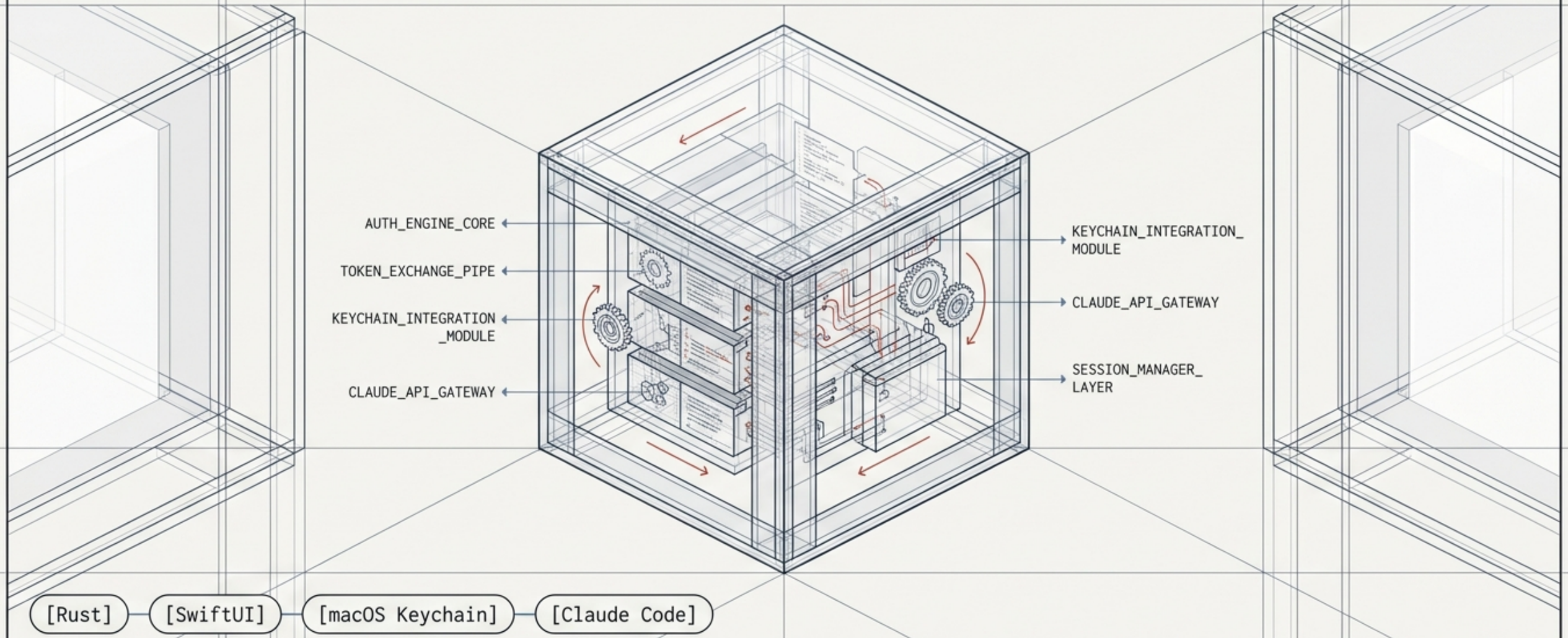
Hot-swapping Claude logins in real-time

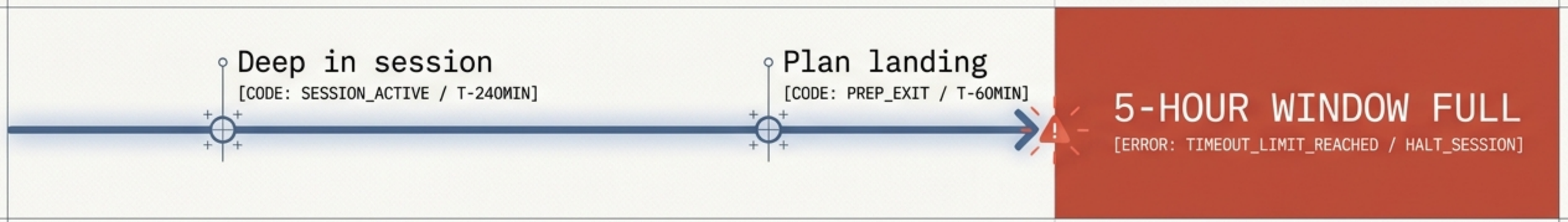
Teaching AI accounts to rotate themselves—and discovering the absolute boundary between human engineering and AI code generation.



[Rust] [SwiftUI] [macOS Keychain] [Claude Code]

The five-hour usage window creates a hard wall mid-session

THE TIMELINE



THE COST ANALYSIS MATRIX

The Obvious Fix

Switch to an account with headroom (Work Max / Personal Max).

[PATH: SYSTEM_SETTINGS > USERS > SWITCH_ACCOUNT]

The macOS Reality

Switching strictly requires a complete manual logout.

[SYSTEM_CMD: LOGOUT_REQUIRED / FORCE_QUIT_APPS]

The Consequence

Re-authenticating drops the running session. **Context is instantly lost.**

⚠ [ERROR: SESSION_TERNINATED / DATA_LOSS_IMMINENT]

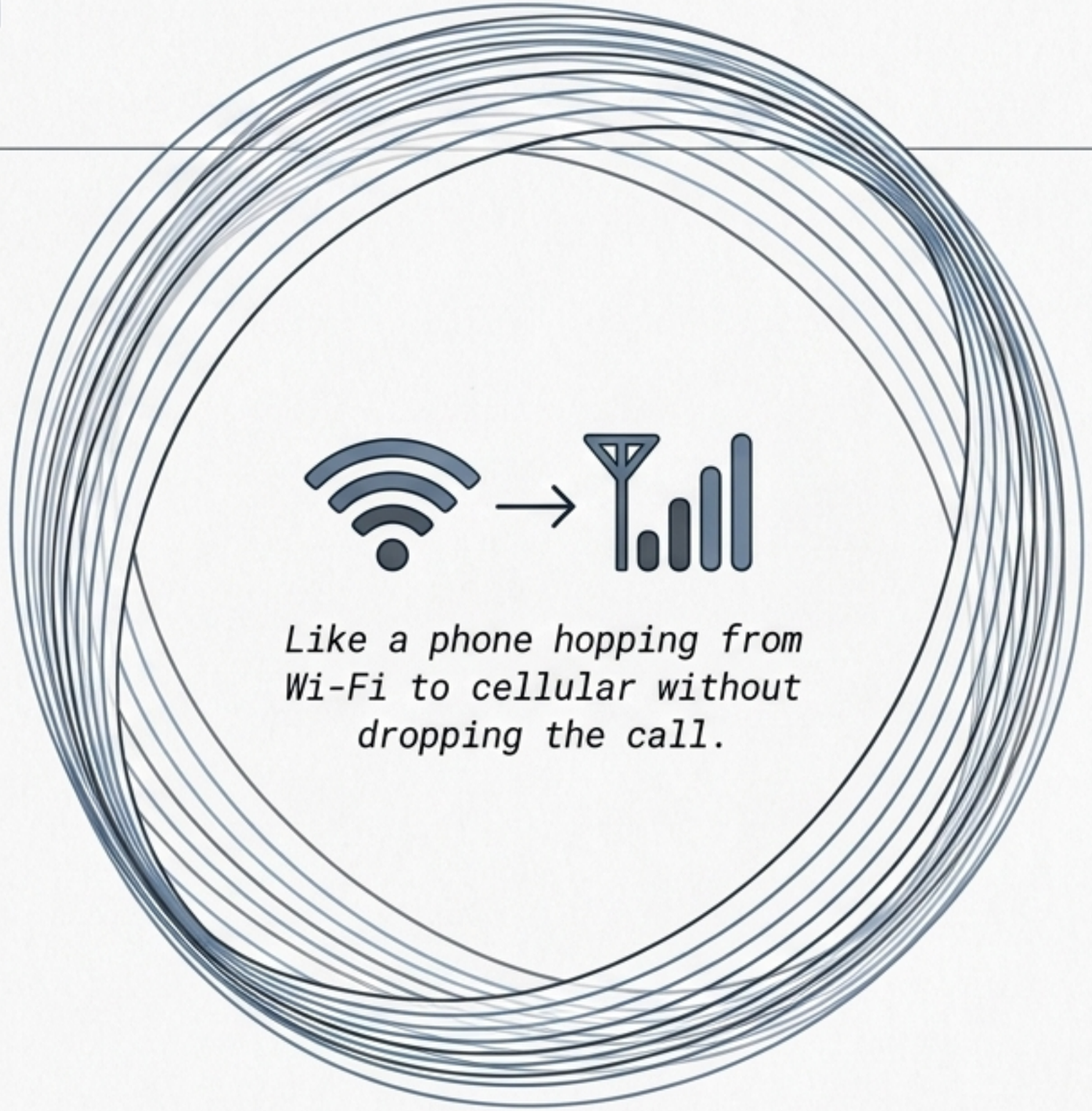
Designing the boring version of account management

Core Objective

The active account must rotate itself to the next one with headroom before hitting the wall, while typing continues.

Design Constraints

1. No terminal prompts
2. No manual logouts
3. No user awareness required



Like a phone hopping from Wi-Fi to cellular without dropping the call.

The primitive: A running process re-reads its login every request

The Assumption

Login

↓

Token saved to `~/.claude/.credentials.json`

↓

Token held in memory

↓

Process dies on switch

Editing the JSON mirror does nothing to a live session.

The Reality

`claude process runs`

↓

Reads Claude Code-credentials from macOS Keychain

Performs read on EVERY single request.

The running process never caches the token permanently.

Rewriting a single Keychain item hot-swaps the underlying account

Security Context

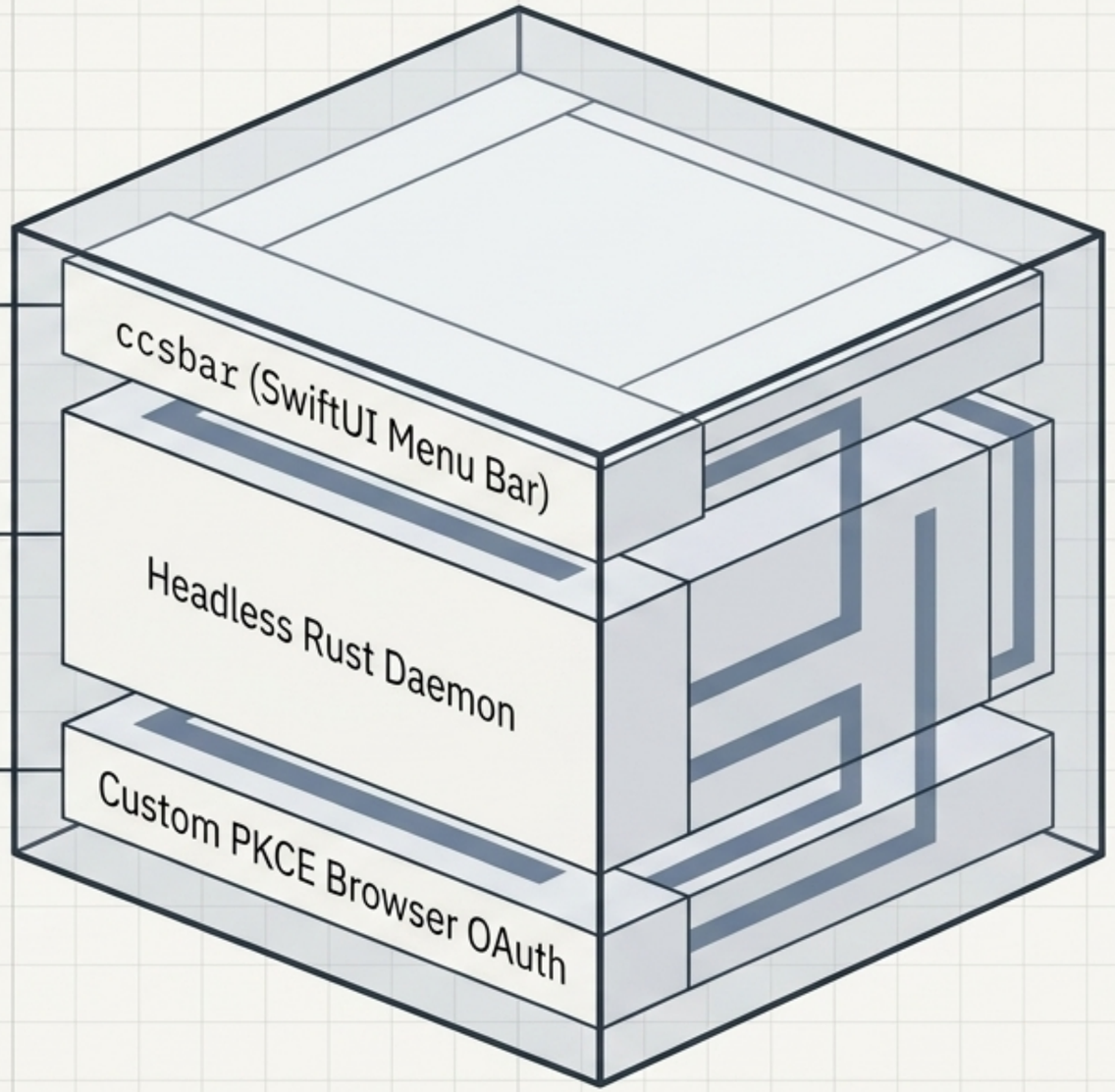
Always Allow Keychain grants are bound to the code signature. Routing writes through Apple's stable stable `/usr/bin/security` ensures the grant survives custom rebuilds.

The plumbing required to automate the hot-swap

Visibility layer. Native UI, reads status, forecasts next moves.

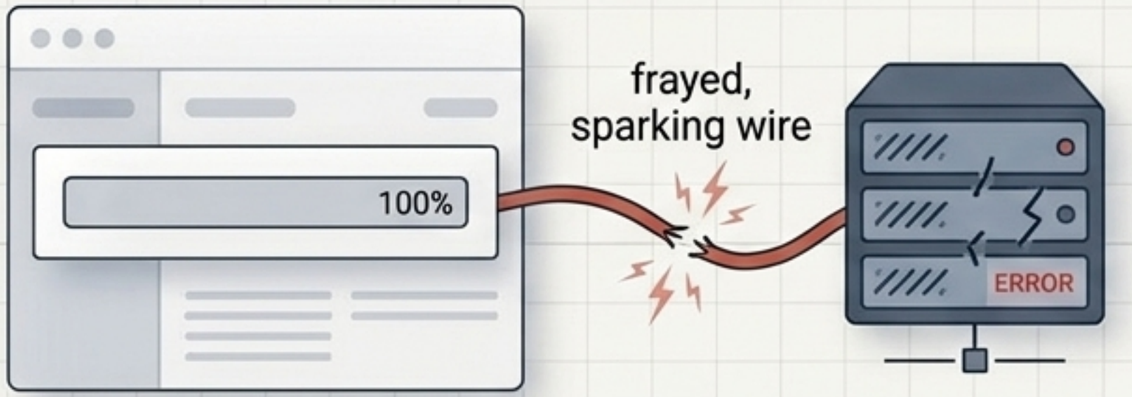
Logic layer. Watches usage, walks fallback chain (triggers at 95%), rewrites Keychain unattended. Publishes `status.json` every second.

Base truth. Self-contained token minter. Bypasses standard directory constraints to write tokens where the fork can access them.



A background daemon’s worst bug is the silent one

The Danger



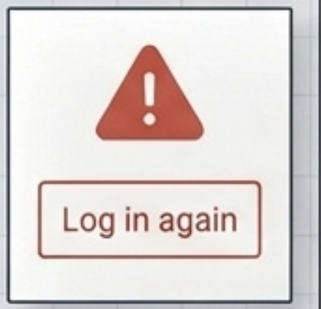
- Daemon dies -> Usage numbers freeze.
- Auto-switch fires at 2am -> Active account is unknown.
- Login expires -> Daemon cheerfully serves stale numbers.

The Design Discipline

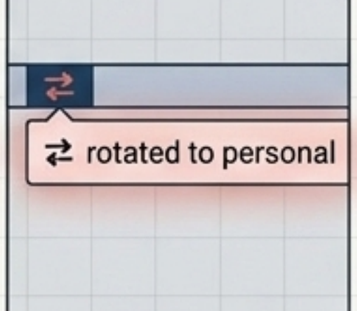
Every silent state the daemon can have gets a **loud, fixed home.**

1. Dead Daemon:

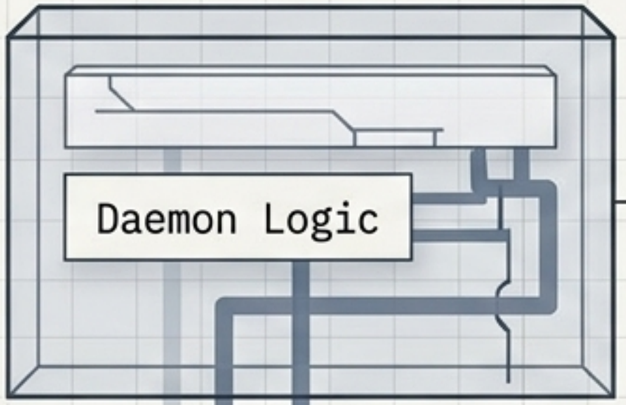
2. Dropped Login:



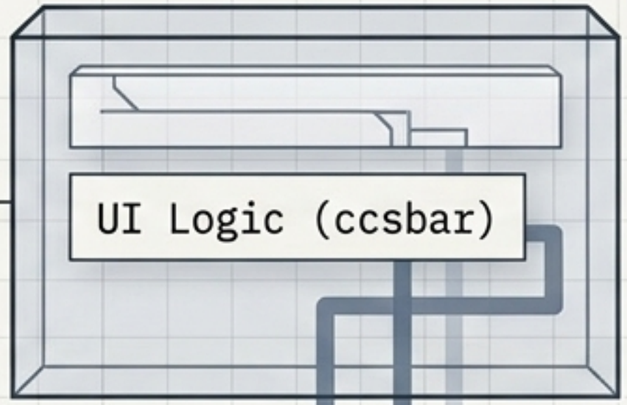
3. Unattended Rotation:



Using pure functions to prevent UI state drift



The Challenge: The menu bar needs to forecast the daemon's next move without guessing.



⇄ Executes chain-walk computation ⇄

⇄ Mirrors chain-walk line-for-line as a pure function ⇄

Mathematical Certainty

The UI prediction ('would switch to personal at 95%') cannot geometrically drift from what the backend will execute.

The bugs you only find by running the code

Test Suite Passed

- ✓ `login.verify()`
- ✓ `session.validate()`
- ✓ `daemon.status()`

Dropped-login detection shipped and passed all fixtures perfectly.

```
Not logged in · Please run /login
```

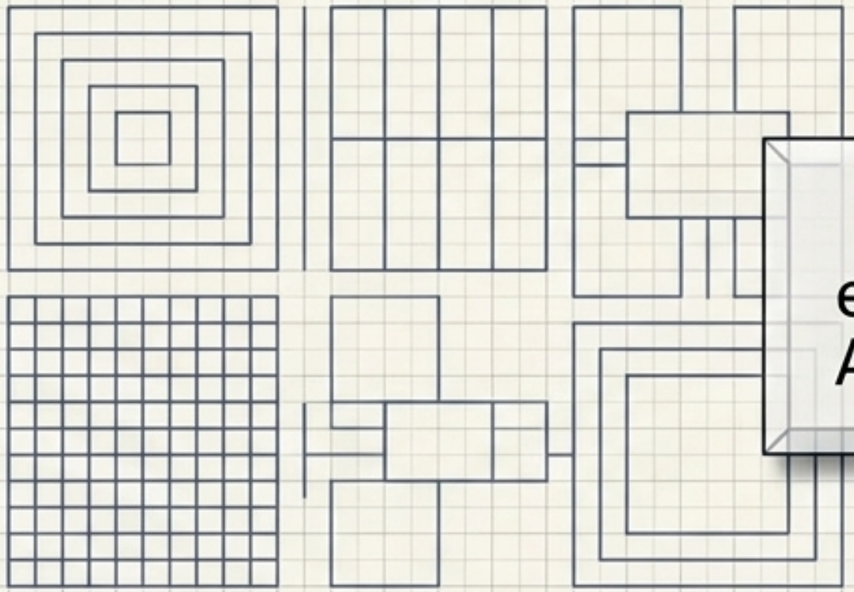
In the very first live run, a session failed on an account the daemon reported as perfectly healthy. Three distinct bugs perfectly stacked to hide one another.

Deconstructing the stacked failure

	Surface (UI)	Below Water (System Truth)	The Fix
The Mask	API returns 429 (Endpoint busy, try later).	Not a rate limit. A dead token with past `clock expiry` wearing a rate-limit costume.	Check `clock expiry` alongside 429s.
The Tooling	‘Log in again’ button fails immediately.	CLI tool `clauth` was built to create profiles, refusing to re-authenticate.	Build a custom `re-authentication` flow.
The Final Boss	Reauth succeeds, JSON `files` files are pristine, session remains broken.	Tokens wrote to `.credentials.json` but skipped the macOS Keychain. Session read dead token.	Force a Keychain write on the active account.

Locating the actual boundary between AI and engineer

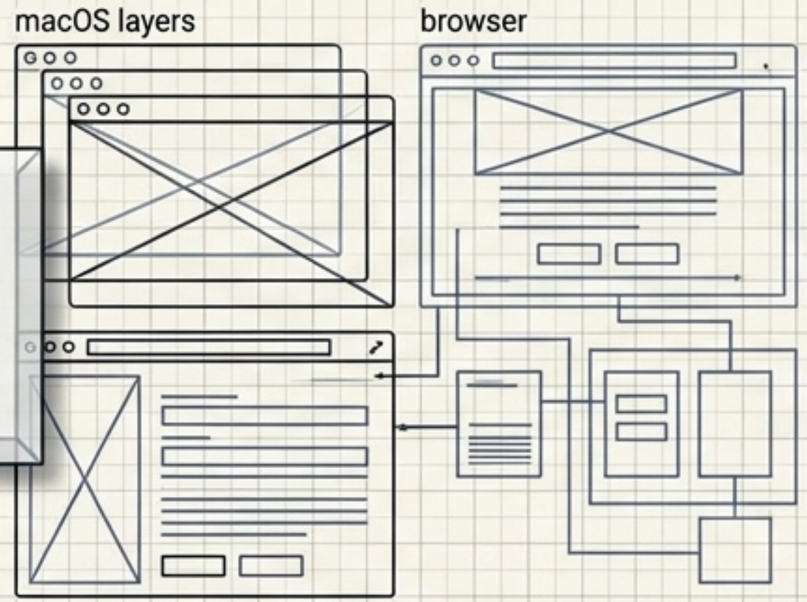
The AI



Claude Code seamlessly wrote thousands of lines: concurrency, pure engines, native UI.

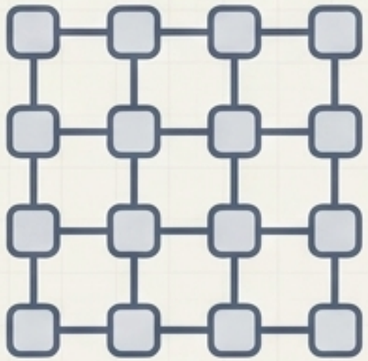
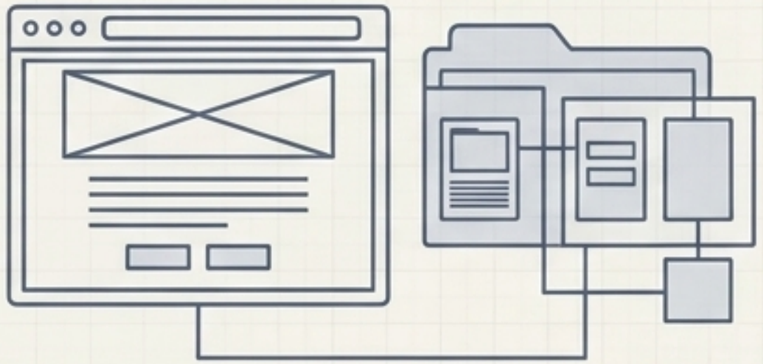
The line is drawn exactly at truth the AI cannot observe.

The Engineer



The boundary isn't 'hard vs. easy' code.

The empirical division of labor

	What the AI Owns (Verifiable Logic)	What the Human Owns (Unobservable Truth)
		
Criteria	Machine-checkable definition of done.	Truth that lives entirely in the physical or running system.
Examples	Pure functions, concurrency safety, socket command round-trips, UI generation.	Undocumented Keychain behaviors, live multi-session states, browser physical clicks.
Outcome	Agent drives to full completion.	Human stays in loop; verification requires running the live experiment.

Building tools that successfully disappear

The system rotates accounts autonomously based on physical usage truths.

The ultimate success state for system infrastructure is complete invisibility. The only time it should be noticed is when it proves it worked exactly as designed—mid-sentence, without ever dropping the connection.