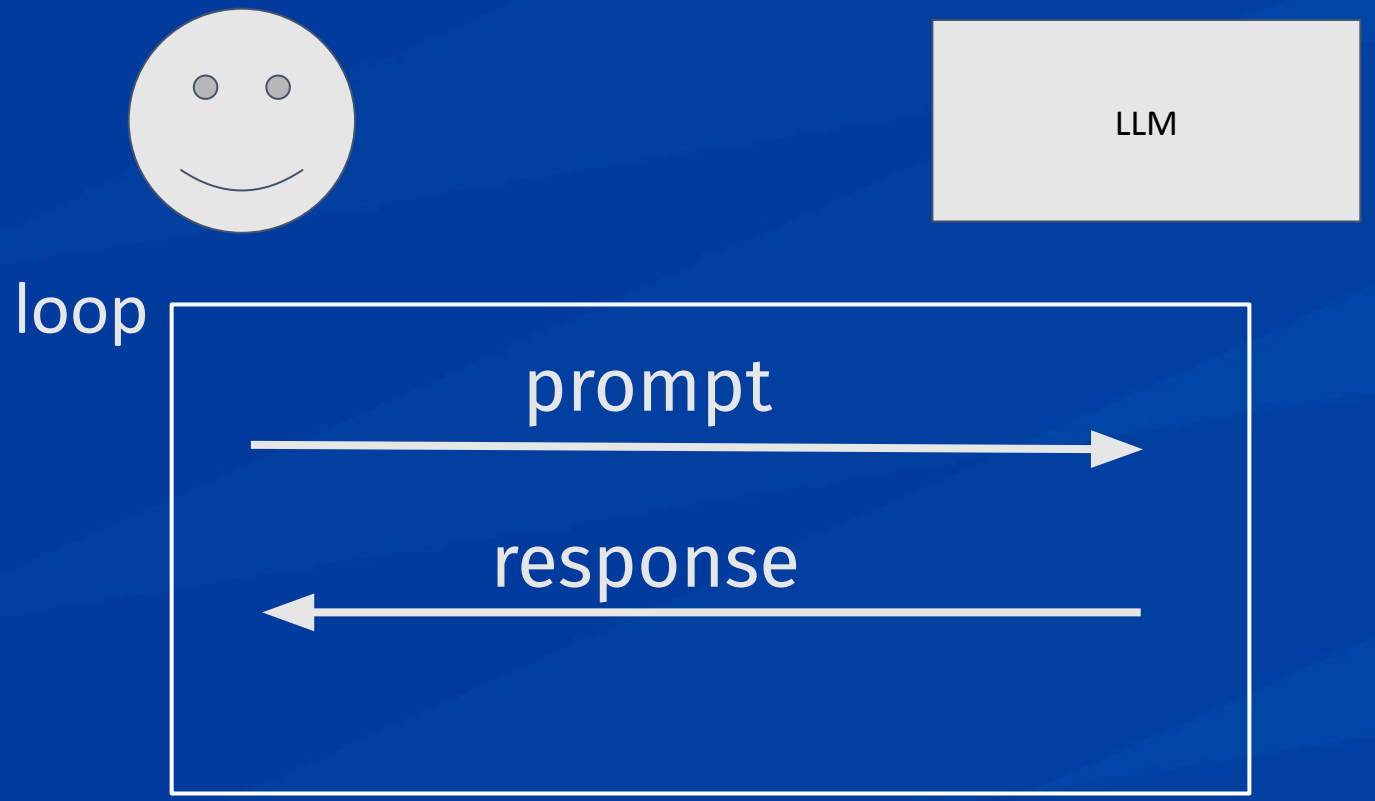
Model Context Protocol

An introduction to agentic AI with MCP

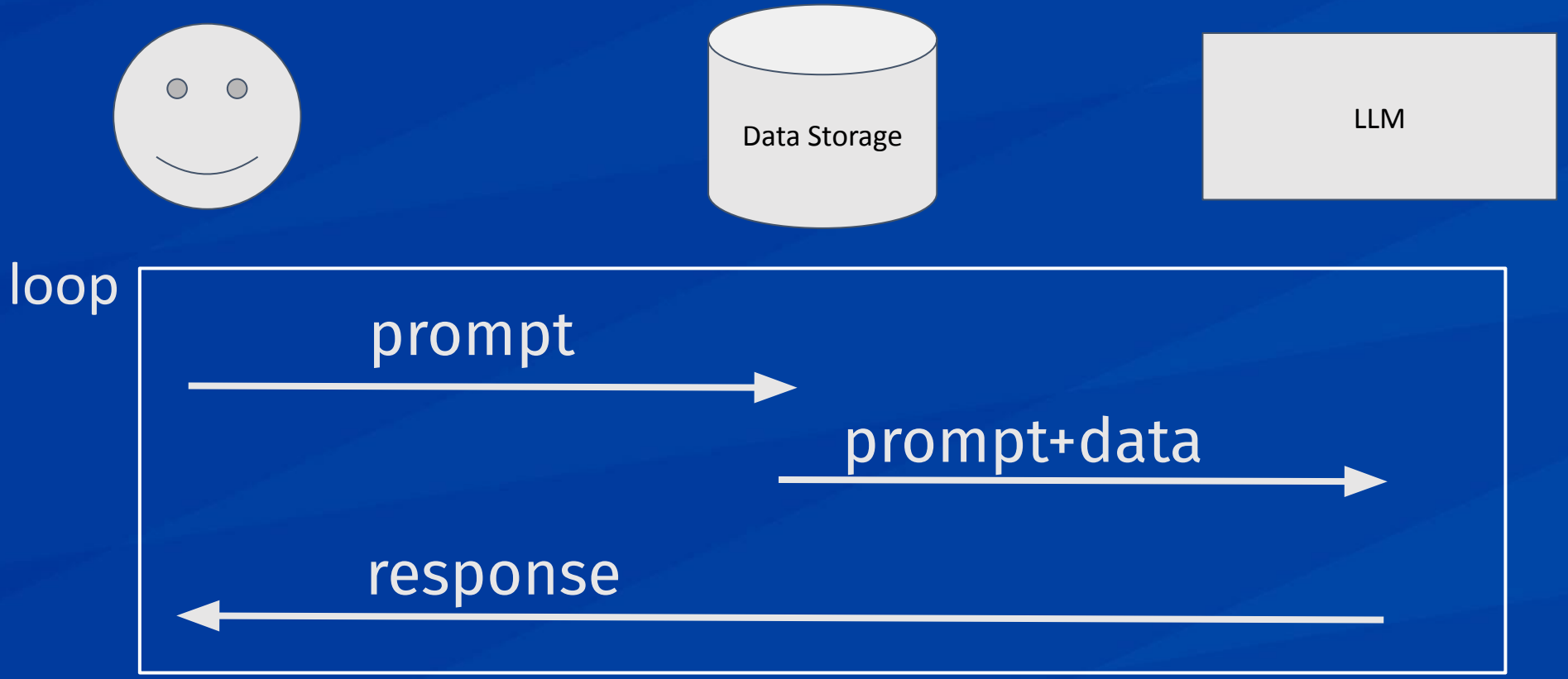
Joshua Zingale
July 2025

UC RIVERSIDE

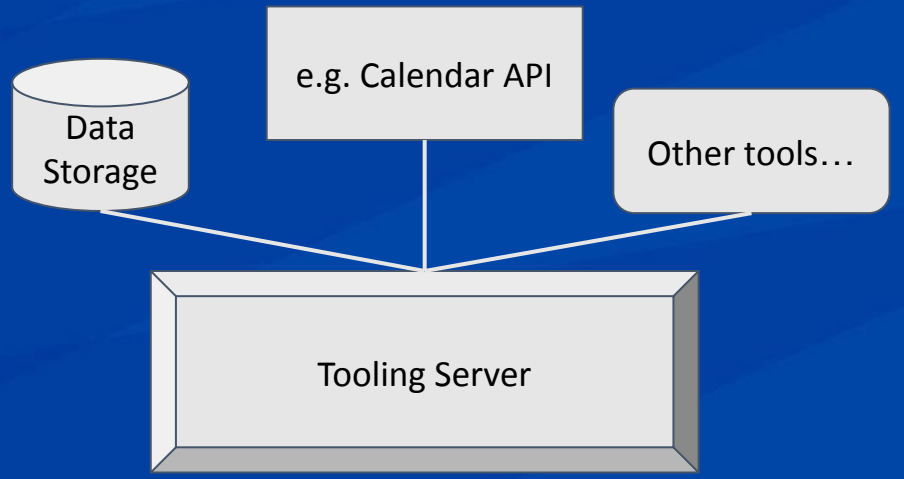
Vanilla LLM Interaction

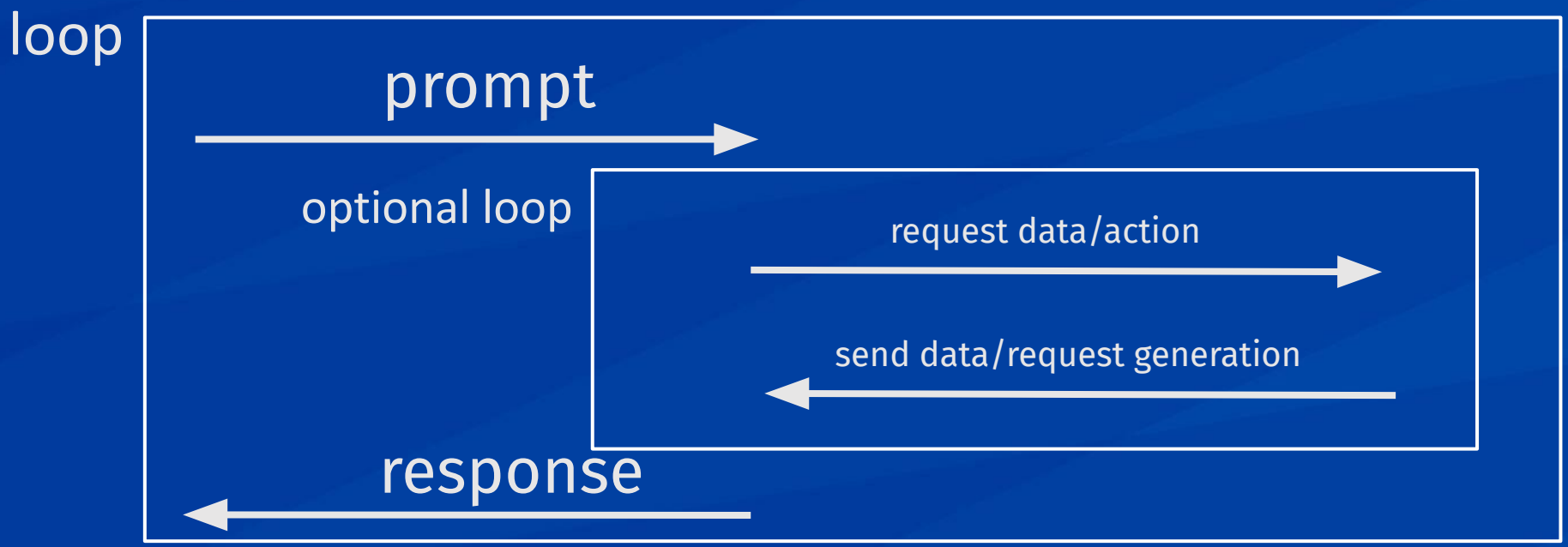


RAG Interaction



Agentic Interaction





UC RIVERSIDE

ANTHROP\C

Introducing the Model Context Protocol

Nov 25, 2024 • 3 min read

Google announces support for Anthropic's MCP standard for Gemini models and SDK

David Uzondu · Apr 10, 2025 05:38 EDT

Model Context Protocol

created by ANTHROP\C

OpenAI May 21, 2025

New tools and features in the Responses API

Today, we're adding new built-in tools to the Responses API—our core API primitive for building agentic applications. This includes support for all remote Model Context Protocol (MCP) servers, as well as tools like image

Learn

What is the Azure MCP Server (Preview)?

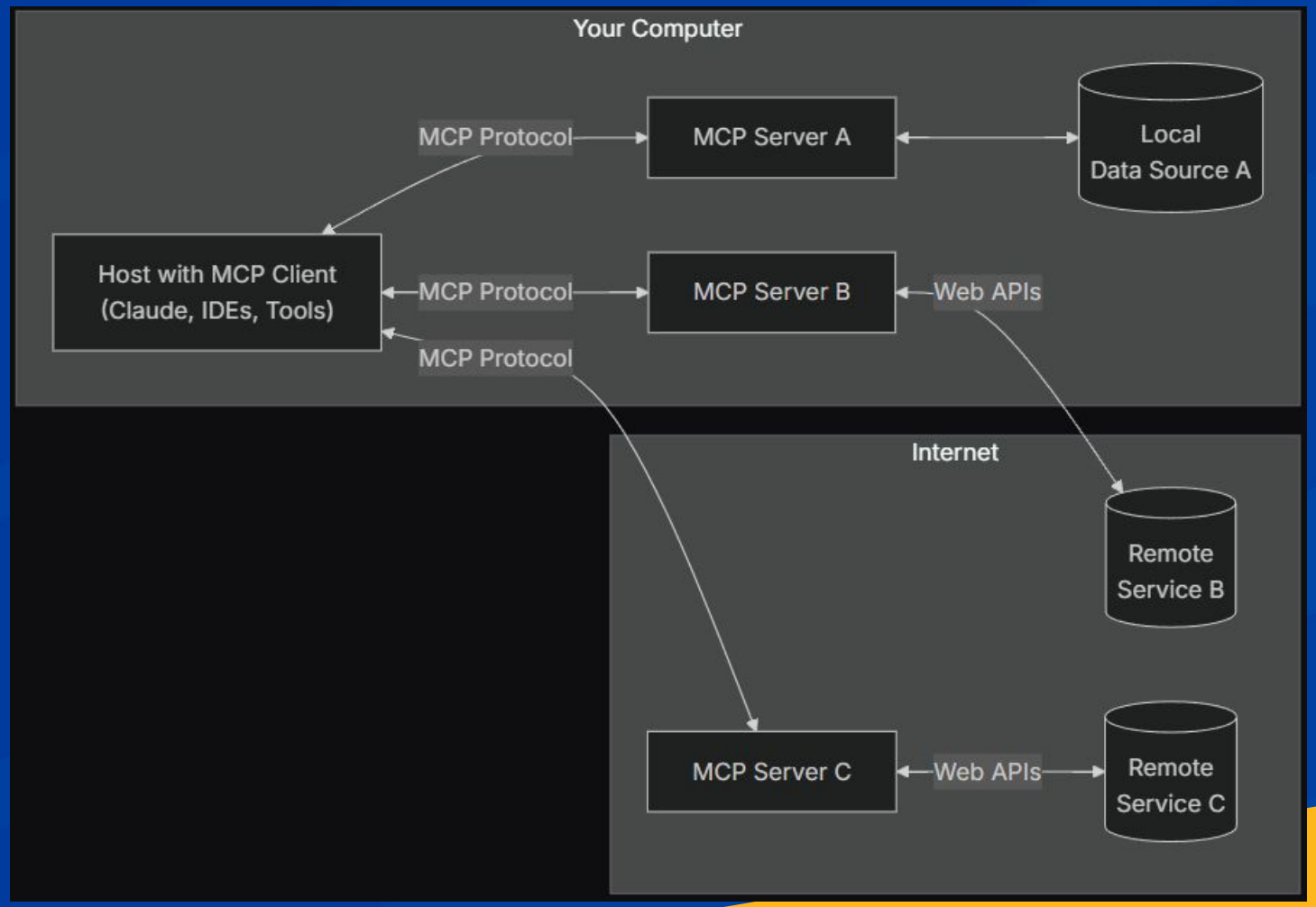
06/11/2025

The Azure MCP Server enables AI agents and other types of clients to interact with Azure resources through natural language commands. It implements the Model Context Protocol (MCP) to provide these key features:

- **MCP support**: Because the Azure MCP Server implements the Model Context Protocol, it works with MCP clients such as GitHub Copilot agent mode, the OpenAI Agents SDK, and Semantic Kernel.
- **Entra ID support**: The Azure MCP Server uses Entra ID through the Azure Identity library to follow Azure authentication best practices.
- **Service and tool support**: The Azure MCP Server supports Azure ser
  Azure Developer CLI (azd).

UC RIVERSIDE

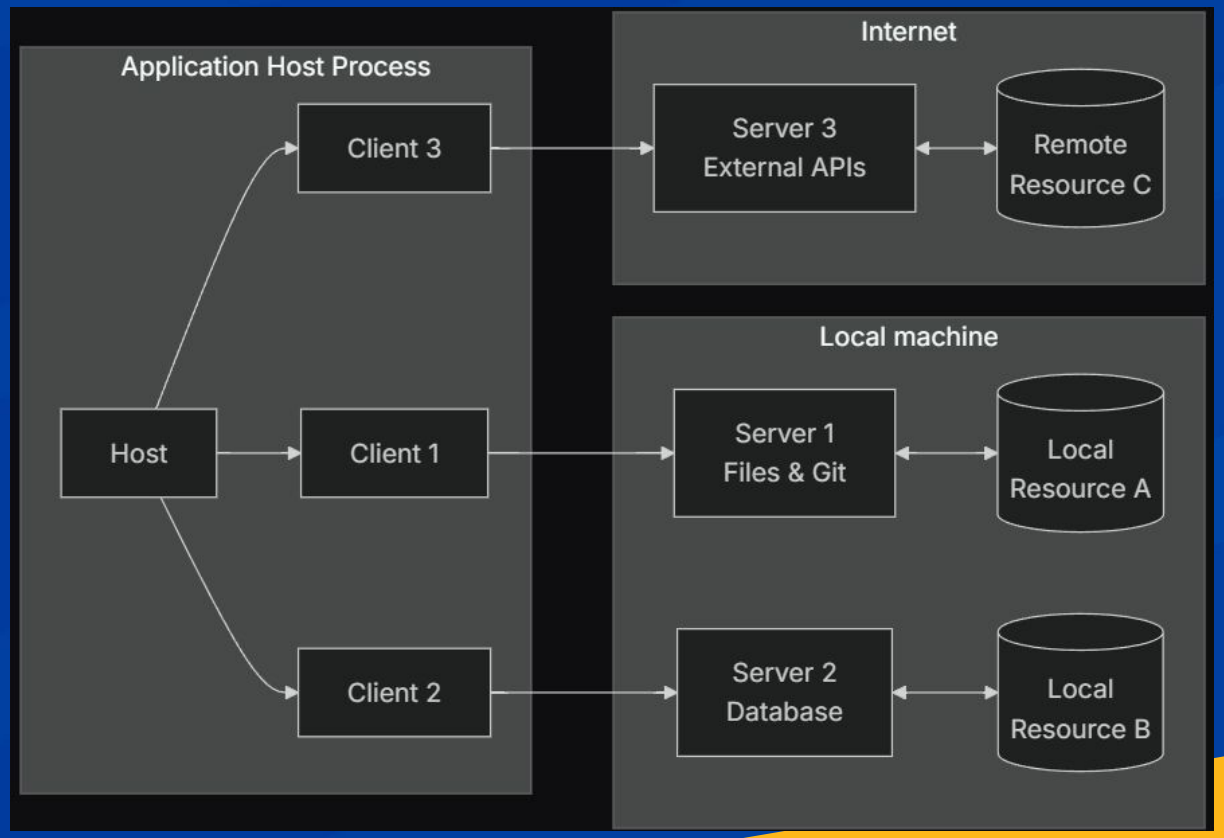
Model Context Protocol (MCP) Overview



UC RIVERSIDE

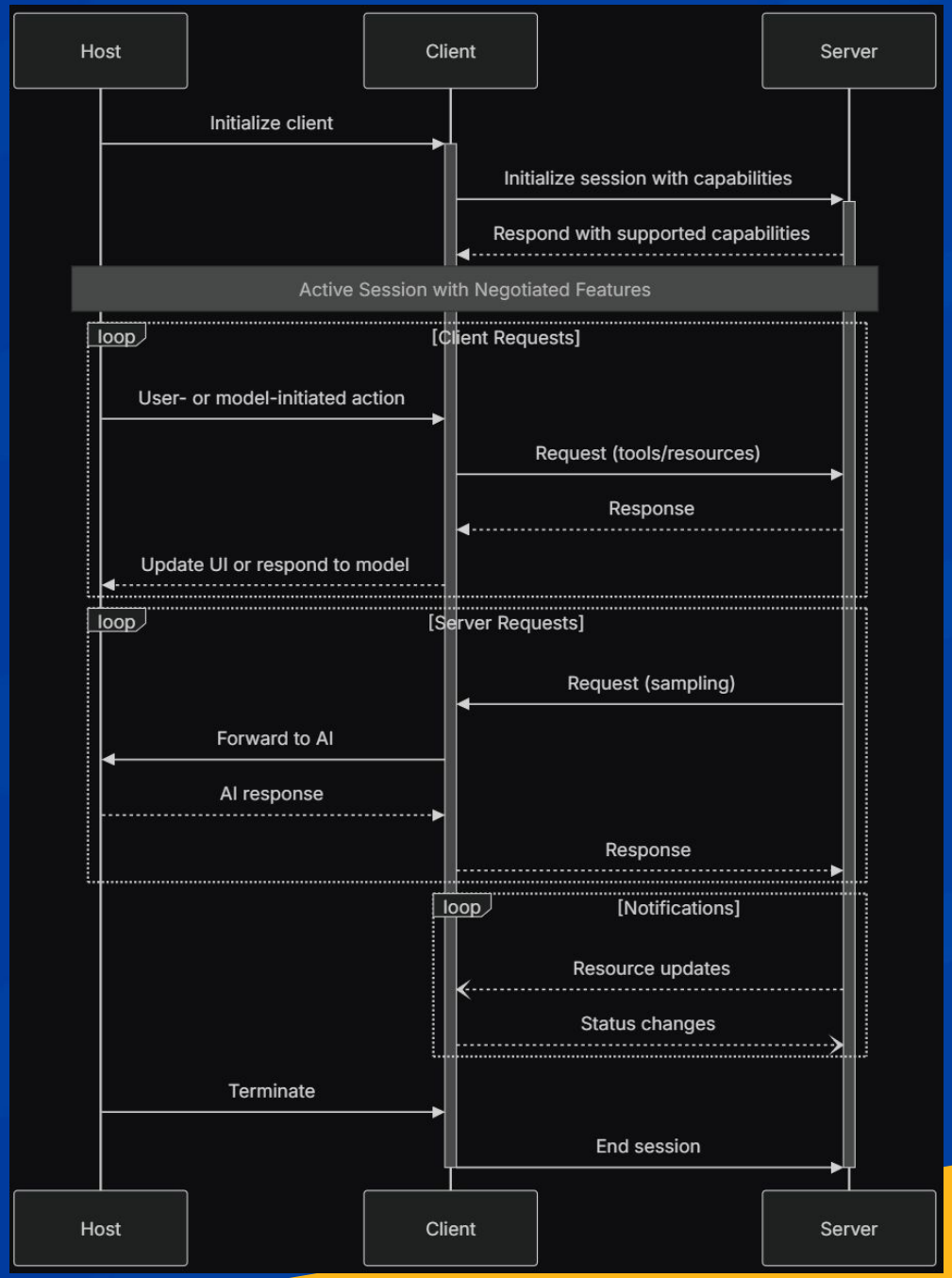
Three Component Architecture

- Host — Application that uses data provided by MCP servers, e.g. IDEs
- Client — Manages stateful connection to MCP servers
- Server — Provides a service, e.g. data retrieval or expression evaluation



UC RIVERSIDE

MCP Stages



UC RIVERSIDE

Communication

UC RIVERSIDE

Data Format: JSON-RPC

MCP uses JSON-RPC for all communication between Clients and Servers

Request

```
{
  jsonrpc: "2.0";
  id: string | number;
  method: string;
  params?: {
    [key: string]: unknown;
  };
}
```

Response

```
{
  jsonrpc: "2.0";
  id: string | number;
  result?: {
    [key: string]: unknown;
  }
  error?: {
    code: number;
    message: string;
    data?: unknown;
  }
}
```

Notification

```
{
  jsonrpc: "2.0";
  method: string;
  params?: {
    [key: string]: unknown;
  };
}
```

UC RIVERSIDE

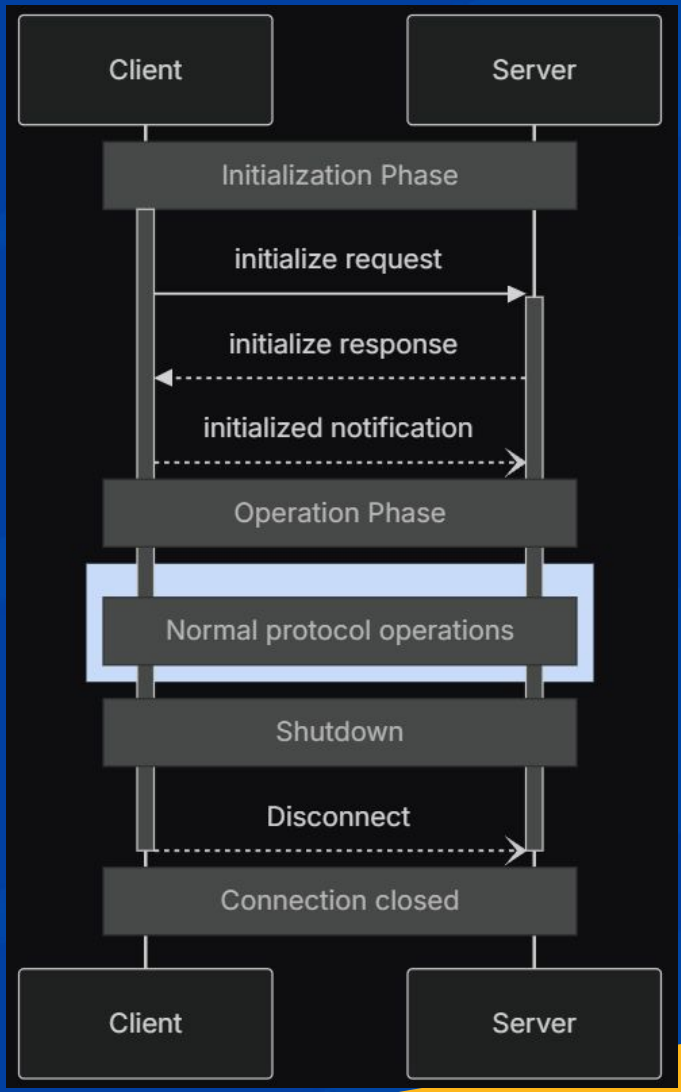
Initialization

Initialization request

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "initialize",
  "params": {
    "protocolVersion": "2024-11-05",
    "capabilities": {
      "roots": {
        "listChanged": true
      },
      "sampling": {},
      "elicitation": {}
    },
    "clientInfo": {
      "name": "ExampleClient",
      "title": "Example Client Display Name",
      "version": "1.0.0"
    }
  }
}
```

Initialization response

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "protocolVersion": "2024-11-05",
    "capabilities": {
      "logging": {},
      "prompts": {
        "listChanged": true
      },
      "resources": {
        "subscribe": true,
        "listChanged": true
      },
      "tools": {
        "listChanged": true
      }
    },
    "serverInfo": {
      "name": "ExampleServer",
      "title": "Example Server Display Name",
      "version": "1.0.0"
    },
    "instructions": "Optional instructions for the client"
  }
}
```

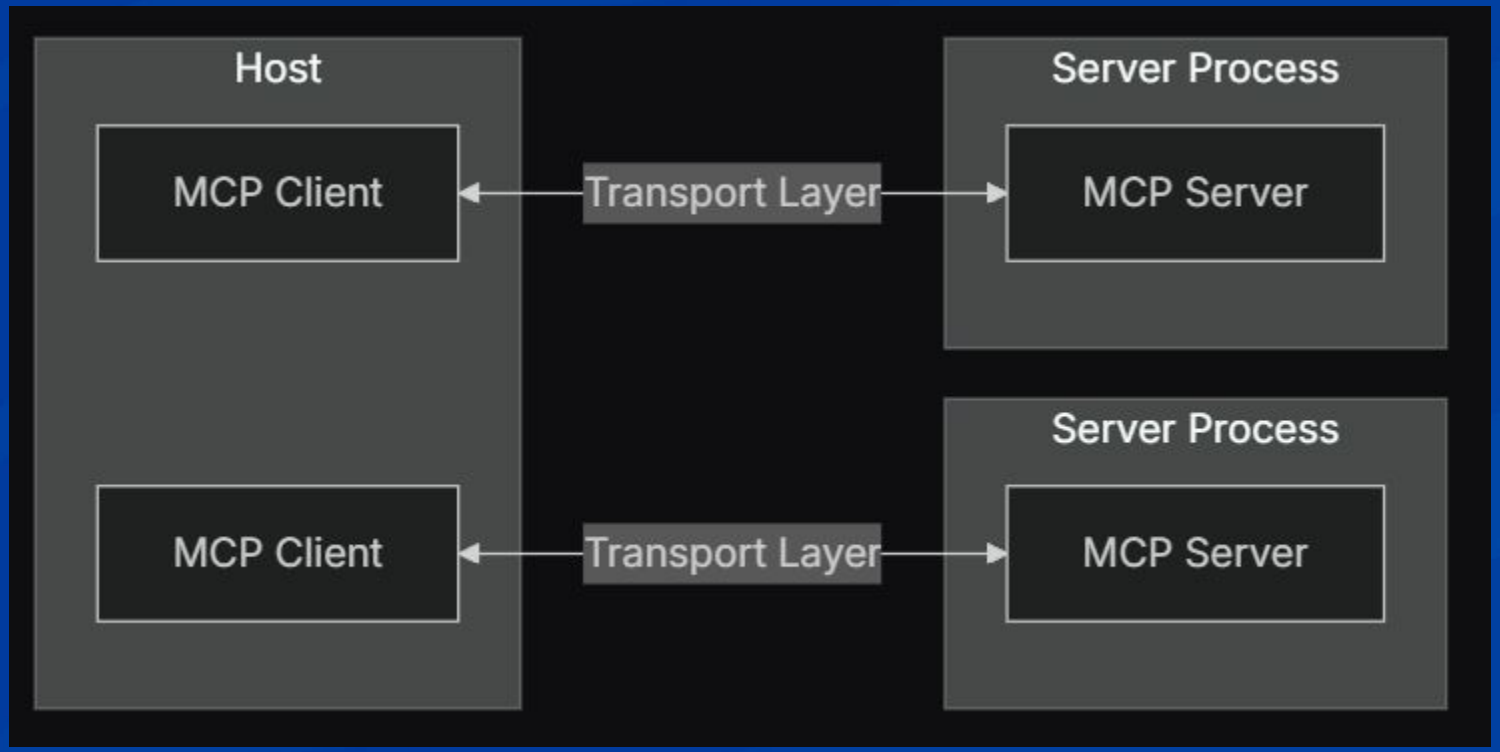


UC RIVERSIDE

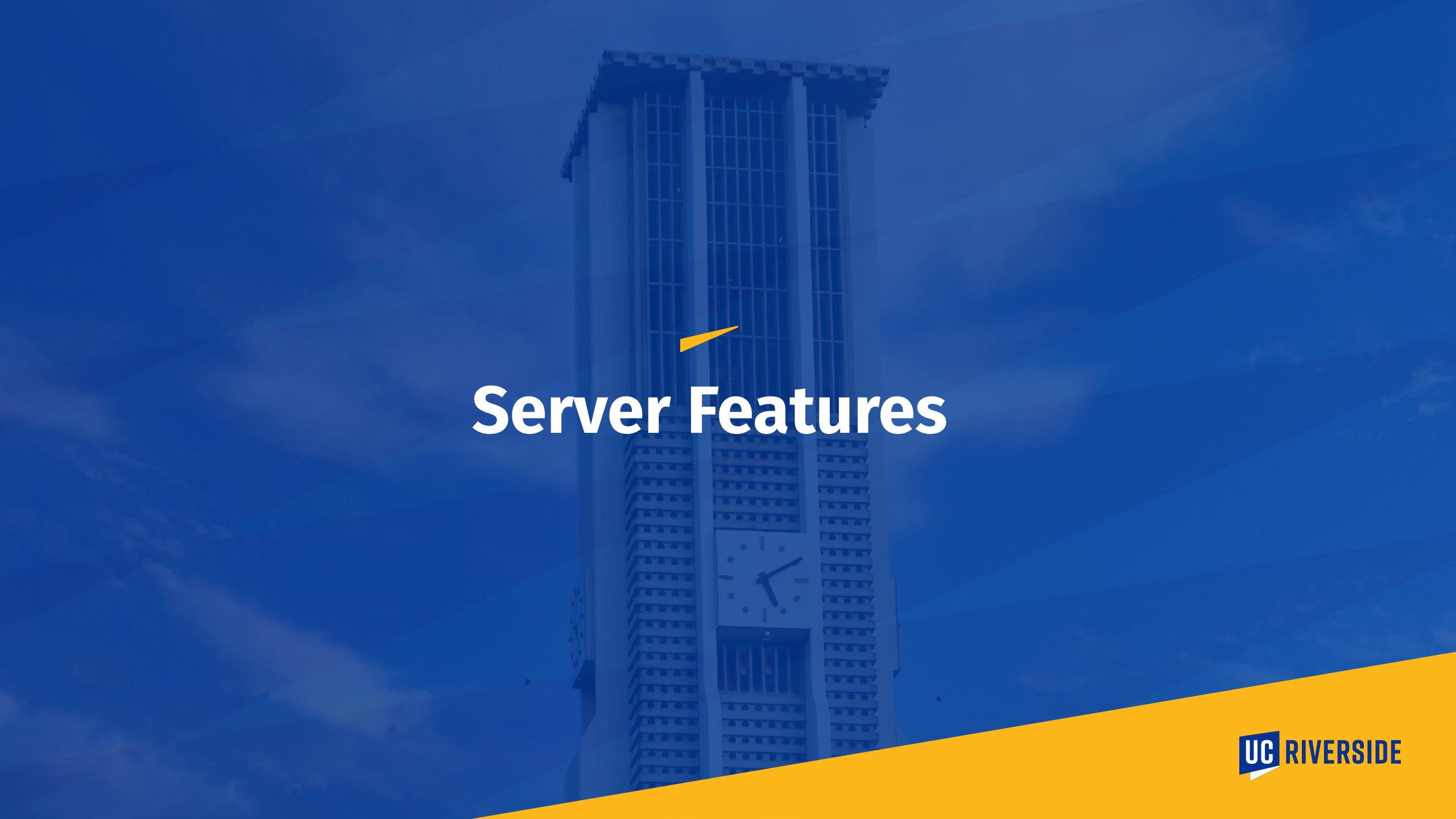
Transport Options

The JSON-RPC messages can be communicated via different methods:

- stdio — Uses standard input/output for communication
- Streamable HTTP — Uses HTTP with optional Server-Sent Events for streaming



UC RIVERSIDE



Server Features

UC RIVERSIDE

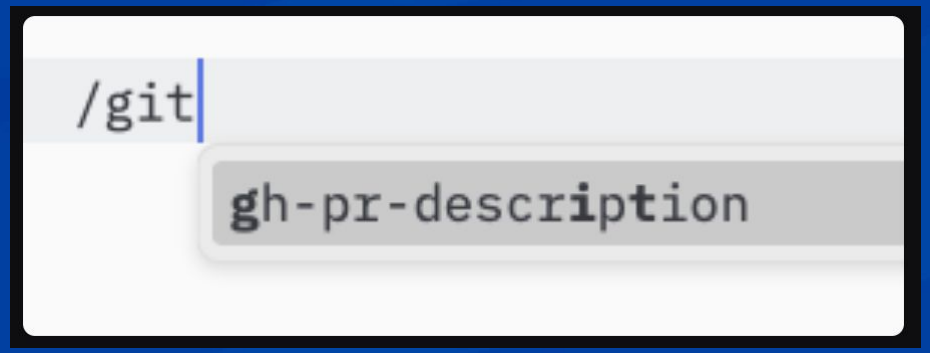
Primitive Server Features

MCP defines three primitive method types, of which a server may support any, many, or all

Primitive	Control	Description	Example
Prompts	User-controlled	Interactive templates invoked by user choice	Slash commands, menu options
Resources	Application-controlled	Contextual data attached and managed by the client	File contents, git history
Tools	Model-controlled	Functions exposed to the LLM to take actions	API POST requests, file writing

Prompts

Prompts typically are directly requested by the user and respond with a prepared prompt.
They could be accessed by a slash command in a host application:



Example request from Client

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "prompts/get",
  "params": {
    "name": "code_review",
    "arguments": {
      "code": "def hello():\n    print('world')"
    }
  }
}
```

UC RIVERSIDE

Resources

Resources are data that can be accessed from a server by a Client.

There are more advanced features as well:

- Resource templates for accessing arbitrary URI
  - For example, to access an arbitrary file path
- Subscription to be notified when a resource is updated
- Pagination to list resources

Example request from Client

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "resources/read",
  "params": {
    "uri": "file:///project/src/main.rs"
  }
}
```

Tools

Tools in MCP are designed to be model-controlled, meaning that the language model can discover and invoke tools automatically based on its contextual understanding and the user's prompts.

Example request from Client

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "get_weather",
    "arguments": {
      "location": "New York"
    }
  }
}
```

Example response from Server

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "Current weath
      }
    ],
    "isError": false
  }
}
```

UC RIVERSIDE

Discovery

To determine what prompts, resources, and/or tools are available for a language model, a list request must be sent from the client to the Server

Example request from Client

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {
    "cursor": "optional-cursor-value"
  }
}
```

Example response from Server

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "get_weather",
        "title": "Weather Information Provider",
        "description": "Get current weather information for a location'
        "inputSchema": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "City name or zip code"
            }
          },
          "required": ["location"]
        }
      }
    ],
    "nextCursor": "next-page-cursor"
  }
}
```

UC RIVERSIDE

Client Features

UC RIVERSIDE

Roots

Clients may expose directory “roots” to servers. This may be used by acode-editing server to gauge where file edits should be made.

Example request from Server

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "roots/list"
}
```

Example response from Client

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "roots": [
      {
        "uri": "file:///home/user/projects/myproject",
        "name": "My Project"
      }
    ]
  }
}
```

UC RIVERSIDE

Sampling

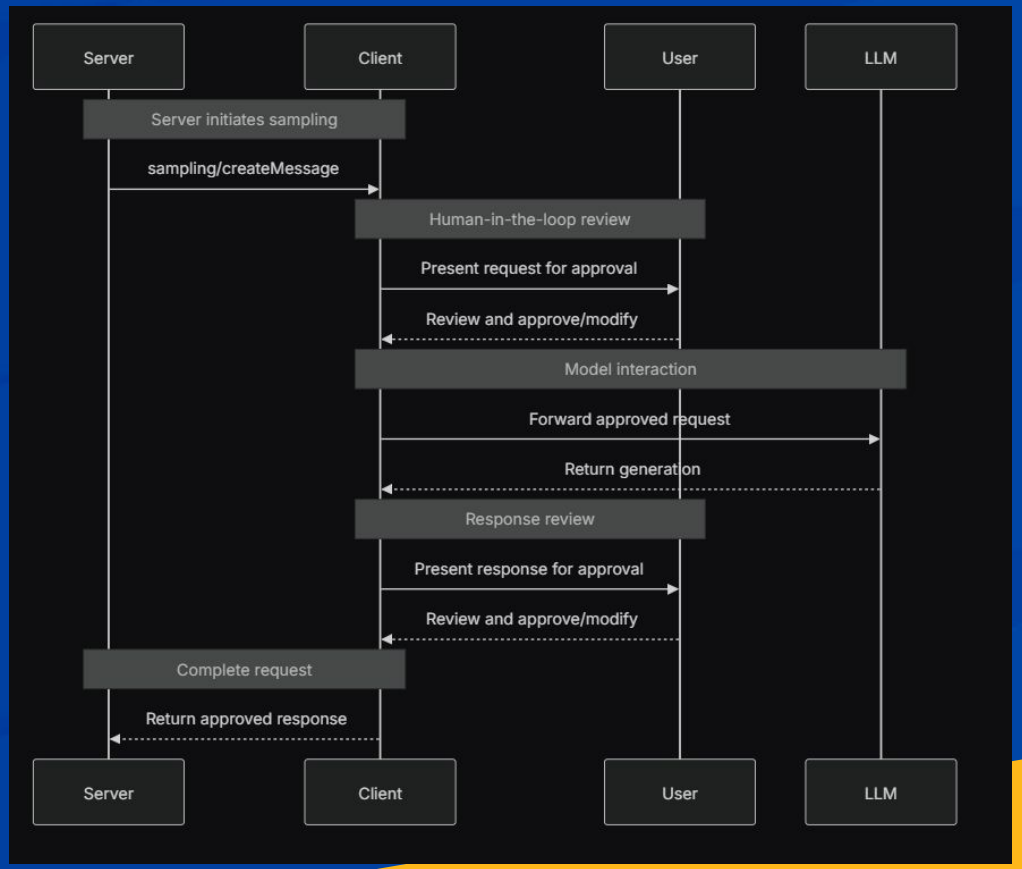
Sampling in MCP allows servers to implement agentic behaviors, by enabling LLM calls to occur nested inside other MCP server features.

Example request from Server

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "sampling/createMessage",
  "params": {
    "messages": [
      {
        "role": "user",
        "content": {
          "type": "text",
          "text": "What is the capital of France?"
        }
      }
    ],
    "modelPreferences": {
      "hints": [
        {
          "name": "claude-3-sonnet"
        }
      ],
      "intelligencePriority": 0.8,
      "speedPriority": 0.5
    },
    "systemPrompt": "You are a helpful assistant.",
    "maxTokens": 100
  }
}
```

Example response from Client

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "role": "assistant",
    "content": {
      "type": "text",
      "text": "The capital of France is Paris."
    },
    "model": "claude-3-sonnet-20240307",
    "stopReason": "endTurn"
  }
}
```



UC RIVERSIDE

Elicitation

Elicitation in MCP allows servers to implement interactive workflows by enabling user input requests to occur nested inside other MCP server features.

Example request from Server

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "elicitation/create",
  "params": {
    "message": "Please provide your GitHub username",
    "requestedSchema": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        }
      },
      "required": ["name"]
    }
  }
}
```

Example response from Client

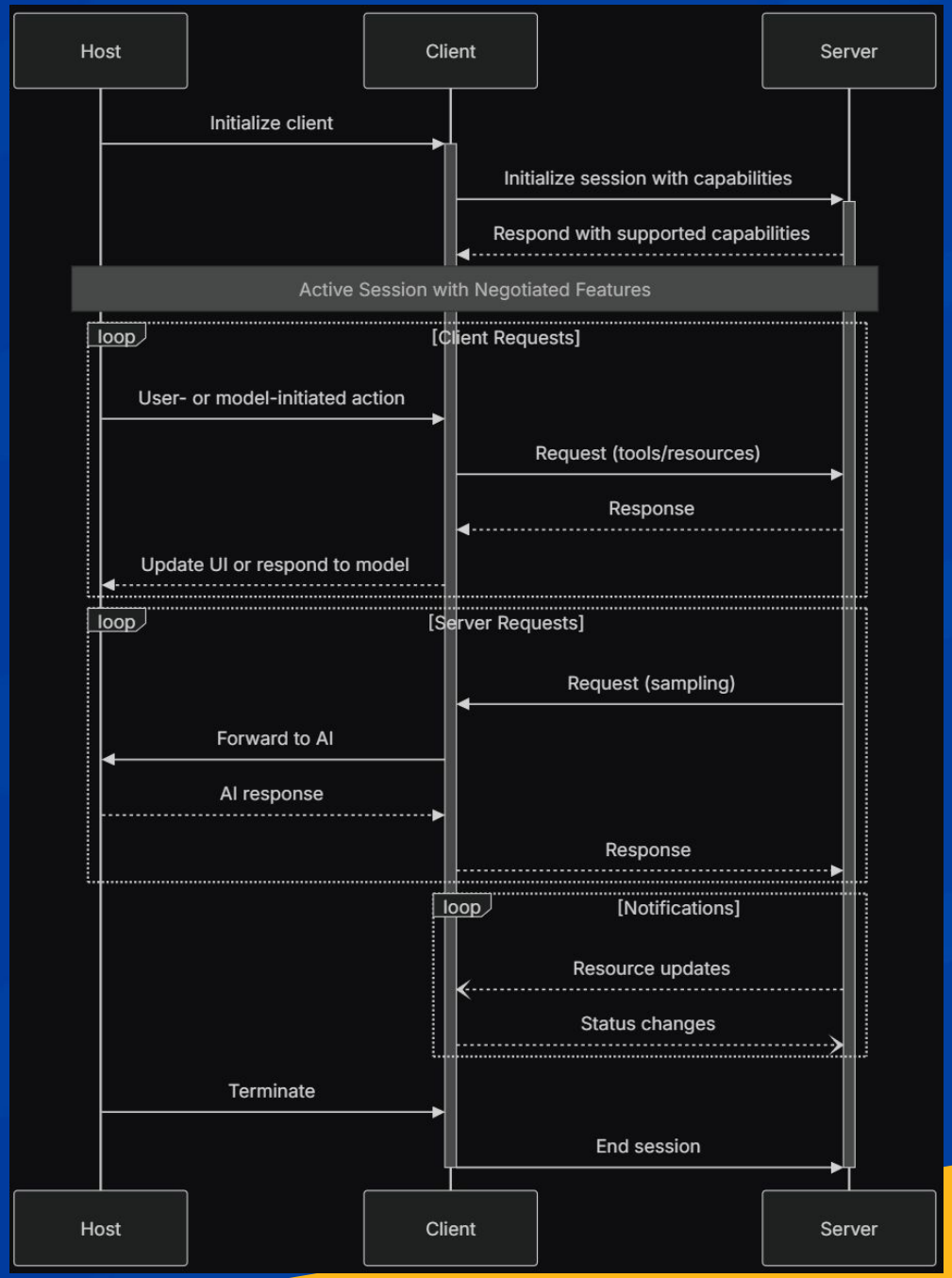
```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "action": "accept",
    "content": {
      "name": "octocat"
    }
  }
}
```

UC RIVERSIDE

Full Picture Review

UC RIVERSIDE

MCP Stages



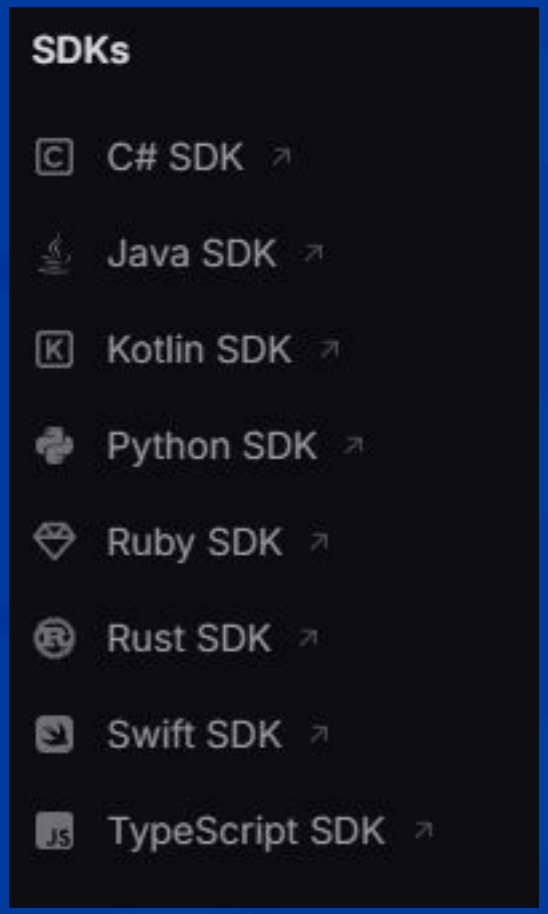
UC RIVERSIDE

Standard Development Kit

UC RIVERSIDE

SDKs

There are SDKs for most of the languages you would want to use.



Python Example

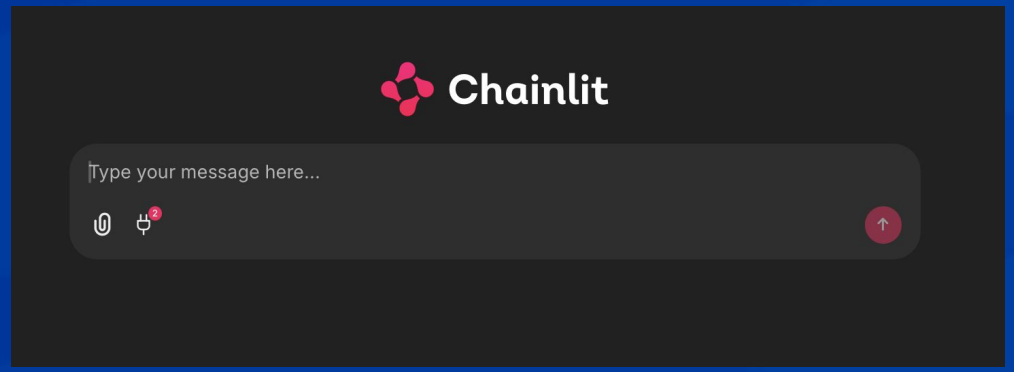
```python
"""
FastMCP quickstart example.

cd to the `examples/snippets/clients` directory and run:
    uv run server fastmcp_quickstart stdio
"""

from mcp.server.fastmcp import FastMCP

# Create an MCP server
mcp = FastMCP("Demo")


# Add an addition tool
@mcp.tool()
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b
```

UC RIVERSIDE

Workshop Activity

UC RIVERSIDE

Workshop: See the Github Repository



Provided host that supports tools

+

Ollama

+

```
"""
FastMCP quickstart example.

cd to the `examples/snippets/clients` directory and run:
    uv run server fastmcp_quickstart stdio
"""

from mcp.server.fastmcp import FastMCP

# Create an MCP server
mcp = FastMCP("Demo")


# Add an addition tool
@mcp.tool()
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b
```

FastMCP servers

=

Agentic AI

UC RIVERSIDE