LinkTec LABS

RESEARCH NOTE N°02 · APPLIED AI & DATA

Agent Memory: remembering without **blowing the budget**

Persistent state for long-horizon agents.

ABSTRACT

A bigger context window solves *capacity* within a session; it does nothing for *continuity* across them. Yet the agents now reaching production run for hundreds of steps and return day after day — and the naive answer, keep everything in the window, is both ruinously expensive and quietly inaccurate. This note treats memory as a first-class engineering discipline: a typed system of working, episodic, semantic and procedural stores, with an explicit lifecycle — encode, consolidate, retrieve, forget — governed by policy rather than left to grow. We survey the architectures that define the state of the art (OS-style memory hierarchies, extraction-based external stores, file-based memory and context editing), the cost mechanics that make them viable, the benchmarks that keep them honest, and the failure modes — unbounded growth, stale facts, privacy — that decide whether they are deployable. We close with the governed, sovereignty-ready memory architecture LinkTec Labs builds.

01 — CAPACITY IS NOT CONTINUITY

Why memory, not just a bigger window

The most common mistake in agent design is to treat memory as a context-length problem. It is not. A long context window solves *capacity* — how much an agent can read at once. Memory solves *continuity* — what an agent carries from one step, one session, one week to the next. The two are orthogonal, and conflating them produces agents that are simultaneously expensive and forgetful.

The symptom is familiar to anyone who has shipped one. An agent performs beautifully for the first dozen turns, then degrades: by the fortieth step the window is choked with tool transcripts and earlier decisions, and the model loses the thread[8]. The reflex — enlarge the window and pour everything back in — fails twice over. It is expensive: attention cost grows super-linearly with sequence length, so every retained token is paid on every subsequent turn. And it is inaccurate: as we argued in Research Note N°01, models do not use long contexts uniformly, so a fuller window is often a *less* reliable one. Continuity bought by brute capacity is continuity that erodes exactly when it is needed most.

Memory is the alternative: a deliberate system that decides what to keep, in what form, where to put it, and when to let it go — so that the working window stays small and sharp while the agent's *knowledge* of its task, its user and its history persists outside it. The framing that has stabilised across the field is exactly this: long context addresses capacity; memory addresses continuity[6].

KEY TERMS

A shared vocabulary for the rest of this note. Specialists may skip ahead.

Context window	The tokens a model can attend to in a single forward pass — its working surface, not its memory.
Working memory	The small, in-context state for the current task; analogous to RAM. Volatile, fast, scarce.
Long-term memory	Information persisted *outside* the window and retrieved on demand; survives across sessions.
Episodic memory	Records of specific past events — what happened, when, in what order (the agent's log).
Semantic memory	Stable, de-contextualised facts and preferences distilled from many episodes.
Procedural memory	Learned skills and workflows — how to do things, including tool-use patterns.
Consolidation	Compressing raw episodes into durable, higher-level memory (e.g. reflection, summarisation).
Compaction	Summarising in-window history into a compact state object to reclaim tokens.

Retrieval (memory)	Selecting and re-injecting the few relevant memories into the window at decision time.
Forgetting / decay	Deliberately removing or down-weighting stale or low-value memories to bound growth.

02 — A WORKING TAXONOMY

Four memories, one agent

Useful memory systems are not monolithic. The canonical academic frame, CoALA, adapts cognitive science for language agents and distinguishes *working* memory from three long-term stores — *episodic*, *semantic* and *procedural*[2]. The distinction is not pedantic; each store has a different write policy, retrieval pattern, and decay rate, and conflating them is a primary cause of bloated, unreliable memory.

Episodic memory is the agent's log — "on Tuesday the user rejected vendor A on price." Semantic memory is what that log *distils into* — "the user is price-sensitive on infrastructure." Procedural memory is the skill that crystallises from repetition — "to reconcile invoices, call tool X then Y." The arc of a maturing agent is the steady promotion of episodes into semantics and procedures: experience becoming expertise[3].

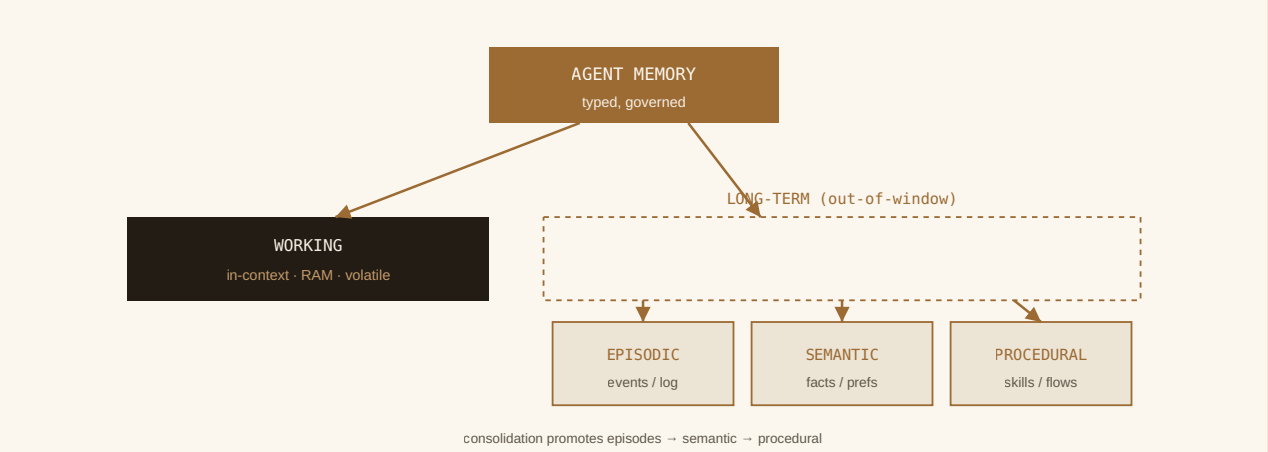

Fig. 1 — A working taxonomy after CoALA[2]. Working memory lives in the window; long-term memory is typed into episodic, semantic and procedural stores, each with its own write, retrieval and decay policy.

03 — THE MEMORY LIFECYCLE

Encode, consolidate, retrieve, forget

Memory is a loop, not a database. Four operations define it, and each is a design decision with cost and quality consequences.

Encode (write). After each interaction the agent must decide what — if anything — is worth keeping, and extract it as a structured entry rather than a raw transcript. This write path is itself an inference cost; recent systems slash it dramatically. Mem0's single-pass, ADD-only extraction reduces write-time

model calls by 60–70% with negligible quality loss[10]. **Consolidate.** Periodically, raw episodes are synthesised into durable insight — the "reflection" mechanism popularised by Generative Agents, which turns a stream of events into higher-level semantic memory[3]. **Retrieve (read).** At decision time, a small set of relevant memories is selected and injected — by recency, importance and relevance — keeping the working window lean. **Forget.** The least-developed and most-needed operation: without decay, stores grow without bound and retrieval quality collapses under its own weight[11].

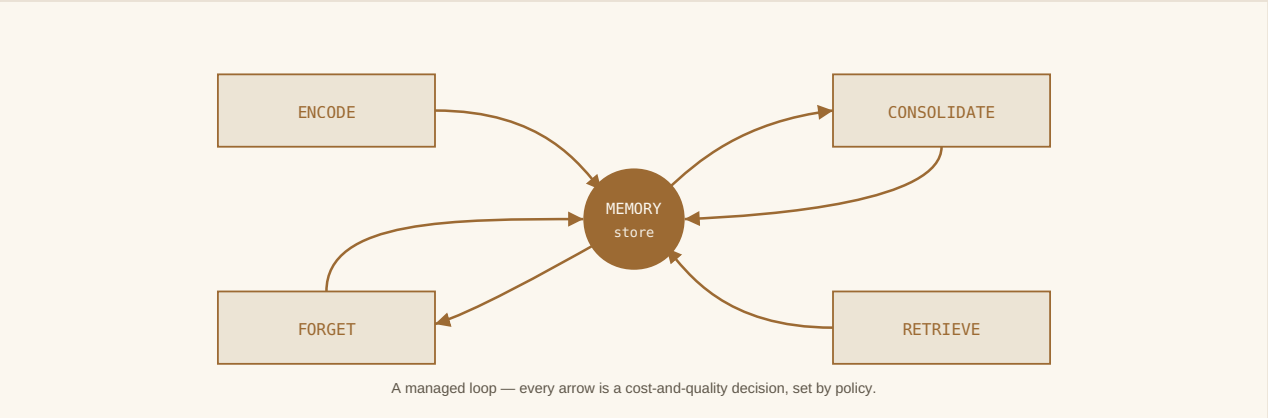

Fig. 2 — The memory lifecycle. Encoding is structured extraction, not transcript dumping[10]; consolidation distils episodes into insight[3]; retrieval keeps the window lean; forgetting bounds growth[11].

04 — ARCHITECTURE PATTERNS

Three ways to hold state

Three architectures dominate production, and they are complementary rather than competing.

The OS-style hierarchy (self-editing memory)

MemGPT — now the Letta framework — frames the LLM as a process on a memory-constrained operating system, with a tiered hierarchy: *core* memory in the window (RAM), *recall* memory for searchable history (a disk cache), and *archival* memory for cold storage[1]. Crucially, the agent *edits its own memory* through tool calls during its normal loop: when the user says "call me Alice," the model itself writes that to its core block. Memory becomes an action, not a side-effect.

The extraction-based external store

Mem0 takes the opposite tack: a dedicated memory layer that extracts salient facts from each exchange, consolidates them into a structured store, and retrieves a compact set on demand — keeping per-call context far below a full-history approach[6]. This is the pattern when memory must scale across many users and sessions with predictable cost.

File-based memory and context editing

The newest production primitive treats memory as a filesystem the model can read and write outside the window. Anthropic's memory tool persists information across conversations via files in the

developer's own infrastructure, while *context editing* automatically clears stale tool results as the window fills — together completing long workflows that would otherwise exhaust context[7].

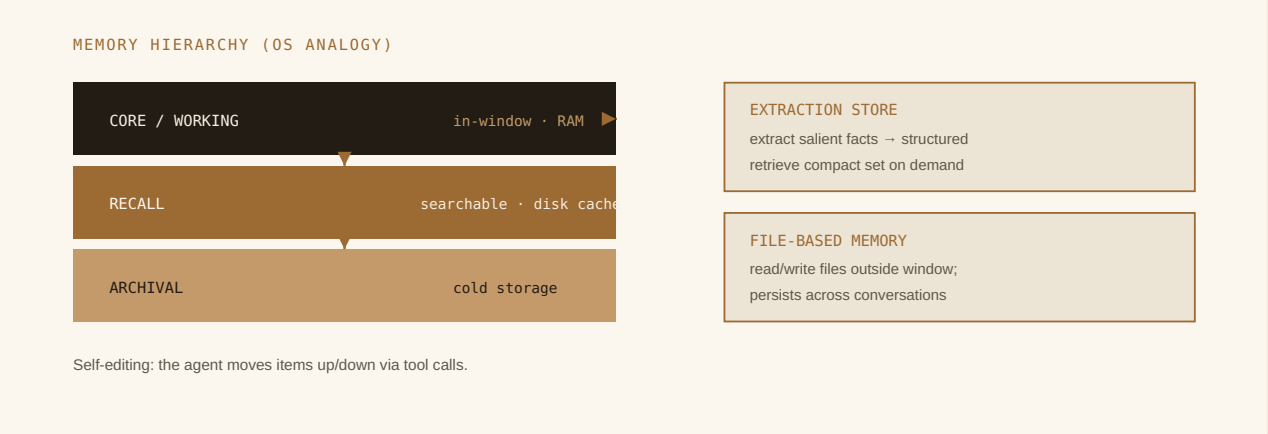

Fig. 3 — Three complementary patterns: an OS-style tiered hierarchy with self-editing blocks[1]; an extraction-based external store[6]; and file-based memory with context editing[7].

05 — THE COST EQUATION

Remembering without blowing the budget

The promise of memory is not only accuracy; it is doing better *and* cheaper than stuffing history into context. The numbers are now decisive. Against a full-context baseline, Mem0 reports a **91% lower p95 latency** and **over 90% token-cost savings**, while *improving* answer quality by 26% (LLM-as-judge) over a leading commercial memory[6]. Across the major long-memory benchmarks, structured memory averages under ~7,000 tokens per retrieval where full-context approaches consume 25,000 or more — a 70–85% reduction in active context[10].

91%

Lower p95 latency vs. processing full conversation history[6].

>90%

Token-cost savings vs. full-context, with higher answer quality[6].

~5×

Less test-time compute for equal accuracy when memory work is done off-peak ("sleep-time")[9].

The deeper lever is *when* the work happens. Memory operations — extraction, consolidation, re-indexing — do not have to sit on the user's critical path. *Sleep-time compute* moves them to idle periods: the agent "thinks" offline, turning raw context into learned context, cutting the test-time compute needed for equal accuracy by roughly 5× and the average cost per query by ~2.5× when amortised across related queries[9]. Memory, done well, is not a tax on every turn; it is an investment made when the meter is cheapest.

STRATEGY	CONTINUITY	COST PER TURN	BEST WHEN
Full history in context	High (until it rots)	Grows every turn	Short tasks only
Compaction / context editing	Medium	Bounded [7]	Long single sessions
Extraction store + retrieval	High, cross-session	Low, flat [6]	Many users / sessions
Sleep-time consolidation	High, improving	Off critical path [9]	Recurring, predictable use

06 — MEASURING MEMORY

Benchmarks and the abilities that matter

Memory claims are cheap; measured memory is not. Three benchmarks anchor the field. LoCoMo evaluates very long-term conversational memory over dialogues averaging ~300 turns and up to 35 sessions [4]. LongMemEval isolates five distinct abilities — information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention — and finds commercial assistants and long-context models suffering roughly a 30% accuracy drop on sustained interaction [5]. BEAM pushes to the extreme, probing memory across conversations of up to ten million tokens [10].

> THE TWO ABILITIES PEOPLE UNDERESTIMATE
>
> **Knowledge updates** — overwriting a fact when it changes ("she switched jobs") — and **abstention** — knowing when the answer was never stored. Pure-recall systems quietly fail both: they surface stale facts and confabulate rather than admit ignorance. These, not raw recall, separate a toy from a trustworthy agent [5].

07 — FAILURE MODES & GOVERNANCE

Where memory becomes a liability

Memory introduces failure modes that retrieval alone never had, and they are precisely the ones a serious buyer asks about. **Unbounded growth:** reflection improves coherence but does not regulate the size of the store; without an explicit decay policy, traces accumulate until retrieval degrades [11]. **Stale and conflicting memory:** an old fact that should have been overwritten resurfaces as a confident error. **Memory poisoning:** a single bad write — accidental or adversarial — persists and contaminates every future session. And, decisively for the enterprise: **privacy.** A memory that remembers a customer remembers their personal data, indefinitely, across sessions.

This is where memory stops being a performance feature and becomes a governance one. A persistent store of user information is regulated data; under the EU's GDPR it carries a right to rectification and erasure. An agent memory that cannot be inspected, corrected, and deleted on request is not production-ready in Europe — it is a liability. The requirement is concrete: every memory must be

attributable to its source, editable, and forgettable, with an audit trail. Memory and sovereignty are the same conversation — the subject of Research Note N°03.

> An agent that cannot forget on request cannot be deployed on real customer data. Forgetting is not a feature gap — it is a compliance primitive.

08 — A REFERENCE ARCHITECTURE

Governed memory, as LinkTec Labs builds it

The patterns above compose into a single governed layer. The design below is the one we deploy — typed by memory class, explicit at every operation, and auditable end to end.

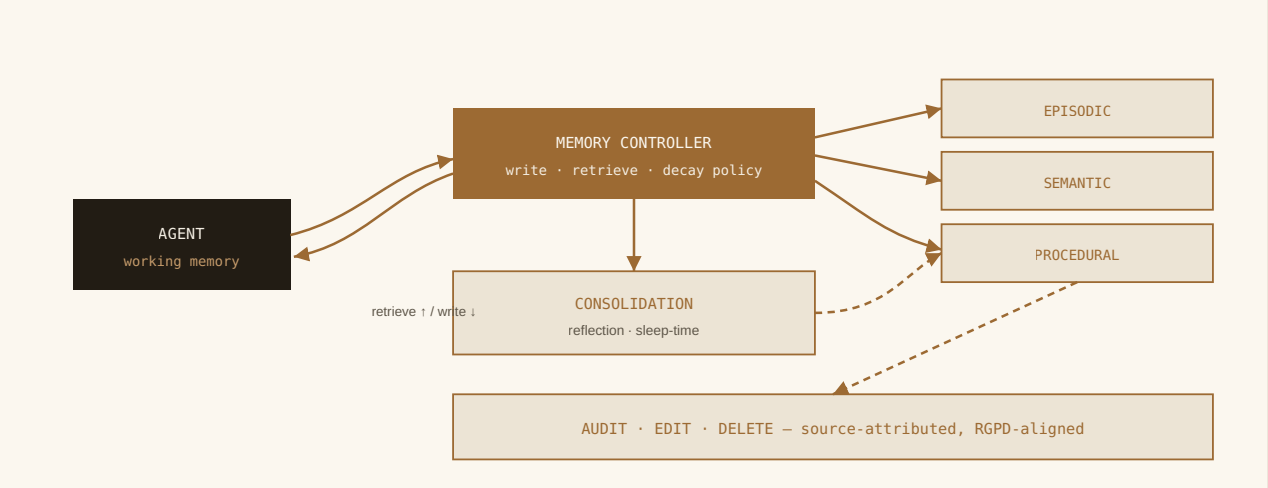

Fig. 4 — LinkTec Labs governed memory architecture. A memory controller mediates every write, retrieval and decay; consolidation runs off-peak; stores are typed by class; and an audit layer makes every memory attributable, editable and erasable. Sovereign-deployable by design.

Four principles govern it. **Memory is typed** — episodic, semantic and procedural stores have distinct write and decay policies; one undifferentiated blob is the road to bloat. **Every operation is explicit** — writing, retrieving and forgetting are governed by policy and instrumented, never emergent. **The budget is the constraint** — retrieval returns a small, ranked working set, and consolidation runs off the critical path. And **everything is governed** — each memory is source-attributed, editable and erasable, with an audit trail. The first three buy performance; the fourth is what makes the system deployable on real customer data.

09 — PRACTITIONER'S PLAYBOOK

What to do on Monday

→ **Separate capacity from continuity.** Don't solve a memory problem with a bigger window[6].

- → **Type your memory.** Working, episodic, semantic, procedural — each with its own write and decay rules[2].
- → **Write structured, not raw.** Extract salient facts; single-pass extraction cuts write cost 60–70%[10].
- → **Consolidate off-peak.** Reflect and re-index during idle time; sleep-time compute pays for itself[3][9].
- → **Retrieve a small set.** Rank by recency, importance and relevance; keep the window lean[3].
- → **Make forgetting a feature.** Decay policy bounds growth; explicit deletion satisfies the law[11].
- → **Test updates and abstention,** not just recall — that is where trust is won or lost[5].
- → **Audit every memory.** Source, edit, delete — or you cannot ship on customer data.

Capacity lets an agent read. Memory lets it learn. The discipline is keeping the second from eating the budget — or the trust.

LinkTec Labs designs governed, sovereignty-ready memory layers for agents that run long and return often — auditable, editable, and sized for real cost. Second in our 2026 research series; the first, *Beyond RAG: Context Architecture*, is its natural companion. **linktec.fr / labs**

REFERENCES

Primary & technical sources

[1] Packer et al. **MemGPT: Towards LLMs as Operating Systems.** 2023. arXiv:2310.08560 — https://arxiv.org/abs/2310.08560

[2] Sumers, Yao, Narasimhan & Griffiths. **Cognitive Architectures for Language Agents (CoALA).** TMLR 2024. arXiv:2309.02427 — https://arxiv.org/abs/2309.02427

[3] Park et al. **Generative Agents: Interactive Simulacra of Human Behavior.** UIST 2023. arXiv:2304.03442 — https://arxiv.org/abs/2304.03442

[4] Maharana et al. **Evaluating Very Long-Term Conversational Memory of LLM Agents (LoCoMo).** ACL 2024. arXiv:2402.17753 — https://arxiv.org/abs/2402.17753

[5] Wu et al. **LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory.** ICLR 2025. arXiv:2410.10813 — https://arxiv.org/abs/2410.10813

[6] Chhikara et al. **Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory.** 2025. arXiv:2504.19413 — https://arxiv.org/abs/2504.19413

[7] Anthropic. **Managing context on the Claude Developer Platform** (memory tool & context editing). 2025 — https://www.anthropic.com/news/context-management

[8] Anthropic. **Effective Context Engineering for AI Agents.** Anthropic Engineering, 2025 — https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

[9] Lin et al. **Sleep-time Compute: Beyond Inference Scaling at Test-time.** 2025. arXiv:2504.13171 — https://arxiv.org/abs/2504.13171

[10] Mem0. **AI Memory Benchmarks 2026: LoCoMo, LongMemEval & BEAM** (cross-benchmark figures; secondary). 2026 — https://mem0.ai/blog/ai-memory-benchmarks-in-2026

[11] Survey. **Memory for Autonomous LLM Agents: Mechanisms, Evaluation, and Emerging Frontiers.** 2026. arXiv:2603.07670 — https://arxiv.org/abs/2603.07670