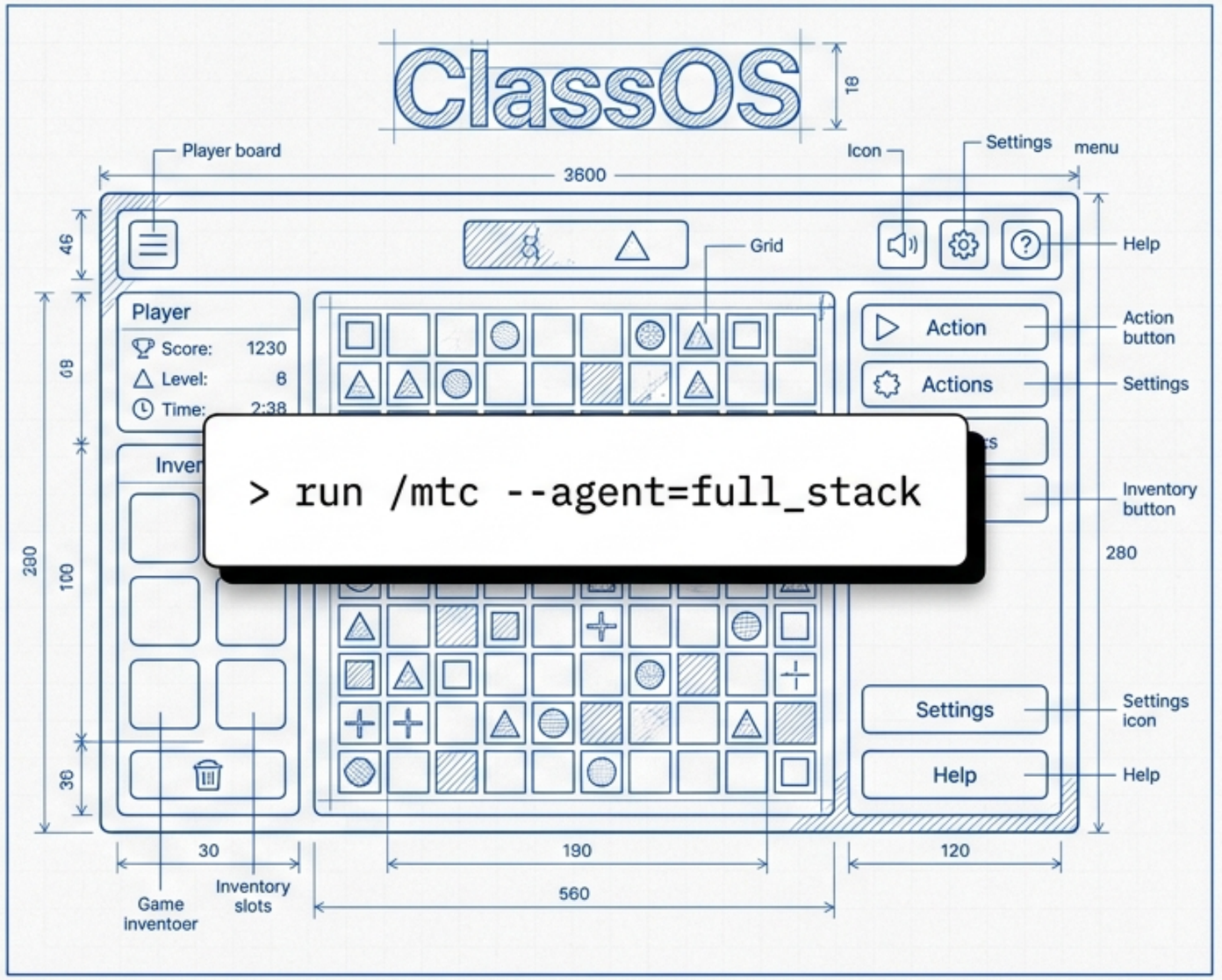
Zero handwritten code. Two hours. One complete game.

ByteDance's Trae IDE introduced MTC (More Than Coding).

Developer Peng Chao used it to build a complete puzzle game called ClassOS in two hours.

Zero handwritten code.

`Replicating the MTC workflow in Claude Code changes how we build software.`



The trap of vibe coding is cascading breakage

Humans start with a vague idea and tell the AI to build.

Halfway through, they realize the concept is flawed.

Changing the concept breaks the data.

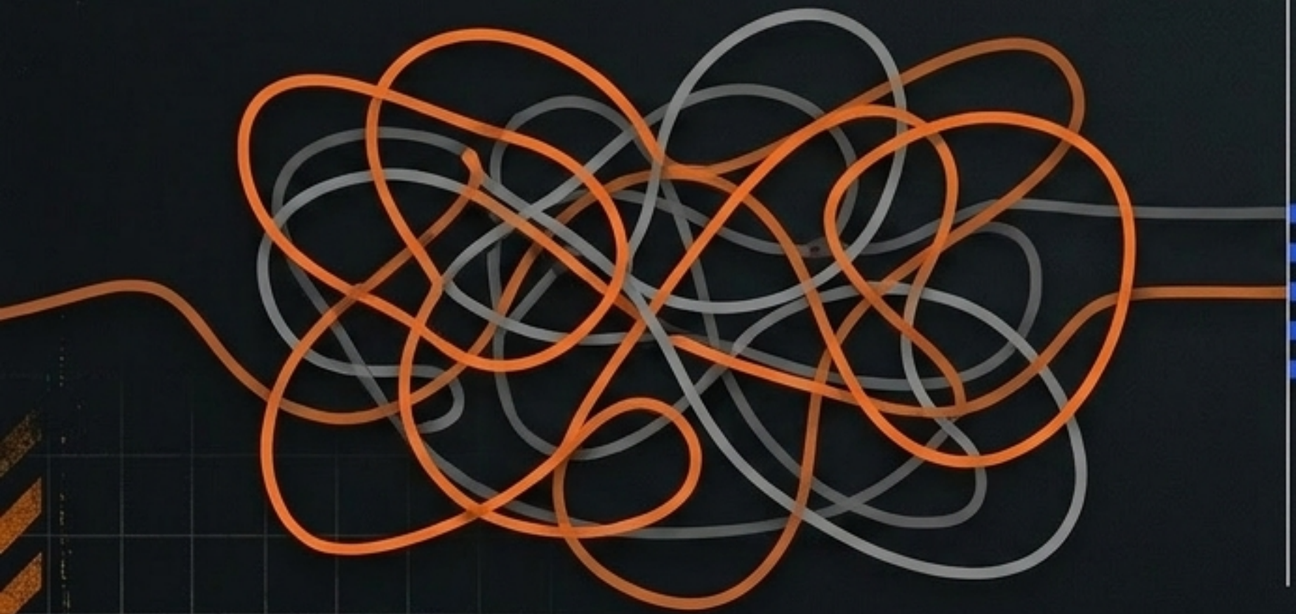
Fixing the data breaks the code.

EVERY PHASE OVERTURNS THE PREVIOUS PHASE. WHAT SHIPS IS A PATCHWORK.

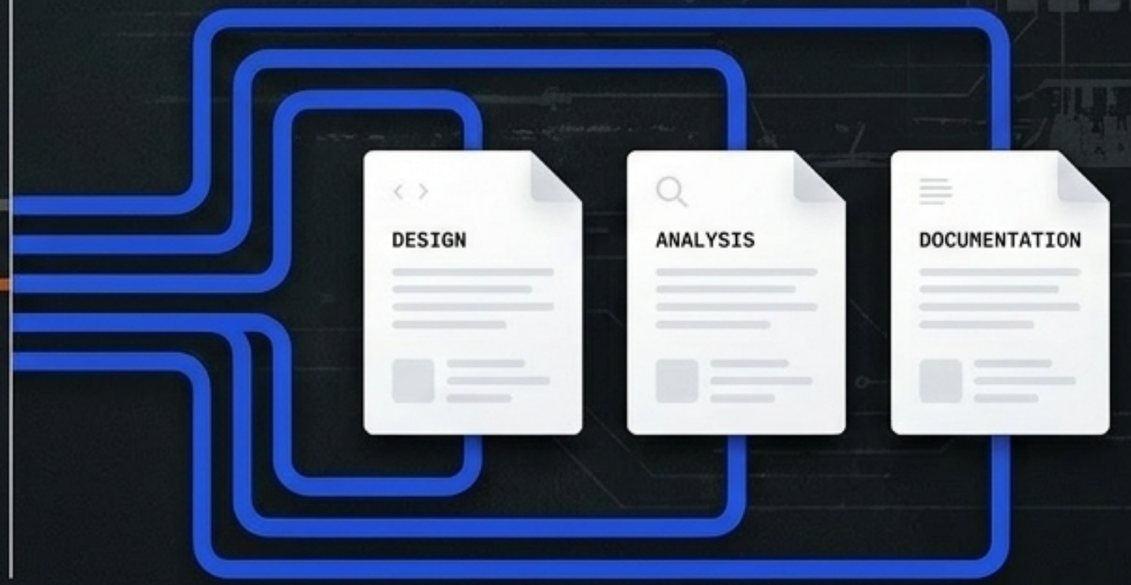
Everything before the code

MTC isn’t a watered-down code mode. It is a full-stack agent workflow handling design, analysis, data generation, and documentation before writing a single line of code.

The Problem



The Solution

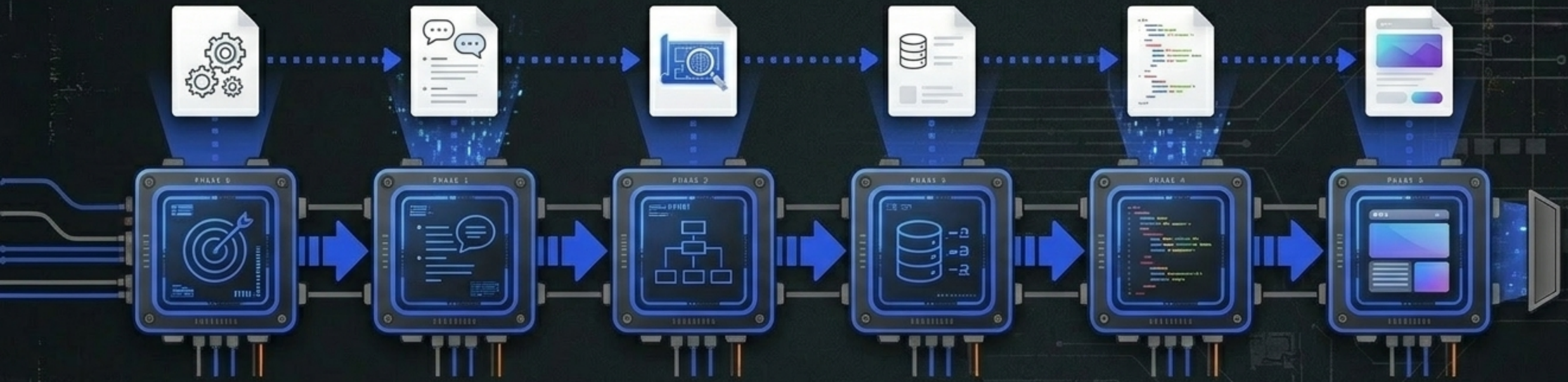


10 minutes of concept work saves 1 hour of rework.

Standard Prompting vs. Structured MTC

Vibe Coding	Structured MTC
Vague idea straight to implementation.	Context alignment and iterative scripts.
Reworking code mid-flight.	Caught in pre-planning documents.
Eager typist.	Forced self-critic and architect.
Skippable, non-linear steps.	Gated pipeline; no phase skipped.
Patchwork logic.	Code guided by a complete design.

The 6-Phase AI Software Factory



Phase 0: Context Gathering
Direction alignment

Phase 1: Concept & Narrative
Script Docs

Phase 2: System Design
Architecture Docs

Phase 3: Data & Content
Mock Data Files

Phase 4: Build
The Code

Phase 5: Polish
Texture & Polish

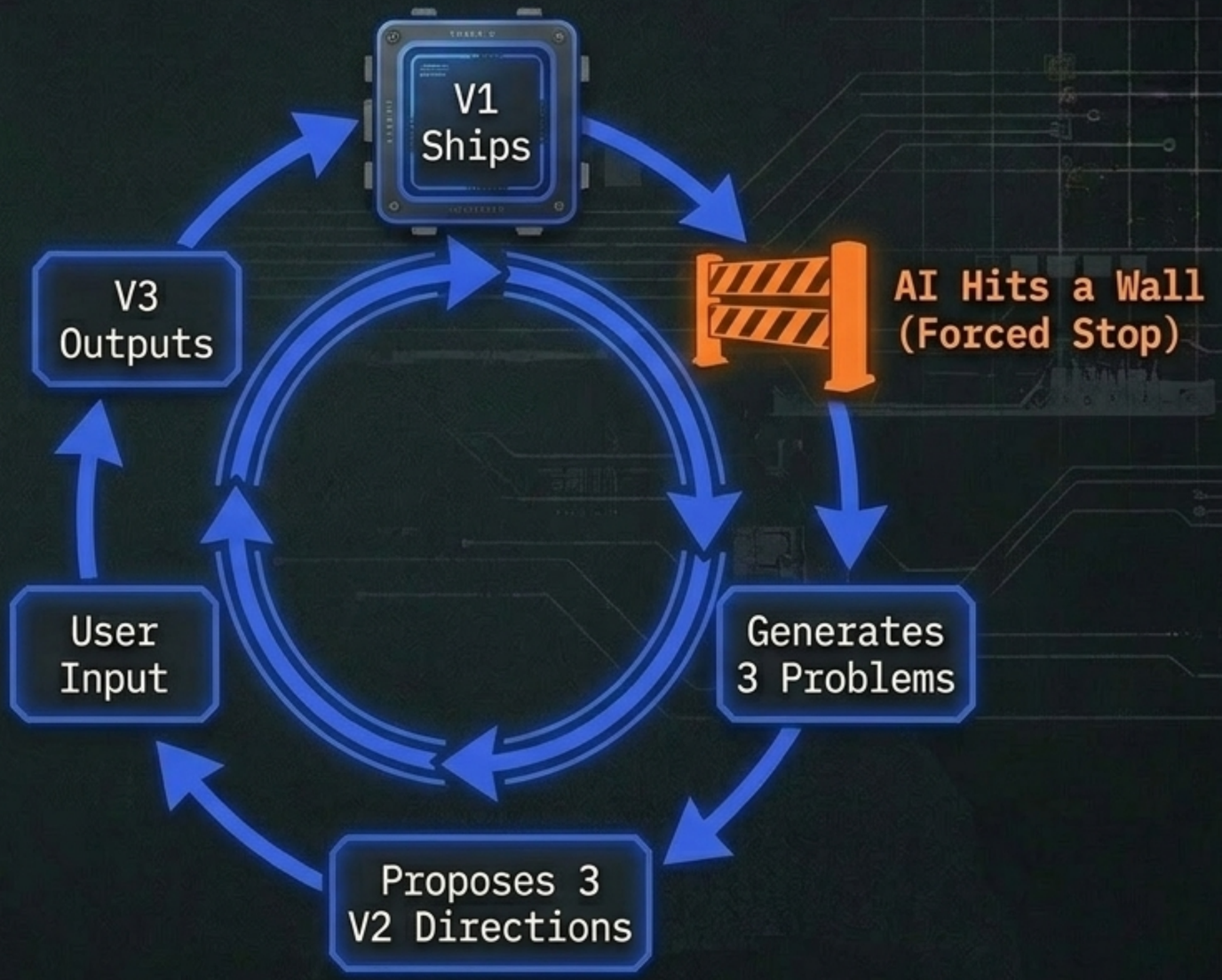
Each phase outputs a file. Each file feeds the next phase. No skipping.

The secret sauce is the forced self-critique loop

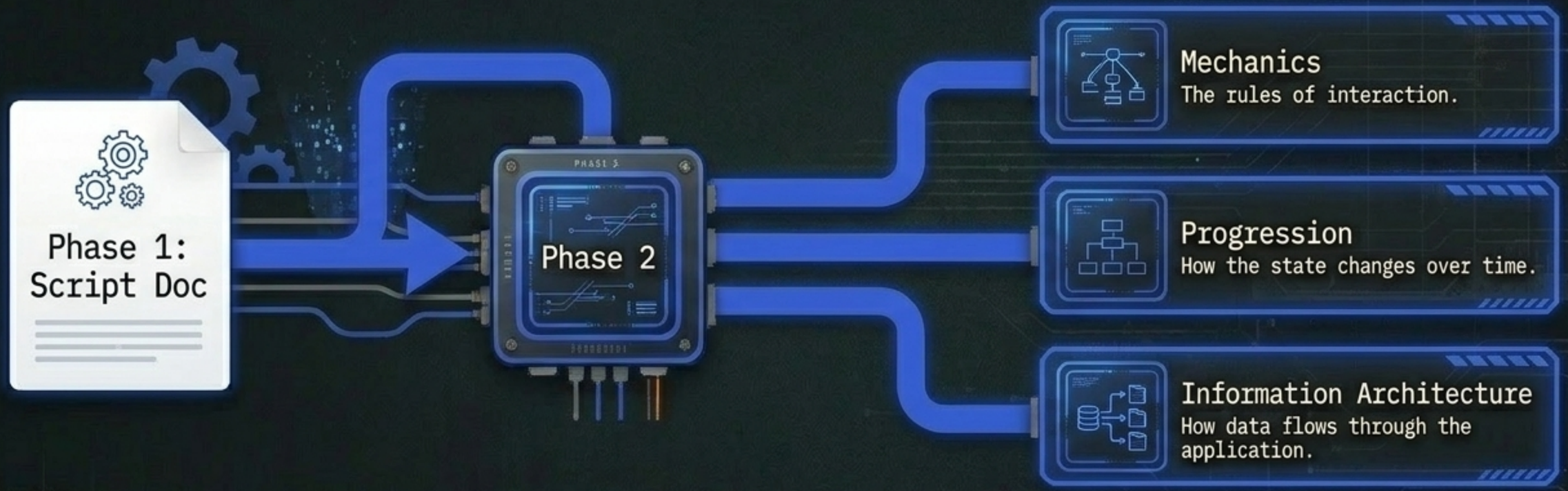
If you let the AI reflect on a concept while writing code, it won't reflect. It will just keep writing. AI does not self-critique by default. The workflow must force it.

Case Study: Inside Job

V1 looked solid but predictable. Claude Code, forced to critique itself, identified three specific issues, proposed three improvements, and ultimately merged them into a complex three-layer narrative structure.

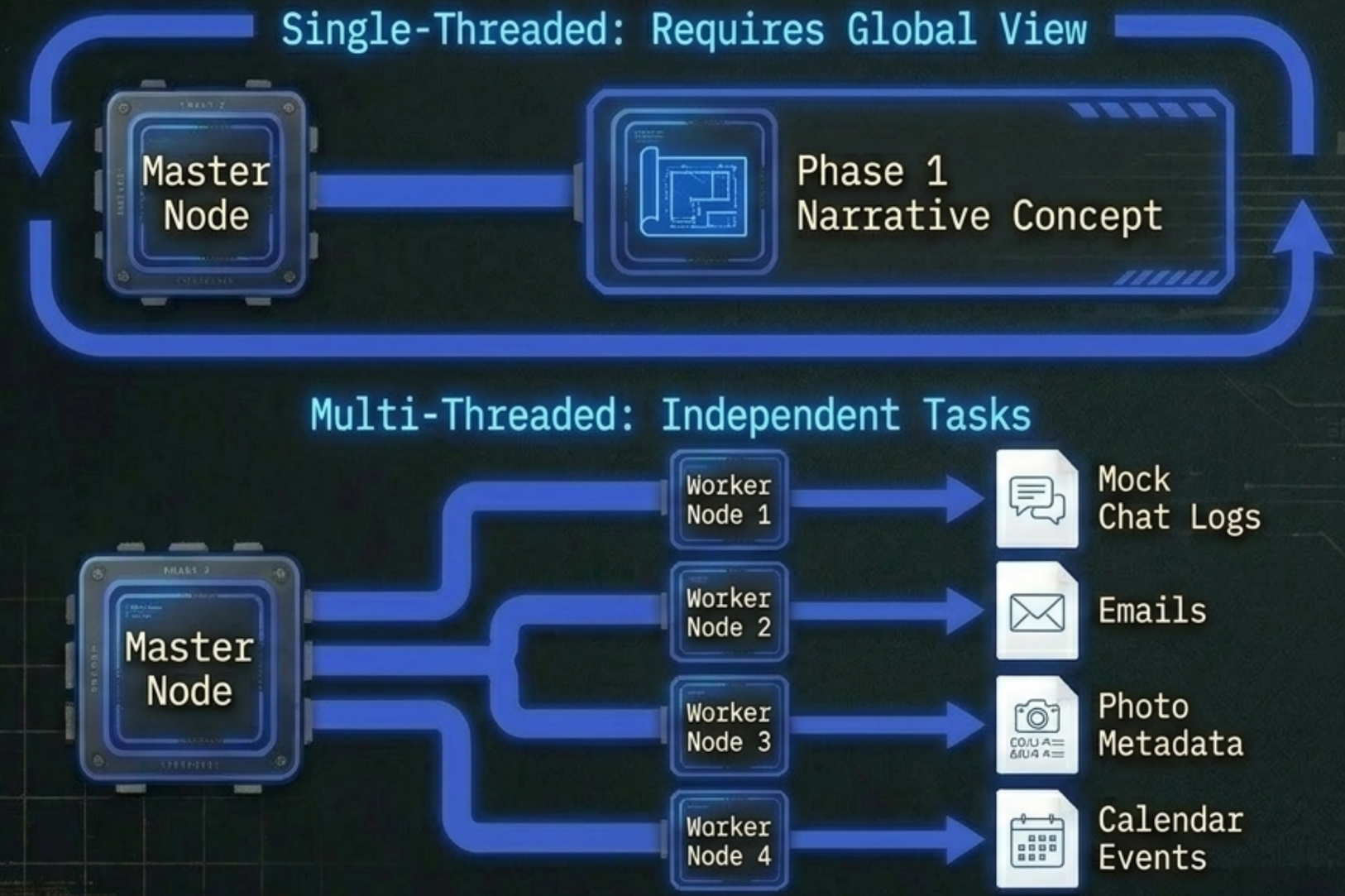


Translating concept into system architecture



Because you forget. Three days from now, you won’t know if a bug was a bad concept or if the AI drifted during implementation.
The System Design Document proves the original decision.

Scaling output with parallel subagents

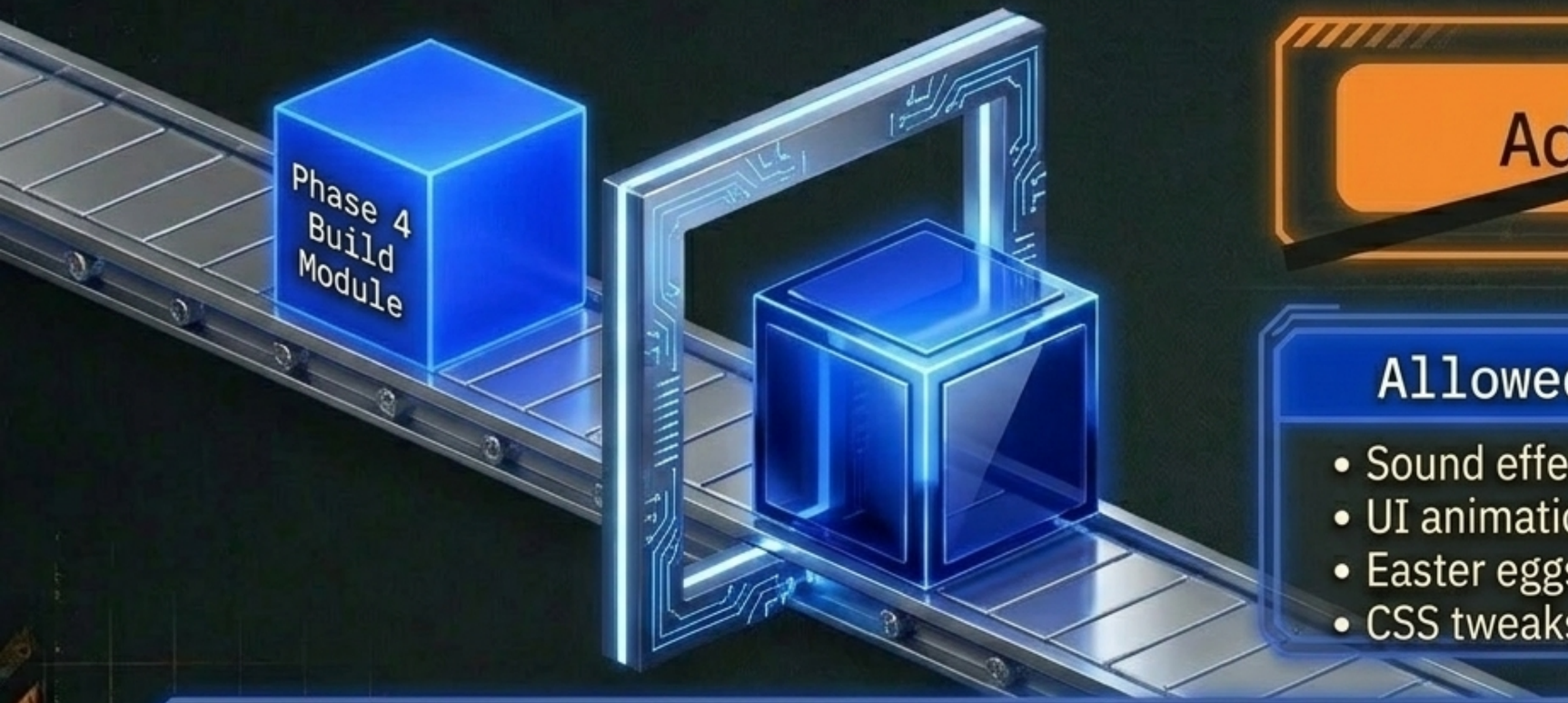


Phase 1 must stay single-threaded to prevent concept chaos.

Phases 3 and 4 thrive on parallelization.

By spawning parallel subagents for independent data types, the system drastically speeds up generation while staying strictly aligned to the master narrative.

Phase 5 is for texture, not new features



~~Add Feature~~

Allowed

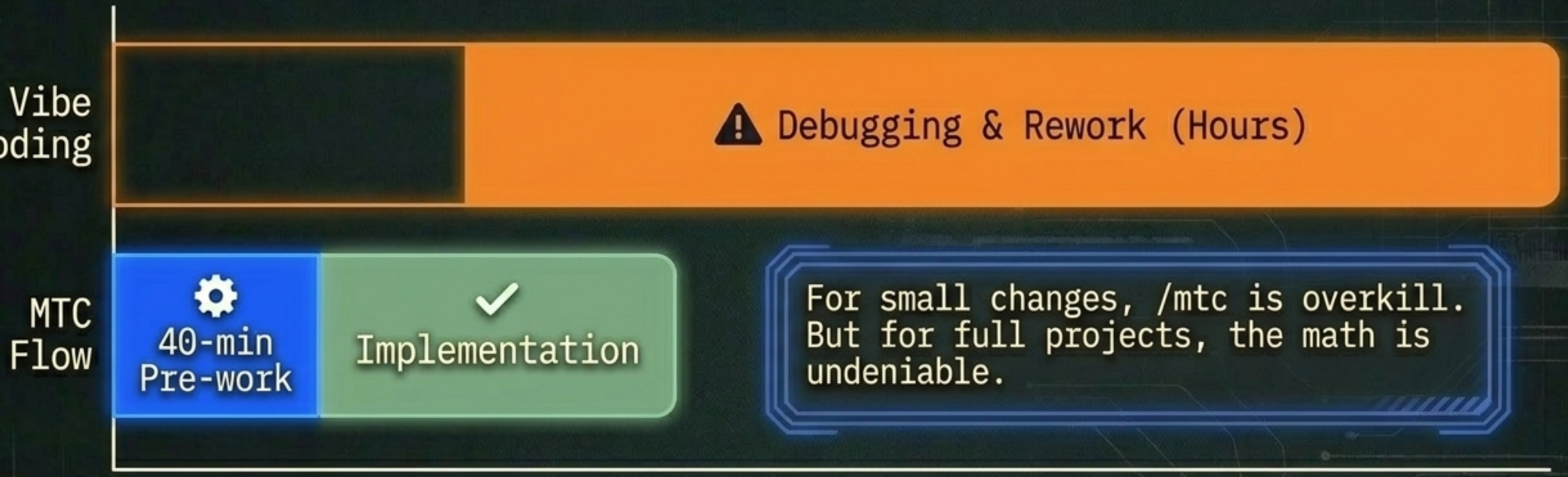
- Sound effects
- UI animations
- Easter eggs
- CSS tweaks

Forbidden

- Core logic changes
- New features
- System mechanics

By Phase 4, the AI doesn't decide what to build, only how to build it. Phase 5 is strictly reserved for polish.

The ROI of forced planning



Inside Job Case Study:

40 minutes: Total time spent on concept and system design pre-work.

2-3 hours: Minimum estimated time saved by avoiding frustrating drift correction and rework.

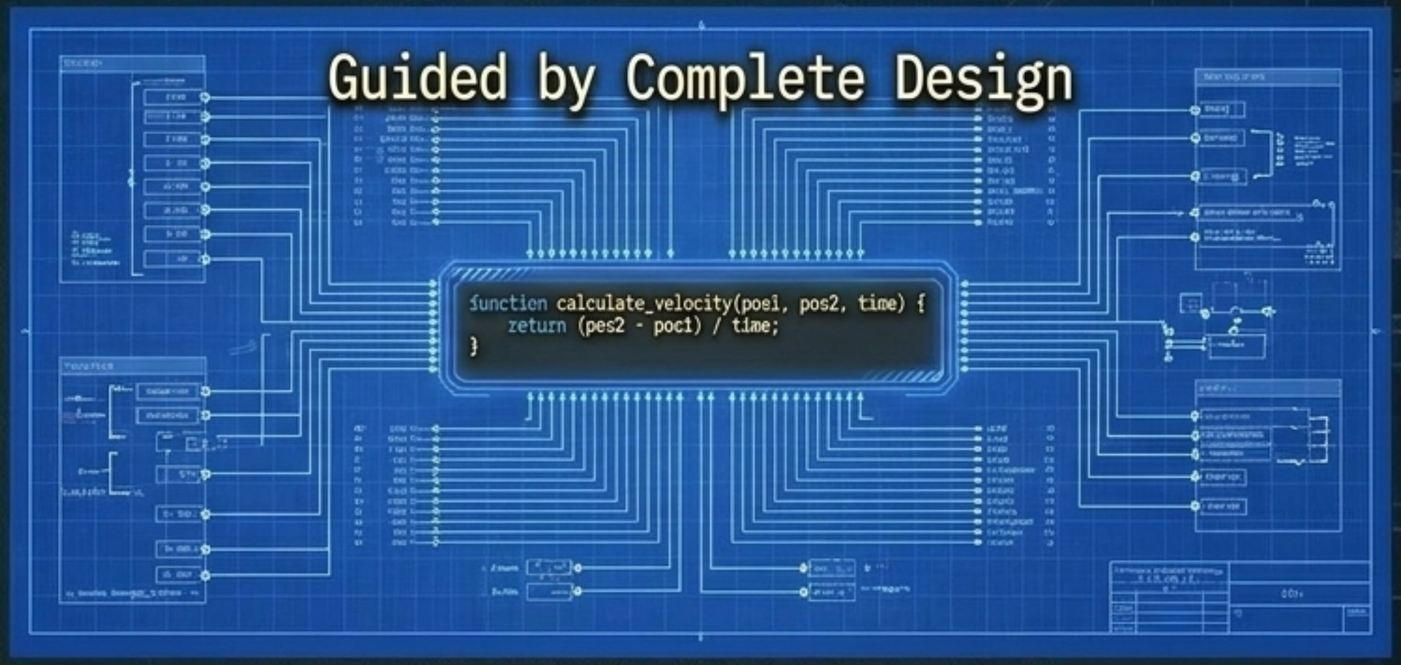
Raising the quality floor of AI generation

This workflow doesn't just prevent bugs. It raises the baseline quality of the output. The AI is no longer writing code in a narrow, local context. It is executing a mathematically sound design architecture.

Local Context

```
function calculate_velocity(posi, pos2, time) {
    return (pos2 - poc1) / time;
}
```

Guided by Complete Design



Transition from an eager typist to an AI Game Producer. Adopt the 6 phases. Don't skip the documents.