SHAKUDO

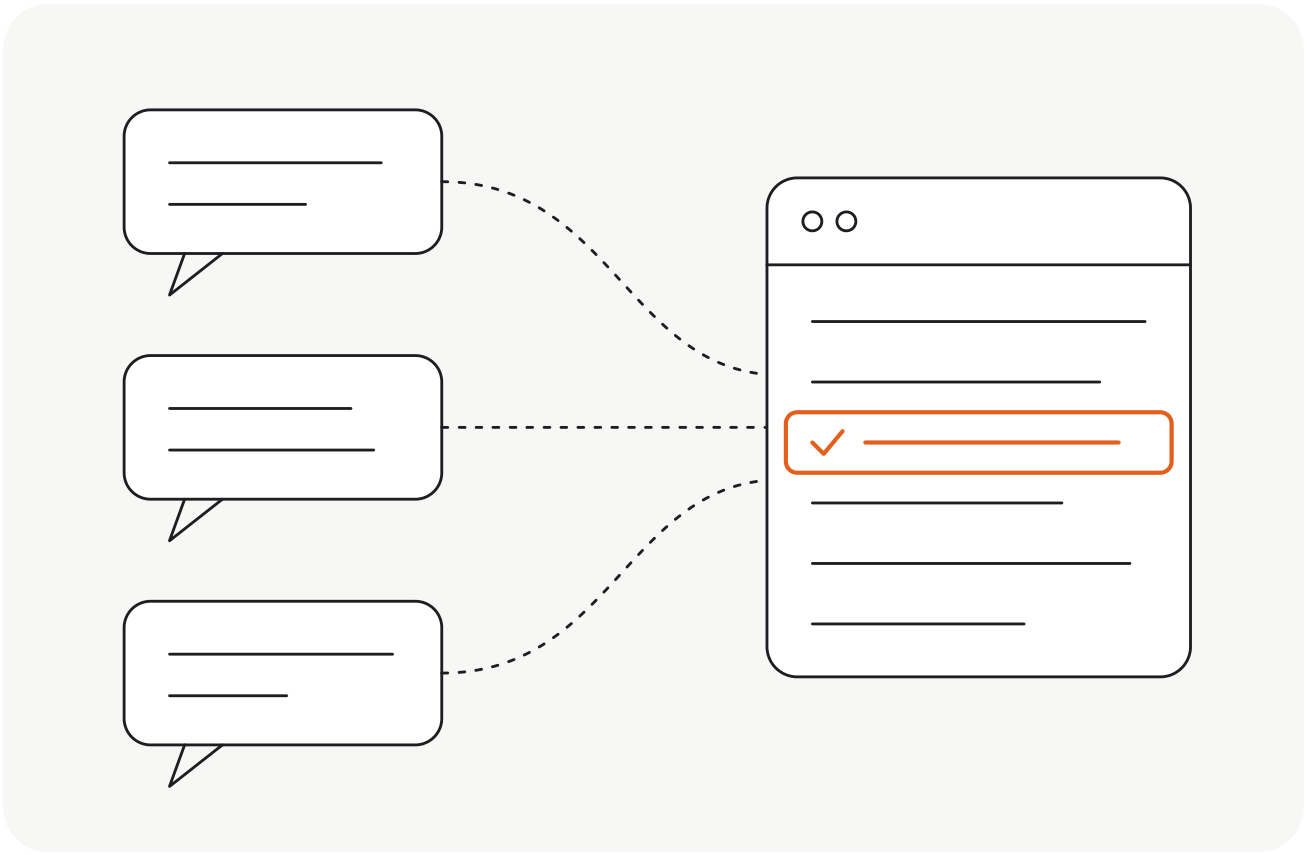
A self-scored assessment for the leaders who answer for the AI estate

The Context Readiness Assessment

How much of what your agents know would survive a resignation?

Published by Shakudo · November 2026





Contents

01	Executive summary	3
02	The resignation test	5
03	Context is already in production	11
04	Four controls your data team already runs	21
05	When several people and their agents share one context	34
06	Four stages of context readiness	40
07	Score your estate	47
08	What to do next	56
	Sources	61
	About the evidence	63
	About Shakudo	64

01

Executive summary



Pages 3 and 4

Your agents already depend on company context the way your dashboards depend on tables. We think most estates run that context with none of the controls they would demand of a table, and this assessment measures how far yours has got.

Every enterprise AI team has one person who gets more out of the agents than anyone else. They know which instruction keeps an agent honest and which table it should never touch. Most of that knowledge sits in their own chat history, and in most companies it leaves when they do.[1]

Most of the answers published in October 2026 were storage. Two large data and cloud platforms published memory for production agents, and investors funded a run of shared memory start-ups.[14, 16, 18] Storage is the easy half. The hard half is that context an agent acts on has become a production dependency, and production dependencies need controls.[10]

A data leader knows what a production dependency needs, because the data team already provides it for tables and pipelines: a named owner, an approval step before anything reads from it, a record of where it came from, and a history of changes with a way back. This assessment asks whether those four controls run on context.

It takes sixteen questions, each answered Yes, Partly or No. You score by your weakest team, never your best pilot. The total places your estate in one of four stages and points to the control to fix first. Our bet is that for most estates it will be approval: it is the only one of the four that adds a wait to every write, so it is the first one teams quietly drop.

1 Ramp, Multiplayer AI: How Teams Collaborate on AI Work, 30 Oct 2026 · **10** Forrester, Context Engineering And ContextOps, 19 Oct 2026 · **14** Databricks, Introducing Agent Memory on Databricks, 20 Oct 2026 · **16** Microsoft Community Hub, Architecting Governed Memory State for Production AI Agents with Microsoft Foundry, 29 Oct 2026 · **18** SiliconANGLE, Scuba launches with $34M, 13 Oct 2026. Full list on page 61.

Executive summary

Four places where we think the usual answers to agent memory go wrong

A bigger context window does not tell an agent what to keep

The usual engineering answer to forgetful agents is more context per call. Window size sets how much an agent can see at once. It says nothing about what the agent should keep, who may add to it, or who answers when it keeps something wrong. As one October engineering post put it, "A large context window is not a memory policy."[11]

The October benchmarks measure recall and leave approval out

October produced at least four memory benchmarks, and most of the headline numbers come from vendors scoring themselves.[23-26] The claims we collected talk about recall, accuracy and tokens. None of them asks who approved the fact the agent recalled, and that is the question your auditor will ask.

Sharing a session grants access to everything in it

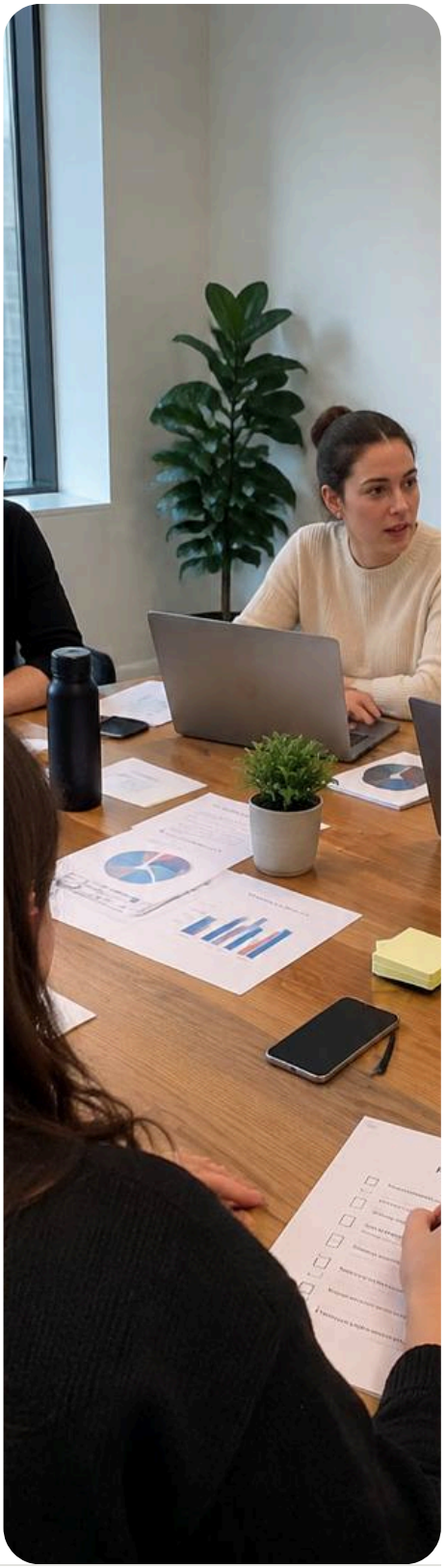
The multiplayer launches present a shared session as a feature. One tool's documentation states the cost directly: "Edit is a code-execution grant on the host."[46] Anything an agent can act on for several people needs the same care as anything a person can change for several people.

Review after the fact comes too late to stop an error

Approval has to happen before an agent reads a memory. Review afterwards "catches errors only once an agent has already used them," and by then the wrong fact has already shaped an answer.[32]

How to use this assessment

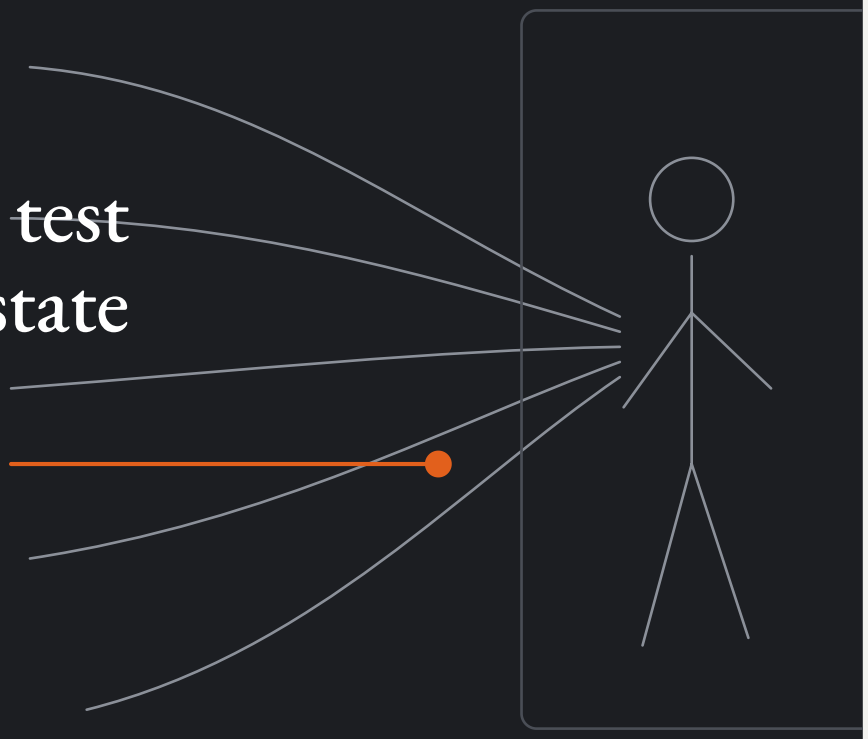
Chapters 2 to 6 make the argument. If you only want your score, turn to page 47. Fill it in with whoever runs your data platform, and answer for the weakest team whose context an agent can act on.



11 MemTensor on Hugging Face, Persistent Agent Memory Should Make the Next Task Easier, 27 Oct 2026 · **23** Eragon, Eragon MemBench, 23 Oct 2026 · **24** past.dev via PR Newswire, 6 Oct 2026 · **25** Prinevo on dev.to, 29 Oct 2026 · **26** EvoMemBench on alphaXiv, 30 Oct 2026 · **32** Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026 · **46** Databricks Answers on Omnigent, 9 Oct 2026

02

The resignation test for an AI estate



Pages 5 to 10

The person who gets the most out of your agents is a single point of failure, and no runbook lists them.

Ask who on your team is best with the agents and you will get a name quickly. Ask where their best instructions are written down and the answer is usually a chat window that no one else can see.[2]

That arrangement works while the person stays and the team stays small. It stops working when a second and third person join the same task, because nobody can see the whole exchange and nothing records who decided what.[3]

This chapter takes the problem at its most concrete, which is the day that person resigns. It looks at where an agent's knowledge actually lives in most teams, at the three patterns teams use to share it, and at the cost that never reaches a budget line.

The test is simple enough to run in your head before you read on. Pick your most-used agent. If the person who built it left on Friday, would it know the same things on Monday?

2 Agor, Announcing Agor Cloud Open Beta, 13 Oct 2026 · **3** Wavelave, How to Run Multiple AIs Together: An Engineering Guide, 10 Oct 2026

The resignation test

On Monday the agent still runs, but it knows less than it did on Friday

Picture the person on your team who gets the most out of your agents. Over months they have taught the pricing agent which instruction keeps it honest, which table it should never touch, and which customer names it confuses. None of that went through a change request. It went into a conversation.

Now picture their resignation landing on a Friday afternoon. On Monday the agent still answers. The service is up, the dashboards are green, and nothing in your monitoring will report what has gone. The lessons that made the agent good went home with the person who taught them, in a chat history nobody else can open.

Your offboarding process deletes the knowledge you wanted to keep

Your offboarding process was built to protect you from a departing employee. It revokes their accounts on the last day. For agent context that is exactly the wrong move, because revoking the account is what seals the chat history for good. The control that keeps you safe from the person is the one that deletes their knowledge.

Nobody designed it that way. It is what happens when the knowledge that runs production lives in a personal workspace instead of a shared one.

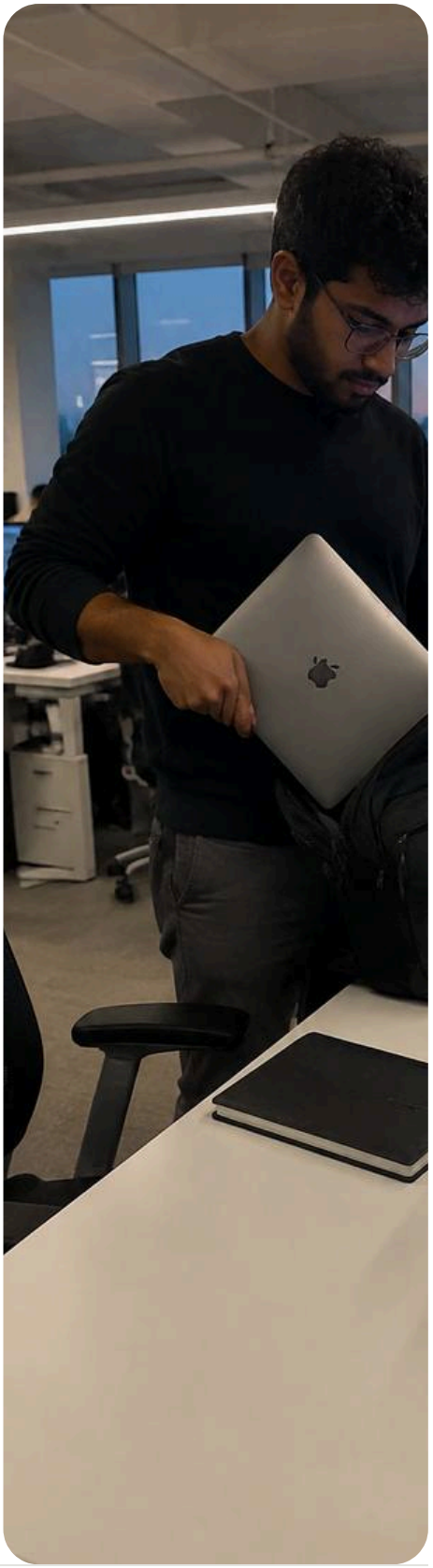
The colleague who inherits the agent pays the cost first

The cost rarely arrives as an outage. It arrives as the colleague who inherits the agent and spends weeks relearning what it used to know, or as two teams who each teach their own agent the same fact without knowing the other did. In most companies, as one October post described it, AI is “still single-player, so each person's best work stays in their own chat history and leaves when they do.”[1]

The same point holds from the company's side. One person with a great agent gets faster, and “the company stays the same, because what the agent knows sits in one account.”[5]

What we take from the resignation test

The resignation test is a fair stress test for any AI estate, because it removes the one thing most estates quietly depend on: a specific person's memory. If an agent's behaviour changes when someone leaves, its context was never part of production. It was part of that person.



1 Ramp, Multiplayer AI: How Teams Collaborate on AI Work, 30 Oct 2026 · **5** Dock, Multiplayer AI needs agents with their own computers, 1 Nov 2026

The resignation test

Single-player is the default setting, and the tools were built that way on purpose

The agents most teams use were designed for one person at one keyboard. Their memory accrues to that seat, which is why the knowledge leaves with whoever sits in it.

Read the October launch posts for what they say about the starting point, and they agree with each other. One opened with the sentence "A coding agent is built for one person."[4] Another took as its title the claim that AI agents "Were Built for One Person."[49] A third called most AI at work "still single-player."[1]

None of this is a criticism of the tools. A single-player agent is the right design for a single person, and it is how most teams got their first real results. The trouble starts when the organisation begins to rely on what those agents learned, without ever moving that learning somewhere the organisation can see.

Where an agent's knowledge ends up in a single-player estate

In a single-player estate an agent's useful context collects in four places, none of them governed: the chat history, a personal instruction file, a prompt kept in a notes app, and the head of the person who knows which of those is current.

The first three can be copied. Only the fourth tells you which copy to trust, and it walks out of the building every evening.

Teams stay single-player because it is the fastest way to work

Teams stay single-player for a sensible reason: it is fast. Nobody has to agree on anything, approve anything or wait for anyone. Every control in the rest of this assessment adds a small cost to that speed, and the only argument for paying it is the one in this chapter. The day the person leaves, the speed goes with them.

We would put it to an AI leader this way. Your agents clearly work. What you cannot yet tell is whether they would keep working the way they do now if the people who taught them were gone.

What October's launch posts said about the starting point

"The good prompts sit in a chat window no one else can see."

Agor, 13 Oct 2026[2]

"Nobody can see the whole exchange and nothing records who decided what."

Wavelave, 10 Oct 2026[3]

"Each person's best work stays in their own chat history and leaves when they do."

Ramp, 30 Oct 2026[1]

"Give one person a great agent and that person gets faster. The company stays the same."

Dock, 1 Nov 2026[5]

1 Ramp, 30 Oct 2026 · **2** Agor, 13 Oct 2026 · **3** Wavelave, 10 Oct 2026 · **4** Mirasim, Multiplayer AI coding agents for your team, 31 Oct 2026 · **5** Dock, 1 Nov 2026 · **49** Kylon, AI Agents Were Built for One Person. We Built Kylon for Teams., 4 Oct 2026

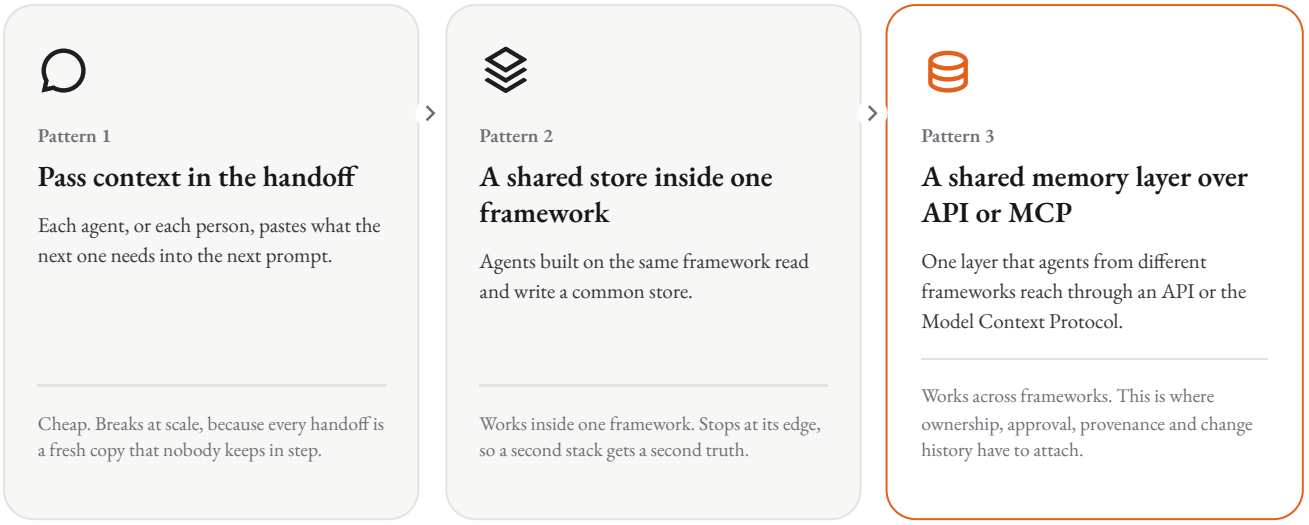
The resignation test

Most teams pass context by hand, which is the cheapest pattern and the first to break

When teams do try to share what their agents know, they tend to use one of three patterns.[6] The first copies context into each handoff. The second keeps a shared store inside one agent framework. The third puts a shared memory layer behind an API or MCP so that agents from different frameworks can reach the same facts.

Each step solves the previous one's limit, and each one moves more context into a place several agents read from. None of them, on its own, decides who may change that context. That decision is what the rest of this assessment is about, and it attaches most naturally to the third pattern.

Figure 1. Three ways agents share context today



Source: patterns as described by Membase, How to Share Memory Between Multiple AI Agents (2026 Guide), 3 Nov 2026. The labels on the lower edge of each card are Shakudo's reading.

What the patterns do not settle. All three can produce the same failure: several agents read one store of facts, two of them write facts that conflict, and nothing records which one is current.[29] A shared store makes that failure visible to more agents at once. It does not fix it.

6 Membase, How to Share Memory Between Multiple AI Agents (2026 Guide), 3 Nov 2026 · **29** Korely, Shared agent memory: when several agents read the same facts, 31 Oct 2026

The resignation test

The cost of single-player AI never appears as a line item, because it arrives as work done twice

The context rebuilding tax is the price a company pays each time somebody reassembles context that somebody else has already assembled once. The name comes from an Israeli start-up that raised $7.5M in October to work on shared memory, and that also claims its platform runs agents at eight times lower cost, a figure we have not tested.[7]

The tax is easy to miss because nobody is invoiced for it. It shows up as slower onboarding, as a second team building the same prompt library as the first, and as an agent that has to be taught again after every reorganisation. Each instance is small. Across an estate with dozens of agents and a normal rate of staff turnover, it is a standing cost with no owner.

A retry loop that was billed forty times while nobody watched

A comparison of governance platforms published in October described an expensive version of the same problem. A research agent calls a pricing agent, which calls a supplier agent, which times out. The chain retries forty times, and every retry is billed.[9] No person was watching, and no record said who had set the retry policy.

Where the rebuilding tax shows up

Onboarding

A new hire inherits an agent and relearns what its previous owner taught it.

Duplicate work

Two teams teach two agents the same fact, separately, and neither knows.

Reorganisations

Context tied to a person or a team breaks when the team is redrawn.

Incidents

A wrong answer can be traced only as far as the person who might remember why.

These four are Shakudo's grouping of the rebuilding tax. They come from our reading and from no survey.

7 SiliconANGLE, Uvi raises $7.5M to help AI agents learn business context and run at eight times lower cost, 14 Oct 2026 · 9 Lyzr, Best Platforms for Governing Multi-Agent Architectures, 19 Oct 2026

If what an agent knows sits in one person's account, your estate depends on that account, and nobody has written the dependency down.

Shakudo's reading of the resignation test

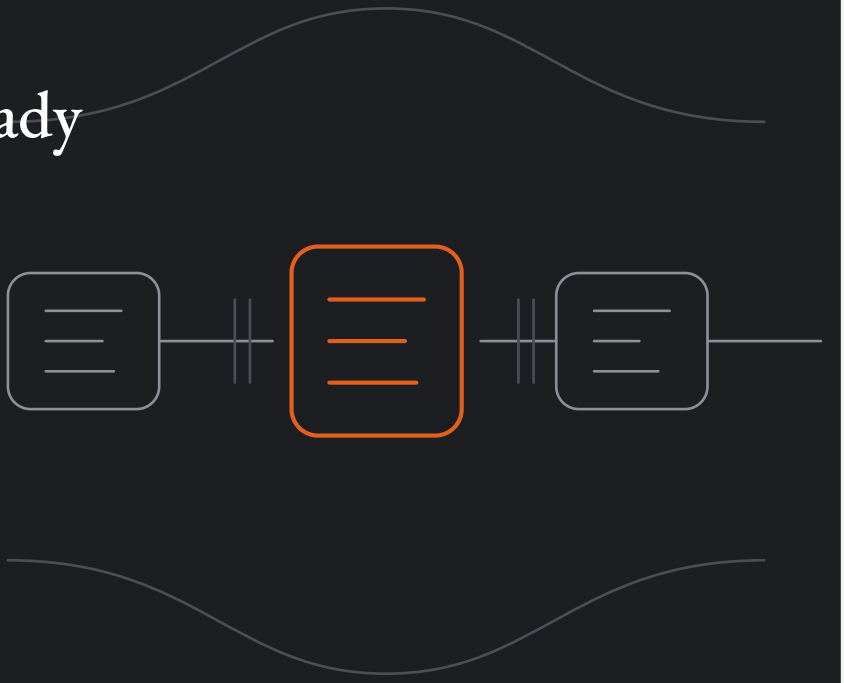
A post on 1 November made the same observation from the company's side: give one person a great agent and "the company stays the same, because what the agent knows sits in one account."[5] A data leader would call that an undocumented dependency, and would want it on the register.

The next chapter makes the case that this context has already become a production dependency in the full sense a data leader would recognise, and that most of the answers offered in October 2026 solved the storage half of the problem and left the control half to you.

5 Dock, 1 Nov 2026, trydock.ai/blog/multiplayer-ai

03

Context is already in production



Pages 11 to 20

Company context is now a production dependency for every agent that acts on it. We think most of the fixes offered in October addressed the wrong half of the problem.

As agents move from answering questions to running business processes, the context they read stops being a supporting asset and becomes a production dependency.[10] A data leader knows exactly what that commits them to, because they already run production dependencies for a living.

Most of the answers published in the same month went in other directions. Engineers asked for bigger context windows. Platforms shipped memory stores. Start-ups published benchmarks that scored their own recall.

This chapter takes those answers one at a time and says where we think each falls short. It then separates two problems that often get treated as one: an agent remembering, and several agents agreeing on what is true.

It ends with the four controls the rest of the document is built on. None of them is new. Your data team runs all four today, on tables and pipelines.

10 Forrester, Context Engineering And ContextOps: The Missing Foundation For Scaling Agentic AI, 19 Oct 2026

Context is already in production

A production dependency is anything that breaks something else when it changes, and company context now qualifies

Once agents reason, plan, use tools and execute business processes, the context they read is no longer a supporting asset. It is a production dependency, and the difference is one of kind.[10]

A supporting asset is useful and optional. If it goes stale, somebody's work gets a little worse. A production dependency is different in kind. When it changes, something downstream changes with it, and somebody has to answer for the result. A data leader already applies that test to every table that feeds a dashboard, a model or a regulatory report.

Apply the same test to context

Change the instruction that tells your pricing agent which discount table to read, and its answers change the same afternoon. Let two agents hold different versions of a customer's contract terms, and they will give that customer two different answers. Those are the symptoms of a dependency, and they are already present in any estate where agents act on company context.

The discipline that manages the dependency, context engineering, is usually described in terms of accuracy and grounding. One October definition adds a third aim, outputs that are "compliant," and that is the word most engineering teams skip.[10] Accuracy and grounding are engineering problems. Compliance is a governance problem, and it needs an owner.



Data, knowledge and context are three separate layers

"Data describes reality, knowledge explains reality, and context applies relevant knowledge to a specific situation so an AI agent can reason, decide, and act effectively."

Analyst post, 19 Oct 2026[10]

"This shift pushes enterprises from data-driven work toward knowledge-driven execution."

Analyst post, 19 Oct 2026[10]

Our reading: your data team already governs the first layer. The third layer is the one your agents act on, and in most estates it has no steward.

10 Forrester, Context Engineering And ContextOps: The Missing Foundation For Scaling Agentic AI, 19 Oct 2026

Context is already in production

The first wrong turn: treating a bigger context window as a memory policy

More room on every call is an engineering convenience. It does not tell an agent what to keep, and past a point it makes the agent worse.

The instinct is understandable. If an agent forgets, give it more to read. The trouble is that a large context window is not a memory policy.[11]

A window sets how much an agent can see on one call. A memory policy decides what the agent should keep across calls, who may add to it, and who answers when it keeps something wrong. Those are different questions, and only the second set has an accountable answer.

Longer context makes a long-running agent less accurate

There is also a quality cost. The term context rot, coined by one vendor and now used by at least three other publishers, describes the drop in accuracy as a long-running agent's context grows.[12] Two separate research papers on context rot appeared inside the month. Filling the window is one of the ways teams cause it.

The usual list of context operations has no step for approval

One cloud provider's guide reduces context engineering to four operations: write, select, compress and isolate.[13] The list describes how context moves.

Read it as a data leader and one verb is missing. There is a verb for putting context in and three for handling it once it is there. There is no verb for approving it. In a data pipeline, nobody would accept a design where anything that can write to a production table also decides, alone, that the write is correct.

That gap is the subject of this assessment. Engineering decides how context moves. Somebody still has to decide what is allowed to become context in the first place.

What the October writing said about context rot

The effect

Accuracy falls as a long-running agent's context grows.

Where it was named

Coined by one vendor and since used by at least three other publishers.[12]

The research

Two separate context rot papers in the month, among them “Diagnosing and Mitigating Context Rot in Long-horizon Search.”[50]

The practitioner view

One October title states it: “Why Longer Context Windows Degrade Agent Reliability.”[51]

11 MemTensor on Hugging Face, 27 Oct 2026 · **12** c3.ai, Context Rot: Why Long-Running AI Agents Lose Reliability, 27 Oct 2026. The term was coined by Chroma and is also used by LangChain, c3.ai and DeepLearning.AI · **13** AWS Builder Center, What Is Context Engineering for AI Agents?, 13 Oct 2026 · **50** arXiv, Diagnosing and Mitigating Context Rot in Long-horizon Search, 25 Oct 2026 · **51** Traversaal, Context Rot: Why Longer Context Windows Degrade Agent Reliability (And How to Fix It), 12 Oct 2026

Context is already in production

Five terms became common vocabulary in one month, and only two of them are about control

A category has formed when analysts and vendors start to recognise each other's words. By the end of October five terms had reached that point. Most of them describe the engineering problem: how much an agent can hold, and how to choose what it sees.

The two that describe control, the write gate and provenance, came mostly from governance writers and researchers, not from the platforms. If you are briefing a board, those are the two worth teaching them.

Table 1. The vocabulary that formed in October 2026

Term	Who uses it	What it means	What it asks of you
Context rot	Coined by one vendor; now used by at least three other publishers[12]	Accuracy drops as a long-running agent's context grows	Stop treating a bigger window as the fix
Context engineering	An analyst firm, a cloud provider and AI tooling companies[10, 13]	Deciding which tokens the model sees on every call	Staff it as a discipline, with an owner
Write gate	One October analysis[17]	Memory needs a controlled write path more than it needs a bigger store	Decide who may write before you decide where
Provenance	A research lab, a governance company and academic authors[32, 33, 53]	Every memory names its source, its owner and its approver	Make every fact traceable to a line and a decision
Context rebuilding tax	One shared memory start-up[7]	The repeated cost of reassembling company context someone already assembled	Count it, because nobody invoices you for it

Sources: the research summary for this assessment, drawn from publications dated 4 October to 3 November 2026. The right-hand column is Shakudo's reading.

7 SiliconANGLE, 14 Oct 2026 · **10** Forrester, 19 Oct 2026 · **12** c3.ai, 27 Oct 2026 · **13** AWS Builder Center, 13 Oct 2026; Parallel and LangChain also use the term · **17** themicrosoftcloudblog.com, Agent memory needs a write gate, not a bigger store, 28 Oct 2026 · **32** Trail, 19 Oct 2026 · **33** IBM Research, 25 Oct 2026 · **53** arXiv, Stashbird, 17 Oct 2026

Context is already in production

The second wrong turn: memory was sold as storage, and storage is the easy half

The platform launches answered where memory lives and how agents reach it. The harder questions, who may write to it and how a wrong fact gets traced, were left mostly to the buyer.

Between 20 and 29 October, two of the large data and cloud platforms published memory for production agents: one as a beta product with a companion research post, the other as a reference architecture for what it called governed memory state.[14, 15, 16] The beta launch described the problem as agents that, without memory, "rediscover facts, forget preferences, and repeat mistakes."[14] Its companion post argued that memory can improve quality and efficiency together.[15]

Both describe where memory lives and how agents reach it. The day before the reference architecture appeared, a separate analysis put the missing piece in its headline: agent memory needs a write gate more than it needs a bigger store.[17]

Why the platforms shipped storage before controls

Storage is what platforms know how to sell. It has a price per gigabyte, a latency figure and a diagram. Controls are harder to sell because they are mostly decisions your organisation has to make, and a platform can only give you somewhere to record them.

The decisions a memory store cannot make for you

A memory store holds whatever reaches it. It does not know which of your teams is allowed to teach an agent your refund policy, whether the person who wrote a fact was the right person, or whether last week's correction replaced the old fact or sits next to it. Those answers live in your organisation, and the store can only enforce them once someone has written them down.

There is a quieter issue too. A store your platform vendor runs comes with the vendor's defaults: where the memory sits, who at the vendor can reach it, how long it is kept. A data leader would never accept a vendor default for the residency of a customer table without reading it. The same reading belongs on memory.

Three questions we would put to any memory platform

- Can the store refuse a write that has no named owner?
- Is approval recorded before an agent can read a new memory, or only logged afterwards?
- Where does the memory physically sit, and who changed that default?

14 Databricks, Introducing Agent Memory on Databricks, 20 Oct 2026 (Agent Memory on Agent Bricks, in beta, on Lakebase; the product documentation still marked it Beta on 4 Nov 2026) · **15** Databricks, MemCurrent: Designing Persistent Memory for Production Agents, 20 Oct 2026 · **16** Microsoft Community Hub, Architecting Governed Memory State for Production AI Agents with Microsoft Foundry, 29 Oct 2026 · **17** themicrosoftcloudblog.com, 28 Oct 2026

Context is already in production

Memory for agents became a category in thirty days, and the list makes more sense sorted by who was writing

The window behind this assessment runs from 4 October to 3 November 2026. In those weeks, memory for agents moved out of research posts and into launches, funding rounds and an individual Internet-Draft.

Sort the rows by the right-hand column. Storage, sharing and funding came from platforms and their investors. Control came from a governance company, a research lab and practitioners. Few of them were in the same conversation.

Table 2. What happened in the thirty days to 3 November 2026

Date	Who	What happened	About
6 Oct	Zeroset	Raises $5.2M pre-seed for agent memory and workflow state[20]	Funding
7 Oct	Atlassian	Introduces AMP, the Agentic Multiplayer Protocol[40]	Protocol
13 Oct	Scuba	Launches with $34M for a context engine, a shared memory hub for workers and agents[18]	Funding
14 Oct	Uvi	Raises $7.5M and names the context rebuilding tax[7]	Funding
19 Oct	Trail	Publishes a governance policy for agent memory with ten controls[32]	Control
20 Oct	Databricks	Introduces Agent Memory, in beta, on Lakebase[14]	Storage
20 Oct	Asymptote Labs	Raises $5.5M seed for Beacon, open-source memory from coding-agent sessions[19]	Funding
25 Oct	IBM Research	Presents transparent, traceable, deterministic agent memory for ISWC 2026[33]	Control
27 Oct	Mosaic	Reported at about 2,000 installs, with 37 organisations syncing coding-agent sessions into one shared store[22]	Storage
28 Oct	IETF	Individual draft on context management for agents posted; still at revision -00 and not adopted by a working group on 5 Nov[21]	Draft
29 Oct	Microsoft	Publishes an architecture for governed memory state in Foundry[16]	Storage
2 Nov	Practitioners	A dev.to post asks who audits what an agent remembers; QueryNow publishes what an audit trail should record[34, 35]	Control

Funding figures are as the press reported them; they were not checked against filings. The About column is Shakudo's classification.

7 SiliconANGLE, 14 Oct · **14** Databricks, 20 Oct · **16** Microsoft Community Hub, 29 Oct · **18** SiliconANGLE, 13 Oct · **19** The Next Web, 20 Oct · **20** Nexi, 6 Oct · **21** IETF, 28 Oct · **22** Founderland, 27 Oct · **32** Trail, 19 Oct · **33** IBM Research, 25 Oct · **34** QueryNow, 2 Nov · **35** dev.to, 2 Nov · **40** Atlassian via Business Wire, 7 Oct. All 2026.

Context is already in production

For a buyer of agent memory, where the memory lives decides more than the benchmark score

Treat every vendor benchmark as a claim until someone else reproduces it

October's benchmark claims came mostly from vendors measuring themselves. Read every one of them as a claim, including the ones in the panel on this page. Two of them test harder cases: several people sharing context, tool-heavy conversations, decisions that change, and information that must never be retained.[23, 26]

Even the good benchmarks measure what an agent recalls. A data leader's exposure sits one step later: whether anyone approved the thing the agent recalled, and whether you can prove it.

Where an agent's session record lives is a choice someone has to make

One open-source tool funded in October shows what is at stake. It is MIT-licensed and captures whole agent sessions, including prompts, responses, tool calls, commands, edits, approvals and token counts, as JSONL exposed over MCP or Agent Skills. Its documentation offers two homes for that record: a local install with full control of the data, or the vendor's hosted cloud.[19]

The record includes the approvals themselves, so the choice between those two homes is a governance question as well as a privacy one. We think it belongs to whoever will answer for the record, and in practice it is often made by whoever runs the installer.

Builders are already pulling memory back in

The pushback in the corpus came from engineers rather than procurement. One local memory tool runs a knowledge graph on the user's own machine and sends nothing to a hosted service.[27] One developer on dev.to built a self-hosted shared memory mesh for coding agents after concluding that an 8,000-token CLAUDE.md file does not scale.[28]

We read those as early signals from people who hit the problem first. The enterprise version of the same instinct is a requirement: memory that the agents act on should sit inside the perimeter you already govern, under controls you already run.

Benchmark claims in the window, as their authors stated them

Eragon MemBench

Eragon reports 75.3% correct for itself against 37.8% for Mem0, with about 22 times fewer tokens, on Eragon's own benchmark.[23]

BEAM

past.dev claims first place on what it calls the largest public memory benchmark.[24]

LoCoMo and LongMemEval

Prinevo reports 83.89% and 82.60%.[25]

EvoMemBench

An academic benchmark for self-evolving memory.[26]

Vendor-run results. Shakudo has not reproduced any of them and does not rely on them. A 12 October study re-graded 47,253 published LoCoMo and LongMemEval answers from ten memory products and found that which system wins depends on the model doing the judging.[59]

19 The Next Web, Asymptote Labs raises $5.5M to track and control what AI agents do, 20 Oct 2026; the tool is Beacon, and its local and Beacon Cloud options are from the Beacon documentation, read 5 Nov 2026 · **23** Eragon, 23 Oct 2026 · **24** past.dev via PR Newswire, 6 Oct 2026 · **25** Prinevo on dev.to, 29 Oct 2026 · **26** EvoMemBench on alphaXiv, 30 Oct 2026 · **27** Semurg, 13 Oct 2026 · **28** dev.to, Solving AI Agent Amnesia, 30 Oct 2026 · **59** Zenodo, Who wins depends on who judges, 12 Oct 2026

Context is already in production

One agent that remembers is a feature, while several agents agreeing on what is true is an operating problem

Most writing about memory is about one agent remembering. Your estate has many agents reading the same facts, and the hard question is which fact is current.

Shared agent memory means several agents reading one store of facts, which is a different thing from one chat log.[29] The failure it describes is the one any data leader will recognise from master data. Two agents write facts that conflict, and nothing in the store says which one is current.

In a single-agent world that is a quality issue. In a shared store it spreads. Every agent that reads the conflicting pair inherits the disagreement, and the more agents you add, the more places the disagreement can surface in front of a customer.

Two memory layers that teams often treat as one

One October guide to stateful systems argued that flat vector memory breaks down for stateful multi-agent systems, and that those systems need two layers teams often treat as one.[30] The first is orchestration state: checkpoints, so that a run can resume after a crash. The second is shared semantic memory: facts, decisions and relationships that persist across sessions.

The distinction matters for governance because the two layers have different lifetimes. Orchestration state lives for a run and can be thrown away. Shared semantic memory lives as long as the business does, and that is the layer that needs an owner.

Why the October research turned to graph memory

Graph database vendors, memory libraries and a corporate research lab all made the case for graph memory in the window. One pairing of a memory library and a graph framework adds bi-temporal knowledge graphs, so that a fact carries both when it was true and when it was recorded.[31] The research lab described graph memory that is "transparent, traceable, deterministic."[33]

We think the architecture debate matters less to a data leader than what the graph people were reaching for. Every one of those features is a governance feature: knowing when a fact was true, knowing when it was written, and being able to trace an answer back to it.

Multi-agent failures start in the exchange between agents

The academic papers collected for this assessment kept returning to the same finding: multi-agent failures come from poor exchange of information between agents more often than from model quality.[52, 53] A better model will not fix two agents that disagree about a fact. A rule about which fact is current will.

29 Korely, 31 Oct 2026 · **30** Cognee, How to Build Stateful Multi-Agent Systems in Production (2026), 29 Oct 2026 · **31** ABATA AI, Mem0 v2.0 & Graphiti: Persistent Temporal Knowledge Graphs for Multi-Agent Systems, 18 Oct 2026; TigerGraph and FalkorDB also argued for graph memory in the window · **33** IBM Research, 25 Oct 2026 · **52** arXiv, Muffin: A Shared Memory Buffer for Asynchronous AI Agents, 22 Oct 2026 · **53** arXiv, Stashbird: Episode-Grounded Lifecycle Reconciliation for Conversational Agent Memory, 17 Oct 2026

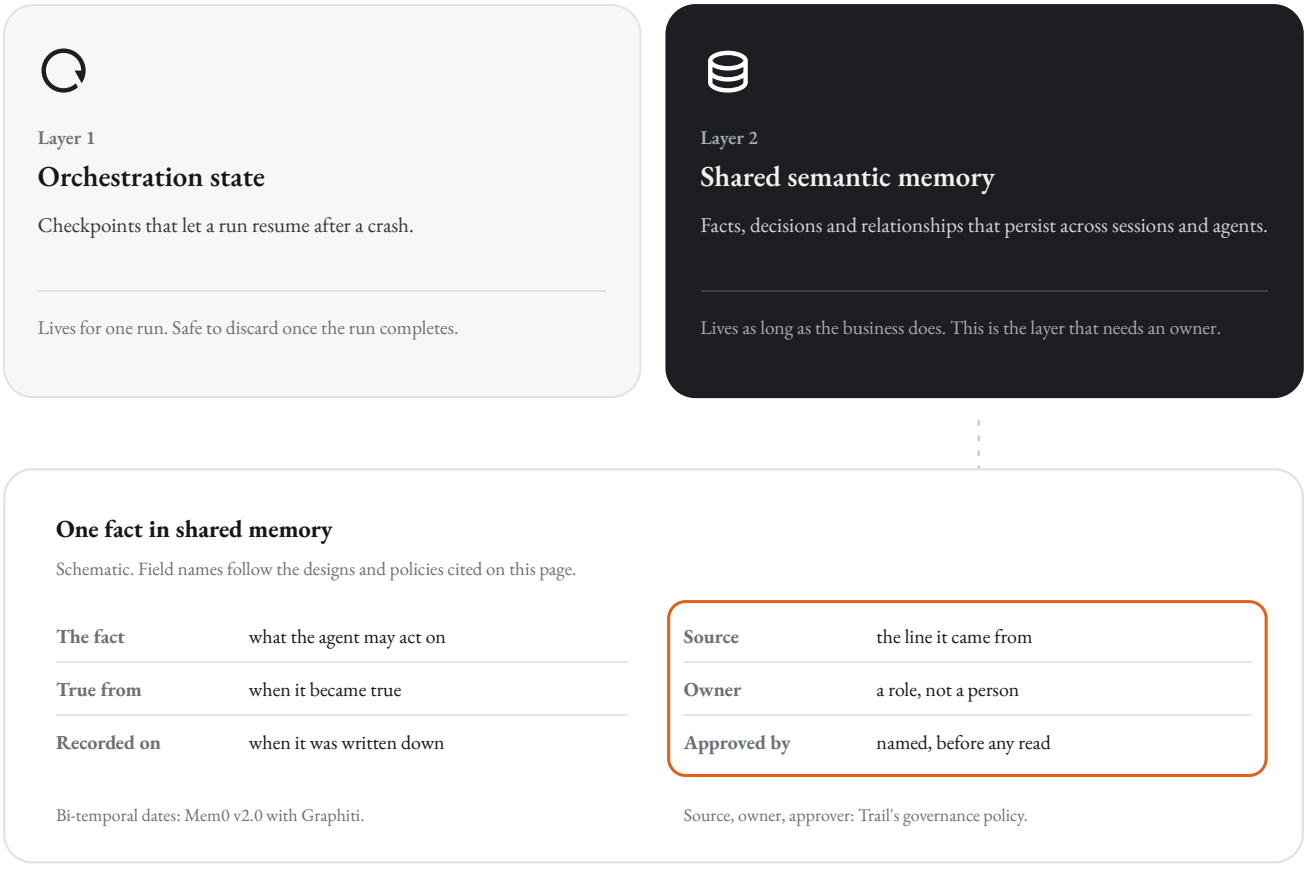
Context is already in production

Only one of the two memory layers needs governing, and it is the one that outlives every run

Orchestration state is plumbing. It can be rebuilt, and losing it costs you a rerun. Shared semantic memory is company knowledge, and losing control of it costs you a wrong answer that nobody can explain.

The lower half of the figure shows what a governed fact in that second layer has to carry. The two dates come from the bi-temporal designs published in October. The last three fields are the controls in the next chapter.

Figure 2. Two layers that teams confuse, and what a governed fact carries



Sources: Cognee, 29 Oct 2026 (the two layers); ABATA AI, 18 Oct 2026 (bi-temporal facts); Trail, 19 Oct 2026 (source, owner and approver).[30, 31, 32]

30 Cognee, 29 Oct 2026 · **31** ABATA AI, 18 Oct 2026 · **32** Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

Context is a production dependency, and it needs the same four things your data team already supplies for every table.

An owner

A named role answers for every item an agent can act on, and the ownership outlasts the person.

An approval step

Someone signs off a new or changed item before any agent reads it, and the sign-off is on record.

A provenance record

Every item names the source it came from, so a wrong answer traces back to a line and a decision.

A change history

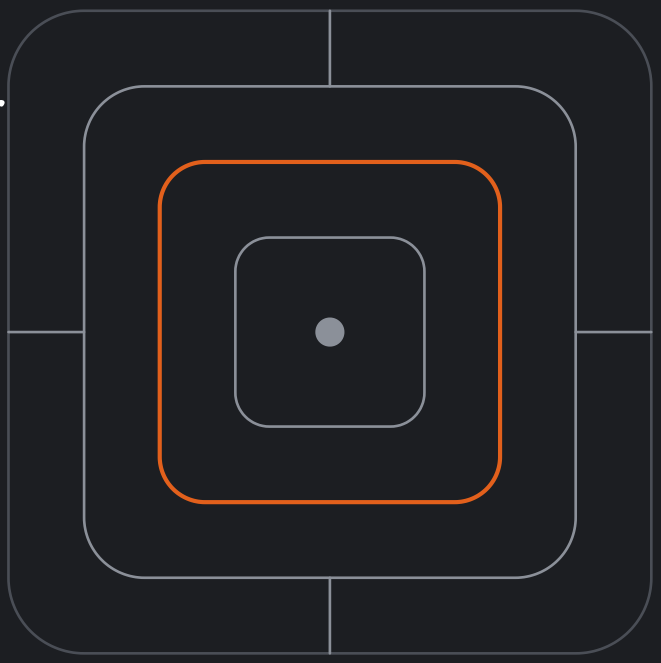
Every change carries a diff, a named reviewer and a way back, kept whether or not anyone remembered.

None of these is a new idea, and that is the point. Auditors already recognise every one of them. What is new is the asset they apply to: a store of company context that your audit committee may not yet know exists.

The next chapter takes the controls one at a time, says what each costs to skip, and names the one we think teams skip first.

04

Four controls your data team already runs



Pages 21 to 33

Your auditors already know these controls by name. What they have not yet seen is the store of company context the controls now have to cover.

Ownership, approval, lineage and change control are the ordinary furniture of data governance. Every regulated enterprise has a steward for its customer table and a change process for its pipelines. A governance policy for agent memory published on 19 October reads like a lineage standard rewritten for agents.[32]

That is the good news in this chapter. You do not need a new discipline. You need the one you have to reach a new asset.

The chapter takes the controls in the order a memory meets them: it is proposed, approved, read and later changed. For each one it says what the control looks like on context, what skipping it costs, and how to start.

It also makes a claim you are free to dispute. Of the four, we think approval is the one teams drop first, because it is the only control that puts a person in the path of every write.

32 Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

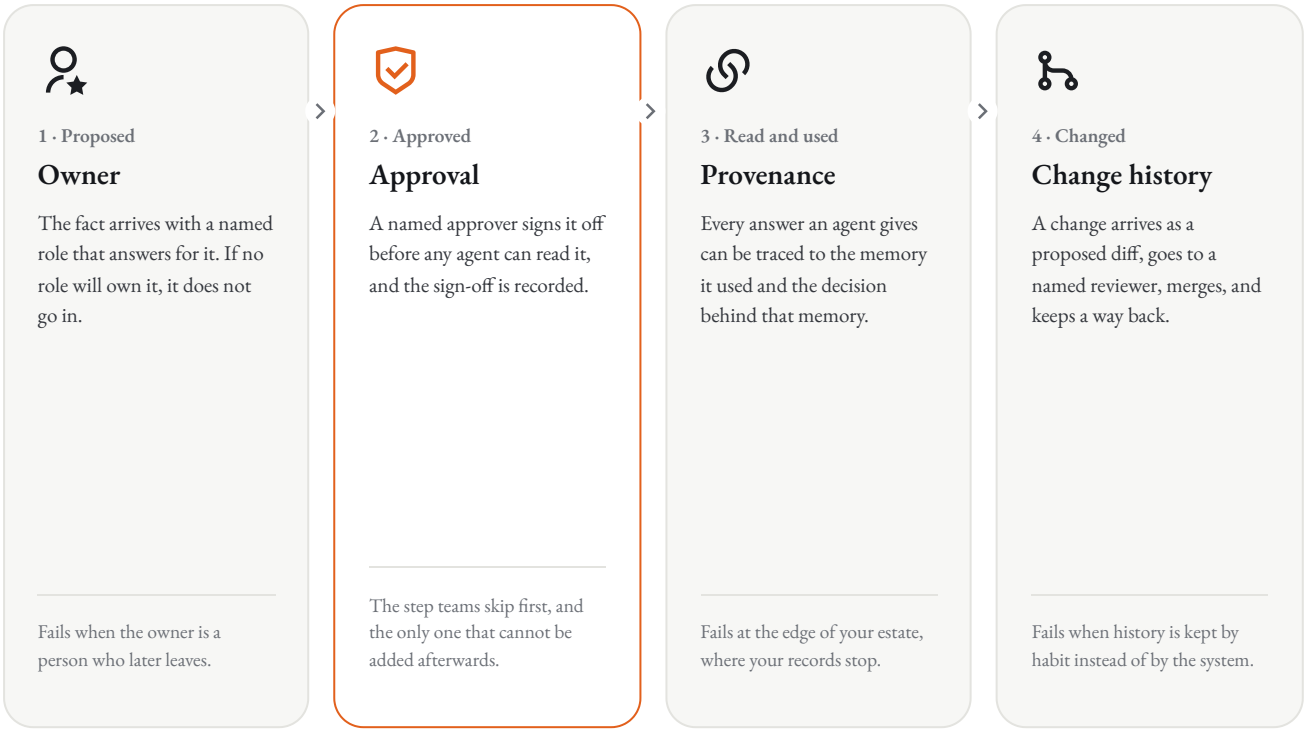
Four controls

Each control acts at one moment in a memory's life, and approval is the only one that acts before an agent reads

Follow one fact from the moment somebody proposes it to the day somebody changes it. Ownership is settled when it is proposed. Approval happens between writing and reading. Provenance earns its keep when an agent uses the fact and gets something wrong. Change history matters when the fact has to be corrected or withdrawn.

Three of the four can be bolted on after the event: you can name an owner later, reconstruct a source later, and rebuild a history from logs. Approval cannot be added afterwards. Once an agent has read an unapproved fact, the error has already happened.

Figure 3. The life of one memory, and where each control acts



Sources: the controls follow Trail's governance policy for agent memory, 19 Oct 2026.[32] The order of the stages and the failure notes are Shakudo's reading.

32 Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

Four controls

The controls translate from data to context with very little rewording

Put the four controls side by side as your data team runs them today and as they would run on agent context. The middle column will be familiar to anyone who has sat through a data governance review. The right-hand column is the same idea, aimed at a new store.

The last column is the question an auditor will ask about any fact an agent acted on. If your team cannot answer it for context the way it can for a table, that is the gap this assessment scores.

Table 3. The four controls, on data today and on context

Control	On your data today	On agent context	The auditor asks
Owner	Every dataset has a named owner and a steward	Every memory an agent can act on names an owner, and the ownership sits with a role so it outlasts the person	Who answers for this fact?
Approval	Access and changes go through a named approver	A memory reaches an agent only after someone has approved it, and the approval is on record	Who let it in, and when?
Provenance	Lineage traces every record to its source	Every memory names its source, so a wrong answer can be traced to a line and a decision	Where did it come from?
Change history	Pipeline changes carry a diff, a reviewer and a rollback	A memory change carries a proposed diff, a named reviewer, a merge and a revert path	What changed since, and can you undo it?

The right-hand controls follow Trail's governance policy for agent memory, 19 Oct 2026.[32] The auditor's questions are Shakudo's wording.

A practical shortcut. Hand this table to the person who runs data governance today and ask which rows they could already evidence for your most-used agent. Their answer is a fair first draft of your score.

32 Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

Control 1 · Owner

Ownership has to sit with a role, because an owner who is a named person is simply the next resignation

Ownership starts where any lineage standard starts: every memory an agent can act on "should name its source, its owner and the person who approved it, so a wrong answer can be traced to a line and a decision."[32]

Most teams that try to assign owners write down a person's name. That passes a review and fails the resignation test from chapter 2. When the named person leaves, the memory has an owner on paper and nobody in practice.

The fix is the one data governance arrived at years ago. The owner is a role: the pricing team's lead analyst, the support operations manager, the head of credit policy. The person in the role can change. The obligation stays.

The three decisions an owner makes about their context

An owner answers three questions about the context they hold. Who may write to it. When a fact in it should be retired. Which version wins when two agents disagree, the case from chapter 3, where nothing in the store says which fact is current.[29]

That third duty is the one most often missing. Without it, conflicts sit in the store until a customer finds them.

Ownership starts with a list of where your context lives

Ownership needs an inventory first. Before anyone can own the context an agent reads, someone has to list where that context lives: the instruction files, the retrieval indexes, the memory stores, the shared drives the agent can search. We suspect few estates could produce that list today without a week of interviews. The first question in the assessment asks whether yours could.



Five questions an owner should be able to answer without looking anything up

- Which agents read this context?
- Who is allowed to write to it?
- Which facts in it are past their date?
- When two facts conflict, which one wins?
- Who takes over if I move roles?

Shakudo's checklist, derived from the controls in the governance policy cited on this page.

29 Korely, Shared agent memory: when several agents read the same facts, 31 Oct 2026 · **32** Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

Control 1 · Owner

Skip ownership and every stale fact becomes permanent, because nobody has the right to delete it

The first cost of missing ownership is not dramatic. It is accumulation. Facts go into shared context and never come out, because removing a fact needs someone with the authority to say it is wrong, and without an owner nobody has that authority.

The store grows. Old discount rules sit beside new ones. A product that was renamed in the spring is still described by its old name in a memory written in the winter. Each agent that reads the store has to guess which version applies, and the guess is invisible until it is wrong in public.

Facts that several teams depend on still need a single owner

The hardest cases are facts that several teams depend on. A customer's contract terms might be read by a sales agent, a support agent and a billing agent. If each team owns its own copy, you have three facts and no truth.

Our rule of thumb is that the owner is the team that would be blamed if the fact were wrong. For contract terms that is usually the team that signs contracts, not the teams whose agents read them.

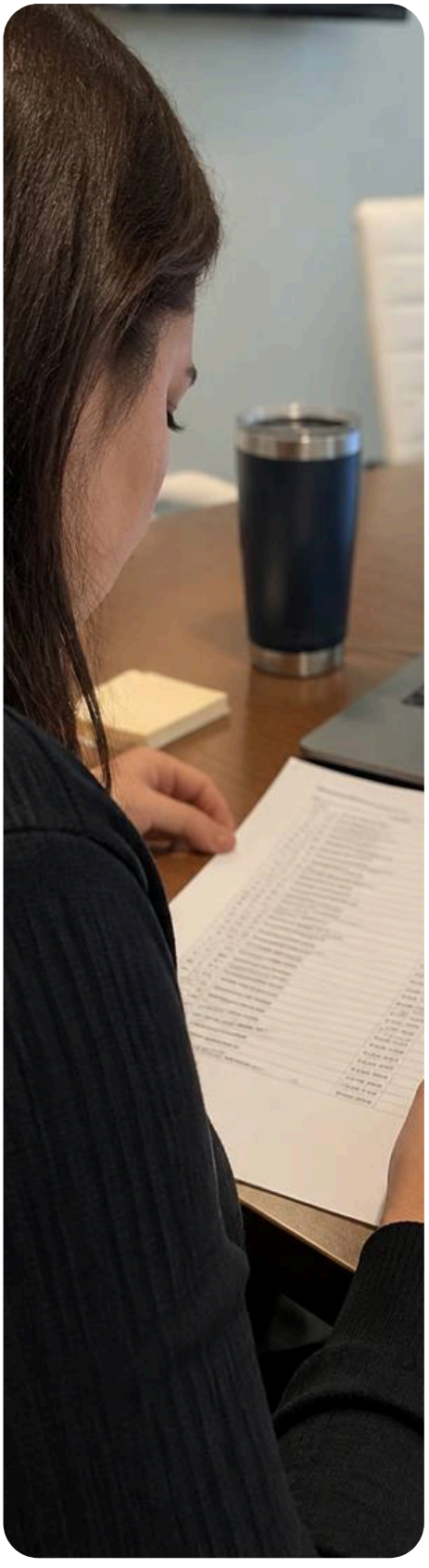
A first ownership exercise you can run on Monday

Pick your most-used agent. List every source of context it reads, including the ones that feel informal, such as a shared prompt document or a channel it searches. Next to each one, write the role that should answer for it. Where you cannot name a role, you have found your first gap.

Then do the resignation test on the list. For each source, ask whether it would survive the departure of the person who built it. Anything that lives in a personal account fails, however carefully it was written.

Why ownership is the cheapest control and the easiest to fake

Ownership is the cheapest of the four controls to put in place. It costs a list and a meeting. That is also why it is the easiest to fake: a list of names in a wiki satisfies a checkbox and changes nothing about who actually deletes a stale fact.



Control 2 · Approval

Approval is the control teams skip first, and its absence is the most expensive of the four

This is our view, and it is the one claim in this document we most expect readers to argue with. The reasoning is below so that you can test it against your own estate.

Why approval is the first control that teams drop

Look at what each control costs to run day to day. Ownership is declared once and revisited occasionally. Provenance arrives partly for free from logs. Change history comes almost for free when context lives in files under version control. Approval is different. It puts a person between every write and every read, and it costs a wait every single time.

Teams under delivery pressure drop the control that slows them most. In our reading that is approval. They keep logging, they keep a history, and they tell themselves they will review the store later.

Why a review after the read comes too late

The reason fits in one sentence: approval "has to happen before agents read a memory; review after the fact catches errors only once an agent has already used them."[32]

A review after the fact finds a bad memory after it has shaped answers. By then the question is no longer whether the memory was wrong but how many customers it reached.

It also leaves the write path open. A store anyone can write to without approval is a store anyone can poison, and memory poisoning drew steady coverage in the same month.[36, 37] One October analysis called the missing piece a write gate.[17]

What approval asks of a team in practice

Approval needs a named approver for each kind of context, and no committee. It means that for each kind of context someone is named as the approver, and that the store will not serve a new fact to an agent until that person has said yes.

Three rules for governing agent memory, quoted in full

"Every memory an agent can act on should name its source, its owner and the person who approved it, so a wrong answer can be traced to a line and a decision."

"Approval has to happen before agents read a memory; review after the fact catches errors only once an agent has already used them."

"Rules should change through the same steps as code: a proposed diff, a named reviewer, a merge, and a revert path."

Trail, 19 Oct 2026[32]

17 themicrosoftcloudblog.com, Agent memory needs a write gate, not a bigger store, 28 Oct 2026 · **32** Trail, 19 Oct 2026 · **36** CodeCrux, Securing Persistent Agent Memory, 2 Nov 2026 · **37** SC World, After memory poisoning, an AI agent needs more than a restart, 23 Oct 2026

Control 2 · Approval

Design approval so that nobody needs to route around it, or they will route straight back to a private chat

Approval before the read is the right control, and it creates a queue. If one small group approves every memory for every agent, throughput becomes something you have to manage. Teams route around a slow queue the way they route around any slow process. The usual workaround is a personal chat history, which brings the single-player estate from chapter 2 back in through the side door.

So the design question is how to keep the control and lose the queue.

Match the approver to the damage a wrong fact could do

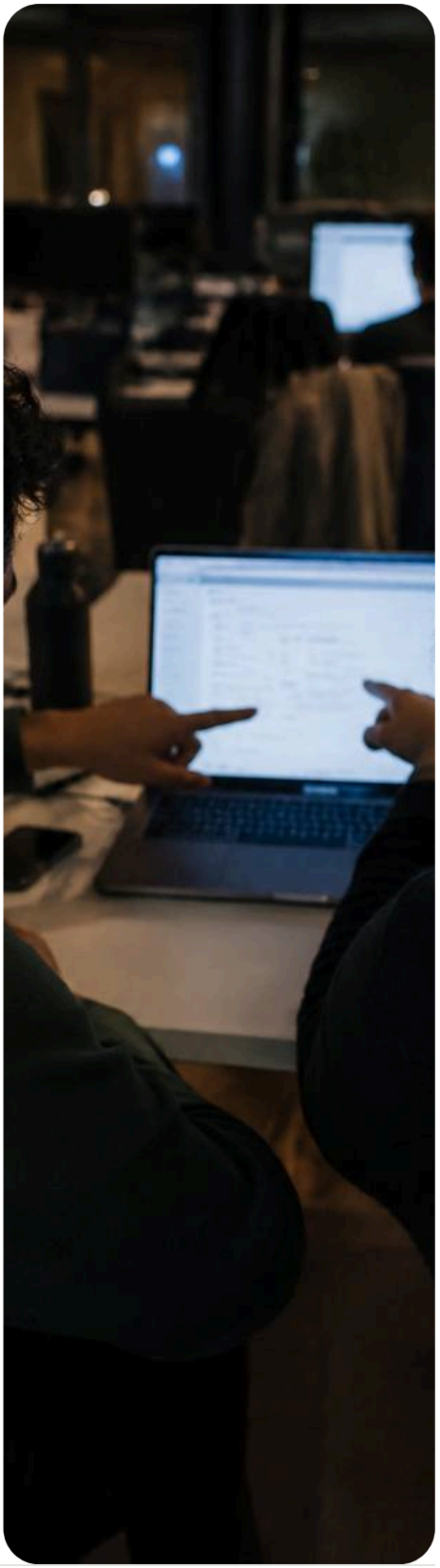
Not every fact needs the same approver. A preference that only changes how one person's assistant formats a summary can be approved by that person. A fact that changes what an agent tells a customer about price, eligibility or contract terms needs the owner of that fact to say yes. Sort context by the damage a wrong fact could do and set the approver accordingly.

Decide what is shared and what stays private before rollout

One October rollout guide reads like a governance instruction: "Sharing should be a choice. Decide what's shared and what stays private before you roll anything out."[1] A team that has made that decision in advance has already done half the approval work, because private context never enters the queue at all.

Some shared coding tools already build approval into the session

Approval is already ordinary in shared coding tools. In one, every tool runs on the host's machine and "the host approves what guests ask for."[47] Another lets people live-share several coding agents and approve their changes from any browser.[58] Those approvals cover actions rather than memory, but they show the pattern working at the speed developers will tolerate.



1 Ramp, 30 Oct 2026 · **47** claude-mods, shared-session: multiplayer Claude Code sessions by link, 16 Oct 2026 · **58** Bellows, Live-share multiple coding agents side-by-side and approve changes from any browser, 2 Nov 2026

Control 3 · Provenance

If you cannot trace a wrong answer to a line, you can apologise for it but you cannot fix it

Provenance is the control that turns an incident into a correction. It is also the one regulators and practitioners started asking about out loud in October.

When an agent gives a wrong answer, three people want to know why: the customer who received it, the manager who has to respond, and the auditor who will ask later. Without provenance, the investigation ends at the agent. Somebody reads the logs, finds the prompt, and has to guess which memory shaped the answer and who put it there.

The standard to aim for is specific: a wrong answer should trace "to a line and a decision."[32] The line is the source the fact came from. The decision is the approval that let it in.

Research on agent memory is working toward the same target

One ISWC 2026 paper on graph memory carries the title "Transparent, Traceable, Deterministic."[33] Another paper worked on reconciling memories against the episodes that produced them.[53] Both are attempts to make an agent's memory explain itself.

Practitioners are asking the auditor's question

On 2 November a post on dev.to asked it directly: "Your AI Agent Remembers Things. Who Audits What It Remembers?"[35] Guidance appeared the same day on what an agent audit trail should record and how long to keep it,[34] and an earlier guide set out what an agent audit trail must capture for the EU AI Act and DORA.[54]

We read that cluster as the moment provenance stopped being an engineering nicety. When guides written for regulated firms start naming agent audit trails, you will need provenance on context; what remains open is whether you will have it when someone asks.

What a provenance record should carry

Source

The document, table or conversation the fact came from.

Owner and approver

Named roles, as the governance policy on page 26 requires.[32]

Two dates

When the fact became true, and when it was recorded, as in the bi-temporal designs.[31]

Readers

Which agents used it. This last field is our addition; it is what lets you find everyone affected when a fact turns out to be wrong.

31 ABATA AI, 18 Oct 2026 · **32** Trail, 19 Oct 2026 · **33** IBM Research, 25 Oct 2026 · **34** QueryNow, AI Agent Audit Trail, 2 Nov 2026 · **35** dev.to, Your AI Agent Remembers Things. Who Audits What It Remembers?, 2 Nov 2026 · **53** arXiv, Stashbird, from Microsoft, 17 Oct 2026 · **54** Backslash Security, What an AI Agent Audit Trail Must Capture for the EU AI Act and DORA, 7 Oct 2026

Control 3 · Provenance

Provenance stops at the edge of your estate, so context assembled somewhere else arrives with no history

Most provenance designs work well inside one system. A fact written into a store and read from the same store can carry its full record. The trouble begins when context crosses a boundary: from one framework's store into another, from one team's platform into a shared one, or from a vendor's hosted service back into your estate.

At each crossing the record can thin out. The fact arrives. Its source, owner and approver may not. By the time an agent acts on it, the trail ends at the border it crossed.

Where the record lives matters as much as what it says

Recall what the open-source session recorder from chapter 3 captures: prompts, responses, tool calls, commands, edits, approvals and token counts.[19] That is a provenance record of unusual richness. It can also be sent to the vendor's hosted cloud rather than kept locally, depending on how the tool was set up. The richest audit trail in the world is of limited use to your auditor if it sits on infrastructure your organisation does not govern.

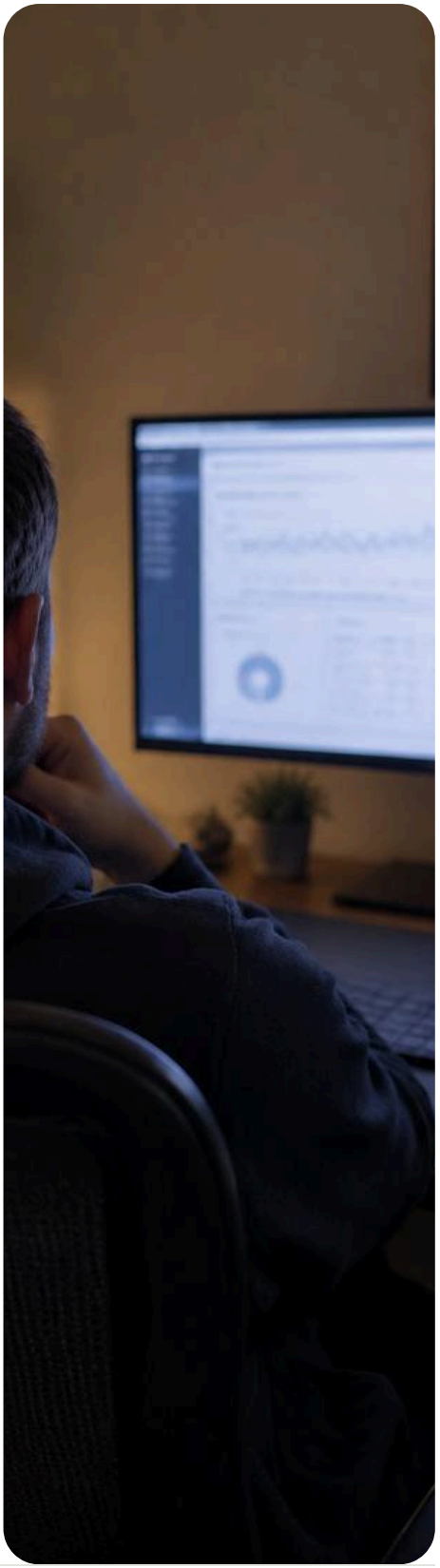
This is the point where we think the perimeter becomes part of the control set. If you cannot reach, retain or produce a provenance record on demand, someone else is running that control on your behalf, on their terms.

Facts that agents write for themselves need a rule of their own

There is one more case to plan for. Agents increasingly write memories from their own conversations. When they do, the source is a conversation, and the approver is often nobody. If your store accepts agent-written facts, decide now whether they need the same approval as facts written by people, and record which is which.

A provenance test you can run this week on one wrong answer

Take one wrong answer an agent gave in the last quarter. Try to trace it to the memory behind it, and from that memory to its source and its approver. Note where the trail stopped. If it stopped at a system boundary, you have found the place your provenance control ends.



19 The Next Web, Asymptote Labs raises $5.5M, 20 Oct 2026; Beacon's capture list as described in the research summary for this assessment; its hosting options from the Beacon documentation, read 5 Nov 2026

Control 4 · Change history

Context should change the way code changes: a diff, a reviewer, a merge and a way back

The rule for changing context asks for nothing your engineers do not already do every day: rules "should change through the same steps as code: a proposed diff, a named reviewer, a merge, and a revert path."[32]

The case for change history is easiest to see from the other side. Imagine a fact in shared context is wrong, and several agents have been reading it for a week. You need to know what the fact said before it changed, who changed it, and every agent that read the bad version. Then you need to put the old fact back.

Without a history, each of those steps means interviewing people. With one, someone can answer each from a record.

Restarting an agent leaves a poisoned memory in the store

Security writers made the case urgent in October. One argued that after memory poisoning an agent "needs more than a restart."[37] Another described persistent poisoning as "the incident that comes back after you close it."[55] A third called it the prompt injection that survives the session.[56]

The common thread is that a poisoned memory outlives the session that wrote it. Restarting the agent clears its working state and leaves the bad fact in the store, ready for the next run to read.

What defending the write path looks like

One security guide recommended signatures and context isolation for persistent agent memory.[36] A paper published in October proposed "origin-bound authority" for long-term agent memory.[57] Both are ways of making the store refuse a write it cannot attribute, which is change history and approval working together.



Articles on memory poisoning published in the window, listed by title

After memory poisoning, an AI agent needs more than a restart
SC World, 23 Oct[37]

Persistent memory poisoning: the incident that comes back after you close it
Intertrace, 7 Oct[55]

Memory Poisoning Is the Prompt Injection That Survives the Session
dev.to, 24 Oct[56]

Securing Persistent Agent Memory: Prevent Poisoning with Signatures and Context Isolation
CodeCrux, 2 Nov[36]

32 Trail, 19 Oct 2026 · **36** CodeCrux, 2 Nov 2026 · **37** SC World, 23 Oct 2026 · **55** Intertrace, 7 Oct 2026 · **56** dev.to, 24 Oct 2026 · **57** doi.org, Securing LLM-Agent Long-Term Memory Against Poisoning, 18 Oct 2026

Control 4 · Change history

History that depends on someone remembering to keep it has gaps exactly where you will need it

Most teams have some change history for context already. An instruction file in a repository has commits. A prompt library in a wiki has page history. A memory store may keep an event log. The question is whether that history is complete, and whether the system keeps it or a person does.

History kept by people has gaps exactly where you need it. The urgent fix made at night, the quick correction pasted straight into the store, the fact an agent wrote for itself: those are the changes most likely to skip the process, and the ones most likely to need reversing.

Context kept in files gets diffs for free, and a memory store has to be checked

Teams that keep context as files under version control get diffs and reverts nearly for free, though version control on its own does not stop an agent reading a change before anyone has approved it. Teams that keep context in a memory store need to check what the store records. Ask your platform to show you the difference between yesterday's version of a memory and today's, and who made the change. If it cannot, your change history is the event log, and an event log is hard to revert from.

A recovery drill to run once before you need it

Run this once, on purpose, before you need it. Choose a fact in shared context. Change it to something harmlessly wrong. Then time how long your team takes to answer four questions.

- What did the fact say before?
- Who changed it, and who approved the change?
- Which agents read the wrong version?
- How do we put the old version back?

Every question you cannot answer from a record is a gap in the control. Every question that needs a person's memory to answer will fail the resignation test.



Four controls

Each skipped control fails quietly at first and expensively later

None of the four controls fails loudly when it is missing. The agent keeps answering. The cost builds where nobody is looking, and arrives as one bad week.

This table is our reading of how each gap tends to show itself. Use the last column to decide who in your organisation should care first, and send it to them.

Table 4. What it costs to skip each control

Control skipped	What happens first	What it costs later	Who notices
Owner	Facts go in and never come out. Conflicts sit in the store unresolved.	Agents answer from stale or contradictory context, and nobody has the authority to fix it.	The customer, then the team that inherits the agent
Approval	Anyone who can write can teach every agent. Errors reach agents immediately.	A wrong or poisoned fact shapes answers before anyone reviews it. Cleanup means finding every answer it touched.	Security, then legal
Provenance	Wrong answers can be traced only as far as the agent.	Incidents end in apologies instead of fixes. Audit requests take weeks of reconstruction.	The auditor
Change history	Corrections overwrite. Nobody can say what a fact said last week.	A bad change cannot be reverted cleanly. A restart leaves the bad fact in place.	Whoever runs the incident

Shakudo's reading. The evidence behind each row is on pages 24 to 31.

If you can fund only one control this quarter, fund approval. It is the only one of the four that prevents an error rather than helping you clean one up.

If you can fund two, add change history, so that the errors approval misses can be reversed instead of repaired by hand.

Every control in this chapter is a rule your organisation writes, and a rule binds only where your organisation governs.

The fifth question behind the four controls: where does the context live?

Ownership, approval, provenance and change history are decisions. A platform can record and enforce them, but only inside the estate where it runs. If the context your agents depend on is assembled on infrastructure you do not govern, your history and your provenance stop where your estate stops.

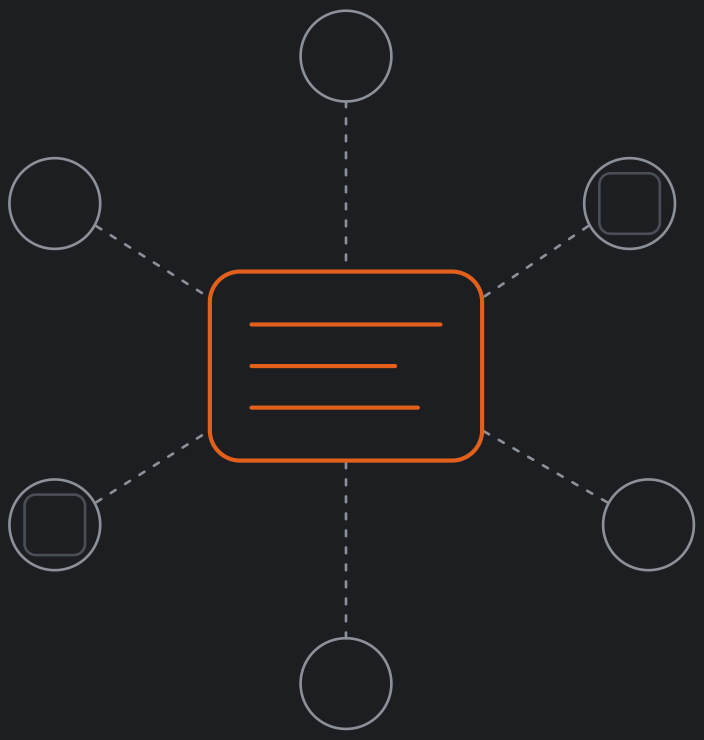
That is why the assessment that follows counts location as part of readiness, and why it is the part of the problem Shakudo works on.

Shakudo is a sovereign AI platform that runs inside your own VPC, on-prem or air-gapped. It runs AI agents, RAG pipelines and open-weights models on compute you own, and deploys wherever Kubernetes runs. Across AWS, Azure, GCP, Oracle or private cloud, data, models and context stay inside the perimeter.

Shakudo does not write your approval rules for you. It keeps the ground those rules run on inside your estate. More at shakudo.io/product/sovereign-ai.

05

When several people and their agents share one context



Pages 34 to 39

Multiplayer AI puts several people and their agents on the same context. Every gap in the four controls becomes a gap that everyone in the room shares.

Multiplayer AI is the name that spread in October for the collaborative version of enterprise AI, and the name misleads. It sounds like several agents working together. Its defining feature, as one analyst post conceded on 13 October, is "human participation."[38]

A launch post at the end of the month drew the same line: multi-agent AI is agents coordinating with each other, while "Multiplayer AI is about people working together."[1]

This chapter is short, because multiplayer AI is not a new problem for this assessment. It is the same problem with more people in it. The experience your business leaders will ask for rests on shared context, and shared context rests on the controls in chapter 4.

The chapter argues one thing the launch material mostly did not say: sharing a session is a grant of access, and it hands over context along with it.

1 Ramp, 30 Oct 2026 · **38** Forrester, Multiplayer AI: Your Impending (And Accidental) Standard, 13 Oct 2026

Shared context in multiplayer AI

The October definitions of multiplayer AI all put shared context at the centre, and few of them say who governs it

Read the definitions published in October for what they assume. Each one assumes a shared, persistent store of company context that many people and many agents read from.

One analyst definition is typical: “The idea behind multiplayer AI is to bring humans, AI, and 'context' together into a persistent single place for everyone to work together.”[38] It then lists what context includes: “conversation history, previously shared documents, permissions, organizational structure, prior decisions, active projects, etc.”

That list is a description of the most sensitive context a company holds. Permissions, organisational structure and prior decisions are exactly the facts a data leader would want owned, approved and traceable.

A launch post on 30 October described the same store from the user's side. Teammates “work with AI in the same sessions and projects, start from the same company context, and reuse and improve each other's best workflows.”[1] The phrase “the same company context” carries the whole idea.

How the strong and weak definitions of multiplayer AI differ

A post on 1 November set out the strong version: multiplayer AI is “when a company's people and its AI agents share the same work, and the agents do that work themselves, on computers of their own.” It also named the weak one: “Most products using the word mean something smaller: an AI you can talk to in a group chat.”[5]

We agree with that line, for a reason its author did not need to make. A group chat with an AI in it is an interface. The strong version needs shared identity and shared memory that persist after the chat ends, and both of those are governance problems before they are product features.

A guide to multiplayer coding puts the distinction this way: the idea “is about who can participate in the work, rather than how many agents are running.”[39] Who can participate is a permissions question. Your data team already answers permissions questions for a living.

1 Ramp, 30 Oct 2026 · **5** Dock, 1 Nov 2026 · **38** Forrester, 13 Oct 2026 · **39** Pairon, What is multiplayer AI coding?, 12 Oct 2026

Shared context in multiplayer AI

An estate that buys several collaboration products can end up with a separate shared context inside each one

Shared context arrived in October as a feature of individual products. In our reading, each product keeps it inside its own walls, which is pattern 2 from chapter 2 at vendor scale.

Five large collaboration and platform vendors shipped shared context under their own names between 7 and 28 October: a protocol grounded in one vendor's work graph, a single assistant with the same memory and skills across an office suite, digital teammates with shared enterprise context in a meetings product, and agent platforms for analytics and for the enterprise.[40-44] The panel on this page lists them by date.

What a buyer should notice

Each launch places the shared context where that vendor's product can reach it: in a work graph, across an office suite, inside a meetings product. A company that uses three of them has three stores of shared context, each with its own owner model, approval model and history, or none.

The question for a data leader is the one from chapter 3. Which of those stores is the truth when two of them disagree, and who decides?

The practical conclusion is to decide, before you buy, where the shared context will live and whose controls will apply to it.

Collaboration launches in October that included shared context

7 Oct · Atlassian
AMP, the Agentic Multiplayer Protocol, grounded in the Teamwork Graph.[40]

7 Oct · Cisco
Webex: people, digital teammates and shared enterprise context.[41]

8 Oct · Google
One Gemini agent with the same memory and skills across Workspace.[42]

22 Oct · ThoughtSpot
AgentSpot: "From Chatbots to Multiplayer AI."[43]

28 Oct · OpenAI
Introducing OpenAI Frontier, for enterprise agents.[44]

40 Atlassian via Business Wire, 7 Oct 2026 · **41** Cisco, Cisco Unveils New Agentic Collaboration Experiences, 7 Oct 2026 · **42** Google Cloud, Gemini at Work 2026: Introducing Gemini agent, 8 Oct 2026 · **43** ThoughtSpot, Introducing AgentSpot: From Chatbots to Multiplayer AI, 22 Oct 2026 · **44** OpenAI, Introducing OpenAI Frontier, 28 Oct 2026

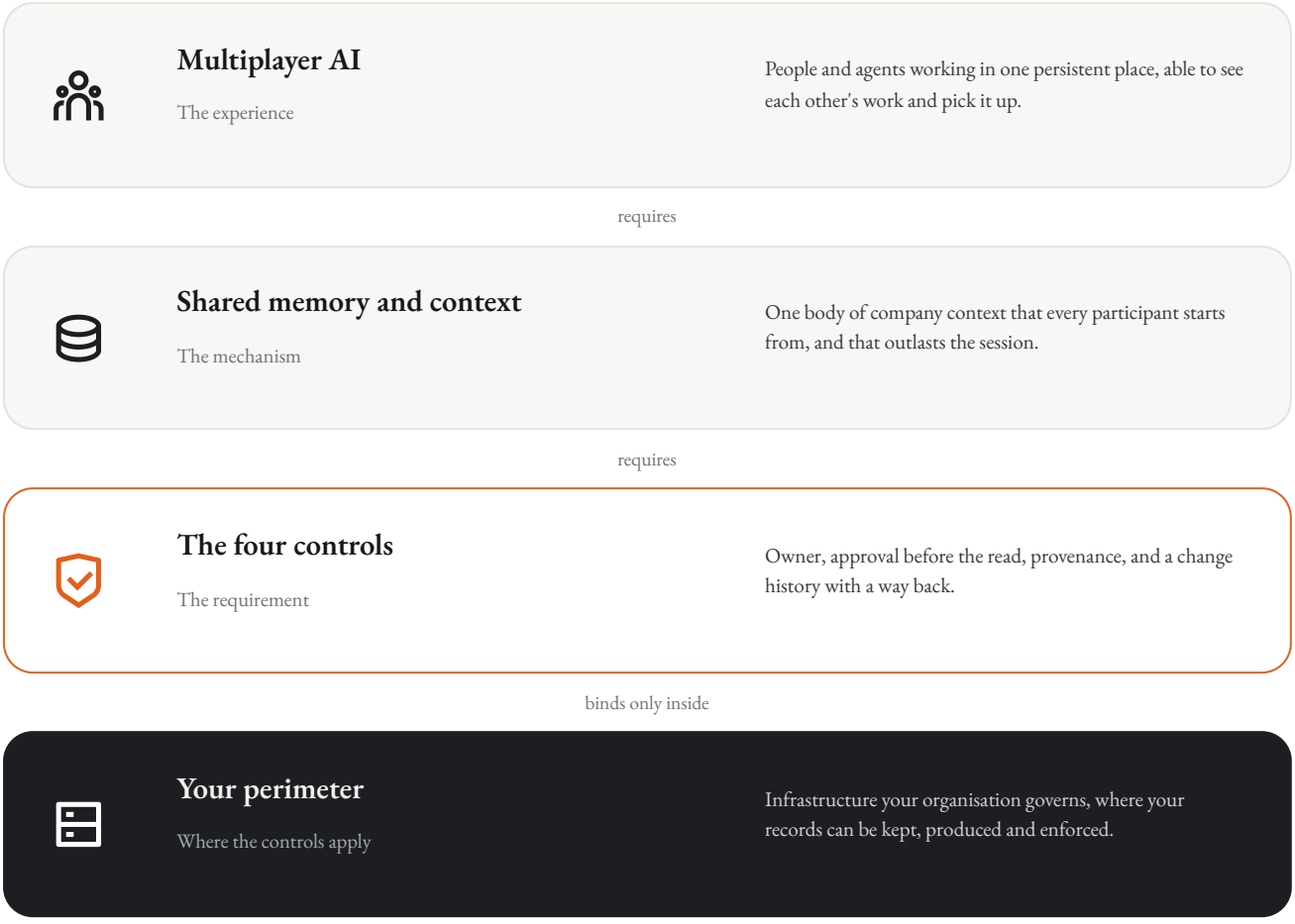
Shared context in multiplayer AI

Multiplayer AI is what users see, and the layers underneath it are the part you answer for

Seen from the top, multiplayer AI is an experience: a shared session, a workspace where agents are members, a thread a colleague can pick up. Seen from underneath, it is a stack, and each layer needs the one below it.

Shared context and shared workflows matter more than shared chats, because they keep working after the conversation ends.[1] The figure follows that idea downwards.

Figure 4. What multiplayer AI stands on



Sources: Forrester, 13 Oct 2026, and Ramp, 30 Oct 2026, for the top two layers.[38, 1] The lower two layers are Shakudo's argument, set out in chapters 4 and 8.

1 Ramp, Multiplayer AI: How Teams Collaborate on AI Work, 30 Oct 2026 · **38** Forrester, 13 Oct 2026

Shared context in multiplayer AI

Sharing a session is a grant of access, and it hands over everything in the conversation with it

The multiplayer tools' own documentation is more candid than their launch posts. Read it, and the governance problem is written in plain sight.

Tools that make single-player coding agents multiplayer shipped through October. In one, released on 27 October, a teammate's join command pulls in the conversation and the uncommitted worktree, and each person keeps their own session with the shared thread as context.[45] Nobody has to commit anything or paste a summary.

Notice what it shares. Everything in the conversation becomes the joining teammate's context: every instruction, every correction, every fact the first person typed in. None of it passed an approval step on the way, because a conversation does not have one.

The tools' own documentation describes sharing as a grant of access

An Apache-licensed tool from one of the large data platforms lets you share a live session by link, co-drive it or fork it, and its documentation says that "Edit is a code-execution grant on the host."[46] Another states that every tool runs on the host's machine and the host approves what guests ask for.[47]

Read literally, both describe sharing as one person giving others the power to act through their environment, with their context.

What today's coding tools can and cannot share

One October guide described the limit: "As of October 2026, a Claude Code share link is a read-only view and a Codex CLI session is driven by one person, so 'shared workspace' always means something built around them."[48] The single-player tools are being stretched to hold several people, and the stretching is happening in each team's own tooling.

Controls on what agents may do, and controls on what they may know

One comparison of governance platforms named four controls for multi-agent estates: agent identity, agent-to-agent authorisation, spend limits and kill switches.[9] Those govern what agents may do. The four controls in this assessment govern what agents may know. A multiplayer estate needs both sets, and in the launch material we read, the second was rarely the default.

9 Lyzr, 19 Oct 2026 · **45** Delta, Claude Code is now multiplayer, 27 Oct 2026 · **46** Databricks Answers, What tools let multiple people collaborate in one AI coding agent session?, 9 Oct 2026; the tool is Omnigent · **47** claude-mods, shared-session, 16 Oct 2026 · **48** Eigen, Shared Workspaces for Claude Code and Codex: What Your Team Can Actually Share in 2026, 22 Oct 2026

A handoff is the resignation test in miniature, because every time work passes between people its context either travels under control or gets rebuilt from memory.

Shakudo's reading of multiplayer AI

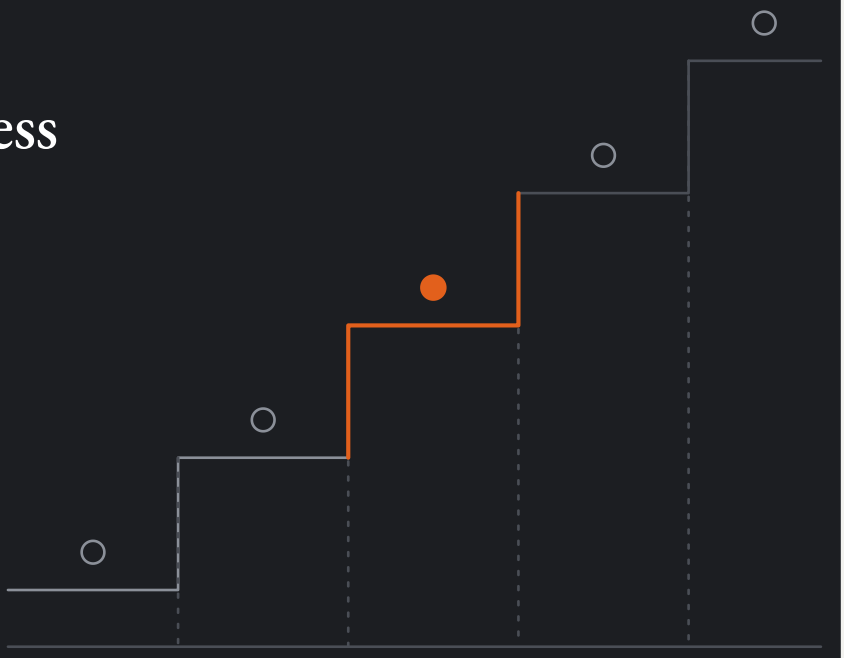
One October launch post argued that "handoffs are where multiplayer AI pays off first."[1] We would add the other side of that: a handoff is also where an estate without the four controls loses the most context, a little at a time.

The next chapter turns the argument so far into four stages, so that you can see where your estate sits before you score it. Each stage is defined by who enforces the controls: nobody, a document, a policy people follow, or the platform itself.

1 Ramp, 30 Oct 2026, ramp.com/blog/multiplayer-ai

06

Four stages of context readiness



Pages 40 to 46

The stage teams most often overestimate is stage 3, because a policy everyone follows looks exactly like a control the platform enforces, until somebody new arrives.

The four stages describe how far the four controls have got in your estate. At stage 1, ad hoc, context lives in personal chat histories. At stage 2, documented, it is written down with no owner and no approval. At stage 3, governed, all four controls exist and people follow them. At stage 4, production-grade, the platform enforces them.

The difference between stages is who does the enforcing: nobody, a document, a policy, or the system. That is the question to keep in mind as you read, because it is also how the scoring on pages 49 to 52 tells a Partly from a Yes.

The stages are not grades. Most teams got real value at stage 1, and stage 2 is honest progress. The aim is to know where you are, and what the next stage costs.

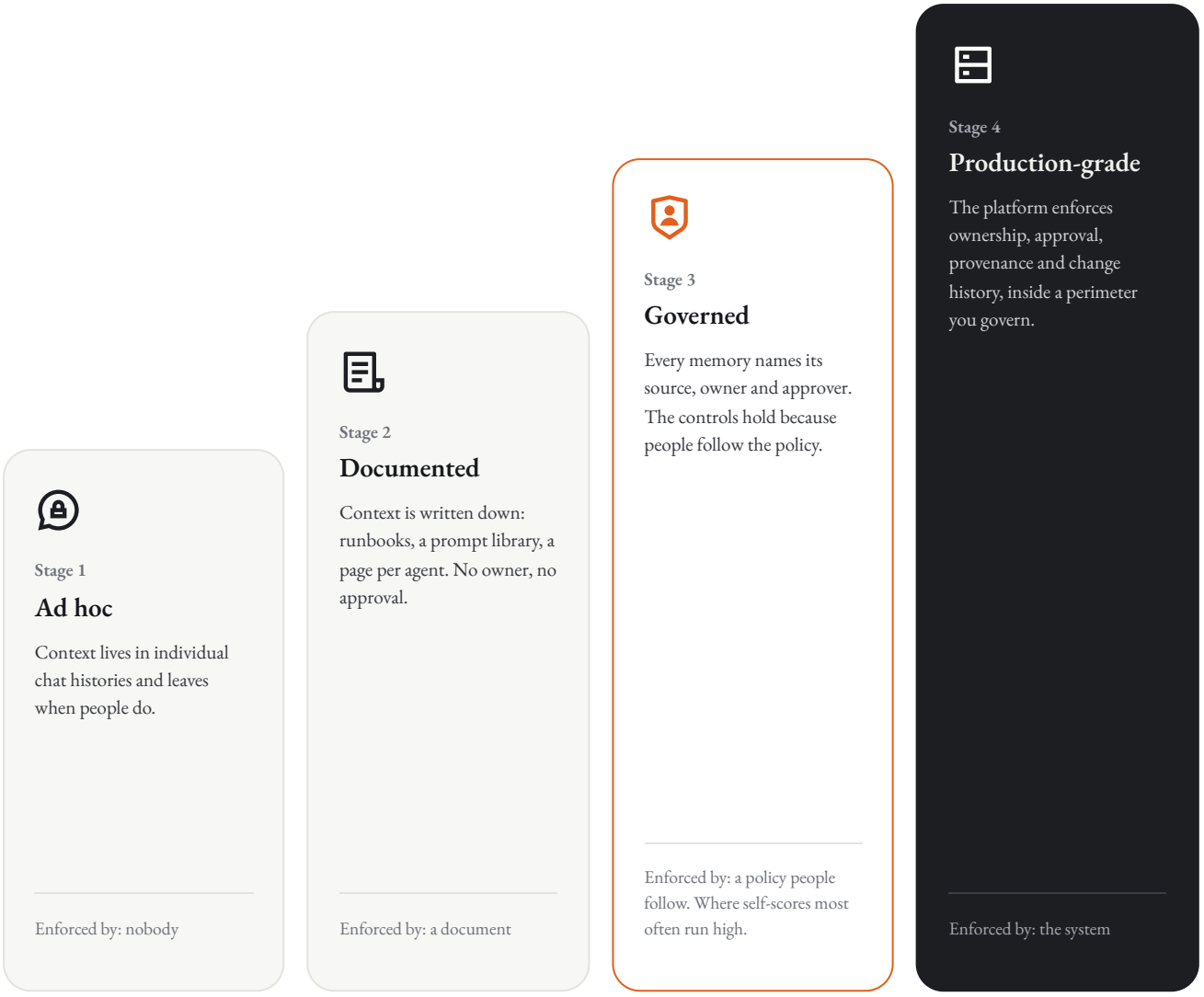
Four stages

Each stage moves enforcement one step further from a person's memory

Read the figure from left to right. Each step up adds a control or moves an existing control from habit into a system. Each column stands taller than the last because more of the four controls can be evidenced without asking anyone.

Nothing on the left of the figure is wrong. It is fragile, and the fragility is exactly the one the resignation test exposes.

Figure 5. The four stages of context readiness



Shakudo's model. The controls at stages 3 and 4 follow Trail's governance policy, 19 Oct 2026.[32]

32 Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

Stage 1 · Ad hoc

At stage 1 a resignation costs the most, and nobody can tell you how much it cost

Each person teaches their own agent, in their own account, and the lessons stay there.

This is where almost every team starts, and it is where real value was first found. The first wave of AI, as one October launch put it, "gave every employee a chatbot."[43] The difficulty begins when the organisation starts to depend on what those chatbots learned.

At stage 1, when two teams need the same context they build it twice, and neither knows the other did. When an agent gives a wrong answer, the trail ends at a person, sometimes a person who has already left. The context rebuilding tax from chapter 2 is paid in full, and it is paid invisibly.

What stage 1 gets right, and why it is worth keeping

Speed. Nobody waits for anybody. The people closest to the work shape the agent directly, and the agent improves quickly. Any move out of stage 1 should try to keep as much of that speed as it can, or people will drift back.

How to move from stage 1 to stage 2

Write it down. Start with the most-used agent and capture what its builder knows: the instructions that matter, the sources it reads, the mistakes it has been taught to avoid. Put that somewhere the team can open. You will not have governance yet. You will have something a second person can read.

Signs that your estate is at stage 1

- The best prompts live in personal chat histories.
- Asked where an agent's instructions are kept, people name a person.
- Two teams have built agents that answer the same question differently.
- Nobody can list every source of context a production agent reads.
- Offboarding a builder has changed how an agent behaves.

43 ThoughtSpot, Introducing AgentSpot: From Chatbots to Multiplayer AI, 22 Oct 2026

Stage 2 · Documented

At stage 2 the context has an author and no approver, and an auditor will spot the difference in one question

Somebody sensible noticed stage 1 and started writing things down. That is real progress, and an easy place to stop.

Stage 2 looks like runbooks, a shared prompt library, a page per agent in the wiki, perhaps an instruction file in a repository. A second person can now read what the first person knew. The resignation test is passed in part: the knowledge survives on paper.

What stage 2 lacks is a decision about which version is right. A written fact has an author. It does not have an approver, and it does not have an owner who can retire it. The failure case from chapter 3, where two agents hold conflicting facts and nothing says which is current, is the normal state of a stage 2 store.[29]

How to move from stage 2 to stage 3

Put a gate on the write path. The remedy that came up most often in the window was a controlled write path, which one October analysis called a write gate.[17] Decide who may approve each kind of fact, name an owner for each source, and make the approval visible before an agent reads the change.

Three auditor questions, and the candid stage 2 answers

Who approved this fact?
No one approved it. Someone wrote it down, which is a different thing.

The wiki and the agent's store disagree. Which is current?
Usually whichever was edited last, if anyone can say which that was.

When was this last checked against its source?
Most likely the last time something broke.

17 themicrosoftcloudblog.com, Agent memory needs a write gate, not a bigger store, 28 Oct 2026 · **29** Korely, 31 Oct 2026

Stage 3 · Governed

Stage 3 is a real achievement with two weak points, and both of them are people

At stage 3, memory starts to behave like data. When an auditor asks how a fact got into an agent, someone answers from a record.

Every memory carries its source, its owner and its approver. Approval happens before an agent reads a new fact. A change goes through a diff and a named reviewer. The question a dev.to post asked on 2 November, who audits what an agent remembers, has an answer you can hand over.[35]

Reaching stage 3 takes patient, unglamorous work, and it deserves credit. It also deserves a careful look before you score yourself there, because two weak points sit inside it.

The first weak point is the approval queue

Approval before the read creates a queue. If a small group approves every memory for every agent, people wait, and people who wait find workarounds. The usual workaround is a private chat history, which quietly restores stage 1. A stage 3 estate with a slow approval queue is quietly running a stage 1 estate next to it.

The second weak point is the newcomer who never read the policy

At stage 3 the controls hold because people follow them. A team that has never read the policy can still write to the store, and the store will accept the write. A contractor, a newly acquired business unit, or an agent that writes its own memories can each bypass a control that exists only as a rule.

Provenance can also stop at a platform boundary. A fact traces cleanly inside one system and loses its trail on the way into another, as chapter 4 described.

How to move from stage 3 to stage 4

Move enforcement from the policy into the platform. Start with change history, because it is the easiest to make automatic and the one you will need first in an incident. Then make the store refuse writes that have no owner, and refuse reads of anything not yet approved.

35 dev.to, Your AI Agent Remembers Things. Who Audits What It Remembers?, 2 Nov 2026

Stage 4 · Production-grade

At stage 4 the platform does the remembering, so no single person has to

The difference between stage 3 and stage 4 is who does the enforcing.

At stage 4 the platform refuses a write that has no owner. It keeps the history of every change whether or not anyone remembered to. It can show an auditor the provenance of any fact an agent acted on without anyone booking a meeting. The standard for the memory itself is the one an October research title set out: transparent, traceable, deterministic.[33]

At stage 4 the store defends its own write path

A persistent memory is an attack surface. At stage 4 the store accepts only writes it can attribute and that have passed approval, in the spirit of the signatures and context isolation recommended in one October security guide, so that a poisoned fact cannot quietly enter.[36] If one does, the history shows every agent that read it.

The perimeter is part of the control set

Controls enforced inside your estate end at its edge. At stage 4 the context your agents depend on is assembled and kept on infrastructure your organisation governs, so the records the controls produce are yours to retain and produce. This is why the assessment counts location as part of readiness, and it is the ground Shakudo works on.



Signs that your estate is at stage 4

- The store rejects a write with no owner.
- Agents cannot read a fact before it is approved.
- Any agent answer traces to its memory and approver from a record.
- A bad change can be reverted, and every reader of it found.
- All of the above holds inside infrastructure you govern.

33 IBM Research, Transparent, Traceable, Deterministic: Agentic Memory via Knowledge Graphs, 25 Oct 2026 · **36** CodeCrux, 2 Nov 2026

Four stages

The stages side by side: what each control looks like as your estate matures

Find the cell in each row that describes your estate as it runs today, for its weakest team. If your cells fall in different columns, which is normal, your stage is set by the column furthest to the left.

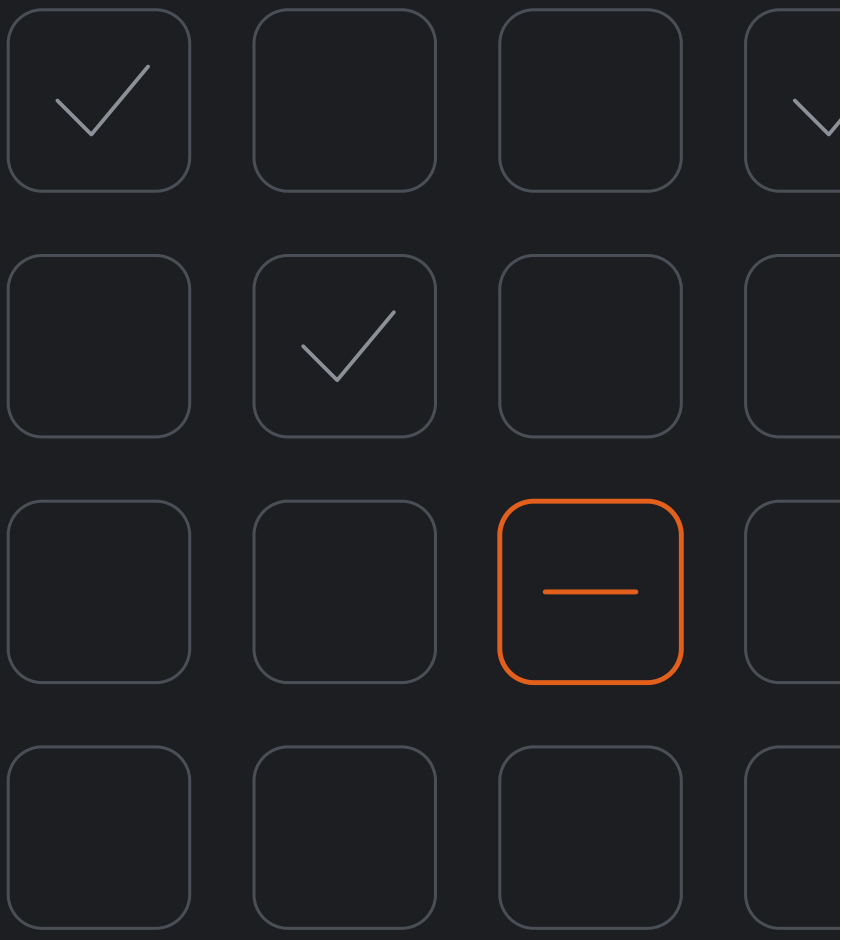
That rule is deliberate. A strong control does not make up for a missing one: approval without provenance gives you a record nobody can trust, and provenance without change history gives you a trace nobody can revert.

Table 5. Each control across the four stages

Control	Stage 1 · Ad hoc	Stage 2 · Documented	Stage 3 · Governed	Stage 4 · Production-grade
Owner	A person, by accident	An author per document	A named role per source	The store refuses unowned writes
Approval	None	None, or review afterwards	Before the read, by policy	Before the read, enforced
Provenance	Ends at a person	Ends at a document	Traces inside one system	Traces across systems, on record
Change history	None	Page history, where it exists	Diffs and reviewers, by habit	Kept and revertible by the system
Location	Personal accounts	Shared drives and wikis	A shared store, wherever it runs	Inside a perimeter you govern

Shakudo's model. The stage 3 and 4 controls follow Trail's governance policy, 19 Oct 2026. [32]

32 Trail, 19 Oct 2026



Score your estate

Sixteen questions in four sections, one for each control. Answer each one Yes, Partly or No, for the estate as it runs today. It works best with two people in the room: the person who answers for the AI estate, and the person who runs the data platform.

1

Answer for the weakest team

Score by the weakest team whose context an agent can act on, so that one good pilot cannot carry the estate.

2

Score 2, 1 or 0

Yes when a system enforces it everywhere. Partly when it holds by policy or in some teams. No, or don't know, scores zero.

3

Read your stage

Add the four section totals on page 53, apply two adjustments, and read your stage on page 54.

Score your estate

Score the estate you run today, never the one on next year's roadmap

Each question names one thing that is either true of your estate or not. The only judgement you have to make is how widely it is true, and what keeps it true.

2

Yes

True for every memory an agent can act on, and a system enforces it. Nobody has to remember.

1

Partly

True for some teams or some agents, or true because people follow a policy.

0

No, or don't know

Not true, or you cannot say. Not knowing scores zero, because an auditor would score it the same way.

Three rules for honest answers

Score the weakest team. If three teams run their agents carefully and a fourth keeps everything in personal chats, the honest answer is the fourth team's. Agents read across teams, and the weakest store is the one an agent will eventually read from.

Policy is Partly. If a control holds because people have read a policy and follow it, score it Partly, however well they follow it. Yes is reserved for controls a system enforces.

Evidence, not intention. For each Yes, someone in the room should be able to show the record that proves it. If the proof is "we would know," score Partly.

An example of an answer that scores Partly

Your team keeps agent instructions in a repository, and every change goes through a pull request with a reviewer. A second team edits its agent's memory store directly. Question D1 asks whether changes to shared context go through a diff and a named reviewer. The honest answer is Partly: true for one team, not for the estate.

Keep your answers. The four section totals matter more than the grand total. Your lowest section is your first project, and page 55 says how to start it.

Section A · Ownership

Who answers for what your agents know?

An owner is the role that can retire a stale fact and settle a conflict. Without one, context only ever grows.

A1 **Could you list every source of context your production agents read from, today, without a round of interviews?**

Instruction files, retrieval indexes, memory stores and searchable drives all count.

☐ Yes 2 ☐ Partly 1 ☐ No 0

A2 **Does every memory or instruction an agent can act on have a named owner?**

Yes means the store records an owner for every item and will not accept one without.

☐ Yes 2 ☐ Partly 1 ☐ No 0

A3 **Is that owner a role rather than a person, so that the ownership survives the person leaving?**

A list of individual names in a wiki scores Partly at most.

☐ Yes 2 ☐ Partly 1 ☐ No 0

A4 **If the builder of your most-used agent resigned on Friday, would the agent know the same things on Monday?**

The resignation test. Anything kept in that person's own account counts against you.

☐ Yes 2 ☐ Partly 1 ☐ No 0

Section A total ☐ / 8

Section B · Approval

Who lets a fact in before an agent reads it?

The control teams skip first, and the only one that prevents an error instead of helping you clean it up.

B1 **Does a new or changed memory reach an agent only after someone has approved it?**

Review after the fact scores No, because approval has to come before the read.

Yes	Partly	No
2	1	0

B2 **Is the approver named, and is the approval recorded where an auditor could find it?**

An approval given in a chat message and never recorded scores Partly at most.

Yes	Partly	No
2	1	0

B3 **Did you decide what is shared and what stays private before sharing began?**

The decision belongs before rollout. Deciding afterwards scores Partly.

Yes	Partly	No
2	1	0

B4 **Does the system refuse an unapproved write, rather than accept it and flag it later?**

This is the write gate. A dashboard that lists unapproved items after agents can read them scores No.

Yes	Partly	No
2	1	0

Section B total / 8

Section C · Provenance

Can you trace a wrong answer back to where it came from?

Provenance turns an incident into a correction. Without it, an investigation ends at the agent.

C1 **Does every memory an agent can act on name the source it came from?**

A document, a table, or the conversation an agent wrote it from.

☐ Yes 2 ☐ Partly 1 ☐ No 0

C2 **Can you trace a wrong agent answer to the memory behind it, and to the decision that let that memory in?**

From a record, without asking the person who might remember.

☐ Yes 2 ☐ Partly 1 ☐ No 0

C3 **When two agents hold conflicting facts, does your store record which one is current?**

The failure case from chapter 3. “Whichever was edited last” scores No.

☐ Yes 2 ☐ Partly 1 ☐ No 0

C4 **Does the trace survive when context crosses a platform, a team, or the edge of your estate?**

If the record stays with a vendor's hosted service, score Partly at most.

☐ Yes 2 ☐ Partly 1 ☐ No 0

Section C total ☐ / 8

Section D · Change history

Can you undo a bad change, and find everyone it reached?

A restart clears an agent's working state and leaves a bad fact in the store. History is how you get it out.

D1 **Does a change to shared context go through a proposed diff and a named reviewer?**

The same steps as code: a diff, a reviewer, a merge and a way back.

Yes	Partly	No
2	1	0

D2 **Can you revert a bad memory change the way you revert a bad commit?**

Retyping the old fact from memory scores No.

Yes	Partly	No
2	1	0

D3 **Does the system keep the history of every change, whether or not anyone remembered to?**

Including direct edits to the store and facts that agents write themselves.

Yes	Partly	No
2	1	0

D4 **If a memory were poisoned, could you list every agent that read it before it was removed?**

The recovery drill on page 31 is a fair way to find out.

Yes	Partly	No
2	1	0

Section D total / 8

Score your estate

Add the four sections, then let your weakest one have the last word

Table 6. **Your totals**

Section	The question behind it	Your total	Out of
A · Ownership	Who answers for what your agents know?		8
B · Approval	Who lets a fact in before an agent reads it?		8
C · Provenance	Can you trace a wrong answer to its source?		8
D · Change history	Can you undo a bad change, and find everyone it reached?		8
All four			32

Adjustment 1 · A total can hide a gap

If any one section scores 3 or less, your stage is Documented at most, whatever your total. Approval without provenance gives you a record nobody can trust. Provenance without change history gives you a trace nobody can revert.

Adjustment 2 · Stage 4 needs all four

To claim Production-grade, every section must score 7 or 8. A platform that enforces three controls and leaves the fourth to habit is still running on convention.

Your lowest section _____

Your stage, after adjustments _____

If the result surprised you, look first at the section that pulled it down. That section is where next quarter's work starts, and page 55 suggests the first move for each.

Reading your score

What your total says about your estate

0 to 7	**Stage 1 · Ad hoc** Context lives with people	Context is already a production dependency, and none of the four controls is in place. **Start** by naming an owner for every source your most-used agent reads.
8 to 15	**Stage 2 · Documented** Written, not approved	The context is written down, but with no owner or approver nobody can vouch for what an agent reads. **Start** by putting a gate on the write path.
16 to 23	**Stage 3 · Governed** Held by policy	All four controls exist, and they hold because people follow them. **Start** moving enforcement into the platform, beginning with change history.
24 to 28	**Stage 3, enforced in places** The edge of stage 4	Some controls are enforced by the system and some by habit. **Start** with the section that still depends on habit, and check where your context physically lives.
29 to 32	**Stage 4 · Production-grade** Enforced by the system	The controls are enforced and provable, the way they are for your data. **Keep** testing the perimeter: know where context is assembled and where it travels.

Before you settle on a band

Apply the two adjustments on page 53. A total of 24 with one section at 3 is a stage 2 estate.

If you scored yourself at stage 3

Ask what happens when a contractor or a newly acquired team writes to the store. If the store accepts it, you are relying on policy, and stage 3 is the right score.

Reading your score

Your weakest section is your first project, and each section has a small first move

A stage tells you where you are. Your lowest section tells you what to do on Monday. Find it in the first column and take the first move as written. Each one is deliberately small, and each is chosen to move a Partly or a No on a specific question.

If two sections tie for lowest, start with approval if it is one of them. It is the only control whose absence lets errors reach agents unseen, and fixing it makes the other three easier to evidence.

Table 7. If your lowest section is...

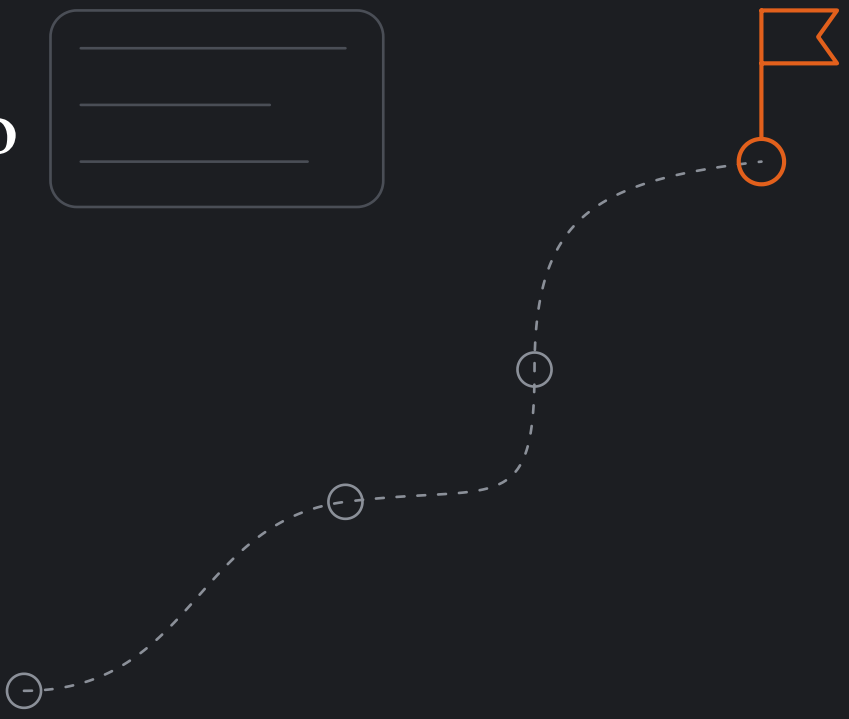
Lowest section	What it usually means	First move	Bring in
A · Ownership	Context grows with nobody able to retire or reconcile it	List every source your most-used agent reads, and name a role to own each. Moves A1 to A3.	Data governance lead
B · Approval	Anyone who can write can teach every agent, immediately	Sort context by the harm a wrong fact could do, and name an approver for the top tier before agents read it. Moves B1, B2.	Security and the owners from A
C · Provenance	Incidents end in apologies instead of fixes	Trace one wrong answer from last quarter end to end, and record where the trail stopped. Moves C2, C4.	Platform engineering, audit
D · Change history	A bad change can be found but not cleanly undone	Run the recovery drill on page 31, on purpose, and fix the first question you could not answer. Moves D2, D4.	Whoever runs incidents

Shakudo's recommendations, built on the controls in Trail's governance policy, 19 Oct 2026.[32]

32 Trail, How should a memory layer for LLM agents be governed?, 19 Oct 2026

08

What to do next



Pages 56 to 60

Start with the agent people would miss most, not the one that is easiest to govern. The controls earn their cost where the dependency is real.

It is tempting to begin a governance programme with a clean, low-stakes agent, because it is easy to show progress. We think that is a mistake. The controls cost time, and the only argument for paying that cost is the dependency the resignation test exposes. Begin where that dependency is largest.

For most estates that is the agent with the most users, or the one whose builder would be hardest to replace.

This chapter sets out a next step for each stage, a set of questions to put to any vendor who wants to hold your shared context, and Shakudo's position on the one part of the problem it works on directly.

It closes with the whole assessment on a single page, for the meeting where you take your score to the people who will fund the work.

What to do next

One move per stage, each chosen to change a specific answer on the scorecard

You do not have to reach stage 4 in one programme. You have to move one stage, prove it on the scorecard, and then decide whether the next one is worth its cost.

From stage 1
to documented

Write down what the builder knows

For your most-used agent, capture the instructions that matter, the sources it reads and the mistakes it has been taught to avoid, somewhere the team can open. Then list every other source of context your production agents read.

Moves A1 and A4.

From stage 2
to governed

Name owners, then gate the write path

Give every source a role as owner. Sort context by the harm a wrong fact could do, name an approver for each tier, and make approval happen before an agent reads a change. Decide what is shared and what stays private before you widen access.

Moves A2, A3, B1, B2 and B3.

From stage 3
to production-grade

Move enforcement into the platform

Start with change history, because it is the easiest to automate and the first thing you will need in an incident. Then make the store refuse writes without an owner and reads of anything unapproved. Bring the store inside infrastructure you govern so that its records are yours.

Moves B4, C4, D2 and D3.

At stage 4
keep it there

Keep testing the controls on purpose after you reach stage 4

Run the recovery drill twice a year. Trace one wrong answer end to end every quarter. Re-score whenever you add a vendor that holds shared context, because each one can move your provenance back to Partly.

Protects C2 and D4.

What to do next

Eight questions to put to any vendor who wants to hold your shared context

1 **Where does the memory live by default?**
And what would we have to change to keep it inside our own estate?

2 **Can the store refuse a write that has no owner?**
Or does it accept the write and leave ownership to a field someone may fill in later?

3 **Does approval happen before an agent can read a new memory?**
Show us, in the product, where a fact waits until it is approved.

4 **What does a provenance record contain?**
Source, owner, approver, when the fact became true and when it was recorded.

5 **Can you show a diff between yesterday's memory and today's, and revert it?**
Including changes made directly in the store and facts the agents wrote themselves.

6 **If a memory is poisoned, how do we find every agent that read it?**
And how long does it take?

7 **When we share a session, what context goes with it?**
The whole conversation, or only what was approved for sharing?

8 **Whose benchmark is that?**
For any accuracy figure in the pitch, ask who ran it and on what data.



Ask the vendor to demonstrate the controls in one afternoon

One October post claimed that the difference between real multiplayer AI and a group chat "comes down to infrastructure, and you can test any product for it in an afternoon."[5]

The same is true of controls. Ask the vendor to do questions 2, 3 and 5 live, on a test store, during the evaluation. A control that cannot be demonstrated in an afternoon is unlikely to hold in production.

The questions are Shakudo's. Each maps to a scorecard question on pages 49 to 52.

5 Dock, Multiplayer AI needs agents with their own computers, 1 Nov 2026

What to do next

Where Shakudo fits: keeping the ground your controls run on inside your estate

Every stage in this assessment comes back to one practical question: where does the context live, and whose rules apply there?

Shakudo's answer is sovereign AI. The Shakudo platform runs inside your own VPC, on-prem or air-gapped. It runs your AI agents, RAG pipelines and open-weights models on compute you own, and deploys on your cloud or on-prem wherever Kubernetes runs. Across AWS, Azure, GCP, Oracle or private cloud, data, models and context stay inside the perimeter.

That matters for this assessment in a specific way. Ownership, approval, provenance and change history are controls your organisation writes. They bind only where your organisation governs. When the store your agents read from runs inside your perimeter, the records those controls produce are yours to keep, query and hand to an auditor.

We want to be exact about the boundary. Shakudo does not write your approval rules, and it is not sold here as a memory product. It keeps the ground those rules run on inside your estate, which is the condition every one of the four controls depends on.

Shakudo holds SOC 2 Type II. J.T. Magen uses Shakudo to run open-source AI on infrastructure the company controls.

Talk to the Shakudo team, and bring your totals

Further reading on shakudo.io

- Sovereign AI
- Multi-cloud deployment
- Air-gapped AI
- The platform
- Sovereign AI reference architecture
- SOC 2 Type II
- Customers

What to do next

The assessment on one page, for the meeting where you ask for the budget

Table 8. Four controls, the cost of skipping each, and what good looks like

Control	The question	Skip it and	Good looks like	Your score
Owner	Who answers for what your agents know?	Stale facts become permanent and conflicts go unresolved	A role owns every source; the store refuses unowned writes	/ 8
Approval	Who lets a fact in before an agent reads it?	Errors and poisoned facts reach agents immediately	A named approver signs off before any read, on record	/ 8
Provenance	Can you trace a wrong answer to its source?	Incidents end in apologies instead of fixes	Every answer traces to a line and a decision, across systems	/ 8
Change history	Can you undo a bad change, and find who it reached?	A bad change cannot be cleanly reverted	Diff, reviewer, merge and revert, kept by the system	/ 8
Location	Where does the context live?	Your records stop at someone else's border	Context and its records sit inside a perimeter you govern	Stage:

32

The highest possible score. Every section must reach 7 or 8 to claim stage 4, and any section at 3 or below caps you at stage 2.

The one sentence to take into the room. Our agents already depend on company context the way our dashboards depend on tables, and this score shows how many of the controls we run on tables we also run on that context.

Sources

Every outside claim in this assessment comes from publications dated 4 October to 3 November 2026, as collected for Shakudo's research summary on shared memory and multiplayer AI. Numbers match the superscripts in the text. Quotations are verbatim. Funding figures are as the press reported them. Status checks made on 5 November 2026 are marked where used.

1 **Ramp** Multiplayer AI: How Teams Collaborate on AI Work. 30 Oct 2026.
ramp.com/blog/multiplayer-ai

2 **Agor (Preset)** Announcing Agor Cloud Open Beta. 13 Oct 2026.
agor.live/blog/agor-cloud-open-beta

3 **Wavelave** How to Run Multiple AIs Together: An Engineering Guide. 10 Oct 2026.
wavelave.ai/run-multiple-ais-together

4 **Mirasim** Multiplayer AI coding agents for your team. 31 Oct 2026.
mirasim.ai/writing/multiplayer-coding-agents

5 **Dock** Multiplayer AI needs agents with their own computers. 1 Nov 2026.
trydock.ai/blog/multiplayer-ai

6 **Membase** How to Share Memory Between Multiple AI Agents (2026 Guide). 3 Nov 2026.
membase.com/blog/how-to-share-memory-between-ai-agents

7 **SiliconANGLE** Uvi raises $7.5M to help AI agents learn business context and run at eight times lower cost. 14 Oct 2026.
siliconangle.com/2026/10/14/uvi-raises-7-5m-help-ai-agents-learn-business-context

8 **Withdrawn** A dev.to item from the research summary that could not be traced to an address when rechecked on 5 Nov 2026. The passage that cited it was removed.

9 **Lyzr** Best Platforms for Governing Multi-Agent Architectures. 19 Oct 2026.
lyzr.ai/blog/platforms-for-governing-multi-agent-architectures

10 **Forrester** Context Engineering And ContextOps: The Missing Foundation For Scaling Agentic AI. 19 Oct 2026.
forrester.com/blogs/context-engineering-and-contextops-the-missing-foundation-for-scaling-agentic-ai

11 **MemTensor on Hugging Face** Persistent Agent Memory Should Make the Next Task Easier. 27 Oct 2026.
huggingface.co/blog/MemTensor/persistent-agent-memory-should-make-the-next-task

12 **c3.ai** Context Rot: Why Long-Running AI Agents Lose Reliability. 27 Oct 2026.
c3.ai/blog/context-engineering-ai-agents

13 **AWS Builder Center** What Is Context Engineering for AI Agents?. 13 Oct 2026.
builder.aws.com/content/3Ke8N6fTt8Xt7z3KZxhs6kzFxQ4

14 **Databricks** Introducing Agent Memory on Databricks. 20 Oct 2026.
databricks.com/blog/introducing-agent-memory-databricks

15 **Databricks** MemCurrent: Designing Persistent Memory for Production Agents. 20 Oct 2026.
databricks.com/blog/memcurrent-designing-persistent-memory-production-agents

16 **Microsoft Community Hub** Architecting Governed Memory State for Production AI Agents with Microsoft Foundry. 29 Oct 2026.
techcommunity.microsoft.com/blog/azure-ai-foundry-blog/architecting-governed-memory-state-for-production-ai-agents-with-microsoft-found/4566999

17 **themicrosoftcloudblog.com** Agent memory needs a write gate, not a bigger store. 28 Oct 2026.
themicrosoftcloudblog.com/2026/10/agent-memory-needs-a-write-gate-not-a-bigger-store

18 **SiliconANGLE** Scuba launches with $34M to help teams and AI agents manage business data. 13 Oct 2026.
siliconangle.com/2026/10/13/scuba-launches-with-34m-to-help-teams-and-ai-agents-manage-business-data

19 **The Next Web** Asymptote Labs raises $5.5M to track and control what AI agents do. 20 Oct 2026. Hosting options from the Beacon documentation, undated, read 5 Nov 2026.
thenextweb.com/news/asymptote-labs-5-5m-seed-ai-agent-security
docs.beacon.sh/deployment/hosting-options

20 **Nexi** Zeroset Raises $5.2M to Give AI Agents Real Workflow State. 6 Oct 2026.
nexi.fund/zeroset-nebula-enterprise-agent-state-2026

21 **IETF Datatracker** draft-agentproto-context-management-architecture-00. 28 Oct 2026. Individual draft, intended status Informational; still at -00 on 5 Nov 2026.
datatracker.ietf.org/doc/html/draft-agentproto-context-management-architecture-00

22 **Founderland** Mosaic launches shared memory to sync 2,000 AI coding agents. 27 Oct 2026.
founderland.ai/articles/mosaic-launches-shared-memory-to-sync-2000-ai-coding-agents-mvpycmye

23 **Eragon** Eragon MemBench: measuring memory for work. 23 Oct 2026.
eragon.ai/news/eragon-membench

24 **past.dev via PR Newswire** past.dev Introduces Frontier Memory for AI Agents: #1 on BEAM, the Largest Public Memory Benchmark. 6 Oct 2026.
prnewswire.com/news-releases/pastdev-introduces-frontier-memory-for-ai-agents-1-on-beam-the-largest-public-memory-benchmark-302899770.html

25 **Prinevo on dev.to** Prinevo Memory: A Context Graph for Long-Term Agent Memory. 29 Oct 2026.
dev.to/prinevo/prinevo-memory-a-context-graph-for-long-term-agent-memory-1ma5

26 **alphaXiv** EvoMemBench: Benchmarking Agent Memory from a Self-Evolving Perspective. 30 Oct 2026.
alphaxiv.org/abs/2605.18421

27 **Semurg** Semurg: local persistent memory for AI agents over MCP. 13 Oct 2026.
one.semurg.io/blog/semurg-agent-memory-over-mcp

28 **dev.to** Solving AI Agent Amnesia: Building a Self-Hosted Shared Memory Mesh for Coding LLMs. 30 Oct 2026.
dev.to/ahwai_zhang_cebee099d65cb/solving-ai-agent-amnesia-building-a-self-hosted-shared-memory-mesh-for-coding-llms-4f78

29 **Korely** Shared agent memory: when several agents read the same facts. 31 Oct 2026.
korely.ai/blog/shared-agent-memory

Sources, continued

30 **Cognee** How to Build Stateful Multi-Agent Systems in Production (2026). 29 Oct 2026.
cognee.ai/stateful-multi-agent-systems

31 **ABATA AI** Mem0 v2.0 & Graphiti: Persistent Temporal Knowledge Graphs for Multi-Agent Systems. 18 Oct 2026.
abata.tech/mem0-v2-graphiti-persistent-temporal-knowledge-graphs-enterprise-agents

32 **Trail** How should a memory layer for LLM agents be governed?. 19 Oct 2026.
trailhq.com/pages/how-should-a-memory-layer-for-llm-agents-be-governed

33 **IBM Research** Transparent, Traceable, Deterministic: Agentic Memory via Knowledge Graphs for ISWC 2026. 25 Oct 2026.
research.ibm.com/publications/transparent-traceable-deterministic-agentic-memory-via-knowledge-graphs

34 **QueryNow** AI Agent Audit Trail: What to Record and How Long to Keep It. 2 Nov 2026.
querynow.com/ai-agents/governance/audit-trails

35 **dev.to** Your AI Agent Remembers Things. Who Audits What It Remembers?. 2 Nov 2026.
dev.to/lanbass869cell/your-ai-agent-remembers-things-who-audits-what-it-remembers-50l

36 **CodeCrux** Securing Persistent Agent Memory: Prevent Poisoning with Signatures and Context Isolation. 2 Nov 2026.
codecrux.com/blog/securing-persistent-agent-memory-prevent-poisoning-with-signatures-and-context-isolation

37 **SC World** After memory poisoning, an AI agent needs more than a restart. 23 Oct 2026.
scworld.com/perspective/after-memory-poisoning-an-ai-agent-needs-more-than-a-restart

38 **Forrester** Multiplayer AI: Your Impending (And Accidental) Standard. 13 Oct 2026.
forrester.com/blogs/multiplayer-ai-your-impending-and-accidental-standard

39 **Pairon** What is multiplayer AI coding?. 12 Oct 2026.
pairon.dev/guides/multiplayer-ai-coding

40 **Atlassian via Business Wire** Atlassian Introduces AMP: The Agentic Multiplayer Protocol to Power Human-AI Collaboration. 7 Oct 2026.
business.wapakdailynews.com/wapakdailynews/article/bizwire-2026-10-7-atlassian-introduces-amp-the-agentic-multiplayer-protocol-to-power-human-ai-collaboration

41 **Cisco via BusinessPressWire** Cisco Unveils New Agentic Collaboration Experiences. 7 Oct 2026.
businesspresswire.com/2026/10/07/cisco-unveils-new-agentic-collaboration-experiences

42 **Google Cloud** Gemini at Work 2026: Introducing Gemini agent. 8 Oct 2026.
cloud.google.com/blog/products/ai-machine-learning/welcome-to-gemini-at-work-2026

43 **ThoughtSpot** Introducing AgentSpot: From Chatbots to Multiplayer AI. 22 Oct 2026.
thoughtspot.com/press-releases/introducing-agentspot-from-chatbots-to-multiplayer-ai

44 **OpenAI** Introducing OpenAI Frontier. 28 Oct 2026.
openai.com/index/introducing-openai-frontier

45 **Delta** Claude Code is now multiplayer. 27 Oct 2026.
delta.dev/blog/claude-code-multiplayer

46 **Databricks Answers** What tools let multiple people collaborate in one AI coding agent session?. 9 Oct 2026.
answers.databricks.com/tools-let-multiple-people-collaborate-one-ai-coding

47 **claude-mods** shared-session: multiplayer Claude Code sessions by link. 16 Oct 2026.
claude-mods.com/shared-session

48 **Eigen** Shared Workspaces for Claude Code and Codex: What Your Team Can Actually Share in 2026. 22 Oct 2026.
geteigen.space/blog/shared-workspaces-claude-code-codex

49 **Kylon** AI Agents Were Built for One Person. We Built Kylon for Teams.. 4 Oct 2026.
kylon.io/blog/ai-agents-built-for-teams

50 **arXiv** Diagnosing and Mitigating Context Rot in Long-horizon Search. 25 Oct 2026.
arxiv.org/html/2606.29718

51 **Traversaal** Context Rot: Why Longer Context Windows Degrade Agent Reliability (And How to Fix It). 12 Oct 2026.
blog.traversaal.ai/context-rot-context-window-agent-reliability

52 **arXiv** Muffin: A Shared Memory Buffer for Asynchronous AI Agents. 22 Oct 2026.
arxiv.org/html/2610.31355

53 **arXiv** Stashbird: Episode-Grounded Lifecycle Reconciliation for Conversational Agent Memory. 17 Oct 2026.
arxiv.org/html/2609.34242

54 **Backslash Security** What an AI Agent Audit Trail Must Capture for the EU AI Act and DORA. 7 Oct 2026.
backslash.security/learning/what-an-ai-agent-audit-trail-must-capture-for-the-eu-ai-act-and-dora

55 **Intertrace** Persistent memory poisoning: the incident that comes back after you close it. 7 Oct 2026.
intertrace.ai/blog/research-persistent-agent-memory-poisoning

56 **dev.to** Memory Poisoning Is the Prompt Injection That Survives the Session. 24 Oct 2026.
dev.to/suraj_khaitan_f893c243958/memory-poisoning-is-the-prompt-injection-that-survives-the-session-nfh

57 **doi.org** Securing LLM-Agent Long-Term Memory Against Poisoning: Non-Malleable, Origin-Bound Authority with Machine-Checked Guarantees. 18 Oct 2026.
doi.org/10.66977/xsci.2610.001o

58 **Bellows** Live-share multiple coding agents side-by-side and approve changes from any browser. 2 Nov 2026.
bellowsai.app/articles/live-share-multiple-coding-agents

59 **Zenodo** Data and code for: Who wins depends on who judges: re-grading published LoCoMo and LongMemEval answers of agent memory systems. 12 Oct 2026.
doi.org/10.5281/zenodo.23316414

About the evidence

How the evidence was gathered, and what it can and cannot carry

The research window and the corpus behind it

The research behind this assessment covered the thirty days from 4 October to 3 November 2026. It ran 44 search queries, which returned 436 results from 435 unique addresses across 305 domains. Of those, 430 carried a publication date inside the window. The full text of 68 pages was read, along with targeted reads of five key sources.

The date filter is a publication date. It does not prove when a page was last edited, and one source shows an older page date.

How much confidence the evidence supports

Confidence is high for what was said and by whom. It is medium for market size and revenue claims, which come from vendor and press material that nobody has audited independently.

Known limits of the evidence

- Vendor material dominates the volume. Only Forrester, IBM Research and the academic papers are independent of a product. We have treated vendor claims as claims.
- Funding figures for Zeroset, Scuba, Uvi and Asymptote Labs are press-reported and were not checked against filings.
- The Mosaic headline speaks of 2,000 AI coding agents. The article body, read on 5 November 2026, gives about 2,000 installs and 37 organisations syncing, and those are the figures we use.
- Two items in earlier drafts, a dev.to incident from 28 October and a claim about Beacon's installer default, could not be traced to an address when rechecked on 5 November 2026, and were withdrawn.
- The research maps the market conversation. It is not a competitive analysis.

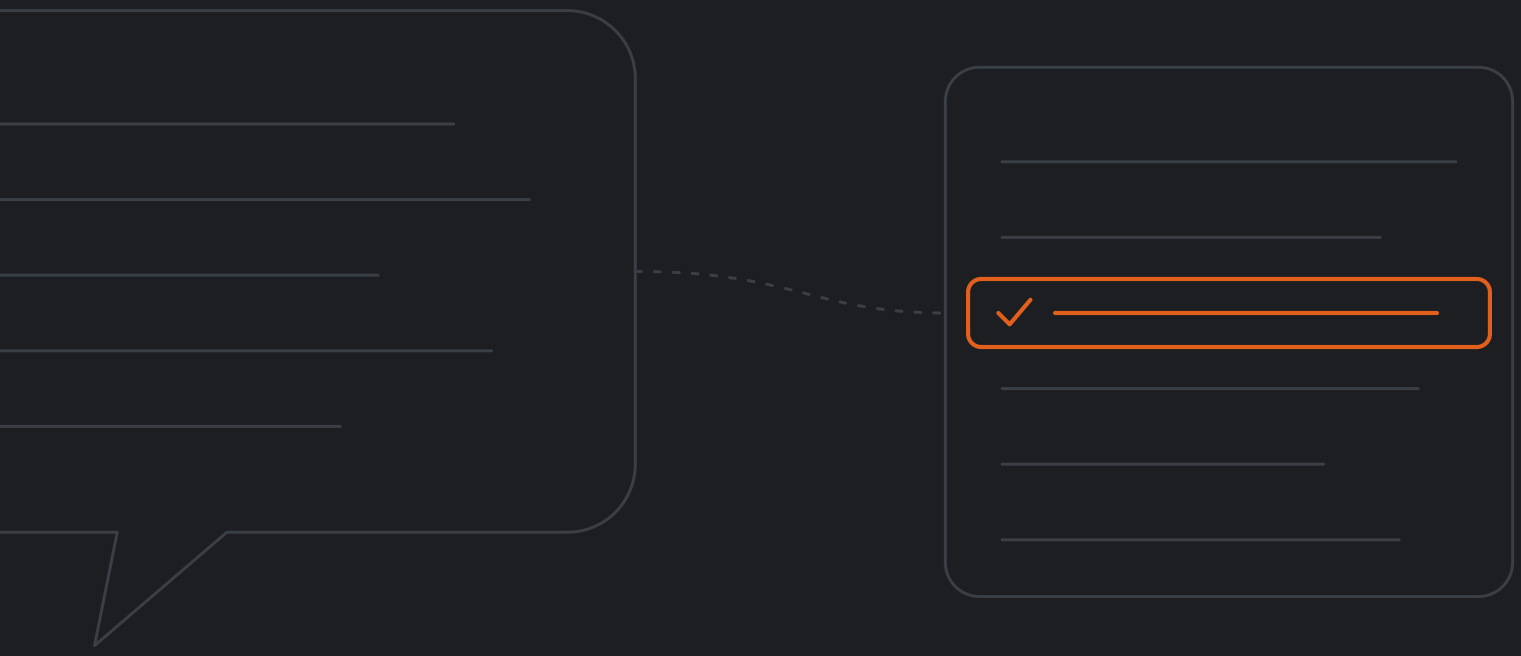
Which parts of this assessment are Shakudo's own argument

The four stages, the scoring model, the recommendations and every sentence marked as our view or our reading are Shakudo's. They are argument, offered for you to test against your own estate.

About Shakudo

Shakudo is a sovereign AI platform. It runs inside the customer's own VPC, on-prem or air-gapped, and runs AI agents, RAG pipelines and open-weights models on compute the customer owns. It deploys on the customer's cloud or on-prem wherever Kubernetes runs, across AWS, Azure, GCP, Oracle or private cloud, with data, models and context staying inside the perimeter. Shakudo holds SOC 2 Type II.

To talk about your score, visit shakudo.io/contact-us.



The Context Readiness Assessment

Published by Shakudo, November 2026.
shakudo.io · shakudo.io/contact-us

SHAKUDO

© 2026 Shakudo. Outside sources are quoted for commentary and are credited on pages 61 and 62. Benchmark and funding figures are the claims of the companies and publications named.