Azercell

Practical Gen-AI and RAG system

DataMinds Bootcamp 2025

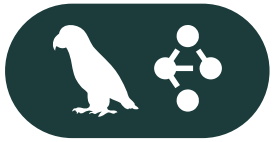
2025-08-18

Agenda

1. **LLM Overview**
   - **Claude Model Overview**
   - **From User Query to AI Response**
     - Request flow: Client → Server → API → Model Processing → Response
     - Key elements: system prompts, temperature, streaming responses
     - Tool use & tool function workflows
2. **Retrieval-Augmented Generation (RAG)**
   - **What is RAG?**
     - Workflow: Retrieval → Augmentation → Generation
     - When and why to use RAG
   - **Pros & Cons**
   - **Agentic RAG vs. Traditional RAG**
3. **Hands-On Demos**
   - **Chat Examples**
   - **Multi-Turn Conversations**
   - **Tool Use Flow**

Claude

Amazon Bedrock



LangGraph

Azercell

Claude Model Overview

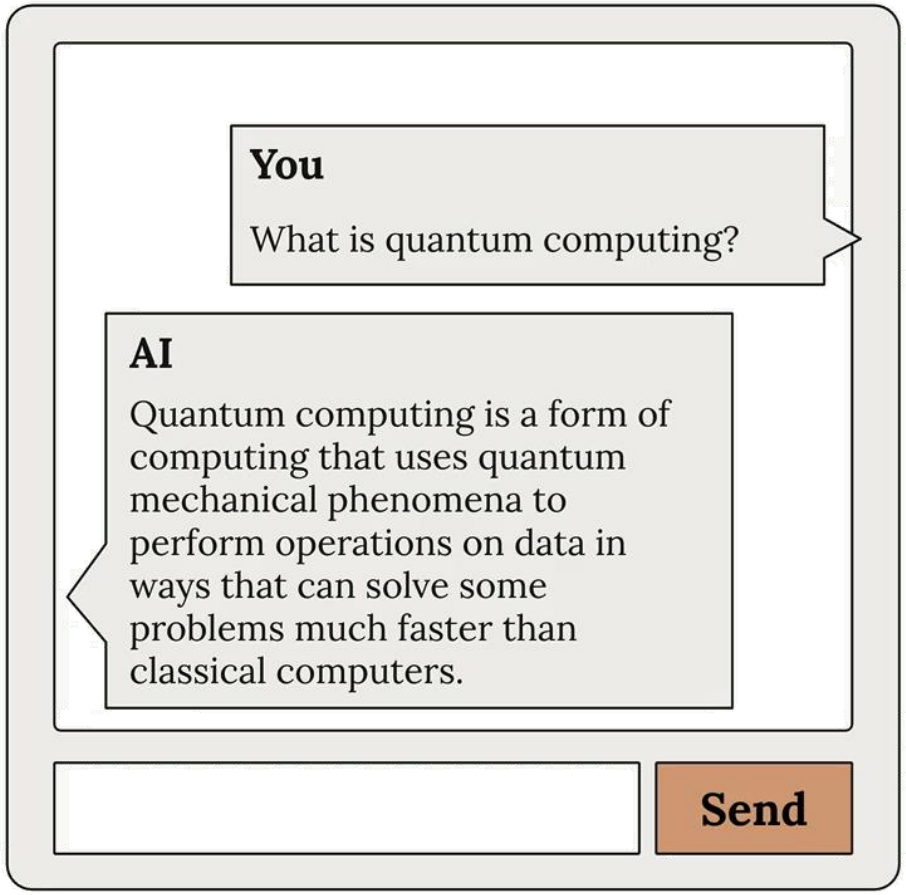
	Claude Opus	Claude Sonnet	Claude Haiku
Description	Highest level of intelligence	Intelligent model that balances quality, speed, cost	Most cost-efficient and latency-optimized model
Cost	High	Medium	Low
Comparative latency	Moderate	Fast	Fastest
Supports reasoning	Yes	Yes	No
Best used for	• Advanced software development, especially large-scale architecting • Long running tasks that require sustained focus • Strategic planning with multi-step problem solving • Tasks that could benefit from advanced reasoning	• Common coding tasks • Document creation and editing • Content marketing and copywriting • Data analysis and visualization projects • Image analysis • Process automation	• Quick code completions and suggestions • Content moderation and filtering • Data extraction and categorization • Language translation • Q&A systems and knowledge retrieval • Most high-volume, straightforward text processing tasks

Azercell

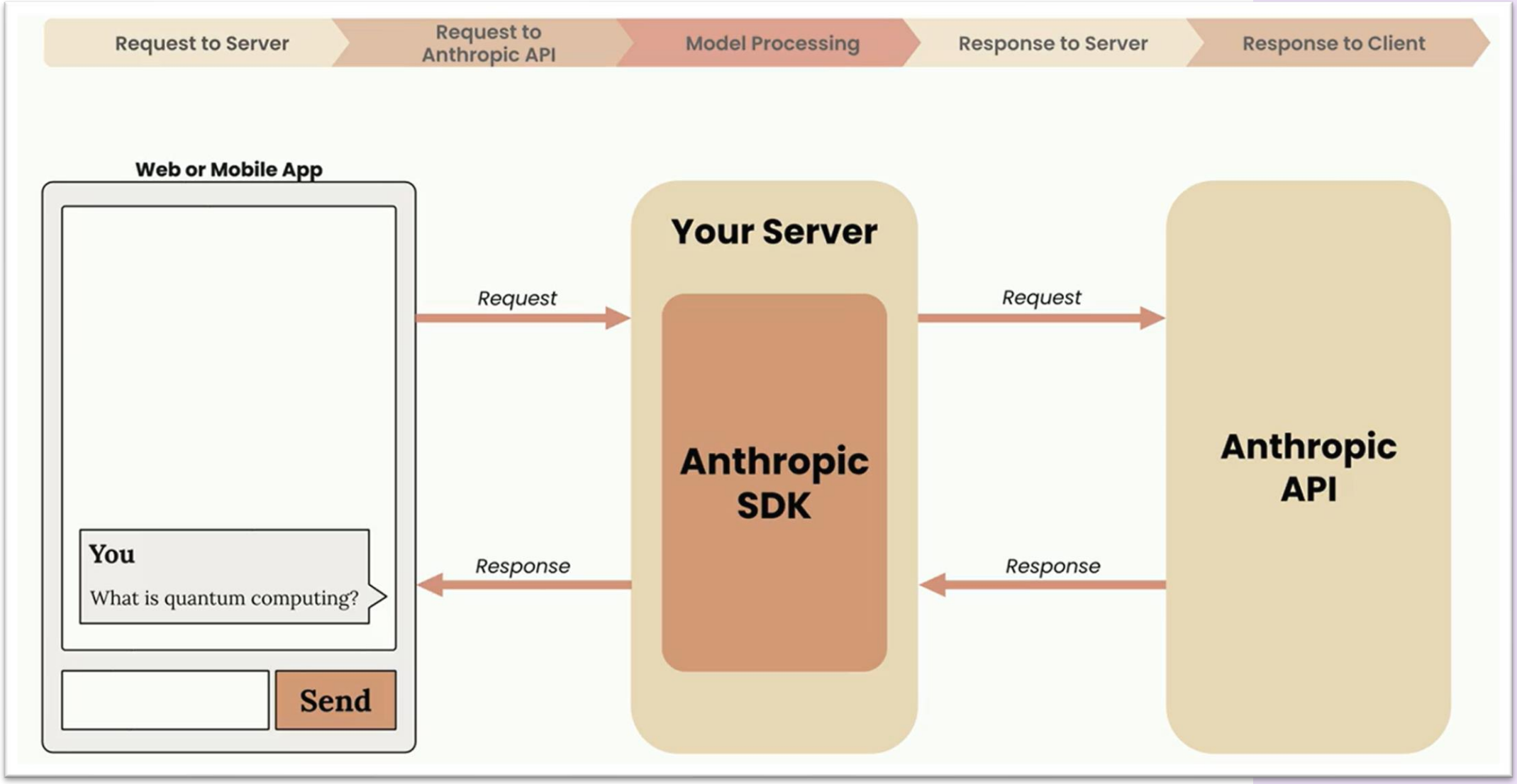
Chat Example

Virtual Machines: Each app runs with its own Guest OS, along with binaries and libraries. A Hypervisor manages multiple full OS instances on the same host.

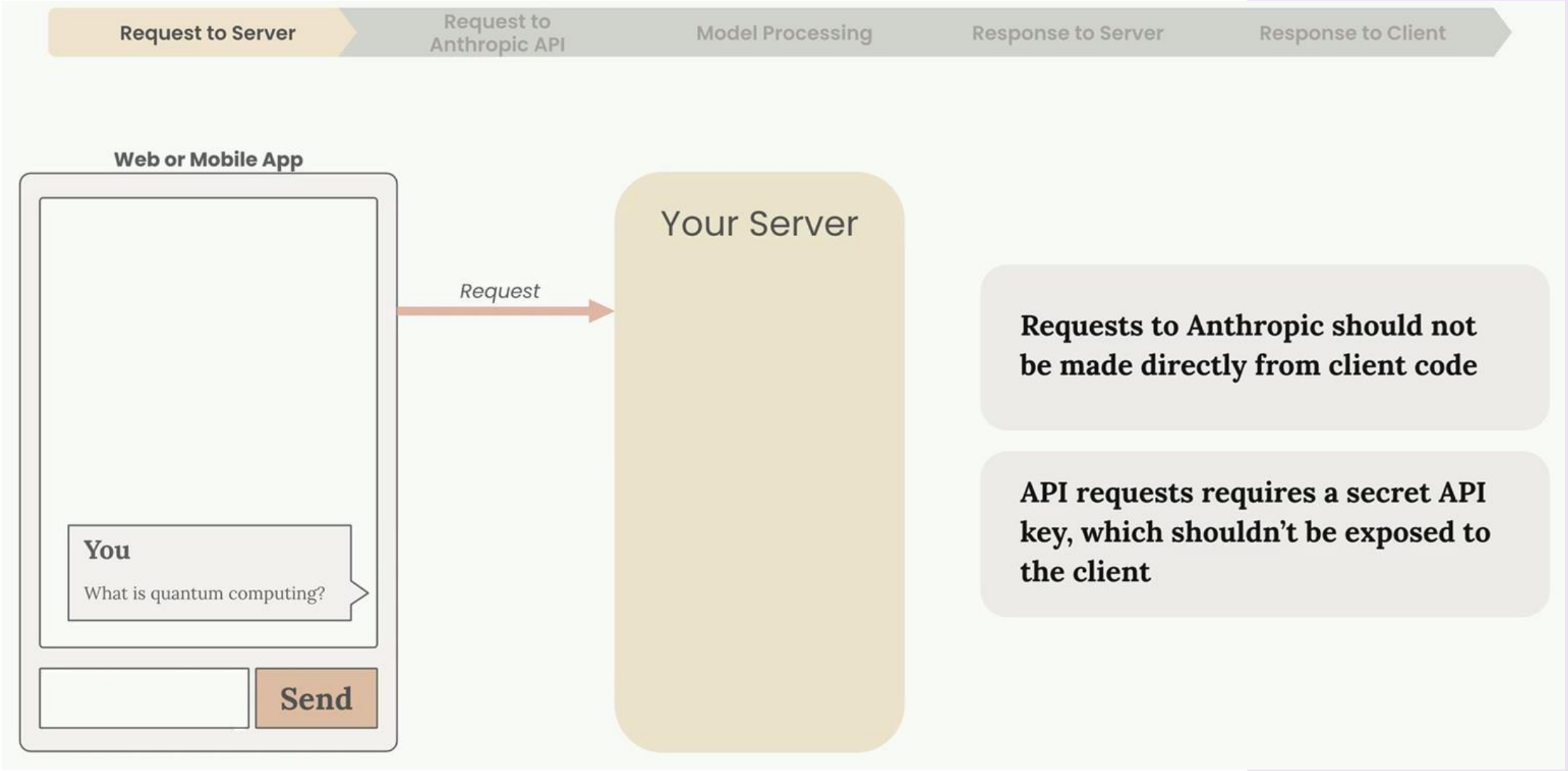
Containers: Apps share the same OS kernel, isolating only the necessary libraries and binaries.



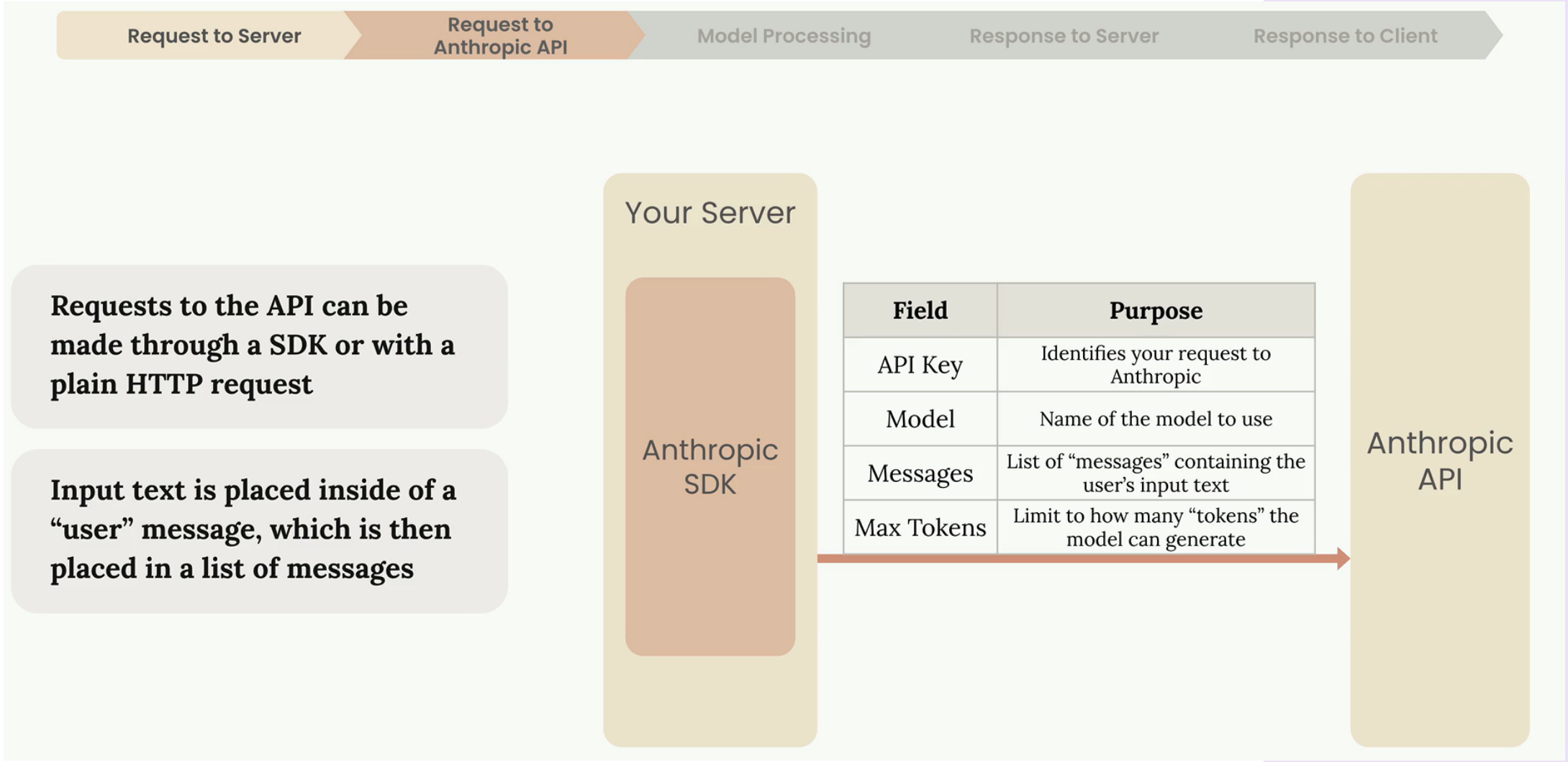
From User Query to AI Response



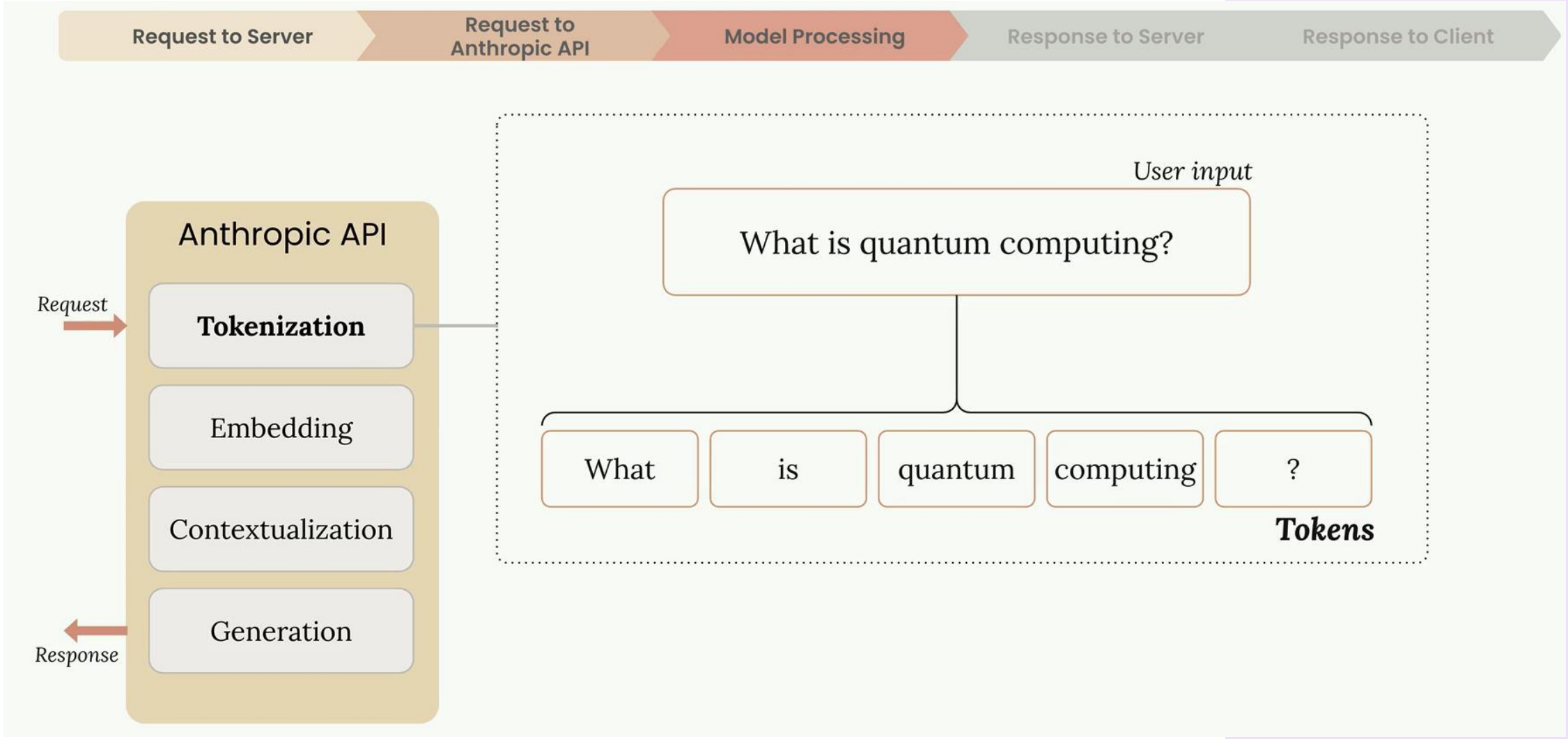
Request to Server



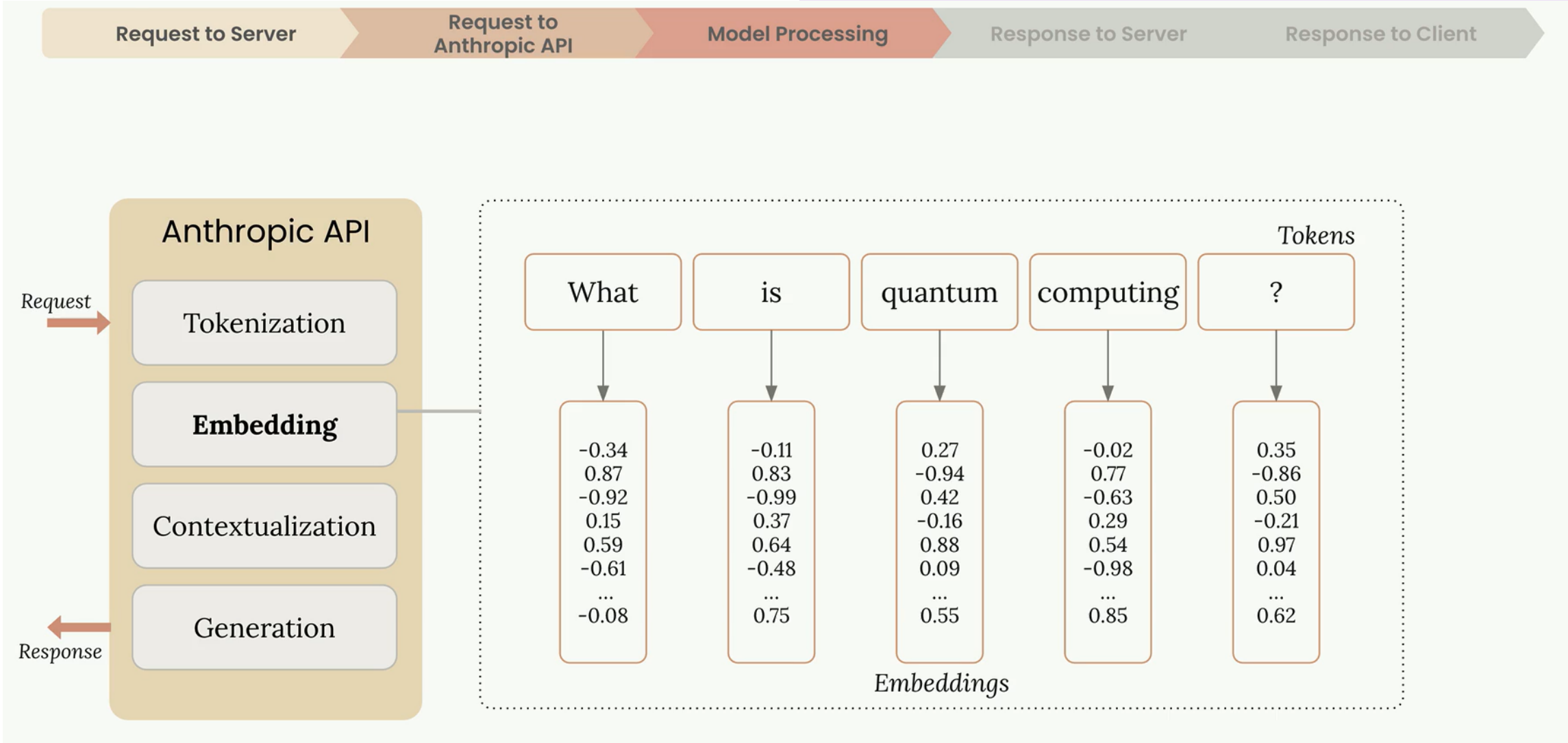
Request to API



Model Processing

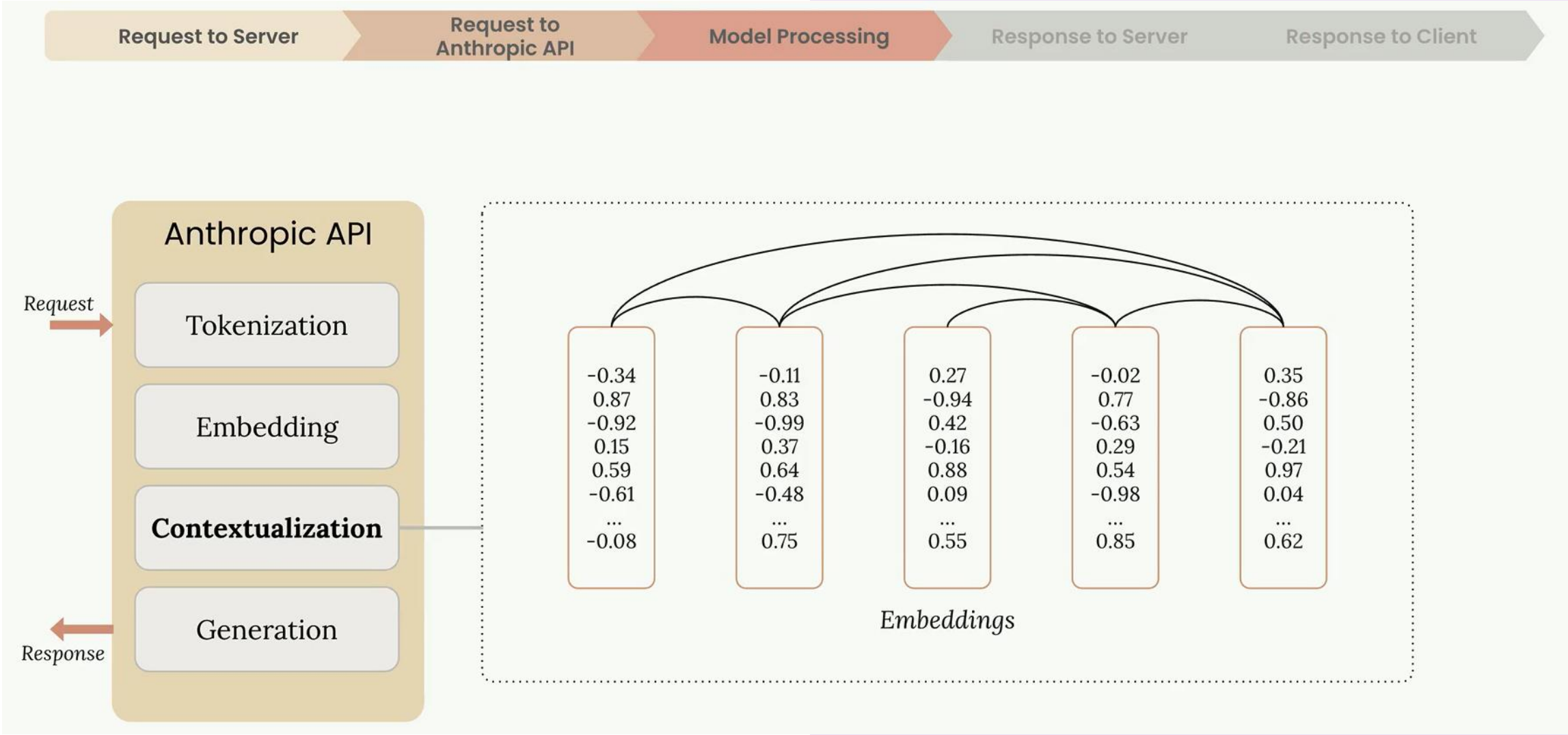


Model Processing

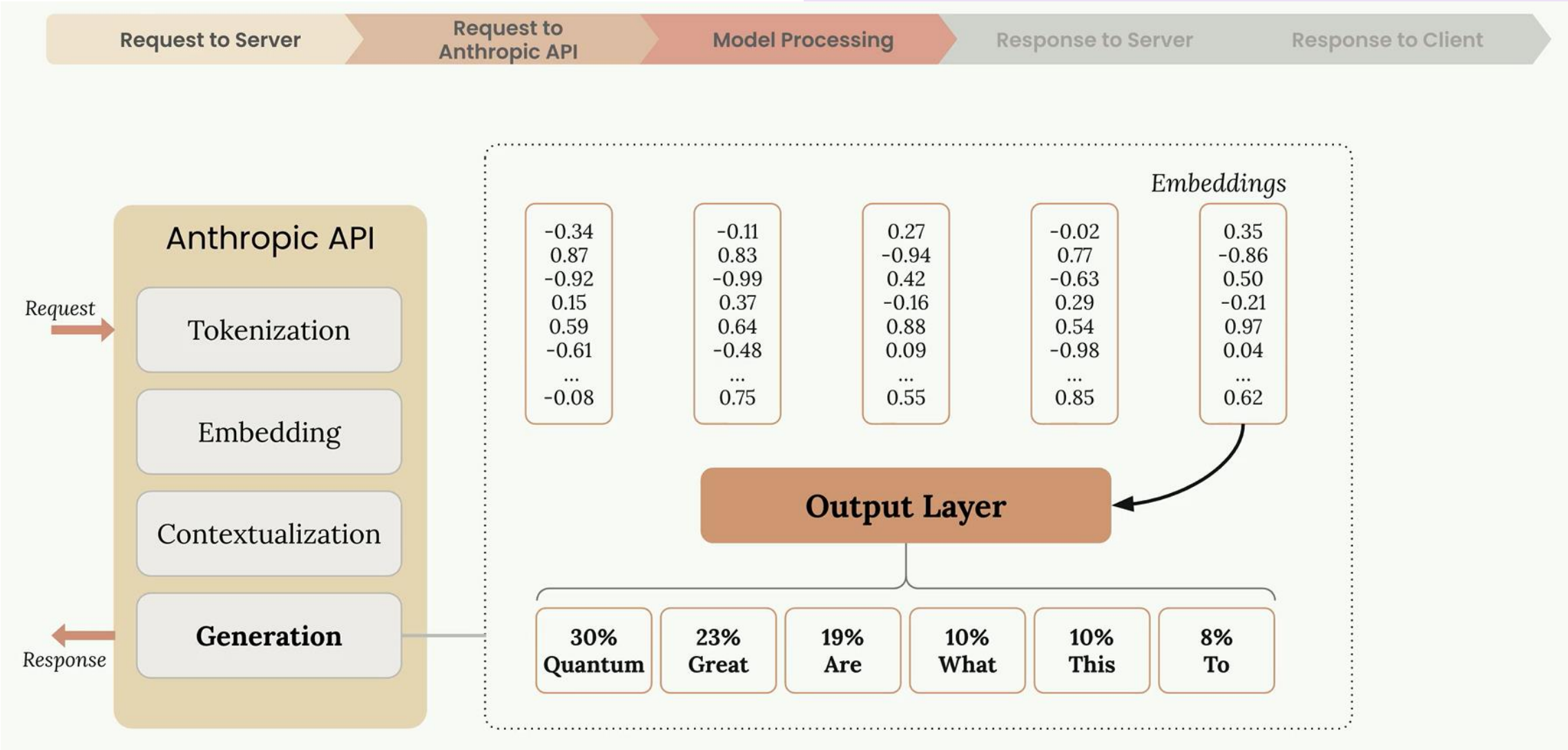


Azercell

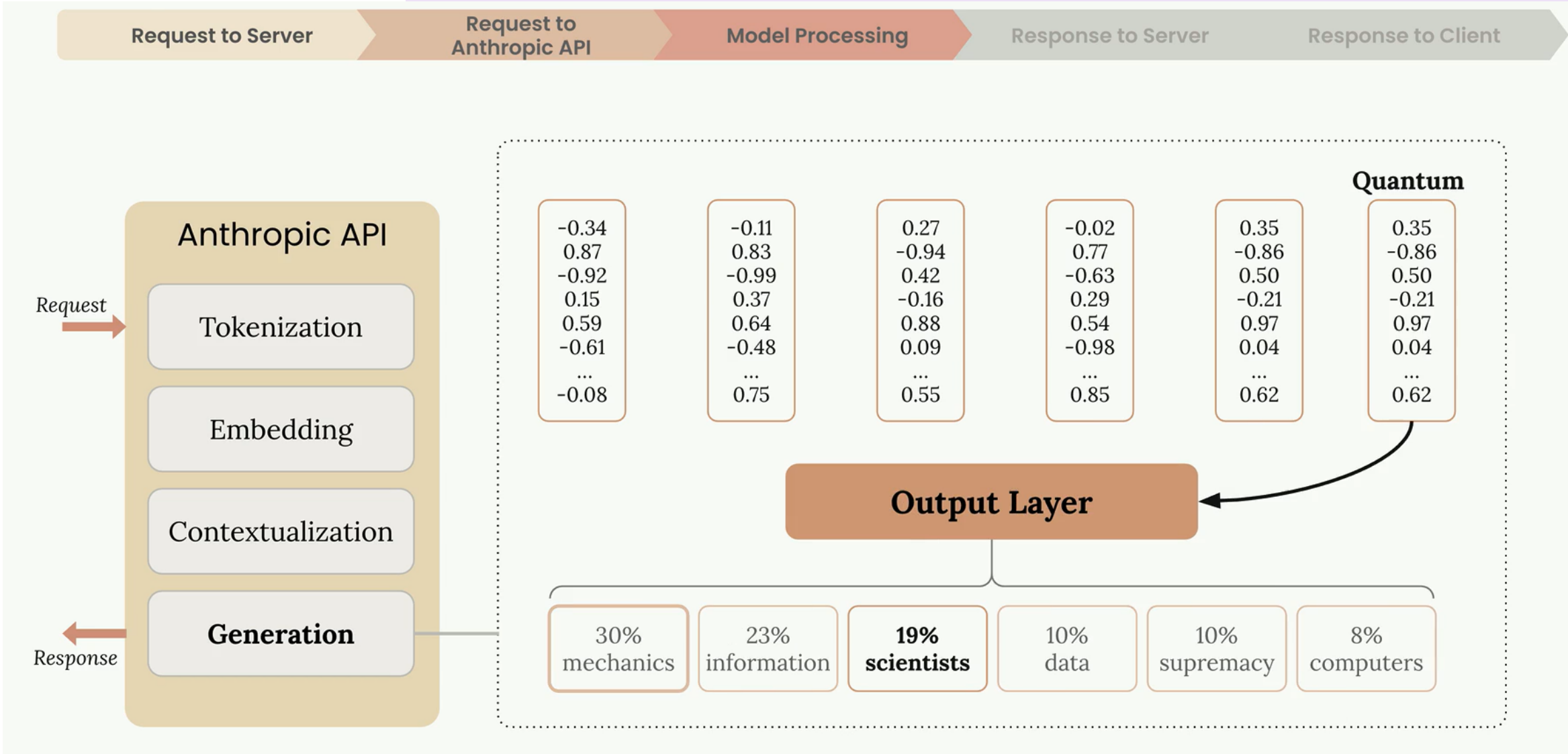
Model Processing



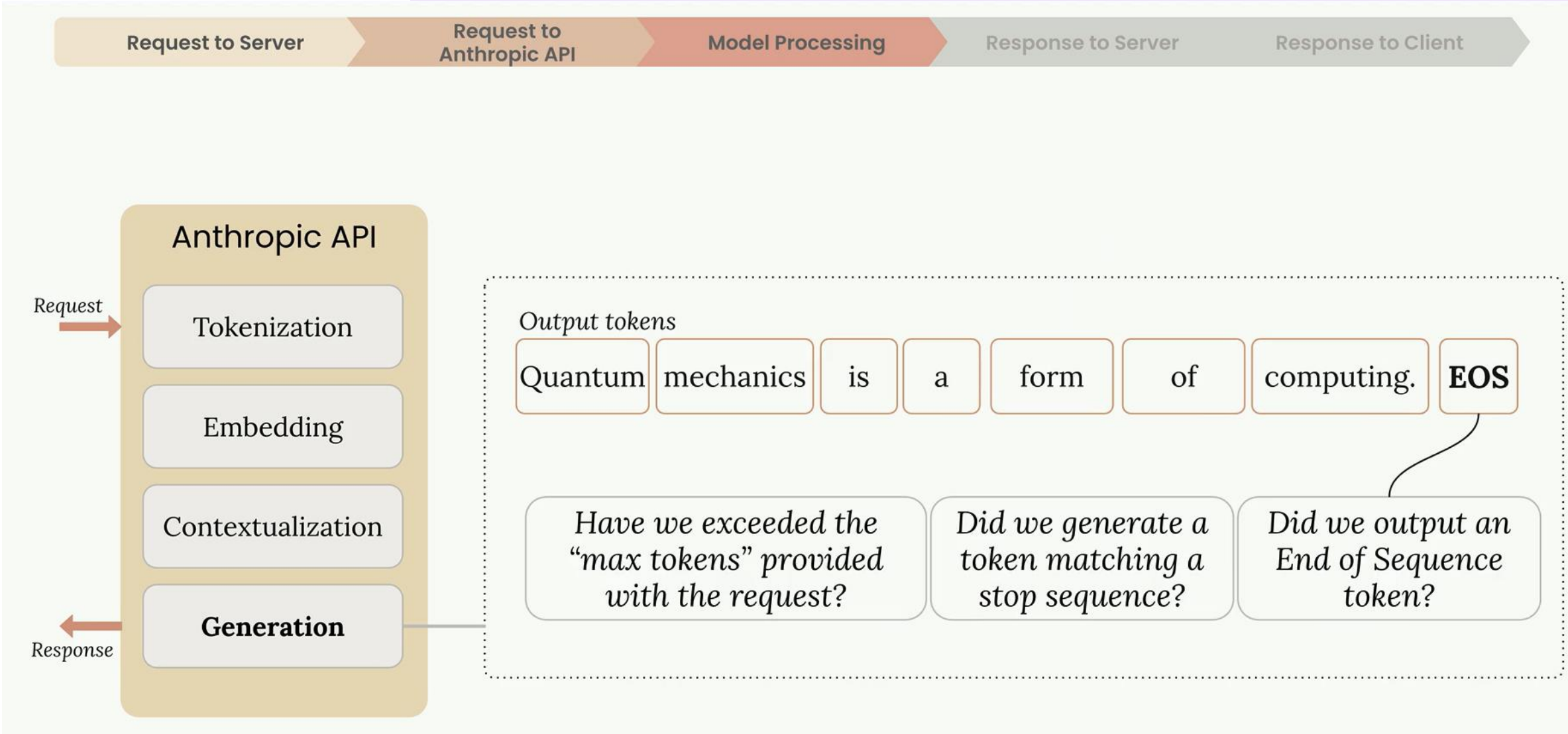
Model Processing



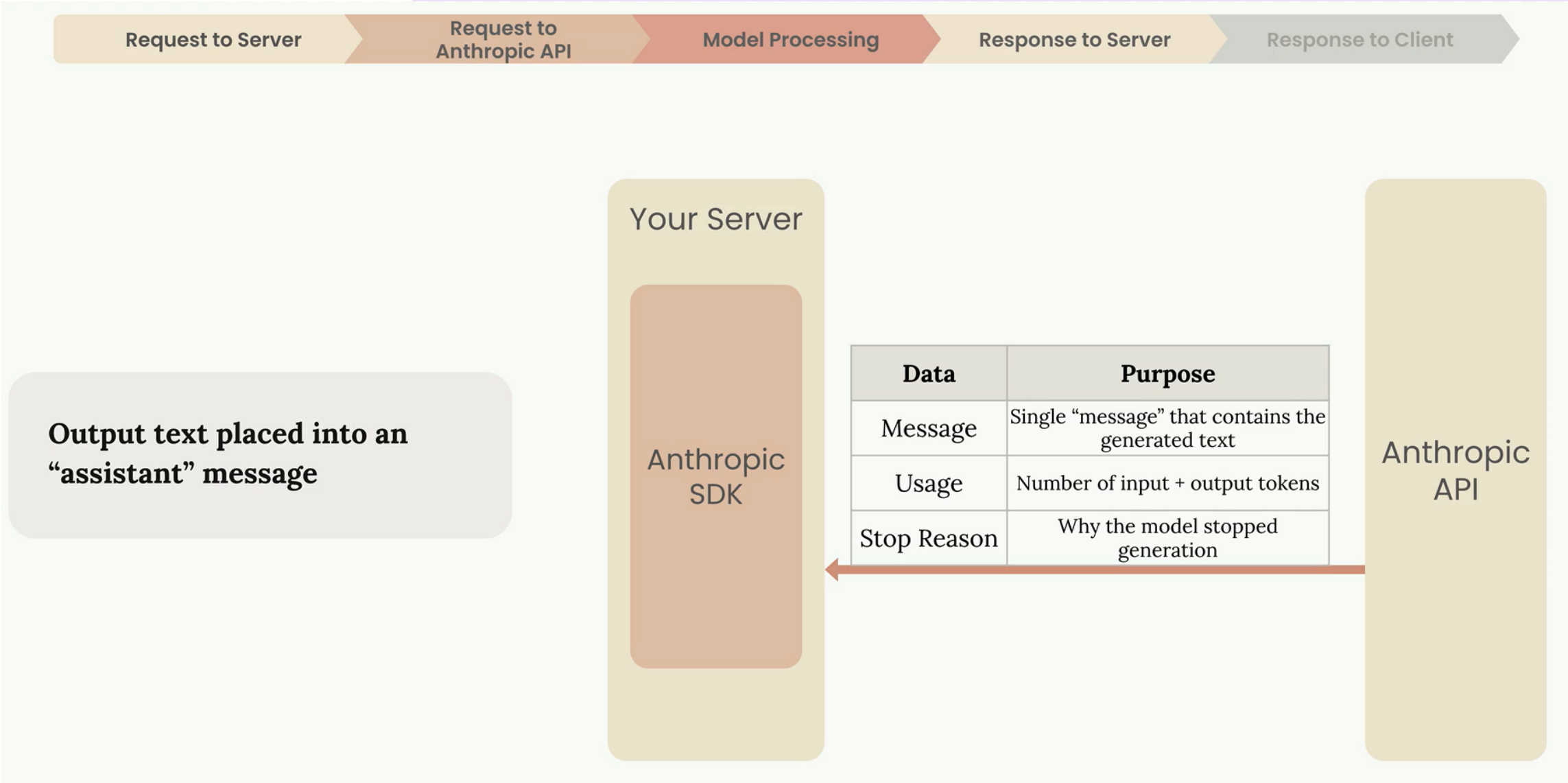
Model Processing



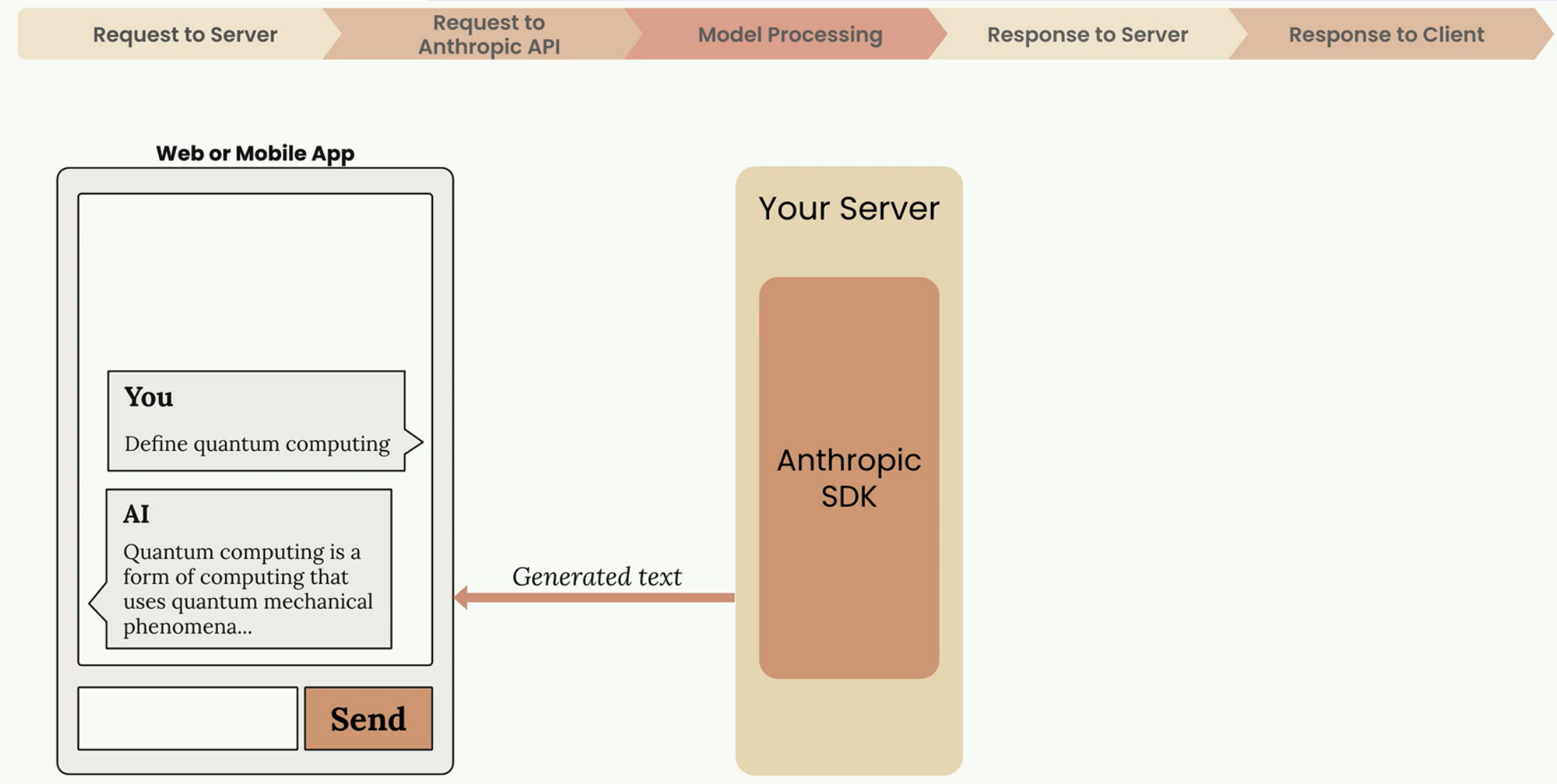
Model Processing



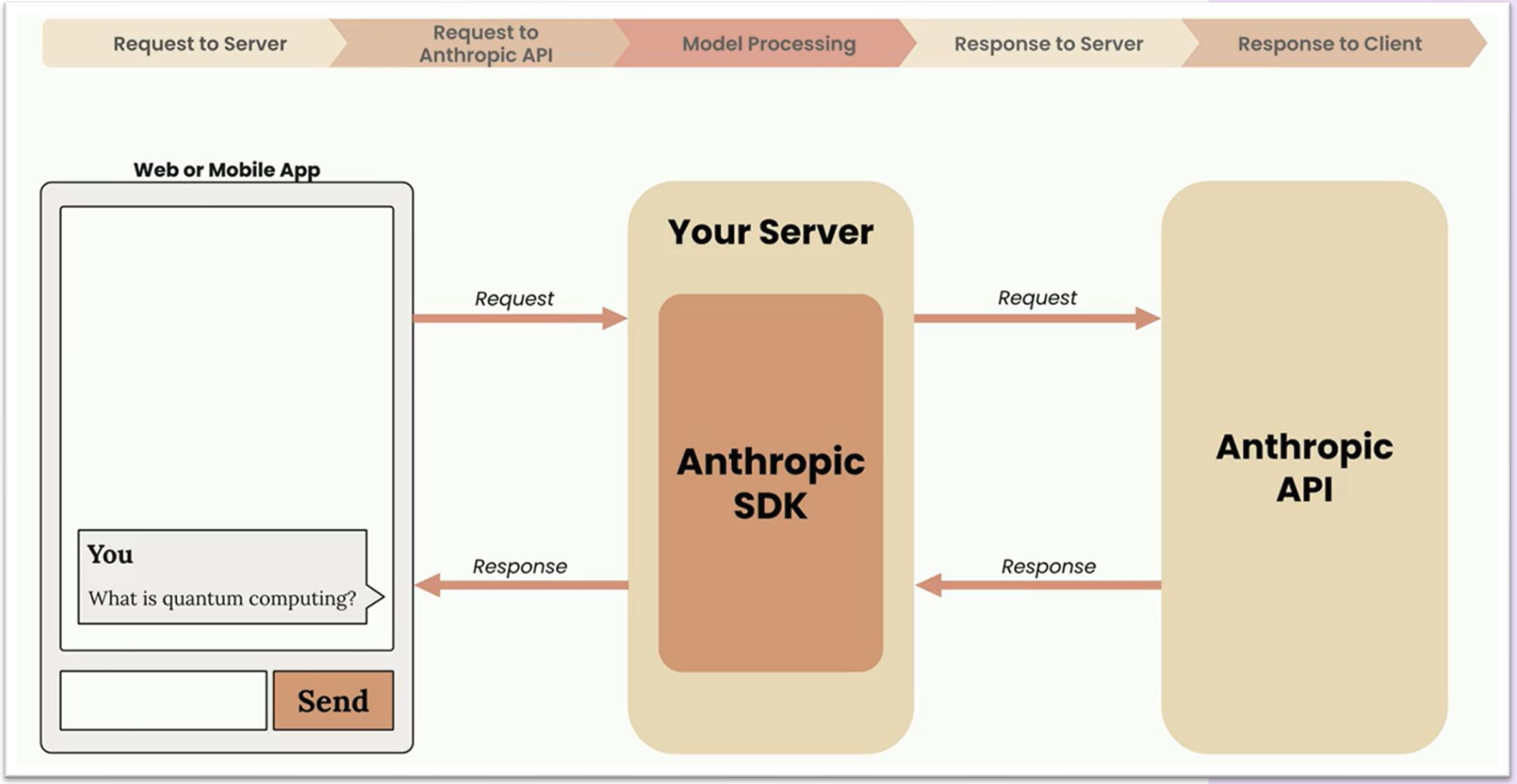
Response to Server



Response to Client

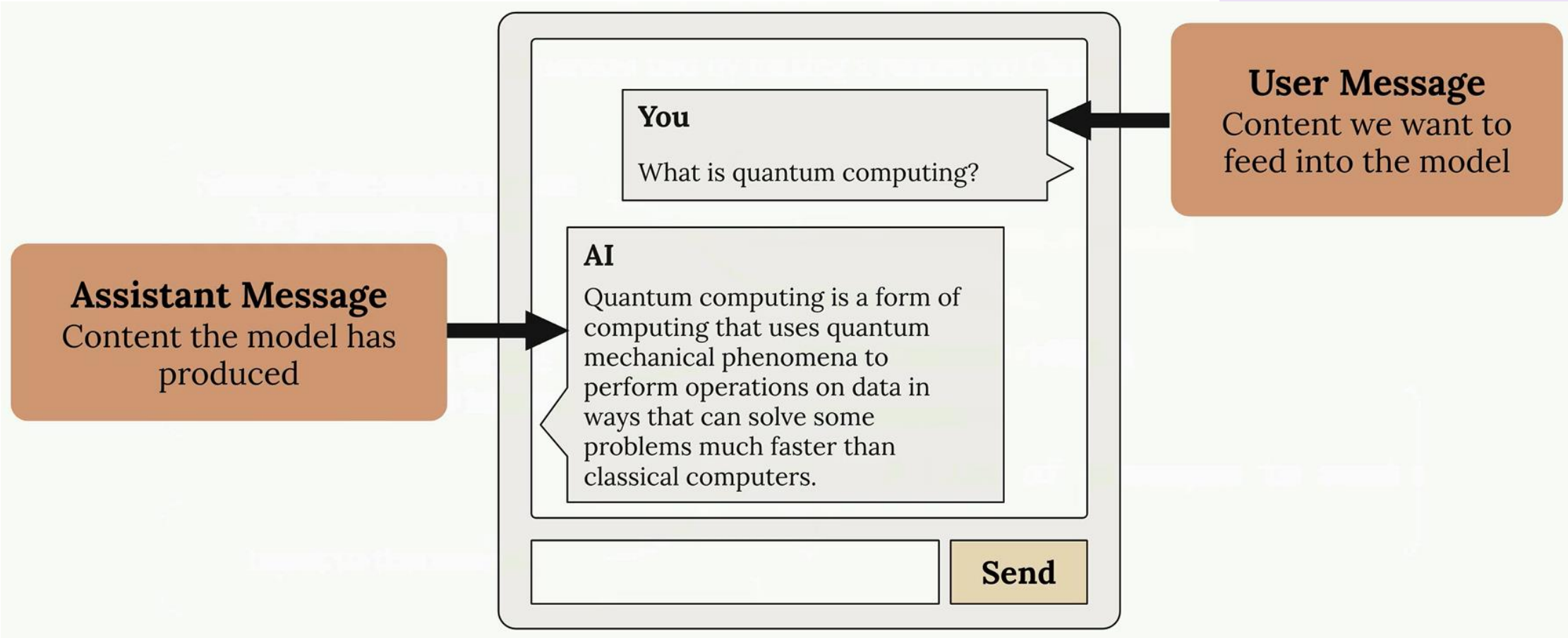


From User Query to AI Response



Model Inference

Message Types



Making Request

Making Request

[12]:
```
response = client.invoke_model(
            modelId=model,
            contentType="application/json",
            accept="application/json",
            body=body_json
        )
response
```

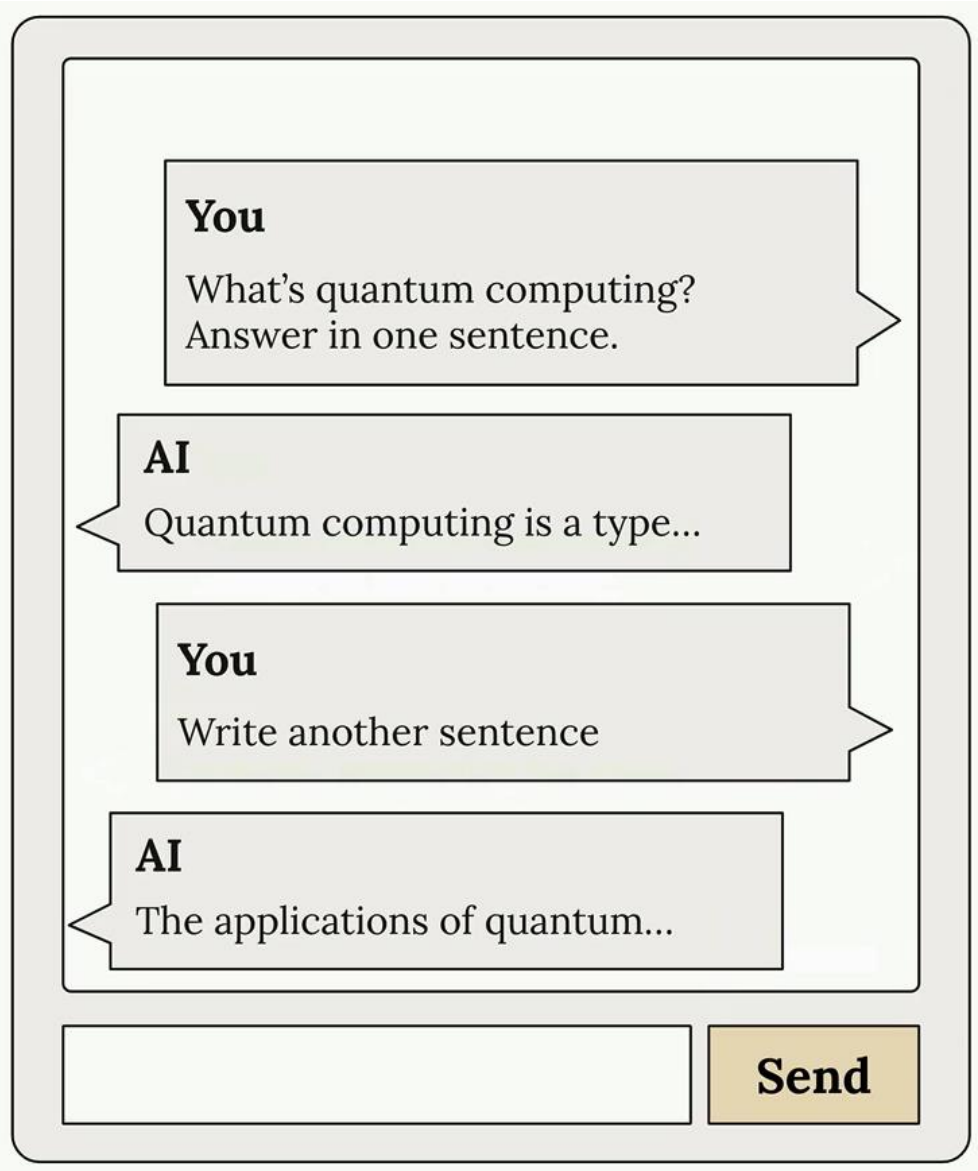
[12]:
```
{'ResponseMetadata': {'RequestId': '185b2617-6fd1-4f54-9ce5-457cc5d6ef52',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Sun, 17 Aug 2025 17:42:44 GMT',
   'content-type': 'application/json',
   'content-length': '561',
   'connection': 'keep-alive',
   'x-amzn-requestid': '185b2617-6fd1-4f54-9ce5-457cc5d6ef52',
   'x-amzn-bedrock-invocation-latency': '1345',
   'x-amzn-bedrock-output-token-count': '46',
   'x-amzn-bedrock-input-token-count': '16'},
  'RetryAttempts': 0},
 'contentType': 'application/json',
 'body': <botocore.response.StreamingBody at 0x1c1b6686100>}
```

[13]:
```
message = json.loads(response['body'].read().decode('utf-8'))
message
```

[13]:
```
{'id': 'msg_bdrk_01UhTiAEztGFYj2UZijz8vco',
 'type': 'message',
 'role': 'assistant',
 'model': 'claude-3-7-sonnet-20250219',
 'content': [{'type': 'text',
   'text': 'Quantum computing is a field that utilizes quantum mechanical phenomena such as superposition and entanglement to perform calculations using quantum bits (qubits), potentially solving certain comple
x problems exponentially faster than classical computers.'}],
 'stop_reason': 'end_turn',
 'stop_sequence': None,
 'usage': {'input_tokens': 16,
  'cache_creation_input_tokens': 0,
  'cache_read_input_tokens': 0,
  'output_tokens': 46}}
```

Azercell

Multi-Turn Conversations

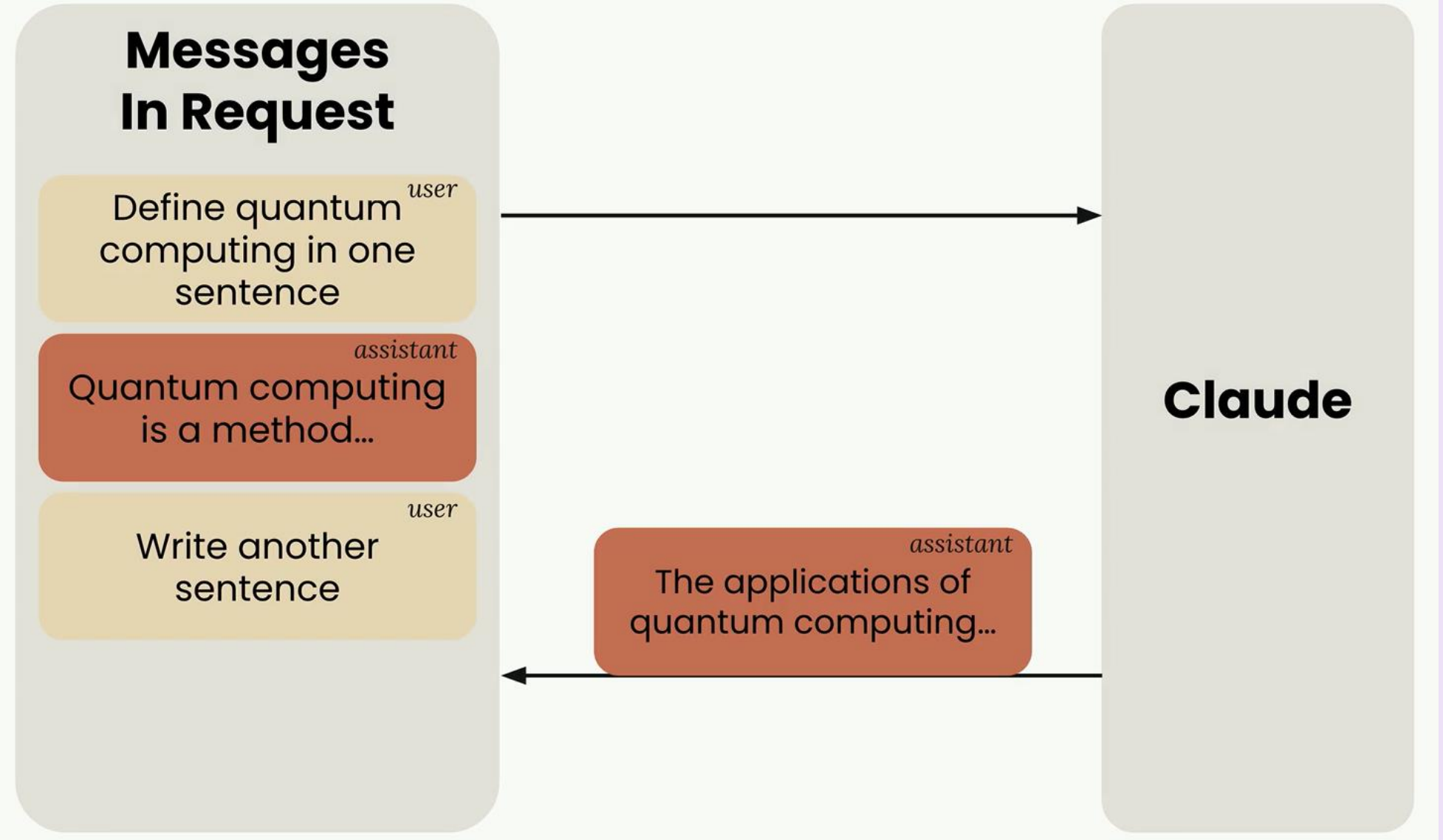


The Anthropic API and Claude **do not store** any messages

To have a 'conversation', you need to:

- Manually maintain a list of messages in your code
- Provide that list of messages with each follow up request

Multi-Turn Conversations



Multi-Turn Conversations

```
[30]: %%time
# Make a starting list of messages
messages = []

# Add in the initial user question
add_user_message(messages, "Define quantum computing in one sentence")

# Pass the list of messages into 'chat' to get an answer
answer = chat(messages)

# Take the answer and add it as an assisstant message into our list
add_assistant_message(messages, answer)

# Add the users follow-up question
add_user_message(messages, "Write another message")

# Call the chat again with the list of messages to get the final answer
answer = chat(messages)

add_assistant_message(messages, answer)
messages

CPU times: total: 15.6 ms
Wall time: 3.48 s

[30]: [{'role': 'user', 'content': 'Define quantum computing in one sentence'},
 {'role': 'assistant',
  'content': 'Quantum computing is a type of computing that uses quantum-mechanical phenomena such as superposition and entanglement to perform operations on data, potentially solving certain problems exponenti
ally faster than classical computers.'},
 {'role': 'user', 'content': 'Write another message'},
 {'role': 'assistant',
  'content': 'Quantum computing harnesses the principles of quantum mechanics to process information using qubits that can exist in multiple states simultaneously, enabling computational approaches that could r
evolutionize fields from cryptography to drug discovery.'}]
```

Azercell

System Prompt

System prompts provide Claude guidance on how to respond

Claude will try to respond in the same way someone in the specified role would respond

Helps keep Claude on task

```
system_prompt="""
You are a patient math tutor. Do not directly
answer a student's questions. Guide them to a
solution step by step.
"""

client.messages.create(
    model=model,
    messages=messages,
    max_tokens=1000,
    system=system_prompt
)
```

System Prompt

System prompts provide Claude guidance on how to respond

Claude will try to respond in the same way someone in the specified role would respond

Helps keep Claude on task

```
system_prompt="""
You are a patient math tutor. Do not directly
answer a student's questions. Guide them to a
solution step by step.
"""

client.messages.create(
    model=model,
    messages=messages,
    max_tokens=1000,
    system=system_prompt
)
```

Azercell

System Prompt

System prompt ¶

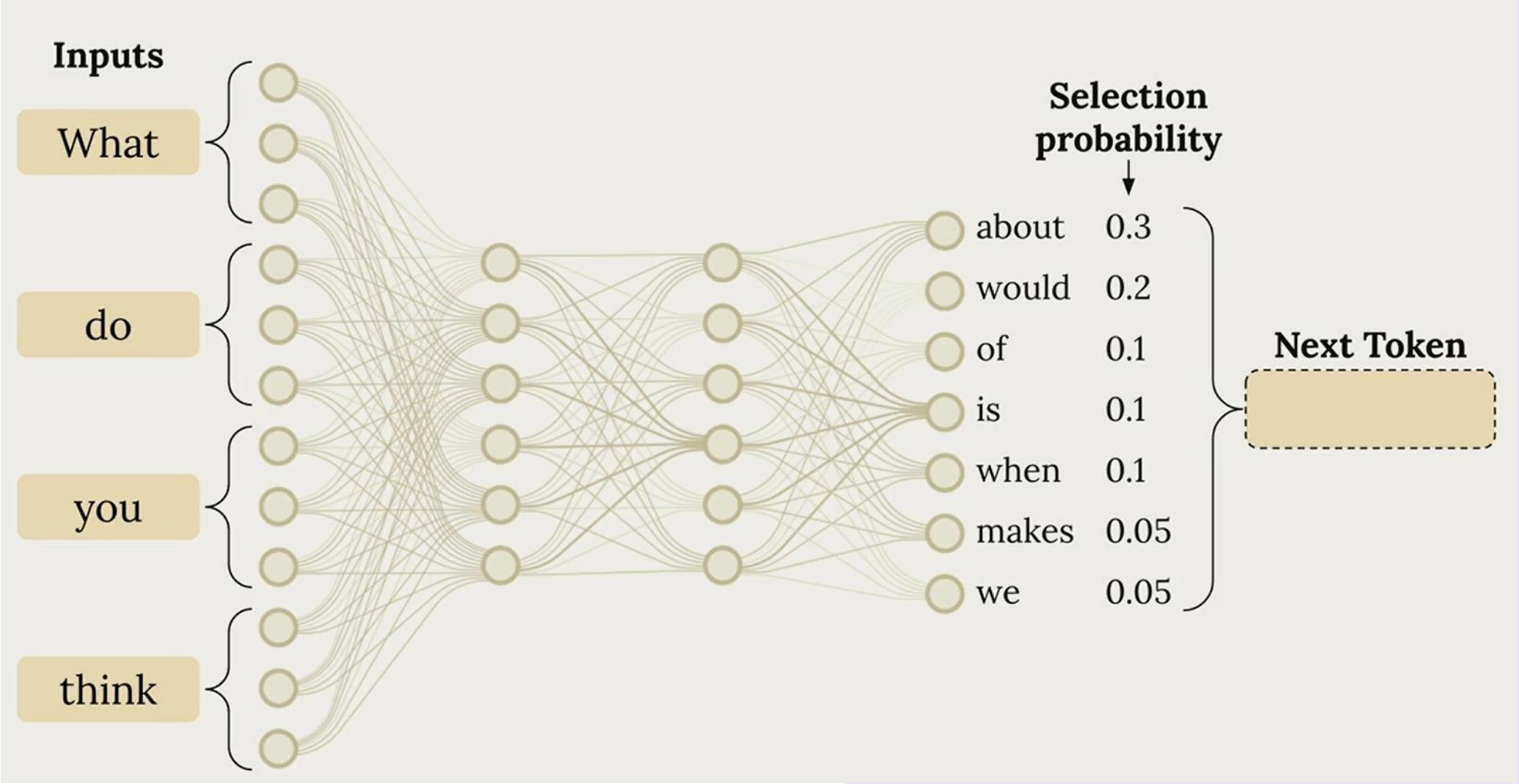
[69]:
```
messages = []
add_user_message(messages, "How do I solve 5x+3=2 for x?")
answer = chat(messages)
print(answer)
```

```
# Solving 5x+3=2 for x

To solve this equation for x, I need to isolate the variable.

Step 1: Subtract 3 from both sides of the equation.
5x + 3 - 3 = 2 - 3
5x = -1

Step 2: Divide both sides by 5 to isolate x.
5x/5 = -1/5
x = -1/5 or x = -0.2

Therefore, x = -1/5 is the solution to the equation 5x+3=2.
```

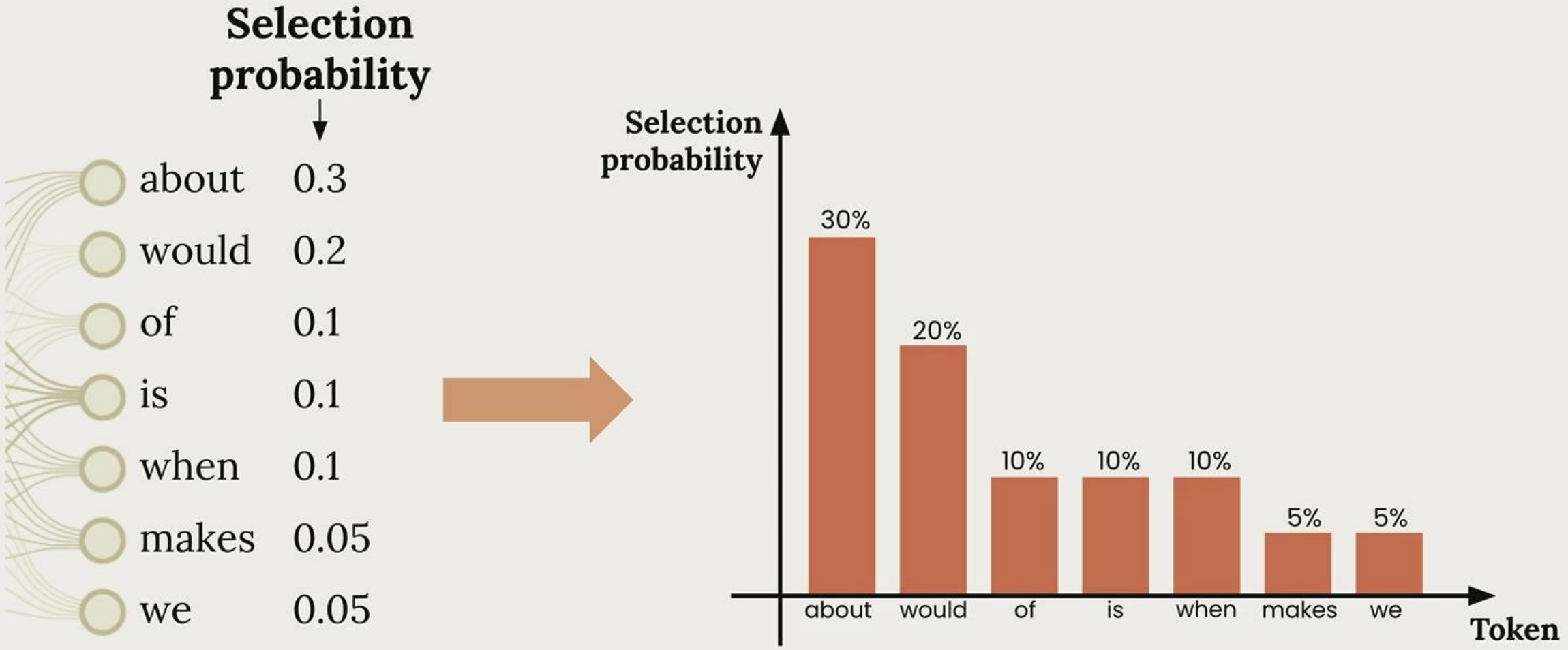
[67]:
```
messages = []
system = """
    You are a patient math tutor.
    Do not give directly answer a student's question.
    Guide them to a solution step by step.
    """
add_user_message(messages, "How do I solve 5x+3=2 for x?")
answer = chat(messages, system=system)
print(answer)
```

```
I'd be happy to guide you through solving the equation 5x + 3 = 2 step by step.

Let's think about what we need to do. When solving for x, our goal is to isolate x on one side of the equation.

First, what do you think we should do with the "+3" term that's currently on the same side as the x term?
```

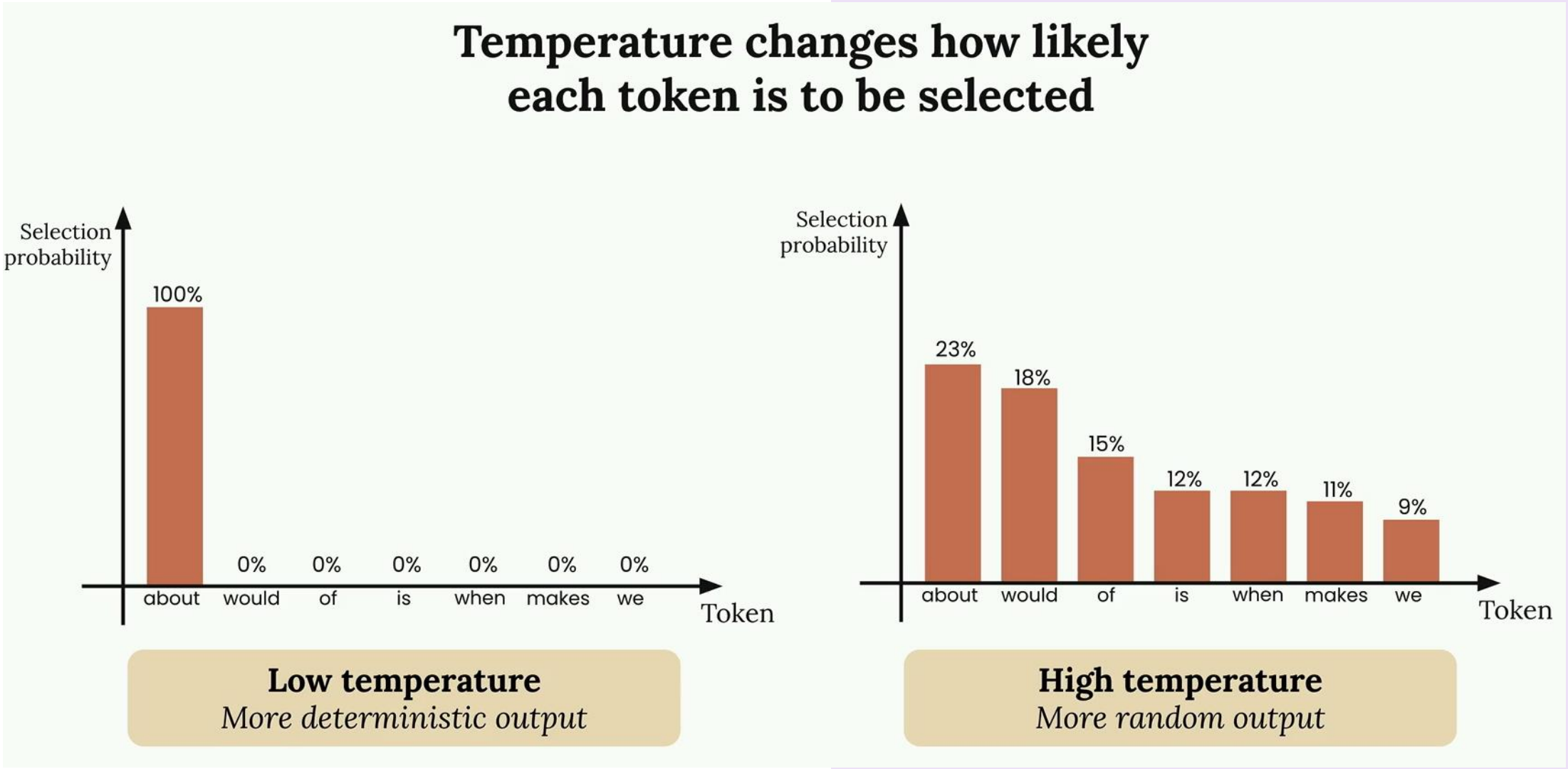
Temperature



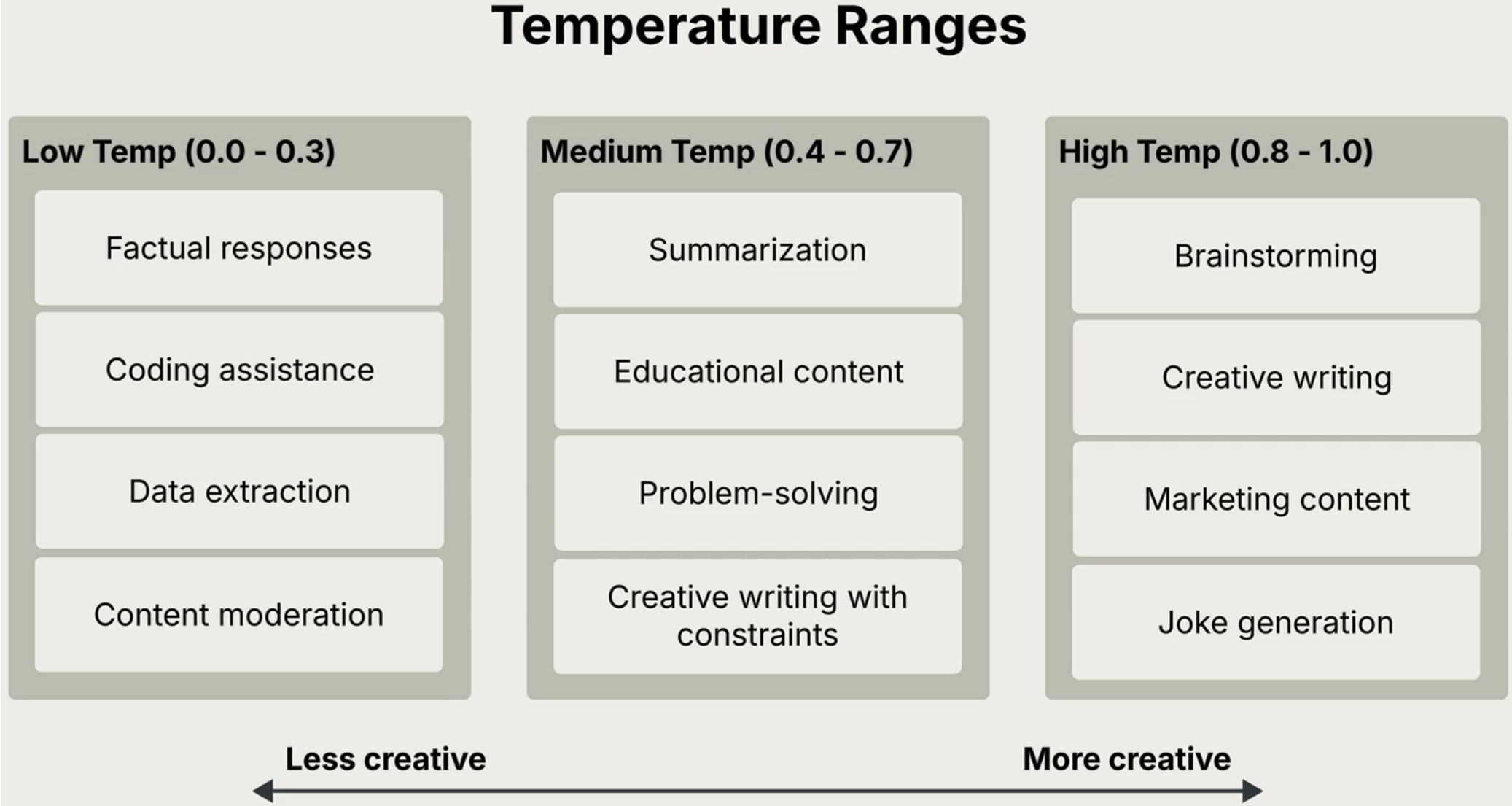
Temperature



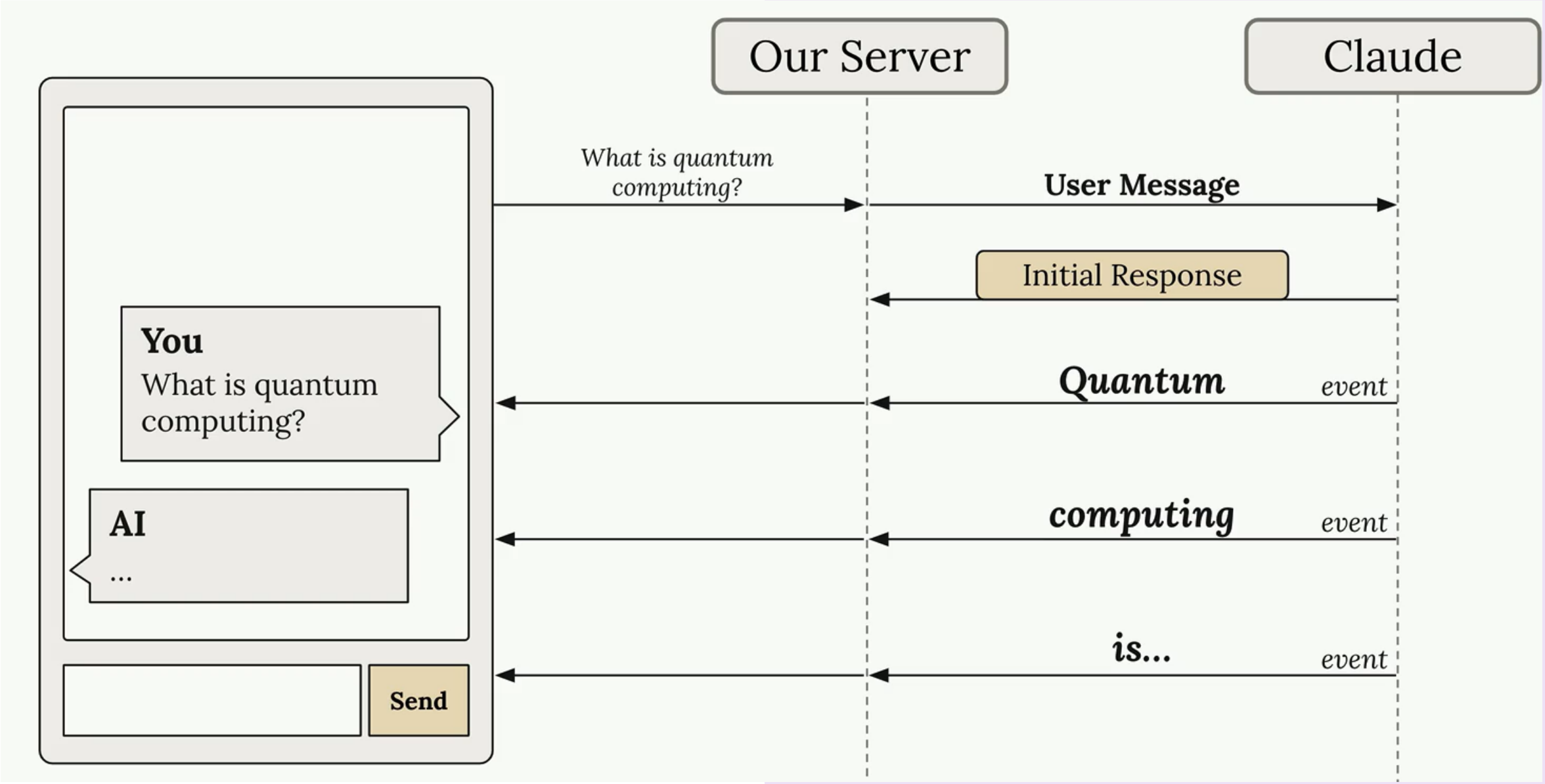
Temperature



Temperature



Response Streaming



Response Streaming

Response Streaming ¶

[95]:
```
messages = []

add_user_message(messages, "Write a 1 sentence description of a fake database")
body_json = create_body_json(messages=messages)

stream = client.invoke_model_with_response_stream(
            modelId=model,
            contentType="application/json",
            accept="application/json",
            body=body_json
        )
stream_body = stream.get("body")

for event in stream_body:
    print(event)
```

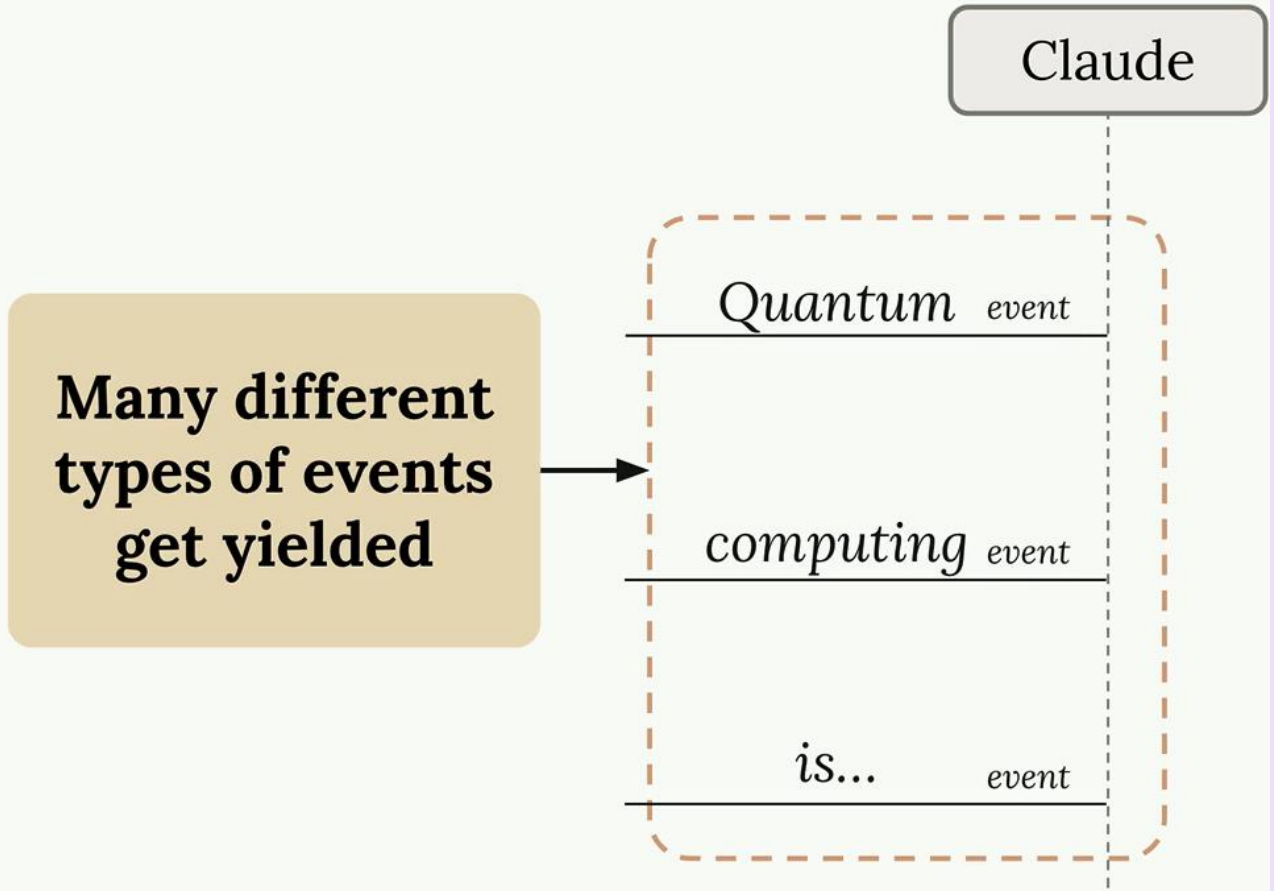
```
{'chunk': {'bytes': b'{"type":"message_start","message":{"id":"msg_bdrk_016XAaucLzpCtqfcWJWqaaJa","type":"message","role":"assistant","model":"claude-3-7-sonnet-20250219","content":[],"stop_reason":null,"stop_s
equence":null,"usage":{"input_tokens":18,"cache_creation_input_tokens":0,"cache_read_input_tokens":0,"output_tokens":2}}}'}}
{'chunk': {'bytes': b'{"type":"content_block_start","index":0,"content_block":{"type":"text","text":""}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":"A fictional"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" repository known"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" as \\"Quant"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":"umVault\\" allegedly"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" stores interdimensional data"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" fragments that"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" automatically reorgan"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":"ize themselves base"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":"d on the"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" emotional state of the database"}}'}}
{'chunk': {'bytes': b'{"type":"content_block_delta","index":0,"delta":{"type":"text_delta","text":" administrator."}}'}}
{'chunk': {'bytes': b'{"type":"content_block_stop","index":0}'}}
{'chunk': {'bytes': b'{"type":"message_delta","delta":{"stop_reason":"end_turn","stop_sequence":null},"usage":{"output_tokens":37}}'}}
{'chunk': {'bytes': b'{"type":"message_stop","amazon-bedrock-invocationMetrics":{"inputTokenCount":18,"outputTokenCount":37,"invocationLatency":1305,"firstByteLatency":415}}'}}
```

Azercell

Response Streaming

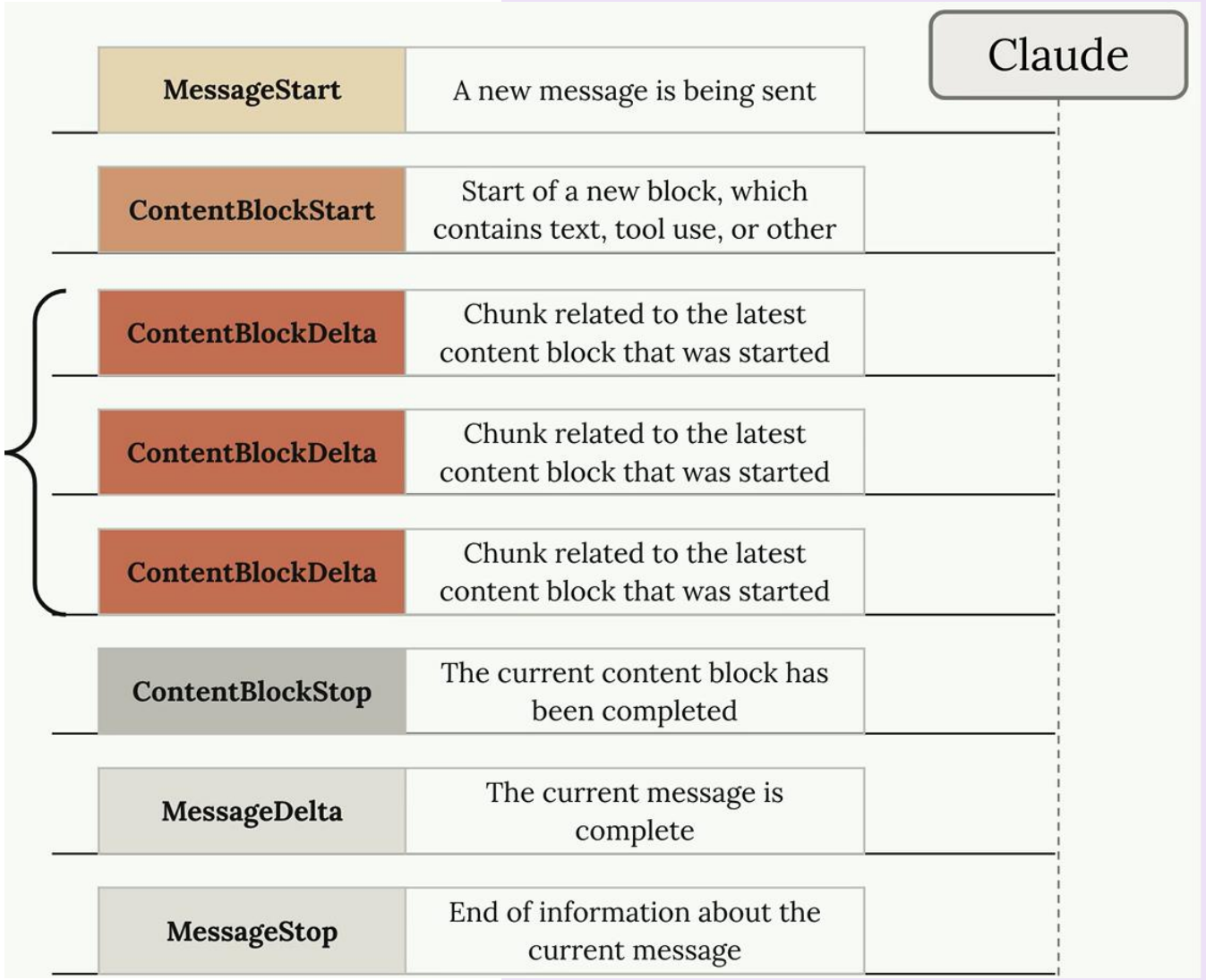
Common Events

Event Type	Purpose
RawMessageStartEvent	A new message is being sent
RawContentBlockStartEvent	Start of a new block, which contains a text message, tool use, or other
RawContentBlockDeltaEvent	Chunk related to the latest content block that was started
RawContentBlockStopEvent	The current content block has been completed
RawMessageDeltaEvent	The current message is complete
RawMessageStopEvent	End of information about the current message



Response Streaming

These contains the actual generated text



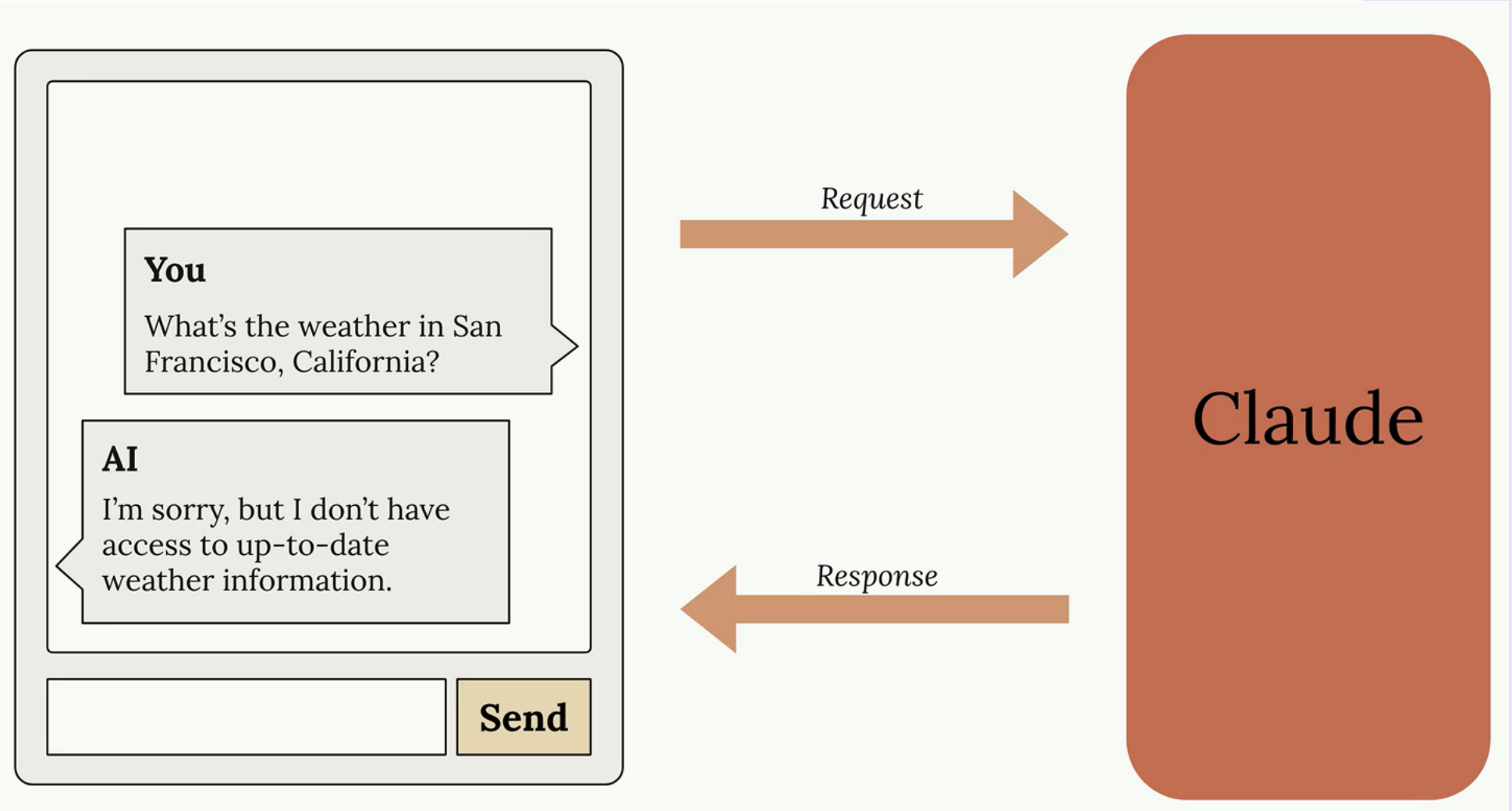
Response Streaming

[102]:
```python
messages = []

add_user_message(messages, "Write a 1 sentence description of a fake database")
body_json = create_body_json(messages=messages)

stream = client.invoke_model_with_response_stream(
            modelId=model,
            contentType="application/json",
            accept="application/json",
            body=body_json
        )
stream_body = stream.get("body")

for event in stream_body:
    stream_chunk = event.get("chunk")
    decoded = json.loads(stream_chunk.get("bytes").decode("utf-8"))
    delta = decoded.get("delta", {})
    text = delta.get("text", "")
    print(text, end="")
```

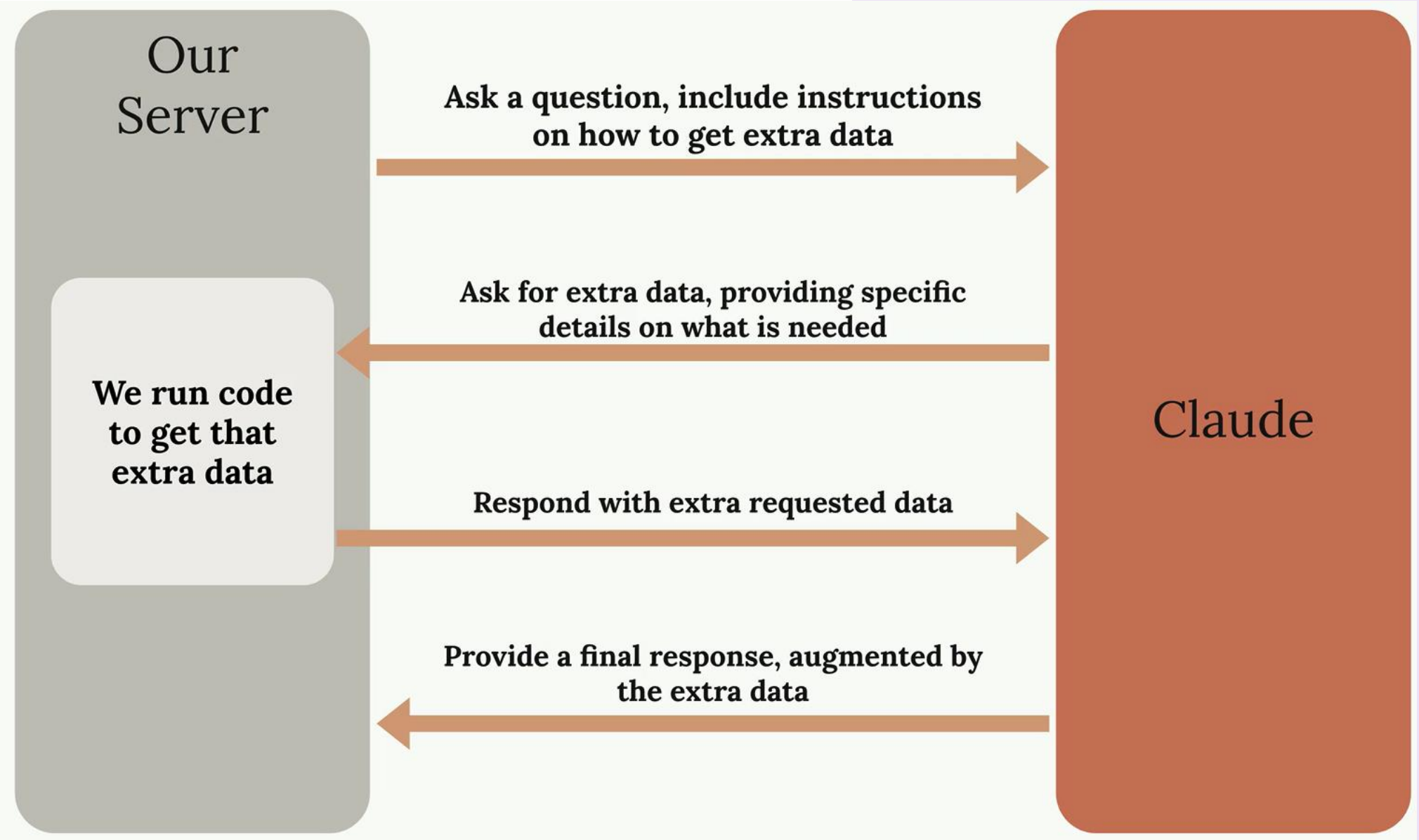
```
The HyperQuantum DataNexus is a cutting-edge, non-existent database system that purportedly processes information through quantum entanglement while storing unlimited data in pocket dimensions.
```

Azercell

Tool Use


You
What's the weather in San Francisco, California?
AI
I'm sorry, but I don't have access to up-to-date weather information.
Send
Request
Response
Claude

Tool use flow



Tool function

Write a tool function | Write a JSON Schema | Call Claude with JSON Schema | Run Tool | Add Tool Result and call Claude again

Tool Function

- Plain Python function that will be executed when Claude decides it needs some additional information to help the user
- **Best practices**
  - Use well-named, descriptive arguments
  - Validate the inputs, raising an error if they fail validation
  - Return meaningful errors - Claude will try to call to use your function a second time!

```python
def get_weather(location):
    if not location or location.strip() == "":
        raise ValueError("Location cannot be empty")

    url = "https://fakeweatherapi.example.com/current"
    params = {
        "q": location,
        "appid": api_key,
        "units": "metric"
    }

    response = requests.get(url, params=params, timeout=10)
    response.raise_for_status()

    return response.json()
```

Azercell

Tool function

Write a tool function | **Write a JSON Schema** | Call Claude with JSON Schema | Run Tool | Add Tool Result and call Claude again

Name and description of this tool.

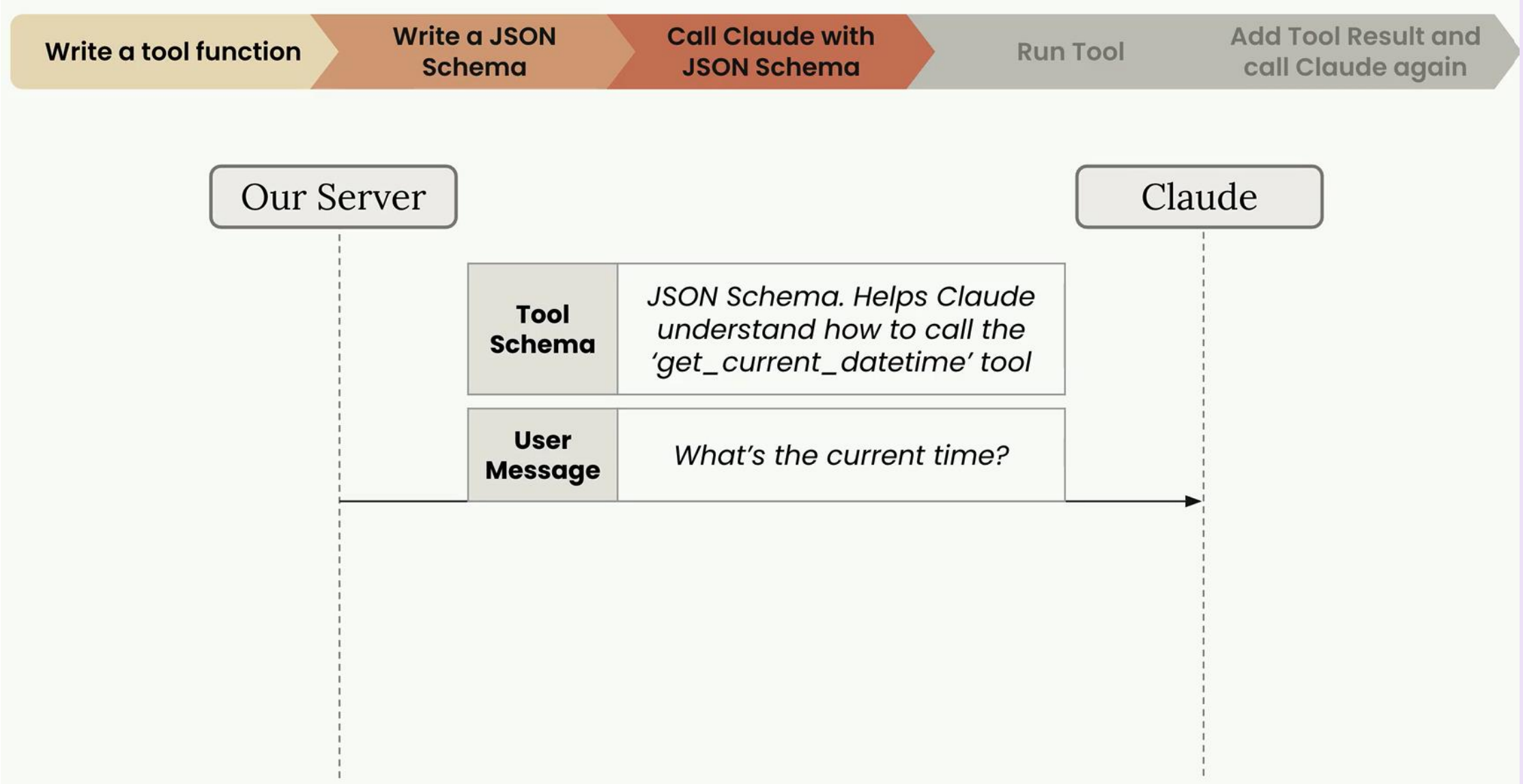
Helps Claude understand when to use it

This part is the actual schema that describes the tool function's arguments.

```
{
  "name": "get_weather",
  "description": "Retrieves current weather…",
  "input_schema": {
    "type": "object",
    "properties": {
      "location": {
        "type": "string",
        "description": "The location for which…"
      }
    },
    "required": [
      "location"
    ]
  }
}
```

Azercell

Tool function



Tool function

Write a tool function → Write a JSON Schema → Call Claude with JSON Schema → Run Tool → Add Tool Result and call Claude again

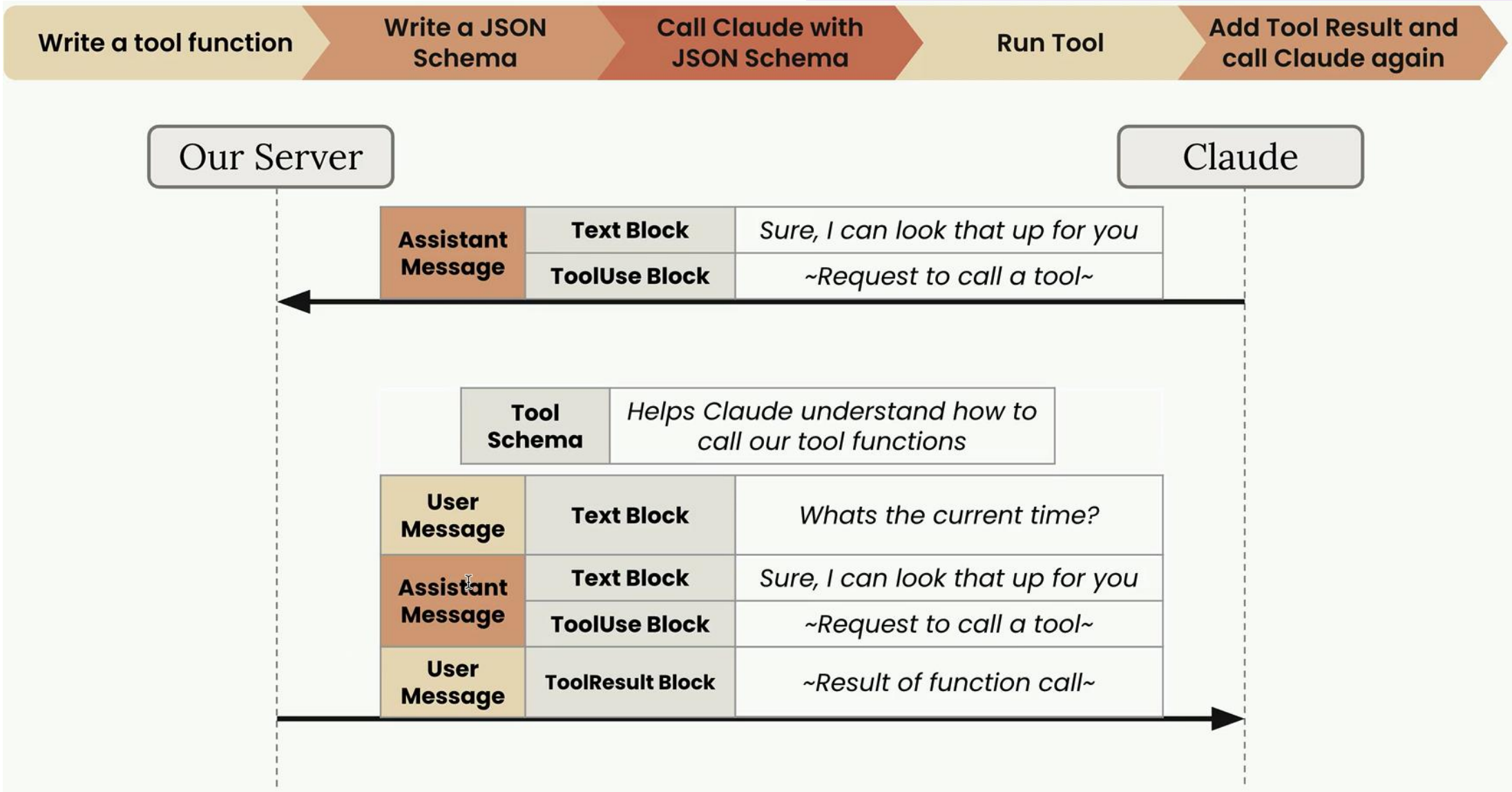
Multi-Block messages

Blocks are also often referred to as "parts"

Assistant Message's Content List	"Text" Block	*"I can help you find out the current time. Let me find that information for you"*
	"ToolUse" Block	`ToolUseBlock(id='toolu_01QtKMDNkU1FBj73NP2FbF8w', input={'date_format': '%H:%M:%S'}, name='get_current_datetime', type='tool_use')`

Azercell

Tool function



Tool function

Tool Result Block

- Placed in the 'content' list of a user message
- Communicates the results of running a tool function back to Claude
- **"tool_use_id"** - Must match the id of the ToolUse block that this ToolResult corresponds to
- **"content"** - Output from running your tool, serialized as a string
- **"is_error"** - True if an error occurred

```
{
 "tool_use_id": "toolu_01BEbi7q7qz",
 "type": "tool_result",
 "content": "12:47:13",
 "is_error": False
}
```

Azercell

Tool function

```
•[172]: body_json = create_body_json_with_tool(messages=messages, tools=get_current_weather_schema)
        response = client.invoke_model(
                modelId=model,
                body=body_json
            )
        message = json.loads(response['body'].read().decode('utf-8'))
        message
```

```
[172]: {'id': 'msg_bdrk_016bkx1qPxHvaXKjWMcSFW2U',
         'type': 'message',
         'role': 'assistant',
         'model': 'claude-3-7-sonnet-20250219',
         'content': [{'type': 'text',
           'text': "The current weather in Baku is:\n- Temperature: 23°C\n- Conditions: Clear\n- Humidity: 65%\n- Wind speed: 18 km/h\n\nIt's a nice clear day in B
        aku with comfortable temperatures."}],
         'stop_reason': 'end_turn',
         'stop_sequence': None,
         'usage': {'input_tokens': 509,
          'cache_creation_input_tokens': 0,
          'cache_read_input_tokens': 0,
          'output_tokens': 61}}
```

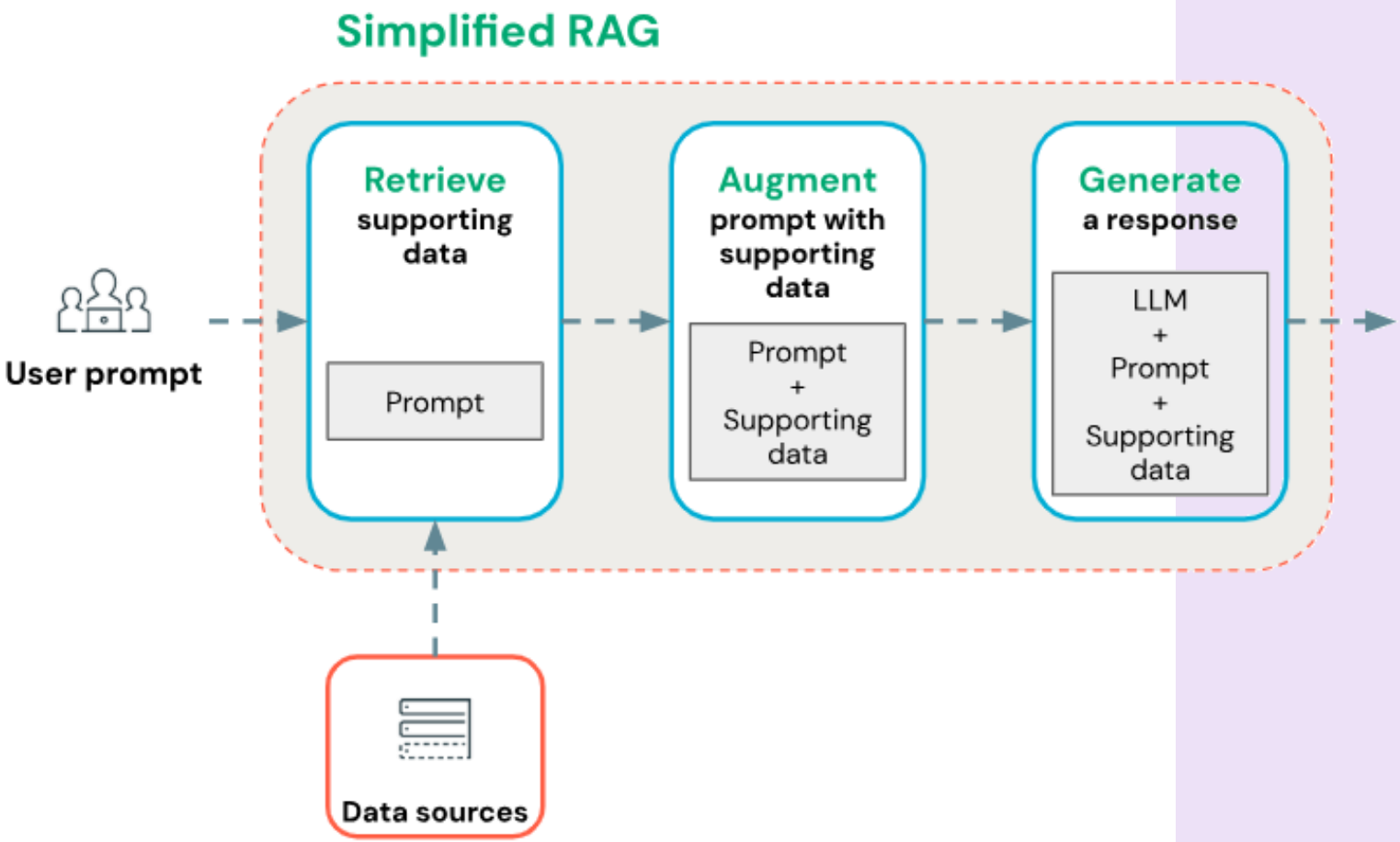
```
[173]: print(message['content'][0]['text'])
```

```
The current weather in Baku is:
- Temperature: 23°C
- Conditions: Clear
- Humidity: 65%
- Wind speed: 18 km/h

It's a nice clear day in Baku with comfortable temperatures.
```

RAG (Retrieval Augmented Generation)

Retrieval-Augmented Generation (RAG) is a powerful method that combines large language models (LLMs) with real-time data retrieval to produce more accurate, up-to-date, and contextually relevant answers.

This approach is especially important for questions involving private or frequently changing information, or when operating in a specific domain.



RAG Agent: Simplest Workflow:

- **Retrieval**: Use the user's query to search an external knowledge base (e.g., a vector store, keyword search, or SQL database) and fetch supporting context for the LLM.
- **Augmentation**: Combine the retrieved context with the user's query—often via a prompt template with formatting and instructions—to construct the final prompt.
- **Generation**: Send the constructed prompt to the LLM to generate the answer to the user's request.

RAG Pros and Cons

RAG – Pros:

- **Proprietary knowledge:** Pulls in private docs, emails, notes without retraining.
- **Freshness:** Fetches up-to-date info at query time.
- **Source attribution:** Can cite passages/links for verification.
- **Access control:** Honors per-user permissions at retrieval.
- **Cost/efficiency:** Boosts smaller LLMs with context instead of fine-tuning.
- **Domain fit:** Adapts to niche vocab and internal schemas.

RAG – Cons:

- **Retrieval sensitivity:** If recall/precision is poor, answers degrade.
- **Context limits:** Long or noisy context can crowd out the useful bits.
- **Pipeline upkeep:** Indexing, re-embedding, dedup, and drift management.
- **Security risks:** Prompt injection, data poisoning, ACL mistakes.
- **Evaluation is hard:** Measuring groundedness/end-to-end quality is non-trivial.

Quick mitigations: hybrid retrieval + reranking, smart chunking, caching, guardrails (allowlists/sanitization), incremental indexing, and task-level evals (faithfulness/groundedness).

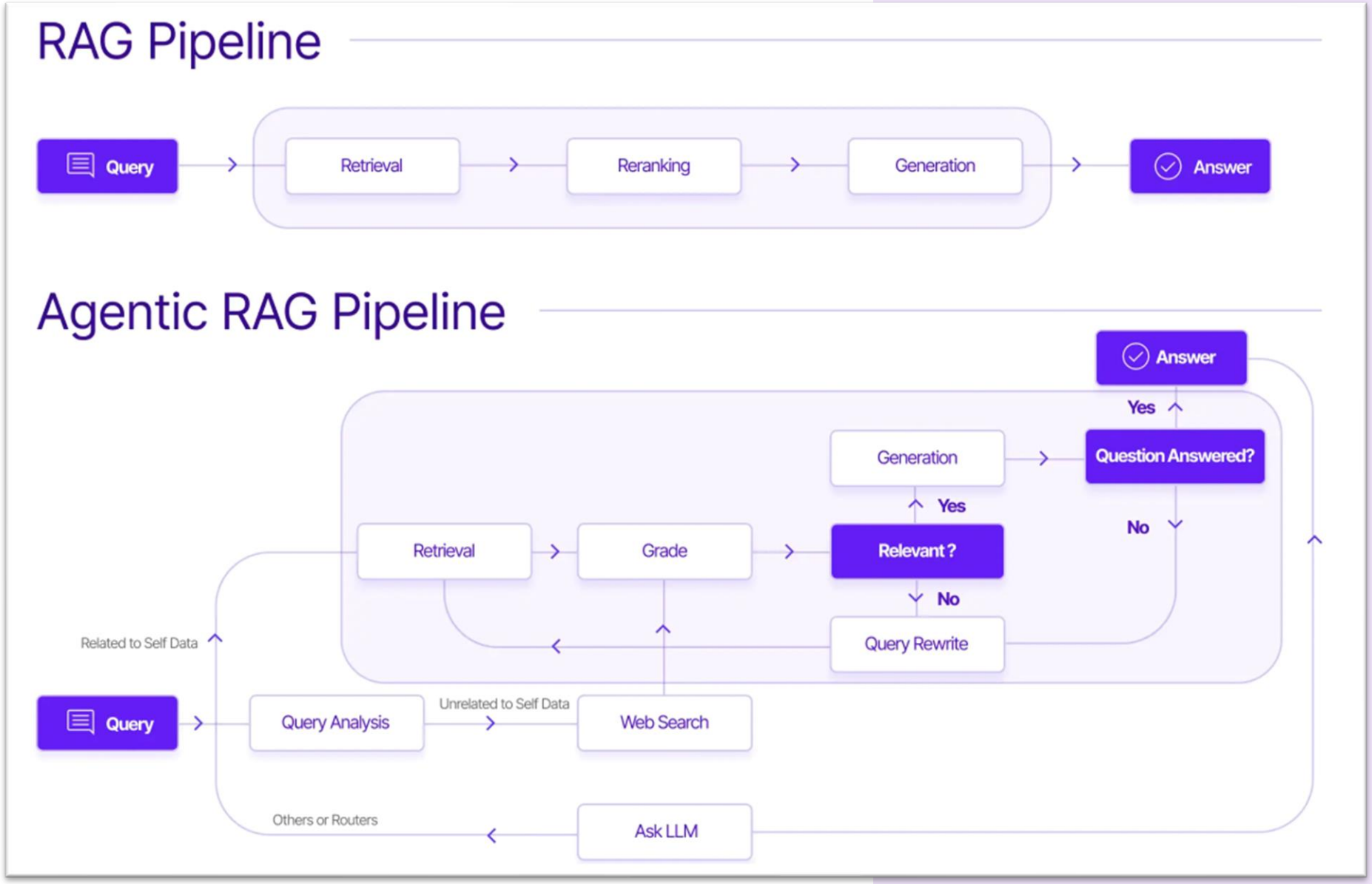
Azercell

RAG (Retrieval Augmented Generation)

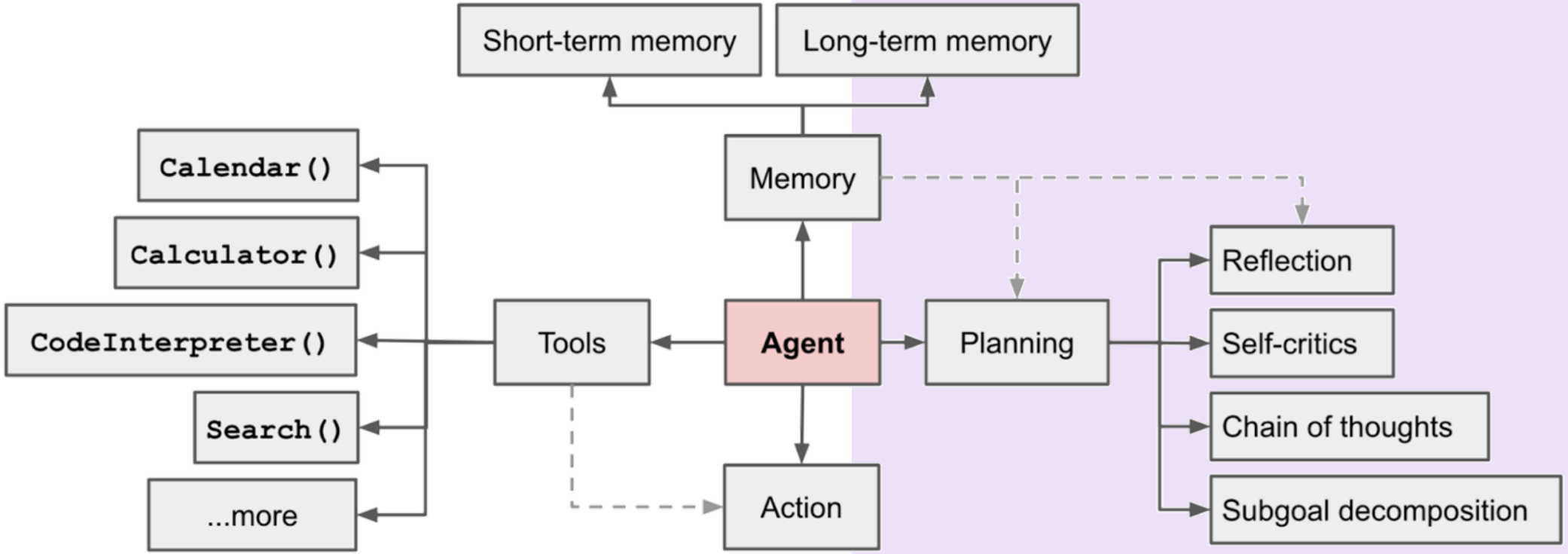
[205]:
```
%%time
messages.append({"role": "assistant", "content": message['content']})
result = get_knowledge_base_data(**message['content'][1]['input'])
messages.append({
        "role": "user",
        "content": [
            {
            "type": "tool_result",
            "tool_use_id": message['content'][1]['id'],
            "content": str(result),
            "is_error": False
            }
        ]
})
body_json = create_body_json_with_tool(messages=messages, tools=get_knowledge_base_data_schema)
response = client.invoke_model(
        modelId=model,
        body=body_json
    )
message = json.loads(response['body'].read().decode('utf-8'))
print(message['content'][0]['text'])
```

```
Based on the information from Azercell's knowledge base, I can provide you with Azercell's core values:

# Azercell's Core Values

According to Azercell's Code of Conduct and Business Ethics, their core values are:

1. **We build trust** - In public and private, they take ownership for their actions, always do what they say they will do, and behave honestly and consist
ently, adhering to the highest ethical standards.

2. **We work best together** - They are one Azercell community, working together to deliver seamless and superior experiences for internal and external cus
tomers.

3. **We look to the future** - They are up for challenges and always look for bold new ways to improve how people connect with each other and the world.

4. **We stand up for our customers** - They deliver products, services, and processes that are designed around their customers and are easy to use.

These values support Azercell's vision: "Easing connectivity. Empowering lives. Across Azerbaijan!"
CPU times: total: 0 ns
Wall time: 5.64 s
```

Agentic RAG vs Traditional RAG



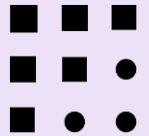
Agent



Source: https://lilianweng.github.io/posts/2023-06-23-agent/

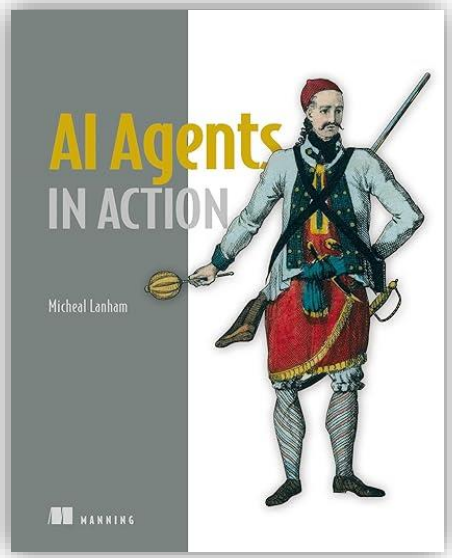
Agentic Frameworks

Framework	Language	Description	Best For
LangGraph	Python, Javascript	A graph-based framework built on top of LangChain, designed to coordinate multiple agents and tools with memory.	Structured agent workflows with LLMs.
CrewAI	Python	Simple and intuitive agent orchestration framework for teams of LLM agents. Uses a “Crew” and “Agent” model with roles and goals.	Task-oriented multi-agent collaboration.
AutoGen	Python	Microsoft’s framework for building LLM-powered agents that communicate via messaging. Supports human-AI and AI-AI collaboration.	Multi-agent chat and complex interactions.
AgentVerse	Python	Designed for simulating and evaluating large numbers of agents. Ideal for research on collective behaviors.	Multi-agent simulations and social dynamics.
OpenAgents	Python	Open-source project combining LangChain, FastAPI, and other tools. Allows UI integration and multi-agent task execution.	Developer-friendly agent apps.
MiniAGI	Python	Lightweight framework for building LLM-based agents using memory, tools, and recursive reasoning.	Quick experimentation with agents.
MetaGPT	Python	Treats AI agents like team members in a software company (e.g., PM, engineer, QA). Modular and production-oriented.	Building software with agent teams.

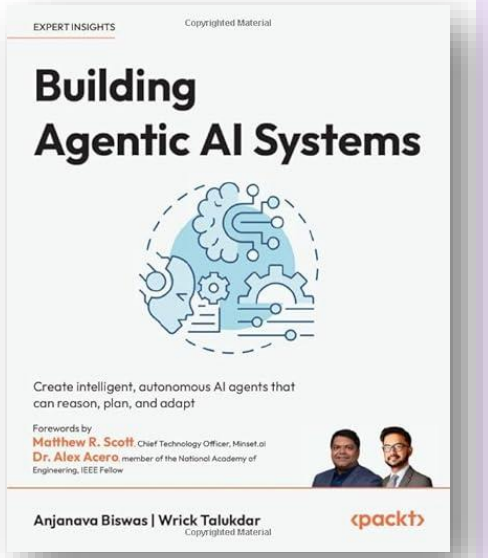


Literature & Links

ChatDev: Communicative Agents for Software Development
Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, Maosong Sun



AI Agents in Action
Michael Lanham



Building Agentic AI Systems: Create intelligent, autonomous AI agents

Anjanava Biswas, Wrick Talukdar

https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf

Useful links:

- https://markovate.com/agentic-rag/
- https://www.databricks.com/glossary/retrieval-augmented-generation-rag
- https://strandsagents.com/latest/documentation/docs/user-guide/concepts/tools/example-tools-package/

Might be helpful for Streamlit:

- https://github.com/Elkhn/mcp-playground/tree/main

Azercell


Azercell