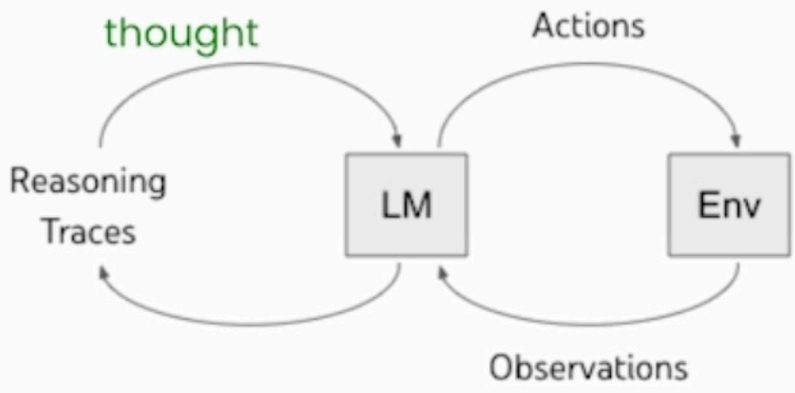
Let's build an agent from scratch



ReAct (Reason + Act)

Published as a conference paper at ICLR 2023

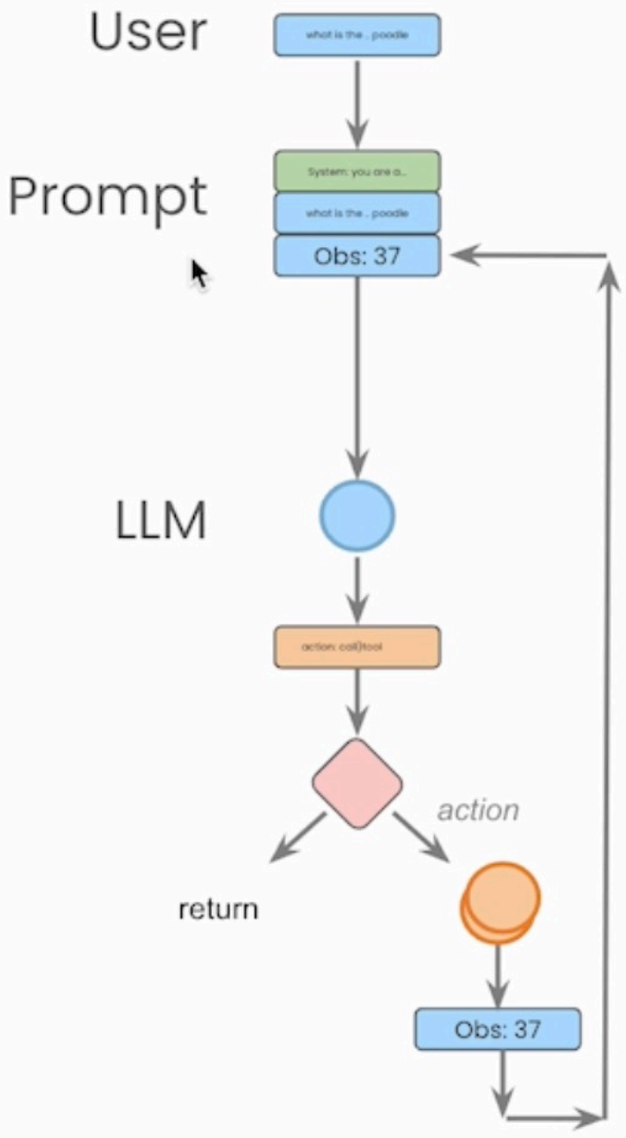
REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao[*,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

https://arxiv.org/pdf/2210.03629.pdf

Break Down



"""

System:
You run in a loop of Thought, Action, PAUSE, Observation.
…
Your available actions are:

calculate:
e.g. calculate: 4 * 7 / 3
…
Example session:
…
User:
… weight of collie…

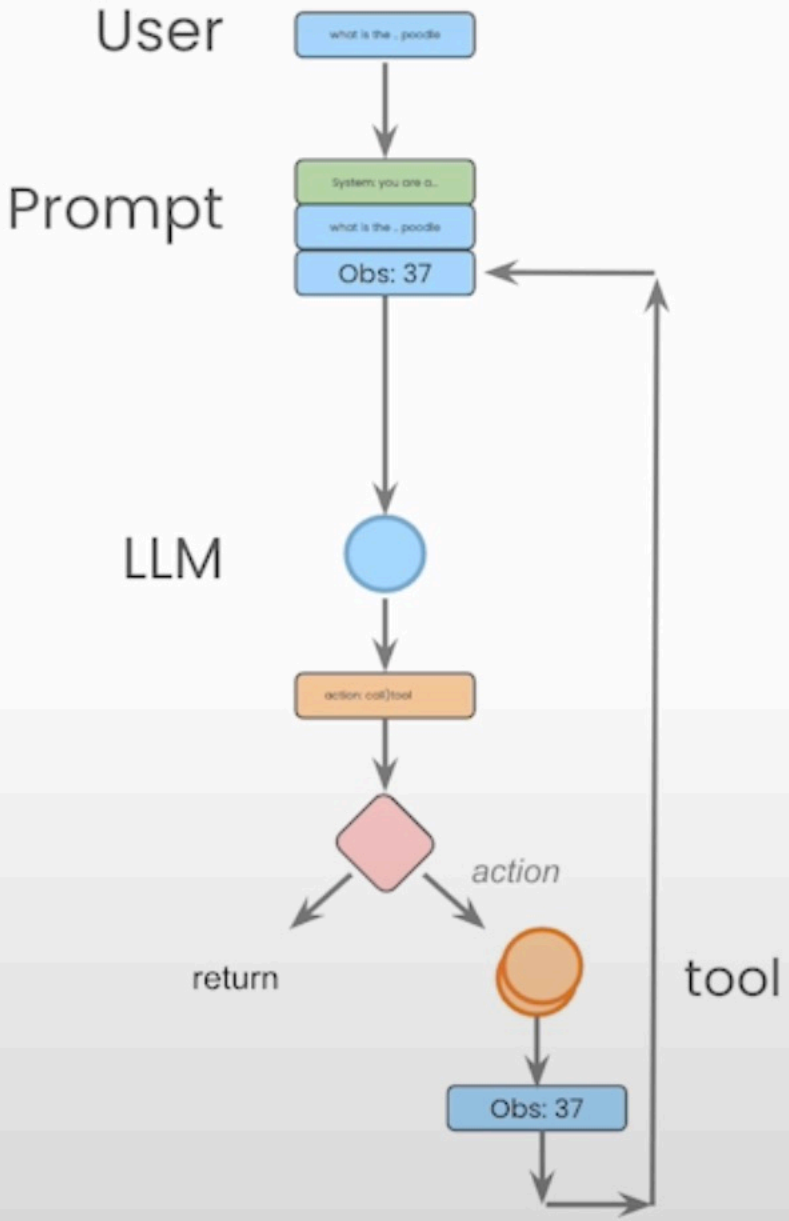
'Thought: To find the combined weight of a Border Collie and a Scottish Terrier, I need to first find the average weight of each breed and then add those weights together.
Action: average_dog_weight: Border Collie\n PAUSE'

Observation: a Border Collies average weight is 37 lbs

```
def query(question, max_turns=5):
  i = 0
  bot = Agent(prompt)
  next_prompt = question
  while i < max_turns:
      i += 1
      result = bot(next_prompt)
      print(result)
      actions = [action_re.match(a) for a in result.split('\n') if action_re.match(a)]
      if actions:
          # There is an action to run
          action, action_input = actions[0].groups()
          if action not in known_actions:
              raise Exception("Unknown action: {}: {}".format(action, action_input))
          print(" -- running {} {}".format(action, action_input))
          observation = known_actions[action](action_input)
          print("Observation:", observation)
          next_prompt = "Observation: {}".format(observation)
      else:
          return
```

```
def calculate(what):
    return eval(what)

def average_dog_weight(name):
    if name in "Scottish Terrier":
        return("Scottish Terriers average 20 lbs")
    elif name in "Border Collie":
        return("a Border Collies average weight is 37
lbs")
    elif name in "Toy Poodle":
        return("a toy poodles average weight is 7 lbs")
    else:
        return("An average dog weights 50 lbs")
```

LangChain: Prompts



Prompt templates allow reusable prompts

```
from langchain.prompts import PromptTemplate
prompt_template = PromptTemplate.from_template(
  "Tell me a {adjective} joke about {content}."
```

There are also prompts for agents available in the hub:

```
prompt = hub.pull("hwchase17/react")
```

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

hwchase17/react

Public

Try it

Prompt Commits

Updated 7 months ago • 15 • 24.2k • 663k

PromptTemplate

Edit in Playground

Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:{agent_scratchpad}

Use object in LangChain

Python SDK

```
from langchain import hub
prompt = hub.pull("hwchase17/react")
```

Copy

DETAILS

TYPE

StringPromptTemplate

COMMITS

1 commit

Commits

d15fe3c4 7 mo
24211 views • 663870 downloads

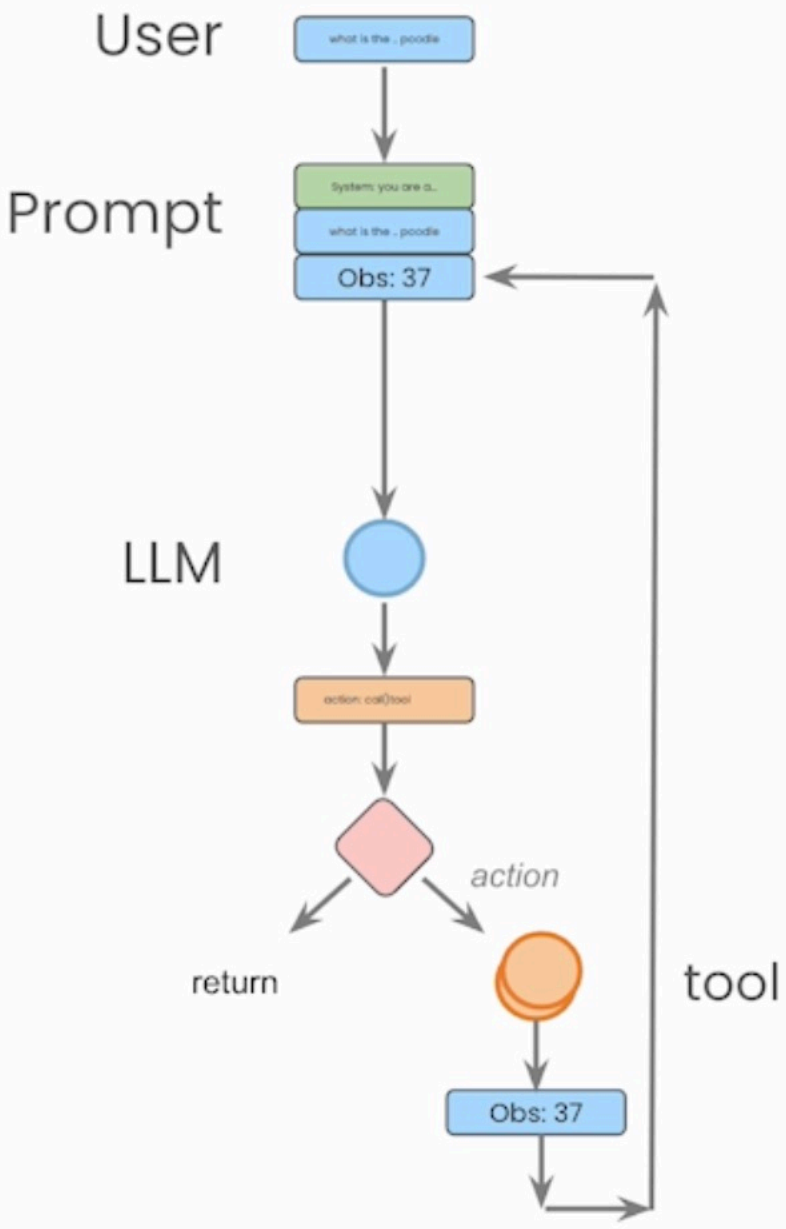
Comments (1)

Matteo Notaro 2 months

i've a problem with this prompt. If used with a multi-input tools it does pass only one input. It's by design this way?

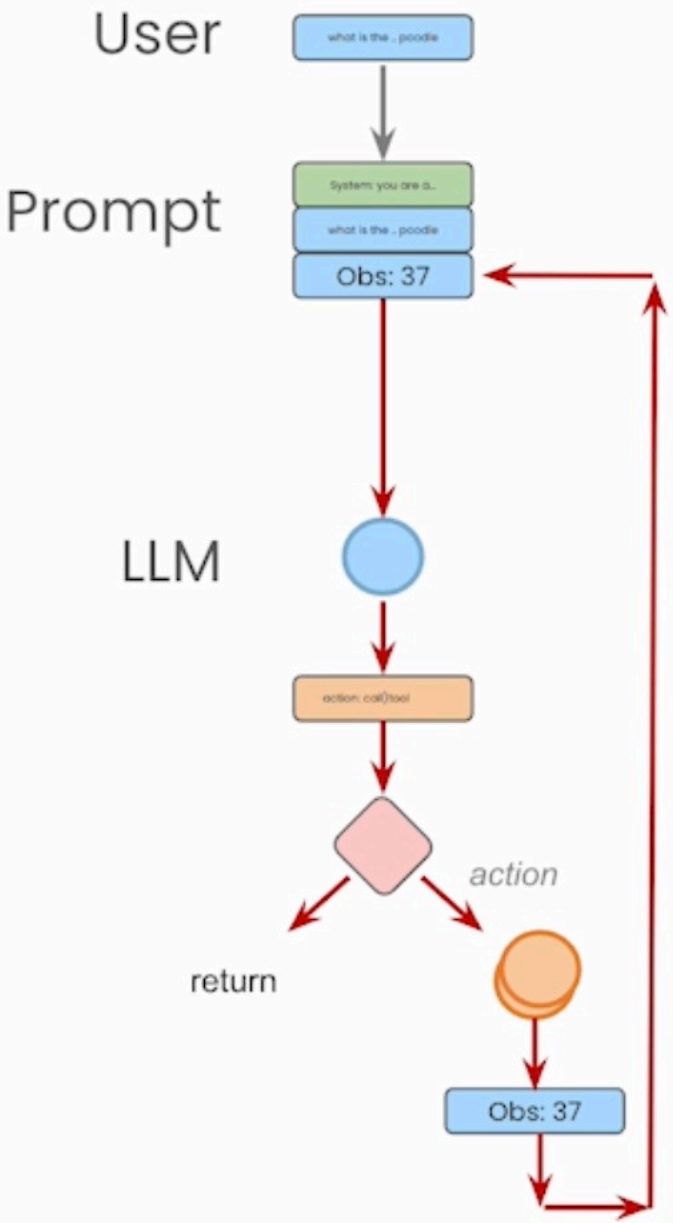
2 replies • 0 likes

LangChain: Tools



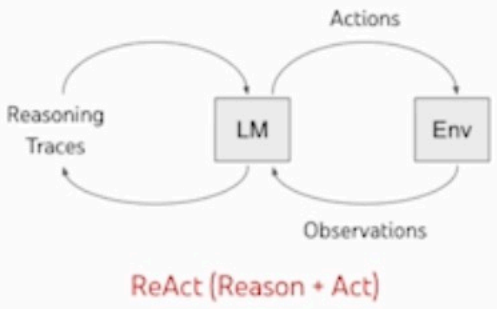

```python
# get a tool from the library
from langchain_community.tools.tavily_search import \
TavilySearchResults
tool = TavilySearchResults(max_results=2)


self.tools = {t.name: t for t in tools}
self.model = model.bind_tools([tool])
```

New in LangGraph

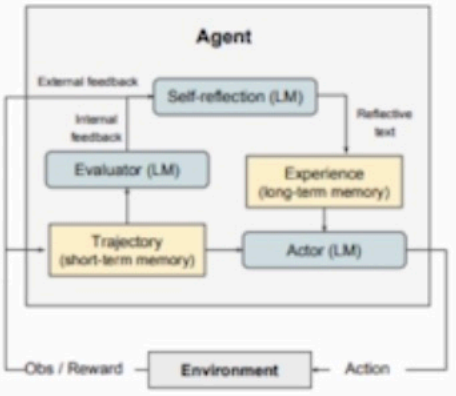


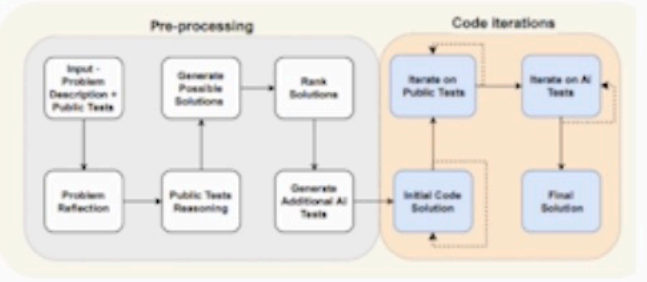
- Cyclic Graphs
- Persistence
- Human-in-the-loop

Graphs



ReAct (Reason + Act)





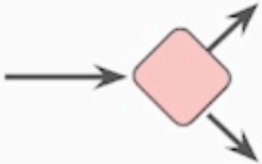
(a) The proposed AlphaCodium flow.

- 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 workflows

Graphs

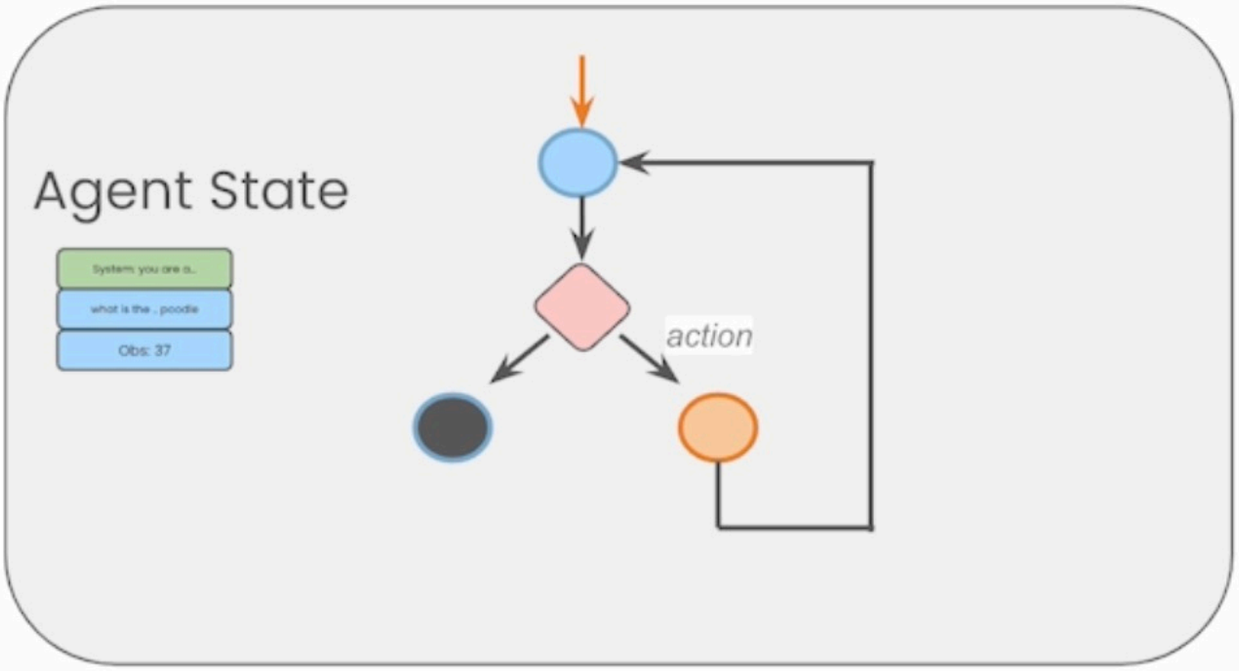
Nodes: Agents or functions

Edges: connect nodes



Conditional edges: decisions

Data/State



- Agent State is accessible to all parts of the graph
- It is local to the graph
- Can be stored in a persistence layer

Simple

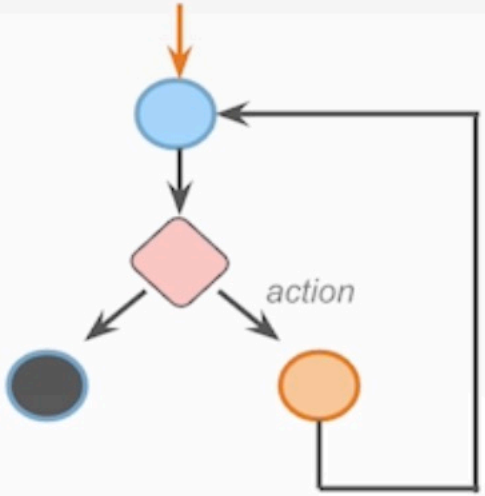
```
class AgentState(TypedDict):
    messages: Annotated[Sequence[BaseMessage], operator.add]
```

Complex

```
class AgentState(TypedDict):
    input: str
    chat_history: list[BaseMessage]
    agent_outcome: Union[AgentAction, AgentFinish, None]
    intermediate_steps: Annotated[list[tuple[AgentAction, str]], operator.add]
```

CODE

Agent State



llm: call_openai

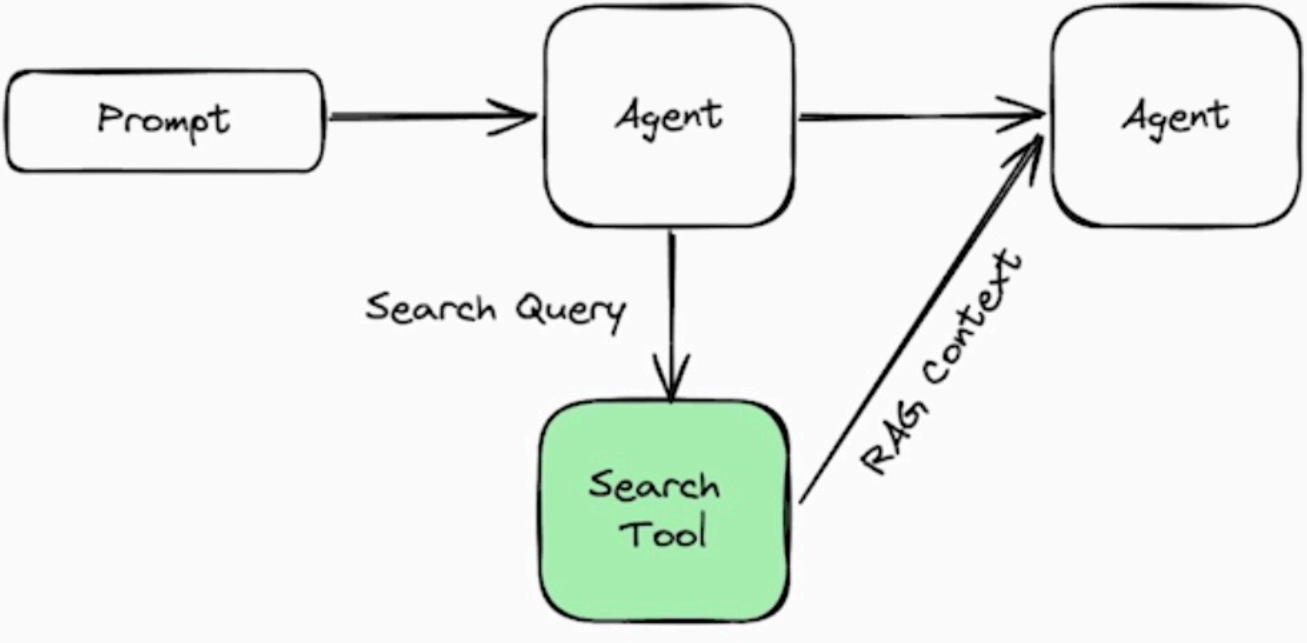
c_edge: exists_action

action:take_action

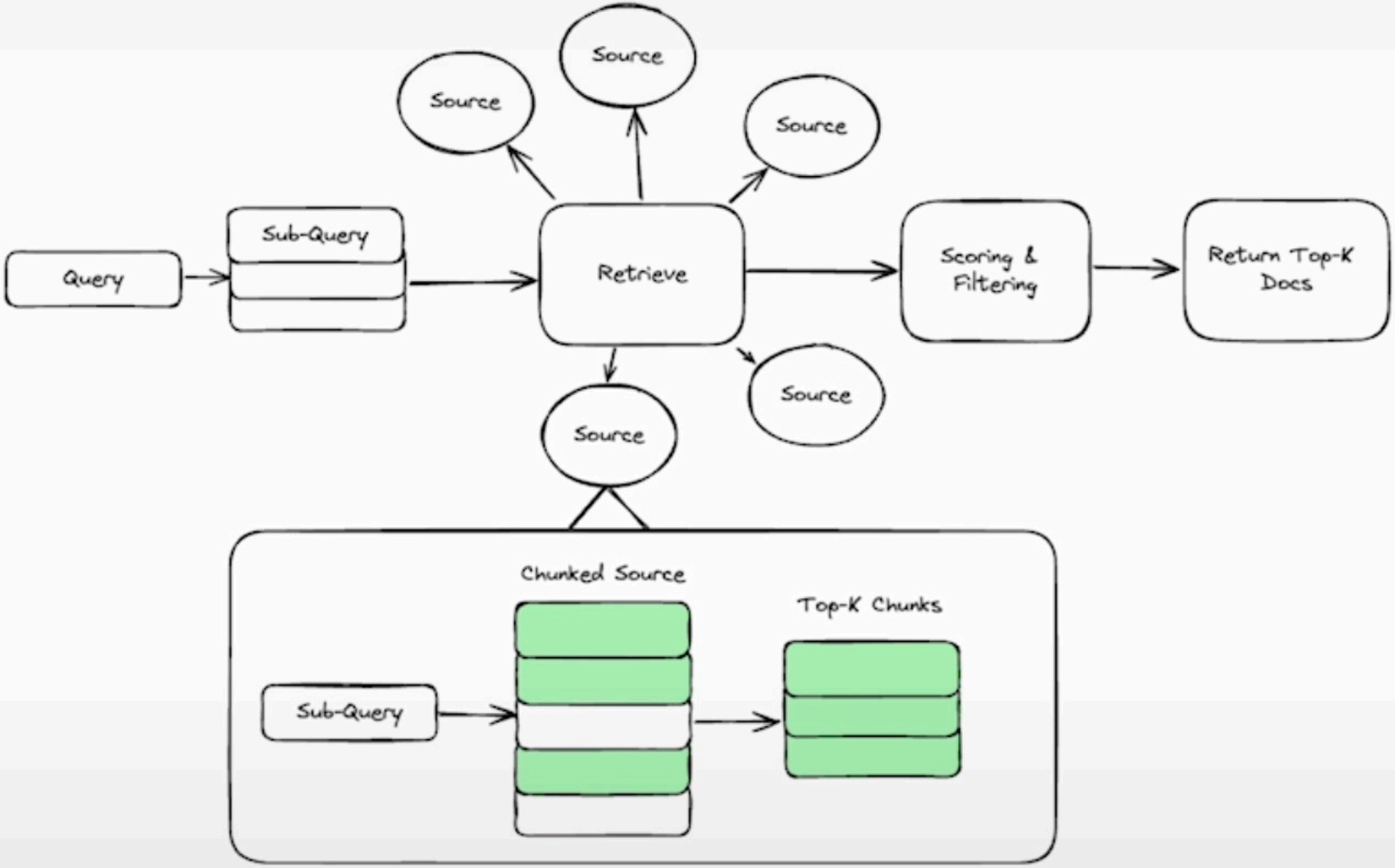
State

```
class AgentState(TypedDict):
    messages: Annotated[list[AnyMessage], operator.add]
```

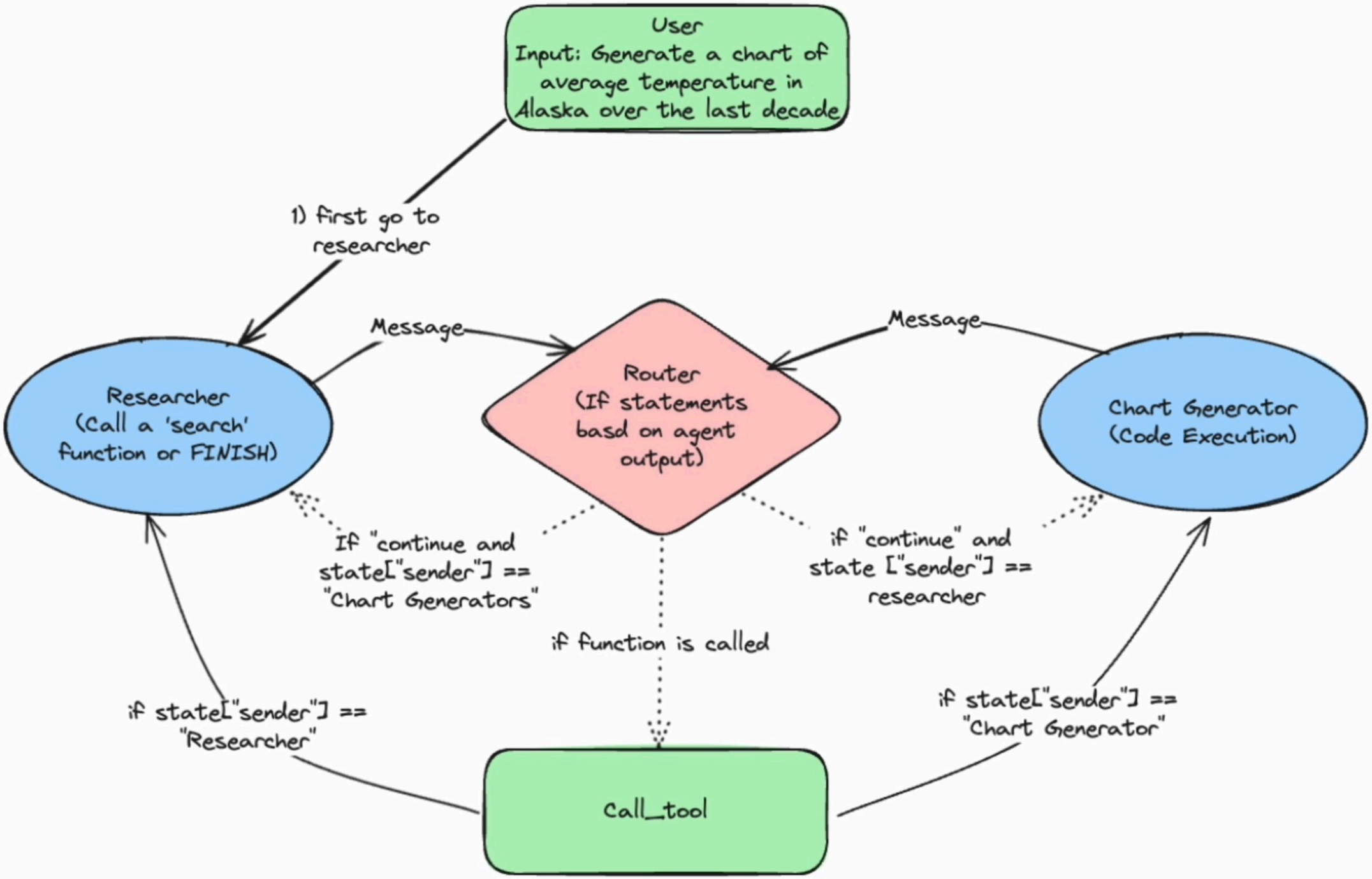
Why Search Tool



Inside a Search Tool

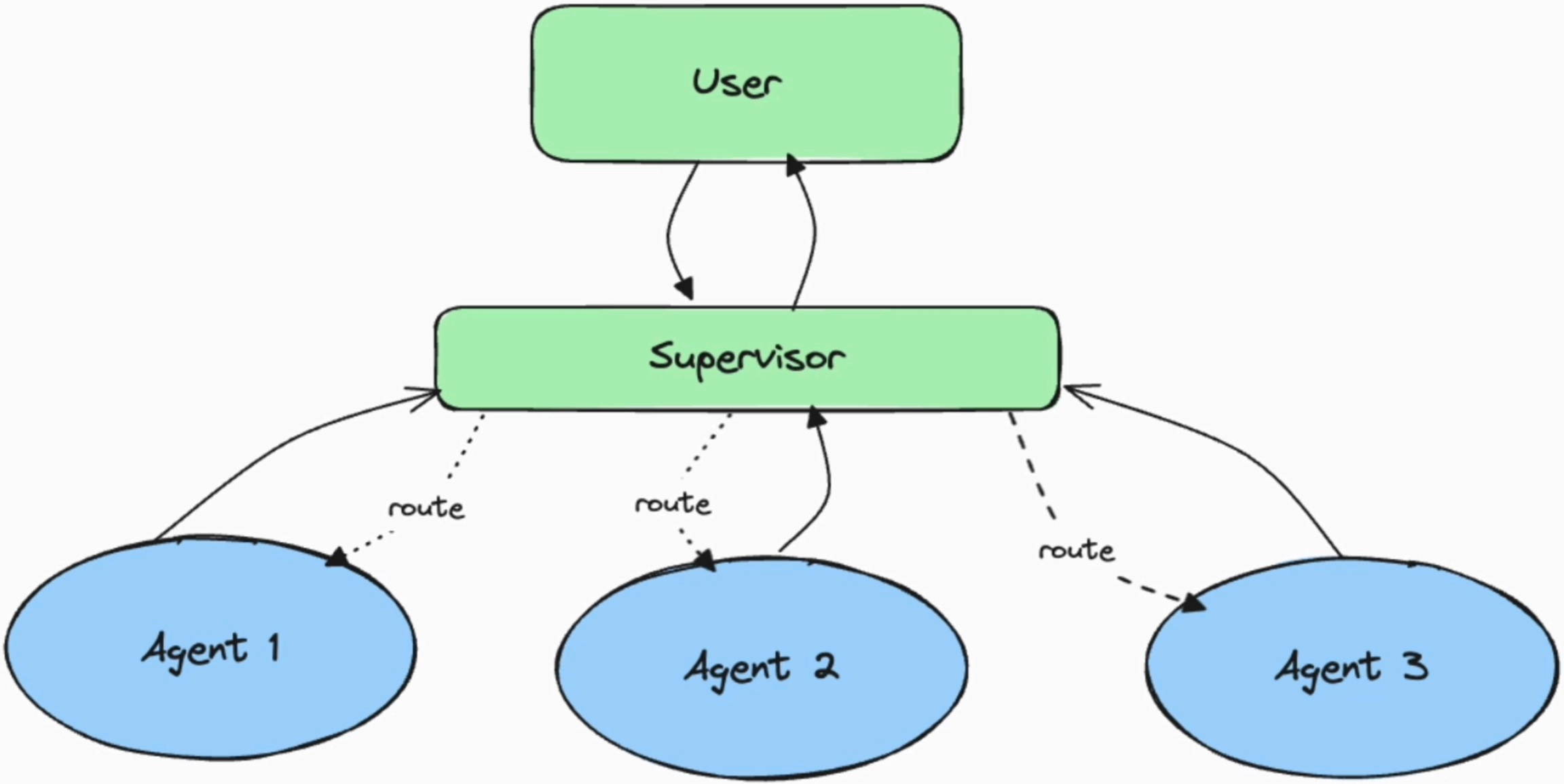


Multi-Agent



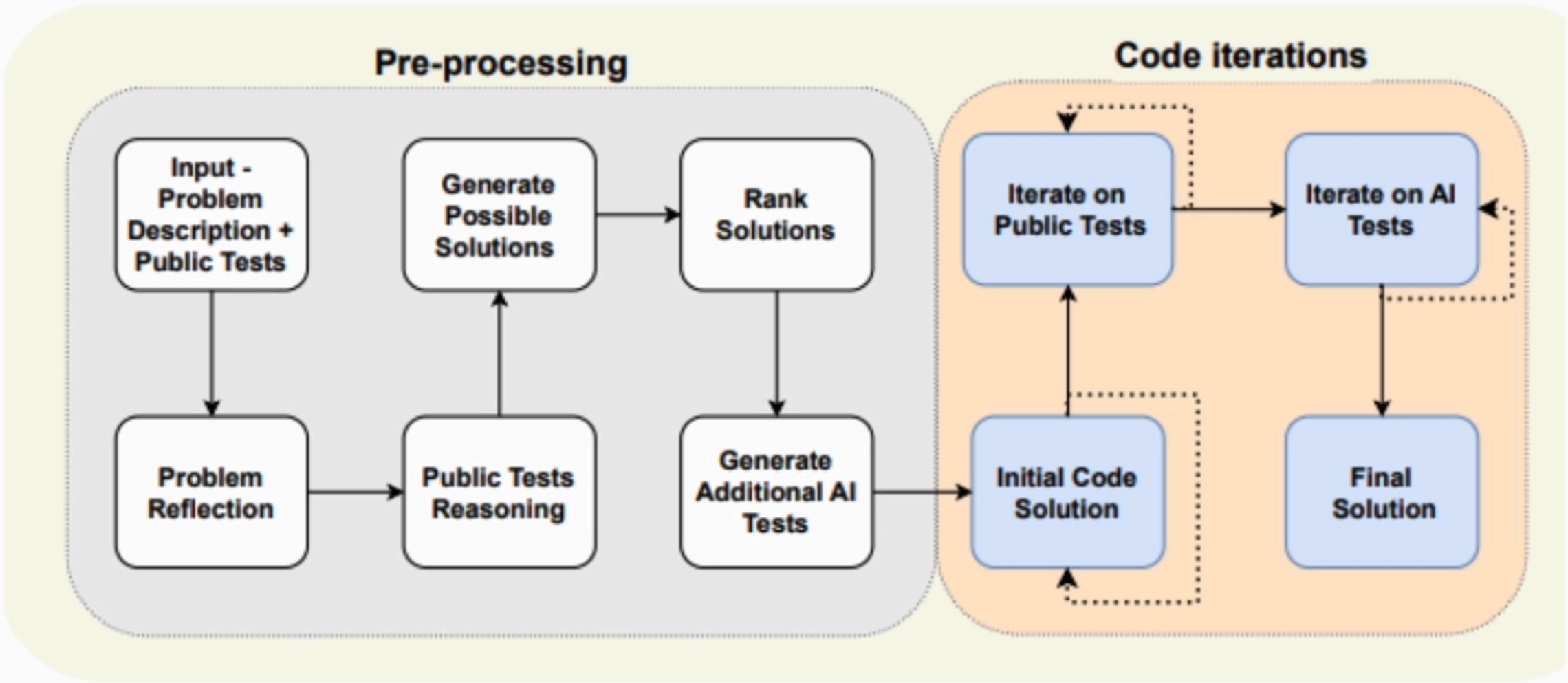
https://github.com/langchain-ai/langgraph/tree/main/examples/multi_agent

Supervisor



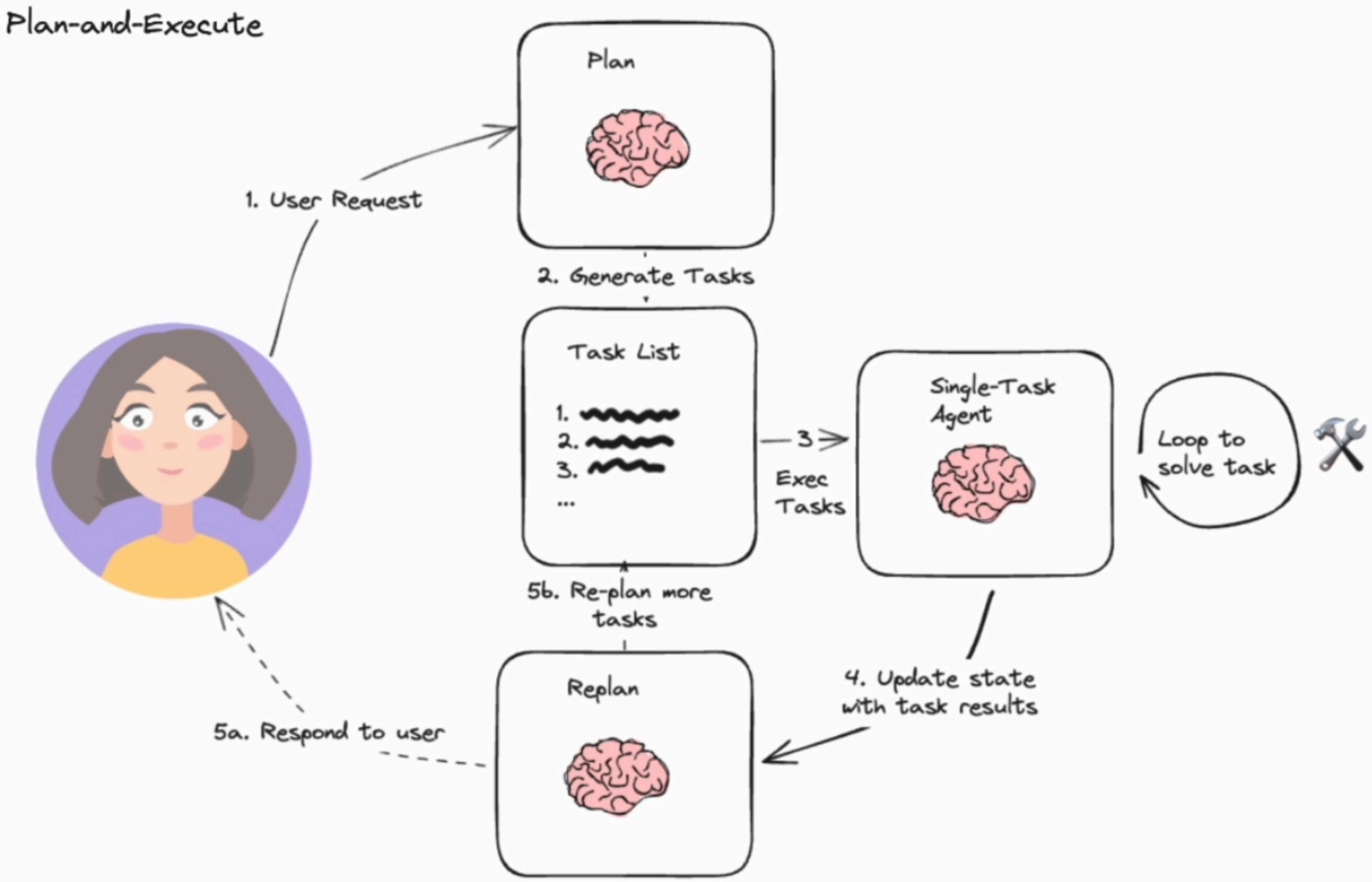
https://github.com/langchain-ai/langgraph/tree/main/examples/multi_agent

Flow Engineering



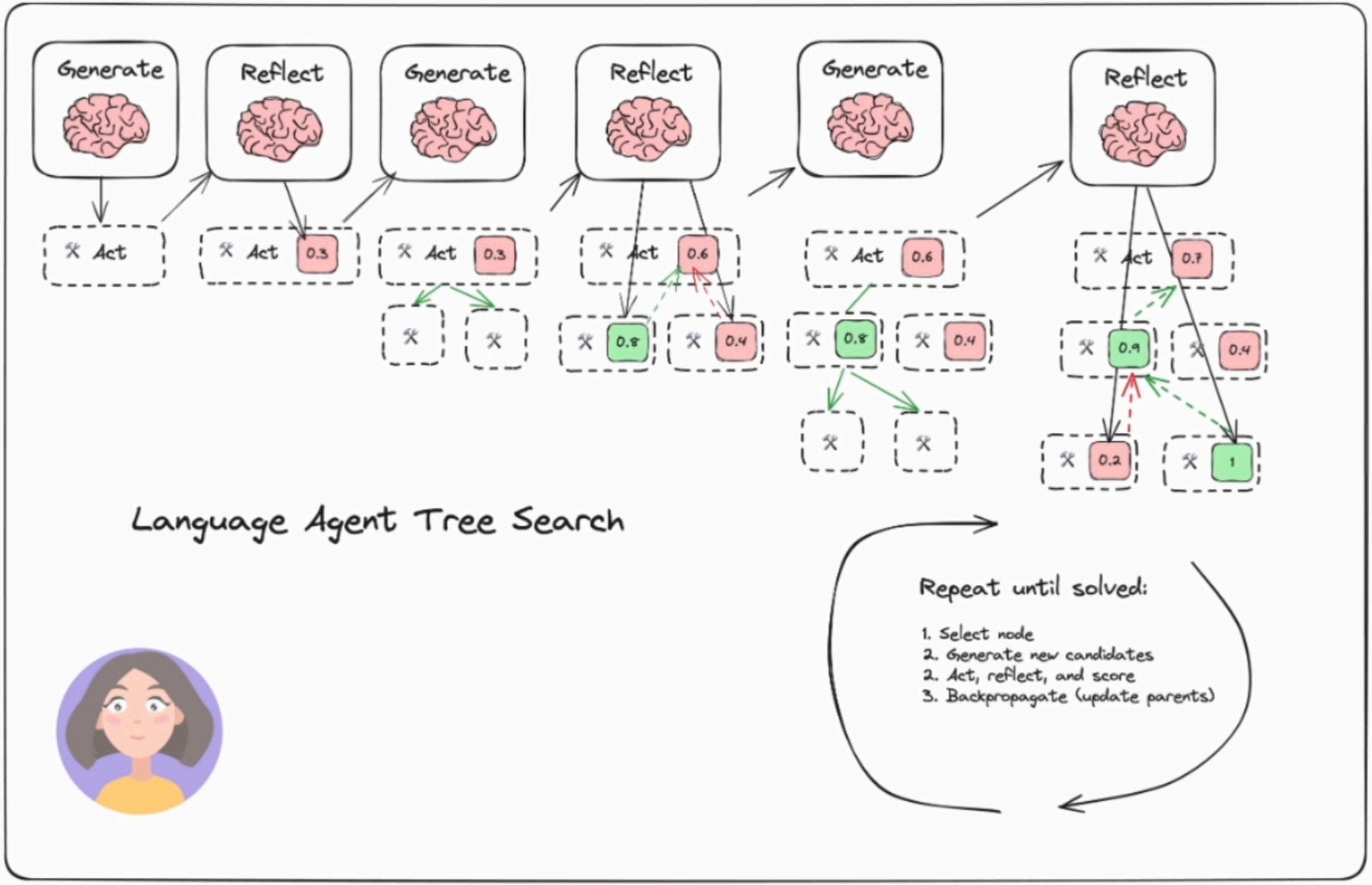
(a) The proposed AlphaCodium flow.

Plan and Execute



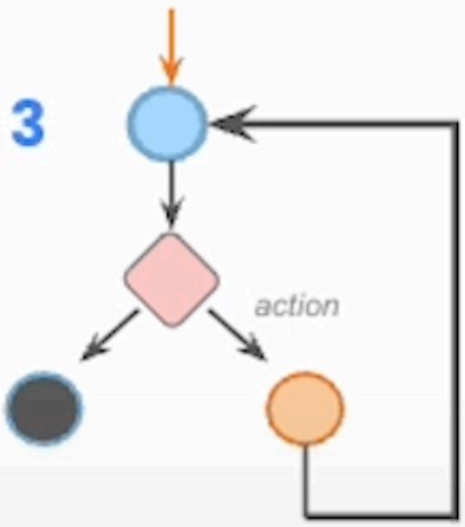
https://github.com/langchain-ai/langgraph/blob/main/examples/plan-and-execute/plan-and-execute.ipynb?ref=blog.langchain.dev

Language Agent Tree Search



https://github.com/langchain-ai/langgraph/blob/main/examples/lats/lats.ipynb

State Memory



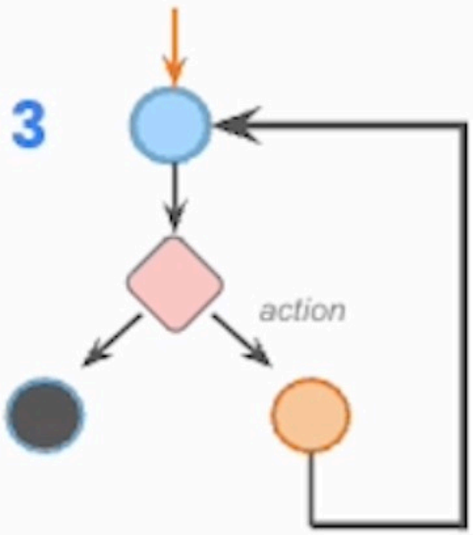
Memory

StateSnapshot: { AgentState, useful_things}

{ .., thread, thread_ts..}

config={'configurable': {'thread_id': '1',
'thread_ts': '1ef17b36-ed06-6185-8001-15cf75dea535'}}

State Memory



Memory

state 3

state 2

state 1

state 0

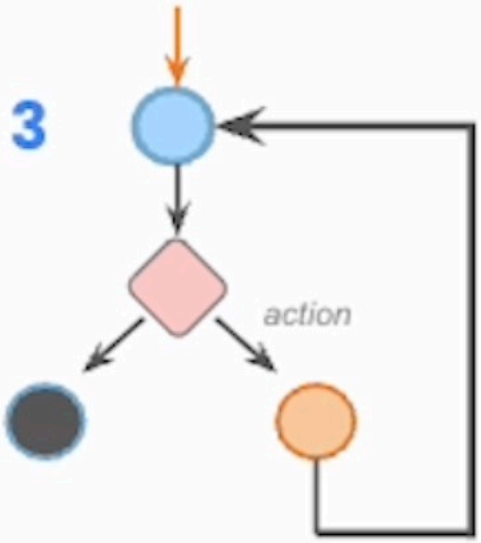
config={'configurable': {'thread_id': '1'}}

g.get_state(thread)

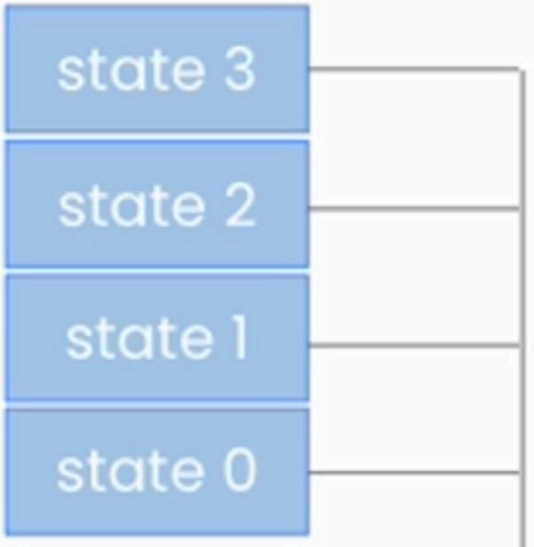
state 3

Returns 'current state'

State Memory



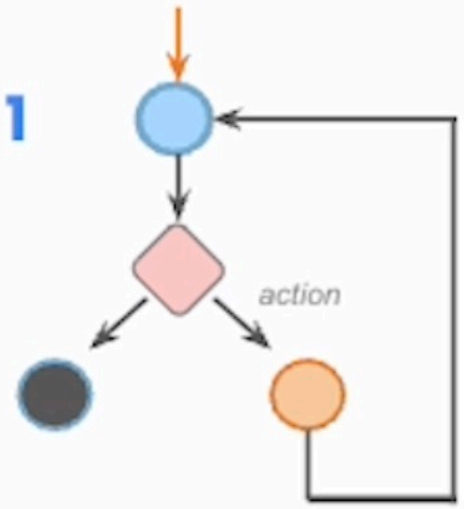
Memory



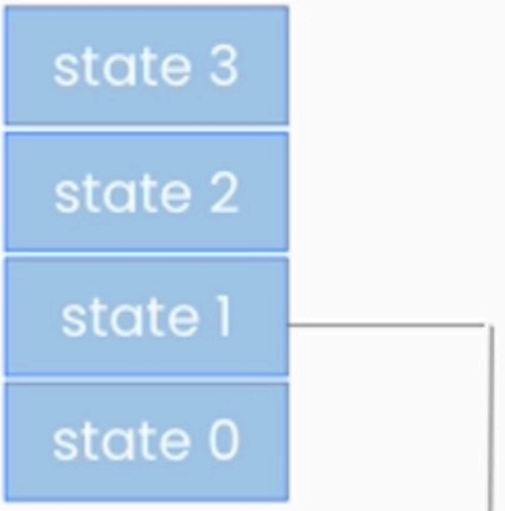
g.get_state_history(thread)

Returns iterator over all StateSnapshots

State Memory



Memory



g.invoke(None,{ .., thread, thread_ts..})

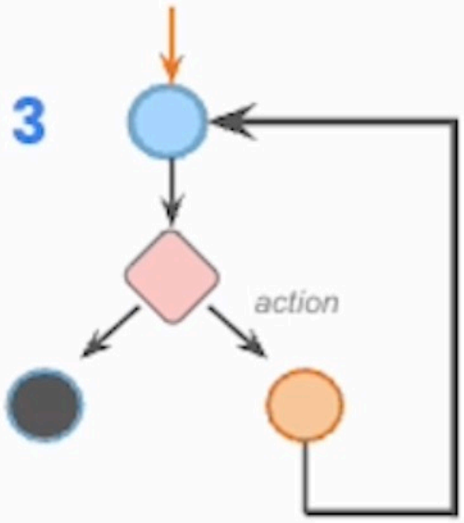
g.stream(None,{ .., thread, thread_ts..})

state 1

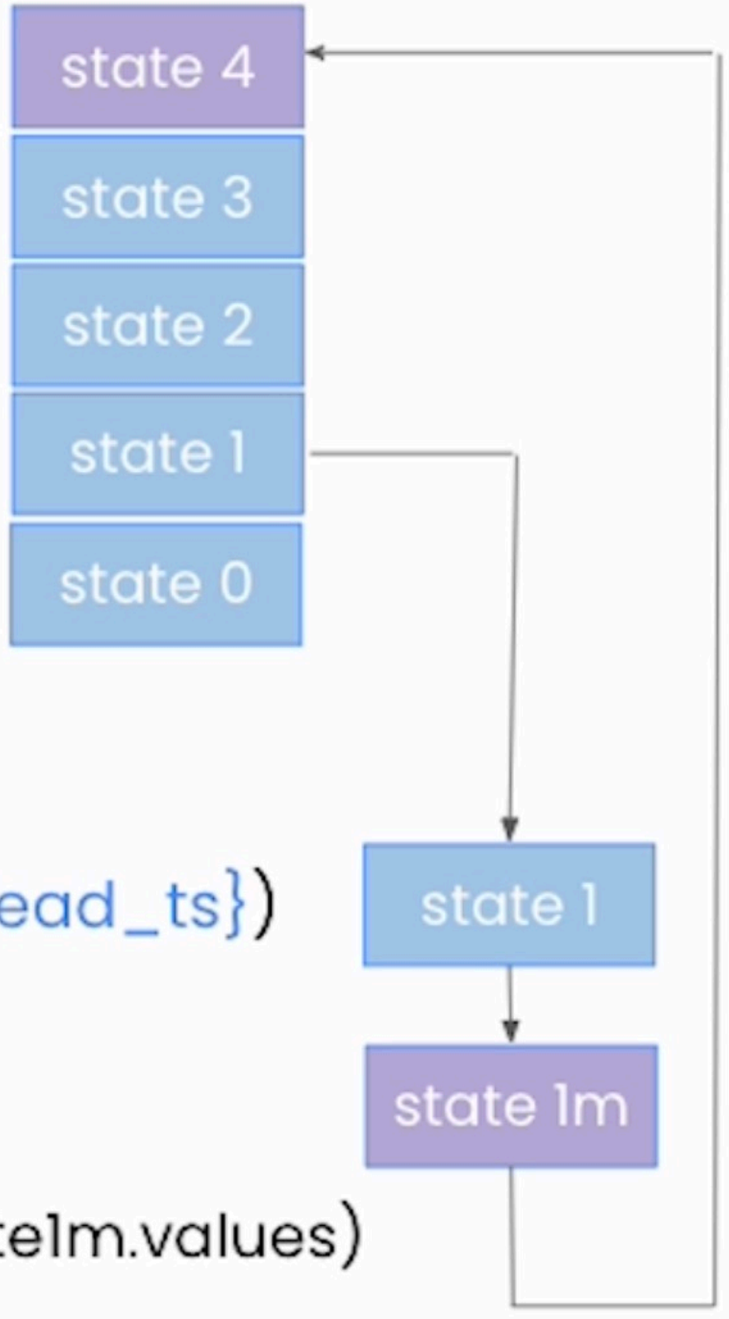
runs with state 1 as starting point

Time Travel

State Memory



Memory



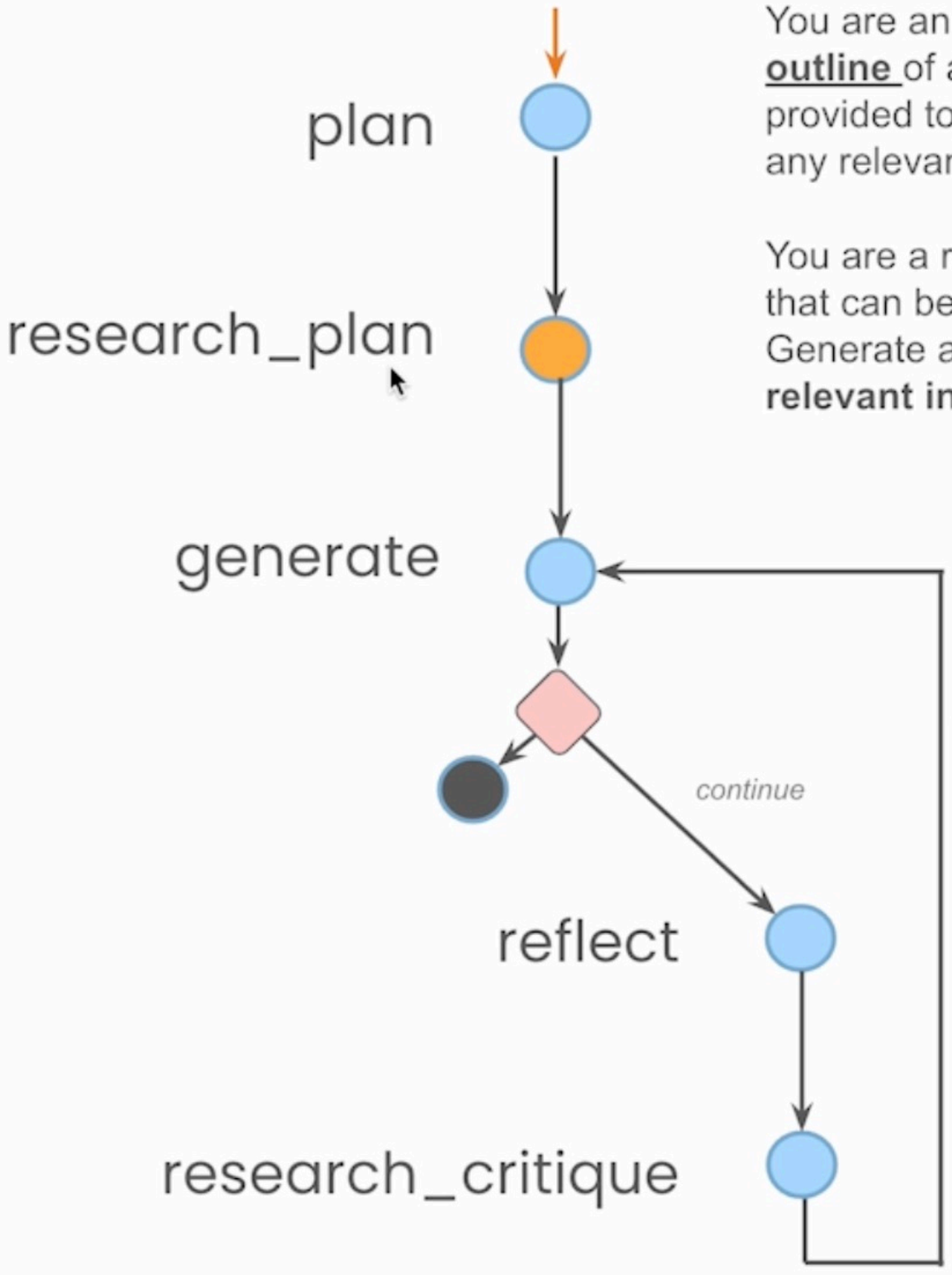
g.get_state({thread, thread_ts})

modify it

g.update_state(thread, state1m.values)

run it g.stream(None, thread)

Essay Writer



You are an expert writer tasked with writing a **high level outline** of an essay. Write such an outline for the user provided topic. Give an outline of the essay along with any relevant notes or instructions for the sections.

You are a researcher charged with providing information that can be used when writing the following essay. Generate a list of **search queries that will gather any relevant information.** Only generate 3 queries max.

You are an essay assistant tasked with **writing excellent 5-paragraph essays.**Generate … If the **user provides critique, respond with a revised version** of your previous attempts. Utilize information: