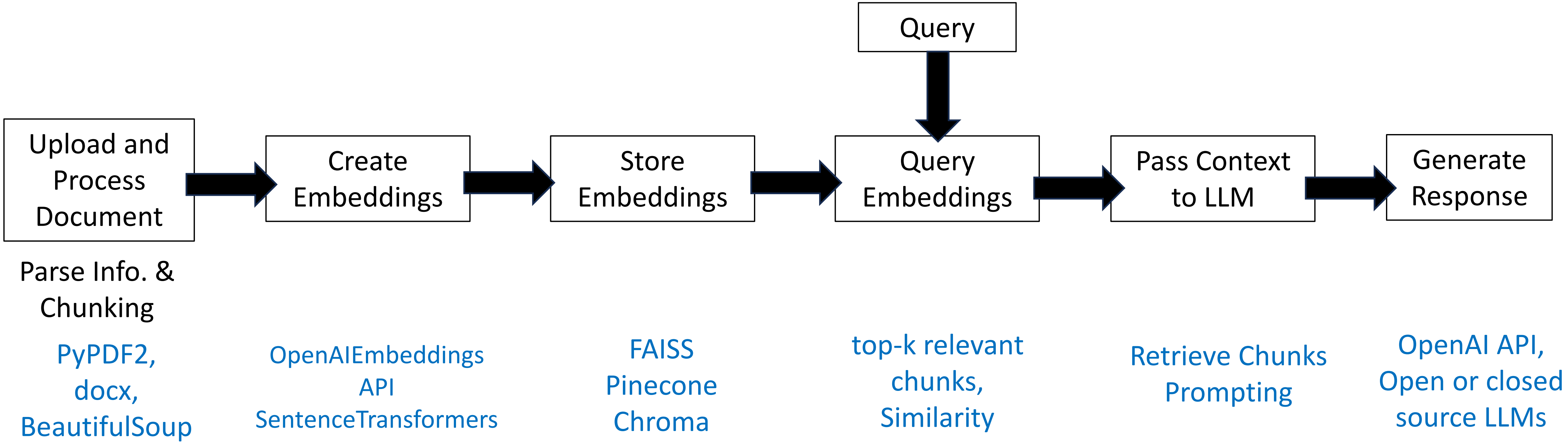
GenAI Frameworks – LangChain and LangGraph

Bhanu Chander V

26th Nov 2024, 29th Nov 2024

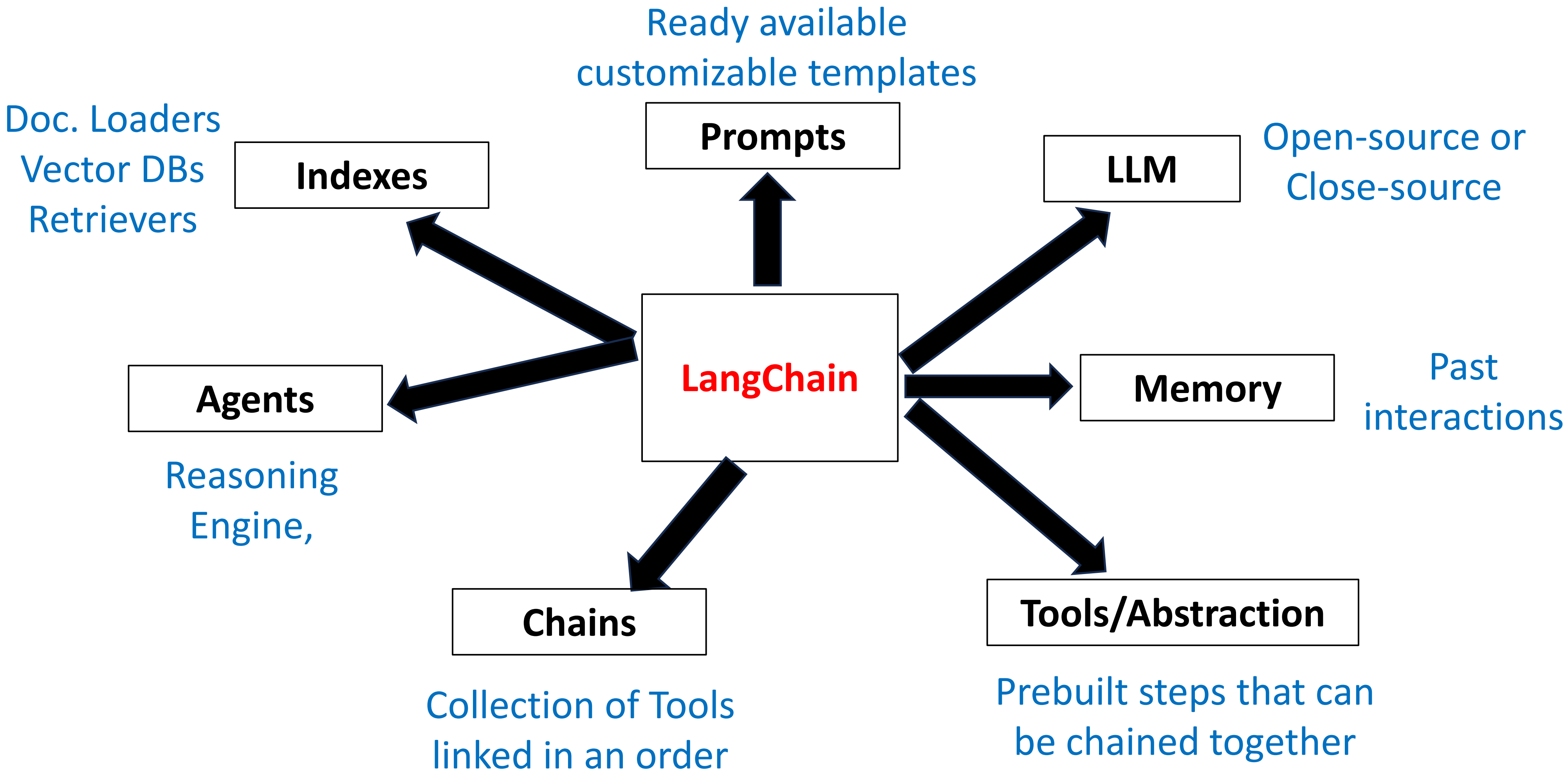
Basic RAG Approach



Langchain can automate the above manual tasks for you

Majorly it can: Interact with External Data, make API Calls, use Memory etc.

Langchain Components



LangChain Components

1. Tools:

1. **Independent components** that can be chained
   - Embeddings, vector stores, Loaders etc.
2. Interfaces that allow language models to **interact with external systems**, such as:
   - APIs
   - Databases
   - Functions etc.
3. Each Tool has:
   - Name
   - Description
   - Inputs
   - Function
4. **Examples:**
   1. A Function that queries a DB
   2. Call an external API
   3. Initiate a document loader or embeddings

2. Chains:

1. Sequences of Tool calls (or actions): Allows you to **combine multiple tools into a workflow.**
2. Actions in chains are **predefined** and **specified in a specific order**
3. **Linear and Static.** They don't change dynamically based on Input**.**
4. **Examples:**
   1. **Chain1**: A chain for Data/Doc Processing Pipeline: Loading, Embedding, & Storing
   2. **Chain2**: A chain to Query Handling: combine query embeddings + generate response int a chain

3. Agents:

1. Manage workflows – dynamically decide **which Tool/Chain to use** and i**n which order**
2. Agents are responsible for:
   1. Overall logic
   2. Can dynamically adjust the workflow based on input & context
3. **Examples:**
   1. An agent that uses a doc loader, embedding model, vectorstore, & lang model to handle user queries
   2. An agent that process Chain2 after Chain1

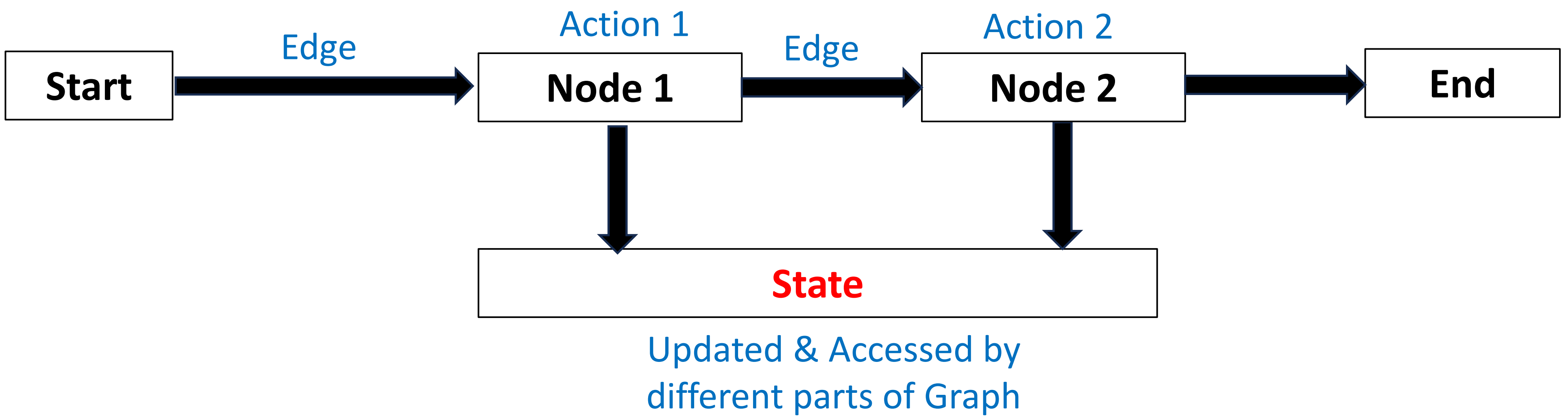
LangChain Demo

Pros & Cons (Langchain)

Aspect	Without LangChain	With LangChain
Ease of Use	Manual setup of components; more complex	Simplified; abstracts many repetitive tasks
Flexibility	High, but requires more coding effort	Moderate; uses pre-built modules
Customizability	Full control over every step	Limited to LangChain's modularity
Speed	Potentially slower due to manual optimization	Faster to prototype and scale
Tooling/Agents	Requires custom im[illegible]entations	Built-in support for tools and agents

LangGraph

1. Built **on top of LangChain** to manage Agents and their workflows**.**
2. LangGraph can create Agents & Multi-agents workflows
3. Consider when app **needs multi-agents interaction** to solve complex problems



Edges: Connect Nodes & define direction of data flow, flow of execution

Nodes: Specific tasks, Individual Actions, like executing LLM, running functions etc.

LangGraph Demo

Pros of LangGraph

Aspect	Without LangGraph	With LangGraph
Code Complexity	Procedural code with manual orchestration	Declarative graph structure simplifies orchestration
Visualization	No visual representation	Generates a clear graph of the workflow
Debugging	Harder to pinpoint issues	Node-level error tracking and dependency isolation
Reusability	Components require manual reuse	Nodes can be reused across workflows
Workflow Flexibility	Static and linear	Dynamic with dependency-based execution

LangSmith

1. Can be used with any framework**.**
2. Monitoring & evaluation tool
3. **Basically LLMOps** - to monitor tokens used, cost, I/p, O/p, execution time etc.
4. If App is simple, LS can be overkill. Mostly use for Deploying & Testing

LangFlow

1. Drag & Drop, without code,
2. For LLM Prototyping - Not for production

Other Tools to explore:

1. **RelevanceAI**
2. **Dify**

https://python.langchain.com/docs/integrations/tools/

Tools Examples:

Function	Description	Tool
Retrieval	Retrieves documents or chunks based on semantic similarity to the query.	**VectorstoreRetriever** (e.g., FAISS, Pinecone, Chroma, Weaviate)
Embeddings	Tools to generate vector embeddings from text data.	**OpenAIEmbeddings, HuggingFaceEmbeddings, CohereEmbeddings**
Databases	Executes SQL queries to retrieve data from relational databases.	SQLDatabase
Web Search	Fetches real-time data from the web for context.	**BingSearchAPI, GoogleSearchAPI**
File I/O	Reads and writes local or cloud-stored files.	**FileTool**
Translation/Language	Performs translation for multilingual queries and contexts.	**GoogleTranslate, DeepL**
Utilities	Performs auxiliary tasks like calculations or JSON parsing.	**Calculator, JSONExtractor, TextCleaner**

LangChain Components Reference for https://python.langchain.com/docs/integrations/components/
– Models, Retrievers, Tools, Doc Loaders, VectorStores, Embedding models, others

Chains Examples:

Chain	Description
RetrievalQA	Combines a retriever (e.g., vector store) with an LLM to answer questions using retrieved documents.
ConversationalRetrievalQA	Enhances `RetrievalQA` with memory for context-aware, multi-turn conversations.
LLMChain	A simple chain that processes inputs with a prompt template and an LLM.
SequentialChain	Chains together multiple subchains or tools in a linear sequence.
GraphChain	Constructs non-linear workflows using dependencies (useful for decision trees or multi-step reasoning).
SQLDatabaseChain	Integrates an LLM with a SQL database for structured data queries.

Agents Examples:

Agent Type	Description
Zero-shot Agent	Executes tasks without predefined steps, relying entirely on the LLM's reasoning.
Conversational Agent	Manages conversational state while dynamically deciding which tools to invoke.
Tool-Driven Agent	Uses a set of pre-defined tools (e.g., retrievers, APIs) for task execution.
Action Plan Agent	Generates a plan of actions (tool invocations or reasoning steps) before execution.
MRKL Agent	Stands for "Modular Reasoning, Knowledge, and Language"; combines reasoning and tool usage dynamically.
ReAct Agent	Combines reasoning (via LLM) and acting (tool usage) iteratively.

Sample Combinations of Tools, Chains & Agents

Single-turn QA

- **Tools**: VectorstoreRetriever
- **Chains**: RetrievalQA
- **Agents**: Zero-shot Agent

Multi-turn Conversational QA

- **Tools**: VectorstoreRetriever, ConversationalMemory
- **Chains**: ConversationalRetrievalQA
- **Agents**: Conversational Agent

Example usecase: **RAG for Tables Info. Extraction**

(Relevant Use cases: Transformers, Technical Bids etc.)

Basic Workflow:

- Process an uploaded document
- Retrieves a document (PDF or HTML) containing tables from a storage.
- Dynamically selects a tool for table extraction (using *Camelot/Tabula-py* for grid-based tables, *pdfplumber* for irregular tables).
- Extracts the relevant table.
- Identifies the column with quantities (e.g., “Quantity”, “Price”).
- Extracts numbers using a *regex* or NLP tool like *spaCy* to recognize quantities with units.
- Sends this data to a database or generates a summary report.

LangGraph could be a good fit in the following scenarios:

- **Complex workflows**: If the task involves multiple steps or a chain of operations,

  e.g., you can combine table extraction, quantity recognition, and data manipulation in a single workflow.

- **Multiple data sources:** need to create a unified workflow for extracting and processing the data from different formats

  e.g., PDFs, word, HTML tables, databases)

- Your workflow **involves decision-making** based on the type of table or content

  e.g., a node that checks the format of a document and then routes the flow to the appropriate extraction tool

- **Combining different tools**: LangGraph allows you to seamlessly integrate multiple tools

  e.g., pdfplumber, Camelot, BeautifulSoup, regex, spaCy) into a cohesive pipeline.

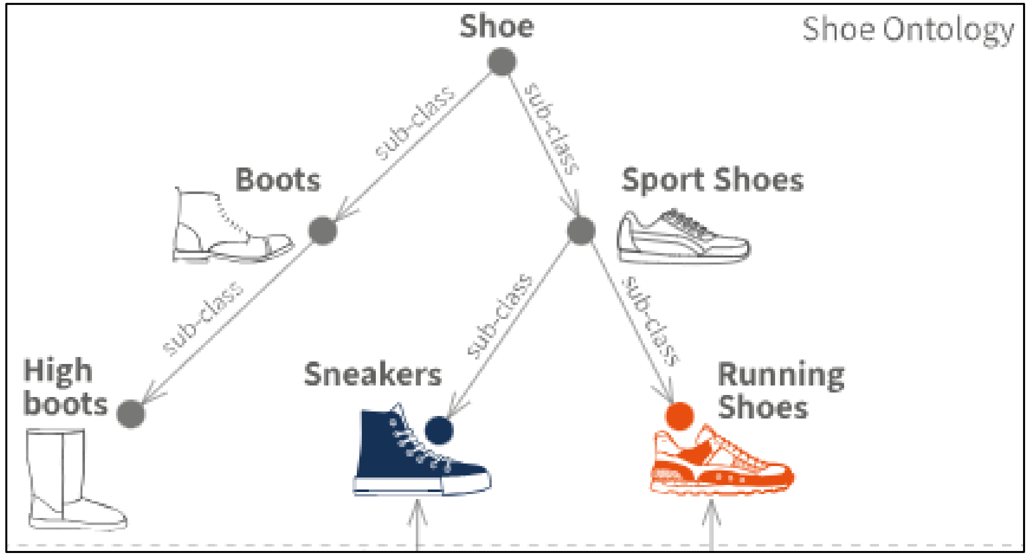
- If you are building a more extensible system that may **need to scale in the future**

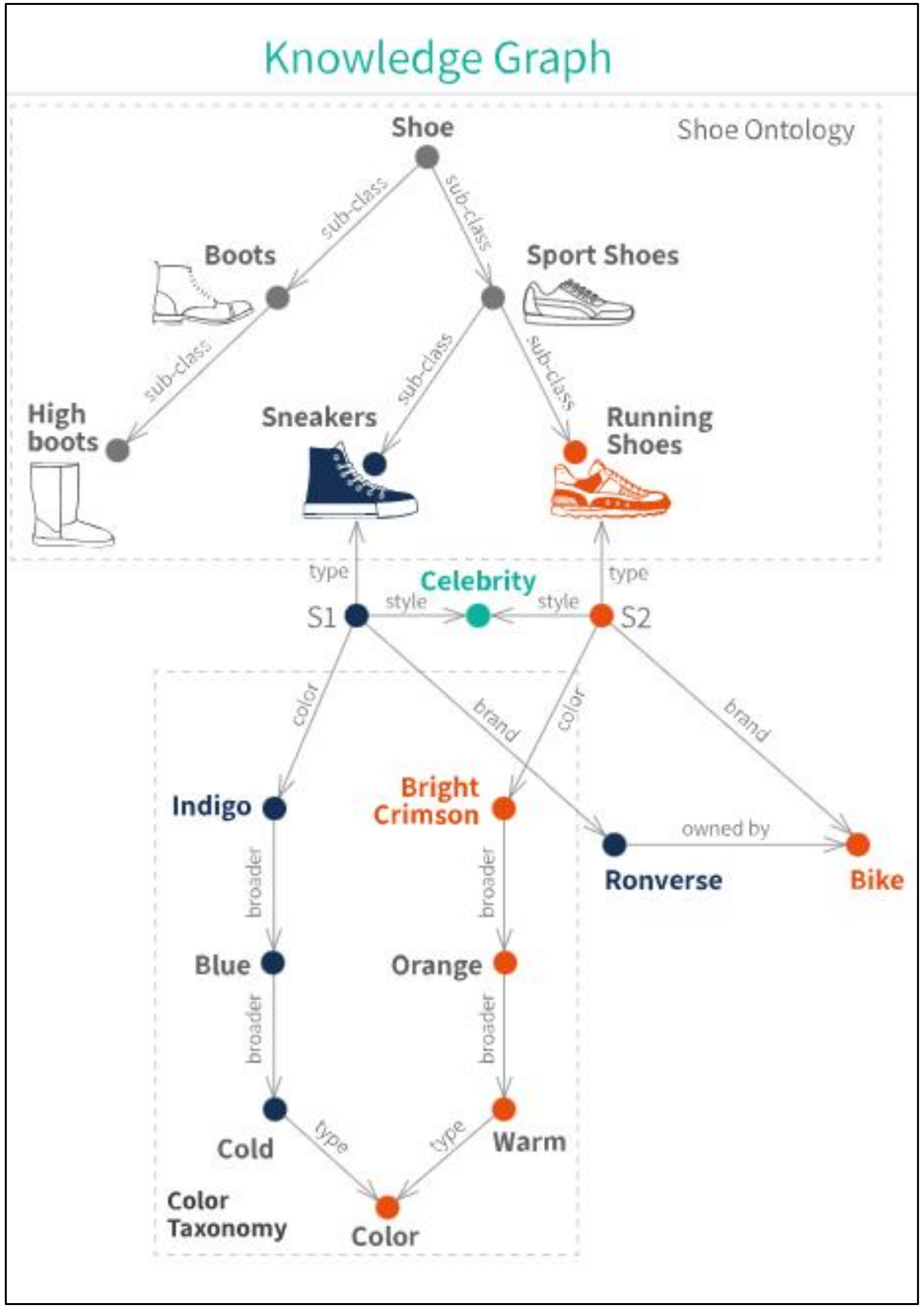
  e.g., adding more tools, changing the extraction logic, or implementing multiple steps

More?

Knowledge Graph

- Structured representation of information, where data is organized in the form of a graph
- Consisting of
  - **Nodes** (entities or concepts) and
  - **Edges** (relationships or associations between nodes)
- Designed to model, store, and organize complex information in a way that makes it easy for both humans and machines to understand, navigate, and use the knowledge it contains.

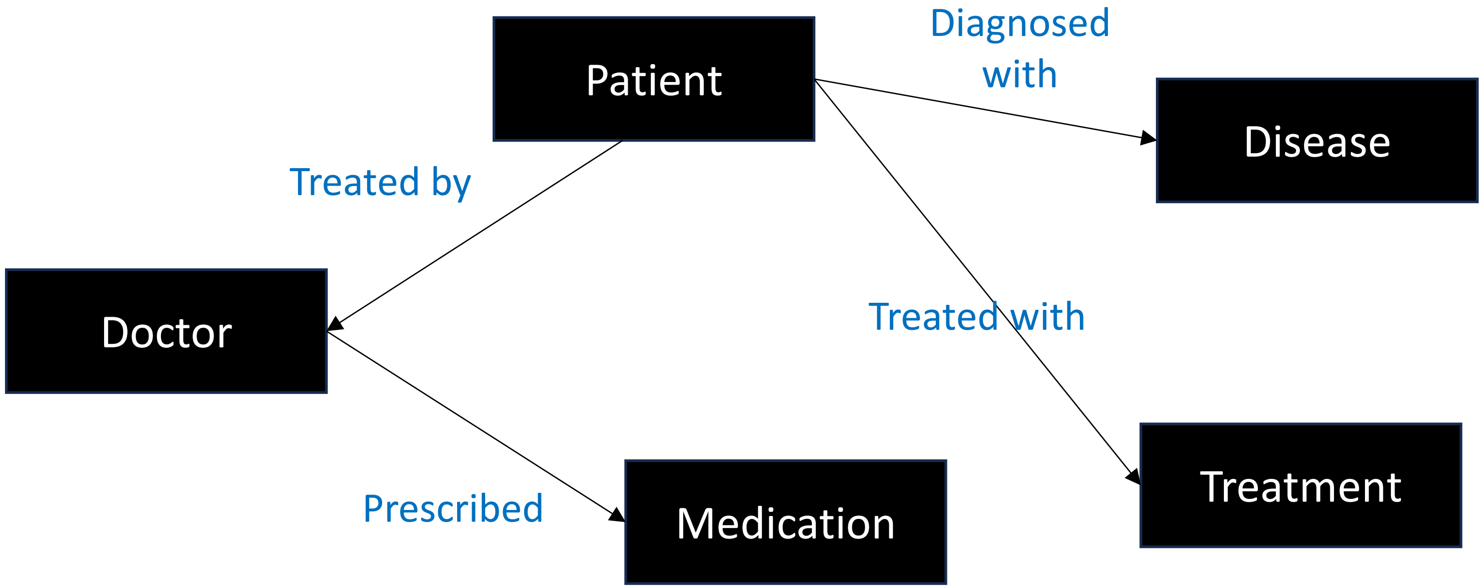




Example of Knowledge Graph for Hospital

- **Nodes** Patient, Doctor, Medication, Treatment, Diagnosis
- **Edges**
  "Diagnosed with" (connecting Patient and Diagnosis),
  "Prescribed" (connecting Doctor and Medication),
  "Treated with" (connecting Patient and Treatment)

Entity Example: people, locations, organizations etc. (basically Nodes)



https://microsoft.github.io/graphrag/

GraphRAG

- **Graph-based** Retrieval-Augmented Generation
- **Leverages knowledge graphs** to enhance the capabilities of LLMs
- Extracts **structured data from unstructured text** and organizes it into a knowledge graph
  - More Accurate and Contextually relevant answers – by connecting different pieces of Info.
- *Imagine an LLM looking at a graphical representation of a data – it will have a clear understanding of connectivity between information and therefore can answer more logically*
- Hence GraphRAG helps an LLM connect the dots

Step-by-Step Example of GraphRAG
- Data Collection
- Entity Extraction
- Knowledge Graph Creation
- Query Processing
- Answer Generation

Aspect	Basic RAG	GraphRAG
Pros	- Enhanced accuracy	- Better contextualization
	- Real-time updates	- Handling complex queries
	- Simplicity	- Reduced hallucinations
		- Explainability
Cons	- Handling complex queries	- Complexity
	- Unstructured context	- Resource intensive
	- Potential for hallucinations	- Scalability issues

GraphRAG Examples

- **Contracts**: Using GraphRAG, you can create a knowledge graph that maps entities like companies, individuals, dates, and legal terms from contract documents. The graph can help in quickly querying specific terms, identifying patterns, and suggesting amendments.
- **Procurement**: Build a knowledge graph connecting suppliers, products, prices, and contract terms. GraphRAG can help identify the most reliable and cost-effective suppliers based on historical data and performance metrics.
- **Supply Chain Management**: Create a knowledge graph mapping out suppliers, logistics, inventory levels, and sales data. Use GraphRAG to predict potential supply chain disruptions and suggest alternative routes or suppliers.
- **Engineering Document Comparison:** Build a knowledge graph that captures the relationships between different components, materials, and specifications. Use GraphRAG to highlight changes and suggest optimizations based on historical data and best practices.
- **Table Information Extraction from Brochures:** Extract tables from brochures, create a knowledge graph with the extracted data, and use GraphRAG to query specific information, like product specifications or pricing.