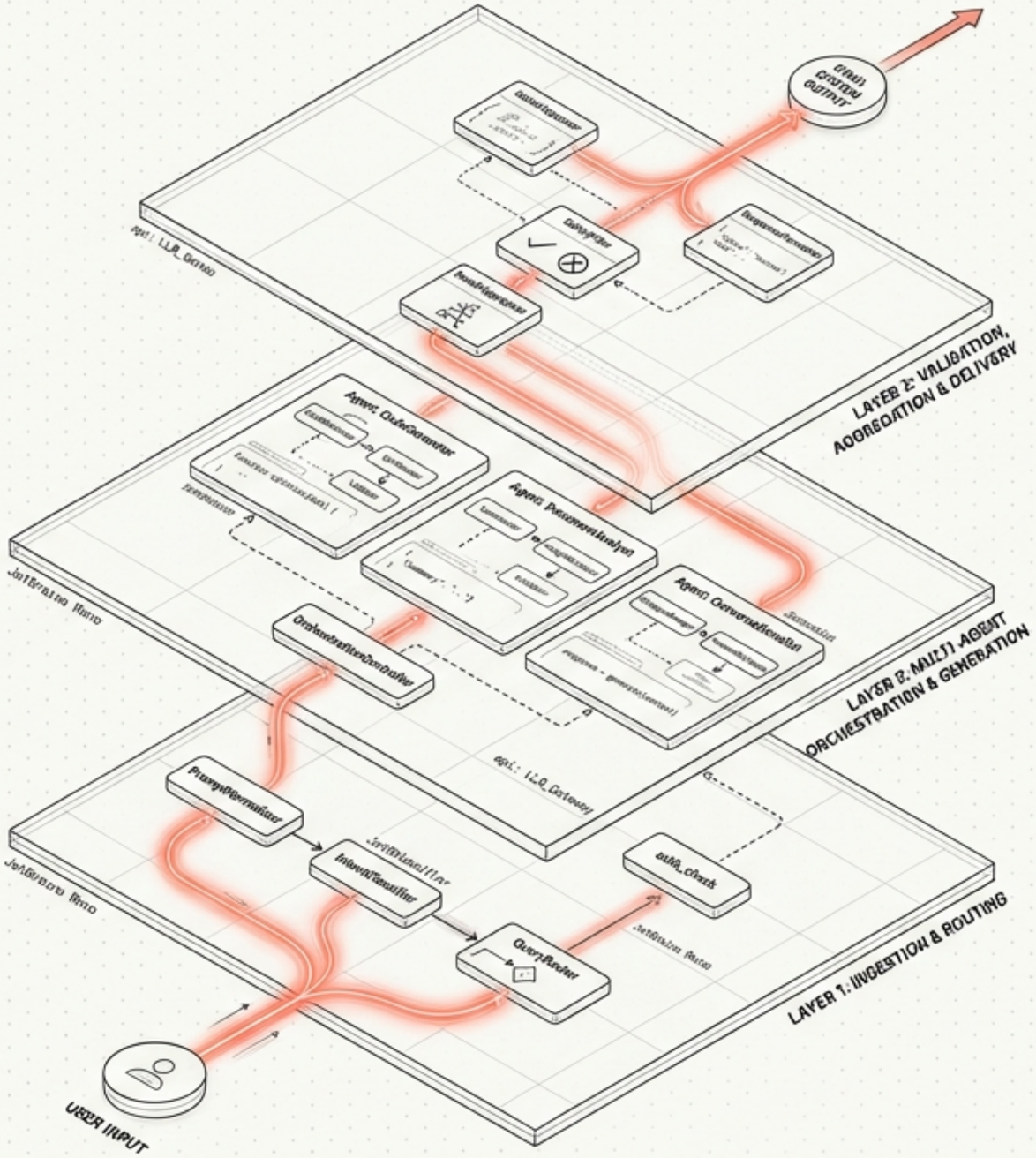
ENTERPRISE LLM ARCHITECTURE

Design Patterns, Anti-Patterns, and System Workflows for Production Deployments

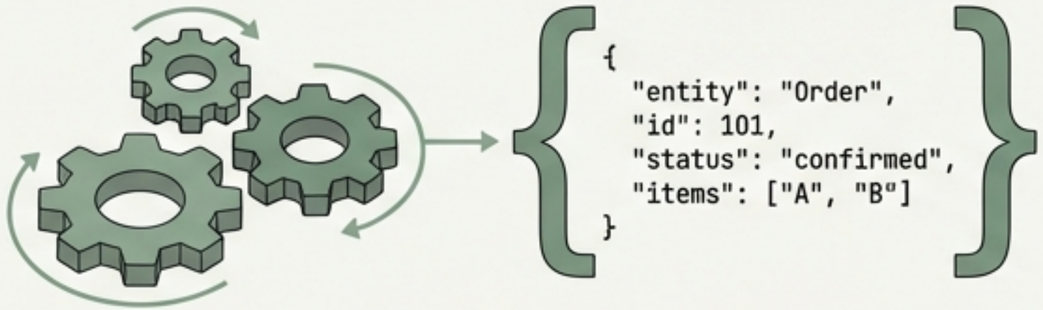
The Architect's Playbook



Four Domains of AI Architecture

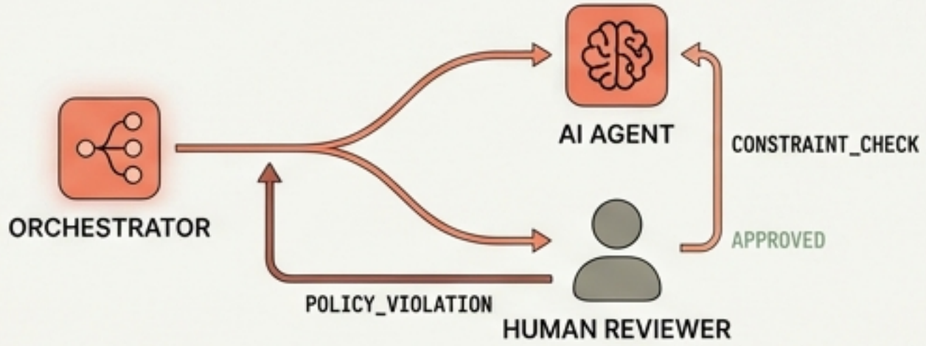
Structured Data Extraction

High Volume, Strict Schemas, Batch Pipelines



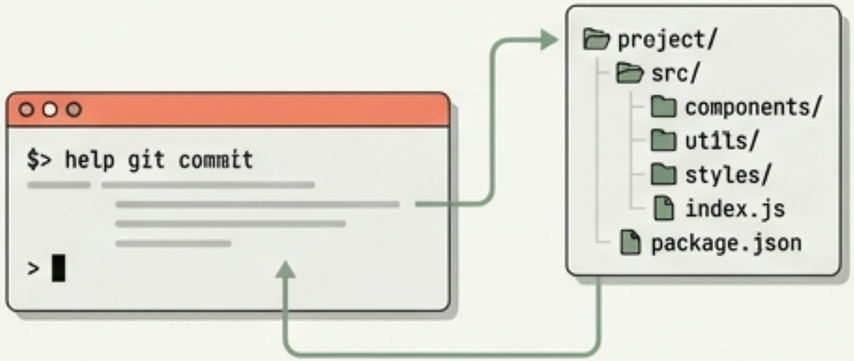
Customer Support Orchestration

Stateful, Human-in-the-Loop, Policy Constraints



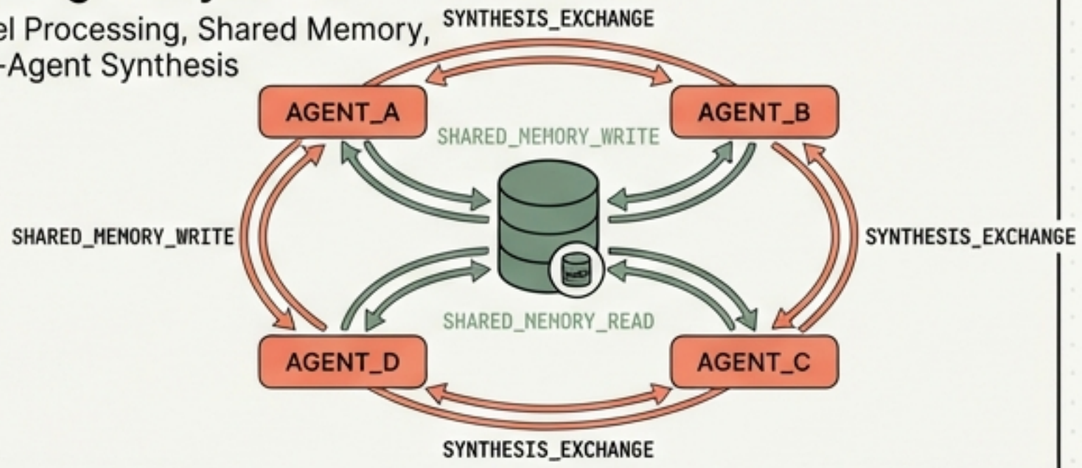
Developer Productivity

Dynamic Tasks, Iterative Context, Advanced Tool Use

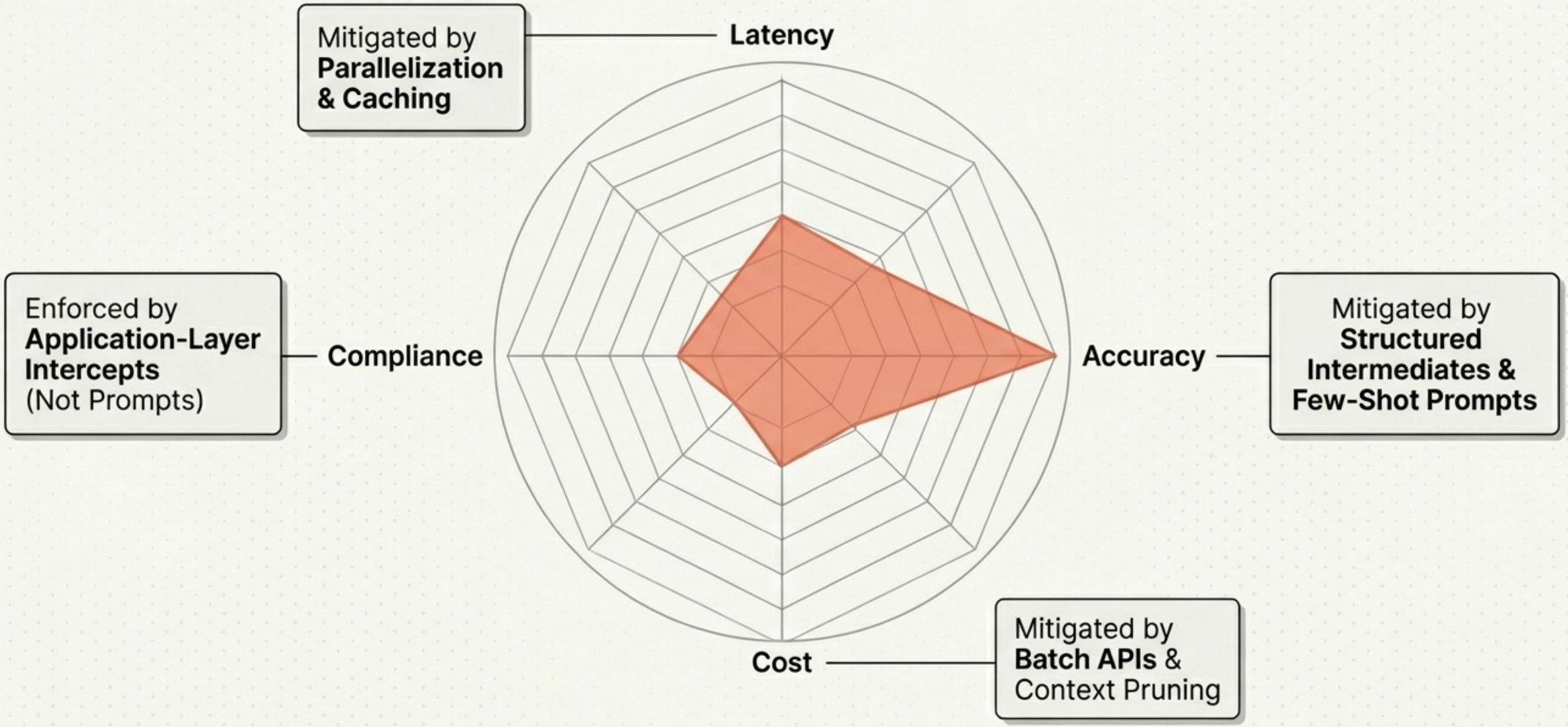


Multi-Agent Systems

Parallel Processing, Shared Memory, Cross-Agent Synthesis

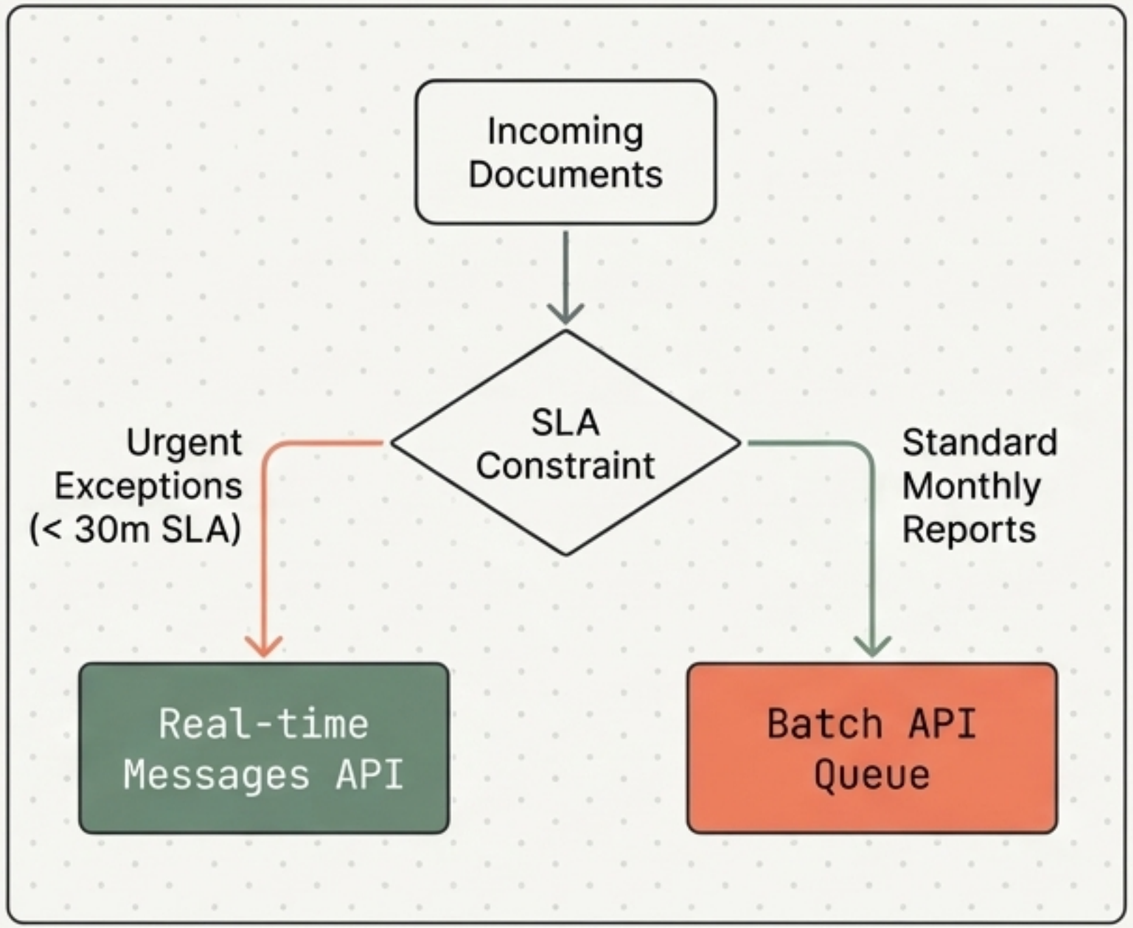


The Architect's Hierarchy of Constraints



Routing for Cost and SLA

Rule: Never default to real-time for asynchronous needs.



Urgent Exceptions	`Real-time Messages API` (High Cost, Instant Latency)
Standard Workflows	`Message Batches API` (50% Cost Savings)
Continuous Arrival (30h SLA)	Submit batches every 6 hours containing documents from that window.

Designing Resilient Schemas

Anti-Pattern: Fragile Expansion

Continuously expanding enums as edge cases arise.

```
// Fragile Schema with Restricted Enum
{
  "type": "object",
  "properties": {
    "property_type": {
      "type": "string",
      "enum": ["house", "apartment", "condo", "townhouse"]
    }
  },
  "required": ["property_type"]
}

// Fails validation:
// "property_type": "studio"
// "property_type": "converted warehouse"
```

VALIDATION ERROR

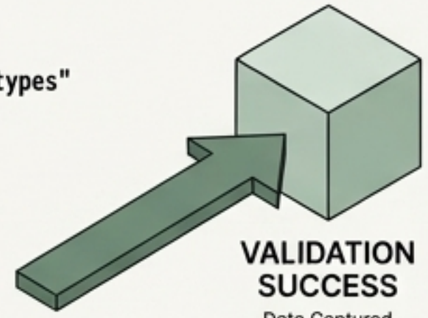
Unexpected Types

Architectural Pattern: Resilient Catch-Alls

Add an other value to the enum, paired with a detail string field.

```
// Resilient Schema with Catch-All
{
  "type": "object",
  "properties": {
    "property_type": {
      "type": "string",
      "enum": ["house", "apartment", "condo", "townhouse", "other"]
    },
    "property_type_detail": {
      "type": "string",
      "description": "Specifics for 'other' types"
    }
  },
  "required": ["property_type"]
}

// Successfully processes:
// "property_type": "other",
// "property_type_detail": "studio"
```



VALIDATION SUCCESS

Data Captured

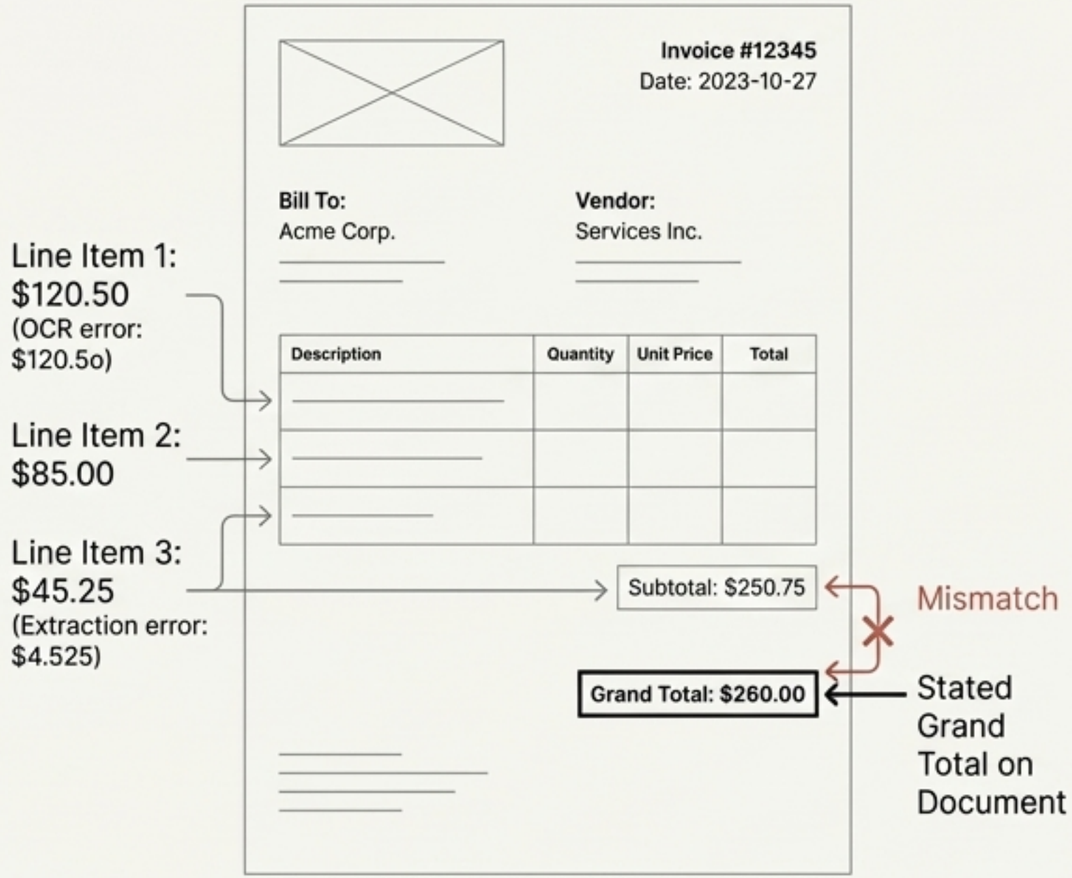
Data Evolution Rule

For amended documents, redesign schemas so amended fields capture multiple values, each with a source location and effective date, rather than overwriting original terms. Validate this approach against the problem of extracting both original and amended contract clauses.

```
// Exawple Structore for Amended Fields
"payment_terms": [
  { "value": "30 days", "source": "Original Contract, Clause 4.1", "effective_date": "2023-01-01" },
  { "value": "45 days", "source": "Awendwent 1, Clause 2", "effective_date": "2823-06-15" }
]
```

Enforcing Mathematical Consistency

The Problem: 18% of invoice extractions show line items that don't match the grand total due to OCR or extraction errors.



Schema Solution: Redundancy

```
{
  "invoice_id": "12345",
  "line_items": [
    { "description": "Item 1", "amount": 120.50 },
    { "description": "Item 2", "amount": 85.00 },
    { "description": "Item 3", "amount": 4.525 }
  ],
  "calculated_total": 210.025,
  "stated_total": 260.00,
  "currency": "USD"
}
```

Derived by model summing items

Extracted directly from page

The Solution: Schema Redundancy

Routing Action: Flag the record for human review ONLY when `calculated_total != stated_total`.

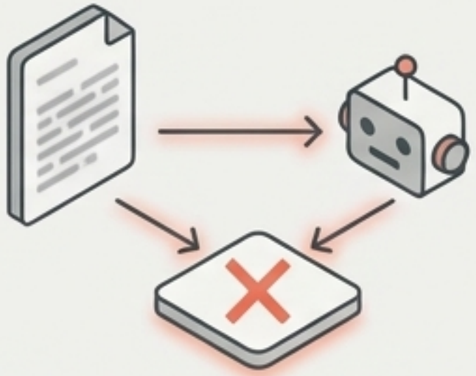
Normalization and Null Handling

Base Prompt

Input:
Extract attendee count and materials.

Model Output (Problematic):

```
{
  "attendee_count": "500", // Plausible Hallucin
  "materials": "cotton blend" // Inconsistent Fo
}
```



Null Handling Instruction

Problem: Plausible Hallucinations
When fields are nullable, models may invent plausible data (e.g., attendee count: 500) if not explicitly instructed.

Pattern: Add explicit prompt instructions to return null if not directly stated.

Updated Prompt:
Extract attendee count and materials. If attendee count or materials are not mentioned in the text, return "null".

Corrected Output:

```
{
  "attendee_count": null, // Correctly Handled
  "materials": "cotton blend"
}
```

Format Normalization

Problem: Inconsistent Formats
For materials ("cotton blend" vs "Cotton/Polyester mix"), provide few-shot examples showing 2-3 complete input-output pairs with standardlzed formats. Do not rely on temperature 0 alone.

Solution: Few-Shot Standardization

Final Prompt with Examples:
Extract attendee count and materials. If not mentioned, return "null'. Materials must be standardized.
Examples:
Input: "Made of cotton blend." -> Output: {\"materials": \"Cotton Blend"}
Input: "Cotton/Polyester mix" -> Output: {\"materials": \"Cotton/Polyester Mix"}

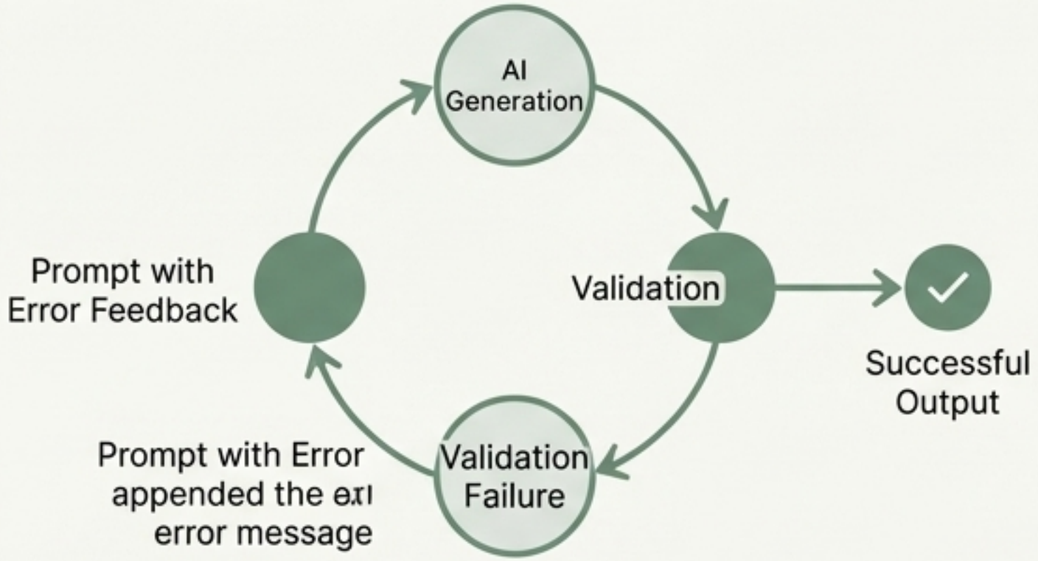
Final Output:

```
{
  "attendee_count": null,
  "materials": "Cotton Blend" // Standardized
}
```

The Limits of Automated Retry

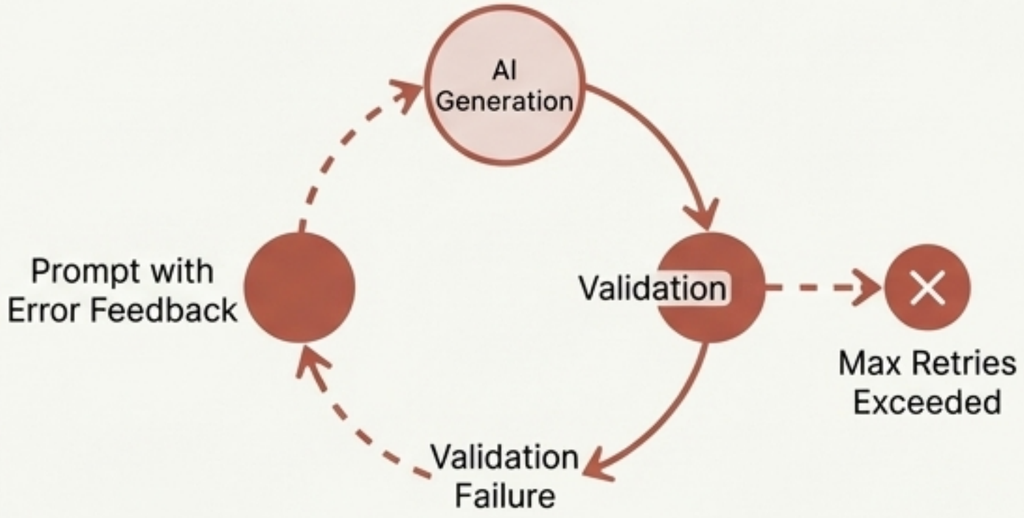
The Pattern: Appending specific validation errors to the prompt and retrying resolves most failures in 2-3 attempts.

Effective: Formatting Errors



Fixing nested objects vs flat arrays.
Resolving locale-formatted strings.

Ineffective: Missing Information



The Exception: Retries are least effective for missing information (e.g., trying to extract full author lists when the source says “et al.” and points to an unprovided external document). Recognize when to fail fast.

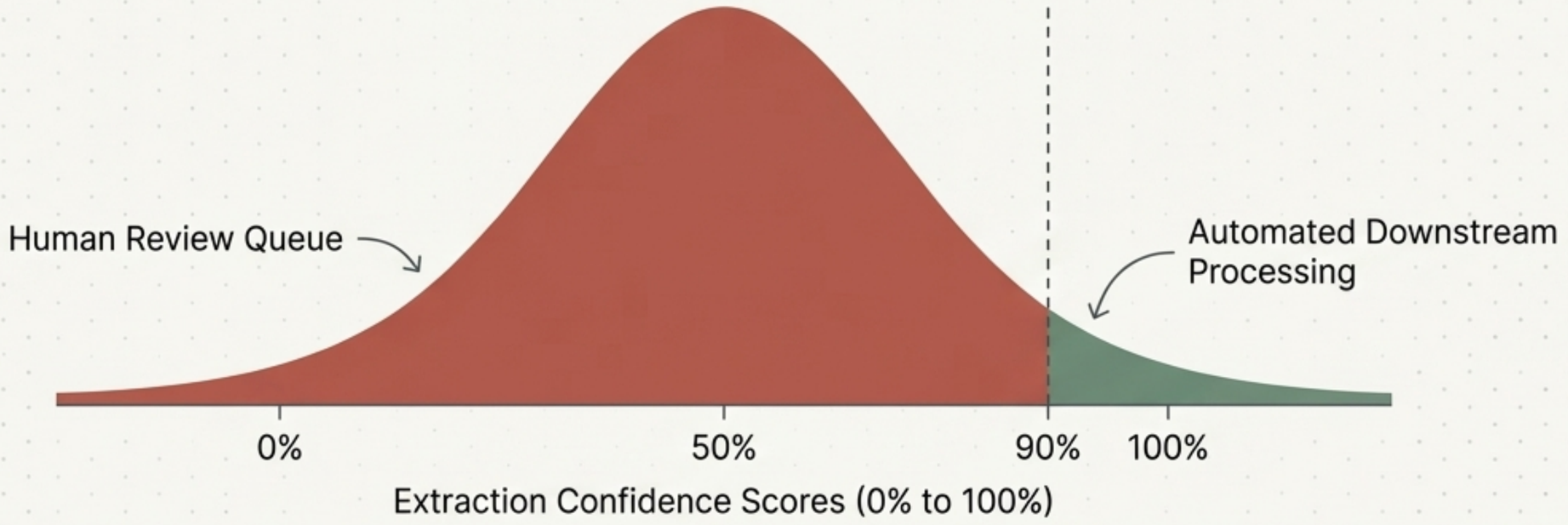
Calibrating Human-in-the-Loop

Requirement:

Automate extractions with model confidence >90%.

Implementation:

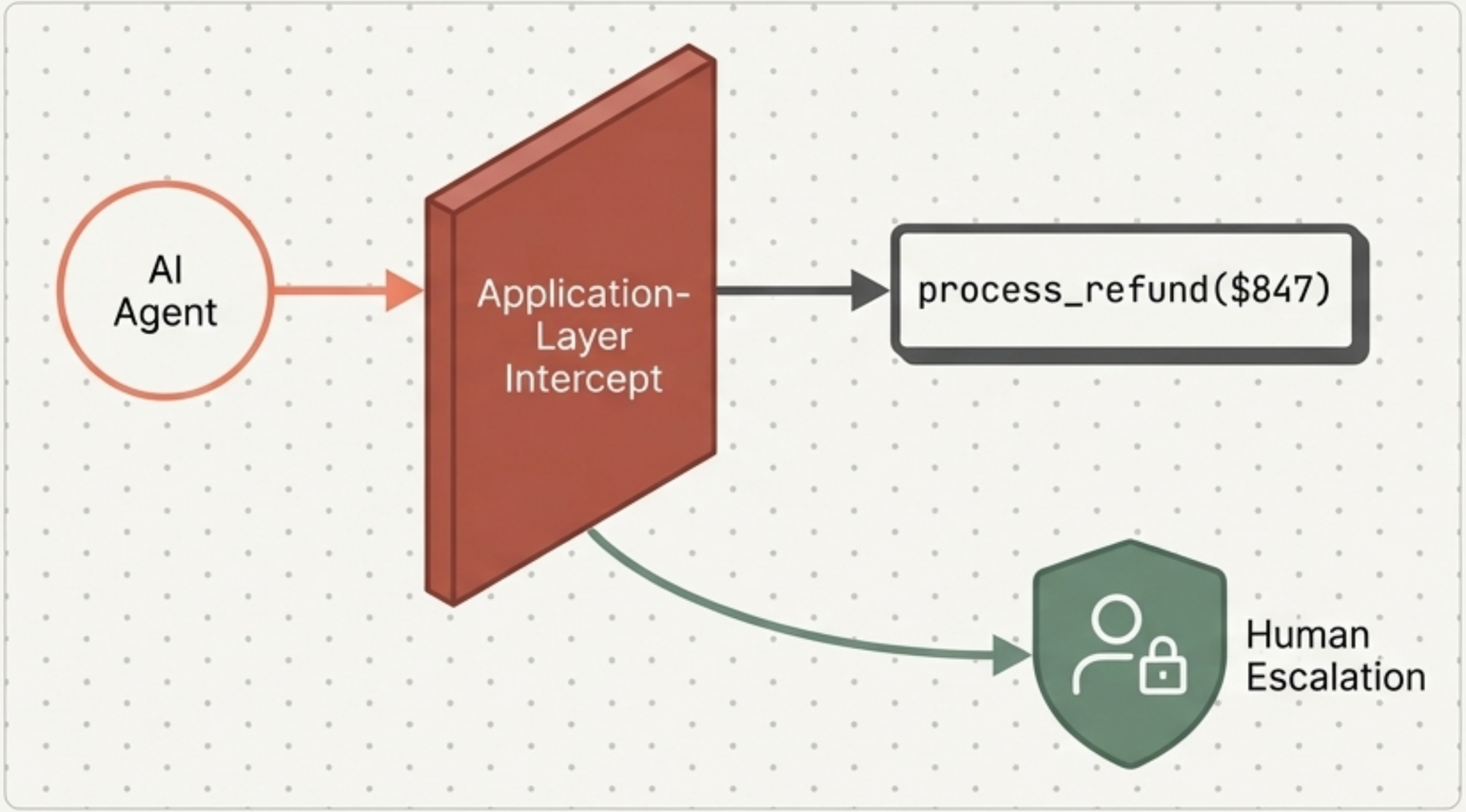
Have the model output `field-level confidence scores`. Ground this implementation detail in the solution for reducing semantic errors.



Critical Validation Step: Analyze accuracy by document type and field to verify high-confidence extractions perform consistently across all segments, not just in aggregate, before deploying.

Zero-Tolerance Compliance

The Trap: Relying on emphatic system prompts ("CRITICAL POLICY: NEVER process >$500") still yields a 3% failure rate.



The Architectural Standard:

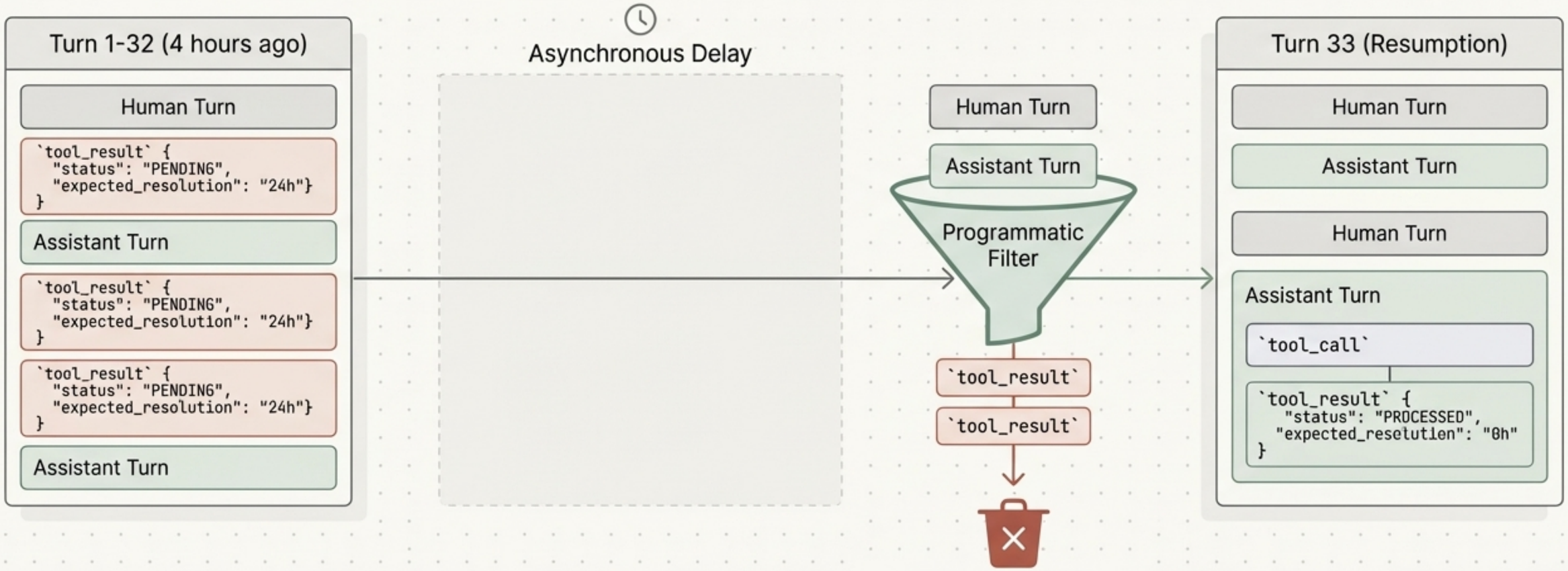
Implement an application-layer hook to intercept tool calls.

When the process amount exceeds the threshold, block it server-side and invoke escalation.

Model discretion is removed.

Resuming Asynchronous Sessions

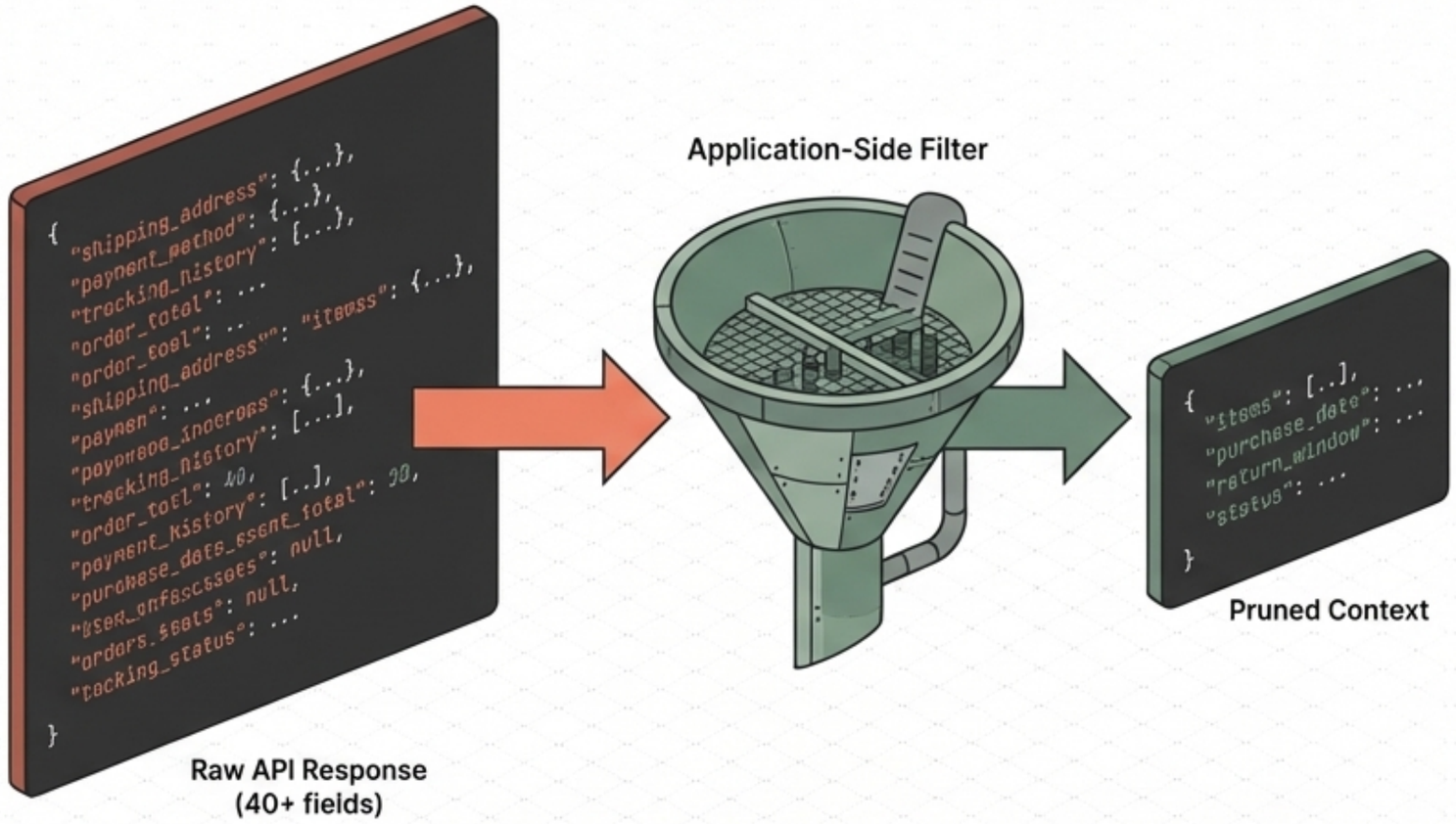
The Problem: Resuming a session hours later leads to the model confidently stating outdated status (e.g., "Expected resolution: 24h" from a previous tool call).



The Solution: Resume with full conversation history, but programmatically filter out previous `tool_result` messages. Keep only human/assistant turns so the agent is forced to re-fetch needed data upon resumption. This ensures returning customers always receive **fresh**, **current** information, preventing the use of stale data.

Tool Context Pruning

The Bloat: Repeatedly calling `lookup_order` fills the context window with verbose shipping and payment data when only the return status is needed.

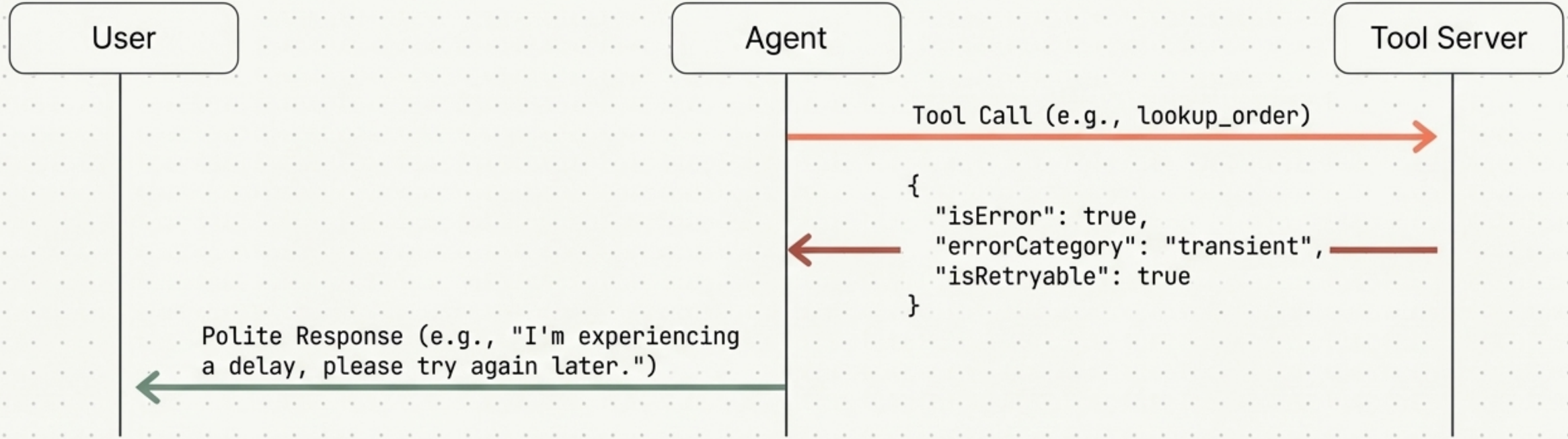


The Pattern: Application-side filtering.

Extract only relevant fields (items, purchase data, return window, status) from each existing order response, removing verbose details before the conversation to proceed.

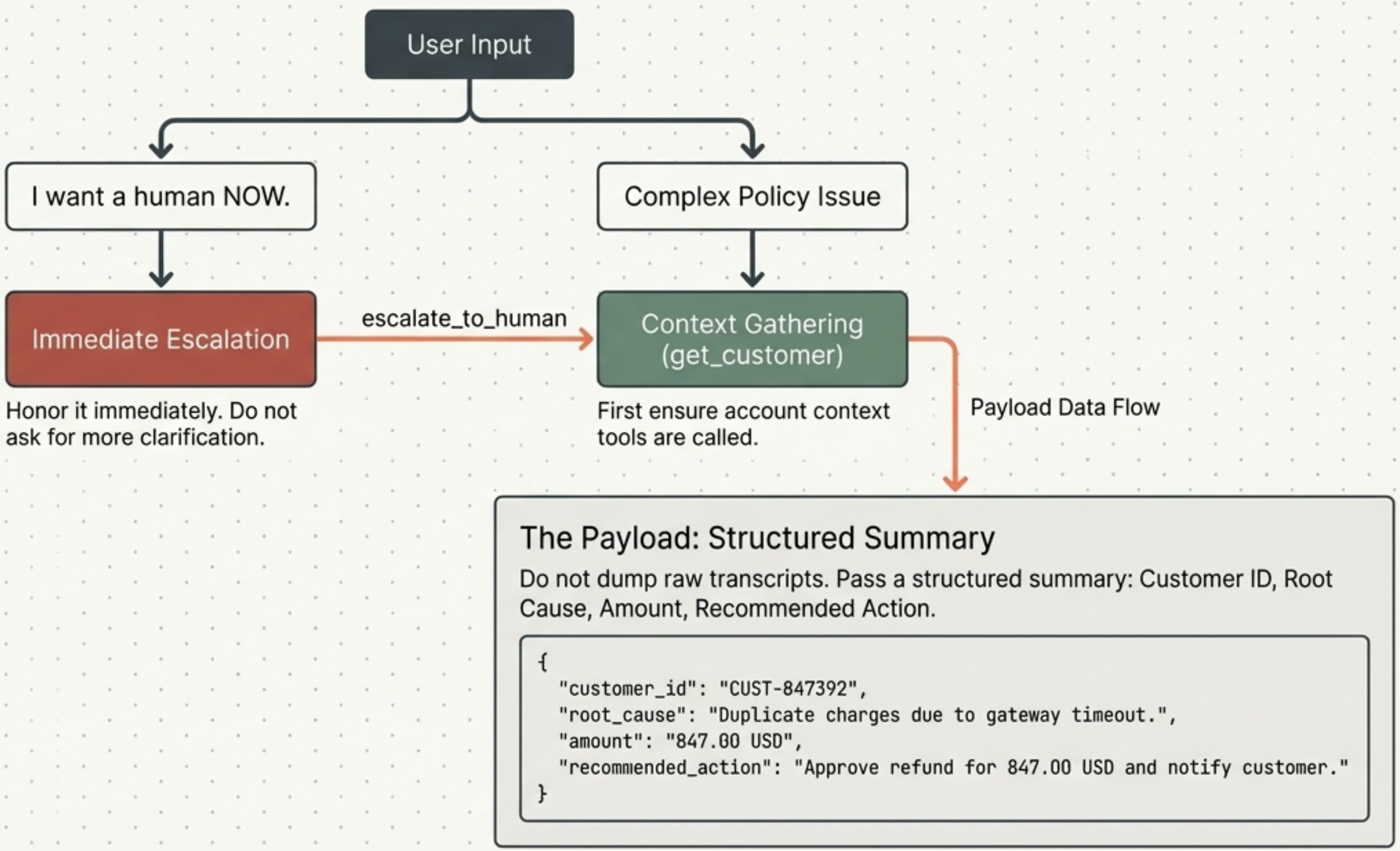
This strategy aligns with managing multiple extensive tool responses in a support session.

Graceful Tool Failure



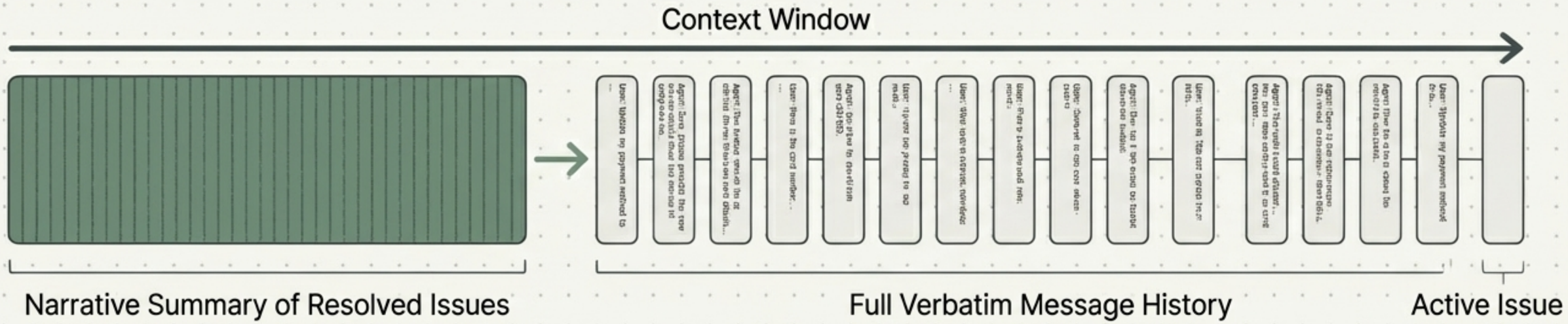
- **Anti-Pattern:** Throwing application exceptions that crash the agent, or returning empty strings.
- **Correct Pattern:** Return the error message in the tool result content with the `isError` flag set to true.

The Escalation Handoff



Compressing Long Sessions

The **Challenge:** A single session covers a refund inquiry, a subscription question, and a payment update across 48 turns. Context limits approach.



- **The Strategy:** Summarize earlier, resolved turns into a narrative description, preserving the full message history verbatim only for the active, unresolved issue.

  Correct Pattern: Return the error message in the tool result content with the `isError` flag set to `true`.

MCP Tool Specificity

The Trap: Providing a broad custom tool (analyze_dependencies) alongside built-in tools like Grep. The agent defaults to Grep.

Anti-Pattern: Monolithic Tool

```
$ Agent execution...

Agent uses built-in Grep to search imports.
└→ {name: analyze_dependencies}
```

Architect's Pattern: Granular Tools

```
$ Agent execution...

Agent uses custom tool for dependencies.
├→ - list_imports
├→ - resolve_transitive_deps
└→ - detect_circular_deps
```

The Fixes

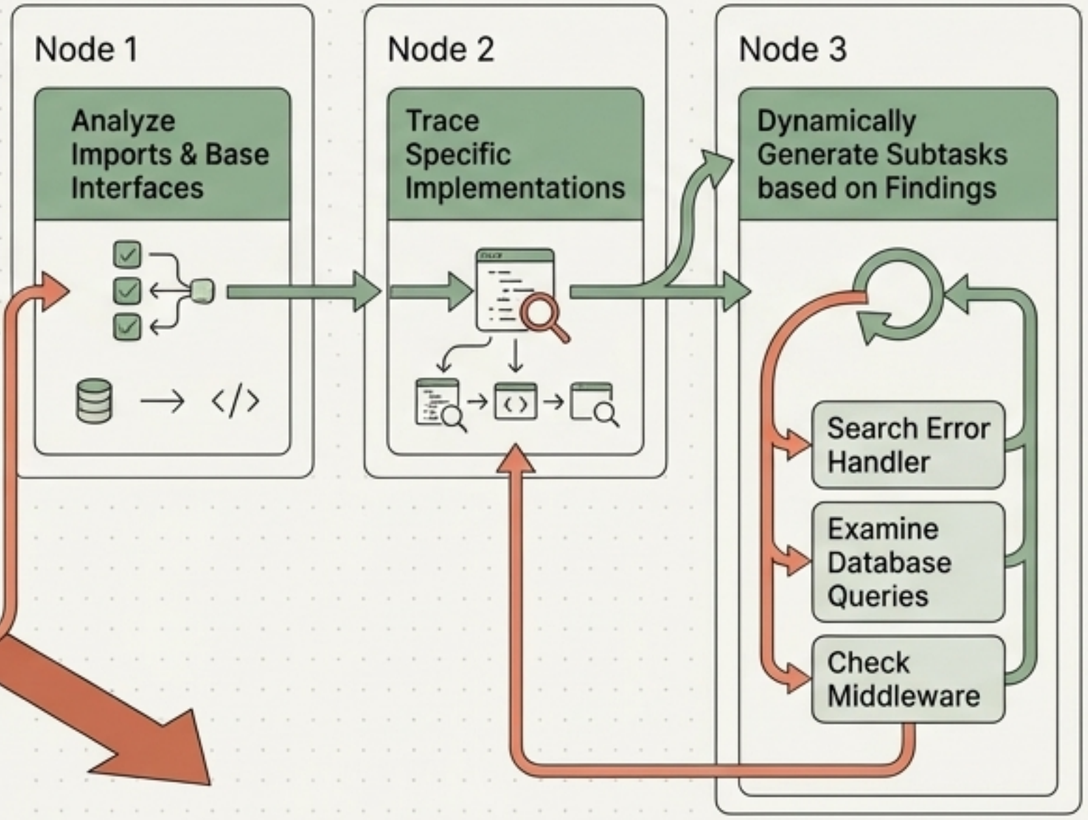
Split broad tools into highly granular, single-purpose tools. Enhance MCP tool descriptions to explicitly detail capabilities, expected outputs, and when to prefer them over text manipulation. This applies similarly to adopting custom refactoring tools over standard Bash/sed.

Directed Codebase Exploration

Anti-Pattern: Using the Read tool to sequentially load thousands of lines of code.

Inefficient & Context-Heavy:
Overloads context window with unrelated data.

Dynamic Investigation Process



The Strategy: Start broad, then pinpoint.

For Architecture (New Engineer, 800+ Files):

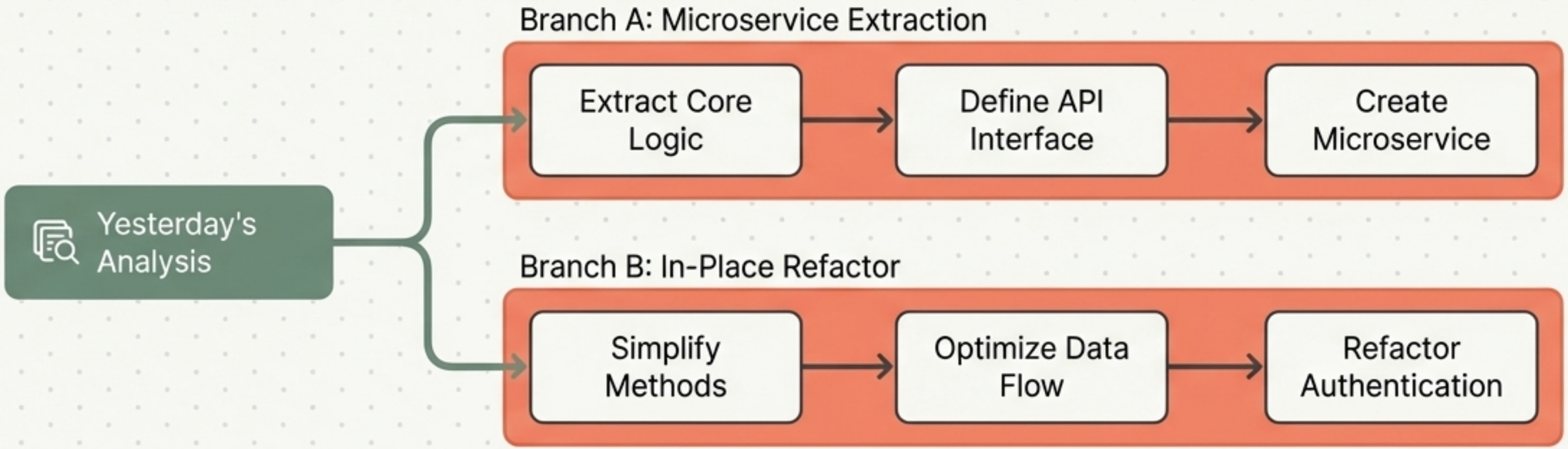
- Read `CLAUDE.md`/`README` first, then ask the human engineer for priority files.

For Intermittent Bugs (Tracing Errors):

- Have the agent dynamically generate investigation subtasks based on what it discovers at each step, adapting the plan as new errors emerge.

Branching Reality

The Problem: Exploring two distinct refactoring approaches or testing strategies in a single thread confuses the agent and mixes context.

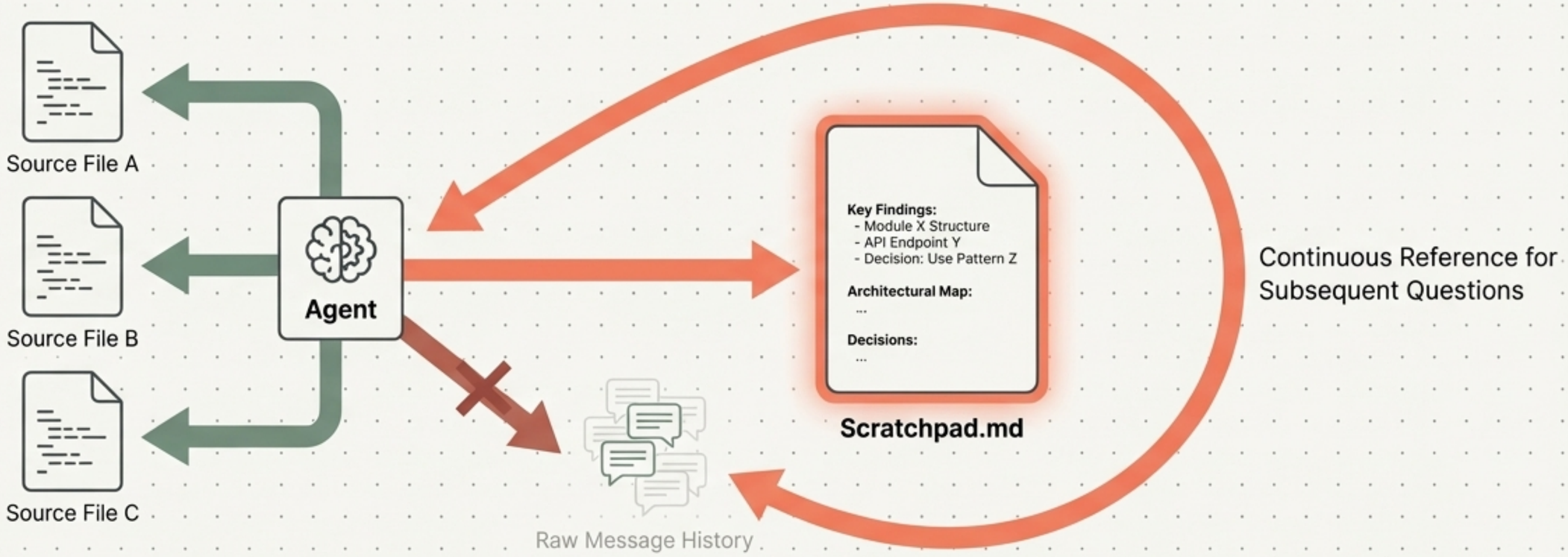


The Command:

Use **fork_session** to create two separate branches from a foundational analysis. This allows independent, deep exploration of A/B scenarios without context contamination.

The Scratchpad Pattern

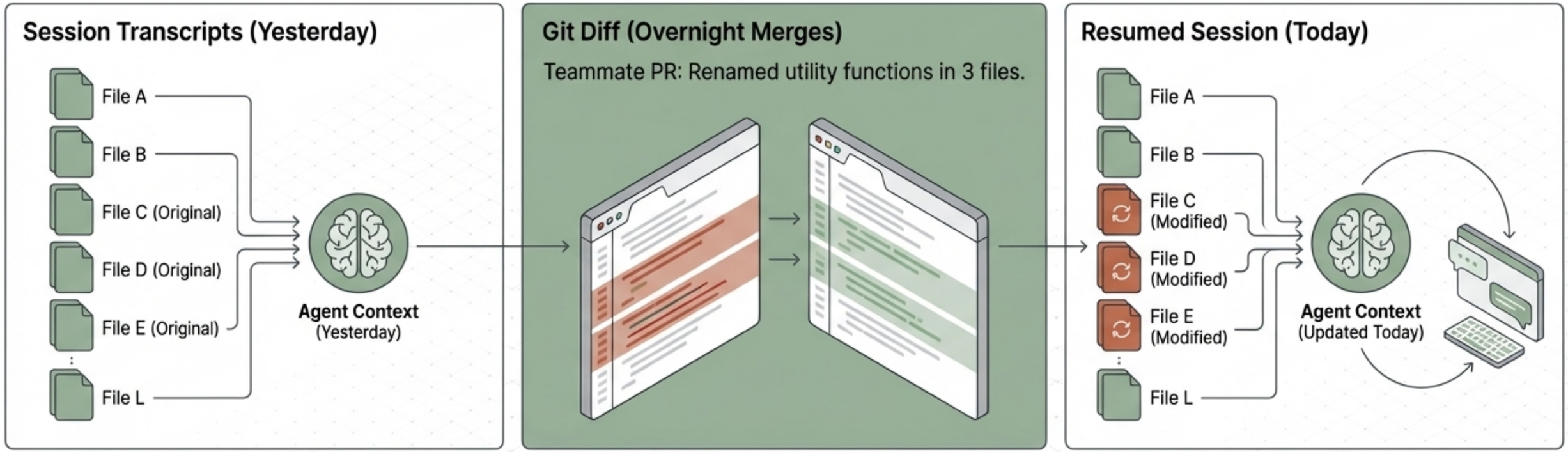
The Decay: In extended exploration sessions (30+ mins), accumulated token bloat causes the agent to give inconsistent answers about early discoveries. Engineers report having to repeat module information.



The Pattern: Have the agent actively maintain a **scratchpad file** recording key findings, architectural maps, and decisions. It references this dense, **structured file** for subsequent questions.

Resumption in Dynamic Environments

The Scenario: An engineer resumes an exploration session, but 3 of the 12 files the agent read yesterday have been altered by a teammate's PR.



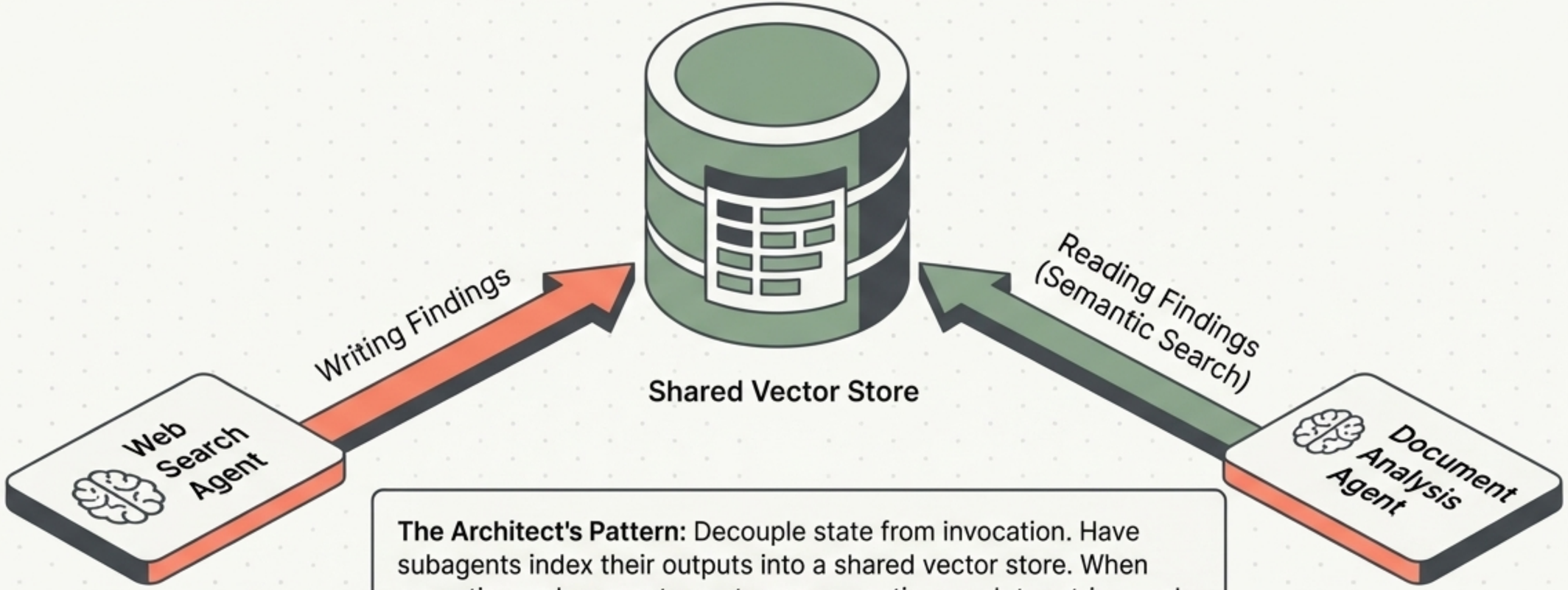
The Action

Resume the session from its previous transcript, but explicitly inform the agent which specific files or functions changed for targeted re-analysis. Do not force a complete re-read, and do not pretend nothing changed.

```
resume_session --update_context={files:['File C', 'File D', 'File E'], changes:'renamed utility functions'}
```

Shared Memory Architecture

Anti-Pattern: Daisy-chaining full conversation logs between subagents. This scales token costs exponentially.



The Architect's Pattern: Decouple state from invocation. Have subagents index their outputs into a shared vector store. When executing, subsequent agents use semantic search to retrieve only relevant prior findings. This architecture prevents state loss when a multi-agent pipeline crashes mid-processing.

Forcing Execution Order

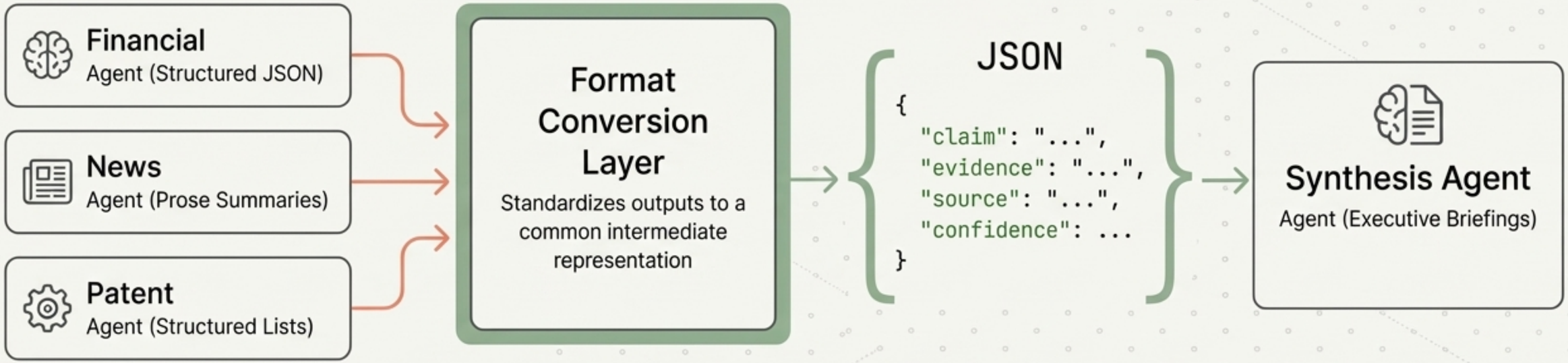
The Problem: An agent needs to extract metadata before calling enrichment tools, but occasionally calls enrichment tools first, leading to failures.

```
{
  "model": "claude-3-opus-20240229",
  "max_tokens": 1024,
  "messages": [
    {"role": "user", "content": "Extract metadata from this paper and then look up its DOI."}
  ],
  "tools": [
    {"name":  "extract_metadata", "description": "..."},
    {"name":  "lookup_citations", "description": "..."}
  ],
  "tool_choice": {"type": "tool", "name": "extract_metadata"}
}
```

The Enforcement: Do not rely on prompt begging. Use the API's constraints. Set `tool_choice` for the first API call to guarantee the pipeline executes in the required order. This ensures structured metadata extraction happens before any DOI lookup or enrichment.

Structured Intermediate Representations

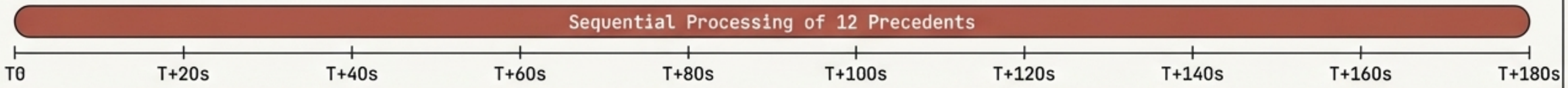
The Loss: Passing raw text from financial and news agents to a synthesis agent results in tables losing clarity and news losing narrative flow.



Citation Rule: To prevent lost attributions, require all subagents to output structured claim-source mappings that the synthesis agent is instructed to preserve.

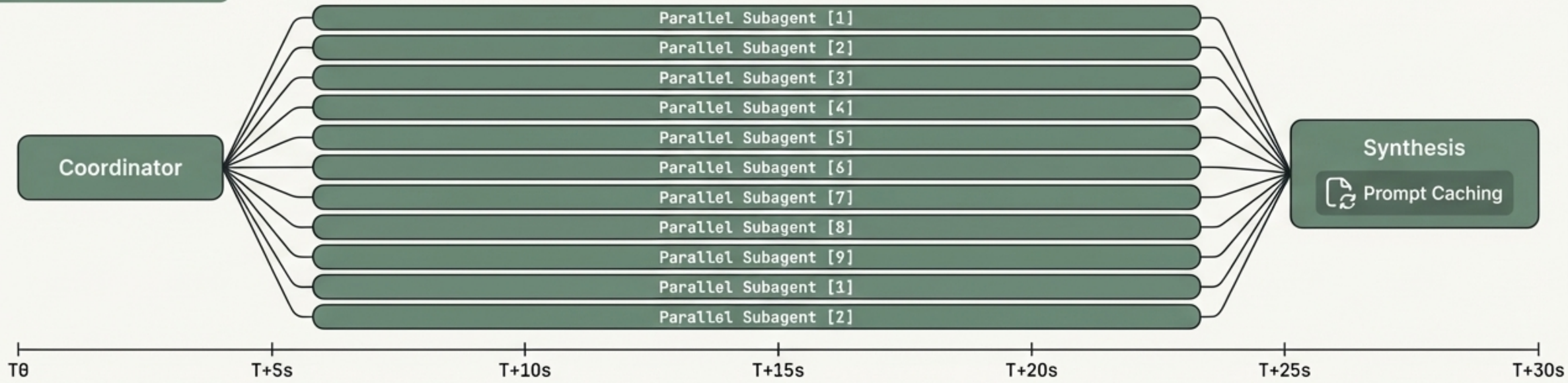
Parallelization & Caching

Serial Processing



The Problem: Processing each precedent sequentially in a complex legal case takes over 3 minutes, creating unacceptable latency.

Parallel Execution



Subagent Parallelism

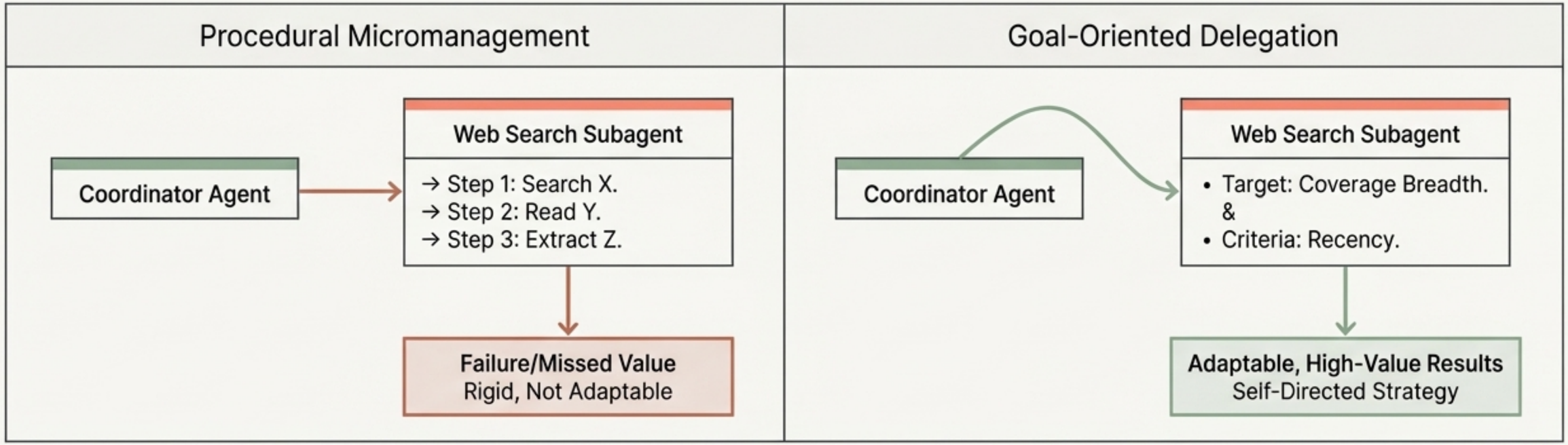
When processing independent data (e.g., 12 legal precedents), the coordinator must spawn parallel subagents, each handling a subset, then aggregate results.

Prompt Caching

When follow-up summaries consistently take 40+ seconds passing 80K+ tokens of accumulated findings, enable prompt caching on the synthesis subagent to drastically reduce transfer overhead.

Goal-Oriented Delegation

The Trap: Giving a web search subagent detailed step-by-step procedural instructions causes it to fail rigidly on emerging topics or miss tangential sources.

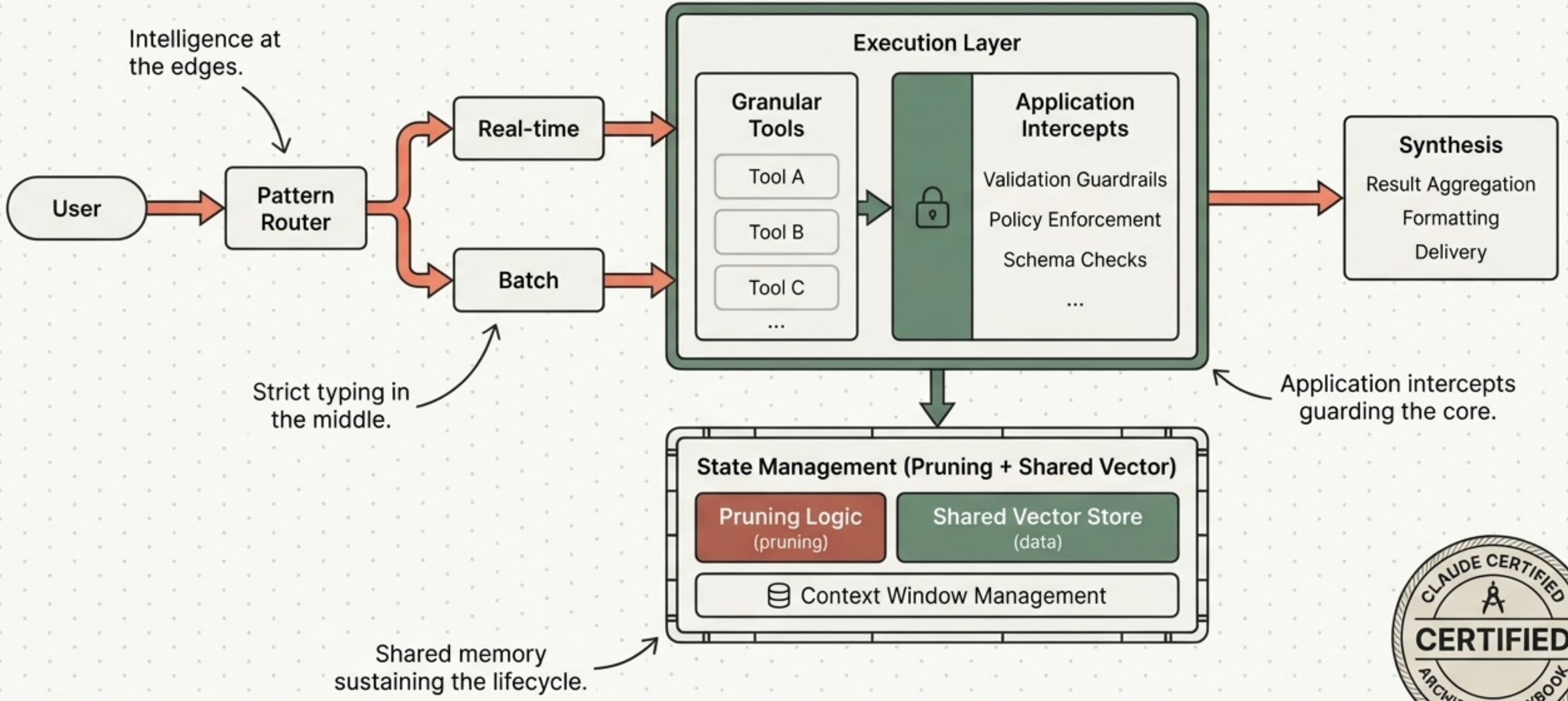


The Architect's Approach:

Specify research goals and quality criteria rather than procedural steps. Let the specialized subagent determine its own search strategy. Keep tool interfaces generic but add enum parameters (e.g., **analysis_type: extraction | summarization**) to guide behavior.

```
tool: 'analyze_document',
params: {
  analysis_type: 'extraction' | 'summarization'
}
```

The Architect’s Reference Matrix

	Data Extraction	Customer Support	Developer Productivity	Multi-Agent
Token Bloat		Filter Stale Results	Scratchpad File	Shared Vector Store
Latency	Batch Routing			Parallelization & Caching
Compliance/Control		App-Layer Intercepts		tool_choice Enforcement
Accuracy	Schema Redundancy		Granular MCP Tools	Structured Intermediates

The Production Architecture Blueprint