AI IN FINANCE CONFERENCE

Beyond Chatbots:

Parallelizing Agents for Speed and Scale

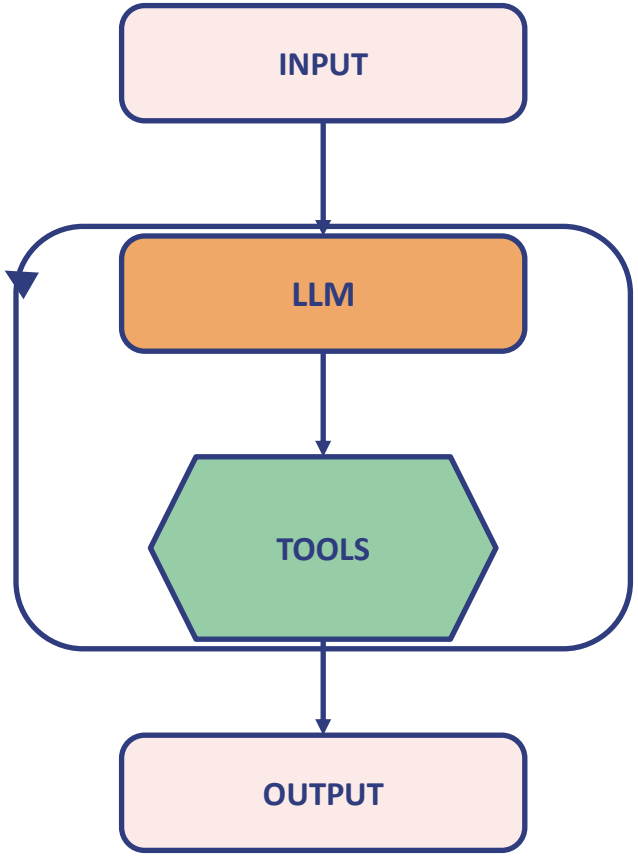
Harkirat | VP, Quantitative Research — Wholesale Credit Risk, JPMorgan Chase

THE STARTING POINT

The Single-Agent Model

A single agent is one LLM-driven loop that perceives, decides, and acts on its own.

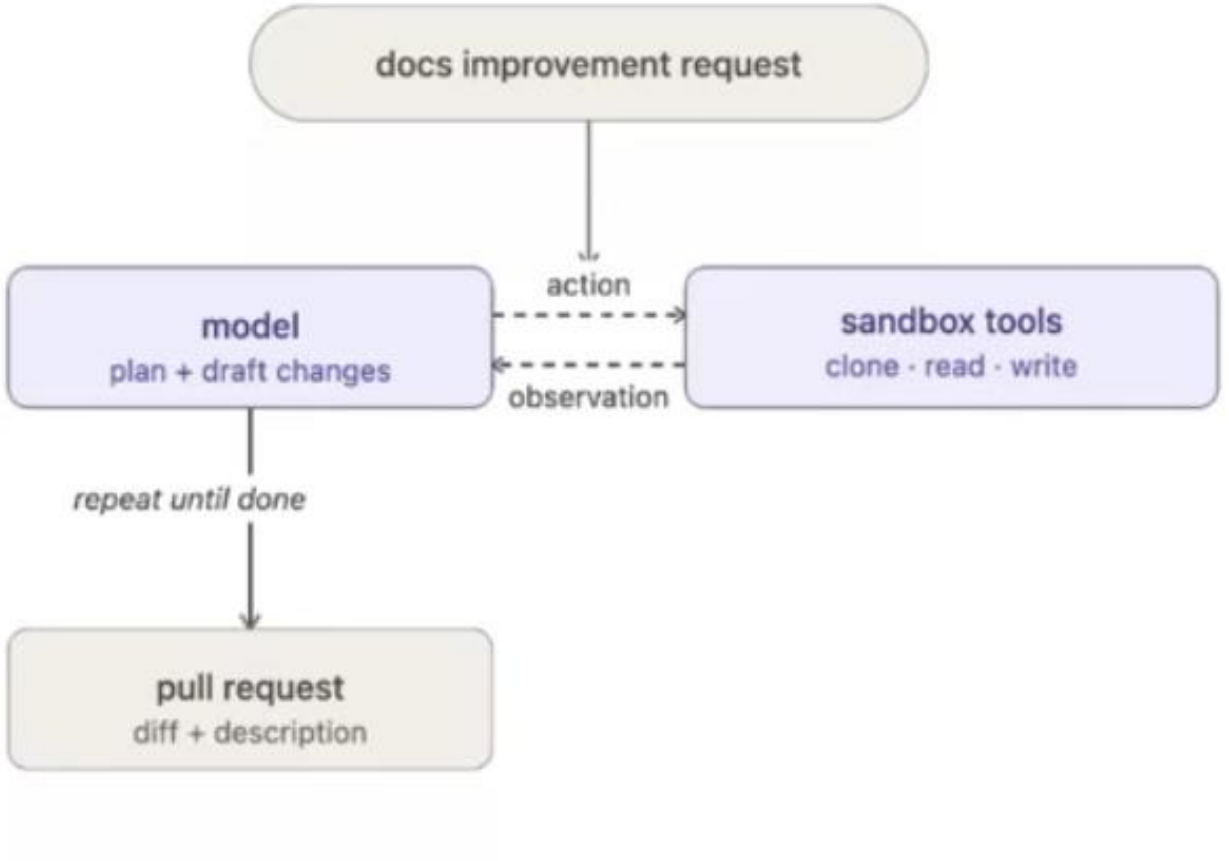
- A goal/task it's given upfront
- Access to tools (search, code execution, APIs, file access, etc.)
- A loop: observe current state → reason/plan next step → take an action (call a tool) → observe the result → repeat until the task is done or it decides to stop
- Memory/context of what it's already tried, usually just the running conversation/scratchpad — no separate memory store shared with other agents



THE STARTING POINT

A Single Agent in Action

A docs improvement agent — one loop, one context, working until the job is done



THE STARTING POINT

Single-Agent Use Cases

1

Querying Data

Takes a natural-language question and turns it into a query, runs it, and returns the result. E.g. “what's our loan exposure in the energy sector?”

2

Generating Metrics

Computes and tracks the numbers that matter (ratios, KPIs, deltas) by applying consistent business logic to raw data.

3

Reporting

Turns computed metrics into a narrative summary a person can actually read. WBR/MBR doc that covers business performance numbers.

4

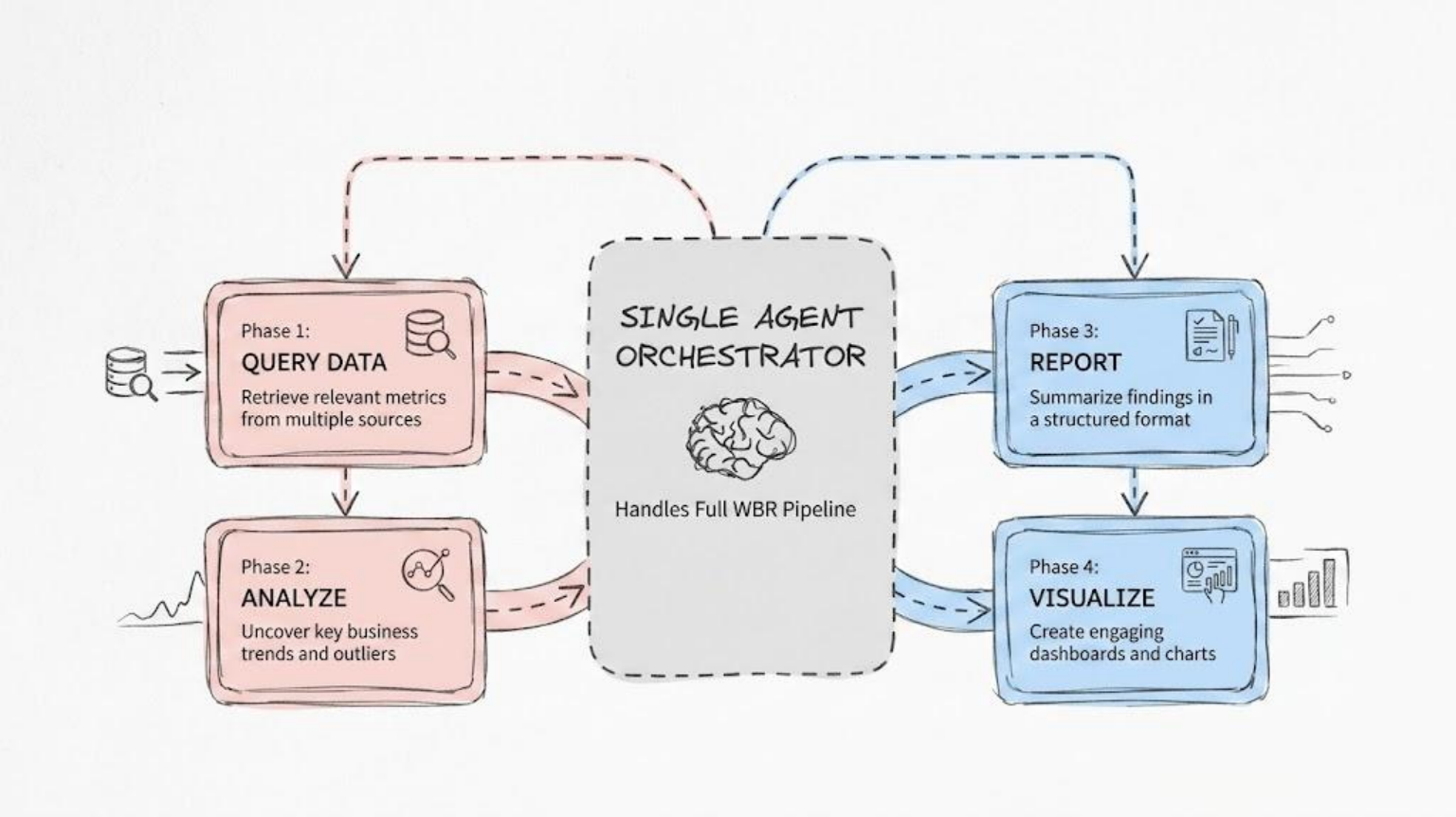
Visualizing

Takes a metric or dataset and produces the right chart or dashboard view for it. E.g. turning a quarter's exposure numbers into a trend chart for a risk committee deck.

THE STARTING POINT

Single Agent Tasks

One agent, handling the full Weekly Business Review pipeline

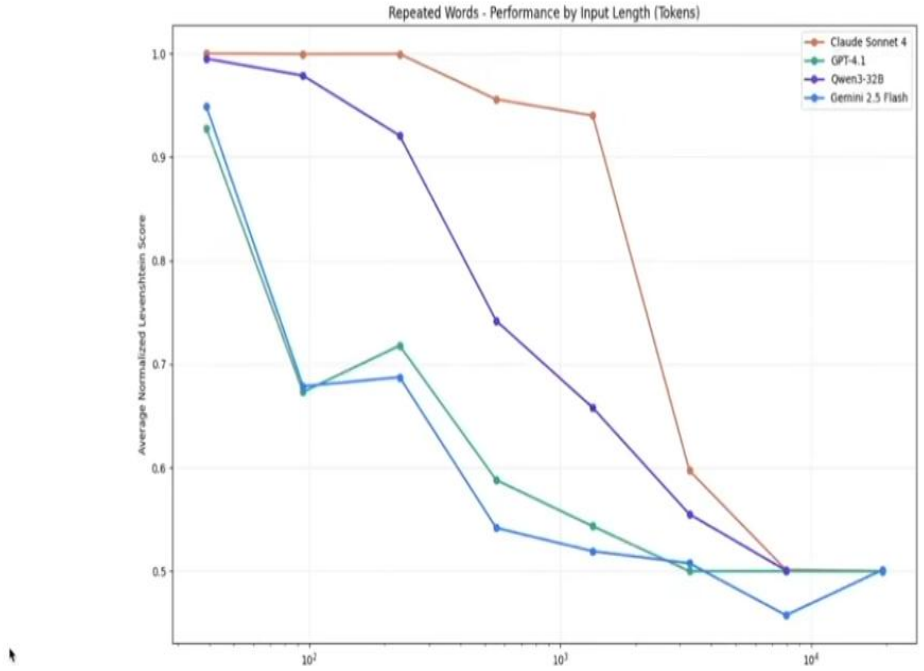


WHERE IT BREAKS

Limitations of Single Agent

- **Too many tools, poor tool choice** — the agent has so many tools available it struggles to pick the right one
- **Context overload** — as the task grows, a single agent can't keep track of everything in its own context
- **One agent, many specialties needed** — real systems need distinct expertise (planner, researcher, math expert) that one generalist agent can't cover well

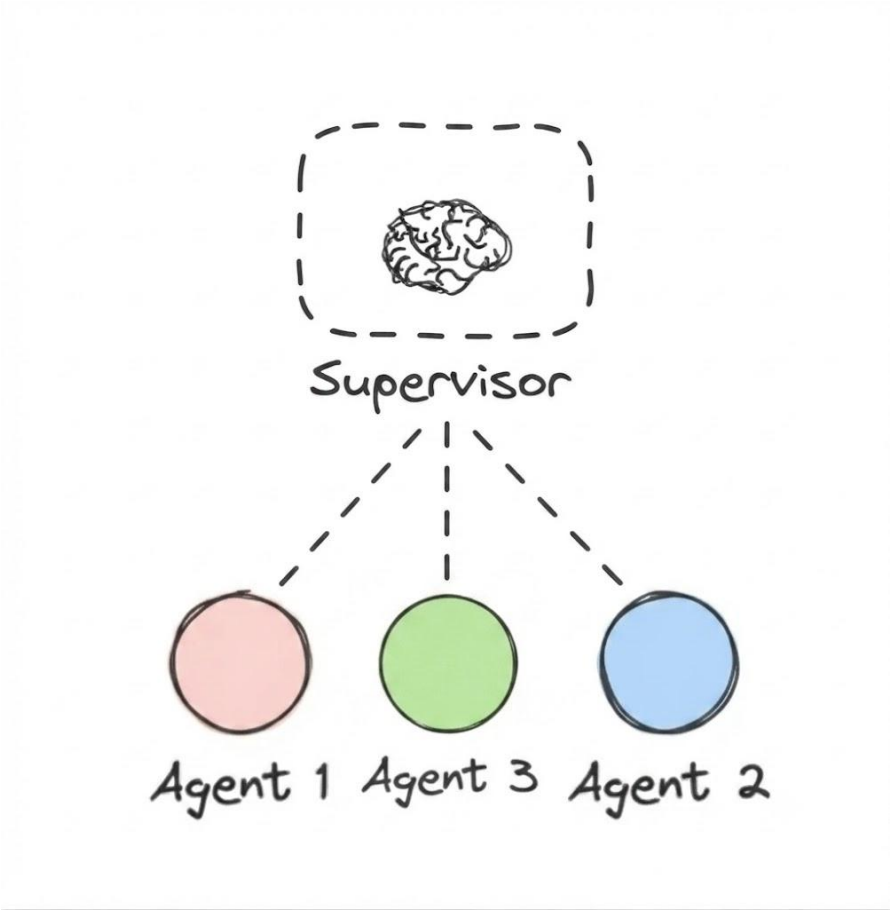
Finite context windows and performance drops



THE SHIFT

What Is a Multi-Agent System?

- **Multiple agents** — each specialized role (researcher, coder, reviewer, etc.)
- **Some coordination mechanism** — decides who acts when, how info passes (orchestrator-worker, sequential, etc.)
- **Separate contexts** — each agent has its own scratchpad, not shared; info must be explicitly passed.



WHY IT'S WORTH THE COMPLEXITY

Benefits of Multi-Agent Systems

1

Speed

Independent subtasks run in parallel instead of sequentially

2

Resilience

One agent's failure doesn't take down the whole workflow

3

Quality

Narrow scope per agent means sharper, more reliable outputs

4

Scalability

Add or swap specialized agents without rebuilding the system

WHERE THIS SHOWS UP TODAY

Multi-Agent Use Cases

1

Autonomous Research

Parallel subagents explore different angles of a query, then synthesize

2

Customer Support Automation

Specialized agents triage, resolve, and escalate in tandem

3

Software Development

Separate agents draft code, run tests, and perform code review

4

Data Ingestion Automation

Agents interpret requirements, ingest sources, validate, and build dashboards

PATTERN 1 OF 4

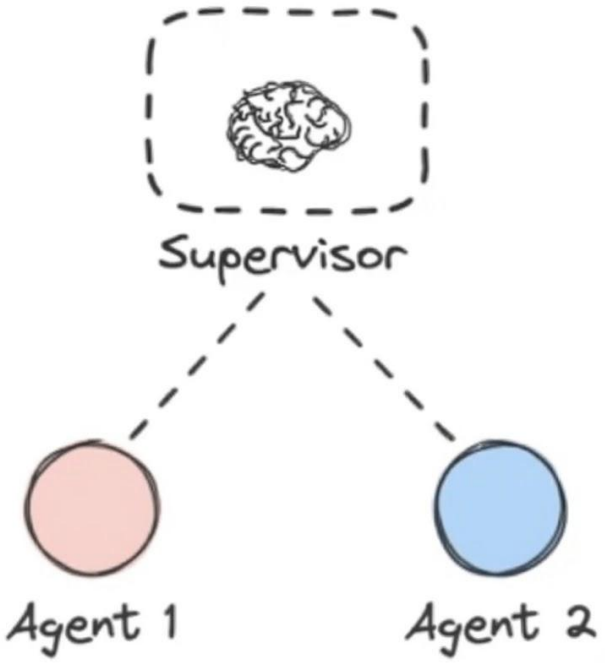
MOST COMMON

Orchestrator-Worker

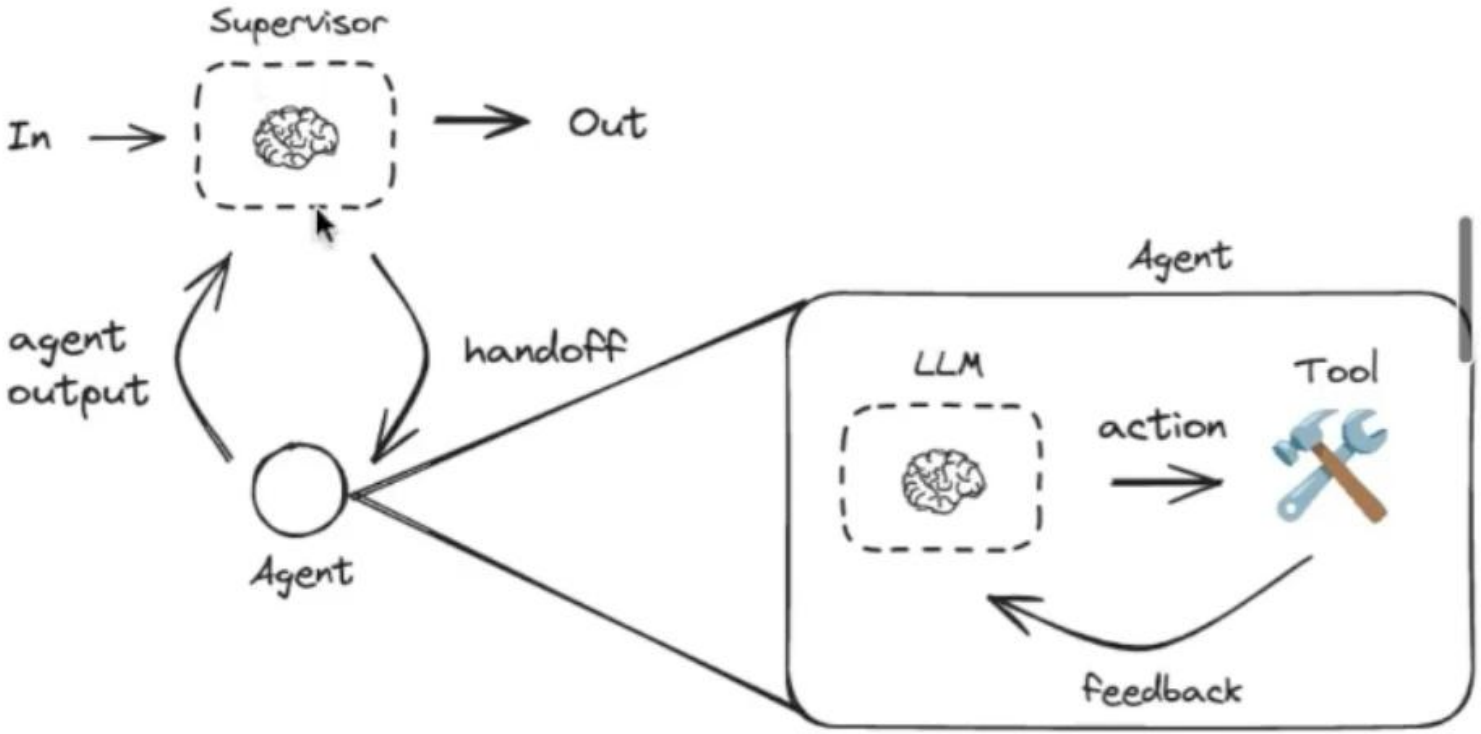
- **Orchestrator decomposes the task** — breaks the request into subtasks and assigns one to each worker
- **Worker agents execute in parallel** — each one owns a narrow piece of the job and runs parallelly and then come back together to synthesize their findings.
- **Results get aggregated into one response** — outputs are merged before anything goes back to the user

USE CASE

WBR example — one orchestrator delegates metrics-pulling, anomaly detection, trend synthesis, and narrative writing to separate agents



Orchestrator-Worker Handoff



PATTERN 2 OF 4

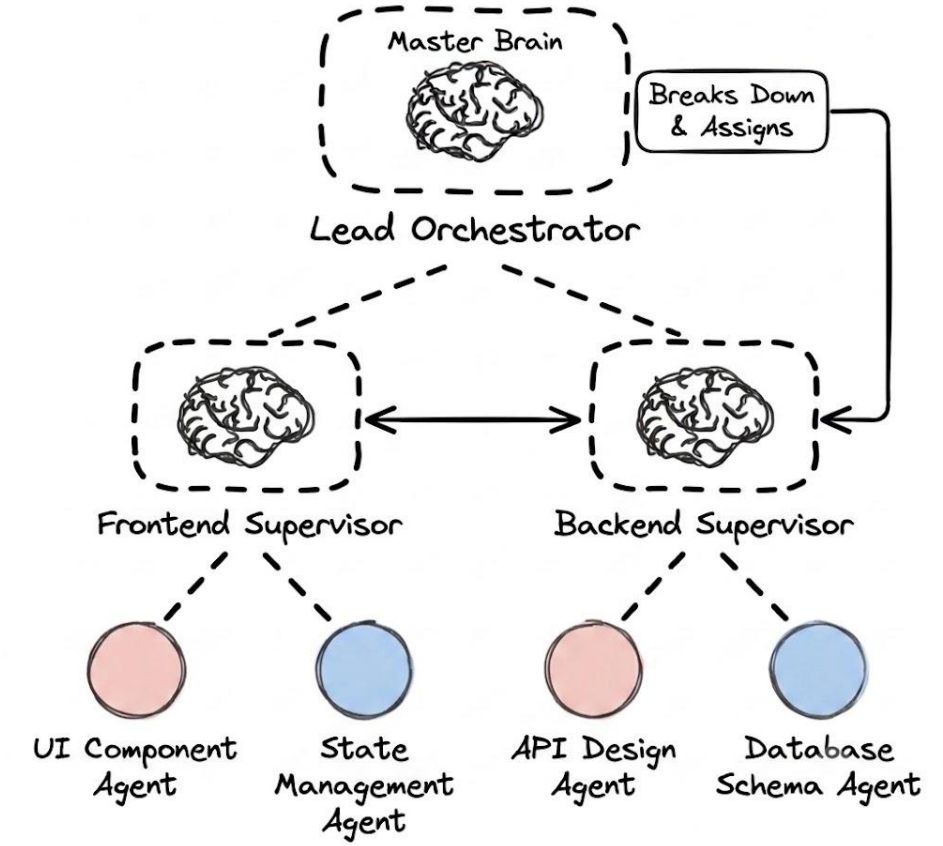
TIERED DELEGATION

Hierarchical

- **Supervisors of supervisors** — the top orchestrator delegates to mid-level supervisors, who each manage their own team of agents
- **Work splits by domain** — each supervisor owns one area and coordinates only the agents under it, keeping scope tight at every level
- **Scales to complex builds** — add a new supervisor and its agents without touching the rest of the tree

USE CASE

Software development pipelines — a Lead Orchestrator breaks down a feature request and assigns sub-tasks. A Frontend Supervisor coordinates UI Component and State Management agents, while a Backend Supervisor coordinates API Design and Database Schema agents.



PATTERN 3 OF 4

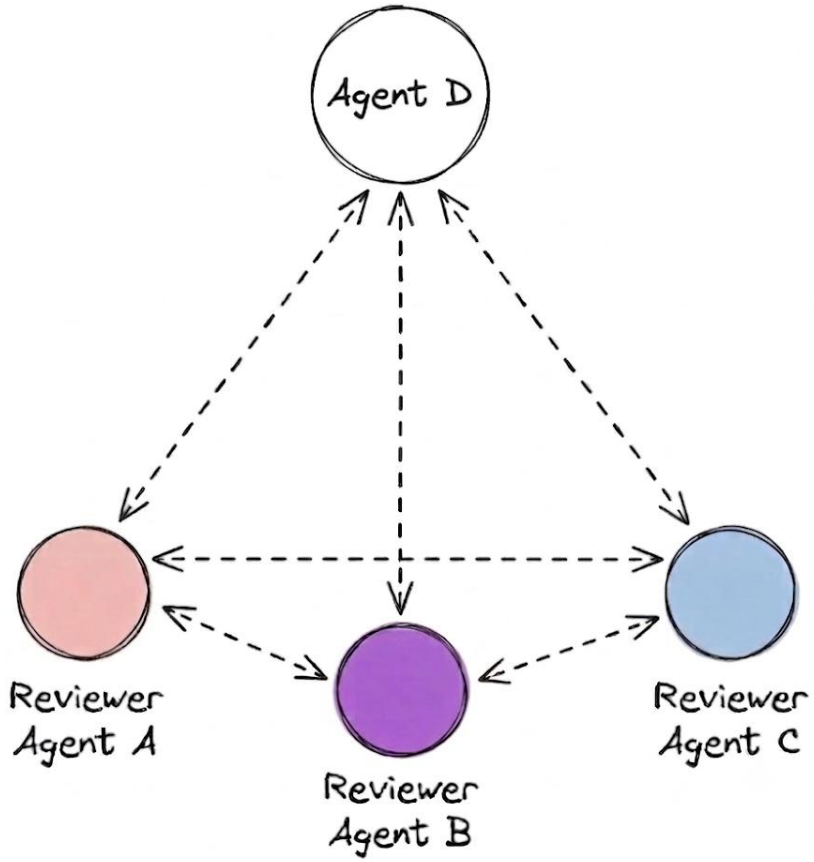
Peer-to-Peer

HARDER TO DEBUG

- **No central coordinator** — agents communicate directly with each other
- **Flexible but hard to trace** — decentralized control makes debugging harder
- **Less common in production** — useful for exploratory, negotiation-style tasks

USE CASE

A code review system — a Correctness, Security, and Readability reviewer each assess the same pull request, debate each other's findings, and converge on a merge decision.



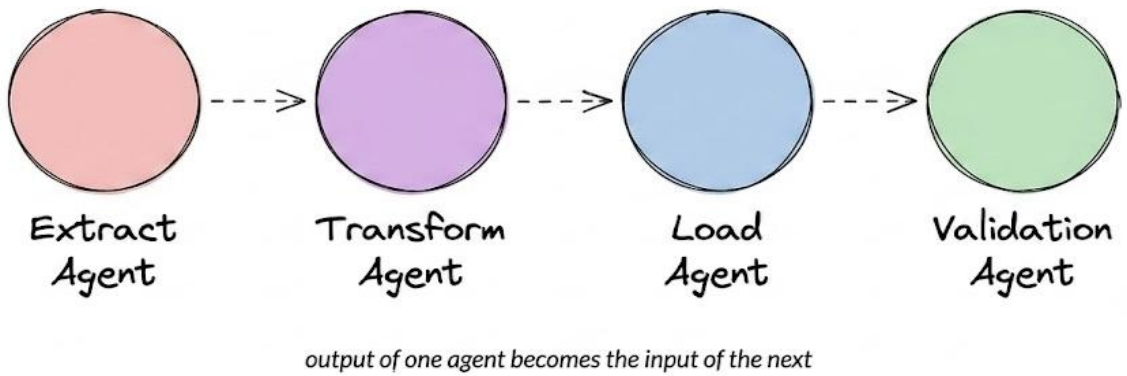
PATTERN 4 OF 4

FIXED ORDER

Pipeline / Sequential

- **Linear handoff** — Agent A finishes, Agent B starts on A's output, Agent C on B's. No branching, no looping back.
- **No orchestrator overhead** — no coordinator deciding who goes next; the chain is the coordination
- **Best for fixed workflows** — document preprocessing, ETL, anything with a stable, repeatable order
- **No parallelism, no recovery** — one slow or failed stage stalls the whole line

USE CASE *Data pipeline ETL — extract agent → transform agent → load agent → validation agent, each dependent on the last*



When the Builder Validates Itself

- **The builder and the validator share one context** — the same agent that made the decisions also checks them
- **Tests get shaped by the code, not the intent** — validation confirms what was built, not what was asked for
- **Blind spots are inherited, not caught** — the check carries the same assumptions that produced the bug
- **You get confirmation, not verification** — passing tests feel like safety but only prove internal consistency
- **The system drifts over time** — each cycle validates against the last flawed step, compounding quietly

THE FIX

Validation Loop

1

Checks against the goal, not the output

Success criteria come from the original ask, never from the builder's work.

2

Separate agents, sub-agents

A fresh conversation with no memory of how the builder made its decisions.

3

Separate instructions and tools

Its own system prompt, tools, even model. The builder is told “build X”; the validator, “you're a critic — here's the goal, here's the output, find where they diverge.”

Mistakes People Make

01

Unclear ownership

Not scoping which agent is responsible for what, up front

02

No failure plan

Not designing for what happens when one agent fails

03

Over-engineering

Building multiple agents for tasks one or two agents could handle

The bottleneck was never intelligence. It was attention.

Multi-agent systems are how you get it back.

Thank you