Model Context Protocol

The idea behind MCP

LLMs are great, but lack interactivity with the "outside" world

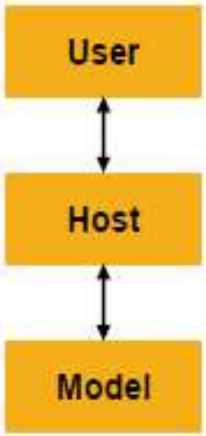
Function or tool calling enables AI / LLM apps to interact with the outside world, making them into agents… but all frameworks have different "tool-calling" mechanism

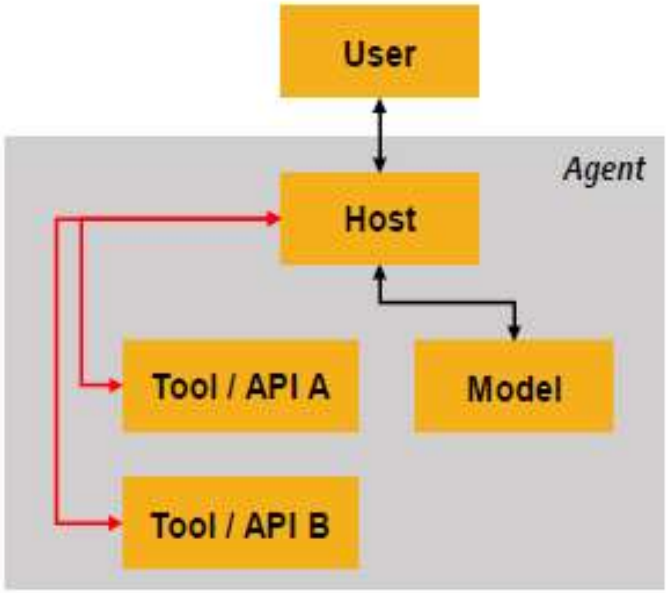
Many function or tool calling scripts were being separately developed to integrate with AI agents; many not endorsed by the underlying API provider

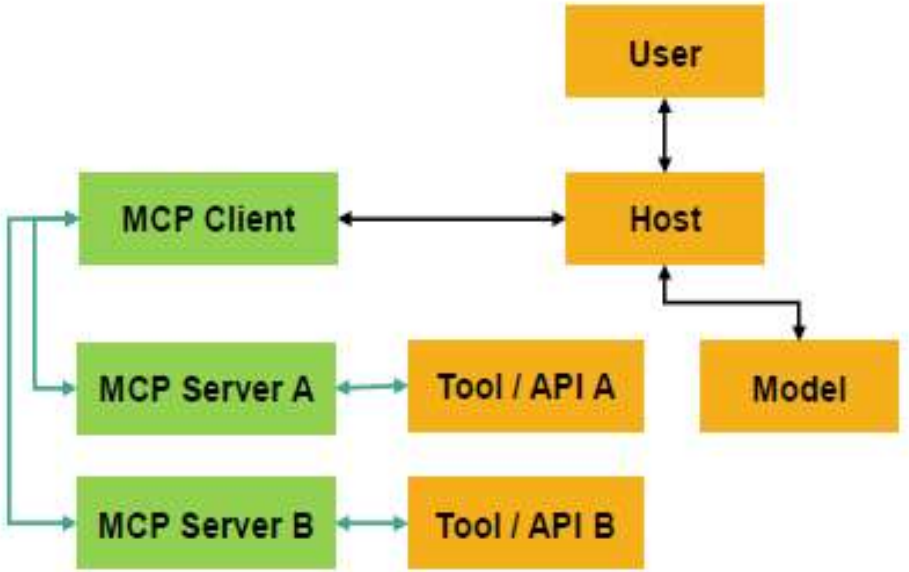
Most function or tool calling were running on servers, thereby making it impossible for LLMs to interact with your local machine

<u>The need:</u> A standardized mechanism (i.e., protocol) for AI systems (like LLMs, agents, etc.) to interact with external systems

History of Agentic AI Development / MCP







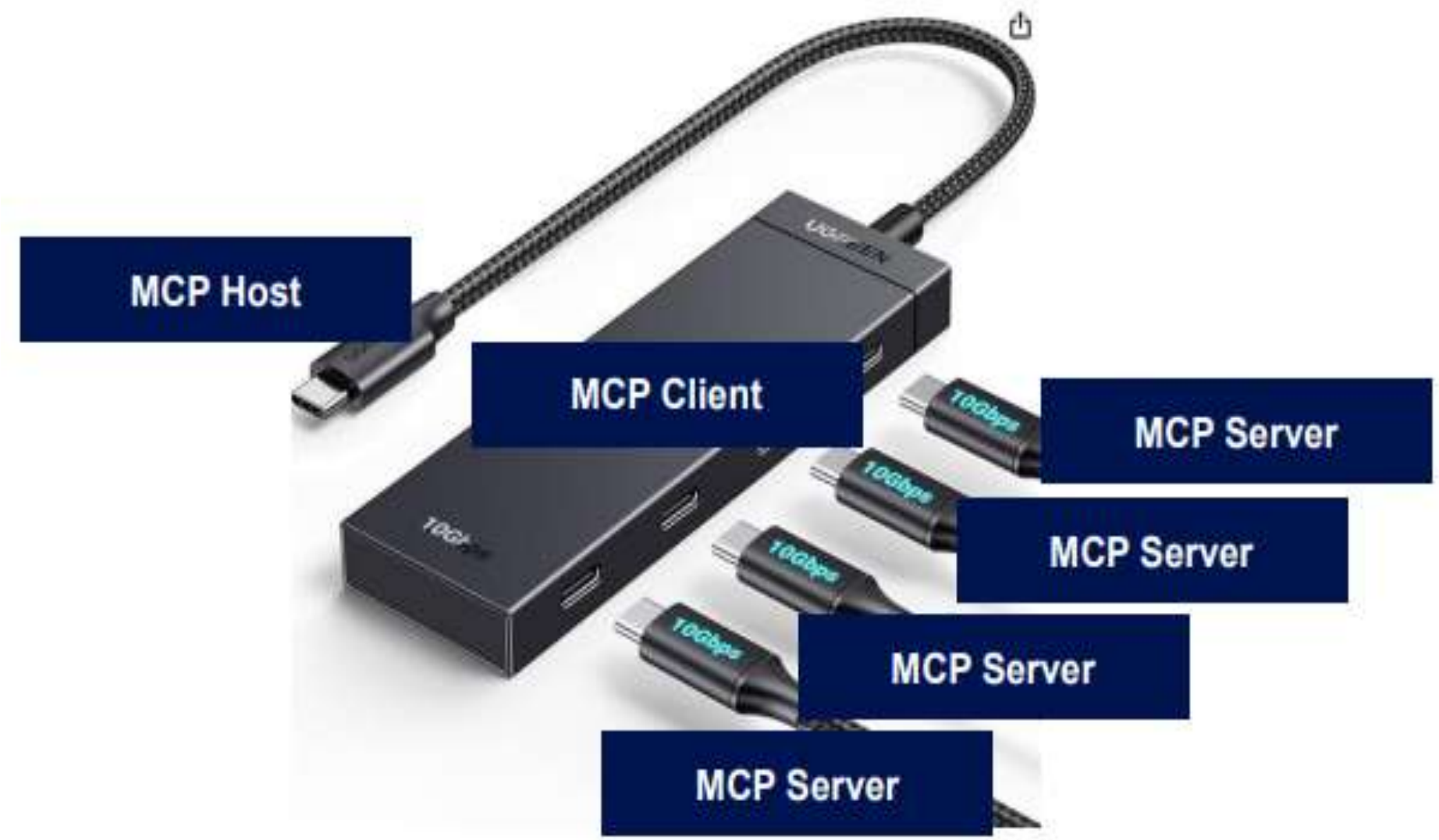
Why this is inefficient:

- Items in the red arrow had to be manually programmed every single time. Each time the "host" would change, or the tool / API would change, the connection needed to be updated and reprogrammed.
- Also, two separate projects that connected to "Tool A" would do it differently and it would be double the work. Why?

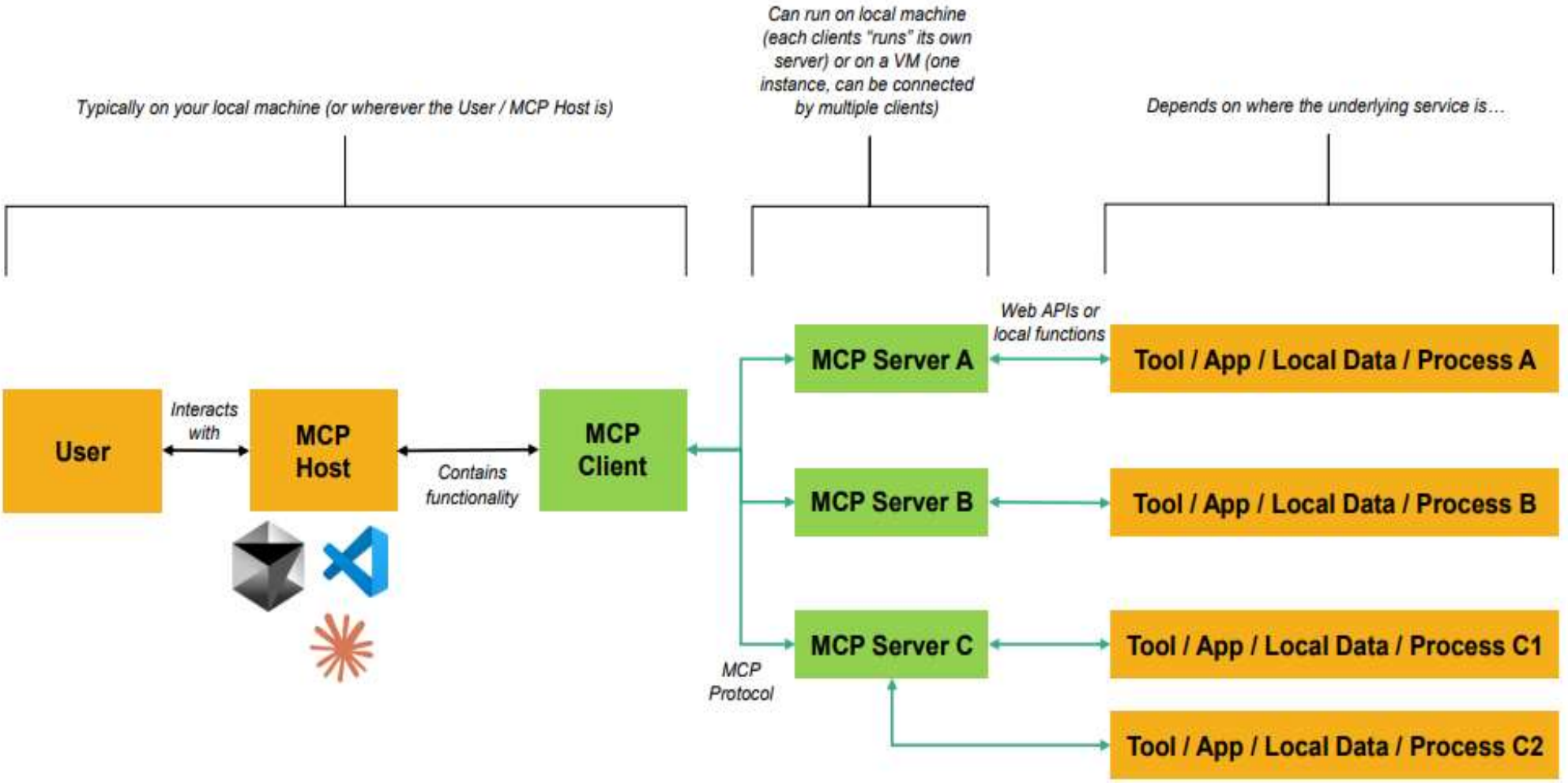
Why this is better:

- Items in the green arrow represents the standardized MCP protocol.
- Now, developers that create agentic AI apps only need to support to an "MCP client" and by doing so, automatically can connect to thousands of MCP servers with a few lines of JSON
- The green arrows are programmed once, updatable, standardized, and even supported by the underlying API providers (build once, run everywhere). Developers don't need to worry about the green arrow.

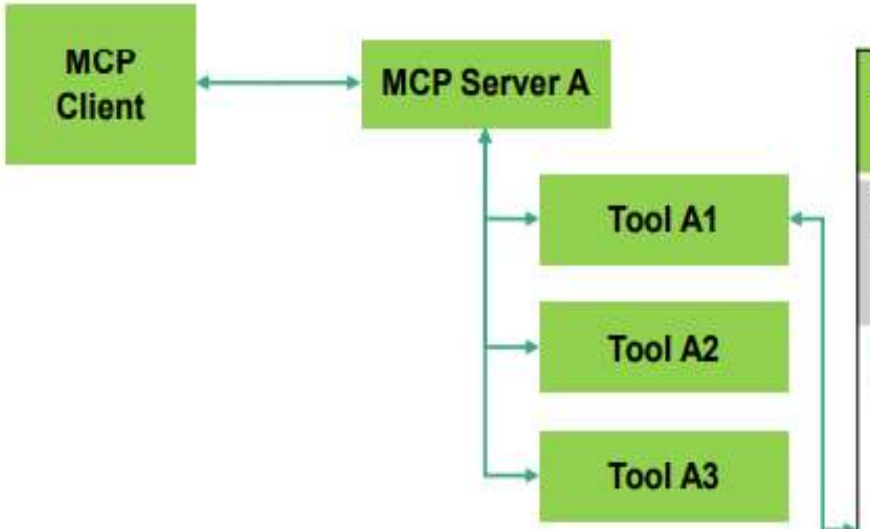
Practical Analogy - Think of MCP like a USB-C port



MCP Architecture

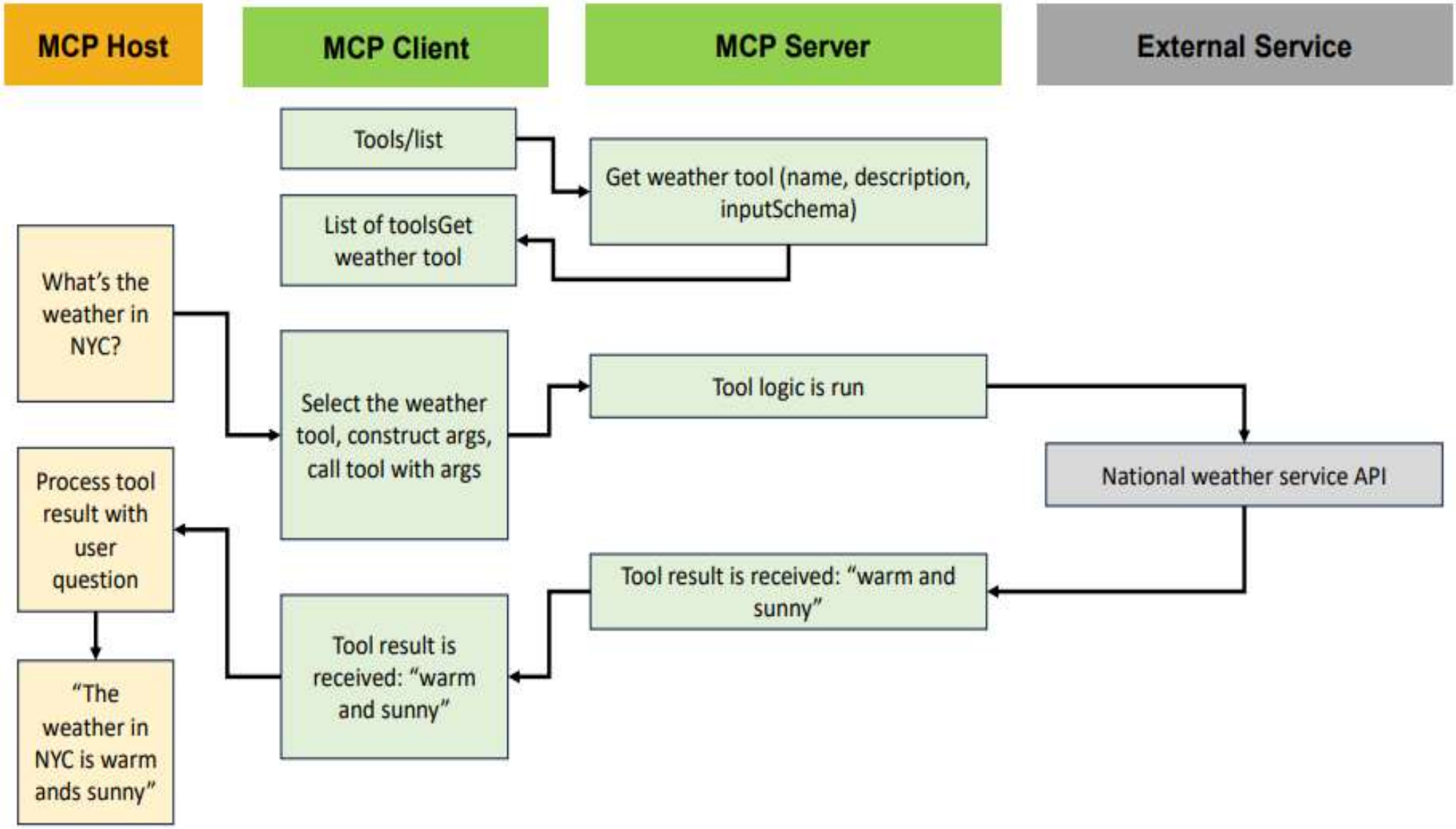


MCP Server Deep Dive

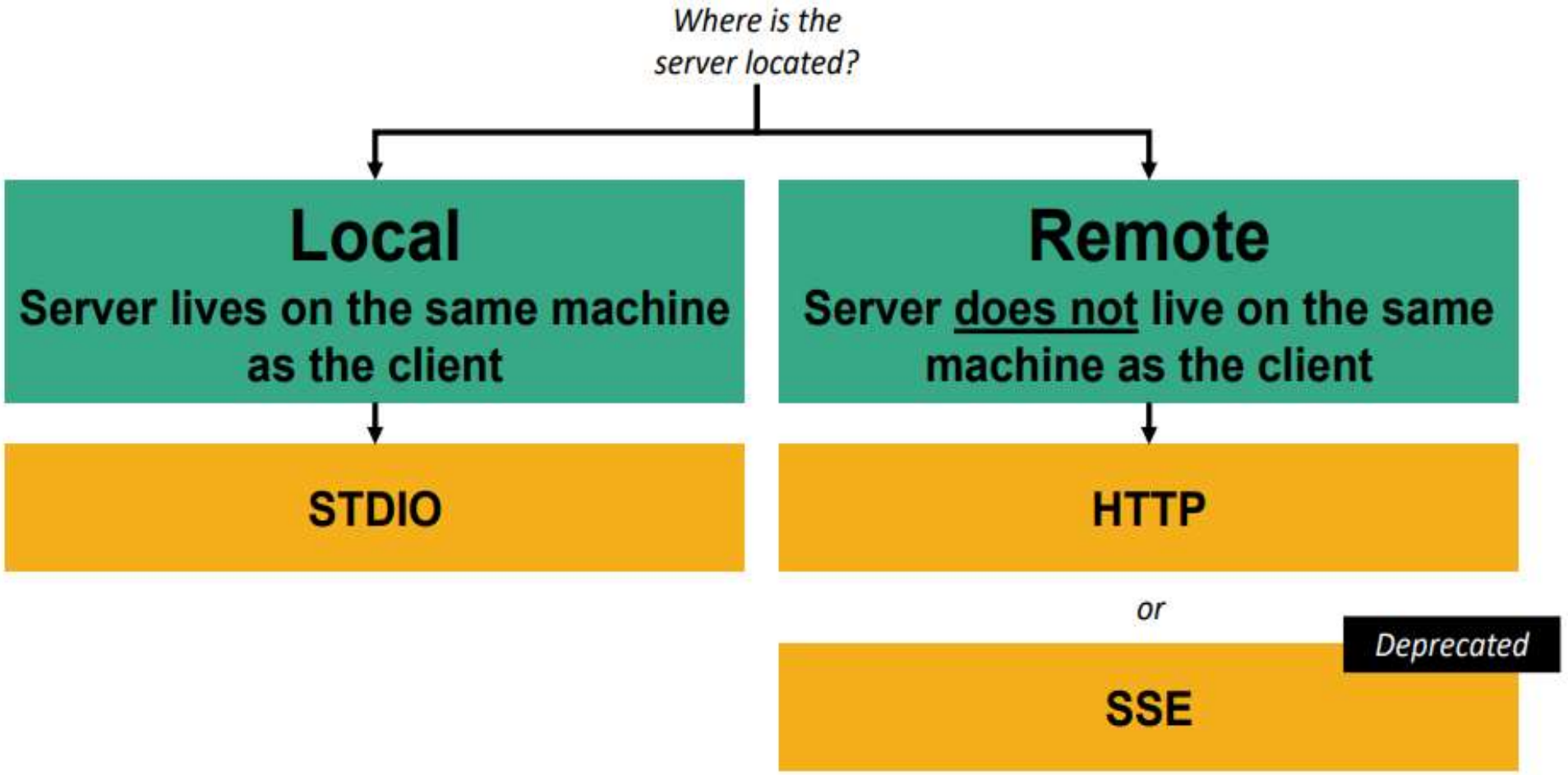


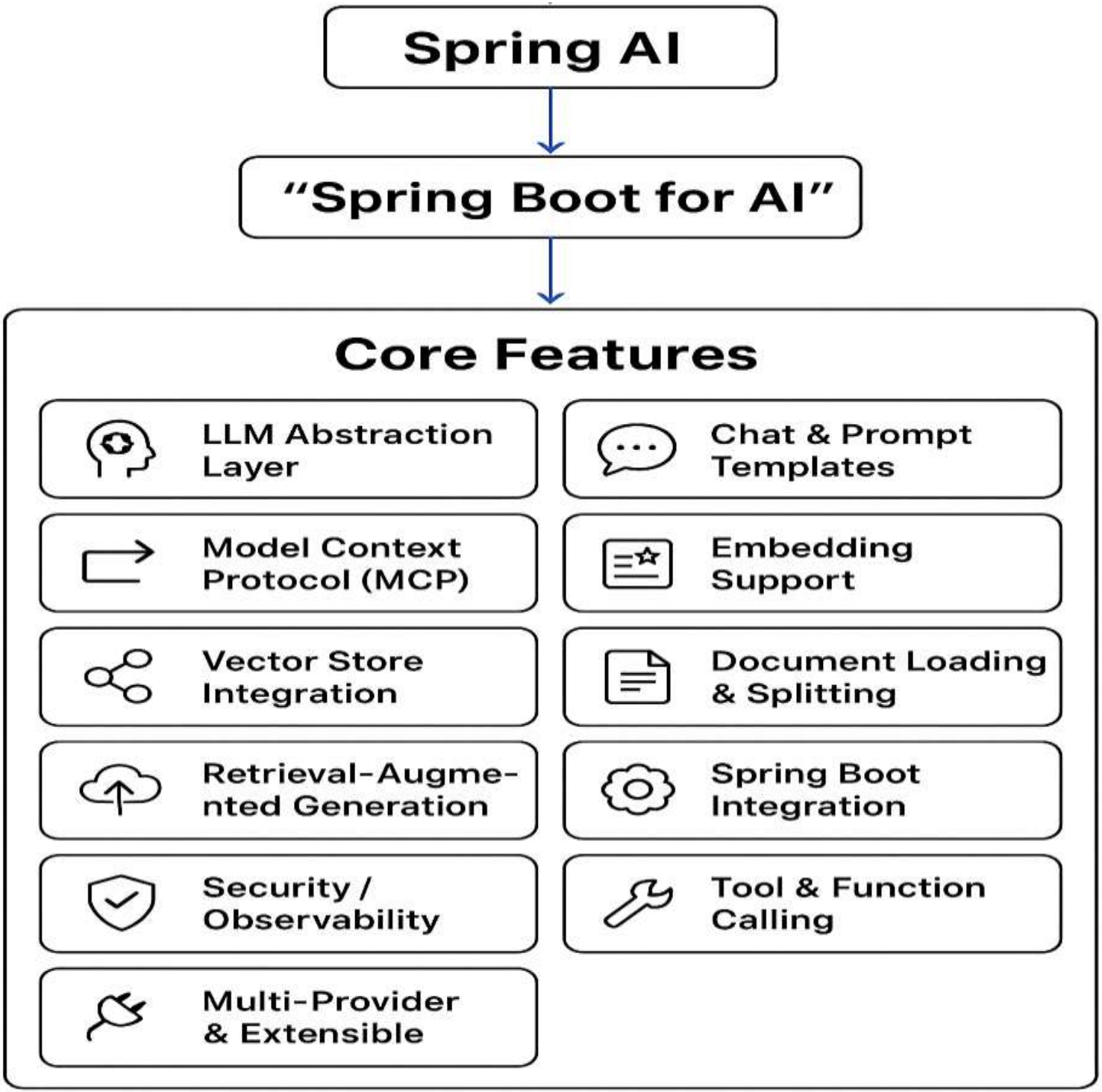
Parameter	Purpose	Example
Name	Name of the tool or function	"get weather"
Description	Description of what the tool or function does, along with what arguments are expected	"gets the weather given a location. Location should be a string"
Input Schema	Dictionary of arguments that the tool or function accepts	{"location": "string"}

MCP Client - Server Communication

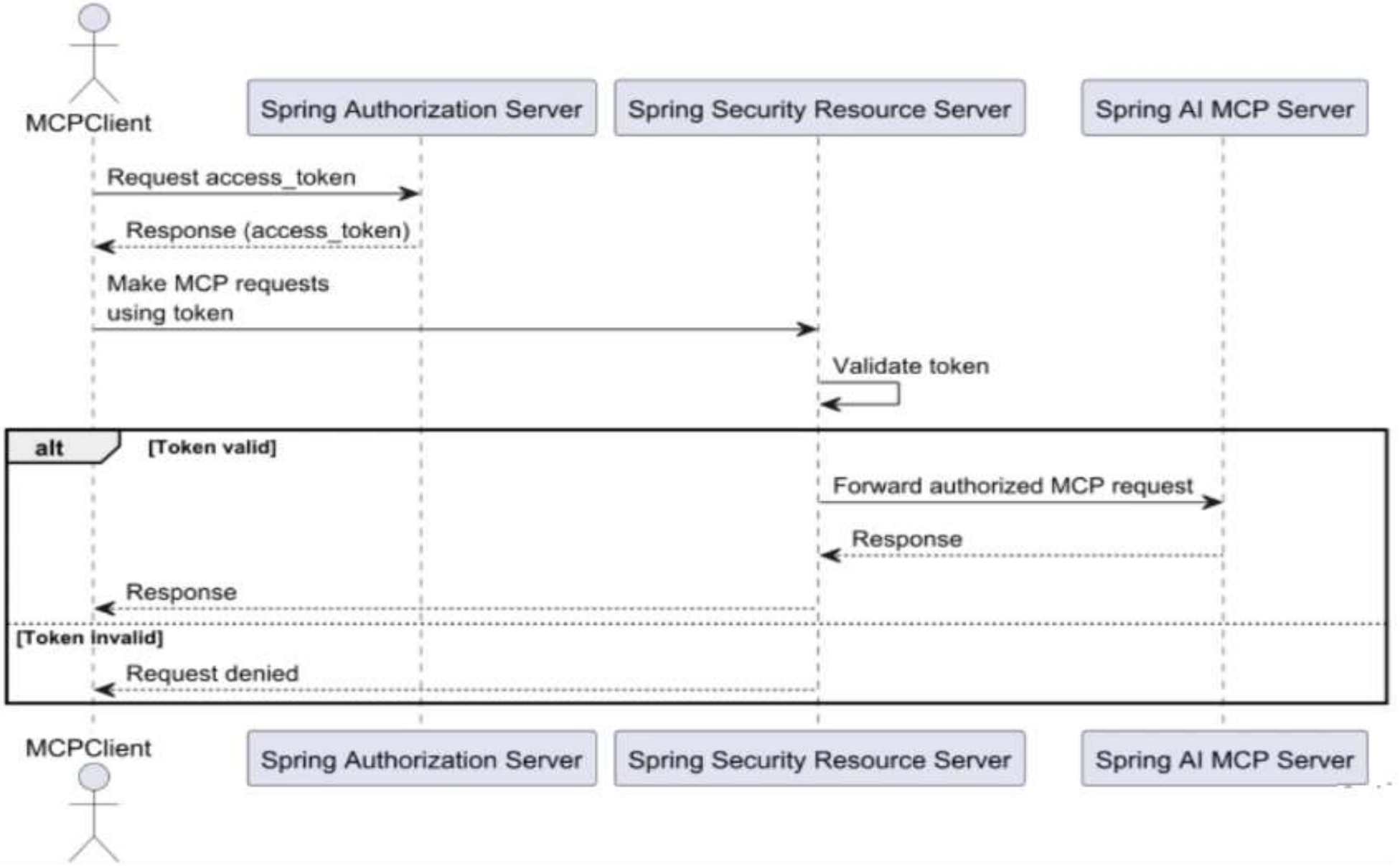


MCP Transport Mechanisms
Local vs. Remote




Spring AI
"Spring Boot for AI"
Core Features
LLM Abstraction Layer
Chat & Prompt Templates
Model Context Protocol (MCP)
Embedding Support
Vector Store Integration
Document Loading & Splitting
Retrieval-Augmented Generation
Spring Boot Integration
Security / Observability
Tool & Function Calling
Multi-Provider & Extensible

Authorization and security in MCP



Useful Links

- MCP Official Documentation
- MCP Github
- Spring AI - MCP
- MCP Clients