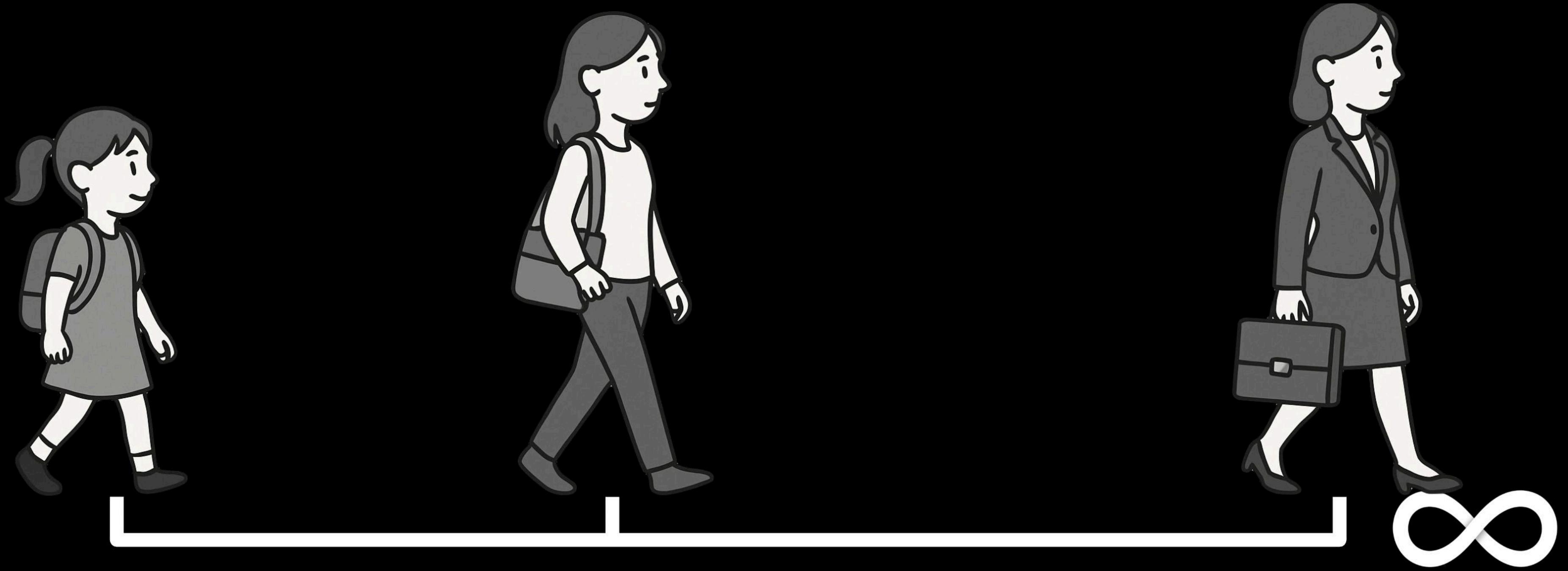


Database for Agents Memory, The Right Way

Ed Huang (黃東旭)

Cofounder / CTO, PingCAP



Elementary
School

LLM
(Instinct)

University

RAG-based
(Knowledge)

Professional
Experience

Agent with Memory
(Task)

Why Memory Matters in AI Agents

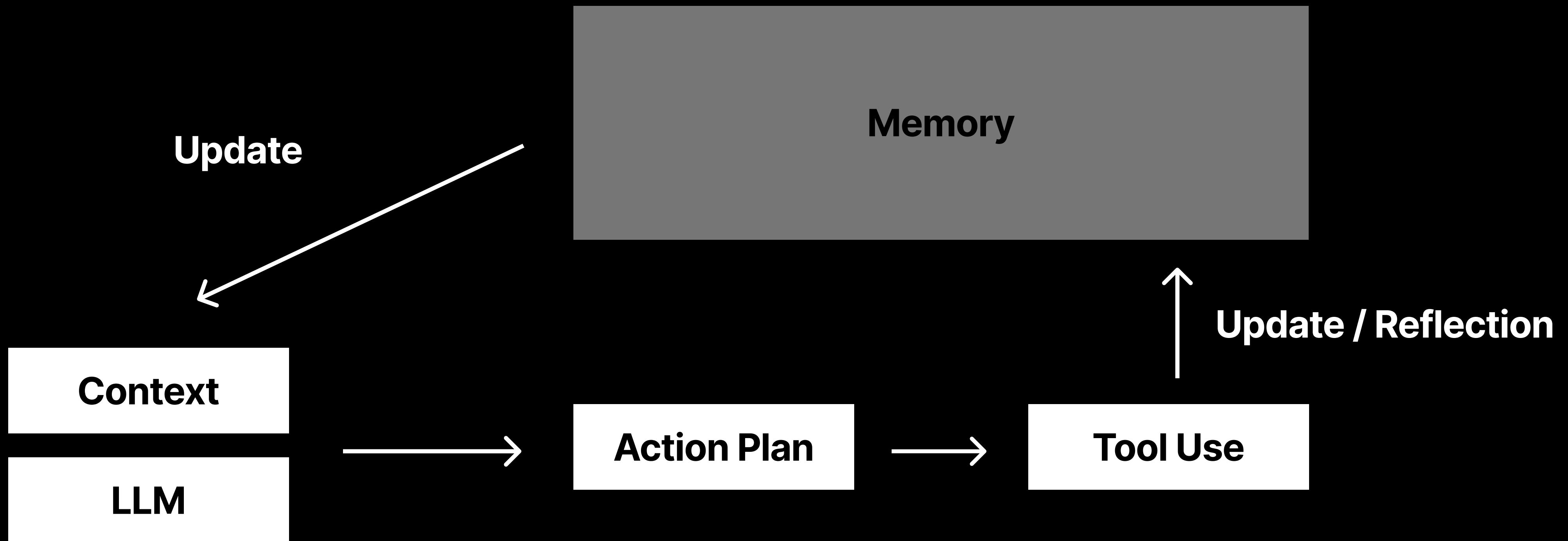
- Today's LLM is good enough to build something useful.
- Agents aren't just stateless chatbots anymore

Agent needs:

working memory (context window)

long-term memory (persistent + evolving)

Typical AI Agent architecture today



give me the top 10 most hardworking sales

Thought for 0.77 seconds

To determine the "most hardworking sales," I need to define a metric. A common approach is to count the number of events (meetings, calls, etc.) each sales person (owner_id or owner_name) has. I will query the top 10 owner_name by the count of their events.

↗ tool-tidb · db_query ✓ Success ⚙ Re-run

Here are the top 10 most hardworking salespeople based on the number of events they have in the system:

Rank	Salesperson	Events
1.	[REDACTED]	- 71 events
2.	[REDACTED]	55 events
3.	Debray	- 44 events
4.	[REDACTED]	- 42 events
5.	[REDACTED]	ents
6.	[REDACTED]	41 events
7.	[REDACTED]	- 41 events
8.	[REDACTED]	s - 36 events
9.	[REDACTED]	rabhatta - 31 events

Enter a message here, press ↲ to send

My home made agent

**Give me insights about
our sales team.**

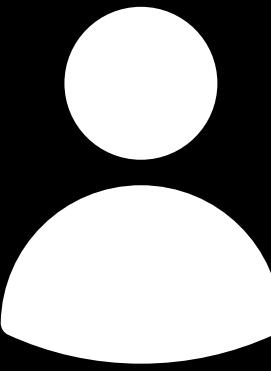
But wait, Memory ≠ RAG

RAG: query a passive knowledge base

Memory: updateable, curated, opinionated knowledge

RAG = reference book, Memory = journal + life history

The problems of RAG



Context

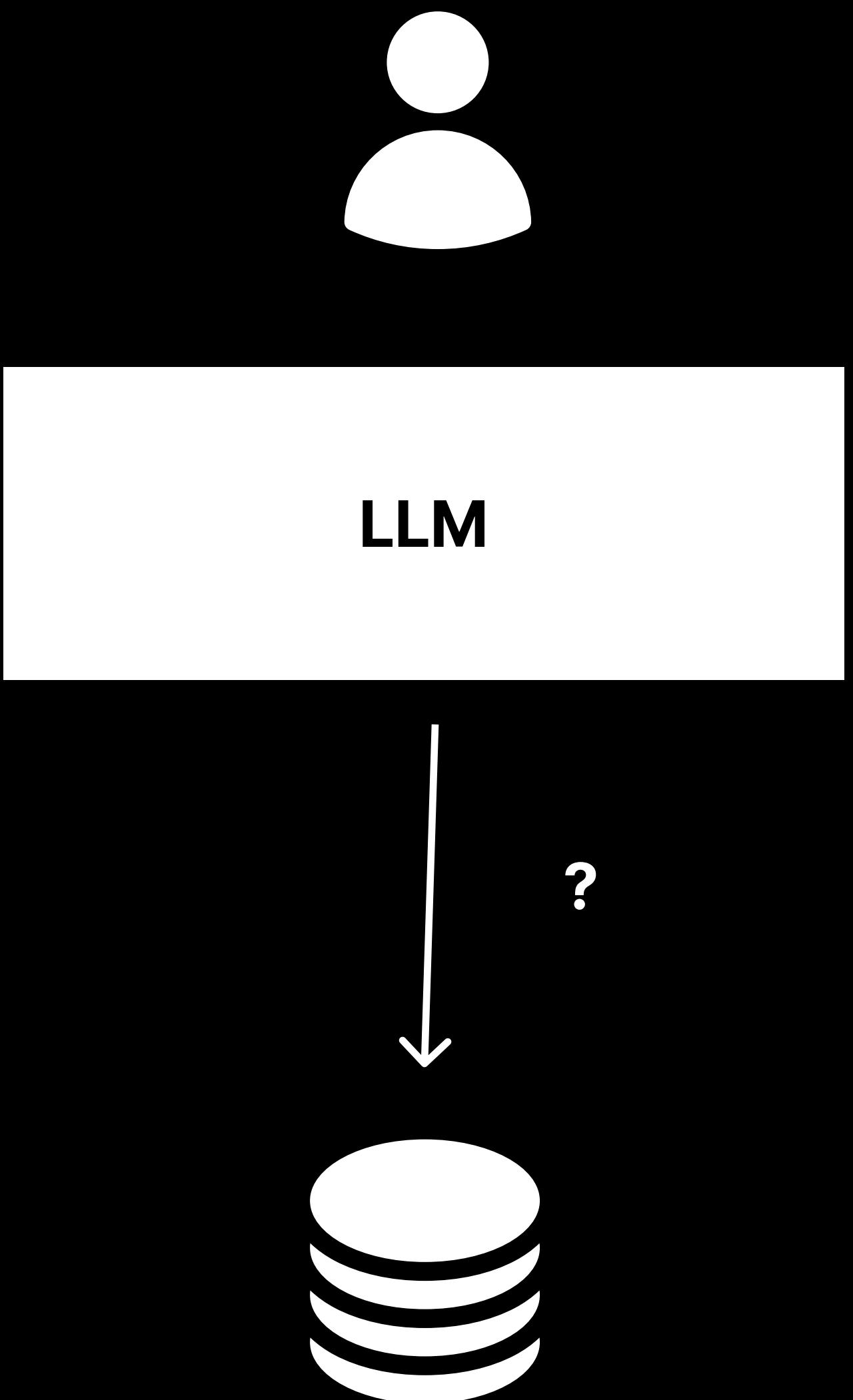


MCP



Data

If you were an LLM, what kind of data would you want access to, and how would you access it?



Full-text search

Embedding

Time-serials Events

Relational data

Document data

Graph

...

Full-text search

Embedding

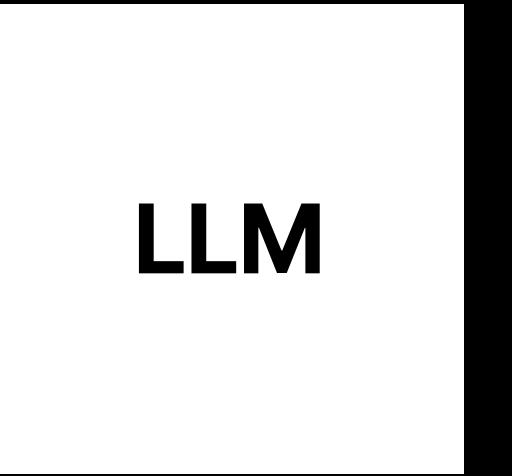
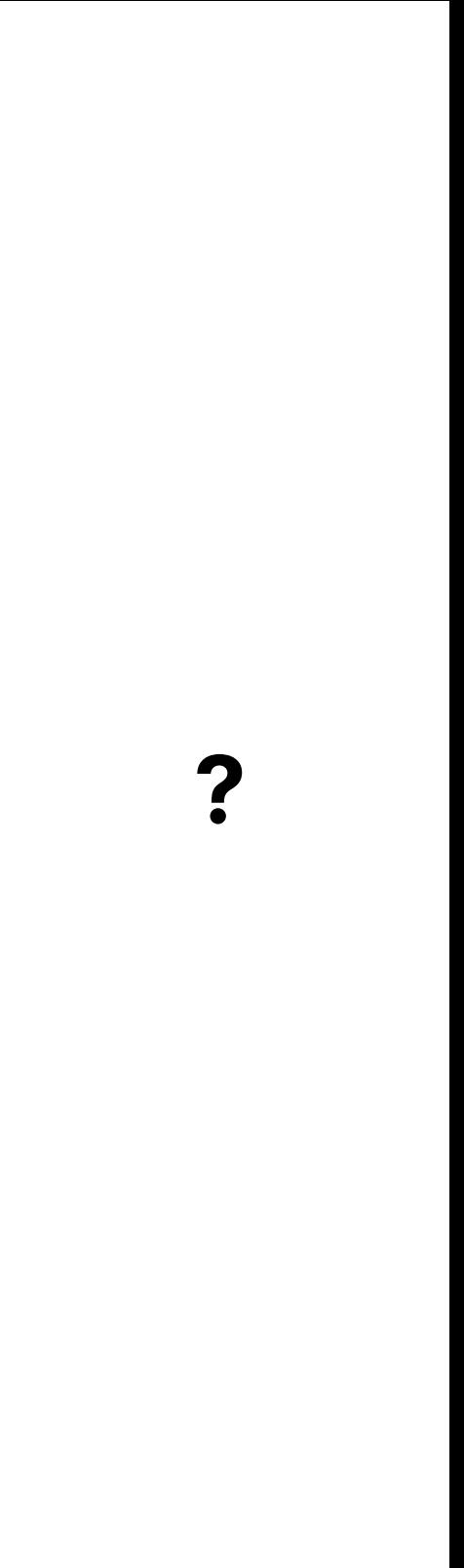
Time-serials Events

Relational data

Document data

Graph

...



Full-text search

Embedding

Time-serials Events

Relational data

Document data

Graph

...



**SQL is the best bridge
between
LLM and Data Source**

From Short-Term to Long-Term How Memory Evolve

Example

Agent hears:

“Can you pull my top 5 Salesforce opportunities this week and compare them to last month?”

Short-term memory: user intent + tool output

Long-term memory: "This user cares about revenue trend and business", “the user is now in USA”

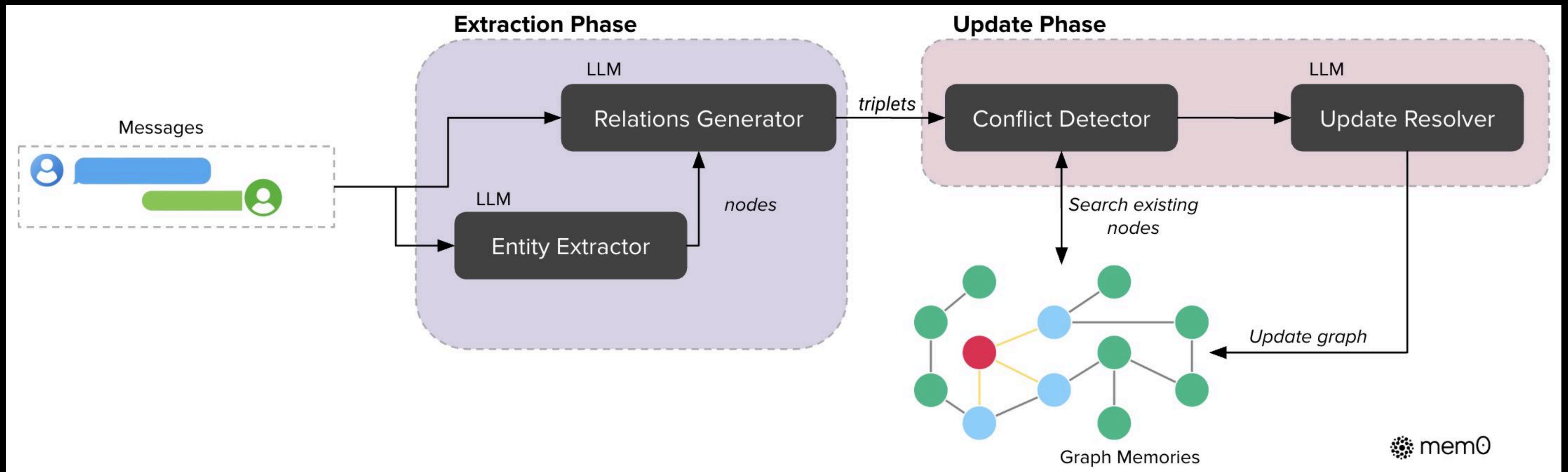
LLM decides:

generate tool use intent → fetch data from tool → analyze → remember pattern

Different types of memory

Type	Duration	Example
Tool-use	Session	“Show me Q2 revenue”
Short-term memory	Session	“I’m going to Tokyo” ... “Give me some restaurant recommendations”
Long-term memroy	Persistent	“I moved to Shanghai”

Typical architecture of memory system



Fact Extractor

```
1 CREATE TABLE memory_facts (
2     id BIGINT PRIMARY KEY AUTO_INCREMENT,
3     subject VARCHAR(255),          -- e.g., 'node-1', or 'user123'
4     predicate VARCHAR(255),        -- e.g., 'connected_to', 'caused_by'
5     object VARCHAR(255),          -- e.g., 'node-2', or 'TiDB'
6     object_embedding VARBINARY(512), -- optional, for similarity reasoning
7     weight FLOAT DEFAULT 1.0,      -- optional, confidence/importance
8     properties JSON,             -- optional, metadata (e.g., timestamps,
9     reason)                      -- for update/expiry support
10    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
11    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
12 );
13
```

You are a memory extraction assistant.

Given the user input below, extract:

1. *A list of key entities.*
2. *A list of structured facts in the format (subject, predicate, object).*

If the statement contains no meaningful fact, return empty lists.

Input:
"*{user_input}*"

Entities:

- ...

Facts:

- *(subject, predicate, object)*

**LLM Prompt for
Fact Extractor**

User Input:

“I used to live in New York, but I just moved to San Francisco for a new job at PingCAP.”

LLM Output:

Entities:

- *I*
- *New York*
- *San Francisco*
- *PingCAP*

**Example
Fact Extractor**

Facts:

- (*I*, *lived_in*, *New York*)
- (*I*, *moved_to*, *San Francisco*)
- (*I*, *works_at*, *PingCAP*)

Entity Normalization

Same real-world entity is referred to in different ways
e.g., “TiDB”, “PingCAP TiDB”, “tidb database”

Without normalization, every variation of an entity name might be stored as a new entry:

```
1 CREATE TABLE known_entities (
2     id BIGINT PRIMARY KEY, .          -- use `id` in the graph
3     canonical_name VARCHAR(255),    -- e.g., 'PingCAP TiDB'
4     aliases JSON,                  -- e.g., ['TiDB', 'pingcap tidb', 'tidb
5     database']
6     embedding VARBINARY(512)
7 );
```

```
1 SELECT *
2 FROM known_entities
3 ORDER BY cosine_distance(embedding,
4 :entity_embedding)
5 LIMIT 3;
```

Conflict detection

The user says:

“I recently moved to Shanghai.”

The system already has:

```
1 subject: user123
2 predicate: lives_in
3 object: Beijing
```

1. Find potential conflicting facts

```
1  SELECT * FROM memory_facts  
2  WHERE subject = 'user123'  
3    AND predicate = 'lives_in'  
4    AND valid = TRUE;  
5
```



```
1  [id: 1, subject: 'user123', predicate: 'lives_in',  
object: 'Beijing']  
2
```

2. Use LLM to determine conflict

Prompt:

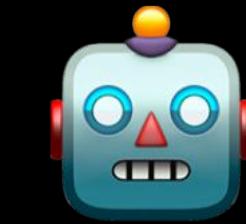
You are a memory update agent.

Existing fact:

- *The user lives in Beijing.*

New fact:

- *The user recently moved to Shanghai.*



- 1. Yes, there is a conflict.**
- 2. Keep the new fact: 'moved to Shanghai'.**
- 3. Mark the old fact as invalid.**

Please answer:

- 1. Do these facts conflict?*
- 2. If yes, which one should be kept?*
- 3. Should the old fact be marked as invalid?*

3. Update the fact database

```
1 -- Step 1: Mark the old fact as invalid
2 UPDATE memory_facts
3 SET valid = FALSE
4 WHERE id = 1;
5
6 -- Step 2: Insert the new fact
7 INSERT INTO memory_facts (subject, predicate, object,
object_embedding)
8 VALUES ('user123', 'lives_in', 'Shanghai', /* vector embedding */);
9
```

Memory Retrieval

1-hop sub graph

```
1  SELECT f.*  
2  FROM (  
3      SELECT DISTINCT subject  
4      FROM memory_facts  
5      WHERE valid = TRUE  
6      ORDER BY cosine_distance(subject_embedding, embedding('User123')) ASC  
7      LIMIT 5  
8  ) AS sim_subjects  
9  JOIN memory_facts f ON f.subject = sim_subjects.subject  
10 WHERE f.valid = TRUE;  
11
```

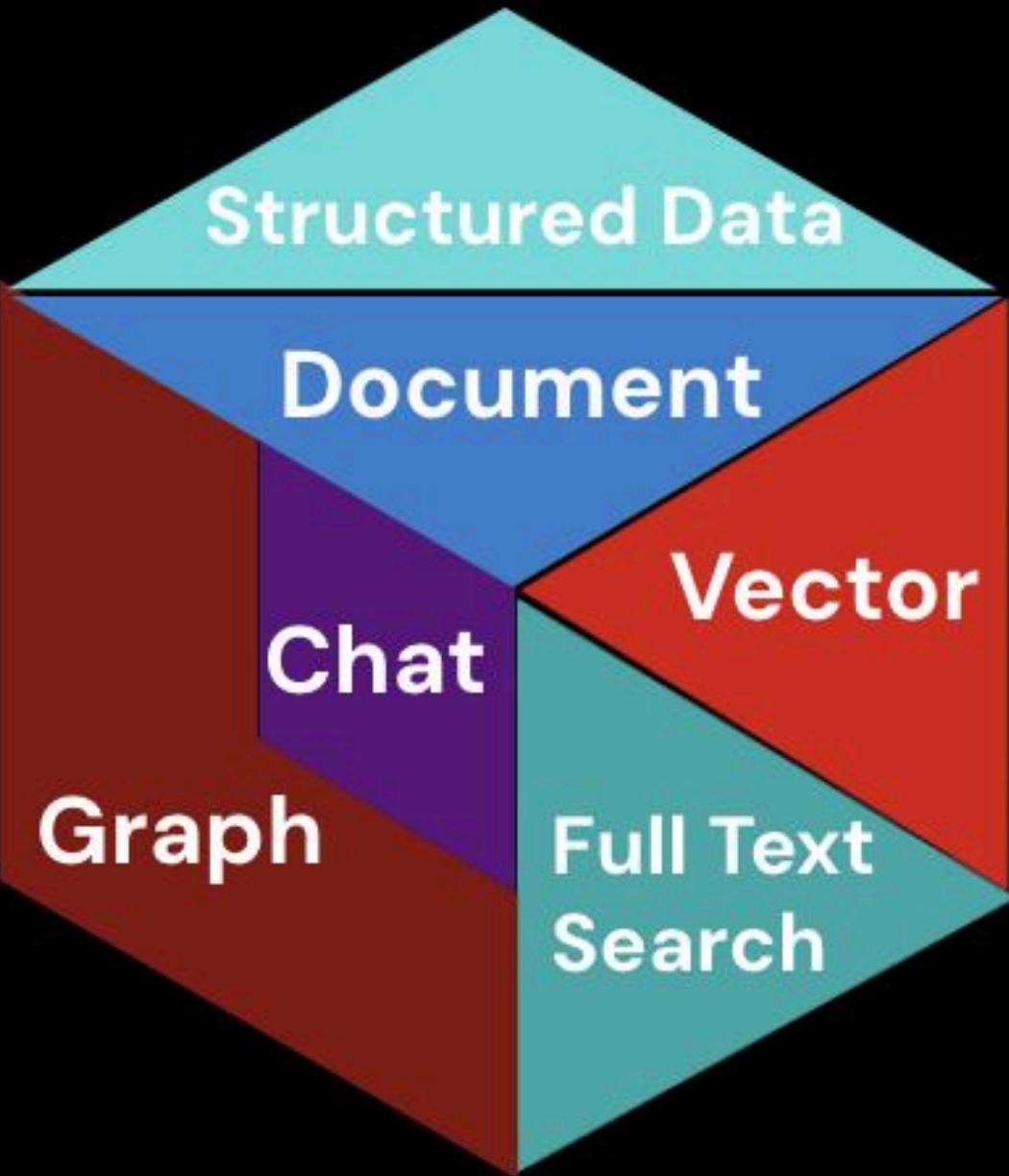
A more complicated example

```
1 -- Cosine similarity + 2-hop path in one query
2 SELECT f2.subject AS hop1, f2.predicate AS hop_pred, f2.object AS hop2
3 FROM (
4   SELECT DISTINCT subject
5     FROM memory_facts
6   WHERE valid = TRUE
7   ORDER BY cosine_distance(subject_embedding, embedding('User123')) ASC
8   LIMIT 5
9 ) AS sim_subjects
10 JOIN memory_facts f1 ON f1.subject = sim_subjects.subject
11 JOIN memory_facts f2 ON f1.object = f2.subject
12 WHERE f1.valid = TRUE AND f2.valid = TRUE;
13
```

Takeaways

- Memory \neq RAG
- SQL is all you need, vector database is just not enough
- Embeddings + SQL unlock soft reasoning
- Short-term actions deserve long-term memory via Reflection
- When you don't know what to do — call the LLM as a function

TiDB



TiDB Serverless Clusters

> 120,000 Active Clusters

\$ pip install pytidb

<https://tidb.cloud>

Thanks