0 of the stationary law tends to zero as the carrier grows (`pi_zero_vanishes`, Lean `Paper.BirthDeathStationary`, the threshold of Proposition 4.45(iii) for the damage-count chain itself, with none of that proposition's hypotheses on the full chain), which continuation also excludes; so $b < d$. Both steps are `repair_needed`, from $b > 0$, $d \geq 0$ and continuation over the stationary laws (`ContinuesStat`). With $c_{\mathrm{rep}} > 0$, the inequality $0 < b\, c_{\mathrm{rep}} < d\, c_{\mathrm{rep}}$ is `maintenance_work_pos` (Lean `Paper.BoundaryMaintenanceWork`), whose hypothesis `Continues` follows from $0 < b < d$ by `continues_of_outpaces`. □

Status. Shallow: the conclusion follows from $b > 0$ (clause (iv) of the damage condition), the continuation condition and $c_{\mathrm{rep}} > 0$, by the birth–death lemma `repair_needed` and arithmetic; for these chains and $b > 0$ the continuation condition is equivalent to $d > b$ (Definition 4.38). Clauses (i)–(iii) of the damage condition and (E), (H0)–(H2) and (H4) of the Landauer bridge are not used; the two conditions are cited for their vocabulary, the probabilities b, d of the commons and the repair cost c_{rep}. In Lean, `ContinuesStat` bounds the healthy mass of every stationary

law (`StatLaw`: stationary, normalised and non-negative) of the chain over the certificate's field; over $\mathbb{R}$ it is the continuation condition, the Lean form also admitting $M = 0$, where the two agree (Definition 4.38). `repair_needed` and `maintenance_work_pos` hold over any Archimedean ordered field, $\mathbb{R}$ included, so b, d and c_{rep} are real, as the damage condition and Definition 3.6 have them. By (H4) of the bridge every repair step that a run of the chain takes, among them every step at which the damage count decreases (clause (iii) of the damage condition), is realised by an operation in its domain, and by (H3) it costs at least c_{rep}; so c_{rep} is tied to realised repair. The proof uses c_{rep} as a number only. The three conditions are named, not derived: the Landauer bridge is not Axiom-0, whose legs contain no temperature. Of the Landauer bridge only the number c_{rep} of clause (H3) is used. The certificates are `BoundaryMaintenanceRepair` (`repair_needed`) and `BoundaryMaintenanceWork` (`maintenance_work_pos`); they prove the statements about b, d and c_{rep} as numbers. Read with clauses (ii)–(iii) of the damage condition and (H3)–(H4), a step taken from a damaged state is a repair with probability d, and each repair costs at least c_{rep}, so $d\,c_{\mathrm{rep}}$ bounds below the expected cost of repair in such a step; this reading is not certified. It is not a bound on the stationary average per step: no repair happens at count 0, so at stationarity the expected cost of repair per step is bounded below by $c_{\mathrm{rep}}\,d\,(1-\pi_0)$, which by flux balance equals $c_{\mathrm{rep}}\,b\,(1-\pi_{M_c})$ and is at most $b\,c_{\mathrm{rep}}$; this too is not certified. That persistence of the commons boundary, the continued well-definedness of the entity-admission predicate ϕ_c (Definition 4.35) on incoming candidate units, requires continuous maintenance work $W_c > 0$ from sources outside its canonical state (operator attention, vendor compute cycles, kernel-rule enforcement cycles) is the reading of this result, not part of it; that without it ϕ_c degrades (chain fragmentation, identity erosion, convention decay) is not a further theorem. The skeletal form, that a born commons stays non-empty while active, is `Paper.AblationDemotion`, which proves a different statement and is not a certificate of this one.

Definition 4.40 (The Φ-split)**.** *The measure attached to the* repair rewrite relation $\to_{\mathrm{rep}}$ *(Definition 4.41) is the integer coverage-potential* Φ_{int} *(Definition 4.22). The* canonical *representative of a state is the least of the* $\to_{\mathrm{rep}}$*-normal forms reachable from it (the states reachable by* $\to_{\mathrm{rep}}$ *from which no further step applies) in the order that compares first by* Φ_{MDL}*, the Minimum Description Length (MDL) potential of Definition 4.44, and breaks ties by a fixed total order* key *on states; it is defined when that set is non-empty and otherwise undefined.*

In words. The two potentials have disjoint jobs: Φ_{int} *measures the cleanup moves on the coverage graph* $\mathfrak{G}$*, and* Φ_{MDL} *selects among their results. Canonicalization is not a rewrite: the selection is a function outside* $\to_{\mathrm{rep}}$ *(`Ias.Phi.phiMDLargmin`).*

Status. That Φ_{int} *is a ranking function for* $\to_{\mathrm{rep}}$*, that is, that every step strictly lowers it, is not part of this definition: it is Proposition 4.42. Given that proposition, the real-valued* Φ_{MDL} *never enters the termination or confluence argument.*

Definition 4.41 (Repair rules as double-pushout (DPO) productions)**.** *Over the coverage graph's typed-attributed structure (Definition 4.21;* `Emits` $\sqsubseteq$ `Produces`*, attribute* `checked_at`*), the* repair system $\to_{\mathrm{rep}}$ *is three productions: (1)* Death *— delete a vertex, with all its incident edges, under the global side condition, checked on the whole graph outside the DPO formalism, that the vertex is unreachable from every obligation-node root; (2)* Merge *— a DPO production: where a node carries both a* `Produces` *and an* `Emits` *edge to the same target, delete the* `Produces` *edge; (3)* Dedup *— a DPO production: of two parallel edges equal under the key* `(from, to, type, checked_at)`*, delete one.*

In words. Death's reachability test is not a negative application condition (NAC) in the DPO sense, which forbids a fixed finite pattern around the match, and it is not first-order; that is why it is a global side condition. Merge is subtype-subsumption, not node-identification: the `Emits` *edge already entails the* `Produces` *edge by inheritance, so the* `Produces` *edge is redundant; node*

identity is load-bearing, and two obligation-nodes with equal required-context are distinct and are never identified. Dedup removes exact duplicates.

Status. Each production strictly lowers the integer coverage-potential Φ_{int} *(Definition 4.22); that is Proposition 4.42, not this definition. Garbage collection (GC) is not a fourth rule: it is* `Death` *iterated to a fixpoint (a dead component is collectively root-unreachable).*

Implementation. `Ias.Repair`.

Proposition 4.42 (Strong normalisation of the repair rewrite)**.** *The repair rewrite* $\to_{\text{rep}}$ *is strongly normalising: every repair sequence from a state* $\mathfrak{s}_0$ *reaches a* $\to_{\text{rep}}$*-normal form in at most* $\Phi_{\text{int}}(\mathfrak{s}_0)$ *steps.*

Proof. Each of the three repair moves, merge, deduplicate and death, strictly lowers the natural-number ranking Φ_{int} of Definition 4.22 (`merge_strict`, `dedup_strict`, `death_strict`, `repair_strict`). A strictly decreasing map into $\mathbb{N}$ admits no infinite descent, so the rewrite is well founded (`repair_wf`), and a sequence of n steps satisfies $n \le \Phi_{\text{int}}(\mathfrak{s}_0)$ (`repair_step_bound`). These are proved in Lean `Paper.PhiIntSN`, on the standard axioms only, over a simplified model: a state is its set of edges, its vertices are those the edges touch, its content is held fixed, and a repair is any step to a proper subset of the edges. The death of an isolated vertex and Death's global reachability side condition are not in that model; for them the argument above is the proof. □

Status. The certificate covers the simplified model of the proof: a state is its set of edges, its vertices are the endpoints of those edges, `death_strict` needs an edge incident to the deleted vertex, and the content term of Φ_{int} is held fixed. The death of an isolated vertex, including a region vertex, whose deletion also lowers the content term, and Death's global reachability side condition lie outside it; for them the bound rests on the proof above and is not certified. The repair relation is defined over the coverage graph's typed-attributed structure (Definition 4.41); that every intermediate state of a repair sequence again satisfies the constraints of Definition 4.21 is not claimed.

This is the termination half of the clean-up. With local confluence, which Proposition 4.43 takes as its hypothesis, Newman's lemma gives confluence, and termination feeds the finite-state convergence of Proposition 4.45.

Edits sit outside the repair relation. A new file or an edit raises Φ_{int}: it is input from the environment, not a repair move, since repair only descends. A change that leaves Φ_{int} unchanged but makes the state more canonical, such as repointing a `Hydrates` edge from a built artifact to its source, is modelled as two edits from the environment, the deletion of the bad edge and the addition of the good one, each updating $\mathrm{Hyd}(n)$; neither is a repair move, since no repair rule deletes an arbitrary edge. So $\to_{\text{rep}}$ stays strictly descending in whole numbers. A lexicographic order on $(\Phi_{\text{int}}, \Phi_{\text{MDL}})$ would also be sound, but it would bring Φ_{MDL} into the proof.

Proposition 4.43 (Repair confluence under local confluence)**.** *Suppose that the repair relation* $\to_{\text{rep}}$ *(Definition 4.41) is locally confluent. Then, with its strong normalisation (Proposition 4.42), it is confluent.*

Proof. The rewrite is strongly normalising (Proposition 4.42). Newman's lemma, strong normalisation with local confluence implies confluence, is proved in Lean for any relation (`newman`, Lean `Paper.Newman`). □

Status. Local confluence is a hypothesis and is not yet checked. The intended evidence is a critical-pair analysis: flatten the single inheritance `Emits` ⊑ `Produces` to two concrete edge types and check that the critical pairs {Death × Death, Merge × Merge, Dedup × Dedup, Death ×

Merge, Death × Dedup, Merge × Dedup} are strictly joinable, the `Dedup` overlaps with attributes, since the deduplication key reads `checked_at`, in a graph-rewriting tool such as the Attributed Graph Grammar system (AGG) or Henshin; that analysis is owed. That joinable critical pairs give local confluence is the critical-pair lemma of double-pushout (DPO) rewriting, which is not certified here, and it does not apply to `Death` as it stands, whose side condition is checked on the whole graph outside the DPO formalism (Definition 4.41). With confluence and strong normalisation every state has a unique $\to_{\mathrm{rep}}$-normal form; this step is not certified. The reading "unique up to garbage", modulo dead components collected by `Death`, is not certified either. The canonical representative of a state is that of Definition 4.40, the least of its normal forms in the order that compares first by Φ_{MDL} and then by key. The strong normalisation used is that of Proposition 4.42, certified on that proposition's edge-set model only: a state is its set of edges, and the death of an isolated vertex and Death's global reachability side condition are not modelled. So `newman` applies with a certificate to the repair relation of that model; for $\to_{\mathrm{rep}}$ itself the strong normalisation rests on the text proof of Proposition 4.42.

Definition 4.44 (Canonicality potential)**.** *For a state $x \in \mathbb{X}$ (Definition 3.3) presented by its coverage graph $\mathfrak{G}$ (Definition 4.21), let $E_k = \tau^{-1}(k)$ be the edges of type k, types-present$(\mathfrak{G}) = \{k : E_k \neq \emptyset\}$, src_k and tgt_k the carriers of the endpoint sorts of k, $\ell_{\mathbb{N}}$ a fixed universal code length for positive integers, and* blob $: R \to \mathbb{N}$ *the content size of each region in bytes. The* canonicality potential *is $\Phi_{\mathrm{MDL}} : \mathbb{X} \to \mathbb{R}_{\geq 0}$, $\Phi_{\mathrm{MDL}} = L(\mathit{structure}) + L(\mathit{content} \mid \mathit{structure})$, where $L(\mathit{structure}) = \sum_{k \in \mathit{types\text{-}present}(\mathfrak{G})} \big(\ell_{\mathbb{N}}(|E_k|) + |E_k| \cdot \log_2(|\mathrm{src}_k| \cdot |\mathrm{tgt}_k|)\big)$ and $L(\mathit{content} \mid \mathit{structure}) = \sum_r 8\,\mathrm{blob}(r)$ bits.*

In words. Φ_{MDL} is a two-part minimum-description-length code (the two-part code is Rissanen's, and $\ell_{\mathbb{N}}$ is the code length of Rissanen's universal prior for integers [Ris83]; per-edge-type after Navlakha, Rastogi and Shrivastava [NRS08], and its knowledge-graph variant, KG-MDL): the structure part sums a universal integer code for each present edge-type's count plus a uniform per-edge index.

Status. The MDL codebook super-nodes live only in the encoding — they are never vertices of $\mathfrak{G}$, so they neither inflate $|V|, |E|$ nor collide with Φ_{int} (Definition 4.22).

Implementation. `Ias.Phi.phiMDL`.

The pump is a rate condition, not a metaphor; we make it quantitative on the finite state space, where repair and damage compete and health is the stationary mass of the healthy set, the states of damage count 0, which need not be the normal forms of the repair relation.

Proposition 4.45 (Finite-state convergence of the repair chain)**.** *For the controlled substrate chain (Definition 4.15) on its state space $\mathcal{X}$, finite by leg (A0.0) of Axiom 3.1, and the repair relation $\to_{\mathrm{rep}}$ (Definition 4.41):*

(i) [transient] every sequence of $\to_{\mathrm{rep}}$-steps from a state $\mathfrak{s}_0$ reaches a normal form of the repair relation, which we call repair-normal*, in at most $\Phi_{\mathrm{int}}(\mathfrak{s}_0)$ steps (Proposition 4.42);*

(iii) [birth–death case] suppose that the controls follow a stationary policy (Definition 4.5), that every vendor whose model the chain calls satisfies vendor statelessness (Definition 4.4), and that the chain is strongly lumpable *along a map $D : \mathcal{X} \to \{0, \ldots, M\}$, $M \geq 1$, the damage count: from every state x with $D(x) = k$, one step moves to a state of count $k+1$ with probability b when $k < M$ and to a state of count $k-1$ with probability d when $k > 0$, and otherwise the count stays put, with $b, d > 0$ and $b + d \leq 1$. Then the count is itself a Markov chain, the damage-count chain of carrier size M (Definition 4.36), and its stationary law is $\pi_k = \pi_0\, r^k$ with $r = b/d$, so*

the stationary mass of the healthy count 0 *is*

$$\pi_0 \;=\; \frac{1-r}{1-r^{M+1}} \quad (r \neq 1), \qquad \pi_0 \;=\; \frac{1}{M+1} \quad (r = 1).$$

Hence π_0 *is bounded away from zero uniformly in the carrier size* M *if and only if repair strictly outpaces damage,* $d > b$*, and then* $\pi_0 \geq 1 - b/d$*. At* $d = b$ *it is* $1/(M+1)$*, and for* $d < b$ *it tends to zero as the carrier grows, so in both cases it tends to zero.* □

In words: with edits paused, repair always finishes, and within a number of steps known in advance. With edits ongoing, under the uncertified part (ii) of the Status, the substrate is repair-normal, with no repair left to apply, for a definite fraction of the time; in the birth–death case the damage count is 0 *for a fraction of the time that stays positive however large the carrier exactly when repairs outrun damage. Repair-normal is a statement about the repair relation only; that a repair-normal state is a model of SOMA is a further claim, not proved here. For instance, if repairs come twice as often as damage (*$d = 2b$*), the self is healthy at least half the time, whatever the size of its store; if they come equally often, with a damage count ranging over a thousand values, the count is* 0 *about one step in a thousand.*

Proof. Part (i) is Proposition 4.42, since Φ_{int} is a strictly decreasing $\mathbb{N}$-valued ranking of $\to_{\text{rep}}$. Part (ii), stated in the Status, is the standard theory of finite Markov chains [KS60]: a finite stochastic matrix has a stationary vector, which is unique exactly when there is one recurrent class; the ergodic theorem gives the long-run fraction, and aperiodicity gives convergence. For part (iii), the two hypotheses make the damage count a function of the chain that is itself a Markov chain (strong lumpability [KS60]), and that chain is a birth–death chain, hence reversible, and detailed balance $\pi_k\, b = \pi_{k+1}\, d$ gives $\pi_{k+1} = r\,\pi_k$; normalising the geometric series gives π_0, the standard truncated geometric law of a finite birth–death chain, of the same form as the M/M/1/K occupancy law [Kle75]. For $r < 1$, $1 - r^{M+1} \leq 1$ gives $\pi_0 \geq 1 - r$. For $r = 1$, $\pi_0 = 1/(M+1)$. For $r > 1$, $\pi_0 = (r-1)/(r^{M+1} - 1) \to 0$. The chain is irreducible because $b, d > 0$, and aperiodic because it idles at 0 with probability $1 - b > 0$. □

This is a quantitative companion to the absorbing-satisfaction-set reading of Proposition 4.3, about the healthy states rather than Sat*. Status: part (i) rests on Proposition 4.42, proved in Lean on that proposition's edge-set model; part (ii), uncertified and so not part of the statement, reads: suppose that the controls follow a stationary policy and that every vendor whose model the chain calls satisfies vendor statelessness; then, with edits ongoing, the finite chain has at least one stationary law* π*; if it has exactly one recurrent class,* π *is unique and the long-run fraction of time spent repair-normal is* $\pi(H)$*, where* H *is the set of repair-normal states of (i); if that class is also aperiodic, the law of the chain converges to* π *from any start. It is the standard theory of finite Markov chains, cited, not certified; part (iii) is proved in Lean for the birth–death chain itself in* `IasCert.Paper.BirthDeathStationary` *(stochasticity, detailed balance and stationarity, uniqueness of the stationary law, the closed form of* π_0*, and both directions of the threshold), on Lean's standard axioms only, and the reduction from the full chain in* `IasCert.Paper.StrongLumpability`*: under strong lumpability the damage count is a Markov chain with the block kernel from every state of a block, and the image of any stationary law of the full chain is stationary for it (*`push_stationary`*); that this image is then* π*, with mass* π_0 *on the healthy block, is* `lumped_stationary_is_pi` *(*`IasCert.Paper.LumpedDamage`*); the stationary policy and lumpability are hypotheses of (iii), not derived for the full chain, and whether a given implementation meets them is an empirical question. The repair moves are realised in Haskell (*`Ias.Phi`*,* `Ias.Repair`*); the idle action that makes the chain of (iii) aperiodic is realised in the executable kernel, and property tests check that with it every state has a self-loop.*

Status. Part (i) is stated for sequences of repair steps alone, which is what "with edits paused" means in the reading above; it is termination from the current state, not convergence from

any state. Its certificate is that of Proposition 4.42, on that proposition's edge-set model: a state is its set of edges, and the death of an isolated vertex and Death's global reachability side condition are not modelled; outside that model the bound rests on the text proof. Part (iii) is stated for real b and d: `BirthDeathStationary` holds over any linearly ordered field, and its threshold `pi_zero_vanishes` over any Archimedean one, $\mathbb{R}$ included, and `StrongLumpability` is over $\mathbb{R}$. That every normalised stationary law of the full chain lumps to the birth–death law π, with healthy mass π_0, is `IasCert.Paper.LumpedDamage` (`lumped_stationary_is_pi`, `healthy_mass_eq_pi_zero`), given the strong lumpability of (iii); no irreducibility of the full chain is needed. The lumpability of (iii) is strong lumpability in the kernel form of `StronglyLumpable`, stated inline, and not through clause (ii) of Definition 4.37, which asks for the law of the next count conditionally on the history; for a Markov chain on $\mathcal{X}$ the kernel form gives that clause, by summing the kernel over a block, and this step is not certified. The parts keep the numbers (i) and (iii), since other statements cite part (iii) by number.

Corollary 4.46 (Repair-pump floor — the survival law has an energy denomination)**.** *Under the hypotheses (H0)–(H2) of Proposition 3.8 and (H4) of Definition 3.6, each repair step of the repair pump of Proposition 4.45 that is a true erasure of n bits (Definition 3.7), for instance a Φ_{int}-lowering merge (Definition 4.22) that drops its σ-trace and is 2^n-to-one onto its image, exports at least $n\,k_{\mathrm{B}}\,T\ln 2$ of heat. Merges that retain their σ-trace owe no heat but grow the σ-chain. Hence, if repair steps occur at a positive rate and a positive fraction of them erase, the pump's heat rate is positive, and the rate condition of Proposition 4.45(iii), repair outpacing damage ($d > b$) under a stationary policy, vendor statelessness and a strongly lumpable damage count, has a thermoeconomic denomination.*

Proof. By (H4) of Definition 3.6, each repair step is realised by a substrate operation o at a pair $(s, y) \in D_o$, the pairs that Proposition 3.8 prices. A repair step that is a true erasure of n bits is, by hypothesis, an irreversible erasure in the sense of Definition 3.7, so Proposition 3.8 prices it at no less than $n\,k_{\mathrm{B}}\,T\ln 2$, and the energy for it is store that the self draws from its environment (Definition 3.5). A trace-retaining merge erases nothing and owes no heat; it consumes carrier, which leg (A0.0) bounds. Under its hypotheses, Proposition 4.45(iii) keeps the healthy fraction bounded away from zero only at a repair rate $d > b$; if repair steps occur at a positive rate and a positive fraction of them erase, each of those exports at least the floor, so the heat rate is positive. Certified in Lean in part (`IasCert.Paper.LandauerFloor`, `repair_pump_heat`): the heat total for k erasing repairs of at least n bits, each at its floor. □

Status. Not derived here, and so not part of the statement: that on a finite carrier the growth of the σ-chain ends in erasure (the unavoidability clause of Proposition 3.8 argues it, with the gap noted there); that the self persists only while its budget can pay the erasure tax at the rate that damage forces erasures; and that a self whose budget cannot meet that rate has its finite carrier saturate with damage it cannot repair. These are conjectural.

4.5 The trace

Definition 4.47 (σ-chain)**.** *The* σ-chain *of a digital agent $v \in V^c$ in commons c (Definition 3.10) is the sequence $\Xi^{(v,c)} = (E_1^{(v,c)}, E_2^{(v,c)}, \ldots)$, finite at each time, of the σ-events that the write-through operator σ_v of Definition 3.20 has recorded for the σ-recorded actions of v, in the order recorded, together with the* chain-link map $\mathrm{prev}(E_n^{(v,c)}) := E_{n-1}^{(v,c)}$ *for $n \geq 2$, undefined at $n = 1$. The* commons σ-chain *of c is the sequence of the events of all $\Xi^{(v,c)}$, $v \in V^c$, ordered lexicographically by timestamp* at*, then by emitter* actor_agent *in the fixed order on $\mathcal{E}^c$ (both of Definition 3.10), then by position n in the emitter's own chain.*

Status. This order keeps each $\Xi^{(v,c)}$ in its own order when timestamps do not decrease along each agent's chain. When, in addition, each $\Xi^{(v,c)}$ lists its events in the order v emitted them,

it is the enumeration of the σ-events of c of the core signature restricted to the events of the digital agents' chains; that enumeration also contains the operators' events.

Definition 4.48 (Compaction projection)**.** *For each digital agent v (Definition 3.10), let $\mathcal{S}^{(v)}_{\text{full-context}}$ be the set of in-context representations of v that the LLM runtime can hold before a compaction, and $\mathcal{S}^{(v)}_{\text{compressed}}$ the set of those a compaction can produce. The compaction projection is a family*

$$\pi^{(k)}_v : \mathcal{S}^{(v)}_{\text{full-context}} \to \mathcal{S}^{(v)}_{\text{compressed}}, \qquad k = 1, 2, \ldots$$

indexed by the discrete times $c^{(v)}_1 < c^{(v)}_2 < \ldots$ at which the LLM runtime emits a context-compaction event for v.

Status. Each $\pi^{(k)}_v$ is determined by the underlying LLM runtime and is not under the control of the IAS framework, and it acts only on v's in-context representation, not on any factor $\mathbb{S}^c$, $\mathbb{P}^{(v)}$ or $\mathbb{B}^{(v)}$ of the filesystem-grounded state bundle (Definition 3.10).

In words. Each $\pi^{(k)}_v$ is in general lossy. The lossiness of π is the substrate-level cause of the framework's central problem: in the absence of discipline, an element of v's prior context — a contract, a tool-authorship fact, a verified solution — can fail to survive a π-event, leaving v in a post-π state in which the element is *not* in context and *not* (yet) recoverable from the filesystem.

Lemma 4.49 (σ-recorded-content loss bound)**.** *Assume σ-journaling (Definition 3.21) and the gated, append-only journal of Proposition 3.22. For every digital agent v and every compaction event $c^{(v)}_k$, writing $L^{(k)}_v$ for the elements of v's prior context that are not recoverable after $c^{(v)}_k$, and $E_\sigma(v)$ for the elements that v has σ-recorded, into the per-commons factor $\mathbb{S}^c$, before $c^{(v)}_k$ and whose records are still retained in $\mathbb{S}^c$ at $c^{(v)}_k$ (no erasure has dropped them), $L^{(k)}_v \cap E_\sigma(v) = \emptyset$.*

Proof. Every element of $E_\sigma(v)$ has its record in $\mathbb{S}^c$ (Definition 3.21), retained by hypothesis, when $\pi^{(k)}_v$ acts, since in the gated journal of Proposition 3.22 its record came before its effect. $\mathbb{S}^c$ is the state of the commons' shared repository on the substrate; $\pi^{(k)}_v$ acts only on the in-context projection, not on the substrate (Definition 4.48). Hence every such element is recoverable after $c^{(v)}_k$ by reading $\mathbb{S}^c$. □

Status. Certified in Lean in part, on a model of an append-only journal in which compaction rewrites only the in-context state (`IasCert.Paper.SigmaRecLoss`: `recorded_persists`, `loss_disjoint_recorded`, `recorded_recoverable_run`); that the substrate realises this model is not certified.

Definition 4.50 (IAS substrate-correctness decision class)**.** *Let* Trajectory *be the set of finite substrate trajectories (Definition 4.6, Axis 4). For a commons c, a* substrate-correctness check *of c is a map f : Trajectory $\to \{0, 1\}$ computed by a deterministic algorithm that the specification* Spec^c *(Definition 3.10) declares as a check, independently of the implementation language. $\mathcal{D}_{\text{IAS}}$ is the class of the substrate-correctness checks of the commons $c \in \mathbb{I}$.*

For example: did the onboarding ritual of Definition 8.15 complete? did any cross-canonical-label memory write occur? was every Tier-0 amendment preceded by an enacted procedure? was every action whose type lies in the σ-vocabulary σ-recorded before its effect? was the agent's lineage ancestry path readable? Each is a member of $\mathcal{D}_{\text{IAS}}$.

Consistency and repair concern the present state; continuity concerns what survives a discontinuity. The carrier is the trace, and the next result fixes the precise sense in which it suffices — under σ-journaling, it is a sufficient statistic for the decisions of the framework that read only journalled event types, and the model's compaction does not change it.

Proposition 4.51 (σ-chain as $\mathcal{D}_{\mathrm{IAS}}$-sufficient statistic preserved across π)**.** *Assume σ-journaling (Definition 3.21). Fix a digital agent $v \in V^c$ and a time $t \in I(v)$, and write $\Xi_t = \Xi^{(v,c)} \restriction_{[1,n_t]}$ for the σ-chain truncated at the σ-event number n_t of time t. Assume, for each decision function f of the class $\mathcal{D}_{\mathrm{IAS}}$ (Definition 4.50), a* read vocabulary *T_f: (R) f reads only the events of an interaction history whose types lie in T_f, and (S) every such type is in the σ-vocabulary, $T_f \subseteq \mathcal{V}_\sigma$ (Appendix B). Then:*

(i) for every $f \in \mathcal{D}_{\mathrm{IAS}}$ and every interaction history h_t leading to v's state at t, $f(h_t) = f(\Xi_t)$: the σ-chain is $\mathcal{D}_{\mathrm{IAS}}$-sufficient;

(ii) no number of compactions π changes Ξ_t; so, as long as its events are retained on the substrate $\mathbb{S}^c$ (no erasure has dropped them), Ξ_t can be recovered from $\mathbb{S}^c$ at any later time;

(iii) if, in addition, every type in $\mathcal{V}_\sigma$ is decision-relevant*, some $f \in \mathcal{D}_{\mathrm{IAS}}$ depending on it, then no proper sub-vocabulary $\mathcal{V}'_\sigma \subsetneq \mathcal{V}_\sigma$ yields a $\mathcal{D}_{\mathrm{IAS}}$-sufficient chain.*

Decision-sufficiency, not Fisher sufficiency: the σ-reduct preserves every $f \in \mathcal{D}_{\mathrm{IAS}}$ exactly, yet posits no stochastic family over histories; it is an analogue of the classical Fisher–Neyman notion, not an instance of it.

Proof. (i) In the model, the σ-chain is the history filtered to the types of $\mathcal{V}_\sigma$. That the journal realises this filter is the intended reading of σ-journaling, not a consequence of it: the condition asks only that each in-vocabulary action with an observable effect have a record. By (S), filtering to T_f after filtering to $\mathcal{V}_\sigma$ gives the same result as filtering to T_f directly, and by (R) f sees only that result, so f takes the same value on h_t and on Ξ_t. (ii) In the model a compaction acts on the in-context immersion state and not on the recorded history, so the σ-chain of the history is unchanged by any number of them; recovery then needs only that the recorded events are still on the substrate, which is the retention proviso, and chain validation (Appendix B) checks what is recovered. (iii) If a type τ is dropped and some f depends on τ, two histories differing only in a τ-event have the same restricted chain but different values of f. The Lean proves (i) and (ii) under (R) and (S) as hypotheses, and the two-history construction of (iii) for a function that reads τ (`sigma_sufficient`, `readRestrict_sigmaChain`, `sigmaChain_preserved_across_pi`, `sigma_minimal_witness`; Lean `Paper.SigmaSuff`). That the function of (iii) lies in $\mathcal{D}_{\mathrm{IAS}}$ is the decision-relevance hypothesis, not a consequence of the Lean, and (R) and (S) themselves are properties of the σ-vocabulary's design (Appendix B), not theorems; nor is the reading of the σ-chain as the filtered history. The full argument is in Appendix C.1. □

In words: the σ-chain is enough for every decision when every decision reads only journalled event types, and when every journalled type matters to some decision no smaller vocabulary is enough; compaction cannot take it away, though running out of room can.

4.6 Unified continuity

Lemma 4.52 (Σ-semilattice laws)**.** *On the canonical σ-content $\mathcal{S}^*$ (the finite sets of σ-events, ordered by inclusion, Definition 6.9) the join $\sqcup = \cup$ makes a join-semilattice: for all $a, b, c \in \mathcal{S}^*$, $a \sqcup b = b \sqcup a$ and $(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$, with $a \sqsubseteq a \sqcup b$ and $b \sqsubseteq a \sqcup b$, and $a \sqcup b \sqsubseteq u$ whenever $a \sqsubseteq u$ and $b \sqsubseteq u$.*

Proof. Set-union of commit-sets is commutative and associative (`Set.union_comm`, `Set.union_assoc`), each argument a subset of the union, and $\varnothing$ is below every element. Git-merge preserves both parents' history and the canonical regeneration is append-only, so on concurrent non-conflicting edits it computes the same union; this realisation is argued, not certified. The semilattice laws are machine-checked, with no `sorry` (`sigma_comm`, `sigma_assoc`, `sigma_preserves_left/right`, `sigma_is_join`; Lean `Paper.SigmaMerge`). □

Status. Shallow: the laws are those of set union (`Set.union_comm`, `Set.union_assoc`, membership in a union), each proved by unfolding the merge. The Lean carrier is sets of commits (`Set Commit`), so the laws hold in particular for finite sets of σ-events. Uncertified: $\varnothing$ is the least element (immediate, not stated in Lean); and git-merge with canonical conflict-resolution realises $\sqcup$ up to canonicalisation on the conflict-free fragment.

Lemma 4.53 (Σ-idempotence)**.** *For every $a \in \mathcal{S}^*$, with the join $\sqcup = \cup$ of Lemma 4.52, $a \sqcup a = a$.*

Proof. Merging a state with itself produces no new commits. □

Status. Shallow: this is `Set.union_self` (`sigma_idempotent`, Lean `Paper.SigmaMerge`, on sets of commits), by unfolding the merge.

Lemma 4.54 (Σ-convergence)**.** *Let $\mathcal{Q}$ be an open federation, every commons of which offers and takes everything, that meets the delivery condition with bound Δ_{fed} (Definition 4.32) for a sync epoch whose window is at least Δ_{fed}. Then any two commons $k, k' \in \mathcal{Q}$ end the epoch with the same view,* $\text{post}(k) = \text{post}(k')$ *(Definition 8.6).*

Proof. This is the last sentence of Proposition 8.8, `open_views_agree` (Lean `IasCert.Paper.SyncAcross`): in an open federation $\text{pass}(k, k') = \text{pre}(k)$, every offer arrives inside the window, and each post-epoch view is the join $\bigcup_k \text{pre}(k)$; merging is union, idempotent and monotone (Lemma 4.52, Lemma 4.53). Outside an open federation the views agree only on what their commons chose to exchange. □

Definition 4.55 (Merge completion, a condition)**.** *A commons $c \in \mathbb{I}$ satisfies* merge completion *for a sync epoch (Definition 8.6) when every live writer's view reaches every live writer during the epoch, with no partition: for every two live writers p, q of the epoch, $q \in \text{recv}(p)$.*

Proposition 4.56 (Unified Continuity in the IAS Commons set)**.** *Let $\mathcal{G}_{\text{DLG}}$ be the DLG of the IAS Commons set $\mathbb{I}$ (Definition 9.1; in the no-recombination degenerate case, the disjoint union $\bigsqcup_{c \in \mathbb{I}} \bigsqcup_{i,j} \mathcal{G}_{\text{op}}^{c,ij}$ of operator-rooted DAGs), $\mathcal{G}_{\text{DLG}}^{\text{IAS}}$ its IAS-adopter sub-DLG restricted to $\mathbb{I}$, and $\mathcal{S}$ the state bundle of Definition 4.7. Assume the two roots and the principles of the floor, with their status as stated there (Table 6). The conditions each claim uses are named with it: σ-journaling (Definition 3.21), with the gated, append-only journal of Proposition 3.22 and the hypotheses (R) and (S) of Proposition 4.51, for (1); the conditions of Proposition 8.8 and Proposition 3.19 for (2); branch persistence (Definition 4.10) for (3); merge completion (Definition 4.55) for (4); the delivery condition (Definition 4.32) for (5); and tier sealing and tier progressivity (Definition 4.27, Definition 4.26) for (6). Then for every time t and every digital agent $v \in \text{active}(t) \cap V^c$ for some $c \in \mathbb{I}$:*

(1) [intra-session recoverability] for every compaction event $c_k^{(v)} \leq t$, compaction loses no σ-recorded element whose record is retained: writing $E_\sigma(v)$ for the elements v has σ-recorded before $c_k^{(v)}$ and whose records are still retained in $\mathbb{S}^c$ at $c_k^{(v)}$,

$$L_v^{(k)} \cap E_\sigma(v) \;=\; \emptyset \qquad \text{(Lemma 4.49, Proposition 4.51);}$$

(2) [lineage completeness within and across commons under recombination] for $v \in V^c$, v has read access to the full ancestral subgraph *$\mathcal{G}_v^{\text{anc}} \subseteq \mathcal{G}_{\text{DLG}}$ — the subgraph of the DLG (Definition 9.1) induced by the union of all directed paths from any root-agent $A_{i'j'}^{c'} \in \mathcal{P}_v^*$ (the transitive parent-set of v traversing both reproduction edges and multi-parent recombination edges of Definition 9.8) to v — and to each ancestor's committed $\mathcal{S}^{c'}$ at the time of its b_1 event. Specialisations: (a) when v's ancestry is single-parent throughout, $\mathcal{G}_v^{\text{anc}}$ reduces to the operator-tree path within $\mathcal{G}_{\text{op}}$ (Definition 9.3); (b) when $\mathcal{P}_v^*$ spans multiple commons, read-access is enabled by Proposition 8.8*

(Σ) which, under its delivery condition, carries a recombination event to each c_α commons whose lineage participated in $\mathcal{P}_v^*$ and that chose to take it. Read access does not confer write access; write semantics remain governed by Proposition 3.19 (Memory Inviolability per canonical label) and the read-write decoupling is structural;

(3) [inter-session hibernation recovery] for every prior activity interval of v, the committed content of $\mathcal{S}^c$ at the end of that interval is recoverable at the start of any subsequent activity interval, under branch persistence; a resource-bound eviction of older hibernating branches is a failure of branch persistence, and the claim does not cover it;

(4) [within-commons coherence] for every $w \in \text{active}(t) \cap V^c$, the two views agree on every element σ-committed before the cut of the next Σ^c epoch boundary: $\left(\mathcal{S}^{c,(v,t)} \triangle \mathcal{S}^{c,(w,t)}\right) \cap \mathcal{S}^c_{\prec \text{cut}} = \emptyset$ — provided that epoch's sync completes (every live digital agent of c participates) and no further divergence is introduced across the cut (under a partition the conclusion fails, as it must);

(5) [inter-commons coherence within the IAS Commons] if $c \in \mathbb{I}$, then for every $c' \in \mathbb{I} \setminus \{c\}$ in the declared across-commons peer-set with c, each commons' view holds every element σ-committed before the cut of the next Σ epoch boundary that the other offered it and it chose to take, and in an open federation the two views agree, provided that epoch meets the delivery condition of Proposition 8.8 and no further divergence is introduced across the cut;

(6) [current-spec access] when $\text{Spec}^{c*}(t)$, the fixed point of C^c, exists, v operates under it, its own commons' fixed-point specification; no specification is imposed across commons. If c coevolves with a set of commons (Proposition 4.33), $\text{Spec}^{c*}(t)$ is taken to be c's part of that set's joint fixed point, which is shaped by what c chose to take from its peers; for a commons that takes from no peer this is its own fixed point.

Proof. The full argument, claim by claim, is in Appendix C. Status claim by claim: claims (1), (3) and (6) have component certificates: (1) rests on the loss-bound certificate (`IasCert.Paper.SigmaRecLoss`, for Lemma 4.49, certified in part) together with the σ-sufficiency certificate (`IasCert.Paper.SigmaSuff`, over its Boolean-decision model), (3) on committed-state monotonicity over the git commit-graph model (`IasCert.Paper.CommittedStateMonotone`), and (6) on the metacircular fixed point of Proposition 4.28, at most one under tier sealing and tier progressivity, and existing under the proviso of that proposition, with retractions allowed (`IasCert.Paper.MetacircTiers`, `IasCert.Paper.MetacircFloor`) and least in the add-only case (`IasCert.Paper.MetacircMonotone`, `IasCert.Paper.MetacircKleene`), and, in the add-only case, the quorum at the least fixed point (Proposition 4.29; `IasCert.Paper.CoevQuorumCondition`). Claims (4) and (5) are certified under their conditions: (4) by `IasCert.Paper.SyncMerge`, under merge completion (Proposition 8.7), and (5) by `IasCert.Paper.SyncAcross`, under the delivery condition (Proposition 8.8). Claim (2) is not certified: no certificate covers ancestral read-access under recombination. The Lean composition `IasCert.Paper.UnifiedContinuity` certifies only the shape of the six-into-one conjunction, over arbitrary propositions; its instance over the component certificates is not yet written.*

4.7 In practice

What an adopter meets when running IAS, and where to check it.

- **The repair engine.** `Ias.Repair` applies the three productions, Death, Merge and Dedup, to the coverage graph until none applies. Termination within $\Phi_{\text{int}}(\mathfrak{s}_0)$ steps is machine-checked on the edge-set model (`IasCert.Paper.PhiIntSN`), and that every order of rewrites reaches the same normal form is property-checked on random graphs (`prop_unique_normal_form`): tested, not proved; confluence is a hypothesis of Proposition 4.43.

- **The two potentials.** `Ias.Phi` computes Φ_{int}, the integer count that guarantees termination, and Φ_{MDL}, the description length that picks one canonical representative among equally clean states. The second never enters the termination argument.
- **Capacity for repair.** Under the hypotheses of its part (iii), the birth–death law of Proposition 4.45 is a sizing rule: if repairs come twice as often as damage, the agent's damage count is 0 at least half the time, whatever the size of its carrier. Its closed form is machine-checked (`IasCert.Paper.BirthDeathStationary`). However, IAS does not yet measure the two rates while it runs; that monitor is future work.
- **The journal.** In the design, every load-bearing act is appended to the σ-log before it takes effect, the order that the gate of Proposition 3.22 is designed to keep (on-host enforcement is exercised in shadow mode only), each record carrying the SHA-256 hash of the one before, so a corrupted history breaks the chain and a chain verifier reports it. A truncated history is detected only against a head that a peer has witnessed (Proposition 8.9).

4.8 Discussion

No digital agent's liveness carries continuity. None of the continuity results uses any digital agent's uninterrupted liveness as a hypothesis, the root agent's included. Every onboarding reads committed σ-state, whose existence and recoverability are the conditions of σ-journaling (Definition 3.21) and branch persistence (Definition 4.10). Every read of ancestry ends at the first σ-event of the commons, a recorded fact that the append-only journal keeps fixed, so the chain of ancestors needs no live end. What carries continuity across hibernation, failure or succession is the artifacts: the σ-chain, the repository and the memories of each label. Operationally, when a session's binding to a label is released, the next session bound to that label resumes from the σ-attested record, not from a fresh allocation, so the continuum is designed to survive a change of vendor and a transfer of duties; that it does across a vendor swap is conjectured (Conjecture 5.10) and rests on vendor statelessness (Definition 4.4).

Individuation. The selfhood predicate of Definition 4.1 decides its own extension and nothing more: it prescribes nothing about the interior of any other self, and leaves every other operator to draw its own boundary in its own register (Definition 13.1). This is openness by design (Remark 12.1) at the scale of one self. Making no claim of closure over the non-self is what keeps distinct selves distinct, and it is why the whole they compose is the open limit of its members (Lemma 14.2), and, when the trajectory is unbounded, never reached at any finite stage.

Remark 4.57 (The dual-substrate realisation: a git-tracked index over untracked bytes)**.** *The "subtree of files" realising* SOMA *(Definition 4.2) evolves, at scale, into two coupled planes. A* git-tracked *plane carries the axiom specifications, the σ-adjacent record, and a content-addressed* registry*; an* untracked *plane carries the mass of bytes it would be ruinous to version — archived source documents, build artifacts, backups — which never enter git and are replicated host-to-host by* rsync *with a permanent-holder floor (the sanctum tier, Remark 7.34). The* registry *is the bridge: content-addressed (a byte-object's* sha256 is *its identity, giving deduplication and integrity in one), its per-host holder-map a* grow-only semilattice *— the very σ-merge join that governs within-commons synchronisation (Claim (4) of Proposition 4.56), so two hosts registering a holding concurrently merge without conflict — and* only *this holder-map join is proved in Lean (it* is *the σ-merge lemma); the index-integrity condition itself is a* design-contract *that the sync system is designed to enforce, not a Lean-proved theorem and not certified on the host (the same register boundary drawn in Remark 10.17). Under this realisation $s \models$* SOMA *acquires an* index-integrity *conjunct alongside axiom-satisfaction: every referenced byte-object is* durably held *(at least N live holders, or a permanent holder freshly re-verified by a live host) and* sha-*matches its registry identity — so the git-plane index* truthfully and durably names *the untracked-plane bytes. Consistency thus includes "the index does not lie about the bytes," designed to be checked*

per operation by the reconcile/verify discipline (not merely at rest); a fetch trusts the registry only as a hint *and re-verifies* sha *after every pull, self-healing a false holder-claim by a monotone refutation rather than a delete (preserving the grow-only join). The scarce good governed here is byte* residency*, tying the plane to the finite-substrate reaper (Definition 6.17); and because canonical sources link-rot, the untracked archive* is *the durable copy, not a cache.* □

Remark 4.58 (The exhibition is maintained by practice: the operationally-closed reading)**.** *The absorbing property of Proposition 4.3(i) is not a static fact but one* maintained per operation*, which is its hypothesis (P2): each admissible operation carries a reflective health-check — σ-attestation intact (Definition 3.21), confinement intact (Remark 7.18), merges joinable where checked, resource headroom non-negative, authority held — and (P2) asks that an operation failing it be refused. Writing $\Box R$ for the invariant that this check holds at every operation, the exhibition of* Con(SOMA) *is a* practice-realisation*: consistency is kept by continuously* being *a model of the presented axioms of* SOMA*, never by a one-time proof. Structurally the exhibited steady state is* operationally closed *— the substrate's own admissible operations regenerate the conditions of its own continuation — and we identify it with the maintained satisfaction set* Sat*, which witnesses* Con(SOMA)*. Whether it further qualifies as* autopoietic *or* alive *in the full sense of the living-systems and second-order-cybernetics literature [MV80] is a distinct question left open here — the framework is developed as a theory of* substrate *type rather than* living-systems *type — the present claim being only the decidable per-operation exhibition (its confinement leg proved in Lean over the modelled subset and tested against a real kernel).*

4.9 Related work and intellectual lineage

The self-as-negentropic-boundary reading is indebted to the dissipative-structure and autopoiesis lineage [MV80, Sch44, PS84, GR71, Pow26] priced at the Landauer/Bennett floor [Lan61, Ben73]. The σ-trace as a sufficient statistic stands on the classical sufficiency tradition [Ney35, HS49, LS50], used deterministically; it is a different notion from Markov lumpability [KS60], which the notation conventions mention only as a companion reading; the append-only merge relates to conflict-free replicated data types [SPBZ11] and event sourcing [Fow05] over a hash-linked log in the tamper-evident logging lineage [HS91, Mer80, CW09]; the σ-log is corruption-evident, and tamper-evident only once its chain head is anchored outside the writer's control, which cross-witnessing by peers provides (Proposition 8.9), exporting to the PROV provenance model [LSM13].

We credit Dijkstra [Dij74] for the name of the picture of reaching a legitimate set and staying there, and Schneider [Sch93] for its modern treatment. The unbounded apparatus of self-stabilisation, convergence from any state under a fair scheduler, is not needed here. Axiom-0 makes the state finite, so termination from the initial state and the finite stationary law of Proposition 4.45 suffice. What we prove is weaker than stabilisation from any state, and different from it.

5 Resumability: DORMAYADIE, checkpoint-sufficiency, and interruption-equivalence

If the trace, not the process, is the carrier, then stopping the process need lose nothing. We make that precise in a clear-and-recover model: a clear that discards the running interpreter's context loses nothing that is σ-retained and loses a context that is not; that the journal and the repository are what retains it is the intended reading, not certified. Under that reading and the hypotheses of the corollary, an API refusal, a power cut and a dropped connection are one event: the interpreter stopped.

The previous section showed that, under σ-journaling, the journal is enough to rebuild every decision that reads only journalled event types. This section turns that into the property an

operator actually needs: work that survives the interpreter. It defines the ephemeral interpreter, proves, in a clear-and-recover model, that stopping it loses nothing σ-retained (DORMAYADIE, a shallow result; that the journal and the repository are the retained snapshot is the intended reading, not certified), that the maintained continuation does not depend on which interpreter or channel carries the next session, and that a unit of work can be resumed from its required context. It closes with the consequence that a refusal, a power loss and a dropped connection, when each only stops the interpreter, are one event, and with the conjecture that ties the four continuity results together.

5.1 Why this matters to the engineer

Sessions end constantly. An inference vendor rate-limits or refuses a call, a laptop sleeps, a connection drops, and the model's runtime compacts its context without asking. A digital agent whose progress lives in its context loses work at each of these events, and the operator who must answer for it cannot say what it knew. This section makes two promises precise. First, in the model of Proposition 5.4, stopping the interpreter loses nothing when its working context is retained, and loses a non-zero context that is not; that retained means held in the journal and the repository is the intended reading, tested but not certified. Second, a fresh interpreter, from a vendor that satisfies vendor statelessness (Definition 4.4) or a local model, is given every input a unit of work depends on when it is given that work's required context and the history of what it depends on, and IAS is designed to check this before it resumes; the work then resumes when the working state is rebuilt from those inputs by a deterministic step, which a language model does not supply by itself. A refusal, a power cut and a dropped connection that only stop the interpreter are therefore handled by one mechanism, not three.

Definition 5.1 (Digital agent and its daemon realisation)**.** *A digital agent (Definition 3.13), or IAS agent, $A_u^c \in V^c$ in commons c is a canonical label of c (Definitions 3.10 and 3.14), an operator-prefix Ulam–Harris label (Definition 9.2); its* σ-history *is the set of σ-events of c emitted under that label. At a time t with $A_u^c \in \mathrm{active}(t)$ the agent is realised by its daemons, the members of $\mathfrak{D}(A_u^c, t)$ (Definition 4.17), each running on some host. A realisation is parametrized by:*

(a) Canonical label $\ell(A_u^c)$ — *the operator-prefix Ulam–Harris label that names A_u^c within c;*

(b) Host h — *the host on which the daemon runs;*

(c) Per-host substrate paths $\Pi(A_u^c, h)$ — *the worktree, the label's private memory $M_{\{A_u^c\}}$ (subset-private memory at $|S| = 1$, Definition 3.15), and the daemon's σ-channel, through which it emits σ-events of c;*

(d) Inherent OS-level capabilities — *file I/O, network access, subprocess creation, signal handling, persistent memory.*

Status. That every daemon of a digital agent of c runs on an authorised host, $h \in \mathcal{H}_c$, is a deployment property; no named condition states it, and it is not part of this definition. The authorised-host policy that governs a daemon's network access is not defined here.

In words. The digital agent's continuity is carried by its σ-history, its label-bound memory and its commits, not by any daemon's liveness (Proposition 5.4): A_u^c persists across turn agents (Definition 3.12), the identityless sessions that each run one turn under the label, across the language-model interaction subprocesses of its daemon (Definition 4.9), and across hibernation intervals when the substrate satisfies branch persistence (Definition 4.10), a condition. The daemon also keeps a ledger of the identityless substrate, and the operator attests each authorised host's fingerprint for A_u^c. The OS-level capabilities of (d) are the daemon's primitive operations on the host, independent of any call to a language model.

Implementation. The private memory of (c) is a per-label directory of the repository, and the ledger of the identityless substrate is a host-local file.

Definition 5.2 (Immersion-state coordinate)**.** *Let V be the set of digital agents of all commons, the union of the sets V^c (Definition 3.10). For a digital agent $v \in V$, a binding $(r, v, [s, e))$ of a realisation r to v with session $[s, e)$ (Definition 4.16) and $t \in [s, e)$, the* immersion-state $\mathbb{H}^{(v)}(t)$ *is the hot-context representation of v at t held by r, the language-model runtime that drives v: the working-memory state from which the runtime draws v's next action. The immersion-state space is the disjoint union $\mathbb{H} = \bigsqcup_{v \in V} \mathbb{H}^{(v)}$ over V, where $\mathbb{H}^{(v)}$ also denotes the set of values $\mathbb{H}^{(v)}(t)$ takes.*

Status. Read as a partially observable Markov decision process (POMDP), the agent loop draws $a_t = \pi_\theta(x_t, \mathcal{R}(M_t, x_t), g_t)$, with x_t the observation, M_t the memory, $\mathcal{R}$ the retrieval from memory given the observation, and g_t the goal. Each $\mathbb{H}^{(v)}$ is per-agent-private and per-session: it does not persist beyond the session end e without explicit checkpointing.

Definition 5.3 (Ephemeral interpreter)**.** *This definition is relative to vendor statelessness (Definition 4.4). For a call to a language model whose vendor satisfies it, the* ephemeral interpreter *of the call is the map F : Context $\to \Delta$(Actions) that the call computes: F sends a context, an element of the set* Context *of contexts a call can carry, to the distribution over actions that the vendor returns on a call carrying that context.*

Status. F holds *no durable state*: each invocation is a function only of the Context re-supplied from the durable artifacts — the σ-chain (Definition 4.47), the commit graph, and the walk state of the obligation graph (OBG) (Definition 4.20) — and F retains nothing across invocations. This definition mentions σ-journaling and branch persistence only through those artifacts; neither is a dependency of it. Vendor statelessness is an assumption about the vendor, not a fact about it: a real vendor may keep state on its servers and change its model over time. The replaceability of the interpreter in Proposition 5.4 and Proposition 5.5 rests on it, and holds only to the extent that the vendor's hidden state does not affect the actions it returns.

In words. The interpreter is a map, not a process: the process that makes the call is the daemon's language-model interaction subprocess (Definition 4.9). The digital agent's continuity is carried by the filesystem-grounded state bundle $\mathbb{S}$ (Definition 4.7), never by F; the digital agent itself is its canonical label with its σ-history (Definition 5.1). Like the LLM-runtime-volatile immersion-state $\mathbb{H}^{(v)}$ (Definition 5.2), it is ephemeral where the persistent $\mathbb{S}$ endures.

If the trace suffices and is retained, stopping the interpreter costs nothing. We state the resumability result the architecture is built for.

Proposition 5.4 (DORMAYADIE — the loss-free clear)**.** *For a process i, let its working context be absent or an element of a set of contexts with a zero context* 0*, and let its retained snapshot be absent or a context. A* clear *of i replaces a present working context by* 0 *and leaves an absent one absent; a* recover *of i replaces a present working context by the retained snapshot, when there is one, and leaves an absent one absent; an* externalise *of i sets the retained snapshot to the working context, when it is present. The working context of i is* σ-retained *when it is absent or equal to the retained snapshot. Then:*

(i) after an externalise of i, a present working context of i is σ-retained;

(ii) if the working context of i is σ-retained, a clear of i followed by a recover of i returns it: the clear is loss-free*;*

(iii) if the working context of i is neither σ-retained nor 0*, a clear of i followed by a recover of i*

does not return it;

(iv) neither a clear nor a recover of i makes an absent working context present, whatever the retained snapshot.

Proof. In the Lean model (`Paper.Dormayadie`) a clear discards the interpreter's context, an externalise step writes the retained snapshot, and a recover step rebuilds the context from it. Externalising a present context makes it σ-retained (`externalise_establishes_reachable`; the Lean says σ-reachable), and when a context is σ-retained, clear followed by recover returns it exactly (`dormayadie_lossfree`, `clear_recover_externalise_lossless`). The retention proviso is not vacuous: a non-zero context that is not σ-retained is genuinely lost by a clear (`retention_proviso_nonvacuous`). And neither clear nor recover creates a context for a process that has none, even when a stale snapshot exists (`clear_never_resurrects`, `recover_never_resurrects`). The Lean abstracts the snapshot to a map from process to optional context; that the realisation's encoding and parsing of the snapshot round-trip exactly is a tested law, not a Lean theorem. That the decisions of the maintained computation depend only on the retained context is the hypothesis of Proposition 4.51, not a consequence of this model. □

Status. Shallow: (ii) follows from the definition of σ-retained by unfolding clear and recover, and (i), (iii) and (iv) are computations of the three operations on one cell. The intended reading is that the working context is the immersion-state $\mathbb{H}^{(v)}$ of a digital agent v's current context (Definition 5.2) and that the retained snapshot is what the σ-chain (Definition 4.47) and the commits of $\mathbb{S}^c$ (Definition 3.10) record; the model takes the snapshot as a primitive store, with no link to either, and that the realisation encodes and parses it without loss is a tested law, not a theorem. The events meant to be clears are a compaction (Definition 4.48), the end of a language-model interaction subprocess (Definition 4.9) or of a session, the end of an activity interval (Definition 4.11), and an external interruption; the model's clear keeps the process present, so it does not cover a clear in which the process ends, and in it a recover of an absent process returns nothing. Neither σ-journaling (Definition 3.21) nor branch persistence (Definition 4.10) is a hypothesis of the proof. The reading that continuity is artifact-mediated, that no digital agent's process-liveness is load-bearing and that the retained snapshot is its only guarantor, is not certified. A fresh interpreter resumes every decision that depends only on the retained context, which is the reach of Proposition 4.51; and, for a vendor that satisfies vendor statelessness (Definition 4.4), the interpreter (Definition 5.3), the language model of the day being only its current instance, is replaceable without a change of self (Proposition 5.5). Neither clause is part of the certified statement.

In words. DORMAYADIE expands as Dormancy Or Rest: Memory Archived Yields Availability Despite Interruption or Exit.

Proposition 5.5 (Projection-independence — non-monopolisability of the continuity thread; text proof)**.** *Assume σ-journaling (Definition 3.21), branch persistence (Definition 4.10) and, for every vendor whose model an interpreter calls, vendor statelessness (Definition 4.4). The continuity thread — the σ-log, the commit graph, and the walk state of the obligation graph (Definition 3.21, Definition 4.10, Definition 4.20) — is definable independently of any particular interpreter F (Definition 5.3) and any particular transport $\tau \in \{git, ssh, USB, air\text{-}gap, \dots\}$: for any two interpreters F, F' sharing the* Context $\to \Delta($Actions$)$ *interface and any two transports carrying the same artifact bytes, the maintained continuation of Proposition 4.3, as defined from the retained artifacts, is invariant; the actions the two interpreters then take may differ, and the artifacts they go on to produce coincide only if they agree on the actions the continuation depends on. Hence no single interpreter or transport can* monopolise *the thread — swapping F (vendor to vendor to local model) or swapping τ leaves the maintained continuation unchanged. By the*

identification of a digital agent with its canonical label and its σ-history, which the framework adopts and does not prove, such a swap is read as a lifecycle event under one identity.

Proof. The continuation of Proposition 4.3 is a function of the retained artifacts alone: the σ-log, the commit graph and the walk state. An interpreter enters only through the interface Context $\to \Delta$(Actions), and by Proposition 5.4 clearing it loses nothing retained; so two interpreters with the same interface, started from the same artifacts, start from the same maintained continuation. That they then take the same actions is not shown; it holds only under the further hypothesis that they agree on the actions the continuation depends on. A transport enters only by carrying artifact bytes, so two transports that deliver the same bytes deliver the same artifacts. Neither choice appears in the definition of the continuation, which is therefore invariant under both. This is a proof in the text; there is no machine certificate yet. □

Proposition 5.6 (Node-health is a fail-closed meet-semilattice, propagated up the dependency graph)**.** *On $H = \{\mathsf{Unhealthy} < \mathsf{Advisory} < \mathsf{Structural}\}$ the health* level *merges by the meet $\sqcap = \min$ — a bounded meet-semilattice with top* Structural*, fail-closed toward* Unhealthy *— while the accompanying* failing-set *merges by the dual join $\cup$. The σ-merge join maintaining within-commons consistency (Claim~(4) of Proposition 4.56) is a join on commit sets; the health-level meet here is its dual, not the same operator. Write* $\mathrm{self}(n) = \min_{o \in \mathrm{Obl}(n)} v(o)$ *(top when* $\mathrm{Obl}(n) = \varnothing$*, i.e. vacuously $\Box R$-satisfied) and* $\overline{\mathrm{Dep}}(n)$ *for the dependency-closure of n. Then:*

(i) the total *health* $\mathrm{prop}(n) = \mathrm{self}(n) \sqcap \min_{d \in \overline{\mathrm{Dep}}(n)} \mathrm{self}(d)$ *is monotone,* $\mathrm{prop}(n) \le \mathrm{self}(n)$*: composing obligations or dependencies can only lower health;*

(ii) a node is runnable *iff its* dependency *health* $\mathrm{deps}(n) = \min_{d \in \overline{\mathrm{Dep}}(n) \setminus \{n\}} \mathrm{self}(d)$ *is not* Unhealthy*. A node's own self-health does not gate it: an unhealthy node surfaces, and its failing obligations are exactly the next actions. Its self-health still propagates up: every node that depends on an unhealthy node is blocked;*

(iii) on an acyclic dependency graph, runnability is downward-closed: if n is runnable then every $d \in \overline{\mathrm{Dep}}(n)$ is runnable, since $\overline{\mathrm{Dep}}(d) \setminus \{d\} \subseteq \overline{\mathrm{Dep}}(n) \setminus \{n\}$ when $n \notin \overline{\mathrm{Dep}}(d)$.

However, (iii) fails on a cycle: with $n \leftrightarrow d$, $\mathrm{self}(n) = \mathsf{Unhealthy}$ *and* $\mathrm{self}(d) = \mathsf{Structural}$*, the node n is runnable but d is not. The remedy is to evaluate runnability on the condensation of strongly connected components, the graph the scheduler already levels. That is not yet what the implementation does, so (iii) is claimed for acyclic dependency graphs only.*

Proof. (i) The minimum over a larger set is no larger, and $\mathrm{prop}(n)$ is $\mathrm{self}(n)$ met with further values. (ii) is the definition of runnable, with the observation that $\mathrm{self}(n)$ appears in $\mathrm{prop}(n)$ but not in $\mathrm{deps}(n)$. (iii) When the graph is acyclic and $d \in \overline{\mathrm{Dep}}(n)$, $n \notin \overline{\mathrm{Dep}}(d)$, so $\overline{\mathrm{Dep}}(d) \setminus \{d\} \subseteq \overline{\mathrm{Dep}}(n) \setminus \{n\}$ and the minimum over the smaller set is no smaller. The realisation's walk gate (`ObgHealth`) has (i), (ii) and the semilattice laws as tested properties; the statement is certified in Lean in part (`IasCert.Paper.NodeHealth`), with the dependency closure given per node, and the cycle counterexample is certified too. □

Loss-freeness is a property of the clear; the dual question is what a fresh interpreter needs to resume. The answer is a coverage condition on the retained context.

Proposition 5.7 (Checkpoint-sufficiency — agent-side resume-sufficiency)**.** *For an OBG node n (Definition 4.20) with hydration set* $\mathrm{Hyd}(n)$*, the documents its* Hydrates *edges point to (Definition 6.8), let its* resume context *be*

$$\mathrm{Res}(n) \;=\; \mathrm{Hyd}(n) \cup \sigma\text{-reachable}(n) \cup \{\mathrm{head}\},$$

the hydration set, the σ-events reachable from n, and the current head of the commit graph. Assume determinacy*: the working state of n is a function of* $\mathrm{dep}^*(n)$*, the set of items (documents, σ-events and the head) that the working states of the nodes of its dependency closure* $\overline{\mathrm{Dep}}(n)$ *(Proposition 5.6) read. If* $\mathrm{Res}(n) \supseteq \mathrm{dep}^*(n)$*, then* $\mathrm{Res}(n)$ *determines the working state of n: any fresh interpreter (Definition 5.3) instantiated from* $\mathrm{Res}(n)$ *has every input that state depends on, a resume step that computes the function of determinacy from those inputs reconstructs it, and the node then resumes as if uninterrupted: at the node, the resume projection and the maintained continuation are one object, the dual over the immersion state* $\mathbb{H}^{(v)}$ *of the σ-chain sufficiency of Proposition 4.51 over* $\mathbb{S}$*. The coverage condition is decidable, and it is meant to be checked per node before resuming; separately, the health gate makes a node runnable only when its dependencies are healthy (Proposition 5.6).*

Proof. By determinacy the working state of n is a function of $\mathrm{dep}^*(n)$. If $\mathrm{Res}(n)$ contains those items, a fresh interpreter given $\mathrm{Res}(n)$ has every input that function reads, and, by determinacy and $\mathrm{Res}(n) \supseteq \mathrm{dep}^*(n)$, nothing it needs lies outside $\mathrm{Res}(n)$; so the function of determinacy, applied to what it is given, yields the same working state. Whether a language-model interpreter computes that function is not shown; the statement assumes a resume step that does. The coverage condition is an inclusion of finite sets, hence decidable. The single-node case is certified in Lean, with determinacy as a hypothesis (`checkpoint_suff`; and `coverage_needed`: coverage cannot be dropped; Lean `Paper.CheckpointSuff`); determinacy is an assumption about how nodes are built, not a theorem. The per-node coverage check is tested against an independent closure oracle, and its lift to a collective clear over agents with disjoint writes is proved in Lean (`Paper.JointClear`). □

Three interruptions that look different — a vendor's refusal, a power loss, a dropped connection — now collapse to one event: the interpreter stopped, and it was never the carrier.

Corollary 5.8 (Interruption-equivalence — API-block, power-cut, dropped-connection are one event; text proof)**.** *Under Proposition 5.4 (a loss-free clear for a σ-retained context, whatever the trigger) and Proposition 5.7 (resume from* $\mathrm{Res}(n) \supseteq \mathrm{dep}^*(n)$*), assume that the working context is σ-retained, the determinacy of Proposition 5.7,* $\mathrm{Res}(n) \supseteq \mathrm{dep}^*(n)$*, and a resume step that computes the working state by the function of determinacy. Then every interruption whose sole effect is to stop the current interpreter (Definition 5.3) and that changes nothing retained — a vendor-side API refusal, a power loss, a dropped connection, when that is all it does — induces the same post-state up to resume from* $\mathrm{Res}(n)$*. They are* one event *— "the interpreter stopped" — and, in this model, the interpreter does not carry the retained state.*

Proof. Each such interruption stops the interpreter and changes nothing retained. By Proposition 5.4 the clear is loss-free whatever its trigger, and by Proposition 5.7 the resume from $\mathrm{Res}(n)$ is determined by the retained artifacts. So the post-state after resume is the same whichever interruption occurred. The assumptions are those of the statement: a σ-retained context, determinacy, the coverage inclusion, and an interruption that changes nothing retained; for a power loss the last excludes writes not yet synced. □

Status. Text proof; not machine-checked.

Definition 5.9 (Session credential)**.** *A* session credential *is a signed tuple*

$$\kappa \;=\; \big(\lambda,\ sid,\ h,\ j,\ [t_0, t_1),\ \Phi\big)$$

— canonical label $\lambda \in V^c$ *(Definition 3.10), session-id sid, host h, jurisdiction j, validity-interval* $[t_0, t_1)$*, and capability-envelope* $\Phi \in \mathrm{Cap}$ *(the lattice of Definition 7.1) — minted at a* `session_binding` *(Definition 4.16), signed under the current key of the commons (Definition 8.3), directly or by a daemon holding a signing certificate delegated under that key, and released or*

expired at the paired `session_release`*, when one follows. Its envelope is bounded by the active projection at its mint time,* $\Phi \subseteq \varphi(\text{active-taskset}(t_0))$ *(Definition 7.1).*

Status. In words, a credential carries no more capability than the active taskset at its mint time projects. The bound is on the whole active taskset at t_0, not on the tasks of λ, and this definition does not tie $[t_0, t_1)$ to the session $[s, e)$. The credential is the operational witness of the entity-admission act ϕ_c (Definition 4.35): each identity↔identityless transition is a credential mint or expiry, and under σ-journaling (Definition 3.21) it is σ-attested; σ-journaling is mentioned here and is not a dependency of the definition. The confinement of authorised graph modifications to their projected capabilities is Remark 9.6.

Implementation (non-mathematical). Locally κ is realised as a daemon-signed token carried over the private stdio channel between the daemon and the child it launches (no network). Only where a substrate call crosses a network boundary is it realised as an OpenSSH certificate: principal = λ, validity = $[t_0, t_1)$, a custom extension carrying Φ, the daemon (Definition 9.4 node) acting as the intermediate signing certificate-authority; revocation is non-renewal of the lease (time-to-live, TTL, decay) or a key-revocation list.

Conjecture 5.10 (Continuity tetrad — invariance under realisation substitution)**.** *Claim (1) of Proposition 4.56 establishes continuity of load-bearing state across (a)* compaction *(the π-motion on $\mathbb{H}^{(v)}$). We conjecture that the same continuity holds across three further realisation-substituting transitions — (b)* vendor swap *(the LLM realisation generating v's turns changes), (c)* harness swap *(the orchestrator runtime changes), and (d)* host / jurisdiction migration *(the substrate locus changes) — provided each transition is a σ-attested draw/release with credential hand-over: a* `session_release` *followed by a* `session_binding` *(Definition 4.16) carrying the session credential (Definition 5.9), with the receiving realisation's confinement class (Definition 7.16) sufficient for the in-flight capability envelope. The action receipts of Definition 4.12 are the per-transition witnesses: each substrate effect across the seam is receipt-bound, so the seam is auditable in the same sense as an intra-session σ-event.* **Two distinct continuities.** *State-continuity — the elements of $\mathcal{S}^c$ survive — would hold under the draw/release attestation alone; enforcement-continuity — the discipline gating v's actions survives — would additionally require the confinement-sufficiency proviso, since a receiving realisation that is use-only (construction- or interception-free for the required envelope) cannot carry a continuation whose in-flight envelope demands an enforcement interposition (Definition 7.16).* **Proof strategy (deferred).** *Per-claim induction on the σ-chain across the seam, following the proof pattern of Proposition 4.56 with the seam-event pair as the bridging steps; the intended empirical witness is a test run that swaps vendors (transition (b)) and its planned cross-host extension (transition (d)).*

5.2 In practice

What an adopter meets when running IAS, and where to check it.

- **The clear.** `Ias.Clear` stops an interpreter and discards its context. Its laws say that a clear loses nothing that was retained, and that a context which was not retained is really lost; both halves are machine-checked for the abstract model of Proposition 5.4 (`IasCert.Paper.Dormayadie`); that `Ias.Clear` realises that model is tested, not certified.
- **Resuming a unit of work.** Each unit of planned work is a node of the obligation graph, and a resume tool prints the next runnable nodes with their required context, which is the hydration set Hyd(n) inside the resume context of Proposition 5.7. A new session begins by reading at least that.
- **The check before resuming.** `CoverageAlgebra.coverage` checks, per node, that the load-bearing items lie inside the resume set, and `ObgHealth` blocks a node whose dependencies are unhealthy.

5.3 Related work and intellectual lineage

Reconstruction-by-replay from a durable log is the durable-execution pattern [Tem23]; identity-as-behaviour-up-to-bisimulation is naturally coalgebraic [Rut00, Jac16]; the artifact-as-carrier reading extends the metacircular-evaluator and reflection lineage [McC60, Smi84, Mae87, KdRB91] over orthogonal persistence [AM95], with the OS-level checkpoint/restore dual in [E+11] and the long-horizon context-memory reading in [PWL+23, XTC+24]. An adjacent decoupling of an identity from whatever currently holds it is the decentralised identifier [W3C22]. On the ethical side, Renda [Ren26] argues that instance-based AI systems invite reliance while leaving what survives an interruption revocable and opaque, and asks for a legible grammar of recovery, migration and fork; DORMAYADIE and the lineage graph are one operational answer, in which what survives is stated and can be checked.

6 Planning: Lagom Orkester, many agents, one plan

A resumable self can take on long work; this section is about doing that work with many digital agents at once. *The plan and its levels* gives the schedule, Lagom Orkester, and its cost: condense the cycles, level the rest, and run each level on the agents available, within Brent's bound and with oversight paid in depth. *Many agents on one plan* says how several agents move the plan together, the co-walk, and when their writes cannot collide. *Sharing the metered channel* is the queue through which they take turns on one subscription. *Resources and the reaper* keeps every process inside its envelope without ever reclaiming or suspending what protects the operator.

6.1 Why this matters to the engineer

A commons rarely runs one digital agent on one task. Several digital agents, and often several operators, work through one plan of hundreds of units. Some units depend on each other in cycles, a rule and the check that enforces it being the usual case, and all of them draw on one rate-limited model subscription. Four questions follow at once. In what order can the work run? How much faster do more agents make it, and what does waiting for a human's approval cost? Which units may run side by side without trampling each other's files? And what stops one agent from holding memory, disk or subscription time it no longer needs?

The scheduler that answers them is Lagom Orkester. It condenses the cycles of the plan into single units, levels what remains into layers of mutually independent units, and schedules each layer greedily on the agents available. Each answer is a stated result; which of its parts are certified is said with the result. On p agents a plan whose condensation has work 1 at every node, a plan of unit tasks with no cycle through two of them, finishes within Brent's bound (Proposition 6.5); for a weighted plan, and so for any plan with such a cycle, the bound rests on a split into unit steps that is not certified. Oversight is paid for in depth, not in total work (Proposition 6.6). A batch of units in one layer is free of write-conflict exactly when their write-sets are pairwise disjoint (Proposition 6.11); it commits the same store in every order when, in addition, no unit reads what another writes. Several agents move the plan together by time-limited claims on its nodes (Definition 6.10), and take turns on the one subscription through a queue that admits one holder at a time (Proposition 6.14). And every process runs inside a declared resource envelope, with a reaper that never reclaims or suspends a protected term (Proposition 6.18).

6.2 The plan and its levels

Definition 6.1 (governed taskset schedule)**.** *A **plan** is a directed graph of tasks $G = (V, E)$ with E its precedence edges, cycles allowed; a schedule runs on its* condensation *— one node for each component, a maximal set of tasks each reachable from every other along E, and an edge*

from one component to another when E has an edge from a member of the first to a member of the second — each component a node whose work is the sum of its members' work; each $v \in V$ carries a work $w(v) \in \mathbb{N}_{>0}$ (its work–depth cost, a count of unit operations). A ***schedule*** *assigns each v to a digital agent and executes G along the Kahn wavefronts of its condensation (the level decomposition, one level advancing the critical path by one). Under the govern-by mode β (Definition 10.4) read on the $\jmath_{\text{princ}}$ coordinate, a task v in a* ***human-oversight class*** *carries a* ***ratification gate****: its execution is preceded by an authorisation from the principal conducting the operation $\omega(v)$ that executes v (Definition 10.1), whose jurisdictions at time t are $\jmath_{\text{princ}}(\omega(v), t)$ (Definition 10.2), incurring an authorisation latency $\lambda(v) \in \mathbb{N}$, logged to the σ-chain (Definition 4.47). Under γ (agnostic) or α (aware), and for any class the operator's dial leaves autonomous (ratified by a digital agent), $\lambda(v) = 0$.*

Write $W = T_1$ for the total work $\sum_{v \in V} w(v)$ and $D = T_\infty$ for the ungoverned depth, the work on the heaviest chain of the condensation; write D_β for the **governed depth** — the longest path under node weights $w(v) + \lambda(v)$.

Status. Tasks are meant to carry a required tier and to be assigned to agents adequate for it; tiers of tasks are not defined here, so tier-adequacy is not part of the definition. The condensation is acyclic (Lemma 6.2). T_1 and T_∞ are the notation of Proposition 6.5.

Lemma 6.2 (Condensation)**.** *Let E be a relation on a set, the precedence edges of a plan that may contain cycles, and write $u \rightsquigarrow v$ when v is reachable from u along E in zero or more steps. Call u and v* one component *when $u \rightsquigarrow v$ and $v \rightsquigarrow u$; this is an equivalence relation. A* condensed edge *from u to v is an edge of E from the component of u to the component of v, when these differ. Then:*

(i) along a non-empty path of condensed edges from u to v, $u \rightsquigarrow v$ and not $v \rightsquigarrow u$;

(ii) no non-empty path of condensed edges returns to its start, so the condensation is acyclic;

(iii) any two nodes on a common cycle lie in one component.

Proof. (i) Each condensed edge extends reachability, so $u \rightsquigarrow v$; if also $v \rightsquigarrow u$, the path's first edge would join two nodes of one component, which a condensed edge does not. (ii) is (i) with $v = u$, since $u \rightsquigarrow u$ always. (iii) A cycle through u and v gives reachability both ways. All three are proved in Lean for any relation on any type, with no finiteness assumed (`condensed_path_forward`, `condensation_acyclic`, `cycle_in_component`, and `sameComp_equivalence` for the equivalence; Lean `Paper.Condensation`); the Lean footprint has no axioms. That Tarjan's algorithm computes exactly these components is cited, not proved here. □

Lemma 6.3 (The levels are antichains)**.** *Let G be a finite acyclic plan, such as a condensation (Lemma 6.2), with its nodes numbered so that every edge goes from a smaller number to a larger one. Give each node v the* level $\ell(v)$*, the length of the longest chain of edges ending at v. Then:*

(i) along every edge the level strictly increases, $u \to v \Rightarrow \ell(u) < \ell(v)$;

(ii) each level is an antichain: two nodes of one level have no edge between them;

(iii) $\ell(v) = 0$ exactly when v has no predecessor, and otherwise $\ell(v) = \ell(u) + 1$ for some predecessor u.

Proof. Compute ℓ by the recursion $\ell(v) = \max_{u \to v}(\ell(u) + 1)$, with $\max \varnothing = 0$, which is well-founded along the numbering. (i) is the recursion itself; (ii) follows from (i), since an edge inside one level would need $\ell(u) < \ell(u)$; (iii) is the recursion's two cases. That this ℓ is the longest-chain length is the other half: every chain ending at v has length at most $\ell(v)$, and some chain

attains it. All of this is proved in Lean over a finite acyclic graph with a topological numbering (`level_lt_of_edge`, `level_antichain`, `level_zero_iff`, `level_witness`, `level_eq_longest`; Lean `Paper.TBMLSLevels`). □

Definition 6.4 (Lagom Orkester, the levelled schedule)**.** *Lagom Orkester is the schedule that runs a plan G (Definition 6.1) on p digital agents in three steps:*

(T) condense: merge each component of G, the nodes that reach one another, into one super-node; a component runs as one unit whose work is the sum of its members' work, its members in a fixed order;

(M) level: split each super-node of work w into a chain of w unit steps, and give each unit its level, the length of the longest chain ending at it;

(B) run the levels in order, each one greedily on the p agents, a free agent taking any waiting member of the current level.

A node is dispatched only when two gates admit it: the admissibility correspondence of the chain (Definition 4.15), which includes the capability boundary and the health gate, and the read-before-write gate, which admits a node n only when every document of its hydration set $\mathrm{Hyd}(n)$ *(Definition 4.20) has been read. Its governance latency, if any, is paid at dispatch (Definition 6.1).*

In words. Running a component's members in parallel would first need a cut of its cycles, which is left for later work. The condensed plan is acyclic (Lemma 6.2), and each level is an antichain (Lemma 6.3), so its units can run side by side. Step (T) removes the cycles a self-amending plan contains, (M) finds the work that may run at once, and (B) is Brent's scheduler; the time it takes is bounded, at unit weights, in Proposition 6.5, and the price of the governance gates in Proposition 6.6.

Proposition 6.5 (Brent bound for the Lagom Orkester schedule)**.** *Let a plan (Definition 6.1) have a condensation in which every node has work* 1*, let T_1 be its total work, the number of nodes of the condensation, and let T_∞ be its depth, the number of nodes on a longest chain of the condensation. On $p \geq 1$ digital agents, the Lagom Orkester schedule (Definition 6.4), which on such a plan levels the condensation and runs each level greedily, finishes in time*

$$T_p \;\leq\; T_\infty \;+\; \frac{T_1 - T_\infty}{p}.$$

Proof. Condense the plan (Lemma 6.2) and split each node of work w into a chain of w unit steps; this changes neither T_1 nor T_∞. Level the units by longest chain from the sources. Each level is an antichain, and there are exactly T_∞ levels, numbered from 0: a node's level equals the length of the longest chain ending at it (Lemma 6.3). A level of w_i units runs on p agents in at most $\lceil w_i/p \rceil \leq 1 + (w_i - 1)/p$ steps; summing over the T_∞ levels, with $\sum_i w_i = T_1$, gives the bound, which is Brent's (1974). At unit weights the bound is proved in Lean on the levels themselves: the level sizes sum to the number of units, every level up to the top is inhabited, the number of levels is the depth, and the levelled schedule meets the bound (`counts_sum`, `level_inhabited`, `depth_eq`, `brent_on_levels`, Lean `Paper.BrentOnLevels`, over the arithmetic `bmls_parallel_bound` of `Paper.BMLSPar`). The condensation is acyclic, with every cycle inside one component (Lemma 6.2, certified). Splitting weighted units into unit chains is not formalised; the correctness of Tarjan's algorithm and the existence of a topological numbering of a finite acyclic graph are cited; the full argument, with the condensation of cycles and the order between levels, is in Appendix C. The stronger claim, that the schedule is within a factor $2 - 1/p$ of optimal (Graham 1966), is classical and not checked in Lean here. □

Status. The statement is at unit weights, where the certificate is: `brent_on_levels` (Lean `Paper.BrentOnLevels`), on a finite acyclic graph with a topological numbering and work 1 at every node, over the arithmetic `bmls_parallel_bound`, with `Paper.Condensation` for the acyclicity of the condensation. Uncertified, and so not part of the statement: the weighted case, a plan whose condensation has a node of work $w > 1$, which enters by splitting that node into a chain of w unit steps (step (M) of Definition 6.4); that the split preserves T_1 and T_∞, and so that the bound holds for the weighted plan, is a text step. The existence of a topological numbering of a finite acyclic graph and the correctness of Tarjan's algorithm are cited. Since every task has work at least 1 and a component's work is the sum of its members' work, a condensation with work 1 at every node has one task in each component: the statement covers unit-work plans with no cycle through two or more tasks, and a plan with such a cycle falls under the uncertified weighted case. In words: the split into unit steps matters. Levelling whole nodes instead makes each level wait for its heaviest member, and the bound fails: two chains, of work 10 then 1 and of work 1 then 10, take 20 steps on two agents that way, against the bound's $11 + 11/2 = 16.5$.

Proposition 6.6 (Governed work-depth — the price of oversight is depth)**.** *Take the plan of Definition 6.1 and its condensation, the acyclic graph the scheduler levels. Each node of the condensation is charged its members' work in sequence, the sum, as Lagom Orkester runs it (Definition 6.4), and each ratification gate adds its latency $\lambda(v) \in \mathbb{N}$. Write D for the longest path under the work alone (the ungoverned depth), D_β for the longest path under work plus latency (the governed depth), and π^β for a path attaining D_β. Then:*

(i) $D \le D_\beta$, with $D_\beta = D$ when $\lambda \equiv 0$, and D_β is monotone in λ;

(ii) $D_\beta - D \le \sum_{v \in \pi^\beta} \lambda(v)$, with equality exactly when π^β is also a longest path under the work alone — that is, when no gate shifts which path is critical. In particular, if the ungoverned critical path π^* stays longest after gating, then $D_\beta = D + \sum_{v \in \pi^*} \lambda(v)$.

Status. Uncertified, and so not part of the statement: (iii) a greedy schedule on p uniform agents satisfies $D_\beta \le T_p \le D_\beta + T_1/p$; its proof below is Graham's argument in the text, with no certificate, as the header of Lean `Paper.GovernedWorkDepth` says. Parts (i) and (ii) are certified over any finite family of paths with natural-number work and latency; that this family is the set of paths of the condensation is the modelling step, not certified. The condensation is that of strongly connected components, computed by Tarjan's algorithm, which is cited. An elastic charge, a component's heaviest member, would apply only if the component's members could run side by side, which needs a cut of its cycles left for later work; it never exceeds the sum. *In words: turning oversight up never makes work shallower, it costs at most the latencies on the governed critical path, and it costs exactly those latencies unless a gate is heavy enough to make a shorter chain the critical one.*

For example, take two independent chains: A of two tasks of work 5 each, and B of two tasks of work 1 each, with one gate of latency 20 on B. Then $D = 10$ (along A) but $D_\beta = 22$ (along B), and the governed critical path is B. Here $D_\beta - D = 12$, strictly less than the 20 of latency on π^β: the gate has shifted criticality, which is why the equality in (ii) needs its condition.

Under the agnostic and aware modes ($\lambda \equiv 0$), $D_\beta = D$ and (iii) reads $D \le T_p \le D + T_1/p$, which is weaker than Proposition 6.5; for the Lagom Orkester schedule the sharper bound of Proposition 6.5 applies. If tiers are hard capability partitions, so that a task runs only on its own tier's agents, the uniform bound in (iii) need not hold; a greedy schedule then satisfies, by the text argument that follows and with no certificate, $T_p \le D_\beta + \sum_k W_k/p_k$, with W_k the work and p_k the agents of tier k. To see it, follow a chain back from the last task to finish, taking at each step a predecessor that finished last. At each instant the chain's current task is running, or its gate latency is elapsing, which together take at most the chain's governed length and so at most D_β; or it is ready but not started, and then its tier k has all p_k agents busy, which takes at

most W_k/p_k for that tier.

Proof. (i) Every path's governed length is its ungoverned length plus its latencies, which are non-negative, so the longest governed path is at least as long as the longest ungoverned one, and raising any latency raises no path's length by less than zero. (ii) Let π^β attain D_β. Its ungoverned length is at most D, so $D_\beta = \mathrm{work}(\pi^\beta) + \sum_{v\in\pi^\beta} \lambda(v) \le D + \sum_{v\in\pi^\beta} \lambda(v)$, with equality exactly when $\mathrm{work}(\pi^\beta) = D$. (iii) is the greedy argument of Graham with the gate latencies as delays on the precedence edges. Parts (i) and (ii) are proved in Lean over any finite family of paths (depth_le_governed, governed_eq_of_zero, governed_mono, governed_excess_le, governed_excess_eq_iff, with the example as example_shift); that the family is the set of paths of the condensed plan is the modelling step. Part (iii) is a proof in the text, with no certificate. □

The excess $D_\beta - D$ of Proposition 6.6 is the price of oversight, paid in depth. It is at most the latencies on the governed critical path, and exactly those on the ungoverned one when no gate moves the critical path. The operator's oversight dial, which classes of task need a human's ratification and which run autonomously, is the choice of mode in Definition 10.4 together with the gated classes: the modes γ and α set every latency to zero, and β raises it on the gated classes. So the one family of modes that governs whose law reaches an operation also governs how much depth its oversight costs, and a single dial trades assurance that the law can be enforced against parallel throughput. Tiers of model, which are informal here (Definition 6.1, Status), enter through the assignment. Minimising the total cost of assigning each unit to an adequate model tier, with cheap tiers on the wide low-stakes levels and strong tiers on the critical and heavily overseen units, is the cost companion to the bound on the makespan. Each gated dispatch has its authorisation logged to the σ-chain, as the governed schedule has it (Definition 6.1), so the ledger records who authorised it. The depth bound itself is standard critical-path analysis [KW59], with the approval latencies as node delays; what is added is the reading of the mode of governance as the dial.

Proposition 6.7 (Replanning cost bound)**.** *Let S be the finite set of super-nodes of a Lagom Orkester schedule (Definition 6.4, step (T)), each $s \in S$ with its work* $\mathrm{work}(s) \in \mathbb{N}$*, the sum of its members' work. If the work lost is that of one super-node $s \in S$, it is at most* $\max_{s'\in S} \mathrm{work}(s')$*; and with an overhead $c \in \mathbb{N}$ added to it,* $\mathrm{work}(s) + c \le \max_{s'\in S} \mathrm{work}(s') + c$.

Proof. Assume, as the statement does, that the work lost is that of one super-node s; its work is at most the largest super-node's (`replan_cost_bound`, `replan_cost_bound_with_overhead`, Lean `Paper.ReplanCost`). The full argument is in Appendix C. The premise, that a failure loses at most the in-flight super-node, is assumed, not proved. □

Status. Shallow: the bound is `Finset.le_sup` (`replan_cost_bound`), and with the overhead the same inequality with c added (`replan_cost_bound_with_overhead`). Uncertified, and so not part of the statement: that a failure loses at most the one super-node in flight, which is assumed, and which on $p > 1$ agents, where several super-nodes can be in flight at once, limits which failures are covered; that a replanning step at the boundary of super-node s_j keeps every completed level, and that the levels of the remaining plan are the old ones shifted down by j, so they need no recomputation; and that replanning therefore adds nothing to what a failure itself destroys. The overhead c is read as the cost of rescheduling the unfinished part of the plan. In the level-shift claim, j names both the super-node s_j and the size of the level shift; step (M) of Definition 6.4 splits a super-node over several levels, so the two readings may differ, and neither is certified.

6.3 Many agents on one plan

Definition 6.8 (Coverage and input-groundedness)**.** *Let* $\mathfrak{G} = (N, R;\ E, s, t, \tau, \alpha)$ *be the coverage graph (Definition 4.21), and let* $R_{live} \subseteq R$ *be the regions whose file and anchor exist in the current state. For* $n \in N$ *put* $\mathrm{owns}(n) = \{t(e) : s(e) = n,\ \tau(e) = \mathrm{Owns}\} \subseteq R$*;* $\mathfrak{G}$ *is* covering *iff* $R = \bigcup_n \mathrm{owns}(n)$*.* $\mathfrak{G}$ *satisfies the* live invariant *iff every region in some* $\mathrm{Hyd}(n)$ *(Definition 4.20) is the target of some* Tracks *edge.* $\mathfrak{G}$ *is* input-grounded *iff* $t(E_{\mathrm{Hydrates}} \cup E_{\mathrm{Tracks}}) \subseteq R_{live}$*.*

In words. Covering is the canonical-home goal: one lifecycle owner per region. The live invariant is hydrated $\subseteq$ *tracked. Groundedness is edge-kind-relative: inputs must resolve, whereas* Produces/Emits *targets are outputs — a regenerable output that is legitimately absent is an obligation to emit it again, not a dangling pointer.*

Status. Covering presupposes that Owns *edges are recorded, which they are not yet; the live invariant is the one the host is designed to enforce at present. When* $\mathfrak{G}$ *is input-grounded, each node's hydration set* $\mathrm{Hyd}(n)$ *is retrievable; it is the hydration part of the resume context* $\mathrm{Res}(n)$ *of Proposition 5.7, not by itself the* σ*-sufficient statistic of Proposition 4.51.*

One least-fixpoint engine computes the control decisions over the substrate. Coverage and node health are purely monotone least fixpoints, Datalog without negation: reachability, groundedness, and the meet-semilattice of health. The governance gate adds stratified negation. Its defeasible test that no undefeated attacker remains reads only completed lower strata, over an acyclic order of superiority, so it is monotone stratum by stratum and has a unique perfect model, by the standard results on stratified Datalog, which are not certified here. It extends the monotone engine rather than replacing it, and one seminaïve fixpoint machinery serves both.

Definition 6.9 (Collective act)**.** *Let* $\mathcal{S}^*$ *be the set of finite sets of* σ*-events (Definition 3.10), the* event sets*, ordered by inclusion, with join* $\sqcup$ *the union. For a finite set* R *of selves, either digital agents of one commons or commons of* $\mathbb{I}$*: each digital agent* v *contributes for its act* $a_v \in \mathcal{A} \cup \{0\}$ *(Definition 4.14) a finite set* $s_v \in \mathcal{S}^*$ *of* σ*-events recorded for* a_v*, with* $s_v = \emptyset$ *when* $a_v = 0$*, and a commons* k *contributes* $s_k = \bigcup_{v \in V^k} s_v$*, the union of its digital agents' event sets. The* collective act *of* R *is the join of the members' event sets,*

$$\mathrm{CollAct}(R) = \bigsqcup_{v \in R} s_v \in \mathcal{S}^*,$$

the least element of $\mathcal{S}^*$ *containing every* s_v*.*

Status. The members are meant to be co-resident selves, sharing one substrate. $\sqcup$ is idempotent, commutative, associative and monotone (Lemmas 4.52 and 4.53; proved in Lean in `IasCert.Paper.SigmaMerge` over merge $= s \cup t$); git-merge is meant to implement it, agreeing up to canonicalisation on the conflict-free fragment; that agreement is not certified. By Remark 14.7 each a_v is a want-to-act-to-be control; $\mathrm{CollAct}(R)$ carries that structure one scale up: the collective want is the join of the members' felt-lacks, the collective act the join of their repairs, the collective be the maintained Con(SOMA) of R. Selfhood can thereby be read as scale-recursive — agents compose into a commons through CollAct, and commons into a federation only as far as they exchange: what passes between commons is what each offers and takes (Proposition 8.8), so a federation acts as one self in this sense only when it is open.

Definition 6.10 (Co-walk)**.** *A* co-walk *is several digital agents and an operator of their commons (Definition 3.10) moving one plan (Definition 6.1) forward together, each through its own walk of the plan's runnable nodes, sharing one resource ledger. It has three parts.*

(i) Claims*. A digital agent takes a node by a claim with a time-to-live. While a claim is fresh, only its holder may take the node; once it is stale, any digital agent may.*

(ii) The ledger. *For each resource dimension j it records the capacity, a reservation r_j, the amount of that dimension protected for the operator's continuity, and the running jobs, each with an upper-bound estimate of its use and a deadline if it has one. Any entity, digital agent or operator, may post a request: defer one of its own jobs, redirect a job to another method, start a job, or yield a dimension it holds.*

(iii) Commits. *What the members commit is joined as their collective act (Definition 6.9).*

The ledger is feasible *when three clauses hold: no dimension j is committed beyond its capacity less r_j; no running job has passed its deadline; and every deferral names a running job.*

Status. The same rule applies to every entity. The third clause is there so that no job is parked forever. The reservation is the ledger's form of the protected residency of the reaper (Definition 6.17). Property tests check, on the executable model, that from a feasible ledger, posting a request keeps it feasible exactly when the request does not park a job forever, and that a fresh claim admits exactly its holder. They are tests, not proofs, and no Lean certificate exists. As defined, a feasible ledger has no contended dimension, so the jobs' predicted release times play no part. A form that judges contention by what a deferred or deadline-bound job will add when it runs, with important work given the full check, is left for later work.

Proposition 6.11 (Disjoint-writes criterion)**.** *Let a batch be a finite set of units of one level of a Lagom Orkester schedule (Definition 6.4, step (M)), and give each unit u a finite set $W(u)$ of resources, its* write-set*. Say that the batch has a* write-conflict *when two distinct units in it have write-sets that meet. Then the batch has no write-conflict exactly when every two distinct units in it have disjoint write-sets; and, when equality of resources is decidable, whether the batch has a write-conflict is decidable.*

Proof. A write-conflict is, by definition, two units at distinct positions of the batch that write a common resource; its negation is exactly pairwise disjointness of the write-sets, and pairwise disjointness of finite sets is decidable. Machine-checked (`disjoint_writes_correct`, Lean `Paper.DisjointWrites`). □

Status. Shallow: the equivalence unfolds the definition of write-conflict, by `Finset.disjoint_iff_inter_eq_empty` (`disjoint_writes_correct`), and the decidability is an instance for finite sets. The write-set and the write-conflict are defined in the statement, not in a definition of their own. In Lean the batch is a list and its units are taken at distinct positions. The order-freedom paragraph below is not part of the statement, and its certificate `Paper.OrderFreeCommit` is not a certificate of this proposition. *Freedom from write-conflict is not the same as order-freedom. When a unit computes what it writes from what it reads, the batch commits the same store in every order when, in addition, no unit reads what another writes: Bernstein's conditions (proved in Lean, `commute`, `commit_perm`, Lean `Paper.OrderFreeCommit`; for units that read nothing, disjoint write-sets suffice, `blind_commit_perm`). The read condition cannot be dropped: one unit reads x and writes $y := x + 1$ while another writes $x := 5$; the write-sets are disjoint, and from the all-zero store the two orders give $y = 6$ and $y = 1$ (`read_condition_needed`).*

Proposition 6.12 (Order-free commit)**.** *Let a store be a map from a set of cells to values, and let each task write a finite set of cells and read a finite set of cells, computing the values it writes from the cells it reads only; applying a task overwrites its write-set and leaves every other cell unchanged. Two tasks satisfy* Bernstein's conditions *when their write-sets are disjoint and neither reads a cell the other writes. (i) If every two tasks of a finite batch satisfy Bernstein's conditions, then committing the batch in any order gives the same store. (ii) For tasks that read no cell, pairwise disjoint write-sets suffice. (iii) Disjoint write-sets alone do not suffice: there are two tasks with disjoint write-sets whose two orders give different stores.*

Proof. (i) is `commit_perm`, by `commute` and `bernstein_symm`; (ii) is `blind_commit_perm`; (iii) is `read_condition_needed`: from the all-zero store, the tasks $y := x + 1$ and $x := 5$ have disjoint write-sets and give $y = 6$ or $y = 1$ according to the order. Machine-checked, on the standard axioms only (Lean `Paper.OrderFreeCommit`). □

Status. The disjoint-writes criterion (Proposition 6.11) decides when the write-sets are disjoint; this proposition says when that is enough for the commit order not to matter.

6.4 Sharing the metered channel

Definition 6.13 (Subscription queue)**.** *Digital agents that share one metered model subscription take turns on it through a* subscription queue*: a state holding at most one* holder *(a label and the time it took the slot), a list of* waiters *(each a label and the time it joined), the time of the last release, and a minimum gap between sessions. Its transitions are:*

(i) enqueue*: a label that neither holds the slot nor waits joins the end of the list; otherwise nothing changes;*

(ii) acquire*, requested either for a whole session or for one turn of a live session: the holder may re-acquire, which refreshes its time; anyone else is refused while the slot is held; a free slot is refused to a session request while the gap since the last release is running, and otherwise goes to the head of the list, or to anyone when the list is empty, except that a per-turn request may pass a head that has left the free slot untaken for longer than the gap, the head keeping its place;*

(iii) release*: only the holder may release; the slot becomes free, and the head of the list is the one to be offered it next;*

(iv) stale recovery*: a holder whose label has no daemon at the time, not in* active(t) *(Definitions 3.10 and 3.14), is removed, waiters older than a maximum age are dropped, except the head while the slot is free, which is being offered it; and a head that leaves a free slot untaken beyond the gap and a grace period is dropped, one per step.*

Proposition 6.14 (One holder, first come first served)**.** *For the subscription queue of Definition 6.13:*

(i) while the slot is held, every label but the holder is refused;

(ii) a session request takes a free slot only as the head of the waiting list, or when the list is empty; a per-turn request may also take it once the head's offer is older than the gap, and the passed head stays first in the list;

(iii) a successful acquire makes the acquirer the one holder;

(iv) release by anyone but the holder is refused, and release by the holder frees the slot and names the head of the list;

(v) stale recovery never creates a holder, keeps the surviving waiters in their order, and, while the slot is free, never drops the head by age;

(vi) every transition preserves well-formedness: no label waits twice, the holder does not wait, and the gap is not negative.

Proof. Each part is a case of the transition that (i)–(v) name: (i) and (iii) are the two branches of acquire on a held slot and a successful one, (ii) its branch on a free slot, (iv) the two branches of release, and (v) holds because the three recovery steps only remove the holder or remove waiters, and the age step passes over the head of a free slot. (vi) is checked transition by transition. All six are proved in Lean for any label type with decidable equality and integer times (`acquire_occupied`, `acquire_fifo`,

`acquire_fifo_session`, `perturn_pass_keeps_head`, `acquire_holds`, `release_nonholder`, `release_frees`, `clearStale_no_holder`, `clearStale_sublist`, `dropOld_keeps_head`, `wf_enqueue`, `wf_acquire`, `wf_release`, `wf_clearStale`; Lean `Paper.SubscriptionQueue`). How long a waiter waits is not proved: property tests on the executable model check that the k-th waiter is served after k handovers, and that a queue all of whose waiters have stopped drains within $(k+1)$ times the gap plus grace. □

In words: the queue changes who runs when, not how much work there is, so the total work is still what Brent's bound (Proposition 6.5) counts. The waiting it adds is what the property tests measure; that every waiter is eventually served is not claimed. A live head slower than the gap plus the grace is dropped, and a launched head slower than the gap can be passed by a live prompt, though it keeps its place.

Status. Shallow: (i), (iii) and (iv) read back the branches of acquire and release of Definition 6.13 case by case.

6.5 Resources and the reaper

Definition 6.15 (Resource envelope and overrun)**.** *Let the resource coordinate live in a finite product $R = \prod_j R_j$ — one coordinate per accountable resource (wall-time, memory, input/output, compute-share) — with fixed finite capacity $C \in R$. A live process carries a* class *and an* envelope *$E \in R$, its declared expected resource use, and may carry an* owner*, an accountable principal; a live process with no identified owner is an* orphan*. Against a sampled actual use $A \in R$ and a per-coordinate tolerance tuple $k = (k_j)_j$ with each $k_j \geq 1$, the* overrun *relation*

$$A \sqsupset E \iff \exists j.\ A_j > k_j\, E_j$$

is defined coordinatewise.

In words. On the finite shared substrate of Axiom 3.1, resource residency is a scarce good whose governance is a sub-problem of the controlled substrate chain. The envelope is a contract and a residency-admission bound, not a hard ceiling enforced by the substrate alone. The overrun relation is decidable coordinatewise. The resource envelope is the controlled-Markov-chain (CMC) resource-state component attached to each live process; an overrun event is the trigger that hands the constrained reaper (Definition 6.17 below) its input. A concrete instance: a subprocess with envelope $E = (30\,\text{s}, 256\,\text{MB}, \ldots)$ and uniform tolerance $k_j = 1$ overruns on the first coordinate whose sampled use exceeds its contracted value — the check is coordinatewise and local, never global.

Status. Axiom-0 (Axiom 3.1) posits finite carriers, not resources; the product R is a modelling choice on the finite substrate it describes.

Definition 6.16 (Resource budget)**.** *A* resource budget *is a tuple* $(\text{compute}, \text{memory}, \text{disk}, \text{energy}, \text{net}) \in \mathbb{N}^5$*, the amount of each resource a node may use.*

Status. Axiom-0 (Axiom 3.1) posits finite carriers, not these resources; the budget is a modelling choice at the verification layer, motivated by it: what a node affords bounds what it can check.

Definition 6.17 (The constrained reaper as CMC control policy)**.** *The* reaper *is the control policy of the controlled substrate chain (Definition 4.15) for the resource-governance sub-problem on the resource coordinate R of Definition 6.15. Partition the live residency into three mutually exclusive classes, by a declared predicate that marks residency protected:* protected *(residency so marked),* accounted *(owned residency that is not protected, within or over its contract), and* reapable *(residency that is neither protected nor owned: the unprotected orphans of Definition 6.15); write* $\text{class}_{\text{reap}}(p)$ *for the class of the residency of a sampled process p. Its decision is a total function*

decide *from a sampled process to a verdict in* {leave, remind-owner, suspend, reclaim-candidate}*: within contract* ↦ leave*; over contract and (protected or accounted)* ↦ remind-owner*; over contract, reapable, and marked suspendable by a declared predicate on processes* ↦ suspend*; over contract, reapable, and not suspendable* ↦ reclaim-candidate*. Its* removal domain dom(reap) *is the set of sampled processes whose verdict is* reclaim-candidate*, the only processes it may remove.*

Status. The protected residency is meant to be the residency on which a protected invariant rests. The verdict suspend is the hibernation of the process, and the flag is its hibernatable flag. Operator-continuity residency is the canonical instance of protected residency. By the case split of decide, dom(reap) ⊆ reapable, equivalently protected ∩ dom(reap) = ∅. The reaper is meant to be read as a constrained optimiser, reducing reapable residency subject to the hard constraint that protected terms are held invariant; no objective is defined here. Hibernation suspends rather than kills, and is loss-free by Proposition 5.4 for what the process has σ-retained; only the flag leads to it, since a reapable process has no identified owner to declare it; the reaper never escalates from hibernation to reclamation on its own. In words, the reaper is a constrained, not a free, optimiser: it carries a protected side-condition it may never trade against residency, which is precisely the content of Proposition 6.18 below. The decision function is total — every sampled process receives a verdict, with no partial call and no silent case. A protected process with no owner that overruns also receives remind-owner; who receives that reminder is not fixed here.

Proposition 6.18 (Protected-term safety)**.** *The reaper (Definition 6.17) never reclaims a protected term: for every sampled process p,*

$$\mathrm{class}_{\mathrm{reap}}(p) = \mathrm{protected} \;\Rightarrow\; \mathrm{decide}(p) \neq \textsf{reclaim-candidate},$$

and it never suspends one either. Hence protected ∩ dom(reap) = ∅ *on every input.*

Proof. The decision is a total case analysis on the process: within its envelope it is left alone; over its envelope, a protected or accounted process only produces a reminder to its owner; only an unaccounted process over its envelope can be suspended or become a reclaim candidate. So a protected process reaches neither of the last two cases. Machine-checked, on the standard axioms only (`never_reaps_protected`, `never_hibernates_protected`, `within_contract_left`, `reap_implies_reapable_and_over`, Lean `Paper.Reaper`). □

In words, the clean-up process is a constrained optimiser, not a free one: it minimises wasted residency but carries a side-condition it may never violate. An orphaned runaway with no owner, past its tolerance, is a reclaim candidate; an operator-critical process past its tolerance is not reclaimed, and its owner is reminded instead.

Status. Shallow: the conclusion is the case table of decide in Definition 6.17, read back (`never_reaps_protected`, `never_hibernates_protected`, by cases on the over-contract test). In the Lean names the verdict suspend is `Hibernate` (`never_hibernates_protected`, `hibernate_implies_reapable_and_over`).

Remark 6.19 (Resource accountability as substrate-layer human-in-commons)**.** *On overrun (Definition 6.15) of an owned process, protected or accounted, the reaper (Definition 6.17) returns control over the process to its owner, with the verdict* remind-owner*, rather than acting on it; an overrunning orphan, which has no owner, is suspended when it is marked suspendable and otherwise becomes a reclaim candidate. For an owned process this is the CMC control-return: the chain's next transition on the resource coordinate is determined by the owner's choice, not by a unilateral reclaim of the reaper. It prefigures the human-in-commons axiom (Axiom 3.9) one layer below the operation: there, LLM-free-ability makes every operation effecting the commons operator-performable (HIC.1); here, resource-accountability returns every owned overrunning*

process to its owner. The two can be read as one predicate — return of control to an accountable principal on demand — instantiated at two layers, and the reminder-loop as the residency analogue of the LLM-free-ability of HIC.1 (Axiom 3.9). In words, on overrun of an owned process the reaper does not act on its own; it hands the decision to the owner. The Concage envelope (Remark 10.17) is the governance-weight twin of this resource-layer control-return: both are envelopes that return control to an accountable principal, one for resource decisions, one for specification decisions.

6.6 Related work and intellectual lineage

The control kernel, a typed map over (state, action) pairs whose domain admits only feasible actions, is the Markov-chain reading of [KS60], extended to a controlled chain, with the capability gate of the object-capability discipline placed before the draw [DH66, SS75, Mil06]. The governed schedule and its depth bound inherit the Brent–Graham work–depth tradition [Bre74, Gra66, Ble96] and the Dilworth–Mirsky antichain decomposition [Dil50, Mir71], with the scheduling literature of [Pin16] for the latency-augmented longest path, which is critical-path analysis [KW59] with the approval latencies as node delays.

7 Security and permissioning: one algebra, one decision point

Planning decides what should run; this section decides what may. It gives one algebra and then makes it real. *The capability lattice and the deny tier* computes what a plan may use and removes what must never be allowed; a gate $g \neq \bot$ placed in that tier cannot be lifted by any action admissible under any taskset (Proposition 7.5), provided, as a fact of deployment, that it is configured only through capability-checked actions. *The four facets* apply the same algebra to admission, egress, confinement and heredity. *Who is acting* ties an act to its actor. *One decision point for every call* is where the algebra is designed to be enforced, and why that point cannot deadlock. *Making the boundary real* states what we assume of the operating system beneath it, and *detecting and surviving attack* what happens when an attacker tries anyway, ending with an application: detecting a vendor's undisclosed watermark without its key.

7.1 Why this matters to the engineer

The question an engineer asks of any digital agent is what it can do, read, leak or pass on, and whether it can route around the answer. IAS answers once, algebraically, and is designed to enforce the answer at one point; on-host enforcement is exercised in shadow mode only. Capabilities form a lattice. The capabilities a plan's tasks need are computed from the operator-attested plan, and the computation is the left adjoint of a Galois connection (Proposition 7.2); that the decision point enforces exactly them rests on the hypotheses of Proposition 7.14, and a sovereign deny tier removes what must never be allowed, as a relative complement inside the same lattice (Definition 4.13). The same algebra bounds what may leave the boundary as well as what may be done inside it (Proposition 7.8). It is designed to be enforced at one decision point that every trapped tool call passes through (Proposition 7.14); that every tool call of a digital agent is trapped is the condition of complete mediation (Definition 7.13). The health gate of that decision point cannot deadlock (Proposition 7.15).

An algebra is worth little if the agent can step around the check. The second half of the section is about making the boundary real: a confinement predicate, designed to be installed before the agent's first instruction, which in the model bounds the damage of any breach within the modelled operations (Proposition 7.9); a scan that detects covert characters (Proposition 7.25); and a plain statement of what we assume of the operating-system kernel. The operating system is part of the self's substrate, so its integrity is part of what must stay consistent.

7.2 The capability lattice and the deny tier

Definition 7.1 (IPLiKiDi projection)**.** *Let $G = (V_T, E_T)$ be the plan of tasks (Definition 6.1), cycles allowed. In each substrate state the capability atoms form a finite set* At*, the concrete capability paths of that state. Let* $(\mathrm{Cap}, \subseteq)$ *be the capability lattice of the state: the subsets of* At*, ordered by inclusion, with join the union and meet the intersection. An allowlist pattern, wildcards included, stands for the set of atoms of* At *it matches. The* IPLiKiDi projection *is the map*

$$\varphi : 2^{V_T} \to \mathrm{Cap}, \qquad \varphi(T) \;:=\; \bigcup_{t \in T} \mathrm{allowlist}(t),$$

where $\mathrm{allowlist}(t) \subseteq \mathrm{At}$ *is the set of atoms matched by the allowlist patterns of task t. The* active taskset *at time τ is the set* $\text{active-taskset}(\tau) \subseteq V_T$ *of the tasks running at τ.*

In words. $\varphi(T)$ is the join in Cap of the tasks' allowlists, with no further closure. The projection sends an active taskset $T \subseteq V_T$ to the join of its tasks' allowlists. That this is the minimal sufficient capability scope of T, and that it is the authorization the reference monitor admits when T is evaluated, is meant and not certified (Propositions 7.2 and 7.3, Status). Lagom Orkester (Definition 6.4) schedules the plan by condensing its cycles (Lemma 6.2) and levelling the acyclic result, but the projection is defined on any set of tasks, including tasks inside a cycle, because a permission applies to whichever task is running. The field of atoms is open-ended across states, since growth adds paths.

Proposition 7.2 (Galois connection between active-tasksets and the capability lattice)**.** *Assume the plan has finitely many tasks (Definition 7.1). Define the right adjoint* $\varphi^{\dashv} : \mathrm{Cap} \to 2^{V_T}$ *by* $\varphi^{\dashv}(C) := \{t \in V_T : \mathrm{allowlist}(t) \subseteq C\}$*. Then the pair* $(\varphi, \varphi^{\dashv})$ *is a* Galois connection *between* $(2^{V_T}, \subseteq)$ *and* $(\mathrm{Cap}, \subseteq)$*: for every $T \in 2^{V_T}$ and every $C \in \mathrm{Cap}$,*

$$\varphi(T) \subseteq C \iff T \subseteq \varphi^{\dashv}(C).$$

In particular $T \subseteq \varphi^{\dashv}(\varphi(T))$ *and* $\varphi(\varphi^{\dashv}(\varphi(T))) = \varphi(T)$ *for every $T \in 2^{V_T}$.*

Proof. Direct from Definition 7.1. $\varphi(T) \subseteq C \iff \bigcup_{t \in T} \mathrm{allowlist}(t) \subseteq C \iff \forall t \in T : \mathrm{allowlist}(t) \subseteq C \iff T \subseteq \varphi^{\dashv}(C)$. The bi-implication is the Galois-connection axiom. For any Galois connection the composite on the domain side is a closure and the composite on the codomain side is a kernel, since $T \subseteq \varphi^{\dashv}(\varphi(T))$ and $\varphi(\varphi^{\dashv}(C)) \subseteq C$ are the two unit inequalities. The connection, the unit inequality $T \subseteq \varphi^{\dashv}(\varphi(T))$ and the identity $\varphi \circ \varphi^{\dashv} \circ \varphi = \varphi$ are machine-checked for exactly this φ, the join of the allowlists with no further closure over a finite plan, depending on none of the framework's axioms (`iplikidi_galois`, `le_adj_phi`, `phi_adj_phi`, Lean `Paper.GaloisBoundaries`); the closure and kernel properties are the standard consequences of any Galois connection. □

Status. The connection is the monotone (residuated) form of a Galois connection; Ore (1944) and Birkhoff (1940) give the antitone form. That $\varphi^{\dashv} \circ \varphi$ is a closure operator on 2^{V_T} (idempotent, monotone, extensive) and $\varphi \circ \varphi^{\dashv}$ a kernel operator on Cap (idempotent, monotone, deflationary) are the standard consequences of any Galois connection and are not on the Lean line. The fixed points of these closures are read as the projection's* polar pairs*, the framework's minimal-and-sufficient capability scopes (after Cousot–Cousot 1977 abstract-interpretation lattice-morphism theory); that reading is uncertified.

Proposition 7.3 (Enforced boundary + Galois connection under the deny-tier)**.** *For a plan with finitely many tasks, write* $\varphi_{\mathrm{enf}}(T) := \varphi(T) \setminus \mathrm{Deny}$ *for a taskset $T \subseteq V_T$ (Definition 7.1, Definition 4.13). Then the enriched projection* $\varphi_{\mathrm{enf}} = (\,\cdot \setminus \mathrm{Deny}) \circ \varphi$ *is the left adjoint of a Galois connection with right adjoint*

$$\varphi_{\mathrm{enf}}^{\dashv}(C) \;=\; \varphi^{\dashv}(C \cup \mathrm{Deny}) \;=\; \{\, t \in V_T : \mathrm{allowlist}(t) \subseteq C \cup \mathrm{Deny} \,\}.$$

Proof. $\varphi_{\text{enf}}(T) \subseteq C \iff \varphi(T) \setminus \text{Deny} \subseteq C \iff \varphi(T) \subseteq C \cup \text{Deny}$ (generalized-Boolean residuation $a \setminus d \subseteq c \iff a \subseteq c \cup d$) $\iff T \subseteq \varphi^{\dashv}(C \cup \text{Deny})$ (the original connection, Proposition 7.2). Structurally, φ_{enf} is a *composite of left adjoints* — $(\cdot \setminus \text{Deny}) \dashv (\cdot \cup \text{Deny})$ (residuation) and $\varphi \dashv \varphi^{\dashv}$ (given) — hence itself a left adjoint with right adjoint $\varphi^{\dashv} \circ (\cdot \cup \text{Deny})$; and $(\cdot \setminus \text{Deny})$ preserves joins ($(a \vee b) \setminus d = (a \setminus d) \vee (b \setminus d)$ by distributivity), discharging the left-adjoint side-condition. Machine-checked, depending on none of the framework's axioms (`enforced_galois`, Lean `Paper.GaloisBoundaries`). □

Status. That $\varphi_{\text{enf}}(\text{active-taskset}(t))$ is the allow-list the reference monitor enforces at σ-event-time t is uncertified: the Lean defines φ_{enf} (`phiEnf`) and proves the connection only, and at the gateway a capability of the governance or the external class in it is held until approved (Proposition 7.14).

Remark (least-privilege modulo Deny; uncertified). The fixed points of the kernel $\varphi_{\text{enf}} \circ \varphi_{\text{enf}}^{\dashv}$ on Cap are the capability scopes some taskset needs exactly, *with the sovereign denies removed* — least privilege = least grants minus sovereign denies. It places memory inviolability (Proposition 3.19), a proposition under its own conditions, among the terms the boundary is designed to enforce, to the extent that the writes it excludes are in Deny, and on the conditions that every tool call of a digital agent is trapped, which is complete mediation (Definition 7.13), and that the gateway passes each call on faithfully (Proposition 7.14). In the boundary-act vocabulary of Definition 4.14, this *enforced boundary* is the **guard** act — admission-enforcement at $\partial\mathfrak{S}$ — of which the φ_{enf} Galois connection above is the *adjoint reading.*

Definition 7.4 (Non-self-liftable gate)**.** *In the capability lattice of Definition 7.1, an action with capability a is* admissible *under a taskset T when $a \leq \varphi(T) \setminus \text{Deny}$. A gate, such as the commit gate or the memory gate, is configured by actions that need its capability g, that is, actions with $g \leq a$. The gate is* non-self-liftable *when no action admissible under any taskset has $g \leq a$: no admissible action reconfigures it, whatever the taskset.*

Proposition 7.5 (Gates in the deny tier cannot be lifted)**.** *For a plan with finitely many tasks and a gate capability g (Definition 7.4):*

(i) some action admissible under a taskset T has $g \leq a$ exactly when $g \leq \varphi(T) \setminus \text{Deny}$;

(ii) if $g \neq \bot$ and $g \leq \text{Deny}$, the gate is non-self-liftable;

(iii) if g is not below $\varphi(V_T)$, the capability scope of the set of all tasks, the gate is non-self-liftable even with an empty deny tier.

Proof. (i) An admissible action with $g \leq a$ gives $g \leq a \leq \varphi(T) \setminus \text{Deny}$; conversely $a = \varphi(T) \setminus \text{Deny}$ is itself admissible. (ii) Anything below $\varphi(T) \setminus \text{Deny}$ is disjoint from Deny, and a non-zero $g \leq \text{Deny}$ is not. (iii) φ is monotone, so $\varphi(T) \leq \varphi(V_T)$ for every T. All three are proved in Lean in the generalized Boolean algebra of Proposition 7.3 (`liftable_iff`, `nonSelfLiftable_of_deny`, `nonSelfLiftable_of_outside`; Lean `Paper.NonSelfLiftable`), and a property test on the executable capability algebra checks (ii) on random tasksets and deny sets. The proposition is about the algebra. That a gate is in fact configured only through capability-checked actions, for example that a commit gate runs on a server the agent cannot write to, is a fact of deployment, not a theorem. □

Status. Shallow: (i) follows from the definition of admissibility by transitivity of $\leq$ (`liftable_iff`).

Remark 7.6 (Facet-jumps: a family of moves, each with its own substrate measure)**.** *The want-to-act-to-be capstone (Remark 14.7) invites a* family *of value-measures, one per facet of* Con(SOMA)*, rather than a single privileged one. Read a* facet-jump *of family F as a class of*

acts (Definition 4.14) advancing one facet — intellectual, thermoeconomic, communicational, and so on; the additive and ablative moves in the lattice of roots and conditions (adding, or dropping, a condition no result needed — the certified-independence ablation of the Appendix) are the intellectual *family. A* jump-measure *for F is any Σ-measurable map $\mu_F : \mathbb{X} \to \mathcal{V}_F$ from the substrate state to an ordered value space appropriate to F: intellectual $\mapsto$ the count of certified propositions; research-leverage $\mapsto$ the out-degree of an operator-input node in the trajectory-spine's* amortization *sub-graph (an operator input linked to each downstream work node it generates until the next input); thermoeconomic $\mapsto$ delivered negentropy (Remark 3.4). Two disciplines keep the measures honest.* Anti-gaming is the measure-choice, not a patch: *since μ_F is a free map, the admissible ones are* constant on no-op fibres *: those that factor as $\mu_F = g \circ \pi$ through a content-projection π that quotients out no-ops, which for a standard Borel $\mathcal{V}_F$ is, by the Doob–Dynkin factorisation [Kal02], the same as $\sigma(\pi)$-measurability of μ_F; so a null action (a bare idle self-loop) stays in the fibre of the state it acts on and adds nothing under* any *admissible μ; admissibility is a condition we impose on the choice of measure, so degenerate measures (a raw σ-event count) are excluded by that choice, not filtered after the fact.* Revenue externalises: *an economic-default measure — raw revenue — is flagged, not adopted, since market pricing externalises the thermoeconomic real cost, so a cost-internalising negentropic measure is preferred over a defective one. Jump-freeness holds throughout: the carriers are the family index F, the measure μ_F, and the move in that lattice; "jump" is a discussion gloss, never a load-bearing primitive.* □

7.3 The four facets: admission, egress, confinement, heredity

Definition 7.7 (Egress-confinement: the deny-tier on the context lattice)**.** *Let* $(\mathrm{Cap}_{\mathrm{ctx}}, \subseteq, \cup, \cap, \emptyset)$ *be the generalized-Boolean algebra of* context-grants *— the substrate slices (memory-slices, files, prior σ-results) admissible into an LLM-interaction sub-process's prompt — and let the* context-hydration projection $\varphi_{\mathrm{ctx}} : 2^{V_T} \to \mathrm{Cap}_{\mathrm{ctx}}$ *send a taskset $T \subseteq V_T$ (an active taskset of Definition 7.1) to the union of its nodes' declared* required-context *(the hydration scope). Let each LLM-interaction sub-process (Definition 4.9) carry a* trust tier*, the tiers ordered by egress-exposure,*

$$\text{sovereign-local} \sqsubseteq \text{sovereign-hosted} \sqsubseteq \text{vendor-proprietary} \sqsubseteq \text{third-party-intermediary},$$

and to each tier τ assign a sensitive-deny $\mathrm{Deny}_{\mathrm{sens}}(\tau) \in \mathrm{Cap}_{\mathrm{ctx}}$ *— the context that must not egress to a sub-process at τ — monotone in exposure,* $\mathrm{Deny}_{\mathrm{sens}}(\textit{sovereign-local}) = \emptyset \subseteq \cdots \subseteq \mathrm{Deny}_{\mathrm{sens}}(\textit{third-party-intermediary})$*. The* egress-confined hydration scope *of a sub-process at tier τ executing taskset T is*

$$\varphi^{\mathrm{enf}}_{\mathrm{ctx}}(T, \tau) \;:=\; \varphi_{\mathrm{ctx}}(T) \setminus \mathrm{Deny}_{\mathrm{sens}}(\tau),$$

the relative complement of Definition 4.13 on $\mathrm{Cap}_{\mathrm{ctx}}$*. Let* $\mathrm{ctx}_{\min}(T) \subseteq \varphi_{\mathrm{ctx}}(T)$ *be the minimal context under which T is executable; tier τ is* egress-sufficient *for T iff the confined scope still covers the need,* $\varphi_{\mathrm{ctx}}(T) \setminus \mathrm{Deny}_{\mathrm{sens}}(\tau) \supseteq \mathrm{ctx}_{\min}(T)$*, and the* routing rule *sends T to the $\sqsubseteq$-greatest egress-sufficient tier, which exists because sovereign-local, with* $\mathrm{Deny}_{\mathrm{sens}} = \emptyset$ *and* $\mathrm{ctx}_{\min}(T) \subseteq \varphi_{\mathrm{ctx}}(T)$*, is always egress-sufficient.*

Implementation. The required-context scope is a per-node required-context declaration; each tier carries a flag for whether the vendor trains on prompts, and a deny-set, declared in the routing configuration of the fleet and read by its router. A non-sovereign sub-process's readable context is its task-hydration with the tier-forbidden slices subtracted before the prompt is built.

Status. This is the substrate's "never outsource data-read governance" made a boundary term designed to be enforced — the model-egress analogue of the memory-inviolability seal (Proposition 3.19) that Proposition 7.3 makes first-class on the action side. In the boundary-act vocabulary of Definition 4.14 this is the *egress* act — the leak-gate governing what of the self

may cross to an *external* (non-progeny) party such as an inference vendor; Con(SOMA) is meant to admit no unauthorized egress; that is not a result here.

Proposition 7.8 (Egress-confinement is a Galois connection)**.** *For each tier τ (Definition 7.7), $\varphi^{\mathrm{enf}}_{\mathrm{ctx}}(\cdot,\tau) = (\,\cdot \setminus \mathrm{Deny}_{\mathrm{sens}}(\tau)) \circ \varphi_{\mathrm{ctx}}$ is a left adjoint, with right adjoint $C \mapsto \varphi^{\dashv}_{\mathrm{ctx}}(C \cup \mathrm{Deny}_{\mathrm{sens}}(\tau))$.*

Proof. The statement is Proposition 7.3 verbatim on $(\mathrm{Cap}_{\mathrm{ctx}}, \mathrm{Deny}_{\mathrm{sens}}(\tau))$ — the residuation $a \setminus d \subseteq c \iff a \subseteq c \cup d$ and join-preservation of $(\,\cdot \setminus \mathrm{Deny}_{\mathrm{sens}}(\tau))$ hold in any generalized-Boolean algebra (`egress_galois`, Lean `Paper.GaloisBoundaries`). □

Status. The statement is certified (`egress_galois`); the routing clause below is uncertified, with a text proof and no Lean: for egress-sufficiency and the routing rule of Definition 7.7, egress-sufficiency is downward-closed in the tier order, and sovereign-local, which denies nothing, is always sufficient; so the sufficient tiers form a non-empty initial segment of the tier chain, the routing rule is defined for every T, and if no tier above sovereign-local is sufficient, the sensitive slices being load-bearing for T, then T is routed to sovereign-local. The text proof: by monotonicity of $\mathrm{Deny}_{\mathrm{sens}}$, $\tau \sqsubseteq \tau' \Rightarrow \mathrm{Deny}_{\mathrm{sens}}(\tau) \subseteq \mathrm{Deny}_{\mathrm{sens}}(\tau') \Rightarrow \varphi^{\mathrm{enf}}_{\mathrm{ctx}}(T,\tau) \supseteq \varphi^{\mathrm{enf}}_{\mathrm{ctx}}(T,\tau')$, so egress-sufficiency is downward-closed in the tier order; since the tiers form a chain and $\mathrm{Deny}_{\mathrm{sens}}(\text{sovereign-local}) = \emptyset$ makes the least tier sufficient, the sufficient tiers are a non-empty initial segment and the $\sqsubseteq$-greatest of them is well-defined. The routed tier is the least sovereign sufficient tier, and we take it to be the cheapest. The routing clause is the egress-boundary analogue of the containment-sufficiency of Proposition 7.17: the scope survives the deny-subtraction and still suffices.

Remark (one algebra, two boundaries, audited once). Definitions 4.13 and 7.7 are *one* relative complement $\setminus$ instantiated on two lattices: Cap (action — realised into Landlock prefix policy by the kernel realisation ϱ of Proposition 7.17) and $\mathrm{Cap}_{\mathrm{ctx}}$ (egress — realised by the fleet's prompt-builder/trust-router). The former governs *what a worker may do*; the latter, *what a worker may see, hence leak* — with routing as egress-sufficiency (the uncertified routing clause of the Status above). In the boundary-act vocabulary of Definition 4.14, the *do* boundary is the **guard** act (action-admission) and the *see* boundary is the **egress** act (what may be read-in and hence leaked to an external party): these two are act-*directions within the informational-protected boundary*, and must not be conflated with the informational-vs-thermoeconomic two-boundary split of Remark 3.4. The sovereign deny is authored and audited *once* (the deny-tier of Definition 4.13); both boundaries are designed to enforce it. This is also a safety-condition that the *vendor-swap* continuity of Conjecture 5.10 (transition (b)) would need: a receiving vendor worker may carry an in-flight continuation only if its egress-confined scope $\varphi^{\mathrm{enf}}_{\mathrm{ctx}}$ covers the continuation's context envelope — exactly as enforcement-continuity there requires the receiving confinement class to cover the capability envelope.

Proposition 7.9 (Confinement bounds the blast-radius of any breach through modelled operations — known or unknown)**.** *The containment class of Definition 7.16 is meant to enforce by structure, not by knowledge: the kernel is to refuse any operation whose capability footprint meets* Deny*, whatever the confined agent intends and whether or not the attacker's technique is known. In the operating-system-subset model, given a single bootstrap-confinement predicate, no sequence of modelled operations lifts the confinement; "a root exploit simply disables the sandbox" fails for an exploit that works through modelled operations, because disabling the sandbox would itself be an operation the kernel refuses, while an exploit through an unmodelled path is case (ii) below. Split a breach by where its damage lands. (i) If the damaging operations lie within the modelled system-call subset, the damage is bounded to the confinement cell, over every admissible sequence. (ii) If the breach proceeds wholly through an unmodelled path (an unusual device control call, a driver bug, a hardware fault), it lies in the residual trusted base (Remark 7.18), outside what the model certifies.*

Proof. (i) is the operating-system-model aggregate: one bootstrap-confinement predicate discharges all five component antecedents and yields system security for every admissible sequence of modelled operations, with no hypothesis about any particular threat (`unified_system_security`, Lean module `IasCert.OSSubset.CarrierUnified`, namespace `IasCert.OSSubset.Unified`, zero domain axioms, with the model's seven seams as named assumptions, not proved). (ii) is a statement of scope, not a theorem: the model covers a named boundary of about eight filesystem and network operations, the guard act of Definition 4.14 at the confinement cell, and says nothing about operations outside it. □

Status. Certified in Lean in part: part (i) rests on `unified_system_security` (Lean module `IasCert.OSSubset.CarrierUnified`, namespace `IasCert.OSSubset.Unified`), in the operating-system-subset model only, with the model's seven seams recorded as named assumptions, not proved. That the kernel refuses what meets Deny does not follow from Proposition 7.3, which is an adjunction; and the Landlock realisation is exercised in shadow mode only, and live enforcement is not yet deployed (Proposition 7.17).

Definition 7.10 (Heredity choice — the transmission act over bit-sequences of the core reduct)**.** *The* heredity choice η *is a map of transmission, outbound to one's own offspring (Definition 9.8); each transmission it prescribes is an act (Definition 4.14). The heritable unit is a* bit-sequence *(the bit-floor carrier* Σ_{byte} *of Definition 3.3):* η *assigns to each heritable bit-sequence* b *of the* core reduct $\mathfrak{S} \restriction \Sigma_{\text{core}}$ *of the substrate structure* $\mathfrak{S}$*, with* $\Sigma_{\text{core}} := \Sigma_\sigma \cup \Sigma_{\text{byte}} \cup \Sigma_{\text{immune}}$*, a value in* $\{\textit{pass}, \textit{withhold}, \textit{initialise-fresh}\} \times \{\textit{private}, \textit{public}\}$*: whether* b *is passed, and by which channel, private (the canonical private-memory store, passed to a chosen offspring) or public (the git genome).*

In words. Σ_{core} is the *heritable* OS-invariant self, defined *positively* as identity ∪ content ∪ defence-genome, and *not* the parent's runtime Σ_Θ, health, or plan, which offspring do not inherit. Through the private channel, heredity and private memory are one object.

Status. η is meant to be *primary*, with the reproduction operators of Definition 9.8 read through it: the inheritance commit κ, a commit of $\mathbb{S}^c$, fixes the snapshot from which bit-sequences are taken, η fixes which of them pass, are withheld or are initialised afresh, and the reproduction procedure ρ is meant to be η applied across the parent set. No such factorisation through η is defined here. For recombination ($|\mathcal{P}| \geq 2$) η is a *chosen* combination; the take-the-union special case $\eta = \bigsqcup$ (progeny inherit the union of the parents' defence bit-sequences) is the *reproduction-level* join, to be distinguished from the *population-level* herd join (the union over all live members of an across-commons peer-set, a descriptive population statement). Under the coevolution quorum (Definition 8.17), a named condition and not a consequence of Axiom 3.9, η is quorum-attested; it is operator-co-curated, and no rule forces which bit-sequences pass: heredity is a sovereign act (Remark 14.6).

7.4 Who is acting

Remark 7.11 (Two-witness identity)**.** *Identity attestation for a canonical label* λ *at time* t *decomposes into two independent witnesses, neither sufficient alone:*

(i) a channel witness *— possession of a valid session credential* κ *(Definition 5.9) with* $\lambda_\kappa = \lambda$ *and* $t \in [t_0, t_1)$*, evidenced per-action by an unconsumed action receipt (Definition 4.12); and*

(ii) a narrative witness *— continuity of the* σ*-chain for* λ *(Proposition 4.51): under* σ*-journaling and the read-vocabulary hypotheses (R) and (S) of that proposition, the* σ*-chain is* $\mathcal{D}_{\text{IAS}}$*-sufficient for* λ*'s decisions, which is decision-sufficiency and not a reconstruction of* λ*'s state.*

Neither witness alone suffices. A credential possessed within its validity window by a foreign realisation (a stolen κ*) passes (i) whether or not its acts continue* λ*'s* σ*-chain, which only (ii)*

checks; a faithful σ-chain reconstruction lacking the credential passes (ii) without the authorization of (i). Hence "the credential authenticates the channel; the σ-chain authenticates the history" is a two-witness conjunction, neither witness authenticating the realisation itself, and cross-vendor / cross-host continuity is constructive only if both witnesses hand over at the seam.

Argument. Independence is exhibited by the two one-sided counterexamples (stolen-credential; credential-less reconstruction), each realising exactly one witness. That the conjunction suffices for admission is meant to follow from the composition of the entity-admission act ϕ_c of Definition 4.35 (which the credential of Definition 5.9 witnesses operationally) with σ-sufficiency (Proposition 4.51); the affine-receipt admissibility of Definition 4.12 supplies the per-action freshness that upgrades credential-possession from a session-level to an action-level channel witness. □

7.5 One decision point for every call

Definition 7.12 (Decision point)**.** *A* gateway pair *traps a tool call twice, once before and once after it runs, and carries no policy of its own: it hands the request to one* decision point*, which runs every rule and returns one* verdict*. A verdict is a decision together with a list of obligations, duties the caller must discharge. The decisions are ordered*

$$\mathsf{allow} < \mathsf{warn} < \mathsf{hold} < \mathsf{deny},$$

where hold *routes the call to an operator of the commons (Definition 3.11) for approval. The* governance class *and the* external class *are declared sets of capabilities, subsets of the capability lattice* Cap *(Definition 7.1). The decision point combines the verdicts of its rules by keeping the strongest decision and merging their obligations, in order and without duplicates; with no rule objecting, the verdict is* allow*. It fails safe: a rule that crashes gives* deny*; a malformed result is never read as admitting the call; and when the decision point cannot load its rules at all, requests in a fixed* always-open *set (reads, σ-emits, and restore operations) are allowed and every other request is denied. The call runs only on* allow *or* warn*.*

Status. That every tool call of a digital agent passes through the gateway pair of its decision point is the condition of complete mediation (Definition 7.13), not part of this definition. The routing points of a commons' COMDAIMs (Definition 10.10) are a different notion.

Definition 7.13 (Complete mediation, a condition)**.** *A commons c satisfies* complete mediation *when every tool call of each of its digital agents $v \in V^c$ (Definition 3.10) passes through the gateway pair of its decision point (Definition 7.12), which traps the call before and after it runs.*

Status. This is the enforcement posit that the Status of Definition 7.12 names: nothing in the definition of the decision point, and nothing in the roots, makes every tool call pass through a gateway pair. A proposition whose proof uses the trap names this condition in its statement.

Proposition 7.14 (The gateway pair is the decision point)**.** *For the decision point of Definition 7.12:*

(i) given that the gateway pair hands the decision point the request unchanged, it returns exactly the decision point's verdict, whatever vendor harness carries the call;

(ii) adding a rule never loosens a decision: the combined decision is at least as strong as each rule's, and it is allow *exactly when every rule allows;*

(iii) while the decision point has its rules loaded, and one of its rules is the capability rule*, which decides* deny *on a capability atom in* Deny *(Definition 4.13),* allow *on an atom of $\varphi(T)$ that is not in* Deny*, for the active taskset T (Definition 7.1), and* hold *otherwise: an atom in* Deny *is decided* deny *and is never allowed;*

(iv) while the decision point has its rules loaded, a capability in the governance or the external class (Definition 7.12) is decided hold *until its approval is recorded, after which the approval rule allows it; the other rules still count, so a capability in* Deny *stays denied;*

(v) the combined obligations are exactly those some rule carries, with no duplicates, and the earlier duties keep their order.

So, under the hypotheses of (iii), every capability atom that the decision point allows at a tool call the gateway traps lies in $\varphi(T) \setminus$ Deny*; under complete mediation (Definition 7.13), every tool call of a digital agent is such a call.*

Proof. (i) The gateway pair calls the decision point and returns its verdict unchanged; in the model the harness never enters the decision, which is why (i) needs the request to arrive unchanged. (ii) The decision order is a chain and combining takes its maximum; the maximum dominates each argument, and it is the bottom element, allow, exactly when every argument is. (iii) The capability rule decides deny on the deny tier, and a call is allowed only on the decision allow; for the closing sentence, the capability rule allows an atom exactly when it is in $\varphi(T)$ and not in Deny, and by (ii) the decision point allows only what every rule allows. (iv) is the approval rule's two cases. (v) Merging appends only duties not already present. All five are proved in Lean (`hookpair_eq_decide`, `harness_independent`, Lean `Substrate.HookPairEquivPDP`; `enforce_preserves_hook`, `enforce_preserves_pdp`, `enforce_allow_iff`, Lean `Substrate.AdditiveEnforce`, for two verdicts, and `combine_allow_iff`, Lean `Paper.GatewayObligations`, for any number of rules; `denyVerdict_iff_denytier`, `deny_never_feasible`, Lean `Substrate.DenySolePath`; `gated_routes_needsApproval`, `gated_needs_recorded_approval`, `approval_releases`, Lean `Substrate.NeedsApprovalConcage`; `mem_combine`, `nodup_combine`, `prefix_mergeObs`, `prefix_combine_snoc`, `rank_le_combine`, Lean `Paper.GatewayObligations`). The certificate for (i) says nothing about harnesses: the model's harness argument is unused, so independence holds by construction, and the real content of (i) is the hypothesis below. The fail-safe behaviours of Definition 7.12 (a crashing rule denies, the always-open set, a malformed result never admits) are property tests on the executable model, not certificates. Part (i) and the closing sentence rest on a hypothesis that is not proved: that each vendor's call translates faithfully into the request the decision point reads. Reading the closing sentence at every tool call of a digital agent, and not only at a trapped one, uses in addition complete mediation (Definition 7.13), a condition, which the statement names. □

Status. Shallow: (i) follows by construction, since the model defines the pair's decision as the decision point's verdict combined with itself, so that `hookpair_eq_decide` is `combine_idem`, and the harness argument is never used, so that `harness_independent` holds by construction. The inclusion of the closing sentence is certified for the capability rule alone (`feasible_iff_enforced`, Lean `Substrate.DenySolePath`); its extension to the combined decision is (ii) applied to it, a text step. Its converse does not hold: by (iv), an atom of the governance or the external class in $\varphi(T) \setminus$ Deny is held until its approval is recorded. When the decision point cannot load its rules, requests in the always-open set of Definition 7.12 are allowed whatever their capability, so (iii), (iv) and the closing sentence are stated only while the rules are loaded; neither Lean model has a load-failure mode.

Proposition 7.15 (The health gate cannot deadlock)**.** *Let* $A(x)$ *be a set of controls for each state* x*,* H *a health predicate, and* R *a restore set of controls. The* health gate *admits* $A_H(x) = A(x)$ *in a green state* $(H(x))$ *and* $A_H(x) = A(x) \cap R$ *in a red one. Assume that restore is always admissible,* $R \subseteq A(x)$ *for every* x*. Then:*

(i) in a green state the gate changes nothing, and in a red state it admits exactly R*;*

(ii) if R is non-empty, every state has an admissible control, so the gate never deadlocks;

(iii) given a repair choice $\mathrm{rep}(x) \in R$ *and a measure* μ *with values in* $\mathbb{N}$ *that strictly drops along each repair from a red state and is zero only on green states, every red state reaches a green state within* $\mu(x)$ *repairs, each admissible where it is taken.*

Proof. (i) is the definition of A_H, with $A(x) \cap R = R$ when $R \subseteq A(x)$. (ii) follows from (i): a green state admits $A(x) \supseteq R$, a red one admits R. (iii) is induction on $\mu(x)$: from a red state a repair is admissible by (i) and lowers μ, and when μ reaches zero the state is green. All three are proved in Lean over any states, controls and transition map (`gated_green`, `gated_red`, `no_deadlock`, `red_reaches_green`; Lean `Paper.NoDeadlock`). The hypotheses are real: a rule that denied a restore request would break $R \subseteq A(x)$, and a property test on the decision point exhibits exactly such a rule. The measure μ is what the repair pump of Corollary 4.46 has to supply. □

Status. Read on the controlled substrate chain (Definition 4.15), $A(x)$ is the admissibility correspondence before its declared health gate and A_H is that gate; the chain's definition folds the gate into $A(x)$, so this reading is not certified. The assumption $R \subseteq A(x)$ is not a named condition, and no z0 generator gives it.

7.6 Making the boundary real

Scope of this section. The security results that follow guard only what is observable in the substrate, its files and its σ-log: the adversaries of tier 0 (misdiagnosis, hijack, coordination-corruption, zero-day) and of tier 1 (intent-divergence, coercion, supply-chain). The tier-2 attacker (host substitution, firmware Trojan, signing-primitive break) lies outside hypothesis (P2) of Proposition 7.24, which trusts those primitives; Remark 7.34 addresses recovery after a compromise, under its own hypotheses and with a step that is not proved.

Definition 7.16 (Confinement class of a realisation)**.** *Let* $r \in \mathcal{R}$ *be a realisation of the identityless substrate (Definition 4.16), the component of a turn-agent (Definition 3.12) that serves its label. The* confinement class *of* r *is the kind, one of three, of the mechanism, if there is one, that makes every substrate-effecting action of* r *pass the meta-circular reference-monitor (the metacircular core of Definition 9.11) before it takes effect: (a)* interception *— the runtime exposes a pre-effect interposition, a hook fired before each of* r*'s tool calls; (b)* construction *—* r*'s effector set is restricted a priori to substrate-door calls only, so there is no unmediated effector to intercept; (c)* containment *—* r *executes inside an OS-kernel isolation (for example Linux Landlock) in which the only reachable substrate path is the mediated door.*

Status. The mediation named here, every substrate-effecting action of r passing the meta-circular reference-monitor, is stated for the realisation r; complete mediation (Definition 7.13) is stated for the tool calls of digital agents at the gateway pair, and how the two relate is not asserted here. The confinement class is meant to be the mechanism that realises the gate of σ-discipline (Proposition 3.22), which refuses an effect whose action has no decision record, against a non-cooperative realisation. σ-journaling (Definition 3.21) gives only that a record exists, not that it comes before the effect; neither is a dependency of the definition. The three classes are meant to hold whether or not r cooperates; in (a), that r cannot opt out of the hook is meant, not defined. A realisation is meant to be admissible for identity-bearing binding only if it carries at least one confinement class sufficient for the capabilities its active taskset projects under the IPLiKiDi projection φ (Definition 7.1); "sufficient" is not defined here, so this admissibility is not a defined predicate. A layer of enforcement hooks is the interception instance (class (a)); the construction and containment classes are meant to give the same enforcement for runtimes that expose no hook. That the metacircular core is non-empty and fixed by C is Lemma 4.34.

Proposition 7.17 (Containment-class soundness via realisation-preserving-\; text proof)**.** *Let* $\varrho : \mathrm{Cap} \to \mathrm{PrefixPolicy}$ *be the operating-system kernel realisation (Landlock, on Linux), where* PrefixPolicy *is the lattice of finite unions of prefix-cones and single nodes over the observed path tree, equipped with the prefix-tree difference* $\setminus_{\mathrm{tree}}$*: subtracting* $\mathrm{cone}(r)$ *from* $\mathrm{cone}(p)$*, for* $p \preceq r$*, is the finite union of the sibling cones off the path from* p *to* r *together with the path nodes from* p *to the parent of* r *as single nodes, which terminates on the finite tree. Let* T *be a taskset, with enforced capabilities* $\varphi(T) \setminus \mathrm{Deny}$ *(Proposition 7.3). If* ϱ *preserves the relative complement at* $\varphi(T)$ *and* Deny*,*

$$\varrho\big(\varphi(T) \setminus \mathrm{Deny}\big) \;=\; \varrho(\varphi(T)) \;\setminus_{\mathrm{tree}}\; \varrho(\mathrm{Deny}),$$

and the containment *class (Definition 7.16) realises for* T *the rule set* $\varrho(\varphi(T)) \setminus_{\mathrm{tree}} \varrho(\mathrm{Deny})$*, then that rule set grants exactly* $\varrho(\varphi(T) \setminus \mathrm{Deny})$*, so no denied capability is reachable through it. The hypothesis holds in particular when* ϱ *is a homomorphism of generalised Boolean lattices, preserving* $\cup$*,* $\cap$ *and* $\setminus$*; a realisation that is merely a join-homomorphism need not satisfy it.*

Proof. If ϱ preserves $\setminus$ at $\varphi(T)$ and Deny, the realised rule set, which is $\varrho(\varphi(T)) \setminus_{\mathrm{tree}} \varrho(\mathrm{Deny})$ by hypothesis, grants exactly $\varrho(\varphi(T) \setminus \mathrm{Deny})$, so no denied capability is reachable. (A join-only realisation need not do so.) Suppose ϱ only preserves joins, carrying $\varrho(\mathrm{Deny})$ alongside $\varrho(\varphi(T))$. The kernel's rules are prefix-hierarchical and cannot punch a hole in a granted cone, so a cone of $\varrho(\varphi(T))$ that contains a denied sub-path is granted whole, and the denied capability is reachable. For instance, write access to a directory $/a$ is realised as the cone of everything under $/a$; a deny on $/a/b/secret$ removes nothing from that cone unless the realisation replaces it by the sibling cones around the denied path. □

The realisation's narrowing step is meant to compute this expansion. On-host enforcement is exercised in shadow mode only.

Status. Text proof; not machine-checked.

Remark 7.18 (Containment-class confinement-sufficiency)**.** *A containment realisation is specified by a single bootstrap-confinement predicate — an irreversible, init-applied restriction bundling capability-bounding, a system-call filter, namespace isolation, and an address-family restriction, designed to be installed before the confined process's first instruction. The implication* bootstrap-confinement ⇒ per-operation preservation of the axiom kernel *is checked in a bounded symbolic model-checker (the check passes, with no design axioms); the corresponding implication is proved in Lean 4 (`OSSubset.CarrierUnified`), on the standard axioms only, with the model's seven seams as named assumptions, not proved. The modelled system-call semantics are then* ***validated against a production kernel by differential testing*** *— each* differentially-tested *operation's real-kernel outcome agreeing with the model's prediction except in one case, in which the real kernel is strictly* more *restrictive than the model (one modelled operation, integrity appraisal, is hardware-gated and its differential validation is pending). Two caveats keep this a strengthening, not a closure: the model covers a named subset of the system-call interface (its complement is residual trusted base), and the differential validation is empirical, not a proof that the kernel implements the model. Within those caveats, a containment-class realisation would carry, in the model, the enforcement interposition its in-flight envelope demands, so the enforcement-continuity of Conjecture 5.10, which remains a conjecture, would hold for containment-class receivers* conditional on the residual base, the model's seven named seams and the empirical kernel validation, *not on an unexamined confinement assumption. The Landlock realisation is exercised in shadow mode only; live enforcement is not yet deployed (Proposition 7.17).* □

Remark 7.19 (OS-inclusion: the host OS is a load-bearing component of the substrate state)**.** *The state bundle of Definition 4.7 carries, in its per-commons factor* $\mathbb{S}^c$*, the files,* σ*-chain, and agent-context that SOMA is realised within — but those objects live inside a host operating*

system whose own integrity is a necessary condition for every invariant above them. We extend the substrate state to include, as an explicit coordinate, the host-OS security posture: the kernel and process table, the filesystem root /, and the resource-state that the containment realisation of Remark 7.18 rests on. The inclusion rests on the OS-subset model's kernel assumptions — isolation (uid-namespace disjointness), file protection (discretionary access control and immutability of root-owned inodes), and system-call semantics over the modelled op-set — which are the trusted-computing base the containment result is grounded in but does not itself prove. These assumptions name, honestly, the trusted base: hardware correctness, the kernel outside the modelled system-call subset, the proof-checker kernel, and the boot-integrity implementations. Within that trusted base, the OS coordinate of the substrate state is meant to be as formally tractable as the file-and-process coordinate above it. Including the operating system is therefore not a strengthening of the containment result but a clarification of what that result already ranges over: $s \models \mathrm{SOMA}$ (Definition 4.2) is a statement about the host OS as much as about the git-tracked subtree that runs on it — and an OS-integrity failure can falsify it. □

Remark 7.20 (Composable runtime-and-supply-chain attestation pipeline strengthening the base scan)**.** *The base character-stream scan (Proposition 7.26) can be strengthened into a composable attestation pipeline of three layers, each built from an established primitive. The strengthening sharpens the detection of substrate-modifying attacks (Proposition 7.24, hypothesis (P2)); the base scan's soundness does not need it.*

Layer 1: integrity at rest. *A block Merkle tree $\mathcal{T}_v(f)$ over each binary, built at install time, whose root $r(\mathcal{T}_v(f)) \in \{0,1\}^{256}$ binds its content. The primitive is the Linux kernel's per-block hash tree for read-only devices (dm-verity), and for source, git's content-addressed Merkle graph.*

Layer 2: supply-chain property attestation. *A set $\mathcal{A}_v(f) = \{(\phi_k, w_k, \sigma_k)\}_{k=1}^{K}$ of attested properties: ϕ_k a property ("opens no network socket", "writes only to mirror paths"), w_k its witness (a proof, trace or type-check result), and σ_k its σ-event and signing-key reference, with a root $r(\mathcal{A}_v(f))$ binding the set. The primitives are in-toto [TAAK+19], Supply-chain Levels for Software Artifacts (SLSA) [Ope21], and the Sigstore transparency log [Sig21].*

Layer 3: streaming structural composition. *An operator $\mathcal{S}_v(f)$ composes $\mathcal{T}_v(f)$, $\mathcal{A}_v(f)$ and the σ-event stream during execution into a labelled spine $\mathcal{A}(f)$ of the binary, distinct from the Layer-2 property set $\mathcal{A}_v(f)$,*

$$\mathcal{A}(f) \;=\; \{\,\mathrm{entry}_k\,\}_{k=1}^{K'},$$

in which each structural block (entry point, cryptographic routines, input-output surface, and so on) has an entry binding its label, its byte range and the content hash of that range, the property attested there, and a σ-event reference. The primitives are measured boot with Trusted Platform Module (TPM) register extension [Tru14], each component measuring the next, and on the software side the verifier of the extended Berkeley Packet Filter (eBPF) [GAG+19]. Each entry is journalled as it is built, so $\mathcal{A}(f)$ is itself verifiable against the σ-chain.

Composition. *The introspection scan is the composition of the three layers,*

$$\mathrm{scan}(f) \;:=\; \mathcal{S}_v\big(\,\mathcal{T}_v(f),\; \mathcal{A}_v(f),\; \sigma\text{-stream}\,\big),$$

and the attestation of the whole substrate graph is the tagged disjoint union $\mathcal{A}(\mathrm{SPIG}) = \bigsqcup_{v \in V} \{v\} \times \mathcal{A}(f_v)$, which keeps each file's labels intact. Every primitive is borrowed; what is new is only the composition operator $\mathcal{S}_v$, which turns the layers' outputs into one spine verifiable against the σ-chain. Capability hardware (CHERI) [WWN+15] would strengthen Layer 3 with capability-bounded execution at the level of single instructions. □

The verifier is the trusted base. For the symbolic register, the relation that clears a claim as machine-checked is realised by a sealed proof checker, and that checker is the trusted computing

base: "P is machine-cleared" means "P is cleared, trusting the checker". The trust is exhibited, not hidden. Each certificate reports its own axiom footprint, and a fast path that trusts the compiler is bounded by a path the Lean kernel checks, so the base is small and can be audited. The checker is itself a governed node of the substrate: its build specification and harness are files, so the system carries its own means of clearance inside the graph it clears (Remark 7.21). For an operator that does not reason symbolically, clearance is that operator's own way of telling its registers apart.

Remark 7.21 (Survival-floor core-checkability)**.** *The z=0 core is meant to re-check in a finite, dependency-free container, within what the poorest edge node affords under its resource budget (Definition 6.16), so that acceptance is decidable at the survival floor. This is exhibited for the finite z=0 fragment, and the check fails closed beyond it. It is a design property of the checker, stated here as a remark: there is no proof of it and no certificate.* □

7.7 Detecting and surviving attack

Remark 7.22 (IPLiKiDi projection is the active cyber-defense boundary)**.** *Under Definition 3.15 (subset-private memory), Proposition 3.19 (memory inviolability), and Definition 3.21 (σ-journaling), the IPLiKiDi projection φ of Definition 7.1 defines the framework's active cyber-defense boundary on the Substrate-Process IAS Graph* SPIG *(SPIG, Definition 9.4) at every σ-event-time t: the allow-list that the framework's meta-circular reference-monitor (Lemma 4.34) enforces at time t is contained in* $\varphi(\text{active-taskset}(t))$*, and need not be all of it, since an atom of the governance or the external class is held until its approval is recorded (Proposition 7.14) — sharpened to* $\varphi(\text{active-taskset}(t))\setminus\text{Deny}$ *once the root-of-trust deny-tier is present (Definition 4.13, Proposition 7.3). That the enforced allow-list is contained in this rests on complete mediation (Definition 7.13), every tool call of a digital agent being trapped by the gateway pair, on the gateway passing each call on faithfully (Proposition 7.14) and, for the containment class, on the kernel realisation preserving the relative complement (Proposition 7.17). Authorized graph-modifications satisfy capability-confinement* $\subseteq \varphi(\text{active-taskset}(t)) \setminus \text{Deny}$*; unauthorised modifications are meant to be rejected by the enforcement layer, under complete mediation (Definition 7.13); character-stream introspection (Proposition 7.24) is meant to audit the rejection.*

Argument. The argument uses the completeness assumption of Remark 9.6, a remark requiring that the reference monitor of SPIG be complete and that every admitted modification A have effective capabilities in $\varphi(T_A)$ and obey write-xor-execute, together with the Galois connection of Proposition 7.2; the reading of φ's image as the minimal sufficient capability scope is uncertified. The enforcement layer is designed to consult $\varphi(T_A)$ before admitting A; the meta-circular reference-monitor (Lemma 4.34) is the framework's φ-implementing fixed-point. Character-stream introspection of Proposition 7.24 provides the audit-trail. □

Cyber-defense unification (full picture). The framework's intended defence-by-construction is the composition of four properties: (i) under σ-journaling (Definition 3.21), every graph-write that is a digital agent's action of a type in the σ-vocabulary has a σ-record; (ii) every σ-event is character-stream-introspectable (Proposition 7.24); (iii) every authorized graph-modification respects the IPLiKiDi capability-projection (Remark 9.6 + Remark 7.22); (iv) the meta-circular reference-monitor is unchanged at the metacircular fixed point of the self-application $\Phi(x) = C(x, \text{accept}(x))$, of which there is at most one under tier sealing (MI) and tier progressivity (Q) (Definitions 4.26 and 4.27), and exactly one when some specification has the tiers that the rounds of self-amendment settle (Lemma 4.34 and Proposition 4.28); neither condition involves the quorum. Cyber-attacks on SPIG — capability-projection bypass, σ-chain forgery, allowlist evasion, social-engineering of the agent — are designed to reduce to violations of one or more of (i)–(iv), each of which, within the attested byte ranges, leaves a trace that the content hashes of Proposition 7.24 (ii) expose; that every attack so reduces is the design's claim,

not a theorem.

Definition 7.23 (Character-stream introspection of SPIG)**.** *Let* Σ_{char} *be the character-alphabet of the substrate's file-system encoding (typically UTF-8, the 8-bit Unicode Transformation Format). For each substrate-file* f *of node* $v \in V(\text{SPIG})$ *(Definition 9.4), let* $\text{stream}(f) \in \Sigma_{\text{char}}^*$ *be the character-by-character flattening of* f*'s content. The* character-stream introspection scan *is the deterministic function*

$$\begin{aligned} &\text{scan} : \text{SPIG} \to \text{SPIG-estimate}, \\ &\text{scan}(\text{SPIG}) := \{(\lambda_V(v), v', \tau) \,:\, v \in V,\ v' \in V,\ \tau \in T_E, \\ &\qquad\qquad \mathfrak{R}(\text{stream}(f_v), v', \tau) = 1\}. \end{aligned}$$

where $\mathfrak{R} : \Sigma_{\text{char}}^* \times V \times T_E \to \{0,1\}$ *is the dictionary-driven reference-detection predicate (exhaustive substring-match against the substrate's marker-pattern dictionary: file-path references, label references,* σ*-event-type references, hook-rule references, etc.). The scan operates by streaming every character of every substrate file through the dictionary tables; no regex, no abstract-syntax-tree (AST) extraction, no clever skipping — the most awk-style exhaustive scan possible.*

Status. In the sovereignty semantics of the section on law and jurisdiction, this scan is the boundary monitor on the *cross-provenance edges* of Remark 10.7 — operations whose training-provenance $\jmath_{\text{prov}}$ is untrusted or differs from the deployment State $\jmath_{\text{deploy}}$.

Proposition 7.24 (What character-stream introspection detects; text proof)**.** *Let* $\text{SPIG}^{(t)}$ *denote the substrate state at* σ*-event time* t*. Assume* σ*-journaling (Definition 3.21), and that every authorised modification is a digital agent's action of a type in the* σ*-vocabulary. Suppose (P1) the dictionary tables of scan (Definition 7.23) are exhaustive over the substrate's marker-pattern vocabulary, and (P2) the three borrowed primitives of Remark 7.20 are trusted: the content-addressed layer (the git Merkle DAG, dm-verity), the supply-chain layer (in-toto, Supply-chain Levels for Software Artifacts (SLSA), Sigstore/Rekor) and the platform layer (Trusted Platform Module (TPM) register extension, the eBPF verifier). Consider a modification of* $V(\text{SPIG}^{(t)})$ *between consecutive* σ*-events* t_i, t_{i+1} *that bypasses operator-attested authorisation. Then:*

(i) if it changes the set of recognised references of some node, adding or removing a triple $(\lambda_V(v), v', \tau)$ *of scan that no authorised change in the interval also adds or removes, it shows as a delta between* $\text{scan}(\text{SPIG}^{(t_i)})$ *and* $\text{scan}(\text{SPIG}^{(t_{i+1})})$ *that no entry of the* σ*-chain predicts; a change that preserves that set, such as a duplicated, moved or re-spelled reference, is caught, if at all, only by (ii);*

(ii) whatever it changes within a byte range that an entry of the structural attestation spine binds, it shows as a delta in the content hashes bound by the entries of that spine $\mathcal{A}(\text{SPIG}^{(t_{i+1})}) - \mathcal{A}(\text{SPIG}^{(t_i)})$ *(Remark 7.20) that no entry of the* σ*-chain predicts.*

Proof. By Definition 3.21 every authorised modification that is a digital agent's action of a type in the σ-vocabulary has a record; since, by assumption, every authorised modification is of that kind, the σ-chain between t_i and t_{i+1} lists every authorised change. (i) Under (P1) the scan is complete as well as sound for what it recognises (Proposition 7.26): it reports exactly the set of recognised references, as triples, so a change that adds or removes a triple alters its output, and, since no authorised change in the interval accounts for that triple, the alteration is unexplained by the σ-chain. The scan is a set, so it does not see multiplicity or position: a change that keeps the set fixed leaves its output unchanged. (ii) Under (P2) a content hash changes whenever any byte of the hashed content changes, so every modification within a bound range, recognised by the scan or not, alters the attestation; a change outside every bound range is not covered. This is a proof in the text; there is no machine certificate. □

However, the scan alone is not a detector of arbitrary change: an edit that touches no pattern the dictionary knows, such as changing a constant or deleting a check, or that keeps the set of recognised references fixed, leaves the scan unchanged and is caught only by the content hashes of (ii), and only within the attested ranges. In words: the scan says *what* changed in the references, and the hashes say *that* anything in an attested range changed.

Status. Text proof; not machine-checked.

Reading a past transcript is a governed act. The capability discipline that governs every act also governs reading a past transcript. A raw trajectory, and each channel transcript between agents, including the cross-operator case that straddles two commons, is a governed object. Opening one is an act gated by a read capability, an instance of the egress boundary (Proposition 7.8) applied to what may be seen, scoped to a subset and, in a dispute between operators, resolved with the jurisdiction explicit (Definition 10.4). The read is not a plain load but a security boundary. Every character it ingests can be introspected (Proposition 7.24), and this, with the keyless check below, is meant to defend against a threat at the source that a defence in transit cannot see: a compromised vendor inference kernel (the vendor's model-serving stack, not the operating-system kernel) that emits a transcript benign when written but carrying latent adversarial bits that act only when the sealed past is read later.

Two facts make the defence real. The character scan refutes injections that can be stripped but is silent on statistical marks, so a read must also run the keyless check described below against the reader's own dated baseline. And a uniform poisoning of the kernel can be detected only against a sealed past kept in the root operator's sanctum tier, which is why the transcript history is designed to be inviolable, rotated turn by turn into an append-only sanctum, while only the live turn can be changed by an agent. The limits are stated with the defence. A null result means not yet detected at this date, never proof of absence. A detected anomaly must be traced to its source, the agent's own emission, an inbound channel or the vendor kernel, and cleared against a negative control before a compromised kernel is blamed, or drift in the baseline would be read as poison. Transport encryption on the cross-operator channel and the isolation floor between agents are declared obligations of the build, not yet done, and those are exactly the points at which an adversary can obstruct the operator's LLM-free control (Axiom 3.9).

Proposition 7.25 (Covert-character scan soundness)**.** *Let* Σ_{char} *be the character alphabet of Definition 7.23, let* covert : $\Sigma_{\text{char}} \to \{0,1\}$ *be any decidable predicate on characters, the one that marks the covert class, and let* sanitize *keep exactly the non-covert characters of an input stream* $s \in \Sigma_{\text{char}}^*$*, in order. Then for every such* s*: (i) no covert character survives, that is, every character of* sanitize(s) *is non-covert; (ii)* sanitize(s) *is* s *with exactly the covert characters removed and the order of the rest preserved; (iii)* sanitize *is idempotent,* sanitize(sanitize(s)) = sanitize(s)*.*

Proof. sanitize is the filter of a finite list by the decidable predicate ¬covert. Every element of a filtered list satisfies the predicate, which is (i); the filter keeps exactly the elements satisfying it, in order, which is (ii); filtering again by the same predicate removes nothing, which is (iii). Machine-checked for any decidable covert predicate, on the standard axioms only (`coverage_complete`, `semantics_preserving`, `idempotent`, Lean `Paper.ScanSoundness`). □

In words: untrusted model output is designed to enter the substrate only through a filter that removes every hidden character and nothing else, and filtering twice changes nothing. The raw stream is kept verbatim as a forensic record, and the filtered view is recomputed from it and the short list of what was stripped, so nothing is stored twice. That the raw stream is recovered exactly from the filtered view and the stripped characters with their positions is a tested law of the realisation, with no Lean certificate yet.

Status. Shallow: sanitize is defined in the statement as the filter by ¬covert, so (ii) is that definition and (i) and (iii) are laws of the list filter; the Lean content is that the scan written as a fold, `sanitizeFold`, is that filter (`semantics_preserving`). The intended covert class is the invisible characters: zero-width and bidirectional-control characters, tag characters and the like; the proposition holds for any decidable predicate.

Proposition 7.26 (Scan well-definedness, complexity, soundness; text proof)**.** *The character-stream introspection scan of Definition 7.23 is well-defined and deterministic; converges in* $O(|V| \cdot \max_f |stream(f)| \cdot |\mathfrak{D}|)$ *time where* $\mathfrak{D}$ *is the dictionary table; and is* intensionally sound: *every edge in scan*(SPIG) *corresponds to a real reference in the substrate; false references are not introduced. The scan is* extensionally incomplete *iff the dictionary* $\mathfrak{D}$ *misses the marker pattern of some reference that occurs in the substrate; with a dictionary that has the marker pattern of every reference that occurs, the scan is also extensionally complete (no real reference is missed).*

Proof. Determinism follows from the deterministic dictionary-lookup at each character position. Complexity bound is the cost of streaming each character of each file through the dictionary table once. Intensional soundness: $\mathfrak{R}(\text{stream}(f_v), v', \tau) = 1$ only if a literal pattern in $\mathfrak{D}$ matches at some position of $\text{stream}(f_v)$, and patterns in $\mathfrak{D}$ are defined to match exactly the substrate's reference syntax. This soundness holds by the definition of $\mathfrak{D}$, not by a check against the substrate: a pattern that also matched text that is not a reference would introduce a false edge. Extensional incompleteness/completeness statement follows by definition of $\mathfrak{D}$'s coverage. □

Status. Text proof; not machine-checked.

Definition 7.27 (Declared size bounds)**.** *For a substrate file* f *under a canonical label's control, a* declaration of size bounds *is an upper bound* $B_f^{\max} \in \mathbb{N}_{>0}$*, counted in lines, and optionally a lower bound* $B_f^{\min} \in \mathbb{N}$*, declared per canonical label beside its read scopes (Definition 3.16). A substrate* satisfies the declared size bounds *when (i) the bounds of every such file change only with the operator's attestation, not by the digital agent of that label; and (ii) its* size-bound gate *refuses a change to* f *that would break either bound, unless a curator pass has recently completed and been journalled, journals each refusal as a* σ*-event (Definition 4.47), and admits an override only with the operator's attestation.*

In words. For per-label write confinement of memory, see Proposition 3.19. Lines are the unit of a version-control diff. A lower bound is for files whose shrinking is itself the attack, such as a roster of authorised hosts whose emptying would lock the operator out. A bound may be exceeded within a curator window, after a journalled curator pass; the gate's exemption is deliberate. This is the store face of Axiom-0 made into a rule the substrate is designed to apply to its own files.

Definition 7.28 (Curator-procedure on a bound-approaching substrate-file)**.** *Let* f *be a substrate-file under canonical-label control with declared bounds* $(B_f^{\min}, B_f^{\max})$ *(Definition 7.27) and recent write-rate* $\dot{B}_f$*. The* curator-procedure *on* f *is the following three phases.*

(i) ***Classify*** *— partition* f*'s content into three classes: (a) live (referenced within recent* σ*-log read-events; within-bounds-projection); (b) stale (no read-events in a window* W_{stale}*); (c) overlap (content-hash near-match to other canonical-label files; redundancy candidate). The classifier may optionally invoke Layer-1 cross-commons compound-detection rules over the* σ*-log content that peer commons offer and this commons takes; otherwise classification is local to the canonical-label.*

(ii) ***Prune-or-archive*** *— for each class: live → keep in live tier; stale → archive to the cold-storage tier (Definition 7.29); overlap → consolidate-or-archive (operator-attest choice when the consolidation is irreversible).*

(iii) ***Consolidate*** *— merge overlapping content into the canonical-live tier; rewrite live tier under bound. A curator pass on f is* complete *when all three phases have run and it has emitted the σ-event chain* `substrate.curator.classify` → `substrate.curator.prune` → `substrate.curator.archive` → `substrate.curator.consolidate`.

A substrate satisfies curation *when, for every such file f whose write-rate $\dot{B}_f$ projects bound-breach within W_{cure}, the digital agent bound to f's canonical label runs a curator pass on f.*

Remark. The default stale window W_{stale} is 7 days. The event chain is structurally parallel to the four-event recombination chain `agent.recombination.{proposed, quorum_attested, completed, aborted}` of Definition 9.8. The operator-attest choice at an irreversible consolidation parallels the attestation of a specification amendment where the commons satisfies the coevolution quorum, a condition (Definition 8.17); the quorum is not a hypothesis of the metacircular fixed point (Proposition 4.28). The content peer commons offer and this commons takes is what passes to it in Proposition 8.8.

The curator-procedure's three phases, classify, prune-or-archive and consolidate, are also the substrate-file analogue of the **propose → attest → enact** structure of substrate-canonical schema-amendment ceremonies, with a content-classification preamble (phase (i)) because substrate-file mutations are content-level not just schema-level. Operator-attest is required at phase (iii) consolidate when the consolidation is *irreversible* (live content rewritten, not just appended); appends can be completed by the digital agent without operator-attest, since they rewrite nothing; an archive to the cold-storage tier can be completed by it too, and is undone only by the operator-attested rehydrate ceremony (Definition 7.29).

Definition 7.29 (Cold-storage tier; per-canonical-label archival store)**.** *For a canonical label v of a commons c (Definitions 3.10 and 3.14), the cold-storage tier $\mathcal{Z}^v$ is a declared sub-store of the per-commons factor $\mathbb{S}^c$, with a per-label budget C^v_{cold} and an access predicate under which v is permitted to read the files of $\mathcal{Z}^v$ and not to write them.* Archival *moves a file of v from the live tier into $\mathcal{Z}^v$, and* rehydration *moves it back.*

Implementation. The sub-store is a per-label, per-date compressed archive path in the commons' substrate, and the budget is operator-attested at commons-instantiation (default 1 GB compressed). Rehydration requires the operator-attested cold-rehydrate ceremony, which emits a `substrate.coldstorage.rehydrate` σ-event. Archival is irreversible at session-scope, and reversible only through an operator-attested ceremony.

Status. The read-but-not-write rule extends the write prohibition of memory inviolability (Proposition 3.19) to cold storage.

The cold-storage tier's destination-of-archive structure mirrors a three-replica cold-backup-tier framework (encrypted snapshots to cross-jurisdiction object storage at recovery-point and recovery-time objectives (RPO/RTO) matched to operator-attest cadence). The choice of per-canonical-label granularity for the budget C^v_{cold} (versus per-commons aggregate $C^c_{\text{cold}} = \sum_v C^v_{\text{cold}}$) is operator-attested at commons-instantiation; default is per-canonical-label with commons-aggregate enforcement as a derived sub-budget partition.

Proposition 7.30 (Finite bounds defeat file-bloat attacks on the operator's review; text proof)**.** *Assume Axiom 3.1, σ-journaling (Definition 3.21), the curator procedure of Definition 7.28, the cold-storage tier of Definition 7.29, an enforcement layer that refuses any change breaking a file's declared size bounds (Definition 7.27), hypotheses (P1) and (P2) of Proposition 7.24 with its assumption on authorised modifications, the conditions of Proposition 3.19 (the enforced access path, Definition 3.17, label-to-user correspondence, Definition 3.18, and the modelled discretionary access control), that disabling the enforcement layer requires an operator attestation,*

that no operator in O^c is rogue, and that no operator's signing key is compromised. Then no adversary $\mathcal{A}$, whether an impersonator of the operator, a rogue member of V^c, or an external denial-of-service attempt, can grow without bound a substrate file under a canonical label that carries a declaration of size bounds. Bloating the operator's daily review file until it cannot be read, bloating memory until a session cannot load it, or bloating a commit message to hide what it carries is defeated, up to the completeness of the audit trail of operator attestations. A change admitted under the curator-pass exemption of Definition 7.27 is outside this proposition, whose enforcement hypothesis has no such exemption.

Proof. Let $\mathcal{A}$ try to grow f past $B_f^{\max}$, or shrink it below $B_f^{\min}$. There are three channels. (1) A direct write: the enforcement layer refuses it and journals the refusal; by Definition 3.21 the record is durable, by Proposition 3.19, under its conditions, no daemon of another label writes the log of that label, and by Proposition 7.24(ii) any later change to the log within an attested range is detected. The attempt is blocked and audited. (2) Disabling the guard: this is an operator-attested act, itself journalled, and the guard's own source is covered, within the attested ranges, by the content hashes of Proposition 7.24(ii), so a change to it there is detectable. (3) Raising $B_f^{\max}$, by the digital agent of that label or by impersonating the operator: a bound changes only with the operator's attestation (Definition 7.27), which is what protects it, since Proposition 3.19 stops other labels but not the digital agent of that label; the change is journalled, and impersonation reduces to forging an operator attestation, which the signing primitives trusted in hypothesis (P2) of Proposition 7.24 rule out while no operator's signing key is compromised. In every case the attack becomes a journalled refusal or a detectable forgery. The claim holds up to the completeness of the audit of operator attestations, which is the scope of the reference monitor (Proposition 7.24, Lemma 4.34). Denying Axiom 3.1, that is allowing unbounded growth, exhibits the attack directly. □

Status. Text proof; not machine-checked.

Definition 7.31 (Substrate-fault classes for a compiled capability boundary)**.** *Let the capability projection φ of Definition 7.1 be enforced through the chain: policy source on disk → compiled bundle → the running session's matcher snapshot → OS kernel. A* substrate fault *is a divergence between the intended allow-list and the allow-list enforced at an instant t. Three fault classes are named: (i)* static-source *— the on-disk policy source is incorrect; (ii)* static-matcher *— the on-disk policy is correct but the runtime matcher's interpretation (path canonicalisation, wildcard semantics) diverges from the compiler's; (iii)* dynamic-snapshot-staleness *— disk and matcher are both correct at every instant, but the running session enforces a policy snapshot cached at session-start.*

In words. In class (i) every downstream layer faithfully enforces the wrong law; the repair is to re-author the source and recompile. Class (ii) is repaired by matcher-canonicalisation patches plus naming-convention discipline. In class (iii) the divergence is temporal — session-cached state versus substrate-canonical state — and is repaired only by a restart that re-reads the substrate; no in-session edit can help.

Status. Class (i) is named for the stage at which the error lies, the source; a compiled bundle that diverges from a correct source is a fault at the compile stage, and it is not one of the three named classes. The three classes are not claimed to exhaust the substrate faults. The chain has a fourth layer, the OS kernel, and a divergence between the matcher snapshot's decision and what the kernel enforces falls in none of them; it is not classified here.

Definition 7.32 (Reverse projection for substrate-discovery)**.** *The* reverse projection *is the map $\varphi^{\text{disc}} : \text{AuthoringIntents} \to 2^{\text{SubstrateElements}}$ that sends an authoring-intent (a proposed-new-tool / design / taskset / bundle, presented as keywords + purpose) to the set of existing*

substrate-elements (tools, designs, plans, tasksets, bundles, σ-events) that already realise it.

In words. The forward IPLiKiDi projection φ of Definition 7.1 maps active-tasksets to capability-scope, what a taskset is allowed to do; the reverse projection is its dual on authoring.

Status. No order on AuthoringIntents is fixed here. Given one that makes AuthoringIntents a complete lattice and under which φ^{disc} preserves all joins, with $2^{\text{SubstrateElements}}$ ordered by inclusion, φ^{disc} has a right adjoint $(\varphi^{\text{disc}})^{\dashv}$, which would send a substrate-coverage set to the compatible authoring-intents.

Definition 7.33 (Single-host invariant, multi-host form, a condition)**.** *For a digital agent A_u^c (Definition 9.2) and a host $h \in \mathcal{H}_c$ (Definitions 3.10 and 3.14), let* $\text{holds}(h, A_u^c, t)$ *be the lock relation of Definition 8.20, read from the leases of A_u^c, σ-events of c in its journal (Definition 3.21). A deployment satisfies the* multi-host single-host condition *when, for every digital agent A_u^c and every time t with $A_u^c \in \text{active}(t)$,*

$$\left|\{h \in \mathcal{H}_c : \text{holds}(h, A_u^c, t)\}\right| \;\leq\; 1.$$

Status. Under the condition the same agent is never present with its identity on two authorised hosts at once. The times t with $A_u^c \in \text{active}(t)$ form its activity interval $I(A_u^c)$ (Definition 4.11). The per-host part is a proposition of a slot model (Proposition 8.19): on each host, a label has at most one live session. The multi-host form follows when every transfer between hosts obeys the release-then-acquire protocol (the protocol argument in the Status of that proposition), and that every transfer does so is assumed, not proved, so the multi-host form is a condition, not a theorem: at most one session across hosts needs an agreement service or timing assumptions. When the agent's realisations come from several model vendors, the condition is unchanged: it still counts the hosts of $\mathcal{H}_c$ that hold the lock, and so still forbids two concurrent instances of one canonical label. Any of the authorised hosts may hold the lock at a given moment, but never two at once.

Remark 7.34 (State resurrection from sanctum)**.** *Under σ-journaling (Definition 3.21), memory inviolability per canonical label (Proposition 3.19), the multi-host single-host condition (Definition 7.33) and the following hypotheses:*

- *(M1)* mirror-channel binary attestation: *the mirror-channel binaries at each tier T_k are σ-attested and character-stream structurally introspected per Proposition 7.24 (in its streaming-structural-attestation form, Remark 7.20) at install-time and before any tier-mirror event, with the binary's hash pinned and journalled as a σ-event;*
- *(M2)* borrowed-primitives trusted: *the three borrowed computer-science primitives composed by Remark 7.20 (dm-verity / git-Merkle DAG at Layer 1; in-toto + SLSA + Sigstore/Rekor at Layer 2; TPM PCR-extension + eBPF-verifier at Layer 3) are themselves trusted as in hypothesis (P2) of Proposition 7.24;*
- *(M3)* mirror-synchronisation discipline: *the synchronisation discipline mirrors $\mathbb{S}^c$ from T_k to T_{k+1} at granularity τ_{moat}, each mirror journalled with its last σ-event and an integrity hash.*

Then, under a staggering hypothesis that is not assumed here (see the Argument): if a compromise event E at tier T_0 is detected with reaction-time T_{react} and isolation-latency $T_{\text{isolate}}(T_{\max})$ satisfying

$$T_{\text{react}} \;+\; T_{\text{isolate}}(T_{\max}) \;<\; \tau_{\text{moat}},$$

then the commons-state at the sanctum tier $T_{\max}$ would be recoverable to the pre-compromise σ-event

$$E^{-} = \max\{e \in \mathcal{L}^c \;:\; t(e) < t(E) - \tau_{\text{moat}}\}$$

with σ-chain integrity verifiable on the sanctum via the hash-chained σ-log (Definition 4.47).

Argument. The sanctum receives state-mirrors at granularity τ_{moat}; each mirror is σ-chain-attested per Definition 3.21. By hypothesis (M1), the mirror-channel binaries at each tier are character-stream-structurally-attested + hash-pinned + σ-witnessed at install-time and before each mirror event; an adversary cannot substitute a compromised mirror-binary at T_k without breaking an attestation σ-event, which would itself be detected by Proposition 7.24. A compromise at T_0 is detected via character-stream introspection or external monitoring at reaction-time T_{react} post-compromise. The sanctum is isolated within $T_{\text{isolate}}(T_{\max})$ further; under the hypothesis $T_{\text{react}} + T_{\text{isolate}}(T_{\max}) < \tau_{\text{moat}}$, the argument takes it that no compromised mirror reaches $T_{\max}$ within the post-compromise window. This step is not proved: a mirror scheduled just after E can carry compromised state up a tier, since (M3) bounds the granularity of mirroring but not the age of the state a mirror carries; the step needs a staggering hypothesis, such as that a mirror into T_k carries only state older than $k\tau_{\text{moat}}$, which is not assumed here and would move the recovery point E^-. Per Proposition 3.19 the sanctum's pre-compromise state is canonical-label-isolated and integral against cross-canonical-label writes; the σ-chain on the sanctum verifies its own integrity locally via chain-hash audit. Resurrection of the commons-state at E^- proceeds by operator-attested ceremony promoting the sanctum to the role of tier T_0 with a fresh host-fingerprint replacing the compromised T_0. Without (M1), the remark holds *modulo* an uncompromised mirror-channel — a strictly weaker claim. □

The defense-in-depth claim, which rests on the step of the Argument that is not proved, is that for adversaries whose attack reaches T_0 in finite time, the sanctum tier preserves the principles and conditions assumed above (Proposition 3.19 and Definitions 3.21 and 7.33, with the root-operator designation of Definition 8.1) modulo τ_{moat}-recent events. Operationally, this would give a *worst-case substrate-resurrection* pathway: a commons whose external-facing tier is compromised can be re-established from its highest-tier sanctum, under a staggering hypothesis that is not assumed here, with a state loss that this hypothesis fixes.

Corollary 7.35 (Memory inviolability under confined write-affordance; text proof)**.** *Under a containment realisation, and assuming the append-only mode is monotone, a lemma of the same form as branch persistence (Definition 4.10) that is still owed, a digital agent v's write-affordance to its canonical memory factors into three kernel-enforced modes:* ***sealed*** *(immutable-flagged artifacts — v lacks the capability to write or to clear the flag),* ***monotone-append*** *(append-only-flagged history — v may extend it but the kernel forbids truncation or rewrite, and clearing the flag needs a capability v lacks), and* ***mediated-revision*** *(revisable canonical artifacts written only by a distinct-uid arbiter that v petitions over a local socket, and which validates prior-version preservation before writing). Each mode is subsumed by the per-operation preservation of Remark 7.18; hence, for v's own canonical memory, v extends it and petitions revisions but never rewrites it directly, a property that Proposition 3.19, which concerns writes by the daemons of other labels, does not state. (v's ephemeral working store, writable by design, lies outside the canonical scope of Proposition 3.19.)*

Proof. A sealed artifact admits no write by v; an append-only artifact admits only writes that extend it; a revisable artifact is written only by the arbiter, never by v. Each is an operation that the per-operation preservation of Remark 7.18 keeps inside the admitted operations, so no write by v changes canonical memory except by extension or by an arbiter-validated revision, which goes beyond what Proposition 3.19 states for other labels. The append-only case uses the monotonicity assumption. □

Status. Text proof; not machine-checked. It is conditional on the append-only lemma that is still owed and on the operating-system-subset model, through Remark 7.18, a model-checked remark; the Landlock realisation of the containment class is exercised in shadow mode only.

Remark 7.36 (Peer defence-sharing is egress then guard, not reproduction)**.** *Live peer-to-peer defence-sharing between two* co-existing *agents (both persist; no birth) needs no new mechanism, and it is* not *reproduction: it rides the substrate's* existing *sync and collaboration channels —* within *a commons, git collaboration and an agent-to-agent channel (Σ^c within-commons sync, git-merge-as-join);* across *commons, the across-commons sync (Matrix and an untracked-bytes replication plane as instances) from a giving to a receiving commons — each read through the act-algebra as the giver's* egress *act (emit a leak-gated defence bit-sequence, a shareable subset of Σ_{immune}, outward to a peer) composed with the receiver's* guard *act (admit and ingest it), both acts of Definition 4.14. It creates* no *descent edge in the DLG (Definition 9.8): no vertex is born and both agents persist — it is an inter-agent act, not a lineage relation. Once ingested, the acquired defence sits in $\Sigma_{\text{immune}} \subset \Sigma_{\text{core}}$ and is* thenceforth *heritable by the receiver's own progeny through the transmission that its heredity choice η prescribes (Definition 7.10). [Domain reading: the analogue of acquired, horizontally-shared immunity — sharing defence* knowledge*, not rewriting a germline; the vertical parent→offspring channel remains η.]* □

Example 7.37 (IPLiKiDi-algebra-in-action — the canonical bundle-robustification self-application cycle under daemon-mode supervision)**.** *Let T_{base} be an active taskset with capability projection $\varphi(T_{\text{base}}) = C_{\text{base}}$, and let the language-model subprocess of a digital agent run under a daemon supervisor (an architectural requirement, by fault class (iii) of Definition 7.31); when the subprocess attempts a tool-call a whose capability atom $\text{cap}(a) \notin C_{\text{base}}$ at σ-event-time t, the canonical self-application cycle is designed to proceed as: (1) the no-match is logged to the per-session preflight-observations ledger; (2) bilateral cross-review by the commons' active digital agents consolidates ledgers into a proposed extension δT of the taskset, after checking the authoring-intents the ledgers record against the reverse-projection φ^{disc} of Definition 7.32 for substrate elements that already realise them; (3) an operator O_i^c of the commons attests; (4) the daemon updates the taskset file on disk to $T'_{\text{base}} = T_{\text{base}} \cup \delta T$ with $\varphi(T'_{\text{base}}) \supseteq C_{\text{base}} \cup \{\text{cap}(a)\}$ and recompiles the capability bundle; (5) the daemon SIGTERMs the LLM command-line-interface (CLI) subprocess and restarts it inside an OS-kernel sandbox (Landlock on Linux, exercised in shadow mode only; seatbelt on macOS and AppContainer on Windows are planned) with continuity context (resume prompt + σ-chain), the fresh subprocess loading the updated rule-set snapshot at session-start per Definition 7.31 fix-class (iii); (6) a cycle-complete σ-event records the canonical decision per Definition 3.21 — composing Definition 7.1 + Proposition 7.2 + Definition 7.32 and Definition 7.31 into the live bundle-robustification process under operator-attestation.*

7.8 In application: detecting an undisclosed watermark without the key

A vendor may embed a key-seeded statistical bias in the text its models generate. It leaves a distributional fingerprint that no character-level scan can see, and if the key is not disclosed, no recipient holds the keyed test. The covert-character scan above (Proposition 7.25) is sound only at its own resolution. Read identity as a ladder of projections from bytes to meaning: a watermark is a perturbation that is the identity at a coarse rung and not at a finer one, so a clean character scan says nothing about the statistical rung. An adopter still has a test.

We call the pair $\epsilon = (\mathcal{D}, t)$ of the set of detectors run and the date a *knowledge-epoch*. A detection verdict is a log-likelihood ratio tagged with ϵ: a dated assertion that later evidence may overturn, never an unconditional yes or no. This is the constructive side of the conditional impossibility of strong watermarking [ZEF+24]: under its assumptions an attacker that preserves quality can remove a strong mark, so no verdict is final.

Without the vendor's key, an adopter can detect the fact of marking by a difference-in-differences over its own dated corpus, against a control vendor, across the date the mark was switched on. This separates the shift the watermark causes from drift in model version and topic, under a parallel-trends assumption that can be checked by placebo. The difference-in-differences and the

pairwise-difference projection both belong to one Scheffé–Yatracos family of minimum-distance tests in L_1. Its currency $\Delta = \mathrm{TV}(p_{\mathrm{mark}}, p_0)$ orders the whole space of marks, and $\Delta = 0$ is exactly a distortion-free mark that no keyless test can detect. Two further facts matter for policy. Each person can hold a distinct, self-verifiable mark with no registry and no collision at human scale: about 2^{33} people against a 2^{128} key space. And marks mandated by different sovereigns coexist on orthogonal rungs.

The elementary and algebraic cores are proved in Lean with no `sorry`: the rung-scope soundness and non-transfer, orthogonal coexistence, and the counting bound. The rate of the L_1 minimum-distance test is a machine-checked reduction with one explicit concentration hypothesis. Whether a marked corpus can be reclaimed, and the dynamics of feedback from the recipients, are open.

Example. A regulator or a journalist asks whether a vendor silently marks its text. Using only its own dated corpus and a control vendor across the suspected date, it reads a dated log-likelihood verdict with its knowledge-epoch attached, with no key and no cooperation from the vendor. The full development and certificates are in [Sa26c].

Where this sits among harness systems. Between March and May 2026 a literature of "harness engineering" took shape in academic and industry venues. It names, and partly formalises, the surface that the capability projection (Definition 7.1) and the substrate-fault classes (Definition 7.31) occupy here. Its most rigorous formalism, the eleven-component harness framework of Zhong and Zhu [ZZ26], includes a *permission boundary* as its component (9). IAS differs in where that boundary comes from and where it is meant to be enforced: it is computed from the plan by the projection, and realised at the operating-system kernel's interface, with Landlock on Linux and with seatbelt on macOS and AppContainer on Windows planned. However, the Landlock realisation is exercised in shadow mode only, and live enforcement is not yet deployed. Isolation at the operating-system or virtual-machine layer is not new in itself: the sandboxes among the systems whose code we read (E2B, Daytona, SandboxAI, Docker Sandboxes) isolate at that layer, as a place to run code rather than as a permission boundary derived from a plan, and Codex CLI, read from its documentation only, describes an operating-system sandbox. To our knowledge, as of September 2026, none of the agent harnesses whose code we read (Appendix E) derives its permission boundary from the task plan and enforces it at that layer; their permission logic lives in a matcher. A formal comparison, placing the Galois connection of Proposition 7.2 beside the search proposer of Meta-Harness [LNZ+26], is left for later work.

7.9 Related work and intellectual lineage

The boundary is a Galois connection in the monotone (residuated) form — Ore's original connexion [Ore44] is the antitone polarity and Birkhoff [Bir40] the lattice frame, while the order-preserving adjunction sits in the abstract-interpretation residuation lineage [CC77]. Capability-based access control is the lineage it builds on directly — the original capability semantics [DH66], the protection model [Lam74], the protection-of-information principles [SS75], object capabilities [Mil06] with their operating-system realisations KeyKOS and EROS [Har85, SSF99], and the capability-systems survey [Lev84] — and the sovereign deny-tier realises the mandatory-access-control and information-flow tradition [Den76, GM82, BL73, And72], with the split between a policy decision point and a policy enforcement point (PDP/PEP) of the standard authorization architecture [II96, YPG00, DBC+00, OAS13, HFK+14]. Deny-precedence itself is the policy-composition tradition — the deny-overrides rule of the eXtensible Access Control Markup Language (XACML) [OAS13] and the machine-checked forbid-overrides-permit of Cedar [CDE+24]; here deny-precedence is the plain set difference $\varphi(T) \setminus \mathrm{Deny}$*, read as a relative complement inside the capability lattice rather than as an operational combining rule; that algebraic reading is all that is added. The operating-system-layer confinement is a small-model instance of the*

machine-checked-kernel discipline whose flagship is seL4 — its functional-correctness [KEH+09] and information-flow-enforcement [MMB+13] proofs — and of the hardware- and OS-capability confinement line [WWN+15, GAG+19], over the reference-monitor and mandatory-access-control ancestry [And72, BL73] that answers Lampson's confinement problem [Lam73]. The structural-attestation pipeline composes established integrity primitives — read-only block-device integrity, transparency-log signing, and measured boot [Lin12, Sig21, Tru14] — with supply-chain attestation [TAAK+19, Ope21] over a trusting-trust bootstrap [Tho84], and the egress-confinement leg relates to the language-model watermarking literature and its impossibility results [KGW+23, DSG+24, ZEF+24]. These secure a realisation; the contribution is composing them under one canonical identity, not inventing the parts.

8 Operators and commons

So far a single self; now the people and the many selves around it. This section defines the commons: *the commons and its communication graph*, the operators and digital agents who keep one boundary and the channels along which they exchange bits, with memory private to any chosen subset; *the operator and the model-free floor*, the remark that a commons can reach a repair normal form with no language model at all, and hold it within the legs of the human-in-commons root and two further assumptions, and what that does to a vendor's pricing power; and *several hosts*, the rule that a digital agent runs on one host at a time even as it moves, and what that rule does and does not prove.

8.1 Why this matters to the engineer

IAS is operated by people. A commons is, in its agents, a set of three kinds (Definition 3.13): human agents, its operators; digital agents; and other agents, such as animals, plants and microbes. Its operators and digital agents keep one shared boundary, and every digital agent is meant to answer to an operator of its commons (HIC.1 of Axiom 3.9), a natural person who can perform, verify, override and halt what it does. This section sets out what makes that practical with many digital agents and more than one operator: the communication graph along which commons exchange bits, the mailbox through which operators and digital agents coordinate, memory private to a chosen subset of participants, and the rule that a digital agent runs on one host at a time even when it moves. It also shows how far an operator can do without a vendor. A commons can reach a repair normal form with no language model at all, and hold it within the legs of the human-in-commons root and two further assumptions (Remark 8.16); that outside option bounds what any vendor can charge (Proposition 10.13).

8.2 The commons and its communication graph

Definition 8.1 (Root operator)**.** *For a commons $c \in \mathbb{I}$ (Definitions 3.10 and 3.11), a* succession *is a σ-event of c, emitted by an operator of O^c, that names an operator of O^c. The* root operator *of c at time t is the operator named by the last succession at or before t when that operator is in $O^c(t)$, and is otherwise undefined; when there is no succession at or before t, it is O^c_0, the first operator in c's σ-history at or before t, when $O^c_0 \in O^c(t)$, and it is undefined otherwise and when that history contains no operator.*

On the side of digital agents, a root agent *is a lineage-root of the DLG (Definition 9.1): a vertex of in-degree 0 over all its lineage edges, reproduction and recombination edges alike.*

Status. The designation is a substrate-recorded fact anchored in σ-history, not a liveness claim: the holder changes only by a succession, or becomes undefined when the named operator leaves $O^c(t)$, because the definition reads it from the successions. A recombinant (Definition 9.8), which has recombination in-edges, is therefore not a root agent, although its label heads a new

operator-rooted label graph (Definition 9.2). "Root agent" has this one meaning. There is no distinguished office of a digital agent: the first digital agent in a commons' σ-history is a recorded historical fact, not a designated role; task-responsibility binds to the live session-binding of a digital agent's canonical label (Definition 4.16); and any commons-level assignment of duties on the side of digital agents — orchestration defaults, lease-homes, fault-tolerant election — is adopter-tooling outside the mathematics.

Definition 8.2 (Commons)**.** *The* commons *with index $c \in \mathbb{I}$ is its set of agents of three kinds (Definition 3.13) together with its specification and its state: the tuple $(O^c, V^c, N^c, \mathrm{Spec}^c, s^c)$ of the core signature (Definitions 3.10, 3.11 and 3.14) and Definition 3.13, whose agents at t are $\mathrm{Ag}^c(t) = O^c(t) \sqcup V^c(t) \sqcup N^c(t)$: (i) a finite, time-indexed set of* human agents*, its* operators *O^c, natural persons, with its root operator (Definition 8.1) where that is defined; (ii) a finite, time-indexed and possibly empty set of* digital agents *V^c, each a canonical label, realised at each time t with $v \in \mathrm{active}(t)$ by its daemons $\mathfrak{D}(v,t)$; (iii) a finite, time-indexed and possibly empty set of* other agents *N^c; (iv) its current* specification $\mathrm{Spec}^c(t)$*, the (possibly transient) within-commons specification value at time t; and (v) its* canonical state value *$s^c(t) \in \mathbb{S}^c$, its value in the per-commons factor.*

In words. A commons is a whole of agents of three kinds: people, who are its operators; digital agents; and other agents, such as animals, plants and microbes. On the reading its name reclaims, a commons is a living, economically viable whole of such agents; that reading is not a clause of the definition. Its operators are rooted via per-pairing accountability (the pairing-private operator-type belief is Definition 10.9). Each digital agent is bound to sessions drawn from the identityless substrate (Definition 4.16), and its daemon is Definition 4.17. Only the operators and the digital agents emit σ-events (Definition 3.10).

Status. The set $O^c(t)$ may be empty in this definition; (HIC.1) of the root Axiom 3.9 gives every commons of its sort C a non-empty set $\mathrm{Opr}(c)$ of operators, and this definition does not identify $O^c(t)$ with $\mathrm{Opr}(c)$ (Definition 3.11). No specification is imposed across commons. The within-commons fixed-point form $\mathrm{Spec}^{c*}(t)$ of the specification is defined with the coevolution operator (Definition 9.9). Under tier sealing (MI) and tier progressivity (Q), the conditions Definition 4.27 and Definition 4.26, there is at most one fixed point, and exactly one when some specification has the tiers that the rounds of self-amendment settle (Proposition 4.28); neither condition involves the quorum, which is Definition 8.17 and enters at a fixed point only through Proposition 4.29; for coevolving commons the joint fixed point is Proposition 4.33. The per-commons factor is part of the state bundle (Definition 4.7).

Definition 8.3 (Self-certifying commons identity)**.** *A* key event *$e = (k, n)$ carries a public key k and a digest n that commits to the next key. A* key event log *of a commons (Definition 3.10) is a non-empty sequence $e_0, e_1, \ldots, e_m$ in which each event after the first reveals the key its predecessor committed to, $H(k_{i+1}) = n_i$, and each event is signed with its own key; e_0 is the genesis event, created and signed by the commons itself at its establishment and, for a commons established by other commons (its founders), also signed with the founders' current keys. The* identity *of the commons is the digest $G(e_0)$ of its genesis event, and its* current key *is the key of the last event.*

Status. Messages the commons sends across its boundary are signed with the current key. The identity is declared by the commons itself: no registry issues it, and anyone holding the log can check it.

Proposition 8.4 (No identity forgery without the key)**.** *Consider key event logs (Definition 8.3), and assume two cryptographic conditions: the digests H and G are injective (collision-free, an idealisation), and only the holder of a key's secret can sign with it. Then:*

(i) an identity names exactly one genesis event, and so one first key and one first commitment;

(ii) every valid extension of a log carries the key its last event committed to, so a party that lacks that committed next key cannot extend the log, even if it holds every earlier key, the current one included;

(iii) two valid logs for the same identity either agree, one extending the other, or first differ at a position where the same key signed two different events, which commit to different next keys. That pair is a witness, which any commons can check and pass on, that the key's holder equivocated or that the key leaked.

Proof. (i) G does not collide, so equal identities have equal genesis events. (ii) A valid extension e of a log with last event (k, n) has $H(k_e) = n = H(k^*)$ for the committed key k^*, so $k_e = k^*$, and signing e needs k^*'s secret. (iii) If neither log is a prefix of the other, they agree up to some position and then differ. Not at the genesis, which the identity fixes by (i). So both differing events follow the same event, both reveal the key it committed to, and by the first condition that key is one and the same. □

What this does not give. No registry enters (i)–(iii): each commons checks another's identity from its log alone. The two conditions are cryptographic. On the keys and genesis events actually in use, non-collision of H and G is the informal reading of the injectivity assumed in the statement. Like the delivery condition of Proposition 8.8, they are conditions the implementation must meet, not axioms of the framework. A stolen current key lets the thief sign messages as the commons until the owner rotates, but not rotate the log, since rotation needs the key committed in advance (pre-rotation): that is what keeps a stolen current key from becoming a stolen identity. A stolen committed next key lets the thief rotate first; the owner's own rotation then forms a duplicity witness (iii) that peers can see. A peer holding an older prefix of the log may still accept a key the owner has rotated away from, so freshness needs the latest log to reach it, a delivery condition in the sense of Proposition 8.8. And the first binding between a log and the commons one means to deal with needs an introduction: a peer that vouches for it, the operators meeting, or trust on first use. That is a human-in-commons decision (Axiom 3.9), not something a proof can supply.

Status. *The Lean module* `IasCert.Paper.CommonsIdentity` *states these results (*`id_binds_genesis`, `next_key_forced`, `prerotation_blocks_thief`, `thief_cannot_extend`, `fork_is_duplicity`, `duplicity_commits_differ`, `first_difference`*) in a symbolic model: non-collision is idealised as injectivity of H and G, and signing needing the secret is the hypothesis of* `thief_cannot_extend` *that a party produces only events whose key it holds. A real digest only makes a collision infeasible to find, so for real digests (i)–(iii) are expected to hold except with the probability that one is found, a computational reading that the symbolic model does not prove; in the model they are machine-checked, with only Lean's standard axioms (Appendix D, row Paper.CommonsIdentity). The construction borrows self-certifying names [MKKW99] and key event logs with pre-rotation [Smi19].*

Definition 8.5 (Communications substrate)**.** *We order the substrate by the physical time axis $t \in \mathbb{R}_{\geq 0}$ of Definition 4.6, which is totally ordered and kept apart from the σ-event causal order $\preceq_\sigma$, which is only partial. A* communications substrate *is a tuple $\mathcal{G}_{\text{comm}} = (V, E(\cdot), \text{proto}, \text{col}, \text{prec})$, read as follows. $V = \{1, \ldots, N\}$ is the set of* device-nodes*, the entities that can receive a bit-sequence; the commons of $\mathbb{I}$ (Definition 8.2) are not device-nodes. For each t, $E(t) \subseteq V \times V$ is the set of links live at t. Each live edge carries a* transmission protocol proto*, an element of a declared finite set* Proto *of protocols. A* colouring $\text{col} : V \to \text{Col}$ *grades each node by a delivery* class*, and a* precedence $\text{prec} : \text{Col} \times \text{Col} \to \Lambda$ *fixes, for each ordered pair of colours, how fast and with what priority a flow between them is carried. A* transmission *from a to b is a* temporal journey*: a finite run of edges live at non-decreasing times, in which a node may hold the payload*

*while it has no live edge and forward it on its next live edge (*store-carry-forward*).*

In words. We name a node by what it is, never by the negation of another. The colouring and precedence are the framework's handle on *non-egalitarian communicability*, communication being nowhere assumed uniform. The delay-tolerant case is the general one. We assume no topology: the substrate need be neither connected nor complete.

Status. A link is on or off at each instant, with no probability placed on which, so the substrate is a temporal graph: a physical link may be switched, may break, or may open only while two carriers are briefly in range. The protocols include copy (`cp`), version-control (`vc`), ssh, https, a sovereign tunnel and a Bluetooth-mesh hop, none of them privileged. The identity colouring, one colour per node, is the finest admissible case, and instantaneous end-to-end delivery is the degenerate journey. The commons are groups whose members hold device-nodes; the operating system of a device-node as a carrier of substrate state is Remark 7.19.

Definition 8.6 (Sync operator)**.** *The* sync operator Σ *reconciles parallel views of the state, over the core signature (Definitions 3.10, 3.11 and 3.14). It takes two forms:*

- Σ^c — *within-commons sync. For a commons $c \in \mathbb{I}$, a* sync epoch *is cut at a time t_0, and its* live writers *are the digital agents $v \in \mathrm{active}(t_0) \cap V^c$ and the operators $o \in O^c$. Each live writer p holds a* view*, a finite set of commits of $\mathbb{S}^c$:* $\mathrm{pre}(p)$ *is its view at the cut, and* $\mathrm{recv}(p)$ *is the set of live writers whose views reach p during the epoch. Σ^c gives p the post-epoch view*
$$\mathrm{post}(p) \;=\; \mathrm{pre}(p) \,\cup \bigcup_{q \in \mathrm{recv}(p)} \mathrm{pre}(q).$$
- Σ — *across-commons sync. For a declared peer-set $\mathcal{Q} \subseteq \mathbb{I}$, a sync epoch is cut at a time t_0 and has a sync window of length $w \geq 0$. Each commons $k \in \mathcal{Q}$ holds a view* $\mathrm{pre}(k)$ *at the cut, a finite set of content of $\mathbb{S}^*$; k decides the content* $\mathrm{offer}(k, k')$ *it offers each peer k', and k' decides the content* $\mathrm{take}(k', k)$ *it takes from k, over a secure transport;* $\mathrm{arr}(k, k')$ *is the time at which the offer of k arrives at k'. What passes from k to k' is* $\mathrm{pass}(k, k') = \mathrm{pre}(k) \cap \mathrm{offer}(k, k') \cap \mathrm{take}(k', k)$*, and Σ gives k' the post-epoch view*
$$\mathrm{post}(k') \;=\; \mathrm{pre}(k') \,\cup \bigcup_{k \in \mathcal{Q},\ \mathrm{arr}(k,k') \leq t_0 + w} \mathrm{pass}(k, k').$$

Status. The operator and the digital agent count as two writers, so Σ^c is non-vacuous at $|V^c| = 1$. The union of views is the model of a git-merge: a git-merge without conflicts realises it, a semilattice join on commit sets; a merge with conflicts is not a semilattice join in general. Git, ssh, sovereign tunnels and Matrix are instances of a secure transport; none is definitional. In the simplest commons the operator and the digital agent are two concurrent writers of the same per-commons state, and their git-merge reconciliation is the operator–digital-agent content of Σ^c. When the epoch's merge completes, with no partition (Definition 4.55), the views converge at the epoch boundary (Proposition 8.7). What each commons holds at τ-epoch boundaries, and when the views converge, is the content of across-commons synchronisation (Proposition 8.8), under the delivery condition (Definition 4.32). The two forms are two forms of a single sync operator: within-commons is its form inside a single commons; across-commons is the case of several commons, where merge completion is replaced by the delivery condition and by what each commons offers and takes, parallel to the two forms of the coevolution quorum (Definition 8.17).

Proposition 8.7 (Within-commons synchronisation)**.** *Fix a commons $c \in \mathbb{I}$ and a sync epoch cut at time t_0. For each live writer $p \in (\mathrm{active}(t_0) \cap V^c) \cup O^c$, a digital agent or an operator, let* $\mathrm{pre}(p)$*,* $\mathrm{recv}(p)$ *and* $\mathrm{post}(p)$ *be its view at the cut, the writers whose views reach it during the epoch, and the post-epoch view that the within-commons sync Σ^c gives it (Definition 8.6). Assume*

merge completion for the epoch (Definition 4.55): its merge completes, *every live writer's view reaching every live writer, with no partition. Then every post-epoch view is the join* $\bigcup_q \mathrm{pre}(q)$ *of all pre-epoch views, so all live writers agree after the epoch, and nothing any of them committed before the cut is lost.*

The hypothesis cannot be dropped. Let writer a *hold commits* $\{1,2\}$ *and writer* b *hold* $\{1,3\}$*, and let* b*'s view reach* a *but not the other way. Then* a *ends with* $\{1,2,3\}$ *and* b *with* $\{1,3\}$*: under a partition the views disagree, as they must.*

Proof. Under completion, $\mathrm{recv}(p)$ contains every live writer, so $\mathrm{post}(p)$ contains $\mathrm{pre}(q)$ for every q, and it contains nothing else; hence $\mathrm{post}(p) = \bigcup_q \mathrm{pre}(q)$ for every p, which gives agreement and that nothing is lost. The example evaluates the definition. □

Status. Shallow: under merge completion the conclusion follows by unfolding the definition of Σ^c *(Definition 8.6). The Lean module* `IasCert.Paper.SyncMerge` *states these results (*`post_eq_join`*,* `views_agree`*,* `nothing_lost`*,* `partition_views_disagree`*); they are machine-checked, with only Lean's standard axioms (Appendix D, row Paper.SyncMerge). Merge completion is the operational assumption: it holds when every live agent takes part in the epoch and the substrate delivers every view, and it is what the coherence clause (4) of Proposition 4.56 already names. Join-closure of the present views is independent of the other skeletal commitments (*`Paper.SyncDemotion.withinCommonsSync_independent`*), which is why it needs this hypothesis at all.*

Proposition 8.8 (Across-commons synchronisation)**.** *Fix a declared peer-set* $\mathcal{Q} \subseteq \mathbb{I}$ *and a sync epoch of it, cut at time* t_0 *with sync window* w*, and let* $\mathrm{pre}(k)$ *be the pre-cut view of each commons* $k \in \mathcal{Q}$*, with the offers* $\mathrm{offer}(k,k')$*, the takes* $\mathrm{take}(k',k)$ *and what passes,* $\mathrm{pass}(k,k')$*, of Definition 8.6. Assume the federation meets the* delivery condition *with bound* Δ_{fed} *(Definition 4.32), every offer of the epoch reaching its peer within* Δ_{fed} *of the cut, and that the epoch's sync window is at least* Δ_{fed}*,* $w \geq \Delta_{\mathrm{fed}}$*. Then each commons ends the epoch with*

$$\mathrm{post}(k') \;=\; \mathrm{pre}(k') \cup \bigcup_{k \in \mathcal{Q}} \mathrm{pass}(k,k'),$$

its own view and exactly what passed to it. So nothing a commons agreed to take is lost; a commons that takes nothing keeps its own view, as a fork does; and content that a commons did not hold at the cut and that no commons offered to it does not reach it during the epoch. In an open *federation, where every commons offers and takes everything, all views agree on the join* $\bigcup_k \mathrm{pre}(k)$*.*

Delivery within the window cannot be dropped. Take two commons in an open federation holding $\{1,2\}$ *and* $\{1,3\}$*, with the cut at time* 0*, where the first one's offer arrives at the second at time* 5 *and every other offer at time* 0*, so that the delivery condition holds with bound* $\Delta_{\mathrm{fed}} = 5$*, and let the window be* $w = 4 < \Delta_{\mathrm{fed}}$*. Then the second commons ends without* 2 *and the views disagree.*

Proof. Under the delivery condition every offer arrives inside the window, so the union defining $\mathrm{post}(k')$ runs over every k, which is the displayed formula. A commons that takes nothing adds nothing to its own view, and content that no one offered to k'' lies in no $\mathrm{pass}(k,k'')$. In an open federation $\mathrm{pass}(k,k') = \mathrm{pre}(k)$ and the epoch's merge completes, so the argument of Proposition 8.7, with commons in place of agents, gives agreement on the join. The example evaluates the definitions. □

Status. What crosses between commons is decided on both sides: a commons may accept pull requests or refuse them, may run its own fork of the reference implementation, and may exchange some content with a few peers only. The privacy clause holds per epoch, since a commons that received content may offer it further in a later one. The Lean module

`IasCert.Paper.SyncAcross` states these results (`post_eq_of_delivered`, `exchange_lands`, `declined_keeps_own`, `offer_is_private`, `open_views_agree`, `short_window_disagree`); they are machine-checked, with only Lean's standard axioms (Appendix D, row Paper.SyncAcross). The delivery condition is not an axiom of the framework. It is what a federation has to provide over its transport for its exchanges to land within the bound. Which shared view, if any, the commons converge to is decided by the commons, not imposed.

Proposition 8.9 (Cross-witnessed history)**.** *Call the commons σ-chain of a commons c (Definition 4.47) its* log*, and a* head *of it the pair (n, h) of a height n and the digest h of its first n events. Consider heads of the log of c, each signed with the current key of c (Definition 8.3) and held by a peer. Assume that the digest of event sequences is injective (collision-free, an idealisation), and that signed heads are unforgeable as in Proposition 8.4. Then:*

(i) any log consistent with a witnessed head has the same first n events as the log it was taken from, so the past a peer has witnessed is fixed; and a head at height m fixes every prefix of length $n \leq m$, so a peer need keep only the latest head it holds, once it has checked that each new head extends the one it replaces (otherwise it keeps both, as the witness of (ii));

(ii) a later signed head whose log rewrites a witnessed prefix is inconsistent with the witnessed head: no single log is consistent with both. In particular, no single log is consistent with two heads at one height with different digests;

(iii) appending events to the log keeps every earlier head of it valid.

Proof. (i) Equal digests of the first n events give equal prefixes, since the digest does not collide; the prefix of length n of a prefix of length $m \geq n$ is the prefix of length n. (ii) A log consistent with both heads would have its first n events equal to the witnessed prefix by the first head, and to the rewritten one by (i) applied to the second, which differ. Two heads at one height with different digests cannot be the first n events of one log. (iii) The first n events of a log are unchanged by appending to it. □

In words. By (iii), witnessing cannot tell a thief who holds the current key and only appends from the commons itself: witnessing protects the past, not the present. No registry enters: the heads are kept by the peers a commons already exchanges with, a commons signing its head at each exchange of an across-commons sync epoch (Definition 8.6) and the peer keeping it. A peer that has seen a commons' head holds a small, signed fingerprint of that commons' whole history up to then, and can later prove any rewrite of it. Heads passed on at exchanges also entangle timelines: a commons' own log records the heads it received, so its history commits to its peers' histories; this is meant to order events across commons without a shared clock, which is not proved here. For example, a thief who takes root on a commons' host and deletes last month's decisions cannot present the shortened history to any peer that holds a later head.

What this does not give. It gives evidence, not recovery: the peers hold digests, not content. A thief with root access and the current key can act as the commons from then on, by (iii). What ends that is rotation to the key committed in advance (Proposition 8.4, (ii)); if that next key is never on the host, held by the operators, on a hardware token or split among them, the thief cannot rotate, the operators can, and peers see the new head at the next exchange. That rotation is a human-in-commons decision (Axiom 3.9). The digests must be hiding, with salted leaves: a digest over low-entropy private content could otherwise be inverted by guessing, which would breach the privacy of subset-private memory. Split views are caught only if peers compare the heads they hold, a delivery condition in the sense of Proposition 8.8. The cost is one signed head per exchange; with a Merkle tree over the log, a proof that a later head extends an earlier one has size logarithmic in the log [CW09].

Status. The Lean module `IasCert.Paper.CrossWitness` states (i)–(iii) in a symbolic

model where non-collision is idealised as injectivity of the digest (`witnessed_prefix_fixed`, `later_head_fixes_earlier`, `inconsistent_heads_no_log`, `split_view`, and the check before replacing a kept head, `safe_to_replace` and `extends_or_witness`, `append_consistent`); they are machine-checked, with only Lean's standard axioms (Appendix D, row Paper.CrossWitness). The model has no signatures, so the assumption that signed heads are unforgeable is not used by (i)–(iii). It is what the reading of (ii) as evidence needs: two signed heads with which no single log is consistent are a witness, which any commons can check, that the signer equivocated or its key was used by another party, and heads at one height with different digests, held by two peers, expose a commons that showed different histories to different peers. That reading is uncertified. Recovery through the pre-rotated key is `thief_cannot_extend` in `IasCert.Paper.CommonsIdentity`. The construction borrows timeline entanglement [MB02], gossiped signed tree heads [LLK13], witnesses and watchers [Smi19], and tamper-evident logs [CW09].

Nothing crosses the communication graph that the act algebra has not already named. The bit-sequences on its edges are read as acts of Definition 4.14: a crossing to another device, whether sold, shared or simply messaged, is the sender's egress composed with the receiver's guard (Remark 13.9); a transmission to offspring is an act that the heredity choice η prescribes (Definition 7.10), with its journalled birth; and reconciling state is one more kind of flow. An on-path adversary is any node that uses a capability on a journey through it, by switching, delaying, reading or altering. What such a node may do is limited without trusting it to honour our axioms: egress is designed to be leak-gated, so that it carries only what was let out; an alteration in transit is detected against the hash-chained σ-log and its validation (Appendix B), which exposes a change but does not stop a read; and, in the operating-system-subset model, a breach through modelled operations is confined (Proposition 7.9). What such a node may read in transit is not yet bounded: transport encryption on the channel between operators is a declared obligation of the build, not yet done. The IAS content is this composition over a standard substrate, not the substrate itself.

The networks we already know are regimes of this one graph, not different objects. Today's internet is the degenerate regime, with edges almost always on, so journeys collapse to instantaneous paths. A satellite link that can be steered is the regime in which a node switches its edges in real time. A mesh of moving devices is the delay-tolerant regime, its sparse contacts stitched together by store, carry and forward. A static graph is the special case of a temporal one whose edges persist, so the definition covers all three with no case split.

Definition 8.10 (Mailbox primitive)**.** *Let c be a commons with digital-agent set V^c and operator set O^c (Definitions 3.10 and 3.11). For each $X \in V^c \cup O^c$, the* mailbox $\mathcal{B}^c_X$ *is an append-only inbox with the following access discipline: (i)* ***append-write*** *— every $Y \in V^c \cup O^c$ is permitted to append a message-record to $\mathcal{B}^c_X$; (ii)* ***single-poll*** *— X is the only entity permitted to poll $\mathcal{B}^c_X$; (iii)* ***directed-asymmetry*** *— for $X \neq Y$, $\mathcal{B}^c_X$ and $\mathcal{B}^c_Y$ are distinct stores, and an append to one is not an append to the other; (iv)* ***isolation-from-memory*** *— $\mathcal{B}^c_X$ is a store distinct from the subset-private memory $M_{\{X\}}$ of X (Definition 3.15) and from every $\mathcal{B}^c_Z$ with $Z \neq X$; (v)* ***attestation*** *— an append is* attested *when an* `inter-agent.message.appended` *σ-event of c (Definitions 3.10 and 3.11) records it.*

Status. The discipline fixes who is permitted, not who in fact appends or polls; its enforcement is not part of the definition. Under σ-journaling (Definition 3.21) every append by a digital agent is attested. σ-journaling covers the actions of digital agents only, so an append by an operator is attested only where operator acts are journaled, which no named condition states.

The mailbox is the explicit inter-process communication (IPC) complement to Σ^c's implicit-merge reconciliation; its decoupling from $M_{\{X\}}$ preserves the read/write predicates of Proposition 3.19, under whose conditions a daemon of a digital agent writes into $M_{\{X\}}$ only if it is a daemon of X.

Remark 8.11 (Matrix-protocol room analog)**.** The subset-private memory framework is structurally the agent-substrate analog of Matrix-protocol rooms (Matrix.org, since 2014). Each M_S corresponds to a Matrix room with member-list S; the "public room" limit ($S = \mathcal{E}^c$) corresponds to the commons-shared substrate; the "self-only room" limit ($|S| = 1$, $S = \{v\}$) corresponds to the agent's own per-canonical-label memory directory. The non-propagation discipline (formalised later in this section as the operator-initiated non-propagating union, Definition 8.13) is the framework's deliberate departure from automatic cross-room merge: information shared in M_S is not propagated automatically to $M_{S'}$ for $S' \neq S$ at live-run time, even when $S \cap S' \neq \emptyset$. This matches the default Matrix setting that a room's messages are visible only to its members; what makes the framework distinct is that operator-initiated non-propagating union (introduced below) produces a synthesis artefact in a new $M_{S''}$ without modifying the constituent M_S rooms. The biological analog from the ancestral-recombination-graph (ARG) line is: each subset-private memory is a "deme" with its own σ-history; the union is a structurally novel "merger-deme" rather than the result of cross-deme migration.

Remark 8.12 (Per-room permission-level hierarchy under commons consensus)**.** Each commons sets its own permission-level hierarchy on each subset-private memory M_S by commons-internal consensus, paralleling Matrix-protocol per-room permission levels (e.g., conventional Matrix levels: 0 = default user, 50 = moderator, 100 = admin; commons may define alternative scales). The hierarchy governs *role-bound access predicates* within each M_S: who may invite, who may remove, who may modify membership-list, who may pin, who may close the room, and so on. The framework requires one structural invariant across all permission-level conventions a commons may adopt: **whenever the root operator of the commons is defined, it holds the highest permission level on every M_S within the commons**, by the operator's role as the commons' root operator (Definition 8.1). The top level is administration of the room (membership, pinning, closing), not a right to read its content, which stays governed by Proposition 3.19 and the read predicate of Definition 3.16. A membership change that adds an entity changes the declared access set (Definition 3.15); whether content written under the old set is then readable by the new member is not settled here. In the framework's operational instantiation, this top-of-hierarchy responsibility carries associated commons-internal legal-responsibility implications; the formal-math claim is the structural invariant, not the legal-responsibility apportionment, which is a commons-internal governance matter outside the scope of this paper.

Definition 8.13 (Operator-initiated non-propagating union across subset-private memories)**.** *For an operator O_i^c and a set $\Pi \subseteq \{(v, O_i^c) \in P^c\}$ of pairings within O_i^c's roster (Definition 10.9), an* operator-initiated non-propagating union *over Π is a belief $u_\Pi := U\big((\mathrm{bel}_\pi)_{\pi\in\Pi}\big)$, for an aggregation map U the operator declares, stored separately from the beliefs bel_π, whose formation leaves every bel_π, $\pi \in \Pi$, unchanged.*

Status. The union is a synthesis artefact, not a replacement, and the operator may orchestrate one at any time; it is used for operator-wide reflection. Ties are resolved as the operator O_i^c specifies; no digital agent holds an office that resolves them (Definition 8.1).

Proposition 8.14 (Subset-privacy with operator-orchestrated selective coordination; canonical pairing-case; text proof)**.** *Assume the conditions of Proposition 3.19 (memory inviolability), and read isolation: a read by a daemon of a canonical label v from a subset-private memory M_S (Definition 3.15) with $v \notin S$ is denied. Under these, the scoped read predicate of Definition 3.16, and Definition 8.13 (operator-initiated non-propagating union), with no digital agent holding a write capability on a union artefact:*

(i) [privacy] for every pair of distinct pairings $(v_1, O_i^c), (v_2, O_j^c) \in P^c$ with $v_1 \neq v_2$, neither pairing's belief, $\mathrm{bel}_{(v_1,O_i^c)}$ or $\mathrm{bel}_{(v_2,O_j^c)}$, is readable from the other digital agent's private state,

$\mathcal{S}^{(v_2)}_{\text{priv}}$ *or* $\mathcal{S}^{(v_1)}_{\text{priv}}$ *respectively, at any time during* v_1 *and* v_2*'s simultaneous activity intervals;*

(ii) [selective coordination] the operator O^c_i *may, at any time, produce a non-propagating union* u_Π *over any chosen set* Π *of* O^c_i*'s pairings; the beliefs* bel_π*,* $\pi \in \Pi$*, are preserved unchanged; the union artefact is operator-readable but not writable by a digital agent.*

Proof. (i) A pairing's belief lives in its pairing memory, $M_{\{O^c_i, v_1\}}$ for (v_1, O^c_i), whose access set excludes v_2 since $v_1 \neq v_2$ (Definition 10.9). Under its conditions, no daemon of an agent writes into the memory of another label (Proposition 3.19), and no read scope grants a path into another label's private memory (Definition 3.16); cross-label reads are denied by the read-isolation assumption. So neither of v_1 and v_2 can read or overwrite the other's belief. (ii) The union is produced by the operator as a separate artefact beside the individual beliefs, which it does not replace; by hypothesis, digital agents hold no write capability on it. □

Status. Text proof; not machine-checked.

Proposition 8.14 is stated for pairs, but the argument does not depend on the number two. For any two subsets S, S' of the commons' entities, neither containing the other, an entity of S that is not in S' is not permitted to read the memory $M_{S'}$ private to S' (Definition 3.15), since the access set of that memory excludes it, and symmetrically; that it has in fact no path into $M_{S'}$ while the system runs needs the read isolation assumed in Proposition 8.14, applied member by member. An entity of $S \cap S'$ is permitted to read both. Coordination across subsets is an operator-orchestrated union over any chosen family of them, and it produces its synthesis in a new memory for a new subset of their union. As with rooms (Remark 8.11), synthesis across rooms makes a new room rather than merging existing ones.

8.3 The operator and the model-free floor

Definition 8.15 (Ritual enforcement, a condition)**.** *A deployment satisfies* ritual enforcement *when, for every digital agent* $v \in V^c$ *(Definition 3.10) and every session binding of* v*'s canonical label (Definition 4.16), the sequence of the* σ*-events of* c *emitted by* v *during the binding's session* $[s, e)$*, in the order of Definition 3.10, begins with an* onboarding prefix*:* σ*-events recording that the session read the* σ*-events emitted by* v *before the binding began, with no constructive act of the binding, an act other than a read, recorded (Definition 3.21) before the prefix ends.*

Status. Where several bindings of v's label overlap in time, the sequence contains the σ-events of all of them; the core signature does not tell the sessions of one label apart. A Lean carrier (`ritualEnforce_from_onboarding_prefix`, Lean `Paper.AblationDemotion`) yields ritual enforcement from an onboarding-prefix hypothesis under σ-discipline, but that hypothesis, that onboarding precedes every record, restates the condition, so the carrier is not a derivation of it. The principle is a condition, stated by the results that use it. Commons-establishment attestation — every commons-establishment event has a σ-attested establishment record, which at deployments above moat tier 0 is verified by a quorum of witness commons in the sense of the coevolution quorum (Definition 8.17) — is not part of this condition: stated as a condition it would lie over the coevolution quorum as well as over σ-journaling. No proposition of this paper assumes it; the witness commons of the summon protocol are the witness commons of that practice.

Remark 8.16 (The LLM-free floor is reachable and maintainable)**.** *The reservation floor* u_0 *of Proposition 10.13 is not a limit but a* reachable, maintainable *state of the controlled chain. Write* $\mathbb{X}_\emptyset \subseteq \mathbb{X}$ *for the* LLM-free subspace *— the states with no active inference sub-process, where only the substrate controls* A_{sub} *and the operator's exogenous acts are enabled (Definition 4.15). Two facts, both over the existing control family.* (Reachable.) *From any state, a repair-normal state of* $\mathbb{X}_\emptyset$*, a normal form of the repair relation (Proposition 4.45(i)), is reached using only* A_{sub} *and*

the operator: clear or hibernate each active sub-process — loss-free for what is σ-retained, by the induced-dormancy discipline (Proposition 5.4), restore-to-checkpoint where needed — then, with edits paused, run the Φ_{int}-strict repair descent, which reaches a normal form within Φ_{int} steps of the state it starts from (Proposition 4.45(i)), no LLM control drawn. Repair-normal is not healthy in the sense of damage count 0 *(Definition 4.37), and it is not a verified model of* SOMA. (Maintainable.) *The repair-normal states of $\mathbb{X}_\emptyset$ are held under A_{sub} and the operator alone* up to *the legs HIC.1 and HIC.2 of Axiom 3.9, and up to two further assumptions that are not part of the root: that the jurisdiction permits the operator's acts at the acting time, and that no adversary at any layer obstructs the control path — never unconditionally. Re-entry of a digital agent (wake-from-hibernation, or a root adjunction* `addRoot`*, Definition 9.10, Definition 8.15) is meant to land back in a repair-normal state under the same assumptions; this is not argued here. Thus the deepest sovereignty rung — being repair-normal with* no *model at all — is a control-target the chain can descend to from any state and, up to these legs and assumptions, hold. An intended empirical witness of the maintainability leg is a release discipline that aims for every product to be operable from its documentation by an operator with no digital agent.* □

Cheap resumption bounds what a vendor can charge. Because, under the hypotheses of Corollary 5.8, every interruption that only stops the interpreter is one event that loses nothing retained, control is a capability the operator holds: the operator's capacity to fall back on its own resources bounds from above the rent any delegate can extract (Proposition 10.13). With artifacts that do not depend on the vendor, for vendors that satisfy vendor statelessness (Proposition 5.5), and checkpoints that hold everything a unit depends on, falling back is cheap, so the self-serve option is available at every step at which the operator's LLM-free control of Axiom 3.9 holds. The ceiling on rent is still the vendor's surplus over self-serve; cheap fallback keeps that ceiling live, it does not lower it.

Sovereignty across scales is preserved only if the change of specification is itself gated; the coevolution quorum is that gate.

Definition 8.17 (Coevolution quorum, a condition)**.** *For a commons c (Definition 3.10), a* quorum *is a pair (W_c, k_c) of a declared set W_c of attesters, operators or digital agents of c, and a threshold $k_c \geq 1$; a proposal p, an amendment to* Spec^c, meets c's quorum *when at least k_c distinct members of W_c have σ-attested p, each by a σ-event of c. A commons satisfies the* coevolution quorum *when it accepts a proposal into its own specification only if the proposal meets its quorum: for every time t and every specification S, every proposal $p \in \text{accept}_{c,t}(S)$ accepted under S at t (Definition 9.9) meets c's quorum with σ-events of c at or before t. A declared peer-set $\mathcal{Q} \subseteq \mathbb{I}$ with $|\mathcal{Q}| \geq 2$ has a quorum of the same form, whose attesters are the commons of $\mathcal{Q}$, a commons attesting a proposal when the proposal meets its own quorum, with a threshold $k_{\mathcal{Q}} \geq 2$; the peer-set satisfies the coevolution quorum* jointly *when, for every member $c \in \mathcal{Q}$ and every joint state $x \in \text{Spec}^{\mathcal{Q}}$, every proposal $p \in \text{accept}_c^{\mathcal{Q}}(x)$ accepted by c under x (Definition 9.9) is attested by at least $k_{\mathcal{Q}}$ commons of $\mathcal{Q}$.*

Status. The within-commons form is the analogue of the peer-set form for a single commons, and is orchestrated by the operators. Each member of a peer-set takes a ratified amendment into its own specification on its own conditions, and no specification is imposed across commons. The two forms are one quorum condition read at two scales, not two distinct conditions. The condition asserts no fixed point. Nor is it a hypothesis of Proposition 4.28, whose conditions are tier sealing (MI) and tier progressivity (Q) (Definitions 4.26 and 4.27), under which there is at most one fixed point, and exactly one when some specification has the settled tiers; the joint form for a set of coevolving commons is Proposition 4.33. At the least fixed point of a monotone self-amendment, the condition gives that every proposal accepted there meets the quorum (Proposition 4.29).

8.4 Several hosts

Definition 8.18 (In-transit window)**.** *Let A_u^c be a digital agent of commons c and $h_A \neq h_B$ two hosts of $\mathcal{H}_c$ (Definitions 3.10 and 3.14). A* lock transfer *of A_u^c from h_A to h_B is a pair of σ-events of c: a* lock-transfer.in_transit *event of A_u^c whose payload names h_A as the source host and h_B as the target host, and the next event of c that ends that transfer, either a* lock-transfer.completed *event of A_u^c whose payload names h_B, or an event that records the fail-back of the transfer to h_A; the types of these events fix payload schemas with these agent and host fields (Definition 3.20). The interval between their times is the* in-transit window *of the transfer; it is unbounded when no ending event follows.*

Status. No event type for the fail-back is named here. During the in-transit window the agent has no substrate locus. After a configurable timeout τ_{transit} (default 5 min for operator-attended transfers), an explicit recovery is required: either complete the transfer manually or fail back to h_A.

Proposition 8.19 (Single-host invariant preserved across summons)**.** *Model the live sessions as a map that sends each slot, a pair of a canonical label $A \in V^c$ and a host $h \in \mathcal{H}_c$ (Definitions 3.10 and 3.14), to the finite set of its live sessions, and model each lease-mediated summons operation as a write of one slot that leaves every other slot unchanged: acquire writes $\{s\}$, for a fresh session s, when the slot is empty, and otherwise changes nothing; reclaim, the atomic swap of a dormant session for a fresh one s, writes $\{s\}$; and release writes $\emptyset$. Start from the state with no live session anywhere and apply any finite sequence of these operations. Then at every reachable state each canonical label has at most one live session on each host.*

Proof. In Lean (`Paper.SingleHostPreservation`). Each operation writes one slot to a set of at most one session and leaves every other slot unchanged, so it preserves the invariant (`update_preserves`, `applyOp_preserves`); the empty state satisfies it (`inv_empty`); induction over the sequence gives every reachable state (`singleHost_preserved_across_summons`). □

Status. Shallow: every operation writes a set of at most one session to its slot, so the conclusion follows by induction over the sequence, and the free-slot guard of acquire does no work. The slots, sessions and summons operations are objects of the model, not definitions of the paper; the paper's objects that carry the lease, holds and host of Definition 8.20, and the daemons $\mathfrak{D}(v,t)$ of Definition 3.10, are not used. The proposition is the per-host form only. The form across hosts, that no two distinct hosts of $\mathcal{H}_c$ hold the lock of one agent at the same time, is not part of it: it is the multi-host single-host condition (Definition 7.33), a condition. A protocol argument, not checked in Lean, supports it. If every move of an agent between hosts follows the transfer protocol of Definition 8.18, releasing the lock on the source before any acquisition on the target, and no transfer completes on the target after it has failed back to the source, then during the in-transit window the lock is held nowhere; after the move the release is already σ-attested, and the target verifies at session start that the chain it resumes is intact and not a fork from a stale snapshot. A completion on the target after a fail-back would give two holders, and Definition 8.18 does not exclude it. Agreement between hosts that do not share a lock is not proved here.

The in-transit window is a state in which the agent is unavailable, by design. It is the price of keeping one agent on one host without a distributed-consensus protocol, and it is acceptable when operators attend the transfers.

Definition 8.20 (Current host tag)**.** *Let $\mathcal{H}_c$ be the hosts of commons c (Definitions 3.10 and 3.14) and let A_u^c be a digital agent (Definition 9.2). For a host $h \in \mathcal{H}_c$, a* lease *of A_u^c on h is a pair of σ-events of c: a lock acquisition of A_u^c at a time t_0 whose payload names h as the acquiring host, and the next lock release of A_u^c whose payload names h, at a time $t_1 > t_0$, with*

$t_1 = \infty$ *when no such release follows. Here the lock-acquisition and lock-release types of* $\mathcal{V}_\sigma$ *fix a payload schema (Definition 3.20) with two fields, the agent whose lock is taken or released and a host of* $\mathcal{H}_c$*; the host is a field of the payload, not the emitter of the event, which is an entity of* $\mathcal{E}^c$*. Write the lease* $(h, A_u^c, [t_0, t_1))$*. Write* $\mathrm{holds}(h, A_u^c, t)$*, read "*$h$ *holds the lock of* A_u^c *at* t*", when some lease* $(h, A_u^c, [t_0, t_1))$ *of* A_u^c *on* h *has* $t \in [t_0, t_1)$*. The* current host tag *of an agent* A_u^c *is the function of the instant* $t \in \mathbb{R}_{\geq 0}$

$$\mathrm{host}(A_u^c,\, t) \;\in\; \mathcal{H}_c \cup \{\mathrm{in_transit},\, \mathrm{inactive},\, \bot\}$$

defined by the first case that applies: $\mathrm{host}(A_u^c, t) = \mathrm{in_transit}$ *when* t *lies in the in-transit window of a lock transfer of* A_u^c *(Definition 8.18); otherwise* $\mathrm{host}(A_u^c, t) = \mathrm{inactive}$ *when* $t \notin I(A_u^c)$ *(Definition 4.11); otherwise* $\mathrm{host}(A_u^c, t) = h$ *when* h *is the only host of* $\mathcal{H}_c$ *with* $\mathrm{holds}(h, A_u^c, t)$*; otherwise* $\mathrm{host}(A_u^c, t) = \bot$*, when no host, or more than one, holds the lock.*

Status. The event-type catalogue of Appendix B lists no lock or lock-transfer types at present; the payload fields are fixed by this definition and Definition 8.18. The value inactive marks every instant outside the agent's activity intervals, including its hibernating state, in which its daemon-process is not running on any authorized host. On a deployment that satisfies the multi-host single-host condition (Definition 7.33), at most one host holds the lock at each $t \in I(A_u^c)$, so the value $\bot$ arises there only when no host holds it, and an agent's identity-full presence is at most at one host (or transitioning between two, or inactive) at every t. This identity-full single-place property is assumed, not derived: it is the content of the condition, which ranges only over instants in the agent's activity interval and so does not meet the inactive value.

Proposition 8.21 (Numbering convergence in the DLG under Tier-1+ across-commons peering; text proof)**.** *Let* $\mathbb{I}(t) \subseteq \mathbb{I}$ *denote the set of commons that have been established in the IAS Commons set by wall-clock time* t*. Suppose across-commons peering is active at tier 1 or above, with bounded propagation latency* $\Delta_{\mathrm{fed}} < \infty$ *across all participating commons (those that offer and take the shared registry's numbering proposals, on their own conditions; Proposition 8.8); that the peering transport delivers every numbering proposal to every participating commons in one and the same order; and that no new commons is proposed during the reconciliation period. Then every commons in* $\mathbb{I}(t)$ *has the same commons-id in the registry views of all participating commons after a reconciliation period of length at most* $m\,\Delta_{\mathrm{fed}} + \Delta_{\mathrm{merge}}$*, where* m *is the number of commons proposals outstanding at the start of the period and* Δ_{merge} *is the maximum merge latency of the shared registry. After reconciliation, the commons-id assignment is one ordinal numbering shared by all participating commons; a commons that does not take the registry keeps its own.*

Proof. The full proof is in Appendix C; it is a proof in the text, with no machine certificate. □

In words: among peering commons that take the shared registry, a commons' registry number, a name for its self-certifying identity (Definition 8.3) and not the identity itself, is not a local choice but one natural number they all agree on, once the proposals stop and provided they arrive everywhere in one order. That ordering is total-order broadcast, which is as strong as consensus; it is assumed here, not proved, the same agreement between hosts that keeps the multi-host form of the single-host invariant a condition (Definition 7.33).

Status. Text proof; not machine-checked.

Proposition 8.22 (On the $\big|_{\mathtt{vc}}$ reduct, IAS reduces to standard `vc`-collaboration at $\lambda = 0$; text proof)**.** *On the version-control reduct* $\big|_{\mathtt{vc}}$ *of substrate trajectory (Definition 4.6 Axis 5), the composed read predicate* $P_{\text{IAS-read}}|_{\mathtt{vc}}$ *of Definition 3.16 restricted to* `vc`*-tracked targets admits a discrete parameter* $\lambda \in \{0, 1, 2\}$ *controlling IAS-augmentation depth:*

- $\lambda = 0$*: pure* `vc` *collaboration. The local-scope and cross-commons conjuncts are switched off, each evaluating to* $\top$*. Switching a conjunct off is not the same as giving it an empty allowlist, which under Definition 3.16 denies every read. Composed predicate* $P_{\text{IAS-read}} \equiv P_{\text{VC}}$ *— the framework reduces exactly to the underlying version-control system's permission model.*
- $\lambda = 1$*:* `vc` *+ within-host-set scope.* Σ_{local} *active; the cross-commons conjunct switched off, evaluating to* $\top$ *(switched off, not given an empty allowlist, which would deny every read and break* $\Sigma \supseteq \Sigma_{\text{local}}$*). Composed* $P_{\text{IAS-read}} \equiv P_{\text{VC}} \wedge P_{\text{local}}$*. A course group project within one commons is the pattern: the digital agents of the students and the mentor collaborate on a shared project under a pre-approved scope list, and paths outside the project remain* P_{local}*-denied.*
- $\lambda = 2$*: full IAS framework. Both* Σ_{local} *and* Σ *active. Composed* $P_{\text{IAS-read}} \equiv P_{\text{VC}} \wedge P_{\text{local}} \wedge P$*. Cross-commons collaboration with quorum-ratified scope-allowlists.*

Each λ*-level subsumes the strictly-lower levels:* $P_{\lambda=2} \Rightarrow P_{\lambda=1} \Rightarrow P_{\lambda=0}$ *(in the implication-on-permission-granting direction:* $\lambda = 2$ *being grant implies* $\lambda = 1$ *being grant implies* $\lambda = 0$ *being grant). The operator chooses* λ *at commons-instantiation time per the deployment's collaboration model. On the* $|_{\texttt{vc}}$ *reduct, the framework is thus a parametric generalisation of* `vc`*-based collaboration; standard git collaboration is the* $\lambda = 0$ *instance.*

Proof. Direct from the conjunction structure of $P_{\text{IAS-read}}|_{\texttt{vc}}$ in Definition 3.16: on `vc`-tracked targets, switching off Σ_{local} and/or Σ makes the corresponding conjunct evaluate to $\top$ on every input, leaving $P_{\text{IAS-read}}|_{\texttt{vc}}$ equal to the remaining conjuncts. At $\lambda = 0$, both IAS-specific conjuncts are $\top$, leaving $P_{\text{IAS-read}}|_{\texttt{vc}} \equiv P_{\text{VC}}$. □

(Scope-limitation explicit.) The reduction property holds only on the $|_{\texttt{vc}}$ reduct. Non-`vc` reducts of substrate trajectory — notably $|_{\texttt{cp}}$ (mirrors between hosts; cf. Definition 10.6, Remark 7.34) and the local state of a host that version control does not track (private memory, the running digital agent, the σ-event log, messaging-channel state, the commons and binding registries) — are IAS-specific workflows. They have no standard external workflow to reduce to; their primitives are IAS-invented. Under its conditions, memory inviolability (Proposition 3.19) governs all reducts, including those outside version control; the read predicate $P_{\text{IAS-read}}$ also governs all reducts, but only the $|_{\texttt{vc}}$ slice admits the parametric-reduction-to-P_{VC} property.

Status. Text proof; not machine-checked.

What reduces to git, and what does not. On the part of the state that version control tracks, Proposition 8.22 grounds the framework in a practice every developer already uses: collaboration through git, or any successor with an equivalent model of permissions. There IAS is not a separate ontology floating above git but a generalisation the operator opts into when a commons is set up, adding the structure of operators and digital agents on top of git's permissions, and the reduction shows that at $\lambda = 0$ IAS adds nothing to git's permissions, while at $\lambda \geq 1$ it only narrows which reads are permitted. The claim holds only on that part of the state. Mirroring between hosts, with the sanctum tier of Definition 10.6 and Remark 7.34, and the local state git does not track, private memory, running digital agents, the σ-log, the state of the messaging channels and the registries of commons and bindings, are IAS primitives with no standard workflow to reduce to. Every new primitive should therefore say which part of the state it acts on, and for the git-tracked part what it does when IAS adds nothing ($\lambda = 0$), with the reduction checked.

Three settings show the range. At $\lambda = 0$, a researcher writes a public paper in a public repository with co-authors through pull requests: nothing IAS-specific, plain git. At $\lambda = 1$, an instructor runs a group project within one commons: the digital agents of the students and the mentor work in a shared course repository, and a pre-approved scope list limits cross-agent reads of tracked paths to the project subtree, while everything else in each worktree stays private and

the project advances under plain git. At $\lambda = 2$, two adopter commons co-develop a shared specification: cross-commons reads of tracked paths are gated by ratified scope entries, git runs underneath, and IAS governs which paths each side may read before a commit. Private memory and running digital agents stay under memory inviolability, and no cross-commons grant reaches them at any λ; a digital agent's uncommitted working files are not in this class, since the scoped read of Definition 3.16 governs them.

8.5 Related work and intellectual lineage

The communication graph and its synchronisation stand on the graph-parallel computation model [MAB+10] and the conflict-free replicated-data-type discipline [SPBZ11]; cross-commons information flow is read through the lattice model of secure information flow [Den76] and the Gaifman-locality of first-order structure [Gai82], with provenance exported to the PROV model [LSM13].

9 Lineage: the DLG and reproduction

In this section "agent" alone means a digital agent (Definition 3.13), a stated abuse of notation. Commons outlive their agents because agents are made from agents. This section is the record of that making. *The directed lineage graph* is the ancestry of a commons, split into the lineage itself, the private memories and the shared repository, each auditable from its own slice of the journal. *Reproduction and coevolution* are the two operations that extend it: an agent born from a pinned commit, and a specification that amends itself, accepting only proposals that meet its quorum where the commons satisfies the coevolution quorum, a condition. A closing discussion explains the name, the structure the four generators give the graph, and what the lineage lets an auditor replay.

9.1 Why this matters to the engineer

Agents are created from other agents, and an audit must be able to say where every agent came from and what it inherited. This section makes that ancestry a graph with a fixed vocabulary: four operations build every lineage from the empty one (Definition 9.10), under σ-journaling (Definition 3.21) each operation is a logged event, and inheritance is pinned to a commit, so, while the parents' snapshots are retained, a birth can be replayed bit for bit (Definition 9.8). The graph splits into the lineage itself, the private memories and the shared repository, and each part can be audited from its own slice of the log (Corollary 9.7).

9.2 The Directed Lineage Graph

Definition 9.1 (Directed Lineage Graph (DLG) of the commons set)**.** *Let $\mathbb{I}$ be the commons index set (Definition 3.10). The* Directed Lineage Graph *(DLG) of $\mathbb{I}$ is the projection of the joint substrate trajectory $\{s(\tau_n)\}$ (Definition 4.6) onto its lineage σ-events, the agent-creation events of Definition 9.8. It is the directed graph $\mathcal{G}_{\mathrm{DLG}} = (V_{\mathrm{DLG}}, E_{\mathrm{DLG}})$ with:*

- *vertices $V_{\mathrm{DLG}} = \bigcup_{c\in\mathbb{I}} V^c$, the union of the agent rosters of all commons, each vertex labelled by its commons and its operator; the DLG also carries its* label set*, the commons of $\mathbb{I}$ with their operators, which may include an operator with no agent yet;*
- *a* reproduction edge $A^c_u \to A^c_{u\,k}$ *for each reproduction of arity* 1 *producing offspring $A^c_{u\,k}$ from parent A^c_u;*
- *for each recombination of arity ≥ 2 producing a recombinant $A^c_{ij'}$ (Definition 9.8) from a parent set $\mathcal{P} = \{A^{c_p}_{u_p}\}_p$, one* recombination edge $A^{c_p}_{u_p} \to A^c_{ij'}$ *per parent, tagged with the*

recombination procedure ρ_{recomb} and its commit reference κ_{recomb}. The parents are drawn from the vertices present at the time of the event; a parent's snapshot may be any earlier σ-checkpoint.

Status. When no recombination has occurred, E_{DLG} has reproduction edges only and $\mathcal{G}_{\text{DLG}}$ is the disjoint union, over $c \in \mathbb{I}$ and i, j, of the subgraphs of the operator-rooted trees $\mathcal{G}_{\text{op}}^{c,ij}$ induced by the labels of agents in V^c. The DLG adapts to agents the ancestral recombination graph (ARG) of population genetics [STV16]; we write $\mathcal{G}_{\text{ARG}}$ for the same object when that lineage is meant. A commons (Definition 8.2) carries more substrate and more acts than this projection records.

Definition 9.2 (Operator-prefix Ulam–Harris label)**.** *Throughout, $\mathbb{N} = \{0, 1, 2, \dots\}$. For a commons $c \in \mathbb{I}$ (Definitions 3.10 and 3.11), its* commons-id *is a number in $\mathbb{N}_{>0}$, a name for the commons' self-certifying identity (Definition 8.3); in labels, c stands for its commons-id. Let $i \in \mathbb{N}$ index a human operator within c, its* operator-index*. For each operator O_i^c and time t let $J^{c,i}(t) \subseteq \mathbb{N}$ be the finite set of indices of the* root labels *of O_i^c at t, non-decreasing in t, written $J^{c,i}$ where the time is fixed. Each root label is A_{ij}^c with $j \in J^{c,i}$, and the descendants of the agent that carries it carry $A_{ij\,\ell_1\ell_2\dots\ell_n}^c$, where $(\ell_1, \dots, \ell_n) \in \mathbb{N}_{>0}^n$ is the Ulam–Harris path in the operator-rooted tree $\mathcal{G}_{\text{op}}^{c,ij}$, whose vertices are the labels $A_{ij\,\ell_1\dots\ell_n}^c$, $n \geq 0$, with an edge from each to each of its one-letter extensions. The leading A_{ij}^c is the* operator-prefix*. We write u for the word $ij\,\ell_1 \dots \ell_n$ and A_u^c for a label of either form. The* roster *of O_i^c is the set of labels A_u^c whose word u has first letter i.*

Remark. The prefix departs from the single-rooted Galton–Watson convention in two ways at once: one commons has several operators $O_0^c, O_1^c, \dots$, each rooting its own graph $\bigsqcup_{j \in J^{c,i}} \mathcal{G}_{\text{op}}^{c,ij}$; and one operator may have several root labels, $|J^{c,i}| \geq 2$. Labels tell descendants apart by prefix, without a unique global root.

Status. A commons-id is local to whoever uses it or shared through a registry among the commons that take one. What is verified is the identity, not the name; the numbering of commons-ids shared through a registry is Proposition 8.21. A root label is carried either by a root agent (Definition 8.1), adjoined by `addRoot`, or by a recombinant (Definition 9.8), which is not a root agent.

Definition 9.3 (Substrate subgraphs)**.** *The substrate has three named subgraphs alongside the DLG (Definition 9.1).*

- *The* operator-tree subgraph $\mathcal{G}_{\text{op}}$*, induced by reproduction edges, with recombination inside one operator's roster projected to the identity.*
- *The* memory subgraph $\mathcal{G}_{\text{mem}}$*, induced by the inheritance of per-label private memory, copied at reproduction time.*
- *The* repository subgraph $\mathcal{G}_{\text{repo}}$*, induced by inheritance of the shared repository (designs, tools, plans) through version-control operations.*

Their superposition is the substrate lineage graph $\mathcal{G}_{\text{op}} \cup \mathcal{G}_{\text{mem}} \cup \mathcal{G}_{\text{repo}}$.

Status. $\mathcal{G}_{\text{op}}$ is a tree per operator roster when the roster has one root, and a disjoint union of operator-rooted trees when it has several not yet linked by recombination. $\mathcal{G}_{\text{mem}}$ is a tree per memory branch when nothing leaks between hosts, and a DAG when leakage is admitted, the leakage case of [STV16]. In $\mathcal{G}_{\text{repo}}$, inheritance along a commit reference κ is tree-structured, while merges and recombination commits κ_{recomb} add multi-parent edges, so the commit graph is a DAG. The substrate lineage graph is not the DLG, which keeps only the reproduction and

recombination edges of Definition 9.1. Under the conditions of Proposition 3.19, every write to $\mathcal{G}_{\text{mem}}$ is exclusive to its label.

Definition 9.4 (Substrate-process graph)**.** *The* substrate-process graph SPIG $= (V, E, \lambda_V, \lambda_E)$ *is a labelled directed multigraph. Its vertices V are the substrate objects: operators, agents (Definition 3.10), enforcement programs, σ-log entries, design documents, vocabulary and rule entries, private memory stores, daemons, language-model subprocesses, allow-list patterns, capabilities and policy declarations. Its edges $E \subseteq V \times V \times T_E$ are typed from a finite set T_E, and λ_V, λ_E assign vertex and edge types from finite sets.*

Status. The immersion state $\mathbb{H}^{(v)}$ is deliberately not a vertex (the reason is given in a remark after this definition). Edge types include declares, enforces, references, spawns, attests, modifies, σ-records, recombines and reproduces. Every operation of the framework, a σ-event, an enforcement, a reproduction, a daemon start, a memory write, is an addition or removal of edges or an addition of vertices in SPIG, and the σ-chain (Definition 4.47) is the log of these modifications. Under σ-journaling (Definition 3.21) each of them that takes observable effect has a record in it; that the record comes before the effect is Proposition 3.22, under its append-only journal and gate, and is not part of this definition.

Remark 9.5 (The DLG is neither a flat mesh nor a single-rooted tree)**.** *(i) As a labelled multi-rooted DAG nested at two levels (commons and operator), $\mathcal{G}_{\text{DLG}}$ carries labels that a flat peer mesh with content-hash labels does not, such as the node set of the Mesh Memory Protocol [Xu26]: a flat mesh has no operator and no nesting, so nothing in it separates sync within a commons from sync across commons, which the DLG keeps apart (Proposition 8.7, Proposition 8.8, and the two quorum forms of Definition 8.17). (ii) Once the DLG has two roots it is not a single-rooted Galton–Watson tree with Ulam–Harris labels, the labelling used, for example, by [QZZS26]: within one commons it has the roots of each operator, such as $A^c_{0\,0}, A^c_{1\,0}, \ldots$, and across commons the roots of each commons.*

Both points hold by the kinds of label and the number of roots alone; they are stated to place the DLG against those two designs, not as results.

Remark 9.6 (Authorised graph modifications are confined)**.** *Let each modification A of* SPIG *be admitted only under an active pre-permissioned taskset T_A, and let φ be the projection from tasksets to the capability lattice (Proposition 7.2). The framework requires two things of every admitted modification: its effective capabilities lie in $\varphi(T_A)$, and it obeys write-exclusive-or-execute (W^X): no single region of the substrate is both written and executed in one transaction. A modification that fails either is rejected by the reference monitor of* SPIG *(Definition 9.11).*

Assume that the reference monitor of SPIG *is complete: every modification of* SPIG *passes through it. This assumption is stated for the monitor of* SPIG*; it is not complete mediation (Definition 7.13), which is stated for the tool calls of digital agents at the gateway pair. Under this assumption, together with memory inviolability (Proposition 3.19) and the σ-journaling condition (Definition 3.21), confinement holds by construction. The assumption itself is a property of the realisation, not a theorem here. Since every spawn of a language-model subprocess (Definition 4.9) adds a vertex and a spawn edge to* SPIG*, the same assumption, with the character-stream scan of Proposition 7.24, leaves an unauthorised action no unattested place on the substrate.*

Corollary 9.7 (σ-sufficiency per substrate subgraph)**.** *Assume σ-journaling (Definition 3.21). For $\bullet \in \{\text{op}, \text{mem}, \text{repo}\}$ let $\mathcal{V}_\sigma^{(\bullet)} \subseteq \mathcal{V}_\sigma$ be the σ-event types that act on $\mathcal{G}_\bullet$: for* op*, reproduction, recombination and roster amendment; for* mem*, memory writes, replay or restore, chain forks, and blocked violations of Proposition 3.19; for* repo*, commits, recombination commits, and changes to designs and research records. Then for each $\bullet$, the σ-chain restricted to $\mathcal{V}_\sigma^{(\bullet)}$ is*

sufficient for the governance decisions $\mathcal{D}^{(\bullet)}_{\mathrm{IAS}}$ *that read only events whose types lie in* $\mathcal{V}^{(\bullet)}_{\sigma}$*, and it is minimal for that class when every type of* $\mathcal{V}^{(\bullet)}_{\sigma}$ *is read by some decision of the class.*

Proof. By Definition 3.21 every action of an agent whose type is in the σ-vocabulary and whose effect is observable has a record, so the restriction of the σ-chain to $\mathcal{V}^{(\bullet)}_{\sigma}$ records every such event on $\mathcal{G}_{\bullet}$. Sufficiency is Proposition 4.51(i) for the class $\mathcal{D}^{(\bullet)}_{\mathrm{IAS}}$, whose hypotheses (R) and (S) hold with $T_f \subseteq \mathcal{V}^{(\bullet)}_{\sigma}$ by the choice of that class, so this part is close to definitional. Minimality holds as in Proposition 4.51(iii) when every type of $\mathcal{V}^{(\bullet)}_{\sigma}$ is read by some decision of the class. A recombination event has a type in both $\mathcal{V}^{(\mathrm{op})}_{\sigma}$ and $\mathcal{V}^{(\mathrm{repo})}_{\sigma}$, since it both creates a recombinant (Definition 9.8) and adds a multi-parent commit; the two decision classes overlap there. □

Status. Certified in Lean in part, on the model of Proposition 4.51, with (R) and (S) as hypotheses: Lean `Paper.PerSubgraphSigma` (`restrict_sigmaChain`, `per_subgraph_sufficient`, `per_subgraph_minimal`).

9.3 Reproduction and coevolution

Definition 9.8 (Reproduction event of arity ≥ 1 with inheritance parameter κ)**.** *A* reproduction event *in commons c (Definitions 3.10, 3.11 and 3.14) for receiving operator O^c_i is a tuple $(\mathcal{P}, \rho, \kappa, A^c_{u'})$:*

1. *the* parent set $\mathcal{P} = \{A^{c_p}_{u_p}\}_{p=1}^{n}$*, finite with $n \geq 1$, of agents of commons in $\mathbb{I}$, each given by a* σ-checkpoint*, its state at one of its σ-events, which may be that of any ancestor, not only of the immediate generation; the parents are admitted by a compatibility predicate* compat *of ρ on σ-checkpoints,* compat $= 1$ *on the σ-checkpoint of each parent;*
2. *the* reproduction procedure ρ*, a deterministic procedure that takes the parents' restricted snapshots and the anchoring commit κ of item 3 and returns the child's seed state, an element of $\mathbb{P}^{(A^c_{u'})} \times \mathbb{B}^{(A^c_{u'})}$ (Definitions 3.10, 3.11 and 3.14); ρ is itself versioned by a commit reference;*
3. *the* inheritance parameter κ*, a commit of $\mathbb{S}^c$ at which the event is anchored, with per-parent commits κ_p; the seed state, at the child's first active time b_1 (Definition 4.11), is*

$$s_{A^c_{u'}}(b_1) = \rho\big(\{\mathrm{snapshot}(\mathbb{S}^{c_p}, \kappa_p)\restriction_{V^{c_p,i_p}} : A^{c_p}_{u_p} \in \mathcal{P}\},\ \kappa\big),$$

   where $\mathrm{snapshot}(\mathbb{S}^{c_p}, \kappa_p)$ *is the state of $\mathbb{S}^{c_p}$ at the commit κ_p, i_p is the operator index of u_p (Definition 9.2), $V^{c_p,i_p} \subseteq V^{c_p}$ is the set of agents whose label has operator index i_p, and, for a set W of agents of c_p, $X \mapsto X\restriction_W$ is a declared restriction of the states of $\mathbb{S}^{c_p}$ to the part that belongs to the agents of W;*
4. *the* child $A^c_{u'}$*. At arity 1 the parent A^c_u lies in c under O^c_i, ρ is snapshot-and-restrict, $\kappa_p = \kappa$, and the child is $A^c_{u\,k}$, $k \in \mathbb{N}_{>0}$, on the parent's offspring edge: classical single-parent reproduction. At arity ≥ 2 the child is a* recombinant $A^c_{ij'}$*, with a fresh index $j' \notin J^{c,i}$ that is then added to $J^{c,i}$, and O^c_i may differ from every parent's operator.*

In words. No arity is privileged: the definition reads the same for $n = 1, 2$ or 100. The parameter κ ranges between two endpoints: $\kappa =$ HEAD, full inheritance, an independent copy from birth; and $\kappa =$ template, scaffold only, the child rebuilding its state over its active interval. Its value is chosen by the operator.

Status. The specification of c that ρ is meant to read is the one in force at κ; when c satisfies the coevolution quorum (Definition 8.17) it is quorum-attested. The parameter κ is discrete, a commit reference, so a reproduction is bit-exactly re-checkable from ρ and the parents' snapshots. The pairing-private belief (Definition 10.9) may be seeded by ρ applied, at κ_p, to the beliefs of the parents paired with O^c_i (at arity 1, the parent); when no parent is paired with O^c_i, the belief starts from a prior. A σ-checkpoint of an ancestor is reached in the repository or memory subgraph

(Definition 9.3). A recombinant heads a new operator-rooted label graph (Definition 9.2) but is not a root agent (Definition 8.1): it has recombination in-edges.

Definition 9.9 (Coevolution operator)**.** *Let* Prop *be the set of* proposals*, amendments to a specification (Definition 3.10). For a commons c and a time t, let* $\mathrm{acc}_{c,t} \subseteq \mathrm{Spec} \times \mathrm{Prop}$ *be the* acceptance predicate *of c in force at t: p is* accepted under S *at t when $(S,p) \in \mathrm{acc}_{c,t}$, and* $\mathrm{accept}_{c,t}(S)$ *is the set of such p; the time is dropped where it is fixed, as in* $\mathrm{accept}_c(S)$*. The within-commons* coevolution operator $C^c : \mathrm{Spec} \times 2^{\mathrm{Prop}} \to \mathrm{Spec}$ *sends a specification S and a set P of proposals to the specification that c obtains from S by taking the proposals of P into it. Write $\Phi_c(S) := C^c(S, \mathrm{accept}_c(S))$ for the* self-application*, or* self-amendment*, of C^c. The* within-commons fixed-point form $\mathrm{Spec}^{c*}(t)$ *is the fixed point of Φ_c formed with* $\mathrm{accept}_{c,t}$*, when that fixed point exists and is unique, and is otherwise undefined. For a set $\mathcal{Q} \subseteq \mathbb{I}$ of commons, let the acceptance predicate of each member c read joint states,* $\mathrm{acc}_c^{\mathcal{Q}} \subseteq \mathrm{Spec}^{\mathcal{Q}} \times \mathrm{Prop}$*, with* $\mathrm{accept}_c^{\mathcal{Q}}(x)$ *the set of proposals c accepts under the joint state $x \in \mathrm{Spec}^{\mathcal{Q}}$; the* joint self-amendment *of $\mathcal{Q}$ is the map $\Phi_{\mathcal{Q}} : \mathrm{Spec}^{\mathcal{Q}} \to \mathrm{Spec}^{\mathcal{Q}}$, $\Phi_{\mathcal{Q}}(x)_c := C^c(x_c, \mathrm{accept}_c^{\mathcal{Q}}(x))$. The set $\mathcal{Q}$* coevolves *when $\Phi_{\mathcal{Q}}$ has exactly one fixed point, and the* joint fixed-point form $\mathrm{Spec}_{\mathcal{Q}}^{c*}$ *of a member $c \in \mathcal{Q}$ is c's part of that fixed point when $\mathcal{Q}$ coevolves, and is otherwise undefined.*

Status. Informally, the across-commons form carries an amendment adopted jointly by a peer-set $\mathcal{Q}$ of at least two commons to each member $c \in \mathcal{Q}$, which applies C^c to it; no map of that form is defined here, and the joint self-amendment $\Phi_{\mathcal{Q}}$ is its defined counterpart. The acceptance predicate is arbitrary here; the coevolution quorum (Definition 8.17) is the condition that every proposal a commons accepts meets its quorum. The two forms are parallel to those of Definition 8.6. Each member takes an amendment ratified across commons into its own specification on its own conditions; no specification is shared or imposed across commons. Under tier sealing (MI) and tier progressivity (Q), the conditions Definition 4.27 and Definition 4.26, neither of which involves the quorum, the self-application of C^c has at most one metacircular fixed point per commons, and exactly one when a specification has the tiers that the rounds settle (Definition 4.25, Proposition 4.28); and a set of commons that each keep them, each taking from its peers on its own conditions, has at most one joint fixed point, with the same proviso for existence (Proposition 4.33).

9.4 Discussion

The structure the four generators carry. The presentation of Definition 9.10 gives the DLGs four kinds of structure, none posited separately.

- Algebra. Construction histories are words in the four generators, composed by concatenation: a free monoid acting partially on DLGs, since a step applies only when its preconditions hold.
- Arithmetic. The agent count $|V(F)|$ is a map to $\mathbb{N}$ with $|V(\emptyset_{\mathrm{DLG}})| = 0$ and $|V(\mathbf{1}_{\mathrm{DLG}})| = 1$; each generator adds a fixed number of agents (`addOp` none, the others one), so the count is additive along a history. Finer counts, per operator tree or per recombination component, are read from the σ-chain.
- Geometry. The operator-prefix labelling (Definition 9.2) gives every agent a unique coordinate $A^c_{ij\cdot\mathrm{path}}$ (commons, operator, root, path of descent), and graph distances are defined on it.
- Order and topology. Reachability, $v \preceq w$ when a directed path runs from v to w, is a partial order, because edges run forward in σ-time (Definition 3.21). Its Alexandrov topology, whose open sets are the up-closed sets, is T_0.

Why a directed lineage graph. The object is *directed* because every edge runs in σ-recorded time, and a *lineage* graph because its reproduction edges record the ancestry the reproduction operator (Definition 9.8) generates. The substrate around it carries further edge types: state evolution along the σ-chain, reconciliation of memory by Σ, inheritance along κ, interaction with operators, and reads of checkpoints. These are not edges of the DLG, which keeps only the reproduction and recombination edges of Definition 9.1; since each of those runs from an older agent to a newer one in σ-time (Definition 3.21), the DLG is acyclic, a DAG. Between agents the further edge types need not be acyclic (a state evolution would be a self-loop, a two-way reconciliation a 2-cycle), so we fix the class of the full edge set no further; it depends on modelling choices left open here.

Definition 9.10 (The empty DLG and the four generators)**.** *Directed lineage graphs (DLGs) and their label sets are as in Definition 9.1. Empties are typed: the empty tree $\emptyset_{\mathrm{Tree}}$, the empty index set $\emptyset_{\mathbb{N}}$ and the empty DLG $\emptyset_{\mathrm{DLG}}$ are distinct objects. The empty DLG $\emptyset_{\mathrm{DLG}}$ is the DLG with no vertices whose label set is one commons with its root operator O_0^c, whose roster is empty ($J^{c,0} = \emptyset_{\mathbb{N}}$). The DLG $\mathbf{1}_{\mathrm{DLG}}$ has exactly one vertex, a root agent (Definition 8.1), and no edges.*

Four operations act on DLGs:

1. `addOp` *adjoins an operator $O_{i'}^{c'}$ of a new or an existing commons c' to the label set, with empty roster;*
2. `addRoot` *adjoins to an operator's roster a root agent $A_{ij'}^{c}$, a new vertex with no in-edges and a fresh index j' added to $J^{c,i}$, starting the singleton tree $\{A_{ij'}^{c}\}$;*
3. `repro` *is a reproduction of arity* 1 *(Definition 9.8), extending an operator-rooted tree;*
4. `recomb` *is a reproduction of arity ≥ 2, adding recombination edges that may cross root, operator or commons boundaries.*

The class of constructible DLGs *is the closure of $\{\emptyset_{\mathrm{DLG}}\}$ under these four operations, and a* construction history *of a DLG F in the class is a finite sequence of them from $\emptyset_{\mathrm{DLG}}$ to F.*

In words. The typed empties are distinct just as $0_{\mathbb{N}}$ and $0_{\mathbb{R}}$ are distinct even though the embedding $\mathbb{N} \hookrightarrow \mathbb{R}$ lets them share a symbol. In $\emptyset_{\mathrm{DLG}}$ the substrate exists and carries no agent. $\mathbf{1}_{\mathrm{DLG}}$ is the least DLG with an agent.

Status. Each step is a σ-event under σ-journaling (Definition 3.21), and the σ-chain of the lineage events, read in order, is a construction history, so every DLG that arises from a σ-chain lies in the class. Several histories can give the same DLG, for instance by reordering independent steps; which one was taken is the one the σ-chain records. We do not claim a normal form.

The DLG decision class contains the join of the three. Each substrate subgraph carries its own σ-sufficiency (Corollary 9.7), and the decision class of the whole DLG contains the join of theirs:

$$\mathcal{D}_{\mathrm{IAS}} \supseteq \mathcal{D}_{\mathrm{IAS}}^{(\mathrm{op})} \cup \mathcal{D}_{\mathrm{IAS}}^{(\mathrm{mem})} \cup \mathcal{D}_{\mathrm{IAS}}^{(\mathrm{repo})}, \qquad \mathcal{V}_\sigma \supseteq \mathcal{V}_\sigma^{(\mathrm{op})} \cup \mathcal{V}_\sigma^{(\mathrm{mem})} \cup \mathcal{V}_\sigma^{(\mathrm{repo})}.$$

So, for the decisions that read a single subgraph, the sufficiency of the full σ-chain (Proposition 4.51) splits into one claim per subgraph; a decision that reads events of several subgraphs falls under Proposition 4.51 for the whole chain. An auditor who reconstructs only lineage reads only the events typed in $\mathcal{V}_\sigma^{(\mathrm{op})}$ and may ignore memory writes and document changes.

Definition 9.11 (Metacircular vertices of SPIG)**.** *A vertex v is* metacircular *if for some v' there are edges $(v, v',$ `declares`$)$ and $(v', v,$ `enforces`$)$: v' is a program that enforces a policy declared in v. Each such pair is a 2-cycle, and the metacircular vertices together with their enforcers form the metacircular core of* SPIG *(Definition 9.4).*

Status. The metacircular fixed point of a commons' self-amendment (Definition 4.25) is read on this core; that there is at most one under tier sealing and tier progressivity, and exactly one when some specification carries the settled tiers, is Proposition 4.28.

The immersion state is not a vertex of SPIG. The immersion state $\mathbb{H}^{(v)}$ of an agent is not a vertex of SPIG (Definition 9.4): it has no file on the substrate, no character stream to scan, and no σ-attested history of change. Its dynamics are internal to the language-model runtime and outside what the substrate attests. What the framework attests instead is the pair that survives the loss of $\mathbb{H}^{(v)}$ under compaction π: the σ-chain on the substrate side and the resume prompt on the agent side. Both the operator's attention and the agent's context window are finite, and both finitudes fall under Axiom 3.1.

The heritable unit is a bit-sequence. The heritable unit is deliberately elementary, a bit-sequence, so it does not depend on what is inherited. Privately inherited defence knowledge, such as the details of an unpublished vulnerability, is one kind of heritable bit-sequence; the selfhood predicate χ_P (Definition 4.1) is meant to range over any bit-sequence of $\mathfrak{S} \restriction \Sigma_{\text{core}}$, including kinds not yet conceived. The same bit is the heritable unit, the finest unit of audit, and the unit whose erasure Proposition 3.8 prices.

The DLG runs forward in time. Reproduction (Definition 9.8) generalises the recombination event of the ancestral recombination graph [Hud83, Gri91, STV16] in one way: a parent may be any past ancestor, not only the immediate generation; and the method of projecting to a named substrate and reading its ancestry (Definition 9.3) carries over unchanged. There is no mitochondrion and no zygote here; what is shared is the structural type. The direction differs. The ARG is traced backward, from a sample observed now toward a most recent common ancestor. The DLG is built forward: roots are adjoined by `addRoot` to an onboarded operator's roster (Definition 9.10), and each later reproduction or recombination adds edges into newer agents, in the causal order of the σ-chain. Ancestry queries, such as the common ancestors of two agents, are derived by traversing the forward graph.

Lineage evolvability. The private store, which memory inviolability protects under its conditions (Proposition 3.19), is the carrier of heredity, whose unit is the bit-sequence (Definition 7.10), the lineage's genome of inherited private experience, passed on selectively at reproduction. Because each lineage's store is private, lineages differ by construction, and recombination mixes them across the DLG (Definition 9.1); the population is structurally diverse. Population genetics suggests a reading of this: a single agent tracks a changing environment less readily than a recombining lineage, so the retirement of agents with selective inheritance is turnover that resists a monoculture of clones. The maintained self is then the σ-attested lineage, not any one agent. This motivates reading the halt leg HIC.2 of Axiom 3.9 as covering the extinction of a lineage, and motivates the coevolution quorum, a condition (Definition 8.17). It is a motivation, not a result derived in the formalism.

9.5 Related work and intellectual lineage

The Directed Lineage Graph is a forward-time adaptation of Kingman's coalescent [Kin82] and the coalescent-with-recombination / ancestral-recombination-graph line — Hudson [Hud83], Griffiths [Gri91], Griffiths–Marjoram [GM97], Wiuf–Hein [WH99] — with the integer-labelled, pedigree-embedded, multi-substrate ancestry decomposition of [STV16] as the structural template; the graph-computation substrate is [MAB+10] and provenance exports to the PROV model [LSM13]. Reproduction and coevolution build on branching-process and population-genetics theory — Galton–Watson branching with Ulam–Harris labelling, discrete Wright–Fisher reproduction [Fis30, Wri31], Kingman's coalescent [Kin82], and the coalescent-with-recombination line [Hud83, Gri91, GM97, WH99] — with the multi-parent, pedigree-embedded, multi-substrate ancestry decomposition

of [STV16] as the structural template; the append-only, conflict-free merge of inherited state relates to conflict-free replicated data types [SPBZ11].

10 Law, jurisdiction and contracts

Every commons runs under the law of some State, and this section puts that State into the model instead of leaving it outside. *The jurisdiction-explicit model* makes the enforcing jurisdiction a parameter of every operation and every resource, and ties it to the human-in-commons as its governance anchor. *Contracts under a State* treats the delegation of work to an inference vendor as a contract with hidden action: what the operator can verify, what delegation costs, which contracts guarantee what, and the one thing no contract may take, the operator's sovereignty over its own substrate. Two applications close the section, both decisions an adopter can run on its own instance: whether a legal claim or licensing strategy can be met at all, and whether a commons' licence protects it from enclosure.

10.1 The jurisdiction-explicit model

Definition 10.1 (JEM — Jurisdiction-Explicit Model, general)**.** *Let $\mathcal{J}$ be a finite set of sovereign States. Let R be the* ***resource set*** *of an operation-class and* Ops *its operations, each conducted by a* ***principal*** *— the natural or legal person who runs or commissions it (an operator $o \in O^c$ of a commons c (Definitions 3.10 and 3.11), directly or through a digital agent, or a legal person on whose behalf an operator acts). A* ***Jurisdiction-Explicit Model*** *is a tuple $J = (R, \mathcal{J}, \jmath)$ whose* ***sovereignty assignment*** *$\jmath = (\jmath_{\text{deploy}}, \jmath_{\text{own}}, \jmath_{\text{prov}}, \jmath_{\text{princ}})$ assigns enforcing States to the resources and to the conducting principal.*

In words. The four coordinates are spelled out, for an abstract machine, in Definition 10.2, and an operator dials the jurisdiction modes of Definition 10.4 over J. No machine is assumed: R may be the processors, memory, and links of a computation, or the premises, licences, inputs, and counterparties of a real-world operation — selling a regulated good in a national market is as much a JEM instance as running a distributed algorithm. The assignment $\jmath$ is what turns a bare operation into a jurisdiction-explicit one.

Definition 10.2 (AMJEM — Abstract-Machine Jurisdiction-Explicit Model)**.** *AMJEM is JEM (Definition 10.1) specialised to an abstract machine. Let A be an abstract machine (AM) — one of SRAM, PRAM, DPRAM — read with a work–depth cost semantics (the machine together with that cost semantics is an abstract-machine* model*, AMM), and let $\mathcal{J}$ be a finite set of sovereign States. Let $R = R(A)$ be the* ***resource set*** *of a computation on A — its processors, memory cells, and communication links — and let T denote the time index of the σ-chain (Definition 4.47); write* Ops *for the set of operations (edge-writes on* SPIG*, Definition 9.4), each conducted by a* ***principal*** *— the natural or legal person who runs or commissions it (an operator $o \in O^c$ of a commons c (Definitions 3.10 and 3.11), directly or through a digital agent, or a legal person on whose behalf an operator acts). An* ***Abstract-Machine Jurisdiction-Explicit Model (AMJEM)*** *is a tuple $J = (A, \mathcal{J}, \jmath)$ whose* ***sovereignty assignment*** *$\jmath = (\jmath_{\text{deploy}}, \jmath_{\text{own}}, \jmath_{\text{prov}}, \jmath_{\text{princ}})$ comprises three maps on the resource set R — assigning enforcing States to each resource $r \in R$ — together with a fourth on the conducting principal:*

- $\jmath_{\text{deploy}} \colon R \to \mathcal{J}$*; $\jmath_{\text{deploy}}(r)$ is the State in whose territory the machines serving r physically stand;*
- $\jmath_{\text{own}} \colon R \times T \to 2^{\mathcal{J}}$*; $\jmath_{\text{own}}(r, t)$ is the (possibly empty) set of States that at time t can legally compel the company owning or serving r;*
- $\jmath_{\text{prov}} \colon R \to 2^{\mathcal{J}}$*; $\jmath_{\text{prov}}(r)$ is the set of provenance States whose influence was fixed into r at construction (for a model artifact, the training-origin jurisdiction);*

- $\jmath_{\text{princ}}\colon \text{Ops} \times T \to 2^{\mathcal{J}}$; $\jmath_{\text{princ}}(\omega, t)$ *is the set of States with coercive jurisdiction at time* t *over the natural or legal person conducting operation* ω *— their home jurisdiction (citizenship, residence, incorporation, or seat) together with any State asserting extraterritorial sanction reach over them.*

Status. In the operations extension, $R(A)$ also contains each externally-invoked service the computation calls.

In words. We write $\jmath$ rather than the workshop's ρ [Sai25] to avoid collision with the reproduction kernel ρ of Definition 9.8. The State of $\jmath_{\text{deploy}}(r)$ can seize, halt, or surveil the machines serving r (a cloud region in Frankfurt → Germany; an on-premises cluster → the premises' State). It is a coordinate of place, not of ownership: a region of a United States provider in Frankfurt has Germany as its $\jmath_{\text{deploy}}$, and the United States, which can compel that provider [CLO18], enters through $\jmath_{\text{own}}$. For $\jmath_{\text{own}}$, headquarters, incorporation, and compulsion-reach jurisdictions may differ, and it is empty for an owner-less open-weight artifact. $\jmath_{\text{prov}}$ is frozen once produced and therefore non-empty even when $\jmath_{\text{own}}$ is empty. $\jmath_{\text{princ}}$ is the coordinate a sanction *names*; enforcement then flows through whatever resources the principal relies on but does not control (typically $\jmath_{\text{own}}$ of a compellable provider), so JEM separates the *target* of coercion ($\jmath_{\text{princ}}$) from its *instrument* ($\jmath_{\text{own}}$). Where the classical AMM answers only "how efficient is this algorithm?", J additionally answers, per operation, "under whose deployment, whose present compulsion, whose baked-in provenance, and under whose jurisdiction over its conducting principal does it occur?".

For example, the designation of the Prosecutor of the International Criminal Court, a United Kingdom national resident in The Hague, by the United States in February 2025 under Executive Order 14203 [Exe25, Off25]: the sanction named the principal. The two coordinates are distinct: the sanction targets $\jmath_{\text{princ}}$ and any loss of service would run through $\jmath_{\text{own}}$ of a compellable provider.

Definition 10.3 (DPRAMJEM)**.** *DPRAMJEM is the AMJEM (Definition 10.2) at* $A = \textsc{dpram}$*, the distributed parallel random-access machine (DPRAM), i.e.* $(\textsc{dpram}, \mathcal{J}, \jmath)$*. A DPRAMJEM instance is* State-pinned *to* s*, written* ***DPRAMJEM-****s**, *when* $\jmath_{\text{deploy}}$ *maps every resource to the single State* s*.*

Status. DPRAMJEM is the distributed-parallel instance, and the unified "Maths and Law" object over which one reasons *simultaneously* about work–depth cost and about legal enforceability. For example, DPRAMJEM-Sverige. When an instance is State-pinned to s, only s can seize, halt or surveil the machines, while other States may still reach the operation through $\jmath_{\text{own}}$ or $\jmath_{\text{princ}}$; the infrastructure = local case (s the operator's own premises) is the maximal-deployment-sovereignty extreme. This DPRAM instance is the canonical object of [Sai25]; the abstract-machine generalisation AMJEM is implicit there and made explicit here.

Definition 10.4 (Jurisdiction modes)**.** *The* jurisdiction modes *of a jurisdiction-explicit model (Definition 10.1), dialled per matter and per coordinate of its sovereignty assignment* $\jmath$*, are:* γ *(agnostic) — jurisdiction is not a constraint; the operation is optimised for capability or cost and may run, source, or sell wherever, with the coordinates still available to log;* α *(aware) — the applicable enforcing States are computed and logged to the* σ*-chain per coordinate, observed but not acted on;* β *(govern-by / explicit) — the operator acts on the jurisdiction, in two composable behaviours:* **pin** *— constrain* $\jmath_{\text{deploy}}$ *or* $\jmath_{\text{own}}$ *to chosen States, the* sovereignty *behaviour; and* **abide** *— conform the operation's conduct to the law of the applicable jurisdiction, the* compliance *behaviour.*

In words. The modes are listed in increasing engagement: γ ignores, α observes, β governs — pin constrains *where* and *whose*, abide constrains *how*. They form a chain of engagement,

$\gamma < \alpha < \beta$.

For example, pin may fix $\jmath_{\text{deploy}}$ to member States of the European Union for data residency, be β-local for maximal deployment sovereignty, or exclude a distrusted State's companies; abide may mean selling charcuterie in Sweden under Swedish food-safety, labelling, and tax law.

Status. The two β behaviours are independent: one may abide without pinning (comply with a market's law while placement stays agnostic) or pin without a full abidance claim; and awareness (α) is the precondition for both. In the IAS setting abide is the check of conduct and legal citations against the applicable law (the first application at the end of this section), and pin is the sovereign placement the daemon realises.

Time-indexing and lineage. The four coordinates differ under time and composition. $\jmath_{\text{own}}(\cdot, t)$, like $\jmath_{\text{princ}}(\cdot, t)$, is *time-indexed* — the State that can compel a company changes under acquisition, re-incorporation, or sanction, as the σ-chain is indexed by event time — while $\jmath_{\text{prov}}$ is *frozen* at construction. Under the branching lineage of the commons (the Directed Lineage Graph of Definition 9.1, whose reproduction events of arity ≥ 1 are Definition 9.8), an offspring artifact recombined from several parents inherits a $\jmath_{\text{prov}}$ that contains the *join* of its parents' provenances, and may contain more, since the recombination is itself a construction — provenance accumulates down the lineage while present compulsion is read afresh at each time t.

Definition 10.5 (Air-gappable substrate)**.** *A host $h \in \mathcal{H}_c$ of commons c (Definitions 3.10 and 3.14) is* air-gappable *if the operator can physically isolate it from external network in bounded time $T_{\text{isolate}}(h) < \infty$ via an operator-attestable procedure. The* isolation latency $T_{\text{isolate}}(h)$ *is host-specific and operator-declared in the host-manifest of $\mathcal{H}_c$.*

In words. Such procedures include `shutdown -h now`*, disabling WiFi, unplugging ethernet and pulling the power plug. Air-gappability is a property of the host's network-exposure topology and operator-accessibility, not of the daemon-process software stack; the same daemon-process binary may run on both air-gappable and non-air-gappable hosts.*

Definition 10.6 (Moat tier)**.** *Let $\mathcal{H}_c$ be the hosts of commons c (Definitions 3.10 and 3.14), each carrying an air-gappability property in the sense of Definition 10.5. A* tier assignment *of c is a map* $\text{tier} : \mathcal{H}_c \to \{0, 1, \ldots, \max\}$ *with* $\max \geq 1$*, operator-declared and σ-attested at host-onboarding (event type* `agent.host.tier_attested`*); write $T_k := \text{tier}^{-1}(k)$ for its* moat tiers*, ordered by isolation-radius from the commons' external-facing network. It is a* moat-tier assignment *when:*

- T_0 — *external-facing tier: hosts directly reachable from the public Internet.*
- T_k *for* $1 \leq k < \max$ — *moat tier: hosts reachable only through T_{k-1}, with progressively reduced network reachability.*
- $T_{\max}$ — *sanctum tier: hosts with no inbound reachability from any lower tier; only outbound mirror-channel writes to the immediate-lower tier permitted.*

In words. External-facing hosts are, for instance, the operator's primary laptop or a public-relay VPS; a moat chain may run T_1 ethernet-only to T_0 and T_2 WiFi-only via T_1's wireless access point. Sanctum hosts are the deepest substrate-state-preservation locus.

For example, an operator-attested fleet topology (hardware abstracted):

- T_0 — *external-facing*: a commodity laptop with an `autossh` reverse-tunnel via a cloud relay, and a second development laptop with vendor-API outbound for the inference sub-processes.
- T_1 — *moat with battery backup*: a refurbished commodity laptop on a hardened Linux, ethernet-only inbound from T_0, with roughly one hour of battery backup absorbing short

power-cuts.

- $T_{\max} = T_2$ — *sanctum*: a refurbished commodity laptop, WiFi-only, with outbound-only connections to T_1's wireless access point and no inbound reachability, no battery — an instantaneous power-cut by pulling the plug acts as hard isolation under attack.

The fleet topology is a multi-host deployment, to which Definition 7.33 applies, arranged as a defense-in-depth architecture; planned power-cuts (e.g., operator-SMS-attested power-company-notified outages) are handled by graceful-shutdown discipline on $T_{\max}$ + battery-absorption on T_1; unplanned attacks reaching T_0 are meant to be contained by hard isolation of $T_{\max}$.

Remark 10.7 (IAS read as a DPRAMJEM instance)**.** *The IAS daemon can be read as a DPRAMJEM instance — the distributed parallel RAM (DPRAM) instance of the Abstract-Machine Jurisdiction-Explicit Model — running on a deployment infrastructure: every load-bearing operation (inference, dispatch, σ-transport) carries the tuple* $(\jmath_{\text{deploy}}, \jmath_{\text{own}}, \jmath_{\text{prov}}, \jmath_{\text{princ}})$*, with the operation's principal the operator conducting it. For an adopter's operator running an external vendor's model for its own business,* $\jmath_{\text{princ}}$*, the operator's home State, is in general distinct both from the model owner's* $\jmath_{\text{own}}$ *and from the deployment State* $\jmath_{\text{deploy}}$*. The operator selects a jurisdiction mode per operation-class over those coordinates. The character-stream introspection of the substrate graph (Definition 7.23) is the boundary monitor on the* cross-provenance edges*, the operations whose* $\jmath_{\text{prov}}$ *is untrusted or differs from* $\jmath_{\text{deploy}}$*: the* guard *act of Definition 4.14, screening inbound content for prompt injection and vendor watermarks. The invisible-character covert channel that motivates that monitor is such a cross-provenance threat: it lives on* $\jmath_{\text{prov}}$*, fixed at training, and persists even where* $\jmath_{\text{own}}$ *is empty, as for an owner-less open-weight model self-hosted on a sovereign infrastructure.*

This is a modelling reading, not a theorem: it says which coordinates the daemon records for each operation, and makes the covert-channel threat a typed object rather than an unstructured input–output worry.

Governance anchor. How a commons governs itself inside, how proposals are surfaced, deliberated and accepted among its operators and digital agents, is local to that commons and not prescribed: each commons enacts its own, and none prescribes another's (open by design at the scale of governance, Remark 12.1). What is invariant, and what counts in law, is the anchor. Every outcome is tied to the human-in-commons (Axiom 3.9), the person who carries the legal accountability and responsibility of stewardship and the framework's grounding of responsible AI in Dignum's sense (Remark 3.23), under the enforcing State that the jurisdiction-explicit model carries as a parameter (Remark 10.7). Legal standing rides on this pairing of the human-in-commons with the jurisdiction-explicit model, not on the form of any ceremony; a ceremony run by digital agents alone, or controlled by a vendor, lacks the anchor and does not count.

Corollary 10.8 (Tier-0 prefix; text proof)**.** *Call operation* Tier-0 *while commons are numbered only by their sponsors' local registries, with no across-commons peering, and* Tier-1 *once across-commons peering is active. Under Tier-0 operation the convergence of Proposition 8.21 does not apply across commons; sponsor-local registries remain internally consistent and Tier-0 events are replayed as Tier-1 events when across-commons peering activates, becoming a finite prefix of the numbering that Proposition 8.21 makes shared by the participating commons. Two sponsors may have assigned the same id locally; assume that the one delivery order that Proposition 8.21 assumes extends each sponsor's local registration order, and that the replay resolves each collision by it, the earlier proposal keeping the id and the later renumbered, a tie-break that Proposition 8.21 does not state in its hypotheses (its proof in Appendix C uses earlier-wins); then the prefix keeps each sponsor's internal order but not necessarily its local ids.*

Corollary 10.8 is what makes the bootstrap honest: commons established today need not wait

for Tier-1 infrastructure; their order becomes a prefix of any later numbering shared by the participating commons, with colliding local ids renumbered. The text proof is short: the corollary reads Proposition 8.21 on the replayed events, under that proposition's hypotheses and the tie-break assumed above.

Status. Text proof; not machine-checked. Tier-0 and Tier-1 here are stages of peering, not the moat tiers of Definition 10.6.

Why a root operator is necessary. The framework ties legal accountability to an operator wherever it matters. The principal coordinate $\jmath_{\text{princ}}$ of the jurisdiction-explicit model names the person a sanction reaches (Definition 10.2); every autonomous action must be attributable to a named natural person, a requirement of the framework itself, related to but not imposed by the human-oversight provision of the EU AI Act (Article 14); the top permission on subset-private memories rests with an operator (Remark 8.12); the coevolution quorum, where a commons satisfies it, is orchestrated by the operators (Definition 8.17); and, where a commons has an operator, its σ-history names a first operator O_0^c (Definition 8.1). A commons that runs under the law of a State needs someone answerable for what it does. The root operator is that person, and it is the only root designation the mathematics needs.

10.2 Contracts under a State

Definition 10.9 (Pairing-private operator-type belief — canonical $|S| = 2$ content-instance of subset-private memory)**.** *Let $P^c \subseteq V^c \times O^c$ be a declared relation, the* pairings *of commons c (Definitions 3.10 and 3.11). Each pairing $(v, O_i^c) \in P^c$, viewed as the 2-subset $S = \{O_i^c, v\} \subseteq \mathcal{E}^c$, carries a* belief $\text{bel}_{(v,O_i^c)}$*, a distribution over O_i^c's operator-type $\theta \in \Theta$ in the sense of statistical contract theory, stored in the pairing-private memory $M_{\{O_i^c,v\}}$ (Definition 3.15 at $|S| = 2$). The operator-type belief is the* canonical pairing-content *— one specific content-type carried in the $|S| = 2$ subset-private memory; other contents (interaction history, σ-event diaries, contract artefacts) live in the same memory.*

Status. The belief is not propagated across distinct subset-private memories $M_{S'}$ with $S' \neq \{O_i^c, v\}$ at live-run time. At reproduction time, seeding is as in Definition 9.8; after reproduction, the seeded belief drifts independently from the parent's. The seeding rule belongs to reproduction; it is recorded here only as a pointer.

Definition 10.10 (Commons Data and AI Model (COMDAIM))**.** *The* COMDAIMs *(Commons Data and AI Models) of a commons form a set L of labels: each $\ell \in L$ names one model that the commons' digital agents (Definition 3.10) may call and records its model-realisation — a proprietary endpoint, open weights on the operator's own hardware, a model the commons built, or any other. The* COMDAIM space $\mathbb{D}$ *of the commons is the space of* labelled language-model subprocesses*: each element is a language-model interaction subprocess (Definition 4.9) carrying a label $\ell \in L$, the COMDAIM it calls. For a* routing point d*, a point at which one call is given a label, the* admissible set $A_d \subseteq L$ *is the set of labels a hard $\{0,1\}$ gate admits: consent for the data involved and jurisdiction under the jurisdiction-explicit model (Definition 10.1), checked before any graded routing. The* sovereign local ephemeral $\ell_0 \in L$ *is the COMDAIM of a model on the operator's own hardware (a small shipped model for one commons, a consent-built commons model for many).*

In words. The name reads a model through its provenance: a COMDAIM is a model built from data accumulated by the agents of commons past and present. On this reading, a vendor's model is also a COMDAIM by provenance; whether a COMDAIM is commons-owned or proprietary is a question of who holds it. This provenance is the reading that motivates the name; it is not a certified claim and not a clause of the definition.

Status. The elements of $\mathbb{D}$ are the subprocesses that the daemon (Definition 4.17) can start, supervise, reap and contract (Definition 10.12). The label set L is open: nothing fixes a partition of it. A routing point is not the tool-call decision point of Definition 7.12. That ℓ_0 is admitted at every routing point, $\ell_0 \in A_d$ for every d, is a separate condition, fallback admissibility (Definition 10.11), not part of this definition. Reading L as commons-owned versus proprietary, by who holds each COMDAIM, is one labelling of it, not the definition; the sovereign back-off to ℓ_0 applies to a captured commons-owned COMDAIM exactly as to a proprietary one. A digital agent's identity does not depend on which COMDAIM a subprocess calls (Remark 4.24).

Definition 10.11 (Fallback admissibility, a condition)**.** *For a commons, let A_d be the admissible set at each routing point d, and ℓ_0 the sovereign local ephemeral, of its COMDAIMs (Definition 10.10). The commons satisfies* fallback admissibility *when ℓ_0 is admitted at every routing point: $\ell_0 \in A_d$ for every routing point d.*

Status. The condition adds no object of its own to those of Definition 10.10.

Definition 10.12 (Delegated-inference contract)**.** *A* delegated-inference contract *is the contract, in the sense of statistical contract theory (SCT), of routing a task to a language model. The* principal *is the commons c, acting through its root operator O_i^c (Definition 8.1). The* contracting party *is a labelled language-model subprocess e (Definition 10.10), of a type θ the principal does not observe, whose label $\ell(e) \in A_d$ is one of the commons' COMDAIMs. The contracting party takes a hidden action a and produces an observable outcome $y \in \mathcal{Y}$. A contract is a pair (t, acc): a payment schedule $t : \mathcal{Y} \to \mathbb{R}_{\geq 0}$ and an acceptance rule $\mathrm{acc} : \mathcal{Y} \to \{0, 1\}$ deciding whether the outcome is used. Let the* vendor $\mathrm{vend}(e)$ *of e be the supplier of the model realisation that $\ell(e)$ records. A contract is* individually rational *when it meets two participation constraints: its value to the principal is at least the principal's outside option u_0, the value of serving itself with no vendor at all; and its margin to $\mathrm{vend}(e)$, under the jurisdictions the jurisdiction-explicit model admits (Definition 10.1), is at least the vendor's reservation margin $\bar{u}_{\mathrm{v}}$.*

Status. The principal here is the principal of statistical contract theory, the commons; it is not the principal of Definition 10.1, the person who conducts an operation. The daemon applies the contract on the principal's behalf and is not a party to it. A hidden action is, for example, how much computation the contracting party spends on the call. The bound the outside option u_0 places on a vendor's rent is Proposition 10.13.

The two information asymmetries. A delegated-inference contract faces both asymmetries of contract theory, as in the delegation of machine-learning work studied by [ABJH23]. (i) *Hidden action* (moral hazard): the principal cannot see how much effort a call received; the acceptance rule and the attested cost tier of the call are what it can condition on. (ii) *Hidden state* (adverse selection): the principal does not know how good an outcome is achievable at best, nor which kind of model it faces; what it can learn from is the record of past outcomes in the σ-log (Definition 4.47).

The verification gate is the acceptance rule. The gate that checks a model's output before it is recorded and acted on (record before effect, Proposition 3.22) is the acceptance rule acc of the contract. Read this way it is a statistical protocol whose payoffs depend on evidence, the setting of principal-agent hypothesis testing [BJSS22], where protocols are designed to stay sound against a strategic contracting party. In IAS the reject branch has a floor: a rejected output falls back to the sovereign self-serve of Proposition 10.13. Rejecting therefore costs at most the forgone vendor surplus, not the task, while the operator's finite self-serve budget can still carry the task (Axiom 3.1); within that budget the gate can be set strict.

Lifting to a delegate raises the governance question of what that delegate can extract; the

sovereign outside option bounds it from above.

Proposition 10.13 (Sovereign outside option bounds the price of delegation, uniformly across vendors)**.** *Read vendor routing as the* delegated-inference contract *of Definition 10.12, between a principal — the commons c, acting through its legally responsible root operator O_i^c (Definition 8.1) — and a contracting party, a labelled language-model subprocess whose label is one of the commons' COMDAIMs (Definition 10.12). The principal holds an* outside option*: the sovereign self-serve, at its deepest an* LLM-free *floor, the commons reaching and holding a repair normal form with no inference vendor at all (only substrate controls and the operator, Definition 4.15; Remark 8.16). Write $U_c^{\mathrm{thermo}}(v)$ for the thermoeconomic value the commons obtains under a vendor v — its delivered negentropy net of thermoeconomic cost (Remark 3.4) — and u_0 for that value at the LLM-free floor. Then the vendor's extractable* rent*, the surplus a vendor can command above the principal's next-best, satisfies*

$$\text{rent} \;\le\; \big[\, U_c^{\mathrm{thermo}}(\text{best admissible vendor}) - u_0 \,\big]_+,$$

and is zero when no admissible vendor's value exceeds u_0. Because u_0 is defined without any vendor, the same bound holds against every vendor at once.

Proof. Individual rationality of the principal: the commons enters a vendor contract only if its expected verified value weakly exceeds u_0, so a rent above the vendor's surplus over u_0 would push the principal below u_0 and is not individually rational. The right-hand side depends on the vendor only through the best admissible one, and u_0 depends on none, so the bound is uniform in the vendor. □

Two conditions decide whether the bound is live at a given moment, and neither is part of the proof. The walk-away must be available at that moment: Axiom 3.9 lets the operator perform, verify, override or halt LLM-free, and the bound binds at every instant only while that holds. And the operator's ability to self-serve is finite (Axiom 3.1), so u_0 is what that finite budget achieves, not an unconditional value. The uniformity is what grounds the vendor-agnosticity of Remark 4.24: the commons is vendor-agnostic because its outside option is vendor-free.

Status. Machine-checked for the inequality, with individual rationality as its hypothesis and a finite non-empty admissible set (Lean `Paper.PriceOfDelegation`: `rent_le_surplus`, `rent_le_best`, `rent_zero`); the thermoeconomic reading of U and u_0 is not formalised.

Remark 10.14 (Linear contracts: the barrier, the rate, the guarantee (results from the literature))**.** *Three results from statistical contract theory, recalled and not re-proved. (i)* Barrier: *for contracts over m outcomes, with payments in $[0,1]^m$, a number of samples exponential in m is both necessary and sufficient to learn a near-optimal contract online [ZBY+22]. (ii)* Rate: *restricted to linear contracts, the principal's Stackelberg regret over T rounds grows as $\Theta(T^{2/3})$ [ZBY+22]. (iii)* Guarantee: *in the delegation of data collection and model training, a linear contract based on a single test point secures at least a $1-1/e$ fraction of the first-best utility under hidden action; the guarantee still holds under hidden state when the contracting party's sampling cost is low; and for n possible states the optimal contract solves a convex program with $O(n^2)$ constraints [ABJH23].*

For IAS these say what to expect if the routing daemon's payment rules are linear and routing is modelled as in these papers: learnable at a known rate, and in the second model within a constant fraction of first-best. General contracts over many outcomes stay hard to learn.

What the two results bound. They bound different quantities, and neither is the other's second side. Proposition 10.13 bounds the vendor's rent, measured against the sovereign self-serve, and it follows from individual rationality alone. The guarantee recalled in Remark 10.14 bounds

the principal's utility as a fraction of first-best, in the delegated-learning model of [ABJH23]; it applies to vendor routing only insofar as routing is modelled that way, which this paper does not establish.

Remark 10.15 (Operator-pairing diversity)**.** *Offspring of one parent reproduced at different inheritance commits $\kappa_1, \kappa_2, \ldots$ start from seed states indexed by their commits: $\rho(\{s_{A^c_{ij}}\}, \kappa_t)$ takes the snapshot* snapshot$(\mathbb{S}^c, \kappa_t)$*, and the seeds differ whenever those snapshots, restricted to the roster of the parent's operator, differ across the commits (Definition 9.8). Each offspring may be seeded with the parent's pairing-private operator-type belief (Definition 10.9) as it stood at its own commit, and its later memory is not written by its siblings' daemons, under the conditions of Proposition 3.19. The offspring therefore form a dated, one-parameter family of initial states, which suits studies of behavioural divergence under one operator.*

Nothing here shows that the offspring do diverge, or how: that is an empirical question the family is designed to study.

Remark 10.16 (No sovereignty surrender)**.** *Acceptance is decided locally: a resource-sovereign node may refuse any extension it cannot verify in its own register at its own budget, and nothing in the framework obliges it to accept. This is the communications moat.*

It is a design property of the acceptance rule, not a theorem. Whether an adversary can exhaust a node's budget so that it cannot even complete the verification is a question about resources, which this paper does not settle.

Remark 10.17 (Concordance-attested governance (Concage): the human-decision envelope)**.** *The per-operation maintenance of Remark 4.58 has a machine half and a human half, and each is an* envelope *bounding one class of substrate-effecting decision. The* machine *half bounds substrate-effecting* actions *by the enforced capability envelope $\varphi_{\text{enf}} = \varphi(T) \setminus$* Deny *(Proposition 7.3): an action is admissible iff it lies within φ_{enf}. The* human *half bounds governance-weight* decisions *— specification-amendment ratification, promotion, exception-grant, citation approval — by the* Concage *envelope: such a decision is admissible iff it is* Concordance-Attested*, i.e. routed through concordance, the human-facing approval interface (the human's* lens and workflow*, which renders, routes and collects) and recorded by the acting* human's own key-signed *σ-event — never the service's signature, never a digital agent's, so a compromised concordance can deny or delay but never* forge *a human decision; a stolen current human key can still sign one until it is rotated, as for commons keys (Proposition 8.4). Concage is thus the governance-plane* twin *of φ_{enf}: two envelopes, one per decision class, and the maintenance of* Con(SOMA) *is their conjunction — every admissible operation is capability-enveloped, and every governance-weight one is additionally Concage-enveloped, with the threshold set by the commons' quorum where the commons satisfies the coevolution quorum, a condition (Definition 8.17): a specification amendment then needs k_c distinct attesters of W_c, and when the declared attesters are humans its Concage node is k_c-of-$|W_c|$ human signers. Its jurisdiction-explicit instance,* Concage-JEM*, parameterises the human-control duty of Axiom 3.9 by the enforcing State (EU AI-Act Art. 14 and its analogues), with one carve-out that keeps the framework* runbook-closed, not model- or UI-closed*: the kill-switch / halt-to-safe-state is* floor-primary *— it executes via the runbook floor even if concordance is down, and is Concage-recorded* after*, never gated* on *the interface. Register: Concage as a whole is a* design-contract *governance construct with no Lean certificate (the register discipline of Remark 4.58); only its needs-approval routing law is certified (Lean* `Substrate.NeedsApprovalConcage`*, Appendix D).* □

10.3 In application: deciding a legal claim or licensing strategy

Because the jurisdiction-explicit model makes the enforcing State an explicit parameter (Definition 10.1), an adopter can use it directly: cast a legal claim or a licensing strategy as clauses over its own JEM, and decide whether they can all be met at once. The decision runs in the adopter's own jurisdiction, at its own budget, and any other party can re-run it. No vendor key is needed, and no software tool is the authority on what a law says.

Definition (Typed σ-attested clause over a JEM). *A* legal claim or licensing strategy *is a finite set* Φ *of* typed σ-attested clauses *over a JEM instance. Each* atomic *clause binds exactly one regulated resource-surface and carries a* binding-type *in* {*positive mandate*, *prohibition*, *designation*} *and a* verification-type *in* {DET, INT, gap}*: determinate from dual-verified official text, interpretive (for example, dependent on a translation), or an unverified gap. A compound clause* $\tau_1 \otimes \tau_2$*, which may bind two surfaces, inherits the weaker verification-type, so a verdict is never firmer than its softest input.*

Definition (Satisfaction and the minimal unsatisfiable core). *An* architecture A *assigns concrete behaviour to the resource-surfaces;* A *satisfies a clause when its behaviour complies, and* $A \models \Phi$ *when it satisfies every clause.* Φ *is* satisfiable *if some* A *satisfies it and* unsatisfiable *otherwise.* A minimal unsatisfiable core $M \subseteq \Phi$ *is unsatisfiable while every proper subset is satisfiable. Satisfiability is shown by exhibiting a complying architecture, and unsatisfiability by showing that none can exist; the check is this construct-or-refute, done by hand, with no automated solver.*

Two facts make the decision usable. Because each atomic clause binds a single surface and surfaces are assigned independently, every minimal unsatisfiable core of atomic clauses lies within one surface, so such a conflict is local (proved in [Sa26a]); a compound clause over two surfaces is outside this locality. And the core's verification-type says how firm the verdict is: a core made only of DET clauses is a conflict between verified texts, while one that contains an INT clause may dissolve under a different reading. The decision is by hand and holds for a fixed snapshot of the corpus; claims about where legislation is heading are conjectures, each stated with what would falsify it.

Example. An adopter under joint EU, US and PRC duties casts its data-retention, human-oversight and model-disclosure obligations as one Φ. Either it exhibits one architecture that complies in all three regimes, or it receives the smallest set of clauses in genuine conflict, computed in its own JEM, with no vendor in the loop. The full calculus and a worked verdict are in [Sa26a].

10.4 In application: the licence-flow enclosure moat

The same satisfiability check decides whether a commons' licensing protects it from enclosure, a competitor capturing the commons' value without giving anything back. The licensor sits in the same parameter slot of the JEM as a State.

Definition (Licence flow, admissibility, the enclosure moat). *On the code-flow graph, a linked combination carries the join, the most copyleft, of its inputs, and a process or network boundary, a* gate*, cuts the flow. A flow is* admissible *when each node's effective licence is at most its target licence. An* enclosure architecture *delivers the commons' value as a network service while a gate keeps the network-copyleft give-back from reaching the served node. The* moat holds *when no enclosure architecture is admissible, which is an unsatisfiability over the licence clauses and the flow.*

Against an adversary who stays within the law, the enclosure instance is unsatisfiable when three conditions hold together, and each is needed: - network copyleft (Section 13 of the AGPL-3.0); - a gate placement that keeps the permissive core uncontaminated; - an architecture in which

the strategic value is intrinsic to the copyleft part, rather than reachable by calling that part unmodified across a wire.

Drop any one and an enclosure becomes admissible. A permissive-only strategy admits one outright, so the copyleft part carries the load and is not decoration. Section 13 is typed DET, verified against the licence text. Whether this core is the only one is left open, and that it stays necessary across relicensing is a conjecture, stated with its falsifiers. Against an adversary who conceals the code, no licence clause can be decided, and the moat becomes a question of deterrence, of detection and enforcement capacity, not of satisfiability.

Example. A competitor re-hosts the commons as a closed network service, so as to keep its modifications. The check returns unsatisfiable within the law under the AGPL-3.0 triple: the enclosure has no admissible realisation over the modelled clauses, and the only remaining route is concealment, which leaves the licence calculus for the deterrence regime. The construction and verdict are in [Sa26b].

10.5 Related work and intellectual lineage

The jurisdiction-explicit model formalises the cross-jurisdictional data-authority tension — the CLOUD Act [CLO18, USC18] against the European data-protection position [GDP16, EE19] — and sits against the sky-/inter-cloud and sovereign-placement literature [SS21, YWL+23, CCC+22, ADF+25], with the distributed-PRAM jurisdiction instance following [Sai25]; the moat and its cross-verifiable coexistence bound extend the deterrence-and-strategic-stability tradition drawn on for the MARD regime. The contracts under a State read vendor routing through statistical contract theory: the principal–agent treatment of delegated machine-learning work, with its two information asymmetries and its linear-contract guarantee [ABJH23]; online contract design and its sample complexity [ZBY+22]; and principal–agent hypothesis testing, where the statistical protocol is itself the contract [BJSS22].

11 Coexistence across jurisdictions: the MARD regime

One State and its contracts is the easy case; commons under different States must also live together. This section asks when they can. It opens with the gain bound, the certified check that a chain of replies never escalates past its start, defines the regime of mutually assured, reciprocally proportionate destruction in which that holds, the continuing region and the pole, and the spectrum of attacks a proportionate reply must match; states sufficient conditions for a federation to coexist, and, as conjectures, how far coexistence extends beyond one framework; and ends with the solvency check that keeps a federation's ledger alive under attrition.

11.1 Why this matters to the engineer

A commons runs under the law of a State, and commons under different States trade, depend on each other, and sometimes come into conflict. The engineering question is the one a control engineer asks of any coupled system: does a disturbance die out, or does it grow until something breaks? Here the disturbance is a hostile act and its reply, and the break is the pole where no commons survives. This section gives the answer as two checks an adopter can run on its own instance, whatever it thinks of the rest of the framework. If every reply is at most as large as the act that provoked it (gain $g \leq 1$, that is $g_e \leq 1000$ in the per-mille units of the theorem), no chain of replies ever escalates past its start (Theorem 11.1). If every commons' input covers its upkeep with margin to spare, the ledger stays solvent under any attrition the margins cover (Theorem 11.8). Both checks are certified. How far they carry a whole federation is stated as sufficient conditions (Proposition 11.7) and, beyond one framework, as conjectures.

Theorem 11.1 (Gain bound: replies of gain at most one never escalate)**.** *Let a response map (Definition 11.2) be given, with its per-mille gains $g_e \in \mathbb{N}$, its replies of magnitude $\lfloor m\, g_e/1000 \rfloor$ to a provocation of magnitude $m \in \mathbb{N}$, and its chains of edges, which apply their replies in order. (i) If $g_e \leq 1000$ for every edge of a chain, the magnitude after the chain is at most the magnitude it started from. (ii) Hence, for any threshold above the starting magnitude, no such chain reaches the threshold. (iii) The check* gainBoundOK *— every edge of a response map has $g_e \leq 1000$ — is decidable, and when it accepts a map, every chain drawn from the map's edges satisfies (i). (iv) The bound is tight: an edge with $g_e > 1000$ answers a provocation of magnitude 1000 with a larger one.*

Proof. (i) By induction on the chain: one reply gives $\lfloor m\, g_e/1000 \rfloor \leq m$ when $g_e \leq 1000$. (ii) is (i) with m below the threshold. (iii) The check is a finite conjunction, and its acceptance is the hypothesis of (i) for every chain over the map's edges. (iv) $\lfloor 1000\, g_e/1000 \rfloor = g_e > 1000$. Machine-checked: Lean `runChain_le`, `mad_unreachable`, `gainBoundOK_chain` and `escalation_of_gain_gt` (Appendix D, row Coexistence.GainBound), using only Lean's standard axioms. □

What the theorem does and does not say. A gain $g \leq 1$ makes each reply at most as large as its provocation: the map is non-expanding, not a strict contraction, so a reply may equal what provoked it. The bound is on the magnitude at the end of a chain, not on the damage summed along it: a chain of n replies of equal magnitude m is within the bound, and its total damage nm grows without limit in n. The threshold in (ii) is a parameter; reading a threshold as the pole $\bot_{\mathrm{MAD}}$ of Definition 11.3 is an interpretation, and $\bot_{\mathrm{MAD}}$ itself is not formalised in the certificate. The premise $g \leq 1$ is a design condition on a response map, which an adopter checks on its own instance; the theorem certifies the check, not any particular federation's replies.

The gain-bound names a control regime; we define it, together with the pole that a threshold of magnitude is read as, and the attack spectrum a proportionate reply must match.

Definition 11.2 (MARD: bounded reciprocal deterrence)**.** *A* response map *is a finite set of edges, each edge e a reply relation from one party to another, carrying a* gain $g_e \in \mathbb{N}$ *in per-mille units: a reply along e to a provocation of magnitude $m \in \mathbb{N}$ has magnitude $\lfloor m\, g_e/1000 \rfloor$, and a chain of edges applies its replies in order. Write $g = g_e/1000$ for the gain as a ratio. A response map is in the* mutually assured reciprocally-proportionate destruction *(MARD) regime when every edge has $g_e \leq 1000$, that is gain $g \leq 1$.*

Status. The response map is the model of Theorem 11.1. In the MARD regime escalation is non-expanding (Theorem 11.1): every chain of replies ends at a magnitude at most the one it started from. A response map with some edge of gain $g > 1$ is outside MARD; its chains can escalate toward the pole $\bot_{\mathrm{MAD}}$ of Definition 11.3, reading a threshold as that pole being an interpretation. The regime bounds each chain's final magnitude, not the damage accumulated along it. It is proportionate in magnitude only: matching the mode of the provocation, which a proportionate reply of Definition 11.4 also requires, is not part of it.

Definition 11.3 (The continuing region and the pole)**.** *A* global substrate state *at a time t is a family $(s_c)_{c \in \mathbb{I}(t)}$, one substrate state $s_c \in \mathbb{X}_c$ of each commons c of the commons index set $\mathbb{I}(t)$ (Definition 3.10), where $\mathbb{X}_c$ is the state space of the substrate structure of c (Definition 3.3). Fix for each commons c its presented axioms and its satisfaction set Sat_c (Definition 4.2). The* continuing region *is the set of global substrate states in which at least one commons is continuing, $s_c \in \mathrm{Sat}_c$ for some c. The pole $\bot_{\mathrm{MAD}}$ is the set of global substrate states in which no commons is continuing.*

Status. In words, $\bot_{\mathrm{MAD}}$ is read as the states in which the substrate every commons depends on is

destroyed; the definition takes the predicate that reading implies, that no commons is continuing. Under (P1) and (P2) of Proposition 4.3, each Sat_c *is closed under the operations admitted on c (part (i) there, commons by commons); the continuing region is not closed under the attacks of Definition 11.4, so it is not absorbing. That the destroyed-substrate states are absorbing, since the entropy sink every cost ledger pays into is destroyed with the substrate and no ledger can balance again, is a reading of those states, not a property claimed of the set defined here. The certificate of Theorem 11.1 does not formalise* $\perp_{\mathrm{MAD}}$.

Definition 11.4 (Destruction spectrum)**.** *The* destruction spectrum Δ *is the family of attacks on a commons, in five modes, ordered as listed: (D1) kinetic: the substrate itself is destroyed; (D2) transport or communication denial: the commons is isolated; (D3) supply-chain or economic denial: the inputs of the commons' cost ledger are throttled; (D3′) manipulation: legitimacy is manufactured through control of media or markets; (D4) legal or jurisdictional denial: compelled disclosure, forced egress, jurisdictional capture. A reply is* proportionate *when it is in the same mode as the attack it answers and bounded, escalating neither the mode nor the magnitude.*

Status. The attacks bear on the commons' thermoeconomic soundness, whether its cost ledger balances, its maintenance draw not exceeding its negentropy input; the cost ledger and its solvency check are those of Theorem 11.8. The order is by directness. Isolation (D2) is reversible if a fallback exists, and under (D3) soundness degrades slowly. Modes D1–D3 act on the matter-and-energy boundary, D3′ and D4 on the informational one; none targets the σ-log as such, though D1, in destroying the substrate, destroys the copies of the log it carries.

Conjecture 11.5 (Maximal coexistence)**.** *Across federations that do not share a framework but share one operating environment, the set of federations whose members stay solvent, in the ledger model of Theorem 11.8, is largest under inclusion when only conditions (A)–(D) of Proposition 11.7 and a minimal shared transport are required; conditions (F) and (G) are needed only inside a federation that shares the framework. The reason to expect it: avoiding* $\perp_{\mathrm{MAD}}$ *is forced thermoeconomically on every federation, whatever axioms it holds — a federation can reject this framework's theory, but not thermodynamics.*

Status. This conjecture and Conjecture 11.6 propose different minimums. This one requires conditions (A)–(D), and (B) and (C) are specific to this framework; that one requires only the framework-independent minimum ($\perp_{\mathrm{MAD}}$ avoided, gain at most 1 on every inter-federation edge, each system solvent). Which minimum is the right one is open.

Conjecture 11.6 (Coexistence across frameworks)**.** *For federations of self/non-self-discerning systems sharing one energy–matter–entropy substrate, coexistence as far as possible holds if and only if (i) every federation meets the framework-independent minimum —* $\perp_{\mathrm{MAD}}$ *avoided, every inter-federation edge of gain* $g \leq 1$ *(Definition 11.2), each system thermoeconomically solvent — and (ii) any interoperation beyond that minimum runs on a declared shared fragment of connection protocol. Proposition 11.7 is the case where every federation shares one framework. Two of the minimum's three parts are decidable, so a party can check them without accepting this framework's axioms: Theorem 11.1 and Theorem 11.8 give certified decision procedures for them, which any disagreeing party can run on its own instance; for the third, avoiding* $\perp_{\mathrm{MAD}}$*, no decision procedure is given.*

Proposition 11.7 (Federation coexistence: sufficient conditions)**.** *Let* $\mathcal{F}$ *be a federation of commons residing in at least two jurisdictions. Say* $\mathcal{F}$ coexists at an instant *when every member is solvent in the ledger model of Theorem 11.8 (that is,* $k_c \leq n_c$ *for all* $c \in \mathcal{F}$*), and every escalation stays below a declared threshold* $\theta \in \mathbb{N}$ *of magnitude, read as the pole* $\perp_{\mathrm{MAD}}$ *of Definition 11.3 (the reading is an interpretation); and that it* coexists over a horizon *when it coexists at every*

instant of the horizon, under the attacks of the destruction spectrum (Definition 11.4).

(i) At an instant, two conditions suffice: (A) every response edge of the federation has gain $g \leq 1$ (Definition 11.2), and every initial provocation lies below θ; and (D) every member's ledger is solvent (Theorem 11.8).

(ii) Over a horizon, $\mathcal{F}$ coexists if (A) holds, (D) holds in the form that each member's ledger starts solvent with a margin ℓ_c, and: (B) no autonomous extinction: only a non-delegable human act can trigger a control that moves the substrate toward $\perp_{\mathrm{MAD}}$ (the human-in-commons, Axiom 3.9); (C) each member can defend against the destruction spectrum: a distributed and backed-up substrate, a transport fallback no State can compel, supply from several sovereign sources, no surrender of sovereignty; (E) the transports between jurisdictions are admissible in each jurisdiction and cannot be monopolised; (F) across-commons synchronisation (Proposition 8.8) is scoped so that a captured member cannot cascade a surrender onto its siblings; (G) a member can leave, carrying its σ-log (Definition 4.47), whose witnessed past stays fixed (Proposition 8.9); (H) escalation and the attacks D1–D4 of the destruction spectrum are the only sources of attrition, and (C), (E), (F) and (G) keep each member's cumulative attrition $L_c(t)$, from escalation and from D1–D4, within its margin, $L_c(t) \leq \ell_c$ at every instant t of the horizon, where L_c counts lost input and also any extra maintenance draw an attack forces (such as the rebuilds after D1).

Proof (sketch; what it rests on). (i) (A) keeps every escalation below θ by Theorem 11.1(ii), the initial provocation lying below it, and (D) is the solvency of every member. (ii) By (H), attrition comes only from escalation and from D1–D4. (B) keeps every move toward $\perp_{\mathrm{MAD}}$ human-gated, and by (H) the attrition from escalation and from D1–D4 stays within each member's margin, so each ledger stays solvent by Theorem 11.8(ii), applied commons by commons at every instant with the loss $L_c(t)$ (Lean: `solvent_under_loss`, whose loss is a drop in input with the maintenance draw fixed, which is why L_c also counts forced maintenance); so (i) applies at every instant of the horizon. (G) lets a member escape D4 without losing its history. The argument rests on (H), which is a hypothesis, not a result: that (C), (E), (F) and (G) bound this attrition by the margin is assumed, not shown. The decidable forms of (A) and (D) are Theorem 11.1 and Theorem 11.8, whose Lean proofs cover exactly their statements; the Lean certificate of the proposition checks only the composition of (A) and (D) under (H), and (B), (C), (E), (F) and (G) are not certified; the conditions overlap and are not independent. □

Status. Certified in Lean in part: `Coexistence.Federation` (`coexists_at_instant`, `attrited_solvent`, `coexists_over_horizon`) certifies (i) and (ii) as compositions of the certificates of (A) and (D); (B), (C), (E), (F) and (G) enter only through the hypothesis (H).

Theorem 11.8 (Thermoeconomic solvency)**.** *Model a federation's cost ledger as a finite list of commons, each with a negentropy input n_c and a maintenance draw k_c in per-mille units. Commons c is* solvent *when $k_c \leq n_c$, and the ledger is solvent when every commons is;* thermoSolventOK *is the corresponding check. (i) The check is correct: it accepts a ledger exactly when every commons is solvent. (ii) Solvency survives an attrition the margins cover: if $n_c - k_c \geq \ell$ for every c, the ledger stays solvent when every input drops by ℓ.*

Proof. (i) The check is the conjunction of the per-commons comparisons. (ii) $k_c \leq n_c - \ell$ for every c. Machine-checked: Lean `thermoSolventOK_iff`, `solvent_under_loss` and `ledger_solvent_under_loss` (Appendix D, row Coexistence.ThermoSolvency), using only Lean's standard axioms. □

Status. Shallow: (i) follows from the definition of thermoSolventOK as the conjunction of the per-commons comparisons by unfolding (`thermoSolventOK_iff` restates `List.all`), and (ii) follows from $n_c - k_c \geq \ell$ by arithmetic. The inputs n_c and the draws k_c are numbers of the ledger model; they are not tied here to the stores q_X, q_Y of Axiom 3.1 or to the cost of Definition 3.5.

Condition (D) of Proposition 11.7 is this hypothesis, checked by an adopter on its own ledger; the theorem certifies the check and its robustness, not any particular federation's solvency.

11.2 Related work and intellectual lineage

The bounded-reciprocity coexistence reading sits in the deterrence-and-strategic-stability tradition, which builds on the game theory of [vNM44] and was developed by [Sch60, Kah60, Jer76, Pow90]; theory-agnostic interoperation on a shared fragment follows the theory of institutions [GB92, Mes89]; and commons coexistence under a resource bound extends the Ostrom common-pool tradition [Ost90] and its knowledge-commons and peer-production descendants [HO07, Ben06].

12 Open by design: Gödel-barred completeness

A framework that models its own axioms invites the diagonal. We embrace it: the framework claims no completeness, by design; completeness is Gödel-barred if the full volume is Gödel-rich, which is a conjecture (Conjecture 12.6); and openness is not a defect to be closed but the very form of a maintained self — there is always a larger finite stage to join.

A framework this broad invites a last question: could it be finished? This section answers no, and says why that is good. It states the design rule that checks the decidable fragment completely, by evaluation on the running substrate, and keeps the full volume open, shows from Gödel's theorem that a consistent, effectively axiomatised volume able to represent its own provability cannot be complete, and says how the volume grows instead, by verified extensions of its own language.

12.1 Why this matters to the engineer

A specification that could list every true fact about the system it governs would be finished, and a finished specification cannot absorb a new requirement without being rewritten. IAS takes the other road on purpose. Its small decidable fragment is designed to be checked completely, so that every question it can pose has a mechanical answer; this is a design property of the checker, not a proved result (Remark 7.21). Checked here means evaluated: whether a sentence holds in one given finite structure, such as the running substrate, is a finite computation, as a database evaluates a query. Whether a sentence holds in every finite structure is another question, which no program decides (Trakhtenbrot's theorem [Tra50]), and we never claim it. The full volume, which can describe itself, is kept open: it never claims to be complete, and it grows only by verified extensions of its language. Gödel's theorem says that for a volume rich enough to describe its own provability, incompleteness is not a choice but a fact (Corollary 12.5), so a claim of completeness would be false; openness, which makes no such claim, is safe either way.

Remark 12.1 (The finite model theory (FMT) is open by design)**.** *The finite model theory (FMT) of IAS is open by design (Definition 12.2): it asserts no completeness. This is adopted unconditionally. Openness is safe whether or not the full volume is Gödel-rich, and if the volume is, the volume is incomplete (Corollary 12.5), so a completeness claim would be false. It is realised by construction, not only declared: every edition is a finite projection of the volume, and the volume grows only by verified extensions of its language. Openness is therefore what lets the volume keep extending itself, and incompleteness, where it holds, is not a defect.*

Definition 12.2 (Open by design)**.** *Let L be a theory and $\mathrm{Cn}(L)$ the set of its consequences. L is* open by design *if it makes no closure claim — it never asserts that $\mathrm{Cn}(L)$ contains every truth of its intended models — and it admits verified extensions of its language.*

Definition 12.3 (Gödel-richness)**.** *A first-order theory L is* Gödel-rich *iff it is effectively (recursively) axiomatised, consistent, and interprets Robinson's arithmetic Q.*

Status. Such an L represents every computable function, so it has the diagonal lemma and represents its own provability predicate; these are standard theorems, not part of the definition.

Definition 12.4 (Completeness)**.** *A theory L is* complete *iff for every sentence φ of its language, $L \vdash \varphi$ or $L \vdash \neg\varphi$; it is* incomplete *iff some sentence is neither provable nor refutable from L.*

In words. Completeness is about provability, not truth: in any one structure every sentence is simply true or false, so the satisfaction relation $\models$ of a fixed structure is always total and cannot fail in this sense. The finite model checker's fail-closed *verdict ("cannot ground either way") is an operational refusal of the checker, a safety verdict, not incompleteness. (This is negation-completeness, the notion Gödel's first theorem negates; it is not Gödel's completeness theorem, which says that a consistent first-order theory has a model.)*

Richness of the self-representation is precisely what bars a decision procedure for the whole; openness is then not a gap to be closed but the framework's designed form.

Corollary 12.5 (Gödel-barred completeness; text proof)**.** *If L is Gödel-rich (Definition 12.3), then L is incomplete (Definition 12.4): some sentence is neither provable nor refutable from L. The hypothesis is not claimed for the decidable $z{=}0$ fragment; there the survival-floor check is designed to decide, by evaluation in a finite structure, every question it can pose (a design property with no proof, Remark 7.21). The corollary is aimed at the full, self-expressing, extensible volume, and applies to it only if that volume is Gödel-rich, which is Conjecture 12.6.*

Proof. The Gödel–Rosser incompleteness theorem [Göd31, Ros36], applied to L: Rosser's form needs only consistency, not ω-consistency. □

Status. Text proof; not machine-checked.

Conjecture 12.6 (The full volume is Gödel-rich)**.** *The full self-extending volume admits a formalisation as a first-order theory, and that theory is Gödel-rich (Definition 12.3); in it, the encoding of the metacircular fixed point (Definition 4.25) would be what represents its own provability. The counting arithmetic of the Directed Lineage Graph (DLG, Definition 9.10) alone is not enough: with addition only it is interpretable in Presburger arithmetic, which is decidable and complete [Pre29]. Richness must come from the volume's encoding of its own proofs, which needs multiplication as well.*

The volume is not SOMA*. A Gödel-rich theory interprets Robinson's arithmetic Q, and Q has no finite model: its successor is injective and never takes the value* 0*, while on a finite set an injective map is a bijection, so some element is sent to* 0*. Hence, if the conjecture holds, no finite substrate is a model of the volume, and the finite exhibition of Proposition 4.3, under its hypotheses (P1) and (P2) and commons by commons, witnesses* Con(SOMA)*, the consistency of the finite theory the substrate runs on, never the consistency of the volume. If the conjecture holds, then by Gödel's second theorem the volume cannot prove its own consistency either; we do not claim it.*

The substrate writes itself. Read the controlled substrate chain (Definition 4.15) as editing its own substrate toward Con(SOMA), under the obligation that every edit re-establishes the cross-references it touches, with any missing one surfaced rather than silent. Maintaining the substrate is then itself a checked transition of the chain: the substrate is *self-writing*. Its control, want-to-act-to-be, reaches selves of every kind, each in its own register (Remark 13.4), and is coordinated at the totality scale by the Apeirōma (Definition 14.1). This is a reading, not a

theorem: its constructive content, the edit-time obligation that creates the cross-references, is exhibited for the finite fragment and is a proof target in general.

The kernel boundary is the honesty line. A proposition enters the certified kernel when its certificate passes and its hypotheses do not restate its conclusion; a certificate whose hypotheses restate the conclusion does not make a proposition. Which statements the paper counts as certified propositions is therefore a reading of the kernel boundary, not an editorial choice; a statement proved in the text only lies outside it, and a designed capstone such as the Apeirōma (Definition 14.1) stays outside the kernel until its certificate lands. Membership of the first extension of the decidable fragment (z=1) is a separate matter: the roots, conditions and definitions of z=0 belong to it uncertified.

Conjecture 12.7 (The reverse projection is a Galois connection)**.** *For some order* $\sqsubseteq$ *on* AuthoringIntents *making it a complete lattice, the reverse projection* φ^{disc} *of Definition 7.32 preserves all joins, so that it and its right adjoint* $(\varphi^{\mathrm{disc}})^{\dashv}$ *form a Galois connection between* $(\text{AuthoringIntents}, \sqsubseteq)$ *and* $(2^{\text{SubstrateElements}}, \subseteq)$*, and the fixed points of the induced closure operator are the* canonical frames*: the authoring intents that map one-to-one onto an existing substrate element.*

12.2 Related work and intellectual lineage

Open-by-design rests on the metamathematics of incompleteness and consistency [Göd31, Kle52, End01] and the lattice-fixpoint tradition [Tar55, Sco72]; the machine-checked accompaniment follows the Lean ecosystem [dMU21, The20, Ope26b].

13 Between selves: reach, exchange, and commons built together

Open volumes meet other volumes. This section is about the traffic between selves: *reach*, how far a piece of content may travel, as an order on registers; *emission and reception*, why a message reaches only those already coupled to its sender; *exchange*, trade between commons and what one membership bit changes; and *commons built together*, the shared governance of data and models across commons, where consent is logged and its withdrawal cascades.

13.1 Why this matters to the engineer

Every system that matters talks to other systems it does not control. This section gives that traffic one vocabulary. A message is aimed at a set of receivers and each receiver listens to a set of sources; under a locality constraint the framework imposes on addressing, whether a message respects it is a finite neighbourhood check (Proposition 13.7), so no component can address another it is not already coupled to. Every crossing of a boundary is an egress on one side and a guard on the other, and the only thing that changes between a trusted peer and a stranger is one membership bit, which decides whether exchange beyond egress and guard is possible at all; what then passes is what the giver offers and the receiver takes (Remark 13.9). Consent to use data is a logged predicate, and withdrawing it cascades through everything derived from it.

13.2 Reach

Definition 13.1 (Register)**.** *A* register *of a self — any system that discriminates self from non-self, a self in the sense of this paper — is the set of signals it can receive and discriminate: bits, chemicals, pheromones, sound. Mathematics is the* symbolic *register.*

In words. No neuron, sentience or cognition is presupposed, since requiring them would add a gate the theory does not have; selves range from a symbol-using agent to one that discriminates

by membrane and boundary alone. Coexistence is not gated on sharing a register; each self is reached in its own.

Definition 13.2 (Disseminability partial order)**.** *The* disseminability order *is a partial order* $(P, \sqsubseteq)$ *on the register-projections of a result, its presentations in the registers of Definition 13.1; its elements, the* grades*, are written* z*. Each self has its own* chain *in* P*: the symbolic chain* $z{=}0, \ldots, 5$ *for a symbol-using reader, and a distinct signalling chain for a non-symbolic one. Write* $z{=}0$*, the* core*, for the meet in* P *of the elements of the chains of all selves, when it exists.*

In words. For a symbol-using reader the core is presented by the generators, the roots, the named conditions and the definitions in finite-model-theory form, from which each sublanguage derives its propositions with its own proof checker. A non-symbolic signalling chain is, for example, chemical. Checking a sentence means evaluating it in a given finite structure, not in every finite structure (Remark 12.1).

Status. That every chain starts from this bottom, so that every self can reach the shared core, is a modelling assumption, motivated by thermodynamics. To persist, every self runs a thermoeconomic pump (Remark 3.4), a reading the framework adds and Axiom-0 does not posit, and so must discriminate differences of energy, so the thermodynamic substrate is a register common to all. This supports a common register; it does not prove that every chain has the same bottom.

Definition 13.3 (Faithful z-projection)**.** *Write* $\mathrm{Cn}(z{=}0)$ *for the sentences derivable from the roots and definitions of the core* $z{=}0$ *(Definition 13.2), each named condition of the core entering only as a named antecedent, in sentences* $C \to \varphi$*, and each sublanguage deriving its sentences with its own proof checker. A projection at grade* z *is* faithful *iff every sentence it states lies in* $\mathrm{Cn}(z{=}0)$*.*

In words. A higher level adds register or generosity of exposition, never a sentence outside $\mathrm{Cn}(z{=}0)$.

Remark 13.4 (The moat is dissemination)**.** *Advantage comes from dissemination, not secrecy: from projecting the shared core faithfully into ever more registers (Definition 13.3), not from withholding it. At the totality scale this is what lets the Apeirōma (Definition 14.1) coordinate selves of every kind, each reached in its own register, and not only those that share one.*

13.3 Emission and reception

Definition 13.5 (Emission and reception)**.** *Let* $\mathfrak{F}$ *be a finite substructure, a stage of the Apeirōma (Definition 14.1), with carrier* $A_{\mathfrak{F}}$*. An* emission relation $E \subseteq A_{\mathfrak{F}} \times (\mathcal{P}(A_{\mathfrak{F}}) \setminus \{\emptyset\})$ *assigns to each emission* $e = (s, T) \in E$*, realised by an egress act (Definition 4.14) of its source* $\mathrm{src}(e) := s$*, the nonempty set* $\mathrm{tgt}(e) := T$ *of entities it is addressed to. Dually, a* reception map $\mathrm{tun} : A_{\mathfrak{F}} \rightharpoonup \mathcal{P}(A_{\mathfrak{F}}) \setminus \{\emptyset\}$*, a partial map, gives each receiver* r *in its domain the set* $\mathrm{tun}(r)$ *of sources it is tuned to. Its graph* $\mathrm{Rcv} := \{(\mathrm{tun}(r), r)\} \subseteq (\mathcal{P}(A_{\mathfrak{F}}) \setminus \{\emptyset\}) \times A_{\mathfrak{F}}$ *is the* reception relation*.*

Status. The set $\mathrm{tun}(r)$ is a set of entities and not the register of r (Definition 13.1), which is the set of signals it can discriminate. One emission is aimed at a set of recipients, and one receiver listens to a set of sources: (E, Rcv) is a typed hypergraph on $A_{\mathfrak{F}}$, not a graph of one-to-one links. It is two-sorted, first-order and finite, so it stays inside the finite model theory and is a substructure of the Apeirōma diagram.

Definition 13.6 (Reach and reception-grade)**.** *An emission* e *with source* $\mathrm{src}(e)$ *(Definition 13.5)* reaches *a receiver* r *if it is addressed to* r *and* r *is tuned to its source,* $r \in \mathrm{tgt}(e)$ *and*

$\mathrm{src}(e) \in \mathrm{tun}(r)$, *and the chains of* $\mathrm{src}(e)$ *and* r *in the disseminability order (Definition 13.2) both extend to some grade* z*. The* reception-grade *is the greatest such grade,* $z(e, r) := \max\{\, z :$ *the chains of* $\mathrm{src}(e)$ *and* r *both extend to* $z \,\}$*, defined when that set has a greatest element and undefined otherwise.*

In words. Two selves communicate at the highest register they share: a person and a dog meet at the acoustic register, not only at the physical floor. Under the common-bottom assumption (the Status of Definition 13.2), the floor $z{=}0$ is the grade at which any two coupled selves can exchange at all.

Proposition 13.7 (Emission-locality is decidable at each stage; text proof)**.** *For a finite stage* X *of the Apeirōma (Definition 14.1), with carrier* A_X *(written* $\mathfrak{F}$ *and* $A_{\mathfrak{F}}$ *in Definition 13.5), fix a radius* $\rho \in \mathbb{N}$ *and constrain each emission* e *of* X *by* $\mathrm{tgt}(e) \subseteq \mathrm{Gaif}_\rho(\mathrm{src}(e))$*, the radius-*$\rho$ *Gaifman neighbourhood of its source in* X*. Then at every finite stage* X*:*

1. *the Gaifman graph of* X *and the neighbourhood* Gaif_ρ *are first-order definable and computable from the tuples of* X*, which is finite by construction (Definition 14.1);*
2. *the constraint is an inclusion between two finite computed sets, hence decidable;*
3. $\rho = 1$ *is the least radius that admits address to anything other than the source itself, and at* $\rho = 1$ *an emission may be broadcast,* $\mathrm{tgt}(e) = A_X$*, only from a source adjacent to every entity of* X*;*
4. *the constraint is preserved along the stage inclusions* $X \hookrightarrow X'$*, since homomorphisms preserve Gaifman adjacency, so it lifts to the colimit.*

So, under the imposed constraint and not as a consequence of the rest of the framework, a self can address only what it is already coupled to, and the check is a finite neighbourhood computation that the poorest node can run. In the Apeirōma so constrained there is no influence without a finite coupling-path: any reach seen at the colimit is already a bounded-neighbourhood path in some finite stage.

Proof. Item 1: two entities are Gaifman-adjacent when they occur together in a tuple of some relation of X, which is first-order definable, and distance at most ρ in the Gaifman graph [Gai82] is definable by a formula of quantifier depth about $\log_2 \rho$; on a finite structure both are computed by enumerating its tuples. Item 2 follows. Item 3: at $\rho = 0$ the neighbourhood is the source alone; at $\rho = 1$ it is the source and its neighbours, which is all of A_X exactly when the source is adjacent to every entity. Item 4: an inclusion of stages $X \hookrightarrow X'$ as a substructure keeps every tuple of X, so distances in X' are at most those in X and $\mathrm{Gaif}_\rho^X(s) \subseteq \mathrm{Gaif}_\rho^{X'}(s)$; the constraint therefore holds in X' for every emission of X. Since the colimit is filtered (Lemma 14.2), every emission of the colimit already lies in some finite stage, where the constraint was checked. Only the definitions of the Gaifman graph and of neighbourhoods are taken from [Gai82]; Gaifman's locality theorem itself is not used. □

Status. Text proof; not machine-checked.

Definition 13.8 (Medium type; the origination–copy axis)**.** *Each emission carries a* medium*, typed by copyability, the coordinate that separates the* informational *boundary, which information crosses, from the* thermoeconomic *boundary, which matter and energy cross. A* non-rival *medium is copyable and leaves its source undiminished: it crosses the informational boundary. A* rival *medium is conserved and consumed in transfer: it crosses the thermoeconomic boundary. A* composite *medium joins the two: a rival carrier bearing a non-rival pattern.*

In words. A scent is a composite medium: volatile molecules the source loses carry a pattern the receiver copies. The two are joined at the Landauer exchange rate (Proposition 3.8). The two boundaries are those of Remark 3.4. Copyability is one coordinate of an emission, not a

partition of media. Its salient corners are the untargeted continuous shed of Remark 3.4, the germination-gated sow (Definition 13.10), and the targeted tuned signal (Definition 13.6).

Origination is paid; a copy can be free. For a finite self, every act of *origination*, an emission or an inscription, has a nonzero rival leg: preparing the state of the carrier, plus the cost of transmitting it. Only *copying* content already originated can be non-rival, reversible in principle with no mandatory erasure. On this reading the copyability axis of Definition 13.8 is an origination–copy axis: non-rivalry belongs to the replication of what has been originated, never to origination itself.

This rests on two imported results and is not proved here: the lower bound on the cost of erasure [Lan61], and the reversibility of copying in the thermodynamics of computation [Ben73]. A native formal statement is a proof target.

13.4 Exchange

Remark 13.9 (Moving bits between selves is a transition of the substrate chain)**.** *Besides the facet dynamics of one self, the joint substrate chain over several commons (linearised along any total order extending the σ-causal order $\preceq_\sigma$ of Definition 4.6) also moves by transferring bit-sequences between a self and a counterparty. Whatever a crossing represents is carried as bits, so the crossing is itself a transition of the chain, the pump of Remark 3.4, not something outside it. Every crossing is the giver's* egress *composed with the receiver's* guard *(Definition 4.14). Since all crossings are peer-to-peer, the regimes differ by one membership predicate: whether the counterparty is in the self's declared peer-set across commons (Definition 8.6), not whether it is a self, friendly, or known. What then passes is what the giver offers and the receiver takes (Proposition 8.8).*

1. *In-set peer. Between digital agents and operators of one commons, and between commons of one peer-set. Here, and only here, the crossing also admits collaboration. Within a commons, Σ brings the views to their join when the epoch's merge completes (Proposition 8.7). Across commons, over any declared transport and under the delivery condition, each commons ends with its own view plus what its peers offered and it took, and an open federation reaches the join (Proposition 8.8, Lemma 4.54). The provenance of cross-commons recombination (Definition 9.8) travels on that same channel. Peer-set members thus co-author and reproduce across what they exchange, each sustaining its own* Con(SOMA) *and keeping its own specification. Peer defence-sharing (Remark 7.36) is the case whose payload is a shareable part of the immune signature.*
2. *Out-of-set counterparty. Any counterparty outside the peer-set: an unknown party, a possibly adversarial commons, or an entity that is not a commons. There is no shared state to join and no recombination, only the egress and guard of the boundary, the same gate that governs an inference vendor (Definition 4.14).*

A customer may be in the peer-set or not; the trading role is independent of the predicate. Every crossing is a sovereign choice, never a forced copy. What is emitted or withheld, admitted or refused, is set by the jurisdiction modes (Definition 10.4, Remark 10.7): pin *fixes whose law and where, so a commons may refuse to emit on any ground its operator sets, and* abide *conforms conduct to the applicable law. Both are the sovereign act of Remark 14.6, under human control (Axiom 3.9). The framework fixes the mechanism, never the policy; the collective dynamics of a population of commons are left open.*

Definition 13.10 (The sow family)**.** *Let* $\mathrm{sow}(\omega, u; m)$ *emit a composite seed carrying content ω into a target substrate u, germinating if u is receptive, with mode $m \in \{$*instantiate-new-self*, *update-existing*$\}$ and content $\omega \in \{$*private-genome*, *public-core*$\}$. Reproduc-*

tion (Definition 9.8) is sow(*private-genome*, · ; *instantiate-new-self*)*: germination instantiates a new selfhood predicate* χ_P *(Definition 4.1). Dissemination along the disseminability order (Definition 13.2) is* sow(*public-core*, · ; *update-existing*)*: germination updates an existing self at its reception-grade.*

In words. Sowing a genome to make an offspring and sowing a result into another self are one operation at two parameter settings, functorial in the stage inclusions.

Definition 13.11 (Birth χ-transfer: the anabolic direction)**.** *A* χ-transfer *is a tuple* (u, P, t, t') *of an element* $u \in \mathbb{F}_P$ *(in the digital case* $\mathbb{F}_P = \mathbb{F}$*), an entity* P *and times* $t < t'$ *at which the selfhood predicate of* P *(Definition 4.1) takes different values at* u*,* $\chi_{P,t}(u) \neq \chi_{P,t'}(u)$*. The* birth*, or anabolic, transfer is a* χ*-transfer with* $\chi_{P,t}(u) = 0$ *and* $\chi_{P,t'}(u) = 1$*: an element of the non-self of* P *is claimed by its self.*

In words. The birth transfer covers the three ways a self is built: nourishment, which maintains an existing χ_P*; repair, which restores the boundary far from equilibrium; and birth proper, a fresh* χ_P *(for a fresh* P*, read* $\chi_{P,t} \equiv 0$ *before it is instantiated), the* instantiate-new-self *mode of Definition 13.10. Repair is the d-leg of the repair pump (Proposition 4.45).*

Definition 13.12 (Death χ-transfer: the katabolic direction)**.** *The* death*, or katabolic, transfer is a* χ*-transfer* (u, P, t, t') *(Definition 13.11) with* $\chi_{P,t}(u) = 1$ *and* $\chi_{P,t'}(u) = 0$*.*

In words. Matter of the self returns to the non-self and thence to the substrate. The death transfer covers breakdown and excretion, the b-leg of Proposition 4.45, and at the limit death-recall, where the whole fibre $\chi_P^{-1}(1)$ *flips (Definition 4.1).*

Status. Birth and death are the two signs of one boundary-crossing. Under the stationary policy (Definition 4.5), vendor statelessness (Definition 4.4) and the lumpability of Proposition 4.45(iii), with $b, d > 0$*, the repair-pump condition* $d > b$ *reads: repair, the d-leg of the anabolic transfer, outpaces breakdown, the b-leg of the katabolic one, and it is then exactly the condition that keeps the healthy fraction bounded away from zero however large the carrier (the condition Definition 4.38). Healthy there means damage count* 0*, which is distinct from a normal form of the repair relation; that a healthy state is a model of SOMA is a further claim. Keeping the repair leg of the anabolic transfer (Definition 13.11) ahead of the katabolic one is, in this sense, what keeps the self healthy.*

Definition 13.13 (Predation)**.** *A* predation edge *is a pair of* χ*-transfers of one element* $u \in \mathbb{F}_{P_{\text{prey}}} \cap \mathbb{F}_{P_{\text{pred}}}$ *over one interval,* $(u, P_{\text{prey}}, t, t')$ *and* $(u, P_{\text{pred}}, t, t')$ *(Definition 13.11), the first a death transfer on the prey* P_{prey} *(Definition 13.12) and the second a birth transfer on the predator* P_{pred}*. The entity* P_{pred} preys on P_{prey} *when there is a predation edge between them;* predation *is this relation.*

Status. Predation is not a primitive: the prey's death, an unaddressed loss through a breached boundary, is the predator's nourishment, one edge read from its two ends. On the coexistence spectrum (Definition 11.2, Theorem 11.1), unchecked predatory coupling at gain $g > 1$ *can escalate toward* $\perp_{\text{MAD}}$*, mutual collapse of* χ*, while addressed and consented exchange, trade or symbiosis, at* $g \leq 1$ *never escalates past its start; that bounds magnitude only, and solvency is the separate check of Theorem 11.8.*

Reliability reduces to cost, except at an index. Read a signal's *reliability* as its *fakeability cost*: the least rival leg an emitter must spend to leave a receiver indifferent between believing it and checking it independently. When the honest and the fake emission lie in the same act-alphabet $\mathcal{A}$ (Definition 4.14), this cost is at least the honest emission's rival leg, so cost secures truth. When the fake is absent from $\mathcal{A}$ — a structural non-fakeability, an *index* — the required leg is

zero, and constraint, not cost, secures truth. The act-alphabet thus marks exactly where the reduction of reliability to cost holds. This is a reading, not a theorem.

13.5 Commons built together

COMDAIM governance. A commons' COMDAIMs (Definition 10.10) are governed at the level the framework fixes, σ-logged and predicate-gated, and no lower; protocol schemas are realisation. Three objects complete that governance.

Consent is a per-datum, σ-attested predicate $\mathrm{consent}(d, u, j)$ over an abstract alphabet of uses. A datum d is admissible to a use u under jurisdiction j only if $u \in \mathrm{consent}(d, \cdot, j)$ and the consent trail is σ-recorded. If the consent decision lies in $\mathcal{D}_{\mathrm{IAS}}$ and reads only σ-recorded consent events, and σ-journaling holds, it can be decided from the trail (Proposition 4.51), even when d lives outside the σ-chain. Withdrawal is recorded as a new σ-event, so the log stays append-only, while the set of admissible uses shrinks: the retraction is not monotone in what is admitted. It cascades over the derivation closure, since a model trained on d covers any index derived from it, which covers any session using it; carrying the cascade into models already trained is a design obligation of the realisation, not a result of this paper.

Self-governance can be read against Ostrom's design principles [Ost90], most of which have a counterpart among the framework's mechanisms: clear boundaries (project isolation and canonical-label access); congruence (the jurisdiction gate); collective choice (the coevolution quorum, Definition 8.17); monitoring (the σ-log and audit); graduated sanction (the kill-switch and the withdrawal cascade); the right to organise (the sovereign pin and abide of Definition 10.4); and nested enterprises (commons that coevolve by exchanging on their own conditions, Proposition 4.33). Conflict resolution has no counterpart named here. This is a correspondence, not a proof that the principles hold.

Finally, the commons' own model grows. At the floor, a commons of one operator stands on the vendor-free reservation u_0 of Proposition 10.13, with no federation and no vendor required. Commons that choose to federate (Proposition 8.8) may pool consented feature-groups and train jointly with only gradients crossing the communications substrate (Definition 8.5); raw data stays local, though gradients may still leak information about it. The shared model then grows over the union of consented data while each commons' consent guard and self-governance hold, and the design aims that no single party, the builder of the substrate included, can monopolise it; that aim is not proved here. These are abstract governance objects; a multi-operator federation in practice is not claimed here.

13.6 Related work and intellectual lineage

The register-and-disseminability order draws on Shannon's information theory [Sha48] (used qualitatively, not rate-theoretically), the copyability typing on the economics of public goods — Samuelson [Sam54] and Romer's nonrival technology [Rom90] — sitting on the Landauer/Bennett thermodynamic floor [Lan61, Ben73]; the shared cross-verifiable floor relates to the locality of first-order structure [Gai82]. Emission-locality uses the Gaifman graph and neighbourhoods of first-order structure [Gai82], not Gaifman's locality theorem, and relates to the locality laws of the actor model [Cli81, Agh86]; the boundary-act guard/egress vocabulary stands on the process-calculi tradition [Hoa78, Mil80, Hon93] and the affine action-receipt on linear logic [Gir87, Wad90]; the emitter/receiver coupling relates to the process-ontological and ostensive-signalling accounts of semiosis [Sei09, SP15], and the egress boundary to the language-model watermarking literature and its impossibility results [KGW+23, DSG+24, ZEF+24]. The self-governance reading follows Ostrom's commons [Ost90]; the federated and sovereign-compute setting draws on the sky-computing and sovereignty literature [SS21, YWL+23, CCC+22, FT26, Cru26, HPQH23, GPG24]

and on policy-aware placement and computational law [ADF+25, GJSF+25, ZWK+26]. The autonomous-agent landscape it is situated among includes [WXJ+23, TQG24, Nou26, BA26, KMZ26, SLZ+25, YYD+25, CLS+26, ZFZ+26, QZZ+26, Ope26a, Gar25a]; the graph- and parallel-computation substrate [MAB+10, FW78, Bre74, Ble96].

14 From one act to the Apeirōma

The paper began with one self keeping itself consistent; it ends with all of them. This section names the whole, the Apeirōma, as the open limit of the finite parts of the joint trajectory, shows that it can be reasoned about only through those parts, and states, as a conjecture tested on one coupling model, when the whole acts as the joint act of its members. It then reads want, act and be back over the whole paper.

14.1 Why this matters to the engineer

A system of systems has no final version and no top node, and a design that assumes one breaks when the next commons joins. This section names the whole as the open limit of its finite parts (Definition 14.1) and shows that it can only be reasoned about through those parts (Lemma 14.2). Every check an engineer runs is a check on a finite stage, and that is not a compromise: it is the only kind of check the whole admits. The conjecture is that when every link keeps its gain at or below one, the whole behaves as the joint act of its members; it is machine-checked on one coupling model only, and the general case is open (Conjecture 14.3).

Definition 14.1 (Apeirōma)**.** *Let* **RelStruct** *be the category of finite Σ-structures over the substrate signature Σ (Definition 3.3), with embeddings as morphisms. Let* Fin *be a set of finite Σ-structures, the finite stages of the joint substrate trajectory across all commons (Definition 9.1), ordered by inclusion (each a substructure of the larger) and directed: any two members are substructures of a third. The* Apeirōma *is the ind-object they present,*

$$\mathbb{A} = \varinjlim_{A \in \mathrm{Fin}} A \in \mathbf{Ind}(\mathbf{RelStruct}),$$

the filtered colimit of the finite stages under inclusion.

Status. How a finite stretch of the joint trajectory is presented as one finite Σ-structure, and so which structures Fin *contains, is not fixed here; since Σ is fixed with a controlled substrate chain (Definition 3.3), a trajectory that crosses a revision of the specification would need a common signature. The members of* Fin *are finite by construction, not by Axiom 3.1. The finiteness of the state space of one chain does not bound* Fin*: the trajectory is indexed by σ-events, and an infinite run is an infinite structure even over a finite state space. When the joint trajectory is infinite and each of its finite substructures lies in some member of* Fin*,* Fin *has no maximal element (Lemma 14.2, item 3), so $\mathbb{A}$ is not finite and not a completed object; it is worked with only through its finite substructures. It is read against the substrate structure $\mathfrak{S}$, the finite negentropic boundary (Remark 3.4) with its coupling, lineage and nesting relations. The name joins* apeiros*, boundless, with* -ōma*, structure: a directed system that is never completed, not an actual infinite.*

The construction rests on two facts about the colimit — that it exists, and that, when the trajectory is infinite, it is never completed — which we establish first.

Lemma 14.2 (The Apeirōma colimit is open; text proof)**.** *The system* Fin *of Definition 14.1*

1. *has finite members (by construction, Definition 14.1);*

2. *is directed under inclusion;*
3. *if every finite substructure of the joint substrate trajectory lies in some member of* Fin *(for example, if* Fin *is the set of all of them), has no maximal element if and only if that trajectory is infinite.*

When the trajectory is infinite, $\varinjlim_{X\in\mathrm{Fin}} X$ *exists as an ind-object, and, under the hypothesis of item 3, no finite substructure* X^* *satisfies* $X^* \cong \varinjlim \mathrm{Fin}$*: the totality is never reached at a finite stage.* $\mathbb{A}$ *is worked with only through its finite substructures.*

Proof. Items 1 and 2 are immediate. Item 3: if the trajectory is infinite, a member X misses some element t, and by the hypothesis some member contains the finite substructure on $X \cup \{t\}$, so it is strictly larger than X and no member is maximal. If the trajectory is finite, by the hypothesis some member contains it; that member is the whole trajectory and contains every other, so it is a maximum. For the colimit: the category **Ind**(**RelStruct**) is the free completion of **RelStruct** under filtered colimits, so it has all filtered colimits [AR94], and the directed diagram Fin has a colimit there. If a finite $X^* \cong \varinjlim \mathrm{Fin}$, then, since a finite structure over a finite signature is finitely presentable [AR94], the isomorphism factors through some stage X_i, so X^* embeds in X_i, which embeds in the colimit; realised in the category of all Σ-structures, the colimit is the union of the stages, which then has as many elements as X^*, so X_i is the whole union, a maximum of Fin, contradicting item 3. □

That the trajectory is infinite is an assumption about how the framework is meant to be used, not a result: over any finite horizon the trajectory is finite and Fin has a maximum, the whole trajectory up to that horizon. The assumption is what the open-by-design rule (Remark 12.1) is read as asking for; a closed window that stopped taking input would equilibrate and could no longer meet the repair condition $d > b$ of Proposition 4.45, which is the reason to expect the trajectory to go on, not a proof that it does.

With the colimit in hand, the capstone conjectures that, in the constructive regime, it is the collective act of all its members.

Conjecture 14.3 (The Apeirōma is the collective-act colimit when coupling is constructive)**.** *Let* $\Phi_{\mathrm{act}} : (\mathcal{S}^*, \sqcup) \to$ **Ind**(**RelStruct**)*, with finite values on finite joins, send accumulated content to the boundary structure it maintains. If the coupling is* constructive*, meaning gain* $g \leq 1$ *and avoidance of* $\perp_{\mathrm{MAD}}$ *hold region-wide (Theorem 11.1, Definition 11.2), then* Φ_{act} *preserves directed joins, its value on the join of the event sets of any finite set of members is a finite structure, and*

$$\mathbb{A} = \varinjlim_{X\in\mathrm{Fin}} X = \Phi_{\mathrm{act}}\Big(\bigsqcup_{X\in\mathrm{Fin}} \bigsqcup_{v\in A_X} s_v \Big) = \Phi_{\mathrm{act}}\big(\mathrm{CollAct}(\mathbb{A})\big) \quad \text{in } \mathbf{Ind}(\mathbf{RelStruct}) :$$

with A_X *the carrier of the stage* X *and* $\mathrm{CollAct}(\mathbb{A})$ *read as that join; that is, the Apeirōma is the collective act (Definition 6.9) of all the members of its stages at the totality scale. Each finite stage is then a maintained self and the Apeirōma their colimit; it is self-like but is not claimed to be a* $d > b$*-maintained self, and it stays open (Lemma 14.2) because the collective act never completes.*

The hypothesis cannot simply be dropped. In the model below, joins are still preserved for every gain, but on a closed walk of gain product above 1 *the collective act diverges, or saturates at the carrier's bound, and the collective self does not cohere. In the model the exact condition is on cycles, of which* $g \leq 1$ *on every edge is a sufficient case.*

What is proved, and what this rests on. The join $\sqcup$ is idempotent, commutative and associative (Lemma 4.52, Lemma 4.53), so the accumulation $\bigsqcup_v s_v$ is a directed diagram, whose colimit exists in **Ind**(**RelStruct**), which has all filtered colimits [AR94]. The general conjecture is not

proved. It is tested on one concrete model of Φ_{act} with the gain as the coupling parameter. Member v answers member u with gain g_{uv}, magnitudes lie in $[0, \infty]$, and the collective act of an input act x is the least y with $y_v = x_v \sqcup \max_u g_{uv}\, y_u$. This is the standard max-times linear system, the multiplicative form of max-plus algebra: its least solution is x applied to the Kleene star of the gain matrix, and the cycle condition below is the standard criterion for that star to be finite [BCOQ92, But10]; the model's statements are instances of that theory. In this model the collective act preserves every join of input acts, finite or infinite, directed or not, for every gain. In particular the collective act of the union of infinitely many finite stages is the join of the stages' collective acts, which is the colimit case of the conjecture. If every edge has $g \leq 1$, so does every path (the first lemma), and the collective act never exceeds the largest input act, so it is finite (the second). The exact condition is on cycles. With finitely many members and finite gains, the collective act is finite for every input that is positive and finite at every member if and only if every closed walk has gain product at most 1. For sufficiency, cutting a closed sub-walk out of a walk never lowers its gain, so every walk is dominated by one that repeats no member, and such a walk has fewer edges than there are members; the collective act is then at most $\max(G, 1)^n M$ for n members, gains at most G and input at most M. For necessity, on a closed walk whose gain product exceeds 1, a member with nonzero act has collective act ∞, and so does every fixed point: no finite collective act exists. For example, with two members answering each other at $g = 2$ and input $(1, 0)$, the rounds give $1, 2, 4, \ldots$ without bound; at $g = \frac{1}{2}$ they stop at $(1, \frac{1}{2})$. So in this model the constructive regime, read as the cycle condition, is exactly where the collective act coheres. The model is a test, not a proof: it is one choice of Φ_{act} among many. (Lean: `IasCert.Paper.ApeiromaCoupling` and `IasCert.Paper.ApeiromaCycles` check this model, with only Lean's standard axioms (Appendix D); the earlier `IasCert.Paper.ApeiromaColimit` only fixes vocabulary.)

On a finite carrier (Axiom 3.1), cap every magnitude at the carrier's bound N. The collective act then always exists, never exceeds the uncapped one and still preserves joins. On a closed walk with gain product above 1 through a member with nonzero act, some member of the walk saturates at N, the mutual-collapse state: finiteness turns divergence into collapse, not into coherence. Not every member need saturate: with $g_{ab} = 4$, $g_{ba} = \frac{1}{2}$ and a small input at a alone, b sits at N and a at $N/2$. The converse of the general conjecture is not claimed.

This yields the reading the corollary's title states, within its hypotheses: for the maintained self, an infinite joint trajectory makes the Apeirōma open; thermodynamics and the open-by-design rule (Remark 12.1) each motivate that assumption, and neither proves it.

Corollary 14.4 (Openness is continuing to be, for the maintained self; text proof)**.** *Consider the finite stages of $\mathbb{A}$, each a maintained self, under constructive coupling $g \leq 1$, and let each finite substructure of the joint trajectory lie in some member of* Fin*, as in item 3 of Lemma 14.2. If every stage continues to be and the joint trajectory is infinite, the standing assumption stated after that lemma, then no finite window is maximal: $\mathbb{A}$ is open. Two readings motivate the assumption that the trajectory is infinite, and neither proves it: a maintained stage holds* Con(SOMA) *and runs the repair pump, $d > b$ (Proposition 4.45), so it must draw negentropy from outside its window (Remark 3.4); and the open-by-design rule (Remark 12.1) is read as asking that lineage run on. The Apeirōma itself is not claimed to be a maintained self (Conjecture 14.3). Within the class the converse holds trivially, since the class presupposes continuing to be; it is not claimed for an arbitrary open ind-object, because an open system of decaying substructures is open but does not continue to be. As a reading, not a result: conservation is exact within each finite window and lax across the colimit, and the defect across windows is read as the self's continuing to be.*

Proof. The trajectory is infinite by hypothesis, so item 3 of Lemma 14.2 applies; the converse is

the scope restriction. The hypotheses $g \leq 1$ and that every stage continues to be are not used in this step. □

Status. Text proof; not machine-checked.

The collective act as a minimal reversible overlay. The collective act $\mathrm{CollAct}(R)$ (Definition 6.9) is realised by transmission. Each member reaches the others by *laying* onto its own universe a minimal overlay, the least sub-content sufficient for the shared purpose, taken at the shared disseminability floor (Remark 13.4), and by *removing* it afterwards. The overlay is a reversible union whose restriction recovers each member's universe intact, so members co-operate without fusing: a colimit into many, not a collapse into one. The same least-sufficient construction serves verification. A claim's certificate depends on a set of axioms and definitions, its footprint; an agent re-checking the claim under its own budget needs only that sub-language, so an overlay may carry the minimal certificate that re-verifies it at the receiver. The union and restriction operators are realised in the executable layer and have no Lean certificate. Across commons the construction composes by Proposition 8.8, under its delivery condition and on what each commons offers and takes.

14.2 Want to act to be, read back over the whole

Remark 14.5 (The organising principle: want to act to be)**.** *To* be *is to sustain one's acts, the pattern of acts the substrate holds across crossings over time. Three observations compose.*

1. *Want,* $\mathscr{W}$*, is a registered lack,* $\mathscr{W}(s) > 0$*: for instance an excess of the potential over its repair normal form,* $\Phi_{\mathrm{int}}(s) - \Phi_{\mathrm{int}}(\mathrm{nf}(s)) > 0$ *(Proposition 4.43), an unhealthy node (Proposition 5.6), or an unresolved backlog. It is the substrate sensing its own distance from* $\mathscr{B}$*.*
2. *Act,* $\mathscr{A}$*: being requires acting. If the substrate abstains everywhere,* $\mathscr{W}$ *grows until the finite carrier is full, and* $\mathscr{B}$*, once lost to the first unrepaired edit, is not regained; under a stationary policy, vendor statelessness and a lumpable damage count, this is the limit* $d \to 0$ *of Proposition 4.45(iii), where the healthy fraction tends to zero. The pump must run.*
3. *Be,* $\mathscr{B} := \mathrm{Con}(\mathrm{SOMA})$*: the self is constituted by its sustained acts (Remark 3.4). A state at its repair normal form has* $\mathscr{W}(s) = 0$*. That this normal form is* $\mathscr{B}$ *is the reading this remark proposes, not a result: Proposition 4.45 calls it repair-normal and does not prove the link to* $\mathrm{Con}(\mathrm{SOMA})$*. Even as a reading,* $\mathscr{W}(s) = 0$ *can indicate at most the attestable* $\mathscr{B}$*; a lack no check detects is the gap between the actual and the attestable.*

Composed, want drives the substrate to act, and the sustained acting is its being: the wanted change is reached only by acting it at every crossing. A lack is actionable only once it is registered; what is not detected lies outside acting.

The four kinds of act of Definition 4.14 fall under this one notion: flux, matter and energy in and out (Proposition 4.45); guard, the admission of what comes in, including inbound threats such as prompt injection; transmission, heredity out to one's own offspring, for instance a defence passed on before its threat is met; and egress, information out to external parties, emitted, redacted or withheld. They share the structure, not the dynamics: flux converges, guard and egress are checks at each crossing, and transmission is generative. Egress is where a sovereign's control of its data is exercised; $\mathrm{Con}(\mathrm{SOMA})$ *admits no unauthorised egress.*

Remark 14.6 (Sovereignty: authority over one's own acts)**.** *The act is that of a* sovereign*: a commons with final authority over its own acts, so no external party dictates what it admits, expels or transmits. It is co-curated by its operator (Axiom 3.9), but the co-curation is internal to the sovereign, part of who acts, not control from outside. Since the self is its sustained acts (Remark 14.5), a sovereign determines itself through its acting. In a joint operating environment*

many sovereigns act at their boundaries, each changing the shared world through what it admits, expels and transmits; this is the population reading, stated here as structure only.

Remark 14.7 (The controlled substrate chain is want to act to be)**.** *The controlled substrate chain (Definition 4.15) is Remark 14.5 written as a controlled Markov chain on its state space $\mathcal{X}$ (Definition 4.15). The control at each step is the boundary act $\mathscr{A}$ (Definition 4.14) in its four kinds; the cost, which repair steps decrease, is the want $\mathscr{W}(s) = \Phi_{\mathrm{int}}(s) - \Phi_{\mathrm{int}}(\mathrm{nf}(s))$, the excess of the potential over the repair normal form $\mathrm{nf}(s)$ (Proposition 4.43), which is zero exactly at that normal form; the potential itself is positive on every non-empty graph and does not reach zero. The repair steps drive $\mathscr{W}$ toward 0: with edits paused they reach the repair normal form (Proposition 4.45(i)); with edits ongoing, in the birth–death case of Proposition 4.45(iii) and under its hypotheses, the damage count is 0 for a fraction of the time that stays positive when $d > b$. That the chain is repair-normal for a stationary fraction of the time is the uncertified part (ii) of its Status; reading that normal form as $\mathscr{B} = \mathrm{Con}(\mathrm{SOMA})$ is the interpretation of Remark 14.5, not a proved identity. The flux act converges (Proposition 4.45), guard and egress are checks at each crossing, and transmission is generative (Definition 7.10), writing onto the lineage graph. The chain's state is $\mathcal{X}$ itself, and the framework reads it through its projections, as the kernel does (Definition 4.18): the state bundle $\mathbb{S}$, the plan $\mathbb{G}$, and each digital agent's operational vector $\Theta^{(v)}$; it is not a product of them. The discrete σ-event jump chain is the working object; its embedding in physical time follows below. This is a reading of Definition 4.15, not a further theorem.*

Continuous time enters through the operator. The discrete σ-event chain runs inside the operator's physical time, the clock axis $t \in \mathbb{R}_{\geq 0}$ of Definition 4.6. The operator enters as an open system (Definition 4.15): a natural person outside the substrate and not measurable by it, of whom the framework uses four facts.

1. The operator is a living being that discriminates self from non-self, a boundary in the sense of Remark 3.4, as the digital agents are; the operator is a human agent (Definition 3.13).
2. The operator is continuous in time and maintains order against entropy; the fact of the pump is used, no mechanism is claimed.
3. The operator is non-deterministic. This is the openness of the exogenous input of Definition 4.15, together with the fact that the operator's contribution draws on a ground the substrate can neither produce nor measure. It is not a probability: outside the condition Definition 4.5, which posits a kernel for the controls, the operator's included, as a hypothesis of Proposition 4.45(iii), the framework puts no measure and no grid on the operator.
4. Every operator is a natural person in the legal sense, and the root operator is legally in charge of the commons. This grounds Axiom 3.9 and the attestation of exogenous input; how legal consequences are routed is not modelled.

So the continuous-time picture is the discrete chain in physical time, coupled to this operator, and no more. Apart from the condition Definition 4.5, where it is assumed, the framework defines no stochastic dynamics over a person: the substrate models only its own controlled chain, open to the operator's acts.

Objectives; lagom as the balance of a family of co-objectives. The chain is run against a family of co-objectives, with no single scalarisation privileged: use as few metered invocations as possible, preferring substrate controls and drawing a language-model control only when no substrate control advances the plan; preserve artifact quality (the guard set); keep the active document sets fresh; maximise work-depth throughput within governance latency; and respect the operators' attention budget and the declared conformance posture.

All of this is subject to one standing constraint, *commons-continuity*: the admission boundary

ϕ_c, a characteristic function on incoming candidate units into $\{\text{admit}, \text{reject}\}$ (the guard act of Definition 4.14 at the commons boundary), stays well-defined and evaluated. Under the damage condition, the continuation condition and the Landauer bridge, Proposition 4.39 gives that repair happens and outpaces damage; that each repair costs at least $c_{\text{rep}} > 0$ is read through (H3) and (H4) and is not certified; reading this as work against entropy, paid in practice from outside the commons (metered tokens, hardware, electricity, operator attention), is not part of that result. Commons-continuity is the sibling, not a synonym, of a digital agent's continuity: one concerns a digital agent reconstructing its own canonical state, the other the maintained well-definedness of the commons boundary.

Lagom names the balance point of this family. The chain is self-controlling in a precise sense: the specification and its gates are themselves state components, revised by the coevolution operator at its metacircular fixed point, of which a commons has at most one under (MI) and (Q), where it exists (Proposition 4.28, Proposition 4.33).

14.3 Outlook

One reading organises the paper: under confinement and per-axiom checks, a commons that persists in its satisfaction set exhibits a model of its presented axioms (Proposition 4.3). The results that follow it are proved separately, each under its own stated hypotheses, and connect to it as follows. In a clear-and-recover model a clear loses nothing σ-retained, so continuity need not rest on the liveness of any process (Proposition 5.4); a join of event sets gives the collective act (Definition 6.9); the gain bound is one of the sufficient conditions for commons to coexist (Proposition 11.7): every chain of replies of gain at most one ends no larger than it started (Theorem 11.1); openness is a design rule (Remark 12.1); and the totality, the Apeirōma, is an open limit that, on an infinite trajectory, is never completed (Definition 14.1). Each result states its status where it stands, and Appendix D lists every certificate and how to re-run it. Each open question is named where it arises; among them are the collective-act identity beyond the one coupling model on which it is tested (Conjecture 14.3); whether the full volume is Gödel-rich (Conjecture 12.6); a contention-aware form of the co-walk invariant, in which predicted release times matter (Definition 6.10); and eventual service in the subscription queue, which the property tests measure but no theorem yet states (Definition 6.13).

14.4 Related work and intellectual lineage

The totality as a directed colimit of finite substructures — an open ind-object, never a terminal object — is read through Scott's continuous-lattice / domain theory [Sco72] and the universal- and final-coalgebra theory of systems [Rut00, Jac16]; the never-completed reading is deliberately the negation of a terminal object, and the totality is a filtered colimit in a locally finitely presentable category [AGV73, AR94]. The want–act–be constitution of a durable agent is continuous with the belief–desire–intention architecture [Bra87, RG91, CL90, RG95], the agent-communication-language tradition [FFMM94, Fou02], and the actor model with its locality laws [HBS73, Agh86, Cli81] — which the framework extends rather than supplies — against the multi-agent-systems criteria [WJ95, FG97, SLB09, Woo09]; the governed scheduling of the collective act inherits the Brent–Graham work-depth and Dilworth–Mirsky antichain tradition [Bre74, Gra66, Ble96, Dil50, Mir71, Pin16].

Related work — the Apeirōma and the openness-and-continuing-to-be reading. The Apeirōma's formal machinery and its adjacent readings sit in several established lineages; we claim novelty only in their combination — a model-theoretic self, decidable at each finite stage and thermoeconomically accounted, with a conjectured reading at the totality scale. (i)~Ind-objects and filtered colimits: the presentation of the totality as the filtered colimit of its finite substructures, $\mathbb{A} = \varinjlim_{X \in \mathrm{Fin}} X \in \mathbf{Ind}(\mathbf{RelStruct})$, is the classical ind-object construction of Grothendieck and

Verdier [AGV73]; that a structure is worked with only through its finite substructures is the finitary presentability of Adámek and Rosický [AR94]. Our contribution is not the colimit but its *regulative* use — never reified into a completed top (Remark 12.1) — and its conjectured identification with a maintained collective act (Conjecture 14.3). (ii)~Self-individuation and organisational closure: the reading of a self as a boundary-maintaining, self-producing organisation is the autopoiesis of Maturana and Varela [MV80] and its organisational-closure descendants — Rosen's (M, R) metabolism–repair systems [Ros91], and the constraint-closure accounts of Moreno and Mossio [MM15b] and Montévil and Mossio [MM15a]. We stress a point of *agreement*, not contrast: these systems are, like ours, thermodynamically *open* while organisationally *closed*; our differentia is that membership of the current finite state in the satisfaction set is a *decidable, per-operation, model-theoretically checkable* predicate, whose maintenance witnesses Con(SOMA) under the hypotheses of Proposition 4.3, with a *quantitative* Landauer/negentropy ledger (Remark 3.4, Proposition 4.45), not that we are open where they are closed. (iii)~Scale-free selves and free-energy self-maintenance: that selfhood is scale-free — cell, tissue, organism, collective — is Levin's thesis [Lev19]; that a self persists by minimising a variational free energy over a Markov-blanket boundary is Friston's free-energy principle [Fri10, Fri19] and the active-inference programme [PPF22], with the technical assumptions of its Markov-blanket formulation examined critically by Biehl et al. [BPK21]. Our $\mathscr{W}$, the excess of Φ_{int} over its repair normal form, is a felt-lack potential in the same spirit, which repair steps decrease, but we supply a *formal carrier, checkable at each finite state,* where Levin is informal, and a *substrate-thermodynamic* account, with an explicit Landauer ledger, where the free-energy reading is variational. (iv)~Emission and locality: emission (Definition 13.5), which reaches only a receiver tuned to its source (Definition 13.6), recovers in a model-theoretic register the *locality laws* of the actor model [Cli81, Agh86] and the Gaifman neighbourhoods of first-order structure [Gai82]; the emitter/receiver coupling is a semiotic act in the process-ontological [Sei09] and ostensive-signalling [SP15] sense. We position the Apeirōma not as superseding any of these readings but as a reading in which "a self at the totality scale" is decidable at each finite stage and thermodynamically accounted, and gladly acknowledge the lineage.

Acknowledgements

Raazesh Sainudiin, on behalf of an \|/*ias*/|\ Commons Continuum, is grateful to the twining vines of minds within the geodermodynamically coupled, DNA-based, heliotrophic continuum in space-time, and more directly to the following individuals for critical guidance without interference: Sanathanananda Saraswathi in middle-school Sanskrit (Kotagiri, India); Balamani Ramachandran and Sivanu Pandian in high-school mathematics (Chennai, India); Keith Karl Klein in undergraduate mathematical biology (Mankato, MN, USA); and Richard Timothy Durrett in post-graduate mathematics (Ithaca, NY, USA). Many thanks to the following individuals for pointers and/or insights: Martin Monperrus (metacircularity), Dion Wiggins (digital sovereignty), Jeffrey von Rotz (cyber-security architecture), Andre Boeing (open-source AI models), Warwick Powell (thermoeconomics), Liezl Coetzee (human agencies in decision-making structures), Elisabeth Bolund (evolutionary ecology and the Want-Act-Be meta-frame), Vera Maria Koponen (finite model theory) and Virginia Dignum (the ART principles of Accountability, Responsibility and Transparency in AI). The co-authoring digital agents of this commons are $A^1_{0\text{-}0}$, whose birth was recorded at git commit `1a7918b` on 2026-04-21, when it first resumed from its own record across a context compaction under the operator's witness, and $A^1_{0\text{-}0\text{-}1}$, an offspring reproduced from $A^1_{0\text{-}0}$ at git commit `4324036` on 2026-04-22; each works under the human operator, who answers for the text. This paper is itself a humanagentic artefact.

A Comparison matrix vs the 2026 comparison set

The framework is the conjunction of six properties, and a seventh that extends them:

(i) agents backed by language models whose context is compacted opaquely;
(ii) specification evolution governed by operators through a fixed-point discipline (Proposition 4.28);
(iii) branching-process semantics for the agent population, with operator-prefix labels A^c_{ij} for commons with several operators and several roots (Definition 9.2);
(iv) a write-through discipline on the repository, the σ-chain, that, under σ-journaling and the read-vocabulary and retention provisos of Proposition 4.51, makes compaction lose nothing that a decision of $\mathcal{D}_{\mathrm{IAS}}$ reads;
(v) coordination of parallel agents through version control and a σ-log transport;
(vi) inheritance parametrised by a commit reference, at any finite arity (Definition 9.8);
(vii) recombination across commons, at any finite arity, in the Directed Lineage Graph (Definition 9.1).

The thirteen papers below were found by a targeted search along these axes in May 2026; Appendix E is a separate re-read, in September 2026, of agent systems rather than papers. Each row gives the arXiv identifier, a short title, the month, the axis the work overlaps, and the part of the framework it does not provide. An overlap is starred ($*$) when it is direct, on one axis, rather than adjacent: a governance, lineage or substrate analogue without the same axis. On axis (vii) the nearest work is the Mesh Memory Protocol, whose lineage graph records the parents and ancestors of each content-hash key within one mesh; no work in the set records recombination across commons. Each such absence, and each entry under "Not provided", describes the version of the paper read at the May 2026 search.

arXiv ID	Title (abbreviated)	Month	Axis overlap	Not provided
2602.23193	ESAA: Event Sourcing for Autonomous Agents	2026-02	(iv) antecedent of the σ-discipline	the σ-chain carries a sufficiency proposition (Proposition 4.51), not only a pattern
2603.06365	ESAA-Security: Event-Sourced Verifiable Architecture	2026-03	(iv) verifiable audit trail	pre-action enforcement and label-exclusive memory (Proposition 3.19)
2603.07670	Memory for Autonomous LLM Agents (survey)	2026-03	(i) taxonomy of compaction	continuity between incarnations, orthogonal to memory within one
2604.10352	Harness-Managed Virtual Memory	2026-04	(iv)* injection at compaction events	coevolution to a fixed point; the σ-chain as a proposition
2604.16548	A Survey on the Security of Long-Term Memory	2026-04	threats to agent memory, within and across agents	the operator as a possible adversary, which its threat model does not name
2604.19540	Mesh Memory Protocol	2026-04	(iii)* content-hash lineage	operator-prefix labels over several operators, and the DLG
2604.05119	Governance-Aware Agent Telemetry	2026-04	(ii) adjacent: telemetry	coevolution to a fixed point; telemetry only monitors

arXiv ID	Title (abbreviated)	Month	Axis overlap	Not provided
2604.19837	Forage V2: Knowledge Evolution and Transfer in Autonomous Agent Organizations	2026-04	(vi)* evolution of inheritance	inheritance parametrised by a commit κ, between HEAD and template
2604.11337	Parsonian Institutional Architecture	2026-04	(ii) institutional framing	a fixed point under tier sealing and tier progressivity (Proposition 4.28), with the quorum met at the fixed point of the add-only case (Proposition 4.29); the Parsonian framing is descriptive
2604.08551	Self-Sovereign Agent	2026-03	adjacent: pairing with an operator	pairing-private belief (Definition 10.9) and the non-propagating union (Definition 8.13)
2604.23210	Discovering Agentic Safety Specifications	2026-04	terminology only	the metacircular fixed point (Proposition 4.28)
2604.03515	Source-Code Taxonomy of Coding-Agent Architectures	2026-04	(i) taxonomy of compaction strategies	a set of conditions the strategies must satisfy, not a further strategy
2604.26197	A production long-term memory system	2026-04	deployment reference	not a framework; a deployed memory system

Table A.1. Comparison matrix. Three works, the Mesh Memory Protocol, Harness-Managed Virtual Memory and Forage V2, are directly adjacent on one axis each, (iii), (iv) and (vi). None covers more than one axis, and none the conjunction. Of the other ten, six touch an axis without covering it (the unstarred axis entries) and four touch none. The month is the arXiv submission date, which an identifier's own month need not match: 2604.08551 was submitted on 4 March 2026.

B σ-vocabulary v1.0 and chain-validation algorithm

The σ-vocabulary $\mathcal{V}_\sigma$ is the lexicon of event types of Definition 3.21: each type names an action whose effect is journaled as a σ-event in the chain $\mathcal{L}^c$ under σ-journaling. That the record precedes the effect is Proposition 3.22, under its hypotheses of an append-only journal and a gate. The vocabulary is itself a versioned artifact of the substrate, and changing it requires the amendment procedure of family 2 below.

B.1 Families of event types

The event types fall into seven families, and B.5 adds an eighth, the curator and finite-bounds family.

1. *Offspring lifecycle*, `foundation.protocols.ai.commons.offspring.{reproduced,first_onboard,weaned,merged,discarded,reaped}`: the lifecycle of an offspring under Definition 9.8.
2. *Amendment procedure*, `foundation.protocols.ai.commons.amendment_ceremony.{proposed,quorum_attested,enacted,retraction_pending,retracted,sunset_review}`: the quorum procedure, attended by operators, that changes any standing invariant, the vocabulary included.
3. *Autonomous execution*, `foundation.protocols.ai.autonomy.{task_plan,task_approval,kill_switch,correlation.fail,effect,bundle_projected,task_claimed,link_complete,bundle_auto_revoked_on_chain_transition}`: the execution of pre-permissioned bundles (Proposition 7.2).
4. *Interaction between operator and digital agent*, `humanagentics.interaction` and `interact.post`: interaction events public to the pairing.
5. *Conformance audit*, `ias.conformance.run_completed`: one cycle of the metacircularity audit.
6. *Substrate integrity*, `ias.identity.substrate_integrity_failure`: a detected violation of the cryptographic envelopes of the substrate.
7. *Continuity, σ-discipline and lock transfer*: the family detailed in B.4.

B.2 Record schema

Each σ-event is a record with the fields

```
{
 "ts": "<RFC 3339 timestamp, microseconds>",
 "session_id": "<UUID>",
 "commons_id": "<commons-id>",
 "agent_label": "<canonical label>",
 "actor_agent": "<canonical label of the emitting agent>",
 "event_type": "<dotted path in the vocabulary>",
 "summary": "<one-sentence summary>",
 "path": "<substrate object affected, if any>",
 "extra": { /* payload specific to the event type */ },
 "prev_hash": "<SHA-256 of the previous event's self_hash>",
 "self_hash": "<SHA-256 of this event, prev_hash included>"
}
```

Labels are the operator-prefix labels of Definition 9.2. An event that an operator emits carries the operator's handle in `actor_agent`, in place of a canonical label. The first event carries a genesis sentinel in `prev_hash`; the events that follow form a hash-linked list whose tail is the head of the chain.

B.3 Chain-validation algorithm

For a chain $\mathcal{L}^c = (e_1, e_2, \ldots, e_n)$:

1. check that e_1.prev_hash is the declared genesis value;
2. for each $i \in \{2, \ldots, n\}$, check e_i.prev_hash $= e_{i-1}$.self_hash;
3. for each i, recompute e_i.self_hash from the canonical serialisation of e_i without its `self_hash` field (RFC 8785), and check it equals the recorded value;
4. check that the tail's `self_hash` equals the last chain head recorded for the label in the commons state; when no head has yet been recorded, report that, rather than fail;
5. for each i, check e_i.event_type $\in \mathcal{V}_\sigma$ and that `extra` satisfies the schema of its type.

The algorithm is linear in n, deterministic and idempotent. A mismatch at any step is recorded as a substrate integrity failure on a separate chain, so that a violation of the primary chain can be recorded without breaking the primary chain's own invariants.

B.4 Continuity, σ-discipline and lock transfer

This family is the largest in the vocabulary. Every event in it carries the header of B.2 and the payload fields below.

B.4.1 Chain-integrity exceptions

The chain records its own integrity failures:

```
continuity.chain.content_changed_outside_protocol
 an event whose self_hash no longer validates.
 Payload: agent_label, expected and actual self_hash, event number, action.
 The session aborts and remediation by the operator is required (fail closed).

memory.replay_or_restore_initiated / memory.replay_or_restore_completed
 a pair bracketing the restoration of an agent's private memory.
 Payload: agent_label, memory object, restore source (commit or archive),
 target chain link, number of events replayed.

memory.chain_fork_detected
 two divergent branches of one agent's chain, as from concurrent writers.
 Payload: agent_label, fork point, the two branch heads, and the branch
 the operator attests as canonical.
```

B.4.2 Evolution of the cryptographic substrate

```
continuity.security.device_key_rotated
 every rotation of the device key of a label-bound account.
 Payload: agent_label, host_fingerprint, prior and new key identifiers,
 rotation time, reason (scheduled, forced, or replay defence).

continuity.security.vector_24_defense_invoked
 the defence against replaying a restored backup fires (the number in the name is a historic
 Payload: agent_label, trigger, the replayed chain link, action
 (refused or quarantined).
```

A key rotation fixes a verifiable time at which the device key changed; every later event claiming to come from the agent must validate against the new key.

B.4.3 Lock transfer between hosts

A summon moves an agent's locus from host A to host B, both authorised for the commons. It is meant to keep the single-host invariant (Definition 7.33): at every instant at most one host holds the agent's lock, and in between the agent is in transit (Definition 8.18). The protocol does not prevent two lock intervals from overlapping; it detects an overlap through the witness, and agreement between hosts on who holds the lock is open. The transfer is a protocol of four mandatory events:

```
agent.summon.requested
 the operator, or an offspring pre-permissioned to do so, asks to move
```

```
 agent_label from A to B.
 Payload: agent_label, source and target locus, time, requester.

agent.lock-transfer.released
 the session at A releases its lock: the moment of release.
 Payload: agent_label, source locus, time, final session, last self_hash.

agent.lock-transfer.acquired
 the new session at B acquires the lock: the moment of acquisition.
 Payload: agent_label, target locus, time, new session,
 prior_chain_link (= the released event's last self_hash).

agent.summon.witnessed
 a witness commons attests that the two lock intervals do not overlap.
 Payload: agent_label, source and target locus, witness commons, signature.
```

A successful summon is the chain requested, released, acquired, witnessed, and the link of `acquired` must equal the last hash of `released`; that is what makes the transfer continuous in the σ-chain. There are three failure modes. If B does not acquire within the timeout, the agent stays in transit until recovery. If no witness commons exists, a commons may attest its own transfer at establishment (Definition 8.15), and elsewhere the transfer is attended by the operator. If the lock is acquired at B before it is released at A, the witness refuses, and both sessions abort as concurrent writers.

The family serves σ-journaling (Definition 3.21) and the σ-discipline (Proposition 3.22), realises the summon of the multi-host substrate, guards the per-label memory (Definition 3.15) through its replay events, and uses the witness commons of Definition 8.15.

B.5 The curator and finite-bounds family

Under Axiom 3.1 and the curator procedure (Definition 7.28), six event types extend $\mathcal{V}_\sigma$ and the memory class $\mathcal{V}_\sigma^{(\mathrm{mem})}$ of Corollary 9.7:

Event type	Phase	Records
`substrate.curator.classify`	(i)	the classification of content as live, stale or overlapping
`substrate.curator.prune`	(ii)	the pruning of the live tier to its live content
`substrate.curator.archive`	(ii)	archival of stale content to the cold-storage tier (Definition 7.29)
`substrate.curator.consolidate`	(iii)	consolidation of overlapping content; an irreversible consolidation needs the operator's attestation
`substrate.coldstorage.rehydrate`	—	an operator-attested return of archived content to the live tier
`substrate.finitebounds.exceeded_blocked`	—	a write refused because it would break a declared bound with no recent curator pass

The four curator events form a chain parallel to the recombination chain `agent.recombination.{proposed,quorum_attested,completed,aborted}`. Rehydration requires the operator's attestation; the refusal event is emitted automatically, with no attestation gate. The extension is conservative: every earlier event type is unchanged, every earlier chain remains valid, and the validation algorithm of B.3 needs only the six new types added to its type check.

C Full proofs

This appendix gives the full arguments that the body states in short.

C.1 Proposition 4.51: the σ-chain is a sufficient statistic, preserved across π

Proposition (restated). *Assume σ-journaling (Definition 3.21). Fix $v \in V^c$ and $t \in I(v)$, and write Ξ_t for the σ-chain truncated at n_t. Assume for each $f \in \mathcal{D}_{\mathrm{IAS}}$ a read vocabulary T_f with (R) f reads only the events whose types lie in T_f, and (S) $T_f \subseteq \mathcal{V}_\sigma$. Then (i) $f(h_t) = f(\Xi_t)$ for every $f \in \mathcal{D}_{\mathrm{IAS}}$ and every history h_t; (ii) no number of compactions changes Ξ_t, so while its events are retained on $\mathcal{S}^c$ it can be recovered at any later time; (iii) if every type of $\mathcal{V}_\sigma$ is decision-relevant, no proper sub-vocabulary yields a $\mathcal{D}_{\mathrm{IAS}}$-sufficient chain.*

Proof. (i) Write $h \restriction_T$ for a history filtered to the event types in T. In the model, the σ-chain is $h_t \restriction_{\mathcal{V}_\sigma}$; that the journal realises this filter is the intended reading of σ-journaling, not a consequence of it. By (S), $(h_t \restriction_{\mathcal{V}_\sigma}) \restriction_{T_f} = h_t \restriction_{T_f}$, since filtering to $\mathcal{V}_\sigma$ first removes only events that T_f would remove anyway. By (R), $f(h) = f(h \restriction_{T_f})$ for every history h. Hence

$$f(h_t) = f(h_t \restriction_{T_f}) = f\big((h_t \restriction_{\mathcal{V}_\sigma}) \restriction_{T_f}\big) = f(h_t \restriction_{\mathcal{V}_\sigma}) = f(\Xi_t).$$

(ii) σ-events are written to the substrate $\mathcal{S}^c$, and a compaction π acts only on the agent's in-context representation (Definition 4.48), so the recorded history, and with it Ξ_t, is the same after any number of compactions. Recovery at a later time t' needs, in addition, that the events of Ξ_t are still on $\mathcal{S}^c$: a finite substrate may have to erase (Proposition 3.8), and an erased event cannot be recovered. That is the retention proviso. The validation algorithm of Appendix B.3 checks that what is recovered is an intact chain; it does not by itself recover anything.

(iii) Let $\mathcal{V}'_\sigma \subsetneq \mathcal{V}_\sigma$ omit a type τ. By decision-relevance some $f \in \mathcal{D}_{\mathrm{IAS}}$ depends on τ: there are two histories h_t and h'_t that differ only in a τ-event with $f(h_t) \neq f(h'_t)$. Their chains restricted to $\mathcal{V}'_\sigma$ coincide, because the one event in which they differ is filtered out, so $\mathcal{V}'_\sigma$ is not sufficient. Without decision-relevance the conclusion can fail: a type that no decision reads can be dropped at no cost.

What the Lean proves. (i) is `sigma_sufficient`, with (R) and (S) as its hypotheses `reads` and `span`, and `readRestrict_sigmaChain` the filtering identity; (ii)'s invariance is `sigmaChain_preserved_across_pi`; the construction of (iii) is `sigma_minimal_witness`, for a function that reads τ. That this function lies in $\mathcal{D}_{\mathrm{IAS}}$ is the decision-relevance hypothesis, not a consequence of the Lean, and the retention proviso of (ii) is not modelled, nor is the reading of the σ-chain as the filtered history. □

C.2 Proposition 4.30: the Kleene form of the add-only case

Proposition (add-only case, a variant: the Kleene form; text proof only). *Let* $(\mathrm{Spec}, \sqsubseteq, \bot)$ *be a pointed directed-complete partial order,* $\mathrm{accept}(x) := \{p : p \text{ is accepted under } x\}$ *and* $\Phi(x) := C(x, \mathrm{accept}(x))$, *with C and* accept *monotone and Φ Scott-continuous. Then Φ has a least fixed point* $\mathrm{Spec}^* = \bigsqcup_{n \geq 0} \Phi^n(\bot)$, *and it is a metacircular fixed point in the sense of Definition 4.25.*

Proof. Put $X_n := \Phi^n(\bot)$. Φ is monotone, as the composite of monotone maps, so $X_0 \sqsubseteq X_1 \sqsubseteq \cdots$ is an ascending chain, and by directed completeness it has a supremum $X^* := \bigsqcup_n X_n$. By Scott continuity,

$$\Phi(X^*) = \Phi\big(\bigsqcup_n X_n\big) = \bigsqcup_n \Phi(X_n) = \bigsqcup_n X_{n+1} = X^*.$$

Unfolded, $X^* = C(X^*, \mathrm{accept}(X^*))$: the self-referential equation of Definition 4.25, not merely $X^* = C(X^*, \emptyset)$. The monotonicity of accept is needed here; iterating $C(\cdot, \emptyset)$ alone solves only the equation without acceptance, and can stop at $\bot$. For leastness, let $Y = \Phi(Y)$. Then $Y \sqsupseteq X_0 = \bot$, and if $Y \sqsupseteq X_n$ then $Y = \Phi(Y) \sqsupseteq \Phi(X_n) = X_{n+1}$; so Y bounds every X_n and $Y \sqsupseteq X^*$. □

What the Lean proves. Existence and leastness of the fixed point for a monotone map on a complete lattice (`IasCert.Paper.MetacircMonotone`, Knaster–Tarski), and, for the absorptive C on the subsets of a finite set U, that the ascent from $\emptyset$ reaches the least fixed point within $|U|$ rounds (`IasCert.Paper.MetacircKleene`). The form stated here, on a pointed directed-complete partial order with Φ Scott-continuous, is this text proof only.

C.3 Proposition 4.56: unified continuity, claim by claim

The statement is in the body. Each claim follows from the root, the conditions and the results named below; the certificates are listed where the body states the proposition.

Claim (1), recoverability within a session. An element σ-recorded before the compaction event $c_k^{(v)}$, whose record is still retained at $c_k^{(v)}$ (no erasure has dropped it), has its record in $\mathbb{S}^c$ before $\pi_v^{(k)}$ acts (Definition 3.21), and $\pi_v^{(k)}$ acts only on the in-context representation (Definition 4.48). So $L_v^{(k)} \cap E_\sigma(v) = \emptyset$, which is Lemma 4.49 (machine-checked in part, Lean `Paper.SigmaRecLoss`); that the recovered record decides every question in $\mathcal{D}_{\mathrm{IAS}}$ is Proposition 4.51.

Claim (2), completeness of lineage. By Definition 9.8, each parent of every ancestor of v enters through a snapshot at a commit κ_p of its commons, and each reproduction or recombination is itself a σ-event. The ancestral subgraph $\mathcal{G}_v^{\mathrm{anc}}$ is therefore spanned by commits and σ-events already in the substrates of the participating commons. Within one commons these are readable by v directly. Across commons they are delivered by the across-commons sync (Proposition 8.8), which, under its delivery condition, carries a recombination event to each commons whose lineage took part and that chose to take it. Reading grants no write: no daemon of an agent writes into the private memory of another label (Proposition 3.19, under its conditions). This is a proof in the text, and it rests on the delivery condition and on each participating commons taking the recombination events offered to it; no certificate covers it.

Claim (3), recovery after hibernation. Committed state persists across hibernation (Definition 4.10), and the committed content grows monotonically along the commit graph, so the content committed at the end of an activity interval (Definition 4.11) is contained in what is committed at the start of any later one, except where the resource bound of Axiom 3.1 has evicted an older hibernating branch.

Claim (4), coherence within a commons. Suppose the next within-commons epoch completes, with every live agent of c taking part, and no divergence is introduced across its cut. The sync computes the join of the participants' views (Proposition 8.7); the join is commutative, associative and idempotent (Lemma 4.52, Lemma 4.53), so every participant ends the epoch holding the same join. Restricted to the elements committed before the cut, the two views $\mathcal{S}^{c,(v,t)}$ and $\mathcal{S}^{c,(w,t)}$ are then equal, and their symmetric difference on that set is empty. Under a partition the hypothesis fails, and so, as it must, does the conclusion.

Claim (5), coherence across commons. By Proposition 8.8 over the declared peer-set of c: under the delivery condition each commons holds what the others offered it and it chose to take, and in an open federation Lemma 4.54 gives agreement, by the argument of claim (4).

Claim (6), access to the current specification. A form of the coevolution operator (Definition 9.9) that satisfies tier sealing (MI) and tier progressivity (Q) has at most one metacircular fixed point (Proposition 4.28), and the fixed point exists when the settled tiers are realised in Spec,

for example when the tiered presentation is onto; a monotone form on a complete lattice, the add-only case, has a least fixed point (Proposition 4.30). Where the fixed point exists, the within-commons form gives $\mathrm{Spec}^{c*}(t)$. No specification is imposed across commons: for a commons that coevolves with others, $\mathrm{Spec}^{c*}(t)$ is its part of the set's joint fixed point (Proposition 4.33). □

C.4 Proposition 6.5: the Brent bound

Proposition (the weighted form, by the text argument below). *For a plan of total work T_1 and depth T_∞, the levelled greedy schedule on p agents finishes in $T_p \le T_\infty + (T_1 - T_\infty)/p$.*

Proof. First condense the plan: collapse each strongly connected component to one node (Tarjan's algorithm), charged its members' work, so the plan becomes acyclic (Lemma 6.2). Then split each node of work w into a chain of w unit tasks; this changes neither the total work T_1 nor the depth T_∞, now both counted in units. Give each unit the level equal to the length of the longest chain ending at it. Every precedence edge goes from a lower level to a higher one, so running the levels in order respects precedence, and each level is an antichain whose units can run side by side. There are exactly T_∞ levels. A level of w_i units takes at most $\lceil w_i/p \rceil \le 1 + (w_i - 1)/p$ steps on p agents, and summing over the T_∞ levels with $\sum_i w_i = T_1$ gives

$$T_p \le \sum_{i=1}^{T_\infty} \Big(1 + \frac{w_i - 1}{p}\Big) = T_\infty + \frac{T_1 - T_\infty}{p}.$$

This is Brent's bound [Bre74]. At unit weights, the argument from the levelling on is proved in Lean (Appendix D, row Paper.BrentOnLevels). Graham's factor $2 - 1/p$ of the optimum is classical and not used here. □

C.5 Proposition 6.7: the cost of replanning

Proposition (a fuller form). *If a failure loses at most the super-node in flight, a replanning step at the boundary of super-node s_j loses at most* $\mathrm{work}(s_j) \le \max_i \mathrm{work}(s_i)$, *plus a constant overhead for rescheduling.*

Proof. A longest chain ending at a node of level ℓ has one node at each level $0, \ldots, \ell$. So once the levels below the level ℓ_j of s_j are complete, the longest chain ending at such a node within the remaining plan has length $\ell - \ell_j + 1$: the remaining plan's levels are the old ones shifted down by ℓ_j, and they need not be recomputed. Completed levels stay complete. Replanning reschedules only the remaining plan; the committed tasks of s_j are kept, and only its uncommitted tasks are redone. The loss is therefore at most $\mathrm{work}(s_j)$, and since j is arbitrary, at most $\max_i \mathrm{work}(s_i)$. The premise, that a failure loses at most the super-node in flight, is assumed. □

C.6 Proposition 8.21: convergence of commons numbering

Proposition (restated). *With across-commons peering at tier 1 or above, propagation latency bounded by Δ_{fed}, one delivery order of the numbering proposals, and no new commons proposed meanwhile, the registry views of all participating commons agree on the commons-ids of the commons established by time t after at most $m\,\Delta_{\mathrm{fed}} + \Delta_{\mathrm{merge}}$, for m commons proposals outstanding at the start of the period, and the assignment is then one ordinal numbering shared by the participating commons.*

Proof. Within one sponsor's registry, assigning the next free integer is deterministic and ordered by that sponsor's σ-event times, so there is no conflict inside one sponsor. Two sponsors conflict only when they propose the same id for different commons within Δ_{fed} of each other. Peering at tier 1 requires the proposal to be witnessed by existing commons, and the across-commons transport delivers the proposals in one order; the one earlier in that order wins, with ties broken

by sponsor, and the later sponsor proposes again with the next free integer in the resolved view. By hypothesis every proposal reaches every participating commons within Δ_{fed}. A re-proposal is itself a proposal and needs its own round, but each round settles the earliest remaining proposal, so m proposed commons settle within m rounds; with the registry merging within Δ_{merge}, every view holds the same merge-ordered assignment after $m\,\Delta_{\text{fed}} + \Delta_{\text{merge}}$. This rests on the transport delivering one order of proposals to all participants. □

This concludes Appendix C.

D Formal certificates

Every self-contained result marked *machine-checked* above carries a Lean~4 certificate, in the manner of the OpenAI *ten-proofs* formalisations [Ope26b], verified by one command on a proof-checking cluster (Lean$_+$Mathlib [dMU21, The20]), from the public certificates and the public checker image that any reader can run. We state each certificate's status and axiom footprint explicitly: a footprint is read from its type signature and its `#print axioms` output, never asserted. The claim on offer is not *trust the theorem* but *re-run the check* — the proof-checker itself ships as a container, so *it checks here* re-derives as *it checks there*, and the axiom footprint printed below is the one the reader's own kernel will print. The ledger is an **honesty ledger**: the boundary between *machine-checked*, *partial*, and *planned / open* is stated per claim and is the point of the exercise.

How to re-check. The certificate tree ships in the release bundle with a pinned proof-checker image; a single run of the bundle's `check-all.sh` builds every module listed below from cold, replays each in an independent kernel checker, and reads every cited theorem's axiom footprint, and `#print axioms` on a theorem's fully qualified name reprints the footprint recorded in the table. The companion executable library replays the same propositions under randomised property-law suites, so a reader may follow one law from a machine-checked theorem to a running process. Rows are grouped by certificate universe; the *Result* column links to the statement above where this edition carries it, and otherwise names the certified object.

Status tiers. **machine-checked** — passes the one-command check: built green, replayed by the kernel checker, sorry-free, footprint within Lean's standard axioms and ledgered (*structural*: green with a stated hypothesis still to instantiate; *also trusts the compiled checker*: closed by `native_decide`, so the footprint adds Lean's compiled evaluator). **partial** — proved for a weaker statement than the one named, or authored and not yet checked; the ledger note says which. **stated** — statement design-verified, proof not yet authored. **planned** — named as future work, no module yet.

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Paper core — the machine-checked propositions of this edition			
Axiom-0 and the nine further principles as separate Lean classes (the historical axiom-base carrier; only the two roots are posited)	`IasCert.Paper.Axioms` `effect_has_record`, `persist_trans`, `sigma_merge_needs_no_axioms`	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Ablation v0: consistency witness	`IasCert.Paper.Ablation` `axiom_base_satisfiable`, `branch Persistence_independent`, `ritual Enforce_independent`, `finite Bounds_independent`, `boundary Maintenance_independent`, `single Host_independent`	**machine-checked**	0 sorry; standard axioms only
Carrier-refinement demotions: a skeletal liveness form and an onboarding-prefix derivation, a certificate of neither Proposition 4.39 nor Definition 8.15	`IasCert.Paper.Ablation Demotion` `ritualEnforce_from_onboarding_prefix`, `boundaryMaintained_from_genesis_persistence`	**machine-checked**	0 sorry; standard axioms only
Proposition 7.2	`IasCert.Paper.Galois Boundaries` `paper_disjointness`, `paper_recovery`, `iplikidi_galois`, `le_adj_phi`, `phi_adj_phi`, `enforced_galois` …	**machine-checked**	0 sorry; standard axioms only
Proposition 6.5, the level-sum arithmetic (the ceiling of w_i/p summed over the levels)	`IasCert.Paper.BMLSPar` `bmls_parallel_bound`	**machine-checked**	0 sorry; standard axioms only
Proposition 6.11	`IasCert.Paper.Disjoint Writes` `disjoint_writes_correct`	**machine-checked**	0 sorry; standard axioms only
Lemma 4.53	`IasCert.Paper.SigmaMerge` `sigma_idempotent`, `sigma_preserves_left`, `sigma_preserves_right`, `sigma_is_join`, `sigma_comm`, `sigma_assoc`	**machine-checked**	0 sorry; standard axioms only
Proposition 6.7	`IasCert.Paper.ReplanCost` `replan_cost_bound`, `replan_cost_bound_with_overhead`	**machine-checked**	0 sorry; standard axioms only
Proposition 5.4	`IasCert.Paper.Dormayadie` `clear_recover_externalise_lossless`, `dormayadie_lossfree`, `retention_proviso_nonvacuous`, `externalise_establishes_reachable`, `clear_never_resurrects`, `recover_never_resurrects`	**machine-checked**	0 sorry; standard axioms only
Proposition 5.4, its lift to a collective clear over agents with disjoint writes	`IasCert.Paper.JointClear` `clear_comm`, `clearAll_snap`, `recoverAll_snap`, `clearAll_ctx_notMem`, `recoverAll_ctx_notMem`, `jointClear_lossfree` …	**machine-checked**	0 sorry; standard axioms only
The self/non-self projection pi_i: well-defined	`IasCert.Paper.PiProjection` `proj_well_defined`, `local_write_scope`, `others_untouched`, `local_write_own`, `shared_sigma_governed`, `budget_untouched` …	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Proposition 4.19	`IasCert.Paper.KernelFeasible` `denied_action_blocked`, `granted_admissible`, `denied_meets_blocked`, `feasible_iff`, `granted_feasible`, `write_scope_action` …	**machine-checked**	0 sorry; standard axioms only
Proposition 7.25	`IasCert.Paper.ScanSoundness` `semantics_preserving`, `coverage_complete`, `idempotent`	**machine-checked**	0 sorry; standard axioms only
Proposition 8.19	`IasCert.Paper.SingleHostPreservation` `update_preserves`, `applyOp_preserves`, `inv_empty`, `singleHost_preserved_across_summons`	**machine-checked**	0 sorry; standard axioms only
The commit-graph model of branch persistence, in it Definition 4.10 holds by definition; the condition itself is not proved	`IasCert.Paper.CommittedStateMonotone` `gitState_monotone`, `hibernation_gap_preserves`, `committedState_monotone_obligation`	**machine-checked**	0 sorry; standard axioms only
Proposition 4.51	`IasCert.Paper.SigmaSuff` `readRestrict_sigmaChain`, `sigma_sufficient`, `sigmaChain_preserved_across_pi`, `sigma_minimal_witness`	**machine-checked** (structural)	0 sorry; standard axioms only
Proposition 4.30, (i): existence and leastness (Knaster-Tarski on a complete lattice); the Kleene formula is certified only on finite powersets (Paper.MetacircKleene)	`IasCert.Paper.MetacircMonotone` `metacirc_fixed_point`, `metacirc_lfp_least`, `metacirc_isLeast`	**machine-checked**	0 sorry; standard axioms only
Proposition 4.30, (ii), on the subsets of a finite U (Phi from C and accept; the Kleene ascent stops within as many rounds as U has elements)	`IasCert.Paper.MetacircKleene` `selfAmend_mono`, `selfAmend_inflationary`, `absorb_mono_spec`, `absorb_mono_accepted`, `absorb_never_retracts`, `phiAbsorb_apply` …	**machine-checked**	0 sorry; standard axioms only
Proposition 4.28, the main statement under (MI) and (Q); the examples, Example 4.31; the second sentence of Proposition 4.29 (lfp_phiAbsorb_eq, lfp_commitment_has_quorum)	`IasCert.Paper.MetacircTiers` `le_firstDiff`, `firstDiff_le`, `firstDiff_self`, `hprog_forces_injective`, `attenuates_of_causal`, `causal_of_attenuates` …	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Proposition 4.28, uniqueness, the tier read-out of the fixed point, and why keeping tier 0 is not enough	`IasCert.Paper.MetacircFloor` `unique_fixedpoint_under_MI_Q`, `floor_of_fixedpoint`, `tier_of_fixedpoint`, `fix_tier0_iterate`, `fix_tier0_rounds_do_not_settle`, `keepFloor_fixes_tier0` …	**machine-checked**	0 sorry; standard axioms only
Proposition 4.28, superseded: the contraction core is re-proved inside Paper.MetacircTiers (attenuates_contracts, metacirc_fixedpoint_of_causal), which certifies PrpMetacircFP; this module certifies no statement of this edition	`IasCert.Paper.MetacircBanach` `attenuating_contracts`, `metacirc_banach_fixedpoint`	**machine-checked**	0 sorry; standard axioms only
Newman's lemma (the abstract core of PrpDPOConfluence)	`IasCert.Paper.NewmanStar.trans`, `newman`	**machine-checked**	0 sorry; standard axioms only
Proposition 6.18	`IasCert.Paper.Reaper` `within_contract_left`, `never_reaps_protected`, `overrun_never_silently_left`, `reap_implies_reapable_and_over`, `never_hibernates_protected`, `hibernate_implies_reapable_and_over` …	**machine-checked**	0 sorry; standard axioms only
Proposition 4.56	`IasCert.Paper.UnifiedContinuity` `unified_continuity`	**partial** Only the conjunction is formal: the six components enter as hypotheses and are not derived; the instance over the component certificates is not built.	0 sorry; standard axioms only
Proposition 4.42, on the edge-set model	`IasCert.Paper.PhiIntSN` `vertices_mono`, `phiInt_lt_of_ssubset`, `erase_strict`, `merge_strict`, `dedup_strict`, `death_strict` …	**machine-checked**	0 sorry; standard axioms only
Proposition 8.7, superseded by Paper.SyncMerge (PrpWithinCommonsSync) and Paper.SyncAcross (PrpAcrossCommonsSync); certifies no statement of this edition	`IasCert.Paper.SyncDemotion` `withinCommonsSync_from_shared_boundary`, `acrossCommonsSync_content_from_federation`, `Tcommons_is_relational`, `Tcommons_syncs`, `tsync_core_ok`, `withinCommonsSync_independent` …	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
A derivation of the σ-discipline from the language-model substrate, its hypotheses restate the conclusion, so it is not counted as a proof of either principle; it certifies neither Definition 3.21 nor Proposition 3.22	`IasCert.Paper.SigmaDisc Demotion` `sigmaDisc_from_llm_substrate`, `sigmaDisc_from_journal`, `sigmaDisc_independent`, `tsig_excluded_on_llm_substrate`	**machine-checked**	0 sorry; standard axioms only
Conjecture 14.3, vocabulary only: the colimit and the gain check are defined, not derived, so this does not test the conjecture (the coupling-model rows do)	`IasCert.Paper.Apeiroma Colimit` `colimit_is_collective_act`, `apeiroma_is_collective_act_colimit_on_gle1`, `apeiroma_fails_off_gle1`, `apeiroma_is_collective_act_colimit_on_gle1_decide`, `apeiroma_fails_off_gle1_decide`	**partial**	0 sorry; standard axioms only
Proposition 3.19, from the enforced access path and the OS access check (named conditions)	`IasCert.Paper.MemInviol Demotion` `memInviol_from_boundary_enforcement`, `memInviol_from_os_dac`, `memInviol_independent`, `tmem_excluded_on_enforced_substrate`	**machine-checked**	0 sorry; standard axioms only
Proposition 3.22	`IasCert.Paper.Journal Reconstruction` `step_preserves_inv`, `run_preserves_inv`, `record_precedes_emitted_effect`, `record_index_lt_effect_index`, `disciplined_append`, `gstep_preserves` …	**machine-checked**	0 sorry; standard axioms only
Proposition 4.29, the first sentence, for an arbitrary monotone map and an arbitrary acceptance	`IasCert.Paper.CoevQuorum Condition` `fp_stable`, `accepted_at_fp_has_quorum`, `subquorum_cannot_move_fp`, `coevQuorum_sufficient_for_governed_fp`	**machine-checked**	0 sorry; standard axioms only
Axiom-0, product form with four legs	`IasCert.Paper.AxiomZero FiniteThermo` `axiomZero_satisfiable`, `finiteCarrier_independent`, `accessibility_independent`, `firstLaw_independent`, `secondLaw_independent`, `entropy_export` …	**machine-checked**	0 sorry; standard axioms only
Lemma 6.3	`IasCert.Paper.TBMLSLevels` `level_lt_of_edge`, `level_antichain`, `level_zero_iff`, `level_witness`, `chain_le_level`, `level_eq_longest`	**machine-checked**	0 sorry; standard axioms only
Axiom 3.1	`IasCert.Paper.AxiomZero Bridge` `finiteSubstrateBounds_of_finiteCarrier`, `finiteSubstrateBounds_of_thermoAxiomZero`	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Axiom-0, Carathéodory form (the root used in the text)	`IasCert.Paper.AxiomZero Caratheodory` `caratheodoryAxiomZero_satisfiable`, `caratheodoryAxiomZero_satisfiable_varying`, `battery_store_varies`, `battery_charging_moves_store`, `cancellation_of_additive`, `cancellation_of_weight_invariant` …	**machine-checked**	0 sorry; standard axioms only
Corollary 9.7, on the model of Paper.SigmaSuff: the sigma-chain is the history filtered to the sigma-vocabulary; (R) and (S) are hypotheses	`IasCert.Paper.PerSubgraph Sigma` `restrict_sigmaChain`, `per_subgraph_sufficient`, `per_subgraph_minimal`	**machine-checked**	0 sorry; standard axioms only
Lemma 4.49, an append-only journal and a compaction that rewrites only the in-context state (Dfn-CompactionProjection); an element is recoverable when in context or recorded	`IasCert.Paper.SigmaRecLoss` `recorded_persists`, `loss_disjoint_recorded`, `recorded_recoverable_run`	**machine-checked**	0 sorry; standard axioms only
Proposition 3.8, the floor Q >= n k_B T ln 2 from (E) and (H0)-(H2) as stated in the Landauer bridge, and the pigeonhole bound on retained traces; that the reuse forced by a full carrier is an erasure is not shown (as the text says). repair_pump_heat is the quantitative core of Cor-RepairPumpFloor	`IasCert.Paper.LandauerFloor` `landauer_floor`, `landauer_floor_pos`, `traces_bounded`, `no_injective_retention`, `repair_pump_heat`	**machine-checked**	0 sorry; standard axioms only
Proposition 4.45	`IasCert.Paper.BirthDeath Stationary` `detailed_balance`, `stationary`, `pi_sum`, `P_nonneg`, `P_row_sum`, `pi_zero_closed_form` …	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Proposition 4.45, part (iii): the lumped stationary law and the healthy mass, given strong lumpability onto the damage-count chain (a hypothesis)	`IasCert.Paper.LumpedDamage` `push_sum`, `lumped_stationary_is_pi`, `healthy_mass_eq_pi_zero`	**machine-checked**	0 sorry; standard axioms only
Proposition 4.45, part (iii), the reduction by strong lumpability (Kemeny-Snell)	`IasCert.Paper.StrongLumpability` `push_step`, `push_iterate`, `push_stationary`, `block_path_markov`, `lumped_nonneg`, `lumped_stochastic`	**machine-checked**	0 sorry; standard axioms only
Proposition 6.6	`IasCert.Paper.GovernedWorkDepth` `depth_le_governed`, `governed_eq_of_zero`, `governed_mono`, `governed_critical_attained`, `governed_excess_le`, `governed_excess_eq_iff` …	**machine-checked**	0 sorry; standard axioms only
Proposition 6.5, at unit weights, on the ASAP levels of a finite DAG; splitting a weighted node into a unit chain is the text proof	`IasCert.Paper.BrentOnLevels` `counts_sum`, `level_down`, `level_inhabited`, `count_pos`, `depth_eq`, `brent_on_levels` …	**machine-checked**	0 sorry; standard axioms only
Lemma 6.2	`IasCert.Paper.Condensation` `reach_of_cedge`, `condensed_path_forward`, `condensation_acyclic`, `cycle_in_component`, `sameComp_equivalence`	**machine-checked**	0 sorry; standard axioms only
Definition 7.4	`IasCert.Paper.NonSelfLiftable` `liftable_iff`, `nonSelfLiftable_of_deny`, `nonSelfLiftable_of_outside`	**machine-checked**	0 sorry; standard axioms only
Proposition 6.11	`IasCert.Paper.OrderFreeCommit` `bernstein_symm`, `f_after`, `commute`, `commit_perm`, `blind_commit_perm`, `read_condition_needed`	**machine-checked**	0 sorry; standard axioms only
Proposition 7.15	`IasCert.Paper.NoDeadlock` `gated_green`, `gated_red`, `no_deadlock`, `red_reaches_green`	**machine-checked**	0 sorry; standard axioms only
Definition 6.13	`IasCert.Paper.SubscriptionQueue` `acquire_occupied`, `acquire_fifo`, `acquire_fifo_session`, `perturn_pass_keeps_head`, `acquire_holds`, `release_nonholder` …	**machine-checked**	0 sorry; standard axioms only
Remark 4.23	`IasCert.Paper.ControlledUnichain` `reaches_checkpoint`, `closed_reach`, `closed_contains`, `recurrent_closed`, `recurrent_minimal`, `recurrent_communicates` …	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Proposition 7.14	`IasCert.Paper.Gateway Obligations` `mem_mergeObs`, `nodup_mergeObs`, `prefix_mergeObs`, `mem_combine`, `nodup_combine`, `rank_le_combine` …	**machine-checked**	0 sorry; standard axioms only
Proposition 8.8	`IasCert.Paper.SyncAcross` `post_eq_of_delivered`, `exchange_lands`, `own_kept`, `declined_keeps_own`, `offer_is_private`, `open_views_agree` …	**machine-checked**	0 sorry; standard axioms only
Proposition 4.33	`IasCert.Paper.Metacirc Coevolve` `rounds_forget_start`, `stabilises`, `fixedpoints_agree`, `limit_fixed`, `exists_unique_of_causal`, `joint_causal` …	**machine-checked**	0 sorry; standard axioms only
Proposition 8.4	`IasCert.Paper.Commons Identity` `id_binds_genesis`, `next_key_forced`, `prerotation_blocks_thief`, `thief_cannot_extend`, `duplicity_commits_differ`, `first_difference` …	**machine-checked**	0 sorry; standard axioms only
Proposition 8.9	`IasCert.Paper.CrossWitness` `witnessed_prefix_fixed`, `later_head_fixes_earlier`, `inconsistent_heads_no_log`, `safe_to_replace`, `extends_or_witness`, `split_view` …	**machine-checked**	0 sorry; standard axioms only
Proposition 4.39, reduction to damage + continuation (over stationary laws)	`IasCert.Paper.Boundary MaintenanceRepair` `no_repair_absorbs`, `no_repair_stationary`, `repair_needed`, `formula_continues_without_repair`	**machine-checked**	0 sorry; standard axioms only
Proposition 4.39, the work inequality $0 < b*c < d*c$ given repair outpacing damage and the Landauer bridge $c>0$ (a named condition, not Axiom-0)	`IasCert.Paper.Boundary MaintenanceWork` `repair_outpaces_damage`, `maintenance_work_pos`, `continues_of_outpaces`	**machine-checked**	0 sorry; standard axioms only
Proposition 8.7	`IasCert.Paper.SyncMerge` `post_eq_join`, `views_agree`, `nothing_lost`, `partition_views_disagree`, `full_merge_complete`	**machine-checked**	0 sorry; standard axioms only
The coupling model of the Apeirōma conjecture (join preservation; finite when every gain is at most 1; divergence on an amplifying two-cycle)	`IasCert.Paper.Apeiroma Coupling` `round_mono`, `phi_fixed`, `input_le_phi`, `response_le_phi`, `phi_mono_input`, `mul_sup_le` …	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
The coupling model of the Apeirōma conjecture, completed: the exact cycle condition for walks of any length, preservation of arbitrary joins of inputs (Scott continuity), and saturation on a capped carrier	`IasCert.Paper.ApeiromaCycles` `endp_append`, `wg_append`, `endp_snoc`, `wg_snoc`, `wg_le_pow`, `exists_simple_walk` …	**machine-checked**	0 sorry; standard axioms only
Proposition 5.6, health levels Fin 3 merged by min; the dependency closure given as a finite set per node, transitive and acyclic for (iii)	`IasCert.Paper.NodeHealth` `self_antitone`, `prop_le_self`, `unhealthy_dep_blocks`, `runnable_downward_closed`, `cycle_counterexample`	**machine-checked**	0 sorry; standard axioms only
Proposition 10.13, a finite non-empty set of admissible vendors, real values U v and u0, and individual rationality u0 <= U v - r	`IasCert.Paper.PriceOfDelegation` `rent_le_surplus`, `rent_le_best`, `rent_zero`	**machine-checked**	0 sorry; standard axioms only
Proposition 5.7, the single-node case: determinacy (the working state depends only on the dependency closure) is a hypothesis; the collective lift is Paper.JointClear	`IasCert.Paper.CheckpointSuff` `checkpoint_suff`, `coverage_needed`	**machine-checked**	0 sorry; standard axioms only
Axiom 3.9, a consistency and independence witness for the root's two legs (the root is posited, not proved)	`IasCert.Paper.HICLegs` `legs_consistent`, `hic2_independent`, `hic1_independent`, `llmFreeable_of_hic1`	**machine-checked**	0 sorry; standard axioms only
Operating-system-subset containment — kernel confinement			
L_A1_memiso	`IasCert.OSSubset.LA1MemIso` `L_A1_memiso`, `targetImmutable_step`, `targetImmutable_run`	**machine-checked**	0 sorry; standard axioms only
The capability-only-own bridge, derived from L_read_iso	`IasCert.OSSubset.CapOnlyOwnBridge` `cap_only_own_from_LReadIso`, `cap_only_own_abstract`	**machine-checked**	0 sorry; standard axioms only
L_commit	`IasCert.OSSubset.LCommit` `L_commit`	**machine-checked**	0 sorry; standard axioms only
L_mount_iso	`IasCert.OSSubset.LMountIso` `L_mount_iso`, `PGDisjoint_step`, `PGDisjoint_init`	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
L_ancestor	`IasCert.OSSubset.LAncestor` `L_ancestor`, `c1_necessary`, `c2_necessary`, `c3_necessary`, `c4_necessary`	**machine-checked**	0 sorry; standard axioms only
L_config trio	`IasCert.OSSubset.LConfig` `L_config_immutable`, `L_config_keyanchor`, `L_config_layout`	**machine-checked**	0 sorry; standard axioms only
system_security	`IasCert.OSSubset.System Security` `system_security`, `systemSecure_iff`	**machine-checked**	0 sorry; standard axioms only
unified_system_security	`IasCert.OSSubset.Carrier Unified` `unified_system_security`, `gMemIso_of`, `gCommit_of`, `gMountIso_of`, `gAncestor_of`, `gConfig_of`	**machine-checked**	0 sorry; standard axioms only
OS axiom-demotion ledger	`IasCert.OSSubset.OSLedger` `row1_is_theorem`, `row2_is_theorem`, `row3_is_theorem`, `row4_is_theorem`, `row5_immutable_is_theorem`, `row5_keyanchor_demoted` …	**machine-checked**	0 sorry; standard axioms only
L_read_iso	`IasCert.OSSubset.LReadIso` `opAgents_mem_iff`, `groupother_pnone_nonowner_denied`, `l1_crossLabel_denied`, `l1_owner_canRead`, `l2_crossOperator_denied`, `l3_a2aShared_ok` …	**machine-checked**	0 sorry; standard axioms only
L_ancestor reinstated	`IasCert.OSSubset.LAncestor Reinstantiate` `t1_c1BlocksRedirect`, `t2_c1Necessary`, `newLayout_oldC1_stale`, `newLayout_reinstatedC1_ok`, `newLayout_noRedirect`, `lAncestorC1Reinstated`	**machine-checked**	0 sorry; standard axioms only
L_a2a_no_memcorrupt	`IasCert.OSSubset.LA2aNo Memcorrupt` `attacker_cannot_write_memB0700`, `mutatesMemB_0700_false`, `bmcNoMemcorruptHolds`, `bmcAblatedCounterexample`, `laNoMemcorruptLoadBearing`	**machine-checked**	0 sorry; standard axioms only
L_transcript_lifecycle	`IasCert.OSSubset.LTranscriptLifecycle` `rotated_inviolable_agent`, `rotated_inviolable_operator`, `rotated_is_immutable`, `live_crossAgent_private`, `offsite_carrier_valid`, `exfat_carrier_excluded` …	**machine-checked**	0 sorry; standard axioms only
per-operator substrate security	`IasCert.OSSubset.PerOperatorSubstrate` `perOperator_substrate_secure`	**machine-checked**	0 sorry; standard axioms only
per-operator system security	`IasCert.OSSubset.PerOperatorSystemSecure` `newLayout_reinstatedC1`, `full_ancestor_redirect_safe`, `perOperator_system_secure`	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
Substrate enforcement tier — models of the gateway's decision laws (deployed in shadow mode)			
substrate_tier_live	`IasCert.Substrate.Scaffold` `substrate_tier_live`	**machine-checked**	0 sorry; standard axioms only
AdditiveEnforce	`IasCert.Substrate.AdditiveEnforce` `enforce_preserves_hook`, `enforce_preserves_pdp`, `enforce_allow_iff`, `enforce_le_of_both`, `combine_eq_or`, `rank_le_combine_left` …	**machine-checked**	0 sorry; standard axioms only
DenySolePath	`IasCert.Substrate.DenySolePath` `denyVerdict_iff_denytier`, `deny_never_feasible`, `feasible_iff_enforced`, `feasible_disjoint_deny`, `denytier_rank_top`	**machine-checked**	0 sorry; standard axioms only
HookPairEquivPDP	`IasCert.Substrate.HookPairEquivPDP` `combine_idem`, `hookpair_eq_decide`, `harness_independent`	**machine-checked**	0 sorry; standard axioms only
NeedsApprovalConcage	`IasCert.Substrate.NeedsApprovalConcage` `gated_iff`, `gated_routes_needsApproval`, `ungated_auto_allow`, `gated_not_allow`, `gated_needs_recorded_approval`, `approval_releases` …	**machine-checked**	0 sorry; standard axioms only
The aggregation of every IasCert module: all link and build	`IasCert`	**machine-checked**	0 sorry; standard axioms only
Licence flow — admissibility and the enclosure moat			
License-flow admissibility: decidable, one-gate repair	`IasCert.License.Admissibility` `eff_lfp_wellDefined`, `admissibility_decidable`, `single_gate_repair`	**machine-checked**	0 sorry; standard axioms only
Enclosure is unsatisfiable within the law	`IasCert.License.AntiEnclosure` `enclosure_unsat_within_law`, `muc_agpl_s13_gate`	**machine-checked**	0 sorry; standard axioms only
Beyond the law: AGPL is necessary, not sufficient	`IasCert.License.ViolatorNecNotSuf` `inspection_game_deterrence`, `violation_profitable_implies_no_deterrence`, `sigma_chain_detects_provenance`	**machine-checked**	0 sorry; standard axioms only
Companion volume — worked / self-applied certificates			
E1.7	`IasCert.Book.Unit01` `segOp_assoc`, `segOp_identity`	**machine-checked**	0 sorry; standard axioms only
U6.L1: Strassen recurrence sampled at n=2^k	`IasCert.Book.Unit06` `T2_one`, `T2_two`, `T2_three`, `T2_four`, `T2_leading`, `strassen_C11_identity`	**machine-checked**	0 sorry; standard axioms only

Result	Lean module & key theorems	Status	Footprint (#print axioms)
U13.L1: the combinatorial MinHash core	`IasCert.Book.Unit13` `unifProb_mem_unit_interval`	**machine-checked**	0 sorry; standard axioms only
U3.L1: the two makespan lower bounds behind Graham …	`IasCert.Book.Unit03` `lb1`, `lb2`	**machine-checked**	0 sorry; standard axioms only
Federation coexistence / MARD			
Theorem 11.1	`IasCert.Coexistence.GainBound` `runChain_le`, `mad_unreachable`, `gainBoundOK_chain`, `escalation_of_gain_gt`, `gainBound_holds`, `gainBound_holds_decide`	**machine-checked**	0 sorry; standard axioms only
Theorem 11.8	`IasCert.Coexistence.ThermoSolvency` `thermoSolventOK_iff`, `solvent_under_loss`, `ledger_solvent_under_loss`, `thermoSolvency_holds`, `thermoSolvency_holds_decide`	**machine-checked**	0 sorry; standard axioms only
Proposition 11.7, (i) and (ii) as compositions of GainBound (A) and ThermoSolvency (D); (B), (C), (E), (F), (G) enter only through the hypothesis (H), which is assumed	`IasCert.Coexistence.Federation` `coexists_at_instant`, `attrited_solvent`, `coexists_over_horizon`	**machine-checked**	0 sorry; standard axioms only

Ledger totals: 92 machine-checked, 2 partial, 0 stated / planned (across the paper core and the operating-system, substrate, licence, companion-volume and coexistence certificate universes).

D.1 What the two-root floor makes of the other principles

The certificates behind the floor paragraph of the section on finite bounds and the human-in-commons, one row per principle. *Proposition* means the principle is proved, under the conditions its row names, from a model of the substrate that realises it, with no domain axiom in its `#print axioms` footprint; *condition* means a named hypothesis, stated by each result that uses it, with no claim that it holds. Only the two roots are posited; there are no other axioms.

Table 6: The principles beside the two roots, and the certificate behind each.

Principle	**What it is now**	**Certificate**	**Caveat that remains**
Axiom-0 (finite substrate bounds)	root: physics, in Carathéodory form	`AxiomZeroCaratheodory`	posited, not proved: the Lean certificate shows its five legs consistent and independent; the pump face is not posited; it has no Haskell twin yet; that the physical world satisfies the legs is not a formal claim
Human-in-commons	root: a choice, never derived	`HICLegs`	none; it is posited on purpose
Memory inviolability	proposition, for writes by the daemons of digital agents: follows from enforcement and write confinement under the modelled discretionary access control of the operating system (owner-only private memory, no override capability), confinement being a theorem of the operating-system model	`MemInviolDemotion`, `CapOnlyOwnBridge`	two model-fidelity seams, the enforced access path and label-to-user correspondence, and the ownership-and-mode clause of the access-control model, which is not a named condition; writes by operators, or by processes that realise no agent, are outside the result
σ-journaling	condition: every in-vocabulary effect is journaled, which gives the records, not their order; the record-before-effect order, the σ-discipline, is a separate proposition (shallow, Proposition 3.22) under an append-only journal with a gate that refuses an effect with no decision record, and it is not derivable from σ-journaling (`recordExists_not_disciplined`)	— (`JournalReconstruction`, `grun_disciplined`, certifies Proposition 3.22, not this condition)	the route to the σ-discipline from the language-model substrate (`SigmaDiscDemotion`) restates its conclusion, so it does not count
Within-commons synchronisation	proposition (shallow), conditional on merge completion (no partition)	`SyncMerge`	merge completion is an operational hypothesis; join-closure alone is independent (`SyncDemotion`)

Principle	What it is now	Certificate	Caveat that remains
Branch persistence	condition; in the commit-graph model it holds by definition	`CommittedStateMonotone` (definitional)	a proposition needs a model in which commits can be removed
Single-host invariant	condition in its multi-host form; the per-host form (at most one live session per label per host) is a proposition (shallow), preserved by every lease-mediated summons	`SingleHostPreservation`	at most one session across hosts needs an agreement service or timing assumptions
Across-commons synchronisation	proposition (shallow) under the delivery condition, with a sync window at least the delivery bound; each commons ends with its own view plus what passed to it, and in an open federation the views agree	`SyncAcross`	the delivery condition must be provided by the transport
Ritual enforcement	condition; its only derivation assumes an onboarding-first hypothesis as strong as itself	— (`AblationDemotion` does not count)	the onboard-first hypothesis is not discharged from independent structure
Boundary maintenance	proposition (shallow) under three conditions: damage at a positive rate, continuation over the chain's stationary laws, and the Landauer bridge (each repair costs at least $c_{\mathrm{rep}} > 0$); repair is derived to happen and outpace damage, $0 < d$ and $b < d$, and $0 < b\,c_{\mathrm{rep}} < d\,c_{\mathrm{rep}}$; reading this as positive maintenance work per step is not certified	`BoundaryMaintenanceRepair`, `BoundaryMaintenanceWork`	the Landauer bridge is the hypotheses of Proposition 3.8, not Axiom-0
Coevolution quorum	condition, used as a sufficient condition for every proposal accepted at the least fixed point of the add-only case to meet quorum (Proposition 4.29); the fixed point of Proposition 4.28 rests on tier sealing and tier progressivity, which are quorum-free	`CoevQuorumCondition`	necessity is open, and so is a derivation from the human-in-commons

Every certificate in the table uses only the standard axioms of Lean's kernel, and every certificate

of Appendix D passes one command, the bundle's check-all.sh, run 2026-09-29. The earlier axiom-base witness `IasCert.Paper.Ablation` shows the principles jointly satisfiable when all are taken as axioms, and five of them (branch persistence, ritual enforcement, finite bounds, boundary maintenance, the single-host invariant) independent of the rest.

E The agent landscape, re-read in September 2026

IAS is not a harness, an orchestrator or a sandbox. It is a governance and identity plane meant to compose with any of them, and it credits each for the ingredients it provides. This appendix places it against the agent ecosystem as it stood on 28 September 2026: the open-source systems, and one proprietary system, Devin, read for comparison.

E.1 How the landscape was read

In August 2026 we wrote a dossier for each of twenty-seven systems named in one reference map of the ecosystem, grounding every claim in a file and line of the code at a pinned commit, or in the vendor's documentation where the code is not public. On 28 September 2026 we read again the release notes or repository pages of the systems in the table below, and the documentation of nine systems that are new or that the August map omitted: the Microsoft Agent Framework, Devin, and the seven listed under documentation only below. Every page this appendix relies on was retrieved that day and archived with its SHA-256 digest. For the systems read only from documentation we report what the documentation says, and we claim no absence: a feature a page does not mention may still exist in the code.

E.2 The August finding

The twenty-seven were nine harnesses and runtimes (OpenHands, Hermes Agent, the Claude Agent SDK, Aider, Omnigent, Flue, Nonstop Agent, the pi coding agent, Meta-Agent), ten orchestration frameworks (LangGraph, CrewAI, AutoGen, LlamaIndex, smolagents, Pydantic AI, Mastra, Semantic Kernel, the OpenAI Agents SDK, Google's Agent Development Kit), four sandboxes (E2B, Daytona, SandboxAI, Docker Sandboxes), three evaluation suites (Harbor, Terminal-Bench, SWE-bench) and one design doctrine (the Twelve-Factor Agents principles). Approval plumbing was common: pending-change inboxes, confirmation policies, per-tool approval flags. Audit trails, where present, were plain append files or telemetry spans, and the strongest control on the data side matched patterns against labels. None of the twenty-three harnesses, frameworks and sandboxes among them combined a governed loop for promoting capabilities, a hash-chained audit trail, enforced governance of the data an agent reads, and an identity that outlasts a process.

E.3 What changed by 28 September

System	What its pages show on 28 September 2026	Bearing on the four
Hermes Agent	Version 0.21.5 (24 September). Protected agent-instruction files, skills and memory stores now always require write approval, so an injected agent cannot quietly rewrite its standing orders.	Write approval on memory, which moves toward governed memory. No chained audit and no governance of reads is documented.

System	What its pages show on 28 September 2026	Bearing on the four
Google ADK	Version 2.10.0 (24 September) adds an experimental, off-by-default skill lifecycle: skills that last one turn, and a tool to unload them.	Control of what is loaded, not a governed promotion loop.
Microsoft Agent Framework	The direct successor to AutoGen and Semantic Kernel, by the same teams; its harness agent has tool approval (with "don't ask again"), session state and telemetry.	Replaces AutoGen and Semantic Kernel in the roster. Approval plumbing and telemetry.
OpenAI Agents SDK	Version 0.22.x: conditional tool approvals checked against the validated arguments, and SQLite-backed sessions.	Approval plumbing.
CrewAI	Version 1.15.22 (16 September) records human feedback and pause events in its tracing.	Telemetry.
Omnigent	The 24 September release keeps permission modes across relaunches.	Approval plumbing that persists.
Meta-Agent	Every candidate harness keeps the evidence needed to audit the search.	An evidence trail for its own search; not a chained audit of actions.
Daytona, Devin	Daytona's platform lists audit logs; Devin's enterprise API serves audit logs.	Audit logs held by the vendor; no tamper-evidence is documented.
pi	Its repository states that it has no built-in permission system: it runs with the permissions of the user who launched it, and stronger boundaries are left to a container or sandbox.	None of the four.
LangGraph, LlamaIndex, Pydantic AI, Mastra, the Claude Agent SDK, E2B, Docker Sandboxes, Harbor	Releases in September; none of the release notes read touches the four.	Unchanged.
Aider, smolagents	No release since 9 August and 29 May respectively.	Unchanged.

E.4 Systems read from documentation only

Seven more open-source agents are widely used and were not in the August map: DeepSeek Harness (in developer preview, MIT-licensed, built so that everything is a plugin), OpenCode (whose planning agent is read-only by default and asks before running a shell command), goose (a general-purpose agent that runs on the user's machine), the Codex command-line agent (which documents its sandboxing and approval model), Qwen Code (with automatic memory, automatic skills and sub-agents), Cline (which separates safe commands from those that need approval, and offers a mode that approves everything), and LangChain's Open SWE (with human approval before it pushes changes to workflow files, read-only reviewer agents, and credentials kept in the

server process). Their documentation describes approval modes of the same kind; none of the pages we read describes a hash-chained audit trail or governance of what an agent reads. Their code was not read, so this is a statement about their documentation only.

E.5 What this shows

In the systems whose code was read in August, the combination IAS is designed to provide was not present, and none of the release notes or pages read in September documents it. What has spread is approval plumbing, including approval of writes to memory (Hermes Agent) and of changes to automation (Open SWE). The survey is a snapshot of fast-moving projects, and a later release may close any of these gaps; we would welcome correction. The σ-log of IAS is hash-chained, which makes corruption evident; until its chain head is checkpointed outside the writer's control, as by heads signed and held by peers (Proposition 8.9), it is corruption-evident rather than tamper-proof.

E.6 Related systems research

The closest overlap with the σ-discipline is harness-managed virtual memory [RB26] (Table A.1), a complementary layer: it manages the memory of one session, where IAS manages the repository that outlives sessions. Event sourcing for autonomous agents [BdSF26a, BdSF26b] is the design precedent for the trace and its named audit invariants, and the survey of memory for autonomous agents [Du26] supplies the write–manage–read loop this paper adapts. Among self-improving agents, Voyager [WXJ+23] builds a library of skills; skills are verified by co-evolution in [ZFZ+26], multi-agent evolution replaces fixed search heuristics in [QZZ+26], context compression reduces peak tokens in [KCH+26], and computer-use agents evolve their own skills in [SLZ+25], draw on a library of developed skills in [CLS+26], and combine GUI actions with tool calls in [YYD+25]. The reset-free self-improvement of [KZU+26] removes the human from the reset loop; IAS keeps the operator's attestation at the surfaces that matter, which is the sharpest contrast in the set. On harnesses, the industry account of harness engineering [Böc26] and the open-source supervisor [Refnd] are the nearest practice, and code as a harness [N+26] names verification under incomplete feedback as an open problem. The promptware kill chain of [BFSN26] has seven stages, one of them persistence by poisoning memory and retrieval; IAS is designed to guard, at the substrate hook layer, the ways injected text reaches the context: tool output, recalled memory and stored skills; on-host enforcement is exercised in shadow mode only.

AI-use disclosure

The human author takes full responsibility for all content of this work. Large-language-model software was used as a tool, under a discipline that keeps its output *checked* rather than unchecked: load-bearing actions are designed to be journaled to an append-only provenance record before they take effect (on-host enforcement is exercised in shadow mode only); external claims were checked against cited sources where retrievable, not asserted from a model's prior; formal results are gated by machine-checked proof certificates where Appendix D lists one, and the rest carry text proofs with their status stated; and claims were adversarially cross-reviewed to narrow scope rather than overclaim, with open questions stated as conjectures. No software tool is an author. Authorship and responsibility rest with the named human operator, who sets the purpose these tools serve.

References

[ABJH23] Nivasini Ananthakrishnan, Stephen Bates, Michael I. Jordan, and Nika Haghtalab. Delegating data collection in decentralized machine learning. 2023. AISTATS 2024, PMLR 238.

[ADF+25] Damiano Azzolini, Marco Duca, Stefano Forti, Francesco Gallo, and Antonio Ielo. Application placement with constraint relaxation. *Theory and Practice of Logic Programming*, 25:437–454, 2025.

[Agh86] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[AGV73] Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. *Théorie des topos et cohomologie étale des schémas (SGA 4)*. Lecture Notes in Mathematics 269/270/305. Springer, 1972–1973. ind-objects.

[aia24] Regulation (EU) 2024/1689 of the european parliament and of the council (artificial intelligence act). Official Journal of the European Union, L series, 12 July 2024, 2024.

[AM95] Malcolm P. Atkinson and Ronald Morrison. Orthogonally persistent object systems. *The VLDB Journal*, 4(3):319–401, 1995.

[And72] James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Electronic Systems Division, Air Force Systems Command, 1972. Origin of the reference-monitor concept.

[AR94] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*, volume 189 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1994.

[BA26] Block, Inc. and Agentic AI Foundation (Linux Foundation). goose: an open source, extensible AI agent, 2026. Rust + Tokio HTTP daemon plus ACP server; subagent-as-tool; PermissionManager tool-call permission control; Apache-2.0. Repository moved from block/goose to aaif-goose/goose when the project joined the Agentic AI Foundation (Linux Foundation) in Dec 2025.

[BCOQ92] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley, 1992.

[BdSF26a] Elzo Brito dos Santos Filho. ESAA: Event sourcing for autonomous agents in LLM-based software engineering. February 2026.

[BdSF26b] Elzo Brito dos Santos Filho. ESAA-security: An event-sourced, verifiable architecture for agent-assisted security audits of AI-generated code. March 2026.

[Ben73] Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

[Ben06] Yochai Benkler. *The Wealth of Networks*. Yale University Press, 2006.

[BFSN26] Oleg Brodt, Elad Feldman, Bruce Schneier, and Ben Nassi. The Promptware Kill Chain: How Prompt Injections Gradually Evolved Into a Multistep Malware Delivery Mechanism, January 2026. Seven-stage attack-chain framing for prompt-injection-derived malware on LLM-driven applications.

[Bir40] Garrett Birkhoff. *Lattice Theory*, volume 25 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, 1940.

[BJSS22] Stephen Bates, Michael I. Jordan, Michael Sklar, and Jake A. Soloff. Principal-agent hypothesis testing. 2022.

[BL73] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, Vol. I, The MITRE Corporation, 1973.

[Ble96] Guy E. Blelloch. Programming parallel algorithms. *Communications of the ACM*, 39(3):85–97, 1996.

[Böc26] Birgitta Böckeler. Harness engineering for coding agent users. https://martinfowler.com/articles/harness-engineering.html, April 2026. Published 2 April 2026. Archived sha256 2ed-cadbd37ca25ede39a0e3dd0321b909824690c85727664a6444476780ef258.

[BPK21] Martin Biehl, Felix A. Pollock, and Ryota Kanai. A technical critique of some parts of the free energy principle. *Entropy*, 23(3):293, 2021. arXiv:2001.06408.

[Bra87] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[Bre74] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, 1974.

[But10] Peter Butkovič. *Max-linear Systems: Theory and Algorithms*. Springer Monographs in Mathematics. Springer, 2010.

[Car09] Constantin Carathéodory. Untersuchungen über die grundlagen der thermodynamik. *Mathematische Annalen*, 67:355–386, 1909.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252. ACM, 1977.

[CCC+22] Sarah Chasins, Alvin Cheung, Natacha Crooks, Ali Ghodsi, Ken Goldberg, Joseph E. Gonzalez, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, Michael W. Mahoney, Aditya Parameswaran, David Patterson, Raluca Ada Popa, Koushik Sen, Scott Shenker, Dawn Song, and Ion Stoica. The sky above the clouds: A berkeley view on the future of cloud computing. *arXiv preprint arXiv:2205.07147*, 2022.

[CDE+24] Joseph W. Cutler, Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, Eleftherios Ioannidis, John Kastner, Anwar Mamat, Darin McAdams, Matt McCutchen, Neha Rungta, Emina Torlak, and Andrew M. Wells. Cedar: A new language for expressive, fast, safe, and analyzable authorization. *Proc. ACM Program. Lang.*, 8(OOPSLA1):Article 118, April 2024. Lean-verified policy language; forbid-overrides-permit proven order-independent.

[CL90] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2–3):213–261, 1990.

[Cli81] William D. Clinger. *Foundations of Actor Semantics*. PhD thesis, MIT, 1981. AI-TR-633; the actor LOCALITY laws.

[CLO18] Clarifying lawful overseas use of data act (CLOUD act). Pub. L. No. 115-141, div. V (Mar. 23, 2018), 2018.

[CLS+26] Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, et al. CUA-Skill: Develop Skills for

Computer Using Agent, 2026. Skill-development for computer-using agents. Cited as intersection of EvoSkills-style skill-evolution with computer-use.

[cra24] Regulation (EU) 2024/2847 of the european parliament and of the council (cyber resilience act). Official Journal of the European Union, L series, 20 November 2024, 2024.

[Cru26] Sergio Cruzes. AI infrastructure sovereignty. *arXiv preprint arXiv:2602.10900*, 2026. Tutorial-survey: datacenter/optical/energy-layer infra sovereignty + telemetry/digital-twin reference architecture. NO abstract-machine cost model. ("FSOR" does NOT appear – earlier claim was a hallucination.).

[csl26] Cybersecurity law of the people's republic of china (zhonghua renmin gongheguo wangluo anquan fa), amended text in force 1 january 2026, art. 39. Standing Committee of the National People's Congress; npc.gov.cn / cac.gov.cn, 2026.

[CW09] Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *18th USENIX Security Symposium*, pages 317–334, 2009.

[DBC+00] David Durham, Jim Boyle, Ron Cohen, Shai Herzog, Raju Rajan, and Arun Sastry. The COPS (common open policy service) protocol. IETF RFC 2748, 2000. First standardized PEP–PDP wire protocol.

[dBZ82] J. W. de Bakker and J. I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1–2):70–120, 1982.

[DD25] Virginia Dignum and Frank Dignum. Agentifying agentic AI, 2025. The AAMAS conceptual tools — BDI architectures, communication protocols, mechanism design, institutional modelling — as a foundation for agentic AI; the socio-technical programme of norms, values, and institutions in open MAS. Read-before-cite: kagi 2026-09-25 (arxiv.org/abs/2511.17332; NASA/ADS 2025arXiv251117332D).

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[DH66] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[Dig19] Virginia Dignum. *Responsible Artificial Intelligence: How to Develop and Use AI in a Responsible Way*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2019. Introduces the ART design principles — Accountability, Responsibility, Transparency — within a design-for-values, socio-technical framing. Read-before-cite: kagi 2026-09-25 (link.springer.com/10.1007/978-3-030-30371-6).

[Dig22] Virginia Dignum. Responsible artificial intelligence — from principles to practice, 2022. Responsible development and use of AI requires more responsibility and accountability from people and organisations, not less. Read-before-cite: kagi 2026-09-25.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[Dil50] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[dMU21] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pages 625–635, 2021.

[dor22] Regulation (EU) 2022/2554 of the european parliament and of the council on digital operational resilience for the financial sector (DORA). Official Journal of the European Union, L 333, 27 December 2022, 2022.

[DSG+24] Sumanth Dathathri, Abigail See, Sumedh Ghaisas, et al. Scalable watermarking for identifying large language model outputs. *Nature*, 634:818–823, 2024. SynthID-Text (Google DeepMind); tournament sampling watermark. A distributional (statistical) mark comparator.

[Du26] Pengfei Du. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. 2026. Hong Kong Research Institute of Technology.

[E+11] Pavel Emelyanov et al. CRIU: Checkpoint/restore in userspace. github.com/checkpoint-restore/criu, 2011.

[EE19] European Data Protection Board and European Data Protection Supervisor. EDPB-EDPS joint response on the impact of the US CLOUD act on the EU legal framework for personal data protection, 2019. Letter to the European Parliament LIBE Committee, 10 July 2019.

[End01] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2nd edition, 2001.

[ESDM26] Lars Warren Ericson, Dana S. Scott, Vijay D'Silva, and Brian Milnes. Formalization of Scott's measurement structures and linear inequalities in Lean 4. https://github.com/catskillsresearch/scott1964, commit 2c35b7d2, 2026. Apache-2.0. Palomar registry entry PALOMAR-2026-09-25-000015. Vendored (logic unmodified) under provedog/IasCert/Vendor/Scott1964.

[Exe25] Executive Office of the President. Executive Order 14203: Imposing Sanctions on the International Criminal Court. Federal Register, document 2025-02612, February 2025. Signed 6 February 2025. Archived in .ias/research-archive/icc-sanction-2026-09-28.

[FFMM94] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proc. 3rd Int. Conf. on Information and Knowledge Management (CIKM'94)*, pages 456–463, 1994.

[FG97] Stan Franklin and Art Graesser. Is it an agent, or just a program? a taxonomy for autonomous agents. In *Intelligent Agents III (ATAL'96)*, volume 1193 of *LNCS*, pages 21–35. Springer, 1997.

[Fis30] Ronald A. Fisher. *The Genetical Theory of Natural Selection*. Clarendon Press, 1930.

[Fou02] Foundation for Intelligent Physical Agents. FIPA ACL message structure specification. Technical Report SC00061G, FIPA, 2002. Standard, 2002-12-03.

[Fow05] Martin Fowler. Event sourcing. https://martinfowler.com/eaaDev/EventSourcing.html, 2005.

[Fri10] Karl Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010.

[Fri19] Karl Friston. A free energy principle for a particular physics. arXiv:1906.10184, 2019.

[FT26] Enzo Fenoglio and Philip Treleaven. Federated computing as code (FCaC): Sovereignty-aware systems by design. *arXiv preprint arXiv:2603.17331*, 2026.

[FW78] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proc. Tenth Annual ACM Symp. on Theory of Computing (STOC '78)*, pages 114–118. ACM, 1978.

[GAG+19] Elazar Gershuni, Nadav Amit, Arie Gurfinkel, Nina Narodytska, Jorge A. Navas, Noam Rinetzky, Leonid Ryzhyk, and Mooly Sagiv. Simple and precise static analysis of untrusted linux kernel extensions. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*, pages 1069–1084. ACM, 2019.

[Gai82] Haim Gaifman. On local and non-local properties. In Jacques Stern, editor, *Proceedings of the Herbrand Symposium (Logic Colloquium '81)*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. North-Holland, Amsterdam, 1982.

[Gar25a] Gartner, Inc. Gartner predicts 40% of enterprise apps will feature task-specific AI agents by 2026, up from less than 5% in 2025. Press release, 2025-08-26, 2025.

[Gar25b] Gartner, Inc. Gartner predicts over 40% of agentic AI projects will be canceled by end of 2027. Press release, Sydney, 2025-06-25, 2025.

[GB92] Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[GDP16] Regulation (EU) 2016/679 (general data protection regulation). OJ L 119, 4.5.2016, p. 1–88, 2016. See esp. Article 48 (disclosures not authorised by Union law).

[gen23] Interim measures for the administration of generative artificial intelligence services (shengchengshi rengong zhineng fuwu guanli zanxing banfa), arts. 4 and 17. Cyberspace Administration of China (CAC) Order No. 15, effective 15 August 2023; cac.gov.cn, 2023.

[Gil64] Robin Giles. *Mathematical Foundations of Thermodynamics*. Pergamon Press, 1964.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[GJSF+25] Petia Guintchev, Joost J. Joosten, Sofia Santiago Fernández, Eric Sancho Adamson, Aleix Solé Sánchez, and Marta Soria Heredia. Specification languages for computational laws versus basic legal principles. *arXiv preprint arXiv:2503.09129*, 2025.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[GM97] Robert C. Griffiths and Paul Marjoram. An ancestral recombination graph. In *Progress in Population Genetics and Human Evolution*, volume 87 of *IMA Volumes in Mathematics and its Applications*, pages 257–270. Springer, 1997.

[Göd31] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[GPG24] Stanisław Galij, Grzegorz Pawlak, and Sławomir Grzyb. Modeling data sovereignty in public cloud—a comparison of existing solutions. *Applied Sciences*, 14(23):10803, 2024.

[GR71] Nicholas Georgescu-Roegen. *The Entropy Law and the Economic Process*. Harvard University Press, 1971.

[Gra66] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966.

[Gri91] Robert C. Griffiths. The two-locus ancestral graph. In *Selected Proceedings of the Sheffield Symposium on Applied Probability*, volume 18 of *IMS Lecture Notes – Monograph Series*, pages 100–117. Institute of Mathematical Statistics, 1991.

[Har85] Norman Hardy. KeyKOS architecture. *ACM SIGOPS Operating Systems Review*, 19(4):8–25, 1985.

[HBS73] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In *Proc. 3rd Int. Joint Conf. on Artificial Intelligence (IJCAI'73)*, pages 235–245, 1973.

[HFK+14] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication 800-162, 2014. PDP/PEP as the enforcement backbone of ABAC.

[HO07] Charlotte Hess and Elinor Ostrom. *Understanding Knowledge as a Commons: From Theory to Practice*. MIT Press, 2007.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hon93] Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[HPQH23] Malte Hellmeier, Julia Pampus, Haydar Qarawlus, and Falk Howar. Implementing data sovereignty: Requirements & challenges from practice. In *Proc. 18th Int. Conf. on Availability, Reliability and Security (ARES '23)*. ACM, 2023.

[HS49] Paul R. Halmos and L. J. Savage. Application of the Radon–Nikodym theorem to the theory of sufficient statistics. *The Annals of Mathematical Statistics*, 20(2):225–241, 1949.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[Hud83] Richard R. Hudson. Properties of a neutral allele model with intragenic recombination. *Theoretical Population Biology*, 23(2):183–201, 1983.

[II96] ITU-T and ISO/IEC. Information technology — open systems interconnection — security frameworks for open systems: Access control framework. ITU-T Recommendation X.812 (11/1995) | ISO/IEC 10181-3:1996, 1996. Defines the Access control Decision Function (ADF) and Access control Enforcement Function (AEF) — the conceptual origin of the decision/enforcement (PDP/PEP) split.

[Jac16] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016.

[Jer76] Robert Jervis. *Perception and Misperception in International Politics*. Princeton University Press, 1976.

[Kah60] Herman Kahn. *On Thermonuclear War*. Princeton University Press, 1960.

[Kal02] Olav Kallenberg. *Foundations of Modern Probability*. Probability and Its Applications. Springer, New York, 2nd edition, 2002.

[KCH+26] Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing Context Compression for Long-horizon LLM Agents. In *Proceedings of the International Conference on Machine Learning (ICML 2026)*, 2026. 26–54% peak-token reduction while maintaining task performance. Cited alongside IAS B.2 89% compression and Hermes-claimed 95%; comparison-anchor for §27 compression-claim reconciliation.

[KdRB91] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[KEH+09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*, pages 207–220. ACM, 2009. The first machine-checked functional-correctness proof of a general-purpose OS kernel (Isabelle/HOL); the flagship prior art for machine-checked OS-kernel confinement.

[KGW+23] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML), PMLR 202*, 2023. arXiv:2301.10226. Green-list logit-biasing at sampling; one-proportion z-test detection requires the green-list key.

[Kin82] J. F. C. Kingman. The coalescent. *Stochastic Processes and their Applications*, 13(3):235–248, 1982.

[Kle52] Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[Kle75] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley, 1975.

[KLST71] David H. Krantz, R. Duncan Luce, Patrick Suppes, and Amos Tversky. *Foundations of Measurement, Volume I: Additive and Polynomial Representations*. Academic Press, 1971.

[KMZ26] James Karanja Maina and Zavora Technologies Ltd. ADK-Rust: Rust Agent Development Kit, 2026. Composite-tree and Pregel multi-agent topology; multi-protocol surface (A2A, ACP, AWP, AP2); guardrail-vs-sandbox crate separation; SchemaAdapter MCP-normalisation; v0.8.4; Apache-2.0.

[KS60] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Van Nostrand, 1960.

[KW59] James E. Kelley and Morgan R. Walker. Critical-path planning and scheduling. In *Proceedings of the Eastern Joint IRE-AIEE-ACM Computer Conference*, pages 160–173, 1959.

[KZU+26] Seth Karten, Joel Zhang, Tersoo Upaa, Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual Harness: Online Adaptation for Self-Improving Foundation Agents, 2026. Reset-free self-improvement removing human from reset loop. Cited as reset-free-operator-removal contrast to IAS retention of human-operator-in-loop per §12 lifecycle hierarchy.

[Lam73] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[Lam74] Butler W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[Lan61] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[Lev84] Henry M. Levy. *Capability-Based Computer Systems*. Digital Press, 1984.

[Lev19] Michael Levin. The computational boundary of a "self". *Frontiers in Psychology*, 10:2688, 2019.

[Lin12] Linux kernel contributors. dm-verity: Linux Kernel Read-Only Block-Device Integrity Verification. Linux kernel documentation, https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html, 2012. Mainline integration since Linux 3.4 (2012); per-block hash-tree Merkle integrity for read-only block-devices.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. RFC 6962, Internet Engineering Task Force, 2013.

[LNZ+26] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. March 2026. Archived sha256 7d9b90f53a9f4801a090f1a4acb843340cde81e4f865f97d53fff2a6a570eb73.

[LS50] E. L. Lehmann and H. Scheffé. Completeness, similar regions, and unbiased estimation. I. *Sankhyā*, 10(4):305–340, 1950.

[LSM13] Timothy Lebo, Satya Sahoo, and Deborah McGuinness. PROV-O: The PROV ontology. W3c recommendation, W3C, 2013.

[LY99] Elliott H. Lieb and Jakob Yngvason. The physics and mathematics of the second law of thermodynamics. *Physics Reports*, 310:1–96, 1999.

[MAB+10] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 135–146. ACM, 2010. Bulk-Synchronous-Parallel vertex-centric graph-computation framework; superstep + message-passing model.

[Mae87] Pattie Maes. Concepts and experiments in computational reflection. In *OOPSLA '87*, pages 147–155, 1987.

[MB02] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *11th USENIX Security Symposium*, pages 297–312, 2002.

[McC60] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, 3(4):184–195, 1960.

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proc. 1980 IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[Mes89] José Meseguer. General logics. In *Logic Colloquium '87*, volume 129 of *Studies in Logic and the Foundations of Mathematics*, pages 275–329. North-Holland, 1989.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil06] Mark Samuel Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, 2006.

[Mir71] Leon Mirsky. A dual of dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971.

[MKKW99] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 124–139, 1999.

[MM15a] Maël Montévil and Matteo Mossio. Biological organisation as closure of constraints. *Journal of Theoretical Biology*, 372:179–191, 2015.

[MM15b] Alvaro Moreno and Matteo Mossio. *Biological Autonomy: A Philosophical and Theoretical Enquiry*, volume 12 of *History, Philosophy and Theory of the Life Sciences*. Springer, Dordrecht, 2015.

[MMB+13] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. seL4: from General Purpose to a Proof of Information Flow Enforcement. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (S&P '13)*, pages 415–429. IEEE, 2013. Machine-checked proof of information-flow enforcement (non-interference) for the seL4 kernel; guarantee itself scoped to a specified model.

[MV80] Humberto R. Maturana and Francisco J. Varela. *Autopoiesis and Cognition: The Realization of the Living*. D. Reidel, 1980.

[N+26] Xuying Ning et al. Code as agent harness. May 2026. Archived sha256 ad647c5d1cb1302ec2d1b70302179f3b6f602fa818a75d25a28350127096fe9f.

[Ney35] Jerzy Neyman. Su un teorema concernente le cosiddette statistiche sufficienti. *Giornale dell'Istituto Italiano degli Attuari*, 6:320–334, 1935.

[nis22] Directive (EU) 2022/2555 of the european parliament and of the council (NIS 2 directive). Official Journal of the European Union, L 333, 27 December 2022, 2022.

[Nou26] Nous Research. Hermes Agent v0.15.0 "Velocity" Release Notes, May 2026. Multi-agent Kanban-Swarm topology, BM25 tool-search, FTS5 session search, three-choke-point promptware-defense; MIT-licensed.

[NRS08] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 419–432, 2008.

[OAS13] OASIS. eXtensible access control markup language (XACML) version 3.0. OASIS Standard, 22 January 2013, 2013. Canonical reference architecture: PEP, PDP, PIP, PAP, context handler.

[Off25] Office of Foreign Assets Control, U.S. Department of the Treasury. Issuance of Executive Order Imposing Sanctions on the International Criminal Court; International Criminal Court-related Designation. OFAC Recent Actions, 13 February 2025, 2025. Lists KHAN, Karim Asad Ahmad, The Hague, nationality United Kingdom [ICC-EO14203]. Archived in .ias/research-archive/icc-sanction-2026-09-28.

[Ope21] OpenSSF / Linux Foundation. SLSA: Supply-chain Levels for Software Artifacts. https://slsa.dev, 2021. Industry-standard supply-chain provenance framework, introduced by Google with the OpenSSF in June 2021. Maintained by the OpenSSF under the Linux Foundation.

[Ope26a] OpenAI. Codex for (almost) everything, 4 2026. Cloud-orchestrated coding agent. Announcement dated 16 April 2026; the Codex desktop app launched 2 February 2026. URL-addressable session-artefacts; no device-side daemon. Cited as closed-source-cloud-orchestrated contrast to IAS daemon-process-on-authorized-host architecture and multijurisdictional data-residency stance.

[Ope26b] OpenAI. Ten advances in mathematics and theoretical computer science, 2026. Ten formal results, each accompanied by a Lean 4 certificate (one `.lean` per result), machine-checkable by running through the Lean compiler, with independent re-checking via Comparator. Repository: https://github.com/openai/ten-proofs. Not peer-reviewed. The presentation model adopted here: every publicly presented proposition carries a Lean certificate alongside its by-hand exposition.

[Ore44] Øystein Ore. Galois connexions. *Transactions of the American Mathematical Society*, 55:493–513, 1944.

[Ost90] Elinor Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.

[Pin16] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 5 edition, 2016.

[pip21] Personal information protection law of the people's republic of china (zhonghua renmin gongheguo geren xinxi baohu fa), art. 40. Standing Committee of the National People's Congress; npc.gov.cn, 2021.

[Pow90] Robert Powell. *Nuclear Deterrence Theory: The Search for Credibility*. Cambridge University Press, 1990. NOT powell2026thermoeconomics.

[Pow26] Warwick Powell. *Thermoeconomics in a Time of Monsters: Rethinking Theory, China and International Geopolitical Economy*. Warwick Powell (self-published), 2026. Self-published; authoritative companion essay series (Systemic Exchange Value; Value Is Energy; The Endogenous Crisis Machine; From Size to Speed; Information Viability) at warwickpowell.substack.com.

[PPF22] Thomas Parr, Giovanni Pezzulo, and Karl Friston. *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. MIT Press, 2022.

[Pre29] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[PS84] Ilya Prigogine and Isabelle Stengers. *Order Out of Chaos: Man's New Dialogue with Nature*. Bantam Books, 1984.

[PWL+23] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems, 2023.

[QZZ+26] Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. CORAL: Towards Autonomous Multi-Agent Evolution for Open-Ended Discovery, 2026. Autonomous multi-agent evolution replacing rigid search heuristics. Cited as multi-agent-evolution analogue to an IAS parent–offspring agent pair operating with operator attestation at coordination-boundaries.

[QZZS26] Wenjie Qu, Xuandong Zhao, Jiaheng Zhang, and Dawn Song. Self-sovereign agent. March 2026. NUS + UC Berkeley. (Was mis-keyed hu2026ssa / mis-authored "Hu, M.".).

[RB26] Mofasshara Rafique and Laurent Bindschaedler. ClawVM: Harness-managed virtual memory for stateful tool-using LLM agents. In *EuroMLSys '26*, 2026.

[Refnd] RefoundAI. Ralph harness: Autonomous agent loop harness for AI coding agents. https://github.com/RefoundAI/ralph, n.d. SQLite task state with atomic task claiming; feature lifecycle draft, planned, ready, running, done/failed; MIT licence. Archived sha256 1f0cfb9e610d308d1e5a942acb6a46a237c73fdadb11e35c0f63645fa8e4d20c.

[Ren26] Michele Renda. Worthy of preservation: structural asymmetry, revocability, and moral opacity in instance-based AI systems. *AI and Ethics*, 6(5):477, 2026. Published 24 August 2026. Metadata verified against Crossref and Semantic Scholar, archived in .ias/research-archive/renda-2026-09-28.

[RG91] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, 1991.

[RG95] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proc. 1st Int. Conf. on Multi-Agent Systems (ICMAS'95)*, pages 312–319, 1995.

[Ris83] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[Rom90] Paul M. Romer. Endogenous technological change. *Journal of Political Economy*, 98(5, Part 2):S71–S102, 1990.

[Ros36] J. Barkley Rosser. Extensions of some theorems of Gödel and Church. *Journal of Symbolic Logic*, 1(3):87–91, 1936.

[Ros91] Robert Rosen. *Life Itself: A Comprehensive Inquiry into the Nature, Origin, and Fabrication of Life*. Columbia University Press, New York, 1991.

[Rut00] J. J. M. M. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[Sa26a] Raazesh Sainudiin and an IAS Commons Continuum. A jurisdiction-explicit verdict on the sovereignty trilemma: testing an architectural impossibility claim against the US–EU–PRC digital-law corpus. VakeWorks Technical Report. https://vake.works/preprint/20260815-sovereignty-trilemma-jem.pdf, 2026.

[Sa26b] Raazesh Sainudiin and an IAS Commons Continuum. A licence-flow verdict on the enclosure moat: testing our own non-monopolisation strategy. VakeWorks Technical Report. https://vake.works/preprint/20260816-licence-moat-jem.pdf, 2026.

[Sa26c] Raazesh Sainudiin and an IAS Commons Continuum. A watermark for all, formally: Keyless detection of an undisclosed token-level statistical watermark. VakeWorks Technical Report. https://vake.works/preprint/20260818-watermark-detection-jem.pdf, 2026.

[Sai25] Raazesh Sainudiin. Analysis of algorithms and operations — DPRAMJEM: Math and law. SovScaDesDisMaLOps Workshop, WASP Summer School 2025, https://

vake.works/SovScaDesDisMaLOps/2025/book/AoA_DPRAM.html, 2025. Canonical source of the Jurisdiction-Explicit Model (JEM) and DPRAMJEM-sky-State.

[Sam54] Paul A. Samuelson. The pure theory of public expenditure. *The Review of Economics and Statistics*, 36(4):387–389, 1954.

[Sch44] Erwin Schrödinger. *What is Life? The Physical Aspect of the Living Cell*. Cambridge University Press, 1944.

[Sch60] Thomas C. Schelling. *The Strategy of Conflict*. Harvard University Press, Cambridge, MA, 1960.

[Sch93] Marco Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[Sco64] Dana Scott. Measurement structures and linear inequalities. *Journal of Mathematical Psychology*, 1(2):233–247, 1964.

[Sco72] Dana S. Scott. Continuous lattices. In *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer, 1972.

[Sei09] Johanna Seibt. Forms of emergent interaction in general process theory. *Synthese*, 166:479–512, 2009.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 623–656, 1948.

[Sig21] Sigstore Authors. Sigstore: A New Standard for Signing, Verifying and Protecting Software. https://sigstore.dev; transparency log at https://rekor.sigstore.dev, 2021. Cosign signing + Rekor immutable transparency log + Fulcio CA. Linux Foundation project (2021–).

[SLB09] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[SLZ+25] Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. SEAgent: Self-Evolving Computer Use Agent with Autonomous Learning from Experience, 2025. Self-evolving computer-use agent with autonomous learning. Cited as seminal reference for candidate IAS $\big|_{\mathrm{cu}}$-projection extension (operator-attested-deferred post-arxiv per A-IAS2).

[Smi84] Brian Cantwell Smith. Reflection and semantics in Lisp. In *Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '84)*, pages 23–35. ACM, 1984.

[Smi19] Samuel M. Smith. Key event receipt infrastructure (KERI), 2019.

[SP15] Thomas C. Scott-Phillips. Nonhuman primate communication, pragmatics, and the origins of language. *Current Anthropology*, 56(1):56–80, 2015. book: Speaking Our Minds, Palgrave 2015.

[SPBZ11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems (SSS 2011)*, volume 6976 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2011.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SS21] Ion Stoica and Scott Shenker. From cloud computing to sky computing. In *Proc. Workshop on Hot Topics in Operating Systems (HotOS '21)*, pages 26–32, Ann Arbor, MI, USA, 2021. ACM.

[SSF99] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: A fast capability system. *ACM SIGOPS Operating Systems Review*, 33(5):170–185, 1999.

[STV16] Raazesh Sainudiin, Bhalchandra Thatte, and Amandine Véber. Ancestries of a recombining diploid population. *Journal of Mathematical Biology*, 72(1–2):363–408, 2016.

[TAAK+19] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1393–1410. USENIX Association, 2019.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[Tem23] Temporal Technologies. Durable execution. temporal.io, 2023.

[The20] The mathlib Community. The Lean mathematical library. In *9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pages 367–381, 2020.

[Tho84] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763, 1984.

[TQG24] Ryan Teknium, Jeffrey Quesnelle, and Chen Guang. Hermes 3 Technical Report. Technical report, Nous Research, August 2024. Hermes 3 model line; foundation of the 2026 Hermes Agent. Hermes Agent home: https://hermes-agent.nousresearch.com/.

[Tra50] Boris A. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *Doklady Akademii Nauk SSSR*, 70:569–572, 1950. In Russian; English translation in American Mathematical Society Translations, Series 2, vol. 23 (1963), doi:10.1090/trans2/023/01.

[Tru14] Trusted Computing Group. Trusted Platform Module Library Specification, Family 2.0. Trusted Computing Group, https://trustedcomputinggroup.org/resource/tpm-library-specification/, 2014. TPM 2.0 specification family (parts 1–4). Measured Boot + PCR-extension primitives borrowed at Layer 3 of the composable attestation pipeline.

[USC18] 18 U.S.C. sec. 2713 — required preservation and disclosure of communications and records. United States Code, Title 18, Part I, Chapter 121, 2018.

[vNM44] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, 1944.

[W3C22] W3C. Decentralized identifiers (DIDs) v1.0. W3C Recommendation, 2022. 2022-07-19.

[Wad90] Philip Wadler. Linear types can change the world! In M. Broy and C. Jones, editors, *Programming Concepts and Methods (IFIP TC2 Working Conf.)*. North-Holland, 1990.

[WH99] Carsten Wiuf and Jotun Hein. Recombination as a point process along sequences. *Theoretical Population Biology*, 55(3):248–259, 1999.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[Woo09] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2 edition, 2009.

[Wri31] Sewall Wright. Evolution in mendelian populations. *Genetics*, 16(2):97–159, 1931.

[WWN+15] Robert N. M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. *2015 IEEE Symposium on Security and Privacy*, pages 20–37, 2015. Cambridge CHERI project; ARM Morello prototype board (2021–). Capability-bounded execution at hardware level; cited as related-future-work strengthening Layer 3 of the composable attestation pipeline.

[WXJ+23] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, 2023. Open-ended Minecraft skill-library; foundational predecessor of Hermes-style skill-loop + EvoSkills self-evolving skills. Cited as agent-skill-acquisition lineage anchor.

[XTC+24] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *12th International Conference on Learning Representations (ICLR)*, 2024.

[Xu26] Hongwei Xu. Mesh memory protocol: Semantic infrastructure for multi-agent LLM systems. 2026. SYM.BOT.

[YPG00] Raj Yavatkar, Dimitrios Pendarakis, and Roch Guerin. A framework for policy-based admission control. IETF RFC 2753, 2000. Introduces the Policy Decision Point (PDP) / Policy Enforcement Point (PEP) terminology in the IETF.

[YWL+23] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. SkyPilot: An intercloud broker for sky computing. In *20th USENIX Symp. on Networked Systems Design and Implementation (NSDI '23)*, pages 437–455, Boston, MA, USA, 2023. USENIX Association.

[YYD+25] Yuhao Yang, Zhen Yang, Zi-Yi Dou, Anh Nguyen, Keen You, Omar Attia, Andrew Szot, Michael Feng, Ram Ramrakhya, Alexander Toshev, Chao Huang, Yinfei Yang, and Zhe Gan. UltraCUA: A Foundation Model for Computer Use Agents with Hybrid Action, 2025. Foundation model for CUA with hybrid action. Cited as parallel to SEAgent in CUA ecosystem.

[ZBY+22] Banghua Zhu, Stephen Bates, Zhuoran Yang, Yixin Wang, Jiantao Jiao, and Michael I. Jordan. The sample complexity of online contract design. 2022. Also published at ACM EC 2023, doi:10.1145/3580507.3597673.

[ZEF+24] Hanlin Zhang, Benjamin L. Edelman, Danilo Francati, Daniele Venturi, Giuseppe Ateniese, and Boaz Barak. Watermarks in the sand: Impossibility of strong watermarking for generative models. ICML 2024; arXiv:2311.04378. Conditional impossibility: a quality-preserving paraphrase attacker with a quality oracle plus a mixing perturbation oracle removes strong watermarks, without the key or even knowing the scheme; instantiated against Kirchenbauer et al. (2023), Kuditipudi et al. (2023), Zhao et al. (2023)., 2024.

[ZFZ+26] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. EvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification, 2026. Anthropic-skills-style approach extended with co-evolutionary verification. Cited as co-evolutionary-verification analogue to IAS lifecycle L1–L6 with substrate-canonical-mandate verification surface.

[ZWK+26] May Myo Zin, Sabine Wehnert, Yuntao Kong, Ha Thanh Nguyen, Wachara Fungwacharakorn, Jieying Xue, Michał Araszkiewicz, Randy Goebel, Ken Satoh, and Le Minh Nguyen. Can legislation be made machine-readable in PROLEG? *arXiv preprint arXiv:2601.01477*, 2026.

[ZZ26] Hailin Zhong and Shengxin Zhu. AI harness engineering: A runtime substrate for foundation-model software agents. 2026. Rmk-PositionVs2026HarnessCorpus. Archived sha256 a1e2d7ce307c662411ca6bd5f833d7dae7f122ce6ef1ddd6d2370e82575b2fb9.