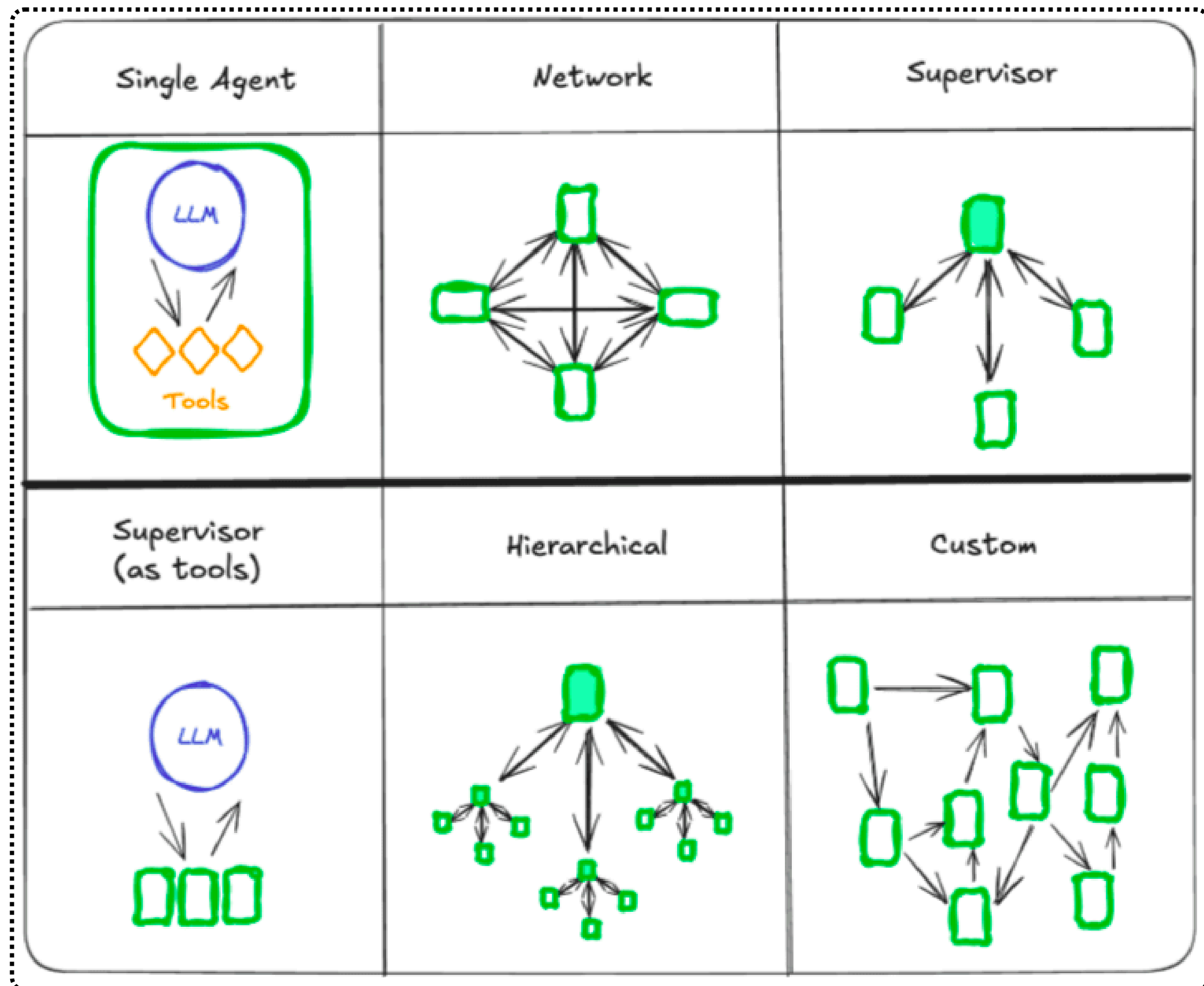


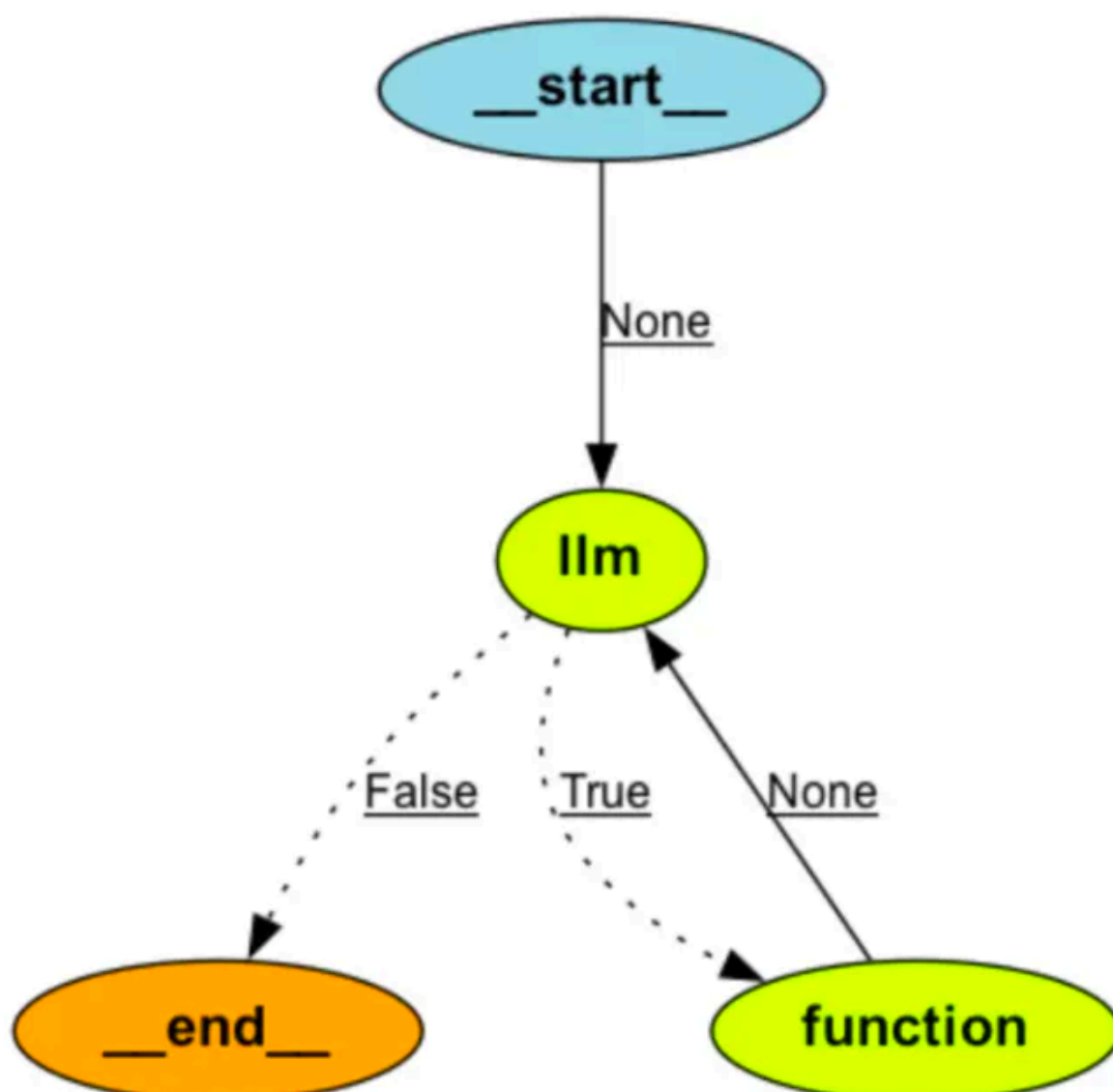
LangGraph Tutorial for Beginners



Understanding LangGraph

LangGraph is a powerful library, which is a part of LangChain tools. It helps streamline the integration of LLMs, ensuring they work together seamlessly to understand and execute tasks. It offers a neat way to build and handle LLM apps with many agents. LangGraph lets developers set up how multiple LLM agents talk to each other. It shows these workflows as graphs with cycles.

This helps in keeping the communication smooth and performing complex tasks well. LangGraph is best when using Directed Acyclic Graphs (DAGs) for straight line tasks. But since it is cyclic and adds the ability to loop back, it allows for more complex and flexible systems. It's like how a smart agent might rethink things and use new information to update responses or change its choices.



Key Concepts of LangGraph

Here are some of the key concepts of LangGraph that you need to know:

1. Graph Structures

LangGraph's core idea is using a graph for the application's workflow. This graph has two main parts – nodes and edges.

- **Nodes:** Nodes are the fundamental building blocks representing discrete units of work or computation within the workflow. Each node is a Python function that processes the current state and returns an updated state. Nodes can perform tasks such as calling an LLM and interacting with tools or APIs for manipulating data.
- **Edges:** Edges connect nodes and define the flow of execution. They can be:
 - **Simple edges:** Direct, unconditional transitions from one node to another.
 - **Conditional edges:** Branching logic that directs flow based on node outputs, similar to if-else statements. This allows dynamic decision-making within the workflow.

2. State Management

Keeping track of what's happening is vital when you have many agents. All agents need to know the current status of the task. LangGraph handles this by managing the state automatically. The library keeps track of and updates a main state object. It does this as the agents do their jobs. The state object holds important information. It's available at different points in the workflow. This could include the chat history.

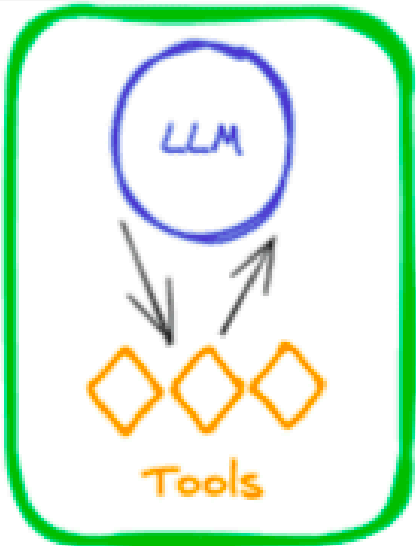
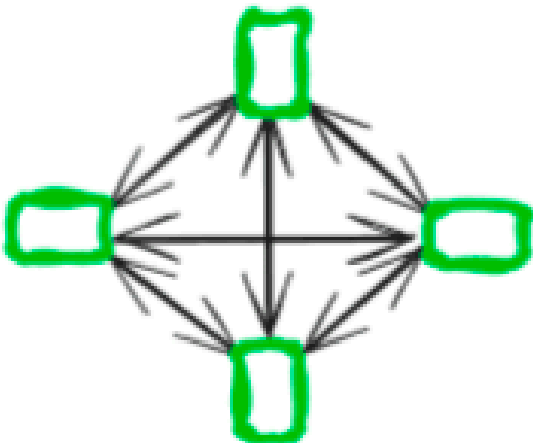
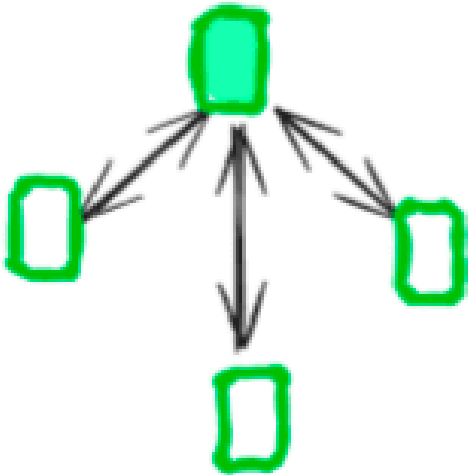
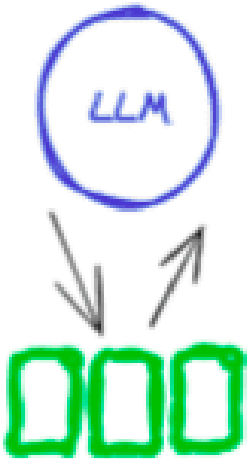
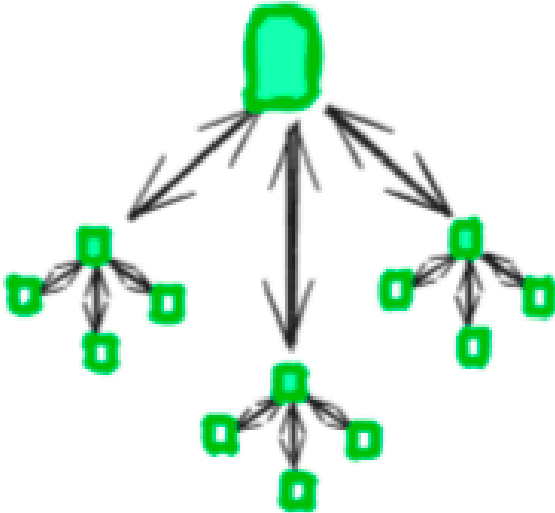
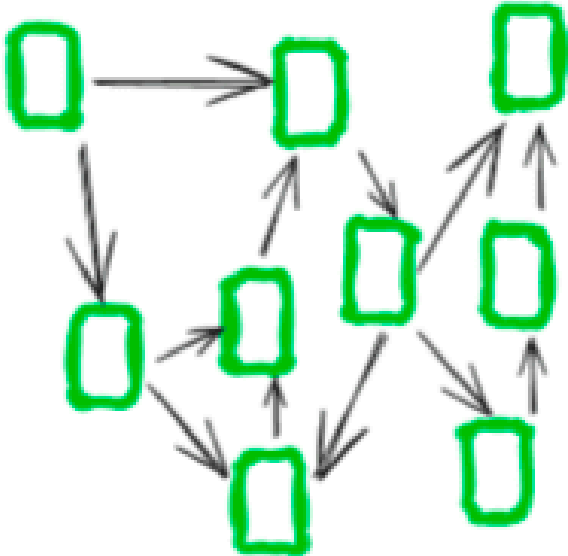
In a chatbot, the state can save the conversation. This helps the bot respond using what was said before. It can also store context data, like user likes, past actions, etc. or external data. Agents can use this for making choices. Internal variables can also be kept here. Agents might use the state to track flags, counts, or other values. These help guide their actions and decisions.

3. Multi-agent Systems

A multi-agent system consists of multiple independent agents that work together or compete to achieve a common goal. These agents use LLMs to make decisions and control the flow of an application. The complexity of a system can grow as more agents and tasks are added. This may lead to challenges like poor decision making, context management, and the need for specialization. A multi-agent system solves these problems by breaking the system into smaller agents, each focusing on a specific task, such as planning or research.

Architectures in Multi-agent Systems

Here are the various types of architectures followed in multi-agent systems.

Single Agent	Network	Supervisor
		
Supervisor (as tools)	Hierarchical	Custom
		

1. Network Architecture: In this architecture, every agent communicates with every other agent, and each can then decide which agent they should call next. This is very helpful when there is no clear sequence of operations. Below is a simple example of how it works using StateGraph.

For more information, kindly read this article



AI Agents

Beginner

Guide

Langchain

LangGraph Tutorial for Beginners

If you are new to LangGraph and wish to learn about it, then this beginner's guide and hands-on tutorial is the best free content for you.