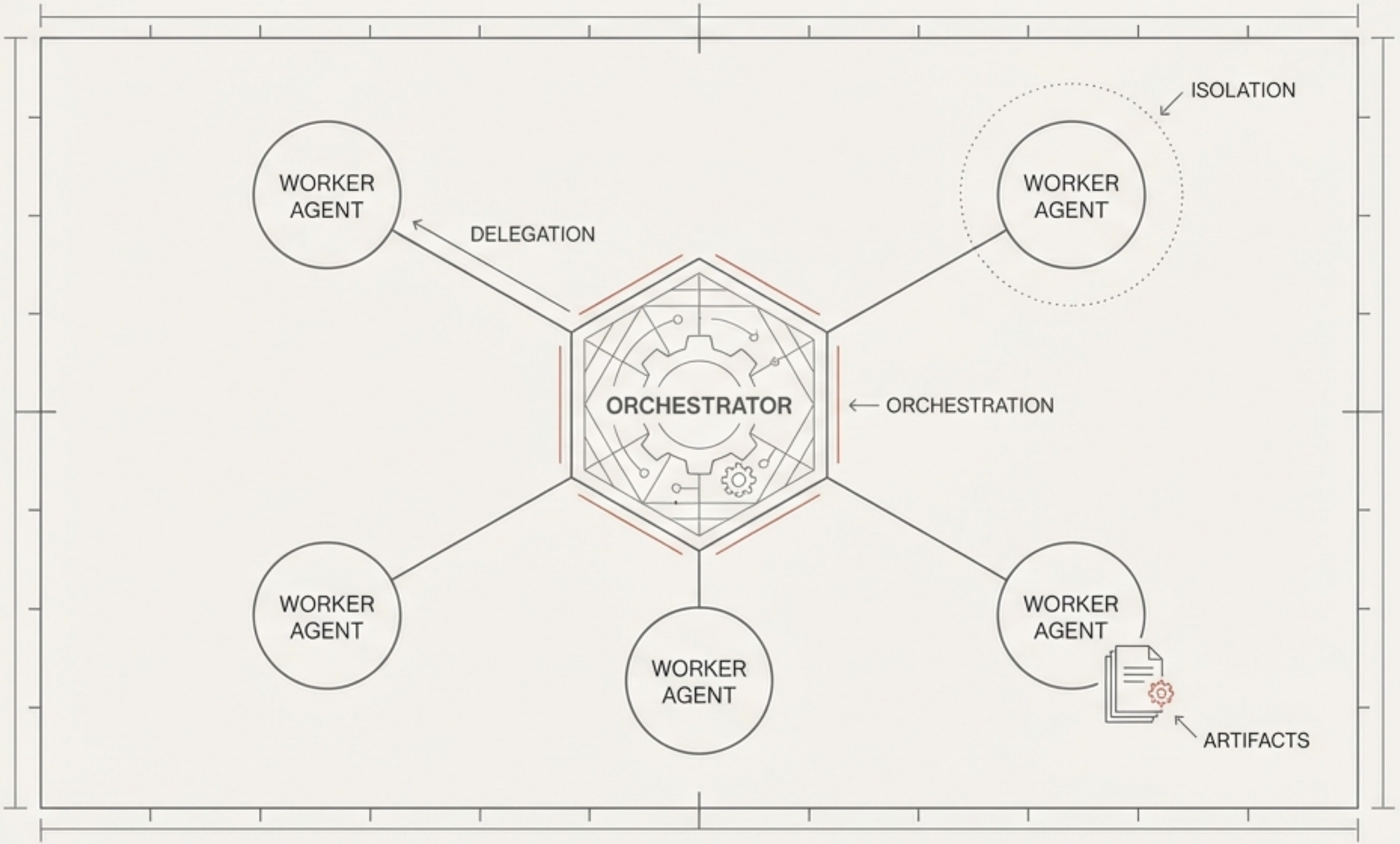
THE MULTI-AGENT PLAYBOOK

An Architectural Blueprint for Production-Ready Systems

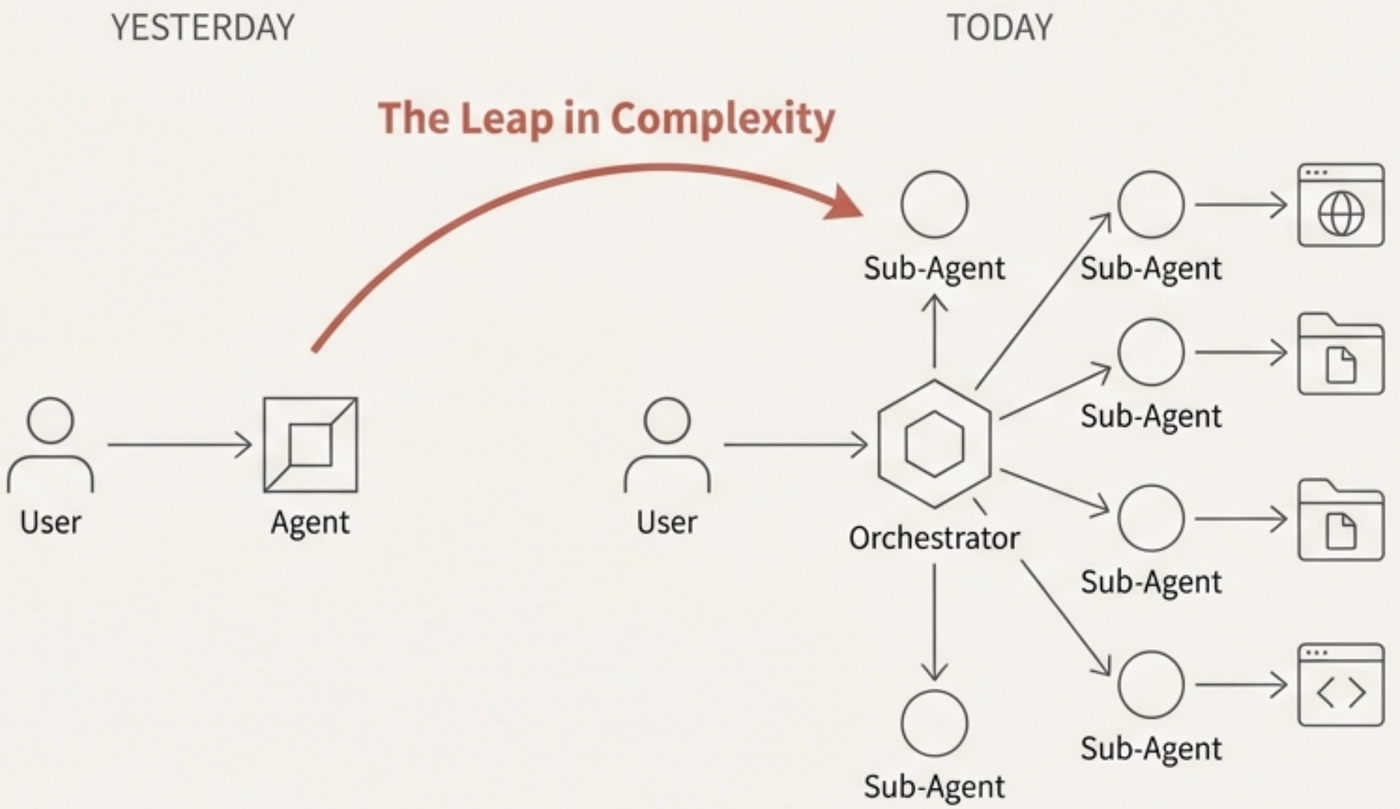


The engineering paradigm has shifted from single agents to coordinated systems.

Single-agent prototypes have proven the potential of agentic work. However, scaling this capability to solve real-world engineering problems requires moving beyond a single chat window.

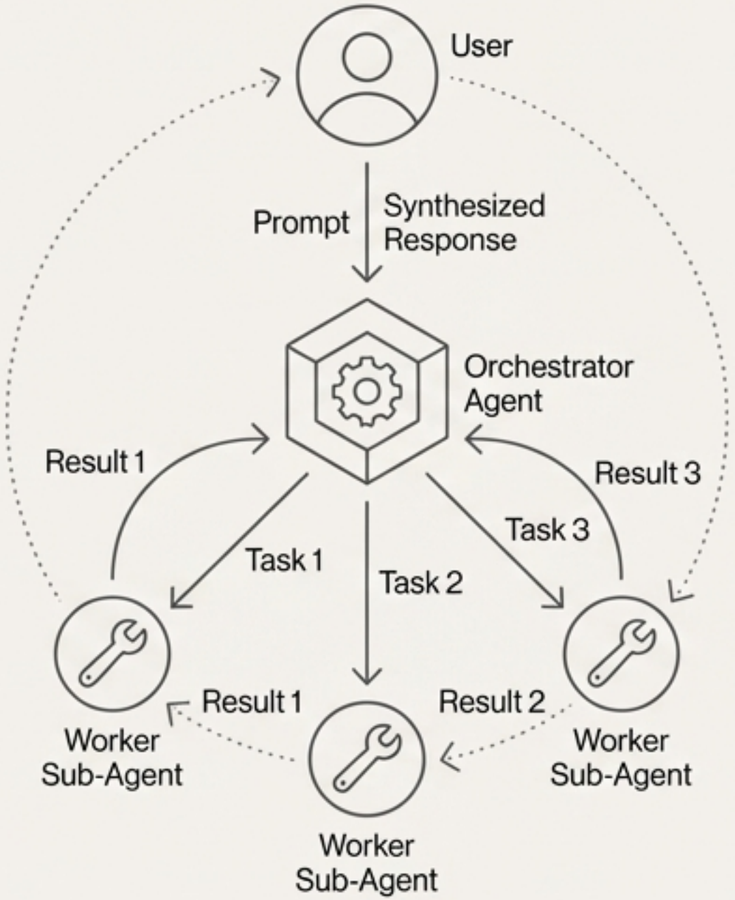
The new frontier is building **well-coordinated multi-agent systems** where specialized agents collaborate on complex, long-running tasks.

This introduces a critical challenge: How do you manage delegation, state, execution, and failure without creating chaos? The answer lies in a robust architectural pattern.



The core pattern is a closed-loop system led by an Orchestrator.

Core Principle Statement: You do not prompt your sub-agents. You prompt your primary agent, and the primary agent prompts your sub-agents. They respond back to the primary agent, which synthesizes the information for you.



The General Contractor Model

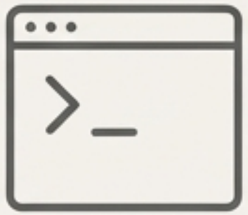
The Orchestrator is a **General Contractor** who holds the master blueprint. The Sub-Agents are specialized trades (eiectricians, plumbers) who receive specific instructions and report their progress back to the contractor, not the homeowner.

The minimum viable stack prioritizes intelligence at every layer.

This architecture is powered by a specific set of tools. The recommended strategy has shifted from a tiered model hierarchy to a uniform, high-intelligence stack. As the source material states: **“Right now it looks like Opus is going to be both the workhorse and the powerful model.”**

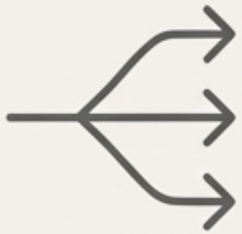
Claude Opus 4.5

Used for both the Orchestrator and all Sub-Agents. It is specifically trained to be a “prompt engineer” for other agents, making it ideal for delegation.



Claude Code (CLI)

The command-line interface that acts as the “agent harness,” providing the execution environment for the Orchestrator to run and spawn sub-processes.



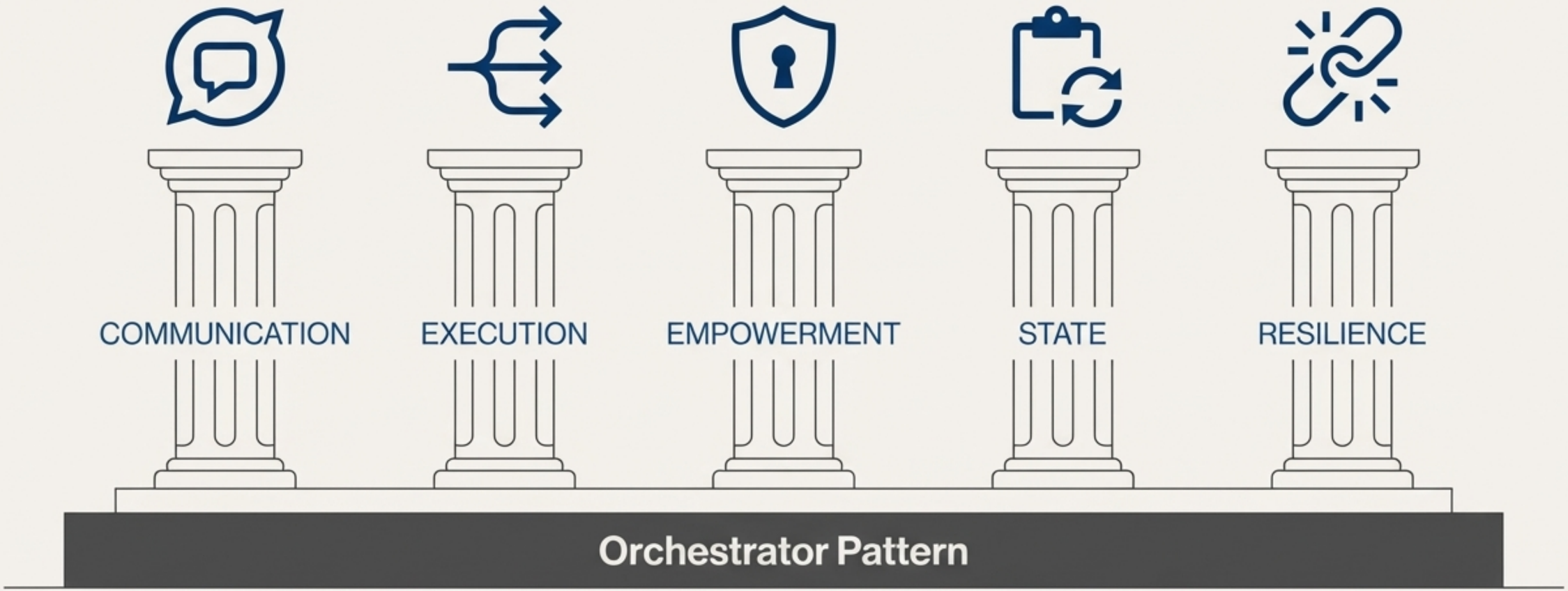
Built-in “Task Tool”

Opus 4.5 uses a native tool to explicitly write prompts for and hand off work to sub-agents. Activated via prompt flags like `parallel=true`.

A Markdown File

The immutable “master plan” containing a task-based list that the Orchestrator ingests to decompose and delegate work.

A robust multi-agent system is supported by five architectural pillars.



We will examine the core principle behind each pillar, the practice of implementing it, and the payoff it delivers.

PILLAR I: COMMUNICATION

Agents communicate through closed loops and verifiable artifacts.

THE PRINCIPLE

Sub-agents must report exclusively to the Orchestrator, transmitting either high-level summaries or unaltered, verifiable artifacts.

Direct user communication is forbidden.

THE PRACTICE

Information to Condense (Synthesis)

- Summaries of raw content (e.g., extracting keywords from a PDF).
- Status and operational metrics (e.g., the "five agent summary").

Information to Preserve (Artifacts & Signals)

- **Files and Assets**: Full, named files (e.g., screenshots, downloads).
- **Test Logs**: The "full complete file" of test results.
- **Error Signals**: Critical failures must be transmitted in full to trigger resolver steps.

THE PAYOFF

This creates **Verifiable Workflows**. The Orchestrator can confirm task completion based on the presence of an artifact and route the system based on clear error signals.

The Detective & The Investigator

An investigator (Sub-Agent) gives the detective (Orchestrator) a quick summary ("The suspect was here") but also hands over the unaltered bagged evidence (Artifacts) needed to close the case.

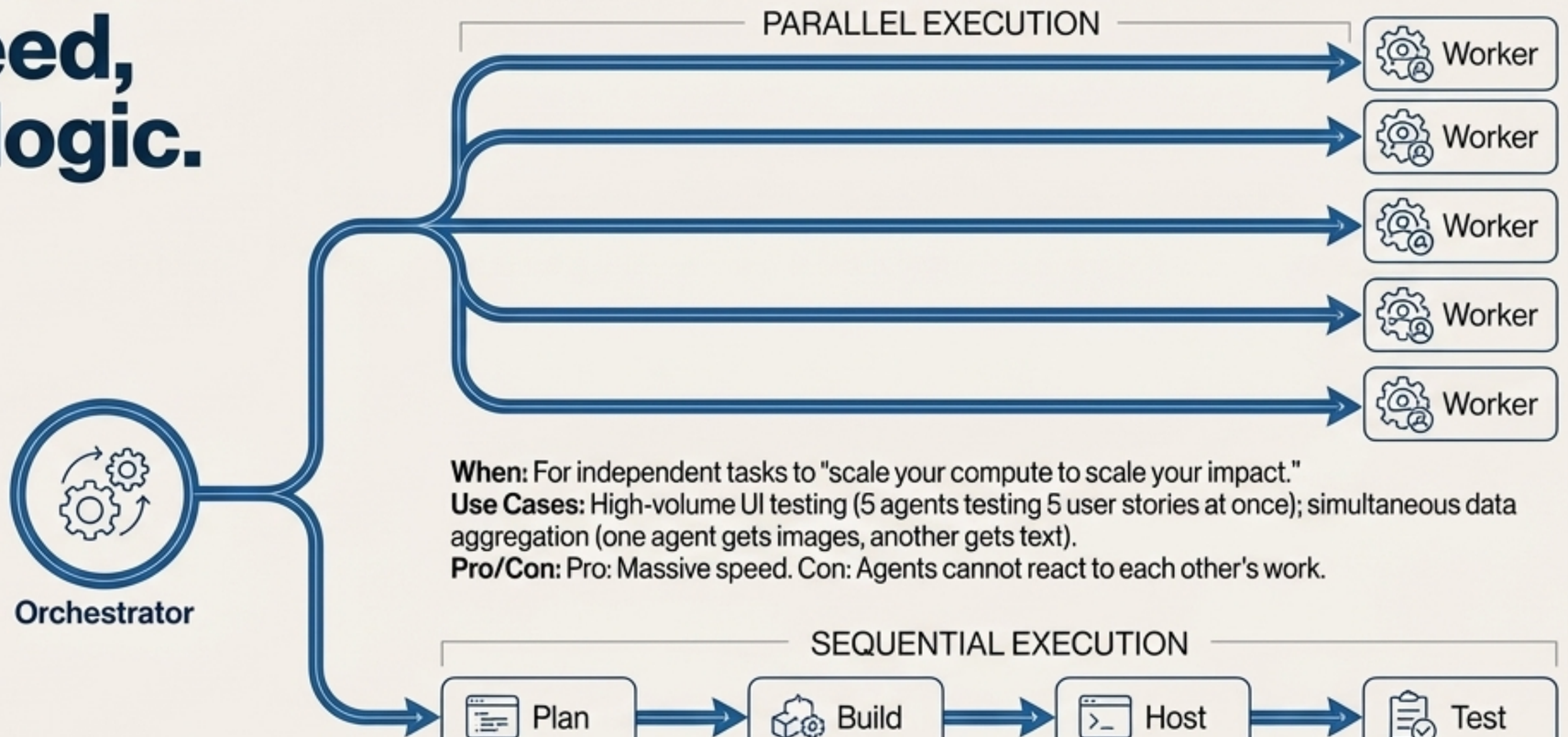
PILLAR II: EXECUTION

Execution is a choice: parallel for speed, sequential for logic.

THE PRINCIPLE

The orchestration pattern must adapt to the task's dependencies. Independent tasks should be parallelized for throughput; interdependent tasks must be sequential to ensure logical integrity.

THE PRACTICE



When: For independent tasks to "scale your compute to scale your impact."
Use Cases: High-volume UI testing (5 agents testing 5 user stories at once); simultaneous data aggregation (one agent gets images, another gets text).
Pro/Con: Pro: Massive speed. Con: Agents cannot react to each other's work.

When: For "real engineering work" with strict dependencies.
Use Case: The Plan → Build → Host → Test workflow. The "Test" step cannot begin until the "Host" step is complete.
Pro/Con: Pro: Logical and stable. Enables error recovery loops. Con: Slower elapsed time.

THE PAYOFF

A Hybrid Approach. Advanced workflows use a sequential master plan (Plan → Build → Host → Test) but trigger a parallel explosion of agents within a single step (e.g., the "Test" phase runs 50 UI tests at once).

PILLAR III: EMPOWERMENT & ISOLATION

Empower agents with tools, but isolate them in sandboxes

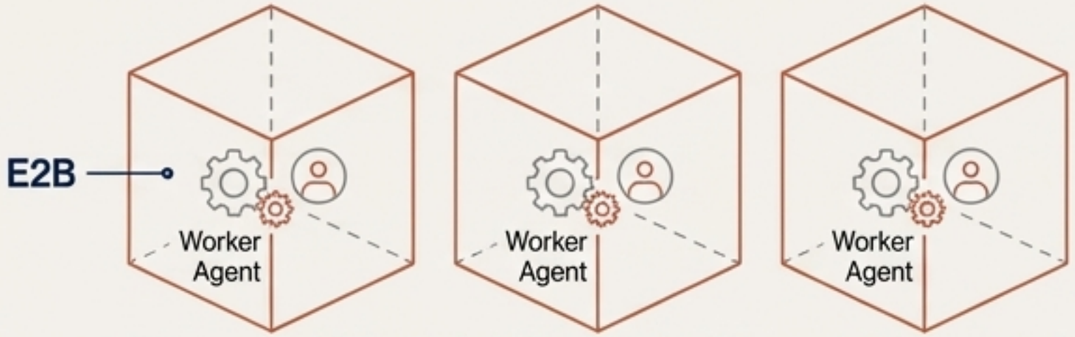
THE PRINCIPLE

Instead of restricting an agent's tools to ensure safety, provide the "right tooling" to maximize capability and restrict the environment to contain risk.

Functional Empowerment - The 'Right Tooling'

- Provide comprehensive suites, not minimal scripts (e.g., an "entire browser suite tool").
- Run browsers in **"headed mode"** for visual verification during complex tasks.
- Deploy a custom "skill codebase" for specialized tools (e.g., decision matrix, graphing tool).

Environmental Isolation - Sandboxes



- Give agents "their own isolated devices" using a sandbox provider like **E2B**.
- This provides "isolation, scale, and autonomy," allowing an agent root-level control to build and test an application without risking the host machine.

THE PAYOFF

This model enables agents to perform "nasty engineering work" on full-stack applications with a contained blast radius. A crashed agent simply means one sandbox "aired out" while others continue unaffected.

The Private Workshop

Instead of a badge that opens one door (Restrictive), you give a worker a fully stocked private workshop (Sandbox) with every power tool imaginable (Right Tooling). They have total freedom inside, but cannot affect anything outside.

PILLAR IV: STATE & HANDOFFS

State lives in the environment, not the agent's memory.

THE PRINCIPLE: To achieve persistence and enable handoffs between agent sessions, you must preserve the work artifacts and the execution environment, not the agent's conversational history.

THE PRACTICE

1. The Environment Artifact: The Hosted Sandbox

The live, running container (E2B) where the application is hosted. A new agent session simply reconnects to this persistent environment.

2. The Instructional Artifact: The "Master Plan" File

The static Markdown prompt file that serves as the immutable mission contract.

3. The Result Artifact: The "Full Complete File"

Summary logs, named assets (e.g., downloaded images), and error logs that agents write to the file system before terminating.

4. The Codebase Artifact: The "Brownfield" State

The source code itself. New agents are instructed to read the existing files and create PRs, treating the file system as the source of truth.

THE PAYOFF

Perfect recall and seamless multi-session handoffs. An agent can resume work on a "brownfield code base" instantly, without needing to "remember" what it did in a previous session.

The Factory Shift Change
The next worker doesn't read the outgoing worker's mind. They look at the machinery (Sandbox), check the SOP on the clipboard (Master Plan), and read the logbook of finished parts and errors (Result Artifacts).

PILLAR V: RESILIENCE

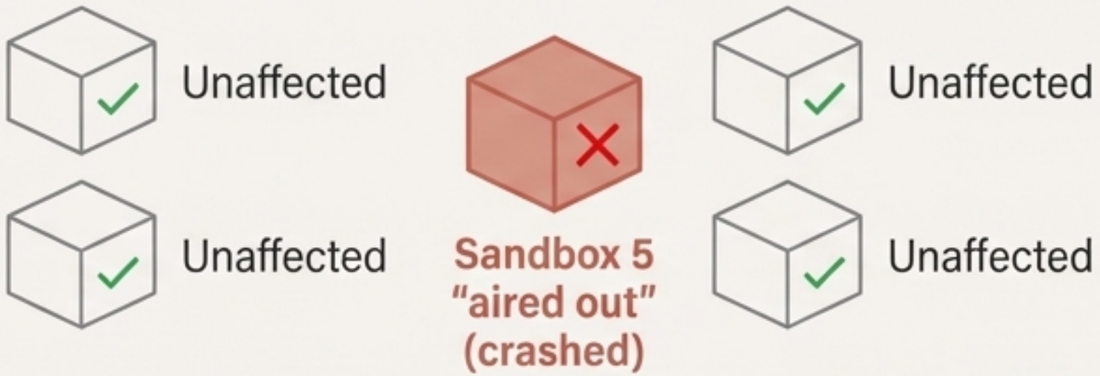
Resilience is architected through isolation and error routing.

THE PRINCIPLE

A single sub-agent failure must not derail the entire orchestration. The system must contain the "blast radius" of a failure and have a built-in logic for recovery.

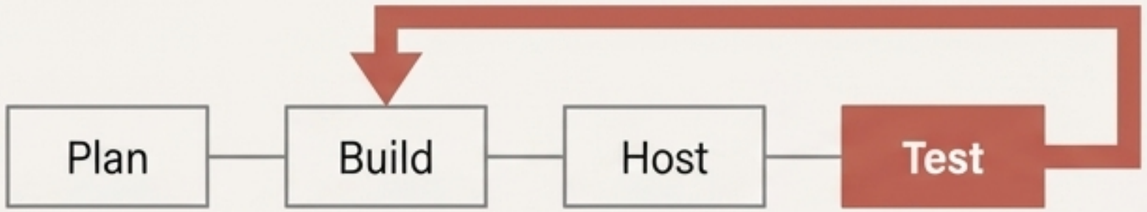
THE PRACTICE

Isolation - Blast Radius Containment



By giving every agent its own sandbox, a catastrophic failure is contained to a single, disposable environment.

Routing - The "Debug or Resolver" Step



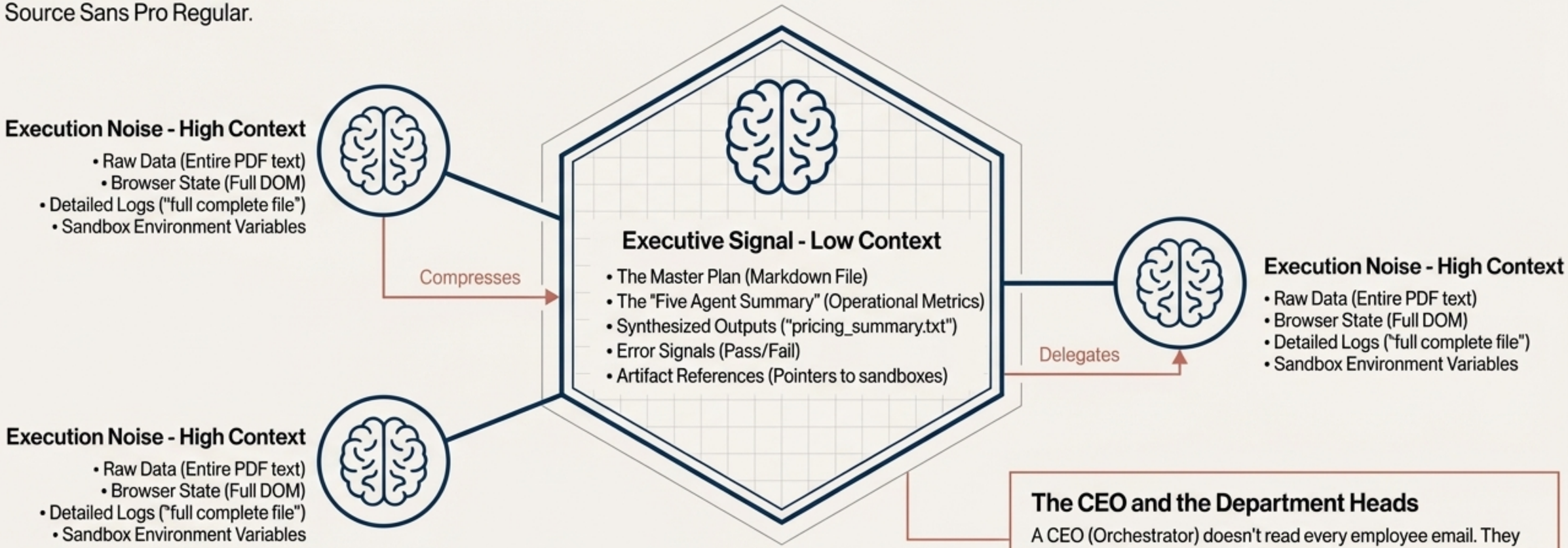
If the "Test" agent reports a failure, the Orchestrator doesn't stop. It receives the error signal and routes the workflow **back to the "Build" step** to apply a fix, creating a self-correcting loop.

THE PAYOFF

A robust, self-healing system that can manage complex, long-running tasks where individual component failures are expected, not exceptional.

Managing Context: The Orchestrator holds the signal, Sub-Agents absorb the noise.

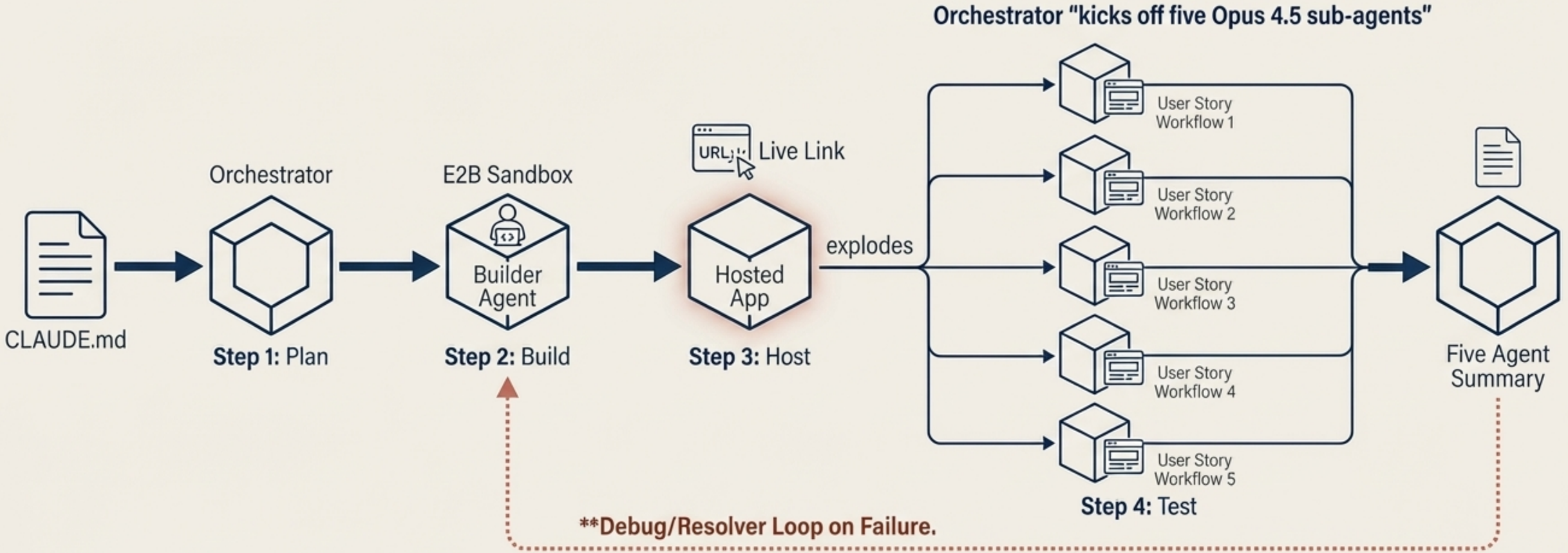
The primary strategy for preventing context window overflow is distributed computing. The token load is split across many distinct agents, with the Orchestrator acting as a compression filter.
Source Sans Pro Regular.



The CEO and the Department Heads

A CEO (Orchestrator) doesn't read every employee email. They rely on department heads (Sub-Agents) to manage the noise and report back only the critical signals required to make decisions.

The complete architecture combines sequential logic with parallel execution.



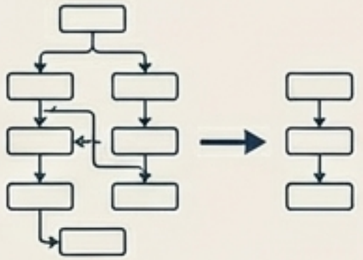
This hybrid model provides the logical stability of a pipeline with the massive throughput of parallel processing at the most critical validation stage.

Performance is measured by review velocity and verifiable outcomes.

Traditional software metrics are insufficient. In agentic engineering, the key constraint is the human review cycle. The best systems are those that deliver self-validated, verifiably correct work.

Review Velocity

The primary metric. How quickly can a human confirm the work is correct? A system that builds an app *and* a passing test suite reduces review time to near-zero.



Tool Calls to Completion

A smarter model is cheaper.
“If Opus does the job in 5 tool calls and Sonnet takes 10, you have still saved money.”

Artifact Validation

The ultimate success metric. Did the required artifact (a file, a PNG, a database entry) get created? Performance is validated by checking the file system, not the agent's chat log.

Resource Observability

The “Five Agent Summary” provides a health check, tracking “tool uses and token usage as value generated” to spot inefficient or looping agents.

The goal is not to build the application. It is to build the system that builds the application.

Moving into production with multi-agent systems requires a fundamental shift in perspective. We are no longer just prompt engineers talking to a single model; we are system architects designing resilient, scalable, and observable agentic frameworks. The agent is the new compositional unit of engineering.