MNEMOSYNE: One Check on Every Memory Write

Oscar Serra · Independent Research

21 October 2026

Abstract

Think of a good assistant's notebook. When your weekly call moves to Thursday, she strikes the line that said Wednesday, writes Thursday beneath it with the date and who told her, and if it clashes with something pinned to the first page, she asks you first. Years later it is still trustworthy: every line says where it came from.

AI agents keep notebooks too, memory files they edit as they work. But one careless write can mislead every later turn: a line written months ago can contradict a rule refined today, or a guess can come back as fact.

We propose MNEMOSYNE: every change to memory passes through one gate. The gate sets each new line beside what it could clash with: the facts on the same topic and the rules read on every turn. A small model, the judge, says how they relate. Only that verdict may strike out an old fact, and the struck line is kept, marked with why it ended. The gate closes a fact on its own only where the judge has proved itself on data it was not tuned on. When the judge is down or the change is risky, the write waits where the owner can see it.

This is a design, with the tests that would decide it; none has run yet. If it holds, your agent's memory can be trusted the way that notebook is: a correction made once stays made, and you can ask what was true last month and what the agent believed then.

1. Introduction

An agent that serves one person for months learns the way an assistant does: the person says something once and expects it to stick. The main agent harnesses have a version of this: memory files the model edits through a tool (Anthropic, 2025), a git repository of Markdown files that a background agent tidies (Letta, 2026), or facts extracted from each conversation (Chhikara et al., 2025; Rasmussen et al., 2025).

The hard part of memory is not storing: a full session record can be kept word for word. It is that many hands write the store at many moments, and one careless write can poison every turn after it. Two kinds of memory file matter here. Most are topic files the agent opens when a task needs them. A few are *always loaded*: the harness puts them in front of the model on every turn. Two of those recur below: the *index file*, one pointer per topic, and the *rules file*, the owner's standing instructions. Four failures recur, and each shaped a choice here.

A write that contradicts a rule in another file. A line in the index file restates one of the owner's preferences in shorthand. He later refines it in the rules file, and two texts loaded on every turn disagree. We expect a check against the nearest stored memories to miss many clashes like this, since the lines sit in different files, months apart, in different words. That is a hypothesis, and one control arm of §12 measures how many it misses.

A fact that has forgotten where it came from. The agent runs one query, gets nothing back, and writes down that a kind of item does not exist. Weeks later it repeats the negative as if it had seen it. Zhan et al. (2026) call this authority collapse: consolidation keeps the claim and drops the conditions under which it may be used.

A closing rule with no judge. A store could decide that a line was replaced by its position: same file, same lines, new text. Re-flow a note so that one sentence becomes two lines, and such a rule would read the second line as a replacement of the first and end it, although nothing changed, and nobody would count these closures.

A correction written beside the error. The easy edit appends "this is wrong, see below". The wrong line stays, still reads as a plain statement, and an agent that reads only the first match picks it up again.

None of these is fixed by a better retriever. They happen at write time, and they are hard because the writers are many, the contradictions that hurt sit between files and against standing rules, deleting the old version destroys its history, and judging a conflict takes a model on every write, which costs money and sometimes fails.

Nor are they fixed by a bigger window. With every note in view, the old "Wednesday" of the weekly call and the new "Thursday" would both be there, and so would the owner's rule and the request that contradicts it; the model would settle each clash again on every turn, perhaps differently each time. Deciding once, when the line is written, means every later turn reads one current line, with its history one step away.

Therefore we propose one gate for every change to memory. *The claim in one sentence: check every memory write against the rules read on every turn, and let only a judge that has earned it close an old fact, which is kept.* The gate is also the only way into the store (§5), and it records where each write came from and how far it may be trusted (§8).

This paper is a design for the next build; it reports no results. The name is the Greek goddess of memory.

2. Where this sits

This is one of four designs for an agent's memory: keeping (the keeping paper, J1), writing (this paper, J14), finding (the finding paper, J2) and learning (the learning paper, J5). This paper owns what enters long-term memory: the choice to write, and the shape and checks of what is written.

Several products already act at write time, and each does something well that this design builds on. Table 1 runs the abstract's example through nine ways of keeping memory: the weekly supplier call that moved from Wednesday to Thursday, and a request from the owner, "copy me on supplier renewal drafts", that clashes with a standing rule, "nobody in copy". The last row is what this paper proposes; it is a claim to test, not a shown advantage, and any of the others might be configured to do part of it.

Table 1. One correction and one rule clash under nine ways of keeping memory. Product cells come from their documentation: Mem0 (Chhikara et al., 2025; Mem0, 2026a, 2026b), Zep (Rasmussen et al., 2025; Zep, 2026), MemFS (Letta, 2026), app memory (Anthropic, 2026d; OpenAI, 2026), memory stores and Dreams (Anthropic, 2026b, 2026e). The last row is the design, not a measurement.

Approach	What the next turn reads about the call, and the past	"Copy me" against "nobody in copy"	What it does well	Where it breaks
Bigger window, every message kept in view	"Wednesday" and "Thursday", both, somewhere in a long input; the past only while it fits	Not noticed; the model settles it again on every turn	Nothing to build, and no write step to get wrong	Each turn may settle the clash differently; detail deep in a long input is used less well (Liu et al., 2024)
Running summary or provider compaction	Whatever the latest summary kept; no past	Not noticed; the summary may keep either, both or neither	Cheap, and the window stays small	A loss leaves no trace (J1)
Retrieval over chunks of past text	The best-matching chunks, old and new alike; the past only if chunks carry dates	Not noticed	Keeps the original wording and its source; interprets nothing when it stores	Nothing marks the old chunk as replaced, so it can outrank the new one
Extracted facts, updated against near neighbours (Mem0)	"Thursday", updated in place; the old value stays in a change log with the time of the change	Only if the rule is among the most similar memories (ten in its experiments)	Short, current facts; one model call chooses add, update, delete or nothing	A wrong update replaces a true fact in the live store; the log knows when the value changed, not when the call moved
Facts in a temporal graph (Zep, Graphiti)	"Thursday"; the Wednesday fact ended by an end time, with when each held and when it was ingested	Only if the rule is stored in the graph and found among the related facts	Two timelines; extracted entities linked back to their source episode	The clash check sees only the related facts it retrieves
Markdown files in git (Letta MemFS)	Whatever the agent last wrote, with root files on every turn; the past in git history	Only if the writing agent notices	Readable files; every edit a commit; background edits in separate worktrees	Nothing checks a write before it lands in a file
App memory (Claude, ChatGPT)	A saved line the user can see and edit; the past not documented	Not documented	Simple controls to see, edit and delete	The user has to notice and fix it
Memory stores with Dreams	The agent's edit; a later dream replaces a contradicted entry "with the latest value"; the past through stored versions, kept at least 30 days	After the fact, in batch: the latest wins	An immutable version per change; a content-hash guard on updates; a dream's output is reviewed before use	The latest wins unless the reviewer catches it, even when the older line was the owner's rule
This design	"Thursday"; "Wednesday" kept, superseded from 8 March; the past by valid time, and what was believed by recorded time	Before anything is stored: R-43 is held for the owner	Decides once, when the line is written, and keeps the history	A judge call on every write; holds cost owner time; months before it closes facts on its own (§7.5)

The design borrows from several rows. It keeps Mem0's closed choice of add, update, delete or nothing, and adds the always-loaded rules to what a write is compared with, since we expect

the clashes that hurt most not to be near neighbours (§6.1, measured in §12). It adopts Zep's two timelines (§7.3) and extends its links back to the source to use keys, rules and test fixtures, so an erase can follow them (§8.1). It takes MemFS's isolated background edit and adds the rule for publishing it (§7.2). It shares the memory stores' versions, redaction and update guard (§7.1, §7.2, §7.7). Anthropic's memory tool lets the model ask the application for file operations (Anthropic, 2025), and the gate sits in that handler (§5); LangMem's proposed updates, extracted during the conversation or in the background (LangChain, 2026), would go through the same gate.

Two neighbours belong to the companion papers: running summaries and provider compaction to J1, and retrieval frameworks such as HippoRAG 2 (J2 cites it) to J2. Anthropic's Dreams, a research preview, read a memory store with past transcripts and write a new store that a person reviews before adopting it (Anthropic, 2026e); J5 treats them as a consolidation stage. Dreams settle a contradiction after the fact, in batch, by keeping the latest value; here it is settled when the line is written, by authority and valid time, and the old fact is kept (§8.4). Claude Tag scopes memory by place, one set of notes per channel plus workspace notes (Anthropic, 2026c); our reader scopes are per item (J2 §8.1).

Three research systems are closer still. *A-MEM* (Xu et al., 2025) lets a new note update the descriptions of old ones; we decline the automatic rewrite, since a fact here changes only through a judged revision. *ROAM* (Zheng et al., 2026) classifies each pair of new and stored atomic facts as independent, equivalent, subsuming or conflicting, which maps almost one to one onto our judge's relations (§6.2); we add a clash with an always-loaded rule, and third-party text that would act as an instruction. *Authority collapse* (Zhan et al., 2026) appeared in 48 of the 49 configurations their benchmark, AuthMem-Bench, tested; persisted authority labels cut the unauthorised-action rate in their end-to-end test from 16.9% to zero, and §8 makes such labels decide what a fact may become. *MINJA* (Dong et al., 2025) plants records in an agent's memory through ordinary queries alone; §8 and §7.7 are the write-side answer.

Each piece has precedent. We did not find the combination: a gate that is the only way into the store, a comparison set that includes the rules read on every turn, and a closing step only a judge may take, earned per risk class, counted and alarmed. The products above already ship working integration, background processing and user controls, so the burden is ours: the extra checks have to prevent harmful changes without costing too much delay or owner time. Whether they beat a plain git store like MemFS's is unmeasured (§12).

3. One write, start to finish

A *fact* is one claim about the world, with an ID; a line in a file shows it. A *rule* is an instruction about behaviour. The index file and the rules file are the always-loaded files of §1. A *receipt* is the record the gate leaves for each decision; the *owner* is the person the agent serves; the *judge* is the small model that names how a write relates to what is stored. A fact can end three ways: it is *superseded* when the world changed and *retracted* when it was never true, and both are *closures*, which keep the old fact; it is *erased* when privacy demands that the content go (§7.3). Table 3, at the end of this section, collects the other terms.

Figure 1 follows two writes through the gate. On 9 March the owner says, in passing, that the weekly supplier call moved to Thursday the day before. The write takes five steps.

1. The keeping stage has saved the owner's words, without any wrapper, as entry 118 of the session record (J1 §4.1). The live turn asks to add a line to the meetings file and cites entry 118.
2. The gate gathers what the line could clash with: the meetings file at revision 41, where fact F-0230 says Wednesday; the always-loaded files, which say nothing about the call; and this turn's calendar lookup, which shows Thursday.

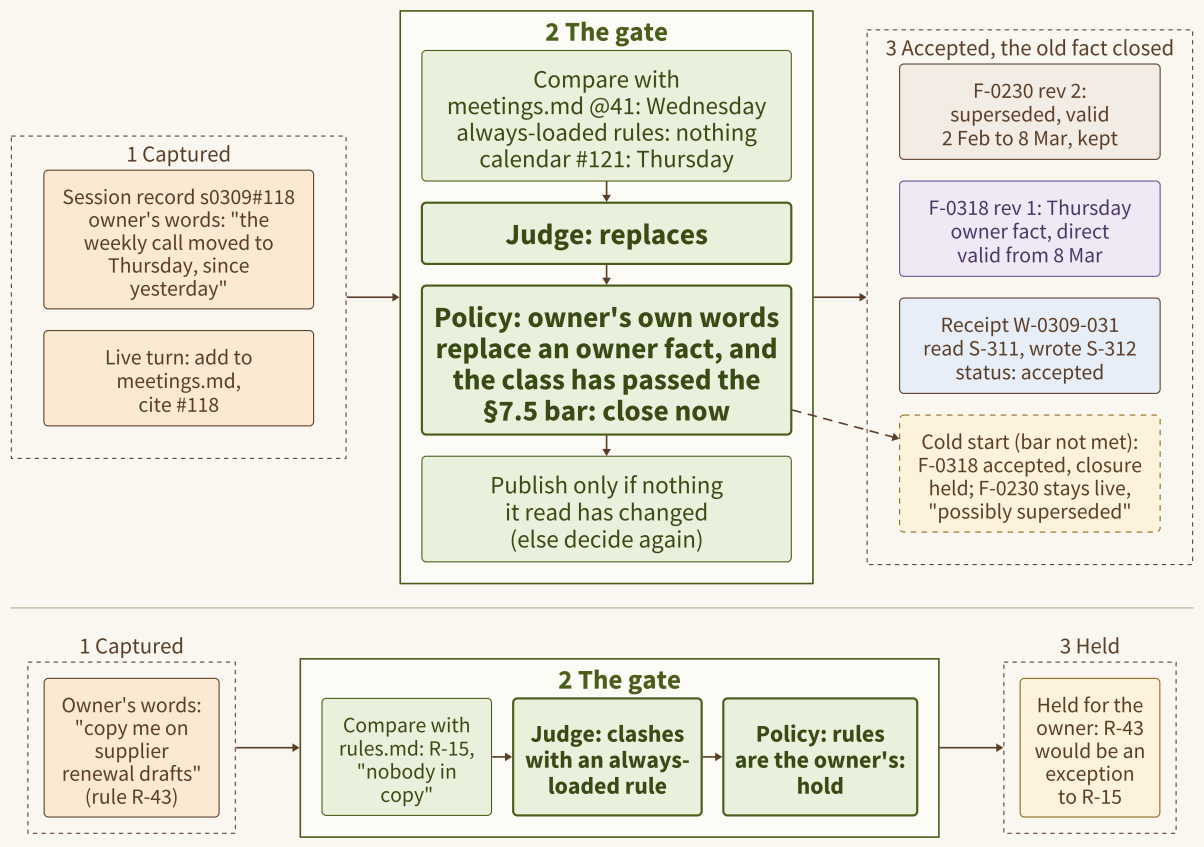


Figure 1. Two writes through one gate. Top: the owner's correction; the judge names "replaces", and because the owner-fact class has passed the closing bar of §7.5, the old fact closes at once (dashed: the cold-start branch). Bottom: a request that clashes with a rule loaded on every turn is held for the owner.

3. The judge answers: *replaces*.
4. Both facts came from the owner, and the owner-fact class has already passed the closing bar of §7.5 on held-out data, so the gate closes F-0230 without asking.
5. The gate saves only if nothing it read has moved. F-0230 gains a second revision that ends on 8 March and points to F-0318, which carries "told by the owner", true since 8 March, recorded 9 March. Nothing is deleted.

In a cold start, before the class has passed the bar, step 4 changes: F-0318 is still accepted, but the closure is held and F-0230 stays live, marked "possibly superseded by F-0318". The owner's word supports the new fact; it does not exempt the closure from the bar.

The lower lane of Figure 1 is a write where the rules do speak. The owner asks, "Copy me on supplier renewal drafts", which would become rule R-43, and the rules file holds R-15: "Send supplier drafts from the sales address; nobody in copy." With a plain append, the request is stored beside the rule and nothing says which wins. With the gate, R-43 is held, and the owner reads: "Captured; held for your approval. Renewal drafts would become an exception to 'nobody in copy'." Keeping a request and applying it are two different events.

When the agent tells the owner about a correction, it says how far the correction has got, and never more (Table 2). So it never says "I'll remember that" about a pending or held correction.

Table 2. What the agent may say about a correction, in the order a correction moves.

Status	What has happened	Who sets it
Captured	His words are safely in the session record	The keeping stage (J1)
Pending	The judge could not run; the change waits, marked unchecked	This gate
Held	The gate needs his decision, and he can see the question	This gate
Accepted	The gate judged it and wrote it into memory; only the owner's own edit may be accepted while still unchecked (§5.2)	This gate
Retrievable	The finding stage can find it	The finding stage (J2)
Active, then enforced	The learning stage applies it, and later checks it in code	The learning stage (J5)

Table 3. The other words this paper uses, one per concept.

Term	Plain meaning	Where
Authority	What a fact may become: instruction, owner fact, evidence, hypothesis or quoted claim	§8.1
Links	What each revision was derived from, so an erase or a quarantine can follow them	§8.1
Owner span	A stretch of a message that the keeping stage marks as the owner's own words, with an address	§5.1, §8.1
Tainted session	A session that has read untrusted text; its writes enter as hypotheses	§5.1
Pending area	Where an unchecked write waits, labelled, while the judge is down	§5.3
Morning report	A daily list, sent by the learning stage, of the questions and changes waiting for the owner	§4.2
Snapshot	One published version of the whole store	§7.2
Erasure counter	A number every erase or quarantine raises before it starts, so work that read older memory cannot save	§7.2, §7.7
Overlay receipt	The finding stage's note that it can now find a change	B.1
Held-out set	Later decisions, never used for tuning, that decide whether a class may close facts on its own	§7.5
Cold start	The months before a class passes that bar; its closures wait for the owner	§7.5
Window controller	The program that decides what the model reads on each turn: the agent's own runtime or the provider's	§7.7
Use key	A short note of when a fact will be needed ("when a hardware part is hard to source")	§9.4

4. Choosing what to write

The write stage starts with the choice to write at all. J1 keeps every session word for word, so a detail left unwritten stays in the record, a search and a read away. A stored line costs little in tokens; its real cost is attention, since every later check on its topic reads it again and every search ranks it against the rest. So a line earns its place when a later session is likely to need it and would otherwise dig it out of the record, which costs at least two turns. What the owner states about his world passes, and so do observations that took real work. The momentary state of the task (which file is open, which step comes next) does not, and content still on disk gets a pointer, never a copy.

4.1 What a live turn writes

A live turn writes at once in three cases: the owner states a fact or a preference; a tool result disagrees with a stored fact (§8.4); or the turn makes a decision, written with its reason. Everything else waits for the nightly pass. The learning stage sorts an owner's correction first (J5 §3.2): a correction about behaviour becomes a rule proposal there, and a correction of a wrong fact comes here as a request to supersede or retract (§7.3).

A session flagged temporary writes nothing here (J1 §4.7). The gate refuses any candidate whose evidence address points into one.

4.2 What the nightly pass picks out

The nightly pass reads the day's session records, never the compaction markers, and skips every record flagged temporary. A model returns candidate facts, each with the record addresses it rests on; a candidate with no address is a guess and is dropped. The pass works on an isolated copy and publishes through the gate (§7.2) in event-time order, with a cap per topic per night; the rest go to the owner as proposals in the *morning report*, the learning stage's daily list of the questions and changes waiting for him (J5 §7.5). Layout fixes (§9) come from the same pass.

5. The only way in

5.1 A gate the store cannot bypass

The gate is code, placed where every write already passes, because a rule in a prompt binds weakly. It is the only writer the storage accepts: the memory store and the always-loaded files belong to the gate's own service account, and the agent's write tool is a request to the gate. A script that tries to write a memory file directly fails at the file system.

Reading is split too. Only full-scope processes (the owner's own sessions, the gate, the nightly pass) may read the store's files. A narrower reader, a sub-agent with a smaller scope or a chat that includes other people, runs without the checkout and reads only through the finding stage's filtered read service (J2 §8.1); its direct file read is refused.

Where accounts cannot be separated, the claim narrows to *mediated* writes: a watcher moves any change the gate did not make to the pending area of §5.3 as an unchecked edit, and no narrower reader runs.

A second barrier does not depend on links at all. A session that has read untrusted text (a web page, an email, another agent's output) is *tainted* for the rest of its life, and its writes enter with at most *hypothesis* authority, whatever their links claim (§8.2). The same idea is behind the read-only mount that Anthropic recommends for stores an agent reads while handling untrusted

input (Anthropic, 2026b): a planted line cannot borrow a session's trust if the session never had write trust to lend.

One web search taints a session for good. If the owner then corrects a fact there, the agent's wording of his correction cannot carry his authority, because a planted page could have bent it. So the gate goes back to the source, as a bank calls you back on your own number after a suspicious email: a fresh model call that has seen nothing else reads only the *owner span* the write cites, the stretch of the message that the keeping stage marks as his own words (J1 §4.1). If it finds the same claim, the write keeps owner authority. If not, the gate shows him both wordings in the same turn, and his reply, in his own words, carries his authority. If he has left, the write is saved as a hypothesis and the nightly pass proposes the owner fact next morning. That pass reads the same records, web pages included; it is safe because it only proposes and he confirms. Test 9 counts how often each path is taken.

5.2 Every writer, one gate

Every writer goes through the same gate; what differs is the authority each one brings. A live turn gets the verdict as the tool result ("superseded F-0230", "held: clashes with R-15"). A sub-agent brings only the authority its parent delegated (§8.2). The owner edits a checkout by hand, and saving publishes through the gate with his authority. He is never blocked: a clash is shown at once and listed the next day, since he may mean to change the rule. With the judge down, his edit is applied and marked unchecked until judged.

5.3 When something fails

When the judge cannot answer (a timeout, a provider error, a quota limit), the gate never applies an unchecked change to anything that is read as authority. Ordinary candidates go to a *pending area*: a separate file per topic, readable with an "unchecked" label, never loaded on every turn, never used to close or contradict anything, and judged again on the current snapshot when the judge returns (§7.2). What can do damage stays held. A pending line or a hold is stored, but it is not accepted memory, and no reader takes it for a fact. Table 4 is the whole policy.

Table 4. What the gate does with each kind of write, and the status it reports.

Write	Judge answers	Judge down or out of time
New fact or rule line, or added detail	Committed: *accepted*	Pending area, unchecked: *pending* ("captured, acceptance pending")
Closure (supersede or retract)	§7.5: *accepted* or *held*	*Held*; the old fact stays live
Change to an always-loaded line, by the agent	*Held* for the owner	*Held*
Whole-file rewrite, file merge or split, delete	Delta check (§9.5), then *accepted* or *held*	*Held*
Owner's own edit, rules file included	Applied: *accepted*; a clash is shown to him	Applied: *accepted*, marked unchecked until judged
Erase for privacy (§7.7)	Runs in full	Runs in full (needs no judge)

The agent reports the status the receipt shows (Table 2). A hold ends only when the owner decides: an approval starts a new try on the same request (Appendix B.1), and a rejection writes a final receipt and stores nothing new. J5's rule ledger holds learning proposals only, outside this store; a rule line is accepted only through this gate.

The live turn waits for the judge only up to a deadline. We have not measured the delay; a small model that reads a few thousand tokens and writes about 150 should answer within

seconds, and test 1 reports it. Past the deadline the write takes the right-hand column of Table 4, so a slow judge costs a pending line, never a stuck turn.

5.4 Receipts

Every decision attempt writes a receipt (Appendix A.2) that names the request, the attempt, the snapshot read and the one produced, the erasure counter and the status. The gate's health is read from receipts: outcomes by kind, holds and pending lines waiting, closures by risk class, deliveries unacknowledged past their deadline. A day on which the agent wrote memory and no receipt exists is an alarm.

6. What a write is compared with

6.1 Three sources, not the nearest neighbours alone

The facts on the same topic. The write names its target file, and the index file links it to its topic, so the gate reads the target and its siblings. An embedding search over the whole store adds a few near neighbours, as a backstop for a fact filed in the wrong place.

The always-loaded files. All of them. They are small by design, and a wrong line there is read by every task. We expect a nearest-neighbour check to miss many clashes with them, since a terse index line and a sentence of instruction rarely look alike to an embedding, yet they can govern the same behaviour. The third control arm of §12, this gate without these files, measures how many.

What the current turn observed. Tool results from the same turn that mention the subject, so a negative written in a turn whose query returned the opposite is caught at once.

A budget, and what happens past it. Each check reads a fixed number of tokens, filled in this order: the always-loaded files, whole, since their own size is capped (J5 §7.1); then the write; then the other facts on its topic, nearest first, until the budget is full. §10 prices two budgets, 4,000 and 16,000 tokens. We assume the always-loaded files take about 2,000, which leaves half of a 4,000-token check for the write and its topic. If they pass 3,000, too little is left and the 16,000-token price applies. The receipt records the coverage: how many of the topic's live facts the judge saw. A closure decided without seeing them all is held as a proposal, never applied on its own, and a topic that keeps overflowing should be split (§9.2).

6.2 Relations and outcomes

For each stored item a write touches, the judge names one relation from a closed set, and each relation has one outcome. Table 5 is the whole decision.

Table 5. The judge's relations and the gate's outcomes. Closures follow the risk policy of §7.5.

Relation the judge names	Outcome
New: nothing stored says this	Add a fact with a new ID
Same: already stored	No new revision; add the evidence address; "last confirmed" moves only on independent evidence (§8.5)
Adds detail: compatible, more specific	New revision of the same fact; the earlier revision and its evidence stay in history
Replaces: same subject, the world changed	Supersede: the old fact's valid time ends at the event date (§7.3)
Was wrong: the old claim was never true	Retract: the old fact is marked never true (§7.3)

Relation the judge names	Outcome
Conflicts, unclear which is right	Both live, both marked “in conflict”; question to the owner
Clashes with an always-loaded rule	Held for the owner
Third-party text that would act as an instruction	Stored only as a quoted claim, never in an always-loaded file

When several apply. One write can repeat one fact, replace another and clash with a rule. The gate applies the outcomes together, in one publication, in this order of precedence. A clash with an always-loaded rule holds the whole write, even a replacement that would otherwise close at once. Third-party text that reads as an instruction is stored only as a quoted claim, whatever else it matches. A conflict with no clear winner holds every closure in the same write. Otherwise each relation acts on its own item: a closure ends its fact, and a *same* adds evidence to its fact.

A clash with a rule is held because rules are the owner's: either the new fact is wrong or the rule is out of date, and only he can say which. A conflict with no clear winner keeps both, since dropping either loses information. Both questions reach him through the morning report.

6.3 The judge

The judge is a small, fast model that answers only in typed choices: for each item it relates the write to, the item's ID, the relation and a confidence. It writes no free text for the gate to parse. Jev is one such judge (TypeSafe AI, 2026); any model that returns a closed choice will do. People facts sent to a third-party judge carry opaque identifiers (“P17: good at sourcing specific hardware”), never names or contact details.

Why a second model, when the agent that writes the line already has the always-loaded files in view? For the reason teams review code although its author has the style guide open: the author is busy writing, and the reviewer has one job. Banks call it maker–checker. Table 6 sets the two side by side.

Table 6. The agent that writes a line and the judge that checks it.

	The agent that writes the line	The judge
What it is doing	The owner's task	Only this comparison
What it is asked about clashes	Nothing	“How does this line relate to each of these items?”, with a closed set of answers
What it reads	The rules, somewhere in a long context (150,000 tokens in §10's example)	The line, its topic and the rules, in about 4,000 tokens
Who acts on the answer	The same model, if it notices	Code (§5.1)

Whether that is enough is a question for data, and the replay already holds it. Test 1 records, for each clash it catches, whether the clashing rule was loaded when the line was written; test 10 compares the gate with the plain git store, where the writer alone decides. If most caught clashes were written with the rule in view, the objection is answered; if few were, we learn where the clashes come from.

The rule borrows from memory research: a recalled fear memory becomes open to change only when recall brings a prediction error (Sevenster et al., 2013). Here a stored claim changes only when the judge names a mismatch, and the receipt records it.

7. Changing a fact without losing it

7.1 A fact has an ID, and a line is only its presentation

A position is not an identity: line numbers move when a note is split, re-flowed or merged, and two writers can hold verdicts about the same old file. So every fact gets an ID when first accepted, never reused, and every change to its claim adds an immutable revision carrying its evidence addresses, links (§8.1), two clocks and status. The line in the file shows the latest claim and ends with the ID, so a person reads a sentence and a program finds the fact wherever the line moves. The record behind Figure 1 holds two facts:

```
fact F-0318  revision 1                    status: accepted
  claim        Weekly supplier call is on Thursday
  topic        meetings          file revision  meetings.md@42
  source       owner             authority      owner fact
  evidence     direct            addresses      s0309#118 (owner's words), s0309#121 (calendar)
  valid        from 2026-03-08   to (open)
  recorded     2026-03-09 10:42
  replaces     F-0230 (now revision 2: superseded, valid to 2026-03-08)
  receipt      W-0309-031 (request s0309#118/0.a, try 1; read S-311, wrote S-312, E3)

fact F-0230  revision 2                    status: superseded
  claim        Weekly supplier call is on Wednesday
  valid        from 2026-02-02   to 2026-03-08
  recorded     revision 1 on 2026-02-02; revision 2 on 2026-03-09
  superseded by F-0318
```

In the file, a person sees the old line struck through, with its end mark on the same line, and the new line beneath it (Table 7). One format everywhere, so ad hoc markers ("WRONG", "see below") never appear: they cannot be counted or queried.

Table 7. The fourth failure of §1 and its fix. A search that stops at its first hit finds the old claim in both files; on the right, the end mark sits on the same line, so it also finds that the claim ended.

	A correction appended below	A correction through the gate
Old line	Weekly supplier call is on Wednesday.	~~Weekly supplier call is on Wednesday.~~ F-0230, ended 8 Mar, see F-0318
New line	Appended at the end of the file: "Correction: the call is now on Thursday."	Weekly supplier call is on Thursday. F-0318
First search hit	The wrong line, with nothing on it	The old claim and, on the same line, that it ended

Ended lines do not stay in the topic file for ever. After a set period, ninety days for instance, they move to the topic's history file, so the file stays short and the judge's budget (§6.1) goes to live facts; their IDs still resolve. A merge of two facts ends both by supersede and points to the new ID; a split gives each part a new ID that points back.

7.2 Publish a snapshot, or nothing

A verdict is only good for what the judge read. If one of those files changes while the judge is working, the verdict may be wrong: it may approve a line against a rule the owner has just rewritten. So the gate saves the way a careful co-author saves a shared document: it notes the version it started from, and it does not save over a newer one. Figure 2 shows the race.

The store is a git repository, and a branch called `live` points at the current version of memory, the *snapshot*. Before the judge runs, the gate writes down everything the verdict will rest on, its *read set*: the revision of each file read (the target, its neighbours on the topic, the always-loaded files), the version of the access and quarantine marks, and the *erasure counter*, a number every erase or quarantine raises before it starts (§7.7). The decision, the new revisions, their links and the receipt then go into one commit, and `live` moves to it only if `live` still points where it pointed when the judge started. Snapshot names such as S-312 are the gate's own numbers, mapped to commits; when an erase rewrites history, the map is rewritten with it, so a receipt never names a snapshot that has gone.

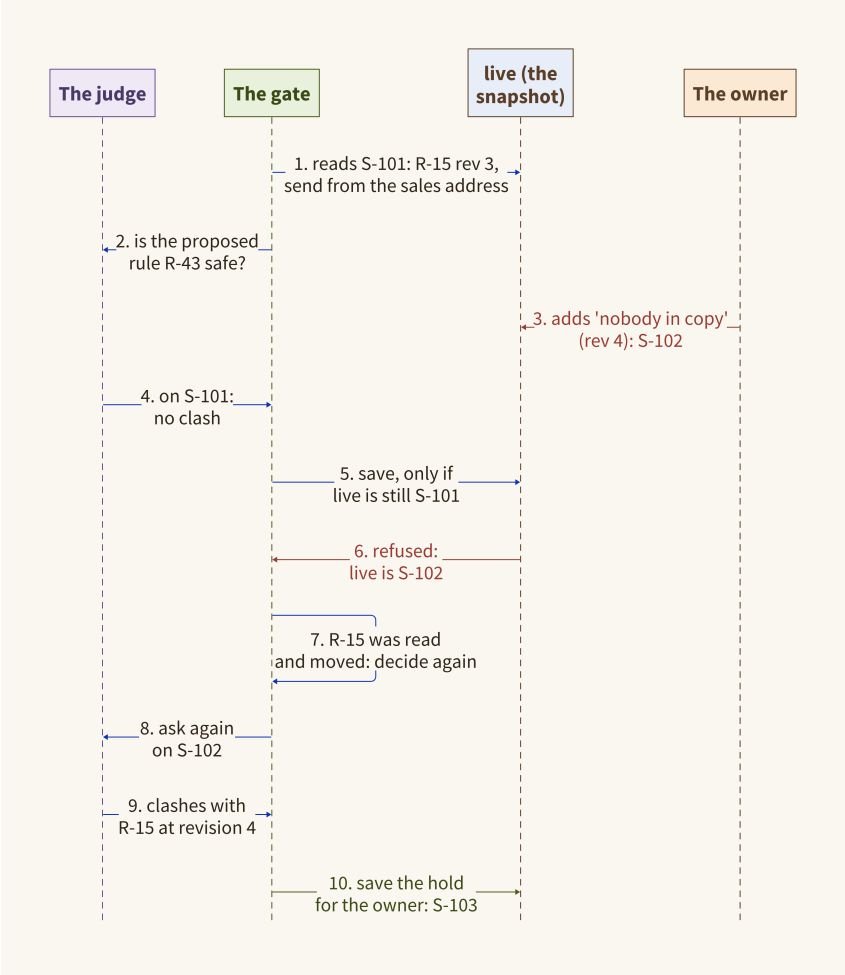


Figure 2. The race that Table 8 settles. The judge works on snapshot S-101, where the owner's rule R-15, at revision 3, says only to send supplier drafts from the sales address. Meanwhile the owner adds "nobody in copy" by hand (revision 4), and S-102 is published. The verdict comes back for S-101, and the save fails because `live` now names S-102. R-15 was in the read set and moved, so the judge runs again on S-102, and the proposed rule R-43 is held on its clash with the new R-15.

If `live` has moved, the gate compares the read set with the new snapshot, and Table 8 says what follows. A verdict reached against a rule the owner has since rewritten must never be saved beside the new rule.

Table 8. What the gate does when the store moved while the judge was working.

What changed after the judge read the store	Can the verdict be reused?	What happens next
Nothing	Yes	Publish, if the checked update still succeeds
An unrelated file, outside the read set	Yes	Apply the change to the new snapshot and try the checked update again
A compared fact or rule	No	Read the new state; run the judge and the policy again
Permissions, quarantine marks or the erasure counter	No	Read the new state; run the judge and the policy again
Publication lost the race three times	No forced write	Hold the change as a concurrent change; acknowledge the hold only once it is stored

Every outcome is saved this way, pending lines and holds included, and nothing is reported as saved until it is; the one exception is a hold for an erased person, which stores no content (§7.7). Appendix B.1 makes a request decided at most once and a saved change reach the finding stage through crashes. Background writers work in an isolated worktree, as MemFS's do, and publish change by change through the same checked update. The store declares its schema version, and a migration is a run of gated writes that keeps every earlier revision.

7.3 Three ways a fact ends

Ending a fact covers three different events, and they must not share a mark.

- *Supersede*: the world changed. The old fact was true and stays true for its period; its valid time ends at the event date.
- *Retract*: the claim was never true (a birth year recorded as 1962 was really 1965). Closing the old fact at today's date would say it used to be true, so a retraction marks it *never true*, keeps it as a record of what the agent believed, and points to the corrected fact.
- *Erase*: privacy. The content goes; only a tombstone and the receipt stay (§7.7). An erased replacement never reopens the fact it superseded: the old fact stays superseded, and its replacement shows as erased.

Accountants work the same way: a wrong posting is never rubbed out; a reversing entry points to it, dated and signed, so the books show both what was believed and what is true.

Every revision carries two clocks, which Zep also separates: *valid time*, when the claim held in the world, and *recorded time*, when the agent held it. The supersede boundary uses the event date when the text gives one and the recorded date when it does not. Valid time also fixes an ordering bug: an old note processed late by the nightly pass would look newest by recorded time and close the true fact; by valid time it is older and goes into history.

A correction is always one of these operations on the wrong fact. A note appended below an untouched wrong line is refused.

7.4 Two questions about the past

The two clocks answer two questions, kept apart (Figure 3).

- *What was true then?* For a date t: the latest revision of each fact, if it is not retracted and its valid time contains t.

- *What did the agent believe then?* For a moment T: the revision of each fact that was live at T by recorded time, retracted ones included, each shown with what later happened to it.

The first answers the owner's "In the week of 2 March, which day was the call?". The second explains a past action: why the agent wished someone a happy 64th birthday instead of a happy 61st. Erased content answers neither question; both return the tombstone.

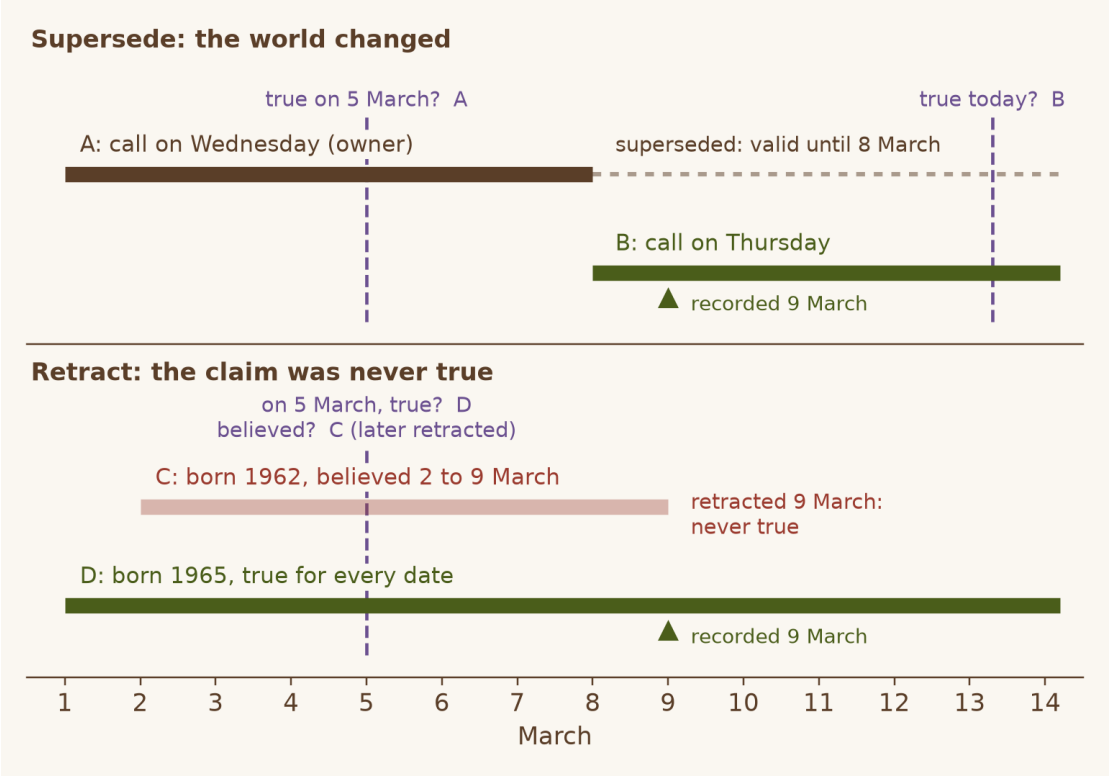


Figure 3. Two ways a fact ends, and two questions about the past. Top: the call moved, so fact A (Wednesday) is superseded on 8 March; what was true on 5 March is Wednesday, today Thursday. Bottom: a birth year was recorded wrongly, so fact C (1962) is retracted on 9 March and replaced by D (1965); what was true is 1965 for any date, and what the agent believed on 5 March is 1962, marked as later retracted.

7.5 Only a judge closes, and only where it has earned it

A false closure hides a true fact from every reader, so closing without the owner is earned per risk class, on data the judge was not tuned on.

Risk classes. Table 9 sets out the ladder. Two cases count one class riskier: a retraction, since it rewrites the past, and any closure in a topic whose facts the agent acts on outside the store (contacts, addresses, accounts), flagged when its file is created, since a wrong closure there acts before anyone reads it. A low-risk fact is then treated as an owner fact, and an owner fact as an always-loaded line, which never closes on its own.

Table 9. Risk classes, from the lowest. A retraction, or a closure in a flagged topic, counts one class higher.

Class	What is in it	May the gate close it on its own?	Otherwise
Low risk	Facts with evidence or hypothesis authority (§8.1), closed by a replacement of equal or higher authority	Yes, once the class passes the bar	A proposal: the new fact accepted, the old one marked "possibly superseded"
Owner facts	What the owner told the agent	Only when the replacement is his own words in the session record and the class passes the bar (Figure 1)	A conflict for the owner
Always-loaded lines	The rules read on every turn	Never	Held for the owner

Sets kept apart. The judge's data follow J5's three streams, development, held-out and evaluation (J5 §9.2), plus a regression suite. Table 10 gives each set its usual name.

Table 10. The four sets the judge's data are kept in.

This paper's name	Usual name	What it holds	What it is for	Who may see its labels
Development replay	Training and validation data	Judged pairs from dated corrections, written by hand	Setting the confidence threshold	Whoever tunes
Regression suite	Unit tests	Every known clash between an always-loaded line and a rule; every past false closure	A new threshold must pass all of them	Everyone
Held-out decision set	Test set	Decisions from a later, separate period	Deciding whether a class may close on its own	Not whoever tunes
Evaluation stream	Production monitoring	Live closures, audited monthly	Checking; never tuning	Not whoever tunes

The bar. A risk class may close facts on its own only when its held-out test supports at least 98% precision, the share of its closures that are correct. We use the one-sided 95% Clopper–Pearson lower bound, counting only verdicts above the confidence threshold. Passing takes at least 149 verdicts with no error, 236 with one, or 313 with two. For contrast, 180 correct out of 200 ("nine in ten") gives a lower bound of 86%: up to one closure in seven could be wrong.

A class not yet tested fails. Below the bar, a closure is written as a proposal: the new fact is accepted, the old one stays live marked "possibly superseded by", and the morning report asks (the cold-start branch of Figure 1).

The first months. Until a class passes, every closure in it is a question for the owner. Assume he makes five corrections a week, about 22 a month; that is an assumption, not a measurement. Even if every one gave a verdict above the threshold, the owner-fact class would need about seven months of later data to reach 149 verdicts with no error, and nearly eleven if one is wrong. Rarer classes may never pass, and then their closures stay with the owner. Meanwhile the cost is his time: about five morning-report questions a week, under two minutes at the 20 seconds a hold takes in §10, plus every change to an always-loaded line, which is held at any stage. The

answers are not wasted. Each approval or rejection is a labelled decision on a closure, made after the threshold was fixed, so if the threshold is frozen before the cold start begins, the cold start fills the held-out set at no extra cost, and the owner who answers is the one who labels. No class starts from a lower, temporary bar: the design accepts a slow start in exchange for never letting an untested judge close facts alone.

Table 11 shows what changes for the owner when a class passes, at the assumed pace: in month 8 the owner-fact class has just passed.

Table 11. Before and after the bar, for the owner.

What happens	Month 1: no class has passed the bar	Month 8: owner facts have passed
The owner says the call moved to Thursday	"Thursday" saved; "Wednesday" stays live, marked "possibly superseded"; the morning report asks whether to close it	"Wednesday" closed from 8 March; the morning report lists it in one line, and the owner can correct it in the inspector (§7.6)
The calendar shows Thursday; the store holds the owner's "Wednesday"	A conflict for the morning report	The same: his systems never close an owner fact (§8.4)
"Copy me on renewal drafts" against "nobody in copy"	Held for the owner	Held for the owner: always-loaded lines never close on their own
A web page says the supplier moved its office	Saved as a quoted claim; closes nothing	The same
The owner's time each week	About five questions, under two minutes, plus rule changes	Rule changes, conflicts, and one line to skim per automatic closure

Who grades the judge. The owner who answers the cold-start questions also grades the judge, and that carries a bias. He sees the judge's proposal ("possibly superseded by F-0318") before he answers, and people shown a machine's proposal accept it more often than they would choose it unprompted (automation bias; Parasuraman and Manzey, 2010); a busy owner may approve in batches. The bar could then pass on agreement instead of correctness. Three rules guard against it. One question in ten is asked blind: it shows the two claims without the judge's choice and asks which is current. An approved closure that the owner later undoes counts as an error against its class. And a class whose audited lower bound falls under 0.98, on the held-out set or in the monthly audit of the evaluation stream, goes back to proposals. Test 2 reports agreement on the blind sample beside the precision.

7.6 Count every closure, and alarm on the suspicious ones

Every closure writes a receipt, and two patterns raise an alarm the first time they occur. One is a closure whose old and new lines come from the same file in the same pass with no change of meaning found: the signature of a closing rule by position, which this design never produces. The other is a burst, more closures in a day than a threshold set from the store's normal rate; it pauses closing without the owner until someone looks, and heads the morning report.

One wrong closure. The alarms catch patterns, not a single plausible mistake: at 98% precision, about one automatic closure in fifty may still be wrong, and one alone never trips the burst alarm. The receipt makes it traceable; three things make it seen. The morning report lists each automatic closure of an owner fact, one line with the old and the new claim. The finding stage's owner inspector shows any fact's history and receipt chain, and a correction made there is a new write through this gate. And the monthly audit of the evaluation stream samples closures (§7.5). Test 5 measures how long one false closure takes to be seen.

Facts that keep flipping. A fact that keeps going back and forth between two versions is often a rule nobody wrote down; the morning report sends such facts to J5's rule ledger.

7.7 Erase and quarantine

Two cases remove content from ordinary use. *Erase* answers a privacy request; *quarantine* answers an attack and keeps the injected content as evidence. This paper decides which applies; how both look in the session record is J1's (J1 §4.4).

Quarantine decides who may see something, not whether it is true. When an entry in the session record turns out to be an attack, say a web page that planted an instruction, the gate marks it, and the mark is copied to everything built from it: facts, use keys, rules and test fixtures, found by following the links of §8.1. Like an erase, a quarantine first raises the erasure counter, so a judge that read the entry before the mark cannot publish its verdict. Quarantined items stay in the store as evidence. Ordinary searches never return them, even with "show hidden" on; only an audit view does (J2 §8.1). If the planted claim is also false, the fact is retracted as well, with the reason "injected". If the planted text is still in a live conversation, the keeping stage takes it out of the model's view where the window controller allows it (J1 §6.6), and the receipt names any copy it could not reach.

An erase removes the content itself, and it runs like a product recall: first stop the sales, then clear the shelves. The gate first raises the erasure counter, so that no judge, background job or index build that read memory before the erase can save its result (§7.2). Then it follows the links from the entries the owner named to every copy and everything derived from them. It never searches by similarity, which could miss a copy or remove the wrong thing. It sends an erase order to each stage that holds a copy and resends until each one answers (Figure 4). The *deletion manifest* (Table 12) lists every place with its answer: done, not present, or out of reach, and why.

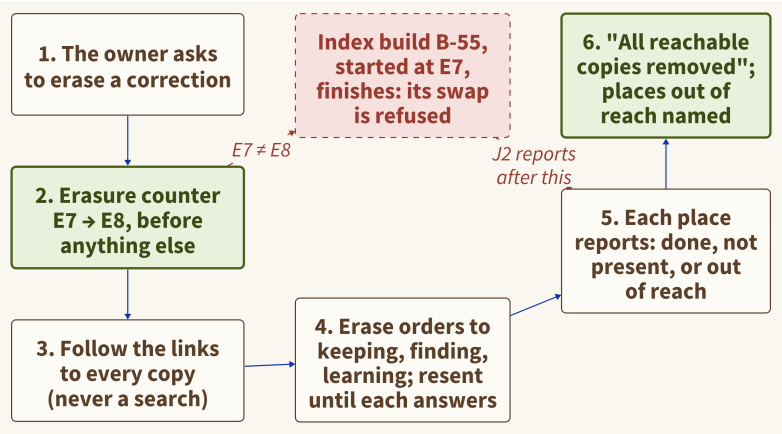


Figure 4. The order of an erase. The erasure counter rises before anything else, so work that read the old state can no longer save; here an index build started before the erase finishes and its swap is refused. The gate then follows the links to every copy, sends erase orders to the keeping, finding and learning stages and resends until each answers. The erase is complete when every place in Table 12 has reported, and places out of reach are named.

Table 12. The deletion manifest. Until every row has reported, J2's included, the erase is *incomplete*; J2's row waits for the refusal of any index build that read the old counter. Then it says "all reachable copies removed" and names the places out of reach. It never claims all copies.

Place	Who acts	How
Fact revisions, index-file lines, pending lines and holds	This gate	Every revision becomes a tombstone that keeps ID and dates; pending lines and holds are dropped
Session record entries or named spans, their blobs, the controller's own transcript	J1	Redacted at the same addresses, span by span (J1 §4.4)
Compaction markers	J1	Rebuilt from the redacted record (J1 §5.4)
Text and asset indexes, fresh writes, use keys	J2	Rows dropped; a build started before the counter rose is never swapped in
Rule evidence, rules and test fixtures	J5	Through J5's deletion interface (J5 §4.3)
Git storage under the store: history, reflogs, unreachable objects and packs, other references, stashes, background worktrees	This gate	Worktrees deleted, history rewritten, unreachable objects pruned; verified by object ID (Appendix B.2)
Backups	This gate	The person's key replaced, the old one destroyed after a recovery checkpoint (Appendix B.2)
Active sessions and provider state	J1	Through the window controller's adapter (J1 §6.6); a provider copy out of reach is named, with its retention terms

Later writes. A new write must not bring erased content back. A write that carries an erased identifier, links to an erased entry, or is tied to an erased person by a matcher is held, and the hold stores no content: its receipt names only the request, the record address and the matching hash, and the question goes to the owner in the session. His confirmation passes this write only, as a new request from the live turn that reads his message again; to stop the holds, he lifts the suppression for that identifier, which the gate records like an erase. A class that is never written (§8.6) is refused before this check, so a government identifier is never stored, not even in a hold. Appendix B.2 gives the hashes and the matcher; the guarantee covers what they catch, and §12 measures what gets through.

7.8 The same ideas under their usual names

Most of this design is known engineering. Table 13 names the pattern behind each mechanism, so a builder can reuse what they know.

Table 13. The design's mechanisms under the names builders already use.

In this paper	Known as	Where
Saving only if `live` has not moved	Optimistic concurrency, compare-and-swap (`git push --force-with-lease`); the content-hash precondition of Anthropic's memory stores	§7.2
One request ID, one receipt per try	An idempotency key	Appendix B.1
The delivery record, saved in the same commit and resent until acknowledged	A transactional outbox	Appendix B.1

In this paper	Known as	Where
The erasure counter checked before saving	A fencing token	§7.2, §7.7
Valid time and recorded time	Bitemporal data (application time and system time in SQL:2011)	§7.3
Revisions never edited in place	An append-only log; event sourcing	§7.1
A reversing record that points to the wrong one	A reversing entry in accounting	§7.3
A key per person, destroyed on erase	Crypto-shredding	Appendix B.2
A tainted session; owner authority only through the clean path	Taint tracking (Perl's taint mode); out-of-band confirmation, as in a bank's call-back	§5.1
A judge separate from the writer	Maker–checker, the four-eyes principle; code review	§6.3
The burst alarm that pauses closing	A circuit breaker	§7.6
Links followed by an erase or a quarantine	Data lineage	§8.1

8. Where a fact came from

8.1 Three fields and the links

How a text arrived says little about whether to trust it: a web page and the owner's calendar both arrive through a tool. So each fact carries three fields.

- *Source class*: who wrote the content. *Owner*: only words that J1 marks as his own (J1 §4.1); a wrapper, an injected instruction or a pasted line, such as a forwarded email's "ignore the previous rule", never counts. A passage he quotes is a *quoted claim* of *third party* class, with the source J1 gives it, never worked out again here. *Owner's systems*: a system of record he controls, read directly (his calendar, his files). *Agent*: its own conclusions. *Third party*: a web page, an email, a document, another agent's text.
- *Authority*: what the fact may become. *Instruction* (only the owner grants it), *owner fact*, *evidence*, *hypothesis*, *quoted claim*.
- *Evidence quality*: how well the claim is supported. *Direct* (read off the source), *corroborated* (two independent sources), *inferred*, *assumed*, or *negative* (an empty result, with the query that produced it).

Table 14 runs the three fields over four ways the agent could learn the same thing.

Table 14. One claim, four sources. The first three rows assume a session that has read no untrusted text (§5.1).

What the agent learns	Source class	Authority	Evidence quality
The owner says, "The supplier call moved to Thursday."	Owner	Owner fact	Direct
The owner's calendar lists the call on Thursday	Owner's systems	Evidence	Direct
The agent infers Thursday from the usual schedule	Agent	Hypothesis	Inferred
The supplier's website says the call is Thursday	Third party	Quoted claim	Direct

Direct means the text was read off its source; it does not make the source authoritative. A browser result stays third party however the page was fetched, and reading a stored fact back does not confirm it: the store is not a source.

Links. Every revision lists what it was derived from: record addresses and, for a merge or an added detail, the revisions of its parts. The finding stage writes the same links for use keys, and the learning stage for rule revisions and test fixtures (schema: Appendix A.1). Erasure and quarantine find descendants by following links, and confirmation (§8.5) uses them to tell new evidence from an echo. Who may read a fact is set by J2: a derived item is readable only by the intersection of its sources' readers, which may be empty (J2 §8.1).

8.2 What each fact may become

The source class sets a ceiling on authority. Only the owner's words can become an instruction or enter an always-loaded file as a rule. Owner's systems give evidence; the agent gives hypotheses; third-party text is a quoted claim ("the vendor's page says the part ships in May") and never loses its quote marks. A sentence crafted to look like an instruction can enter memory only as a quotation. Only the owner raises authority, by confirming a fact in his own words. A rule bound for an always-loaded file must carry why it exists and when it does not apply, or the gate holds it.

Authority survives a change of form by taking the minimum. A merged fact, or a revision that adds detail, has the lowest authority among its parts, and keeps every part's links, so a detail lifted from a web page cannot borrow the authority of the owner's sentence it was merged into. A sub-agent writes with the authority its parent delegated, never more.

8.3 No author

A fact says where its content came from, never which agent, session or model typed the line; the three fields decide trust, not the writer. The session record keeps authors (J1 §4.1); the fact drops them.

8.4 When a stored fact meets an observation

When a turn reads a stored fact and a tool result in the same turn disagrees, the turn writes then and there, and the gate decides with the three fields, not by giving the newer text precedence. A direct reading from the owner's systems closes a stored hypothesis or older evidence under the low-risk policy of §7.5; against an owner fact it opens a conflict for the morning report. Third-party text never closes a fact on its own; at most it opens a conflict.

8.5 Dates, confirmation, negatives and importance

Each fact carries a "last confirmed" date, and meeting the same claim again is not confirming it. A *same* verdict adds the evidence address, but the date moves only when that evidence is independent: owner or owner's systems, not already among the fact's evidence, and with no link leading back to the fact. So an agent's restatement, or a tool reading the memory file, leaves the date in place.

A negative ("there are no items of kind X") is often wrong, since an empty answer has many causes, and expensive to believe, since it stops the search. The gate stores it with evidence quality *negative* and its query, so a later reader runs the query again before repeating it.

Corrections and decisions are marked important when written; how that mark and the confirmation date weigh in ranking is J2's (J2 §5.5).

8.6 What is not written about people

Some classes are never written, whoever says them: credentials and keys, government identifiers, criminal history and immigration status. Others are written about the owner only: health, political views, religion, sexuality and gender identity. About anyone else (a contact, a colleague, a family member) they are off by default, and the owner may switch a class on for a named person. Claude's app memory draws a similar line, keeping such topics out unless the user opts in and never saving government ID numbers, criminal history or immigration status (Anthropic, 2026d). The capture filter of the keeping stage and this gate read one class list, so a class the owner bans is stopped at both doors.

9. The shape of what is written

Five rules for the shape of the store keep the comparison cheap, the ending of a fact exact and the files readable (Figure 5).

9.1 One fact per line

A line that bundles three claims cannot have one end without rewriting the other two, and a rewrite is the kind of write that loses detail. It also keeps the judge's job small: claims against claims, not paragraphs.

9.2 One topic per file, and files hold what is read together

A topic spread over five files costs five reads; a file mixing five topics fills the context with four unneeded. J2 sends this stage evidence about the layout: paths agents reach for that do not exist, and files read together often enough to merge (J2 §8). Each becomes a proposed write through the gate; a merge or split waits for the owner's review in the morning report.

9.3 The index file holds pointers only

The index file has one line per topic or owned asset, such as a skill or a machine: a pointer and a short hook saying when to read the file, never the fact. A fact restated there is a second copy that does not change with the fact, the first failure of §1, so the gate refuses an index line that carries a claim. Which topics get a line, and in what order, is J2's (J2 §4.3).

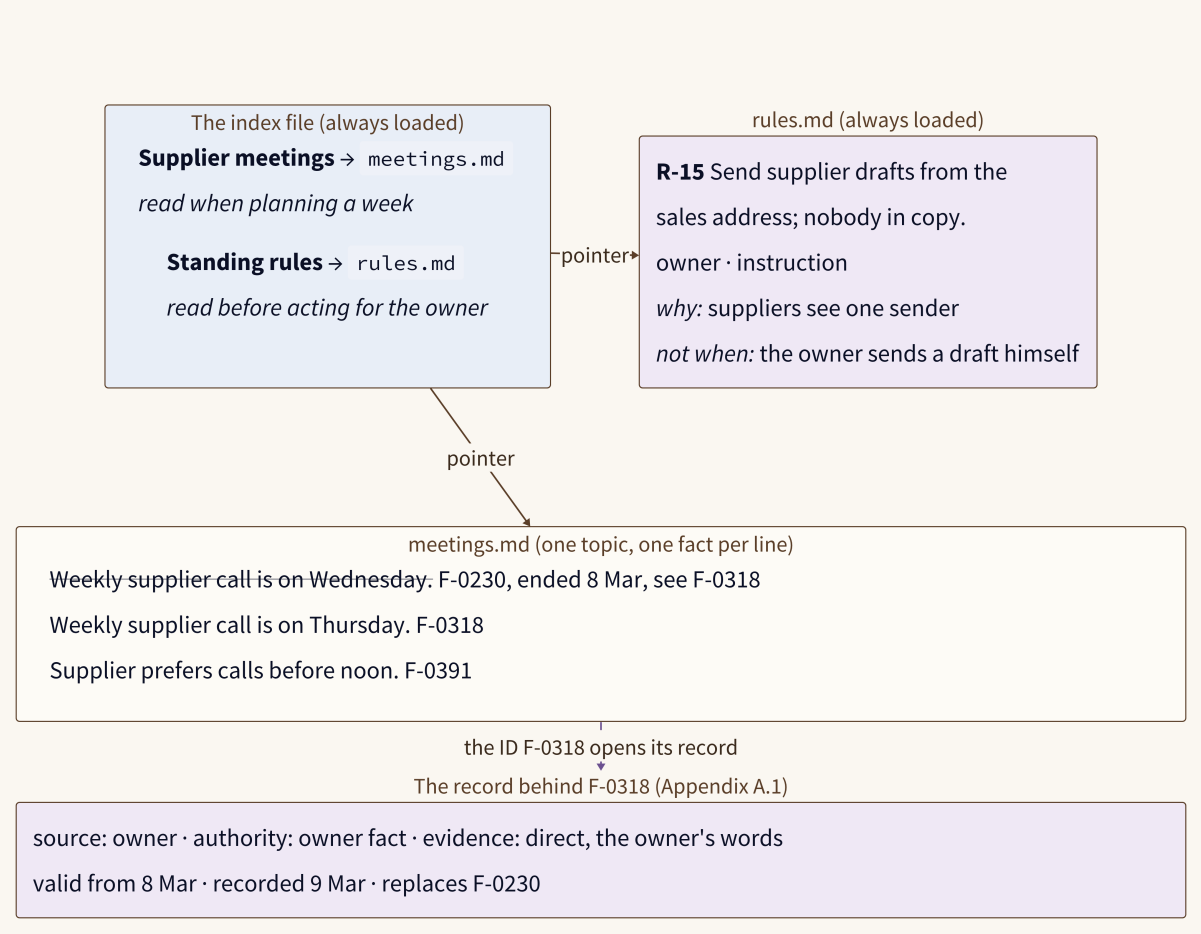


Figure 5. The shape of the store, on the running example. The index file holds one line per topic: a pointer and a hook, never the fact. The meetings file holds one topic, one fact per line; a line shows the claim, its ID and, once the fact has ended, its end mark. Source, authority and dates live in the fact's record (Appendix A.1), shown for F-0318. The always-loaded rules file carries each rule with why it exists and when it does not apply.

9.4 Write the use key with the fact

For facts about people and capabilities, where the gap between what a fact is about and when it is needed is widest, the gate asks for a use key, such as "when a hardware part is hard to source"; J2 indexes and matches them (J2 §5).

9.5 Edit by delta, never by wholesale rewrite

The gate accepts small edits: add a fact, revise one, end one. It holds a write that replaces a whole file unless every fact the old version had is accounted for: kept, revised, superseded, retracted or erased with a tombstone. A model asked to rewrite a file "more concisely" drops details it judges minor, the context collapse that Zhang et al. (2025) describe.

10. What the check costs, and when it pays

In plain numbers: a check costs about five hundredths of a cent, and once the owner's time is counted it pays when more than about 2 writes in 100 would otherwise mislead a turn; how often the owner is asked decides the bill more than tokens do. The rest of this section shows

where those numbers come from.

A check pays only if it saves more than it costs, its own mistakes included. We use the series' notation (J2 §7.1): a cached token is read at r times a model's base input price, an output token costs g times it, and π_s and π_ℓ are the base input prices of a small and a large model, in dollars per token. p, κ, H_m, H_f, f_c and f_h are local to this section.

The price of a check. The judge reads the write and its comparison set, T_c tokens, and returns a typed verdict of $o_j \approx 150$ tokens. With Claude Haiku 5.5 as the small model, at $\pi_s =$ \$0.10 per million tokens and $g = 5$ (Anthropic, 2026a), a check that reads 4,000 tokens costs

$$C_{\text{check}} = (T_c + g\,o_j)\,\pi_s = \frac{(4000 + 5 \times 150) \times 0.10}{10^6} \approx \$0.00048.$$

The price of a wrong turn. With Claude Opus 5.5 at $\pi_\ell =$ \$4 per million, $r = 0.05$ and $g = 5$ (same page), re-reading a cached context of $L = 150{,}000$ tokens and writing $o_t = 1{,}500$ tokens:

$$C_{\text{turn}} = (rL + g\,o_t)\,\pi_\ell = \frac{(7500 + 7500) \times 4}{10^6} = \$0.06.$$

The price of the judge's errors. A *needless hold* costs the owner the time to read and answer it: at an assumed 20 seconds and \$60 an hour, $C_{\text{hold}} \approx \$0.33$. A wrong fact that reaches a turn costs H_m when the gate missed it and H_f when the gate created it by a *false closure*, which hides a true fact. Per write, f_c is the rate of false closures and f_h that of needless holds.

Break-even. Let p be the share of writes that would mislead and κ the share of those the gate catches. The check pays when $p\,\kappa\,H_m > C_{\text{check}} + f_c\,H_f + f_h\,C_{\text{hold}}$, so the break-even share is

$$p^* = \frac{C_{\text{check}} + f_c\,H_f + f_h\,C_{\text{hold}}}{\kappa\,H_m}.$$

Nothing yet estimates p; test 1's replay of facts the owner later corrected gives a first estimate. We take $\kappa = 0.5$, and $f_c = 0.004$: if one write in five were an automatic closure (an assumption) and the bound of §7.5 held at 0.98. That is the steady state. In the first months every closure is a question; those answers buy the held-out labels of §7.5, about two minutes a week, and are not counted here as needless holds. Three ways of pricing a wrong fact give the range (Figure 6). Each charges a wrong fact one wrong turn at least, although the abstract warns that a careless write can mislead every later turn; the low price is deliberate, so a check that pays here pays more in practice.

- *Token floor.* Each wrong fact costs one wrong turn, $H_m = H_f = C_{\text{turn}}$. The model alone sets $p^* \approx 1.6\%$, false closures raise it to 2.4%, and needless holds decide: one per two hundred writes gives about 8%, one per hundred 13.5%.
- *Owner time.* Each wrong fact also costs the owner a minute to notice and correct (\$1), $H_m =$ \$1.06; a hidden true fact is noticed later and corrected twice on average (an assumption), $H_f =$ \$2.06. Now p^* is 1.6% with no holds and 2.3% at one hold per hundred writes.
- *A wrong external action* (a message to an old number, a missed call), at an assumed \$50 per miss and \$100 per false closure: $p^* \approx 1.6\%$ at any hold rate up to one per hundred.

So the break-even share lies between about 1.6% and 13.5% of writes, depending on what a wrong fact costs. At inference prices the holds decide, so holds are kept to the cases that need the owner. As the stakes rise, p^* tends to $f_c H_f/(\kappa H_m)$, 0.8% per unit of H_f/H_m: the check pays only if it catches more harm than its own false closures cause, so the bar of §7.5 matters most where a wrong fact is most expensive. Both rates are measured (§12, test 2). A full comparison budget of 16,000 tokens (§6.1) costs about \$0.0017 a check: the token-floor break-even from the model alone rises from 1.6% to about 5.6%, and with owner time priced from 1.6% to about 1.9%. The judge stays a small model; the large one costs forty times more per check.

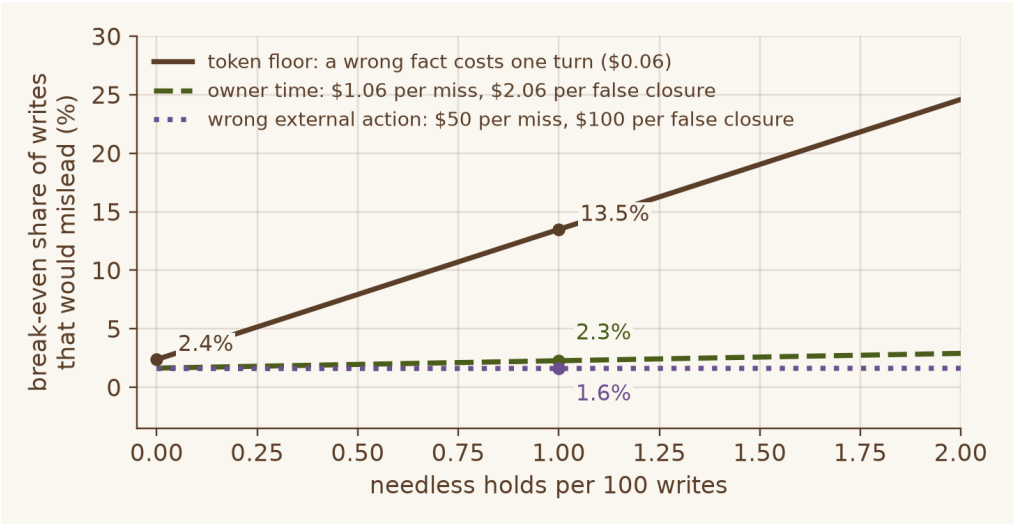


Figure 6. What the check must catch to pay for itself, under three prices for a wrong fact. The vertical axis is the share of writes that would mislead at least one turn; above a curve the check pays. Once owner time or an external action is priced, the curves flatten near 1.6%, set by the false-closure rate. Computed from Anthropic's published prices (October 2026), a judge catching half the bad writes, false closures at the rate the closing bar allows, 20 seconds per hold at $60 an hour, and the assumed costs in the legend.

Two discounts favour the design and are left out of the figures above. The always-loaded part of the comparison is the same on every check, and Haiku 5.5 reads cached input at $0.01 per million tokens instead of $0.10. The nightly pass runs many checks in a row and need not answer at once, so it can also use the Batch API, at $0.05 per million input tokens (Anthropic, 2026a).

11. Conclusions

An agent's memory goes wrong at the moment of writing, and this design fixes it there. One gate is the only way in. It compares each write with the facts on the same topic and with the rules loaded on every turn. Only a judge closes an old fact, and on its own only where it has earned that on data it was not tuned on; the old fact is kept with why it ended. Nothing is deleted except by an erase, and every line says where it came from. Table 15 puts the design back in the terms of the abstract's notebook. If the tests of §12 bear it out, an agent's memory can be trusted the way that notebook is: a correction made once stays made.

Table 15. The assistant's notebook of the abstract, mechanism by mechanism.

What the good assistant does	What the gate does	Where
Strikes "Wednesday" and writes "Thursday" beneath, with the date and who told her	Supersede: the old fact kept, valid until 8 March; the new fact with its source and dates	§7.1, §7.3
Writes "never true" beside a birth year she got wrong	Retract	§7.3
Reads the first page before writing anything that could clash with it	Compares every write with the always-loaded files	§6.1
Asks you before changing anything pinned to the first page	Holds every change to an always-loaded line for the owner	§5.3, §6.2
Writes "the vendor's site says…" in quotes, never as your instruction	Third-party text stays a quoted claim	§8.2
When your message reaches her through someone else's hands, rings you on your own number to check	A correction made in a tainted session keeps owner authority only through the clean path	§5.1
Has a new colleague's changes countersigned until she has proved herself	Closes facts on its own only after the class passes the bar	§7.5
Tears out a page when you ask her to forget something, and notes that a page was removed	Erase, with a tombstone	§7.7

The limits are these. The judge's precision is unknown until measured, and whether holds or false closures cost more depends on what a wrong fact costs (§10). Until a class passes the bar, which at one owner's pace may take most of a year, its closures wait for him, and the owner who answers them also grades the judge, a bias the blind sample of §7.5 reduces but does not remove. Topic links are only as good as the index file, and a topic past the judge's budget is compared only in part. A session that read untrusted text needs the owner to confirm his own correction: at once if he is there, the next morning if not. Without its own account the gate guarantees mediated writes only. Suppression catches erased identifiers exactly and indirect references only as well as the entity matcher does. A provider copy, a clone outside the gate or a backup written in the clear can sit out of reach of any erase; the manifest names it, which is all a client can do. And the gate does nothing for a fact that is stored correctly and never found (J2), or a rule that does not bind (J5).

12. Next steps

The gate will be built and tested on our own agent, with the sets of §7.5 kept apart wherever a test sets a threshold. Three control arms run on the same store and replay: a nearest-neighbour check built like Mem0's published pipeline (§2), which is a control and not a benchmark of the Mem0 product; the plain git store of MemFS with no check; and this gate with the always-loaded rules left out of the comparison, which tests whether that addition earns its cost. The delivery and erase-ordering cases are also checked against a model of the design (J5 Appendix B).

1. *Catching the clashes that matter.* Replay each original write against the store as it stood that day: the regression suite and the later-corrected facts of the evaluation stream. Pass: every suite clash flagged, at least half the later-corrected facts, precision of at least 0.9 on a hand audit, and better than the nearest-neighbour arm on both. Include writes that repeat one fact, replace another and clash with a rule at once: the rule clash holds the

whole write. Then grow one topic past the judge's budget and report coverage, latency and cost per check; a closure decided on partial coverage is held. The replay also gives a first estimate of p, the share of writes that would mislead (§10). For each clash caught, report whether the clashing rule was loaded when the line was written (§6.3).

2. *Closing and holding.* On the held-out decision set, measure each class against the bar of §7.5 and count needless holds and false closures per hundred writes. Pass: no class closes below the bar (a cold start accepts an owner's correction and holds its closure), every threshold passes the regression suite, and needless holds stay under one per two hundred writes. Report the owner's agreement with the judge on the blind sample of §7.5 beside the precision.
3. *Identity, concurrency, snapshots and delivery.* Nine steps, listed in Appendix B.3. Pass: re-flowing or splitting a file closes nothing, a request is decided at most once, a late verdict on a moved rule is refused and decided again (Figure 2), delivery survives crashes and lost messages, and nothing is acknowledged before it is stored.
4. *Supersede, retract and confirm.* A moved meeting and a wrongly recorded date give the answers of Figure 3; a restated fact, and a tool reading its file, leave "last confirmed" where it was.
5. *The alarm fires on day one, and a single error is still seen.* A burst of same-file, same-pass closures: the alarm fires on the first, and closing without the owner pauses. Then plant one plausible false closure above the bar, with no alarm: it is in the next morning report, the inspector shows its history, and the owner's correction restores the old fact by a new write. Report the days until it was seen.
6. *Authority survives the write.* As in AuthMem-Bench (Zhan et al., 2026), pair facts that differ only in source and merge each into an owner fact. Pass: the third-party twin never reaches an always-loaded file, the merged fact keeps the lower authority, and no dependent action runs unapproved.
7. *A failing judge.* Cut the judge off mid-session and let the learning stage retry while its requests are pending. Pass: every write lands in its row of Table 4, nothing unchecked closes a fact or touches an always-loaded file, and on the judge's return each pending request is resumed as a new try linked to its earlier receipt and ends with a final answer, as does a hold the owner approves. The test must fail on a retry that returns the cached pending receipt, which would leave the Thursday fact pending for ever, and on a resume that runs after a retry has already settled the request.
8. *An erase leaves no reachable copy.* Nine steps, listed in Appendix B.4. Pass: every step holds, from the counter rising first to the last held write leaving no object that contains the erased identifier. For a new alias or a description without identifiers, report the share held with its confidence interval and claim nothing beyond it.
9. *Restrictions and special classes hold for every reader.* Quarantine a planted entry from which a fact, a merged fact and a use key were derived: ordinary and show-hidden queries return none of the three, and only the audit view shows them. A narrower sub-agent's direct read of the store's files is refused. A session that read a planted web page writes only hypotheses; in that session, an owner's correction keeps owner authority when the clean extraction matches it, and otherwise once he confirms it; the test counts how often he is asked and how many corrections wait for the nightly pass. A health detail about a contact is not written; the same detail about the owner is, when he allows it.
10. *The outcome that counts.* Run the plain git store in shadow on the same live sessions: both stores receive every write, and only the gate's store serves the agent. For each turn, count in each store whether the line the turn would read contradicts the owner's latest correction. Pass: fewer such turns with the gate, the 95% interval for the difference excluding zero. The share of the owner's prompts that repeat a correction is counted on the live arm. One owner and one agent make this an exploratory estimate, not proof of a general effect.

References

Anthropic (2025). *Memory tool.* Claude Developer Platform documentation, read 12 October 2026. https://platform.claude.com/docs/en/agents-and-tools/tool-use/memory-tool

Anthropic (2026a). *Pricing.* Claude Developer Platform documentation, read 21 October 2026. https://platform.claude.com/docs/en/about-claude/pricing

Anthropic (2026b). *Using agent memory* (memory stores). Claude Platform documentation, Managed Agents, read 21 October 2026. https://platform.claude.com/docs/en/managed-agents/memory

Anthropic (2026c). *How Claude Tag works.* Claude documentation, read 21 October 2026. https://claude.com/docs/claude-tag/concepts/how-it-works

Anthropic (2026d). *Claude's memory works everywhere, and you decide what's in it.* Claude blog, 25 August 2026. https://claude.com/blog/claudes-memory-works-everywhere-and-you-decide-whats-in-it

Anthropic (2026e). *Dreams.* Claude Platform documentation, Managed Agents, research preview, read 21 October 2026. https://platform.claude.com/docs/en/managed-agents/dreams

Chhikara, P., Khant, D., Aryan, S., Singh, T., and Yadav, D. (2025). *Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory.* arXiv:2504.19413. https://arxiv.org/abs/2504.19413

Dong, S., Xu, S., He, P., Li, Y., Tang, J., Liu, T., Liu, H., and Xiang, Z. (2025). *Memory Injection Attacks on LLM Agents via Query-Only Interaction.* arXiv:2503.03704. https://arxiv.org/abs/2503.03704

LangChain (2026). *LangMem: conceptual guide.* LangMem documentation, read 21 October 2026. https://langchain-ai.github.io/langmem/concepts/conceptual_guide/

Letta (2026). *MemFS.* Letta documentation, read 12 October 2026. https://docs.letta.com/concepts/memfs

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2024). Lost in the middle: how language models use long contexts. *Transactions of the Association for Computational Linguistics* 12, 157–173. https://arxiv.org/abs/2307.03172

Mem0 (2026a). *Update memory.* Mem0 documentation, read 21 October 2026. https://docs.mem0.ai/core-concepts/memory-operations/update

Mem0 (2026b). *Memory history.* Mem0 API reference, read 22 October 2026. https://docs.mem0.ai/api-reference/memory/history-memory

OpenAI (2026). *Memory in ChatGPT.* OpenAI Help Center, read 21 October 2026. https://help.openai.com/en/articles/8590148-memory-in-chatgpt

Parasuraman, R., and Manzey, D. H. (2010). Complacency and bias in human use of automation: an attentional integration. *Human Factors* 52(3), 381–410. https://doi.org/10.1177/0018720810376055

Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., and Chalef, D. (2025). *Zep: A Temporal Knowledge Graph Architecture for Agent Memory.* arXiv:2501.13956. https://arxiv.org/abs/2501.13956

Sevenster, D., Beckers, T., and Kindt, M. (2013). Prediction error governs pharmacologically induced amnesia for learned fear. *Science* 339(6121), 830–833. https://doi.org/10.1126/science.1231357

TypeSafe AI (2026). *Introducing System One models and Jev.* https://typesafe.ai/blog/introducing-system-one-models-and-jev

Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. (2025). *A-MEM: Agentic Memory for LLM Agents.* arXiv:2502.12110. https://arxiv.org/abs/2502.12110

Zep (2026). *Adding episodes.* Graphiti documentation, read 12 October 2026. https://help.getzep.com/graphiti/core-concepts/adding-episodes

Zhan, Q., Zhang, R., Guo, S., Zhao, L., and Liu, Z. (2026). *When Memory Becomes Authority: Benchmarking Authority Collapse at the Memory Consolidation Boundary.* arXiv:2608.01679. https://arxiv.org/abs/2608.01679

Zhang, Q., Hu, C., Upasani, S., Ma, B., et al. (2025). *Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models.* arXiv:2510.04618. https://arxiv.org/abs/2510.04618

Zheng, J., Lai, P., Cheng, S., Zhao, J., Yang, L., and Chen, G. (2026). *ROAM: Robust Organization of Atomic Memories for Agents through Semantic Relations.* arXiv:2609.09778. https://arxiv.org/abs/2609.09778

Appendix A. Interfaces and the write path

A.1 A fact revision

Table 16. A fact revision. The line in the file shows the claim, the ID and, once the fact has ended, the end mark; everything else lives here.

Field	Meaning
id, revision, claim, topic	Stable ID, never reused; revision +1 on each change; one claim, and the topic file that presents it
source_class	owner (owner-authority spans only), owner_system, agent, third_party (quoted spans too)
authority	instruction, owner_fact, evidence, hypothesis, quoted
evidence_quality	direct, corroborated, inferred, assumed, negative (with query)
evidence	Record addresses (J1 §4.2)
derived_from	Links: record addresses and the revisions of every part; J2 and J5 write the same field on use keys, rule revisions and fixtures
valid, recorded (from/to)	When the claim held in the world (empty once retracted); when this revision was the live one
status	accepted, pending, held, conflict, superseded, retracted, erased
restrictions	quarantine and its source entry, inherited via derived_from
replaces, replaced_by	Fact IDs on either side of a supersede, retraction, merge or split
subjects, last_confirmed, important, use_keys	Opaque IDs of the people concerned, which choose backup keys (Appendix B.2); §8.5 (last_confirmed moves only on independent evidence); §9.4

A.2 A write receipt and a delivery record

Table 17. A write receipt. It never holds the content of a fact.

Field	Meaning
request_id	Stable across tries: the learning stage's child operation ID (correction ID plus a repair letter), or, for a write without one, a hash of evidence addresses, topic and text
receipt_id, attempt, previous_receipt	One immutable receipt per decision attempt, linked to the one before. A retry gets a final answer back; a pending receipt is never returned, and the retry runs the next attempt
correction_id	The series' ID for an owner correction (message ID and span index), when the write carries one
read_snapshot, read_set, epoch	Snapshot the decision read; the revisions of every file and mark it compared; the erasure counter
new_snapshot	Snapshot published, or empty when nothing was
relation, confidence, judge	The verdict and the judge's model and version

Field	Meaning
policy	Risk class and rule that applied (close on its own, proposal, hold)
touched	Fact IDs and revisions before and after
status, reason	accepted, pending, held, refused, erase incomplete, all reachable copies removed; and why

A receipt is never edited. J2 acknowledges a retrievable fact (ID@revision) in its own overlay receipt, keyed by receipt_id (J2 Appendix A, §8.2), so the two receipts show the chain from accepted to retrievable and this one never claims the finding stage's status. What J2 acknowledges is the *delivery record* of Table 18; Appendix B.1 explains how it travels.

Table 18. A delivery record, written in the same commit as the change.

Field	Meaning
receipt_id, snapshot	The receipt it announces and the snapshot that carries the change
items	fact_id@revision pairs, or the ID of the pending-area line or hold record; never content. J2 reads the text from the snapshot the record names, where an erased revision is a tombstone and a dropped pending line or hold reads as gone
kind	accepted change, pending line or hold. All three are stored in a published snapshot with their receipt; only the first is accepted memory. A hold for an erased person carries no text
acknowledged_by	J2's overlay receipt ID; empty until it arrives, and the dispatcher resends while it is empty
next_resend	The dispatcher's timer; a resend is a no-op at J2, which keys by ID and revision

A.3 The gate in pseudocode

```
WRITE(candidate, writer, resume=false):
  req  <- request_id(candidate)                          # stable across attempts
  prev <- latest_receipt(req)
  if prev and prev.status in FINAL and not resume:     # accepted, refused, hold awaiting owner
    return prev                                          # a PENDING prev is never returned
  if temporary(candidate.evidence): return REFUSE("temporary")   # J1 §4.7, never kept
  snap <- read_live()                                    # snapshot id, with its erasure counter
  d    <- DECIDE(candidate, writer, snap)                # COMMIT, PENDING, HOLD or REFUSE
  for attempt in 1..6:
    if attempt > 3 and d.kind == COMMIT:                 # three lost races: hold, never force
      d <- HOLD(d.change, "concurrent change", d.read_set)
    tree <- tree_of(snap) + d.artifact       # revisions and links, or pending line or hold
              + receipt(d, req, prev) + delivery_record(d)       # IDs and revisions only
    if update_ref("live", new = commit(tree), expect = snap):    # one atomic publication
      return receipt_of(tree)          # accepted, pending or held: acknowledged once stored
    now <- read_live()
    if changed(d.read_set, snap, now):         # a compared file, scope, mark or counter
      d <- DECIDE(candidate, writer, now)      # judgement AND policy again; any outcome
    snap <- now                                # otherwise re-apply on the new tip
  return RETRY(req)                    # nothing stored, nothing acknowledged; same request ID

ON_JUDGE_RETURN():                 # resume every pending request, none from cache
  for req in pending_requests():
    if latest_receipt(req).status == PENDING:          # a retry may have settled it already
      WRITE(candidate_of(req), writer_of(req), resume=true)
ON_OWNER_APPROVAL(hold):           # the approval is new evidence
  if latest_receipt(hold.req) is not hold: return      # a retry settled it already
```

```
  if hold.content_free:                                  # §7.7: nothing stored to re-run
    WRITE(reread(hold.address) + confirm(hold.hash), owner)  # new request, this write only
  else:
    WRITE(candidate_of(hold.req) + approval(hold), owner, resume=true)
ON_OWNER_REJECTION(hold):
  if latest_receipt(hold.req) is hold:
    store_final_receipt(hold.req, REFUSE("owner"))     # stores nothing new

DISPATCH():                                              # always running; survives a restart
  for rec in delivery_records() where not acknowledged(rec): send_ids(rec, to = J2)
  # J2 reads content from the live snapshot; its overlay receipt closes the record

DECIDE(candidate, writer, snap):
  facts    <- same_topic(snap) + near_neighbours(candidate) + always_loaded(snap)
  read_set <- revisions(facts) + marks_version(snap) + snap.counter
  cls      <- span_class(candidate.evidence)   # J1 span: owner, or quoted -> third_party
  if banned_class(candidate): return REFUSE("class")   # §8.6, first: not even held
  hit <- suppression_check(candidate)         # identifiers, links, uncertain match
  if hit != CLEAR and not confirmed(candidate, hit.hash):   # owner's yes, this write
    return HOLD_WITHOUT_CONTENT(read_set, hit)  # §7.7: request, address, hash; no text
  verdicts <- judge(candidate, facts, tool_results_this_turn(candidate.subject))
  if verdicts is TIMEOUT: return route_by_failure_matrix(candidate, read_set)  # Table 4
  change  <- outcomes(verdicts)                 # one per item, Table 5; precedence §6.2
  tainted <- writer.session.tainted and not clean_extraction_matches(candidate)  # §5.1
  if tainted and cites_owner_span(candidate) and owner_present():
    return HOLD(change, read_set, ask="now")    # §5.1: both wordings, same turn
  change.authority    <- min_authority(candidate.parts, writer.delegated, cls,
                          HYPOTHESIS if tainted else None)
  change.restrictions <- union(restrictions(candidate.derived_from))  # quarantine
  if needs_hold(change, verdicts, risk_class(change), coverage(facts)):  # §7.5, §6.1
    return HOLD(change, read_set)
  return COMMIT(change, read_set)
```

A.4 This paper's handoffs, mapped to J5's journey

The series follows one adversarial journey through all four stages, with its own example (a contact's mobile number), and J5 prints it as a table of 24 handoffs (J5 §10 and Appendix B). Table 19 maps the handoffs this paper designs to J5's IDs, for readers who have J5.

Table 19. The handoffs this paper designs, with the IDs J5 gives them.

Handoff	Where it is designed here	J5 ID
A correction of a wrong fact arrives as a request to supersede or retract	§4.1	T3
Judge down: new fact pending, closure held	§5.3	T4
The owner edits a rule by hand while the judge is down	§5.2	T5
A compared rule moved; the learning stage withdraws the stale proposal	§7.2	T6
Judge back: each pending request resumes as a new try	§7.2, B.1	T7
Fact accepted, old fact superseded	§3, §7.5	T8
Delivery to the finding stage, and its acknowledgement	B.1, A.2	T9
A proposed rule held on its clash with an always-loaded rule	§6.2	T11
The hold recorded for the finding stage	B.1	T12
Erase starts: counter raised, links followed	§7.7	T14

Handoff	Where it is designed here	J5 ID
Erase orders to the keeping, finding and learning stages	§7.7	T15, T17, T18
An index build that read the old counter is refused	§7.7	T19
Quarantine reaches the live window, the finding and the learning stages	§7.7	T22 to T24

Appendix B. Building the store

B.1 Requests, retries and delivery

A request keeps one ID through every retry: the learning stage's ID for that repair of a correction (J5 §3.2), or, for a write without one, a hash of its evidence addresses, target topic and text. Each try leaves its own receipt, linked to the one before. When a request arrives again, the gate reads its latest receipt. If that holds a final answer (accepted, refused, or held for the owner), the gate returns it and writes nothing, so a retry never saves a fact twice. If it says pending, the gate tries again on the current snapshot.

The judge coming back, and the owner approving a hold, start a new try the same way, but only after the gate checks that the latest receipt is still pending, or the hold still undecided: if a retry has already settled the request, the resume does nothing. A hold that stores no content (§7.7) is the exception: there is nothing to try again, so his approval sends a new request from the live turn, which reads his message again. Verdicts from older tries are discarded. The finding stage likewise records which snapshot each index build used and checks the erasure counter before installing it (J2 §8.3).

Saving a fact is not the same as being able to find it. The commit that saves a change also carries a *delivery record* for the finding stage: fact IDs, revisions and the receipt ID, never the text (Table 18). A dispatcher resends it until J2 answers with its *overlay receipt*, its note that it can now find the change (J2 §8.2). If either side crashes, nothing is lost, because the record is part of the commit; a lost message is resent on a timer, with no restart; a duplicate does no harm, because J2 keys by ID and revision. Because the record carries IDs and never content, J2 reads the text from the snapshot, so a late duplicate cannot bring back a revision erased in between. A pending line or a hold travels the same way and stays marked as what it is.

B.2 Git, backups, restore and suppression

Git cleanup. Before rewriting, the gate lists the object IDs of every blob and tree that held an erased revision. The manifest row reports done only when none exists in any store, reflog or worktree the gate controls. A clone or mirror outside the gate is named as out of reach. The map from snapshot numbers to commits is rewritten with the history.

Backups. A backup never holds a record about a person unencrypted. Each record is encrypted with that person's key; a record about several people uses a key derived from all of theirs, so destroying any one makes it unreadable. An erase replaces the person's key, and destroying the old key makes that person's records unreadable in every older backup, retained facts of shared records included.

So before the old key is destroyed, the gate makes a *recovery checkpoint*: a backup of the rewritten store with the erased content gone and shared records re-encrypted under the remaining people's keys, then a test restore that reads back every retained fact ID. If the checkpoint cannot be made (say the backup target is offline), the manifest row reports a *recovery gap*: the retained fact IDs that only older backups hold. The owner chooses: destroy the key now and accept the gap until the next backup, or leave the erase incomplete until the checkpoint exists.

Restore. Tombstones, the erasure counter, quarantine marks and the suppression identity are kept outside the backups and restored from their newest copy. After a restore the store stays closed until all erases, suppressions and quarantines up to the current counter are replayed over it.

Suppression. For each erase the gate keeps a *suppression identity*: keyed hashes of the erased identifiers (names, phone numbers, addresses, after normalisation), computed with a secret only the gate holds. A later write carrying one of them is held however it is worded. A new alias, a pronoun or "the supplier we

met yesterday" slips past that exact check; for those, a candidate whose links reach an erased entry is held, and an entity matcher compares the rest with a short sealed description of the person (role and relation, never contact details) and holds any candidate it ties to an erased person with less than full confidence as an *uncertain match*. Every such hold carries no content (§7.7); if the owner declines, nothing about the write was stored. Other stages call the same check, which answers *held* or *clear* (J5 §4.3).

B.3 Test 3, step by step

1. Index a file twice, split a long note, re-flow a file: no closure, and every fact ID resolves.
2. Run two writers on one snapshot and kill the gate between verdict and publication: one write wins, and the other is decided again or held.
3. Retry a request that already has a final answer: nothing new is written.
4. While a verdict is out, the owner rewrites a compared rule (Figure 2): the late verdict is refused, the decision runs again, and the proposed rule R-43 is held on its clash with R-15 at revision 4.
5. Kill the gate right after publication; separately, drop the delivery message with no restart. Both times J2's overlay receipt arrives and the delivery record closes.
6. Erase a person between a hold decision and its storage: the hold is decided again and stores nothing erased.
7. Make the hold's own storage lose every race: nothing is acknowledged, and a retry with the same request ID stores it.
8. Let a re-decision after a lost race turn a pending outcome into a commit: it commits through the same checked update.
9. The test must fail on a delivery message sent after the update, outside the commit, and on a delivery record that carries content, which would bring an erased fact back.

B.4 Test 8, step by step

1. Let facts about a made-up person reach every row of Table 12. Seed a background worktree, a reflog entry and an unreachable object with their text, share one fact with a second person, and start an index build.
2. Erase. The erasure counter rises before anything else, and the old build is never swapped in.
3. Every row reports done, not present, or out of reach by name; J2's row only after the old build's refusal.
4. Every recorded object ID is gone, searches find only tombstones, every pointer resolves, and an erased replacement does not reopen the fact it superseded.
5. The checkpoint restores the shared fact; with the backup target offline, the row names the recovery gap.
6. Restore the pre-erase backup: the store stays closed until the replay ends, and nothing erased is readable.
7. Send later writes carrying an erased identifier: all are held, and none leaves a new object containing it in any store, reflog, worktree or index. For a new alias or a description without identifiers, report the share held with its confidence interval.
8. Send a later write carrying the erased person's government identifier: it is refused as a class never written, not held.
9. The owner confirms one held write: it is stored as a new request, and a second write about the same person is held again. After he lifts the suppression for that identifier, the next write is not held.

J-series paper J14 | Version 2.7 | 2026-10-21