13

THE STACK BEHIND THE AI COWORKER

What an Agent Remembers, and Cannot Forget

Dr Peter McCann Strain, CTO and senior AI engineer, DPhil/PhD in AI from Oxford University

A customer asks to be forgotten. The honest answer is not yes or no; it is a receipt.

An essay in the series **Architecting the AI Coworker**.
Approx. 24 minute read · Essay 13 of 22

Dr Peter McCann Strain
CTO, DPhil/PhD in AI from Oxford University

> A customer named Maria writes in and asks to be forgotten. It is a short request, polite, the kind a support team sees every day in 2026. Somewhere in the building, an operator now has to answer one question. Has the agent forgotten her?

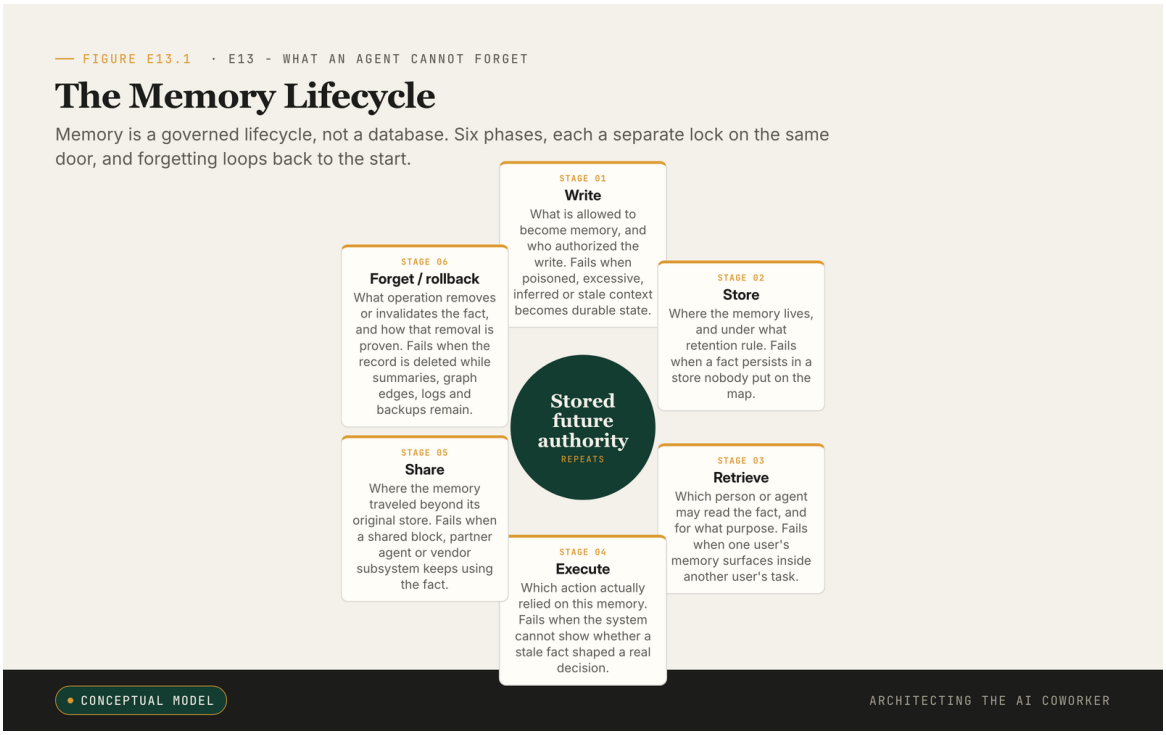
Maria is a composite, a worked case assembled to test the argument and not a report of any real incident. The mechanics under her request, though, are exactly the ones a real operator faces.

The question sounds as though it should have an answer, the way "did you delete the file?" has an answer: yes or no, easily checked. It does not. We sold this agent on what it remembers. It remembers your preferences, the project, the customer's history, the thing that did not work last week. That is the feature. The trouble is that an agent that has served Maria for months has not held her in one place. It has carried her through a working context, the live window of text the model reads while it thinks. From there she has been copied outward.

The obvious stores hold her first: a profile field, a vector index, a graph edge linking one fact to another. The vector index is the searchable store of numerical representations of past text, and it is the one operators most often forget to chase. Then, less visibly, come the derivatives: a background summary written overnight, an audit log, a backup, and possibly a shared memory block that a second agent has been reading too. The deletion request does not land on one object. It lands on a whole system of derivatives, copies and offshoots of the original fact, scattered across stores that nobody drew on a single map.

So the same property we paid for becomes the property we cannot undo. An agent remembers in many places at once, which is exactly why it cannot reliably forget in even one. That gap, between what a system retains and what it can prove it has released, is the subject of this essay. The previous essay followed how routing and effort quietly decide how much authority each step of a workflow receives; this one follows the same authority through time. If an agent can remember a fact and later retrieve, route, approve, deny, summarise or act on it, then memory is not a convenience. It is stored future authority. It is also evidence. The operator has to know what was written, where it lives, who may read it, which actions relied on it, where it travelled, and what operation actually proves it was forgotten.

The narrow claim of this essay is not that agents should remember less. The claim is sharper than that. What an agent remembers is durable state that can change a future action, and a system that cannot govern its own forgetting does not control what it remembers either.



Memory is a governed lifecycle, not a database. Six phases, and forgetting loops back to the start. After Lin et al. ([1], preprint).

A 2026 preprint, research published before formal peer review, gives this obligation a useful vocabulary. It is a survey by Lin and colleagues, and it calls the idea Mnemonic Sovereignty: it treats agent memory not as a store but as a governed lifecycle, in which a fact is written, kept, retrieved, acted on, shared, and eventually forgotten or rolled back ([1], preprint). The lifecycle framing does not stand on the preprint alone. The peer-reviewed literature is already moving on the same axis: Chen and colleagues at NeurIPS 2024 show that an agent's long-term memory and retrieval-augmented knowledge base are themselves a security surface, and that an attacker can compromise downstream behaviour by poisoning the memory store [15]. And the same shape is visible in two regulator instruments below, which both treat memory as a thing with an end date and a deletion obligation rather than as a passive store: one requires that personal data be kept only as long as it is needed, the other gives a person the right to have it deleted [13][14].

One honest disclosure before I lean on it. The six phases I use here are the survey's own six-phase lifecycle, not my invention; I follow that structure and set aside the survey's separate set of nine governance primitives, the cross-cutting operations it identifies that no published architecture yet covers in full. The phrasing is mine, but the six-phase lifecycle and the underlying obligation are the survey's, and it rests on a research scaffold rather than settled industry law, so it should not be treated as a vendor scorecard ([1], preprint).

What the framing is good for is procurement. It gives an operator a precise language for asking whether a system actually has control over its own state. The old memory question was "can

the agent recall this later?" The production question is "can the operator govern the fact after the system has made it useful?"

What the demo calls continuity is stored authority

Memory is sold as continuity, and the pitch is attractive. The demo shows the agent picking up the project where you left it, recalling the customer's history, sparing you the explanation you gave last week.

That continuity is real and valuable. It is also a misdirection. Recall is the visible behaviour, the part the demo shows you. Underneath it sits a state-management system, and that system changes what the agent will be allowed, nudged or expected to do in the future. The remembered preference is not just a nicety. It is an input to the next decision the agent makes.

The context window, the live working memory inside the prompt, is the most visible piece and the least durable one. It clears when the session ends. Persistent memory increasingly lives outside the prompt entirely: structured user records, vector stores, graph memories, session histories, summaries and background consolidation jobs that reorganise everything while the system is idle. A serious production map has at least four layers. There is the working context. There are schema-grounded records, structured facts stored against a defined template. There is archival or graph memory, the cold long-term store. And there is the governance layer, the part that decides how all of the others may be written, read, shared and erased.

The second layer is where a 2026 preprint called xmemory makes a quietly important argument. Unstructured recall, it says, is often the wrong foundation for reliable memory ([2], preprint). Instead of repeatedly rereading old prose and hoping that retrieval plus generation will reconstruct the current state, the system should perform schema-grounded extraction: it writes validated records against a known template ([2], preprint). On the paper's own end-to-end test, that schema-aware approach reported a specific high-nineties F1, a combined measure of how often it both found the right facts and avoided wrong ones, against unstructured-recall baselines reported between 80 and 87 percent ([2], preprint). I have deliberately not quoted the exact headline figure, and the asymmetry is the point: a single-team benchmark number should not become a procurement target, so I band the result it is tempting to repeat while leaving the baseline range as the paper states it. The honest caveat is the one any single preprint carries: this is one team's benchmark, not yet independently reproduced, so the gap matters more than the decimal. The same design choice is visible in peer-reviewed work: Modarressi and colleagues (TMLR 2024) finetune an LLM to use an explicit, structured read-write memory rather than its parameters, and report better factual handling and easier updating once memory becomes a named module instead of a hope [16]. The durable point is not the number. It is the design choice: schema first, recall second.

And that design choice is also a governance choice, which is the part teams miss. A schema says what may be remembered, what must be ignored, what must be updated when it changes, and what must not be inferred at all. It moves the responsibility to the write path, the

moment the fact is committed. Instead of asking a model to expose half a customer's life history at read time and hoping it exposes the right half, it asks the system to commit narrow, named facts under a known contract. You can govern a contract. You cannot govern a hope.

The hardest layer remains the fourth one, governance, and this is where the lifecycle framing earns its place. The same Mnemonic Sovereignty source is also candid that verified, cross-substrate forgetting, proving a fact is gone across every store it reached, is a goal rather than a widely demonstrated production property ([1], preprint). That is the gap the whole industry sits inside. The market has memory stores. It has delete surfaces. It does not yet have a generally accepted proof that a memory was actually erased or invalidated across all the derivatives that made it useful in the first place.

Forgetting is six locks on one door, not a single switch

The six-phase lifecycle is worth taking seriously because each phase asks a different governance question, and a system can pass one while failing the next. A vendor can demonstrate a clean write path and still have no honest answer for sharing. It can show a delete button and have nothing to say about the action that relied on the fact a week earlier. The phases are not a maturity ladder where passing the later ones implies the earlier; they are six separate locks on the same door, and a memory surface is only as governed as its weakest one.

The questions are plain enough to put on a whiteboard. What is allowed to become memory at all, and who authorised it? Where does the memory live, and under what retention rule? Which person or agent may retrieve it, and for what purpose? Which action actually relied on it? Where did it travel beyond the original store? What operation removes, invalidates or suppresses it, and how is that proven?

Those six questions catch six different failures: poisoned context becoming durable state; a fact persisting in a store nobody mapped; one user's memory surfacing in another user's task; a stale fact shaping a real decision nobody can reconstruct; a shared block or vendor subsystem continuing to use a fact after the owner has lost sight of it; and the deletion failure the operator dreads, where the primary record is gone while summaries, graph facts, logs and backups quietly remain.

A memory surface that cannot answer those six questions is not merely incomplete. It is ungoverned, and ungoverned is a precise word, not a rhetorical one. It means there is stored state that can shape a future action and nobody can account for it.

This is not only a privacy problem, and it is worth a second example to show why. Consider an engineering agent that helps a team manage its environments. Early in a project, while the staging database is a throwaway, someone tells the agent, in passing, that it is safe to reset. The agent writes that down: staging is disposable, resetting it is routine. Months later the project has matured. Staging now mirrors production closely, carries real reference data, and is shared by three teams. Nobody updated the memory. So when the agent is asked to clear a corrupted table and reaches for the fastest path it knows, it resets staging on a stale fact, and the blast

radius is no longer small. Notice that nothing here touched a customer's personal data. The fact was operational, the memory was routine team state, and the failure was a write that was once true and was never marked when the world moved on. Memory is stored future authority whether the fact is "Maria prefers not to discuss pricing by email" or "staging is safe to reset". Both shape what the agent will do next.

Notice how different this is from a privacy-policy promise. "We delete personal data on request" is a sentence. The lifecycle asks for the machinery behind the sentence: the stores affected, the deletion operation itself, the path along which it propagates, the evidence receipt it produces, and an honest statement of what residue is left. The sentence is a claim. The lifecycle is the proof obligation.

Find one fact, then chase it across every store it reached

The six phases describe what governed memory must do. The harder task is checking whether a particular system actually does it, and that needs an artefact you can carry into a procurement meeting.

Here is that artefact. Use it the next time a vendor tells you its agent has memory and supports deletion. Pick one concrete customer fact and force the system through it, surface by surface. The fact to test with is the one we started with: Maria's. To make the worked example precise, give her fact a specific shape: "Maria is in contract renegotiation and prefers not to discuss pricing by email."

This is the one place in the essay where a grid earns its keep, because what matters is the cross-product: every memory surface, against the same five columns, so a gap shows up as an empty cell. Two terms in it need glossing first.

A tombstone, in data engineering, is not a deletion of a record but a marker written in its place, a small entry that says "the value that was here is gone". Any system reading later then knows the absence is deliberate rather than an accident. That is why tombstones matter to forgetting: a downstream store that simply stops seeing a fact cannot tell erasure apart from a missed sync, and a tombstone tells it.

"Beyond use" comes from UK data protection practice. When data cannot be erased the instant it is requested, as with a backup that will be overwritten only on its normal cycle, putting it beyond use means locking it so that no one can access or act on it in the meantime, and committing to delete it when the cycle comes round. It is a recognised holding position, not an excuse.

With those in hand, read the rows below as the surfaces Maria's fact could be hiding in. If you read nothing else, read the right-hand column down the page: each entry there is the question that catches the fact still hiding after someone has pressed delete, and a system with no answer to it has an empty cell.

MEMORY SURFACE	WHERE THE FACT MAY LIVE	FORGET OPERATION TO ASK FOR	PROOF ARTEFACT	RESIDUAL QUESTION
Working/context memory	Current prompt, pinned memory block, short-lived session state	Clear, detach, or regenerate the active context without the fact	Session diff, block deletion event, or memory-block mutation log	Can the next turn rehydrate the fact from another store?
Schema-grounded record	User profile, preference field, CRM-linked memory object	Delete the field, or mark it deleted with a tombstone, and revalidate downstream records	Record mutation log with actor, time, and policy version	Was the field inferred elsewhere under another name?
Vector/archive memory	Embedding store, archival passages, old conversation chunks	Delete source chunks, remove embeddings, reindex affected collections	Deletion receipt plus reindex receipt	Do cached retrievals or summaries still contain the fact?
Graph memory	Entity node, edge, temporal fact, episode link	Delete or invalidate the specific edge and any orphaned references	Graph mutation log and validity-window update	Does another episode still support the same edge?
Summaries and learned context	Conversation summaries, background consolidation output, shared memory	Regenerate derivative summaries from cleaned sources	Before/after summary diff with lineage	Which derivatives were discovered by lineage rather than keyword search?
Logs and backups	Audit log, trace store, object backup, vendor support export	Apply retention, suppress future use, or put backup data beyond use where immediate overwrite is not feasible	Backup handling record and access-suppression note	When does the backup age out, and who can still access it?
Shared/vendor memory	Cross-agent shared block, vendor analytics, partner integration	Notify recipients, revoke shared state, request vendor-side deletion	Recipient acknowledgement or processor deletion certificate	Can the organisation responsible for the data prove recipient propagation?

It is worth watching one cell get filled, because the columns read generic until a real fact moves through them. Take the graph-memory row, carrying Maria's renegotiation status. In a graph store that status lives as an edge, (Maria)-[in_state]→(renegotiation), plus an episode link back to the call where she said it. The forget operation is not one act but two: invalidate that edge, then check whether a second episode, the follow-up email thread, independently re-supports it, because if it does, deleting the edge once will not hold. The proof artefact is the graph mutation log showing the edge invalidated at time T with its validity window closed. And the residual question that actually bites is the one in the right-hand column made concrete: the renegotiation fact had also been inferred into a separate "sensitive-account" flag under a different name, so the surface you forgot from was not the only surface that knew.

The table is deliberately plain, and the plainness is the point. Governed forgetting is not a mood or a good intention. It is a set of receipts. Each row of that table is a place Maria's fact could be hiding after someone has pressed delete, and the right-hand column is the question that catches it hiding.

Regulators are already pushing in this direction, even though none of them wrote their guidance with agent memory in mind. Start with the strongest signal, because it is an enforcement finding rather than advice. In a coordinated review published in February 2026, the European Data Protection Board looked across organisations at how erasure requests are actually handled, and found the same failures recurring on the ground: weak procedures, poor staff training, thin information given to individuals, and confusion over exceptions, retention, backups, anonymisation, and gaps in proof and process [5].

That is the operational texture the UK and French regulators have been describing in advance. The UK regulator treats backups and the notification of recipients as part of the right to erasure, including the circumstances where data may have to be put beyond use rather than overwritten the instant a request arrives [3]; the French regulator, working the same ground from a different angle, separates a person's rights over training data from their rights over the model itself, and stresses internal procedures, explanation to the people whose data it is, and the practical limits around retraining and filtering [4]. None of those documents is an agent-memory specification. Together they are more than enough to make the engineering conclusion unavoidable. A memory system needs a forgetting workflow and an evidence workflow, and it needs both as deliberate machinery rather than as an afterthought.

The cross-jurisdictional position is not weaker. Canada's federal private-sector privacy law sets out fair information principles that bind every organisation it covers, and three of them read directly onto the lifecycle: keep personal data only as long as it is needed, be open about how it is handled, and let a person see what is held about them [13]. A Canadian deployer also has to map the same operational requirement onto Quebec's private-sector privacy regime, now fully in force. In the United States, California gives the obligation explicit statutory grip: a consumer can request deletion of the personal information a business has collected, and the business must delete it and direct its service providers and contractors to do the same, subject to the usual statutory exceptions [14]. The statute does not name embeddings, summaries, graph

edges, or backups. It does not have to. Once a consumer's request lands, the obligation is to delete, and the engineering question becomes: across which surfaces, with what proof, and in what time?

So why is a delete button not yet a proof of forgetting?

Vendor documentation is useful here, because it shows both real progress and the precise edge of that progress.

Letta documents persistent memory blocks and archival memory surfaces, including deletion or detach semantics for some objects and passage-level deletion in its interfaces [6][7]. Mem0 documents delete-memory operations, including broad deletion conditions with filters and project-wide wipe semantics [8]. Zep positions user deletion explicitly as a right-to-be-forgotten operation, and describes deleting threads, artefacts and user graph data [9]. Zep's graph documentation is also where the details get sharp: deleting a session and deleting from the graph are not the same operational act, and a graph fact may persist or disappear depending on which episodes and references still support it [10][11]. Cognee's memory documentation distinguishes permanent graph memory from session memory, and shows how information in a session can bridge into a longer-lived store [12].

That is real progress, and I do not want to be sour about it. But it is not a third-party-attested forgetting-propagation audit, and the distinction between those two things is the whole argument of this essay. A delete operation proves that a known object can be removed from a known surface under stated conditions. It does not, by itself, prove that every derivative summary, every graph edge, every embedding, every cache, every backup, every shared block and every downstream recipient has been found and invalidated. That second claim may well be true in a particular deployment. The point is that the operator needs evidence of it, not assurance.

There is an easy way past this, and it is worth naming only to set it aside: that all of this is too much machinery, and that retention settings, scoped storage, delete APIs and a larger context window handle most practical memory problems without it. They handle fragments, and solving fragments is worth doing. But not one of those, on its own, answers the lifecycle question: where did the fact travel, which action used it, and which derivative artefacts were invalidated when it was removed? The perfect should not be the enemy of the useful, and a delete endpoint that clears a known store is genuinely useful. It is just not the same thing as proof.

The objection that should keep an honest engineer awake is harder, because it does not say verified cross-substrate forgetting is merely immature. It says it may be, in the strict sense, impossible. A fact that has been used to write a summary, train a consolidation pass, or shift the weights of a fine-tuned model does not sit in that derivative as a row you can locate and strike out. It has been dissolved into it. You can regenerate the summary from cleaned sources; you cannot always prove the influence of the original is gone, because the influence was never stored as a separate object. Once a fact has propagated into enough lossy, statistic-

al derivatives, "delete it everywhere and prove it" may be asking for a receipt the architecture cannot, even in principle, produce. I take that seriously, and I do not think it can be fully dissolved. What it changes is the standard, not the obligation. If perfect verified erasure is unreachable for a given derivative, the honest response is not to abandon the lifecycle. You name the derivative as one where erasure is unprovable. You constrain it in advance, so that fewer facts reach it and the ones that do are less sensitive. And you fall back to the regulator-recognised position for exactly this case: where overwriting is not feasible, you suppress future use and put the data beyond use, and you say so plainly [3].

The constraint step needs a handle, because "constrain it" is too soft to act on, so here is a concrete admission test: a fact may enter an unprovable derivative, a fine-tune set or a consolidation pass, only if it would survive being made permanent and public. That is the standard you are effectively committing to the moment you admit you can never prove its removal, so it should be the gate at the door. The obligation sharpens from "document the residue" to "gate what is allowed to become residue." An obligation you cannot perfectly meet is still an obligation you must locate, bound and document. The failure is not the residue. The failure is the residue nobody charted.

So the right standard is not perfection. It is specificity. If a vendor cannot delete from backups immediately, the honest move is to say so and show the beyond-use control that limits the backup in the meantime [3]. If erasure at the model level is impractical, or testable only through limited procedures, say so and explain the limit [4]. If a graph memory keeps or removes a fact because of which other episodes still support it, say so and expose the graph operation that decides it [10][11]. The failure, very often, is not that a system cannot forget perfectly. It is pretending that one delete button has answered all six phases of the lifecycle when it has answered one.

The last objection is the one a procurement-savvy vendor raises in the room, and it is aimed straight at the instrument this essay proposes: a Forgetting Receipt is self-attested, so why is a document you generated about your own deletion worth more than the delete log we already distrust? The concession comes first. The Receipt is first-party by default, and the operator who runs it is also the party with the most reason to want it clean. But the load-bearing difference is not who signed it. A delete log asserts that an action was taken. A Receipt asserts an action and records an independent verification probe whose method is disclosed, so a third party can re-run the probe without trusting the issuer. Its value is that it is falsifiable by the auditor, not that it is trustworthy because the vendor signed it. That is the line between an attestation, which you either believe or you do not, and an audit-ready artefact, which you can test.

So before you buy or deploy agent memory, ask three questions, and listen carefully to how far the answers reach. Where does it live? A good answer names every substrate: the working context, the durable record, the vector index, the graph memory, the summaries, the logs, the backups, and any shared or vendor state. How do you propagate deletion? Here a good answer says how a request moves from the primary object out into embeddings, summaries,

graph edges, recipient systems and backup handling. And how do you prove it? The good answer produces artefacts: deletion events, mutation logs, reindex receipts, regenerated-summary diffs, backup suppression records, recipient acknowledgements. A vendor who can only name the substrates has an inventory of its memory and nothing more. Add a credible account of how deletion propagates and it has memory operations. Only when proof comes too, in artefacts an auditor can test, is the vendor starting to have memory governance.

Strip it down and the research, the regulators and the vendor surfaces are all pointing at the same missing layer: governed memory as a lifecycle ([1], preprint; [2], preprint; [15]; [16]), erasure and proof treated as operational obligations rather than vague privacy language [3][4][5], and real delete surfaces that are not yet the same thing as an independently attested propagation proof [6][7][8][9][12]. What I cannot stand behind is narrower, and I will say it plainly: there is no public third-party forgetting-propagation audit, no independent reproduction of xmemory's reported numbers ([2], preprint), no settled answer to how courts and regulators will treat every derivative agent memory where a fact is neither raw personal data nor clearly anonymous, and no guarantee that current vendor deletion semantics will stay stable as product surfaces keep changing.

So here is the test to run this week. Take one fact your agent remembers about a real customer, employee, patient, student or user. Try to fill the forgetting table for that one fact all the way across, every surface, including the proof artefact for each. Then look for the blank cell, and do not file it under unfinished paperwork. Read it for what it is: something the agent remembers and cannot prove it can forget, a piece of stored future authority that nobody can yet account for. The honest reply to "has the agent forgotten her?" is not yes or no; it is the document that fills those cells, and a receipt the whole industry could agree on would do more for governed forgetting than another delete button. It is the named instrument this essay leaves you with, and the box below is its shape.

> **Tool: The Forgetting Receipt.** A Forgetting Receipt is a single artefact, attached to one erasure request, that records what was forgotten, how it was forgotten, and what residue remains. It is what the operator hands back when an auditor, a regulator, or a court asks whether the agent has forgotten Maria.
>
> - **Request id and fact id.** The claim is about one named fact, not a vague gesture at "her data".
> - **Surfaces searched.** Every store the inventory recognises (working context, schema records, vector index, graph memory, summaries, logs, backups, shared and vendor surfaces) is listed, so an auditor can see the inventory rather than trust it.
> - **Operations performed.** The records deleted, the embeddings removed, the summaries regenerated from cleaned sources, the graph edges invalidated, and the backups put beyond use where immediate overwrite was not feasible.
> - **Recipients notified.** Each downstream system or partner the fact reached, with the acknowledgement or processor deletion certificate.
> - **Residuals declared.** The derivatives where erasure is unprovable, bounded and beyond-use rather than pretended away.

- **Verification probe.** The system, after the operation, asked to surface the fact, and the result recorded as passed or failed. One retrieval is not enough, because a paraphrase or a near embedding can surface what a literal lookup misses, so the probe must run on at least two paths: a direct lookup of the stored value, and a semantic or paraphrase query that avoids the original wording (asking "what are this customer's email preferences?" rather than searching the stored string). The paths tried are recorded on the Receipt, and "passed" means the fact failed to surface on every one of them, so a second auditor can re-run the same probe identically.

A Forgetting Receipt with an honest failed probe is worth more than a clean delete log that was never tested, because it tells the operator exactly where the residue still lives.

That receipt is a worse answer than the one Maria deserves, the one we would like to give her. It is also the only one that is true, and the operator who can produce it is governing the memory rather than hoping about it.

Carry This Forward

```
Memory is stored future authority. If an agent can remember a fact and later
retrieve, route, approve, deny, summarise, or act on it, memory is no longer a
convenience feature. It is governance state, and it needs inventory, permission,
mutation, deletion, and proof.

THE ONE MOVE
Take one fact your agent remembers about a real customer, employee, patient,
student, or user, and fill the forgetting table all the way across: every
surface, with the proof artefact for each. The blank cell is the fact the agent
cannot prove it can forget. Do not stop at the visible memory setting; follow
the derivative records, summaries, indexes, backups, and shared agent state,
because that is where future authority hides.

NEXT
Once authority persists, the issue is no longer recall. It is stopping: the rule
by which an agent decides that continuing to act is no longer appropriate.
```

Three regulatory lenses US · EU · UK

Operating questions, not legal advice. The frameworks stay the same; the regulator changes.

US Can you show inventory, deletion semantics, retention rules and proof, mapped to the relevant sectoral and state privacy regimes?

EU Can you handle the data subject rights GDPR and the AI Act expect, including erasure, restriction and explanation of automated decisions?

UK Can you meet ICO expectations on erasure, including backups and downstream propagation, and document it as evidence?

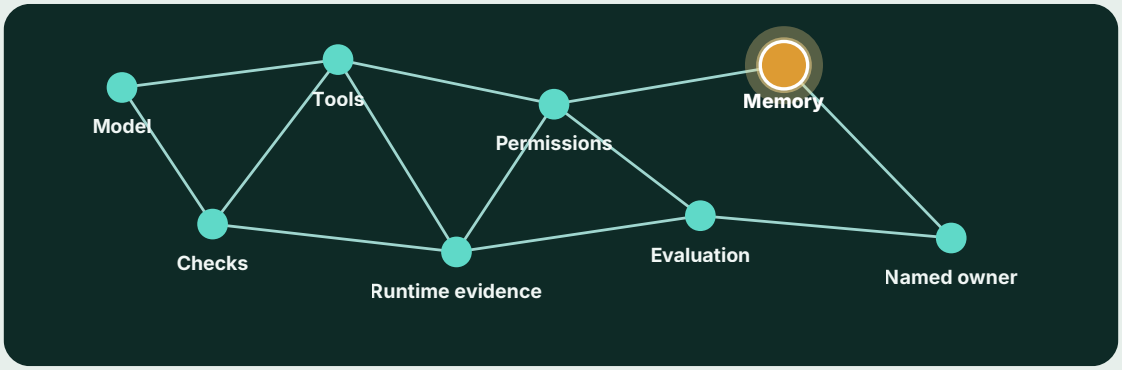
CANADA Canadian deployers should also map the operational requirement to PIPEDA and applicable provincial regimes (for example Quebec Law 25).

THE STACK SO FAR

E13 · Essay 13 of 22 complete · Arc III: Runtime control

The Stack So Far. *Every essay adds one instrument to the operating model. The constellation shows which eight you are building, which are lit by essays you have read, and which is added right here.*

I See the object | II Evidence and authority | III **Runtime control** ESSAY 3 OF 4 | IV Proof and accountability | V Operating model



built in earlier essays · added in this essay · coming in later essays

01 02 03 04 05 | 06 07 08 09 10 | 11 12 13 14 | 15 16 17 18 19 | 20 21 22

You have just added.

The memory lifecycle

You can now map memory as stored future authority.

Next. E14 asks when an agent should stop.

← PREVIOUS
E12 · The Cheapest Token

Essay 13 of 22 complete

NEXT →
E14 · Agents Don't Know When To Stop

References

Reference links for sources cited in this essay.

1. **A Survey on the Security of Long-Term Memory in LLM Agents: Toward Mnemonic Sovereignty**
   Lin et al.
   https://arxiv.org/abs/2604.16548

2. **From Unstructured Recall to Schema-Grounded Memory: Reliable AI Memory via Iterative, Schema-Aware Extraction**
   Petrov et al.
   https://arxiv.org/abs/2604.27906

3. **Right to erasure: backups**
   ICO
   https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/individual-rights/individual-rights/right-to-erasure/

4. **Ensuring and facilitating data-subject rights**
   CNIL
   https://www.cnil.fr/en/ensuring-and-facilitating-exercise-data-subjects-rights

5. **EDPB CEF 2025 Right to Erasure report**
   European Data Protection Board
   https://www.edpb.europa.eu/system/files/2026-02/edpb_cef-report_2025_right-to-erasure_en.pdf

6. **Letta memory blocks documentation**
   Letta
   https://docs.letta.com/guides/agents/memory-blocks/

7. **Letta archives documentation**
   Letta
   https://docs.letta.com/api/python/resources/archives/

8. **Mem0 Delete Memories API**
   Mem0
   https://docs.mem0.ai/api-reference/memory/delete-memories

9. **Zep users and user graphs documentation**
   Zep
   https://help.getzep.com/users-and-user-graphs

10 **Deleting data from the graph**
Zep
https://help.getzep.com/deleting-data-from-the-graph

11 **Zep sessions documentation**
Zep
https://help.getzep.com/v2/sessions

12 **Cognee remember documentation**
Cognee
https://docs.cognee.ai/core-concepts/main-operations/remember

13 **PIPEDA Fair Information Principles (Principles 4.5 retention, 4.8 openness, 4.9 individual access)**
Office of the Privacy Commissioner of Canada
https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/p_principle/

14 **California Civil Code section 1798.105: consumer right to delete personal information**
State of California
https://leginfo.legislature.ca.gov/faces/codes_displaySection.xhtml?lawCode=CIV§ionNum=1798.105.

15 **AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases**
Chen et al.
https://proceedings.neurips.cc/paper_files/paper/2024/file/eb113910e9c3f6242541c1652e30dfd6-Paper-Conference.pdf

16 **MemLLM: Finetuning LLMs to Use An Explicit Read-Write Memory**
Modarressi et al.
https://arxiv.org/abs/2404.11672

About the Author

ARCHITECTING THE AI COWORKER

Dr Peter McCann Strain

Dr Peter McCann Strain is a CTO, founder, and senior AI engineer with a DPhil/PhD in AI from Oxford University. He builds production AI systems and writes about making agentic AI useful, inspectable, governable, and safe enough for real work.

READ THE FULL SERIES

Substack (canonical)	petermccannstrain.substack.com
Medium	@peter.mccann.strain
LinkedIn	peter-strain-dphil-15a607128
Web	petermccannstrain.com
Cadence	New essays twice weekly, 2 June – 21 July 2026