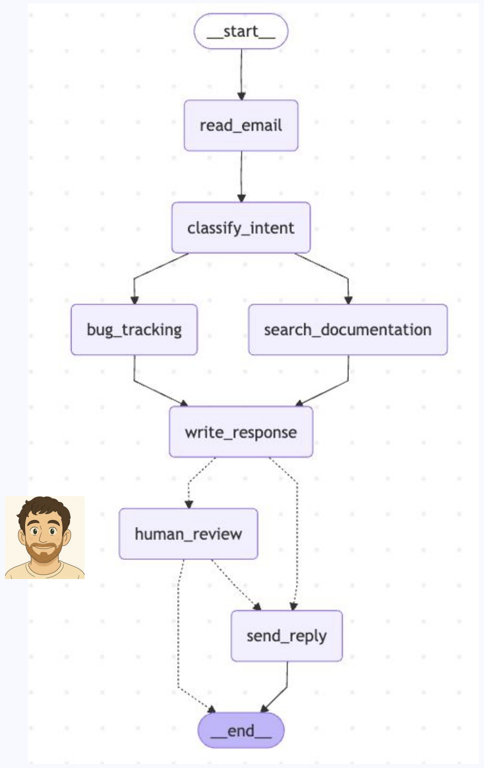
LangChain

LangGraph: StateGraph Essentials

Course Outline

1. LangGraph Orientation
2. LangGraph Foundations
3. Build an Application



LangGraph Orientation

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
    - Parallelization – to save actual latency

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
  - Parallelization – to save actual latency
  - Streaming – to save perceived latency

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
    - Parallelization – to save actual latency
    - Streaming – to save perceived latency
- Long-Running Agents can fail, which is expensive and time consuming

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
  - Parallelization – to save actual latency
  - Streaming – to save perceived latency
- Long-Running Agents can fail, which is expensive and time consuming
  - Checkpointing – to reduce the cost of each retry

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
  - Parallelization – to save actual latency
  - Streaming – to save perceived latency
- Long-Running Agents can fail, which is expensive and time consuming
  - Checkpointing – to reduce the cost of each retry
- The non-deterministic nature of AI requires checkpoints, approvals, and testing

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
  - Parallelization – to save actual latency
  - Streaming – to save perceived latency
- Long-Running Agents can fail, which is expensive and time consuming
  - Checkpointing – to reduce the cost of each retry
- The non-deterministic nature of AI requires checkpoints, approvals, and testing
  - Human-in-the-loop - to collaborate with the user

https://blog.langchain.com/building-langgraph

LangGraph

Agents and LLM applications have these challenges

- Latency in the seconds vs ms
  - Parallelization – to save actual latency
  - Streaming – to save perceived latency
- Long-Running Agents can fail, which is expensive and time consuming
  - Checkpointing – to reduce the cost of each retry
- The non-deterministic nature of AI requires checkpoints, approvals, and testing
  - Human-in-the-loop - to collaborate with the user
  - Tracing, Observation and Evaluation (LangSmith)

https://blog.langchain.com/building-langgraph

Layer Diagram

Your Application

StateGraph SDK

Functional SDK

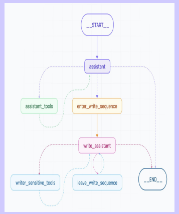
PregelLoop - Message Passing Runtime

LangGraph

LangChain

Hosted on LangSmith Deployments

Studio



LangSmith

Debugging

Playground

Prompt Management

Annotation

Testing

Monitoring

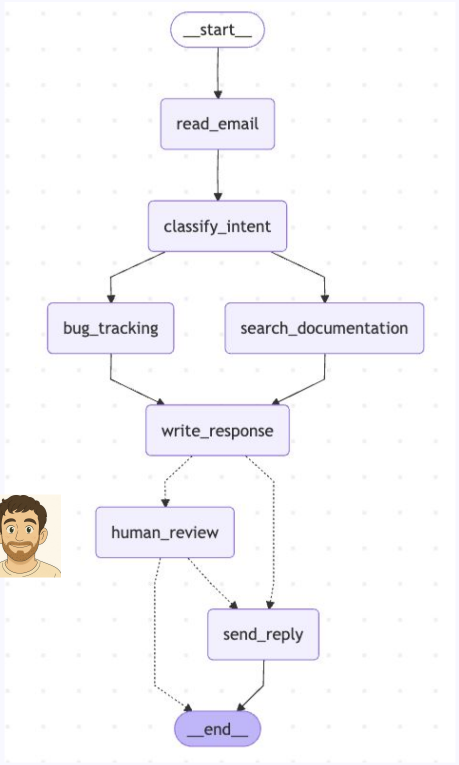
COMMERCIAL

Course Outline

1 LangGraph Orientation

2 LangGraph foundations

3 Build an Application

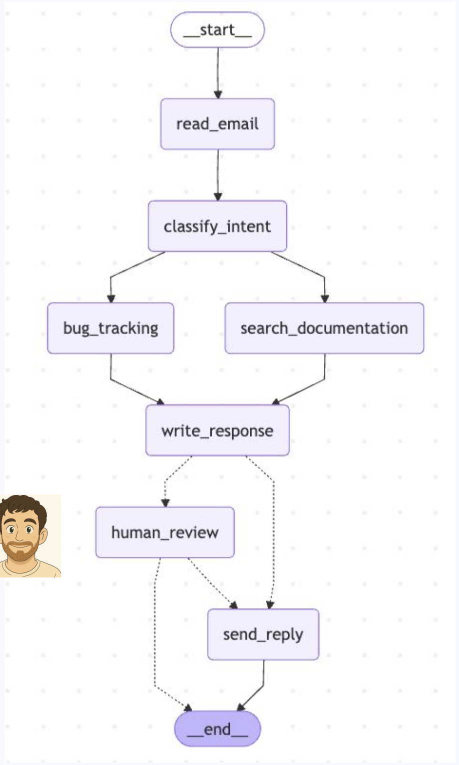


Course Outline

1 LangGraph Orientation

2 LangGraph foundations

3 Build an Application



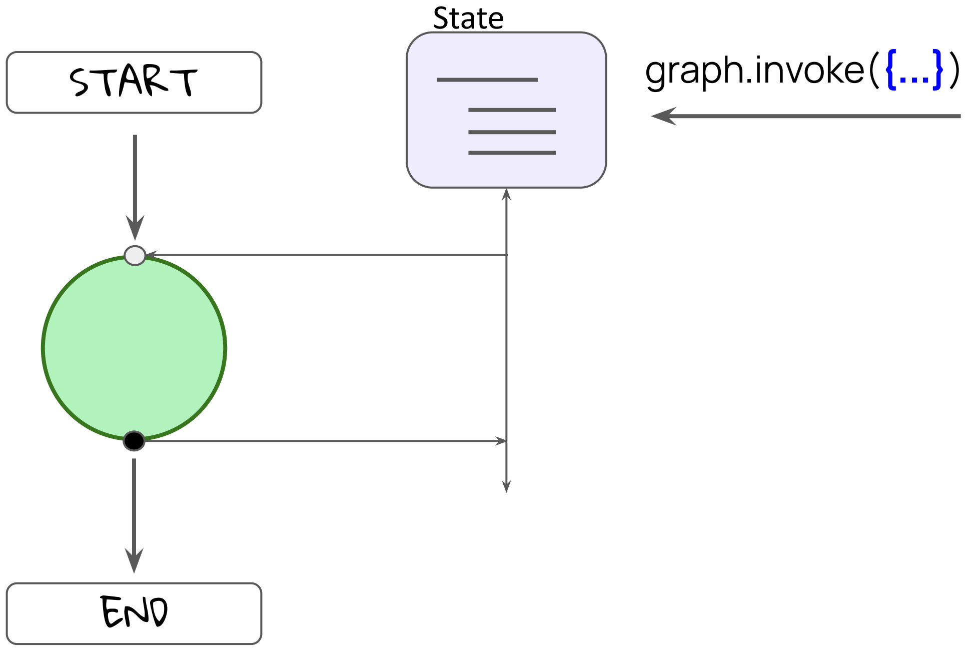
LangGraph: StateGraph

Components and Capabilities

- State: Data
- Node: Functions
- Edges: Control Flow
    - Serial, Parallel
    - Conditional
- Checkpointing/Memory
- Human In the Loop: Interrupts

Nodes and State

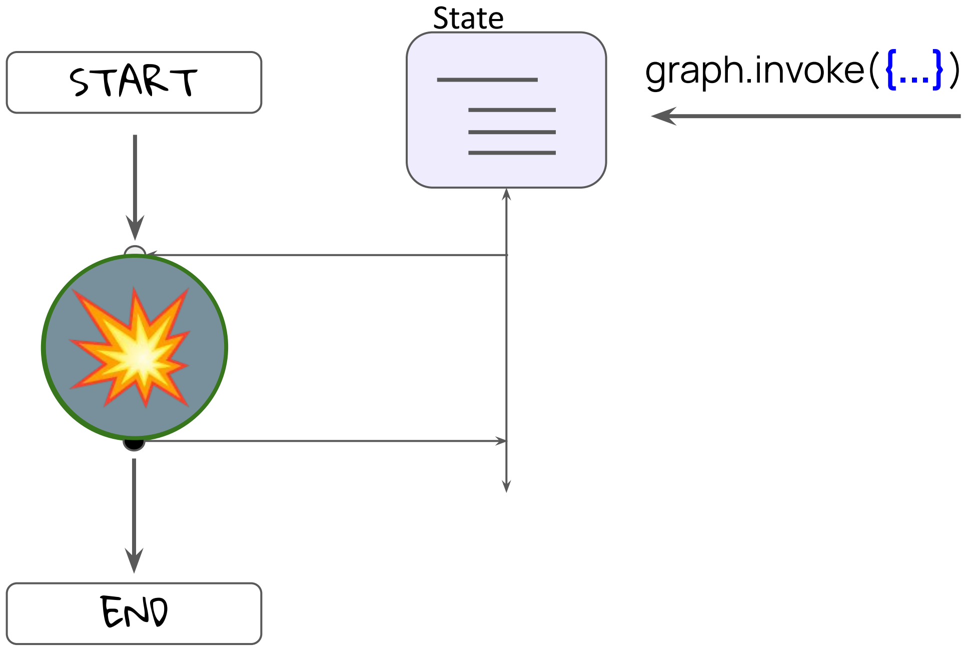
State, Nodes



```
class State(TypedDict):
    nlist : List[str]
```

```
def node_a (state: State):
    ...
    return({"nlist": [note]})
```

State, Nodes



```
class State(TypedDict):
    nlist : List[str]
```

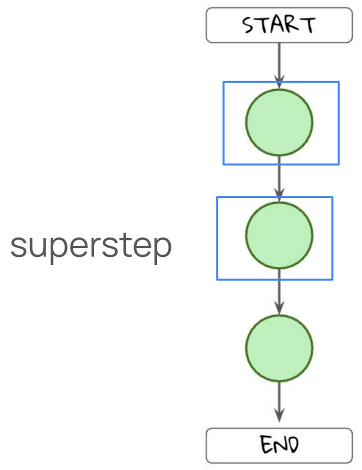
```
def node_a (state: State):
   ...
    return({"nlist": [note]})
```

Edges

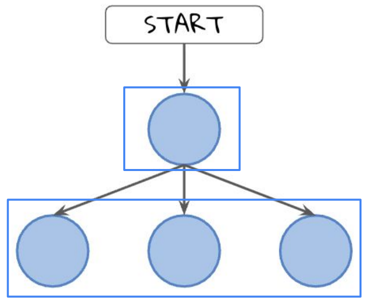
Edges: Control Flow

Edge

Serial

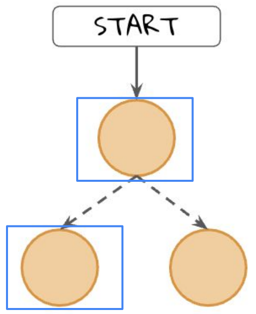


Parallel

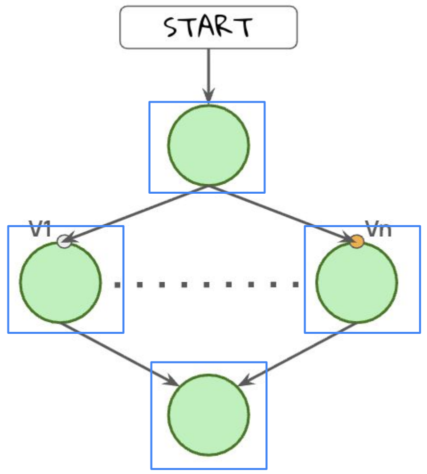


Conditional Edge

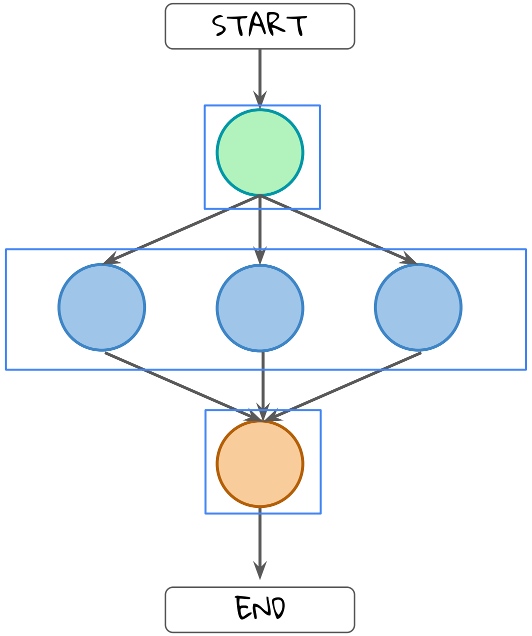
Conditional



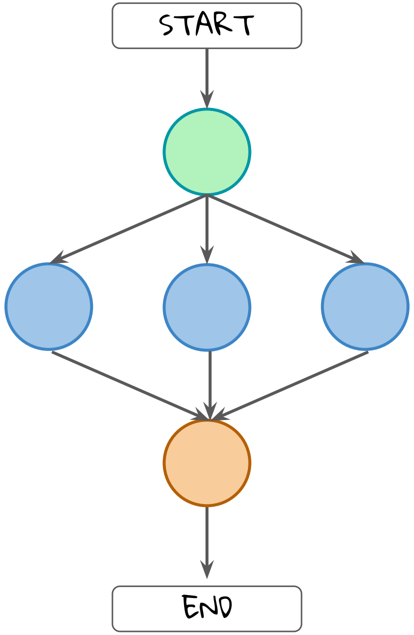
Map-Reduce

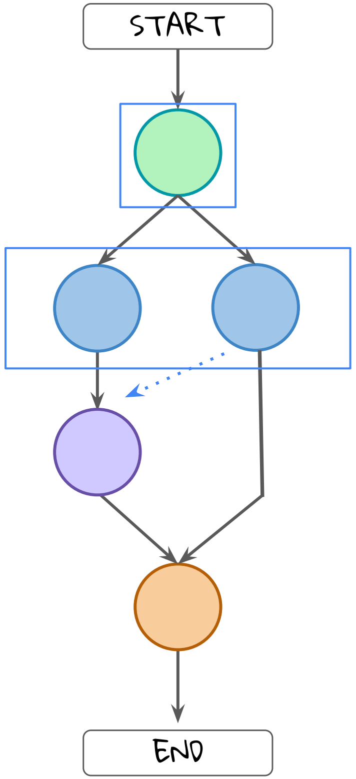


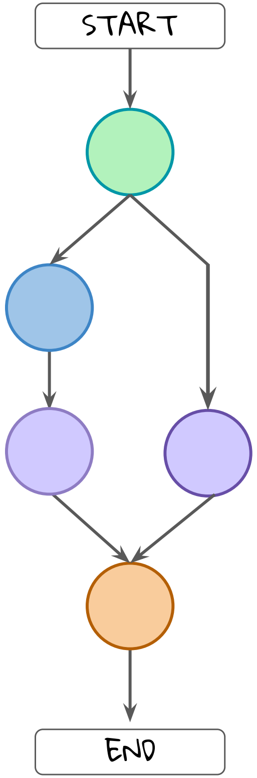
Super Steps



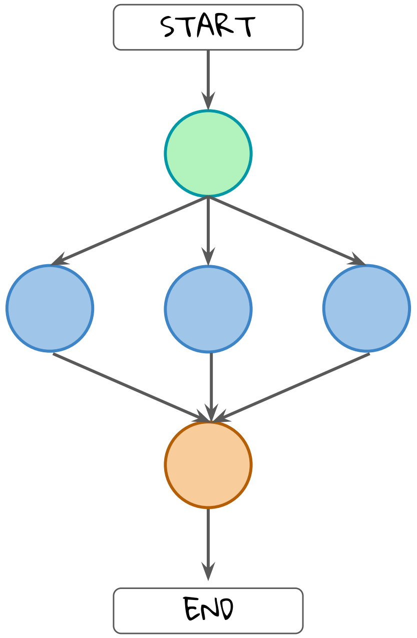
Super Steps

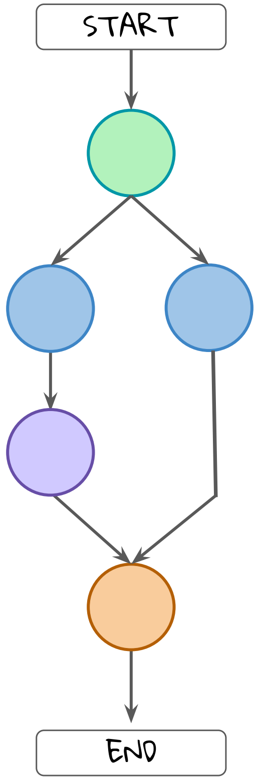


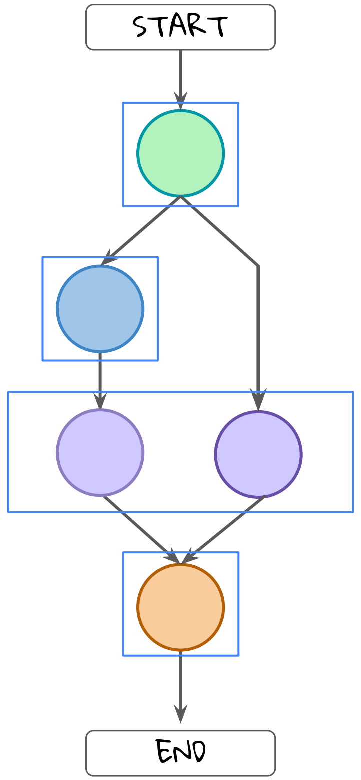




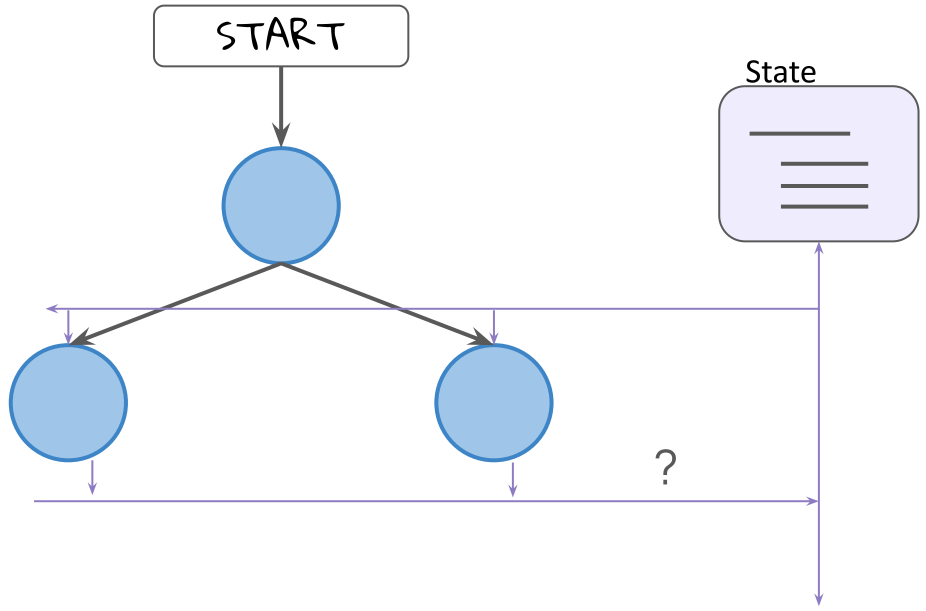
Super Steps





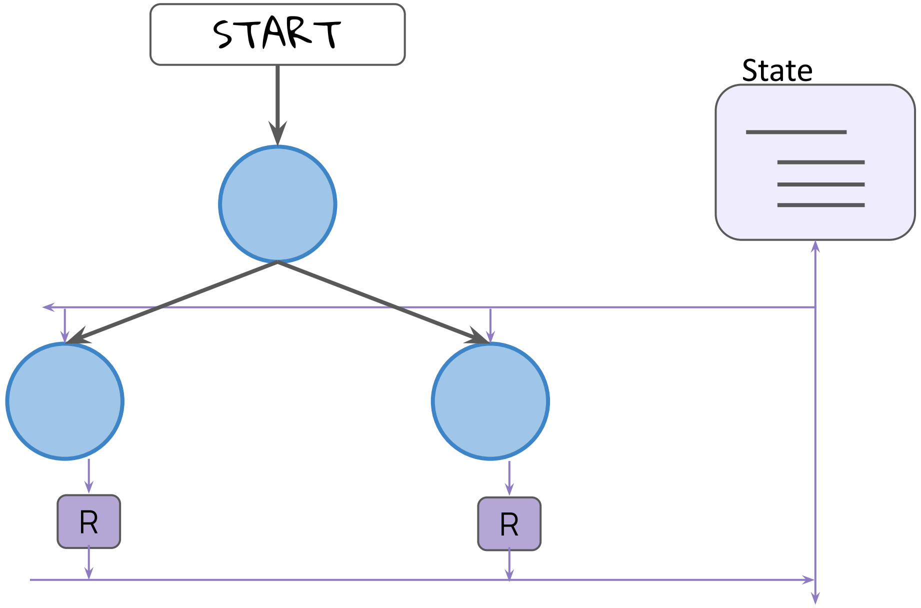


Reducers

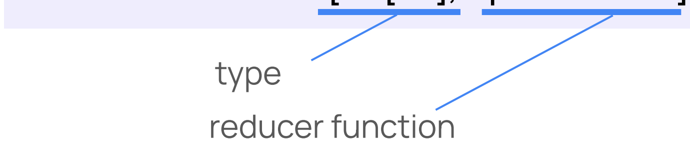


```
class State(TypedDict):
    nlist : List[str]
```

Reducers



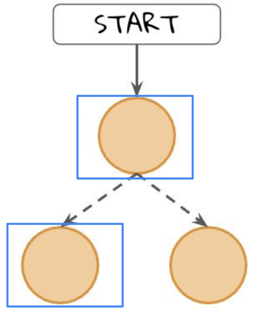
```
class State(TypedDict):
    nlist : Annotated[list[str], operator.add]
```



Conditional Edges

Edges: Control Flow

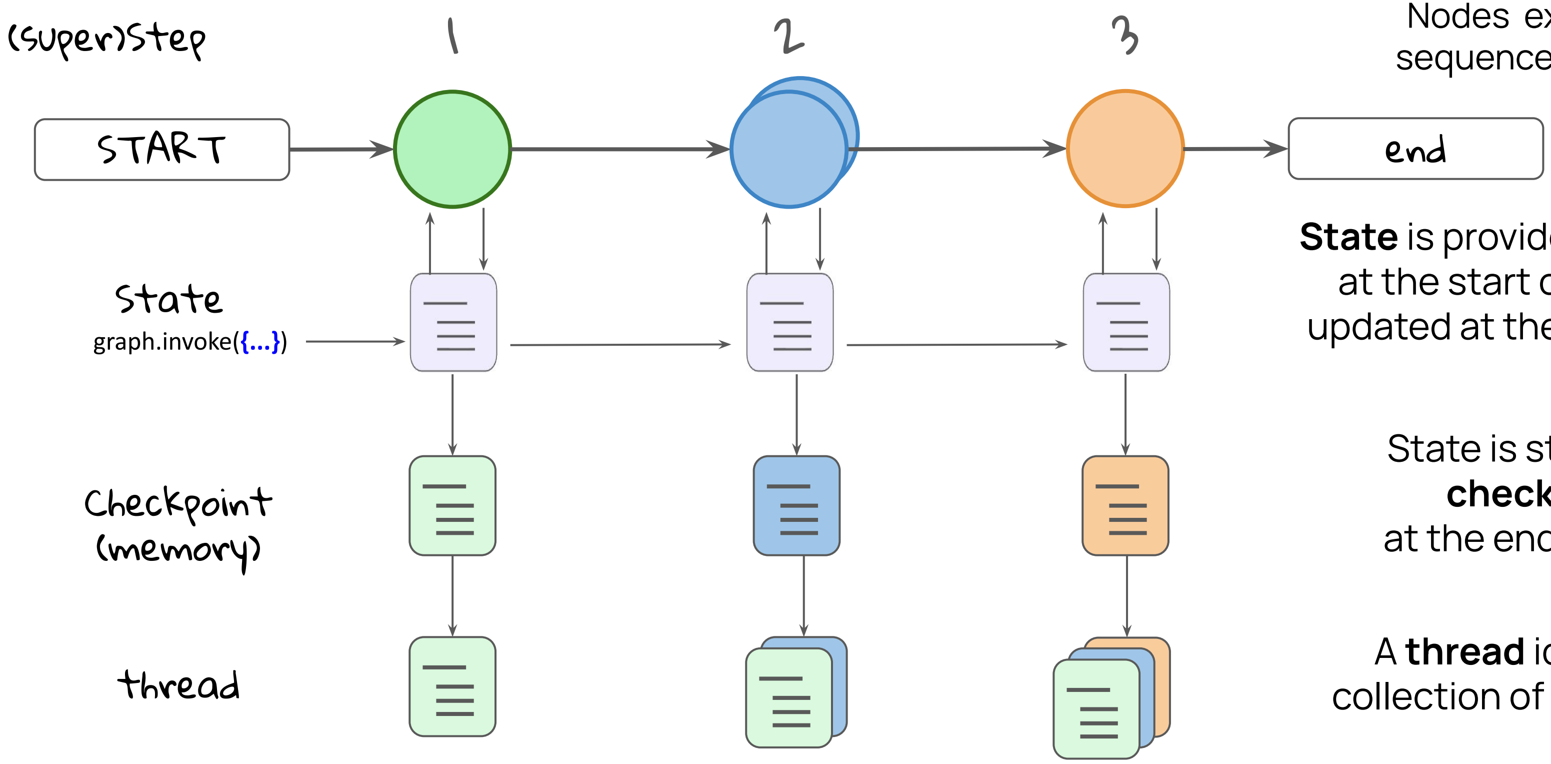
Conditional



- Conditional Edge
- Comman

Memory

Memory / Checkpointers



Nodes execute in sequence, per step

State is provided to the node at the start of a step and updated at the end of a step

State is stored to a **checkpoint** at the end of a step

A **thread** identifies a collection of checkpoints

Memory

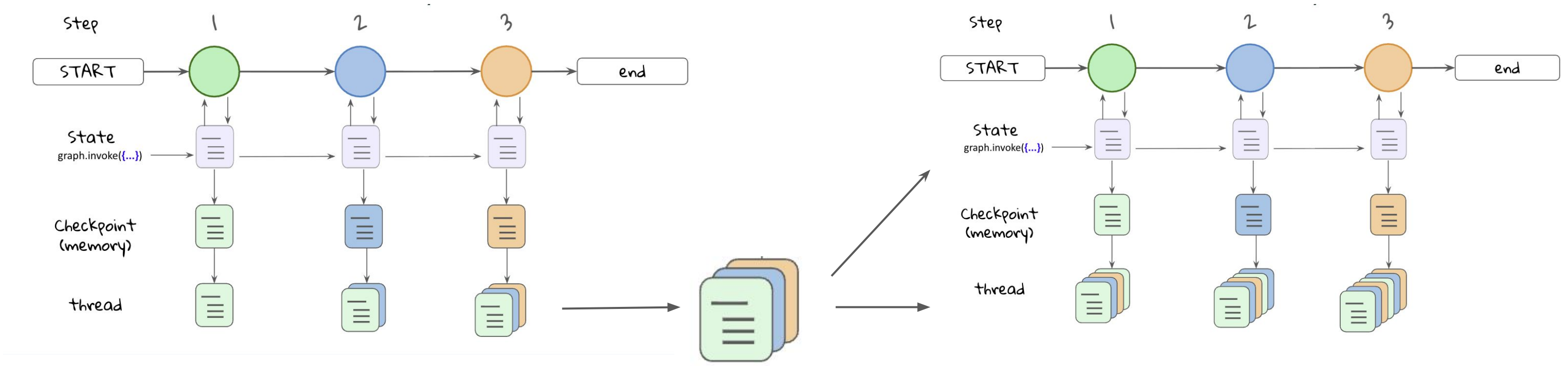
Benefits

- **Recover gracefully from failures** — resume without losing progress.
- **Time travel** — roll back to a known good point and continue forward.

Memory / Checkpointers

Benefits

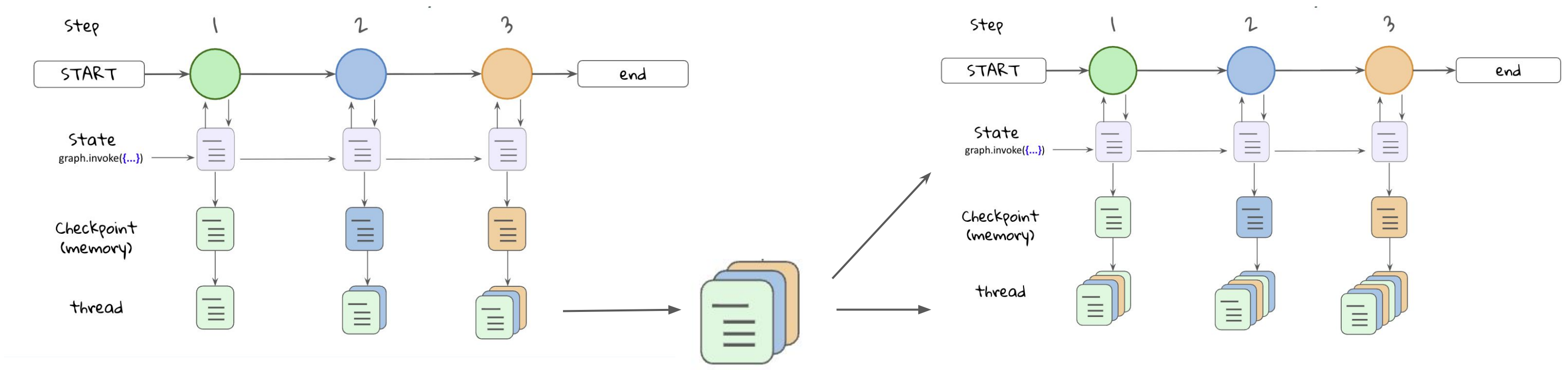
- **Recover gracefully from failures** — resume without losing progress.
- **Time travel** — roll back to a known good point and continue forward.
- **Persistent state** — data is preserved even when the graph is not running.



Memory / Checkpointers

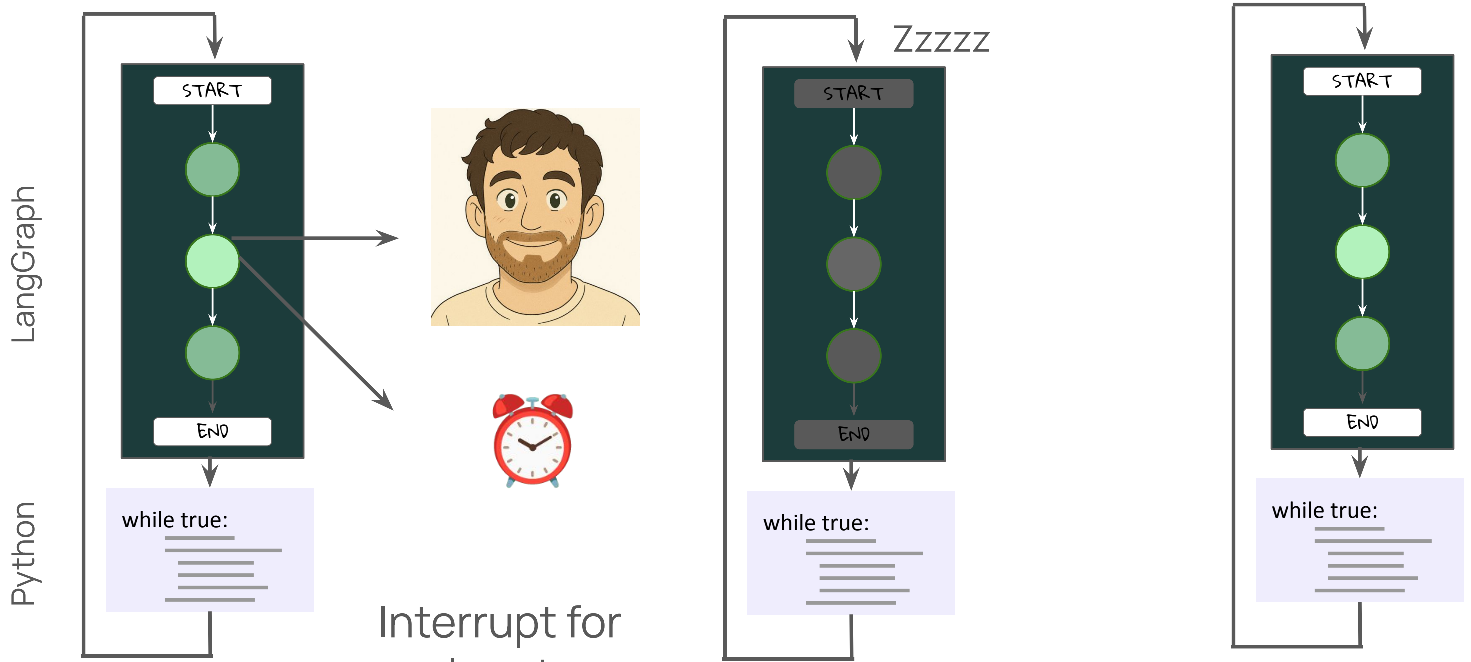
Benefits

- **Recover gracefully from failures** — resume without losing progress.
- **Time travel** — roll back to a known good point and continue forward.
- **Persistent state** — data is preserved even when the graph is not running.
- **Restore state at any step** — pick up execution from where you left off.



Interrupts

Human In the Loop: Interrupt



Interrupt for Input

Suspend while waiting

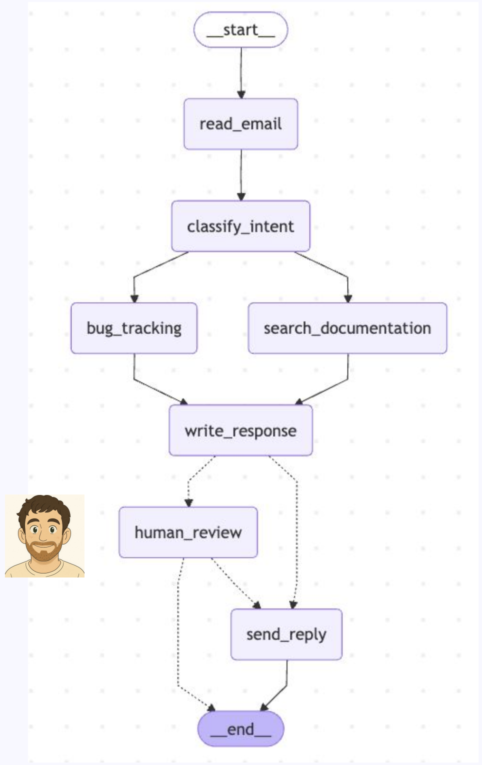
Resume

Course Outline

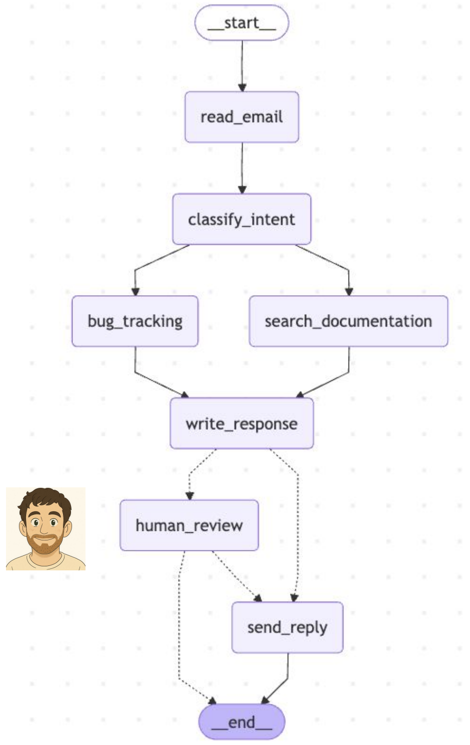
1 LangGraph Orientation

2 LangGraph foundations

3 Build an Application



Email Support Workflow



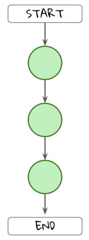
Simulate a email customer support workflow

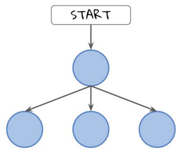
Focus on LangGraph aspects:

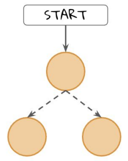
- State, Nodes,
- Edges
  - Serial
  - Parallel
  - Conditional
- Memory
- Interrupt

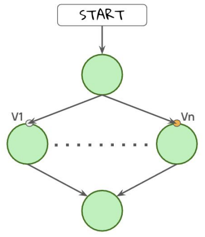
Conclusion

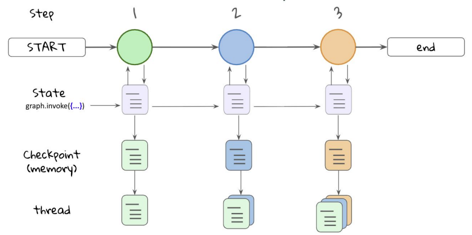
Congratulations!

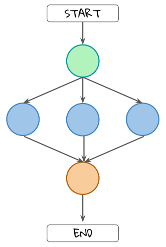


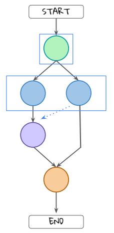


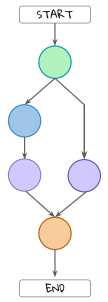


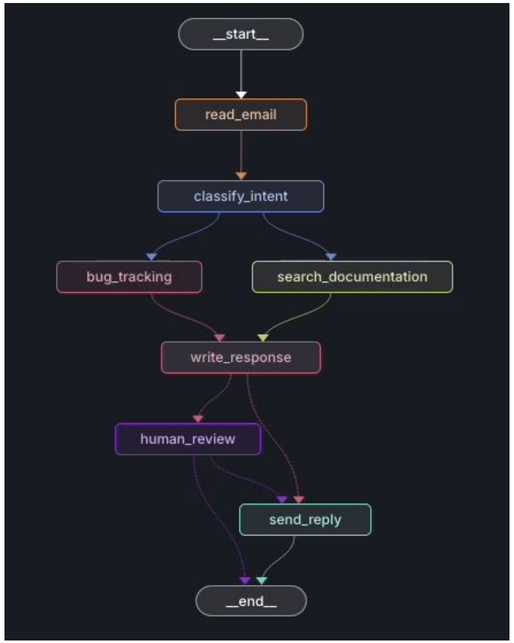



Step
1
2
3
START
end
State
graph.invoke({...})
Checkpoint (memory)
thread








__start__
read_email
classify_intent
bug_tracking
search_documentation
write_response
human_review
send_reply
__end__