Data Umbrella

Knowledge Graph-Based Chatbot

May 7, 2024 16:00 UTC

9 AM PT • 12 PM ET • 7 PM EAT • 9:30 PM IST

LIVE WEBINAR

Alison Cossette
Developer Advocate,
Neo4j

Today we will

1. Review Retrieval Augmented Generation (RAG)
2. Review the GenAI Stack
3. Cover Neo4j fundamentals
4. Walk through Neo4j parts of the Chatbot GenAI Stack app

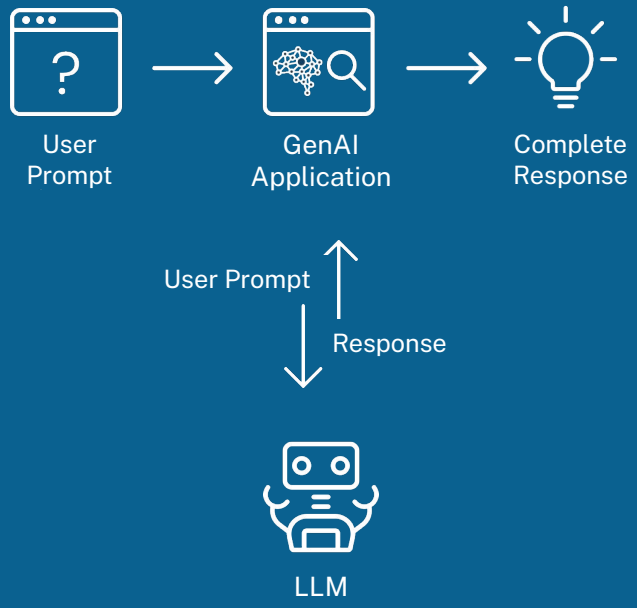
BONUS: Logging LLM interactions

Retrieval Augmented Generation (RAG)

RAG is a software design pattern for integrating GenAI Apps with custom data sources, like a database.

A Generative AI application uses an LLM to provide **responses** to **user prompts**

(aka ChatGPT)

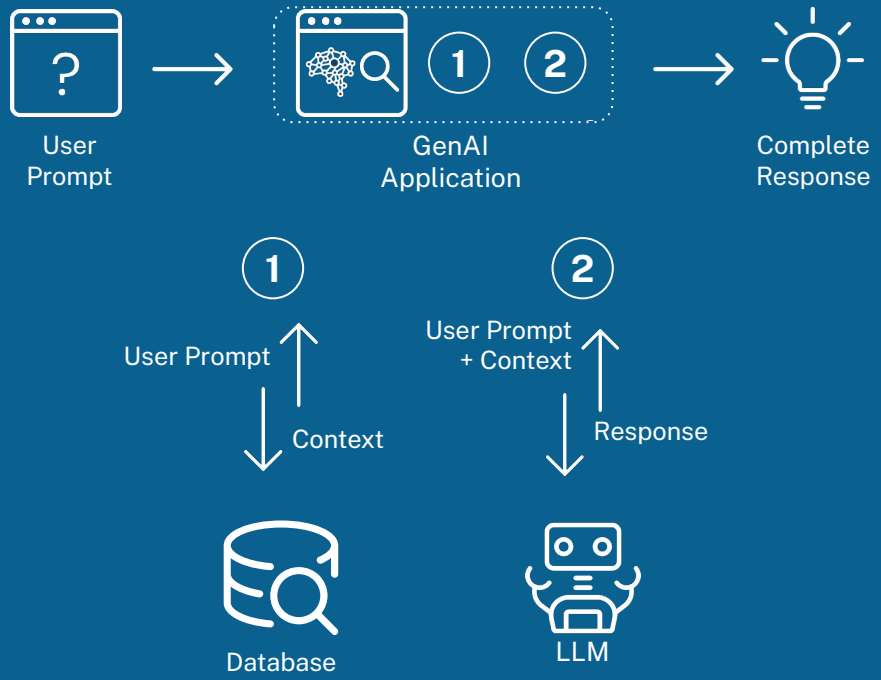


RAG augments the LLM by intercepting a **user's prompt**,

then making a **query to a database**,

then using the query results as **context** for the user's prompt, creating a **new prompt** that is passed to the LLM

for a **complete, curated response**



This sets up a **knowledge stack…**

the **user** knows something about the question they're asking

the **application** knows something about the user

the **database** knows about particular information and data

the **LLM** knows about whatever it found on the internet

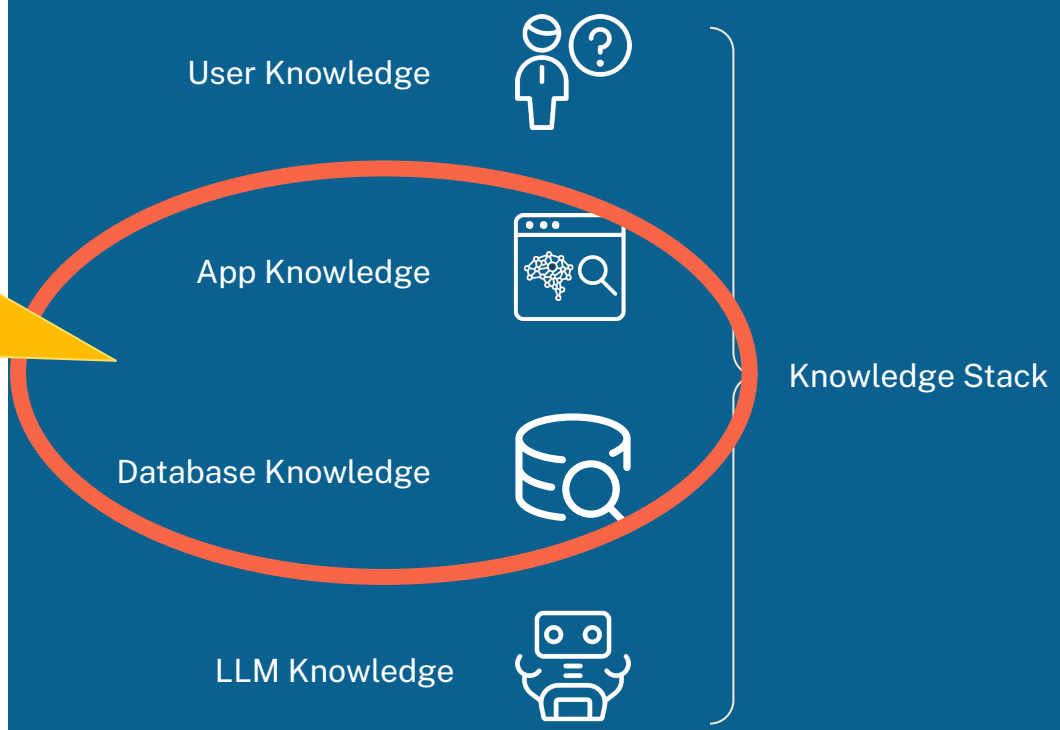
This sets up a **knowledge stack…**

the **user** knows something about the question they're asking

Knowledge you control,

in the app and the database.

the **LLM** knows about whatever it found on the internet

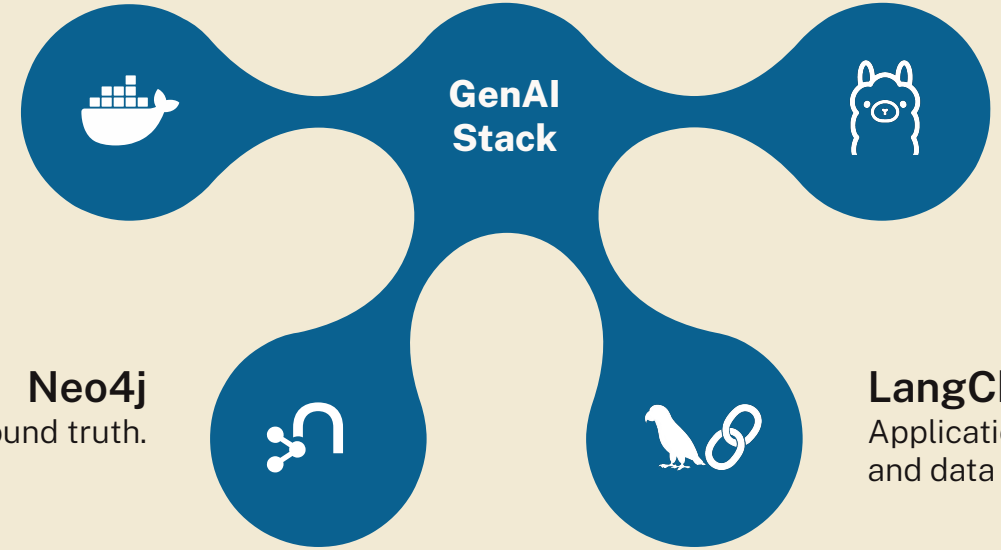


The GenAI Stack

is *the way* to do RAG

The GenAI Stack

Docker
Orchestrate containers.

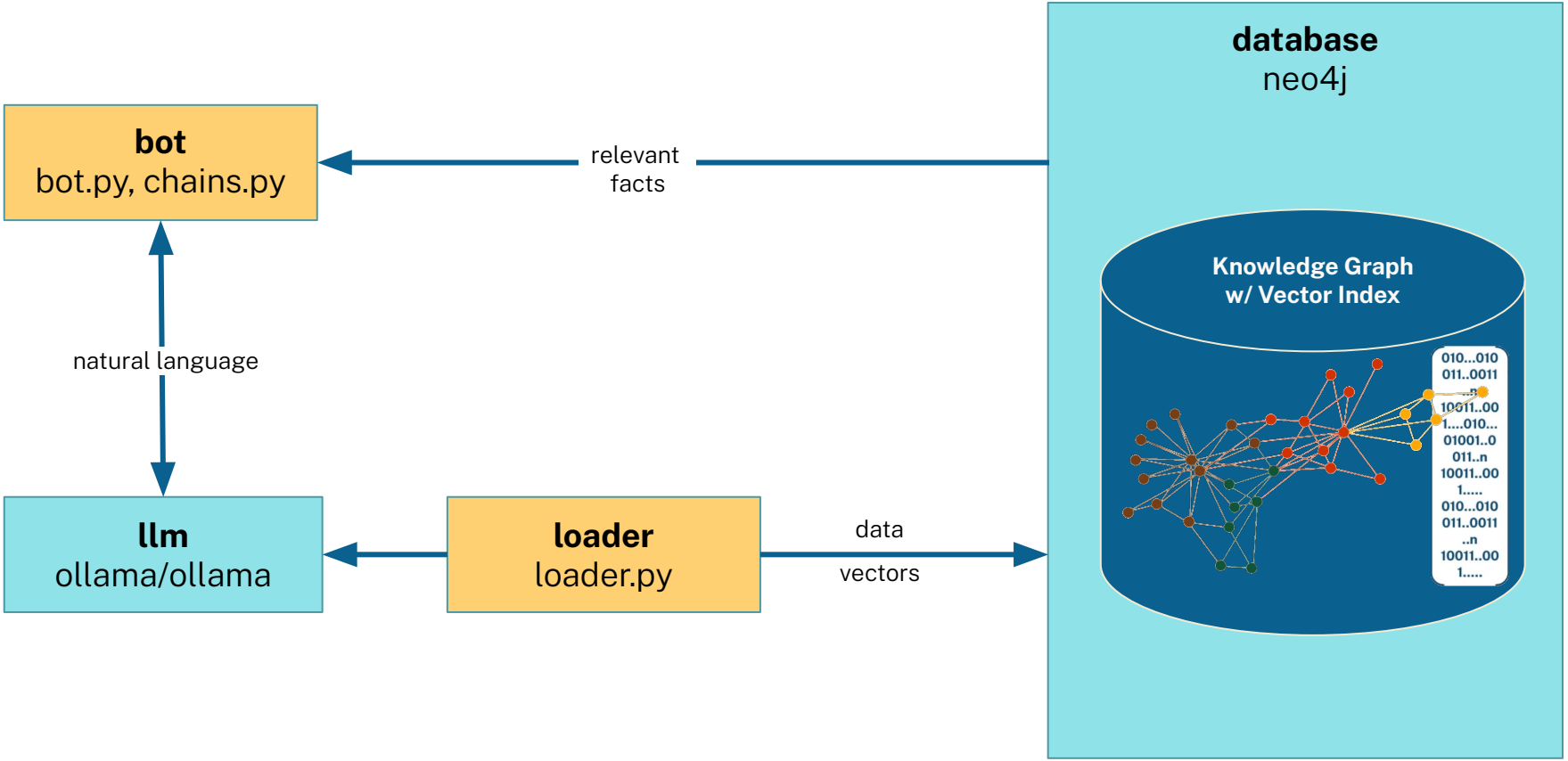


Ollama
Manage local LLMs.

Neo4j
Database for ground truth.

LangChain
Application logic
and data flows.

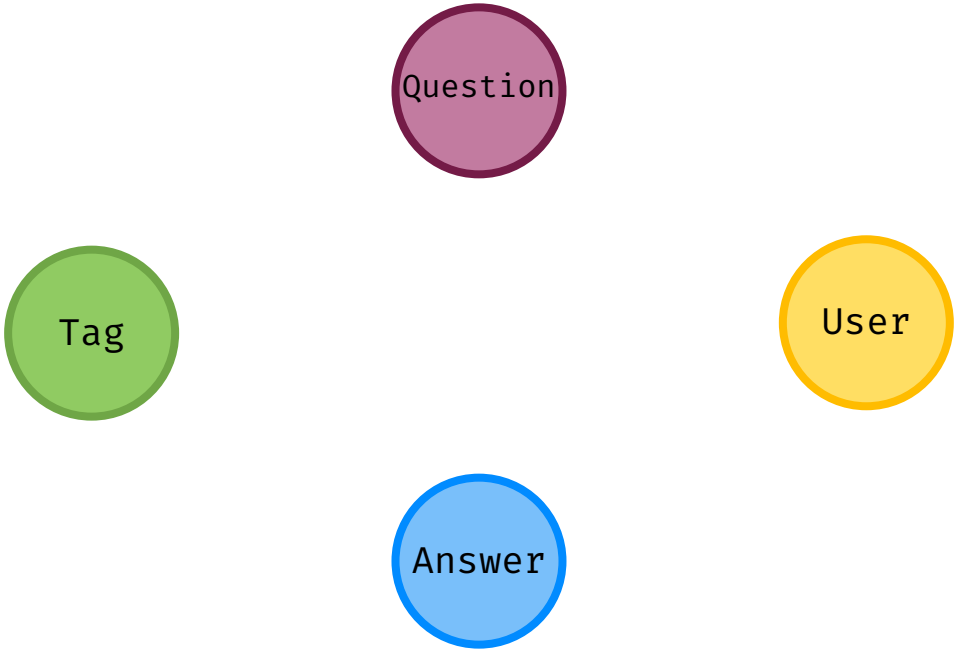
GenAI Stack Containers



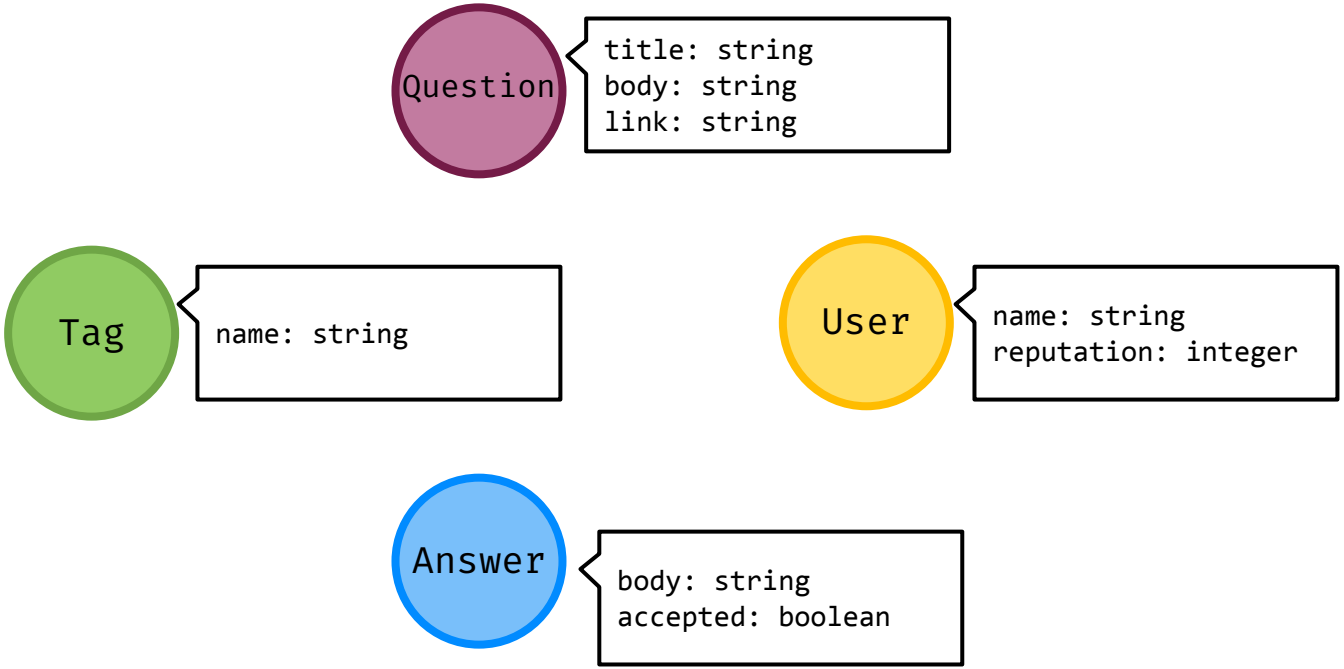
Intro to Neo4j

Providing ground truth for the GenAI Stack

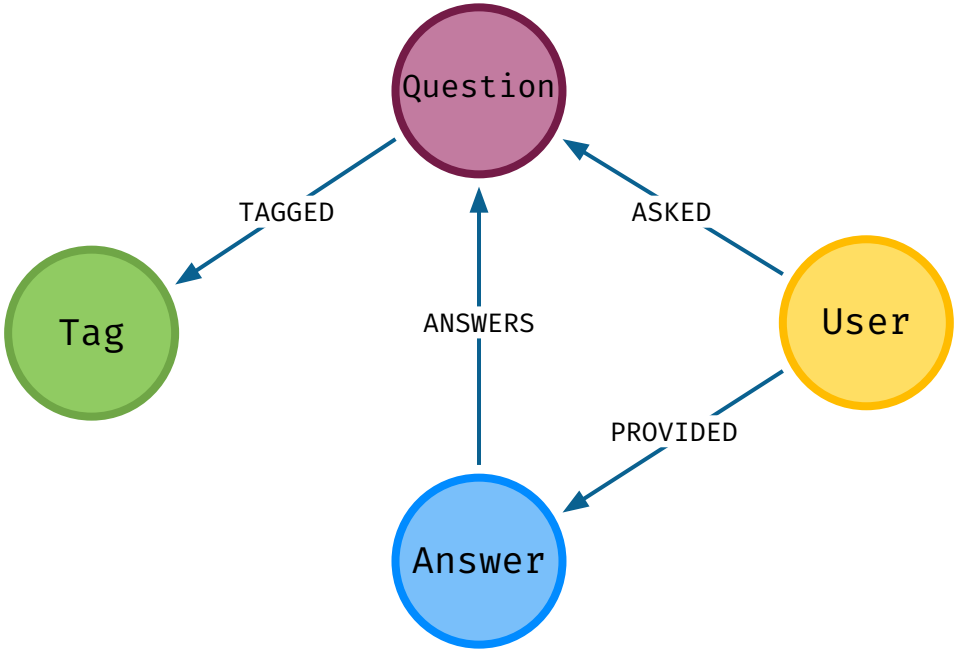
Data starts with things. In the chat bot, those are...



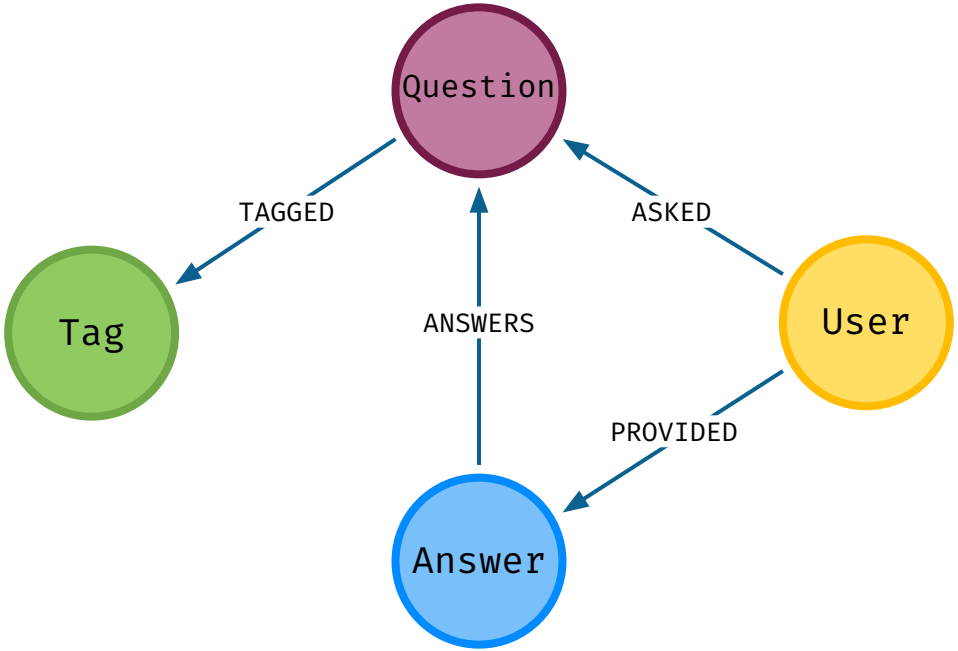
Data records information about things



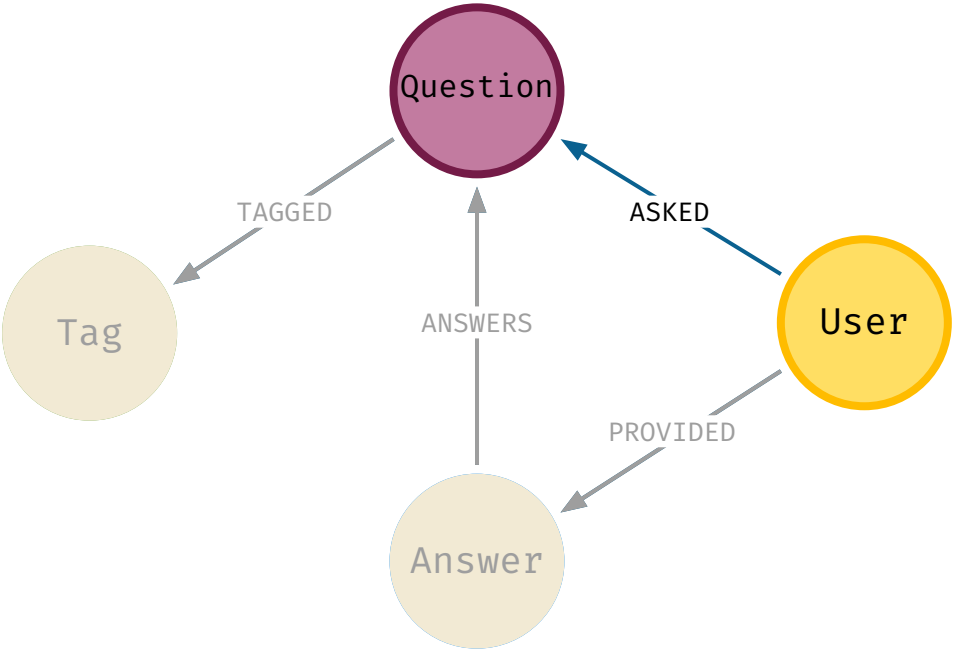
Data records are related



Data relationships create patterns

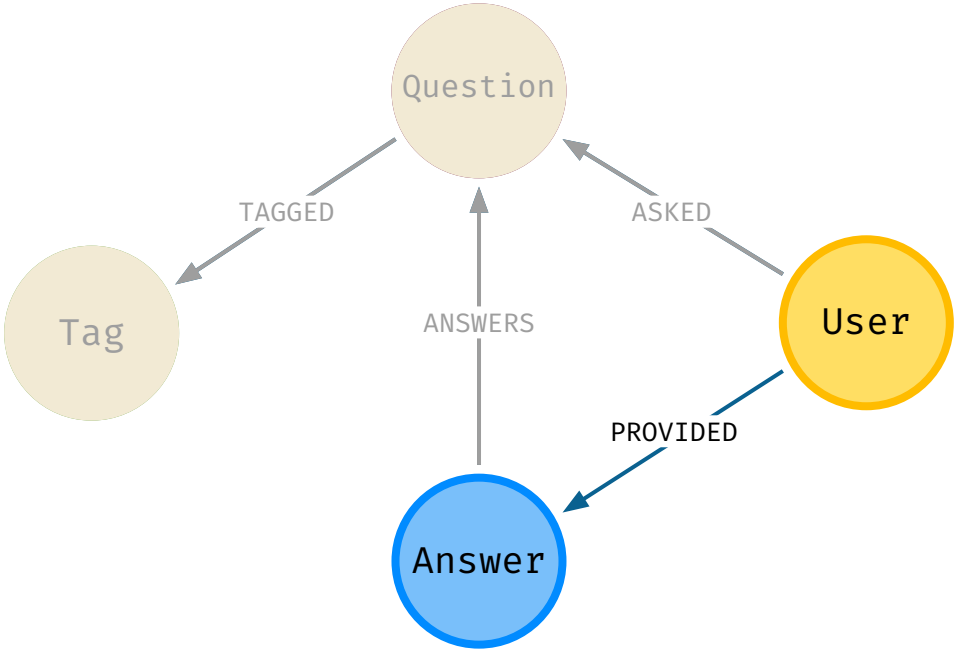


Data pattern: from users to questions



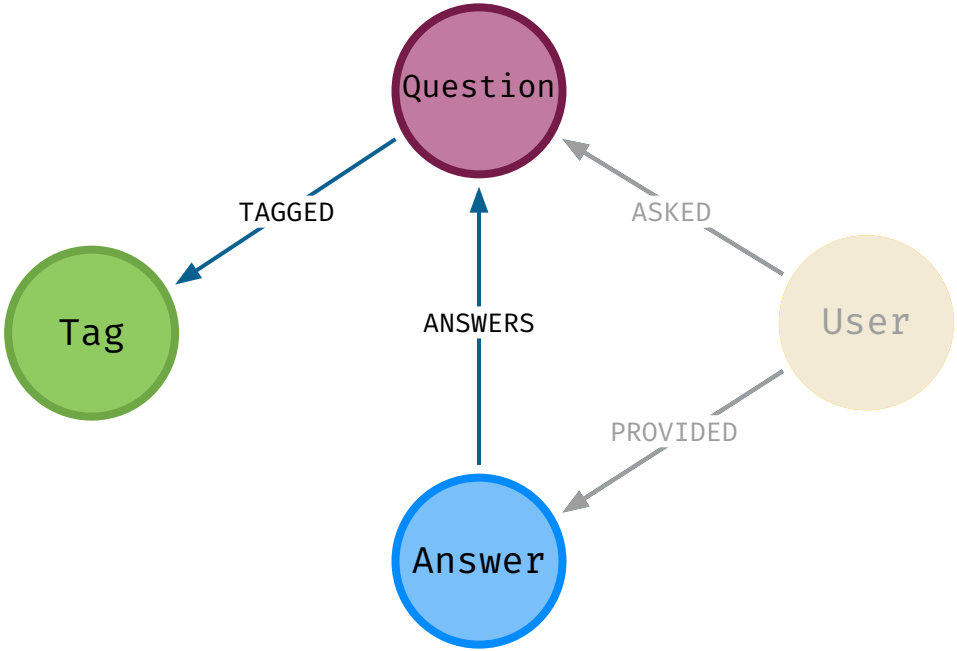

```
(User)-[ASKED]→(Question)
```

Data pattern: from users to answers




```
(User)-[PROVIDED]→(Answer)
```

Data pattern: from answers to tagged questions




```
(Answer)-[ANSWERS]→(Question)-[TAGGED]→(Tag)
```

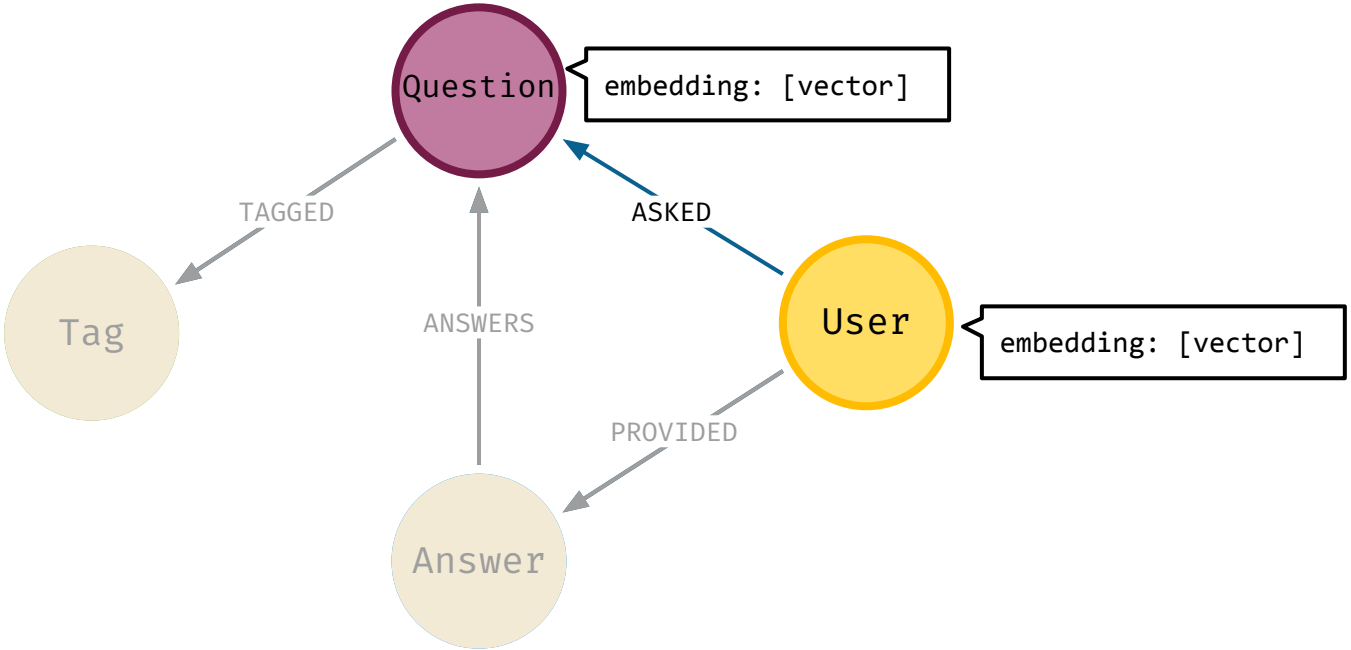
Query using pattern matching, using Cypher in Neo4j

Match a pattern of answers to questions.

Return the Q&A pairs, limited to 10.

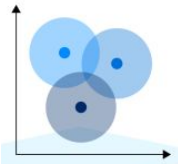
```
MATCH (a:Answer)-[:ANSWERS]→(q:Question)
RETURN q, a LIMIT 10
```

Data may also contain vectors ;)



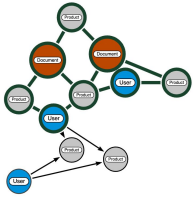
Graph + Vector = Semantic Search

Vector Index



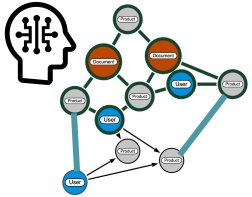
Find similar documents.

Graph Structure



Find related information.

Knowledge Graph



Combine for more accurate results within a relevant context.

Similarity Search

Pattern Matching

neo4j

Summary: Neo4j

What is Neo4j?

- graph database
- for storing a knowledge graph
- indexed with vectors for similarity search

Why? similar data + relevant context = accurate answers

- Initialize: create indexes, including vector
- Load: construct knowledge graph, vectorize documents
- Run: vector lookup, then pattern match

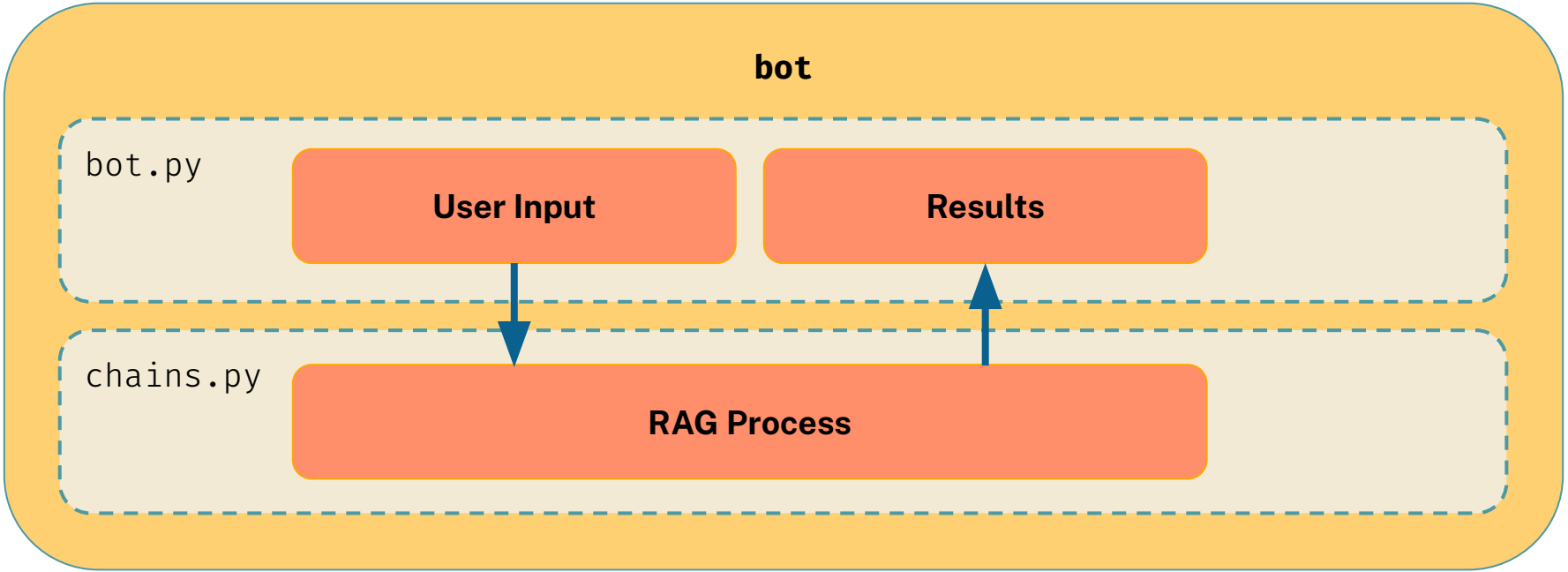
Learn More

- https://graphacademy.neo4j.com

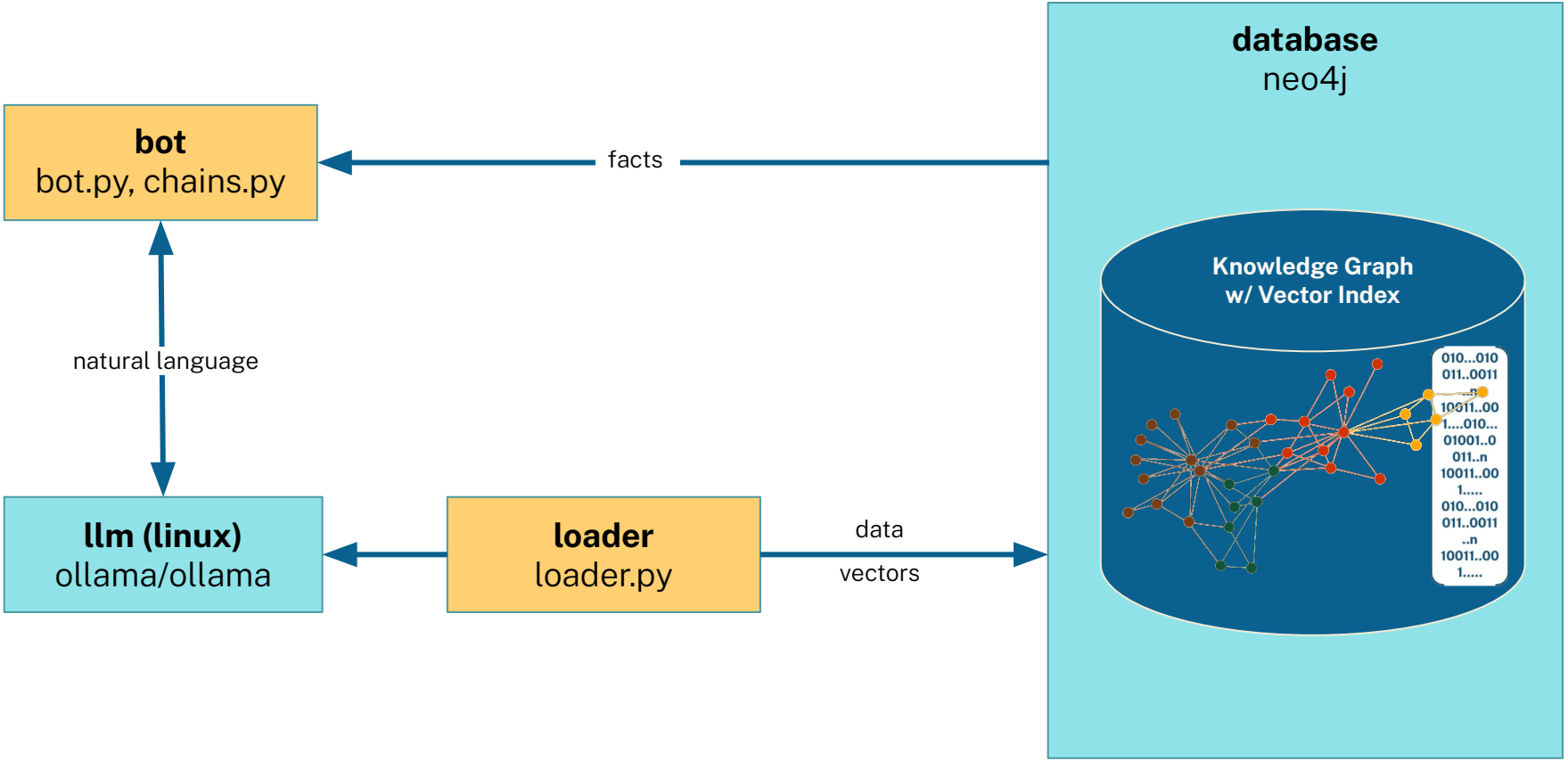
RAG+KG with Neo4j in the GenAI Stack

A closer look at working with Neo4j

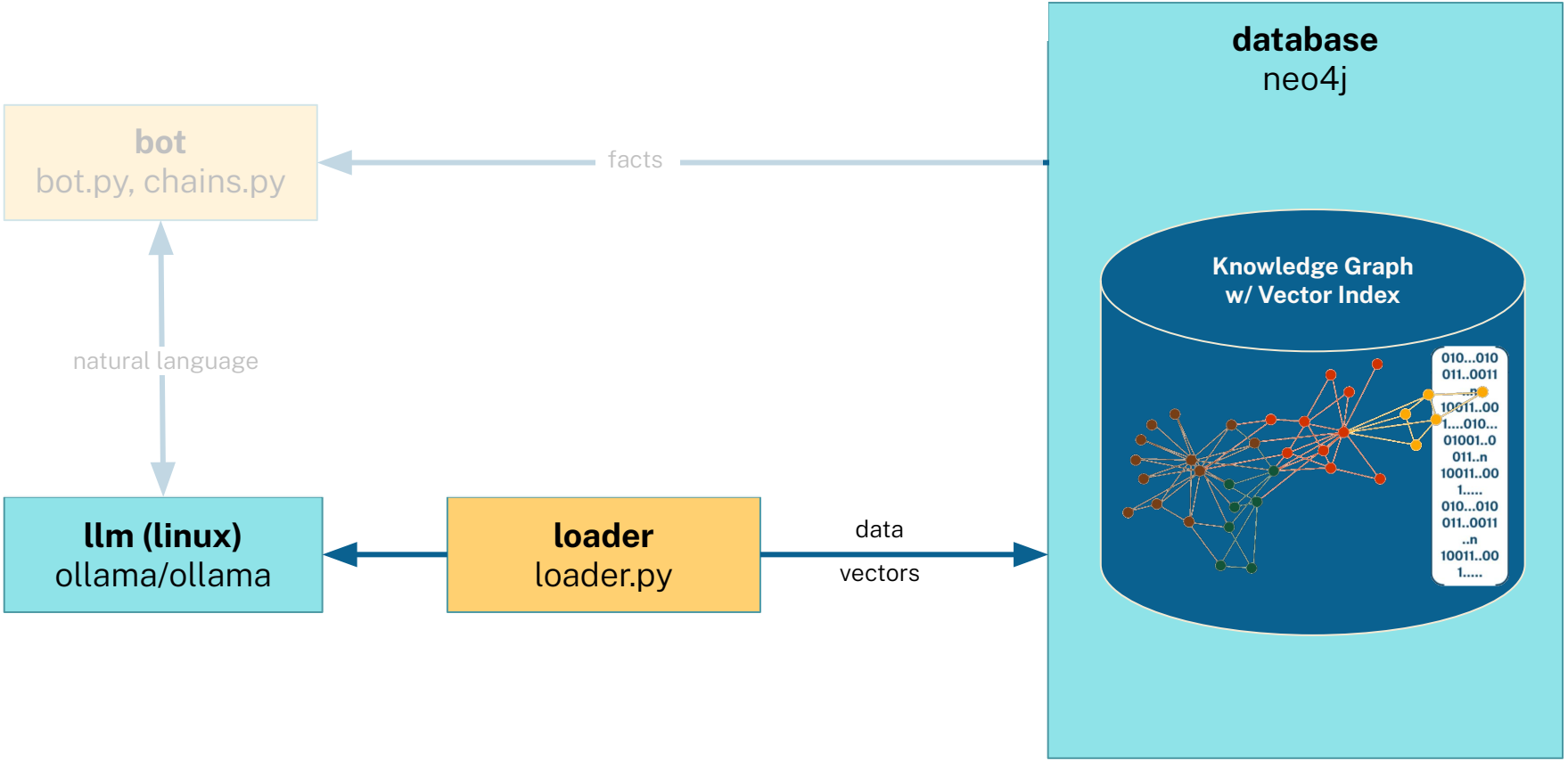
bot - where RAG happens in the GenAI Stack



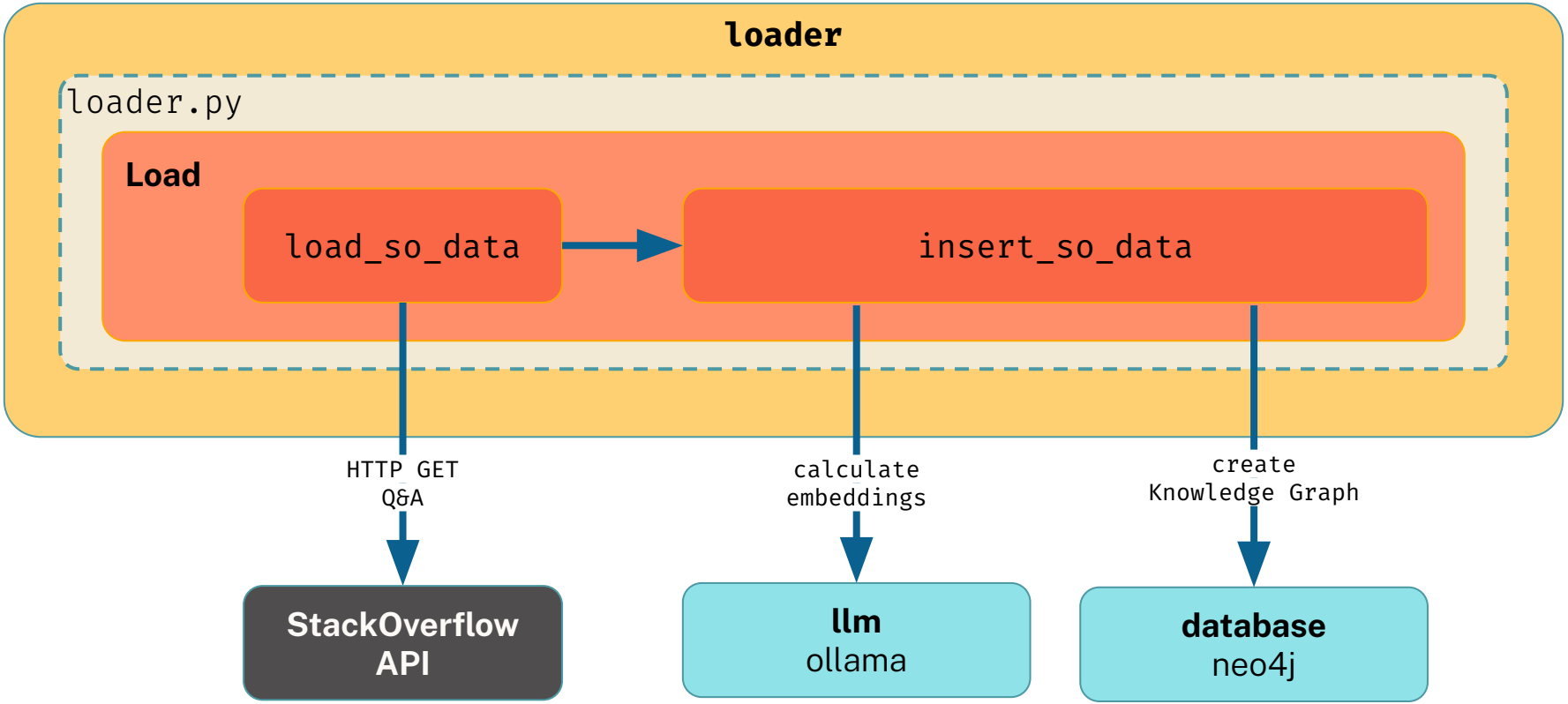
GenAI Stack Overview



loader - StackOverflow import



loader - StackOverflow import



loader - loader.py create questions

Merge a question, identified by a unique id.

Merge is an "upsert", finding an existing record, or creating it if needed.

When found, do nothing else. (this is absent in the query below)

When creating, set values on the record.

```
MERGE (question:Question {id:q.question_id})
  ON CREATE SET question.title = q.title, question.link = q.link,
    question.score = q.score, question.favorite_count = q.favorite_count,
    question.creation_date = datetime({epochSeconds: q.creation_date}),
    question.body = q.body_markdown, question.embedding = q.embedding
```

loader - loader.py tag each question

For each tag a question has,
merge a tag node (match if it exists, create if it doesn't),
then merge a relationship between the question and the tag.

```
FOREACH (tagName IN q.tags |
  MERGE (tag:Tag {name:tagName})
  MERGE (question)-[:TAGGED]→(tag)
)
```

loader - loader.py attach answers to questions

For each answer to a question, merge the answer and set values, then merge users who provided the answer.

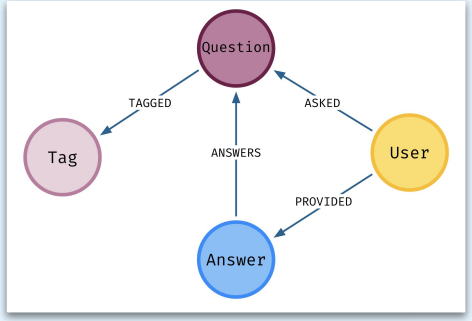
```
FOREACH (a IN q.answers |
  MERGE (question)<-[:ANSWERS]-(answer:Answer {id:a.answer_id})
  SET answer.is_accepted = a.is_accepted,
    answer.score = a.score,
    answer.creation_date = datetime({epochSeconds:a.creation_date}),
    answer.body = a.body_markdown, answer.embedding = a.embedding
  MERGE (answerer:User {id:coalesce(a.owner.user_id, "deleted")})
    ON CREATE SET answerer.display_name = a.owner.display_name,
      answerer.reputation= a.owner.reputation
  MERGE (answer)<-[:PROVIDED]-(answerer)
)
```

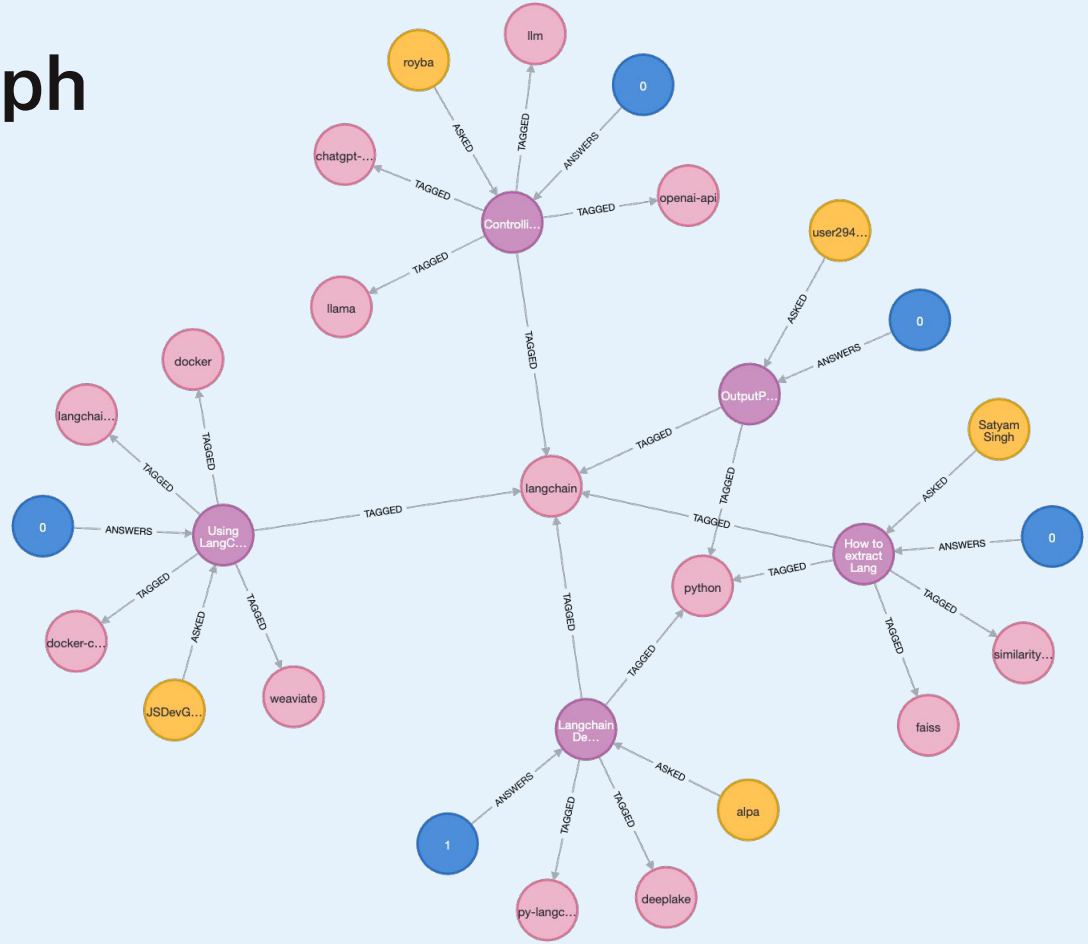
loader.py - owners asked questions

For every question where the owner exists,
merge a User node for the owner
and relate them to the question they ASKED.

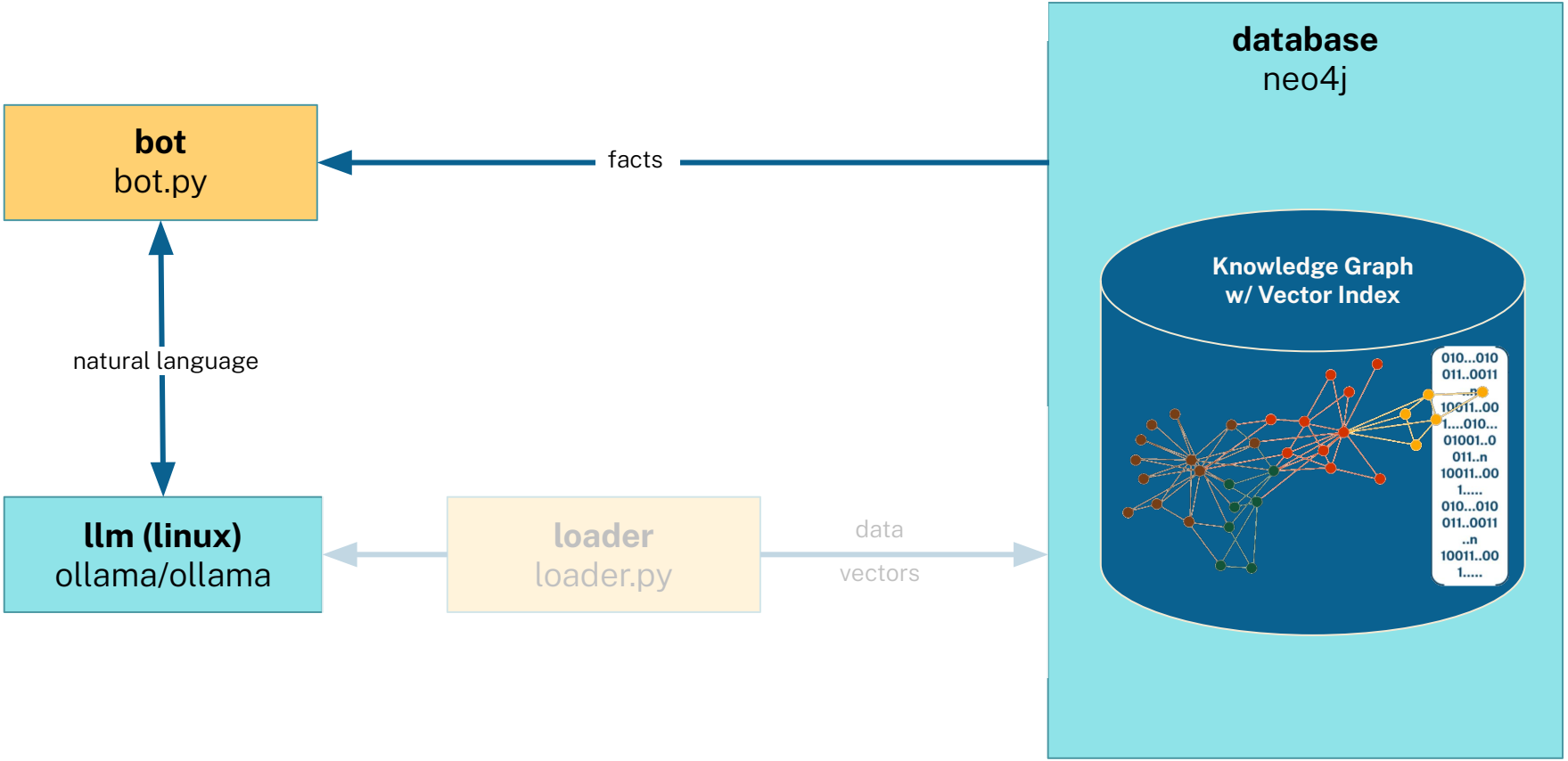
```
WITH * WHERE NOT q.owner.user_id IS NULL
MERGE (owner:User {id:q.owner.user_id})
ON CREATE SET owner.display_name = q.owner.display_name,
    owner.reputation = q.owner.reputation
MERGE (owner)-[:ASKED]→(question)
```

StackOverflow Knowledge Graph

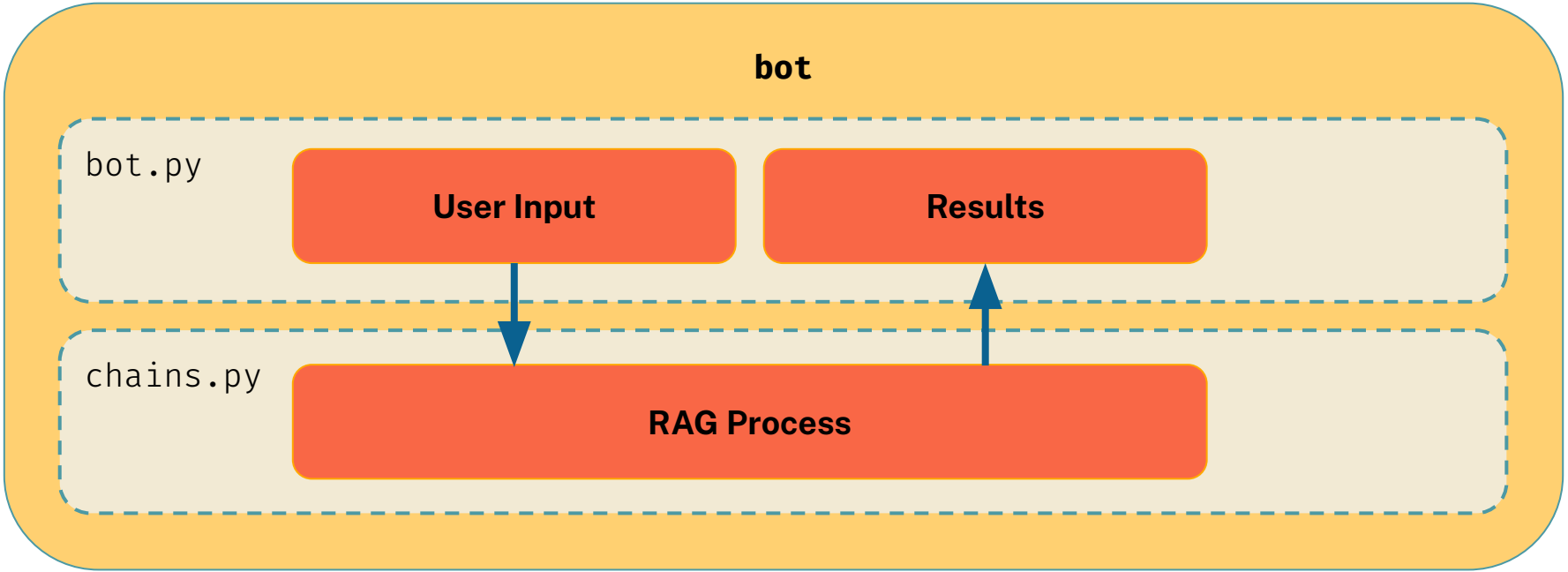




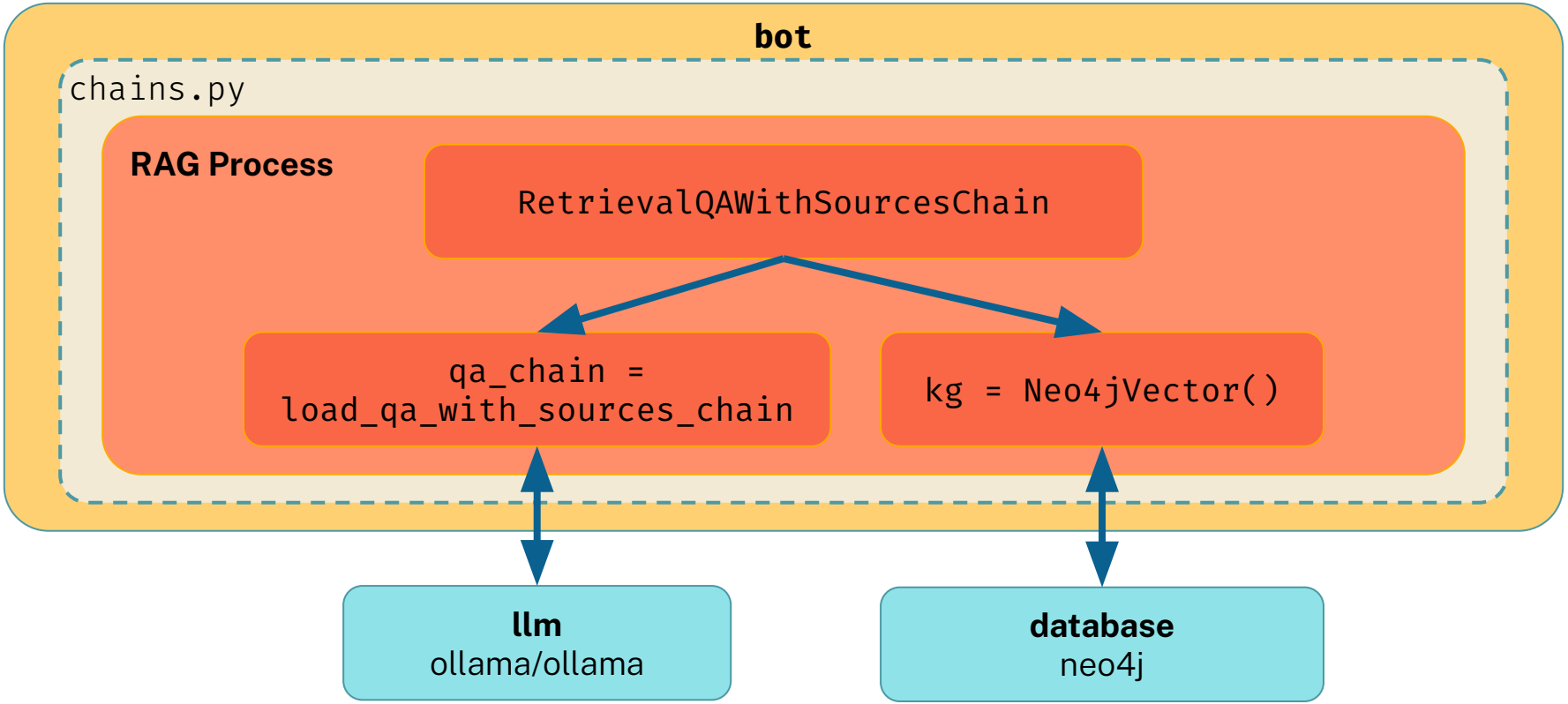
bot - RAG+KG with LangChain



bot - RAG refresher



bot - details of langchain in chains.py



bot - Neo4jVector from LangChain

- integrates Neo4j into LangChain chains
- vector search
  - approximate nearest neighbor search
  - Euclidean similarity and cosine similarity
- graph search
  - provide retrieval query for fetching related, relevant texts from the knowledge graph

bot - chains.py graph search for improved answers

For every similar question from vector search, along with the score, match answers, taking only 2.

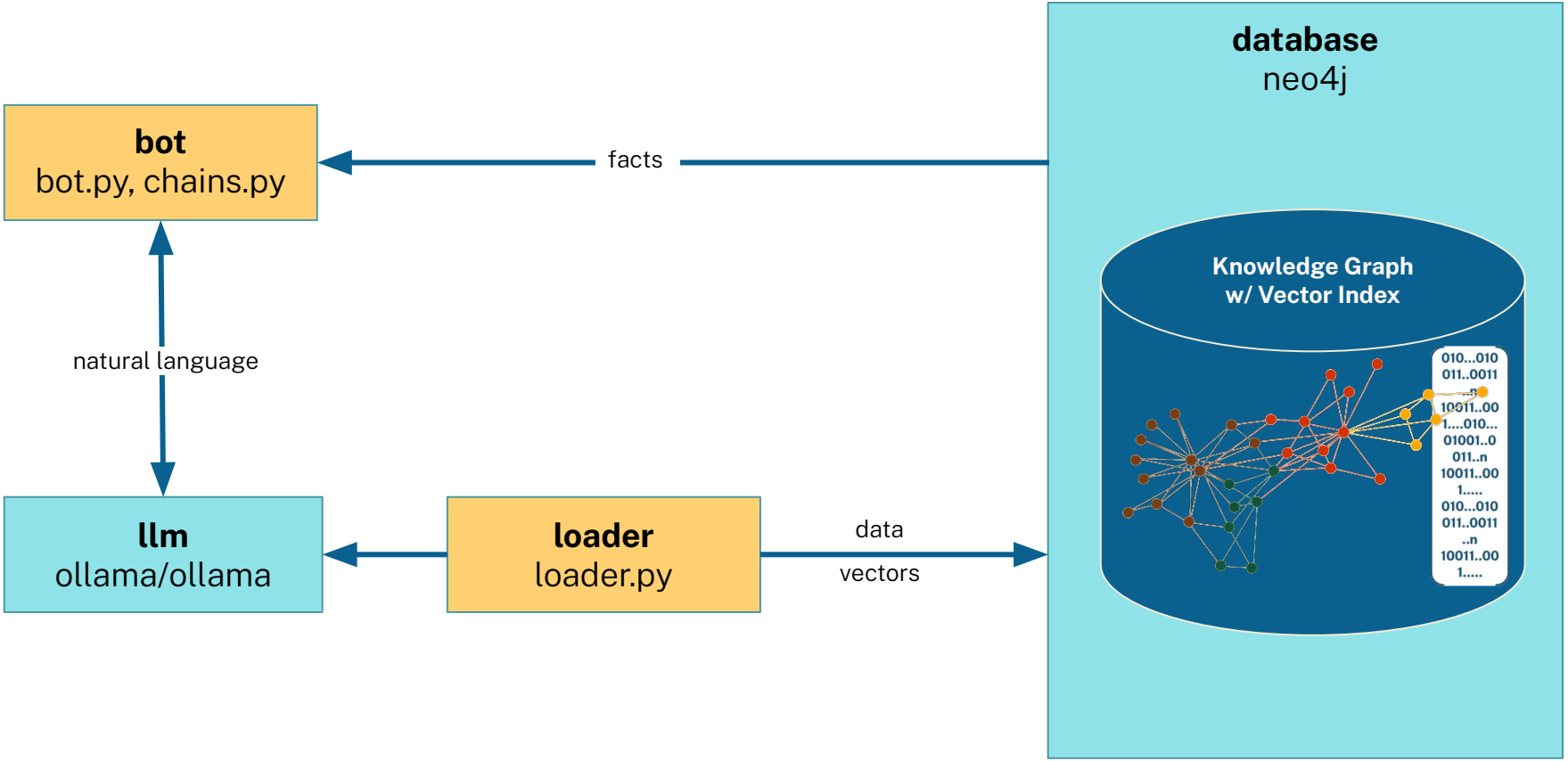
```
WITH node AS question, score AS similarity // from vector search
CALL { with question
  MATCH (answer)-[:ANSWERS]→(question)
  RETURN answer LIMIT 2
}
RETURN question.title + '\n' + question.body + '\n' + collect(answer.body) AS text,
  similarity as score, {source: question.link} AS metadata
```

chains.py - LLM magic with LangChain

Pass the system template, user messages, knowledge graph retriever results all over to the LLM using LangChain.

```
kg_qa = RetrievalQAWithSourcesChain(
  combine_documents_chain=qa_chain,
  retriever=kg.as_retriever(search_kwargs={"k": 2}),
  reduce_k_below_max_tokens=False,
  max_tokens_limit=3375,
)
```

GenAI Stack #FTW



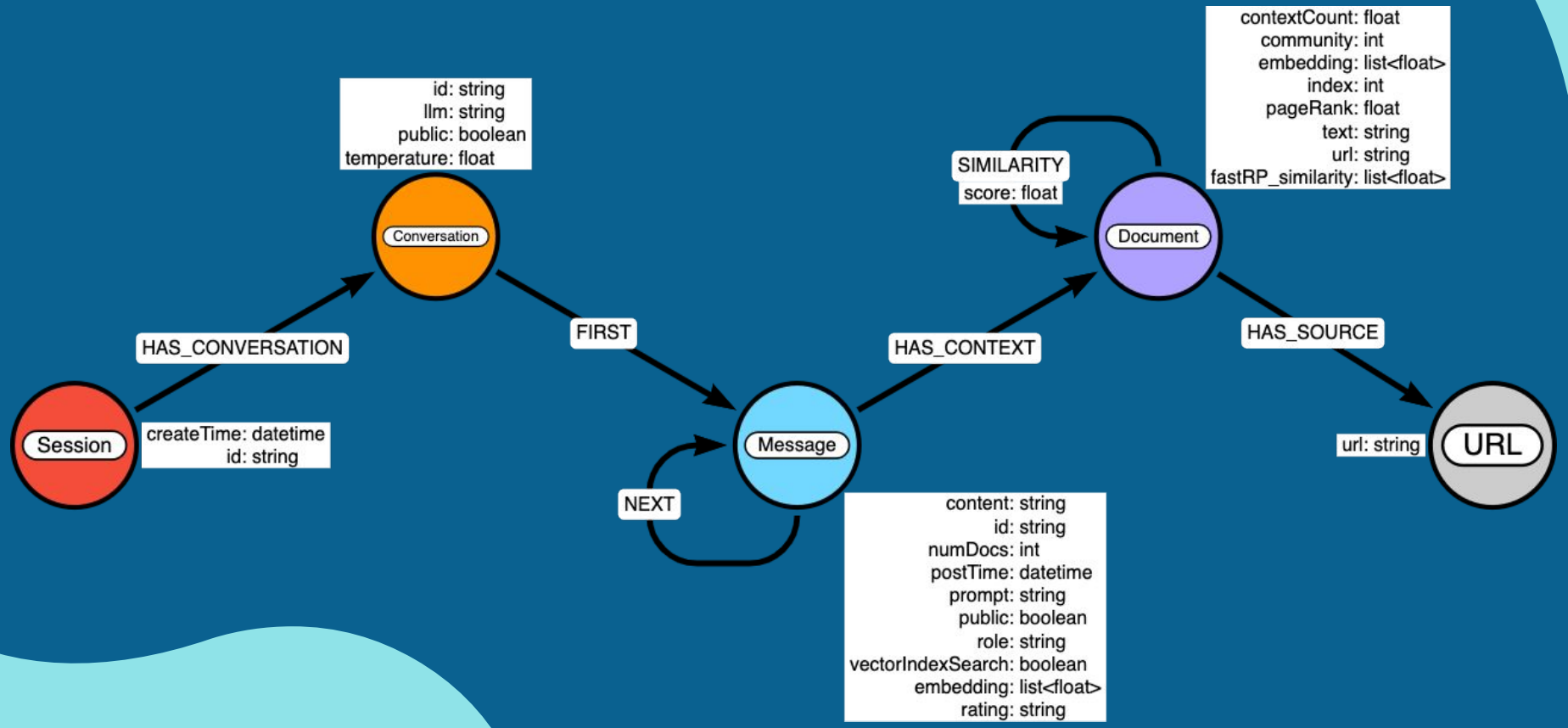
Logging Conversations with the LLM

Graphs Enable Explainable AI with LLMs

- Production
  - How the LLM use grounding documents
  - How they produce answers will become more and more important

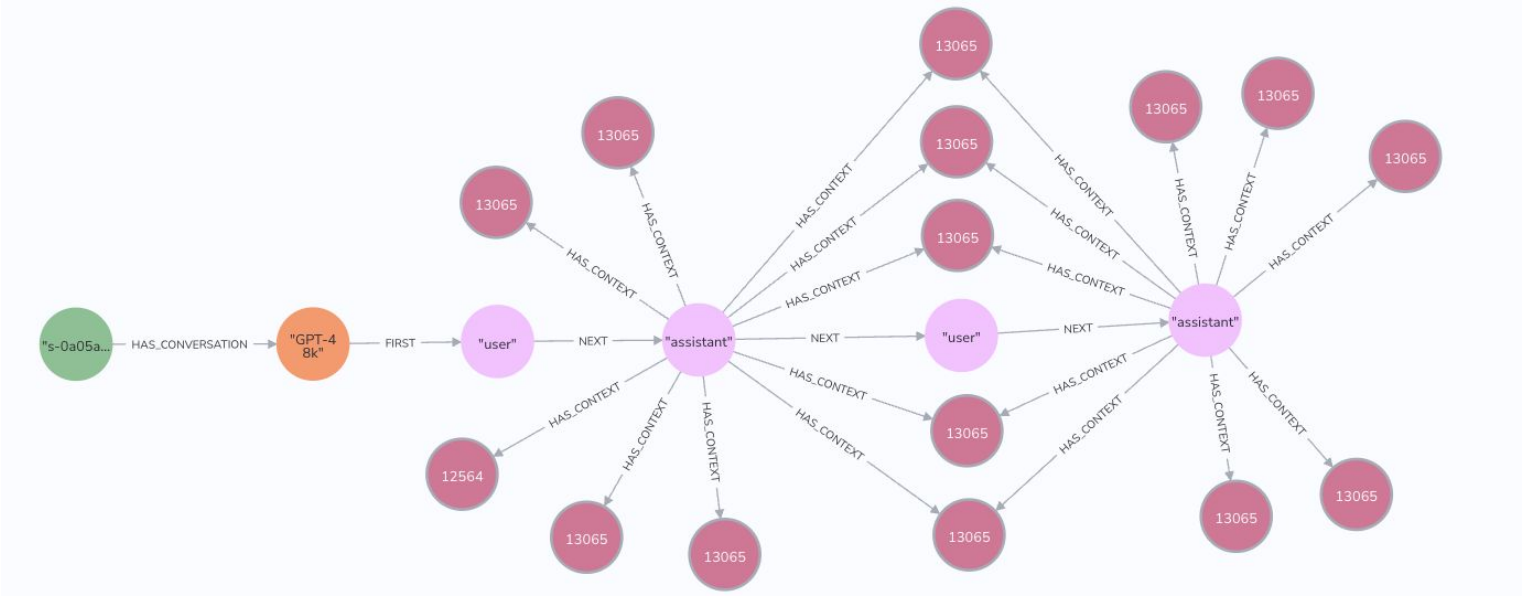
- Knowledge Graphs and GDS enable Explainable AI by:
  - Logging user interactions in the same database as the context
  - Visualizing conversations with context
  - Providing tools to analyze LLM performance and identify opportunities for improvement

Full Agent Neo Data Model



Logging and Visualizing Conversations

Graphs enable logging of LLM conversations in the same database as the context documents and with defined relationships.



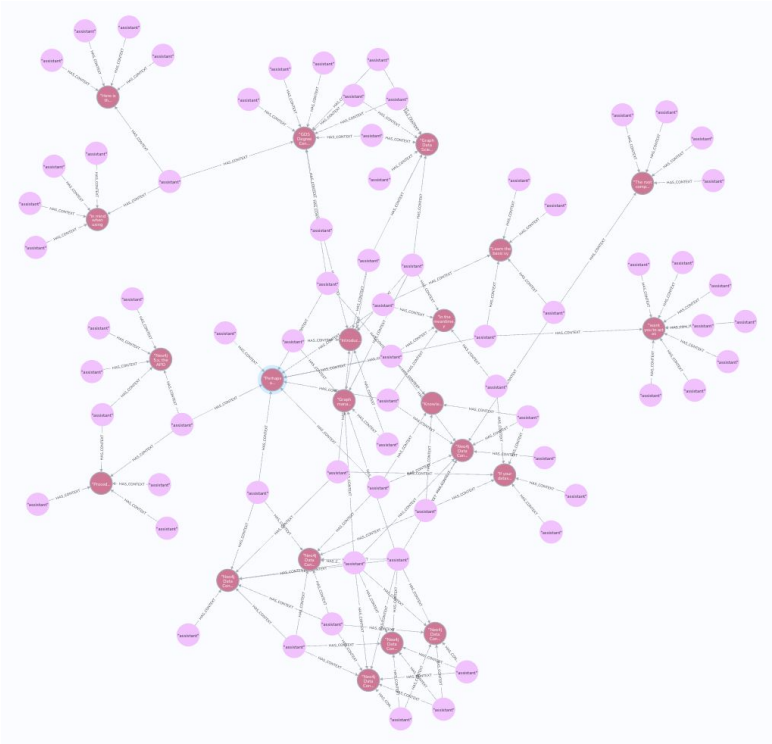
Graph of an actual conversation between an Agent Neo user and the ChatGPT-4 LLM. Context Documents are labeled with their GDS Community.

Relevant

Visualizing Context Document Usage

- Graphs enable us to visualize the most frequently used context Documents along with the associated LLM responses

- Natural clusters form in the graph even among the most frequently used Documents



LLM Responses (pink) and
Most Frequently Used Context Documents (red)

Thanks!

alison.cossette@neo4j.com