Supersession Semantics: Decision Lifecycle in an Immutable Log

kgai maintainers

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

ABSTRACT

Engineering decision records are conventionally maintained by editing them in place, deleting them when they no longer apply, or expiring them through staleness heuristics. Each of these practices destroys information that teams and their AI agents later need. We describe the decision lifecycle of kgai, an open-source decision memory for AI-assisted development teams, in which a recorded decision is never edited and never deleted. Change is expressed exclusively through supersession, a new immutable decision that carries its own rationale and explicitly supersedes the prior head decisions of the elements it reshapes. Recall filters to the decisions currently in force, while superseded decisions, including rejected approaches, remain permanently queryable. Genuinely contradictory decisions surface as branches on an element and are resolved by one new decision that supersedes both heads, with the disagreement and its resolution both preserved. We contrast this model with expiry and reconciliation heuristics common in agent memory systems and state its limitations.

1. Introduction

Software teams make structural decisions continuously, and the rationale behind those decisions decays faster than the code it explains. The architecture decision record practice [1] responded to this by recording each significant decision as a short, dated document, and it introduced an important discipline that we retain: a decision record, once written, is not rewritten to say something else. In most real repositories, however, decision records remain ordinary files. They are edited, deleted, or silently abandoned, and the systems that manage memory for AI agents typically go further, updating or expiring stored facts automatically.

This report describes a stricter model, implemented in kgai, a local-first decision memory used by AI-assisted development teams. In this model a decision is an immutable event in an append-only log. There is no update operation and no delete operation anywhere in the data model. The only way to change what the system considers current is to record a new decision that supersedes an old one. We describe the semantics of that supersession link, how recall filters to the decisions in force, why abandoned approaches

remain queryable, and how contradictory decisions surface and resolve. We close with a comparison to heuristic staleness management and an account of what the model costs.

2. Background: an Immutable Log Projected into a Graph

kgai is event-sourced in the standard sense [2]. The source of truth is an append-only log of decisions. Each decision is a content-addressed event that carries who made it, why, and when, together with a list of structural mutations against a small graph of domain elements, the features, services, and business objects of the system under development. Events are stamped with Lamport clocks [3] and replay deterministically, in the order given by the pair of Lamport clock value and content hash, into a derived read model with two planes: the live element graph, which always shows the current shape, and a decision plane, which records every decision and its relationships to the elements it shaped. The projection is disposable and can be rebuilt from the log at any time. Companion reports in this series specify the ordering and the projection in full (TR-2026-02 and TR-2026-03).

Because a decision's identifier is a hash of its content, recording identical content twice is idempotent, and any change of content is by definition a new decision. Immutability is therefore not a policy layered on top of the store but a structural property of it, and the broader systems argument for designs of this kind has been made well elsewhere [4]. Everything in the remainder of this report rests on that property.

3. Supersession Semantics

We call a decision a head decision for an element if it shaped that element and no later decision has superseded it there. When a new decision structurally changes an element, the system records an explicit supersedes link from the new decision to the prior head decision or decisions of that element. Three properties define the semantics.

First, supersession is reasoned. The superseding decision is a full decision record in its own right, carrying its author, its date, and its rationale, so the link never appears without an accompanying why. A reader who asks how an element reached its current shape receives not only the current answer but the stated reason the previous answer stopped being the answer.

Second, supersession is not deletion. The superseded decision remains in the log and in the decision plane. It participates in history queries, in replay, and in point-in-time reconstruction. Its status changes from in force to superseded, and that status is derived from the supersedes relation rather than stored in a mutable field, so there is no lifecycle state machine that can be corrupted or skipped.

Third, supersession is per element. A decision that reshapes two elements supersedes the prior heads of both. Conversely, two decisions that touch disjoint elements never supersede one another, whatever their texts say. The relation tracks structural change, not textual similarity.

4. In-Force Filtering at Recall

The read paths divide cleanly along the two planes. Recall, the query an agent runs before touching an area, returns the head set of the elements concerned, meaning every decision still in force there. Superseded decisions never enter this view, so an agent assembling context is never handed advice the team has already revoked.

History is a separate, explicit query. It returns the ordered chain of decisions that shaped one element, superseded entries included and labeled, so the question how did this get this way has a complete, dated answer. A third query reconstructs the exact structure of the whole graph at a past moment, replaying the record as it stood then into an ephemeral projection.

Readers familiar with temporal databases will recognize the shape of this design. The log records what the temporal database literature calls transaction time [5], and the point-in-time query is a transaction-time query. The difference lies in the unit of record. Temporal databases version tuples, and the version history explains what changed but not why. Here the unit of history is a reasoned decision, so the same mechanism that reconstructs past structure also reconstructs past intent.

5. Dead Ends Are Preserved and Queryable

A consequence of never deleting is that the store accumulates dead ends, and we consider this a feature rather than an accident. An approach that was tried and rejected is recorded with the reason it failed, and it stays in the graph after a superseding decision replaces it. When an engineer, or an agent, later proposes the same approach again, a history query surfaces the earlier attempt and its recorded reason, and the team avoids re-walking a path it has already proved wrong.

In a mutable store the dead end is precisely the record most likely to be cleaned up, because it describes something the system no longer does. The supersession model inverts that judgment. The record of what the system no longer does, and why not, is among the most valuable content the store holds, and the lifecycle guarantees it survives.

6. Semantic Conflicts as Branches

Teams record concurrently. kgai merges per-writer shards during synchronization, so two people, or their agents, can decide the same element differently in the same window without either decision superseding the other. After the logs merge, that element has two competing head decisions.

We define this situation, and only this situation, as a real conflict, and the system represents it as a branch rather than an error. A dedicated query lists every element in that state. Resolution is itself a decision: someone records one new decision on the element, and it supersedes both heads. The branch and its

resolution both remain in history, so the record shows not only what the team decided but that the team briefly disagreed, and how the disagreement ended.

The contrast with replicated data type research is instructive. CRDTs [6] guarantee convergence by merging concurrent updates automatically, which is the correct behavior for data structure state. For engineering intent we argue it is the wrong behavior, because a disagreement between two recorded decisions is information, and an automatic merge would destroy it. The synchronization protocol that makes this the only conflict class in the system is the subject of the synchronization report in this series (TR-2026-06).

7. Comparison with Expiry and Staleness Heuristics

Agent memory systems in the current generation commonly manage change through time-to-live values, relevance decay, or model-driven reconciliation in which an extraction pipeline decides that a new statement updates or deletes an old one. We compare at the level of the category, along four axes.

Authority. Under a staleness heuristic, the judgment that a fact no longer holds is made by a timer or a model. Under supersession it is made by the author of a new decision, and the record names that author.

Rationale. A heuristic records no reason. Supersession attaches the reason to the exact point of change, because the superseding decision carries one.

History. Reconciliation that updates or deletes leaves no trail, or leaves one only as an implementation detail. Supersession preserves the full chain as a first-class query target.

Determinism. Model-driven reconciliation is probabilistic, and two runs over the same input can disagree. Supersession is recorded data, and replaying the same log yields the same lifecycle on every machine [7].

The comparison also has an honest other side. Heuristic maintenance demands no discipline from its users and operates over arbitrary prose, which is exactly what makes it attractive for general assistant memory. Supersession presumes that decisions are captured as structured decisions against named elements. It is a model suited to a curated engineering record rather than a universal memory substrate, and applying it requires the capture discipline that kgai automates elsewhere, described in the capture report in this series (TR-2026-07).

8. Limitations

The log grows without bound, by design. We accept the storage cost of keeping every superseded decision, and the model offers no mechanism to reclaim it.

Immutability cuts both ways. A decision recorded in error is permanent, and the only remedy is a superseding decision that says so. Content that should never have been recorded cannot be expunged from

the log by any supported operation.

Supersession tracks structural change to shared elements. Two decisions that contradict each other semantically but are recorded against different elements will coexist without a supersedes link and without appearing as a branch. The quality of the lifecycle therefore depends on the quality of element naming, which the system supports with deterministic identity but cannot fully guarantee.

Finally, branches do not resolve themselves. The system surfaces them and preserves their resolution, but a team that ignores the conflicts query will accumulate unresolved branches, and recall over a branched element honestly reflects that unresolved state.

9. Conclusion

We have described a decision lifecycle in which nothing is edited, nothing is deleted, and all change is expressed as new immutable decisions linked to what they supersede. In-force filtering keeps the working view current, history keeps every superseded decision and every dead end reachable, and contradictory decisions surface as explicit branches whose resolutions are themselves recorded. Compared with expiry and reconciliation heuristics, the model trades generality and storage for authority, rationale, history, and determinism. For the specific problem of a development team, human and machine, that must not lose the why behind its own system, we consider that trade favorable.

References

[1] M. Nygard. 2011. Documenting Architecture Decisions. Cognitect blog. https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions

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

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

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

[5] R. T. Snodgrass and I. Ahn. 1985. A Taxonomy of Time in Databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, 236-246.

[6] M. Shapiro, N. Preguica, C. Baquero, and M. Zawirski. 2011. Conflict-free Replicated Data Types. Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011), Lecture Notes in Computer Science, Vol. 6976. Springer, 386-400.

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