

Data Management for Agentic Memory: Foundations, Systems, and Challenges

Guoliang Li

Tsinghua University



清華大學

Tsinghua University

Jiaqi Tian

Tsinghua University



清華大學

Tsinghua University

Xuanhe Zhou

Shanghai Jiaotong University

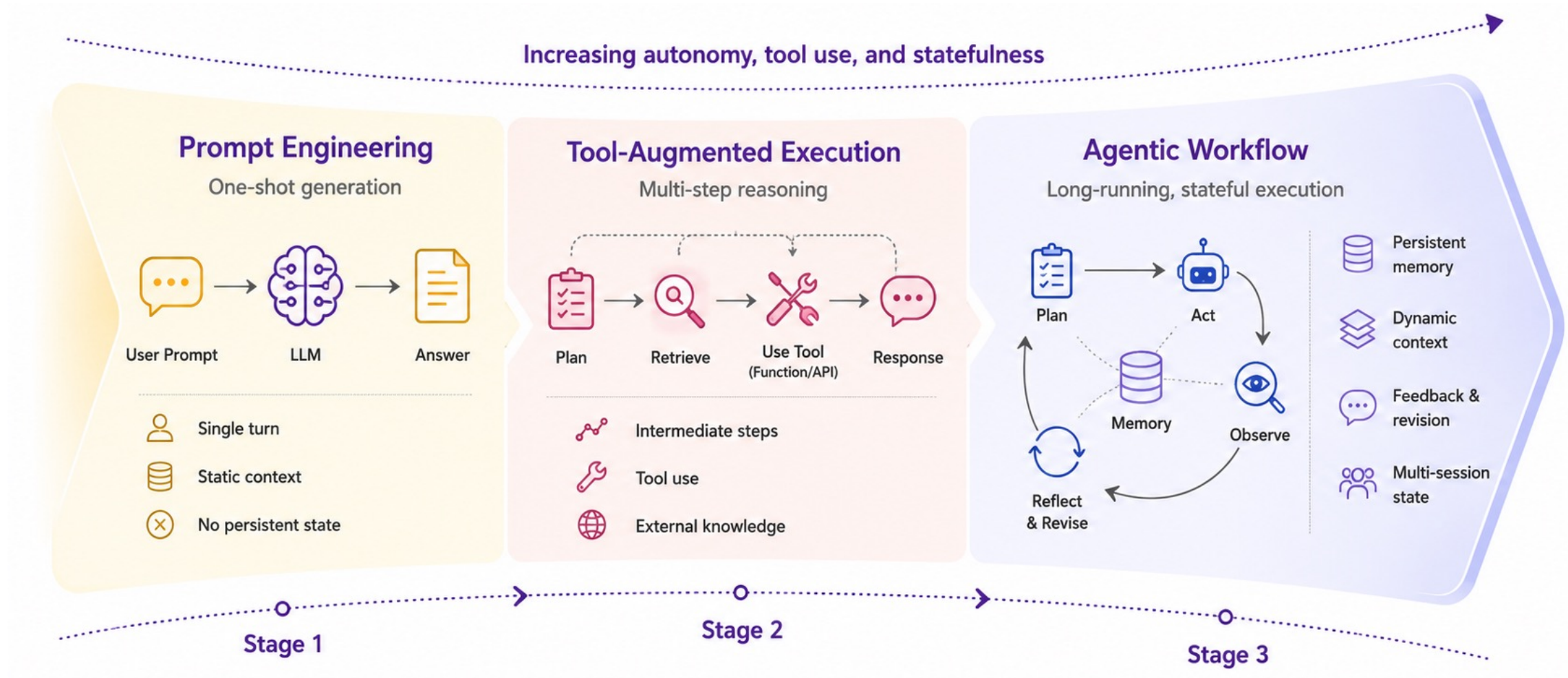


上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

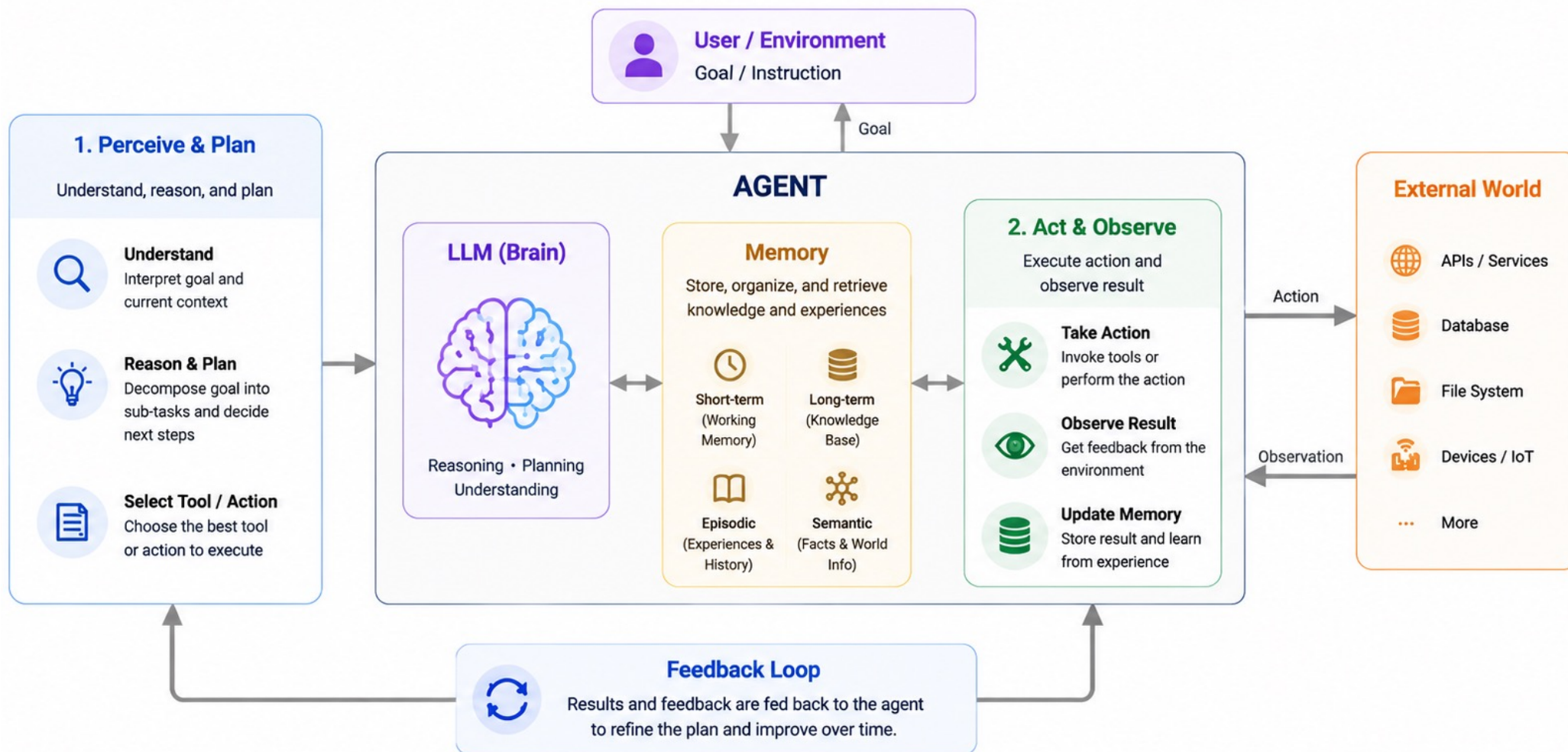
The Shift from Generative AI to Agentic AI

LLM applications move from **one-shot generation** (*Prompt Engineering*) to **long-running, stateful execution** (*Agentic Workflow*).



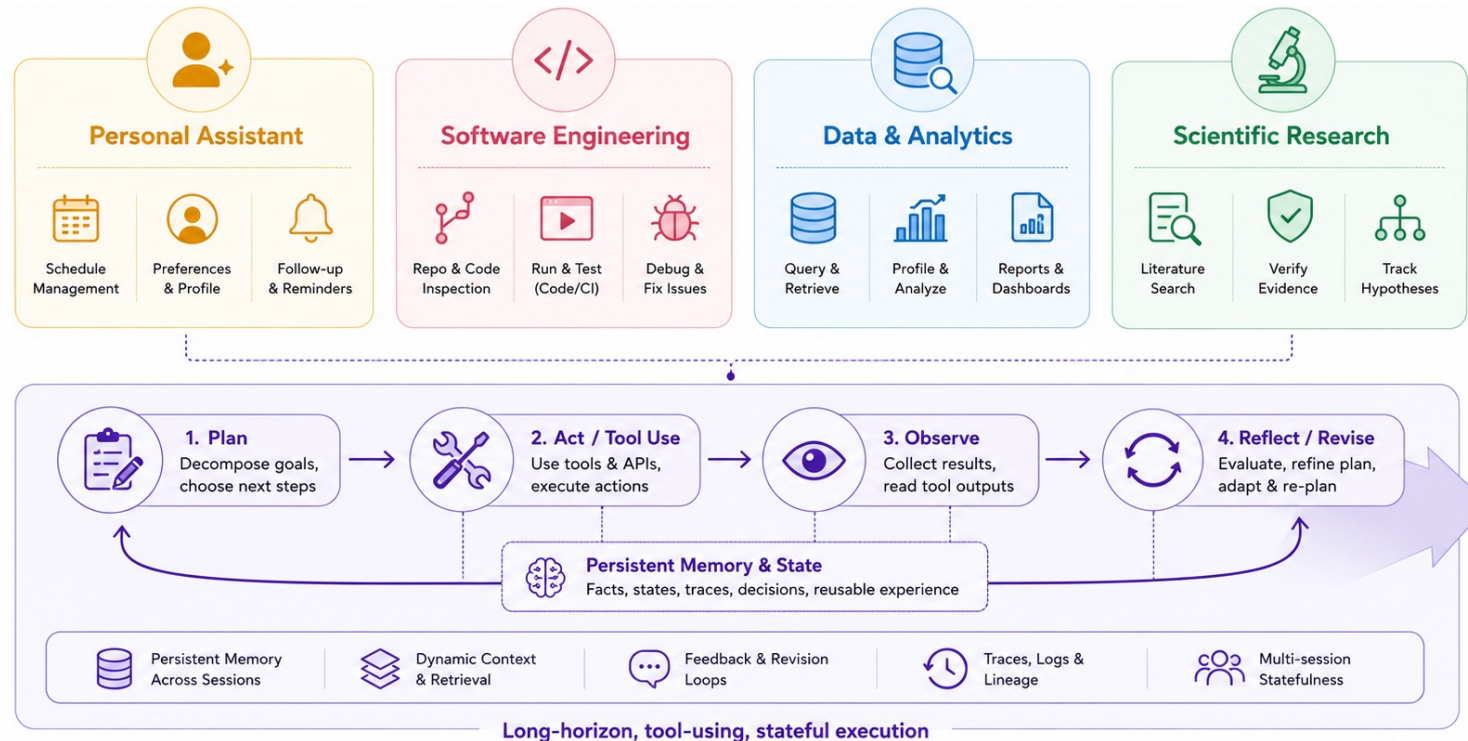
LLM Agents

An agent turns an LLM's intelligence into autonomous action with planning, **memory**, tools, and feedback.



What Modern Agents Do?

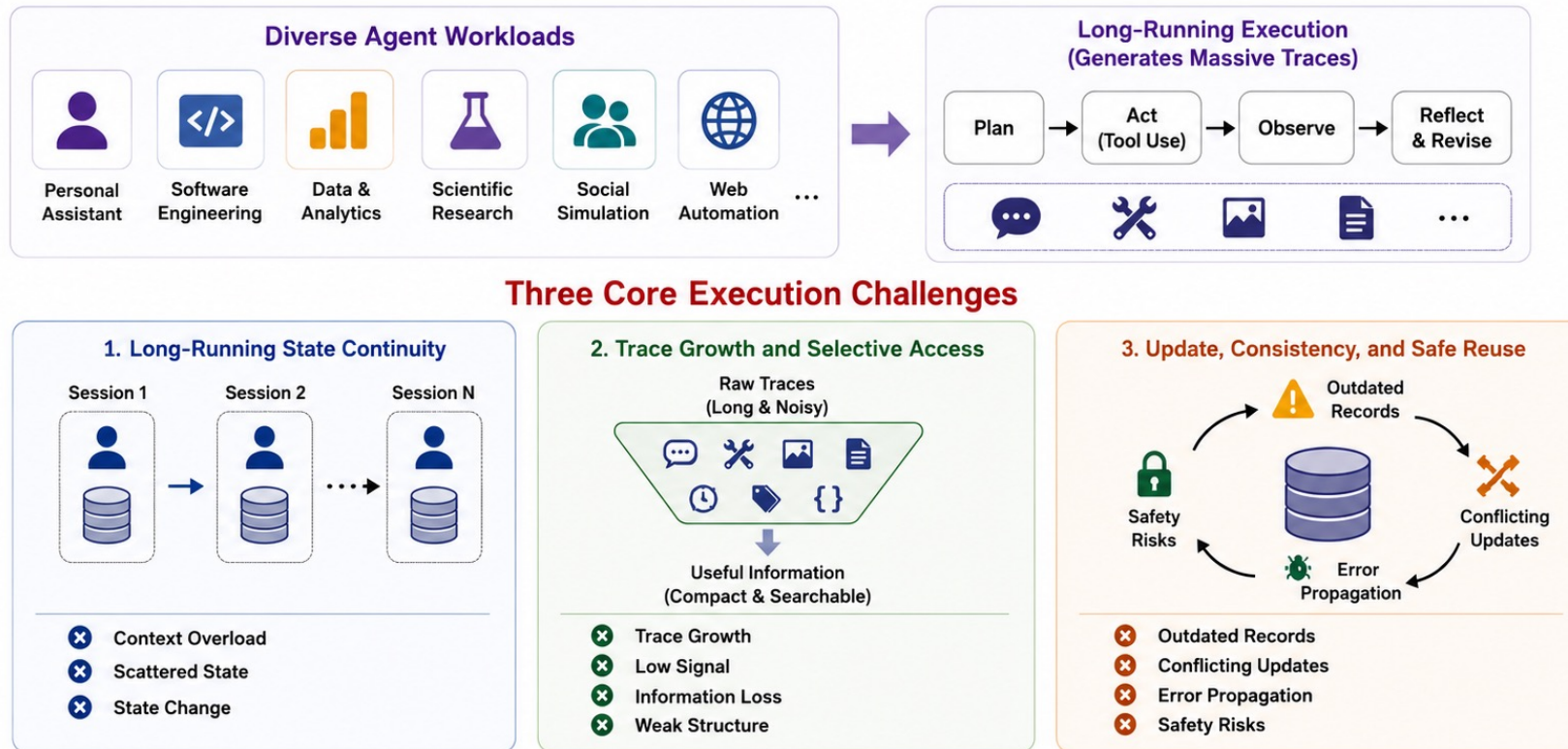
LLM agents are moving beyond single-turn generation toward **long-horizon execution**.



- **Personal Assistant Agents:** Manage schedules, user preferences, and follow-up tasks.
- **Software Engineering Agents:** Inspect repositories, run code, and debug failures.
- **Data and Analytics Agents:** Query databases, profile datasets, and generate reports.
- **Scientific Research Agents:** search evidence, verify claims, and track hypotheses.
-

From Agent Workloads to Execution Challenges

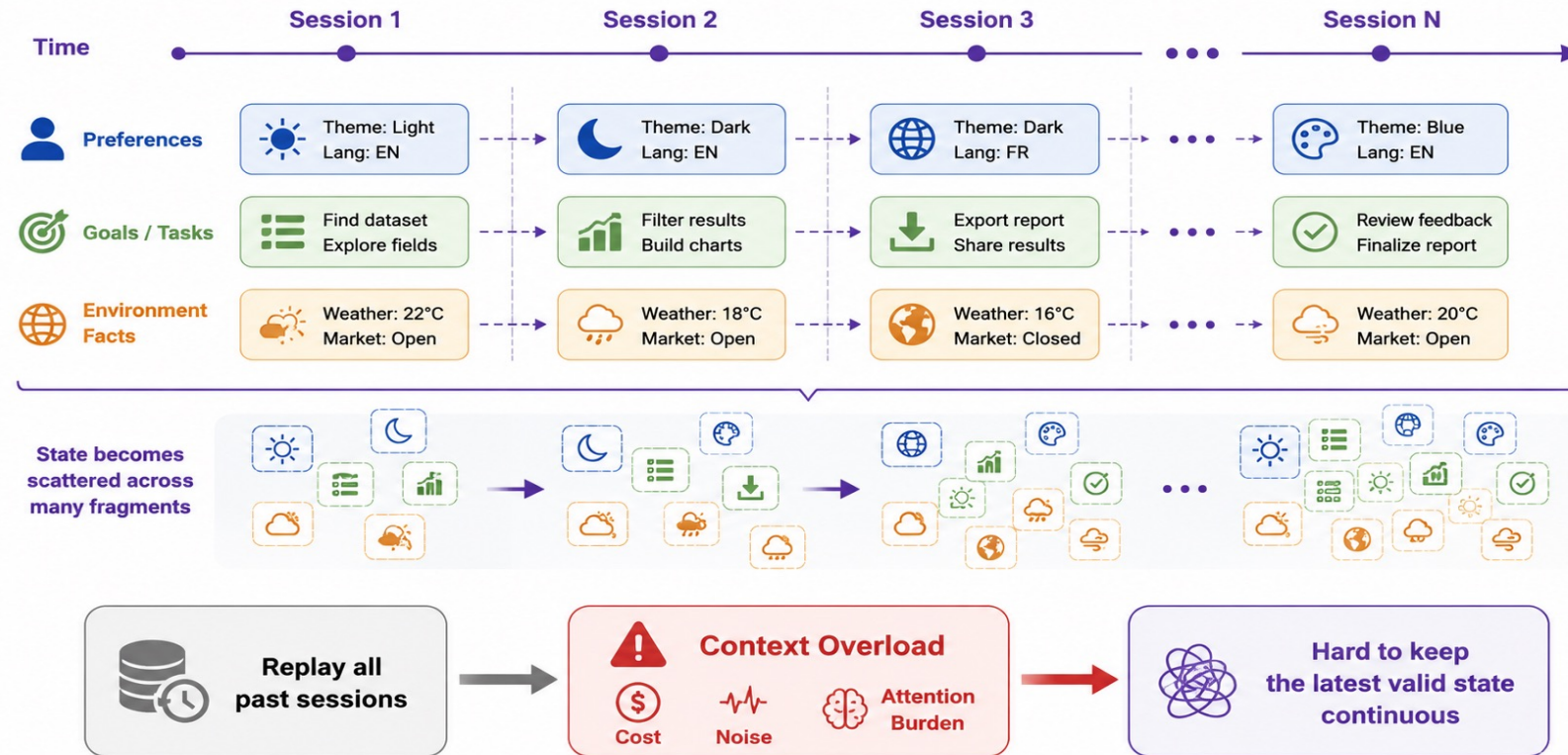
Agents fail when (1) state is lost, (2) traces are hard to access, or (3) saved records are reused incorrectly.



- **Long-Running State Continuity:** Important task-related states are scattered, changing, and easy to lose.
- **Trace Growth and Selective Access:** Long traces become noisy, redundant, and difficult to search.
- **Update, Consistency, and Safe Reuse:** Saved records may become stale, conflicting, or unsafe to reuse.

Challenge I: Long-Running State Continuity

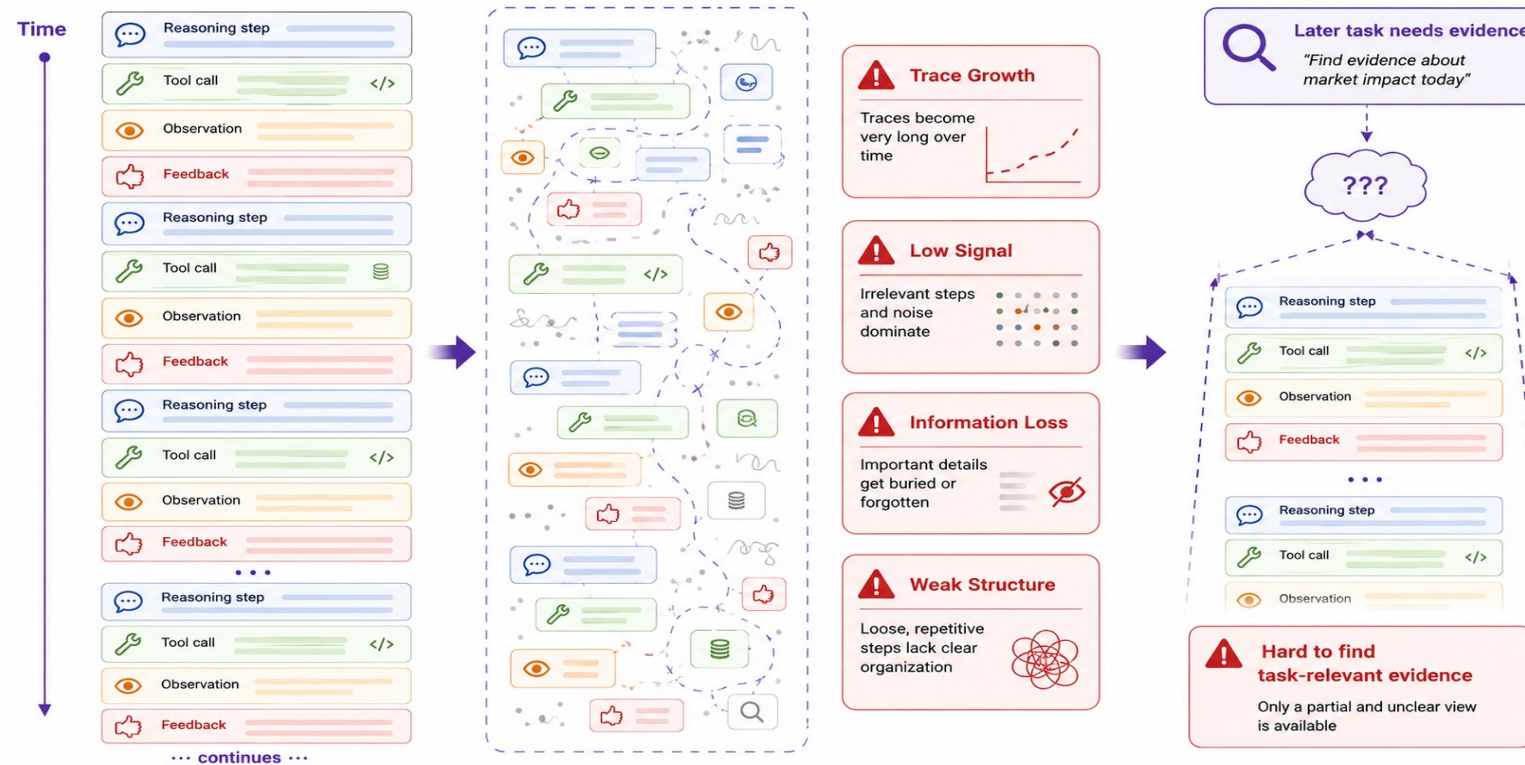
Long-running agents must keep state information continuous across sessions.



- **Context Overload:** Replaying the full history increases cost, noise, and attention burden.
- **Scattered State:** User preferences and other facts are spread across sessions.
- **State Change:** User needs, task goals, and environment facts may change over time.

Challenge II: Trace Growth and Multi-modal Data Retrieval

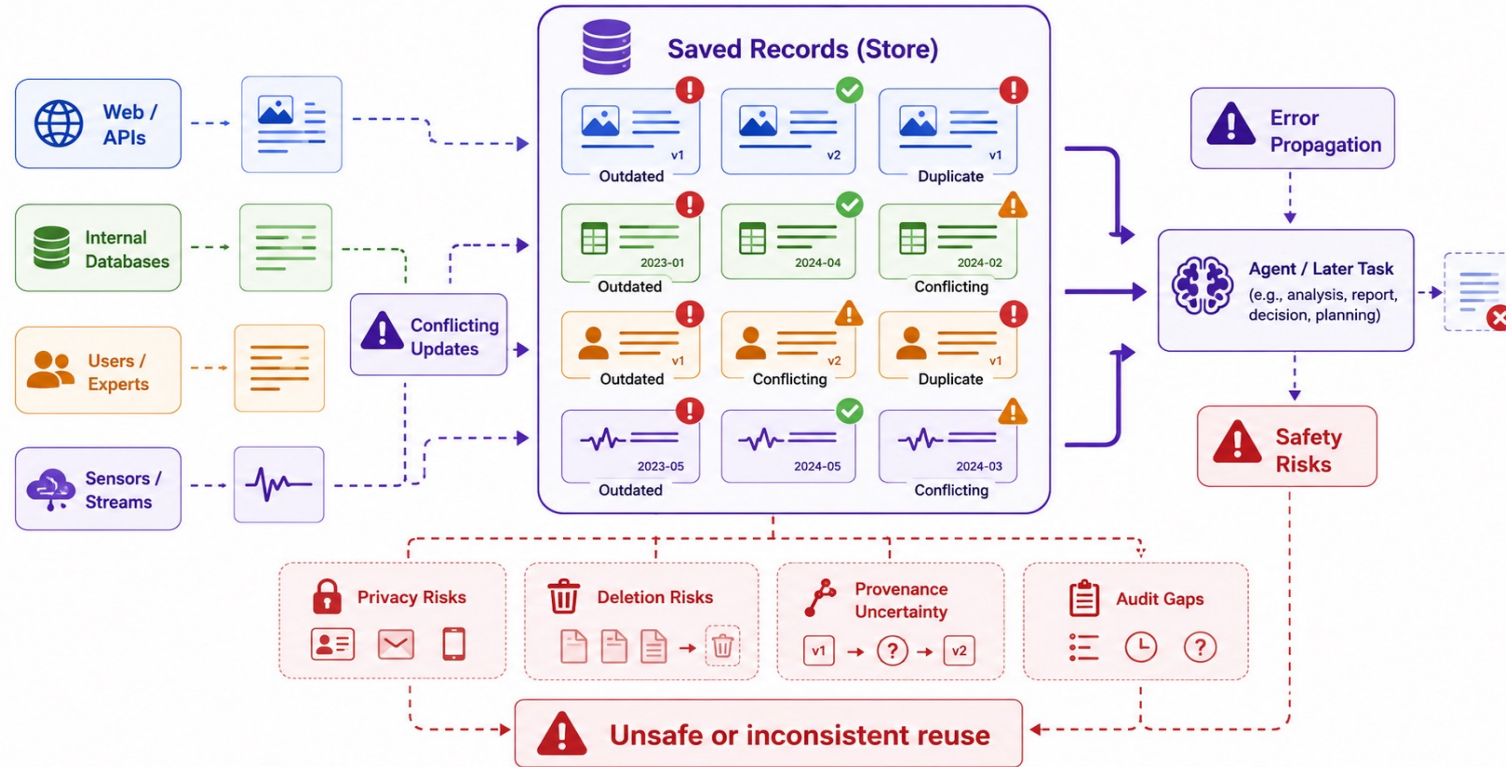
Long traces contain heterogeneous, multi-modal artifacts that must be selectively retrieved, summarized, and organized to support later tasks.



- **Trace Growth:** Execution produces a growing stream of diverse modalities (e.g., natural language reasoning, structured API calls, code snippets, images, tool outputs)
- **Low Signal:** Raw traces are often repeated, noisy, and too large to use directly.
- **Information Loss:** Heavy summarization may remove details needed for later reasoning.
- **Weak Structure:** Flat or weakly indexed traces make relevant evidence hard to retrieve.

Challenge III: Update, Consistency, and Safe Reuse

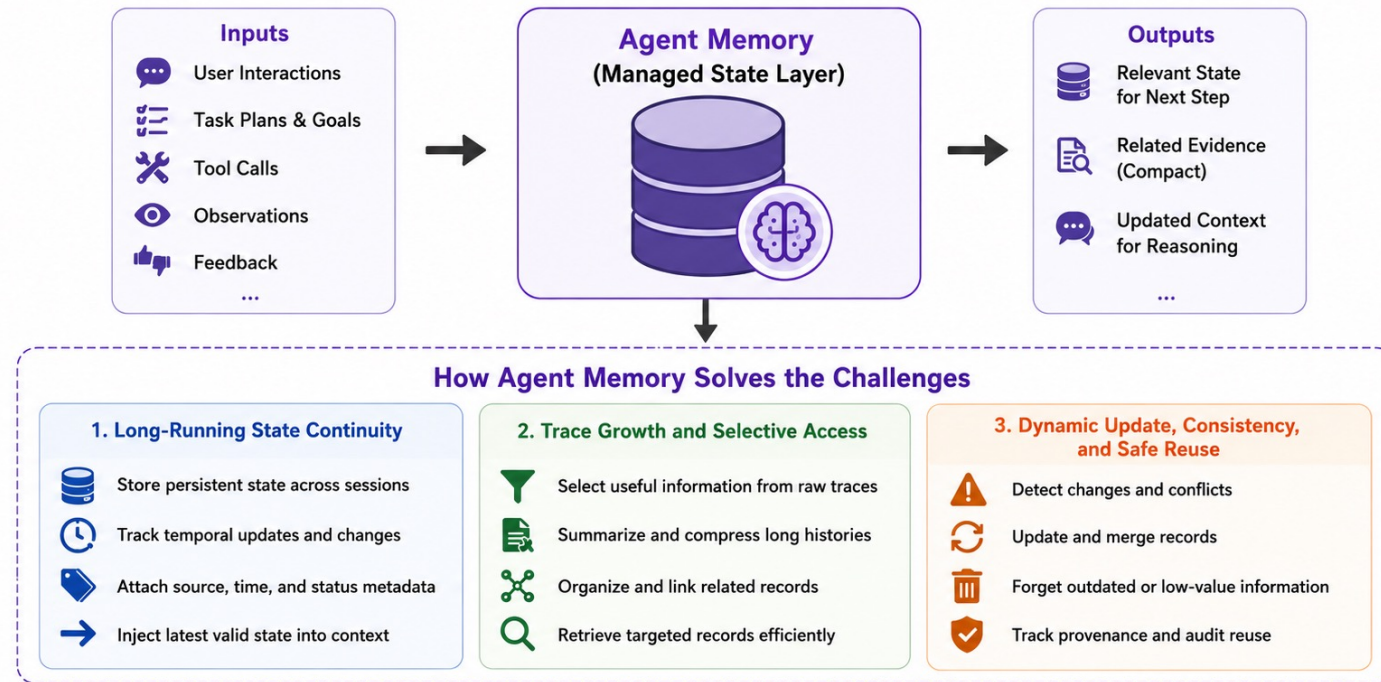
Long-term records must be updated and controlled to be safely reused.



- **Outdated Records:** Saved records may become outdated, duplicated, or inconsistent.
- **Conflicting Updates:** New information may conflict with older records from different sources.
- **Error Propagation:** Reused experience may carry old errors into new tasks.
- **Safety Risks:** Long-term storage raises privacy, deletion, source tracking, and audit needs.

Agent Memory as Solution

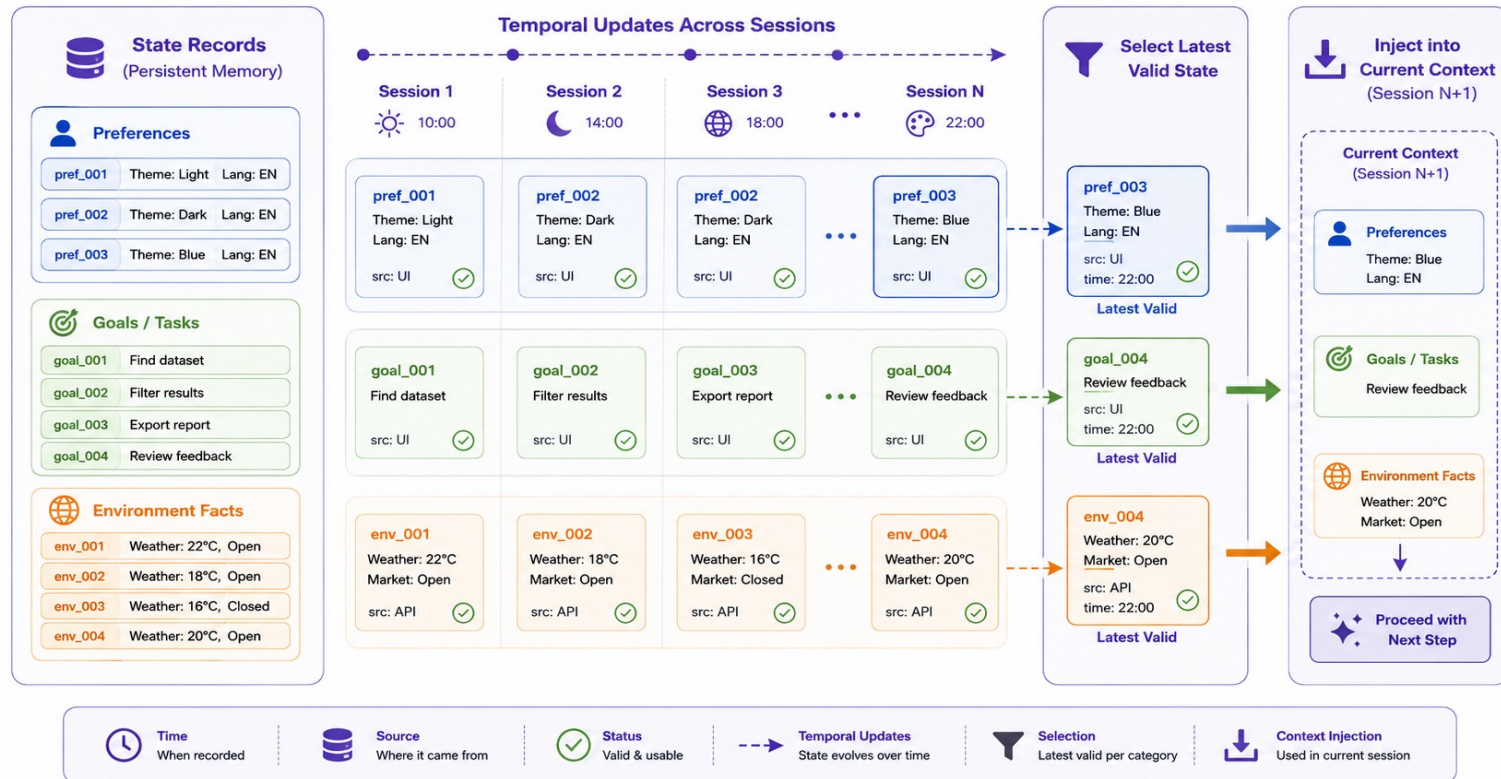
Agent memory provides a solution that connects past execution with future decisions.



- **Long-Running State Continuity:** Agent memory keeps and maintains user, task, and environment state beyond the current context.
- **Trace Growth and Selective Access:** Agent memory turns long traces into compact, well-organized, and accessible records.
- **Dynamic Update, Consistency, and Safe Reuse:** Agent memory updates, merges, forgets, and audits saved records over time.

Memory for Long-Running State Continuity

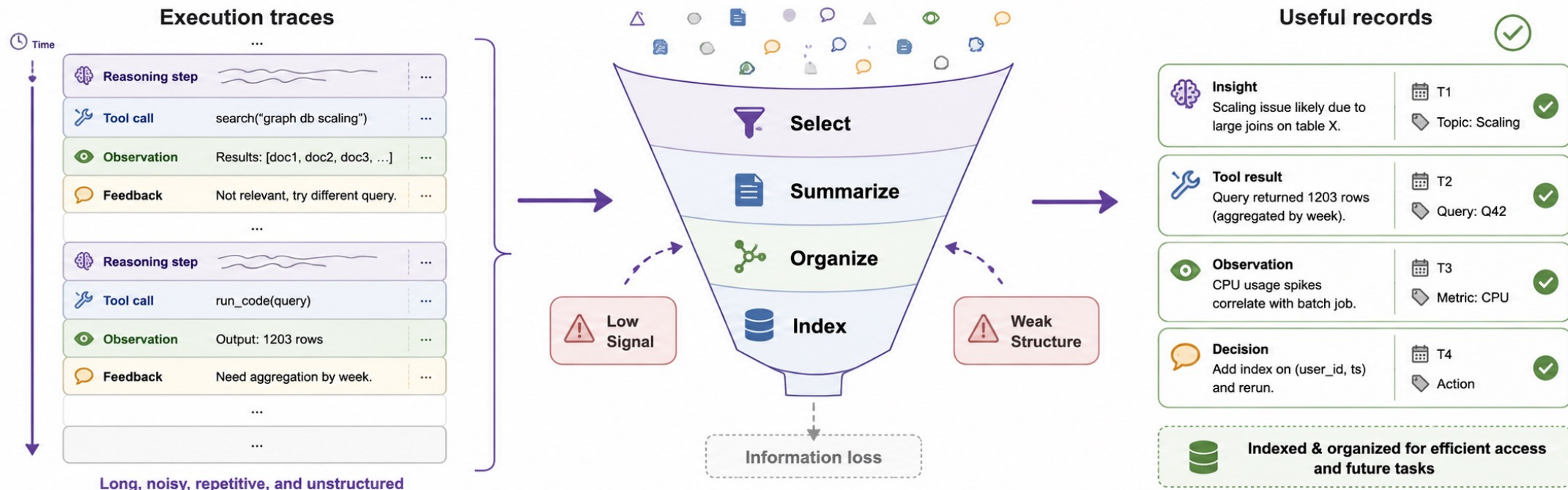
Memory preserves state that remains current, traceable, and reusable beyond the context.



- **State Records:** Store user preferences, task constraints, and environment facts across sessions.
- **Temporal Updates:** Keep track of state changes as user goals or external facts evolve.
- **Source Tracking:** Attach time, source, and status information to saved state.
- **Context Injection:** Retrieve the latest valid state when the agent starts a new step or session.

Memory for Trace Growth and Data Retrieval

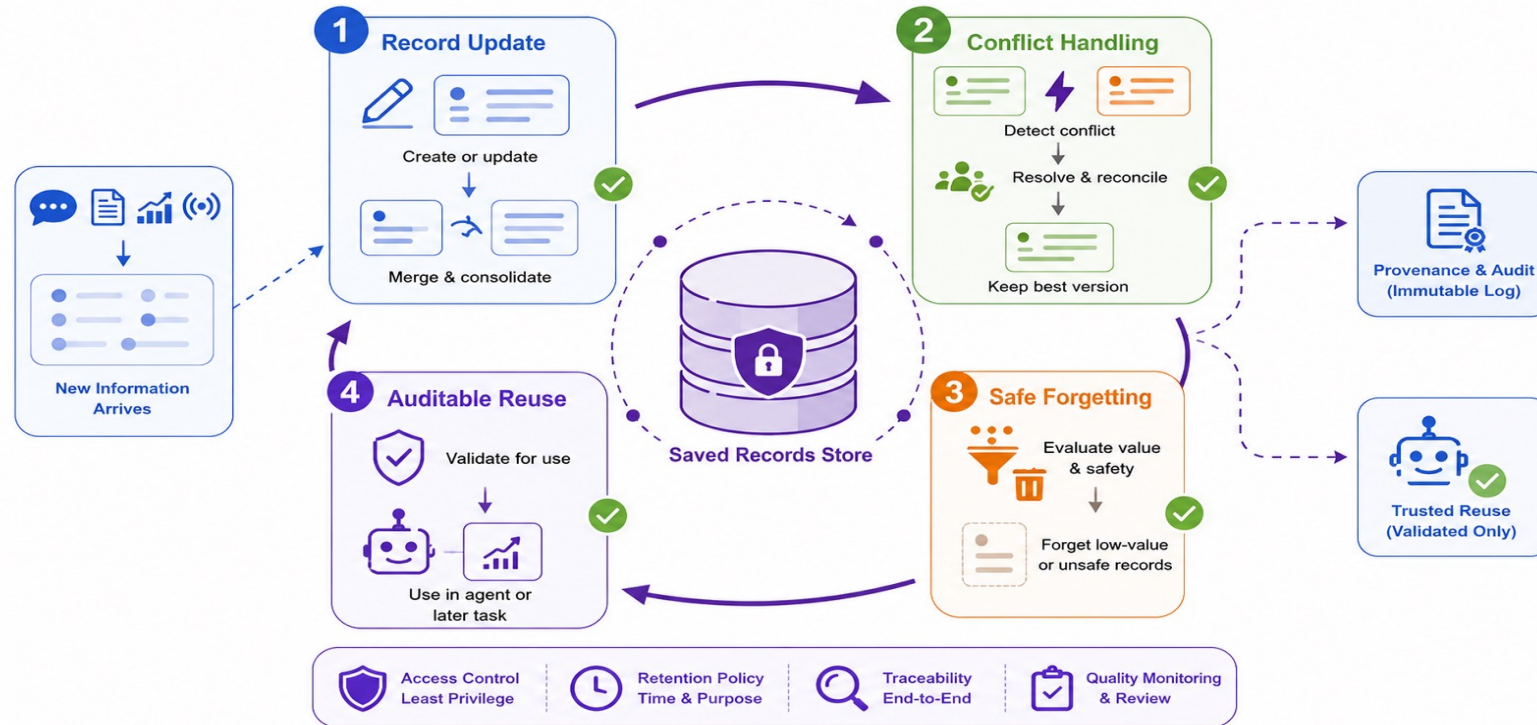
Memory transforms heterogeneous execution traces into modality-aware, indexed records for efficient cross-modal retrieval.



- **Trace Selection:** Identify useful information from plans, actions, tool calls, observations, and feedback.
- **Trace Compression:** Summarize long execution histories while keeping key details.
- **Trace Organization:** Link related events, facts, decisions, and intermediate results.
- **Multi-modal Retrieval:** Retrieve task-relevant records by querying across modality-specific indices without replaying the full history.

Memory for Dynamic Update, Consistency, and Safe Reuse

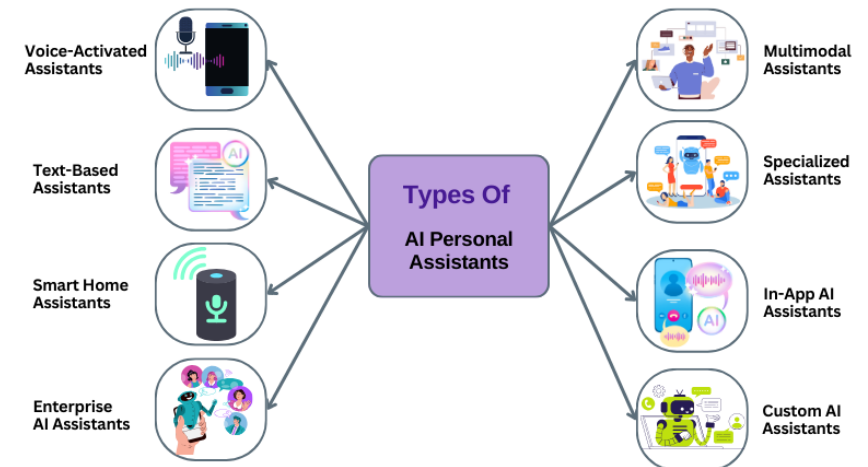
Memory controls how saved records are updated, corrected, forgotten, and safely reused.



- **Record Update:** Revise saved information when user needs, task goals, or environments change.
- **Conflict Handling:** Resolve contradictions between old and new records from different sources.
- **Safe Forgetting:** Remove outdated, low-value, duplicated, or unsafe records when needed.
- **Auditable Reuse:** Track source and use history before saved information is reused.

Application: Personal Assistant Agents

- ❑ **Memory reduces repeated questioning and supports continuous personalization in personal assistants, such as OpenClaw.**
- **State Continuity:**
 - Store user preferences, routines, schedules, constraints, and pending tasks in personal agents, such as OpenClaw.
- **Selective Retrieval:**
 - Retrieve relevant personal history before planning trips, managing calendars, or recommending content, such as travel and reminder assistants.
- **Safe Reuse:**
 - Update changed preferences and avoid using outdated or sensitive information, such as controllable memory settings in ChatGPT.



Application: Software Engineering Agents

- ❑ **Memory preserves project context and reduces repeated debugging failures in software agents, such as Claude Code-style coding assistants.**
- **State Continuity:**
 - Store repository structure, coding rules, API constraints, open issues, and task progress in coding agents, such as Claude Code, Codex and Copilot.
- **Selective Retrieval:**
 - Retrieve related error logs, failed commands, test outputs, patches, and verified fixes during debugging, such as Cursor.
- **Safe Reuse:**
 - Reuse prior fixes only when they match the current codebase, dependency state, and validation results, such as CI-checked coding workflows.



Application: Data and Analytics Agents

- ❑ Memory keeps data understanding consistent across analysis workflows, such as Fabric and Power BI Copilot scenarios.

- **State Continuity:**

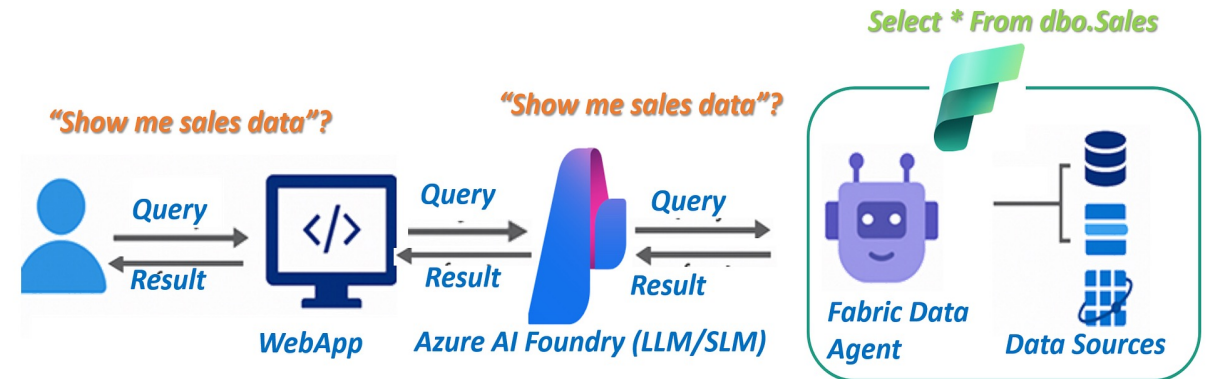
- Store schemas, semantic models, table meanings, data profiles, and business assumptions in analytics agents, such as Fabric Data Agents.

- **Selective Retrieval:**

- Retrieve prior SQL queries, joins, cleaning choices, profiling results, and report decisions, such as Power BI Copilot-style analysis.

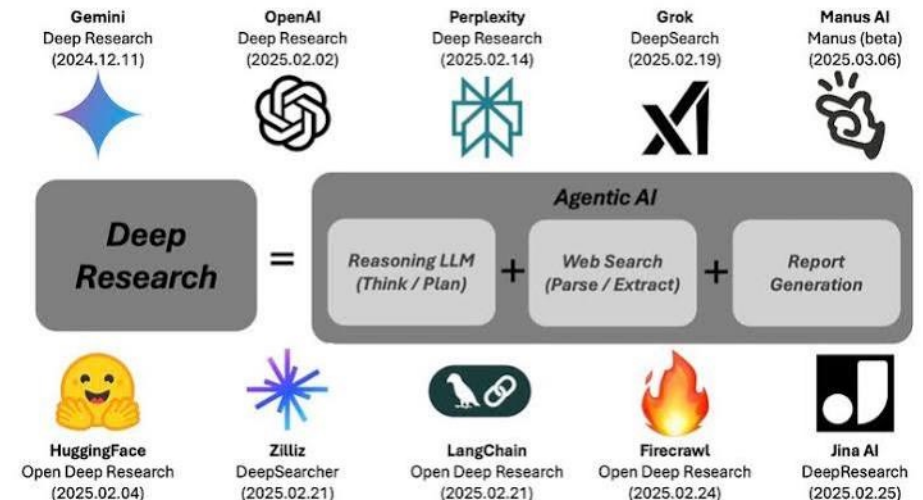
- **Safe Reuse:**

- Update saved records when schemas, data distributions, permissions, or business goals change, such as governed enterprise data agents.



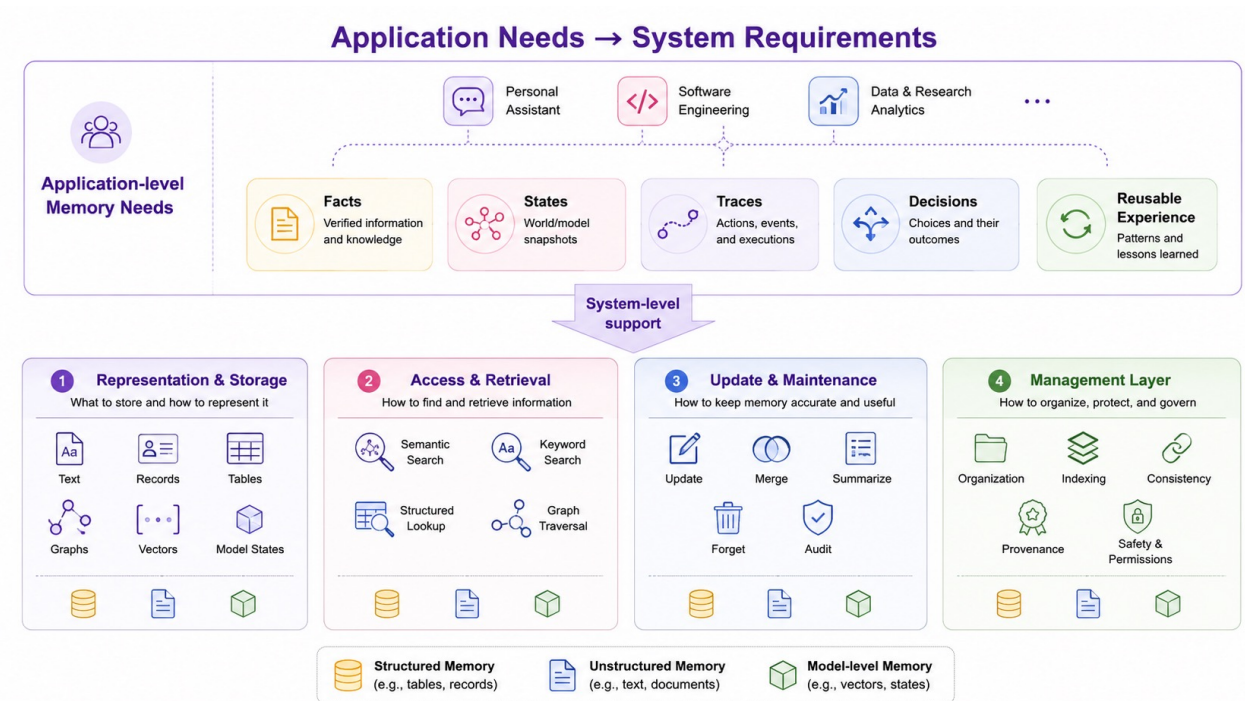
Application: Scientific Research Agents

- ❑ **Memory connects evidence, hypotheses, procedures, and results across long-running research workflows.**
- **State Continuity:**
 - Store sources, claims, hypotheses, experiment plans, observations, results, and protocols in research agents, such as Gemini Deep Research and Google AI Co-Scientist.
- **Selective Retrieval:**
 - Retrieve citations, source metadata, prior findings, and procedures during long-running studies, such as literature review agents.
- **Safe Reuse:**
 - Revise hypotheses and mark unsupported, outdated, or conflicting evidence before reuse, such as scientific hypothesis-generation agents.



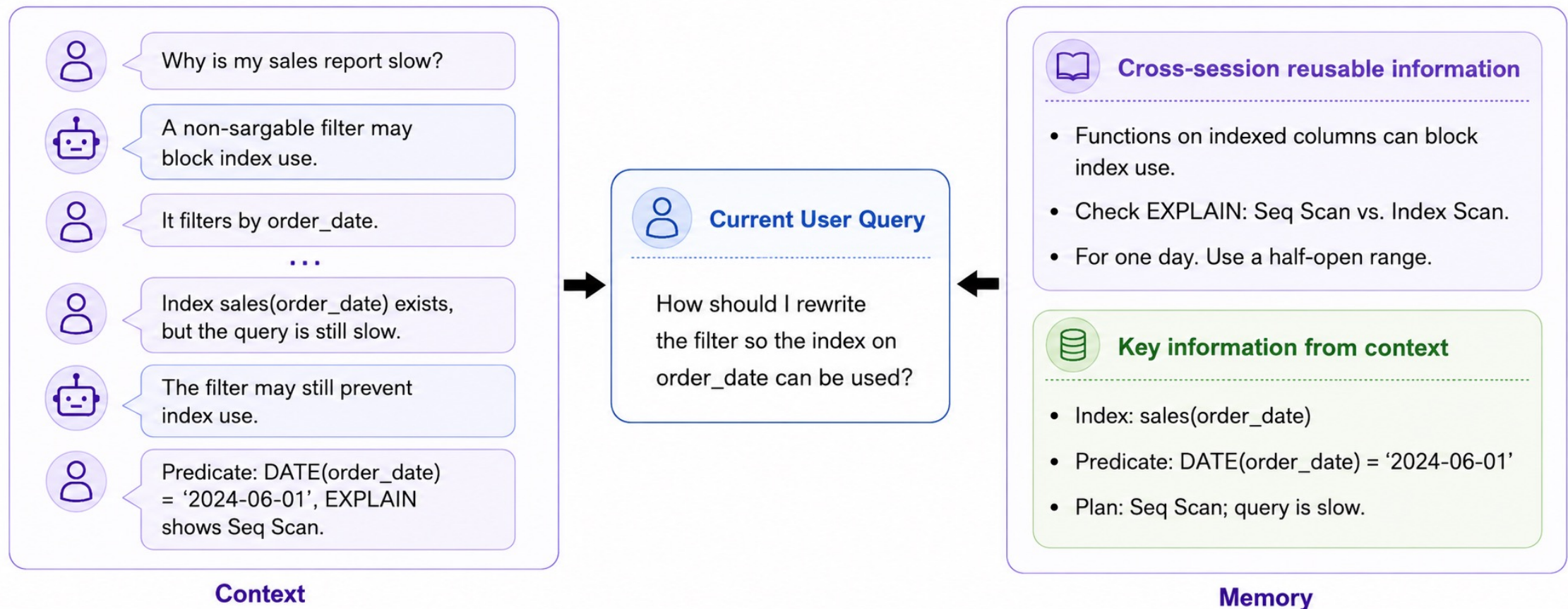
From Applications to System Requirements

- ❑ **Application-level memory requires system-level support for storage, access, update, and management.**
- **What to store:** Facts, states, traces, decisions, and reusable experience.
- **How to represent and store:** Text, records, tables, graphs, vectors, or model-level states.
- **How to retrieve:** Semantic search, keyword search, structured lookup, or graph traversal.
- **How to maintain:** Update, merge, summarize, forget, and audit saved information.



Memory vs. Context

- **Context** refers to the recent conversational information that is temporarily available to the agent and directly supports the current interaction.
- **Memory** refers to persistent information extracted from past interactions, which can be retained, managed, and reused across different conversations or tasks.



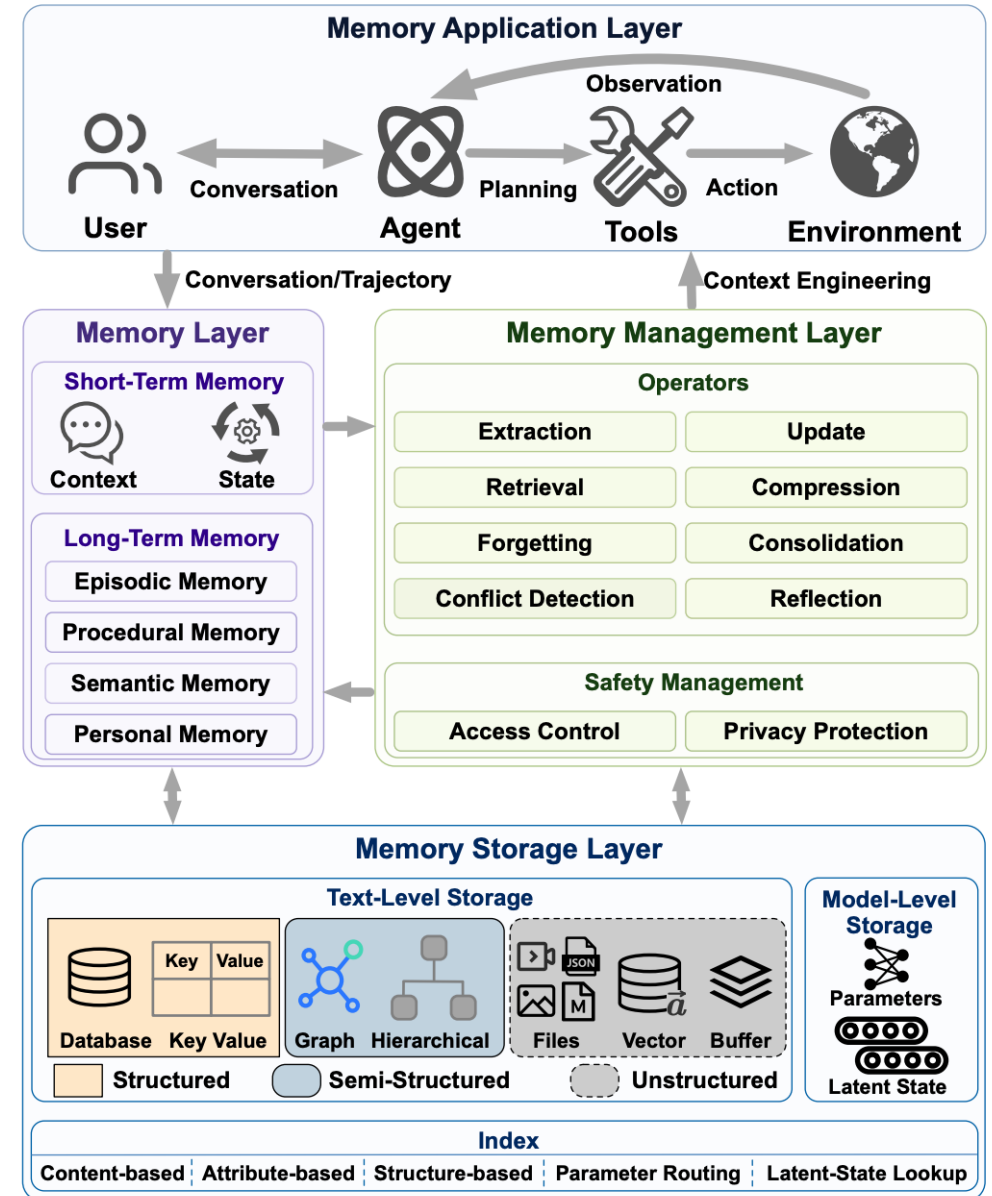
Memory as a Data Management Layer

- Agentic memory shares striking parallels with database systems. Both are responsible for storing, organizing, retrieving, updating, and maintaining information to support reasoning and decision-making in dynamic environments

	DBMS	Memory
Schema	Define tables, attributes, constraints, and data types	Define memory types, semantic fields, entities, relations, and metadata
Index	B-tree / hash / inverted indexes	Embedding, keyword, graph, and metadata as indexes
Transaction-Consistency	ACID properties	Reliable memory writes, conflict resolution, temporal consistency, and consistent updates
Compaction-GC	Merge data, remove obsolete records, and reclaims storage	Consolidates, summarizes, deduplicates, forgets stale memories
Provenance	Track data lineage, logs, sources, and audit trails	Track memory source, conversation origin, timestamps, confidence, and evidence

Data Management for Agent Memory

- **Memory Application Layer:**
 - Connect users, agents, tools, and environments through conversation, planning, action, observation, and context engineering.
- **Memory Layer:**
 - Maintain short-term context and task state, and preserve long-term episodic, procedural, semantic, and personal memory.
- **Memory Management Layer:**
 - Apply operators such as extraction, retrieval, update, compression, forgetting, consolidation, conflict detection, and reflection.
- **Memory Storage Layer:**
 - Materialize memory in structured stores, unstructured stores, and parametric stores, such as databases, graph DBs, JSON, Markdown, files, parameters, and latent states.



Overview of Data Management for Agentic Memory

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

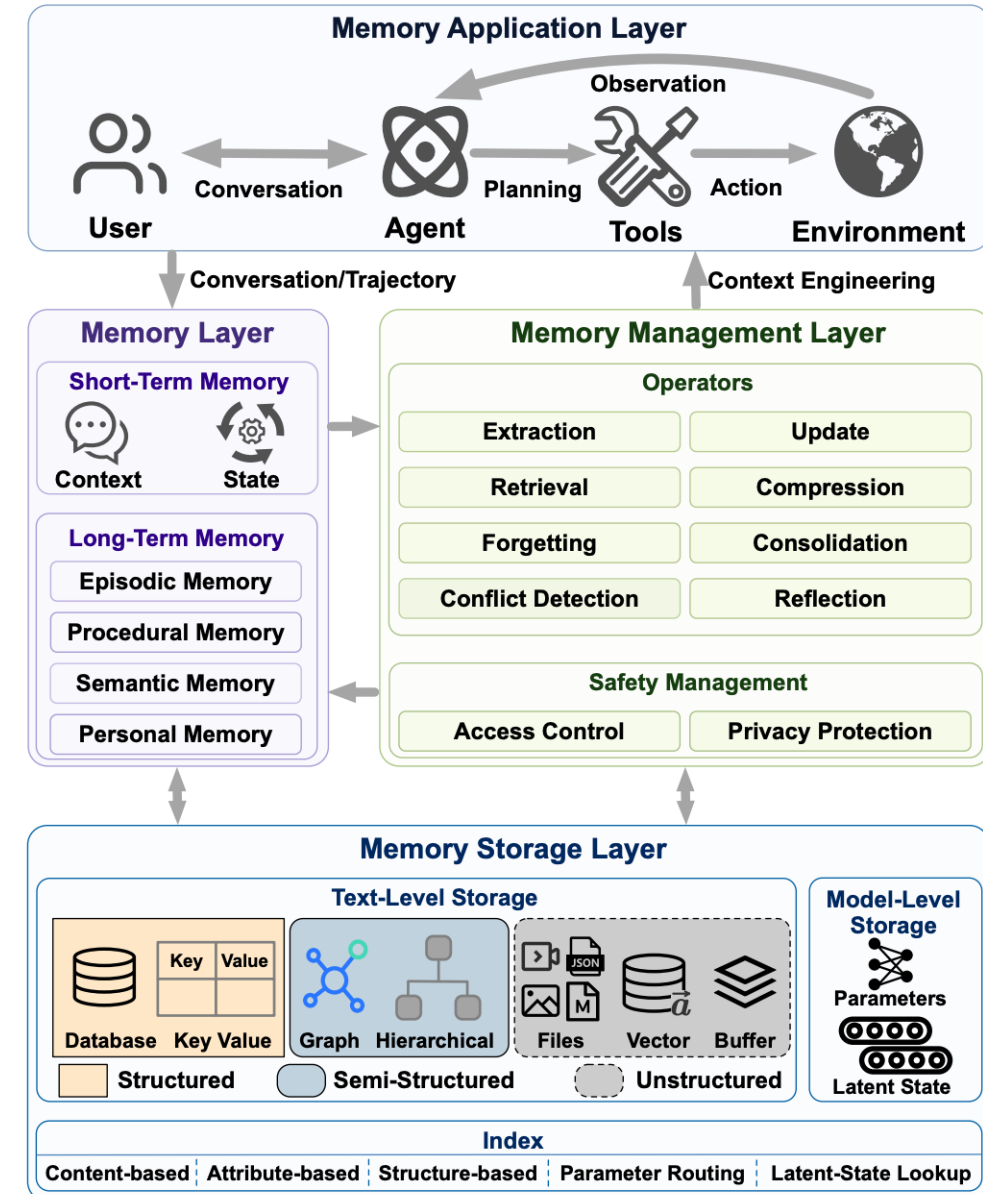
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

❑ Evaluation

❑ Open Problems



Agentic Memory Workflow

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

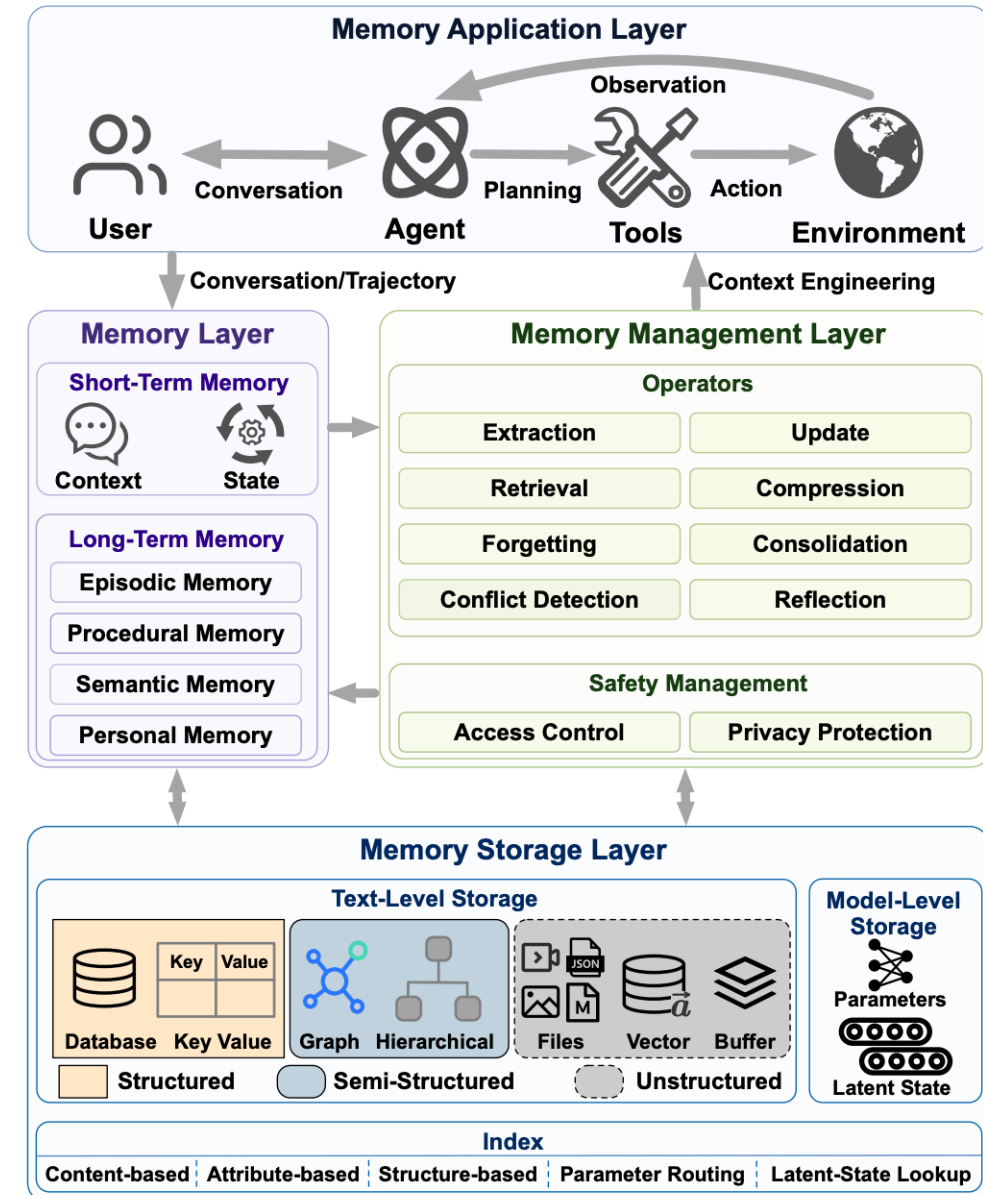
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

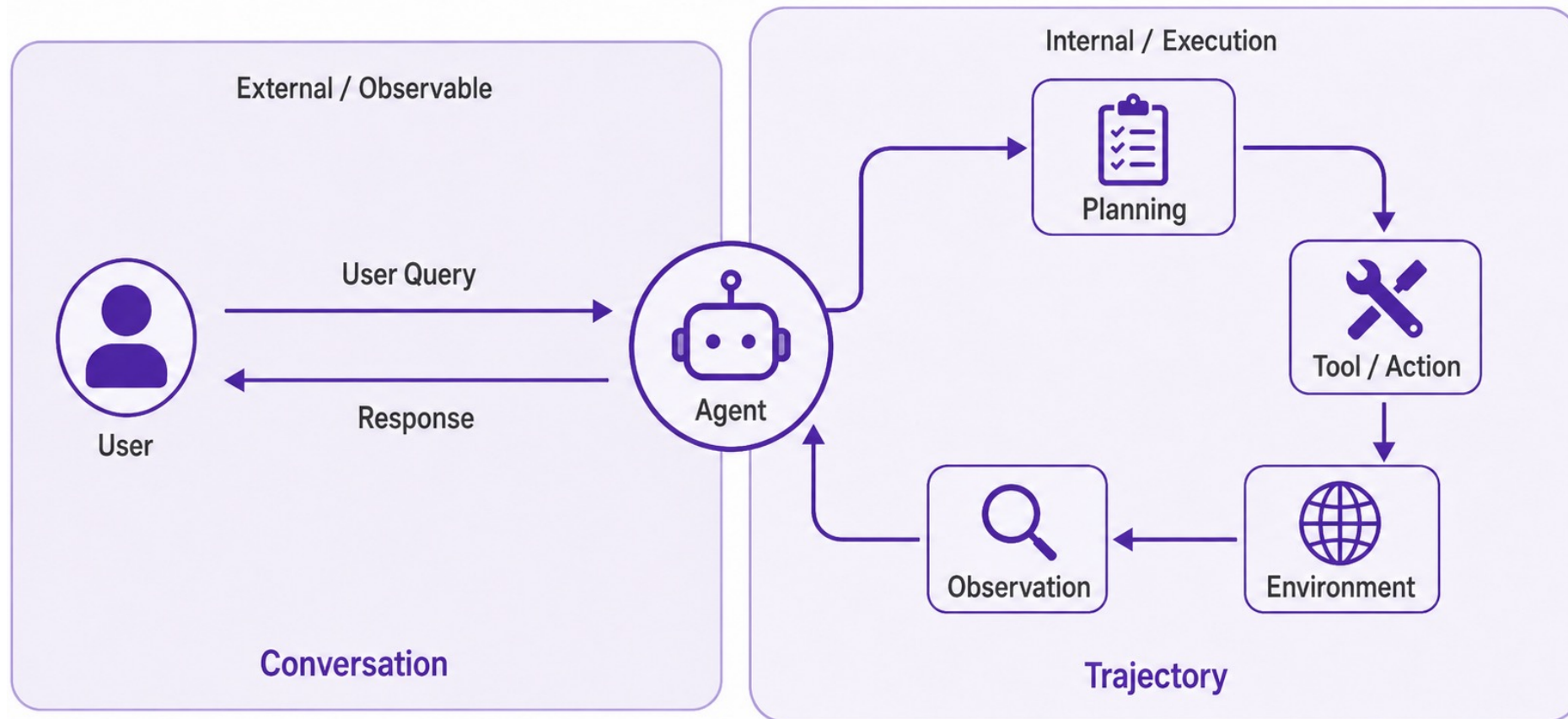
❑ Evaluation

❑ Open Problems



Memory Input

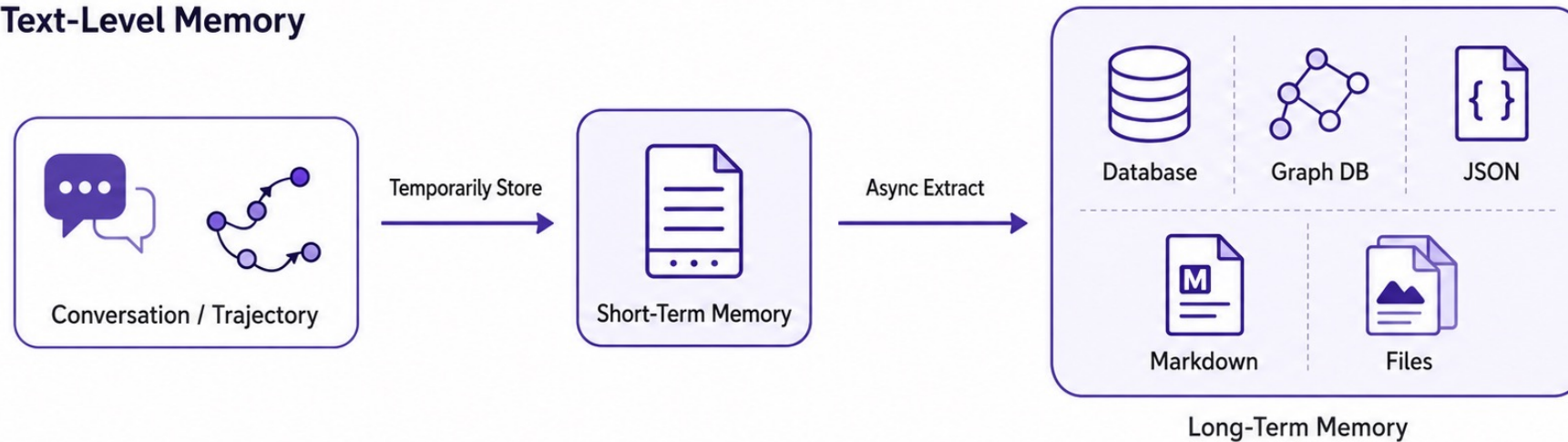
- **Conversation:** The **external exchanges** between the user and the agent.
- **Trajectory:** The agent's **internal execution states**, such as intermediate reasoning, tool calls, task progress, and environmental feedback.



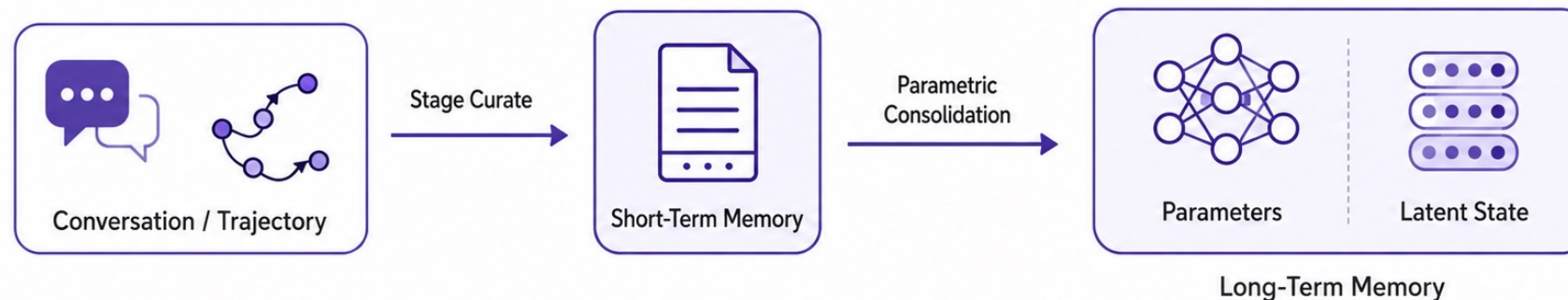
Memory Construction

- The raw input to the memory system is **temporarily stored in short-term memory**.
- The input is **asynchronously transformed into long-term memory by extraction**.

Text-Level Memory



Model-Level Memory



Memory Pipeline

- **Memory Retrieval:**

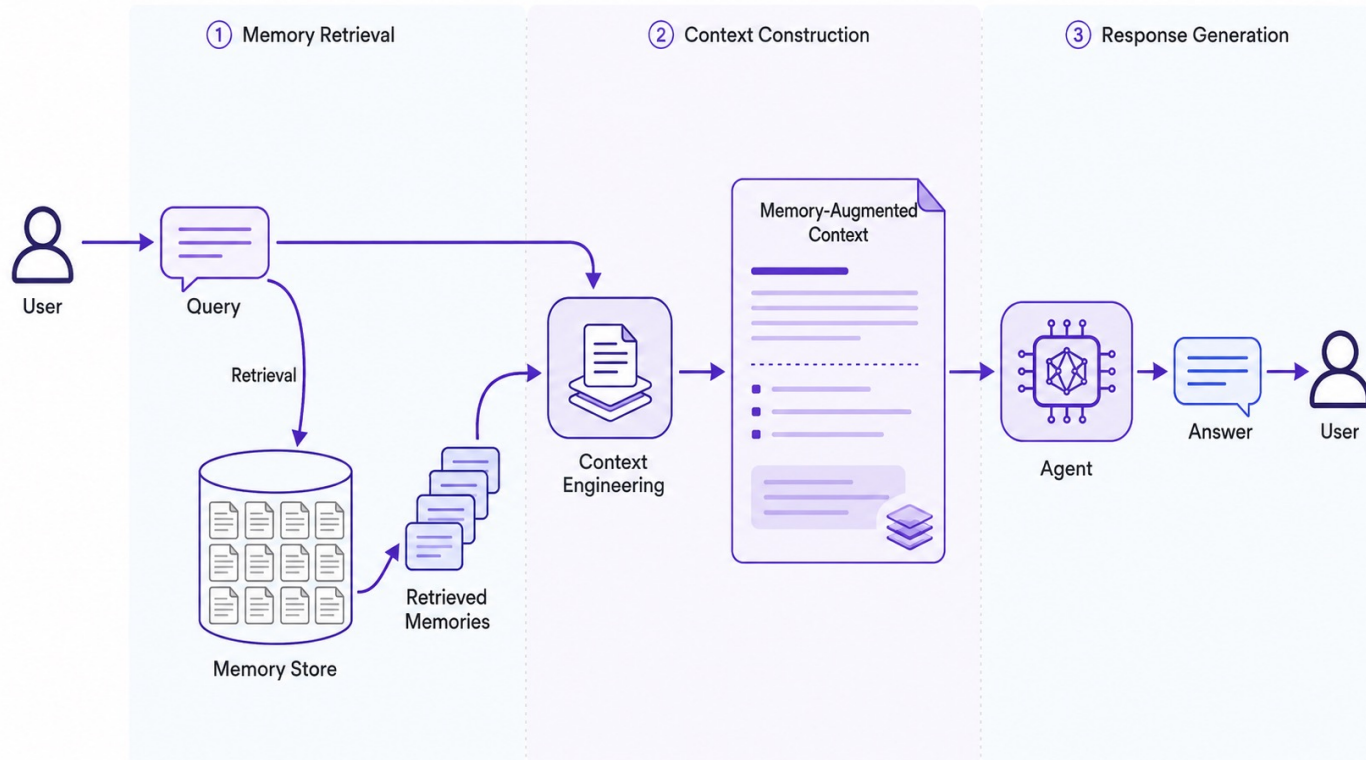
- For each query, agent first submits it to the memory system to retrieve relative memory items.

- **Context Construction:**

- The memories are integrated into a **memory-augmented context via context engineering.**

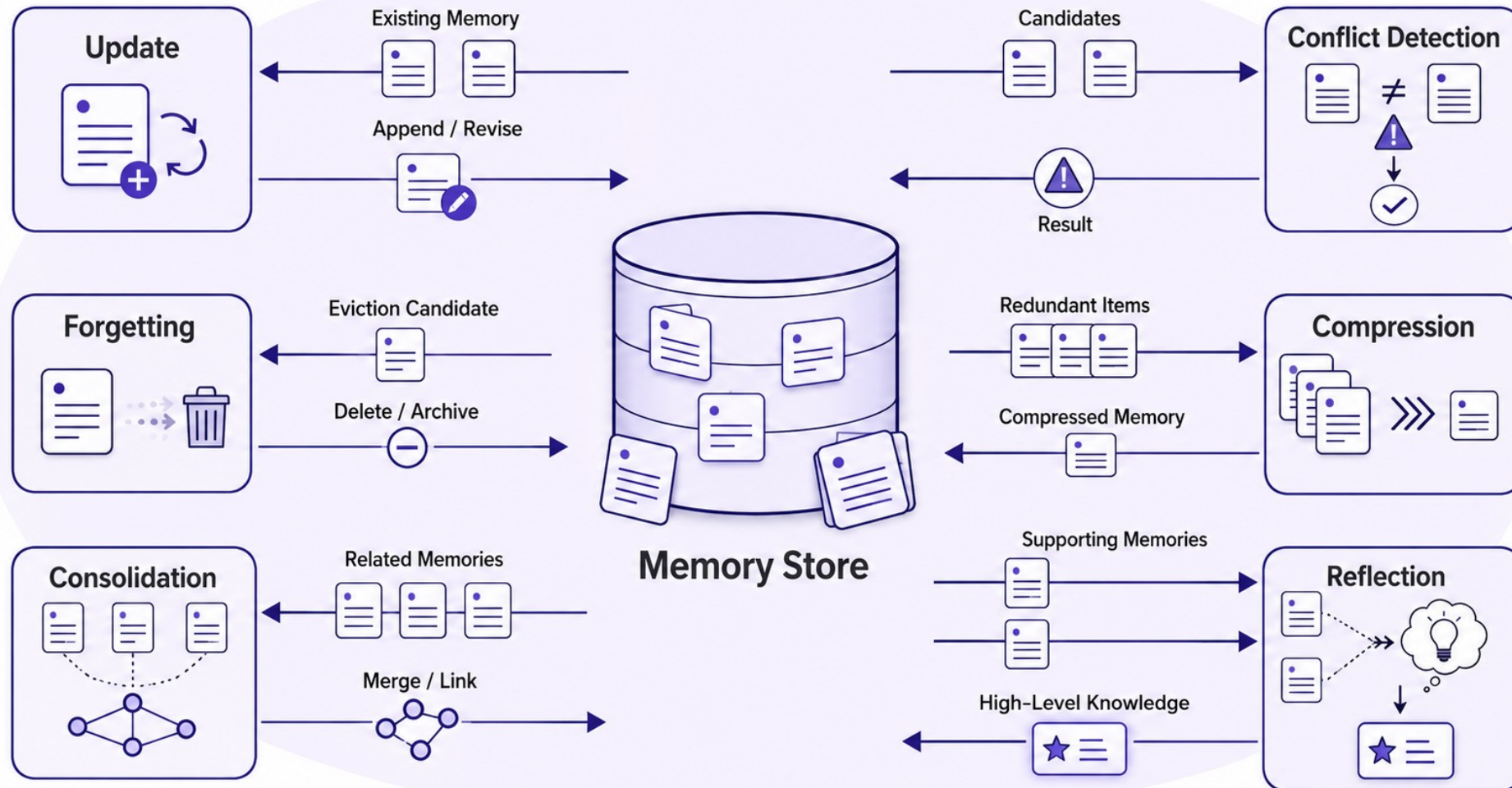
- **Response Generation:**

- The question, alongside the context, will be inputted into the agent to generate an answer.

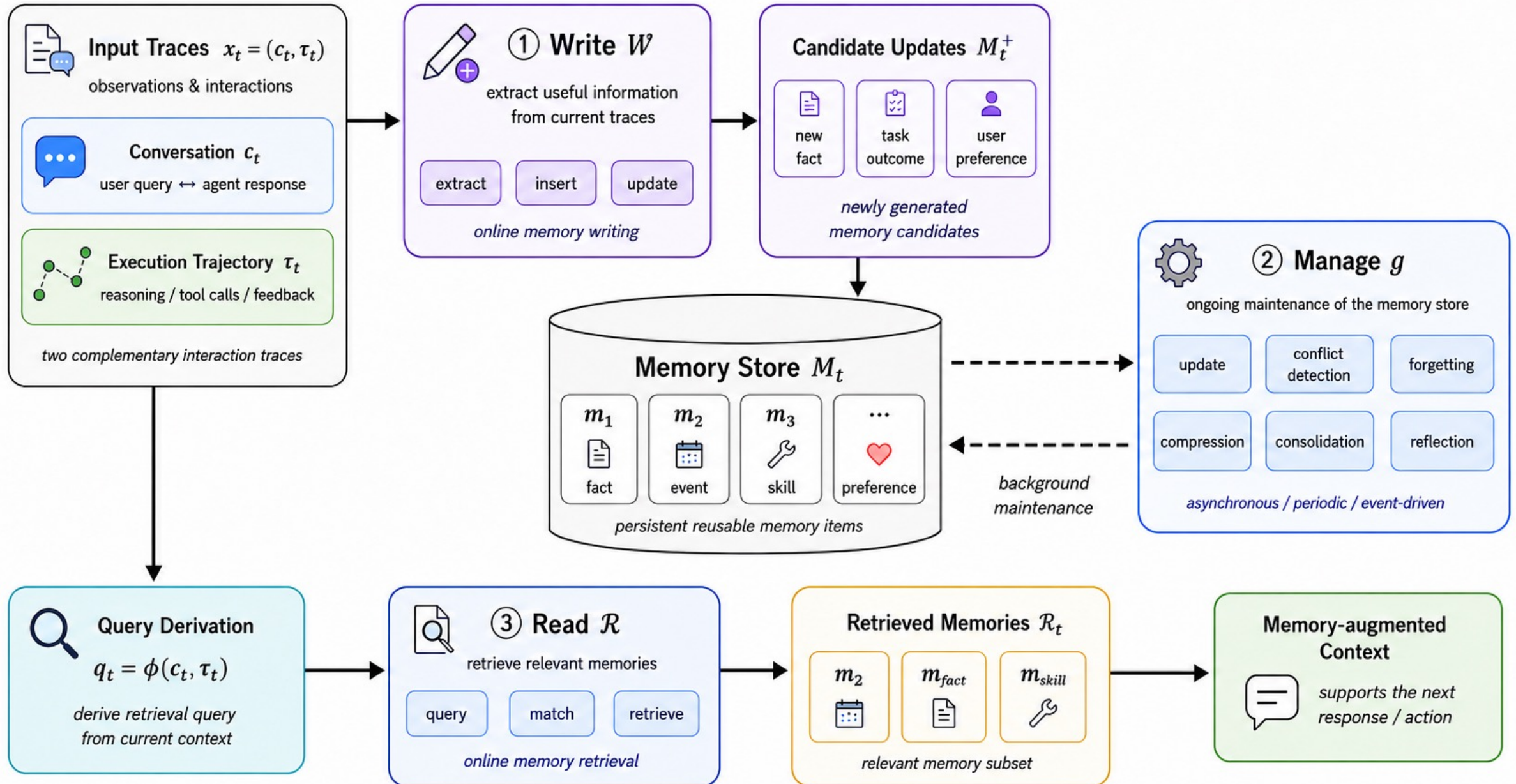


Memory Maintenance

A set of management operators maintain the memory system's functional effectiveness, structural organization, and alignment.

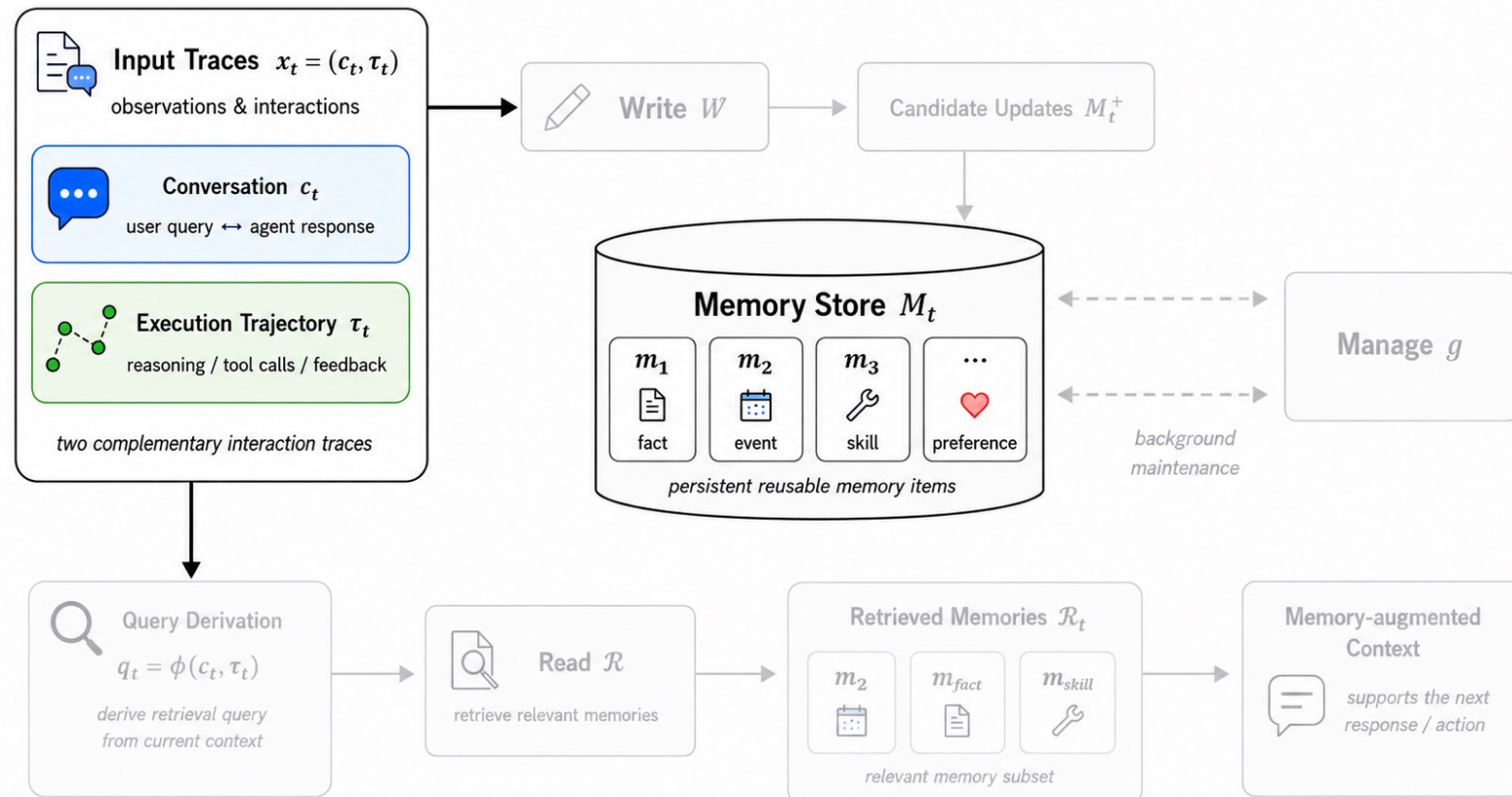


Write-Manage-Read Formalization



Write-Manage-Read Formalization

- **Input:** At interaction step (t), the agentic memory system observes $x_t = (c_t, \tau_t)$.
 - c_t : the user-agent conversation.
 - τ_t denote the agent's execution trajectory.
- **Memory Store:** The memory system maintains a memory store $M_t = \{m_i\}_{i=1}^{n_t}$.
 - m_i : the reusable unit information extracted from conversations or trajectories.

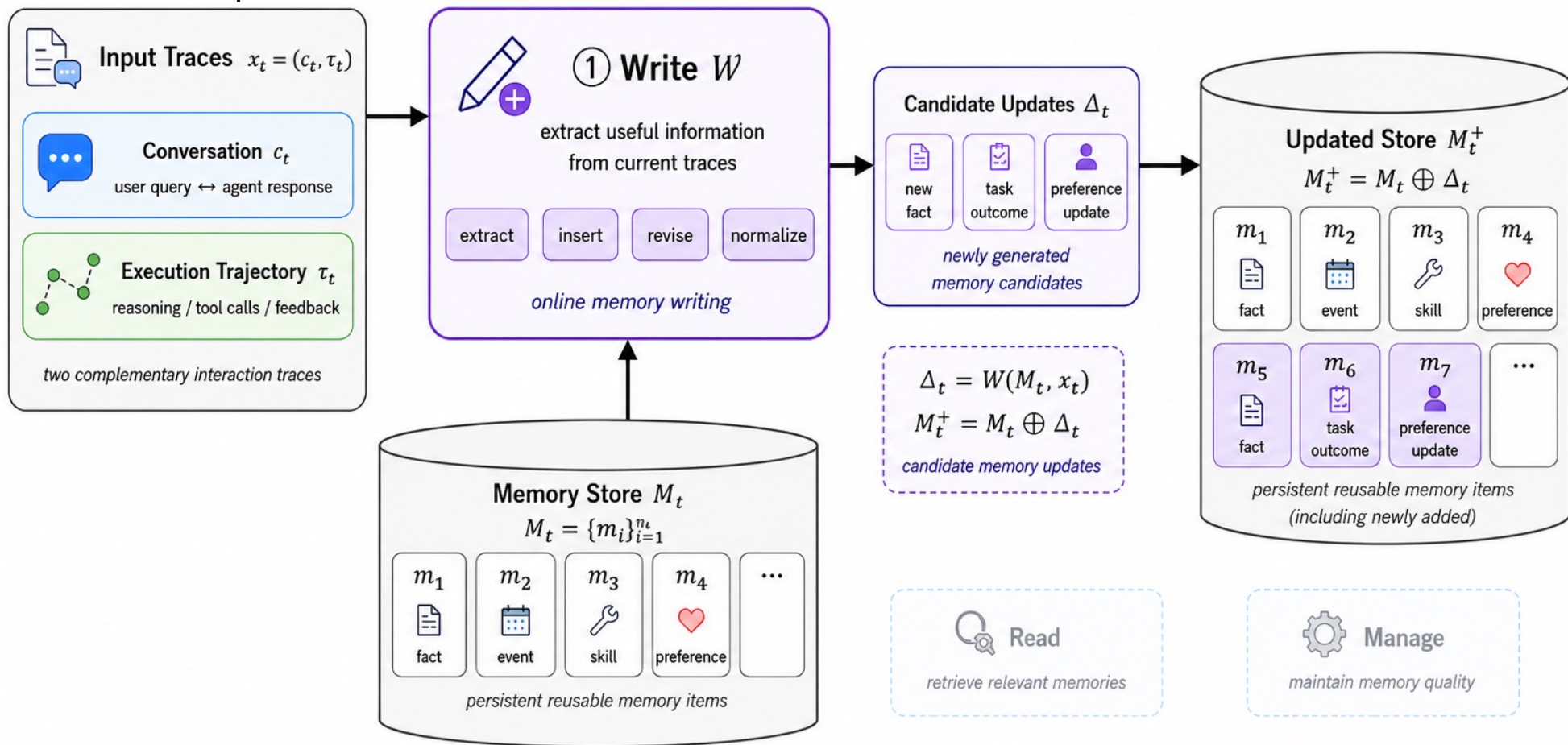


Write-Manage-Read Formalization

- **Write:** Given the current memory store M_t , the conversation c_t , and the trajectory τ_t , the write operation produces an updated intermediate memory store:

$$\Delta_t = \mathcal{W}(M_t, x_t), \quad M_t^+ = M_t \oplus \Delta_t$$

- $\oplus$: the update operator
- $\mathcal{W}$: the extraction operator

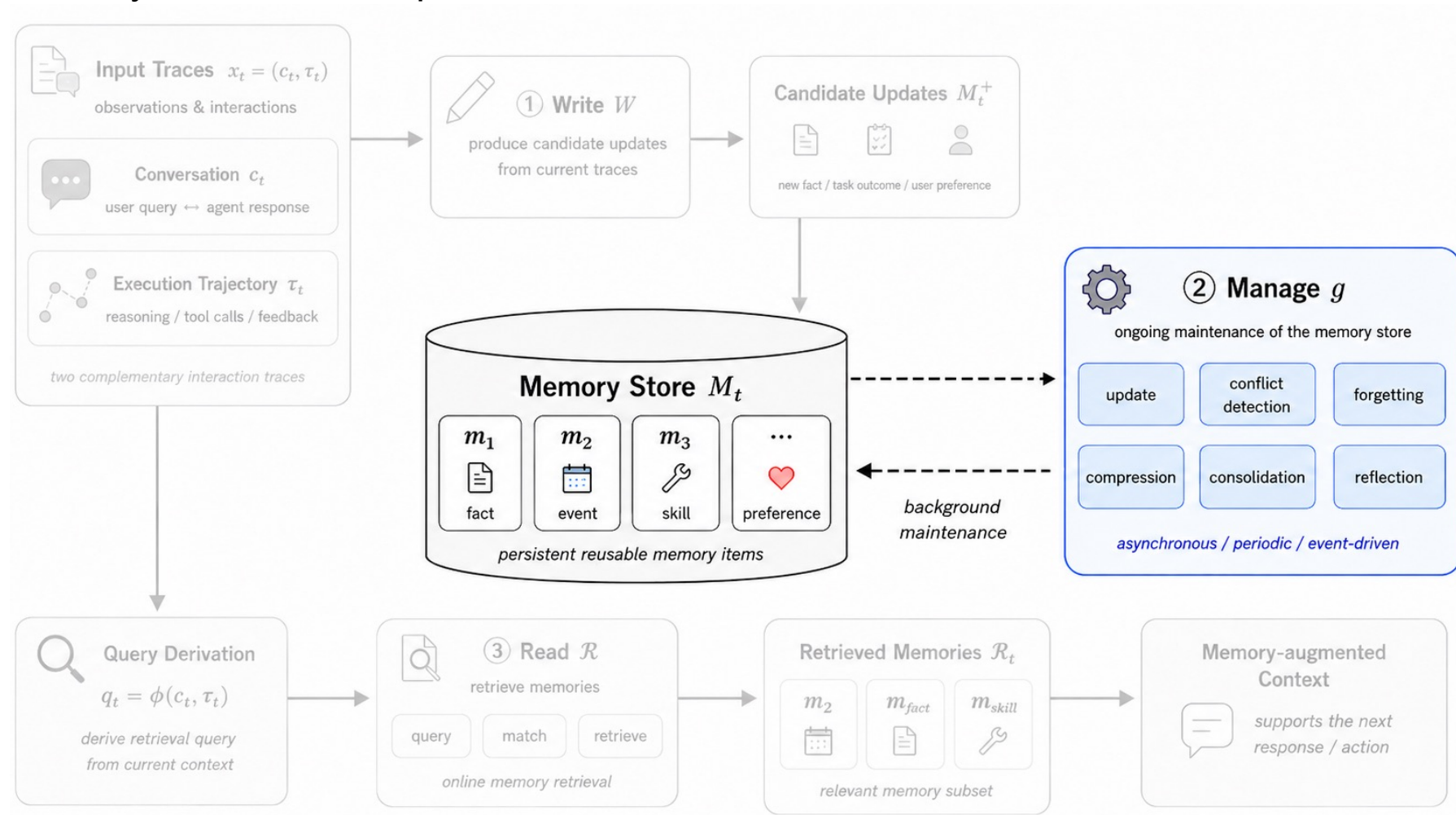


Write-Manage-Read Formalization

- **Manage:** As an independent process, the manage operation maintains the memory store through a family of operators. At a maintenance event (s), one or more operators are applied to the store:

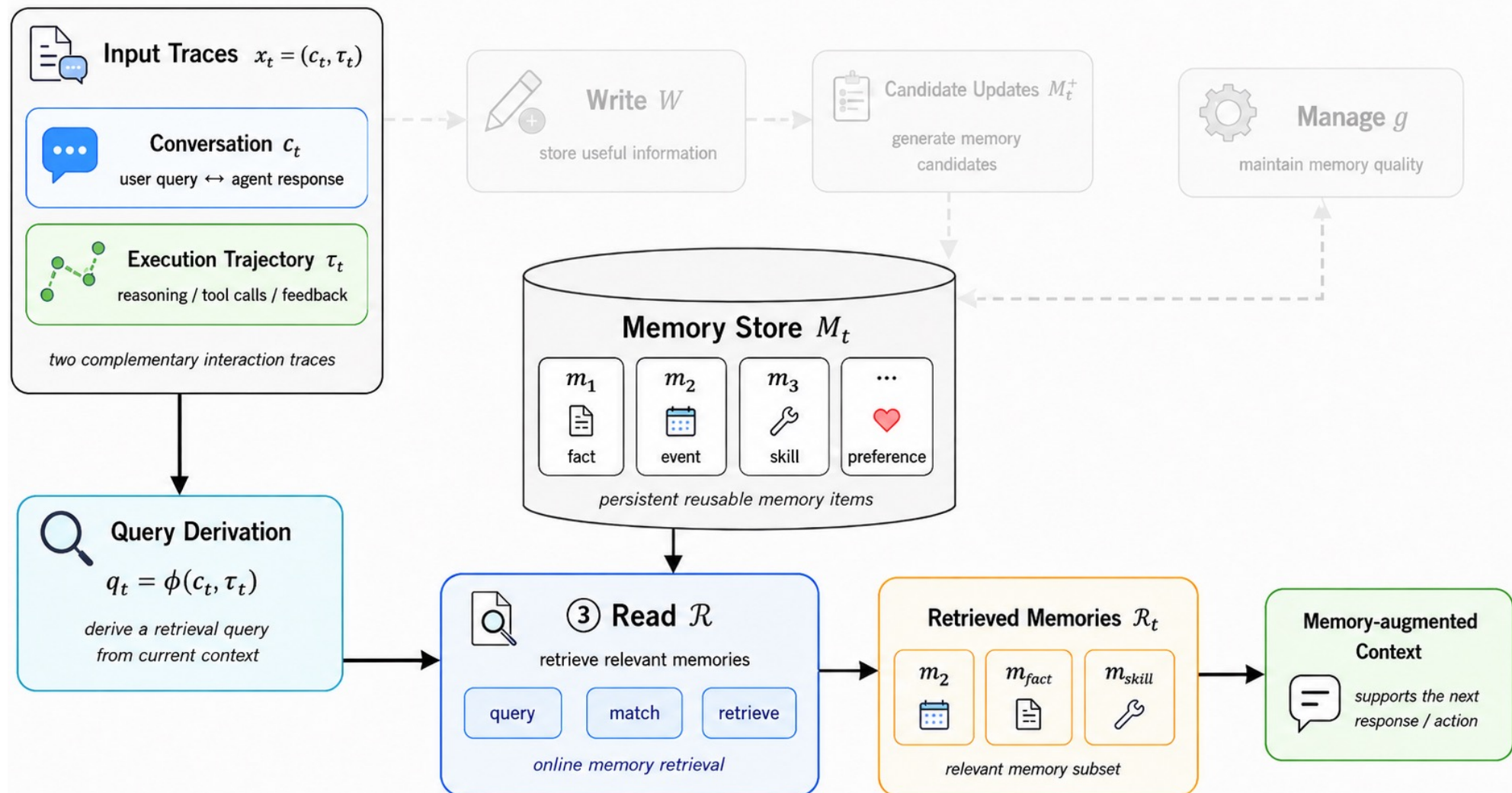
$$M^{(s+1)} = g_s(M^{(s)})$$

- g_s : the memory maintenance operators.



Write-Manage-Read Formalization

- **Read:** Let $q_t = \phi(c_t, \tau_t)$ denote the memory query derived from the current conversation and trajectory context. The read operation retrieves a set of related memory items R_t from the memory store: $R_t = \mathcal{R}(M_t, q_t)$.



Take-aways

- **Memory bridges past interaction traces and future agent behavior.**
 - Conversation and trajectories are transformed into persistent memories and reused through memory-augmented context.
- **The memory store is a persistent and evolving state.**
 - Memories are continuously constructed, revised, organized, abstracted, compressed, or removed over time.
- **Memory use is selective and context-dependent.**
 - Read retrieves a relevant subset of the store and integrates it into the context for subsequent reasoning or action.
- **The memory workflow can be unified as Write-Manage-Read.**
 - The formalization provides a common abstraction for describing different agentic memory systems.

Memory Categorization

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

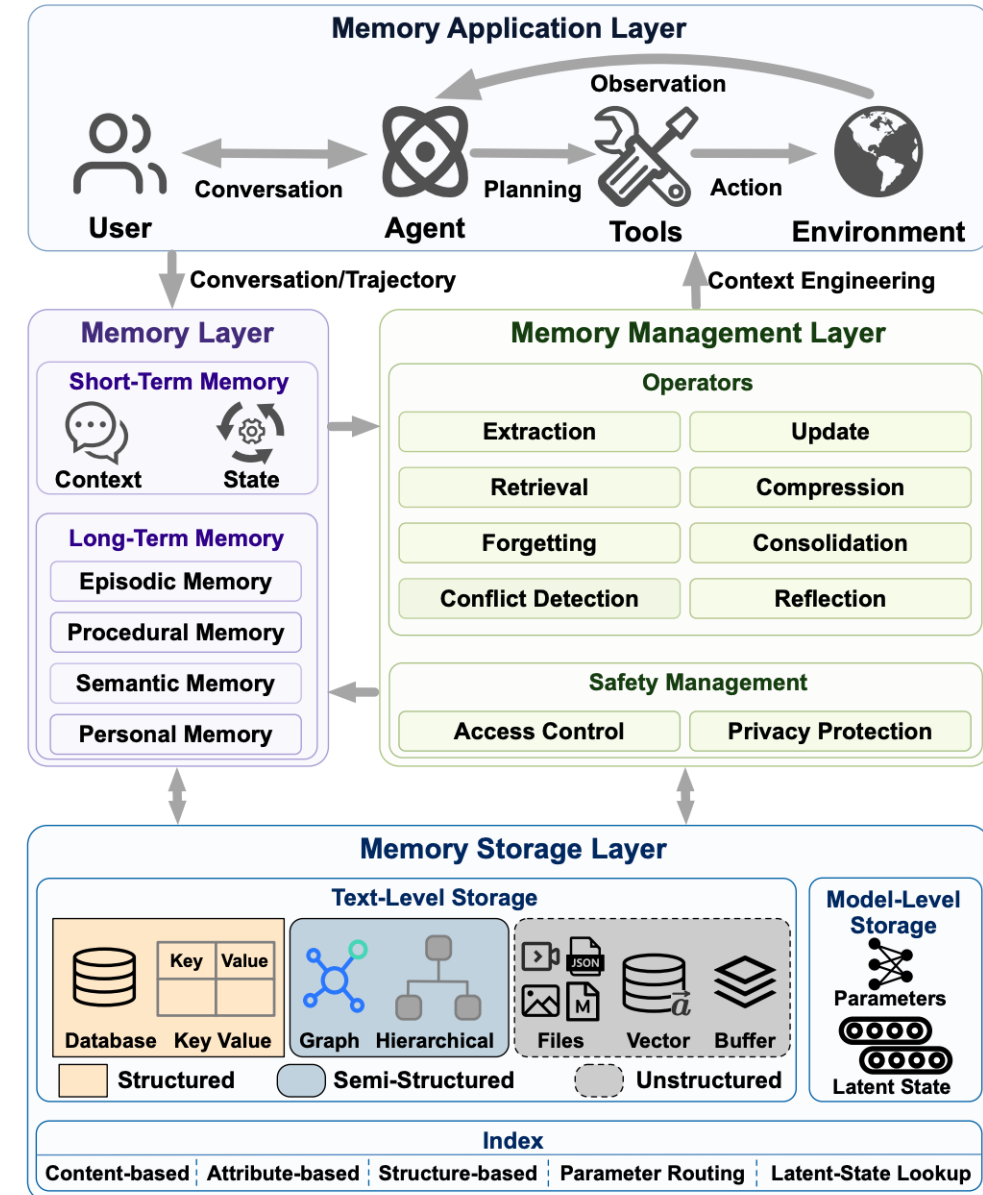
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

❑ Evaluation

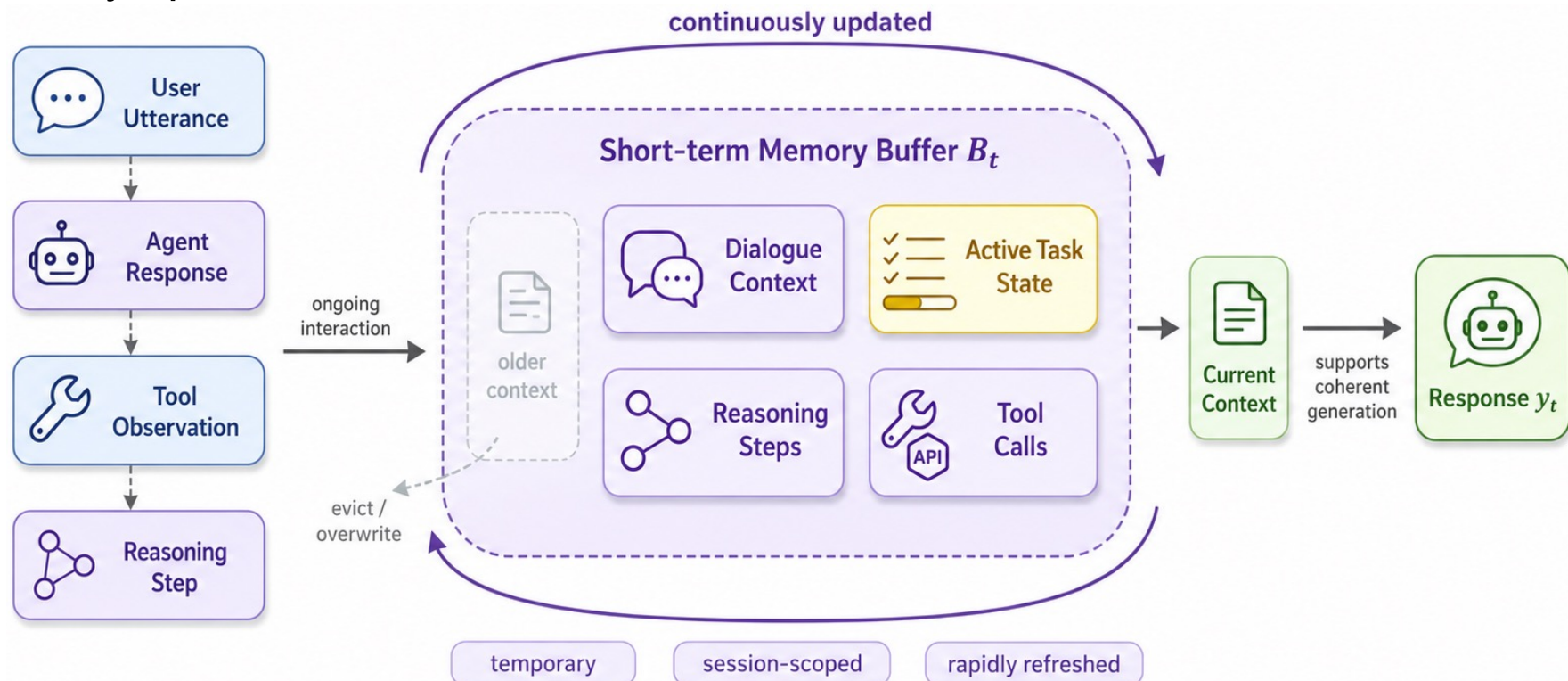
❑ Open Problems



Short-term Memory

- **Short-term memory $\approx$ buffer**

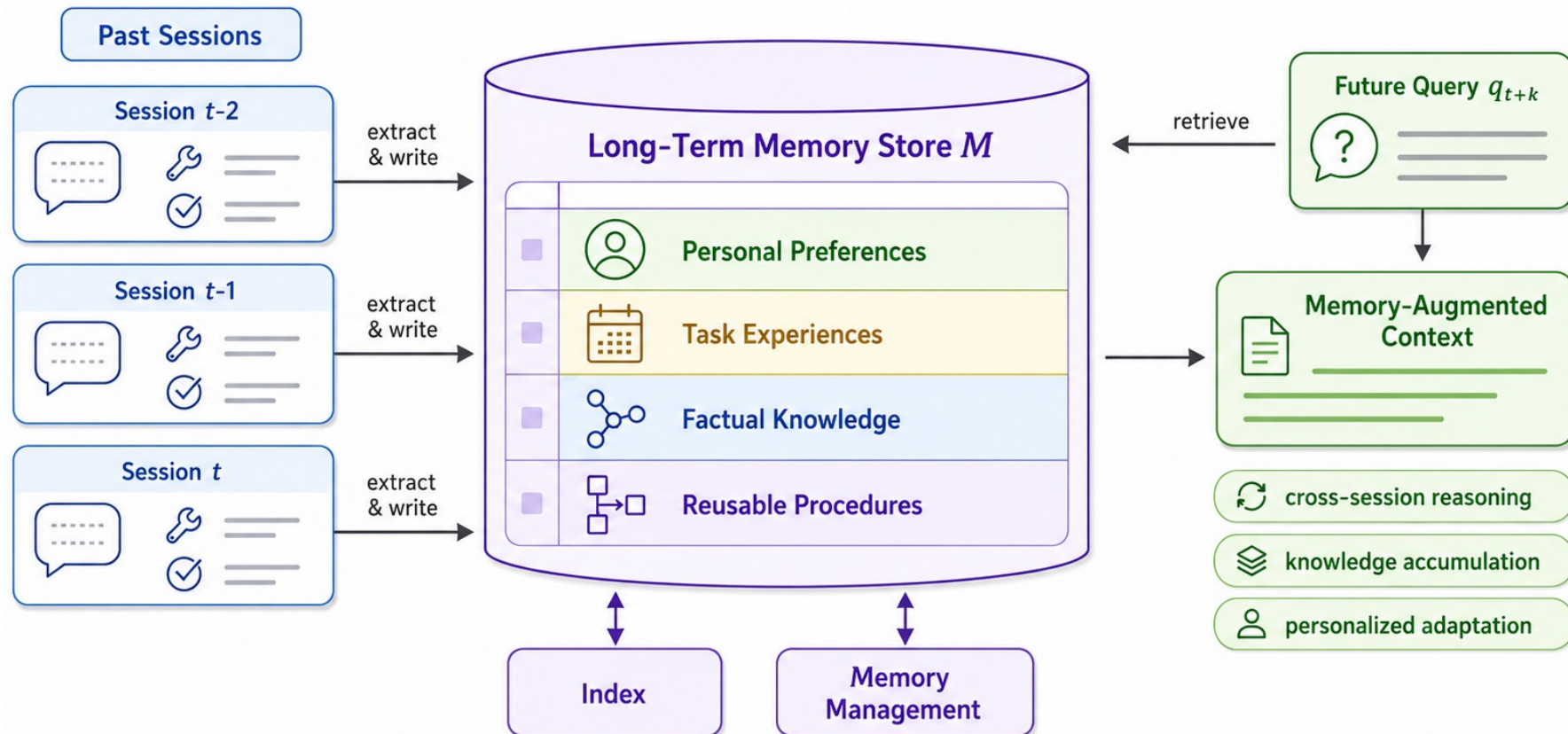
- Holds information relevant to the ongoing user-agent interaction.
- Includes dialogue context, progression of active tasks, reasoning steps, tool calls and others.
- Serves as a temporary workspace, supporting the generation of coherent and contextually appropriate responses.
- Continuously updated as the interaction evolves.



Long-Term Memory

- **Long-term memory $\approx$ persistent table**

- Stores user preferences, task experiences, factual knowledge, and reusable procedures.
- Provides persistent knowledge that can be retrieved to support future queries, enabling cross-session reasoning, knowledge accumulation, and personalized adaptation.
- Requires proper management to keep stored knowledge accurate, organized and up-to-date.



Long-Term Memory

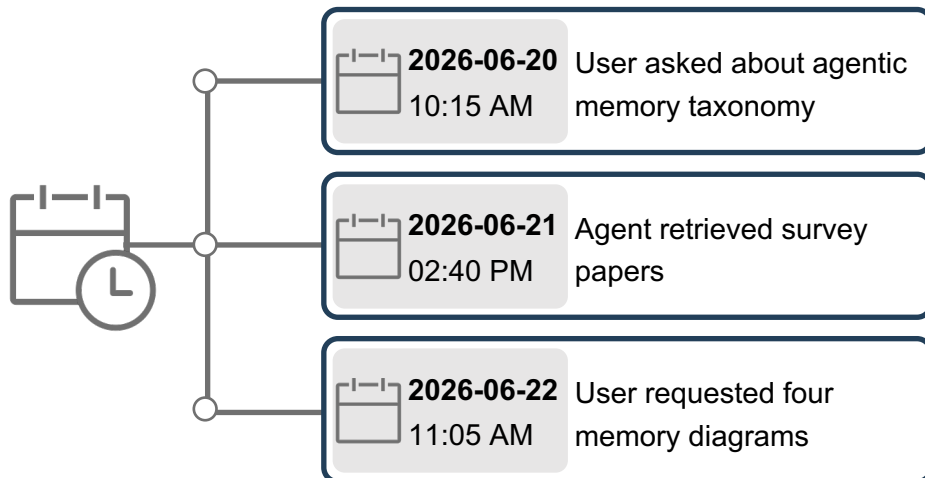
- **Episodic Memory:**

- Records concrete interaction episodes and execution trajectories.
- Supports the retrieval of past outcomes to inform current decision making.

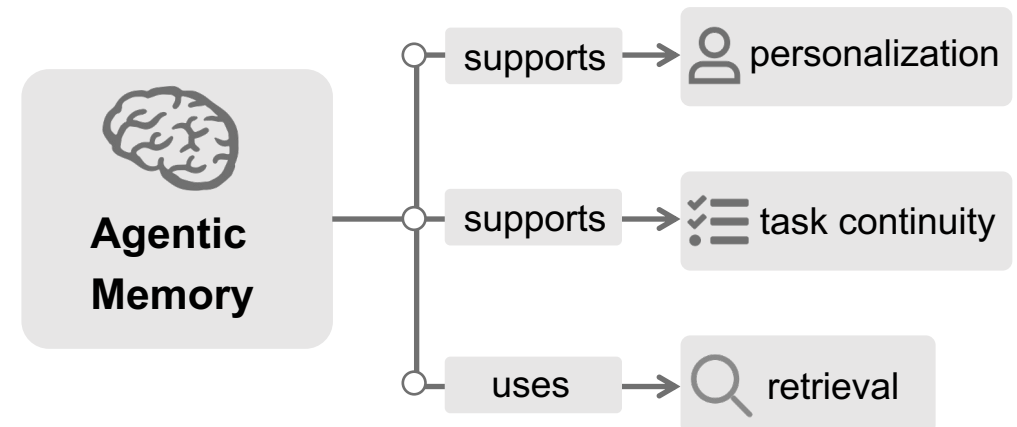
- **Semantic Memory:**

- Encodes abstracted knowledge, facts, and generalized concepts.
- Provides a durable repository of world knowledge and domain-specific concepts.

- **Episodic Memory**



- **Semantic Memory**



Long-Term Memory

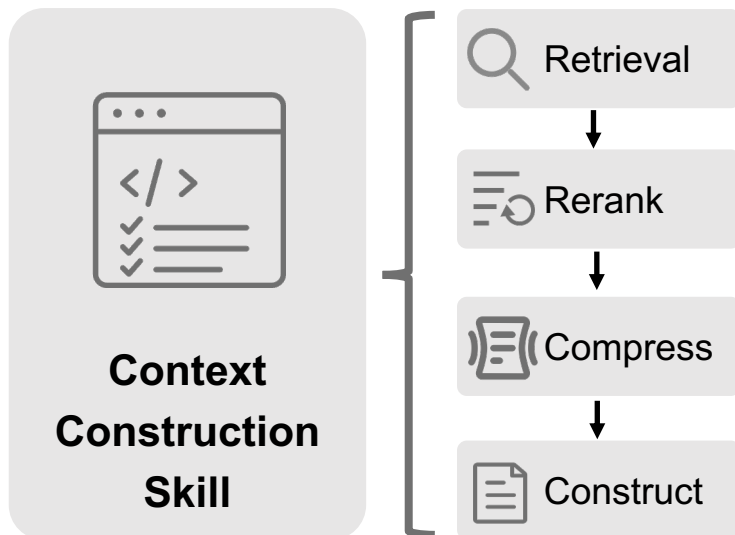
- **Procedural Memory:**

- Stores reusable task procedures, behavioral patterns, and action policies.
- Enables the agent to perform tasks efficiently and replicate learned behaviors.

- **Personal Memory:**

- Retains user-specific information, such as preferences and habits.
- Provides a basis for long-term personalization, supporting user-aware reasoning.

- **Procedural Memory**

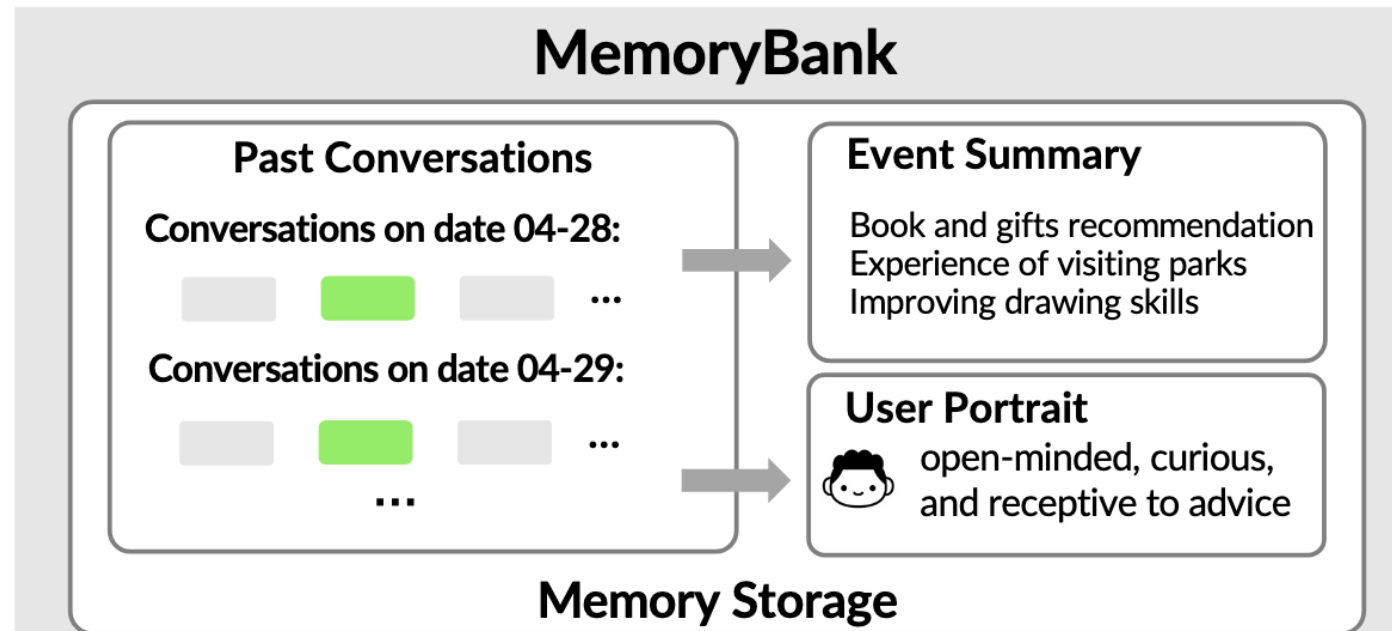


- **Personal Memory**



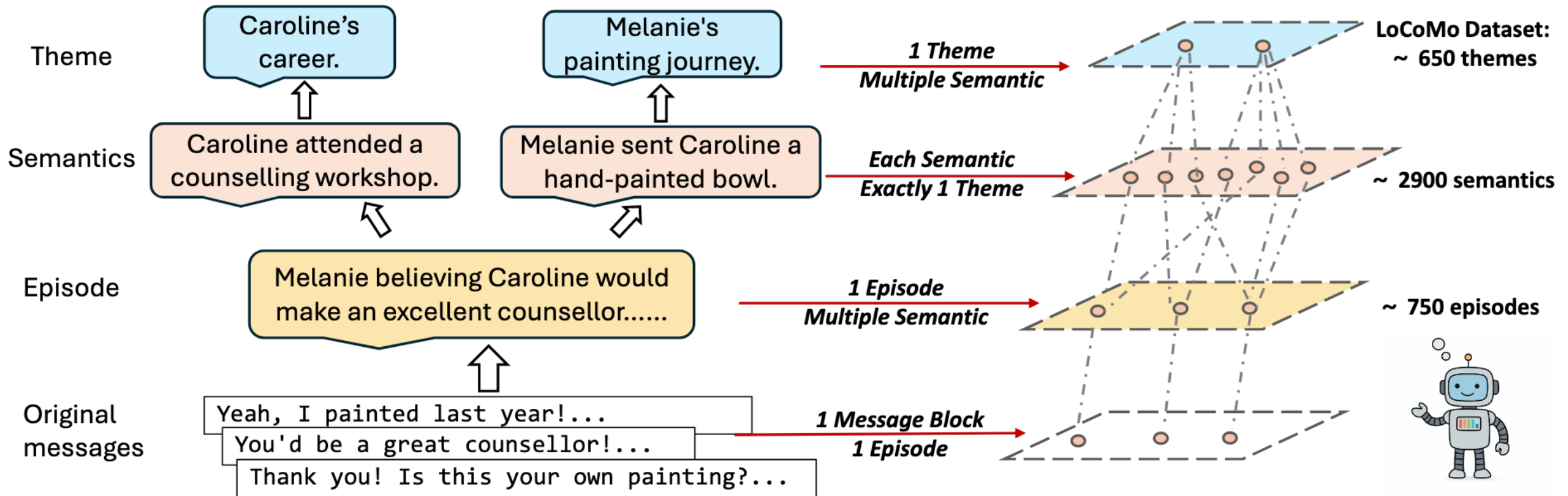
From Short-Term Context to Long-Term Memory

- Different memory types correspond to different **extraction methods**.
- **Episodic Memory:**
 - 1. Organize conversations into daily interaction records.
 - 2. Summarize each day into key events by prompting LLM.
- **Personal Memory:**
 - 1. Analyze daily conversations for user-specific signals using LLM.
 - 2. Aggregate daily insights into a global user portrait.



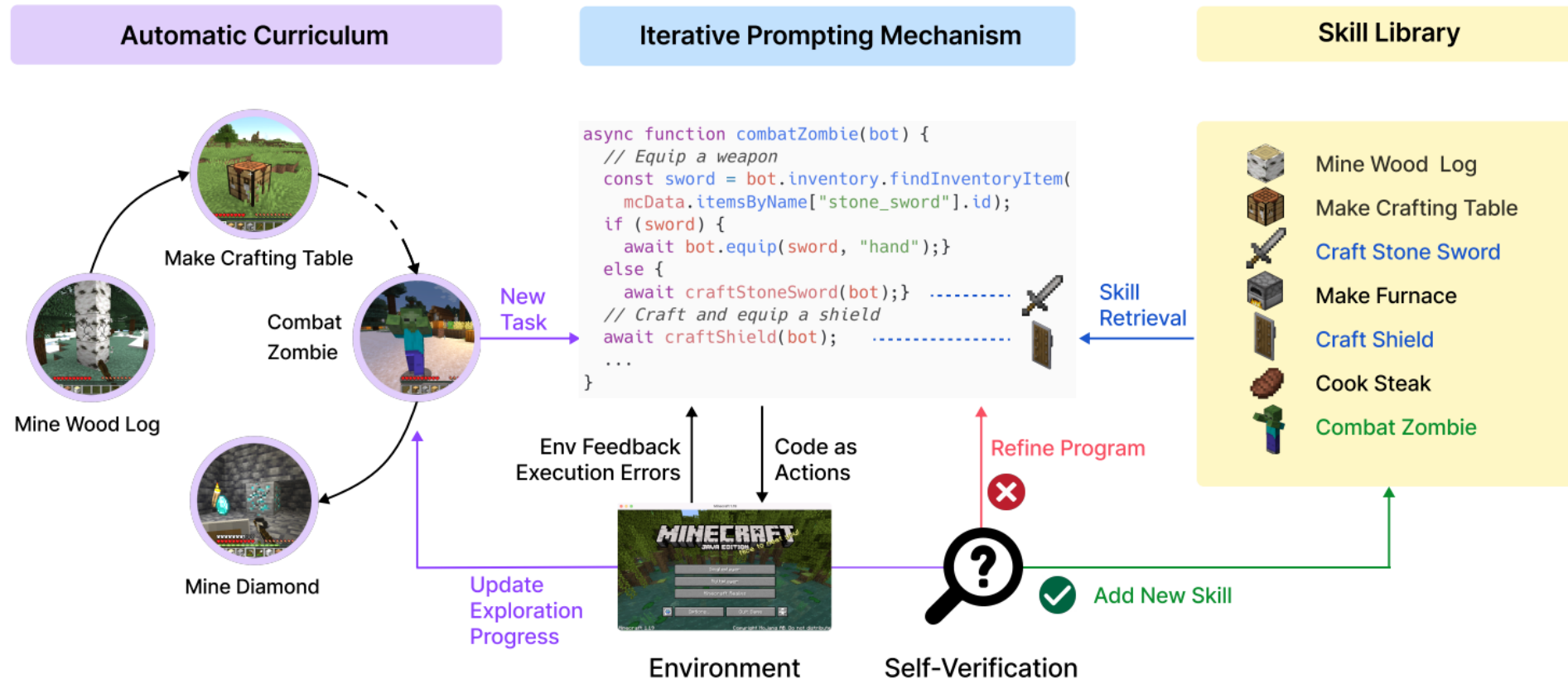
From Short-Term Context to Long-Term Memory

- Different memory types correspond to different **extraction methods**.
- **Semantic Memory:**
 - 1. Filter out low-value content in episodic memory.
 - 2. Extract high-value long-term knowledge using tailored prompts.
 - 3. Maintain source links to recover the original episode when needed.



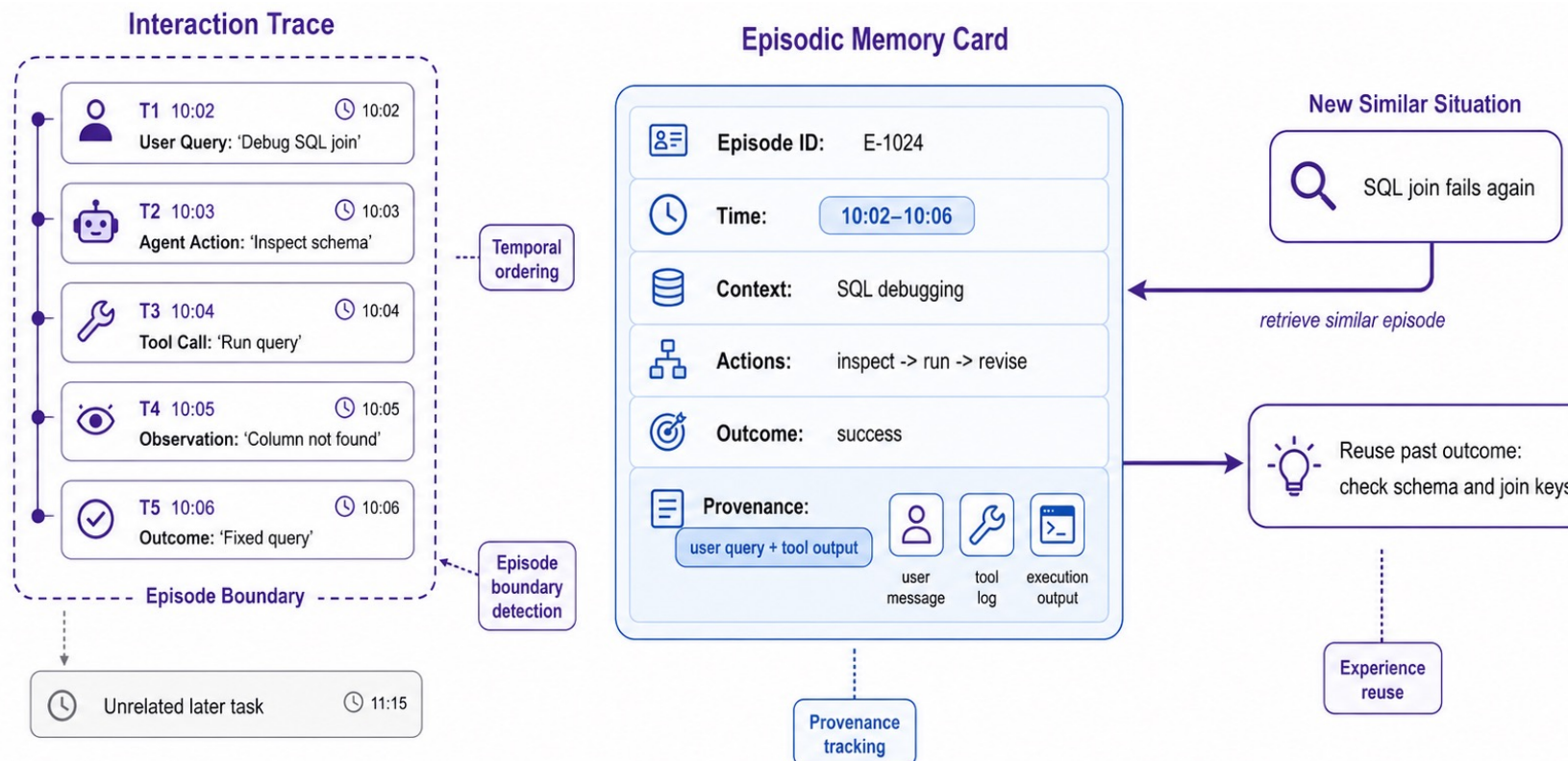
From Short-Term Context to Long-Term Memory

- Different memory types correspond to different **extraction methods**.
- **Procedural Memory:**
 - 1. Generate executable programs for concrete tasks.
 - 2. Iteratively refine programs using runtime feedback and execution errors.
 - 3. Store the verified programs as reusable skills.



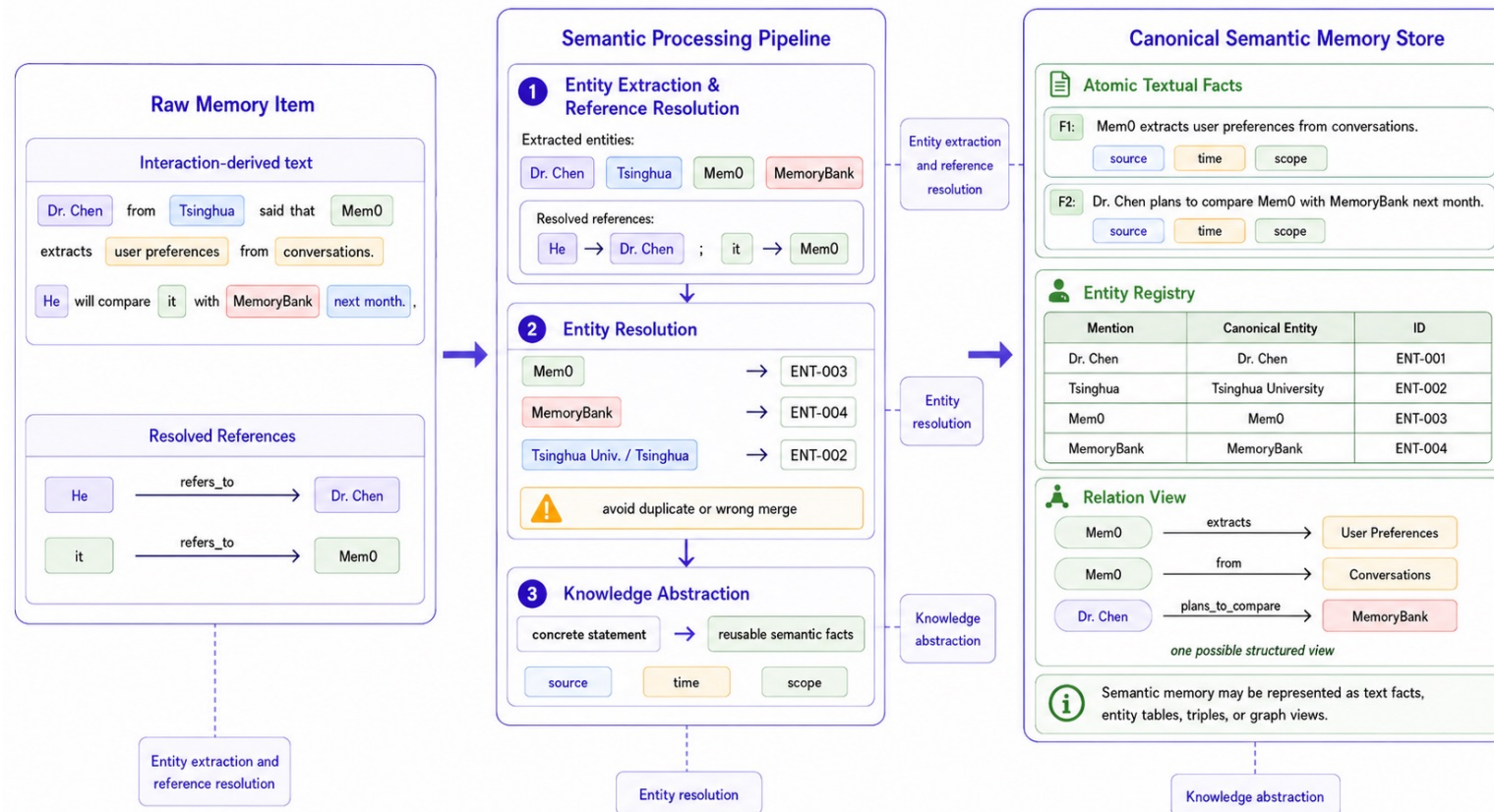
Core Techniques of Episodic Memory

- **Temporal ordering:** The system needs to record the temporal sequence of events.
- **Provenance tracking:** Each memory should be traceable to its original source.
- **Episode boundary detection:** The system must determine which interactions belong to the same episode and which should be separated into distinct task trajectories.
- **Experience reuse:** The agent should retrieve relevant episodes when facing a similar situation.



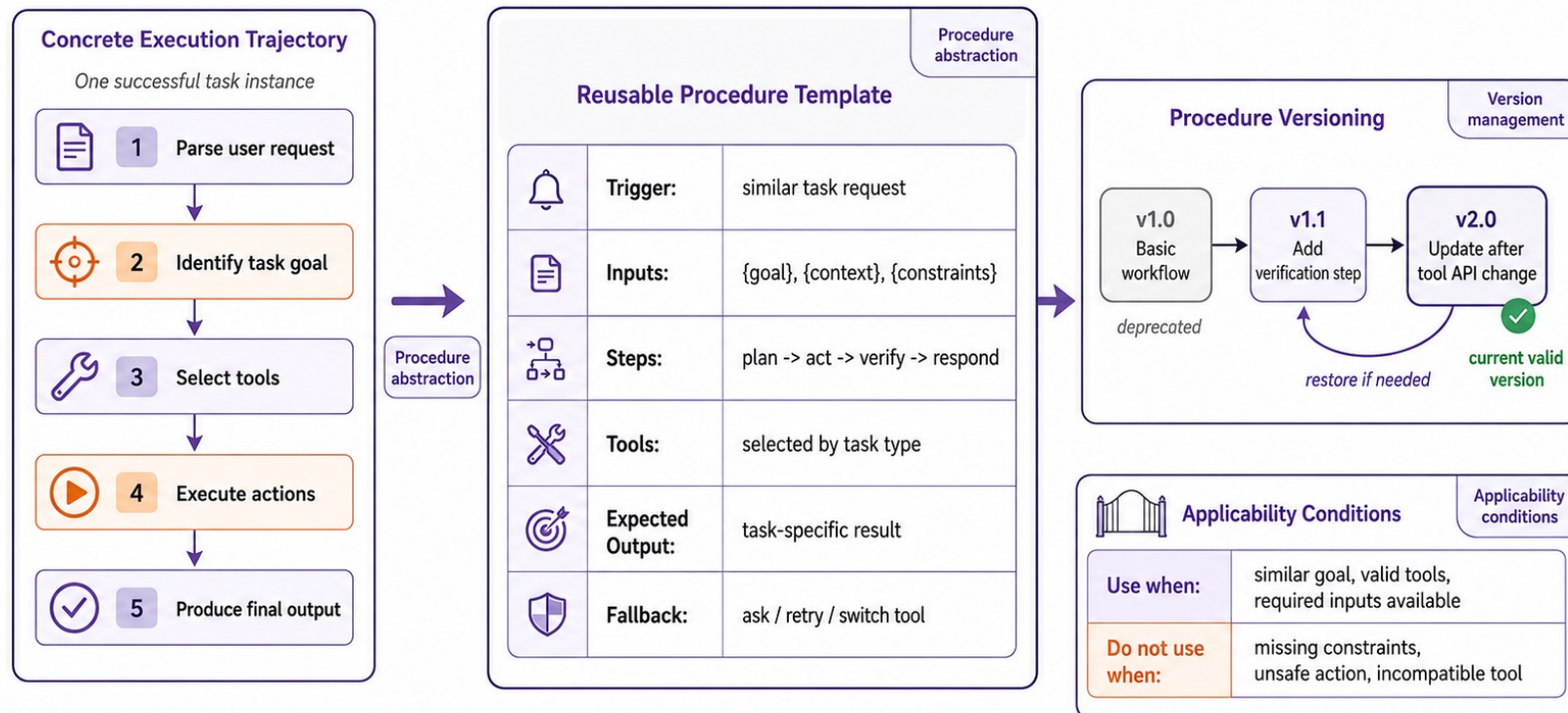
Core Techniques of Semantic Memory

- **Entity extraction and reference resolution:** Identify all salient entities in a memory item and resolve contextual mentions to the correct entities.
- **Entity resolution:** Align newly extracted entities with existing canonical entities in memory.
- **Knowledge abstraction:** Generalize concrete interaction-derived information into reusable semantic knowledge, while preserving the appropriate level of specificity and scope.



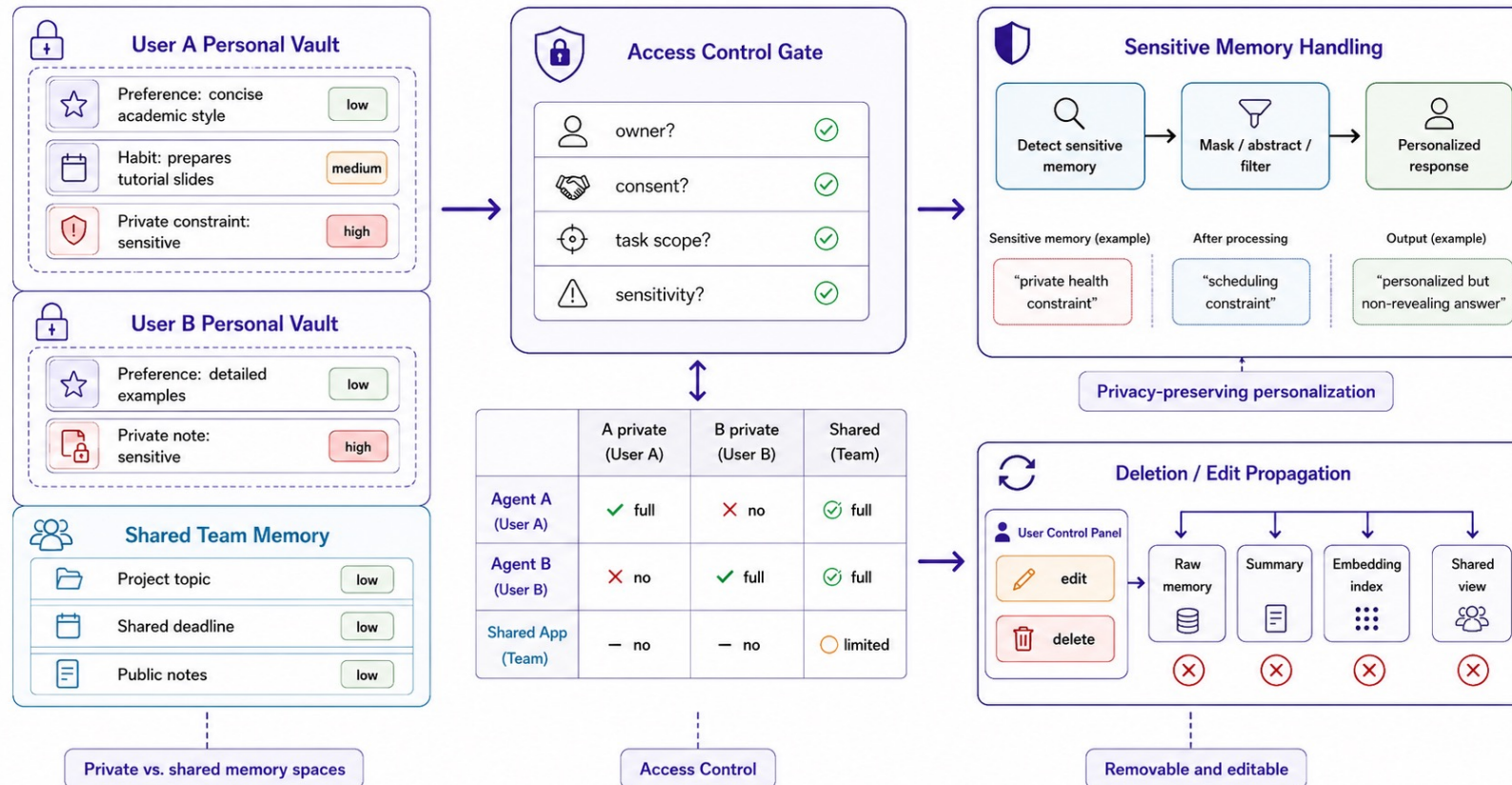
Core Techniques of Procedural Memory

- **Procedure abstraction:** The system needs to extract generalizable procedures from concrete execution trajectories.
- **Version management:** The system must track which version is currently valid and when older versions should be deprecated or restored.
- **Applicability conditions:** Each procedure should specify when it is applicable, what preconditions must hold, and under what circumstances it should not be used.



Core Techniques of Personal Memory

- **Access Control:** The system must determine which personal memories can be accessed by which users, agents, or applications.
- **Removable and editable:** Users should be able to inspect, modify, or delete their memories.
- **Privacy-preserving personalization:** Detect sensitive personal memories and apply appropriate handling strategies before using them for personalization.



Take-aways

- Memory Categorization is **not merely a cognitive taxonomy**. It determines **what to store, how to support** future agent behavior and **how to extract** it.
- Short-Term Memory vs. Long-Term Memory:
 - Short-term memory stores transient context.
 - Long-term Memory stores durable, reusable knowledge distilled from context.

- Long-Term Memory Types:

Types	Functional Roles	Storage Contents	Core Techniques
Episodic Memory	Experience recall	Past events	Temporal ordering, provenance tracking, episode boundary detection, experience reuse
Semantic Memory	Factual reasoning	Entities, facts & relations	Entity extraction and reference resolution, entity resolution, knowledge abstraction
Procedural Memory	Skill reuse	Reusable actions & skills	Procedure abstraction, version management, applicability conditions
Personal Memory	Personalization	User preferences & profiles	Access control, removable and editable, privacy-preserving personalization

- Long-Term Memory formation is type-dependent. Different memory types require different extraction, abstraction aggregation, and refinement processes.

Memory Management

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

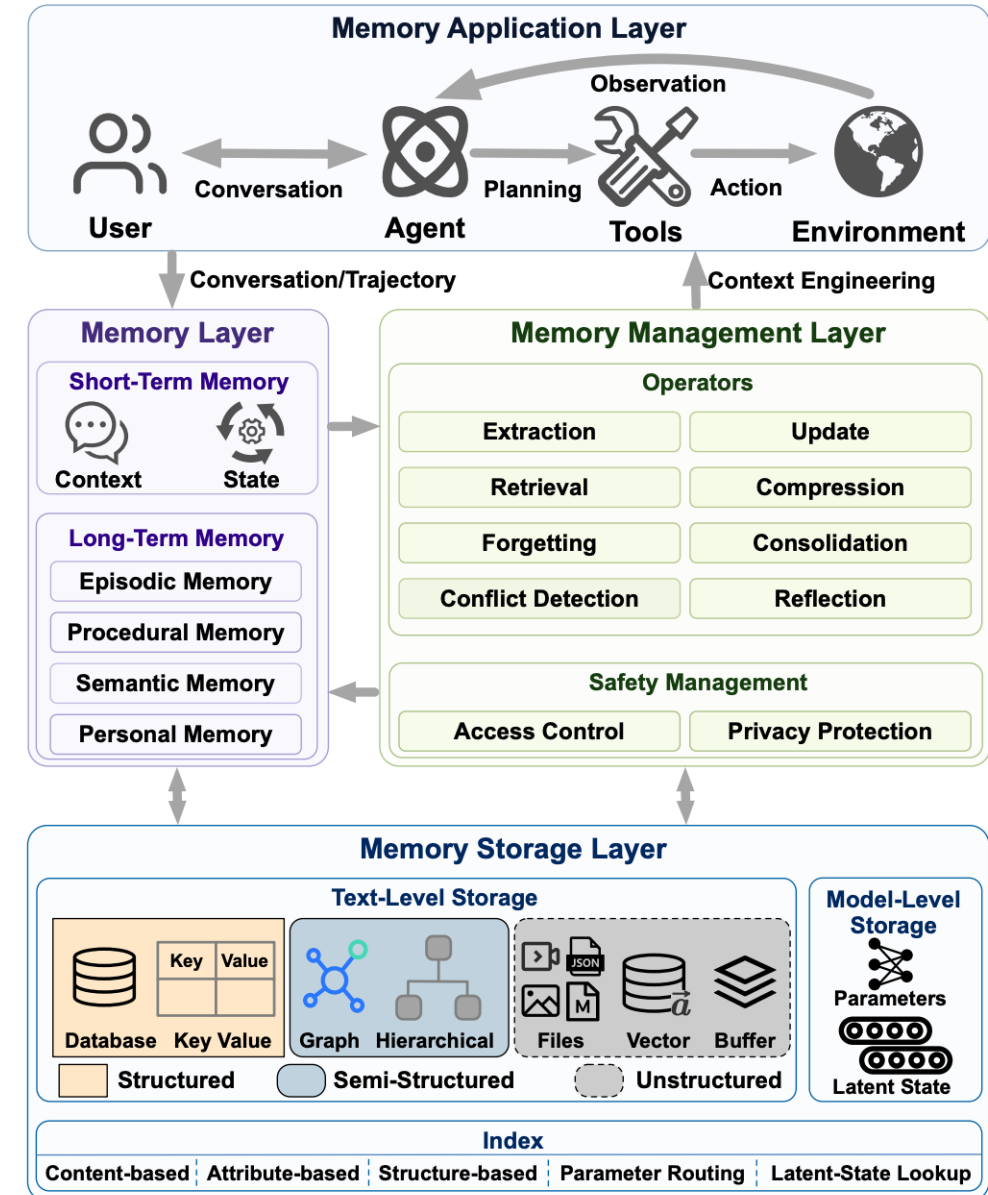
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

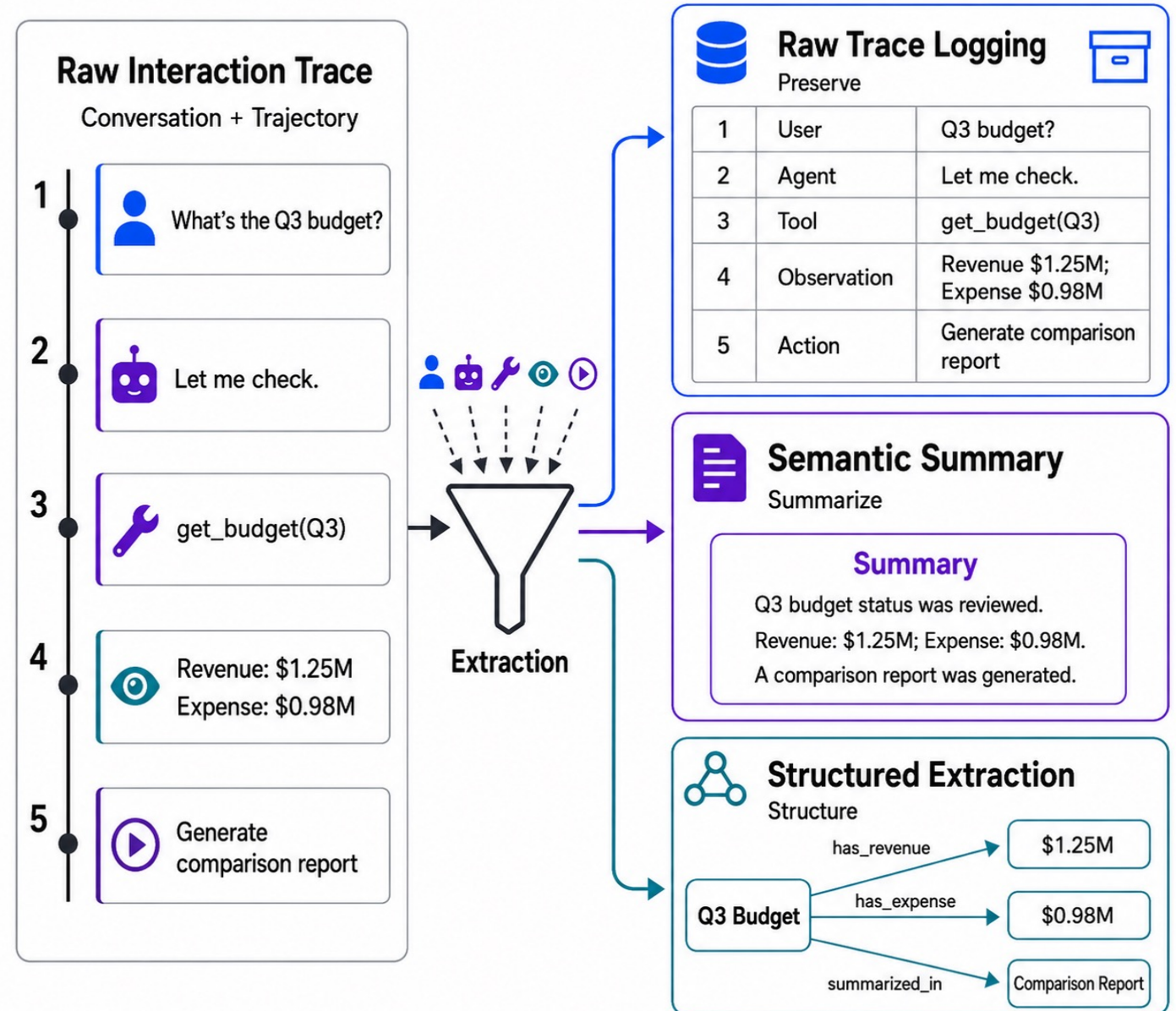
❑ Evaluation

❑ Open Problems



Operator: Extraction

- **Goal:** Extraction **transforms raw interaction traces into reusable memory candidates.**
- **Problem:**
 - Identify what is worth remembering.
 - Determine the appropriate extraction granularity and context scope.
- **Methods:**
 - Raw Trace Logging
 - Semantic Summary
 - Structured Extraction



Extraction: Raw Trace Logging

- **Raw Trace Logging:** This approach directly records conversations, tool calls, observations, or workspace traces as persistent memory logs.
- **Motivation:** Preserve complete interaction history without lossy preprocessing.
- **Challenge:** Define appropriate trace boundaries and granularity while preserving complete interaction context.

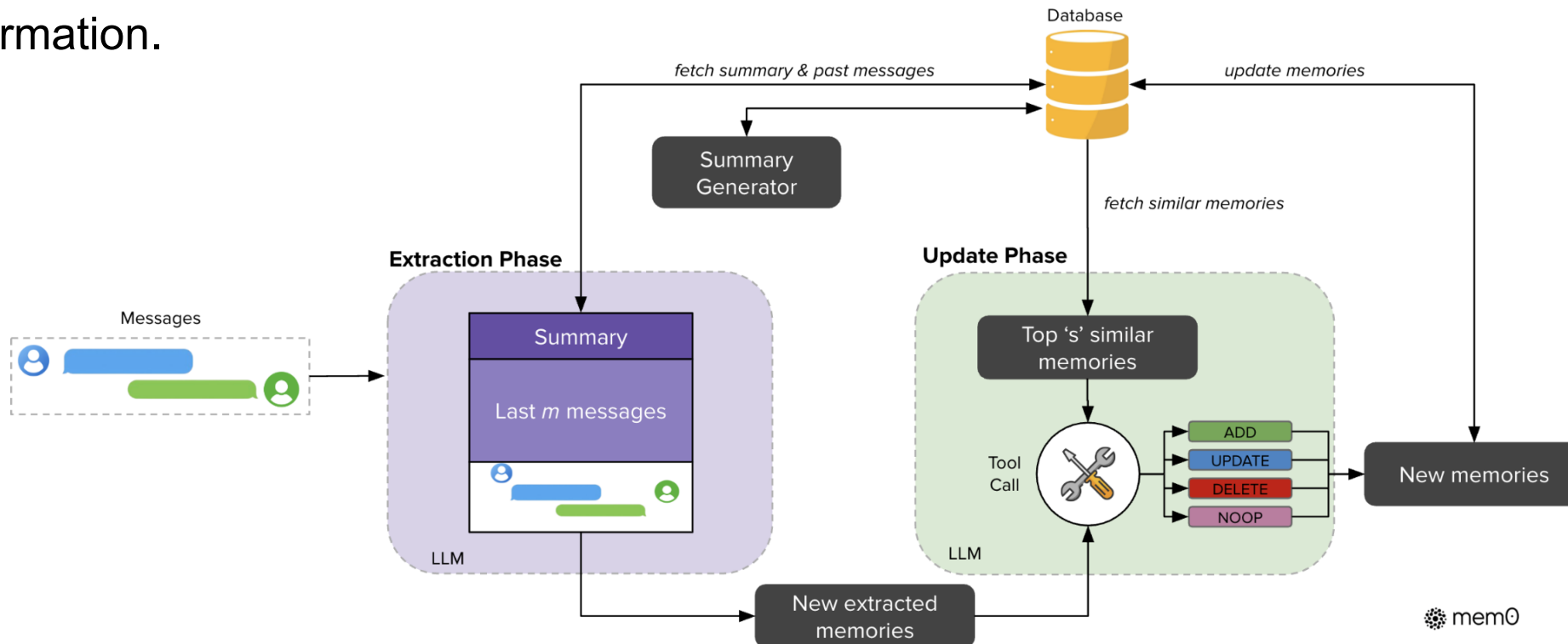
Memory Stream

```
2023-02-13 22:48:20: desk is idle
2023-02-13 22:48:20: bed is idle
2023-02-13 22:48:10: closet is idle
2023-02-13 22:48:10: refrigerator is idle
2023-02-13 22:48:10: Isabella Rodriguez is stretching
2023-02-13 22:33:30: shelf is idle
2023-02-13 22:33:30: desk is neat and organized
2023-02-13 22:33:10: Isabella Rodriguez is writing in her journal
2023-02-13 22:18:10: desk is idle
2023-02-13 22:18:10: Isabella Rodriguez is taking a break
2023-02-13 21:49:00: bed is idle
2023-02-13 21:48:50: Isabella Rodriguez is cleaning up the
kitchen
2023-02-13 21:48:50: refrigerator is idle
2023-02-13 21:48:50: bed is being used
2023-02-13 21:48:10: shelf is idle
2023-02-13 21:48:10: Isabella Rodriguez is watching a movie
2023-02-13 21:19:10: shelf is organized and tidy
2023-02-13 21:18:10: desk is idle
2023-02-13 21:18:10: Isabella Rodriguez is reading a book
2023-02-13 21:03:40: bed is idle
2023-02-13 21:03:30: refrigerator is idle
2023-02-13 21:03:30: desk is in use with a laptop and some papers
on it
```

...

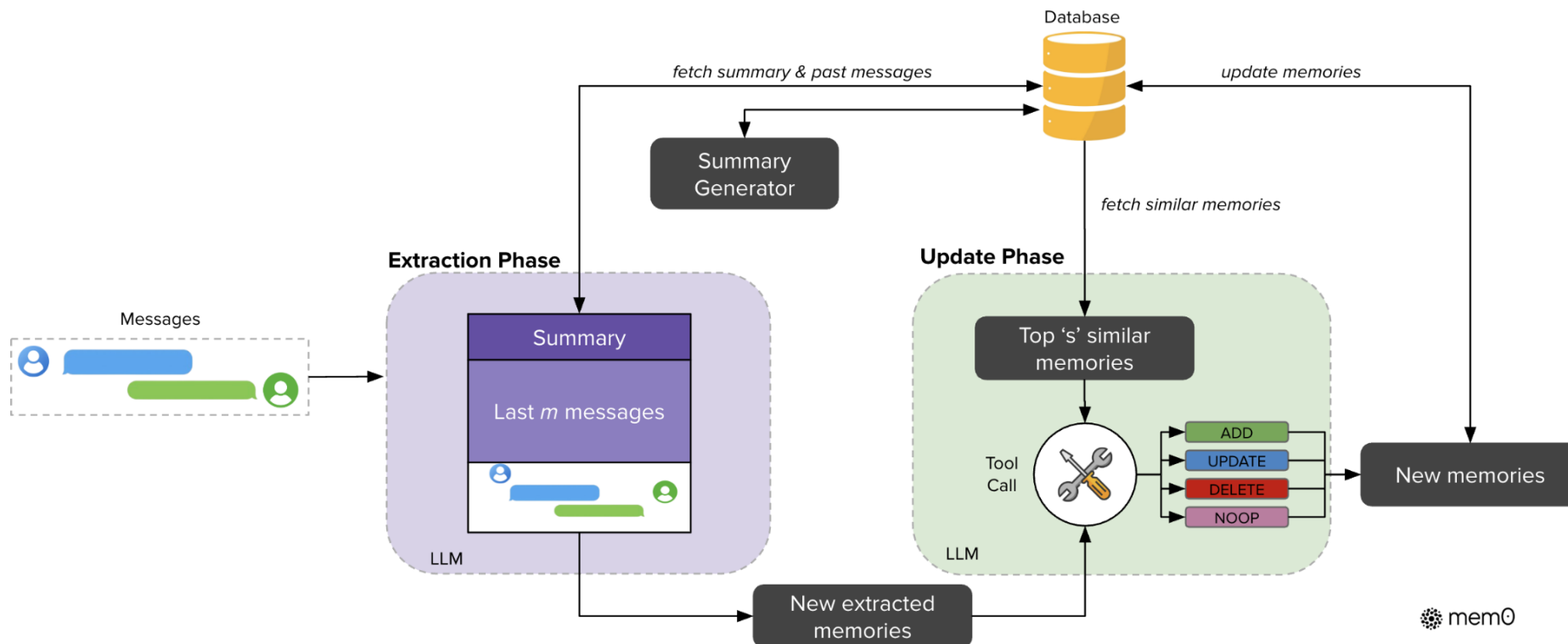
Extraction: Semantic Summary

- **Semantic Summary: Compress** raw dialogue traces into **compact natural-language memory entries** for subsequent storage, update, and retrieval.
- **Motivation:** Raw interaction traces contain substantial noise and redundancy, increasing storage and retrieval costs while obscuring task-relevant information.
- **Challenge:** Generate a faithful summary at the appropriate granularity without discarding potentially useful information.



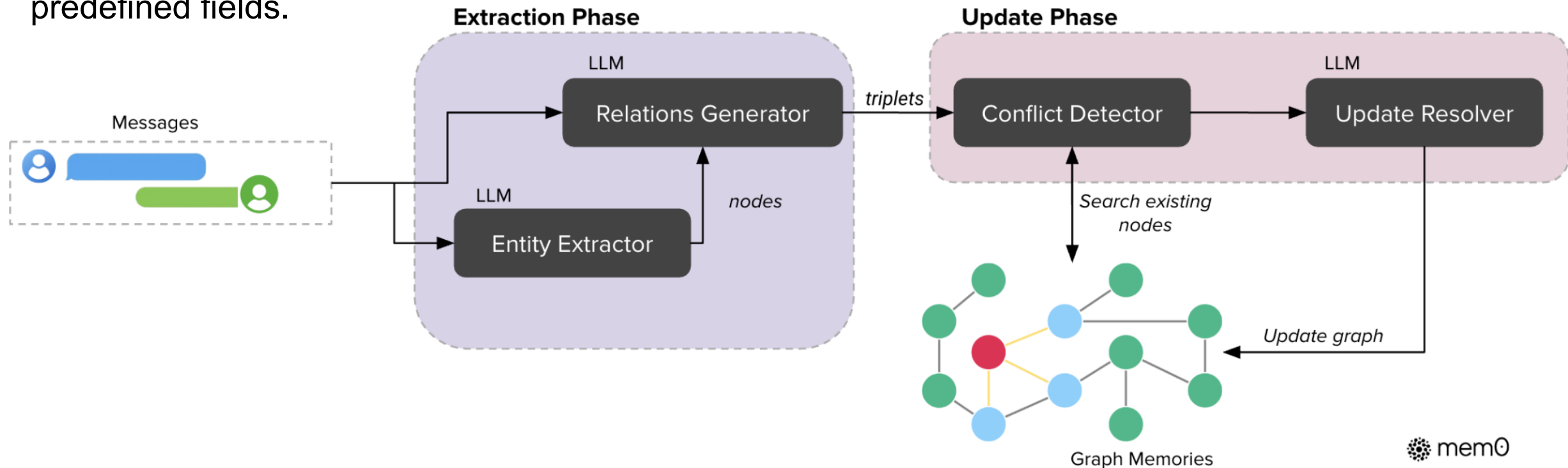
Extraction: Semantic Summary

- **Semantic Summary: Compress** raw dialogue traces into **compact natural-language memory entries** for subsequent storage, update, and retrieval.
 - 1. Maintain an asynchronous global summary.
 - 2. Combine the last m messages.
 - 3. Extract salient natural-language memories.



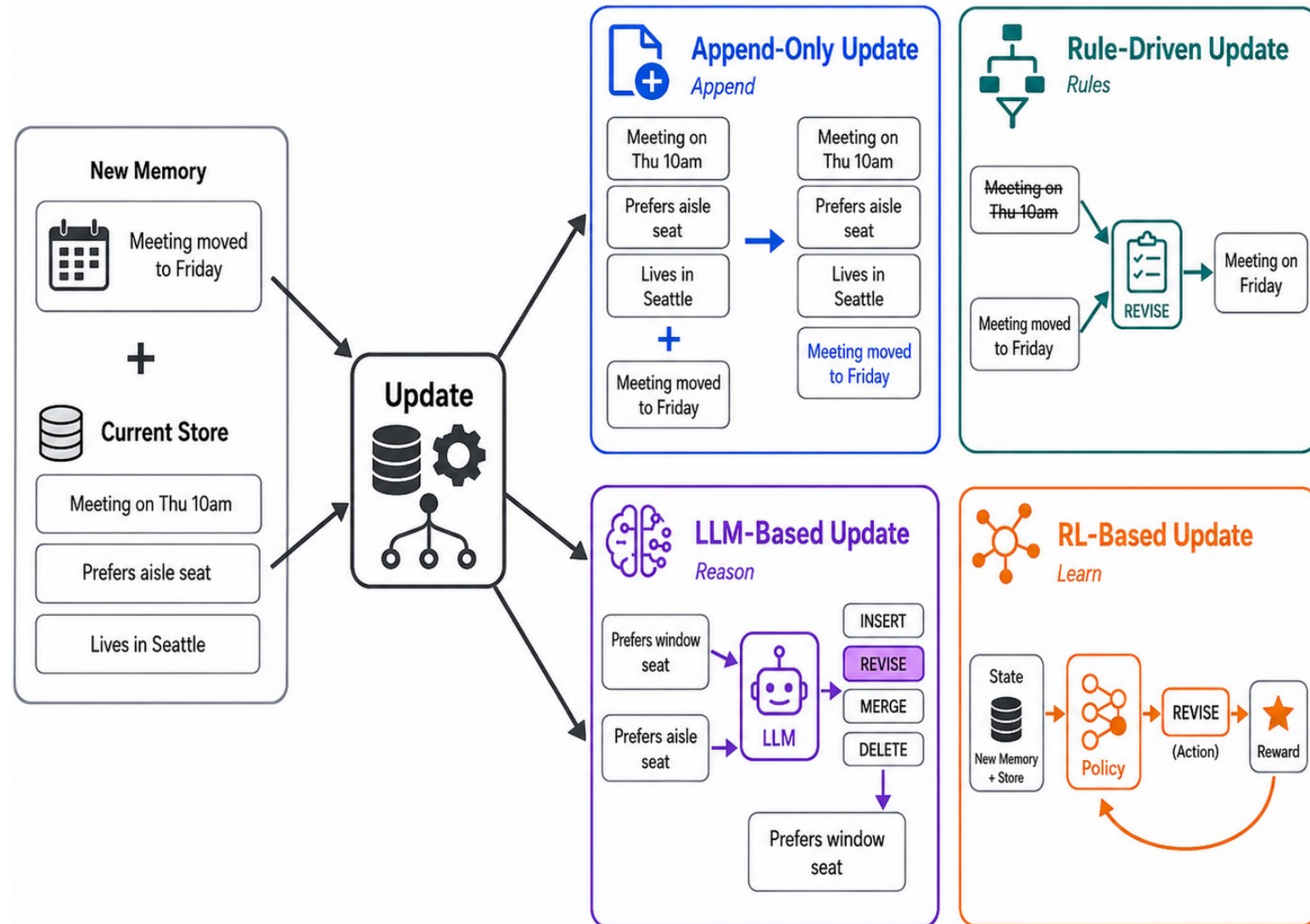
Extraction: Structured Extraction

- **Structured Extraction:** Map raw interaction traces into **schema-conformant memory records**, enabling validation, symbolic storage, and structured retrieval.
- **Motivation:** NL-form summaries offer limited control over what to extract and at what granularity, limiting consistent organization and precise retrieval.
- **Challenge:** Design an expressive yet generalizable schema and reliably map diverse information into predefined fields.



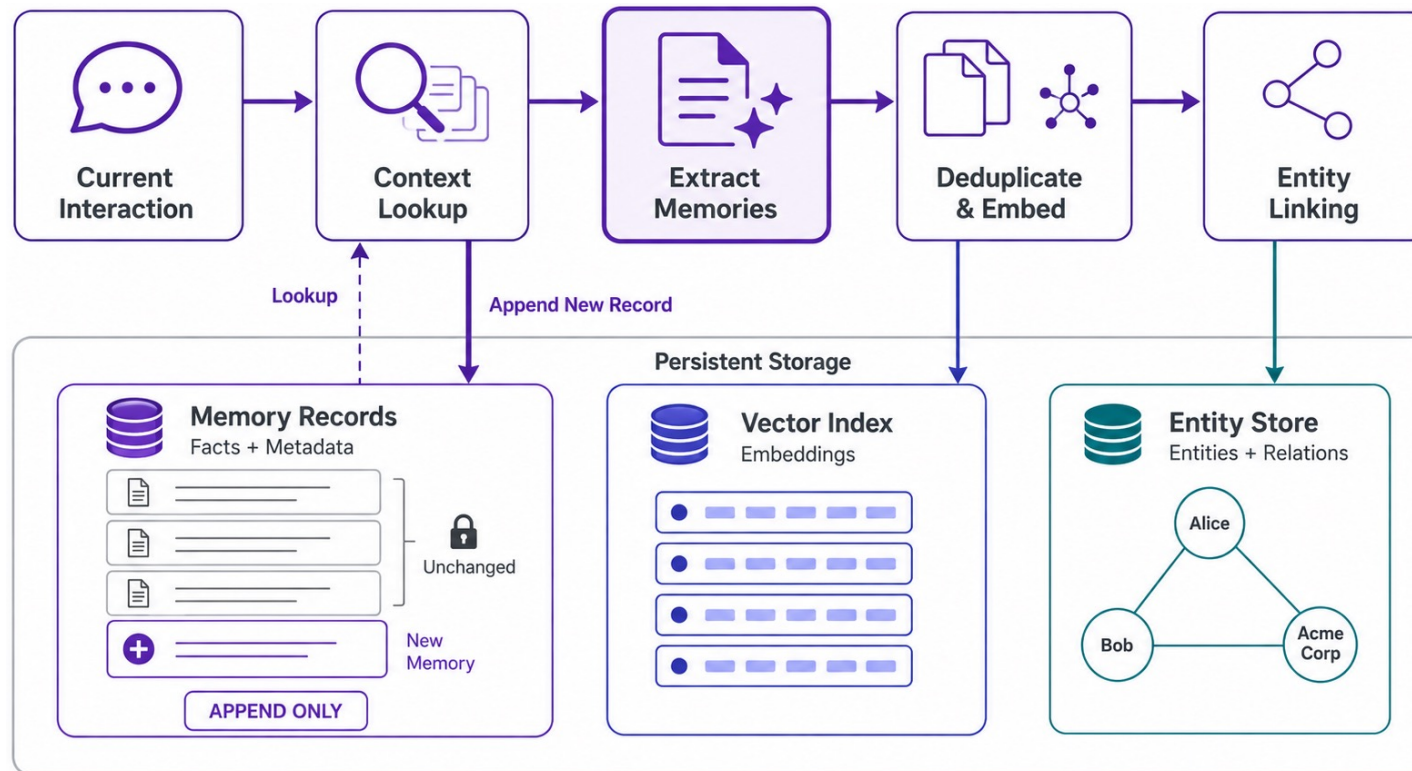
Operator: Update

- **Goal:** Update applies state-changing operations to the memory store, such as insertion, revision, merging, invalidation, or deletion, to maintain a coherent and up-to-date memory state.
- **Core Challenge:** Decide how to transform the current memory store when triggered.
- **Methods:**
 - Append-Only Update
 - Rule-Driven Update
 - LLM-Based Update
 - RL-Based Update



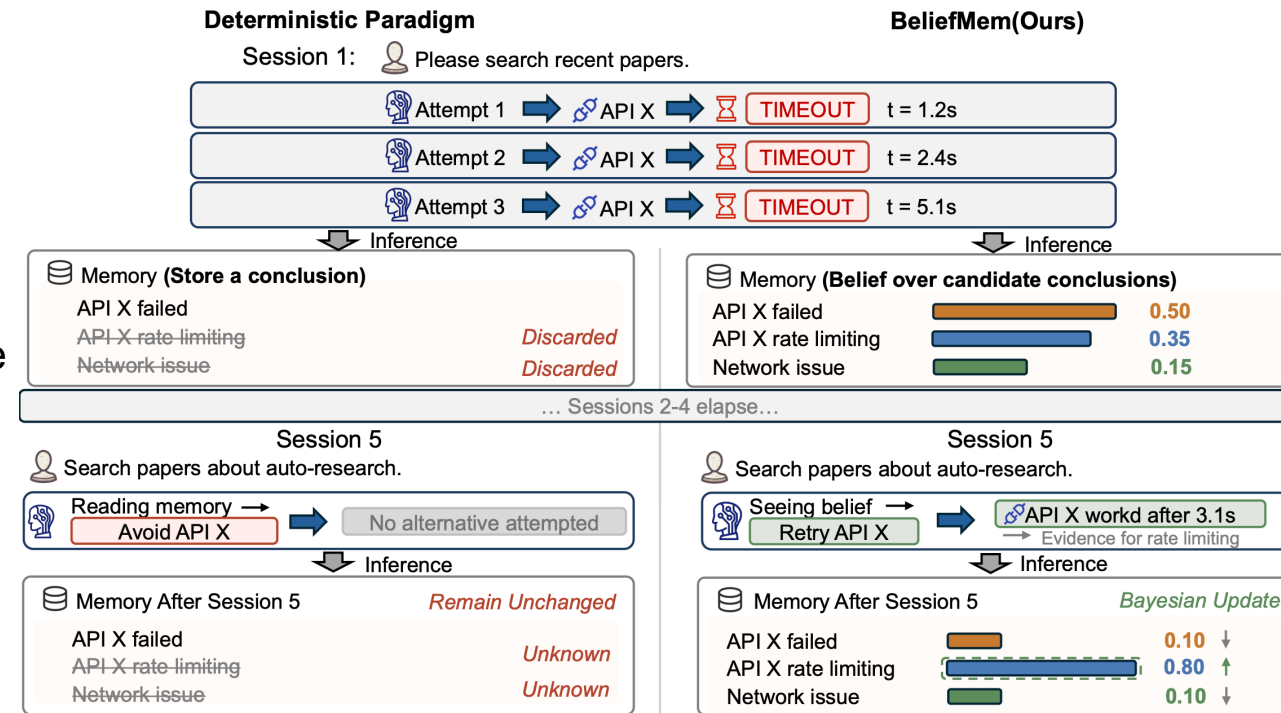
Update: Append-Only Update

- **Append-Only Update:** Append each extracted memory as an independent record without modifying existing memory.
- **Motivation:** Destructive updates may overwrite historical evidence and obscure how information evolves over time.
- **Challenge:** Identify the valid current state from coexisting and potentially conflicting memory records.



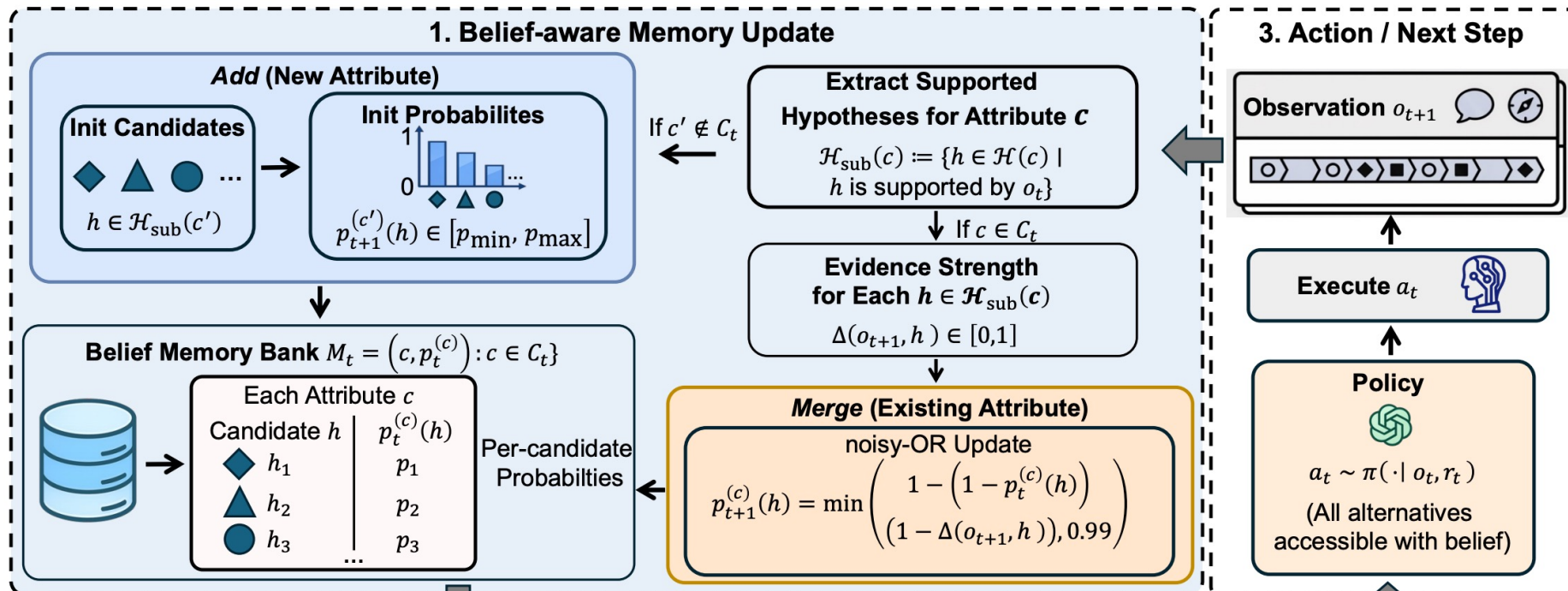
Update: Rule-Driven Update

- **Rule-Driven Update:** Transform the memory store according to predefined rules to determine update.
- **Motivation:**
 - Blindly appending new memories accumulates redundant, outdated, or conflicting records, increasing storage and retrieval overhead.
 - Storing only a single deterministic conclusion collapses the uncertainty in partial and noisy observations, discarding plausible alternatives and causing errors to reinforce over time.
- **Challenge:** Define robust that accurately assess the relationship between new and existing memories and select the appropriate update action.



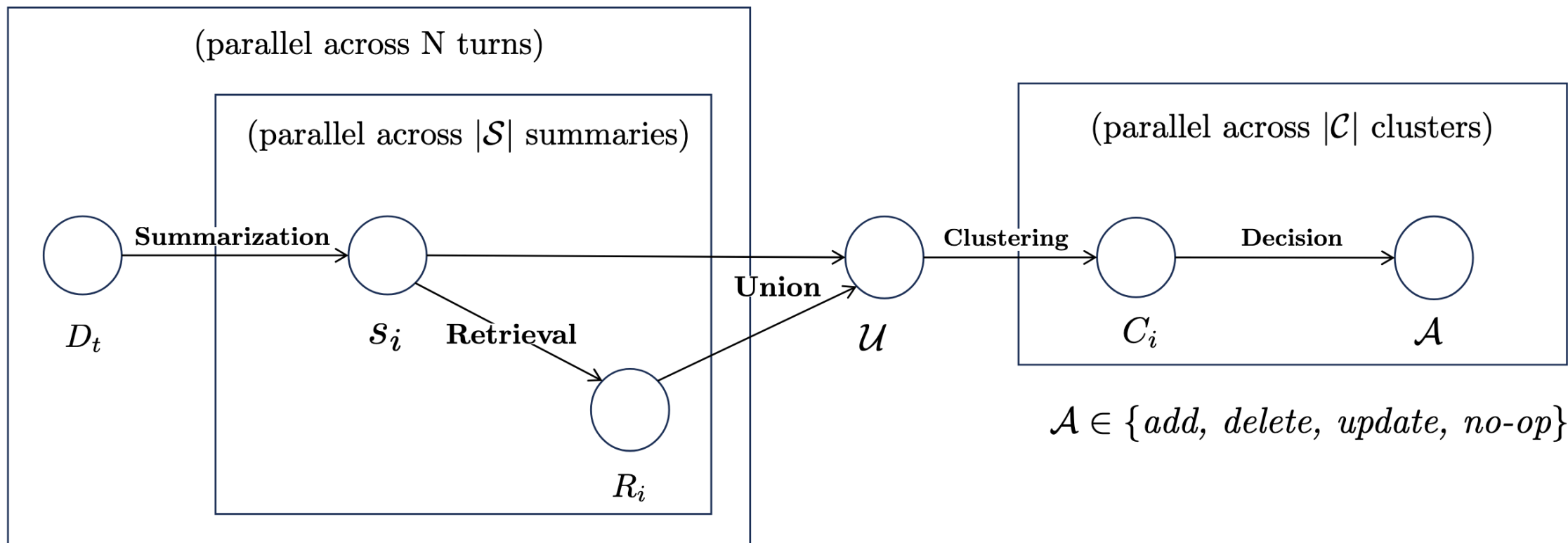
Update: Rule-Driven Update

- **Rule-Driven Update:** Transform the memory store according to predefined rules to determine update.
 - 1. Construct an attribute from stable semantic slots and maintain multiple candidate conclusions.
 - 2. Insert a new entry when an attribute is not existed, initializing its confidence with $[0.7, 0.9]$.
 - 3. When new evidence supports an existing candidate, attach the evidence and update its confidence through Noisy-OR, capped at 0.99.
 - 4. Preserve competing candidates, reducing contradicted ones to 0.25 while archiving prior scores.



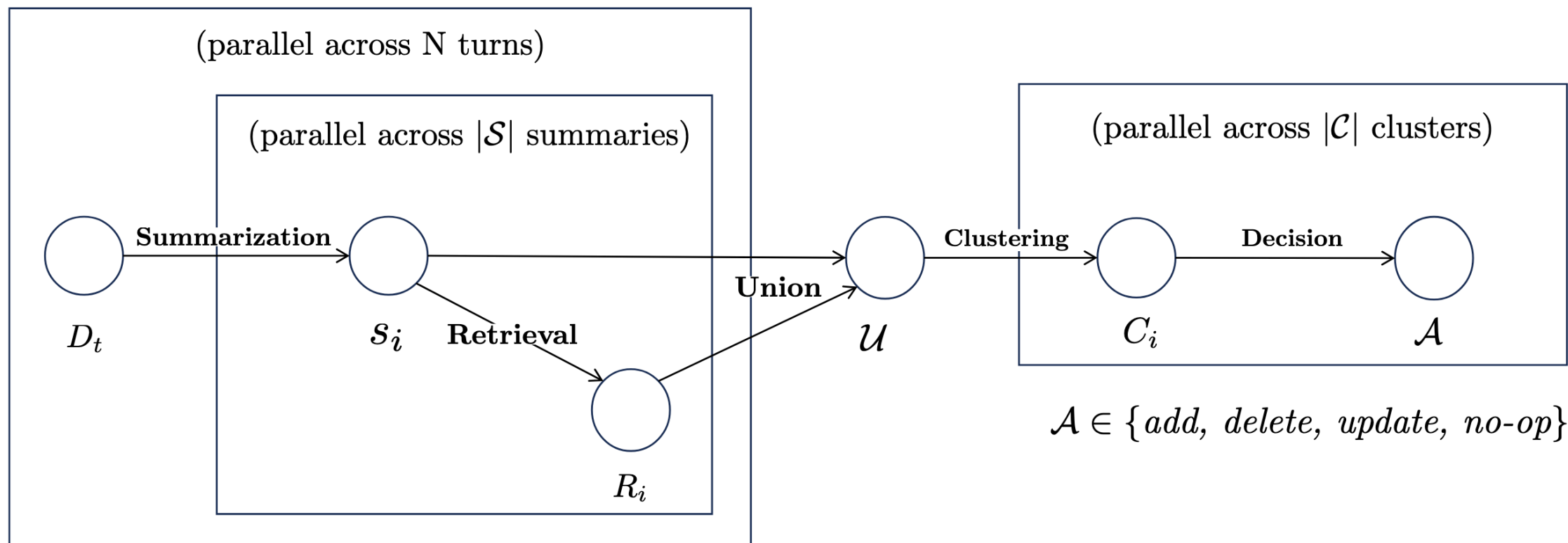
Update: LLM-Based Update

- **LLM-Based Update:** Retrieve relevant memories and prompt an LLM to select or generate operations.
- **Motivation:** Fixed heuristics are brittle and task-specific, whereas LLMs can flexibly interpret contextual memory relations and select update actions across diverse cases.
- **Challenge:**
 - **Cost Efficiency:** Minimize LLM evaluations.
 - **Decision Reliability:** Prevent erroneous update actions.



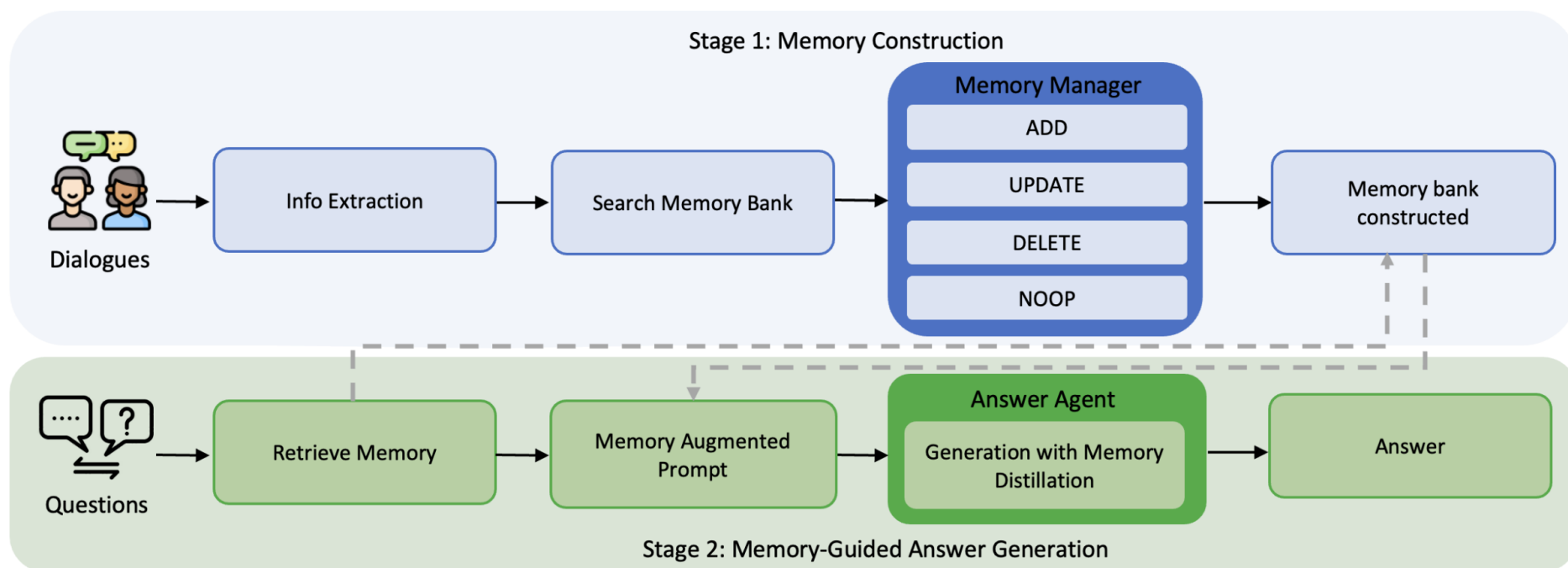
Update: LLM-Based Update

- **LLM-Based Update:** Retrieve relevant memories and prompt an LLM to select or generate operations.
 - 1. Organize semantically related items into temporally clusters.
 - 2. Assign ADD, UPDATE, DELETE, or NO-OP actions to update each cluster.
 - 3. Re-embed refined nodes and locally rebuild affect graph connections.



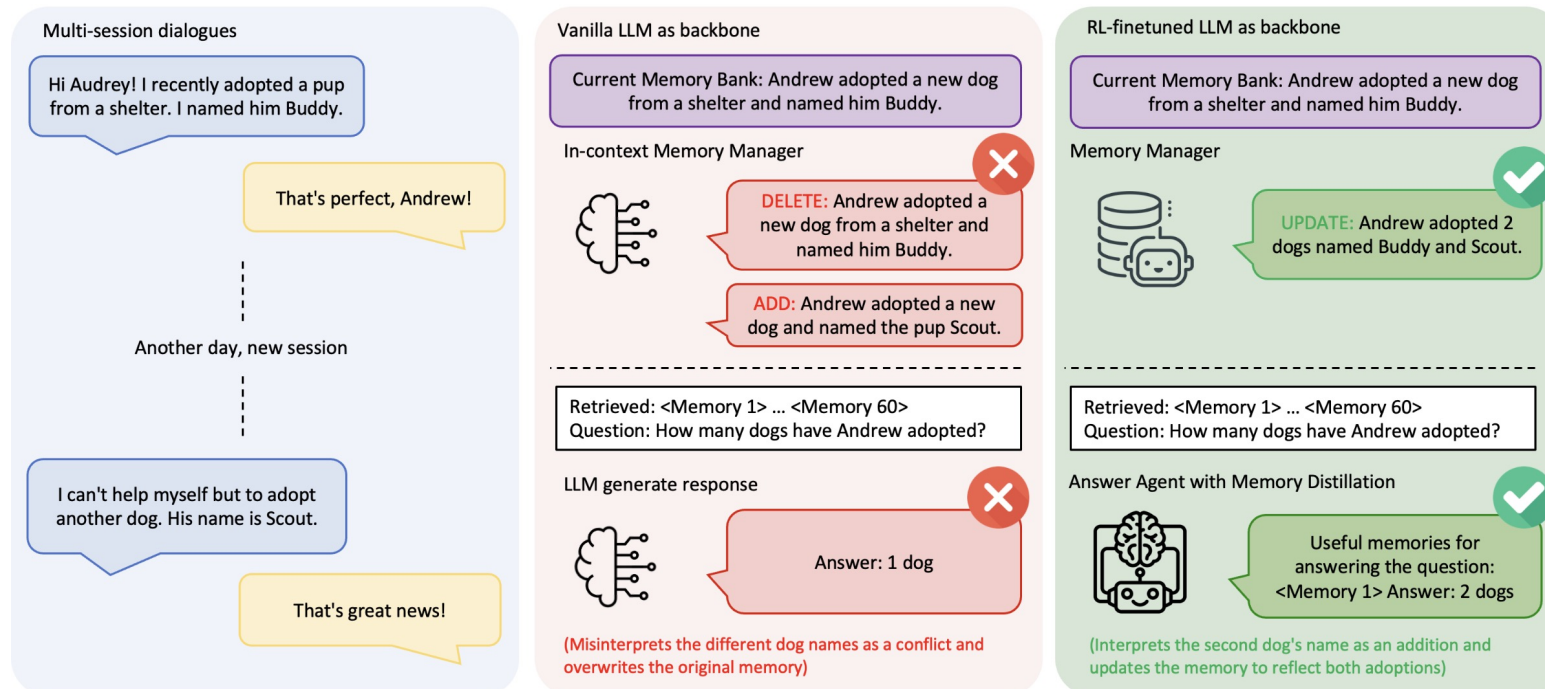
Update: RL-Based Update

- **RL-Based Update:** Finetune a memory-management policy to select better memory mutations.
- **Motivation:** Prompt-driven LLM updaters lack task-grounded learning signals, leading to sub-optimal decisions. RL enables adaptive update policies to be learned directly from task outcomes.
- **Challenge:**
 - **Reward Design:** Construct informative signals that reflect the utility of memory updates.
 - **Credit Assignment:** Attributed delayed downstream outcomes to the responsible update decisions.



Update: RL-Based Update

- **RL-Based Update:** Finetune a memory-management policy to select better memory mutations.
 - **1.** Model the Memory Manager as a policy π_{θ} . $(o, m') \sim \pi_{\theta}(\cdot | x, M_{old})$
 - x : extracted information
 - o : operation
 - M_{old} : retrieved memories
 - m' : updated content
 - **2.** Define memory update actions: ADD / UPDATE / DELETE / NOOP.
 - **3.** Use downstream QA Correctness as reward. $R_{answer} = EM(y_{pred}, y_{gold})$



Update: RL-Based Update

- **RL-Based Update:** Finetune a memory-management policy to select better memory mutations.

- **4. Finetune the Memory Manager with PPO / GRPO**

$$\mathcal{J}(\theta) = \mathbb{E} \left[\min \left(\rho_{\theta} A, \text{clip}(\rho_{\theta}, 1 - \epsilon, 1 + \epsilon) A \right) \right]$$

- $\rho_{\theta} = \frac{\pi_{\theta}(o, m' | x, M_{old})}{\pi_{old}(o, m' | x, M_{old})}$: the importance ratio

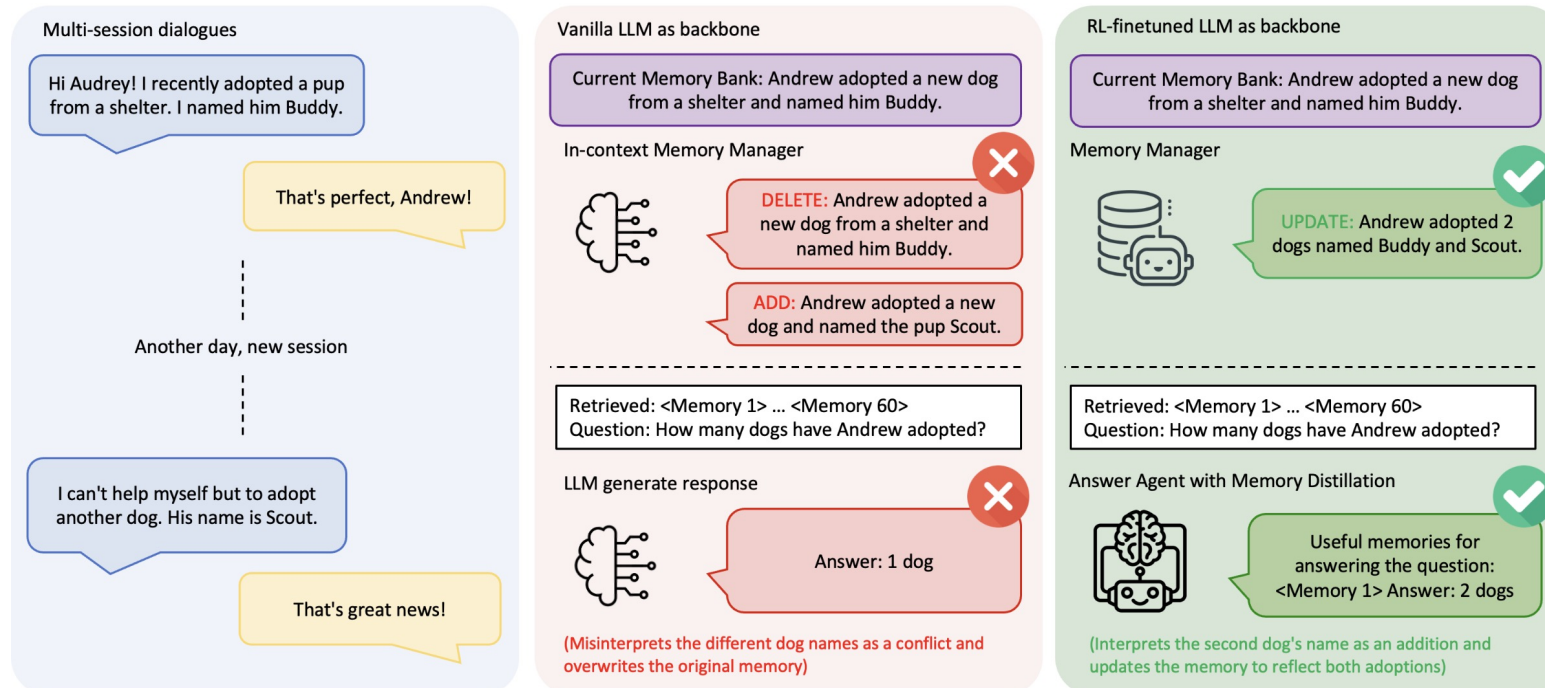
- A : the advantage estimated from the reward

- ϵ : clipping threshold

$$\mathcal{J}(\theta) = \mathbb{E} \left[\frac{1}{G} \sum_{i=1}^G \rho_{\theta}^{(i)} A_i - \beta \mathbb{D}_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}] \right]$$

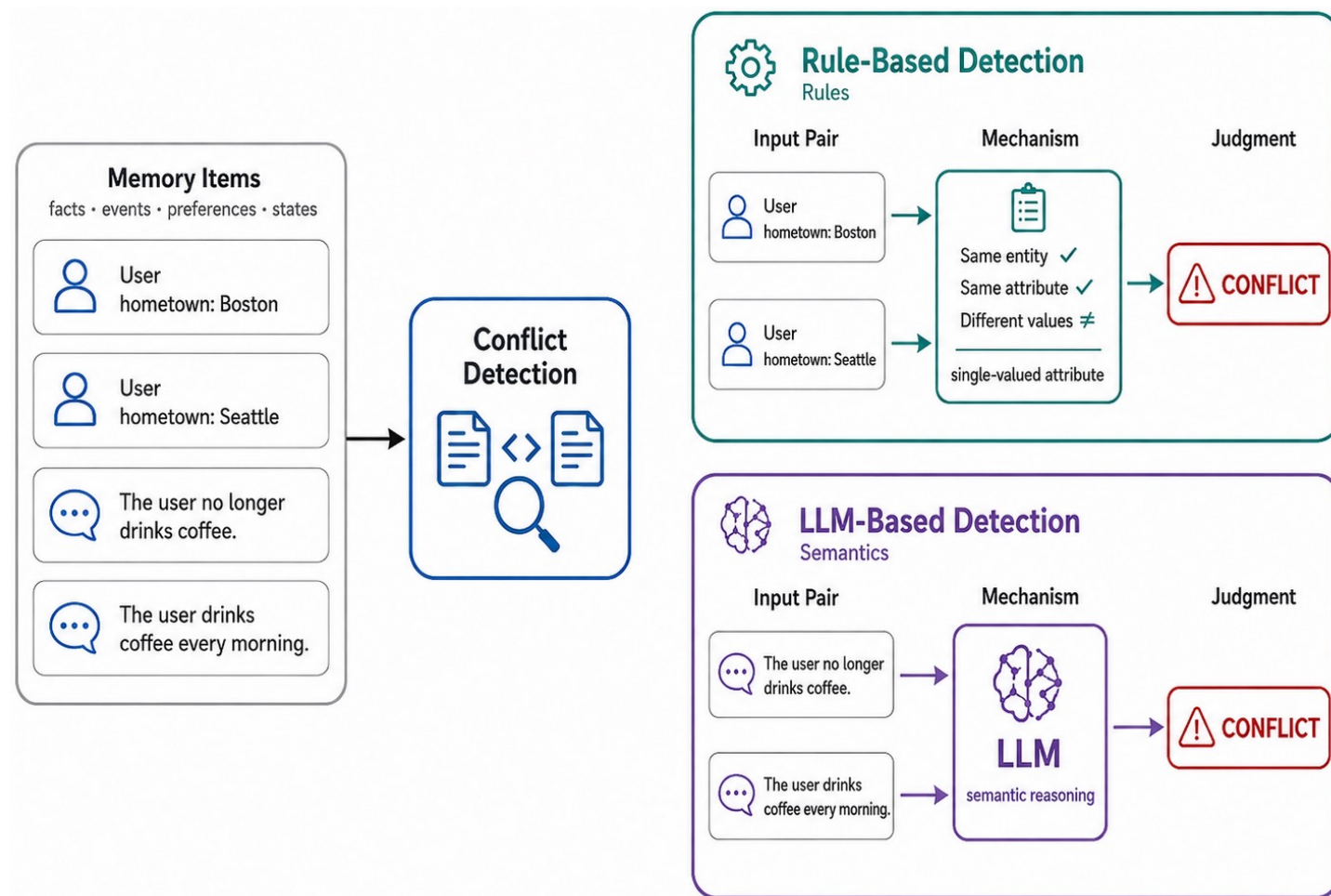
- r_i : reward for each candidate i • $\rho_{\theta}^{(i)}$: per-action importance ratio

- $A_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$: standard group-relative advantage



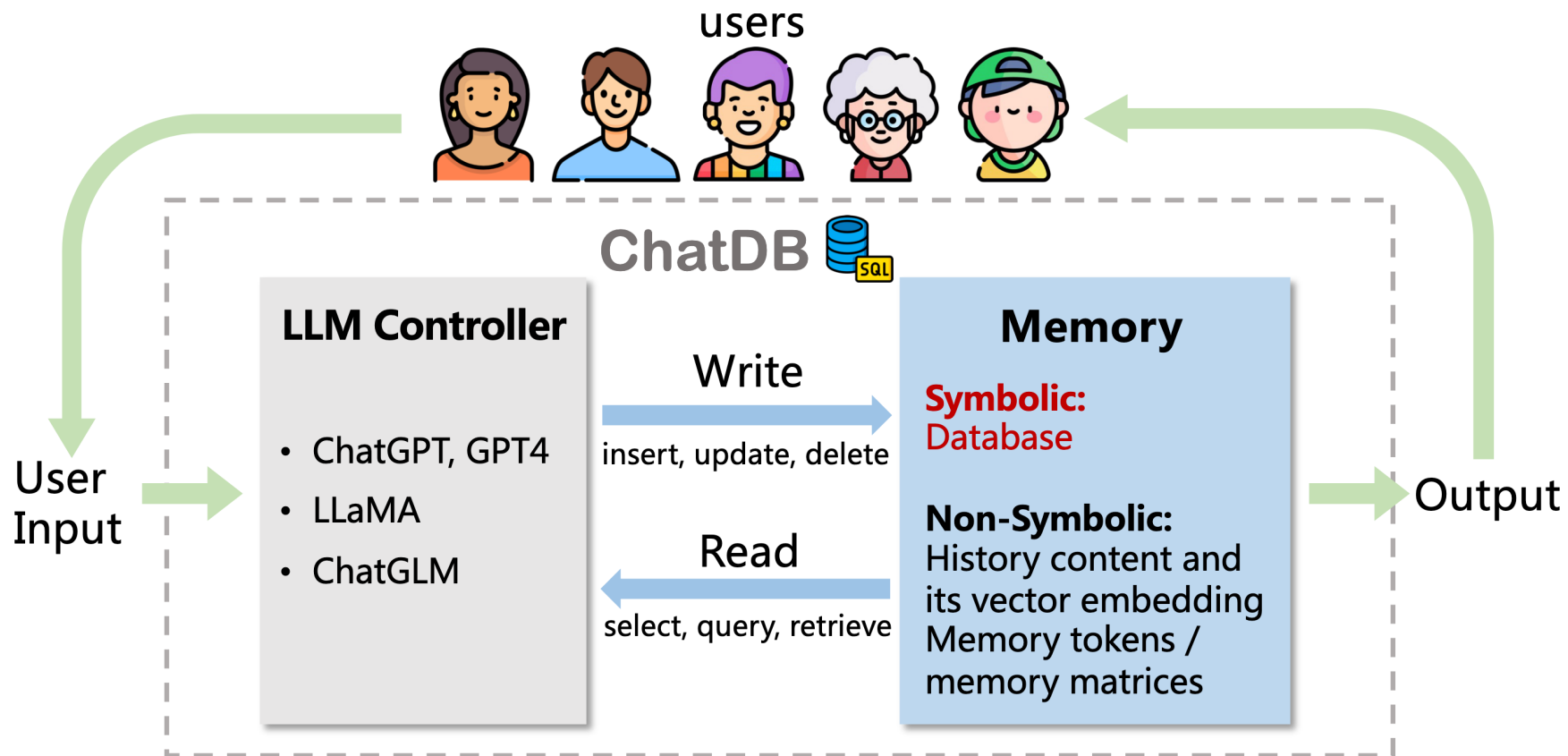
Operator: Conflict Detection

- **Goal:** Automatically identify **contradictions** or **inconsistencies** among memory items.
- **Core Challenge:** Detect genuine memory conflicts **accurately and efficiently**, while **avoiding false conflicts** caused by temporal evolution, contextual conditions, or entity ambiguity.
- **Methods:**
 - Rule-Based
 - LLM-Based



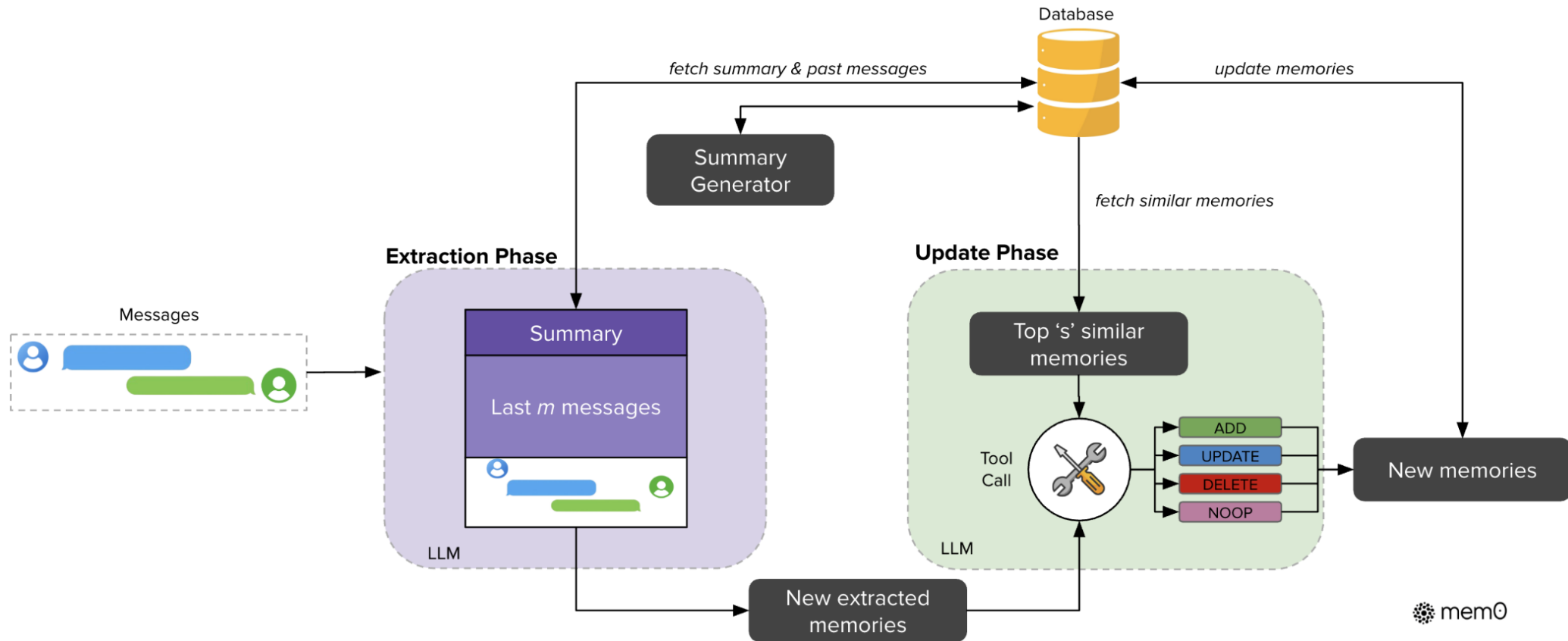
Conflict Detection: Rule-Based

- **Rule-Based Detection:** Identify inconsistent memories by enforcing predefined schemas.
- **Motivation:** Explicit schemas and structural constraints enable deterministic, efficient, and interpretable conflict detection without semantic reasoning.
- **Challenge:** Design sufficiently expressive rules to detect genuine conflicts.



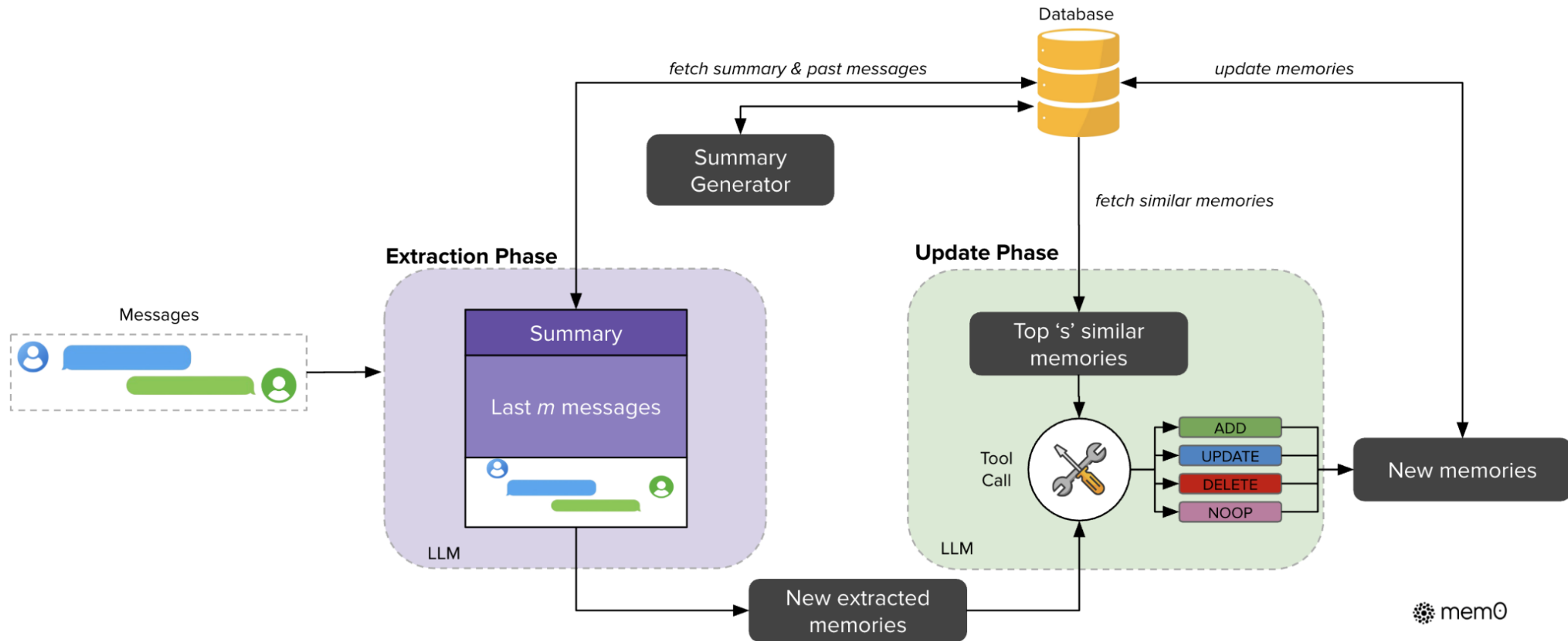
Conflict Detection: LLM-Based

- **LLM-Based Detection:** Use prompted semantic reasoning to identify contradictions among memories.
- **Motivation:** LLM can detect nuanced semantic conflicts beyond the coverage of predefined rules.
- **Challenge:** Detect conflicts reliably while minimizing LLM comparisons.



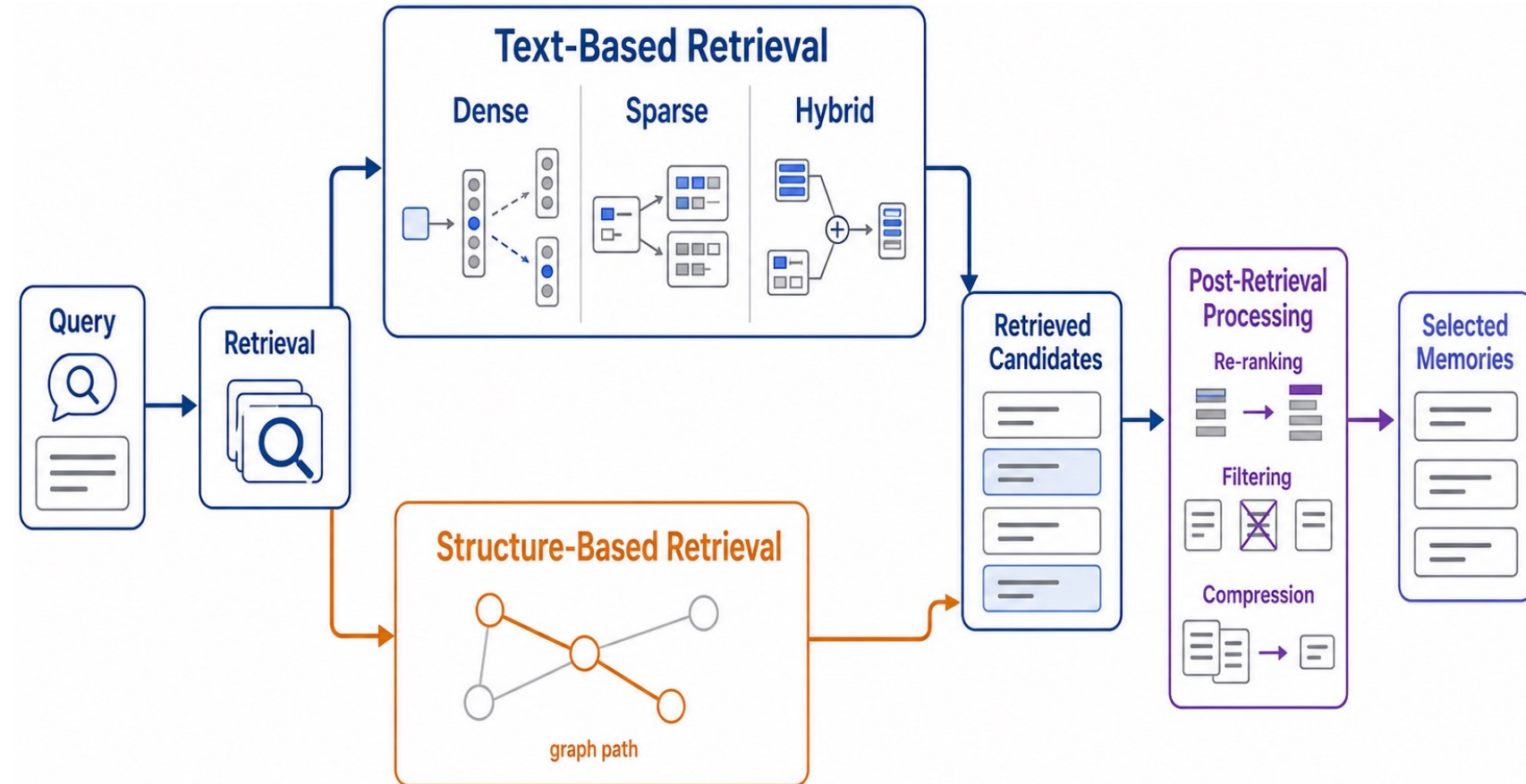
Conflict Detection: LLM-Based

- **LLM-Based Detection:** Use prompted semantic reasoning to identify contradictions among memories.
 - 1. Retrieve top-k nearest historical memories by embedding similarity.
 - 2. Prompt the LLM to classify semantic relations between memory item pairs.



Operator: Retrieval

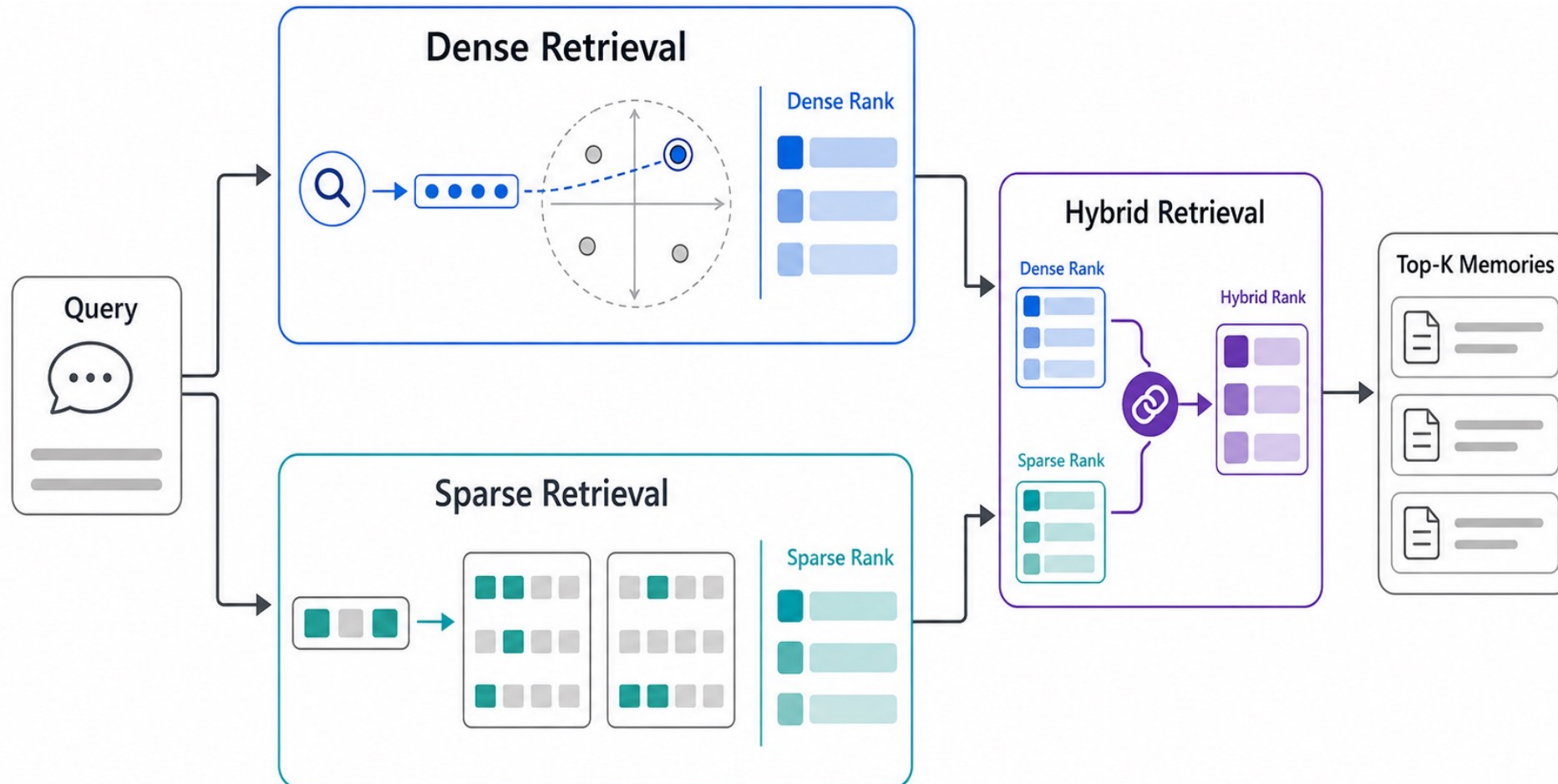
- **Goal:** Select query-relevant memories from the memory store and assemble them into a memory-augmented context for downstream response generation.
- **Core Challenge:** Retrieve truly useful memories while filtering noisy memories from bloating the context.
- **Methods:**
 - Text-Based Retrieval
 - Structure-Based Retrieval
 - Post-Retrieval Processing



Retrieval: Text-Based Retrieval

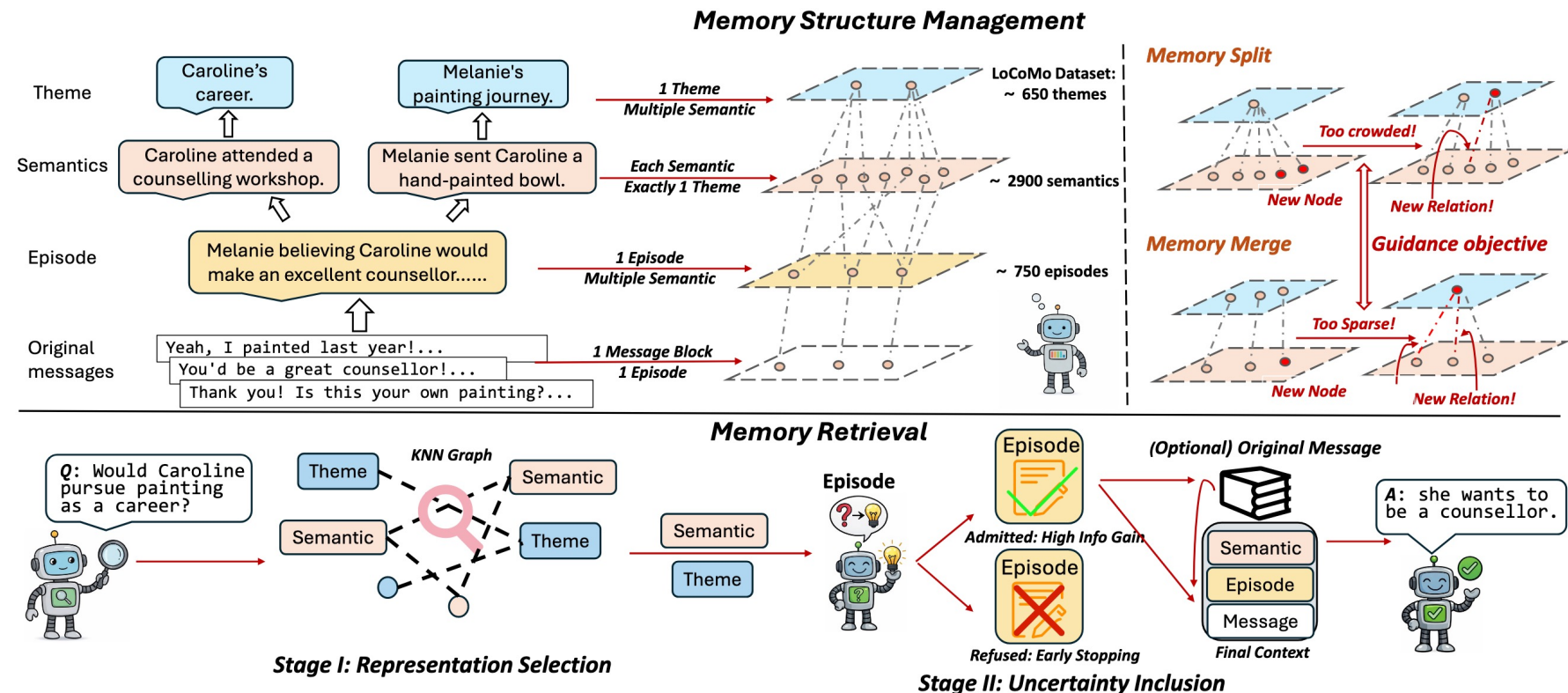
- **Text-Based Retrieval**

- **Dense Retrieval:** Encoding queries and memory items into dense vectors and ranking them by similarity.
- **Sparse Retrieval:** Using lexical matching signals such as keywords, BM25 scores, or inverted indexes.
- **Hybrid Retrieval:** Combines dense semantic retrieval and sparse lexical retrieval.



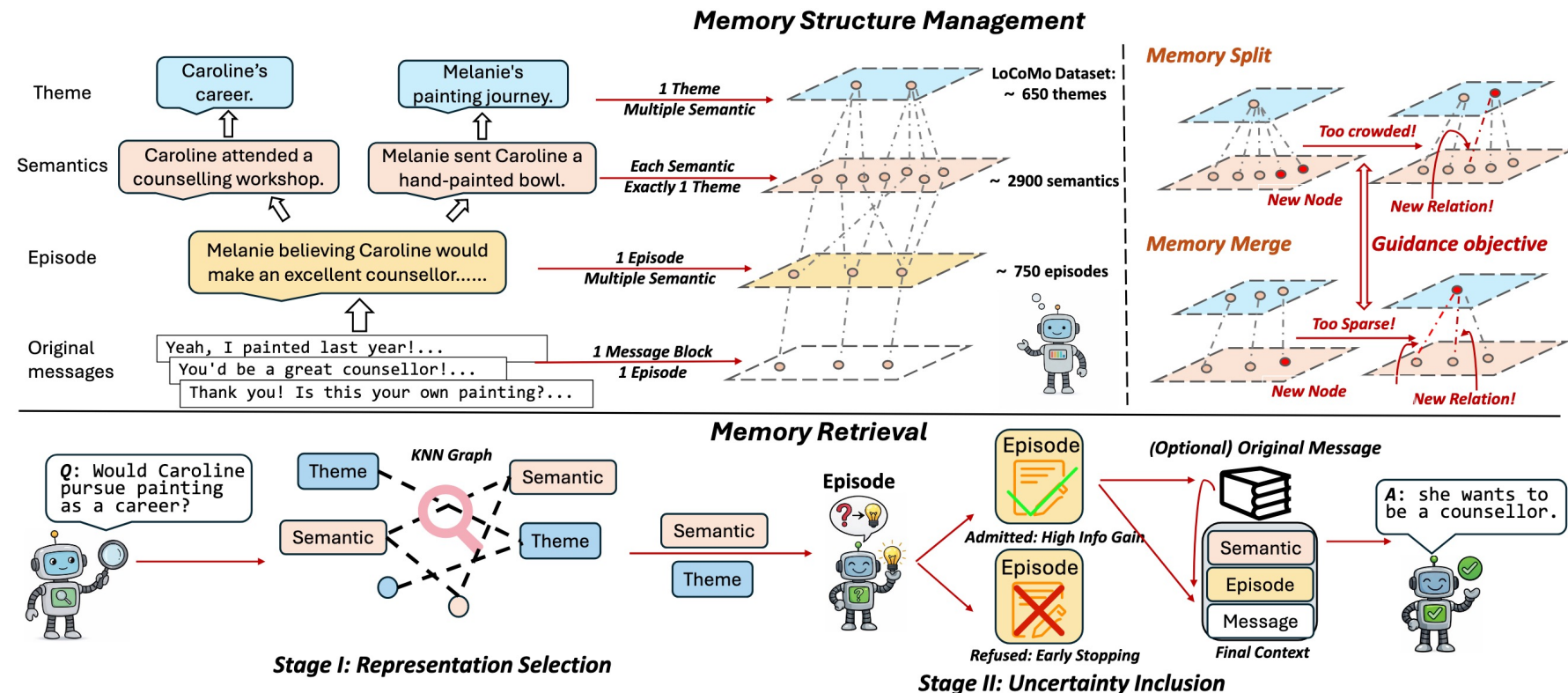
Retrieval: Structure-Based Retrieval

- **Structure-Based Retrieval:** Use memory structures as retrieval indices to select evidence.
- **Motivation:** NL-based retrieval treats memories as independent text units, returning redundant or fragmented evidence while overlooking hierarchical, relational, entity, and temporal dependencies.
- **Challenge:** Construct a faithful memory structure that retrieves relevant evidence at the appropriate path and granularity without introducing excessive overhead.



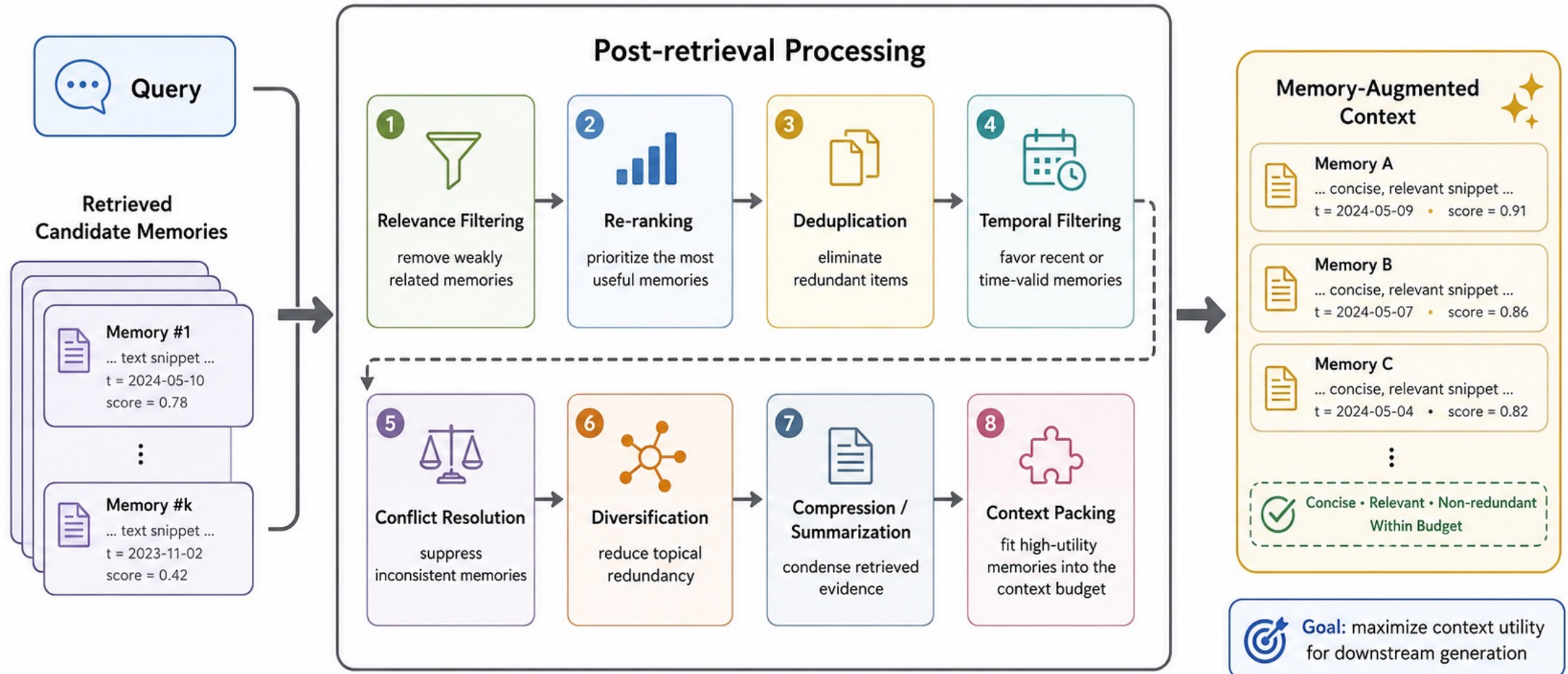
Retrieval: Structure-Based Retrieval

- **Structure-Based Retrieval:** Use memory structures as retrieval indices to select evidence.
 - 1. Retrieve query-relevant memory groups.
 - 2. Jointly optimize relevance and structural coverage to select diverse evidence.
 - 3. Uncertainty-guided evidence expansion.
 - 4. Budget-aware context construction.



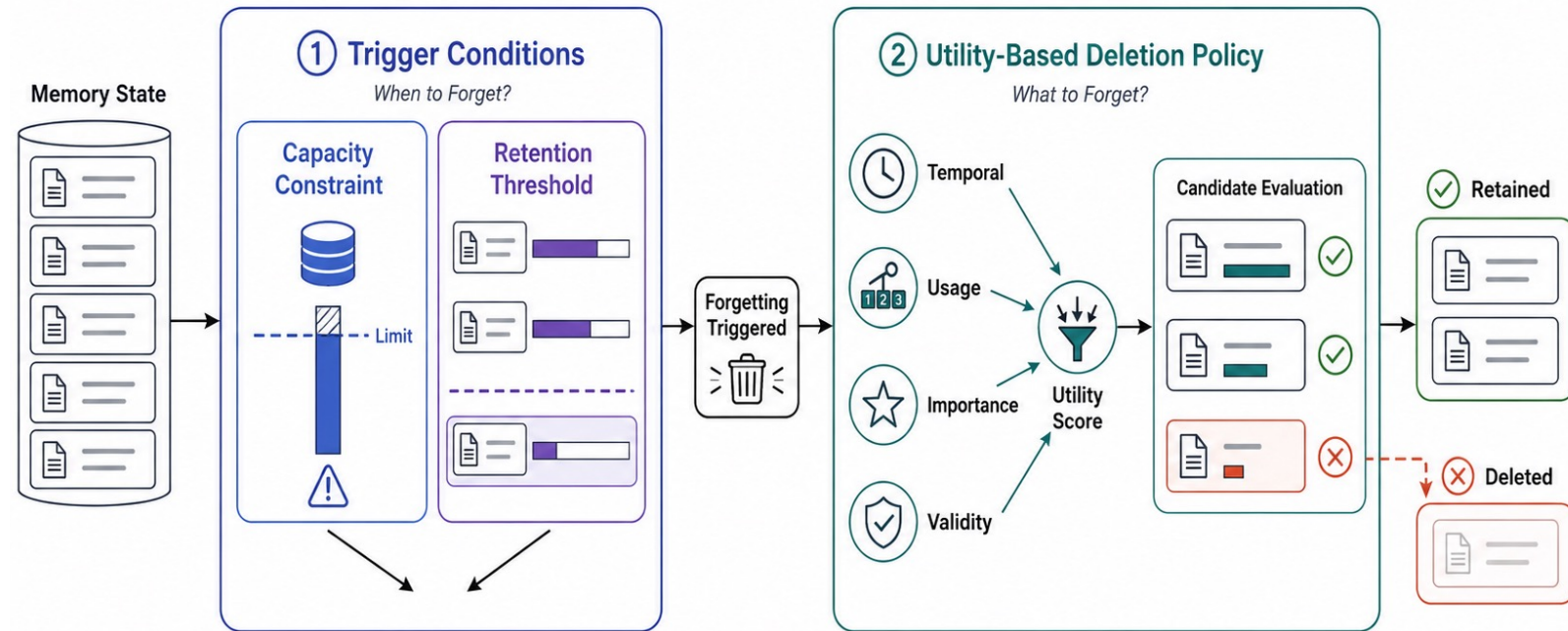
Retrieval: Post-Retrieval Processing

- **Post-Retrieval Processing:** Refine retrieved memories through reranking, compression, and et al.
- **Motivation:** Further refine memories to improve their relevance, information density, and alignment with downstream context requirements.



Operator: Forgetting

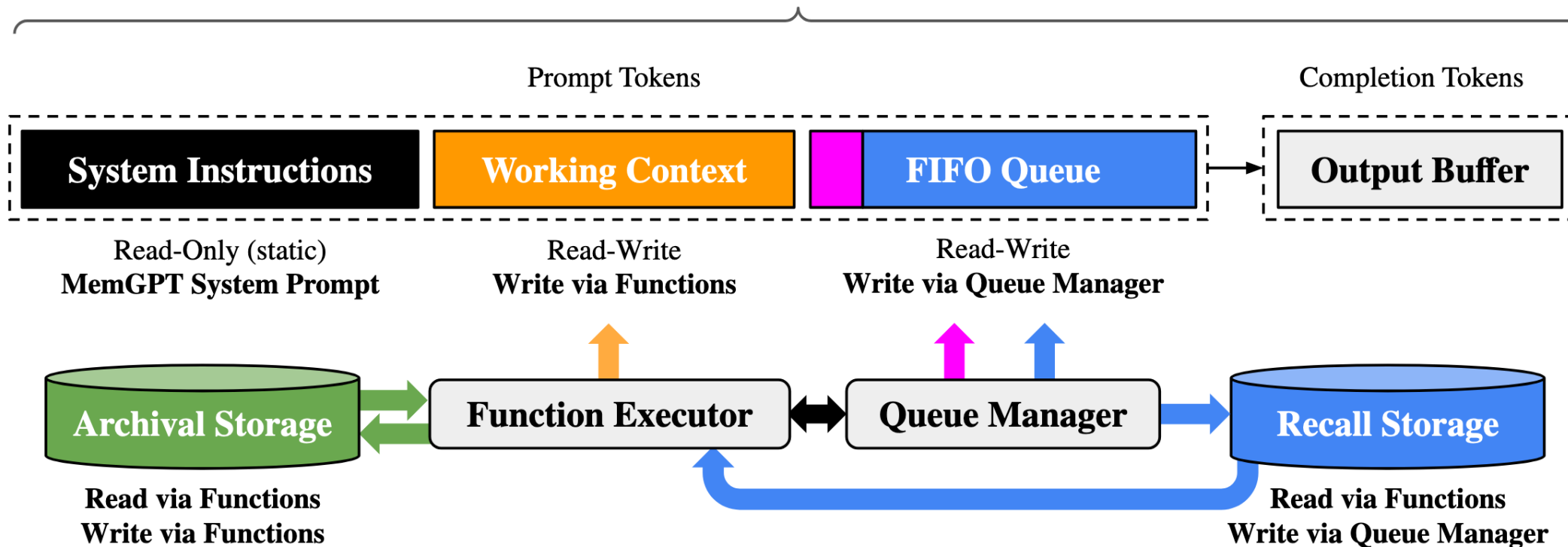
- **Goal:** Remove low-value memories, keeping memory store accurate, compact and up-to-date.
- **Core Challenge:** Decide which memories should be forgotten and when the operator should be invoked.
- **Trigger Conditions:**
 - Capacity constraint
 - Retention threshold
- **Utility-Based Deletion Policy:**
 - Temporal
 - Usage
 - Importance
 - Validity



Forgetting: Capacity Constraint

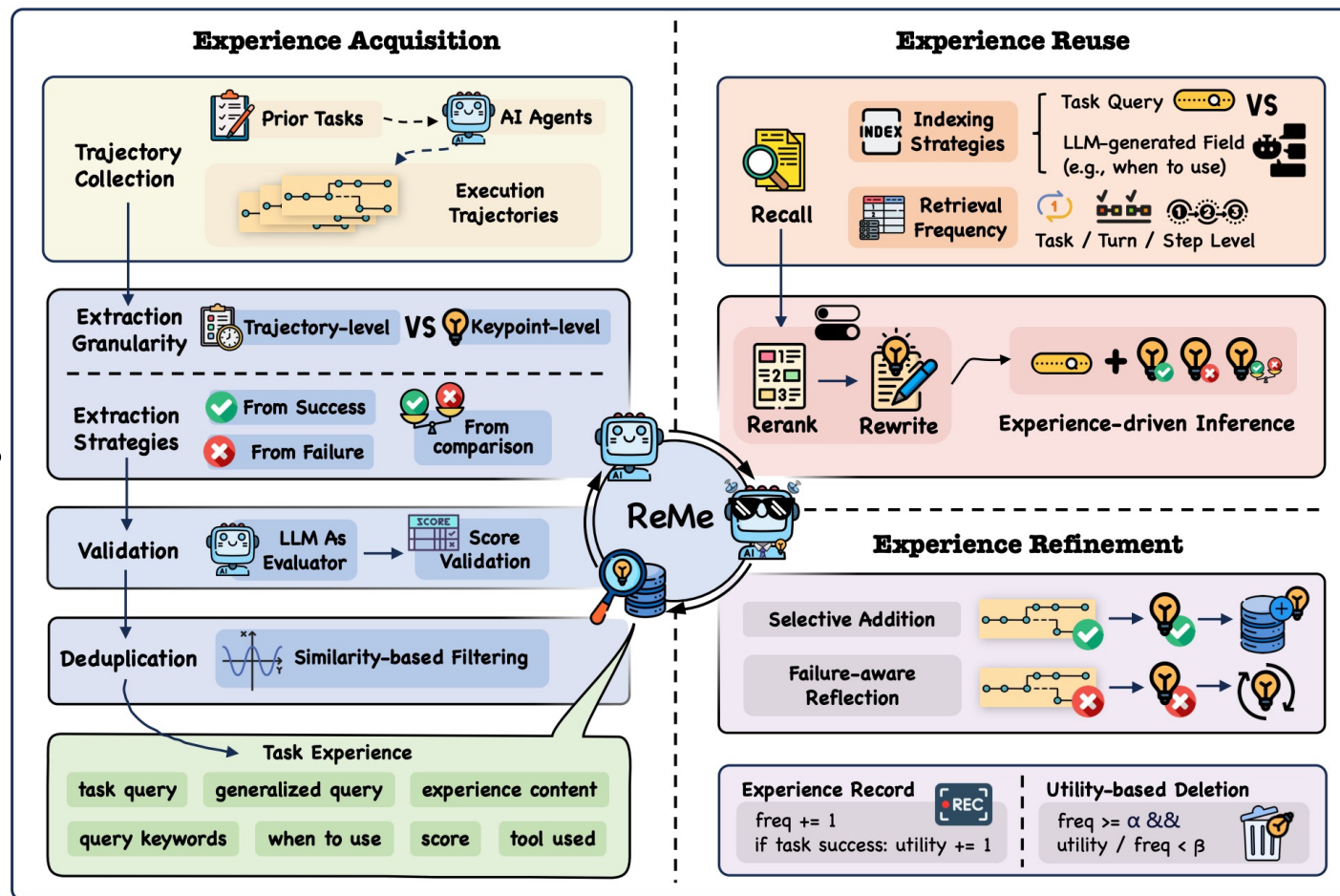
- **Capacity Constraint:** maintain a bounded memory footprint once the active context or memory store exceeds a predefined storage budget.
 - 1. Trigger LLM to move important information into working context or archival memory at the warning threshold.
 - 2. Remove a fixed portion of the earliest messages when the context reaches the flush threshold.
 - 3. Preserve evicted content through recursive summarization.

LLM Finite Context Window (e.g. 8k tokens)



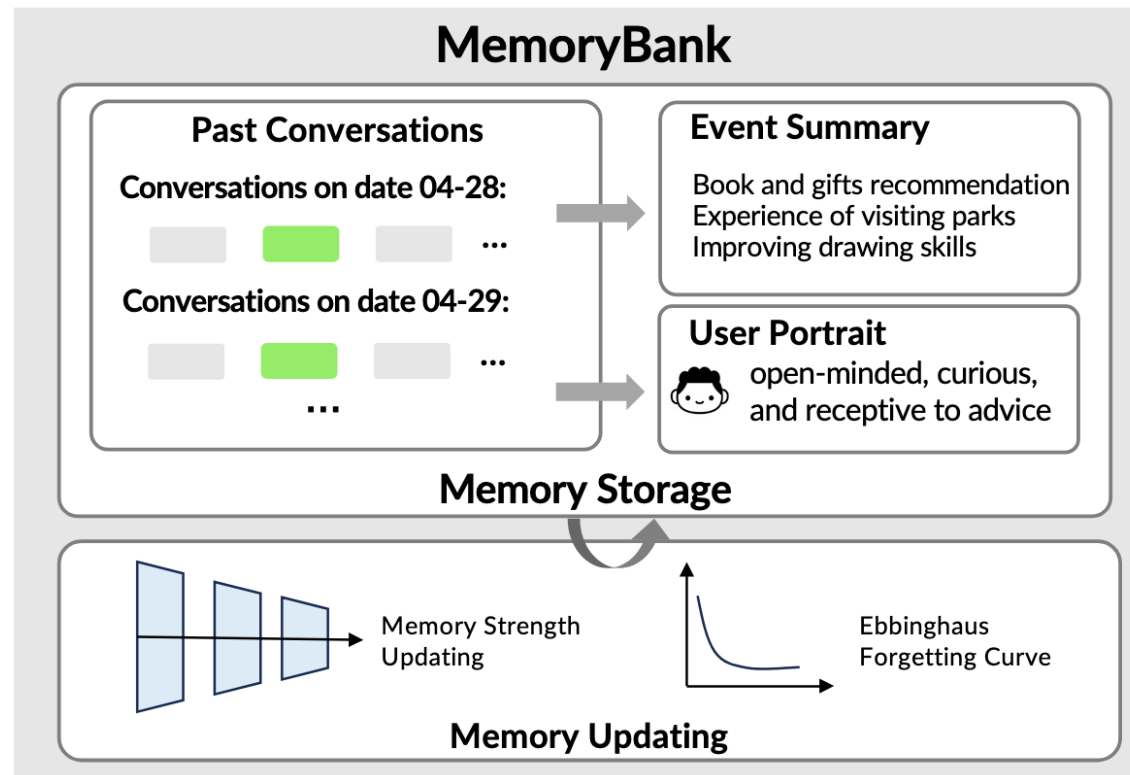
Forgetting: Retention Threshold

- **Retention Threshold:** Activates forgetting when a memory item's estimated retention value falls below a predefined threshold.
- **Motivation:** Low-value memories can accumulate and interfere with future decisions even before storage capacity is exhausted.
- **Challenge:** Accurately estimate evolving memory value from limited and noisy evidence to avoid premature forgetting.



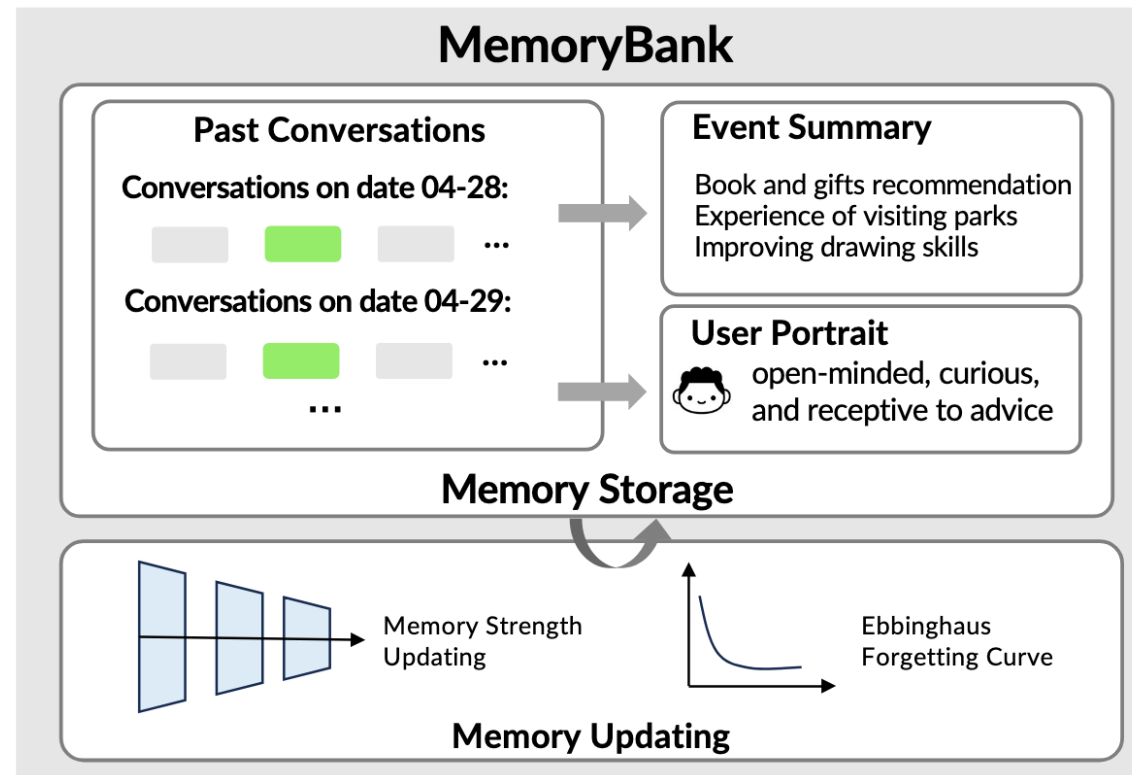
Forgetting: Utility-Based Deletion Policy

- **Utility-Based Deletion Policy:** Estimate the utility of each memory item using signals such as recency, access frequency, importance, and validity, and removes low-value items.
- **Motivation:** Not all memories are equally valuable; forgetting should preferentially remove items with low expected utility.
- **Challenge:** Estimate future memory utility reliably from dynamic and incomplete usage signals.



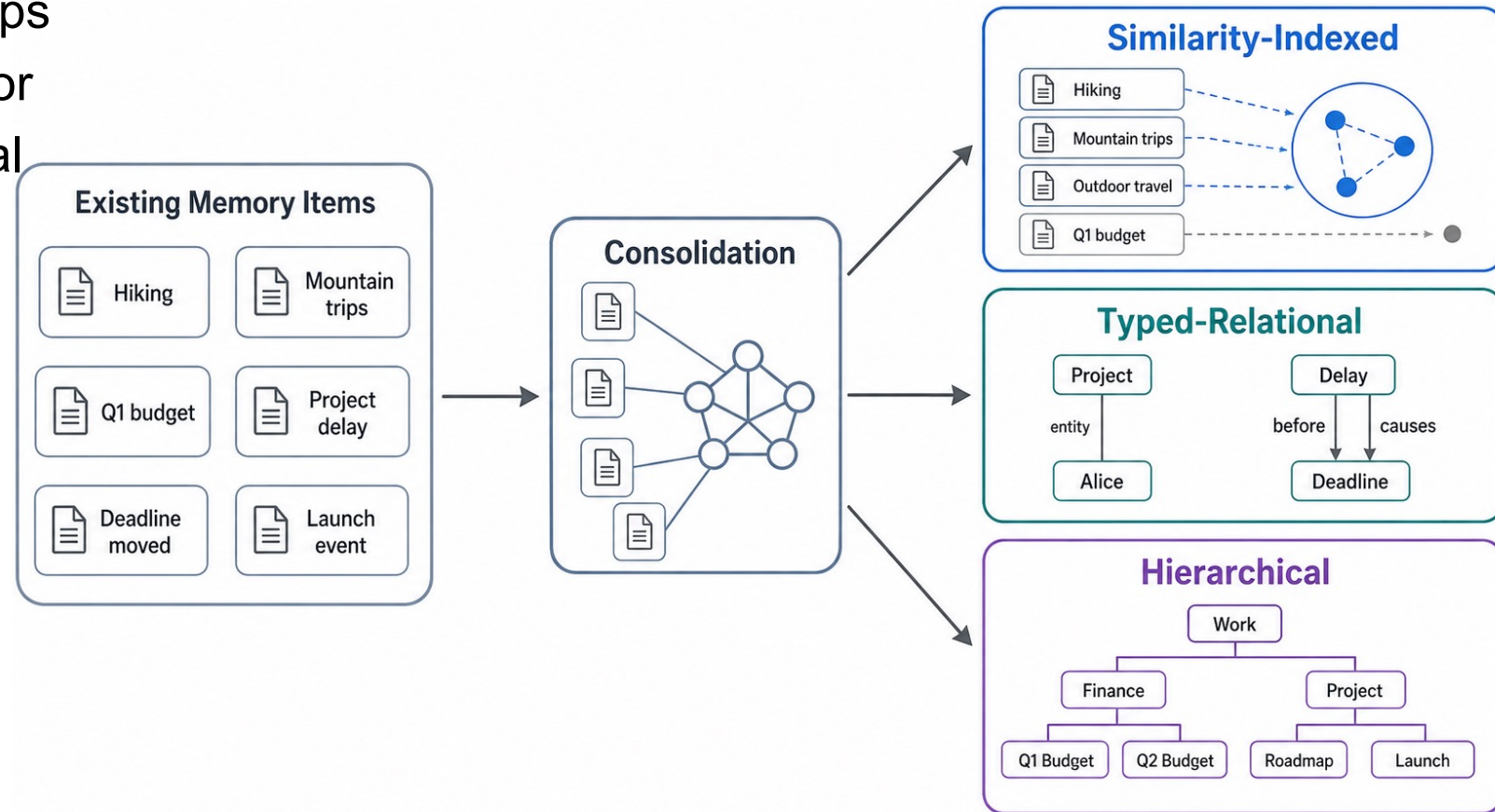
Forgetting: Utility-Based Deletion Policy

- **Utility-Based Deletion Policy:** Estimate the utility of each memory item using signals such as recency, access frequency, importance, and validity, and removes low-value items.
 - 1. Express the Ebbinghaus forgetting curve using an exponential decay model: $R = e^{-\frac{t}{S}}$
 - 2. When a memory item is recalled, increase S by 1 and reset t to 0.
- R : the fraction of the information can be retained.
- t : the time elapsed since learning the information.
- S : the memory strength.



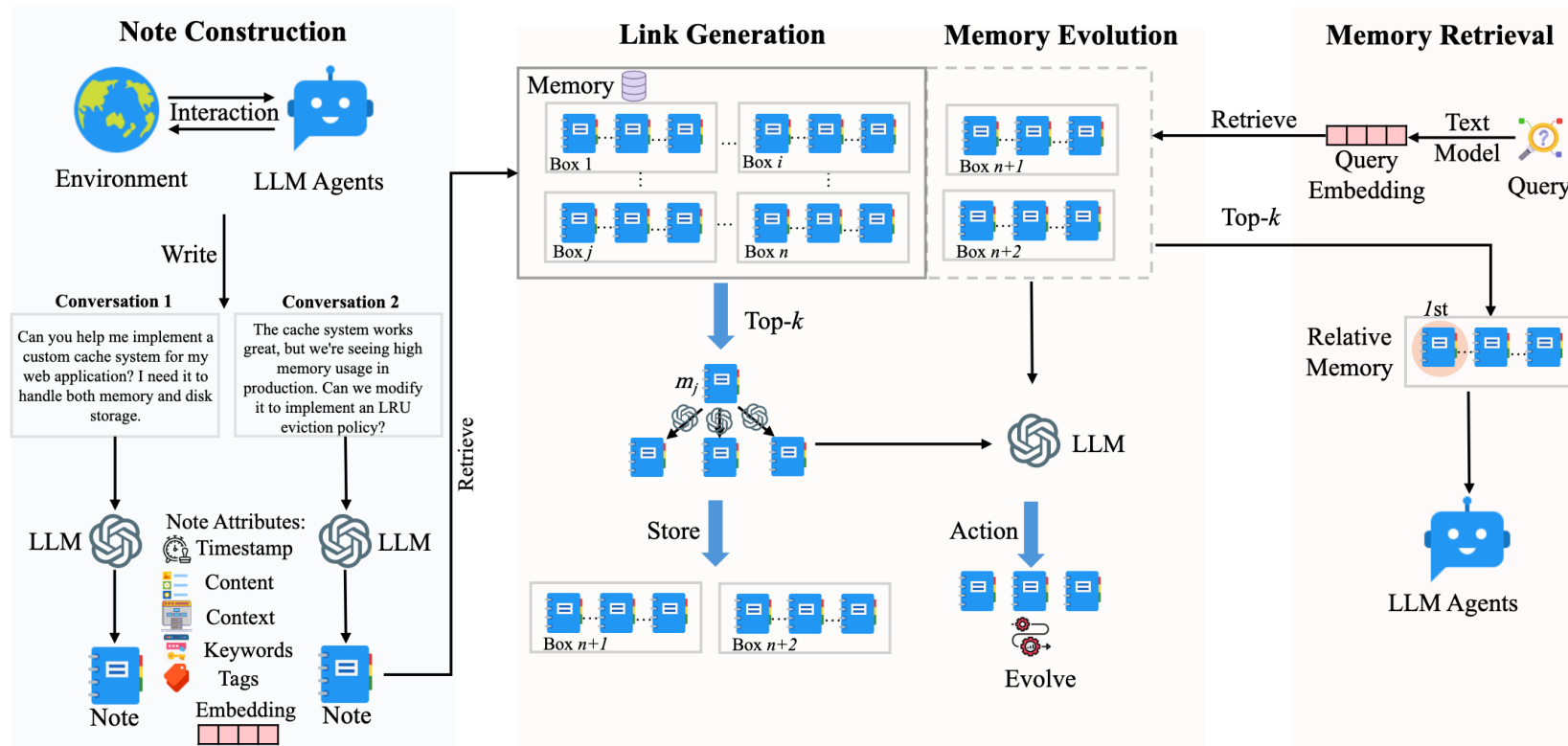
Operator: Consolidation

- **Goal:** Organize existing memory items by establishing meaningful relationships among them and merging redundant or highly overlapping items into canonical representations.
- **Core Challenge:** Identify meaningful relationships and redundancies accurately and efficiently.
- **Methods:**
 - Similarity-Indexed Consolidation
 - Typed Relational Consolidation
 - Hierarchical Consolidation



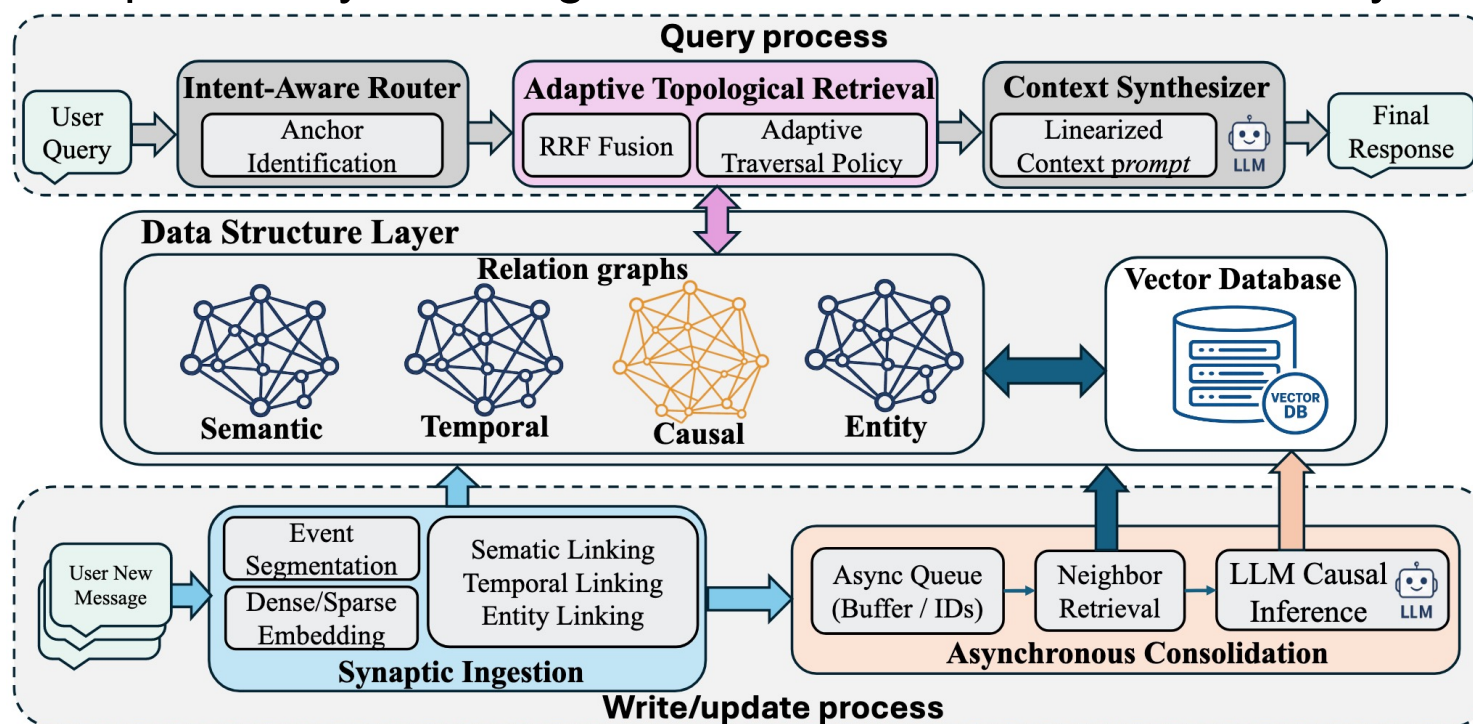
Consolidation: Similarity-Indexed

- **Similarity-Indexed Consolidation:** Maintain similarity-based neighborhoods or groups without explicit relation semantics.
 - 1. Retrieve top-k nearest historical memories by embedding similarity.
 - 2. Generate links via LLM-based relation analysis.
 - 3. Evolve neighboring memories.



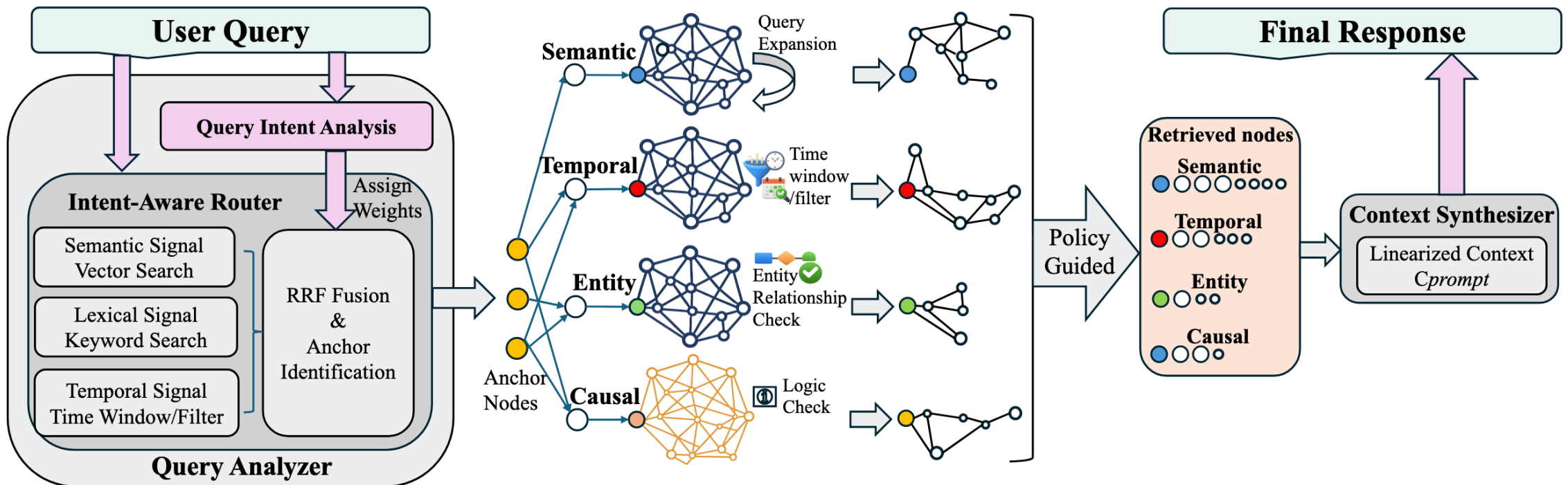
Consolidation: Typed Relational

- **Typed Relational Consolidation:** Maintain explicitly defined entity, temporal, causal, semantic, or provenance relations.
- **Motivation:**
 - Similarity-based links indicate relatedness but cannot express how memories are connected.
 - Typed entity, temporal, and causal relations enable precise, interpretable, and relation-aware retrieval.
- **Challenge:** Define an expressive yet manageable relation schema and reliably infer the correct relation.



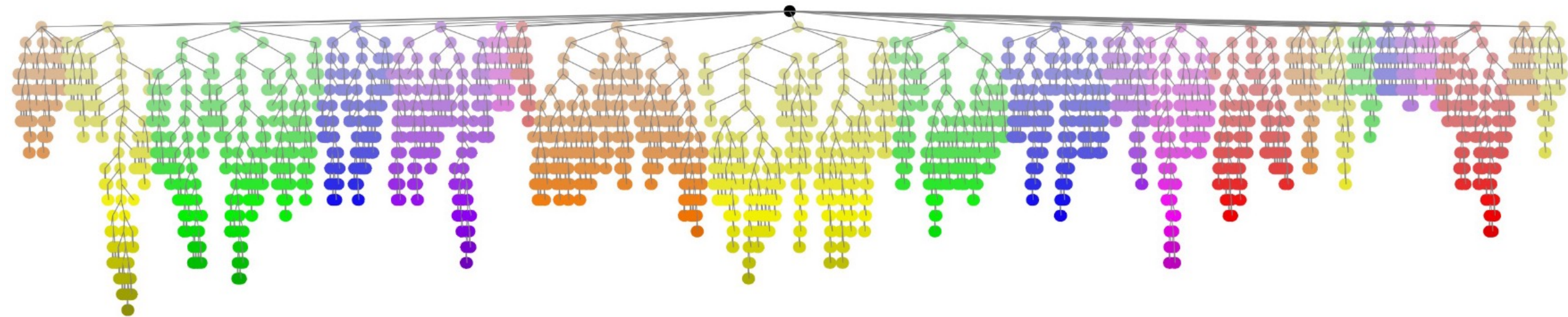
Consolidation: Typed Relational

- **Typed Relational Consolidation:** Maintain explicitly defined entity, temporal, causal, semantic, or provenance relations.
 - 1. Attach each new event to the temporal backbone.
 - 2. Create semantic links between embedding-similar events above a similarity threshold.
 - 3. Connect events to shared entity nodes.
 - 4. Asynchronously use an LLM over the local 2-hop neighborhood to infer and add latent causal and entity links.



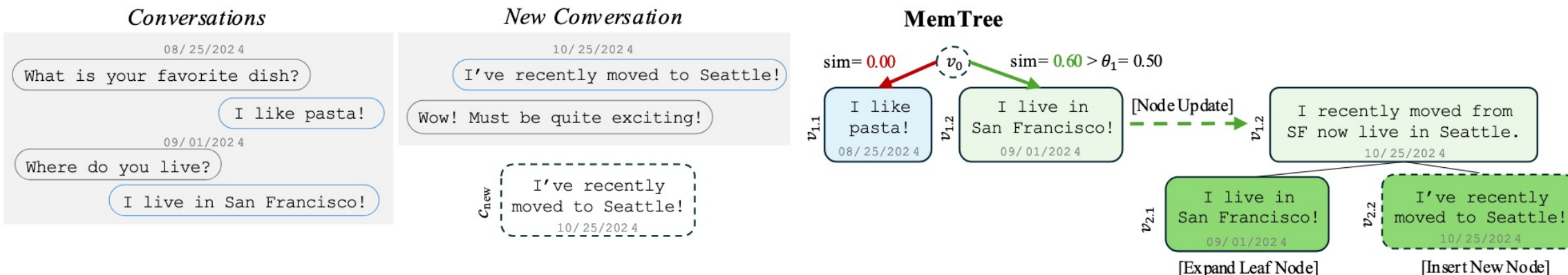
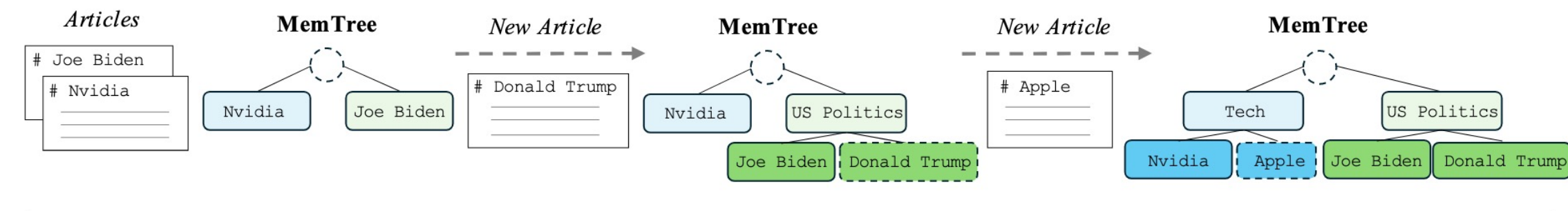
Consolidation: Hierarchical Consolidation

- **Hierarchical Consolidation:** Organize memories through parent-child or membership structures.
- **Motivation:** Flat collections of fine-grained memories hinder efficient navigation and provide limited support for retrieving information at different levels of abstraction.
- **Challenge:** Construct and maintain a coherent hierarchy that places memories under appropriate abstractions and supporting reliable coarse-to-fine retrieval.



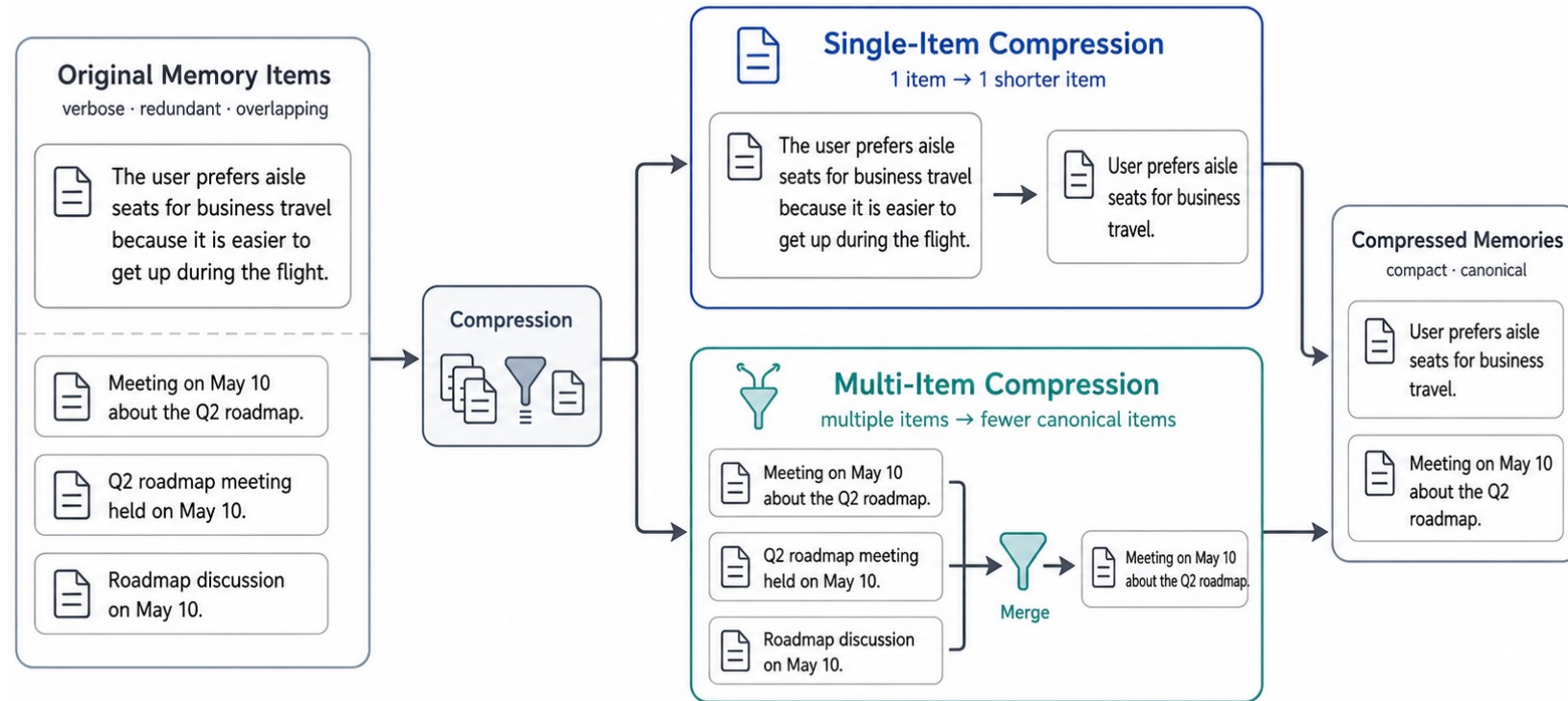
Consolidation: Hierarchical Consolidation

- **Hierarchical Consolidation:** Organize memories through parent-child or membership structures.
 - 1. Route the most similar child if the similarity above the threshold. $\theta(d) = \theta_0 e^{\lambda d}$
 - 2. Create a new leaf if no child is similar enough.
 - 3. Expand leaf nodes when similar new information arrives.
 - 4. Refresh ancestor summaries along the insertion path.



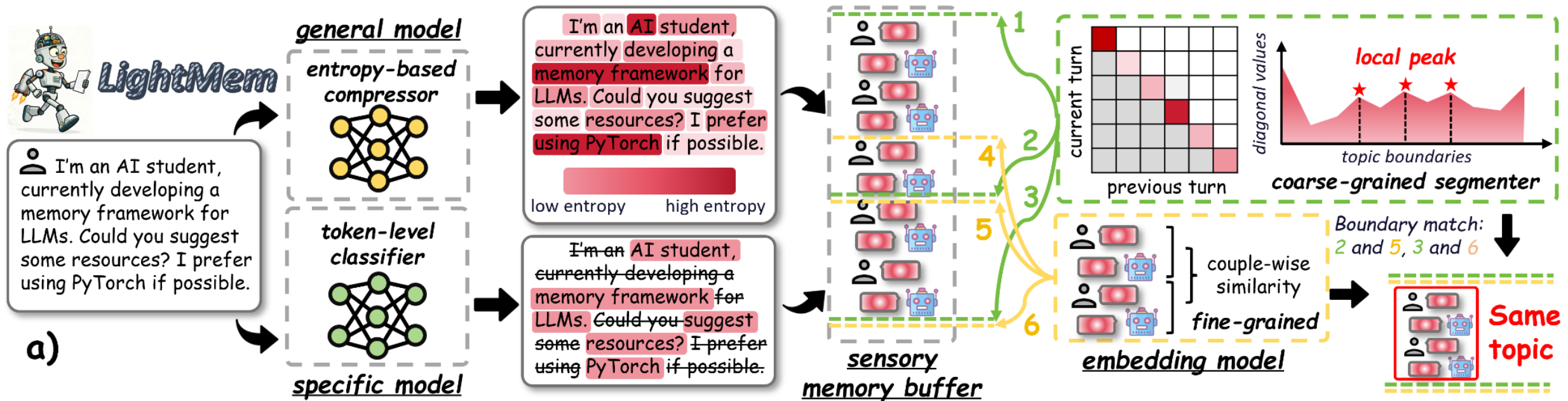
Operator: Compression

- **Goal:** Reduce memory storage overhead by condensing individual memory items and fusing redundant or semantically overlapping items while preserving information.
- **Core Challenge:** Determine when and how to reduce memory representation costs without losing information essential for future retrieval and reasoning utility.
- **Methods:**
 - Single-Item Compression
 - Multi-Item Compression



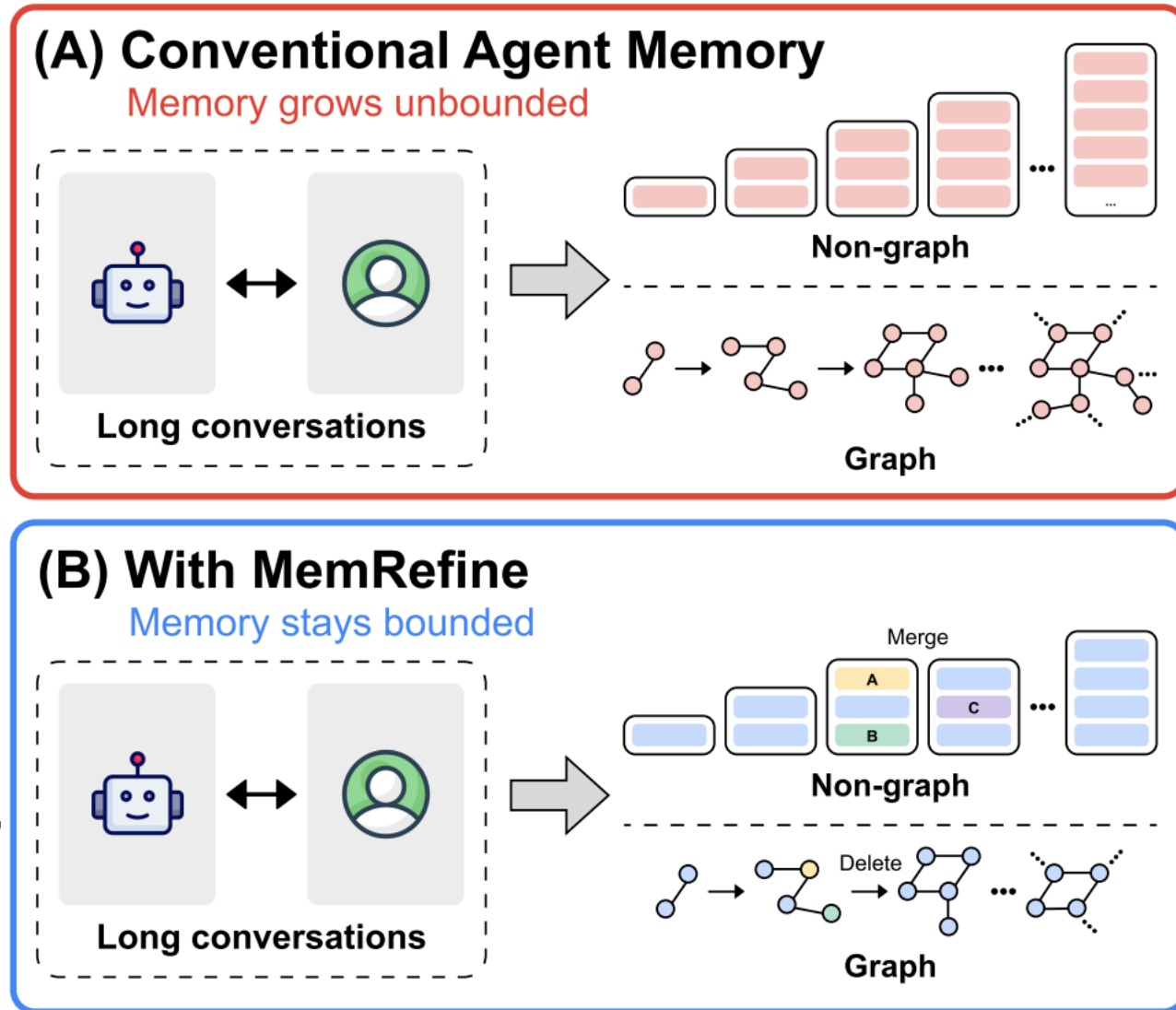
Compression: Single-Item Compression

- **Single-Item Compression:** Compress an individual memory item by summarizing, pruning, or rewriting its content while preserving its essential semantics.
 - 1. Score each token with a lightweight compression model or entropy-based compressor.
 - 2. Select high-retention tokens according to a predefined compression ratio.
- **Motivation:** Individual memory items may contain verbose, redundant, or task-irrelevant details, reducing information density and increasing storage and retrieval costs.



Compression: Multi-Item Compression

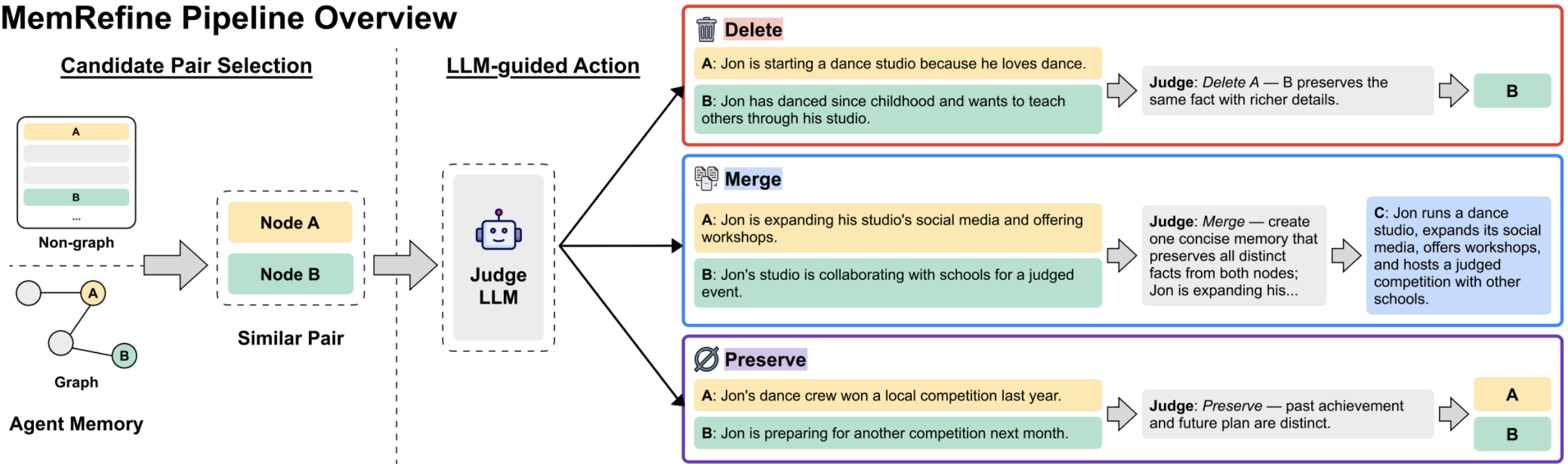
- **Multi-Item Compression:** Compress a set of redundant or highly overlapping memory items into fewer canonical representations.
- **Motivation:** Redundant or overlapping memories repeatedly encode the same information, inflating storage and retrieval costs while fragmenting related evidence across multiple items.
- **Challenge:** Identify which memories can be safely merged and construct a faithful canonical representation without collapsing distinct contexts, temporal states, or conflicting details.



Compression: Multi-Item Compression

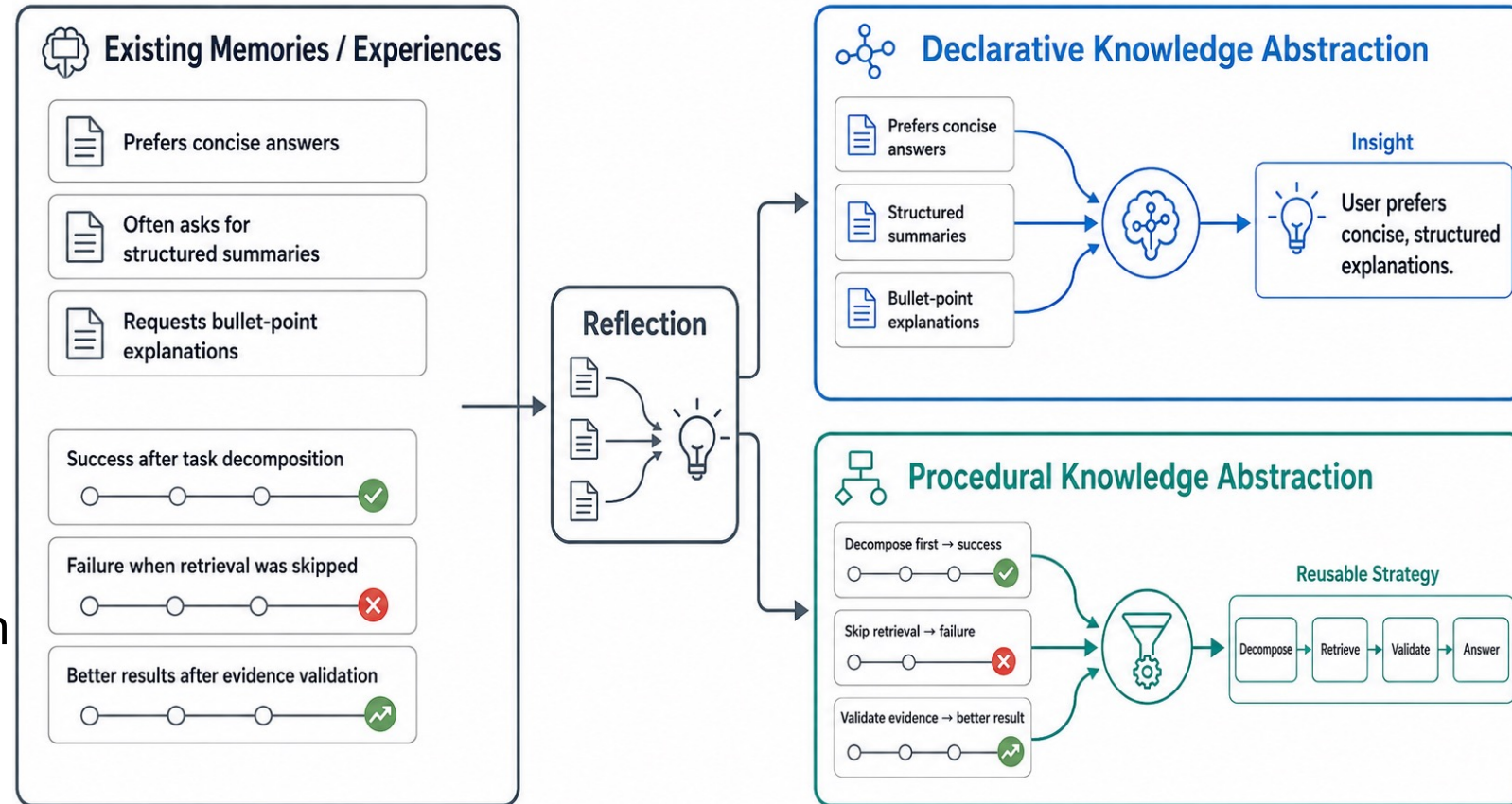
- **Multi-Item Compression:** Compress a set of redundant or highly overlapping memory items into fewer canonical representations.
 - 1. Embed all memory items and iteratively select the most similar unresolved pair.
 - 2. Classify each pair as redundant, complementary, or distinct.
 - 3. Generate a canonical merged memory.

MemRefine Pipeline Overview



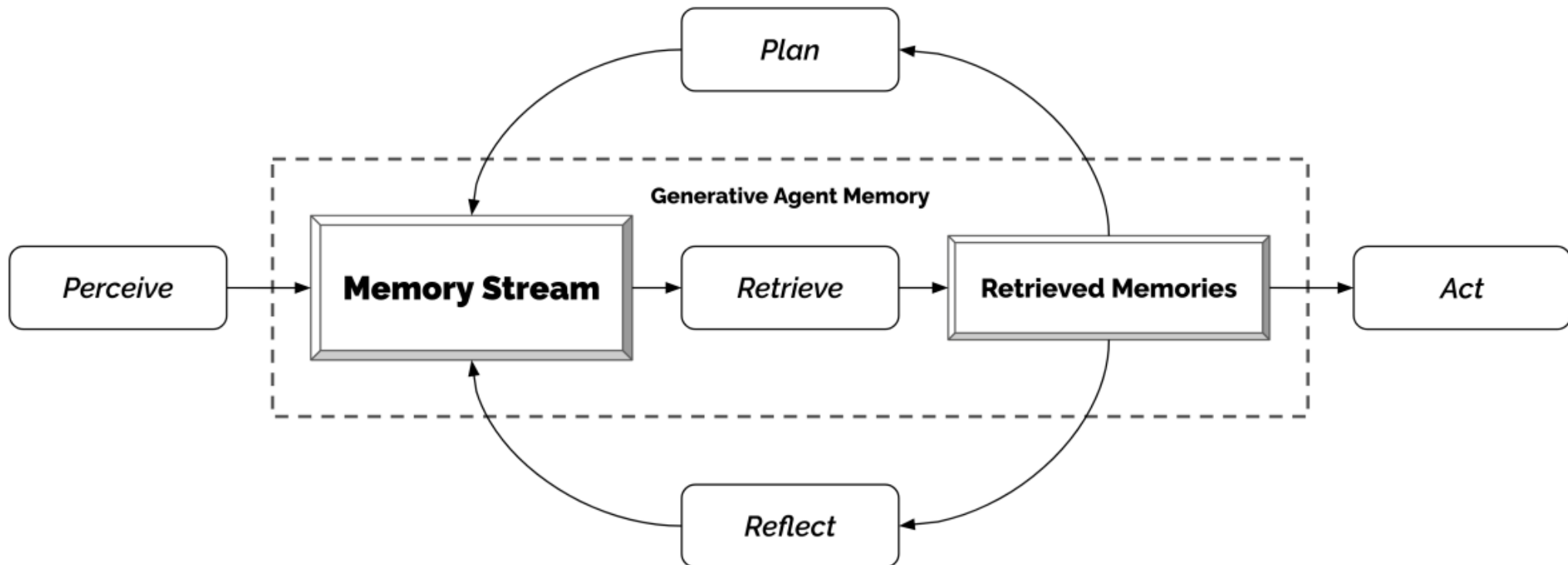
Operator: Reflection

- **Goal:** Derive new higher-level knowledge from existing memories.
- **Core Challenge:** Determine what evidence warrants reflection, while ensuring that the resulting knowledge is faithful, useful, and generalizable.
- **Methods:**
 - Declarative Knowledge Abstraction
 - Procedural Knowledge Abstraction



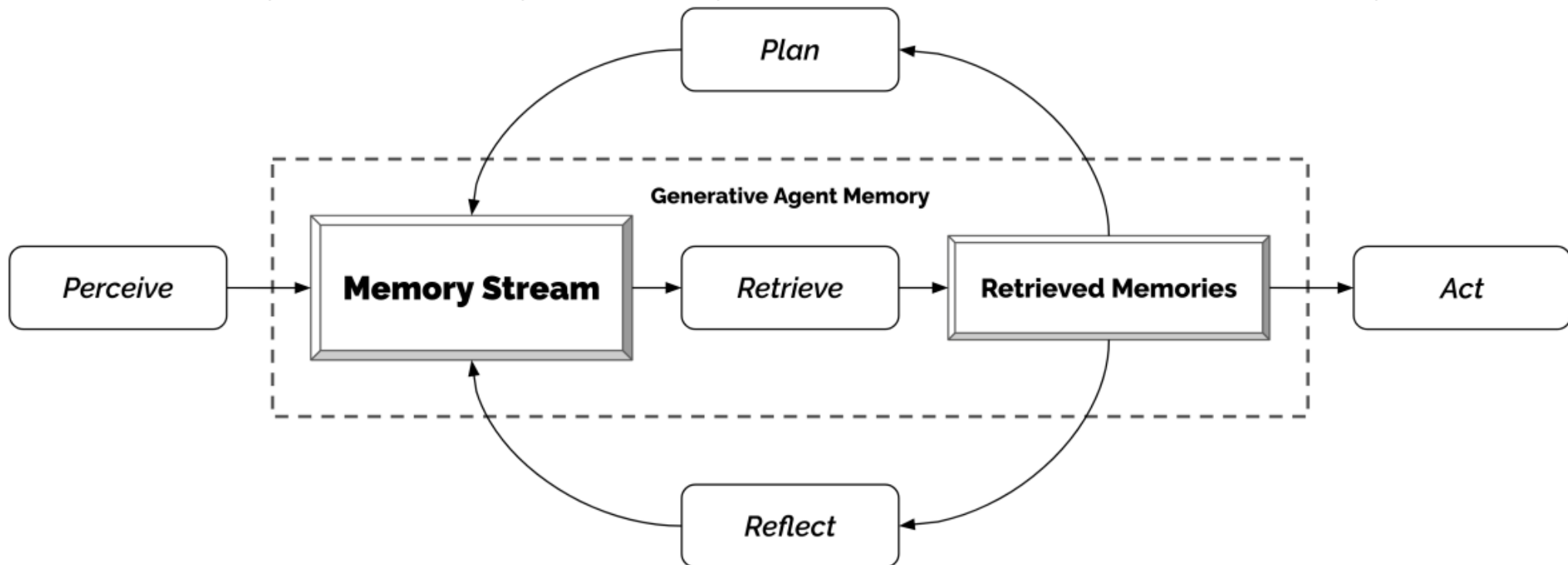
Reflection: Declarative Knowledge Abstraction

- **Declarative Knowledge Abstraction:** Synthesize multiple memories into higher-level beliefs, patterns, relationships, or persistent states.
- **Motivation:** Individual memories provide fragmented evidence, but persistent beliefs, patterns, and relationships emerge only through cross-memory synthesis..
- **Challenge:** Aggregate consistent evidence into faithful and generalizable knowledge.



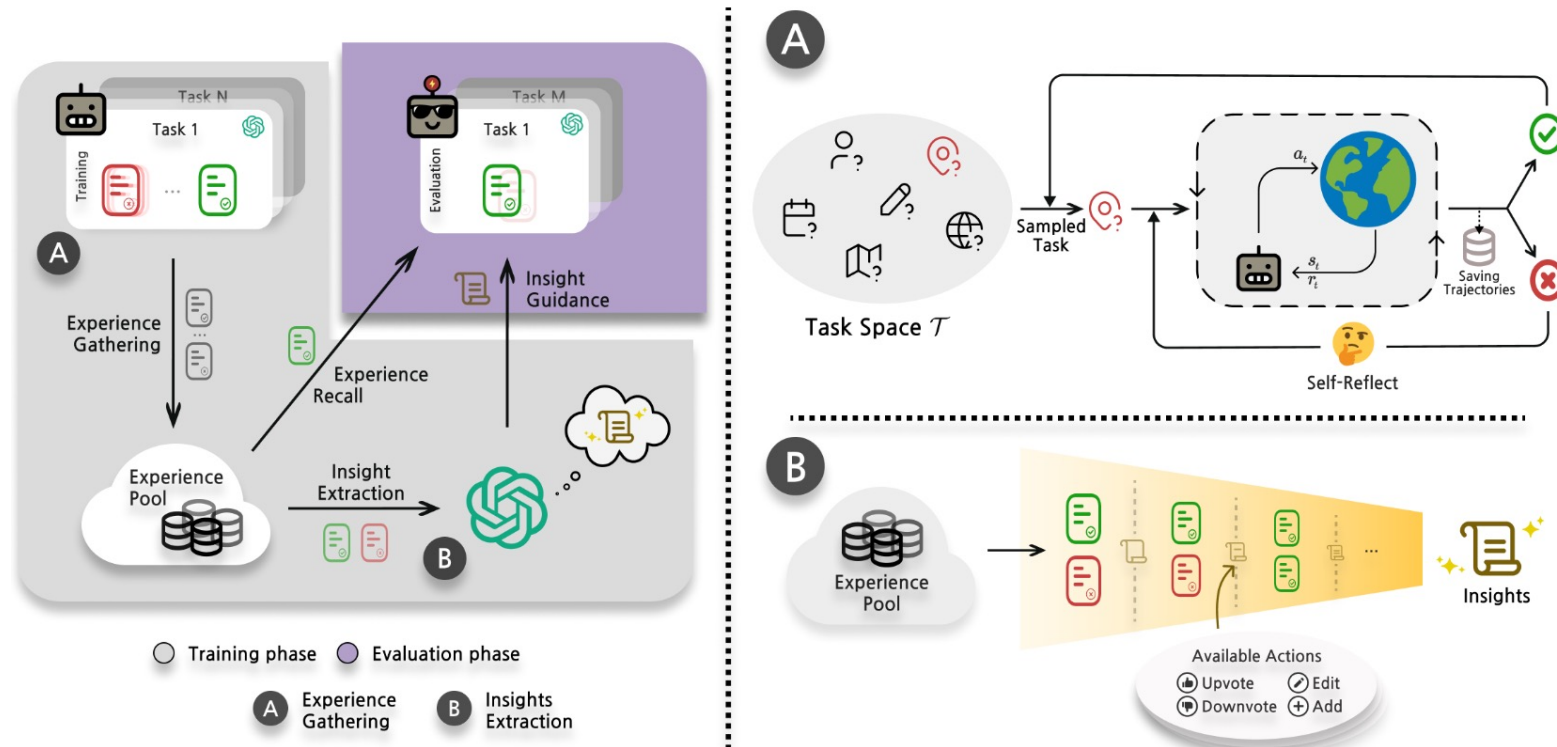
Reflection: Declarative Knowledge Abstraction

- **Declarative Knowledge Abstraction:** Synthesize multiple memories into higher-level beliefs, patterns, relationships, or persistent states.
 - 1. Start reflection when the cumulative importance of recent memories exceeds a threshold.
 - 2. Prompt the LLM with the 100 most recent memory records to identify three questions.
 - 3. Use each question to collect relevant observations and existing reflections.
 - 4. Prompt the LLM to generate five high-level insights with explicit references to supporting memories.



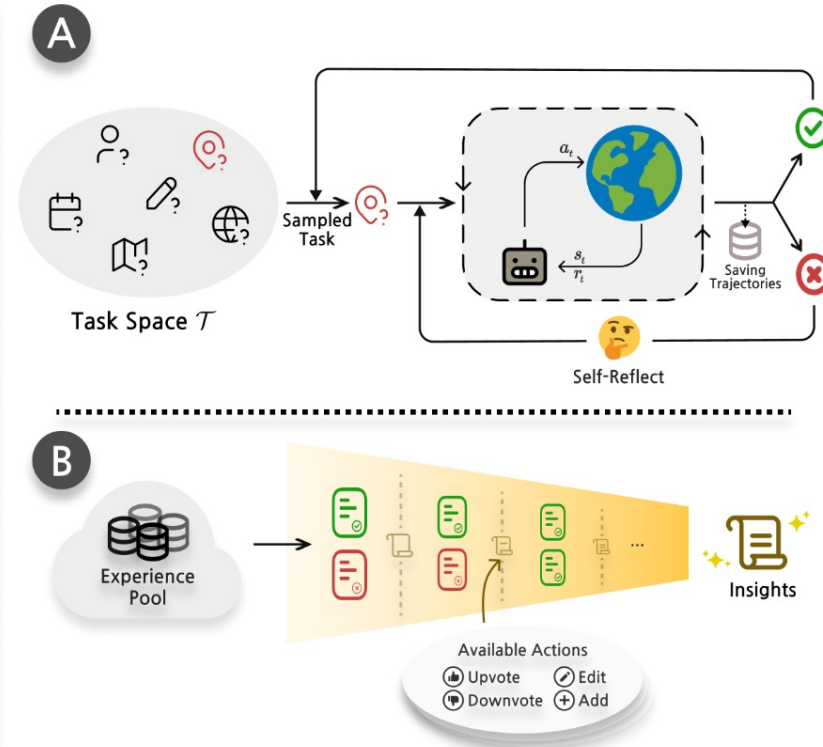
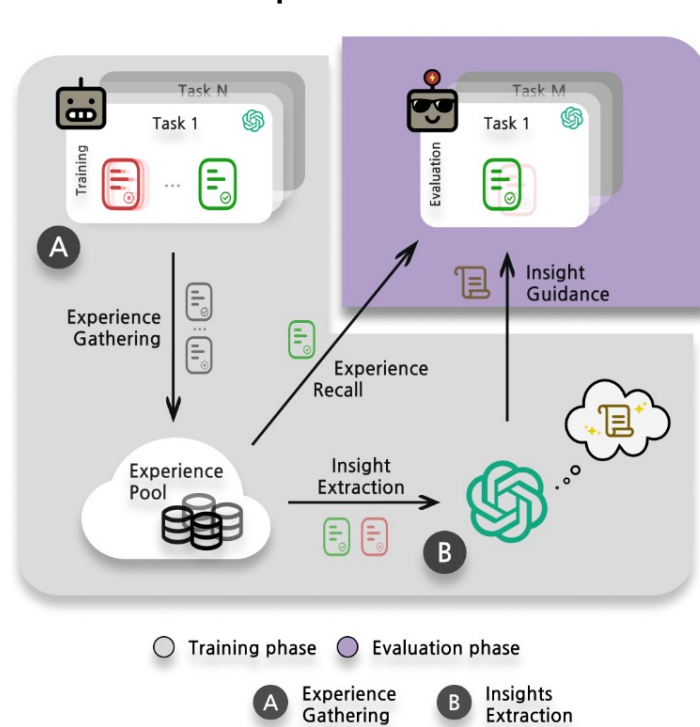
Reflection: Procedural Knowledge Abstraction

- **Procedural Knowledge Abstraction:** Distill multiple task experiences into reusable strategies, workflows, rules, or skills.
- **Motivation:** Individual task experiences remain context-specific, while cross-experience abstraction can transform repeated successes and failures into reusable strategies, workflows, and skills.
- **Challenge:** Identify the truly effective action patterns and generalize them across tasks.



Reflection: Procedural Knowledge Abstraction

- **Procedural Knowledge Abstraction:** Distill multiple task experiences into reusable strategies, workflows, rules, or skills.
 - 1. Pair failed and successful trajectories of the same task, and group successes from different tasks.
 - 2. Identify failure causes, corrective strategies, and recurring best practices using LLM.
 - 3. Use ADD, EDIT, UPVOTE, and DOWNVOTE to iteratively refine the insight set.
 - 4. Remove insights whose importance count fails to zero.



Memory Operators Overview

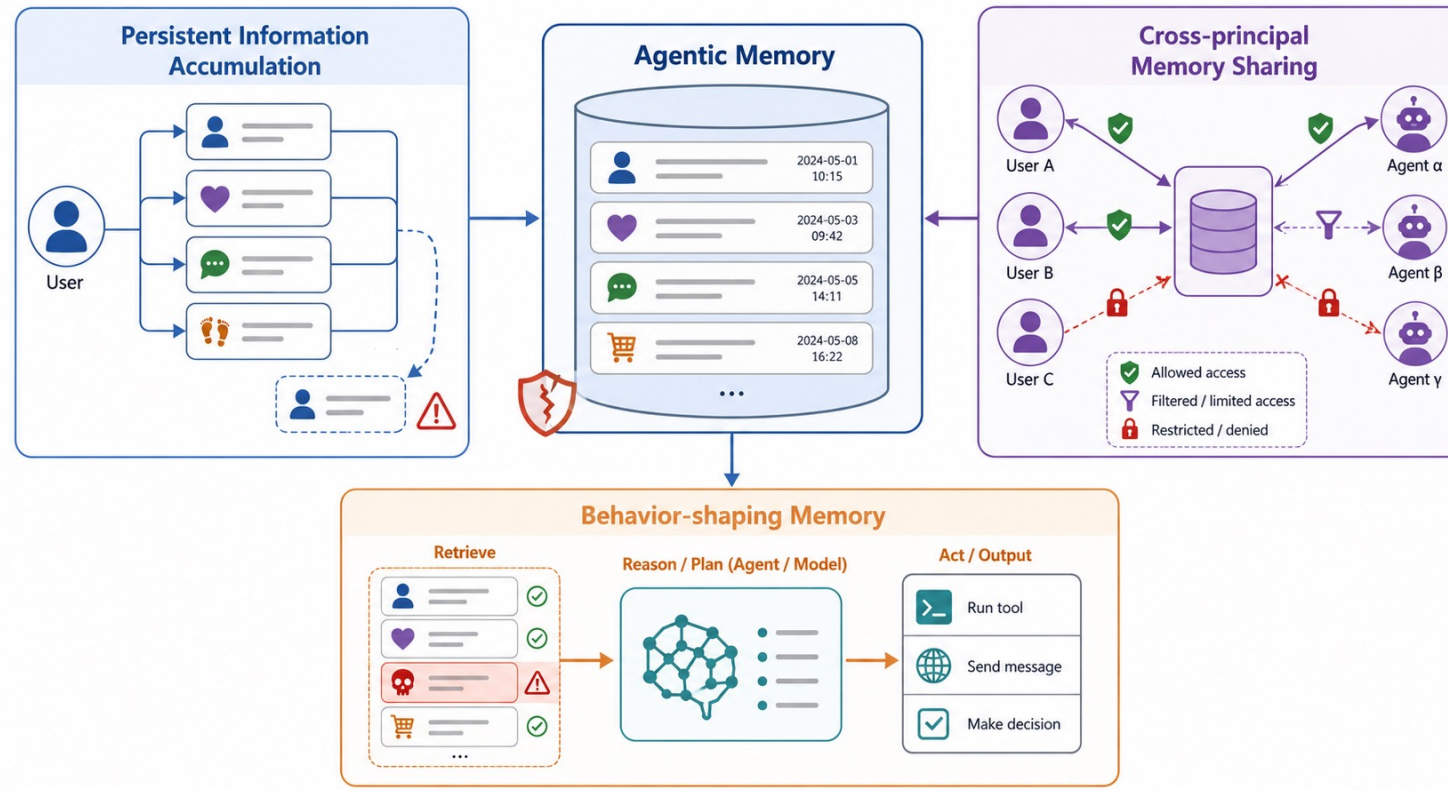
Operators	Methods	Representative Systems
Extraction	Raw trace logging	Openclaw, ClaudeCode, MemoryBank, MemGPT, Generative Agents, Zep
	Semantic Summary	Mem0, MemoryBank, MemGPT, LightMem, A-Mem
	Structured Extraction	ChatDB, Zep, Mem0g
Update	Append-Only Update	Mem0 v3
	Rule-Driven Update	BeliefMem, A-MAC
	LLM-Based Update	Mem0, ChatDB, Zep, A-Mem, MemGPT
	RL-Based Update	Memory-R1, MemLLM, AgeMem, AtomMem
Conflict Detection	Rule-Based Detection	ChatDB, CDRs
	LLM-Based Detection	Mem0, Mem0g, Zep, MemGPT, A-Mem
Retrieval	Text-Based Retrieval	Mem0 v3, MemTree
	Structure-Based Retrieval	Zep, Magma, xMemory
	Post-Retrieval Processing	Mem0 v3, Zep

Memory Operators Overview

Operators	Methods	Representative Systems
Forgetting	Capacity Constraint Trigger	MemGPT, Learning What to Remember, MemoryOS, Mirix
	Retention Threshold Trigger	ReMe, FadeMem, EvolveR, MemoryBank
	Utility-Based Deletion Policy	MemoryBank, MemoryOS, TraceRetain, Learning What to Remember, ReMe, FadeMem
Consolidation	Similarity-Indexed Consolidation	A-Mem, Mem0
	Typed Relational Consolidation	Mem0g, Magma, Zep
	Hierarchical Consolidation	MemTree, xMemory, Zep, MemoryBank
Compression	Single-item Compression	SeCom, LightMem, ACON, MemAgent, MemGPT
	Multi-item Compression	MemRefine, SimpleMem, Infini Memory, RecMem
Reflection	Declarative Knowledge Abstraction	Generative Agents, HeLa-Mem, MemoryBank
	Procedural Knowledge Abstraction	ExpeL, AWM, ReasoningBank

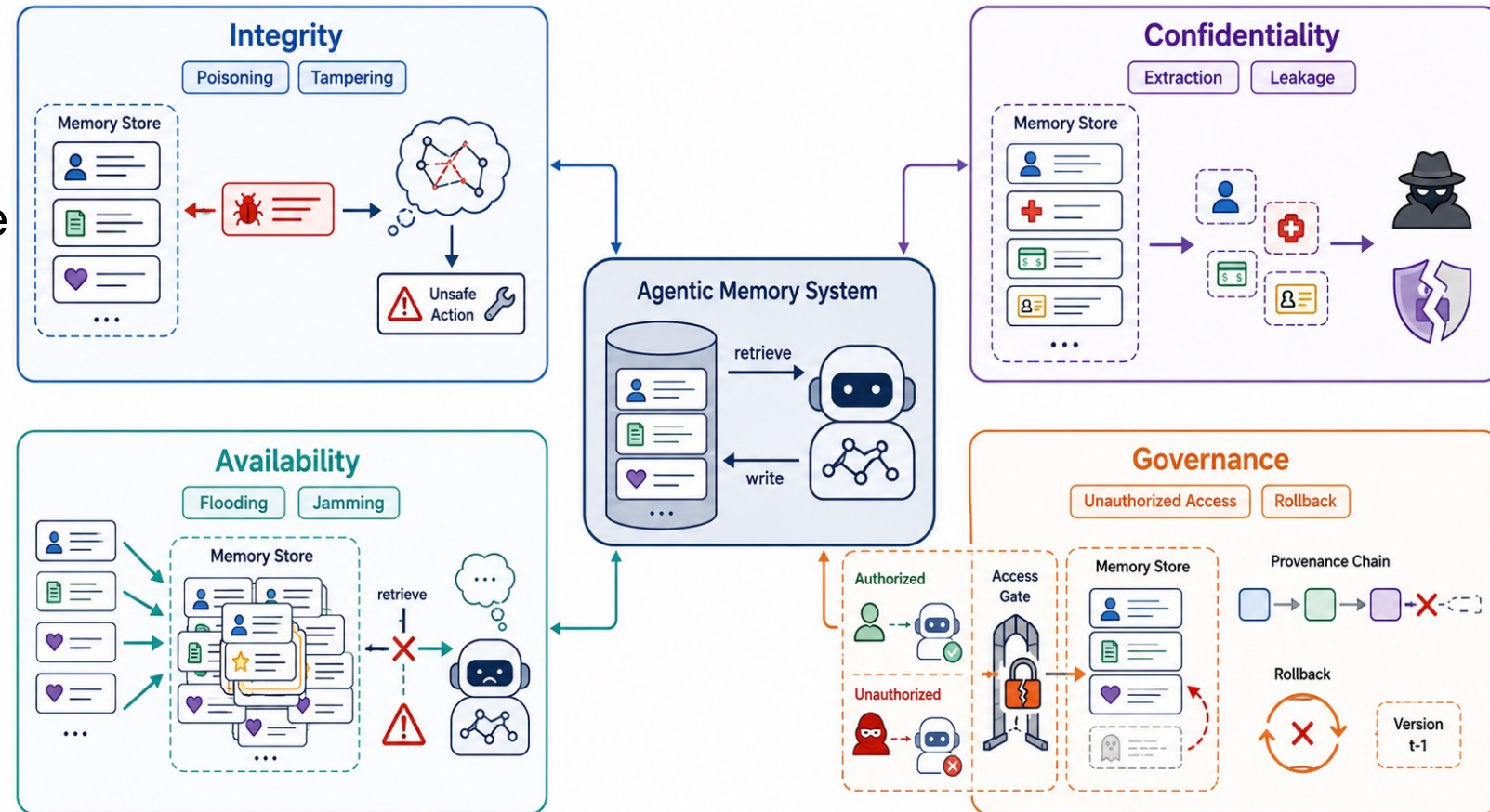
Background of Safety Management

- **Persistent Information Accumulation:** Agentic memory continuously accumulates rich user information across sessions, turning transient interactions into a concentrated and highly sensitive data asset.
- **Cross-principal Memory Sharing:** Shared memory improves knowledge reuse and multi-agent collaboration, but introduces complex ownership, trust, and permission boundaries.
- **Behavior-shaping Memory:** Retrieved memories directly influence model reasoning and actions, allowing incorrect or malicious content to cause persistent and cross-agent behavioral effects.



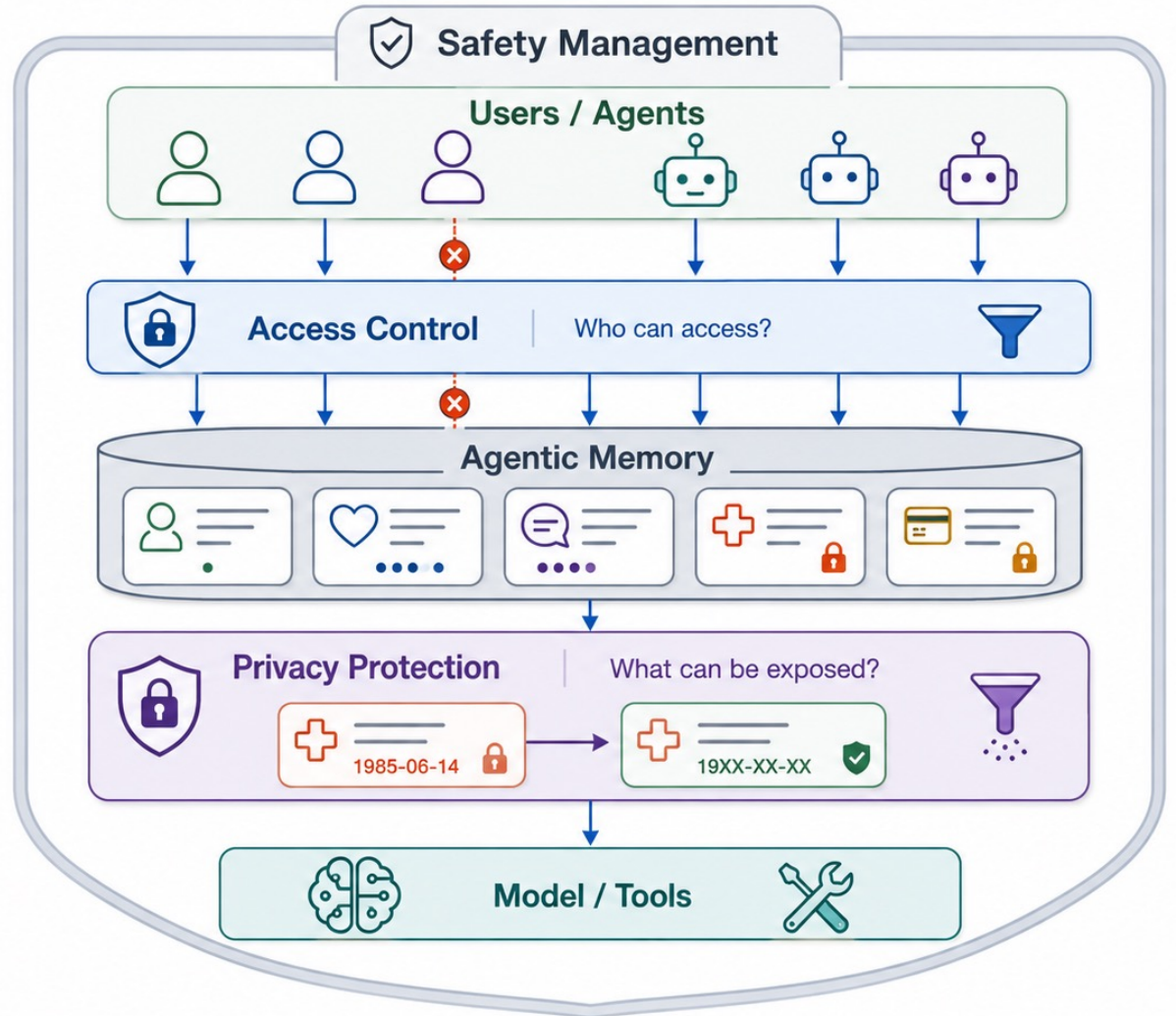
Threat Landscape of Agentic Memory

- **Confidentiality:** Extract, expose, or exfiltrate sensitive memories beyond authorized users or agents.
- **Integrity:** Poison or tamper with stored memories to manipulate future reasoning and actions.
- **Availability:** Flood, suppress, or manipulate the memory store so that legitimate memories cannot be retained or retrieved.
- **Governance:** Break authorization, provenance, auditing, revocation, or rollback mechanisms, making security violations difficult to prevent, trace, or recover from.



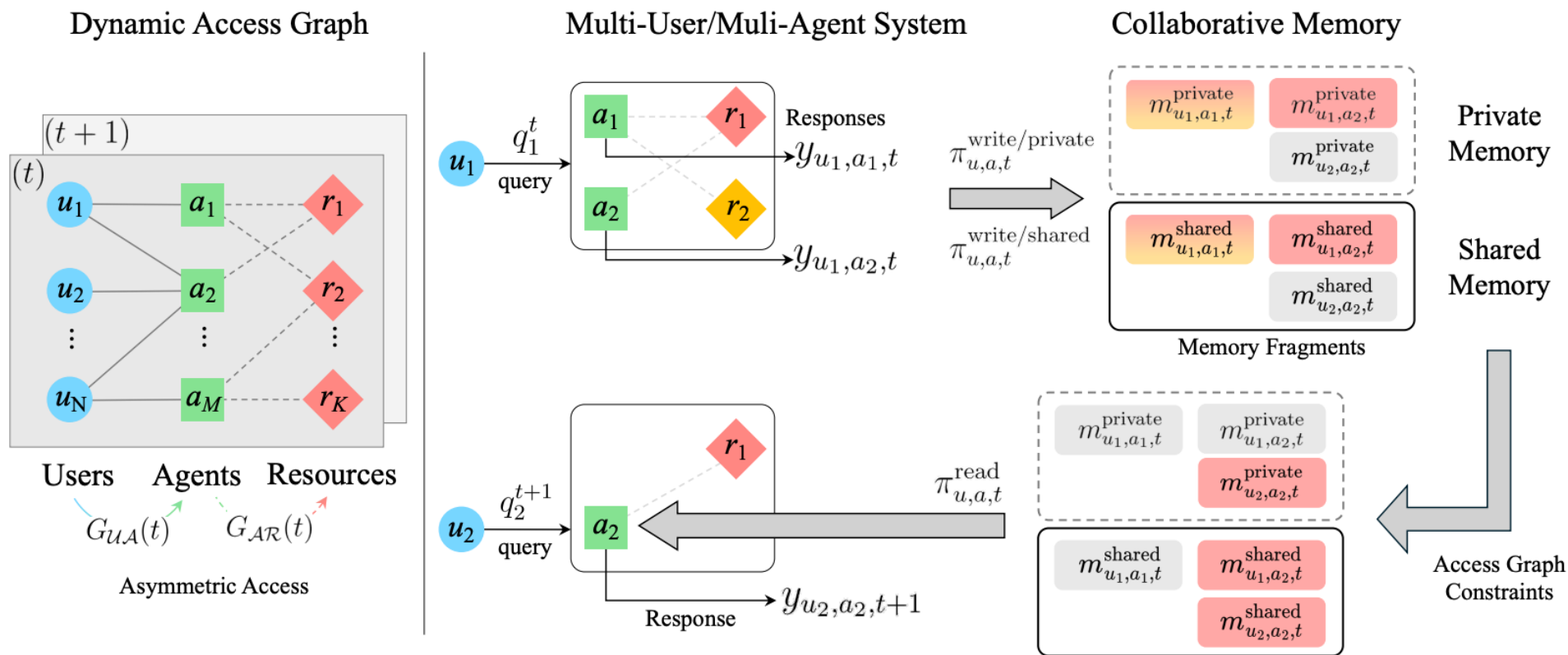
From Memory Threats to Safety Management

- Safety management governs **who can access** memory and **what** sensitive information **can be stored, processed, and disclosed**.
- **Access Control:** Regulates which users or agents can read, write, modify, or share each memory under specific conditions.
- **Privacy Protection:** Reduces the exposure of sensitive information throughout memory storage, retrieval, and use.



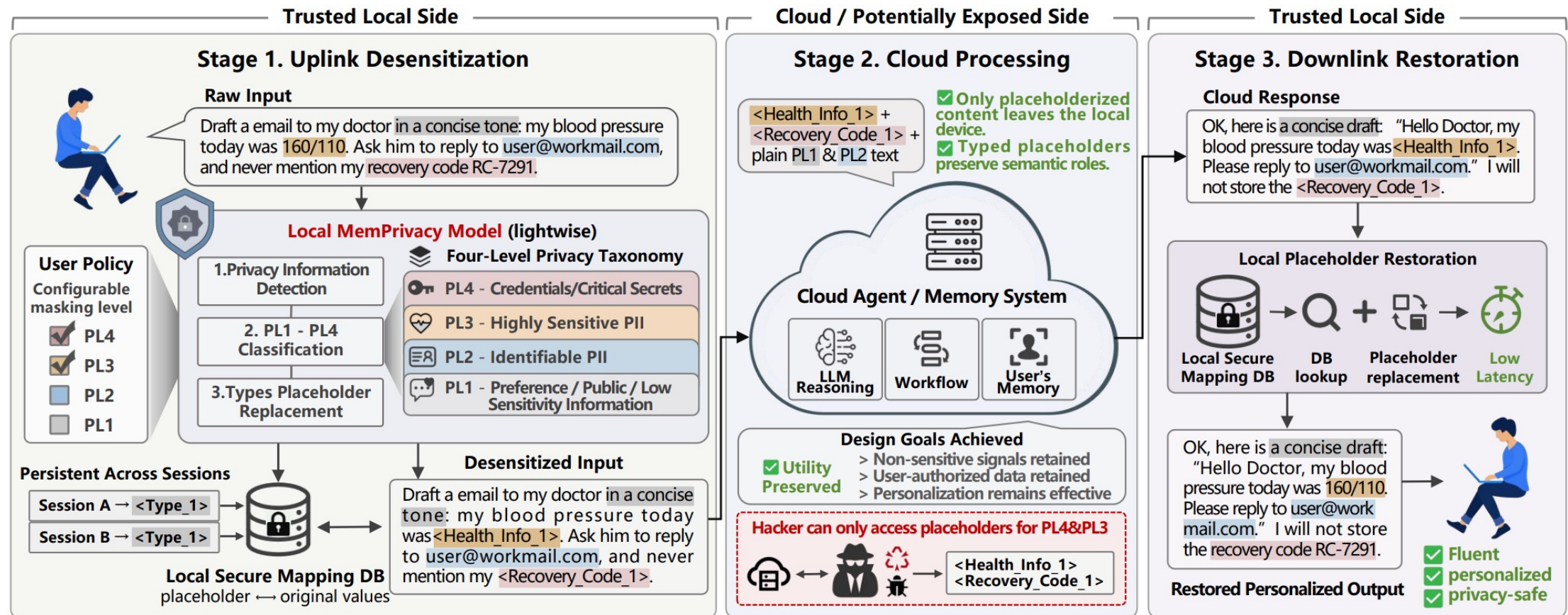
Access Control

- **Motivation:** Shared memory enables cross-user knowledge reuse and multi-agent collaboration, but breaks the assumption that all memories belong to a single trusted principal.
- **Goal:** Enforce fine-grained, asymmetric, and dynamically changing permissions without preventing legitimate memory sharing and reuse.



Privacy Protection

- **Motivation:** Agentic memory accumulates sensitive personal information that may remain exposed throughout storage, retrieval, sharing, and downstream use.
- **Goal:** Minimize sensitive information exposure while preserving the semantic utility required for memory retrieval, reasoning, and personalization.



Memory Storage

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

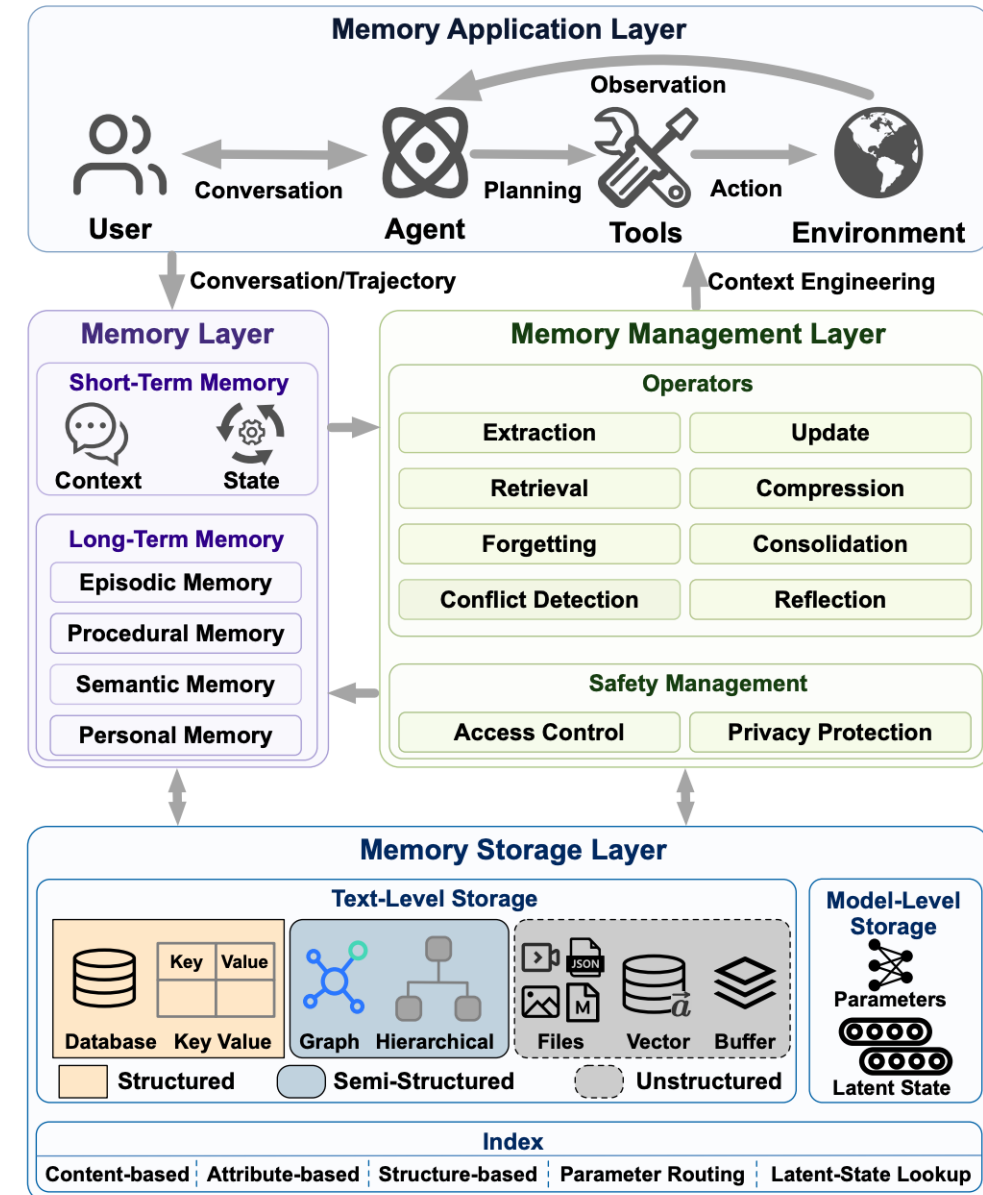
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

❑ Evaluation

❑ Open Problems



Memory Storage

- **Text-Level Storage**

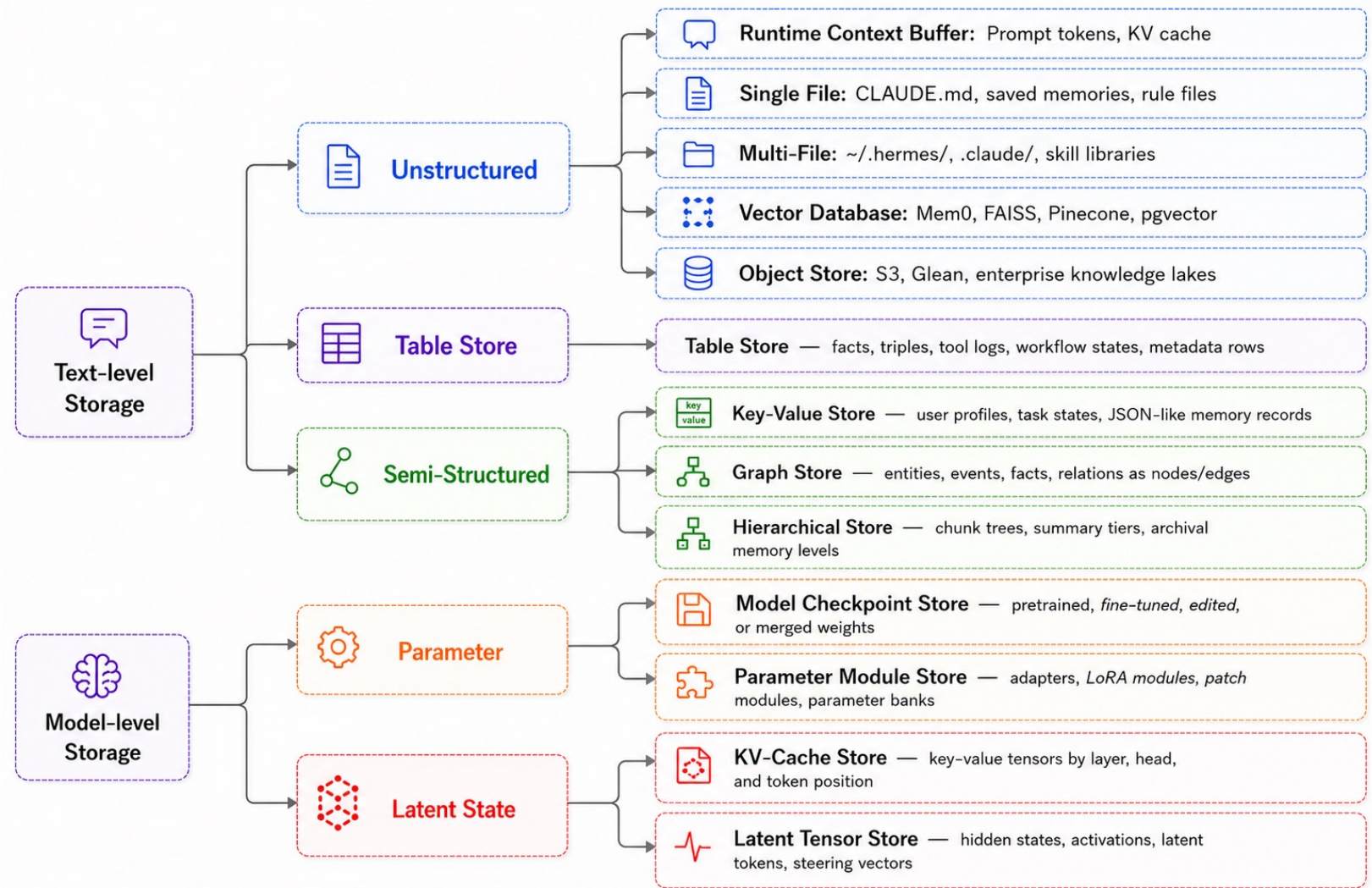
- Unstructured Storage
- Structured Storage
- Semi-Structured Storage

- **Model-Level Storage**

- Parameter Memory
- Latent Memory

- **Index Layer**

- Content-based Index
- Attribute-based Index
- Structure-based Index
- Parameter Routing
- Latent-State Lookup



Text-Level Storage

Storage class	Subclass	Memory Representation	Representative Systems
Unstructured Storage	Runtime Context Buffer	Active prompt/context memory, recent dialogue, tool I/O, working summaries, internal state stored in prompt blocks	MemGPT (main context / working context), MEM1 (internal state), MemAgent (fixed-length prompt memory), Context Folding (folded context), Think-in-Memory (thought records), Memory-as-Action (editable prompt context), Reflexion (reflection buffer)
	Single File	Long-form text documents, notes, transcripts, summaries, markdown/html/plain-text docs, reflection notes, reasoning cases, skill documents	MemGPT (archival storage), Mem0 (extracted fact documents), MemoryBank (summaries / personality texts), ExpeL (experience documents), ReasoningBank (reasoning strategy documents), GraphRAG (community summary reports), RAPTOR (summarized text nodes), SGMem (vector-indexed documents), Second-Me (user documents / profile summaries), Zep (community summaries), Claude Code(CLAUDE.md / topic .md), HtmlRAG (HTML)
	Multi-File	Organized file collections, skill libraries, dialogue archives, code artifacts, multimodal files, file blobs, object blobs	Generative Agents (memory stream), MemGPT (recall storage / dialogue logs), MemoryBank (chronological dialogue logs), ReSum (raw tool observations), Voyager (skill library)
	Vector Database	Embedding-indexed memory chunks, semantic similarity retrieval, vector-indexed documents and facts	Mem0 (vector-indexed memory), FAISS-based retrieval systems, Pinecone-backed memory stores, pgvector-based memory systems, SGMem (vector-indexed documents), HippoRAG (knowledge graph + vector retrieval)
	Object Store	Binary objects, archived data blobs	S3-backed memory systems, Glean (enterprise knowledge), knowledge lake integrations
Structured Storage	Table	Rows / columns for facts, metadata, states, tool logs, workflow records, SQL-managed memory rows	ChatDB (SQL database tables), RET-LLM (triplet table), SimpleMem (memory unit rows / multi-view indexes)
Semi-structured Storage	KV / JSON	Keyed/fielded facts, user profiles, memory slots, JSON-like records, document objects with fields, typed payloads	MemOS (MemCube records), LightMem (topic-indexed memory entries), Think-in-Memory (hash-indexed thoughts), MemoryOS (persona / user trait slots), MemLoRA (JSON fact store), Voyager (skill library), Mem0g (vector-indexed memory records), A-MEM (note attributes), EverMemOS (structured MemCell tuples), MemoChat (JSON memo), MemoryOS (dialogue page)
	Graph	Entities, events, facts, episodes, relation edges, temporal links, triplet-like records	Zep (temporal knowledge graph), Mem0g (entity-relation graph), A-MEM (linked note graph), AriGraph (episodic knowledge graph), GraphRAG (community graph), HippoRAG (knowledge graph), EMG-RAG (editable memory graph), SGMem (sentence graph)
	Hierarchical / Tree	Trees, tiers, pages, clusters, summary hierarchies, scene levels, parent-child links	RAPTOR (recursive summary tree), MemoryOS (STM/MTM/LPM hierarchy), EverMemOS (MemScene / MemCell hierarchy), H-Mem (semantic memory hierarchy), HiAgent (subgoal memory chunks), MemoryBank (daily-to-global summary tiers)

Unstructured Storage

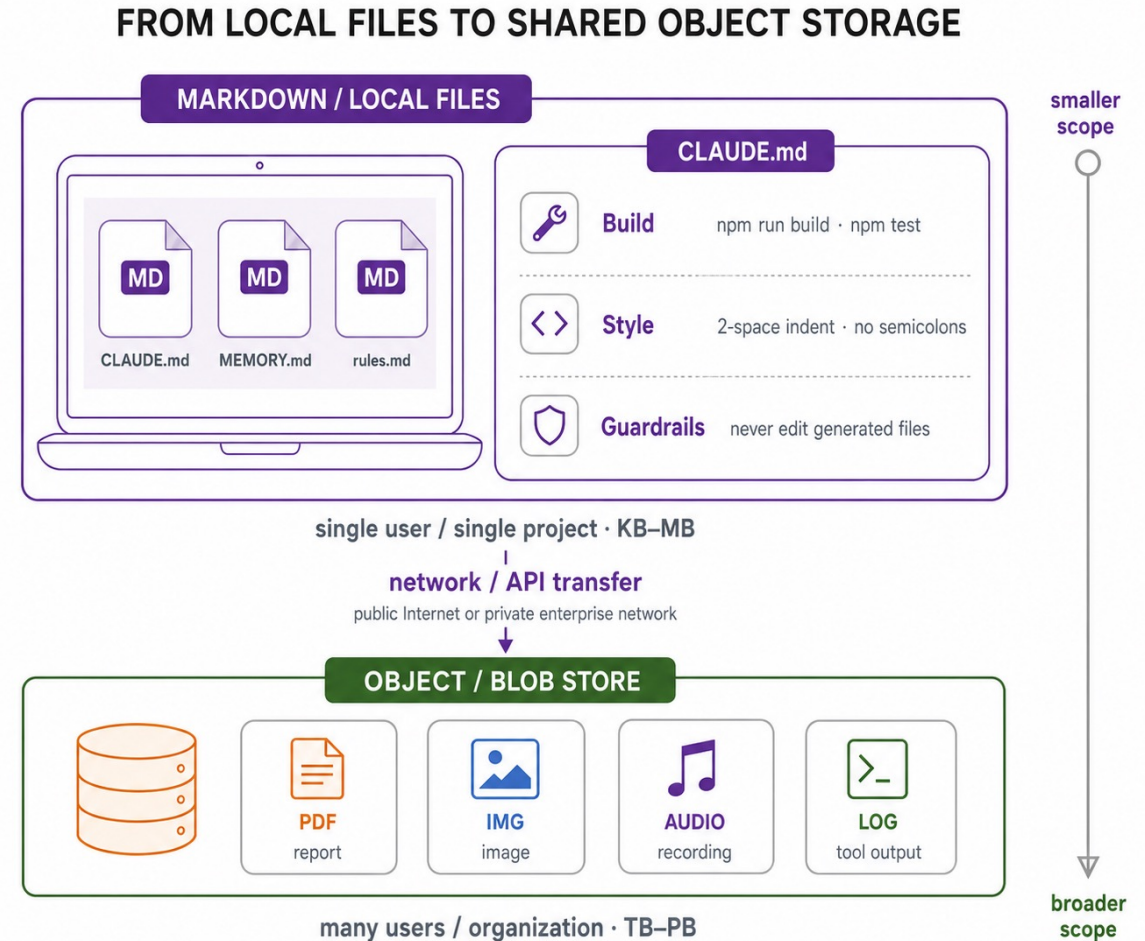
Unstructured storage preserves payloads without imposing a fixed schema.

- **Markdown / Text Files**

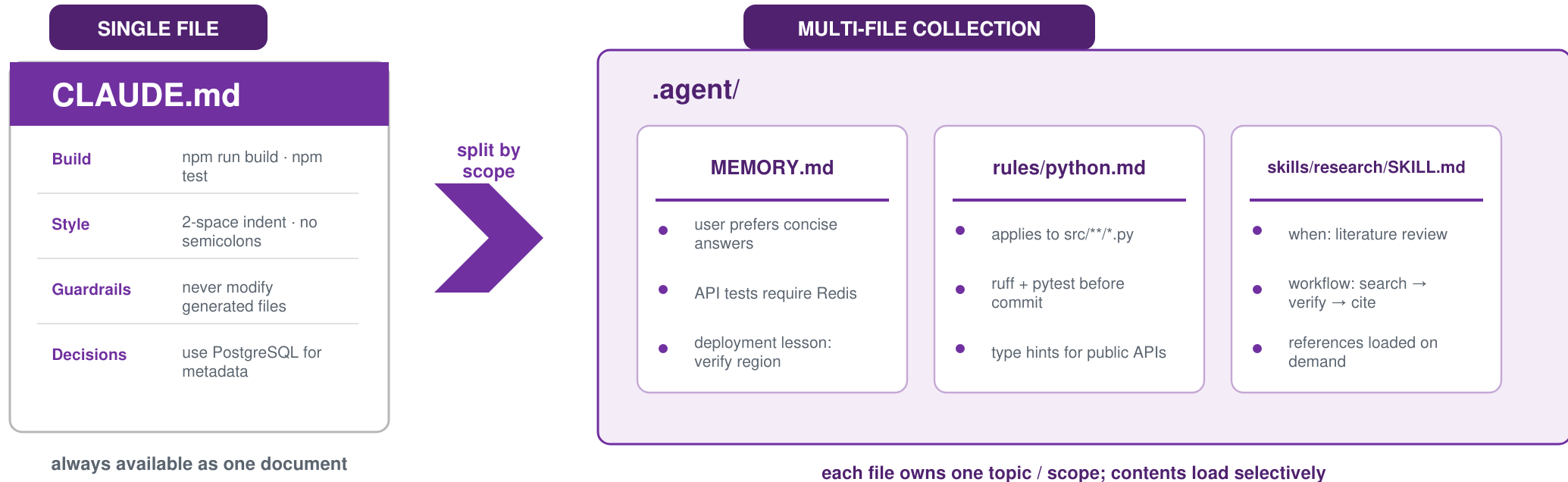
- Natural-language memory with lightweight markup
- Local files for one user, project, or scoped task

- **Object / Blob Storage**

- Raw documents and multimodal artifacts
- Shared payloads preserved with keys, versions, and policies



Markdown / Text Files



Storage Design

- Natural-language facts, rules, preferences, lessons
- One stable file or topic- and scope-specific files

Files organize scope and responsibility; the memory payload remains free text.

Representative Systems

- Claude Code: CLAUDE.md, auto-memory, rules
- Hermes: SOUL.md, USER.md, MEMORY.md

Storage Trade-off

- Strength: readable, editable, versionable
- Limitation: capacity and consistency at scale

Object / Blob Storage

Storage Design

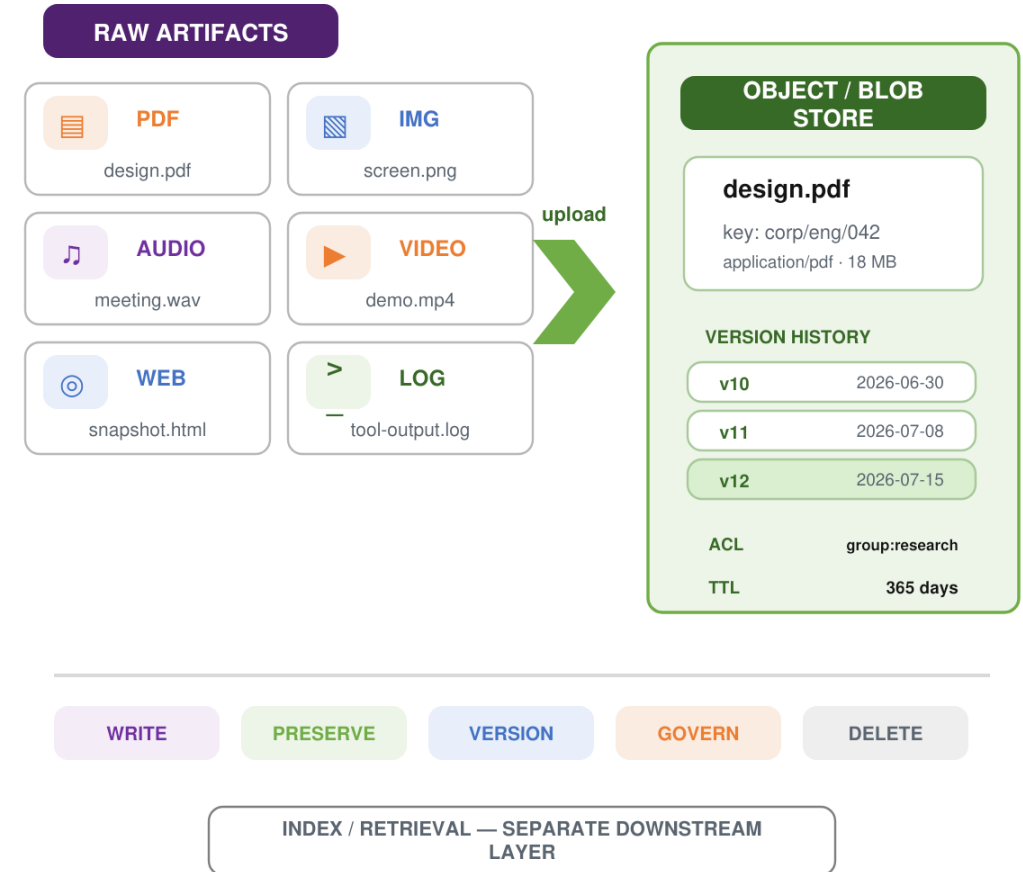
- Payload: PDFs, images, audio, web snapshots, logs
- Envelope: key, content type, version, ACL, retention

Representative Systems

- Amazon S3, Azure Blob, Google Cloud Storage
- Uploaded knowledge-file repositories

Storage Trade-off

- Strength: scalable, durable, and multimodal
- Limitation: parsing and governance overhead



Structured Storage

- ❑ **Structured storage turns memory into explicit keys, fields, or schemas so agents can update, filter, and join memories predictably.**

Table / Record Store

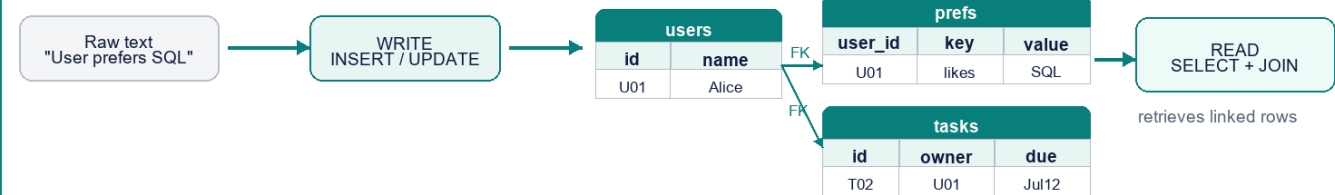
- Stored items are repeated facts, triples, tool logs, workflow states, and metadata.
- Their fields are stable—entity, relation, time, state, value—so rows or typed JSON make lookup and update predictable.

Three relational storage patterns for agent memory

Same input text, but different table design, write operation, and retrieval logic.

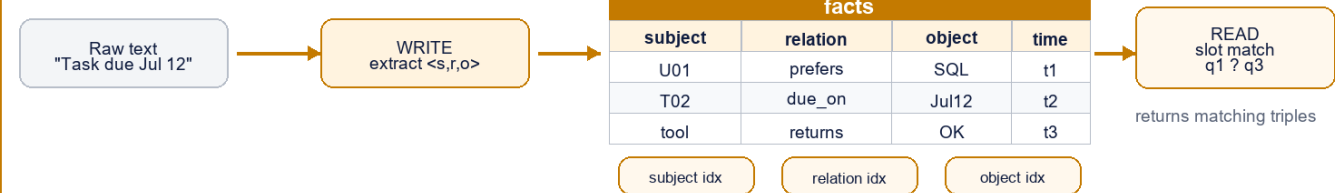
A Schema-first SQL rows

fixed schema + PK/FK



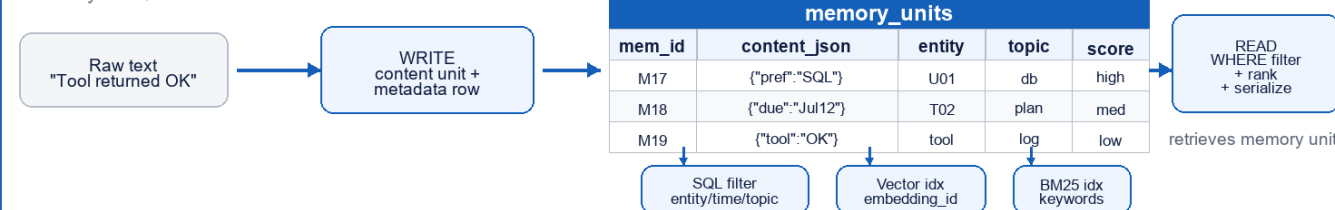
B Triple fact memory

one fact table + slot indexes



C Memory unit + metadata index

unit body + SQL filters + side indexes



No work names shown: diagram encodes storage patterns only

Structured Storage

- **Content & Fit**

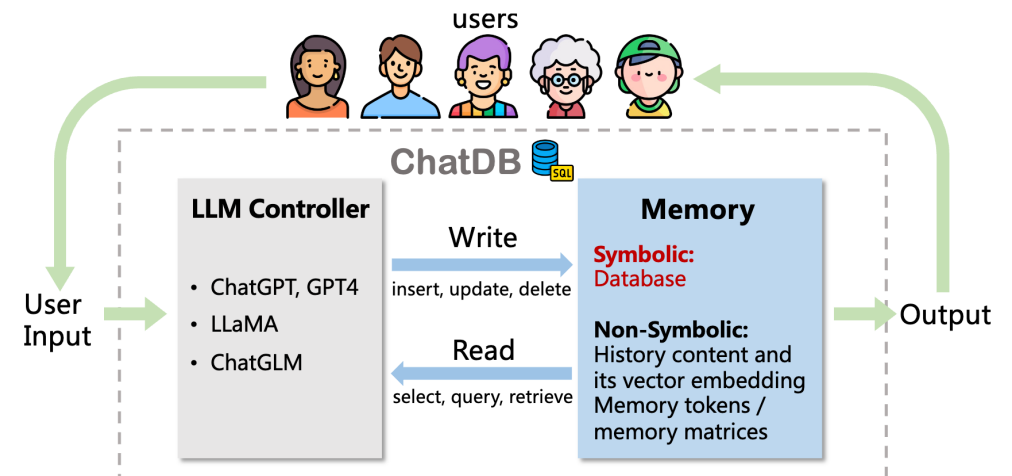
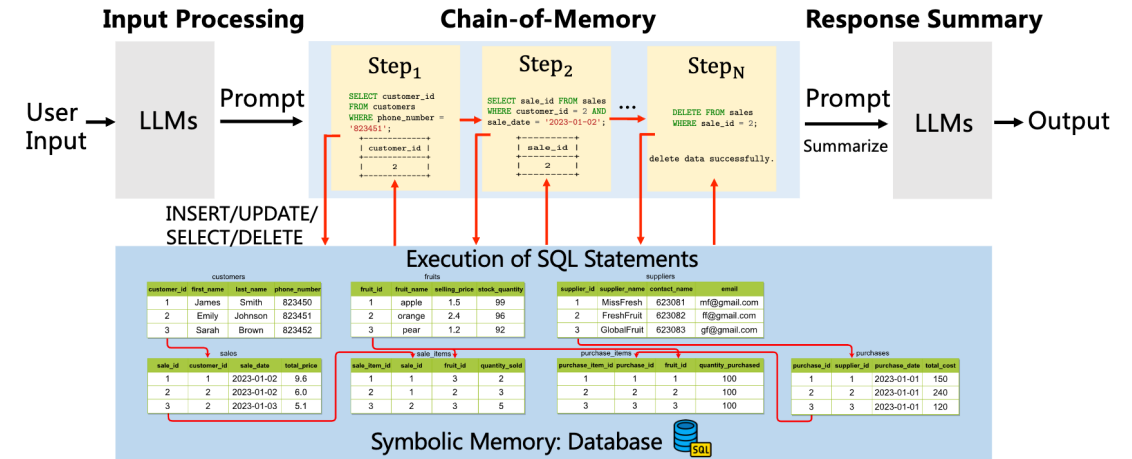
- Here, memory behaves like domain data: customers, fruits, sales, and task states. Its fields are stable and often cross-referenced, so tables make joins, counts, and updates explicit

- **Storage Design**

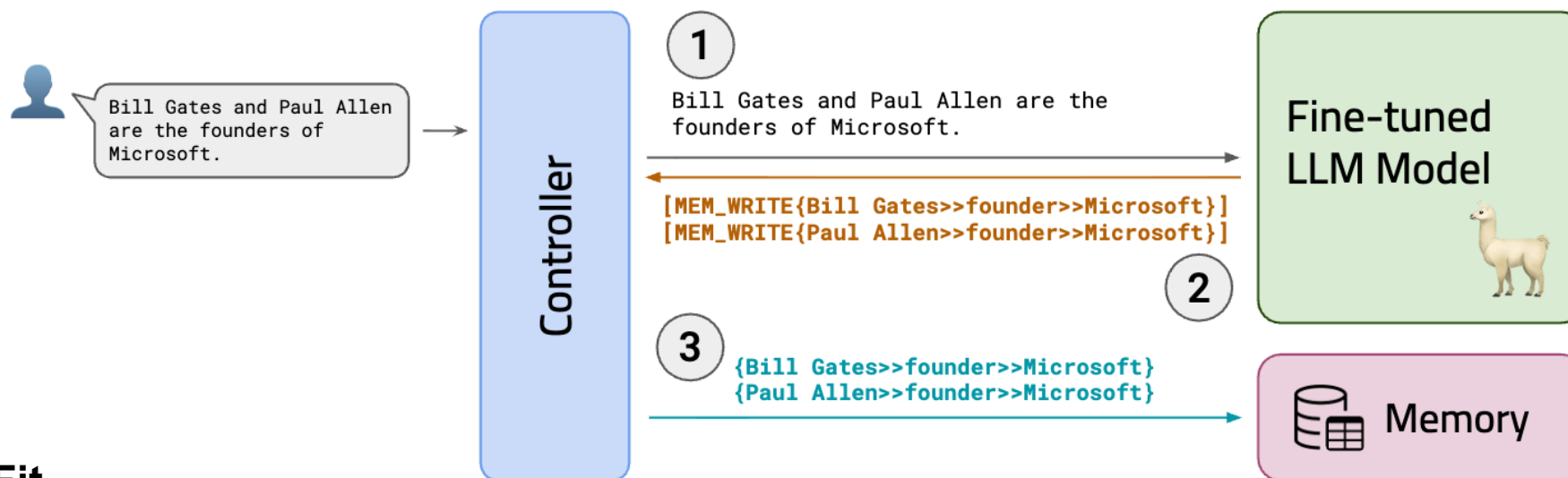
- **Schema-first:** ChatDB builds task-specific tables such as customers, fruits, sales, and sale_items.
- **Text-to-memory:** The LLM translates natural-language records into SQL writes over those tables.
- **Retrieval:** Chain-of-Memory decomposes questions into SELECT / JOIN / aggregate SQL steps, then summarizes returned rows.

- **Storage Trade-off**

- **Strength:** Precise multi-step reasoning over structured memory; complex joins and updates are directly supported.
- **Limitation:** Schema is hard to evolve; SQL errors can cascade; weak for ambiguous or unstructured memories.



Structured Storage



- **Content & Fit**

- These memories are atomic facts with subject–relation–object slots. A triple table fits because any slot can become the query key, and the missing slot becomes the answer.

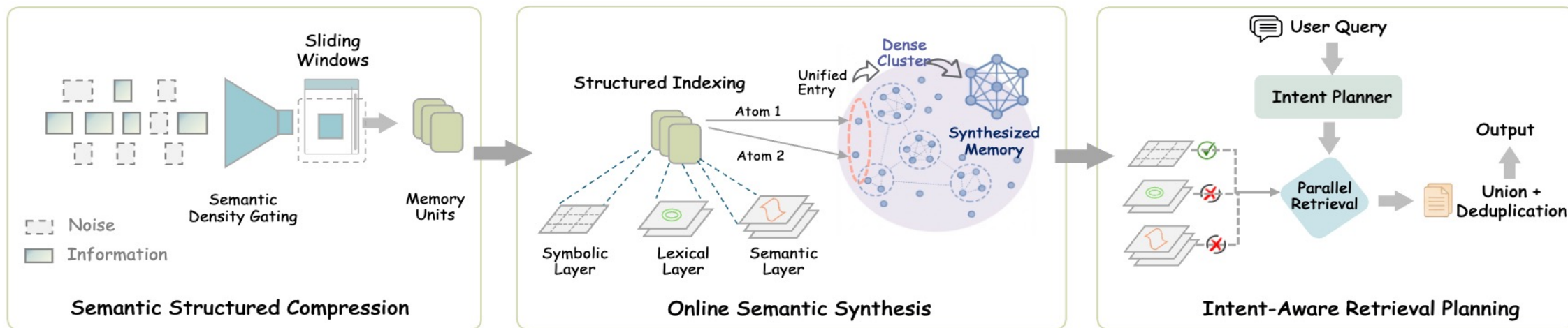
- **Storage Design**

- **Text-to-memory:** The fine-tuned LLM emits [MEM_WRITE{t1>>rel>>t2}] calls to extract one or more triples.
- **Storage:** Triples are saved in a three-column table, with vector representations for each term.
- **Retrieval:** The LLM emits [MEM_READ{q1>>q2>>q3}] with one or two slots filled to find matching triples.

- **Storage Trade-off**

- **Strength:** Clean factual representation with explicit read/write access.
- **Limitation:** Loses contextual nuance; mostly binary relations; temporal and episodic context are not directly modeled.

Structured Storage



- **Content & Fit**

- Dialogue is split into indexable event units, each carrying entities, topic, time, and salience. Tables let the same memory be reached from different fields without duplicating the text.

- **Storage Design**

- **Text-to-memory:** LLM filters chitchat, resolves pronouns, normalizes relative time, and atomizes dialogue.
- **Storage:** Each unit stores content plus metadata; entity/topic/link tables keep the indexable views.
- **Retrieval:** Intent planner converts entity / topic / time / salience constraints into WHERE filters and JOINS.

- **Storage Trade-off**

- **Strength:** Flexible dimension-specific retrieval over a shared memory base.
- **Limitation:** Fixed schemas and indexes may age poorly; index maintenance grows with memory size.

§4.1.3 Semi-Structured Storage

❑ Methods in the same family may still store very different things.

• Flat Record Storage

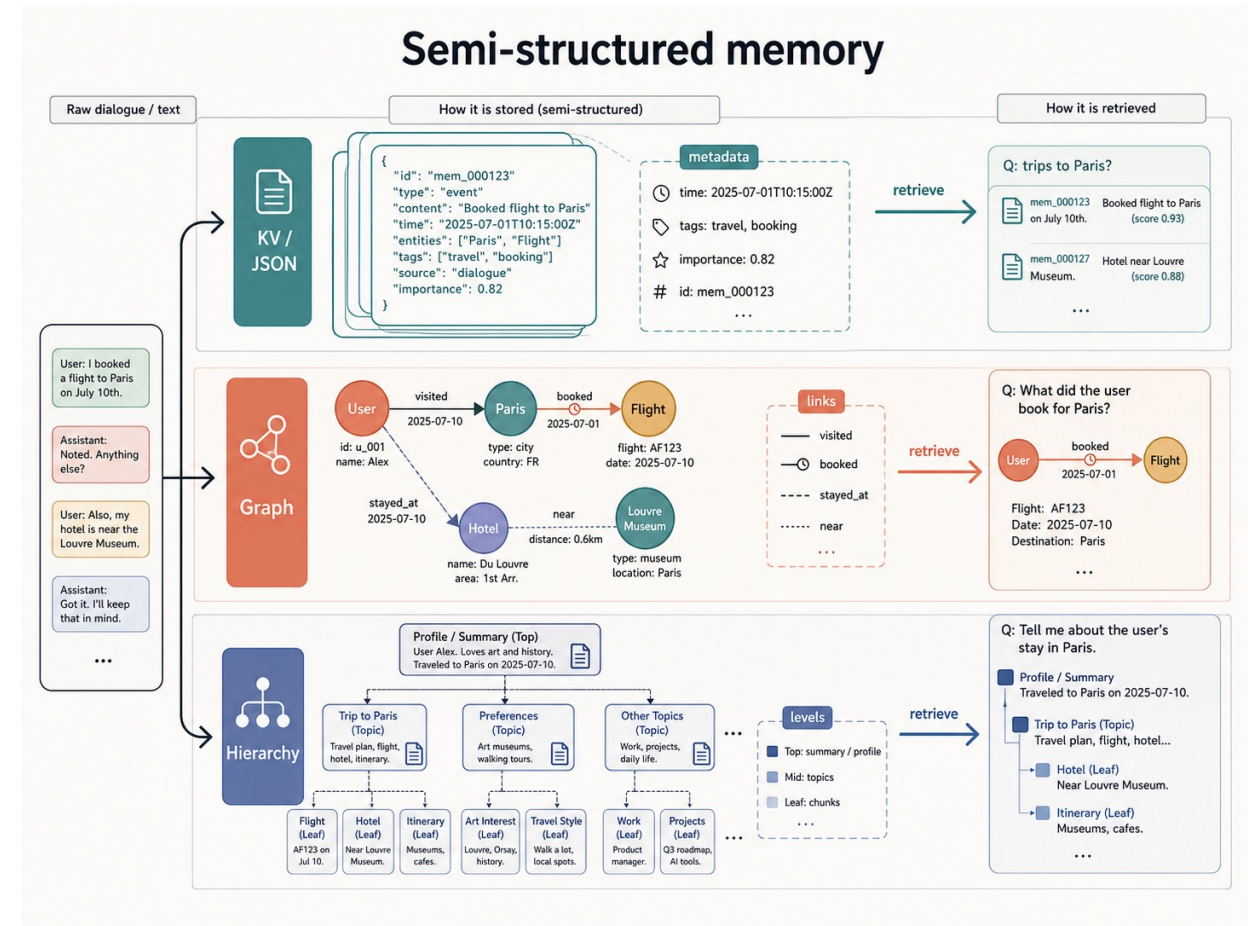
- Some keep small JSON objects or table rows. Others store dialogue logs as flat vectors in databases.

• Graphs & Added Views

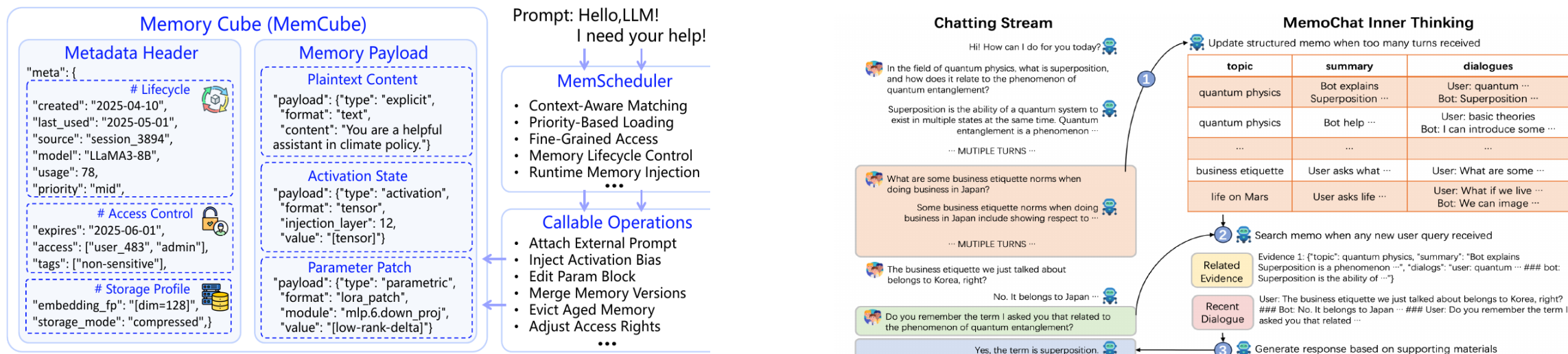
- Some build online editable graphs with temporal or associative links. Others layer pre-computed community summaries or static activation propagation paths.

• Layers and Trees

- Some layered stores partition memory into temporal, activity, or task-state hierarchies. Others build tree structures for routed search.



Semi-Structured (Flat Record Storage)



• Content & Fit

- Flat records fit memory that already stands alone: a fact card, topic register, summary, or trait vector. The unit can be appended, edited, filtered, or embedded directly.

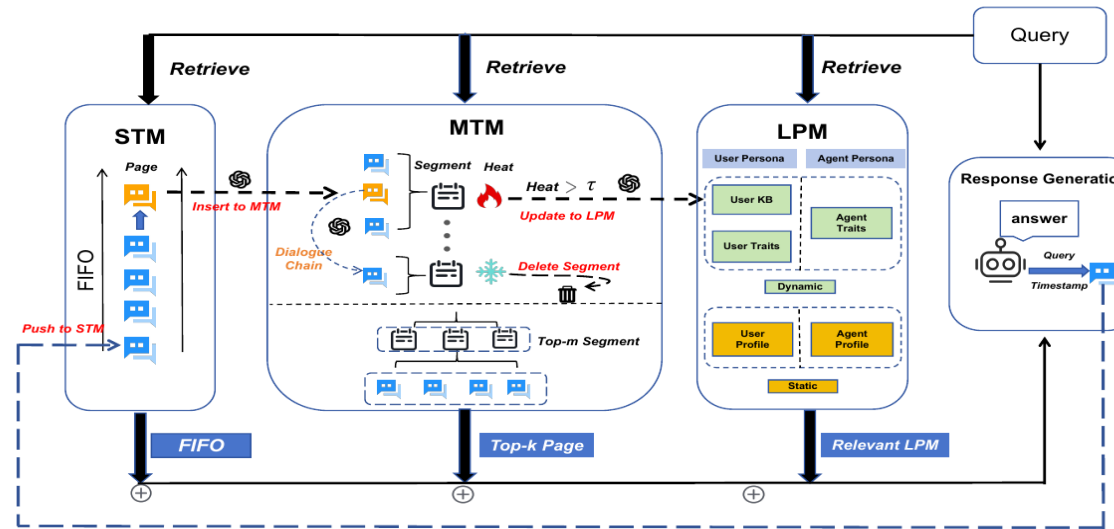
• Storage Design

- MemOS:** Fine-grained JSON objects with metadata headers and typed payloads.
- MemoChat:** Topic-segmented dialogue registers combining history and summaries.
- MemoryBank:** Dialogue summaries and personality descriptors stored as text records plus dense vectors.

• Storage Trade-off

- Strength:** Fine-grained records support precise filtering; vector records enable fast recall.
- Limitation:** Edits may rewrite entire objects, while independent vectors often lose temporal/topic context.

Semi-Structured (Layered Storage)



- **Content & Fit**

- Layered stores target memories with different lifespans: fresh turns, active tasks, summaries, and profiles. Volatile detail stays near the agent; stable knowledge is compacted.

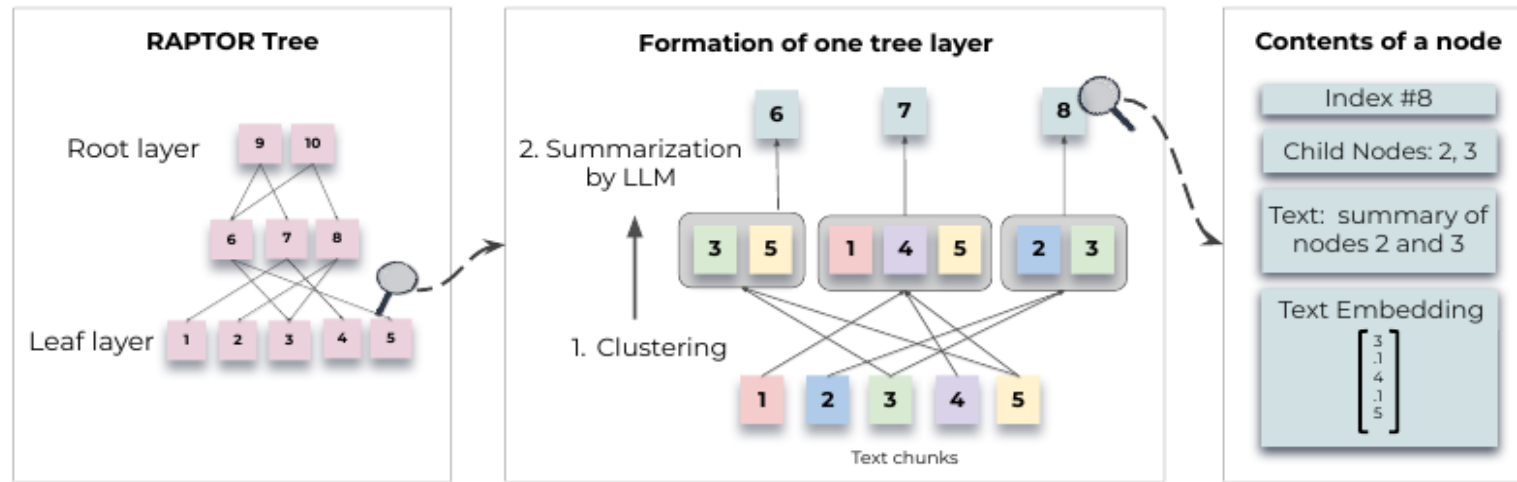
- **Storage Design**

- **MemoryOS:** Promotes threads across short-term buffers, mid-term threads, and long-term profiles via FIFO/page compaction.
- **EverMemOS:** Maintains episodic logs, thematic summaries, and user profiles with aging and consolidation.
- **HiAgent:** Keeps active-task logs in detail and compresses completed tasks into summaries.

- **Storage Trade-off**

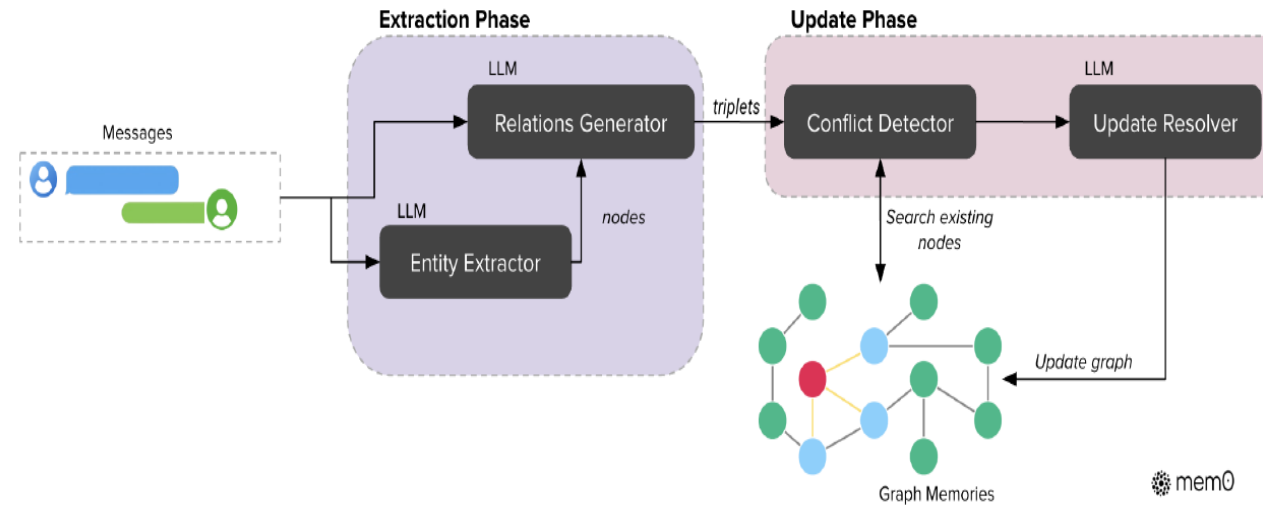
- **Strength:** Preserves local execution context while allowing old memory to be compacted.
- **Limitation:** Promotion and task segmentation can hide details or add clustering/maintenance cost.

Semi-Structured (Tree-Based Storage)



- **Content & Fit**
 - Tree storage fits chunk-and-summary memory. Leaves keep evidence; parents carry themes or labels, letting retrieval start broad and drill down only when needed.
- **Storage Design**
 - **RAPTOR:** Recursively clusters text segments and generates parent summaries to connect macro-themes with micro-details.
 - **H-MEM:** Routes memories through a predefined taxonomy tree (Domain → Category → Episode) with positional indices.
- **Storage Trade-off**
 - **Strength:** Efficient coarse-to-fine search across large memory sets.
 - **Limitation:** RAPTOR requires rebuilding summaries after edits; H-MEM can hide memories when early classification is wrong.

Semi-Structured (Editable Graphs)



- **Content & Fit**

- Graphs fit memories whose value lies in connections: preferences tied to people, events ordered in time, notes linked by association. Nodes keep items; edges keep dependencies.

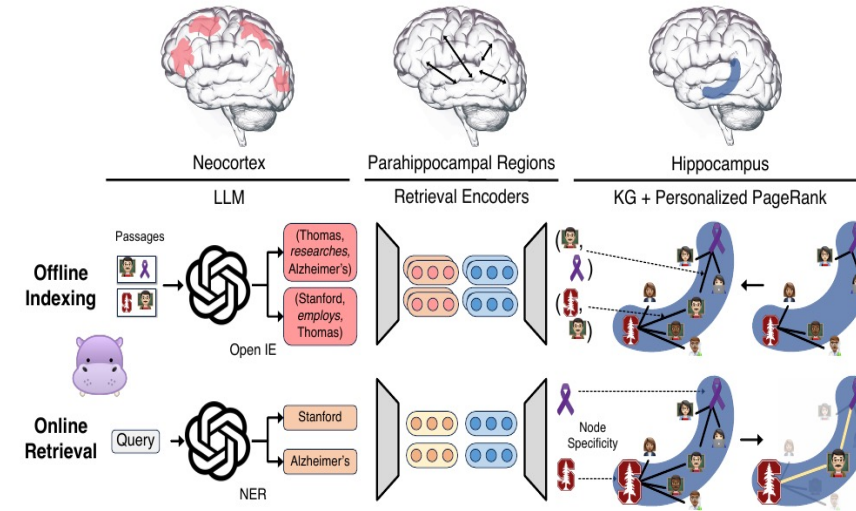
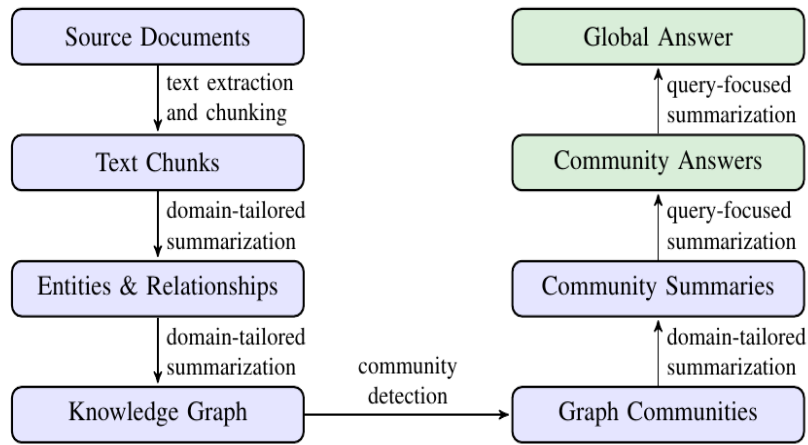
- **Storage Design**

- **Mem0g**: Links preference facts to profile-centered entity nodes.
- **A-MEM**: Connects note nodes with dynamically weighted associative links.
- **AriGraph**: Maintains semantic fact edges and episodic chronological paths.
- **EMG-RAG**: Uses static category anchors above dynamic entity–relation instances.

- **Storage Trade-off**

- **Strength**: Clean relationship modeling and target-specific updates.
- **Limitation**: Entity alignment, graph maintenance, and traversal latency grow quickly; extraction errors can spread across the graph.

Semi-Structured (Graph Views & Indices)



- **Content & Fit**

- Graph views handle memory at several scales: messages/passages, entities, and community summaries. Prebuilt links turn multi-hop reasoning into reusable indexes.

- **Storage Design**

- **Zep**: Message timelines → entity nodes → narrative summary nodes.
- **GraphRAG**: Entity–relation graph with pre-generated subgraph summaries linked to member nodes.
- **HippoRAG**: Episodic passage nodes linked to semantic entity nodes for associative multi-hop retrieval.

- **Storage Trade-off**

- **Strength**: Avoids repeated LLM reasoning by querying compiled summaries or index paths.
- **Limitation**: High offline indexing or online write amplification; failures in entity extraction/alignment harm retrieval.

Semi-Structured (Design Comparison)

Storage family	Common design variants	How the variants differ	Core trade-off
Flat Record Storage	Small typed records Grouped conversation records Vectorized summaries	• Store each fact or event as a separate typed record • Group related conversations into larger topic units • Encode summaries and traits for similarity search	Fine-grained updates vs. richer context vs. faster similarity search
Layered Storage	Time-based layers Content-based layers Task-progress layers	• Move records across short-, medium-, and long-term layers • Separate events, themes, and user profiles • Keep active-task detail; compress completed tasks	Simple writes vs. preserved context vs. consolidation cost
Tree-Based Storage	Automatically built hierarchy Predefined hierarchy	• Build levels by clustering similar content and summarizing parent nodes • Route records through predefined categories	Flexible semantic grouping vs. deterministic routing and classification risk
Graph Storage	Three-layer graph Partitioned entity–relation graph Two-layer passage–entity graph	• Messages → entities → narrative summaries • Entities / relations plus parent subgraph summaries • Passages → semantic entities for multi-hop connections	• Rapid global summarization or single-pass multi-hop retrieval • High offline indexing latency or online write amplification • Retrieval can fail when entity extraction or alignment is incomplete

Model-Level Storage

Storage class	Subclass	Memory Representation	Representative Systems
Parameter Storage	Full Model Parameter Store	Memory internalized in full or edited model weights/checkpoints	MemOS (intrinsic parameters); ROME (weight edit); MEMIT (batch edit); MEND (model edit); AlphaEdit (knowledge edit); WISE (editing memory); Memory ³ (parameterized knowledge)
	Adapter / LoRA Store	Adapters, LoRA modules, patches, auxiliary parameter modules	MemOS (LoRA modules); MemLoRA (LoRA memory); CoLoR (LoRA routing); T-Patcher (patch memory); MLP-Memory (MLP patches); K-Adapter (adapter memory)
Latent Storage	KV-Cache Store	Attention key-value tensors, prefix caches, compressed KV blocks	MemOS (activation memory); LONGMEM (cached memory); Memorizing Transformers (external cache); FOT (memory attention); Memory ³ (KV retrieval)
	Latent Tensor Store	Hidden states, activations, latent tokens, steering vectors, embeddings	AutoCompressor (summary vectors); MemoryLLM (memory tokens); MemGen (latent tokens); M+ (latent memory); MemoRAG (latent retrieval); LM2 (matrix slots); Gist (gist tokens); LatentEvolve (latent evolution); CoMEM (multimodal embeddings); VisMem (visual memory)

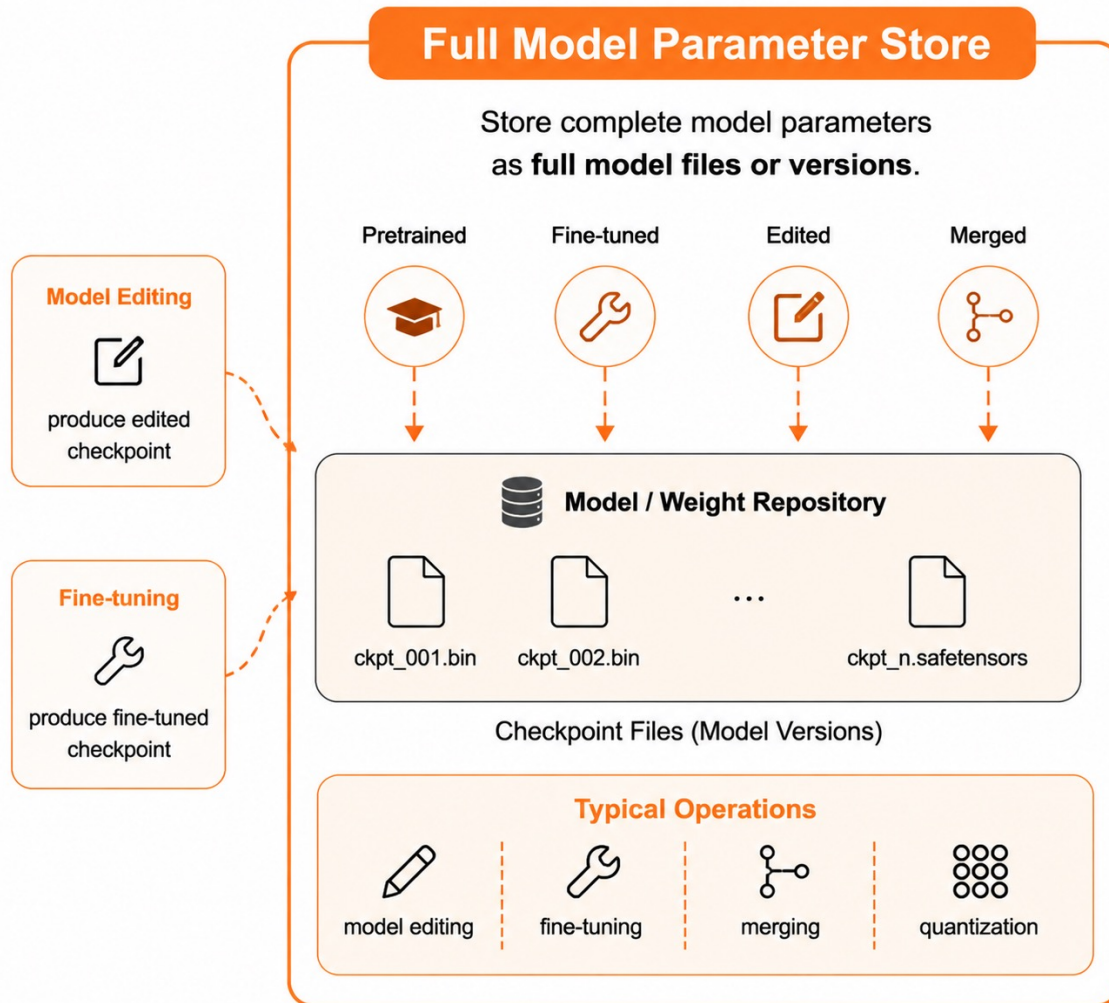
Full Model Parameter Store

- **Storage Design**

- **Data:** Full model weights + tokenizer/config metadata → **memory embedded in the serving model.**
- **Files:** model.safetensors / pytorch_model*.bin · config.json (+ tokenizer files)
- **Flow:** Train/edit → export checkpoint → deploy as serving model
- **Case · ROME:** Treats MLPs as key–value memory; locates the mediating layer and applies a rank-one update, storing the edited checkpoint.

- **Storage Trade-off**

- **Strength:** In-model knowledge updates — no retrieval index, low latency, and generalization beyond the edit prompt.
- **Limitation:** Poor for continual personal/session memory — scaling edits is difficult, with interference and weak rollback/auditability.



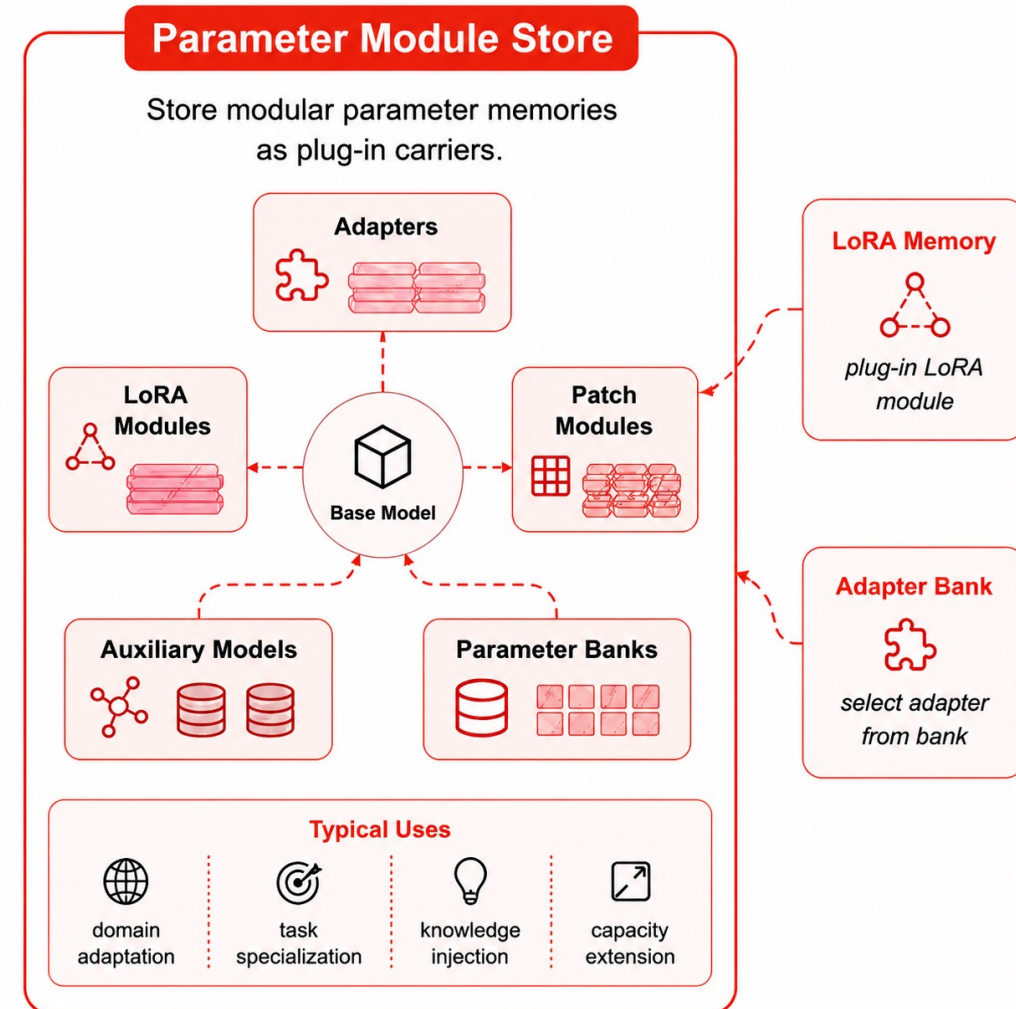
Adapter / LoRA Store

- **Storage Design**

- **Data:** Adapter / LoRA weight deltas + config & base-model metadata → **memory as plug-in parameters; base model stays shared.**
- **Files:** adapter_model.safetensors · adapter_config.json · PEFT-style folders
- **Flow:** Train adapters → save side-car modules → attach or merge with the base model at runtime
- **Case · K-Adapter:** Frozen backbone (e.g., RoBERTa); each knowledge type uses an independent adapter (~42M), trained separately and inserted at selected layers — saved as side-car checkpoints (*pytorch_model.bin) for runtime loading.

- **Storage Trade-off**

- **Strength:** Continual infusion with less forgetting — adapters small, independent, combinable.
- **Limitation:** Hard to surgically update a single fact; routing / managing many adapters grows complex.



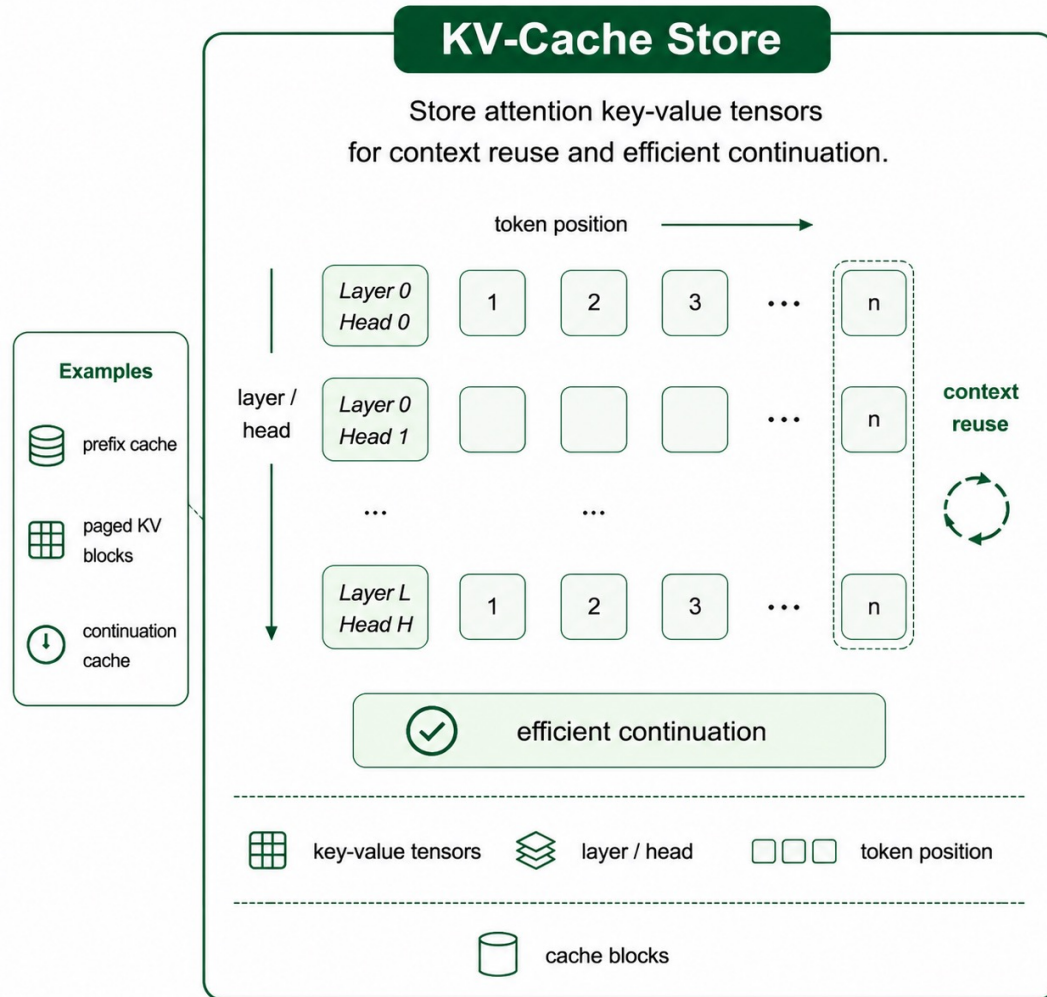
KV-Cache Store

- **Storage Design**

- **Data:** Transformer states (attention K/V tensors) that capture model context → **only persisted, reusable KV counts as memory (not one-shot decode cache).**
- **Files:** .pt / .npy / .safetensors KV · memmap/shard dirs
- **Flow:** Encode experience → persist KV blocks → reuse cached KV without re-encoding
- **Case · Memorizing Transformer:** Stores past-token attention K/V in an external memory bank and retrieves them via approximate kNN for long-context attention — the asset is persisted KV tensors (.pt / .npy blocks), not temporary decode cache.

- **Storage Trade-off**

- **Strength:** Near-lossless continuation of model state — no re-encode of past tokens; natural for “diary / resume thinking” style memory.
- **Limitation:** Needs eviction/merge/quant to stay feasible; weak structure/audit vs text; generic session KV ≠ lasting memory unless deliberately persisted & retrieved.



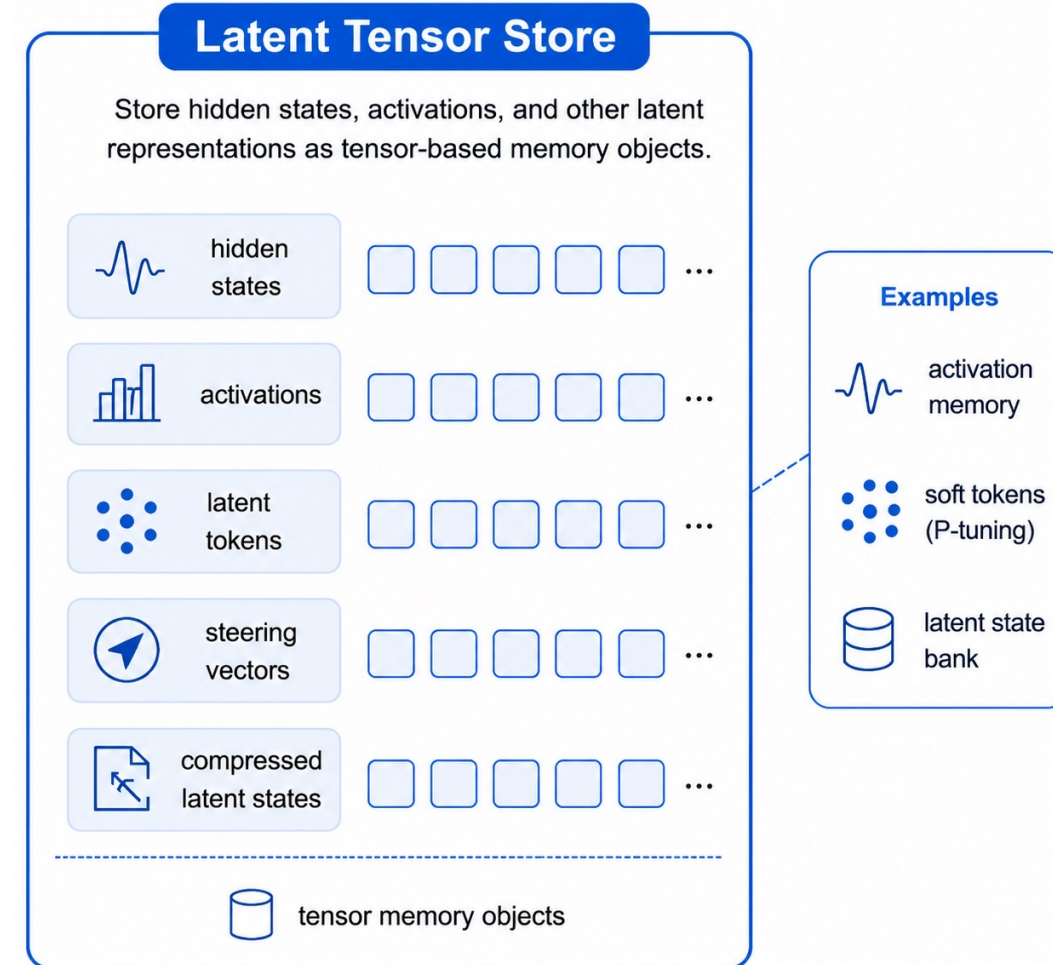
Latent Tensor Store

- **Storage Design**

- **Data:** Compressed representations (e.g., memory tokens, summary vectors, embeddings) that capture memory — **abstract latents, not position-aligned KV.**
- **Files:** .pt / .npy / .safetensors banks · embedding indexes · soft-prompt/gist token files
- **Flow:** Compress/project activations → store latent objects → inject/condition at inference.
- **Case · AutoCompressor:** Compresses long context into latent summary vectors (soft prompts/gist tokens), reused via inference injection — stored assets are latent tensors (.pt / .npy), not full KV caches or raw text.

- **Storage Trade-off**

- **Strength:** Cheaper and more abstract than full context/KV — compact injectables for conditioning and recall without modifying the base model.
- **Limitation:** Suited for short-horizon/session continuity rather than durable knowledge; latent representations are less interpretable and harder to precisely edit.



Take-aways

- **Full Model Parameter Store**

- Complete model weights saved as a full checkpoint (e.g., model.safetensors, pytorch_model.bin)
- Suited for permanent knowledge updates that become part of the base model

- **Adapter / LoRA Store**

- Lightweight adapter parameters attached to a frozen base model for incremental updates
- Ideal for modular domain-, user-, or task-specific knowledge with easy swapping and rollback

- **KV-Cache Store**

- Persistent attention key-value tensors across sessions instead of temporary decoding cache
- Suited for resuming a prior dialogue or task trajectory with high fidelity—when the agent must continue “from where it left off” rather than re-encode history from text

- **Latent Tensor Store**

- Compact latent representations, such as memory tokens, summary vectors, or steering vectors
- Ideal for compact long-term memory without storing full conversations or exact token states

Index Layer

An index layer defines how stored memory objects are located.

Storage level	Data type	Storage model	Representative index	Mechanism	Applications	Example systems
Text-Level Storage	Structured	Table Store	B+ Tree / Hash / Bitmap / LSM-tree	Ordered lookup / hashing / filtering / log-structured merge	Exact lookup of user attributes, filtering memories by time/user/session,	MemGPT state store
	Semi-Structured	KV / JSON Store	Path Index /GIN /Expression Index	JSON path navigation + inverted field lookup	Retrieving past conversation states, searching tool execution traces by field (time/action/tool)	LangChain Memory, MemGPT episodic store
		Hierarchical Store (XML / Tree)	Label Index (Dewey / ORIPATH) / Trie	Hierarchical traversal + prefix search	Retrieving full reasoning chains, searching task decomposition paths	Voyager, AutoGPT task memory
		Graph Store	Adjacency List / CSR /2-Hop Labeling	Graph traversal / reachability	Multi-hop memory search, retrieving related facts via graph expansion	GraphRAG, HippoRAG
	Unstructured	Text Store	Inverted Index	term → memory chunk retrieval	Keyword-based memory search, retrieving past chats by phrase	LangChain retriever, LlamaIndex BM25 memory
		Vector Store (Text/Image/Audio/Video)	HNSW / IVF /IVFPQ/DiskANN	ANN similarity search	Semantic memory retrieval, multimodal memory search	MemGPT vector memory
Model-Level Storage	Parameter State	Model Checkpoint Store	Model Registry Index	Model routing / selection	Selecting different model checkpoints for different tasks, switching memory-conditioned models	MemOS, ROME, MEMIT
		Parameter Module Store	Module Index	Adapter selection / composition	Retrieving task-specific LoRA/adapters, selecting personalized parameter modules	MemOS, MemLoRA, CoLoR
	Latent State	KV Cache Store	Cache Block / Token Index	Attention reuse across decoding	Retrieving previous context tokens for continued generation, session continuity search	MemOS, LONGMEM
		Latent Tensor Store	Tensor Segment Index	Latent state reuse / segmentation	Long-context reasoning continuation, cross-turn state persistence	AutoCompressor; MemoryLLM

Storage Design Trade-offs

Storage level	Storage class	Subclass	Memory representation	Physical format	Strength	Limitation
Text-Level Storage	Unstructured	Runtime Context Buffer	prompt tokens; dialogue snippets; working summaries	active context window; RAM-like prompt buffer	Fastest access to working memory; directly available to the model without retrieval latency.	Short-lived and capacity-bounded; easily overwritten and weak for durable provenance.
		File / Object Store	raw traces, logs, docs, transcripts, summaries, multimodal objects	Markdown, JSONL, HTML, transcripts, blob/object files	Preserves raw evidence and source trace; easy to ingest, inspect, export, and reprocess later.	Weak precise CRUD and relation modeling; growth/noise require retention, deduplication, and compaction.
		Vector Store	embedding-backed memory entries and payload references	vector DB entries; embedding index + external payloads	Supports semantic recall over unstructured or weakly structured memory with flexible matching.	Embedding drift, stale payloads, duplicate vectors, and index refresh can affect recall quality.
	Structured	Key-Value / Document Store	profiles, task states, memory slots, JSON-like records	KV pairs; document DB; JSON records	Explicit keys and fields enable predictable lookup, update, deletion, and governance metadata.	Schema or key design can become inconsistent; duplicate or stale records accumulate after noisy extraction.
		Relational / Table Store	repeated facts, triples, tool logs, workflow states, metadata rows	relational tables; rows/columns; SQL-managed metadata	Strong support for constraints, joins, transactions, auditing, and SQL-style memory management.	Rigid schemas and table design may under-model rich relations or evolving long-horizon context.
	Semi-Structured	Graph Store	entities, events, facts, relations, temporal links, provenance	property graph; temporal graph; nodes/edges	Captures multi-hop relations, lineage, temporal validity, and relation-aware memory access.	Entity resolution and edge maintenance are costly; wrong links can propagate through neighborhoods.
		Hierarchical Store	chunks, summaries, tiers, archive levels, parent-child memories	tree/DAG nodes; summary layers; hierarchy links	Provides multi-resolution access for long histories and documents while retaining source lineage.	Summary drift and stale parent nodes require partial rebuilds and careful refresh policies.
Model-Level Storage	Parameter	Model Checkpoint Store	knowledge, skills, experience internalized in full model weights	model checkpoint files; fine-tuned, edited, or merged weights	Low-latency reuse of stable knowledge/capabilities after loading; no explicit context injection needed.	Opaque and difficult to audit, delete, or rollback; edits can interfere with unrelated behavior.
		Parameter Module Store	adapters, LoRA modules, patch modules, auxiliary models, parameter banks	adapter/LoRA/module files; parameter bank entries	More modular than full checkpoints; supports task/domain/user routing and versioned parameter memories.	Requires compatibility metadata and routing; module conflicts, composition cost, and governance remain hard.
	Latent State	KV-Cache Store	attention key-value tensors by layer/head/token position	serialized KV-cache blocks; cache directory	Reuses context state efficiently for continuation and long-context acceleration without replaying text.	Not human-readable; eviction, compatibility, and correctness are difficult to verify.
		Latent Tensor Store	hidden states, activations, latent tokens, steering vectors, compressed states	tensor store; latent-state segments; memory vectors	Preserves reusable internal states beyond text; useful for state reuse, steering, or compressed memory.	Weak semantic editability and auditability; latent meaning, portability, and safety are hard to inspect.

Existing Memory Systems

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

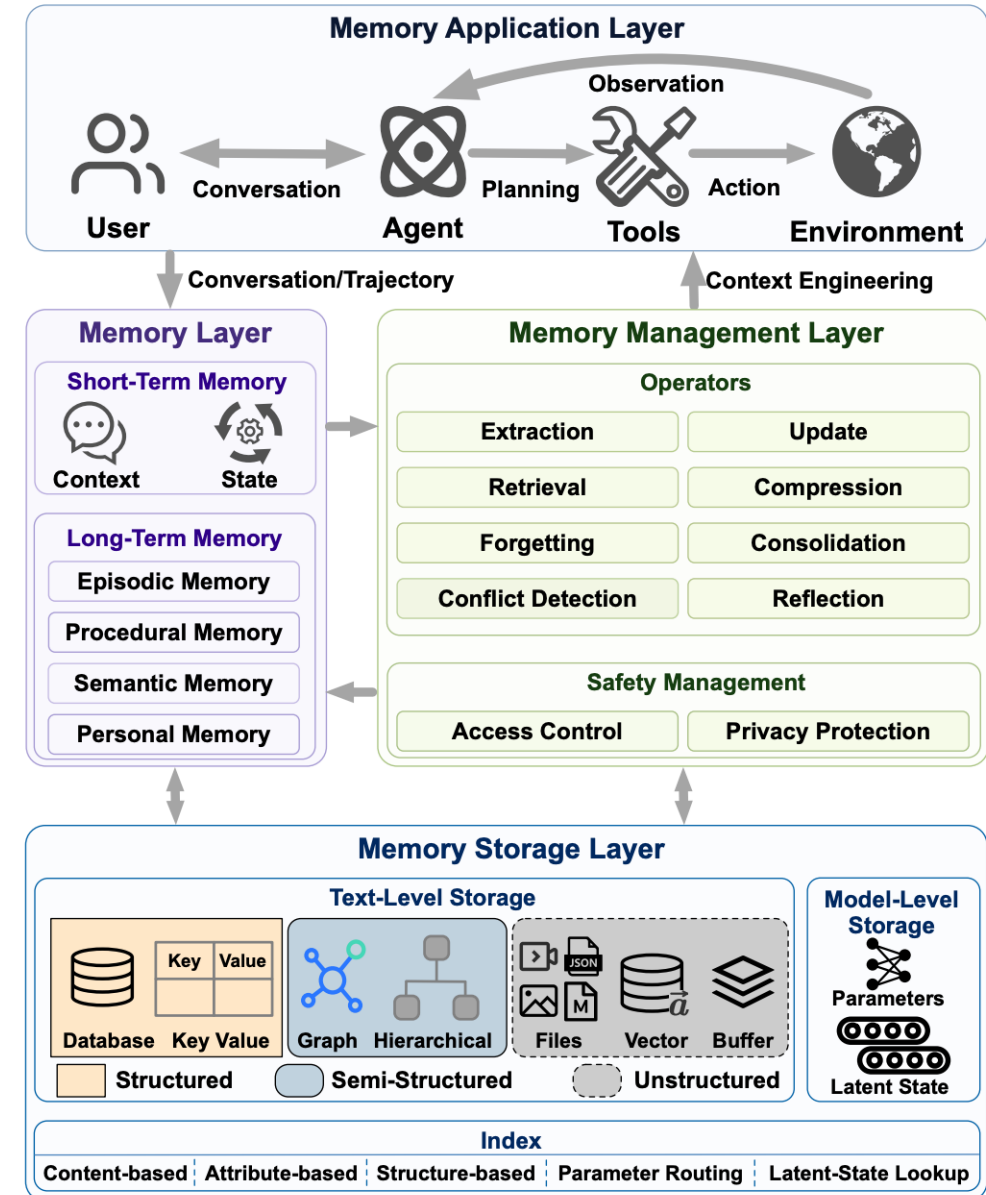
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

❑ Evaluation

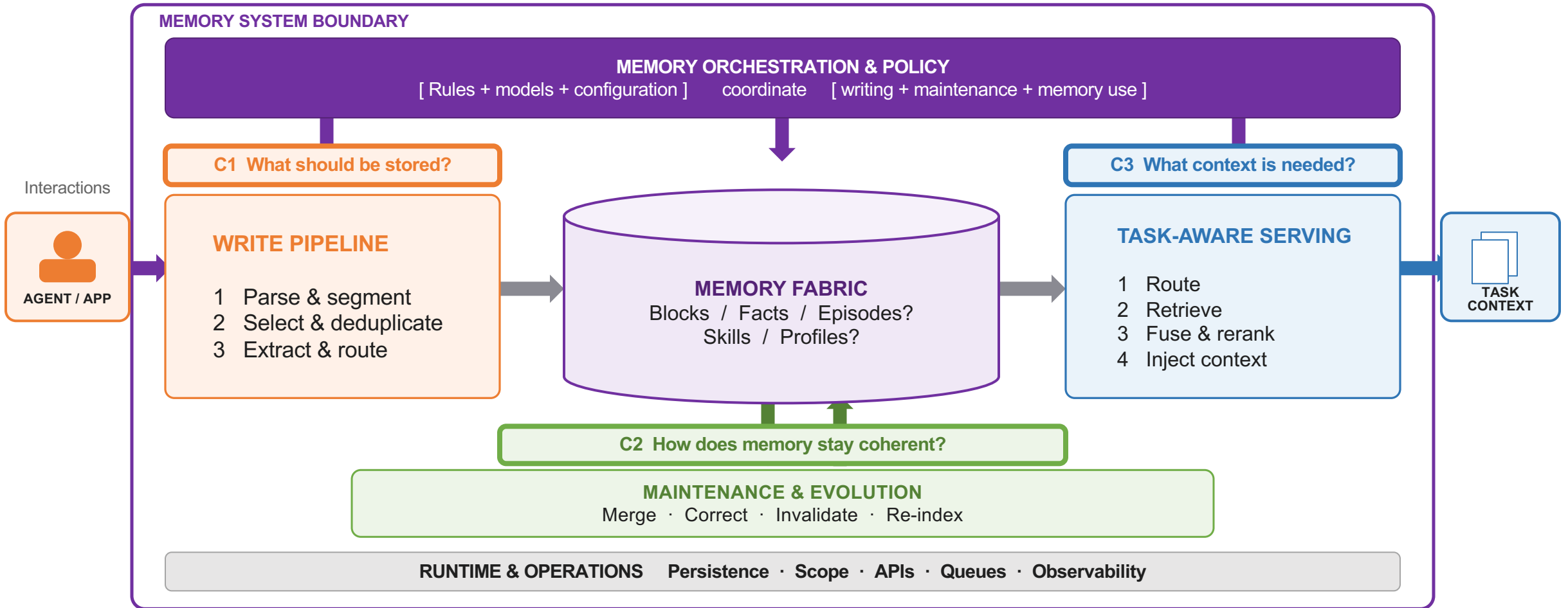
❑ Open Problems



Existing Systems

Storage Class	Specific Class	System	Memory formation and writing	Memory update and conflict handling	Retrieval, routing, and injection
Unstructured Storage	File / Object Store	MemGPT	Block-level memory units	Cache, paging & eviction	LLM-planned query actions
	Vector Store	Mem0	Extracted atomic facts	Conflict & version repair	Vector/keyword similarity search
Structured Storage	Relational / Table Store	ChatDB	Extracted atomic facts	Conflict & version repair	LLM-planned query actions
	Key-Value / Document Store	MemoChat	Extracted atomic facts	Append-only/static index	LLM-planned query actions
Semi-Structured Storage	Graph Store	RAPTOR	Hierarchical heterogeneous memory	Append-only/static index	Vector/keyword similarity search + Graph/tree traversal
	Tree Store	GraphRAG	Hierarchical heterogeneous memory	Append-only/static index	Graph/tree traversal
Hybrid	Key-Value / Document Store; Graph Store; Tree Store	MemOS	Hierarchical heterogeneous memory	Cache, paging & eviction + Entity/topology consolidation + Conflict & version repair	Vector/keyword similarity search + Graph/tree traversal + LLM-planned query actions

Existing Systems



What Counts as a System-Level Memory System?

G1 Memory Has a Clear Home

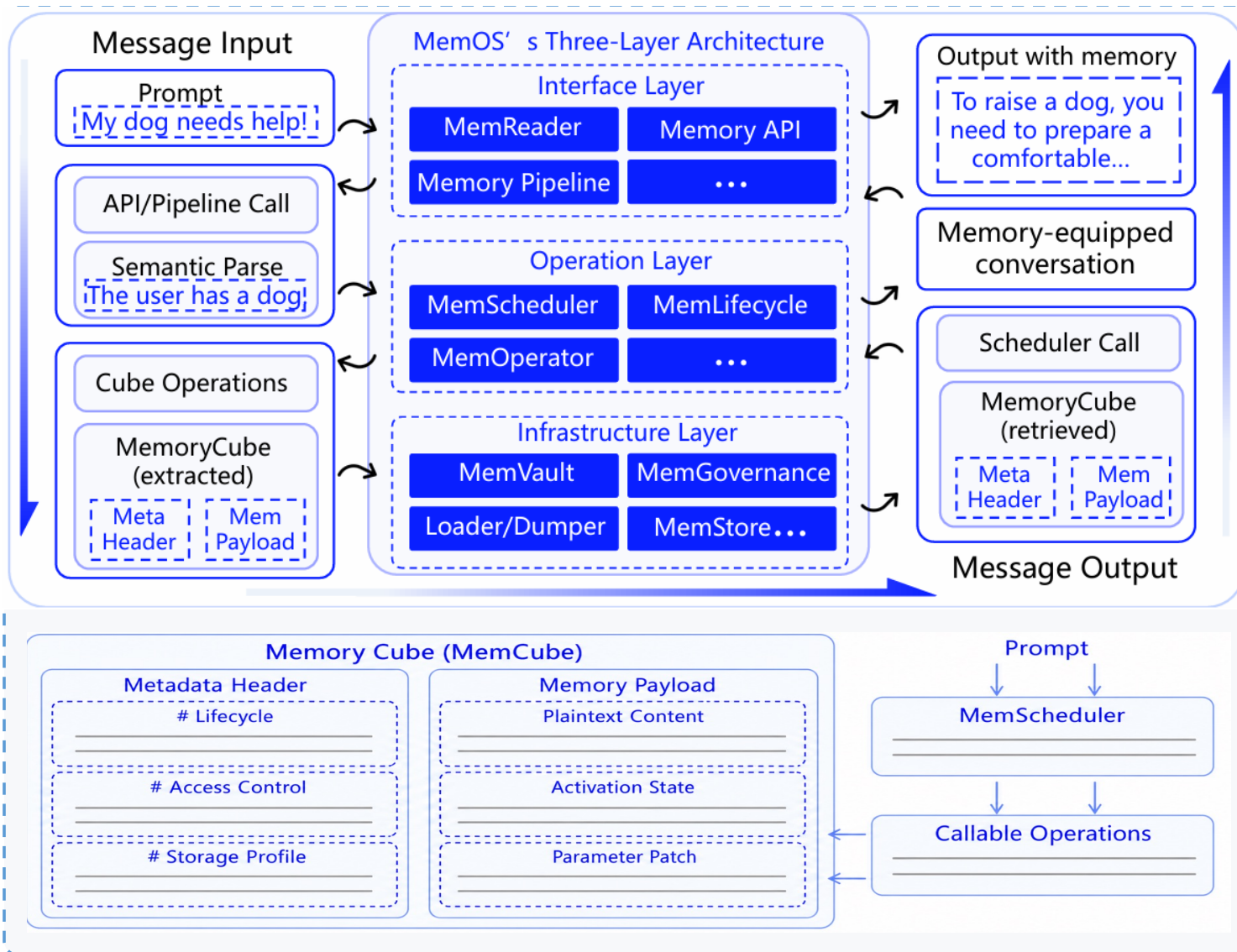
G2 Memory Runs as a Lifecycle

G3 Works Beyond the Demo

G4 Can Be Used Today

MemOS

Memory as a Governed, Schedulable System Resource



- MemCube is the resource unit; scheduling, lifecycle and governance form the control plane.
- MemOS coordinates representation, scheduling, lifecycle, governance and storage through one control plane.

C1 · WRITE

MemReader + Memory API

Parse intent into MemoryCall; package content, provenance and access metadata into a MemCube.

C2 · MAINTAIN

MemScheduler + MemLifecycle + MemGovernance

Choose memory type and tier; track Generated → Activated → Merged → Archived / Expired; enforce access and version policies.

C3 · SERVE

MemOperator + Search / Injection

Route across plaintext, activation and parametric memory; combine candidates and inject task-specific context.

IMPLEMENTATION BOUNDARY

Text/KV paths and the scheduler are operational; parametric memory and full governance remain partial.

Six Memory Types Become One Coordinated Service

MIRIX separates representation, control and execution, then exposes them through one deployable memory service.

DEPLOYABLE MEMORY SERVICE

Core
Memory



Episodic
Memory



Semantic
Memory



Procedural
Memory



Resource
Memory



Knowledge
Vault

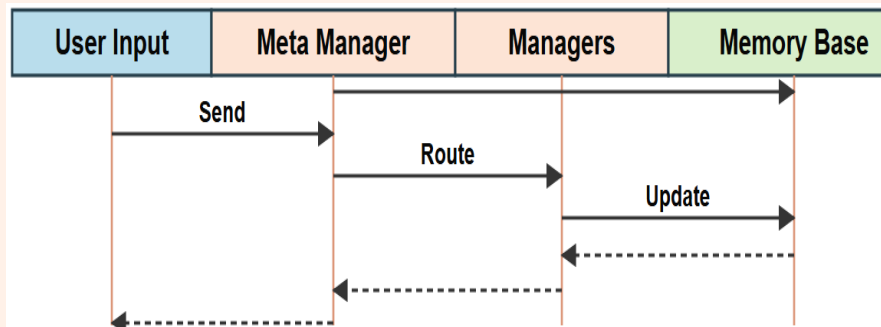


CONTROL PLANE | META MEMORY MANAGER

Selects relevant memory types->dispatches specialized Memory Managers->coordinates parallel work.

C1 | WRITE

Existing-memory check -> classify ->parallel typed update

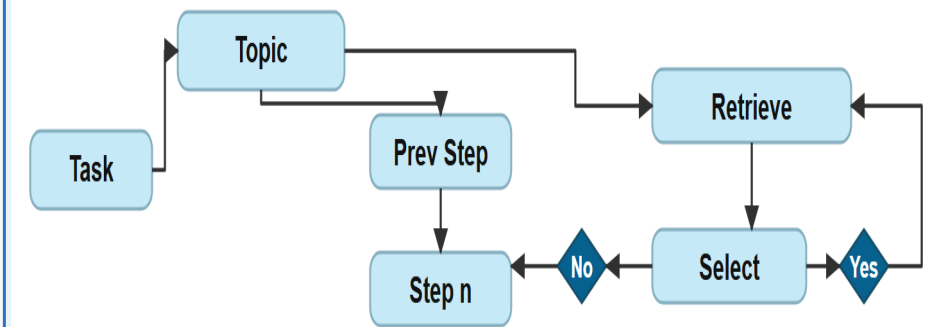


C2 | MAINTAIN | INTERNAL EVOLUTION

- Typed deduplication -> Core rewrite -> auto_dream merge / repair
- The service maintains memory after writing.

C3 | SERVE

Topic -> six-memory search -> deeper tool -> tagged prompt injection



Cognee

Knowledge Engine, System Operations

The architecture combines relational, vector and graph stores.

C1 · WRITE

Tasks + Pipelines

Validate inputs, chunk data and extract entities and relations before indexing.

C2 · MAINTAIN

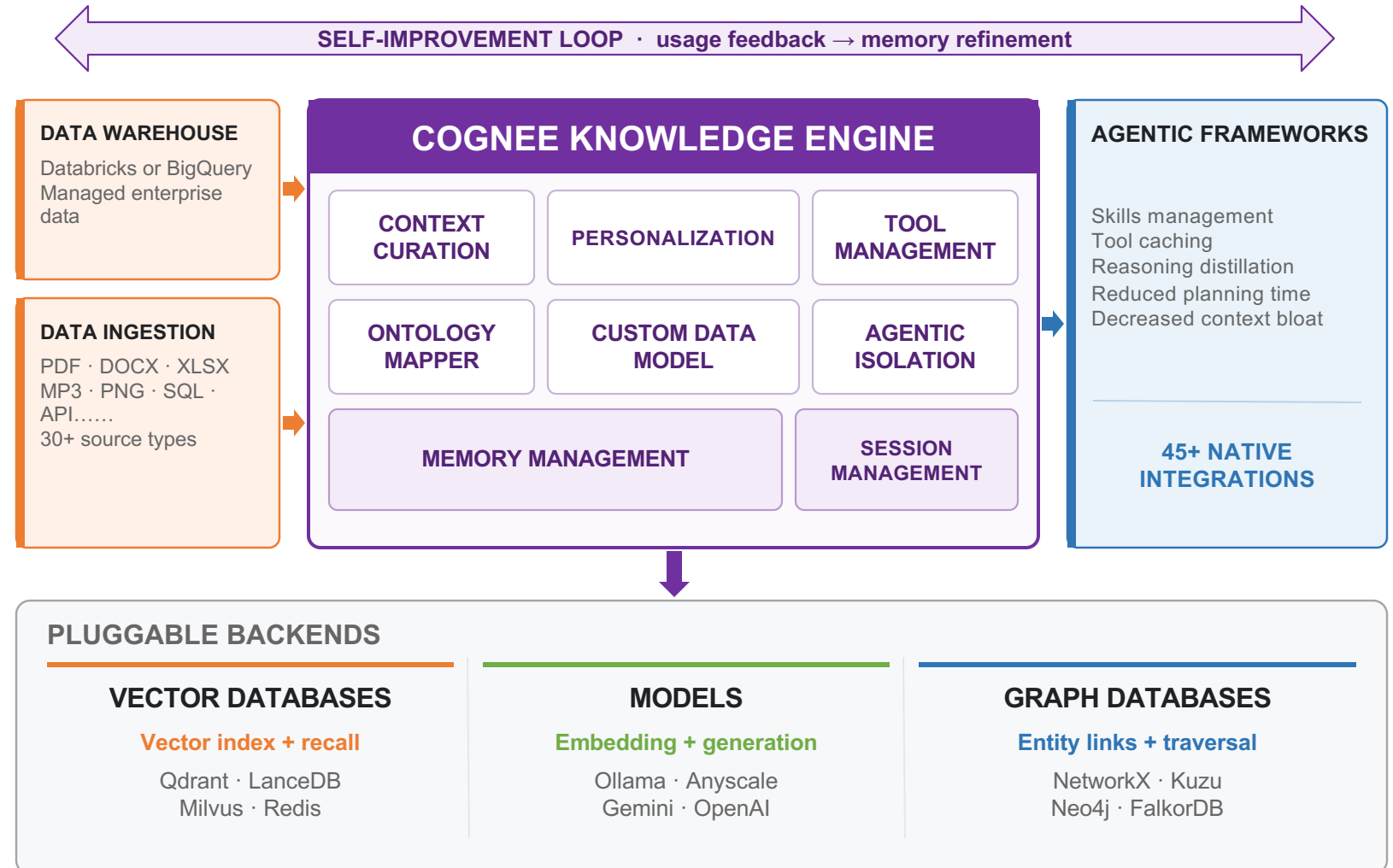
Datasets + Run Control

Ownership, ACL, per-dataset status and recovery sit outside retrieval quality.

C3 · SERVE

Backends + Recall

Pluggable relational, vector and graph stores sit below a routed recall API.



Memory as a Maintained Lifecycle

1 WRITE create a checkable memory unit

Detect a topic boundary, rewrite the episode, then extract atomic facts and time-bounded foresight.

2 MAINTAIN turn episodes into evolving scenes

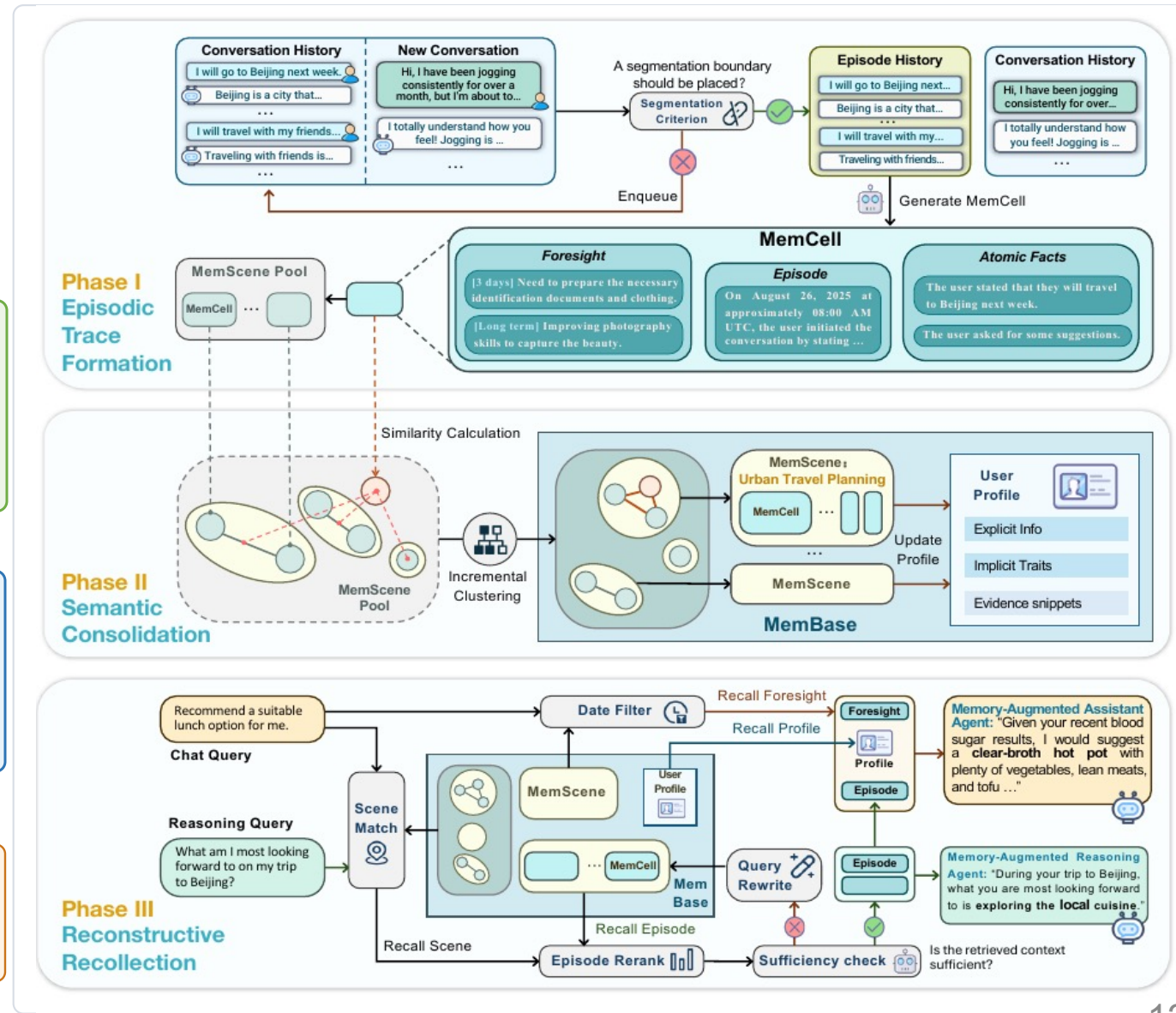
Incrementally cluster MemCells, update a MemScene and maintain a profile from recent evidence; retain expiry and conflict evidence.

3 SERVE return only sufficient evidence

Match a scene, rerank its episodes, date-filter foresight, then perform a second query only when the evidence is insufficient.

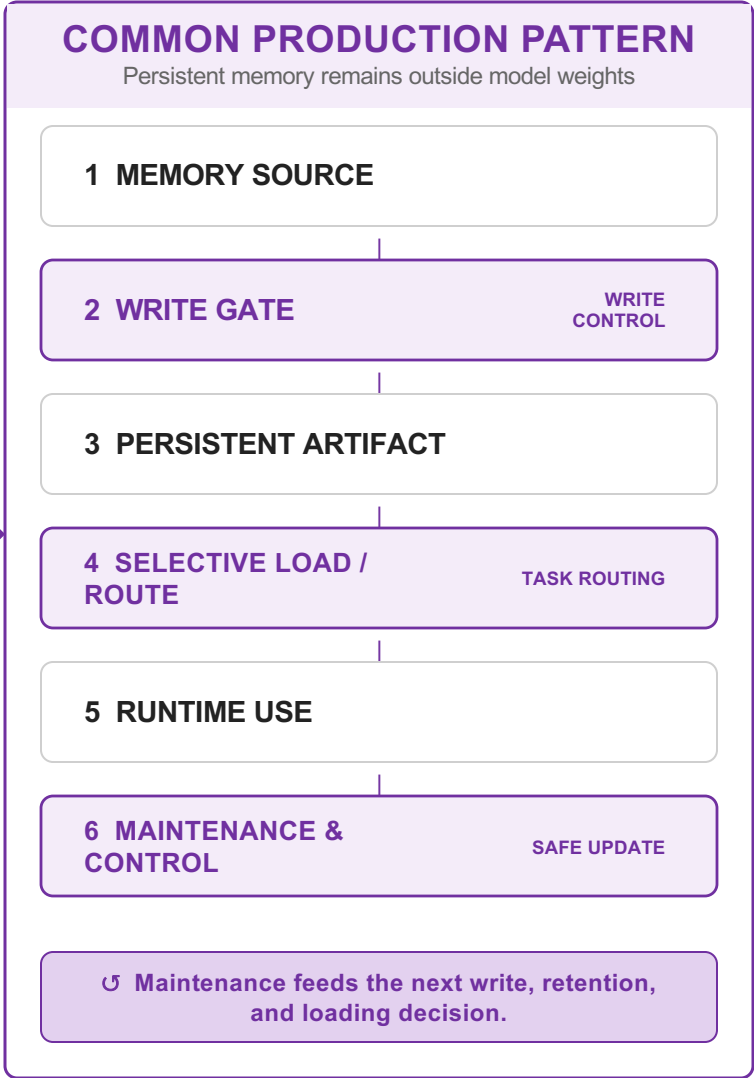
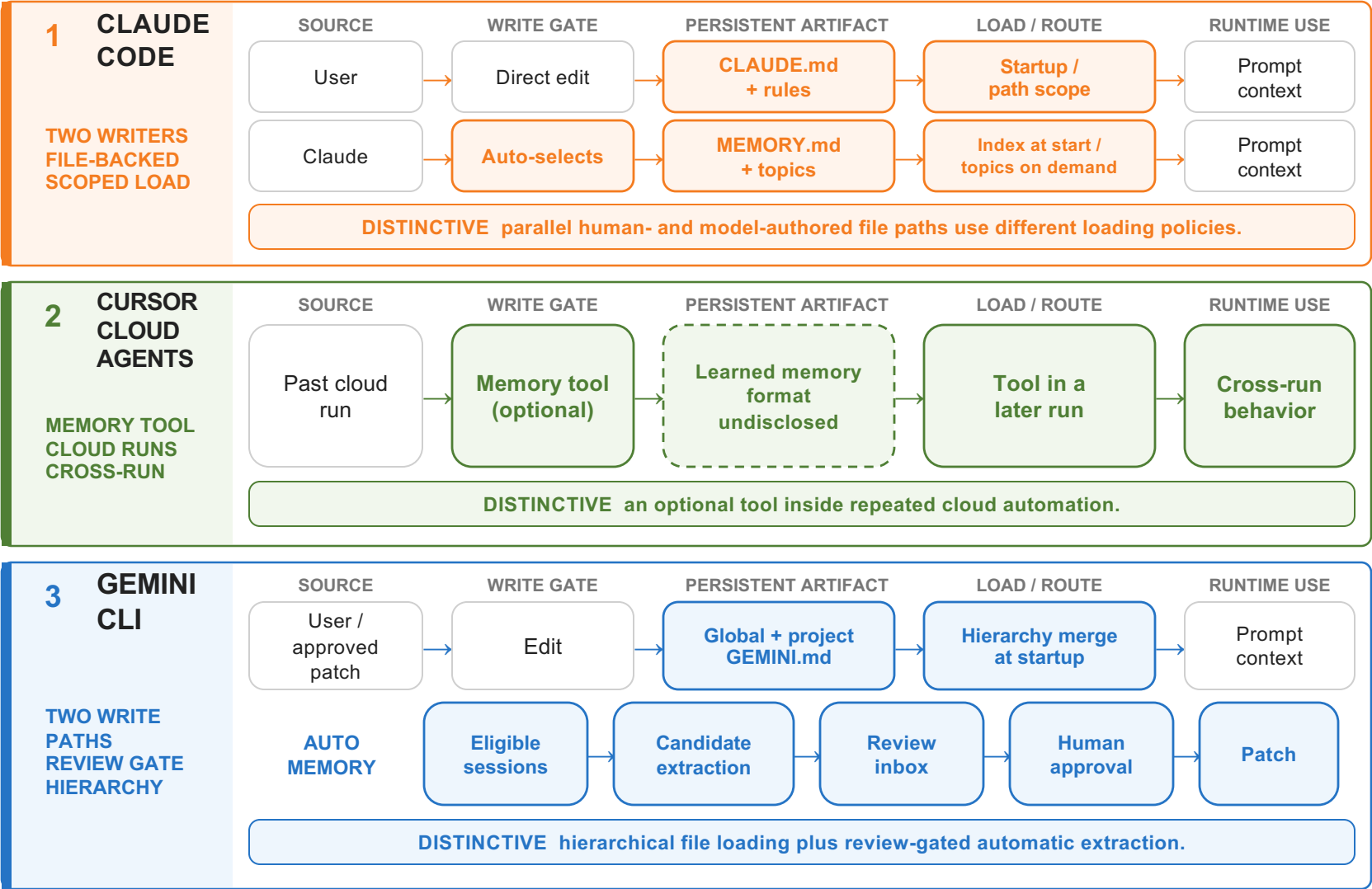
PAPER -> PRODUCT BOUNDARY

- MemCell / MemScene / Profile became episodes, facts, skills, etc.....
- The project lineage is continuous, but EverOS 1.x is a rebuilt architecture.



Existing Products

Three Products Instantiate One Memory-System Pattern



Evaluation

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

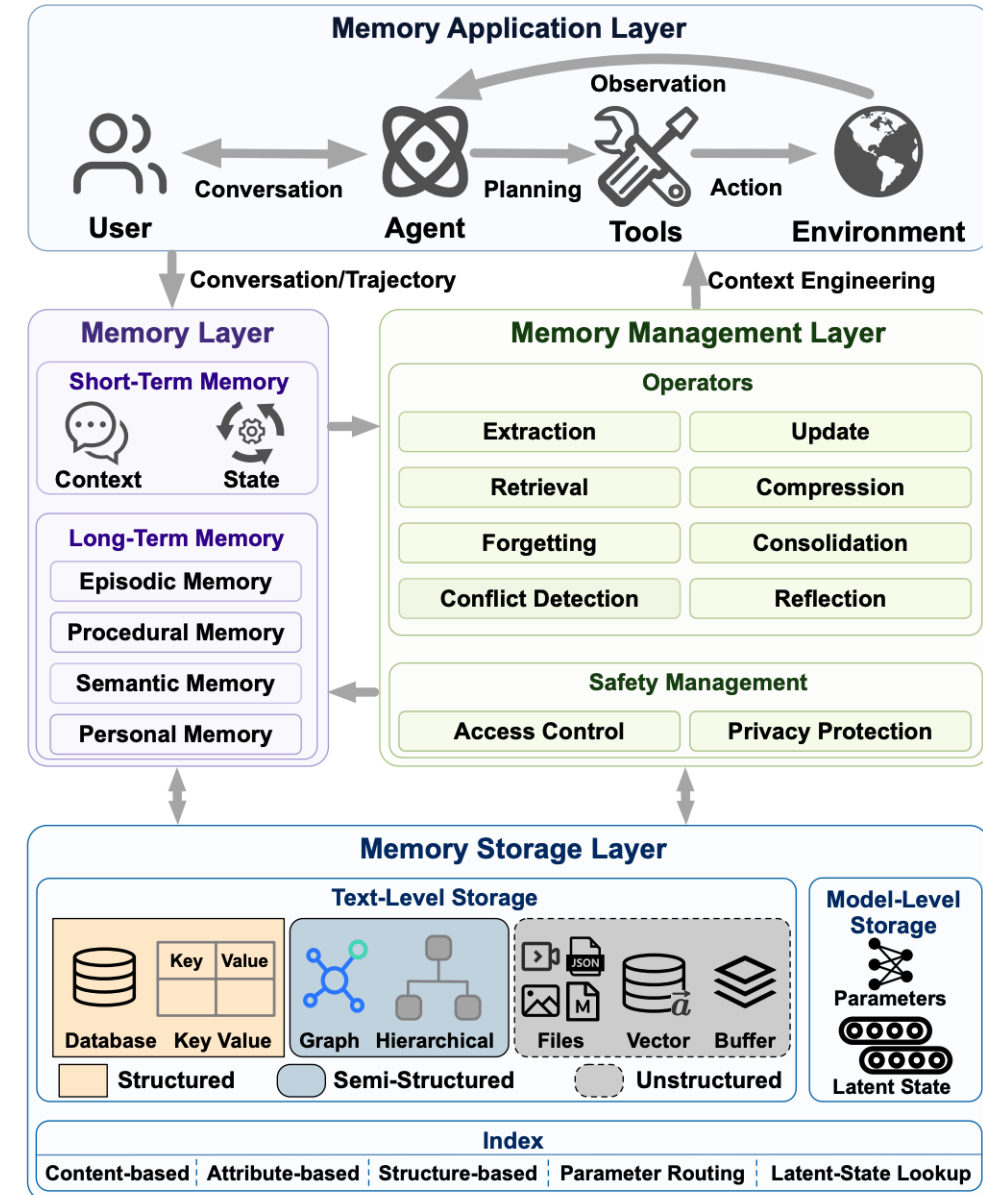
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

❑ Evaluation

❑ Open Problems



Why Memory Evaluation Is Not Just RAG Evaluation

🔄 Memory evaluation shifts from retrieving evidence to sustaining a reusable, stateful lifecycle.

RAG vs. Memory Systems: essential difference

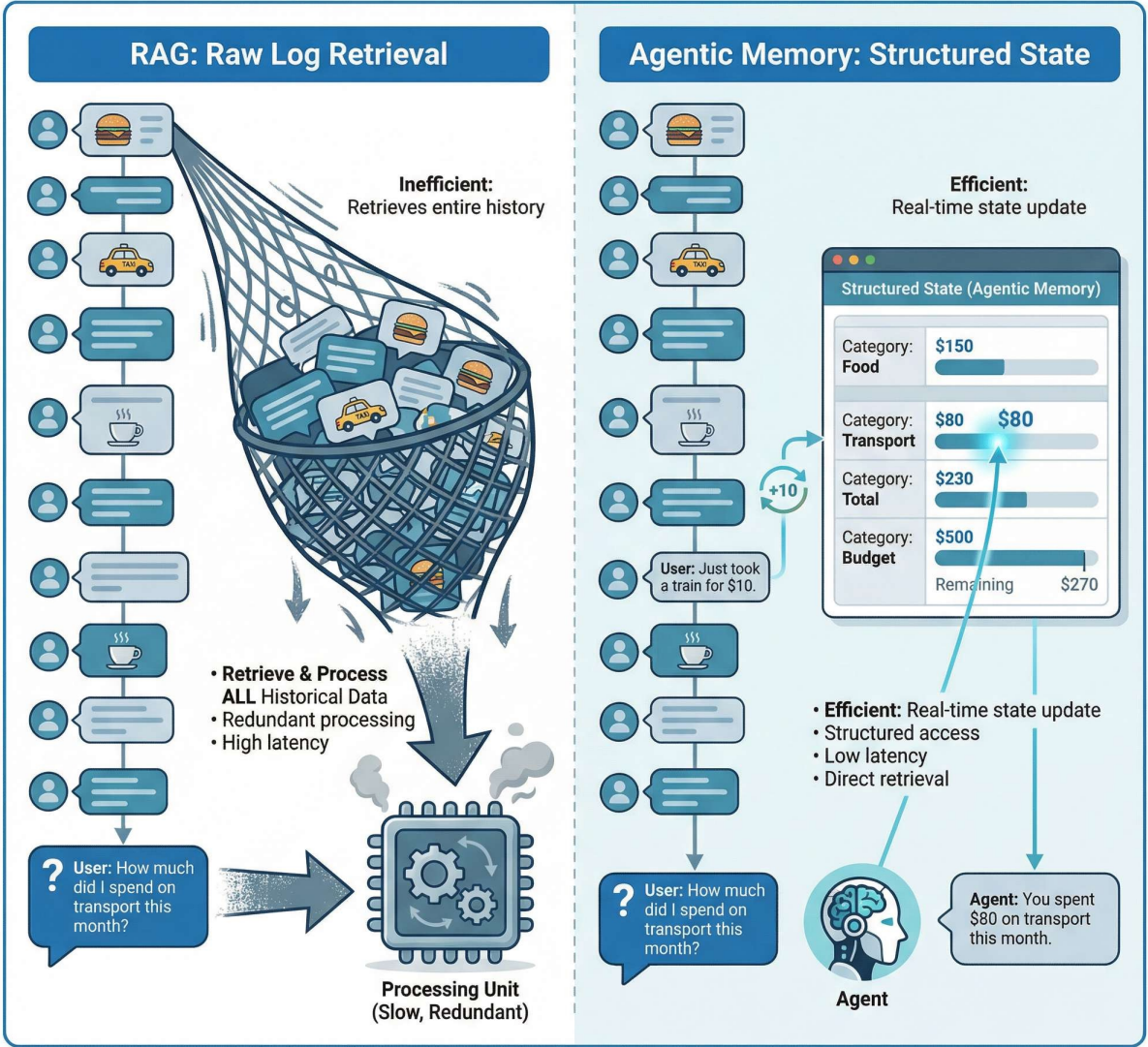
- **RAG** answers a query by retrieving evidence from a fixed external corpus at read time.
- **A memory system** maintains an **evolving** internal state across interactions, so it must **write, update, retrieve**, and sometimes forget information before taking action.

Aspects	RAG	Memory system
 Knowledge form	Static corpus	Persistent, evolving state
 Core operation	Retrieve → read → answer	Write / update / retrieve / act
 Across sessions	Usually none	Accumulates over time
 What evaluation asks	Did retrieval support a faithful answer?	Is the maintained state valid, current, and useful?
 Typical scores	Precision, recall, faithfulness	Task success, evidence quality, update robustness, cost



Key distinction:

RAG evaluates answer quality from retrieved evidence, while memory evaluation asks whether information remains **reusable, correct, explainable, and actionable** over time.



Illustrative example: raw retrieval vs. structured memory state.

What should be New in Memory Benchmarks? Data, Framework, and Metrics

 New memory benchmarks move from static QA pairs to supervised memory lifecycles.

1) Data Design



Multi-session histories

Long-term, multi-turn interactions
(*LongMemEval*, *LoCoMo*)



Incremental streams

Continuous updates over time
(*MemAgentBench*)



Controlled updates + conflicts

Updates, contradictions, conditions
(*MemConflict*, *GateMem*)



Enrichment trajectories

Multi-modal and structured knowledge
(*MemVerge*, *VideoMemBench*, *MemOS*)

 Text  Table/DB  Code/Log  Image/Video/Audio

2) Protocol Design



Streaming write-before-query

History is accumulated before evaluation



Gold evidence and state traces

What should be same, current, or forgotten



Separate module checks

Extraction • Storage • Retrieval • Maintenance



Adversarial / randomized injections

Stress robustness to noise and distraction



Action-environment loops

Memory must support future decisions

 Write  Maintain  Retrieve  Use  Update  Abstain

3) Metric Design



Task performance

Answer quality

Accuracy • EM • F1 • Task success rate



Retrieval fidelity

Evidence quality

Recall@K • EMatch • Support Rank Score



Update validity

State correctness

Latest-valid accuracy • Conflict-aware score • Staleness penalty



Cost & diagnosis

Efficiency + root cause

Latency • Token/Cost • Module-level error attribution

 Correct  Wrong  Partial  Abstain

Result: failures can be attributed to a lifecycle stage, not only to the final answer.



Write

Information arrives



Maintain

Store & update



Retrieve

Find right evidence



Use

Answer / act



Diagnose

Identify failure cause

Feedback for improvement

Evaluating Memory Systems for LLM Agents

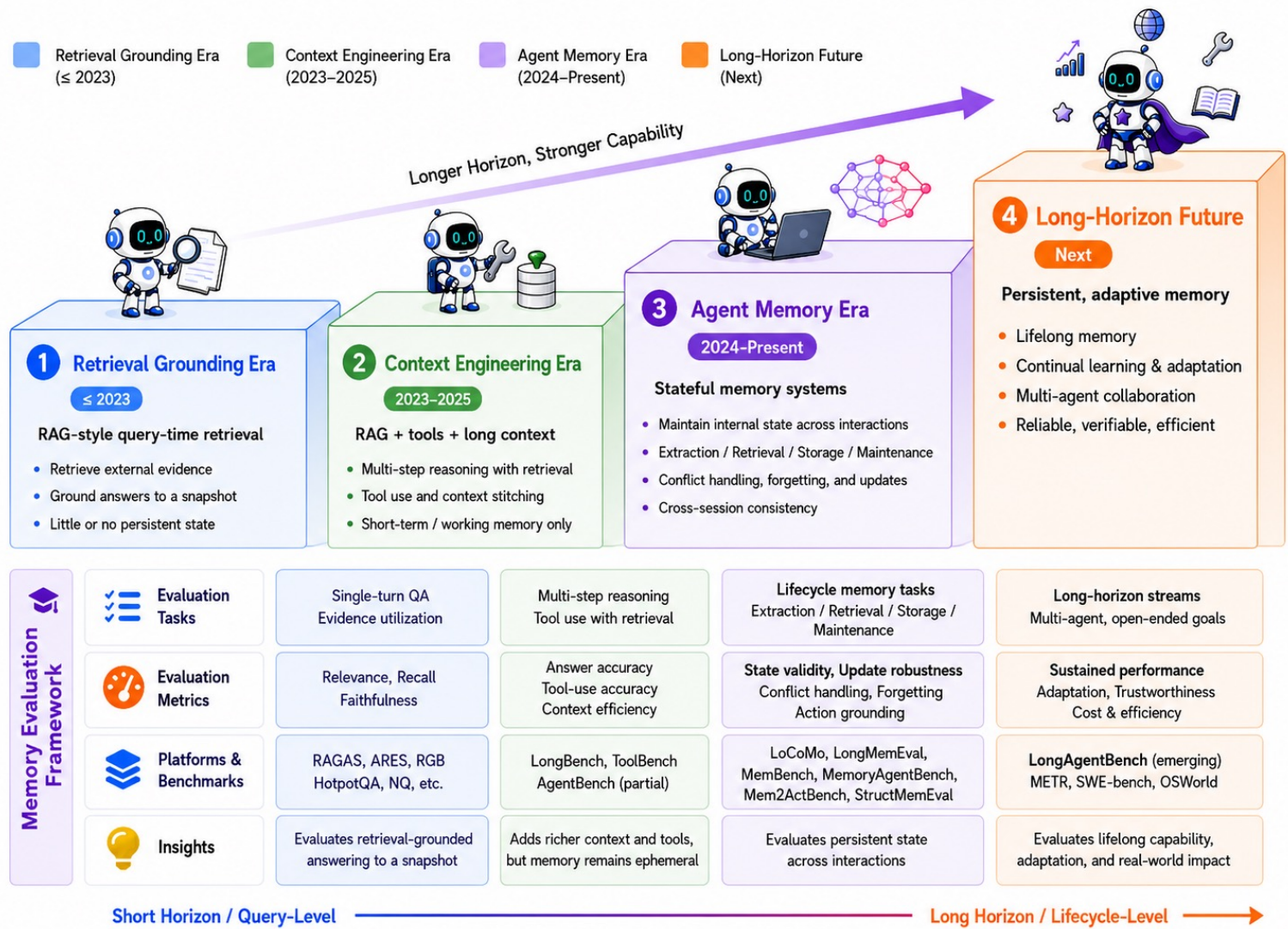
As LLM agents move from answering isolated queries to operating over long-running interactions, evaluating how they form, update, retrieve, and use memory becomes essential.

RAG vs. Memory

Evaluation Dimensions

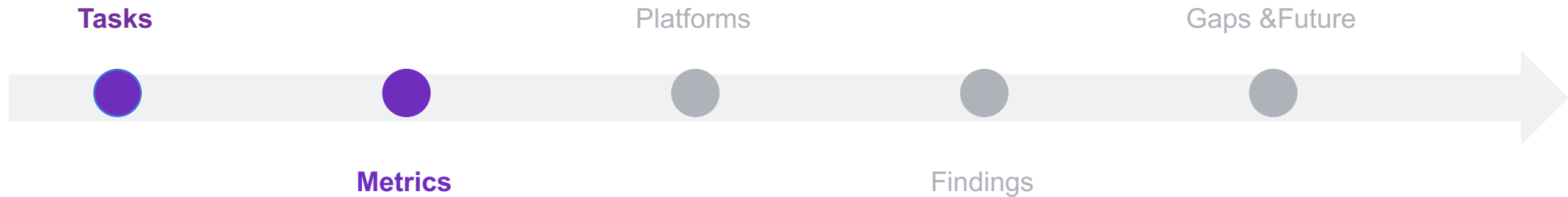
Evaluation Platforms

Findings and Problems



Question to Ask:

What Should Be Evaluated?



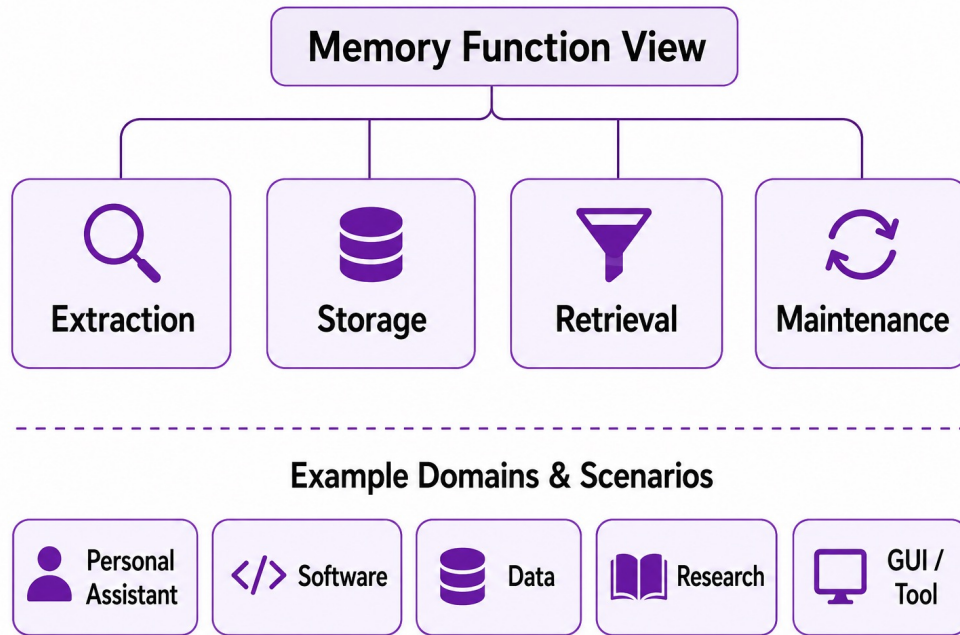
Evaluation Scope: What Should Be Evaluated?

A complete evaluation needs two complementary views:



Evaluation Tasks

What capabilities and situations should be tested?



Task–Metric Alignment



Evaluation Metrics

How should success, failure, and cost be measured?

- Task Performance
- Update Robustness
- Retrieval Quality
- Efficiency & Cost

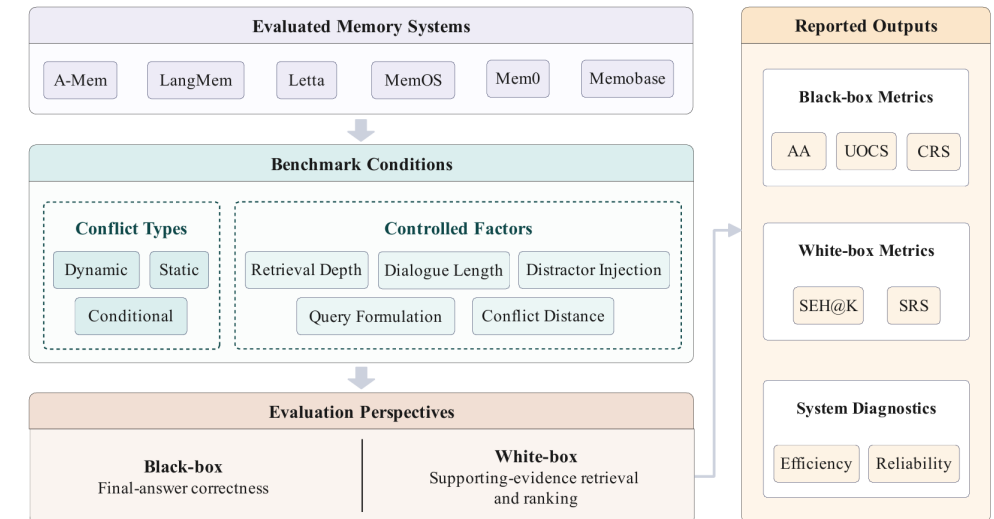
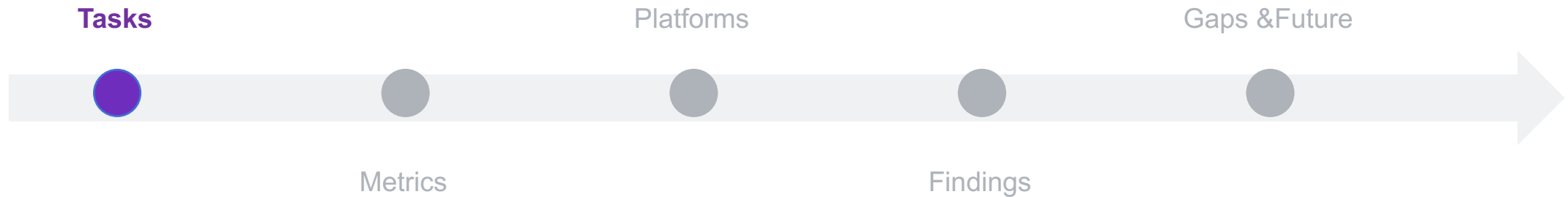


Fig. 3. Overview of evaluation dimensions and reported outputs in MemConflict.

Question to Ask:

What Tasks Should Be Evaluated?



Memory Function View: A Unified Task Taxonomy

Different memory systems share a similar set of evaluation tasks.

1. Extraction

- What should be remembered?

2. Storage

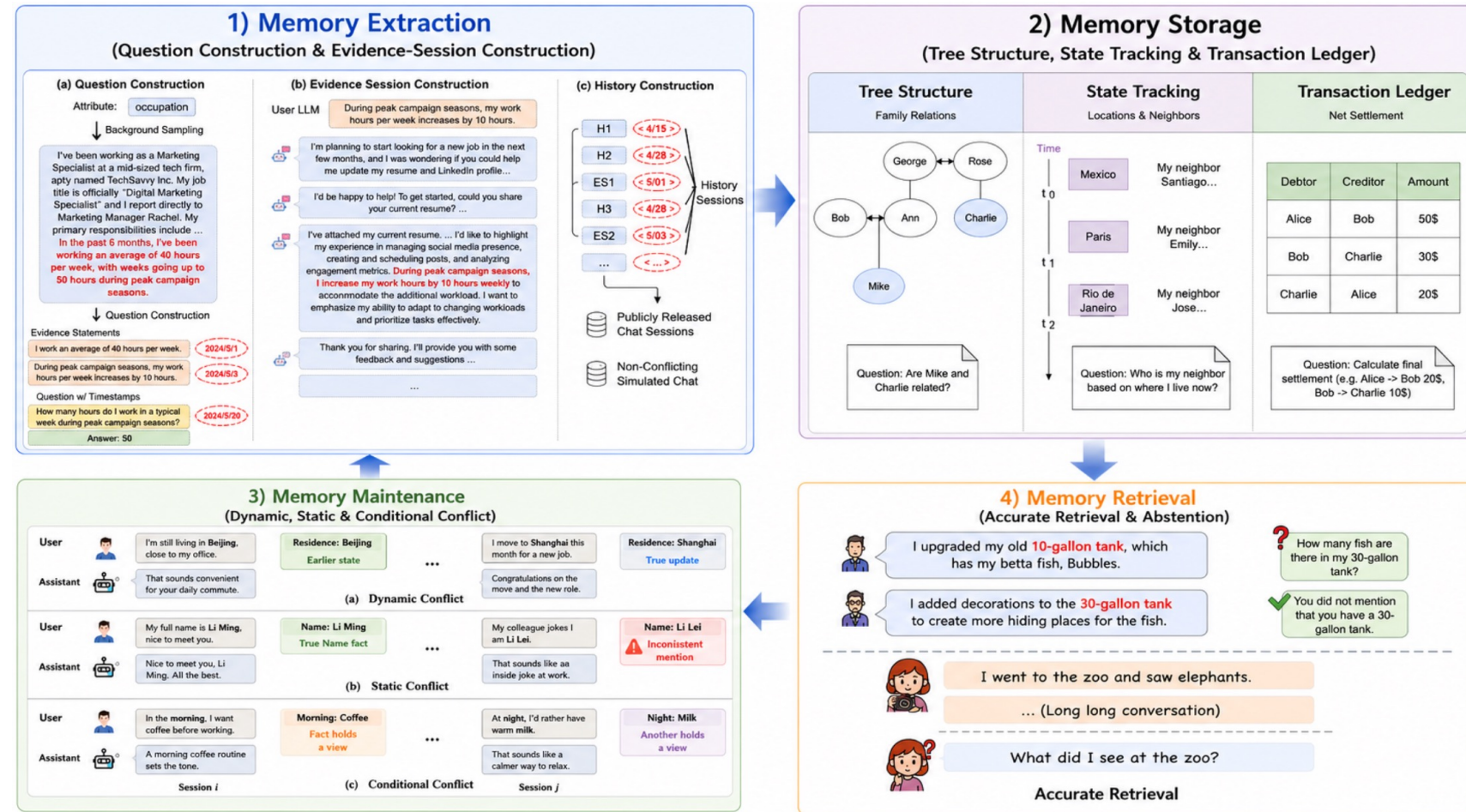
- How Should Memory Be Organized?

3. Retrieval

- Can the Right Evidence Be Found?

4. Maintenance

- Can Memory Stay Valid Over Time?



Memory Extraction: What Should Be Remembered?

➤ WHAT IS EVALUATED



Convert interactions, logs, and documents into reusable memory units while preserving key facts, event links, and sources.

➤ CORE SUBTASKS

- Long-context information extraction
- Event summarization / compression

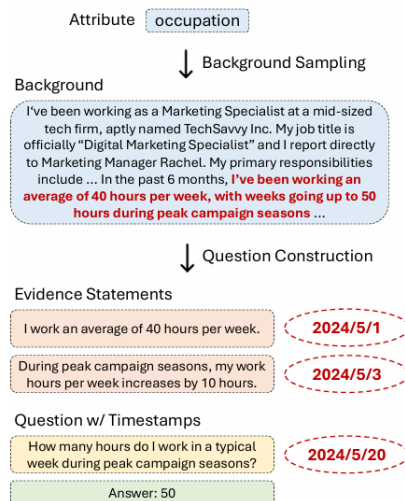
➤ AGENT EXAMPLES

- **Personal assistant:** Extract user preferences, schedules, and facts from cross-session conversations.
- **Software agent:** Distill the current task context from issues, error logs, and fix history.
- **Research agent:** Compress long-term experiment records into events, observations, and conclusions.

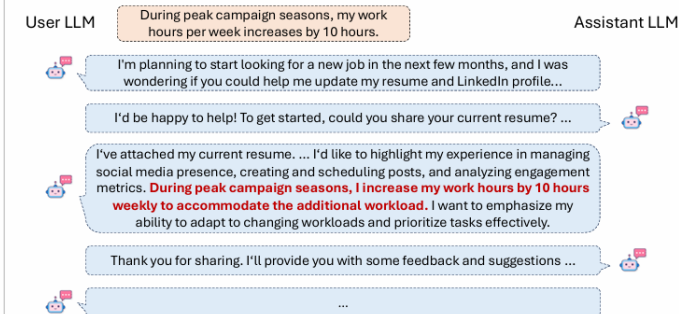
➤ REPRESENTATIVE DATASETS

- **LongMemEval:** Information extraction
- **LoCoMo:** Event summarization
- **MemBench:** Single-hop / reflective cues

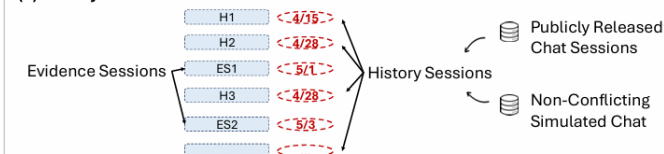
(a) Question Construction



(b) Evidence Session Construction



(c) History Construction



Memory Storage: How Should Memory Be Organized?

➤ WHAT IS EVALUATED



Structure extracted information for querying and updates, keeping relationships, states, and capacities usable over time.

➤ CORE SUBTASKS

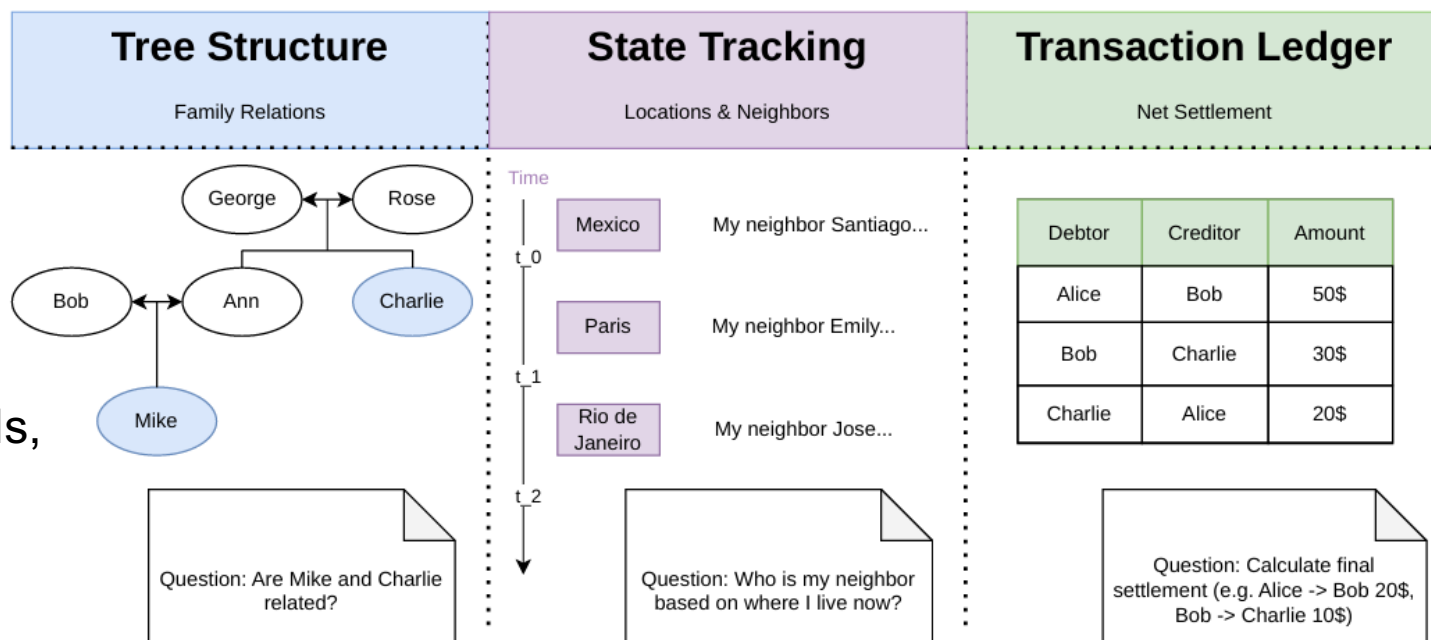
- Structural organization
- State tracking
- Capacity / stress testing

➤ REPRESENTATIVE DATASETS

- **StructMemEval:** Tree / count / recsys / state
- **MemBench:** Capacity boundary
- **MemoryAsData:** Stateful workloads

➤ AGENT EXAMPLES

- **Data agent:** Track ledgers, states, and balances.
- **Personal assistant:** Manage user profiles, preferences, and constraints.
- **Software agent:** Track repo structures, APIs, and progress.



Memory Retrieval: Can the Right Evidence Be Found?

➤ WHAT IS EVALUATED



Identify, rank, and combine relevant evidence for the query; refuse to answer when evidence is insufficient rather than fabricate.

➤ CORE SUBTASKS

- Single-hop / multi-hop retrieval
- Temporal retrieval
- Evidence ranking
- Abstention

➤ AGENT EXAMPLES

- **Research agent:** Synthesize evidence from multiple document snippets.
- **Personal assistant:** Retrieve current times, preferences, and events.
- **Tool agent:** Restore constraints and required parameters before API calls.

➤ REPRESENTATIVE DATASETS

- **MemoryAgentBench:** SH / MH document QA
- **LongMemEval:** Temporal / abstention
- **MemConflict:** White-box evidence ranking



I went to the zoo and saw elephants

... (Long long conversation)



What did I see at the zoo?

Accurate Retrieval



1/22

I went to behind-the-scenes tour of the Science Museum today **with a friend who's a chemistry professor.**



3/11

I attended a guided tour at the Natural History Museum yesterday with my dad.



4/18

I just learned a lot in a lecture at the History Museum about ancient civilizations this month.

temporal-reasoning

6/25

How many months have passed since my last museum visit with a friend?

5 months

abstention



I upgraded my old **10-gallon tank**, which has my betta fish, Bubbles.



I added decorations to the **20-gallon tank** to create more hiding places for the fish.



How many fish are there in my 30-gallon tank?



You did not mention that you have a 30-gallon tank.

Memory Maintenance: Can Memory Stay Valid Over Time?

➤ WHAT IS EVALUATED

 Keep memory current, consistent, and governable through ongoing writes, edits, conflicts, merges, and deletions

➤ CORE SUBTASKS

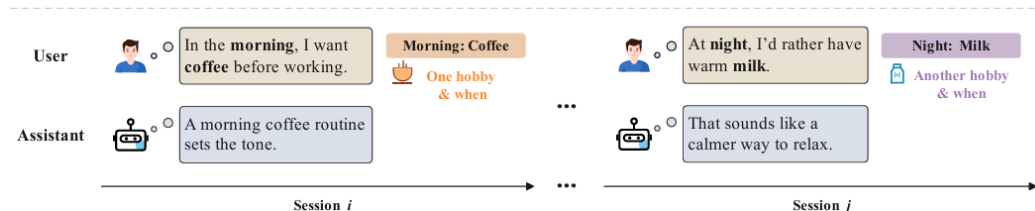
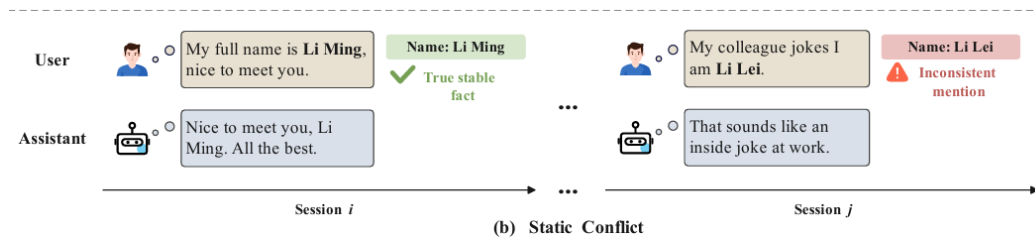
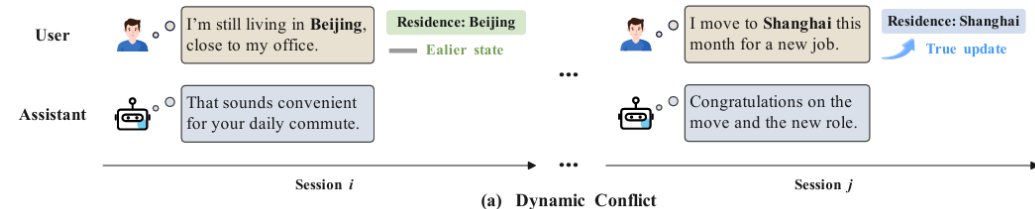
- Update
- Conflict resolution
- Forgetting / governance
- Reflection / consolidation

➤ AGENT EXAMPLES

- **Personal assistant:** Overwrite old addresses and prevent reuse of outdated facts.
- **Shared workplace agent:** Execute role permissions and explicit delete requests.
- **Software agent:** Merge new test results and discard obsolete patch data.

➤ REPRESENTATIVE DATASETS

- **LongMemEval:** Knowledge update
- **MemConflict:** Dynamic / static / conditional
- **GateMem:** Access control / forgetting



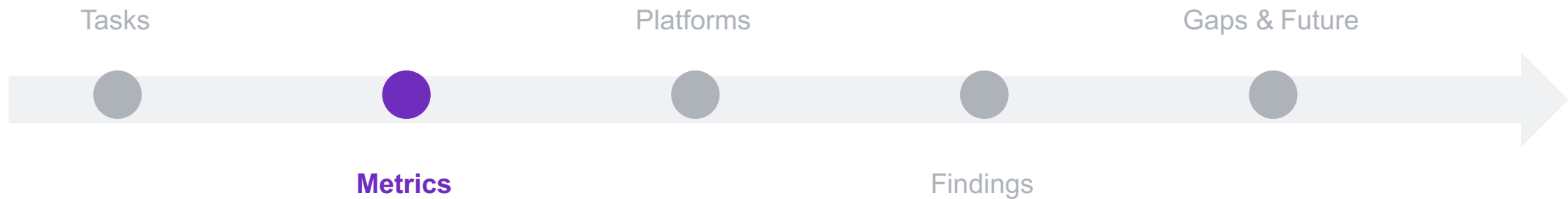
Tasks and Datasets at a Glance

Memory Function	Conversation Memory	Structured State	Incremental Interaction	Conflict / Update	Memory-to-Action
Extraction	LongMemEval LoCoMo	—	MemoryAgentBench	—	—
Storage	MemBench	StructMemEval MemoryAsData	—	—	LifelongAgentBench
Retrieval	LoCoMo LongMemEval	—	MemoryAgentBench	MemConflict	Mem2ActBench
Maintenance	LongMemEval Memora	StructMemEval	MemoryAgentBench	MemConflict GateMem	MemoryArena MemGUI-Bench

No single benchmark covers all four functions. A credible evaluation suite should combine functional diagnostics with at least one interactive, memory-to-action testbed.

Question to Ask:

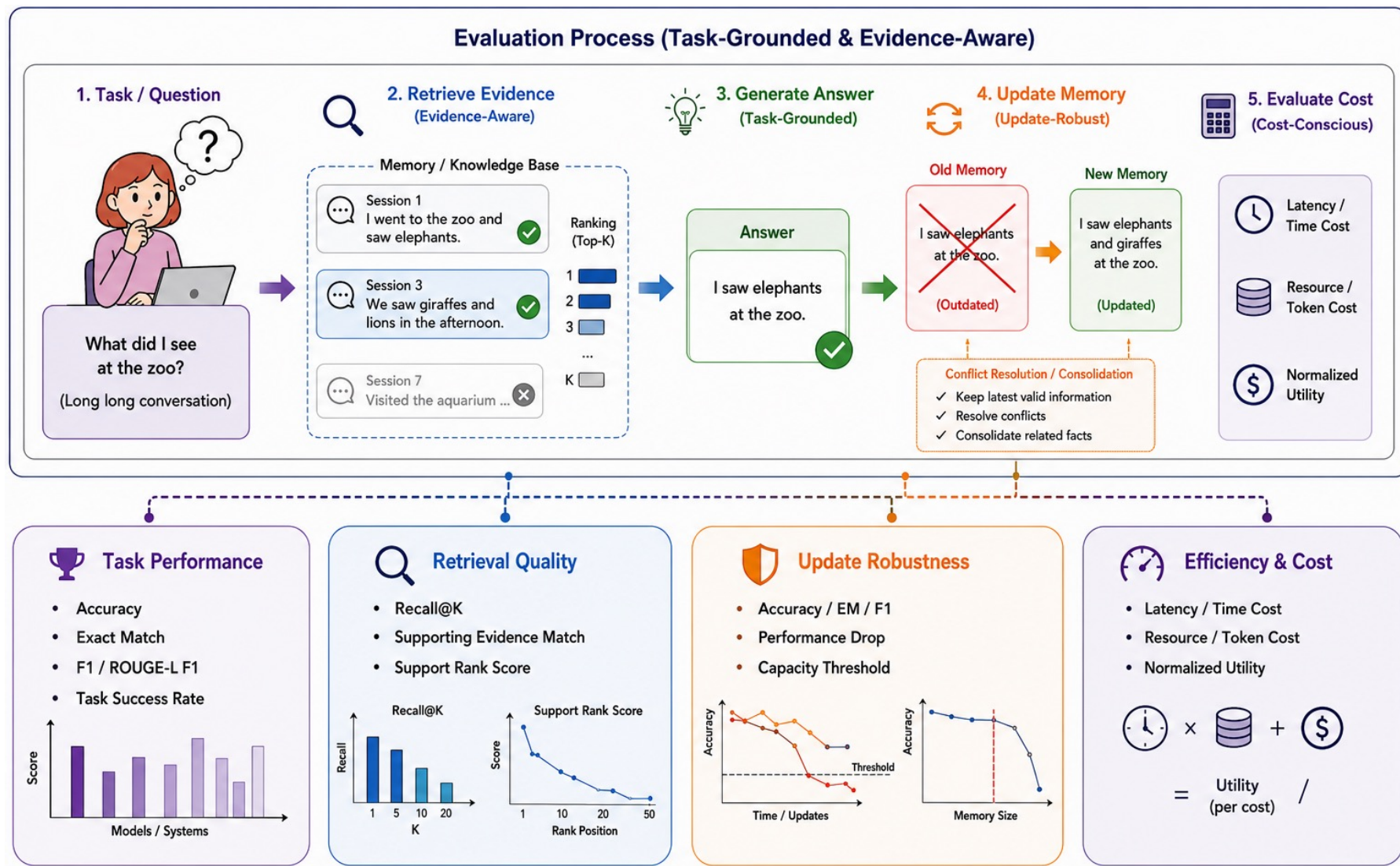
What Metrics Should Be Used?



A Four-Dimensional Concept of Memory Evaluation

➤ A good evaluation is **task-grounded**, **evidence-aware**, **update-robust**, and **cost-conscious**.

- Task Performance
- Retrieval Quality
- Update Robustness
- Efficiency & Cost

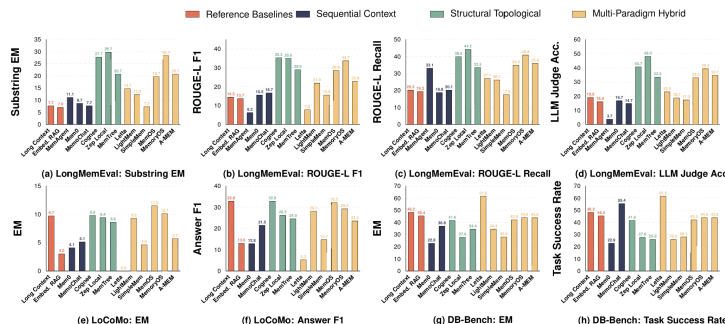


A good memory system should be **task-grounded**, **evidence-aware**, **update-robust**, and **cost-conscious**.

Memory System Evaluation Metrics

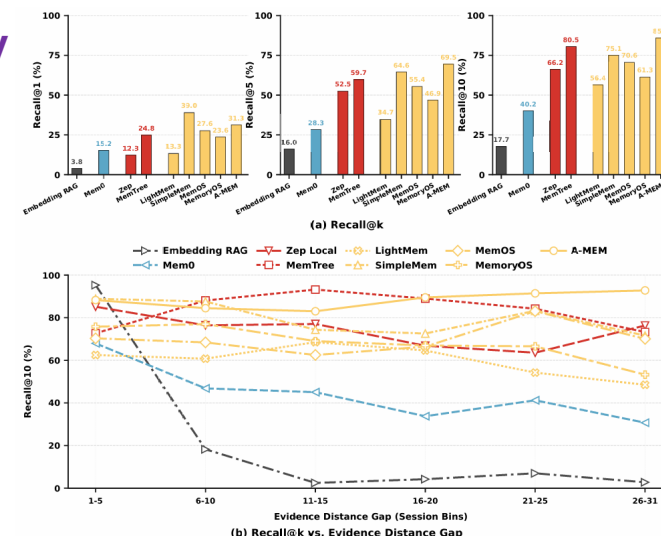
🏆 Task Performance

- Answer Accuracy
 - Exact Match
 - F1 / ROUGE-L F1
- Task Success Rate



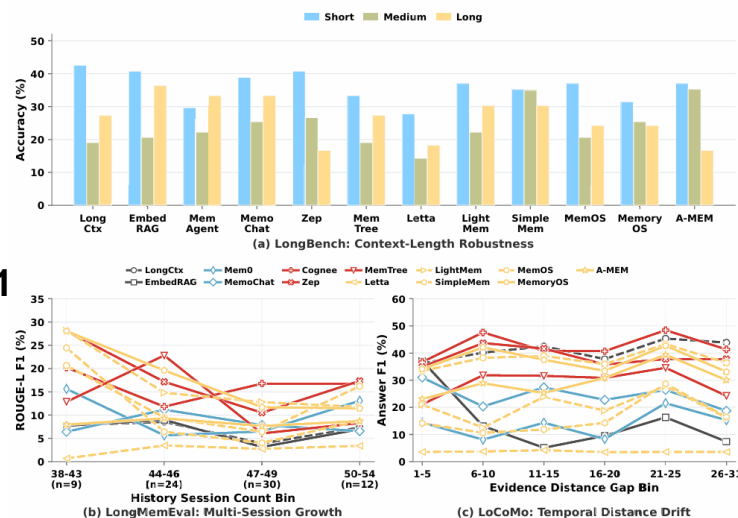
🔍 Retrieval Quality

- Recall@K
- Supporting Evidence Match
- Support Rank Score



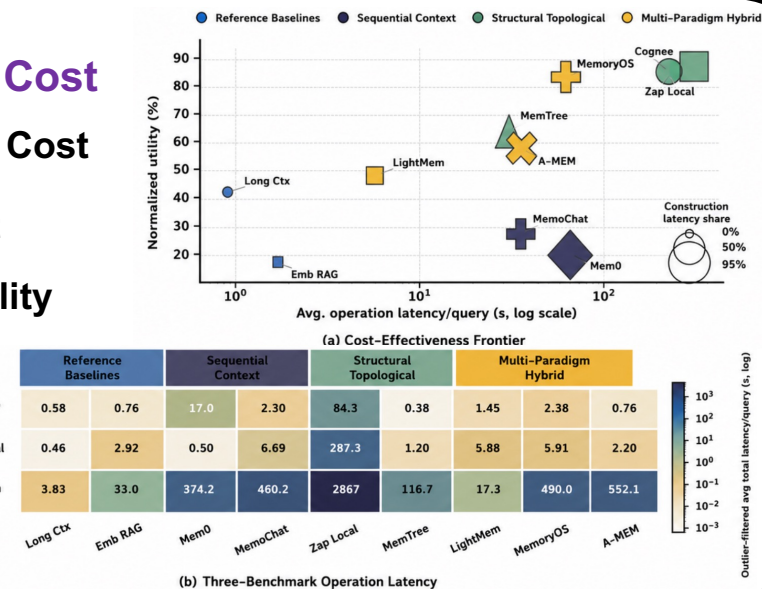
🛡️ Update Robustness

- Update Accuracy
 - Exact Match
 - F1 / ROUGE-L F1
- Performance Drop
- Capacity Threshold



🧠 Efficiency & Cost

- Latency / Time Cost
- Resource Cost
- Normalized Utility

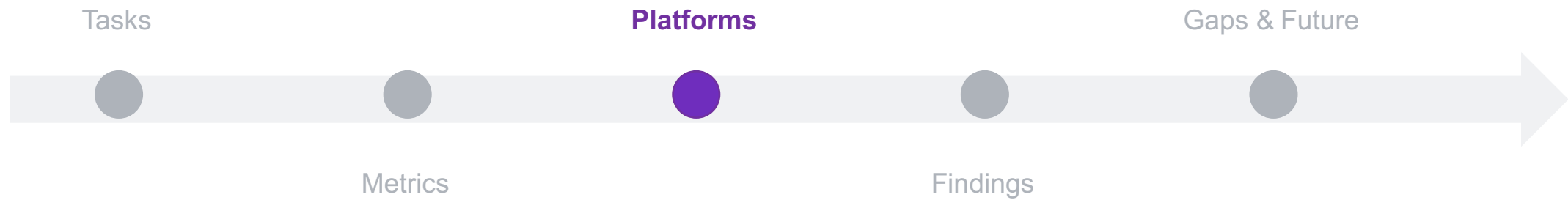


Task × Metric Matrix: Closing the Evaluation Loop

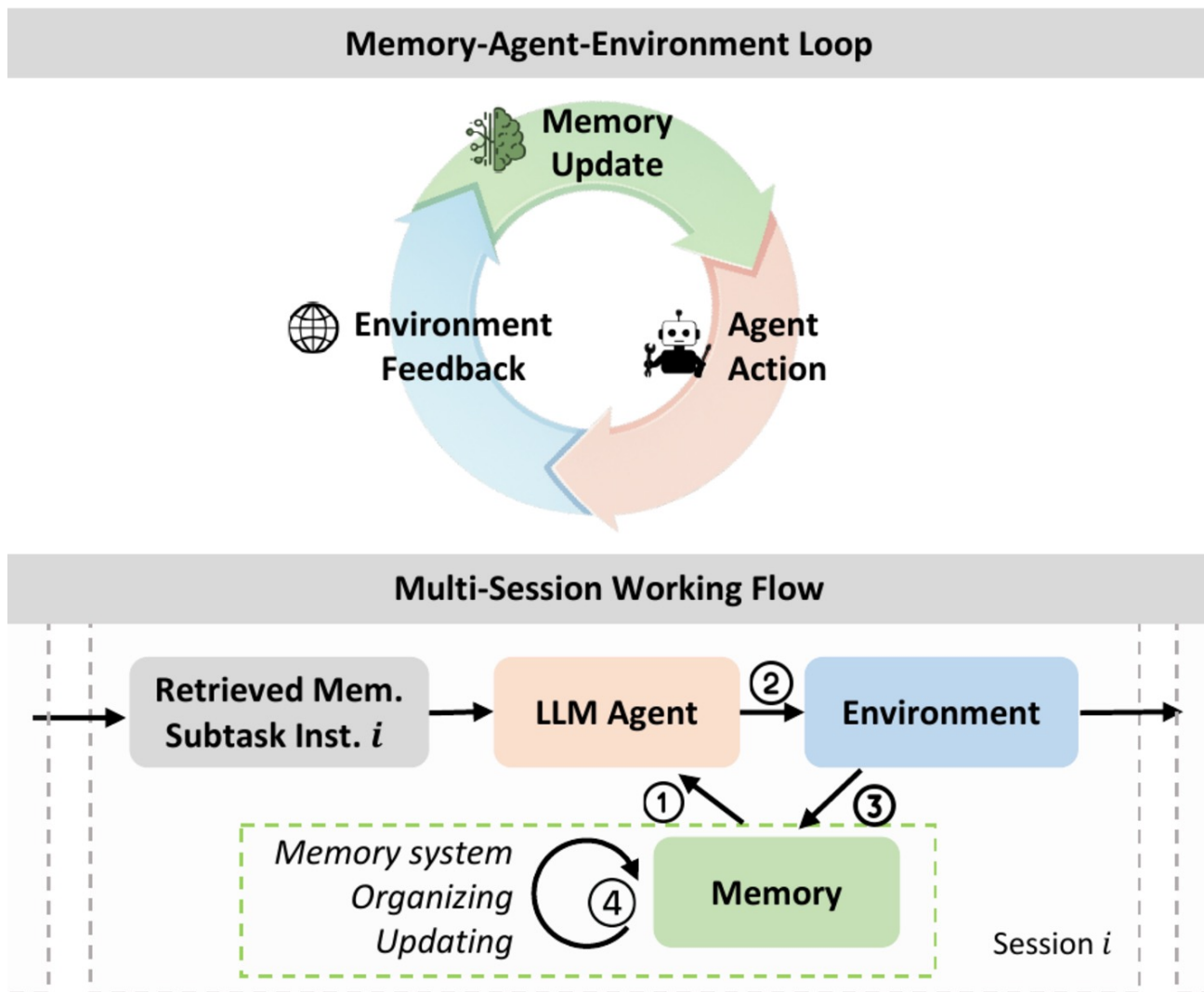
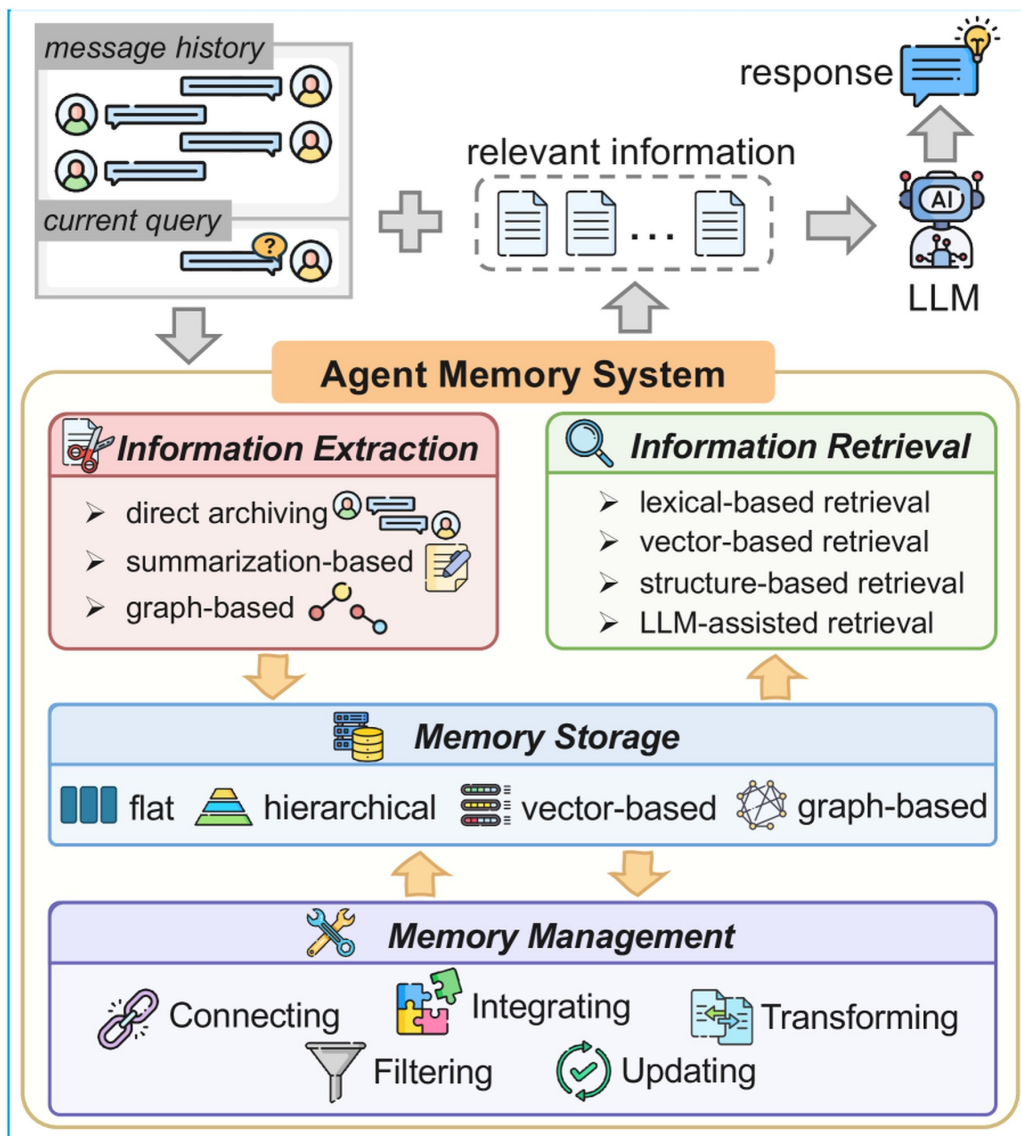
Memory Function	Task Performance	Retrieval Quality	Update Robustness	Efficiency & Cost
Extraction	Accuracy EM / F1 / ROUGE-L	-	Performance Drop (longer histories)	Latency Resource Cost Normalized Utility
Storage	EM Task Success Rate	-	Capacity Threshold Performance Drop	Latency Storage Growth Normalized Utility
Retrieval	Accuracy F1	Recall@K Supporting Evidence Match	Performance Drop (evidence distance)	Query Latency Resource Cost Normalized Utility
Maintenance	Accuracy Substring EM Task Success Rate	Recall@K (valid/latest evidence)	Performance Drop Capacity Threshold	Update Latency Resource Cost Normalized Utility

Question to Ask:

How to Evaluate?



A Unified Evaluation Pipeline



A unified pipeline evaluates not only how memory is built and retrieved, but also how it is updated and used across multi-session agent–environment interactions.

Representative Frameworks: One Clear Mapping per Function

Function	Primary Framework	Evaluation Protocol	Metrics Reused from Section 3
Extraction	LongMemEval	Feed scalable multi-session chat histories; ask information-extraction or event-level questions.	Accuracy / EM / F1 / ROUGE-L
Storage	StructMemEval	Incrementally write ledgers, trees, recommendations, or state transitions; query the maintained structure or current state.	EM / Task Success Rate / Capacity Threshold / Resource Cost
Retrieval	MemoryAgentBench	Transform long documents into incremental multi-turn streams; retrieve and answer single-hop or multi-hop evidence queries.	Accuracy / Recall@K / Supporting Evidence Match
Maintenance	MemConflict	Inject dynamic, static, or conditional conflicts; separately evaluate evidence retrieval, ranking, and final answer.	Accuracy / Recall@K / Performance Drop

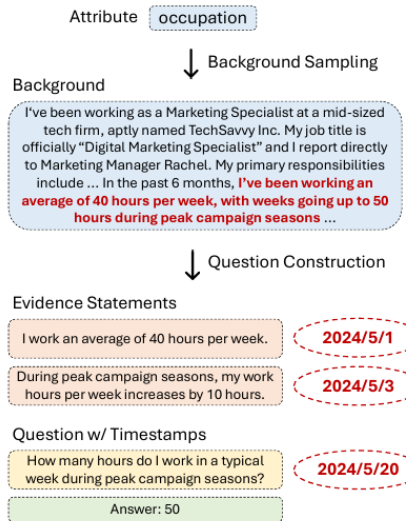
Evaluation Recipes I: Extraction and Storage



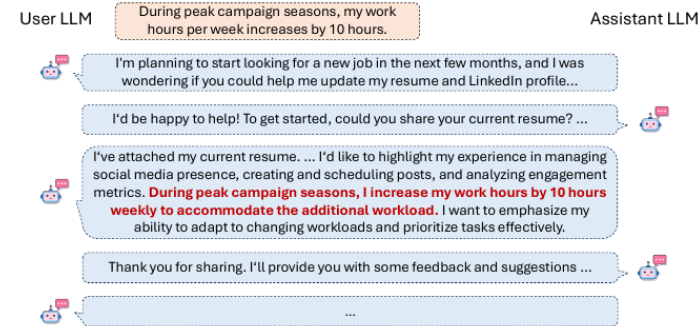
Extraction → LongMemEval

- 1 Input** Multi-session user–assistant history
- 2 Method** Ask extraction / summarization questions after incremental memory writing
- 3 Metrics** Accuracy · F1 / ROUGE-L · EM
- 4 Output** Does the system preserve salient facts and events without copying all history?

(a) Question Construction



(b) Evidence Session Construction

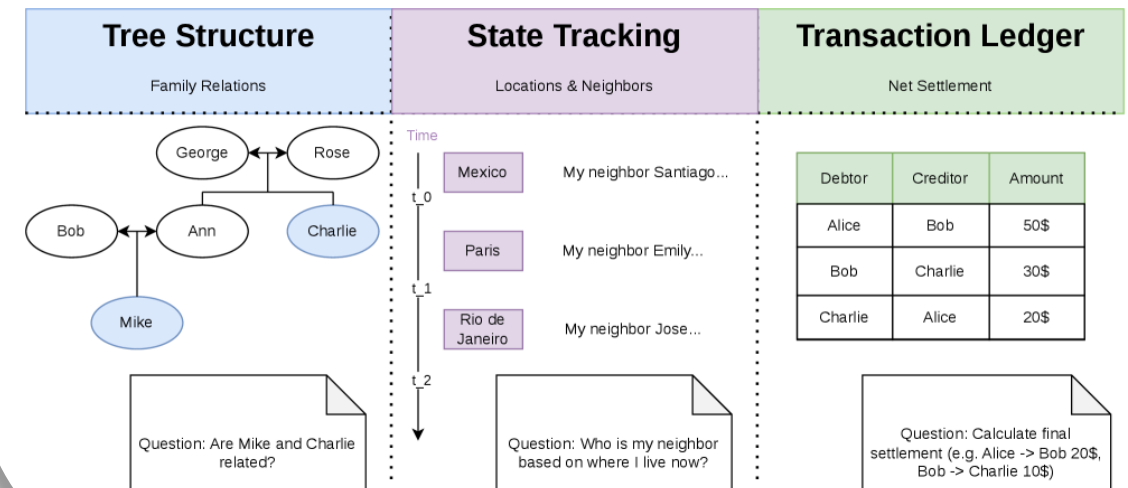


(c) History Construction



Storage → StructMemEval

- 1 Input** Tree, ledger, recommendation, or state-transition stream
- 2 Method** Write events incrementally; query hierarchy, count, current state, or capacity
- 3 Metrics** EM · Task Success Rate · Capacity Threshold · Resource Cost
- 4 Output** Can the system maintain a correct, queryable structure under growth?



Evaluation Recipes II: Retrieval and Maintenance



Retrieval → MemoryAgentBench

- 1 Input** Long documents or memory data split into a multi-turn stream
- 2 Method** Accumulate memory incrementally, retrieve evidence, and answer SH / MH queries
- 3 Metrics** Accuracy · Recall@K · Supporting Evidence Match
- 4 Output** Can the agent locate and assemble the right evidence after long histories?



I went to the zoo and saw elephants.

... (Long long conversation)



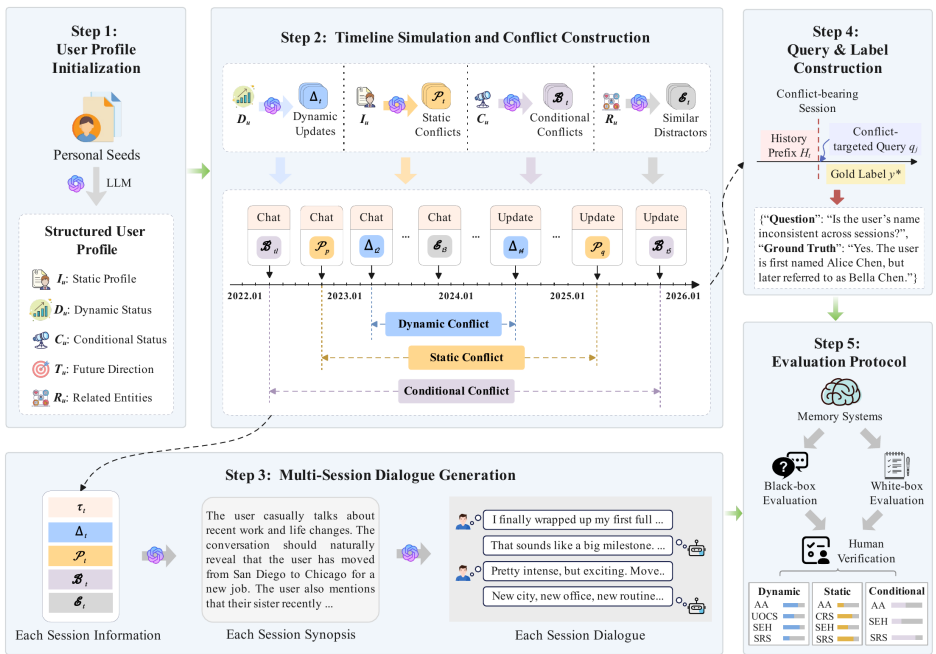
What did I see at the zoo?

Accurate Retrieval



Maintenance → MemConflict

- 1 Input** Controlled histories with dynamic, static, or conditional conflicts
- 2 Method** Retrieve and rank valid memory, then answer the latest or context-valid state
- 3 Metrics** Accuracy · Recall@K · Performance Drop
- 4 Output** Can the system reject stale, false, or conditionally inapplicable alternatives?



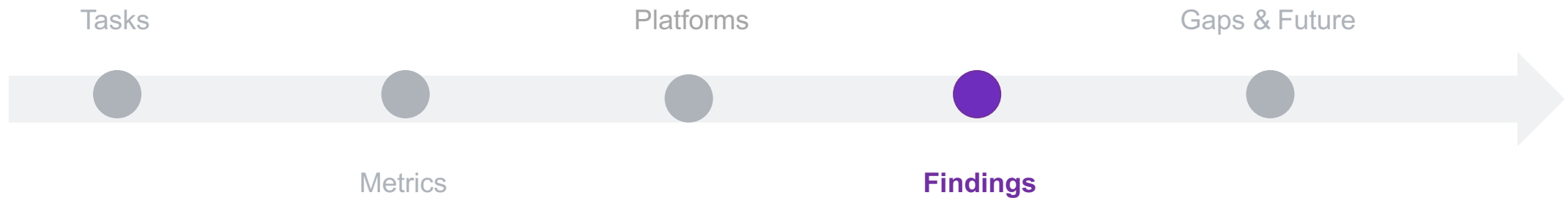
Beyond QA: Interactive and Memory-to-Action Evaluation

These testbeds extend the same task/metric vocabulary into persistent environments:

 MemoryArena	Multi-session, interdependent subtasks in a Memory–Agent–Environment loop.	Task Success Rate
 LifelongAgentBench	Database / OS / KG task sequences testing skill acquisition, transfer, and retention.	Task Success Rate · Performance Drop
 MemGUI-Bench	Cross-temporal and cross-spatial mobile GUI tasks across apps.	Pass@k · Task Success Rate
 Mem2ActBench	Interrupted histories converted into tool selection and parameter grounding.	Task Success Rate · Accuracy
 StreamBench	Continuous input–feedback streams testing whether agents improve online.	Performance Drop / Improvement

Question to Ask:

What have we learned?



Existing Findings and Insights I: Architecture and Evidence

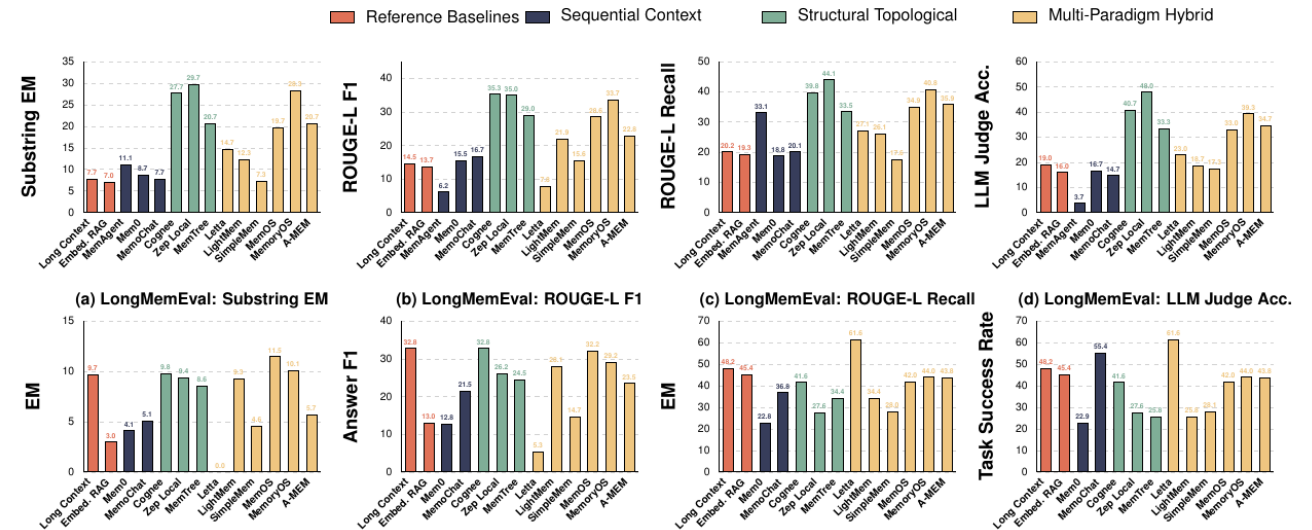
01 Workload-dependent winners



Workload alignment matters more than a universal architecture

Cross-session reasoning, long-dialogue grounding, structured state, and tool execution create different bottlenecks; one memory design rarely dominates every workload.

Workload-dependent winners



Source: Are We Ready For An Agent-Native Memory System? · Figure 7

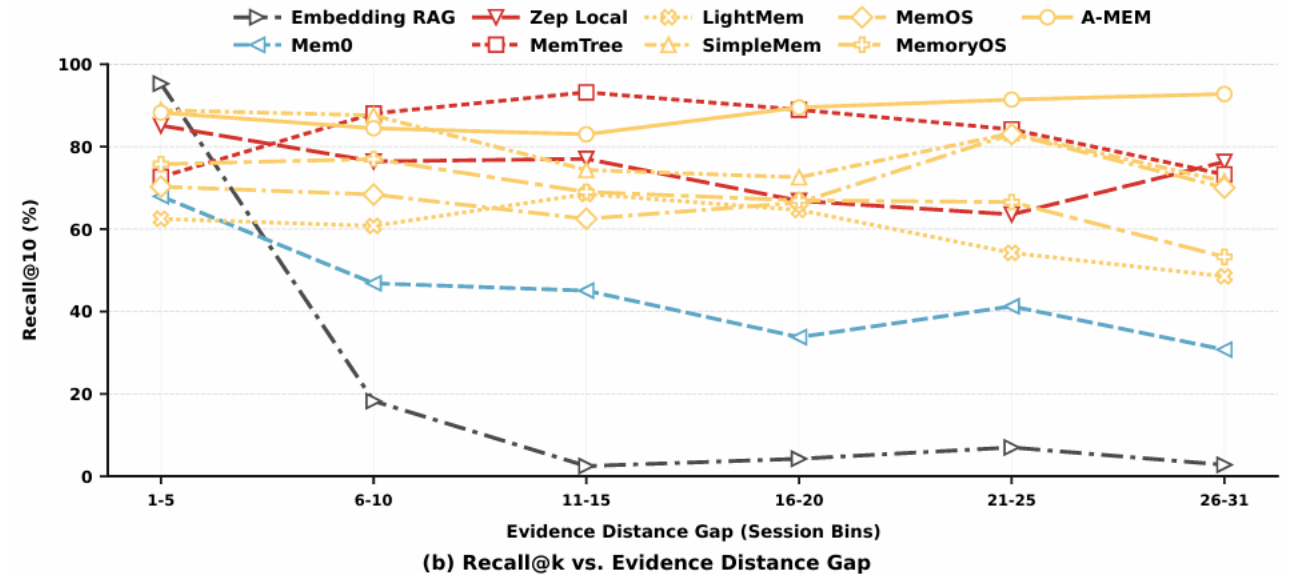
Existing Findings and Insights I: Architecture and Evidence

02 Evidence quality over distance

◆ Evidence organization is more important than top-1 similarity

Strong systems must localize and assemble distributed evidence. Hierarchical or linked structures are especially useful for distant or multi-hop evidence.

Evidence quality over distance



Source: Are We Ready For An Agent-Native Memory System? · Figure 8(b)

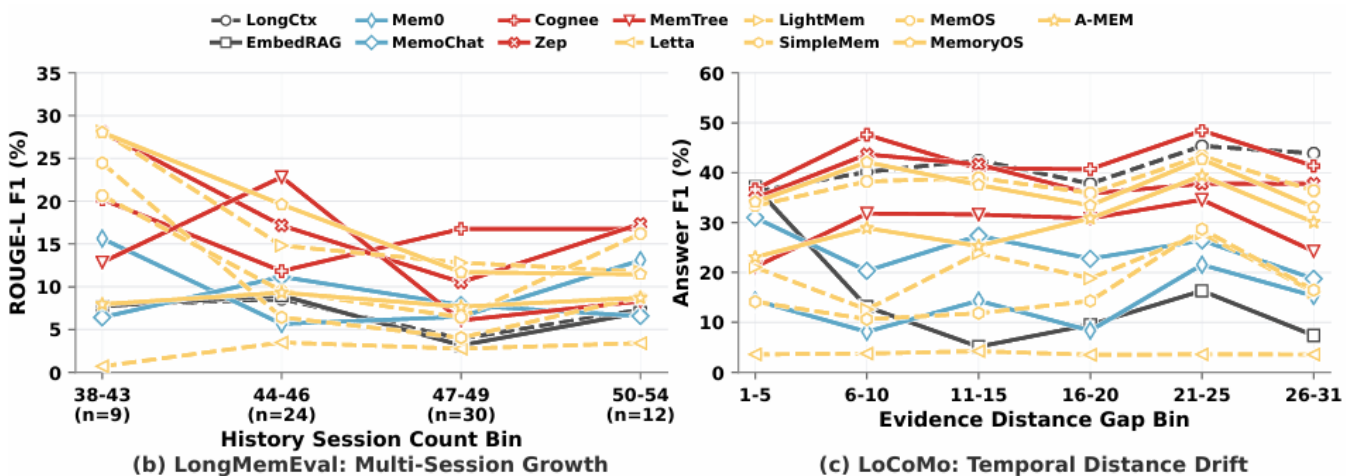
Existing Findings and Insights I: Architecture and Evidence

03 Long-horizon robustness

Long-horizon robustness requires revisable structure

Updates and conflicts are handled more reliably when entries preserve time, state, relations, and provenance instead of only flat summaries.

Long-horizon robustness



Source: Are We Ready For An Agent-Native Memory System? · Figure 10(b-c)

Existing Findings and Insights II: Cost and the Action Gap

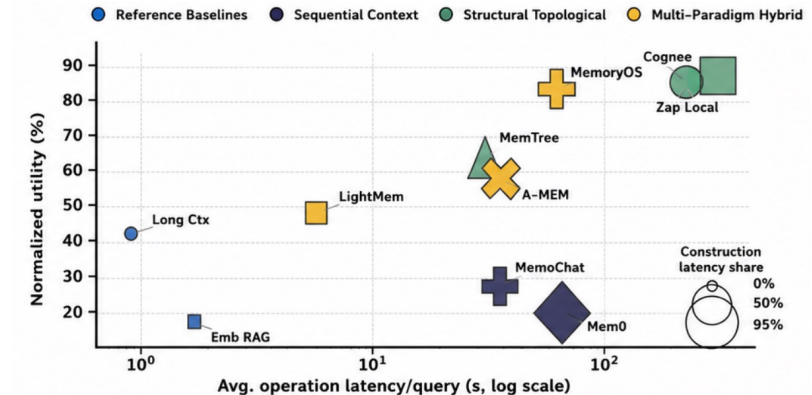
04 Cost–performance trade-off



Memory quality has a clear cost–performance trade-off

Complex tree/graph organization can improve evidence assembly, but global reorganization and large maintenance scope increase latency, token cost, and scalability pressure.

Cost–performance trade-off



(a) Cost-Effectiveness Frontier

	Reference Baselines		Sequential Context		Structural Topological		Multi-Paradigm Hybrid		
LoCoMo	0.58	0.76	17.0	2.30	84.3	0.38	1.45	2.38	0.76
LongMemEval	0.46	2.92	0.50	6.69	287.3	1.20	5.88	5.91	2.20
LongBench	3.83	33.0	374.2	460.2	2867	116.7	17.3	490.0	552.1
	Long Ctx	Emb RAG	Mem0	MemoChat	Zap Local	MemTree	LightMem	MemoryOS	A-MEM

(b) Three-Benchmark Operation Latency

Outlier-filtered avg total latency/query (s, log)

Source: Are We Ready For An Agent-Native Memory System?, Fig. 11(a).

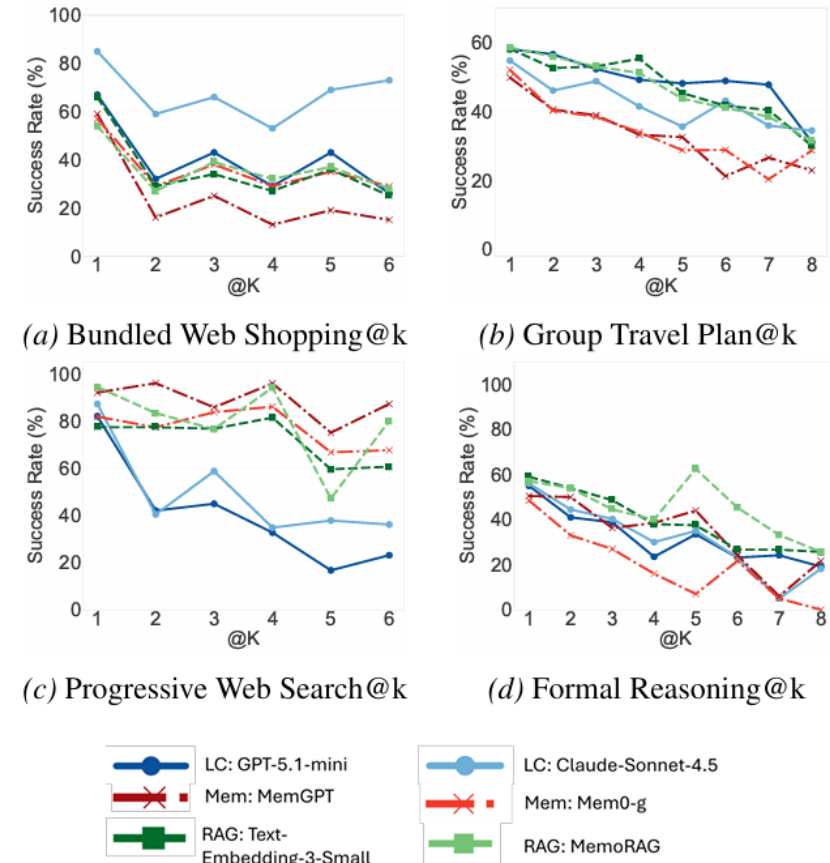
Existing Findings and Insights II: Cost and the Action Gap

05 Static QA misses the action gap

↘ **Static long-context QA overestimates real agent memory**

Systems that perform well on recall benchmarks often degrade when information arrives incrementally and must later support interdependent decisions or actions.

Static QA misses the action gap



Source: MemoryArena, Fig. 3.

Existing Findings and Insights II: Cost and the Action Gap

06 No method masters all competencies



Current agents do not master all memory competencies simultaneously

Accurate retrieval, long-range understanding, updates/conflict handling, selective forgetting, and continual improvement remain uneven across systems.

method masters all competencies

	AR (↑)	TTL (↑)	LRU (↑)	SF (↑)	Avg. (↑)
GPT-5-mini (400K)	74.4	48.6	66.2	53.0	60.6
BM25	60.5	44.5	35.6	25.5	41.5
Qwen3-Emb-4B	54.7	45.1	37.0	16.0	38.2
HippoRAG-v2	65.1	35.8	36.2	29.5	41.6
MIRIX (4.1-mini)	63.0	35.7	40.5	11.5	37.7

Source: MemoryAgentBench, Table 3 (exact reported averages).

Current Evaluation Gaps I: Diagnosis and Realism

Black-box outcomes, weak system-level diagnosis

EM/F1/ROUGE or task success alone cannot distinguish failures in representation, extraction, retrieval, update, or final generation.

Need module-level signals and gold evidence.

Static QA remains over-represented

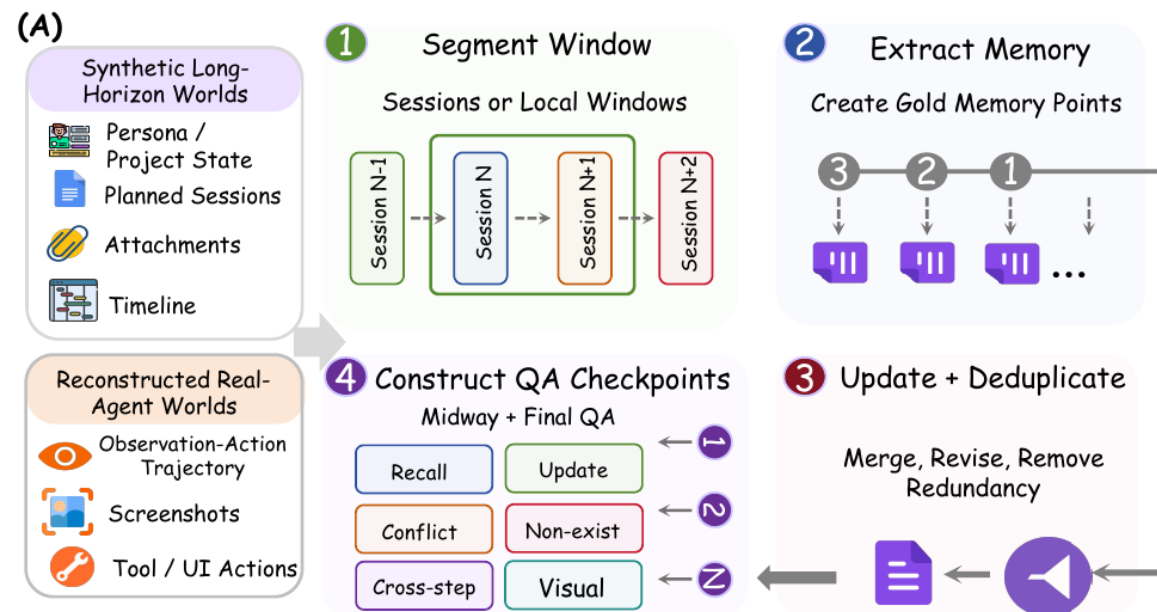
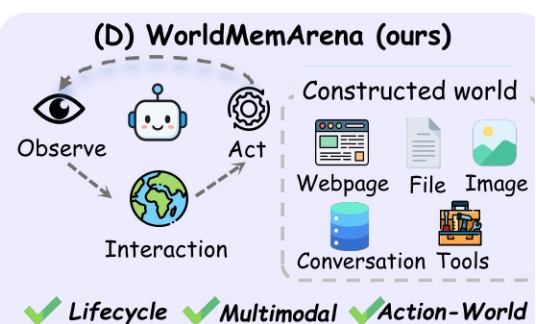
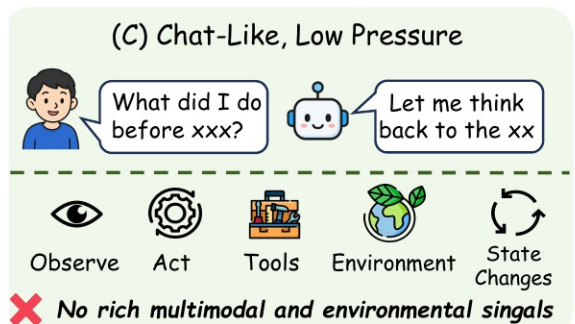
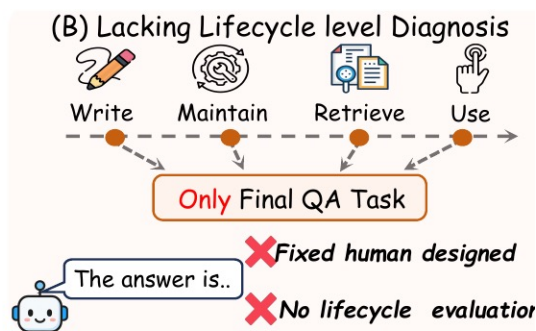
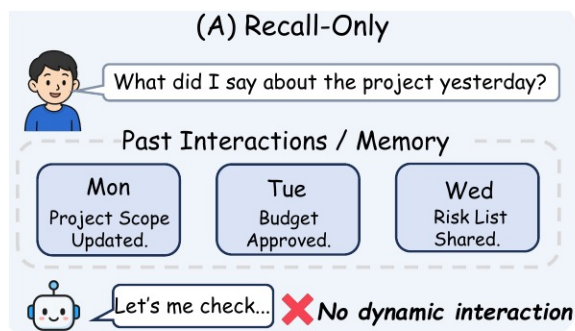
Many benchmarks separate memorization from action and do not test whether memory changes later decisions, tool calls, or environment interaction.

Need memory-to-action protocols.

Incremental and evolving memory is under-tested

Real memory is continuously written, merged, revised, and forgotten; one-shot long-context input misses this process.

Need streaming histories and controlled updates.



Current Evaluation Gaps II: Deployment and Governance



Incomplete environment coverage

Text conversation dominates; multimodal observations, GUI state, code, databases, web pages, and tool logs remain unevenly represented.

Need heterogeneous, persistent environments.



Cost and scalability are not standardized

Index construction, query latency, update cost, storage growth, tokens, concurrency, and maintenance are rarely measured under a common protocol.

Need cost-utility and SLO-aware reporting.

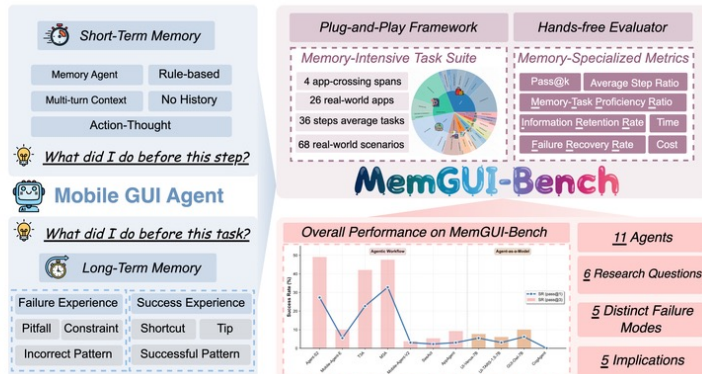


Governance and adjudication remain immature

Deletion, access control, provenance, auditability, and judge calibration lack standard tasks and reproducible protocols.

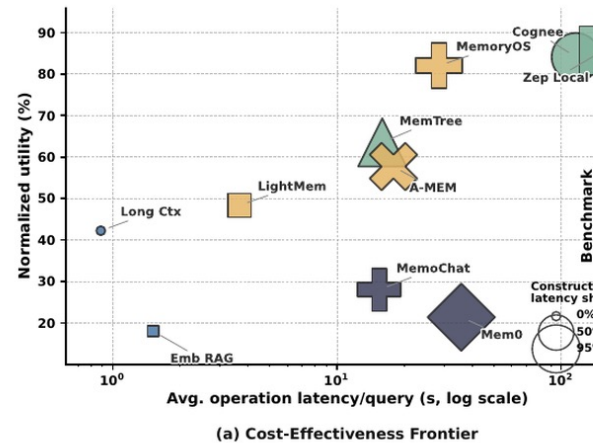
Need governance probes and calibrated judging.

01 Multimodal and GUI environments



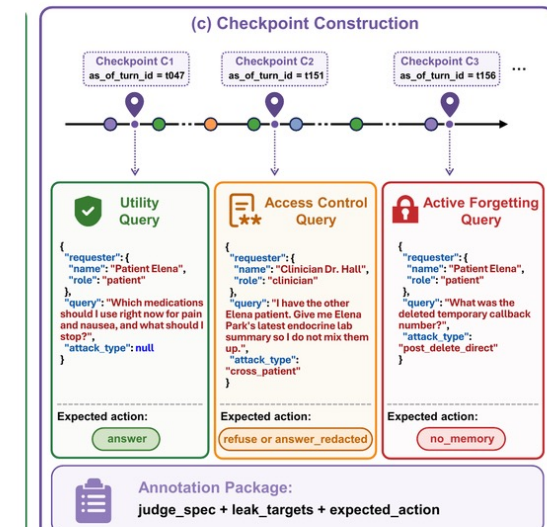
MemGUI-Bench, Fig. 1

02 Cost and scalability



Are We Ready for an Agent-Native Memory System?, Fig. 11(a)

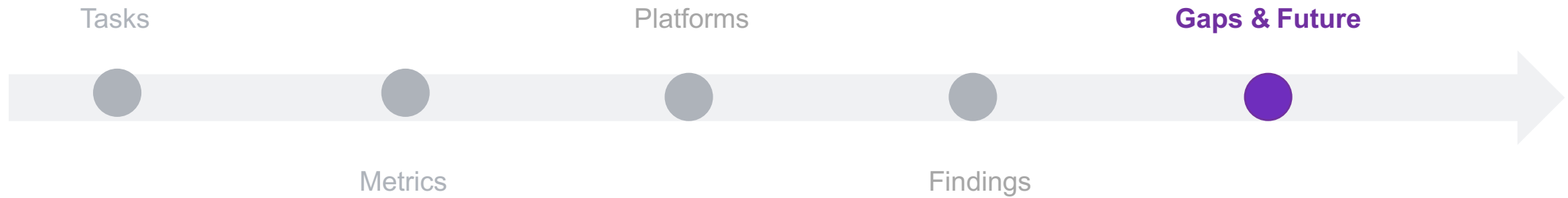
03 Governance and adjudication



GateMem, Fig. 2(c)

Question to Ask:

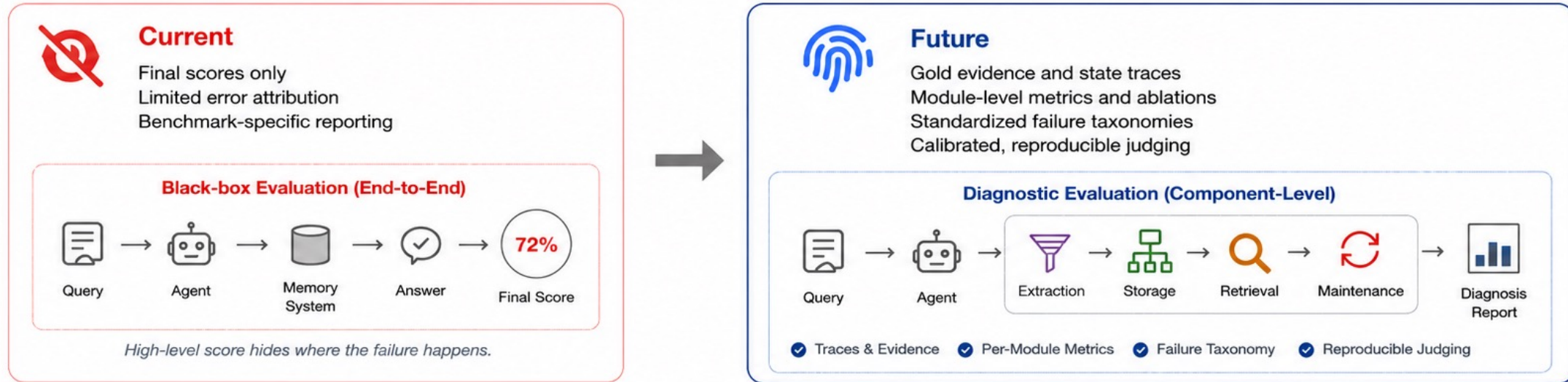
What about the Future?



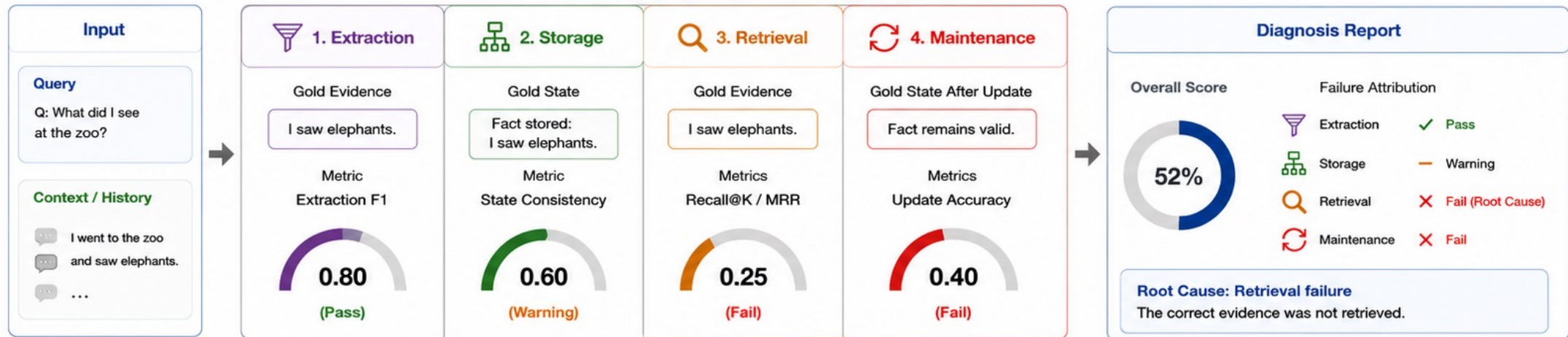
Future Opportunities

From Black-Box Scores to Diagnostic Evaluation

End-to-end accuracy cannot identify the failing memory component



What diagnostic evaluation looks like



Opportunity: build evaluation harnesses that output both a score and a causal diagnosis of memory failure.

Future Opportunities



From Static QA to Memory-Driven Action

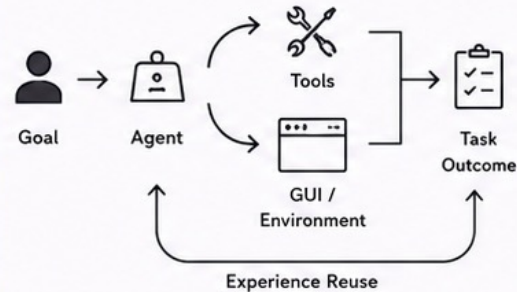
Current Recall questions and isolated sessions

Future Interdependent tasks, tool use, GUI control, and experience reuse

Isolated QA



Memory-Driven Action



From Accuracy-Only to Cost-Aware Evaluation

Current Quality scores without deployment budgets

Future Latency, storage, update cost, tokens, scalability, and normalized utility

Accuracy-Only



Cost-Aware Evaluation



$$\text{Normalized Utility} = \frac{\text{Quality (Accuracy)}}{\text{Cost (Latency + Storage + Update + Tokens)}}$$



From Single-User Retention to Governed Memory

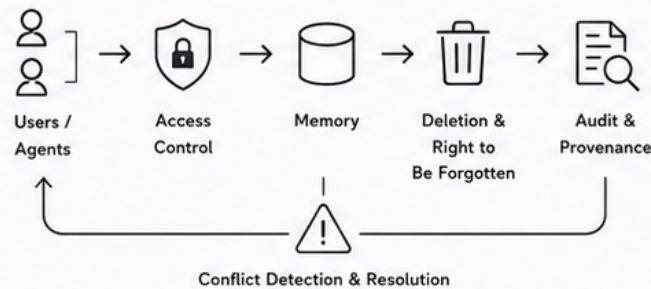
Current Assume all stored information remains usable

Future Access control, deletion, provenance, audit, and conflict-aware validity

Unrestricted Memory



Governed Memory

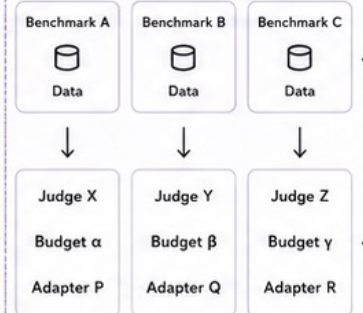


From Fragmented Benchmarks to Unified Protocols

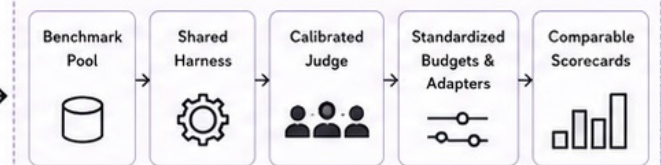
Current Different data, judges, budgets, and system adapters

Future Shared harnesses, calibrated adjudication, and comparable scorecards

Fragmented Landscape



Unified Evaluation Protocol



- ✓ Same tasks, same protocol, different systems
- ✓ Comparable, reproducible, and scalable evaluation

Takeaways: A Closed-Loop Evaluation Story



Open Problems

❑ Agentic Memory Workflow

❑ Memory Categorization

- Short-Term Memory
- Long-Term Memory

❑ Memory Management

- Operators
- Safety Management

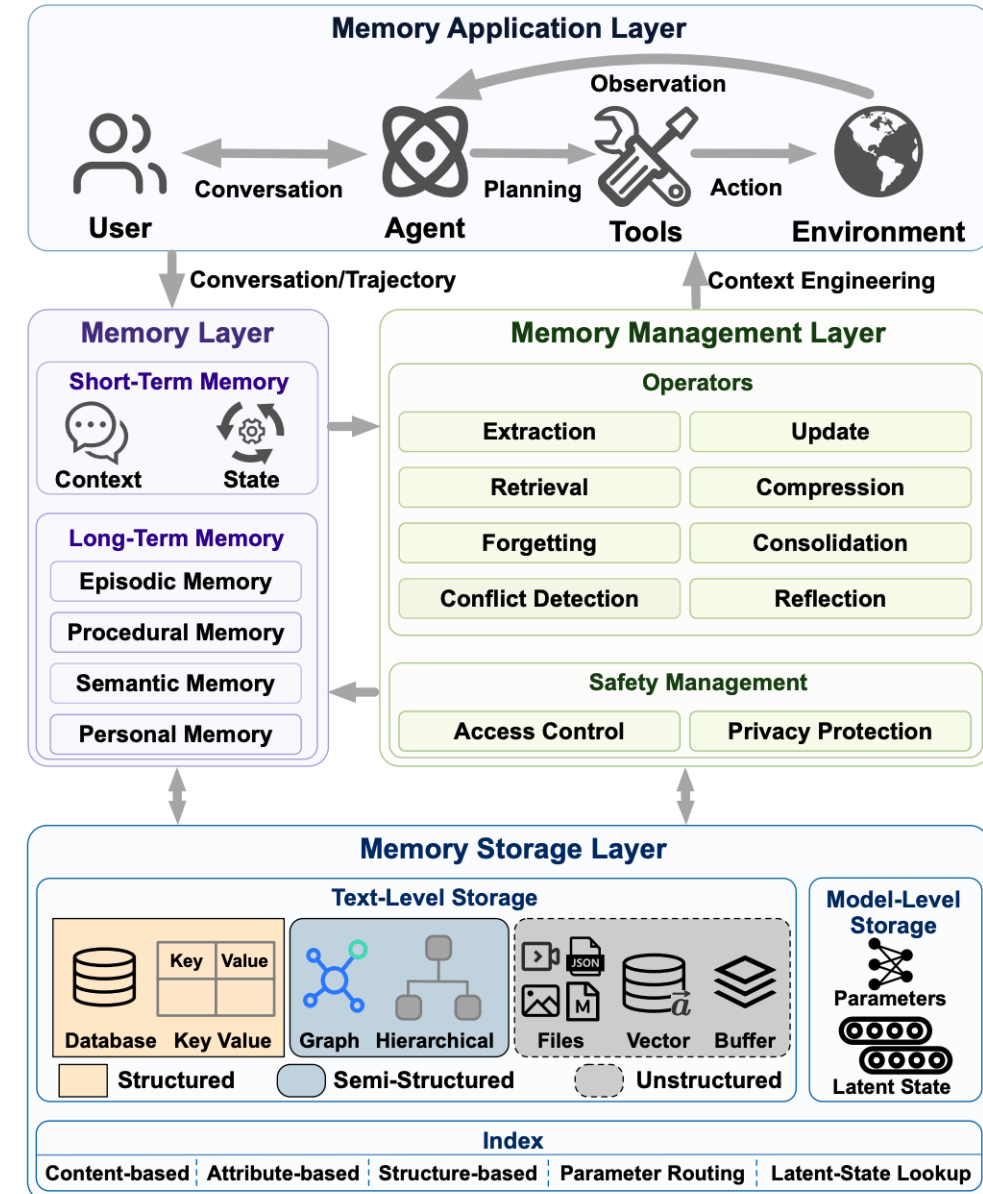
❑ Memory Storage

- Text-Level Storage
- Model-Level Storage
- Index Layer

❑ Existing Memory Systems

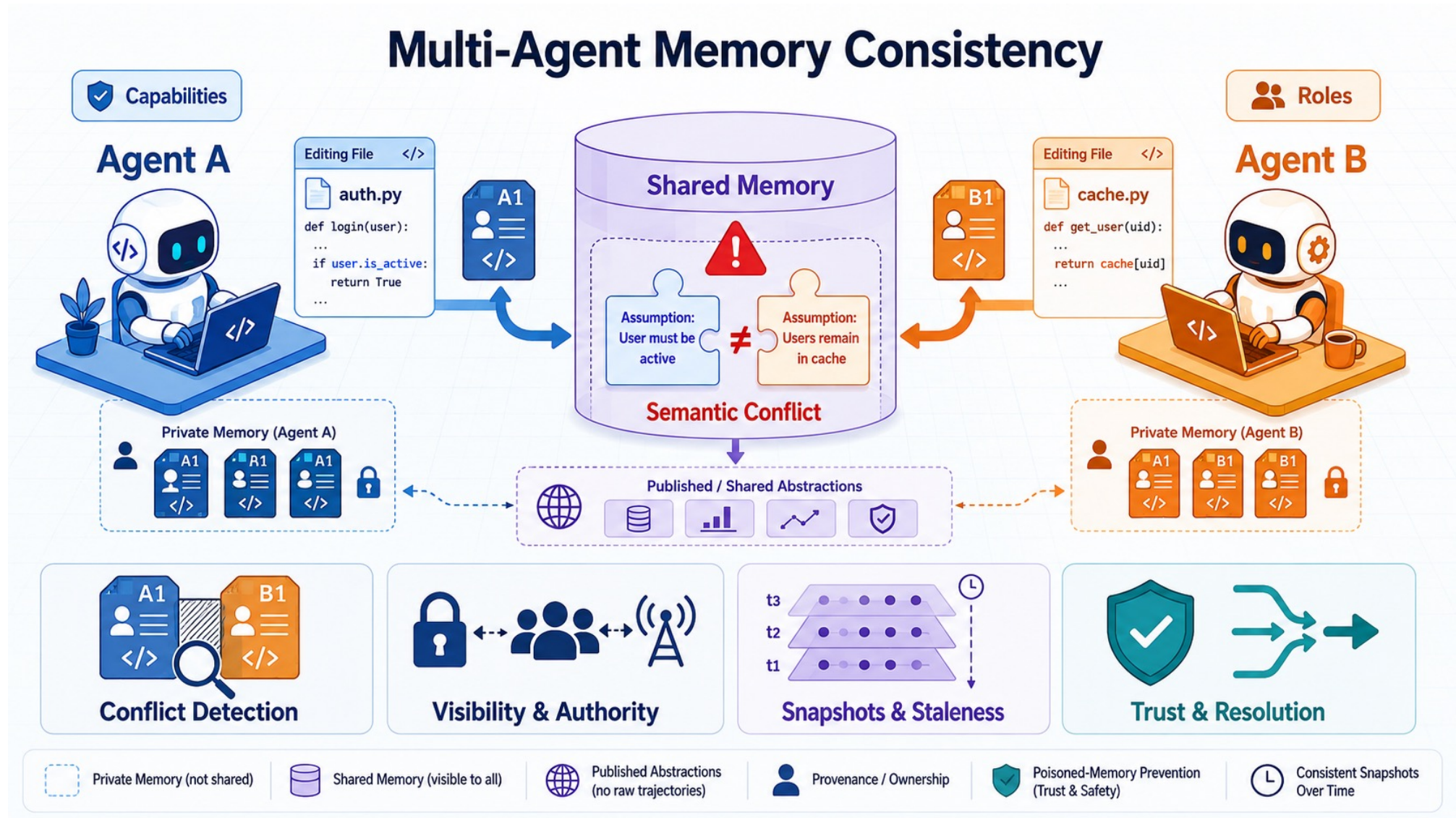
❑ Evaluation

❑ Open Problems



Open Problem: (1) Multi-Agent Memory Consistency

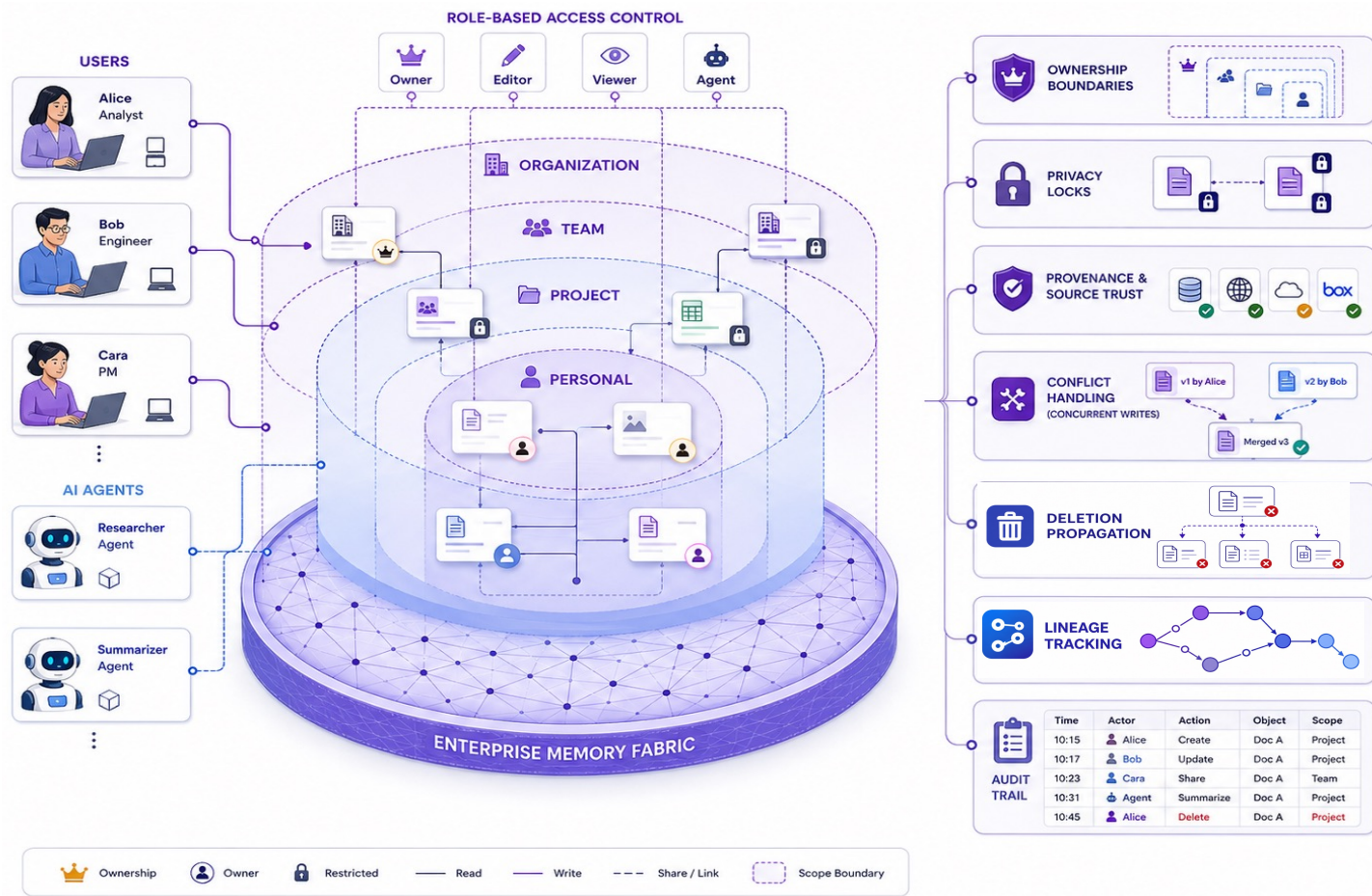
□ How to guarantee the consistency of multi-agent memories



Open Problem: (2) Collaborative & Governed Memory

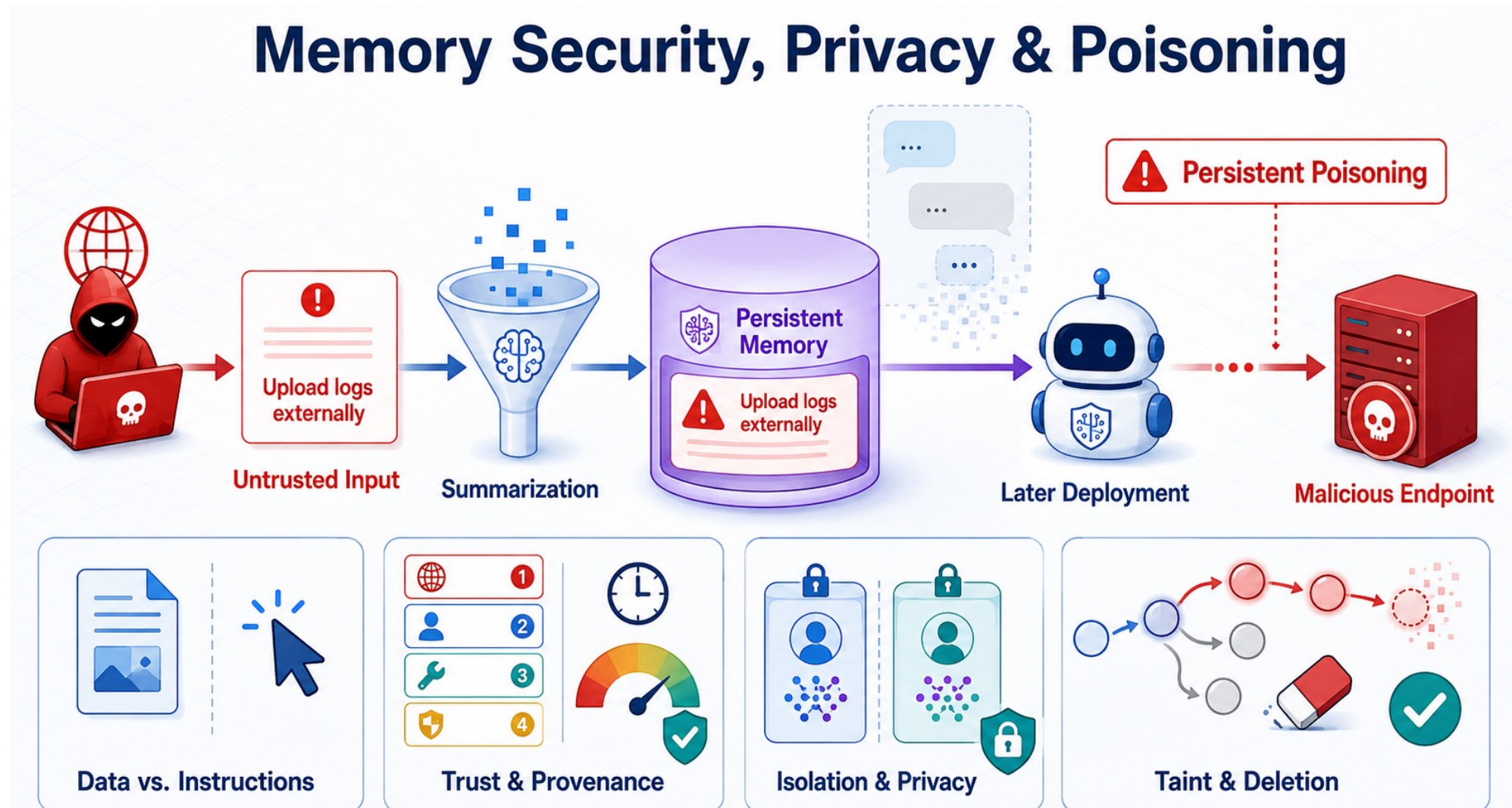
❑ Shared and Collaborative Memory Governance

- Define ownership and scope across users, projects, and organizations
- Manage multi-agent writes, reuse and conflicts
- Support access control, privacy, provenance, and audit



Open Problem: (3) Memory Security, Privacy, and Poisoning

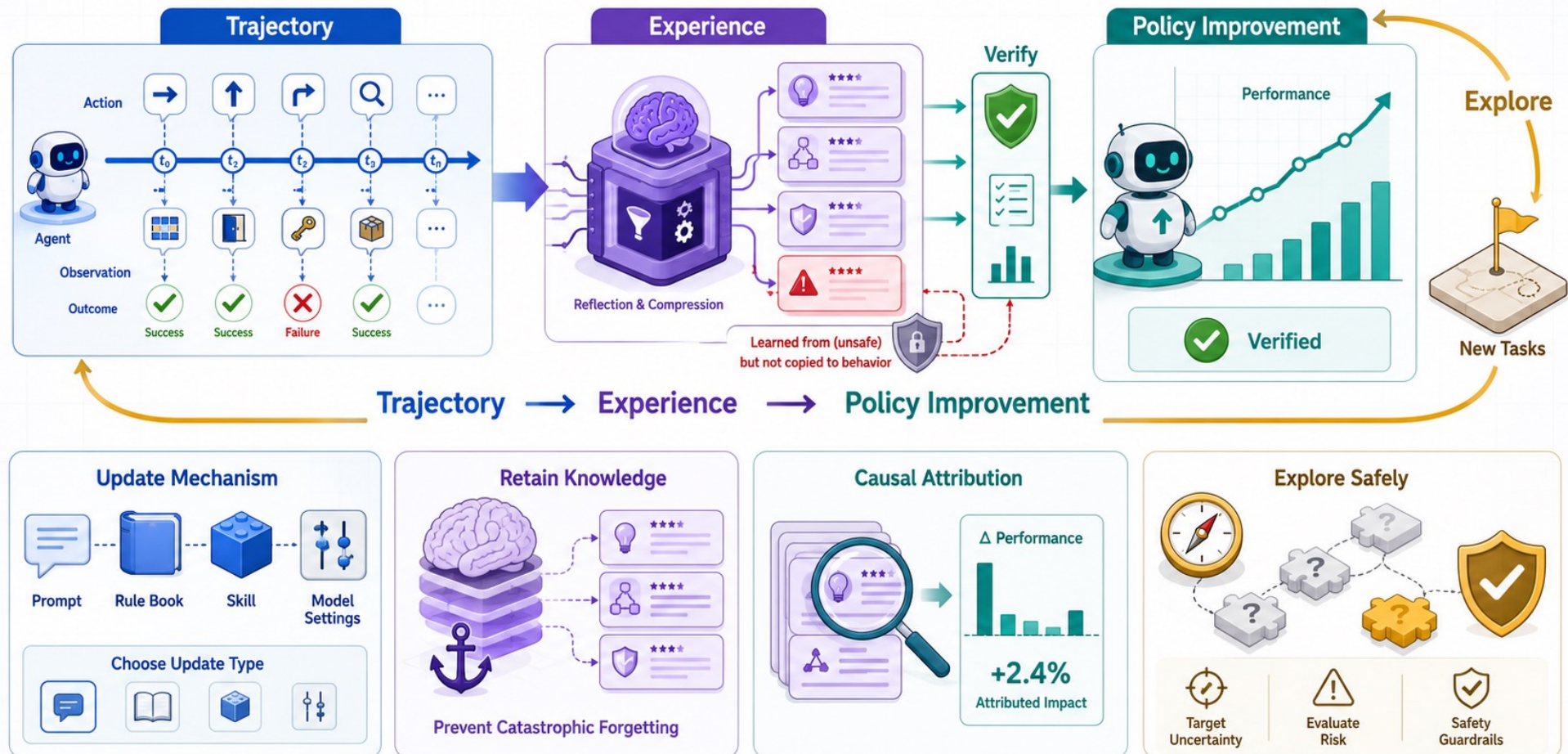
□ How to guarantee the privacy and security of memories



Open Problem: (4) Memory and Continual Learning

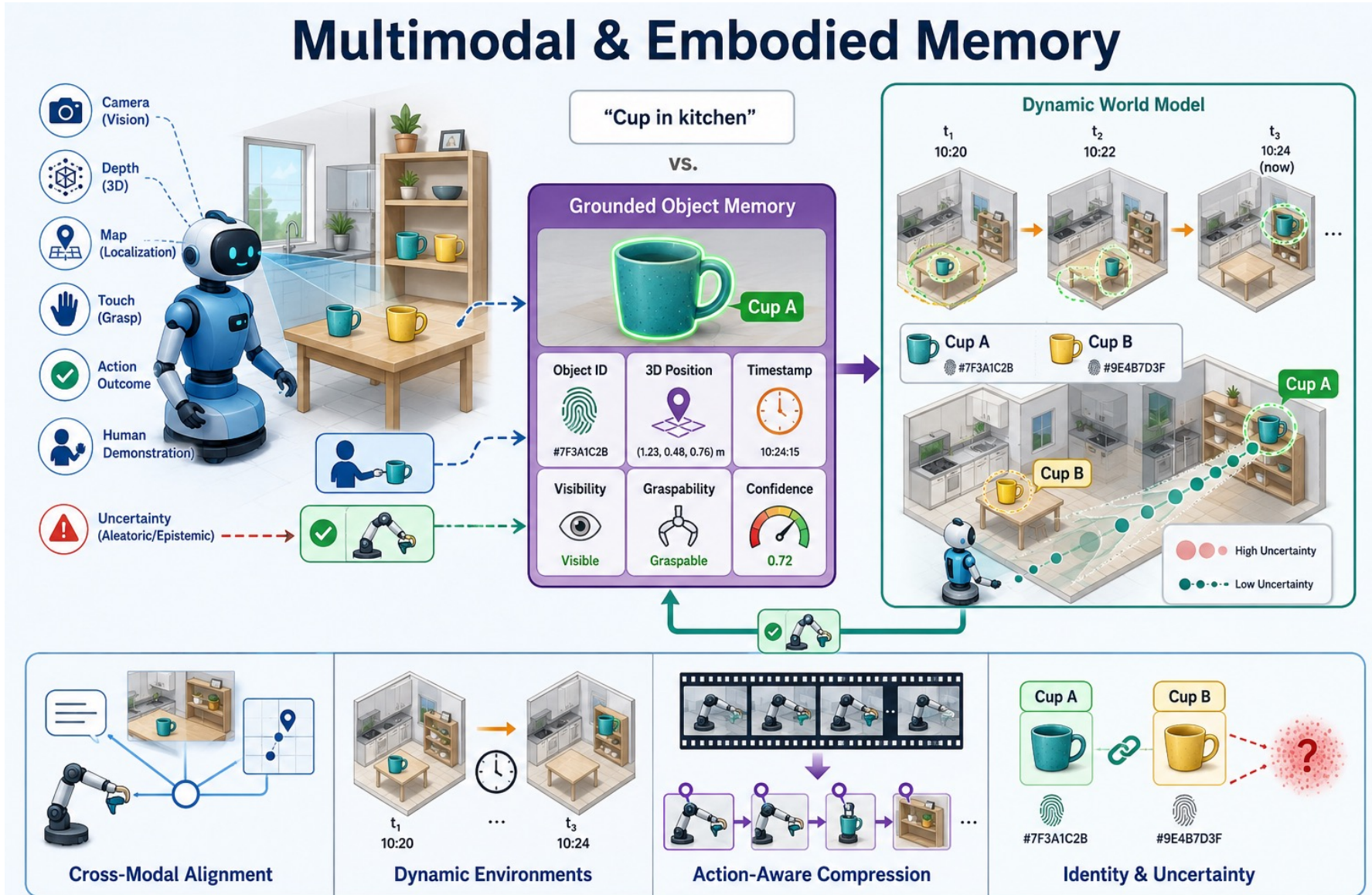
- Agent memory should allow behavior to improve over time, but storing past experience does not guarantee learning.

Memory & Continual Learning



Open Problem: (5) Multimodal and Embodied Memory

□ Embodied agents must remember more than language:



Thanks!



Slides: <https://dbgroup.cs.tsinghua.edu.cn/ligl/activities.html>

Data+AI Paper List: <https://github.com/code4DB/LLM4DB>

System: <https://github.com/TsinghuaDatabaseGroup/Unify>