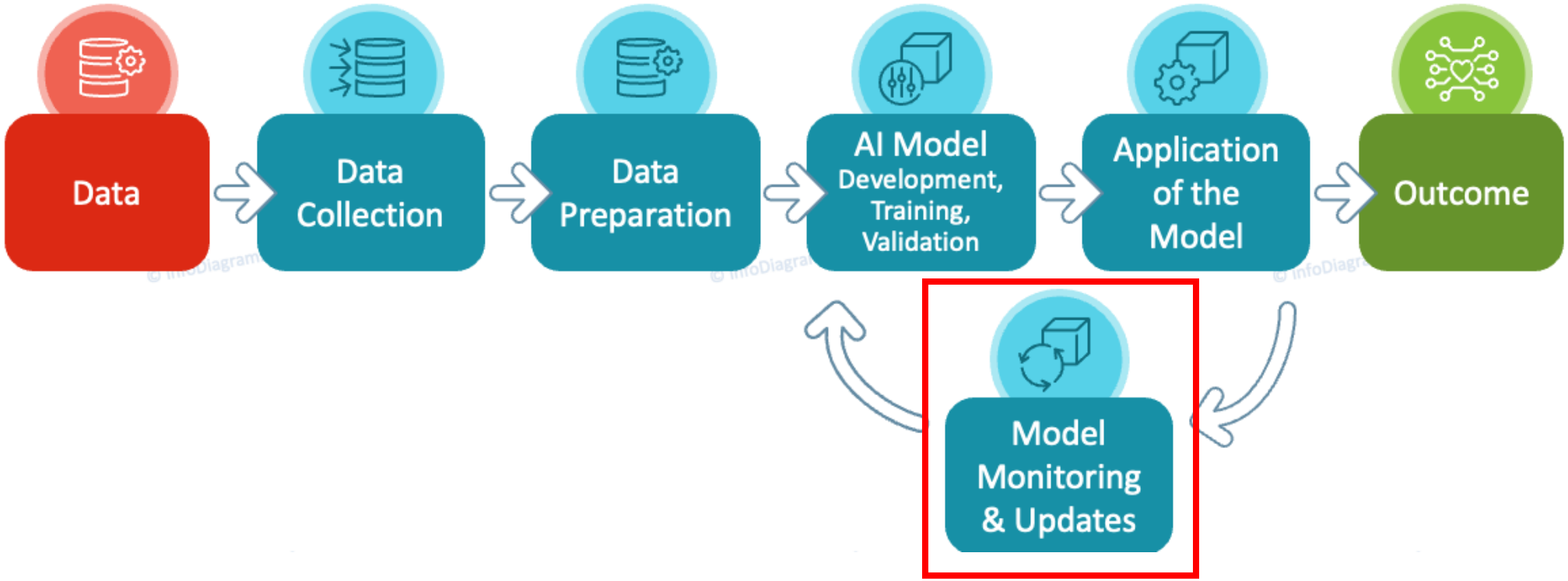
IMEN891K/IDSC501 Data Engineering

19. **LLM Systems III**: Agents

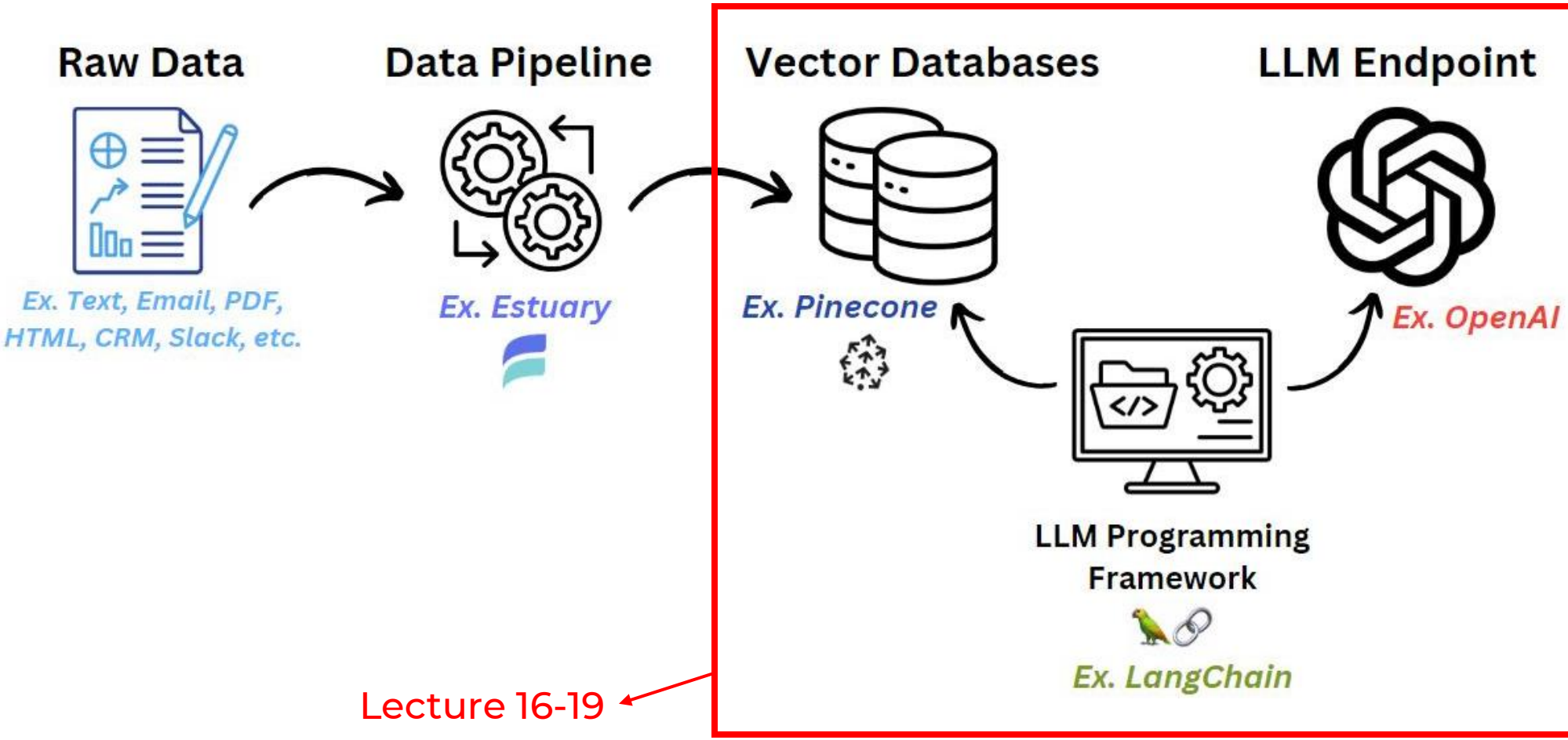
Fall 2025

Instructor: **Sangwoo Mo**

Recap: Course Overview

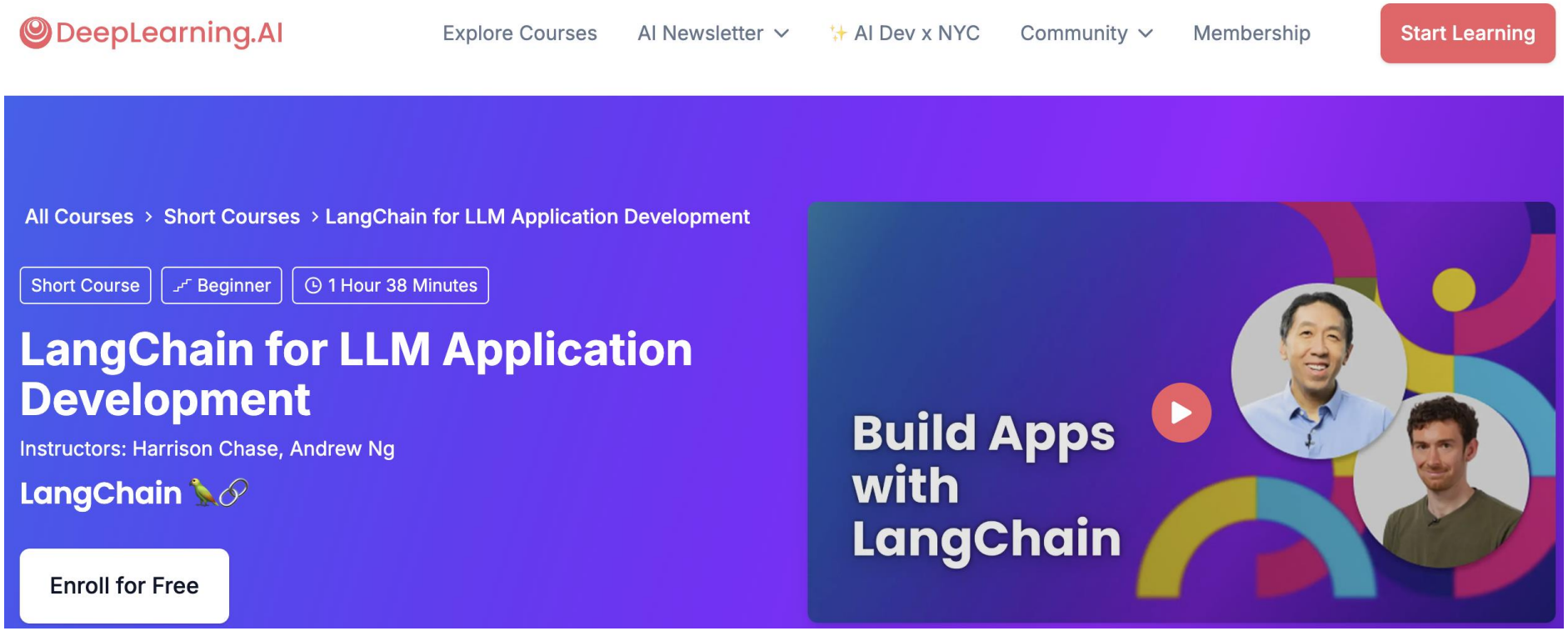


Recap: Course Overview

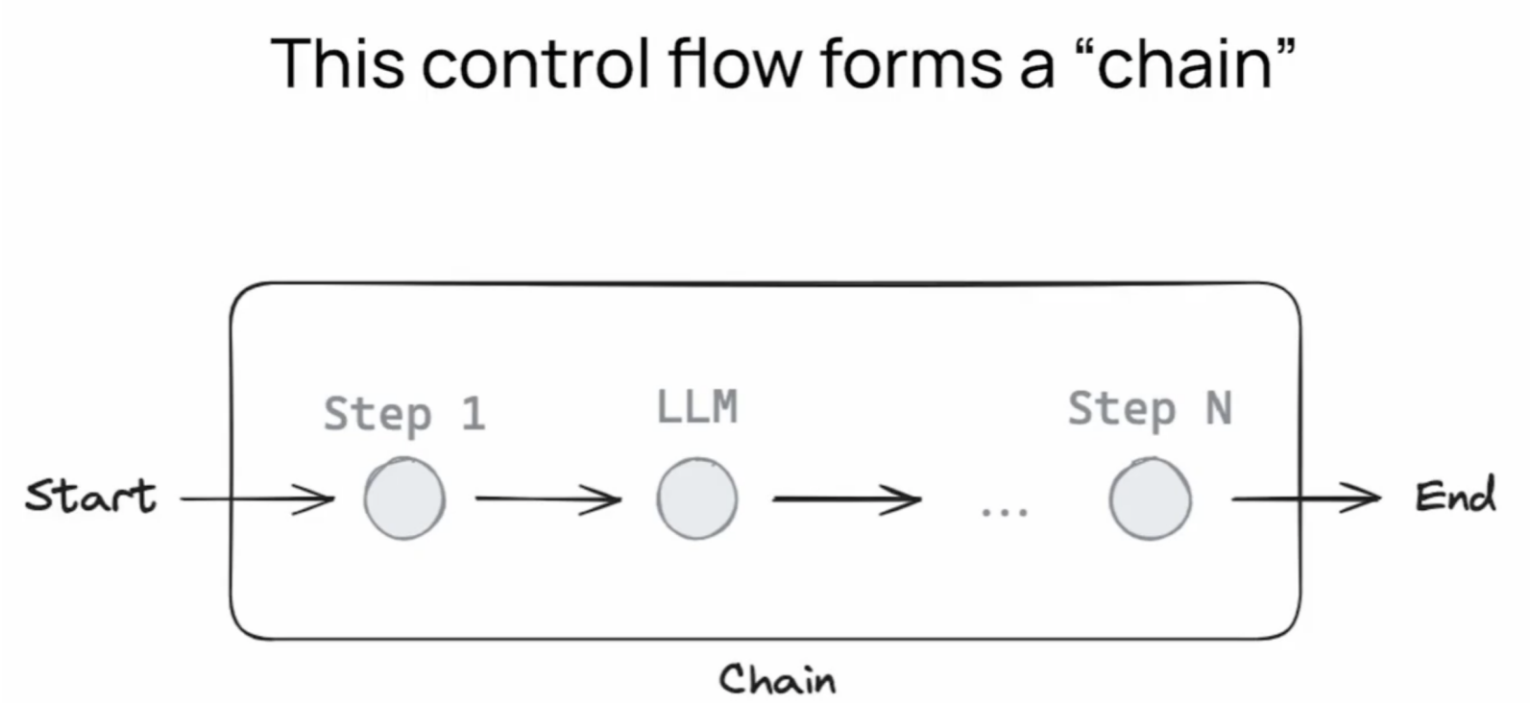


Disclaimer

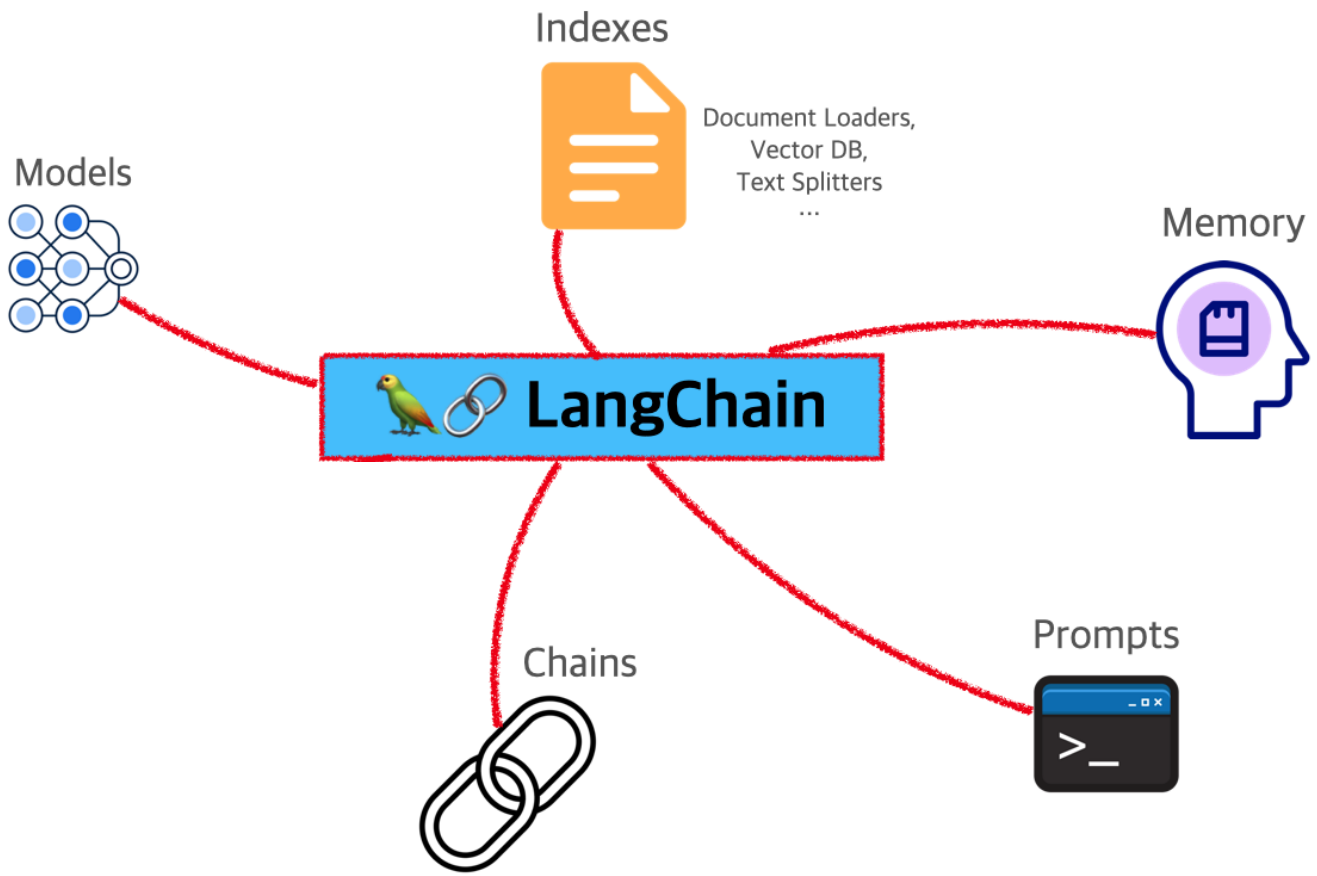
The LangChain part was heavily copy-pasted from DeepLearning.AI



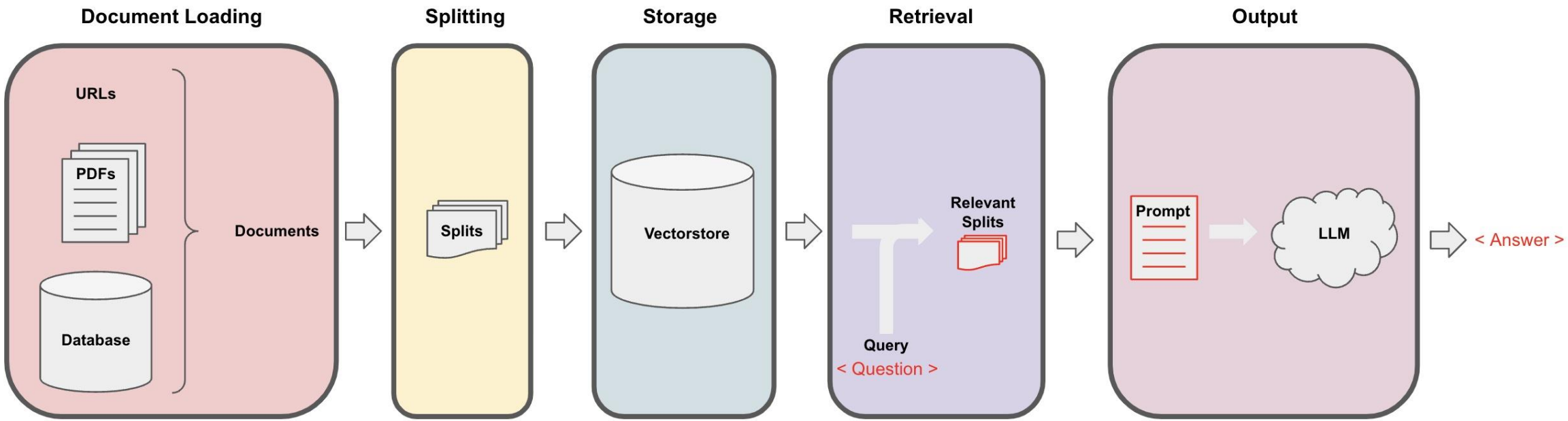
Recap: LangChain = LLM + Chain



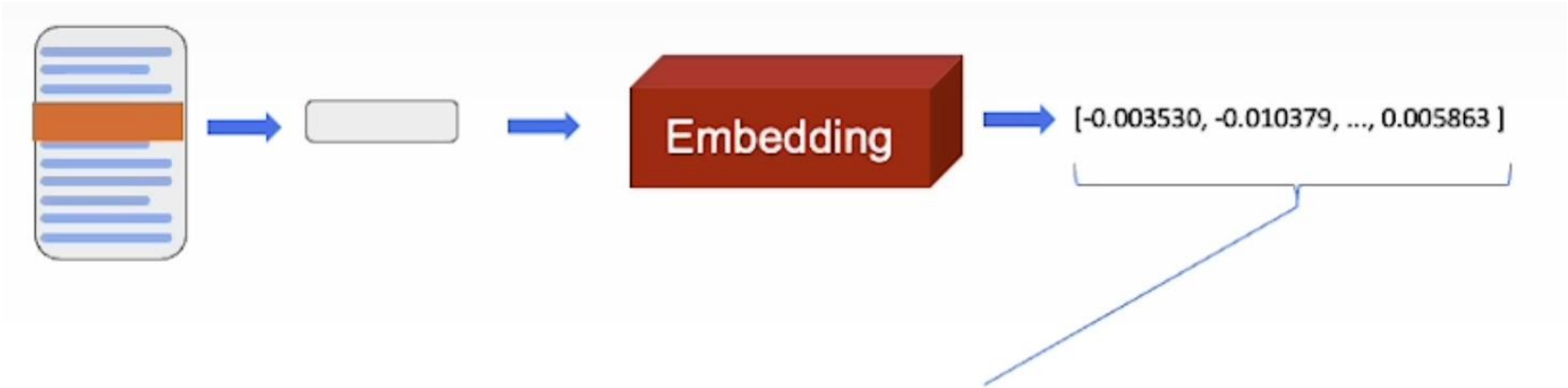
Recap: LangChain – What We've Learned



Recap: Retrieval Augmented Generation (RAG)

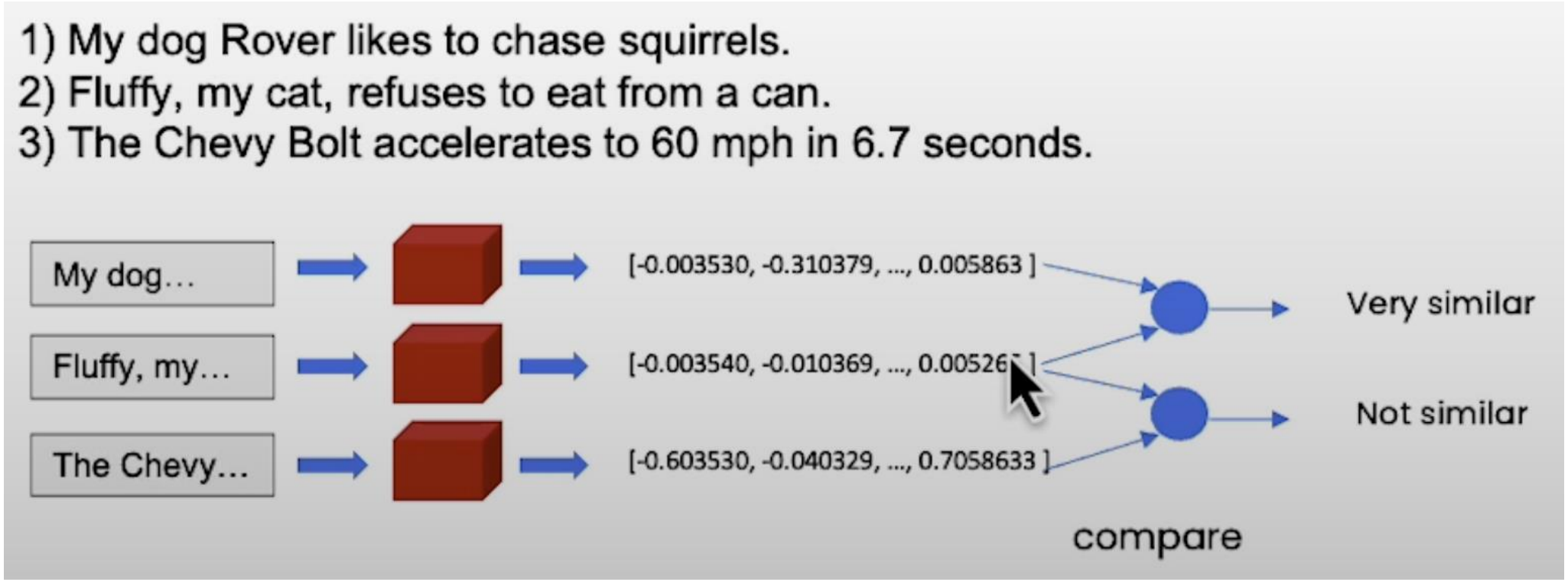


Recap: RAG – Embeddings

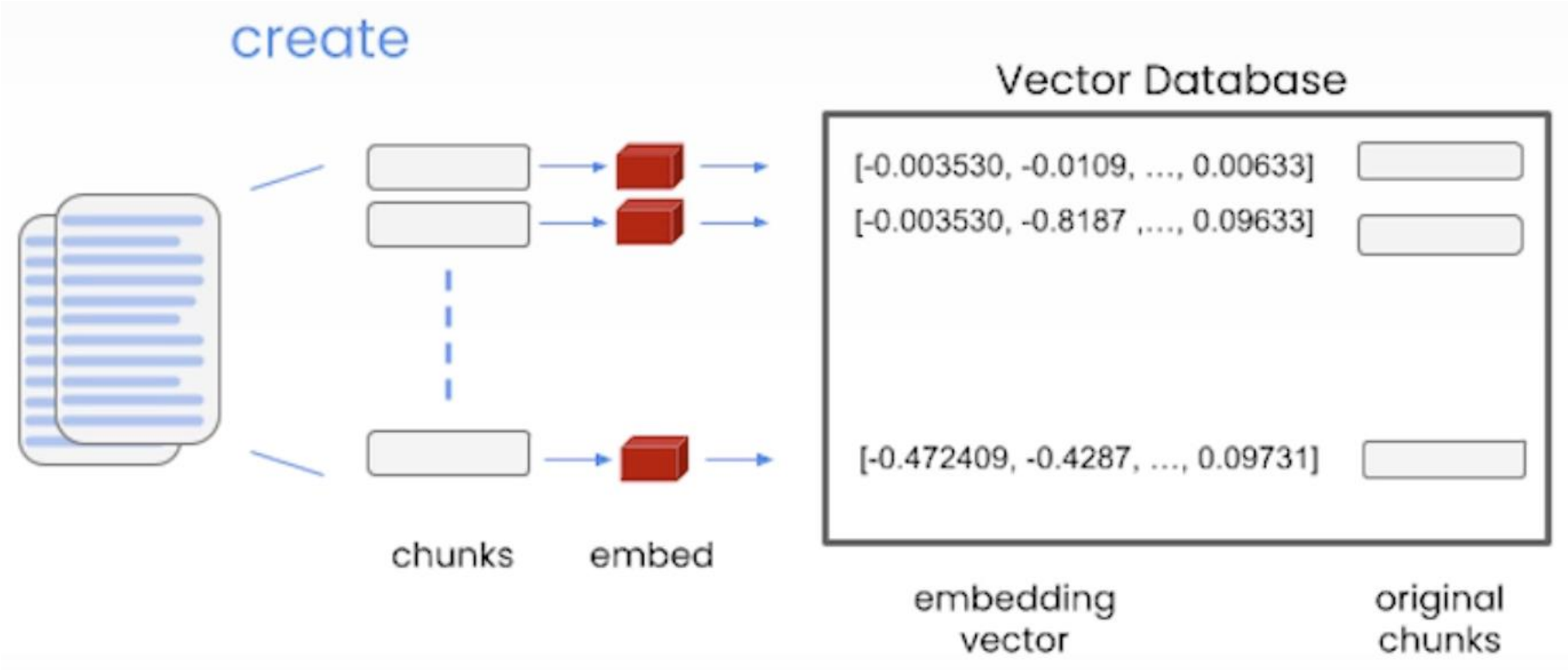


- Embedding vector captures content/meaning
- Text with similar content will have similar vectors

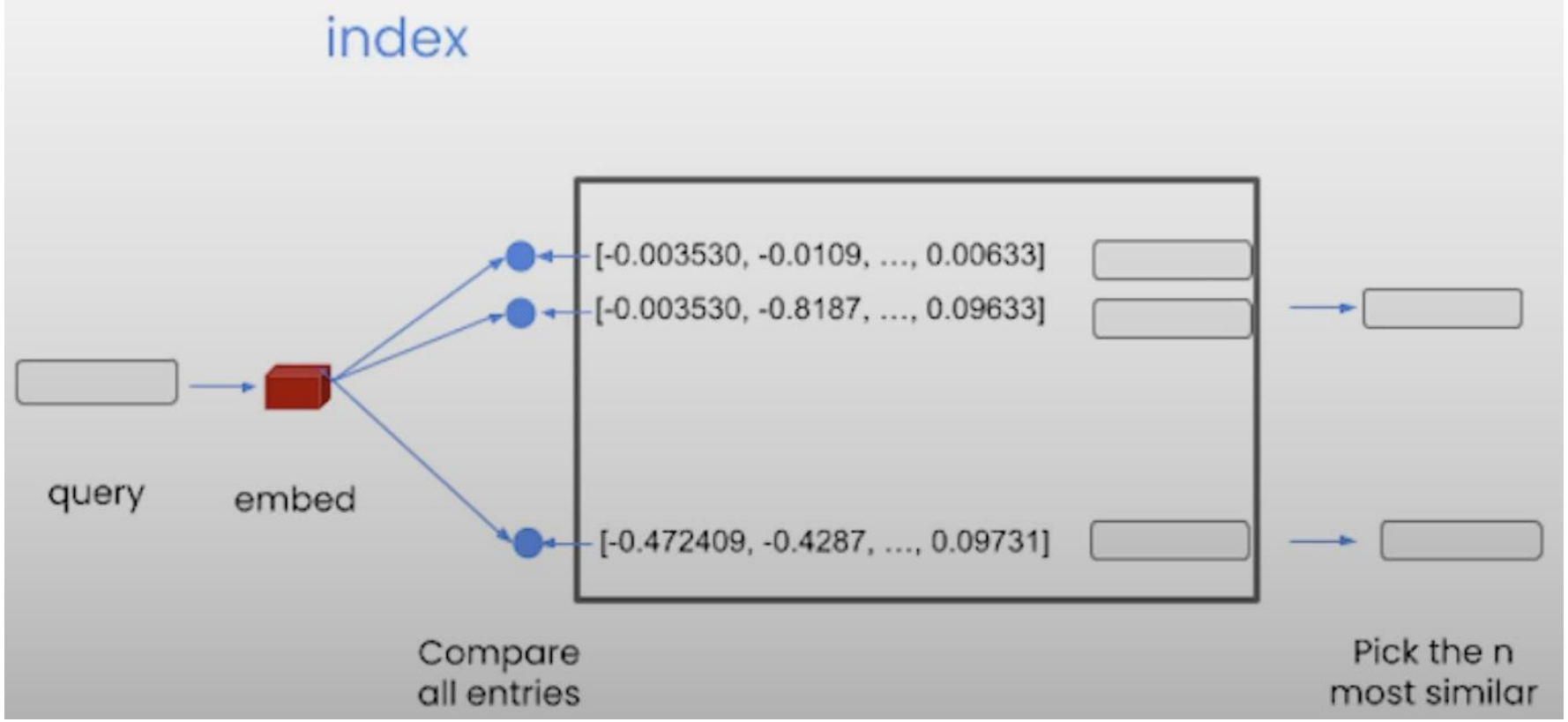
Recap: RAG – Embeddings



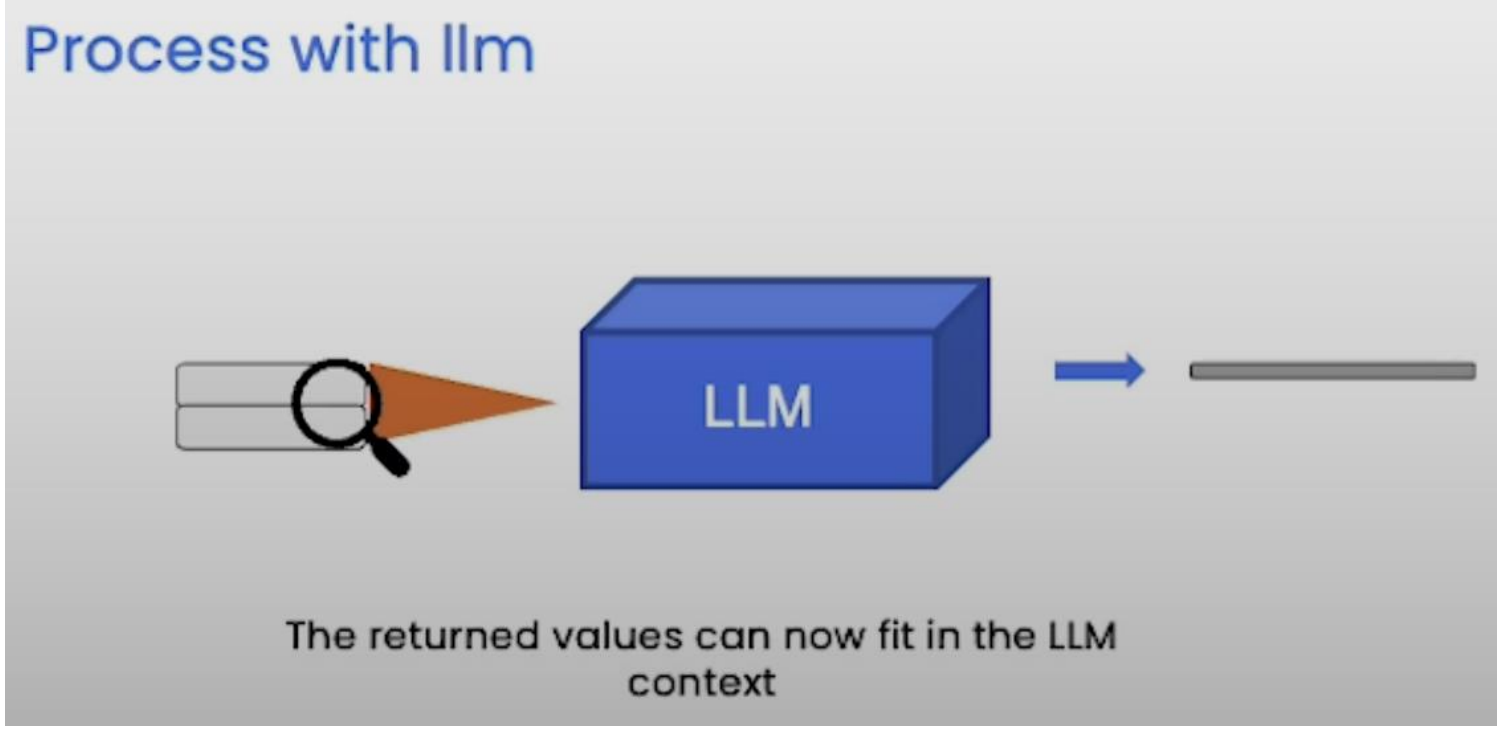
Recap: RAG – Vector DB



Recap: RAG – Vector DB



Recap: RAG – Process w/ LLM



Recap: RAG – Example (All-in-one)

```
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import CSVLoader
from langchain.vectorstores import DocArrayInMemorySearch
from IPython.display import display, Markdown
from langchain.llms import OpenAI
```

```
file = 'OutdoorClothingCatalog_1000.csv'
loader = CSVLoader(file_path=file)
```

```
from langchain.indexes import VectorstoreIndexCreator
```

```
#pip install docarray
```

```
index = VectorstoreIndexCreator(
    vectorstore_cls=DocArrayInMemorySearch
).from_loaders([loader])
```

Recap: RAG – Example (Step-by-step)

```
db = DocArrayInMemorySearch.from_documents(
    docs,
    embeddings
)
```

```
query = "Please suggest a shirt with sunblocking"
```

```
docs = db.similarity_search(query)
```

```
len(docs)
```

```
4
```

Recap: RAG – Example (Step-by-step)

```
retriever = db.as_retriever()
```

```
llm = ChatOpenAI(temperature = 0.0)
```

```
qdocs = "".join([docs[i].page_content for i in range(len(dc
```

```
response = llm.call_as_llm(f"{qdocs} Question: Please list
shirts with sun protection in a table in markdown and summa
```

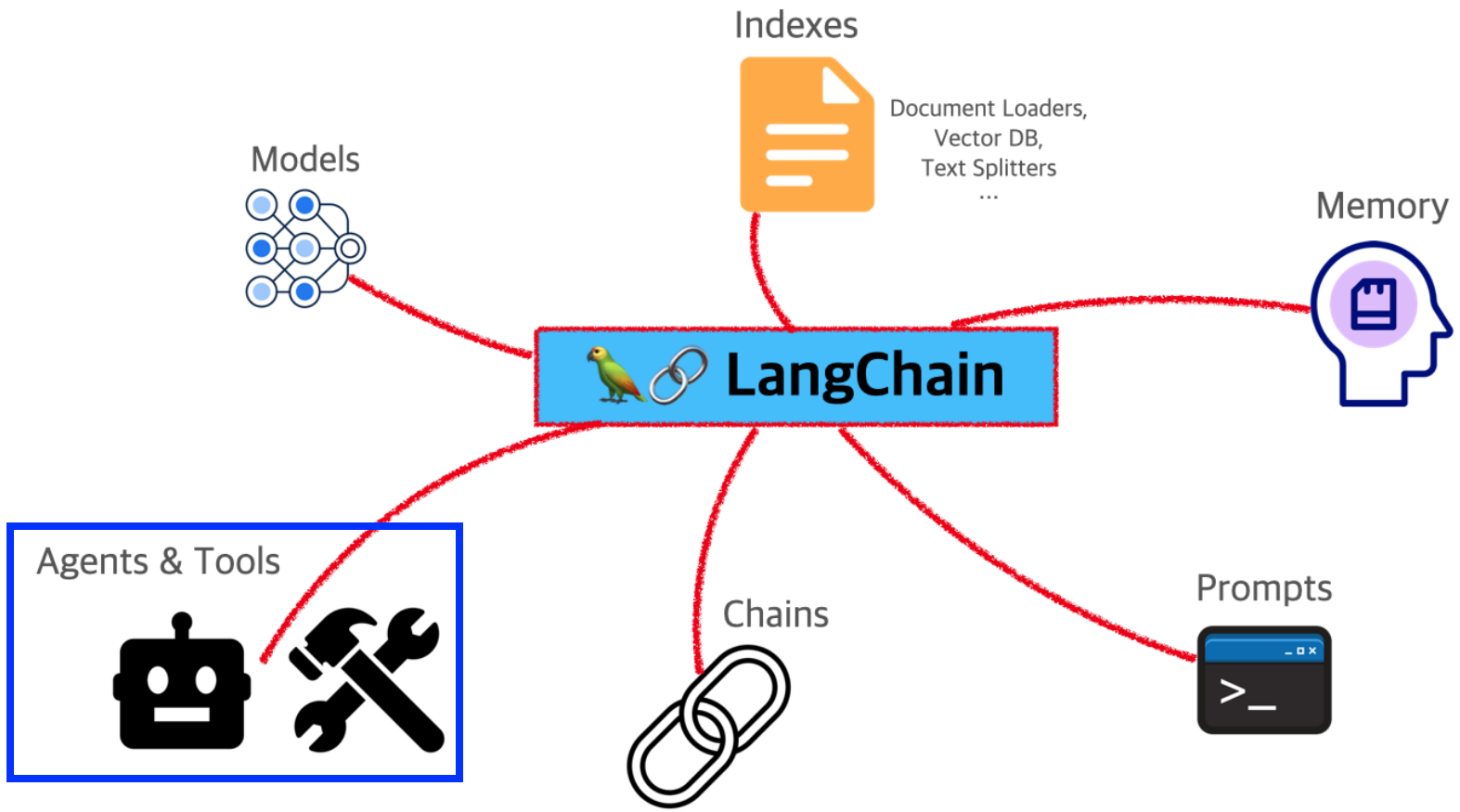
Recap: RAG – Example (Step-by-step)

```
qa_stuff = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    verbose=True
)
```

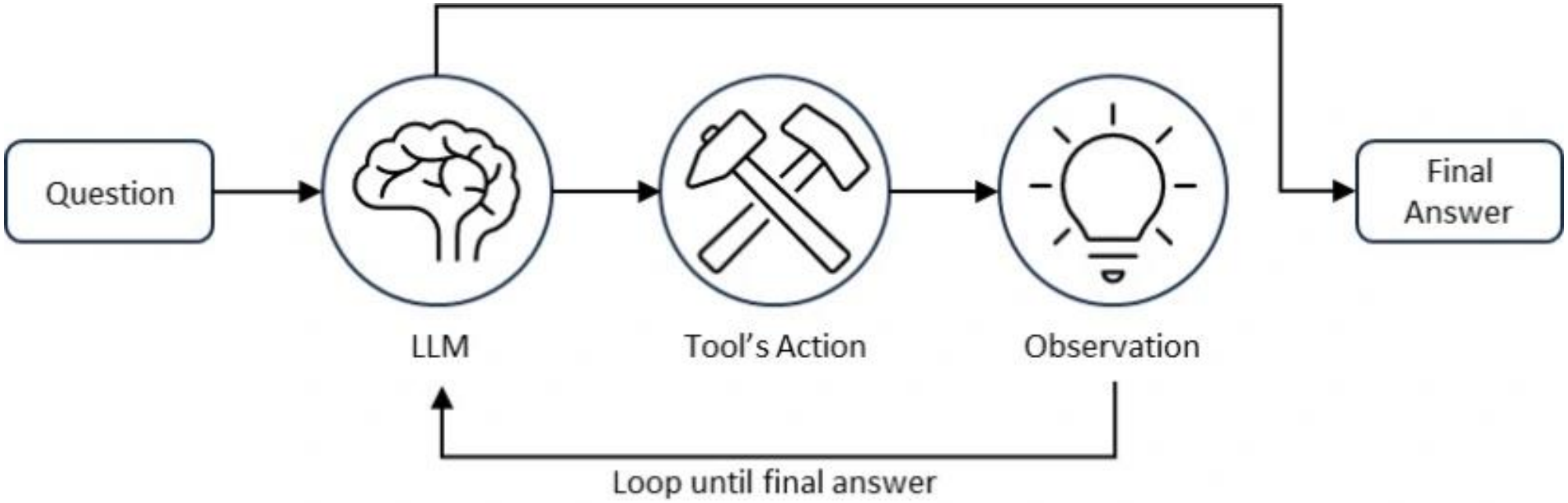
```
query =  "Please list all your shirts with sun protection i
in markdown and summarize each one."
```

```
response = qa_stuff.run(query)
```

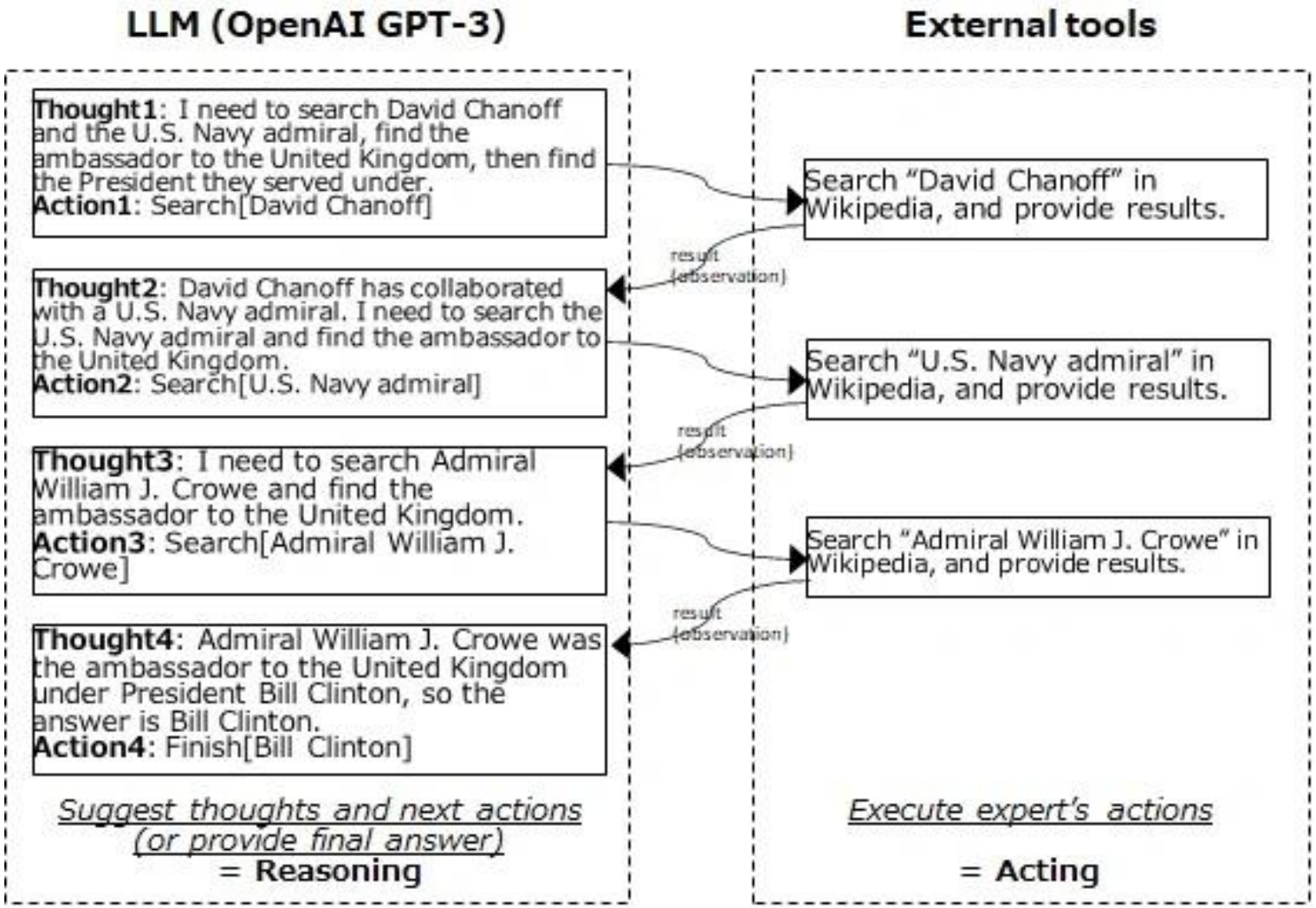
Agents & Tools



ReAct = Reasoning + Acting



ReAct = Reasoning + Acting



Function Calling

- OpenAI has fine-tuned the `gpt-3.5-turbo-0613` and `gpt-4-0613` models to:
  1. Accept additional arguments through which users can pass in descriptions of functions.
  2. If it is relevant, return the name of the function to use, along with a JSON object with the appropriate input parameters.

Function Calling

```python
import json

# Example dummy function hard coded to return the same weather
# In production, this could be your backend API or an external API
def get_current_weather(location, unit="fahrenheit"):
    """Get the current weather in a given location"""
    weather_info = {
        "location": location,
        "temperature": "72",
        "unit": unit,
        "forecast": ["sunny", "windy"],
    }
    return json.dumps(weather_info)
```

```python
# define a function
functions = [
    {
        "name": "get_current_weather",
        "description": "Get the current weather in a given locatio
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San F
                },
                "unit": {"type": "string", "enum": ["celsius", "fa
            },
            "required": ["location"],
        },
    }
]
```

Function Calling

```
# Call the ChatCompletion endpoint
response = openai.ChatCompletion.create(
    # OpenAI Updates: As of June 2024, we are now using the GPT-3..
    model="gpt-3.5-turbo",
    messages=messages,
    functions=functions
)
```

Function Calling

```
print(response)
```


```
{
  "id": "chatcmpl-89yAVKkWOZN4qeGCSE58lliwwKIOY",
  "object": "chat.completion",
  "created": 1697386931,
  "model": "gpt-3.5-turbo-0613",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": null,
        "function_call": {
          "name": "get_current_weather",
          "arguments": "{\n  \"location\": \"Boston, MA\"\n}"
        }
      },
      "finish_reason": "function_call"
    }
  ],
  "usage": {
    "prompt_tokens": 82,
    "completion_tokens": 18,
    "total_tokens": 100
  }
}
```

Function Calling

```
response_message = response["choices"][0]["message"]
```

```
response_message
```

```
<OpenAIObject at 0x109cf5f90> JSON: {
  "role": "assistant",
  "content": null,
  "function_call": {
    "name": "get_current_weather",
    "arguments": "{\n  \"location\": \"Boston, MA\"\n}"
  }
}
```

```
response_message["content"]
```

```
response_message["function_call"]
```

```
<OpenAIObject at 0x109cf01d0> JSON: {
  "name": "get_current_weather",
  "arguments": "{\n  \"location\": \"Boston, MA\"\n}"
}
```

Function Calling

```
args = json.loads(response_message["function_call"]["arguments"])
```

```
get_current_weather(args)
```

```
'{"location": {"location": "Boston, MA"}, "temperature": "72", "unit":
"fahrenheit", "forecast": ["sunny", "windy"]}'
```

Tools & Routing

- Functions and services an LLM can utilize to extend its capabilities are named “tools” in LangChain
- LangChain has many tools available
  - Search tools
  - Math tools
  - SQL tools

Tools & Routing

```python
from langchain.agents import tool
```

```python
@tool
def search(query: str) -> str:
    """Search for weather online"""
    return "42f"
```

Tools & Routing

```python
from langchain.agents import tool
```

```python
@tool
def search(query: str) -> str:
    """Search for weather online"""
    return "42f"
```

```python
search.name
```

```
'search'
```

```python
search.description
```

```
'search(query: str) -> str - Search for weather online'
```

```python
search.args
```

```
{'query': {'title': 'Query', 'type': 'string'}}
```

Tools & Routing

```python
import requests
from pydantic import BaseModel, Field
import datetime

# Define the input schema
class OpenMeteoInput(BaseModel):
    latitude: float = Field(..., description="Latitude of the loca
    longitude: float = Field(..., description="Longitude of the lo

@tool(args_schema=OpenMeteoInput)
def get_current_temperature(latitude: float, longitude: float) ->
    """Fetch current temperature for given coordinates."""

    BASE_URL = "https://api.open-meteo.com/v1/forecast"

    # Parameters for the request
    params = {
        'latitude': latitude,
        'longitude': longitude,
        'hourly': 'temperature_2m',
        'forecast_days': 1,
    }

    # Make the request
    response = requests.get(BASE_URL, params=params)

    if response.status_code == 200:
        results = response.json()
    else:
        raise Exception(f"API Request failed with status code: {re

    current_utc_time = datetime.datetime.utcnow()
    time_list = [datetime.datetime.fromisoformat(time_str.replace(
    temperature_list = results['hourly']['temperature_2m']

    closest_time_index = min(range(len(time_list)), key=lambda i:
    current_temperature = temperature_list[closest_time_index]

    return f'The current temperature is {current_temperature}°C'
```

Tools & Routing

```python
import wikipedia
@tool
def search_wikipedia(query: str) -> str:
    """Run Wikipedia search and get page summaries."""
    page_titles = wikipedia.search(query)
    summaries = []
    for page_title in page_titles[: 3]:
        try:
            wiki_page =  wikipedia.page(title=page_title, auto_sug
            summaries.append(f"Page: {page_title}\nSummary: {wiki_
        except (
            self.wiki_client.exceptions.PageError,
            self.wiki_client.exceptions.DisambiguationError,
        ):
            pass
    if not summaries:
        return "No good Wikipedia Search Result was found"
    return "\n\n".join(summaries)
```

Tools & Routing

Routing

In lesson 3, we show an example of function calling deciding between two candidate functions.

Given our tools above, let's format these as OpenAI functions and show this same behavior.

In [ ]:
```
functions = [
    format_tool_to_openai_function(f) for f in [
        search_wikipedia, get_current_temperature
    ]
]
model = ChatOpenAI(temperature=0).bind(functions=functions)
```

Tools & Routing

```
model.invoke("what is the weather in sf right now")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'get_
current_temperature', 'arguments': '{\n  "latitude": 37.7749,\n  "longit
ude": -122.4194\n}'}})
```

```
model.invoke("what is langchain")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'sear
ch_wikipedia', 'arguments': '{\n  "query": "langchain"\n}'}})
```

Agents

- Agents
  - are a combination of LLMs and code
  - LLMs reason about what steps to take and call for actions.
- Agent loop
  - Choose a tool to use
  - Observe the output of the tool
  - Repeat until a stopping condition is met
- Stopping conditions can be:
  - LLM determined
  - Hardcoded rules

Agents

```
from langchain.schema.agent import AgentFinish
def run_agent(user_input):
    intermediate_steps = []
    while True:
        result = chain.invoke({
            "input": user_input,
            "agent_scratchpad": format_to_openai_functions(interme
        })
        if isinstance(result, AgentFinish):
            return result
        tool = {
            "search_wikipedia": search_wikipedia,
            "get_current_temperature": get_current_temperature,
        }[result.tool]
        observation = tool.run(result.tool_input)
        intermediate_steps.append((result, observation))
```

Agents

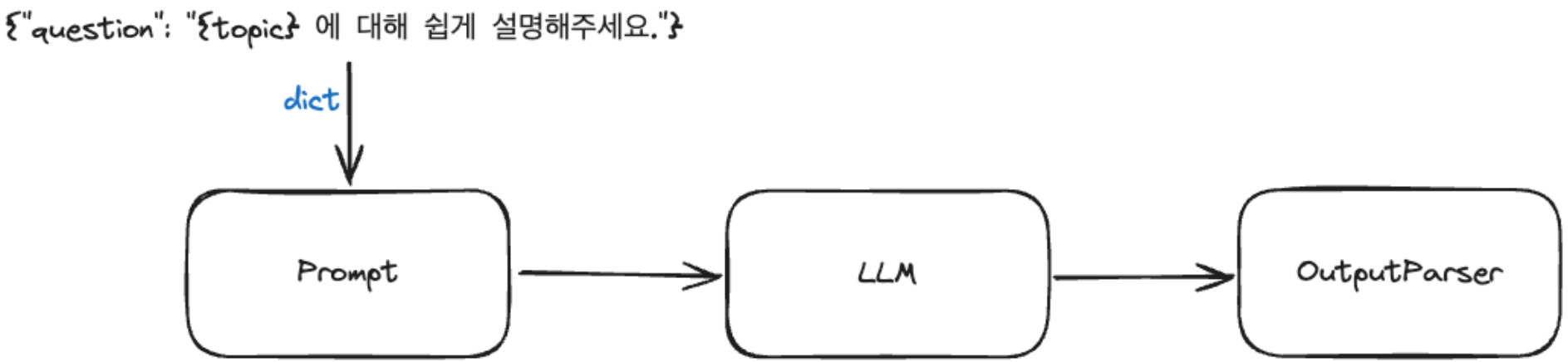
```
agent_executor = AgentExecutor(agent=agent_chain, tools=tools, ver
```

```
agent_executor.invoke({"input": "my name is bob"})
```

```
agent_executor.invoke({"input": "whats my name"})
```

```
agent_executor.invoke({"input": "whats the weather in sf?"})
```

LangChain Expression Language (LCEL)



LangChain Expression Language (LCEL)

Without LCEL

```
from typing import List

import openai


prompt_template = "Tell me a short joke about {topic}"
client = openai.OpenAI()

def call_chat_model(messages: List[dict]) -> str:
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
    )
    return response.choices[0].message.content

def invoke_chain(topic: str) -> str:
    prompt_value = prompt_template.format(topic=topic)
    messages = [{"role": "user", "content": prompt_value}]
    return call_chat_model(messages)

invoke_chain("ice cream")
```

LangChain Expression Language (LCEL)

Without LCEL

```python
from typing import List

import openai


prompt_template = "Tell me a short joke about {topic}"
client = openai.OpenAI()

def call_chat_model(messages: List[dict]) -> str:
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
    )
    return response.choices[0].message.content

def invoke_chain(topic: str) -> str:
    prompt_value = prompt_template.format(topic=topic)
    messages = [{"role": "user", "content": prompt_value}]
    return call_chat_model(messages)

invoke_chain("ice cream")
```

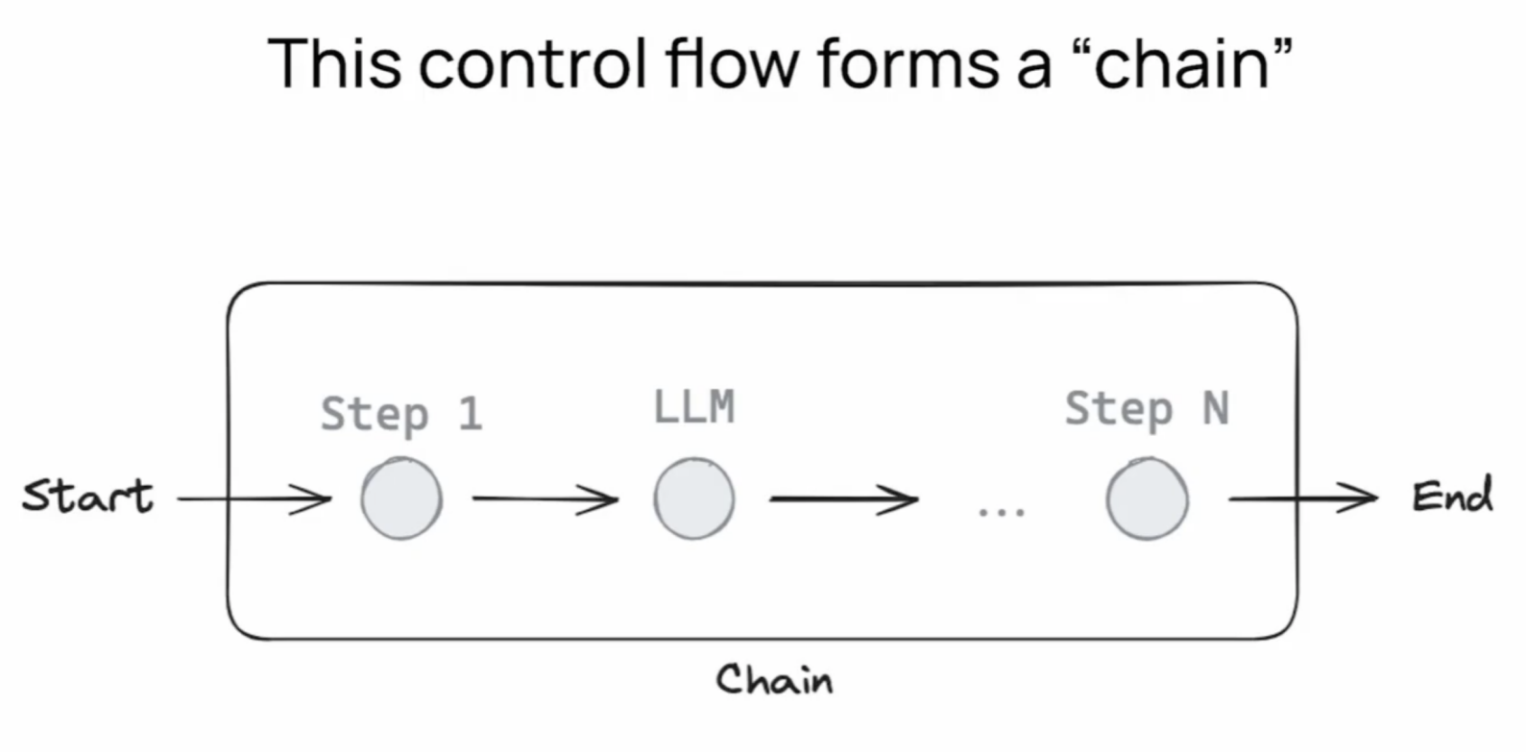
LCEL

```python
from langchain_core.runnables import RunnablePassthrough


prompt = ChatPromptTemplate.from_template(
    "Tell me a short joke about {topic}"
)
output_parser = StrOutputParser()
model = ChatOpenAI(model="gpt-3.5-turbo")
chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | model
    | output_parser
)

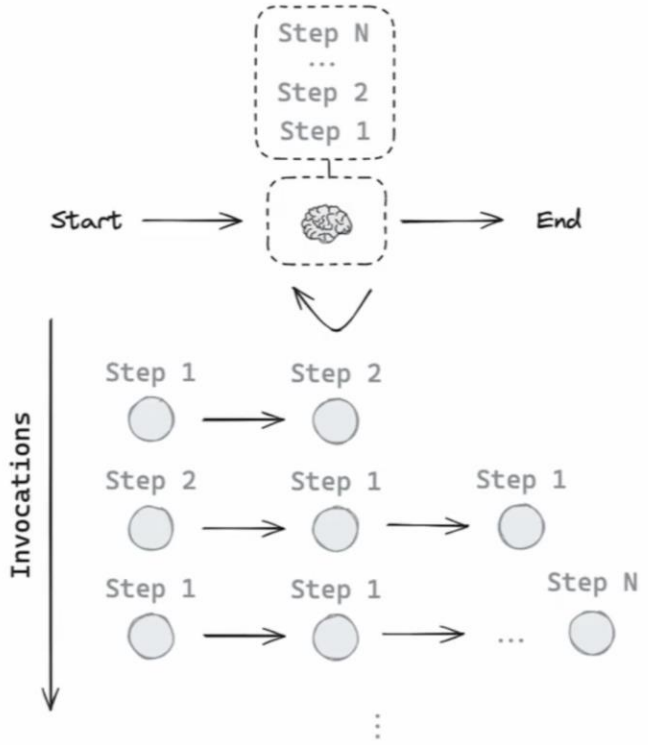
chain.invoke("ice cream")
```

LangGraph – Motivation

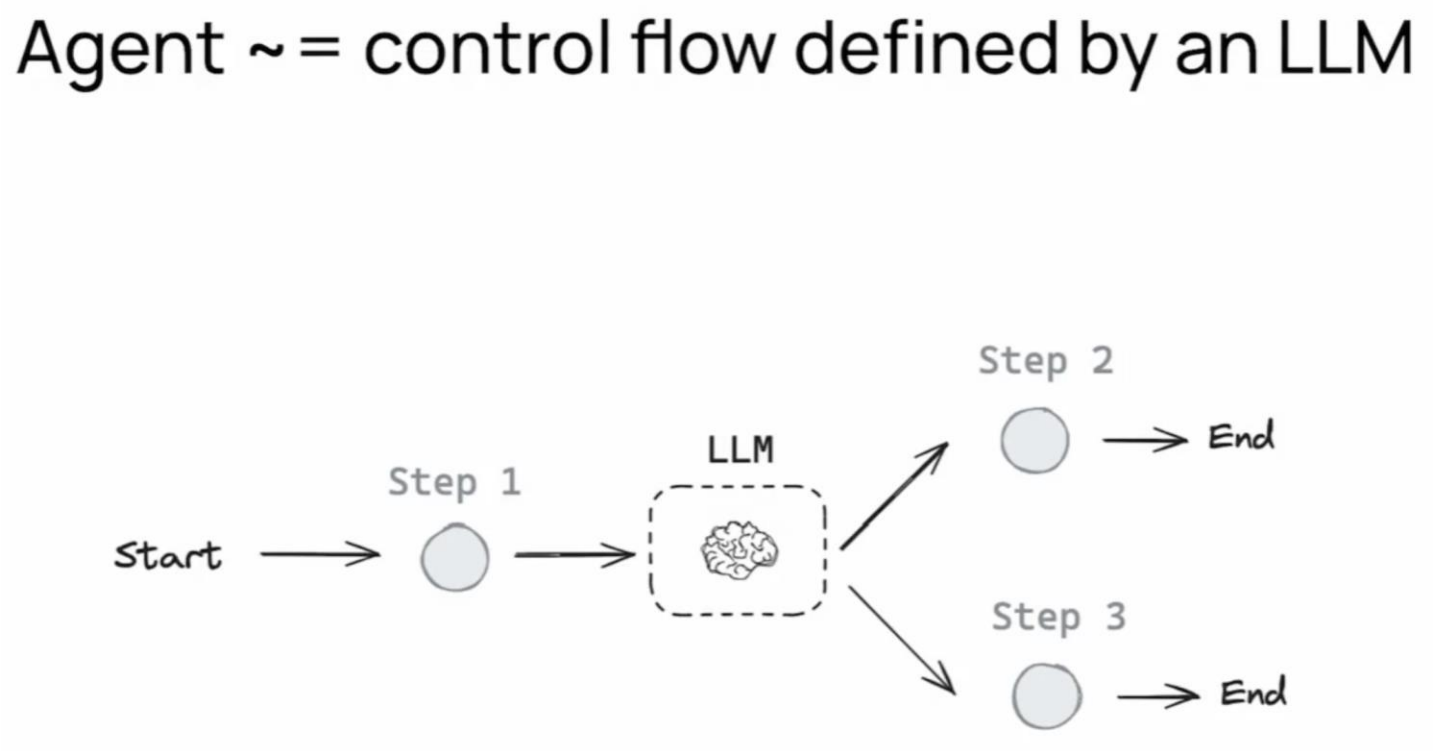


LangGraph – Motivation

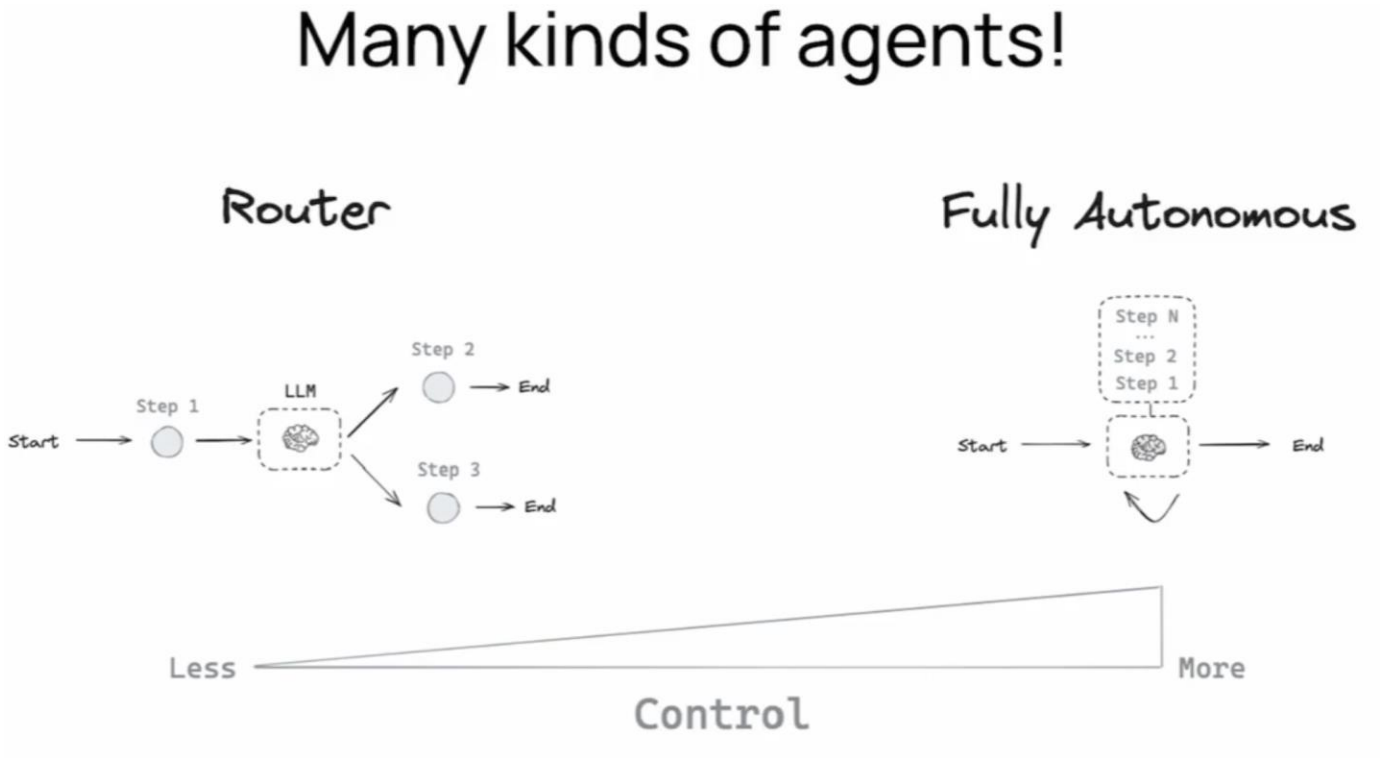
But, we want LLM systems that can pick their own control flow!



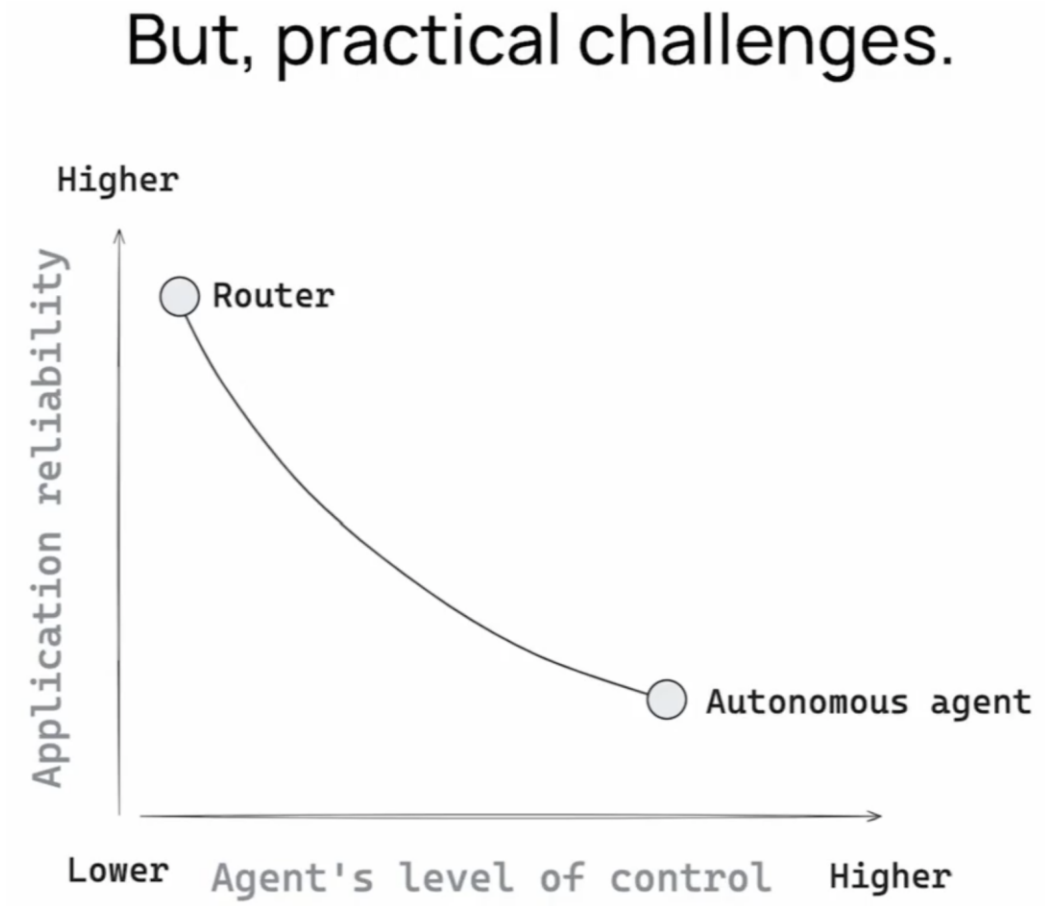
LangGraph – Motivation



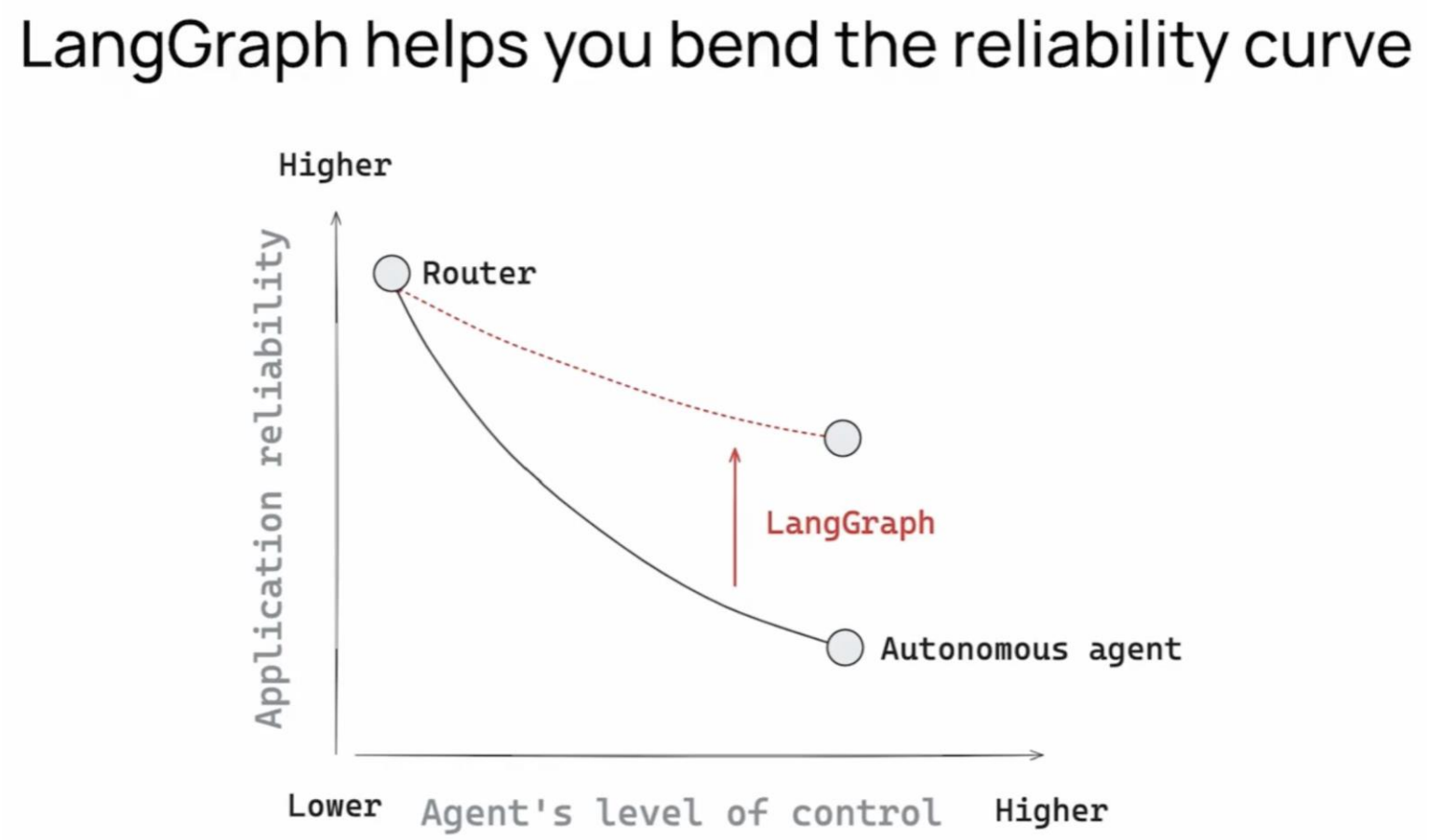
LangGraph – Motivation



LangGraph – Motivation

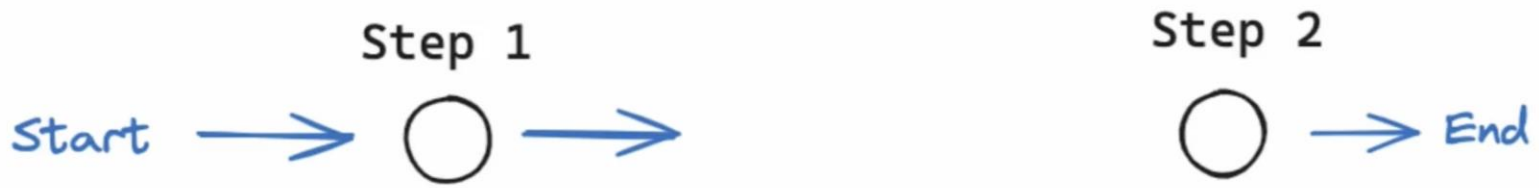


LangGraph – Motivation



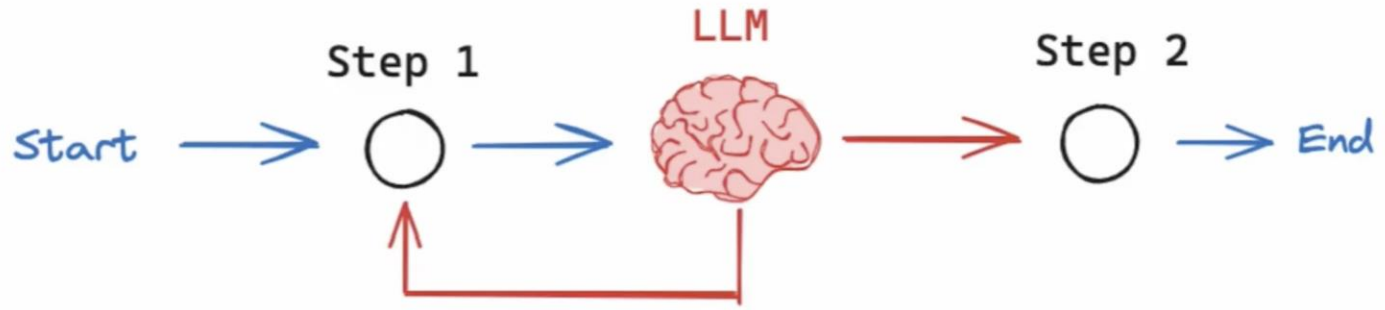
LangGraph – Overview

Intuition: Let developer set parts of control flow (reliable)

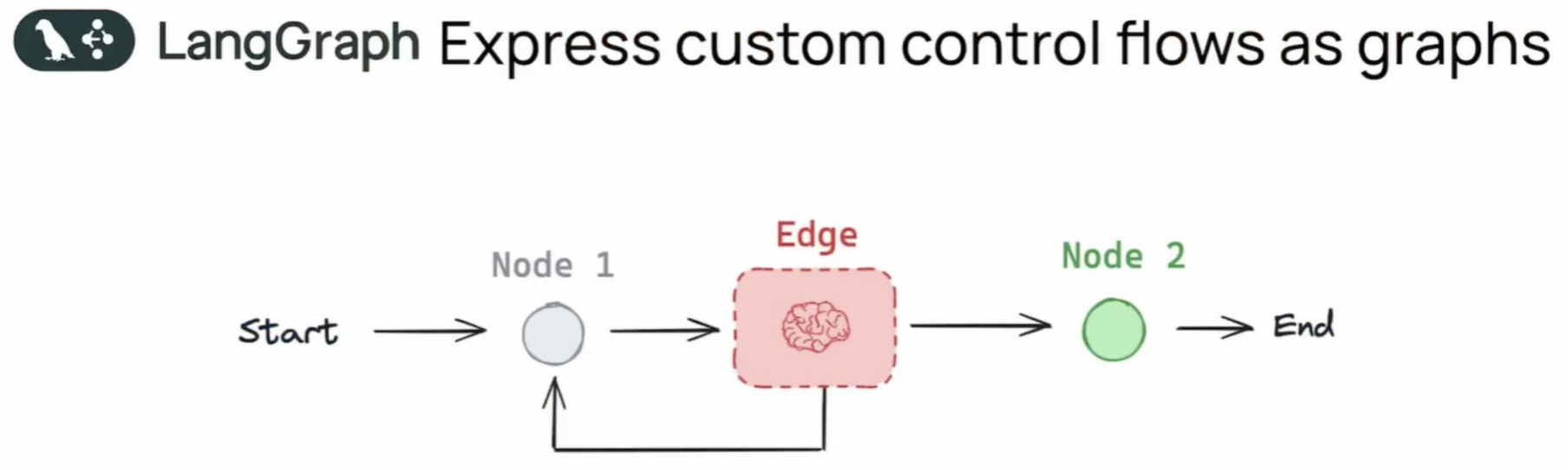


LangGraph – Overview

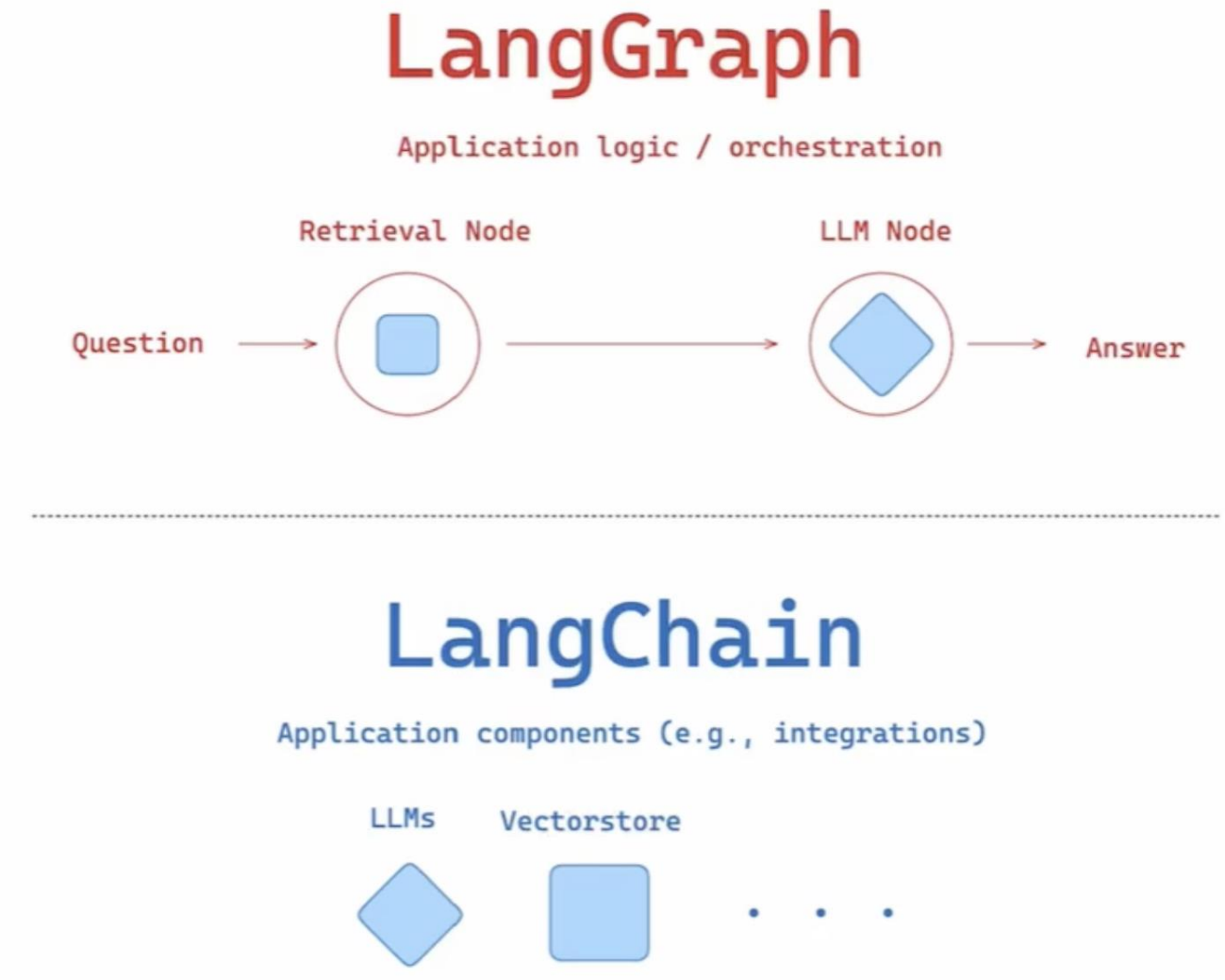
Intuition: Inject LLM to make it an agent (control)



LangGraph – Overview



LangGraph 🤝 LangChain



LangGraph – Example

```python
from IPython.display import Image, display
from langgraph.graph import StateGraph, START, END

# Build graph
builder = StateGraph(State)
builder.add_node("node_1", node_1)
builder.add_node("node_2", node_2)
builder.add_node("node_3", node_3)

# Logic
builder.add_edge(START, "node_1")
builder.add_conditional_edges("node_1", decide_mood)
builder.add_edge("node_2", END)
builder.add_edge("node_3", END)

# Add
graph = builder.compile()

# View
display(Image(graph.get_graph().draw_mermaid_png()))
```

LLMOps: LangSmith

