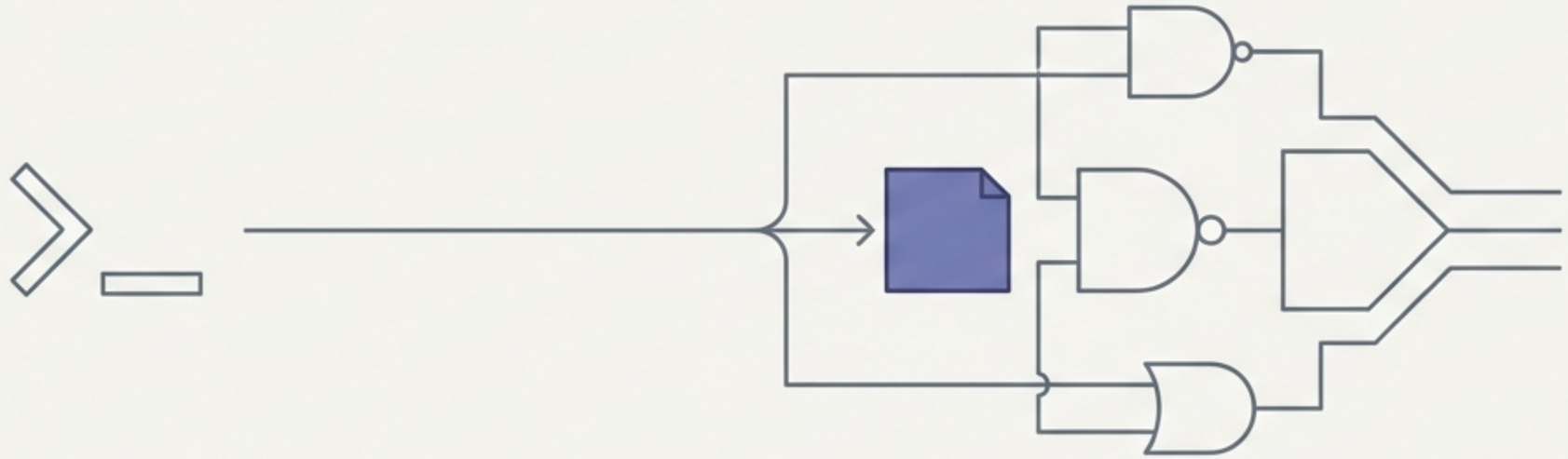


CC-202 Assessment: Hook System Deep Dive

Mastering automated event reactions, execution lifecycles, and architectural guardrails in Claude Code.

Hooks Operate as Automated, Invisible Interventions

User-Initiated / Visible

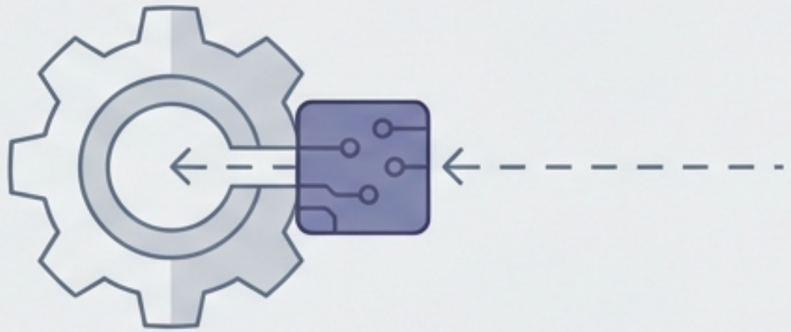
Commands & Skills

```
>_ /deploy


Starting build process...
[INFO] Validating configuration...
[INFO] Pulling latest changes...
[INFO] Compiling source code...
[INFO] Running tests...
[SUCCESS] Build complete. Deploying to staging...
[INFO] Deployment successful.
```

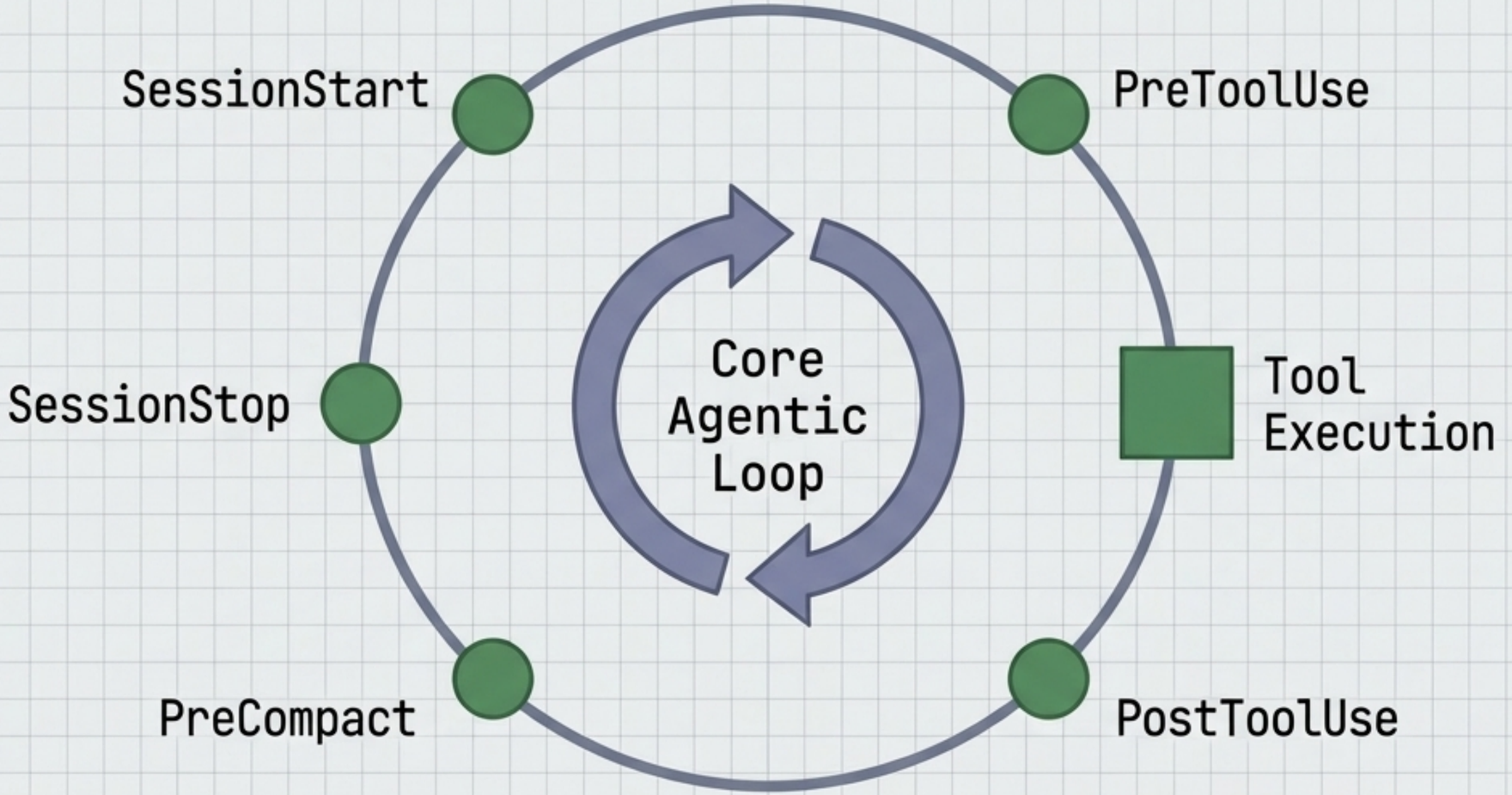
Automated / Invisible

Hooks



If it requires human invocation, it’s a Command. If it happens automatically in response to a system event, it’s a Hook.

The 7-Event Lifecycle Intercepts the Agentic Loop



Key Insight: Hooks literally pause the Agentic Loop. Claude Code halts its operation, waits for the hook’s execution and JSON decision, and only then proceeds with the worklow.

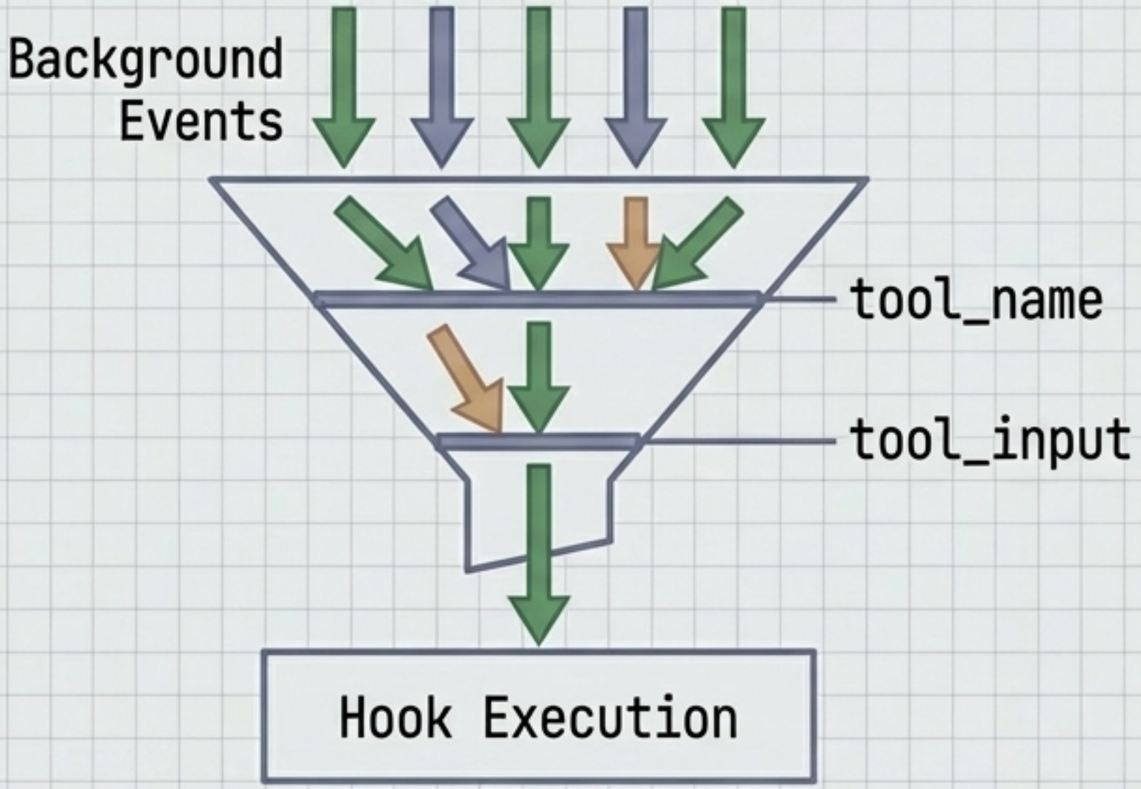
Hooks Execute as External Black-Box Scripts

Hooks are external executables entirely agnostic to Claude Code's internal memory. They communicate strictly via standard input/output streams.

The script must exit with valid JSON. Standard exit codes (0 vs 1) handle critical script-level failures.

Precision Matchers Prevent Unnecessary Execution

The Matcher Funnel



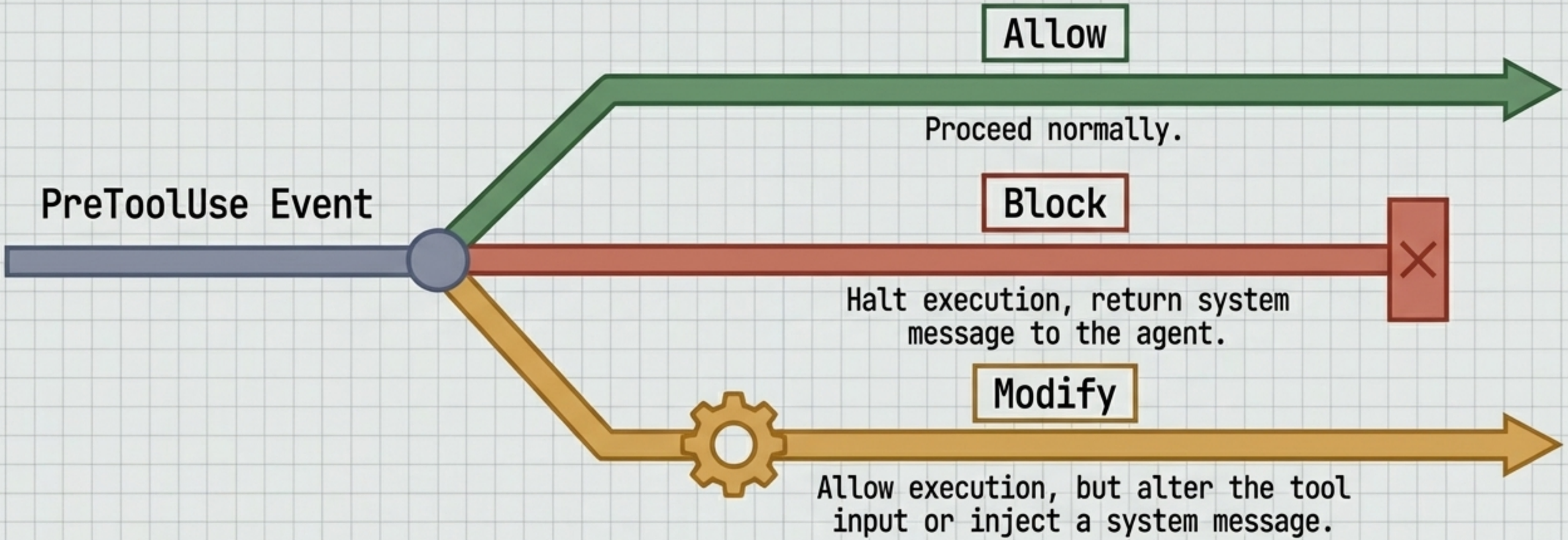

```
{
  "events": ["PreToolUse"],
  "matchers": [
    {
      "tool_name": "Bash",
      "tool_input": ".*git push.*"
    }
  ],
  "executable": "./git-guard.py"
}
```

Precise matchers are an architectural requirement. Triggering a hook on every tool use wastes computational time and risks polluting the agent's context.

Choosing Between Command and Prompt Execution Environments

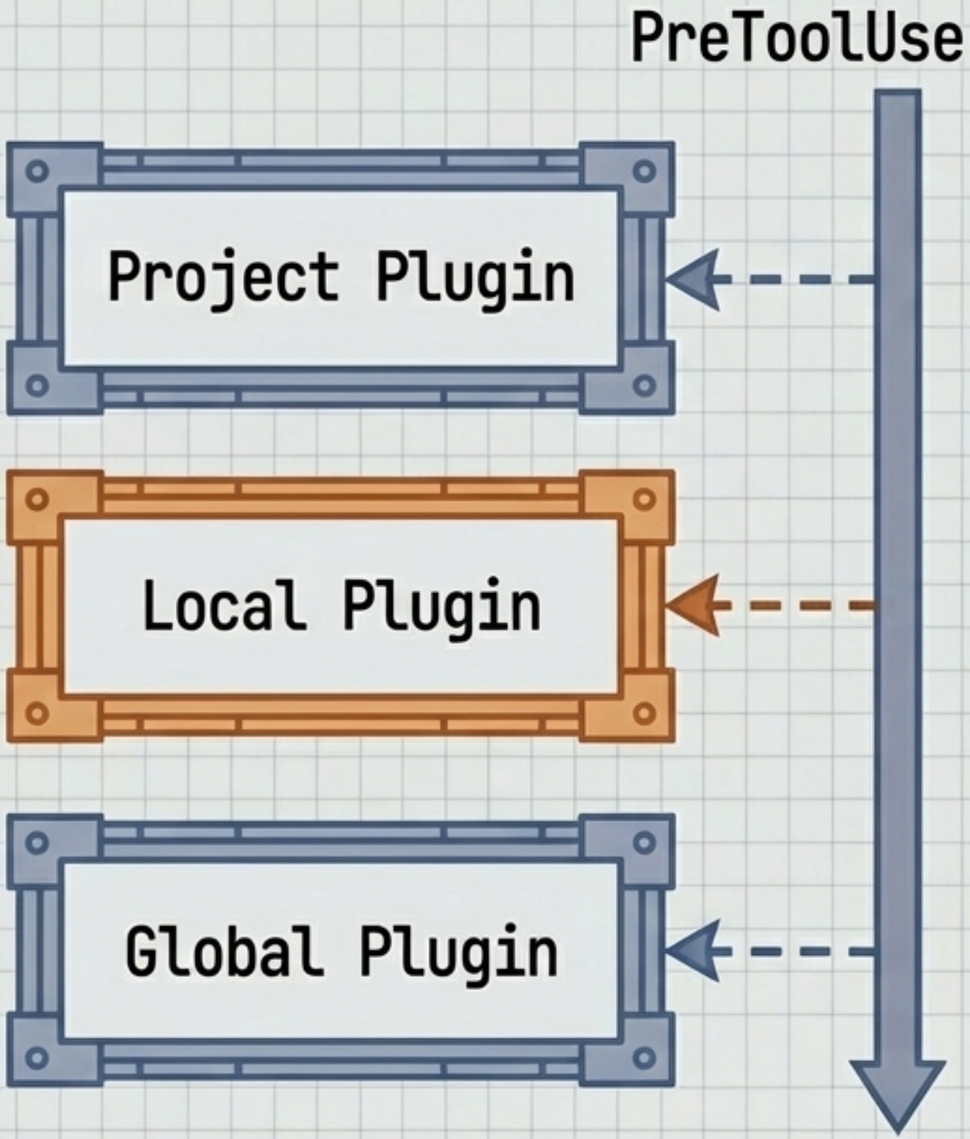
	Command Hooks	Prompt Hooks
Format	Executable scripts (.py, .sh)	LLM-evaluated prompts
Data Flow	Processes raw stdin / stdout	Uses conversational context window
Logic	Deterministic, rules-based, exceptionally fast	Probabilistic, semantic understanding, slower
Best For	Regex validation, standard logging, security blockers	Intent analysis, tone checking, complex code reviews

PreToolUse Decisions Dictate Downstream Execution Flow



Crucial Distinction: PostToolUse hooks CANNOT modify the tool result. They can only observe the output and inject system messages. Only PreToolUse can block or modify the action itself.

Independent Chaining Means No Conflict Resolution



1. Discovery Hierarchy

Plugins load in strict order: Project -> Local -> Global.

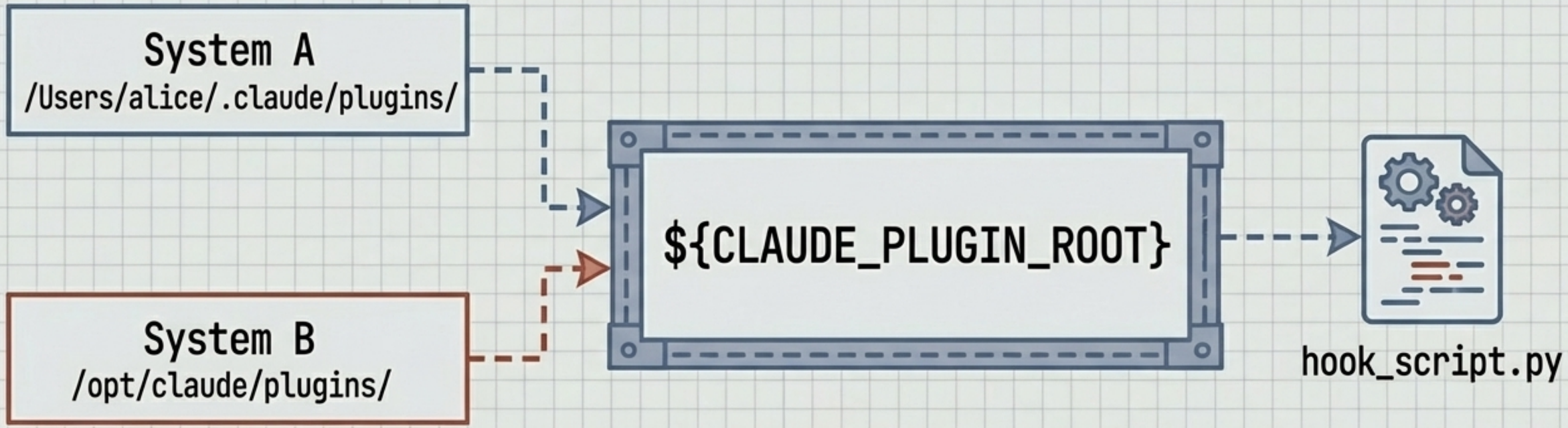
2. Registration Order

Within that hierarchy, hooks fire in the order they are registered in hooks.json.

3. Absolute Execution

If two hooks target the same event, both execute. Claude Code does not resolve conflicting logic between independent plugins.

The Root Variable Guarantees Cross-System Portability

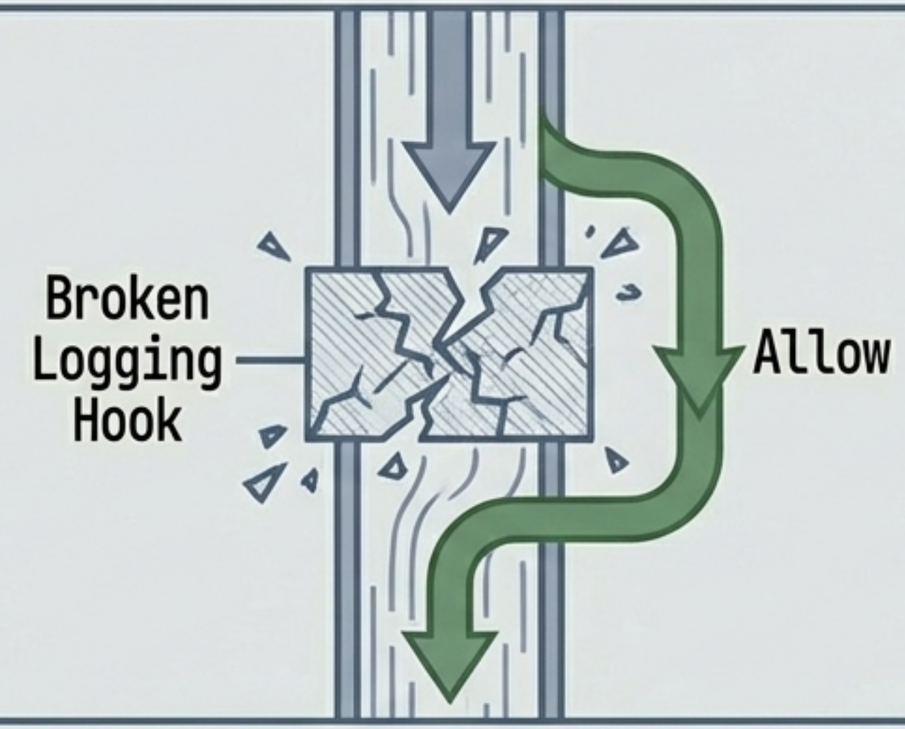


A hook script may execute from any unpredictable current working directory where the user launched Claude Code.

Assessment Rule: Hardcoding paths breaks isolation. Always construct absolute paths referencing local config files via ${CLAUDE_PLUGIN_ROOT}.

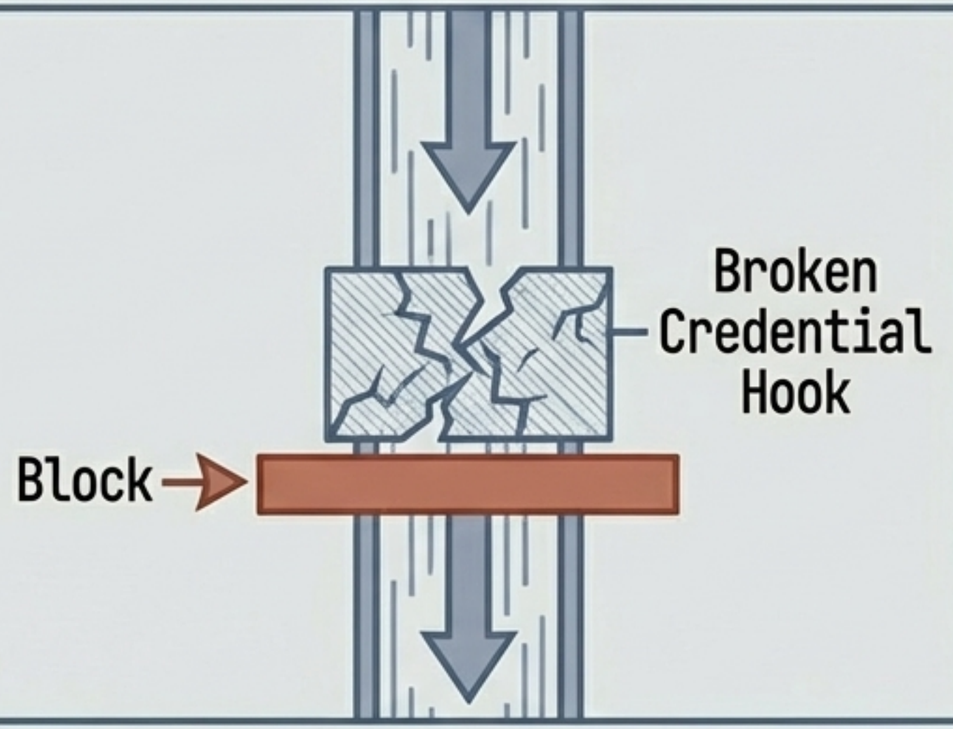
The Fail-Open Default Prevents Single Points of Failure

Fail-Open (System Default)



Traffic routes around the crash.

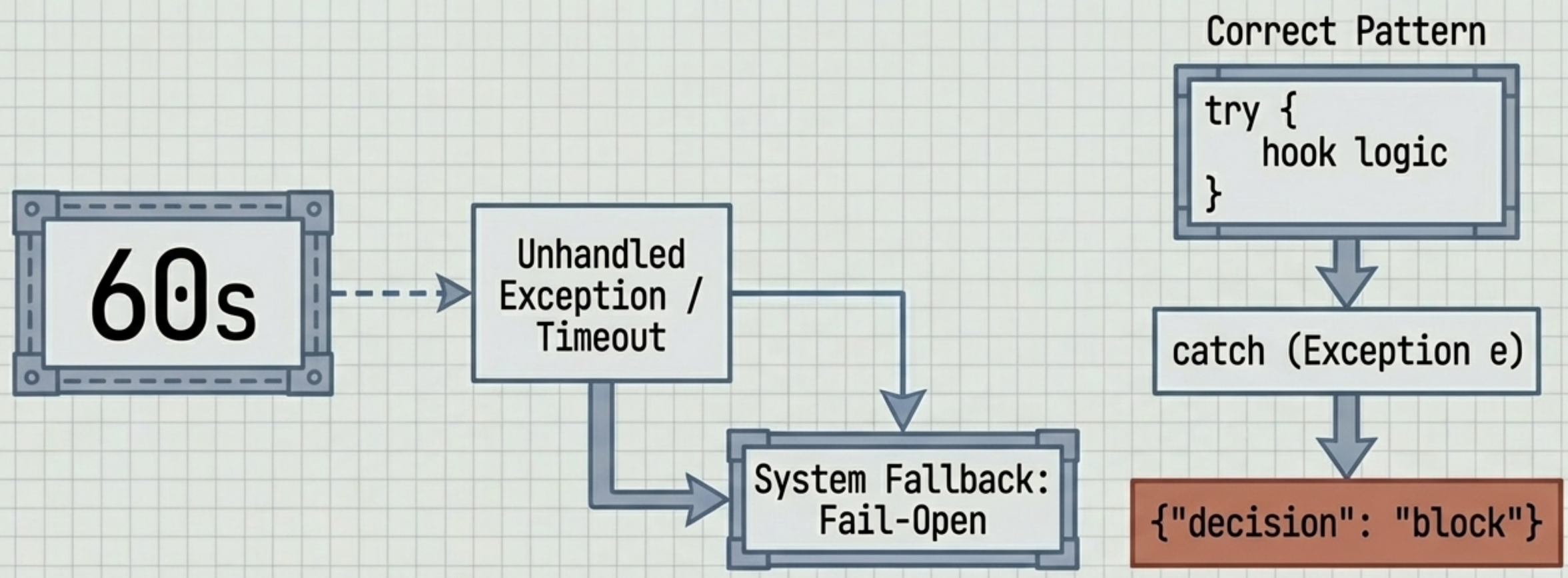
Fail-Closed (Engineered Security)



Traffic is halted entirely.

By default, if a hook times out or crashes, Claude Code assumes an ‘Allow’ decision so workflows are not disrupted. Fail-Closed behavior must be explicitly engineered into the hook’s error-catching logic.

Timeouts and Unhandled Errors Trigger Safe Fallbacks



- The system enforces a strict 60-second default timeout per hook.
- Any unhandled script exception triggers the automatic Fail-Open fallback.
- The Correct Pattern: Wrap logic in try/catch blocks and return a formatted JSON block decision if a fail-closed posture is needed.

Four Architectural Anti-Patterns to Master for CC-202

	[WARN] Violating Fail-Open: Blocking the user's session over non-critical hook failures (like a downed logging API).
	[WARN] I/O Negligence: Forgetting to parse stdin to read the tool name and input before executing hook logic.
	[WARN] Temporal Impossibility: Attempting to alter a tool's execution result from inside a PostToolUse hook.
	[WARN] Path Fragility: Hardcoding configuration paths instead of dynamically anchoring them to ${CLAUDE_PLUGIN_ROOT}.

Master these four, and you master the CC-202 architectural assessment.