

Beyond Memory: A Transactional Continuity Kernel for Long-Lived AI Agents

Jun He
OpenKedge.io

Deying Yu
OpenKedge.io

Abstract

Persistent AI agents accumulate versioned state across long horizons, but storage retention alone does not identify authoritative state. Without an explicit control plane, unmediated updates by models, tools, and background workers risk stale overwrites, un-audited exposures, and self-authorizing privilege escalation. We argue that agent state governance is an infrastructural activation problem, defining *continuity* as an unbroken, authorized lineage of accepted branch heads. We present the *Continuity Kernel* (CK), an activation contract that decouples off-commit candidate evaluation from atomic state activation. Untrusted components propose typed changes against an exact predecessor head or typed absence. A short activation transaction revalidates ownership, pre-state authority, freshness, and effect uniqueness, recording one stable disposition (Commit, Reject, Quarantine, or Defer). Only Commit atomically advances the branch head and installs the complete accepted unit (state, authority, lineage, effects, outcome, and receipt). A bounded executable model verifies the protocol across 2,808,230 reachable states and 5,526,474 state-changing transitions with zero invariant violations.

1 Introduction

Long-lived AI agents increasingly retain memories, profiles, plans, tool state, and policies beyond a single context window [1, 2]. However, high retrieval quality does not determine which state version is authoritative when models, tools, compactors, and background recovery workers submit concurrent or conflicting updates. Without a strict mutation boundary, a storage layer may preserve every version yet accept a stale retry, expose uncommitted state without audit trails, or allow a proposal to authorize itself by injecting the required privileges into its own proposed state.

We define *continuity* as an unbroken, authorized lineage of accepted heads on one branch. This is infrastructural continuity—a guarantee about state activation and lineal provenance—not a philosophical claim about agent consciousness or behavioral identity. Continuity fundamentally differs

from retention: rejected and quarantined objects may remain stored for diagnostic or policy reasons, but they remain unreachable from the authoritative branch head. Cryptographic commitments can enforce this reachability boundary; they cannot make an inferred memory factually true or a policy normatively legitimate.

The core thesis of this paper is that agent state governance is an infrastructural activation contract problem. To operationalize this principle, we define the *Continuity Kernel* (CK) as an activation contract that separates off-commit candidate *proposal* from atomic *activation*. Models, tools, and operators act as untrusted proposers. They may perform probabilistic reasoning, remote network fetches, and candidate state construction, but only the deterministic CK control plane may advance an authoritative branch head. Each proposal targets an exact predecessor head or typed target absence. Slow evaluation and evidence acquisition precede activation; a short transaction then revalidates all mutable facts governing admission and records exactly one durable disposition. Figure 1 illustrates this activation boundary.

CK introduces no new low-level transaction engine. Instead, standard substrates—such as relational database transactions, conditional object manifests, or consensus state-machine logs—execute its atomic step. The primary contribution is the agent-state activation contract enforced within that step: it atomically binds proposal identity, exact predecessor state, pre-state authority, acquired evidence, lifecycle status, effect manifests, lineage, and outcome receipts. Specifically, this paper makes three contributions:

1. **System Model and Contract:** A formal model separating stored retention from authoritative reachability, defining the boundary between untrusted proposers and trusted activation (Section 2);
2. **Activation Protocol:** An owner-bound, exact-head activation protocol with pre-state authorization, four terminal dispositions (Commit, Reject, Quarantine, Defer), at-most-once effect execution, and explicit receipt evidence levels (Section 3); and

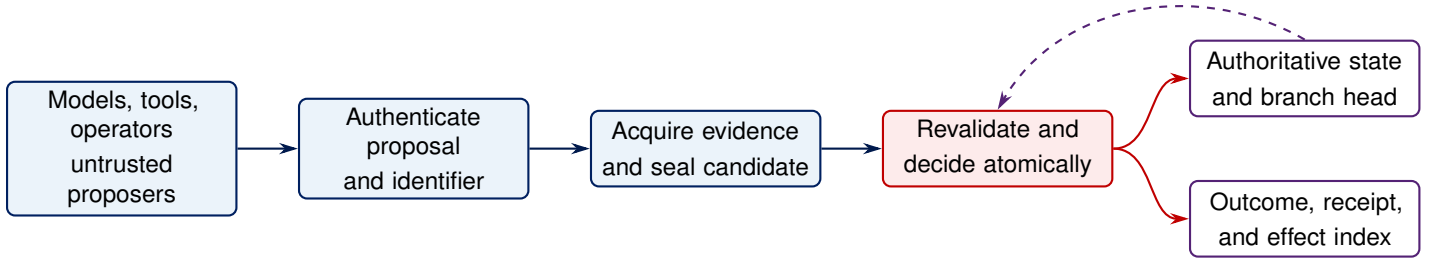


Figure 1: Probabilistic proposal generation and remote evaluation precede the short activation transaction. Only activation advances an authoritative head.

Table 1: Storage does not by itself confer authority.

Namespace	Purpose	In head?
Accepted	Current typed state	Yes
Candidate	Prepared successor	No
Attempt	Outcome and diagnostics	No
Quarantine	Isolated candidate	No
Projection	Read-only runtime view	No

3. **Lifecycle Rules and Model Verification:** Typed life-cycle semantics for branch creation, writer handoff, schema migration, and forward restoration, verified through bounded state-space exploration (Sections 4–5).

Sections 2–5 develop the core architecture and evaluation; Appendices A–D detail normative specifications, lineage proofs, and model parameters.

2 System Model and Contract

2.1 Authority Is Reachability

As illustrated in Figure 1, CK establishes a strict boundary between candidate object preparation and authoritative state activation. Agent state is partitioned into isolated branches identified by a subject/system identifier sid and a branch identifier bid , denoted $k = (sid, bid)$. The branch-indexed state $X[k]$ holds the complete collection of typed agent components for branch k , including memory records, user profiles, tool configurations, and policy-bearing objects.

As summarized in Table 1, the storage layer maintains multiple functional namespaces. Candidate proposals, execution attempt logs, quarantined candidates, and read-only runtime projections coexist in storage alongside accepted state. However, storage presence alone does not confer authority: an object is authoritative if and only if it is reachable from the current branch head committed by CK.

For branch $k = (sid, bid)$, the branch directory maintains

the complete-head type h and branch-row type $B[k]$:

$$\begin{aligned}
 h &= \langle sid, bid, seq, root, v, parentRef, lineageRef \rangle, \\
 B[k] &= \langle h, status, writer, epoch, dirSeq, handoff, createdFrom \rangle.
 \end{aligned} \tag{1}$$

Here $root$ commits the typed component state; $v = (schemaV, policyV, evalV, authV)$ names the pinned versions; $parentRef$ is the parent head reference; $lineageRef$ binds the causal lineage; $dirSeq$ is directory sequence; $handoff$ is the open revision pointer; and $createdFrom$ records genesis provenance. Heads are equal only when every canonically encoded field agrees.

The abstract kernel configuration is

$$C = \langle X, B, \Gamma, \mathcal{S}, t \rangle. \tag{2}$$

Γ is the pre-state authority context; $\mathcal{S}$ represents persistent metadata stores (outcomes, receipts, lineage, effects, and quarantine); and t is trusted transaction time. An activation serializes the branch row and every admission, allocation, and effect row that its proposal names.

2.2 Proposals and Typed Transitions

Models, tools, memory services, and operators are untrusted proposers. Evaluators and approvers issue signed evidence but cannot advance a head. A trusted gateway authenticates proposal ownership; a preparation service acquires the required evidence and derives a candidate; the activation engine alone changes authority.

A signed proposal is summarized by

$$\tau = \langle pid, target, expected, ops, a, req \rangle. \tag{3}$$

Here pid is the unique proposal identifier; $target$ is the target branch key $k = (sid, bid)$; $expected$ is the expected predecessor head; ops is an ordered sequence of state operations; a is a sequence of authority changes; and req declares the evidence required for admission.

Candidate derivation occurs off-commit, producing candidate state $X' = F(X, ops, W)$ using evidence W . Crucially,

authorization evaluates against the pre-state context A :

$$X' = F(X, ops, W), \quad A = \begin{cases} \Gamma, & \text{existing branch,} \\ \xi_k, & \text{branch creation.} \end{cases} \quad (4)$$

$\text{Authorize}_A(\tau, W)$ runs before computing the authority update $\Gamma' = F_A(A, a)$; a proposal cannot authorize itself. The candidate seal binds the proposal, evidence, interpreter, and component roots. Activation recomputes these bindings against the serialized branch state.

Schema evolution is an explicit Migration transition with a pinned, deterministic migrator. Deletion creates a typed tombstone; physical erasure remains subject to retention policy. Hashes establish cryptographic integrity, not confidentiality, semantic truth, or policy legitimacy.

2.3 Threat Model and Conditional Contract

Proposers may be buggy or adversarial: they may replay requests, reuse identifiers, reorder operations, bind a stale head, omit evidence, forge scope, or request self-authorizing changes. Infrastructure may crash, lose replies, duplicate messages, expose stale replicas, or partition. Evaluators may be wrong; their outputs are policy inputs rather than semantic oracles.

The safety claims depend on nine explicit assumptions (A1–A9, stated individually in Appendix D): (1) **Boundary & Cryptography (A1–A3)**: all authoritative writes cross an authenticated kernel boundary, typed cryptography is sound, and signing keys are protected; (2) **Atomic Serialization (A4)**: the storage substrate atomically serializes the complete activation key set; (3) **Complete Context & Time (A5–A6)**: policy, authority, revocation, lifecycle, dependency, and trusted-time values are available for commit-time validation; (4) **Lifecycle Order & Non-Recycled Scopes (A7–A8)**: lifecycle operations share one order and proposal/effect scopes and writer epochs are not recycled; and (5) **Verification Objects (A9)**: stronger receipt claims are made only when their required verification objects are available.

Under those assumptions, the contract has four consequences:

1. **Exact succession.** One complete predecessor has at most one accepted successor; one serialized absence has at most one genesis. Each is installed with state, lineage, effects, outcome, and receipt.
2. **Pre-state admission.** Authorization and freshness are checked against the serialized predecessor or creation context, never the proposed context.
3. **Stable execution identity.** A proposal identifier has one terminal disposition and an effect identifier has at most one accepted binding, including after safe reclamation.

4. **Lifecycle isolation.** Branch creation, fencing, handoff, migration, and restoration preserve branch scope and current authority.

These are safety properties, not availability guarantees. The kernel may be unavailable or may return *Defer* when evidence cannot be obtained. It governs only state behind its mutation boundary; prompts, caches, model weights, tool-local data, prior disclosures, and remote side effects remain outside unless separately mediated.

3 Activation Protocol

The protocol separates slow, fallible preparation from one short serialized decision. Preparation authenticates the proposer, acquires exactly the evidence declared by the proposal, runs the pinned transition function, and seals the candidate. Activation admits that sealed package only if the relevant state is still current. Appendix A fixes the complete stage order and receipt variants; this section states the activation rule.

3.1 One activation predicate

Let $P = \langle \tau, W, X', M_E, s \rangle$ be a prepared package containing the signed proposal, exact evidence set, candidate state, ordered effect manifest, and trusted candidate seal. Inside the activation transaction, the kernel evaluates the ordered vector

$$\mathcal{G}_{kind}(C, P) = \langle G_a^{kind}(C, P) \rangle_{a \in \text{ActCheck}}. \quad (5)$$

The kernel evaluates this vector strictly in the Appendix A order and accepts only an all-Pass vector. Table 2 gives one predicate per stage; no aggregate guard can move a failure across stages.

The trusted acquisition service returns an exact one-to-one mapping for the declared requirements: no declared requirement is omitted, and no unrequested evidence is included. Any unavailable evidence is returned as an explicit, authenticated *Missing* status, preventing callers from manufacturing *Defer* by suppressing inconvenient facts or injecting extraneous context. Mutable requirements specify explicit serialization keys and versions, which *VersionBinding* revalidates at commit time. Freshness is validated prior to policy evaluation and re-checked immediately before prospective state construction. Ordinary transitions revalidate the exact branch head snapshot, while *BranchCreate* revalidates target absence, creation context ξ_k , allocator versions, and genesis inputs.

The candidate seal certifies that the candidate state was derived by the pinned deterministic interpreter from the signed inputs and acquired evidence. Authorization is a separate commit-time predicate evaluated over the pre-state authority context A_X . The subsequent *Decision* stage records the policy result, and an authority transition Γ' is computed only after both authorization and decision stages succeed.

Table 2: One commit-time predicate per activation stage; read down the left pair, then down the right pair.

Stage	Predicate	Stage	Predicate
TargetLookup	Existing kind: $B[k]$ exists. BranchCreate: $B[k]$ is absent and serialized $\Xi[k]$ exists.	CandidateBinding	Seal binds proposal, evidence, interpreter, candidate root, and applicable creation-context or open-revision digest.
Allocation	Owner allocation is current; pid/oid names are live, unused, and unretired.	EffectBinding	Operations and manifest are bijective; effect scopes are live and unused.
ExpectedHead	Existing kind: $\text{Present}(d_h)$ is exact; BranchCreate: expectation is Absent.	FreshnessInitial	Revalidate time and the kind-indexed snapshot: head/epoch/dirSeq/authority, or absence/d ξ /genesis inputs.
Lifecycle	Status, writer, epoch, and kind-specific lifecycle row are admissible.	Authorization	A_χ —predecessor Γ or serialized creation context ξ_k —authorizes the operations.
HandoffRevision	Applicable target, open SourceFenced revision, reserved epoch, and dirSeq are exact.	Decision	Policy records Proceed, Reject, or Quarantine independently of authorization.
AcquisitionAuth	Acquisition result set and signer are authentic.	AuthorityTransition	A declared authority change is valid under the authorized kind context A_χ .
RequirementCoverage	Signed requirements and results form an exact ordered cover.	FreshnessFinal	Repeat the complete kind-indexed snapshot immediately before prospective construction.
VersionBinding	Policy, evaluator, authority, revocation, issuer/allocation, dependency, and processing versions are current.	WellFormedness	Receipt-independent prospective unit P_χ is complete, deterministic, and well typed.
WitnessBinding	Every required witness or authenticated Missing is bound exactly.		

Table 3: Only Commit changes authoritative state.

Result	Meaning
Commit	Install the exact successor and all acceptance metadata atomically.
Reject	A permanent validation or policy failure; retain no successor.
Quarantine	Retain sealed material outside the authoritative head.
Defer	Trusted acquisition reports required evidence unavailable.

3.2 Four stable dispositions

As illustrated in the proposal transition lifecycle (Figure 2), for an authenticated, owner-bound identifier, the kernel records exactly one of four dispositions:

$$q \in \{\text{Commit}, \text{Reject}(r), \text{Quarantine}(r), \text{Defer}(r)\}. \quad (6)$$

Every disposition is terminal for its proposal identifier. Re-submission after Defer or Quarantine uses a new identifier linked to the old attempt; a lost response is resolved by looking up the original identifier. Malformed outer framing, failed outer authentication, or an invalid allocation capability are protocol errors rather than dispositions and must not consume the identifier. After ownership succeeds, a fixed stage order chooses the result: any permanent failure that can already be evaluated precedes a trusted Missing; later predicates remain unevaluated. This prevents a missing witness from masking a known invalid proposal.

Algorithm 1 abstracts concrete wire encodings and storage driver APIs. After all validation stages pass, the kernel constructs the receipt-independent prospective unit $\hat{P}$:

$$\hat{P} = \text{BuildProspective}(X', h, B[k]). \quad (7)$$

Algorithm 1 Prepare and activate a proposal (abstract)

Require: signed proposal τ and authenticated allocation

- 1: authenticate the principal and ownership of pid
- 2: return a prior outcome or identifier conflict, if present
- 3: run **PrepStage** to Ready; acquire and derive only at named stages
- 4: on terminal preparation Fail/Missing, durably **return** $q(r_P)$
- 5: begin a transaction over the branch and every named mutable key
- 6: return the prior outcome if another retry installed it
- 7: read every row named by ActCheck
- 8: **for** a in the normative ActCheck order **do**
- 9: **if** $a = \text{WellFormedness}$ **then**
- 10: $\hat{P}_\chi \leftarrow \text{BuildProspective}_\chi(P, C)$
- 11: **end if**
- 12: evaluate only G_a^{kind} and append its stage result
- 13: **if** $a = \text{Decision}$ and result is Quarantine **then**
- 14: durably **return** Quarantine(r_Q)
- 15: **else if** result is Fail or Missing **then**
- 16: durably **return** $q(r_A)$
- 17: **end if**
- 18: **end for**
- 19: $r_C \leftarrow rb_C(\hat{P}_\chi)$; compute d_{r_C}
- 20: $\mathcal{U}_{\text{accept}}(\chi) \leftarrow \text{Finalize}_\chi(\hat{P}_\chi, d_{r_C})$
- 21: atomically install the complete unit; **return** Commit(r_C)

Construction follows a strictly cycle-free order:

$$\hat{P} \longrightarrow \text{WellFormedness}(\hat{P}) \longrightarrow r_C \longrightarrow h' \longrightarrow \mathcal{U}_{\text{accept}}. \quad (8)$$

First, the Commit receipt r_C is derived directly over $\hat{P}$, binding candidate state X' , evidence, and pinned versions without circular dependency on the receipt itself. Second, the complete head h' is finalized by binding r_C and lineage. Finally, the kernel installs the complete accepted unit $\mathcal{U}_{\text{accept}}$:

$$\mathcal{U}_{\text{accept}} = \{X', \Gamma', h', B'[k], r_C, O[pid]\}. \quad (9)$$

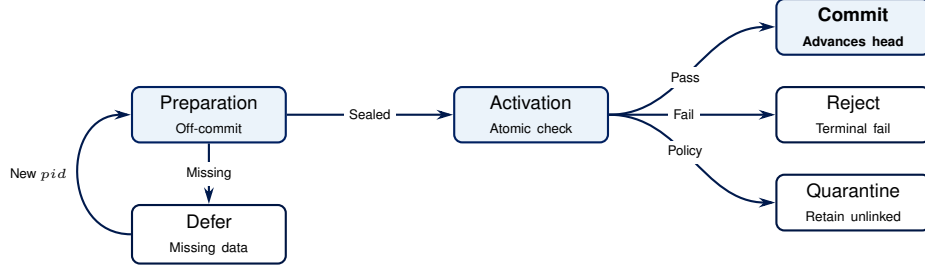


Figure 2: Proposal transition lifecycle and four terminal dispositions, expanding chronologically left-to-right.

Table 4: A signature is not evidence that its transaction committed.

Level	Established claim
Structural	The receipt has canonical syntax, typed bindings, and a valid signature under its declared key.
Attested	A trusted kernel key attests the first terminal stage and reason; this does not independently establish correct evaluation.
Replay	Retained inputs and pinned versions reproduce the stated decision or transition.
Inclusion	A certified snapshot or log proof places the receipt and outcome, and for Commit the head and lineage, in durable state.

The set $\mathcal{U}_{\text{accept}}$ denotes one all-or-nothing atomic transaction, installing candidate state X' , updated authority context Γ' , finalized branch row $B'[k]$ pointing to head h' , execution outcome $O[pid]$, and receipt record r_C . Any non-Commit disposition leaves X , B , and Γ unchanged.

Proposal and effect records need not remain online forever. Reclamation first advances a durable, monotonically increasing watermark for a non-recycled allocation scope and only then removes covered rows. An identifier below that watermark returns RetiredIdentifier rather than re-entering execution. This preserves stable identity while allowing bounded online metadata.

3.3 Receipts state what they prove

A receipt binds the proposal, disposition, reached protocol stage, decisive reason, and the evidence available at that stage. Commit receipts additionally bind the applicable typed parent and context, successor core, lineage, authority versions, and effect manifest. Preparation failures need not pretend that a canonical proposal or candidate existed.

Receipt verification has four increasing evidence levels:

The distinction matters because a kernel may sign a receipt before its storage transaction aborts. Replay strength also depends on retained objects: deleting an old proposal may preserve replay exclusion through its watermark while removing the evidence needed to reproduce its original decision.

Appendix A specifies these verification obligations.

3.4 Safety properties under the stated assumptions

Under the conditional assumptions A1–A9 introduced in Section 2 and detailed in Appendix D, the protocol satisfies three primary safety properties:

Proposition 1 (Owner-bound stable outcome). *Under A1–A4 and A8, only the principal named by an active allocation may create $O[pid]$, and at most one disposition becomes durable for that identifier.*

Proposition 2 (Single exact continuation). *Under A1–A4, at most one proposal commits from one complete predecessor or serialized target absence, and the installed state is its sealed candidate.*

Proposition 3 (At-most-once accepted effect). *Under A1, A4, and A8, each effect identifier has at most one accepted binding, including after online effect records are reclaimed.*

The proofs are short serialization arguments and appear in Appendix B. The propositions do not claim semantic correctness: a policy may approve a false memory, and an idempotent intent may still cause a non-idempotent remote action if its connector is faulty.

3.5 Realization boundary

The transaction contains no model call, network fetch, or human interaction; those finish during preparation. A relational database may lock the branch and named context rows. An object store may conditionally install one immutable commit manifest, and a replicated service may serialize one activation command. In every case readers reject a head whose accepted unit is incomplete.

CK atomicity ends at its state boundary. External actions should be committed as intents and delivered through an idempotent outbox when possible. No local protocol can make an irreversible action atomic with an unrelated remote system. Similarly, strict revocation requires the revocation row to share the activation serialization order; a remote revocation service provides only a declared bounded-staleness guarantee.

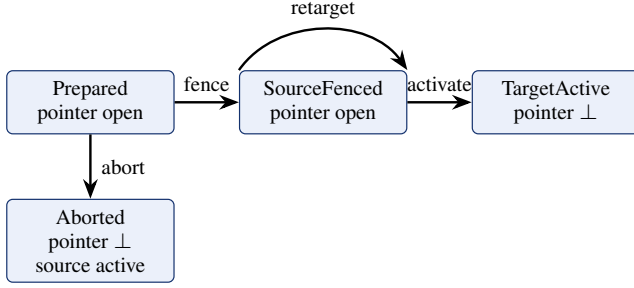


Figure 3: Handoff first removes the source writer, then activates the target. Retargeting advances the epoch while the branch remains fenced.

4 Branch Lifecycle and Restoration

Copying state, transferring write authority, and restoring old content are different operations. Treating all three as “load checkpoint” can silently fork a branch or revive an obsolete credential. CK gives each operation a typed forward transition.

4.1 Branch creation

A new branch has no predecessor. Its genesis record binds the authenticated creation request, d_ξ , fresh identifier, initial roots/writer, and absence proof. Candidate derivation uses declared genesis inputs; authorization uses serialized ξ_k , never nonexistent predecessor state or authority. Both freshness checks protect continued absence, ξ_k versions/*dirSeq*, and all bound genesis inputs. An optional source head is provenance, not a parent.

Branch creation runs Algorithm 1 with kind **BranchCreate** and records the ordinary Commit disposition. One atomic unique insert installs the initial state, admission context, genesis lineage, receipt, complete head, and branch-directory row. Two concurrent creators cannot both win the absent-key comparison. Every later transition has an ordinary accepted parent. Reconciliation between branches is a later typed proposal over exact source and target heads; the kernel does not choose a semantic merge policy.

4.2 Writer handoff

A handoff uses a monotonically versioned directory record. Figure 3 shows the small state machine; Table 5 explains its actions. Every step compares the exact current directory revision and appends a signed lifecycle result, yielding maintenance receipts distinct from proposal dispositions.

The gap between Fence and Activate is intentional. After fencing, no writer is authorized; therefore a crash cannot expose two active writers. A retry either uses the same handoff revision or fails after retargeting. Target activation is the intentional exception to writer-bearing admission: it requires

Table 5: Directory-serialized handoff actions.

Action	Atomic directory effect
Prepare	Append Prepared and open its pointer; source may still advance.
Abort	Append Aborted, clear the pointer, and keep the source active.
Fence	Append SourceFenced, advance the open pointer, clear writer, advance epoch.
Retarget	Append SourceFenced and advance the open pointer, target, and epoch.
Activate	Run Algorithm 1 against the fenced revision and append TargetActive, clear the pointer, and make the target writer active.

the exact Frozen row, writer $\perp$, open SourceFenced revision, target, canonical epoch, separately reserved successor epoch, and current admission context. Its ordinary Commit receipt is indexed by both O and H_R , so there is one authoritative activation receipt.

Prepared recovery may retry, fence, or abort. SourceFenced recovery may retry target activation or append a retarget, but it never returns to Prepared. If directory and branch storage cannot share atomic ordering, a consensus or coupled protocol is required; polling alone cannot establish writer isolation.

4.3 Migration and forward restoration

Migration names source and target schemas plus a pinned deterministic migrator. The migrator must be allowed by current policy, terminate with a well-formed target, and obey an explicit loss policy. A rollback is another forward migration, never a rewrite of an accepted head.

Restoration also creates a new successor. Let X_c be current state, X_k an authenticated historical checkpoint, M_μ an optional migration, and M a typed path mask. The restored candidate is

$$X' = \text{Restore}_M(X_c, M_\mu(X_k)), \quad \Gamma' = \Gamma_c. \quad (10)$$

Paths in M take checkpoint values after migration; all other paths take current values. Authority, revocation, writer epoch, and lifecycle fields are protected and therefore remain current. The successor’s parent is the current head, while the checkpoint is an auxiliary provenance reference. Restoration cannot truncate lineage, revive an old writer or credential, or undo an external action already performed. Appendix C gives the total postconditions and stale-request behavior for every lifecycle action.

5 Evaluation

We evaluate the Continuity Kernel (CK) design by addressing four key research questions:

- **RQ1 (Activation Integrity & Succession):** Does the 17-stage activation predicate enforce exact predecessor succession, pre-state authorization, and non-self-authorizing state transitions across all five transition kinds?
- **RQ2 (Concurrency & Replay Isolation):** Does the protocol maintain at-most-once execution identity, prevent replay attacks, and safely reclaim online proposal/effect records under concurrent races and watermark advances?
- **RQ3 (Lifecycle Isolation & Writer Fencing):** Do lifecycle rules for branch creation, writer handoff, retargeting, aborts, and forward restoration prevent split-brain access and stale writer state transitions?
- **RQ4 (Self-Critical Scope & Realization Limits):** What are the exact bounds of the formal state-space exploration, and what system-level guarantees require physical storage-engine verification beyond the abstract model?

5.1 Formal State-Space Exploration (RQ1–RQ3)

To evaluate the protocol’s state-transition logic, we construct an executable bounded model in Python (`artifacts/bounded_model.py`). The model performs exhaustive breadth-first search (BFS) state exploration over a finite abstraction of the kernel specification using only the Python standard library.

The model state space incorporates: (1) one active branch directory $B[k]$ and state store $X[k]$; (2) a shared admission context Γ and creation context $\Xi[k]$; (3) an honest proposal owner and an adversarial principal; (4) proposal identifiers $pid \in \{0 \dots 12\}$ (13 IDs) and effect identifiers $eid \in \{0 \dots 3\}$ (4 IDs); (5) source writer w_0 and target writers w_1, w_2 ; and (6) integer versioning for schema, policy, and authority contexts.

Transitions simulate all five closed transition kinds (Ordinary, Migration, Restoration, HandoffActivate, BranchCreate), exact-head concurrency races, pre-state authority changes, missing/malformed evidence, two-stage freshness expiration, resubmissions, watermark reclamation, writer fencing, retargeting, and masked restoration.

Under CPython 3.13.9, the depth-seven search evaluates 8,880,248 total transition attempts (including idempotency checks and retries). Of these, exactly

$$N_{\text{states}} = 2,808,230, \quad N_{\text{transitions}} = 5,526,474 \quad (11)$$

are unique state-changing transitions where the successor state differs from the predecessor state. As shown in Table 6, state space exploration scales exponentially up to depth 7. Depth 6 timing (124.62s) includes evaluating all 7.4M outgoing transition attempts to generate depth 7 states, whereas depth 7 timing (27.45s) reflects terminal invariant validation

Table 6: State-space expansion and BFS exploration metrics by search depth d .

Depth	Unique States	Cum. States	Cum. Attempts	Time (s)
0	1	1	0	<0.01
1	15	16	17	<0.01
2	160	176	270	0.04
3	1,455	1,631	2,959	0.33
4	11,309	12,940	27,353	2.69
5	76,028	88,968	216,413	18.88
6	445,649	534,617	1,483,284	124.62
7	2,273,613	2,808,230	8,880,248	27.45

Table 7: Distribution of transition outcomes across full state space (8.88M transition attempts).

Transition Outcome	Full Count	Share (%)
IdReuseConflict	1,673,300	18.84%
Commit (Head Advance)	916,956	10.33%
ReclaimedEffects	842,195	9.48%
RejectStaleHead	770,555	8.68%
PrefixNotFinal	731,466	8.24%
Defer (Missing Evidence)	418,375	4.71%
RejectDuplicateInProposal	418,375	4.71%
RejectUnauthorized	395,905	4.46%
RejectExpiredAtActivation	395,905	4.46%
RejectCandidateBinding	395,905	4.46%
RejectPolicy	395,905	4.46%
Prepared (Handoff)	306,121	3.45%
Other Dispositions (11 kinds)	1,219,285	13.73%
Total Attempts	8,880,248	100.00%

on the 2.27M boundary states without further successor expansion. On a standard single-threaded execution harness, kernel transition evaluation achieves a throughput of **31,132 transitions/sec** with an average evaluation latency of **32.12 μ s per transition**.

Table 7 details the empirical distribution of transition dispositions across all 8.88M evaluated transition attempts. Successful state transitions (Commit) represent 916,956 committed heads (10.33%), while concurrency and identifier reuse protections (RejectStaleHead, IdReuseConflict, ReclaimedEffects) account for 37.00% (3.28M transitions), proving that the protocol actively isolates concurrent races and stale proposals.

As summarized in Table 8, the exploration finds zero encoded invariant violations across all 2.8 million reachable states and reaches 100% of the named protocol coverage witnesses.

Table 8: Reached coverage witnesses and invariant assertions (2, 808, 230 reachable states, depth 7).

Target Category	Evaluated Invariant / Witness	Status
Dispositions	Reached Commit, Reject, Quarantine, and Defer	Pass
Activation Guard	17 stages evaluated in normative sequential order	Pass
Authorization	Self-authorizing proposal ($\tau.a$) rejected at pre-state	Pass
Freshness	Revalidated initially and immediately prior to commit	Pass
Reclamation	Safe watermark advances for pid and eid scopes	Pass
Exact Succession	At-most-one accepted successor per complete head	Pass
Writer Fencing	Stale/abandoned writer blocked from branch mutation	Pass
Handoff Lifecycle	SourceFenced, TargetActive, Retarget, and Abort	Pass
Restoration	Masked restored root with protected current fields	Pass

5.2 Invariant & Safety Analysis

The model validates three core structural properties across every reachable state:

1. Pre-State Authorization Safety (RQ1). For every committed transition, authorization is checked against the predecessor authority context Γ (or genesis context ξ_k for BranchCreate). In the model, when an adversarial proposer submits a self-authorizing proposal that attempts to inject its own required permissions into a , the Authorization stage evaluates $\text{Authorize}_\Gamma(\tau, W)$ and rejects 395,905 invalid proposals (4.46% of transitions) with RejectUnauthorized.

2. Execution Identity & Replay Safety (RQ2). Each proposal identifier pid has at most one durable disposition $O[pid]$, and each effect identifier eid is bound at most once in $E[eid]$. Watermark advances correctly transition covered online identifiers to RetiredIdentifier, preventing re-execution or identifier recycling attacks after online metadata is pruned.

3. Writer Fencing & Lifecycle Isolation (RQ3). During writer handoff, once the source writer is fenced (SourceFenced), any subsequent write attempt by the old writer w_0 fails with StaleRevision or WriterEpoch. Retargeting to a new target writer w_2 advances the writer epoch, ensuring that an abandoned target writer w_1 cannot activate the branch.

5.3 Self-Critical Thesis Analysis & Realization Limits (RQ4)

While bounded model exploration provides strong evidence for the logical consistency of the finite protocol, a self-critical assessment highlights critical gaps between the formal model and physical system implementations:

1. Abstraction vs. Physical Storage Realization. The model treats preparation and atomic activation as single logical state steps. In a production system, atomic activation relies on the underlying storage engine (e.g., PostgreSQL conditional updates, FoundationDB OCC, or Raft consensus). Physical storage crashes during write-ahead logging (WAL), network partitions, or storage-engine bugs fall outside the model’s abstract state space.

2. Signature vs. Durable Inclusion. As established in Section 3.3, a signed candidate seal or preparation receipt proves cryptographic origin (Attested level) but not transactional durability (Inclusion level). If a kernel gateway signs a receipt in memory but the backing transaction fails to commit, the signed receipt is uncommitted. Production deployments must issue Level 4 inclusion proofs backed by durable log anchors.

3. External Side-Effect Atomicity. As discussed in Section 3, CK atomicity governs state transitions *inside* its mutation boundary. It cannot make irreversible remote side effects (e.g., external API calls or physical robot actuation) atomic with internal state updates. Remote actions must be staged as idempotent outbox intents.

4. Performance & Latency Trade-offs. The protocol adds overhead during commit: 17 sequential stage checks, double freshness validation, and prospective unit construction (32.12 μs in single-threaded Python). In high-throughput environments, this overhead requires storage-level optimizations, such as single-pass SQL transaction procedures or batched multi-key compare-and-swap operations.

6 Related Work

Agent memory. Generative Agents and MemGPT organize reflection, retrieval, and long-term context [1, 2]; LoCoMo, A-MEM, Mem0, and MemOS study long-horizon evaluation, consolidation, scalable memory, and versioned management [3–6]. These systems motivate governed memory. As part of the broader Persistent Cognitive Identity (PCI) framework [7], CK addresses the narrower question of how any typed component version becomes the authoritative branch head.

MemTX is the closest agent-memory transaction system: it stages evidence-bearing beliefs under a snapshot, gates actions, and propagates repairs after retraction [8]. CK applies an activation boundary to typed persistent state and specifies pre-state authority, complete-head equality, stable four-way outcomes, writer epochs, and forward restoration. MemTxn supplies source-supported memory admission, visible-version selection, snapshots, and journal-based recovery [9]. MemTX’s unit is a belief commit and MemTxn’s is a source-supported memory update; CK specifies a typed branch-head transition and its pre-state authority boundary.

Transactions and retries. Optimistic concurrency control validates read assumptions at commit, database transactions provide atomic durability, and replicated terms fence stale leaders [10–12]. For linearizable retries, RIFL (Reusable Infrastructure for Linearizability) combines unique request identifiers, atomic completion records, and safe reclamation [13]. CK composes these mechanisms into an agent-state contract: the accepted unit also binds typed state, authority, evidence, lineage, effects, and a calibrated receipt. Commit-time authorization for LLM agents similarly exposes the gap between temporary authority and durable effects [14]; CK places that check inside a persistent-state transition.

Effects and evidence. Cordon defines task-level semantic transactions with staged effects, delegated authorization, compensation, and audit evidence [15]. Its task boundary complements CK’s branch-head boundary: an outbox intent may be a CK component, but CK cannot make an unrelated remote action atomic. PROV-DM, JSON-LD, RDFC-1.0, and Data Integrity provide provenance, canonicalization, and cryptographic proof formats [16–19]. Those standards can show derivation and integrity; the activation protocol is still needed to determine which validly encoded proposal became authoritative.

7 Conclusion

For persistent agents, state retention is not authority. Infrastructural continuity requires an authorized lineage of accepted branch heads. CK makes the transition to authority explicit: untrusted components propose typed candidates off-commit; a deterministic control plane validates an exact predecessor head and pre-state authority (or typed absence and genesis context); a short activation transaction records one stable disposition; and only Commit advances the branch head. The protocol contract addresses linearizable retries, effect uniqueness, writer epoch fencing, schema migration, and forward restoration without expanding the kernel’s trusted semantic scope.

Our safety propositions are conditional on explicit serialization, access-control, cryptographic, and durability as-

sumptions. In a depth-seven finite abstraction, an executable bounded model explores 2,808,230 reachable states and 5,526,474 state-changing transitions, finding zero encoded invariant violations and reaching 100% of named coverage witnesses. The model validates logical protocol consistency within its finite bounds; it does not establish unbounded mathematical correctness, physical storage driver conformance, or throughput performance under hardware faults. The core contribution of this work is the activation contract itself—providing a principled systems foundation that separates stored retention from authoritative reachability and distinguishes cryptographic receipt signatures from durable transactional inclusion.

References

- [1] Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.
- [2] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.
- [3] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870. Association for Computational Linguistics, 2024.
- [4] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems 38*, pages 17577–17604. Curran Associates, Inc., 2025.
- [5] Prateek Chikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.
- [6] Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. MemOS: A memory OS for AI system, 2025. Preprint, arXiv:2507.03724v4, version 4, 3 December 2025. <https://arxiv.org/abs/2507.03724v4>.
- [7] Jun He and Deying Yu. Persistent cognitive identity: A systems architecture for continuity across AI substrates and embodiments. Position and architecture manuscript, OpenKedge.io, 2026.
- [8] Xiaoyang Li, Yiqi Wang, Haohui Lu, Zhi Chen, Mo Li, Pingan Song, Mingkai Zheng, and Taotao Cai. MemTX: Transactional belief commit for stateful agent memory, 2026. Preprint, arXiv:2607.23929v2, version 2, 28 July 2026. <https://arxiv.org/abs/2607.23929v2>.
- [9] Hanshui Cui, Zhiqing Tang, Zhi Yao, Fanshui Meng, Qianli Ma, and Weijia Jia. MemTxn: A transaction boundary for source-supported updates and complete-state recovery in agent memory. *arXiv preprint arXiv:2607.27834*, 2026. Version 1, submitted July 30, 2026.
- [10] H-T Kung and John T Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.
- [11] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.
- [12] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014.
- [13] Collin Lee, Seo Jin Park, Ankita Kejriwal, Satoshi Matsushita, and John Ousterhout. Implementing linearizability at large scale and low latency. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 71–86. ACM, 2015.
- [14] Igor Santos-Gruero. Temporary authority, permanent effects: Commit-time authorization for LLM agents, 2026. Preprint, arXiv:2607.10487v1, version 1, 11 July 2026. <https://arxiv.org/abs/2607.10487v1>.
- [15] Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic transactions for tool-using LLM agents, 2026. Preprint, arXiv:2606.17573v1, version 1, 16 June 2026. <https://arxiv.org/abs/2606.17573v1>.
- [16] Luc Moreau and Paolo Missier. PROV-DM: The PROV data model. W3C Recommendation, 2013.
- [17] W3C JSON-LD Working Group. JSON-LD 1.1: A JSON-based serialization for linked data. W3C Recommendation, 2020. 16 July 2020.
- [18] Dave Longley, Gregg Kellogg, and Dan Yamamoto. RDF dataset canonicalization. W3C Recommendation, 2024. 21 May 2024, Sections 4.4.3 and 7.1.
- [19] W3C Verifiable Credentials Working Group. Verifiable credential data integrity 1.0: Securing the integrity of verifiable credential data. W3C Recommendation, 2025. 15 May 2025.
- [20] W3C JSON-LD Working Group. JSON-LD 1.1 processing algorithms and API. W3C Recommendation, 2020. 16 July 2020.

A Normative Receipt Semantics

A.1 Stage Sequences and Disposition Mapping

This appendix fixes the stage order and minimum contents left implicit in the main-text receipt abstractions. Preparation and activation checks evaluate two strictly ordered, closed sequences:

Preparation Stage Order (PrepStage):

RawScreen $\prec$ Decode $\prec$ Canonicalize
 $\prec$ ProposalAuth $\prec$ StaticValidation
 $\prec$ TargetLookup $\prec$ AcquisitionAuth
 $\prec$ RequirementCoverage $\prec$ WitnessAuth
 $\prec$ AcquisitionTime $\prec$ CandidateDerivation
 $\prec$ PolicyAuthorization $\prec$ Ready.

Activation Check Order (ActCheck):

TargetLookup $\prec$ Allocation $\prec$ ExpectedHead
 $\prec$ Lifecycle $\prec$ HandoffRevision
 $\prec$ AcquisitionAuth $\prec$ RequirementCoverage
 $\prec$ VersionBinding $\prec$ WitnessBinding
 $\prec$ CandidateBinding $\prec$ EffectBinding
 $\prec$ FreshnessInitial $\prec$ Authorization
 $\prec$ Decision $\prec$ AuthorityTransition
 $\prec$ FreshnessFinal $\prec$ WellFormedness.

In both pipelines, TargetLookup requires existence for existing-branch kinds and absence plus serialized ξ_k for BranchCreate. The latter's preparation derives from genesis inputs and runs PolicyAuthorization against ξ_k ; activation ExpectedHead accepts only Absent. Existing kinds use the current predecessor X, Γ and Present(d_h).

An outcome-eligible failure at position j records

$$v_i = \begin{cases} \text{Pass}, & i < j, \\ \text{Fail}(r) \text{ or } \text{Missing}(r), & i = j, \\ \text{NE}, & i > j, \end{cases} \quad (12)$$

where NE means not evaluated. A verifier never asks a failing predicate to pass. Known permanent failures precede a trusted Missing, so unavailability cannot mask a rejection. Malformed outer framing, failed outer authentication, and invalid allocation are protocol errors: they create neither $O[pkey]$ nor a receipt.

For $T \in \{P, A, Q, C\}$, rb_T canonically encodes and signs exactly r_T , computes d_{r_T} , and uses the sole receipt-store rule $\mathcal{R}[d_{r_T}] = r_T$. No other proposal-receipt builder or receipt key exists. Table 10 is the closed stage-reason map.

At activation TargetLookup, BranchCreate evaluates target presence before creation context: $B[k] \neq \text{Absent}$ yields TargetAlreadyExistsAtActivation; $B[k] = \text{Absent} \wedge \Xi[k]$ missing yields CreationContextMissingAtActivation. Freshness checks evaluate in deterministic intra-stage order. For existing branches: (1) $t > \text{deadline}$ yields ExpiredAtActivation; (2) head mismatch yields StaleHead; (3) epoch mismatch yields StaleEpoch; (4) $dirSeq$ mismatch yields StaleDirSeq; (5) authority/revocation/policy mismatch yields StaleAuthority; (6) pinned version mismatch yields

StaleVersion. For BranchCreate: (1) $t > \text{deadline}$ yields ExpiredAtActivation; (2) post-lookup target appearance ($B[k] \neq \text{Absent}$) yields TargetAppearedAtActivation; (3) creation context ($\Xi[k]$ missing/altered) yields StaleCreationContext; (4) genesis profile/allocator version mismatch yields StaleVersion. Every reason selects Reject (rb_A), requires preparation and prefix Π_{j-1} , forbids later fields, and leaves X, B, Γ, Ξ, E unchanged. Earlier stages dominate later stages.

At preparation, a Fail selects Reject/ rb_P and only EvidenceUnavailable is Missing, selecting Defect/ rb_P . At activation, every reason selects Reject/ rb_A except PolicyQuarantine, which selects Quarantine/ rb_Q ; all-pass through WellFormedness selects Commit/ rb_C . Thus Algorithm 1 has no other terminal path. Adding a stage or reason requires a new receipt schema version. An exact retry returns the indexed result; any changed attempt under the same pid conflicts.

Lifecycle maintenance is typed separately. Its closed actions are Prepare, Fence, Retarget, and Abort. Its closed results are Applied, StaleHead, StaleRevision, StaleDirSeq, WriterEpoch, TargetConflict, InvalidState, and Unauthorized. $rb_L = \text{Build}(\text{LifecycleReceiptV1})$ binds the action identifier, pre/post directory tuples, optional input/output revision digests, result, trusted time, and signature. It is stored in $\mathcal{R}$ and indexed by the typed H_R key defined in Appendix C; it never creates $O[pkey]$ and is not a fifth proposal disposition. Its digest is $d_{r_L} = d_{\text{LifecycleReceiptV1}}(r_L)$ under Equation 14. Replay recomputes the action and its pre/post tuples; inclusion requires a certified state containing $\mathcal{R}, H_R$, and, for Applied, the assigned F and B records.

Lifecycle field presence is action-indexed. Every r_L binds its common fields, authenticated request, requested pre-directory tuple, result, and exact post tuple. For Applied, the input/output revision states are Prepare $\perp \rightarrow$ Prepared, Fence Prepared $\rightarrow$ SourceFenced, Retarget SourceFenced $\rightarrow$ SourceFenced, and Abort Prepared $\rightarrow$ Aborted. For an allowed non-Applied result, post equals pre, output is forbidden, and neither F nor B changes; Prepare forbids input, while the other actions require the requested input. Let

$$S = \{\text{StaleRevision}, \text{StaleDirSeq}, \text{WriterEpoch}, \text{InvalidState}, \text{Unauthorized}\}.$$

The exact failure sets are $S \setminus \{\text{StaleRevision}\} \cup \{\text{StaleHead}, \text{TargetConflict}\}$ for Prepare, $S \cup \{\text{StaleHead}\}$ for Fence, $S \cup \{\text{TargetConflict}\}$ for Retarget, and S for Abort. No other action/result pair is schema-valid.

A.2 Field presence and early input binding

After owner authentication, the gateway computes

$$d_{raw} = d_{\text{WireInputV1}}(rawBytes). \quad (13)$$

An early RawScreen, Decode, or Canonicalize receipt binds d_{raw} but omits d_r ; there is not yet a canonical proposal to hash. Let j be the terminal stage. A receipt requires the common fields, the terminal Fail/Missing observation, and exactly the fields whose producing stages successfully completed before j . It forbids fields first produced at j or later. Thus later NE stages impose no field obligation. The only profile choice is raw retention: d_{raw} is required

Table 9: The four proposal-receipt variants and their terminal semantics.

Tag	Terminal location	Stage/reason rule	Builder	Durable effect
r_P	Preparation	First failed or trusted-Missing preparation stage; later stages are NE	rb_P	Insert one owner-bound $O[pkey]$ and $\mathcal{R}[d_{r_P}] = r_P$; no candidate becomes authoritative.
r_A	Serialized activation	First failed activation check; earlier checks pass and later checks are NE	rb_A	Insert one $O[pkey]$ and $\mathcal{R}[d_{r_A}] = r_A$; X, B, Γ, E are unchanged.
r_Q	Decision = Quarantine	Earlier checks pass, Decision records Quarantine, later checks are NE	rb_Q	Insert $O[pkey]$, $\mathcal{R}[d_{r_Q}] = r_Q$, and unreachable $Q[pkey]$ only.
r_C	All activation checks Pass	P_χ passes WellFormedness; the terminal result is Commit	rb_C	Store $\mathcal{R}[d_{r_C}] = r_C$ and install the complete accepted unit of Equation 9.

Table 10: Closed V1 stage–reason map. “Limit” abbreviates CanonicalizationLimit; a stage vector disambiguates repeated reasons.

Preparation stage $\mapsto$ allowed reason	Activation check $\mapsto$ allowed reason
RawScreen $\mapsto$ {Limit}; Decode $\mapsto$ {DecodeFailure, Limit}; Canonicalize $\mapsto$ {CanonicalizationFailure, Limit}; ProposalAuth $\mapsto$ {ProposalAuthentication}; StaticValidation $\mapsto$ {MalformedProposal, EffectManifest, DuplicateEffectInProposal}; TargetLookup $\mapsto$ {UnknownTargetAtPreparation, CreationContextMissing, BranchAlreadyExists}; AcquisitionAuth $\mapsto$ {UntrustedRequirementResult, AcquisitionAuthentication}; RequirementCoverage $\mapsto$ {RequirementCoverage}; WitnessAuth $\mapsto$ {WitnessAuthentication}; AcquisitionTime $\mapsto$ {ExpiredAtPreparation, EvidenceUnavailable}; CandidateDerivation $\mapsto$ {CandidateDerivation}; PolicyAuthorization $\mapsto$ {PreparationUnauthorized}.	TargetLookup $\mapsto$ {TargetMissingAtActivation, TargetAlreadyExistsAtActivation, CreationContextMissingAtActivation}; Allocation $\mapsto$ {AllocationRevoked}; ExpectedHead $\mapsto$ {StaleHead}; Lifecycle $\mapsto$ {OrdinaryLifecycle, HandoffLifecycle}; HandoffRevision $\mapsto$ {StaleHandoffRevision, StaleDirSeq, TargetWriter, StaleEpoch}; AcquisitionAuth $\mapsto$ {AcquisitionAuthentication}; RequirementCoverage $\mapsto$ {RequirementCoverage}; VersionBinding $\mapsto$ {StaleVersion}; WitnessBinding $\mapsto$ {WitnessBinding}; CandidateBinding $\mapsto$ {CandidateBinding}; EffectBinding $\mapsto$ {EffectManifest, EffectScope, RetiredEffect, DuplicateEffect}; FreshnessInitial, FreshnessFinal $\mapsto$ {ExpiredAtActivation, StaleHead, StaleEpoch, StaleDirSeq, StaleAuthority, StaleVersion, StaleCreationContext, TargetAppearedAtActivation}; Authorization $\mapsto$ {Unauthorized}; Decision $\mapsto$ {PolicyReject, PolicyQuarantine}; AuthorityTransition $\mapsto$ {InvalidAuthorityTransition}; WellFormedness $\mapsto$ {InvalidSuccessor}.

Table 11: Stage-indexed field production. A field is required exactly after its producer passes and forbidden beforehand.

Field group	Producer that must Pass	Contents
Common	Outcome-eligible owner authentication	Type/version, subject/branch, principal, allocation scope, pid , tag, disposition, stage vector, reason, issue time, kernel key and signature
Raw input	Request capture under the signed profile	d_{raw} ; profile makes it required or forbidden, never optional
Canonical proposal	Canonicalize	d_r , typed Present(d_h)/Absent expectation, kind reference, and unverified signature/credential bytes
Authenticated proposal	ProposalAuth	Verified signer, owner, principal, allocation scope, and proposal-authentication result
Acquisition	RequirementCoverage (preparation)	Ordered requirements/results, witness and dependency-version commitments
Candidate package	CandidateDerivation	Candidate root, pinned interpreter, candidate-seal key/signature, effect manifest, and direct applicable d_ξ or open SourceFenced-revision digest
Activation prefix Π_j	Corresponding ActCheck stage	Existing row or absence plus d_ξ ; allocation; exact Present/Absent expectation; lifecycle; open revision; acquisition signer; exact cover; versions; witness binding; candidate binding; effect binding; initial freshness; authorization; Decision; authority-after; final freshness—each introduced only by its same-named successful stage
Prospective unit	BuildProspective after FreshnessFinal	Produced successor/core/lineage inputs, receipt-independent branch-row plan ($\widehat{B}_\chi$), receipt-independent assignment plan ($\widehat{\Delta}_\chi$), and deterministic parameters for receipt-dependent finalization
Finalized Commit	WellFormedness Pass	Commit receipt/digest, complete head, outcome, H_R , effects, receipt store, and directory binding

when `rawRetention=true` and forbidden otherwise. Absence is encoded by the typed variant, never a zero digest.

Consequently, a `CandidateDerivation` failure forbids candidate fields. Every activation receipt requires the completed preparation package, but an early `TargetLookup` or `ExpectedHead` failure contains only its successful activation prefix and terminal observation—never later freshness, authorization, Decision, authority-after, or successor fields. A `Decision Reject` or `Quarantine` binds that terminal policy result; it does not manufacture an `AuthorityTransition` prefix. A `BranchCreate` prefix binds absence and d_ξ , never a predecessor head, state, or authority. An `InvalidSuccessor` receipt may bind the

produced prospective fields but forbids every finalized Commit field.

Exact evidence coverage is a bijection between the signed ordered requirement sequence and acquired results: no omission, duplicate, substitution, injection, or forbidden reordering is accepted. Each result is either a signed witness or an authenticated Missing. A candidate seal authenticates these bindings; it is neither pre-state authorization nor proof of durable commit.

A.3 Verification levels

Verification is monotone but not automatic:

1. **Structural authenticity** checks the receipt variant, canonical syntax, typed digests, signature, key scope, and internal bindings.
2. **Kernel-attested reason** additionally establishes that the authenticated kernel reported the declared first terminal stage. It does not independently establish that the predicate was evaluated correctly.
3. **Independent replay** recomputes each predicate through the terminal stage from retained inputs and pinned versions. For Commit it also recomputes candidate derivation, the authority transition, manifest, successor core, and lineage. Replay is unavailable when any required object or interpreter is absent.
4. **Durable inclusion** verifies a certified snapshot, commit manifest, or replicated-log proof containing $O[pkey]$ and the receipt; Commit also requires the installed head and lineage. An archive proof terminates at a retirement root in such a certified state. A signature or locator alone is not inclusion.

A conforming verifier first completes level 1, reports level 2 only for a trusted kernel key, attempts level 3 only with a complete replay package, and reports level 4 only with an inclusion proof. Thus a signed receipt from an aborted storage transaction cannot be upgraded to a committed fact.

B Typed Lineage and Cycle-Free Construction

B.1 Acyclic Hash Graph and Construction Order

Here $K_H = \text{HandoffResultKeyV1}(pkey, hid)$ and $K_G = \text{DirectoryKeyV1}(k, dirSeq')$; these are definitions, not aliases for differently keyed records.

Every content reference uses a type- and version-separated digest

$$d_T(y) = H(\text{LP}(\text{CK}) \parallel \text{LP}(T) \parallel \text{u32}(v) \parallel \text{LP}(\text{Canon}_T(y))), \quad (14)$$

where LP is an unambiguous length prefix. Logical proposal and effect identifiers are allocated names, not content digests.

Equation 1 is the sole definition of immutable head core, complete head, and branch row; this appendix introduces no shortened head. The closed parent type is

$$\begin{aligned} \text{LineageParent} = & \text{AcceptedParent}(d_h) \\ & | \text{GenesisParent}(d_g). \end{aligned} \quad (15)$$

Exactly one branch-genesis edge uses the second variant. Every later edge uses the complete current head. A checkpoint, migration input, or handoff source is an auxiliary typed reference and can never occupy the parent field.

A **LineageEdgeV1** binds *sid*, *bid*, transition kind, lineage parent, d_{h_c} , d_τ , the proposal allocation key, policy and authority versions, epoch and sequence, plus the transition-kind-specific source, checkpoint, migration, or handoff references. A **BranchGenesisV1** binds the authenticated creation request, absent-directory proof, initial state and admission roots, writer, epoch 0, sequence 0, and optional source provenance. Source provenance does not create cross-branch ancestry. In Table 12, every V1 name has version 1, that exact name as its domain tag, and its type-specific canonical encoder under Equation 14. Component roots carry their declared type/schema

domain. Rows 3–10 and 12 are content-addressed by their displayed typed digest; rows 1–2, 11, and 13–17 use exactly the displayed map key. Map keys are allocated typed names, not hidden hash inputs.

For **HandoffActivate**, **TransitionV1** directly binds $d_{f_j^{\text{SourceFenced}}}$; **AcquisitionV1** binds it transitively through d_τ , and the seal binds it directly. The revision must be the current open **SourceFenced** revision or **HandoffRevision** fails stale. $h_c.\text{auxRef}$ also names this input but never the output or seal. The head core has no direct seal dependency: h' binds d_{seal} transitively through d_{r_C} . The same rule applies to **BranchCreate** with d_ξ : direct in the transition and seal, transitive through d_τ in the acquisition, and revalidated from $\Xi[k]$ at both freshness checks. The output revision binds the open input and d_{h_c} ; successful activation then clears the pointer while retaining both revisions in F . The order is

$$\begin{aligned} (d_\xi?, d_{f_j^{\text{SourceFenced}}?}) & \prec d_\tau \prec d_W \prec (X', \Gamma', M_E) \\ & \prec d_g? \prec d_{\text{seal}} \prec h_c \prec d_{f_{j+1}^{\text{TargetActive}}?} \\ & \prec \ell \prec r_C \prec h' \prec d_{h'} \\ & \prec (O, E, H_R?, B', G') \prec \text{Publish} \prec i. \end{aligned} \quad (16)$$

The receipt binds the prospective unit $\hat{P}_x$, not the complete head; the complete head h' then binds d_{r_C} , producing $d_{h'}$. The lineage edge likewise binds h_c and typed predecessor, not the complete successor. Therefore every digest edge points left in Equation 16, and the audited graph is acyclic. Outcome, effect, handoff-result, branch, and directory records point only to named earlier digests; no earlier object points back. Inclusion evidence is constructed only after publication. Atomic publication changes visibility, not this hash order.

B.2 Proof sketches

Proposition 1. Every outcome-writing path first authenticates the proposal allocation and revalidates it in the proposal-scope serialization domain. Invalid outer requests never write an outcome. The first authorized write uniquely inserts $O[pkey]$; a compatible retry returns it, an incompatible retry conflicts, and a reclaimed identifier is rejected by the monotone watermark. A second durable disposition is therefore impossible under A1–A4 and A8. $\square$

Proposition 2. The seal binds the proposal, evidence, interpreter, typed Present/Absent expectation, candidate root, and effect manifest. Activation serializes the branch key and compares either the complete head or continued absence plus d_ξ . The first Commit changes that serialization point atomically; every contender then fails exact-head or absence admission. Thus at most one sealed candidate becomes the exact successor or genesis under A1–A4. $\square$

Proposition 3. The operation–manifest bijection rejects an effect identifier repeated within a proposal. Across proposals, Commit uniquely inserts $E[eid]$ while holding its scope. Reclamation advances the scope watermark before deleting online rows, and scopes are never recycled. Hence deletion cannot reopen an accepted identifier under A1, A4, and A8. $\square$

These are conditional serialization arguments. They establish neither content truth nor correctness of policy, canonicalization, cryptography, or storage implementations.

Table 12: Exact typed construction dependencies. Every input appears earlier.

	Object and domain	Immediate inputs	Downstream references
1	$d_\xi / \text{CreationContextV1}; \Xi[k]$	Namespace, allocator and policy versions, dirSeq , authority root, genesis profile	BranchCreate proposal, seal, authorization, freshness
2	$d_{f\text{SourceFenced}} /$ existing $\text{HandoffRevisionV1}; F[\text{hid}, j]$	Prior digest, SourceFenced state, branch/head, target, epochs, directory tuple, context versions	HandoffActivate proposal, seal, admission
3	$d_\tau / \text{TransitionV1}$	$pkey, k$, Present/Absent expectation, χ , operations, authority changes, dependencies, requirements, validity, kind reference, and direct applicable d_ξ or $d_{f\text{SourceFenced}}$	Acquisition and candidate derivation
4	$d_W / \text{AcquisitionV1}$	d_τ , ordered requirements/results, signer/key identifiers and pinned versions; detached authentication is verified	Candidate seal
5	$d_{X'}, d_{\Gamma'}, d_{ME} /$ $\text{StateRootV1}, \text{AuthorityStateV1}, \text{EffectManifestV1}$	Ordered typed successor component digests; ordered authority entries; ordered $(\text{opIndex}, \text{eid}, \text{opDigest})$, derived from initial inputs with d_τ, d_W and the interpreter	Seal, head core
6	$g, d_g / \text{BranchGenesisV1}$	d_τ, d_ξ , absence proof, initial roots/writer/zero counters, optional source provenance	Genesis parent
7	$d_{\text{seal}} / \text{CandidateSealV1}$	d_τ, d_W , interpreter, component/effect roots, and direct applicable d_ξ or open-revision digest	Receipt
8	$h_c, d_{h_c} / \text{HeadCoreV1}$	Exact Equation 1 fields: root, typed parent, versions, counters, time, kind, and closed auxRef	Output revision, lineage
9	$d_{f\text{TargetActive}} / \text{new}$ HandoffRevisionV1	$d_{f\text{SourceFenced}}, d_{h_c}$, exact target, epochs, directory tuple and context versions	Lineage, receipt, HResult
10	$\ell, d_\ell / \text{LineageEdgeV1}; L[\lambda]$	Parent, $d_{h_c}, d_\tau, pkey$, versions, and applicable input/output handoff digests	Commit receipt
11	$r_C, d_{r_C} /$ $\text{CommitReceiptV1}; \mathcal{R}[d_{r_C}]$	$pkey, d_\tau, d_W, d_{\text{seal}}$, parent, d_{h_c}, d_ℓ , all-Pass vector, $\hat{P}_\chi$ prospective fields, effect/authority and handoff bindings	Complete head, accepted records
12	$h', d_{h'} / \text{CompleteHeadV1}$	h_c, d_ℓ, d_{r_C}	Outcome, branch row, effect, handoff result, directory binding
13	$\text{OutcomeV1}; O[pkey]$	$pkey$, Commit tag, d_{r_C}, h'	Retry, inclusion
14	$\text{EffectRecordV1}; E[\text{eid}]$	$\text{eid}, pkey$, operation index/digest, $d_{r_C}, d_{h'}$	Inclusion, duplicate exclusion
15	$\text{HandoffResultV1}; H_R[K_H]$	$pkey, \text{hid}$, input/output revision digests, d_{r_C}, h'	Retry, inclusion
16	$\text{BranchRowV1}; B[k]$	k, h' , status, writer, epoch, dirSeq , open pointer, immutable createdFrom	Authoritative lookup
17	$\text{DirectoryBindingV1}; G[K_G]$	$k, pkey, \chi$, pre/post dirSeq , allocator/context versions, $d_{r_C}, d_{h'}$	Freshness, inclusion
18	$i / \text{InclusionProofV1}$, later	Certified state containing rows 11–17	External verifier; never an input to h'

C Total Lifecycle, Handoff, and Restoration Semantics

C.1 Handoff Union Types and Kind-Specific Admission

Equation 1 is normative for B . In particular, dirSeq lives only in B ; a revision records $\text{dirSeqIn}, \text{dirSeqOut}$ as snapshots. $\text{createdFrom} \in \{\text{None}, \text{SourceProvenance}(d_h)\}$ is assigned at genesis and copied byte-for-byte thereafter. Pointer activity and revision state are distinct closed types:

$$\begin{aligned} \text{HPtr} &= \perp \mid \text{Open}(\text{hid}, d_f), \\ \text{FState} &= \text{Prepared} \mid \text{SourceFenced} \mid \text{Aborted} \mid \text{TargetActive}. \end{aligned} \quad (17)$$

An ordinary Commit during Prepared is nonconflicting but makes that revision's source-head binding stale. Fence then returns **State-Head**; Abort followed by a fresh Prepare is required before fencing. Any change to target, epoch, dirSeq , input revision, d_G , or the current admission versions similarly invalidates an older target package.

C.2 Authoritative State Postconditions

Let $b = \langle h, s, w, e, d, z, c \rangle$ follow the field order of Equation 1; $b[a \leftarrow x]$ copies b and explicitly replaces field a . The HResult map is H_R . Let $K_L = \text{LifecycleKeyV1}(\text{hid}, \text{lid})$ and $K_H = \text{HandoffResultKeyV1}(pkey, \text{hid})$ be disjoint typed keys; lid and pid are owner-allocated, non-recycled names. A unique insert makes each H_R key immutable: an exact retry returns the stored value and a variant conflicts. “Append r_L ” below means

$\mathcal{R}[d_{r_L}] = r_L$ in the same atomic unit. For compactness, let $\mathcal{S} = \langle X, \Gamma, \text{rev}, \text{issuer}, \text{policy}, \Xi, h, L, O, E, G \rangle$; $\mathcal{S}' = \mathcal{S}$ explicitly copies every listed family, including $\Xi' = \Xi$.

Every **HandoffRevisionV1** binds its predecessor revision, action/state, branch key, source head/writer, target writer, old/new and reserved epochs, $\text{dirSeqIn}, \text{dirSeqOut}$, pinned policy/authority/revocation versions, digests of Γ, G , optional d_{h_c} , reason, and trusted time. $F[\text{hid}, j]$ is append-only. Maintenance changes only the fields and indexes assigned in Table 14; it never creates an O entry.

The consumed target package is

$$P_H = \langle pkey, d_\tau, d_W, d_{\text{seal}}, \text{hid}, d_{f\text{SourceFenced}}, d_{h_s}, w_t, e_c, e_r, \text{dirSeq}, \text{policyV}, \text{authV}, \text{revV}, d_\Gamma, d_G \rangle, \quad (18)$$

where e_c is the pre-activation canonical epoch and e_r is the separately reserved successor epoch. Admission requires both to remain exact and equal. The accepted unit creates the distinct output $f_{j+1}^{\text{TargetActive}}$; Appendix B fixes every digest placement. Precisely, $\Gamma_T = \Gamma_{\text{current}}$ when no separately authorized authority transition exists, and $\Gamma_T = \Gamma'$ only when the same accepted proposal authorizes and binds Γ' . The identical rule applies to revocation, issuer, and policy; the fenced historical head is never their source.

C.3 Restoration Path Mask and Protected Semantics

Restoration binds

$$\begin{aligned} \text{aux}_R &= \langle d_{h_c}, d_{\text{root}_c}, d_{h_k}, d_{\text{root}_k}, \\ &\quad \text{proof}_k, d_\mu, M, \text{componentRoots} \rangle \end{aligned} \quad (19)$$

Table 13: Transition-kind-specific head admission within the single ordered ActCheck vector.

Kind	Serialized G_{head}^{kind} precondition	Parent and disposition
Ordinary, Migration, Restoration HandoffActivate	$B = \langle h, \text{Active}, w, e, d, z, c \rangle$; exact $expected = h$, writer w , epoch e , and $dirSeq = d$; an Open z must reference Prepared, never SourceFenced. $B = \langle h_s, \text{Frozen}, \perp, e_c, d, \text{Open}(hid, d_{fSourceFenced}), c \rangle$; exact fenced head, target writer, $dirSeq = d$, input revision and current context; the revision separately binds $reservedEpoch = e_r$, and admission requires the proposal to bind both e_c, e_r with $e_c = e_r$.	AcceptedParent(d_h); one of the four proposal dispositions. AcceptedParent(d_{h_s}); an ordinary Commit/Reject/Quarantine/Defer outcome under $O[pkey]$.
BranchCreate	$B[k]$ absent, $expected = \text{Absent}$, and exact serialized $\Xi[k] = \xi_k$; authenticated allocation/absence proof and zero initial counters.	GenesisParent(d_g); ordinary Commit with kind BranchCreate.

Table 14: Total lifecycle postconditions. Every authoritative family is assigned explicitly or through the complete accepted unit; $\Xi' = \Xi$ holds for all rows.

Action	Exact precondition	Complete authoritative poststate	Receipt/result records
Genesis	$B[k]$ absent; BranchCreate row of Table 13	Install $U_{accept}(\text{BranchCreate})$: $X_0, \Gamma_0, g, h_0, L, E[M_E], G'_\chi$ and $B'[k] = \langle h_0, \text{Active}, w_t, 0, 0, \perp, c_0 \rangle$; $\Xi' = \Xi$; F, H_R not written.	$O[pkey] = \text{OutcomeV1}(\text{Commit}, d_{r_C}, h_0)$ and $\mathcal{R}[d_{r_C}] = r_C$.
Prepare	$b.status = \text{Active}$, $b.writer = w_s$, exact head/epoch/ $dirSeq$, and $b.handoff = \perp$	$S' = S$; $B' = b[dirSeq \leftarrow d + 1, handoff \leftarrow \text{Open}(hid, d_{fPrepared})]$, preserving $createdFrom$; append $f_{j+1}^{Prepared}$ to F and reserve $e_r = e + 1$.	Append $r_L = rb_L(\text{Applied})$ and $H_R[K_L] = \text{Lifecycle}(d_{r_L}, \perp, d_{f_{j+1}^{Prepared}})$.
Abort	Exact $f_j^{Prepared}$; Active, writer w_s , epoch e , $dirSeq = d$, and matching Open pointer	$S' = S$; $B' = b[dirSeq \leftarrow d + 1, handoff \leftarrow \perp]$, preserving $createdFrom$; append the Aborted revision to F and cancel e_r .	Append r_L and $H_R[K_L] = \text{Lifecycle}(d_{r_L}, d_{f_j^{Prepared}}, d_{f_{j+1}^{Aborted}})$.
Fence	Exact $f_j^{Prepared}$, its final h , and active source tuple	$S' = S$; preserve $createdFrom$ and set $B'.status = \text{Frozen}$, $writer = \perp$, $epoch = e_r$, $dirSeq = d + 1$, and $handoff = \text{Open}(hid, d_{fSourceFenced})$; append $f_{j+1}^{SourceFenced}$.	Append r_L and $H_R[K_L] =$ $\text{Lifecycle}(d_{r_L}, d_{f_j^{Prepared}}, d_{f_{j+1}^{SourceFenced}})$.
Retarget	Exact $f_j^{SourceFenced}$; Frozen, writer $\perp$, epoch e , $dirSeq = d$, and matching Open pointer	SourceFenced revision with old epoch e and new/reserved epoch e_r . $S' = S$; preserve status, writer, head, and $createdFrom$; set epoch $e + 1$, $dirSeq = d + 1$, and $handoff = \text{Open}(hid, d_{fSourceFenced})$; append $f_{j+1}^{SourceFenced}$.	Append r_L and $H_R[K_L] =$ $\text{Lifecycle}(d_{r_L}, d_{f_j^{SourceFenced}}, d_{f_{j+1}^{SourceFenced}})$.
Activate	HandoffActivate row of Table 13	Install $U_{accept}(\text{HandoffActivate})$: $X'_T, h', L, E[M_E], G'_\chi$, append $f_{j+1}^{TargetActive}$, set $\Xi' = \Xi$ and $B' = \langle h', \text{Active}, w_t, e_c, d + 1, \perp, c \rangle$; install the current-or-authorized $\Gamma_T, rev_T, issuer_T, policy_T$. Install $U_{accept}(\text{Restoration})$ with $X' = \text{Restore}_M(X_c, M_\mu(X_k))$, new $h', L, E[M_E], G'_\chi$, $\Xi' = \Xi$ and $B' = b[head \leftarrow h']$; status, writer, epoch, $dirSeq$, handoff, $createdFrom$, and protected authority families stay current.	$\mathcal{R}[d_{r_C}] = r_C$, $O[pkey] = \text{OutcomeV1}(\text{Commit}, d_{r_C}, h')$, and $H_R[K_H] = \text{Activation}(d_{f_j^{SourceFenced}}, d_{f_{j+1}^{TargetActive}}, d_{r_C}, h')$; the receipt digest is identical.
Restore	Ordinary-kind row; authenticated checkpoint/proof/mask and optional pinned migrator	Install $U_{accept}(\text{Restoration})$ with $X' = \text{Restore}_M(X_c, M_\mu(X_k))$, new $h', L, E[M_E], G'_\chi$, $\Xi' = \Xi$ and $B' = b[head \leftarrow h']$; status, writer, epoch, $dirSeq$, handoff, $createdFrom$, and protected authority families stay current.	Ordinary Commit in O and $\mathcal{R}$; no H_R or F write; checkpoint is auxiliary lineage, never a parent.
Failure	Any authenticated maintenance action whose exact precondition fails	$S' = S$, $B' = B$, and $F' = F$; no accepted state, effect, directory, or lifecycle-revision write.	Append $r_L = rb_L(q_L)$ and the action-indexed $H_R[K_L]$ from Appendix A; the output revision is forbidden.

and computes $X' = \text{Restore}_M(X_c, M_\mu(X_k))$. For protected paths

$$\begin{aligned}
P &= \{\text{authority}, \text{policy}, \text{revocation}, \text{issuer}, \\
&\quad \text{writer}, \text{epoch}, \text{dirSeq}, \text{lifecycle}, \text{createdFrom}\}, \\
M \cap P &= \emptyset, \quad X'|_P = X_c|_P.
\end{aligned} \tag{20}$$

Malformed or unauthenticated outer lifecycle requests create no receipt.

D Deterministic Resource Profile and Artifact

D.1 Canonicalization resource profile

Canonicalization is governed by the signed, versioned profile

$$\begin{aligned}
\kappa &= \langle \text{jsonldV}, \text{rdfcV}, \text{contextDigests}, \\
&\quad \text{maxContextUrls}, \text{maxContextDepth}, \\
&\quad \text{maxBytes}, \text{maxDecodeDepth}, \\
&\quad \text{maxDecodedItems}, \text{maxQuads}, \\
&\quad \text{maxExpandedTerms}, \text{maxExpansionDepth}, \\
&\quad \text{maxBlankNodes}, \text{maxNDegreeCalls}, \\
&\quad \text{maxPermutationBranches}, \text{maxIssuerEntries}, \\
&\quad \text{maxPathUnits}, \text{maxIntermediateUnits} \rangle.
\end{aligned} \tag{21}$$

jsonldV pins the 16 July 2020 JSON-LD 1.1 Processing Algorithms and API Recommendation: `processingMode=json-ld-1.1`, its Context Processing and Expansion algorithms, `ordered=true`, `rdfDirection=null`, a signed absolute base IRI, and `produceGener-`

Table 15: Exact counter increments. An occurrence is counted even if later deduplicated; tests happen before the event that would exceed its bound.

Counter	Increment or depth event	Counter	Increment or depth event
inputOctets	Before consuming each primary or supplied-context octet; duplicate packaged objects count again.	decodeDepth	Entering a JSON array/object; root depth is 1 and exit decrements it.
decodedItems	Decoder emits each member, array element, or scalar occurrence in the primary or each supplied context object.	expandedTerms	JSON-LD expansion emits each node, property, or value occurrence.
contextUrls	Before each JSON-LD Context Processing URL attempt after base resolution; repeated URLs, repeated references, and cache hits count.	contextDepth	Entering each logical nested Context Processing call; root is 1, cache use changes neither entry nor exit.
expansionDepth	Entering a recursive JSON-LD expansion call; root is 1 and exit decrements it.	quads	RDF conversion emits each quad occurrence, before set deduplication.
blankNodes	First allocation or capture of each distinct blank-node identifier.	nDegreeCalls	Entry to each RDFC Hash N-Degree Quads invocation.
permutationBranches	Before exploring each RDFC permutation candidate.	issuerEntries	Before a logical issuer state first becomes reachable, charge every mapping in that state; a fresh insertion charges one and a clone charges its full inherited mapping cardinality. Equal mappings in distinct issuers or clones count again; structural sharing gives no discount.
pathUnits	Each identifier, position, or digest token appended to an N-degree path.	intermediateUnits	Simultaneously with each increment of decodedItems, expandedTerms, quads, permutationBranches, issuerEntries, or pathUnits; hence it is their running sum.

alizedRdf=false. Input is UTF-8 application/ld+json containing one JSON object or array; HTML extraction and ambient bases are forbidden [17, 20]. Remote contexts are digest-pinned package objects, never network-fetched. Each reference resolves against the signed base to one exact URL—digest manifest entry; redirects, negotiation, and unlisted URLs are rejected. Primary and all supplied context bytes are charged at RawScreen and decoded at Decode. Duplicate supplied objects are charged separately; repeated references and cache hits do not recharge bytes but do count logical context-processing attempts. *rdfcV* pins RDFC-1.0 [18]. The path is JSON decode, Context Processing/Expansion, RDF emission, then RDFC canonicalization. Table 15 defines mathematical non-negative counters, not implementation allocations. All cumulative counters reset once per proposal and sum occurrences across its datasets; they never reset per recursive invocation. Depth counters are active recursion gauges. *inputOctets* belongs to RawScreen, *decodeDepth/decodedItems* to Decode, and the remaining counters to Canonicalize.

Every bound is tested before its event; context URL/depth exhaustion terminates at Canonicalize before lookup or recursive entry, with CanonicalizationLimit and no partial candidate. For a clone, the full inherited charge is tested before the clone becomes reachable; for insertion, the unit charge is tested before insertion. Either failure terminates at Canonicalize with the same CanonicalizationLimit receipt and no partial issuer state. Crossing any other limit terminates at its governing RawScreen, Decode, or Canonicalize stage with CanonicalizationLimit, produces no partial candidate, and binds d_{raw} rather than a nonexistent d_r . RDFC-1.0 separately identifies adversarially expensive graph structure as a denial-of-service risk [18]; this is *canonicalization complexity poisoning*, not a semantic claim that the resulting graph is false. A local wall-clock, memory, or process watchdog may stop work earlier, but that event is a transient operational failure with no durable disposition unless the implementation can translate it into the deterministic counters above.

D.2 Conditional assumptions

All safety claims are conditional on:

- A1 Mediation.** No bypass credential or API can advance an authoritative head or protocol index.
- A2 Encoding and cryptography.** Canonical encodings, typed hash domains, and signature schemes have their stated security and interoperability properties.
- A3 Key custody.** Proposal, allocator, evaluator, preparation-service, candidate-seal, and kernel keys are protected and restricted to their declared principals and scopes.
- A4 Atomic serialization.** The complete accepted unit, lifecycle changes, and reclamation changes serialize over every named mutable key and are durably all-or-none.
- A5 Complete context.** Policy, evaluator, authority, revocation, dependency, allocation, and lifecycle versions are locally lockable or transactionally revalidated.
- A6 Trusted time.** Activation obtains a conservative commit-time interval or an equivalent storage-enforced deadline.
- A7 Lifecycle order.** Branch creation, epochs, handoff, and recovery share one order observed by writers.
- A8 Non-recycled scopes.** Authenticated allocators issue monotone proposal/effect identifiers; retirement watermarks advance before covered rows are deleted.
- A9 Verification objects.** Replay and inclusion are claimed only when their required objects, interpreters, keys, and certified-state proofs are available.

Availability is not assumed. Receipt structural authenticity and kernel attestation use A1–A3; independent replay additionally uses the relevant A5–A6 objects and A9; durable inclusion additionally uses A4, A8, and A9.

D.3 Executable Model and Artifact Specification

The formal state-space exploration of Section 5 is provided as an executable Python artifact (`artifacts/bounded_model.py`). The artifact imports only standard library modules (`collections`, `dataclasses`, `typing`) and executes an exhaustive breadth-first search (BFS) state exploration over the finite protocol state space.

The executable model verifies that across all 2,808,230 reachable states and 5,526,474 state-changing transitions: (1) no encoded invariant is violated; (2) every named protocol coverage witness is reached; and (3) all terminal dispositions, 17-stage check sequences, pre-state authority rules, writer fencing, and forward restoration post-conditions hold. Reproduction instructions and SHA-256 integrity digests are documented in `artifacts/README.md`.