# Beyond local tools – Deep dive into MCP with Spring AI

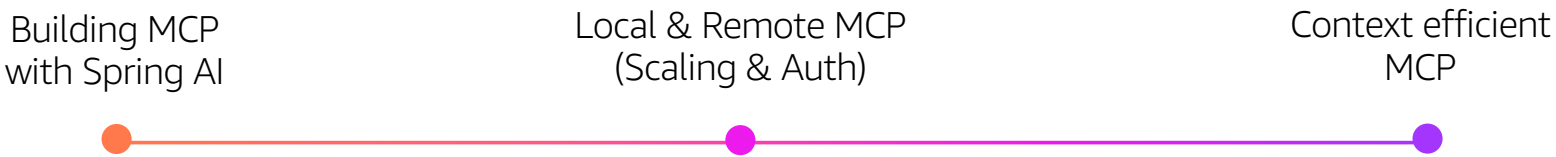
Maximilian Schellhorn

Principal Solutions Architect
AWS – Amazon Web Services

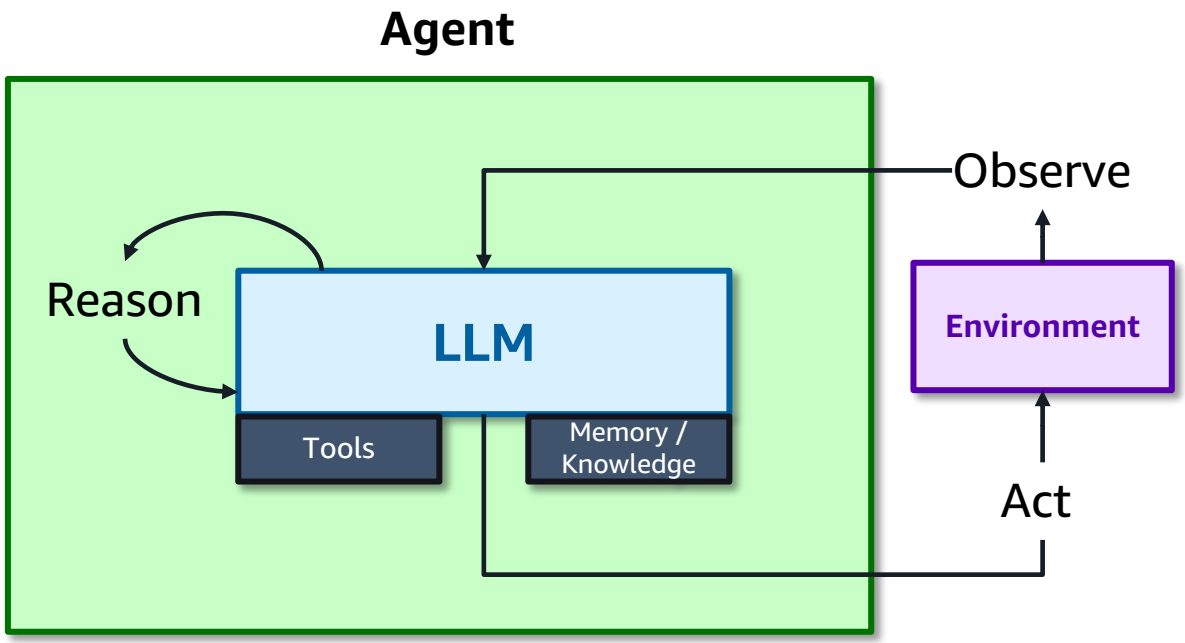
James Ward

Principal Developer Advocate
AWS – Amazon Web Services

# Agenda



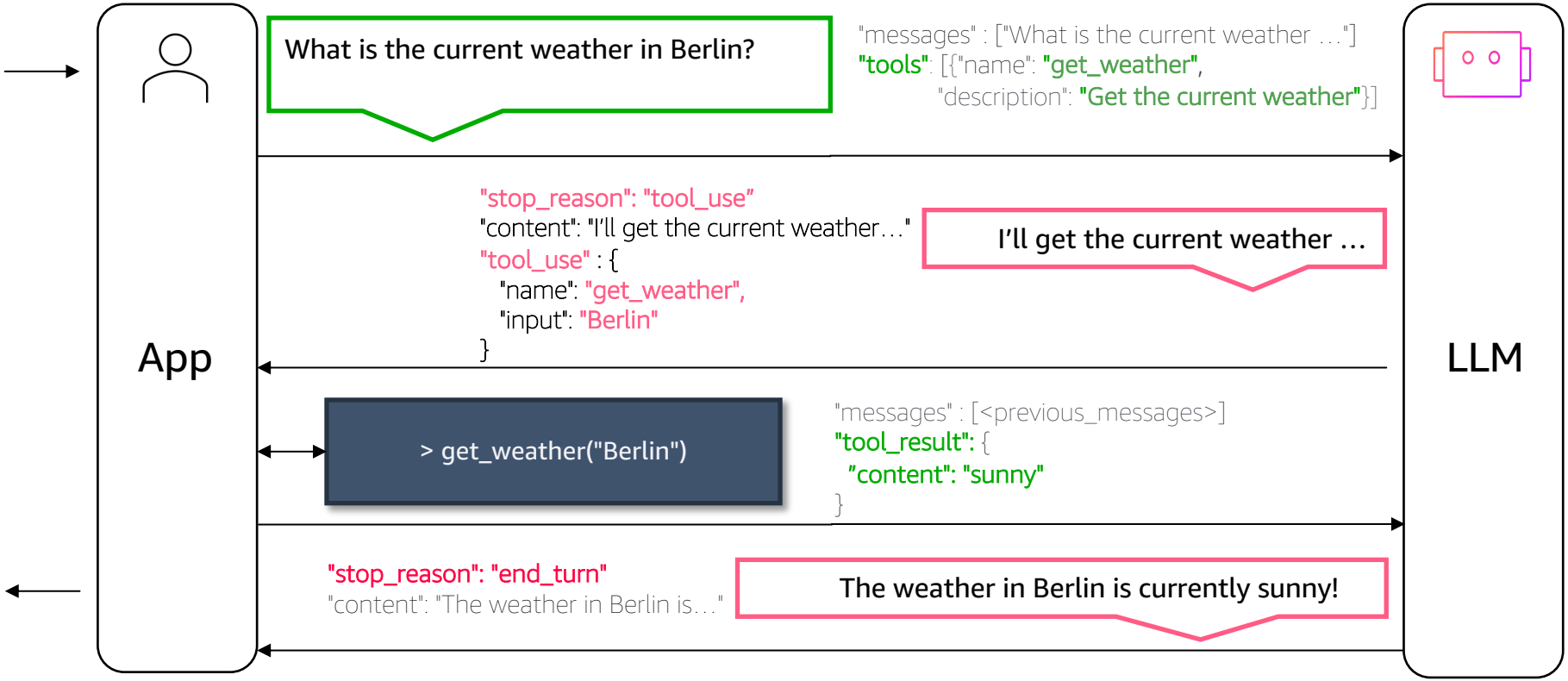
# The augmented LLM



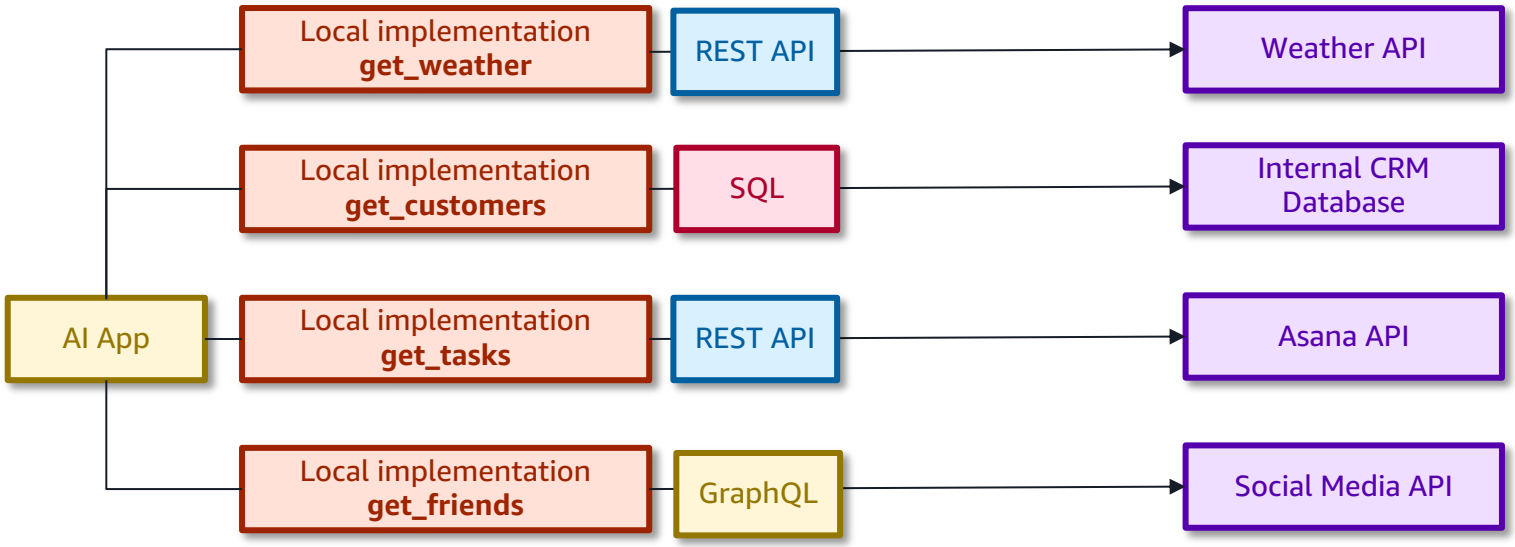



*Inspired by: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-llm-agents*

# Tool calling



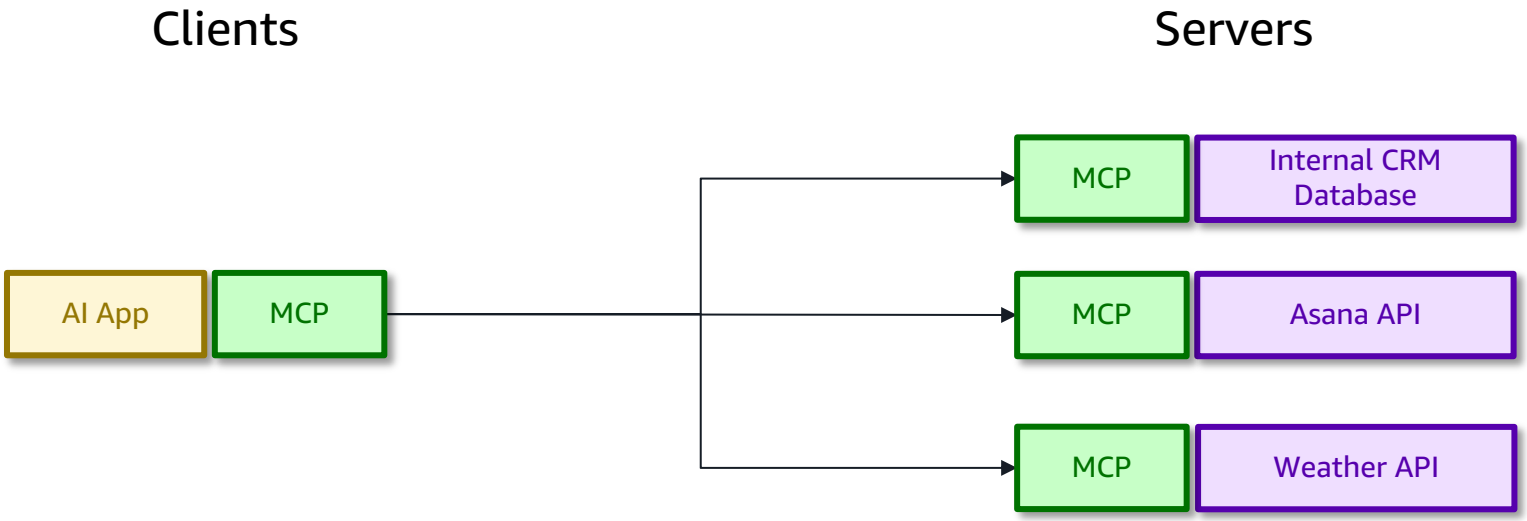
# Challenge: Fragmented AI Development




Builders are responsible to integrate with every tool they want to support

Vendors are dependent on builders to support their tools and protocols

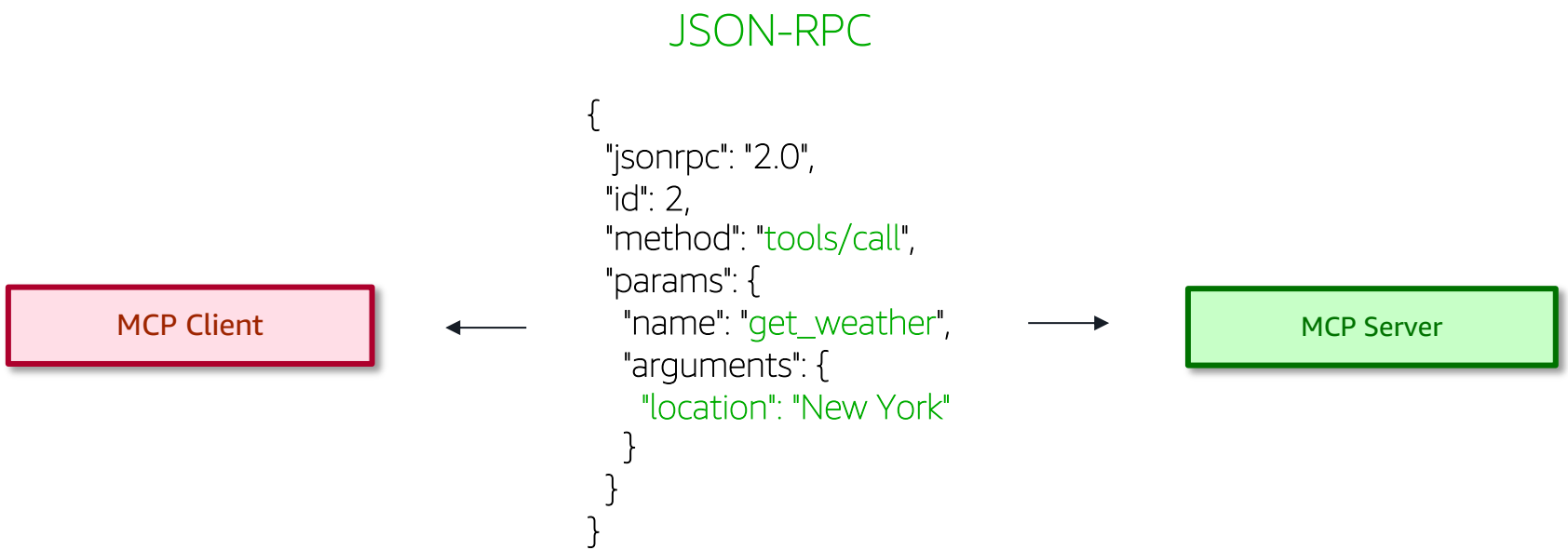
# With MCP: Standardized AI Development



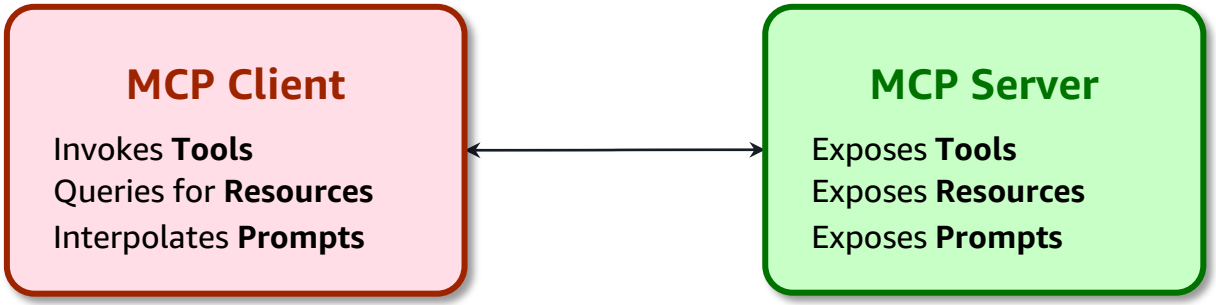

Builders only need to support a single protocol

Vendors provide and maintain MCP-enabled capabilities

# Communication between clients and servers



# Main MCP Components




## Tools

***Model-controlled***

Functions invoked by the model

- Retrieve / search
- Send a message
- Update DB records

## Resources

***Application-controlled***

Data exposed to the application

- Files
- Database Records
- API Responses

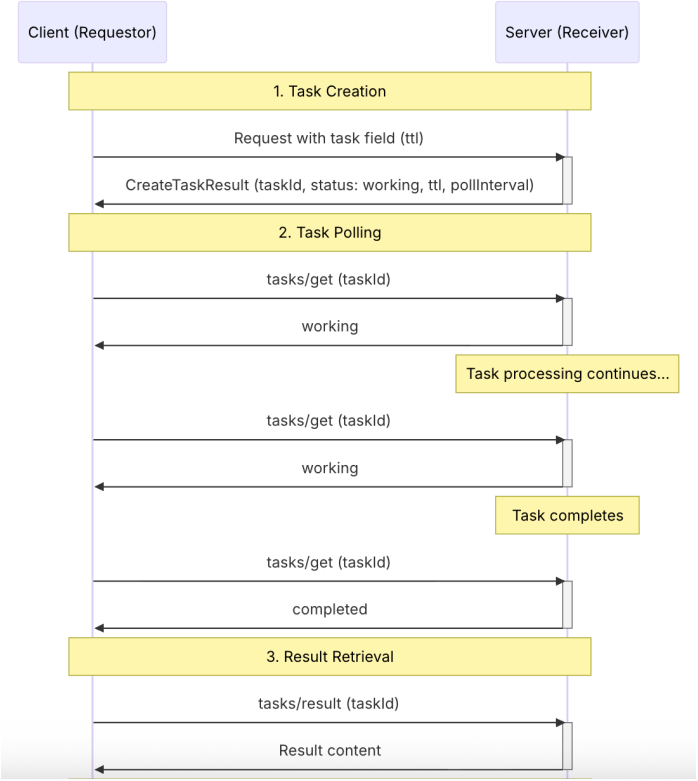
## Prompts

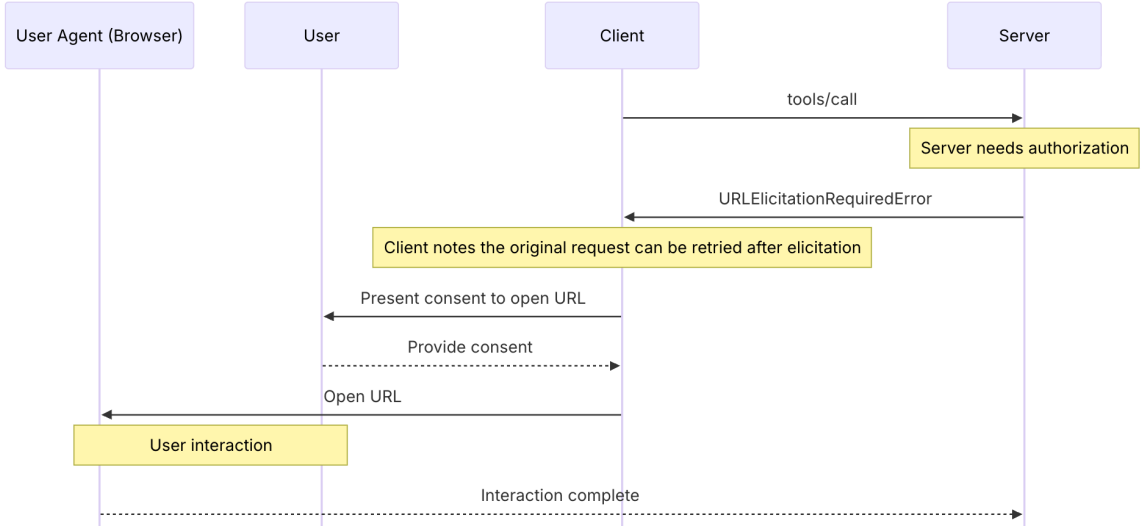
***User-controlled***

Pre-defined templates for AI interactions

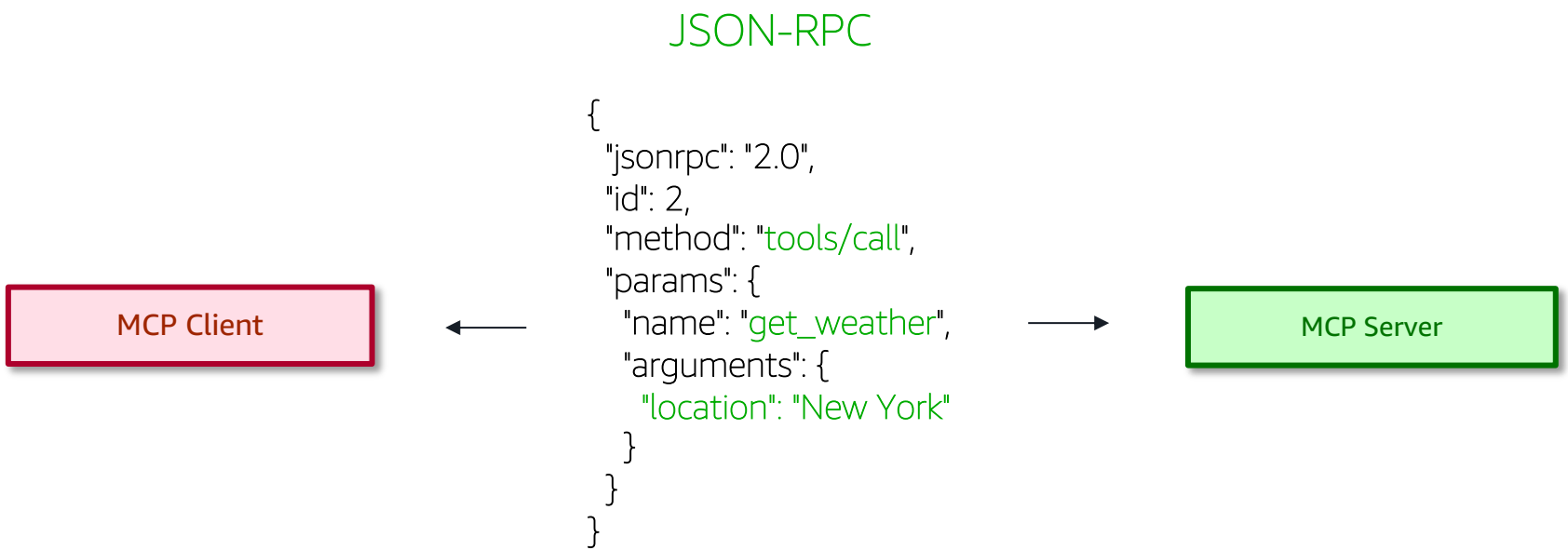
- Document Q&A
- Transcript Summary
- Output as JSON

# Work in progress: Tasks and URL-based elicitation

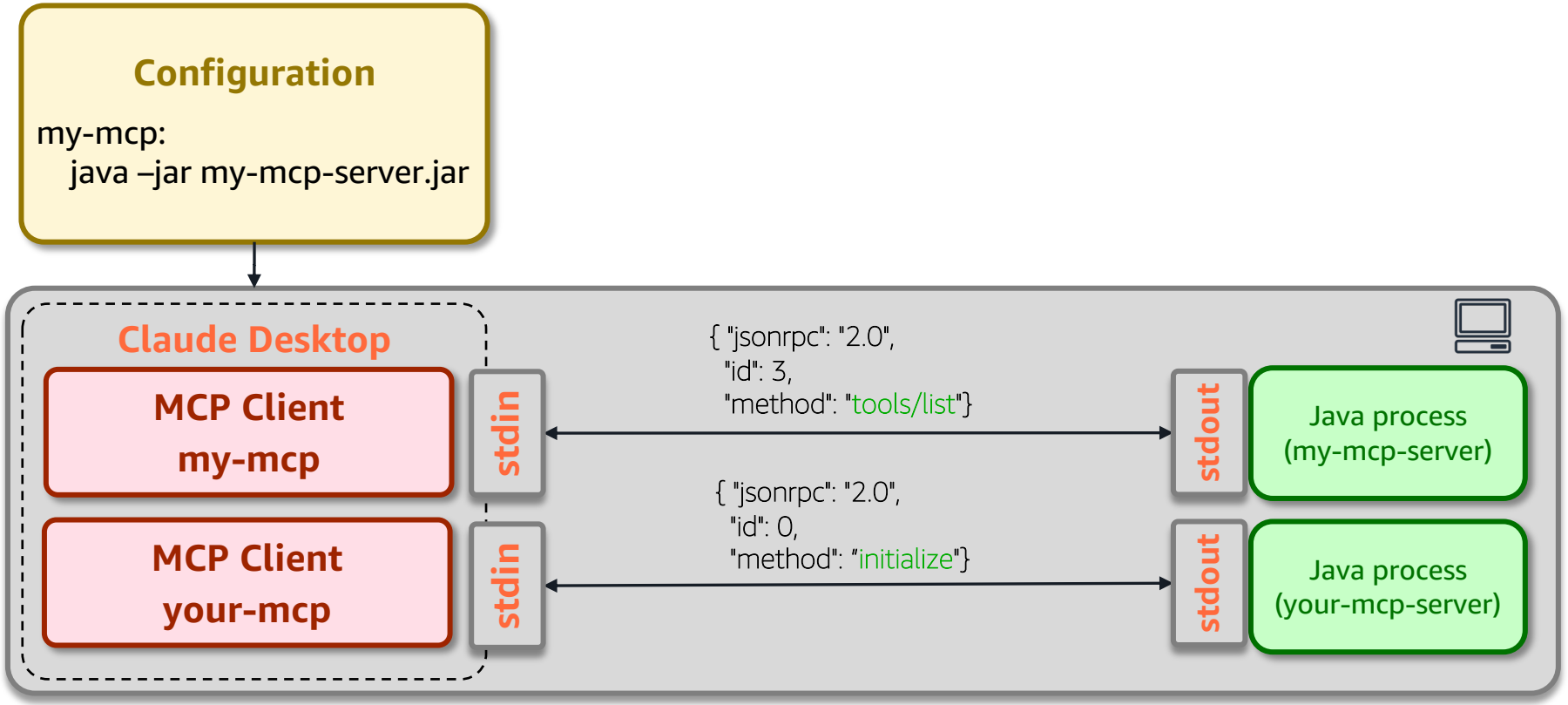


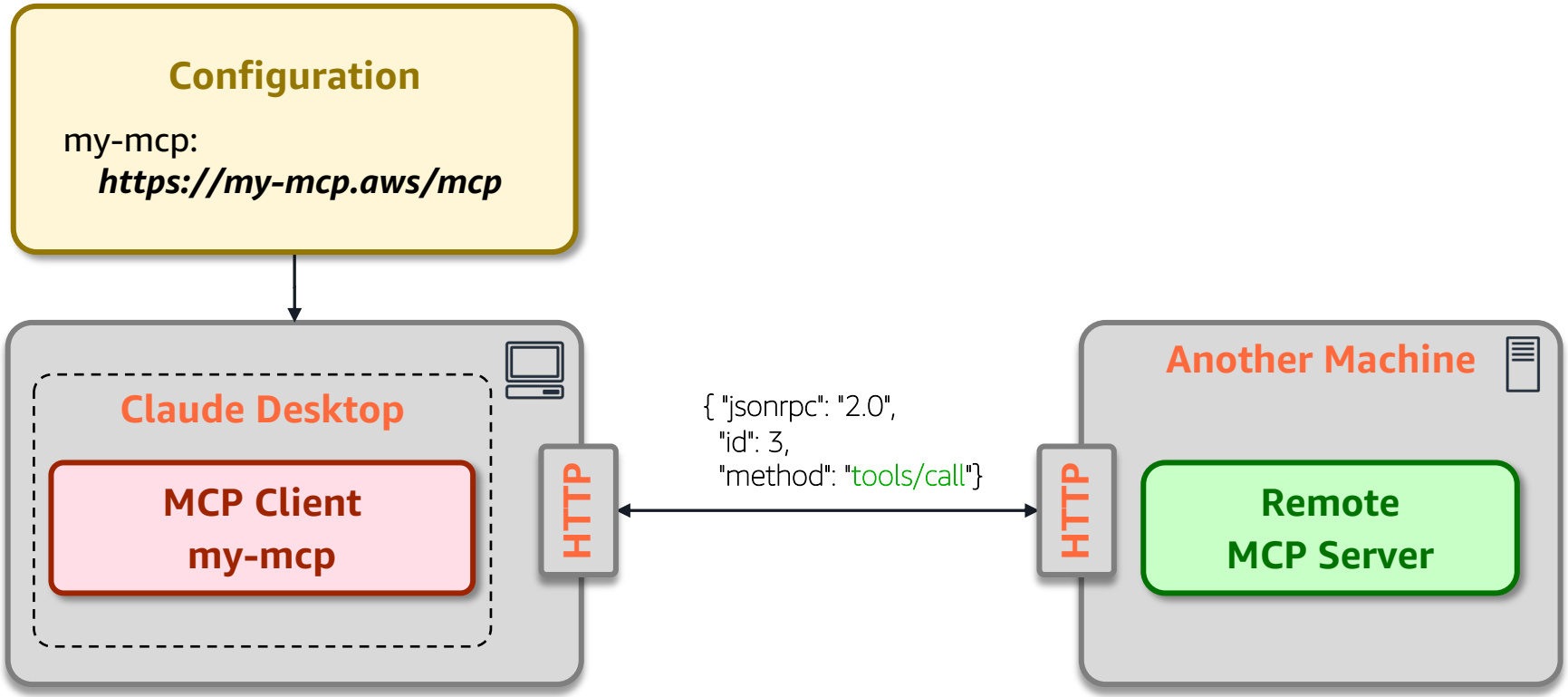
# Transports: Communication between clients and servers



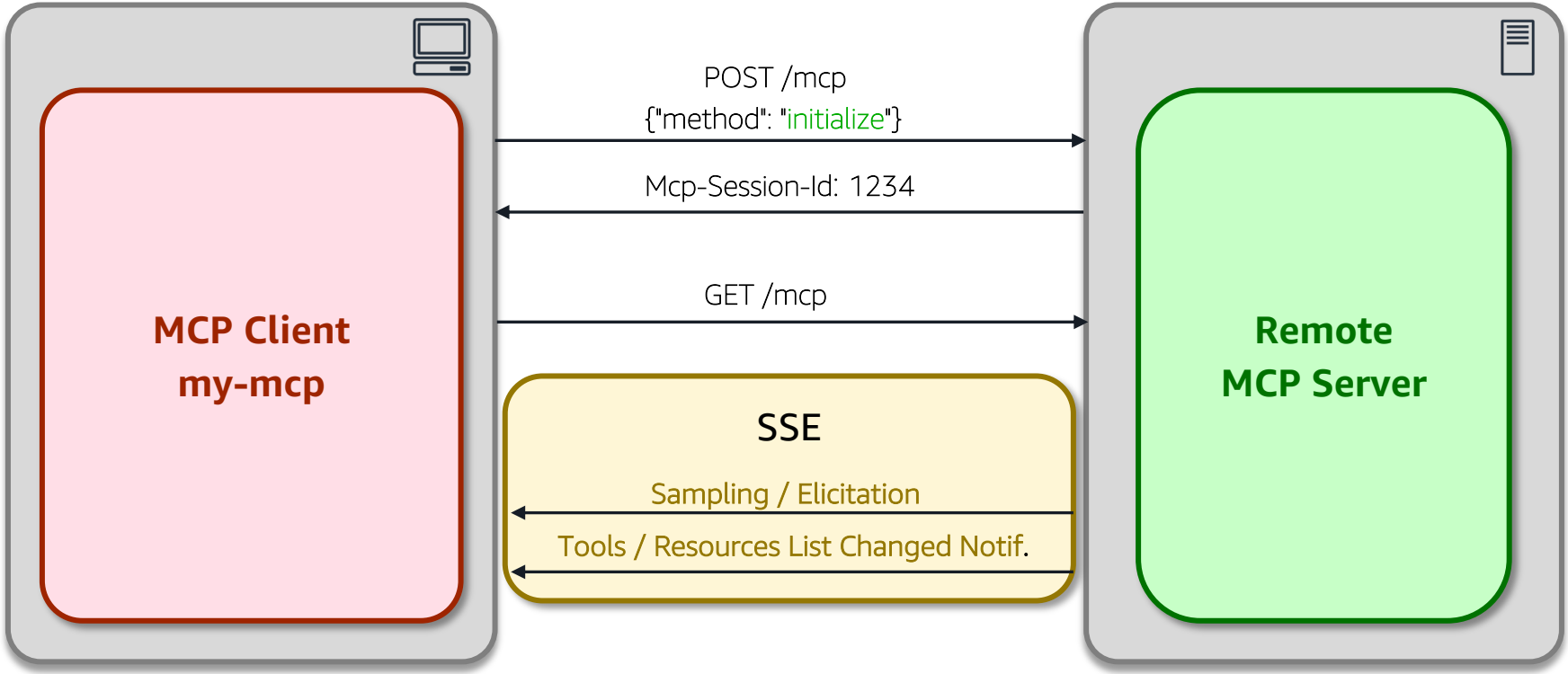
# Transports: Local MCP Servers via stdio



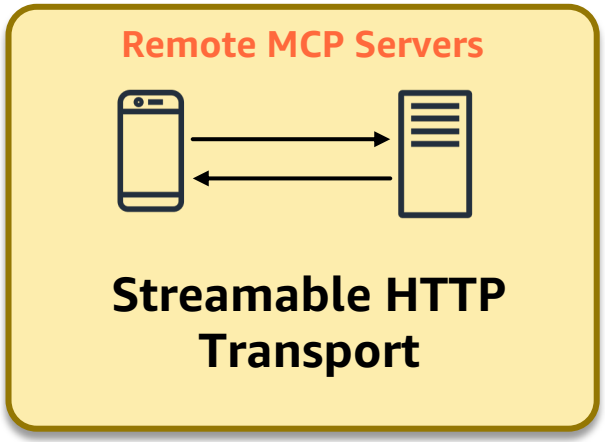
# Transports: Remote MCP Servers via Streamable HTTP



# Upgrading Streamable HTTP to Server Sent Events (SSE)



# Remote MCP Servers



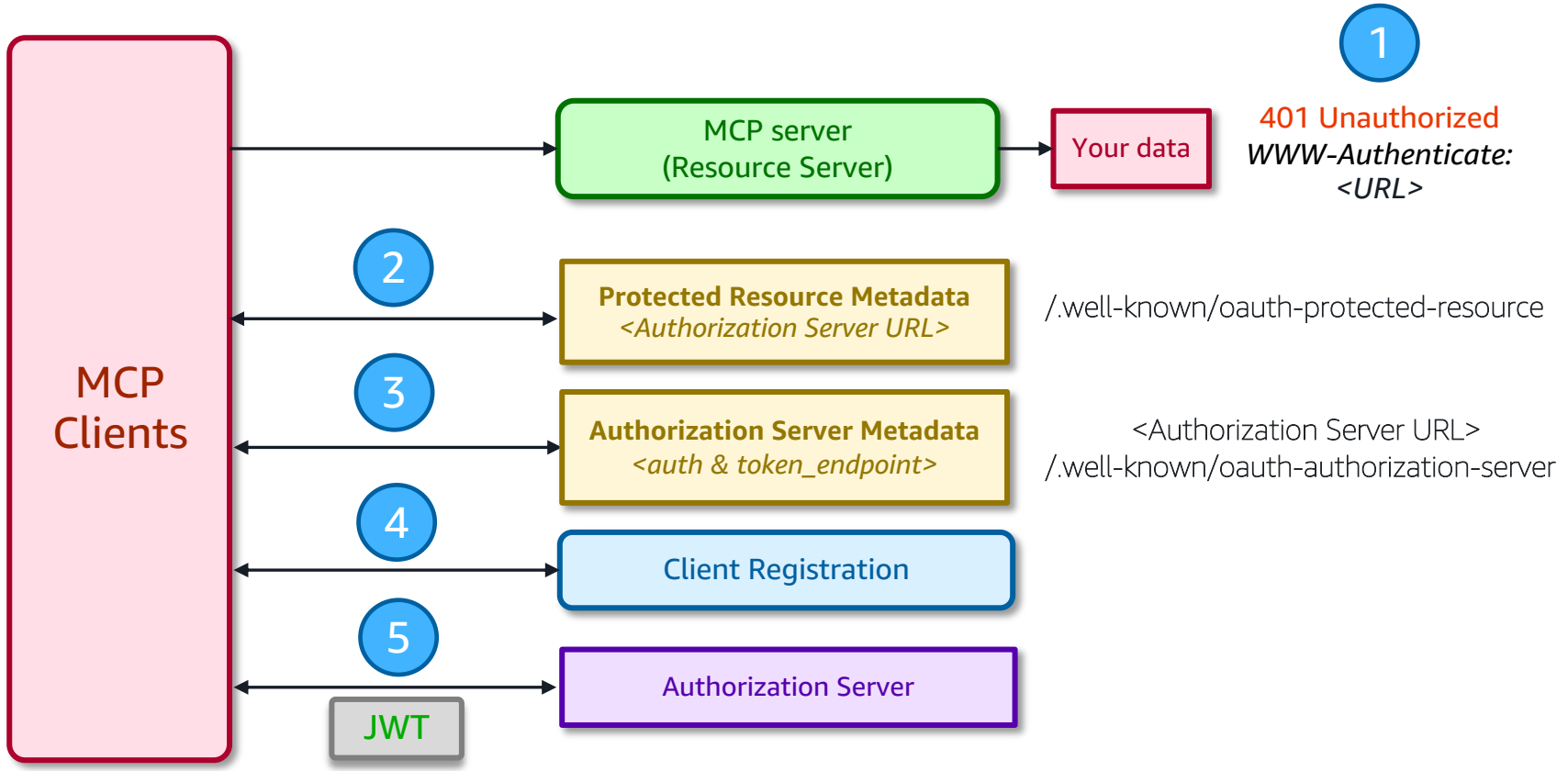

## Advantages

- Managed and maintained
- Authorization framework
- Easy to setup
- Scalable *(We'll get there..)*

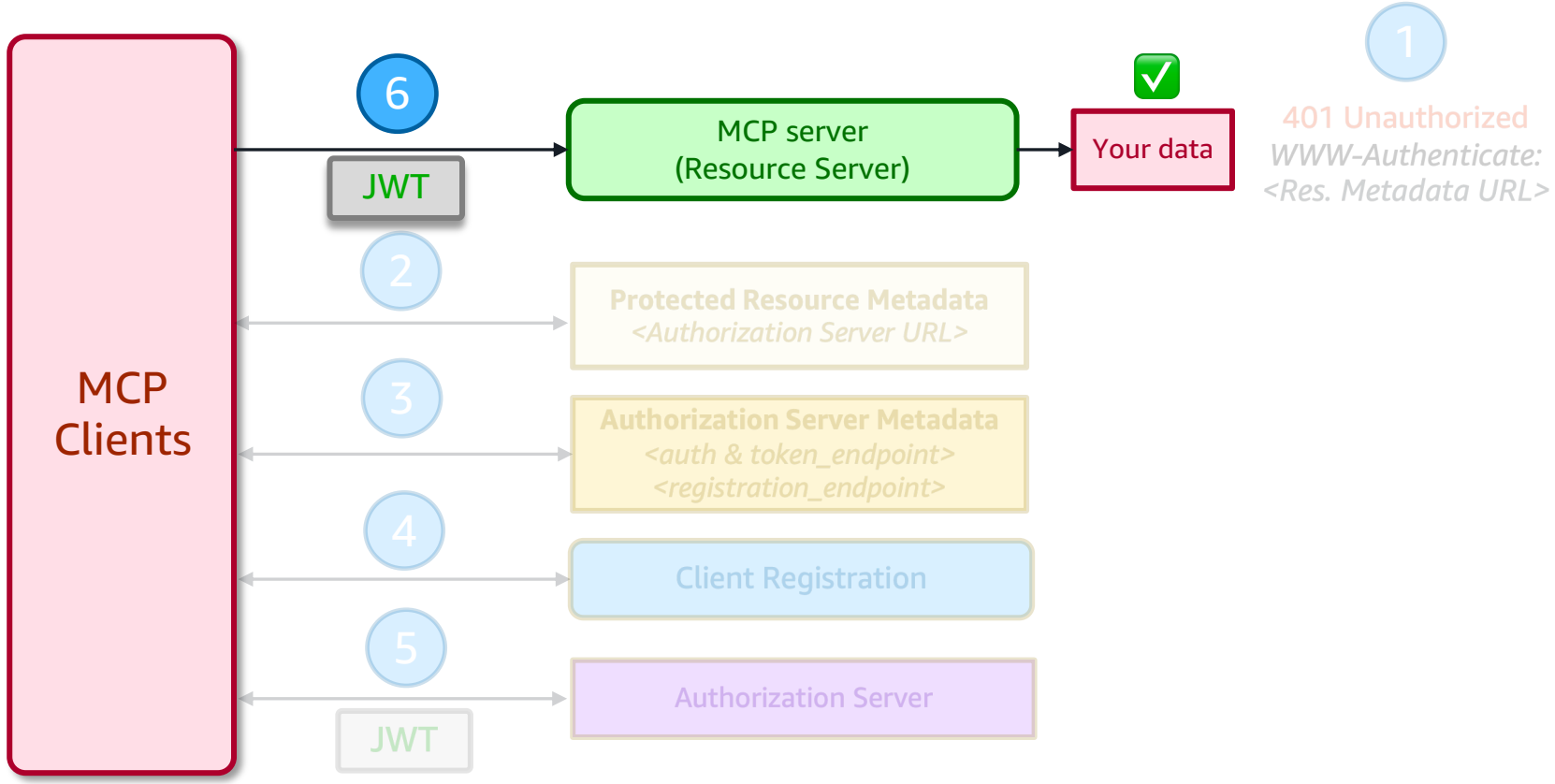
## Challenges

- Development effort
- Operational effort
- Latency & Scaling

# Authorization spec update (2025-11-25)



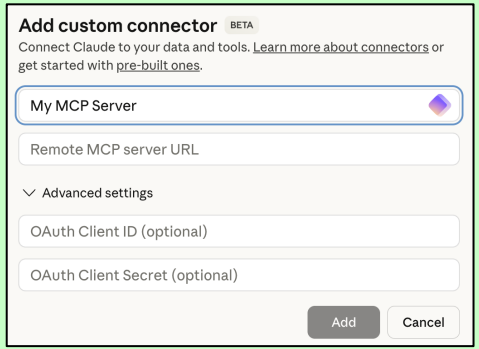
# Authorization spec update (2025-11-25)



# Client registration approaches and alternatives

## OAuth with static credentials

Pre-register an application client to provide client_id and client_secret manually




## OAuth Client ID Metadata Documents (CIMD)

Provide HTTPs URL as client_id with client details (name, logo, redirect_uris etc

client_id: https://../metadata.json

```
"client_name": "Example MCP Client",
"client_uri": "app.com",
"logo_uri": "app.com/logo.png",
"redirect_uris": [https://.../callback],
"grant_types": ["authorization_code"]
```

## Dynamic Client Registration (DCR)

## MCP Auth Extensions (Client Credentials)

## API Keys

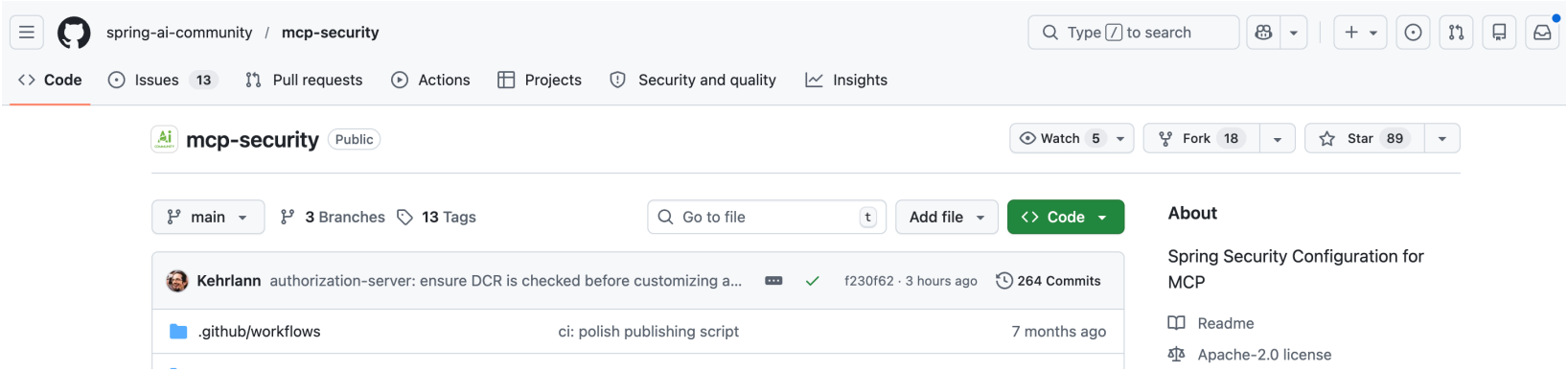
Pass pre-registered API-Keys via http headers

**x-api-key: <your-key>**

## None (Public)

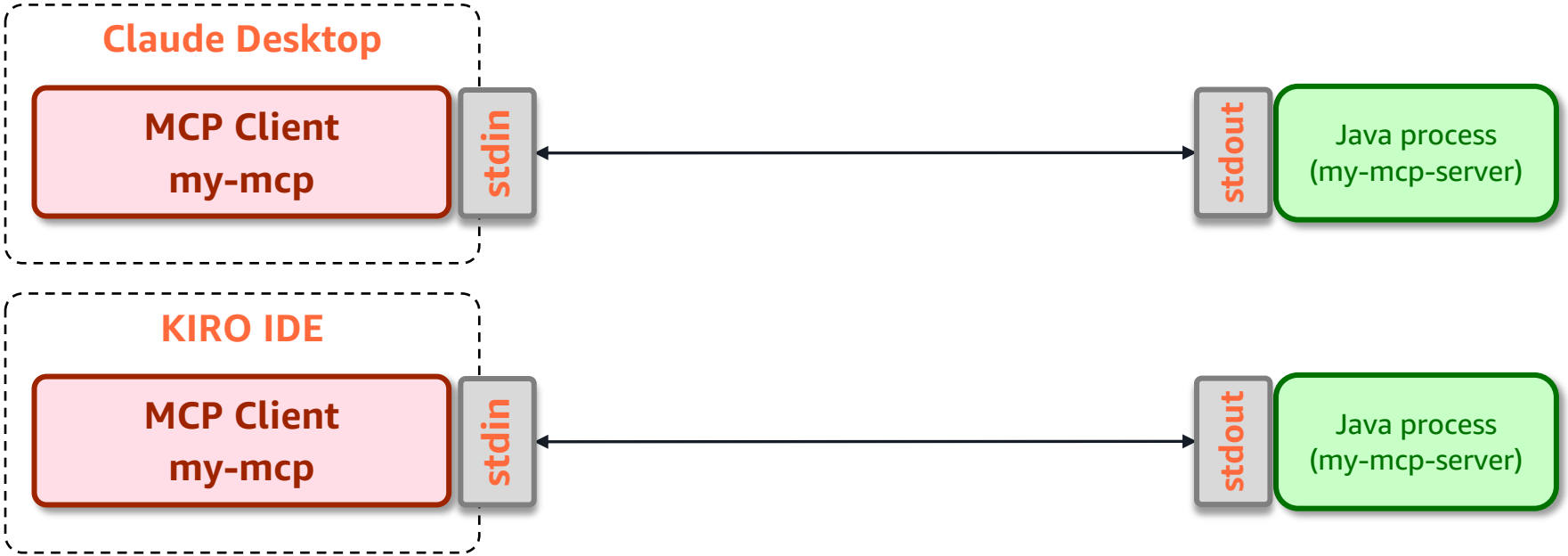
# Help?

- Application Layer
  - spring-ai-community/mcp-security

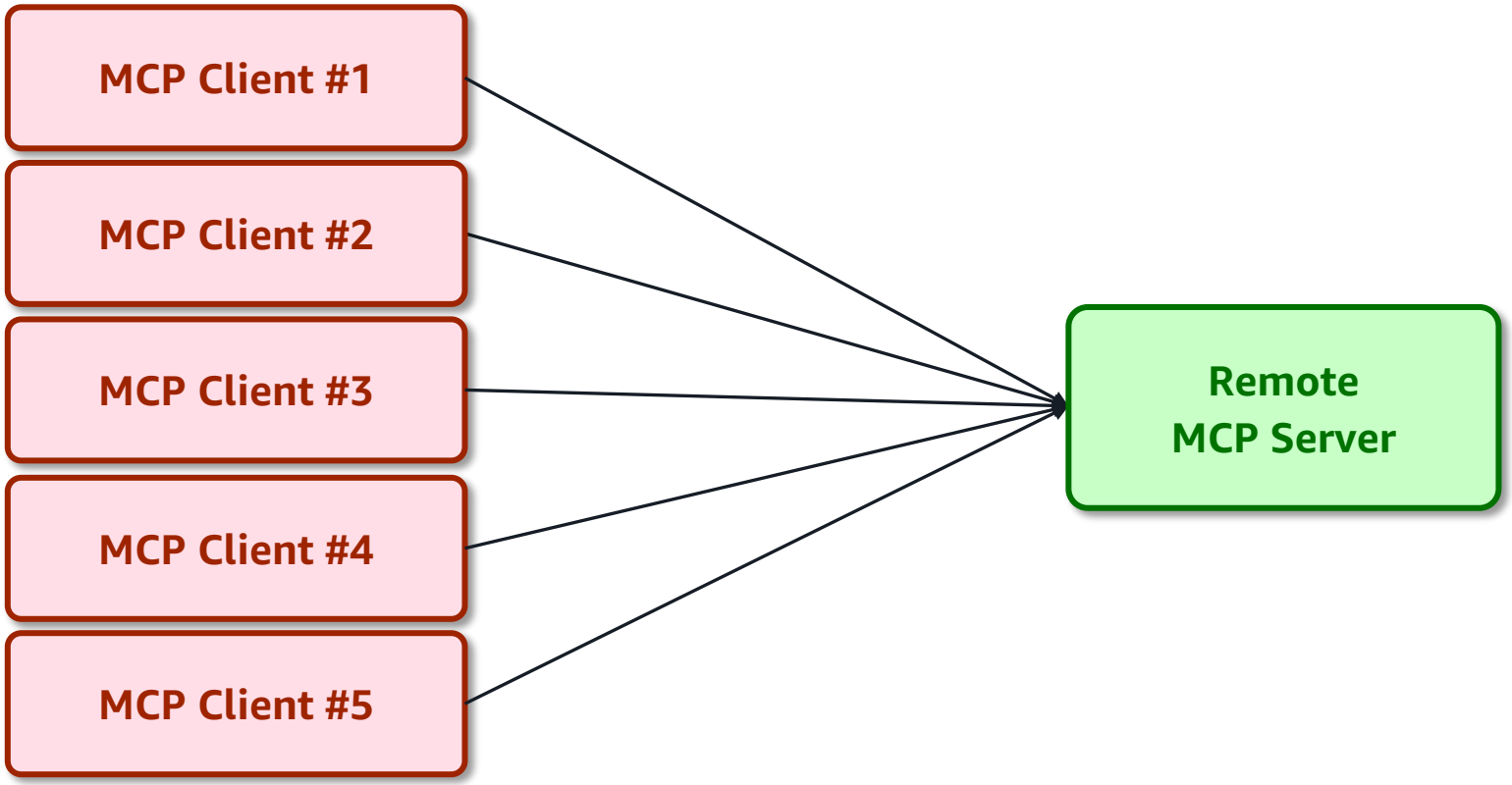



- Infrastructure Layer
  - For example: AgentCore Runtime + Identity (See sample repo)

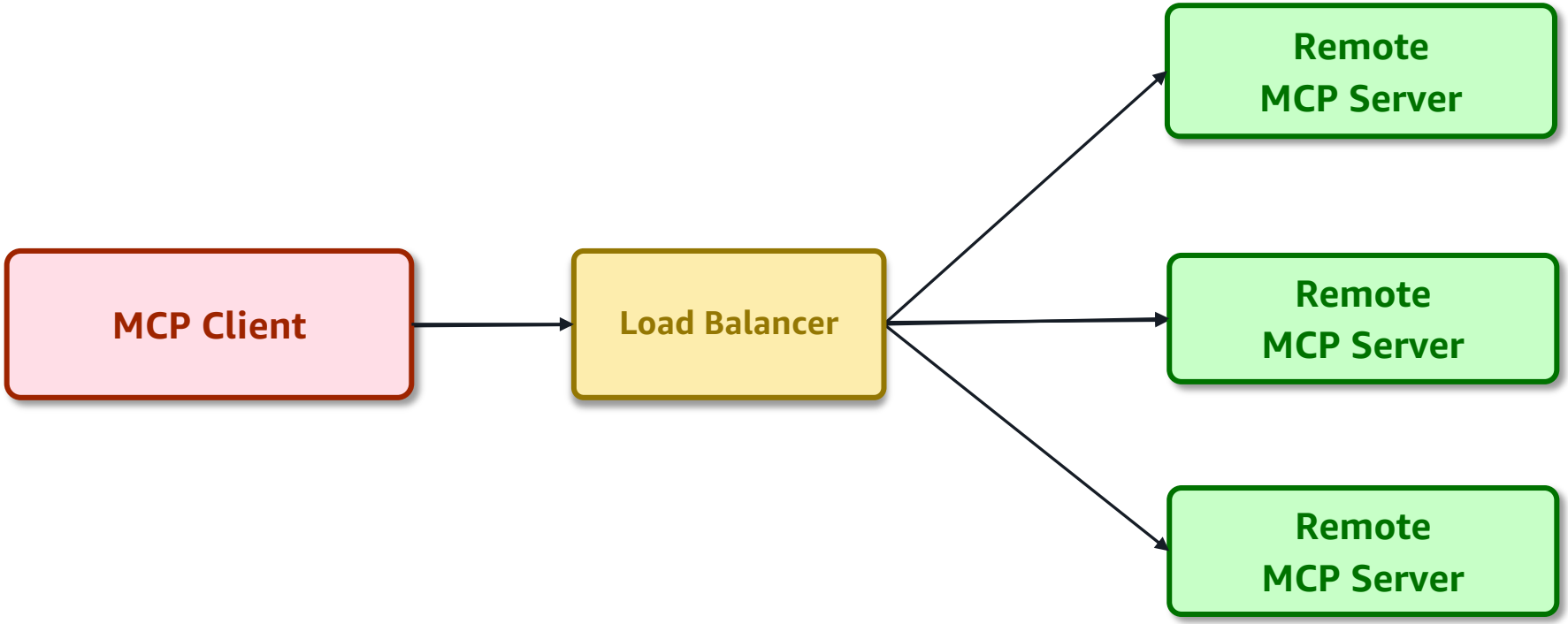
# Recap: Scaling local MCP Servers one by one



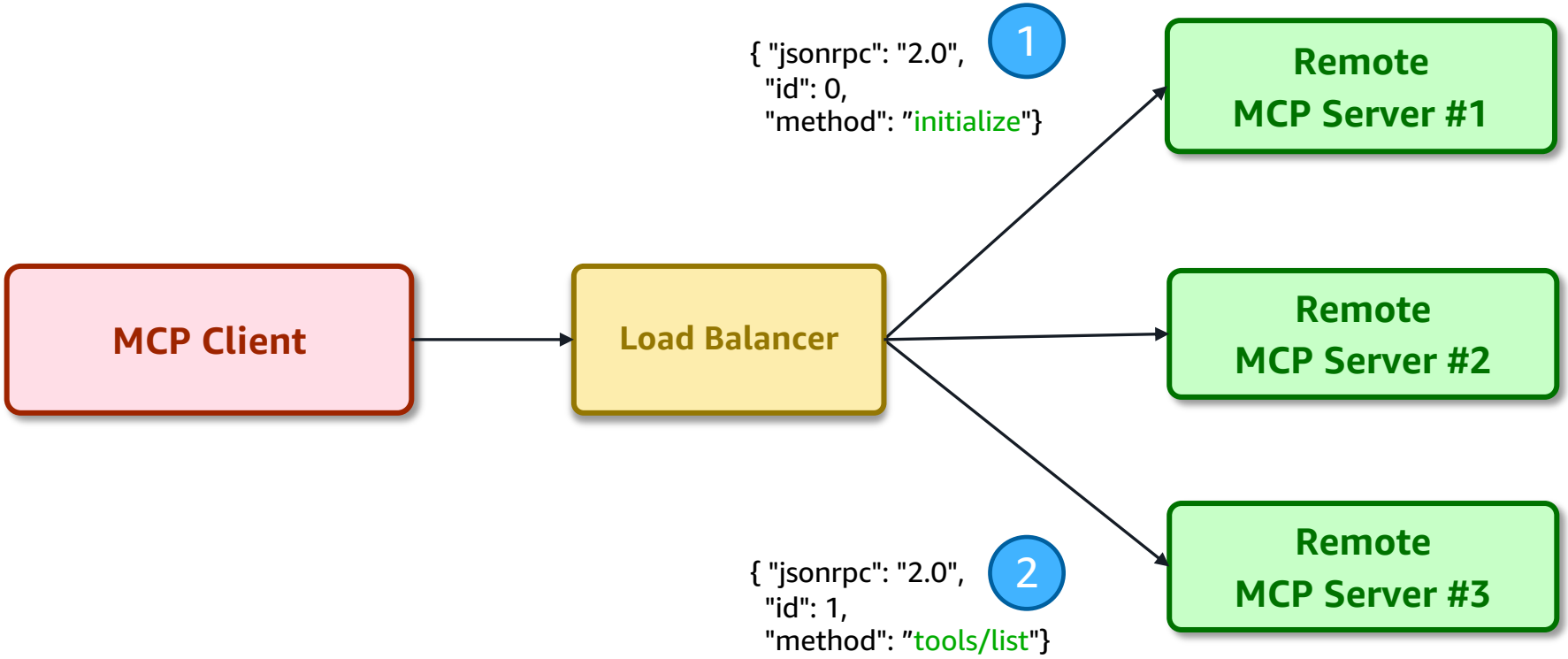
# Scaling remote MCP Servers?



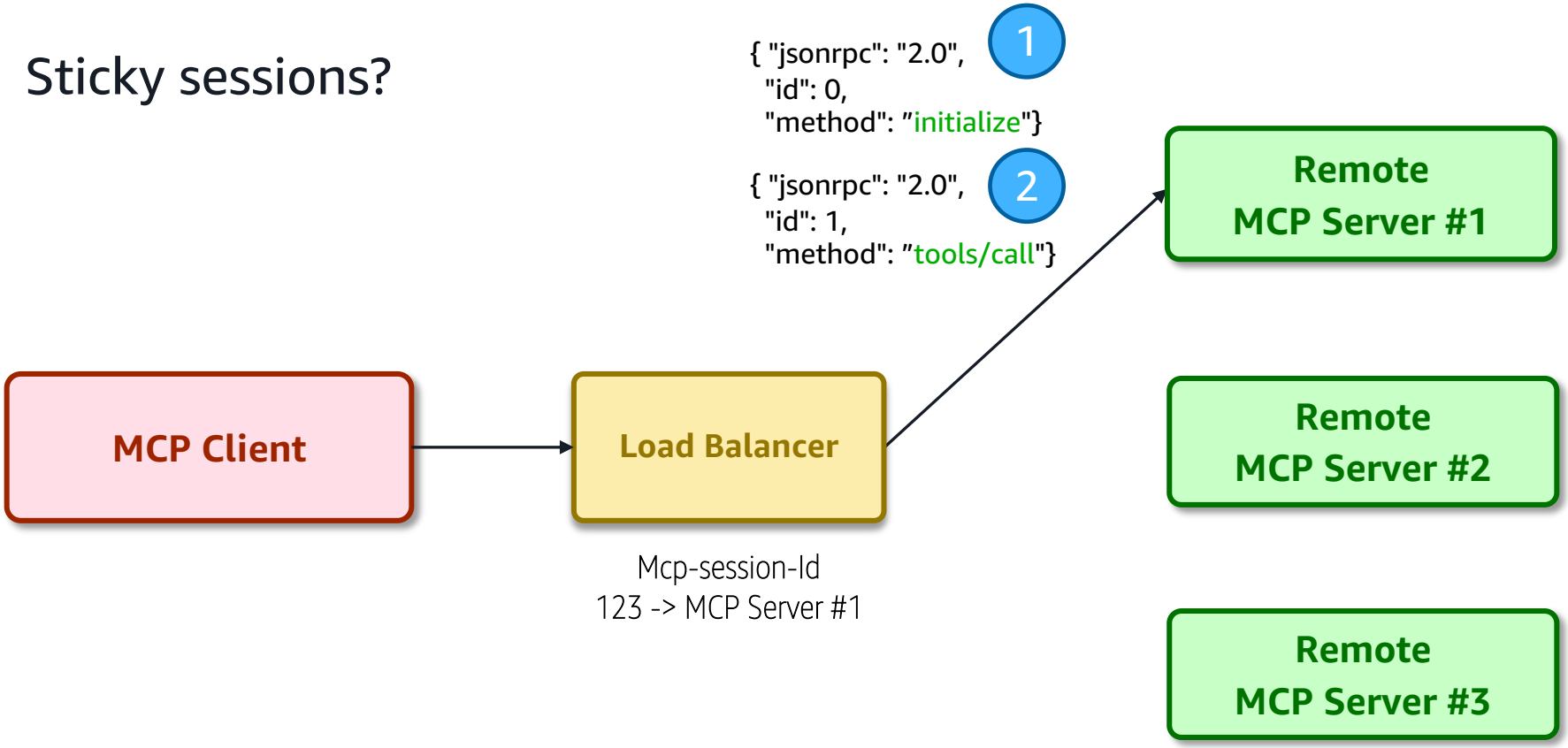
# Load balancing MCP servers



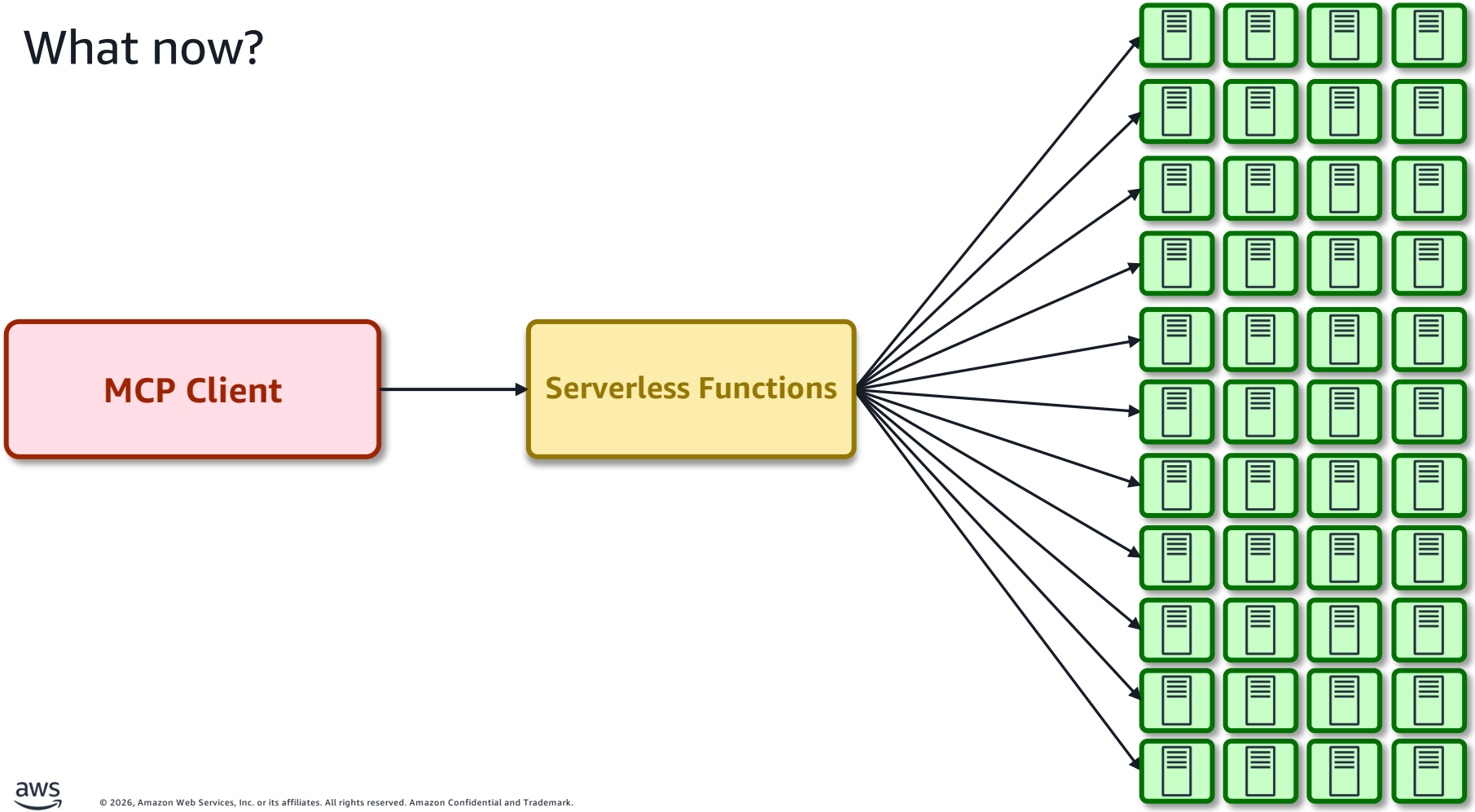
# Load balancing MCP servers



# Sticky sessions?



# What now?



# 1. Improve state at the protocol level

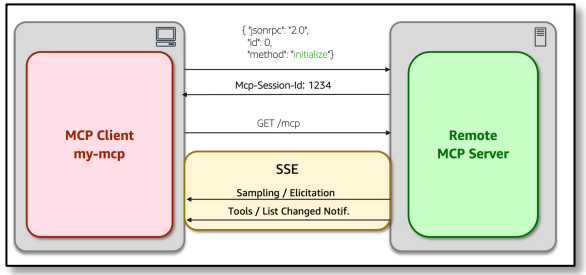
- Requests **MUST** include a string or integer ID.
- Unlike base JSON-RPC, the ID **MUST NOT** be `null`.
- The request ID **MUST NOT** have been previously used by the requestor within the same session.

### 2.5 Session Management

An MCP "session" consists of logically related interactions between a client and a server, beginning with the **initialization phase**. To support servers which want to establish stateful sessions:

1. A server using the Streamable HTTP transport **MAY** assign a session ID at initialization time, by including it in an `Mcp-Session-Id` header on the HTTP response containing the `InitializeResult`.

**SEP-1442**
**Make MCP Stateless (by default)**




**What we want to achieve:**

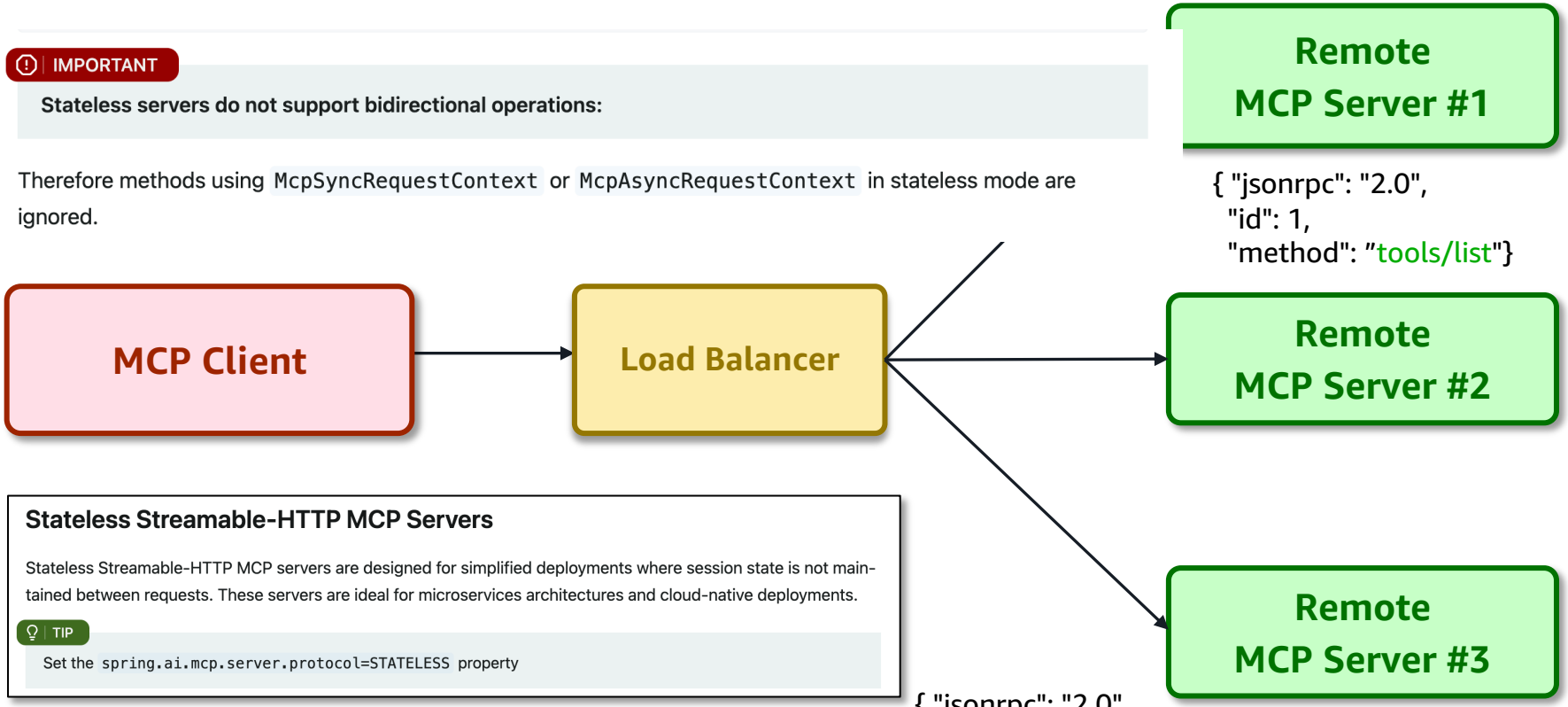
- **Next-generation transport**: evolve Streamable HTTP to run statelessly across multiple server instances and behave correctly behind load balancers and proxies.
- **Scalable session handling**: define how sessions are created, resumed, and migrated so that server restarts and scale-out events are transparent to connected clients.

# 2. Avoid state - Streamable HTTP in stateless mode

**IMPORTANT**

**Stateless servers do not support bidirectional operations:**

Therefore methods using `McpSyncRequestContext` or `McpAsyncRequestContext` in stateless mode are ignored.



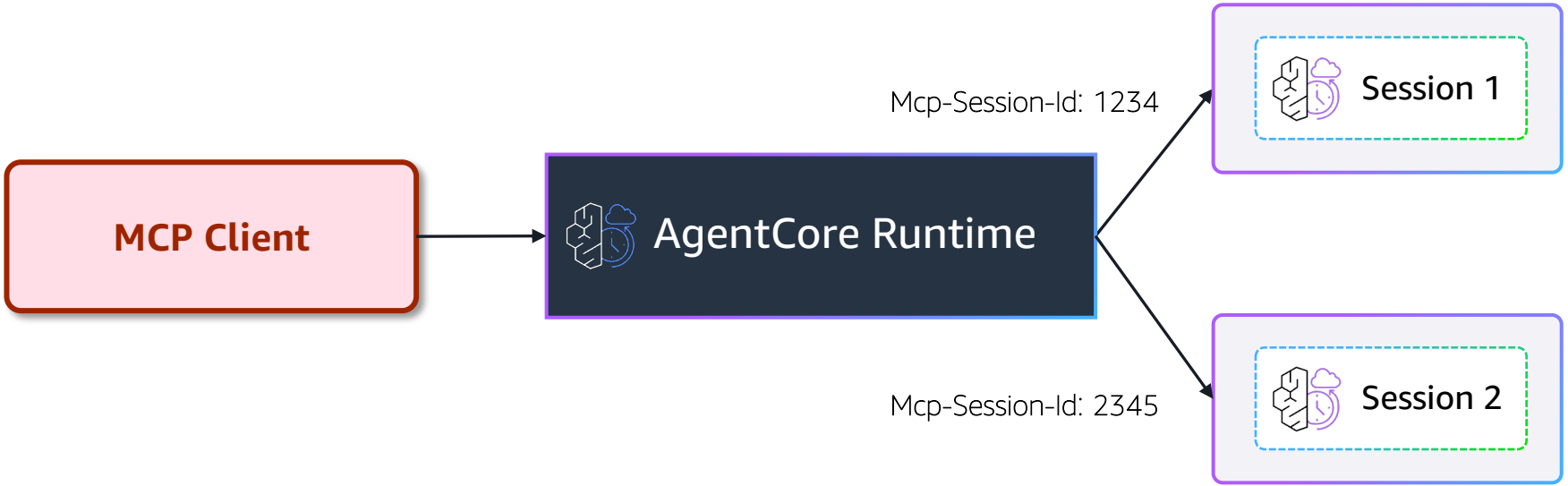

### Stateless Streamable-HTTP MCP Servers

Stateless Streamable-HTTP MCP servers are designed for simplified deployments where session state is not maintained between requests. These servers are ideal for microservices architectures and cloud-native deployments.

**TIP**

Set the `spring.ai.mcp.server.protocol=STATELESS` property

# 3. Accept state at the infrastructure level



# Context efficient MCP

# Challenges with MCP servers

- Performance and accuracy (Too many tools confusion)
- Cost – Increased token usage due to tool overhead

# MCP vs. CLIs

- CLIs can be an option for local developer workflows
- However: Important tradeoffs for discoverability, distribution, unified and fine grained auth, installation and updates

```
please use the gh cli to check open issues on

Shell gh issue list --repo aws/aws-durable-execution-sdk-java --state open
  313 OPEN  [Bug]: fix behavior differences in local runner bug
  2026-04-04T00:05:49Z
  307 OPEN  [chore]: Fix the auto approve bot dependencies
  301 OPEN  [Bug]: stuck executions with parallel/map operation bug
  2026-04-01T22:16:12Z
  300 OPEN  [chore]: add a workflow to add new issues to the project
  automatically  enhancement 2026-04-08T17:16:31Z
  298 OPEN  [Feature]: Cross context operation access check enhancement
  2026-03-31T17:40:07Z
  ...+14 lines (ctrl+o to toggle)
```

# MCP vs. Skills

- Skills provide progressive disclosure for knowledge and scripts
- However: Important tradeoffs for unified and fine grained auth, maintenance, installation and updates

```
---
name: example-skill
description: >
  A skill is a folder with a SKILL.md file containing metadata and instructions.
  Agents discover skills by their description, then load the full instructions
  only when a task matches — keeping context lightweight until needed.
---

# Example Skill
Instructions for the agent go here — any amount of markdown describing the workflow, steps,
rules, etc.

## Available scripts
- **`scripts/validate.sh`** — Validates configuration files
- **`scripts/process.py`** — Processes input data

## Reference files
- See [reference.md](reference.md) for bundled supporting docs
```

# 1. Reduce and curate available tools and context

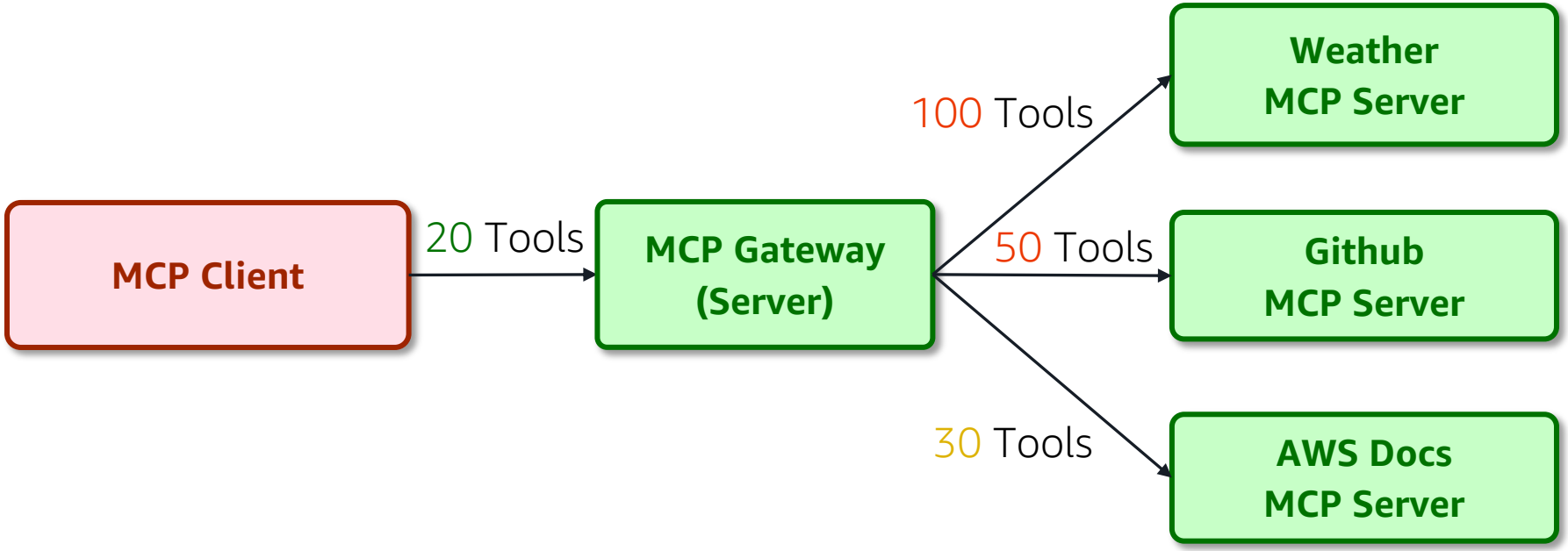
- Tool filtering: Explicitly add only the tools that you need

```
// Same MCP server — each agent only sees its relevant tools
var mathTools = SyncMcpToolCallbackProvider.builder()
    .mcpClients(mcpClients)
    .toolFilter((conn, tool) -> tool.name().matches("add|multiply|sub|divide"))
    .build();

var creativeTools = SyncMcpToolCallbackProvider.builder()
    .mcpClients(mcpClients)
    .toolFilter((conn, tool) -> tool.name().matches("loudJoke|random|roll-the-dice"))
    .build();

var mathAgent     = ChatClient.builder(chatModel).defaultToolCallbacks(mathTools).build();
var creativeAgent = ChatClient.builder(chatModel).defaultToolCallbacks(creativeTools).build();
```

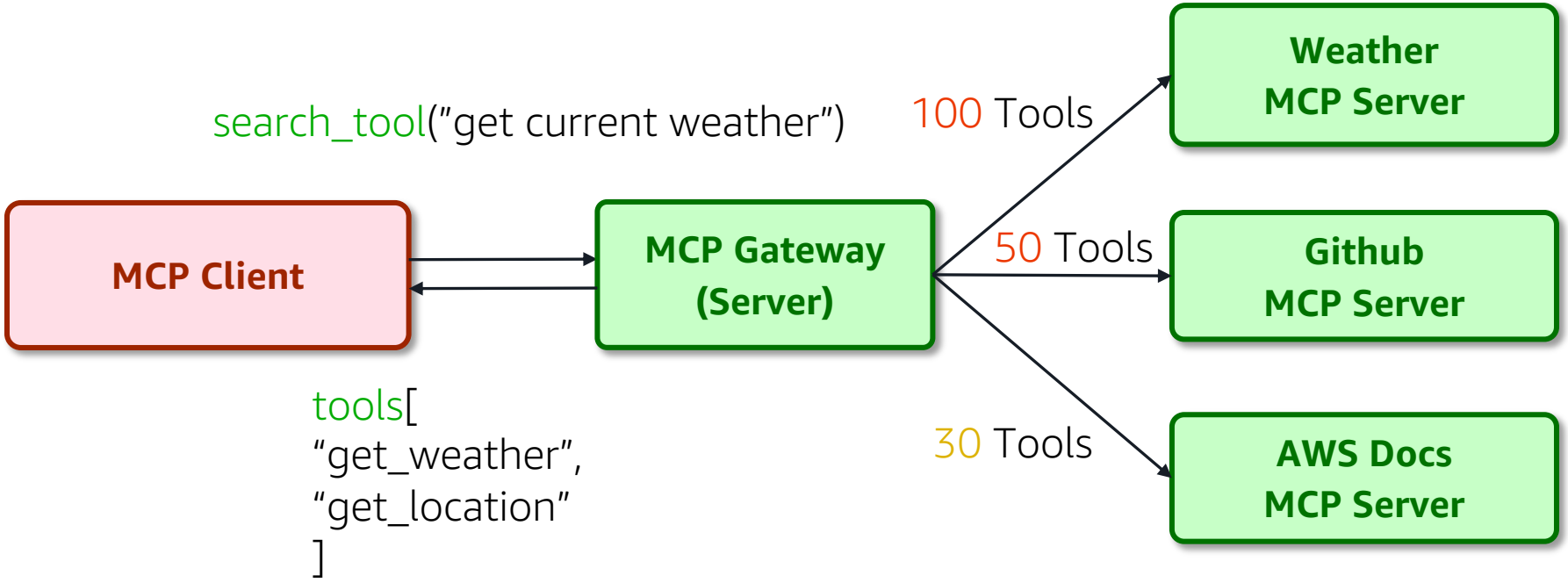
# Restricting tools with MCP Gateways



# 2. Progressive disclosure

- Dynamically load / unload MCP Servers and tools when needed
- Examples
  - Tool Search Tool
  - KIRO Powers (MCP)
  - Skills?

# Searching tools

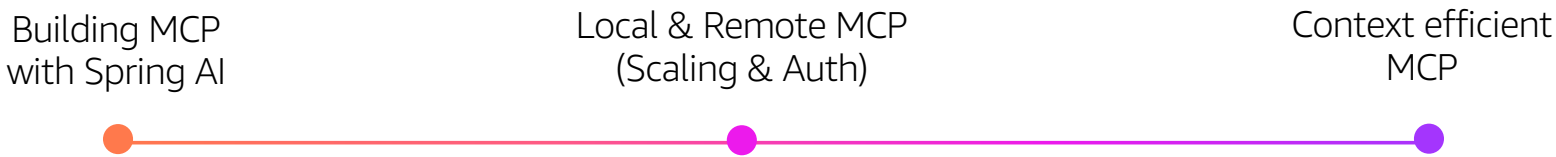


# 3. Code mode

- Agent writes and executes code that calls MCP tools via API
- Structured output valuable for result handling
- Context efficient chaining of tool calls (add, multiple, substract)

```
// Code generated by the LLM on the fly and executed in a sandbox - can be stored for reuse
var sum     = client.callTool(new CallToolRequest("add",      Map.of("x", 3,       "y", 4)));   // → 7
var product = client.callTool(new CallToolRequest("multiply", Map.of("x", sum,     "y", 5)));   // →
var answer  = client.callTool(new CallToolRequest("sub",      Map.of("x", product, "y", 10)));  // →
25
```

# Recap



# Spring AI MCP Demo

https://github.com/jamesward/spring-ai-mcp-demo

# Thank you!

Maximilian Schellhorn
https://www.linkedin.com/in/maxschell

James Ward
https://www.linkedin.com/in/jamesward