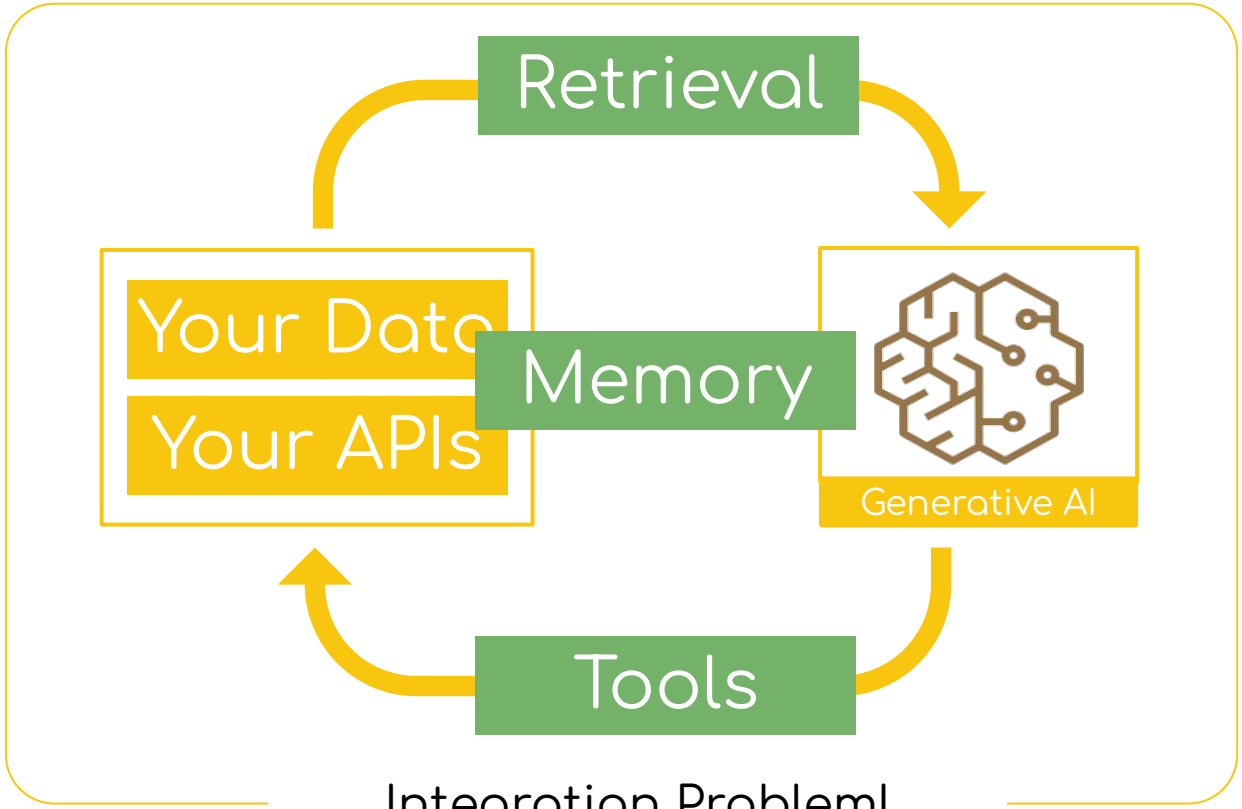
AI Integration with Spring AI and Model Context Protocol

Christian Tzolov
Spring AI & MCP Java SDK
@christzolov , @tzolov

Applying Generative AI



Integration Problem!

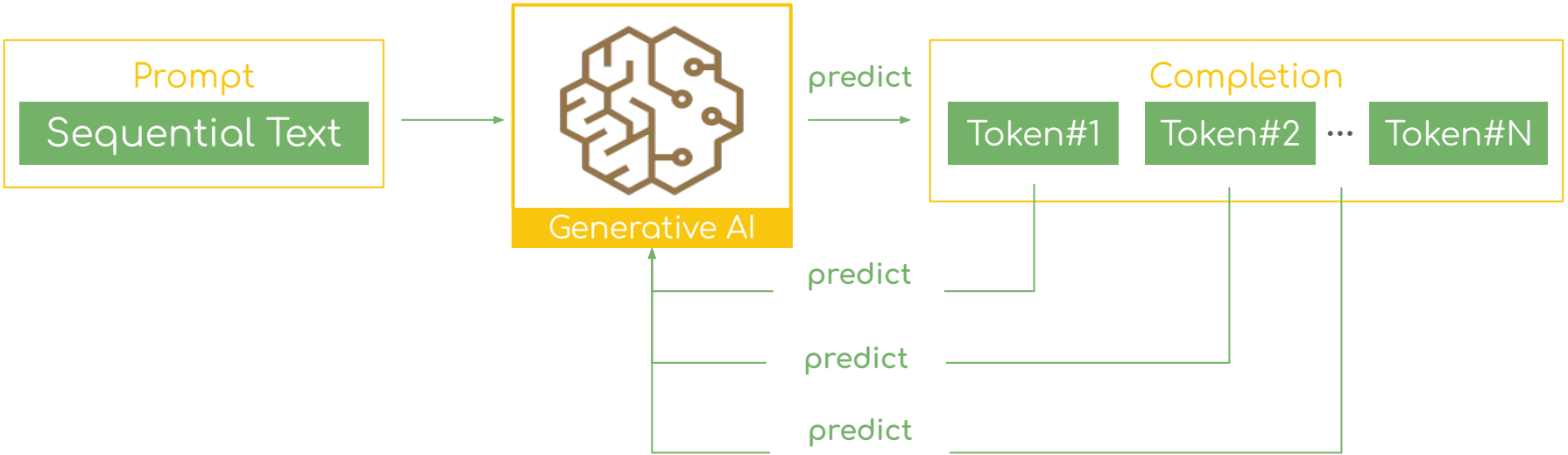
Spring AI

SpringAI.java

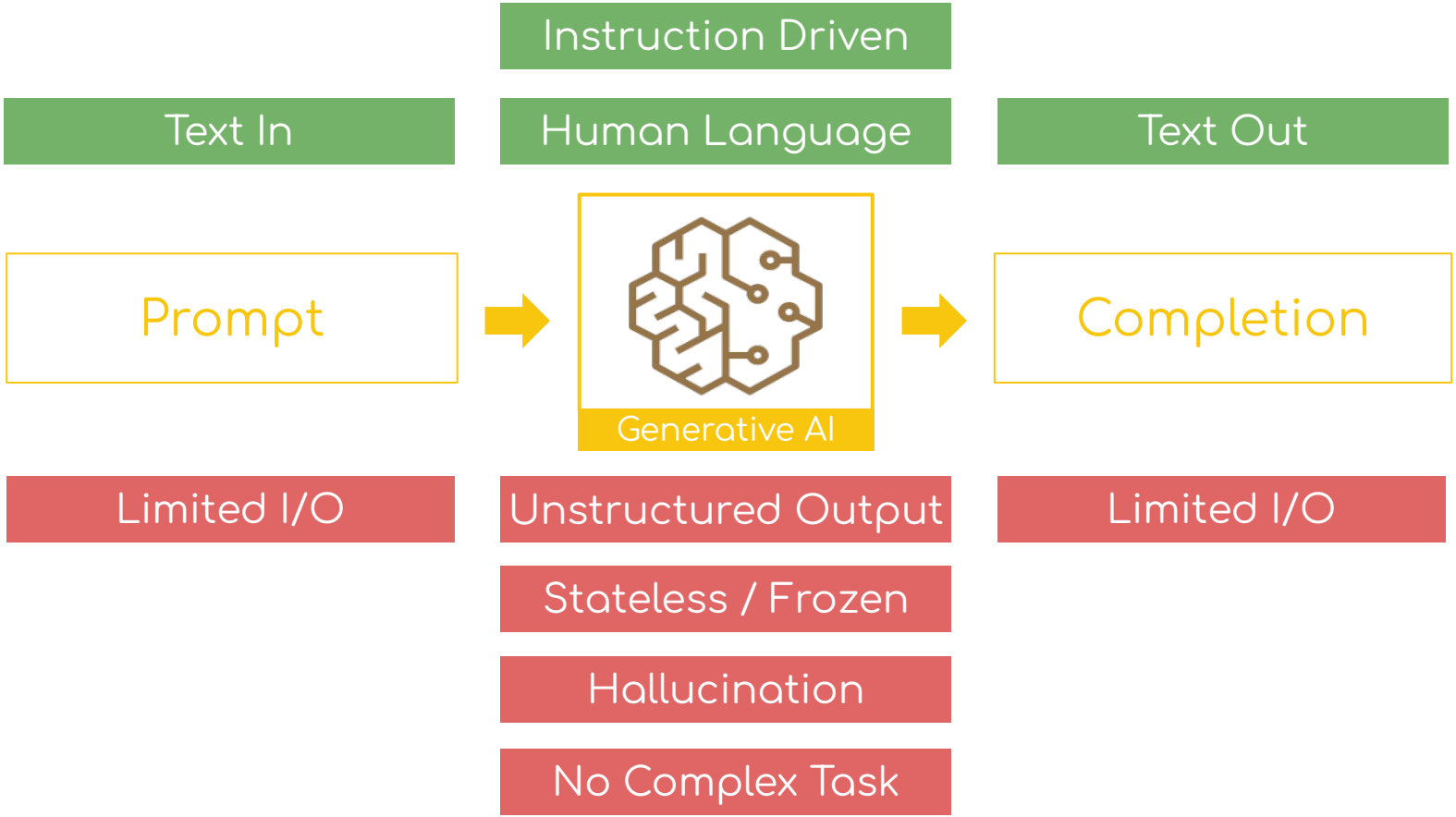
```
public class Spring
  private final Cha

  public SpringAI(C
    this.chatClient
  }

  public String tel
    return chatClie
        .call()
        .content();
  }
}
```

Provider	Multimodality	Tools/Functions	Streaming	Retry	Observability	Built-in JSON	Local	OpenAI API Compatible
Anthropic Claude	text, pdf, image	✔	✔	✔	✔	✖	✖	✖
Azure OpenAI	text, image	✔	✔	✔	✔	✔	✖	✔
DeepSeek (OpenAI-proxy)	text	✖	✔	✔	✔	✔	✔	✔
Google VertexAI Gemini	text, pdf, image, audio, video	✔	✔	✔	✔	✔	✖	✔
Groq (OpenAI-proxy)	text, image	✔	✔	✔	✔	✖	✖	✔
HuggingFace	text	✖	✖	✖	✖	✖	✖	✖
Mistral AI	text, image	✔	✔	✔	✔	✔	✖	✔
MiniMax	text	✔	✔	✔	✔	✖	✖	
Moonshot AI	text	✖	✔	✔	✔	✖	✖	
NVIDIA (OpenAI-proxy)	text, image	✔	✔	✔	✔	✖	✖	✔
OCI GenAI/Cohere	text	✖	✖	✖	✔	✖	✖	✖
Ollama	text, image	✔	✔	✔	✔	✔	✔	✔
OpenAI	In: text, image, audio Out: text, audio	✔	✔	✔	✔	✔	✖	✔
Perplexity (OpenAI-proxy)	text	✖	✔	✔	✔	✖	✖	✔
QianFan	text	✖	✔	✔	✔	✖	✖	✖
ZhiPu AI	text	✔	✔	✔	✔	✖	✖	✖
Watsonx.AI	text	✖	✔	✖	✖	✖	✖	✖
Amazon Bedrock Converse	text, image, video, docs (pdf, html, md, docx ...)	✔	✔	✔	✔	✖	✖	✖

LLM as a System - Prediction Engine

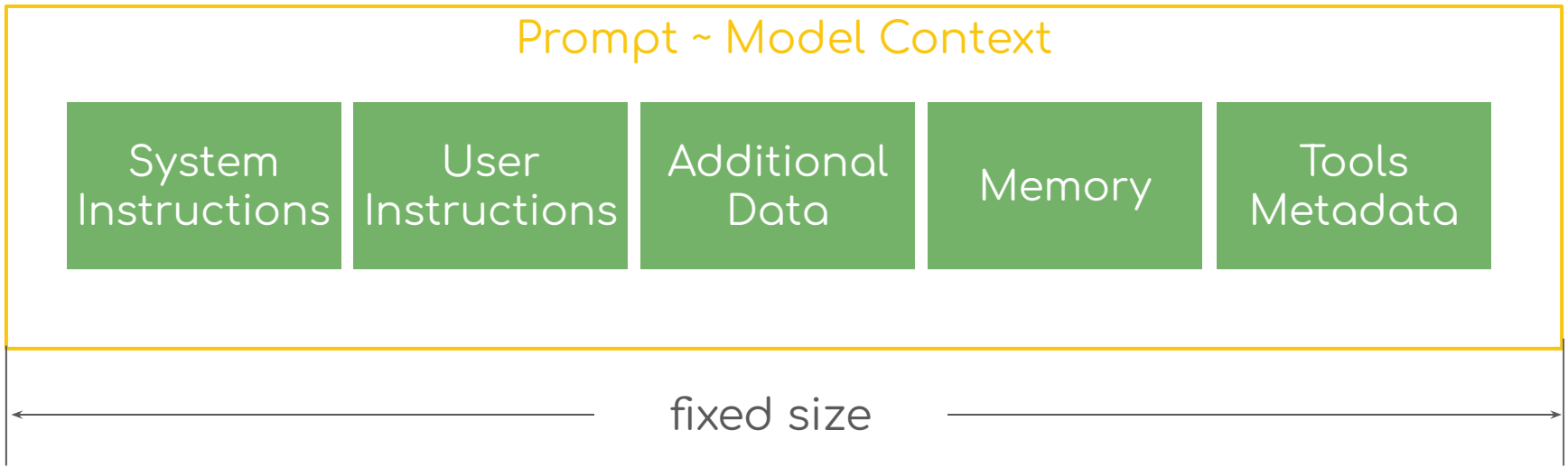


LLM Single-turn Model & Constraints



Prompt ~ Model Context

AI Models are only as good as the Model Context provided to them



LLM Prompt Augmentation Strategies

- System Instructions
- Short-term Chat Memory
- Long-term Chat Memory

Conversation Memory

- Prompt Stuffing
- Retrieval Augmented Generation (RAG)

Additional Context

- Structured Output
- Tool Calling

Tool Metadata

Agentic System

"Leverages an AI model to interact with its environment in order to solve a user-defined task"

Combines **Planning** & **Actions** to fulfill the tasks

Brain (LLM)

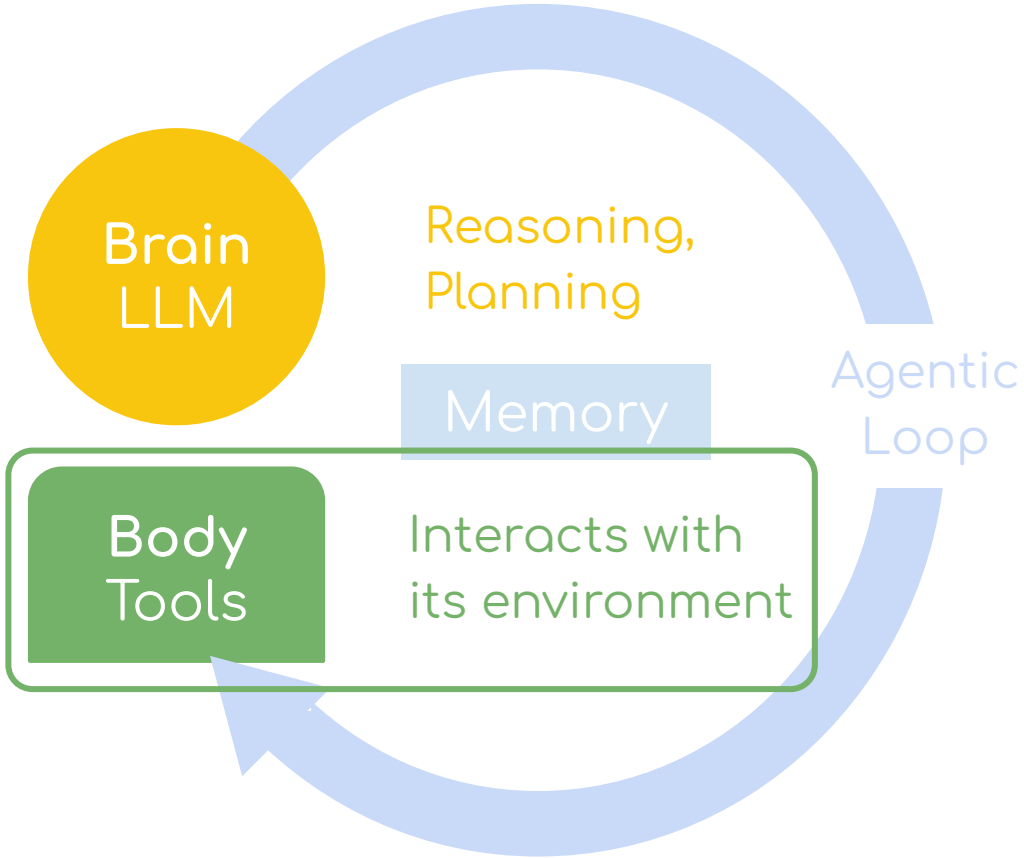
Handles reasoning and planning

Decides which Actions to take

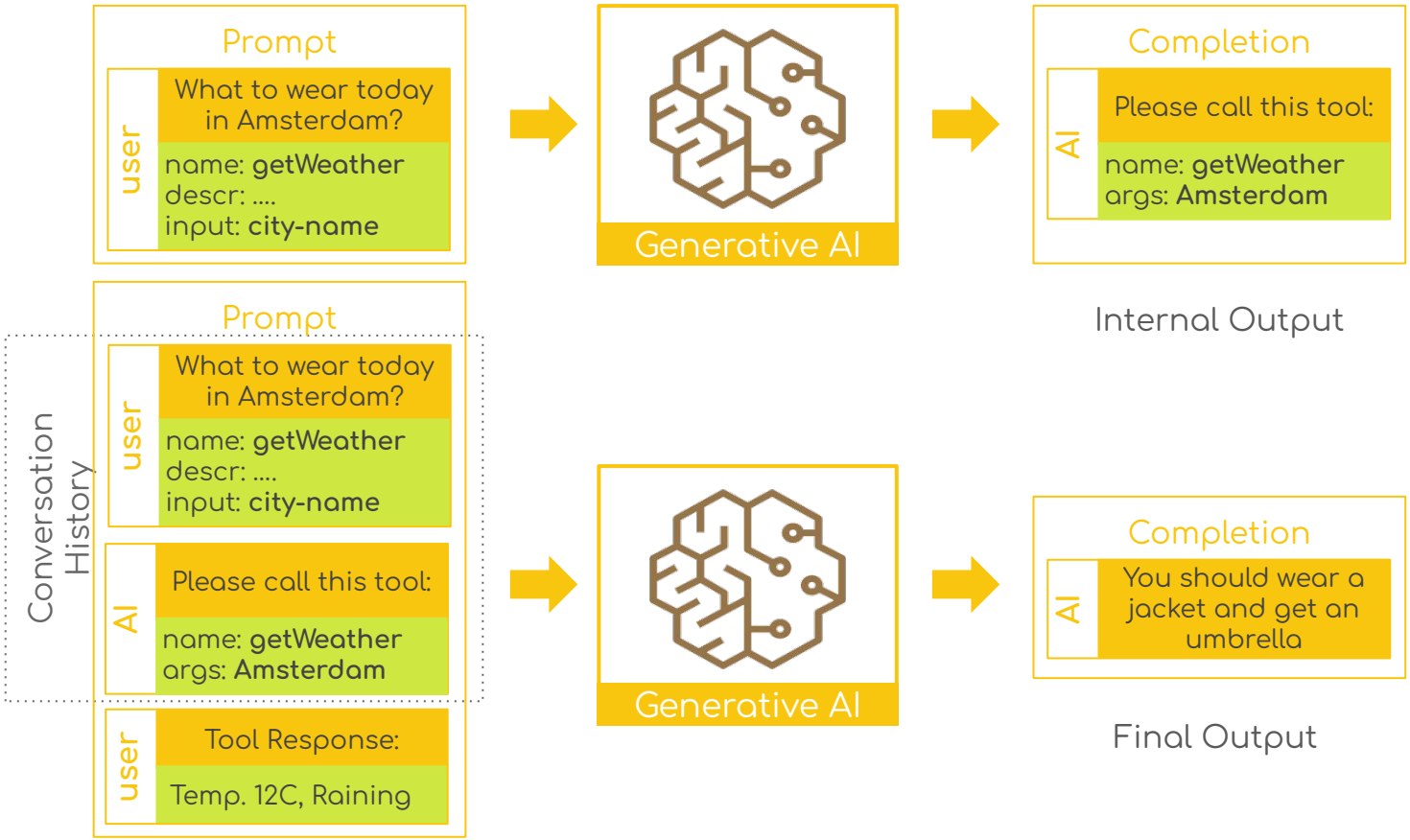
Body (Tools)

What Agent is equipped to do

Interact with its environment


Brain
LLM
Reasoning,
Planning
Memory
Agentic
Loop
Body
Tools
Interacts with
its environment

Tool Calling



Tool Calling - Spring AI Demo

```java
@Tool(description = "Get the temperature (in celsius) for a specific location")
public WeatherResponse getTemperature(@ToolParam(description = "The location latitude") double latitude,
    @ToolParam(description = "The location longitude") double longitude,
    @ToolParam(description = "The city name") String city) {

  WeatherResponse response = restClient
      .get()
      .uri("https://api.open-meteo.com/v1/forecast?latitude={latitude}&longitude={longitude}&current=tempera
          latitude, longitude)
      .retrieve()
      .body(WeatherResponse.class);

  logger.info("Check temparature for {}. Lat: {}, Lon: {}. Temp: {}", city, latitude, longitude,
      response.current);

  return response;
}
```

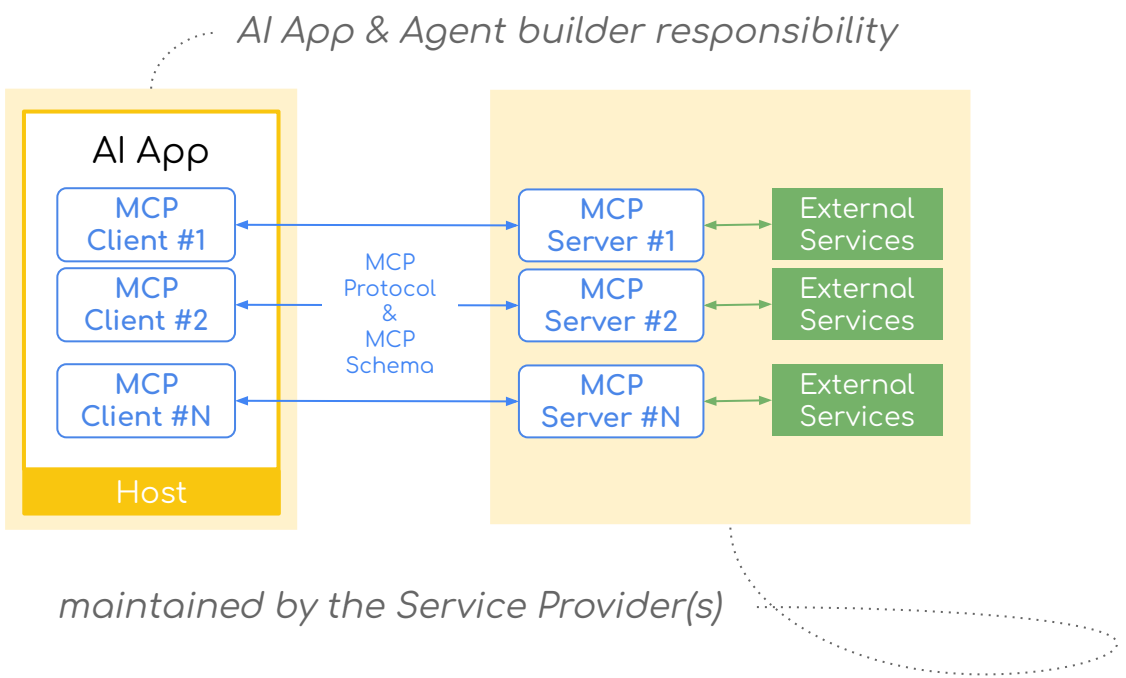
```java
WeatherTools myTools = ...
var output = chatClient.prompt()
  .tools(myTools)
  .user("What to wear today in Amsterdam and in Barcelona?")
  .call()
  .content();
```

Wouldn’t it be nice ...

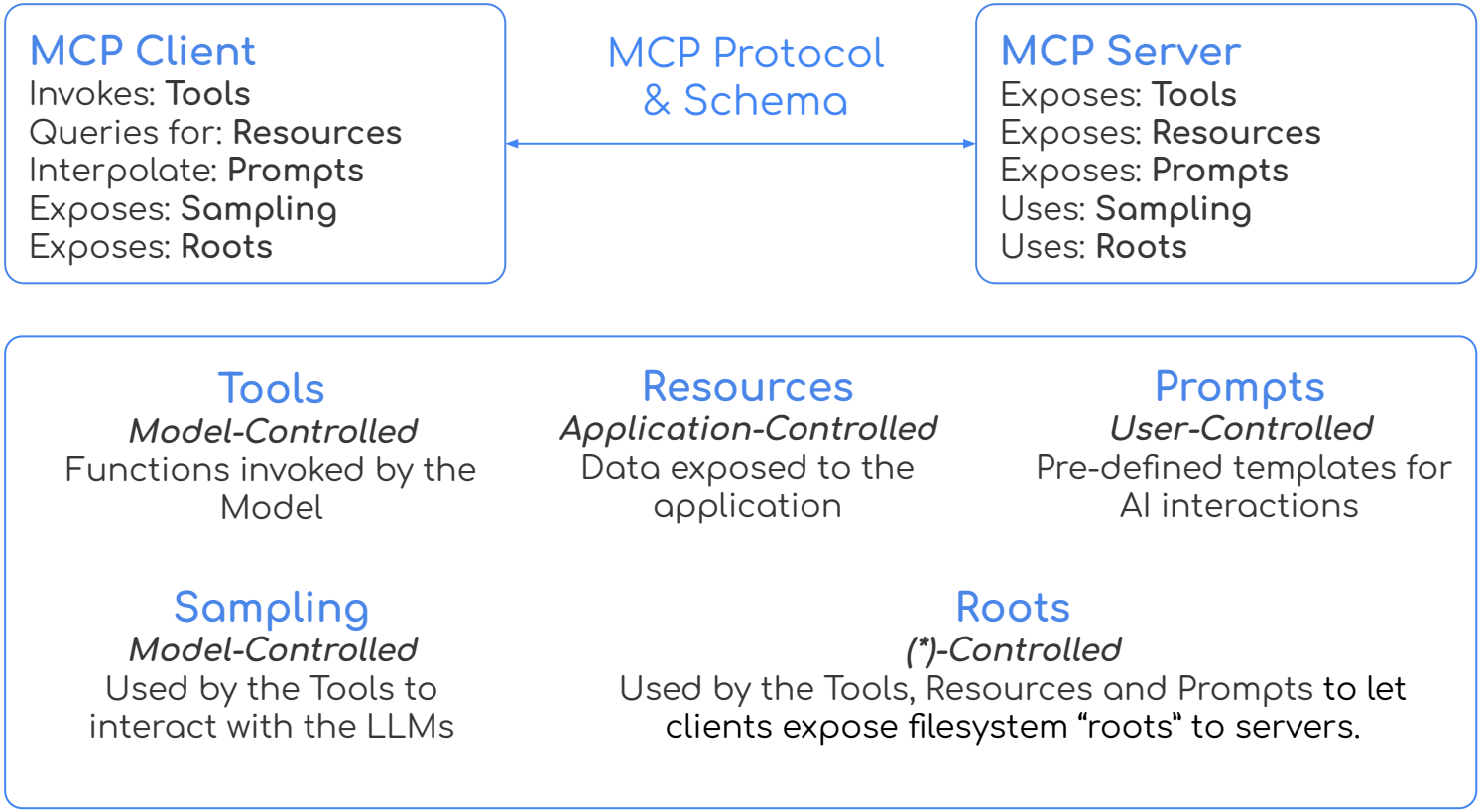
- To be able to use **Tools** written in other languages?

- To have Spring AI Tools used in non-Java envs?

MCP Components



MCP Deep-Dive



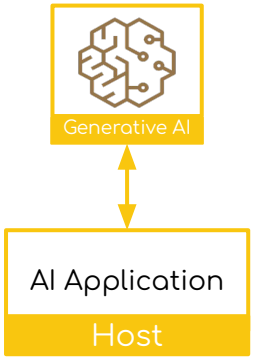
What We'll Cover?

- Introduction
- LLM as a System
- Agentic AI System
- Model Context Protocol

Unified way to connect AI applications and Agents to different **data sources, tools, ...**

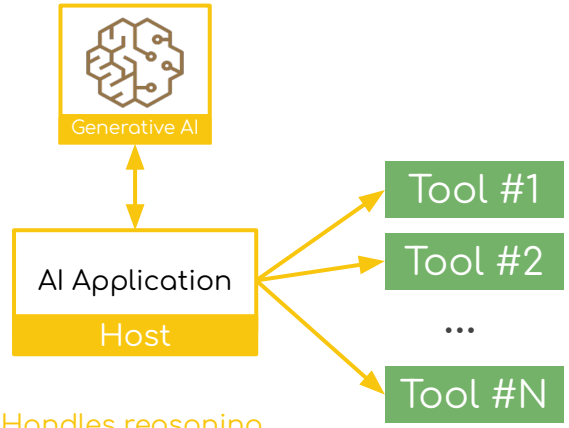
The Road ...

1 LLM



Text-In /
Text-Out

2 LLM + Tool Calling



Handles reasoning and planning. Decides which Actions to take

Brain LLM

Agent

Body Tools

Equipment to Interact with the environment

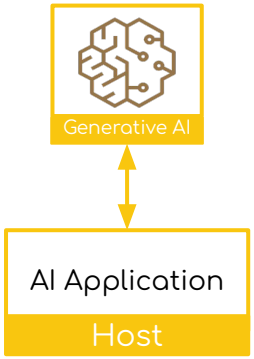
Wouldn’t it be nice ...

- To be able to use **Tools** written in other languages?

- To have Spring AI Tools used in non-Java envs?

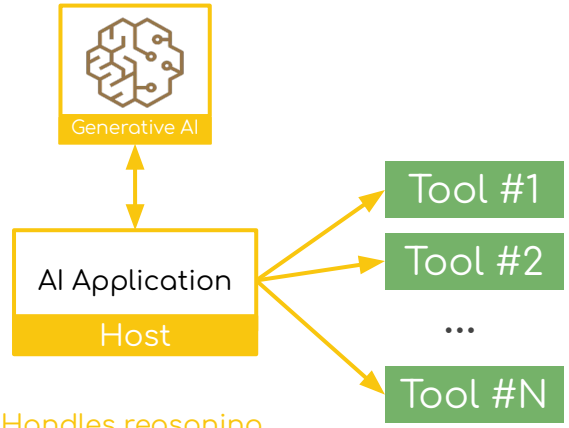
The Road to Model Context Protocol (MCP)

1 LLM



Text-In / Text-Out

2 LLM + Tool Calling



Handles reasoning and planning. Decides which Actions to take

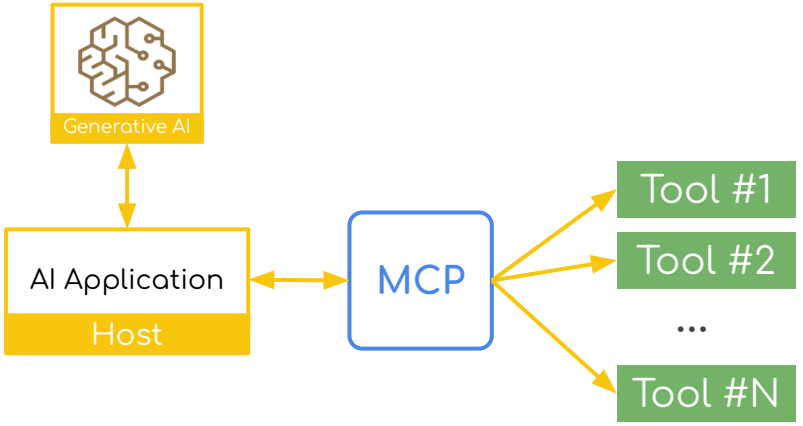
Brain LLM

Agent

Body Tools

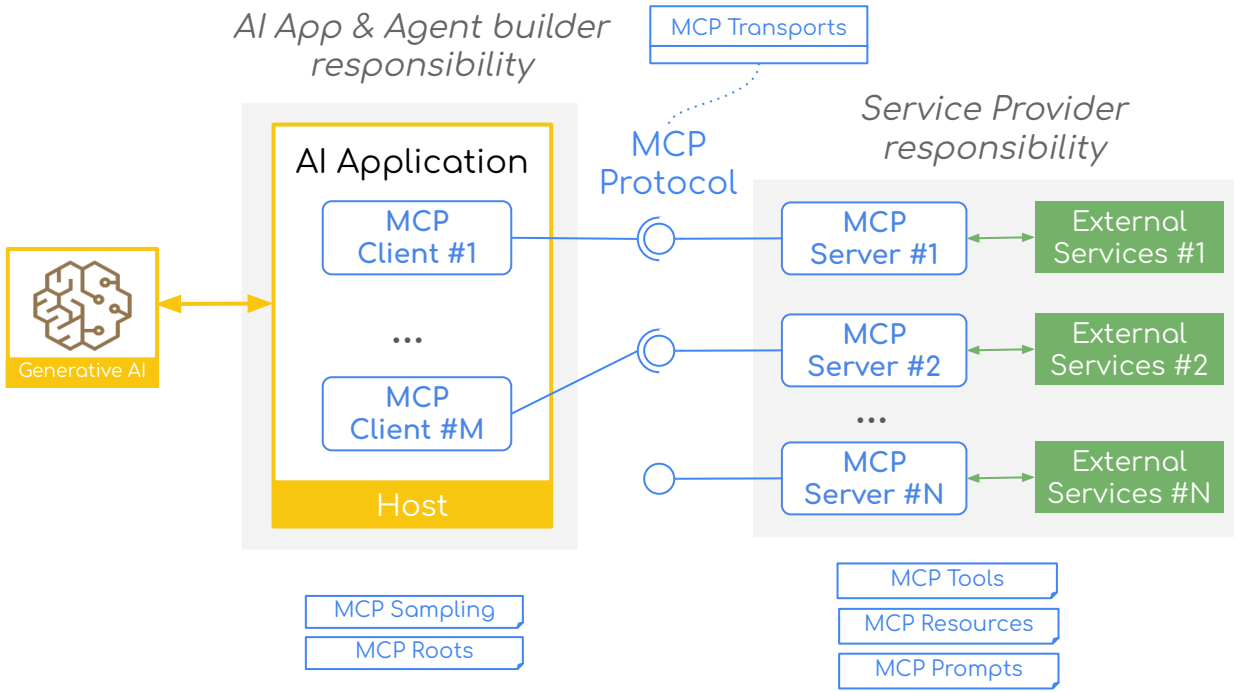
Equipment to Interact with the environment

3 LLM + MCP

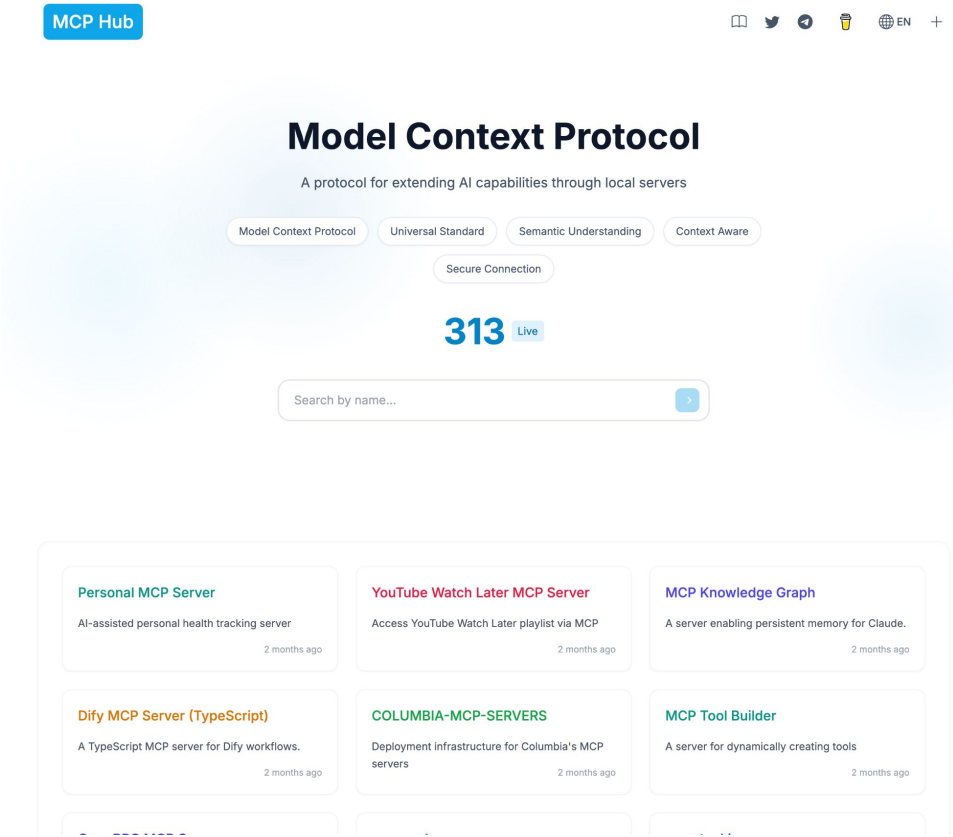


Unified way to connect **AI applications** to different **data sources, tools, ...**

MCP Components

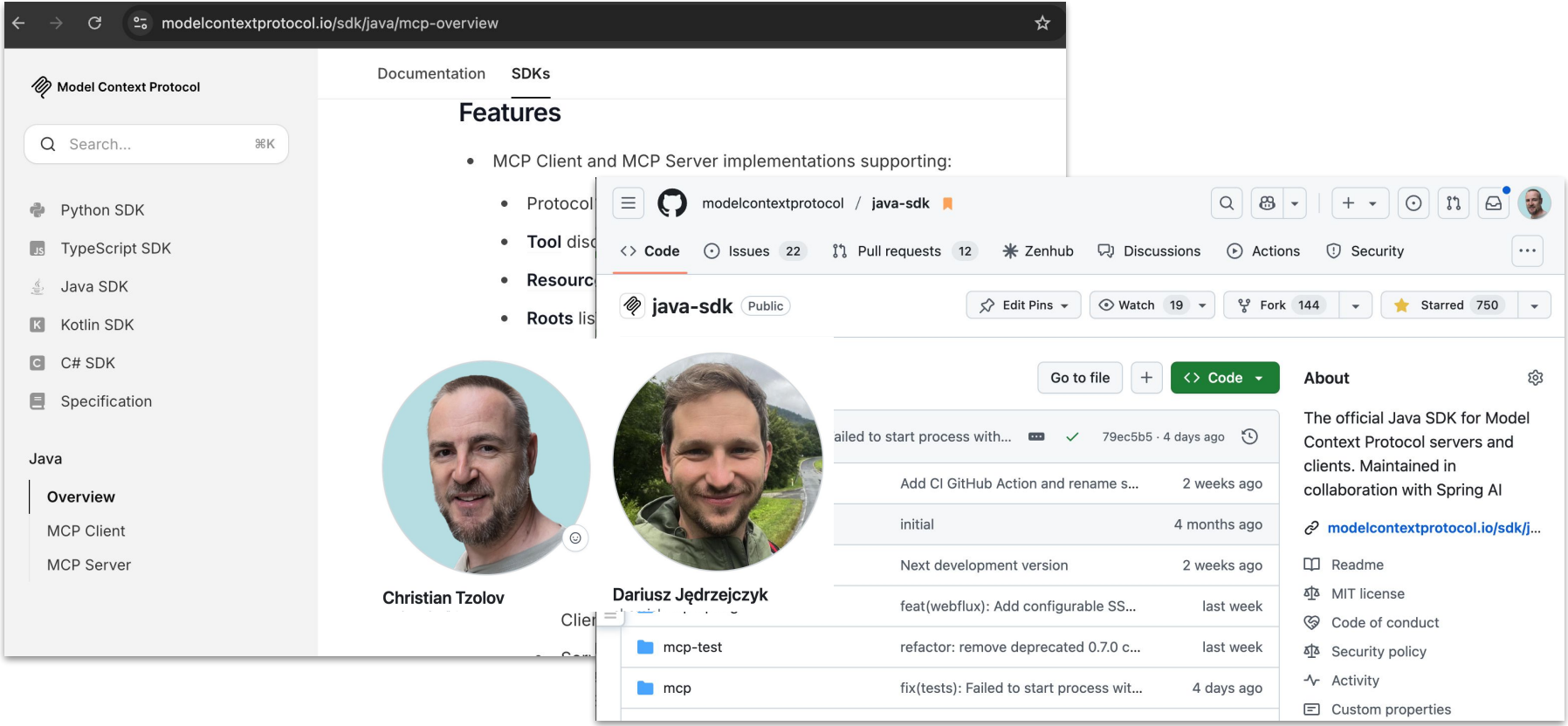


MCP Client & Server Ecosystem

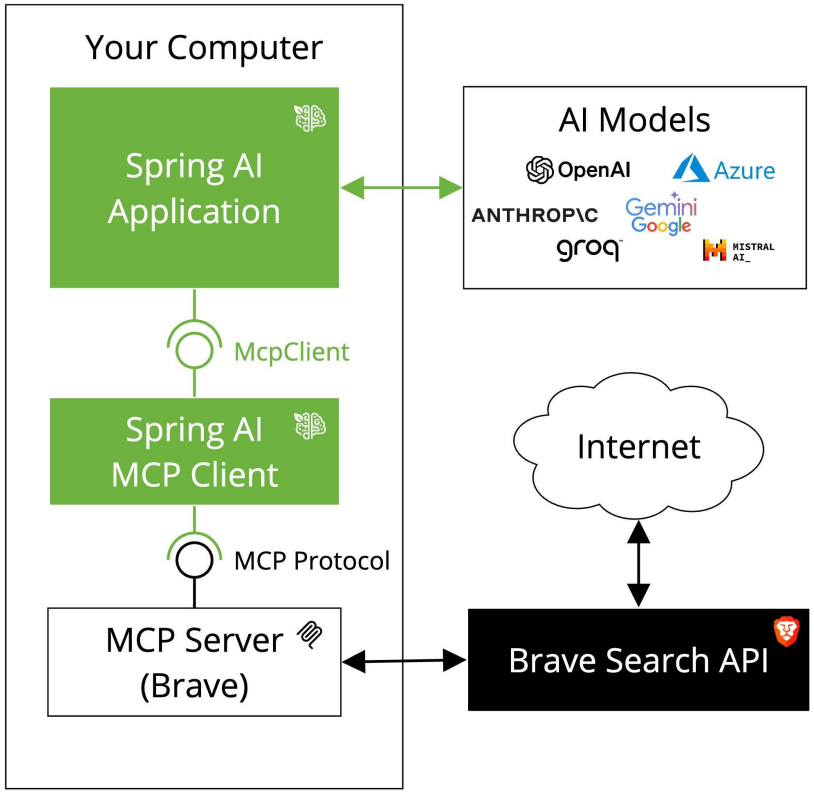


MCP Client & Server Ecosystem

MCP Official Java SDK



MCP Spring AI - WebSearch & Filesystem Tools



mcp-servers-config.json

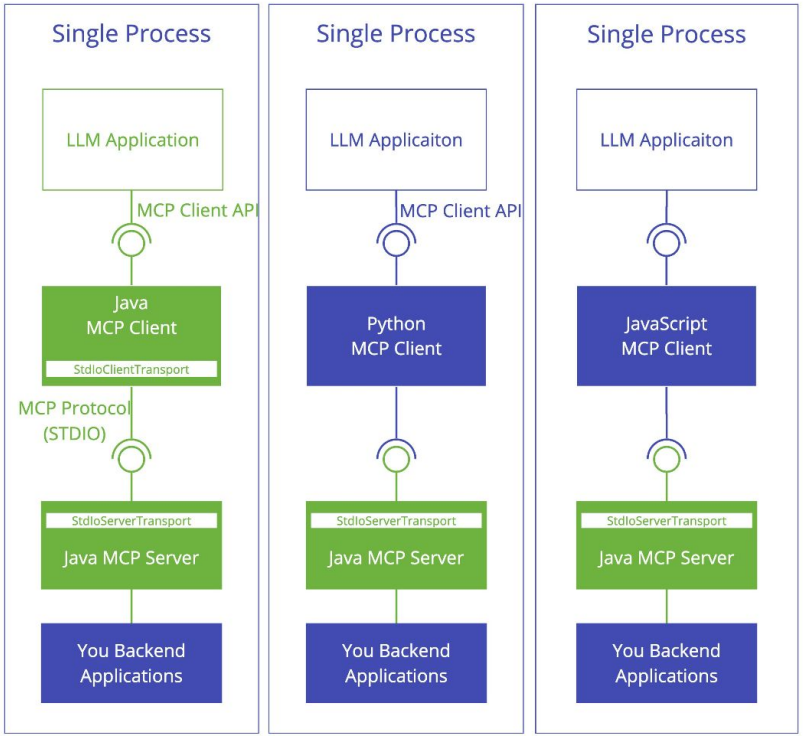
```
{
  "mcpServers": {
    "brave-search": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-brave-search"
      ],
      "env": {
      }
    },
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/Users/christiantzolov/Desktop/tmp"
      ]
```

Application.java

```
@Bean
public CommandLineRunner chatbot(ChatClient.Builder chatClientBuilder, ToolCallbackProvider tools)

  return args -> {
    var output = chatClientBuilder.build().prompt()
        .system("You are useful assistant and can perform web searches Brave's search API to reply
        .user("Create a summary about the Talent Arena conference and save it as markdown talent-a
        .tools(tools)
        .call()
        .content();
  };
}
```

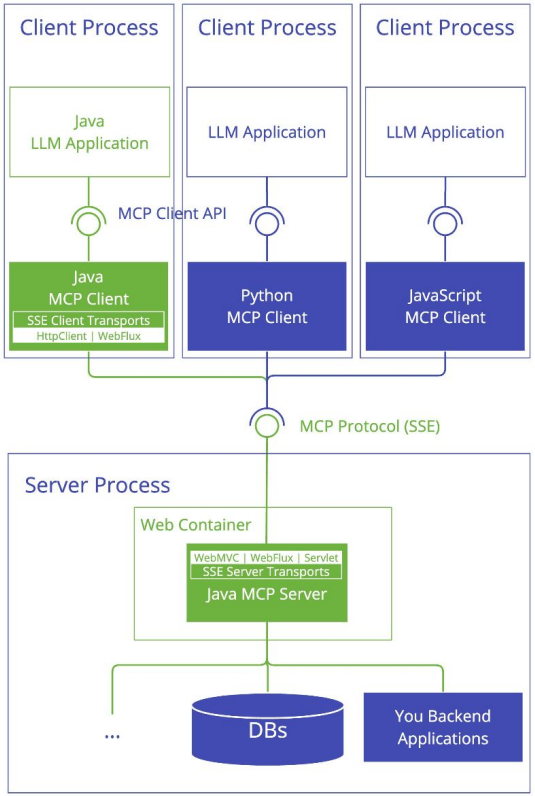
MCP Transports - STDIO

- The client launches the MCP server as a subprocess
- JSON-RPC messages over (stdin/stdout)



MCP Transports - HTTP SSE

- Server is independent process
- Multiple client connections
- SSE endpoint - connection establishment & server to client communication
- HTTP POST endpoint - client to server communication



MCP Transports - Java SDK

Default MCP transports

- No external web frameworks required
- Stdio, client/server transports
- **Java HttpClient-based SSE client**
- **Servlet-based SSE server**

Optional Spring-based transports

- WebFlux SSE client and server transports
- WebMVC SSE server transport

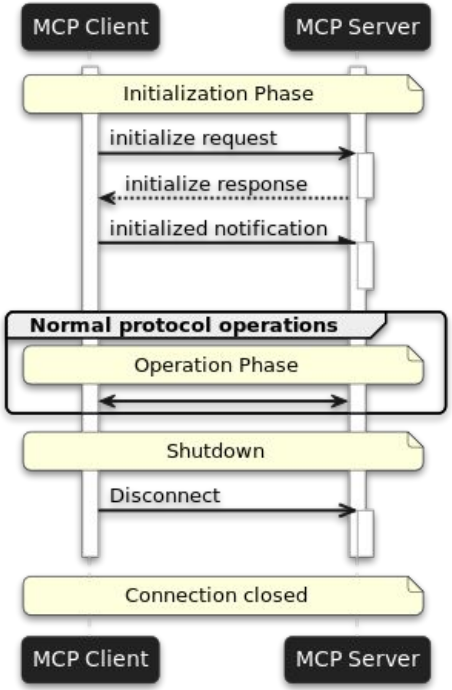
MCP Java SDK - Initialization

MCP Client

```java
McpSyncClient client = McpClient.sync(transport)
    .requestTimeout(Duration.ofSeconds(10))
    .capabilities(ClientCapabilities.builder()
        .roots(true)      // Enable roots capability
        .sampling()       // Enable sampling capability
        .build())
    .sampling(request -> new CreateMessageResult(response))
    .build();

// Initialize connection
client.initialize();
```

MCP Server

```java
// Create a server with custom configuration
McpSyncServer syncServer = McpServer.sync(transportProvider)
    .serverInfo("my-server", "1.0.0")
    .capabilities(ServerCapabilities.builder()
        .resources(true)     // Enable resource support
        .tools(true)         // Enable tool support
        .prompts(true)       // Enable prompt support
        .logging()           // Enable logging support
        .build())
    .build();
```



MCP Java SDK - Initialization -Spring AI MCP

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-mcp-client</artifactId>
</dependency>
```

Alternatively, you can configure stdio connections using an external JSON file using the Claude Desktop format:

```
spring:
  ai:
    mcp:
      client:
        stdio:
          servers-configuration: classpath:mcp-servers.json
```

The Claude Desktop format looks like this:

```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/Users/username/Desktop",
        "/Users/username/Downloads"
      ]
    }
  }
}
```

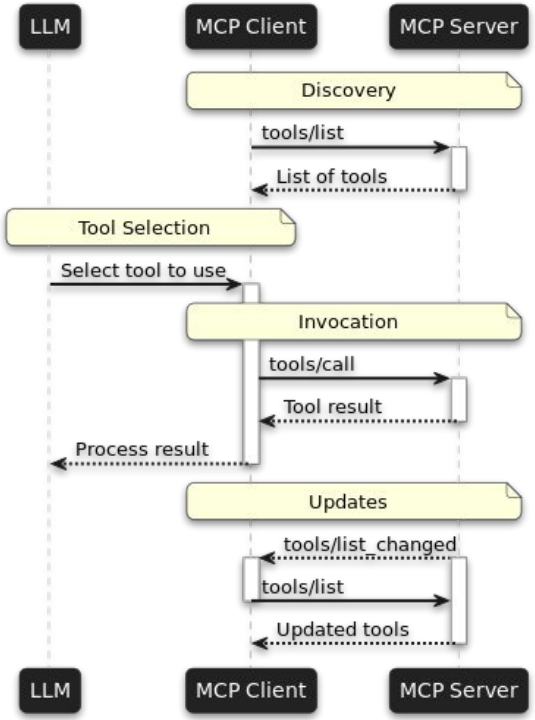
MCP Java SDK - Tools

MCP Client

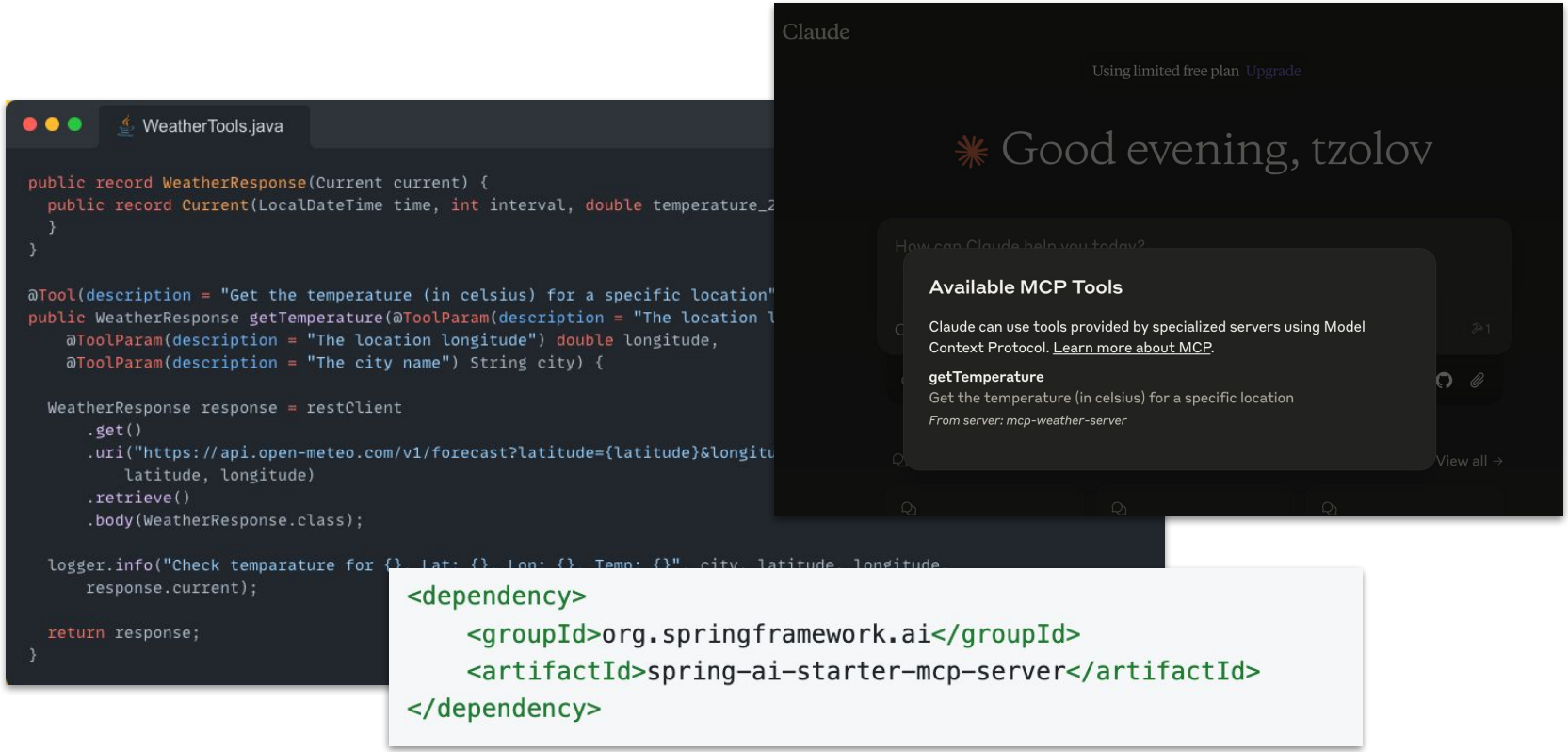
```
var tools = client.listTools();
tools.forEach(tool -> System.out.println(tool.getName()));

// Execute a tool with parameters
var result = client.callTool("calculator", Map.of(
    "operation", "add",
    "a", 1,
    "b", 2
```

MCP Server

```
var syncToolSpecification = new McpServerFeatures.SyncToo
    new Tool("calculator", "Basic calculator", schema),
    (exchange, arguments) -> {
        // Tool implementation
        return new CallToolResult(result, false);
    }
```



MCP Java SDK - Tools - Spring AI MCP - Demo




```xml
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-mcp-server</artifactId>
</dependency>
```

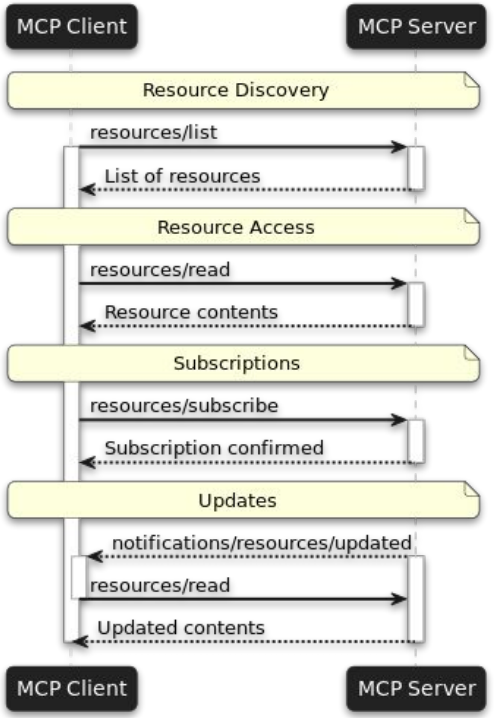
MCP Java SDK - Resources

MCP Client

```
var resources = client.listResources();
resources.forEach(resource -> System.out.println(resource.get

// Retrieve resource content using a URI template
var content = client.getResource("file", Map.of(
    "path", "/path/to/file.txt"
));
```

MCP Server

```
var syncResourceSpecification = new McpServerFeatures.syncResourc
    new Resource("custom://resource", "name", "description", "mim
    (exchange, request) -> {
        // Resource read implementation
        return new ReadResourceResult(contents);
    }
);
```



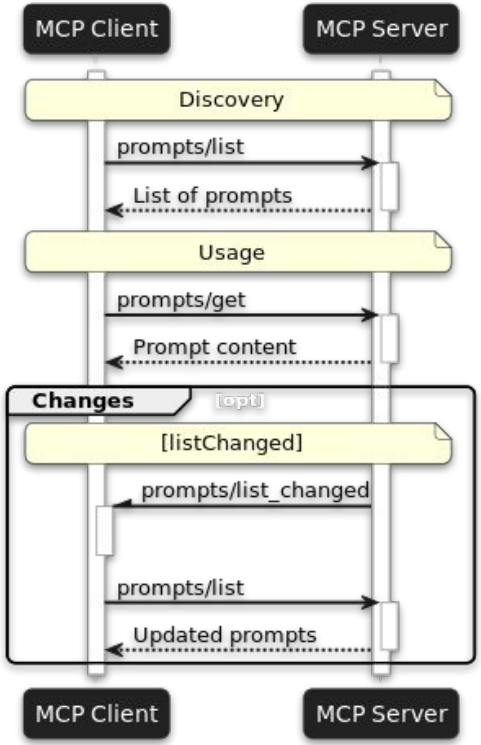
MCP Java SDK - Prompts

MCP Client

```java
// List available prompt templates
var prompts = client.listPrompts();
prompts.forEach(prompt -> System.out.println(prompt.getName()));

// Execute a prompt template with parameters
var response = client.executePrompt("echo", Map.of(
    "text", "Hello, World!"
));
```

MCP Server

```java
var syncPromptSpecification = new McpServerFeatures.syncPromptSpec
    new Prompt("greeting", "description", List.of(
        new PromptArgument("name", "description", true)
    )),
    (exchange, request) -> {
        // Prompt implementation
        return new GetPromptResult(description, messages);
    }
);
```



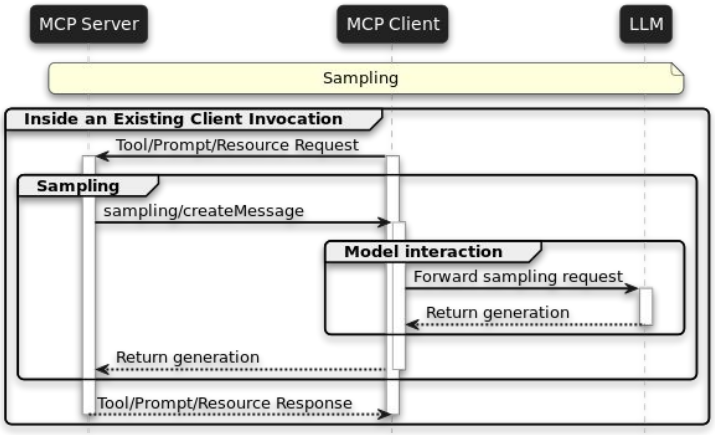
MCP Java SDK - Sampling

MCP Client

```
McpSyncServer server = McpServer.sync(transportProvider)
    .serverInfo("my-server", "1.0.0")
    .build();

// Define a tool that uses sampling
var calculatorTool = new McpServerFeatures.SyncToolSpecification(
    new Tool("ai-calculator", "Performs calculations using AI", schema),
    (exchange, arguments) -> {
        // Check if client supports sampling
        if (exchange.getClientCapabilities().sampling() == null) {
            return new CallToolResult("Client does not support AI capabilitie
        }

        // Create a sampling request
        McpSchema.CreateMessageRequest request = McpSchema.CreateMessageReque
            .content(new McpSchema.TextContent("Calculate: " + arguments.get(
            .modelPreferences(McpSchema.ModelPreferences.builder()
                .hints(List.of(
                    McpSchema.ModelHint.of("claude-3-sonnet"),
                    McpSchema.ModelHint.of("claude")
                ))
                .intelligencePriority(0.8)  // Prioritize intelligence
                .speedPriority(0.5)         // Moderate speed importance
                .build())
            .systemPrompt("You are a helpful calculator assistant. Provide on
            .maxTokens(100)
            .build();

        // Request sampling from the client
        McpSchema.CreateMessageResult result = exchange.createMessage(request

        // Process the result
        String answer = result.content().text();
        return new CallToolResult(answer, false);
    }
);

// Add the tool to the server
server.addTool(calculatorTool);
```

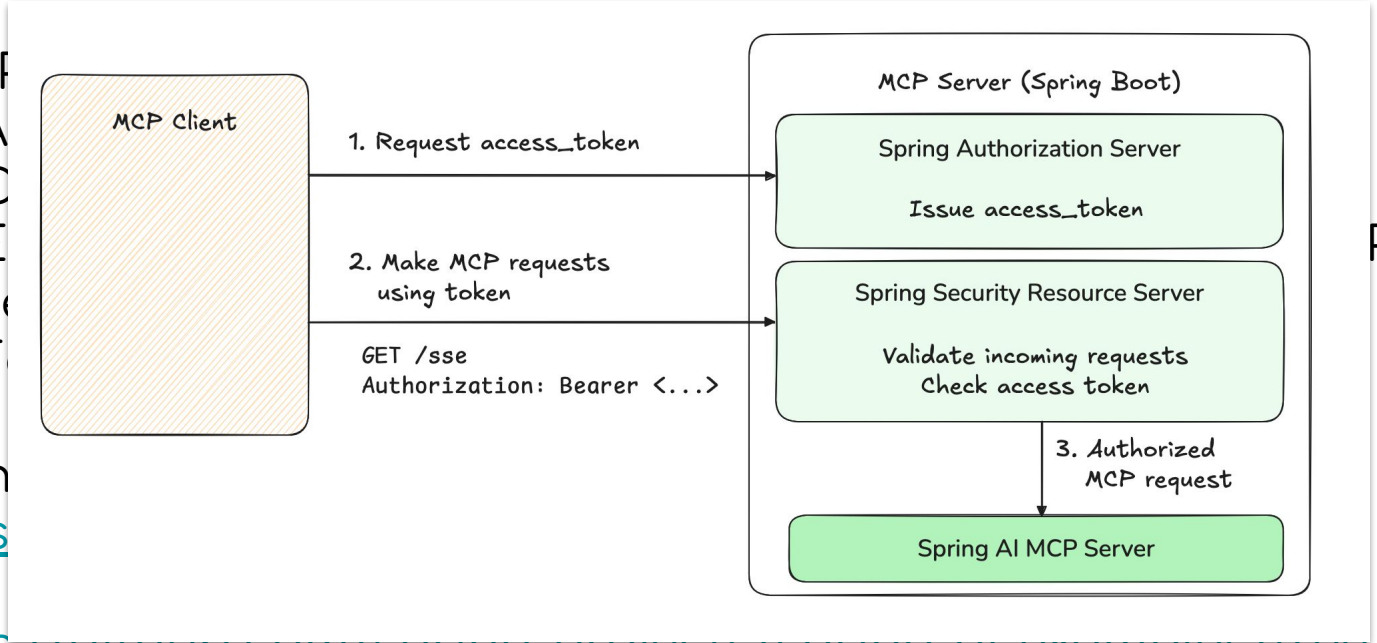
MCP Server

```
Function<CreateMessageRequest, CreateMessageResult> samplingHandler =
    // Sampling implementation that interfaces with LLM
    return new CreateMessageResult(response);
};

// Create client with sampling support
var client = McpClient.sync(transport)
    .capabilities(ClientCapabilities.builder()
        .sampling()
        .build())
    .sampling(samplingHandler)
    .build();
```



MCP Authorization

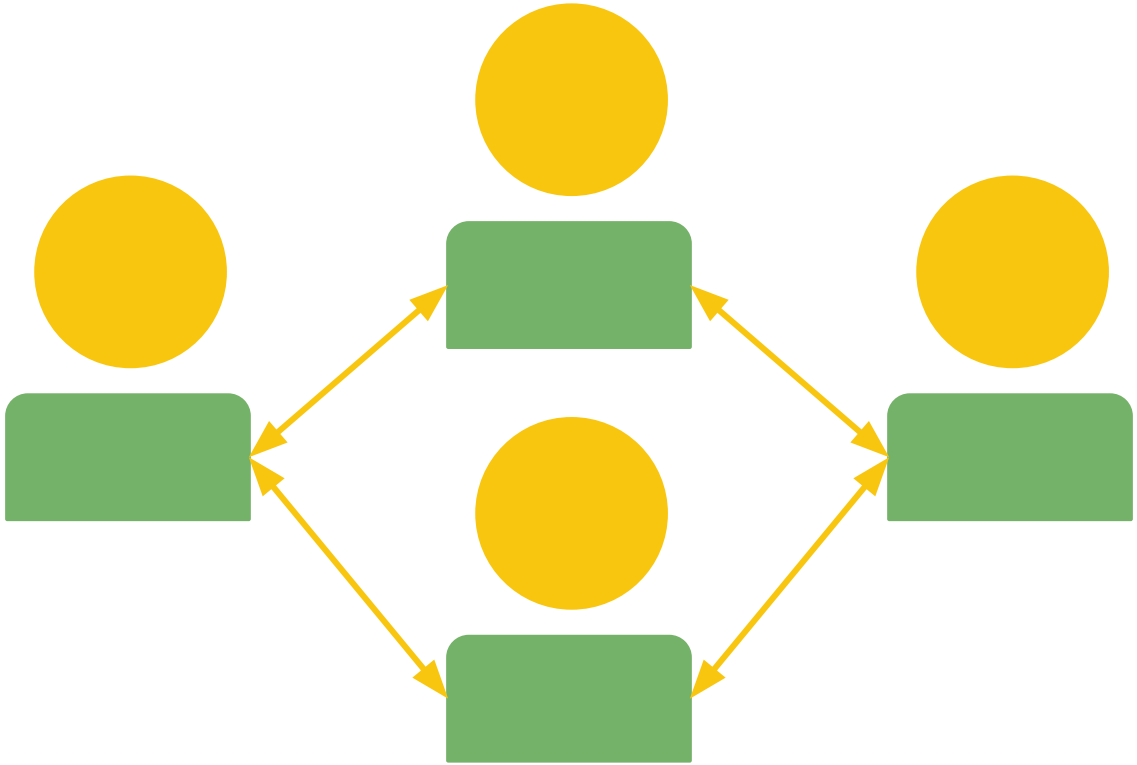
- New F
  - A
  - C
  - E
    s
  - F
- Sprin
  https



P

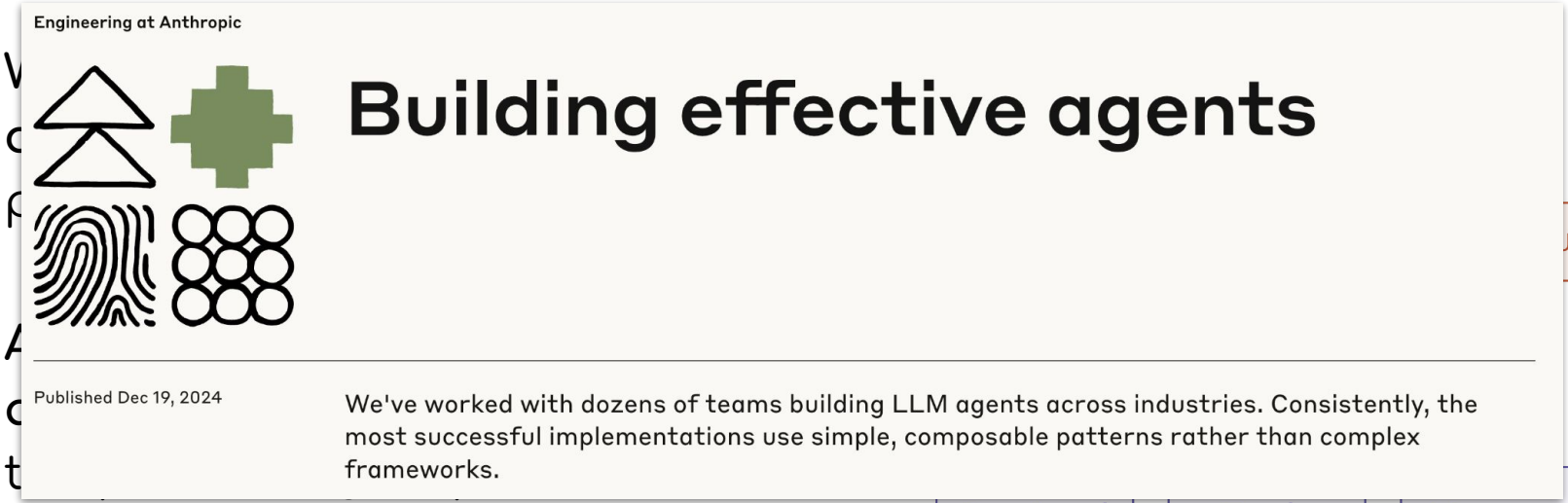
https://github.com/spring-projects/spring-ai-examples/tree/main/model-context-protocol/weather/starter-webmvc-oauth2-server

Agentic Systems



Agentic Systems Types

- W
  c
  p
- A
  c
  t
  accomplishing the tasks - e.g **autonomous**



UT

Retrieval

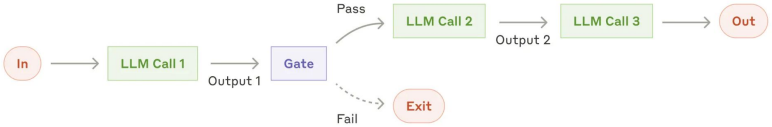
Tools

Memory

Agentic Workflows

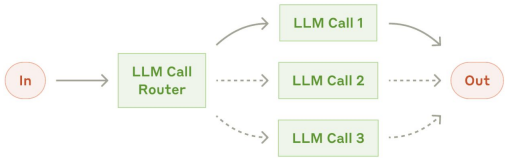
Chain Workflow

Break complex tasks down into simpler, more manageable steps



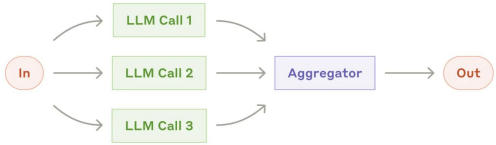
Routing Workflow

Complex tasks with different input types, handled by specialized processes. An LLM analyzes the input content and routes it to the specialized handler.



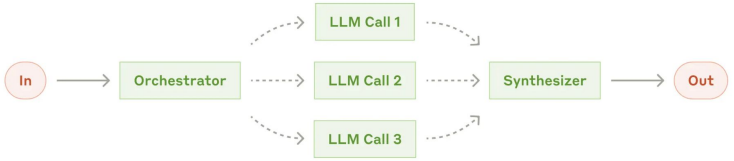
Parallelization Workflow

Work simultaneously on tasks and aggregate outputs



Orchestration - Worker Workflow

Central LLM orchestrates task decomposition. Specialized workers handle specific subtasks

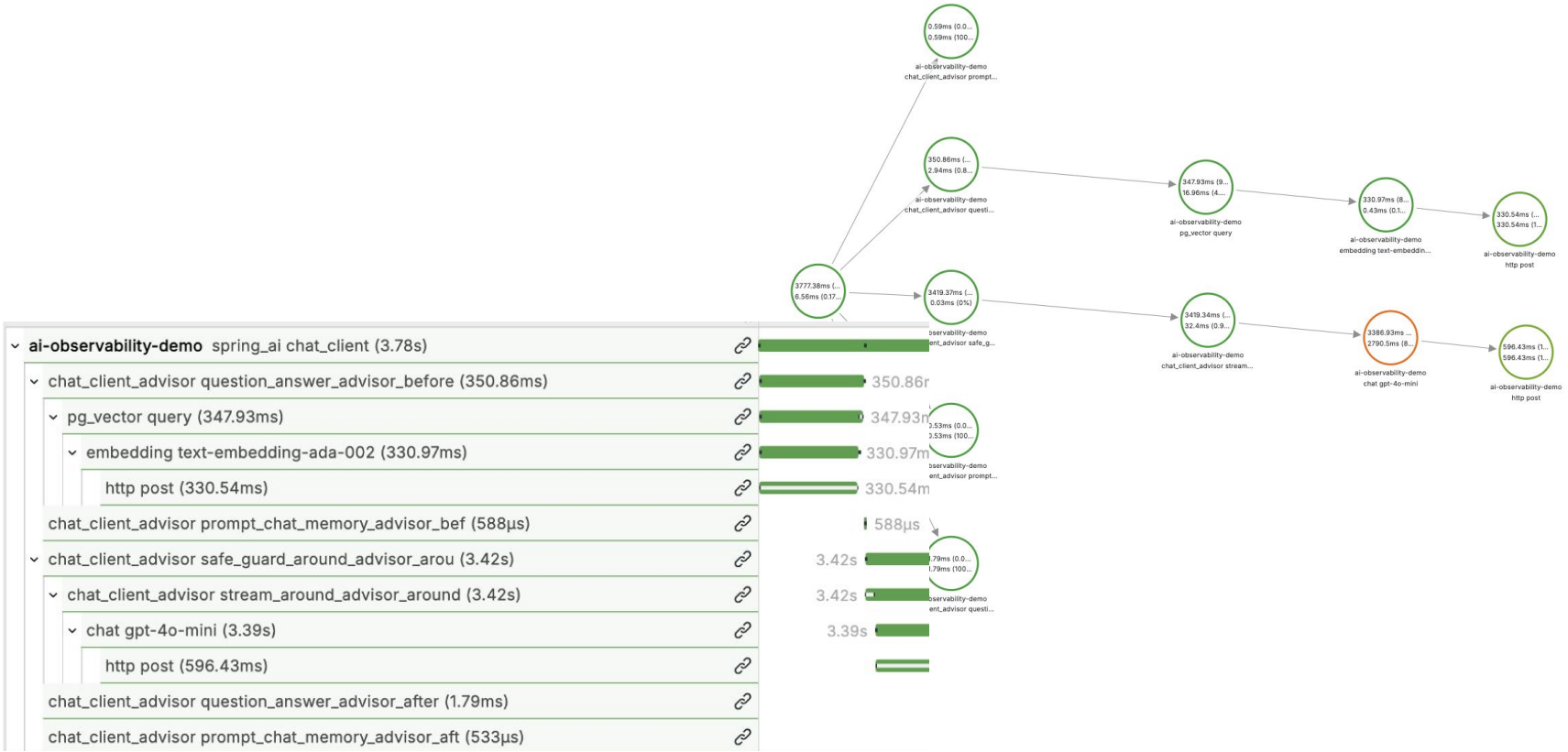


Evaluator - Optimizer Workflow

Dual-LLM process - one LLM generates responses while another provides evaluation and feedback in an iterative loop

Building effective agents: https://www.anthropic.com/engineering/building-effective-agents
Building Effective Agents with Spring AI (Part 1): https://spring.io/blog/2025/01/21/spring-ai-agentic-patterns
GitHub Repo: https://github.com/spring-projects/spring-ai-examples/tree/main/agentic-patterns

Spring AI Observability



Resources

Docs

- https://docs.spring.io/spring-ai/reference/index.html
- https://docs.spring.io/spring-ai/reference/api/mcp/mcp-overview.html
- https://modelcontextprotocol.io/sdk/java/mcp-overview
- https://spring.io/blog/2025/01/21/spring-ai-agentic-patterns

Demos

- https://github.com/spring-projects/spring-ai-examples
- https://github.com/tzolov/talent-arena-demo

Spring AI (overview) Talk - 2024

- https://www.youtube.com/watch?v=vuhMti8B5H0

Thank You

Christian Tzolov  Spring AI
tweeter: @christzolov , linkedin: @tzolov