The Synchronic Web

Thien-Nam Dinh
Sandia National Labs
thidinh@sandia.gov

Nicholas Pattengale
Sandia National Labs
ndpatte@sandia.gov

Mitch Negus
Sandia National Labs
mnegus@sandia.gov

Abstract

Networked information is easy to publish and difficult to keep intelligible over time. Content changes, software evolves, institutions disappear, and evidence usually remains bound to the operator that currently serves it. The Synchronic Web addresses this fragility by making digital state portable—with its integrity, history, and means of interpretation intact—across independently governed systems. A minimal integrity substrate supports verifiable partial possession; self-encoding objects bind durable definitions to the state they interpret; and reciprocal bridges incorporate foreign heads into later local commits, entangling otherwise independent histories. Journal-relative paths project this structure as a plural Web of resources reached from chosen viewpoints. The architecture builds on secure timeline entanglement and combines it with partial state, executable interpretation, and local authority. The result is a foundation for preserving and comparing evidence across institutional and temporal boundaries while keeping trust decisions with each journal.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction and Motivation

1.1 Durable Memory in a Mutable Web

Networked information is easy to publish and difficult to keep intelligible over time. A URL can remain stable while its content, operator, policy, software, and meaning all change. The database behind it usually carries richer context, yet that context travels poorly: a recipient receives a representation while the operator retains the history that explains it. When the service closes or the relationship changes, evidence that seemed durable may collapse into screenshots, exports, and institutional memory. Version control, archives, local-first software, and user-controlled data each recover part of this continuity [32, 44]. Their value points toward a broader goal: a resource should remain identifiable across versions, interpretable beyond its current application, and citable from another institution's history. Corrections and disagreements should accumulate as later evidence. A record gains social value when several parties can preserve it from their own viewpoints and still compare what they retained. This need becomes acute wherever decisions outlive the systems that informed them: scientific results, production records, public claims, personal identity, or an agent's sources and corrections. AI systems make the scale visible because they combine more context than a person can inspect directly [37, 55, 3, 23]. The architectural question reaches further than agents: how can independently operated systems share durable evidence while preserving their own authority over storage, policy, and interpretation?

1.2 Independent Control and Shared Evidence

A plural Web needs both local control and structural connection. Local control lets each entity choose what to retain, expose, execute, and accept. Structural connection lets evidence cross institutional boundaries with its identity and historical context preserved. Together they create common evidence structures beneath independent roots: local stores gain compositional reach, and naming, ordering, and governance remain distributed among their operators. The security model begins with an explicitly accepted commitment and separates four responsibilities:

- **Structural integrity.** Collision-resistant hashing binds available material to the commitment, canonical encoding makes hashed and signed forms reproducible, and signatures associate a key with a committed head.
- **Adversarial behavior.** Storage services, journals, intermediaries, and networks may omit, replay, delay, substitute, or selectively present material; witnesses retain and compare signed views.
- **Operational assurance.** Replication and institutions sustain availability, disclosure policy and secure transport protect confidentiality, and domain evidence supports judgments about truth.
- **Local trust.** The client, evaluator, integrity primitives, runtime configuration, operating-system boundary, and key store form the trusted base from which local policy interprets keys, paths, and operations.

Compromise has a temporal shape: heads witnessed before key compromise remain evidence of the earlier branch, while a compromised key can authorize later branches until peers recognize the event. The implementation separates a stable randomized journal identifier from the active signing key: bridges and preapprovals bind the identity commitment, heads carry the current signing key, and predecessor-authorized transitions establish key continuity from each receiver's retained checkpoint.

This supports routine rotation without making catastrophic recovery automatic; loss or compromise of the accepted signing authority still requires an explicit governance decision. The same evidence may support different decisions at different journals because each receiving journal evaluates trust from its own state.

1.3 Portable State, Plural Trust

The Synchronic Web makes digital state portable—with its integrity, history, and means of interpretation intact—across independently governed systems. Its compact thesis is *self-encoding, partially knowable, historically composable state with verifiable integrity*:

- **Integrity** gives a logical state one stable identity.
- **Partial knowledge** lets observers retain different verifiable regions of that state.
- **History** orders successive states and composes selected foreign heads into later local commits.
- **Self-encoding structure** carries durable definitions alongside the data they interpret.

A *journal* is the software an entity runs together with all data persisted on it; it advances a local history and supplies one operational viewpoint. A *resource* is a path-addressed value or object interpreted through that viewpoint. A *bridge* is a reciprocal relationship through which journals retain verified heads for one another. A specialized client combines these layers so that applications can read resources, follow history, invoke objects, and traverse relationships through ordinary interactions. Figure 1 shows this network: each journal contains integrity-protected state and advances independently; sparse bridges preserve foreign heads and create cross-timeline ordering when those heads enter later commits; and the highlighted journal supplies the selected viewpoint from which paths, identities, routes, and policy acquire meaning. Trust therefore remains plural: structural verification answers whether evidence agrees with a commitment, signatures identify the key that authorized a head, historical routes connect that key to a journal known by the receiver, and authorization applies the receiver's rules to the resulting principal. The architecture makes these steps composable and inspectable while leaving the final decision with the relying journal.

1.4 Scope and Roadmap

This paper is an architectural design and rationale. The implemented core includes the integrity substrate, partial structures, self-encoding objects, rooted paths, local histories, authorization machinery, stable journal identity and signing-key rotation, reciprocal signed-head exchange, exact committed-object routing, direct signed invocation, terminal audience checks and route-bound endpoint selection, terminal-local key-index authorization, and origin verification of historical results. Multi-node HTTP and browser acceptance covers direct and multi-hop application access, historical resolution, proof retention, bridge lifecycle, and denial behavior. Section 2 develops verifiable partial state; Section 3 turns state into history and entangles independent histories; Section 4 places durable interpretation inside committed structure; and Section 5 projects these layers as a journal-relative Web. Sections 6–8 test the design through applications, map its intellectual lineage, and state its assurance boundaries, costs, and route toward specification. The original Synchronic Web proposed a globally synchronized notary architecture [19]. The present design retains the name and broad aspiration while beginning from independent journals, partially retained state, and sparse historical composition. Secure timelines and timeline entanglement supply the direct lineage for cross-history ordering [41]; the architecture developed here asks what becomes possible when those histories carry ordinary resources and the definitions needed to interpret them.

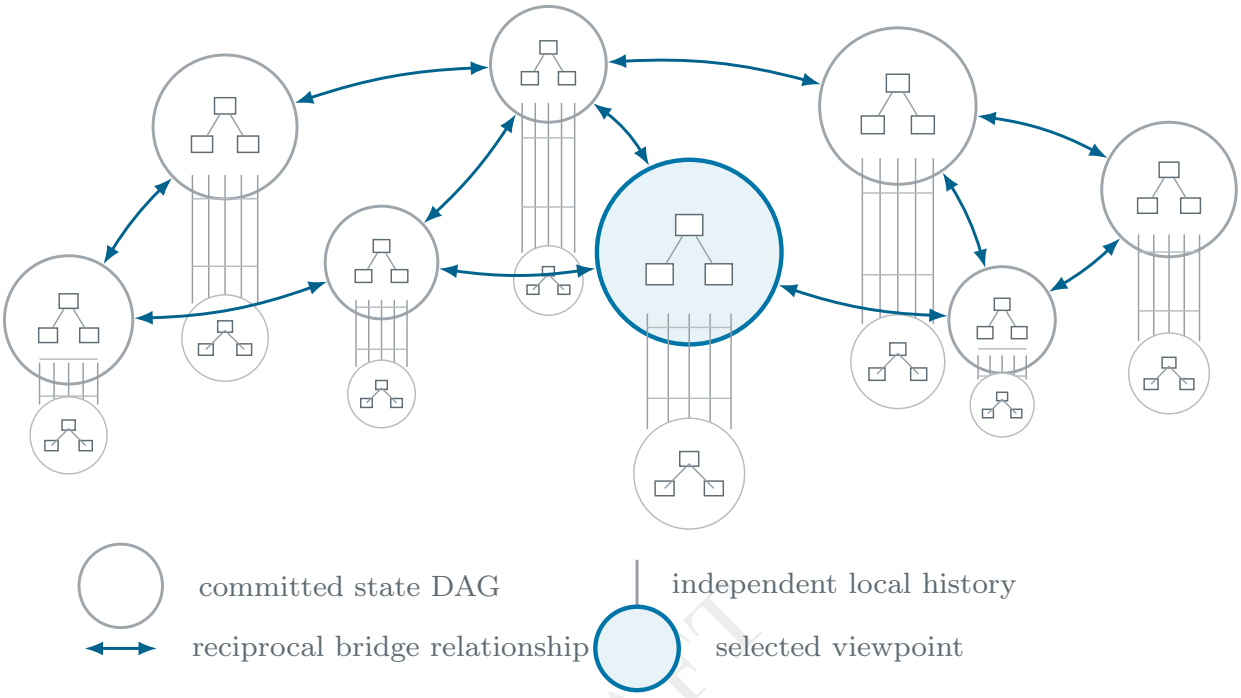

Figure 1: The Synchronic Web

2 Integrity-Protected State

2.1 The Value of Verifiable State

Versioned systems preserve the path by which information changed. Git turns a working tree into a succession of content-addressed snapshots. Append-only databases and blockchains bind later entries to earlier ones. Transparency systems publish authenticated views that clients and witnesses can compare [24, 35, 47]. Their common achievement is a durable answer to a simple question: does the evidence now presented belong to the state that was previously committed? That answer changes the character of a record. A mutable database row asks the operator for its current value. A committed state can also be held by a recipient, cited by digest, compared with another copy, and revisited after the operator changes its presentation. History becomes portable evidence that recipients can preserve alongside the service. This remains useful even when the record is incomplete, disputed, or false: integrity preserves the object of later analysis. Most authenticated structures assume that a verifier receives a query result and a compact proof against a known root [45, 48]. The Synchronic Web extends this idea into a persistent representation of partial knowledge. A recipient can retain one region of a state, discard another, and later combine compatible evidence while referring throughout to the same logical commitment. That capability begins with a deliberately small structural model.

2.2 A Minimal Verifiable Structure

The native substrate is an integrity-verifiable binary data-structure DAG in which immutable substructures may be shared by many parents. A *logical digest* names recursively defined content or structure, while a *local handle* retrieves the representation available in one store. This separation lets the same logical state use different persistence layouts and lets one handle retrieve either complete structure or a compact record of its digest. Maps, objects, histories, and resources grow from five

substrate forms, shown in figure 2:

- **Root:** a stable local name mapped to a node handle.
- **Branch:** two child handles with a logical digest derived from their digests.
- **Leaf:** raw bytes stored under a double-hashed handle and logical content digest.
- **Stump:** one retained digest for structure whose internal representation is unavailable.
- **Null:** the distinguished all-zero word denoting empty structure.

A digest commits recursively to everything beneath it. Changing one leaf produces new digests along every dependent path to the root; the old node digest continues to identify the old state, while advancing the durable root handle selects its successor. The stump gives missing structure a precise place in this model by preserving the digest expected at one position and ending traversal at an identified boundary. Null records actual emptiness, an unknown value records the absence of a usable commitment, and a stump records committed structure awaiting evidence. Local mutation is therefore movement among immutable logical states, and partial possession is a precise statement about which regions of one such state are currently known.

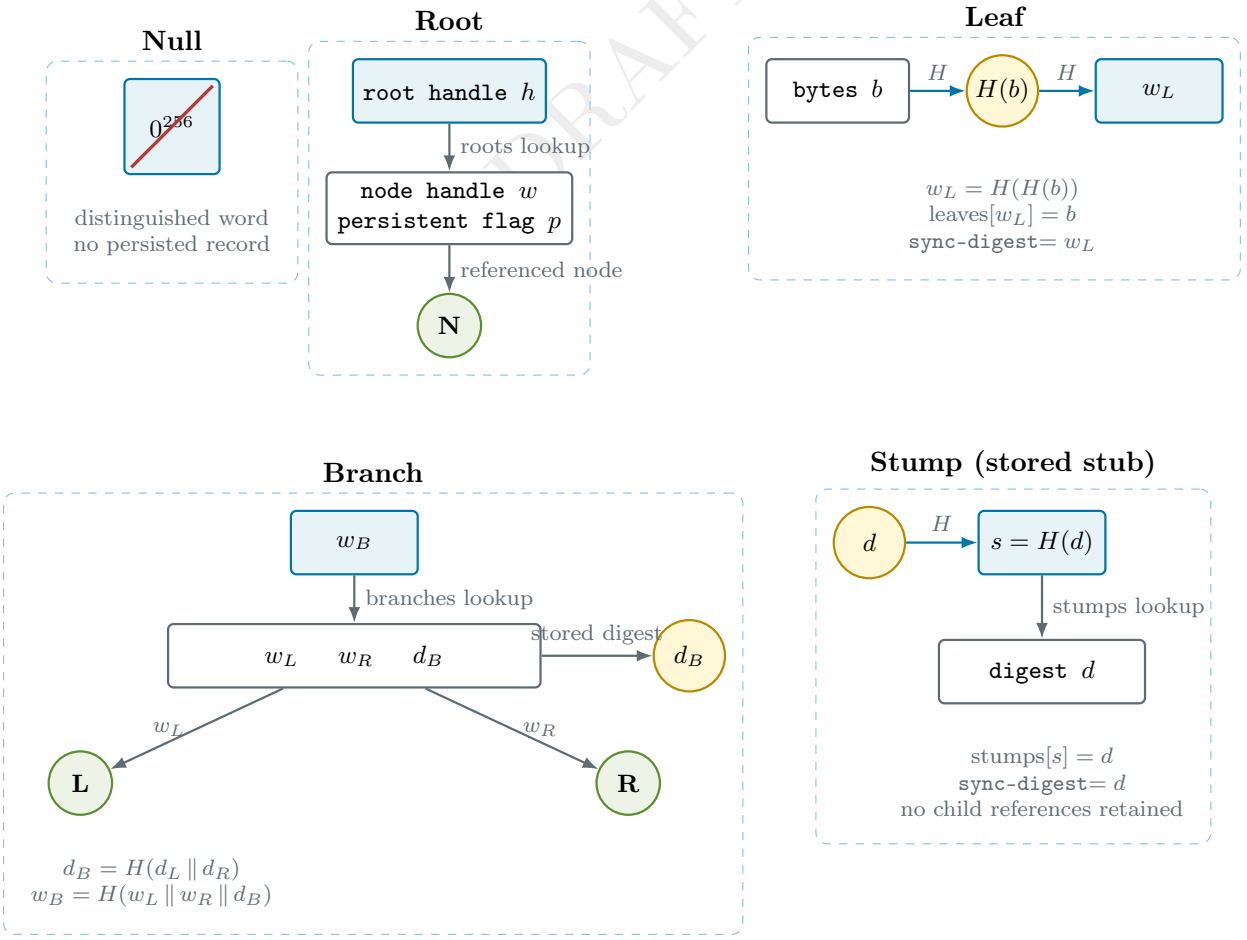

Figure 2: Persisted DAG Node Forms

2.3 Proof Under Partial Possession

A slice follows selected semantic paths through a complete state and replaces unrelated material with stumps; because the result retains the original root digest, every included resource remains bound to the complete state from which it came. Object-aware slicing follows documented protocols as well as physical DAG edges, extending the economy of compact Merkle multiproofs [57]. Pruning applies the same transformation to material already held locally, shrinking possession while the logical state remains fixed; deletion instead constructs a successor state in which the resource is absent, allowing a journal to release storage while preserving an exact historical commitment [15]. Merge moves in the other direction by replacing a stump with matching structure or recursively combining compatible branches; a digest mismatch aborts the operation and leaves conflict resolution to domain policy. Serialization then carries node kinds, bytes, and retained digests across a transport boundary so a client can reconstruct the same logical root, retain one proof, and accumulate neighboring evidence through later responses.

Algorithm 1 Deep Slice Along Object Path

Require: Object node x and nonempty path p
Ensure: The selected path remains available and the logical digest is unchanged

function DEEPSLICE(x, p)
  $o \leftarrow$ EVALUATEOBJECT(x)
  $k \leftarrow$ FIRST(p)
  if $|p| = 1$ **then**
    o.SLICE(k)
    return NODE(o)
  end if
  $c \leftarrow o$.GET(k)
  $d \leftarrow$ DIGEST(c)
  $q \leftarrow$ REST(p)
  $c' \leftarrow$ DEEPSLICE(c, q)
  if DIGEST(c') $\neq d$ **then**
    FAIL("slice changed child digest")
  end if
  o.SET(k, c')
  o.SLICE(k) ▷ discard siblings not needed for k
  return NODE(o)
end function

2.4 Integrity as Infrastructure

Partial possession separates auditability from bulk custody. An auditor can retain the report, definitions, path, and ancestor evidence relevant to one decision while the source keeps operational data in its native system; another institution can preserve a complementary slice, and their copies remain comparable because both resolve to the same commitment. Availability then grows from replication, pinning, media renewal, and operating institutions: a digest may outlive every full copy of the bytes it names, integrity makes each recovered copy checkable, and retention determines whether recovery remains possible. Selective transfer, citation, archival sampling, and cross-organizational inspection all follow from distributing these responsibilities among parties with different interests and storage budgets. A small substrate concentrates trust in construction, hashing, persistence,

Algorithm 2 Deep Prune Along Object Path

Require: Object node x and nonempty path p
Ensure: Evidence at the selected path is removed while the logical digest remains fixed

```
1: function DeepPrune(x, p)
2:     o ← EvaluateObject(x)
3:     k ← First(p)
4:     if |p| = 1 then
5:         o.Prune(k)
6:         return Node(o)
7:     end if
8:     c ← o.Get(k)
9:     if c is not an available pair node then
10:        return Node(o)
11:    end if
12:    d ← Digest(c)
13:    q ← Rest(p)
14:    c′ ← DeepPrune(c, q)
15:    if Digest(c′) ≠ d then
16:        Fail("prune changed child digest")
17:    end if
18:    if c′ is a stub then
19:        o.Prune(k)
20:    else
21:        o.Set(k, c′)
22:    end if
23:    return Node(o)
24: end function
```

Algorithm 3 Merge Compatible Partial Structures

Require: Partial structures a and b
Ensure: The result contains the union of available evidence and retains the common digest

```
1: function MergePartial(a, b)
2:     if Digest(a) ≠ Digest(b) then
3:         Fail("cannot merge different digests")
4:     end if
5:     if a is null then
6:         return a
7:     else if a is a byte-vector leaf then
8:         return a
9:     else if a is a stub then
10:        return b
11:    else if b is a stub then
12:        return a
13:    else
14:        a_L ← Left(a); b_L ← Left(b)
15:        a_R ← Right(a); b_R ← Right(b)
16:        l ← MergePartial(a_L, b_L)
17:        r ← MergePartial(a_R, b_R)
18:        return Pair(l, r)
19:    end if
20: end function
```


and controlled evaluation while rich semantics remain visible and historically evolvable in stored objects above it. Verification still needs bounds on encoded size, node count, recursion, allocation, and traversal because structurally valid evidence may be adversarially deep or broad. Within those bounds, the substrate adds verifiable internal partiality to content addressing: structure can be selectively possessed and later restored under one identity. Once roots advance through signed committed steps, that identity becomes durable evidence about change; Section 3 develops this succession and shows how independent histories can become evidence for one another.

3 History and Entanglement

3.1 Integrity Across Boundaries

Organizations already exchange information through shared clouds, data lakes, APIs, replicated databases, signed documents, and ledgers. Each arrangement chooses a center of gravity. A shared platform centralizes custody and policy. Replication creates additional copies while usually aiming at one application state. Signed files travel independently but carry limited context about succession. Webs of trust connect keys and attestations while leaving application data to other systems. Blockchain bridges transfer claims between consensus domains and inherit the complexity of both sides. Many important relationships require a different shape. A manufacturer and buyer, laboratory and regulator, publisher and archive, or two neighboring communities may each need its own history. They still need evidence that a foreign state existed before a later local event, and they may need that evidence after one participant becomes unreachable. The desired result connects their histories while each institution retains its own governance. Secure timelines provide the starting

point. A service repeatedly commits its state into a one-way succession, allowing later authenticators to establish the order of earlier states. Timeline entanglement carries signed authenticators across service boundaries and incorporates them into later foreign authenticators [41]. This creates portable temporal mappings: the past of one service becomes an input to the future of another. Synchronic Web bridges apply that principle to histories whose states contain ordinary resources and executable interpretation.

3.2 A Bridge Between Histories

A journal first turns local mutation into a committed step. Applications accumulate a proposed state, inspect it, and then ask the journal to advance; the step compares the proposal with the previous head, records declared mutation metadata and explicit state such as time and retained remote heads, incorporates eligible evidence, constructs a canonical pre-signature view, signs its digest, and advances the durable root. An authorized signing-key transition also requires a step even when application state is unchanged. Time, remote responses, randomness, and operator choices enter as explicit values under the normative model, while applications needing exhaustive provenance store durable event objects or batch descriptions. Figure 3 shows the logical result: application state, declared transition information, retained foreign heads, time evidence, identity and key-lineage material, and signatures share one integrity-protected tree. Algorithm 4 exposes the signing sequence by inserting the stable identity commitment, current rotation index and optional transition, journal and interface descriptor fields, blanking both generic and journal-namespaced public-key and signature fields to create the canonical view, signing its digest, and finally filling those signing fields.

A bridge is a reciprocal signed-head-retention relationship. The journal that creates it becomes the sole ongoing synchronization initiator, and the acceptor returns its own signed head in the same replay-accepted exchange; this scheduling role supports private or intermittently reachable journals through outbound connectivity and keeps commit cadence under local control. As shown in figure 4, the initiator sends its stable identity and head signed by its current journal key, and the acceptor verifies nonempty history, the identity bound to the relationship, any predecessor-authorized key transition from its retained checkpoint, the committed descriptor endpoint, monotonic index, and predecessor continuity before staging the new head and returning corresponding evidence. The initiator performs the same checks locally. The implementation rejects older heads, same-index forks, unrecognized identities or key transitions, and newer heads that omit the retained predecessor digest. Acceptance records the foreign head as staged local state and returns an `accepted-index`; incorporation occurs only when a later local `step!` includes that head in the receiving journal's committed tree. If B incorporates A's head A_i into B_j, then A_i precedes B_j, and B_j together with an inclusion proof becomes evidence that B incorporated A. Each journal may advance immediately, slowly, or after administrative review, so repeated exchange creates a weave of coarse ordering relations while the journals retain distinct sequences through disagreement. Re-establishing a locally matching relationship may request a signed, read-only establishment check: only an exact reciprocal descriptor, identity, and pair of retained indexes returns `established?`; any difference falls back to the full exchange that advances or repairs the relationship. Ordinary scheduled synchronization continues to exchange and apply heads.

3.3 Retention and Practicality

Historical identity and historical possession are separate choices. A journal may retain recent heads in full while permanent history keeps chain continuity, signatures, selected proofs, and explicit pins;

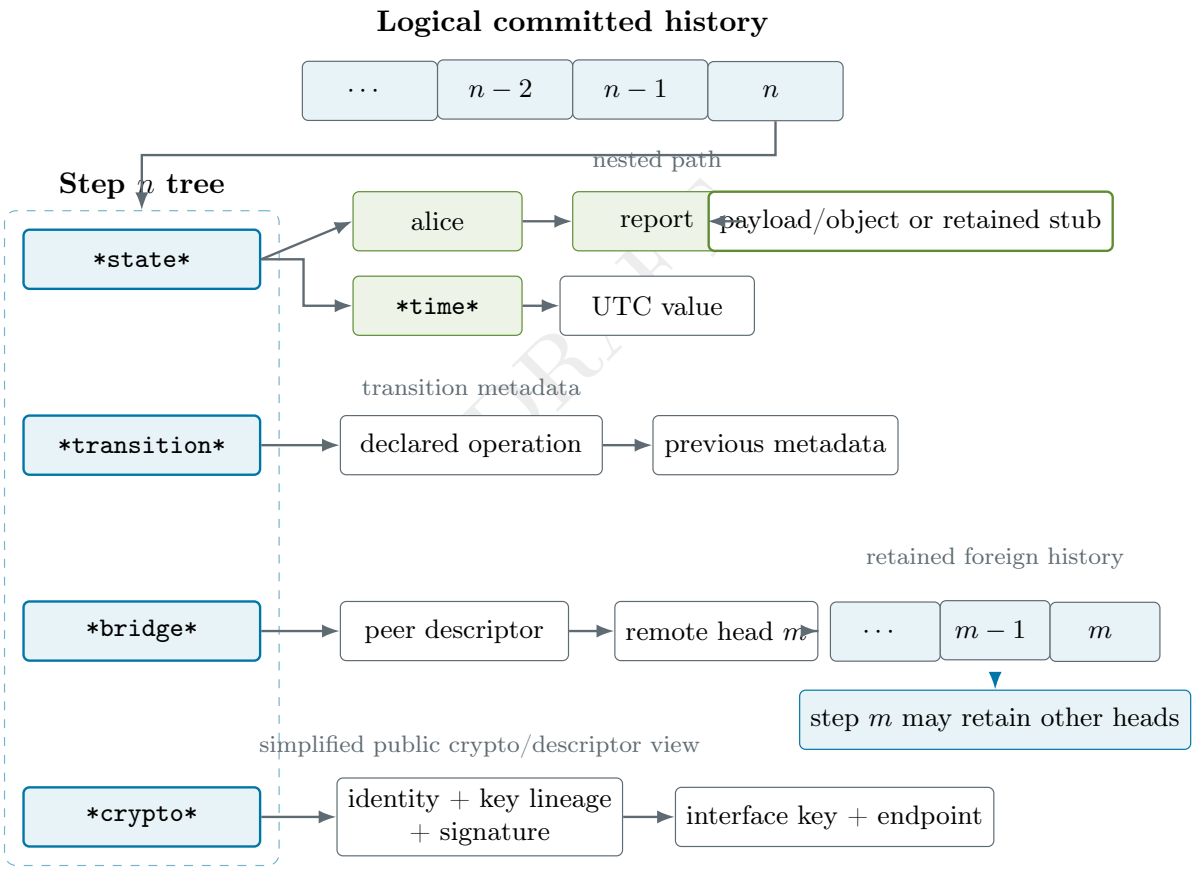

Figure 3: Committed Step Anatomy

Algorithm 4 Commit Journal Step

Require: Ledger L, Unix time t, signing public key p_k, signing secret key s_k
Ensure: Either no step is needed or a signed step is appended and retention is updated

function CommitStep(L, t, p_k, s_k)
    ▷ Comparable state omits transition and cryptographic metadata.
    $(S, P, T) \leftarrow$ EvaluateLedgerFields(L)
    $r \leftarrow$ PendingKeyTransition(L)
    $k_0 \leftarrow$ AcceptedSigningKey(L)
    rotate $\leftarrow (p_k \neq k_0)$
    if *rotate* **then**
        VerifyAuthorizedTransition($L, r, k_0, p_k, |P|$)
    else if r is present **then**
        Fail("pending transition does not match signing key")
    end if
    $d_{cur} \leftarrow$ ComparableStateDigest(S)
    if Size(P) $= 0$ **then**
        $d_{prev} \leftarrow$ Digest(*null*)
    else
        $h \leftarrow$ Get($P, -1$)
        $d_{prev} \leftarrow$ ComparableStateDigest(h)
    end if
    if $d_{cur} = d_{prev}$ and not *rotate* **then**
        return Size(P)
    end if
    $u \leftarrow$ UTC(t)
    S.RecordTransition([`*state*`, `*time*`], u)
    P.Push(Node(S))
    S.Delete(`*transition*`)
    $H \leftarrow P[-1]$
    $H \leftarrow$ EmbedHeadMetadata($H, L.identity, L.name, L.endpoint, L.interfaceKey, r$)
    $C \leftarrow$ BlankSigningFields(H) ▷ Generic and journal-namespaced key/signature fields.
    $d_s \leftarrow$ Digest(C)
    $\sigma \leftarrow$ Sign(s_k, d_s)
    $P[-1] \leftarrow$ FillSigningFields(C, p_k, σ)
    if *rotate* **then**
        CommitKeyTransition(L, r, p_k)
    end if
    T.Push($P[-1]$)
    $q_t \leftarrow$ DeepSlice(Node(T), `(-1 (*state* *time*))`)
    if $L.window$ is set and Size(T) $> L.window$ **then**
        T.Prune($-(L.window + 1)$)
    end if
    $q_p \leftarrow$ DeepPrune(Node(P), `(-1 (*state*))`)
    $L.stage \leftarrow$ Node(S)
    $L.permanent \leftarrow$ MergePartial(q_t, q_p)
    $L.temporary \leftarrow$ Node(T)
    return Size(P)
end function

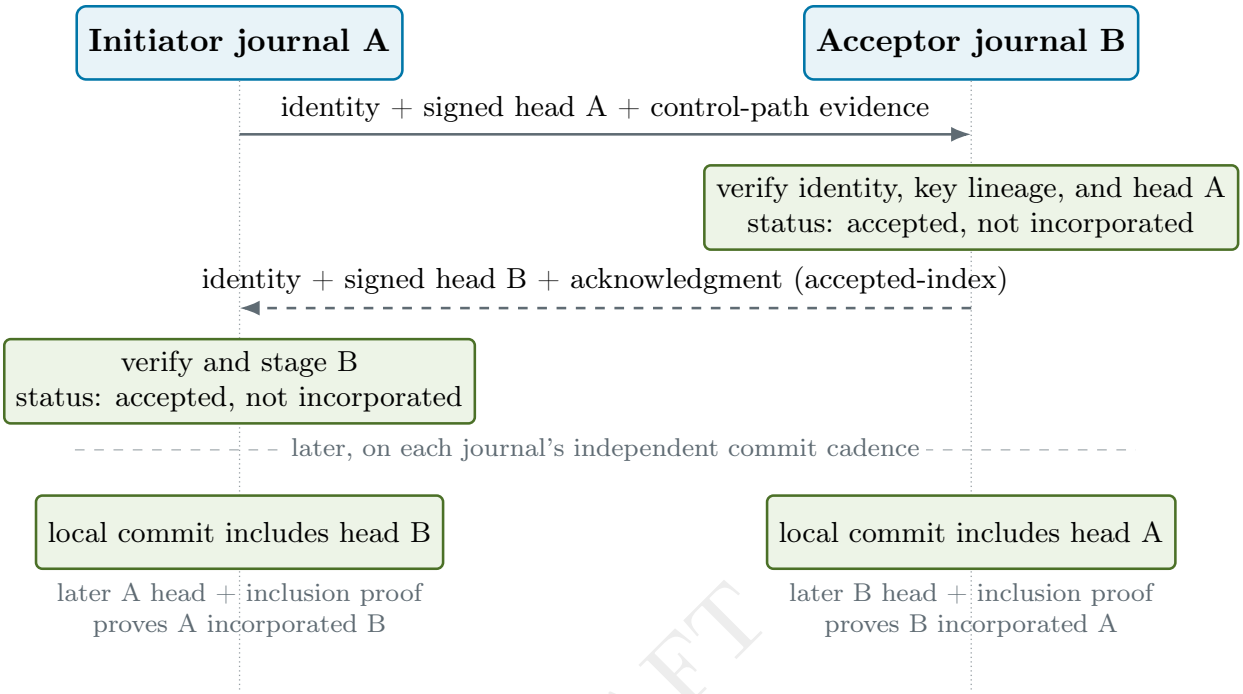

Figure 4: Bidirectional Head Exchange

pruning replaces released material with stumps, pinning merges a chosen slice into the evidence held for a historical head, and unpinning releases that local obligation. A finite recent-history window is therefore a possession policy rather than an archival promise: shrinking it immediately releases newly excluded unpinned material, while later widening changes future retention without reconstructing what was released. Preservation becomes purpose-driven: a buyer can pin an inspection report and its definitions, an archive can retain a complete site capture, and a witness can store signed heads and incorporation receipts. Each role pays for the evidence it values, immutable sharing reduces duplication among related states, and replication and media renewal turn local retention into long-term availability. Bridge operation exposes a similar set of practical controls. Synchronization cadence determines temporal resolution and network cost, while the signed-head and control-path protocol determines the public material exchanged. Locked-down journals may preapprove a stable journal identity; open journals may automatically accept valid creator-initiated relationships whose creator retains initiation responsibility. Repeated synchronization safely confirms or advances acceptance, local stepping remains independent of remote traffic, and a later signed head with an inclusion proof establishes incorporation.

3.4 History at Network Scale

Sparse bridges turn entanglement into a network property. A foreign head may itself contain other incorporated heads, so historical reachability composes recursively: from one selected journal, a verifier can follow committed checkpoints across several relationships while preserving the path by which each piece of evidence became known. The result is a Web of historical viewpoints, each retaining its own timeline. At this scale witnesses become central because a journal can sign distinct branches for isolated peers; when witnesses compare heads or later histories incorporate conflicting views, the signed divergence becomes durable evidence. Transparency overlays and gossip develop this comparison as a general accountability technique [11]; entangled journals contribute the retained

cross-history checkpoints, signatures establish authorship of each branch, and witness topology determines how quickly divergence becomes visible. Figure 5 captures the two forms of change now in play. Along one timeline, the normative explicit-input model produces a deterministic successor. Across timelines, a foreign head enters another journal's committed state and establishes an ordering edge. Together they allow computation to remain local while evidence becomes historically connected. A large network will evolve while these histories accumulate: object formats, authorization rules, query languages, and application protocols will change on different schedules, and historical evidence remains most useful when the definitions needed to interpret it travel with the state itself. This change-management problem leads to the next layer, self-encoding structure.

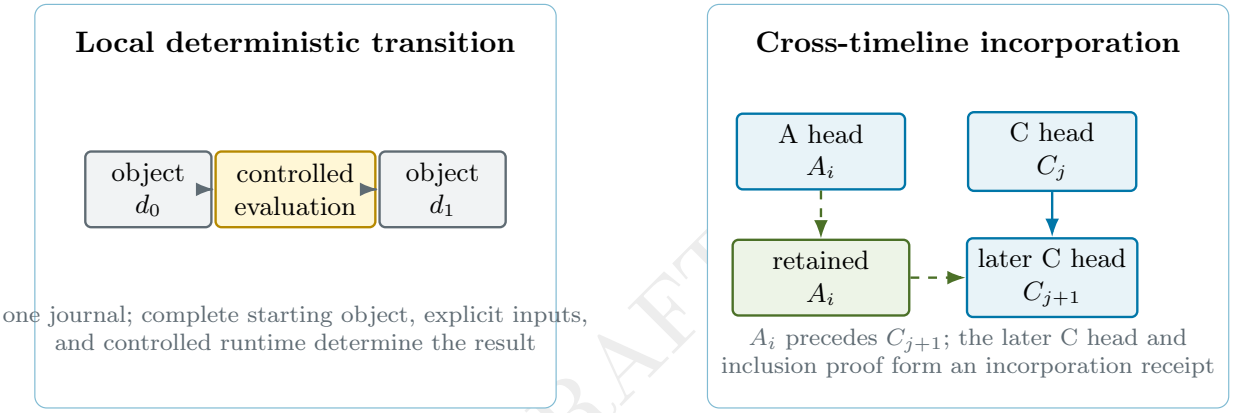

Figure 5: Computation and Entangled State

4 Self-Encoding Structure

4.1 Code at Web Scale

A durable network must survive software change as well as storage failure. Data written today may be read by another organization years later, after schemas, libraries, runtimes, and deployment practices have moved on; preserving bytes answers only the first half of the archival question, because the reader also needs a durable account of how those bytes were structured and what operations gave them meaning. The Web already offers several approaches: JavaScript carries behavior to a widely available runtime, Protocol Buffers and similar schema-driven encodings define stable wire forms and evolution rules whose schemas supply interpretation beyond each payload, package managers and containers capture dependencies, and reproducible-build and software-transparency systems make artifacts traceable to declared sources and environments [53]. Each binds a different portion of the path from stored bits to behavior. At Web scale, change management crosses administrative boundaries: a recipient may accept data while declining the producer's newest software, or retain an old object long after its originating service has migrated. Reproducibility also demands explicit treatment of time, randomness, network responses, and operator choices. Self-encoding structure answers these pressures by placing a changeable definition inside the same commitment as the state it interprets, creating a historical pair that can travel, be sliced, and be cited together. The runtime becomes a smaller shared foundation, application semantics become ordinary versioned material, and transitions can expose the environmental inputs on which their results depend.

4.2 Lisp as Durable Structure

Lisp gives code and data one inspectable representation: a method definition, query, path, transition, or configuration is an expression that can be hashed, stored, transmitted, and evaluated. Homoiconicity turns semantic bootstrapping into structural composition because the network can preserve the form describing an operation with the same substrate that preserves its inputs. The implemented runtime uses s7 Scheme inside the journal; native primitives provide construction, hashing, persistence, slicing, network preparation, and controlled evaluation, while Scheme builds maps, chains, ledgers, authorization, and object protocols above them. This follows the tier-unifying intuition of single-language Web systems [14], preserving explicit boundaries while reducing translations between durable state and the logic that uses it. The bootstrap terminates in native storage and evaluator semantics, which define how expressions are read, nodes are hashed, and capabilities are exposed. Keeping this foundation small makes it practical to inspect and reproduce while every higher layer can evolve as identified structure in journal history. A shared language still permits heterogeneous protocols: objects may implement different methods and evolve under different authorities while remaining transportable through the same node and evaluator model, with documented public behavior providing compatibility across class systems. The implemented `sync-let` boundary copies inert inputs into a reduced allowlisted environment, excludes ambient journal, network, time, randomness, and host authority, and copies inert results back to the caller; finer per-object capability assignment is an optional extension rather than a prerequisite for portable execution.

Capability confinement and evaluator isolation address different failures. Current Linux images and binaries use `wasmer-nofuel` exclusively and fail closed unless a supplied target-specific AOT matches its configured SHA-256. Each evaluation and request graph runs in a fresh instance without ambient WASI authority; the native parent retains persistence, networking, capabilities, commit, and retry. Attestation binds the exact AOT and target, but native AOT bytes are not portable. Accepted Scheme/API behavior and the `sync-node-v2` durable representation remain exact.

4.3 Self-Encoding Objects

A standard object is a node with executable definition on the left and durable state on the right; its digest commits to both components and to their association. Evaluating that node with `sync-eval` produces a transient callable object: a zero-argument call returns its current node, documented method dispatch exposes behavior, and object-specific paths expose state according to the object's protocol. Figure 6 follows one transition as definition D_0 and state S_0 enter a controlled runtime with explicit arguments and prepared environmental values, producing a candidate with revised definition D_1 and successor state S_1. An authorized boundary chooses whether to persist that result, while the starting object remains independently identifiable and semantic change becomes historical succession.

Live mutation and durable mutation meet at explicit persistence: a method may update the live object's state, while the journal records the change after extracting the new node and storing it into its parent or root. Nested mutation rebuilds each containing object along the return path, keeping every predecessor available by digest and making the durable boundary visible in code. Objects compose through public protocols: deep reads traverse semantic boundaries, deep mutations rebuild ancestors, deep slicing and pruning ask each object to retain evidence appropriate to its layout, and deep merge restores compatible partial material under matching commitments. Table 1 summarizes this implemented family. Portable definitions arrive from parties who may use different software and policy, so the caller places shared computation inside `sync-let` before loading an object with

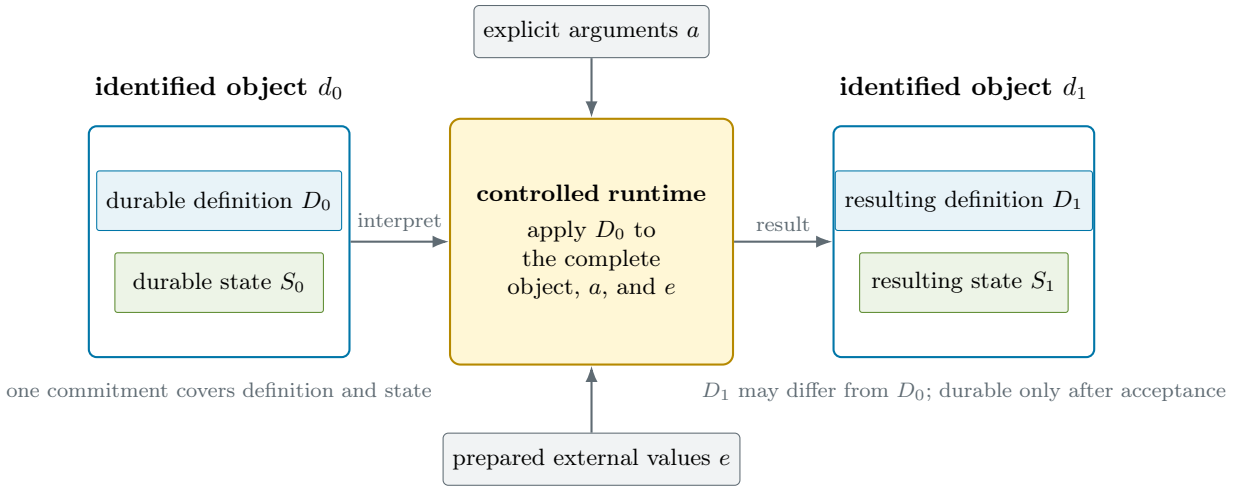

Figure 6: Self-Encoding Object Transition

`sync-eval`; privileged host orchestration obtains time, remote responses, and other environmental values first and passes them as copied ordinary data; and authorization and explicit storage turn the candidate result into history.

Operation	Signature	Behavior
`deep-get`	`(deep-get object path)`	Traverse nested public `get` methods and return the value at the path.
`deep-set!`	`(deep-set! object path value)`	Set the terminal value and rebuild each containing object.
`deep-slice!`	`(deep-slice! object path)`	Retain evidence along the path while preserving the logical digest.
`deep-prune!`	`(deep-prune! object path)`	Remove evidence along the path, rebuilding parents or replacing exhausted children with stubs.
`deep-merge!`	`(deep-merge! source target (path '()))`	Merge digest-compatible evidence into the target or into the optional target path.
`deep-copy!`	`(deep-copy! object source-path target-path)`	Read a value from one nested path and set it at another.
`deep-call`	`(deep-call object path callback)`	Evaluate a callback against the selected object and return the callback result while leaving ancestors unchanged.
`deep-call!`	`(deep-call! object path callback)`	Evaluate a callback, retain the mutated child node, and rebuild every ancestor.

Table 1: Deep Object Operations

4.4 A Controlled Semantic Web

The project's `standard.scm` object system grows semantics from a small evaluator: portable classes compose through public protocols and ledgers place results into signed history. Application-defined objects carry domain concepts, while staged programs use a journal capability to compose reads, writes, historical resolution, retention, and further calls in the native path, history, and object vocabulary. One administrator-managed staged program may also launch through detached `call!`

after each successful commit, receiving that commit's index; launches may overlap and effects retain ordinary non-atomic semantics. This is periodic execution rather than a durable scheduler or outbox, because process failure between commit and dispatch can lose a launch. Evidence-producing read-only queries offer another possible surface. These facilities extend the substrate under three controls:

- **Deterministic transition.** Successors derive from committed definitions, explicit arguments, evaluator semantics, and controlled primitives; environmental facts enter shared computation only as prepared inputs.
- **Resource governance.** The optional sandbox bounds one malicious top-level operation: a private fixed 25-second host deadline covers guest and blocking nested work, blocking and detached descendants share a causal 512 MiB guest-memory envelope, and host-call, byte, fan-out, message, and leaf limits constrain mediation. Failure rolls back request-local state without replay, and each next request starts fresh. There is no deterministic Scheme-operation meter, caller budget, or public accounting contract [36, 52]; coordinated concurrent exhaustion still requires host admission control or cgroups.
- **Explicit evolution.** Public protocols and layouts are compatibility contracts. Migrations must state authority, validation, rollback, partial-object behavior, and runtime compatibility rather than treating executable history as automatic semantic compatibility.

5 The Journal-Relative Web

5.1 A Web of Resources

The conventional Web organizes interaction around sites, origins, URLs, resources, representations, links, and clients [29, 7]; the Synchronic Web gives each term a historical analogue. A journal resembles a site with its own administration and evolving state, its root supplies an origin for interpretation, a path identifies a resource from that root, and a committed head selects the historical view in which resolution occurs. The returned representation may be bytes, an expression, a partial proof, or a self-encoding object. Resource identity has two dimensions: a path such as `components/17/inspection` gives continuity across versions, while a digest identifies the exact state found there at one head. A client can therefore request the current resource, resolve the same path at an earlier index, or cite one particular representation by digest. Paths cross semantic as well as structural boundaries: one segment may enter a tree, another an object, and another a foreign history retained beneath a bridge, with each object interpreting its documented portion while preserving end-to-end evidence. Because historical states remain addressable, ordinary links embedded in representations can cite a persistent version rather than only a moving current resource; relative references remain ordinary URI references interpreted by the client against the selected resource [58, 46]. Every journal thus induces a distinct coordinate system over local and reachable foreign state; a bridge segment records the relationship through which a foreign coordinate became reachable, a historical index selects the committed viewpoint, and content-derived identifiers name exact representations [25, 34, 67]. A prior state becomes part of the ordinary address space, partial evidence can travel as its representation, and clients can distinguish unknown structure, empty structure, unavailable data, policy denial, and unreachable endpoints.

5.2 Access Across the Web

Most useful data begins in an operational form—relational rows, object records, files, indices, transactions, or application-specific graphs—while an addressable network form favors stable resource

boundaries, linking, selective transfer, citation, and plural retention. Table 2 presents these as complementary views. A Web projection maps selected operational state into journal resources under a contract covering deterministic encoding, paths, resource boundaries, progress manifests, correction, privacy, deletion, and replay. The source can remain load-bearing while an asynchronous projection supplies historical evidence, and mature clients may gradually let selected projected resources assume an operational role.

Aspect	**Addressable Network Form**	**Query-Optimized Operational Form**
Primary unit	Independently identified resource and relationship	Application record, row, object, index, or transaction
Identity	Root-relative path plus resolved state commitment	Application key interpreted within one operational system
Strengths	Linking, citation, selective transfer, plural ownership, and historical reference	Locality, transactions, compression, indexing, and efficient bounded queries
History	Durable resource states and cross-history relationships are first-class	Commonly application-defined, compacted, or maintained in separate logs
Typical costs	Traversal, synchronization, repeated metadata, and derived indexing	Provider dependence, schema coupling, and application-specific portability
Architectural role	Shared network projection and durable evidence	Load-bearing operations and rebuildable derived views

Table 2: Complementary Data Forms

Local paths provide access within one journal; federation extends access through exact journal objects already committed beneath those paths. The canonical public form is one flat journal-relative path: an optional origin index, followed by directional bridge aliases with an optional destination index after each alias, then a namespace and terminal-local path. Thus `(*state* alice report)` is local and current, `(carol bob *state* alice report)` follows two current bridge aliases, and `(20 carol 15 bob -3 *state* alice report)` selects a historical index at every journal. Omitted indexes mean current, aliases are receiver-local rather than global journal names, and repeated `*bridge*` markers remain an internal storage and compatibility detail. Live `get` and `set!` follow the current committed object at each hop; `resolve` combines that current route for authentication and endpoint context with independently selected historical indexes for content. Missing material may fill a stump only when its digest remains fixed, and a live continuation never substitutes a newer embedded checkpoint. The implemented signed application surface comprises scalar `get`, `set!`, and `resolve` plus dedicated `get-batch` and `set-batch!` over one current working route. A staged batch signature binds ordered paths and values, codec choice, and the exact presence and content of optional expected values; the terminal authenticates first and then authorizes every member, requiring both read and write authority for a conditional write. Public `resolve-batch` instead groups committed paths by compatible route and history internally, while batch pinning and unpinning mutate retention only at the origin; direct `route`, constrained `trace`, `info`, `size`, and `synchronize!` remain bridge/control protocol. Finite routes may revisit a journal, with terminal authorization deciding whether the resulting principal is permitted. Figure 7 shows the exact-object handshake and direct terminal invocation:

- **Exact forward route.** From A's selected committed head, each hop follows the nested journal object already stored there until A reaches the exact B object containing B's stable journal audience, interface endpoint, and index/digest context.
- **Receiver-relative return.** B returns a partial object at that same historical B head containing

B's reverse route to A's interface key and root-sequence context; A requires its index, digest, and reverse sequence to agree with the B object already committed beneath A's forward route [17].

- **Direct application call.** A signs and sends the B-rooted object with a `get`, `set!`, or `resolve` invocation directly to the committed canonical endpoint; intended production deployments use HTTPS. B matches the raw supplied-object digest against its own history before evaluation, reconstructs the canonical bridge principal, checks audience and signature, and authorizes the request locally. Intermediaries see public route/control metadata while the private function, path, arguments, and value travel only on the final connection.
- **Result and retention.** Live `get` results and `set!` acknowledgements receive operational assurance from the direct connection to the committed endpoint; `resolve` additionally returns proof that A verifies against the selected historical B object. A may then retain that proof through a separate origin-local `pin!` over the full origin-relative historical path.

The records implementation and Scheme tests cover exact-object multi-hop routing, descriptor/key/endpoint/audience binding, terminal authorization, signed staged batches and conditional writes, grouped committed resolution, origin verification, local proof retention, self-returning finite routes, and rejection of local-only operations. Multi-node HTTP and browser QA has exercised direct and multi-hop `get`, `set!`, and `resolve`; ancestor projection and unauthorized-child denial; proof round-trip and origin-local pinning; outage recovery; and bridge deletion, exact establishment checks, repair, and re-establishment.

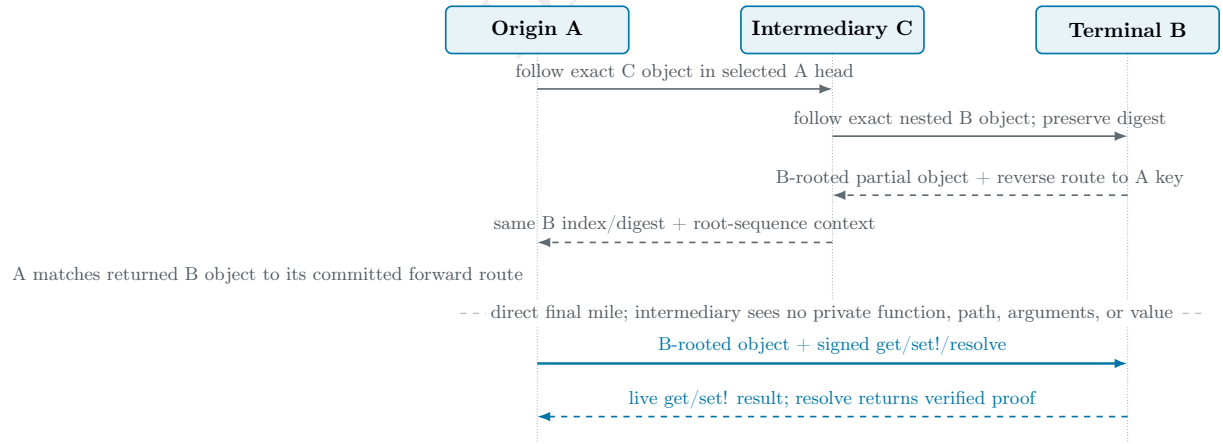

Figure 7: Exact-Object Route and Invocation

5.3 Identity as History

Identity, integrity, and authorization form a continuous historical path. A journal creates a stable identity commitment as the SHA-256 digest of a domain-separated random nonce; bridges, preapprovals, and invocation audiences bind that identifier rather than a permanent signing key. Each head carries the current journal signing key, and a receiver accepts a later key only through a predecessor-authorized transition rooted at its retained checkpoint. The same verified committed head binds the journal name, interface key, and canonical endpoint, while `info` supplies convenience metadata. For an invocation, the origin supplies an explicit scalar—its journal, a locally authenticated user, or a public caller—which the exact route turns into a receiver-relative principal

such as `(alice *state* carol)` or `(alice bob *state* carol)`. The terminal-local `key-index` records the terminal history index whose committed bridge object authenticated that origin key, and an authorization rule may constrain it relative to the terminal's current index; it is a local historical key window rather than an origin freshness marker. Terminal rules grant path-scoped `get`, `set!`, and `resolve` permissions only, including the corresponding authorization applied to each batch member. Administrators are local `(*state* name)` principals, and bridge management, configuration, authorization changes, retention mutation, secrets, windows, and root-plane calls remain local. A local administrator's `bridge!` performs implemented two-phase orchestration by synchronizing the peer before re-entering the authenticated local mutation with prepared data. Replay is intentionally accepted: `set!` keeps ordinary repeatable-write semantics, transport retries may repeat it, and applications add identifiers, idempotency, or state preconditions when their domain requires them. Across all cases, the committed route identifies the origin, its signature authenticates the invocation and optional user attestation, and terminal state maps the resulting principal to permitted operations.

5.4 A Web of Viewpoints

A mature Synchronic Web begins from the journal a person, organization, or application chooses to trust locally. That journal is simultaneously a store, historical memory, policy engine, and gateway into other histories; following an ordinary link is a client action from this viewpoint, with the target resolved through the same historical addressing and local policy as any other resource. Discovery supplies candidate journals, endpoints, resources, and routes through descriptors, indices, social exchange, existing links, and out-of-band knowledge. Historical continuity lets candidates update endpoints and interface keys while preserving identity, local policy promotes selected candidates into bridges or trusted routes, and the journal's graph gradually records both reachable resources and the relationships through which they became meaningful. Clients make this architecture usable. A browser, filesystem, gateway, projection worker, or inspection tool chooses roots and heads, fetches and merges evidence, invokes objects, enforces policy, and applies retention intent; routine interactions present ordinary resources while audit views expose paths, signatures, slices, and receipts. Application vocabularies such as W3C PROV, domain schemas, scientific objects, and future protocols remain resources above the generic history and integrity layers [50, 51], with definitions that evolve through self-encoding objects and relationships that can identify resources across journal boundaries. The governing principle is that every resource is reached from a chosen viewpoint, every historical claim carries its evidence, and every act of trust belongs to the journal that relies on it.

6 Application Sketches

The following sketches place the same architecture in four settings with different institutions, retention horizons, and consequences. They are design exercises grounded in the implemented integrity, object, and local-history layers; their federated stages inherit the implementation and acceptance-test boundaries described in Section 5.

6.1 Decentralized Web Archives

A conventional site can continue serving pages from its content system while a projection worker gives selected resources durable journal-relative paths. Each capture records its deterministic representation, interpretation metadata, source path, and projection cursor; the site's current page

remains convenient for browsing, while the projected resource adds a history that independent archives can retain and verify. Several archives may preserve overlapping portions of that history—one complete captures, another text and metadata, and a third selected high-value resources—with slices that remain comparable through shared commitments. Reciprocal bridges can then place important source and archive heads into one another's later histories, leaving incorporation receipts that survive the loss of any one service. This model supports gradual adoption. Static sites, databases, and content-management systems remain operational while the projection adds an audit path. Native clients can later resolve historical resources directly, follow ordinary links into retained versions, or compare captures from several archival viewpoints. Long-term operation still demands capture-fidelity policy, privacy review, legal-erasure procedures, malicious-content handling, media renewal, and cryptographic migration [12]. The journal gives those activities a stable evidentiary structure.

6.2 Production Traceability

Consider a manufacturer M that records an inspection report for component 17 in its quality-management database. A projection worker encodes the report at `components/17/inspection`, where the path names the continuing production resource and an application-defined object binds this version's bytes, metadata, and interpretation; M then records declared transition context, signs the successor state, and advances to head M_i, extending existing traceability models with independently retained history [63, 27]. When buyer B requests evidence, M slices M_i along the inspection path to retain the report, its definition, and the ancestor evidence needed to verify the root. B can merge and pin that slice according to procurement policy while M prunes its local historical copy after the retention window. If M and B maintain a reciprocal bridge, B accepts M_i and later incorporates it into head B_j, whose inclusion proof becomes a receipt establishing that the inspection state preceded B's later history. Protected details travel through an exact-object route handshake: B follows the manufacturer object already committed beneath its selected head, matches M's returned reverse-key object to that same checkpoint, and then sends the signed request directly to M for terminal authorization. A correction later produces identifiable successor M_{i+1}, and B can record acceptance, dispute, or follow-up inspection in its own history. Cryptographic evidence preserves the reports and their ordering; physical identifiers, inspection practice, and institutional judgment connect that evidence to the component itself.

6.3 Self-Sovereign Identity Anchors

A person or organization can establish a stable path beneath a selected well-known root and use it as an anchor for a profile hosted by an evolving journal. The profile can contain current endpoints, public keys, service relationships, and links to selected claims, all interpreted by observers from roots they already recognize. Changing providers then becomes a historical transition: the earlier state points toward the successor relationship, the new host publishes its descriptor and key, and observers follow the retained trail according to their own policy. Historic name trails developed a close model for changing personal online identifiers [42]; journal history extends the trail to richer state, reciprocal receipts, and application-defined relationships. Self-sovereignty here means continuity under chosen governance. Anchor operators, witnesses, and communities define how recovery, delegation, impersonation disputes, privacy, and endpoint discovery work. The journal structure preserves the evidence those rules consume: prior keys, authorized transitions, witnessed heads, and competing branches. A person can change hosts while keeping the history through which others recognize the change.

6.4 Agent Reputation Discovery

A user can begin from journals and communities already trusted for a task, then follow explicit relationships to reporting agents whose reports cite sources, record methods, receive corrections, and accumulate later outcomes. Reputation becomes a view computed under local rules from inspectable history that can travel across ranking services. Different communities may retain different evidence and value prediction calibration, source transparency, or specialized performance; their judgments can diverge while still citing common reports and outcomes. Secure per-node logs and deterministic replay offer a related accountability model [26], while partial journal history makes selected evidence portable across evaluators. Sybil resistance, collusion analysis, key protection, and source quality remain parts of the reputation institution. Historical integrity contributes stable inputs—which claim appeared, which sources accompanied it, which correction followed, and which community endorsed the result—that a chosen journal can combine into a local discovery view and preserve for inspection. Across all four sketches, operational systems continue doing the work they perform best: projection gives selected state durable coordinates, commitments preserve succession, bridges relate institutional histories, partial evidence distributes retention, and self-encoding objects carry interpretation. The architecture becomes valuable where several of these needs meet in the same resource.

7 Related Work and Lineage

The literature usually calls a structure whose answers can be checked against a commitment an *authenticated data structure*. This paper uses *integrity-verifiable* for that structural property and reserves authentication for identity, origin, and authority. The vocabulary keeps digest consistency, signatures, and authorization visible as separate steps.

7.1 Integrity and Entangled History

Cryptographic timestamping establishes one-way succession among committed documents [24]. Persistent authenticated dictionaries make earlier versions queryable [1]; general authenticated-structure models and generic programming techniques extend the pattern across data structures [45, 48]. Compact multiproofs reduce repeated Merkle evidence, while tamper-evident logs organize efficient historical queries and controlled deletion [57, 15]. Transparency systems apply these structures to public accountability. Certificate Transparency, CONIKS, and CHAINIAC use append-only or collectively witnessed histories to make authority behavior inspectable [35, 47, 53]. PeerReview combines attributable logs with deterministic replay for distributed systems, and transparency overlays generalize witness comparison across applications [26, 11]. Maniatis and Baker's secure timeline entanglement is the direct precedent for bridge-composed history [41]. Timeweave imports signed foreign timeline threads into later authenticators, yielding temporal mappings, incorporation receipts, and proofs that survive the originating service. Synchronic Web bridges carry this established mechanism into sparse relationships among journals whose histories contain path-addressed, partially retained, self-encoding state.

7.2 Provenance and Preservation

Untrusted-storage systems show how clients can retain compact state and detect server equivocation or tampering [38, 40, 30]. The Synchronic Web adopts the same client-held-commitment discipline while allowing each journal to maintain its own state and history. Database provenance distinguishes

why- and where-provenance, while Web provenance models processes, evidence, and responsibility [10, 49, 50]. W3C PROV standardizes entities, activities, agents, derivations, and responsibility for interoperable exchange [51]. Nanopublications package scientific assertions with provenance and publication context [22]; secure provenance protects provenance records as they cross untrusted environments [27]. These models can inhabit journal resources as application vocabularies. Content-derived naming and distributed preservation supply another lineage. IPFS, Trusty URIs, secure names, and self-certifying paths bind identifiers to content or cryptographic authority [6, 34, 25, 46]. Merkle-CRDTs use Merkle DAGs for convergent replication, Named Data Networking centers retrieval on named data, and LOCKSS combines peer replication with auditing for preservation [59, 67, 43]. Tiered fault tolerance treats rare failures and cryptographic aging as ordinary long-horizon events [12]. The Synchronic Web joins these concerns through resources whose content, history, and interpretation can be retained at different levels of completeness.

7.3 Decentralized Webs and Identity

Web architecture separates resource identification, representations, and interaction; RFC 3986 supplies global URI syntax and relative-reference resolution [29, 7]. Memento adds datetime negotiation and TimeMaps for earlier resource states [65]. Synchronic Web paths apply rooted and historical resolution within a journal's verifiable state. SDSI develops linked local namespaces, and SPKI/SDSI chain discovery builds proofs of naming and delegated authority for a relying party's policy [58, 13]. Decentralized trust management likewise evaluates presented credentials under local rules [9]. Historic name trails follow changing online identifiers [42], while earlier Synchronic Web identity work describes verifiable chains between anchors [17]. Receiver-relative principals and exact committed-object routes give these ideas a construction from checkpoints retained in journal history. Local-first software emphasizes local ownership and responsive operation; Solid separates user-controlled stores from applications [32, 44]. Webstrates persists and synchronizes mutable DOM state, including code, while AT Protocol combines signed per-user repositories, decentralized providers, and account portability [33, 31]. Data spaces develop governance among organizations, and the Semantic Web pursues machine-interpretable relationships among resources [54, 8]. Verifiable credentials provide a mature vocabulary for issuer, holder, verifier, claim, and proof roles [60]. The journal-relative Web places these application and governance choices above a common layer of integrity, history, and local policy.

7.4 Persistent Computation and Agents

Orthogonal persistence makes values durable independent of type-specific storage code, while tierless languages reduce representational boundaries across application layers [4, 14]; Emerald supports mobile objects, Argus structures distributed programs through guardians and atomic actions, Safe Tcl confines portable scripts, and proof-carrying code attaches statically checkable evidence to behavior [56, 39, 36, 52]. Distributed computation offers neighboring models: Ethereum and Hyperledger Fabric couple execution to shared ledger governance [66, 2], Linda provides associative tuple spaces [21], Bayou supports weak connectivity, Dynamo prioritizes availability, and CRDTs define convergent replicated types [64, 16, 61]. CALM characterizes monotonic programs that avoid coordination, while Merkle search trees organize state-based CRDT exchange [28, 5]. Self-encoding objects draw from these traditions by binding behavior and state inside an integrity-protected representation evaluated locally while histories preserve relationships among independently advancing states. Agent systems illustrate the demand for durable structured context. Retrieval-augmented generation supplies external documents; Generative Agents stores experience streams and reflections;

AriGraph combines semantic and episodic memory; and HippoRAG builds a long-term retrieval index [37, 55, 3, 23]. Agent-oriented programming and KQML provide an earlier lineage for explicit state, intention, and communication [62, 20]. Journals offer these systems portable evidence, historical correction, and locally evaluated reputation while leaving agent orchestration to the application. Every major ingredient therefore has a substantial precedent. The architectural contribution is their composition: self-encoding structures retain identity under partial possession, independent histories become entangled, journal-relative paths expose those histories as resources, and clients turn proof handling into ordinary interaction. Composable-ledger work provides an earlier step toward this synthesis [18]; the present architecture develops it into a plural Web.

8 Discussion and Conclusion

8.1 Bounded Guarantees

The Synchronic Web composes bounded assurances: a root and verifier establish structural integrity; a signature authenticates a head's key; historical resolution identifies earlier states; entanglement orders a foreign head before incorporation; a committed route connects a remote key and endpoint to retained evidence; and authorization applies local rules. Authentication gives a key historical meaning, domain evidence supports truth, preservation sustains availability, witnesses expose divergent branches, and selective disclosure, terminal policy, secure transport, and protected storage sustain confidentiality. Confidentiality extends into persistence: journals durably hold private data and credentials, so the client, evaluator, runtime, persisted state, backups, and key store form the trusted computing base. The Root secret remains host-only and unpersisted; a distinct rotatable Interface bearer lives in private Root state but is absent from Ledger and Federation objects, proofs, traces, user state, and public configuration, making backups live credential material. Table 3 summarizes these mechanisms and dependencies. Their limits clarify failure: a surviving digest identifies lost evidence, two signatures can expose equivocation, and a deterministic transition can reproduce a supplied timestamp without establishing its accuracy. The architecture preserves evidence for judgment and keeps the boundary visible.

8.2 Costs and Architectural Fit

Integrity adds hashing, persisted handles, proof paths, and verification work; history adds retained states and key stewardship; partial possession saves content storage while keeping ancestor evidence, stumps, and serialization metadata; self-encoding objects add evaluation, confinement, debugging, compatibility, and migration; and federation adds exact-object routing, direct transport, terminal-local authorization, origin verification of historical results, and application handling of accepted replay. A controlling exact-checkpoint social comparison measured 86.22% of native throughput and 1.193× Journal CPU per cycle; four-Journal PSS medians of approximately 1.42× at the measurement boundary and 1.44× after idle were material but noisy. No hard stop or parity claim follows, and negative host tuning leaves the portable AOT as the sole retained direction. These costs make architectural fit important, as summarized in table 4. The Synchronic Web becomes compelling when independent roots, selective possession, durable interpretation, historical correction, and cross-institutional evidence coincide, as they may in production traceability, portable identity, or decentralized archives. Applications may still add coordination above the journal layer for payments, settlement, exclusive allocation, deadlines, or physical custody; journals give these domain protocols durable inputs and histories, while each application defines its shared decision procedure.

Mechanism	Assurance	Operational Dependence
Integrity structure	Available material agrees with an accepted commitment	Accepted anchors, canonical encoding, correct verification, and retained copies for availability.
Partial structure	Selected evidence retains one complete-state identity	Proof metadata, disclosure policy, resource bounds, and custodians for omitted material.
History	Successive states and earlier identities remain verifiable	Key continuity, storage, migration, and witness comparison among signed branches.
Self-encoding structure	State remains durably associated with its interpretation	Compatible runtimes, capability confinement, request isolation, host admission, documentation, validation, and migration.
Entanglement	Foreign history precedes a later incorporating head	Synchronization, committed inclusion, receipt retention, witnesses, and local trust policy.
Federated route	A terminal authenticates an origin through the exact journal object committed in its own history	Exact committed route objects, a reachable committed endpoint, authenticated production transport, terminal-local authorization, origin verification of `resolve`, and application handling of accepted replay.
Rooted paths	Resources retain continuity from a selected viewpoint	Root choice, discovery, route maintenance, historical context, and indexing.

Table 3: Architectural Assurance Boundaries

System	Suitable Situation
Ordinary database	One operator with efficient current-state queries.
Signed documents	Complete immutable payloads from known publishers.
Content-addressed store	Immutable retrieval by digest.
Transparency log	Public accountability for one authority.
CRDT	Replicas expected to converge on one application state.
Global ledger	Participants sharing transaction order and consensus governance.
Synchronic Web	Independent roots requiring selective possession, durable interpretation, historical correction, and cross-institutional evidence.

Table 4: Architectural Fit

8.3 From Architecture to Specification

The records layer supplies a reference implementation of the architecture: digest and handle separation, partial possession, root advancement, reciprocal continuity, stable journal identity with predecessor-authorized signing-key rotation, exact-object multi-hop routing through directional aliases, signed descriptor and audience binding, direct `get`/`set!`/`resolve`, dedicated staged batches and grouped committed resolution, terminal-local authorization, and origin verification and retention of resolved proofs. A durable specification should define these observable invariants without freezing the current object layouts or service decomposition, and conformance tests should connect each invariant to accepted and rejected traces. Its assurance boundary must remain equally explicit. Acceptance of a foreign head is not incorporation; only a later signed head plus inclusion proof establishes the cross-history ordering claim. Routine signing-key rotation preserves a stable journal identity, but catastrophic loss or compromise of the accepted signing authority still requires a governance rule outside that lineage. The shared execution boundary isolates capabilities and copies inert values, while request isolation can contain one causal operation through a private fixed host deadline and mediation bounds; it exposes no deterministic Scheme-operation or public accounting contract and cannot prevent aggregate out-of-memory failure from coordinated concurrent requests without host admission controls. These are bounded properties of the architecture, not unfinished protocol features: they identify where applications, operators, and future specifications must add policy without weakening the integrity claims made here.

8.4 Toward a Plural Web

The architecture begins from a modest observation: durable information needs both continuity and a point of view. Integrity-protected state, signed succession, retained definitions, and cross-history receipts supply continuity; a journal that chooses roots, preserves evidence, follows relationships, and applies local policy supplies the viewpoint. Their composition produces a distinctive Web in which resources survive changes in representation and host, histories intersect while retaining independent order, observers possess different portions of one committed state, definitions evolve beside the objects they interpret, and federated paths preserve the historical routes through which other journals became meaningful. This is the deeper value of timeline entanglement in a resource architecture [41]: foreign history becomes locally retained evidence, self-encoding structure turns evidence into durable meaning, journal-relative paths make that meaning findable and citable, and specialized clients keep the machinery quiet during ordinary use and explicit during inspection. A plural Web will still contain disagreement, loss, compromise, and institutional change; its strength lies in giving those events durable form through corrections, receipts, witnesses, and independently held roots. Networked information can thereby become easier to carry across time while preserving the diversity of systems and institutions that give it meaning.

Glossary

Reference definitions for recurring terms introduced throughout the paper.

Authentication
: Verification of an asserted identity, origin, or authority through evidence such as credentials and signatures. Merkle integrity supplies the complementary structural check against a commitment.

Authorization
: A policy decision determining whether a principal may perform an operation. In current federation, terminal-local rules grant remote principals path-scoped `get`, `set!`, and `resolve` permissions; administrators remain local principals.

Bridge
: A reciprocal signed-head-retention relationship through which each journal preserves a verified head for the other. Reciprocity provides identity and history reachability, while terminal policy independently grants application-data authority.

Digest
: A cryptographic hash derived from a node or larger structure and used as its integrity identifier.

Historical cursor
: The independently selected origin and per-hop indexes used only by `resolve`. A canonical public journal-relative path interleaves these indexes with directional aliases; the Interface separates that path into the current route used for authentication and endpoint context and the historical cursor used to select committed content.

History
: Used in its ordinary broad sense for prior states and changes. Concrete chains, heads, and receipts represent history using ordinary Synchronic Web structures.

Integrity
: The property that modification can be detected relative to a known commitment. Authentication and authorization supply the associated claims about identity, origin, and authority.

Journal
: The software an entity runs together with all data persisted on it. A journal supplies the root from which that entity views the Synchronic Web and is analogous to a site or origin on the conventional Web.

Journal identity
: A stable commitment formed from a domain-separated random nonce and bound by bridges, preapprovals, and invocation audiences. Active signing keys may evolve beneath this identity through predecessor-authorized transitions.

Key-index
: The terminal-local history index whose committed bridge state authenticated a remote interface key. Authorization may constrain this index relative to the terminal's current history.

Merge
: To combine compatible partial structures whose integrity relationships agree.

Node
: A generic term whose meaning follows from context. It may denote a node in the integrity-verifiable data-structure DAG or an entity in the graph of journals and relationships.

Object
: A durable structure that couples executable behavior with state. Its documented protocol determines how the state is read, transformed, and represented; its internal layout remains implementation-specific.

Origin-local pin
: Retention of verified remote proof material in the originating journal under the full origin-relative historical path. It follows a federated `resolve` and mutates the origin's retention state.

Path
: A sequence interpreted from a selected journal root to address a resource or intermediate structure. A canonical federated path interleaves optional historical indexes with directional aliases before the terminal namespace and local path. Like a relative URL, its meaning is completed by a starting root.

Prune
: To remove retained material while preserving the integrity relationships needed to identify the larger structure.

Record
: Any snippet of data stored on a journal or transmitted between journals. The term is intentionally fluid across object shapes, historical entries, and storage units.

Resource
: A path-addressed value or object in a journal-relative view. Its low-level representation may be a leaf or a larger self-encoding structure.

Root
: A distinguished reference from which persisted structure is reached and interpreted. Unless otherwise qualified, a journal root is also the starting point for paths in that journal's view.

Root commitment
: A digest or integrity reference accepted as the fixed point against which a structure is verified. Additional evidence is required to establish who created, supplied, or endorsed it.

Slice
: A proof-preserving partial structure retaining the material required for selected paths while replacing unrelated material with stubs.

State
: Used in its ordinary broad sense for the condition or contents of a system, structure, object, or resource at a relevant point or context.

Step
: A discrete journal transition that turns staged changes into a new historically resolvable state.

Stub
: A representation that retains the digest of unavailable, undisclosed, or intentionally pruned structure while omitting its contents. A stub represents unknown structure; null represents empty structure.

Stump
: The low-level persisted node form used to represent a stub. It stores a retained logical digest under a distinct local node handle.

Synchronic Web
: The network, data structure, computational medium, and protocol family formed when

independently rooted journals make resources and histories reachable through journal-relative paths and bridges.

Terminal journal
: The journal that directly receives and authorizes a federated application invocation after the origin follows a working route to its committed endpoint and authentication object.

Working route
: The ordered receiver-local directional aliases used from the origin's latest committed view to select the terminal authentication object, endpoint, and reverse path to the origin key. Its implicit indexes are current; `resolve` may select historical content independently.

Entanglement
: Cross-history ordering created when one journal incorporates another's committed head into its own later state. The term follows Maniatis and Baker's secure-history protocol [41].

References

[1] Aris Anagnostopoulos, Michael T. Goodrich, and Roberto Tamassia. Persistent Authenticated Dictionaries and Their Applications. In *Lecture Notes in Computer Science*, pages 379–393. Springer Berlin Heidelberg, 2001.

[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15. ACM, 2018.

[3] Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. AriGraph: Learning Knowledge Graph World Models with Episodic Memory for LLM Agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 12–20. International Joint Conferences on Artificial Intelligence Organization, 2025.

[4] M. P. Atkinson, P. J. Bailey, K. J. Chisholm, P. W. Cockshott, and R. Morrison. An Approach to Persistent Programming. *The Computer Journal*, 26(4):360–365, 1983.

[5] Alex Auvolat and François Taïani. Merkle Search Trees: Efficient State-Based CRDTs in Open Networks. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 1–10. IEEE, 2019.

[6] Juan Benet. IPFS: Content Addressed, Versioned, P2P File System, 2014.

[7] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. Internet Standard RFC 3986, RFC Editor, January 2005.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[9] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE, 1996.

[10] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and Where: A Characterization of Data Provenance. In *Lecture Notes in Computer Science*, pages 316–330. Springer Berlin Heidelberg, 2001.

[11] Melissa Chase and Sarah Meiklejohn. Transparency Overlays and Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 168–179. ACM, 2016.

[12] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Tiered fault tolerance for long-term integrity. In *7th USENIX Conference on File and Storage Technologies*, pages 267–280. USENIX Association, 2009.

[13] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[14] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. Links: Web Programming Without Tiers. In *Lecture Notes in Computer Science*, pages 266–296. Springer Berlin Heidelberg, 2007.

[15] Scott A. Crosby and Dan S. Wallach. Efficient Data Structures for Tamper-Evident Logging. In *18th USENIX Security Symposium*. USENIX Association, 2009.

[16] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[17] Thien-Nam Dinh, Justin Ding-Yuan Li, Mitchell Negus, and Kenneth Goss. Synchronic Web Digital Identity: Speculations on the Art of the Possible. Technical Report SAND2025-06684R, Sandia National Laboratories, May 2025.

[18] Thien-Nam Dinh and Nicholas Pattengale. Composable Ledgers for Distributed Synchronic Web Archiving. In *2023 ACM/IEEE Joint Conference on Digital Libraries*, pages 242–244. IEEE, 2023.

[19] Thien-Nam Dinh, Nicholas Dylan Pattengale, and Steven Elliott. The Synchronic Web. Technical Report SAND2023-11382R, Sandia National Laboratories, January 2023.

[20] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an Agent Communication Language. In *Proceedings of the third international conference on Information and knowledge management - CIKM '94*, pages 456–463. ACM Press, 1994.

[21] David Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[22] Paul Groth, Andrew Gibson, and Jan Velterop. The Anatomy of a Nanopublication. *Information Services and Use*, 30(1-2):51–56, 2010.

[23] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. In *Advances in Neural Information Processing Systems*, 2024.

[24] Stuart Haber and W. Scott Stornetta. How to Time-Stamp a Digital Document. *Journal of Cryptology*, 3(2):99–111, 1991.

[25] Stuart Haber and W. Scott Stornetta. Secure Names for Bit-Strings. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 28–35. ACM, 1997.

[26] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, pages 175–188. ACM, 2007.

[27] Ragib Hasan, Radu Sion, and Marianne Winslett. Preventing history forgery with secure provenance. *ACM Transactions on Storage*, 5(4):1–43, December 2009.

[28] Joseph M. Hellerstein and Peter Alvaro. Keeping CALM: When Distributed Consistency Is Easy. *Communications of the ACM*, 63(9):72–81, 2020.

[29] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. W3c recommendation, World Wide Web Consortium, December 2004.

[30] Beom Heyn Kim and David Lie. Caelus: Verifying the Consistency of Cloud Services with Battery-Powered Devices. In *2015 IEEE Symposium on Security and Privacy*, pages 880–896. IEEE, 2015.

[31] Martin Kleppmann, Paul Frazee, Jake Gold, Jay Graber, Daniel Holmgren, Devin Ivy, Jeromy Johnson, Bryan Newbold, and Jaz Volpert. Bluesky and the AT Protocol: Usable Decentralized Social Media. In *Proceedings of the ACM Conext-2024 Workshop on the Decentralization of the Internet*, pages 1–7. ACM, 2024.

[32] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. Local-First Software: You Own Your Data, in Spite of the Cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 154–178. ACM, 2019.

[33] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 280–290. ACM, 2015.

[34] Tobias Kuhn and Michel Dumontier. Trusty URIs: Verifiable, Immutable, and Permanent Digital Artifacts for Linked Data. In *Lecture Notes in Computer Science*, pages 395–410. Springer International Publishing, 2014.

[35] Ben Laurie. Certificate Transparency. *Communications of the ACM*, 57(10):40–46, 2014.

[36] Jacob Y. Levy and John K. Ousterhout. A Safe Tcl Toolkit for Electronic Meeting Places. In *First USENIX Workshop on Electronic Commerce*, New York, NY, July 1995. USENIX Association.

[37] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[38] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis Shasha. Secure Untrusted Data Repository (SUNDR). In *6th Symposium on Operating Systems Design and Implementation*. USENIX Association, 2004.

[39] Barbara Liskov. Distributed Programming in Argus. *Communications of the ACM*, 31(3):300–312, 1988.

[40] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Depot: Cloud Storage with Minimal Trust. In *9th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 2010.

[41] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *11th USENIX Security Symposium*, pages 297–312, San Francisco, CA, August 2002. USENIX Association.

[42] Petros Maniatis and Mary Baker. A historic name-trail service. In *Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems and Applications*, pages 88–99. IEEE, 2003.

[43] Petros Maniatis, Mema Roussopoulos, T. J. Giuli, David S. H. Rosenthal, and Mary Baker. The LOCKSS Peer-to-Peer Digital Preservation System. *ACM Transactions on Computer Systems*, 23(1):2–50, 2005.

[44] Essam Mansour, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. A Demonstration of the Solid Platform for Social Web Applications. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 223–226. ACM, 2016.

[45] Charles Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A General Model for Authenticated Data Structures. *Algorithmica*, 39(1):21–41, 2004.

[46] David Mazières and M. Frans Kaashoek. Escaping the Evils of Centralized Control with Self-Certifying Pathnames. In *Proceedings of the 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*, pages 118–125. ACM, 1998.

[47] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing Key Transparency to End Users. In *24th USENIX Security Symposium*. USENIX Association, 2015.

[48] Andrew Miller, Michael Hicks, Jonathan Katz, and Elaine Shi. Authenticated Data Structures, Generically. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 411–423. ACM, 2014.

[49] Luc Moreau. The Foundations for Provenance on the Web. *Foundations and Trends® in Web Science*, 2(2–3):99–241, 2010.

[50] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The Open Provenance Model Core Specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, 2010.

[51] Luc Moreau and Paolo Missier. PROV-DM: The PROV Data Model. W3c recommendation, World Wide Web Consortium, April 2013.

[52] George C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '97*, pages 106–119. ACM Press, 1997.

[53] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive Software-Update Transparency

via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security Symposium*. USENIX Association, 2017.

[54] Boris Otto and Matthias Jarke. Designing a Multi-Sided Data Platform: Findings from the International Data Spaces Case. *Electronic Markets*, 29(4):561–580, 2019.

[55] Joon Sung Park, Joseph O'Brien, Carrie Jun 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. ACM, 2023.

[56] Rajendra K. Raj, Ewan Tempero, Henry M. Levy, Andrew P. Black, Norman C. Hutchinson, and Eric Jul. Emerald: A General-Purpose Programming Language. *Software: Practice and Experience*, 21(1):91–118, 1991.

[57] Lum Ramabaja and Arber Avdullahu. Compact Merkle Multiproofs, 2020.

[58] Ronald L. Rivest and Butler Lampson. SDSI: A Simple Distributed Security Infrastructure, 1996.

[59] Hector Sanjuan et al. Merkle-CRDTs: Merkle-DAGs Meet CRDTs, 2020.

[60] Johannes Sedlmeir, Reilly Smethurst, Alexander Rieger, and Gilbert Fridgen. Digital Identities and Verifiable Credentials. *Business & Information Systems Engineering*, 63(5):603–613, 2021.

[61] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In *Lecture Notes in Computer Science*, pages 386–400. Springer Berlin Heidelberg, 2011.

[62] Yoav Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[63] Keith Stouffer, Michael Pease, Joshua Lubell, Evan Wallace, Connor Freeberg, Steve Granata, Vivian Martin, Andrew Noh, and Harvey Reed. Blockchain and related technologies to support manufacturing supply chain traceability: Needs and industry perspectives. Technical Report NIST IR 8419, National Institute of Standards and Technology, April 2022.

[64] D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the fifteenth ACM symposium on Operating systems principles - SOSP '95*, pages 172–182. ACM Press, 1995.

[65] Herbert Van de Sompel, Michael Nelson, and Robert Sanderson. HTTP Framework for Time-Based Access to Resource States—Memento. Informational RFC RFC 7089, RFC Editor, December 2013.

[66] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger, 2014.

[67] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.