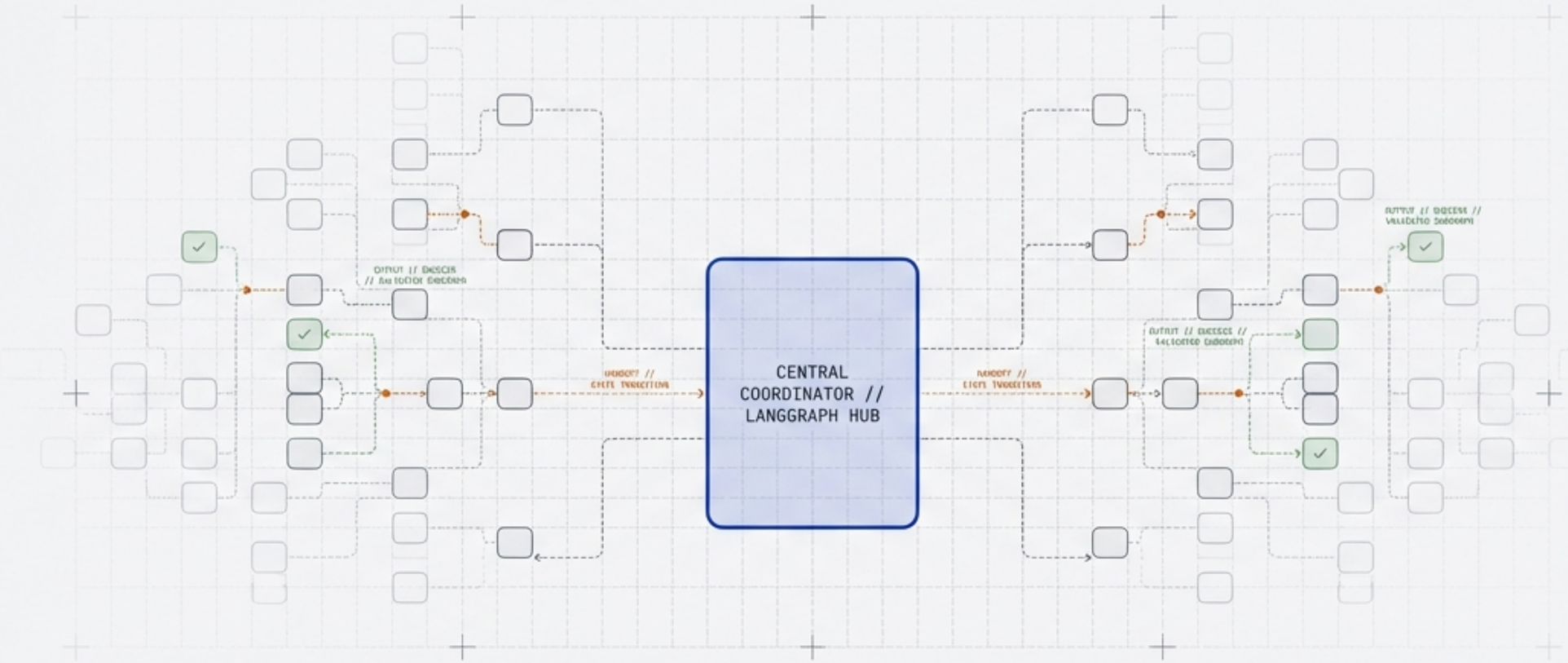



# Architecting Multi-Agent Systems

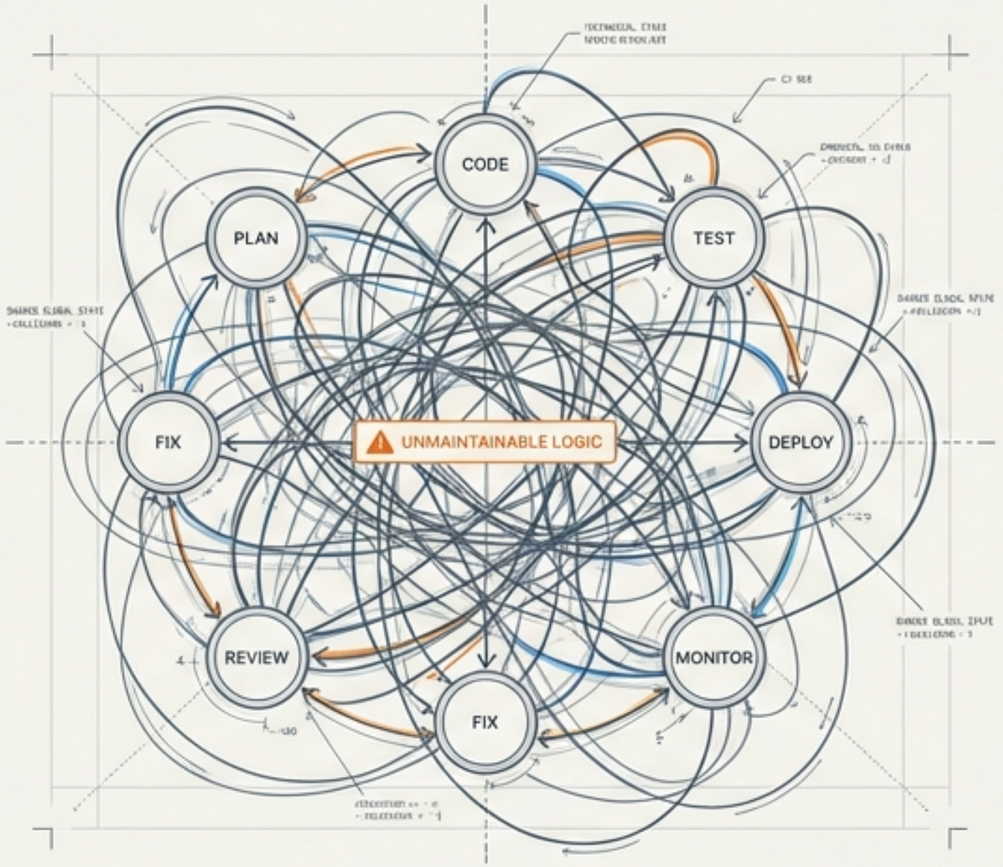
## Mastering Subgraphs, Supervisors, and Handoff Patterns with LangGraph

TECHNICAL DEEP DIVE // FOR DEVELOPERS & ARCHITECTS // SERIES A

# THE COMPLEXITY IMPERATIVE

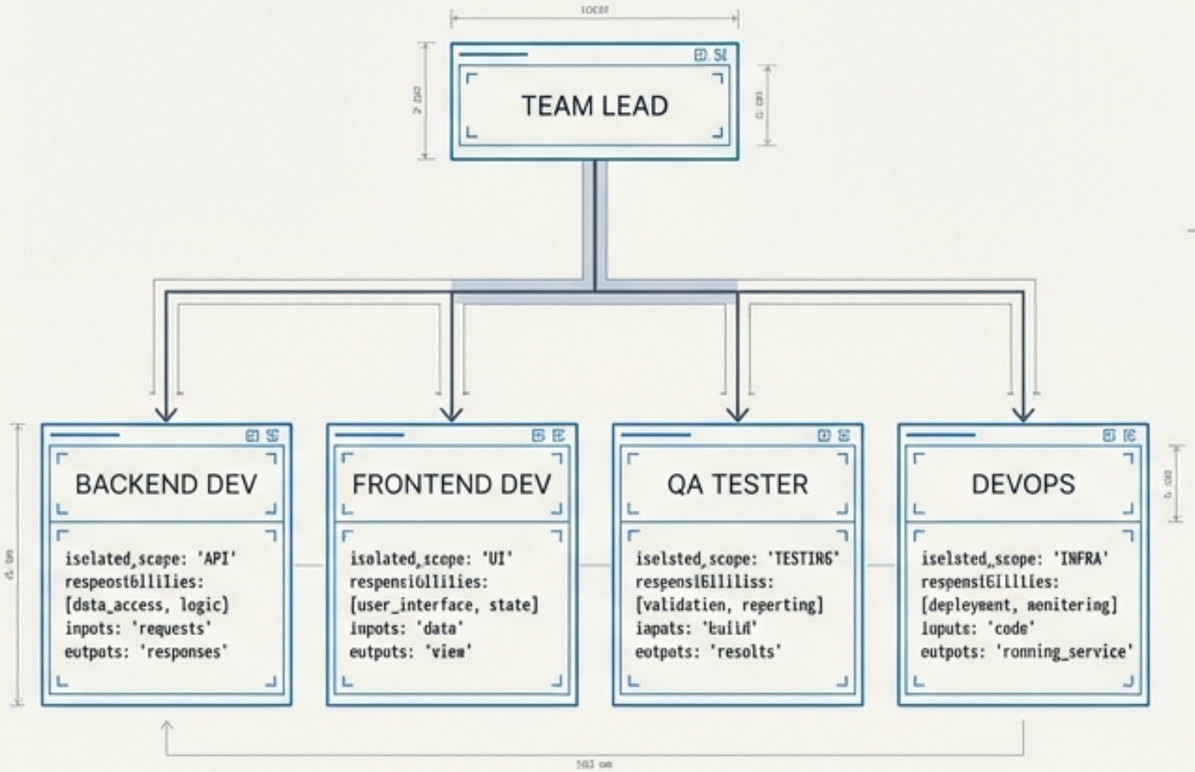
Why monolithic graphs collapse under real-world engineering constraints.

## THE MONOLITH (ANTI-PATTERN)




SHARED GLOBAL STATE

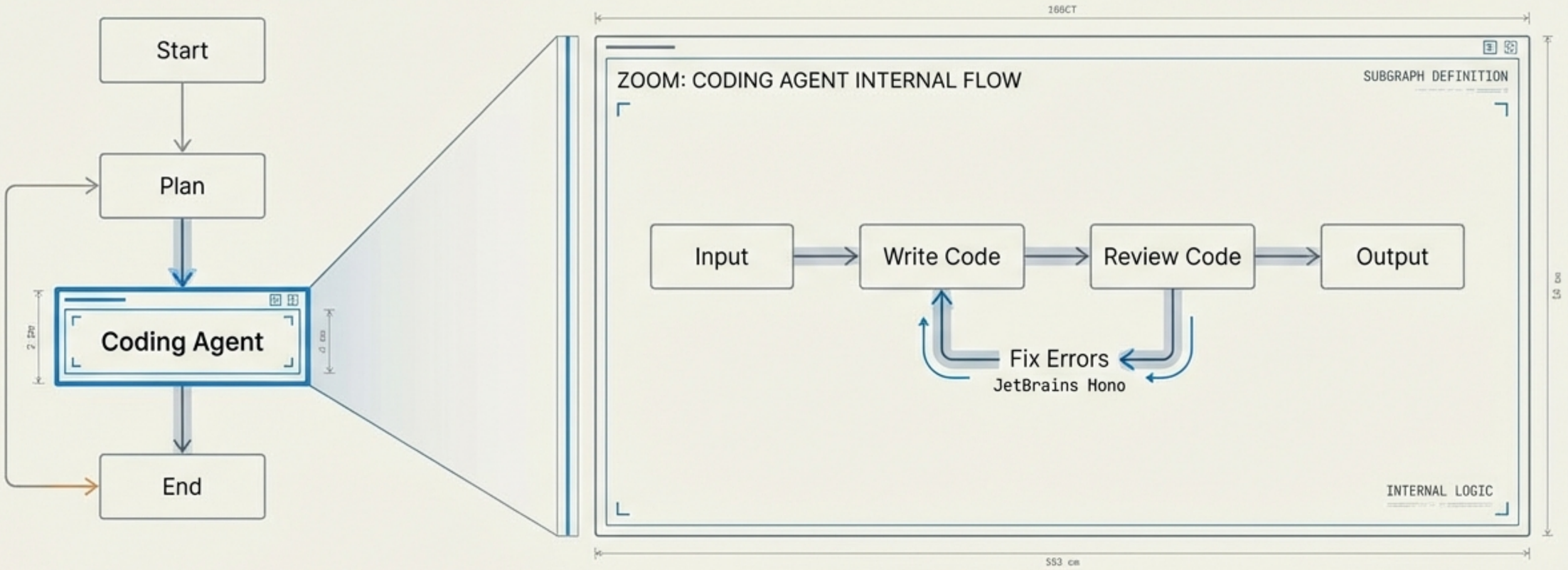
## THE SOLUTION: SUBGRAPHS




SOFTWARE TEAM ANALOGY (ISOLATED RESPONSIBILITIES)

# DEFINING THE SUBGRAPH

A graph executed as a single node within a parent graph.




## MODULARITY

Decouple complex logic into isolated functions.

## REUSABILITY

Deploy the same agent structure for Backend and Frontend tasks.

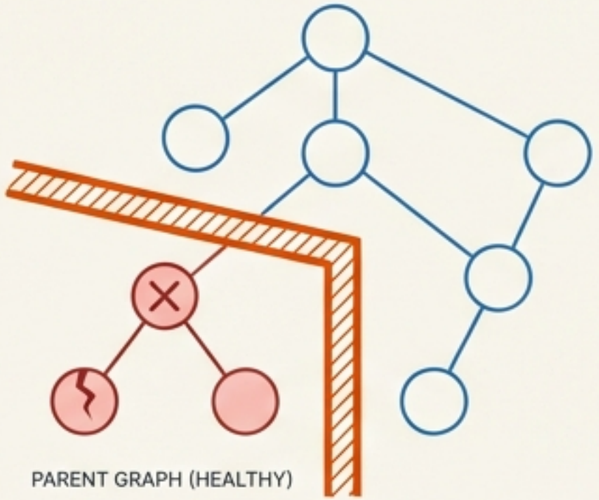
## MAINTAINABILITY

Debug specific flows without traversing the entire system history.

# LangGraph Specific Advantages

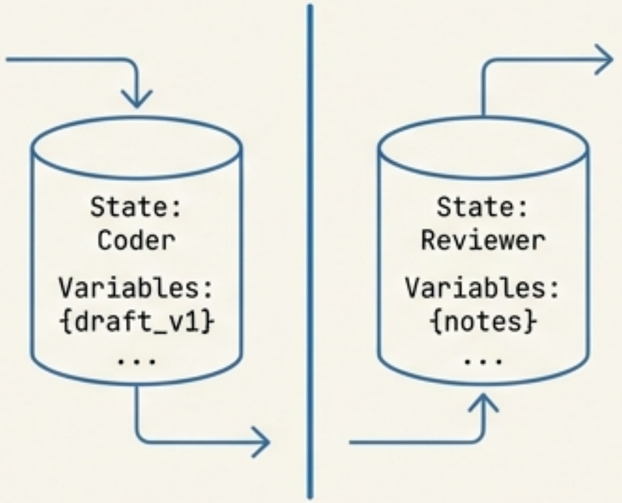
Beyond code organization: Why the framework matters.

## Failure Isolation



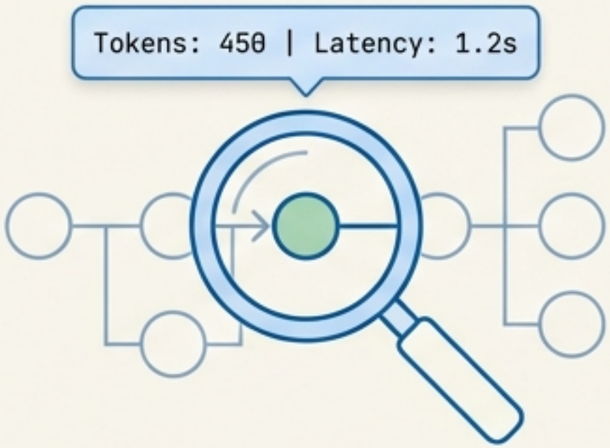

Child graph errors are contained. The Parent Graph handles failures gracefully without a total system crash.

## State Separation




Namespace isolation. Internal variables in one agent cannot accidentally overwrite variables in another.

## Granular Observability




Trace costs and performance per subgraph. Track the "Backend Agent" efficiency independently of the whole.

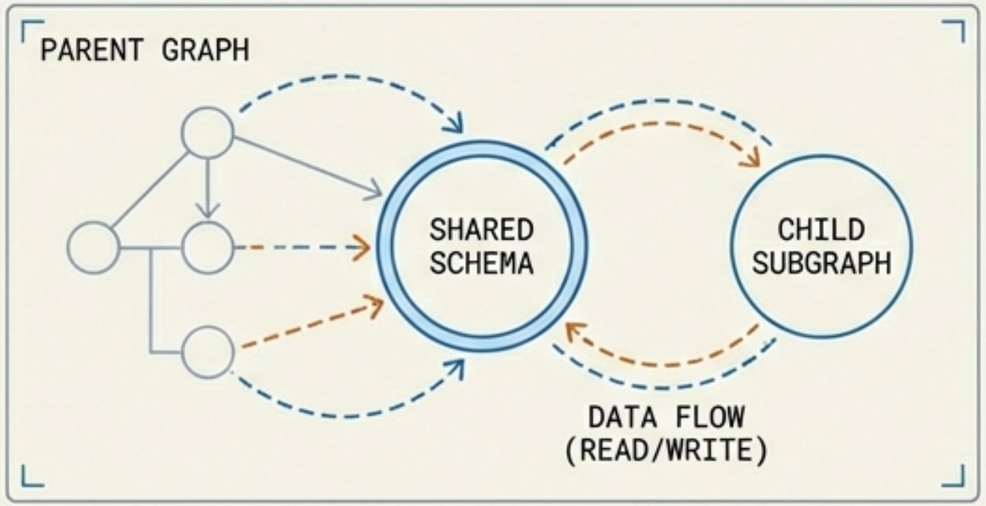
# TWO IMPLEMENTATION MECHANISMS

Architectural choice: How the parent and child communicate.

## Subgraph as a Node

Shared State Architecture

```
builder.add_node("subgraph", subgraph)
```
JetBrains Mono

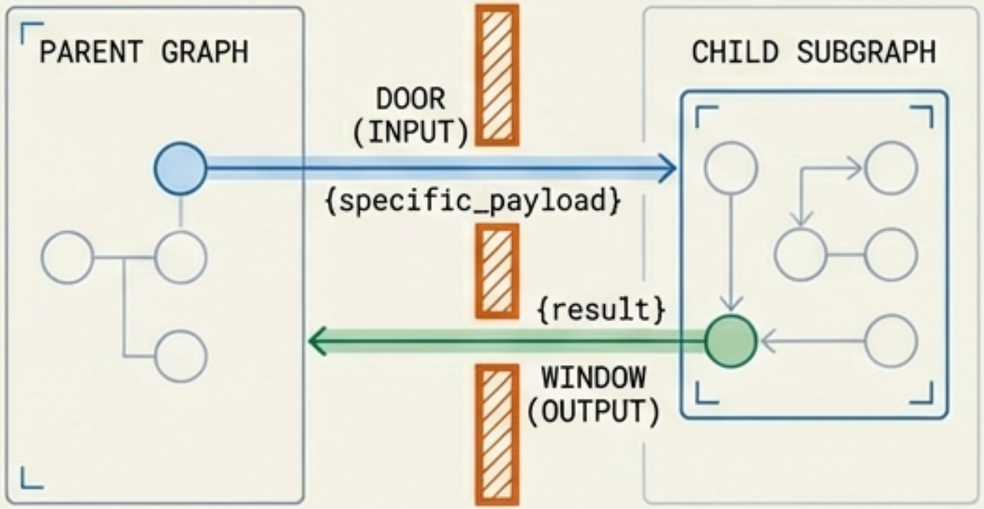



Tight coupling. The child reads/writes directly to the Parent's state.

## Invoking from a Node

Isolated State Architecture

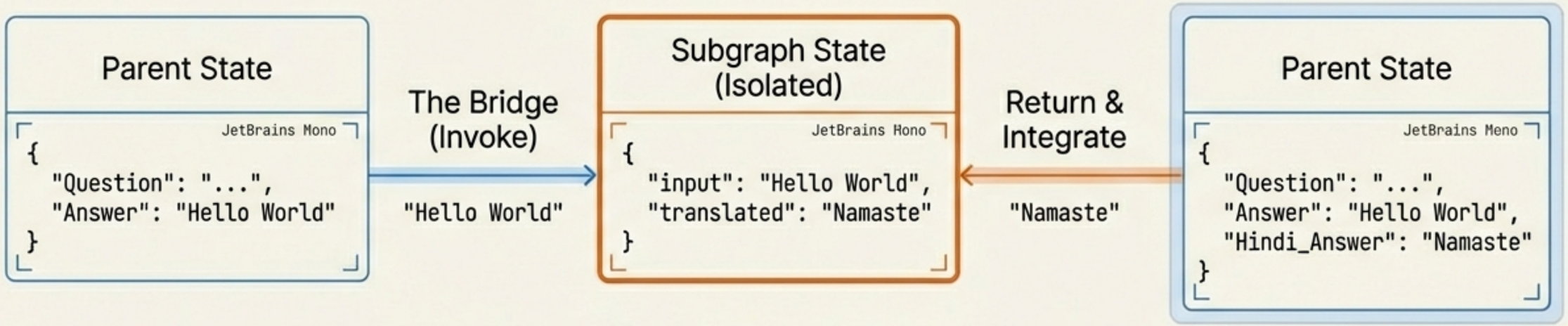
```
subgraph.invoke(input)
```
JetBrains Mono




Loose coupling. The parent passes specific input; the child processes in isolation and returns a result.

# Case Study: State Isolation

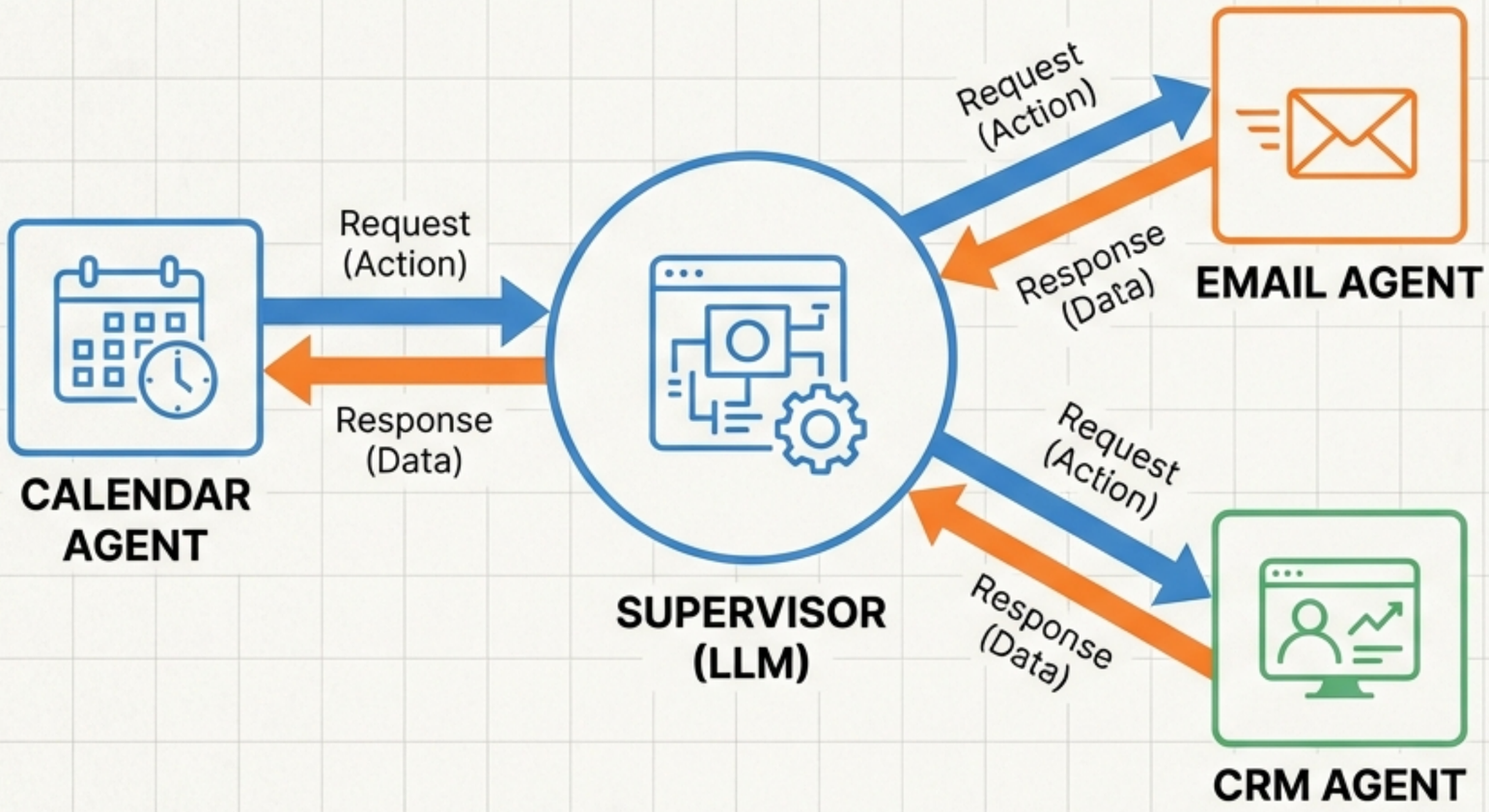
Data transformation in a Translation Workflow.




No Variable Pollution

# PATTERN A: THE SUPERVISOR

The ‘Hub-and-Spoke’ model for breadth and orchestration.




ROLE: Orchestration & Routing.

STATE: Supervisor maintains global context. Workers are often stateless tools.

BEST FOR: Personal Assistants, diverse toolsets, distinct domains.

Inter Tight Bold, Slate · Inte, Slate Grey

# The Three-Layer Abstraction

Protecting the orchestrator from low-level complexity.

**Supervisor Layer**

Strategic Routing

"Schedule a meeting for next Tuesday."

**Sub-Agent Layer**

Translation (NL -> Structured)

```
Parses "Next Tuesday" -> "2024-01-16T14:00:00"
```

**Tool / API Layer**

Rigid Execution

Google Calendar API

SendGrid API

```
POST /events { "start": "..." }
```

Abstraction Level Increases

Mono JetBrains Mono

Technical JetBrains Mono

Inter Tight Bold, Slate · Inte, Slate Grey

# PATTERN B: HANDOFFS (THE RELAY)

State-driven flow for depth and sequential constraints.


**STATE OBJECT**

```
{
  "active_agent": "Refund",
  "warranty_id": "VALID",
  "refund_status": "PENDING"
}
```

**MECHANISM:**
Agents use tools or `Command.PARENT` to switch the active controller.

**BEST FOR:**
Customer support, troubleshooting, strict sequential logic (Step A must unlock Step B).

Mono: JetBrains Mono

Inter Tight Bold, Slate Inte, Slate Grey

# Technical Implementation: Handoff Context

The critical message-pairing requirement.

## DO THIS: Message Pairing (Required)

1. `AIMessage(...)` The Tool Call (The Trigger)

```
  content="",
  tool_calls=[{"name": "transfer_to_sales"}],
}
```

2. `ToolMessage(...)` The Acknowledgement (The Artificial Response)

```
  tool_call_id=...,
  content="Successfully transferred to sales."
}
```

**CRITICAL:** You must manually insert the ToolMessage so the receiving agent sees a valid history. 'Dangling' tool calls cause hallucinations.

## NOT THAT: Dangling Tool Call

```
AIMessage(...)
  content="",
  tool_calls=[{"name":
    "transfer_to_sales"}],
}
```

Dangling, No Response

**Causes Hallucinations/Errors**

Inter JetBrains Mono

Technical JetBrains Mono

Inter Tight Bold, Slate Inte, Slate Grey

# Decision Matrix

Choosing the right architecture for your system.

## Supervisor (Hub)

- Distinct, unrelated domains (Email vs. CRM)
- Centralized control is required
- Parallel execution (calling multiple agents at once)
- Sub-agents do NOT talk to user directly

## Handoffs (Relay)

- Deep, state-driven flows (Customer Support)
- Strict sequential steps (A unlocks B)
- Agents need multi-turn conversation with user
- Dynamic behavior changes based on state

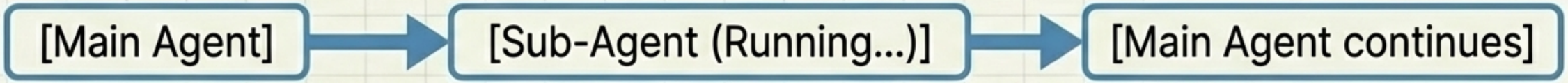
Technical JetBrains Mono

Inter Tight Bold, Slate Inte, Slate Grey

# Execution Strategies

Managing time: Synchronous vs. Asynchronous flows.

## Synchronous (Blocking)




Main agent waits. Good for dependencies.

## Asynchronous (Background)


Main agent continues. Good for long tasks.

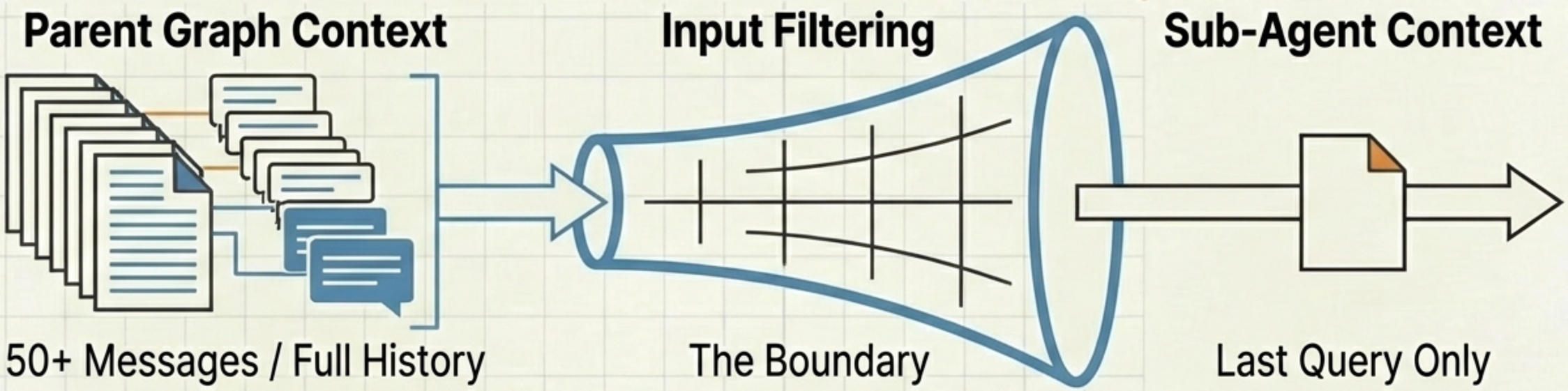
Pattern: 'Three-Tool Pattern' (Start Job -> Check Status -> Get Result).

Technical JetBrains Mono

Inter Tight Bold, Slate Inte, Slate Grey

# Context Engineering

Optimizing token usage and model performance.

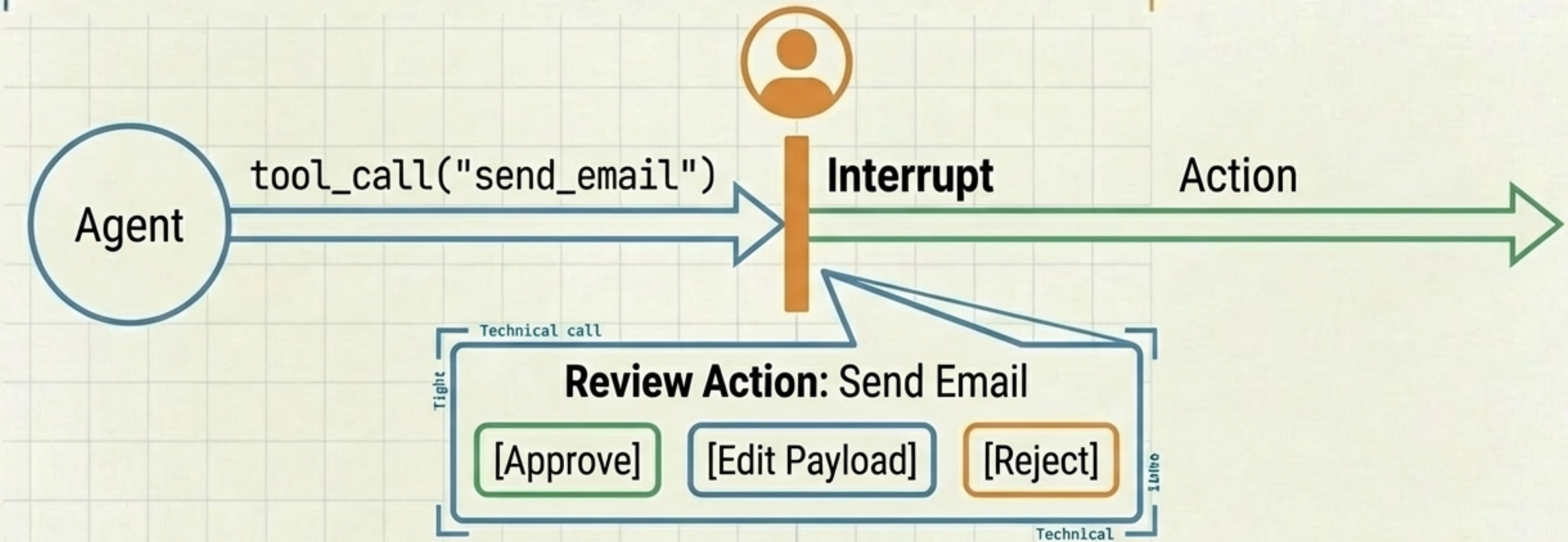



- **Input Filtering:** Don't leak 50 messages of history if the sub-agent only needs the last question.
- **Output Formatting:** Strip 'reasoning' traces. Return only the final answer to the parent.
- **Summarization:** Compress history at graph boundaries.

Inter Tight Bold, Slate Inte, Slate Grey

# Human-in-the-Loop (HIL)

Safety mechanisms for autonomous subgraphs.



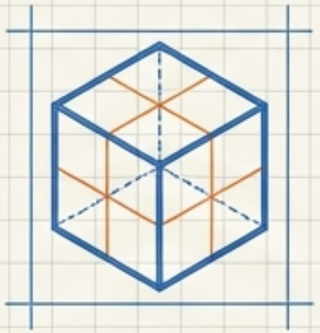

A checkpointer on the PARENT graph automatically enables HIL for child subgraphs.

# Architecture Summary

Key patterns for scalable LangGraph systems.

Technical call

Inter Tight Bold, Slate Grey



## Modularity

Break monoliths into **Subgraphs** to isolate failure and simplify logic.


Hub Relay

## Patterns

Use **Supervisors** for tool breadth. Use **Handoffs** for conversational depth.

## State

**Isolated State** for distinct tasks. **Shared State** for tight collaboration.

## Context

**Filter** inputs and pair messages during handoffs to prevent hallucinations.

Build for Observability. Trace agents individually.