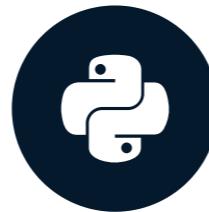


From vectors to graphs

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN



Meri Nova
Machine Learning Engineer

Vector RAG limitations

"How can I install Python locally?"

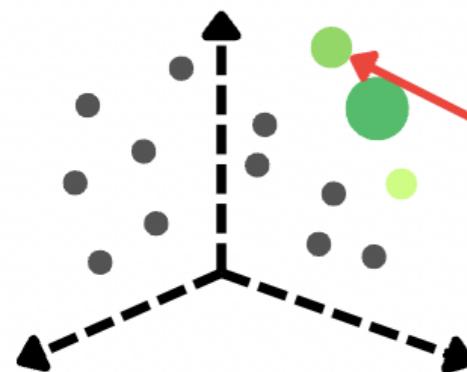
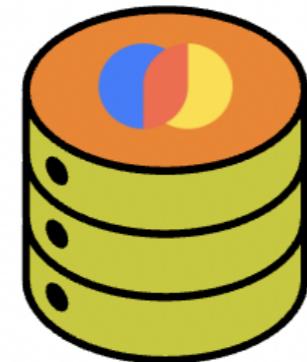


Embedding Model

[-0.829647...]



Vector Database



"Install Python in 3 easy steps?"



Vector RAG limitations

"How can I install Python locally?"

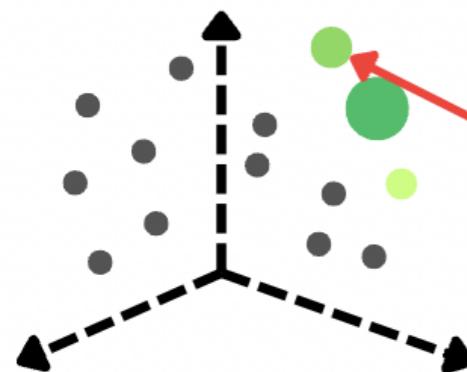
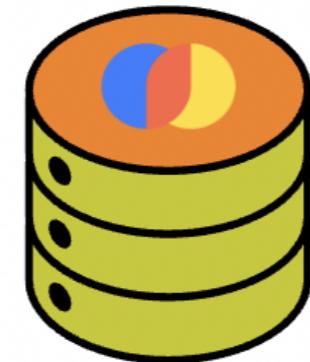


Embedding Model

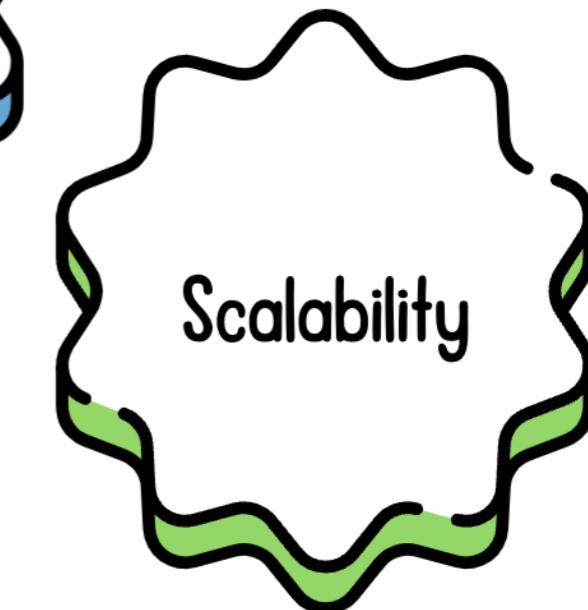
[-0.829647...]



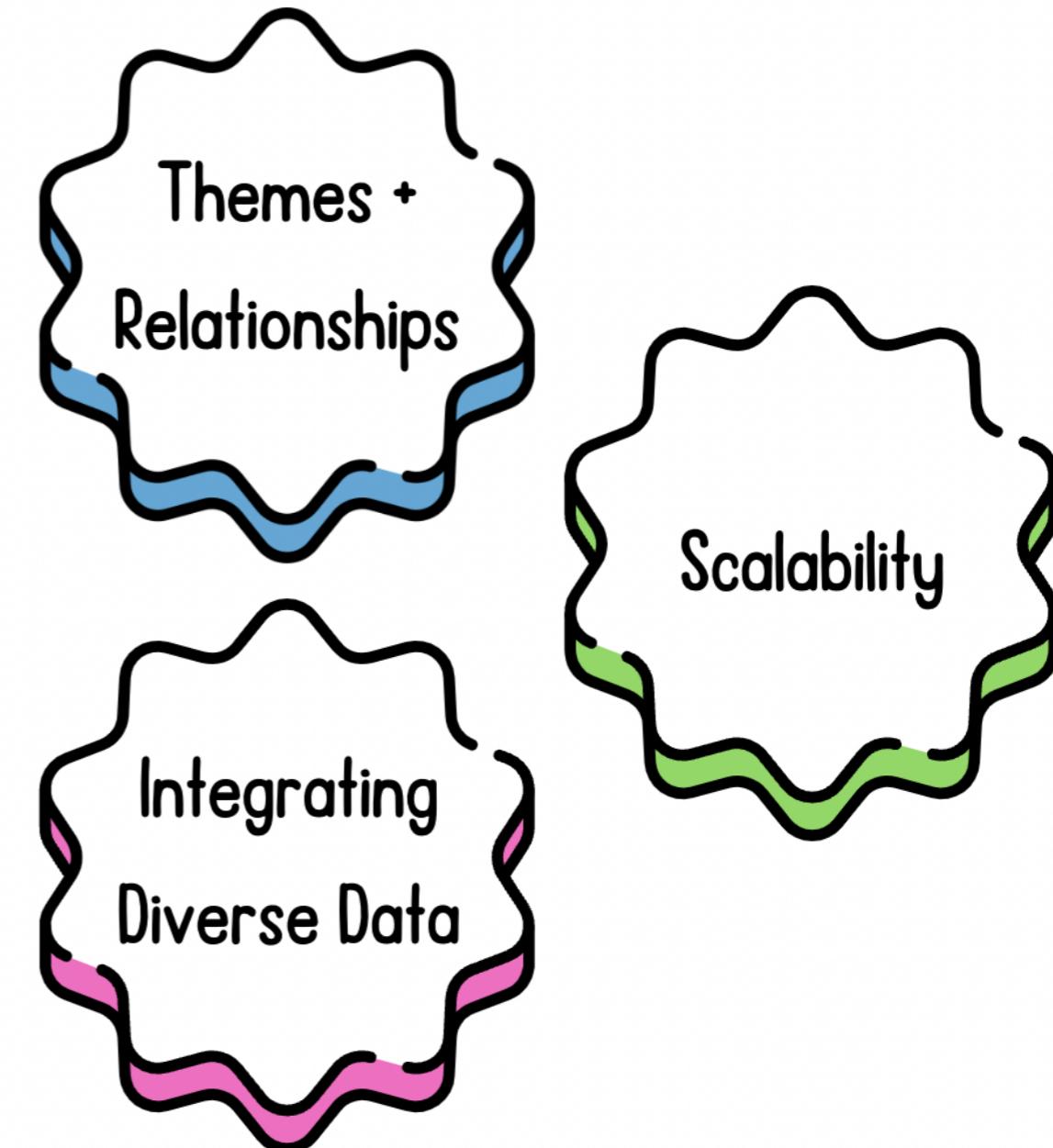
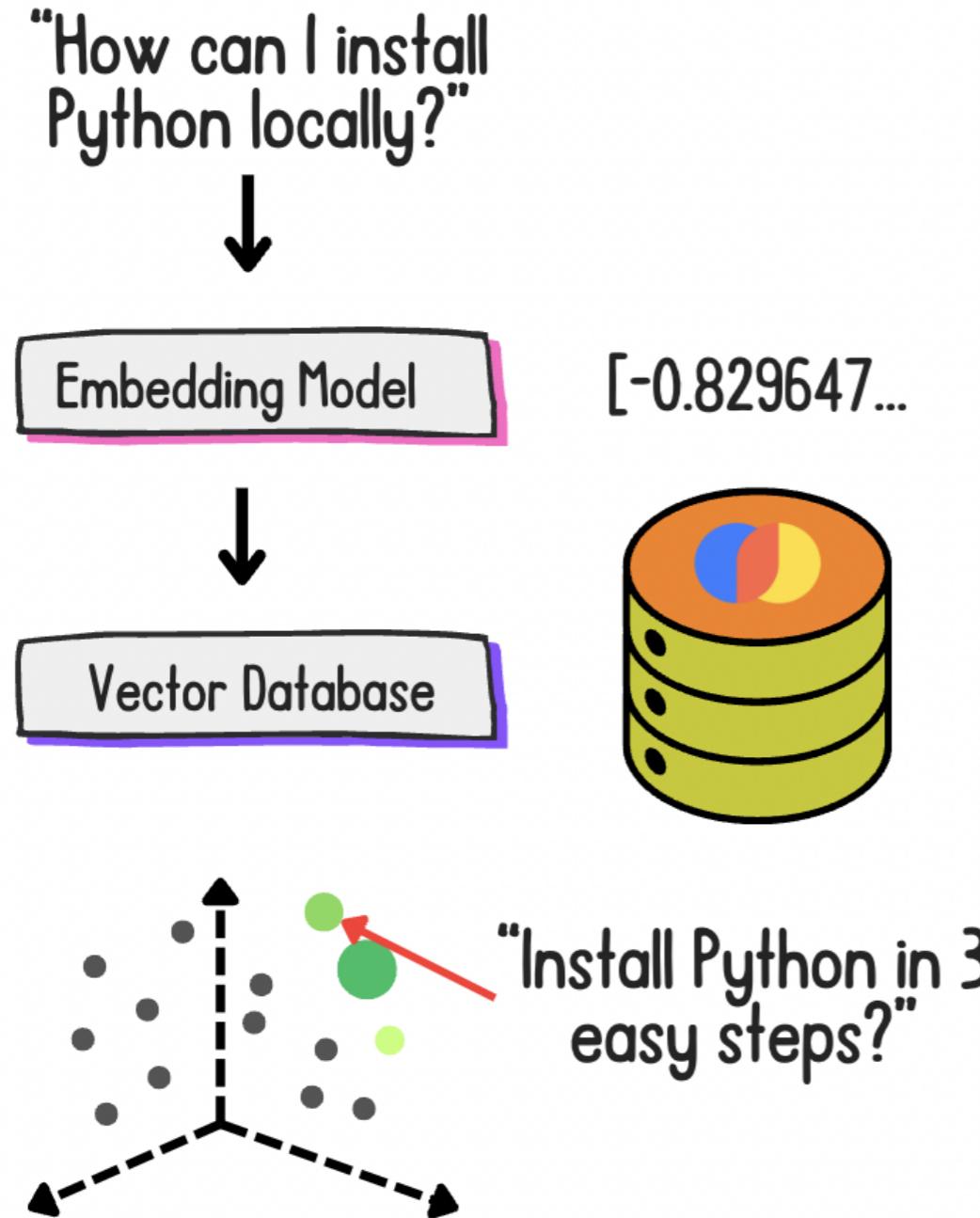
Vector Database



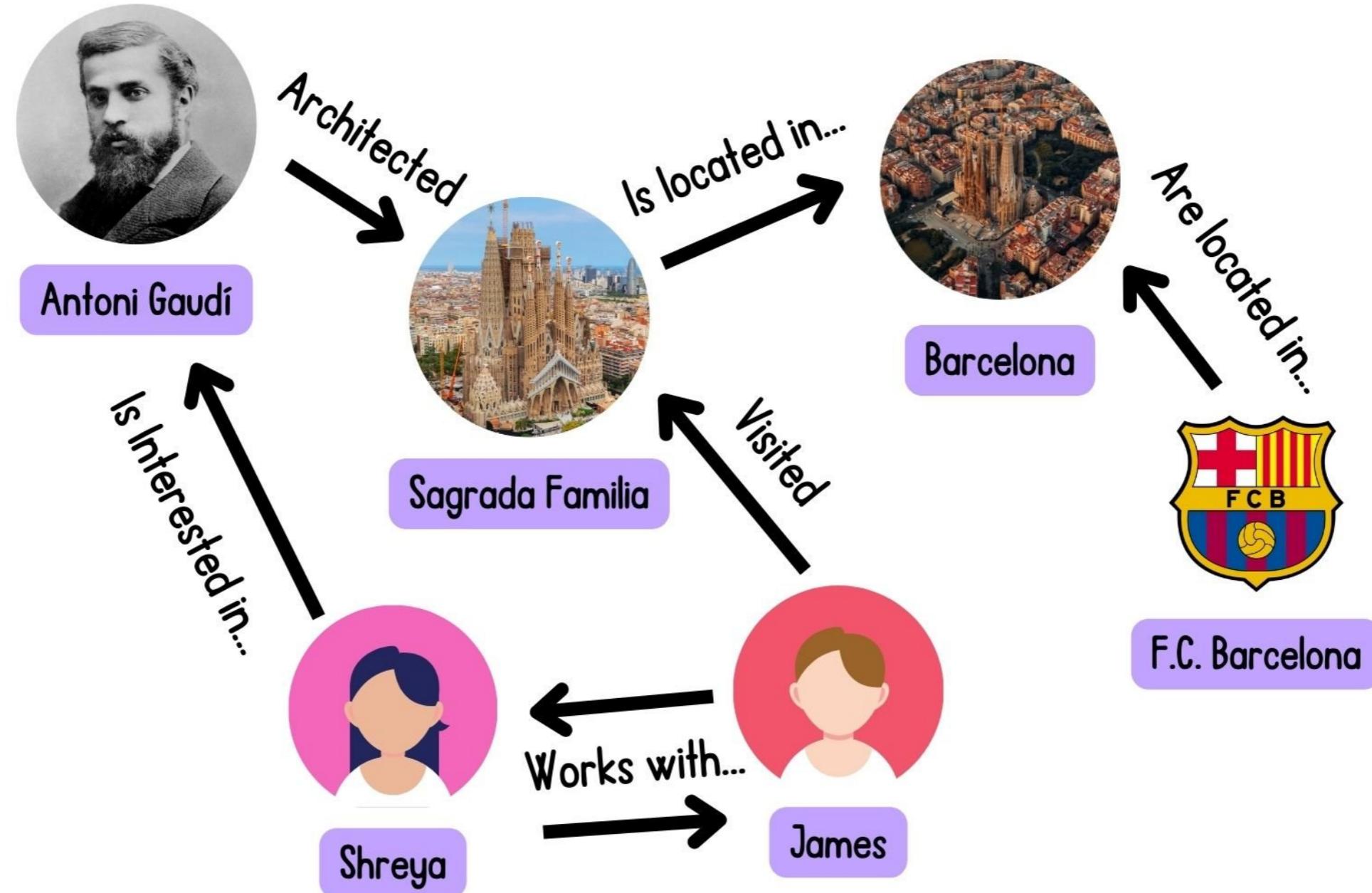
"Install Python in 3
easy steps?"



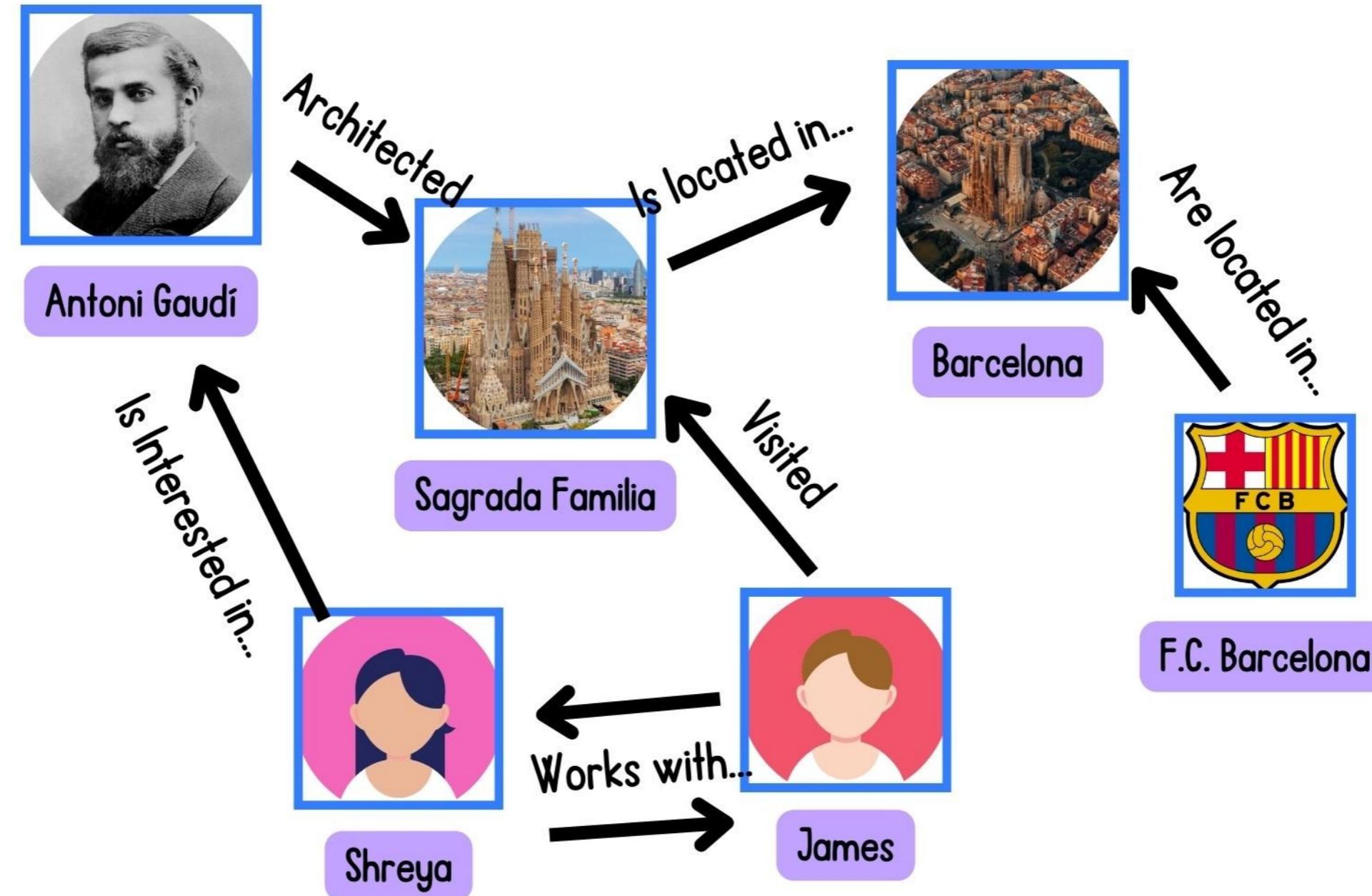
Vector RAG limitations



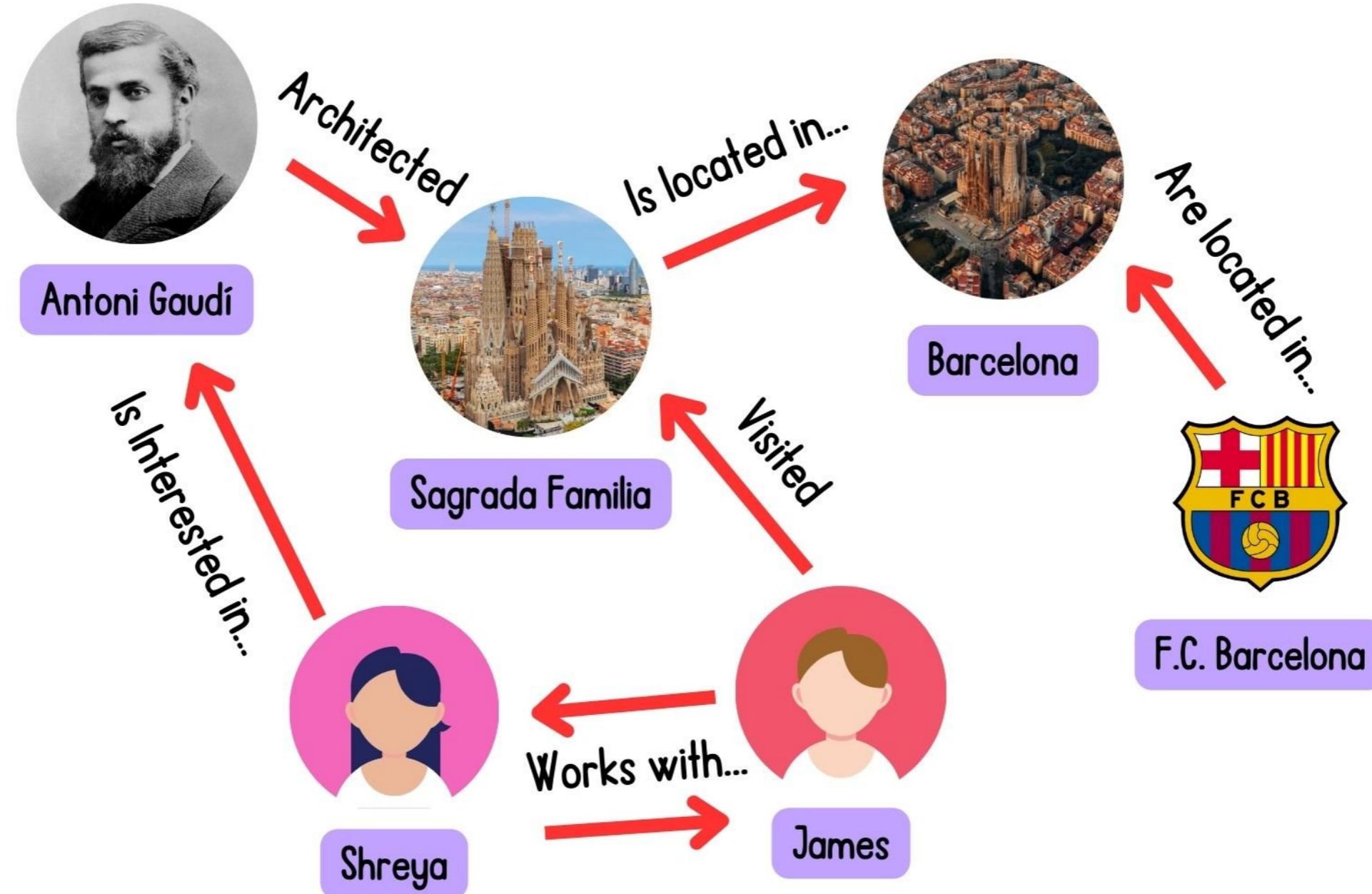
Graph databases



Graph databases - nodes



Graph databases - edges

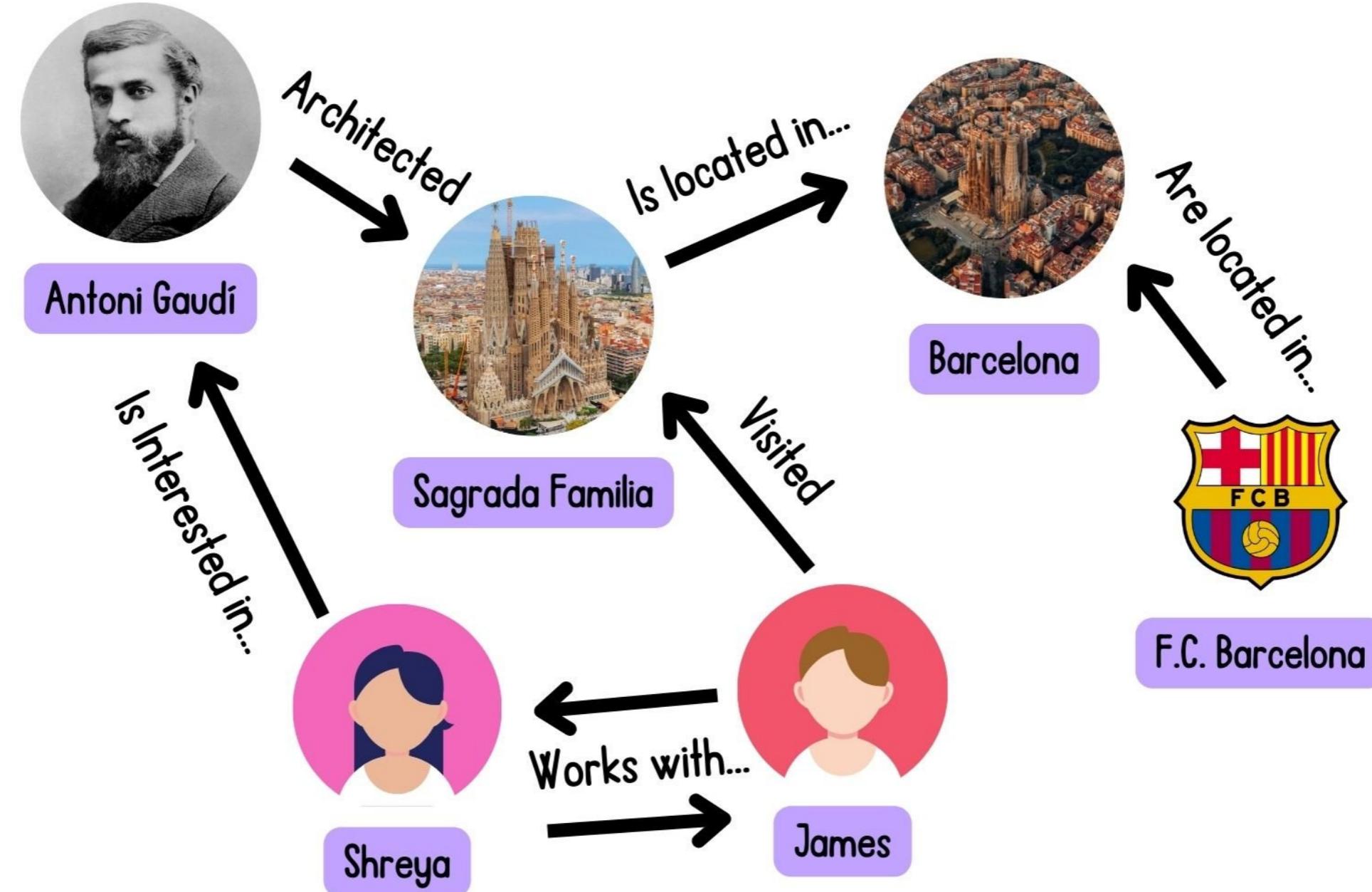


Neo4j graph databases

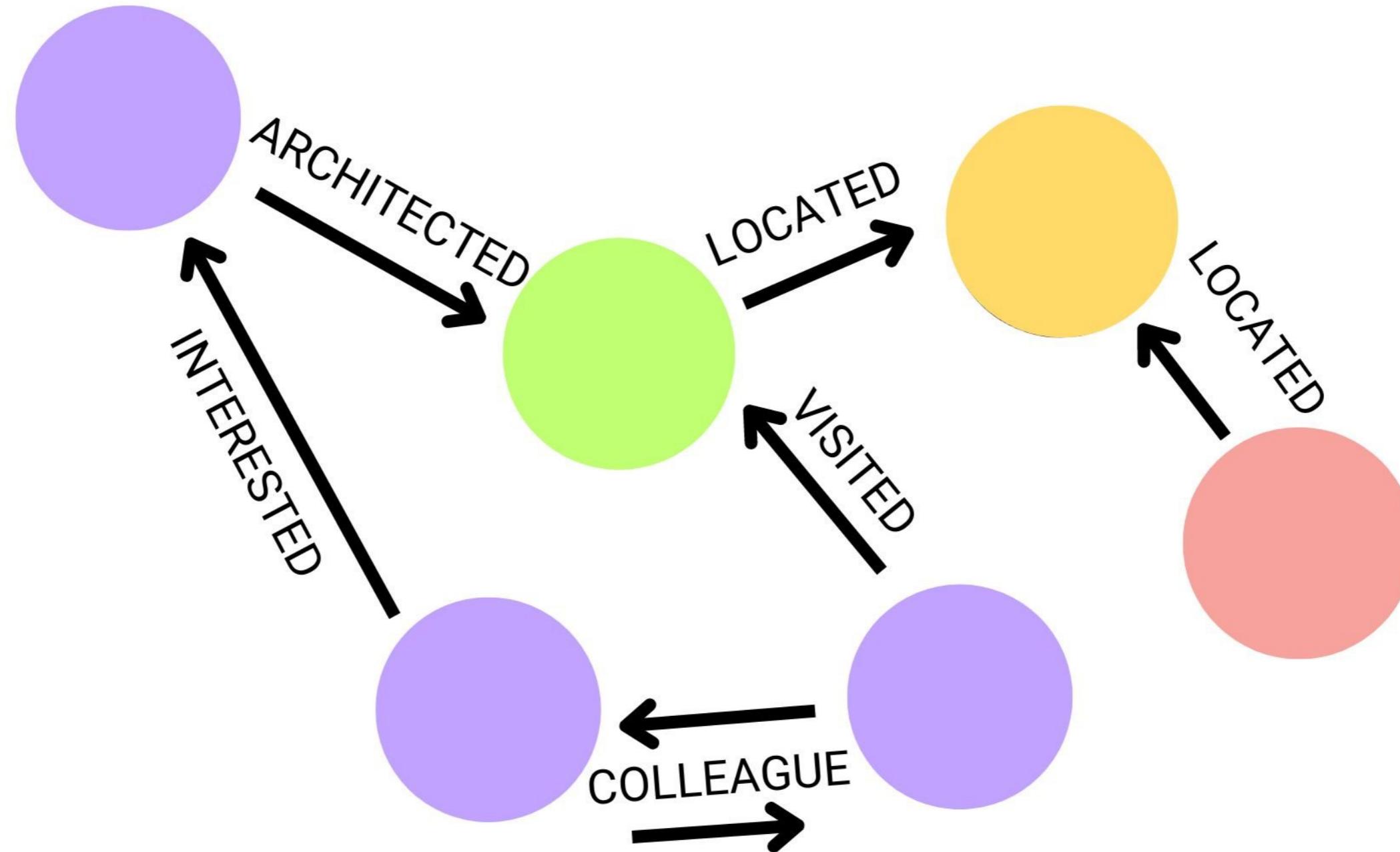


LangChain

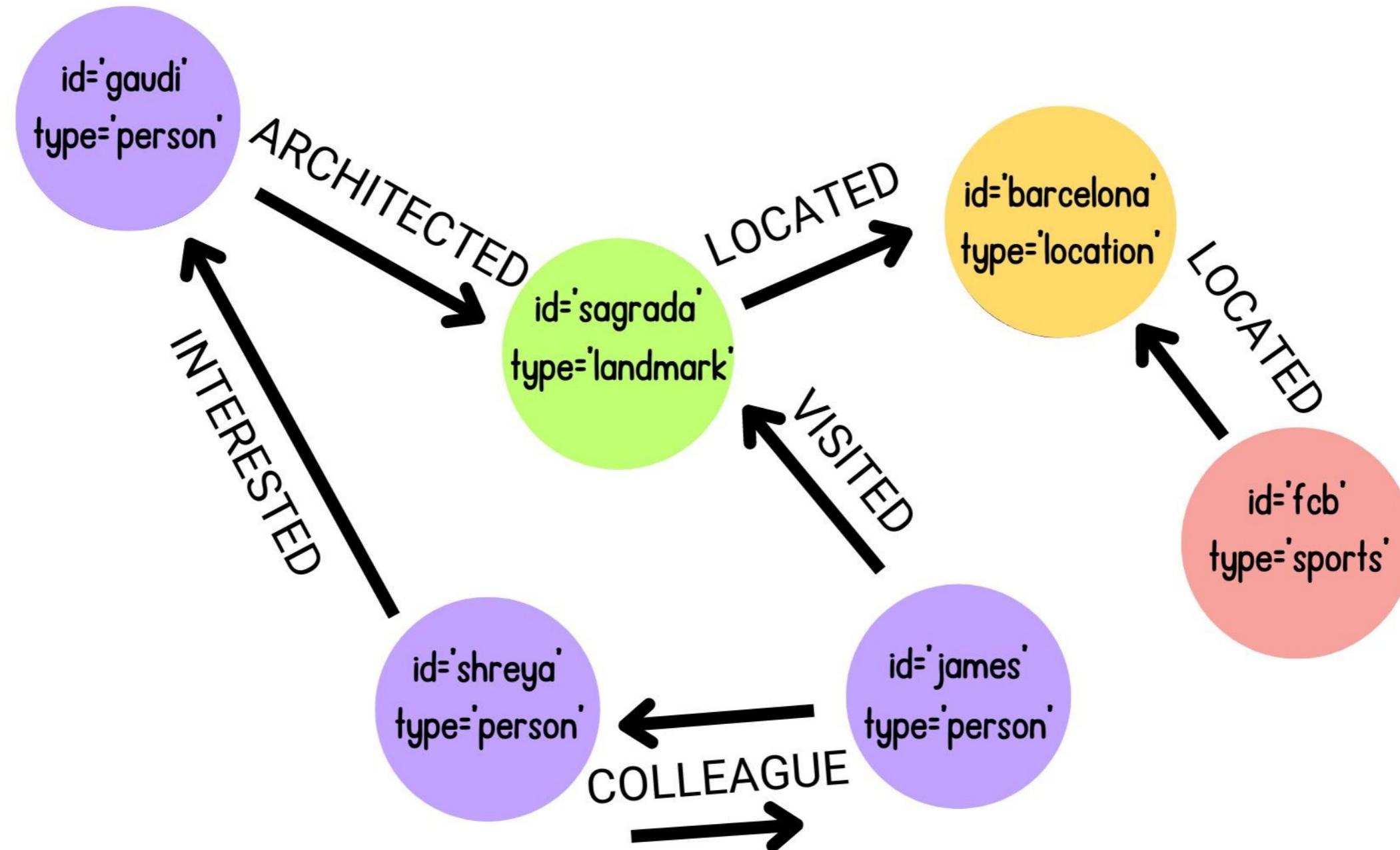
From graphics to graphs...



From graphics to graphs...



From graphics to graphs...



Loading and chunking Wikipedia pages

```
from langchain_community.document_loaders import WikipediaLoader
from langchain_text_splitters import TokenTextSplitter

raw_documents = WikipediaLoader(query="large language model").load()
text_splitter = TokenTextSplitter(chunk_size=100, chunk_overlap=20)
documents = text_splitter.split_documents(raw_documents[:3])

print(documents[0])
```

```
page_content='A large language model (LLM) is a computational model capable of...'
metadata={'title': 'Large language model',
          'summary': "A large language model (LLM) is...",
          'source': 'https://en.wikipedia.org/wiki/Large_language_model'}
```

From text to graphs!

```
from langchain_openai import ChatOpenAI
from langchain_experimental.graph_transformers import LLMGraphTransformer

llm = ChatOpenAI(api_key="...", temperature=0, model_name="gpt-4o-mini")
llm_transformer = LLMGraphTransformer(llm=llm)

graph_documents = llm_transformer.convert_to_graph_documents(documents)
print(graph_documents)
```

From text to graphs!

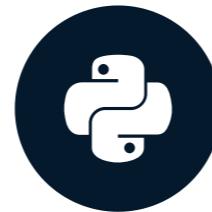
```
[GraphDocument(  
    nodes=[  
        Node(id='Llm', type='Computational model'),  
        Node(id='Language Generation', type='Concept'),  
        Node(id='Natural Language Processing Tasks', type='Concept'),  
        Node(id='Llama Family', type='Computational model'),  
        Node(id='Ibm', type='Organization'),  
        ..., Node(id='Bert', type='Computational model')],  
    relationships=[  
        Relationship(source=Node(id='Llm', type='Computational model'),  
                    target=Node(id='Language Generation', type='Concept'),  
                    type='CAPABLE_OF'),  
        ...])]
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Storing and querying documents

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN



Meri Nova
Machine Learning Engineer

Instantiating the Neo4j database

```
from langchain_community.graphs import Neo4jGraph
```

```
graph = Neo4jGraph(url="bolt://localhost:7687", username="neo4j", password="...")
```

```
import os
```

```
url = os.environ["NEO4J_URI"]
```

```
user = os.environ["NEO4J_USERNAME"]
```

```
password = os.environ["NEO4J_PASSWORD"]
```

```
graph = Neo4jGraph(url=url, username=user, password=password)
```

¹ <https://neo4j.com/download/>

Storing graph documents

```
from langchain_experimental.graph_transformers import LLMGraphTransformer

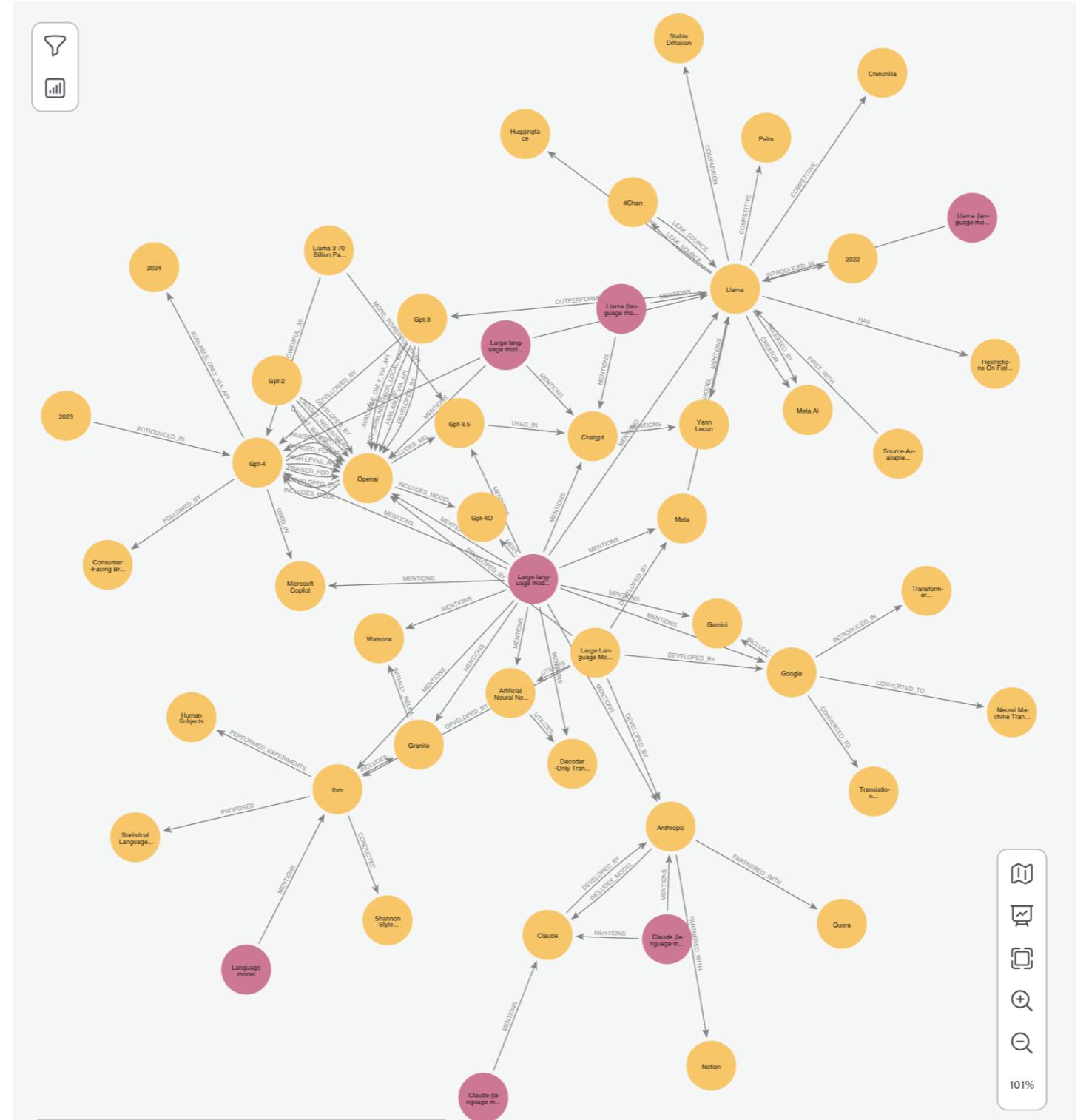
llm = ChatOpenAI(api_key="...", temperature=0, model="gpt-4o-mini")
llm_transformer = LLMGraphTransformer(llm=llm)

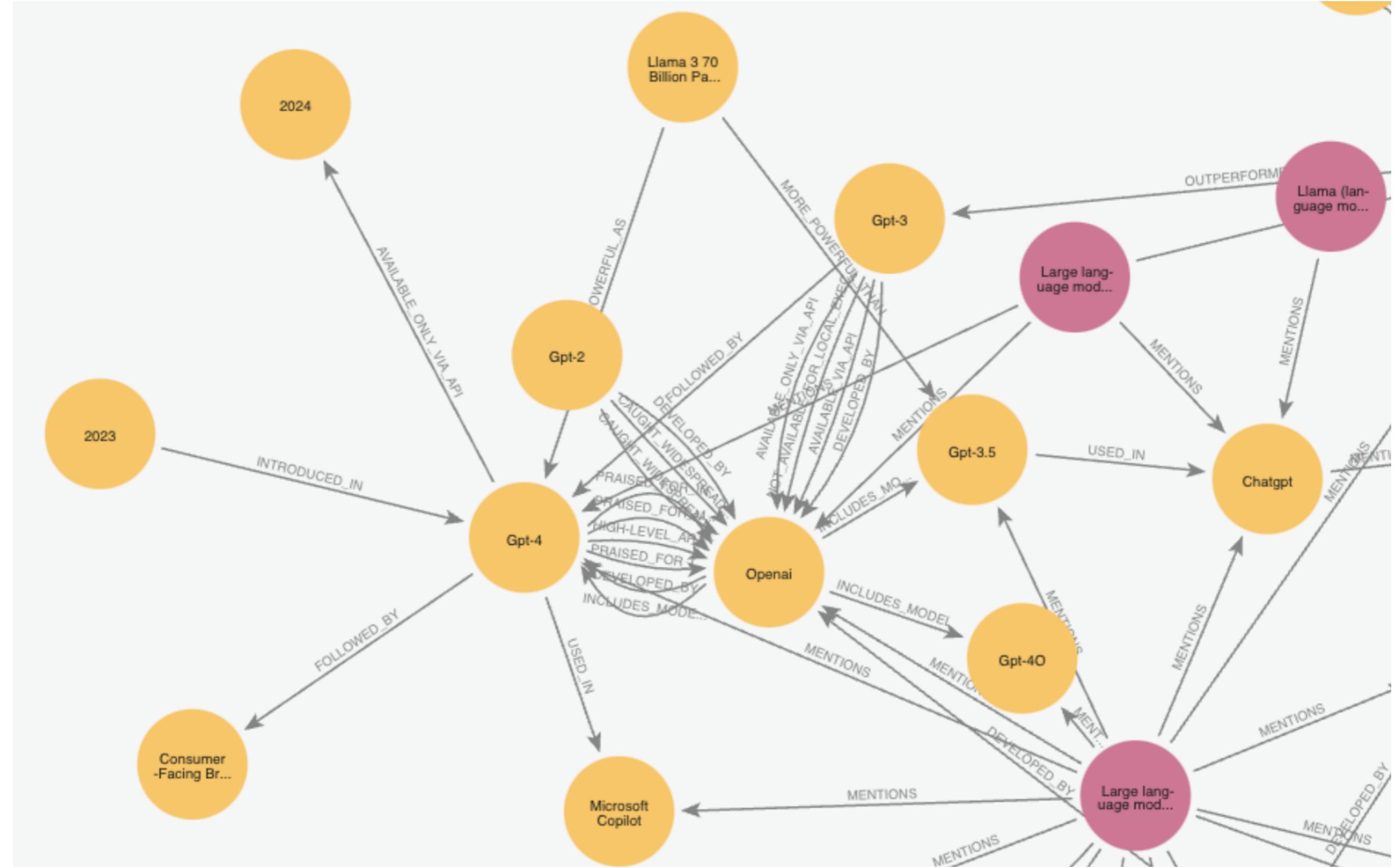
graph_documents = llm_transformer.convert_to_graph_documents(documents)
```

Storing graph documents

```
graph.add_graph_documents(  
    graph_documents,  
    include_source=True,  
    baseEntityLabel=True  
)
```

- `include_source=True` : link nodes to source documents with `MENTIONS` edge
- `baseEntityLabel=True` : add `__Entity__` label to each node





Database schema

```
print(graph.get_schema)
```

Node properties:

Concept {id: STRING}

Architecture {id: STRING}

Organization {id: STRING}

Event {id: STRING}

Paper {id: STRING}

The relationships:

(:Concept)-[:DEVELOPED_BY]->(:Person)

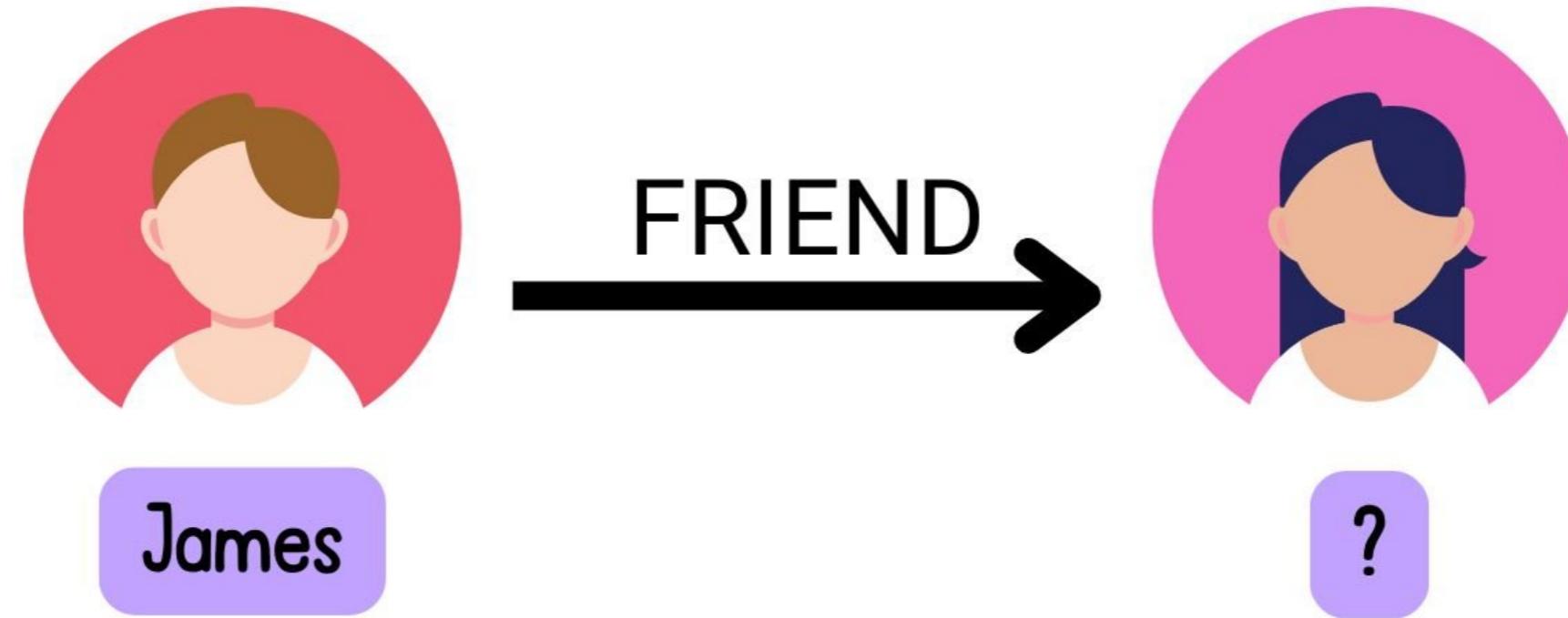
(:Architecture)-[:BASED_ON]->(:Concept)

(:Organization)-[:PROPOSED]->(:Concept)

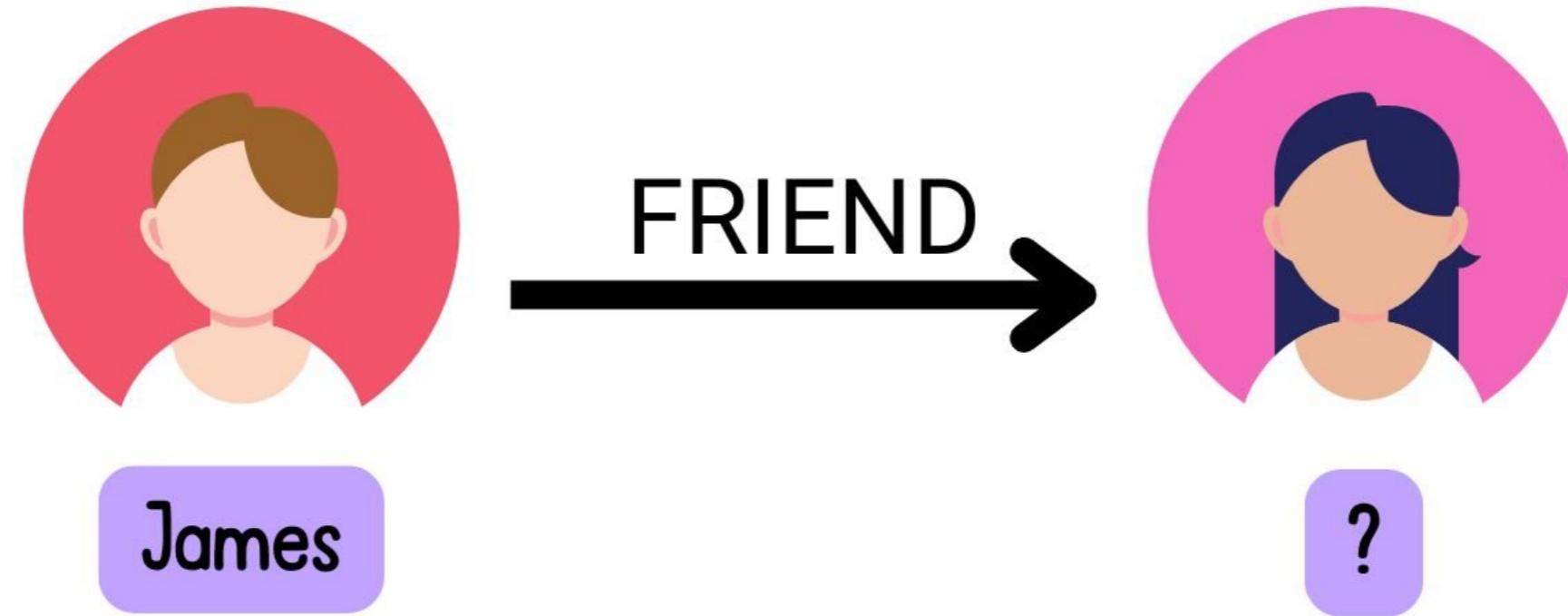
(:Document)-[:MENTIONS]->(:Event)

(:Paper)-[:BASED_ON]->(:Concept)

Querying Neo4j - Cypher Query Language



Querying Neo4j - Cypher Query Language



```
MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend
```

Querying Neo4j - Cypher Query Language



James

?

Node

Relationship

Node

```
MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend
```

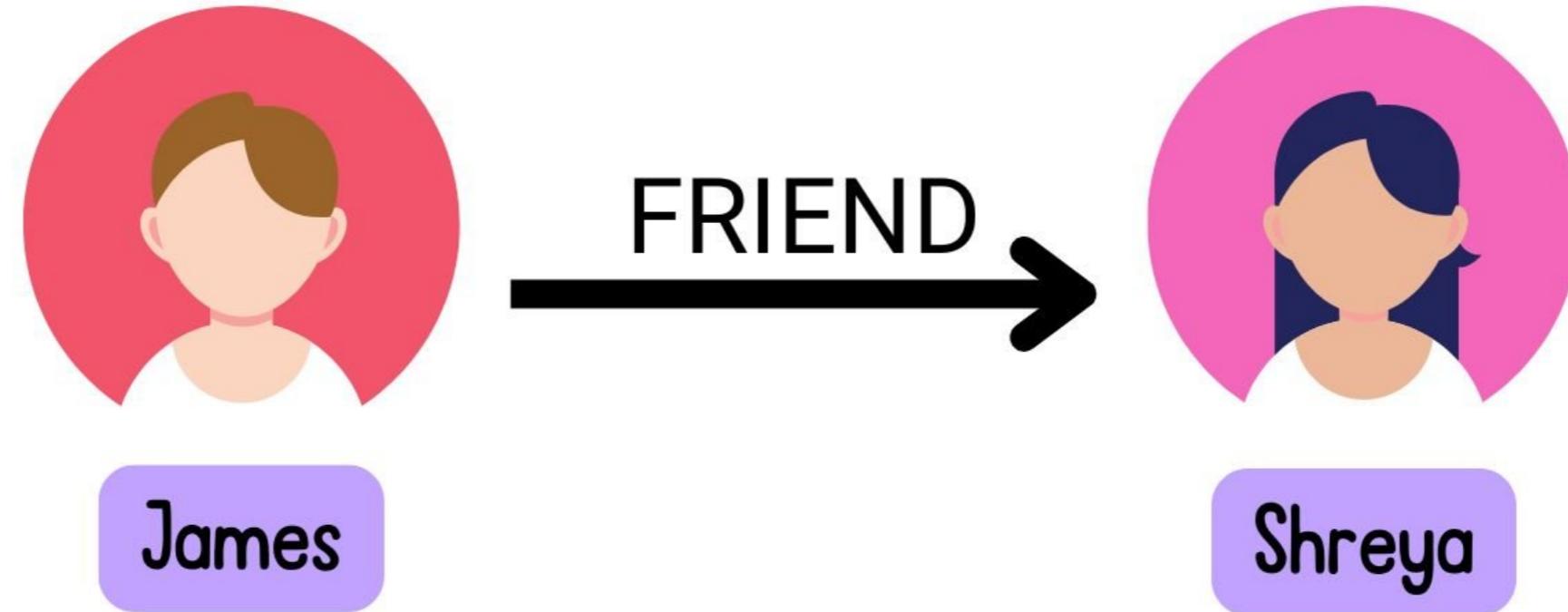
Querying Neo4j - Cypher Query Language



MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend

Node Relationship Node
_____ _____ _____
LABEL PROPERTY VARIABLE

Querying Neo4j - Cypher Query Language



MATCH (james:Person {name: "James"})-[:FRIEND]->(friend) RETURN friend

Node Relationship Node
_____ _____
LABEL PROPERTY VARIABLE

Querying the LLM graph

```
results = graph.query("""  
MATCH (gpt4:Model {id: "Gpt-4"})-[:DEVELOPED_BY]->(org:Organization)  
RETURN org  
""")  
  
print(results)
```

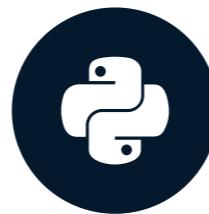
```
[{'org': {'id': 'Openai'}}]
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

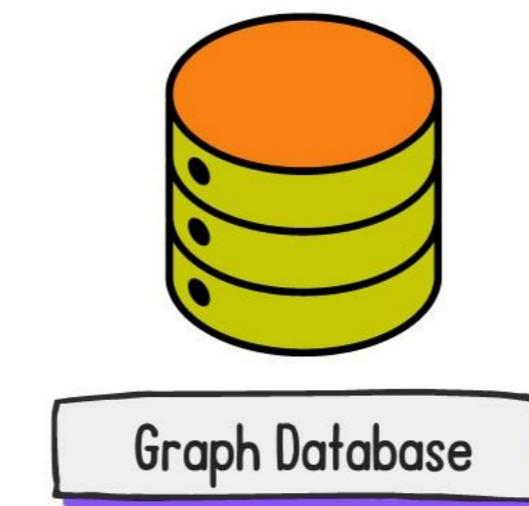
Creating the Graph RAG chain

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN



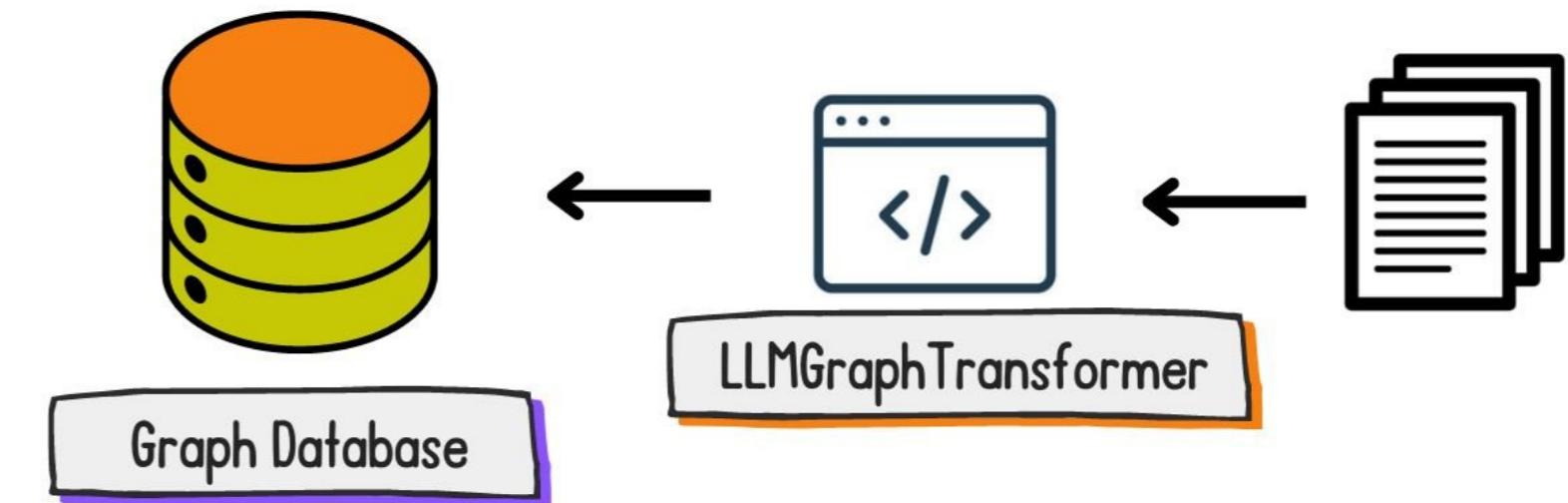
Meri Nova
Machine Learning Engineer

Building the Graph RAG architecture

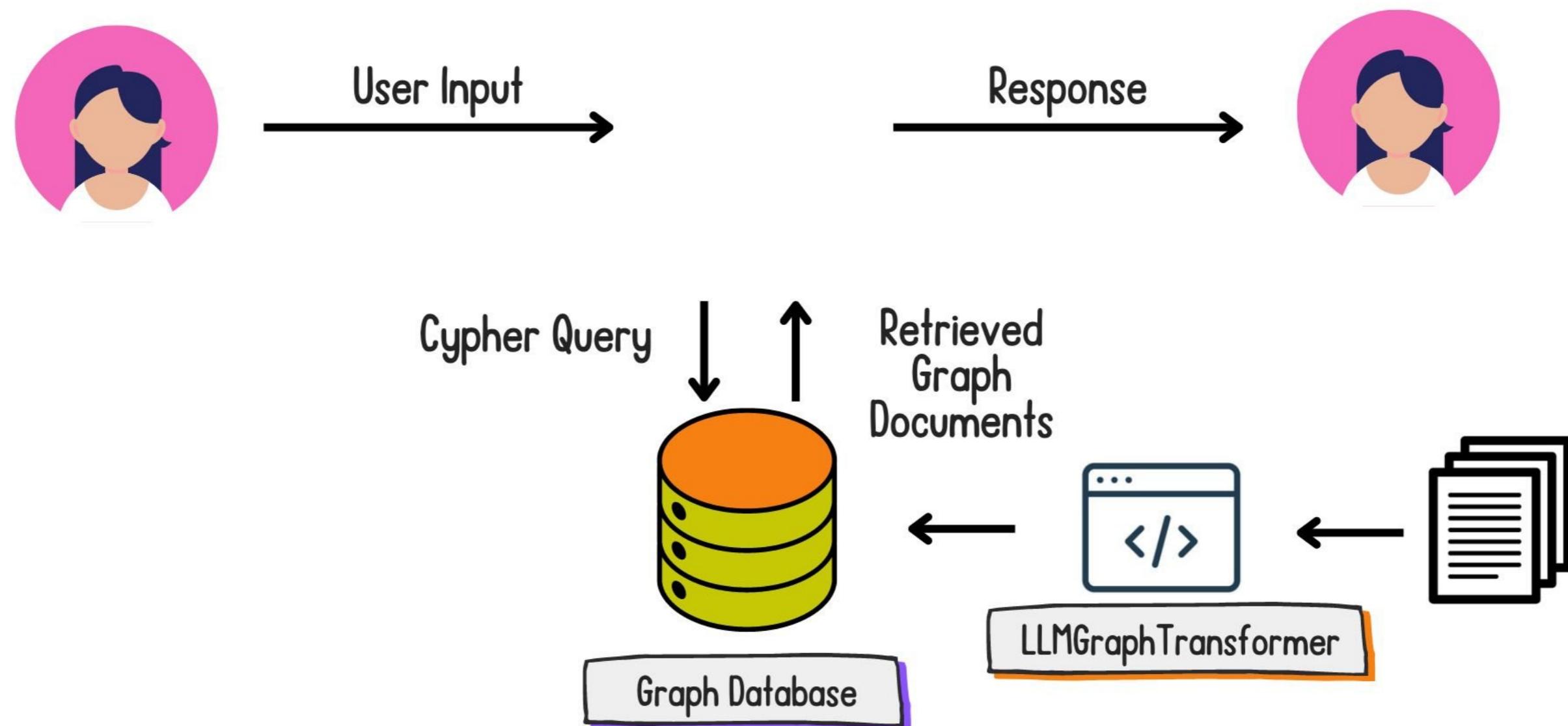


Graph Database

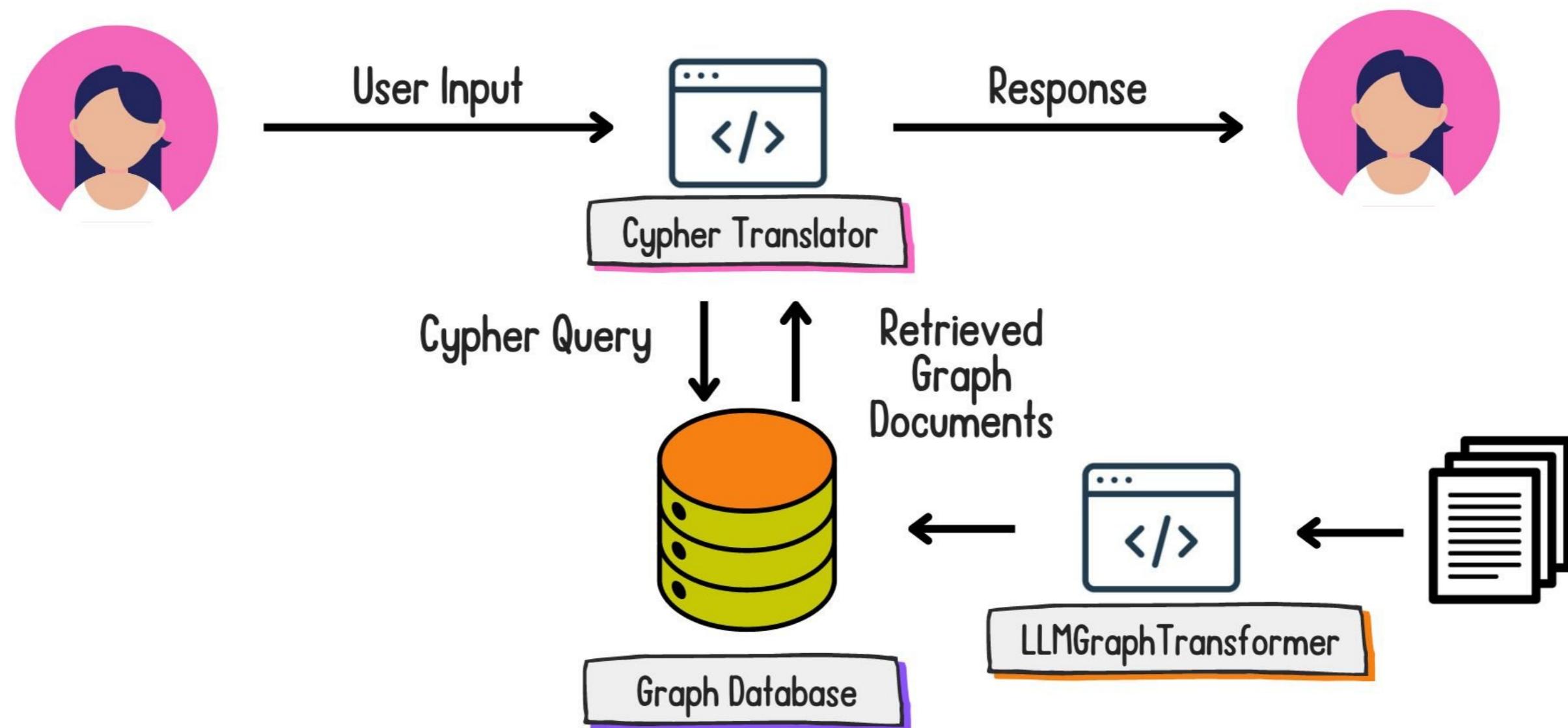
Building the Graph RAG architecture



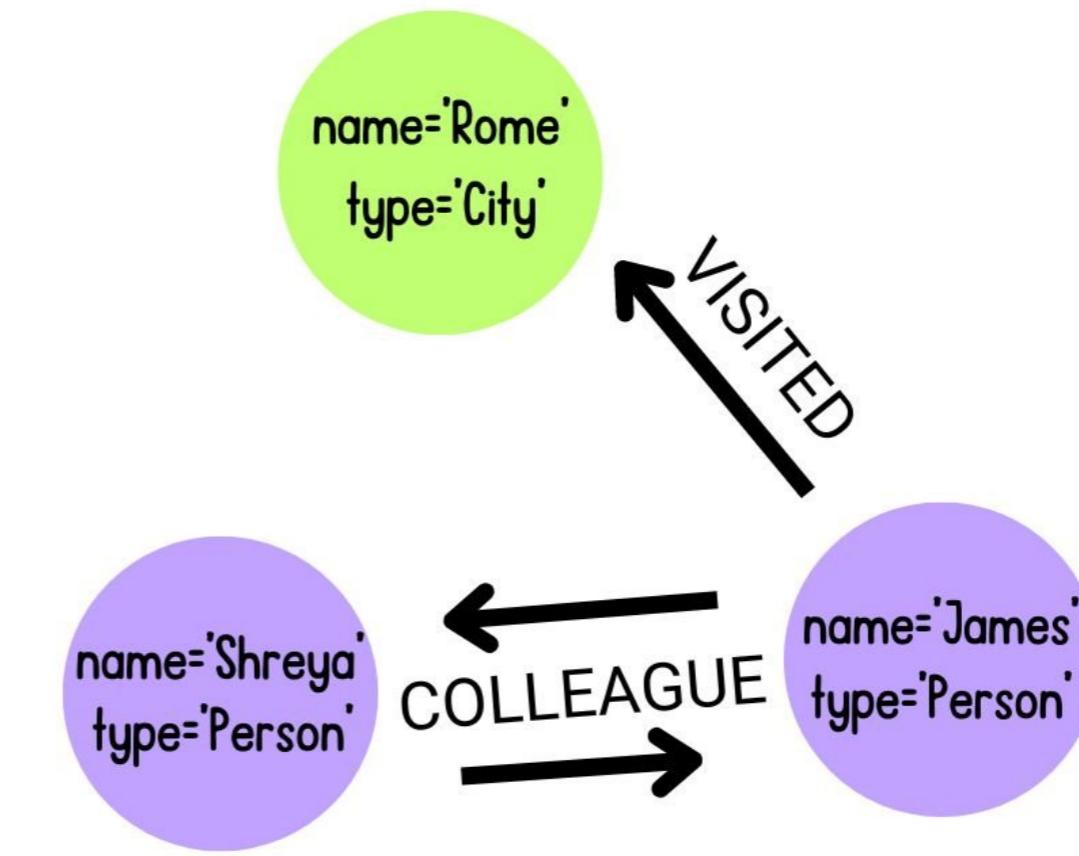
Building the Graph RAG architecture



Building the Graph RAG architecture



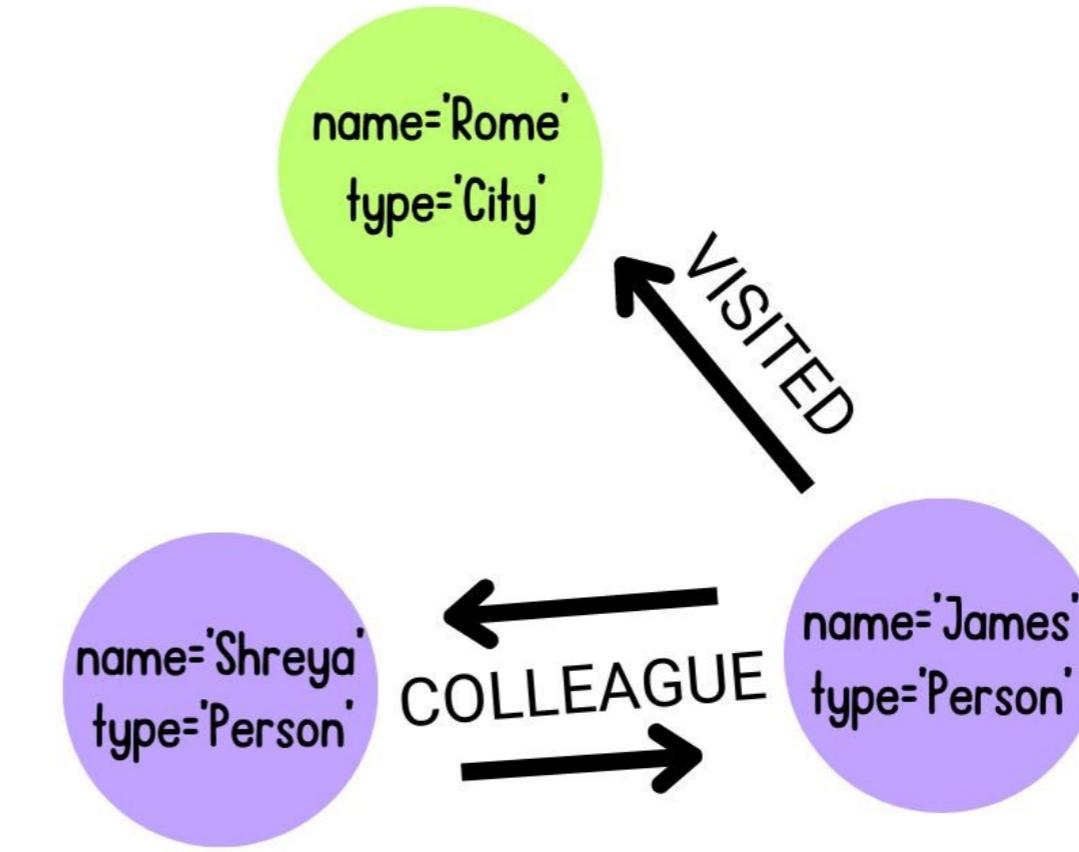
From user inputs to Cypher queries



From user inputs to Cypher queries

Where has James visited?

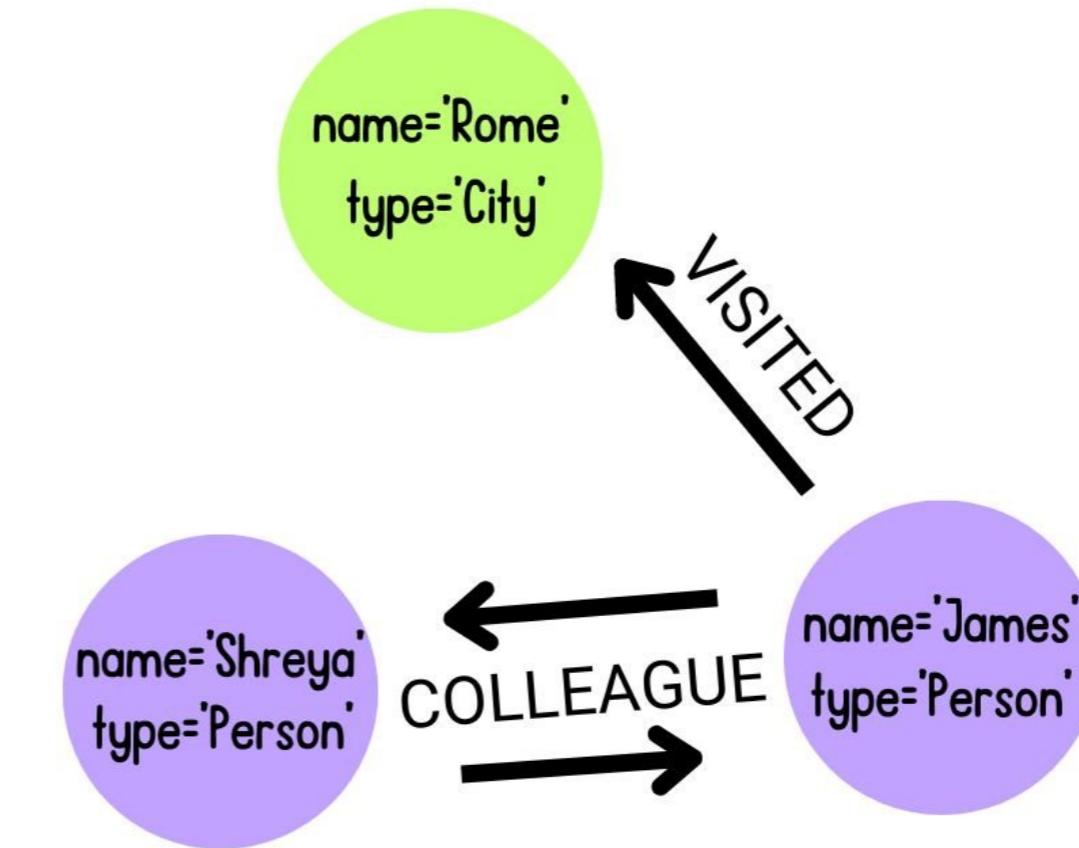
+



From user inputs to Cypher queries

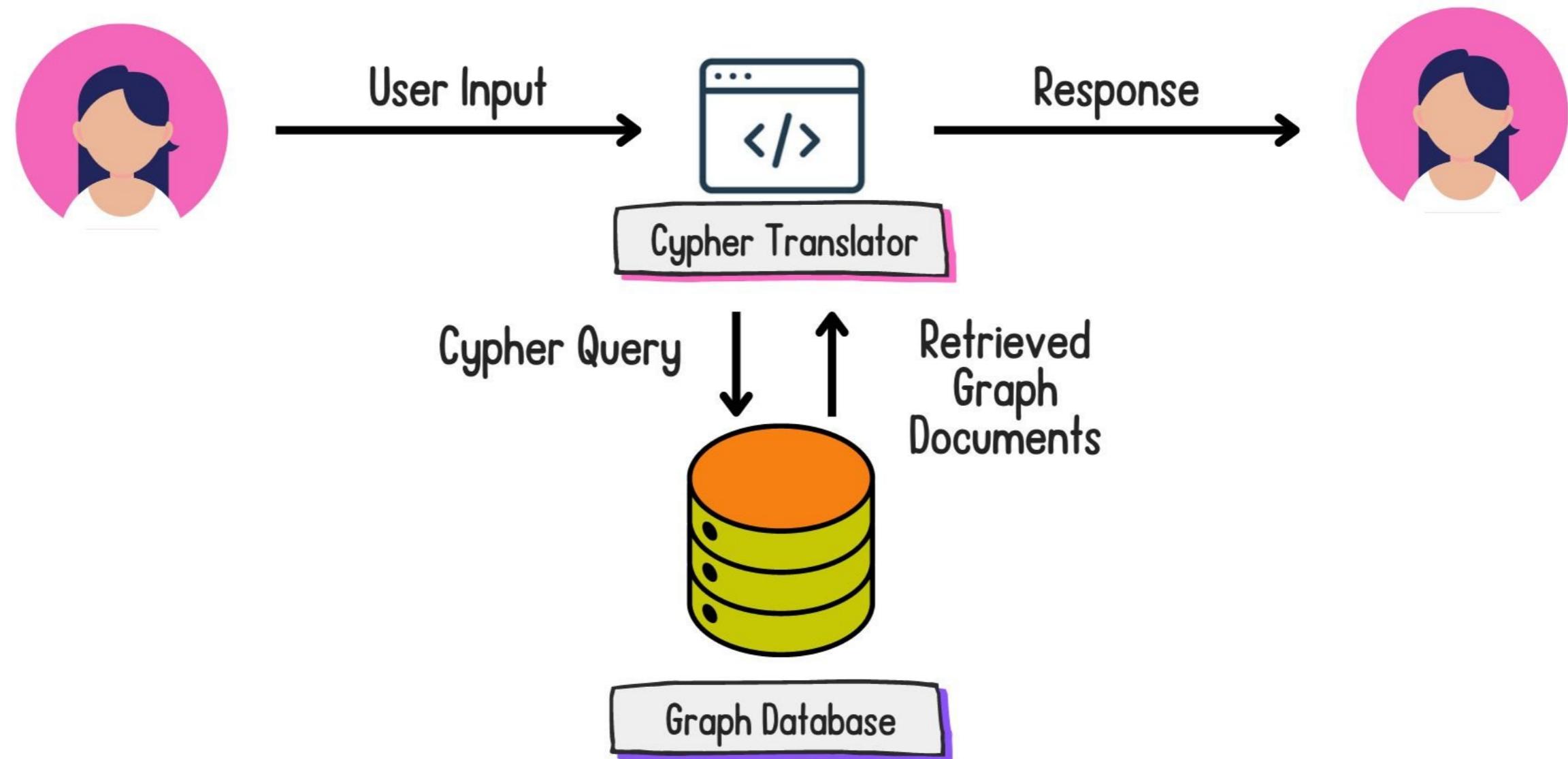
Where has James visited?

+

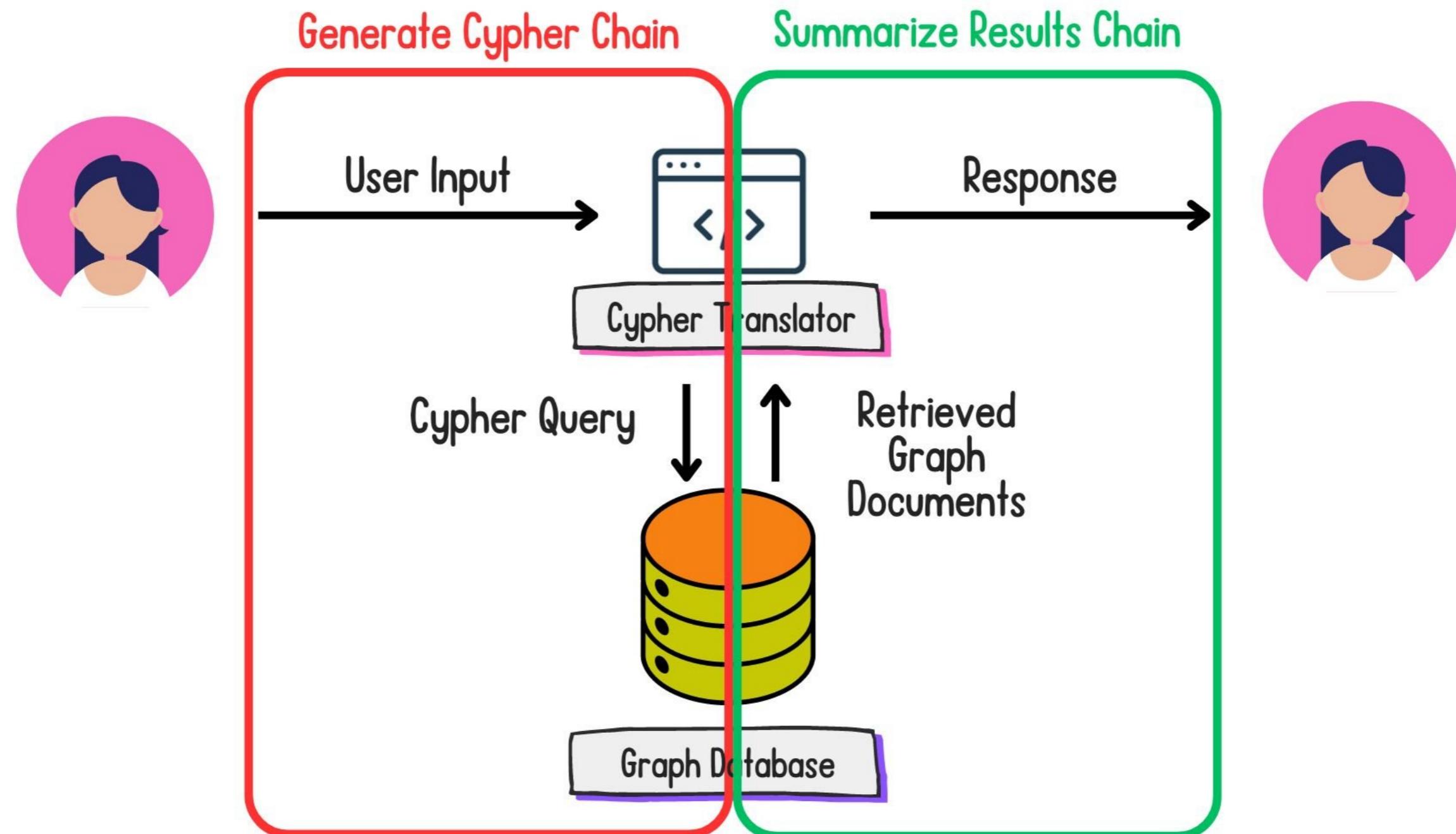


```
MATCH (james:Person{name:"James"}) -[:VISITED]> (location)
RETURN location
```

GraphCypherQAChain



GraphCypherQAChain



Refresh schema

```
graph.refresh_schema()  
print(graph.get_schema)
```

Node properties:

Document {title: STRING, id: STRING, text: STRING, summary: STRING, source: STRING}

Concept {id: STRING}

Organization {id: STRING}

Relationship properties:

The relationships:

(:Document)-[:MENTIONS]->(:Organization)

(:Concept)-[:DEVELOPED_BY]->(:Person)

Querying the graph

```
from langchain_community.chains.graph_qa.cypher import GraphCypherQACChain

chain = GraphCypherQACChain.from_llm(
    llm=ChatOpenAI(api_key="...", temperature=0), graph=graph, verbose=True
)

result = chain.invoke({"query": "What is the most accurate model?"})
```

¹ https://api.python.langchain.com/en/latest/chains/langchain_community.chains.graph_qa.cypher.GraphCypherQACChain.html

Querying the graph

```
print(f"Final answer: {result['result']}")
```

```
> Entering new GraphCypherQAChain chain...
```

```
Generated Cypher:
```

```
MATCH (m:Model)
RETURN m
ORDER BY m.accuracy DESC
LIMIT 1;
```

```
Full Context:
```

```
[{'m': {'id': 'Artificial Neural Networks'}}]
```

```
> Finished chain.
```

```
Final answer: Artificial Neural Networks
```

Customization

```
chain = GraphCypherQACChain.from_llm(  
    llm=ChatOpenAI(api_key="...", temperature=0), graph=graph, verbose=True  
)
```

- `qa_prompt` : Prompt template for result generation
- `cypher_prompt` : Prompt template for Cypher generation
- `cypher_llm` : LLM for Cypher generation
- `qa_llm` : LLM for result generation

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Improving graph retrieval

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN



Meri Nova
Machine Learning Engineer

Techniques

Main limitation: reliability of user → Cypher translation

Strategies to improve graph retrieval system:

- Filtering Graph Schema
- Validating the Cypher Query
- Few-shot prompting

Filtering

```
from langchain_community.chains.graph_qa.cypher import GraphCypherQACChain

llm = ChatOpenAI(api_key="...", model="gpt-4o-mini", temperature=0)

chain = GraphCypherQACChain.from_llm(
    graph=graph, llm=llm, exclude_types=["Concept"], verbose=True
)
print(graph.get_schema)
```

Node properties:

Document {title: STRING, id: STRING, text: STRING, summary: STRING, source: STRING}

Organization {id: STRING}

Validating the Cypher query

- Difficulty in interpreting the *direction* of relationships

```
chain = GraphCypherQAChain.from_llm(  
    graph=graph, llm=llm, verbose=True, validate_cypher=True  
)
```

1. Detects nodes and relationships
2. Determines the directions of the relationship
3. Checks the graph schema
4. Update the direction of relationships

Few-shot prompting

```
examples = [
    {
        "question": "How many notable large language models are mentioned in the article?",
        "query": "MATCH (m:Concept {id: 'Large Language Model'}) RETURN count(DISTINCT m)",
    },
    {
        "question": "Which companies or organizations have developed the large language models mentioned?",
        "query": "MATCH (o:Organization)-[:DEVELOPS]->(m:Concept {id: 'Large Language Model'}) RETURN DISTINCT o.id",
    },
    {
        "question": "What is the largest model size mentioned in the article, in terms of number of parameters?",
        "query": "MATCH (m:Concept {id: 'Large Language Model'}) RETURN max(m.parameters) AS largest_model",
    },
]
```

Implementing few-shot prompting

```
from langchain_core.prompts import FewShotPromptTemplate, PromptTemplate

example_prompt = PromptTemplate.from_template(
    "User input: {question}\nCypher query: {query}"
)

cypher_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="You are a Neo4j expert. Given an input question, create a syntactically correct
    Cypher query to run.\n\nHere is the schema information\n{schema}.\n\n
    Below are a number of examples of questions and their corresponding Cypher queries.",
    suffix="User input: {question}\nCypher query: ",
    input_variables=["question"],
)
```

Complete prompt

You are a Neo4j expert. Given an input question, create a syntactically correct Cypher query to run.

Below are a number of examples of questions and their corresponding Cypher queries.

User input: How many notable large language models are mentioned in the article?

Cypher query: MATCH (p:Paper) RETURN count(DISTINCT p)

User input: Which companies or organizations have developed the large language models?

Cypher query: MATCH (o:Organization)-[:DEVELOPS]->(m:Concept {id: 'Large Language Model'}) RETURN DISTINCT o.id

User input: What is the largest model size mentioned in the article, in terms of number of parameters?

Cypher query: MATCH (m:Concept {id: 'Large Language Model'}) RETURN max(m.parameters) AS largest_model

User input: How many papers were published in 2016?

Cypher query:

Adding the few-shot examples

```
chain = GraphCypherQACChain.from_llm(  
    graph=graph, llm=llm, cypher_prompt=cypher_prompt,  
    verbose=True, validate_cypher=True  
)
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Congratulations!

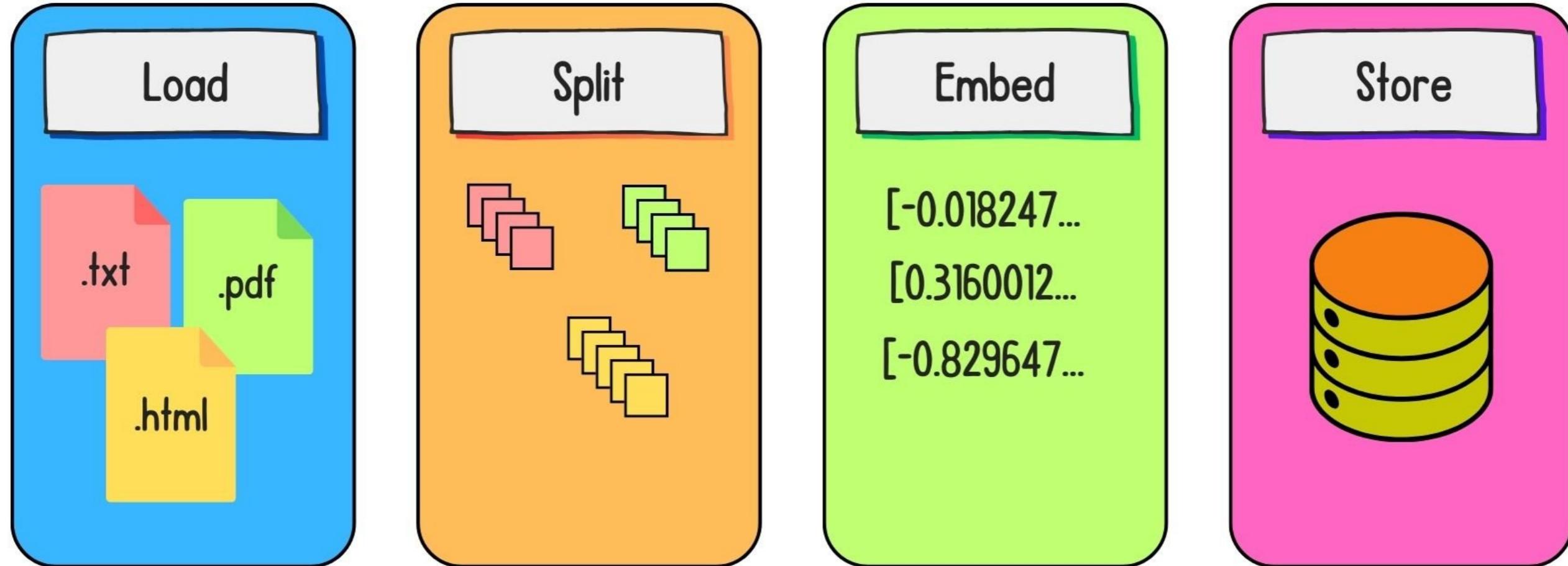
RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN



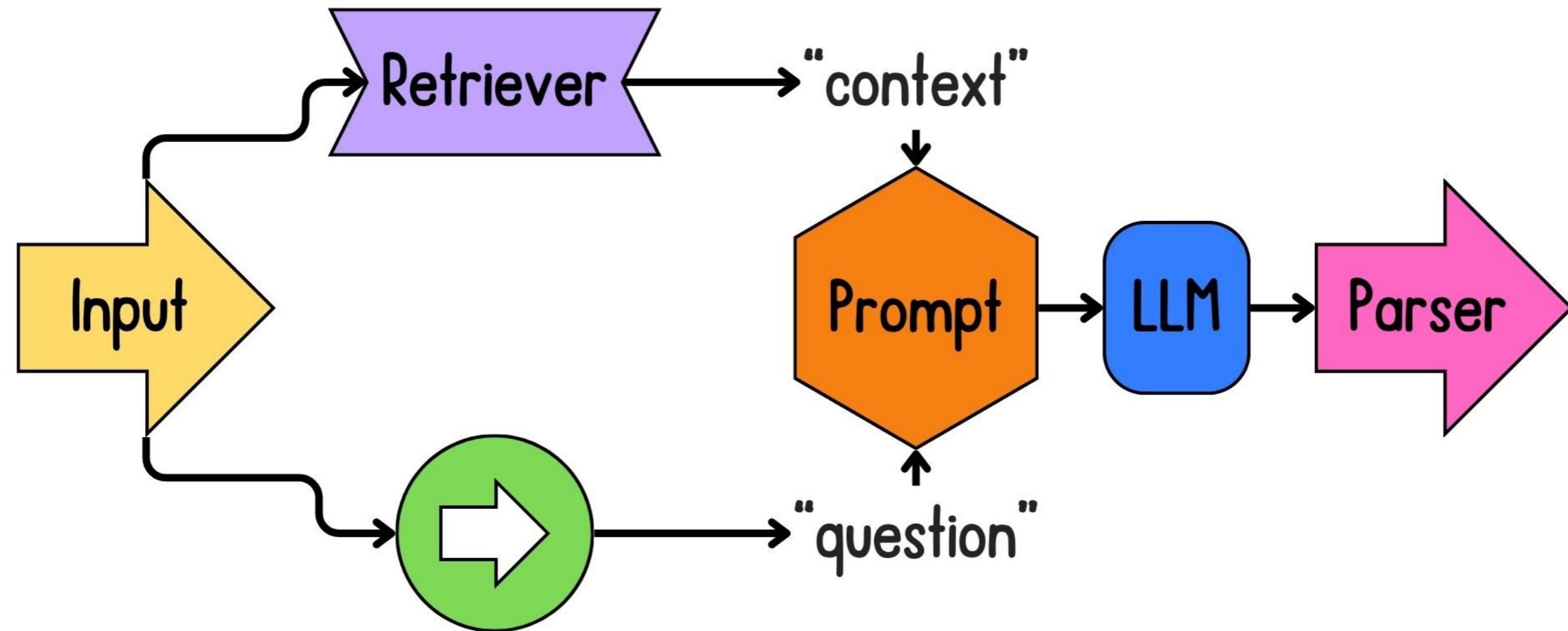
Meri Nova

Machine Learning Engineer

Chapter 1



Chapter 1



RunnablePassthrough

Chapter 2

Markdown files

- UnstructuredMarkdownLoader

Python files

- PythonLoader
- language=Language.PYTHON

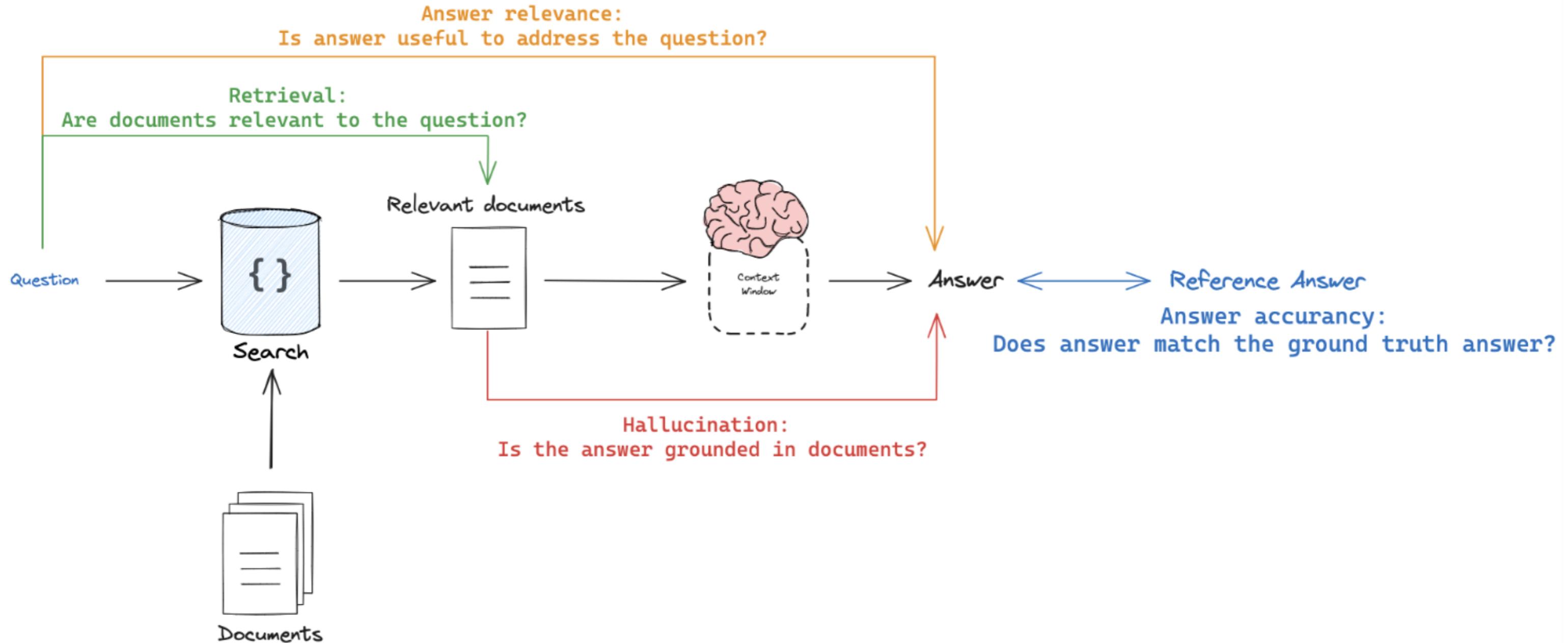
Token splitting → TokenTextSplitter

Semantic splitting → SemanticChunker

RAG applications has numerous use cases,
but are most frequently deployed in chatbots.

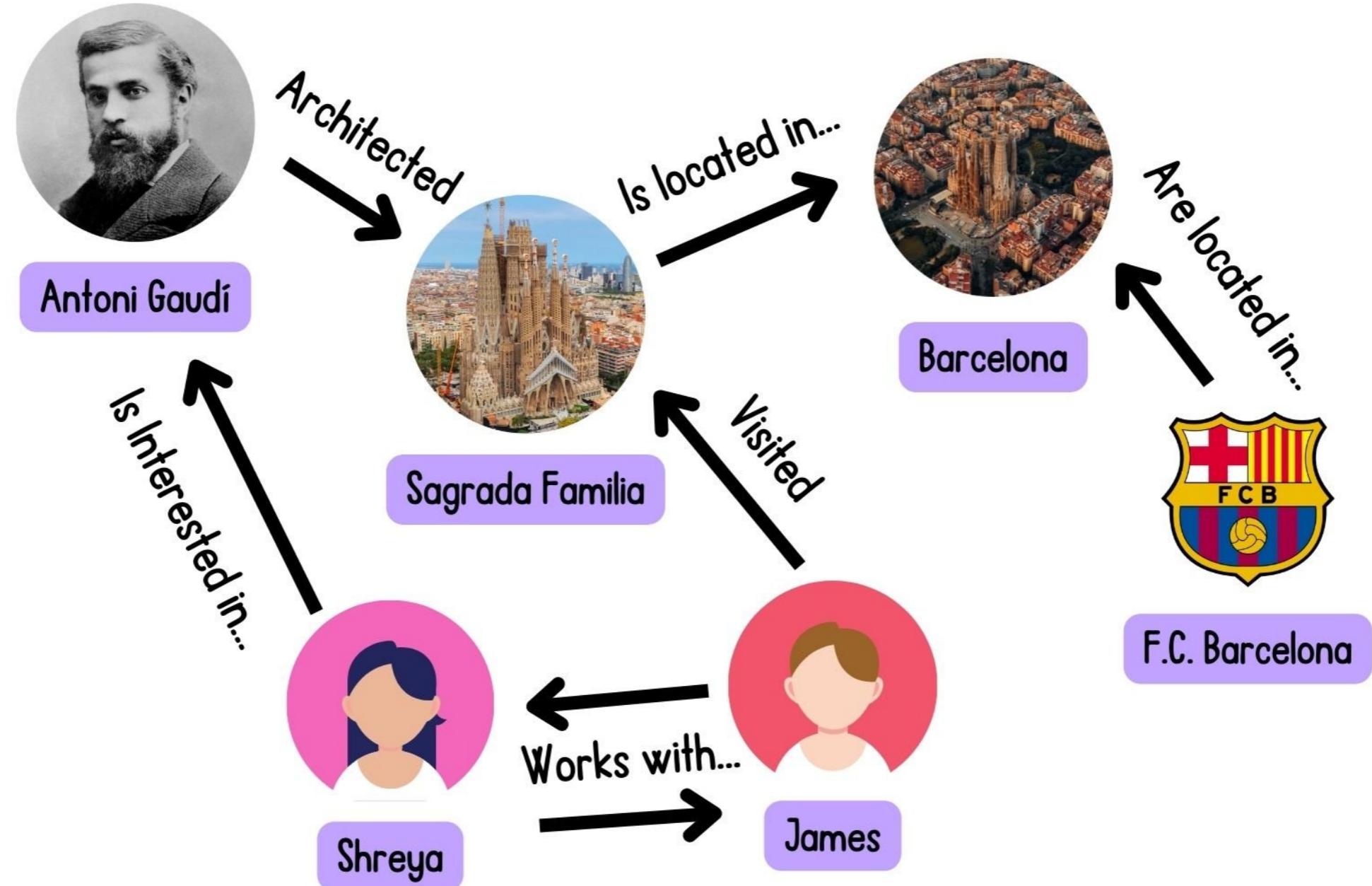
Domestic dogs are descendants from an
extinct population of Pleistocene wolves over
14,000 years ago.

Chapter 2

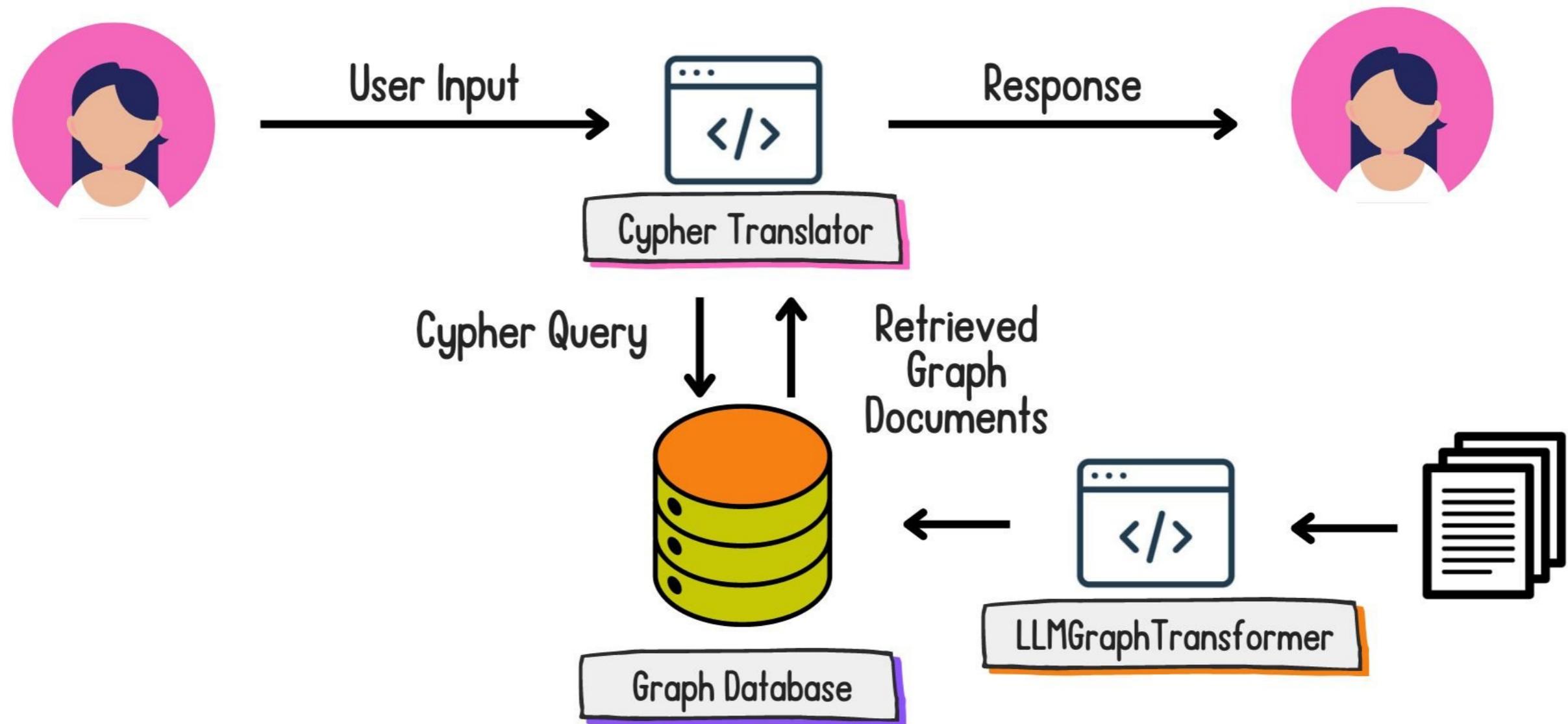


¹ Image Credit: LangSmith

Chapter 3



Chapter 3



Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN