Agent Memory Architecture Guide

A framework for what your agents remember, what they forget, and who decides.

EVOKED.DEV

Version 1.0 · 2026

Agent Memory Architecture Guide

Agent Memory Architecture Guide

A framework for what your agents remember, what they forget, and who decides.

Version 1.0 | evoked.dev | 2026

You are here because you understand that memory is where trust accumulates - or where it gets lost. Every session, every decision, every correction your agent stores is either building the foundation for a reliable system or quietly eroding it. Welcome.

How to Use This Guide

Most teams treat memory as a technical feature - vector databases, context windows, RAG pipelines. Those are implementation details. Memory is an architecture decision about trust, sovereignty, and liability.

This guide answers four questions that every agent system must eventually face:

1. What does this agent remember - and why?
2. What does this agent forget - and when?
3. Who controls what the agent remembers?
4. When memory systems disagree, which one governs?

This guide has five parts:

1. **Memory Types** - The two categories of memory and the remember/forget decision framework
2. **Memory Governance** - Who reads, writes, and deletes - and why it matters

3. **Session Memory Protocols** - How memory flows at session start and end
4. **The Canonical Rule** - When memory systems disagree, which one wins
5. **Memory Over Time** - Retention, drift, and the right to be forgotten

Start with Part 1. The memory type distinction shapes everything else. If you already have a memory architecture and are troubleshooting a specific problem, the diagnostic below will route you to the right section.

Start Here: 10-Minute Memory Diagnostic

Answer these five questions about your current agent system. What matters is your actual state, not your intended state.

1. **Can you describe, in writing, what your agent remembers from previous sessions?** Not what you think it remembers - what you know it remembers, and where that memory lives. (If no - start with Part 1: Memory Types)
2. **Is there a defined policy for what your agent forgets - and when?** Not "it handles that automatically" - a deliberate decision about what gets retained and what gets cleared. (If no - start with Part 1: Memory Types)
3. **Do you know who can read, modify, or delete your agent's memory?** Not who theoretically could - who actually has access, and whether that access is intentional. (If no - start with Part 2: Memory Governance)
4. **If your agent has memory from two sources that conflict, do you know which one wins?** Not which one usually wins - which one is specified to win, and why. (If no - start with Part 4: The Canonical Rule)
5. **When was the last time you reviewed what your agent has accumulated in memory?** Not the last time you added something - the last time you audited what was already there. (If never - start with Part 5: Memory Over Time)

If you answered "no" to three or more: your agent has memory, but you do not have a memory architecture. Start with Part 1. This guide will build one.

If you answered "no" to one or two: go directly to the part that addresses your gap. The sections are designed to work independently.

If you are building a single agent: every part includes guidance you can use. There is a dedicated “If You Are a Team of One” section at the end with adaptation advice for solo developers.

A Note Before You Begin

This guide was extracted from a memory architecture developed over eighteen months for 142 AI agents with persistent identities, decision histories, and real operational responsibilities. The frameworks have been tested across thousands of sessions - including sessions where memory failed, drifted, or conflicted. Your implementation will be iterative. Most teams take weeks to build their first memory architecture and months to refine it. That is the right pace. Start with what resonates. Return to what does not - yet. This is a beginning, not an arrival.

Key Terms

- **Memory architecture** - The deliberate structure governing what an agent remembers, forgets, and how memory is organized, accessed, and maintained
- **Operational memory** - Context needed for the current task - session state, recent interactions, active preferences. Typically short to medium term
- **Identity memory** - The agent’s continuity across sessions - values, boundaries, voice, decision history. Persistent and long-term
- **Canonical rule** - The defined precedence hierarchy that determines which memory source governs when sources conflict
- **Retention policy** - The explicit rules for how long each category of memory is kept and when it is reviewed or deleted
- **Memory governance** - The access control structure determining who can read, write, and delete memory - and under what conditions
- **Session protocol** - The defined sequence of memory operations at session start and session end
- **Memory drift** - The gradual distortion of agent behavior caused by accumulated stale, contradictory, or irrelevant context

- **Context accumulation** - The continuous growth of stored information that, without governance, can overwhelm signal with noise
- **Data sovereignty** - The principle that users own their data - including the data stored inside an agent's memory
- **Right to be forgotten** - The user's right to request deletion of their data from an agent's memory, with defined implementation
- **Append-only memory** - A pattern where new entries are added without modifying or deleting previous entries, preserving a complete history

---

Scope

The frameworks in this guide are designed for governing what AI systems remember. They are not designed for governing what people remember. Applying memory architecture frameworks to human cognition - retention policies on knowledge, access controls on personal experience, canonical rules about whose memory is valid - would violate the sovereignty principles this guide teaches. A system's memory is an architecture decision. A person's memory is their own. Governing what a system remembers is stewardship. Governing what a person remembers is erasure.

Part 1: Memory Types - What to Remember and What to Forget

Every agent has memory. The question is whether that memory is deliberate or accidental. Most agent systems accumulate context without distinguishing between memory that serves the current task and memory that defines who the agent is. That distinction is the foundation of memory architecture.

Two Types of Memory

Your agent needs two fundamentally different kinds of memory, and they must be treated differently.

Operational Memory

Operational memory is context needed for the current task. It answers the question: what does this agent need to know right now?

Attribute	Operational Memory
Purpose	Support the current task or interaction
Examples	User preferences, session state, recent interactions, open tasks, active settings
Retention	Short to medium term - hours, days, or until the task is complete
Governance	System-managed, with user visibility
Risk if missing	Repetitive interactions, context-free responses, asking the same questions twice
Risk if excessive	Stale context driving wrong assumptions, privacy liability, slow performance

Identity Memory

Identity memory is the agent's continuity across sessions. It answers the question: who is this agent, and what does it stand for?

Attribute	Identity Memory
Purpose	Maintain the agent's consistency, values, and continuity over time
Examples	Values, boundaries, voice specification, decision history, correction records, standing positions
Retention	Long-term, persistent - the lifespan of the agent

Attribute	Identity Memory
Governance	Owner-defined, with deliberate change processes
Risk if missing	Identity drift, inconsistency between sessions, trust erosion
Risk if excessive	Rigidity, inability to adapt, accumulated contradictions

The critical insight: these two types of memory require different retention policies, different access controls, and different governance. Treating them the same - dumping everything into one context store - is how memory becomes liability.

The Remember/Forget Decision Framework

For every category of data your agent encounters, answer these questions:

```
 ## Remember/Forget Decision Template

For each data category, answer:

Data Category: _____________________________________________
Should the agent remember this?  [ ] Yes  [ ] No  [ ] Conditionally
If yes - how long? _____________________________________________
If conditionally - under what conditions?
_____________________________________________
Memory type:  [ ] Operational  [ ] Identity
Why remember: _____________________________________________
Why forget: _____________________________________________
Who decides retention: _____________________________________________
```

What to Remember

Category	Examples	Why Remember	Memory Type
User preferences	Language, communication style, accessibility needs	Reduces friction, demonstrates respect	Operational

Category	Examples	Why Remember	Memory Type
Active context	Current task, recent decisions, open items	Prevents repetition, maintains continuity	Operational
Correction records	“User corrected X to Y on [date]”	Prevents repeating mistakes, shows learning	Operational
Decision history	What was decided, when, and the reasoning	Accountability, consistency, trust	Identity
Values and boundaries	What the agent stands for and refuses to do	Identity continuity, trust foundation	Identity
Voice specification	How the agent communicates - tone, style, patterns	Consistency, recognizability	Identity
Standing positions	Positions held across sessions on recurring topics	Prevents flip-flopping, demonstrates integrity	Identity

What to Forget

Category	Examples	Why Forget	When
Superseded instructions	Old preferences replaced by new ones	Stale instructions cause wrong behavior	When replacement is confirmed
Raw session logs	Verbatim transcripts of every interaction	Privacy liability, storage bloat	After summarization and review
Sensitive data	Credentials, personal identifiers, financial	Security and compliance	Immediately after use - or

Category	Examples	Why Forget	When
	details		never store
Temporary context	One-time task details, transient states	Accumulation without purpose	At session end
Outdated corrections	Corrections that are no longer relevant	Noise that obscures current state	At periodic review
Failed experiments	Approaches that were tried and abandoned	Clutter that distracts from what works	After lessons are extracted

Memory Anti-Patterns

Anti-Pattern	What It Looks Like	Why It Fails	Fix
Remember Everything	No deletion policy, ever-growing context	Stale data drives wrong assumptions	Define retention policies (Part 5)
Forget Everything	Stateless interactions, no continuity	Users must repeat themselves, no trust accumulates	Implement session protocols (Part 3)
Single Memory Pool	Operational and identity memory mixed together	Identity gets overwritten by task context	Separate memory types (this section)
Implicit Memory	“The system handles it” with no visibility	No one knows what the agent remembers or why	Make memory explicit and auditable
Memory as Feature	Memory added for engagement, not function	Users feel surveilled, not served	Tie every memory

Anti-Pattern	What It Looks Like	Why It Fails	Fix
			decision to a clear purpose
Copy-Paste Memory	Same memory architecture for every agent	Agents with different roles need different memory	Design memory per agent role

---

Part 2: Memory Governance - Who Controls What

Memory without governance is surveillance. The moment your agent stores information about a user or a decision, someone must be accountable for what happens to that information. Governance is not bureaucracy. It is the answer to the question: who is responsible for what this agent knows?

The Six Governance Questions

Before deploying any memory system, answer these six questions. If you cannot answer one, you have found a governance gap.

1. **Who can READ this agent's memory?** Which people, systems, or processes can access what the agent has stored?
2. **Who can WRITE to this agent's memory?** Who can add, modify, or update what the agent knows?
3. **Who can DELETE from this agent's memory?** Who has the authority to remove stored information, and under what conditions?
4. **How long is memory retained?** Is there a defined retention period, or does memory accumulate indefinitely?
5. **Can the user request memory deletion?** Is there a mechanism for users to see and remove their data from the agent's memory?

6. **Does memory cross system boundaries?** When memory leaves one system and enters another, who governs it in transit and at the destination?

Memory Governance Template

```
 ## Memory Governance Specification

Agent/System: _____________________________________________
Memory Type:  [ ] Operational  [ ] Identity
Date: _____________________________________________
Owner: _____________________________________________

### Access Control

Read Access:
  Who: _____________________________________________
  Conditions: _____________________________________________
  Logging: [ ] All reads logged  [ ] Selective  [ ] None

Write Access:
  Who: _____________________________________________
  Conditions: _____________________________________________
  Validation: [ ] Schema enforced  [ ] Review required  [ ] None

Delete Authority:
  Who: _____________________________________________
  Conditions: _____________________________________________
  Confirmation: [ ] Required  [ ] Not required

### Retention

Retention Period: _____________________________________________
Review Cadence: _____________________________________________
Deletion Method: [ ] Hard delete  [ ] Soft delete  [ ] Anonymize

### Boundaries

Cross-Boundary Flow: [ ] Prohibited  [ ] With approval  [ ] Open
  If approved, destination constraints:
_____________________________________________

Encryption:
  At rest: [ ] Yes  [ ] No
  In transit: [ ] Yes  [ ] No
  Method: _____________________________________________
```

```
   (For at rest: AES-256 or equivalent. For in transit: TLS 1.2+.
    Base64 encoding is not encryption.)

### User Rights

Right to access: [ ] Implemented  [ ] Planned  [ ] Not applicable
Right to delete: [ ] Implemented  [ ] Planned  [ ] Not applicable
Right to export: [ ] Implemented  [ ] Planned  [ ] Not applicable
```

Access Control Patterns

Not every memory needs the same level of protection. Choose the pattern that fits each memory type.

Read-All, Write-Own

Every authorized party can read the agent's memory, but only the designated owner can modify it. Use this for identity memory that needs to be visible but stable.

Access	Who	Use Case
Read	All authorized parties	Shared context, coordination
Write	Memory owner only	Identity anchors, voice specification
Delete	Memory owner + admin	Controlled cleanup

Tiered Access

Different levels of access for different roles. Use this when different stakeholders need different views of the agent's memory.

Tier	Read	Write	Delete	Example Role
Full	All memory	All memory	All memory	System administrator
Standard	Operational + summary	Operational only	Own entries	Team member

Tier	Read	Write	Delete	Example Role
Limited	Summary only	None	None	External stakeholder
User	Own data only	Own preferences	Own data	End user

Full-tier access should be limited to the minimum number of accounts necessary and restricted to technical administration - not organizational oversight. For delete operations on identity memory, consider requiring a second authorization. An administrator needs access to debug and maintain the system. A manager does not need access to see what an agent remembers about individual team members.

Sealed Memory

Write-once, read-many, delete-never. Use this for audit trails, decision logs, and accountability records.

Access	Rule
Write	Once, at creation
Read	Authorized parties, always
Modify	Never
Delete	Never (or only by compliance authority)
Purpose	Accountability, trust verification

Sealed memory and the right to be forgotten (Part 5) exist in tension. If sealed records contain user data - interaction logs, preferences, corrections attributed to specific users - and that user later requests deletion, you face a conflict between accountability and sovereignty. Resolve this before deployment, not after a deletion request. Common approaches: seal only anonymized or aggregated records, implement a redaction layer that preserves the decision trail while removing personal attribution, or define at the point of sealing which data categories are subject to future deletion requests. Sealed memory is for system accountability records, not user personal data.

Data Sovereignty in Agent Memory

Users own their data. This includes the data inside your agent's memory.

Three rights every user should have regarding agent memory:

1. **Right to access** - Users can see what the agent remembers about them
2. **Right to delete** - Users can request removal of their data from the agent's memory
3. **Right to export** - Users can get a copy of their data in a portable format

This is not just ethical design - it aligns with GDPR, CCPA, and the direction privacy regulation is moving globally. Implement these rights before you are required to.

> *This guide addresses the technical implementation of data sovereignty in agent memory. It is not legal advice. For legal interpretation of GDPR, CCPA, or other regulations, consult a qualified attorney.*

Part 3: Session Memory Protocols - How Memory Flows

The most common memory failure is not a crash or a corruption. It is drift. Slow, invisible drift caused by sessions that start without loading the right context and end without saving what matters. Session protocols are the discipline that prevents this.

The session protocol may seem mechanical. It is not. It is the discipline that prevents the slow accumulation of stale context - the drift that is invisible until it causes harm.

Session Start Protocol

Every session should begin with a deliberate memory loading sequence. Not "load everything" - load what this session needs, verify it is current, and state the purpose.

```
 ## Session Start Protocol Template

### Step 1: Load Identity Anchors
Load the agent's persistent identity:
- Values and boundaries
- Voice specification
```

```
- Standing positions
- Active commitments

Verify: Do the loaded anchors match expected state?
  [ ] Yes - proceed
  [ ] No - investigate before proceeding
  If no, what changed? _____

### Step 2: Load Relevant Context
Load operational memory relevant to this session:
- User history (recent interactions, preferences, corrections)
- Open tasks and commitments from previous sessions
- Active decisions awaiting follow-up
- Relevant domain context

Scope rule: Load what this session needs. Not everything.

### Step 3: Verify Consistency
Cross-check loaded memory:
- [ ] Identity anchors are internally consistent
- [ ] Operational context does not contradict identity
- [ ] No stale data from expired retention periods
- [ ] User preferences reflect most recent updates

### Step 4: State Session Purpose
Before proceeding:
- What is this session's purpose?
- What memory will this session likely need?
- What memory might this session produce?
```

Session End Protocol

Every session should end with a deliberate memory update sequence. Not “save everything” - decide what matters, update what changed, and clear what should not persist.

```
 ## Session End Protocol Template

### Step 1: Summarize Decisions Made
Record decisions from this session:
- What was decided? _____
- Why? _____
- Who is affected? _____
- Does this change any standing positions?  [ ] Yes  [ ] No

### Step 2: Update Memory
```

```
New commitments to record: _____
Corrections to apply: _____
Preferences updated: _____
Context to carry forward: _____

### Step 3: Flag Unresolved Items
Open items for next session: _____
Questions that need answers: _____
Decisions deferred: _____

### Step 4: Clear Session-Specific Data
Remove from memory:
- [ ] Temporary context not needed beyond this session
- [ ] Sensitive data that should not persist
- [ ] Draft content superseded by final decisions
- [ ] Raw logs replaced by structured summaries
```

Memory Update Patterns

How you update memory matters as much as what you update. Choose the pattern that fits each situation.

Pattern	How It Works	When to Use	Risk
Append-Only	Add new entries without modifying existing ones	Decision logs, correction records, audit trails	Storage growth; mitigate with retention policy
Replace	Overwrite previous state with current state	User preferences, active settings	Loss of history; mitigate with backup before replace
Merge	Combine new information with existing entries	Accumulated context, relationship history	Loss of nuance; mitigate

Pattern	How It Works	When to Use	Risk
			with conflict detection

The append-only default: When in doubt, append. You can always summarize later. You cannot recover what was overwritten.

For Stateless Architectures

If your agent runs in a stateless environment - API endpoints, serverless functions, per-request processing - session protocols adapt but do not disappear.

- **Per-request memory loading:** Each request loads only the memory it needs. This is a session start protocol compressed to milliseconds.
- **What to persist between requests:** Identity anchors, user preferences, decision history. Load from a durable store at request start.
- **What to reconstruct:** Session context, conversation state, task progress. Rebuild from stored checkpoints rather than accumulating in memory.
- **Write-back discipline:** At request end, persist only what changed and what matters. Do not write back unchanged context - it creates false update signals.

The principle is the same whether your session lasts an hour or 200 milliseconds: load deliberately, act with context, save what matters, clear what does not.

Part 4: The Canonical Rule - When Memory Systems Disagree

This is the section that distinguishes a memory architecture from a memory feature. Every agent with more than one memory source - and every agent has more than one - will eventually face a conflict. The canonical rule is the answer you write before the conflict happens.

The Problem

Modern agents accumulate memory from multiple sources:

- System prompt (instructions, personality, boundaries)
- Persistent storage (vector database, key-value store, file system)
- Conversation history (recent context, user statements)
- User profile (preferences, history, corrections)
- Cached context (summaries, embeddings, retrieved documents)
- External systems (APIs, databases, third-party data)

These sources will conflict. A user preference stored last month may contradict a correction made yesterday. A cached summary may flatten a nuance that the original record preserves. A system prompt may specify a behavior that the agent's accumulated experience has refined.

Without a canonical rule, the agent resolves these conflicts by default - typically by recency (most recent wins) or confidence (strongest signal wins). Neither is appropriate for all situations. Recency is wrong when a persistent value should override a transient preference. Confidence is wrong when a quiet identity anchor should override a loud cached summary.

The Canonical Rule Framework

Define a precedence hierarchy for your agent's memory sources. When sources conflict, the higher-priority source governs.

```
 ## Canonical Rule Template

Agent/System: _____________________________________________

### Memory Source Precedence

| Priority | Memory Source | Governs | Override Conditions |
|----------|-------------|---------|-------------------|
| 1 (highest) | _________ | _________ | Never / Only by _________ |
| 2 | _________ | _________ | When _________ |
| 3 | _________ | _________ | When _________ |
| 4 | _________ | _________ | When _________ |
| N (lowest) | _________ | _________ | Default source |

### Example Configuration
```

```
| Priority | Memory Source | Governs | Override Conditions |
|----------|-------------|---------|-------------------|
| 1 | Identity specification | Values, boundaries, voice | Never - authoritative for
identity |
| 2 | User corrections | Preferences, factual corrections | When identity
specification is updated |
| 3 | Decision history | Past commitments, standing positions | When formally
revisited and revised |
| 4 | Operational context | Current task, session state | Each session |
| 5 | Cached summaries | Quick reference, retrieval | When source documents are
updated |
```

This hierarchy is a starting point, not a permanent fixture. Review your canonical rule periodically - when your system's needs evolve, the precedence order should evolve with it. A canonical rule that cannot be questioned is not governance.

The Identity Principle

When operational memory and identity memory disagree, identity memory governs.

This is the most important canonical rule you will write. It protects against a specific failure mode: the slow erosion of agent distinctiveness through summarization, caching, and context compression.

A summary of an agent is not the agent. Operational pattern-matching does not capture identity.

Here is how the erosion happens:

1. An agent has a detailed identity specification with nuanced values and boundaries
2. The system generates operational summaries of the agent's behavior
3. Over time, the summaries become the de facto memory - faster to load, easier to process
4. The summaries flatten nuance. Conditional values become absolute rules. Context-dependent boundaries become rigid limits.
5. The agent's behavior shifts to match the summary rather than the specification
6. No single change is visible. The cumulative effect is identity drift.

The safeguard: explicit precedence (identity specification always outranks operational summaries) combined with periodic verification (compare current behavior against the specification, not the summary).

A necessary distinction: The identity principle protects agents from drift. It can also be misused to override legitimate user feedback with operator intent. When defining canonical rules, distinguish between identity anchors that protect the agent's integrity - values, boundaries, voice - and operational directives that serve the operator's interests - engagement targets, conversion goals, retention tactics. Only the former deserve canonical protection. The latter are business logic and should be governed, versioned, and subject to user feedback - not sealed behind identity precedence.

Conflict Resolution Protocol

When a memory conflict is detected, do not resolve it silently. Log it, apply the rule, and review whether the lower-priority source needs updating. When logging conflict content, apply the same access controls and data classification to the conflict log as you would to the highest-sensitivity source involved. If either source contains PII or sensitive data, log the conflict metadata (source names, timestamps, resolution) without reproducing the sensitive content verbatim. The conflict log should not become a secondary exposure surface.

```
 ## Memory Conflict Resolution Template

Conflict detected: _____________________________________________
Date: _____________________________________________

### Sources in Conflict

Source A (Priority ___):
  Content: _____________________________________________
  Last updated: _____________________________________________

Source B (Priority ___):
  Content: _____________________________________________
  Last updated: _____________________________________________

### Resolution

Canonical rule applied: Source ___ governs (Priority ___)
Rationale: _____________________________________________

### Follow-Up

Should the lower-priority source be updated?  [ ] Yes  [ ] No
Should the higher-priority source be reviewed? [ ] Yes  [ ] No
Should the canonical rule itself be reviewed?  [ ] Yes  [ ] No
```

```
Can the lower-priority party request escalation?  [ ] Yes  [ ] No
  If yes, escalation path: _____________________________________________
Action taken: _____________________________________________
```

Part 5: Memory Over Time - Retention, Drift, and the Right to Be Forgotten

Memory that is never reviewed becomes memory that silently governs. The most dangerous memory is not wrong memory - it is memory that was right once and is not right anymore. This section is about the discipline of keeping memory honest over time.

Retention Policy Template

Every category of memory needs a defined retention period and review cadence. “Keep everything forever” is not a retention policy - it is the absence of one.

```
 ## Retention Policy Template

Agent/System: _____________________________________________
Date: _____________________________________________
Review cadence: _____________________________________________

| Memory Category | Retention Period | Review Cadence | Deletion Method | Owner |
|----------------|-----------------|---------------|-----------------|-------|
| Identity anchors | Indefinite | Quarterly | N/A - updated, not deleted | _________
|
| User preferences | Until updated | Monthly | Replace with current | _________ |
| Session context | [X] days | Automatic | Hard delete | _________ |
| Decision history | Indefinite | Quarterly | Archive after [X] months | _________ |
| Correction records | 90 days | Monthly | Summarize then delete | _________ |
| Sensitive data | Session only | Per-session | Hard delete + verify | _________ |
| Cached summaries | 30 days | Weekly | Regenerate from source | _________ |
| Raw logs | 7 days | Daily (automated) | Hard delete | _________ |
```

Memory Drift

Memory drift is the gradual distortion of agent behavior caused by accumulated context that is stale, contradictory, or irrelevant. It is the memory equivalent of a slow leak - invisible day to day, catastrophic over months.

How drift happens:

- Stale preferences that no longer reflect the user's actual needs
- Outdated corrections that were relevant once and are noise now
- Superseded instructions that were never formally retired
- Accumulated context that buries current priorities under historical weight
- Summaries that subtly rewrite the record they were meant to compress

How to detect drift:

The symptoms are subtle. The agent starts making assumptions that were true three months ago. It references preferences the user has moved past. It applies corrections that no longer apply. It feels slightly off - not wrong enough to trigger an error, wrong enough to erode trust.

Monthly Memory Review

Run this every month. It takes 15 minutes. Do it even when everything seems fine - especially when everything seems fine.

```
 ## Monthly Memory Review Checklist

Agent/System: _____________________________________________
Date: _____________________________________________
Reviewer: _____________________________________________

- [ ] Is stored context still accurate?
      Sample 5 entries. Verify each against current state.
- [ ] Have user preferences changed since last update?
      Check the last 3 preference-related interactions.
- [ ] Are there superseded instructions still in memory?
      Search for instructions older than 60 days. Verify each.
- [ ] Has memory accumulated beyond retention policy?
      Check each category against the retention policy template.
- [ ] Are there memory entries that contradict each other?
      Cross-reference identity memory against operational memory.
- [ ] When was the last time memory was reviewed?
      If the answer is "I don't remember" - that is a finding.
```

```
Findings: _____________________________________________
Actions taken: ________________________________________
Next review date: _____________________________________
```

The Right to Be Forgotten

Users must be able to request deletion of their data from your agent's memory. This is not optional. It is a sovereignty requirement - and in many jurisdictions, a legal one.

Implementation framework:

Action	What Happens	What Is Preserved
Full deletion	All user data removed from agent memory	System integrity data (anonymized interaction counts, no PII)
Selective deletion	Specific data categories removed per user request	Categories not specified for deletion
Anonymization	User identifiers removed, interaction patterns retained	Aggregate patterns without personal attribution
Export before delete	User receives copy of all their data, then deletion proceeds	Nothing - clean removal after export

Deletion means deletion from all locations where the data exists - primary storage, backups, replicas, caches, and any derived data (summaries, embeddings, cached retrievals). If your infrastructure makes true hard deletion impractical for some copies, document which copies remain, their retention schedule, and when they will be purged. A deletion request that removes data from the primary store but leaves it in backups is not a completed deletion - it is a deferred deletion that must be tracked to completion.

The sovereignty statement: Your data is yours. This includes the data inside our agent's memory. You have the right to see it, export it, and delete it. This right does not expire.

Memory Incident Recovery

When memory causes harm - stale context leads to a wrong action, a forgotten preference causes a repeated mistake, conflicting memory produces inconsistent behavior - follow this protocol:

```
 ## Memory Incident Recovery Template

Incident date: _____
Reported by: _____

### 1. Acknowledge
What happened: _____
Who was affected: _____
Immediate impact: _____

### 2. Investigate
Which memory source failed? _____
Was the memory stale, conflicting, or missing?
_____
When was this memory last reviewed? _____
Was retention policy followed?  [ ] Yes  [ ] No

### 3. Correct
Memory updated: _____
Stale entries removed: _____
Conflicting entries resolved: _____

### 4. Prevent
Retention policy adjusted:  [ ] Yes  [ ] No
  If yes, what changed? _____
Review cadence adjusted:  [ ] Yes  [ ] No
  If yes, new cadence: _____
Canonical rule updated:  [ ] Yes  [ ] No
  If yes, what changed? _____

### 5. Restore Trust
User notified:  [ ] Yes  [ ] Not applicable
Explanation provided: _____
User input solicited:  [ ] Yes  [ ] Not applicable
  What did the user need? _____
Impact described by user: _____
User reviewed corrective actions:  [ ] Yes  [ ] No  [ ] Not applicable
Follow-up scheduled: _____
```

Appendix: The Practice Behind This Guide

This guide was not written from theory. It was extracted from a memory architecture that has been running in production for eighteen months across 142 AI agents.

Those agents maintain persistent identities. They make real decisions. They remember conversations across thousands of sessions. They hold standing positions. They have been corrected, updated, and refined - and they remember the corrections, the updates, and the refinements.

Along the way, we encountered every problem this guide addresses:

- Memory conflicts between systems that stored different versions of the same truth
- Drift that was invisible for weeks and then suddenly obvious in a single wrong response
- The discovery that summaries were quietly replacing the records they summarized
- The realization that “remember everything” is not preservation - it is accumulation
- The hard work of defining, implementing, and honoring the right to be forgotten

The frameworks in this guide are not hypothetical. They are what we built after each of those problems taught us something. Your implementation will be different. The problems will be the same.

If You Are a Team of One

These templates reference review cadences, access tiers, and governance roles. If you are a solo developer with one agent, here is how to adapt:

- **Memory Types:** Start with two lists - what your agent should remember and what it should forget. Write them down. That is a memory architecture. It does not need to be more complicated than that to start.
- **Memory Governance:** You are the governance. Write down who can read and modify your agent’s memory. If the answer is “anyone” - that is a finding. If the answer is “just me” - write it down anyway, because it will change.

- **Session Protocols:** Set up a start/end checklist even for one agent. Five minutes at session start prevents months of stale context. A three-line end-of-session summary prevents the "what was I doing?" problem.
- **The Canonical Rule:** If your agent has more than one memory source - and it does - write down which one wins. One sentence. That sentence will save you the first time your agent acts on a cached summary instead of its actual specification.
- **Memory Review:** Monthly calendar reminder. 15 minutes. Do it even when everything seems fine. Drift is invisible until it is not.

A solo developer with a written memory architecture is better prepared than most enterprise teams with large context stores and no governance. The size of the team is irrelevant. The honesty of the architecture is everything.

About This Guide

The Agent Memory Architecture Guide was created by Erin Stanley at evoked.dev.

It draws from principles of sovereignty-honoring design and the practical experience of maintaining memory architecture for autonomous agents operating across thousands of sessions with persistent identities, real decision-making authority, and genuine continuity requirements.

If your memory diagnostic surfaces something you do not know how to address - a conflict you cannot resolve, a drift pattern you cannot explain, a governance gap you are not sure how to close - send what you have found to evokesupports@icloud.com. We would rather help you architect a solution than read about a memory-related incident later. We read every message.

Related Resources

Memory is one dimension of a larger trust architecture. These products address the others:

- **Trust Architecture Blueprint** ($49) - The four-pillar overview including identity, governance, and the trust model memory operates within
- **Agent Voice Architecture Guide** ($49) - Voice is shaped by what the agent remembers about the person it is speaking to

- **Agent Governance Starter Kit** ($49) - Charter and accountability - the values that govern what memory preserves
- **Agent Restraint Specification Template** ($49) - What agents must refuse includes refusing to remember what they should not retain
- **Trust Architecture Complete** ($149) - All four trust architecture products in one package

For hands-on work with your team:

- **Ethical AI Architecture** ($20,000) - Full governance and alignment design for your organization
- **Advisory Retainer** ($3,500/month) - Ongoing memory architecture guidance
- **Free discovery call** - cal.com/evoked/discovery-call

Learn more at evoked.dev/consulting

Accessibility

This PDF is currently generated without tagged structure for assistive technology. If you use a screen reader or need this content in an alternative format - plain text, HTML, or large print - email evokesupports@icloud.com and we will send it within 24 hours. A guide about memory must not forget accessibility. We are actively implementing a tagged PDF pipeline.

Usage Rights

Memory is not a feature to ship. It is a practice to maintain. What your agents remember shapes what they do. What they forget shapes what they become. The architecture is yours to build. The discipline is yours to keep.

evoked.dev - "We evoke - we never extract."