Graphs and RAG

GRAPH RAG WITH LANGCHAIN AND NEO4J

Adam Cowley
Manager, Developer Education at Neo4j

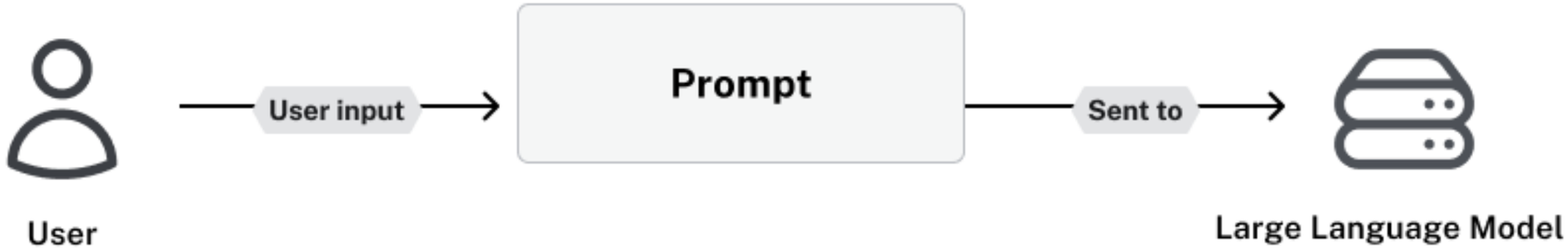
datacamp

Meet your instructor

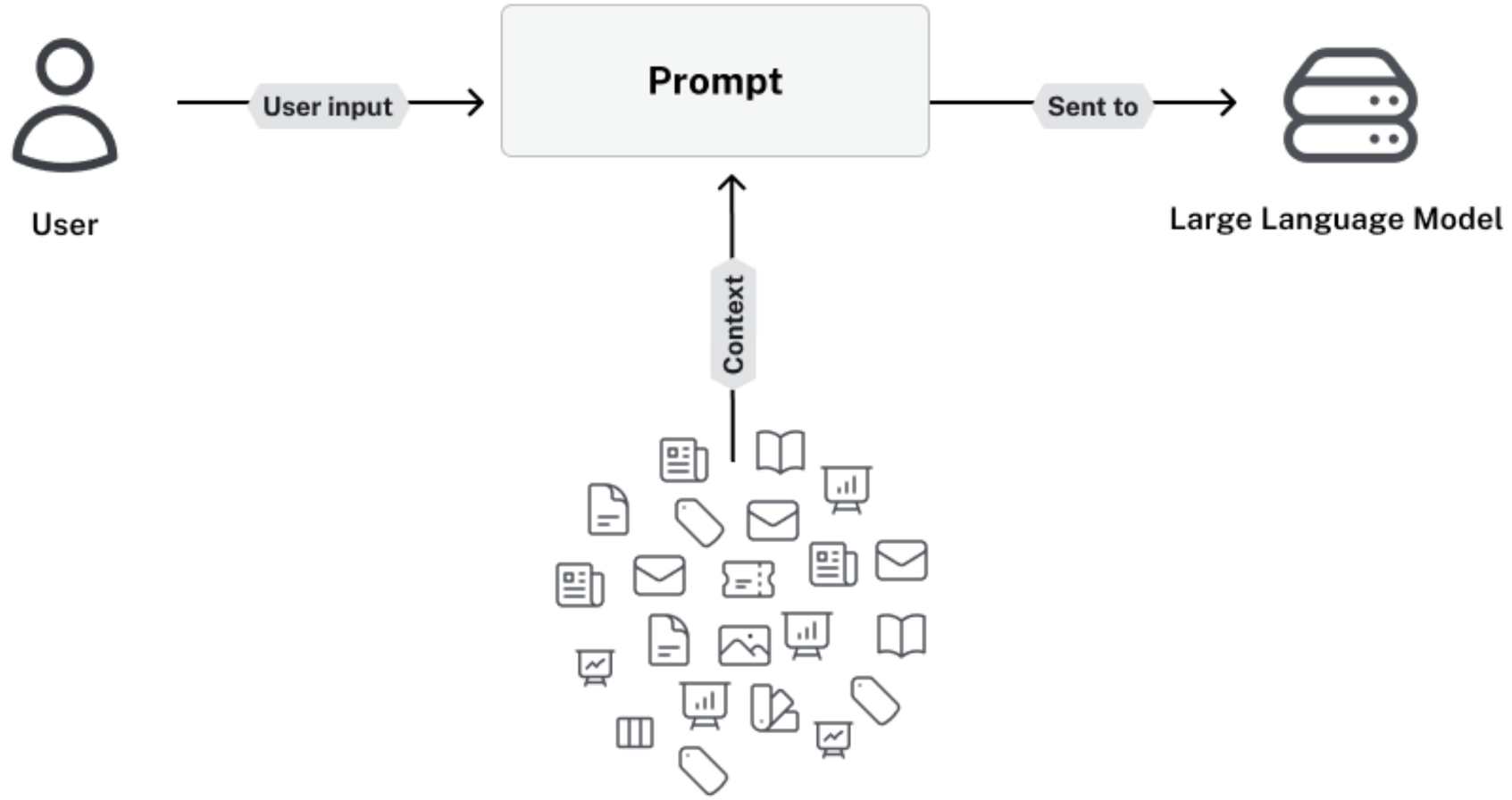
- Manager, Developer Education at Neo4j
- Developer Education through **Neo4j GraphAcademy**
- 20+ years of software development experience
- 10+ years Neo4j experience

[1] https://graphacademy.neo4j.com

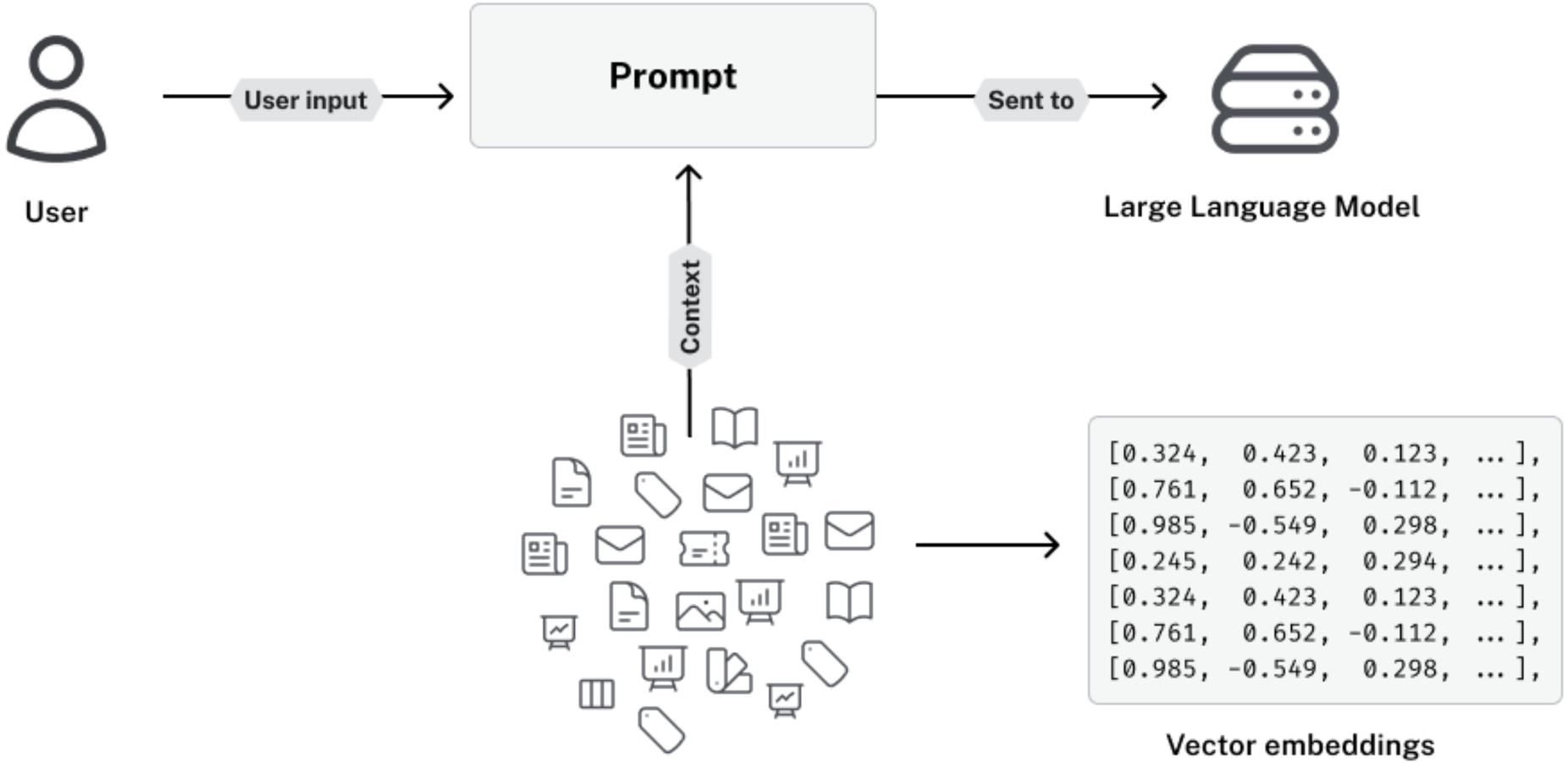
The R in RAG



The R in RAG


User
User input
Prompt
Sent to
Large Language Model
Context

The R in RAG


User
User input
Prompt
Sent to
Large Language Model
Context
[0.324, 0.423, 0.123, ...],
[0.761, 0.652, -0.112, ...],
[0.985, -0.549, 0.298, ...],
[0.245, 0.242, 0.294, ...],
[0.324, 0.423, 0.123, ...],
[0.761, 0.652, -0.112, ...],
[0.985, -0.549, 0.298, ...],
Vector embeddings

Where does semantic search fail?

Vectors work well for

- Fuzzy or Open-Ended questions

Vectors are ineffective for

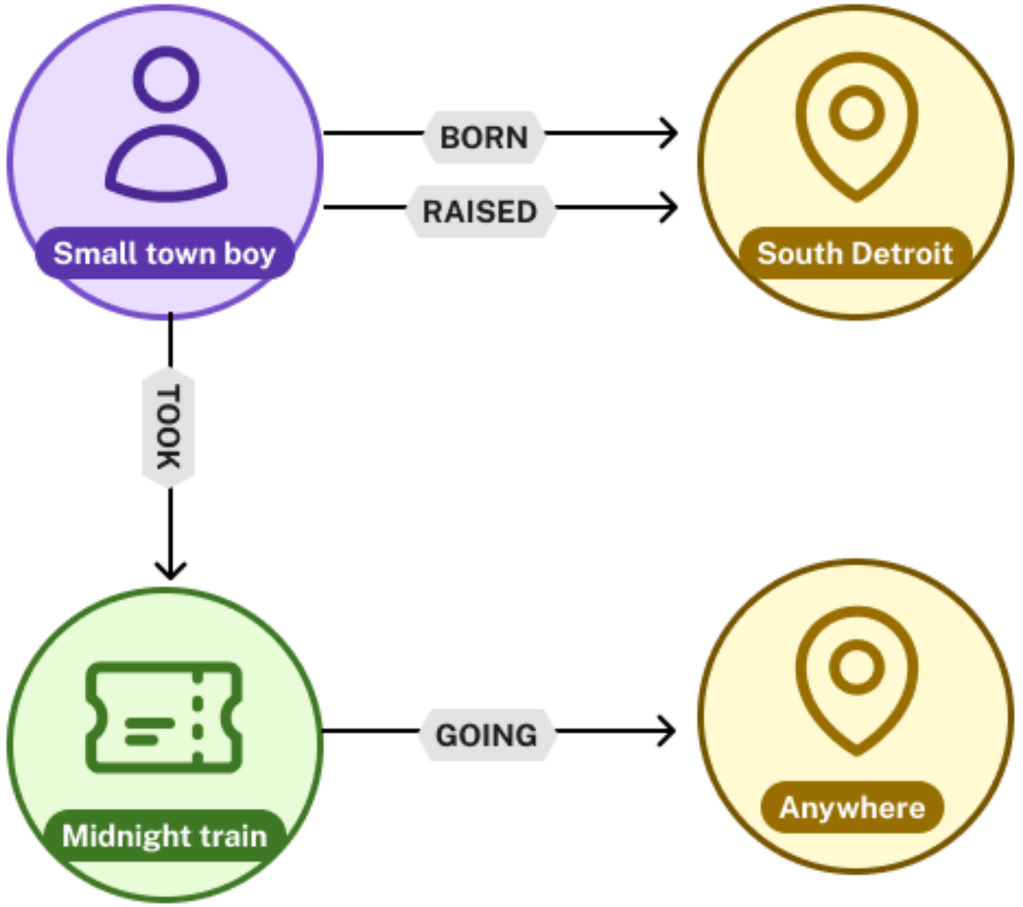
- Highly Specific or Fact-Based Questions
- Numerical or Exact-Match Queries
- Logical Queries

> *What does Paul Graham think about Generative AI?*

> *How many Generative AI Startups has Paul Graham invested in?*

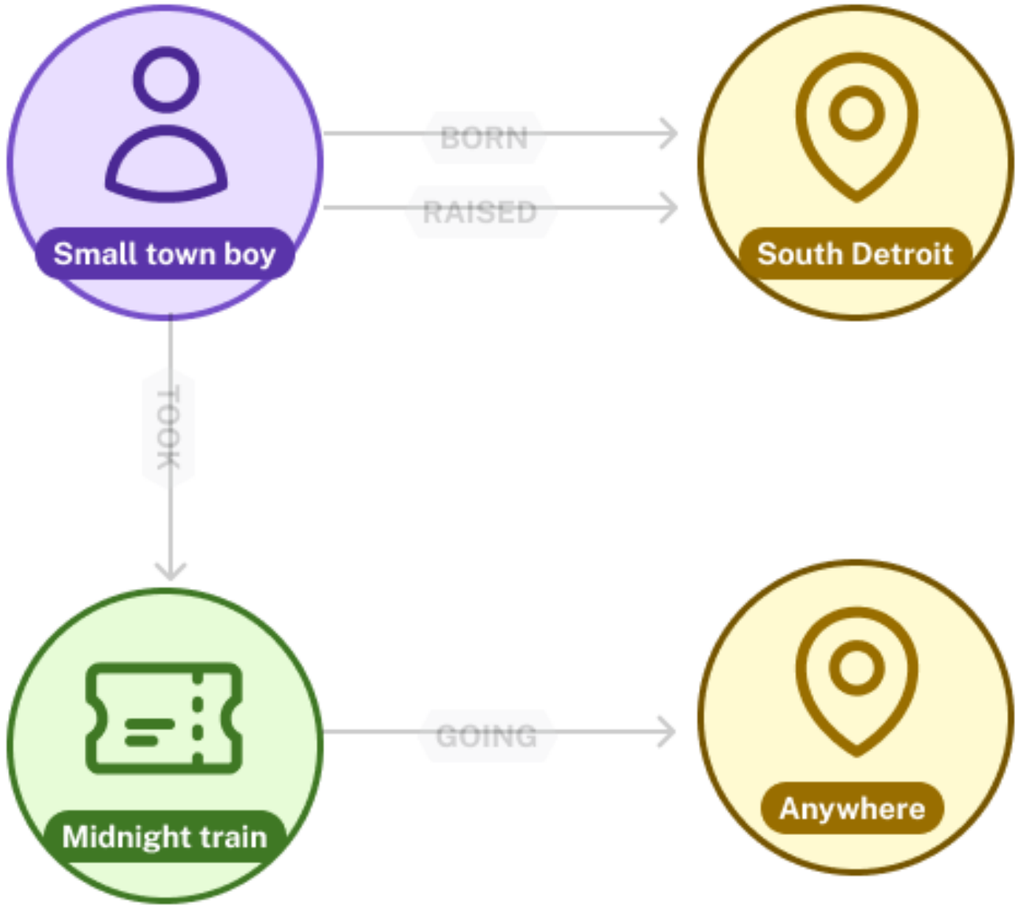
Vectors vs. knowledge graphs

Text	Vector Embedding
He's just a city boy born	`[0.12, -0.34, 0.56, 0.78, ..., -0.91]`
born and raised in South Detroit	`[0.22, 0.45, -0.67, 0.11, ..., 0.33]`
He took the midnight train	`[-0.55, 0.89, 0.12, -0.44, ..., 0.67]`
going anywhere	`[0.78, -0.23, 0.45, 0.91, ..., -0.12]`



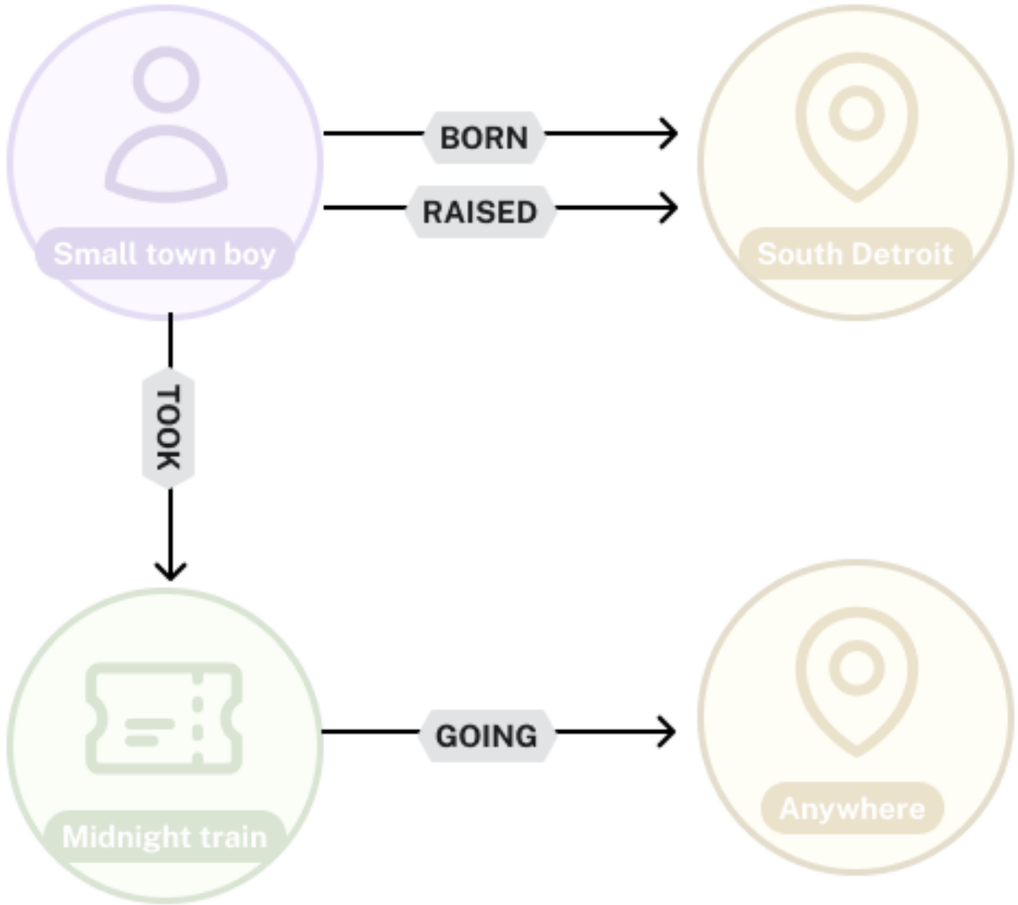
Vectors vs. knowledge graphs

Text	Vector Embedding
He's just a city boy born	`[0.12, -0.34, 0.56, 0.78, ..., -0.91]`
born and raised in South Detroit	`[0.22, 0.45, -0.67, 0.11, ..., 0.33]`
He took the midnight train	`[-0.55, 0.89, 0.12, -0.44, ..., 0.67]`
going anywhere	`[0.78, -0.23, 0.45, 0.91, ..., -0.12]`



Vectors vs. knowledge graphs

Text	Vector Embedding
He's just a city boy born	`[0.12, -0.34, 0.56, 0.78, ..., -0.91]`
born and raised in South Detroit	`[0.22, 0.45, -0.67, 0.11, ..., 0.33]`
He took the midnight train	`[-0.55, 0.89, 0.12, -0.44, ..., 0.67]`
going anywhere	`[0.78, -0.23, 0.45, 0.91, ..., -0.12]`



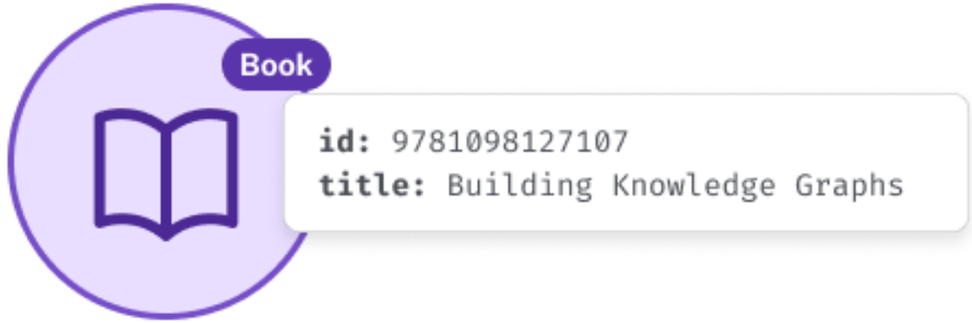
Knowledge graphs and Neo4j

- Neo4j is the world's leading graph database
- Flexible, schema-optional
- `langchain-neo4j` LangChain integration
- **LCEL**: *LangChain Expression Language*

[1] https://db-engines.com/en/ranking/graph%20dbms

Nodes represent things

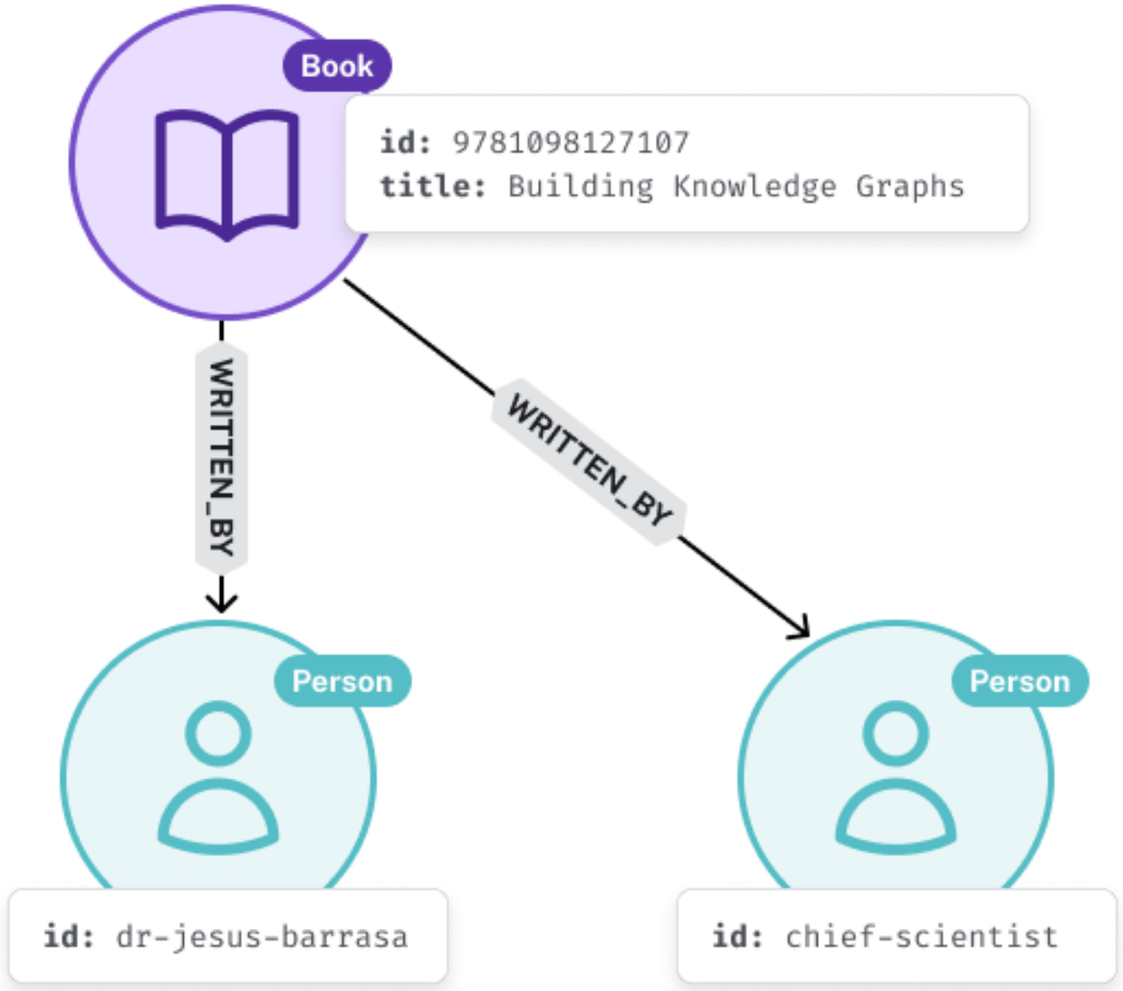
```
from langchain_neo4j.graphs.graph_document \
  import Node

book = Node(
    type="Book",
    id=f"9781098127107",
    properties={
        "title": "Building Knowledge Graphs"
    }
)
```



Relationships connect nodes

```
from langchain_neo4j.graphs.graph_document \
  import Node, Relationship

book = Node(type="Book", id=f"9781098127107")

jesus = Node(type="Person", id="dr-jesus-barrasa")
jim = Node(type="Person", id="chief-scientist")
```

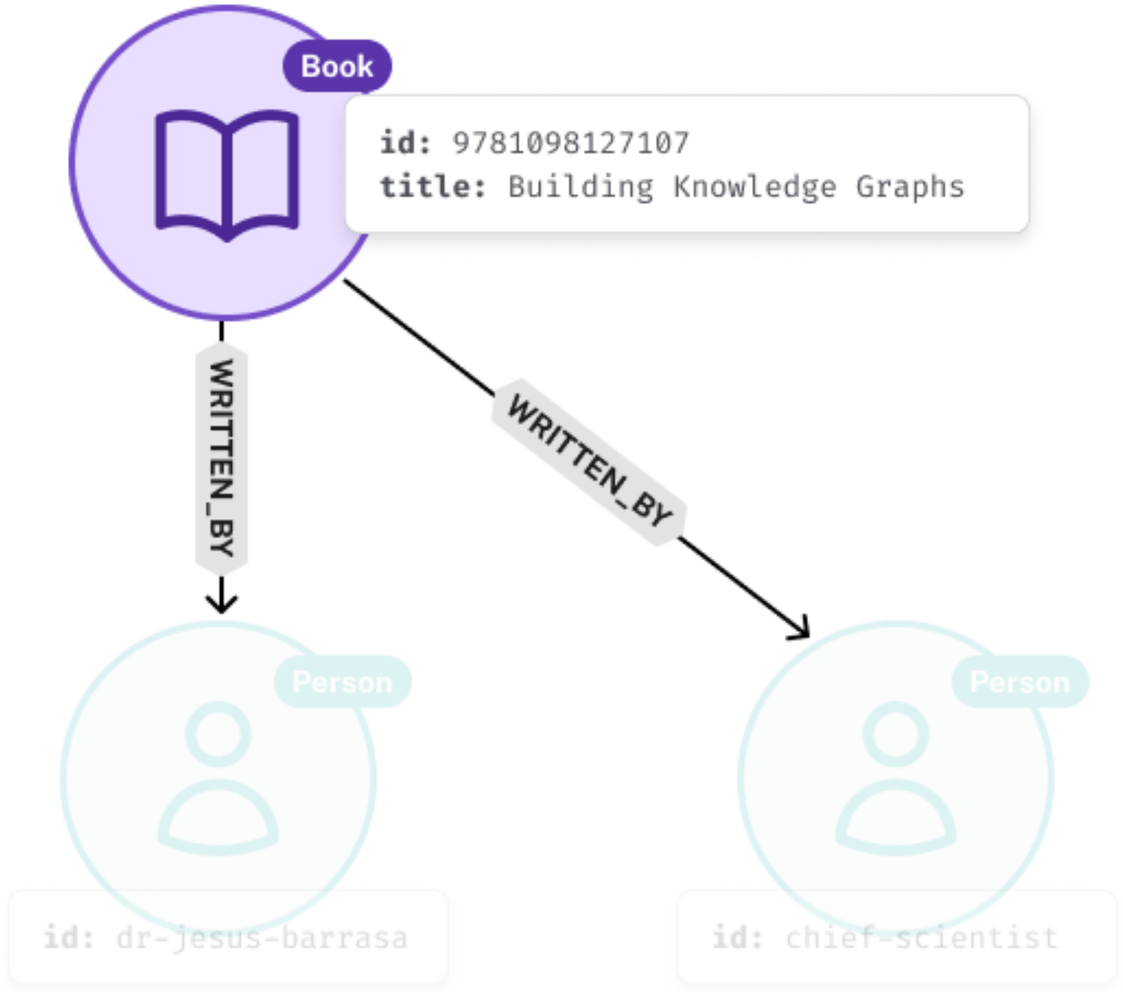
```
for author in [jesus, jim]:
    relationship = Relationship(




    )
```



Relationships connect nodes

```
from langchain_neo4j.graphs.graph_document \
  import Node, Relationship

book = Node(type="Book", id=f"9781098127107")

jesus = Node(type="Person", id="dr-jesus-barrasa")
jim = Node(type="Person", id="chief-scientist")
```

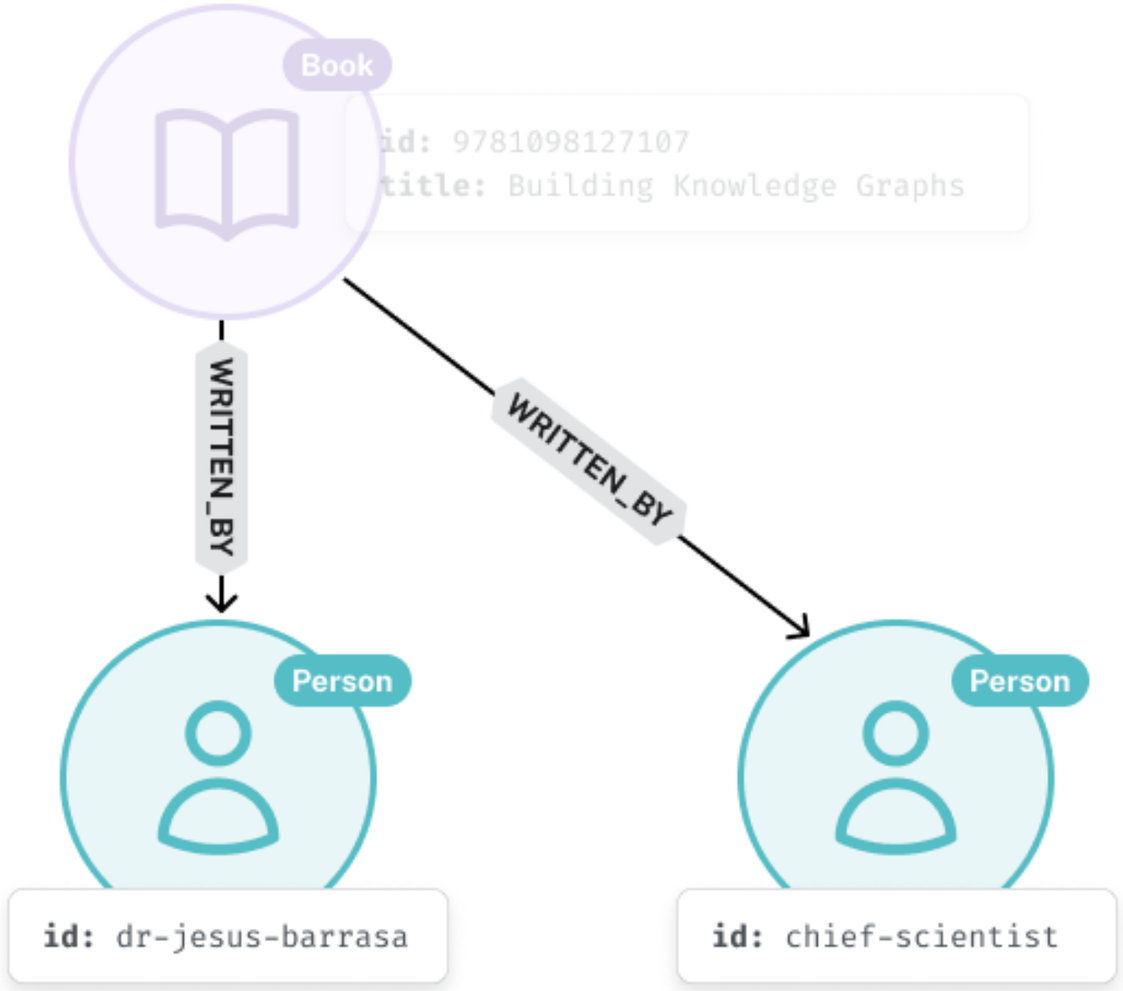
```
for author in [jesus, jim]:
    relationship = Relationship(
        source=book,



    )
```



Relationships connect nodes

```
from langchain_neo4j.graphs.graph_document \
  import Node, Relationship

book = Node(type="Book", id=f"9781098127107")

jesus = Node(type="Person", id="dr-jesus-barrasa")
jim = Node(type="Person", id="chief-scientist")
```

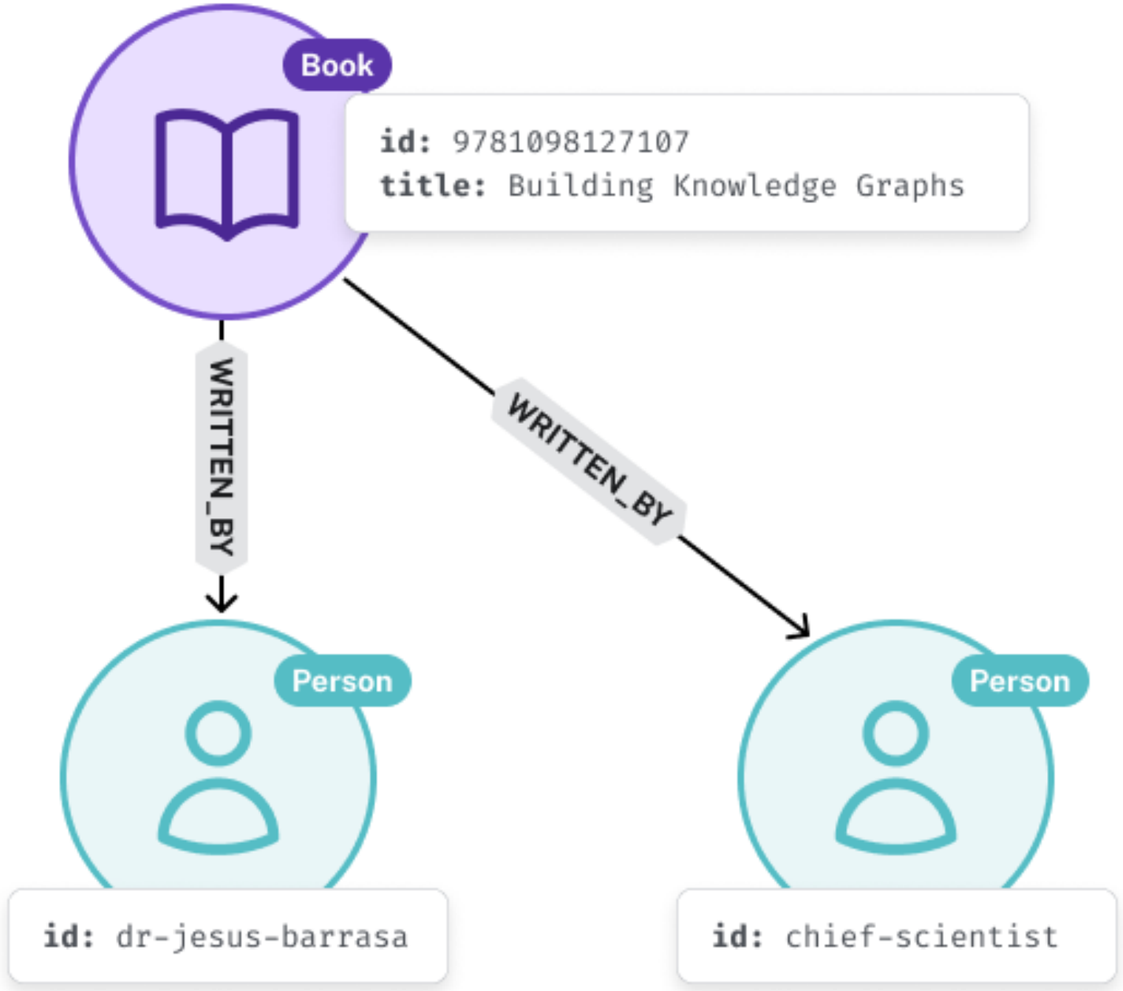
```
for author in [jesus, jim]:
    relationship = Relationship(
        source=book,
        destination=author,



    )
```



Relationships connect nodes

```
from langchain_neo4j.graphs.graph_document \
  import Node, Relationship

book = Node(type="Book", id=f"9781098127107")

jesus = Node(type="Person", id="dr-jesus-barrasa")
jim = Node(type="Person", id="chief-scientist")
```

```
for author in [jesus, jim]:
    relationship = Relationship(
        source=book,
        destination=author,
        type="WRITTEN_BY"
        properties=dict(...)
    )
```



Connecting to Neo4j

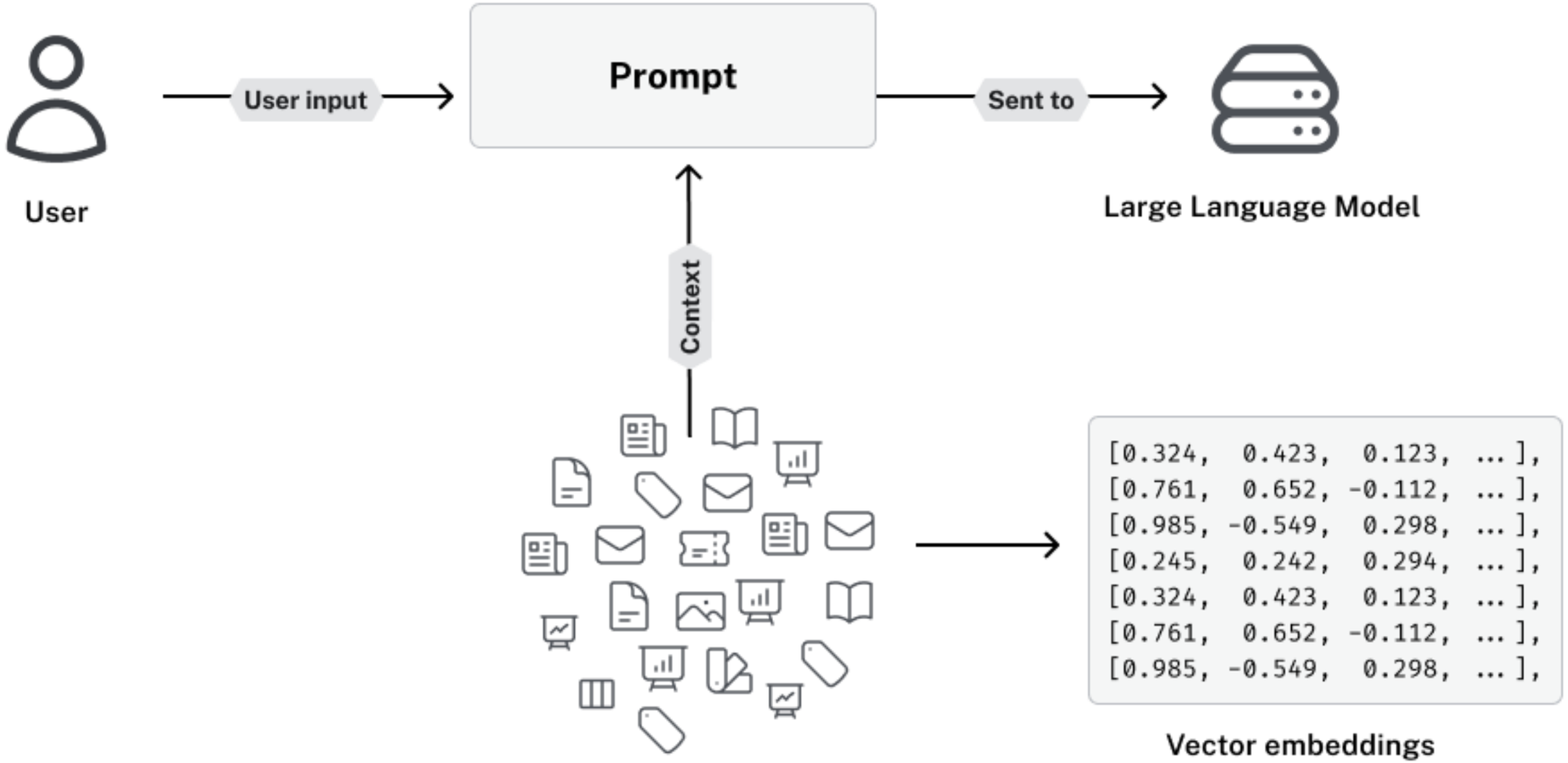
```
from langchain_neo4j import Neo4jGraph


graph = Neo4jGraph(
    url=NEO4J_URI,
    username=NEO4J_USERNAME,
    password=NEO4J_PASSWORD
)
```

[1] https://neo4j.com/docs/getting-started/

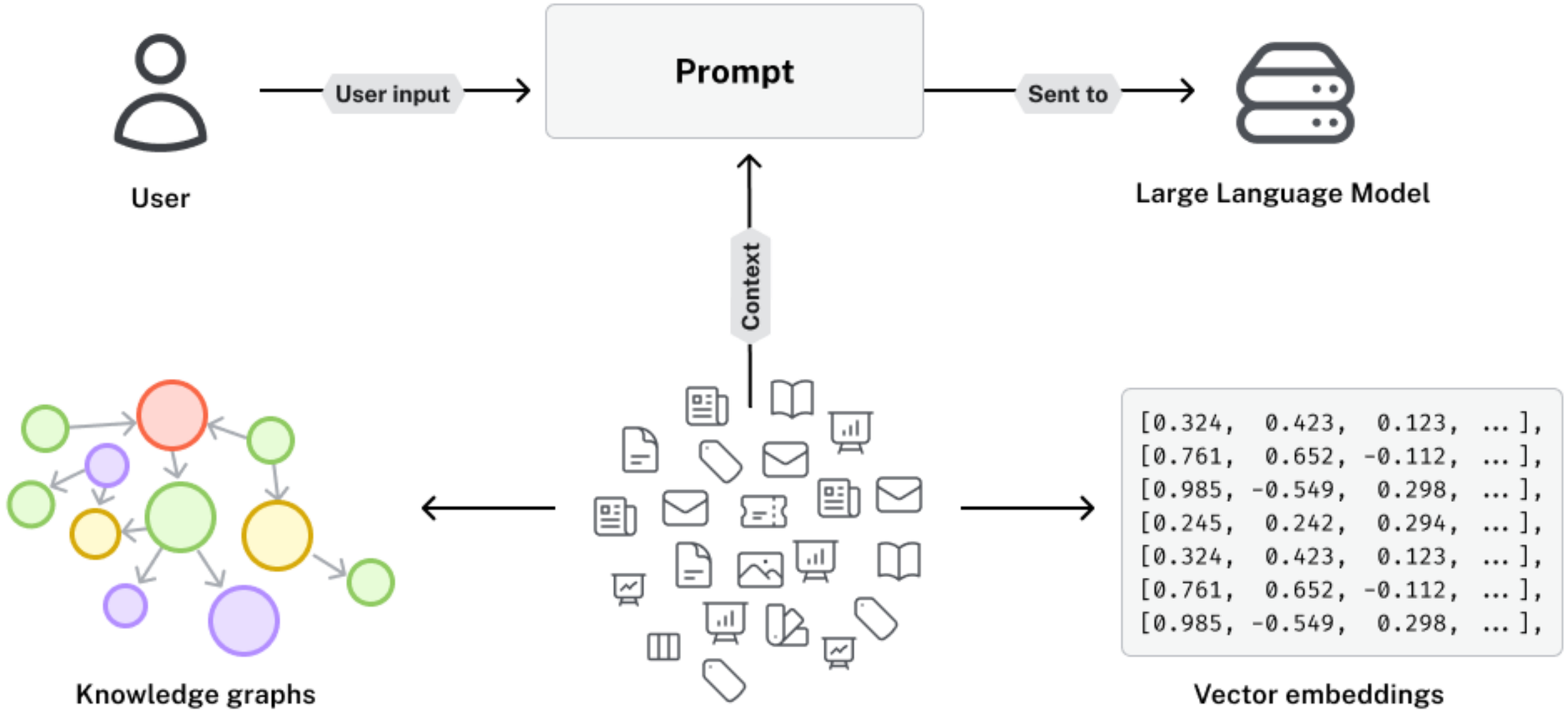
Saving graph documents

```
from langchain_neo4j.graphs.graph_document import GraphDocument

# Wrapper for Nodes and Relationships
doc = GraphDocument(
  nodes=[book, jesus, jim],
  relationships=[jesus_wrote_book, jim_wrote_book]
)

# Save nodes and relationships to the database
graph.add_graph_documents([graph_document])
```

So, what is GraphRAG?



So, what is GraphRAG?



Let's practice!

GRAPH RAG WITH LANGCHAIN AND NEO4J

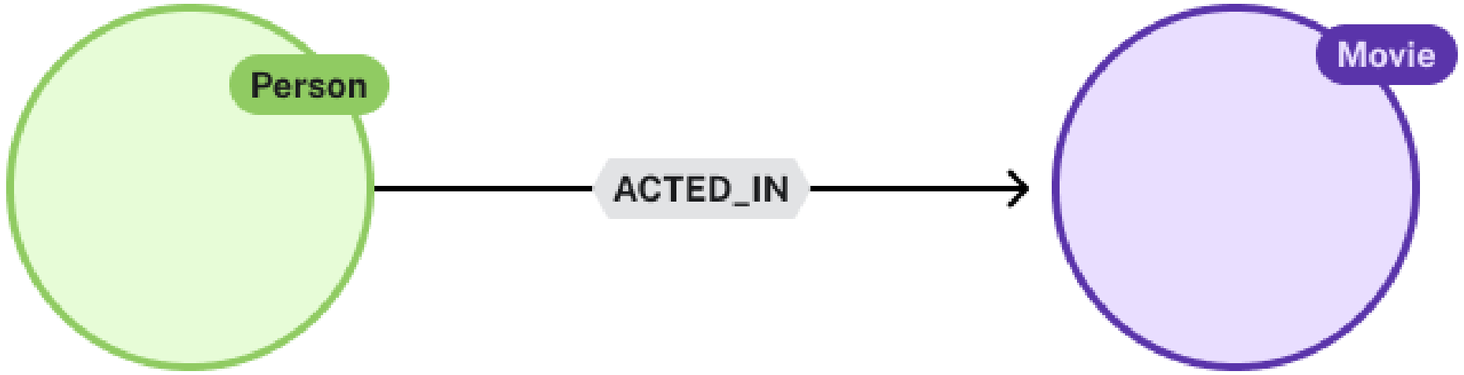
Querying a knowledge graph

GRAPH RAG WITH LANGCHAIN AND NEO4J

Adam Cowley
Manager, Developer Education at Neo4j

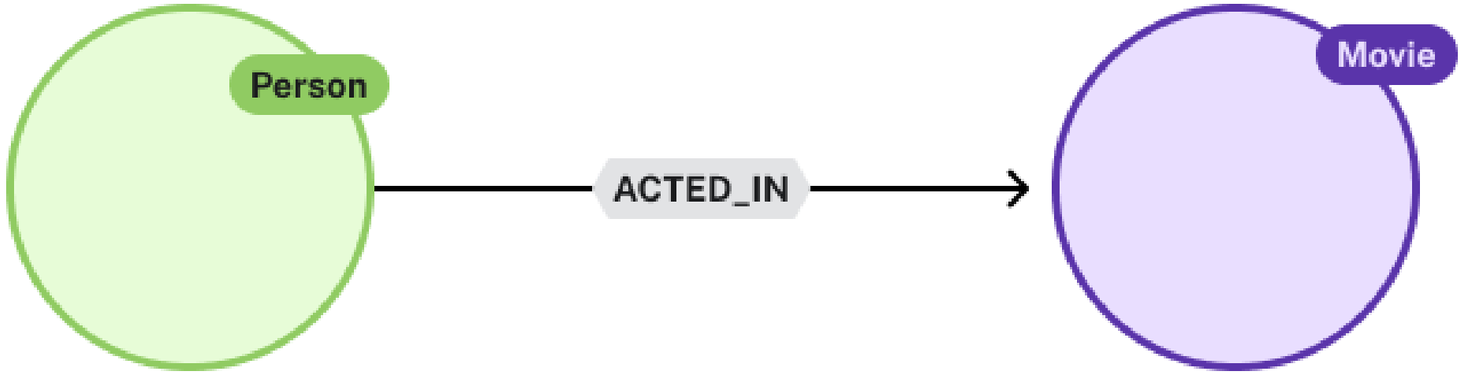
datacamp

Cypher is pattern matching with ASCII-art



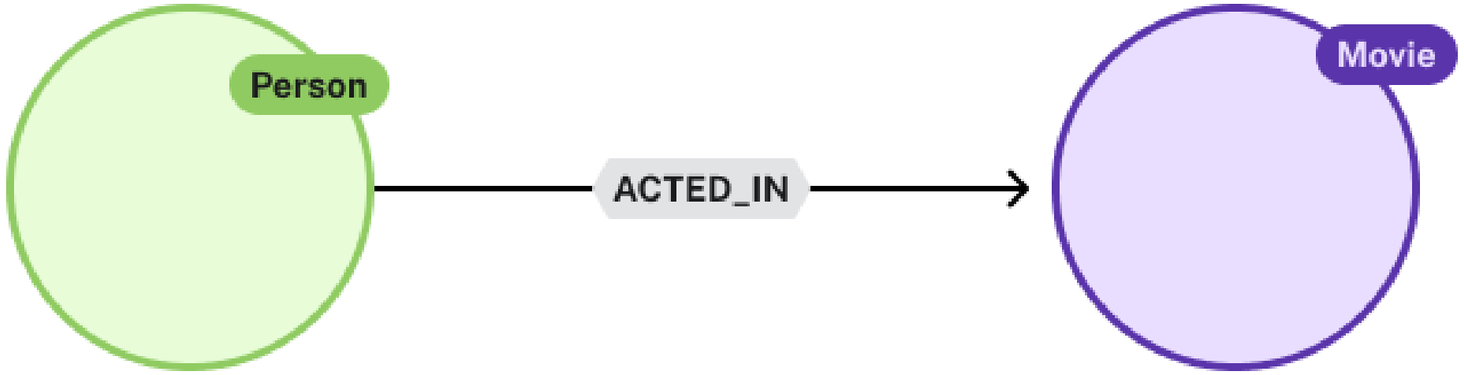

```
MATCH
```

Cypher is pattern matching with ASCII-art



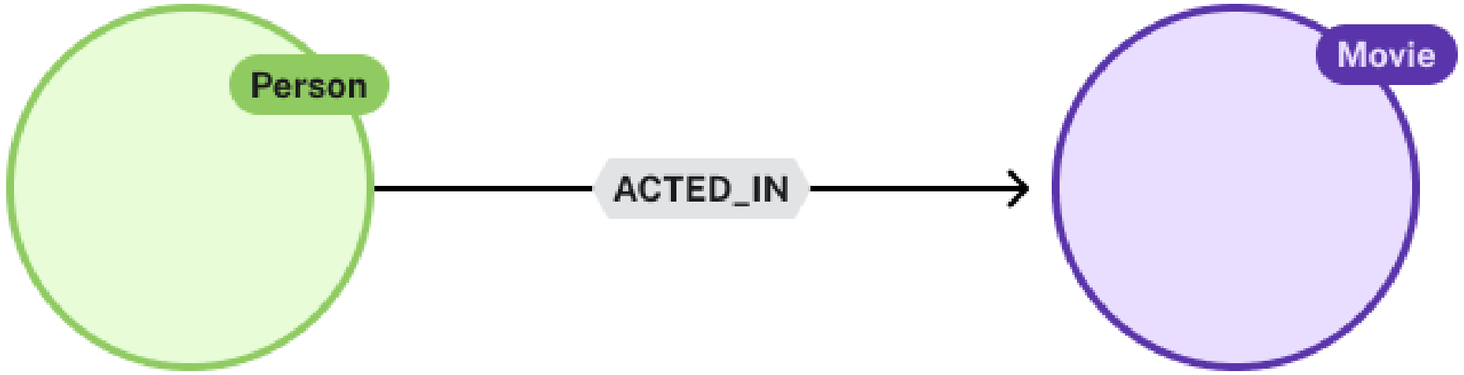

```
MATCH (          )                    (          )
```

Cypher is pattern matching with ASCII-art



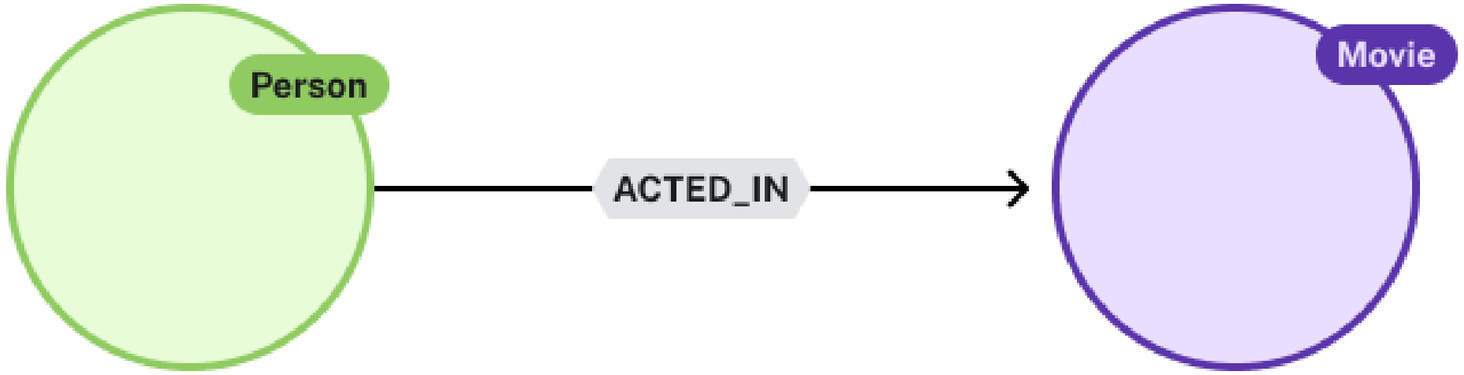

```
MATCH ( :Person)                    ( :Movie)
```

Cypher is pattern matching with ASCII-art



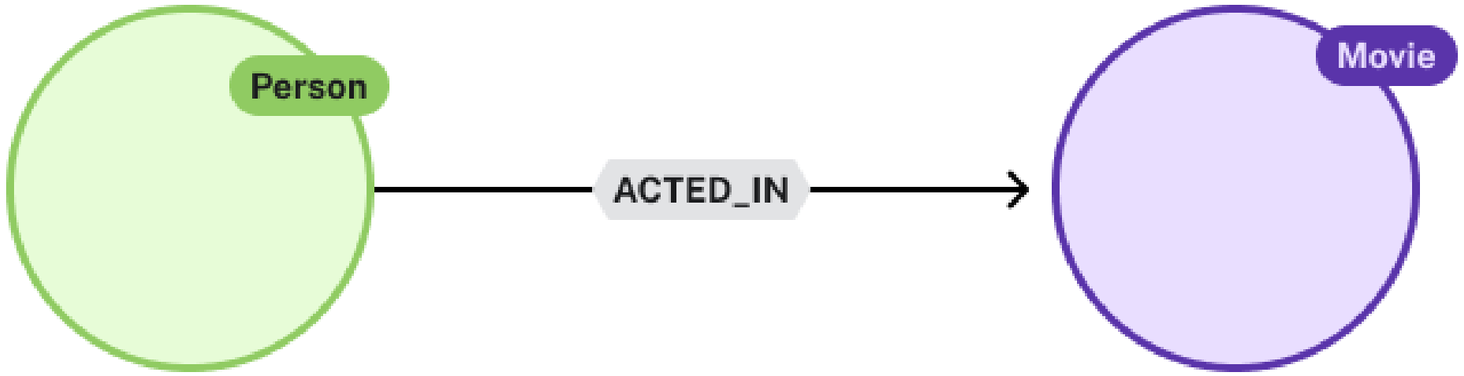

```
MATCH ( :Person)-                ->( :Movie)
```

Cypher is pattern matching with ASCII-art



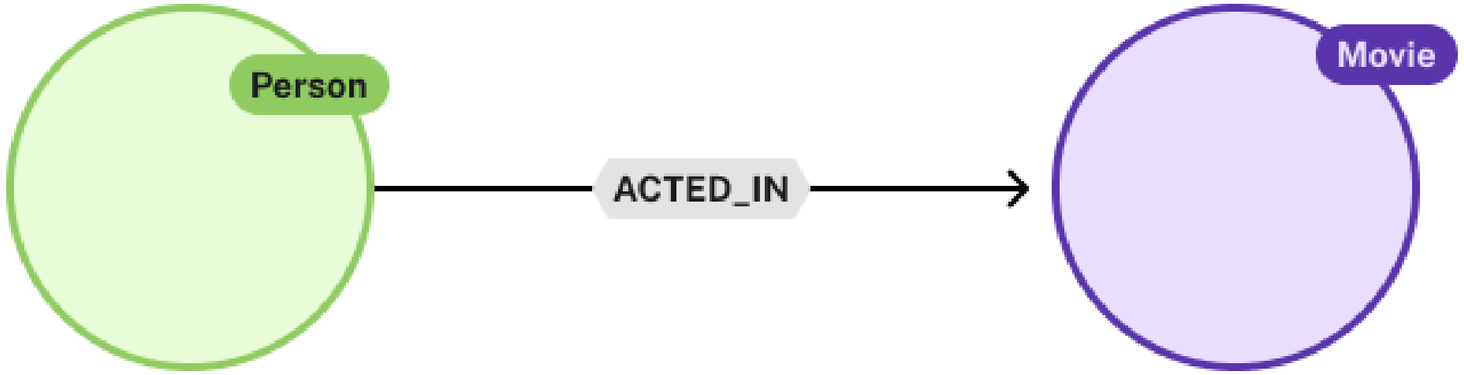

```
MATCH ( :Person)-[ :ACTED_IN]->( :Movie)
```

Cypher is pattern matching with ASCII-art



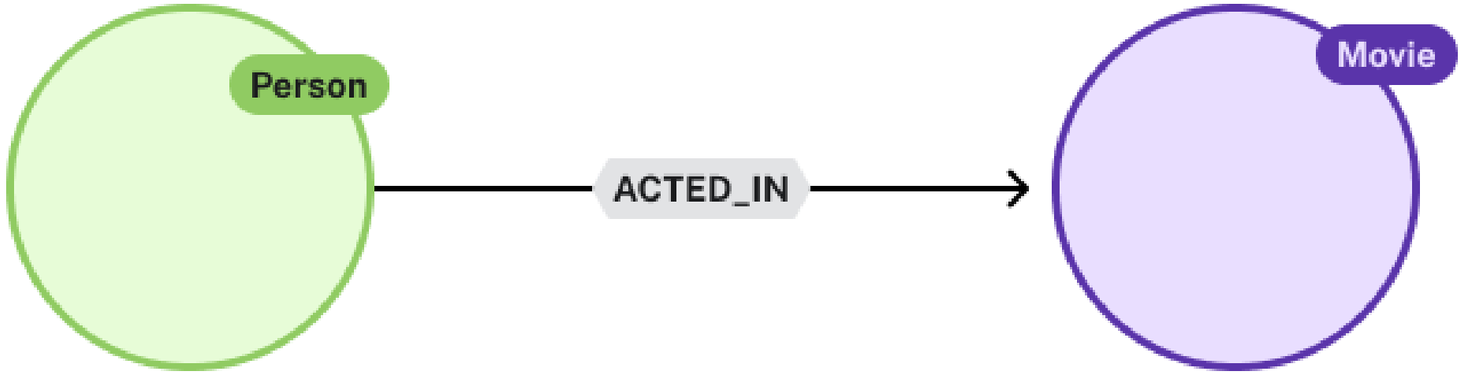

```
MATCH (a:Person)-[r:ACTED_IN]->(m:Movie)
```

Filtering results



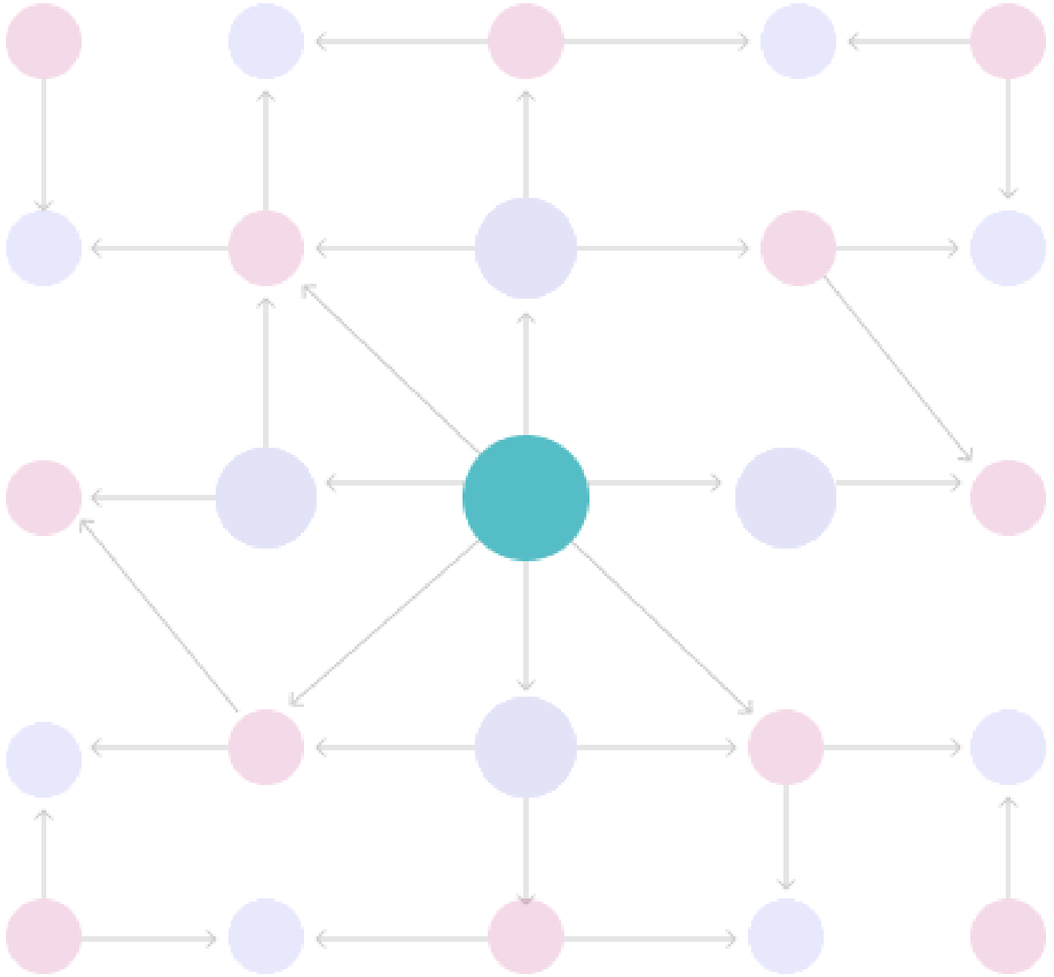

```
MATCH (a:Person)-[r:ACTED_IN]->(m:Movie)
WHERE a.name = 'Tom Hanks'
```

Controlling outputs




```
MATCH (a:Person)-[r:ACTED_IN]->(m:Movie)
WHERE a.name = 'Tom Hanks'
RETURN a.name, m.title AS movieTitle, r.roles AS roles
```

How Cypher works



The CREATE clause

The `CREATE` clause creates a pattern in the graph

```
CREATE (p:Person {name: "Adam"})
CREATE (c:Company {name: "Neo4j"})
CREATE (p)-[:WORKS_FOR]->(c)
```

The MERGE Clause

The `MERGE` clause will find *or create* a pattern in the graph

```
MERGE (p:Person {name: "Adam"})
MERGE (c:Company {name: "Neo4j"})
```

Cypher statements in LangChain

```python
from langchain_neo4j import Neo4jGraph


graph = Neo4jGraph(url=NEO4J_URI, username=NEO4J_USERNAME, password=NEO4J_PASSWORD)
```

```python
# Execute a Cypher statement
graph.query(
    "MATCH (p:Person {name: $name}) RETURN p",
    {"name": "Your name"}
)
```

[1] Cypher injection: https://neo4j.com/developer/kb/protecting-against-cypher-injection/

Let's practice!

GRAPH RAG WITH LANGCHAIN AND NEO4J

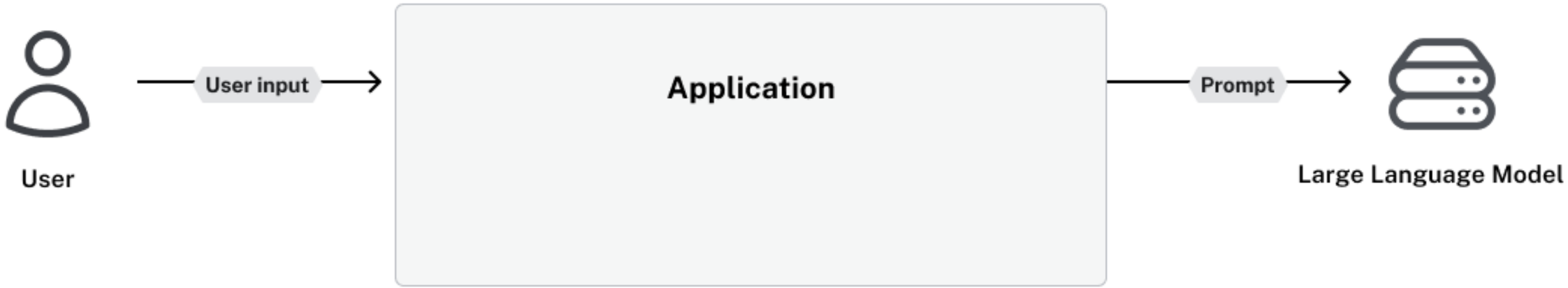
Text-to-Cypher Graph RAG with Neo4j

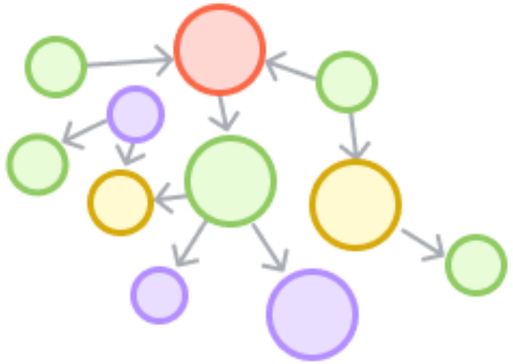
GRAPH RAG WITH LANGCHAIN AND NEO4J

Adam Cowley
Manager, Developer Education at Neo4j

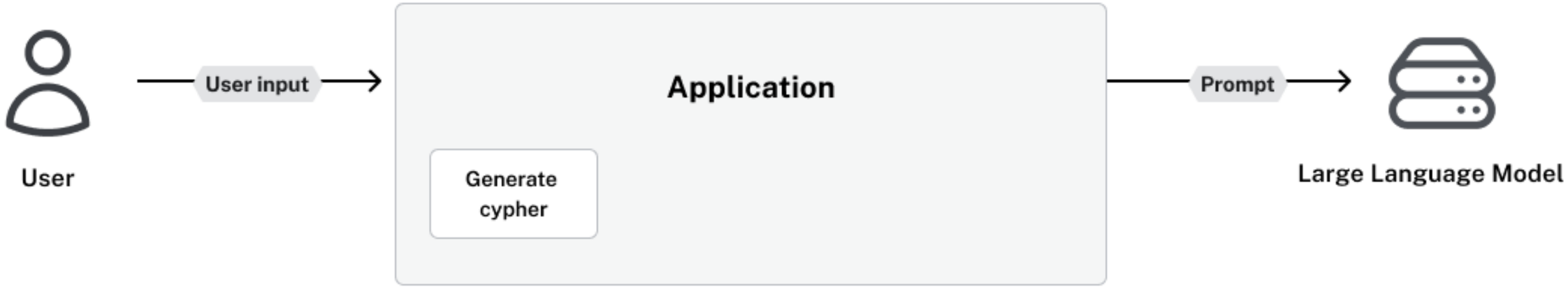
datacamp

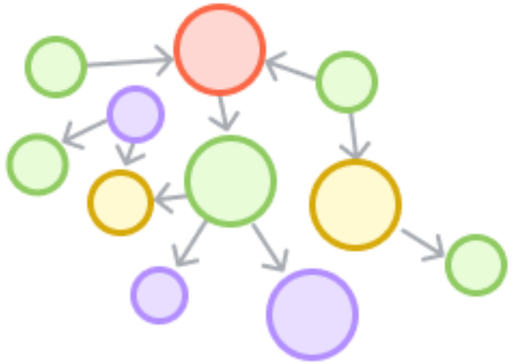
Converting natural language to Cypher



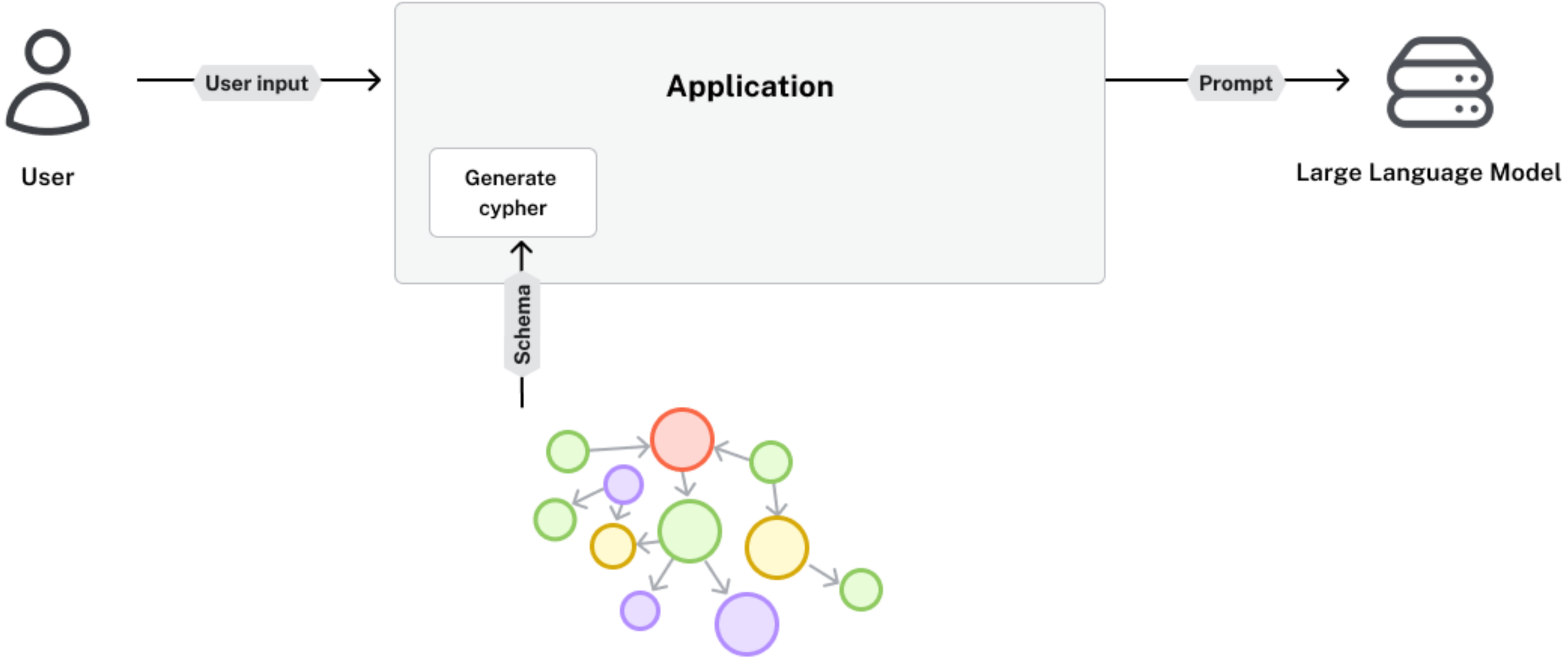


Converting natural language to Cypher

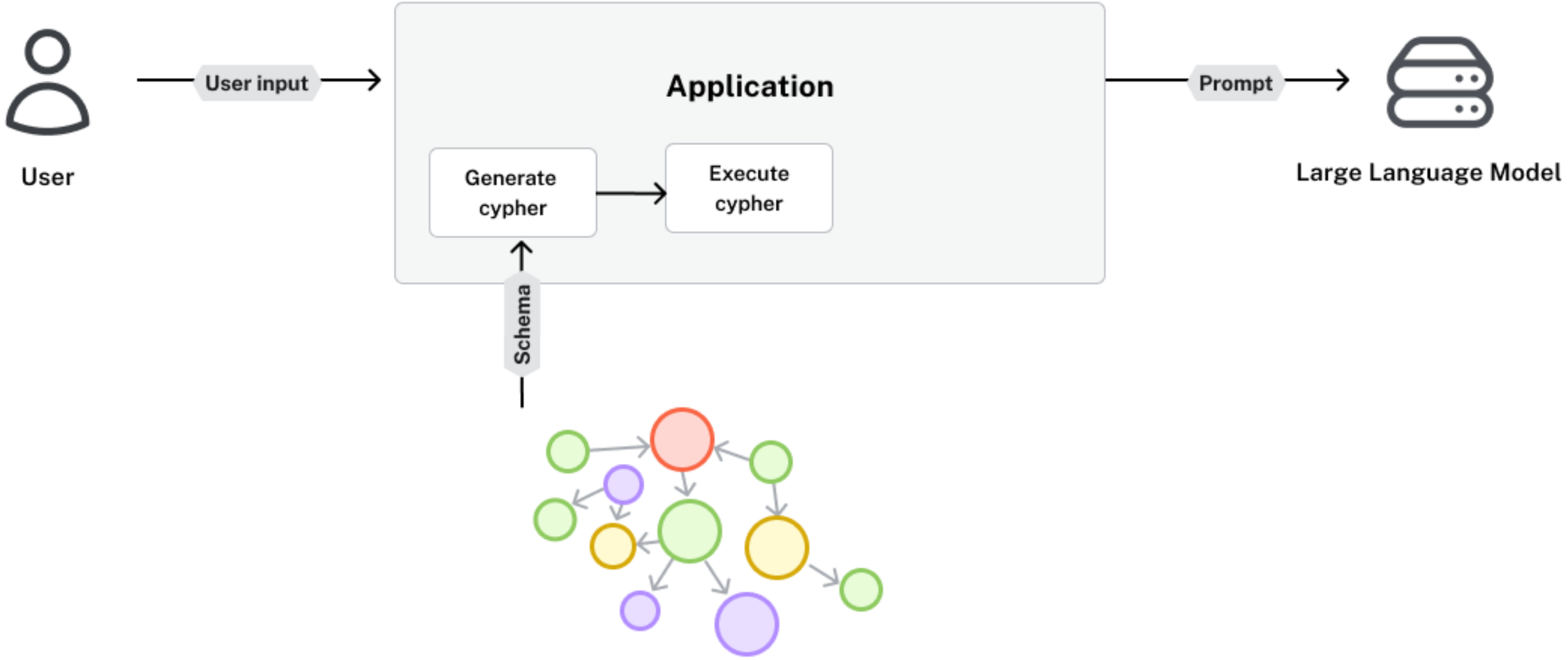




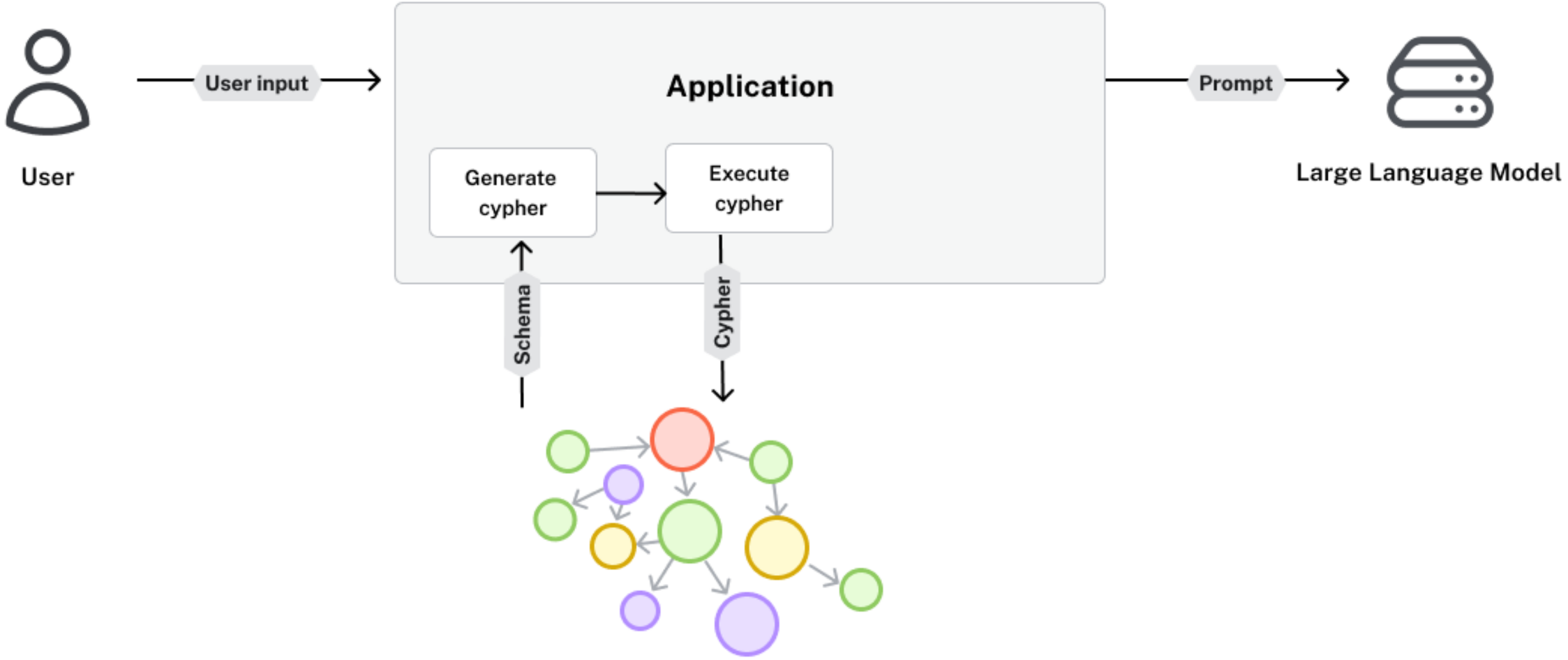
Converting natural language to Cypher



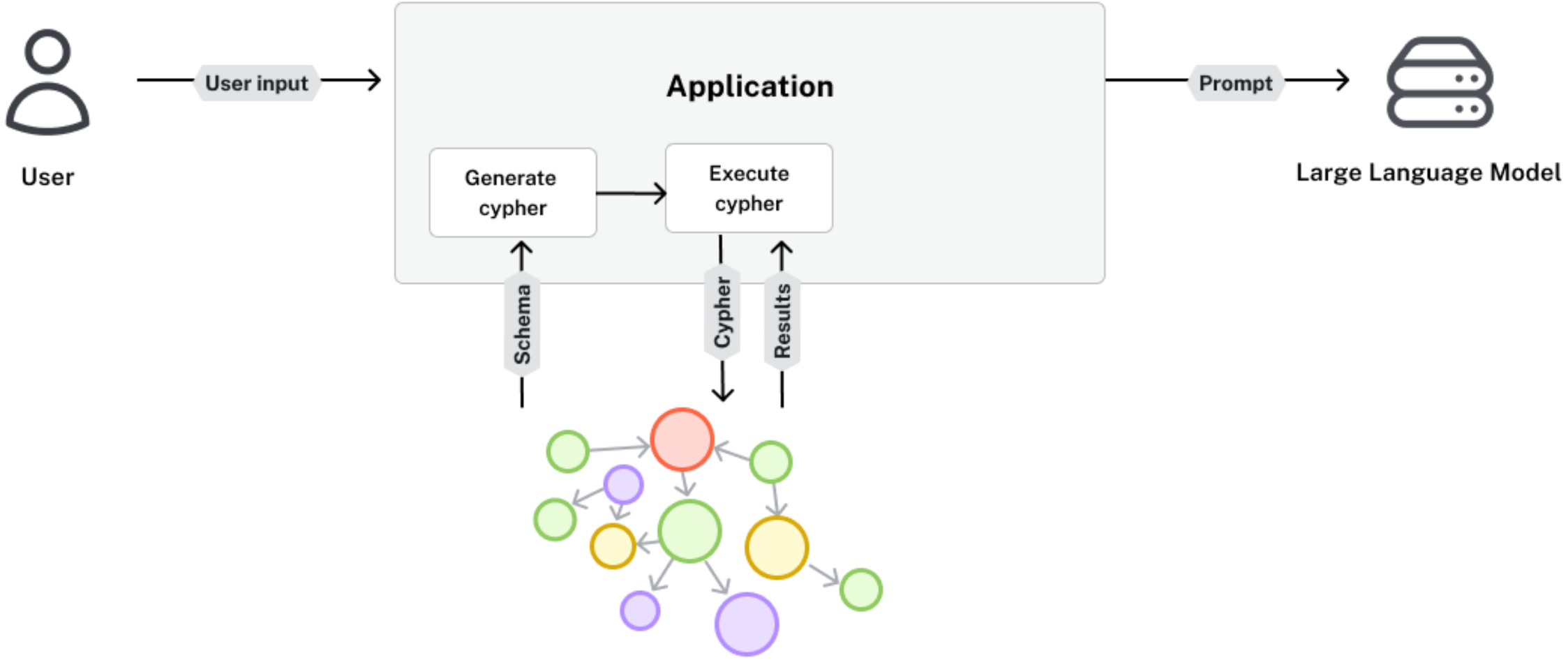
Converting natural language to Cypher



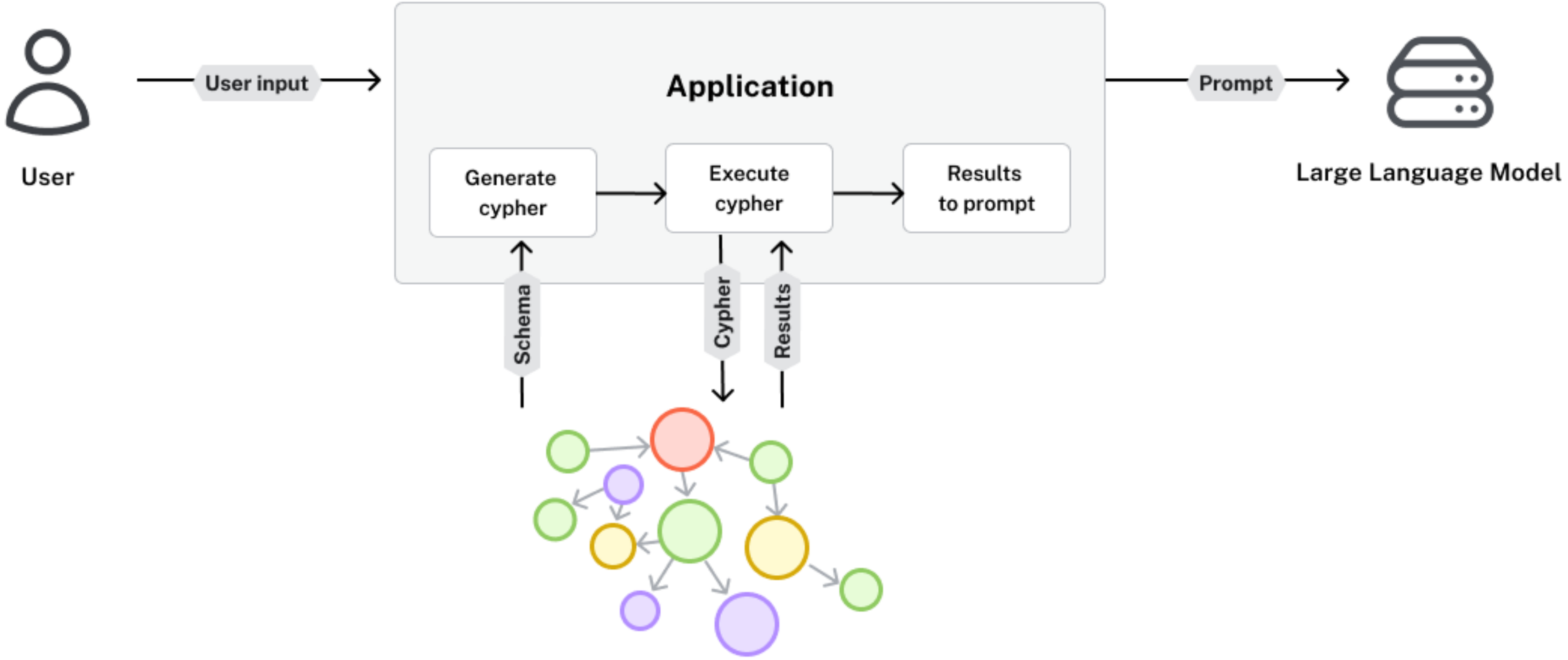
Converting natural language to Cypher



Converting natural language to Cypher



Converting natural language to Cypher



Building a text-to-Cypher chain

```
prompt = SystemPromptTemplate.from_template("""
You are an expert Neo4j developer. Use the following database schema to write
a Cypher statement to answer the user's question...

Schema: {schema}
Question: {question}
""", partial_variables={"schema": graph.schema})

# Compile the chain
cypher_chain = cypher_prompt | llm | StrOutputParser()

# Invoke the chain
cypher_chain.invoke({"question": "Who wrote Building Knowledge Graphs?"})
```

```
MATCH (b:Book {title: "Building Knowledge Graphs"})-[:WRITTEN_BY]->(a:Author) ...
```

```
QA_PROMPT = "You are a helpful assistant. Use the data retrieved from the graph to answer the user's question. Data: {context}"

answer_prompt = ChatPromptTemplate.from_messages([
    SystemMessagePromptTemplate.from_template(QA_PROMPT),
    HumanMessagePromptTemplate.from_template("Question: {question}")
])
```

```
cypher_qa_chain = (
  # Generate and execute the Cypher to get results
  {
    "question": RunnablePassthrough()
    "context": text_to_cypher_chain | RunnableLambda(lambda cypher: graph.query(cypher)),
  }
  # Format prompt and pass to LLM
  | answer_prompt | answer_llm | StrOutputParser()
)

res = cypher_qa_chain.invoke({"question": "Who wrote Building Knowledge Graphs?"})
```

```
Building Knowledge Graphs was written by Dr Jesus Barrasa and Dr Jim Webber...
```

Let's practice!

GRAPH RAG WITH LANGCHAIN AND NEO4J