Observer-Declared Timelines for Real-World Agent Work

Keren Dong
Kungfu Origin Technology Limited
keren.dong@kungfu.link

Draft: July 2026

Abstract

Long-lived agent work crosses sessions, machines, repositories, interfaces, and human decisions. Its facts can remain stable while different participants receive different cuts and projections of them. A system that hides those boundaries can turn a selected view into an apparent global history. This paper derives a response from the KFD Foundation Triad: if facts must not drift, trust must start from facts, and cooperation must start from trusted value, then a perspective-bearing timeline must declare the observer for whom its facts were selected and ordered. We call this rule *observer-declared timelines*. We define a minimal contract over an observer, accepted fact cut, projection policy, causal constraints, consequence position, known gaps, and verification evidence. We then derive a second-order requirement: when the component presenting a view restarts or changes, continuity of perspective must be evidenced rather than inferred from a reused name or live process. Kungfu's Episode authority, exact-cut readiness, and fenced recovery mechanisms provide bounded implementation evidence for parts of this contract. They do not yet establish a reproducible multi-machine observer projection. The contribution is therefore both a design rule and a staged proof obligation: stable facts are necessary, but a trustworthy and actionable timeline must also say from where, for whom, and under which evidence boundary those facts became a view.

1 Introduction

Consider a long-running agent task that crosses two sessions. The first session changes a repository, starts remote work, records one failed check, and ends before the task is complete. A later participant opens the local product and asks what happened and what action is safe now. The local interface, remote runtime, repository, build system, and retained session evidence may each contain valid facts. They need not contain the same facts, reach the same cut, or order concurrent facts in the same way.

The KFD Foundation Triad describes how such work can remain one shared world across participants and time [3]:

$$\text{Facts} \rightarrow \text{Trust} \rightarrow \text{Cooperation} \rightarrow \text{New Facts}. \tag{1}$$

KFD-1 preserves stable reference, KFD-2 binds reliance to evidence, and KFD-3 makes relevant value and constraints visible to a participant before action. Cooperation then creates consequences that return through review into a successor fact world. That loop solves a continuity problem, but it creates a new pressure: no participant consumes the entire world from every position.

The pressure appears separately at each layer of the triad. If a selected or transformed fact cut is presented without perspective, selection can masquerade as reality and violate the stable reference required by KFD-1. If a claim does not identify which cut and projection support it, a later

participant cannot reconstruct the warrant required by KFD-2. If a view hides whose consequence position, available actions, and known gaps it represents, a participant cannot inspect the value and constraints needed by KFD-3; cooperation degrades into guessing, obedience, or repeated reconstruction.

This yields the first derived pressure from the Foundation Triad:

$$\text{non-drifting facts} + \text{multiple participants} \Longrightarrow \text{declared perspective.} \tag{2}$$

KFD-4 names the corresponding rule: timelines must declare their observer [10]. An observer-declared timeline does not deny facts or causality. It states which observer position is represented, which facts were accepted, how the view was projected, which causal relations dominate that projection, what consequences or actions the view is meant to support, and where evidence remains incomplete. The result is not a view from nowhere. It is a reproducible claim about how one view was formed from a shared fact world.

The core rule leads to a second question only after the perspective is defined: what happens when the observer implementation itself restarts, upgrades, or is replaced? A stable product label does not prove that a new process accepted the same facts or applied the same projection semantics. We therefore distinguish a logical observer from its incarnations and treat continuity between them as an evidence-bearing transition. This is a derived refinement of the core rule, not a prerequisite for first understanding why perspective must be declared.

This paper makes five contributions:

- a derivation of observer declaration from the fact, trust, and cooperation responsibilities of the KFD Foundation Triad;
- a minimal model over accepted fact cuts, causal constraints, projection policy, consequence position, degradation, and verification;
- an incarnation and continuity-witness refinement for observers that fail, restart, or change implementation;
- a mapping from that generic model to Kungfu's Episode fact authority, exact-cut readiness, fenced recovery, and product surfaces; and
- bounded implementation evidence together with an explicit account of the observer-level evaluation that remains incomplete.

Section 2 develops the problem through the running example and distributed-systems context. Section 3 defines the generic model without requiring a Kungfu-specific causal object. Section 4 then introduces Episodes and runtime mechanisms as one concrete realization. Section 5 separates evidence for that substrate from the still-open end-to-end observer claim, and Section 6 compares related work.

2 Background and Motivation

2.1 One work world, several fact cuts

We use *real-world agent work* to mean work in which agents help users manage concrete tasks rather than only produce isolated model outputs. Such work may involve local state, repositories, remote terminals, GUI and TUI controls, model calls, releases, documents, and human decisions. The

product problem is not only how to run an agent. It is how to keep the work legible when no session or interface contains its complete identity.

Return to the cross-session task. The repository may prove which files changed. The remote runtime may contain later execution facts. The local GUI may have accepted only an earlier remote range. A build service may know that one artifact failed after the first session ended. The transcript may explain an intention without proving that the intended action occurred. None of these sources is automatically the whole world, and combining their timestamps does not remove their different authority, freshness, and consequence boundaries.

A useful view therefore needs more than recorded facts. It needs to say which facts it accepted, what it omitted, how it ordered concurrent material, and for which participant and action context the projection is being shown. Otherwise two participants can appear to disagree about reality when they actually hold different cuts or apply different policies.

2.2 The global-clock temptation

Distributed systems theory has long shown that causal order is not equivalent to physical time. Lamport's happened-before relation separates causality from wall-clock order [13]; vector-clock style systems make concurrency explicit [14]; and distributed snapshots show that a useful global state is a constructed consistent cut rather than a simultaneous physical observation [2].

Yet product surfaces often sort mixed-source facts by timestamp and render one history. That may be sufficient for casual logs. It is not sufficient when a human or agent must audit a claim, recover after interruption, challenge an ordering assumption, or decide what action is safe. A deterministic global sequence can still be misleading if the product does not declare which facts entered it and which policy resolved concurrency.

Observer declaration is not a replacement for clocks, consensus, tracing, or replication. It is a product-level obligation above them. Even a centrally sequenced view has an accepted input boundary and a consumer for whom the view has consequences.

2.3 The KFD-4 obligation

The Foundation Triad can be summarized as:

KFD-1: facts must not drift.
KFD-2: trust must start from facts.
KFD-3: cooperation must start from trusted value.

KFD-4 applies those responsibilities to perspective-bearing surfaces:

KFD-4: timelines must declare their observer.

The word *observer* does not mean that truth is subjective. It marks the position from which a bounded fact world becomes a view. A declaration must preserve source identity and known causality, expose the evidence boundary for trust, and make the view's action relevance and limitations inspectable to the participant expected to use it. Different observers may legitimately receive different projections; they may not silently invent facts or invert known causal relations.

2.4 The observer is not immortal

Once a view has an observer, long-lived work exposes another boundary. The component producing the view may restart, a frontend may upgrade, a process identifier may be reused, or a new agent may attach to surviving work. Reusing the same name does not prove that the successor accepted the same fact cut, reconstructed the same projection, or applies compatible semantics.

Failure-detector theory separates suspicions about process failure from stronger guarantees [1]. At the product layer, liveness is likewise weaker than continuity. A replacement may continue a logical perspective only after it identifies the authority it accepted, rejects stale claims, rebuilds or verifies derived state, and declares the semantics under which it presents the result. Otherwise the product should expose a discontinuity or degraded view.

This incarnation problem refines KFD-4 after the basic perspective obligation is established. It must not obscure the simpler first claim: before asking whether an observer continued, the system must say which observer's view it is.

3 Observer-Declared Timeline Model

An observer-declared timeline is a deterministic, evidence-bounded projection of accepted facts from a declared perspective. The model is a surface and verification contract, not a physical storage schema. In particular, it does not require facts to be packaged as Episodes; Section 4 introduces Episodes as Kungfu's implementation choice.

3.1 Facts, authority, and accepted cuts

A *fact* is an event, record, manifest, receipt, payload commitment, causal edge, human decision, or verification result treated as evidence. Each fact has source identity, provenance, and an authority boundary. Occurrence, record, and admission remain distinct: something may happen without being recorded, and a record does not automatically enter every accepted fact world.

For a view V, let A_V be its *accepted fact cut*. The cut names the source ranges, heads, cursors, manifests, watermarks, freshness bounds, or other coordinates admitted for that view. It also records known omissions. A view need not accept every fact the product has seen, but it must not present an undeclared subset as complete reality.

Accepted facts and their source commitments are authority inputs. A database index, folded state, GUI history, exported ordering, or summary is derived. Deleting and rebuilding a projection must not rewrite authority, and disagreement between authority and projection must be reported as drift.

3.2 Observer and consequence position

An *observer* is the perspective for which a view is formed. It may be a human, agent, workspace, runtime location, service, or product surface. The observer is not necessarily the author or owner of the facts.

A useful declaration also states the observer's *consequence position*: the purpose, responsibility boundary, and action context in which the view is expected to be used. The same accepted facts may support different relevance or presentation for a local operator, a remote worker, a release reviewer, or an auditing agent. Consequence position does not permit invention of facts. It explains why a bounded projection matters and which actions it can responsibly support.

This field carries KFD-3 into the model. Without it, observer identity can become inert metadata while the interface still hides whose value, constraints, and available choices shaped the view.

3.3 Causality and projection policy

Let $\prec_V$ be the known causal relation induced by A_V, including declared cross-source dependencies. A projection policy P_v selects and orders accepted facts for display, replay, export, or agent consumption under semantic version v. For observer O, the visible timeline is

$$T_{O,P_v} = \Pi_{O,P_v}(A_V, \prec_V). \tag{3}$$

The projection Π must be deterministic for its declared inputs and must produce a linear extension of known causality. If $a \prec_V b$, a valid projection cannot place b before a. Concurrent or causally unrelated facts may be ordered by observer policy, source priority, local sequence, or a stable tie-breaker, but the choice must be inspectable and versioned. When accepted evidence cannot support a total order, the result is partial, degraded, or rejected rather than silently repaired with wall-clock order.

3.4 Trust and cooperation obligations

An observer declaration is useful only if later participants can test two classes of claim.

Reliance. The declaration must expose enough authority coordinates, policy identity, causal constraints, gaps, and verification evidence to reconstruct why the view may be trusted for its stated purpose. It is not a universal trust score; another purpose may require another assessment.

Action. The declaration must expose enough consequence position, safe remaining operations, responsibility, and constraint information for a participant to choose, refuse, or bound a next action. A technically reproducible ordering that hides its action relevance does not complete the KFD-3 obligation.

These obligations keep the timeline inside the Foundation loop. The view starts from a fact cut, supports bounded reliance, enables inspectable cooperation, and allows resulting consequences to return as new evidence without letting the action certify its own history.

3.5 Degraded evidence

A timeline may remain useful when evidence is incomplete. Instead of hiding that incompleteness, it should declare states such as missing causality, incomplete accepted ranges, stale remote facts, redacted payloads, schema mismatch, unverifiable source, unresolved conflict, unknown policy, or a projection that has not reached its required cut.

Degradation is operation-sensitive. Missing payload bytes may prevent replay while still allowing manifest inspection; a stale index may prevent indexed query while still allowing rebuild; an incomplete remote range may block a completion claim while still supporting local inspection. A single health bit is weaker than a report that names the evidence gap and the capabilities that remain safe.

3.6 Observer incarnation and continuity

The core model identifies whose perspective a view represents. A long-lived logical observer may then have multiple physical or software incarnations. We model one incarnation as

$$O^e = (o, e, r, v, c_a, c_p), \tag{4}$$

where o is the logical observer identity, e is a monotonic incarnation or epoch, r identifies the implementation performing the projection, v is the policy semantics, c_a is the accepted authority cut, and c_p is the cut reached by derived projection. A name, PID, socket, heartbeat, or GUI instance cannot substitute for these claims.

Replacing an incarnation creates a transition rather than identity by assertion. Let

$$W_{e \to e+1} = (O^e, O^{e+1}, c, q, R), \tag{5}$$

where c is the predecessor cut accepted by the successor, q states the policy or runtime compatibility relation, and R is retained recovery evidence. A valid witness shows how the successor rejected stale authority, accepted a non-regressing cut, rebuilt or verified derived state, and reached a declared readiness condition. A semantic change may preserve fact continuity while producing a new view identity; the witness must not erase that change.

This separates three claims: *fact-authority continuity*, in which facts remain intact across replacement; *observer continuity*, in which a successor proves its relation to the prior perspective; and *presentation continuity*, in which an interface reconnects and displays a coherent view. Presentation continuity cannot prove the first two.

3.7 Minimal declaration

The core observer-declared surface contains:

- stable view identity and subject;
- observer identity and consequence position;
- accepted fact sources, ranges, authority cut, and known omissions;
- projection cut, policy version, causal dominance, and tie-breaker;
- degraded evidence states and safe remaining operations; and
- verification status and evidence references.

When continuity across implementations is claimed, the surface additionally contains current incarnation, predecessor relation, runtime or implementation identity, and continuity-witness status. These fields refine the published KFD-4 observer-perspective schema [10]; they are proposed from implementation experience and are not yet a published schema revision. A schema defines vocabulary and a verification boundary. Importing it cannot by itself prove that an adopter's runtime or displayed view is correct.

4 Kungfu Instantiation and Prototype

The preceding model does not require one physical ledger, causal-object type, or runtime topology. This section maps it to Kungfu, infrastructure for managing real-world work with agents. Kungfu realizes accepted causal units as *Episodes* and supplies single-host authority, projection-readiness, and continuity mechanisms around them. Episodes are an implementation substrate in this paper, not a logical prerequisite for observer declaration. Their broader role as a candidate primitive belongs to companion work. The current Kungfu path does not yet implement the complete multi-machine observer projection.

4.1 Runtime fact ledger

The runtime fact ledger preserves source-local facts, provenance, causal links, payload evidence, schema bindings, and verification results. Future sync views also need accepted source ranges and freshness boundaries. The visible timeline is a projection over the ledger, not the ledger itself.

This distinction matters. If storage flattens remote facts into anonymous local records, the product loses the ability to explain what was accepted and why. If the GUI presents a global-looking order without observer metadata, the user or agent cannot reproduce the view during recovery. Runtime recovery must likewise rejoin authority at an accepted cut rather than infer continuity from a live process.

4.2 Implemented Episode authority

An Episode is a bounded causal segment with independent identity, manifest, frame references, content commitments, provenance, schemas, dependencies, and verification evidence. It gives Kungfu a unit that one observer view can accept, move, inspect, or reject without treating a physical journal page or product session as the universal causal boundary. The prototype separates four authority layers [11]:

1. An append-only yijinjing manifest journal stores fixed-layout Episode lifecycle, frame, reference, and dependency records as authority.

2. One typed C++ fold derives the canonical current Episode view. Python, Node, CLI, and GUI edges do not define independent Episode semantics.

3. Content-addressed immutable storage resolves payload and schema references, with hashes and lengths verified before use.

4. Structural fsck checks manifest integrity, causal closure, dependencies, content evidence, lifecycle state, and safe capabilities. Recovery and repair append evidence instead of silently rewriting verified history.

The current Episode query path folds the manifest directly. Kungfu also has rebuildable SQLite projections for other query surfaces, but the source-registry projection must not be reported as an Episode projection. A dedicated Episode projection and its drift/rebuild qualification remain pending. This correction is important: authority/projection separation is a design invariant, not evidence that every planned derived store already exists.

4.3 Exact-cut runtime readiness

Kungfu's runtime activation contract distinguishes process placement from semantic readiness [6]. A runtime handle binds a workspace, runtime identity, generation, capabilities, and readiness evidence. A live projection capability is Ready only after a verified durable cut exists and the projection has reached an explicit projection cut. PID liveness, sockets, heartbeats, service installation, and GUI presence remain diagnostics; none establishes the cut.

Recovery creates a new authority-bearing generation. Handles, leases, and receipts from an older generation fail as stale rather than being rebound to a reused PID or route. This supplies concrete counterparts for e, c_a, and c_p in the observer-incarnation model without making activation a second fact authority.

4.4 Fenced continuity across coordinator restart

Live Peer continuity uses a two-dimensional fence

$$(runtime\_generation, coordinator\_epoch)$$

and one shared registration schema [8]. The runtime generation advances when the admitted runtime changes; the coordinator epoch advances for every coordinator process within a generation. Peers report the last authority they admitted. Both sides reject regressing or non-advancing authority instead of treating PID, wall clock, or journal presence as proof.

After coordinator loss, a Peer reconnects through bounded retries, clears candidate projection state, and rejoins the replacement command journal at the accepted registration cut. Only the corresponding new start request may move it from Recovering to Ready. Delayed frames from the predecessor cannot satisfy the new bootstrap. This is a concrete continuity witness for one single-host runtime boundary: it proves a fenced transition, not an immortal process.

4.5 Session and presentation continuity

Kungfu's Agent Session Capsule further separates stable work identity, one physical provider attempt, control authority, and presentation attachments [7]. Coordinator and session-stream epochs are independent. A GUI restart may reattach to a surviving stream, while a Capsule failure ends the old physical attempt unless provider-supported semantic resume creates a new one. Product surfaces expose states such as `recovering`, `action-required`, and `ended`; a lost worker is not rendered as an empty successful session.

This demonstrates why presentation continuity must remain separate from fact and observer continuity. A terminal snapshot can restore usability, but it does not establish work completion, evidence quality, or causal truth.

4.6 Versioned upgrades

The staged upgrade control plane installs immutable runtime images before Core selects them [9]. Each workspace generation pins a runtime build, artifact root, entrypoint, manifest digest, and protocol/schema contract. Compatibility produces an explicit plan: apply now, defer until idle, require semantic resume, or block an incompatible transition. Downloading bytes does not grant live authority.

Readiness commits a generation switch. Rollback changes routing but does not rewrite Episode, journal, work, or provider-session facts. This supports policy and implementation identity in the observer model: fact-authority continuity may survive an upgrade while observer semantics change visibly. The contract and synthetic fixtures are implemented, but signed platform delivery and a native upgrade fault campaign remain separate stages.

4.7 KFD-4 implementation map

Table 1 distinguishes the generic KFD-4 obligation, its current Kungfu realization, and the remaining observer-view work. Observer incarnation and continuity are refinements of the current KFD-4 interpretation, not claims that its published schema already requires these exact fields.

Surface	Current Kungfu evidence	Remaining view work
Logical observer	Source, writer, workspace, runtime, and work-console identity	First-class observer and stable view identifier
Incarnation	Runtime generation, coordinator epoch, process-start identity, session-stream epoch	Exported observer-incarnation declaration
Accepted facts	Episode manifests, refs, dependencies, durable cut, and projection cut	Multi-source accepted ranges, watermarks, and freshness
Projection policy	Deterministic typed fold plus pinned runtime/protocol/schema identity	Versioned cross-source ordering policy, tie-breaker, and Episode projection
Causal constraints	In-Episode closure and declared Episode dependencies	Projection-level enforcement over a selected multi-source set
Continuity	Stale-authority rejection, accepted-cut rejoin, projection rebootstrap, and recovery states	Cross-host continuity witness and deterministic reproduction
Degradation	Structured fsck issues and explicit recovering/action-required states	View-level degradation for incomplete ranges and unknown policy
Verification	Episode oracle, fsck, retained runtime report, Peer campaign, and release gates	Exported declaration and projection-level causal fsck

Table 1: Current Kungfu evidence mapped to the observer-declared timeline contract.

4.8 Dual-first interfaces and remote work

The same timeline contract should be usable by humans and agents. A GUI can show the default observer view while exposing metadata on demand. A CLI or API can return the same cuts, policy, continuity, and degradation state directly. Agent-facing surfaces are not secondary integrations; they are first-class control surfaces for participants that may inspect the system more frequently than humans.

Real-world agent work often happens on remote machines. A server-side agent may write facts into a remote Kungfu home while the local GUI remains the user's primary control plane. Importing or syncing those facts must not imply that the local observer saw them live. The resulting view needs source location, accepted range, freshness, policy, and continuity evidence. Generic range- and Episode-based source sync is not yet complete, so this remains an end-to-end research boundary.

4.9 Verification gates

KFD-4 can be used as a release or product gate for perspective-bearing surfaces. Passing the gate should mean that the surface exposes the observer, accepted facts, projection policy, causal constraints, continuity status, and degraded evidence metadata. It should not mean that every adapter, remote source, or runtime payload is complete. Stronger claims require their own KFD-2-style trust assessment.

Kungfu's Episode, runtime-activation, and Peer-continuity qualifiers demonstrate that evidence discipline. Each binds a claim to source revision, platform, profile, raw evidence, and non-claims. None automatically certifies every timeline surface.

5 Preliminary Evaluation

Evaluation must follow the paper's dependency order. The primary claim concerns whether an observer declaration makes a perspective reproducible, auditable, and actionable. Evidence for a fact substrate or recovery mechanism is necessary but cannot substitute for that observer-level result. The present implementation evaluates two prerequisites: a verifiable Episode authority and a single-host continuity substrate that does not infer readiness from process liveness. The complete multi-machine observer view remains unevaluated. We keep evidence from different source revisions separate rather than combining it into one imaginary run.

5.1 Questions and method

The evaluation program asks:

- **RQ1:** Can two human- or agent-facing surfaces reproduce the same declared view from its observer, accepted cut, policy, causality, consequence position, and degradation metadata? This remains future work.
- **RQ2:** Does declaration expose hidden ordering assumptions, missing evidence, and differences in safe next action that an undeclared global-looking timeline hides? This remains future work.
- **RQ3:** Does the Episode path preserve the causal identity, evidence, recovery, and safe-capability semantics needed by an accepted fact cut?
- **RQ4:** Can runtime readiness be bound to exact authority and projection cuts rather than inferred from process existence?
- **RQ5:** Can a replacement coordinator reject stale authority, rebootstrap surviving Peers at an accepted cut, and expose recovery instead of false continuity?

Kungfu's Episode qualification harness drives the public Python storage service backed by the C++ implementation. A structurally independent Python oracle models abstract lifecycle and evidence transitions without importing Kungfu or reading journal bytes. The current semantic profile compares production observations with the oracle and requires every named evidence dimension, including capability soundness, to execute and pass [11].

Runtime activation uses a separate retained report. It binds clean source and tree revisions, platform, exact commands, suite results, raw-log hashes, and explicit non-claims [12]. Peer continuity and runtime upgrade use their own contracts and campaigns. Passing one layer does not upgrade a claim owned by another.

5.2 Observer-level evaluation program

The direct test of KFD-4 should begin with one shared causal scenario and vary the observer declaration. At minimum, a local operator, remote worker, release reviewer, and recovery agent should receive controlled overlaps and gaps in their accepted fact cuts. The experiment should compare an undeclared wall-clock timeline with declared projections that preserve the same known causality.

The primary measures are deterministic reproduction from exported metadata, causal-order preservation under clock skew, hidden assumptions found during audit, evidence gaps surfaced rather than silently repaired, and the ability of humans and agents to identify a justified next action or

refusal. This last measure is necessary: a view may be factually reproducible yet fail to expose the value and constraints needed for cooperation.

No current result in this paper completes that program. The remaining subsections report prerequisite evidence and preserve that qualification.

5.3 Episode semantic evolution

The historical numerical baseline used the v1 Trust Report at clean revision `b6961891731b`. It predated the complete safe-capability contract, so its unexercised capability dimension is not reinterpreted after the fact. The later C++ fold/fsck path emits `kungfu.episode.qualification/v1`; current Semantic v1 fixtures compare the exact safe-capability set across healthy, open, ended, aborted, degraded, and failed states. Both over-advertising and unnecessary contraction fail the required `capability_soundness` dimension.

The distinction illustrates the paper's evidence rule. A stronger current contract can close a previously identified gap, but it does not mutate an older report. Trust starts from the source-bound facts available for each claim.

The current Episode query path is manifest-direct. Earlier prototype text described an absent/stale/rebuilt SQLite Episode projection; current public qualification explicitly states that rebuilding a source-registry SQLite index is not an Episode projection rebuild. Projection derivation therefore remains a required semantic property and an implementation gap for a future dedicated Episode projection.

5.4 Historical scale and contention evidence

The metadata accumulation result from revision `b6961891731b` is shown in Table 2. The run used `darwin/arm64`, Node 22.22.3, 20 logical CPUs, 128 GiB RAM, and one reported seed for the largest tables.

Episodes	Ingest/s	List	Fsck	Inspect p95	RSS
1,000	1,670	40 ms	9 ms	1.1 ms	54 MB
10,000	937	444 ms	82 ms	11.0 ms	115 MB
100,000	316	4.20 s	804 ms	87.5 ms	698 MB

Table 2: Historical metadata-only Episode accumulation baseline.

At 100,000 retained Episodes, all 100,000 objects and 200,000 semantic Episode records matched after fresh-process readback. Clean recovery found no interrupted Episode. The physical manifest contained 200,002 frames; the two additional page-control frames were accounted for separately rather than misclassified as semantic records.

The contention profile kept total work fixed at 10,000 Episodes while using one, two, five, and ten physical writers. Aggregate ingest ranged from 1,143 to 1,230 Episodes/s. At ten writers, 8,252 explicit writer-busy results were retried; none exhausted the retry bound, and the run reported no count, readback, fsck, recovery, unexpected-error, or progress-timeout violation.

These observations support a bounded correctness claim, not a capacity promise. Throughput did not scale with writer count, whole-manifest costs increased with retained population, the workload was metadata-only, and only one of three declared baseline seeds supplied the largest tables.

5.5 Exact-cut activation evidence

The current retained runtime report is bound to clean Kungfu revision `b325b97392e6` on Darwin arm64. It passed all eight required suites: activation Core, product distribution, Profile action admission, runtime surface parity, bounded performance observation, frozen-product runtime smoke, full product verification, and product catalog verification [12].

The activation Core suite exercises daemonless storage, direct no-fork coordination, exact-cut readiness, first-call serialization, generation replacement, crash windows, lease expiry, and idle drain. Native readiness coordinates are published only after durability and projection authorities establish the requested cut. Consumers revalidate those authorities; descriptor bytes and process liveness are not promoted to readiness proof.

This supports a named-platform single-host claim: the implementation can bind a runtime incarnation to semantic readiness and a projection cut. It does not establish distributed election, cross-machine leases, high availability, physical-host power-loss recovery, Linux or Windows product qualification, or a universal latency SLO.

5.6 Peer continuity and upgrade evidence

The Peer-continuity implementation includes native contract tests, Python runtime tests, Agent Session transport tests, and a process campaign covering coordinator kill/restart, stale and future authority rejection, projection rebootstrap, and product-visible failure handling [8]. Stages 1–3 of the implementation are complete. The macOS process campaign passes locally, while retained Linux and Windows execution remain promotion boundaries. We therefore treat it as strong implementation evidence for the continuity-witness model, not as a cross-platform or cross-host qualification result.

The versioned upgrade control plane similarly has a registered contract, negative fixtures, and Python tests for immutable runtime installation, generation-fenced planning, activation, rollback, and reference-aware garbage collection [9]. The implementation is staged. Signed platform distribution, public product messaging, and native fault campaigns remain outside the present evidence. The paper consequently claims that the system can represent policy/runtime transitions explicitly, not that every desktop upgrade path is qualified.

5.7 Remaining end-to-end evaluation

The observer-declared timeline claim still requires the following case studies:

- **Remote agent work.** An agent performs a task on a server while the user later reviews the work from a local GUI.
- **Multi-machine sync.** Facts from local and remote runtimes are imported with different freshness boundaries and causal completeness.
- **Observer replacement.** A successor exports a continuity witness, reproduces the predecessor view at a selected cut, and then advances without accepting stale authority.
- **Policy upgrade.** The same accepted fact set is projected under two declared policy versions, with semantic difference made visible.
- **Replay and debugging.** A user or agent reconstructs why a task reached a given state after partial failure.

The corresponding measures include recovery time, hidden-ordering assumptions found during audit, deterministic reproduction from exported metadata, causal order preservation under clock skew, surfaced degradation rather than silent repair, and human/agent ability to select the next safe action.

Required artifacts include fixture ledgers, selected Episode sets, accepted source ranges, projected timeline exports, observer-incarnation and continuity declarations, KFD-2 trust assessments, and machine-readable qualification reports. The campaign must include clock skew, late remote evidence, missing ranges, missing causal links, policy-version mismatch, observer restart, and deterministic reproduction by both human-facing and agent-facing surfaces.

The paper should not rely on private operational logs for evidence unless they are sanitized, permissioned, and explicitly marked.

6 Related Work

6.1 Logical and physical time

Lamport clocks and happened-before relations establish that causal order is not the same as physical timestamp order [13]. Vector-clock systems make concurrent events explicit [14]. The observer-declared timeline rule does not replace these mechanisms. It brings a similar discipline to product surfaces: when the product shows a timeline, it should expose enough perspective metadata for the ordering to be audited.

Distributed snapshots construct a consistent global state without requiring a physically simultaneous observation [2]. An observer-declared view has a related concern, but its contract extends beyond one snapshot algorithm: it names accepted heterogeneous facts, projection semantics, evidence degradation, and the incarnation that claims the view.

6.2 Failure detection and recovery epochs

Failure detectors reason about what distributed processes can suspect and what reliable services can be built from those suspicions [1]. The incarnation model addresses a product-layer consequence: replacement must not turn liveness evidence into authority continuity. Generation and epoch fences are implementation techniques for rejecting stale actors; the proposed surface additionally asks a timeline to expose the cut and semantic relation that make a successor view reproducible.

6.3 Event sourcing

Event sourcing treats state as a derivation from an append-only sequence of events [5]. Observer-declared timelines share the idea that records matter, but differ in emphasis. The visible timeline is not assumed to be the authoritative event log. It is a perspective-bearing projection over accepted facts that may come from several sources. Episodes add a bounded causal and verification object above physical append blocks, while keeping projections rebuildable from authority.

6.4 Distributed tracing

Distributed tracing systems such as OpenTelemetry capture spans and causal relationships across services [15]. This is valuable evidence for runtime behavior. The proposed rule is broader at the product surface: a timeline may combine traces, repository facts, user decisions, release manifests, local files, and remote agent actions. An Episode may contain trace evidence, but its boundary is defined by causal closure and portable trust rather than by one tracing backend or span vocabulary.

6.5 CRDTs and replicated state

CRDTs provide convergence properties for replicated data types under concurrent updates [16]. Observer-declared timelines are not a conflict-free data structure. They are a view contract: even when underlying state uses CRDTs, consensus, or a central sequencer, a mixed-source product timeline should still declare what perspective the user or agent is seeing.

6.6 Human-computer and human-agent cooperation

Engelbart framed computing as augmenting human intellect rather than merely automating tasks [4]. Real-world agent work extends that problem. The product must support cooperation between human and agent participants that observe, remember, and act through different surfaces.

7 Discussion and Limitations

7.1 Not relativism

Declaring an observer does not mean that any story is acceptable. Facts, causality, hashes, manifests, schemas, cuts, and verification results remain load-bearing. The rule rejects an undeclared view from nowhere, not the existence of facts. Two observers may order concurrent facts differently; they may not silently invert known causality or invent evidence.

7.2 Not merely a metadata footer

Declaring an observer is not satisfied by attaching a username to an otherwise opaque timeline. The declaration is load-bearing only when it exposes the accepted fact cut, projection semantics, consequence position, gaps, and verification evidence needed for bounded reliance and action. This is where KFD-2 and KFD-3 remain active: a participant must be able to reconstruct why a view is trustworthy for a purpose and what choices or constraints follow from using it.

A system may therefore preserve fact identity and still fail KFD-4 at the surface. It may also produce a reproducible view that remains unusable for cooperation because responsibility or safe remaining operations are hidden.

7.3 The observer is neither omniscient nor immortal

The first version of the proposal emphasized that no observer has an absolute global vantage point. Runtime implementation exposed a second requirement: an observer also has a history. A logical identity can outlive a process, but the successor must not inherit authority or semantics by name alone.

This second-order refinement is useful beyond Kungfu. Long-lived local applications routinely restart coordinators, reconnect clients, replace caches, and upgrade binaries. Without incarnation metadata, a product may preserve visual continuity while silently changing its accepted cut or interpretation. The continuity witness makes that transition inspectable. It does not require one specific runtime topology; embedded, process-hosted, and remote implementations can provide different evidence for the same surface obligation.

7.4 Continuity is layered

Fact-authority continuity, observer continuity, and presentation continuity fail independently. Treating one as proof of the others creates common but dangerous shortcuts:

- a live PID does not prove that the process owns current authority;
- a reattached terminal does not prove task completion or evidence quality;
- an intact ledger does not prove that the current projection reached it;
- a successful software download does not prove that a new runtime may safely activate; and
- a familiar GUI does not prove unchanged policy semantics.

Making these claims separate may expose more recovering or action-required states. That is a product cost, but hiding them would exchange usability today for unauditable recovery later. Dual-first interfaces can reduce the burden by giving agents the same structured state that humans see in the product.

7.5 Not universal consensus

Some systems require consensus, total ordering, or a central sequencer. The observer-declared timeline rule does not forbid those designs. It says that when a product presents a timeline to a human or agent, the view should still declare the accepted facts, cut, projection policy, and observer incarnation behind it. A consensus result can itself be one accepted fact source.

7.6 Adopter responsibility

A schema can standardize vocabulary, but it cannot prove an adopter's runtime correctness by itself. An adopter that claims a concrete timeline is complete, fresh, continuous, or causally valid must provide product-specific evidence. This is why KFD-4 should be assessed through KFD-2-style trust claims when used as a release or product gate.

7.7 What the prototype establishes

The current prototype establishes that the proposed view can stand on a nontrivial fact and single-host continuity substrate: bounded causal objects, append-only authority, safe-capability assessment, exact-cut readiness, generation and epoch fencing, stale-authority rejection, explicit recovery states, and versioned runtime selection. It also demonstrates that different evidence revisions can remain honest instead of being retroactively merged into one stronger result.

It does not establish that observer declaration improves human recovery time, agent decision quality, or the independence of participant choice. Nor does it establish a distributed consistency protocol, conflict resolution between independently written authority roots, cross-host continuity, or a total order acceptable to every observer.

7.8 Implementation and evidence limits

The Episode numerical baseline is metadata-only, uses a single reported seed at the largest scale, and does not cover realistic payload volume, deep dependency DAGs, distributed writers, long soak, storage exhaustion, or every publication crash point. The retained runtime-activation result is Darwin arm64 and single-host. Peer continuity has a passing local macOS process campaign but not retained Linux and Windows promotion evidence. The upgrade implementation has contract and synthetic coverage but not complete signed-platform and native fault-campaign evidence.

Most importantly, the first-class observer policy object, accepted range synchronization, multi-source projection command, dedicated Episode projection, exported observer declaration, continuity

witness export, and projection-level causal fsck have not yet landed. These are not incidental implementation details. They are the remaining bridge between a qualified substrate and an evaluated multi-machine observer timeline.

8 Conclusion

The KFD Foundation Triad preserves a shared work world through a recursive kernel: facts support bounded trust, trust supports inspectable cooperation, and cooperation creates consequences that return as new facts. Operating that kernel across multiple participants reveals a necessary boundary. Stable facts do not produce a view from nowhere.

KFD-4 follows from that pressure. A perspective-bearing timeline must declare the observer, accepted fact cut, projection policy, causal constraints, consequence position, known gaps, and verification evidence that produced the view. KFD-1 prevents the projection from silently changing the underlying reference. KFD-2 requires the view's warrant to remain reconstructable. KFD-3 requires its value, constraints, and safe action boundary to be inspectable to the participant expected to use it. Observer declaration is therefore not a fourth slogan beside the triad. It is one mechanism required when the triad is used by participants who cannot occupy every position at once.

The core rule also exposes a second-order problem: an observer implementation may fail or change. A logical observer can outlive one process, but continuity between incarnations must be established through accepted cuts, policy and runtime identity, recovery evidence, and explicit degradation rather than a reused name or familiar interface.

Kungfu provides bounded implementation evidence beneath these claims. Episodes realize accepted causal units with append-only authority and safe-capability evidence. Exact-cut runtime readiness, generation and coordinator-epoch fencing, Peer rebootstrap, explicit recovery states, and immutable runtime selection show how one observer incarnation can change without silently acquiring stale authority. These mechanisms instantiate the model; they are not required to understand it, and their qualification does not certify the complete observer-level result.

That result remains open. It requires first-class observer declarations, accepted source ranges, reproducible cross-source projection, exported continuity evidence, projection-level causal verification, and experiments in which humans and agents reconstruct the same declared view and use it to choose or refuse a next action. Preserving this boundary is part of the argument: trust in a timeline must start from the facts available, while cooperation must start from a view whose perspective and consequences are visible.

References

[1] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[2] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[3] Keren Dong. Facts, trust, and cooperation: The KFD foundation triad. Public alpha release at GitHub, 2026. Version 0.1.0-alpha.8.

[4] Douglas C. Engelbart. Augmenting human intellect: A conceptual framework. Technical report, Stanford Research Institute, 1962.

[5] Martin Fowler. Event sourcing. `https://martinfowler.com/eaaDev/EventSourcing.html`, 2005.

[6] Kungfu Origin Technology Limited. ADR-0080: Live runtime activation is capability-driven and topology-neutral. Public architecture decision at GitHub, 2026.

[7] Kungfu Origin Technology Limited. ADR-0081: One fenced agent session capsule owns each live agent pty. Public architecture decision at GitHub, 2026.

[8] Kungfu Origin Technology Limited. ADR-0086: Live peers reconnect through fenced coordinator authority. Public architecture decision at GitHub, 2026.

[9] Kungfu Origin Technology Limited. ADR-0087: Product upgrades install immutable runtimes before core activates them. Public architecture decision at GitHub, 2026.

[10] Kungfu Origin Technology Limited. KFD-4: Timelines must declare their observer. Public KFD source at GitHub, 2026.

[11] Kungfu Origin Technology Limited. Kungfu episode object model and atomicity qualification. Public qualification source at GitHub, 2026. Source and evidence reviewed at revision b0165579c576c282b1805cacf8b11ed928765ce5.

[12] Kungfu Origin Technology Limited. Runtime activation and product-delivery qualification. Public retained report and checksummed raw evidence at GitHub, 2026. Retained Darwin arm64 report at revision b325b97392e6253de11e4b7e61b40b9e2fc639ce.

[13] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[14] Friedemann Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[15] OpenTelemetry Authors. Opentelemetry documentation. `https://opentelemetry.io/docs/`.

[16] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, 2011.