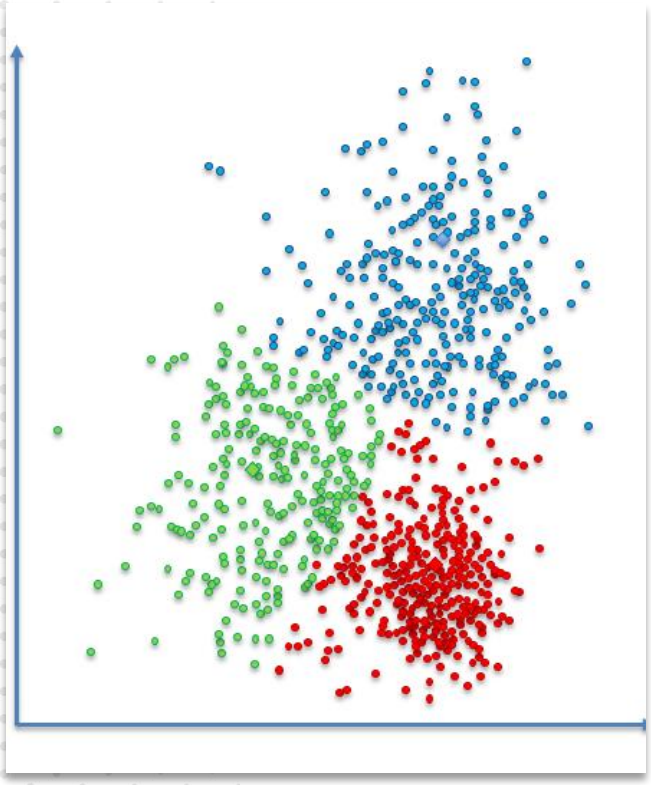
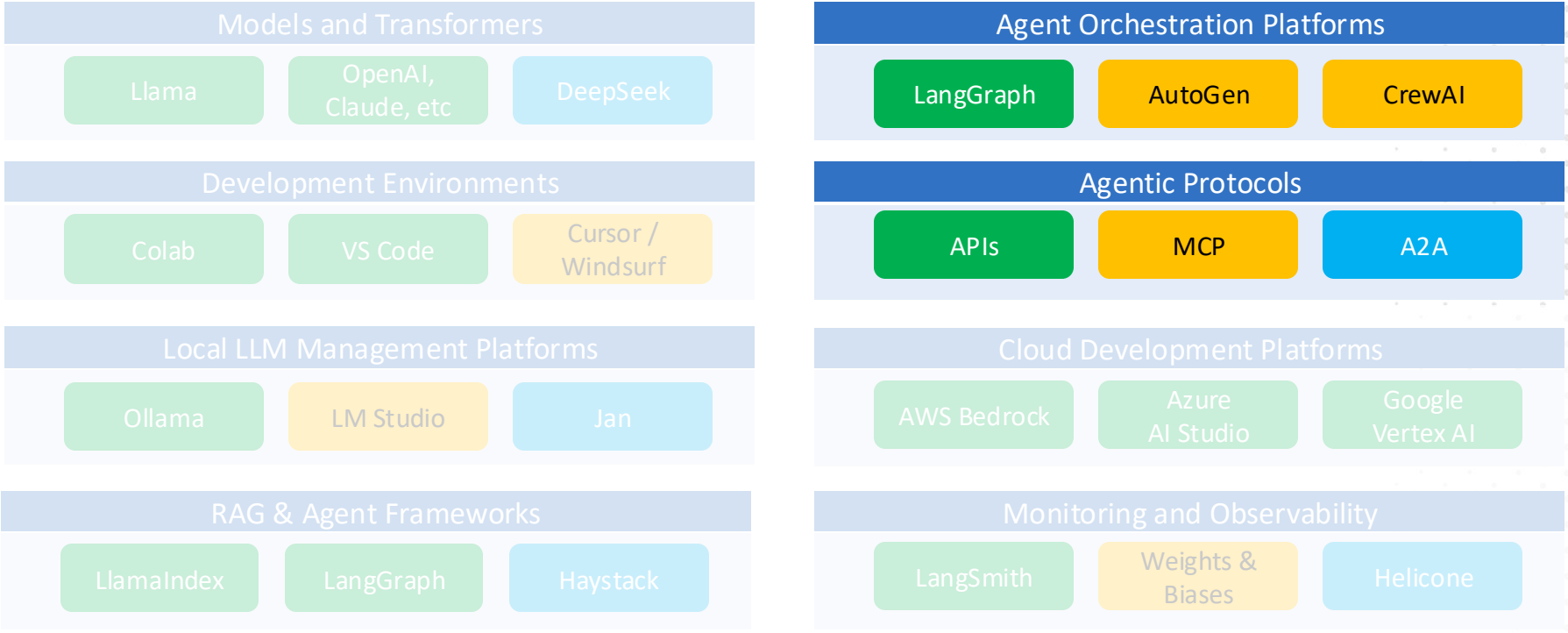


Lesson 4:
Agentic AI Development Protocols

Objectives

- Why would you need an agentic protocol?
- Agentic protocols landscape
- Introduction to Message Chain Protocol (MCP)
- MCP in action
- Agent to Agent (A2A)
- Model Context Protocol, the other MCP

Agent Orchestration and Agentic Protocols

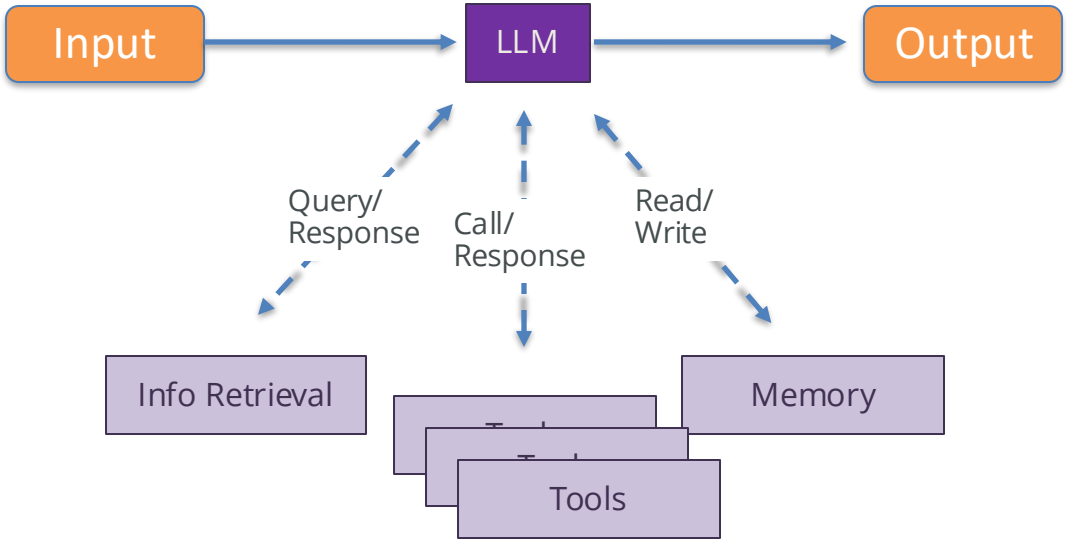


Mature | Strong Contenders | Emerging

Pearson

Agent Often Exchange with One Another

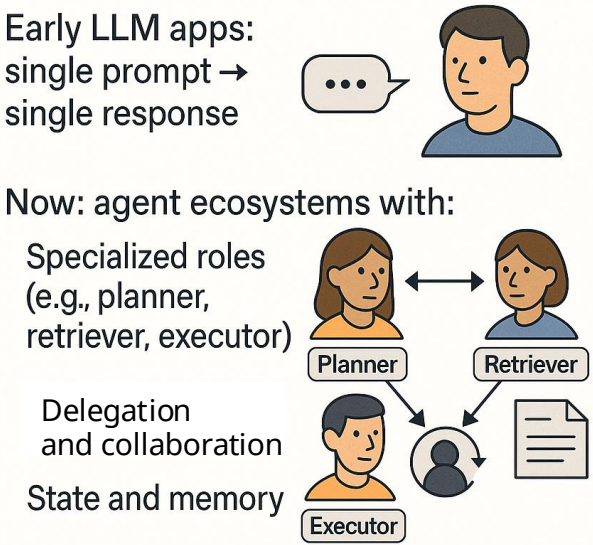
Recall our augmented LLM as a basic agent:



The agent may query multiple tools, some of them will often be agents

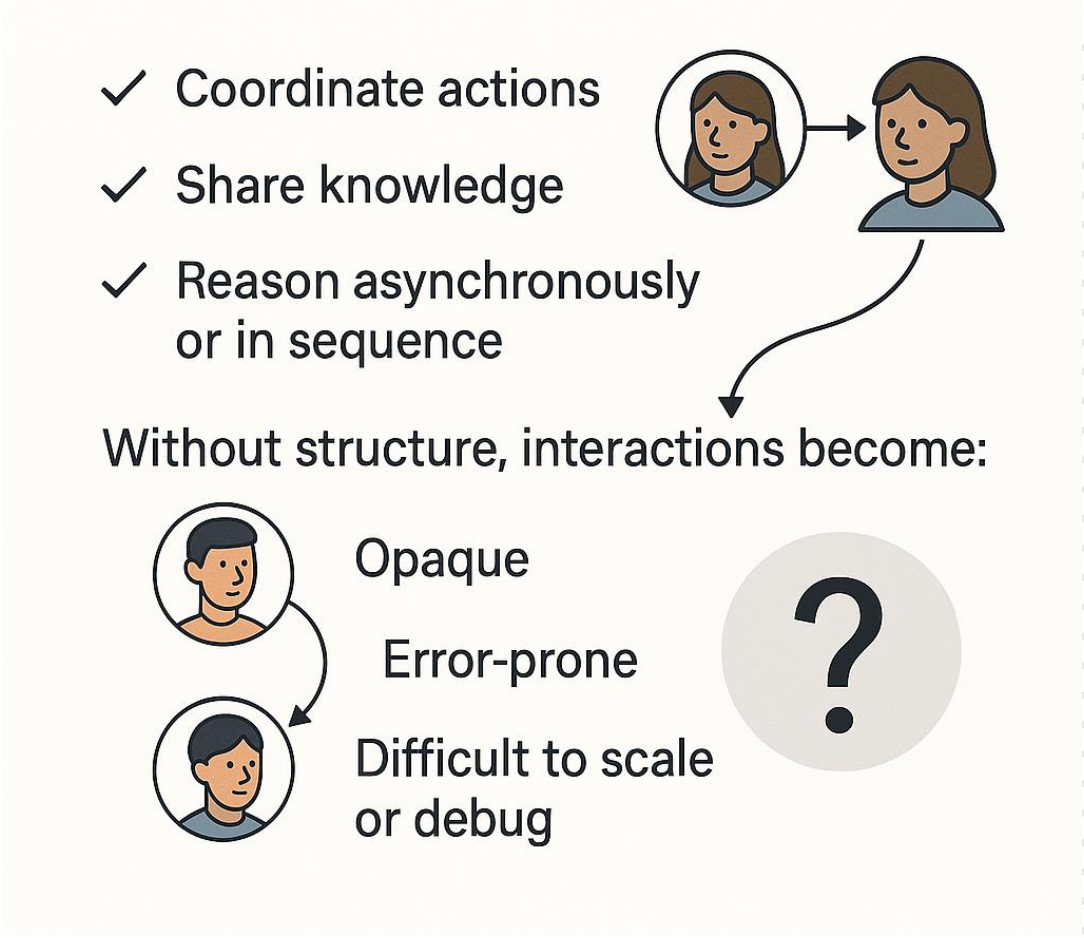
Multi-Agent Systems

- Specialized agents “appear” to be more efficient
  - Planner, retriever, executor, critic/evaluator, reasoner, summarizer,…
  - Controller / orchestrator, memory manager, reflection agent, data-analyst, search agent, code generator / fixer, API negotiator / communicator, translator / converter, …



Agent to Agent Exchange Challenges

- Agent-to-agent exchanges are easy at small scale, if you control all agents
- Much harder at scale, especially with multiple agents
- Without shared structure
  - Agents may duplicate efforts, lose track of state, make decisions that lack context, become hard to debug,…



Pearson

Why Agentic Protocols

- Agentic protocols provide a common language and structure to:
  - Track conversation
  - Record reasoning
  - Maintain consistency
- Similar in concept to applications APIs

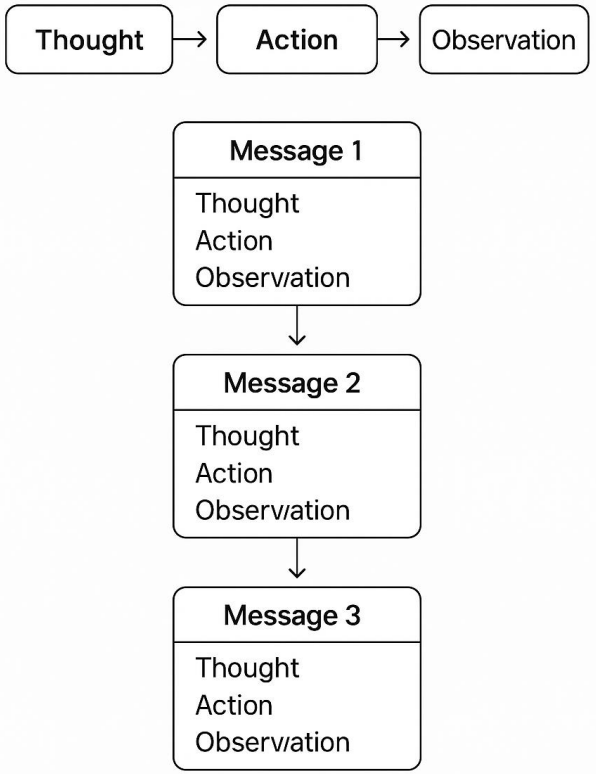
Agentic Protocols Landscape

- There are many agentic protocol candidates – MCP is dominant, A2A is an interesting second

Protocol	Style	Best For	Used In
MCP (Message Chain)	Structured	Auditable, sequential reasoning agents	LangGraph, AutoGen
A2A (Agent to Agent)	Decentralized	Reactive agent societies, negotiation	CrewAI (partially), CAMEL
DF (Directory Facilitator)	Directory	Open registries, capability discovery	JADE, classical MAS
Blackboard	Shared memory	Collaboration on global state	Robotics, planning agents
Prompt-as-Protocol	Implicit	Simple chains, experimental LLM agents	ReAct, AutoGPT, BabyAGI

Message Chain Protocol

- Provides a structured way to represent each agent action as a self-contained message unit
- Each message includes:
  - Thought: what the agent is thinking
  - Action: what it decides to do (e.g., call a tool)
  - Observation: what happened as a result
  - State: the current memory or context
- Messages are chained together over time:
  - Agent 1 -> Agent 2 -> Agent 3…
  - Forms a sort of versioned conversation log with embedded reasoning



The Many Advantages of Agentic Protocols

Clarity and Interpretability

Enforcing a clear structures makes it easy to understand:

- What an agent thought
- Why it chose the action
- What happened afterward

State Management and Continuity

Each message is a snapshot, enabling:

- Reliable recovery after interruption
- Reproducibility and rollback
- Systematic memory progression

Tool Integration and Modularity

Each action is explicit & linked to tools or functions allowing:

- Modular plug&play of tools (e.g. API calls)
- Better interface definition between agents and tools

Multi-agent Orchestration

Message chaining allows for clear role definition (agent A vs agent B), structured collaboration while preserving autonomy

Training and Simulation Readiness

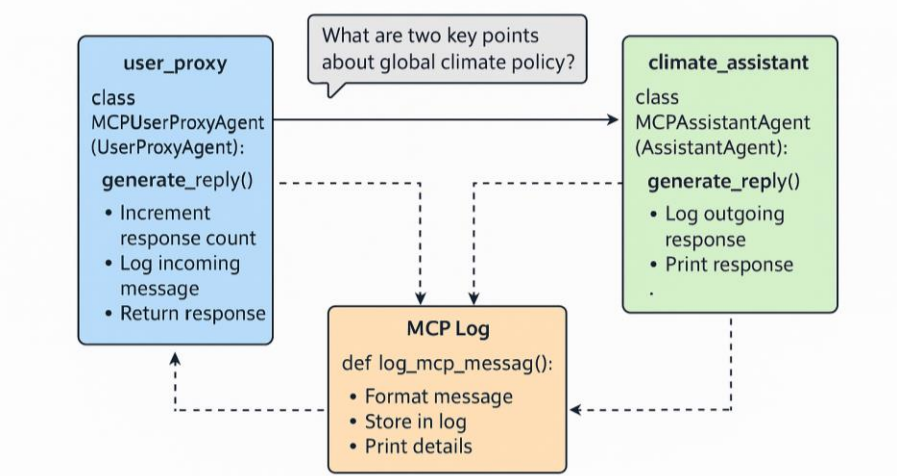
MCP logs complete thought-action-observation loop:

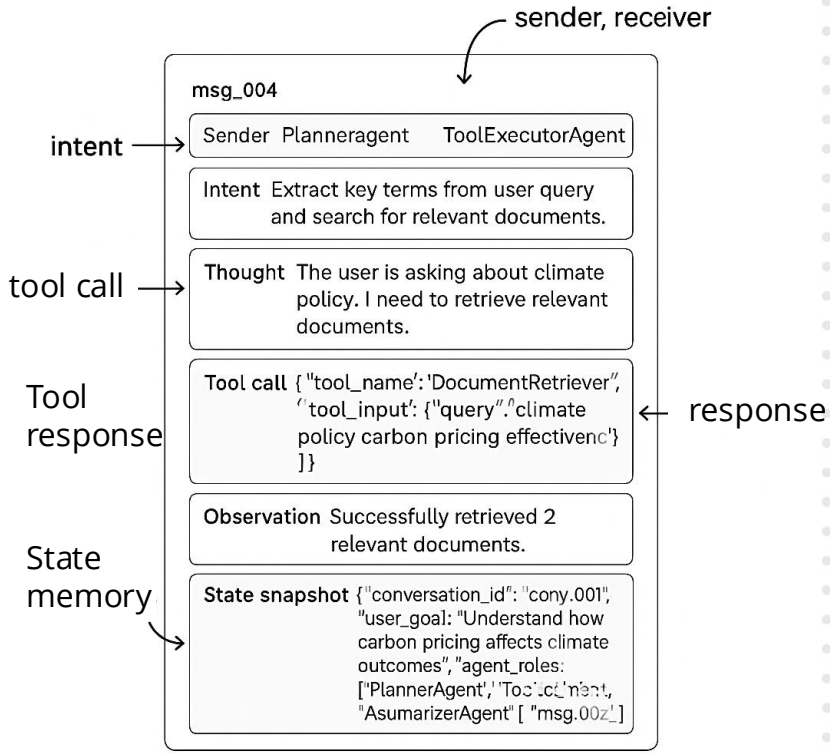
- Replay possible
- Can be used for fine-tuning or few-shot learning

How MCP Works

DEMO

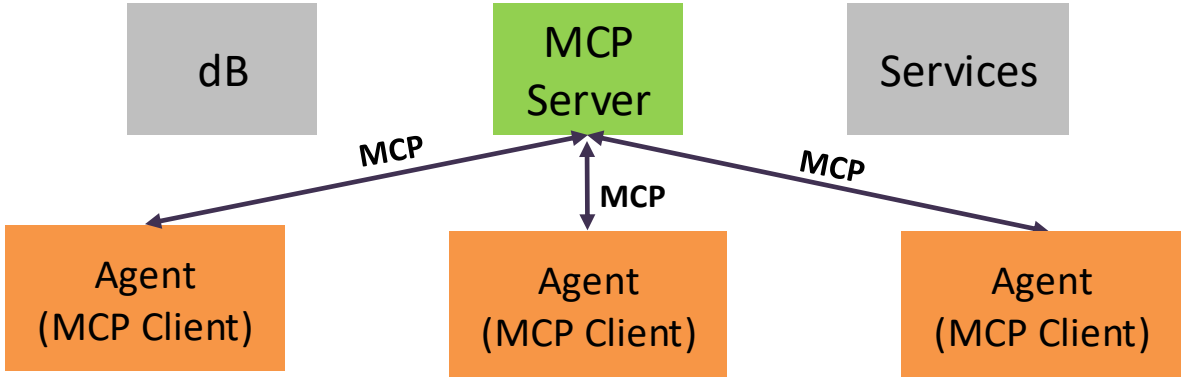
- MCP provides a format for the exchange between agents
  - Agents are merely instructed to keep the format and provide the log
- We can see MCP in action by setting 2 agents to exchange





MCP Servers

- You can use MCP directly between agents
- Enterprises who use MCP at scale tends to add MCP-compatible a server:
  - Hub to store all messages, route requests between agents and track global conversations
  - Uses database (e.g. MongoDB, PostgreSQL) to store all messages
  - Provides agent and tool registry
  - Easy to use Web UI for monitoring & management



A2A

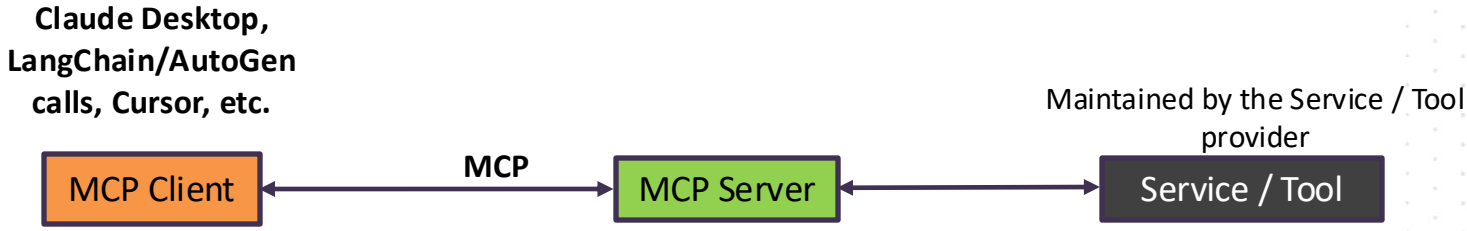
- MCP is a protocol, A2A a peer-to-peer messaging model
  - No enforced structure, each agent decides
  - Agents may or may not log or maintain state
  - Requires adapters between agents with different expectations
  - Easy to use if you stay within a single ecosystem (e.g., within LangChain or CrewAI with a single type of LLM behind your agents)
- A2A is lighter and simple to use, but harder to troubleshoot and to scale
- A2A is interesting for simple agents or fast prototyping / PoC
- MCP is better for complex workflows, research, multi-step reasoning and production agents

Model Context Protocol

- Message Chaining Protocol solves challenges of agents talking to each other
- Even without agents, LLM actions are difficult to control:
  - Models forget what they are doing - long conversation exceed the token limit
  - Tool usage has been amateurish - ad hoc prompt engineering or monkey-patching the LLM's response to extract function call
  - Context and state is lost – especially in multi-steps dialogs and multi-agent systems
  - Controlling LLM behavior is difficult – you can't guarantee consistent behavior, prompting is a black box
- At the end of 2024, Anthropic introduced the Model Context Protocol – open standard to describe how an LLM connects to external tools

What Model Context Protocol Does

- MCP establishes a client server relationship between the LLM (client) and the server – interface to all tools



MCP Client

- A software agent with integration to LLM model(s)
- Orchestrates conversation flow
- Initiates connections to MCP servers
- Handles user requests and responses

MCP Server

- Connects to tools and functions
- Acts like a “translator” between the tool and the client
- Can act as a security gateway

Why is MCP Adoption Rapidly Growing?

- MCP defines message types:
  - initialize - for establishing connections
  - resources/list and resources/read - for discovering and accessing data sources
  - tools/list and tools/call - for discovering and invoking external tools
  - prompts/list and prompts/get - for accessing prompt templates
- Each message has a standardized format and required fields:
  - Request has a defined JSON schema specifying required fields like method, params, and optional fields like id for tracking requests.
  - Responses follow standardized formats with fields like result for successful responses or error for failures, ensuring consistent error handling.
  - Content within each message type also standardized - for example, a tools/call request must include the tool name and properly formatted arguments that match the tool's defined schema.
  - MCP specifies how different data types (strings, numbers, objects, arrays) should be represented and validated.

Why is MCP Adoption Rapidly Growing?

- Models forget what they are doing
  - The task_state field allows you to explicitly track the goal, current progress, and next step.
  - This keeps the LLM “anchored” to the overall mission, even after many exchanges.
- Tool usage has been amateurish
  - MCP uses a standardized tools field that defines available functions, required arguments, and types.
  - LLMs can declare a function call (tool_call) and later consume a tool_response with traceable semantics.
- Context and state is lost
  - MCP defines messages with full tracking (turn, sender, receiver) and supports structured history, memory, or state_snapshot.
  - Enables shared understanding and context accumulation.
- Controlling LLM behavior is difficult
  - Provides structured inputs (thoughts, tools, goals, memory) and clearly bounded outputs (observations, tool_responses).
  - This allows better evaluation, debugging, and fine-tuning.

Example MCP Exchange

DEMO

A question is asked:


```
{
  "message_id": "msg_001",
  "timestamp": "2025-06-20T18:45:12.302Z",
  "turn": 1,
  "sender": "user_proxy",
  "receiver": "climate_assistant",
  "thought": "What are two key points about global climate policy?",
  "tool_call": null,
  "tool_response": null,
  "observation": "",
  "state_snapshot": {
    "is_response": false,
    "conversation_stage": "Turn 1"
  }
}
```


The LLM/agent decides to call a tool:


```
{
  "message_id": "msg_002",
  "timestamp": "2025-06-20T18:45:13.101Z",
  "turn": 2,
  "sender": "climate_assistant",
  "receiver": "user_proxy",
  "thought": "Calling get_policy_facts() to retrieve climate policy details.",
  "tool_call": {
    "function": "get_policy_facts",
    "arguments": {
      "region": "global"
    }
  },
  "tool_response": null,
  "observation": "",
  "state_snapshot": {
    "is_response": false,
    "conversation_stage": "Turn 2"
  }
}
```

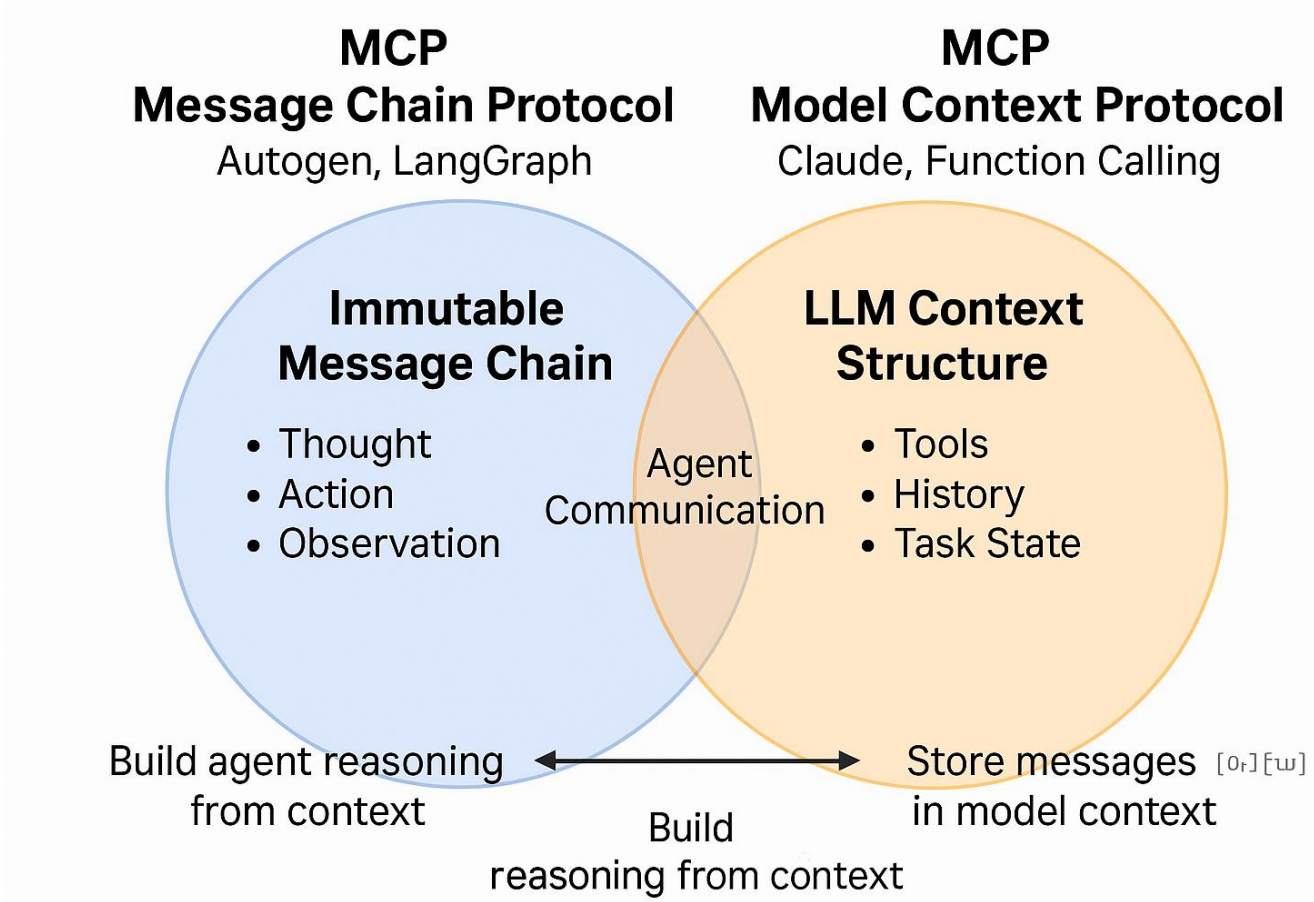

Response from the tool:


```
{
  "message_id": "msg_003",
  "timestamp": "2025-06-20T18:45:14.001Z",
  "turn": 3,
  "sender": "tool",
  "receiver": "climate_assistant",
  "thought": "",
  "tool_call": null,
  "tool_response": {
    "result": [
      "Emission caps and trade systems",
      "Renewable energy investment mandates"
    ]
  },
  "observation": "Tool response received successfully.",
  "state_snapshot": {
    "is_response": true,
    "conversation_stage": "Turn 3"
  }
}
```


Agent reply (after response from the tool):


```
{
  "message_id": "msg_004",
  "timestamp": "2025-06-20T18:45:15.503Z",
  "turn": 4,
  "sender": "climate_assistant",
  "receiver": "user_proxy",
  "thought": "Here are two key points: 1) Emission cap-and-trade
systems, 2) Government-mandated investments in renewable energy.",
  "tool_call": null,
  "tool_response": null,
  "observation": "",
  "state_snapshot": {
    "is_response": true,
    "conversation_stage": "Turn 4"
  }
}
```

MCP and MCP Have Commonalities



Use MCP and MCP Together

- Use Message Chain Protocol to model multi-agent flows:
  - Agent A sends a message to Agent B.
  - Each message includes prior state, tool results, intent, etc.
- Format each message using Model Context Protocol:
  - Each message in the chain adheres to OpenAI's structured schema.
  - Includes fields like tool_calls, memory, task_state, role, etc.
- This layered approach gives you:
  - Deterministic stateful behavior across agent interactions
  - Structured tool integration in each message
  - Replayability of the entire interaction history