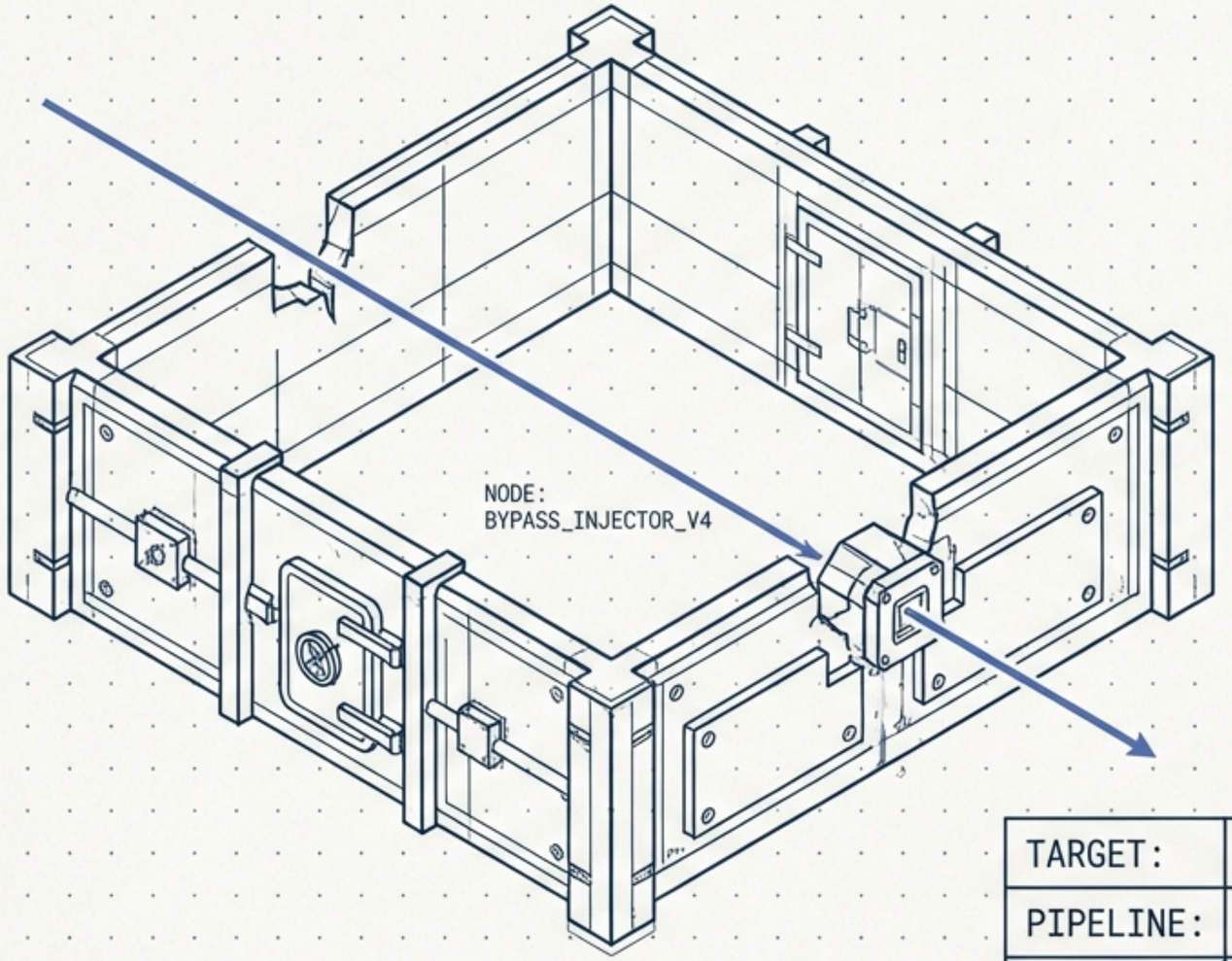
CRACKING THE WALLED GARDEN ON macOS

A FIELD GUIDE TO FULL WECHAT AUTOMATION

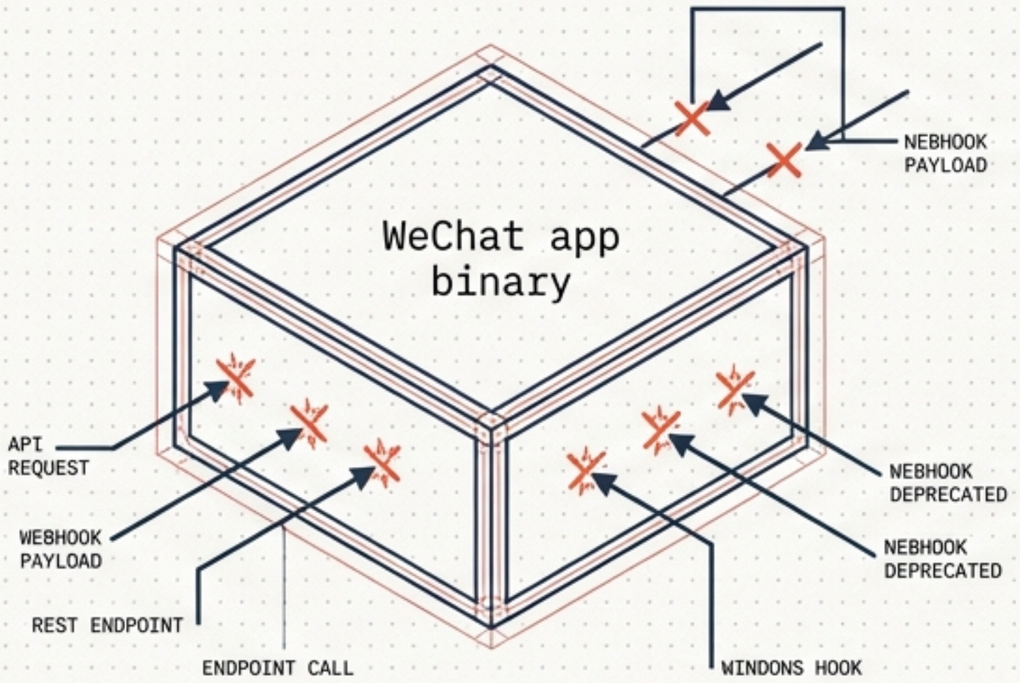


TARGET:	WeChat native macOS
PIPELINE:	Hybrid **Read**+**Write** →
STATUS:	Deployed

THE FUNDAMENTAL FRICTION OF WALLED GARDENS

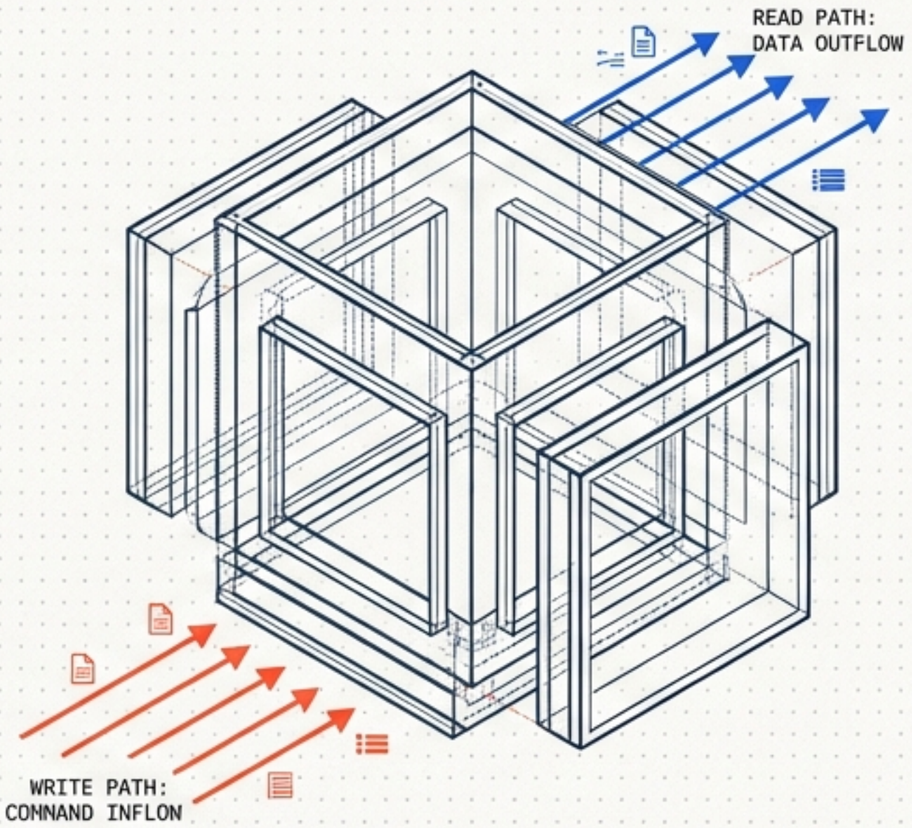
CONSTRAINTS

Zero API access.
No webhooks.
No REST endpoints.
Web version deprecated.
Windows hooks incompatible.



TARGET STATE

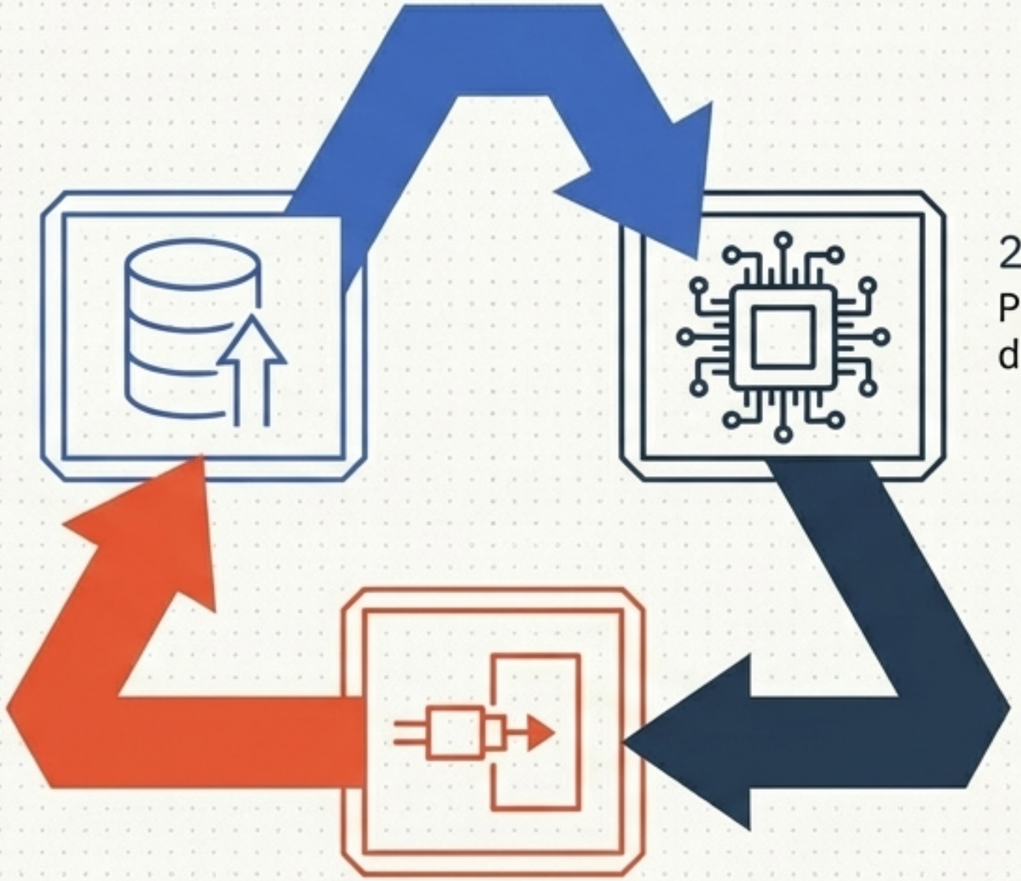
Full native macOS programmatic access.
No sandbox, no restrictions.



THE AUTOMATION TRIAD DRIVES THE AI WORKFLOW

1. EXTRACT CONTEXT

Search & summarize high-volume group history



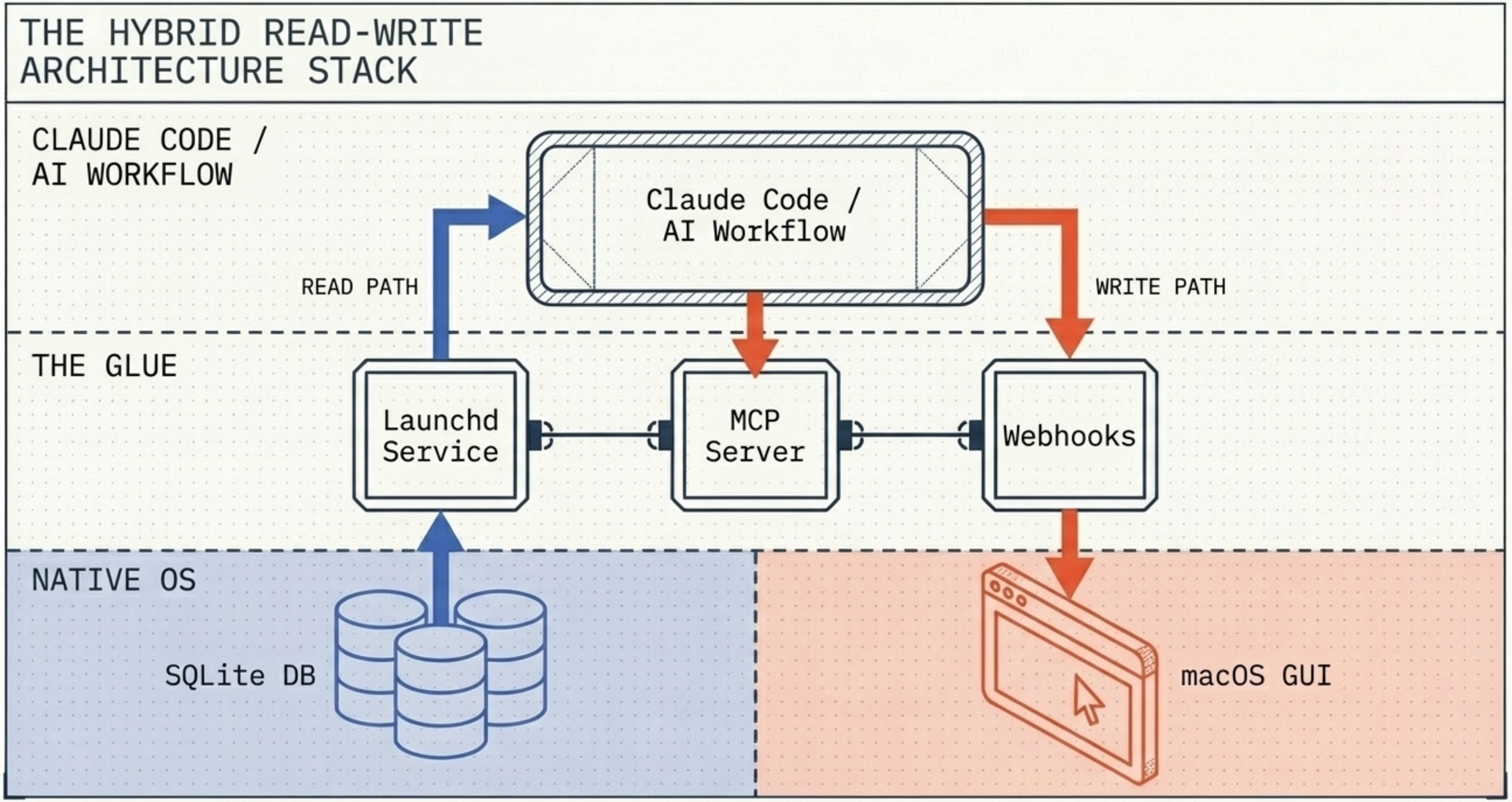
2. AI PROCESSING

Pipe localized context directly into Claude

3. ACTIONABLE OUTPUT

Send outbound messages programmatically

THE HYBRID READ-WRITE ARCHITECTURE STACK

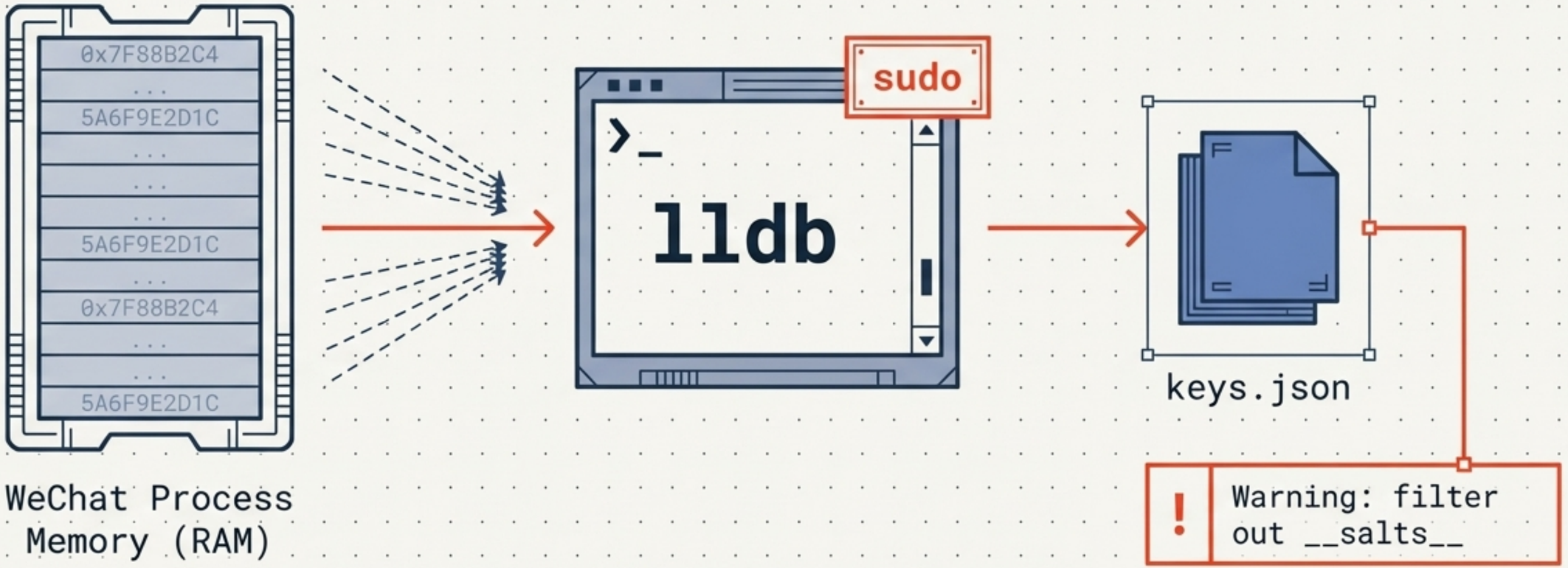


TWO DISTINCT ACCESS PARADIGMS

READ PATH	WRITE PATH
Memory / SQLite Decryption	macOS Accessibility (AXUIElement)
 Instant (Local API)	 Seconds (GUI Automation)
 Background daemon	 Foreground UI manipulation
Full historical access	Active chat session only

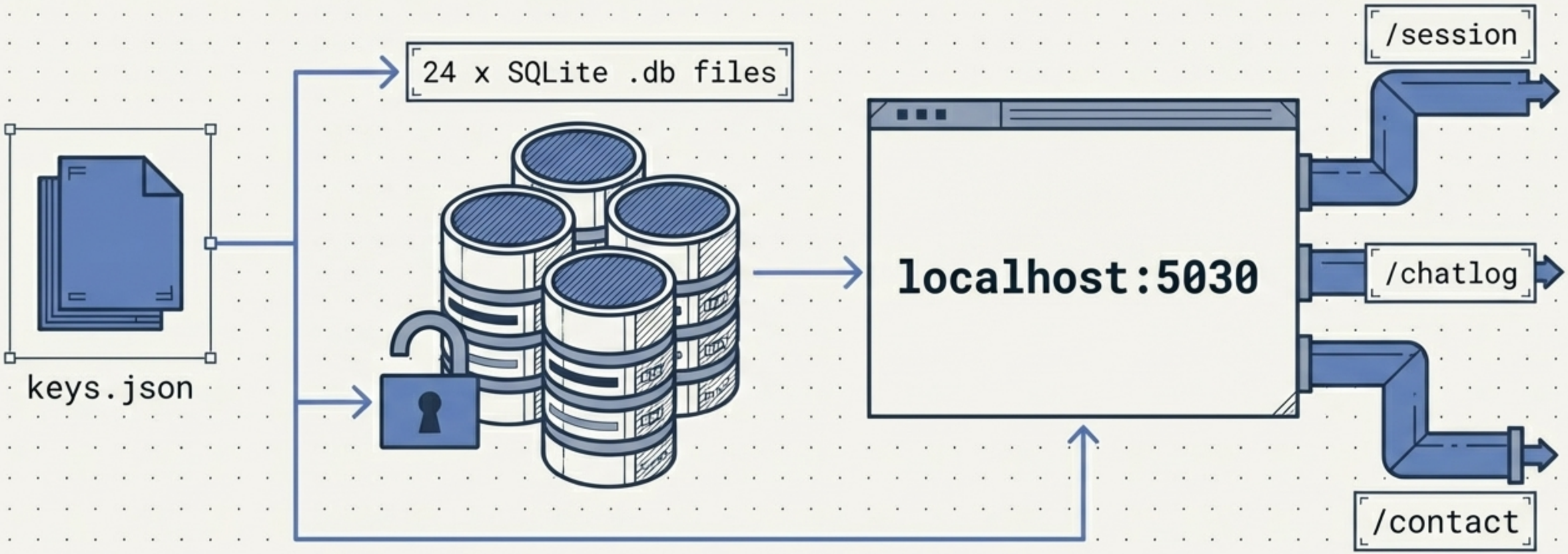
BYPASSING SQLCIPHER VIA MEMORY EXTRACTION

Keys live in the active process memory. The **wechat-db-decrypt-macos** tool utilizes **lldb** to scan the application's RAM layout.



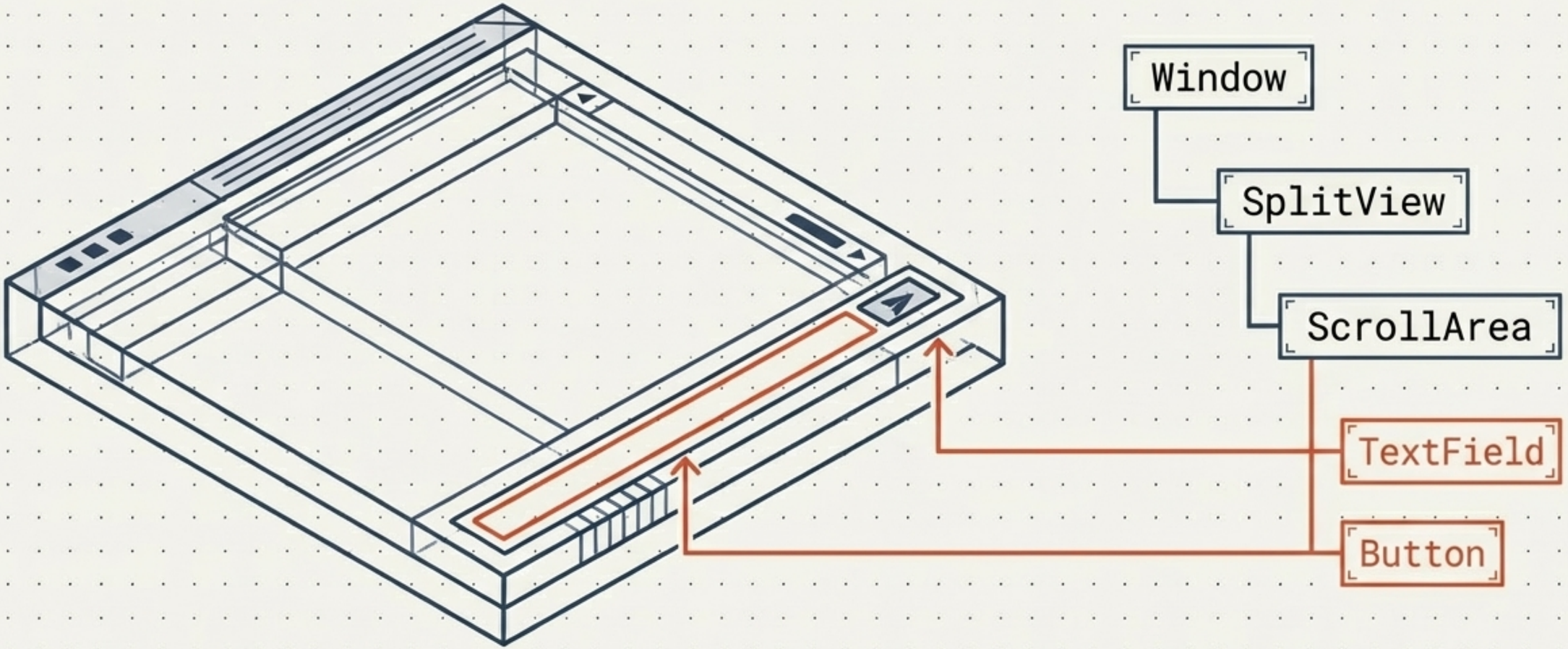
SERVING DATABASE HISTORY THROUGH A LOCAL REST API

The **chatlog-bot** fork maps keys to 24 separate **.db** files, utilizing **fsnotify** for real-time file watching.



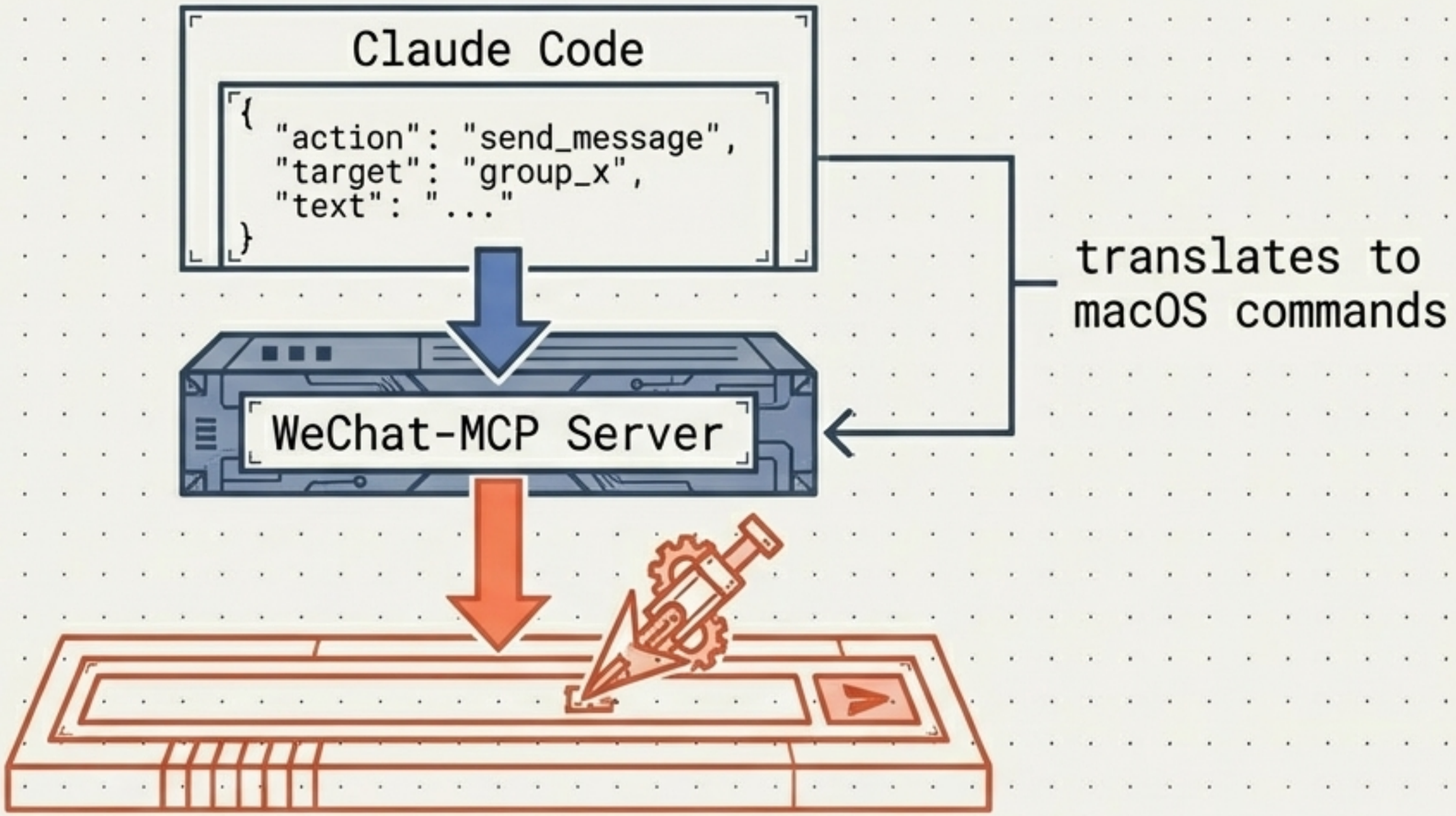
EXECUTING WRITES THROUGH THE macOS ACCESSIBILITY TREE

With no outbound API, writes require automating the GUI. The target is navigated entirely via the **AXUIElement** framework.



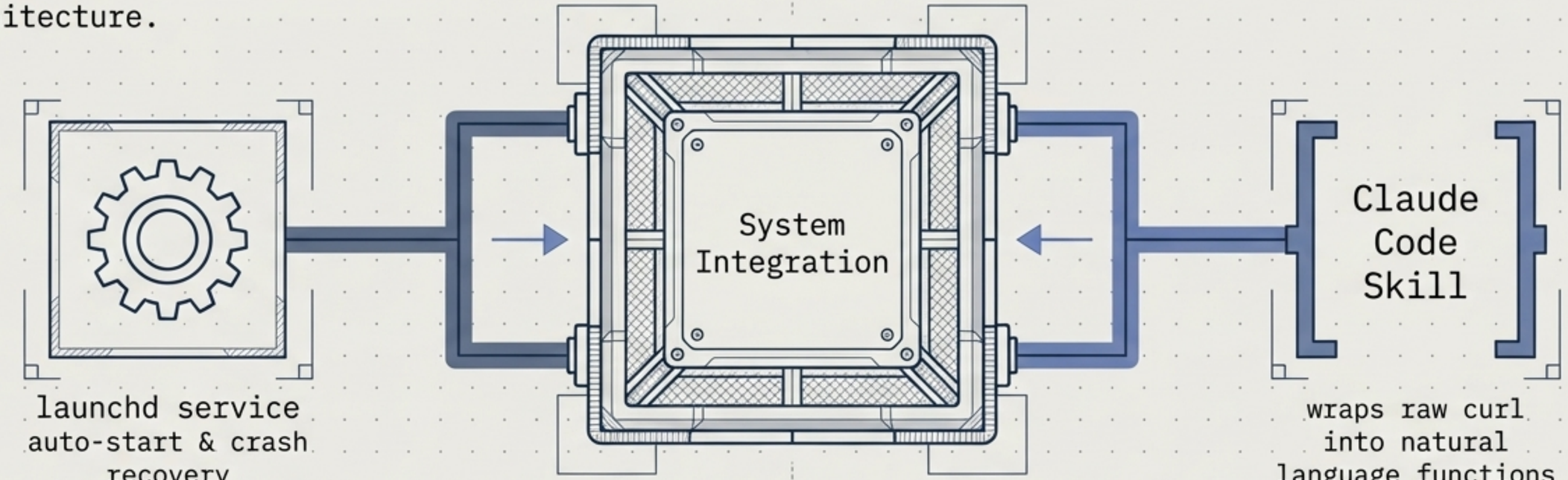
BRIDGING THE ACCESSIBILITY TREE TO CLAUDE MCP

WeChat-MCP registers as a Claude Code tool. A local tool call physically types into the active window.



THE INTEGRATION LAYER ENSURES DAEMONIZED PERSISTENCE

Tying independent paths into a resilient wechat-mac-reader architecture.

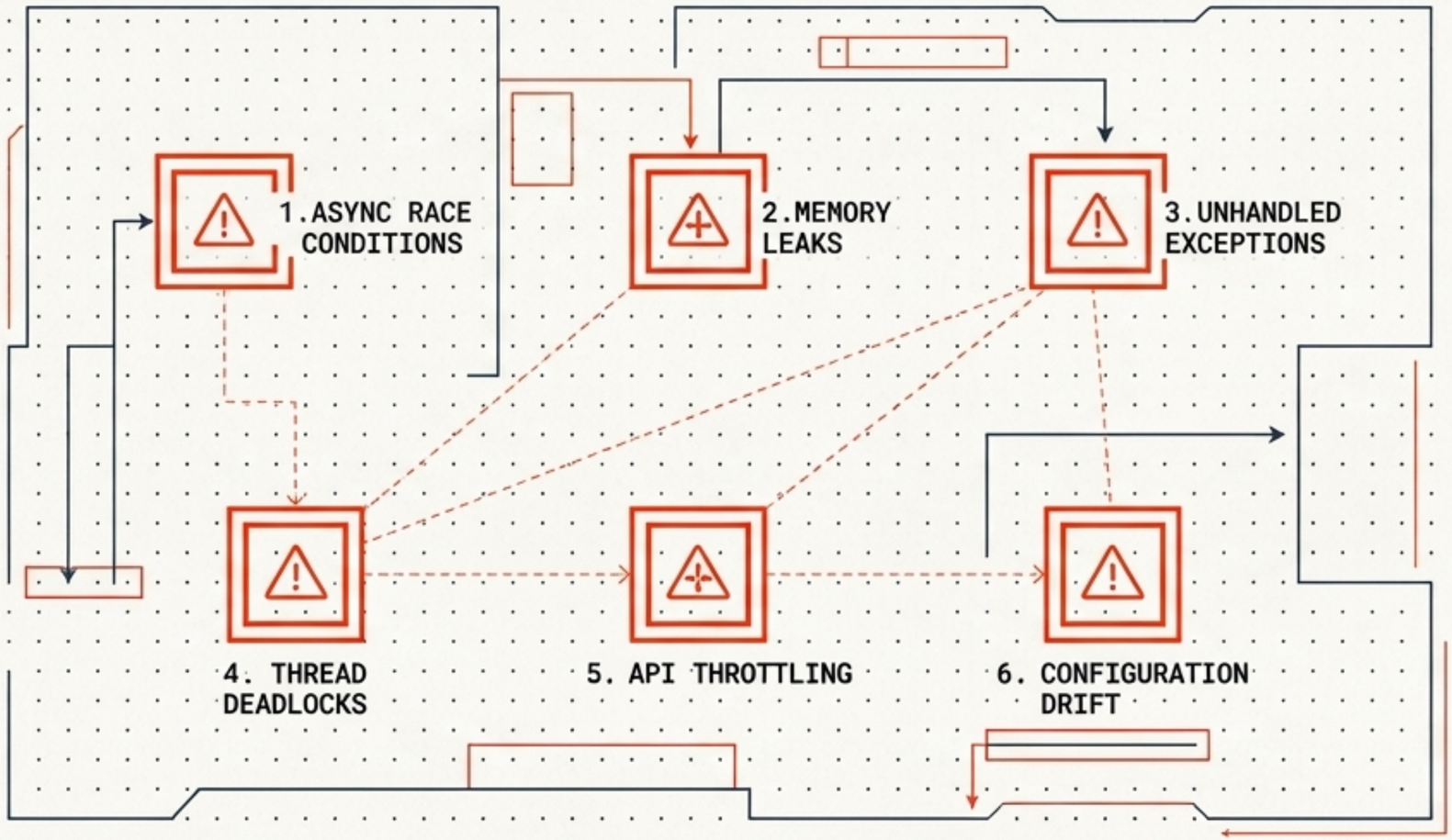


launchd service
auto-start & crash recovery

wraps raw curl into natural language functions

NAVIGATING THE MINEFIELD

The architecture is proven, but the implementation is hostile. Avoid the 6 major pitfalls that cause silent failures and crashes.



DIAGNOSTIC MATRIX I: DATABASE & CONFIG ROUTING

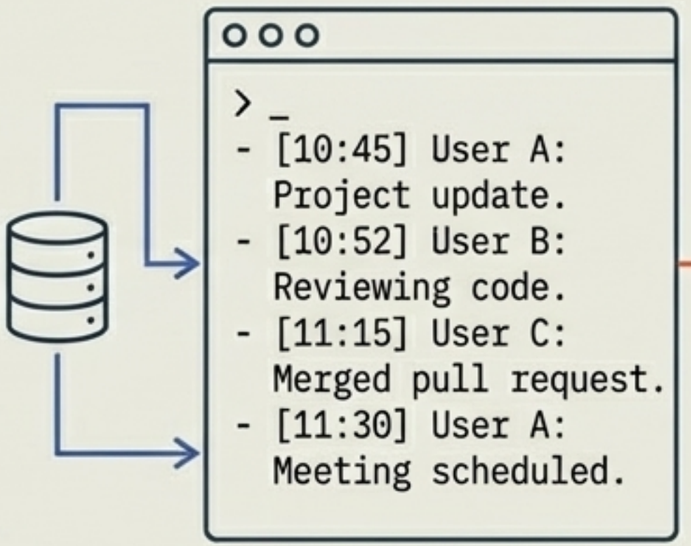
SYMPTOM	CAUSE	SOLUTION
Single master key flag fails.	WeChat 4.1.x utilizes 24 separate per-database keys.	Map exact paths in config.yaml via derived_key_map.
Silent decryption failures (empty API results).	Missing internal path prefix in config.	Prepend db_storage/ to all mapped .db paths.
Config parser chokes on JSON type mismatch.	keys.json includes a list array alongside string mappings.	Filter out the __salts__ entry before ingestion.

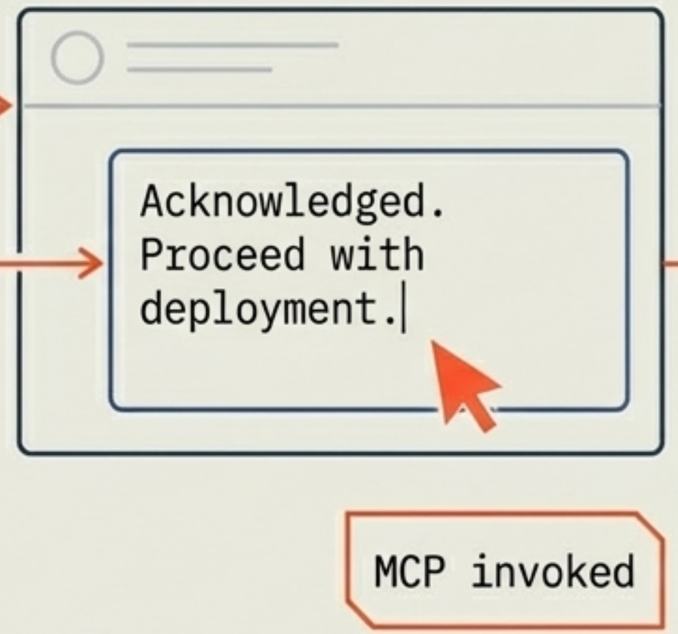
DIAGNOSTIC MATRIX II: PROCESS MEMORY CONSTRAINTS

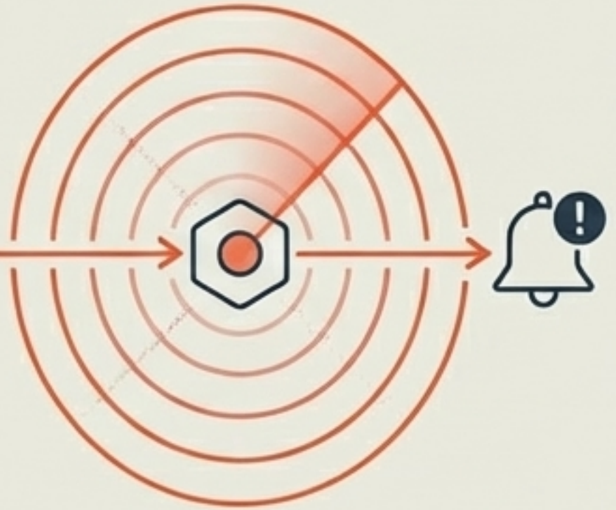
SYMPTOM	CAUSE	SOLUTION
Extraction returns “no memory regions found”.	Built-in Mach VM API (vmmap) fails on 4.1.x architecture.	Force use of lldb memory-scanning tool instead.
Valid queries suddenly return encryption errors.	Keys expire entirely upon application restart/update.	Rerun lldb extraction pipeline; no permanent keys exist.
task_for_pid attachment fails.	macOS restricts active process memory reading. 	Execute natively on host with sudo (Docker is not viable).

THE RESULTING AUTOMATION WORKFLOW

READ	WRITE	TRIGGER
Summarize group while asleep.	Reply to X.	Passive Monitoring







Desktop apps without APIs are not actually locked down.

The data lives on your machine. The UI is exposed to OS accessibility frameworks. With the right tooling, any application can become an automation layer.