Resources in MCP Servers

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

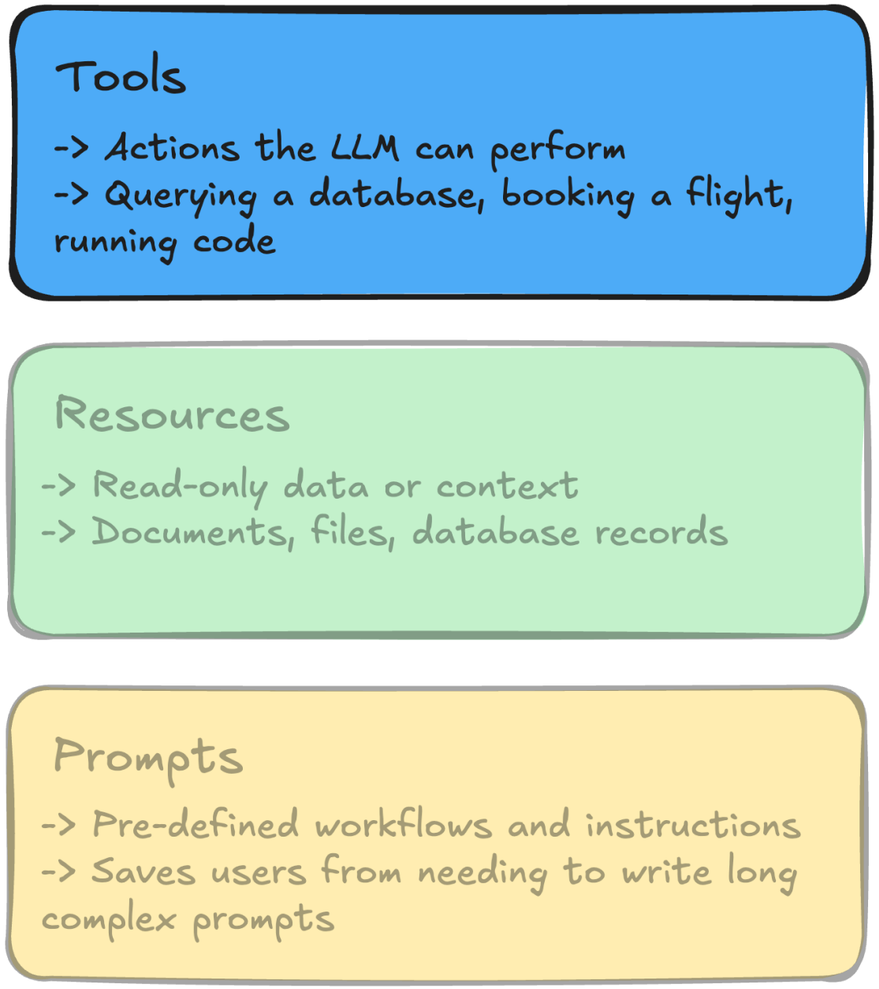


James Chapman
AI Curriculum Manager, DataCamp

What are MCP Resources?

Chapter 2

- Resources + Prompts



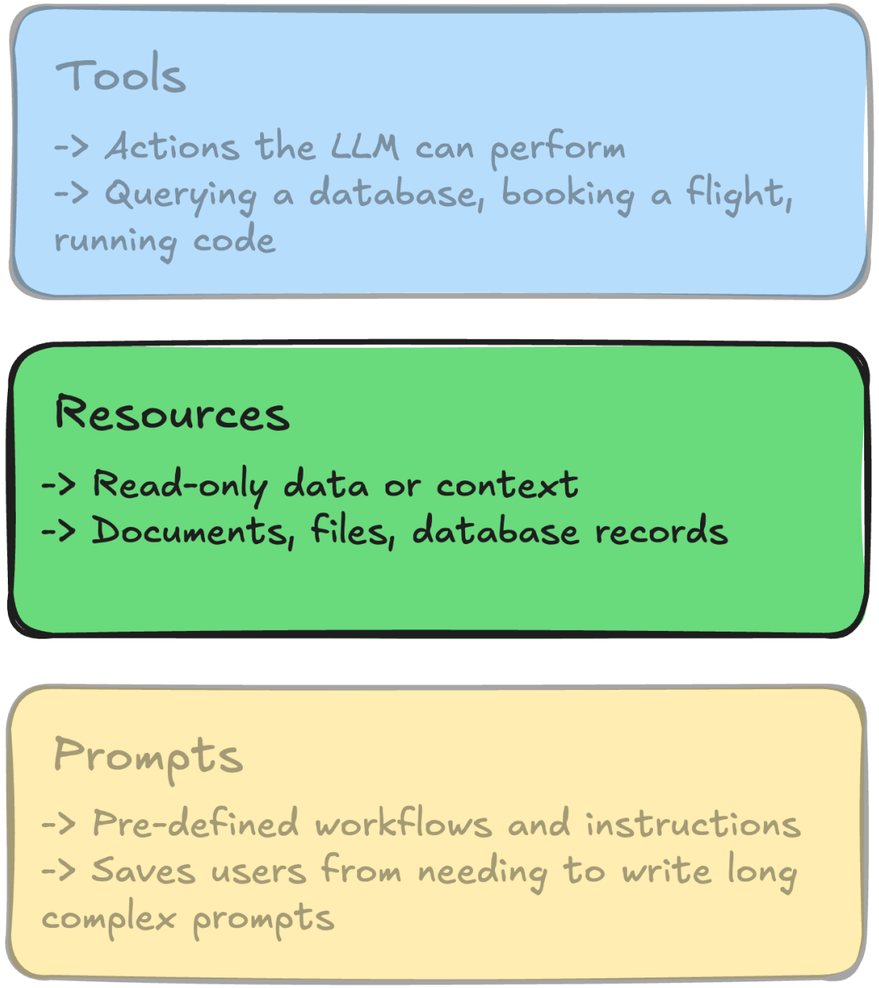
What are MCP Resources?

Chapter 2

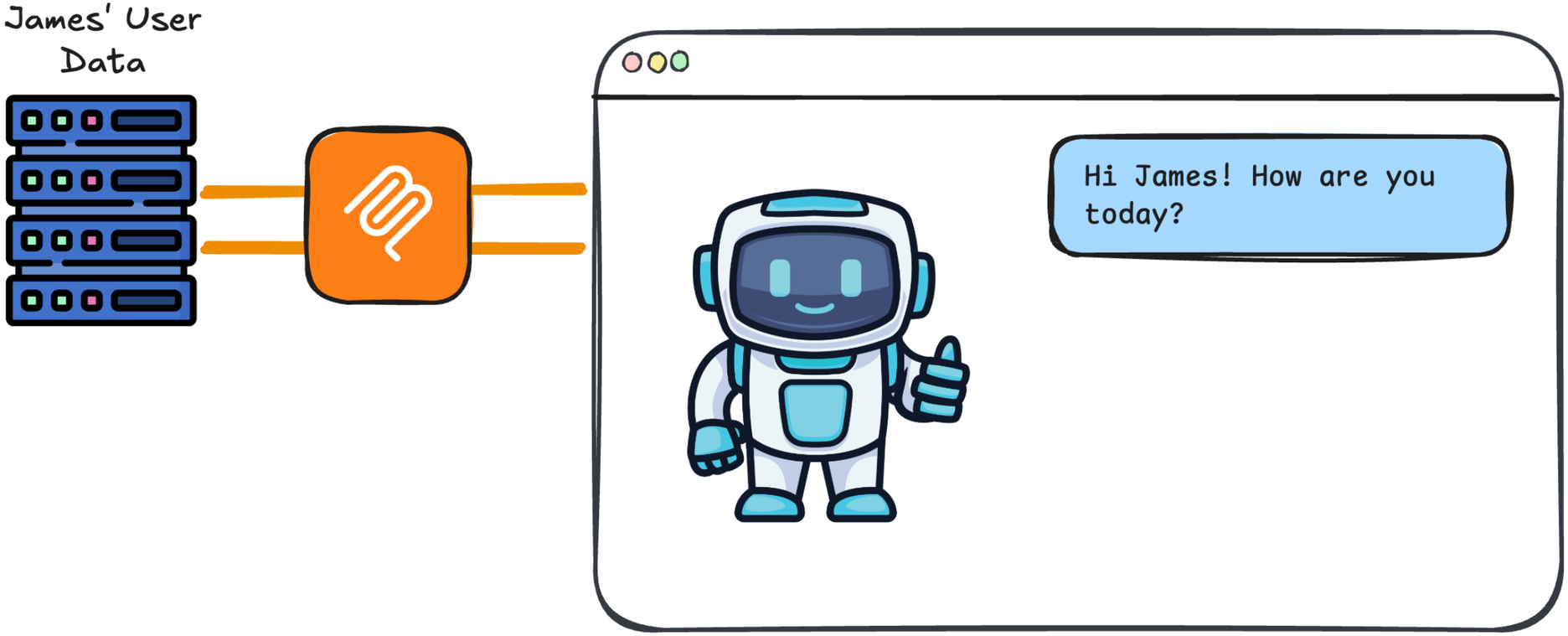
- Resources and prompts
- Integration with APIs and databases

Resources: read-only data or data objects retrieved by the MCP client

- *Not* necessarily called by an LLM
- *Application-Driven* → e.g., user preferences
- *User-Driven* → e.g., file upload



MCP Resources in AI Apps



Defining MCP Server Resources

```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Timezone Converter")

@mcp.resource("file://locations.txt")
def get_locations():
    try:
        with open('locations.txt', 'r') as f:
            content = f.read()
        return content
    except FileNotFoundError:
        return "locations.txt file not found"
```

- **URI**: uniform resource identifier
  - Can point to local files, APIs, remote servers, and more
  - Can dynamically insert parameters like User ID
- `locations.txt` (server-side) → list of timezone locations

Defining MCP Server Resources

```python
@mcp.resource("file://locations.txt")
def get_locations() -> str:
    """
    Get the list of cities for timezone conversion.

    Returns:
        Contents of the locations.txt file with city names
    """
    try:
        with open('locations.txt', 'r') as f:
            content = f.read()
        return content
    except FileNotFoundError:
        return "locations.txt file not found"
```

Saving the Server: timezone_server.py

```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Timezone Converter")

# Tools from before...

@mcp.resource("file://locations.txt")
def get_locations() -> str:
    # ...

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Client: Listing Resources

```python
async def list_resources():
    """List all available resources from the MCP server."""
    params = StdioServerParameters(command=sys.executable, args=["timezone_server.py"])

    async with stdio_client(params) as (reader, writer):
        async with ClientSession(reader, writer) as session:
            await session.initialize()
            response = await session.list_resources()

            print("Available resources:")
            for resource in response.resources:
                print(f" - {resource.uri}")
                print(f"   Name: {resource.name}")
                print(f"   Description: {resource.description}")

            return response.resources
```

Client: Listing Resources

```
print(asyncio.run(list_resources()))
```

```
Available resources:
 - file://locations.txt/
   Name: get_locations
   Description:
    Get the list of cities for timezone conversion.

    Returns:
        Contents of the locations.txt file with city names

[Resource(name='get_locations', title=None, uri=AnyUrl('file://locations.txt/'),
          description='\n    Get the list of cities for timezone conversion.\n\n...',
          mimeType='text/plain', size=None, icons=None, annotations=None, meta=None)]
```

Client: Reading Resources

```python
async def read_resource(resource_uri: str):
    """Read a specific resource by URI."""
    params = StdioServerParameters(command=sys.executable, args=["timezone_server.py"])

    async with stdio_client(params) as (reader, writer):
        async with ClientSession(reader, writer) as session:
            await session.initialize()

            print(f"Reading resource: {resource_uri}")
            resource_content = await session.read_resource(resource_uri)

            for content in resource_content.contents:
                print(f"\nContent ({content.mimeType}):")
                print(content.text)

            return resource_content
```

Client: Reading Resources

```
print(asyncio.run(read_resource('file://locations.txt/')))
```

```
Reading resource: file://locations.txt

Content (text/plain):
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
...

ReadResourceResult(meta=None, contents=[TextResourceContents(uri=AnyUrl('file://locations.txt/'),
                   mimeType='text/plain', meta=None, text='Africa/Abidjan\nAfrica/Accra...')])
```

Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

Prompts in MCP Servers

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)



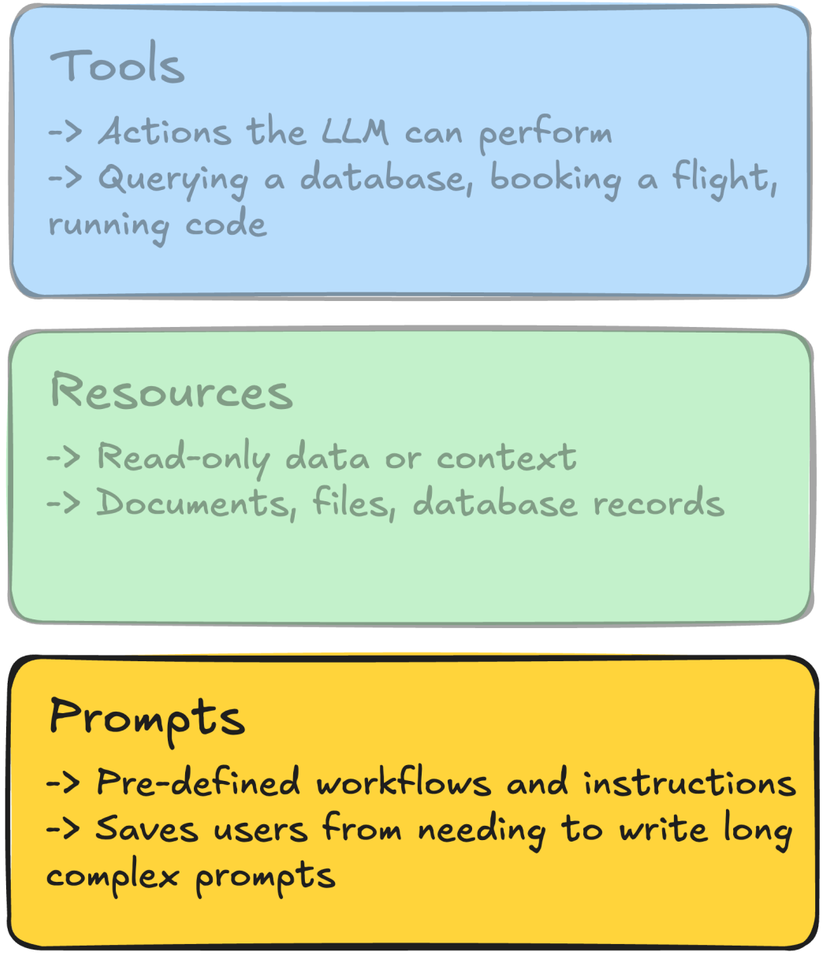
James Chapman
AI Curriculum Manager, DataCamp

datacamp

What are MCP Prompts?

Prompts: reusable templates that optimize LLMs for specific tasks

- Workflows and behaviors
- Reduces prompt-load on users



The Need for Prompting

Missing Source Time

What time is it now in Dublin, Ireland?

Missing Source Timezone

It's 3:27pm here. What time is it in Lisbon, Portugal?

Relative Time References

What was the time in Madrid, Spain at 6:13pm last Tuesday in the UK?

Ambiguous Target Timezone

Convert 3am UK time into Canadian time.

The Timezone Converter Prompt

```
You are a timezone conversion engine.

Your task is to:
1. Extract the source datetime from the user's natural language input.
2. Identify the source timezone (explicit or inferred).
3. Convert the datetime into the target timezone.

Rules:
- If the date is ambiguous (e.g., "next Friday"), resolve it relative to the provided datetime.
- If the input cannot be resolved confidently, seek clarification.
```

→ Expose this prompt for the timezone conversion task

Defining MCP Server Prompts

```
@mcp.prompt(title="Timezone Conversion")
def convert_timezone_prompt(user_input: str) -> str:
    return f"""You are a timezone conversion engine.

Your task is to:
1. Extract the source datetime...

Rules:
- If the date is ambiguous (e.g., "next Friday"), resolve it relative to the provided datetime.
- If the input cannot be resolved confidently, seek clarification.

    User's timezone conversion request: {user_input}"""
```

Local MCP Server: timezone_server.py

```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Timezone Converter")

# Tools and resources from before...

@mcp.prompt(title="Timezone Conversion")
def convert_timezone_prompt(timezone_request: str) -> str:
    # ...

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Client: Listing Prompts

```
async def list_prompts():
    """List all available prompts from the MCP server."""
    params = StdioServerParameters(command=sys.executable, args=["timezone_server.py"])

    async with stdio_client(params) as (reader, writer):
        async with ClientSession(reader, writer) as session:
            await session.initialize()
            # List available prompts
            prompts = await session.list_prompts()
            print(f"Available prompts: {[p.name for p in prompts.prompts]}")
```

Client: Listing Prompts

```
print(asyncio.run(list_prompts()))
```

```
Available prompts: ['convert_timezone_prompt']
```

The prompt's `name` is not the same as the `title`

```
@mcp.prompt(title="Timezone Conversion")
def convert_timezone_prompt(timezone_request: str) -> str:
    # ...
```

Client: Listing Prompts

```
print(asyncio.run(list_prompts()))
```

```
Available prompts: ['convert_timezone_prompt']
```

The prompt's `name` is not the same as the `title`

```
# List available prompts
prompts = await session.list_prompts()
print(f"Available prompts: {[p.name for p in prompts.prompts]}")
```

- Used the `.name` attribute in the client (not `.title`)

Client: Retrieving Prompts

```python
async def read_prompt(user_input: str, prompt_name: str = "convert_timezone_prompt") -> str:
    """Retrieve a prompt from the MCP server with user input."""
    params = StdioServerParameters(command=sys.executable, args=["timezone_server.py"])

    async with stdio_client(params) as (reader, writer):
        async with ClientSession(reader, writer) as session:
            await session.initialize()

            prompts = await session.list_prompts()
            # Retrieve the prompt
            if prompts.prompts:
                prompt = await session.get_prompt(prompt_name, arguments={"timezone_request": user_input})
                print(f"Prompt result: {prompt.messages[0].content.text}")

            return prompt.messages[0].content.text
```

```
print(asyncio.run(read_prompt(user_input="It is 9:50 AM in the UK in January. What time is
    it in Lisbon, Portugal?")))
```

```
You are a timezone conversion engine.

Your task is to:
1. Extract the source datetime from the user's natural language input.
2. Identify the source timezone (explicit or inferred).
3. Convert the datetime into the target timezone.

Rules:
- If the date is ambiguous (e.g., "next Friday"), resolve it relative to the...
- If the input cannot be resolved confidently, seek clarification.

User's timezone conversion request: It is 9:50 AM in the UK in January. What time is it in
Lisbon, Portugal?
```

Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

MCP and LLMs: Tools

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)



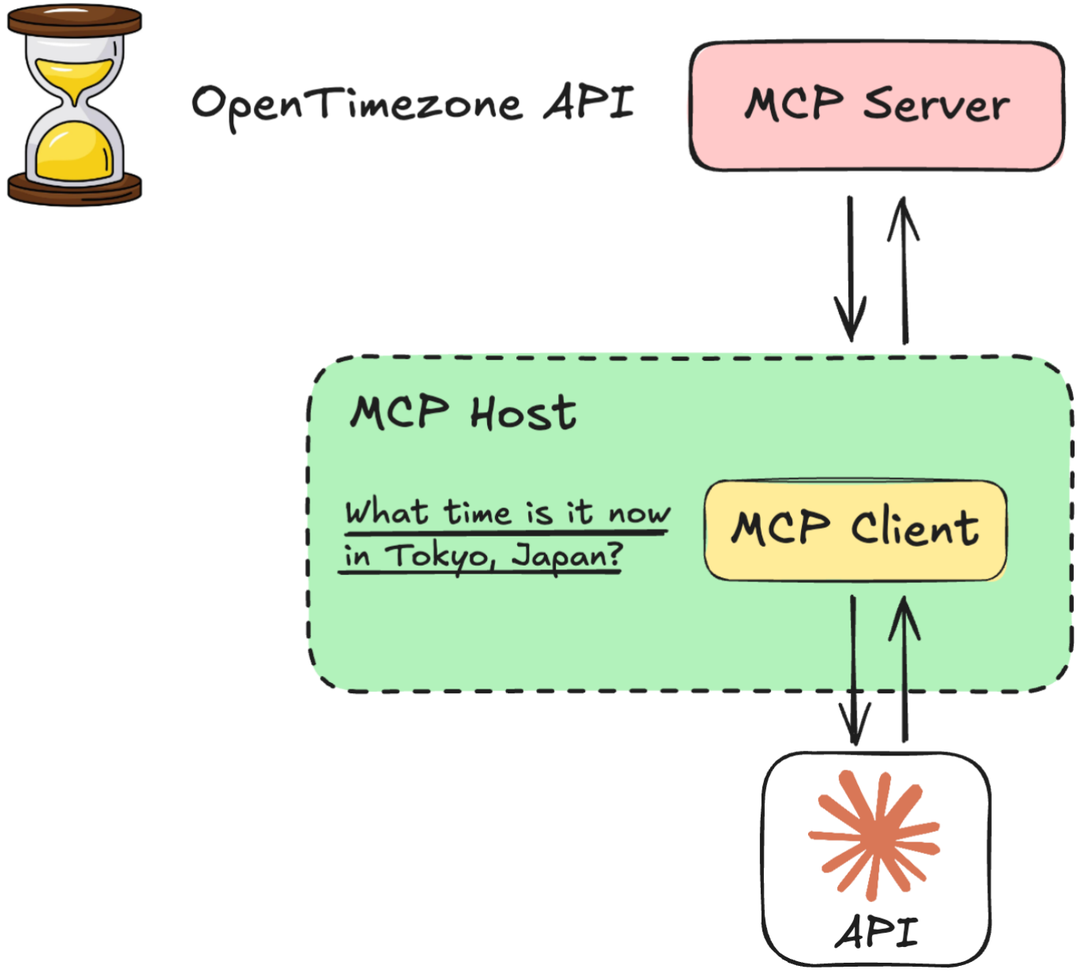
James Chapman
AI Curriculum Manager, DataCamp

datacamp

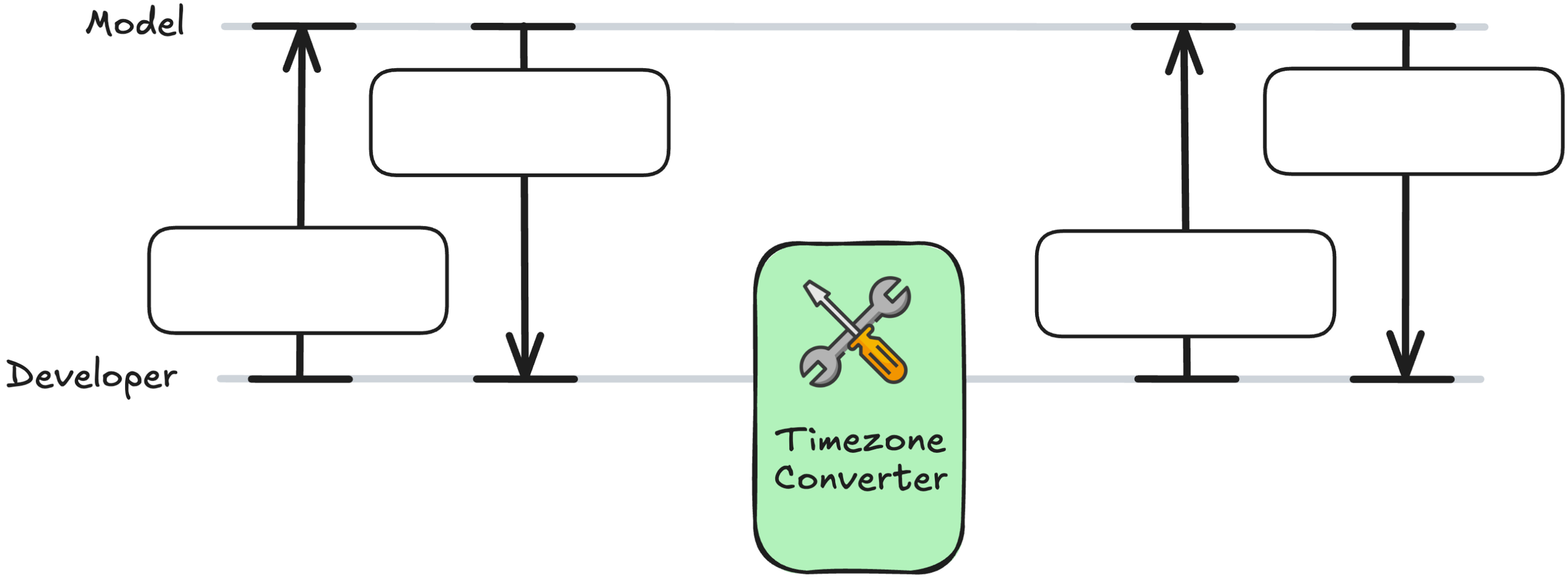
MCP Tools to LLM Tools

> **Tool ⇄ LLM integration is highly LLM-dependent**

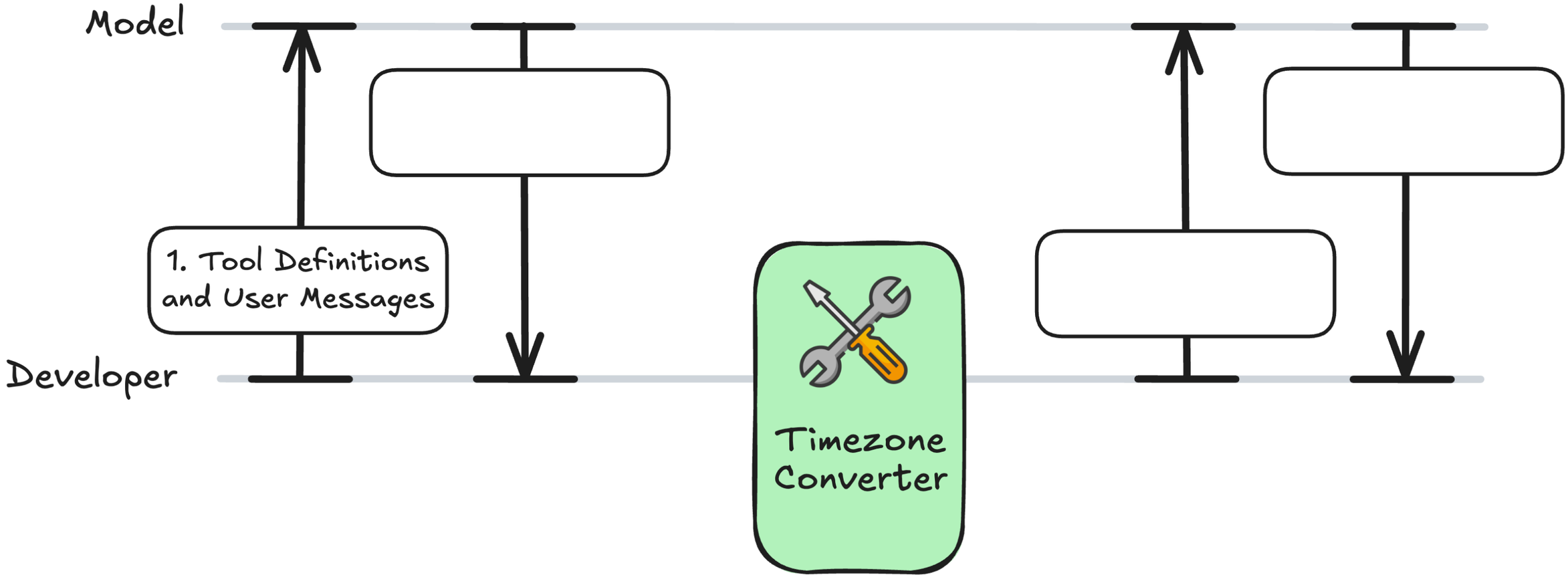
- LLMs accept tools in different ways and with different syntax
- Use **Claude** via **Anthropic's Messages API**
- **Focus:** processes and transformations



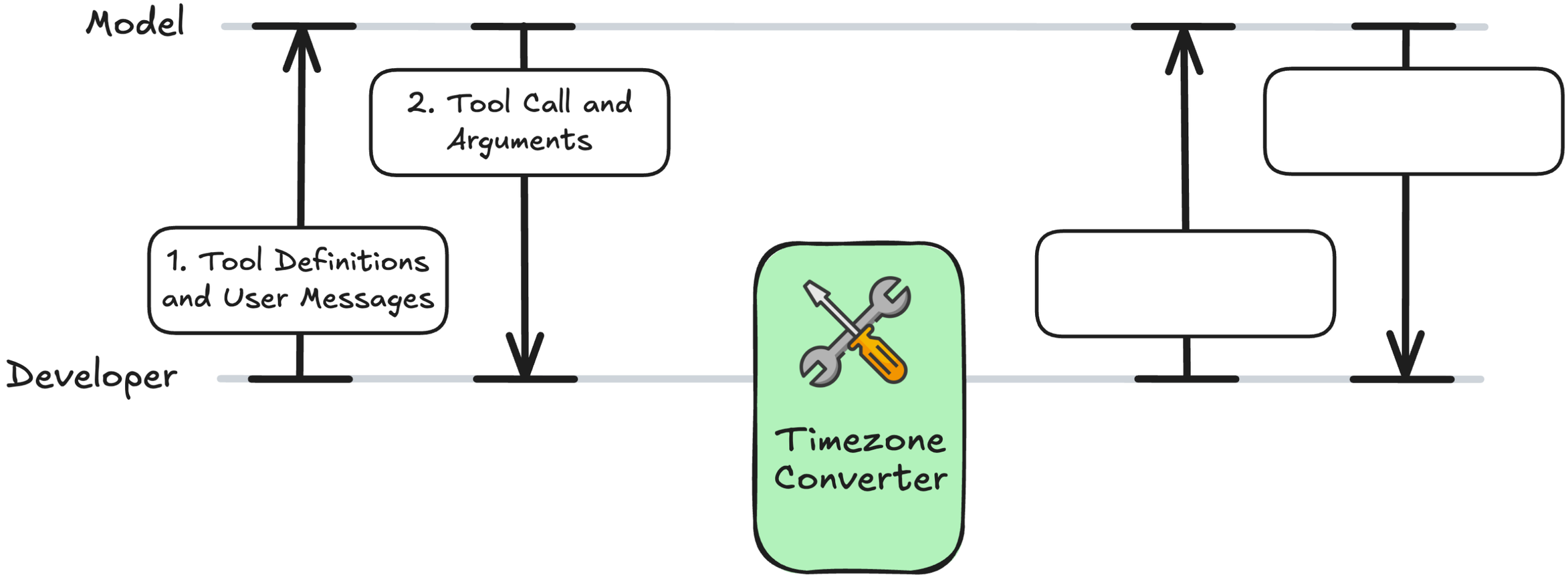
The Tool-Calling Workflow



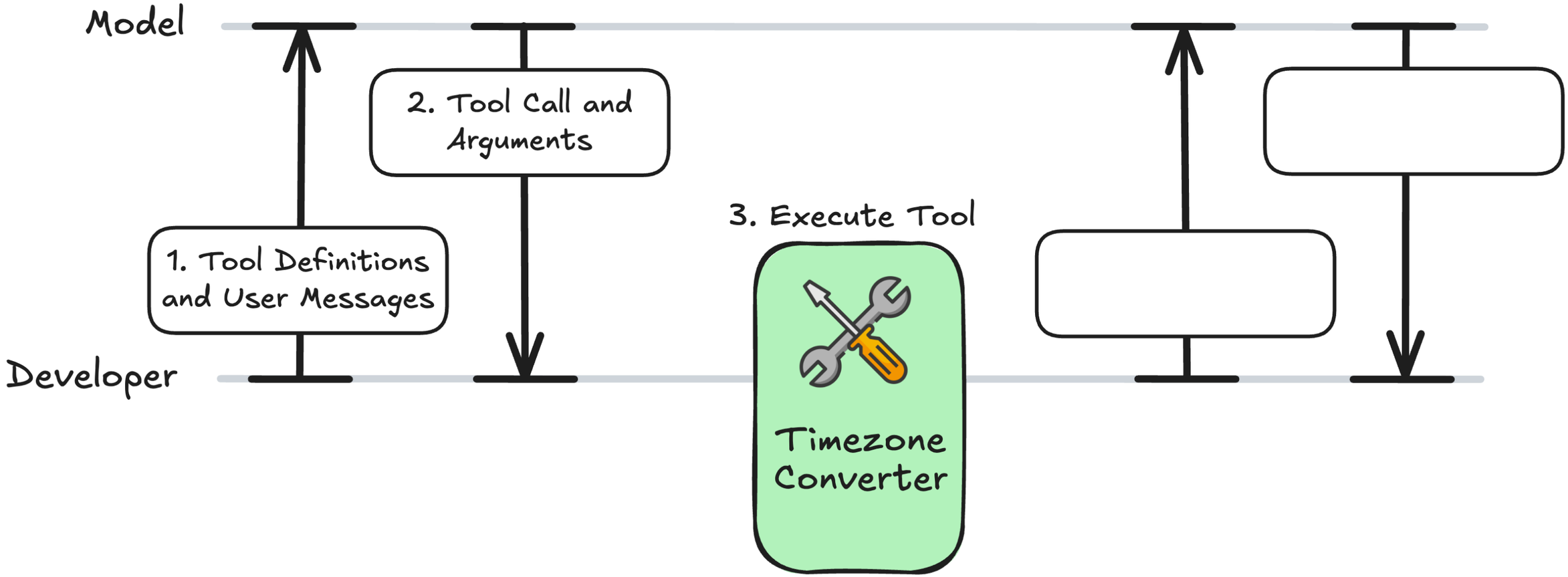
The Tool-Calling Workflow



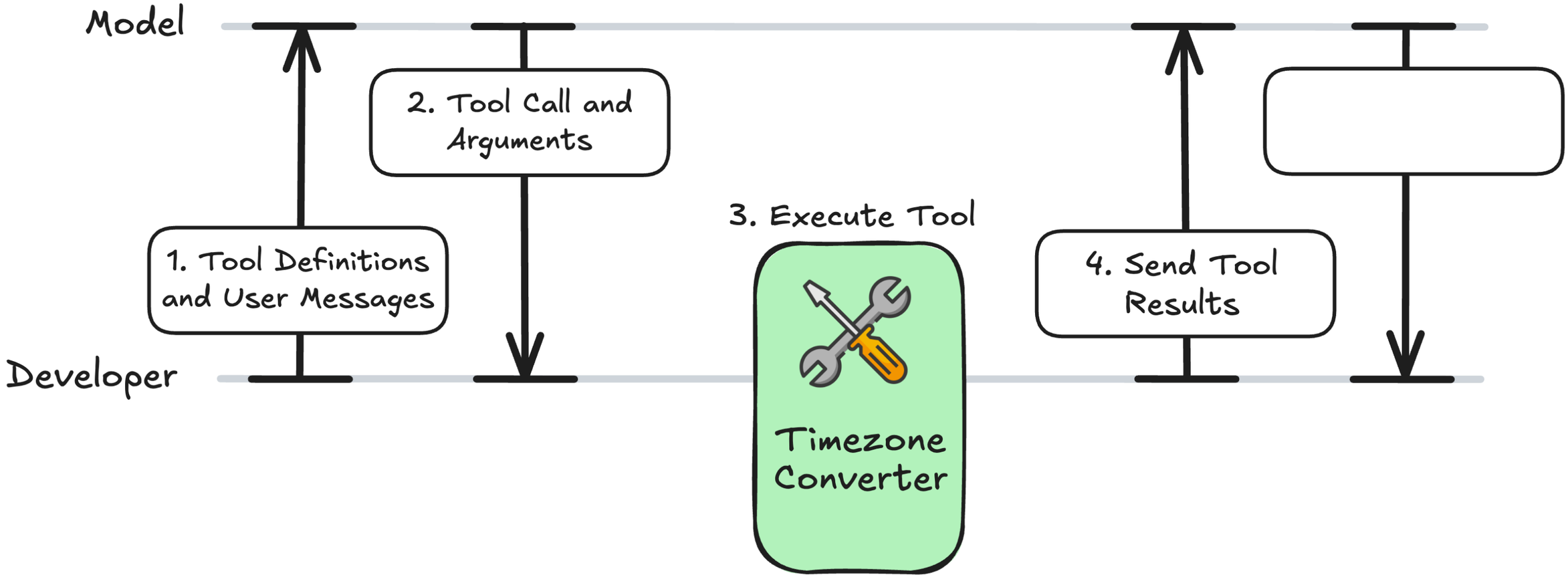
The Tool-Calling Workflow



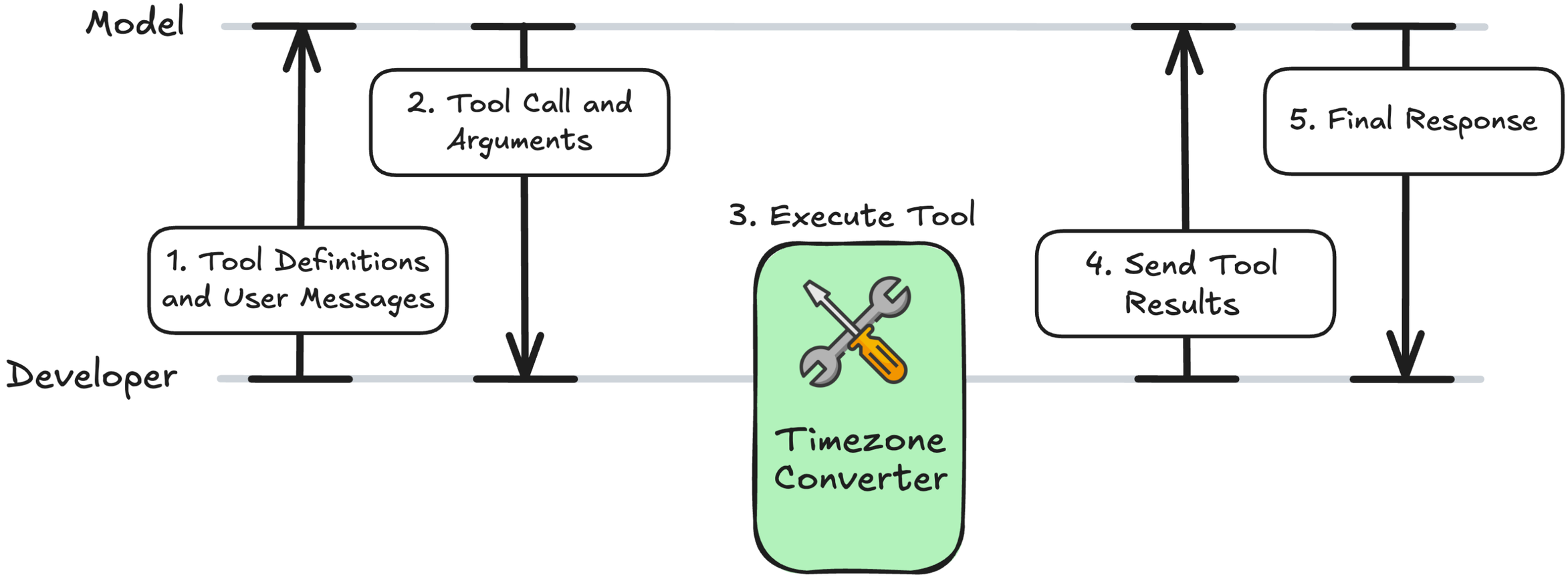
The Tool-Calling Workflow



The Tool-Calling Workflow



The Tool-Calling Workflow



MCP Server: timezone_server.py

```python
from mcp.server.fastmcp import FastMCP
import requests

mcp = FastMCP("Timezone Converter")

@mcp.tool()
def convert_timezone(date_time: str, from_timezone: str, to_timezone: str) -> str:
    ...

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Expanding the Client Code

```
async def get_tools_from_mcp():
            # ...
            return response.tools

async def call_mcp_tool(tool_name: str, arguments: dict) -> str:
            # ...
            result = await session.call_tool(tool_name, arguments)
            return str(result.content[0].text)

async def call_anthropic_llm(user_query: str):
```

Setup: Format the Tools for the LLM

```python
async def call_anthropic_llm(user_query: str):
    """Call Claude with MCP tools."""

    mcp_tools = await get_tools_from_mcp()

    anthropic_tools = []
    for tool in mcp_tools:
        anthropic_tool = {
            "name": tool.name,
            "description": tool.description or "",
            "input_schema": tool.inputSchema,  # MCP uses JSON Schema format
        }
        anthropic_tools.append(anthropic_tool)
```

1. Send the Query and Tools to the LLM

```
from anthropic import AsyncAnthropic
```

```
async def call_anthropic_llm(user_query: str):
    # ...

    client = AsyncAnthropic(api_key="<ANTHROPIC_API_TOKEN>")

    response = await client.messages.create(
        model="claude-sonnet-4-6",
        max_tokens=1024,
        messages=[{"role": "user", "content": user_query}],
        tools=anthropic_tools,
    )
```

2. Checking for a Tool Call

```python
async def call_anthropic_llm(user_query: str):
    # ...

    if response.stop_reason == "tool_use":
        tool_use = next(b for b in response.content if b.type == "tool_use")
        name = tool_use.name
        args = tool_use.input

        print(f"Model decided to call: {name}")
        print(f"Arguments: {args}\n")
```

3. Calling the Tool | 4. The Follow-Up Message

```
async def call_anthropic_llm(user_query: str):
    # ...

    if response.stop_reason == "tool_use":
        # ...

        result = await call_mcp_tool(name, args)
        tool_result = {"type": "tool_result",
            "tool_use_id": tool_use.id, "content": result}
        followup = await client.messages.create(
            model="claude-sonnet-4-6", max_tokens=1024,
            messages=[
                {"role": "user", "content": user_query},
                {"role": "assistant", "content": response.content},
                {"role": "user", "content": [tool_result]},
            ],
            tools=anthropic_tools,
        )
```

5. The Final Response

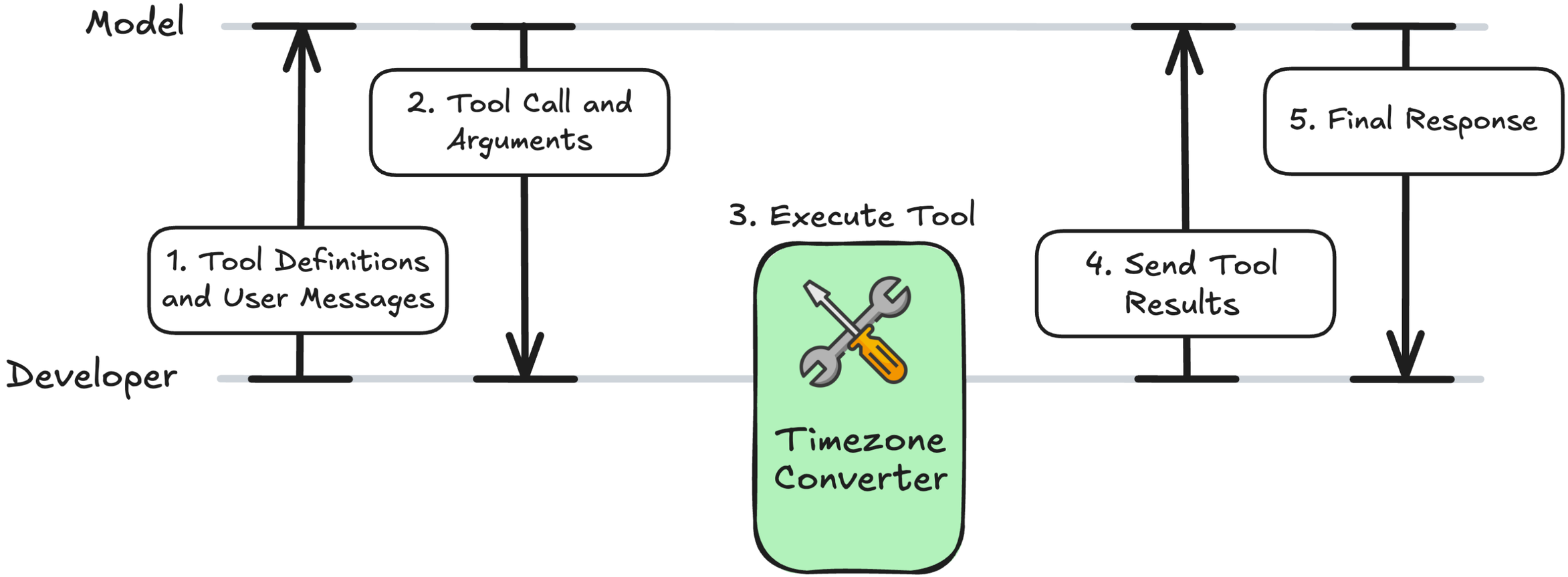
```
async def call_anthropic_llm(user_query: str):
    # ...

    if response.stop_reason == "tool_use":
        # ...
        final = next((b.text for b in followup.content if b.type == "text"), None)
        if final:
            print(f"\nAssistant: {final}")
        else:
            print("No follow-up message from model.")

    else:
        text = next((b.text for b in response.content if b.type == "text"), "")
        print(f"\nAssistant: {text}")
        return str(text)
```

Testing the Function

```
if __name__ == "__main__":
    asyncio.run(call_anthropic_llm("It is 9:50 AM in the UK in January. What time
    is it in Lisbon, Portugal?"))
```

```
Assistant: It's 9:50 AM in Lisbon as well.
```

The Tool-Calling Workflow



Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)

MCP and LLMs: Prompts and Resources

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)



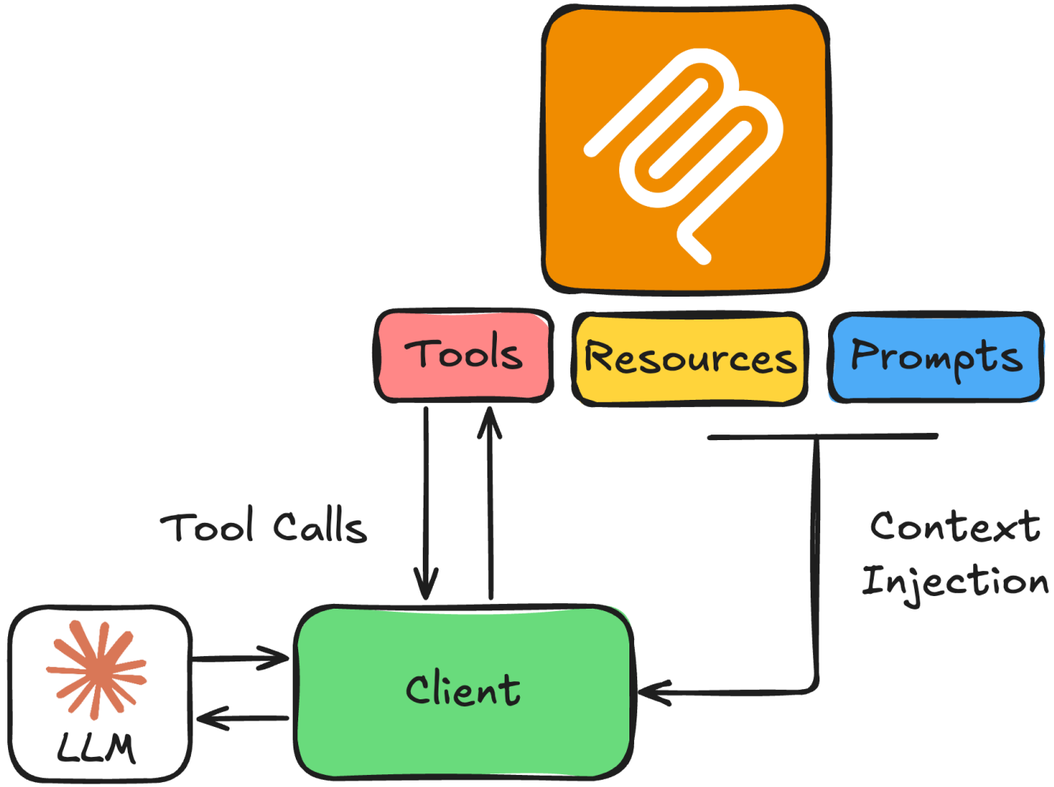
James Chapman
AI Curriculum Manager, DataCamp

datacamp

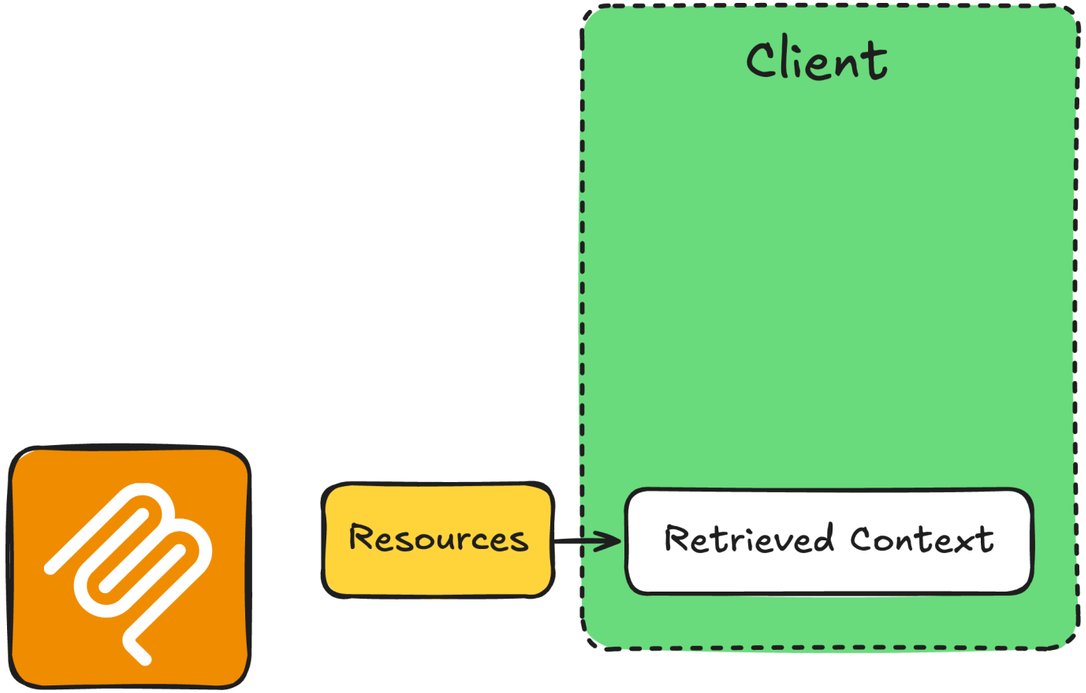
Resources and Prompts in the LLM Flow

- **Resources:** read-only context
- **Prompts:** instruction templates to set the behavior and optimize the model for the task

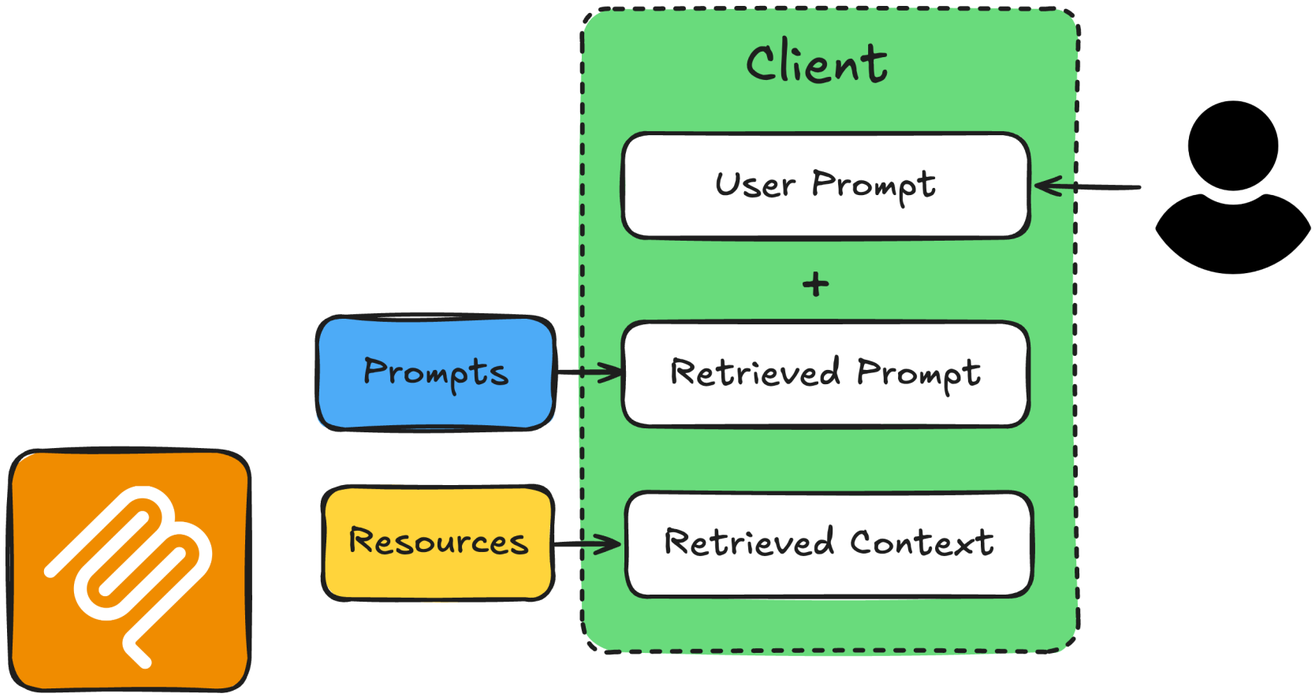
→ Continue with the **Anthropic Messages API**



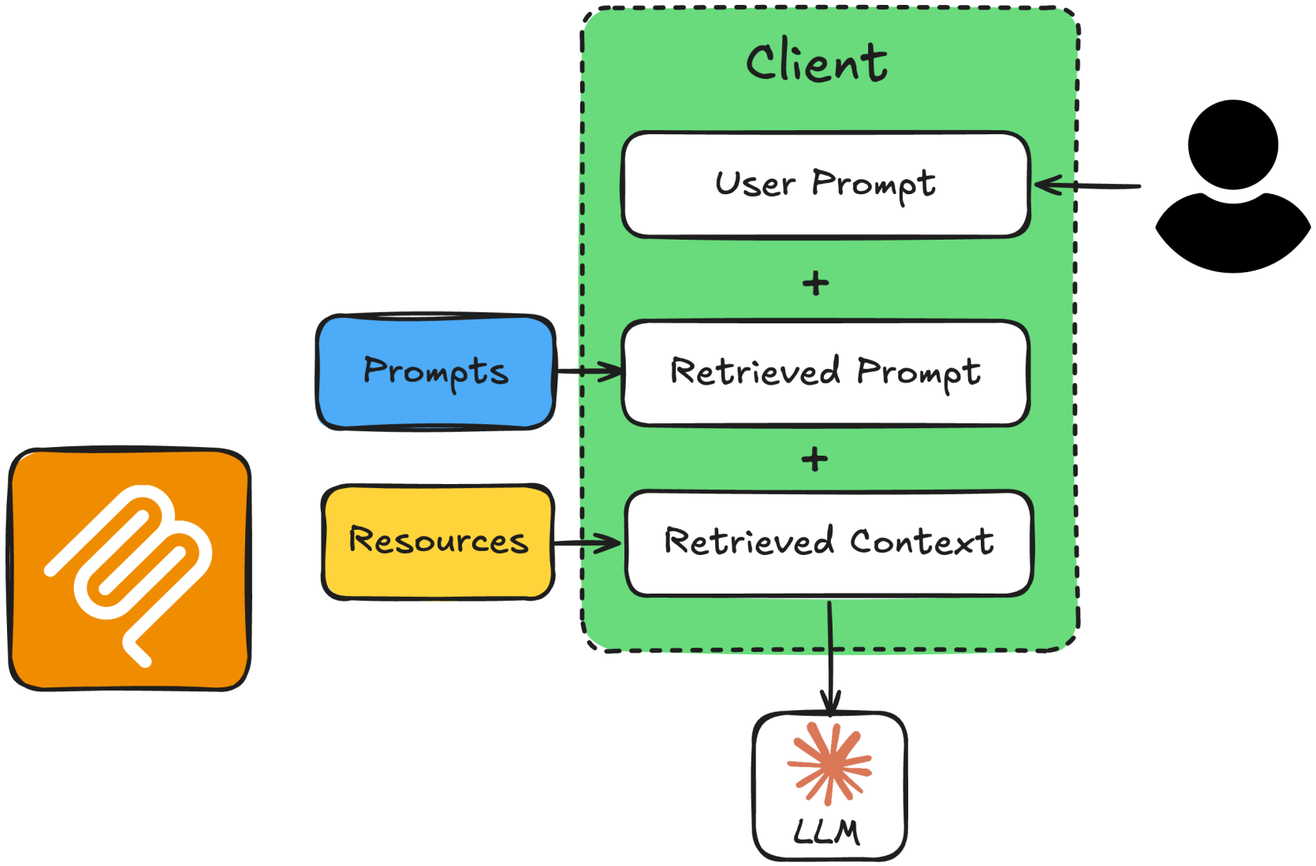
The Prompt-Resource Workflow



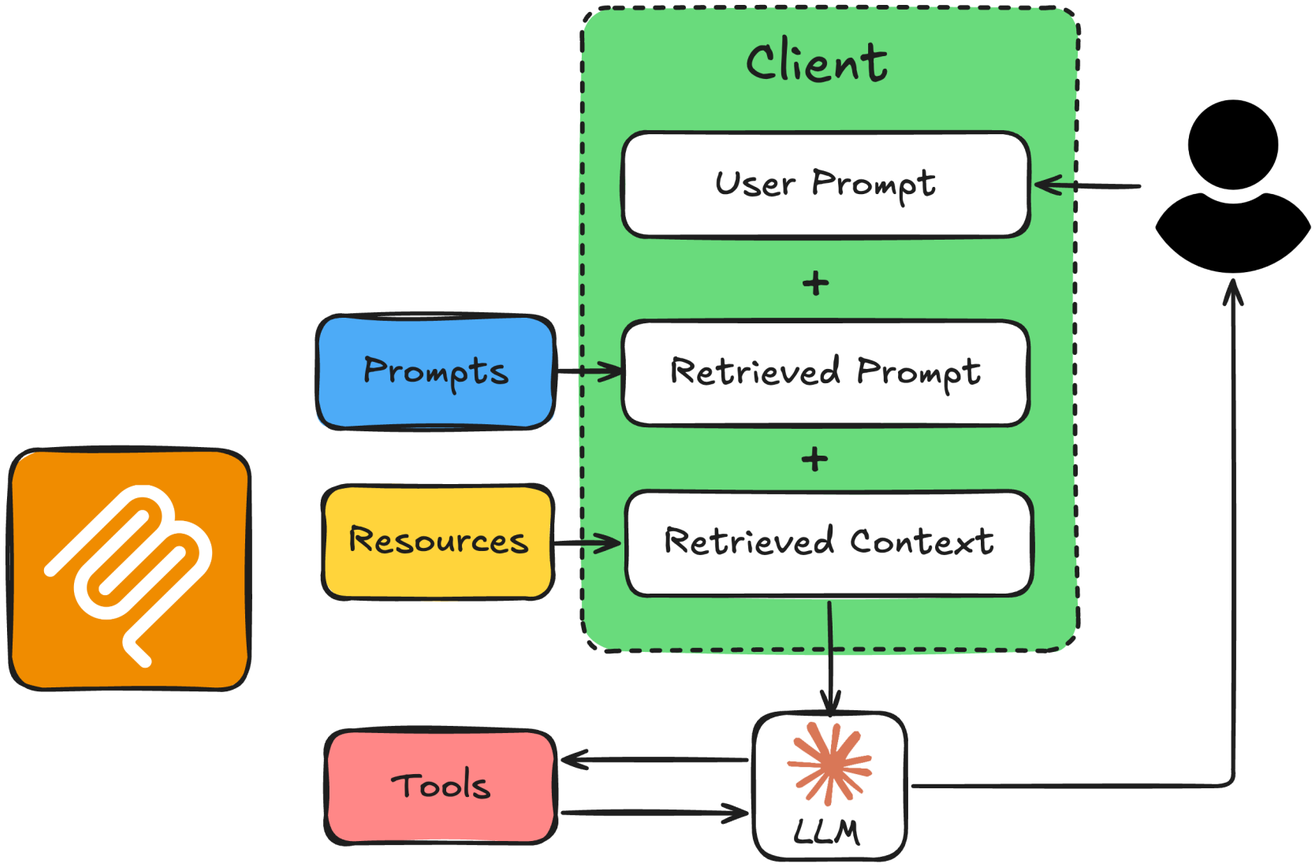
The Prompt-Resource Workflow



The Prompt-Resource Workflow



The Prompt-Resource Workflow



```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Timezone Converter")

@mcp.tool()
def convert_timezone(date_time: str, from_timezone: str, to_timezone: str) -> str:
    # ...

@mcp.resource("file://locations.txt")
def get_locations() -> str:
    # ...

@mcp.prompt(title="Timezone Conversion")
def convert_timezone_prompt(timezone_request: str) -> str:
    # ...

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Client Helper Functions

- `read_resource(resource_uri)` : fetches a resource's contents by URI
- `read_prompt(prompt_name, user_input)` : fetches the prompt template with the user's request injected

1. Fetch Resource and Prompt

```
async def get_context_from_mcp(user_query: str) -> tuple[str, str]:
    """Fetch resource content and prompt text from the MCP server."""
    params = StdioServerParameters(command=sys.executable, args=["timezone_server.py"])

    async with stdio_client(params) as (reader, writer):
        async with ClientSession(reader, writer) as session:
            await session.initialize()
            # Get the resource (supported locations)
            resource_result = await session.read_resource("file://locations.txt")
            resource_text = resource_result.contents[0].text
            # Get the prompt with the user's query
            prompt_result = await session.get_prompt("convert_timezone_prompt",
                                                     arguments={"timezone_request": user_query})
            prompt_text = prompt_result.messages[0].content.text
            return resource_text, prompt_text
```

2. Build the System Message and Call the LLM

```python
async def call_llm_with_context(user_query: str):
    """Call the LLM with resource and prompt context from MCP."""
    resource_text, prompt_text = await get_context_from_mcp(user_query)
    # Combine prompt (task + rules + user request) with resource (supported locations)
    full_prompt = prompt_text + "\n\nSupported locations:\n" + resource_text
    client = AsyncAnthropic(api_key="<ANTHROPIC_API_TOKEN>")
    response = await client.messages.create(
        model="claude-sonnet-4-6",
        max_tokens=1024,
        messages=[{"role": "user", "content": full_prompt}],
        tools=anthropic_tools,  # from get_tools_from_mcp(), formatted for Anthropic
    )
```

3. Handling the Response

```
    if response.stop_reason == "end_turn":
        text = next((b.text for b in response.content if b.type == "text"), "")
        print(f"\nAssistant: {text}")
        return str(text)
    if response.stop_reason == "tool_use":
        tool_use = next(b for b in response.content if b.type == "tool_use")
        result = await call_mcp_tool(
            tool_use.name, tool_use.input)
        # ... send the tool_result back in a follow-up
        #     request, then print the reply, as before
```

Example: Ambiguous Request

```
if __name__ == "__main__":
    asyncio.run(call_llm_with_context("What time is it in Canada?"))
```

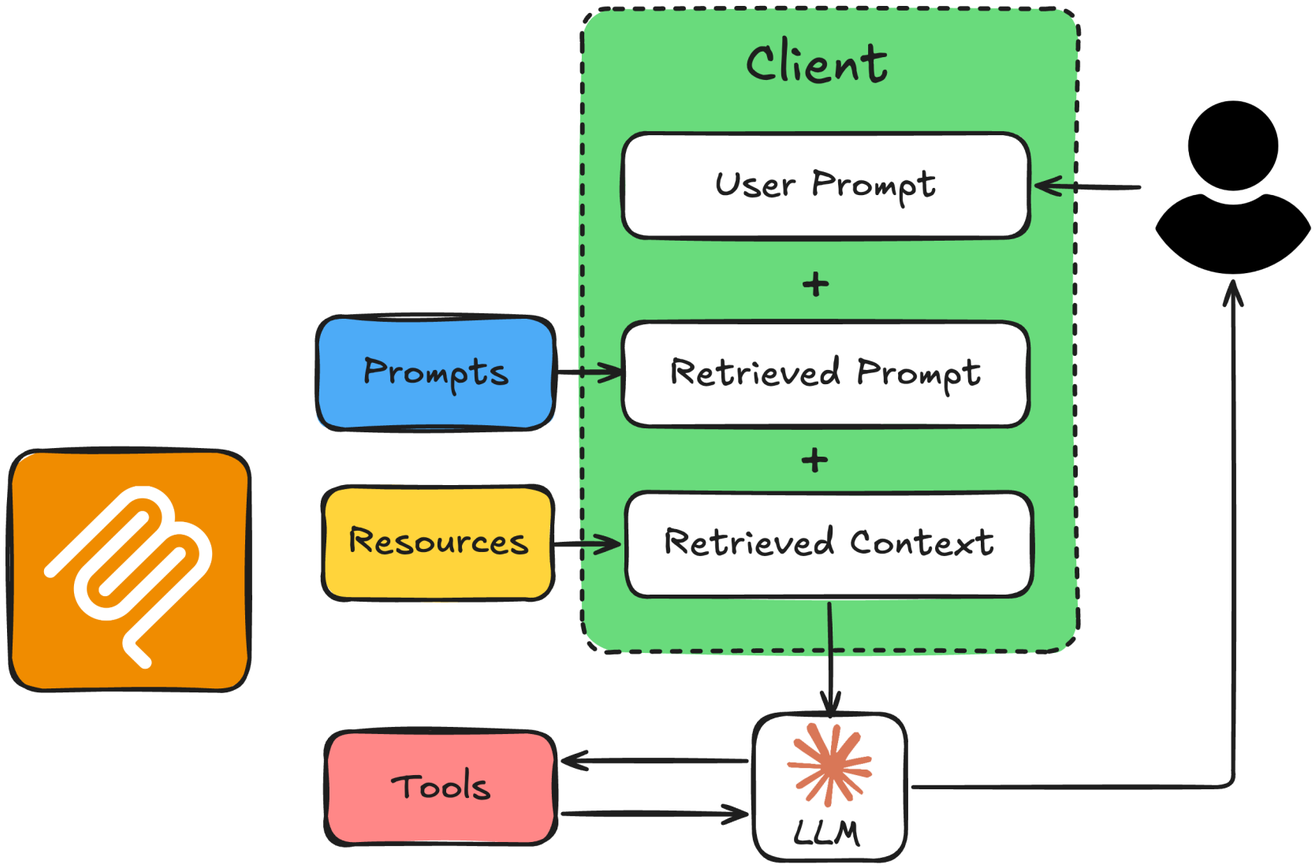
```
Assistant: Canada has several time zones. Which city or region do you mean?
For example, Toronto, Vancouver, or Halifax?
```

Example: Clear Request

```
if __name__ == "__main__":
    asyncio.run(call_llm_with_context("It is 9:50 AM in the UK in January. What
        time is it in Lisbon, Portugal?"))
```

```
Assistant: It's 9:50 AM in Lisbon as well.
```

Recap: Resources and Prompts with the LLM



Let's practice!

INTRODUCTION TO MODEL CONTEXT PROTOCOL (MCP)