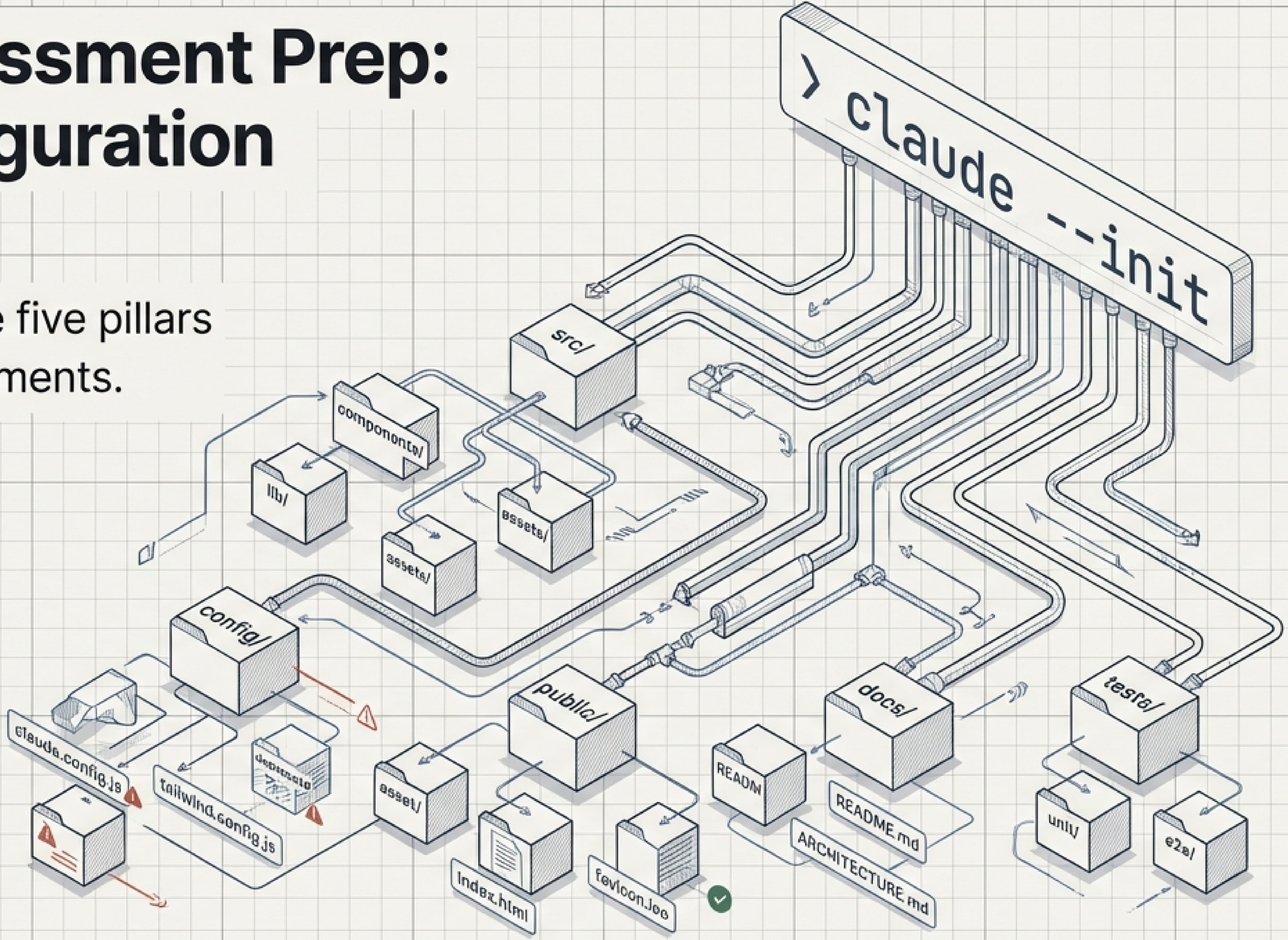
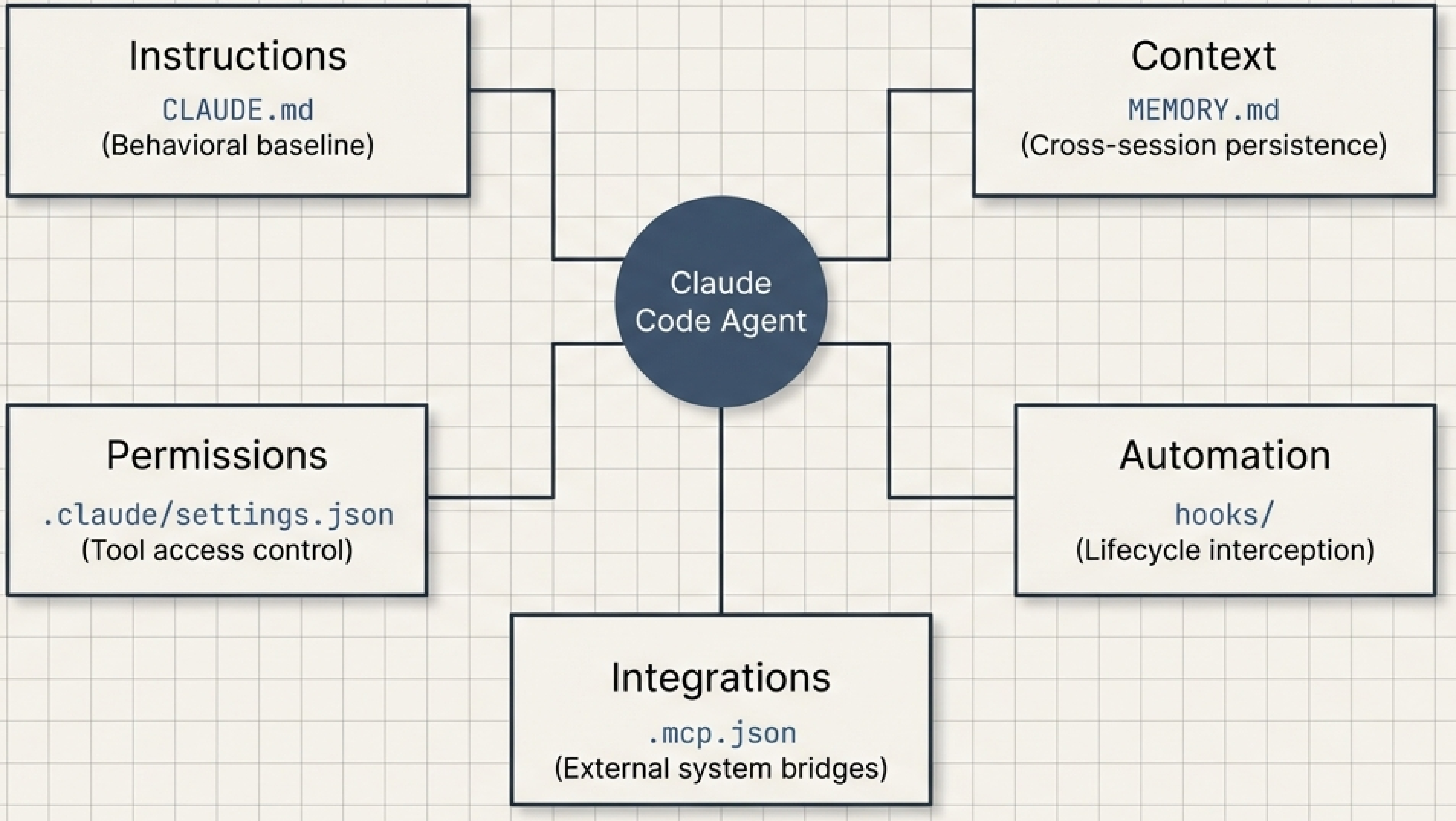
CC-103 Assessment Prep: Project Configuration Mastery

A structural guide to the five pillars of Claude Code environments.



The Five Pillars of Project Configuration



Pillar 1: Instruction Engineering (CLAUDE.md)

❌ Anti-Pattern: Verbose Persona

```
You are a helpful expert
developer who writes very good
code. Please make sure to use
TypeScript and if you can,
try to avoid using console.log
for debugging.
```

✅ Best Practice: Hard Rules

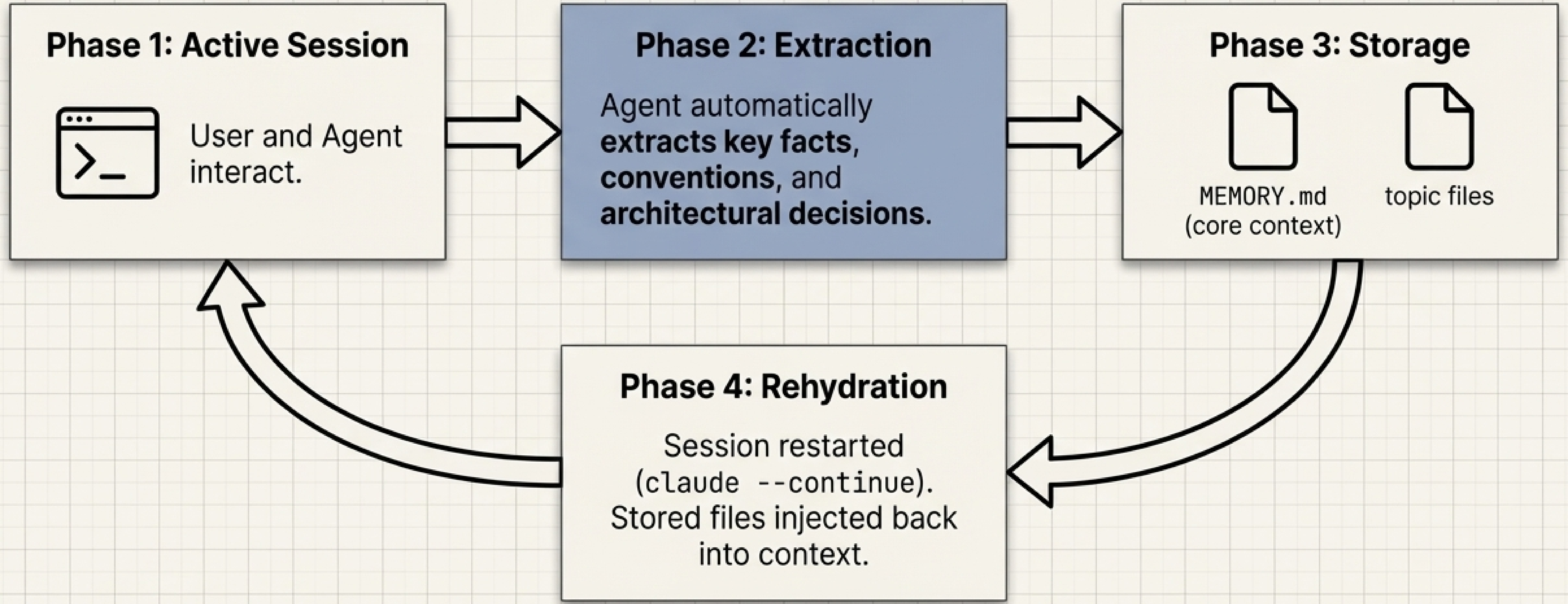
```
- Always use TypeScript.
- Never use console.log; use
  the custom logger.
- Enforce strict typing on all
  interfaces.
```

Claude Code operates optimally with 5-6 rigid, unambiguous rules. Verbose personas dilute instruction weight and waste context window.

Scope Precedence: Global vs. Project Settings

	Global Scope (`~/.claude/CLAUDE.md`)	Project Scope (`./CLAUDE.md`)
Location	User home directory	Root of the current repository
Impact Radius	Affects every Claude Code session on the machine	Affects only the current specific project
Precedence	Overridden by project rules	Absolute highest priority
Ideal Use Case	User-specific preferences (e.g., 'Always use vim keybindings in output')	Project-specific architecture (e.g., 'Use React 18 and Tailwind CSS')

Pillar 2: Auto-Memory Architecture



Phase 1: Active Session
User and Agent interact.
Phase 2: Extraction
Agent automatically **extracts key facts**, **conventions**, and **architectural decisions.**
Phase 3: Storage
MEMORY.md (core context)
topic files
Phase 4: Rehydration
Session restarted (claude --continue). Stored files injected back into context.

Memory is not passive. The agent actively summarizes and commits architectural facts to MEMORY.md to guarantee context survives session restarts.

Pillar 3: Settings & Tool Permissions

.claude/settings.json

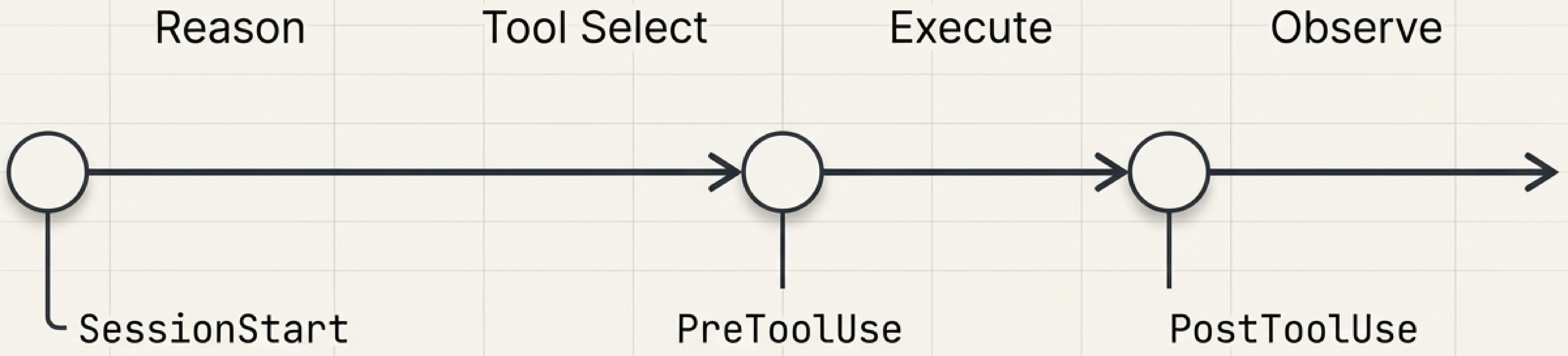
```
{
  "allowed_tools": ["Read", "Glob"],
  "denied_tools": ["Bash", "Write"]
}
```

Bypasses 'Ask' mode; Auto-accepts silently

Hard block; Agent cannot invoke under any circumstance

Project settings dictate the operational boundaries of the agent.
Explicit denial overrides all agent intentions.

Pillar 4: Hook Lifecycle Interception



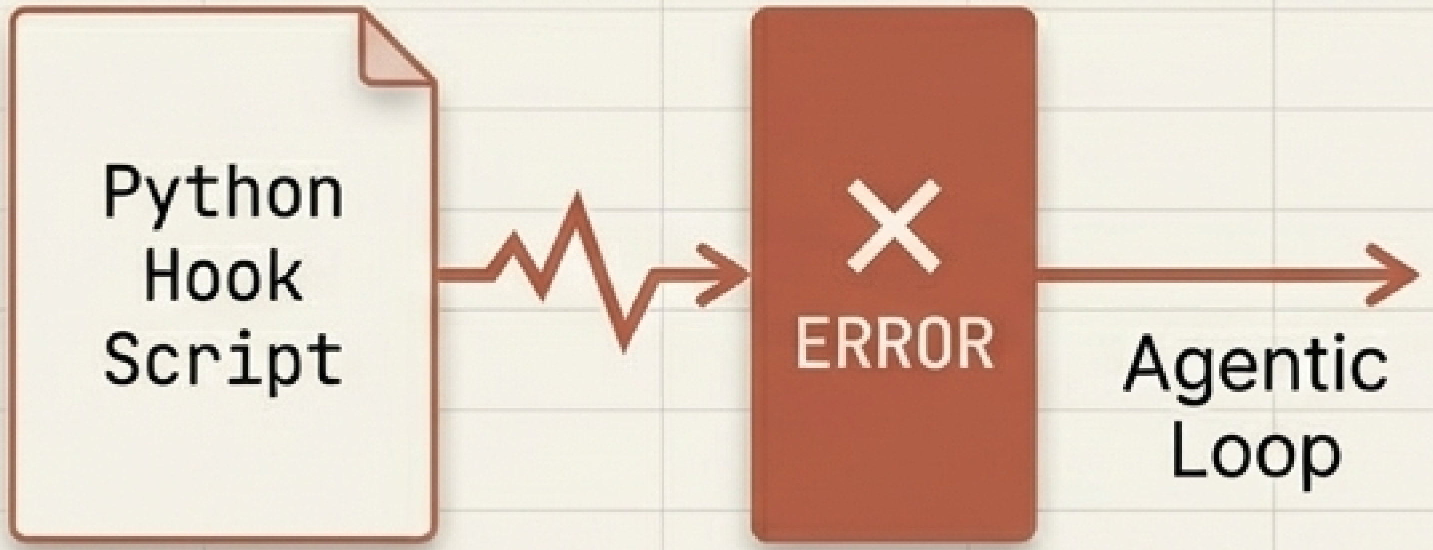
Fires once on launch. Ideal for injecting initial project context or greeting.

Intercepts selected tool. Can validate inputs, modify arguments, or block dangerous commands.

Intercepts tool result. Ideal for background auditing, logging, or state tracking.

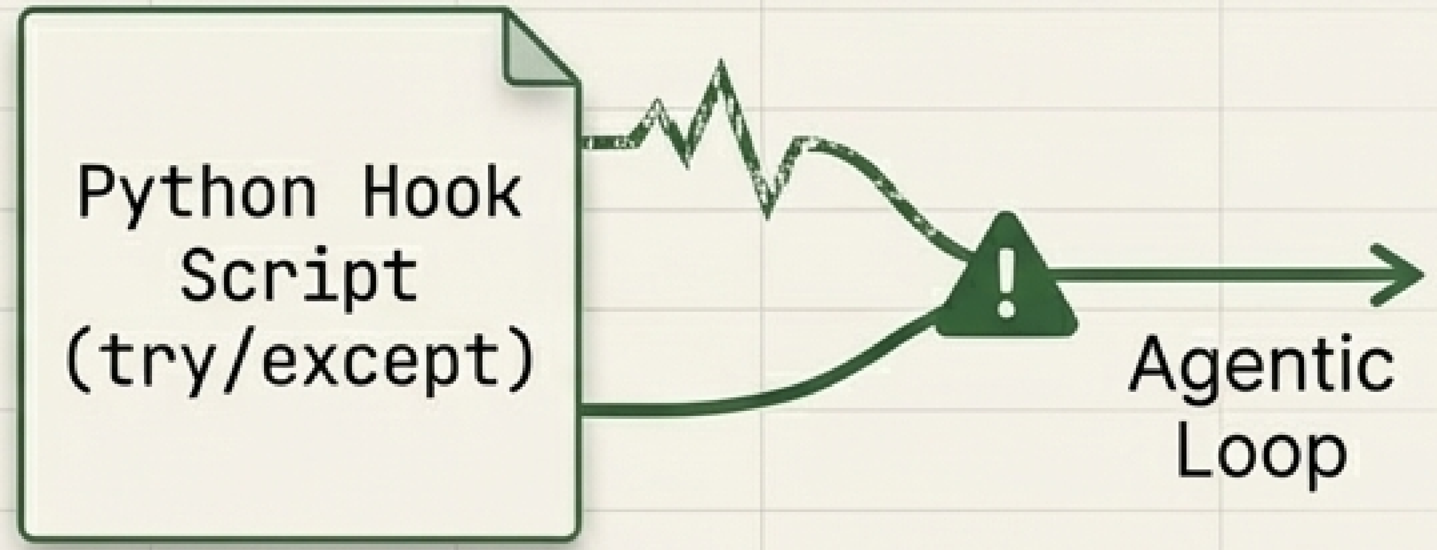
The Golden Rule of Hooks: Fail-Open

Fail-Closed (Anti-Pattern)



A crashing hook halts the entire Claude Code session. The agent is paralyzed.

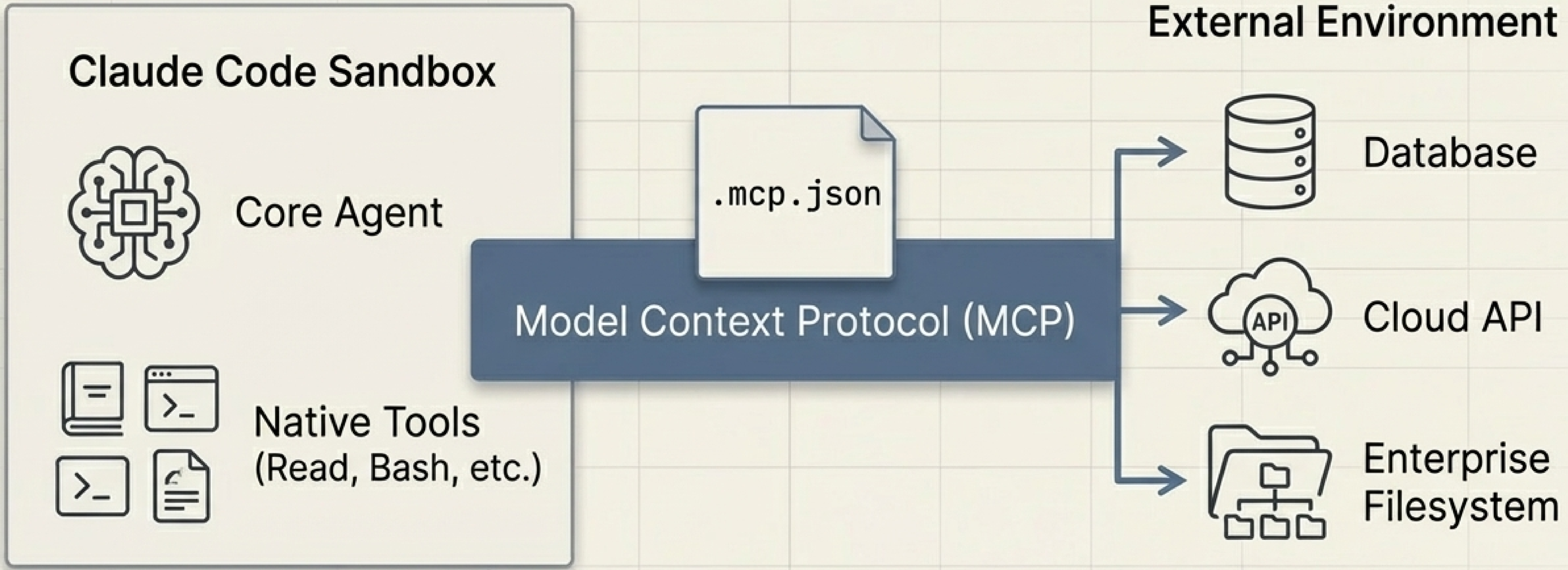
Fail-Open (Best Practice)



The hook gracefully logs the error and returns a 'proceed' signal. The agent continues its work uninterrupted.

Plugins must enhance the system, never become a single point of failure.

Pillar 5: MCP Server Configuration (.mcp.json)



Native tools operate locally. `.mcp.json` configures the servers that expose external databases, SaaS APIs, and custom tooling as callable native functions.

CC-103 Assessment: Practical Deliverables

☐ **1. Instruction Engineering**
Create a concise, project-level `CLAUDE.md` with 5+ hard rules.

☐ **2. Memory Initialization**
Demonstrate auto-memory persistence across a session restart.

☐ **3. Lifecycle Hook**
Write and register a non-blocking `SessionStart` hook.

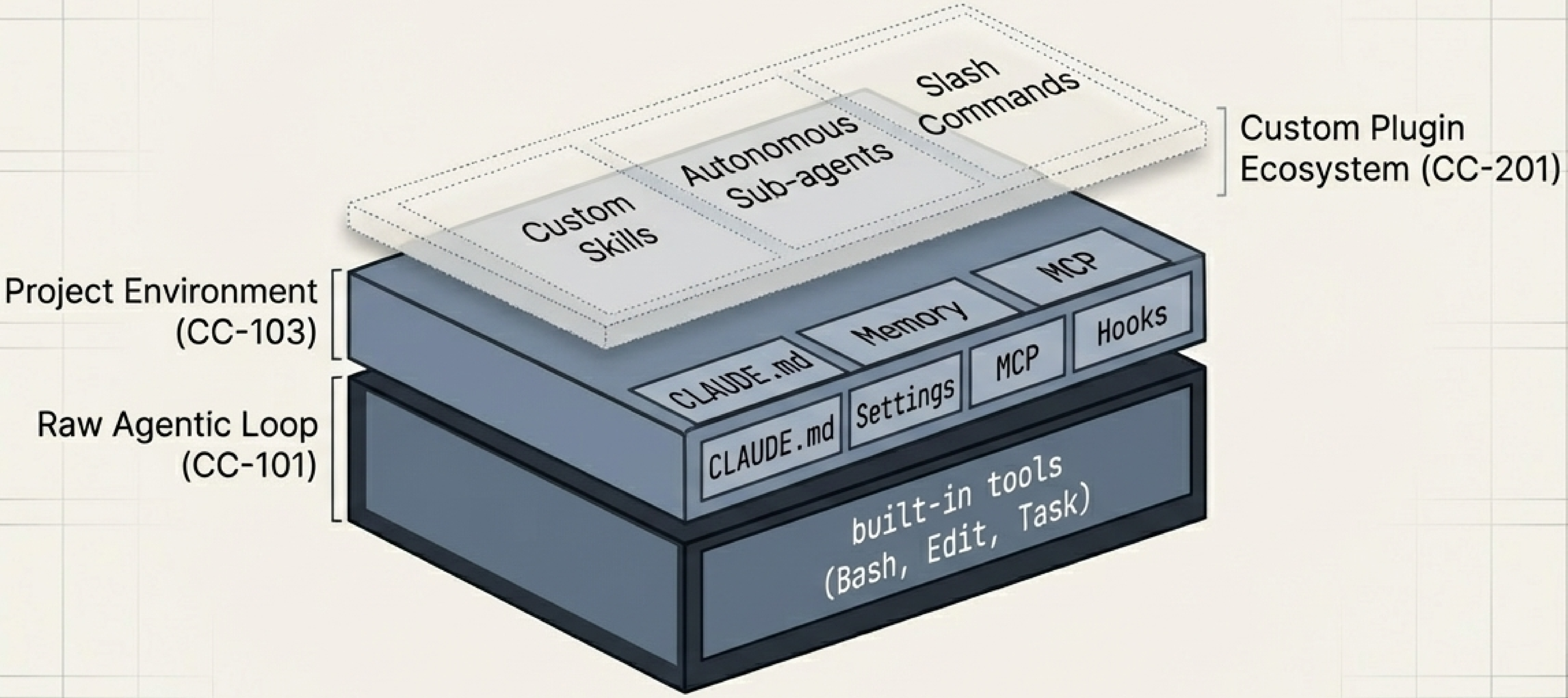
☐ **4. External Integration**
Configure a valid `.mcp.json` server connection.

Passing requires structurally valid configurations that load without syntax errors and gracefully handle execution failures.

Diagnostic Matrix: Instructor Pitfalls

Symptom	Likely Cause	Resolution
Agent ignores stated project conventions.	Verbose/Conversational `CLAUDE.md`.	Condense to 5-6 rigid imperative rules.
Agent uses tools that should be forbidden.	Permissions set in Global config, overridden by Project config.	Verify `.claude/settings.json` `denied_tools` at the project root.
Entire Claude Code session hangs or crashes instantly.	Hook script encountering an unhandled error.	Implement try/except blocks; enforce fail-open hook architecture.

Synthesis: The Configured Environment


Slash Commands
Autonomous Sub-agents
Custom Skills
Custom Plugin Ecosystem (CC-201)
MCP
Memory
Hooks
MCP
CLAUDE.md
Settings
CLAUDE.md
Project Environment (CC-103)
Raw Agentic Loop (CC-101)
built-in tools (Bash, Edit, Task)

Raw capabilities (**CC-101**) combined with rigid project configuration (**CC-103**) create the stable, predictable foundation required for custom Plugin Engineering (**CC-201**).