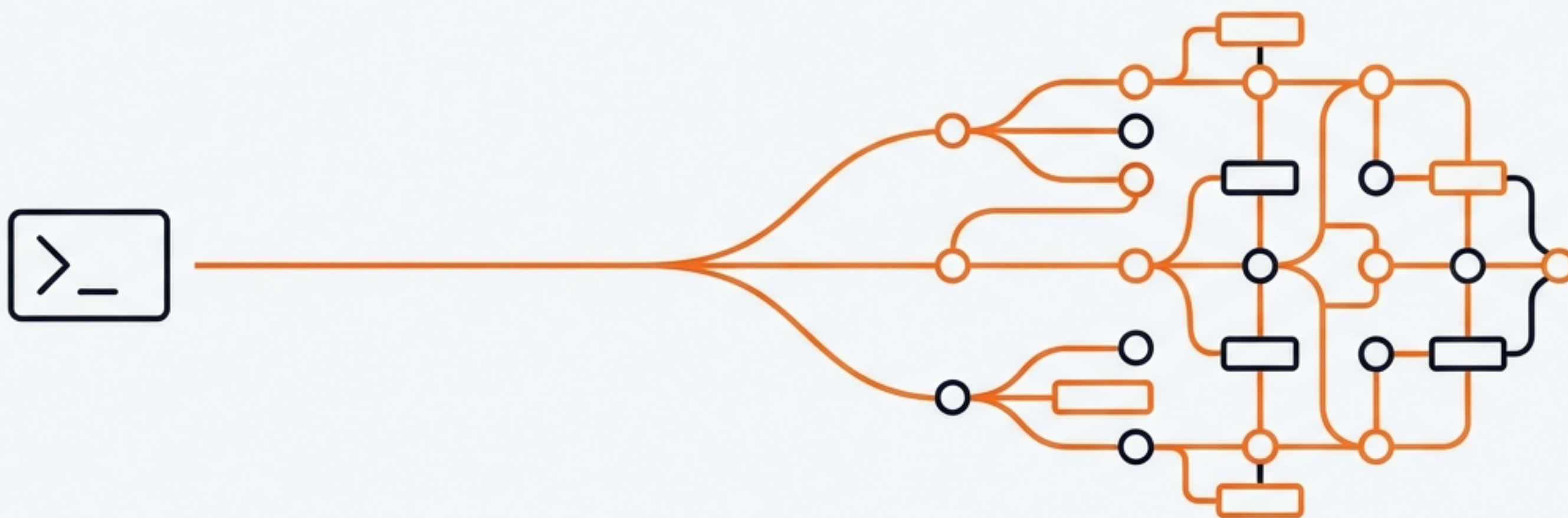


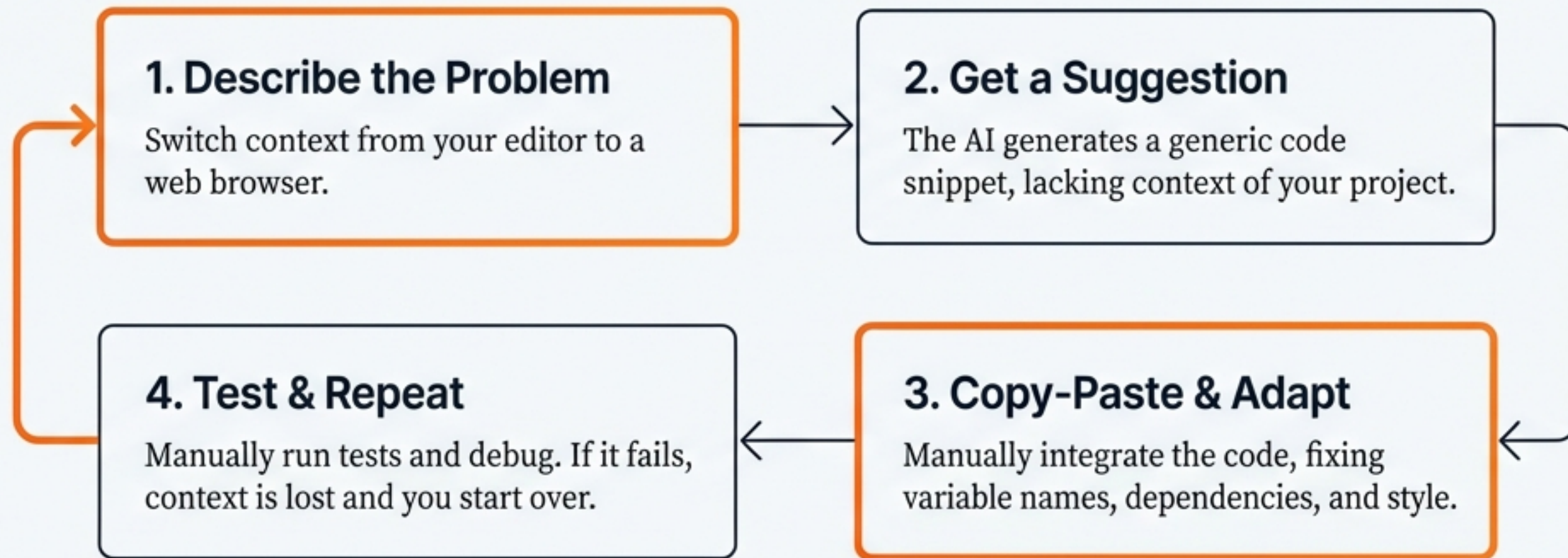
The Architecture of an AI Development Partner

Moving from passive tools to agentic collaborators with Claude Code.



Today's AI assistants are powerful, but their workflow is broken.

We've been taught to use AI as a separate consultant. This creates a cycle of friction that slows down development:



Every interaction starts from zero context. This is the fundamental limitation of the **Passive Assistance Model**.

Claude Code introduces an agentic model of collaboration.

Instead of a consultant you visit, the AI becomes an active agent—a pair programmer collaborating directly in your workspace.

Passive vs. Agentic AI Assistance

Passive AI (Chat-based)	Agentic AI (Claude Code)
Context Awareness: No file access; relies on your descriptions.	Context Awareness: Reads your actual codebase; understands project structure.
Interaction Model: Q&A: You ask, AI answers.	Interaction Model: Collaborative: AI proposes, you approve, AI executes.
Code Integration: Manual copy-paste and adaptation.	Code Integration: Direct file modifications with version control.
Workflow: Constant context-switching to a browser.	Workflow: Stays in the terminal; maintains development flow.

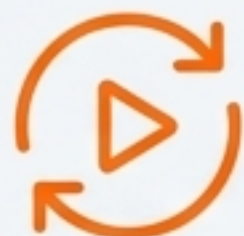
The terminal is the natural habitat for an agentic partner.

Terminal integration is not a matter of preference; it is essential to the agentic paradigm. It provides the foundation for trust and true collaboration.



Direct File System Access

It can read your ``src/`` folder, check ``requirements.txt``, and analyze Git history because it lives where your code lives.



Real-Time Execution

It runs tests, executes scripts, sees the error messages, and iterates on solutions in real-time.



Version Control Integration

It works with Git to create commits and suggest diffs, respecting your existing workflow.



Trust Through Transparency

You see the exact diffs and command outputs before approving anything. Every action action is explicit and visible.

First, we give our partner a persistent memory with CLAUDE.md.

Friction

Every new session starts with zero context. You waste time re-explaining your project's tech stack, directory structure, and coding conventions.

Solution

`CLAUDE.md` is a simple markdown file in your project root that Claude Code automatically loads at the start of every session. It is a persistent project brief.

```
# CLAUDE.md: Persistent Project Brief
## 1. Project Overview
This project is a FastAPI backend for...
## 2. Technology Stack
- Python 3.13, FastAPI, PostgreSQL
## 3. Directory Structure
- `src/` for application code
- `tests/` for pytest tests
## 4. Coding Conventions
- Google-style docstrings, type hints required.
## 5. Key Commands
- Run server: `uvicorn main:app --reload`
- Run tests: `pytest`
```

Specify your context once, and every future session starts with full understanding.

Next, we give our partner senses to access the outside world.

Friction

The AI is limited to your local files. It can't browse websites for research, check the latest library documentation, or query an API.

Solution

The **Model Context Protocol (MCP)** connects Claude Code to external tools and data sources in a standardized, safe way. Think of it as a phone directory of approved specialists.



Playwright MCP: Gives Claude the ability to browse and interact with websites.



Context7 MCP: Lets Claude fetch up-to-date documentation from knowledge sources.



GitHub MCP: Allows Claude to query GitHub repositories.



Database MCP: Enables Claude to safely query databases.

We build a team of specialists using Subagents.

Friction

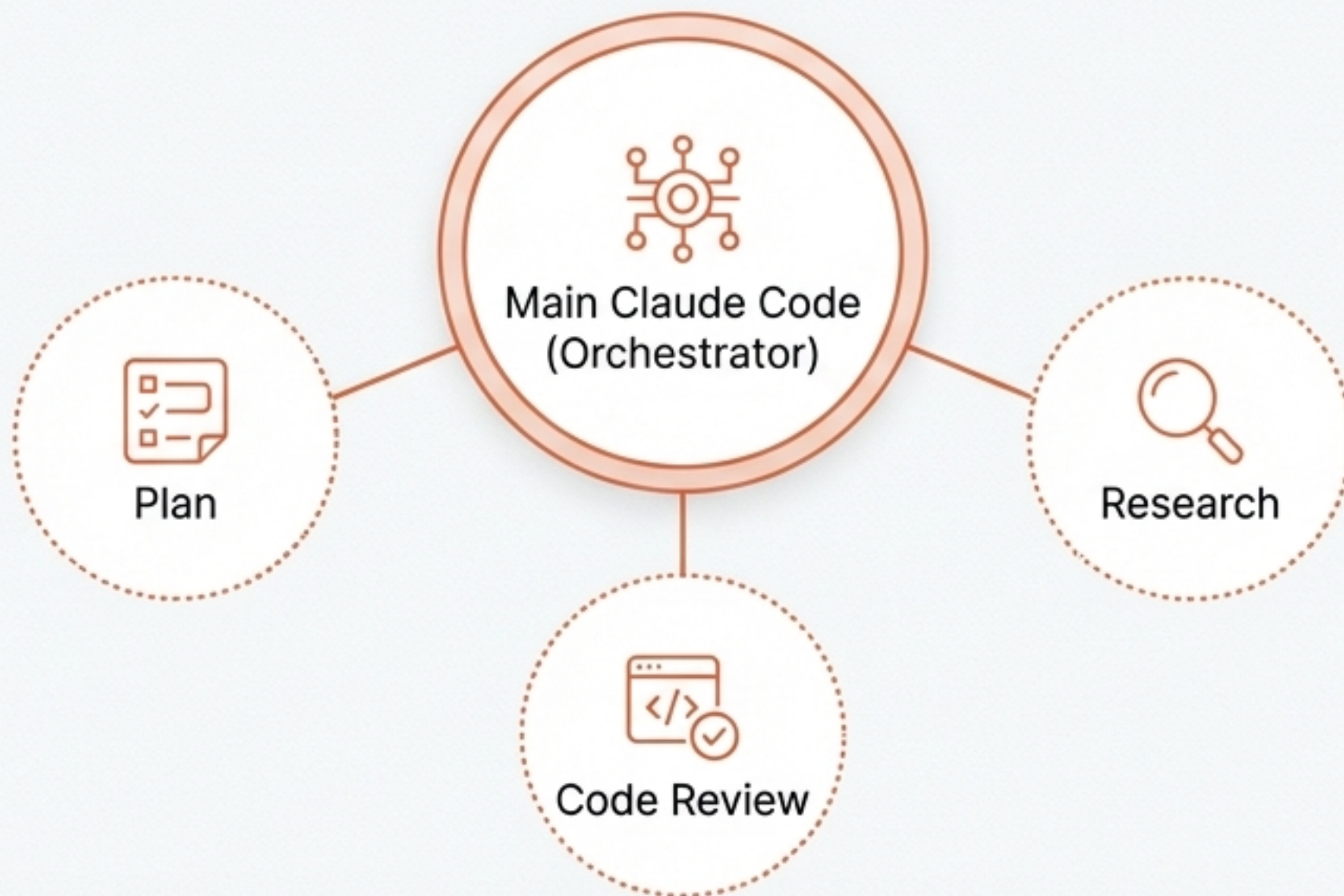
For complex tasks like “add user authentication,” a single conversation context becomes cluttered with research, plans, and code, leading to confusion and poor results.

Solution

Subagents are specialized AI assistants with their own instructions and isolated context windows. Claude Code acts as the orchestrator, delegating tasks to the right specialist.

Key Example: The built-in **Plan** subagent

1. It first researches your codebase to understand the current structure.
2. It creates a multi-phase strategic plan for your approval.
3. Only after approval does execution begin, step-by-step.





We teach our partner reusable capabilities with Skills.

Friction

You find yourself repeating the same instructions for common tasks, like “structure this blog post with 5 headlines and short paragraphs,” every single time.

Solution

An **Agent Skill** is a reusable capability that Claude Code discovers and applies autonomously when it recognizes a relevant context. You create a `SKILL.md` file once, and Claude applies its instructions automatically forever.

Subagents	Best for complex, isolated tasks that need guaranteed, explicit invocation.	
Skills	Best for lightweight, repeatable capabilities that benefit from automatic discovery.	

Skills transform one-time solutions into persistent intelligence. You are not just reusing code; you are **reusing reasoning patterns**.

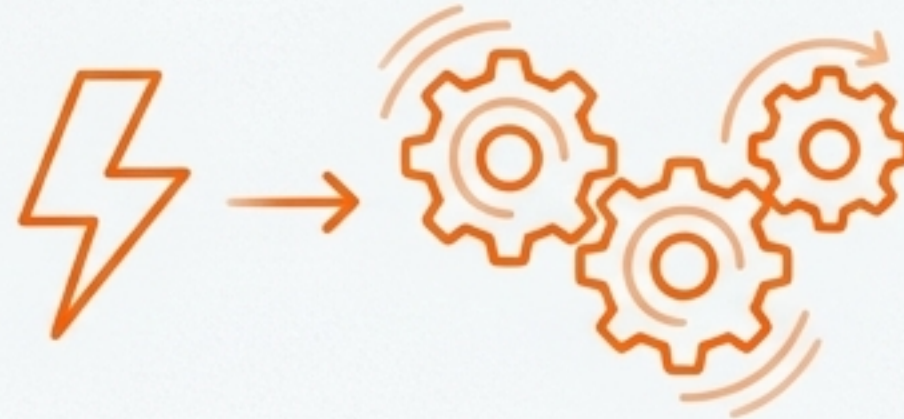
We automate the workflow with event-driven Hooks.

Friction

Your workflow is filled with repetitive follow-up actions: format code after an edit, run tests after a save, load environment variables at startup. These are manual, tedious, and easy to forget.

Solution

Hooks are automated scripts that run when specific events occur in Claude Code. They automate the predictable parts of your workflow.



Key Hook Events

SessionStart

Runs when you open a session.

Use case: Load environment variables and project context.

PostToolUse

Fires **after** a tool (like file editing) completes.

Use case: Automatically run a code formatter or linter on the changed file.

PreToolUse

Fires **before** a tool runs.

Use case: Validate inputs or check requirements.

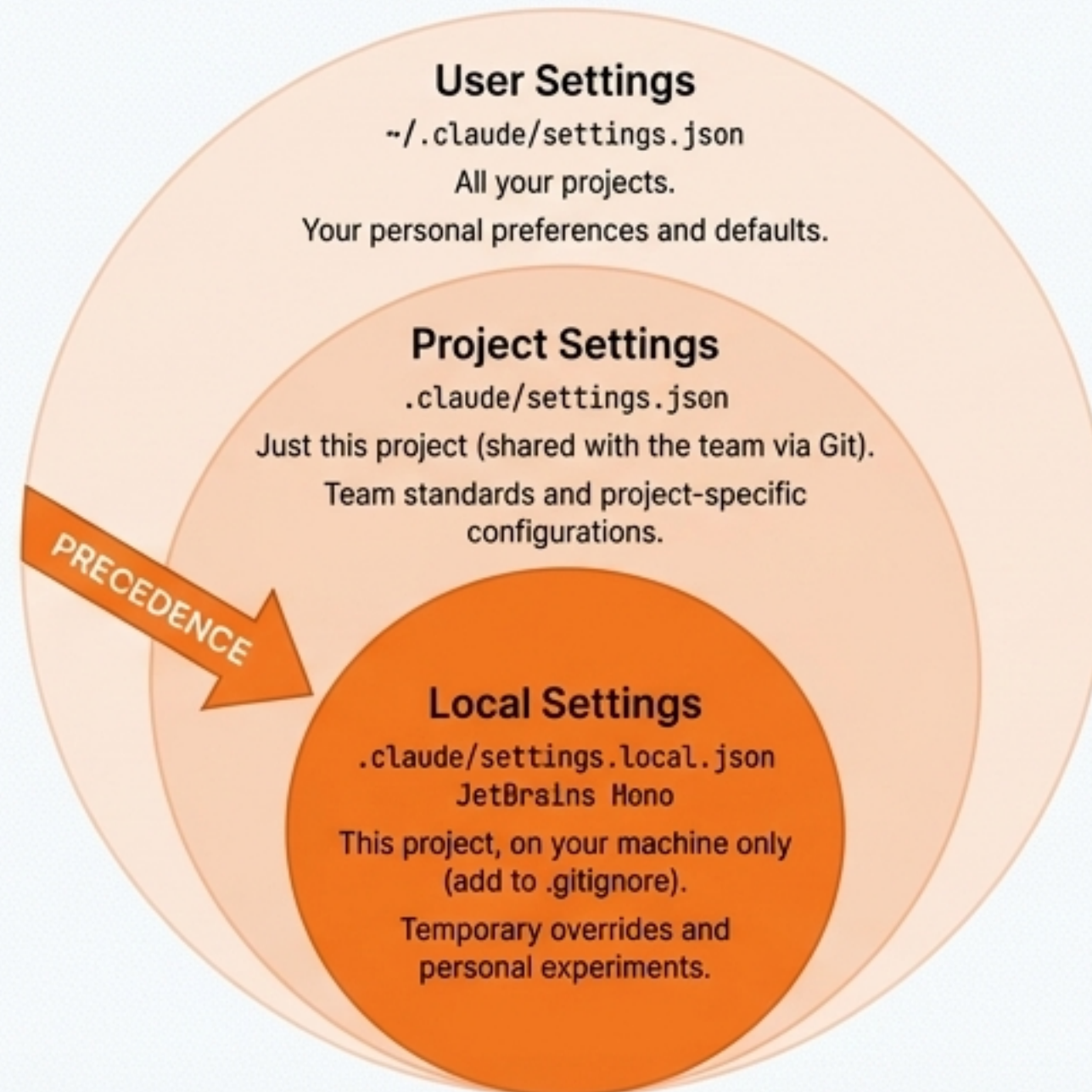
An intelligent partner adapts to its environment

Friction

How do you enforce team-wide security standards while allowing for personal preferences and temporary local experiments without creating configuration conflicts?

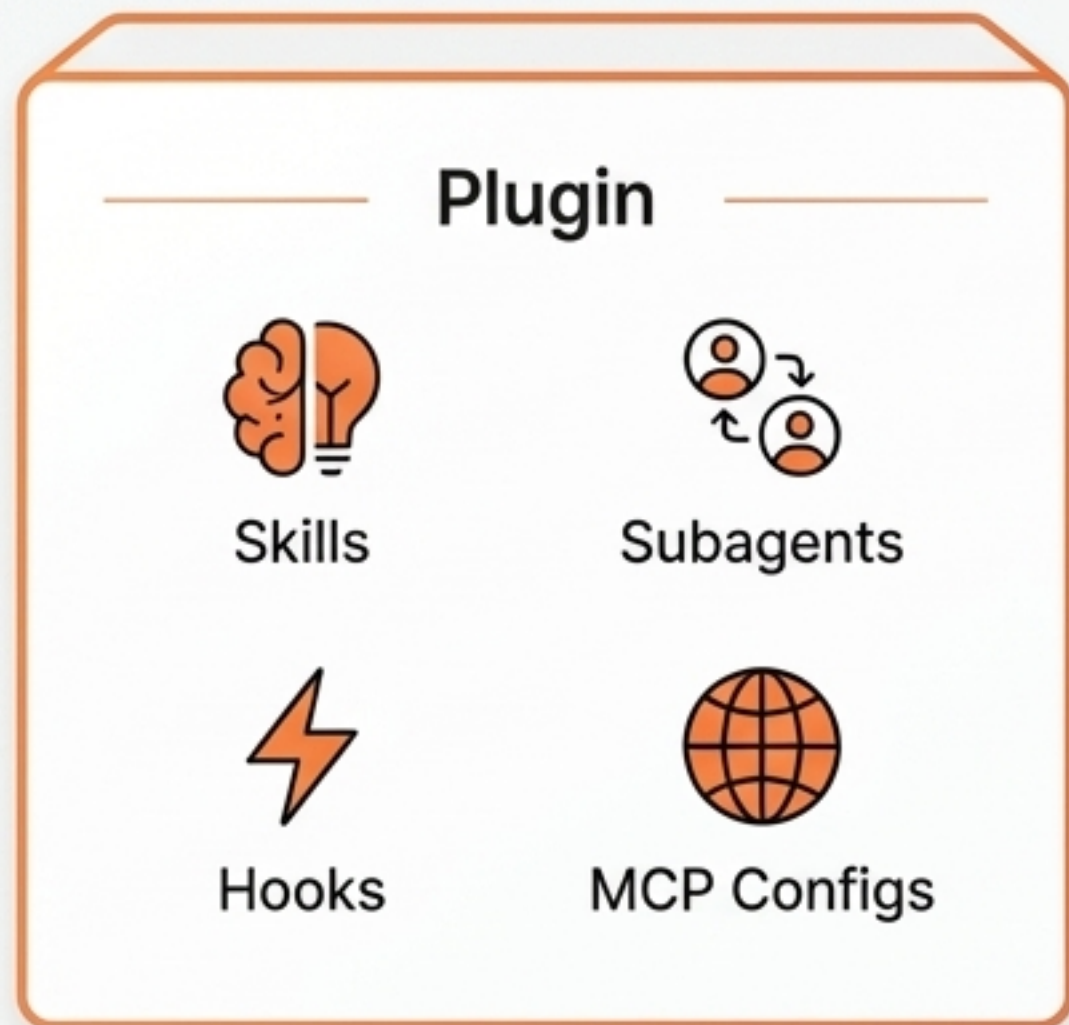
Solution

Claude Code uses a three-level settings hierarchy with a clear precedence order: Local > Project > User.



Finally, intelligence becomes composable and shareable via Plugins.

You don't have to build everything from scratch. A Plugin is a bundled package that can include Skills, Subagents, Hooks, and MCP configurations, created by Anthropic and the community.



Getting Started

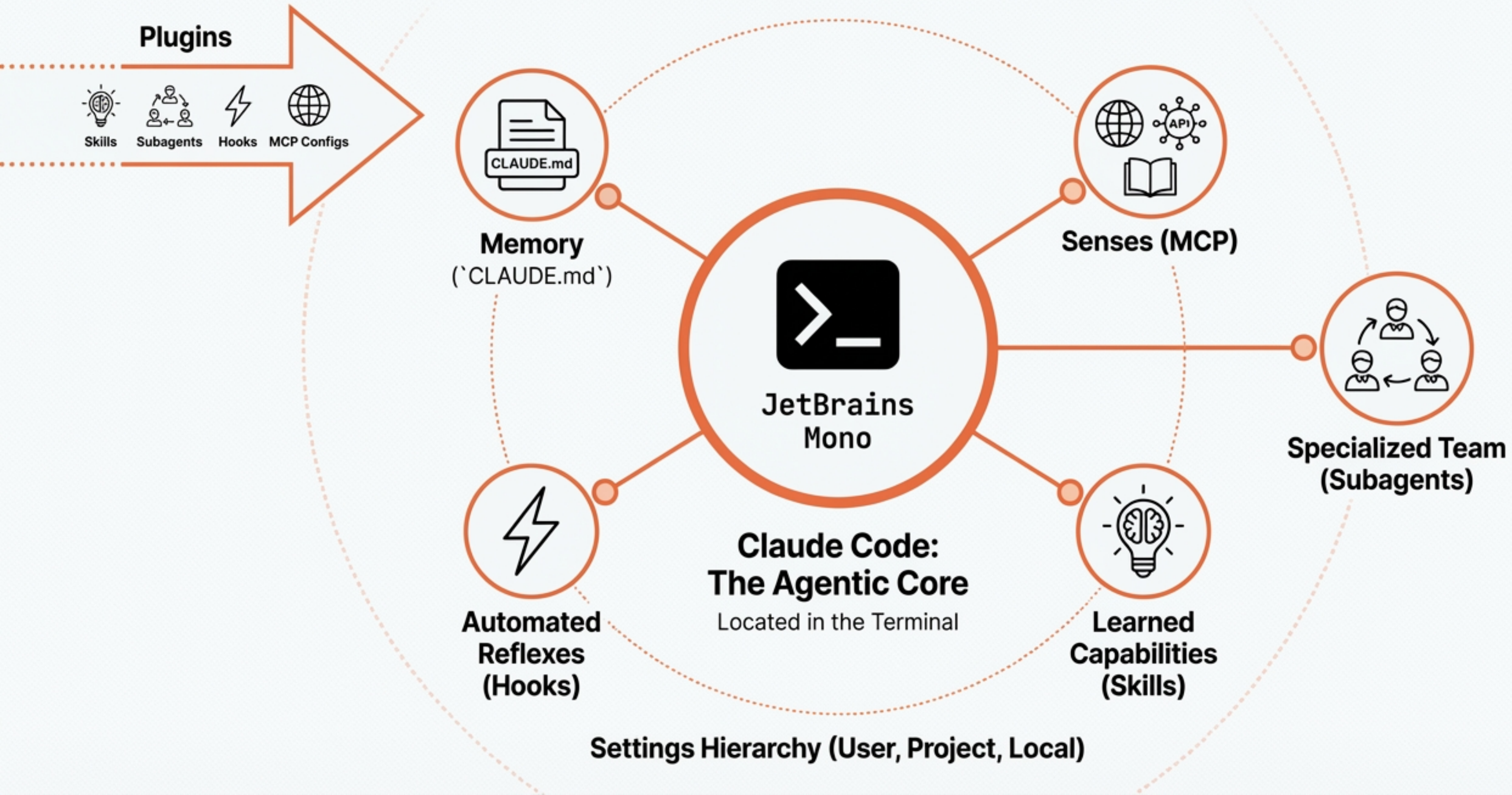
1. Discover plugins in marketplaces like **Anthropic's official skills repository**.
2. Connect to a marketplace with a simple command:

```
claude /plugin marketplace add anthropic-agent-skills  
https://github.com/anthropics/skills
```

3. Install a bundle of capabilities:

```
claude /plugin install anthropic-agent-skills/example-skills
```


The Complete Architecture of an AI Partner



The paradigm shift is from writing code to orchestrating intelligence.

The most powerful AI assistance isn't about replacing developers. It's about removing the friction that slows them down:

- Context-switching
- Manual integration
- Repetitive setup
- Isolated problem-solving

Agentic tools like Claude Code handle the friction—the tedious, error-prone, context-heavy tasks—so you can focus on the creative work of designing solutions. We are moving from “AI as a tool you consult” to “AI as an integrated part of your development environment.”