Deterministic Projection: Deriving a Byte-Identical Graph from an Append-Only Log

kgai maintainers

kgai.dev · team@kgai.dev · August 2026

ABSTRACT

kgai stores a team's engineering decisions as an append-only log of immutable, content-addressed events, and serves queries from a property graph derived from that log. This report describes the derivation. The log is the single source of truth. The graph is a read model that any machine can discard and rebuild, and that every machine rebuilds identically, byte for byte, from the same events. We present the replay semantics that make this hold. Events are applied in a canonical order, every projection write is idempotent under a per-event watermark, and references to facts that have not yet arrived are held by stub nodes that later replay completes, so rebuild is robust to partially transferred logs. We explain why the graph is rebuilt in full after synchronization rather than patched incrementally, and why determinism is a correctness property rather than an aesthetic one. When AI agents assemble working context from a shared memory, replicas that silently diverge would have teammates' agents acting on different versions of the same recorded history.

1. Introduction

Systems that keep an authoritative log and derive their queryable state from it are a well established lineage, from write-ahead logs through event sourcing to log-centric data infrastructure [1, 2]. kgai belongs to it. The companion reports in this series describe its unit of record, an immutable content-addressed decision event (TR-2026-01), and the coordination-free total order over events from many writers (TR-2026-02). This report describes the remaining step, the projection that turns the ordered log into the graph that readers query.

The design goal is stronger than convenience. kgai claims that two machines holding the same events derive the same graph, byte for byte, and it ships the means to verify the claim. We describe the mechanisms that make the projection deterministic, and we argue that for this system determinism is a correctness property. The graph is the working memory that AI coding agents read before changing a codebase. A shared memory that silently disagreed with itself across a team would be worse than none, because nothing would look wrong.

Section 2 states the division of authority between log and graph. Section 3 describes the read model. Section 4 gives the replay semantics as three invariants. Section 5 explains the decision to rebuild rather than patch. Section 6 develops the correctness argument, and Section 7 states limitations.

2. The Log as the Single Source of Truth

A kgai store is, authoritatively, a set of append-only shards of newline-delimited JSON, one shard per writing installation, each line one immutable decision event identified by a cryptographic hash of its content. Everything else a store contains is derived and disposable.

This division of authority is total, and it has practical corollaries that the system leans on.

Synchronization operates on logs alone. Teams share memory by exchanging the contents of their shards through an object store of their own. Because each shard has a single writer and events are immutable, merging is a union, and no synchronization step ever inspects or reconciles graph state.

Migration, likewise, is a log operation. Moving a project's history into a shared store is a copy of its shard files followed by a replay. The rationale, the author, and the original decision date all survive, because the shards are the record and everything else is derived from them.

Survivability rests on logs. The shards are plain text, open to reading and search with no help from the system's code. A team's decision history does not depend on the continued existence of the tooling that wrote it.

The graph, by contrast, carries no authority. It can be deleted at any moment and rebuilt from the log with a single command, and the system does exactly that as a matter of routine, as Section 5 describes.

3. The Graph as a Derived Read Model

The read model is an embedded property graph, queryable with Cypher, held locally beside the log. It arranges one body of data into two planes.

The live plane holds the current shape of the system. Its nodes are domain elements, features, services, and business objects, joined by typed links. It is small by design and always reflects only the present.

The decision plane holds provenance. Every decision appears as a node, connected by a SHAPES relationship to each element it touched and by supersession relationships to the decisions it replaced. History queries read this plane directly. How an element came to its current shape is read straight off the ordered chain of decisions shaping it, rationale by rationale, with no replay involved.

The arrangement is a specific instance of the general pattern of separating the write model from read models optimized for their queries [3], with event sourcing supplying the write side [2]. One log serves

several read shapes, which is the general benefit of keeping the system of record in event form and deriving views from it [7]. The live graph answers recall, the context query an agent runs before acting. The decision plane answers history queries. Replaying the log up to a chosen moment into an ephemeral graph reconstructs the system as it stood on that date. Free-text search is deterministic lexical matching of query words with element names and decision texts, so a given store answers identically on every machine.

4. Replay Semantics

Replay applies every event's mutations to the graph in a canonical order. Three invariants govern it.

Total order. Events are sorted by the pair of Lamport clock value and content hash, a coordination-free total order whose construction and guarantees are the subject of the ordering report in this series (TR-2026-02) [4]. The order is a pure function of the event set, so any two machines holding the same events replay the same sequence.

Idempotence. Every projection write is expressed as a merge keyed by the target's primary key, and each applied event is recorded under a watermark keyed by its hash. Replaying an event a second time is a no-op. Idempotence is what lets rebuild, partial replay after a pull, and retry after interruption all share one code path without special cases [5].

Dependency tolerance. A pulled log may be incomplete at the moment of replay. An event can reference an element or a superseded decision whose introducing event has not yet arrived. The projection attaches such references to stub nodes, created by the same merge discipline, and the stub is filled in when the real fact replays. Rebuild is therefore robust to incomplete logs, and completeness is restored by later synchronization rather than demanded up front.

Within the order, each decision's mutation list is applied atomically as one unit, so the graph never holds half a decision. The mutation vocabulary itself is small, element upsert, link addition, link retirement, and last-writer-wins property writes with canonically sorted keys, which keeps every projection step a deterministic function of the event and the graph state the canonical order has already produced.

5. Rebuild Rather Than Incremental Repair

After every synchronization, kgai rebuilds the projection from the full log in canonical order rather than patching the existing graph with the newly arrived events. Two facts force this choice.

First, arrival order and canonical order disagree. A pulled event may sort before events already projected. Appending its effects to the current graph would apply it as if it were newest, which it is not.

Second, some operations are last-writer-wins. Two events writing the same property key on the same element must land in canonical order on every machine. A replica that patched incrementally in arrival order could let an older write land after a newer one and silently diverge from a replica that received the same events in a different sequence. Divergence of exactly this kind is the failure mode the system exists to exclude, so the projection never applies events out of canonical order at all.

Rebuild is affordable because the model makes it so. The live graph is deliberately small, a property TR-2026-01 argues for on modeling grounds, and the projection is idempotent, so a rebuild is a plain replay with no compensation logic. The read model is treated as a cache with a warranty. Discarding it costs a replay, and holding it costs nothing in trust, because it can always be re-derived from the record [1].

6. Determinism as a Correctness Property

The guarantee the mechanisms above combine to give is byte identical derivation. Two stores holding the same events produce identical graphs, independent of arrival order, machine, or timing. The claim is verifiable rather than asserted. Exporting the graph in canonical form yields a sha256 digest, and machines whose digests agree hold the same graph. This is the state machine argument, deterministic operations applied in an agreed order yield identical replicas [5], with the agreed order obtained by construction instead of consensus.

For this system, the guarantee is not hygiene. It is what makes a shared memory shared.

The readers of a kgai store are AI coding agents assembling context before acting on a codebase. If replicas could diverge, two teammates' agents would assemble different context from the same recorded history, and would do so silently, with each machine internally consistent and nothing visibly wrong. The failure would surface only as agents acting on different versions of the team's memory. Determinism closes this class of failure at the storage layer. Same events, same graph, same answers to the same queries, on every machine.

Determinism also underwrites audit. Because the graph is a pure function of an immutable log, the state an agent read at a given point is re-derivable from the record, and a canonical digest can demonstrate that two parties are discussing the same state. An append-only record whose derived views can be regenerated and compared is the property Helland identifies as the reason immutable data composes so well across systems [6].

It is worth stating what makes this determinism attainable, because the choices are visible in the design. There is no learned or probabilistic component anywhere in the storage or retrieval path. Capture writes explicit events, identity is a hash of content, order is a sort over values each event carries, and projection is a merge. Retrieval matches words and traverses edges deterministically, and the bridging of vocabulary,

asking again with a synonym, is deliberately left to the agent doing the asking, which is the one component of the wider system that is probabilistic and the one component kgai does not contain.

Convergence has been exercised at scale in archived benchmark runs, with stores of one million decisions across thirty writers' shards replaying to matching canonical digests and answering decision lookups in around one hundred milliseconds, though recall and free-text search are slower than decision lookups. Performance characterization is not this report's subject, and the scale report in this series (TR-2026-09) treats the measurements in full.

7. Limitations

The determinism claim ends at the store's boundary. kgai guarantees that the same events yield the same graph and that the same query yields the same result. It does not guarantee that an agent asks the same questions or uses the answers the same way, and the wider workflow inherits the agent's variability.

Deterministic lexical retrieval is a trade. Matching is by word overlap over element names and decision texts, so synonym bridging depends on the asking agent rephrasing, and fuzzy semantic recall over large bodies of prose is deliberately out of scope.

Rebuild cost is bounded by the smallness of the live graph, which is a modeling assumption rather than an enforced invariant. A store whose writers mint elements promiscuously erodes the assumption, and with it the cheapness that makes rebuild after every synchronization comfortable.

Finally, byte identical replicas require identical event sets, and the system can only witness convergence, via digest comparison, for the events a machine has received. A replica behind on synchronization is consistent with the record it holds, not with the record as a whole.

8. Conclusion

kgai's projection turns an append-only log of decision events into the graph a team's agents query, and it does so as a deterministic function. Authority lives entirely in the log. The graph is derived, disposable, and rebuilt in full, in canonical order, whenever synchronization changes the record. Idempotent merges and stub nodes make replay insensitive to repetition and to incompleteness, and the canonical order makes it insensitive to arrival history. The result is a property most storage systems do not attempt, independent machines that provably hold the same memory, byte for byte. For a record whose consumers are autonomous agents acting on a shared codebase, we consider that property foundational. A memory that can drift is a memory a team cannot trust.

References

[1] J. Kreps. 2013. The Log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn Engineering blog. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

[2] M. Fowler. 2005. Event Sourcing. martinfowler.com. https://martinfowler.com/eaaDev/EventSourcing.html

[3] M. Fowler. 2011. CQRS. martinfowler.com. https://martinfowler.com/bliki/CQRS.html

[4] L. Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM 21(7), 558-565.

[5] F. B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. ACM Computing Surveys 22(4), 299-319.

[6] P. Helland. 2015. Immutability Changes Everything. ACM Queue 13(9). https://queue.acm.org/detail.cfm?id=2884038

[7] M. Kleppmann. 2017. Designing Data-Intensive Applications. O'Reilly Media.