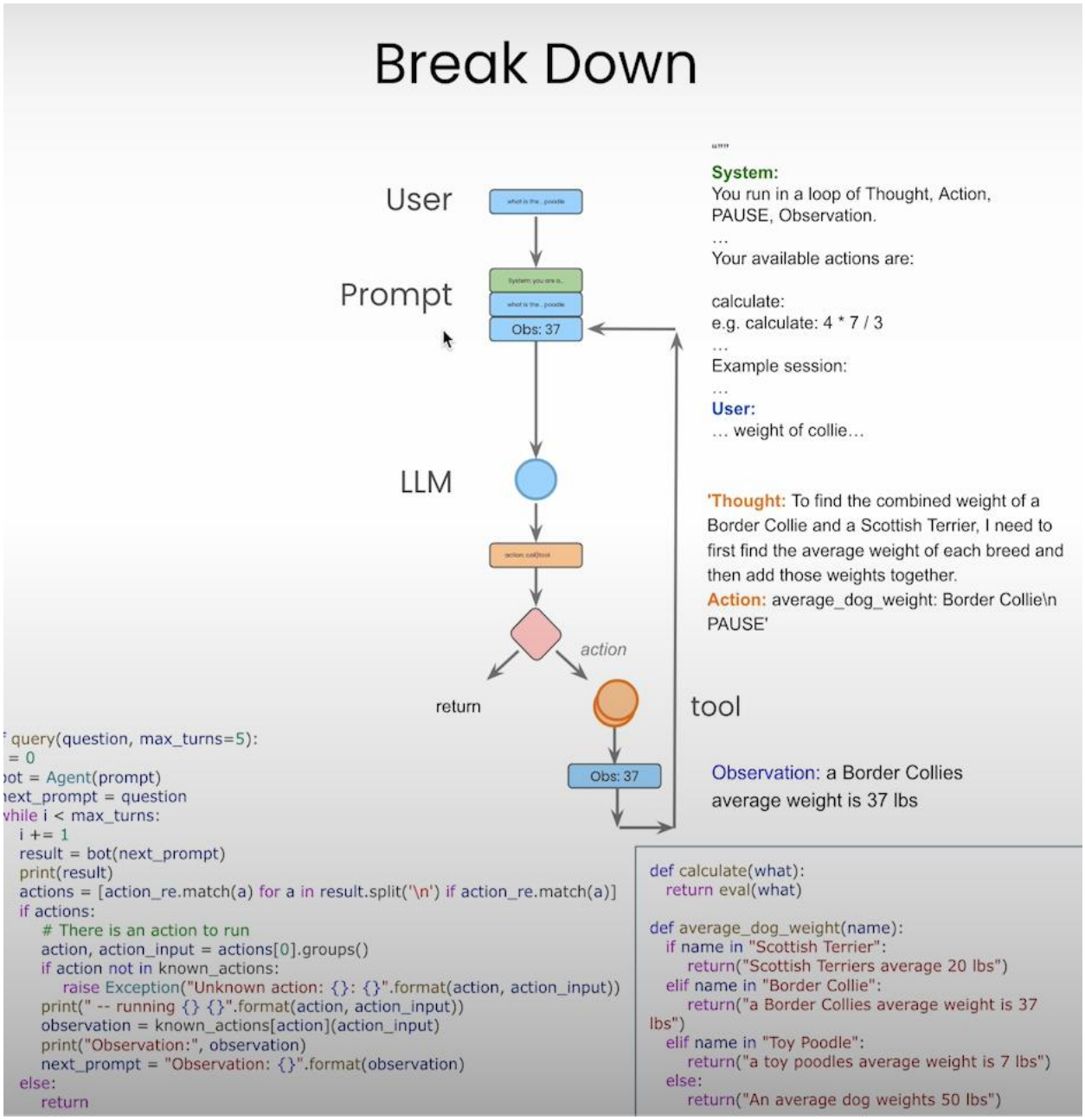
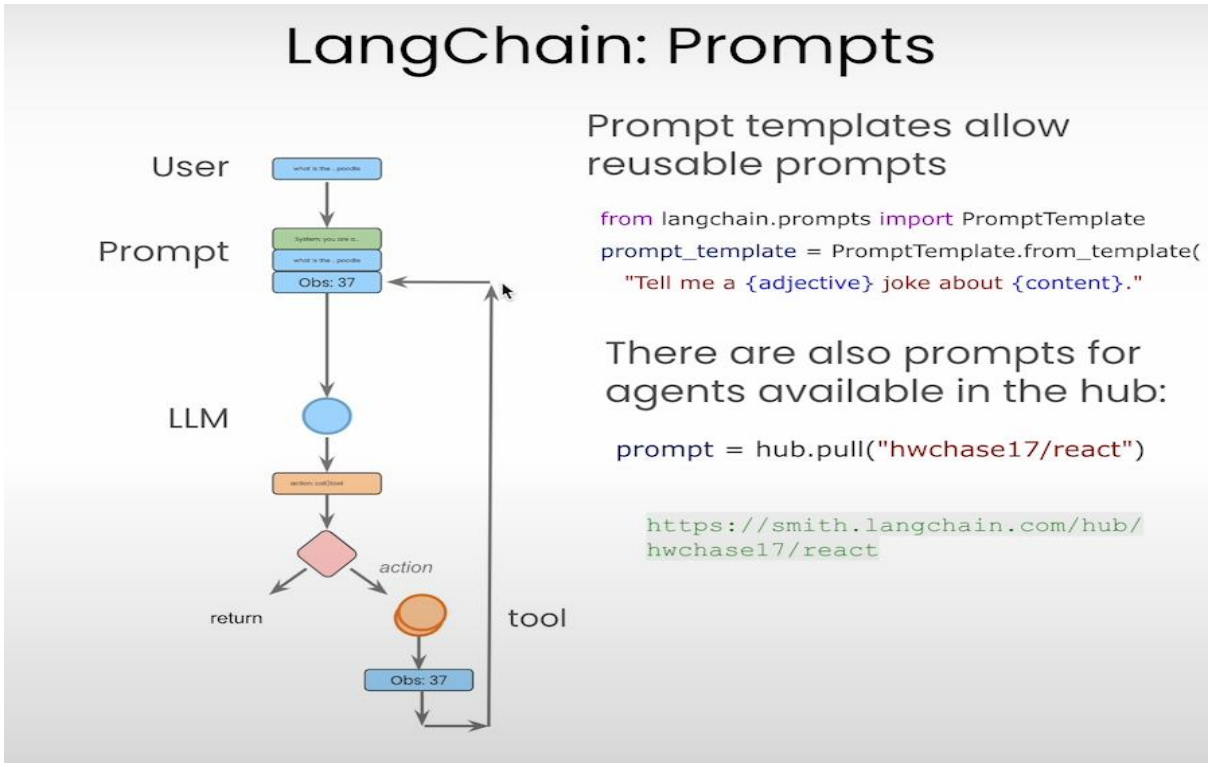
Lang graph:

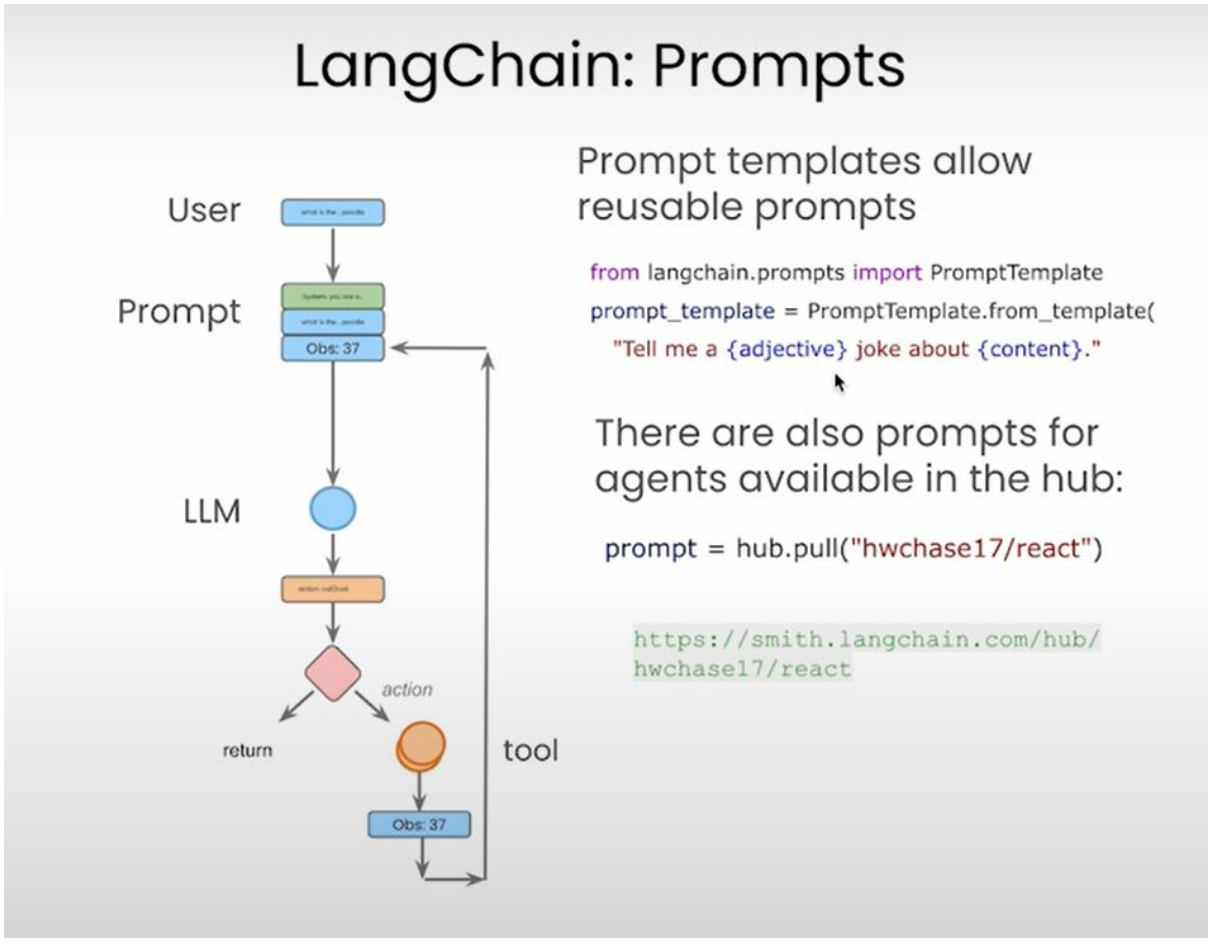
In my previous post :

https://www.linkedin.com/posts/dsp1729_ai-machinelearning-openai-activity-7205741303840608256-F60Q?utm_source=share&utm_medium=member_desktop

I built an Agent from Scratch using only Python and LLM but now we will implement the same using Lan graph



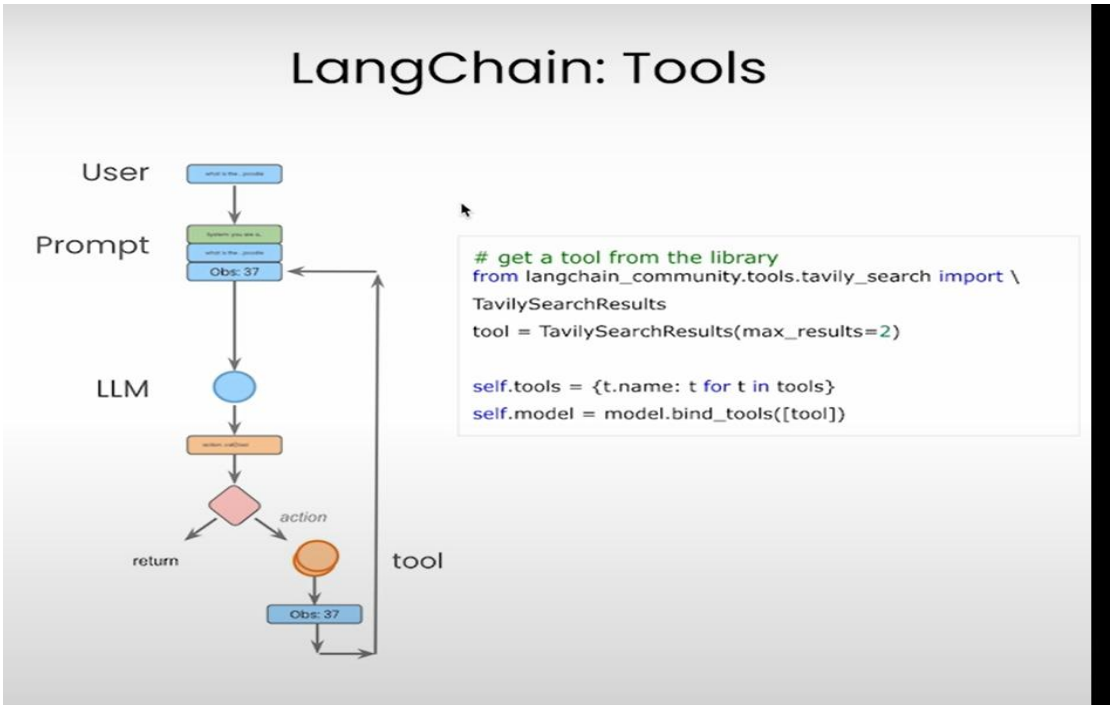




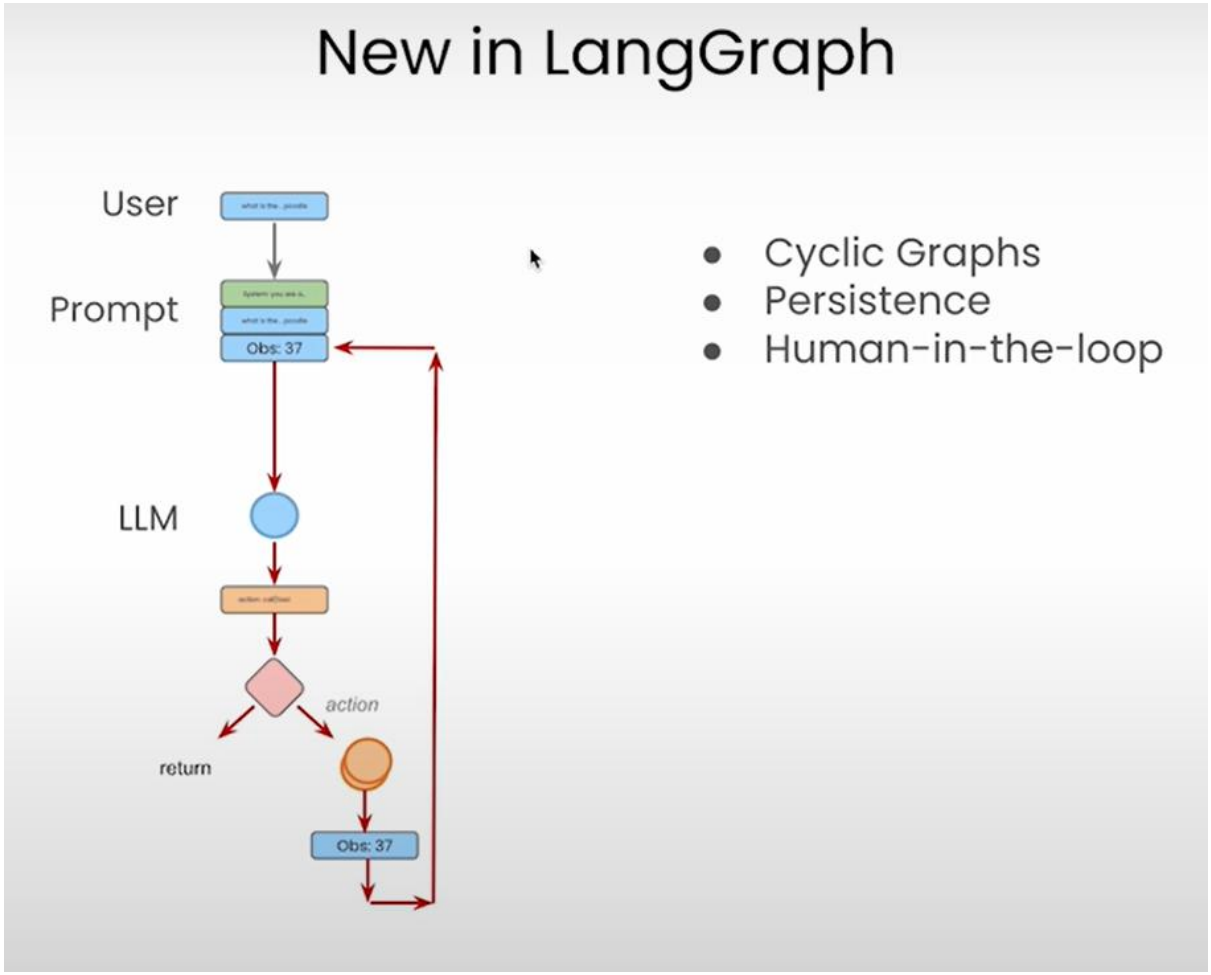
Prompt Templates are reusable components to use and we can format them however we want

Here are some of the examples:

https://smith.langchain.com/hub/hwchase17/react



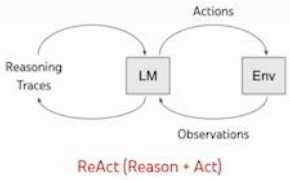
Tavily Tool for Search

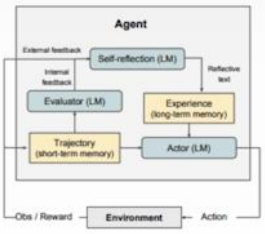


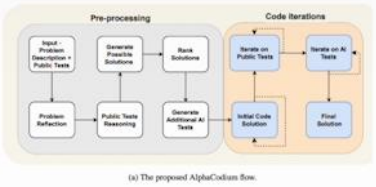
Lan graph helps you to strive and orchestrate.

Lang graph helps to describe and Orchestrate the Control Flow. It allows to create Cyclic Graphs.

Graphs







(a) The proposed AlphaCodium flow.

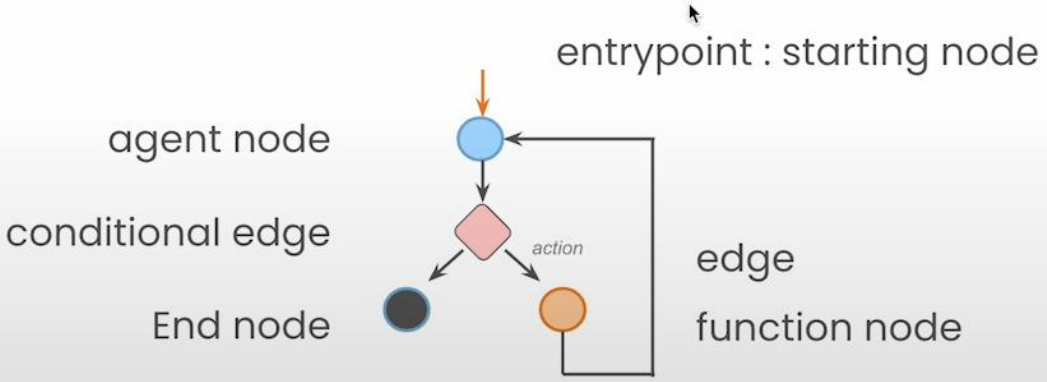
- LangGraph is an extension of LangChain that supports graphs.
- Single and Multi-agent flows are described and represented as graphs.
- Allows for extremely controlled “flows”
- Built-in persistence allows for human-in-the-loop

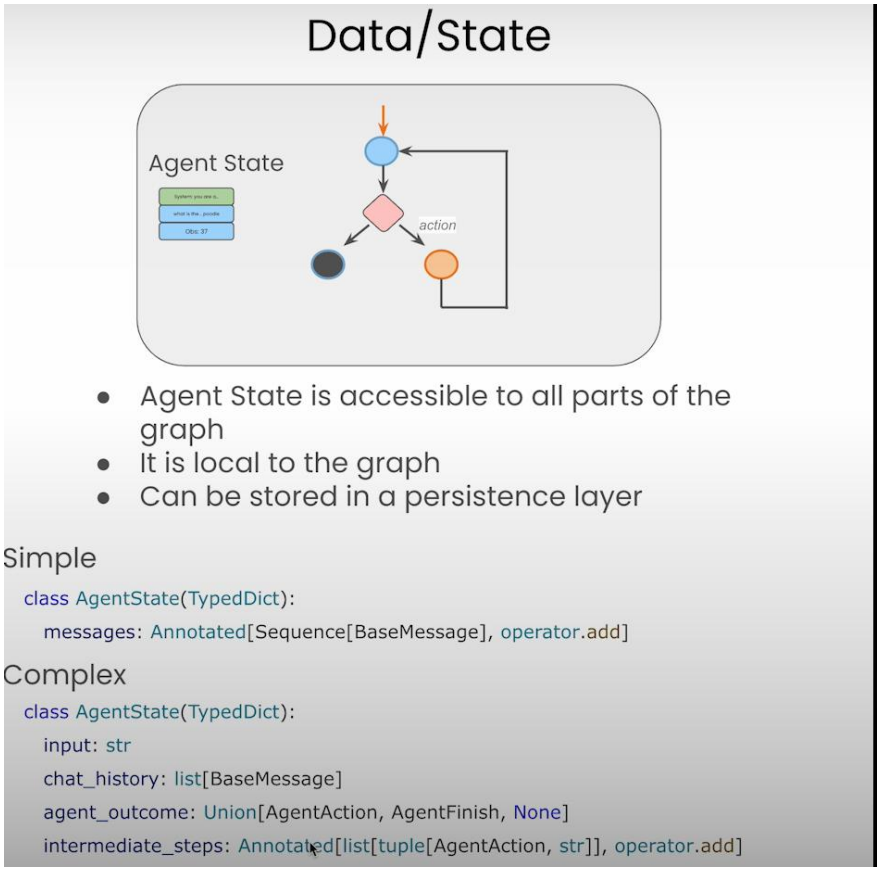
Graphs

Nodes: Agents or functions

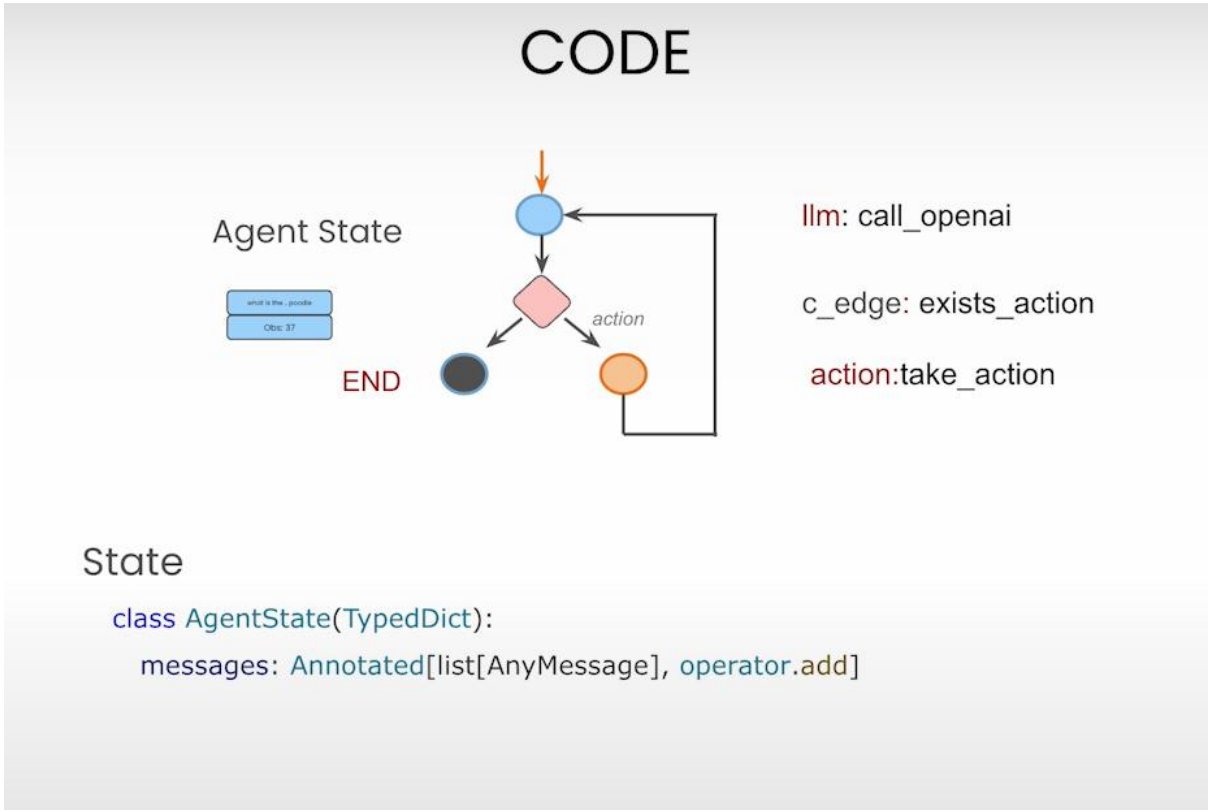
Edges: connect nodes

Conditional edges: decisions





Here intermediate_steps are annotated



Tavily Search

```python
class Agent:

    def __init__(self, model, tools, system=""):
        self.system = system
        graph = StateGraph(AgentState)
        graph.add_node("llm", self.call_openai)
        graph.add_node("action", self.take_action)
        graph.add_conditional_edges(
            "llm",
            self.exists_action,
            {True: "action", False: END}
        )
        graph.add_edge("action", "llm")
        graph.set_entry_point("llm")
        self.graph = graph.compile()
        self.tools = {t.name: t for t in tools}
        self.model = model.bind_tools(tools)

    def exists_action(self, state: AgentState):
        result = state['messages'][-1]
        return len(result.tool_calls) > 0

    def call_openai(self, state: AgentState):
        messages = state['messages']
        if self.system:
            messages = [SystemMessage(content=self.system)] + messages
        message = self.model.invoke(messages)
        return {'messages': [message]}

    def take_action(self, state: AgentState):
        tool_calls = state['messages'][-1].tool_calls
        results = []
        for t in tool_calls:
```

```
    def take_action(self, state: AgentState):
        tool_calls = state['messages'][-1].tool_calls
        results = []
        for t in tool_calls:
            print(f"Calling: {t}")
            result = self.tools[t['name']].invoke(t['args'])
            results.append(ToolMessage(tool_call_id=t['id'], name=t['nam
        print("Back to the model!")
        return {'messages': results}
```

```
prompt = """You are a smart research assistant. Use the search engine to
You are allowed to make multiple calls (either together or in sequence).
Only look up information when you are sure of what you want. \
If you need to look up some information before asking a follow up questi
"""
model = ChatOpenAI(model="gpt-4-turbo")
abot = Agent(model, [tool], system=prompt)
```

Let's visualize the graph we had created:

```
from IPython.display import Image

Image(abot.graph.get_graph().draw_png())
```