Vector, Graph, or Key Value?

Choosing Your Agent's Memory

Your use case decides. One test tells you which memory.

Elizabeth Fuentes Leone

Developer Advocate, GenAI · AWS

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

It Lives in the Context Window, Not in Memory

In the context window

- Everything the agent knows rides here
- Nothing is persisted as memory
- More context is not more accurate
- The middle gets lost (U-curve)

When the window resets

- New process, new request: window gone
- Same question, generic answer
- Nothing carried over, because nothing was stored

Stateless by design. A bigger window resends history every turn and still forgets.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

How You Recall It Decides the Store

By a known KEY

Key Value

Exact lookup by a known key.

“Book my usual cabin.”

By MEANING

Vector

Similarity over embeddings.

“Anything I should avoid eating here?”

By RELATIONSHIP

Graph

Entities linked by edges.

“Which trips go on the Madrid project?”

One stored fact, three ways to ask for it back. The way you recall picks the store.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Key Value

Key Value: Store the Result Under a Key

“Book my usual cabin” needs an exact value under a known key.



WRITE: the tool copies a field from the result into state. No model, no learning.

- Named key in, named key out: fast, cheap, exact
- Deterministic code: the tool reads the offer and writes the value
- The store is the only variable: a dict, a SQL row, DynamoDB, or S3

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Key Value

Key Value: The Code

```python
@tool(context=True)                          # context=True gives the tool agent.state
def book_flight(offer_id, tool_context):
    offer = flights_api.get_offer(offer_id)      # the tool's result

    prefs = tool_context.agent.state.get("user_preferences") or {}
    prefs["preferred_cabin"] = offer["cabin"]    # copy a field
    tool_context.agent.state.set("user_preferences", prefs)  # store it
```

STRANDS AGENTS SDK

@tool(context=True) is what gives the tool access to agent.state. That's the key-value store.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Wiring Memory into the Agent

In the Strands harness, memory plugs into the agent two ways, both one line.

You manage it

- A tool reads/writes agent.state
- @tool(context=True)
- Best for key value: exact, cheap

The framework manages it

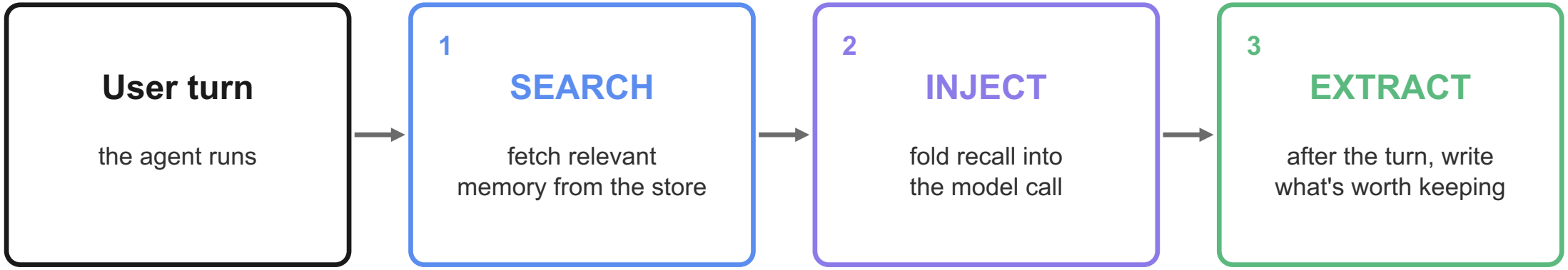
- memory_manager=MemoryManager(...)
- Extract, recall, inject, automatic
- Any MemoryStore: file, KB, vector, graph

Let's see the framework path: how Strands handles memory in the agent loop.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

How Strands Memory Works: One Hook

Attach a MemoryManager with any store. It hooks three points of every turn.



You attach it once: memory_manager=MemoryManager(stores=[store]). The manager does the rest.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Vector

Vector in the Agent Loop

```
# 1) Pick the store. 2) Hand it to the agent. That's the wiring.
store = VectorMemoryStore(name="traveler", extraction=ExtractionConfig(...))
agent = Agent(model=MODEL, memory_manager=MemoryManager(stores=[store]))

agent("I'm vegetarian and allergic to peanuts.")    # turn ends → EXTRACT + store

agent("Anything I should avoid eating here?")  # before model → INJECT recall
# the agent answers with the dietary note, matched by meaning
```

STRANDS AGENTS SDK

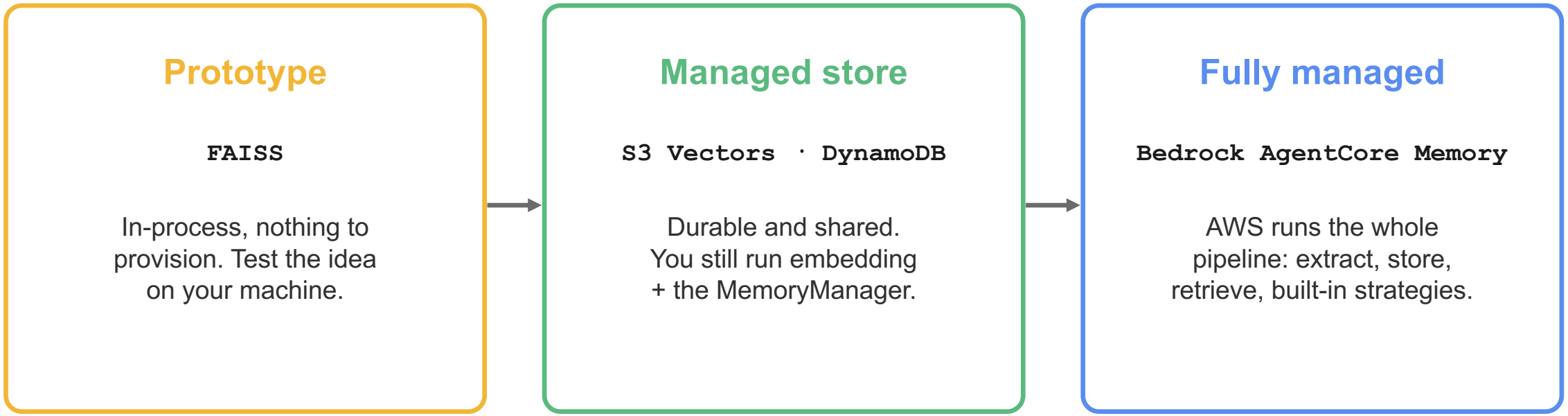
You never call the store directly. The MemoryManager extracts, recalls, and injects in the loop.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Vector

Vector: From Local Test to Fully Managed

Same store contract, three levels. You operate less, left to right.



You operate less, left to right. Same embeddings, same answer, only the operating model changes.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Graph

Where Graph Is Better

“Which of my trips go on the Madrid project?”

Vector retrieves

similar fragments

- “flight to Madrid, June”
- “Madrid project kickoff”
- “holiday in Barcelona”

Pieces, never connected

Graph traverses

the relationships

Trip → Project → Expense policy

Follows edges to the exact trips

Answers the question

Similarity finds pieces. Only traversal connects them into an answer.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Graph

Graph: The Code

```python
from neo4j_graphrag import SimpleKGPipeline, VectorCypherRetriever

class GraphMemoryStore(MemoryStore):      # wraps Neo4j as a Strands store
    async def add(self, content, metadata=None):
        await SimpleKGPipeline(driver, schema).run_async(text=content)
    async def search(self, query, options=None):
        r = VectorCypherRetriever(driver, ...).search(query)   # vector + Cypher
        return [MemoryEntry(content=i.content) for i in r.items]

agent = Agent(model=MODEL, memory_manager=MemoryManager(stores=[store]))
```

STRANDS AGENTS SDK

The store wraps neo4j-graphrag (SimpleKGPipeline to write, VectorCypherRetriever to traverse).

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

On Top

Now: Store Only What Your Agent Needs

You know how to store memory. The next step: keep the right things, block the bad. Same Strands hook.

Keep

The MemoryManager's extractor decides what's worth keeping and drops the noise, in the background.

Block

A write-gate screens each memory before it persists, so dirty or injected content never gets in.

Same hook, no new plumbing. The selective and hygiene demos are in the repo →

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

On Top

Blast Radius Depends on the Store

Key value

- One key holds the bad value
- Only reads of that key are wrong
- Contained: cannot spread

Vector

- Bad entry has an embedding
- Surfaces on any similar query
- Spreads by meaning, not by key

Graph

- Bad fact adds a new edge
- Every traversal crossing it is wrong
- Widest reach, most sensitive

More connected memory reaches further when it goes bad. That is why you gate the write.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Which Memory to Pick

Match what your agent needs to recall to the store. Start simple, add only when needed.

Key Value	Recall by a known id	*Support agent: account tier by user id*
Vector	Answer by meaning, no key	*Shopping agent: “something warmer for the trip”*
Graph	Connect entities across hops	*Copilot: “who owns the service this touches?”*

The whole talk in one test: is it a known key, a meaning, or a relationship?

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Elizabeth Fuentes Leone

Developer Advocate, GenAI · AWS

Match the memory to your use case

Talk resources

bit.ly/4xvIRRr

Blog & Socials

elifuentes.tech

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.