AI Agents in PHP: tool calling and Model Context Protocol (MCP)

Enrico Zimuel, *Tech Lead & Principal Software Engineer @ Elastic*

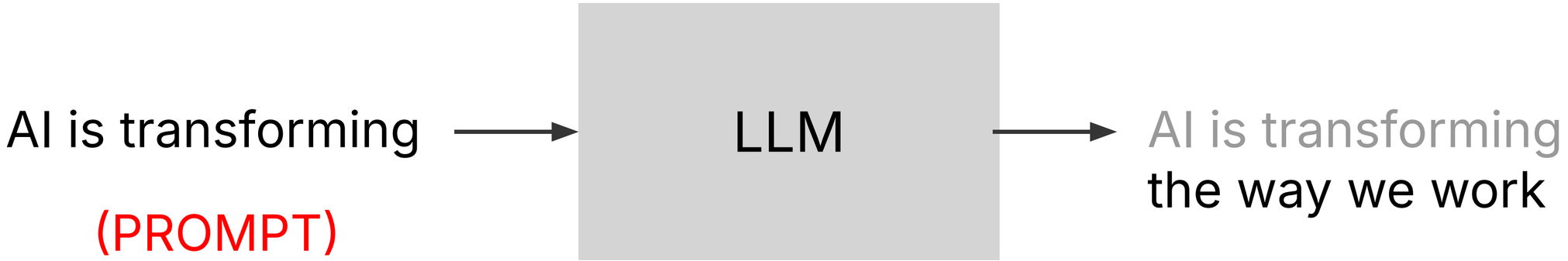
Programmer User Group Milan - June 16, 2025

Agenda

- Large Language Model: a quick intro
- Tool calling (or Function calling)
- Agentic AI
- Multi-agent systems
- Model Context Protocol
- Examples in PHP

LLM: a quick introduction

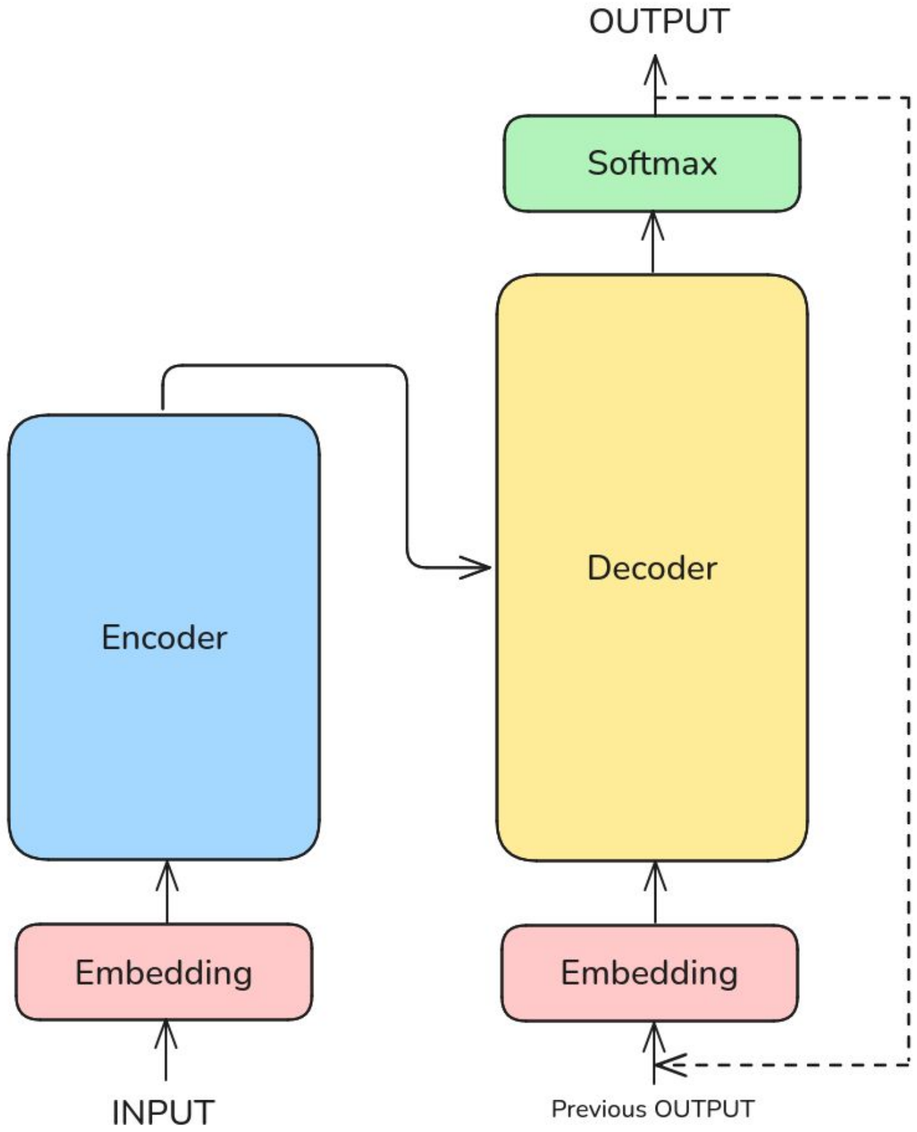
- **Large Language Model** (LLM) are probabilistic models that produce sentence in natural language
- These models work by completing sentences



elastic

Transformer architecture

- Introduced in [Attention is All You Need] paper in 2017
- Basement of all LLMs
- The sentences are analyzed using a **self-attention** mechanism: each part of a sentence is evaluated in relation to every other part to understand contextual relationships and assign appropriate weights



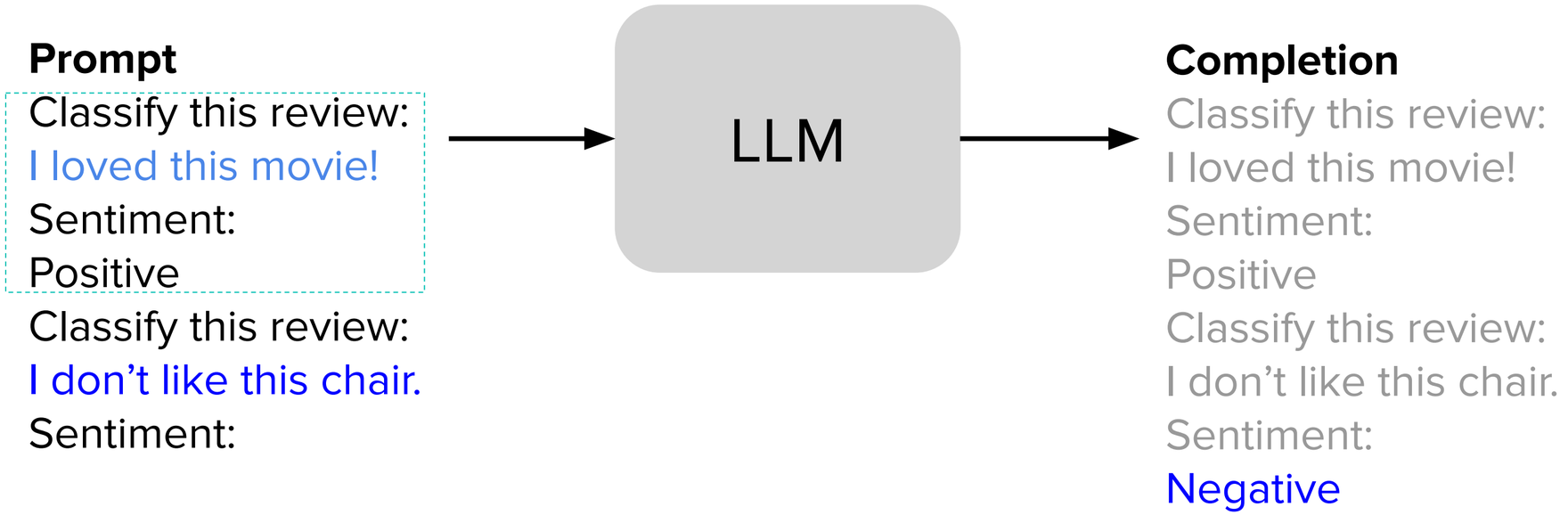
elastic

Prompt engineering

- You can encounter situations where the model doesn't produce the outcome that you want on the first try
- You may have to revisit the language several times to get a good answer
- The development and improvement of the prompt is known as **prompt engineering**
- One powerful strategy is to include examples of the task that you want the model to carry out inside the prompt
- This is called **In-Context Learning (ICL)**

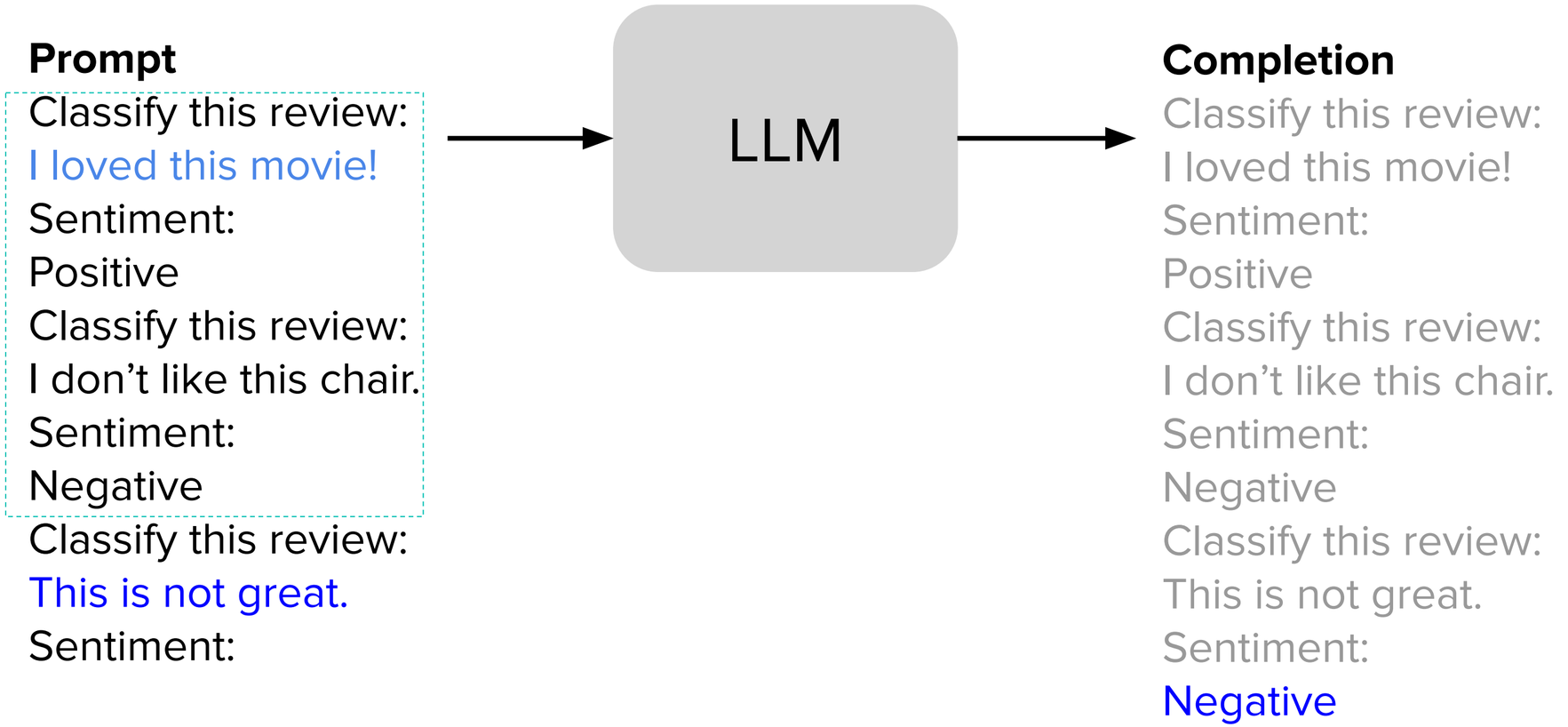
ICL - zero shot inference

ICL - one shot inference



elastic

ICL - few shot inference



elastic

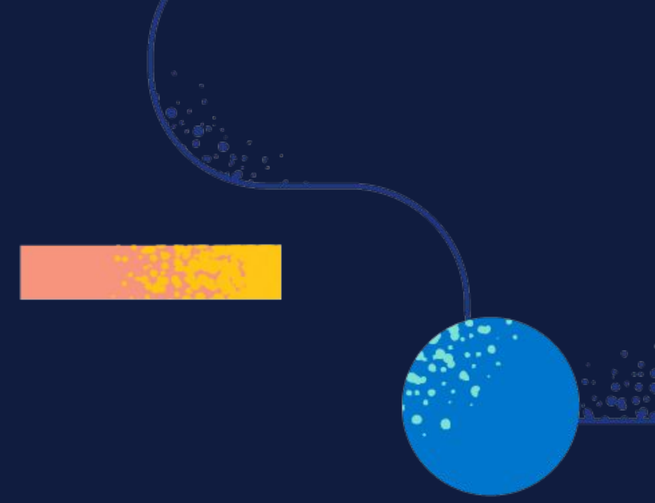
LLM limitations

- **Prone to Hallucinations**: Since an LLM is a probabilistic model, it can generate incorrect or nonsensical information
- **No sources:** The output of an LLM does not provide sources for its information (again hallucinations)
- **Fixed Knowledge**: The model's knowledge is static, meaning it does not learn or adapt from interactions
- **Difficult to Update**: Expanding an LLM’s knowledge requires retraining or fine-tuning, which is complex, resource-intensive, and time-consuming

Emerging properties

- *"Emergence is when quantitative changes in a system result in qualitative changes in behavior."* (P. Anderson, 1972)
- LLM models (big) appear to exhibit emergent properties (Wei et al. 2022)
- Emerging properties:
    - Question Answering
    - Summarization
    - In-Context Learning
    - Tool Calling and Coding
    - etc

Tool calling (Function calling)



elastic

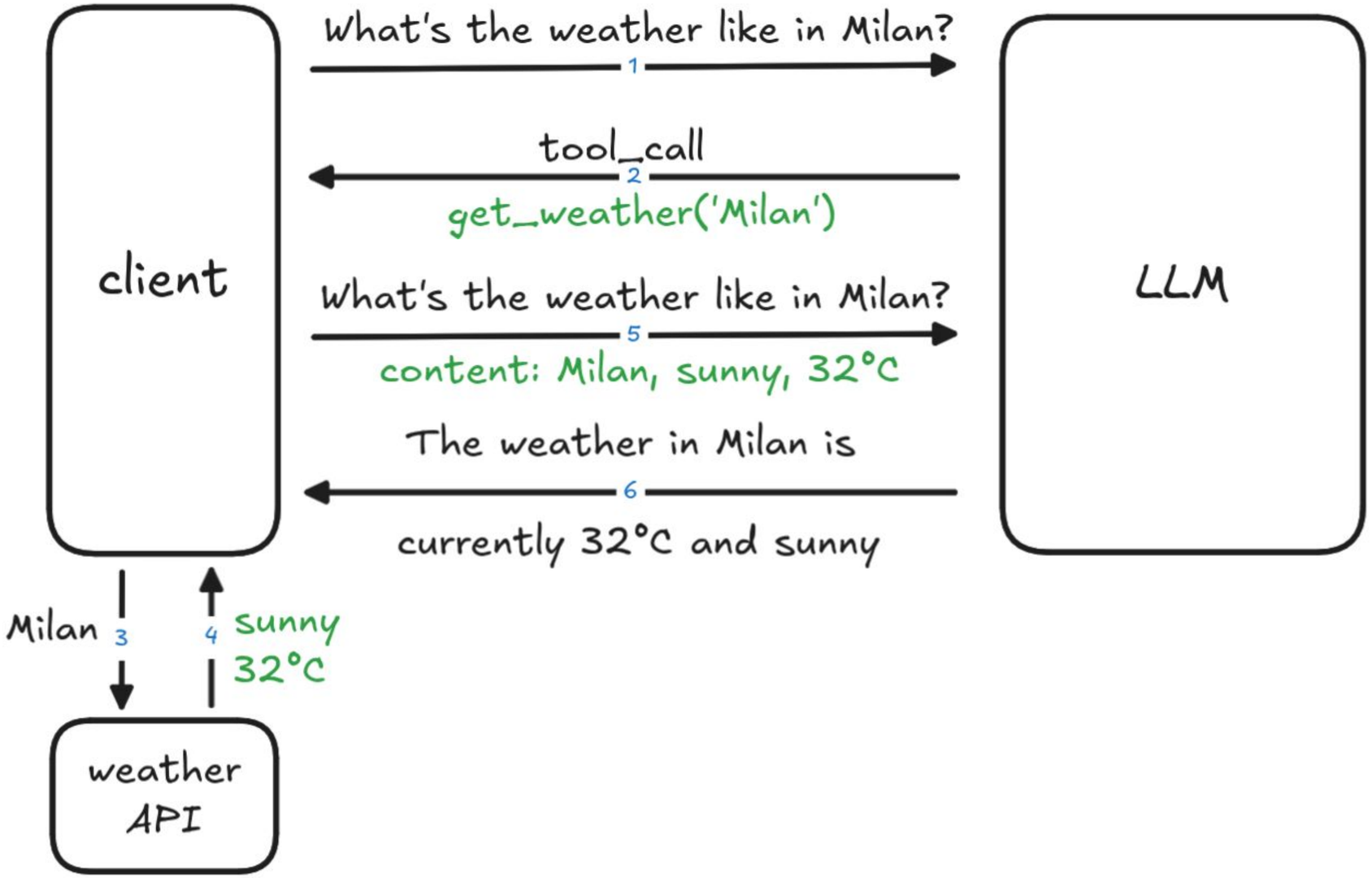
History of tool calling

- **Tool calling** (or function calling) is an emerging property in LLM
- Relevant papers that investigated the topic:
  - Nakano et al., WebGPT: Browser-assisted question-answering with human feedback, OpenAI, 2022
  - Timo Schick et al., Toolformer: Language Models Can Teach Themselves to Use Tools, Meta AI Research, 2023
  - Function calling and other API updates, OpenAI, June 13, 2023

Tool calling (or Function calling)

- Tool calling is the ability of LLM to recognize the need to execute external functions (tools) as part of its reasoning process
- The LLM recognizes when it needs of additional information or actions and request the usage of tools (preparing the generation a function call in JSON function)
- The client is responsible for executing the function call (not the LLM) and this step is usually monitored by a human

A diagram of Tool calling



Tool calling in OpenAI

```
POST  https://api.openai.com/v1/chat/completions
{
   "model": "gpt-4.1",
   "messages": [
      {
         "role": "user",
         "content": "What is the weather like in Milan today?"
      }
   ],
   "tools": [
      {
         "type": "function",
         "function": {
            "name": "get_weather",
            "description": "Get current temperature for a given location.",
            "parameters": {
               "type": "object",
               "properties": {
                  "location": {
                     "type": "string",
                     "description": "City and country e.g. Rome, Italy"
                  }
               },
               "required": [
                  "location"
               ],
               "additionalProperties": false
            },
            "strict": true
         }
      }
   ]
}
```

Response

```
[{
   "id": "call_12345xyz",
   "type": "function",
   "function": {
      "name": "get_weather",
      "arguments": "{\"location\":\"Milan, Italy\"}"
   }
}]
```

elastic

Tool calling in PHP

- OpenAI PHP client, supercharged community-maintained PHP API client that allows you to interact with OpenAI API
- LLPhant, A comprehensive PHP Generative AI Framework
- LLM Chain (experimental), library for building LLM-based and AI-based features and applications

Agentic AI

elastic

Agentic AI

- **Agentic AI** refers to AI systems that can act autonomously to achieve goals, making decisions and taking actions without direct human intervention
- An **agent** is usually a software that makes decisions and takes actions often by calling tools or APIs in a loop, based on its current understanding of the world and its objective
- Key features of an agent:
  - **Goal-oriented**
  - **Autonomous**
  - **Interactive**
  - **Iterative**: reason → act → observe → repeat
  - **Memory** (optional)

Agent workflow example

- **Goal:** Book a flight to Paris under $500.
  - **Plan (reason)**: “Check available flights.”
  - **Act**: Calls search_flights(to="Paris", max_price=500)
  - **Observe**: Gets a list of flights.
  - **Reason**: “There’s a flight on Tuesday under budget. Book it!”
  - **Act**: Calls book_flight(flight_id)
  - **Finish**: Reports the result back to the user.
- The agent monitors its own process, deciding what to do next based on each result

Multi-agent systems

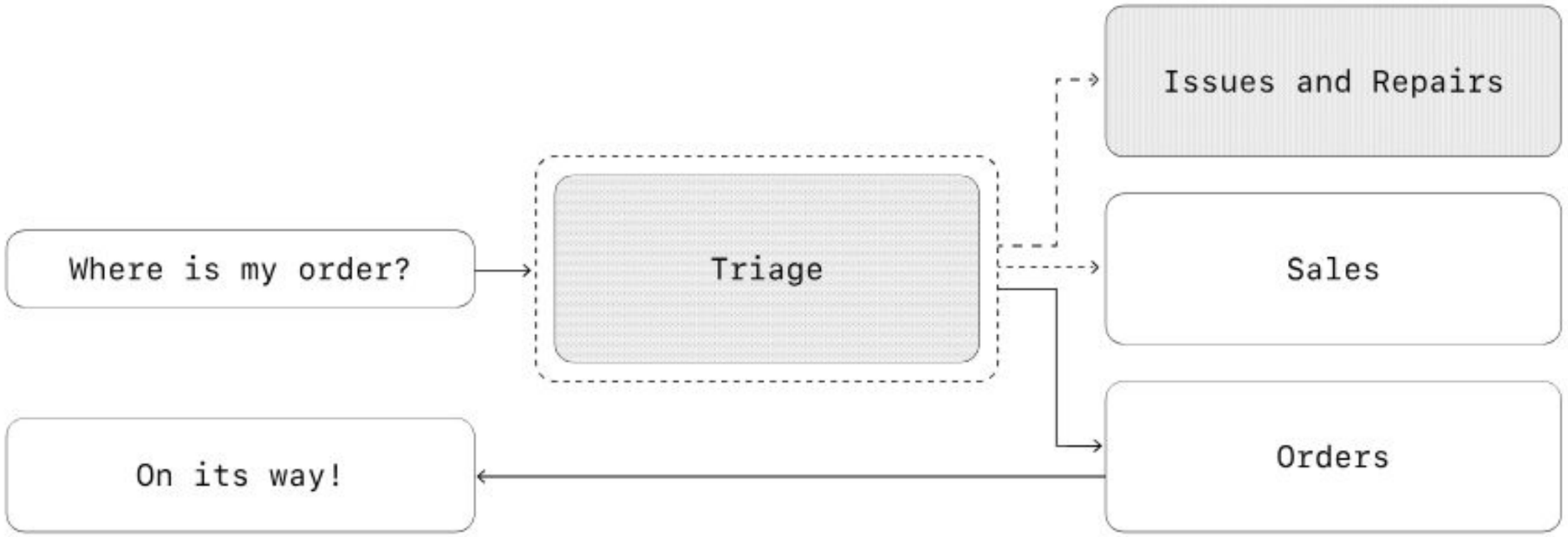
- Multi-agent systems consist of multiple agents that may be able to communicate with one another.
- The two most common architectures are:
  - **Manager pattern** – where agents are treated as tools by a central agent.
  - **Decentralized pattern** – where agents pass control to one another.
- These systems can be represented as graphs, where nodes are agents. In the manager pattern, edges represent tool calls. In the decentralized pattern, edges represent handoffs, where one agent transfers execution to another.

Manager pattern

Source: A practical guide to building agents, OpenAI

elastic

Decentralized pattern



Source: A practical guide to building agents, OpenAI

elastic

Limitations

- **Execution time:** not really fast since it requires many steps
- **Expensive**: agents typically use about 4x more tokens than chat interactions, multi-agent use about 15x more tokens than chat
- **Complexity**: multi-agent are good with parallel tasks but not so good to manage many dependencies between agents*
- **Unpredictability**: agent interactions can lead to unexpected outcomes. Needs of Guardrail systems.
- **Evaluation**: it's difficult to measure the collective success or alignment of multiple agents

* How we built our multi-agent research system, Anthropic

Examples in PHP

- Neuron-AI, PHP Agent Development Kit to build customizable, production-ready LLM applications

Model Context Protocol

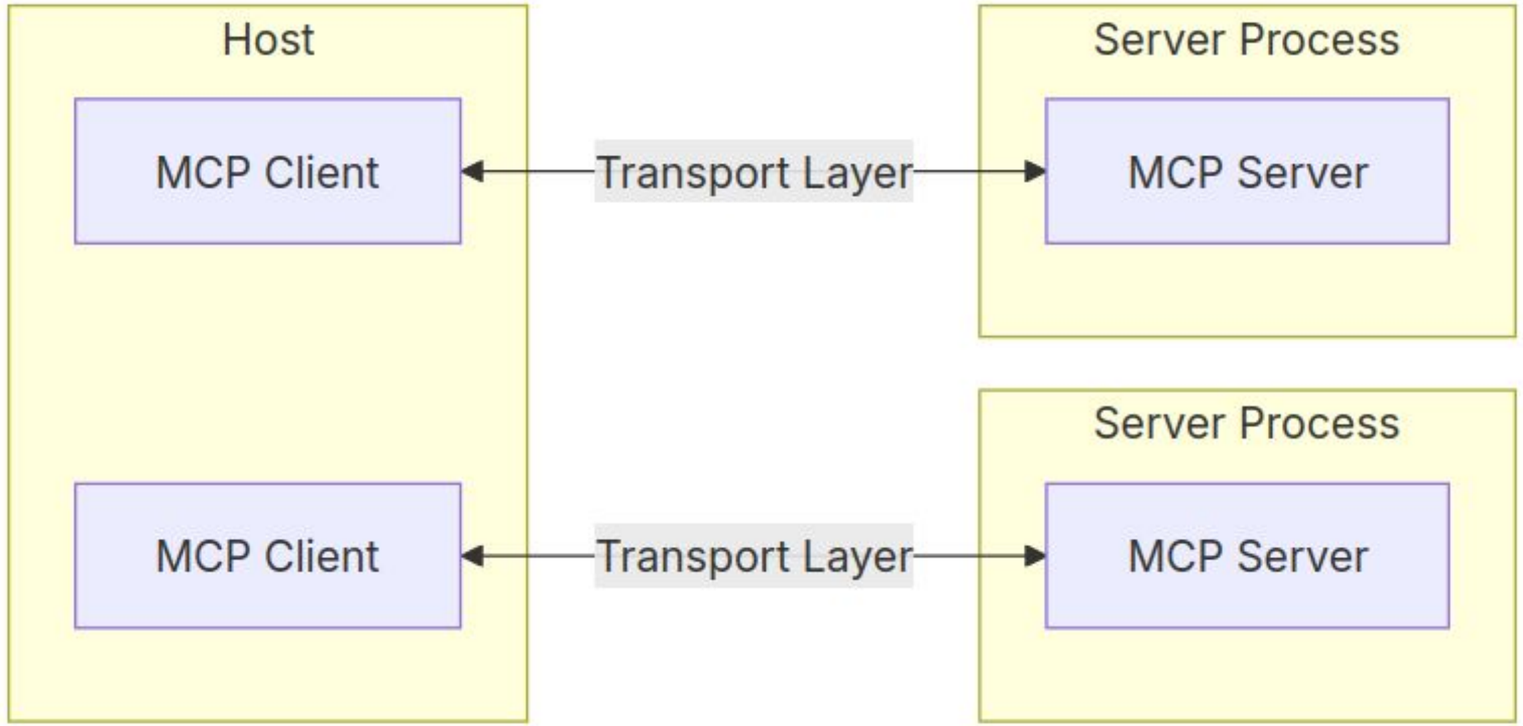
elastic

Model Context Protocol

- Model Context Protocol (MCP) is an open protocol that standardizes how applications provide context to LLMs
- MCP provides a standardized way to connect AI models to different data sources and tools
- Client/server architecture

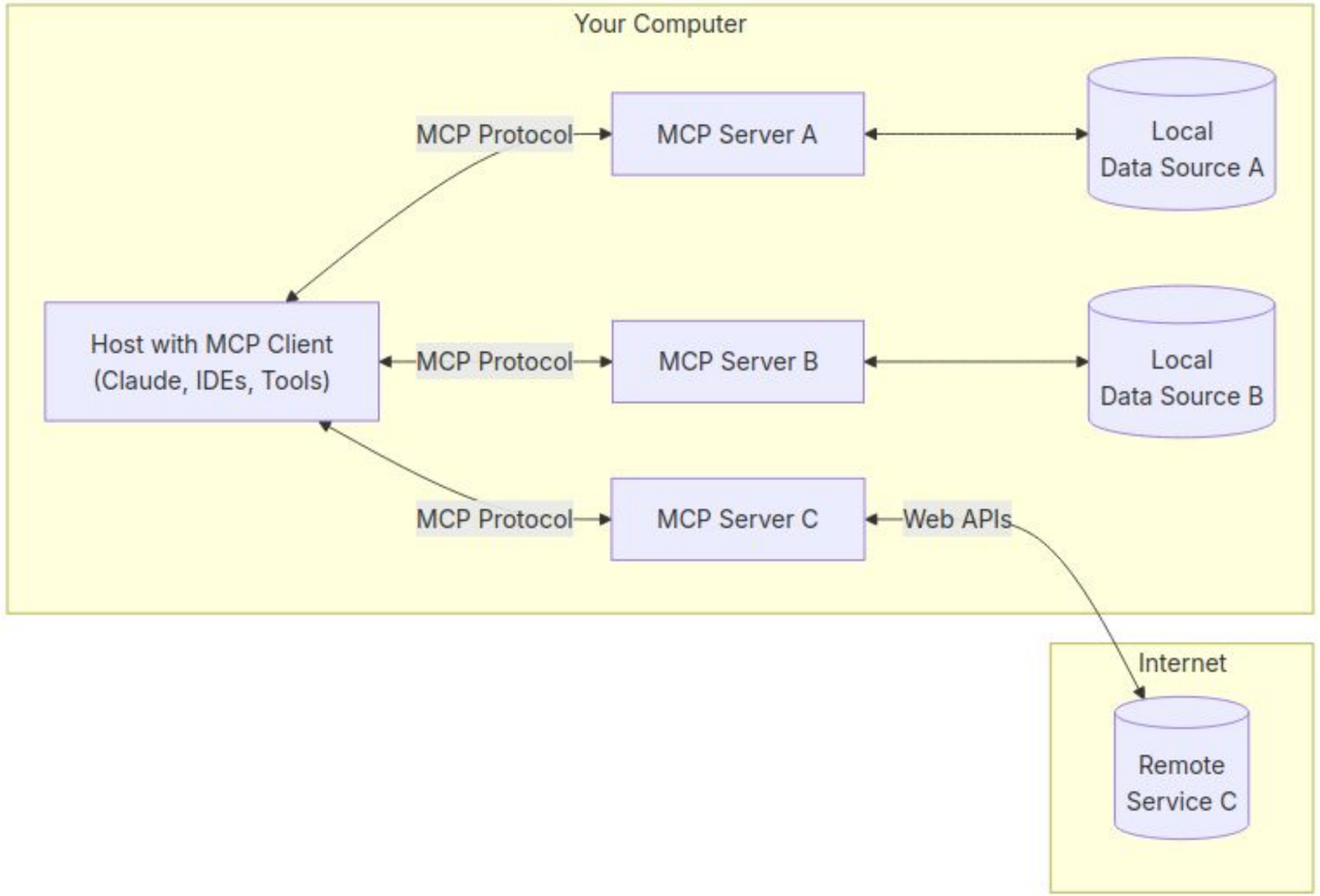
Core architecture

- MCP follows client-server architecture
- Transport layer: Stdio, Streamable HTTP
- All transport use JSON-RPC 2.0 to exchange messages



elastic

Example: multiple MCP servers



elastic

Core MCP concepts

- MCP servers can provide three main types of capabilities:
  - **Resources**: file-like data that can be read by clients (like API responses or file contents). Clients can discover available resources using the *resource/list* endpoint
  - **Tools**: functions that can be called by the LLM (with user approval). Clients can list available tools through the *tools/list* endpoint
  - **Prompts**: pre-written templates that help users accomplish specific tasks
  - **Sampling**: allows servers to request LLM completions through the client
  - **Roots**: define the boundaries where servers can operate

Examples in PHP

- php-mcp/server, core PHP implementation for the Model Context Protocol (MCP) server
- php-mcp/client, core PHP implementation for the Model Context Protocol (MCP) Client

References

- Ashish Vaswani et al., Attention Is All You Need, NIPS, 2017
- Jason Wei et al., Emergent Abilities of Large Language Models, Published in Transactions on Machine Learning Research, 2022
- Ilan Bigio, Function Calling is All You Need, OpenAI
- Reiichiro Nakano et al., WebGPT: Browser-assisted question-answering with human feedback, OpenAI, 2022
- Timo Schick et al., Toolformer: Language Models Can Teach Themselves to Use Tools, Meta AI Research, 2023
- Function calling and other API updates, OpenAI, June 13, 2023
- A practical guide to building agents, OpenAI
- How we built our multi-agent research system, Anthrophic, 2025

Thank you!