

Your **Agent Memory** might be in the **Wrong Database**

1. The requirements an agentic workload puts on a database
2. A checklist to test any database against them
3. How Oracle AI Database measures up

What you leave with

1

The requirements

What an agentic workload actually asks of a database, drawn from the research research and from teams running memory memory in production.

2

The checklist

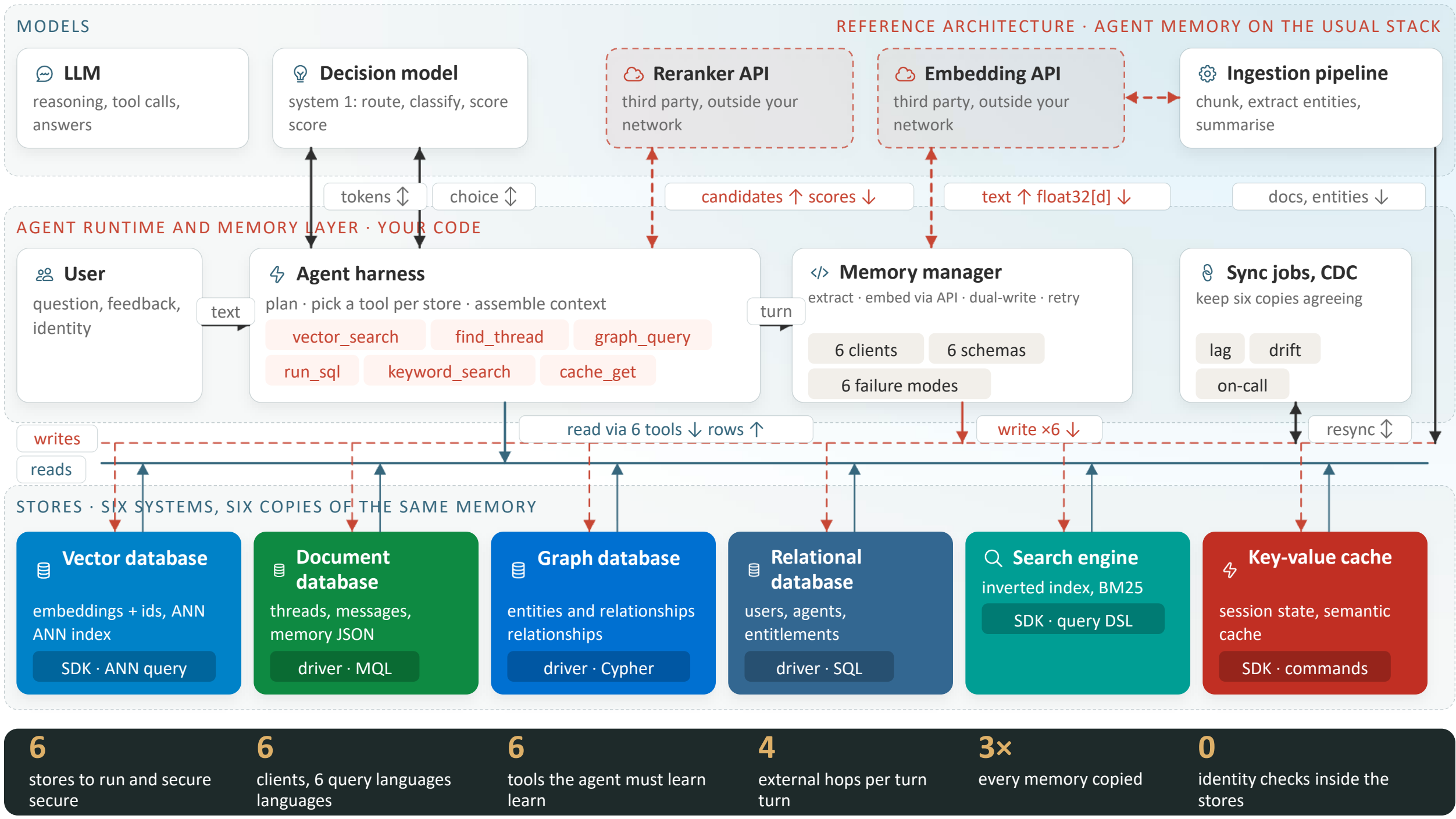
Twelve questions to ask of any database before you put agent memory on it. Score your current stack against it this week.

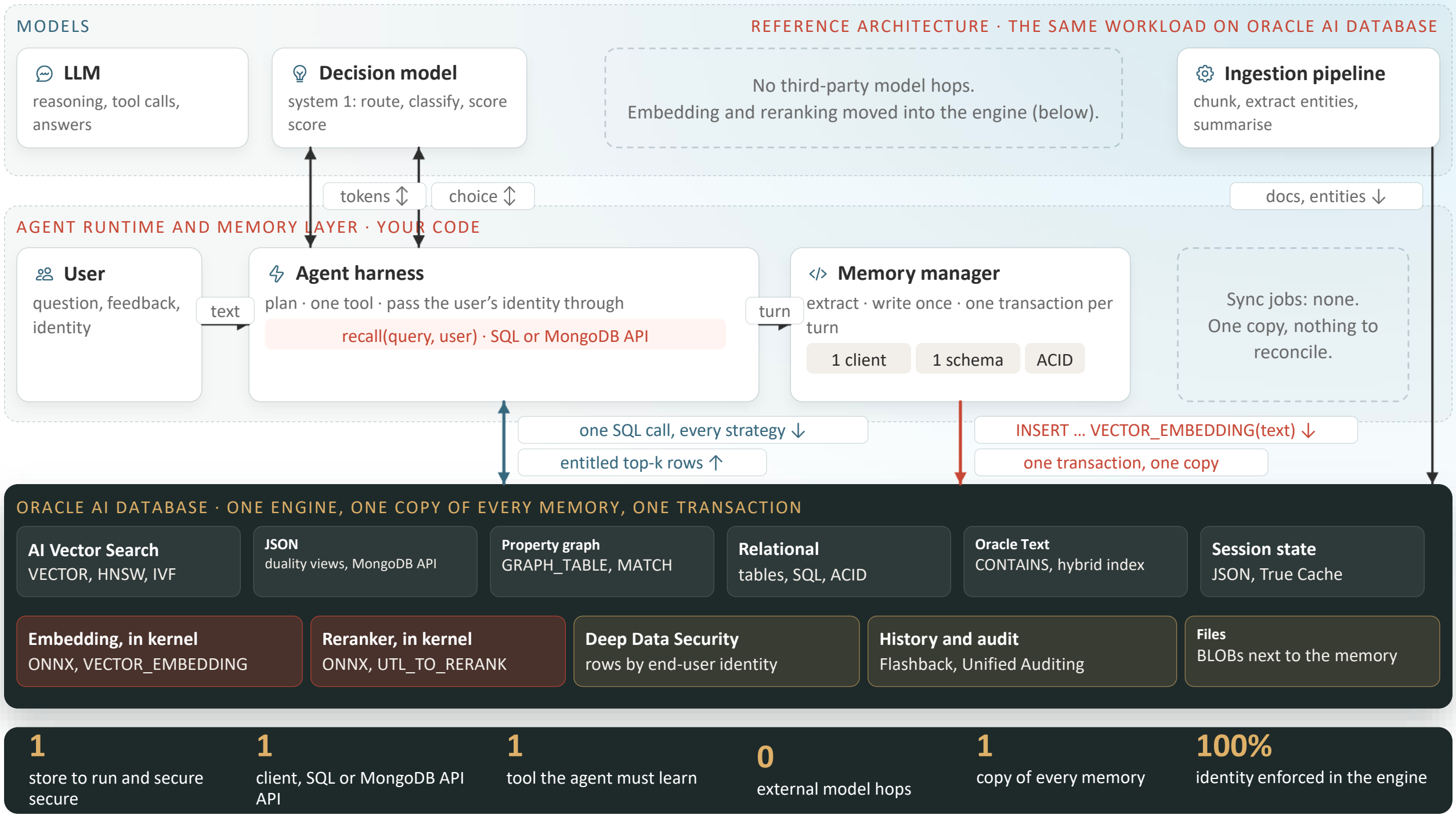
3

The proof

One notebook: the checklist proven live on Oracle AI Database Free, item by item, in SQL from Python.

The order matters: requirements first, so the product section has something to be measured against.





DEFINITION

Agent memory is the conceptualisation of the systems, memory management mechanisms and techniques by which an agent **retains**, **reuses**, **refines** and **recalls** what it learns across turns, sessions, agents and long-horizon tasks.

WHERE MEMORY LIVES

PARAMETRIC

Model weights

What the LLM or a decision model learned in training. Changed only by fine-tuning.

WORKING

Context window

What the model sees on this call, plus the KV cache that is retained and reused.

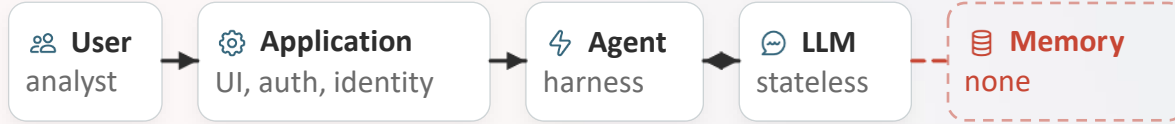
PERSISTENT

Database

What survives across sessions and agents, governed like the rest of your data. Today is about this one.

⚠ WITHOUT MEMORY · EVERY SESSION STARTS FROM ZERO

THE SYSTEM



SESSION 1 · USER

Explains the setup: “Desk A, report net exposure, not gross.”



APPLICATION → AGENT

Passes the message to the agent. Nothing is attached but this session’s transcript.



AGENT + LLM

Builds the context from this session only and answers correctly.



SESSION ENDS

Context window cleared. The explanation and the correction are gone.



SESSION 2 · USER

Asks for the report. Has to explain the desk and the rule all over again.



AGENT + LLM · MISTAKE

No way to recall session 1: reports gross exposure. The user corrects it a second time.



SESSION 3 · A SECOND AGENT

Same user, a different agent, the same explanation a third time.

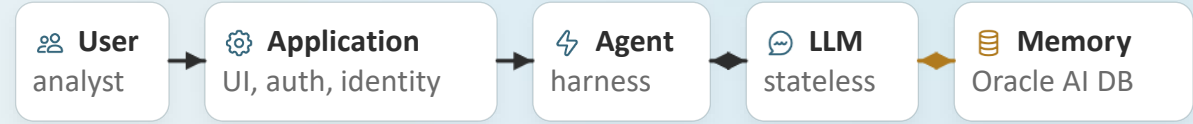


COST

Three times the tokens, the same mistake twice, a user who stops trusting it.

✅ WITH MEMORY · SESSIONS START FROM WHAT IS KNOWN

THE SYSTEM



SESSION 1 · USER

Explains once: “Desk A, report net exposure, not gross.”



APPLICATION → AGENT

Passes the message and the user’s identity. The memory layer is attached to the agent.



AGENT · RETAIN

Answers, then the memory manager extracts the preference and the correction and stores them with who and when.



SESSION ENDS

Context window cleared. The memory persists in the database, scoped to this user.



SESSION 2 · RECALL

Opens with the facts for this user already in context. Nothing re-explained; net exposure, correct.



USER CHANGES A DETAIL · REFINE

“Desk A moved to desk B.” The old fact is superseded, not deleted; history stays queryable.



SESSION 3 · A SECOND AGENT · REUSE

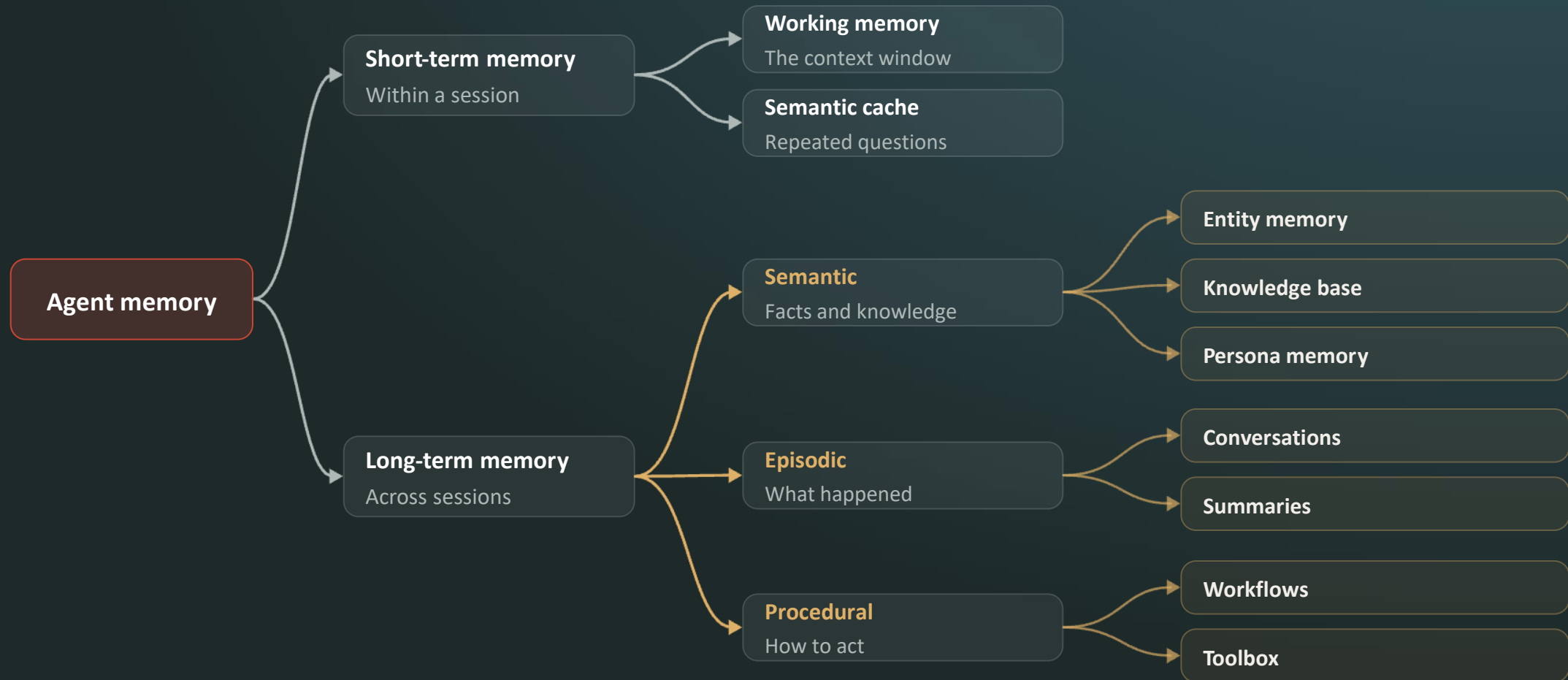
A different agent, same user, starts from the same memory. No third explanation.



RESULT

Shorter prompts, no repeated mistake, a user who is understood.

A taxonomy of agent memory



The 4 Rs of agent memory

R1

Retain

Decide what is worth keeping.
Not every message is a
memory.

extraction

R2

Reuse

Apply it again, across sessions
sessions and agents, so nobody
nobody pays for the same work
work twice.

shared scopes

R3

Refine

Correct, merge and update
what is known, and let what is
stale expire.

consolidation

R4

Recall

Bring back the right piece at
the right time, within a token
budget.

retrieval

Memory that only retains is a log. Memory that refines and recalls is a system, and a system needs a database built for it.

Every R is a database operation

THE R	WHAT THE AGENT DOES	WHAT THE DATABASE MUST DO
Retain	Extract facts, episodes and procedures from turns; turns; attach who, when and where they came from from	Write structured rows, documents, vectors and relationships in one place, with provenance, under one transaction
Reuse	Share what one agent learned with other agents and users, without leaking across them	Serve concurrent readers with consistent results and enforce scope by by identity inside the engine
Refine	Correct a fact, merge duplicates, supersede the old old version, let the stale expire	Atomic multi-record updates, read-your-writes, versions and validity windows instead of hard deletes
Recall	Find the right memories for this question, this user, this moment, inside a token budget	Vector, lexical, hybrid, filtered, graph and temporal retrieval, then reranking, all without copying data out

Memory layer is the code | Memory Management is the code | **The Memory Core is the database**

What is an agentic memory workload?

DEFINITION

An agentic memory workload is the **read and write traffic agents generate against a memory store** as they retain, reuse, refine and recall across turns, sessions, agents and long-horizon tasks.

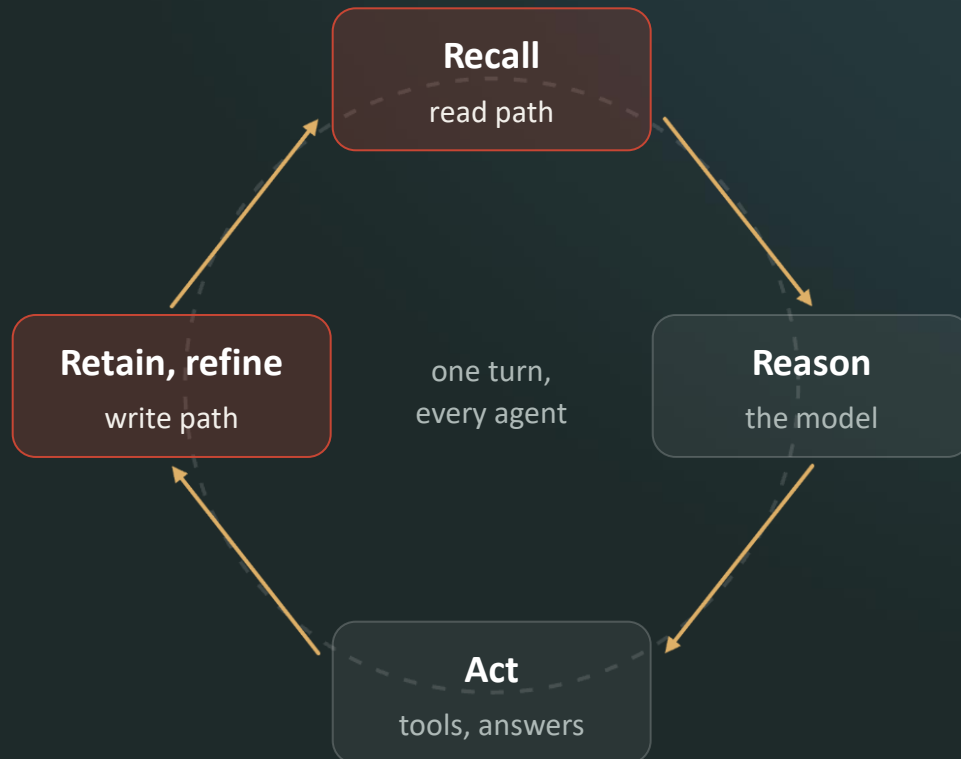
It is not a cache of chat history. It is many concurrent readers and writers, questions of many shapes, strict read-after-write inside a turn, and data that is scoped to people, long-lived and always changing.

YOU HAVE ONE IF YOUR AGENT...

- ✓ ...needs to recall a correction made two turns ago
read after write
- ✓ ...runs as several agents that share what they learn
concurrency
- ✓ ...asks “similar to”, “exactly this”, “this”, “related to”, “since when”
many retrieval shapes
- ✓ ...serves many users whose memories must not leak
scoping by identity
- ✓ ...keeps working across sessions, over days or weeks
long horizon
- ✓ ...corrects, merges and expires what it knows
refine, not just append
- ✓ ...has to show what it knew when it acted
history and audit
- ✓ ...keeps files and tool outputs next to facts
many data shapes

The agentic memory workload

Every turn of every agent runs this loop against the memory core.



Many agents, one memory

Fleets read and write the same memories concurrently



Read after write, inside a turn

A correction made now must be recalled on the next call



Questions of many shapes

Similar to, exactly this, related to, since when, for whom



Personal, shared and private at once

Some memories belong to a user, some to a team, some to no one



Long-lived, always changing

Facts get corrected, expire and need their history kept

Recall needs every retrieval strategy, because questions arrive in every shape

VECTOR

“Similar to this”

Semantic memory, paraphrased questions, the fuzzy match that makes memory feel intelligent

LEXICAL

“Exactly this”

Names, ticket numbers, error codes, product SKUs: embeddings blur precisely what these need sharp

HYBRID

“Both, fused”

Most real questions mix meaning and identifiers; fusion fusion of the two lists beats either alone

FILTERED

“For this user, since Monday”

Scope by user, agent, type, project and time on on every query, inside the index, not after it

GRAPH

“Related to this”

Entities and their relations, multi-hop: the customer, customer, their account, the incident last quarter

TEMPORAL

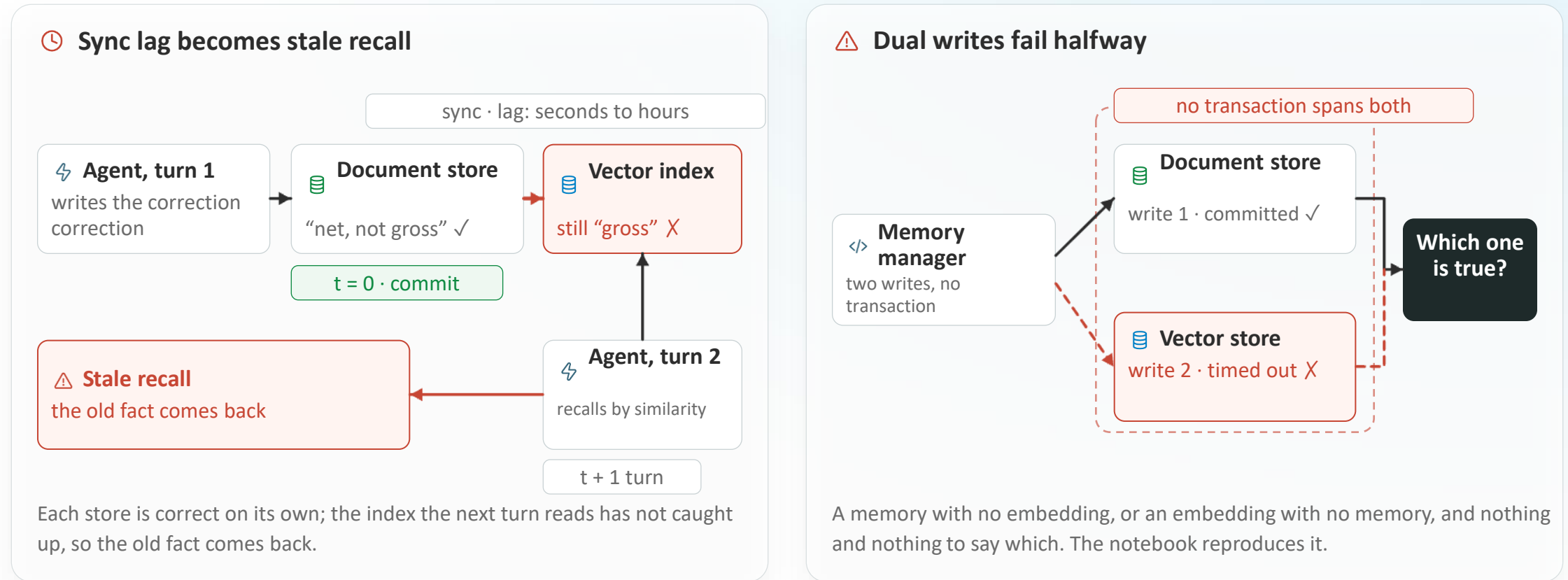
“What was true then”

Recency weighting, validity windows, point-in-time time reads, so corrections win over stale facts

RERANK

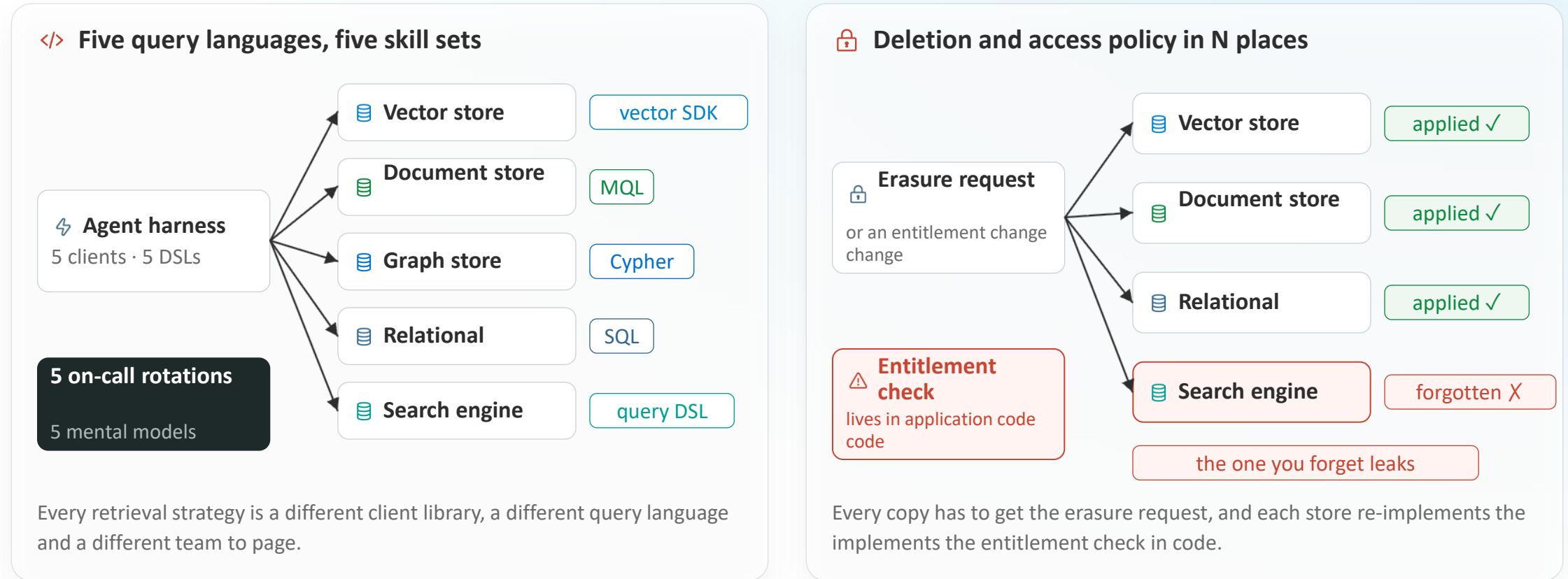
“The best five of the top fifty”: a second model re-scores candidates against the question. In the ICLR 2026 bottleneck study, hybrid retrieval plus reranking cut retrieval failures from 35% to 11%.

Those strategies have to run in one engine, over one copy (1 of 2)



A memory held in three stores is **three memories to keep consistent**. The overhead grows with the number of systems, not the number of memories.

Those strategies have to run in one engine, over one copy (2 of 2)



The requirement is not “fewer boxes”. It is **one index set over one copy**, so recall is consistent by construction and the tax never starts. never starts.

Refine is a transaction, and recall must see it immediately

ONE CORRECTION, THREE WRITES

1

Insert the corrected memory

with its embedding, its owner and its provenance
provenance

2

Close the superseded one

set its validity end, never delete it

3

Link old to new

a supersedes edge, so history is a graph you can
walk

ALL OR NOTHING

If write 2 fails after write 1, the agent now holds two contradictory facts with equal scores. Only a multi-record ACID transaction removes that state.

READ YOUR WRITES

The next call, from this agent or any other in the fleet, must recall the correction. Near-real-time indexing is a write-path bug for memory.

Shared memory across a fleet is a concurrency workload. **Consistency is the feature, not the overhead.**

Memory is the most personal data you hold, so it is governed like data

Access bound to the end user's identity

The agent carries the user's identity to the database and the database decides which rows exist for that user. Similarity is not authorisation, and filtering in application code is a leak waiting for a prompt injection.

Provenance on every memory

Which thread, which turn, which agent, which model extracted it, when. A memory with no provenance cannot be trusted, corrected or explained to an auditor.

Forget, don't delete

Versions, validity windows and point-in-time reads, so a corrected fact stops fact stops being recalled but its history stays. A real erasure request is the exception, and it must reach every copy.

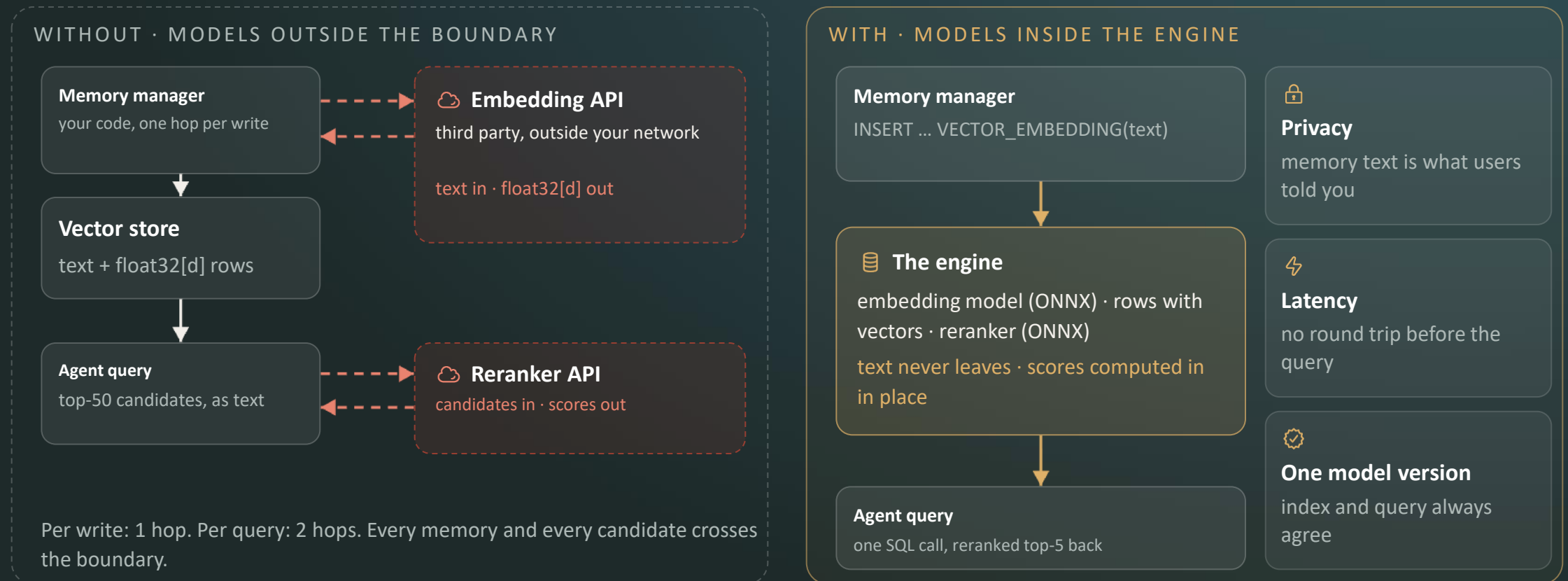
An audit trail under the user's identity

Who recalled what, and on whose behalf. Agents will be asked to explain themselves; the memory store has to be able to answer.

Personal, shared and global memories live in the same store. **The policy has to live there too.**

REQUIREMENT 5 · THE MODELS

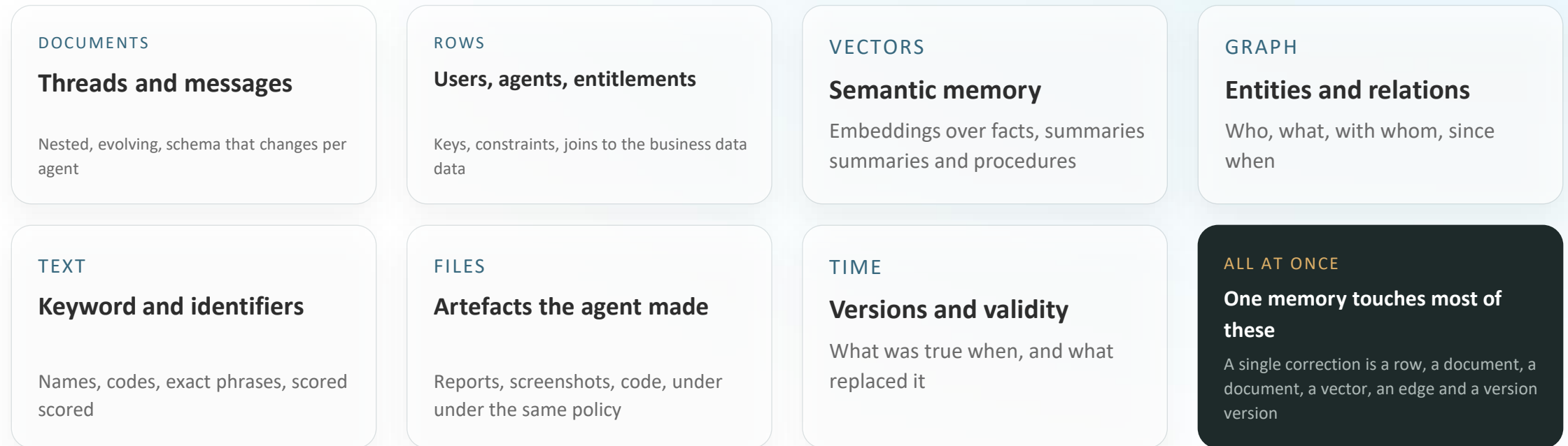
Embedding and reranking must be able to run inside the boundary



Not every workload needs the models inside. The regulated and latency-sensitive ones do, and the database should make it possible.

Memory is multi-model by nature

Map the taxonomy onto data shapes and every one of these appears.



A database that holds some of these natively and bolts on the rest will push the joins into your application. **The requirement is all of them, queryable together.**

Read-heavy and regulated workloads add two needs of their own

Caching

THE NEED · HOT RECALL, SESSION STATE

Session state and hot recall are read far more often than written; fleets ask the same top-k for the same user again and again.

A cache bolted on in front is one more copy to keep consistent; a stale cache is stale recall, only faster.

ON ORACLE · TRUE CACHE

In-memory, consistent, automatically managed SQL cache in front of the front of the primary. Read-only and kept current by redo apply, so it so it returns only committed data: consistent by construction.

Same SQL, same driver; JDBC Thin routes read-only queries to it. Session state Session state and the semantic cache stay as JSON rows in the engine.

Enterprise Edition, and one Free True Cache per Free primary. Needs a second instance, so not in today's notebook.

Auditability

THE NEED · WHO READ WHAT, AND WHEN

Every memory read and write must be attributable to an end user and an agent, and a regulator will ask what the agent knew when it acted.

An audit log kept per store, in application code, is the bolt-on tax again: N places to keep complete.

ON ORACLE · UNIFIED AUDITING + FLASHBACK

One audit trail for the whole engine: an audit policy on the memories table records reads and writes with the database user, the client identifier set for the end user, the SQL and the time, in UNIFIED_AUDIT_TRAIL.

Flashback answers the second question: AS OF TIMESTAMP reads the memory as it was when the agent acted.

In today's notebook: item 14 audits the recall, item 10 reads the store as of a timestamp.

Neither is on the twelve-question checklist. If your workload is read-heavy or regulated, score them too.

Not a vendor list: the research and the database community got there first

1 • Recall

Hybrid retrieval plus reranking cut retrieval failures from 35% to 11% on LoCoMo (ICLR 2026). The 2026 memory survey names lexical, semantic, graph and hybrid as the strategies; Zep and HippoRAG 2 fuse them.

2 • One engine

Stonebraker and Pavlo: a vector database is a second copy of the data (2025). Berkeley at CIDR 2026: a separate memory store drifts out of sync. turbopuffer: the vector index is one secondary index among many.

3 • Consistency

MemTX: stage memory writes in snapshot-isolated transactions (2026). Concordia: every memory operation runs in a transaction with transaction with policy checks before commit. Berkeley: multi-agent, multi-version isolation.

4 • Governance

MemOS: permissions, TTL, audit log and a version chain on every memory unit. Zep and Mem0 invalidate rather than delete. Always-On Agents: erasure must reach summaries and embeddings too.

5 • The models

Zep's cross-encoder reranker is its most accurate and most expensive stage; Mem0 documents a local reranker as "no external API calls". The in-engine case rests on hops and data boundaries, and we say so.

6 • Shapes

VLDB 2026 tutorial: memory spans relational, graph and files, a data-management problem. Gartner: 40% of data-management spend on multistructured data by 2027. Governed Memory: open facts plus typed properties in one record.

THE CHECKLIST

Is this database ready for agent memory? Twelve boxes to tick.

RECALL

01 ☐ Can it run vector search with an ANN index?

02 ☐ Does lexical search return relevance scores?

03 ☐ Can it fuse hybrid search inside one query?

04 ☐ Are filters applied inside the vector search?

05 ☐ Can a graph query traverse the same rows?

CONSISTENCY

06 ☐ Multi-record ACID with read-your-writes?

07 ☐ Point-in-time reads and versions, not deletes?

SHAPES

08 ☐ JSON documents and rows in one engine, joinable?

09 ☐ Files kept under the same transaction and policy?

MODELS

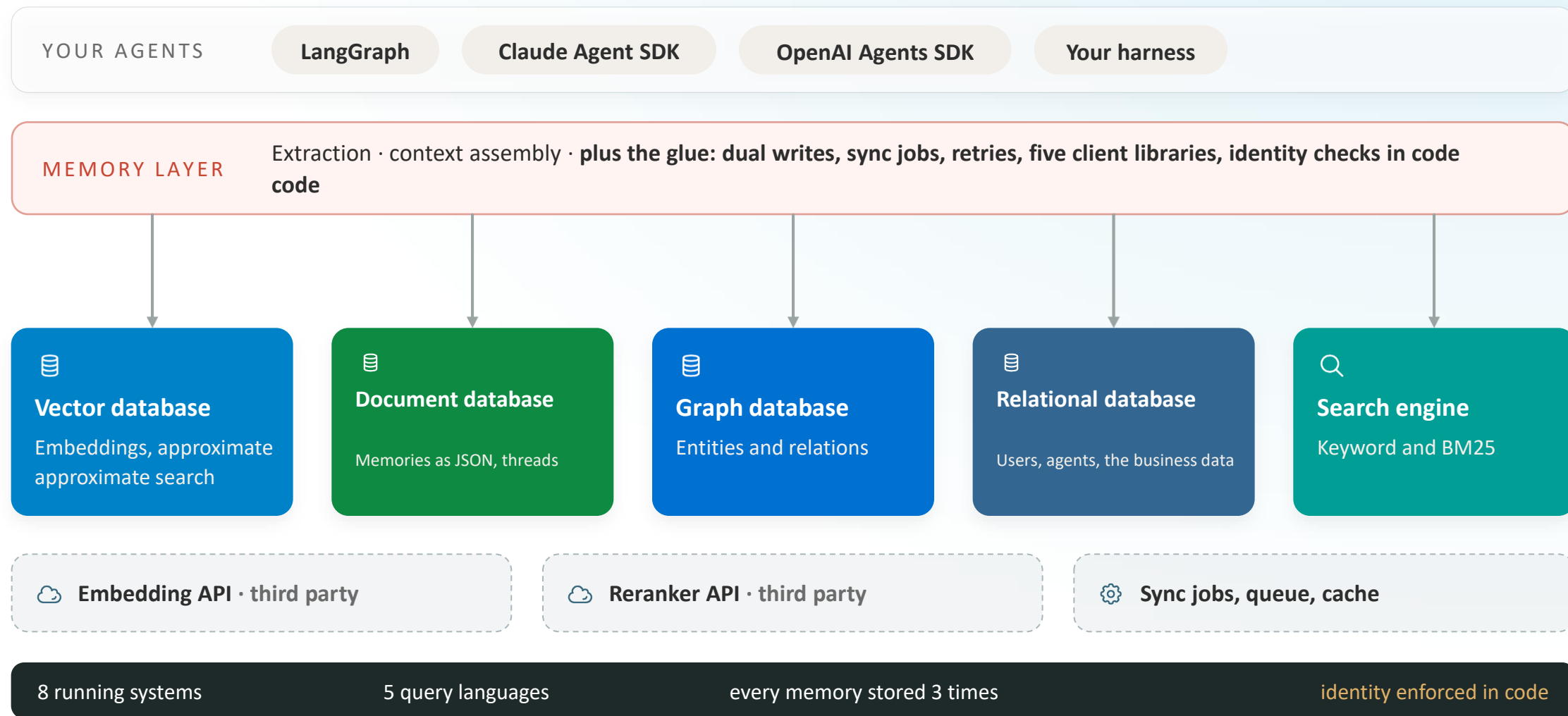
10 ☐ Can an embedding model run inside the engine?

11 ☐ Can reranking run inside the boundary?

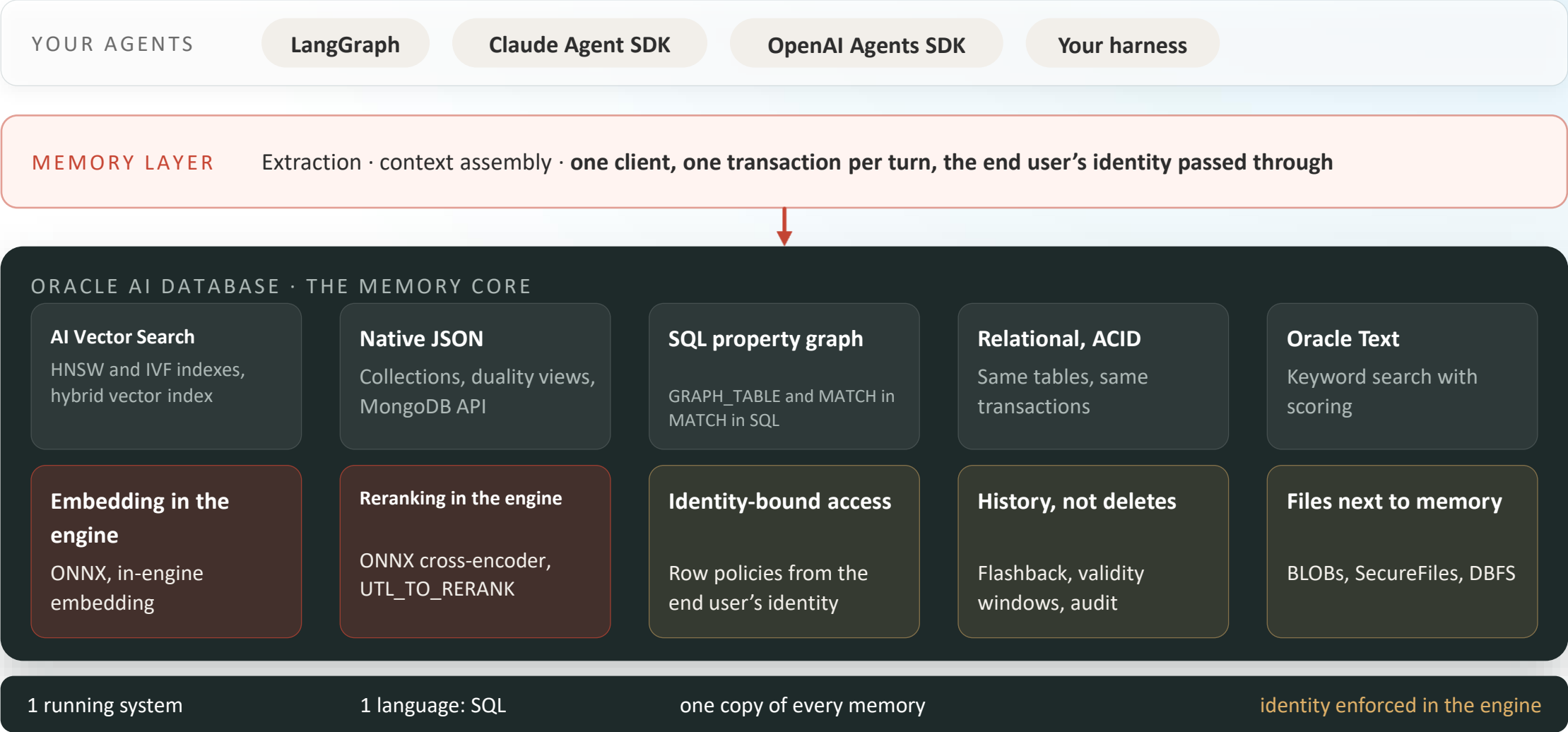
GOVERNANCE

12 ☐ Row access bound to the end user's identity, audited?

Reference architecture: memory on a polyglot stack



Reference architecture: memory on a unified engine





THE SAME CHECKLIST, ONE ENGINE

Oracle AI Database

Twelve requirements, one SQL engine, no bolt-ons

THE SCORECARD

Twelve questions, twelve boxes ticked, one engine

01	☒ Vector search, approximate index VECTOR type · HNSW, IVF · FETCH APPROX	Native
02	☒ Lexical search with scoring Oracle Text · CONTAINS, SCORE	Native
03	☒ Hybrid, fused in one query HYBRID VECTOR INDEX · RSF, RRF, WRRF	Native
04	☒ Filters inside the vector search WHERE + pre-filter, in-filter · included columns	Native
05	☒ Graph query over the same rows CREATE PROPERTY GRAPH · GRAPH_TABLE, MATCH	Native
06	☒ Multi-record ACID, read-your-writes MVCC · readers never block writers	Native
07	☒ Point-in-time reads, versions AS OF TIMESTAMP · Flashback Time Travel	Native
08	☒ JSON and rows, one engine, joinable JSON type · Duality Views · MongoDB API	Native
09	☒ Files under the same transaction SecureFiles LOBs · DBFS	Native
10	☒ Embedding model in the engine LOAD_ONNX_MODEL · VECTOR_EMBEDDING	Native
11	☒ Reranking inside the boundary ONNX reranker · UTL_TO_RERANK · PAISC	Native
12	☒ Row access by end-user identity, audited Deep Data Security · Unified Auditing	Native

Every row runs in Oracle AI Database 26ai Free. The in-kernel reranker needs its ONNX model exported once with OML4Py; the notebook says which route it says which route it used.

✓ CHECKLIST 04 · FILTERS INSIDE THE SEARCH

Post-filter, pre-filter, in-filter: where the predicate runs decides what the agent sees

POST-FILTER · WITHOUT

top k → then filter

Search first, filter afterwards. The agent for a user with few memories gets a near-empty context, because the top k barely contained them.

What application code and many vector stores do. Cheap, and wrong at the edges.

PRE-FILTER · WITH

filter → then search

Evaluate the predicate first, then search only only the rows that survive. Exact, and the right right choice when the predicate is selective: selective: one user, one memory type.

Plan: VECTOR INDEX HNSW SCAN PRE-FILTER

IN-FILTER · WITH

search + filter as you go

Apply the predicate while traversing the graph index, skipping candidates that fail it as they are met. The right choice when the predicate is broad.

Plan: VECTOR INDEX HNSW SCAN IN-FILTER



The predicate is ordinary SQL in the same WHERE as the vector order. The optimiser picks pre-filter or in-filter per query, and the hybrid vector index carries FILTER BY columns for the same reason.

Embedding and reranking run in the kernel as ONNX models

```
-- load an ONNX embedding model into the database
EXEC DBMS_VECTOR.LOAD_ONNX_MODEL('MODELS',
  'all_MiniLM_L12_v2.onnx', 'MINILM');
-- embed where the memory is written; no text leaves
UPDATE memories SET embedding =
  VECTOR_EMBEDDING(MINILM USING content AS data);
-- rerank the candidates with an ONNX cross-encoder
SELECT DBMS_VECTOR_CHAIN.UTL_TO_RERANK(
  :question, :candidates,
  json('{"provider":"database",
    "model":"BGE_RERANKER","top_n":5}'))
FROM dual;
```

IN THE KERNEL

Embedding and reranking models in the database process

Oracle ships pre-built embedding models; export your own, including a cross-encoder cross-encoder reranker, with the OML4Py ONNX pipeline. Works on Free.

IN THE BOUNDARY

Private AI Services Container

Larger embedding, reranking and LLM services on your own hardware, called hardware, called from the same SQL functions, so bigger models still never leave your network.



Write path, query path and rerank path stay where the data lives. For a regulated workload that is the difference between a memory store and a data transfer.

UTL_TO_RERANK also takes Cohere, Vertex AI and OCI Generative AI as providers when you want a hosted model instead.

One engine, every shape: the converged database applied to memory

DOCUMENTS AND ROWS

JSON Relational Duality

The same memory is a JSON document to the agent and relational rows to the policy engine and the analysts. One copy, ACID in both directions.

DOCUMENT DEVELOPERS

Oracle Database API for MongoDB

Point an existing MongoDB driver at the database. The database. The collection you write is a table the SQL engine can join, index, secure and audit.

ENTITIES AND RELATIONS

SQL property graphs

CREATE PROPERTY GRAPH over the memory and entity tables, then MATCH patterns with GRAPH_TABLE inside an ordinary SQL query, next to next to the vector predicate.

CONSISTENCY

ACID transactions and Flashback

The three-write correction commits or rolls back as one. SELECT AS OF TIMESTAMP reads the memory memory store as it was before, with nothing deleted. deleted.

FILES

Artefacts next to memory

SecureFiles LOBs hold the report or screenshot the the agent produced, in the same transaction and under the same policy; DBFS mounts the same same storage as a file system.

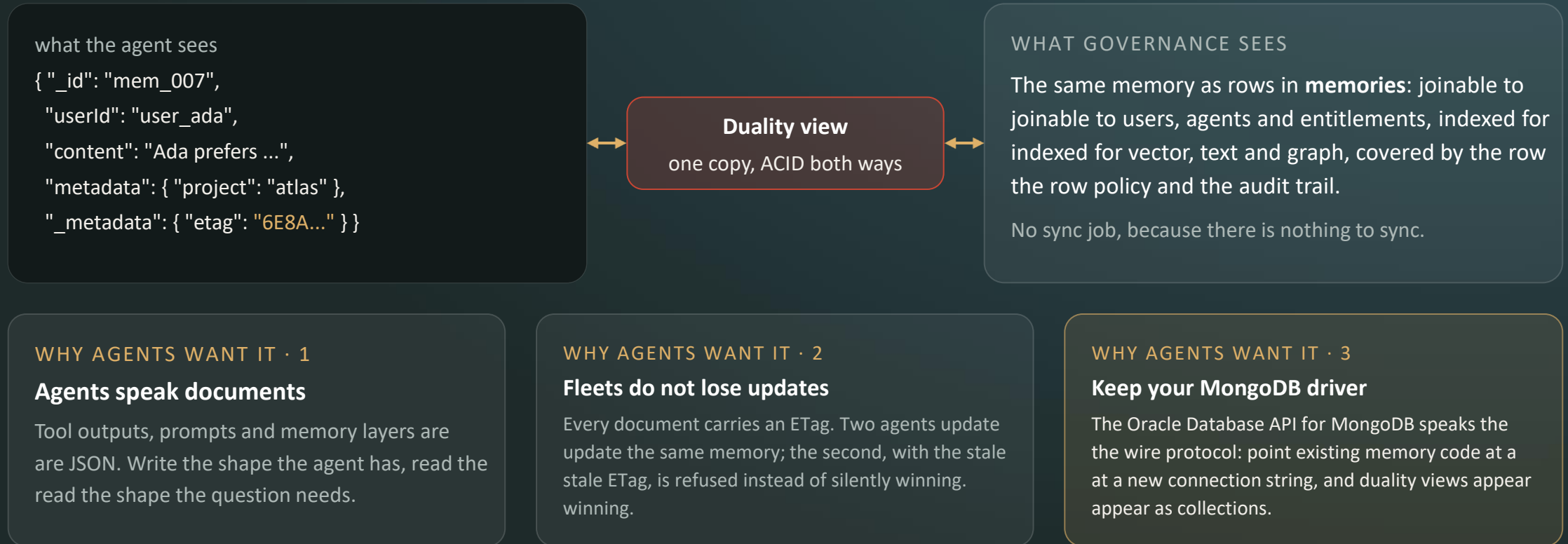
RECALL

Vector, text and hybrid in one index

AI Vector Search with HNSW or IVF, Oracle Text, and Text, and a hybrid vector index that answers both both and fuses the scores in one query.

Every card is a data type or an index on the same tables. **One memory, one copy, every shape.**

JSON Relational Duality: the agent writes documents, governance reads rows, one copy



Duality views and the MongoDB API are both in Oracle AI Database Free; the MongoDB API is served by ORDS, which ships with Free.

Deep Data Security: the end user's identity decides which rows exist

WHAT IT DOES

- The user's identity reaches the database as an OAuth token on every token on every call
- Declarative data grants: row, column and cell, in SQL
- Predicates read the verified end-user context, not a session variable the app set
- Denied rows are absent from recall, not filtered afterwards
- The agent's connection needs only CREATE SESSION
- Same grants for apps, SQL clients, REST, MCP and agents
- Every access lands in Unified Auditing under the user's identity

THE POLICY, ONCE

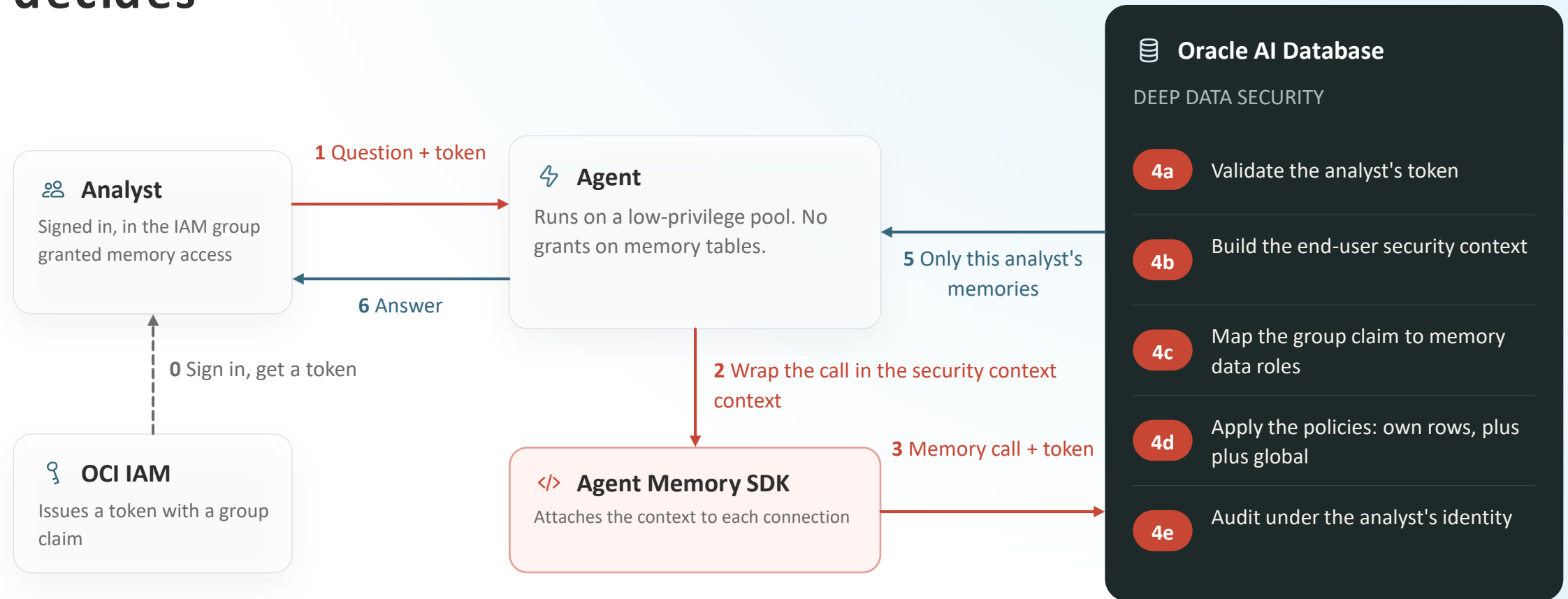
```
CREATE DATA ROLE memory_reader_role;  
CREATE DATA GRANT own_memories AS SELECT ON memories  
  WHERE user_id = LOWER(ORA_END_USER_CONTEXT.username)  
  OR user_id = 'global' TO memory_reader_role;  
SET USE DATA GRANTS ONLY ON memories ENABLED;
```

WHAT USER A GETS BACK

- Their own memories, plus global ones
- Nothing from user B, even a perfect semantic match
- Same SQL as user B, two different result sets
- Oracle AI Agent Memory 26.8 enforces this on its own tables

Shipped in the April 2026 release update of Oracle AI Database 26ai; included in Free, Standard and Enterprise editions. The notebook runs it on the free container with local container with local end users logging on directly, no identity provider needed. VPD and Real Application Security remain supported.

Checklist 12 in practice: identity flows into memory, the database decides



The agent never holds anyone's memories. It carries their identity, and the database decides.

THE SAME CHECKLIST ACROSS THE FIELD

Community and open-source editions, scored with the same four answers

Checklist question	PostgreSQL	MongoDB	Redis 8	Elasticsearch	Oracle AI Database
01 Vector search, ANN index	Extension: pgvector	Separate: mongot	Native	Native	Native
02 Lexical search, scored	Native, no BM25	Separate: mongot	Native	Native	Native
03 Hybrid fused in one query	App code: RRF SQL	Separate: mongot	Native: FT.HYBRID	Paid tier	Native
04 Filters inside vector search	Extension, post-filter	Separate: mongot	Native	Native	Native
05 Graph query, same rows	Extension: AGE	Partial: lookup	Not available	Not available	Native
06 Multi-record ACID	Native	Native, index lags	No rollback	Not available	Native
07 Point-in-time reads	App code	App code	Not available	App code	Native
08 JSON and rows, joinable	Native	Partial: lookup	Partial: no joins	Partial: lookup	Native
09 Files in the engine	Native	Partial: GridFS	Partial: in RAM	Not available	Native
10 Embedding in the engine	Not available	Separate: Atlas	Not available	Paid tier	Native
11 Reranking in the boundary	Not available	Atlas only	Not available	Paid tier	Native
12 Row access by identity	Native	Not available	Not available	Paid tier	Native

Community and open-source editions as of October 2026: PostgreSQL 18 with pgvector 0.8.7, MongoDB 9.0 with mongot, Redis 8.10, Elasticsearch 9.5, Oracle AI Database 26ai Free. Sources in the notes.

Where each engine is strong, and where memory breaks it

PostgreSQL

STRONG AT

Ubiquity, SQL, ACID, read-your-
your-writes, native row-level
security policies, and pgvector,
mature and shipped by every
managed provider.

WHERE IT BREAKS

BM25 and graph are extensions that
RDS, Cloud SQL, Neon and Supabase
do not ship, hybrid fusion is SQL you
write, no maintained in-database
embedding exists since pgai was
archived, and there is no point-in-
time read.

MongoDB

STRONG AT

Document ergonomics for threads
and messages; text, vector and rank
fusion free on Community since July
2026; transactions on a replica set;
first-party Voyage models.

WHERE IT BREAKS

Search is a second process you run
run and secure yourself; automated
automated embeddings and rerank
rerank are Atlas-hosted services;
services; graph is a bounded lookup;
lookup; access control stops at the
the collection.

Redis

STRONG AT

Latency. Working memory, session
state, TTL decay, semantic caching,
and since 8.4 in-engine hybrid search
with rank fusion, all open source.

WHERE IT BREAKS

Memory-resident economics for
for long-term memory, durability by
durability by configuration (up to a
to a second of loss by default), no
no graph since RedisGraph's end of
end of life, no rollback in MULTI and
and EXEC, key-level access control.
control.

Elasticsearch

STRONG AT

The strongest retrieval engine of
the four: BM25, dense and sparse
vectors, rank-fusion retrievers, and
embedding and reranking models
on the cluster's own nodes.

WHERE IT BREAKS

No multi-document transactions,
transactions, near-real-time refresh
refresh on the write path, no
recursive traversal, and rank fusion,
fusion, in-cluster models and
document-level security sit in the
the Enterprise tier.

Each was built for a workload it still serves well. **Agent memory is the first workload that needs all of them at once.**



DEMO
O

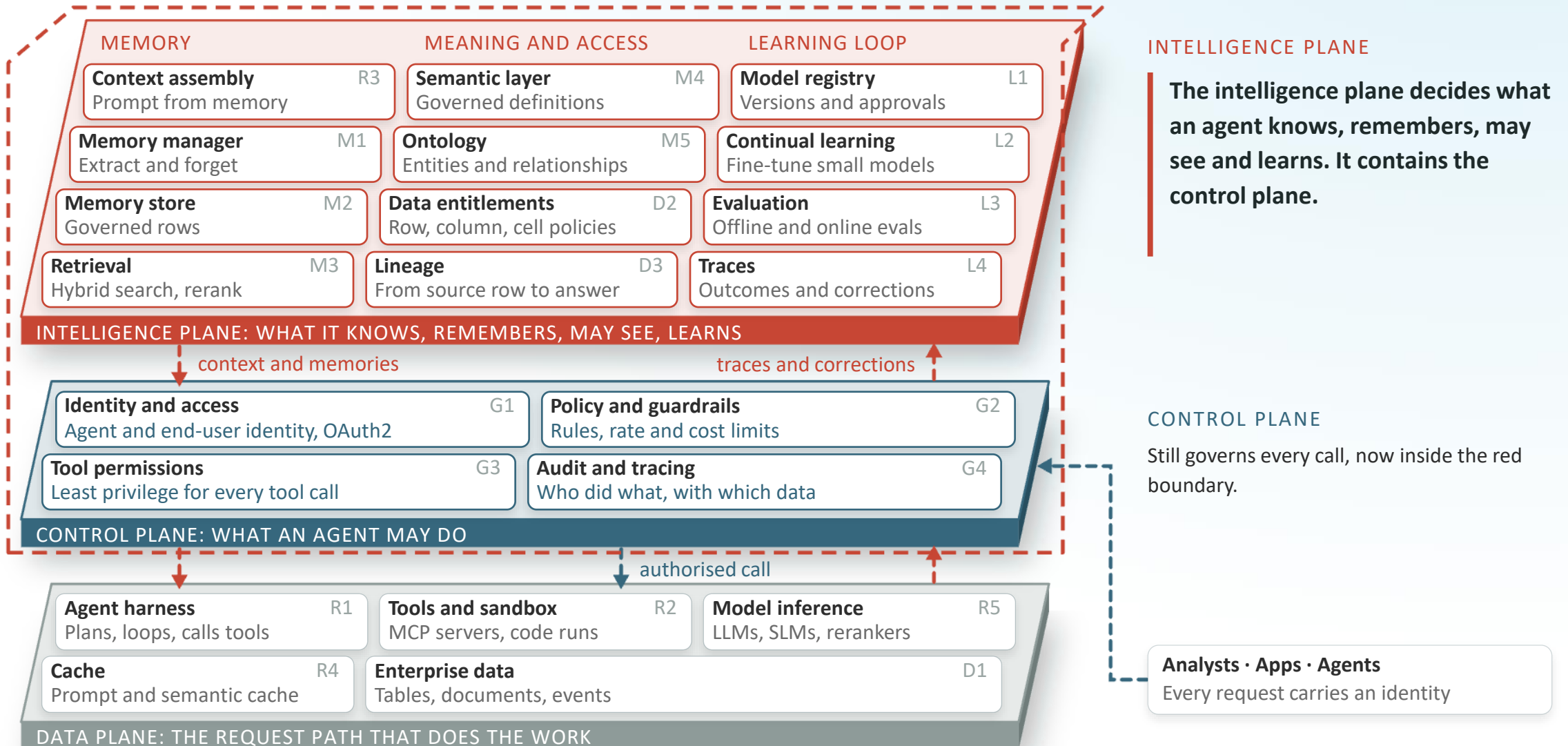
The checklist, live

One notebook, one free container, every requirement proven in SQL from Python
Python

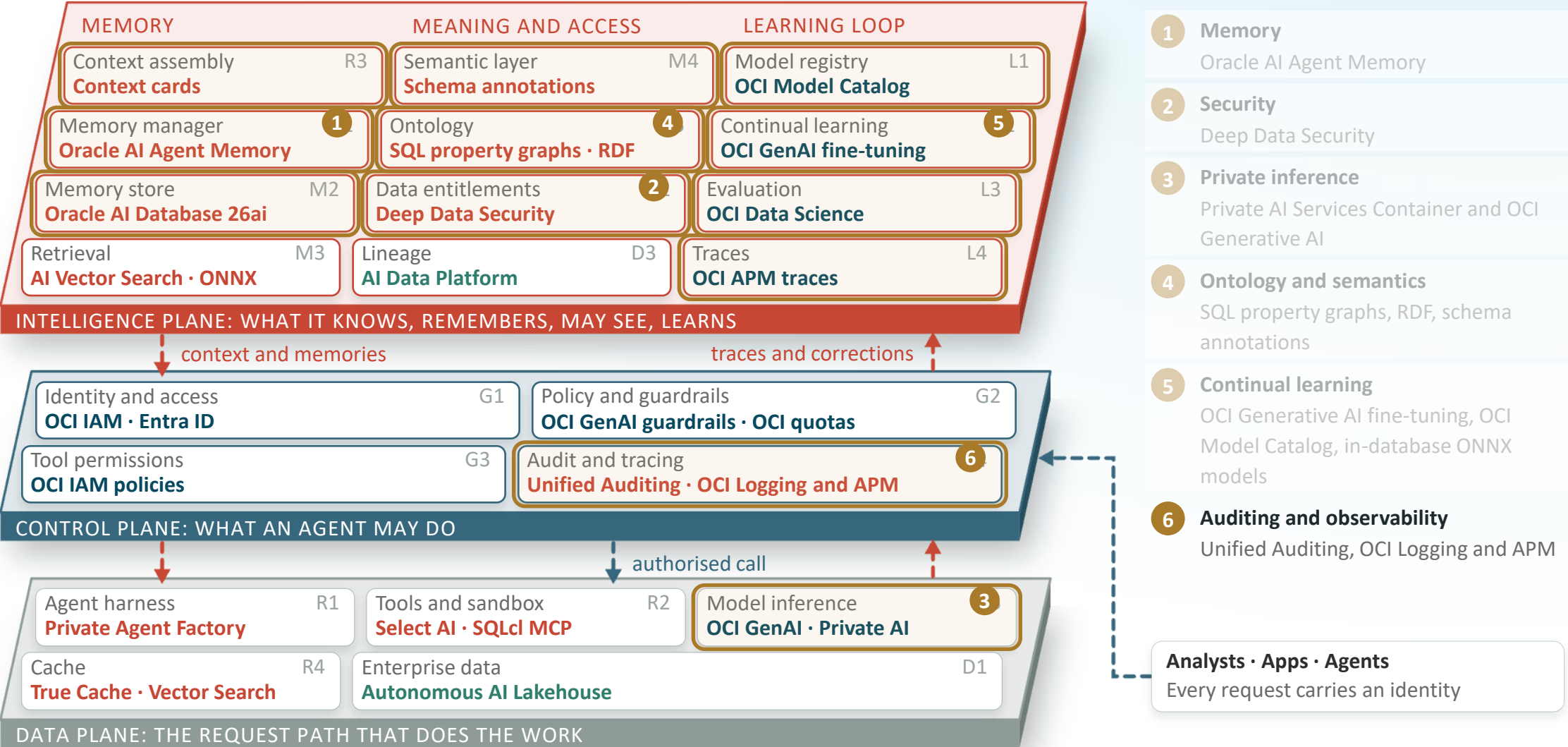
[docker compose up oracle](#)

[01_the_checklist_on_oracle.ipynb](#)

What's next: memory is one plane of a larger picture, the intelligence plane



Ground your intelligence plane in one unified, secure, trusted foundation



Memory is the only moat. You should own it.

Own it in a database that can carry the whole workload: every retrieval strategy, every transaction, every identity, every version. In one engine, with your data.

Where to go next

TRY IT

Memory engineering in a box

Oracle AI Agent Memory on Oracle AI Database Free, in a container on your laptop.

`pip install oracleagentmemory`

LEARN IT

Agent Memory: Building Memory-Aware Agents

The DeepLearning.AI short course, built with with Oracle.

deeplearning.ai

REPRODUCE IT

Today's notebook

The checklist on Oracle, item by item. The four-engine comparison is in the same repository.

[repository link]

CONNECT

Richmond Alake

linkedin.com/in/richmondalake



ORACLE Developer *days*

A developer-first enablement program for
Enterprise customers

