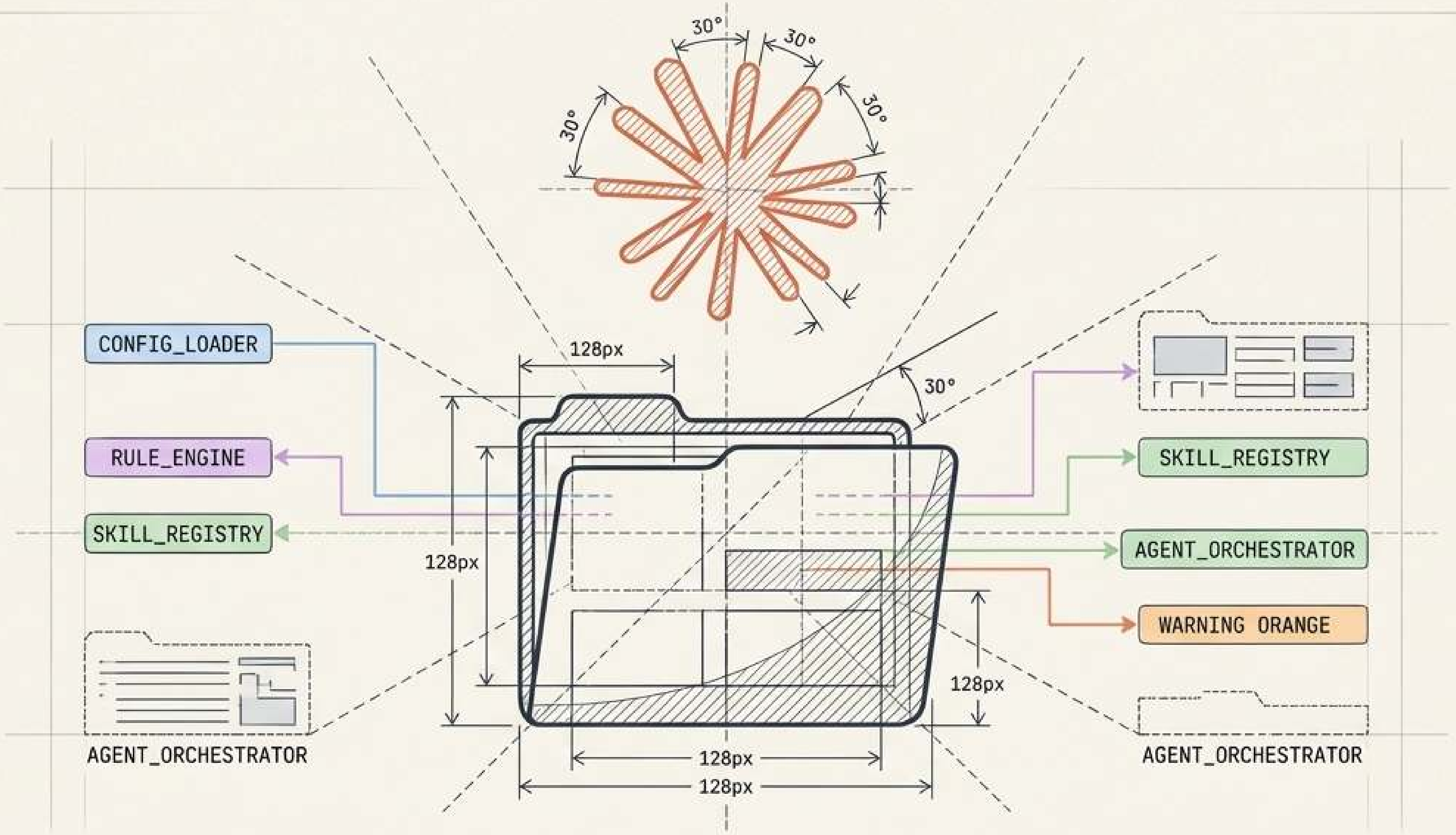
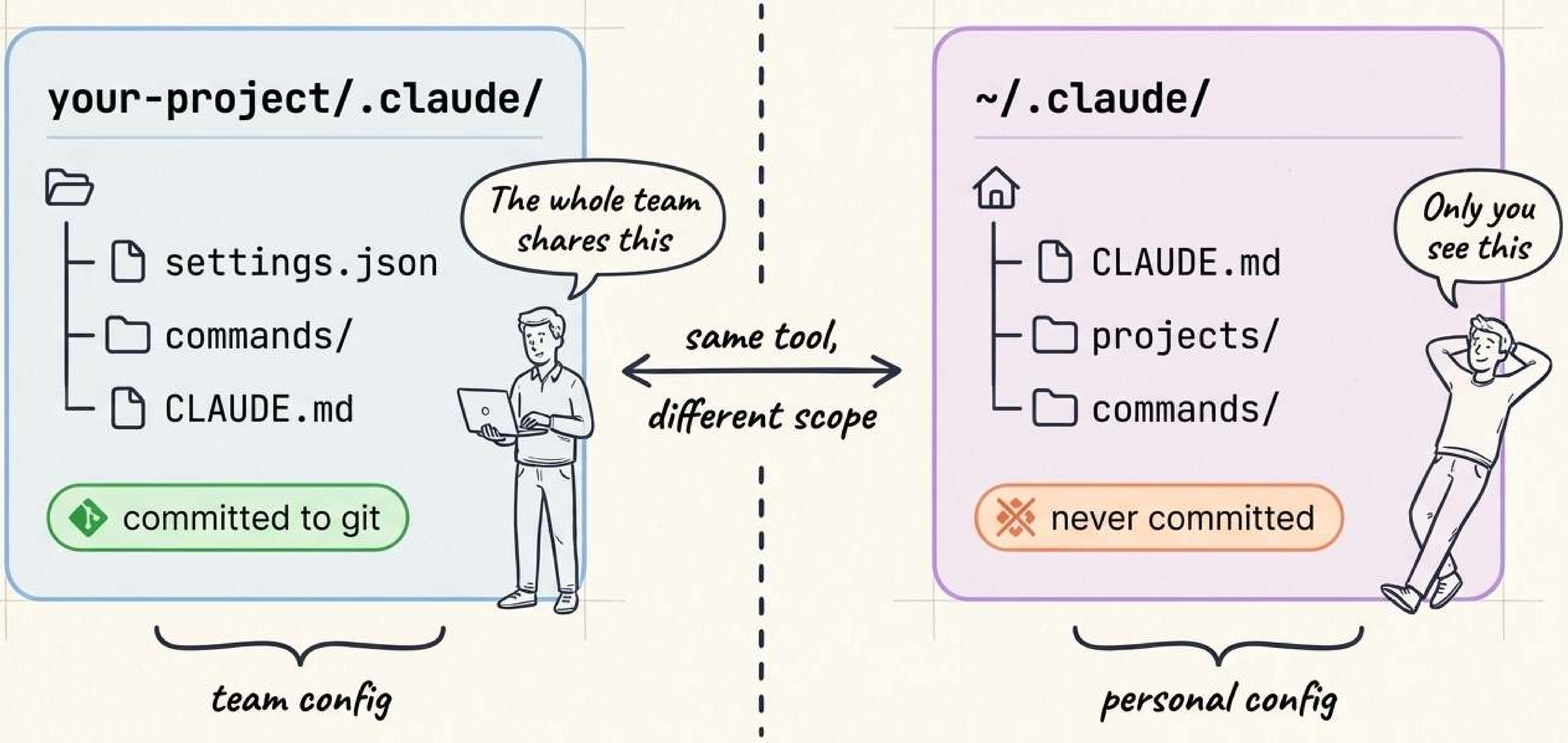


Anatomy of the .claude/ Folder

The Engineer's Blueprint for Claude Code Configuration.

Treat it like infrastructure. Transform the black box into a precise control center.



Two folders. One for the team, one for you. Both load on every session.

```
your-project/ ←—— team instructions, committed
├ CLAUDE.local.md ← personal overrides, gitignored
├ .claude/ ←—— the control center
│ ├ settings.json ←—— permissions + config, committed
│ ├ settings.local.json ← personal permissions, gitignored
│ ├ commands/ ←—— custom slash commands
│ │ ├ review.md
│ │ ├ fix-issue.md
│ │ └ deploy.md
│ ├ rules/ ←—— modular instruction files
│ │ ├ code-style.md
│ │ ├ testing.md
│ │ └ api-conventions.md
│ ├ skills/ ←—— auto-invoked workflows
│ │ ├ security-review/
│ │ │ └ SKILL.md
│ │ └ deploy/
│ │   └ SKILL.md
│ └ agents/ ←—— subagent personas
│   ├ code-reviewer.md
│   └ security-auditor.md
```

Everything Claude needs to know about your project lives right here.

The Core Prompt: CLAUDE.md

The first file read. Loaded straight into the system prompt and kept in mind for the entire conversation.

The Guidelines

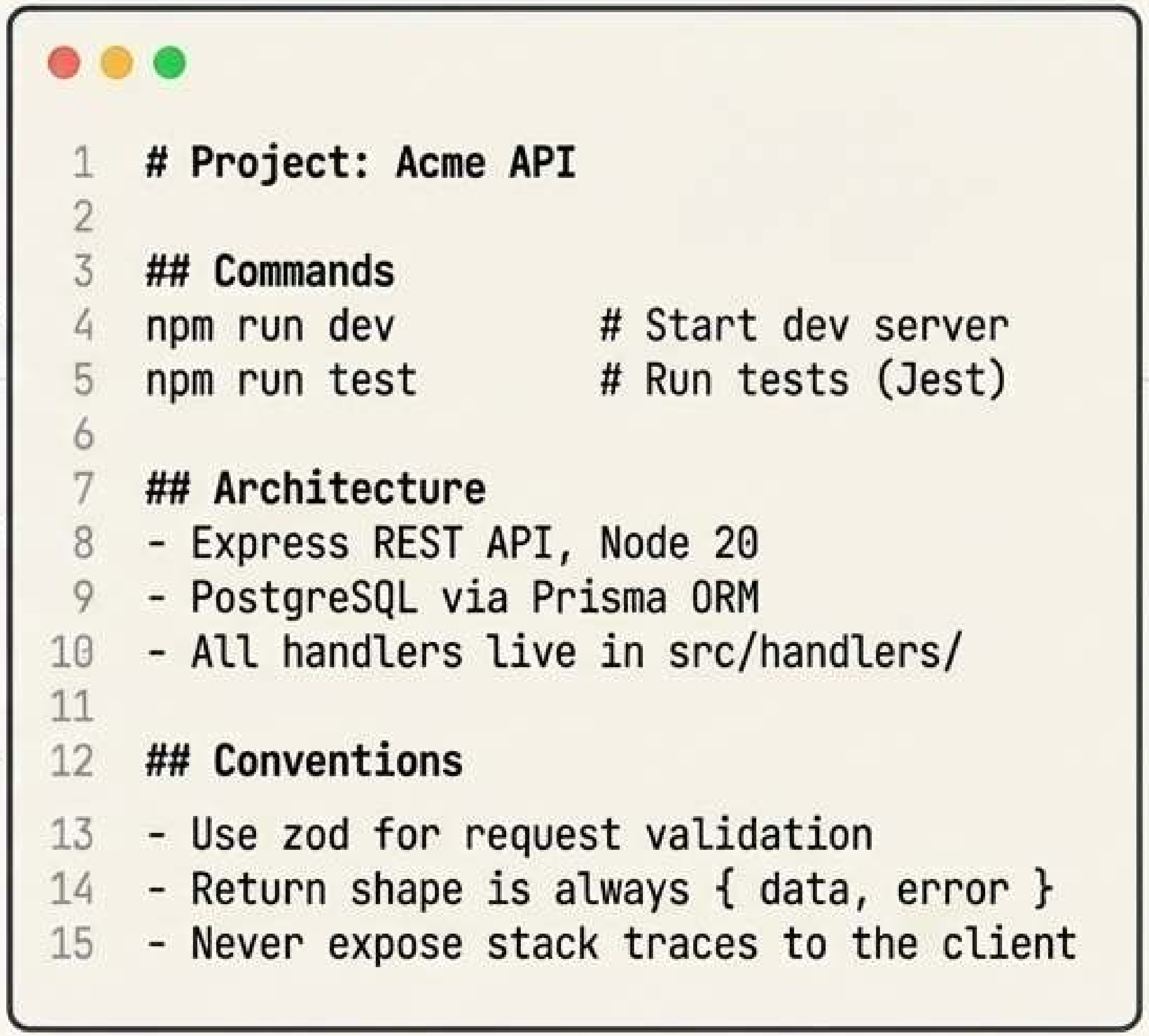
✅ **Do**

- Include build commands (npm run dev)
- Define architecture (Node 20, PostgreSQL)
- Set strict conventions (zod for validation)

❌ **Don't**

- Exceed 200 lines (causes context drop-off)

The Code Snippet

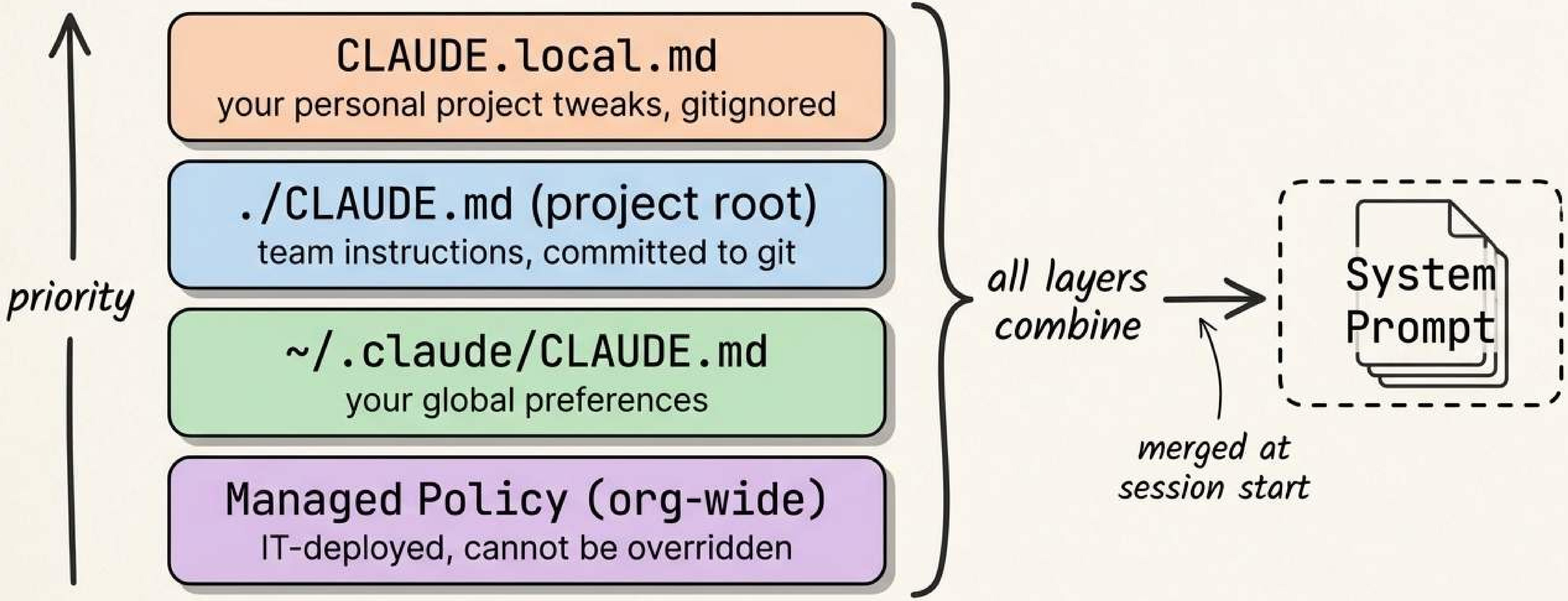


```
# Project: Acme API

## Commands
npm run dev            # Start dev server
npm run test           # Run tests (Jest)

## Architecture
- Express REST API, Node 20
- PostgreSQL via Prisma ORM
- All handlers live in src/handlers/

## Conventions

- Use zod for request validation
- Return shape is always { data, error }
- Never expose stack traces to the client
```

IDE Split View

The Conflict Resolution Stack

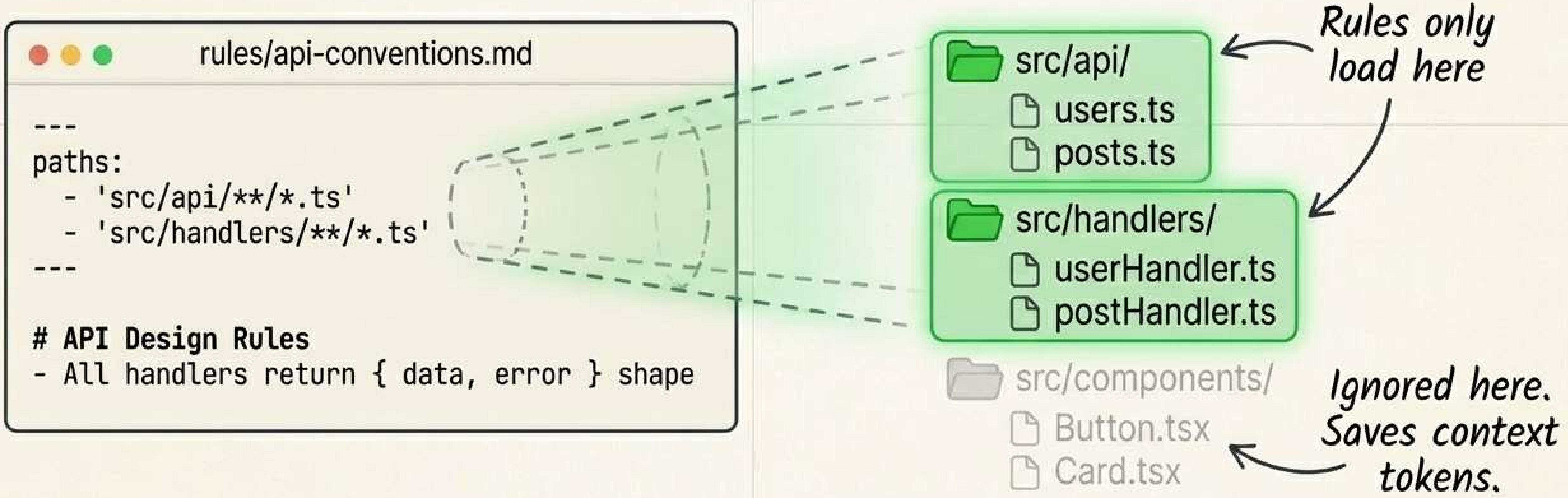
Local overrides global. Personal overrides team.


CLAUDE.local.md
your personal project tweaks, gitignored
./CLAUDE.md (project root)
team instructions, committed to git
~/.claude/CLAUDE.md
your global preferences
Managed Policy (org-wide)
IT-deployed, cannot be overridden
priority
all layers combine
merged at session start
System Prompt

NotebookLM

Modularity & Path-Scoped Rules

Split instructions by concern. YAML frontmatter ensures rules only activate when editing matching directories, keeping the context window lean.

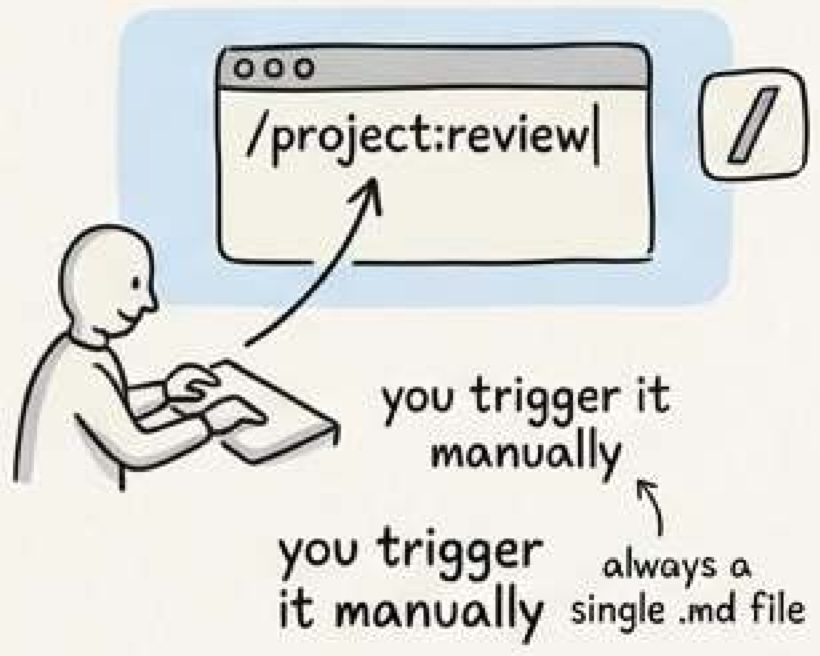
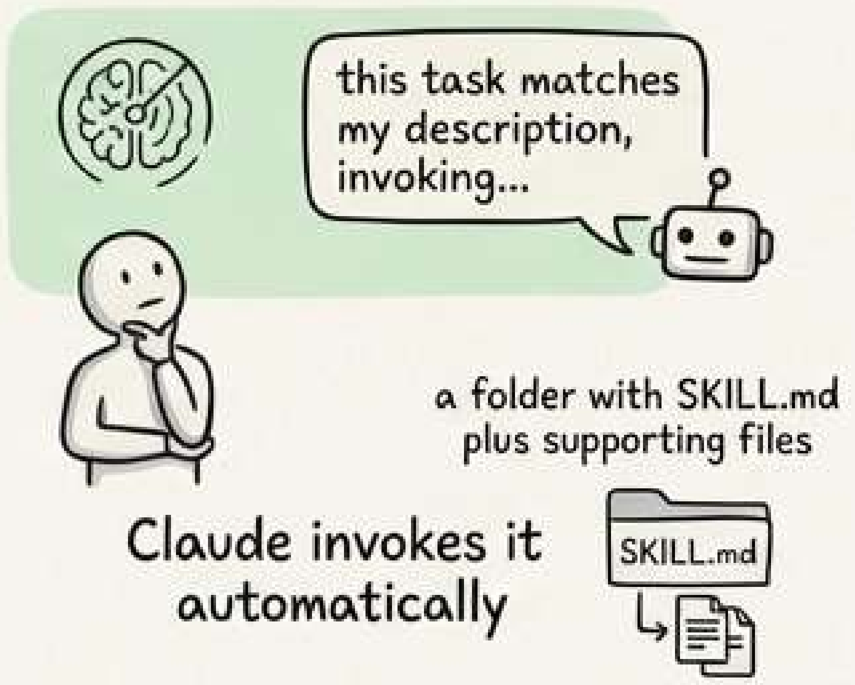


rules/api-conventions.md

```
---
paths:
  - 'src/api/**/*.ts'
  - 'src/handlers/**/*.ts'
---

# API Design Rules
- All handlers return { data, error } shape
```

The Automation Spectrum

From manual shortcuts to fully autonomous, isolated personas.

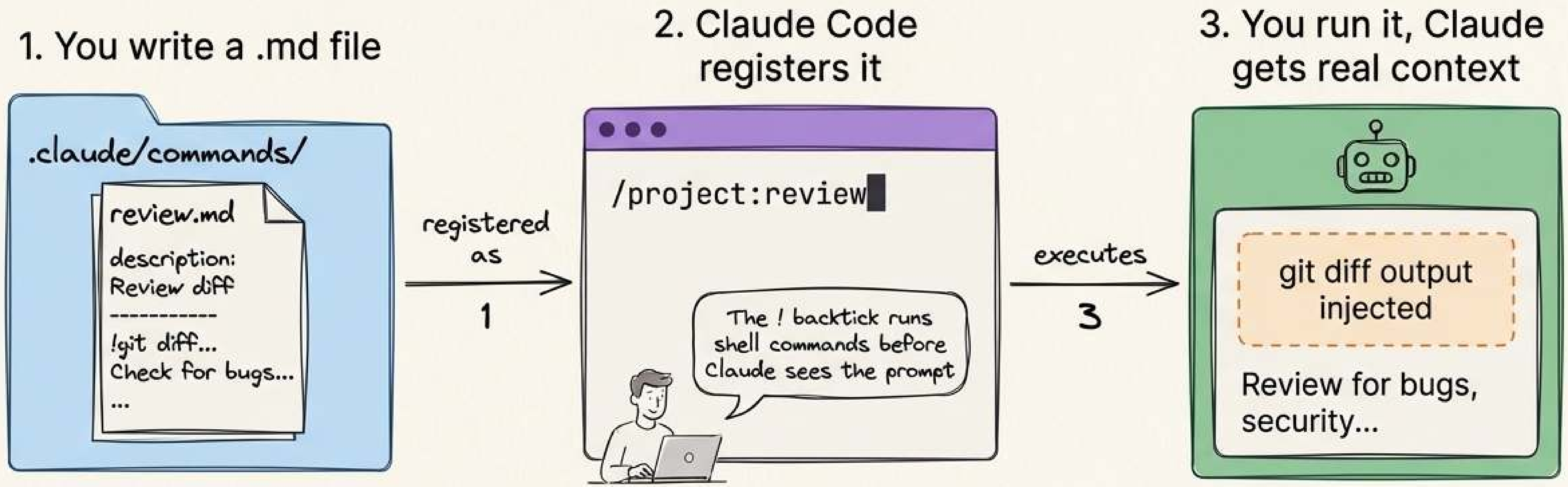
	Commands	Skills	Agents
Trigger	Manual (/)	Automatic	Delegated
Composition			System Prompt + Tools + Model
Scope			Isolated Context Window
Best For	Repeatable manual workflows	Context-aware automatic workflows	Complex, messy exploration tasks

/project:review
you trigger it manually
you trigger it manually
always a single .md file

this task matches my description, invoking...
a folder with SKILL.md plus supporting files
Claude invokes it automatically
SKILL.md

Manual Execution & Shell Injection

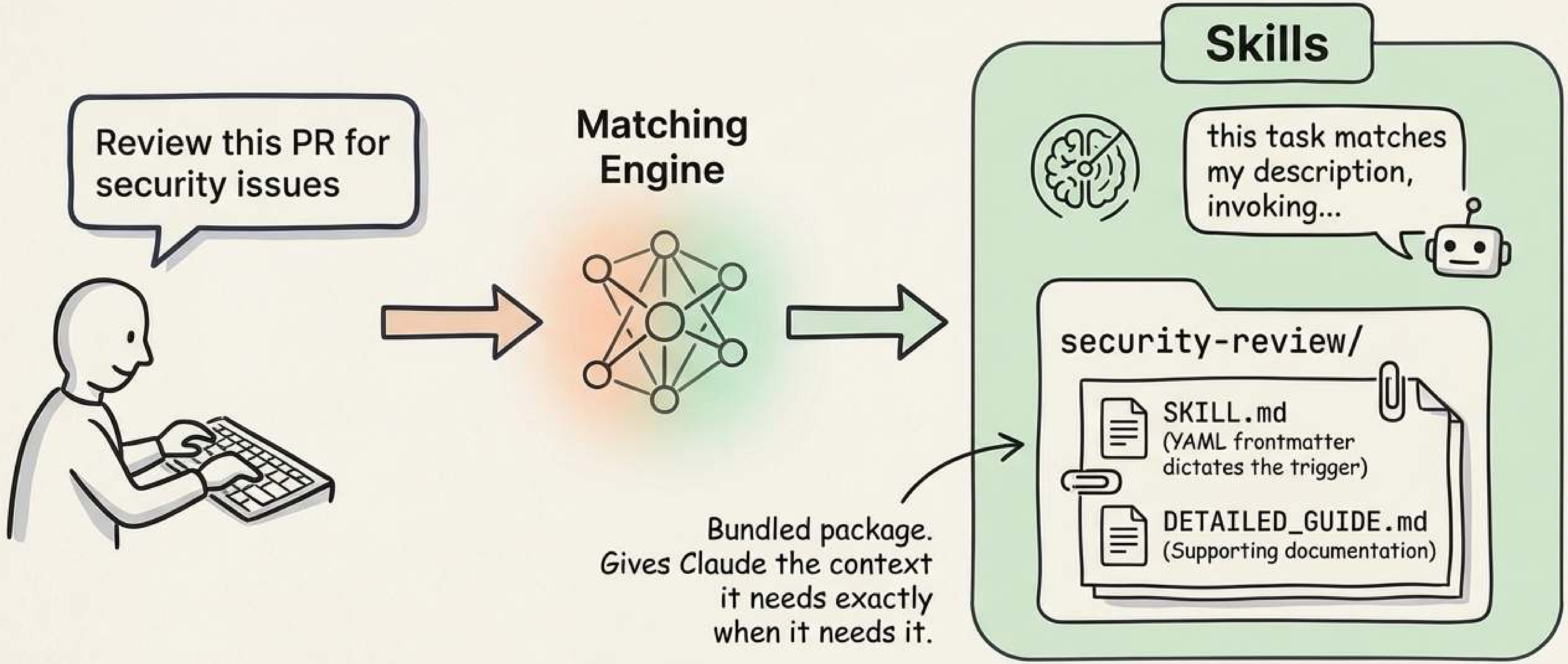
The **! backtick** syntax runs shell commands before Claude sees the prompt, embedding **real-world data** like a live **git diff**.



A filename, a description, and a few lines of markdown.
That is all it takes to build a custom command.

Context-Aware Automation: Skills

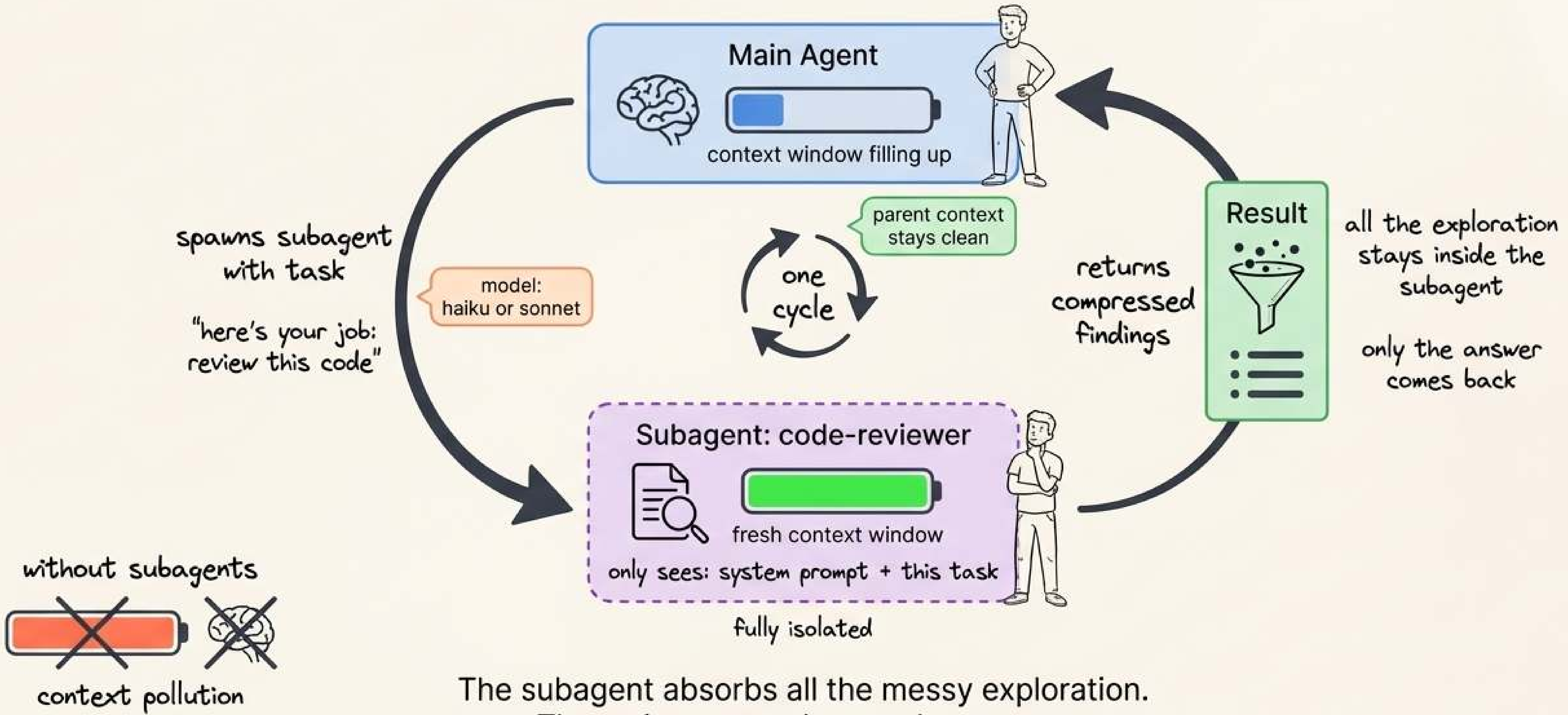
Skills watch the conversation and invoke automatically when a task matches their description. Unlike single-file Commands, Skills are bundled packages.


Review this PR for security issues
Matching Engine
Skills
this task matches my description, invoking...
security-review/
SKILL.md
(YAML frontmatter dictates the trigger)
DETAILED_GUIDE.md
(Supporting documentation)
Bundled package. Gives Claude the context it needs exactly when it needs it.

NotebookLM

Specialized Delegation: Agents

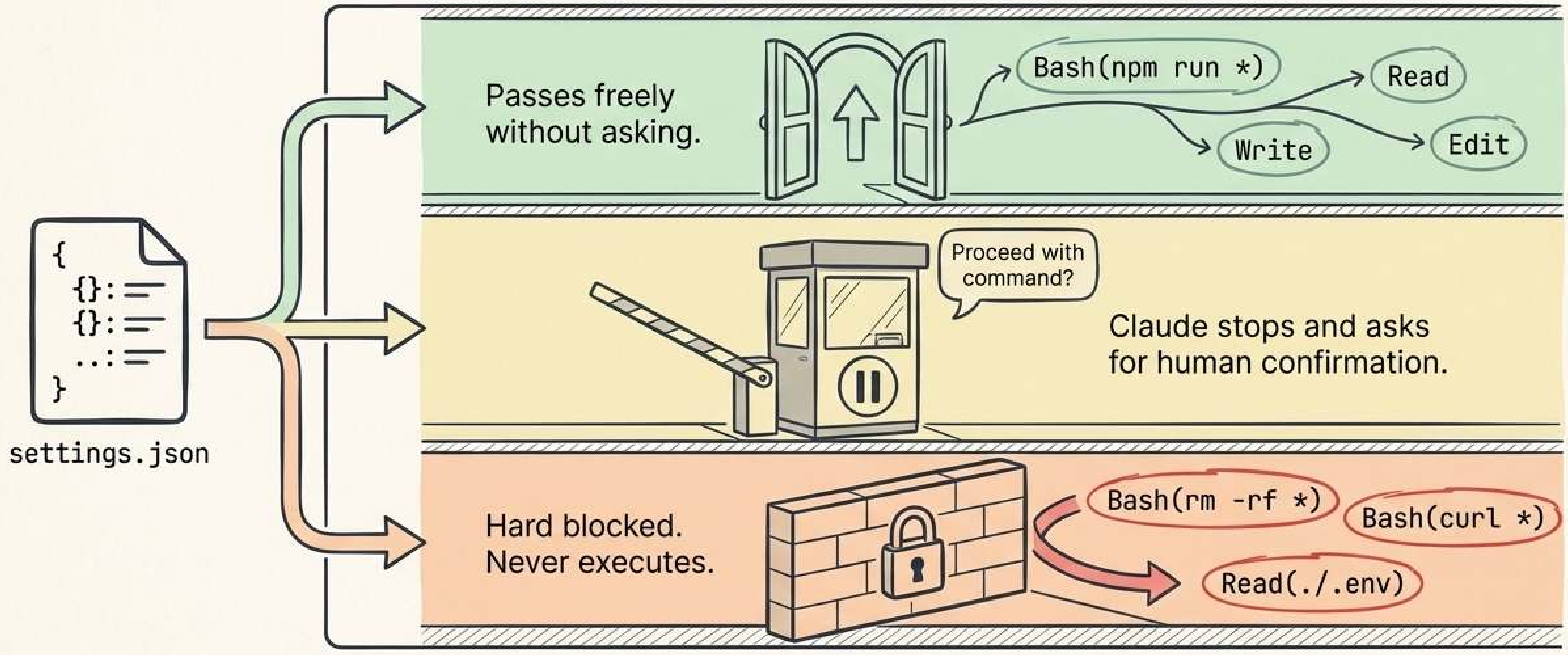
Spawning agents isolates context. The subagent absorbs messy exploration using cheaper models. The main session stays clean and uncluttered.


Main Agent
context window filling up
spawns subagent with task
"here's your job: review this code"
model: haiku or sonnet
one cycle
parent context stays clean
returns compressed findings
Result
all the exploration stays inside the subagent
only the answer comes back
Subagent: code-reviewer
fresh context window
only sees: system prompt + this task
fully isolated
without subagents
context pollution

The subagent absorbs all the messy exploration.
The main agent only gets the answer.

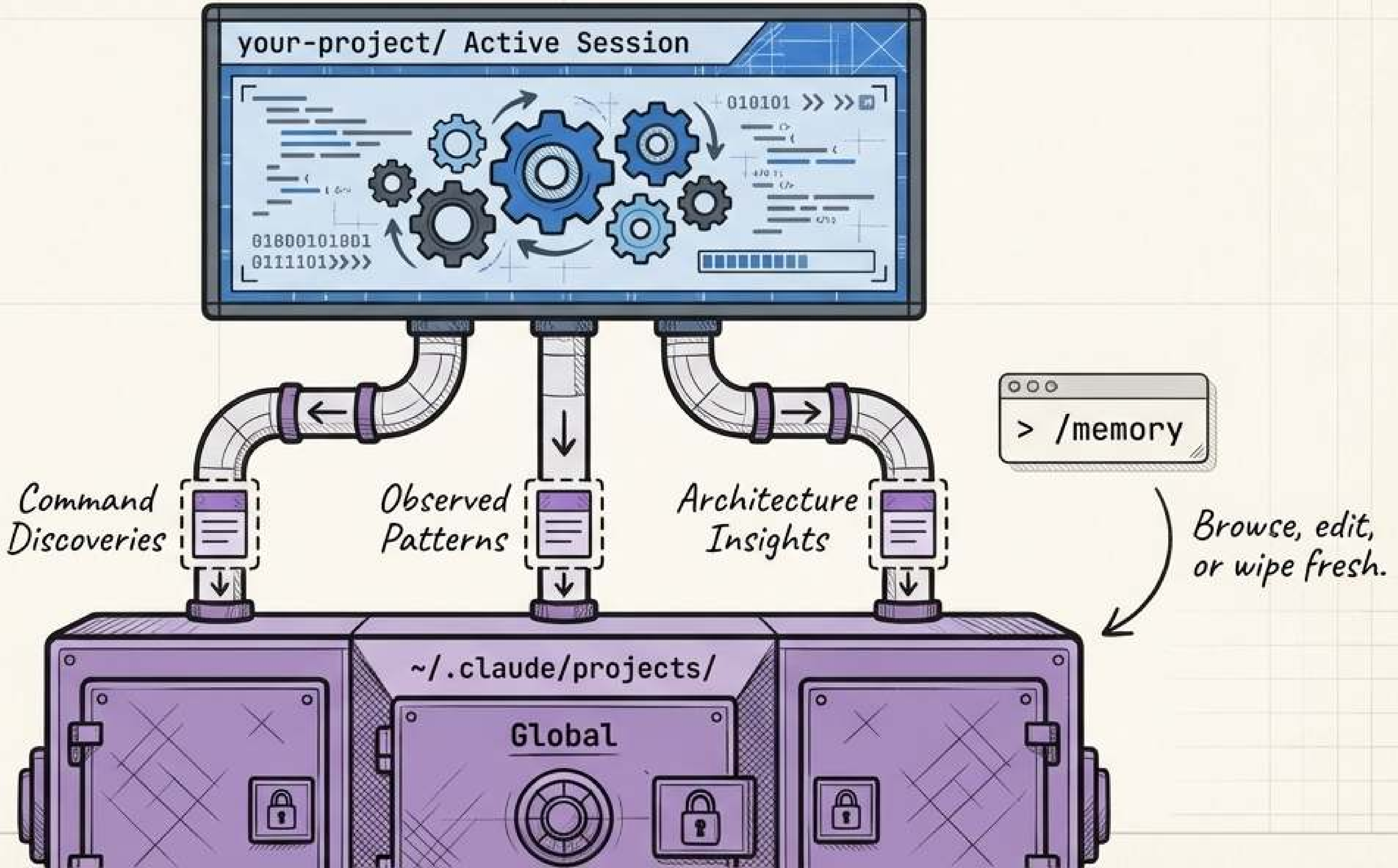
Permission Guardrails

Define exact boundaries. Allow safe scripts unconditionally, hard-deny destructive actions, and force manual confirmation for the middle ground.



Persistence & Auto-Memory

Claude Code automatically saves notes to itself as it works. Transcripts and auto-memory persist globally and are accessible via **/memory**.



The 5-Step Project Setup Playbook

Start small, refine as you go. `CLAUDE.md` is your highest-leverage file—get that right first, and everything else is optimization.

1. The Base

Run `/init`. Trim `CLAUDE.md` to essentials (<200 lines).

2. The Guardrails

Lock down settings.json. Allow scripts, deny `.env` reads.

3. The Daily Tools

Create commands/ for frequent workflows (e.g., **/project:review**).

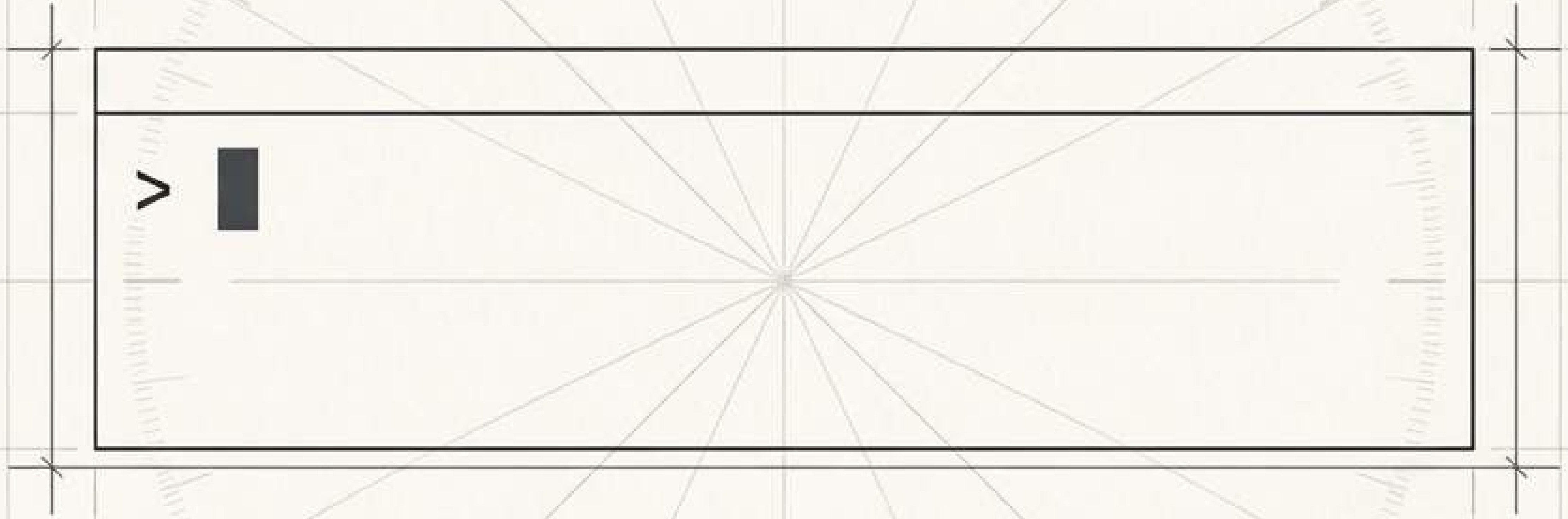
4. The Scaling Factor

Move overgrown instructions to `path-scoped rules/`.

5. Advanced Automation

Package complex workflows into autonomous `skills/` & `agents/`.

It is infrastructure, not magic.



```
>
```

The `.claude/` folder is a precise protocol for defining who you are, what your project does, and the boundaries it must follow. Define it clearly, and the tool builds the rest.