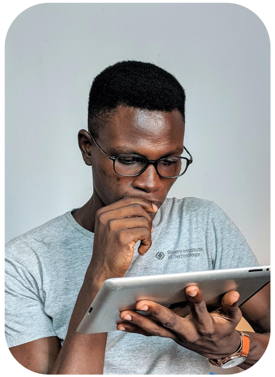
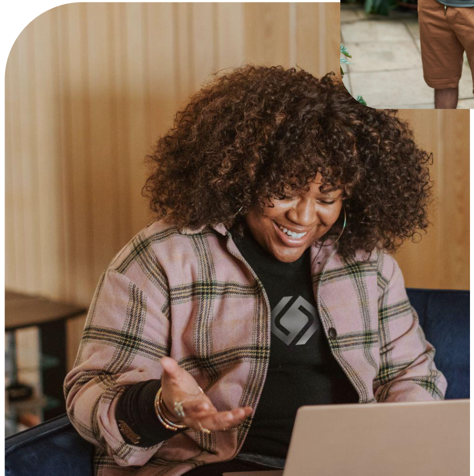
Bloom Institute of Technology

LangGraph

LangGraph is a library for building stateful, multi-actor applications with LLMs, inspired by Pregel and Apache Beam. It allows coordination and checkpointing of multiple chains (or actors) using Python or JS, with a public interface inspired by NetworkX.



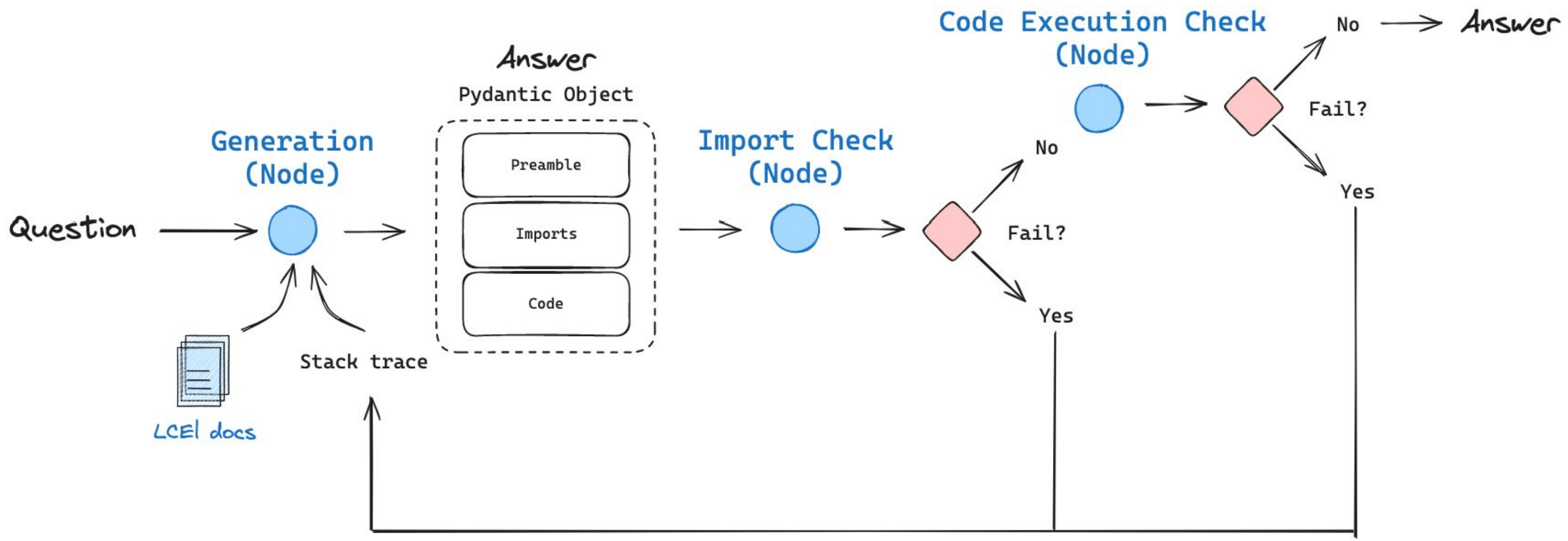


Core Competencies

The student must demonstrate…

1. Components of an AI graph (5 min)
2. The value of state in a graph (5 min)
3. Useful cases for graphs (10 min)
4. The multi-agent graph concept (10 min)
5. The supervisor concept (10 min)
6. Hierarchical teams (10 min)

Components of AI Graphs



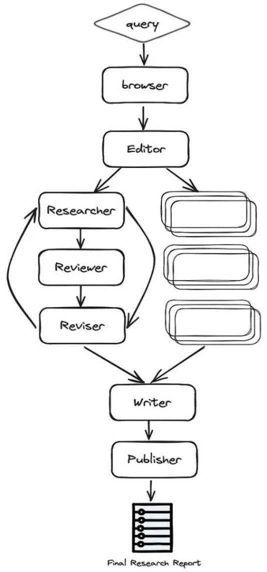
Node: In LangGraph, a node represents a function that, when executed, receives inputs, processes them, and returns results to be passed to the next node.

Edge: An edge represents the path connecting nodes, guiding the flow of data. Edges can be static (always moving from one node to a specific next node) or conditional (determining the next node based on criteria).

State in LangGraph

State is a dictionary of relative information. The state is passed to the next node in the graph and can be updated as the graph is executed. This allows the graph to make decisions based on the state.

```
class ResearchState(TypedDict):
    task: dict
    initial_research: str
    sections: List[str]
    research_data: List[dict]
    # Report layout
    title: str
    headers: dict
    date: str
    table_of_contents: str
    introduction: str
    conclusion: str
    sources: List[str]
    report: str
```



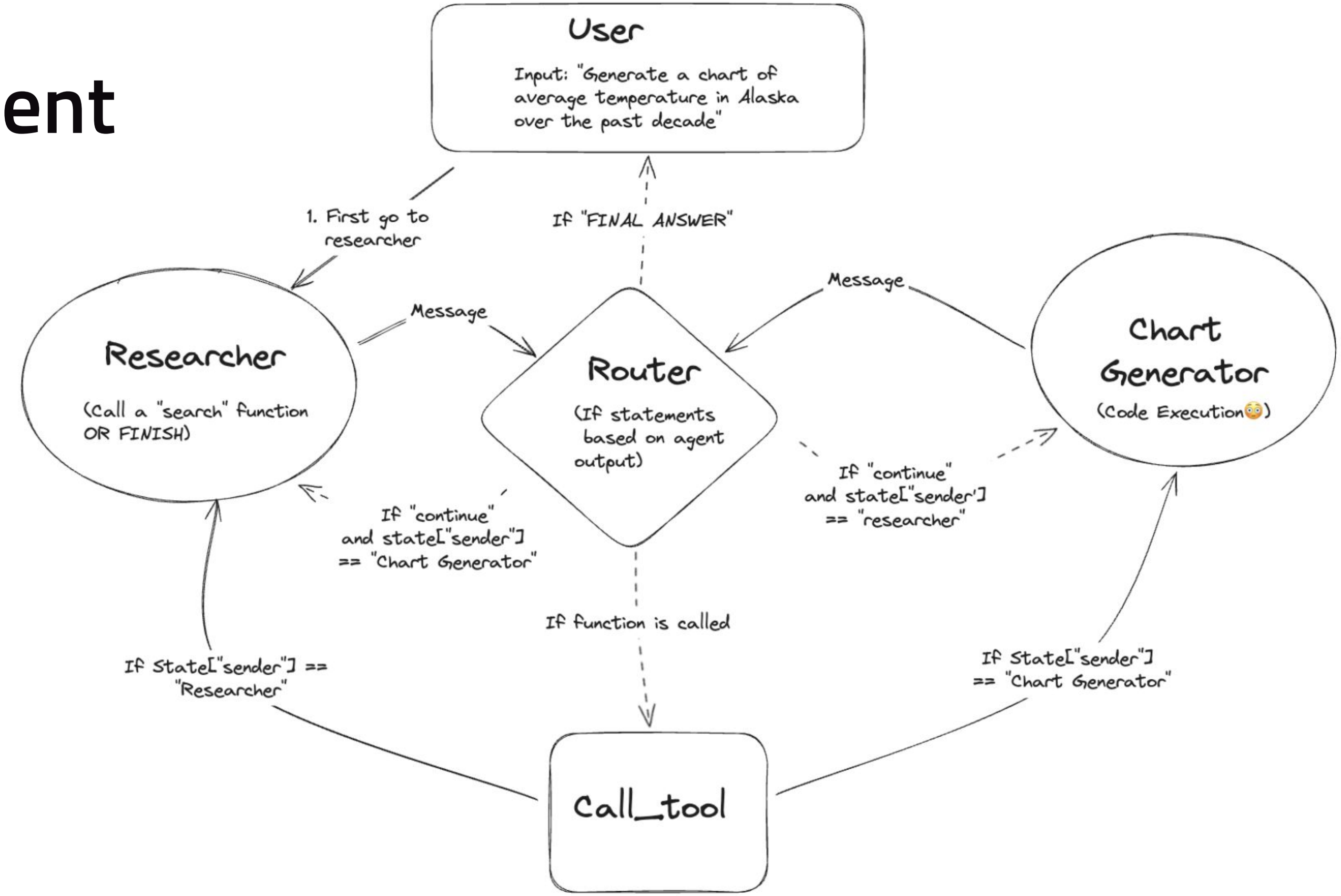
Check for Understanding: How does `ResearchState` flow through the graph?

Chains, Agents, or Graphs?

Graphs provides developers with a high degree of controllability and is important for creating custom agents and flows. Nearly all agents in production are customized towards the specific use case.

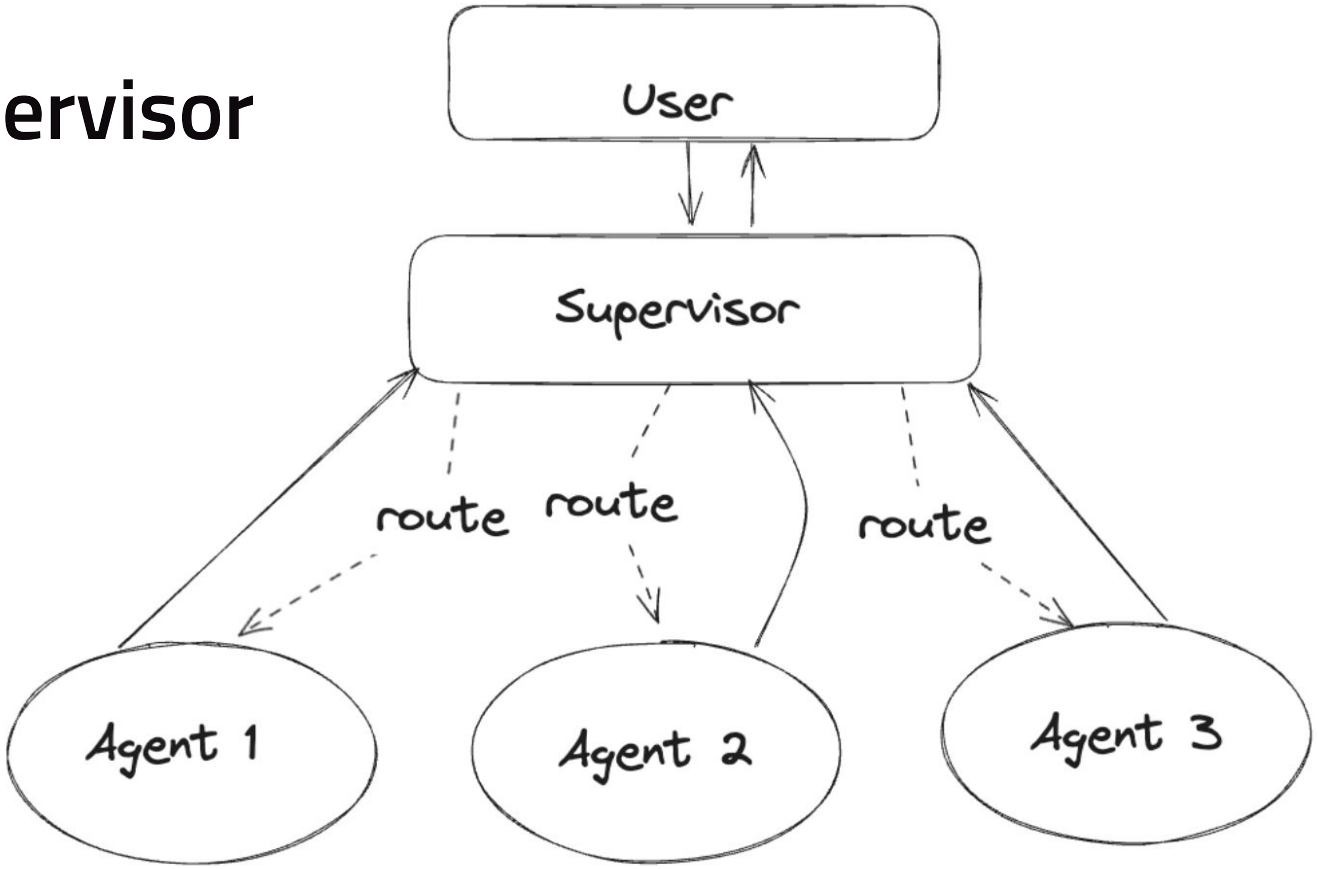
Concept	Special Attributes	Pros	Cons	Use Cases
Chains	• Sequential execution • Composable and modular	• Easy to understand and debug • Reusable components	• Can be too complex • Limited to predefined flow	• Data preprocessing • Multi-step data transformation
Agents	• Dynamic decision making • Interacts with environment	• Flexible • Can handle complex tasks	• Harder to debug • Higher complexity and less control	• Real-time recommendation • Interactive chatbots
Graphs	• Node and edge representation • Visualization of dependencies	• Clear, complex relationships • Easier to manage dependencies	• Can become visually cluttered • Requires graph theory	• Knowledge representation • Complex workflows with dependencies

Multi-Agent Graph


User
Input: "Generate a chart of average temperature in Alaska over the past decade"
1. First go to researcher
If "FINAL ANSWER"
Message
Message
Researcher
(Call a "search" function OR FINISH)
Router
(If statements based on agent output)
Chart Generator
(Code Execution😳)
If "continue" and state["sender"] == "Chart Generator"
If "continue" and state["sender'] == "researcher"
If function is called
If State["sender"] == "Researcher"
If State["sender"] == "Chart Generator"
Call_tool

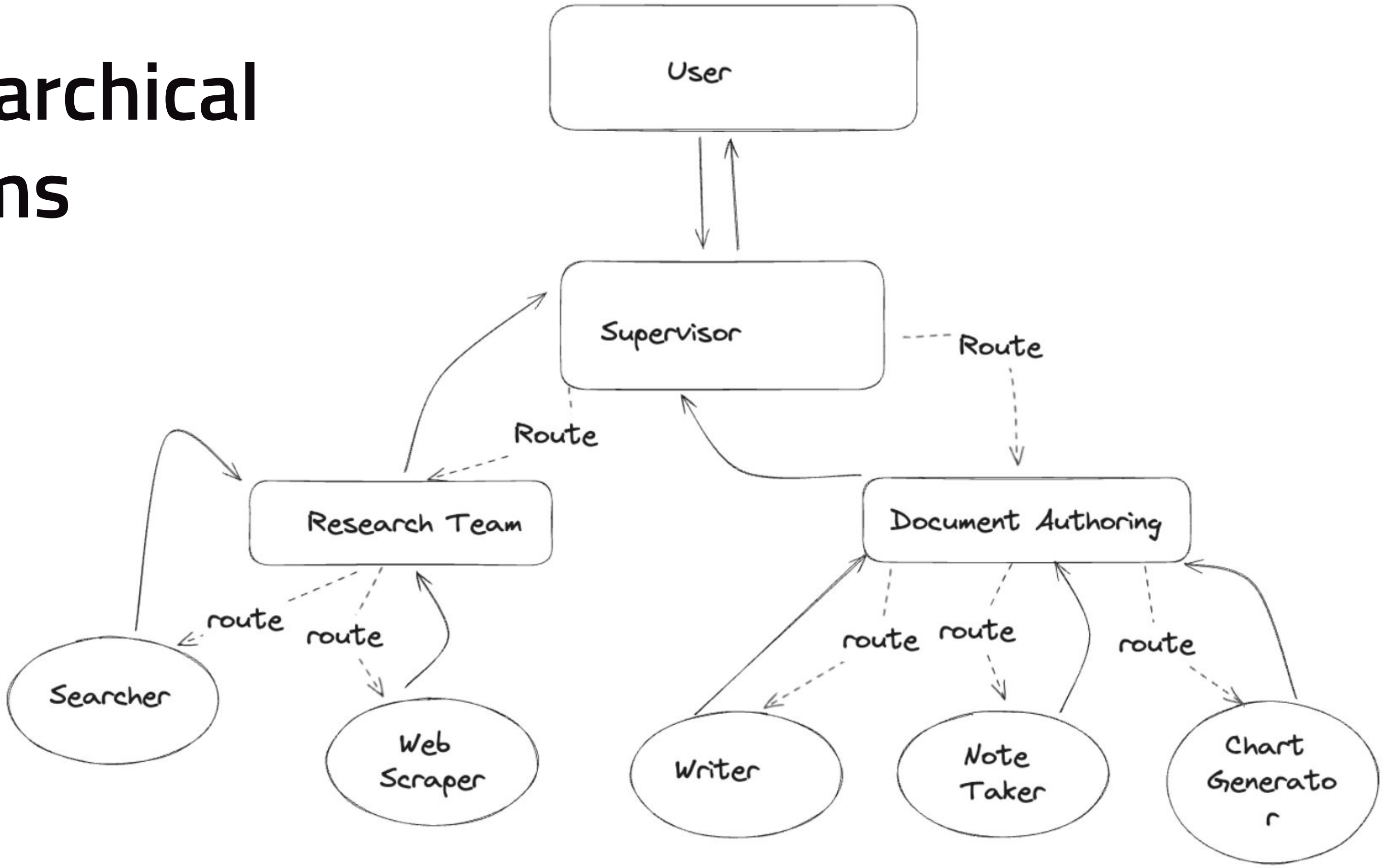
Accessed 4 June 2024, <https://github.com/langchain-ai/langgraph/tree/main/examples/multi_agent>.

Agent Supervisor



Accessed 4 June 2024, <https://github.com/langchain-ai/langgraph/tree/main/examples/multi_agent>.

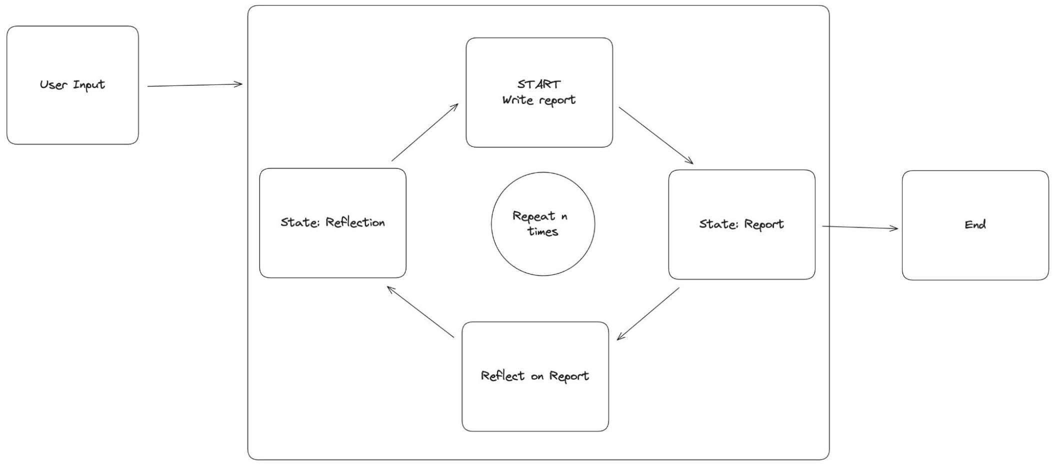
Hierarchical Teams



Accessed 4 June 2024, <https://github.com/langchain-ai/langgraph/tree/main/examples/multi_agent>.

Hands-On Homework

Objective: Code a AI graph that writes a report, provides feedback, and rewrites the report n number of times. You'll use LangGraph to build the nodes, track state, and produce a result.


User Input
START
Write report
State: Report
Reflect on Report
State: Reflection
Repeat n times
End