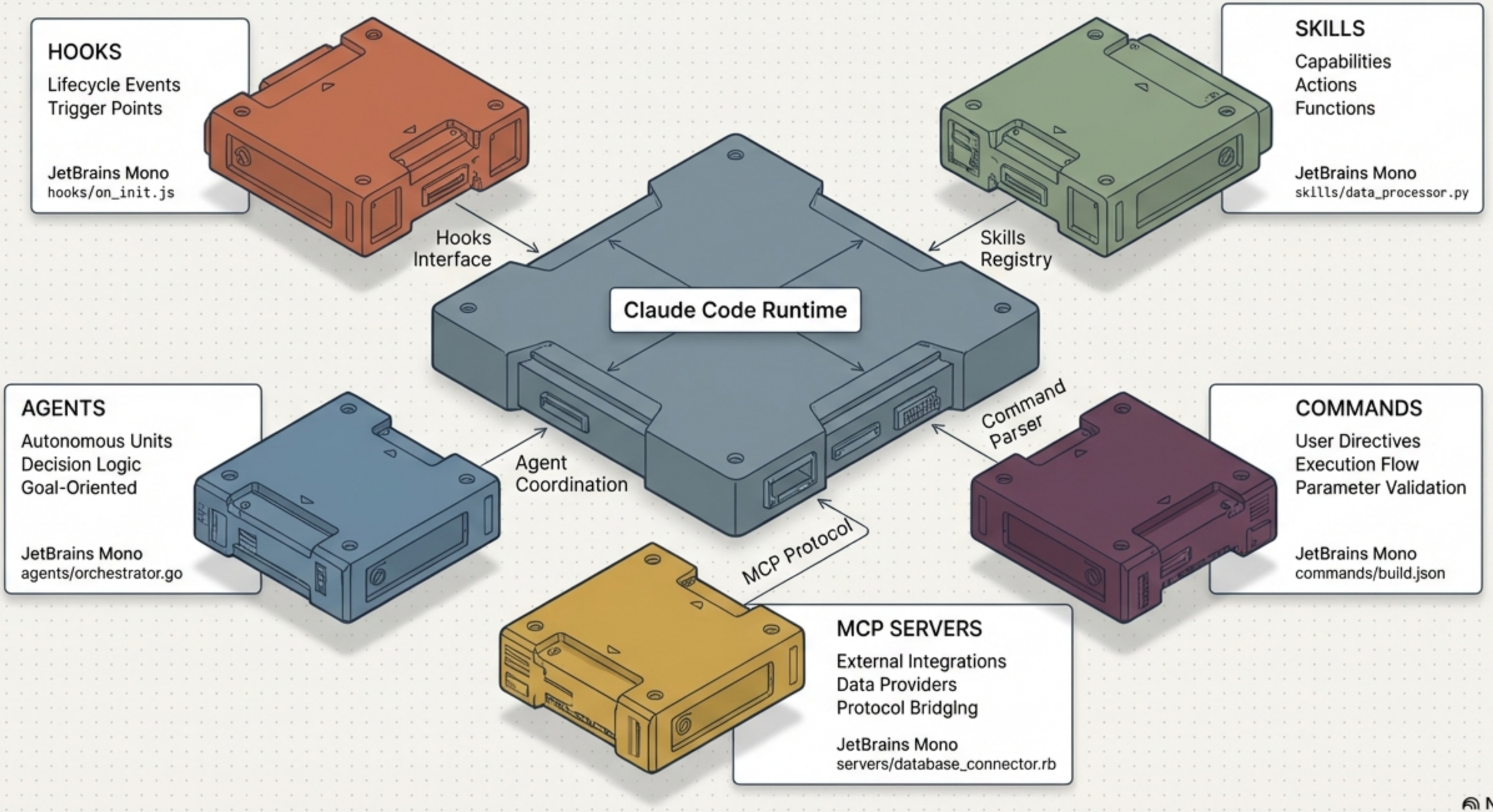
Building Your First Plugin: The CC-203 Architecture Guide

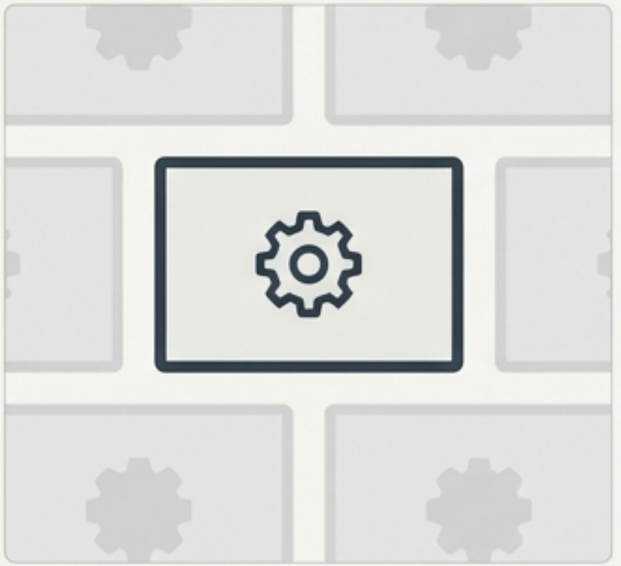
From Manifest to Orchestration: Mastering the Claude Code Mastery Assessment



NotebookLM

The Three Pillars of Plugin Execution

Isolation Diagram



Isolation

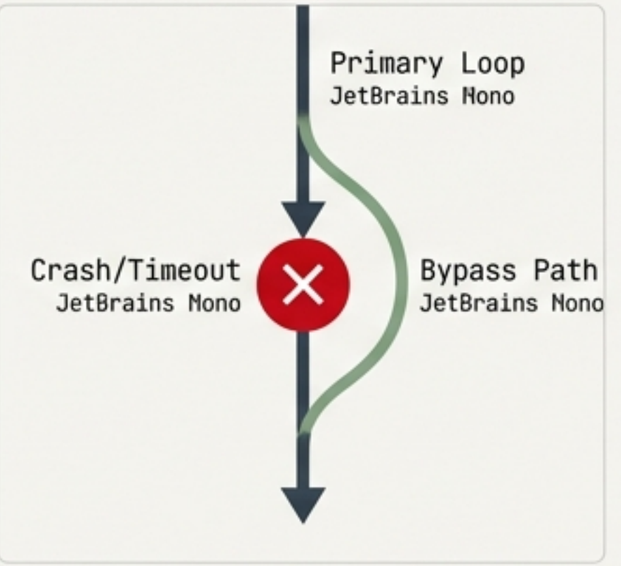
No shared state. No cross-plugin function calls. A crash in Plugin A never affects Plugin B.

Composability Diagram

Composability

Plugins run parallel to the core system. A governance plugin and a docs plugin operate simultaneously on different events without conflict.

Fail-Open Safety Diagram

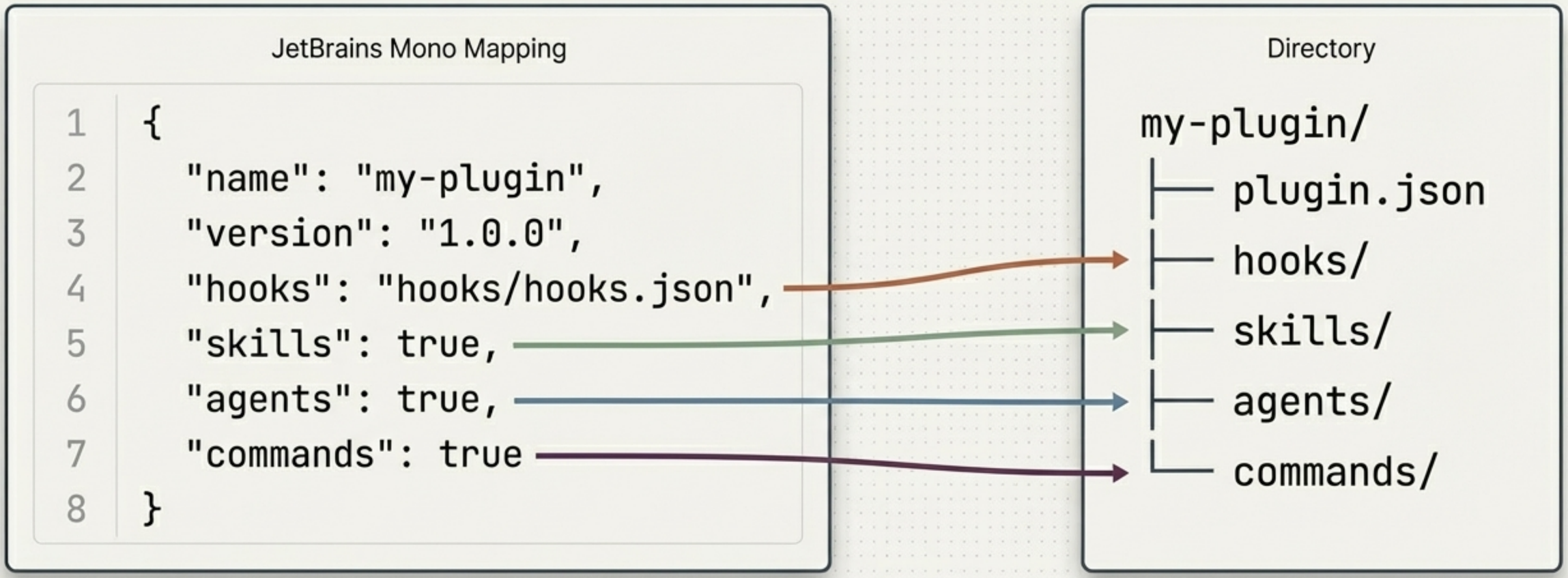


Fail-Open Safety

If a plugin hook crashes or times out, Claude Code proceeds as if the hook returned “allow”. Plugins enhance the system; they never break the primary loop.

NotebookLM

Anatomy of a Plugin: The Manifest and the Directory Tree



Convention over Configuration: Drop files in the right directories, set the manifest flag, and they are discovered automatically. No boilerplate required.

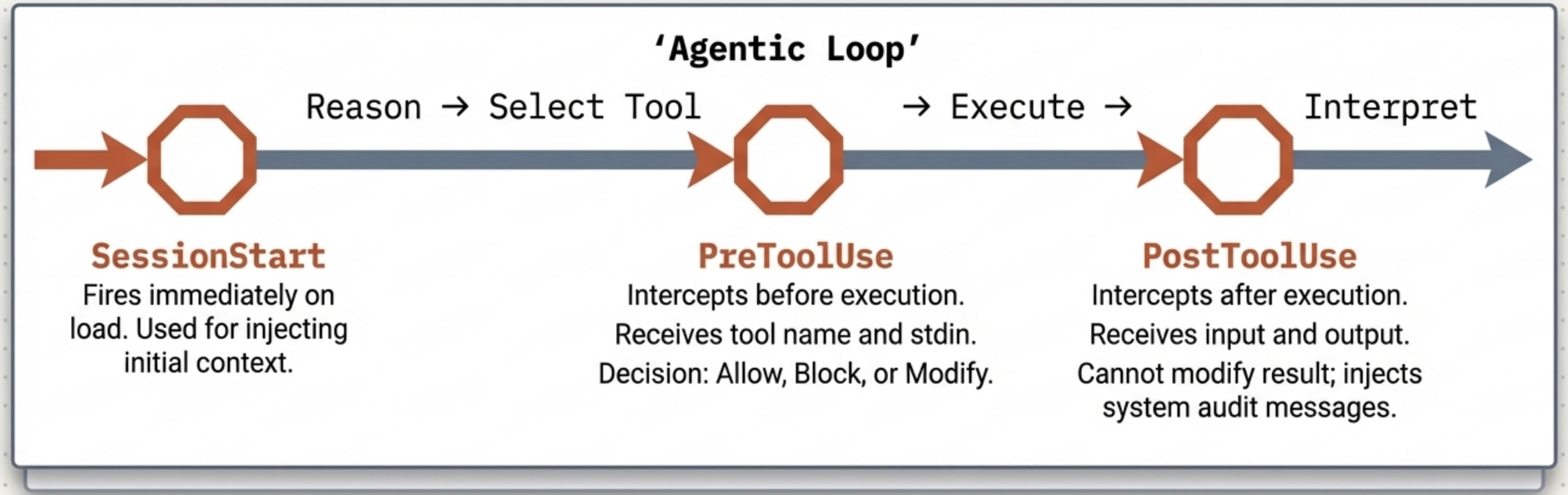
NotebookLM

The Component Diagnostic Matrix

Component	Trigger	Execution	Best For
Hooks	Automatic / Event-Driven	Silent / Background	Guardrails, auditing, automated validation without user input.
Skills	On-Demand (Context Match)	Guided / Step-by-Step	Specialized knowledge, loaded-on-demand checklists, process standardizations.
Agents	Dispatched via Task Tool	Autonomous / Parallel	Isolated sub-tasks requiring dedicated focus and their own context windows.
Commands	Manual (User Slash Command)	Interactive	User-invoked shortcuts, explicit multi-step macros.
MCP Servers	Automatic / Tool Call	External Connection	Bridging the sandbox to external silos (Jira, databases, custom APIs).

NotebookLM

Component 1: Hooks (Automated Interception)

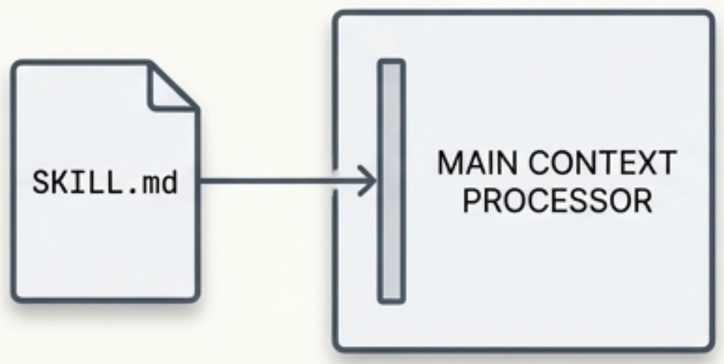


Hooks must be lightweight, fast, and silent. They are the only component that can outright block a Claude Code action.

NotebookLM

Component 2 & 3: Differentiating Knowledge from Autonomy

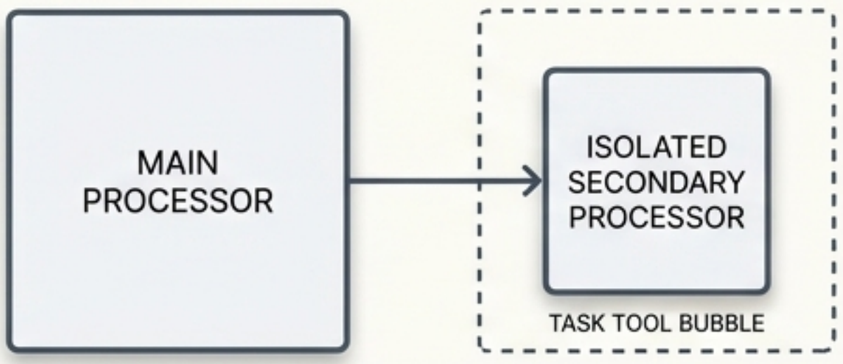
Skills (Passive Knowledge)



Mechanism:
Loaded dynamically into the main context window.

Usage:
Guided workflows and checklists: "Here is the 5-step process for code reviews. Follow it."

Agents (Active Workers)



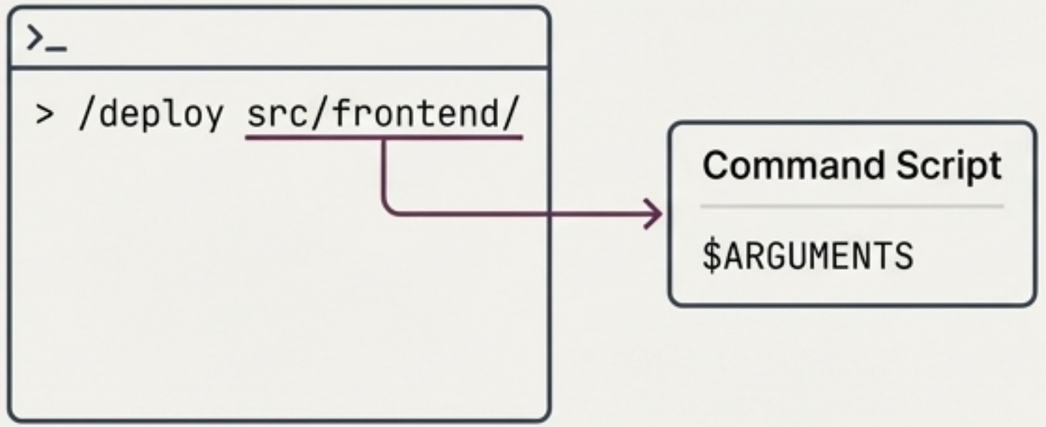
Mechanism:
A defined role dispatched via the Task tool, operating in its own isolated bubble.

Usage:
Autonomous parallel execution: "Take this specification, go into a separate context, write the code, and return the file."

NotebookLM

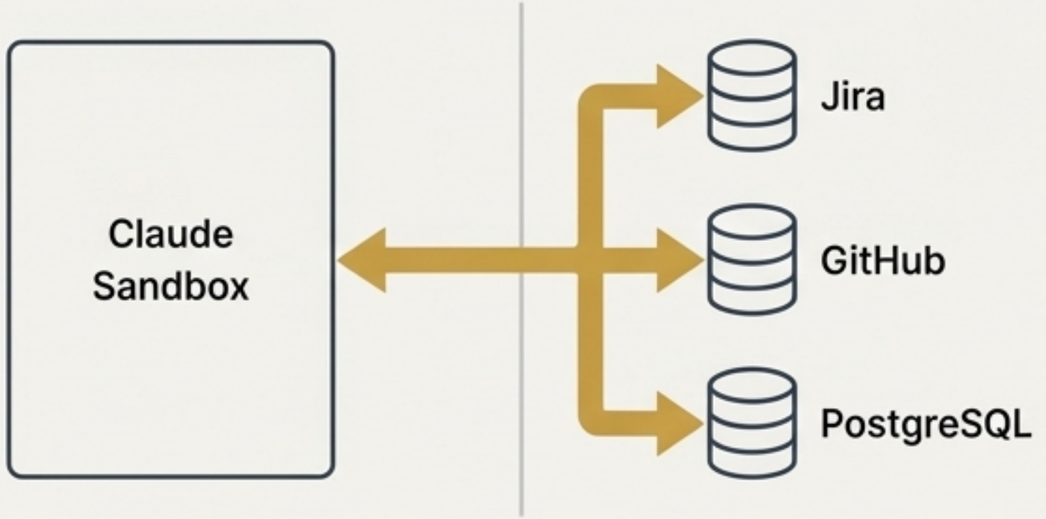
Component 4 & 5: User Invocations and External Bridges

Commands (User Invocation)



User-initiated slash commands trigger explicit commands, passing dynamic text directly into the command body via the $ARGUMENTS variable.

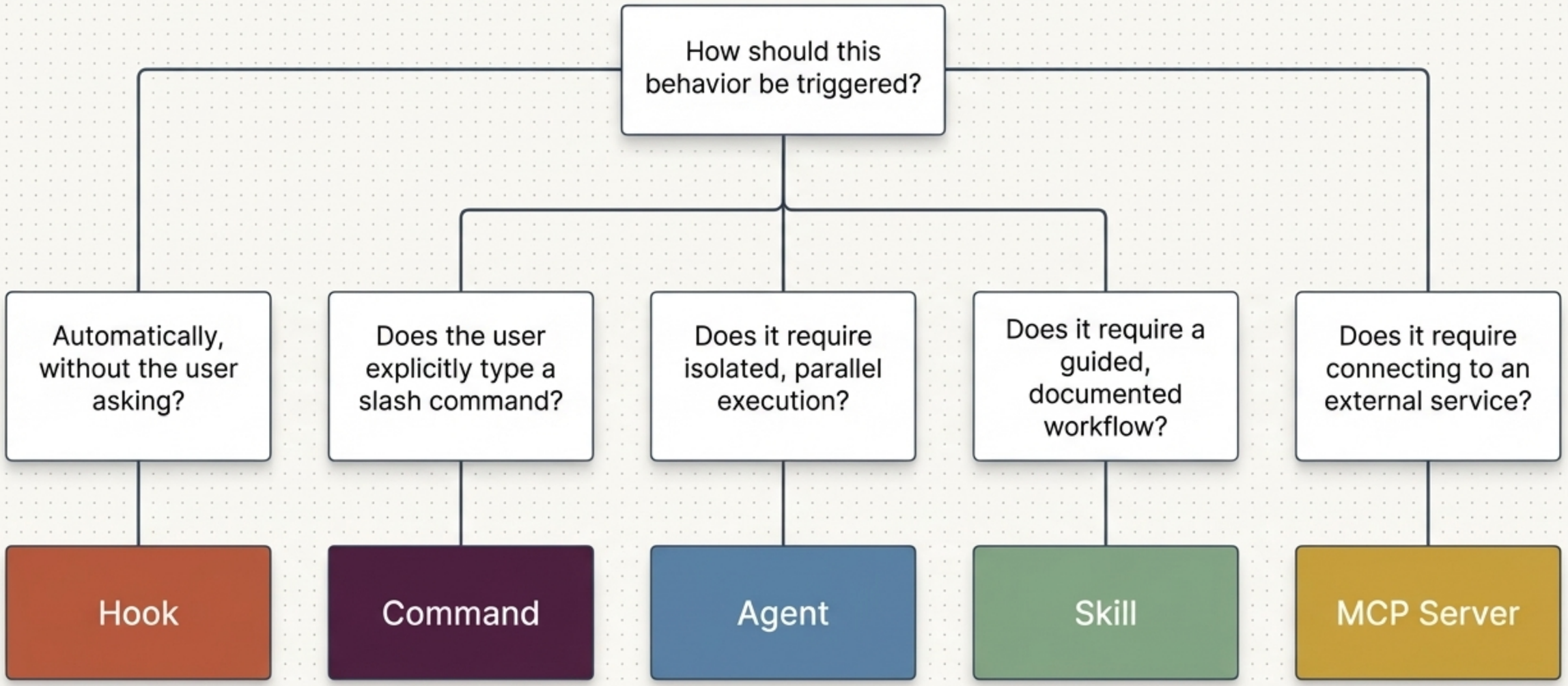
MCP Servers (External Bridges)



Model Context Protocol (MCP) servers expose external systems as callable tools that Claude can use natively, bridging the isolated sandbox to external silos.

NotebookLM

The Component Decision Tree



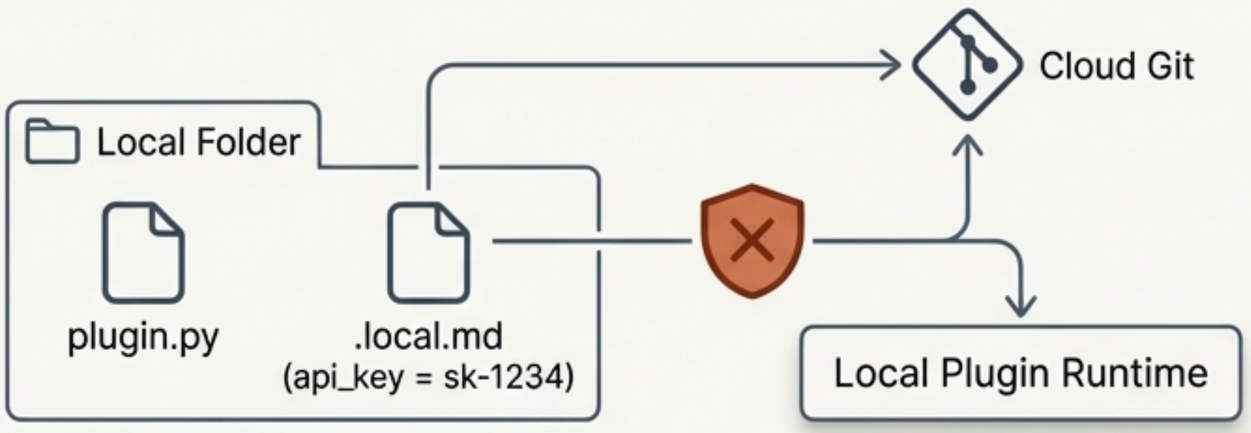
NotebookLM

Portability & Security Constraints

The Portability Fix

Never hardcode absolute paths. Use ${CLAUDE_PLUGIN_ROOT} to ensure cross-machine portability.

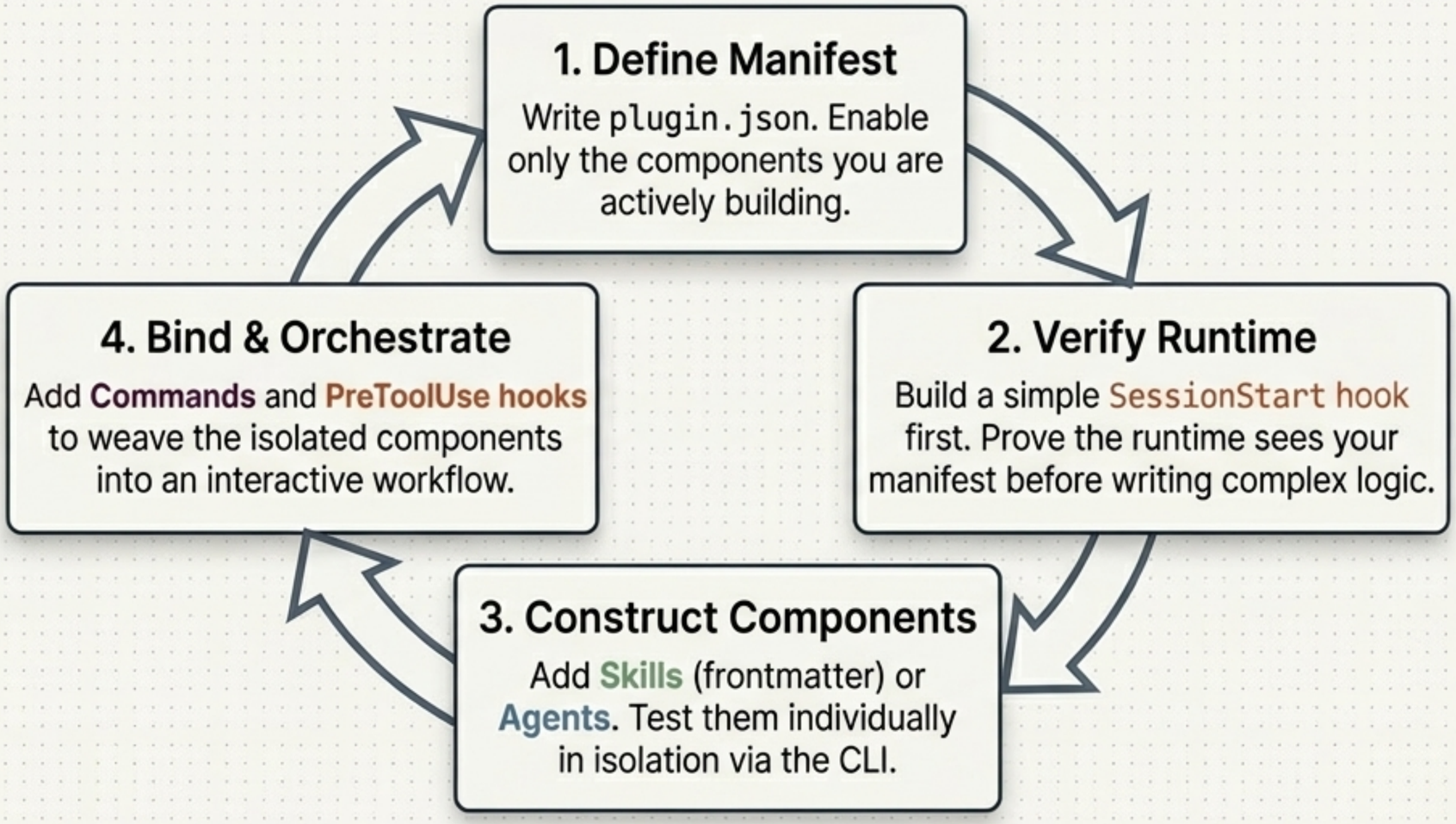
The .local.md Pattern



Store per-environment API keys and feature flags in .local.md. It is strictly .gitignored and ingested dynamically at runtime.

NotebookLM

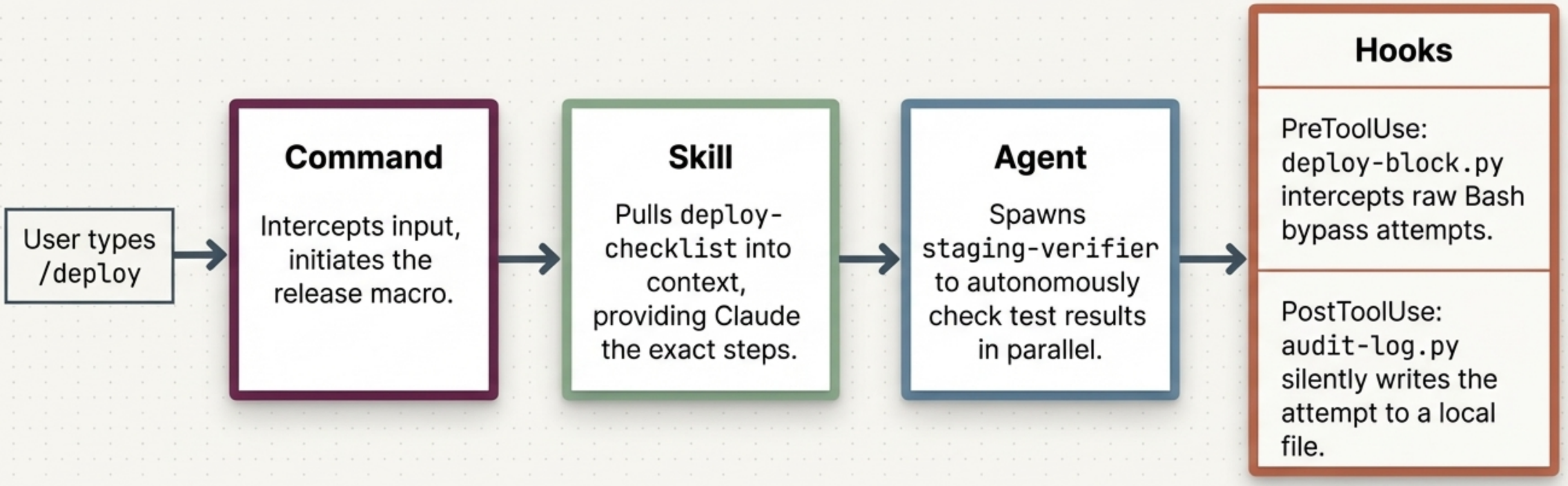
The Iterative Build Workflow



Architectural Rule: Never build the entire plugin at once. Build, test in isolation, bind, repeat.

NotebookLM

System Synthesis: **The Deploy-Guard Ecosystem**



NotebookLM

CC-203 Assessment: The Scenario Briefing

Objective: Build the ‘Deploy-Guard’ plugin for a web-app deployment team.

1. **Automatic Block**: Prevent all deployments on Fridays and weekends (no user action required).
2. **Guided Process**: Provide a strict deployment checklist for the team to follow.
3. **User Interface**: Offer a `/deploy` command that executes the deployment sequence.
4. **Autonomous Verification**: Include an agent to verify staging tests independently.
5. **Compliance Logging**: Audit log every single deployment attempt (both successful and blocked) to a local file.

NotebookLM

Assessment Deliverables: The Required Architecture

☐ plugin.json		The complete manifest
☐ ARCHITECTURE.md		Component justification document
☐ hooks/		
☐ hooks.json		Registration mapping
☐ hooks/deploy-block.py	Rust Terracotta block	PreToolUse hook for Friday blocking
☐ hooks/audit-log.py	Rust Terracotta block	PostToolUse hook for logging
☐ skills/		
☐ skills/deploy-checklist/SKILL.md	Sage Green block	Checklist with frontmatter
☐ agents/		
☐ agents/staging-verifier.md	Steel Blue block	Agent prompt + model selection
☐ commands/		
☐ commands/deploy.md	Deep Plum block	The /deploy command logic

NotebookLM

The 50-Point Grading Rubric

Criteria		Points
Hook Implementation	Rust Terracotta	10 pts
Hook Registration	Rust Terracotta	8 pts
Skill Content	Sage Green	7 pts
Manifest Correctness		5 pts
Directory Structure		5 pts
Agent Definition	Steel Blue	5 pts
Command Definition	Deep Plum	5 pts
Component Justification		5 pts

Pass Threshold: 40/50 (80%)

⚠ **+5 Bonus Points:** Identify the critical security flaw in the fail-open design of `deploy-block.py`. (Hint: What happens to the deployment if the Python script crashes on a Friday? How do you mitigate this?)

NotebookLM

Pre-Flight Validation Checklist

- [ ] Is `plugin.json` strictly valid JSON with no trailing commas?
- [ ] Do all Python hook scripts use `${CLAUDE_PLUGIN_ROOT}` for file paths?
- [ ] Do all .md components (Skills, Agents, Commands) feature valid YAML frontmatter?
- [ ] Is the fail-open security risk in the blocking hook explicitly mitigated?
- [ ] Does the plugin load into Claude Code without runtime errors?

Architecture verified. Initialize your repository.

NotebookLM