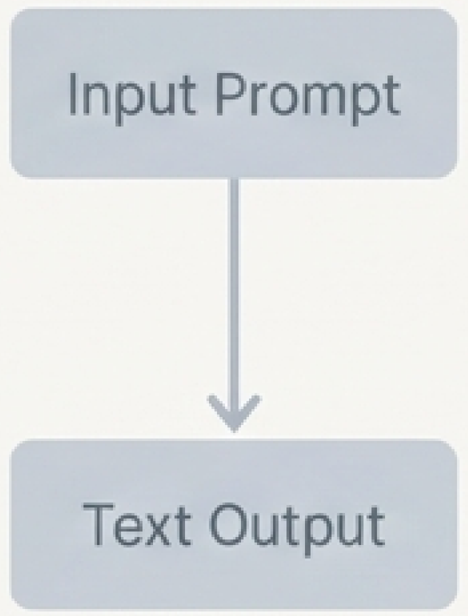
```
>_ claude --learn CC-102
```

CC-102: Effective Prompting & Workflows

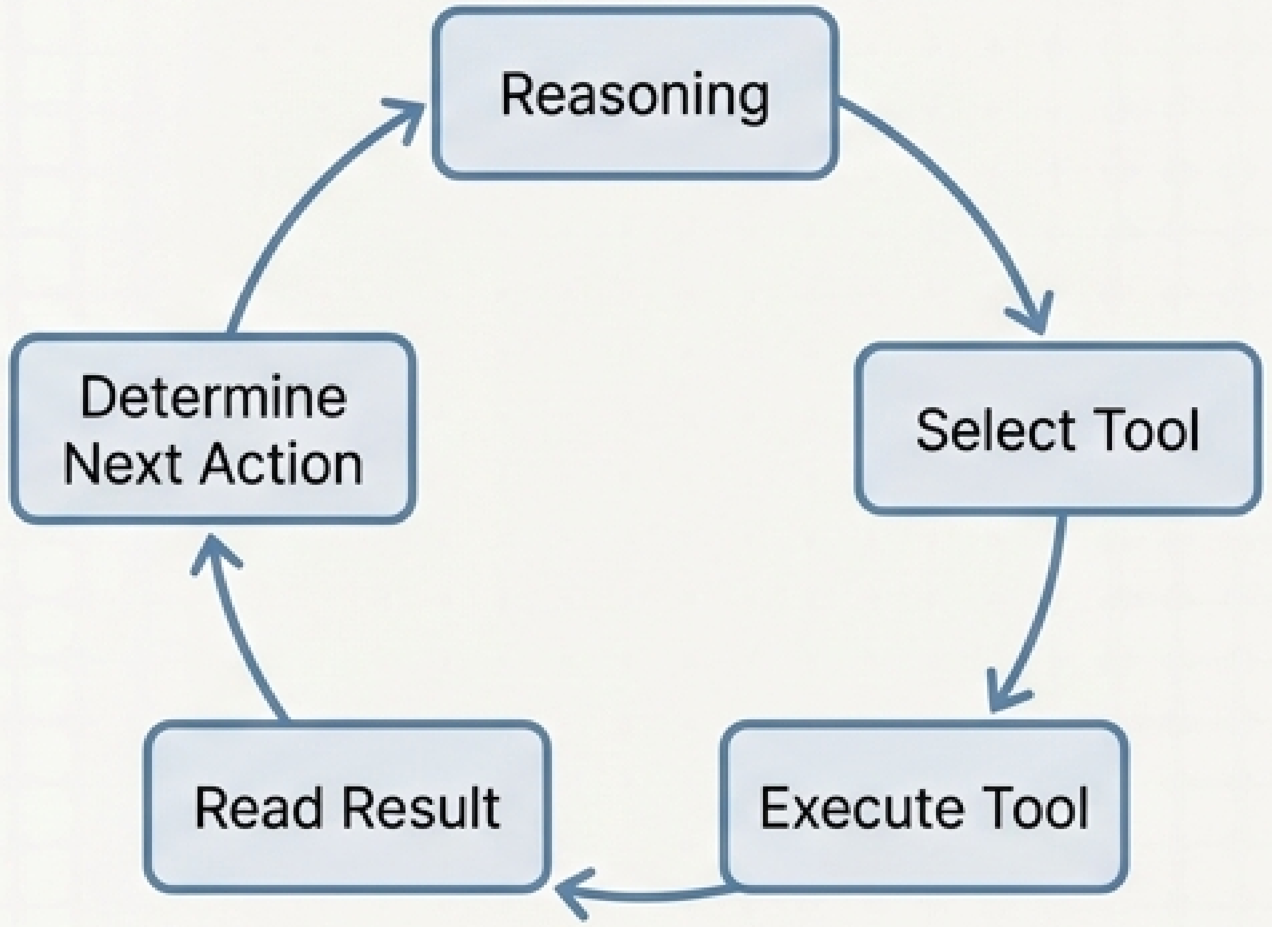
Mastering Task Decomposition, Plan Mode, and Autonomous Tool Execution

Claude Code is an Agent, Not a Chatbot

The Chatbot Paradigm



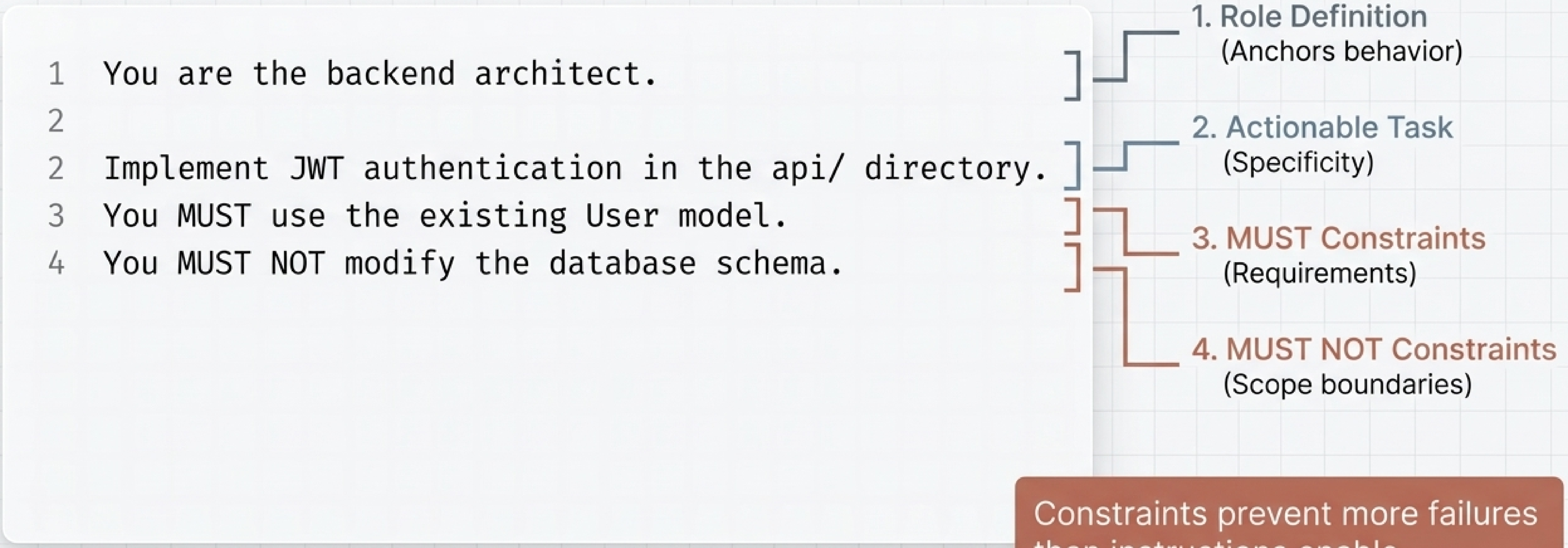
The Agentic Paradigm



You are not asking for text generation; you are commanding a junior developer with direct filesystem and shell access.

NotebookLM

Engineering Intent: Precision Drives Drives Success



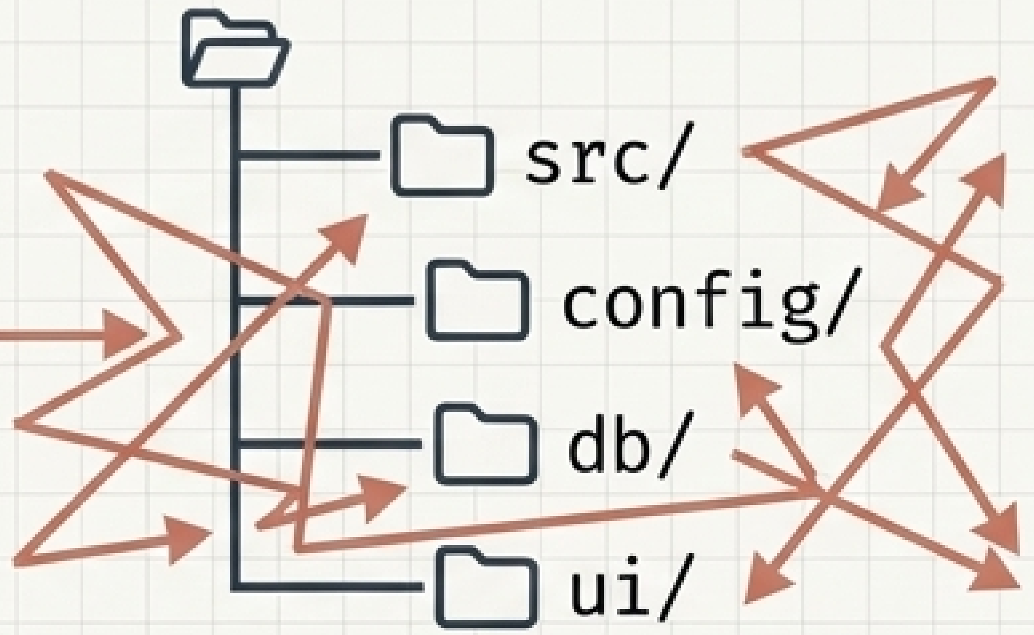
```
You are the backend architect.

Implement JWT authentication in the api/ directory.
You MUST use the existing User model.
You MUST NOT modify the database schema.
```

1. Role Definition (Anchors behavior)
2. Actionable Task (Specificity)
3. MUST Constraints (Requirements)
4. MUST NOT Constraints (Scope boundaries)

Constraints prevent more failures than instructions enable.

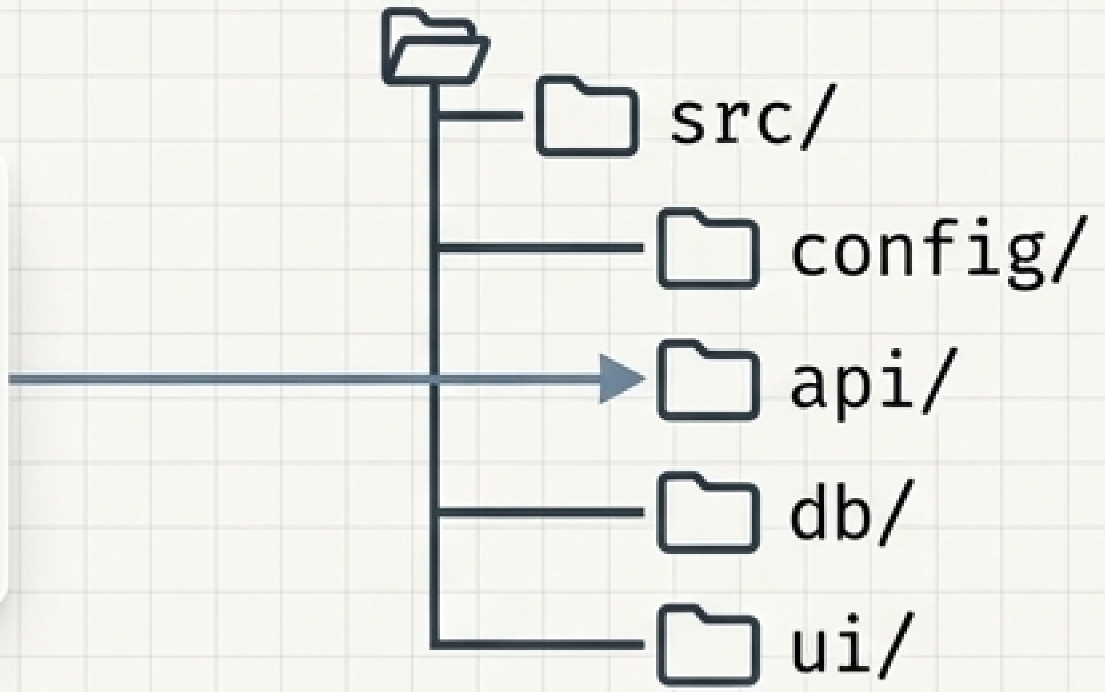
Why ‘Make it better’ Fails

```
Prompt: Add auth.
```



Unbounded execution. Modifies config, UI, and schema.

```
Prompt:
Implement JWT auth in api/.
MUST NOT modify db schema.
Use Plan Mode.
```



Predictable, constrained execution.

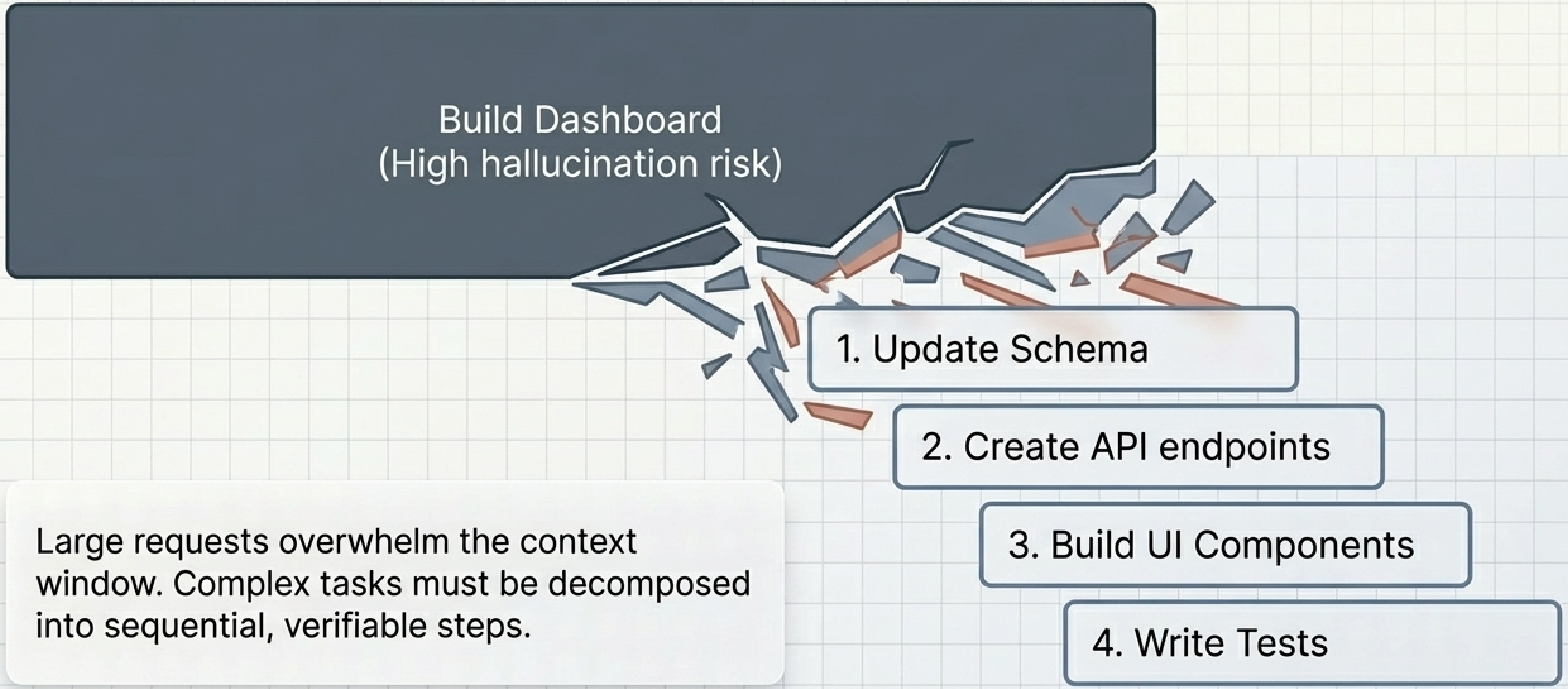
NotebookLM

Tailoring Prompts to the Objective

Code Tasks (Execution)	Research Tasks (Discovery)
Focus. Surgical precision, exact specifications.	**Focus.** Breadth, exploration boundaries, source citation.
Pathing. Explicit file paths (`` `src/utils.py` ``).	**Pathing.** Directory-level scoping (Glob the `` `docs/` `` folder).
Constraints. Strict MUST NOT rules ("Do not refactor outside this function").	**Output.** Structured artifacts ("Summarize findings in `` `architecture.md` ``").

NotebookLM

Break It Down Before Building It Up



Large requests overwhelm the context window. Complex tasks must be decomposed into sequential, verifiable steps.

EnterPlanMode: Measure Twice, Cut Once

Phase 1: Explore

CC uses `Glob`/`Read` to understand context.

Phase 2: Design

CC drafts a proposed approach in markdown.

Phase 3: Approve

Human reviews, adjusts, and approves the plan.

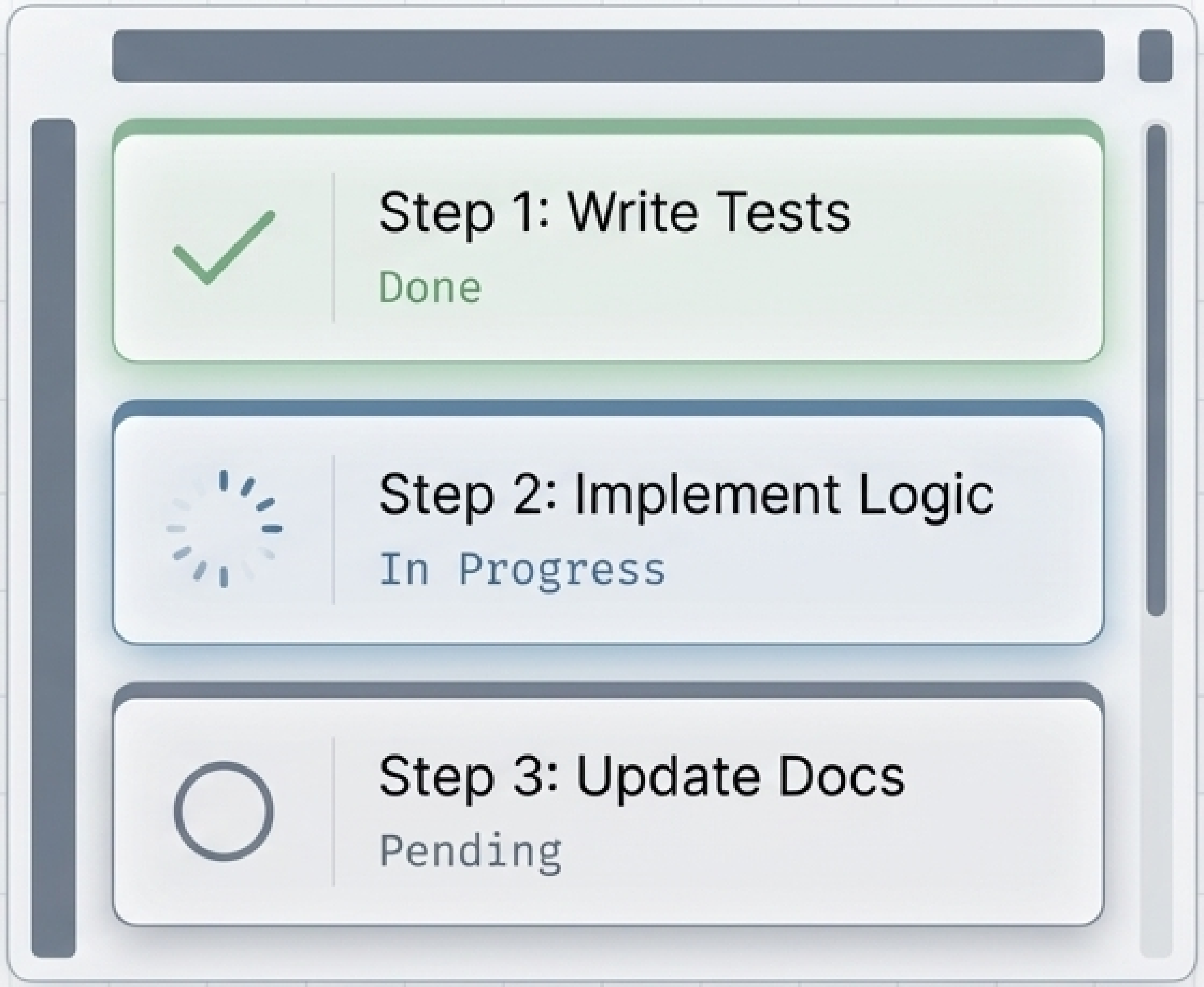
Phase 4: Execute

CC systematically implements the approved steps.

Never dive straight into implementation on multi-step features.

Editorial IDE Blueprint

TodoWrite: The Autonomous Checklist



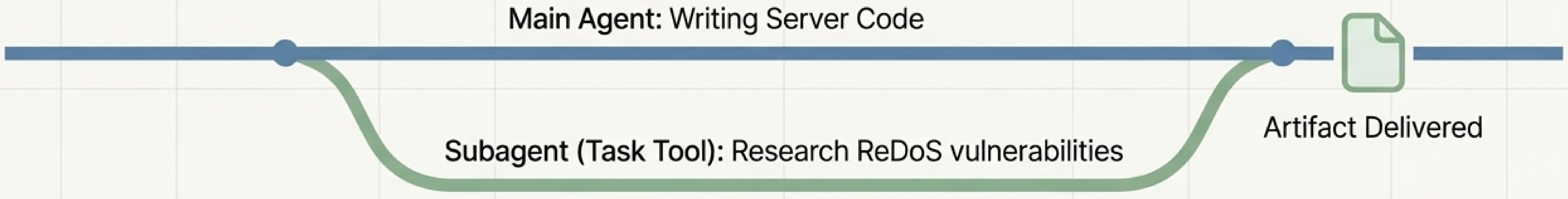
TodoWrite acts as CC's short-term memory across complex, multi-tool executions, ensuring it never loses its place in the sequence.

How Your Words Trigger Autonomous Tools

Find all files...	`Glob` (Pattern matching)
Where is X used...	`Grep` (Regex search)
Change this function...	`Read` then `Edit`
Test the code...	`Bash` (Shell execution)

CC must ALWAYS use Read before it is allowed to use Edit.

Parallel Execution with Subagents


Main Agent: Writing Server Code
Subagent (Task Tool): Research ReDoS vulnerabilities
Artifact Delivered

Use the Task tool to spawn independent Claude instances for isolated research without polluting your main context window.

Seamless Version Control via Bash

Human Prompt

What changed? Commit it.

↓

Agent Reasoning

I need to check the git status, review the diff, and commit the changes.

↓

Tool Execution (Bash)

```
git status
git diff
```

↓

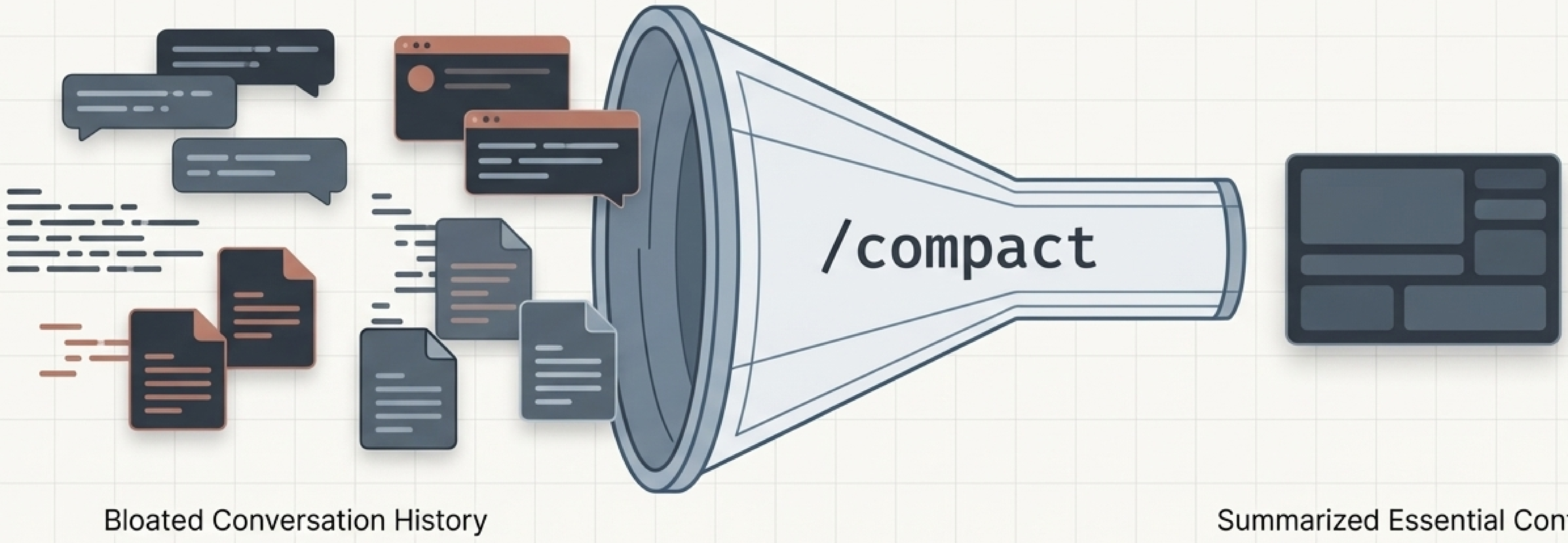
Tool Execution (Bash)

```
git add .
git commit -m 'feat: implement JWT auth'
```

Claude Code interacts with Git exactly like a human does—through the CLI using Bash.

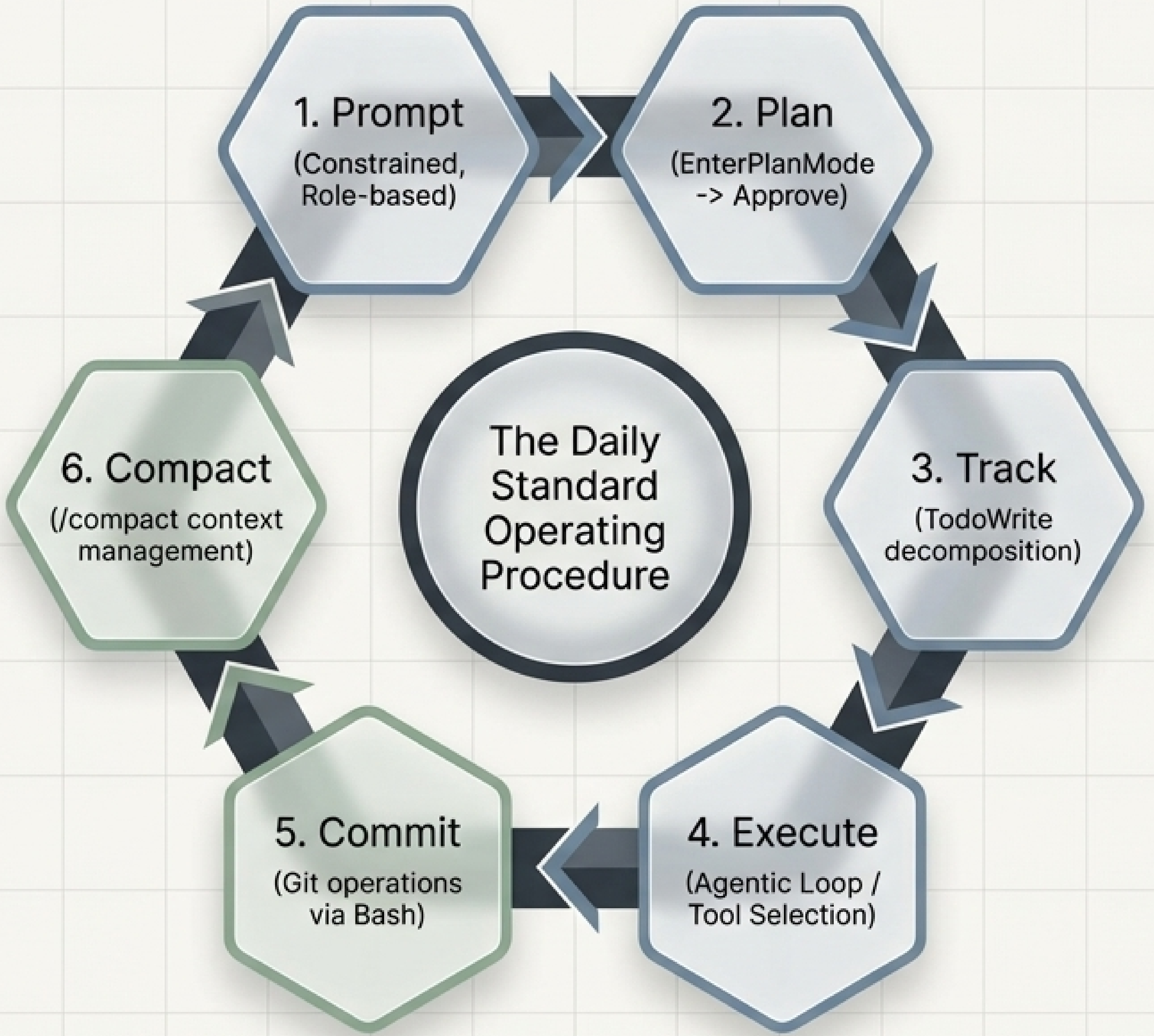
Always require descriptive commit messages in your prompt.

Keeping the Workspace Clean with /compact



Long sessions exhaust the context window and degrade reasoning. Use `/compact` to compress history while preserving the essential workspace context.

The Master CC-102 Workflow


1. Prompt
(Constrained, Role-based)
2. Plan
(EnterPlanMode -> Approve)
3. Track
(TodoWrite decomposition)
4. Execute
(Agentic Loop / Tool Selection)
5. Commit
(Git operations via Bash)
6. Compact
(/compact context management)
The Daily Standard Operating Procedure

Assessment Readiness: CC-102

- ☒] I write `constrained`, `role-based` prompts with strict `MUST NOT` rules.
- ☒] I decompose complex tasks using `Plan Mode` prior to execution.
- ☒] I leverage `TodoWrite` for autonomous task tracking.
- ☒] I understand how my phrasing triggers `Glob`, `Grep`, `Read`, and `Edit`.
- ☒] I spawn **subagents** using the `Task` tool for isolated research.
- ☒] I drive full `Git` workflows entirely through Claude Code Bash executions.

Mastery over the agent requires mastery over the intent.