

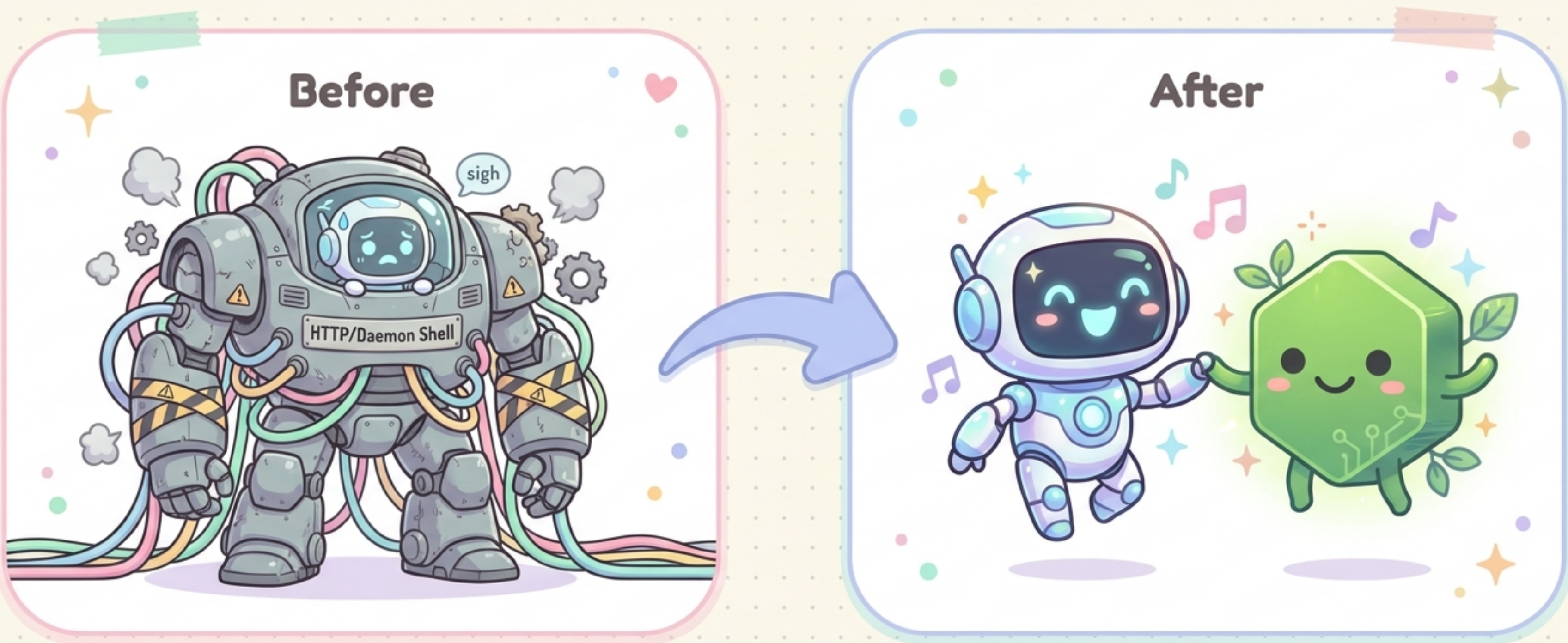
Traveling Light with **cmem-sdk**

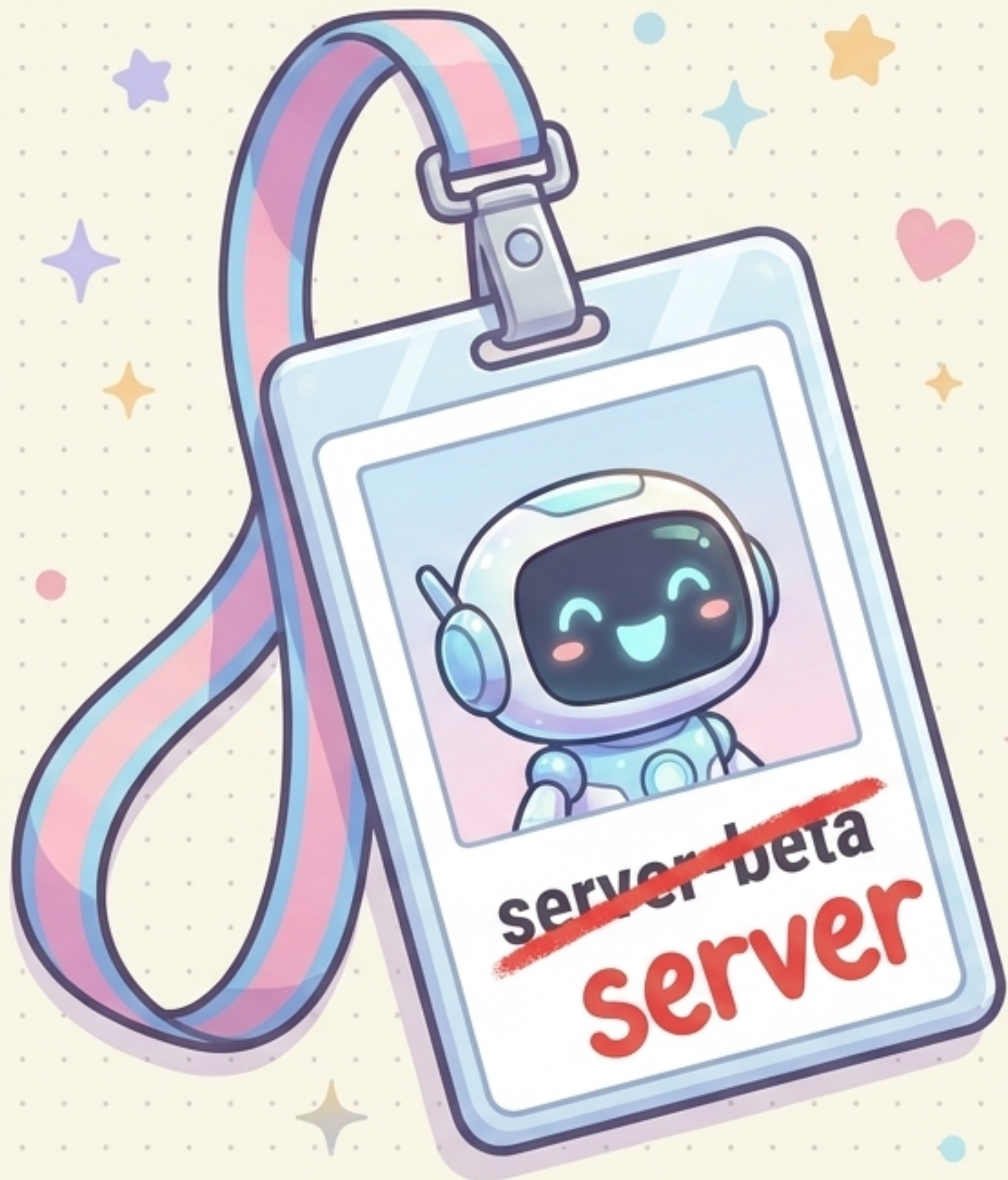
Embeddable claude-mem I/O on the Server Runtime



Not a New System, Just a Lighter One

The cmem-sdk is the exact same in-process server runtime you already trust. We haven't rewritten the core—we've simply removed the heavy daemon shell so it can be imported directly as a library.








The Great Rename

We are dropping the beta tag. The core architecture is stable, and the literal string `server-beta` was causing silent fallback regressions.

- Hooks now accept 'server' as the canonical runtime.
- Code symbols transition from `ServerBeta*` to `Server*` (e.g., `createServerService`).
- Back-compat warning: Persisted values like the schema migrations table and `job_type` enums dual-accept the old strings to prevent breaking existing databases.

The Heavy vs. Light Companion

	Legacy Worker Runtime	New cmem-sdk
	Process Model: Daemonized HTTP worker	Process Model: In-process library
	Dependencies: Requires bun:sqlite, Redis, Express	Dependencies: Pure Node, zero external process dependencies (except Postgres)
	Generation Engine: Sub-process / worker queues	Generation Engine: Inline, direct fetch

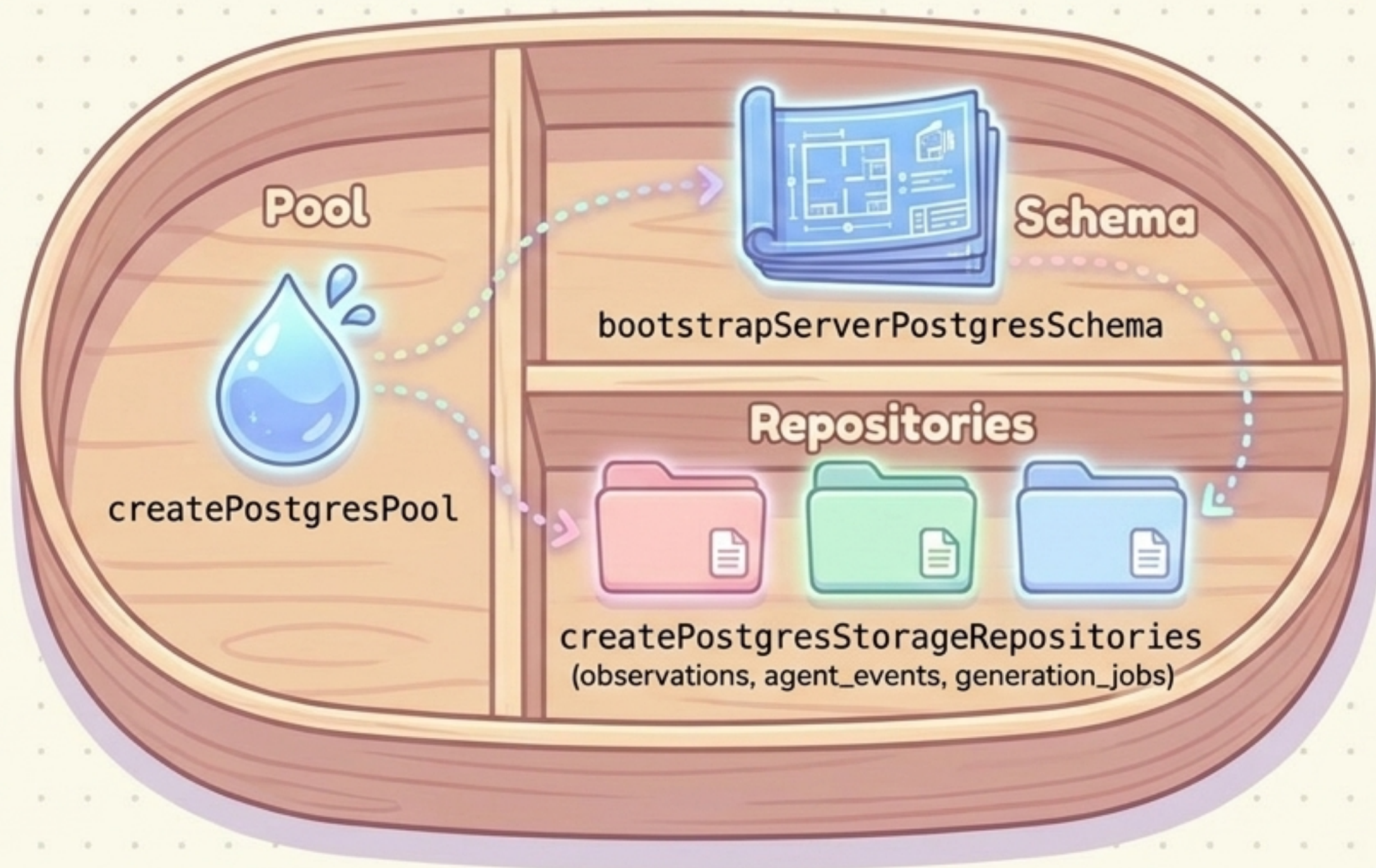
Leaving the Heavy Luggage Behind

To fit inside your application, Cmem must travel light. The SDK build explicitly guards against pulling in the old shell.



Packing the Bento Box

The SDK reproduces the internal graph, not the HTTP service.





Meeting the Elephant

The SDK connects directly via `CLAUDE_MEM_SERVER_DATABASE_URL` (or an injected pool).

It automatically runs `await bootstrapServerPostgresSchema(pool)`. This is strictly in-process and idempotent—running it twice does nothing, ensuring safe boot-ups every time.

The Tenancy Tickets

Unlike SQLite, Postgres requires a `teamId` and `projectId` on every call. Crucially, the `ProjectsRepository` has no lookup-by-name (`getByIdForTeam` only).



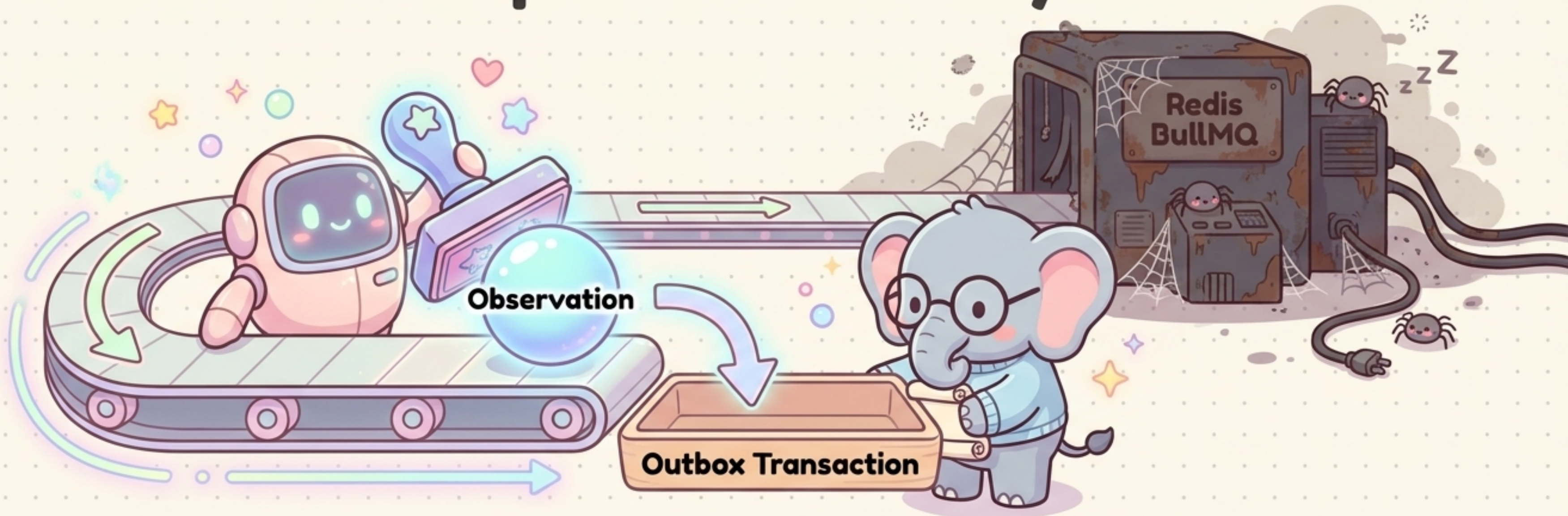
The Rule

Production consumers must pass explicit IDs to access their data.

The Fallback

If omitted, the SDK runs `ensureDefaults()` once to create a default project, persisting the IDs to a local `$CLAUDE_MEM_DATA_DIR/sdk-tenant.json` file for seamless reuse.

The Capture Assembly Line



Ingestion

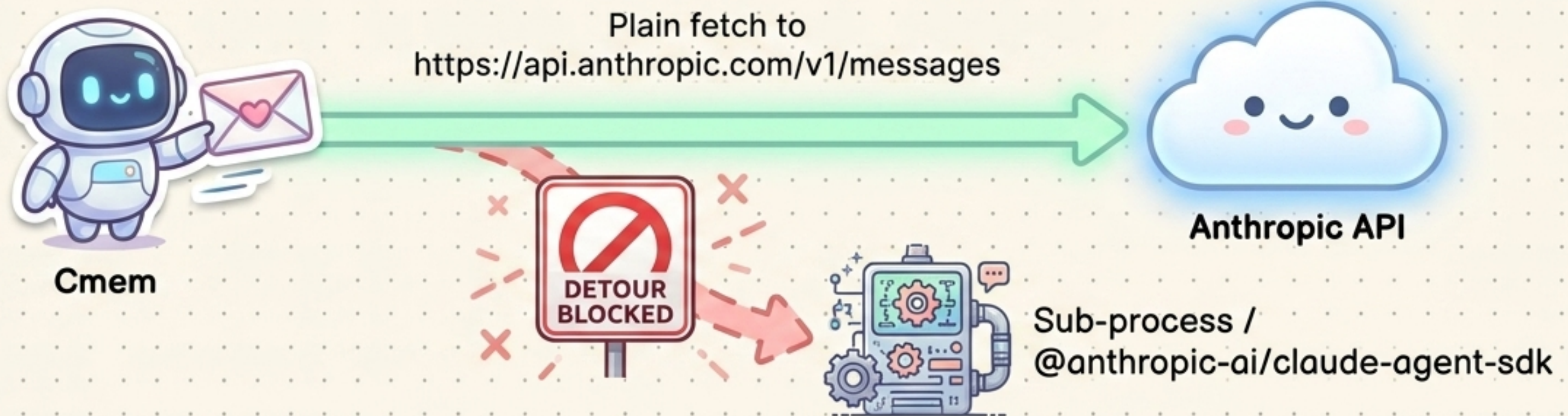
Ingestion writes the `agent_event` and a queued `generation_job` outbox row in one single transaction via `ingestOne({ generate: false })`.

The Secret:

The Secret: We bypass the queue entirely by setting `resolveEventQueue: () => null`. The BullMQ enqueue simply becomes a no-op (`enqueueState='queued_only'`).

The Inline Generation Engine

The ClaudeObservationProvider handles generation entirely in-process.



Key Constraints:

- Uses a plain fetch.
- NO @anthropic-ai/claude-agent-sdk dependency.
- NO external sub-processes spawned.

The Rules of the Lockbox

The generation transitionStatus function enforces a strict state machine mimicking the legacy outbox.



The Warning: You cannot transition from queued directly to completed. It will throw an error at `generation-jobs.ts:390`. The SDK must manually transition to processing first.

Parsing the XML

The `parseAgentXml` function is strictly required to structure the raw LLM response into useful metadata objects.

`<concept>Data</concept>`

`<fact>Value</fact>`

`<meta>Info</meta>`

`<chunk>Raw</chunk>`

```
{  
  "concept": "Data",  
  "fact": "Value",  
  "meta": "Info"  
}
```

```
{  
  "type": "chunk",  
  "content": "Raw"  
}
```

ModeManager

The Catch:

The parser will instantly throw an error (`parser.ts:105`) without an active mode. The SDK automatically ensures **ModeManager** is initialized with the default mode at client construction to prevent this.



The Search Acrobats

Search requires two buddies working perfectly together.



Semantic Search

Delivered entirely by the existing local Chroma engine.



Full-Text Search

Native FTS operations handled cleanly inside the database.



The Chameleon's Glue

The Chroma to Postgres glue is the only genuinely new code, and it is kept incredibly minimal by using the storage-agnostic document layer.



- **Implementation:** We build a ChromaDocument using the Postgres UUID string and pass it to the storage-agnostic chroma_add_documents layer.
- **Rule:** We do NOT reuse sync0bservation, as it is hardcoded strictly for legacy SQLite integers.

The Search Safety Net

If Chroma is disabled, or if the chroma-mcp subprocess slips and fails mid-air, search automatically and safely falls back into the **Postgres FTS** net.

The Golden Rule: Postgres semantic search is entirely out of scope. Do not add pgvector or a vector column to the schema.

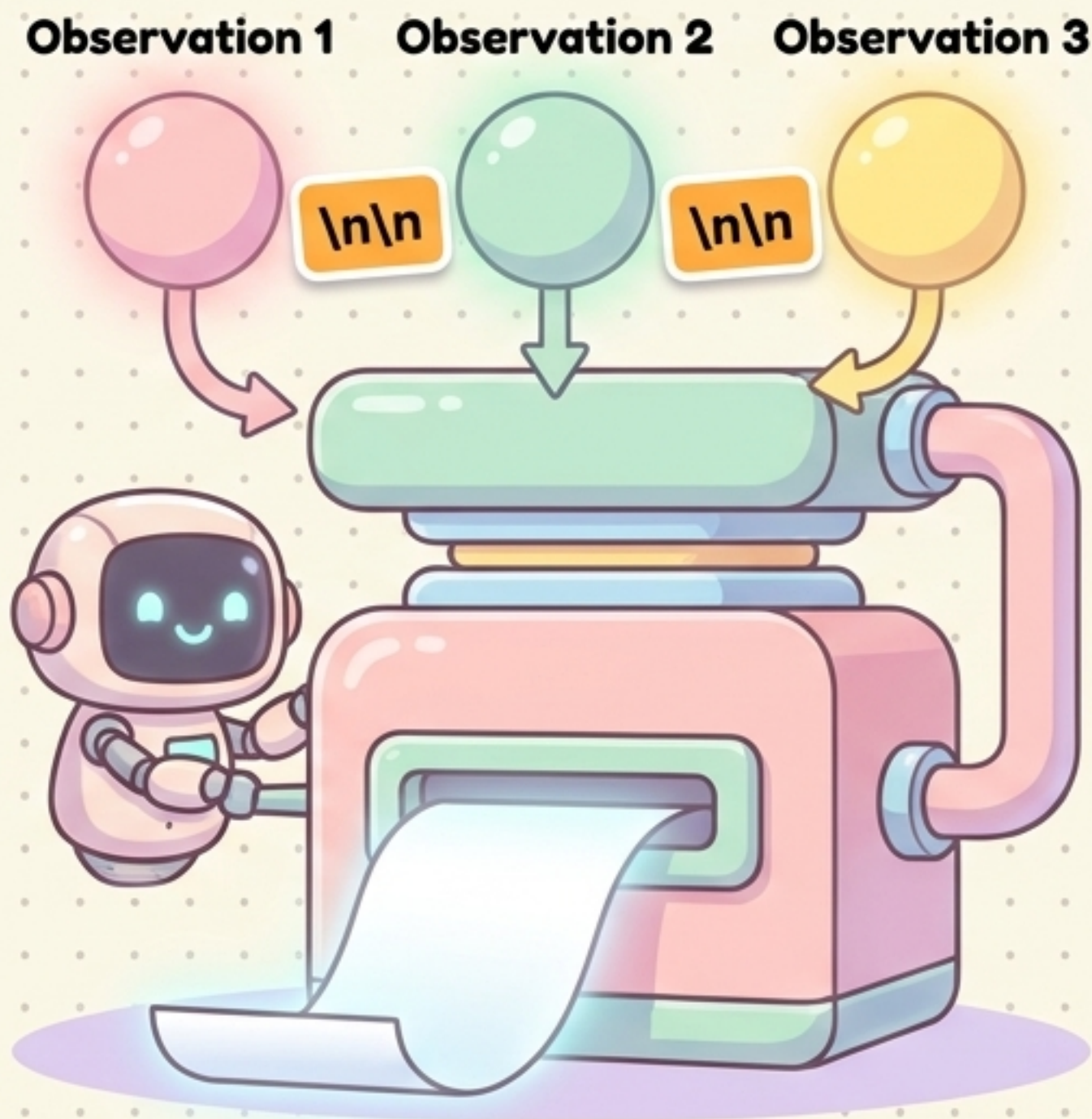
Chroma-MCP
Subprocess

Postgres FTS
(websearch_to_tsquery)

**NO VECTOR SEARCH
PERMITTED**

Compiling the Context

The **client.context({ query, limit })** method acts as the final assembler, building the **string payload** specifically formatted for the LLM prompt.



The context wrapper handles the branch logic, hydrates the Postgres rows from Chroma UUIDs, and seamlessly joins the resulting **.content** strings with line breaks.

The Final Package

CmemClient ties the entire architecture together into a stable, minimal, and deeply satisfying public API.



```
capture() / captureBatch()  
generate() / captureAndGenerate()  
search() / context()  
startSession() / endSession() / close()
```



The Worker-Free Promise

The ultimate proof of the SDK architecture lives in `examples/sdk-node`.



- ☒ No HTTP Server
- ☒ No Pidfile
- ☒ No Daemon

The Result: A plain Node script that imports `claude-mem/sdk`, points at the database URL, and successfully runs full capture, compression, and semantic search—with absolutely no worker process alive.