WHITE PAPER · AGENT MEMORY · V1.0

The hippocampal index for *AI agents*.

Most AI agents with long-term memory get slower and more expensive the more they remember, because some step on every turn reads the whole store to find out what is in it. A human brain does not: the hippocampus keeps a small **index** that points at where memories live, and recall goes through the index. This paper proposes a **hippocampal index** for agents: one compact record per memory, kept in step by the write path and repaired offline, so a turn learns what the agent knows in a fixed handful of reads however large its memory grows.



Author Bizer **Audience** People who build AI agents **Published** October 2026

Download the PDF

THE CORE IDEA

Don't scan the store. Keep an index that points into it.

Each memory gets one small index entry: what it is called and when it last changed. A turn reads the index to learn everything the agent knows, and opens

only the few memories the model chooses. Every write keeps the index current, and a nightly pass repairs it, the way sleep does the brain's heavy housekeeping.

CONTENTS

01 The problem
02 What brain research says
03 How agents find memories today
04 The hippocampal index
05 With the episodic buffer
06 Design rules
07 Reference implementation
08 Use cases
09 Evaluating it
10 Case study: Miles at Bizer
11 Limits and open questions

§01 · THE PROBLEM

Memory that costs more the more it knows

An agent with long-term memory has to know what it knows before it can use any of it. Most designs show the model a list of what is stored (note titles, file paths, entity names) so it can choose what to open. Building that list is where the trouble starts.

The easy way to build it is to read the store. In a document database that means fetching every memory record on every turn, often with its full content, only to keep the names. At 100 memories nobody notices. At 5,000 memories averaging 2 KB, every message pays 5,000 reads and moves about 10 MB before the model sees a word. The

cost of a turn grows with everything the agent has ever learned, on the one path where latency matters most.

The usual fixes either hide memories or move the cost:

- **Cap the list** and the prompt stays small, but anything past the cap disappears from what the model can see, and from any check that trusts the list.
- **Search only** keeps the turn cheap, but the model must already suspect a memory exists before it looks.
- **A bigger context window** fits more names and still pays to read the whole store first. Models also use the middle of a long context worst (Liu et al., 2023).

A human brain stores a lifetime of experience, and finding one memory does not get slower as that store grows.

---

§02 · WHAT BRAIN RESEARCH SAYS

Five findings, one design

1. The hippocampus holds an index, not the memory

Hippocampal indexing theory proposes that the content of an experience stays in the cortex where it was processed. The hippocampus stores a compact pattern that points at it, and reactivating that pattern brings the content back (Teyler and DiScenna, 1986; updated by Teyler and Rudy, 2007). Finding a memory goes through the index, never through the store.

2. Each memory is a small, sparse pointer

Engram research finds that a memory is held by a sparse set of neurons, allocated competitively to the cells most excitable at the time of learning (Josselyn and Tonegawa, 2020). The pointer is small however large the experience was.

3. Recall is completed from a cue

Models of the hippocampus give its CA3 region recurrent connections that complete a whole stored pattern from part of it, while the dentate gyrus keeps similar memories apart (Yassa and Stark, 2011). The cost of recall depends on the cue, not on how much else is stored.

4. Heavy reorganizing happens offline

Complementary learning systems theory pairs a fast store for episodes with a slow one for structure, linked by replay (Kumaran, Hassabis and McClelland, 2016). The synaptic homeostasis hypothesis adds that sleep renormalizes what waking learning strengthened, keeping the strongest connections (Tononi and Cirelli, 2014). The waking path stays a lookup because the expensive work waits for sleep.

5. Storage is not the limit; access is

Measured synapses hold about 4.7 bits each, and total capacity may be an order of magnitude higher than earlier estimates (Bartol et al., 2015). Forgetting tracks how likely a memory is to be needed, judged by how recently and how often it was used (Anderson and Schooler, 1991). What the brain manages is what reaches attention, not what it keeps.

Taken together: a brain keeps everything, finds things through a small index driven by cues, and does its expensive housekeeping while asleep. An agent's memory can work the same way.

---

§03 · HOW AGENTS FIND MEMORIES TODAY

Each approach reads too much, shows too little, or waits for a guess

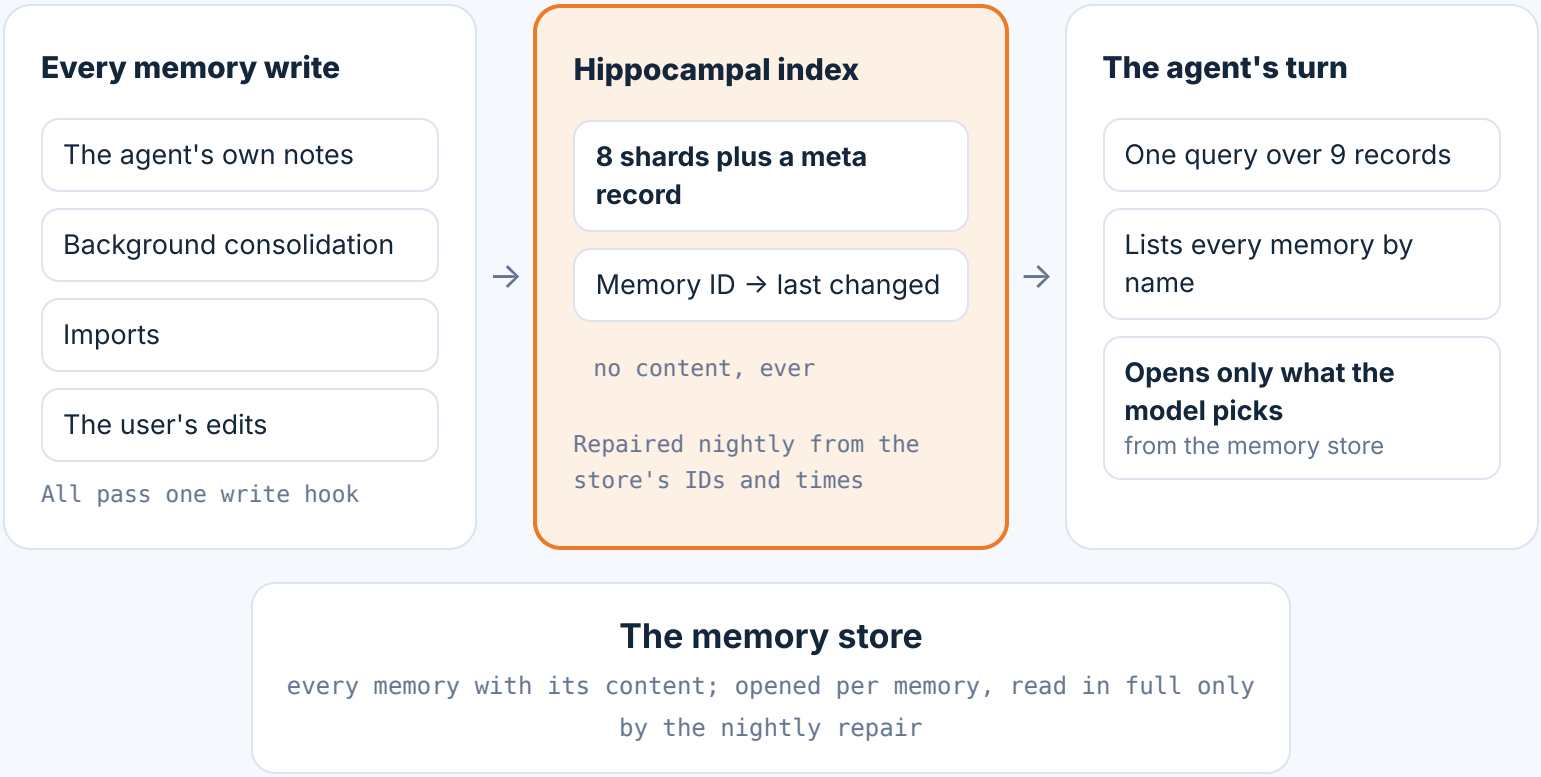
APPROACH	WHAT A TURN READS	COST AS MEMORY GROWS	WEAK POINT
Scan the store for a list	Every record, often with content	Grows with every memory	Latency and reads on every message
Small always-loaded file (Hermes)	One or two short files	Flat	The file is capped, so most memories are not in it
Search only (MemGPT archival, RAG)	Results of a search the model chose to run	Flat	The model must suspect a memory exists before it looks
Graph index (HippoRAG)	Graph neighbourhood of the query's entities	Flat per query	Needs an extraction pass to build the graph
Hippocampal index	A fixed handful of small index records	**Flat**	Eventual: a new memory appears once its write is indexed

Hermes Agent keeps two small files in the prompt (about 2,200 and 1,375 characters) and reaches the rest through a full-text search over past sessions. MemGPT pages information between the context window and external storage with function calls. HippoRAG builds a knowledge graph as an artificial hippocampal index and runs Personalized PageRank from the query; its authors report single-step retrieval 10 to 20 times cheaper than iterative retrieval. Generative Agents rank memories by recency, importance and relevance. All of them keep the turn's cost flat by never reading the whole store. The gap this paper fills is narrower and more common: agents that show the model a list of what they know, and build that list by scanning.

It is a companion to our earlier paper, The Episodic Buffer for AI Agents; §05 shows how the two work together.

One small entry per memory, read in one query

A hippocampal index is one small record per memory, naming the memory and when it last changed, kept apart from the memory itself. A turn reads the index to learn what the agent knows, and touches the store only to open the few memories the model chooses.



Every write passes one hook that updates the index, the turn reads the index in one query and opens only what the model picks, and a nightly pass repairs the index from the store.

§05 · WITH THE EPISODIC BUFFER

Two indexes, one store

Our earlier paper, The Episodic Buffer for AI Agents, indexes one conversation: a line for each stretch that scrolled out of view, naming the memory pages it was filed into. The hippocampal index indexes the whole store. A turn reads both, each in a fixed handful of reads, so its cost stays flat as threads get longer and as memory grows. They meet in three places.

WHERE THEY MEET	WHAT HAPPENS
One pass writes both	The memory pass after a reply files facts into pages and writes the buffer line that names them. Each page write passes the index's write hook, so the line and the index entries for its pages land together.
Recall	The latest buffer lines are added to the recall query, steering it toward old topics the user's words don't name. Every hit must be on the index's list before it reaches the model: the buffer suggests, the index confirms.
Following a pointer	Before a buffer line is shown, each page it points at is checked against the index. A page merged or deleted since is left out, never guessed at, and the line keeps its gist. If a pointer still misses, the closest-name suggestions come from the index, not a scan.

● what the agent sees · an illustrative example

```
[EARLIER IN THIS THREAD]
- 2026-10-02: Owner wants a second location in Houma (filed in pages/Second Location)
- … more of this thread happened here; search_chat_history finds it.
- 2026-10-11: Compared two lease quotes; leaning to Main St (filed in pages/Second Location)
  pages/Old Lease was merged into pages/Second Location on 2026-10-14,
  so the index no longer lists it and the line no longer points at it.
```

The check costs nothing extra: the turn has already read the index to build its list. Both papers follow one rule, the brain's: point at memory, don't load it, and let the model choose what to open.

---

§06 · DESIGN RULES

Six rules, each from the research

- **Point, don't hold.** An index entry is an ID and the time the memory last changed. Never its content. The content stays where it is, read only for the few memories a turn actually opens.

- **Fixed cost per turn.** The index lives in a small, fixed number of records (shards), read in one query. A turn's cost is the shard count, at ten memories or a hundred thousand.
- **One write path keeps it current.** Every way a memory changes (the agent's own writes, background consolidation, imports, the user's edits) passes through one hook that updates the index. No writer has to remember it.
- **Sleep repairs it.** An offline pass that already reads the store rebuilds the index from it, so a missed or out-of-order update is gone by morning. A rebuild never drops an entry newer than the scan it rebuilt from.
- **A cache, never a source of truth.** Every entry is derived from a stored memory. A missing or outdated index falls back to the old scan, reading names only, and starts a build. Nothing is ever invented from the index.
- **The model still chooses.** The index changes how the list is built, not what the model sees. The same list, in the same order, reaches the model, and the model decides which memory to open.

---

§07 · REFERENCE IMPLEMENTATION

A record, a hook, a reader and a repair

The index fits any agent whose memories are stored as separate records with stable IDs.

1. The index record

A few shard documents per agent or tenant, plus one meta document, stored where only the server writes. Make the memory ID reversible to its name (for example, the path in base64url) so the index never stores names twice. At about 70 bytes an entry, eight shards of 1 MB hold roughly 100,000 memories.

FIELD	MEANING
shard `e`	A map of memory ID to last-changed time, for the memories hashed to this shard
meta `version`	The layout version; any other value means unbuilt
meta `shards`	How many shards the layout uses
meta `builtAt`	When the last full build or repair ran

2. The write hook

Attach one trigger to the memory store, not to each writer. Use a stable hash of the ID to pick the shard, and a merge write so two updates to the same shard never clobber each other.

● memory store trigger · pseudocode

```
on memory written(id, record):
    if record is deleted: shard(id).e[id] = delete
    else:                 shard(id).e[id] = record.updated_at
```

3. The read path

Replace the scan with one query, and return the same list, in the same order, that the scan returned. Everything downstream keeps working unchanged.

● turn assembly · pseudocode

```
docs = index(tenant).get_all()
if meta missing or meta.version != VERSION:
    build_soon(tenant)
    return scan(store, fields=["name"]).order_by(updated_at desc)
return decode(all entries).order_by(updated_at desc, name)
```

4. The repair pass

Run it inside the offline pass the agent already has:

nightly maintenance · pseudocode

```
scan_started = now()
scanned = store.ids_and_times()
for each shard: merge-set scanned entries
remove entries not in scanned and older than scan_started
```

The age check is what makes a repair safe to run while the agent is live: a memory written during the scan is newer than `scan_started`, so it is never removed.

5. Guardrails

Only the server writes the index; a client that could write it could plant a name in the agent's memory list. Treat an index entry as a pointer only: if the memory it names is gone, opening it returns nothing, as it would for any missing name.

---

§08 · USE CASES

Any agent whose memory keeps growing

These are the cases where we expect the index to pay off; only the last has been built.

AGENT	WHAT PILES UP	WHAT THE LIST SHOWS	WITHOUT AN INDEX, EACH TURN READS
Customer support	One note per customer and per resolved issue	Customer and issue names	Every customer note ever written
Sales and CRM	Account, contact and deal notes	Account names	Every account's full notes
Coding agent	Design notes, decisions, file summaries	Note titles	Every note, including long file summaries

AGENT	WHAT PILES UP	WHAT THE LIST SHOWS	WITHOUT AN INDEX, EACH TURN READS
Executive assistant	Years of preferences, people and plans	Topics and people	A lifetime of notes
Imported memory	Hundreds of records from an import	Imported record names	Every imported record, at full size
Business operations (Bizer)	Pages per customer, supplier and topic, plus imports	Page paths	Every page and import, content included

WHERE IT WON'T HELP

An agent with a few dozen memories gains little; the scan is already cheap. An agent that never shows its model a list, and finds memories only by search, has no list to build. And the index does not make search cheaper: a search that scans every memory's content needs its own fix.

§09 · EVALUATING IT

Same list, flat cost?

Two questions matter: is the list exactly what the scan would have produced, and does a turn's cost stop growing with memory? Test both, before and after.

- **Equivalence.** On real stores, build the list both ways and compare. Same names, same order. Any difference is a bug, not a trade-off.
- **Cost curve.** Seed stores of 100, 1,000, 5,000 and 20,000 memories. Measure reads, bytes moved and time to first token for one turn on each. The scan should rise in a line; the index should stay flat.
- **Freshness.** Write a memory, then start a turn after 1, 5 and 30 seconds. Record how often the new memory is in the list.

- **Repair.** Drop index updates at random, or apply a delete before its write, then run the repair pass. The index should match the store afterward, and a memory written during the repair should survive it.

Also measure the write side: one extra small write per memory change, and the repair pass's reads. Both should be small next to what a turn used to read.

---

§10 · CASE STUDY

Miles at Bizer

Bizer, a business development platform, built the index into its agent Miles in October 2026. Miles keeps each business's memory as markdown notes: pages it maintains, a daily journal, the owner's notebook, and sources imported from other AI assistants. On every chat and voice turn, Miles's prompt lists those notes by path (up to 120 pages, 40 sources and 60 notebook entries), and the same list feeds recall: title matching, and the check that a semantic search hit still exists.

The list was built by reading every note in full, content included, on every turn. The question that found it was an admin flag: one business, with 93 pages and 66 imported sources, showed "over cap" because 66 is more than the 40 sources the prompt names. The cap was fine. Reading all 159 notes, every message, to name 133 of them was not.

CHOICE	VALUE AT BIZER
Index layout	8 shards plus a meta document, per business
Entry	Note ID (the path in base64url) to last-changed time
Write hook	The trigger that already kept the semantic index current
Repair	The nightly 03:00 maintenance pass, ahead of its plan checks

CHOICE	VALUE AT BIZER
Fallback	Scan paths only, never content, and start a build
Buffer pointers	Checked against the index before each episodic buffer line is shown
Who writes	Server only
Turn cost	One query over 9 small documents

Illustrative cost per turn, assuming notes average 2 KB:

NOTES IN MEMORY	READS BEFORE	DATA MOVED BEFORE	READS AFTER
159	159	about 0.3 MB	**9**
1,000	1,000	about 2 MB	**9**
5,000	5,000	about 10 MB	**9**

Existing businesses move over on their first turn after deploy, with no migration. Bizer has tested the design with unit tests and against a Firestore emulator (build, repair, deletes, a write during a repair), but has not yet measured it on live traffic.

---

§11 · LIMITS AND OPEN QUESTIONS

Simple, unmeasured, worth testing

- **No live measurement yet.** Ours included. The evaluation above is the next thing anyone adopting this should run.
- **Eventual, not instant.** A memory written mid-turn reaches the index when its trigger runs, usually seconds later. Nothing is lost in that gap: the conversation is still in the agent's context window, its short-term memory, and the page opens by name the

moment it is saved. Only the list of names catches up, the way the passage index always has.

- **Stale entries until repair.** A delete applied before its write can leave a name in the list until the nightly pass. Opening it returns nothing, so no wrong content reaches the model, but the list is briefly longer than the truth.
- **A renamed page loses its pointer.** A rename is a delete plus a write, so a buffer line pointing at the old name drops the pointer rather than following the rename. The gist still says what the stretch was about, and search still finds the page.
- **It fixes listing, not searching.** A search tool that scans every memory's content still grows with memory. So does any background pass that reads every page each time it runs.
- **Is it new?** Indexes are as old as databases, and HippoRAG already builds a hippocampal index for retrieval. What we have not seen written down is the narrower point: that an agent's own list of what it knows is built by scanning, on every turn, and should be an index kept by the write path and repaired offline. Treat this as a claim to check.

Three extensions

- **Search through the index too.** Send the search tool through a vector index of passages, as recall already does, and scan only a store that hasn't been embedded yet.
- **Accessibility from use.** Record how often each memory is recalled and found helpful, and rank the list by an ACT-R style base level with power-law decay, so the names the model sees first are the ones it actually uses. Nothing is deleted; only the order moves.
- **Background passes on the index.** Let consolidation and other post-turn passes pick pages through the index and semantic search, so they stop reading every page after every reply.

A cap on the list is a smaller notebook. An index is how the brain stopped paying for everything it knows.

Keep one small entry per memory, update it on every write, and repair it while the agent sleeps. An agent can do it with one index record per memory, one hook on its memory store and one step in a pass it already runs at night.

SOURCES

- Teyler, T. J. and DiScenna, P. (1986). The hippocampal memory indexing theory. *Behavioral Neuroscience* 100: 147–154. Record.
- Teyler, T. J. and Rudy, J. W. (2007). The hippocampal indexing theory and episodic memory: updating the index. *Hippocampus* 17(12): 1158–1169. Record.
- Josselyn, S. A. and Tonegawa, S. (2020). Memory engrams: recalling the past and imagining the future. *Science* 367: eaaw4325.
- Yassa, M. A. and Stark, C. E. L. (2011). Pattern separation in the hippocampus. *Trends in Neurosciences* 34(10): 515–525.
- Kumaran, D., Hassabis, D. and McClelland, J. L. (2016). What learning systems do intelligent agents need? Complementary learning systems theory updated. *Trends in Cognitive Sciences* 20: 512–534.
- Tononi, G. and Cirelli, C. (2014). Sleep and the price of plasticity. *Neuron* 81: 12–34.
- Bartol, T. M. et al. (2015). Nanoconnectomic upper bound on the variability of synaptic plasticity. *eLife* 4: e10778.
- Anderson, J. R. and Schooler, L. J. (1991). Reflections of the environment in memory. *Psychological Science* 2: 396–408.
- Liu, N. F. et al. (2023). Lost in the Middle: How Language Models Use Long Contexts. *TACL*.

- Gutiérrez, B. J. et al. (2024). HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *NeurIPS 2024*.
- Packer, C. et al. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
- Park, J. S. et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442.
- Nous Research. Hermes Agent: memory.
- Bizer (2026). The Episodic Buffer for AI Agents.