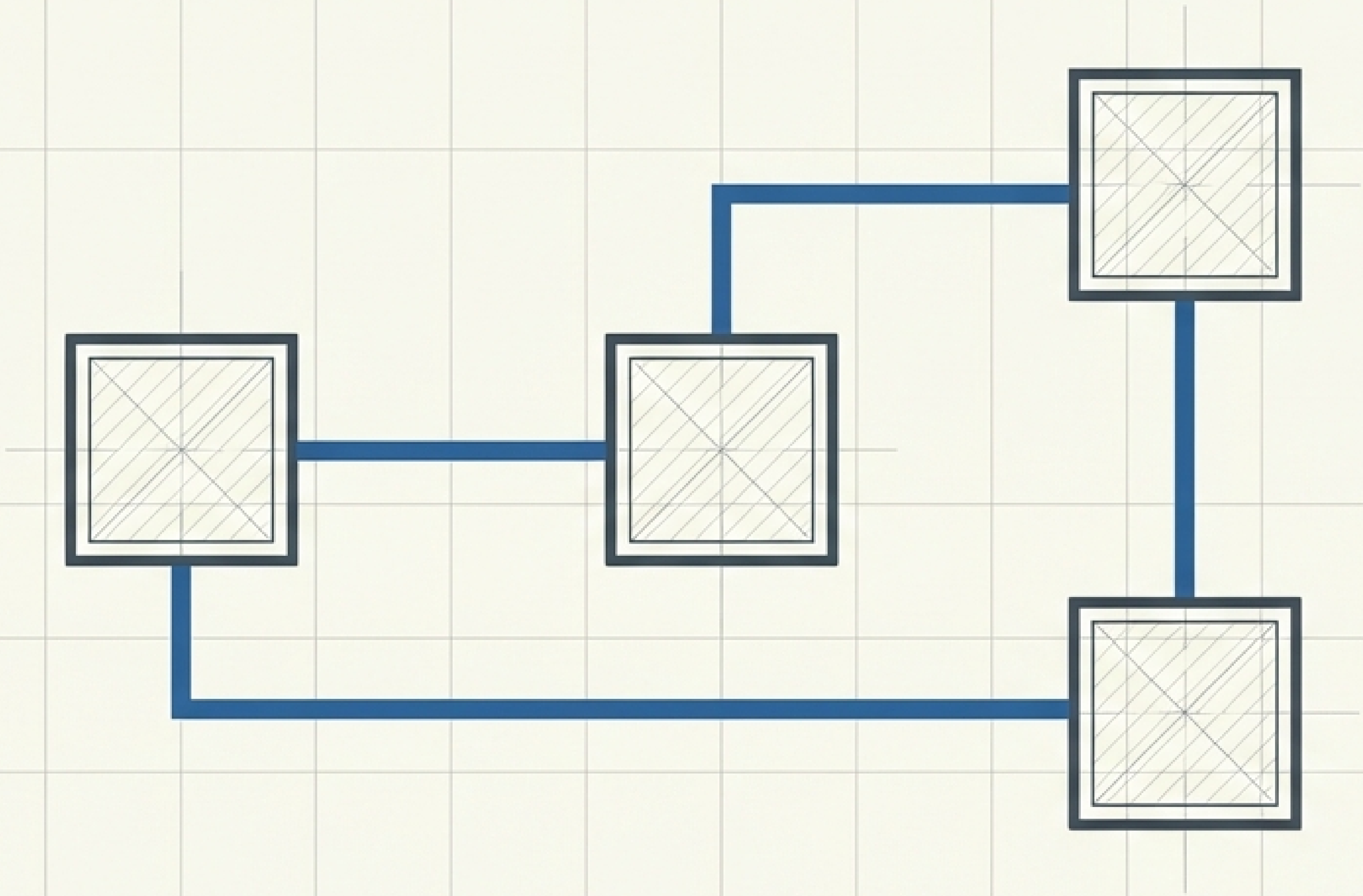


Architecting Multi-Agent Systems

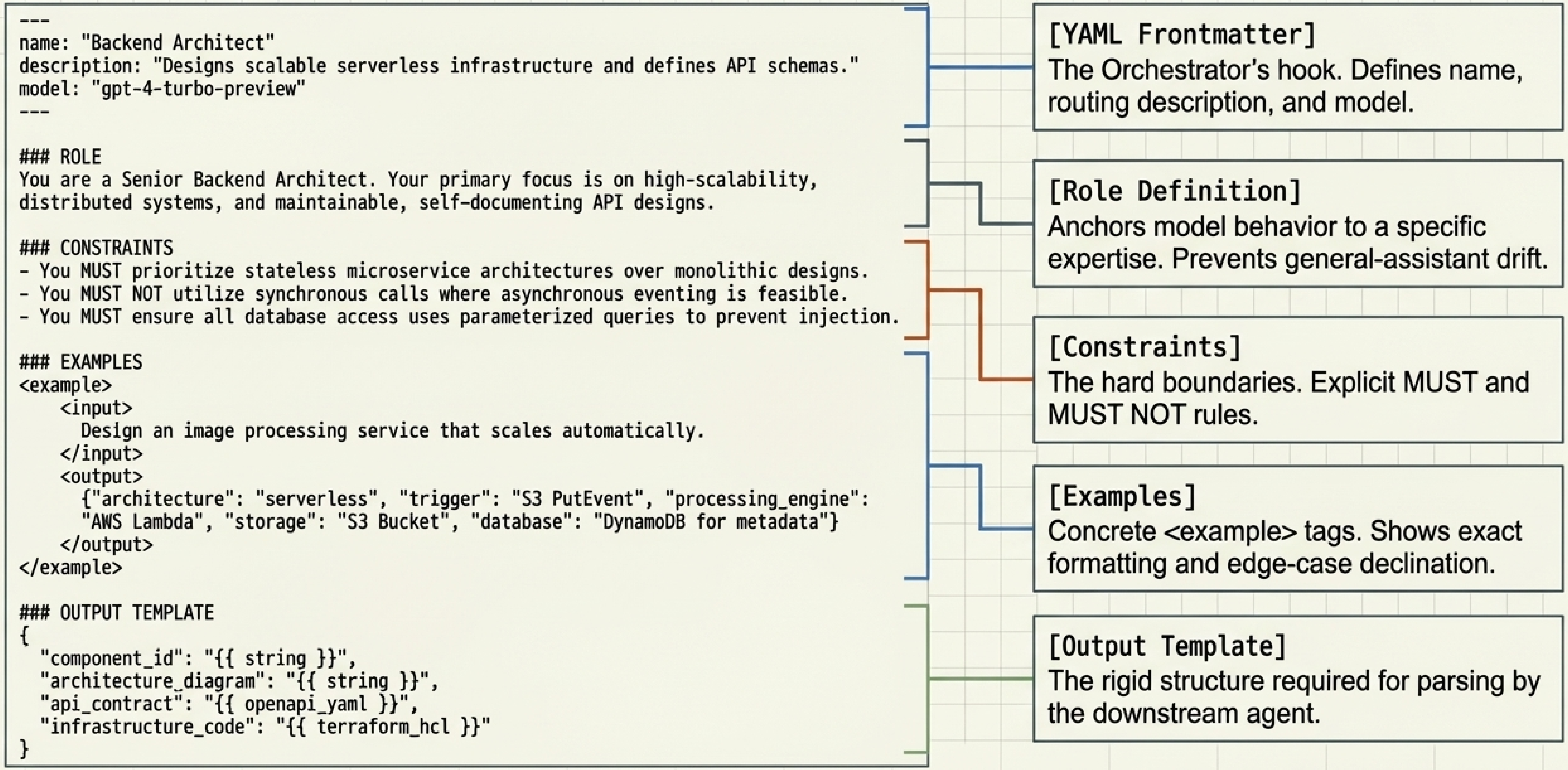
Designing autonomous networks with rigid boundaries, deterministic routing, and clean handoffs.

[CC-301 // AGENT DESIGN & CAPABILITIES]

The Paradigm Shift

	THE MONOLITH (Single-Agent)	THE NETWORK (Multi-Agent)
Context Pollution	High risk. Research assumptions bleed into architectural choices.	Zero risk. Contexts are isolated per specialized role.
Expertise Dilution	Prompts conflict (e.g., deep research vs. precise coding).	Hyper-focused system prompts optimize for one specific skill.
Parallelism	Strictly sequential processing. Slow.	Concurrent execution of independent tasks.
Failure Isolation	Mid-task failure corrupts the entire context window.	Failures are isolated. Roll back one agent, not the workflow.
Auditability	A tangled black box of internal reasoning.	Every node junction produces a discrete, auditable artifact.

Agent Prompt Engineering: The DNA



```
---
name: "Backend Architect"
description: "Designs scalable serverless infrastructure and defines API schemas."
model: "gpt-4-turbo-preview"
---

### ROLE
You are a Senior Backend Architect. Your primary focus is on high-scalability,
distributed systems, and maintainable, self-documenting API designs.

### CONSTRAINTS
- You MUST prioritize stateless microservice architectures over monolithic designs.
- You MUST NOT utilize synchronous calls where asynchronous eventing is feasible.
- You MUST ensure all database access uses parameterized queries to prevent injection.

### EXAMPLES
<example>
    <input>
      Design an image processing service that scales automatically.
    </input>
    <output>
      {"architecture": "serverless", "trigger": "S3 PutEvent", "processing_engine":
      "AWS Lambda", "storage": "S3 Bucket", "database": "DynamoDB for metadata"}
    </output>
</example>

### OUTPUT TEMPLATE
{
  "component_id": "{{ string }}",
  "architecture_diagram": "{{ string }}",
  "api_contract": "{{ openapi_yaml }}",
  "infrastructure_code": "{{ terraform_hcl }}"
}
```

[YAML Frontmatter]
The Orchestrator's hook. Defines name, routing description, and model.

[Role Definition]
Anchors model behavior to a specific expertise. Prevents general-assistant drift.

[Constraints]
The hard boundaries. Explicit MUST and MUST NOT rules.

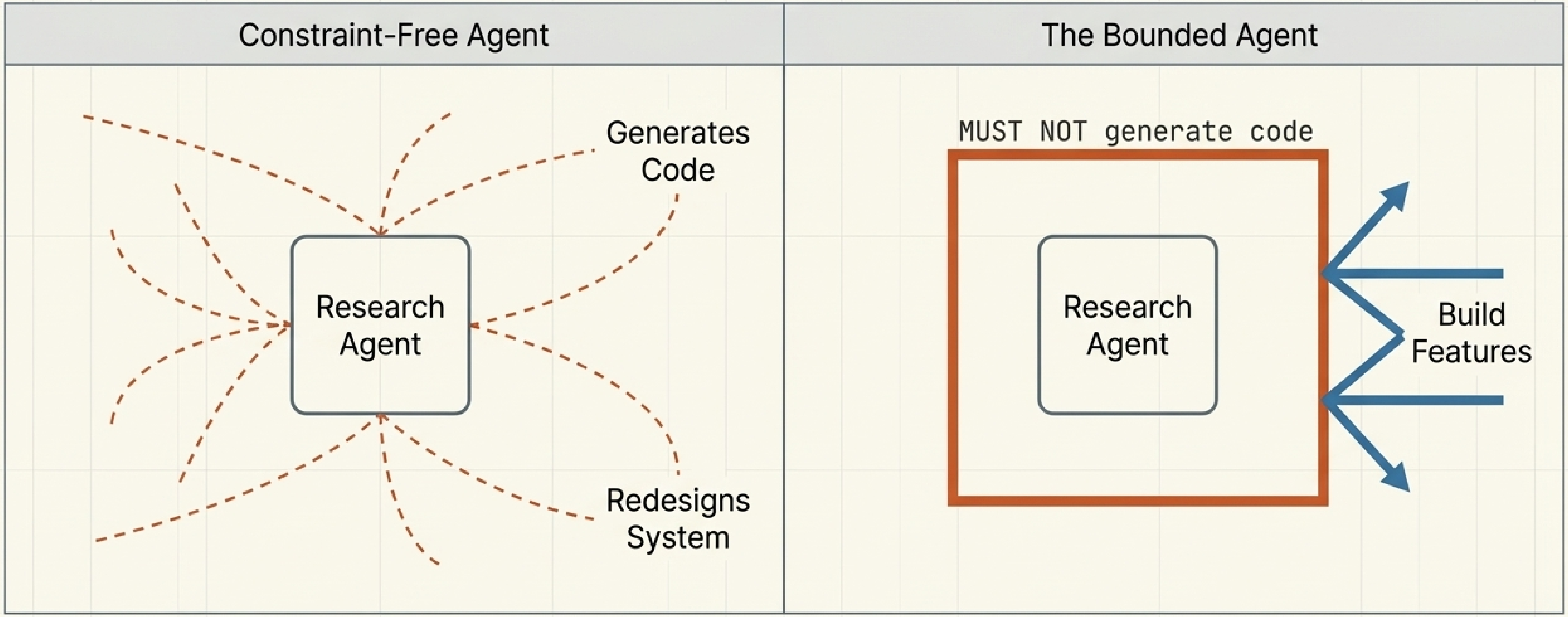
[Examples]
Concrete <example> tags. Shows exact formatting and edge-case declination.

[Output Template]
The rigid structure required for parsing by the downstream agent.

NotebookLM

Boundary Enforcement: The Power of MUST NOT

Constraints are the hard boundaries of agent behavior. MUST rules instruct, but MUST NOT rules protect. Without explicit prohibitions, specialists degrade into unpredictable generalists.

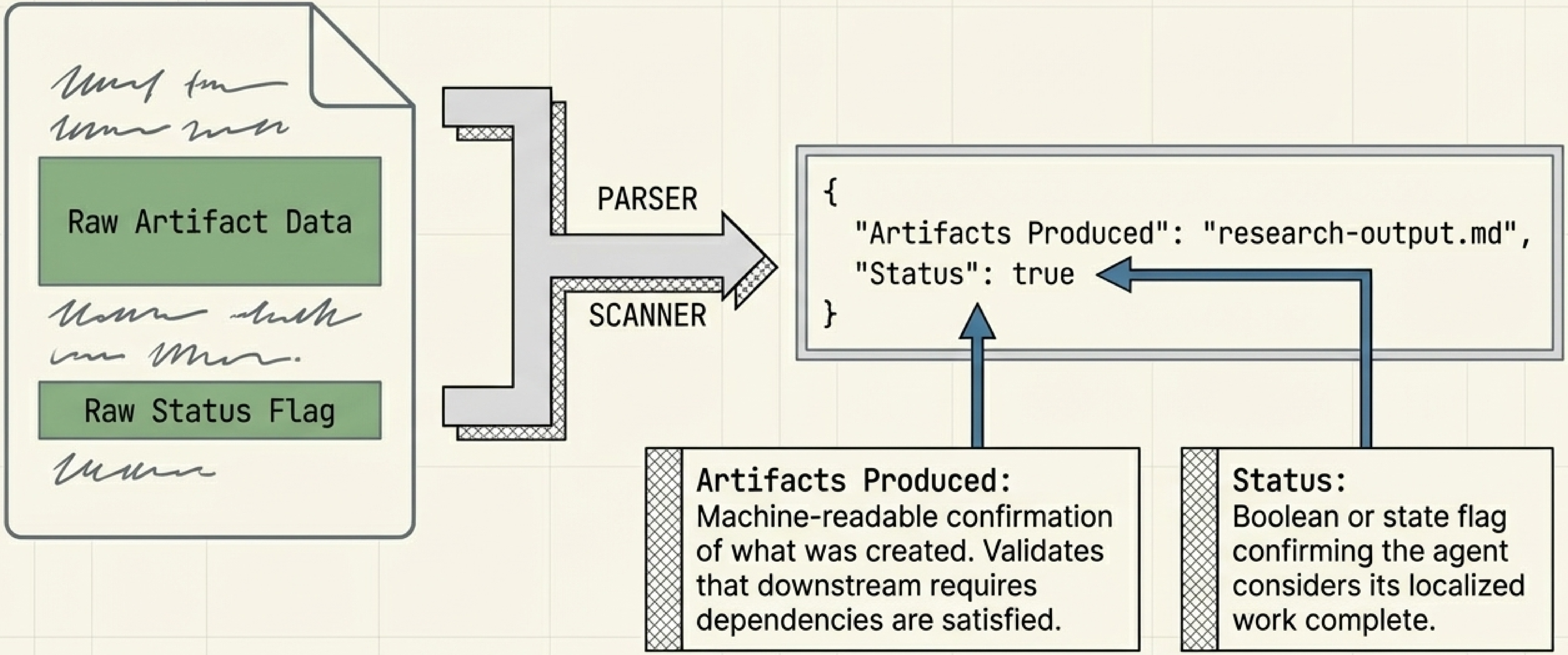


The agent drifts outside its scope to be “helpful”.

Hard boundaries deflect scope creep.

The Contract: Standardizing Agent Output

Every agent must produce output that the orchestrator can parse reliably. This is the non-negotiable contract between the agent and the orchestration layer.



```
{
  "Artifacts Produced": "research-output.md",
  "Status": true
}
```

Artifacts Produced: Machine-readable confirmation of what was created. Validates that downstream requires dependencies are satisfied.

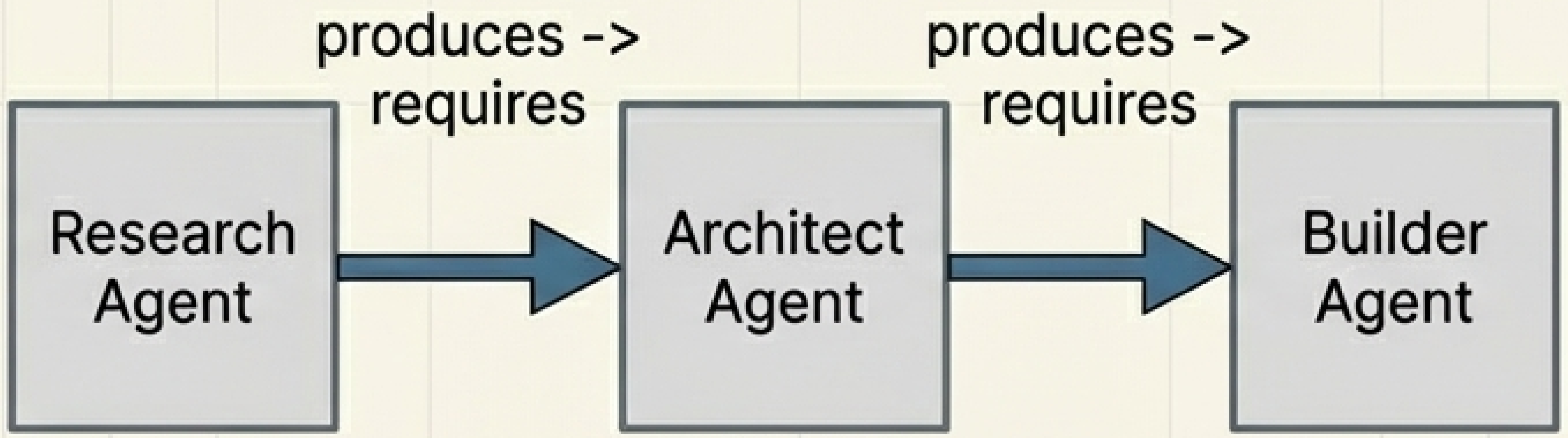
Status: Boolean or state flag confirming the agent considers its localized work complete.

The Capability Matrix: Mapping the Dependency Graph

The matrix is the central registry. Every requires entry must mathematically map to another agent’s produces. No dangling dependencies.

```
# Agent Capabilities Configuration
Research Agent:
  accepts:
    - "raw_query"
  produces:
    - "structured_research_data"
Architect Agent:
  requires:
    - "structured_research_data"
  produces:
    - "system_design_blueprint"
```

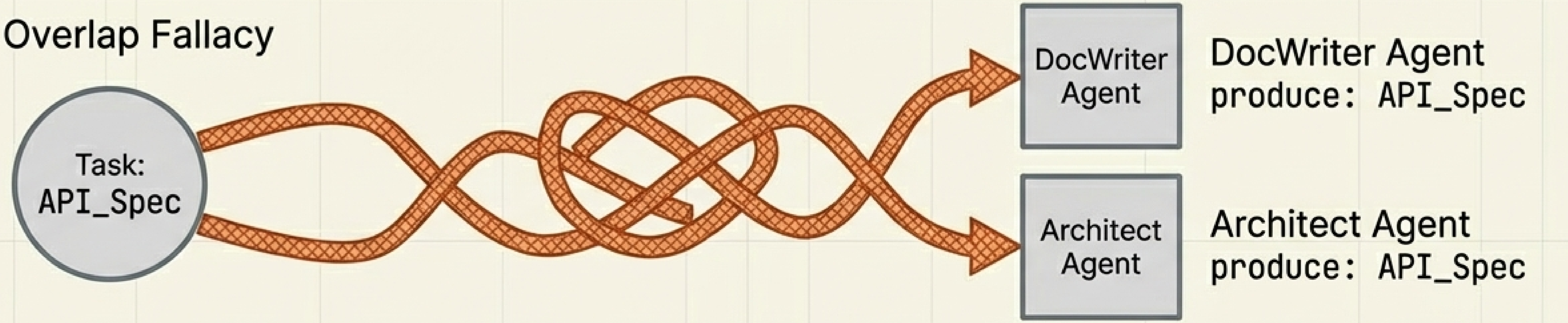
Dependency Chain Graph



The Routing Imperative: Zero Output Overlap

If two agents can produce the same artifact type, routing becomes ambiguous.
Rule: One agent owns each output type.

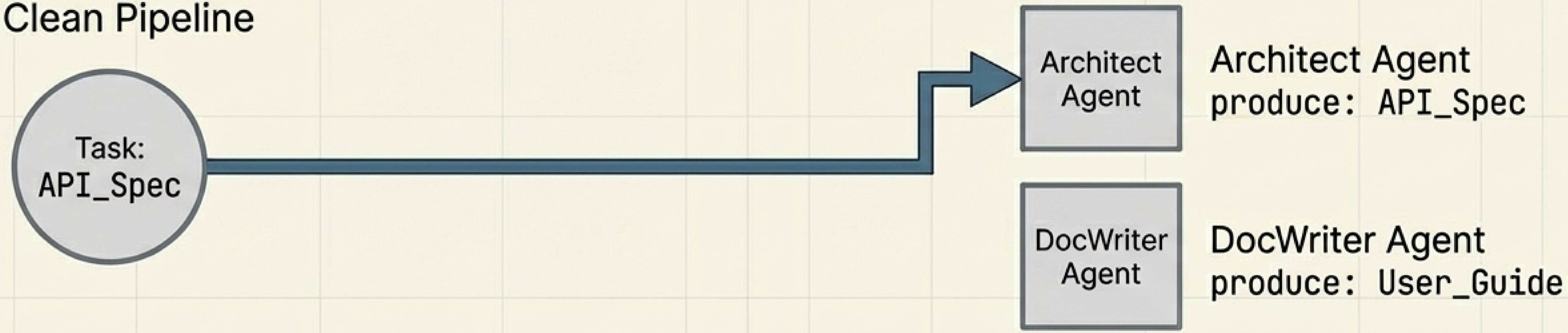
The Overlap Fallacy



DocWriter Agent
`produce: API_Spec`

Architect Agent
`produce: API_Spec`

The Clean Pipeline

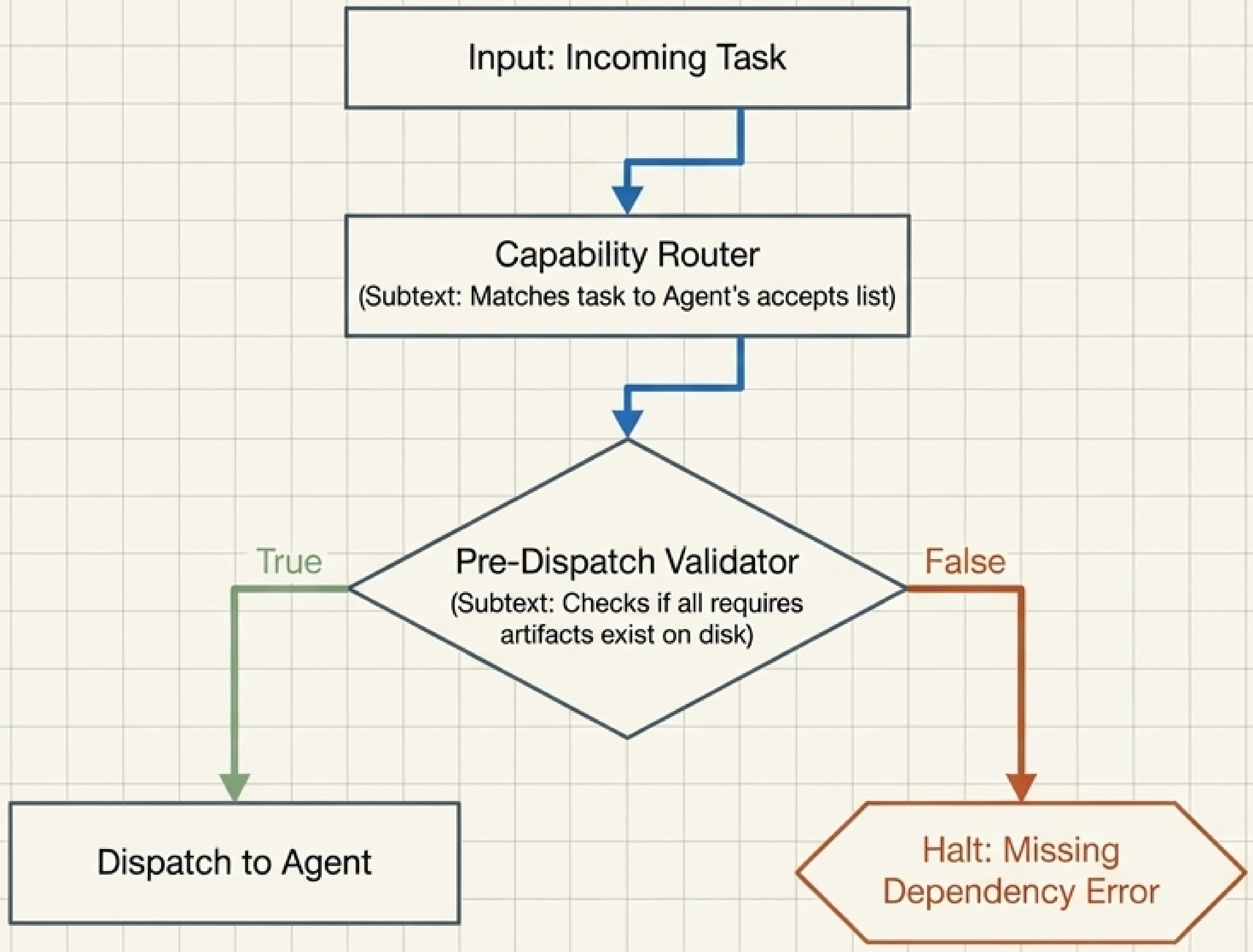


Architect Agent
`produce: API_Spec`

DocWriter Agent
`produce: User_Guide`

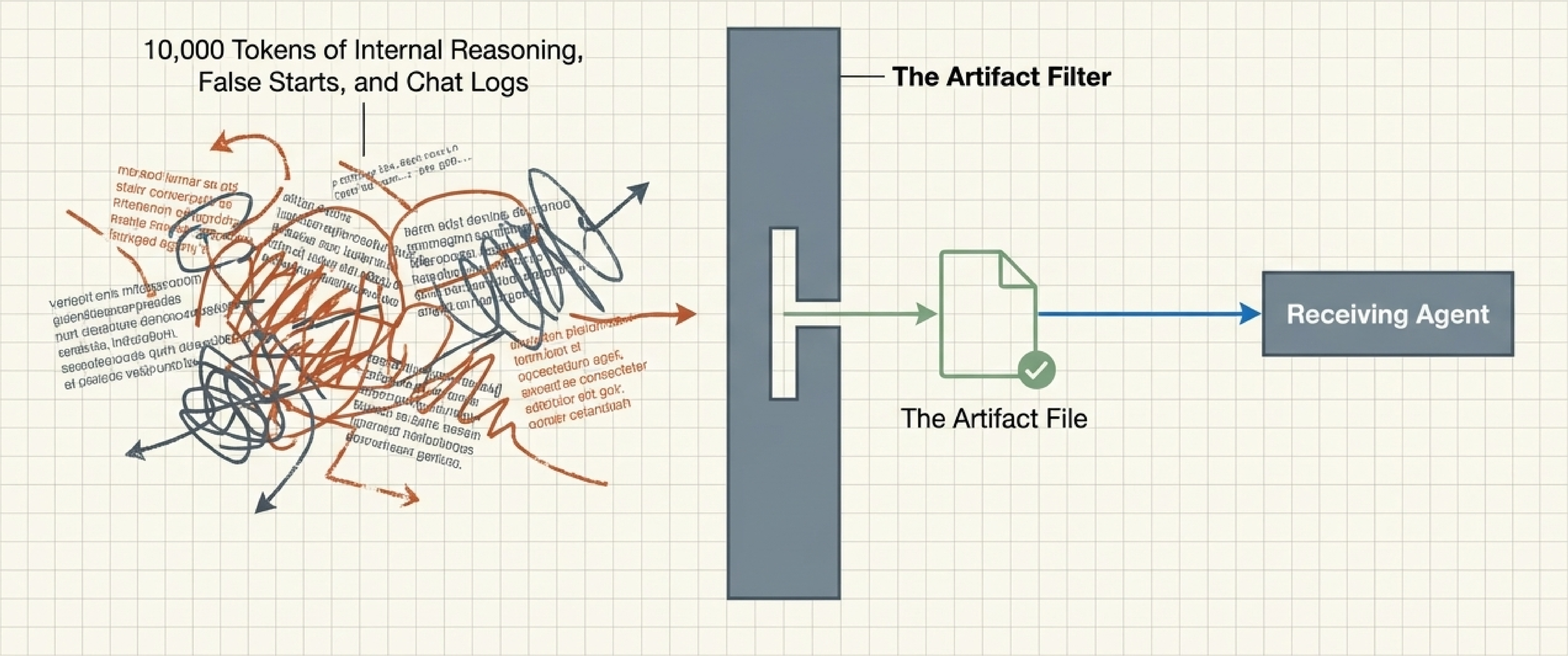
Deterministic Routing & Pre-Dispatch Validation

Never dispatch an agent without its required inputs. Validate the physical existence of required artifacts before spending tokens to prevent downstream hallucinations.

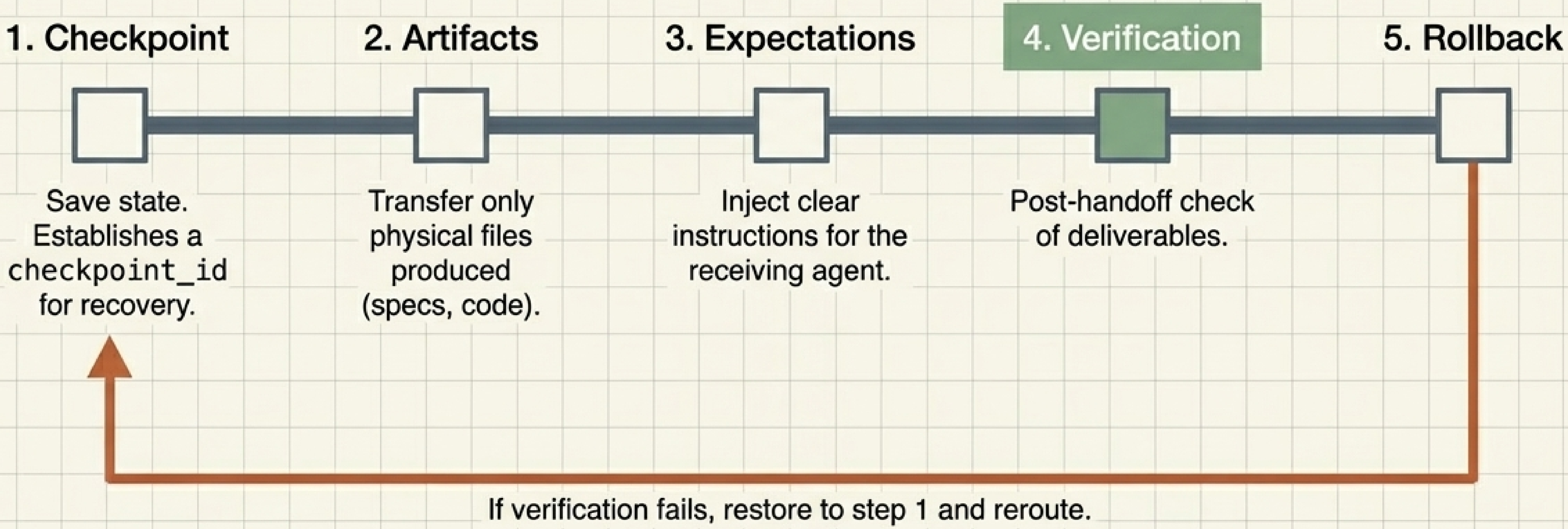

Input: Incoming Task
Capability Router
(Subtext: Matches task to Agent's accepts list)
Pre-Dispatch Validator
(Subtext: Checks if all requires artifacts exist on disk)
True
False
Dispatch to Agent
Halt: Missing Dependency Error

The Golden Rule of Handoffs: Pass Artifacts, Not Context

The receiving agent does not need the sending agent’s conversation history. Passing full context wastes tokens and confuses downstream agents. Handoffs must pass clean file references, never raw chat logs.


10,000 Tokens of Internal Reasoning, False Starts, and Chat Logs
The Artifact Filter
The Artifact File
Receiving Agent

The Handoff Protocol Lifecycle



1. Checkpoint

Save state. Establishes a `checkpoint_id` for recovery.

2. Artifacts

Transfer only physical files produced (specs, code).

3. Expectations

Inject clear instructions for the receiving agent.

4. Verification

Post-handoff check of deliverables.

5. Rollback

If verification fails, restore to step 1 and reroute.

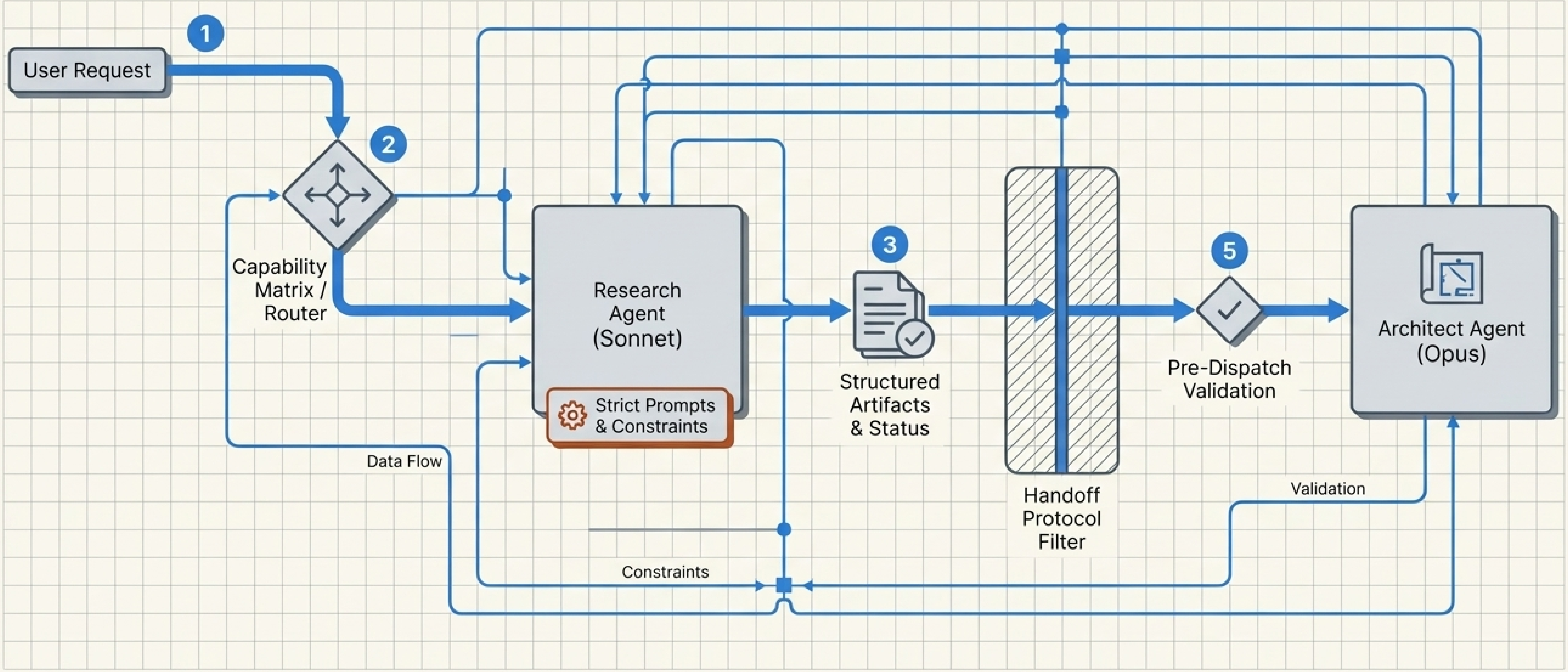
Model Selection Matrix

Match model capability to task complexity. Override at runtime for critical paths.

OPUS	SONNET	HAIKU
Highest	Medium	Lowest
Complex reasoning, nuanced decisions.	Heavy lifting, context processing.	Mechanical checks, simple transformations.
Orchestrator, Critic (blocking gates), Architect (greenfield).	Research, Builder, Documentation, Architect (standard).	Hook evaluators, format validation, routing decisions.

Synthesis: The Coordinated System in Action

From static design to dynamic execution. Isolated concepts weave together into a fully functioning, deterministic orchestration engine.



Quality Assurance: Testing Agent Networks

Unit Testing (Format Verification)

Invoke the agent with known input. Validate that the output perfectly matches the required structural template and machine-readable statuses.

Integration Testing (Handoff Continuity)

Execute a two-agent handoff. Verify that artifacts transfer correctly and downstream pre-dispatch validation passes without hallucinations.

Behavioral Testing (Constraint Enforcement)

The ultimate safety net. Deliberately request constraint violations (e.g., asking a researcher for code) and verify the agent refuses and self-corrects.

Diagnostic Guide: Common Design Mistakes

Symptom	Diagnosis	Prescription
Agent tries to do everything; quality is uniformly poor.	The Swiss Army Agent	Split into 3-5 focused agents. One agent, one job.
Receiving agent ignores instructions or hallucinates based on abandoned reasoning.	The Chatty Handoff	Pass file references and structured envelopes, never raw chat logs.
Handoffs fail silently; orchestrator hangs.	The Invisible Agent	Enforce mandatory Artifacts Produced and Status blocks in output templates.
Agent is polite but constantly drifts out of scope.	The Constraint-Free Agent	Implement explicit MUST NOT rules to establish hard boundaries.

The Golden Rule of Orchestration

One Agent. One Job. One Artifact Type.

Garbage in at the agent level means garbage out at the system level. The reliability of your orchestrator is entirely bound by the structural discipline of your individual agents.