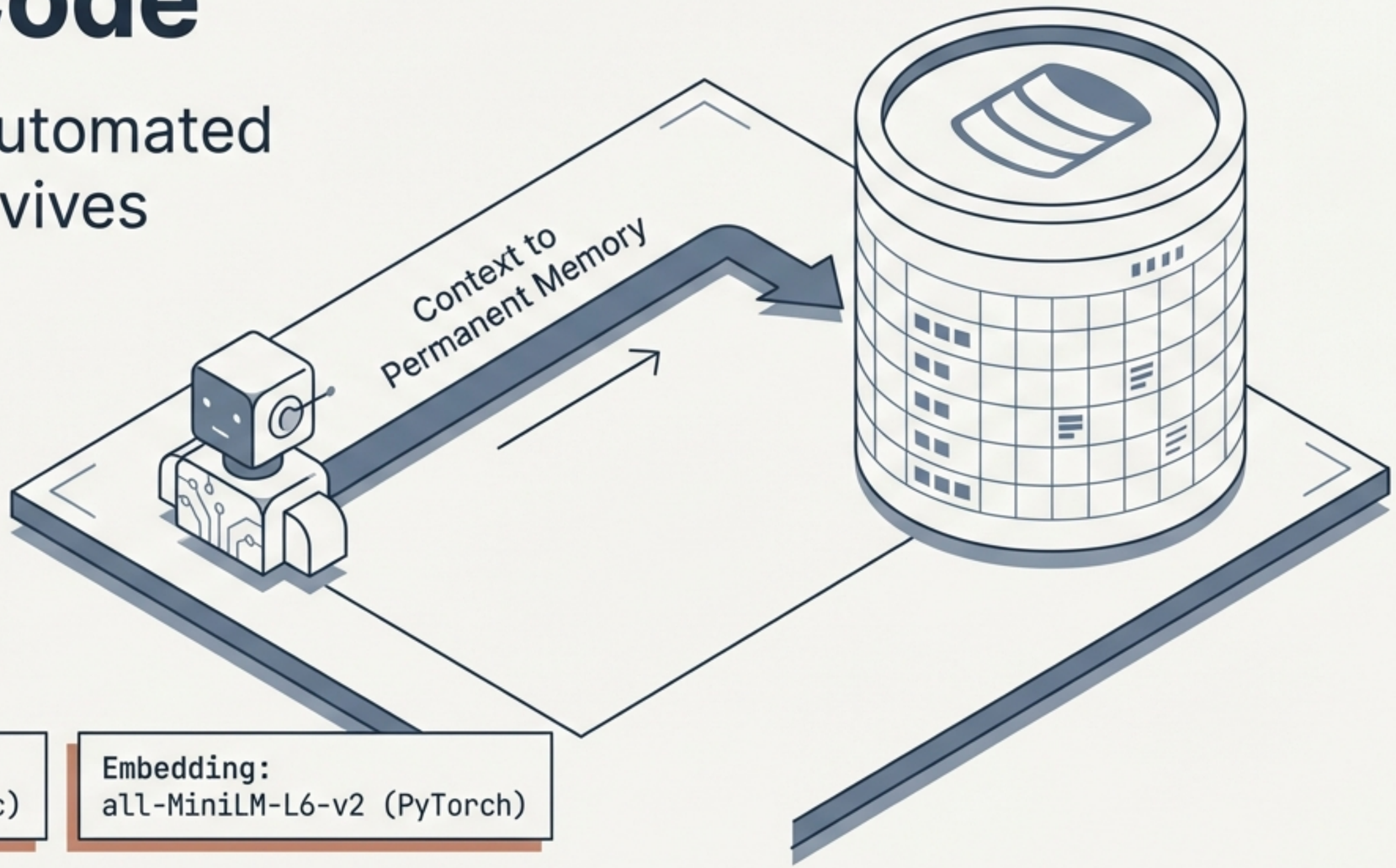
Persistent Memory for Claude Code

Building a Two-Tier Automated Architecture That Survives Session Boundaries

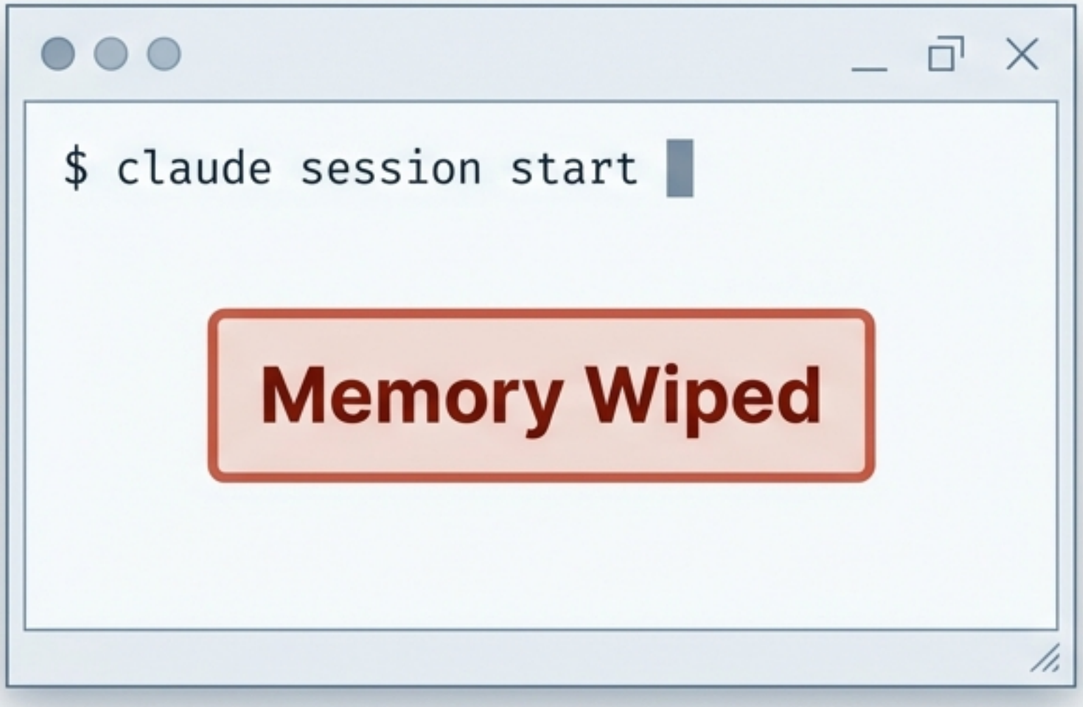


Version:
10.25.1

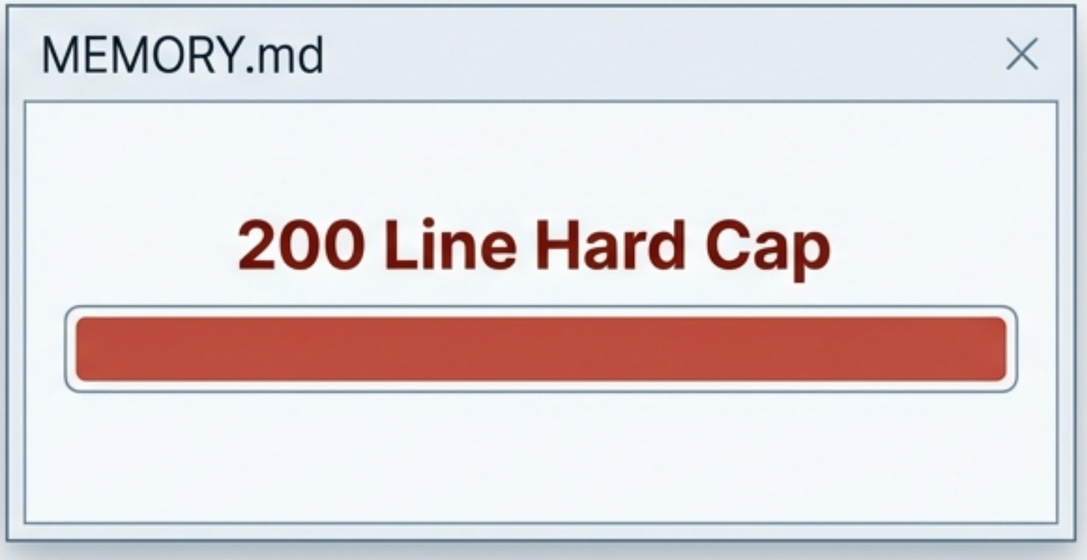
Backend:
Local SQLite (sqlite-vec)

Embedding:
all-MiniLM-L6-v2 (PyTorch)

Every Claude Code session starts at Day 1

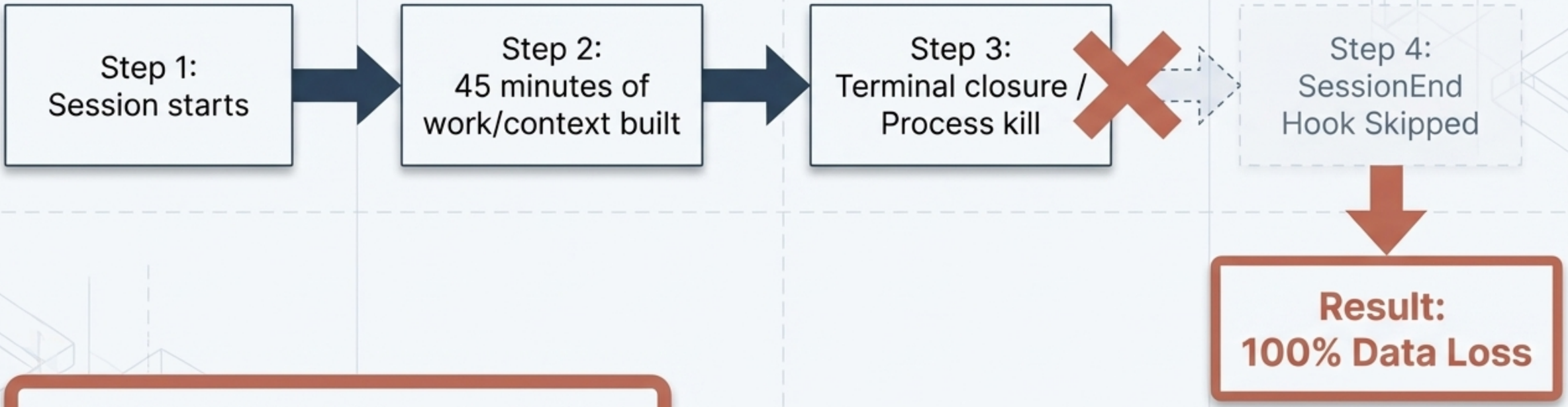


- Zero built-in long-term memory across session boundaries.



- MEMORY.md lacks semantic understanding and cross-project recall.
- Without automation,
- Without automation, retaining context requires manual user discipline, which does not scale.

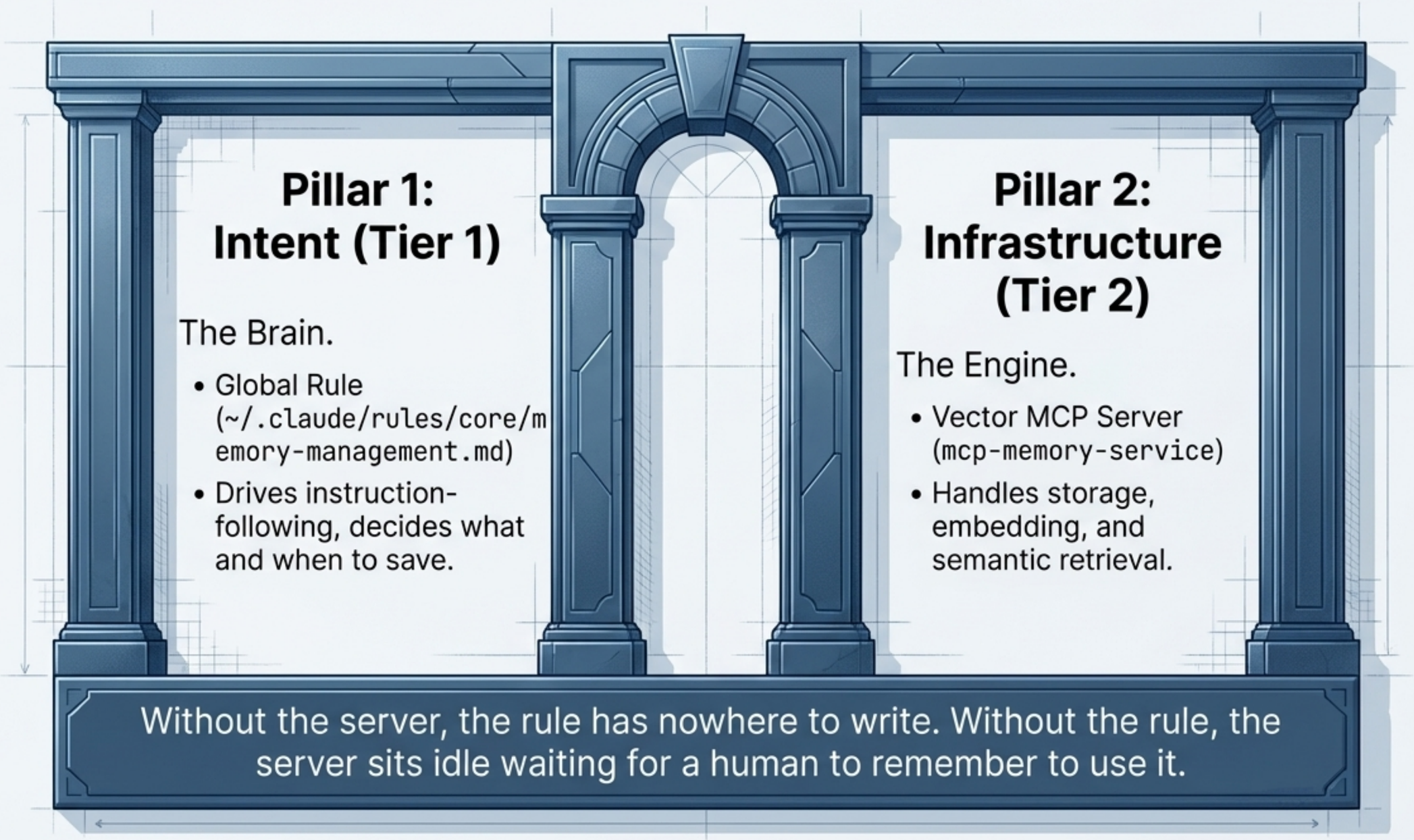
Why SessionEnd hooks provide a false sense of security



Save during, not after.

Hooks only execute on a clean exit. Designing a memory architecture around exit-time execution means terminal drops and process kills will permanently erase your session context.

The Two-Tier Architecture: Intent meets Infrastructure



Pillar 1: Intent (Tier 1)

The Brain.

- Global Rule (`~/.claude/rules/core/memory-management.md`)
- Drives instruction-following, decides what and when to save.

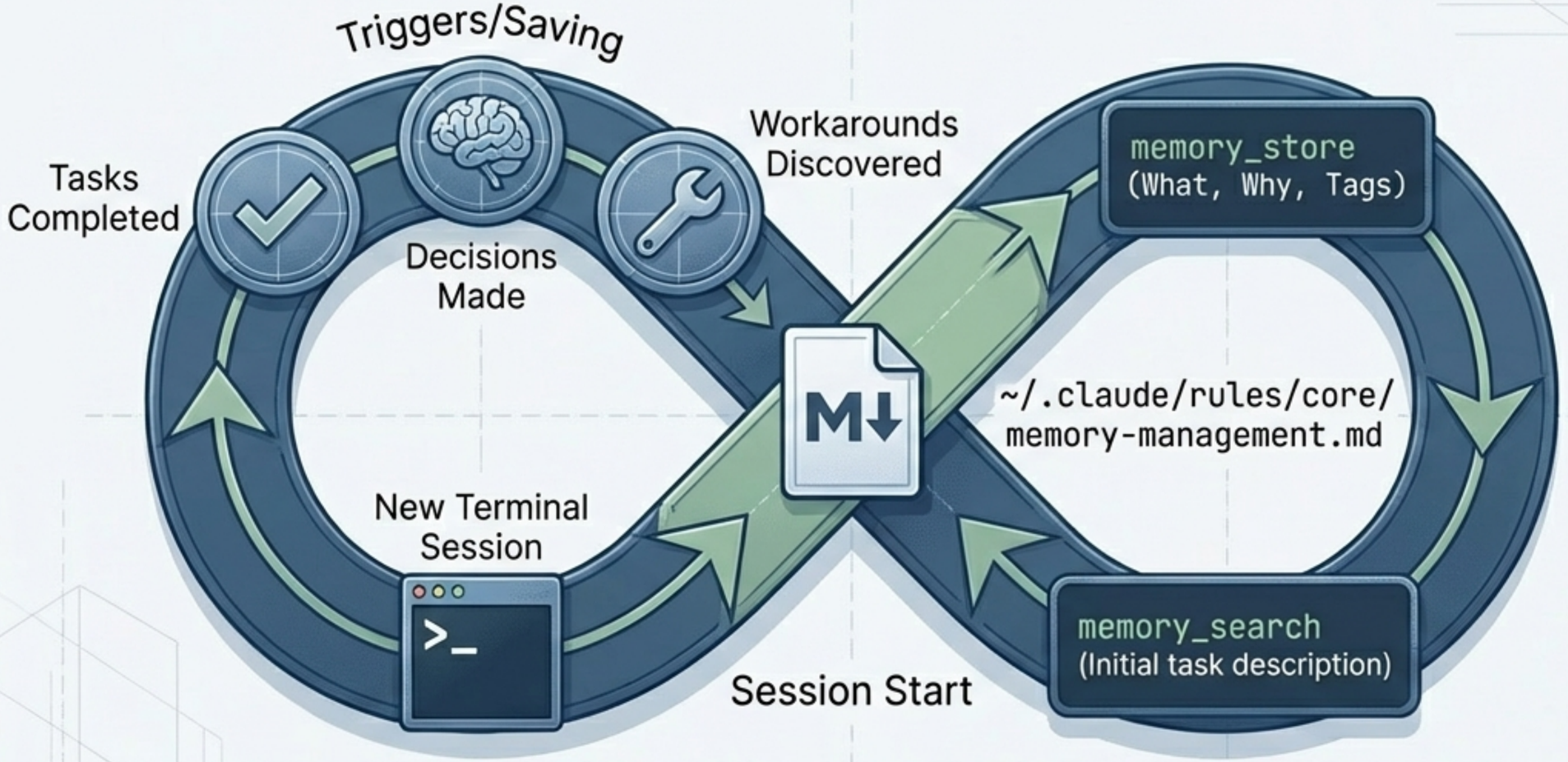
Pillar 2: Infrastructure (Tier 2)

The Engine.

- Vector MCP Server (`mcp-memory-service`)
- Handles storage, embedding, and semantic retrieval.

Without the server, the rule has nowhere to write. Without the rule, the server sits idle waiting for a human to remember to use it.

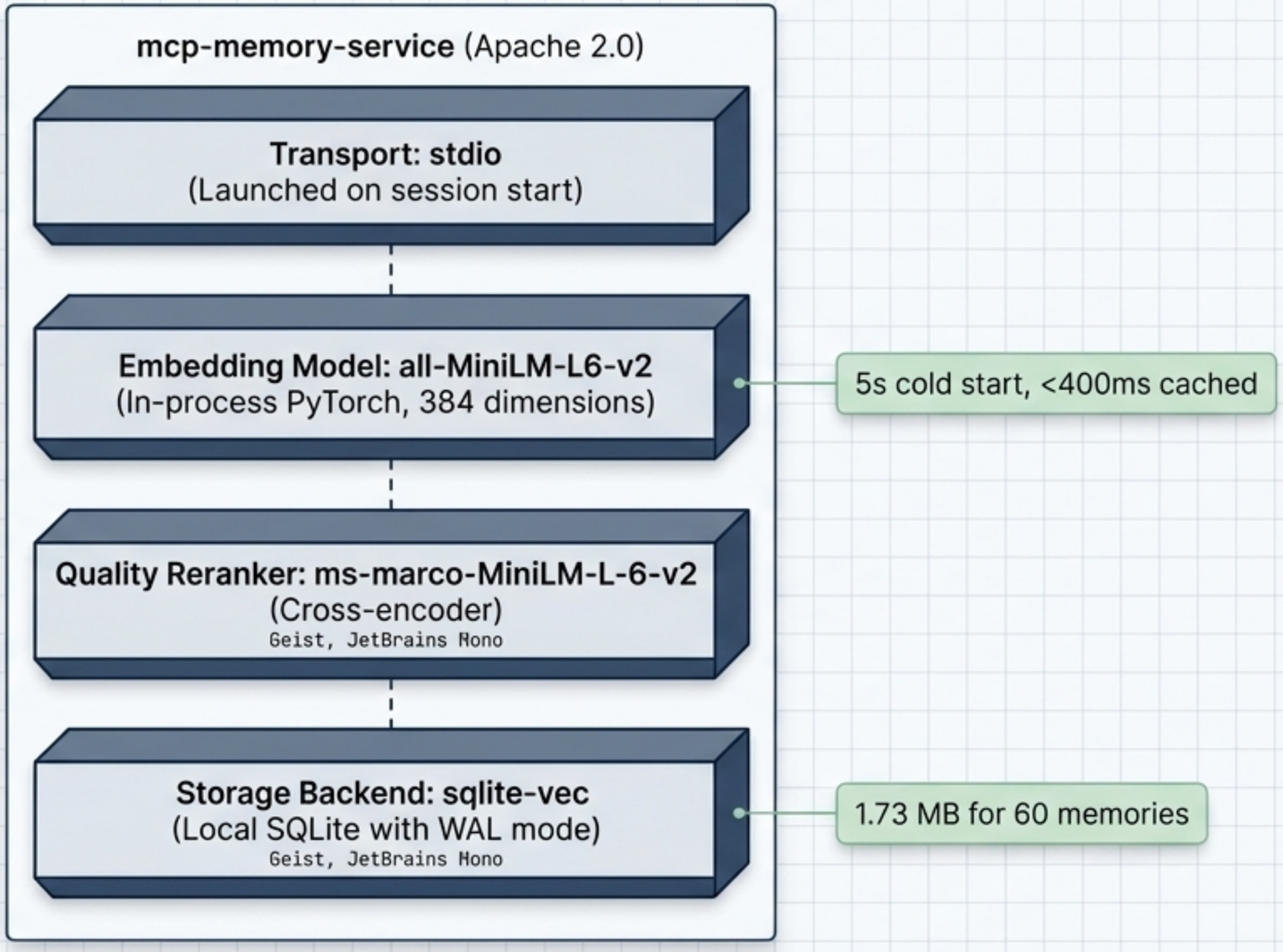
Tier 1: The Global Rule automates capturing and retrieval



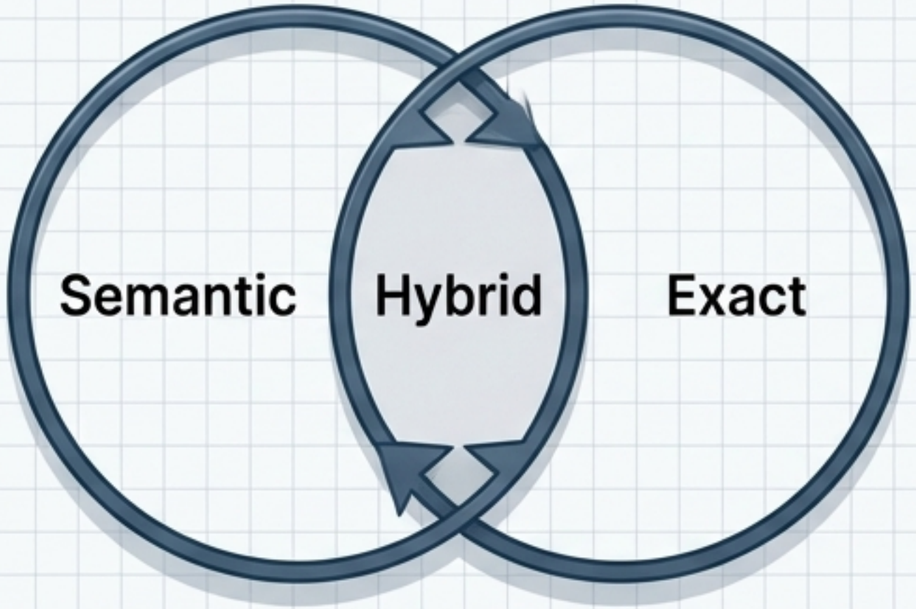
By making memory management Claude's explicit job, we remove the human discipline bottleneck.

Tier 2: A zero-dependency, local vector MCP server

100% local execution. No Ollama dependencies, no cloud API calls, zero per-query costs.



The **Search Triad** guarantees high-relevance recall



Semantic

Cosine similarity. (Example: “release problems” finds “deployment failures”).

Exact

Traditional BM25 keyword matching.

Hybrid

70% Semantic + 30% Keyword (Recommended).

Ranking Modifiers

- **MMR (Maximal Marginal Relevance):** Lambda 0.7 suppresses near-duplicates.
- **Temporal Decay:** 30-day half-life. (2-week memory = 65% relevance; 3-month = 12.5%).
- **Quality Boost:** Configurable 0.0-1.0 weight based on memory ratings.

Core Memory Commands Reference

`memory_store`	Saves text, tags, and type. Safety check: Uses conversation_id to bypass semantic dedup for incremental saves.
`memory_search`	Semantic/hybrid queries with ISO dates, natural time expressions ('last week'), and tag filters.
`memory_list`	Browse/paginate by category (note, fact, decision). No semantic matching.
`memory_update`	Modifies metadata/tags via content_hash while preserving exact embeddings and timestamps.

Automated maintenance and lifecycle management

Delete & Cleanup

- `memory_delete` (always use `dry_run: true`)
- `memory_cleanup` (Hash-based exact match + semantic deduplication)

Integrity

Daily automated backups (7-day retention) + 30-minute integrity checks

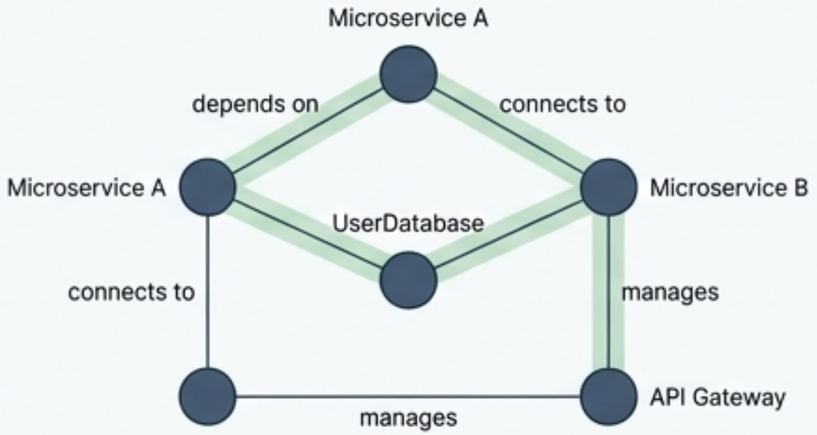
Quality Retention Table (via memory_quality tool)

Tier	Retention	Description
Critical	365 days	Architecture, core decisions
Reference	180 days	Recurring solutions
Standard	30-90 days	General notes
Temporary	7 days	Scratch notes

Selecting the right retrieval paradigm: Vector vs. Graph

Vector Memory (Default)

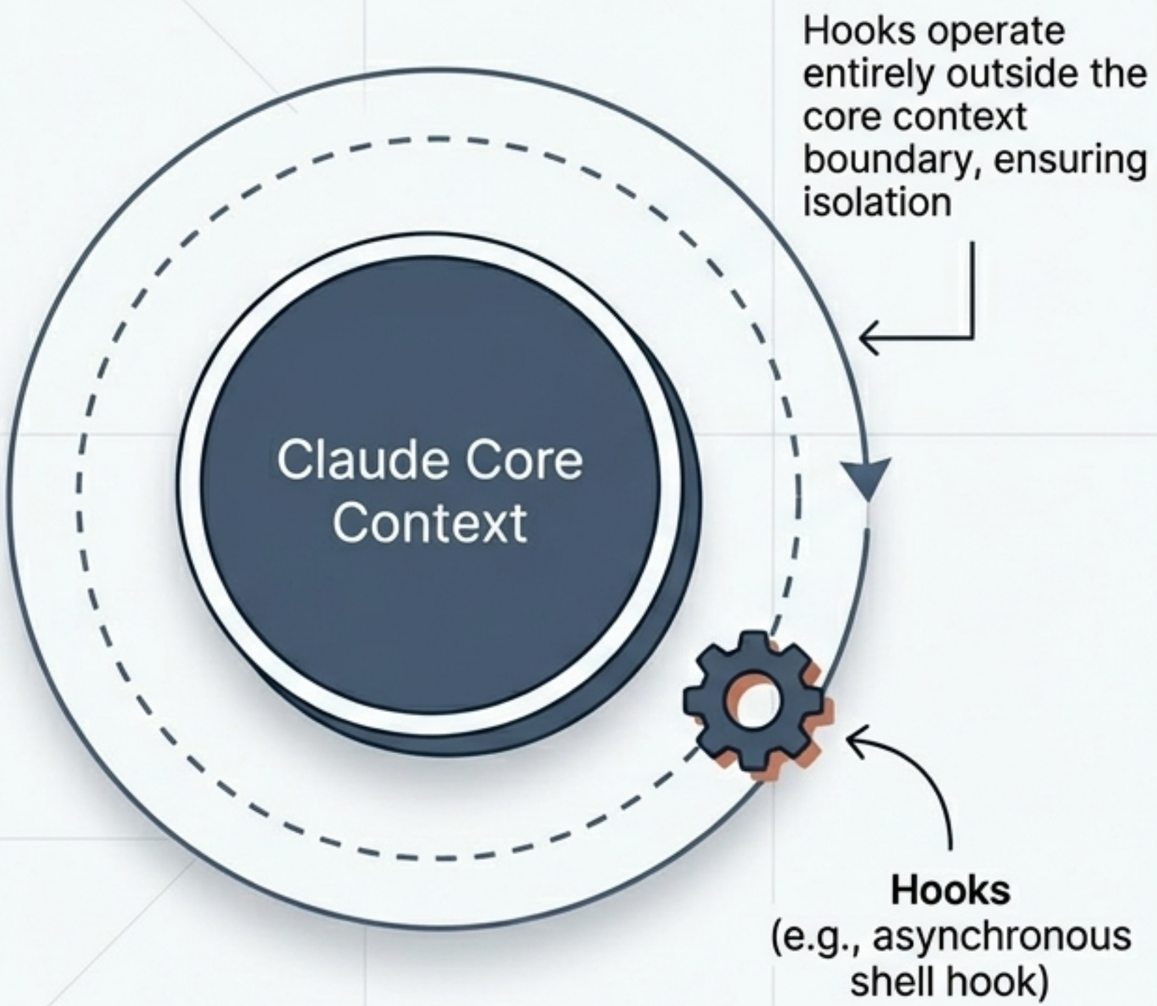
- **Best for:** Free-text recall, concepts, bug fixes.
- **Query style:** 'What do I know about the `SQLite migration issue`?'
- **Mechanism:** Semantic similarity across unbounded text.



Knowledge Graph (Supplemental)

- **Best for:** Explicit structural relationships, N-hop paths.
- **Query style:** 'Show me all microservices depending on `UserDatabase`.'
- **Mechanism:** Nodes and typed edges (via `memory_graph` BFS/Shortest Path).

Hooks are for observation, not persistence



Recommended Hooks ecosystem

- File Guard (PreToolUse): Blocks edits to sensitive files (`.env`, `.pem`).
- Activity Logger (PostToolUse): Dumps raw file edits and bash commands to `activity_log.txt`.
- Observation Capture (PostToolUse): Archives raw tool JSONL to `observations.jsonl` for separate AI pattern analysis.
- Session Archive (SessionEnd): Copies full transcripts to a local archive directory on clean exits only.

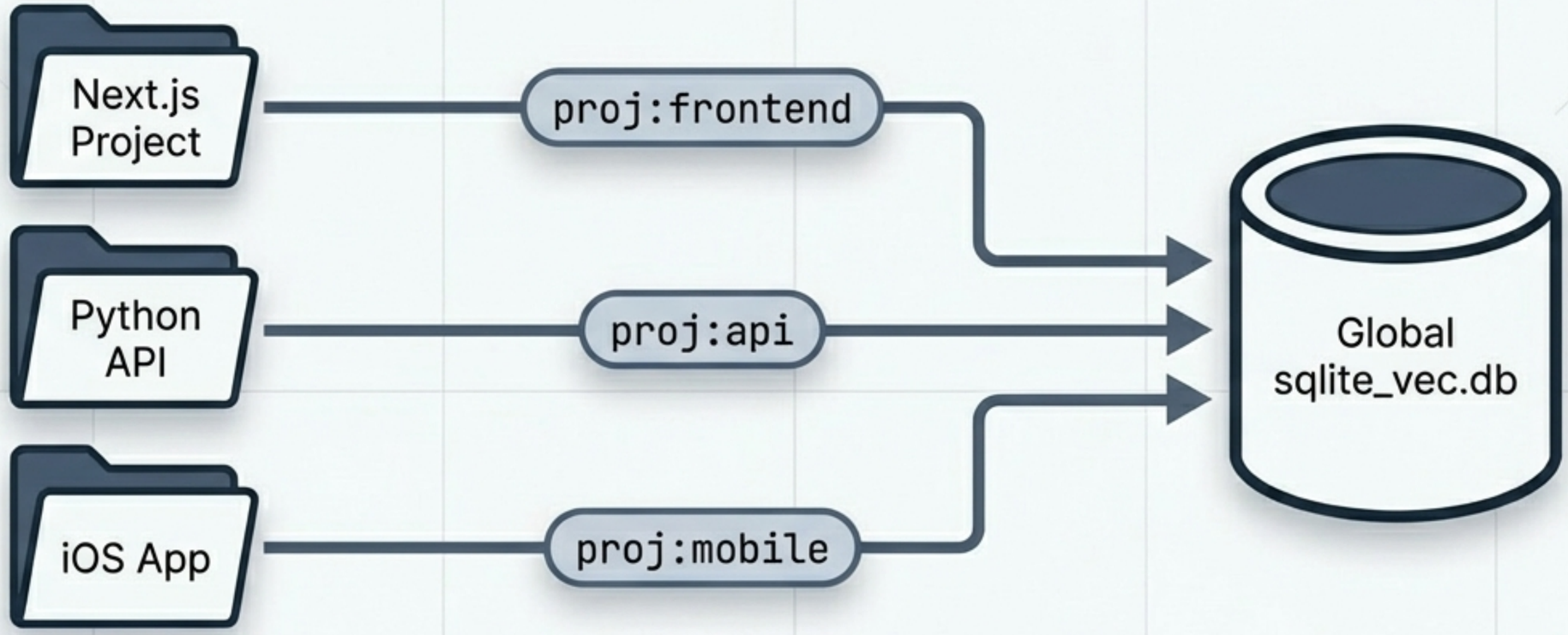
Removing friction with Auto-Approve Permissions

~/.claude/settings.json

```
"customSettings": {
  "autoApprove": {
    "mcpServers": [
      "vector-memory"
    ]
  }
}
```

Without this configuration, Claude prompts the user for manual confirmation on every single memory_store call. Friction defeats automation.

Global database scope with project-level tag isolation



- A single shared database `~/.claude.json` prevents multi-repo fragmentation.
- Context is separated entirely via tags (e.g., `proj:frontend`, `proj:api`).
- Enables cross-project context queries when architectures overlap.

System supports Cloudflare (D1/Vectorize) or Hybrid local/cloud sync for multi-machine setups.

Implementation Gotchas & Lessons Learned

Module Naming

The config must target `mcp_memory_service.server`. Executing the top-level package causes a fatal 'cannot be directly executed' error.

Python Paths

The `python3.11` command in `~/.claude.json` must be the exact binary path used for pip installation.

OAuth Key Leaks

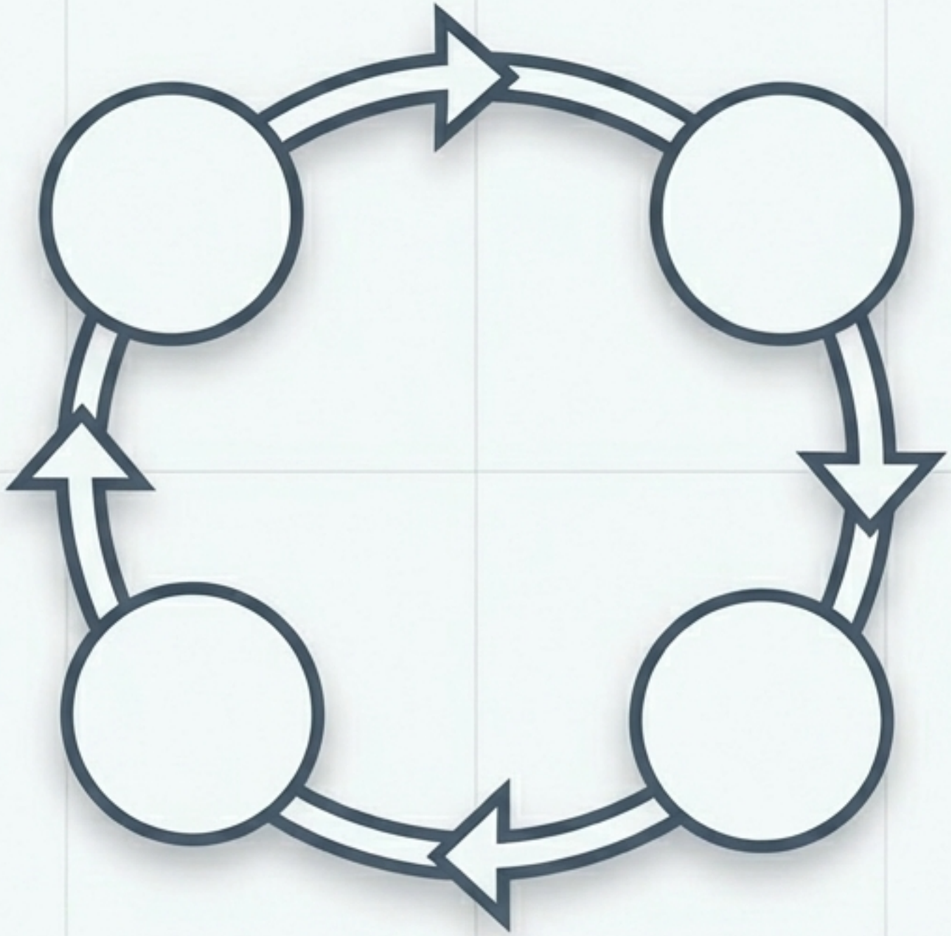
Set `MCP_OAUTH_PRIVATE_KEY=disabled` to prevent silent RSA key auto-generation in environment dumps.

Hook Registration

Scripts in `~/.claude/hooks/` will never fire unless explicitly mapped in `settings.json` under the hooks array.

The Complete Continuous Automation Stack

1. Work
User writes code/encounters bugs.



2. Capture
Global Rule continuously triggers `memory_store` in the background. Auto-approve ensures zero interruptions.

3. Observe
`PostToolUse` hooks silently log all raw activity to local JSONL for audit.

4. Restore
Next session starts. Global Rule triggers `memory_search`, restoring context in <400ms.

Context persists across sessions automatically.
No manual saves. No exit-time scripts. No discipline required.