INF 385T · FALL 2026

TEXAS
The University of Texas at Austin

INF385T: Special Topics in Information Science: Language Models and Agentic Workflows

Week 9: Multi-Agent Systems

Dr. Abhijit Mishra

Announcements

- **Group Project 1** – Project Report (2-3 pages)
  - Deadline : Nov 5
- **Group Project 1** – Presentation (Nov 5)
- **Group Project 2** – Proposal submission
  - Deadline: 10/30
- **Industry Keynote:** Nov 19
- **Group Project 2** – Final Presentation (Dec 3)
- **Group Project 2** – Final Report (Dec 10)

Week 8 Recap

- **MCP Fundamentals:** Learned how MCP standardizes commu nication between AI agents and external tools.
- **Tool Schema & Handlers:** Defined tools and implemented thei r execution logic.
- **Prompt Schema & Handlers**: Created reusable prompts with d ynamic arguments.
- **Tool & Prompt Discovery:** Explored listing available tools and prompts through MCP.

Recap: Agent Tool Integration issues

Growing number of AI models

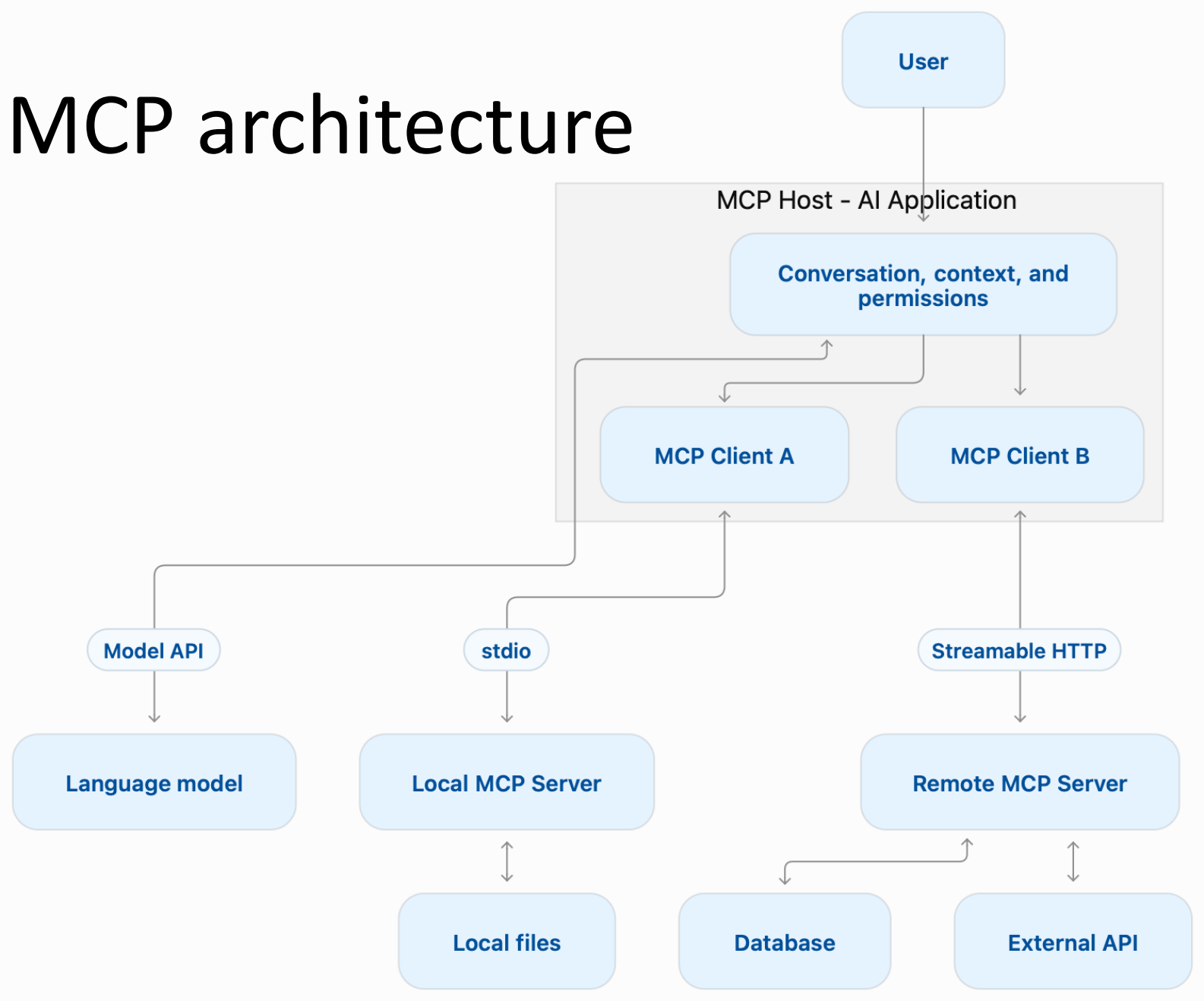
Increasing application complexity

Rise in development time and cost

Recap: Model Context Protocol (MCP)

- An Open Source Protocol that enables seamless integration between AI agents with tool and data sources
- **Analogy:** Rest API for web development

Recap: MCP architecture



Recap: building an MCP server

- 6 different components

Component	Why is it needed?
Requires Tool Definitions	So MCP clients know which tools are available.
Defining Tool Schema	Specifies the tool's name, purpose, and expected inputs.
Defining Tool Handlers	Implements the actual logic when a tool is called.
Requires Prompt Definitions	So MCP clients can discover and reuse predefined prompts.
Defining Prompt Template	Specifies the prompt's structure and required arguments.
Implementing Prompt Handler	Generates the actual prompt using the supplied arguments.

Week 9: Roadmap

- **Why Multiple Agents?**
  - Specialization, parallelism, robustness, and tradeoffs
- **Multi-Agent Architectures**
  - Centralized, decentralized, hierarchical, and swarm
- **Coordination Mechanisms**
  - Shared state, message passing, delegation, and consensus
- **Communication and Collaboration**
  - Protocols, roles, debate, and reflection
- **Challenges and Evaluation**
  - Coordination costs, scalability, safety, and reliability

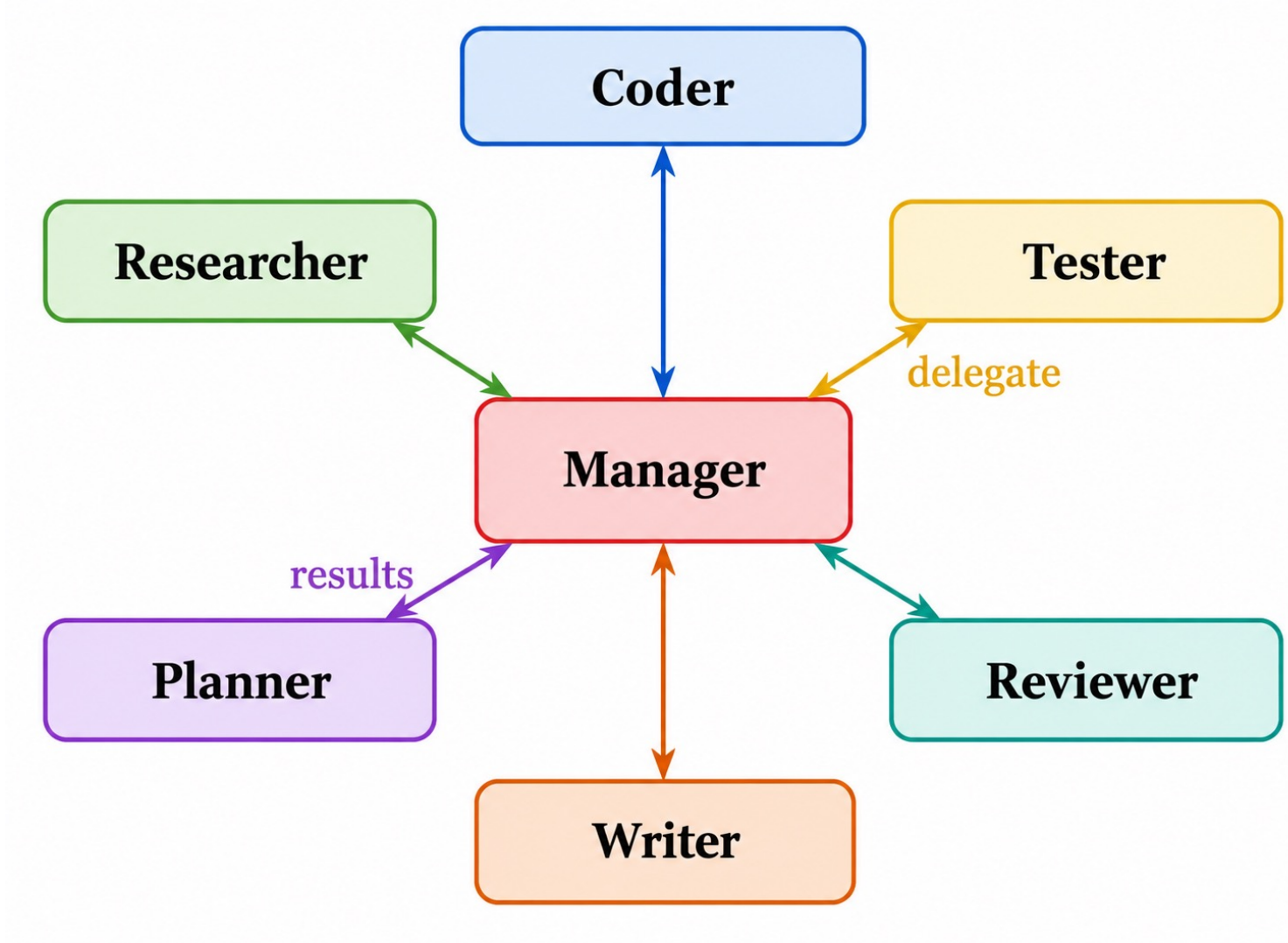
Why have multiagent systems?

Example: A software development process includes

- Coding
- Testing
- Deployment
- Security reviewing

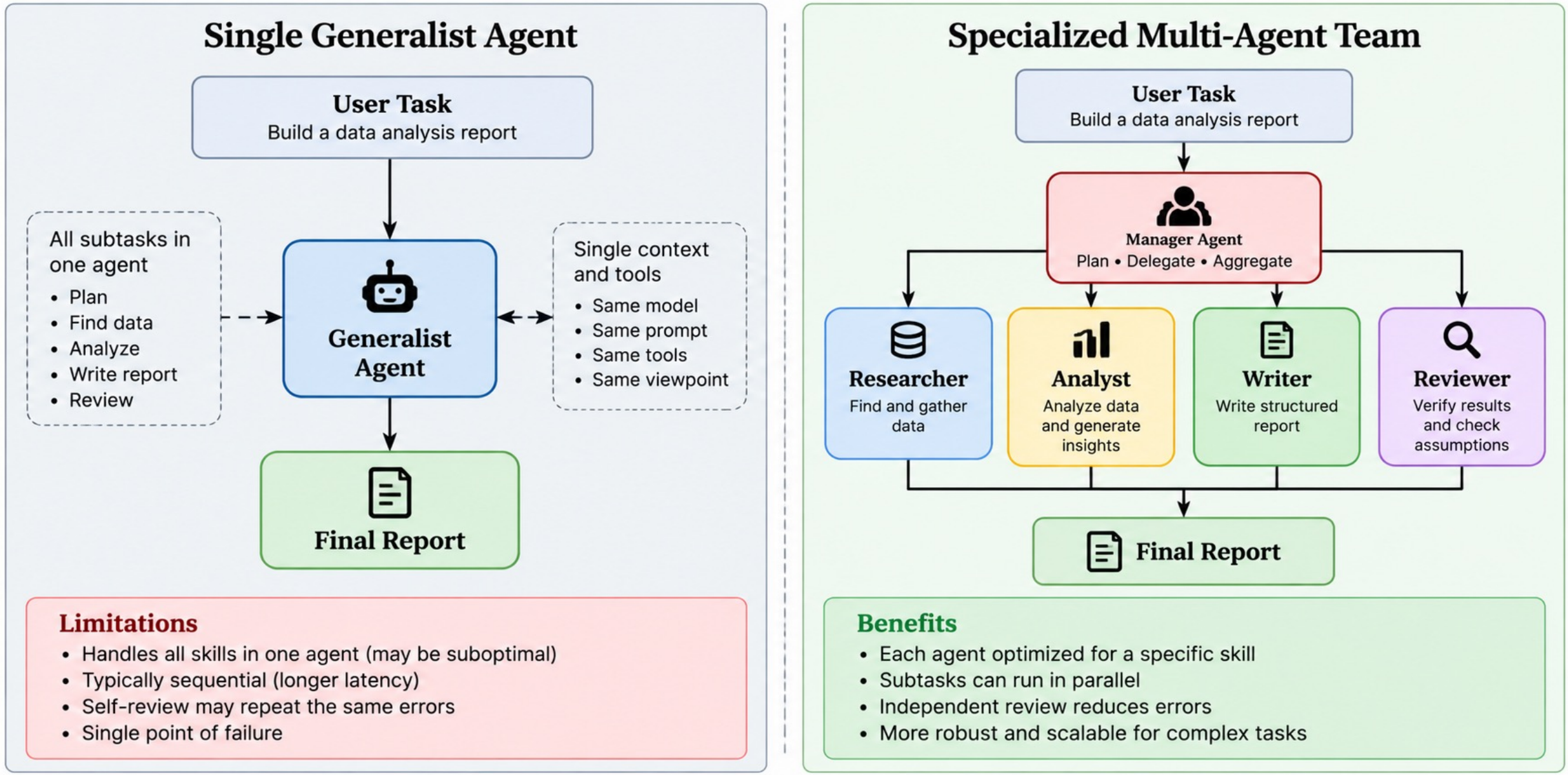
Can a single person do it?

Can a single agent do it?


Coder
Researcher
Tester
delegate
Manager
results
Planner
Reviewer
Writer

Why multiple agents (or a team)?

- **Specialization:** coder, tester, and reviewer use different tools and objectives.
- **Parallelism:** independent literature searches can run concurrently.
- **Robustness:** a second reviewer may catch an unsupported claim.
- **Emergence:** debate can produce a conclusion neither initial response offered.


Single Generalist Agent
User Task
Build a data analysis report
All subtasks in one agent
Plan
Find data
Analyze
Write report
Review
Generalist Agent
Single context and tools
Same model
Same prompt
Same tools
Same viewpoint
Final Report
Limitations
Handles all skills in one agent (may be suboptimal)
Typically sequential (longer latency)
Self-review may repeat the same errors
Single point of failure
Specialized Multi-Agent Team
User Task
Build a data analysis report
Manager Agent
Plan • Delegate • Aggregate
Researcher
Find and gather data
Analyst
Analyze data and generate insights
Writer
Write structured report
Reviewer
Verify results and check assumptions
Final Report
Benefits
Each agent optimized for a specific skill
Subtasks can run in parallel
Independent review reduces errors
More robust and scalable for complex tasks

When is a team worth it?

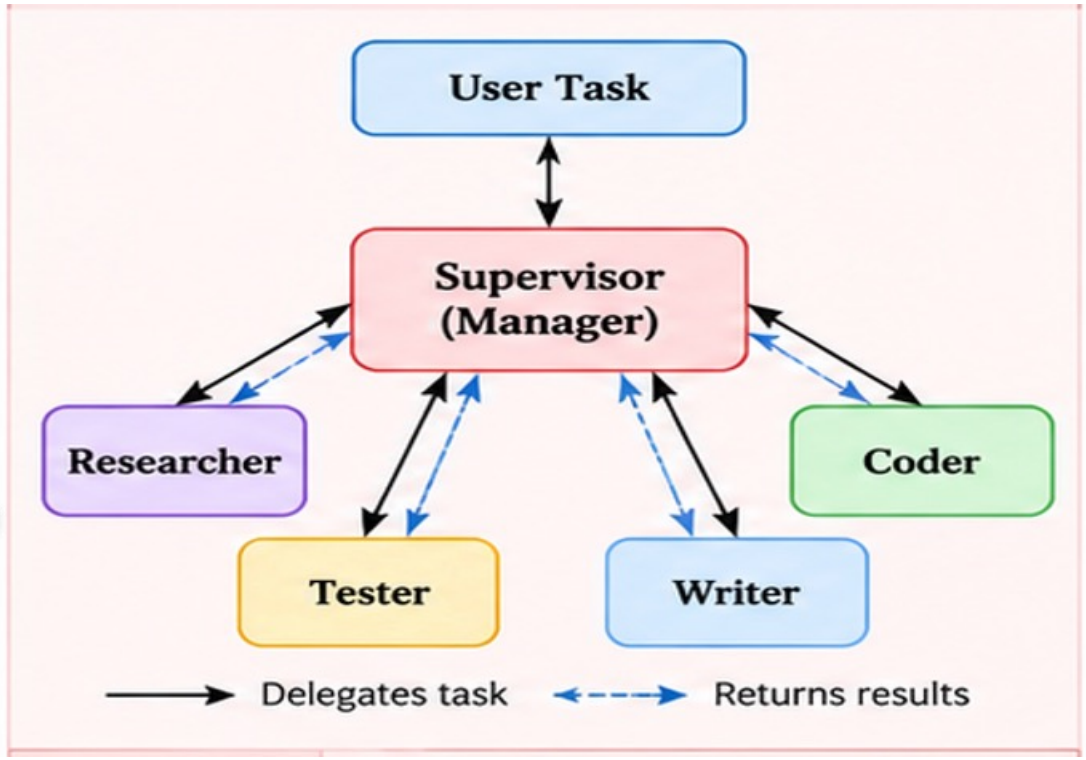
- **Good fit:** separable subtasks, distinct tools, isolated context, or high-value verification.
- **Poor fit:** one lookup, tightly sequential work, or repeated shared context
- **Costs:** tokens, communication loss, latency, and larger failure surface.
- **Example:** three identical agents reading one source can repeat the same error.
- **Decision:** compare against a strong single-agent baseline.

Architecture is a control decision

- **Topology determines who delegates, speaks, updates state, and resolves failure.**
- **Centralized:** supervisor owns global task and aggregation.
- **Decentralized:** peers negotiate without one global coordinator.
- **Hierarchical:** domain managers supervise local teams.
- **Swarm:** local routing through handoffs, without global state.

Centralized

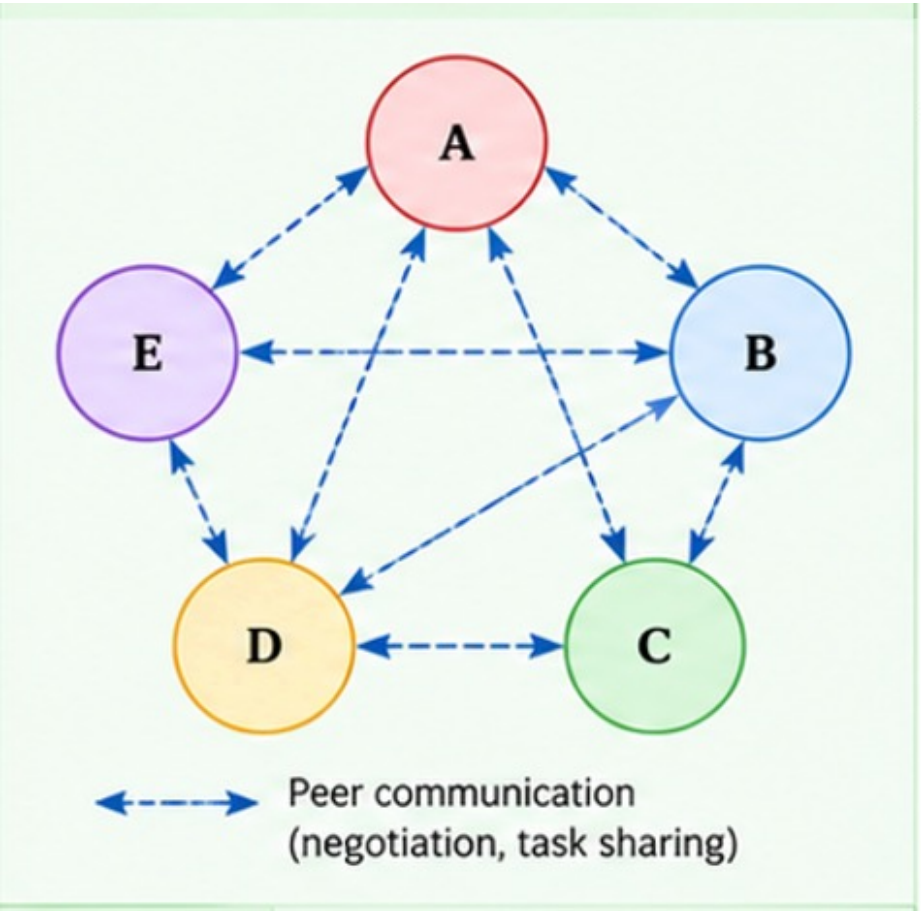
Supervisor owns global task and aggregation.



Delegates	Supervisor to workers
Speaks	Mainly via supervisor
State	Centralized (supervisor)
Failure	Supervisor detects and re-routes
Example	Research report with planning, coding, testing, and writing

Decentralized

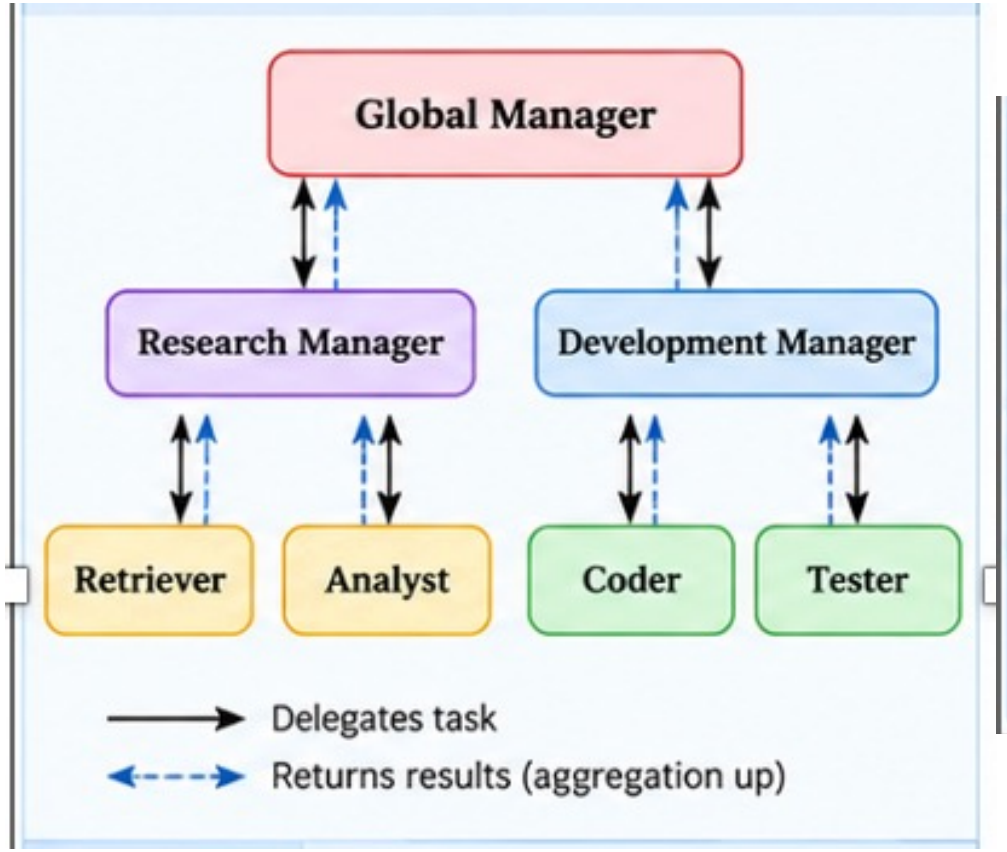
Peers negotiate without one global coordinator.



Delegates	Peers to peers
Speaks	Direct peer-to-peer
State	Distributed (shared/replicated)
Failure	Peers detect and recover locally (e.g., re-negotiate)
Example	Multi-agent research where agents bid for subtasks

Hierarchical

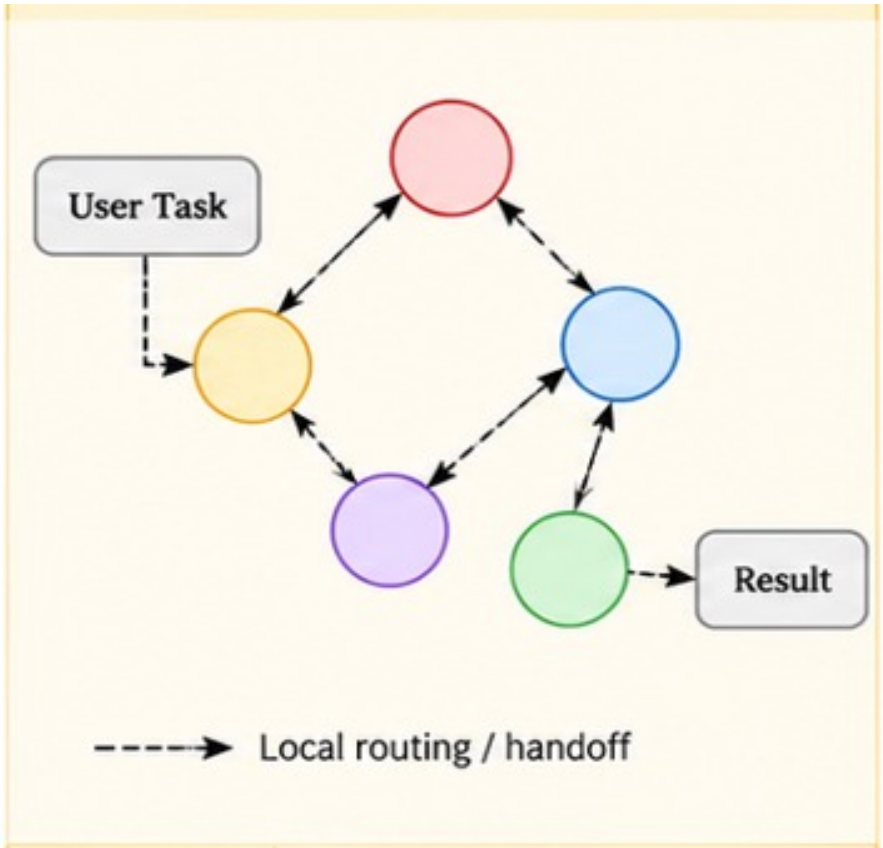
Domain managers supervise local teams.



Delegates	Top-down (multi-level)
Speaks	Within teams and to parent
State	Hierarchical (local at each level + summaries up)
Failure	Local manager handles and escalates if needed
Example	Enterprise workflow with domain-specific teams

Swarm

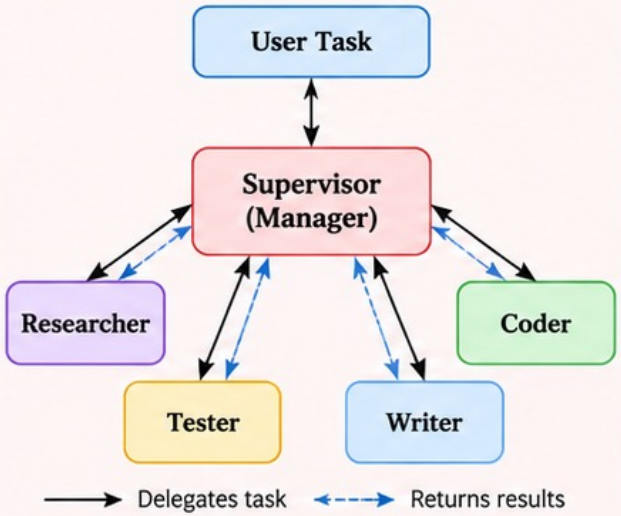
Local routing through handoffs,
without global state.



Delegates	Dynamically via handoffs
Speaks	Direct with next agent
State	Local (no global state)
Failure	Next agent can take over or reroute
Example	Open-ended task solved by a chain of specialized agents

Centralized

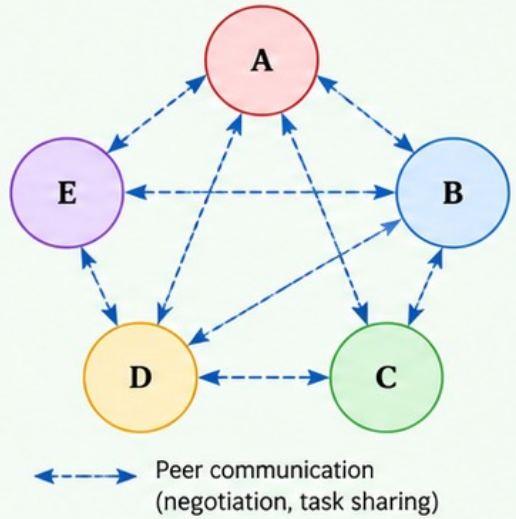
Supervisor owns global task and aggregation.



Delegates	Supervisor to workers
Speaks	Mainly via supervisor
State	Centralized (supervisor)
Failure	Supervisor detects and re-routes
Example	Research report with planning, coding, testing, and writing

Decentralized

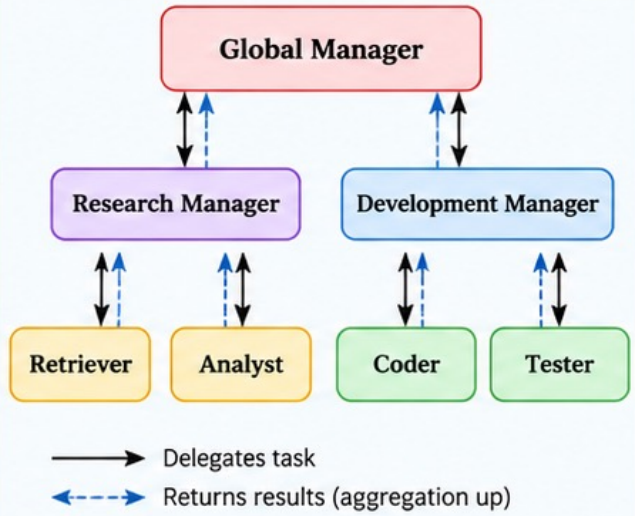
Peers negotiate without one global coordinator.



Delegates	Peers to peers
Speaks	Direct peer-to-peer
State	Distributed (shared/replicated)
Failure	Peers detect and recover locally (e.g., re-negotiate)
Example	Multi-agent research where agents bid for subtasks

Hierarchical

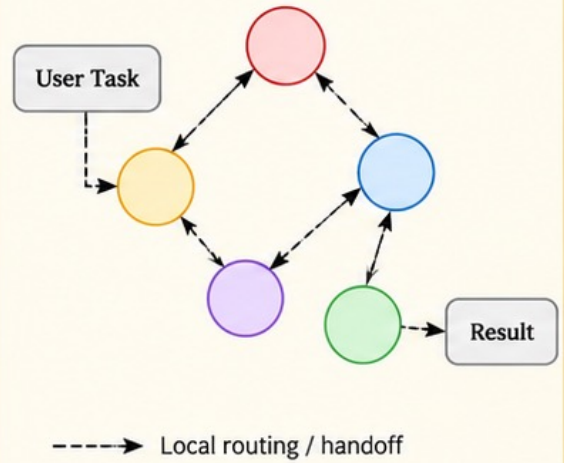
Domain managers supervise local teams.



Delegates	Top-down (multi-level)
Speaks	Within teams and to parent
State	Hierarchical (local at each level + summaries up)
Failure	Local manager handles and escalates if needed
Example	Enterprise workflow with domain-specific teams

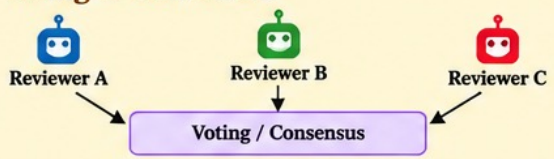
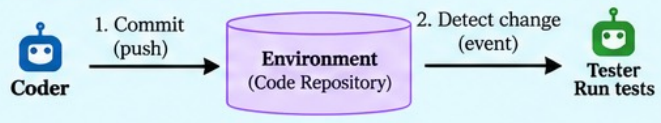
Swarm

Local routing through handoffs, without global state.



Delegates	Dynamically via handoffs
Speaks	Direct with next agent
State	Local (no global state)
Failure	Next agent can take over or reroute
Example	Open-ended task solved by a chain of specialized agents

Coordination Mechanisms

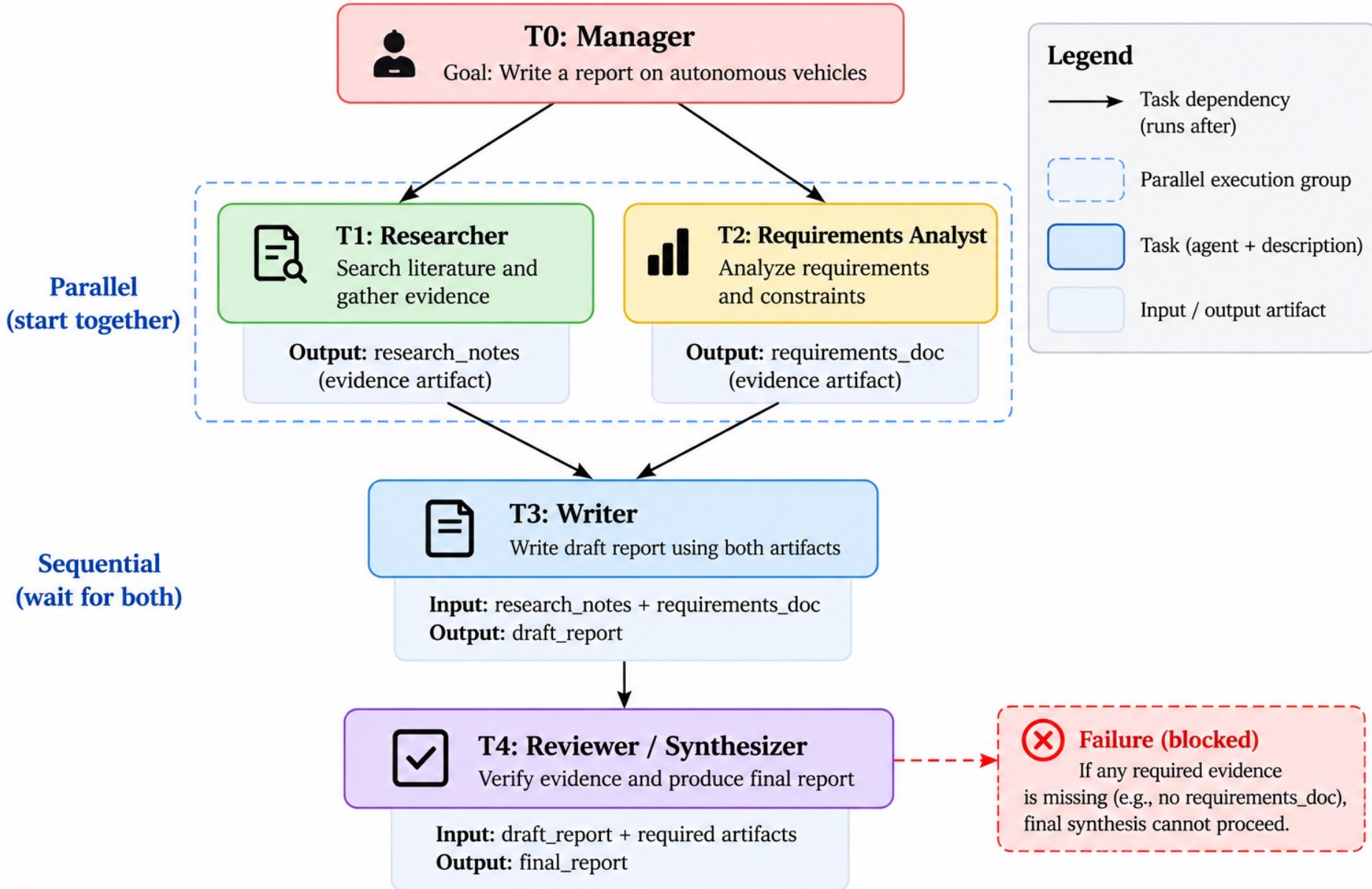
Mechanism	How agents coordinate	Concrete example
1. Shared State (Blackboard) 	Read and update a common workspace.	Researcher and analyst write findings to a shared database.
2. Message Passing 	Exchange explicit, structured messages.	Tester sends a bug report to the coder.
3. Planning & Decomposition 	Divide a goal into dependent tasks (DAG).	Research and analysis run in parallel; writing waits for both.
4. Voting & Consensus 	Aggregate competing judgments.	Three reviewers vote on whether code is safe.
5. Market-Based Coordination 	Bid for tasks based on capability and cost.	Agents bid to complete a data-analysis task.
6. Stigmergy 	Coordinate indirectly through environmental changes.	Coder commits code; tester detects the commit and runs tests.

Which architectures use them

Architecture	Common coordination mechanisms
Centralized	Planning, delegation, message passing, shared state
Decentralized	Peer messaging, consensus, auctions, stigmergy
Hierarchical	Multi-level planning, message passing, shared state
Swarm	Local handoffs, message passing, stigmergy

Decomposition and task dependencies

- Manager decomposes a goal into a DAG of tasks.
  - **Ready task** = pending and every dependency completed.
  - **Parallel:** researcher and requirements analyst start together.
  - **Sequential:** writer waits for both evidence artifacts.
- Failure: missing required evidence blocks final synthesis.


T0: Manager
Goal: Write a report on autonomous vehicles
Legend
Task dependency (runs after)
Parallel execution group
Task (agent + description)
Input / output artifact
Parallel (start together)
T1: Researcher
Search literature and gather evidence
Output: research_notes (evidence artifact)
T2: Requirements Analyst
Analyze requirements and constraints
Output: requirements_doc (evidence artifact)
Sequential (wait for both)
T3: Writer
Write draft report using both artifacts
Input: research_notes + requirements_doc
Output: draft_report
T4: Reviewer / Synthesizer
Verify evidence and produce final report
Input: draft_report + required artifacts
Output: final_report
Failure (blocked)
If any required evidence is missing (e.g., no requirements_doc), final synthesis cannot proceed.

Consensus and voting

Consensus and voting matter when multiple agents produce competing answers and the system must reach a collective decision.

Majority: most common answer wins.

Weighted: reliability weights change the vote.

Debate: competing positions go to a judge.

Delphi: agents revise after seeing others' reasoning.

Why is consensus important?

- **Resolve disagreement:** Agents may reach different conclusions from the same task.
- **Improve reliability:** Aggregating independent judgments can reduce individual errors.
- **Avoid arbitrary authority:** Decisions need not depend on one supervisor's opinion.

Paradigm	Role of consensus
Decentralized (peer-to-peer)	**Central mechanism:** peers negotiate or vote without a supervisor.
Centralized (supervisor)	Optional: supervisor aggregates votes or adjudicates disagreements.
Hierarchical	Local teams reach consensus; unresolved conflicts escalate.
Swarm	Possible through local interactions, but not necessarily explicit voting.

Communication and Collaboration

Design specialist roles

Role-based assignment: fixed labels, predictable routing.

- Researcher gathers evidence; planner decomposes; coder implements.
- Tester runs checks; reviewer critiques; orchestrator routes.

Capability-based assignment: match demonstrated skills to requirements.

> Example: route a database migration to a specialist with schema tools.

Personas alone do not guarantee independent evidence.

Interaction patterns

Debate: positions → rebuttal → judge.

Reflection: generator → critic → bounded revision.

Division of labor: independent parallel work → synthesis.

Pipeline: research → outline → draft → edit.

Ensemble: independent outputs → best-of-N selector.

Red team: adversarial agent searches for failures.

Ensemble selection is not free

- Multiple outputs can improve reliability but multiply compute.
- Best-of-N uses a score from a verifier, judge, or reward model.
  - Example: three proposed SQL queries, choose the one passing tests.
- Shared models and shared evidence can produce correlated mistakes.

Only answers the “why” question

Coordination failures

- Duplicate work, missing context, stale versions, circular delegation.
- Prompt injection can propagate from one agent to another.
- A supervisor can omit or distort a specialist’s evidence.
- Example: required security reviewer times out; do not claim complete review.
- Set limits on agents, depth, messages, retries, time, and cost.

Communication and scaling costs

- Every inter-agent message consumes time and tokens.
- Communicate when expected information value exceeds message cost.
- Naive all-to-all message exchange scales as $O(n^2)$.
- Use hierarchy, sparse edges, topic subscriptions, or async messaging.
- Example: 20 agents need not broadcast every intermediate observation

Evaluation

Evaluate the entire team

- **SYSTEM:** task completion, latency, total token cost.
- **AGENT:** accuracy, communication efficiency, contribution.
- **COORDINATION:** gaps, duplicate work, conflict, evidence loss.
- Compare with same tasks, evidence, tools, and strong single-agent baseline.
- **Example:** parallel review catches one severe bug but costs 4× tokens

W9 Takeaway: Choose the simplest justified design

- Independent subtasks → parallel specialists.
- Clear sequential dependencies → pipeline.
- Debuggability critical → centralized supervisor.
- Resilience at scale → hierarchy or sparse peer network.
- Example: software team: architect → coder → tester → reviewer.
- Decision gate: retain extra agents only if measured gains

Over to tutorial