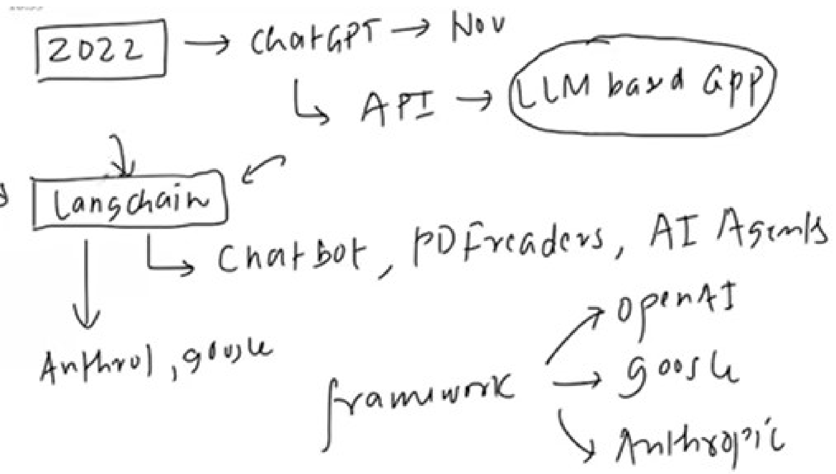
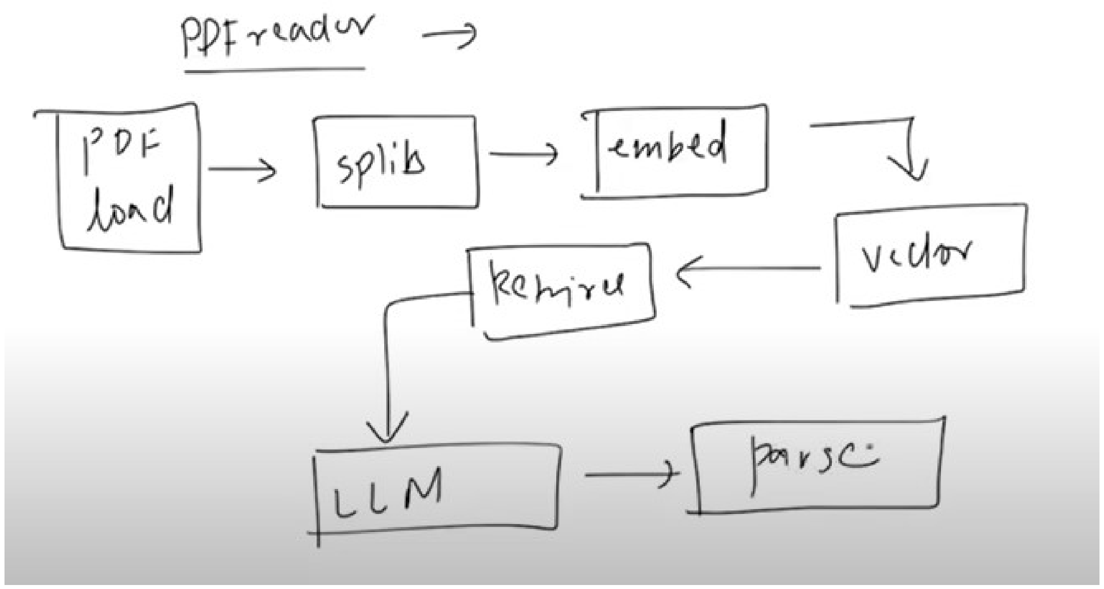
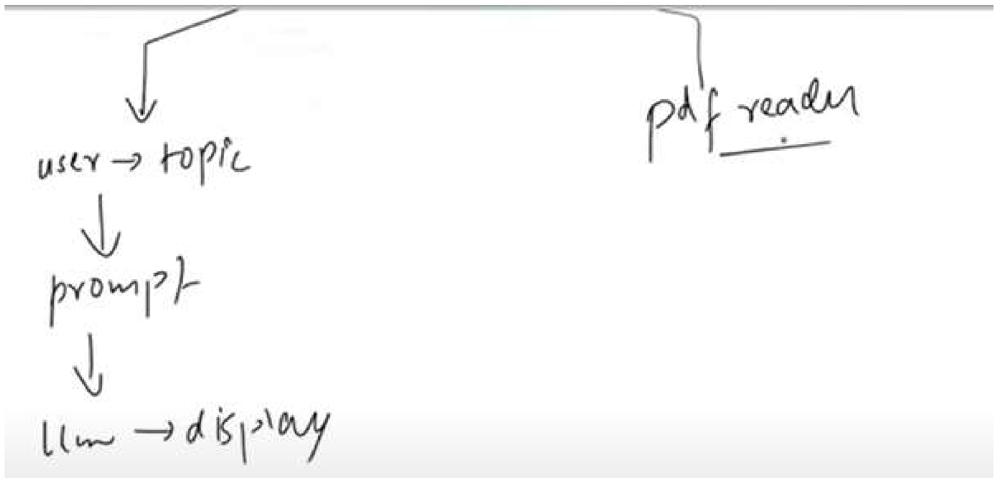
Why runnable

- Chains are built with the help of Runnables.
- Runnable concept is necessary to run the chain effectively.

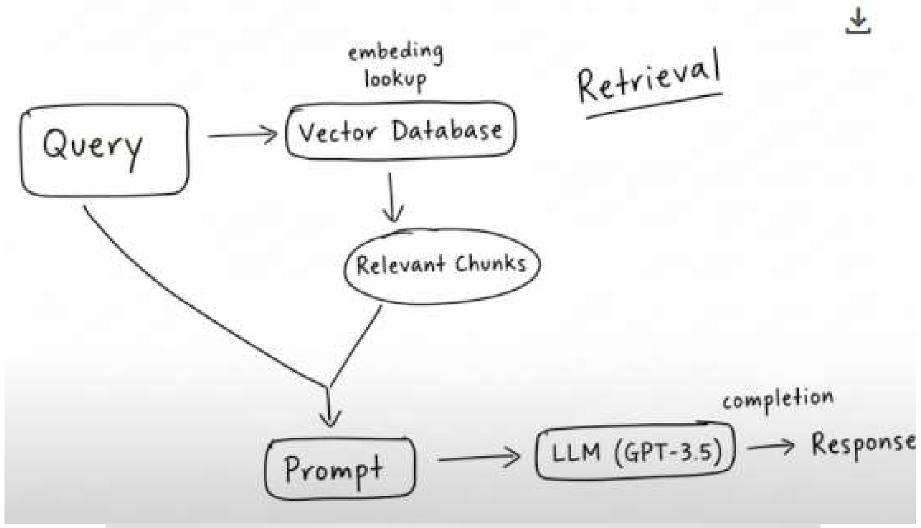


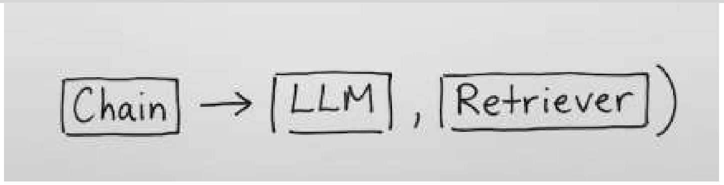
Wrote function for each component




user → topic
prompt
llm → display
pdf reader

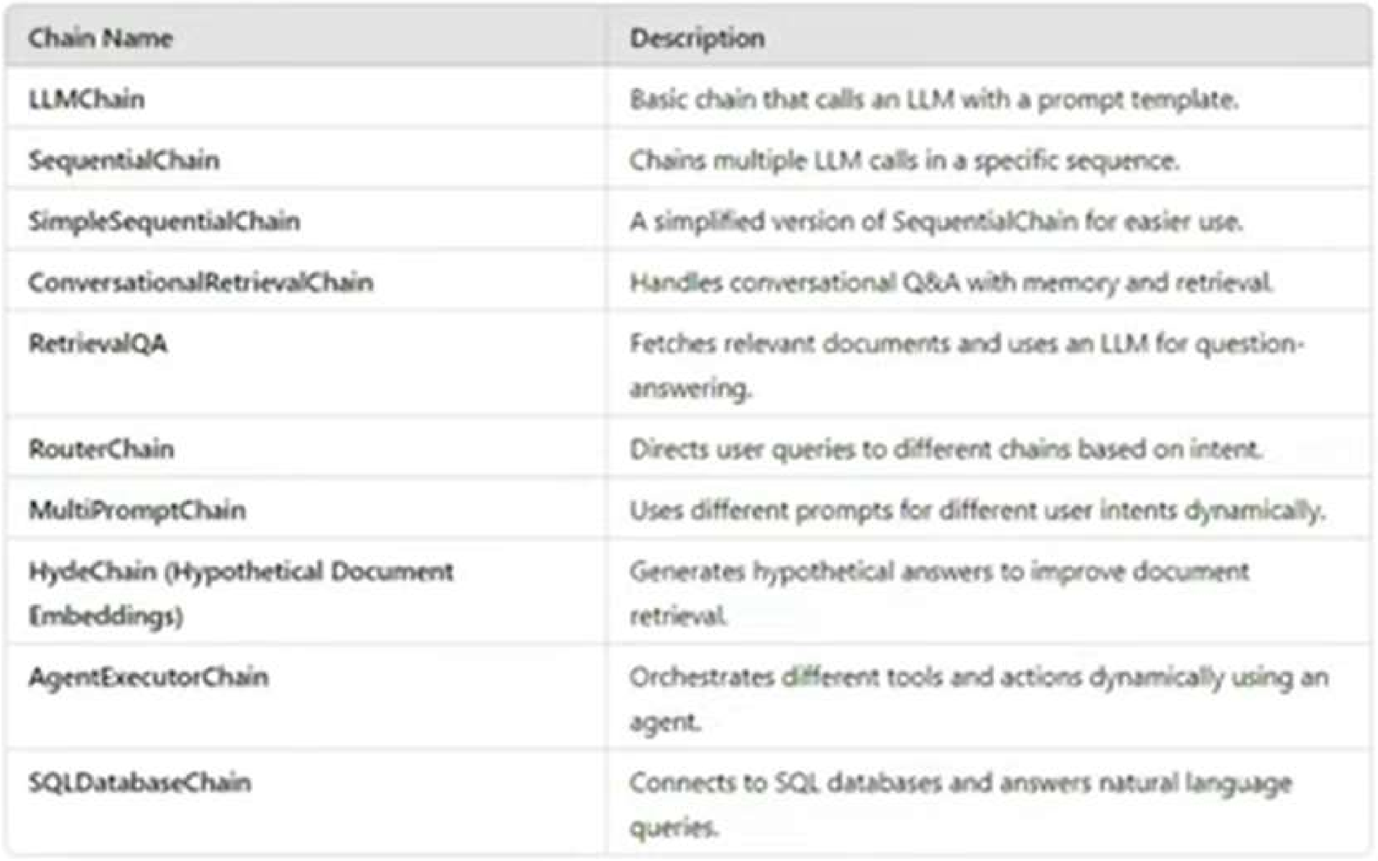
- Build the different components.
- Plugin in a frame to perform the task→ done manually
- What if task if we make a build in function → langchain called it chain
- E.g LLMChain(llm, prompt)





Reteriver QA Chain

Most popular chain



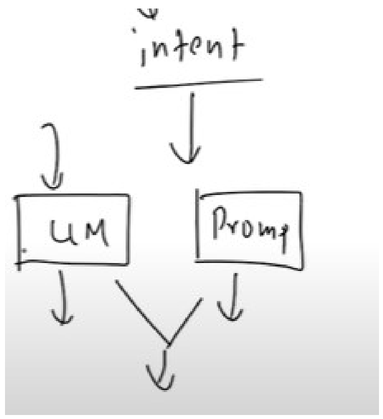
Chain Name	Description
LLMChain	Basic chain that calls an LLM with a prompt template.
SequentialChain	Chains multiple LLM calls in a specific sequence.
SimpleSequentialChain	A simplified version of SequentialChain for easier use.
ConversationalRetrievalChain	Handles conversational Q&A with memory and retrieval.
RetrievalQA	Fetches relevant documents and uses an LLM for question-answering.
RouterChain	Directs user queries to different chains based on intent.
MultiPromptChain	Uses different prompts for different user intents dynamically.
HydeChain (Hypothetical Document Embeddings)	Generates hypothetical answers to improve document retrieval.
AgentExecutorChain	Orchestrates different tools and actions dynamically using an agent.
SQLDatabaseChain	Connects to SQL databases and answers natural language queries.

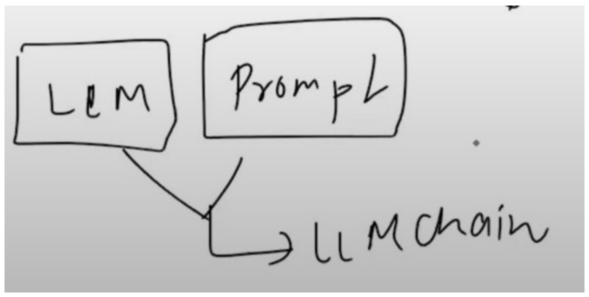
Problem with chains

- Too many chains
  - Lengthy code base
  - AI Engineer(difficult to memorize)

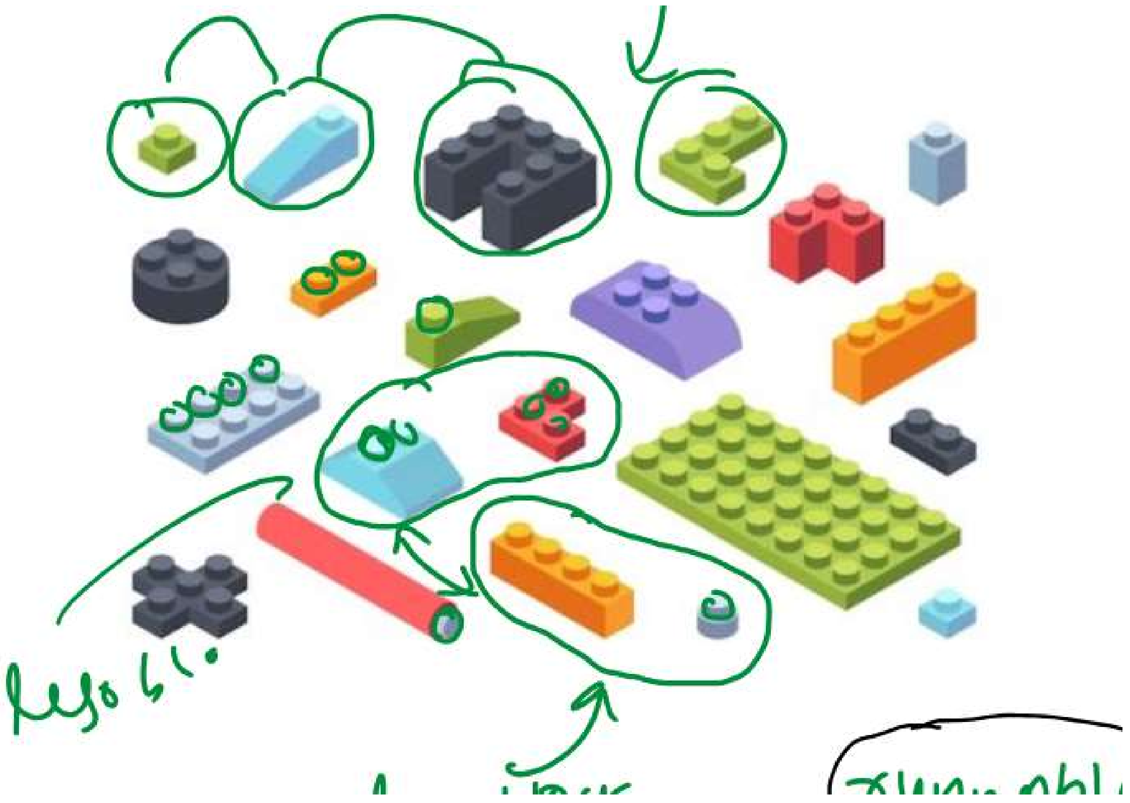
Intent
LLM+Prompt→

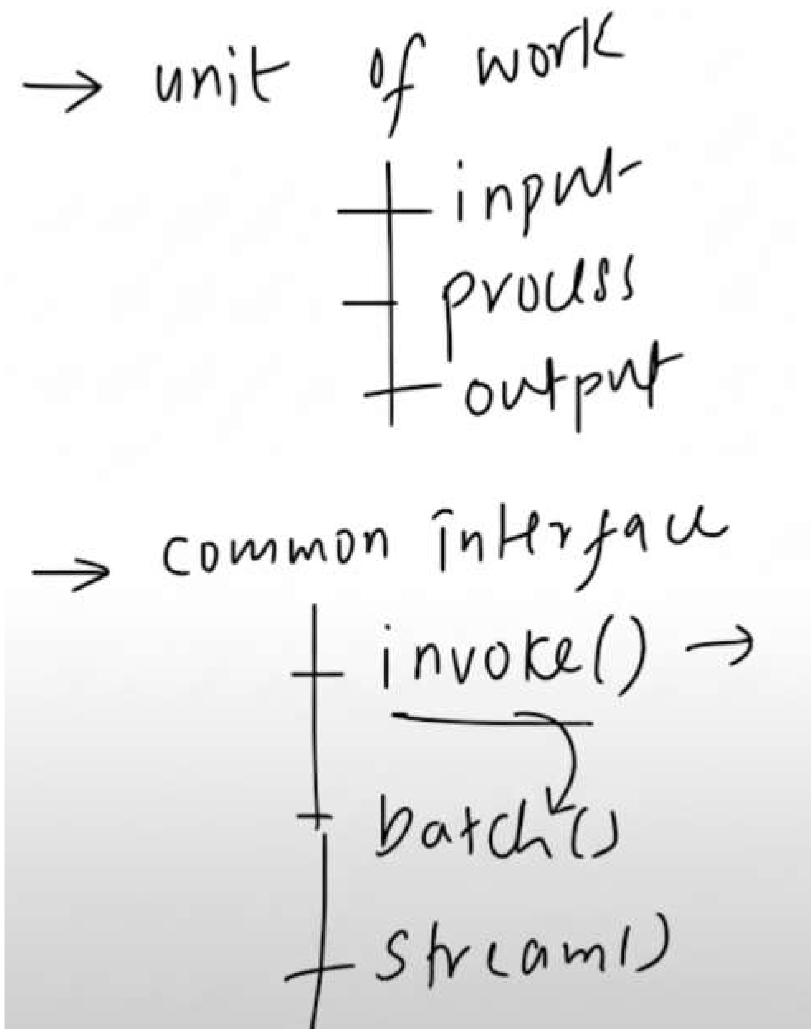
Chains: wrote a lot of functions





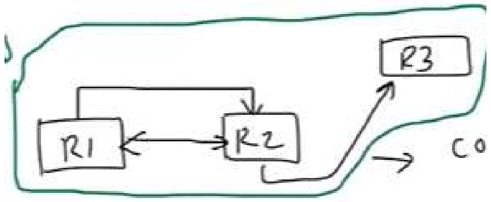
The What

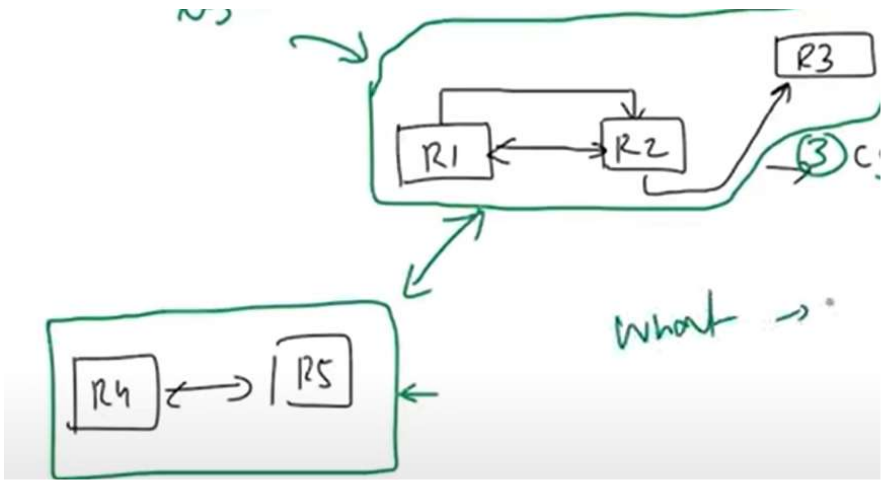


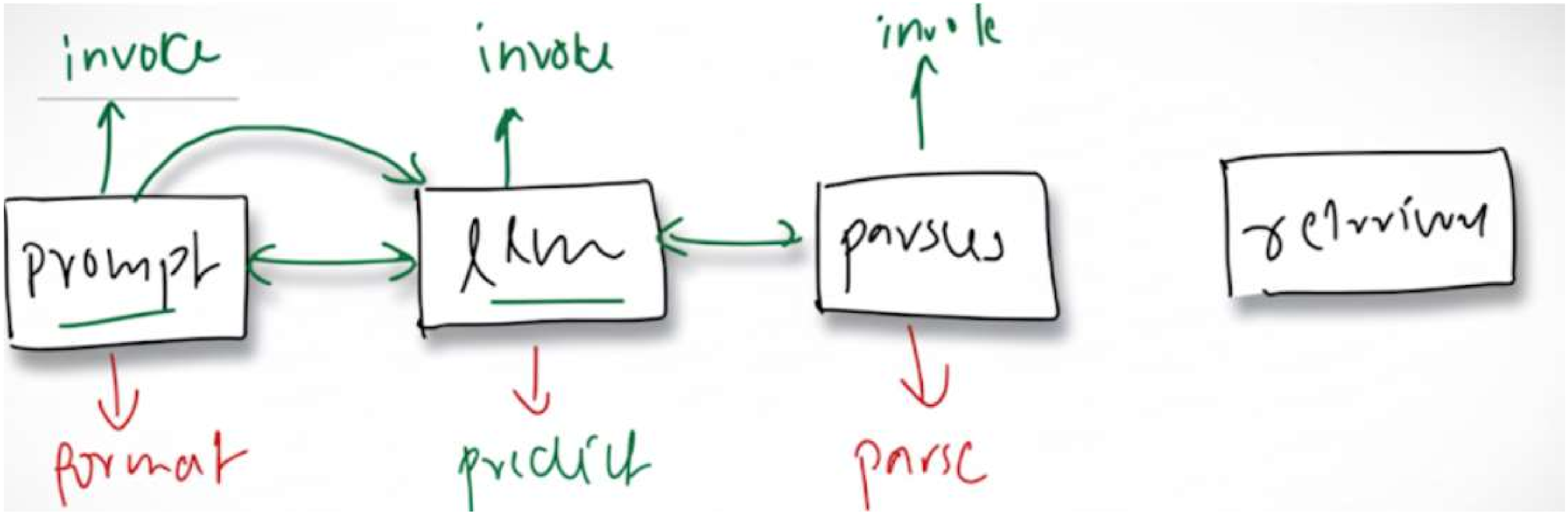

→ unit of work
- input
- process
- output

→ common interface
- invoke() →
- batch()
- stream()

- Below its self a runnable

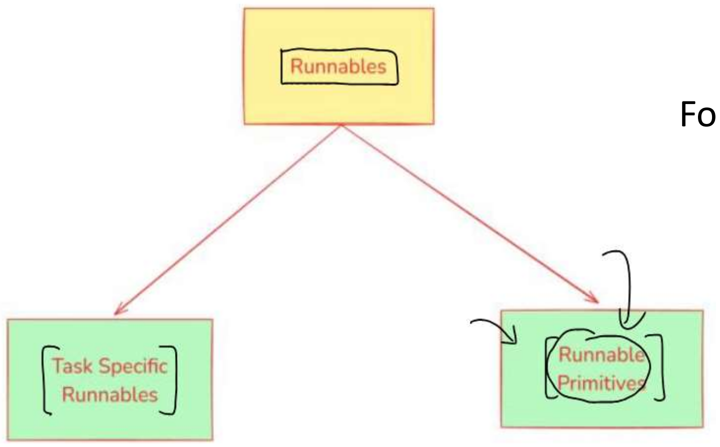




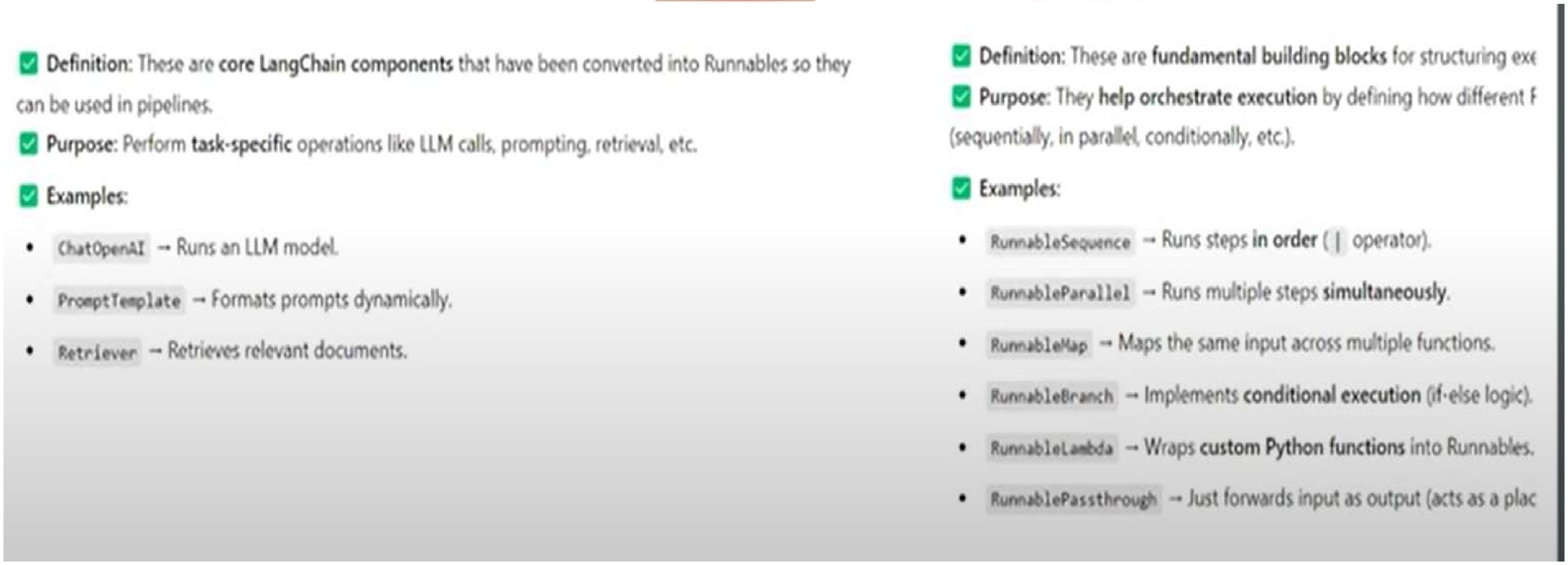


Technique used for this :
Runnable→ (abstract class)
All the classes inherit from this abstract class
all the classes implement the invoke function of abstract class,
Gain Standardization

Types of Runnables



For complex workflow



✅ **Definition:** These are **core LangChain components** that have been converted into Runnables so they can be used in pipelines.

✅ **Purpose:** Perform **task-specific** operations like LLM calls, prompting, retrieval, etc.

✅ **Examples:**

- `ChatOpenAI` → Runs an LLM model.
- `PromptTemplate` → Formats prompts dynamically.
- `Retriever` → Retrieves relevant documents.

✅ **Definition:** These are **fundamental building blocks** for structuring exe

✅ **Purpose:** They **help orchestrate execution** by defining how different R (sequentially, in parallel, conditionally, etc.).

✅ **Examples:**

- `RunnableSequence` → Runs steps **in order** (`|` operator).
- `RunnableParallel` → Runs multiple steps **simultaneously**.
- `RunnableMap` → Maps the same input across multiple functions.
- `RunnableBranch` → Implements **conditional execution** (if-else logic).
- `RunnableLambda` → Wraps **custom Python functions** into Runnables.
- `RunnablePassthrough` → Just forwards input as output (acts as a plac

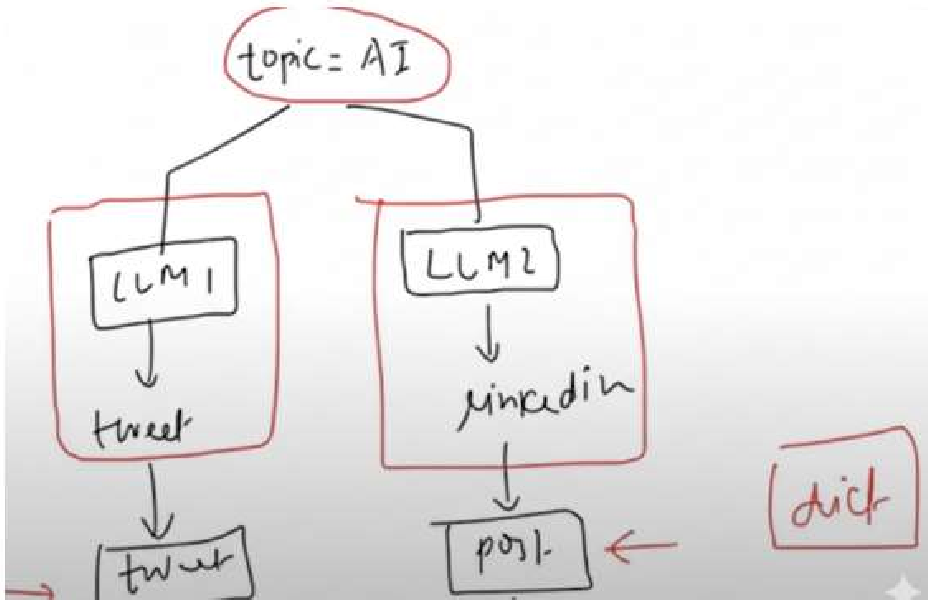
RunnableSequence

- RunnableSequence is a sequential chain of runnables in LangChain that executes each step one after another, passing the output of one step as the input to the next.

- It is useful when you need to compose multiple runnables together in a structured workflow.

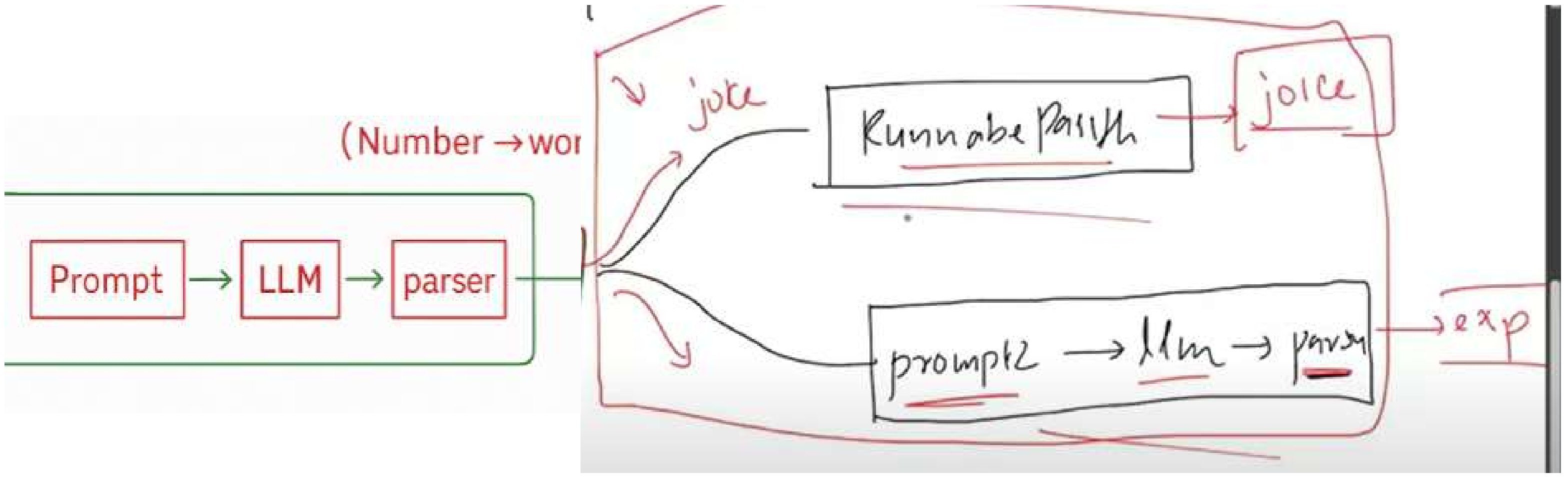
RunnableParallel

- RunnableParallel is a runnable primitive that allows multiple runnables to execute in parallel.
- Each runnable receives the same input and processes independently, producing a dictionary of outputs.



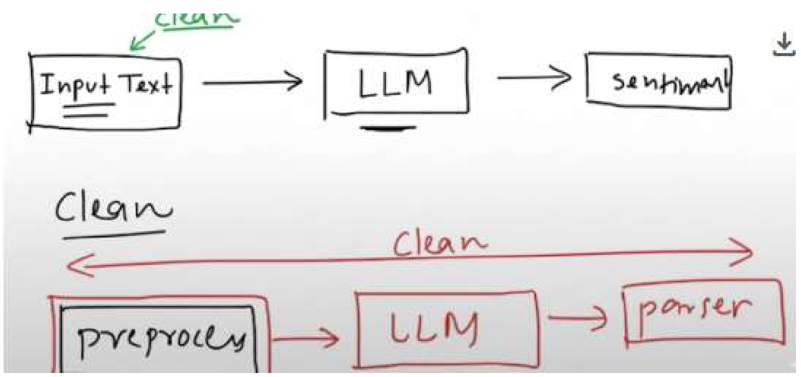
RunnablePassthrough

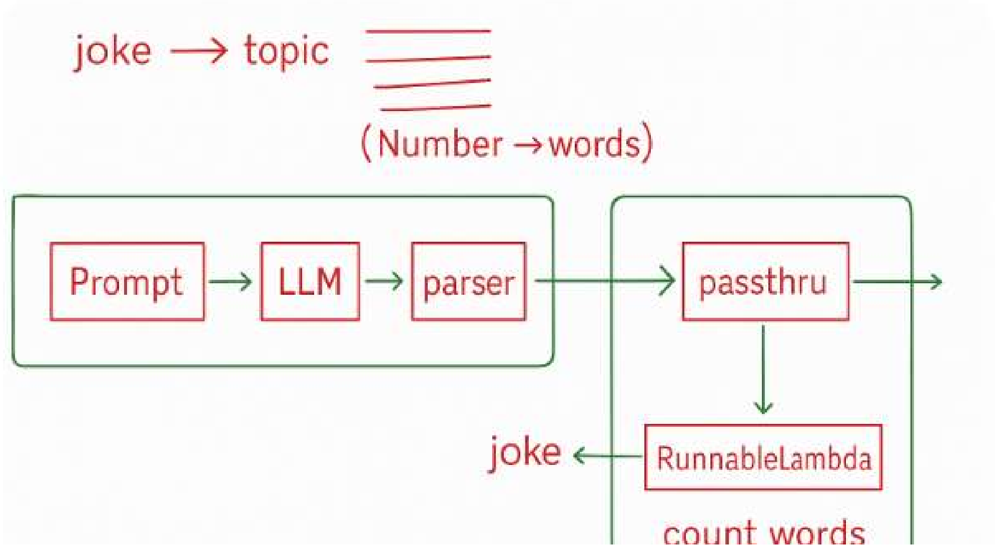
- RunnablePassthrough is a special Runnable primitive that simply returns the input as output without modifying it.



RunnableLambda

- RunnableLambda is a runnable primitive that allows you to apply custom Python functions within an AI pipeline.
- It acts as a middleware between different AI components, enabling preprocessing, transformation, API calls, filtering, and post-processing in a LangChain workflow.

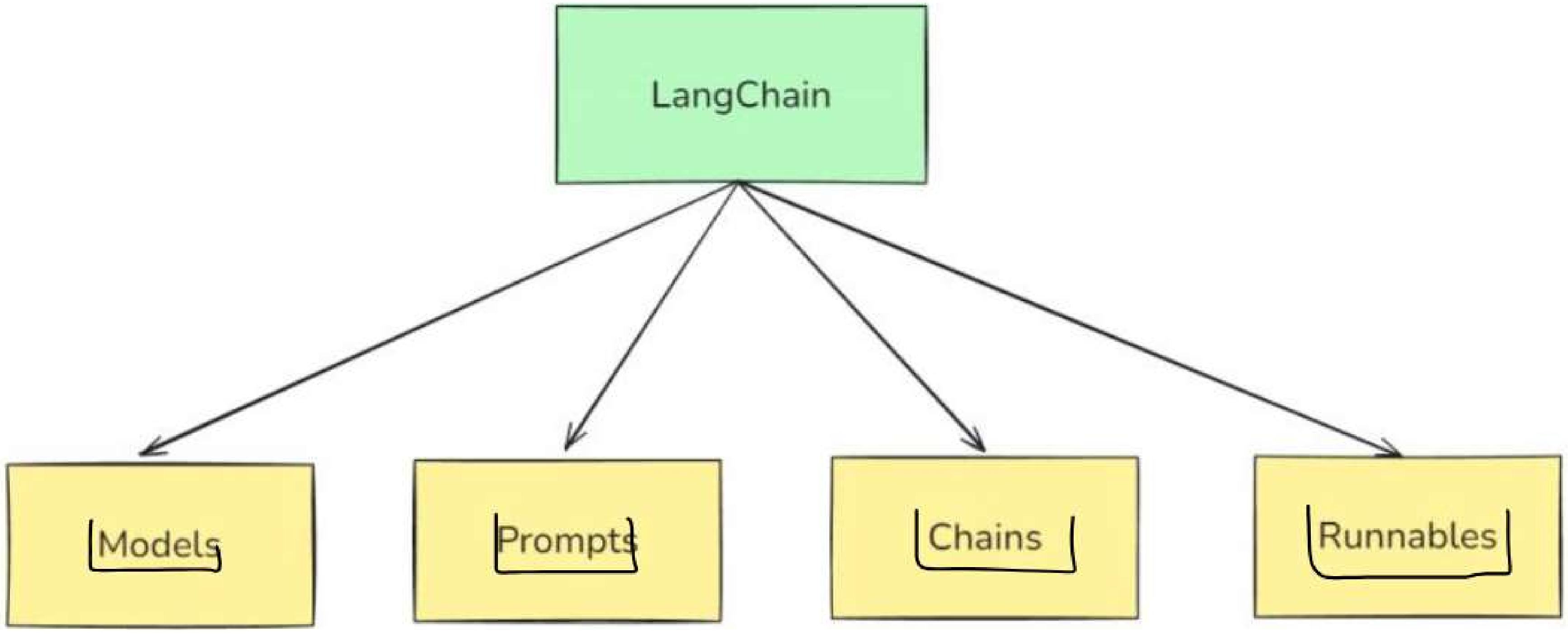




RunnableBranch

- RunnableBranch is a control flow component in LangChain that allows you to conditionally route input data to different chains or runnables based on custom logic.
- It functions like an if/elif/else block for chains — where you define a set of condition functions, each associated with a runnable (e.g., LLM call, prompt chain, or tool). The first matching condition is executed. If no condition matches, a default runnable is used (if provided)

LCEL


LangChain
Models
Prompts
Chains
Runnables