REACT AGENT, MULTI-AGENT COLLABORATION, INTRO TO LANGRAPH

initialzie_agent

vs

- create_react_agent
- and AgentExcecutor

Swiggy

REACT DESIGN PATTERN

ReAct = Reasoning + Acting

It’s a prompting and control-flow pattern where an LLM doesn’t just “answer questions” —
it reasons about what to do, decides on an action, executes it (using tools or APIs), observes the result, and then repeats the loop until it reaches a final answer.

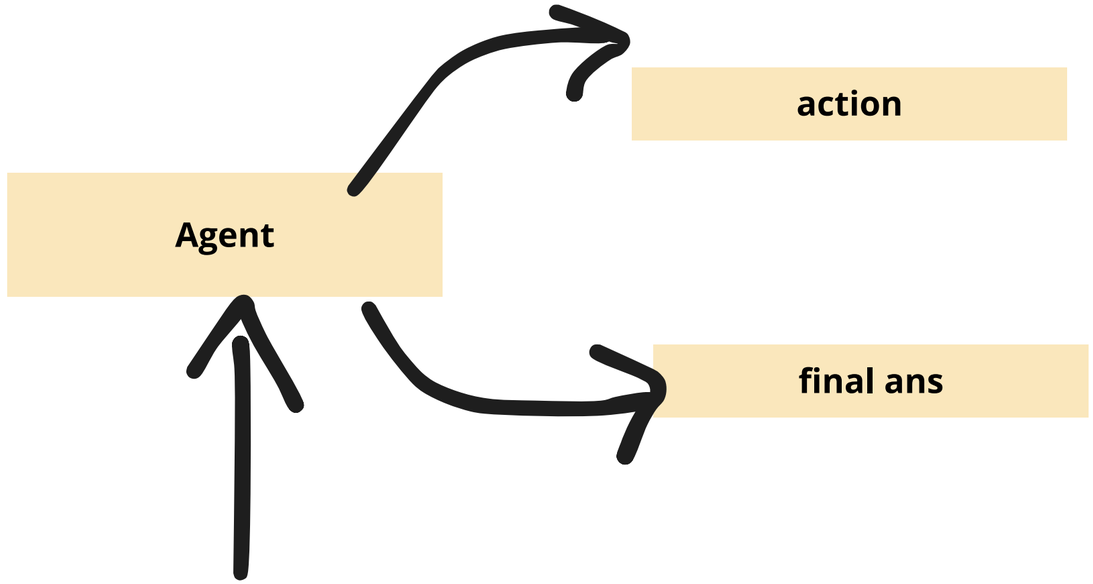
COMPONENTS

Question / Input	User provides a query.
Thought	The LLM reasons about what to do next (internal logic).
Action	The LLM decides which tool to use (e.g., search, calculator).
Action Input	The LLM specifies what input to give the tool.
Observation	The system executes that tool and returns the result.
(Repeat)	The LLM sees the Observation, updates its Thought, maybe picks another Action.

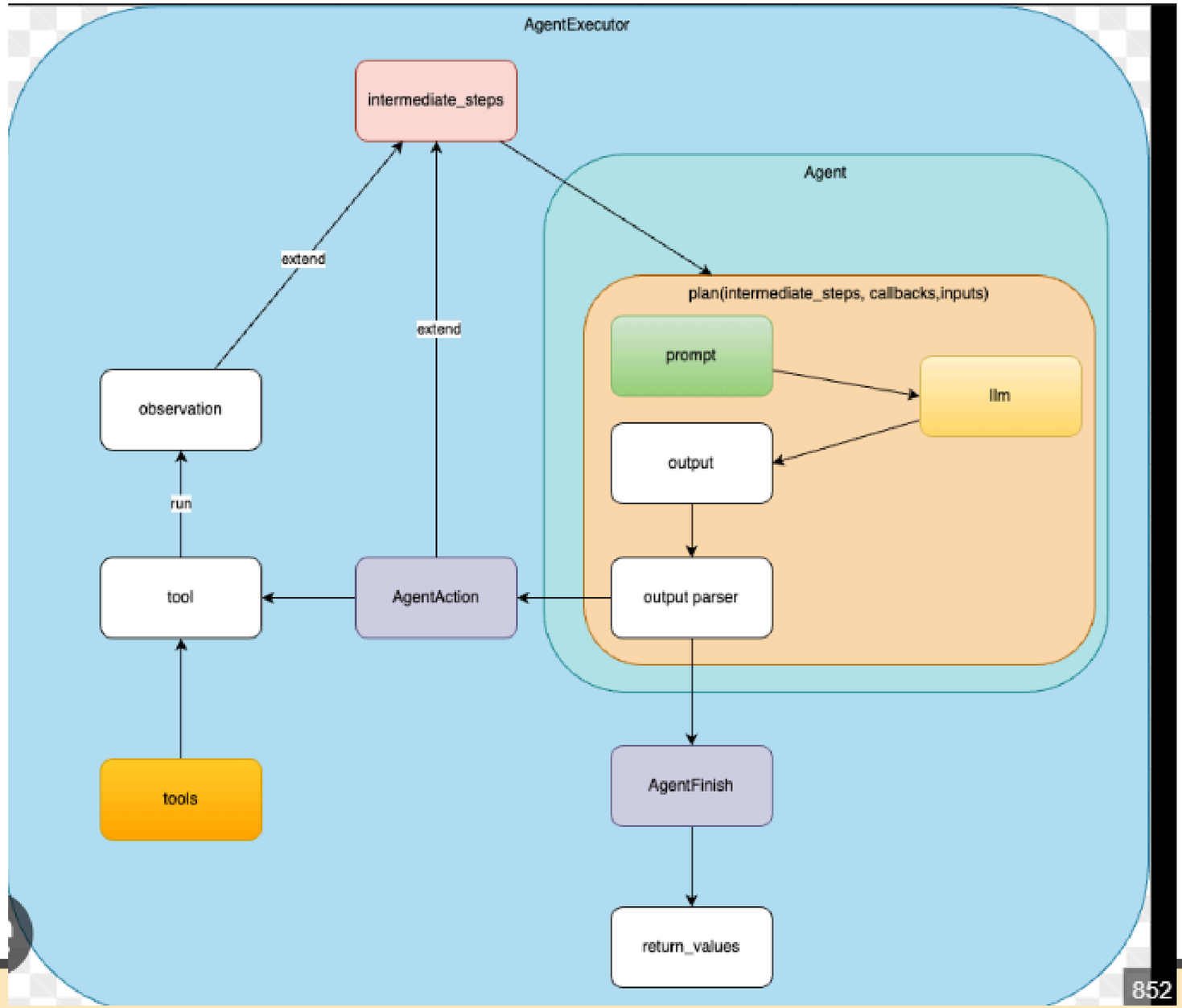
LETS TAKE AN EXAMPLE

can u tell me the population of the captital of France

HOW DOES AGENT AND EXEC



AGENT IN ACTION



LANGCHAIN LIMITATION

No control over flow

The agent can decide to keep looping or call tools in weird orders. You can’t easily say: “first do X, then Y.”

No multi-agent orchestration

If you have 2–3 agents (researcher, math, jude), you have to manually route messages between them

LANGCHAIN VS LANGGRAPH

- FOR LLMS
- Linear Workflows
- Simple agents

- FOR COMPLEX WORKFLOWS
- For multi-step, graphs
- Multi-agent systems
- complex apps

LANGGRAPH WUT?

LangGraph is a stateful graph-based orchestration framework **built on top of LangChain** that enables the construction of deterministic, multi-step, and multi-agent workflows for Large Language Model (LLM) applications.

It models workflows as graphs of nodes (each representing an agent, tool, or chain) connected by edges that define explicit control flow and data dependencies, allowing for state persistence, conditional routing, looping, and fault-tolerant execution.

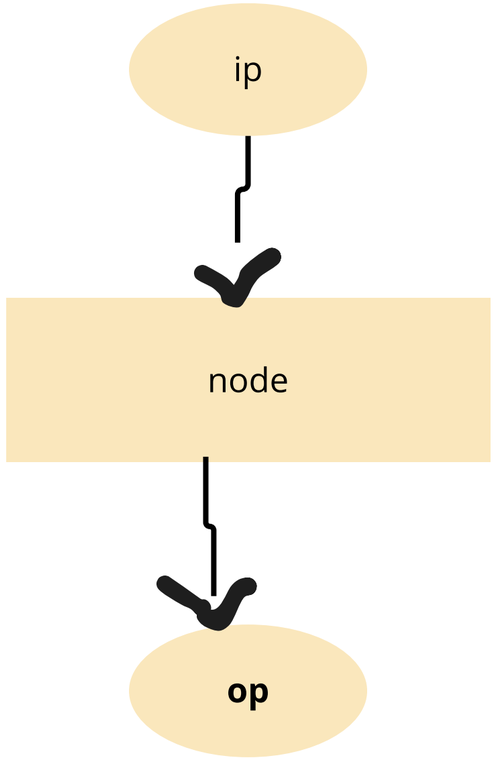
LANGGRAPH COMPONENTS

Node : Recives an ip, processes it, produces an op
Edge: workflow direction, which node goes where?
State: When you build a LangGraph app, each node (like a ResearchAgent, MathAgent, or Summarizer) can read from and write to this shared memory.
That memory is called the state.

VERY SIMPLE GRAPH

Hello, world :

(no llm)



GRAPH W MULTIPLE IPS

(no llm)

Hello, world :