Who Am I?

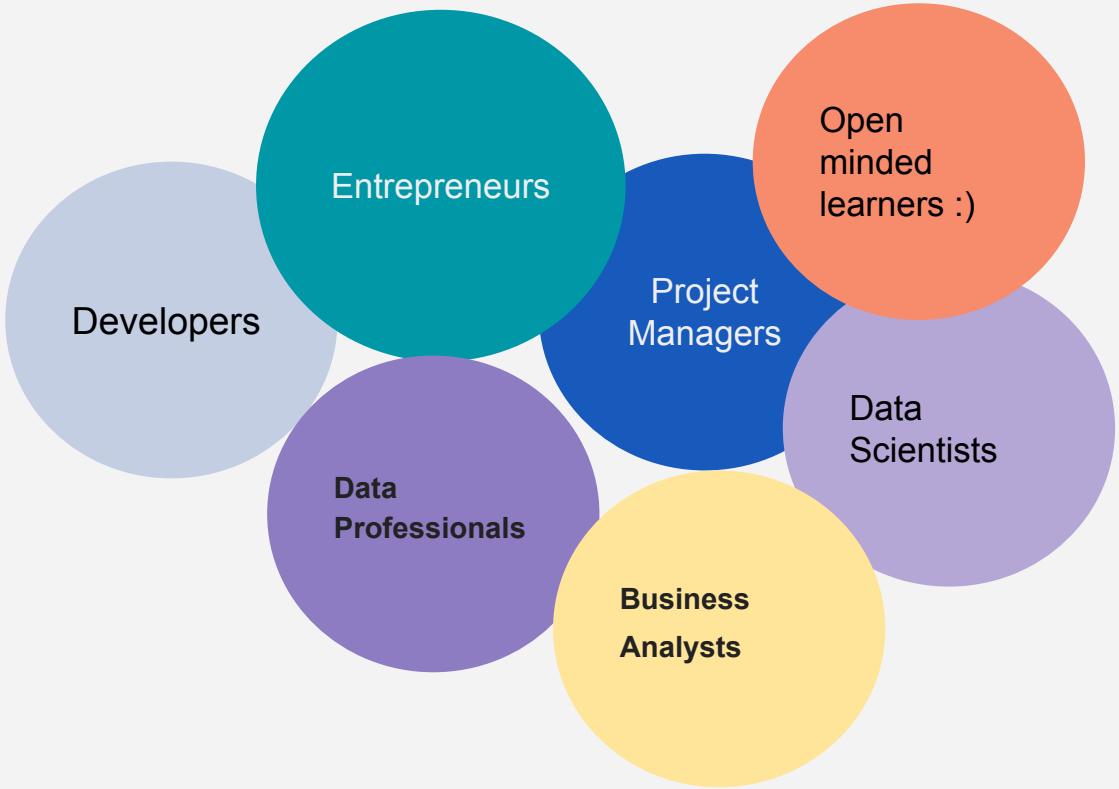
Paulo Dichone

Software, Cloud, AI Engineer
and Instructor

What Is This Course About?

- Building AI Agents -
  - AI Agents fundamentals
  - Implement AI Agents from Scratch
  - Optimization and Best Practices

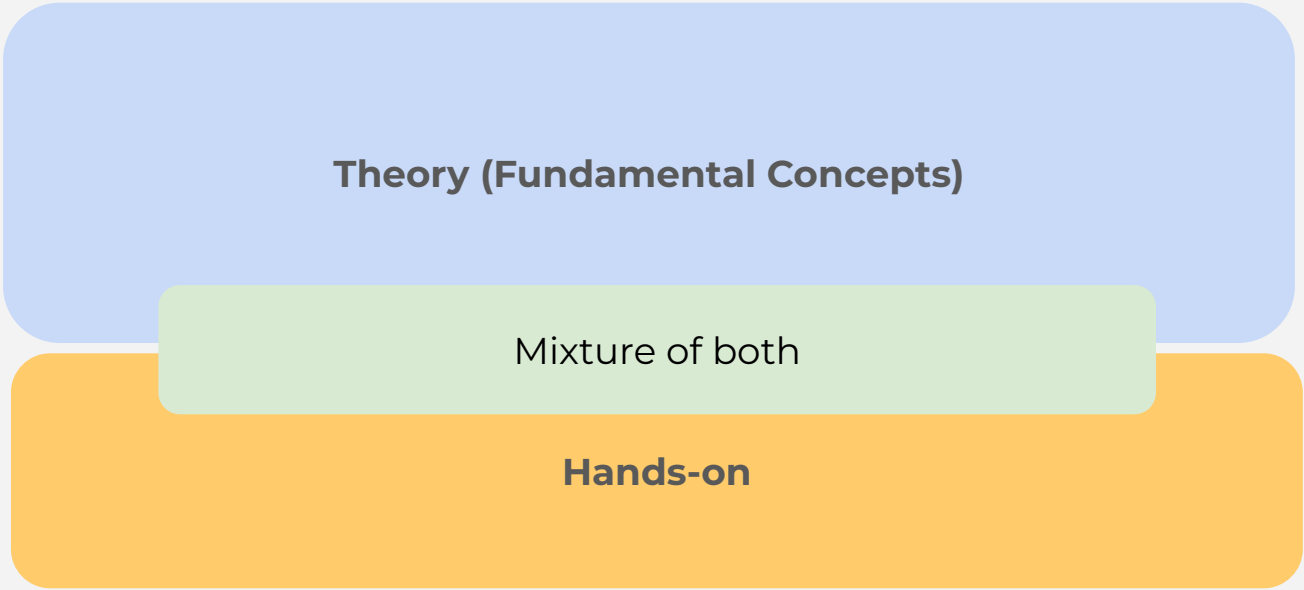
Who Is This Course For



Course Prerequisites

1. Know Programming (highly *preferred… at least the basics)*
    a. *There will be Python code*
    b. *Basics of LangChain, LLM…*
2. *This is not a programming course*
3. Willingness to learn :)

Course Structure



Development Environment setup

- Python
- VS Code (or any other code editor)
- OpenAI API Account and API Key

Set up OpenAI API Account

*** Please note** that you will need an API key to use OpenAI services, and there may be some costs associated with using the API. However, these costs should be minimal.

OpenAI API - Dev Environment Setup

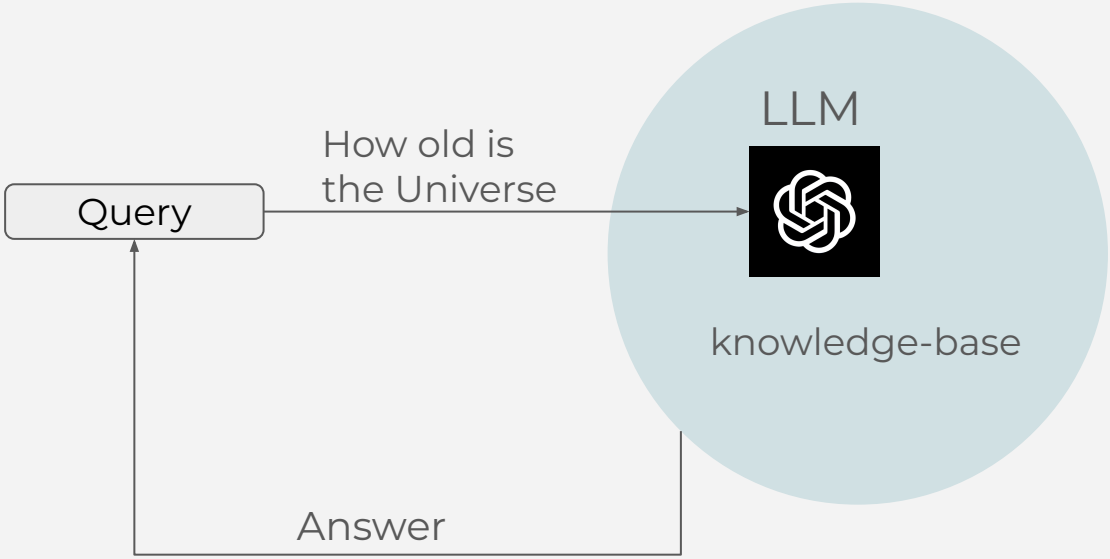
Python (Win, Mac, Linux)

https://kinsta.com/knowledgebase/install-python/

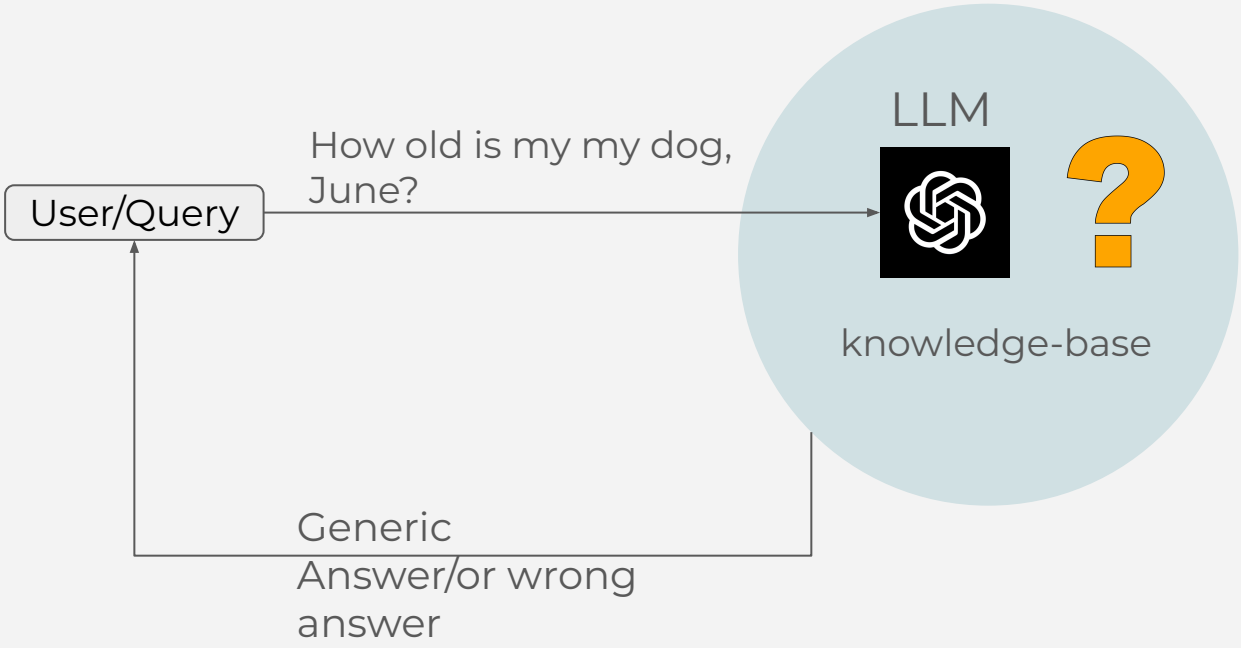
AI Agent Deep Dive

- What is it?
- Why (motivation)?
- Advantages

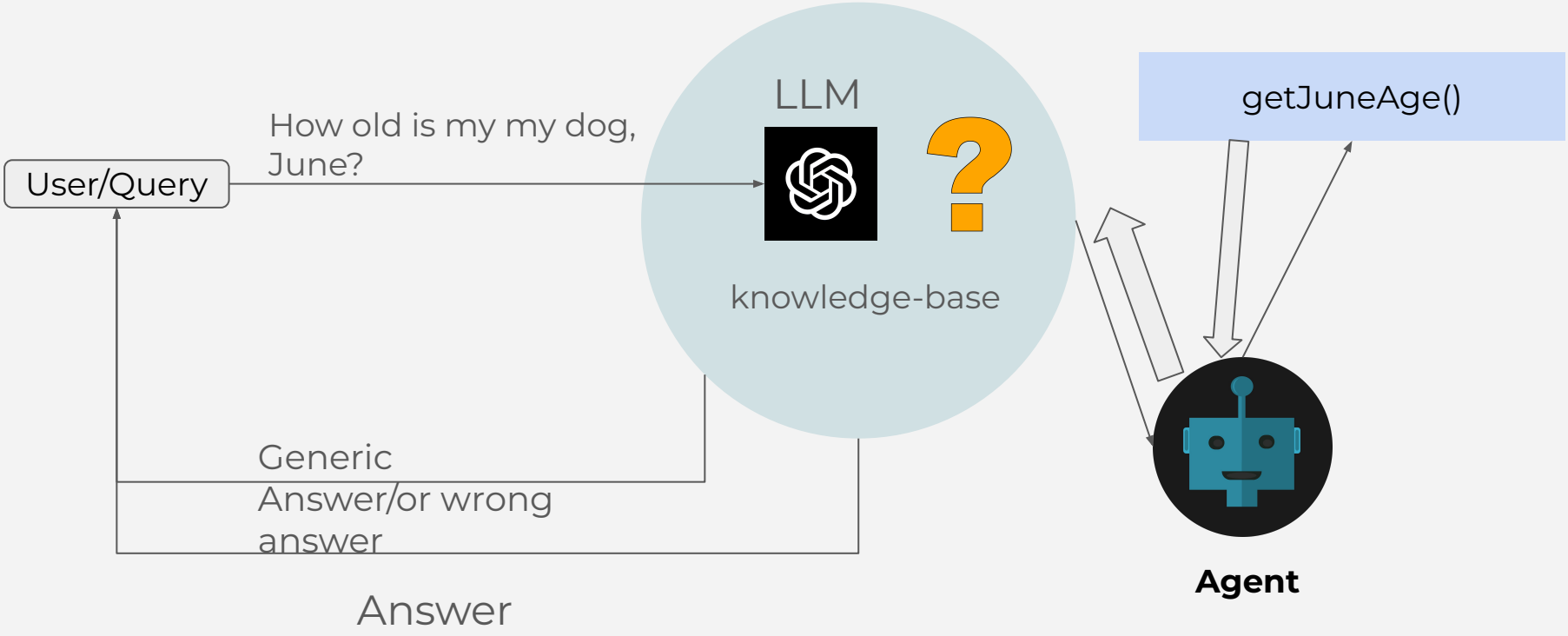
What is an (AI) agent (Motivation)?



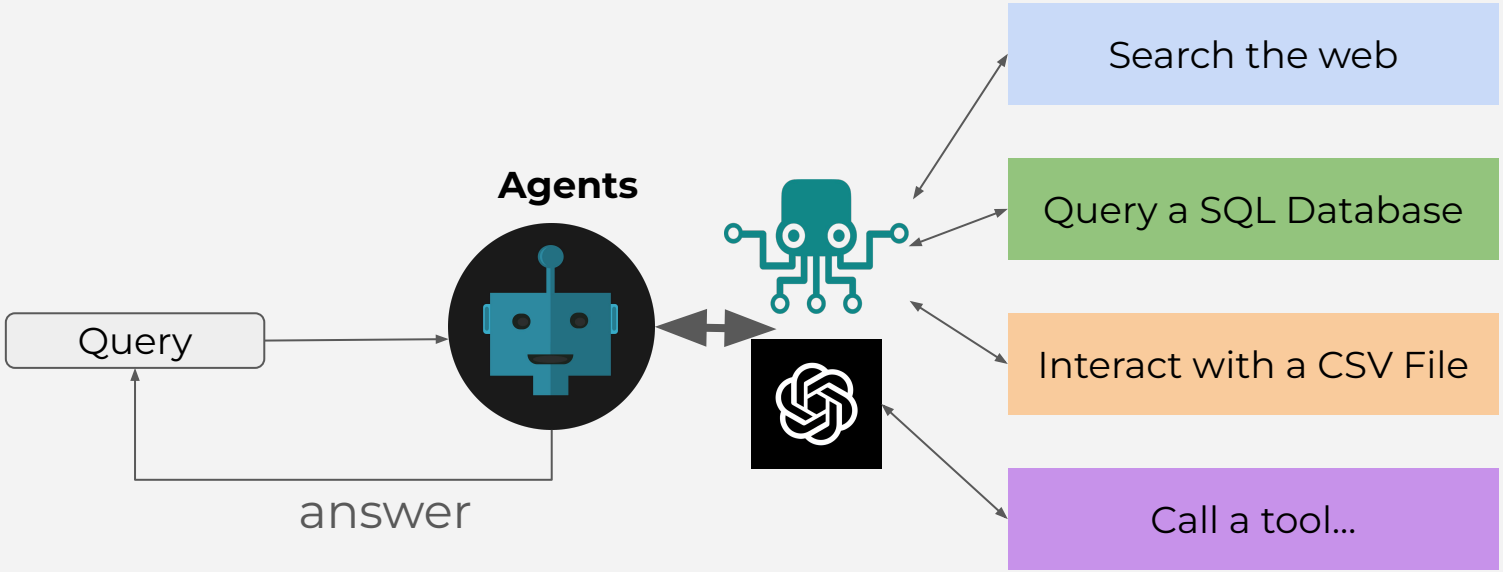
What is an (AI) agent (Motivation)?



What is an (AI) agent (Motivation)?

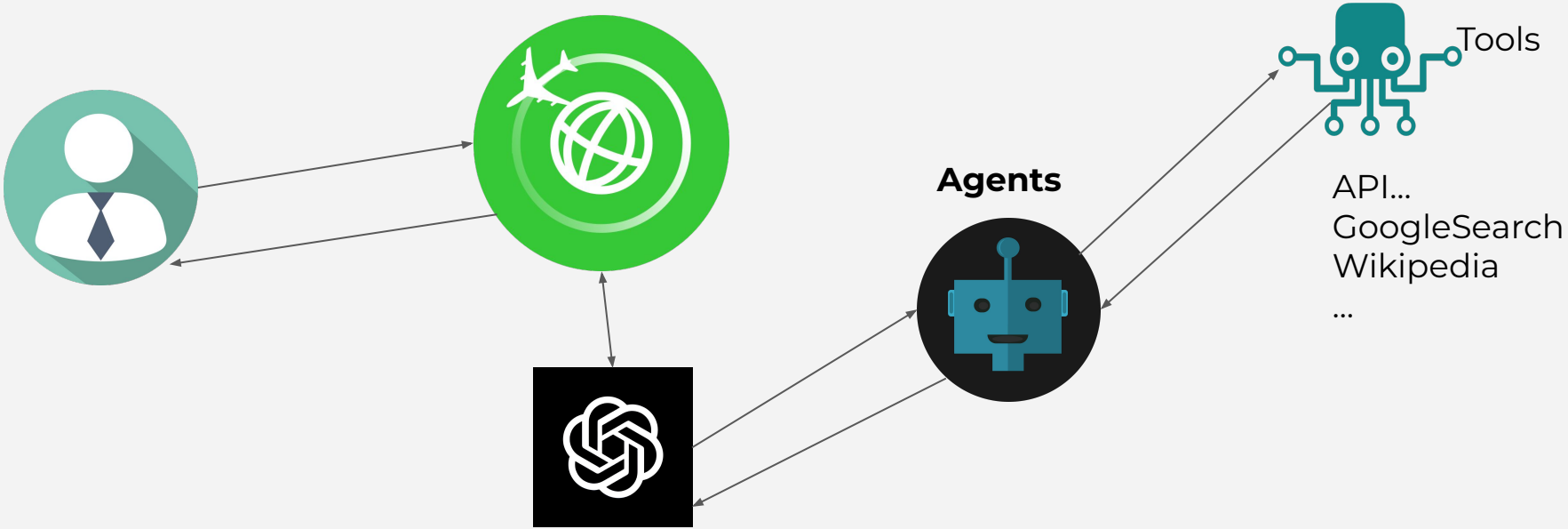


What is an (AI) agent?

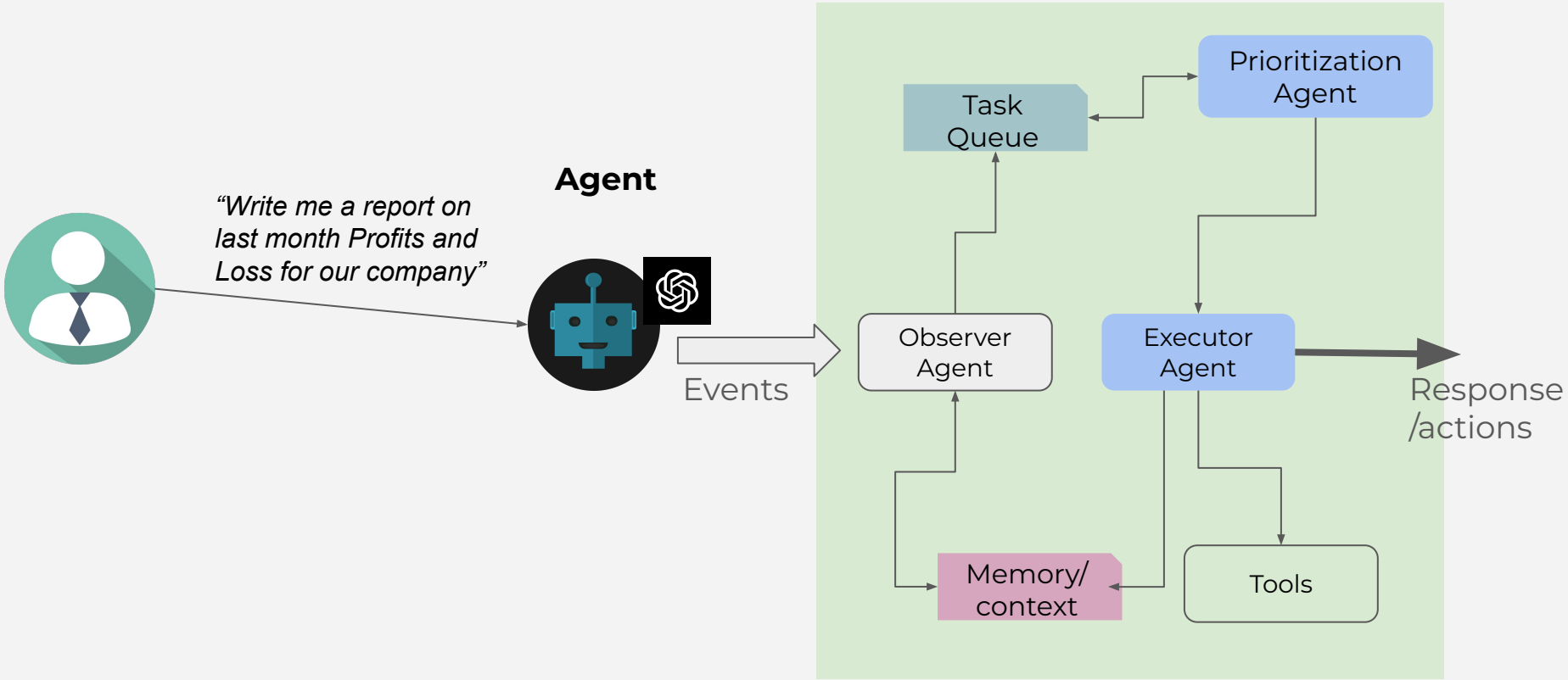


What is an agent & what they can do?

Personalized recommendations
Browse history
Previous vacations and activities...



Motivation Behind AI Agents: Solving Real-World Problems



Key characteristics

Autonomy

Learning and adaptation

Interaction

Goal-driven

Use cases

Customer service chatbots

Personal assistants

Data analysis

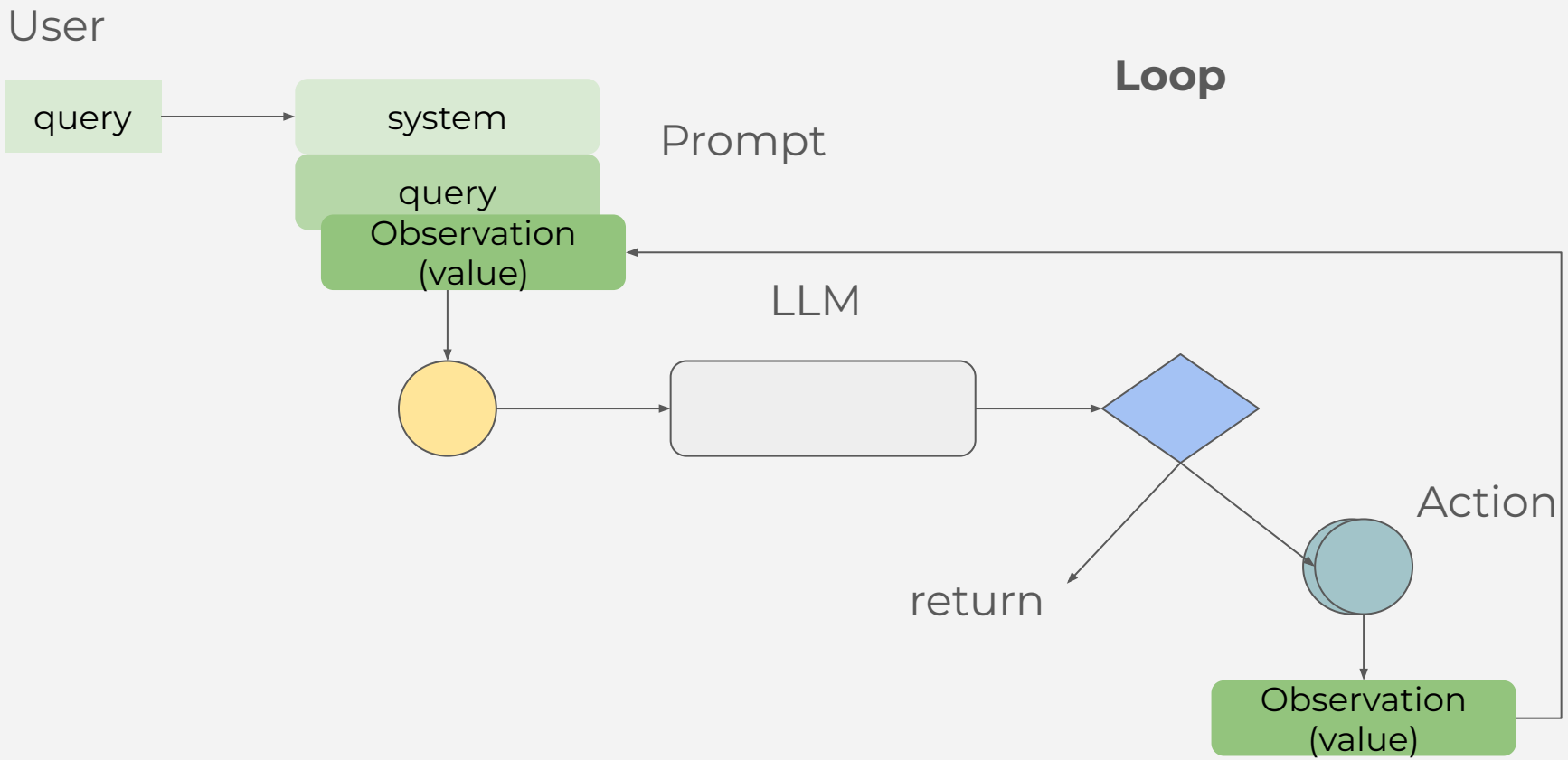
Smart home systems...

First
AI Agent

Build your very first Agent

- OpenAI model
- Python

First Agent

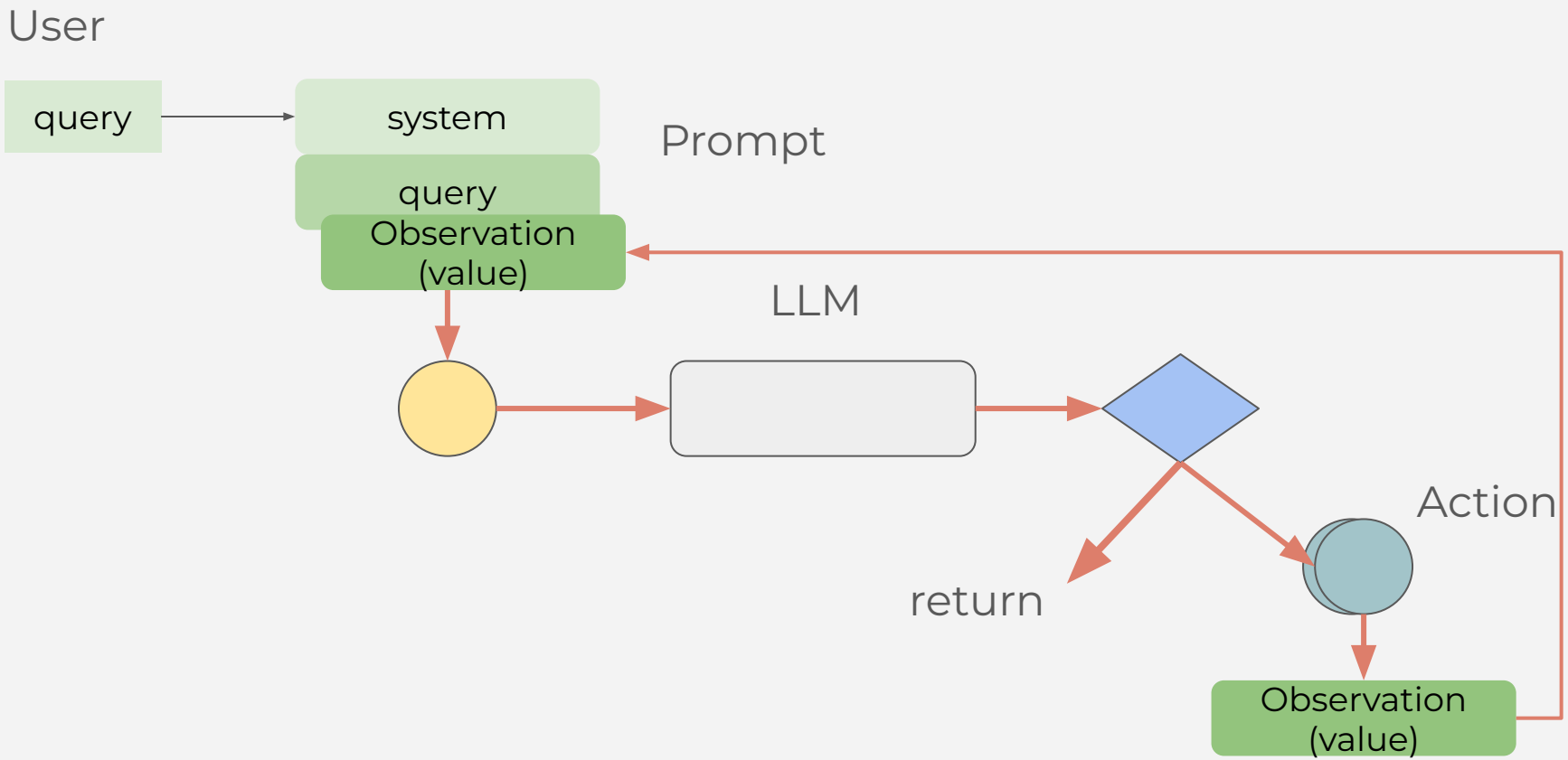

User
query
system
query
Observation
(value)
Prompt
Loop
LLM
Action
return
Observation
(value)

LangGraph
Deep Dive

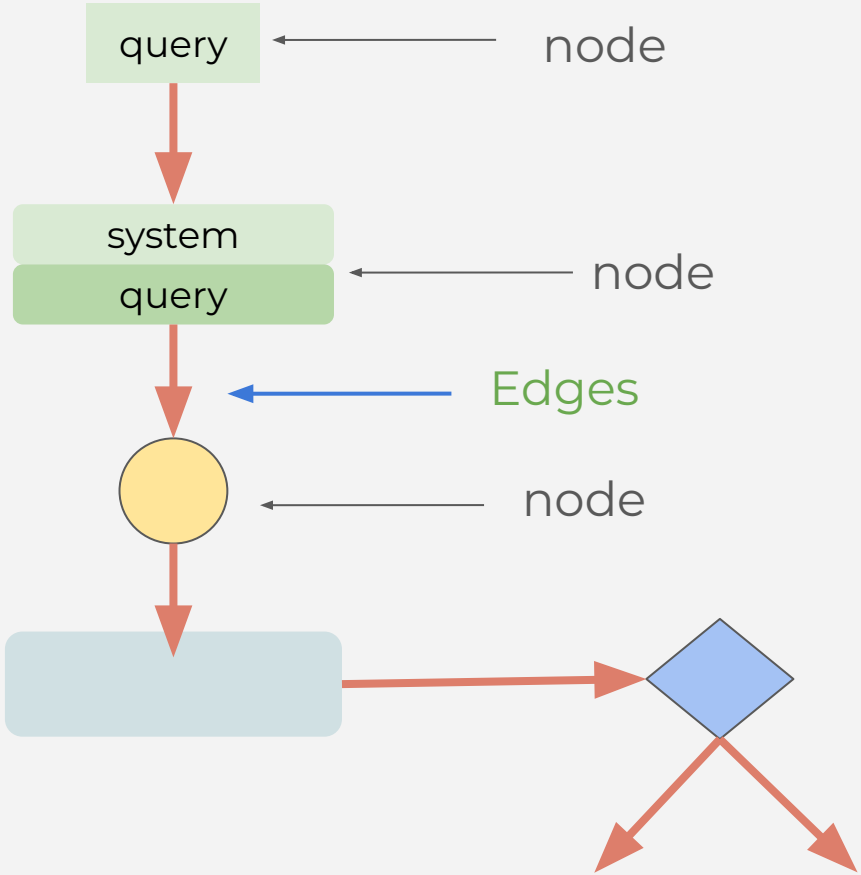
Understanding LangGraph

- What is it?
- How does it work?
- How to use it to build AI Agents

What is LangGraph?



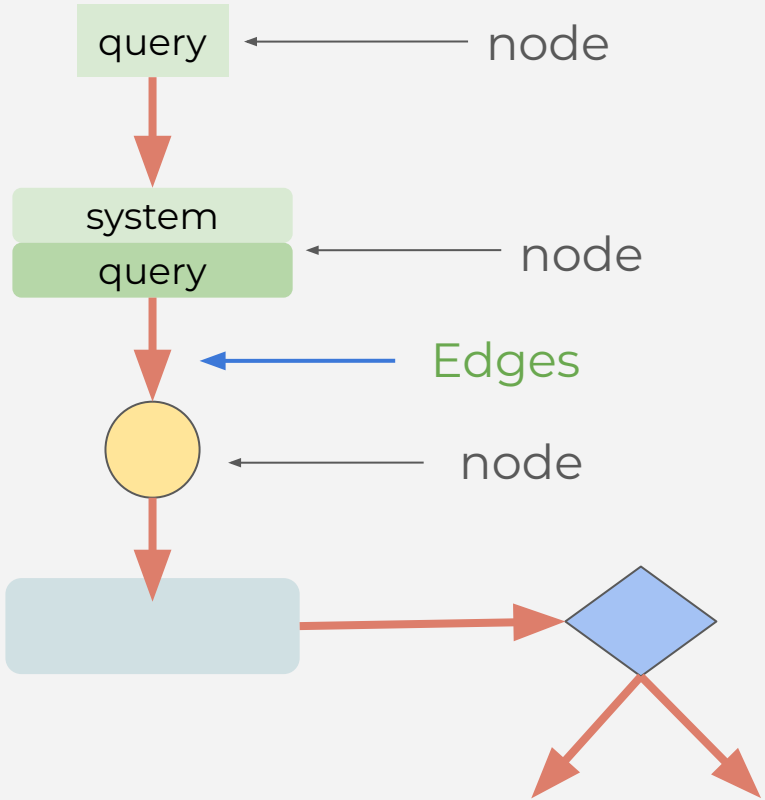
What is LangGraph?



Nodes - entities, concepts

Edges - relationships between nodes.

What is LangGraph - Key Components

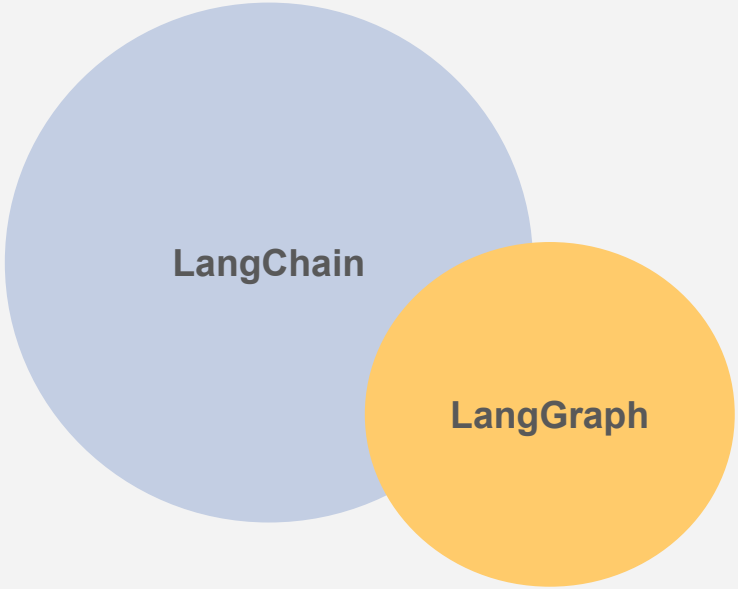


1. **Graph-Based Knowledge Representation**
   a. Uses graphs to represent knowledge
2. **Natural Language Processing**
   a. Uses NPL to parse and understand user input
3. **Reasoning and Inference**
   a. Logical reasoning and inference to make decisions
4. **Interaction and Dialogue Management**
   a. Maintain context

How LangGraph Helps in Building AI Agents

1. **Enhanced Knowledge Management**
    a. Graphs are amazing for organizing and querying knowledge
2. **Improved Natural Language Understanding**
    a. NLP + graph-based representation is a great combination for agents
3. **Efficient Reasoning and Decision Making**
    a. Graph-based reasoning enables agent to become more efficient
4. **Scalability and Flexibility**
    a. It’s easy to scale graphs to handle large datasets and complex relationships

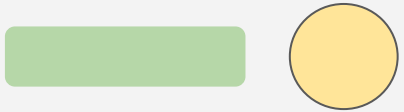
LangGraph and LangChain



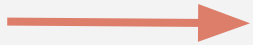
LangGraph is an extension of **LangChain** that support Graphs

- Allows to create **cyclic graphs**
- **Persistence** - *remembering previous conversations or context*
  - **Human-in-the-loop** - *human interaction and feedback* (thanks to persistent feature)

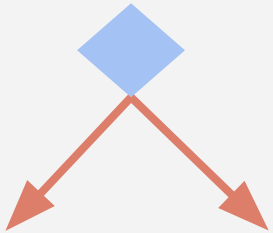
LangGraph Core Concepts



Nodes: Agents (entities) or functions

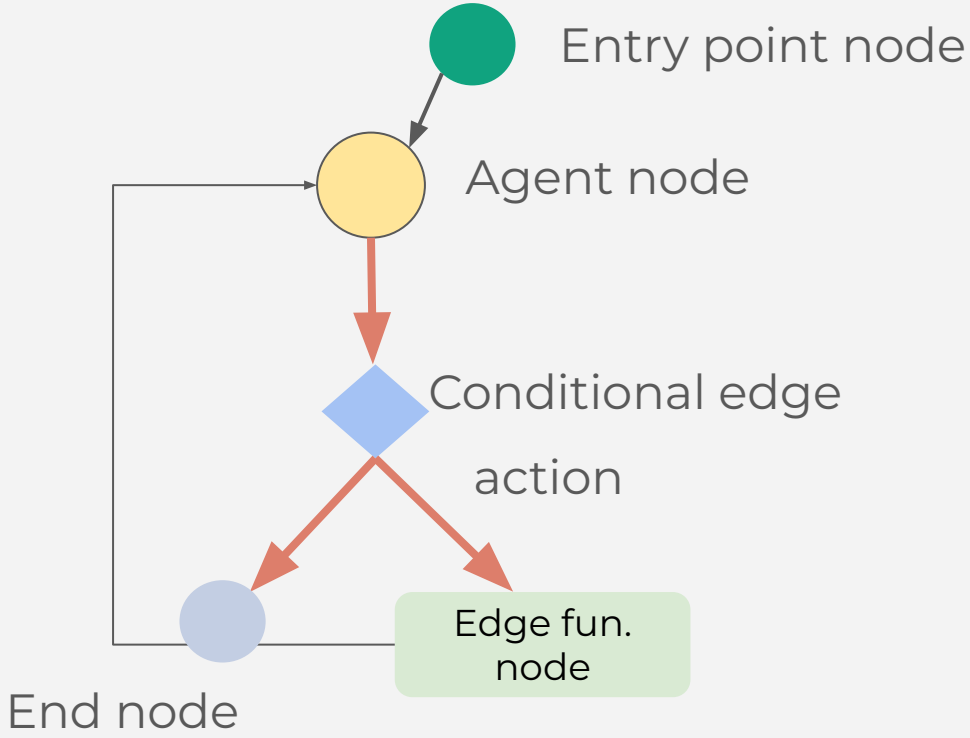


Edges: connect nodes (relationships)

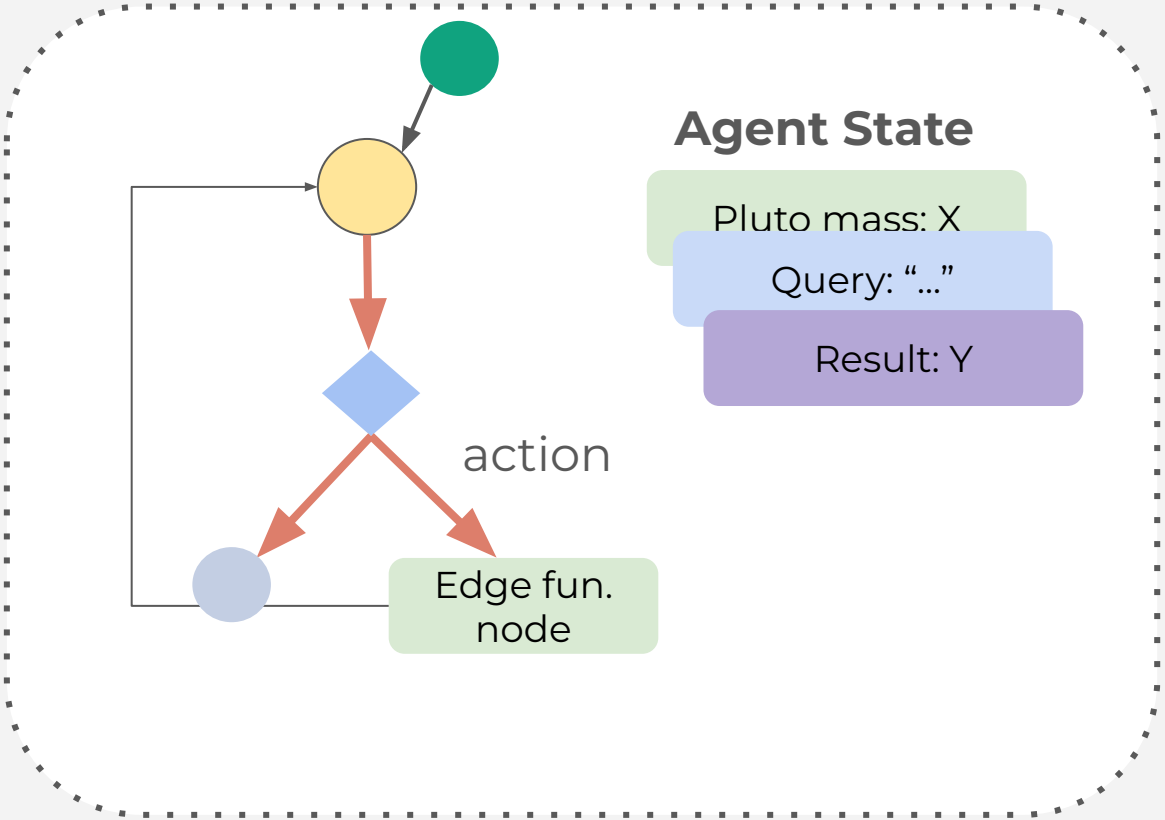


Conditional Edges: Decision

LangGraph Flow Example



LangGraph Data & State



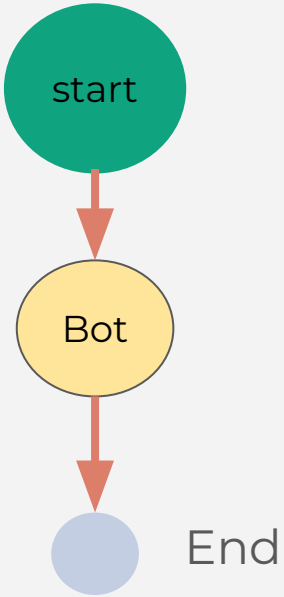
- State is accessible to all parts of the graph
- Local to the graph
- Can be stored (DB, etc)

LangGraph
Hands on

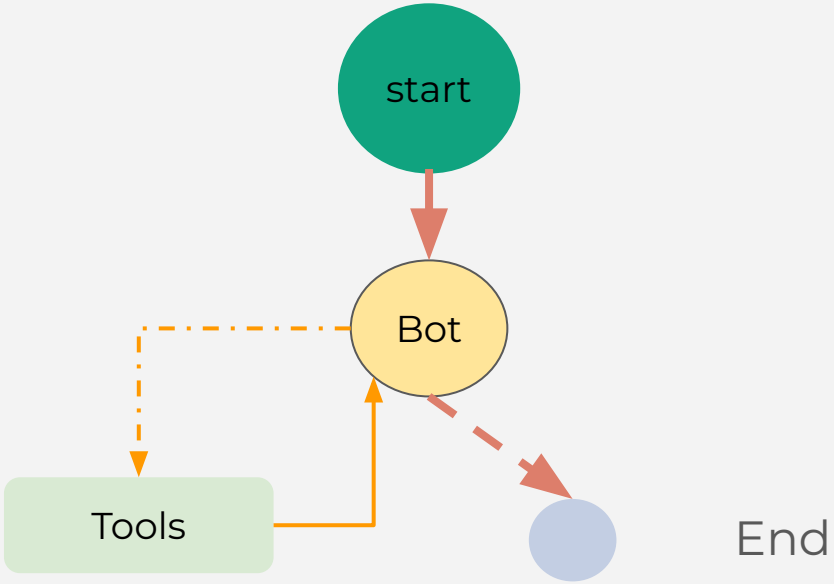
LangGraph Hands on

- Build a simple Agent with...
  - LangGraph
    - Basics of LangGraph

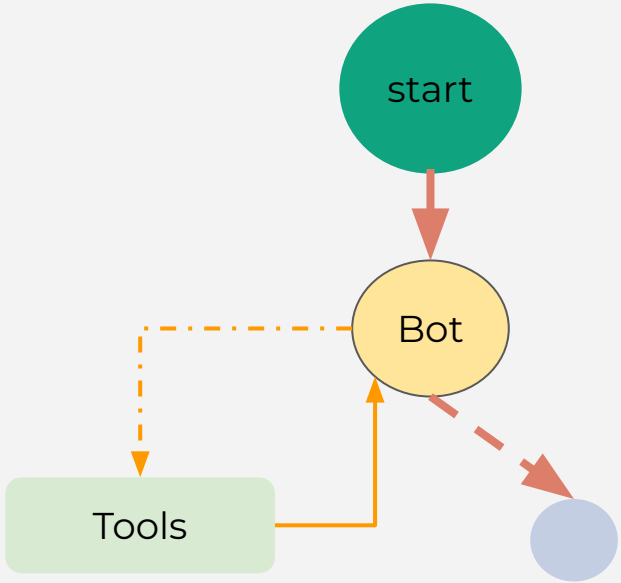
LangGraph - Basic Agent



LangGraph - Basic Agent - Add Tools



LangGraph - Basic Agent - Add Memory



End

Agent Memory

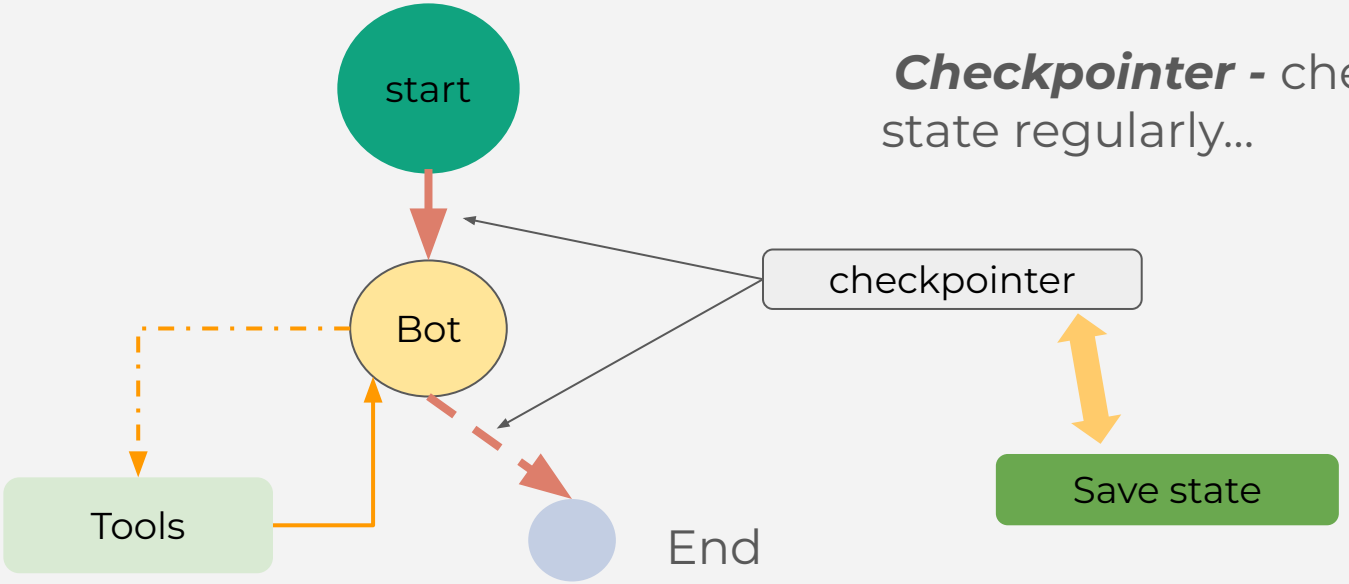
Context 1

Context 2

Thread_id

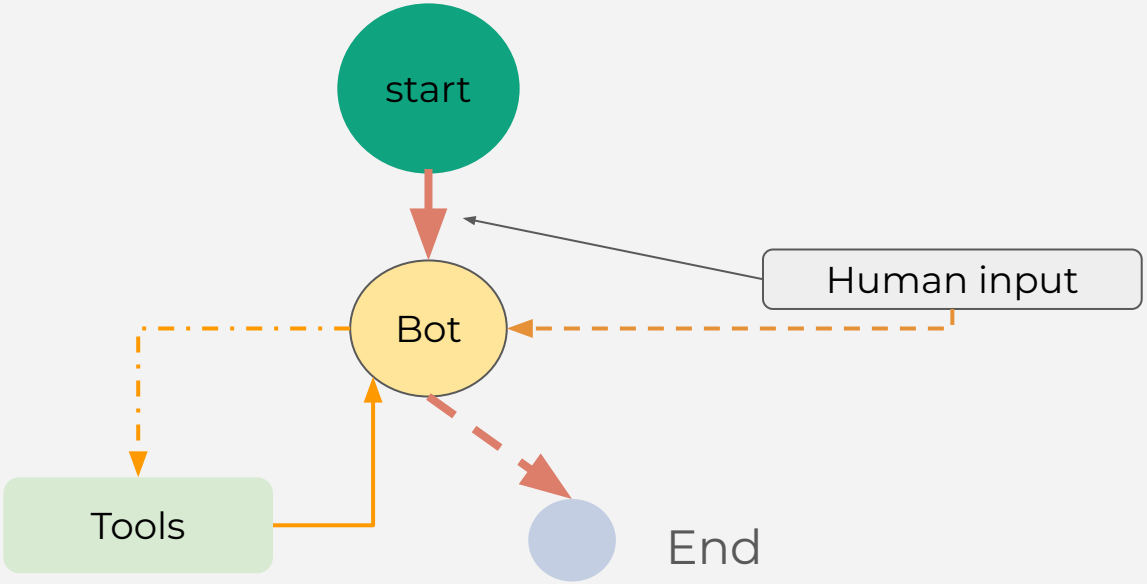
Through a ***checkpointer***

LangGraph - Basic Agent - Add Memory



Checkpointer - checks the state regularly...

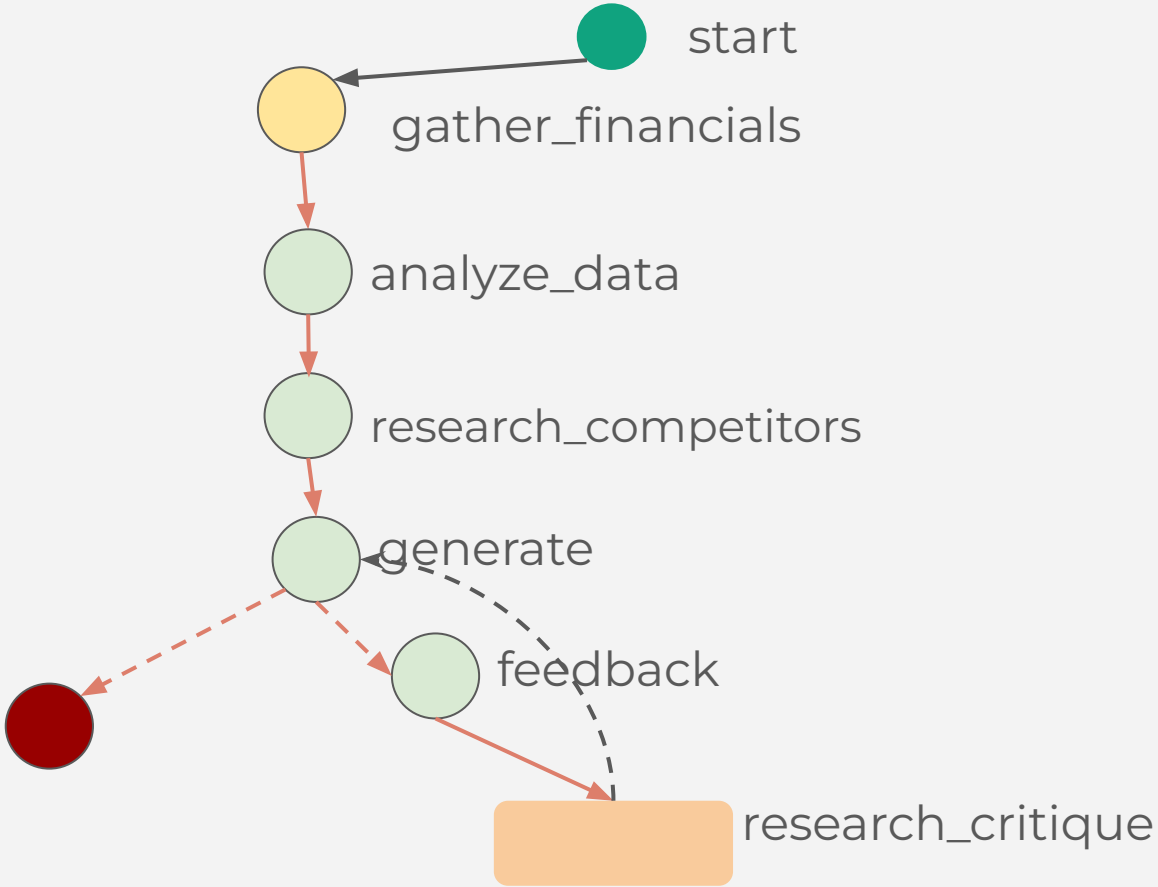
LangGraph - Basic Agent - Human-in-the-loop



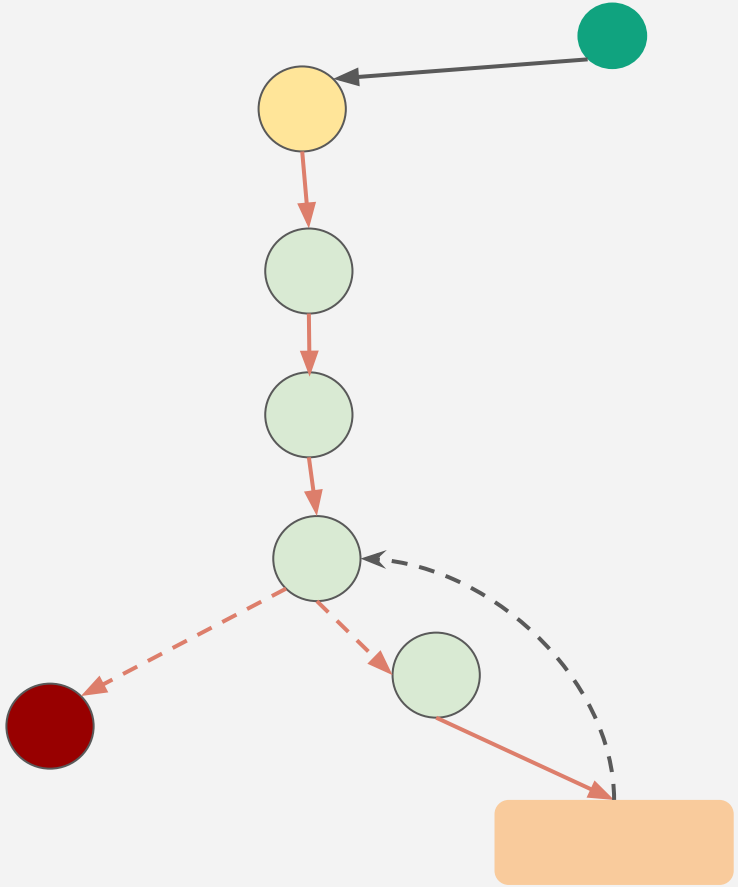
Build a Full *AI Agent*

Build a full AI Agent
Financial Performance Reporting Writer Agent

Financial Report Writer Agent



Agent Optimization - Overview



1. **Model Optimization**
    a. Use efficient models
        i. Utilize smaller, efficient models for less intensive tasks
    b. Model Quantization
    c. Fine-Tuning
2. **Parallel Processing**
    a. Distributed computing
    b. Batch processing
3. **Caching and Preprocessing**
    a. Data caching
    b. Preprocessing

Agent Optimization - Model Optimization

```python
# === Model Optimization ===
from langchain_openai import ChatOpenAI
from langchain_core.messages import SystemMessage, HumanMessage

# Use a smaller model for initial data processing tasks
small_model = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)

def gather_financials_node(state: AgentState):
    csv_file = state['csv_file']
    df = pd.read_csv(StringIO(csv_file))
    financial_data_str = df.to_string(index=False)
    combined_content = f"{state['task']}\n\nHere is the financial data:\n\n{financial_data_str}"

    messages = [
        SystemMessage(content=GATHER_FINANCIALS_PROMPT),
        HumanMessage(content=combined_content)
    ]

    response = small_model.invoke(messages)
    return {"financial_data": response.content}
```

Agent Optimization - Parallel Processing

```python
from concurrent.futures import ThreadPoolExecutor

def research_competitors_node(state: AgentState):
    content = state['content'] or []

    def fetch_competitor_data(competitor):
        queries = model.with_structured_output(Queries).invoke([
            SystemMessage(content=RESEARCH_COMPETITORS_PROMPT),
            HumanMessage(content=competitor)
        ])
        for q in queries.queries:
            response = tavily.search(query=q, max_results=2)
            return [r['content'] for r in response['results']]

    with ThreadPoolExecutor() as executor:
        results = executor.map(fetch_competitor_data, state['competitors'])
        for result in results:
            content.extend(result)

    return {"content": content}
```

Agent Optimization - Caching & Pre-processing

```python
import functools
from cachetools import TTLCache
# Cache with TTL of 1 hour
cache = TTLCache(maxsize=100, ttl=3600)

@functools.lru_cache(maxsize=128)
def cached_tavily_search(query):
    return tavily.search(query=query, max_results=2)

def research_competitors_node(state: AgentState):
    content = state['content'] or []

    def fetch_competitor_data(competitor):
        queries = model.with_structured_output(Queries).invoke([
            SystemMessage(content=RESEARCH_COMPETITORS_PROMPT),
            HumanMessage(content=competitor)
        ])
        for q in queries.queries:
            response = cached_tavily_search(q)
            return [r['content'] for r in response['results']]

    with ThreadPoolExecutor() as executor:
        results = executor.map(fetch_competitor_data, state['competitors'])
        for result in results:
            content.extend(result)
    return {"content": content}
```

Course Summary

- Building (AI) Agents
  - Agents deep dive
  - Key characteristics
  - Agent use cases
  - Deep dive into Building Agents
    - Built first simple Agent (using LLM only)
    - Building Agents with LangGraph
    - LangGraph Introduction (Deep dive)
      - Building blocks
      - Main components
  - Build a full fledged Financial Report Writer/Researcher Agent
    - Test
    - Build a GUI with Streamlit

Hands on

Congratulations!

You made it to the end!

- Next steps...

Wrap up - Where to Go From Here?

- Keep learning
  - Extend the projects we worked on in this course
  - Implement your own AI agent
- Read more on LangGraph- https://langchain-ai.github.io/langgraph/
- Read the OpenAI documentation: https://platform.openai.com/docs/overview
- Challenge yourself to keep learning new skills!

Thank you!