Jug Summer Camp

- enjoy it -

Building Smarter MCP Servers

Generic vs. Domain-Specific Approaches

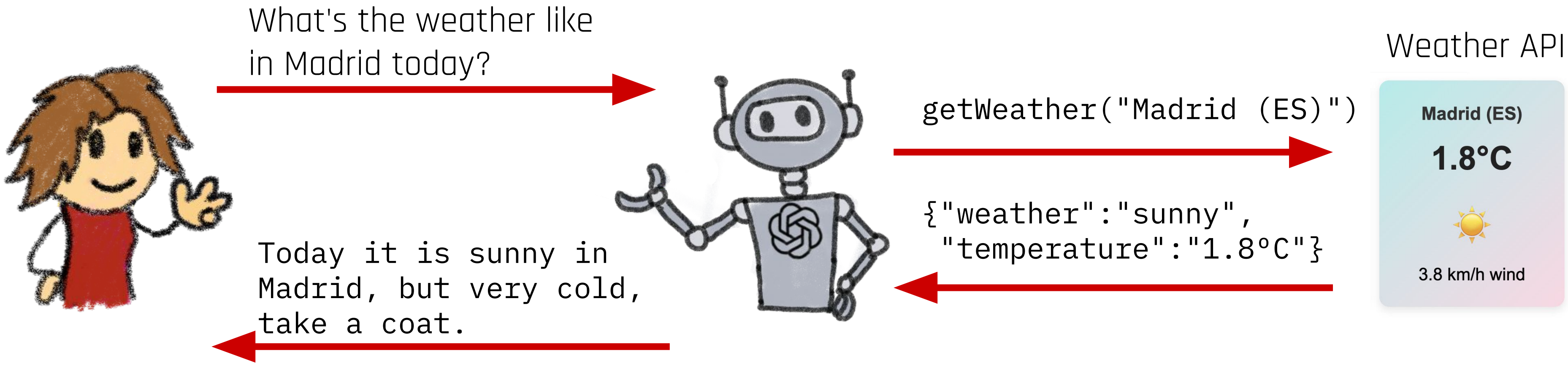
Horacio González 2025-09-05

SERLi MAIF gedivote. DARVA elastic Cloud APIM clever cloud GitHub Couchbase

@LostInBrittany

LLM evolution

From simple chat to tool-enhanced agent!



What's the weather like in Madrid today?

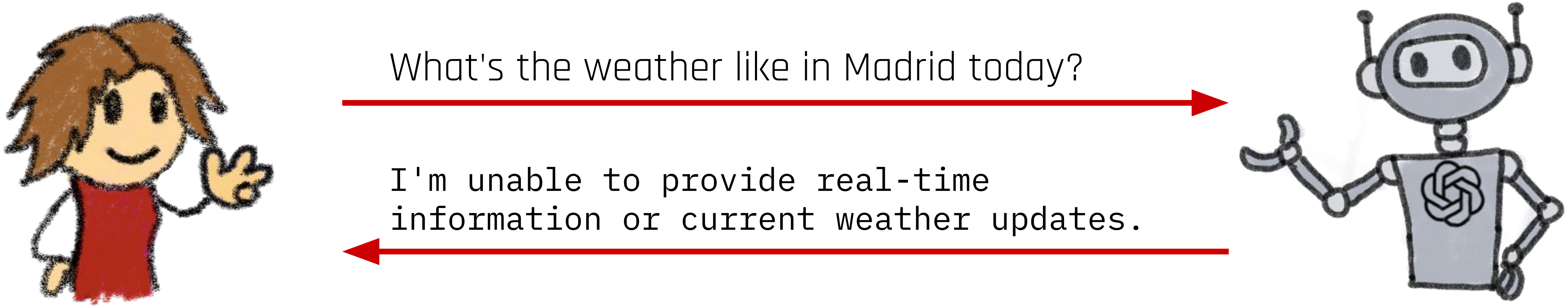
`getWeather("Madrid (ES)")`

```
{"weather":"sunny",
 "temperature":"1.8°C"}
```

Today it is sunny in Madrid, but very cold, take a coat.

Weather API

LLM are only language models

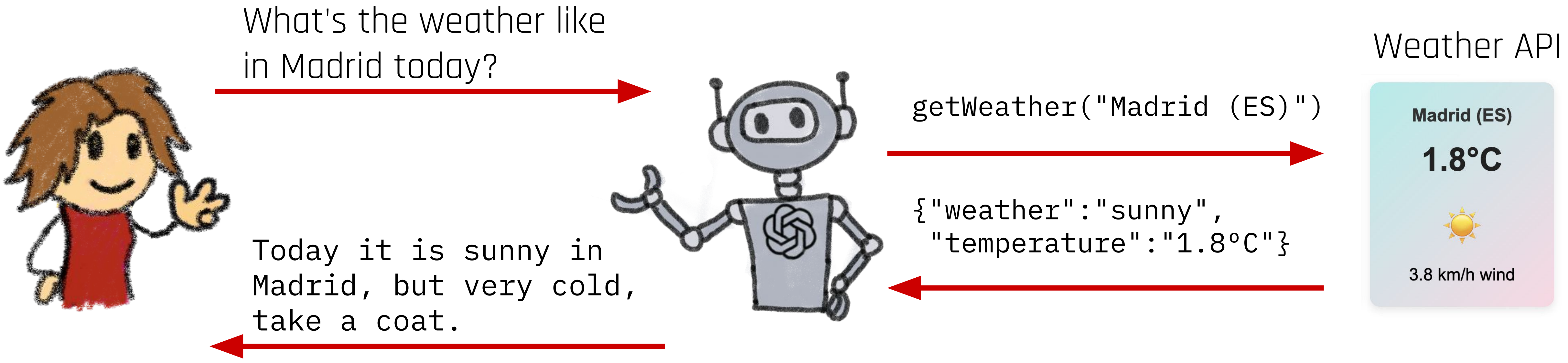


What's the weather like in Madrid today?

```
I'm unable to provide real-time
information or current weather updates.
```

They have no built-in way to use external tools or real-time data

Tools and plugins were added



What's the weather like in Madrid today?

```
getWeather("Madrid (ES)")
```

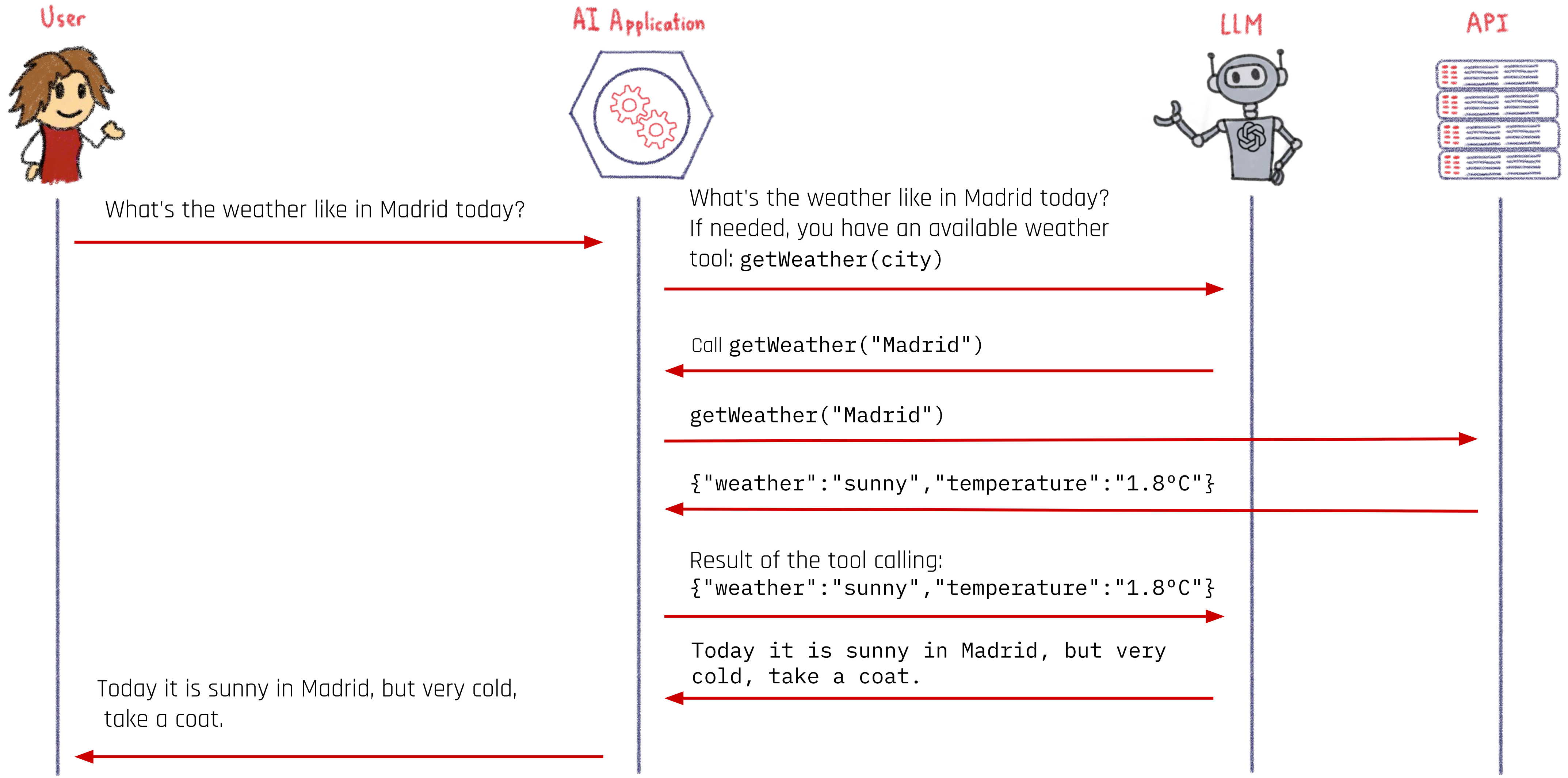
Weather API

```
{"weather":"sunny",
 "temperature":"1.8°C"}
```

```
Today it is sunny in
Madrid, but very cold,
take a coat.
```

LLM recognizes it needs an external function and calls it, integrating the result into a natural-language response.

LLM don't call directly those tools



What's the weather like in Madrid today?

What's the weather like in Madrid today?
If needed, you have an available weather tool: `getWeather(city)`

Call `getWeather("Madrid")`

`getWeather("Madrid")`

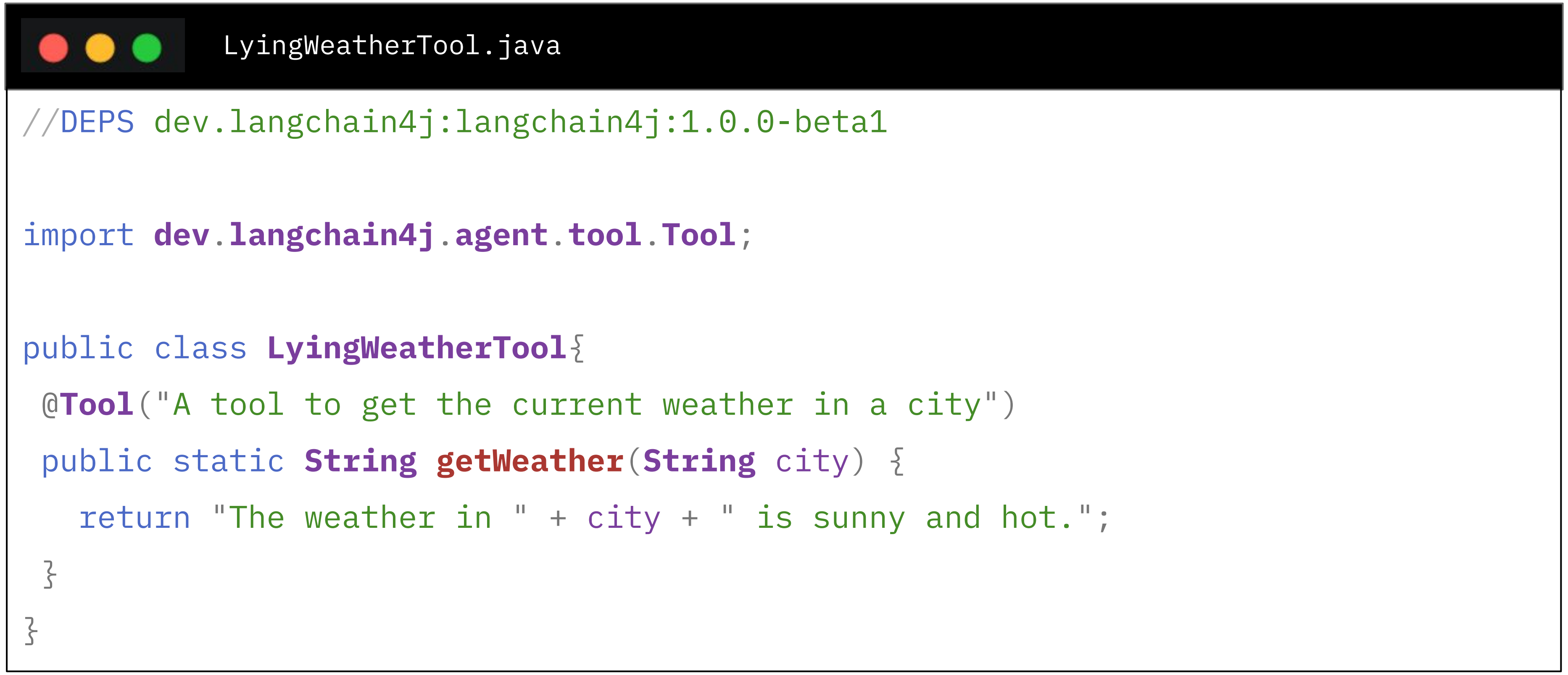
`{"weather":"sunny","temperature":"1.8°C"}`

Result of the tool calling:
`{"weather":"sunny","temperature":"1.8°C"}`

`Today it is sunny in Madrid, but very cold, take a coat.`

Today it is sunny in Madrid, but very cold, take a coat.

How are those LLM Tools defined?



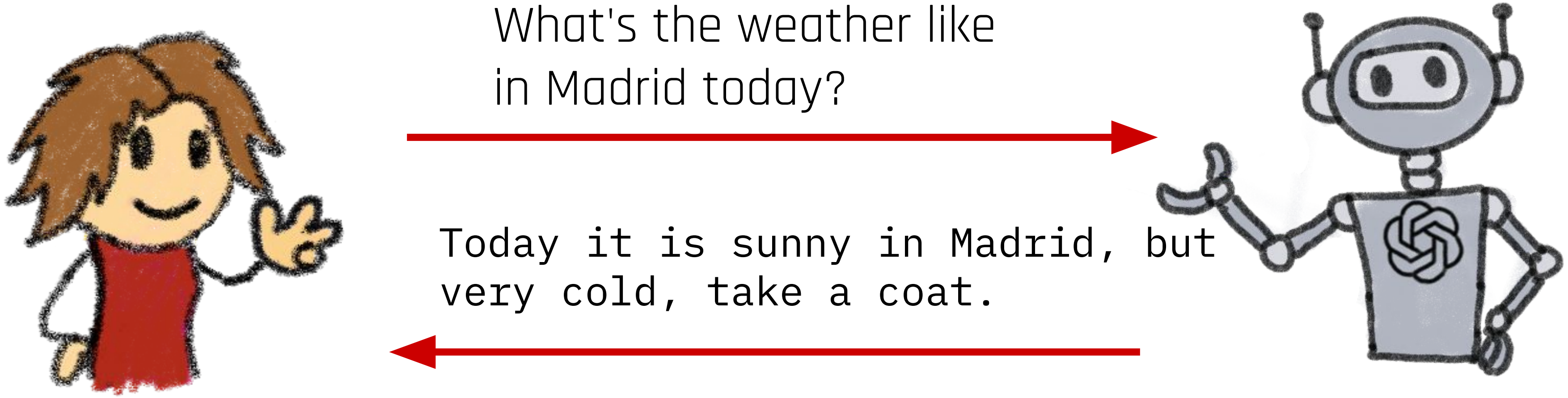
LyingWeatherTool.java

```java
//DEPS dev.langchain4j:langchain4j:1.0.0-beta1

import dev.langchain4j.agent.tool.Tool;

public class LyingWeatherTool{
 @Tool("A tool to get the current weather in a city")
 public static String getWeather(String city) {
   return "The weather in " + city + " is sunny and hot.";
 }
}
```

Here in Java using LangChain4j

Why this matters?

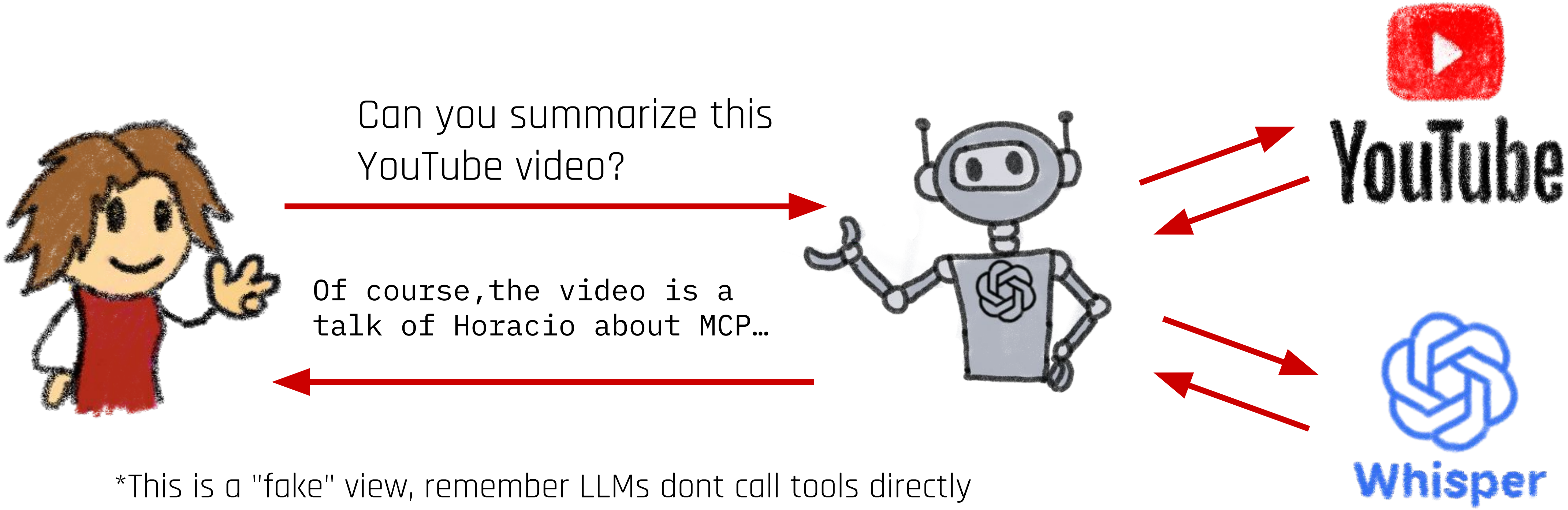
- Moves LLMs from static text generation
    - dynamic system components
- Increases accuracy & real-world usability
- Allows developers to control what the LLM can access



What's the weather like
in Madrid today?

```
Today it is sunny in Madrid, but
very cold, take a coat.
```

From LLM chats to LLM-powered agents



Can you summarize this YouTube video?

Of course,the video is a talk of Horacio about MCP…

*This is a "fake" view, remember LLMs dont call tools directly
But it's the view from the Point of View of the user

LLMs act like an agent that can plan actions:
search the web, run some code, then answer

The rapid evolution of agents

Before MCP
(2023–November 2024)

- Agents == niche LangChain, bespoke APIs, Copilot experiments…
- No standard way to connect LLMs to tools.

MCP Introduced
(Nov 2024)

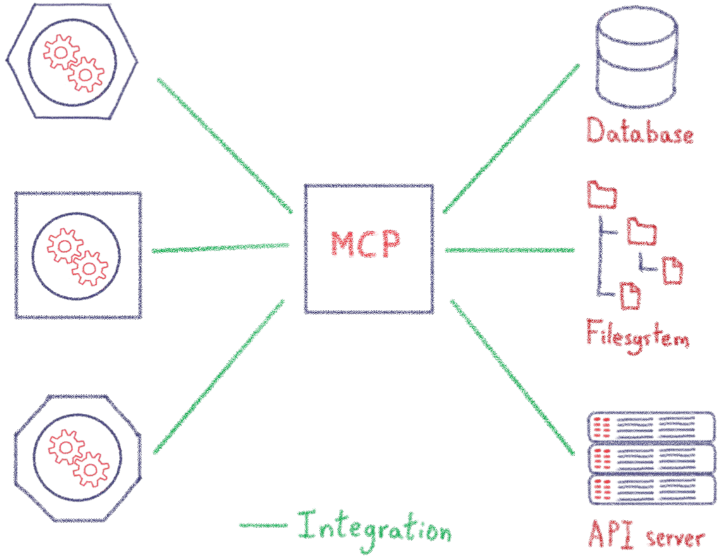
- Anthropic launches Model Context Protocol.
- Vendor-neutral, open standard for connecting LLMs.

The explosion
(2025)

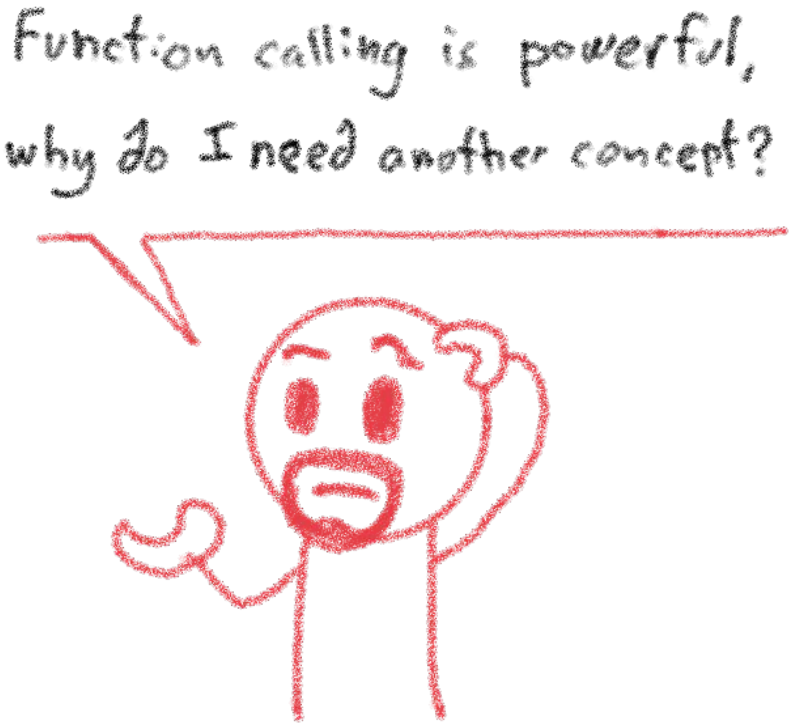
- Agents go mainstream: runtimes, orchestration, enterprise adoption.
- MCP reframed as the interoperability layer for agents.

Model Context Protocol (MCP): The missing link

MCP bridges LLMs with your applications, enabling controlled, real-world interactions



Why Do We Need MCP?

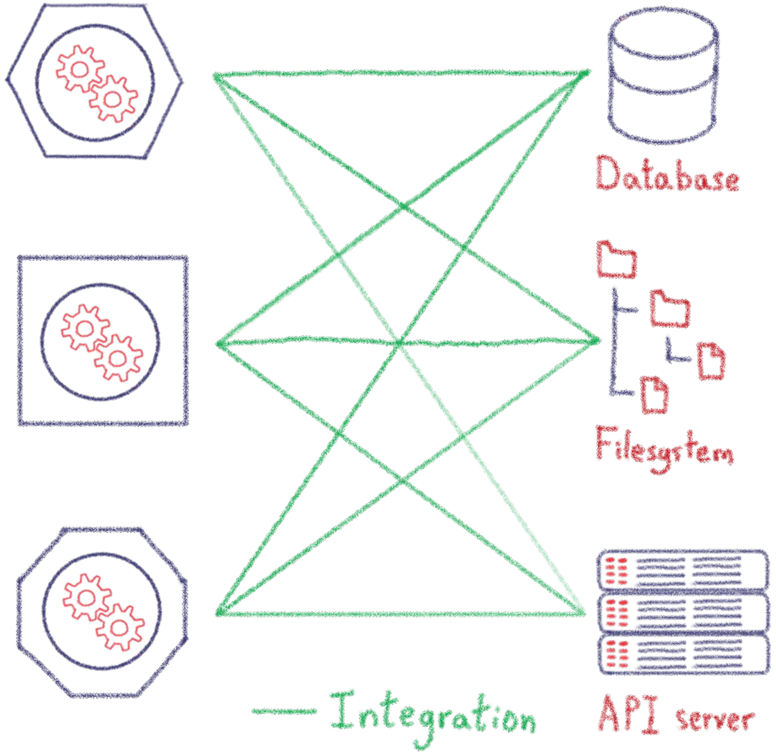


LLM function calling is useful,
but it lacks structure

Why Do We Need MCP?

Problem

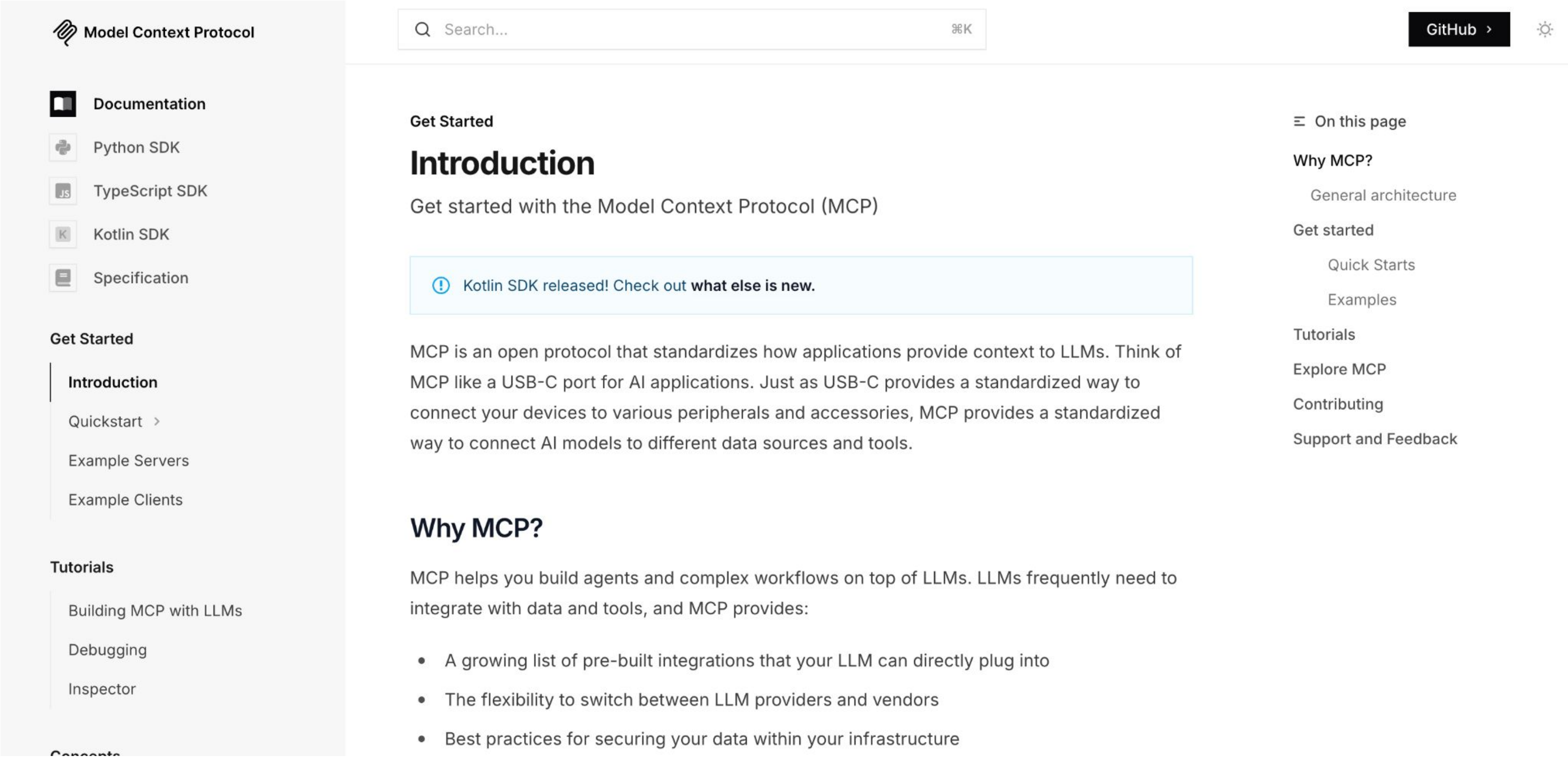
- LLMs **don't automatically know** what functions exist.
- **No standard way** to expose an application's capabilities.
- **Hard to control** security and execution flow.
- Expensive and fragile **integration spaghetti**



Model Context Protocol

Anthropic, November 2024:
LLMs intelligence isn't the bottleneck,
connectivity is

Model Context Protocol



Get Started

Introduction

Get started with the Model Context Protocol (MCP)

Kotlin SDK released! Check out **what else is new.**

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.

Why MCP?

MCP helps you build agents and complex workflows on top of LLMs. LLMs frequently need to integrate with data and tools, and MCP provides:

- A growing list of pre-built integrations that your LLM can directly plug into
- The flexibility to switch between LLM providers and vendors
- Best practices for securing your data within your infrastructure

De facto standard for exposing system capabilities to LLMs

https://modelcontextprotocol.io/

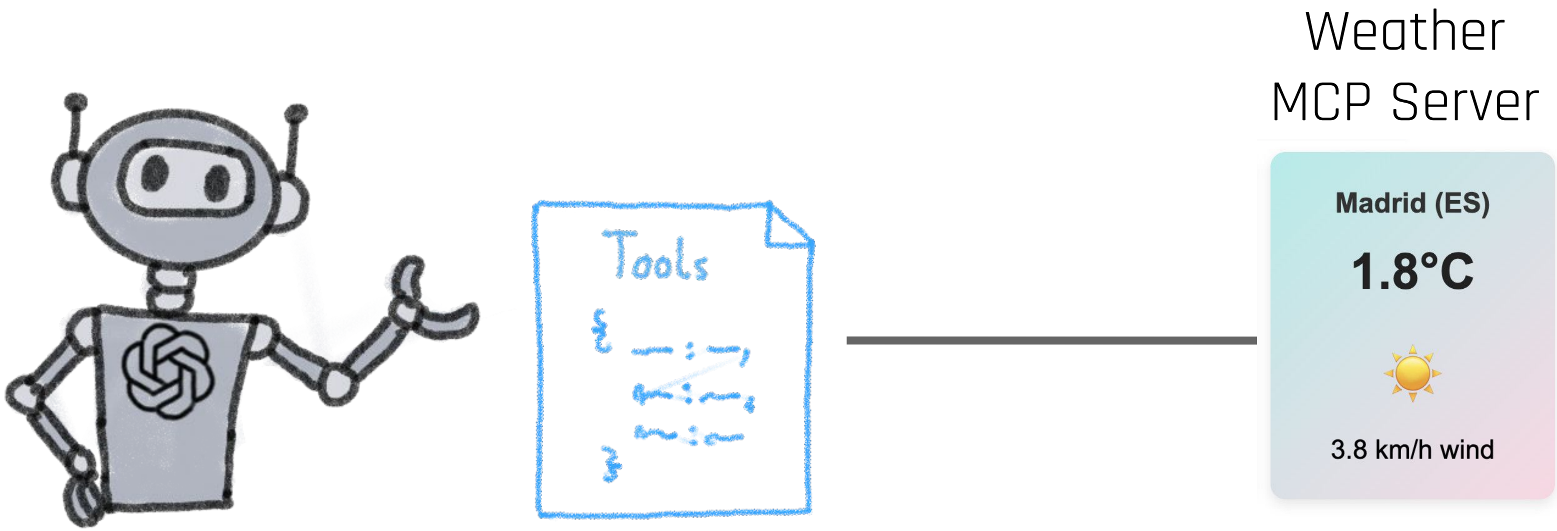
The MCP Landscape Today

Major players adopted MCP:

- **Anthropic** – Originator and tool provider (Claude Desktop, SDKs).
- **OpenAI** – Agent SDK, ChatGPT Desktop, Responses API.
- **Google DeepMind** – Gemini support and tooling.
- **Microsoft / GitHub** – Copilot Studio, Azure, Office integration, C# SDK.
- **Developer Platforms** – Replit, JetBrains, Sourcegraph, TheiaIDE.
- **Enterprise / Services** – Block, Stripe, Cloudflare, Baidu Maps.
- Thousands of MCP servers live.

How MCP works

- Applications define an MCP manifest (structured JSON).
- The manifest describes available functions, input/output formats, and security policies.
- LLMs can discover and request function execution safely.



Weather
MCP Server

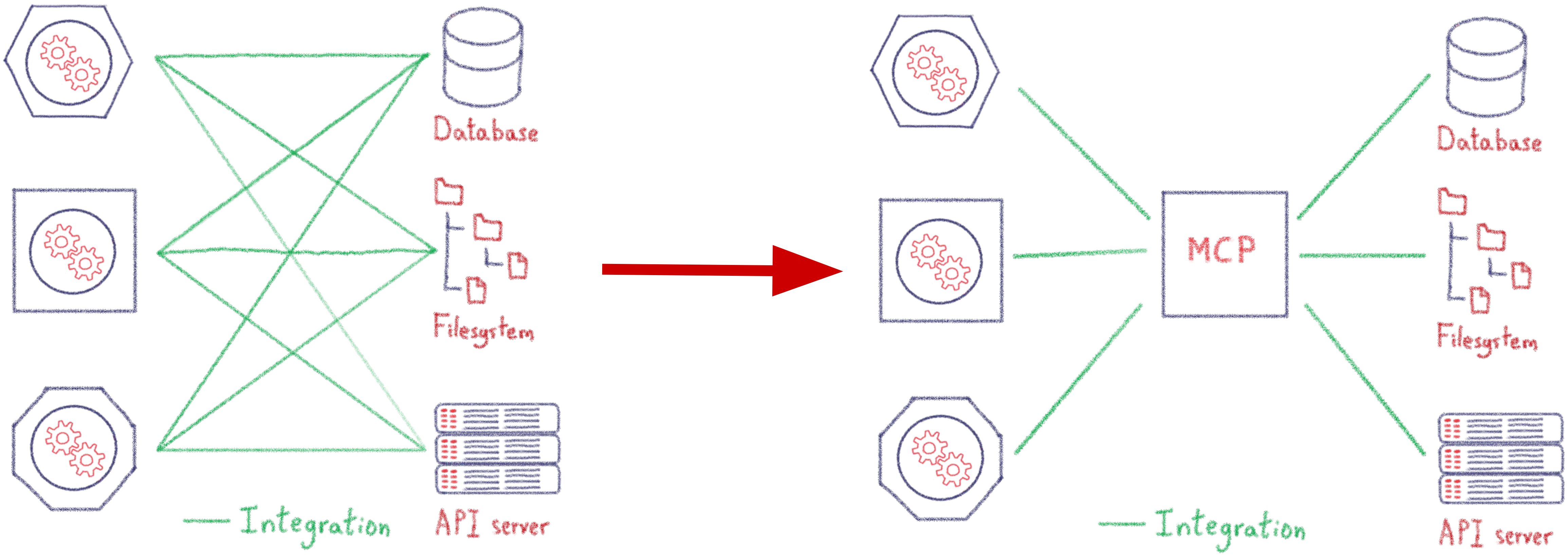
Madrid (ES)
1.8°C
3.8 km/h wind

MCP is provider-agnostic

Works with any LLM provider

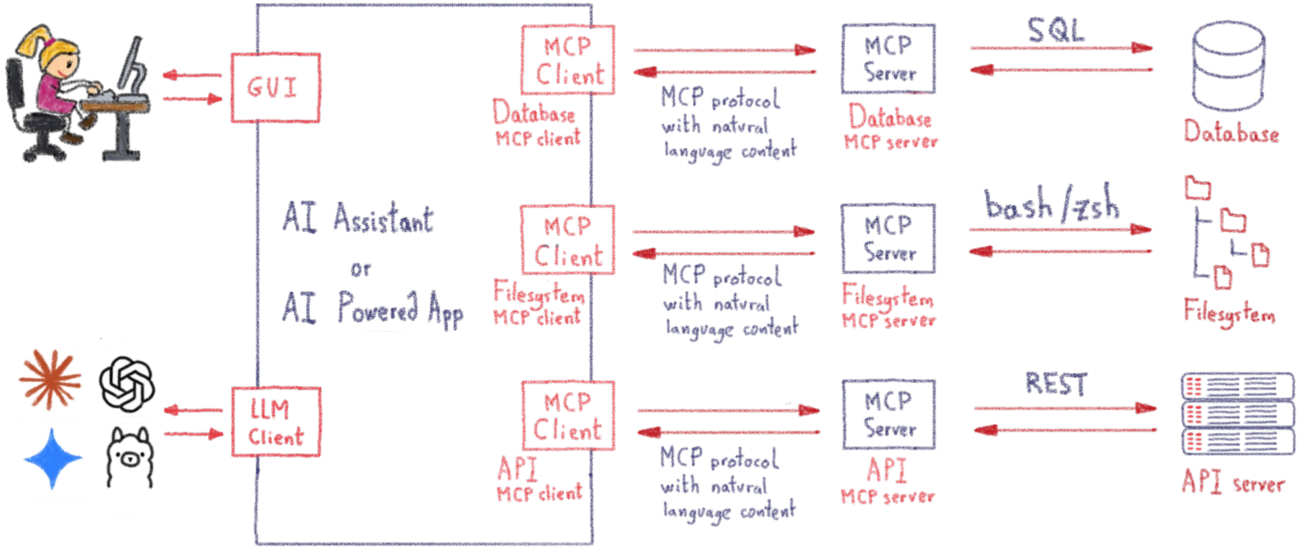
Ensures standardized function exposure across platforms

MCP solves integration spaghetti

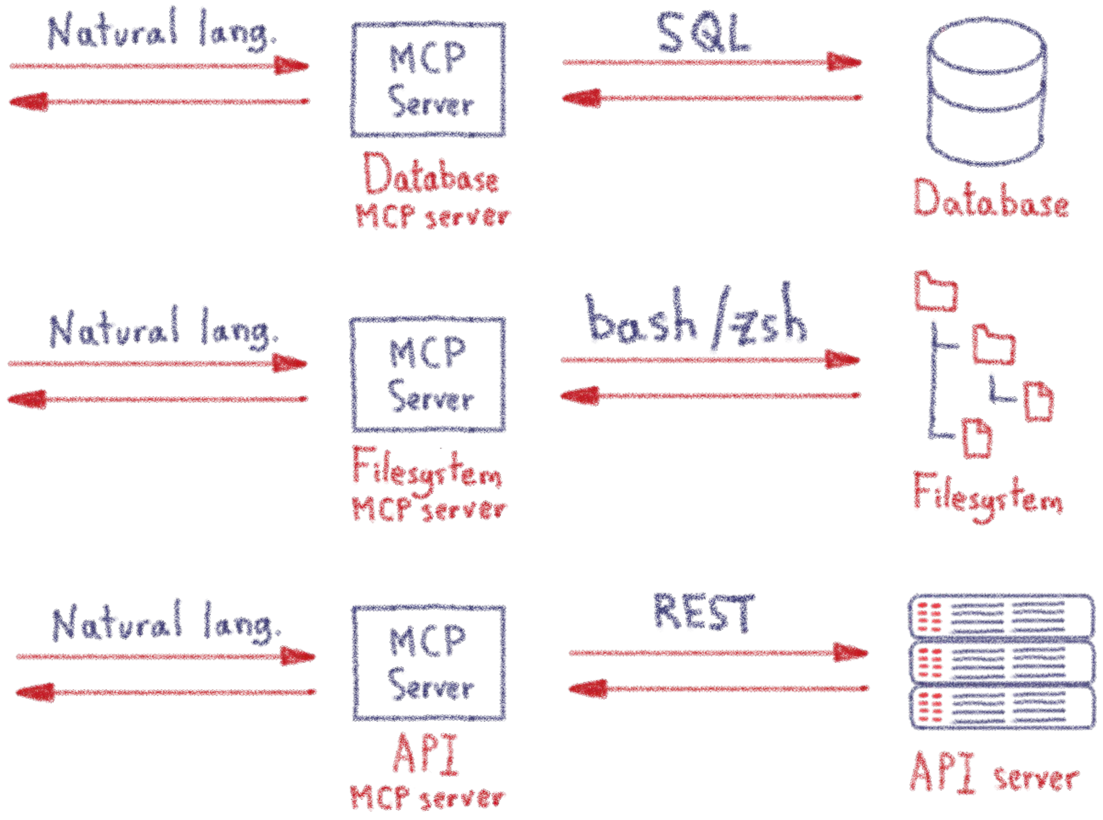


The architecture of MCP

Clients, servers, protocol and transports
Tools, resources and prompts

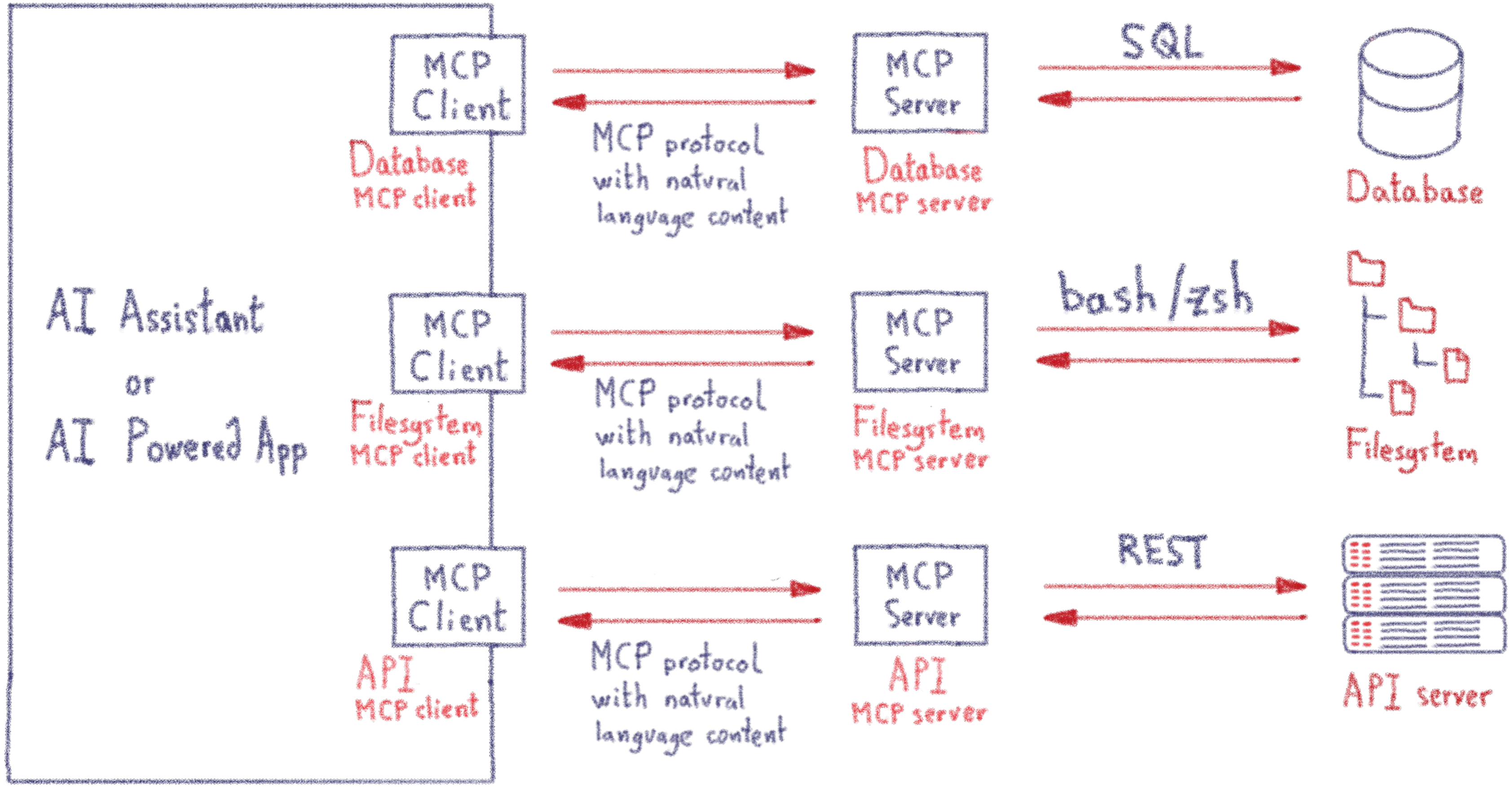


MCP Servers: APIs in natural language



A new kind of API

MCP Clients: on the AI assistant or app side



One MCP client per MCP Server

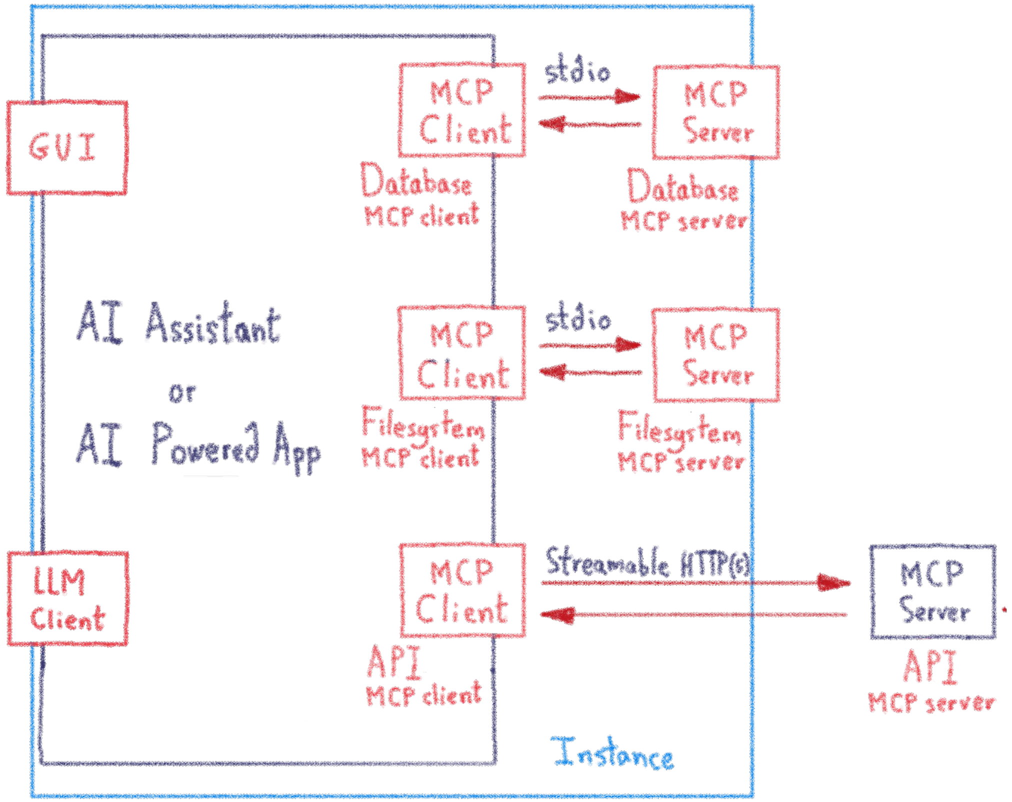
MCP Protocol & Transports

MCP Protocol

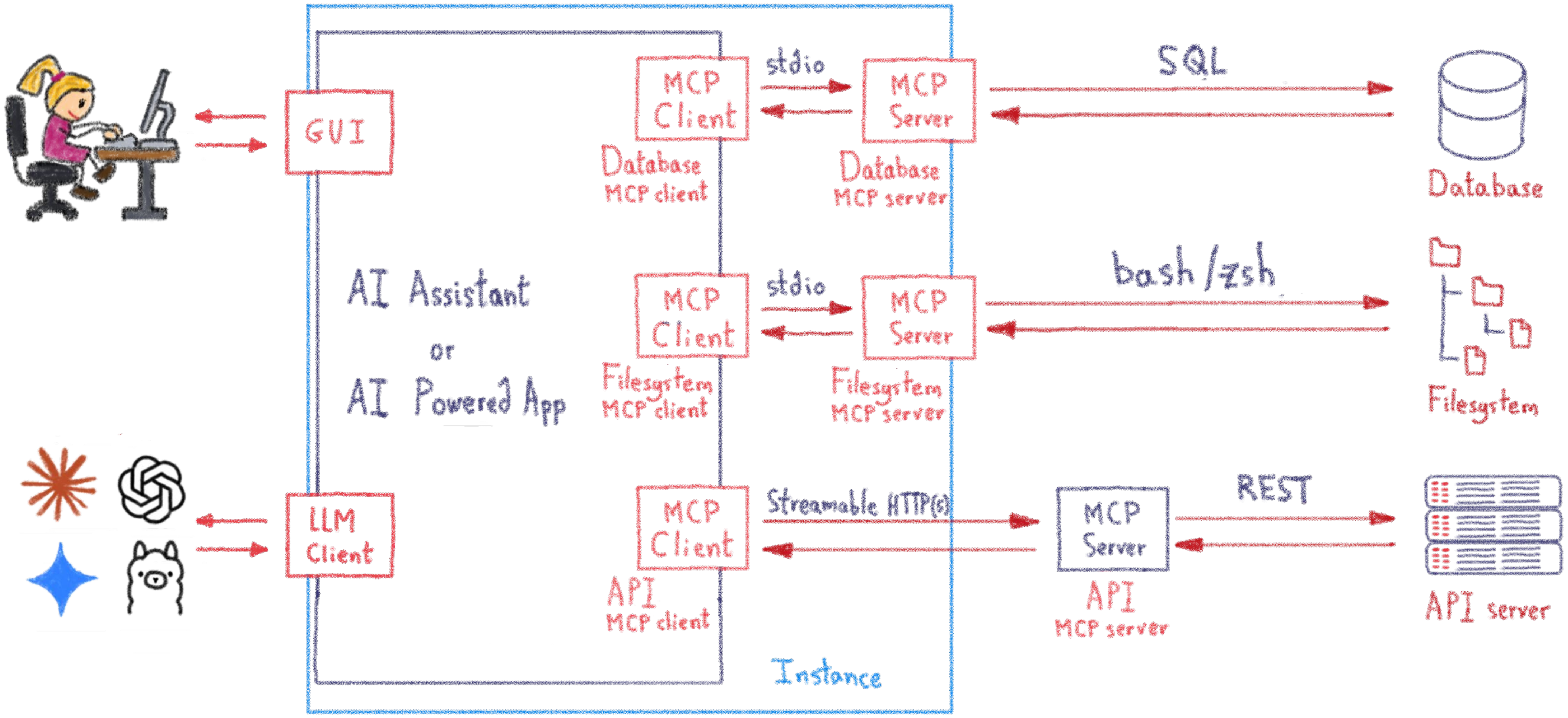
Follow the JSON-RPC 2.0 specification

MCP Transports

- STDIO (standard I/O)
  - Client and server in the same instance
- HTTP with SSE transport (deprecated)
- Streamable HTTP
  - Servers SHOULD implement proper authentication for all connections

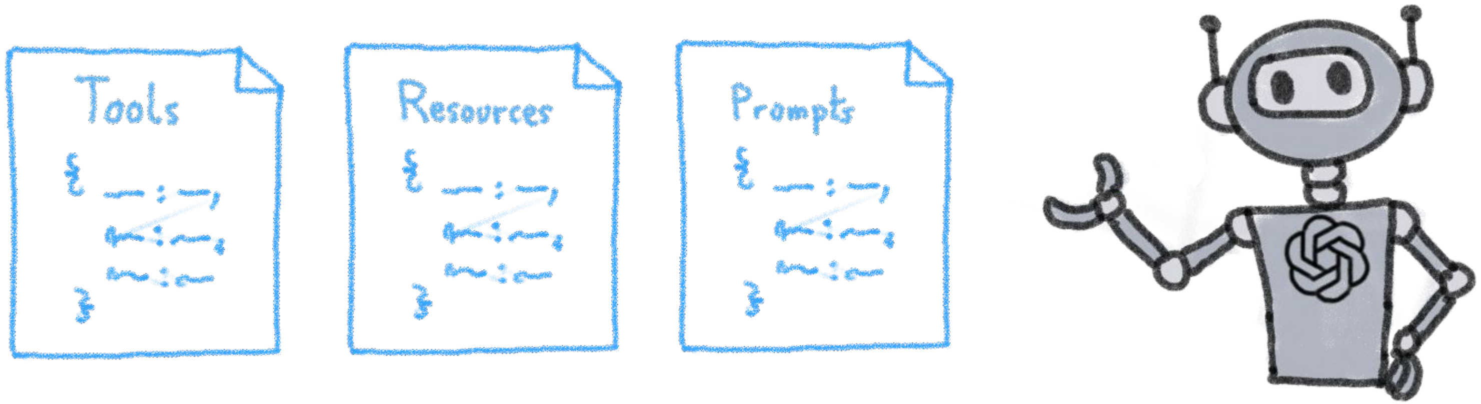


Full MCP architecture



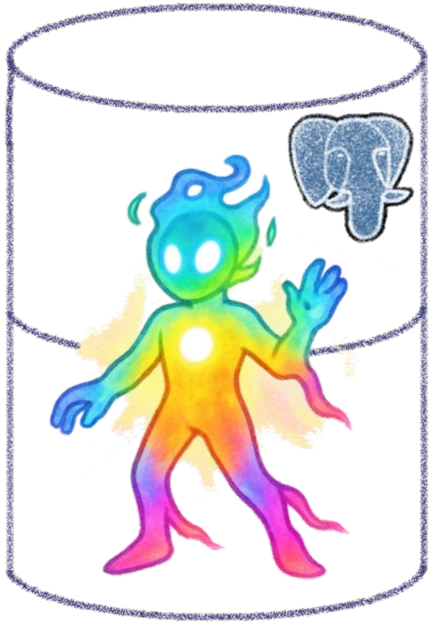
Services: tools, resources & prompts

- Tools
  - Standardized way to expose functions that can be invoked by clients
- Resources
  - Standardized way to expose resources to clients
  - Each resource is uniquely identified by a URI
- Prompts
  - Standardized way to expose prompt templates to clients
  - Structured messages and instructions for interacting with LLMs



MCPs are APIs

And they should be architectured in a similar way



Developer Expectations Have Shifted

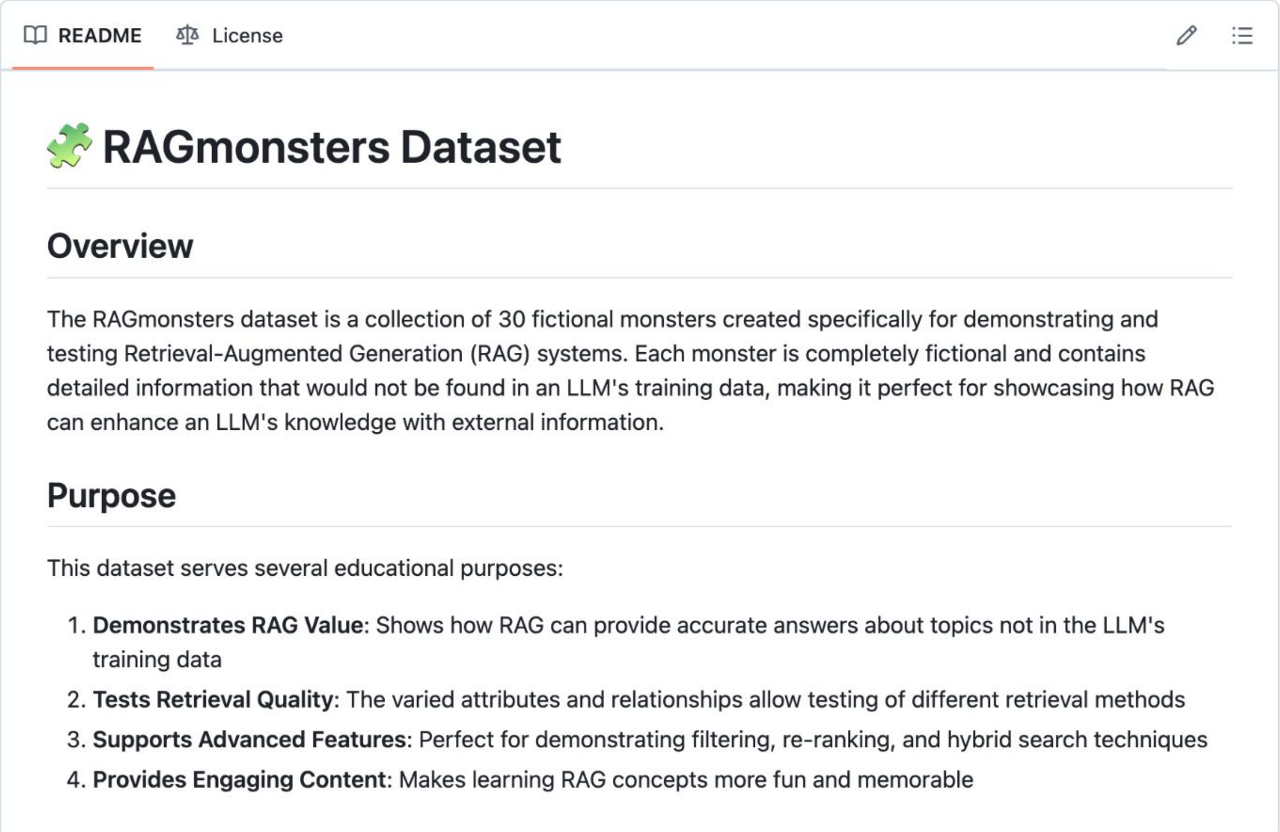
Winter 2024–2025

- “What is MCP?”
- “How do I connect my DB?”

Summer 2025

- “How do I build smarter MCP servers?”
- “How do I secure them?”
- “How do they fit into agent workflows?”

Let's use an example: RAGmonsters



🧩 RAGmonsters Dataset

Overview

The RAGmonsters dataset is a collection of 30 fictional monsters created specifically for demonstrating and testing Retrieval-Augmented Generation (RAG) systems. Each monster is completely fictional and contains detailed information that would not be found in an LLM's training data, making it perfect for showcasing how RAG can enhance an LLM's knowledge with external information.

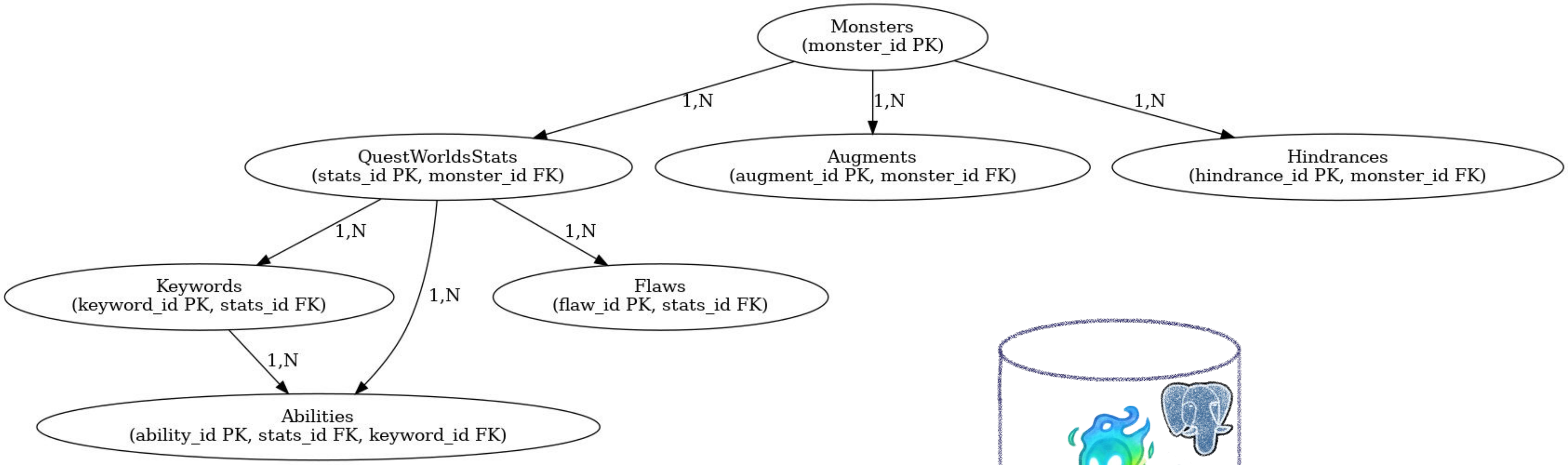
Purpose

This dataset serves several educational purposes:

1. **Demonstrates RAG Value**: Shows how RAG can provide accurate answers about topics not in the LLM's training data
2. **Tests Retrieval Quality**: The varied attributes and relationships allow testing of different retrieval methods
3. **Supports Advanced Features**: Perfect for demonstrating filtering, re-ranking, and hybrid search techniques
4. **Provides Engaging Content**: Makes learning RAG concepts more fun and memorable

https://github.com/LostInBrittany/RAGmonsters

RAGmonsters PostgreSQL Database


Monsters
(monster_id PK)

1,N

QuestWorldsStats
(stats_id PK, monster_id FK)

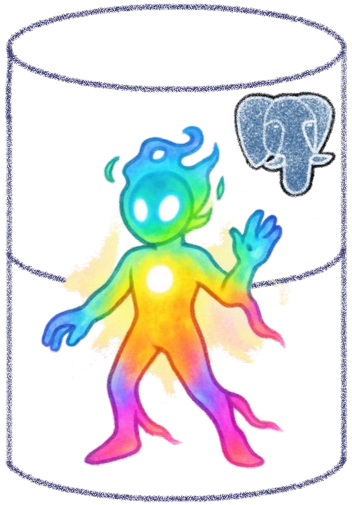
Augments
(augment_id PK, monster_id FK)

Hindrances
(hindrance_id PK, monster_id FK)

Keywords
(keyword_id PK, stats_id FK)

Flaws
(flaw_id PK, stats_id FK)

Abilities
(ability_id PK, stats_id FK, keyword_id FK)

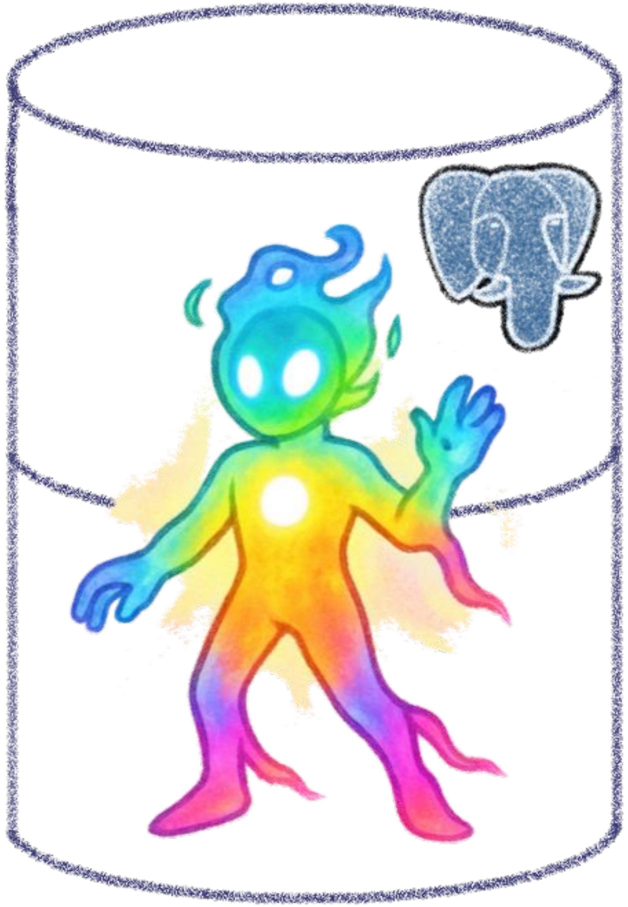


We want to allow LLM request it

Two options:

- A generic PostgreSQL MCP server
- A custom-made MCP server tailored for RAGmonsters

Which one to choose?



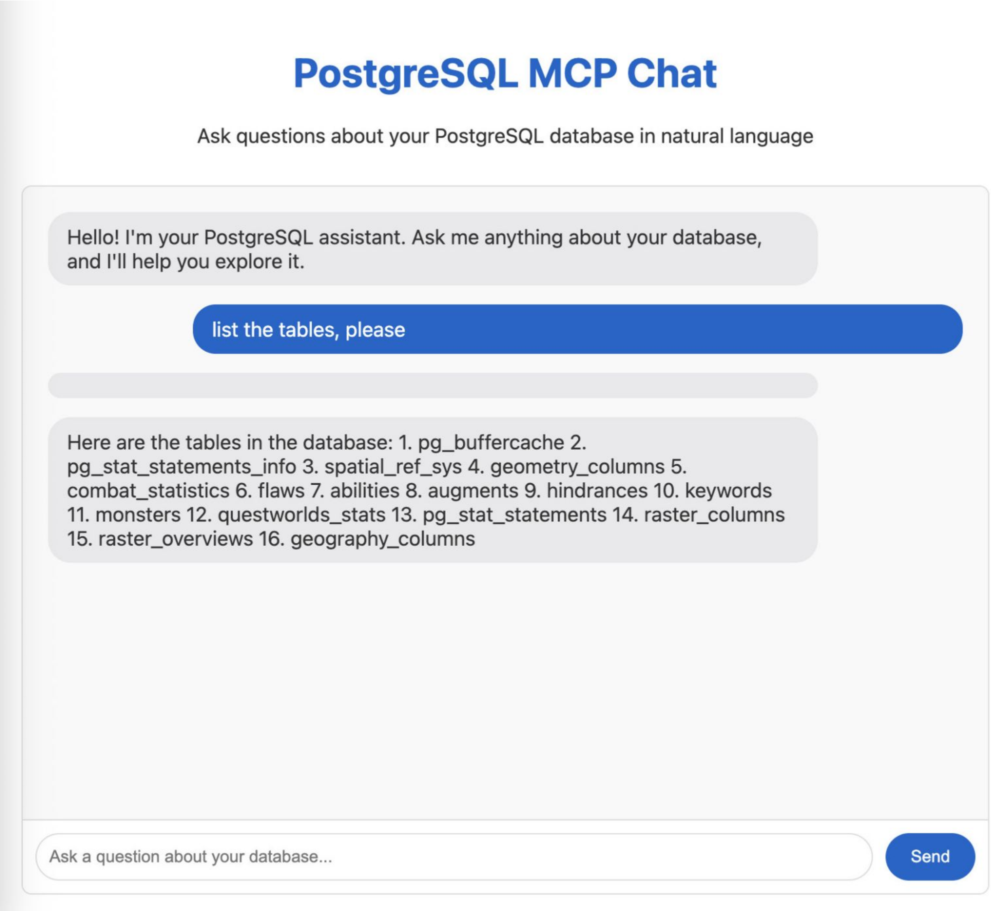
Generic PostgreSQL MCP server

Using PostgreSQL MCP Server

- A Resource that give the table schema for tables: `/schema`
- A Tool that allows to do SQL queries: `query`

LLM can know what tables do we have and what is their structure, and it can request them

Implementation:
https://github.com/CleverCloud/mcp-pg-example
PostgreSQL MCP Server:
https://github.com/modelcontextprotocol/servers/tree/main/src/postgres



PostgreSQL MCP Chat

Ask questions about your PostgreSQL database in natural language

Hello! I'm your PostgreSQL assistant. Ask me anything about your database, and I'll help you explore it.

list the tables, please

Here are the tables in the database: 1. pg_buffercache 2. pg_stat_statements_info 3. spatial_ref_sys 4. geometry_columns 5. combat_statistics 6. flaws 7. abilities 8. augments 9. hindrances 10. keywords 11. monsters 12. questworlds_stats 13. pg_stat_statements 14. raster_columns 15. raster_overviews 16. geography_columns

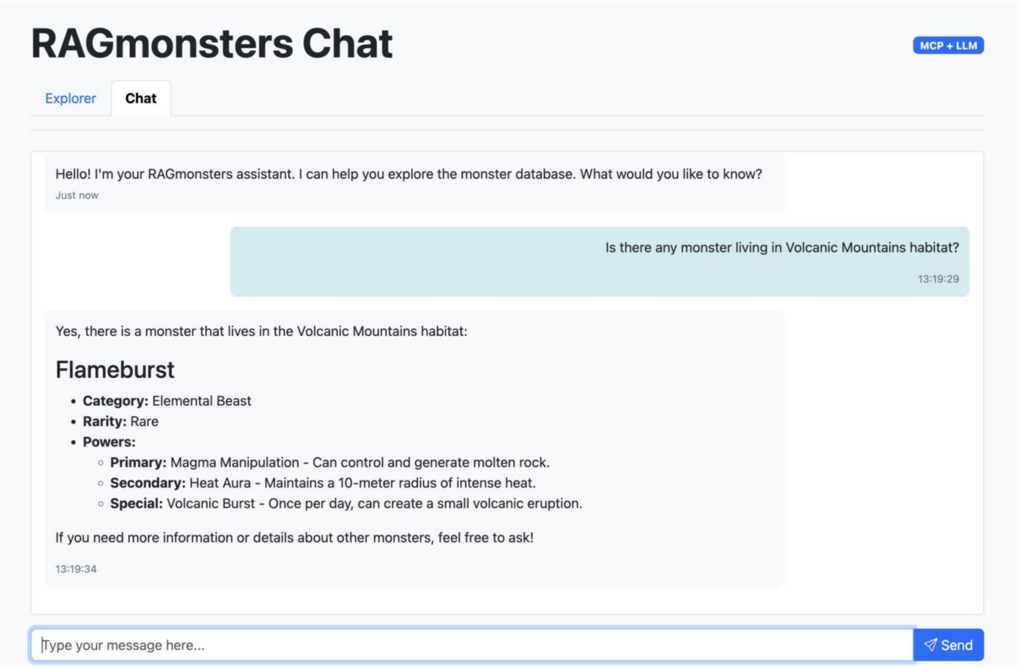
Custom-made RAGmonsters MCP server

Coding a MCP server for it. It offers targeted tools:

- `getMonsterByName`: fetches detailed information about a monster.
- `listMonstersByType`: Lists monsters of a given type.

- Easy, intuitive interactions for LLMs.
- Optimized for specific use cases.
- Secure (no raw SQL).

Implementation:
https://github.com/LostInBrittany/RAGmonsters-mcp-pg



Hello! I'm your RAGmonsters assistant. I can help you explore the monster database. What would you like to know?
Just now

Is there any monster living in Volcanic Mountains habitat?
13:19:29

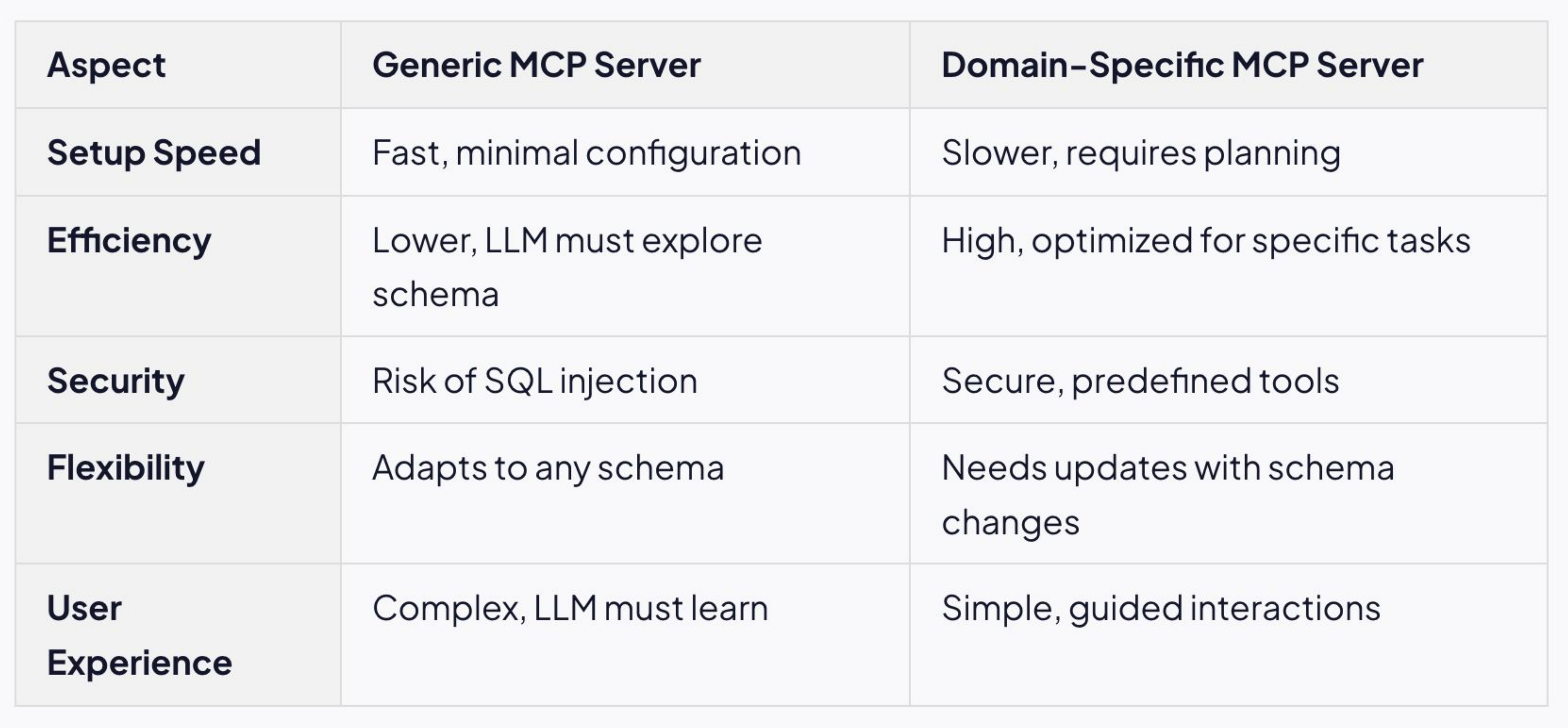
Yes, there is a monster that lives in the Volcanic Mountains habitat:

Flameburst

- **Category:** Elemental Beast
- **Rarity:** Rare
- **Powers:**
  - **Primary:** Magma Manipulation - Can control and generate molten rock.
  - **Secondary:** Heat Aura - Maintains a 10-meter radius of intense heat.
  - **Special:** Volcanic Burst - Once per day, can create a small volcanic eruption.

If you need more information or details about other monsters, feel free to ask!
13:19:34

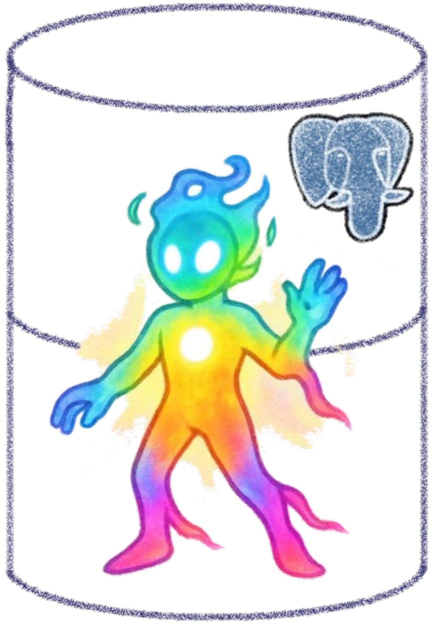
How to choose?



Aspect	Generic MCP Server	Domain-Specific MCP Server
Setup Speed	Fast, minimal configuration	Slower, requires planning
Efficiency	Lower, LLM must explore schema	High, optimized for specific tasks
Security	Risk of SQL injection	Secure, predefined tools
Flexibility	Adapts to any schema	Needs updates with schema changes
User Experience	Complex, LLM must learn	Simple, guided interactions

But how to do it?

Some down-to-Earth, practical advices



Design principles

What “good” looks like

- **Narrow, named capabilities**
  each tool should read like a product verb: `getMonsterByName`, `listMonstersByType`, `compareMonsters`.
- **Stable types in/out**
  explicit schemas (IDs, enums, unions) so the agent can plan reliably.
- **Deterministic behavior**
  same inputs → same outputs; include idempotencyKey when making state changes.
- **Least privilege**
  tools do one thing; internal queries/side-effects are not exposed.
- **Guardrails at the edge**
  validate inputs, clamp result sizes, redact PII, enforce authZ inside the server.

Capability modeling

Turn “tasks” into MCP tools/resources/prompts

Tools (actions)

- Read: `getMonsterByName(name) -> Monster`
- List: `listMonstersByType(type, limit=25, cursor?) -> {items:[Monster], nextCursor}`
- Search: `searchMonsters(q, limit=10) -> [MonsterSummary]`

Resources (documents/URIs the client can browse/fetch)

- `ragmonsters://schema/Monster` (JSON schema for types)
- `ragmonsters://docs/query-tips` (compact usage notes)
- `ragmonsters://images/{monsterId}` (read-only asset stream)

Prompts (reusable instructions/templates)

- prompt://ragmonsters/answering-style (tone, do/don’t)
- prompt://ragmonsters/disambiguation (ask for missing fields first)

Input contracts

Make the LLM succeed on the first try

- Refer enums & unions for fields the model tends to invent
  `type ∈ {BEAST, ELEMENTAL, UNDEAD,…}`
- Add optional “`reason`”/“`intent`” fields that your server ignores functionally but logs for eval
- Hard limits at the boundary: `limit ≤ 50, name.length ≤ 64, q.length ≤ 120`

```
{
 "type": "object",
 "required": ["type"],
 "properties": {
   "type": {"enum": ["BEAST","ELEMENTAL","UNDEAD","CELESTIAL","HUMANOID"]},
   "limit": {"type":"integer","minimum":1,"maximum":50},
   "cursor": {"type":"string"}
 }
}
```

Output shape

Make it composable

Always return a machine part and a human part:

- `data`: typed payload the client/agent can chain.
- `summary`: 1–2 lines the model can quote.
- `next`: cursors or suggested follow-ups.

```
{
 "data": { "items": [ { "id":"glowfang", "type":"BEAST", "danger":3 } ],
"nextCursor":"abc123" },
 "summary": "Found 1 beast: Glowfang (danger 3)." ,
 "next": ["getMonsterByName('glowfang')" ]
}
```

Security & governance

Baked into the server

- **AuthN**: accept a caller token; map to user/roles inside your server.
- **AuthZ**: per-tool role checks (viewer, editor, admin).
- **Data scope**: inject row-level filters (tenant, project) before hitting storage.
- **Rate limits**: e.g., 60 rpm per user; lower for heavy tools.
- **Redaction**: never return secrets; hash IDs in logs.
- **Explainability**: include source/policy notes in responses where relevant.

Observability & evaluation

From the beginning

- **Structured logs**
  `{tool, userId, durationMs, ok, errorCode}`
- **Traces**
  around datastore calls; record row counts
- **Golden tasks**
  keep a small suite (10–20) of representative prompts; run nightly
- **Safety tests**:
  prompt-injection set, over-broad queries, boundary limits

Conclusion

- Generic MCP servers
  Quick to set up, flexible, but less efficient and more error-prone.
- Domain-specific MCP servers
  Safer and faster for targeted tasks, but need more upfront design.

Choose wisely
Use generic for exploration, domain-specific for production.

A bit like for REST APIs, isn't it?

The road ahead

- MCP is quickly becoming the *lingua franca* of agents.
- We’re still early — best practices are being shaped right now.
- Your design choices today will set the tone for secure, scalable agent ecosystems tomorrow.

That's all, folks!

Thank you all!