AMERICAN UNIVERSITY OF BEIRUT

Fall 2025

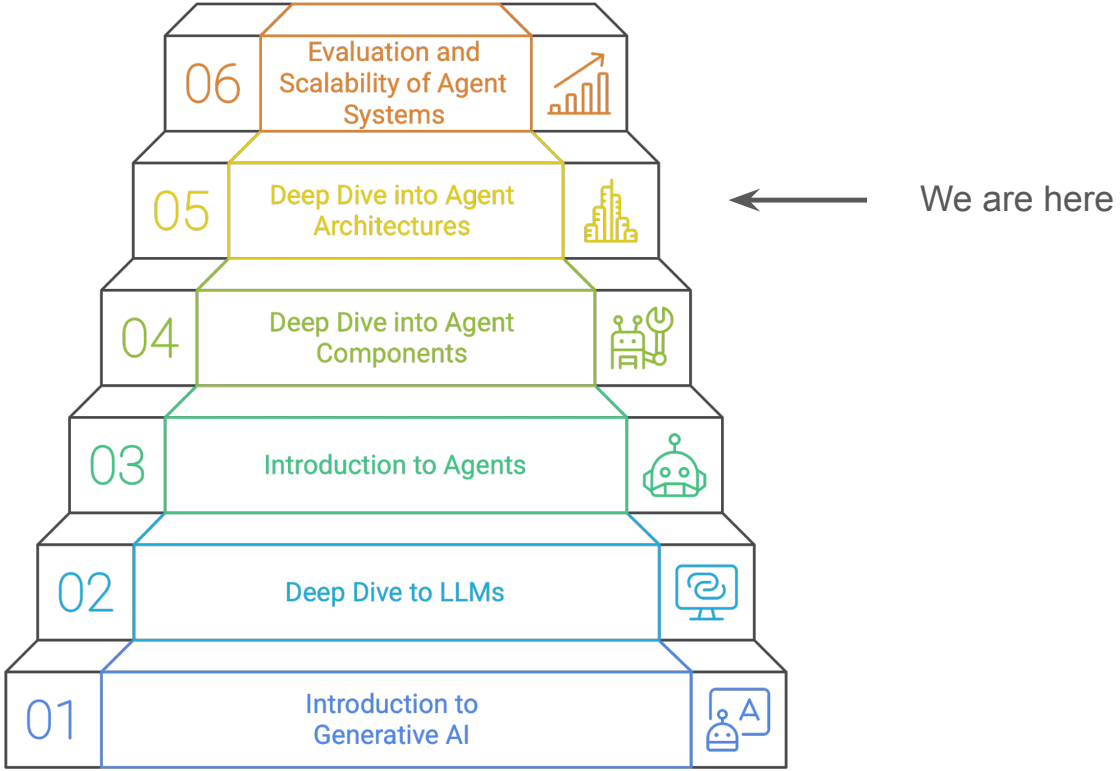
EECE 503P/798S: Agentic Systems

Chapter 8: Multi-Agent Systems

Learning Objectives

- Explain when **multi-agent systems (MAS)** are preferable to **single-agent designs**
- Compare major **MAS architectures** and select an appropriate **topology** for a given task
- Specify **agent-to-agent (A2A)** messaging schemas and coordination protocols
- Design **memory strategies** for MAS (private vs shared vs mediated).
- Implement a minimal MAS with an **A2A protocol** and justify safety/guardrail choices

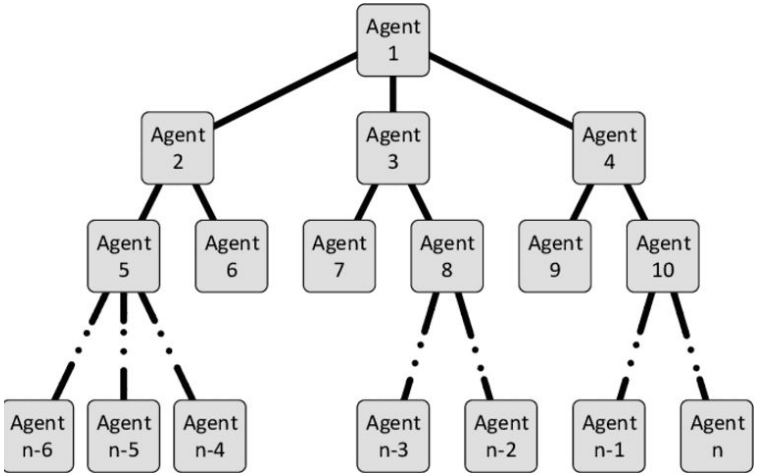
Course Outline



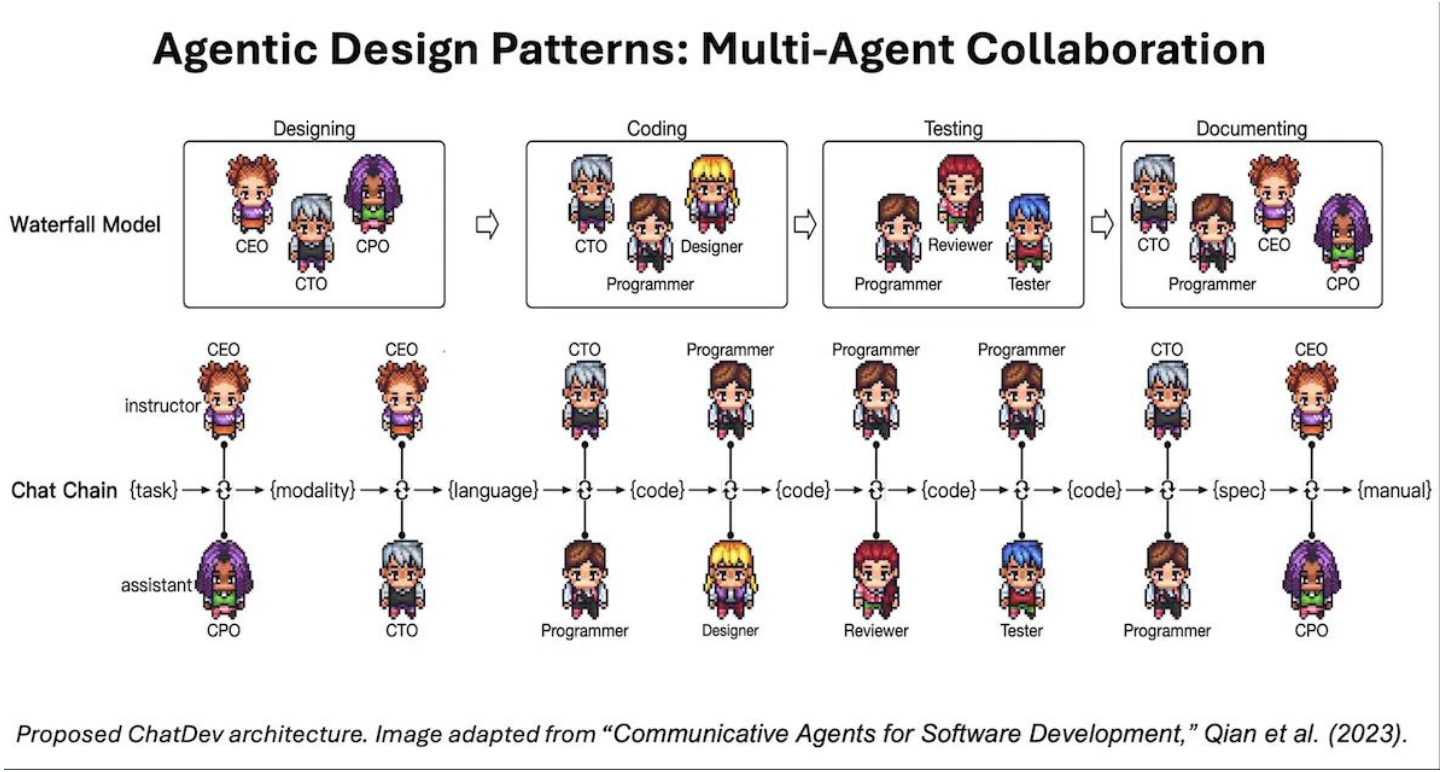
When do we Need Multiple Agents

Why do we need multiple agents?

- As the complexity of your tasks increases, **one ‘helpful assistant’ agent won’t be sufficient** to accurately execute your task.
- If your agent’s task is to create a fully functional multi-page website, you would need a planner, designer, programmer, quality assurance, and devops agents.
- The **quality of output would plummet** if you gave all the design, programming, and deployment tools to one agent



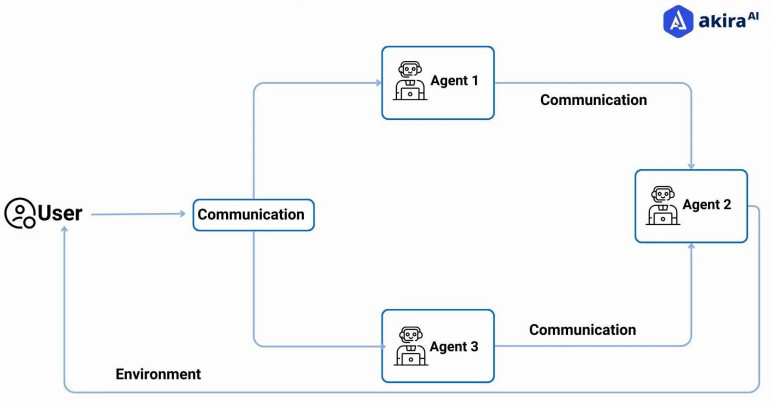
Why do we need multiple agents?



Proposed ChatDev architecture. Image adapted from "Communicative Agents for Software Development," Qian et al. (2023).

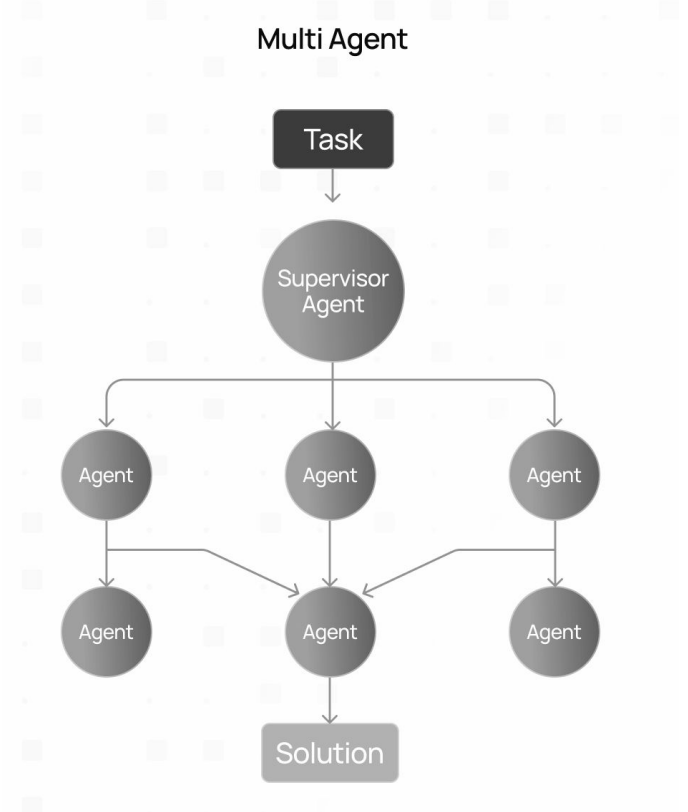
Why do we need multiple agents?

- It might seem counterintuitive that, although we are making multiple calls to the same LLM, we apply the programming abstraction of using multiple agents.
- Even with long context windows, LLMs struggle with **complex inputs**.
- Agentic workflows improve performance by guiding the model to handle **one subtask at a time**, such as focusing on what matters when acting as a software engineer.



Why do we need multiple agents?

- Multi-agent design breaks **complex tasks** into manageable **subtasks**.
- Similar to splitting a program into **processes or threads.**
- Makes building **complex systems** (e.g., a web browser) **easier**.

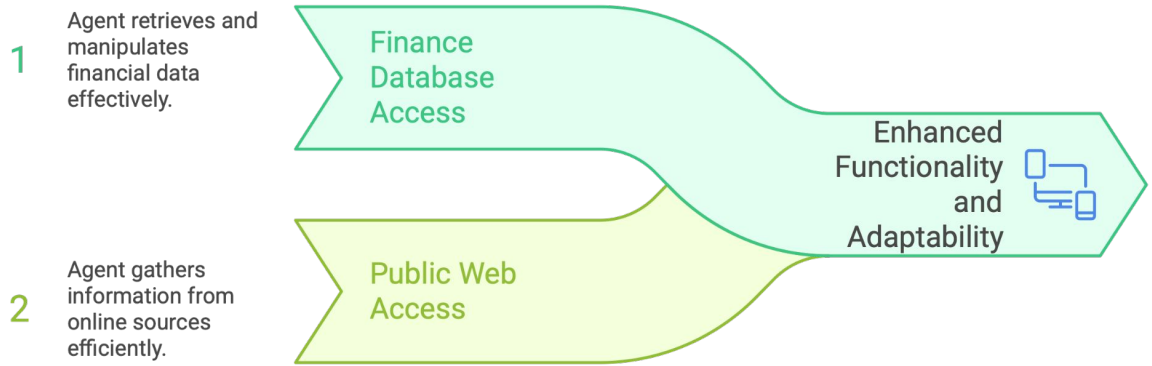


When to use and not to use Multi-Agent Systems

When should we use Multi-Agent Systems?

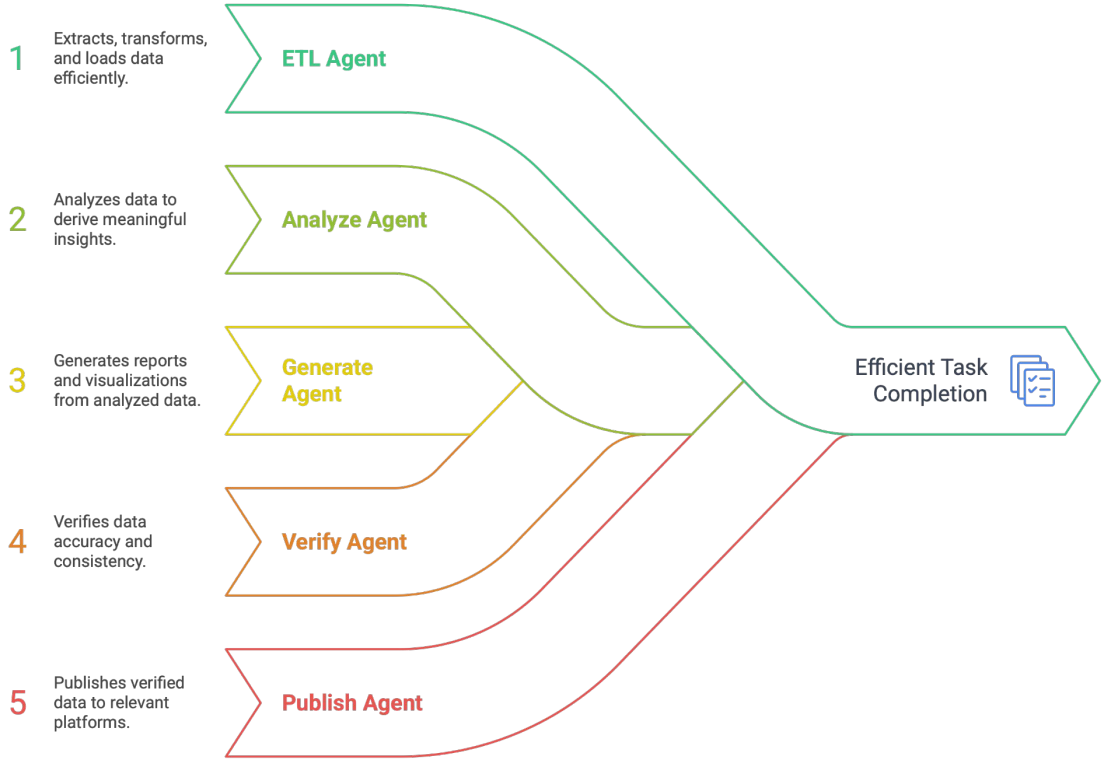
When should we use Multi-Agent Systems?

1. **Heterogeneous toolsets/permissions:**
   - Deciding on how to query a public dataset is very different from configuring an online search query.
   - Having **a specialized agent for each tool** would allow you more room to carefully instruct the agents on how to use those tools.



When should we use Multi-Agent Systems?

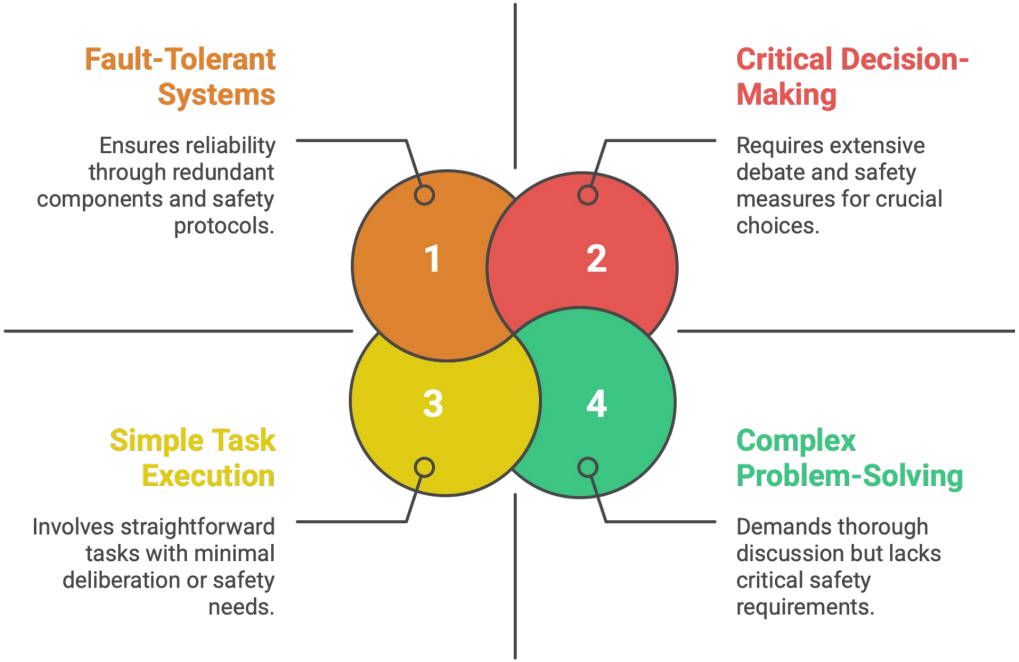
2. Tasks with natural pipelines
(ETL → analyze → generate → verify → publish)



When should we use Multi-Agent Systems?

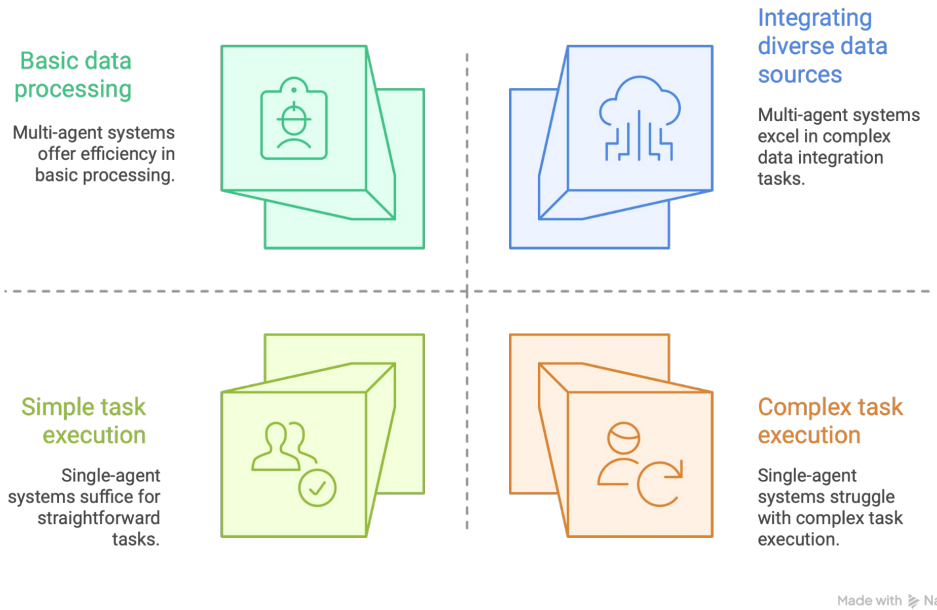
3. Need for:

- **deliberation** (debate/critique),
- **redundancy** (ensembles),
- or **safety separation** (untrusted vs trusted zones).



When should we use Multi-Agent Systems?

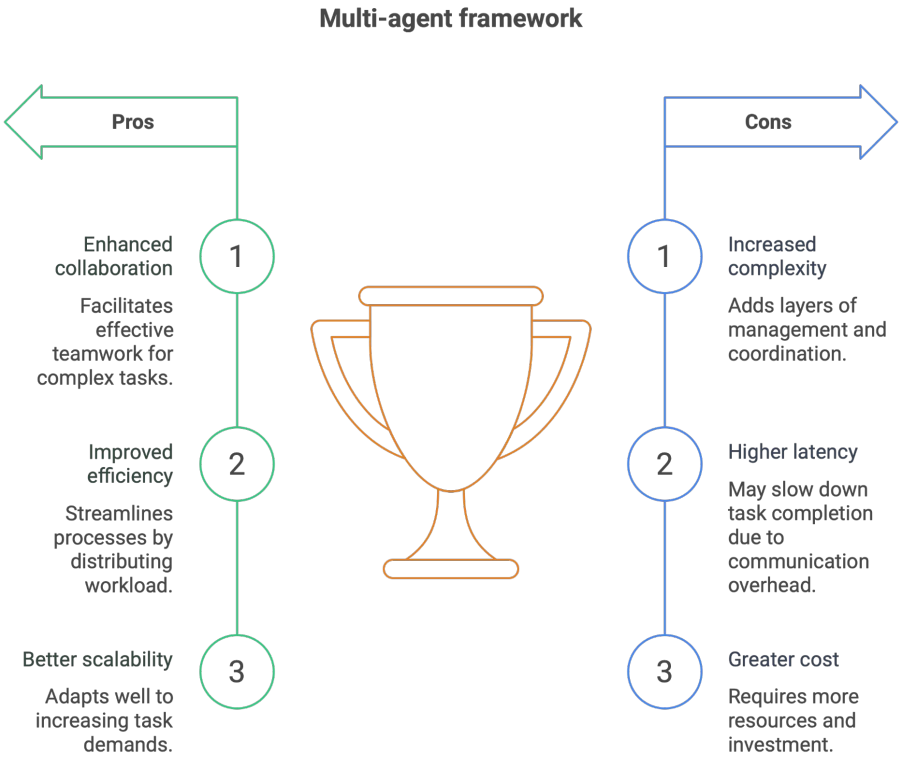
4. Multi-modal inputs or multi-domain expertise



When not to use Multi-Agent Systems?

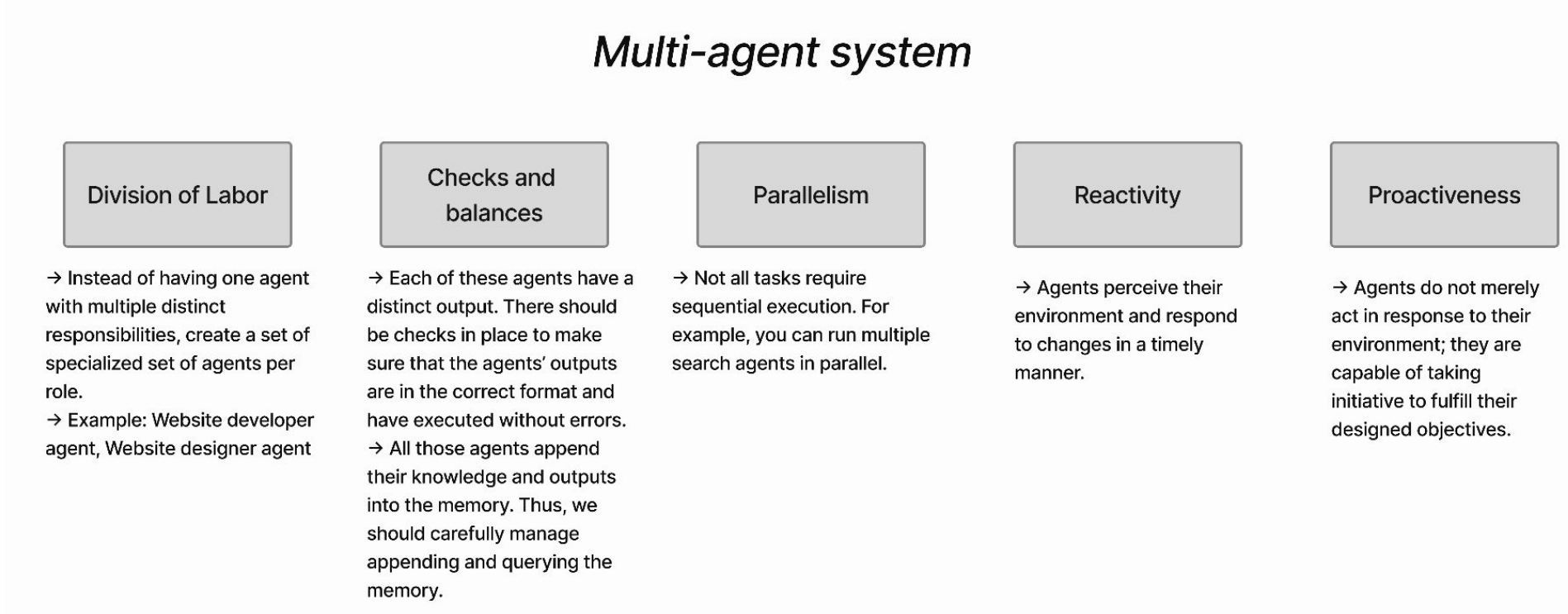
When should we not use Multi-Agent Systems?

- **orchestration complexity** outweighs **benefits**
- **latency/cost budgets** are tight

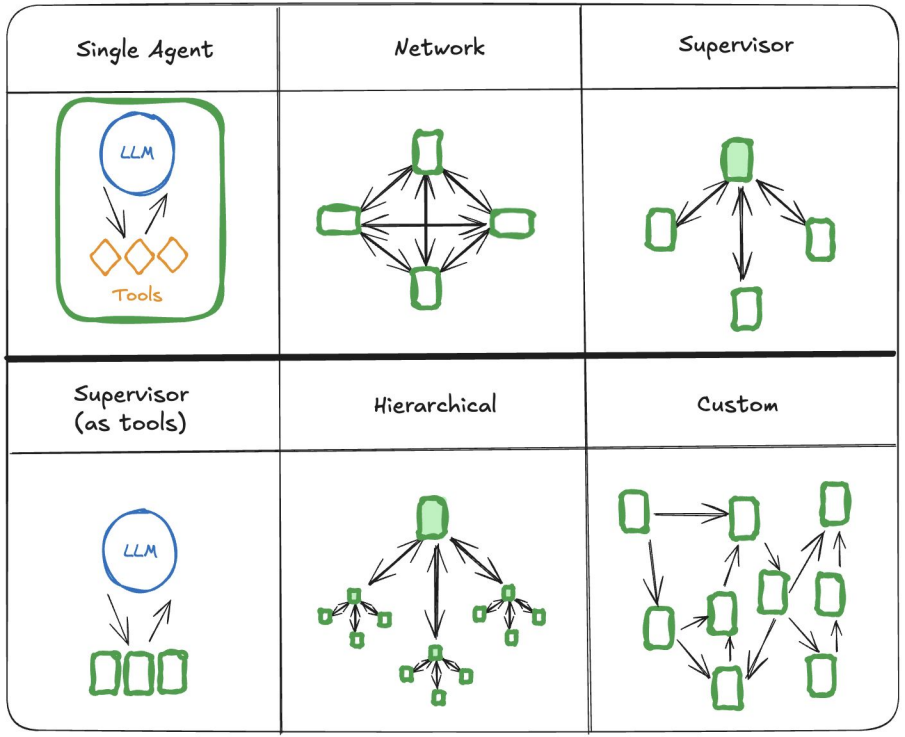


How do we Design a Multi-Agent System

Pillars of a Multi-agent System

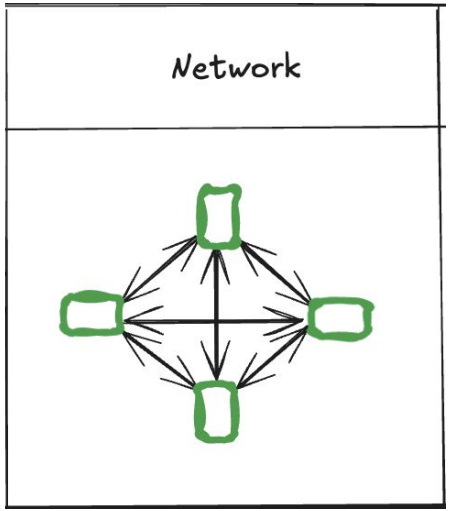


Multi-agent System Architectures



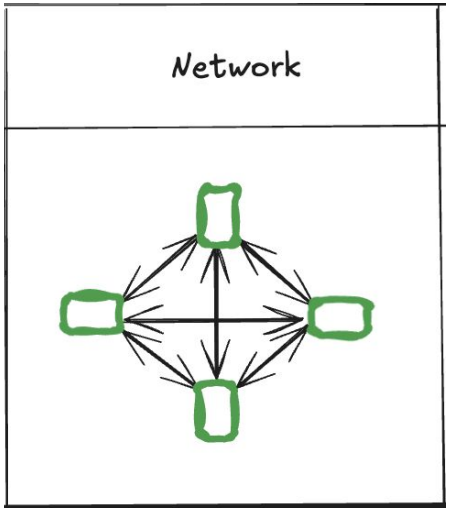
Network Architecture - Swarm / Peer-to-Peer Mesh

- **Structure:** No central controller; agents share signals or stigmergic cues.
- **Pros:** Robust to failures; scalable via local rules; emergent behavior.
- **Cons:** Hard to debug; non-deterministic; needs careful termination design.
- **Use-cases:** Exploratory search; content curation; distributed monitoring.



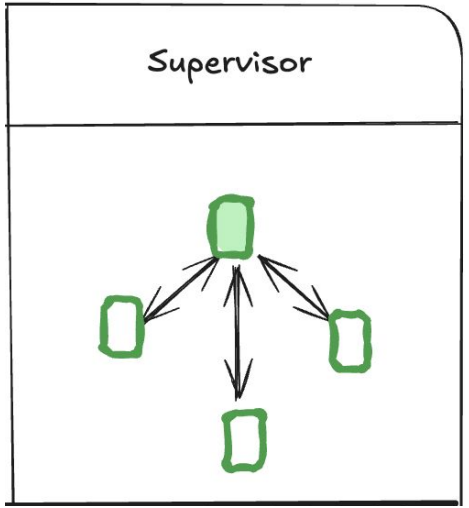
Network Architecture - Blackboard Architecture

- **Structure:** Agents read/write a **blackboard** (shared world state) and react to changes.
- **Pros:** Loose coupling; emergent cooperation; good for partial info fusion.
- **Cons:** Race conditions; requires arbitration/locks; provenance tracking needed.
- **Use-cases:** Multi-modal synthesis; incident response; analytics with many sources.



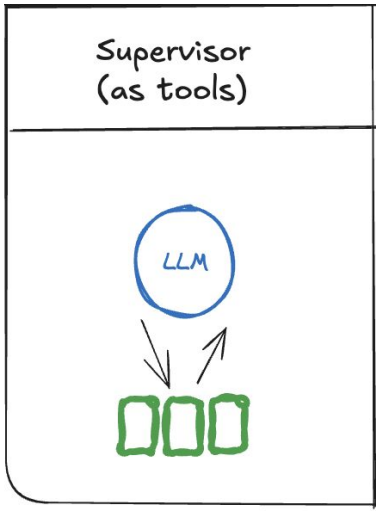
Supervisor Architecture

- **Structure:** Central **Conductor** routes tasks among **Specialist** agents.
- **Pros:** Simple mental model; global visibility for guardrails; easy logging.
- **Cons:** Single bottleneck; conductor becomes complex; central point of failure.
- **Use-cases:** Customer-support triage; research-then-write flows; retrieval → drafting → QA.



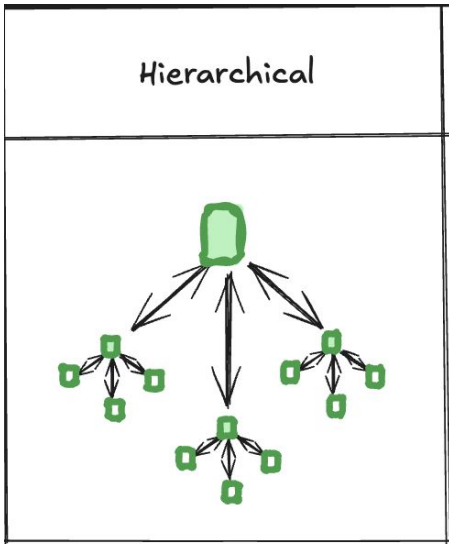
Supervisor Architecture - Agents as tools

- **Structure:** A single **Supervisor** plans and invokes **sub-agents** via *tool interfaces*. Each sub-agent exposes a narrow contract (input/output schema) and runs its own internal loop; the supervisor only observes structured I/O, not internal prompts.
- **Pros:** Strong isolation & testability; simple orchestration (tool-call API); per-tool auth & budgets; easy fallback/retries; compatible with mixed frameworks/models; unit-test sub-agents as black-box tools.
- **Cons:** Supervisor bottleneck; reduced shared context; risk of long tool chains; over-serialization (no peer-to-peer);



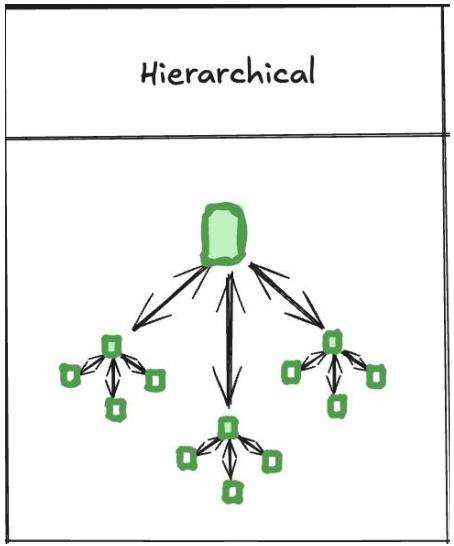
Hierarchical Architecture - Planner

- **Structure:** Multi-level planning with **task decomposition** and escalation.
- **Pros:** Scales to large tasks; clear ownership; supports recursive refinement.
- **Cons:** Overhead in planning; risk of over-decomposition; requires termination rules.
- **Use-cases:** Long-horizon projects; enterprise workflows with approvals.



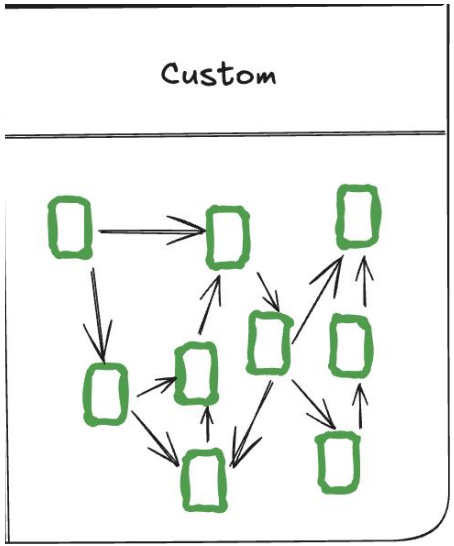
Hierarchical Architecture - Marketer

- **Structure: Manager** announces task; **Bidders** propose; winner executes.
- **Pros:** Dynamic allocation; handles heterogeneous costs/skills.
- **Cons:** Auction overhead; strategy gaming; pricing alignment is nontrivial.
- **Use-cases:** Tool/time-cost-aware routing; resource-bounded compute.



Hierarchical Architecture - Agents as tools

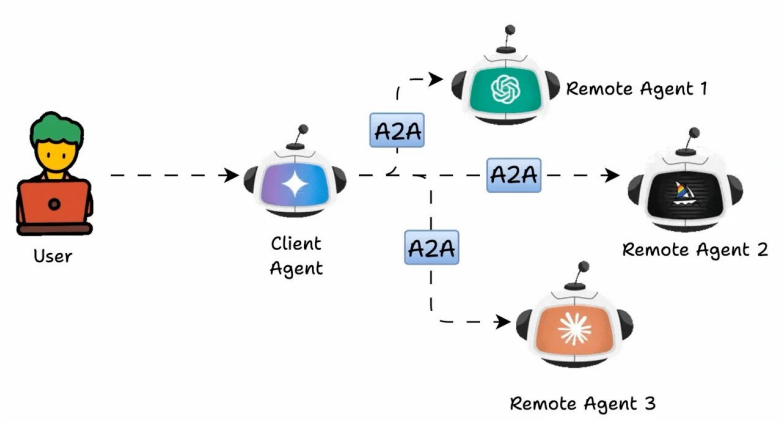
- **Structure:** Fixed sequence/graph of roles; outputs become next inputs.
- **Pros:** Deterministic, testable; parallel stages possible; great for SLAs.
- **Cons:** Rigid; backtracking requires side-channels; error propagation.
- **Use-cases:** Data ingestion & enrichment; report generation; code CI assistants.



Communication between Multiple Agents

Agent 2 Agent Protocol

- Launched by Google in April 2025, the Agent-to-Agent protocol is designed to enable **seamless interoperability within multi-agent systems.**
- It allows a user-facing **"client" agent to orchestrate** and delegate tasks to **specialized "remote" agents**.
- The protocol is built on standard web technologies like HTTP, JSON-RPC, and Server-Sent Events (SSE).

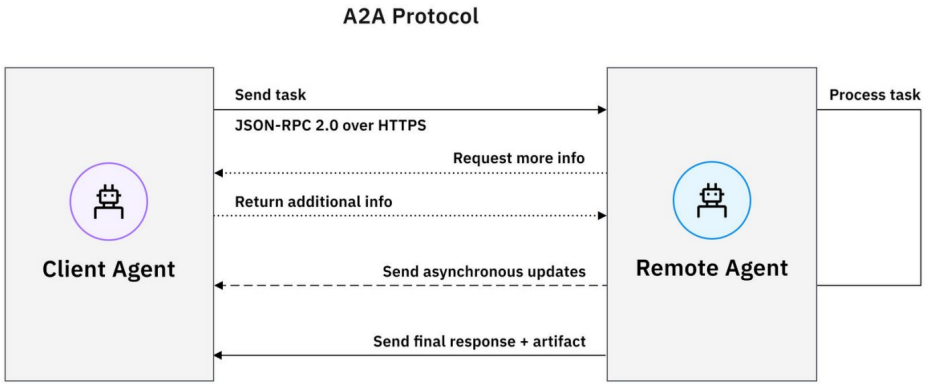


Agent 2 Agent Protocol - Components

1- **Agent Card**: Each remote agent publishes a standardized JSON file, or "Agent Card," that advertises its capabilities, skills, and authentication requirements. This allows client agents to discover the right specialist for a task.

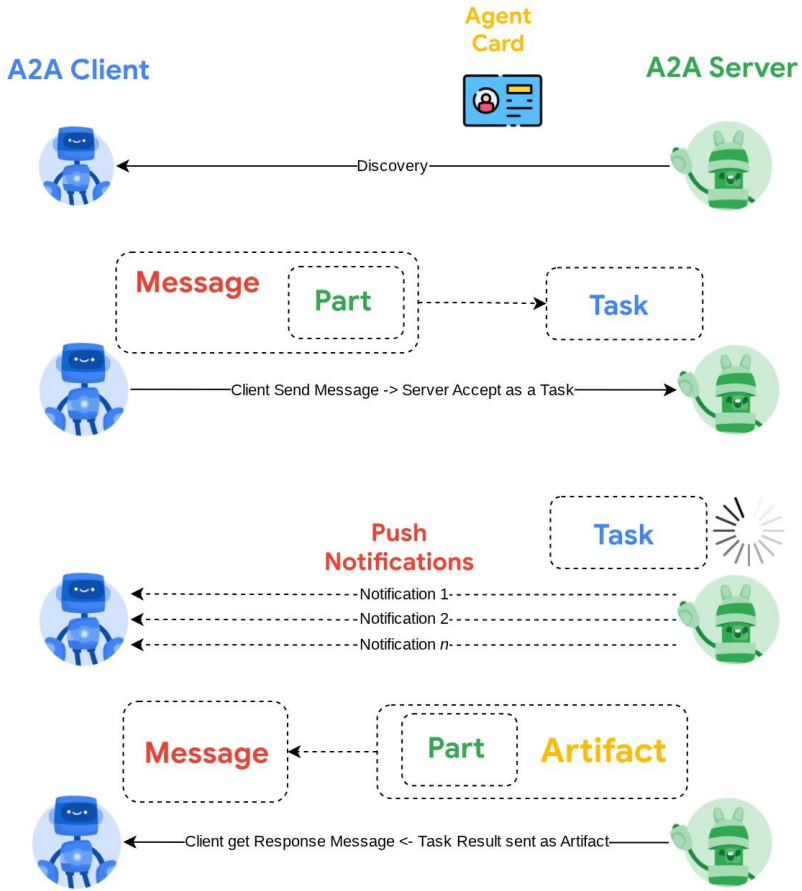
Agent 2 Agent Protocol - Components

2- **Task**: The central unit of work. When a client agent sends a request, it creates a task that moves through a well-defined lifecycle (e.g., submitted, working, completed, input-required).



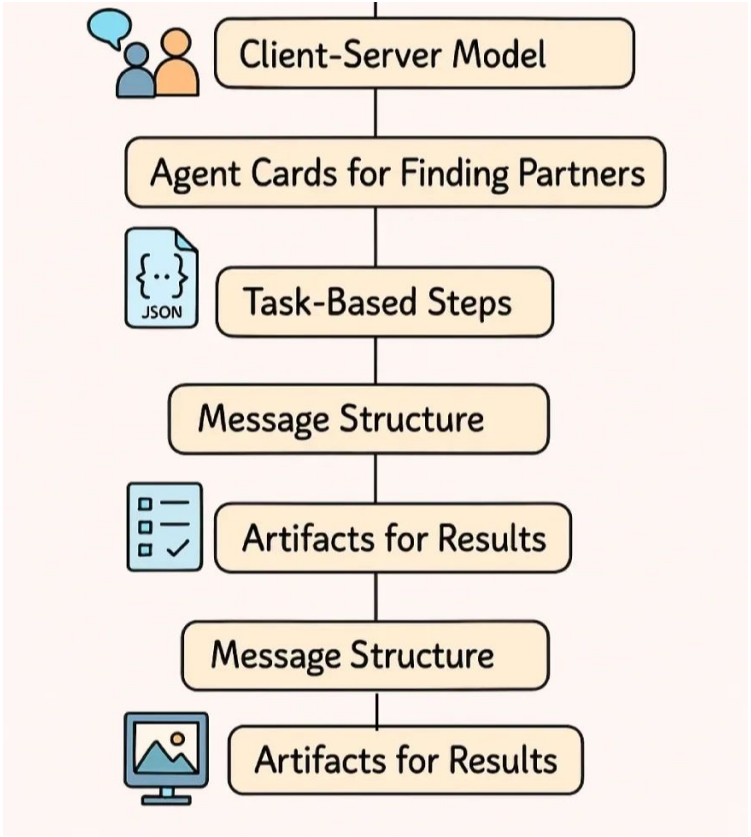
Agent 2 Agent Protocol - Components

3- **Messaging**: Agents communicate by exchanging messages. A single message can contain one or more "parts" for different content types, such as text, structured data (JSON), or files.

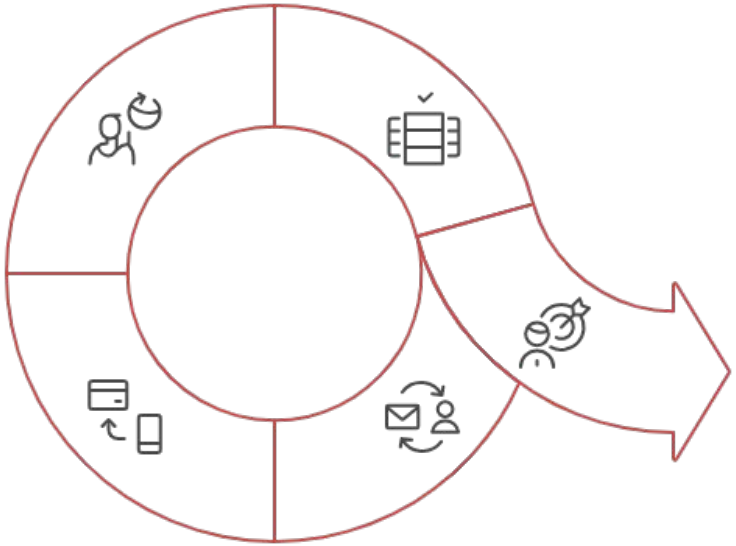


Agent 2 Agent Protocol - Components

4- **Artifacts**: Immutable outputs or results generated by a remote agent upon completing a task.

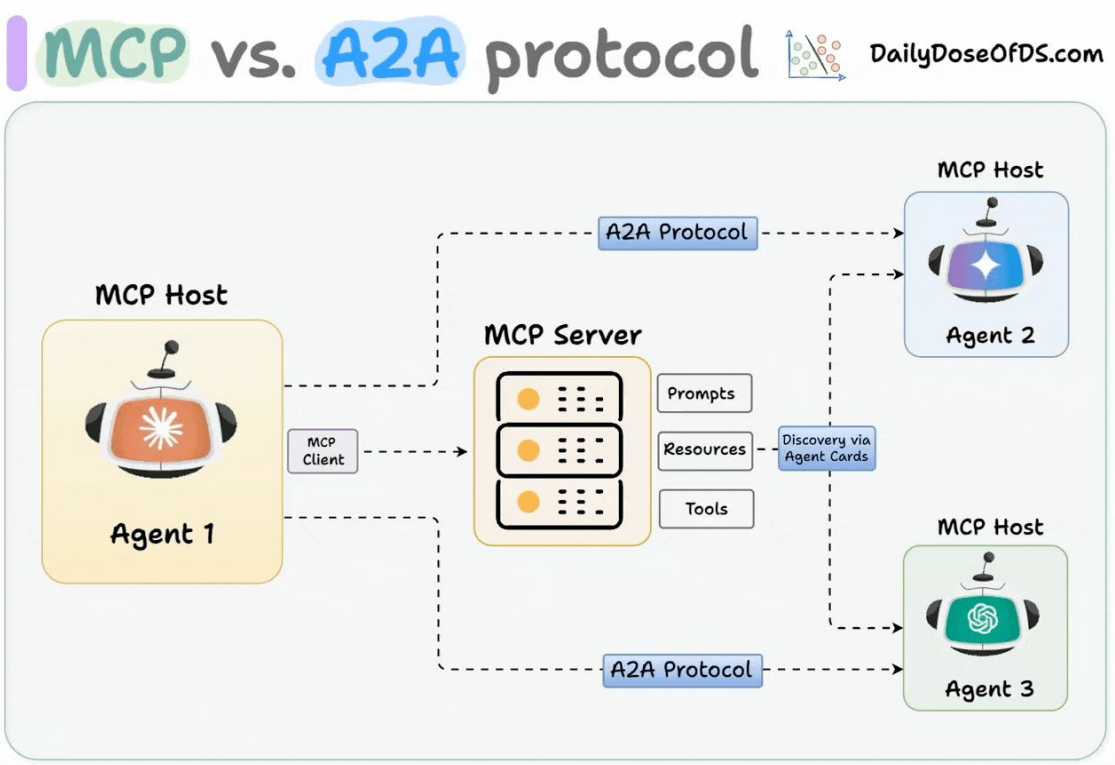


Agent 2 Agent Protocol



1	2	3	4	5
Receive Request	**Select Agent**	**Send Task**	**Process Task**	**Return Result**
Client agent receives user request	Client agent chooses suitable remote agent	Client agent sends task to remote agent	Remote agent processes the task	Remote agent sends back the completed task

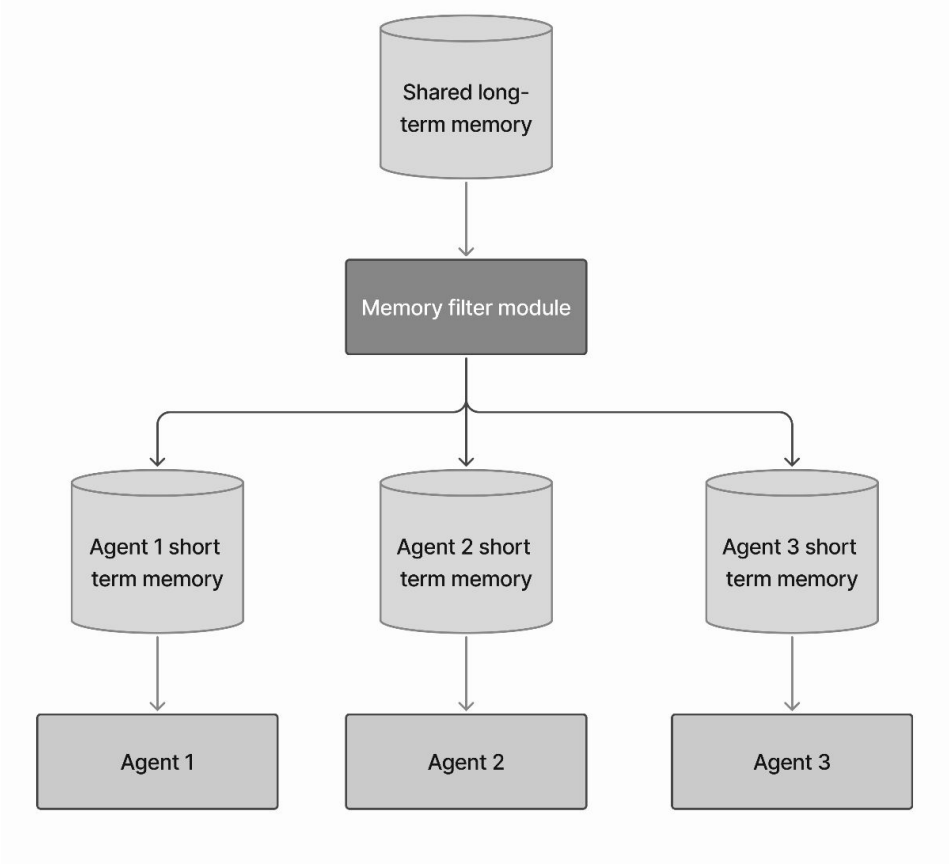
MCP vs A2A


MCP vs. A2A protocol
DailyDoseOfDS.com
MCP Host
Agent 1
MCP Client
MCP Server
Prompts
Resources
Tools
Discovery via Agent Cards
A2A Protocol
MCP Host
Agent 2
A2A Protocol
MCP Host
Agent 3

Hands on (Architectures + A2A): 1_multi_agent_architectures.ipynb

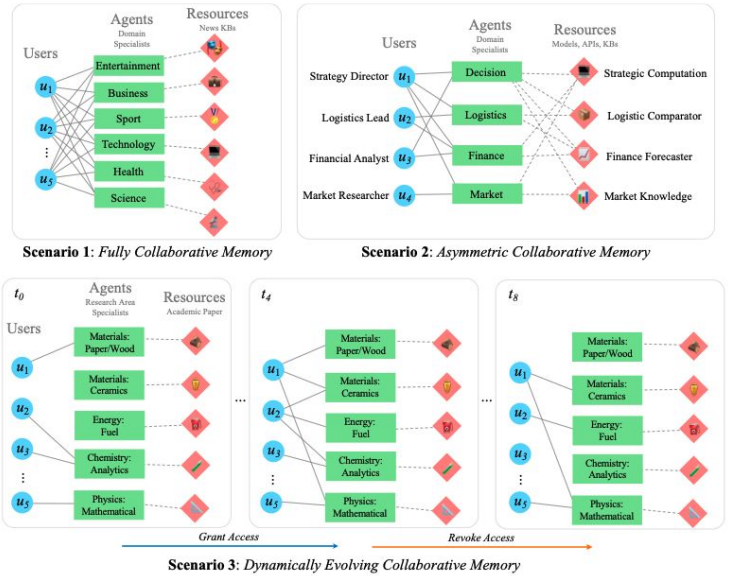
How do Agents Share Memory

Multi-agent shared memory



Collaborative Memory Framework (Rezazadeh et al., 2025)

- Introduces **two-tier memory**: **private** memory (user-specific) and **shared** memory (selectively shared).
- Uses **bipartite graphs** to encode asymmetric, time-evolving access permissions.
- **Immutable provenance metadata** ensures **auditability** and **retrospective permission** checks.



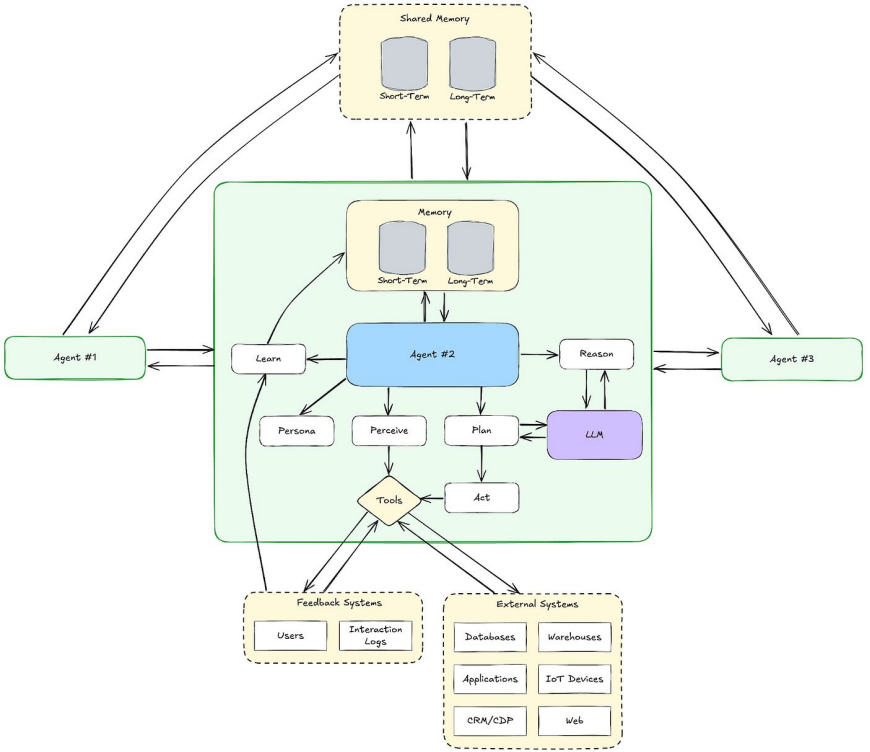
Scenario 1: *Fully Collaborative Memory*

Scenario 2: *Asymmetric Collaborative Memory*

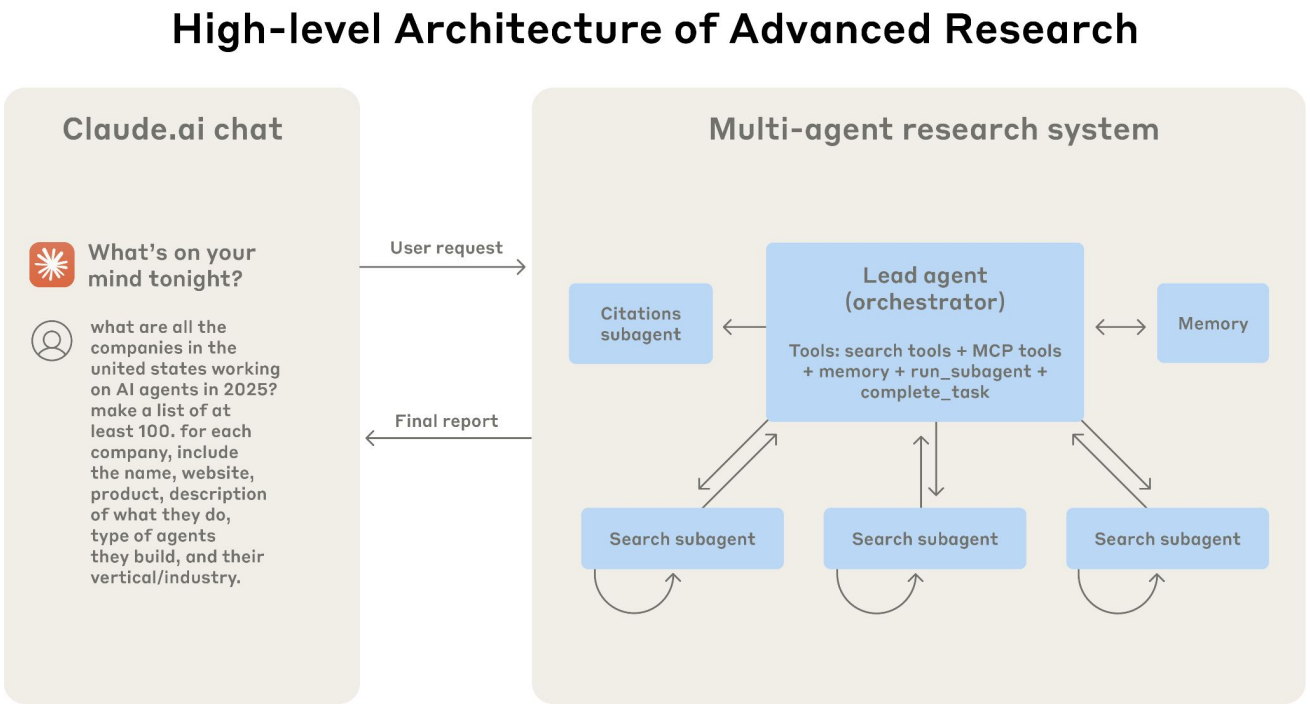
Scenario 3: *Dynamically Evolving Collaborative Memory*

Memory Sharing in Practice: Real-Time and Persistent Memory

- Memory-Sharing (MS) framework aggregates **query-response pairs** from multiple agents.
- Enables agents to **retrieve relevant examples** from a **shared memory pool** to improve open-ended task performance.
- Empirically shown to **enhance reasoning** and **knowledge transfer** in specialized domains.



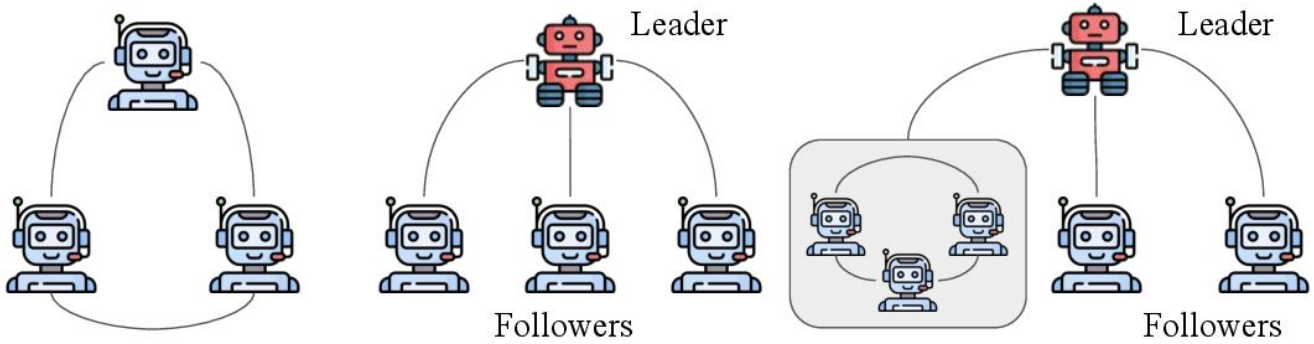
Anthropic's Research Agent Memory



Challenges of Multi-Agent Systems

Challenges of multi-agent systems

- Instruction **leakage** between roles; prompt **drift**; duplicated work; **infinite** loops.
- **Inconsistent** schemas; brittle coupling to tool **I/O**; **silent failures** on the bus.
- **Grounding gaps:** agents cite each other rather than sources.



a. Equi-level structure b. Hierarchical structure c. Nested structure

Conclusion

- Agents are stateful and errors compound.
- **Agents can run for long periods of time, maintaining state across many tool calls.** This means we need to durably execute code and handle errors along the way.
- Without effective mitigations, **minor system failures can be catastrophic for agents.**
- When errors occur, we can't just restart from the beginning: **restarts are expensive** and frustrating for users.
- Instead, we built systems that can **resume from where the agent was when the errors occurred.**

Thank you