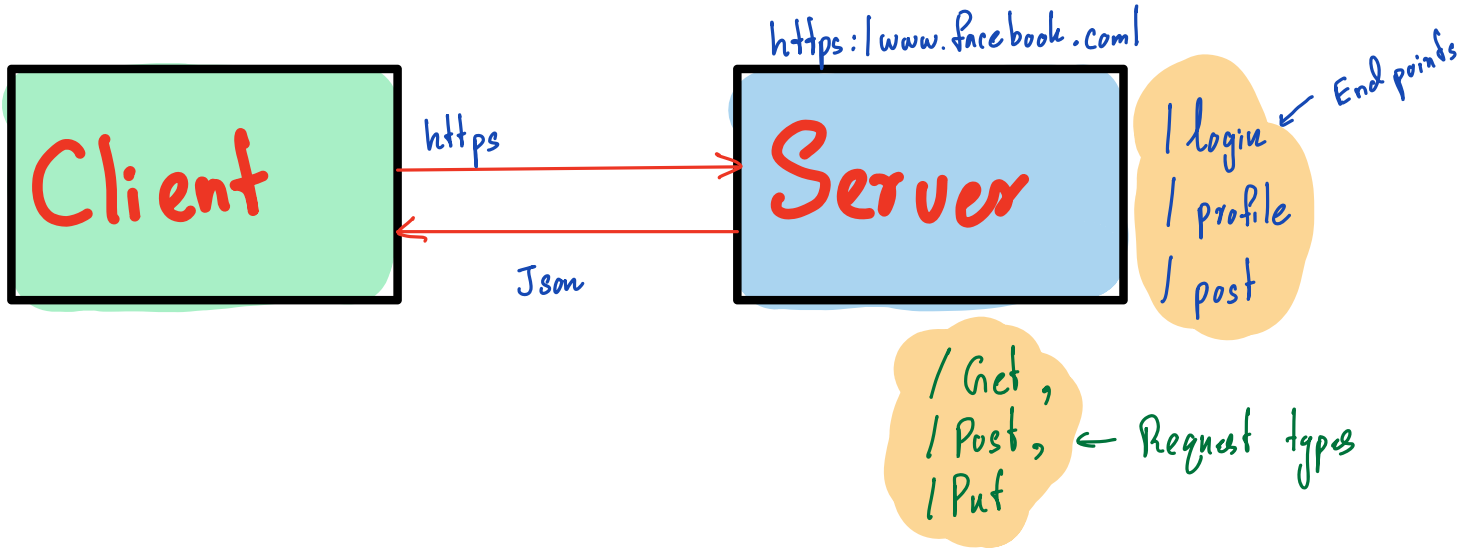
Model Context Protocol (MCP).

What is Model Context Protocol (MCP)?

MCP is an open protocol that standardizes how applications provide context to LLMs. Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect your devices to various peripherals and accessories, MCP provides a standardized way to connect AI models to different data sources and tools.

Idea behind Protocals (website)



- **Standardized Communication:** Protocols (like HTTPS) establish a common set of rules or a "language" for communication between web clients (browsers) and web servers.

- **Medium for Interaction:** They act as the medium through which a client can request information from a server and receive a response.

- **Request/Response Cycle:** Clients send various types of requests (e.g., GET to retrieve data, POST to submit data) to the server.

- **Server Processing:** The server processes these requests and delivers the appropriate response (often in a structured format like JSON).

- **Enabling Services:** Protocols allow servers to expose various services (e.g., login, live classes, courses) that clients can access by hitting specific URLs.

- **REST API as Common Language:** REST API is highlighted as a common architectural style that uses HTTP protocols to enable communication and access to backend services, providing a standardized way for different applications to interact.

- **Interoperability:** The core idea is to enable different systems (client and server) to understand and interact with each other seamlessly, regardless of their underlying implementation, by adhering to a shared protocol.

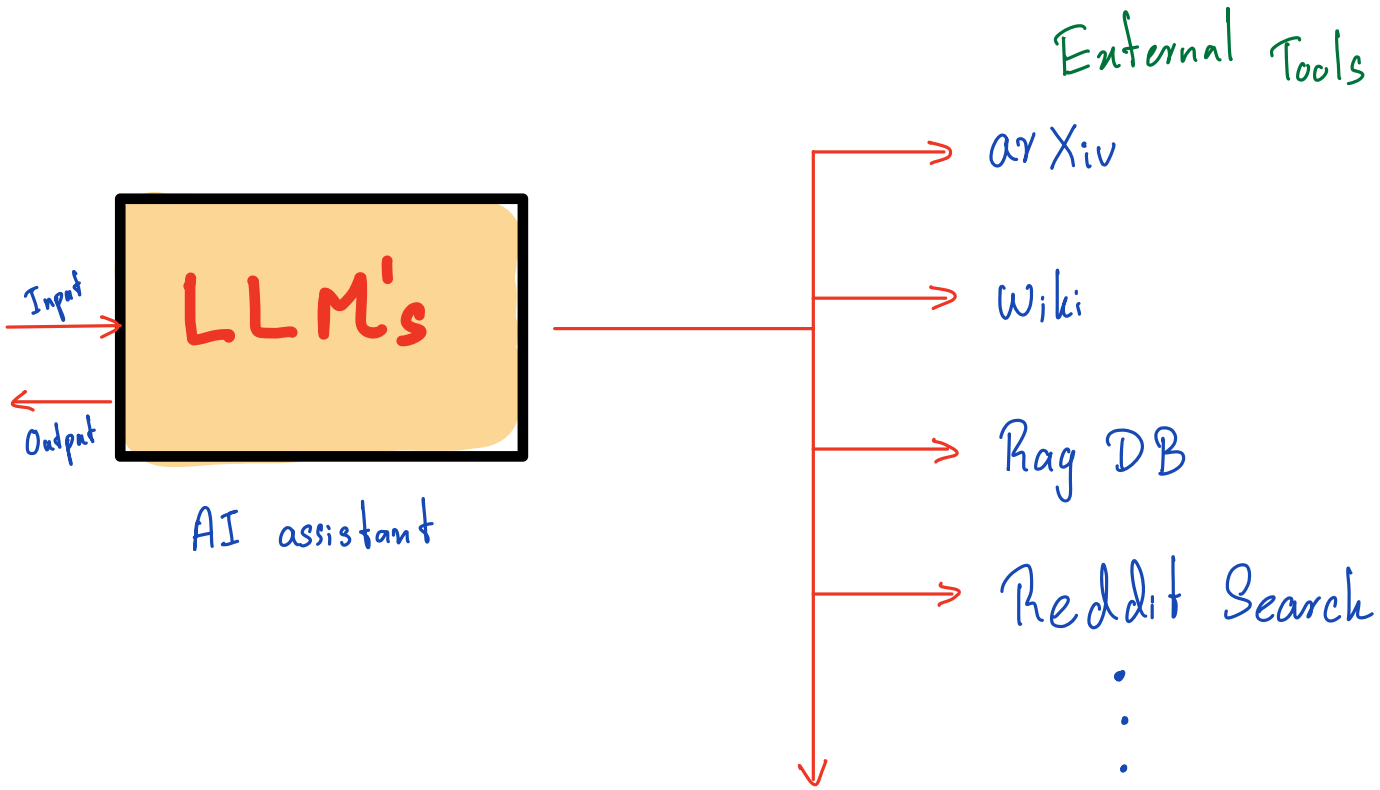
1) Generative AI Models

- AI models designed to create new content (text, images, audio, etc.) that is similar to the data they were trained on.
- They learn patterns and distributions from vast amounts of training data.
- Often referred to as "Generative AI" because they generate content in response to input.
- Examples include GPT-3.5, Google Gemini, Llama models.
- Initially, their primary function was to take input and produce a generated output.

Downsides:

- **Limited Scope:** Primarily focused on content generation and not designed for complex actions or real-world interactions.
- **Lack of External Knowledge:** Only know what they were trained on; cannot access real-time information or external resources directly.
- **Hallucinations:** Can generate factually incorrect or nonsensical information that sounds plausible.
- **Bias:** May reflect and amplify biases present in their training data.
- **Resource Intensive:** Training requires significant computational power and energy.
- **Ethical Concerns:** Potential for misuse in generating fake news, deepfakes, or harmful content.

2) Multi Model Agents, LLM with Tools.



- These are AI systems (based on large language models) that connect with external tools or services (like search engines, APIs, databases, etc.) to handle complex tasks.

- "Multimodal" means they can understand and process different types of input (text, image, speech, etc.).

- Instead of just answering questions from their training, they use tools to find real-time or external information.

Example:

If you ask an LLM: "What's the weather in California today?" — it uses a weather API tool to fetch live data, instead of just guessing from its training.

Downsides of Multimodal Agents / LLMs with Tools:

- **Manual Tool Integration Needed**
  - Each tool (like a weather API or PDF reader) must be manually integrated with custom code.
  - Developers need to write wrappers, define inputs/outputs, and handle responses.

- **Hard to Maintain at Scale**
  - If you integrate 10+ tools, maintaining and updating each one becomes time-consuming and error-prone.

- **Breaks on API/Tool Changes**
  - If a tool or API changes its structure or behavior, the integration can break and must be manually updated.

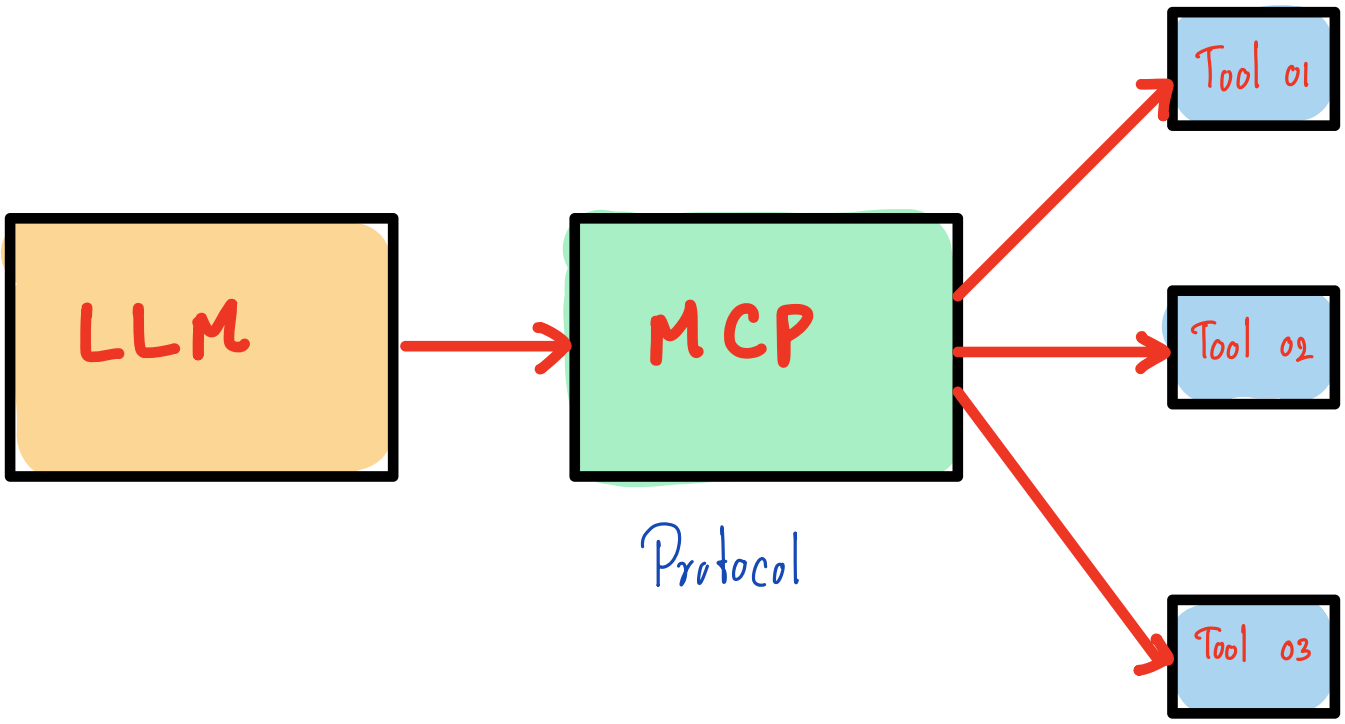
- **Complex Architecture**
  - Requires a lot of infrastructure setup — clients, servers, tool handlers, and communication protocols.

- **Latency Issues**
  - Calling external tools can add delays to response times, especially if multiple tools are used per query.

- **Security & Privacy Risks**
  - Data passed to tools (especially 3rd-party ones) could expose sensitive information.

- **Tool Compatibility Issues**
  - Not all tools work well with every LLM — integration may require format adjustments or tool-specific logic.

3) LLMs + MCP



How LLMs + MCP Solve This Problem

The Model Context Protocol (MCP) solves these scaling issues by acting as a universal communication layer between LLMs and external tools.

Key Solutions from MCP:

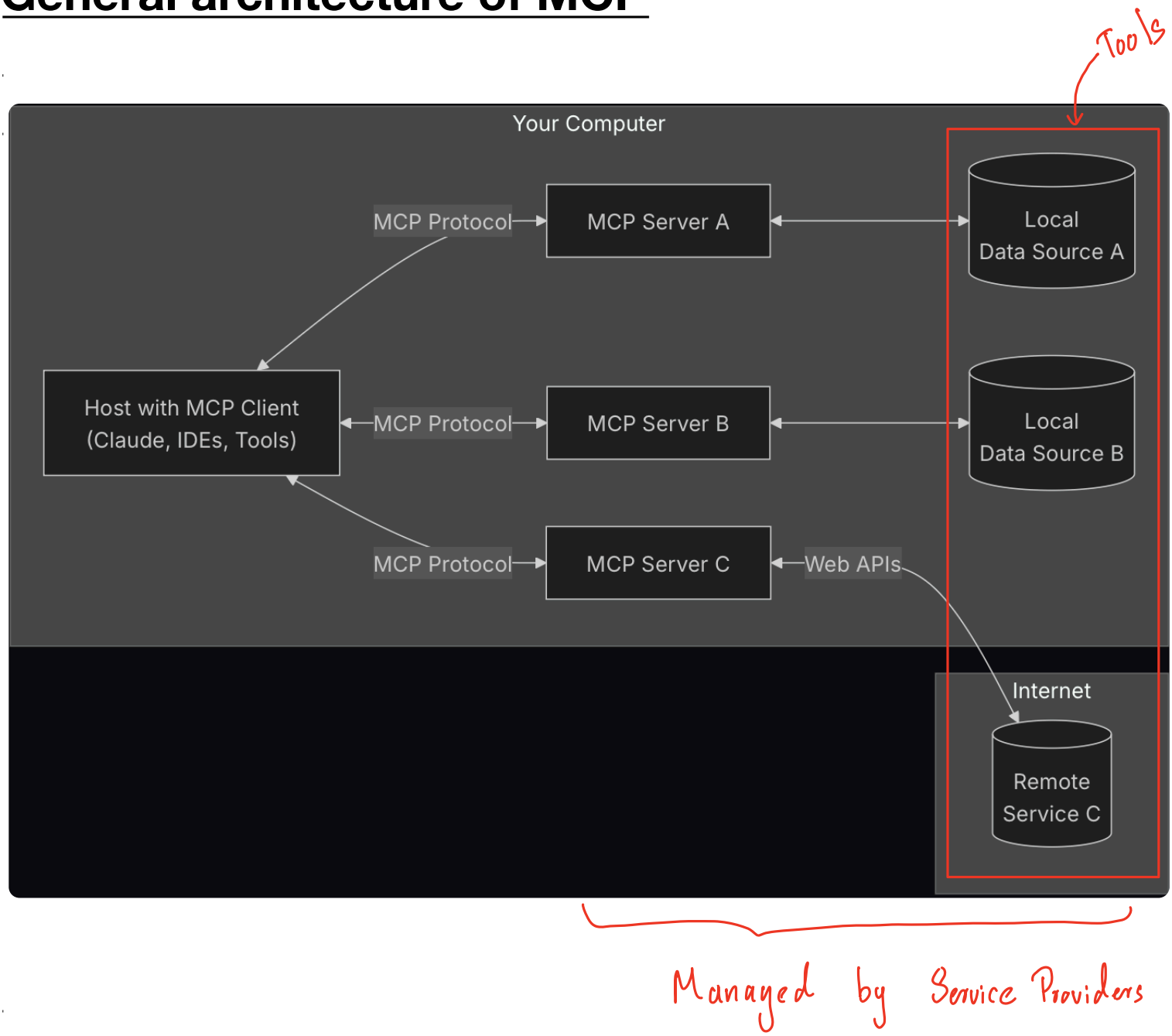
- **Standardized Communication**
  - MCP defines a common protocol (like a USB-C port for tools) that all tool providers follow.
  - No more writing custom code for each tool — just plug and play.

- **Plug-in Architecture**
  - Tools connect to MCP servers.
  - LLMs (via an MCP client) can interact with any compliant tool without needing custom integration code.

- **Centralized Updates**
  - Tool providers manage their own tool logic on their MCP servers.
  - Even if a tool updates, your LLM integration doesn't break — no code changes required on your side.

- **Smarter LLM Interaction**
  - The LLM queries MCP to discover available tools.
  - Then, based on the user's input, it decides which tool to use — all handled dynamically.

General architecture of MCP



- **MCP Hosts:** Programs like Claude Desktop, IDEs, or AI tools that want to access data through MCP
- **MCP Clients:** Protocol clients that maintain 1:1 connections with servers
- **MCP Servers:** Lightweight programs that each expose specific capabilities through the standardized Model Context Protocol
- **Local Data Sources:** Your computer's files, databases, and services that MCP servers can securely access
- **Remote Services:** External systems available over the internet (e.g., through APIs) that MCP servers can connect to

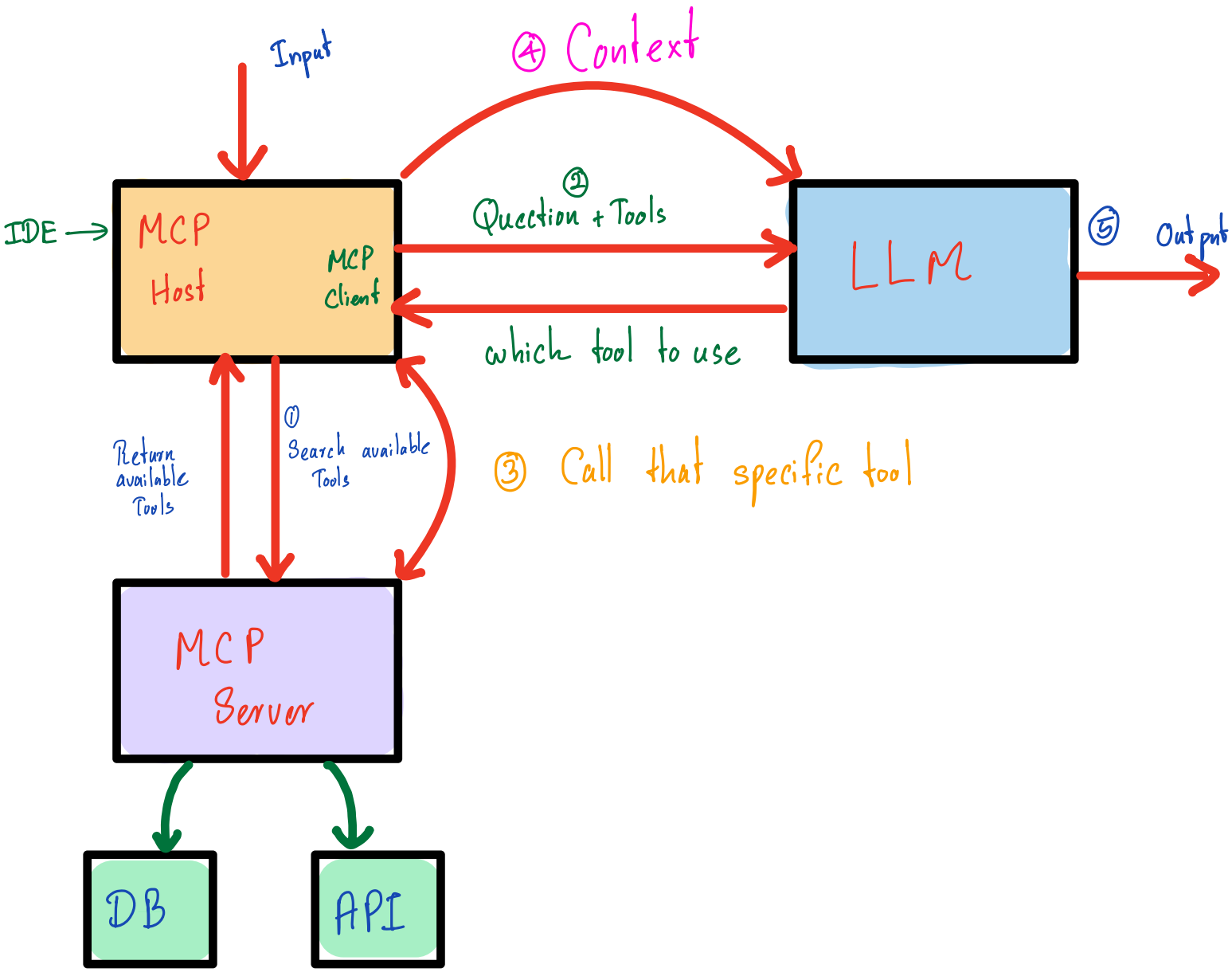
How Communication Happens .

1. User gives input (e.g., a question or task).
2. MCP Host (e.g., IDE) sends this input to the MCP Client.
3. Client sends the request to the MCP Server to fetch available tools.
4. MCP Host sends input + tool list to the LLM (AI assistant).
5. LLM decides which tool to use based on the input.
6. MCP Host then calls the selected tool via the MCP Server.
7. The tool executes the task, returns the response.
8. Response is sent back to the LLM, which uses it to generate a final answer/output.

Advantages Highlighted

- Tool discovery is dynamic – LLMs decide which tool to use at runtime.
- No custom code needed per tool – thanks to standardization.
- MCP Server and tools are maintained by providers, not by you.
- If a tool updates, you don’t need to change anything — integration stays intact.

Communication between components



Components Involved:

- MCP Host (e.g., VS Code, Cursor IDE)
- MCP Client (inside the Host)
- MCP Server (connected to tools like APIs, DBs)
- LLM / AI Assistant (OpenAI, Claude, etc.)
- Tool Providers (Weather API, RAG DB, etc.)

Communication Flow:

1. **User Input**
   - A user gives an input/question via the Host (e.g., asks: “What’s the weather in California?”).
2. **MCP Client Sends Input to MCP Server**
   - The MCP Host sends this input to the MCP Client, which then forwards it to the MCP Server.
3. **Tool Discovery**
   - The MCP Server returns a list of available tools/services (e.g., weather API, DB, search).
4. **Send Input + Tool List to LLM**
   - The Host sends the user’s input + list of tools to the LLM.
   - This allows the LLM to reason over the tools and select the best one to use.
5. **LLM Chooses a Tool**
   - The LLM decides which tool is most suitable for the task (e.g., it selects “weather API”).
6. **Tool is Called via MCP Server**
   - The Host calls the selected tool (e.g., weather API) via the MCP Client → MCP Server.
   - The tool executes the task and sends the result back to the MCP Server.
7. **Response to LLM**
   - The tool’s output/context is returned to the LLM.
8. **Final Output**
   - The LLM uses the tool’s result to generate a complete answer, which is shown in the Host interface.

Example Summary:

- If the LLM needs real-time weather data:
- It asks MCP for tool options.
- It chooses the weather tool.
- MCP calls the tool, gets the weather.
- LLM uses the weather to give a useful response.

Why This Matters:

- Communication is modular and dynamic.
- Tools can be plugged in and swapped out easily.
- No need to hard-code logic or update the client-side when tools change.