Tool Calling, Frameworks, & MCP

Shreya Havaldar

Announcements

- **If you missed your demo, reschedule ASAP** (link is on Ed)

- Final Project Proposal details are on website
  - Groups of 2
  - Proposal is due in 1 week!

Today’s Lecture

1. **Tool Calling**
2. Frameworks
3. MCP

Key failures of plain prompting

- Stale knowledge
  - *“Who was the best figure skater at the 2026 winter olympics?”*

- Inability to take actions
  - *“Can you check if this code compiles?”*

- Inability to access private or real-time context
  - *“Check my calendar and see if I am free tomorrow”*

Tool Calling fixes these issues

- *“Who was the best figure skater at the 2026 winter olympics?”*
  - → Web search tool updates stale knowledge

- *“Can you check if this code compiles?”*
  - → Code execution or calculator tools enable actions

- *“Check my calendar and see if I am free tomorrow”*
  - → Integration with email, calendar, Slack, etc. allows access to private context

What is a tool call?

Tool calling: The model emits a structured request indicating that an external capability should be invoked.

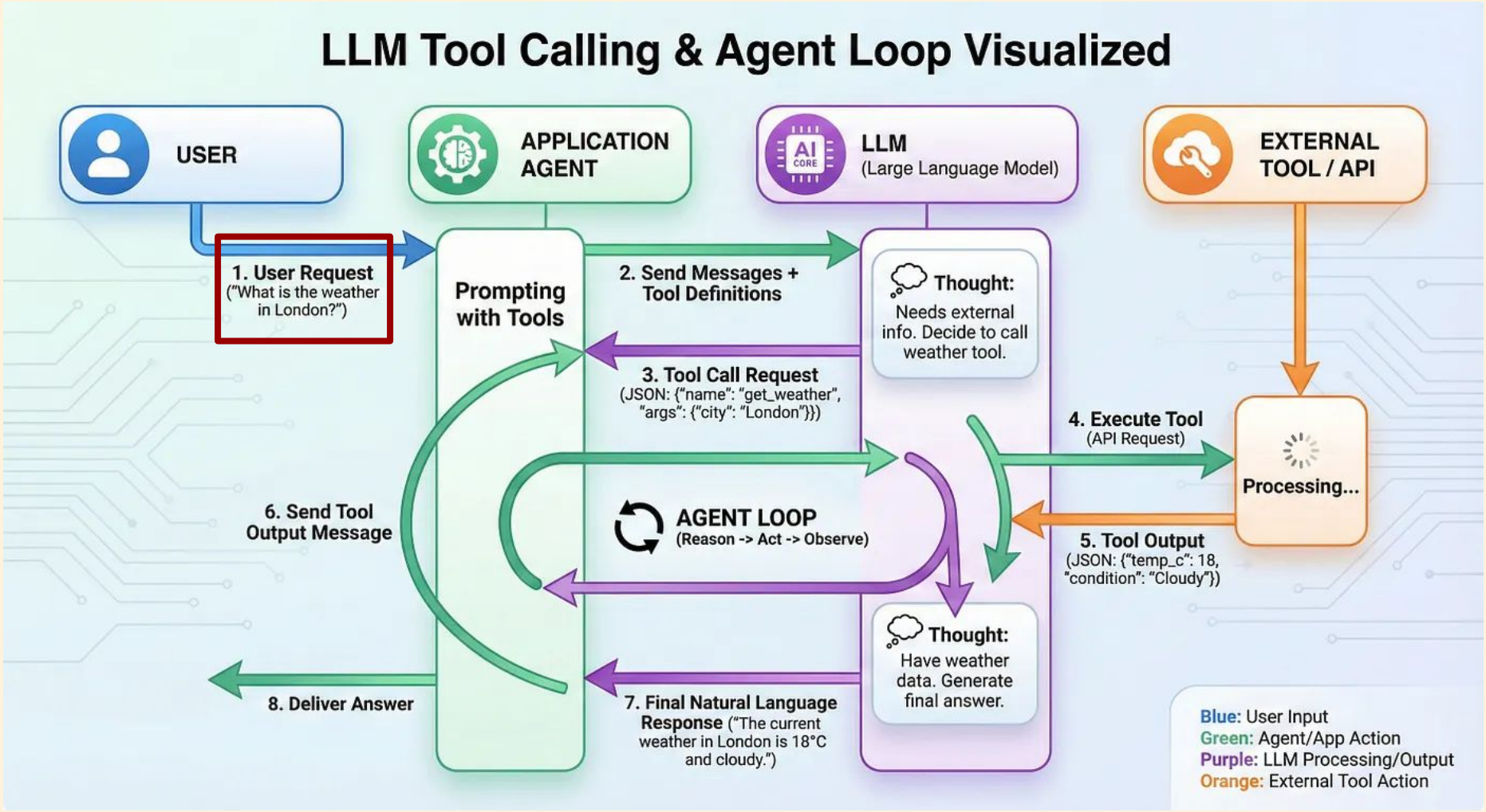
Important distinction:

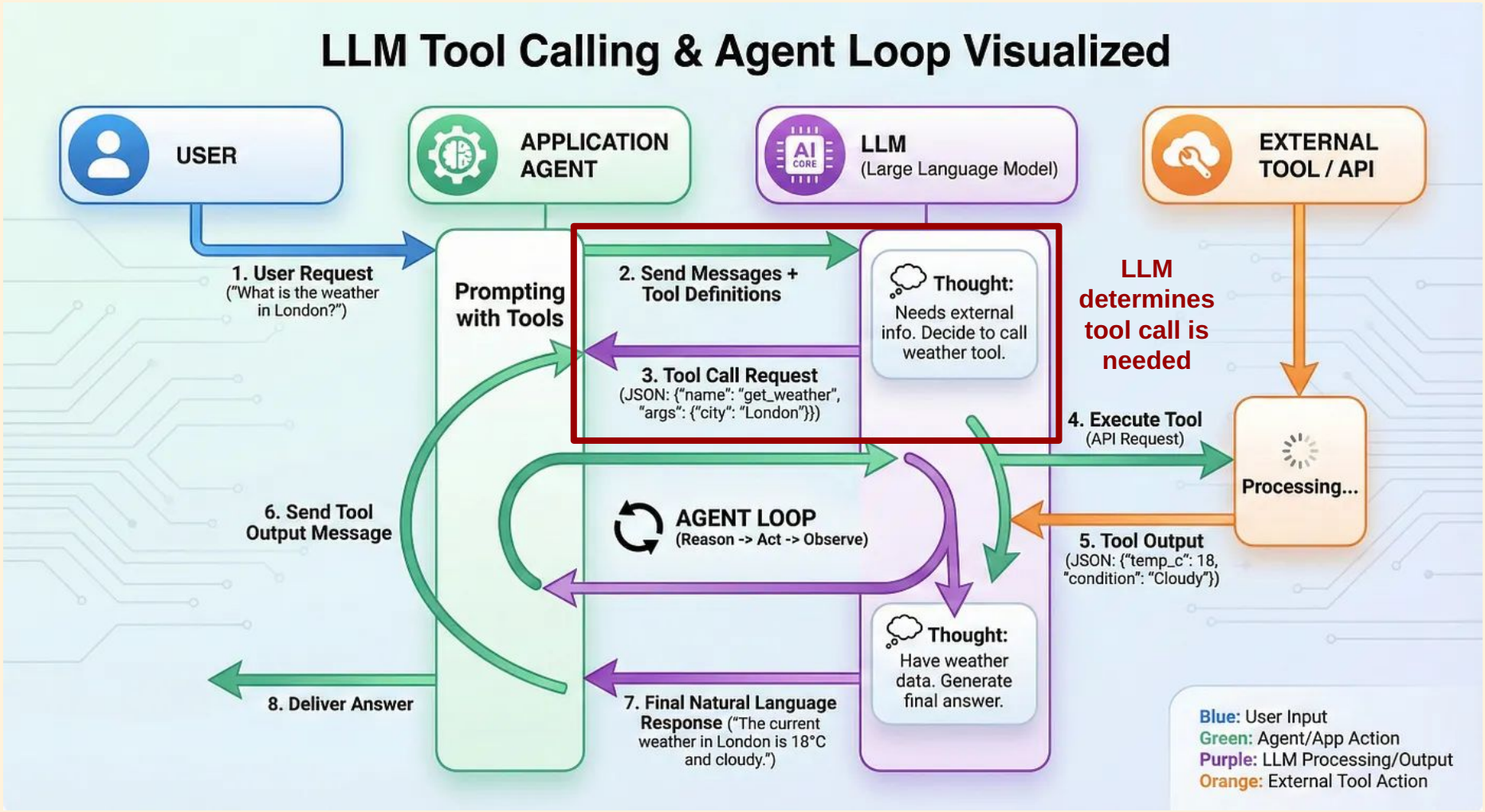
- Model chooses or recommends a tool call
- Application runtime executes the tool call
- *Model does not directly execute code or APIs*

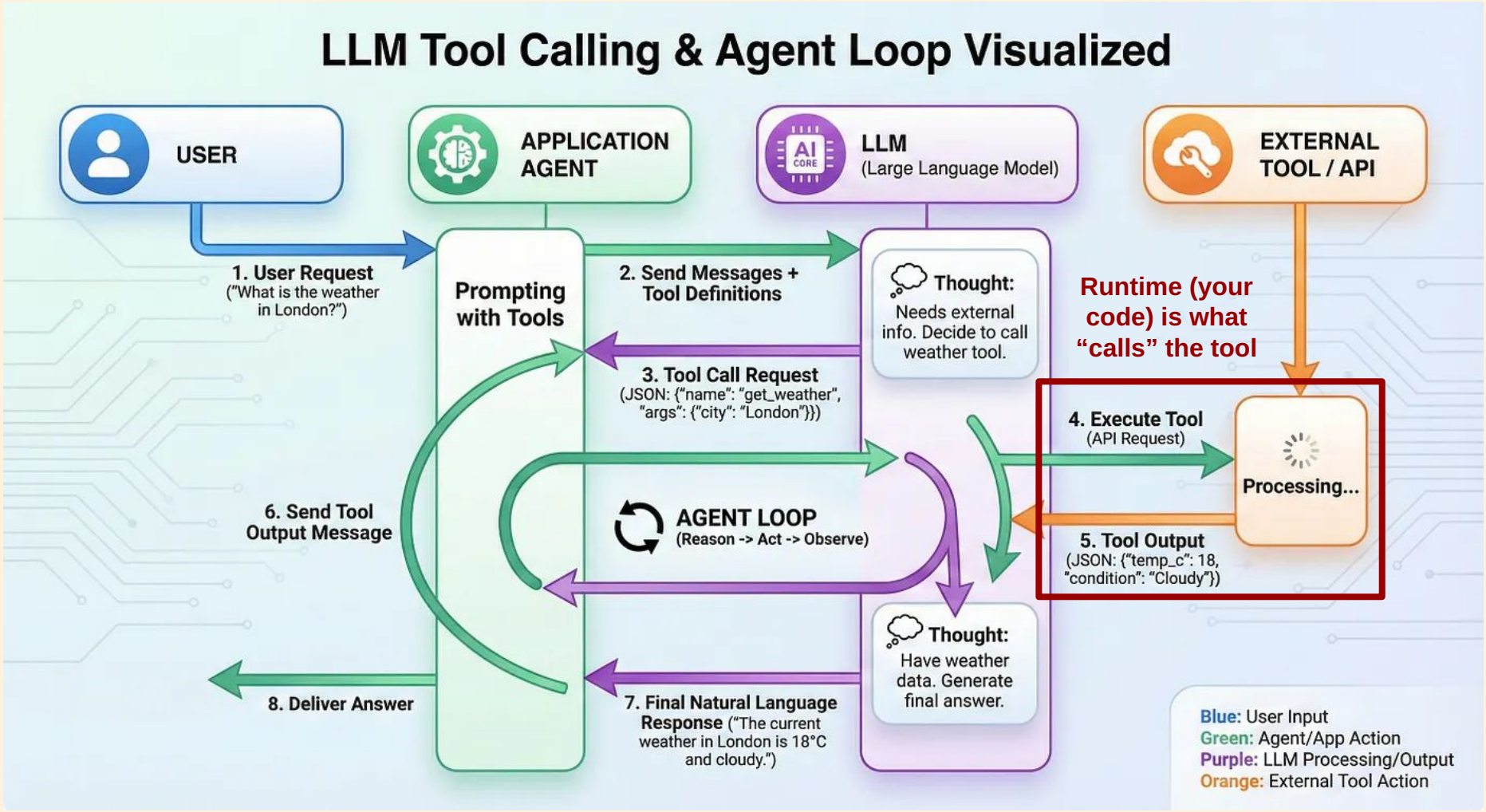
Tool calling loop

1. User asks for something
2. Model decides whether a tool is needed
3. Model emits tool name + arguments
4. Runtime validates arguments
5. Runtime executes tool
6. Tool result returns to model
7. Model produces final answer or calls another tool

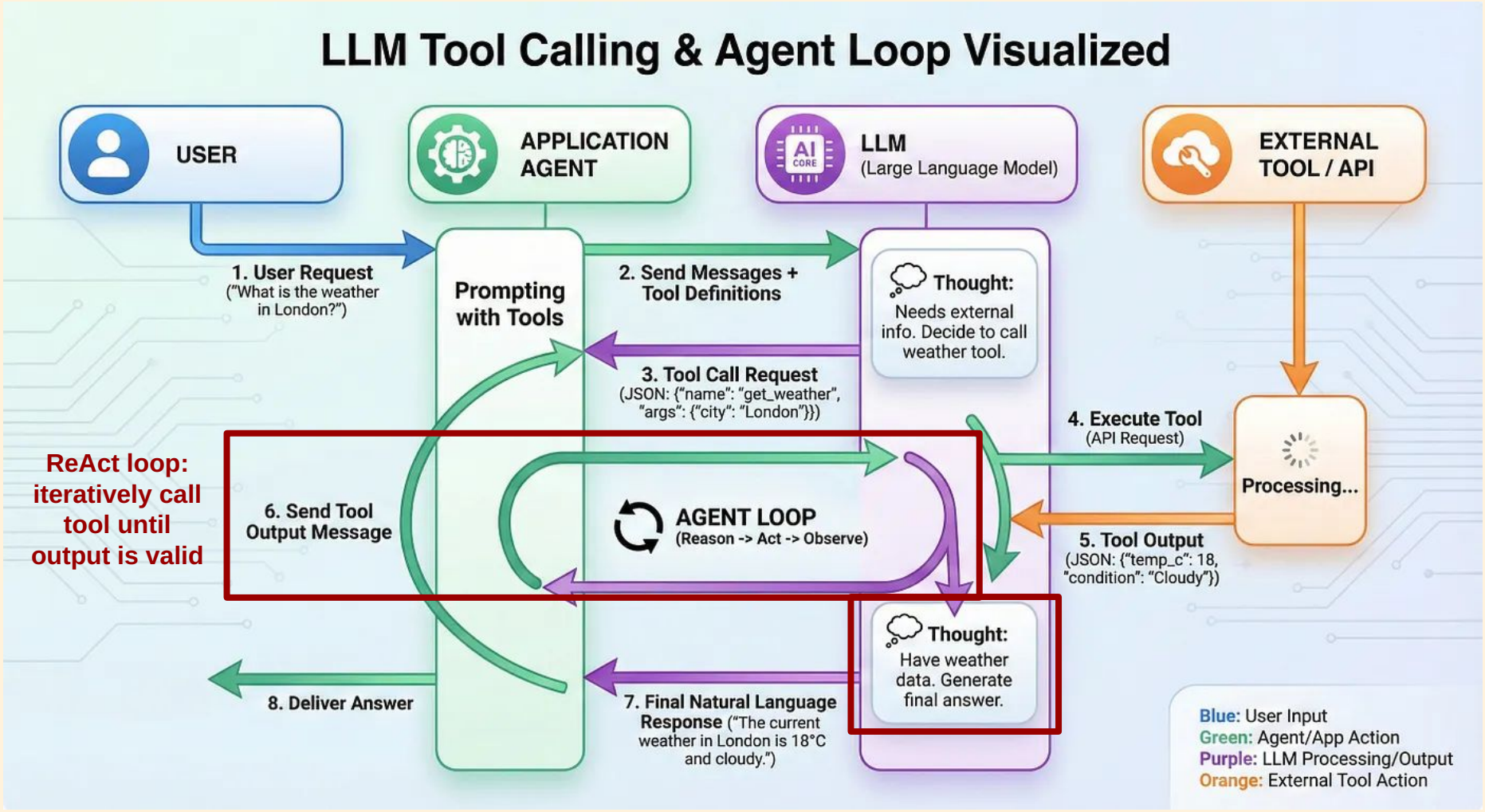
LLM Tool Calling & Agent Loop Visualized



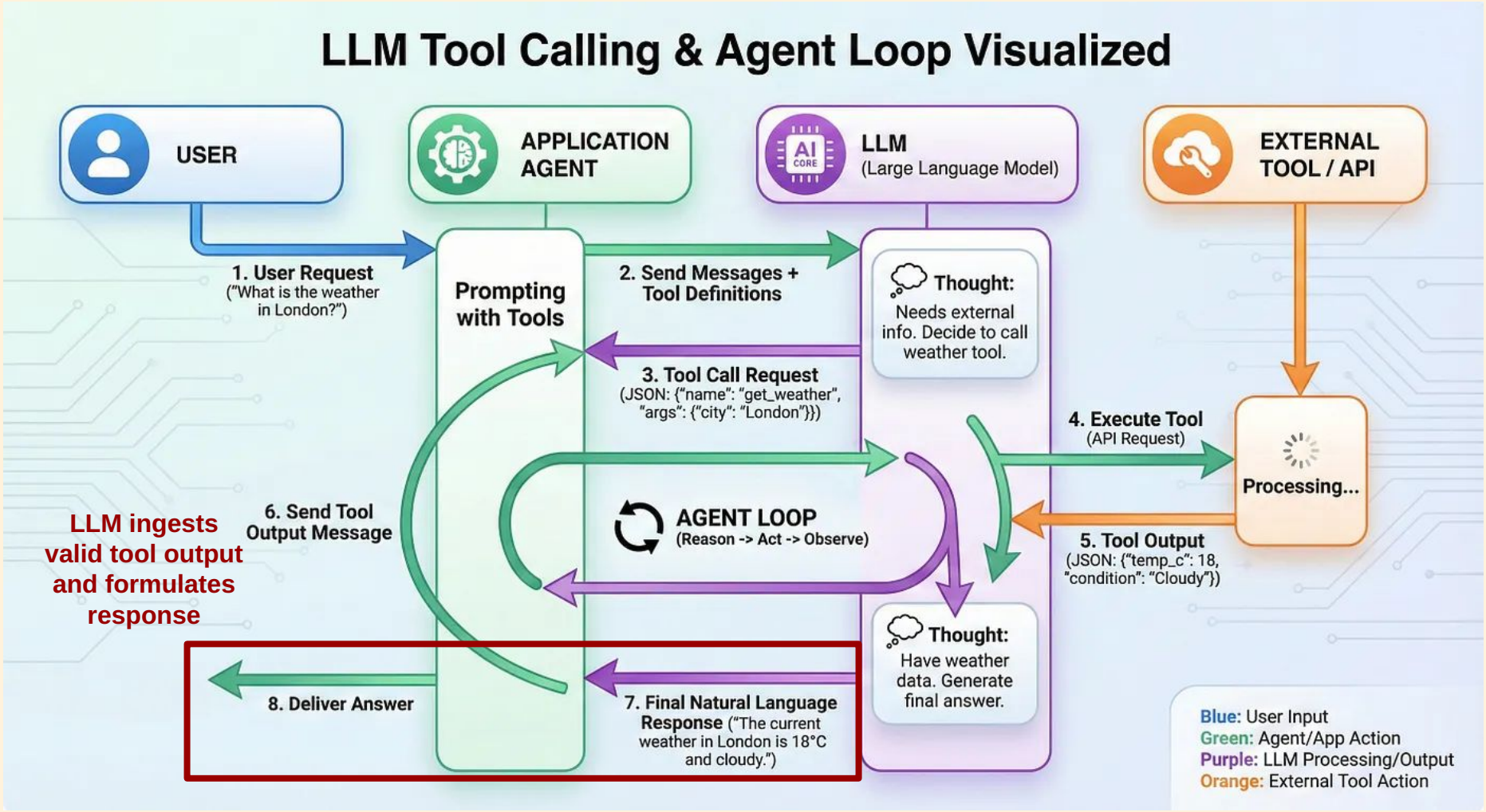

LLM Tool Calling & Agent Loop Visualized
USER
APPLICATION AGENT
LLM (Large Language Model)
EXTERNAL TOOL / API
1. User Request ("What is the weather in London?")
Prompting with Tools
2. Send Messages + Tool Definitions
Thought: Needs external info. Decide to call weather tool.
3. Tool Call Request (JSON: {"name": "get_weather", "args": {"city": "London"}})
LLM determines tool call is needed
4. Execute Tool (API Request)
Processing...
5. Tool Output (JSON: {"temp_c": 18, "condition": "Cloudy"})
AGENT LOOP (Reason -> Act -> Observe)
6. Send Tool Output Message
Thought: Have weather data. Generate final answer.
7. Final Natural Language Response ("The current weather in London is 18°C and cloudy.")
8. Deliver Answer
Blue: User Input
Green: Agent/App Action
Purple: LLM Processing/Output
Orange: External Tool Action


LLM Tool Calling & Agent Loop Visualized
USER
APPLICATION AGENT
LLM (Large Language Model)
EXTERNAL TOOL / API
1. User Request ("What is the weather in London?")
Prompting with Tools
2. Send Messages + Tool Definitions
Thought: Needs external info. Decide to call weather tool.
3. Tool Call Request (JSON: {"name": "get_weather", "args": {"city": "London"}})
Runtime (your code) is what "calls" the tool
4. Execute Tool (API Request)
Processing...
5. Tool Output (JSON: {"temp_c": 18, "condition": "Cloudy"})
AGENT LOOP (Reason -> Act -> Observe)
6. Send Tool Output Message
Thought: Have weather data. Generate final answer.
7. Final Natural Language Response ("The current weather in London is 18°C and cloudy.")
8. Deliver Answer
Blue: User Input
Green: Agent/App Action
Purple: LLM Processing/Output
Orange: External Tool Action

LLM Tool Calling & Agent Loop Visualized



LLM Tool Calling & Agent Loop Visualized



What counts as a tool?

A tool can be:

- Generic Python function
- Database query function
- Web search endpoint
- Calendar or email action
- Code executor
- Browser or shell interface
- Remote MCP-exposed capability
- Another LLM!

A “tool” is not defined by complexity. It is defined by the fact that the model can request it.

Benefits of Tool Calling

Tool calling enables:

- Access to current information
- Access to proprietary/private information
- Exact computation
- Structured changes to environment state
- Decomposition of complex tasks

Overall, tools improve agent reliability + accuracy

- Some tasks are better handled by deterministic systems than by text generation alone

Tool Calling vs Retrieval

Retrieval

- usually fetches information
- often read-only
- often vector or keyword search

Tool calling

- broader category
- includes retrieval, computation, and action
- may return data or change environment state

Tool Schemas

Typical schema includes:

- tool name
- description
- Parameters
- parameter types
- required fields
- constraints

Descriptions matter because the model uses them to infer when to call the tool, what arguments to provide, and which tool is best (if multiple exist).

Good Tool Descriptions

Bad:

```
search(query)
```

Better:

```
Searches internal documents for factual answers. Use when the
user asks for policy, docs, or recent project context. Do not use
for sending messages or editing files.
```

```
search_documents(query: string, sources: list[str])
```

Common tool failure modes

At the tool level:

- LLM generates wrong tool arguments
- Lack of robust argument validation
- LLM may hallucinate tool names or parameters

At the system level:

- Wrong tool selection
  - Tools too granular → too many tools to parse through
  - Tools not granular enough → LLM may erroneously call tool
- Tools might open up security vulnerabilities

From tools to frameworks

Why not just call functions manually?

Because production agents need:

- Retries upon failure
- Branching
- Evaluation
- Human approval points
- Multi-step workflows
- etc.

Frameworks provide structure + organization when implementing complex agents!

Today’s Lecture

1. Tool Calling
2. **Frameworks**
3. MCP

What Is an Agentic Framework?

An agentic framework is the foundational structure that governs how agents communicate, coordinate, reason, and act across systems.

It provides the structure for:

- Reasoning loops
- Message passing
- Tool orchestration
- Memory/state
- Workflow coordination
- Governance and observability

Not all frameworks are the same

Different frameworks focus on different parts of an agentic system.

Some focus on single-agent setup, others focus on multi-agent communication or schema validation.

When deciding what framework to use, you must first figure out what layer of your system you need help structuring.

Components of an agentic system

Think of an agentic system as having 4 layers:

Model: GPT-5, Gemini, Claude
Tooling: function calling, provider tools, MCP
Runtime: code that governs and executes the system
Infrastructure: databases, memory store, logging

Different frameworks focus on different layers

Framework 1: LangChain

LangChain: Best when you want to get an agent working quickly

- A fast way to build an LLM app with tools
- Gives you reusable building blocks
- Good for building one agent and its capabilities

Use it when:

- You want to prototype quickly
- You need lots of integrations
- You do not want to build all the “plumbing” yourself

Framework 2: LangGraph

LangGraph: Best when the agent is really a workflow

- Useful for agents that are less about “chatting” and more goal-focused
- More about steps, state, and transitions
- Useful when the system must pause, branch, or recover

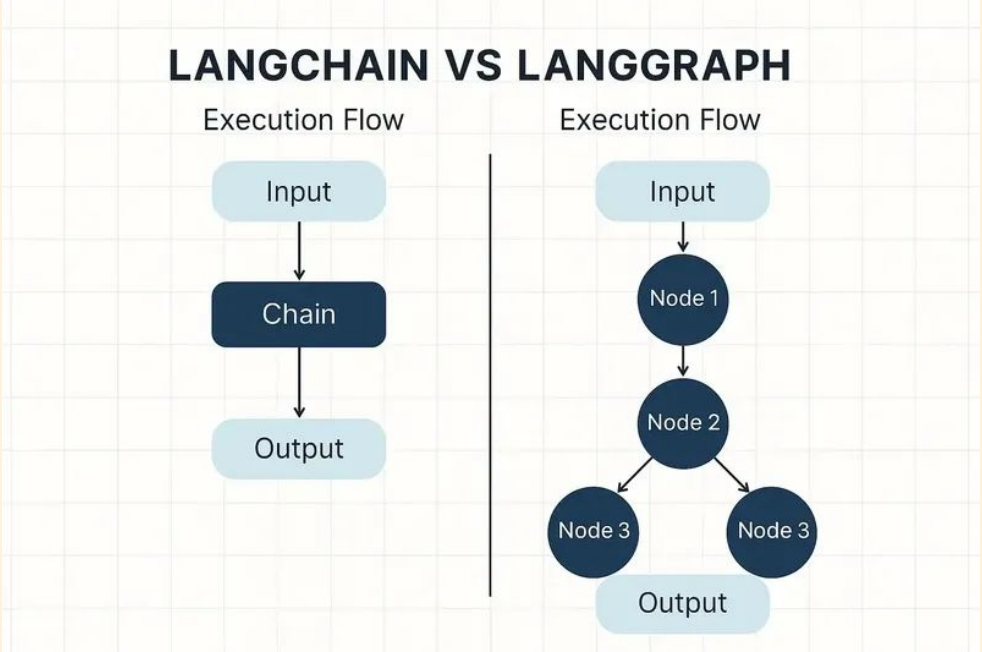
Use it when:

- The agent has multiple stages
- You need retries or approval steps
- The system may run for a long time

LangChain vs LangGraph

LangChain: the agent is the main object. Useful for managing sequential execution

LangGraph: the workflow is the main object. Useful for system governance.



Framework 3: CrewAI

CrewAI: Best for a coordinated team of agents

- The system contains multiple agents
- Agents have different roles for different tasks
- More like an organization than a single assistant

Use it when:

- Division of labor makes the task clearer
- You want multi-agent collaboration
- One agent is becoming overloaded

Framework 4: AutoGen

AutoGen: Best when agent-to-agent interaction is the focus

- Agents talk to each other
- They delegate, critique, and refine
- The conversation between agents is part of the design

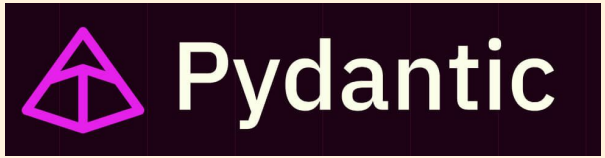
Use it when:

- Collaboration between agents is central
- Back-and-forth reasoning is important

Framework 5: PydanticAI

PydanticAI: Best when you want strong types and validation

- More disciplined, structured Python engineering
- Strong schemas for tool inputs and outputs
- A good fit for controlled systems



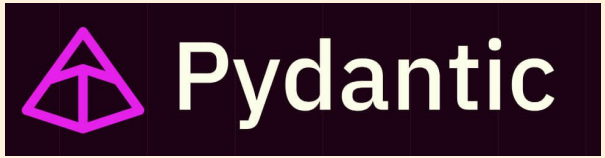
Use it when:

- Validation matters a lot
  - (e.g. output must adhere to a specific JSON format)
- You want clear, typed interfaces

Framework 5: PydanticAI

PydanticAI: Best when you want strong types and validation

- More disciplined, structured Python engineering
- Strong schemas for tool inputs and outputs
- A good fit for controlled systems



Use it when:

- Validation matters a lot
  - (e.g. output must adhere to a specific JSON format)
- You want clear, typed interfaces

Today’s Lecture

1. Tool Calling
2. Frameworks
3. **MCP**

Model Context Protocol (MCP)

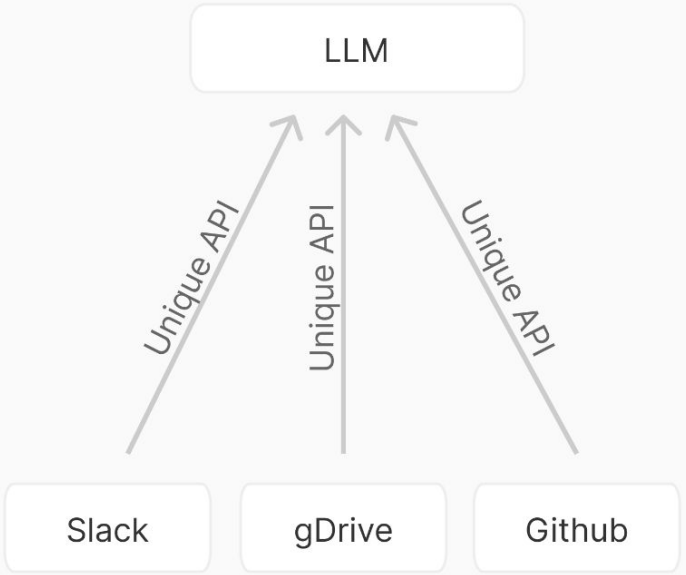
Before MCP:

- Every model + tool combo needed custom integration
- Integration burden scaled significantly as more tools were made

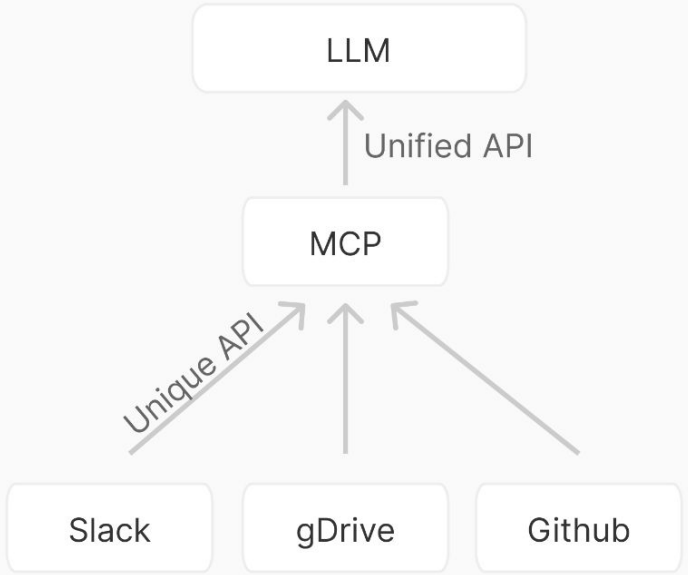
With MCP:

- Agents and tool providers speak a common “language”
- Tool providers only need to make 1 version of their tool that follows MCP
    - Not separate tools for GPT, Gemini, Claude, etc.

Before MCP



After MCP



What is MCP?

A standardized protocol for connecting AI agents to:

1. Tools
2. Resources
3. Prompts
4. External systems

MCP is a clear, standard way to extend models with additional tools and knowledge.

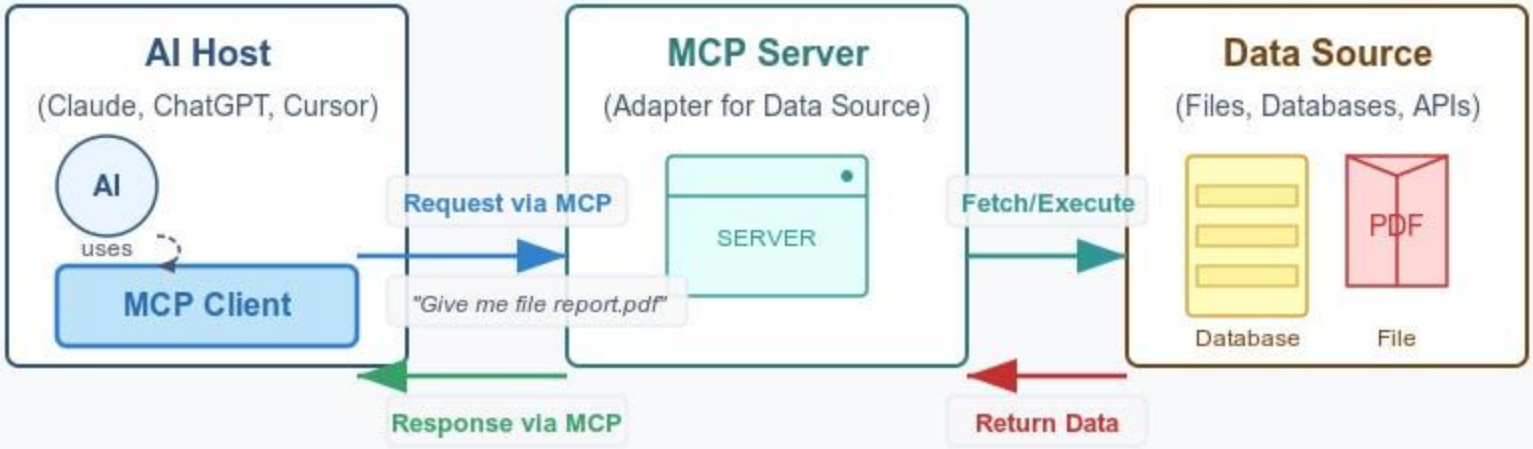
MCP Architecture

MCP client: the AI application or agent
MCP server: how to tools are exposed
Model: Underlying LLM that decides what tools to call

During an MCP Interaction:

1. Client sends user question to model
2. Model analyzes available tools
3. Client executes chosen tool through MCP server
4. Results return to model, model formulates answer

Model Context Protocol (MCP) Architecture



Model Context Protocol (MCP) Flow

The MCP Client translates AI requests into the standardized protocol format, communicates with MCP Servers, which then interact with external Data Sources.

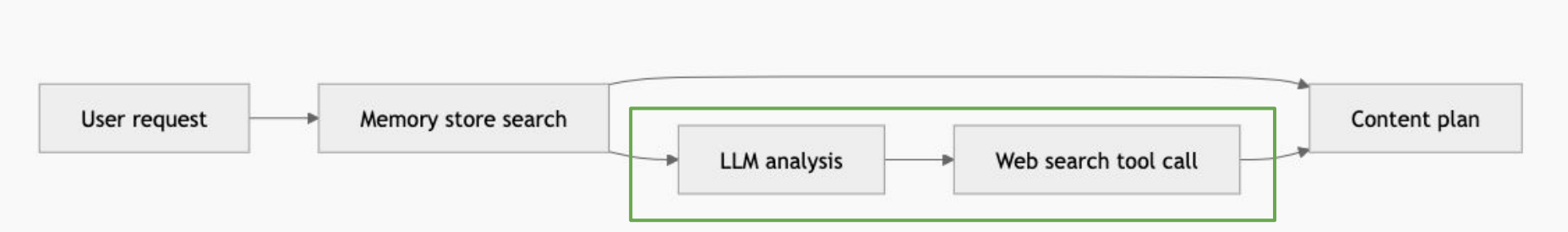
Build your celebrity agent a tool!

You had to build a memory store because your agent did not have access to a web calling tool…

How would you ensure your agent stays up-to-date with your celebrity?

Augment the memory store with a tool call!

Vibe code a web search tool



If the memory store search does not retrieve relevant info, perform a web search to get the needed information.

https://app.tavily.com/ Sign up with your student email for 1,000 free web searches. You can search based on a query term, get the top urls, and read the web pages.