



Building a personal search engine with llama-index

Judith van Stegeren & Yorick van Pelt

Before this tutorial

1. Install Python
2. Download the files from [url]
3. Follow the readme to install the requirements

Tutorial (1 hour)

Part 1: using a vector index (20 mins)

Short talk about search engines, llama-index, vector indices

Notebook 1: retrieval and querying

Part 2: building an index from your own documents (30 mins)

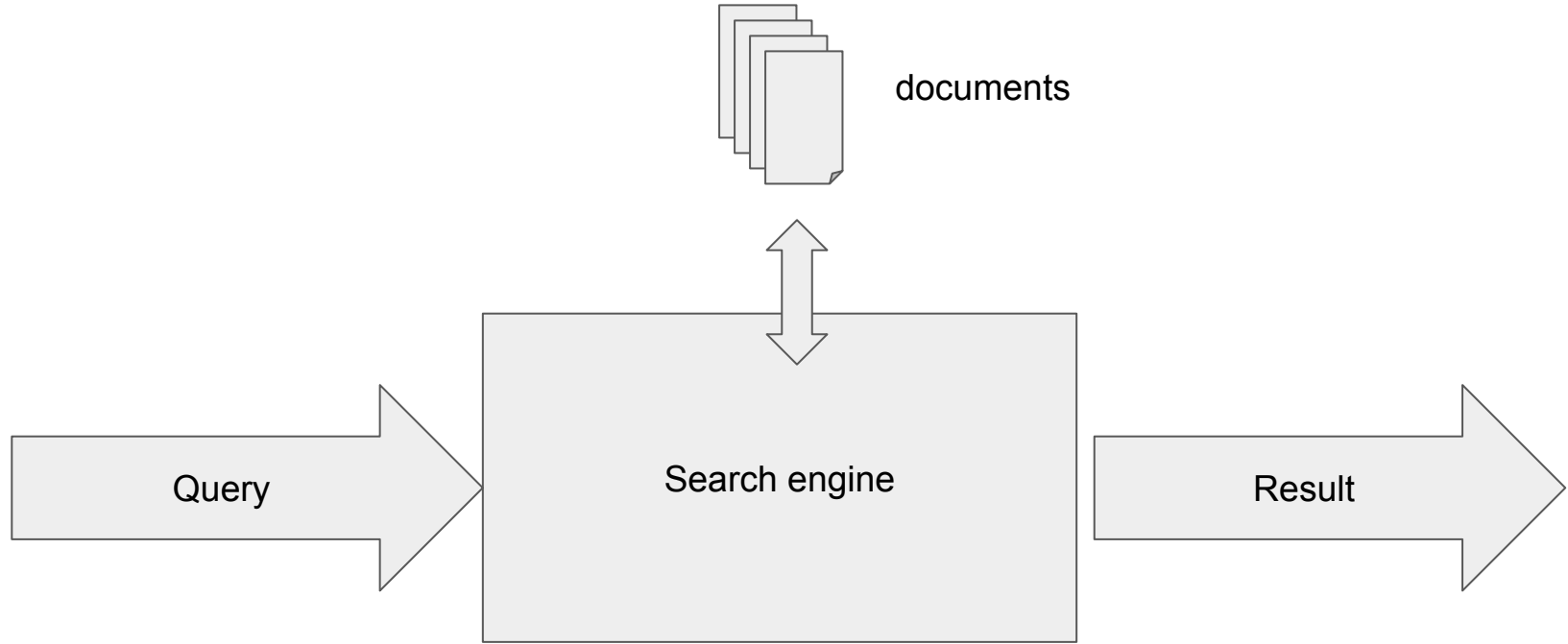
Short talk about loading documents, putting these into an index, metadata

Notebook 2: building an index from text documents

Wrap-up: things to explore after this tutorial

Part 1: searching with llama-index

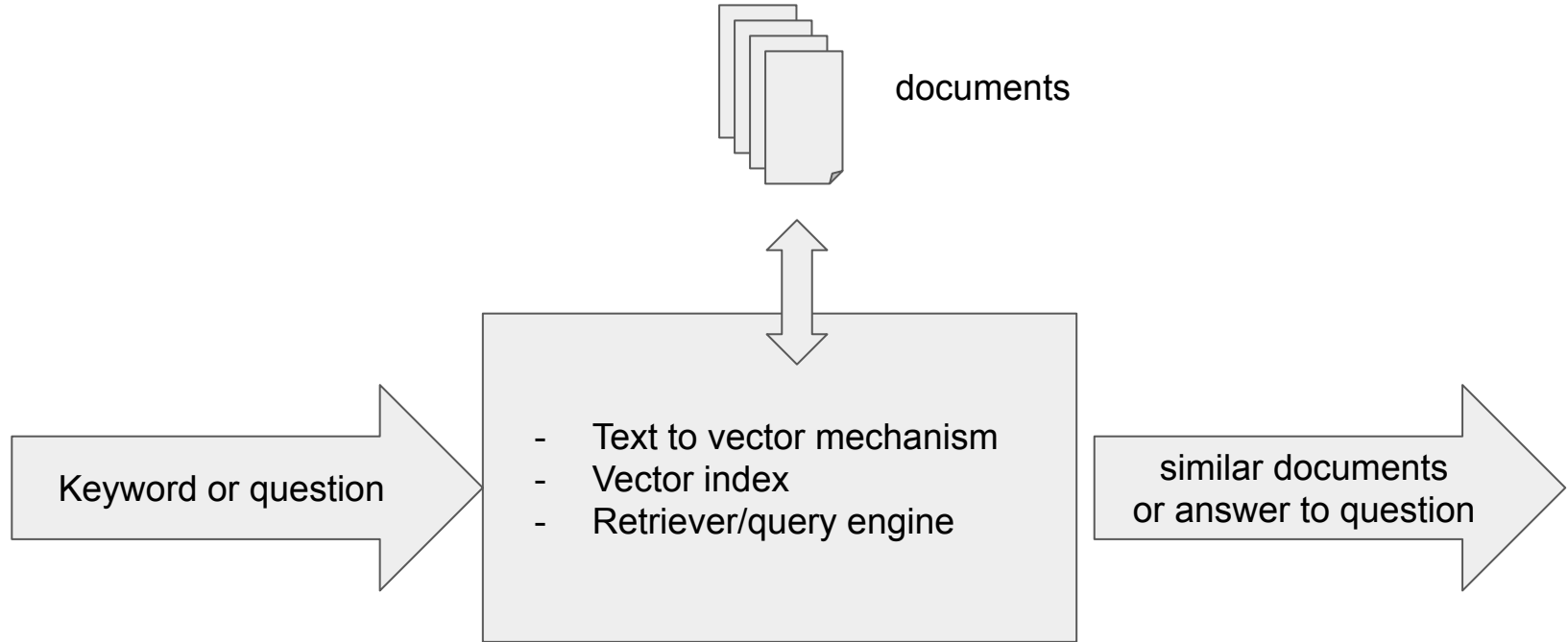
Personal search engine



```
import llama_index
```



Personal search engine



Vector index

Index: a list of items, each of which identifies a particular record in a computer file or database and contains information about its address

Vector: row of numbers

Vector index: index (lookup thing) for vectors

Vector index based on documents: index for vectors of documents

Retrieving vs querying

Retrieval

Ingredients: index, query

Result: most “similar” things in the index

Querying

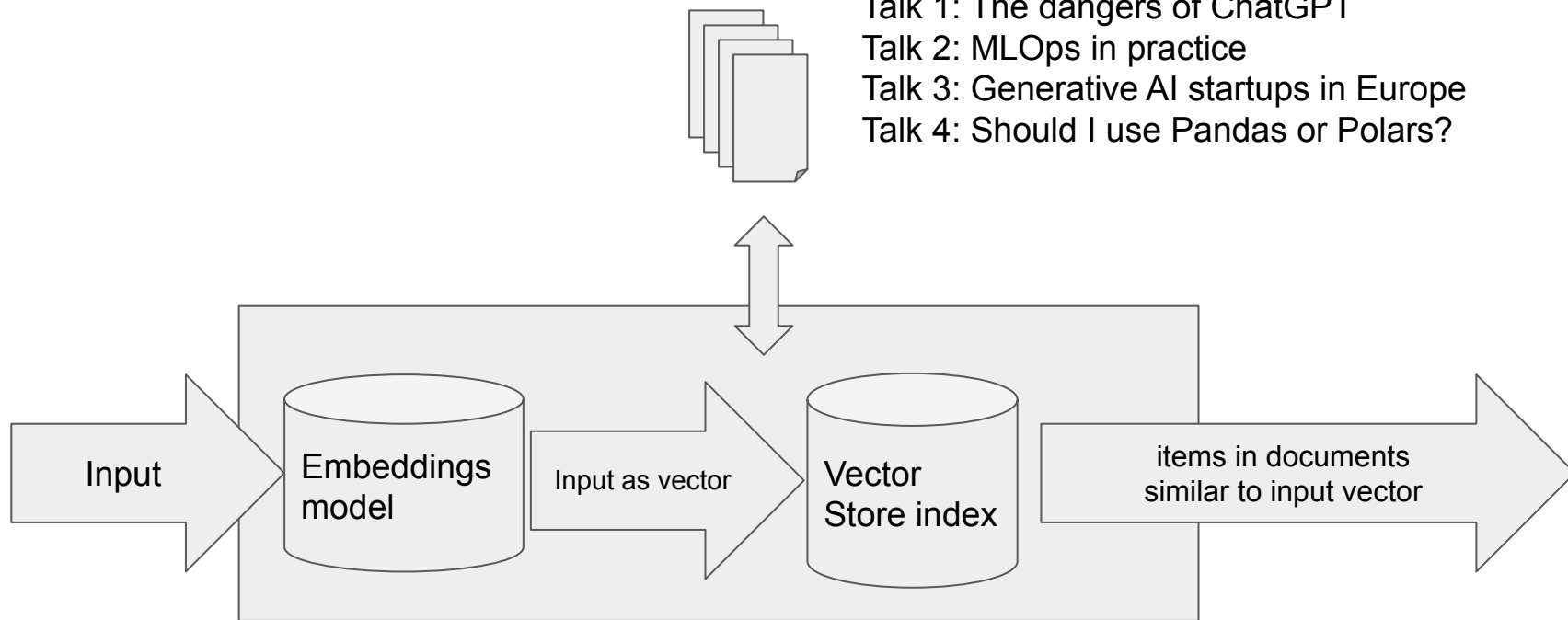
Ingredients: index, query, context

Result: LLM uses the context to write an answer to the query

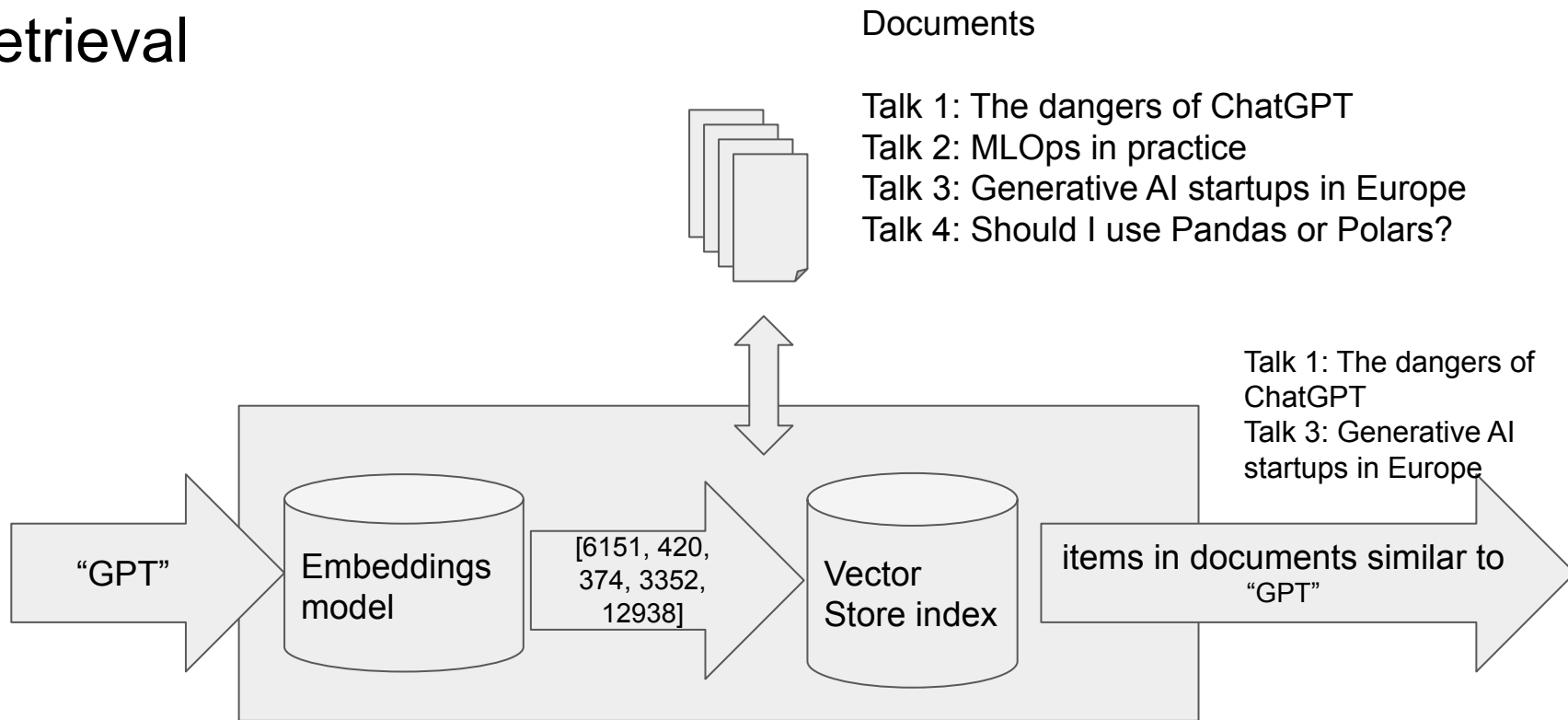
Retrieval

Documents

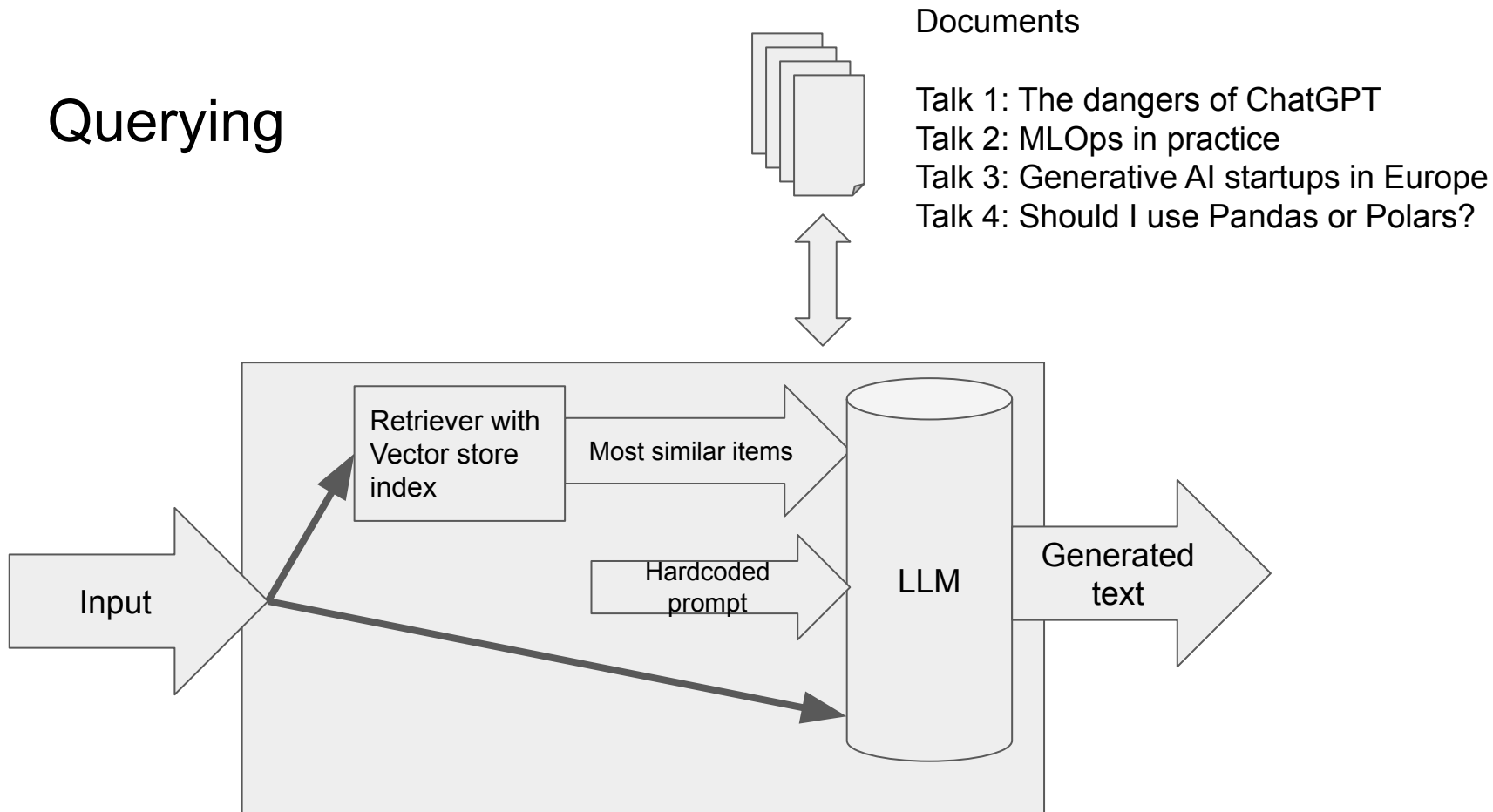
Talk 1: The dangers of ChatGPT
Talk 2: MLOps in practice
Talk 3: Generative AI startups in Europe
Talk 4: Should I use Pandas or Polars?



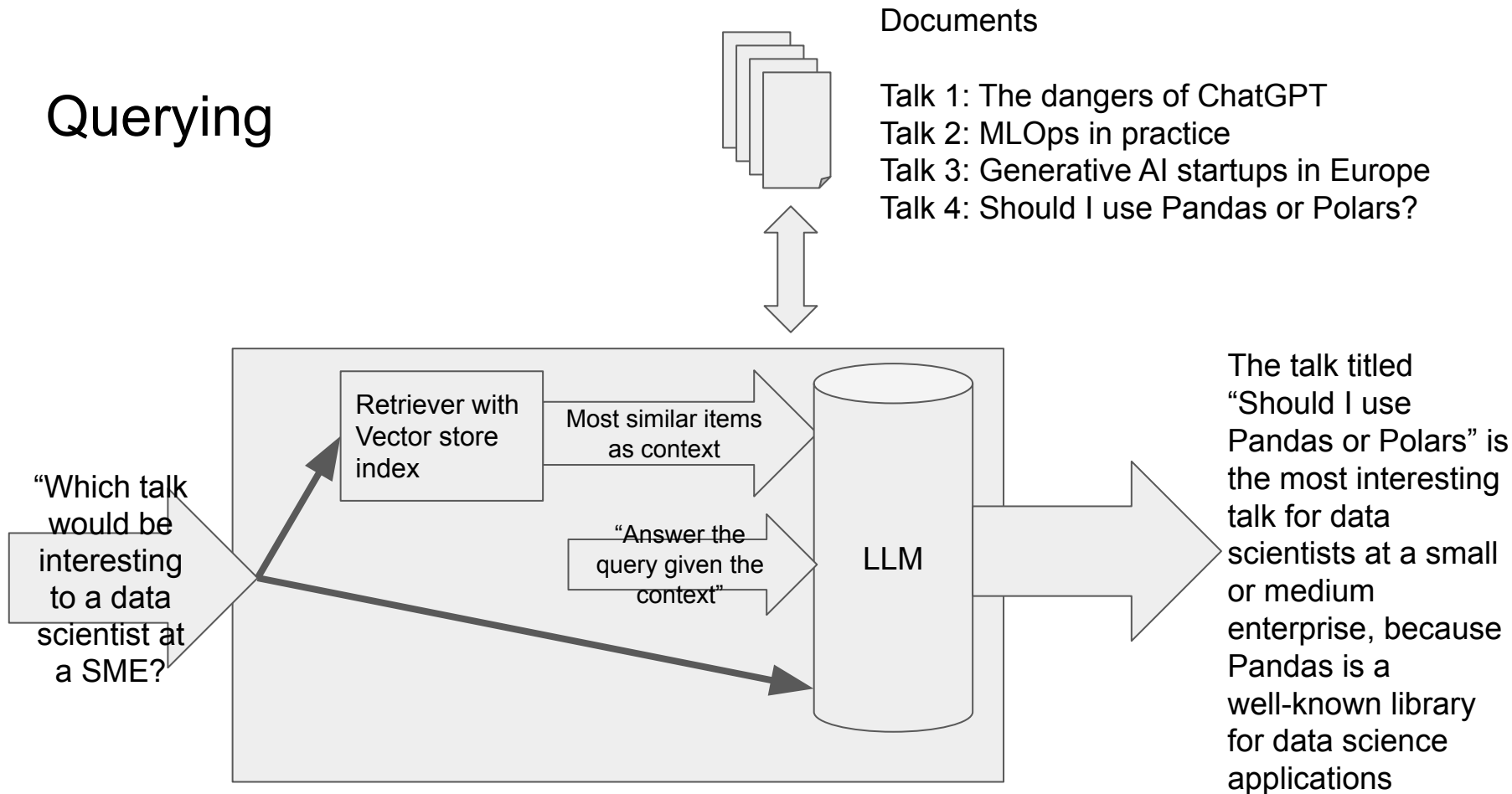
Retrieval



Querying



Querying



Our first index

`index` is a vector store index with PyData Amsterdam 2023 talks!

Index contains of document snippets called nodes.

Our first index

`index` is a vector store index with PyData Amsterdam 2023 talks!

Index consists of document snippets called nodes.

A `node` has `text` and `metadata`

```
node.text
```

```
"Building a personal search engine with llama-index\n\nWouldn't it be great to have a Google-like search engine,\nbut then for your own text files and completely private?\nIn this tutorial (...)"
```

```
node.metadata
```

```
{'title': 'Building a personal search engine with llama-index', 'speakers': 'Judith van Stegeren,\nYorick van Pelt'}
```

Retrieving with llama-index

`index` is a vector store index with the PyData schedule

```
retriever = index.as_retriever()
```

```
search_results = retriever.retrieve("your query here")
```


Dealing with search results

`index` is a vector store index with the PyData schedule

`search_results` is a list of retrieval results

A result has a node and a score

```
first_result = search_results[0]
```

```
first_result.node
```

```
first_result.score
```

Querying with llama-index

`index` is a vector store index with the PyData schedule

```
query_engine = index.as_query_engine()
```

```
search_results = query_engine.query("your query here")
```

Time for a notebook!

`index` is a vector store index with the PyData schedule

```
node.text
```

```
node.metadata
```

```
retriever = index.as_retriever()
```

```
search_results = retriever.retrieve("your query here")
```

```
query_engine = index.as_query_engine()
```

```
search_results = query_engine.query("your query here")
```

```
result.node
```

```
result.score
```

Part 2: building a vector index

Recap: our first index

`index` is a vector store index with PyData Amsterdam 2023 talks!

Index consists of document snippets called nodes.

A `node` has `text` and `metadata`

```
node.text
```

```
"Building a personal search engine with llama-index\n\nWouldn't it be great to have a Google-like search engine,\nbut then for your own text files and completely private?\nIn this tutorial (...)"
```

```
node.metadata
```

```
{'title': 'Building a personal search engine with llama-index', 'speakers': 'Judith van Stegeren,\nYorick van Pelt'}
```

Building a new vector store index

An index is created from a set of Documents.

Documents are split into Nodes (we've seen these before!)

Four steps:

1. Load data from source (vanilla Python)
2. Transform data to "Document" format
3. Put documents in vector store index
4. Save the vector store index to file

Step 1: loading data

from plaintext file, from json, from the web, etc.

All kinds of custom data loaders on <https://llamahub.ai/>. (Quality varies)

```
# plaintext
```

```
with open(my_research_notes.txt, 'r') as infile:  
    research_notes = infile.read()
```

```
# json
```

```
with open(movie_descriptions.json, 'r') as infile:  
    movies = json.loads(infile.read())
```

```
# from web
```

```
response = requests.get(httpx://myfavoriteapi.com/endpoint?query=somedata)  
api_data = json.loads(response.text)
```

Step 2: creating Documents from data

Transform data to a structure that llama_index understands.

All Document properties are completely optional!

```
document = llama_index.Document() # empty document

Document(
    id_='82c3f6b8-4e64-4d76-9629-f51aaa7446a0',
    embedding=None,
    metadata={},
    excluded_embed_metadata_keys=[],
    excluded_llm_metadata_keys=[],
    relationships={},
    hash_='44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff8a',
    text='',
    start_char_idx=None,
    end_char_idx=None,
    text_template='{metadata_str}\n\n{content}',
    metadata_template='{key}: {value}',
    metadata_seperator='\n'
)
```


Step 2: creating Documents from data

Transform data to a structure that llama_index understands.

All Document properties are completely optional!

```
document = llama_index.Document() # empty document
```

```
Document(  
    id_='82c3f6b8-4e64-4d76-9629-f51aaa7446a0',  
    metadata={},  
    text='',  
)
```

Step 2: creating Documents from data

Simplest way to create a document: just add text.

```
simple_text = "Hello world"

document = llama_index.Document(text=simple_text)

Document(
    id_='993d3bfd-83df-4fb5-b6f7-5ea688df8032',
    metadata={},
    text='Hello world',
)
```

Step 2: creating Documents from data

You can add custom identifiers to Documents.

This is good for tracking Document sources!

Important: use `id_` instead of `id` (this will fail silently)

```
my_filename = "somefile.txt"
with open(my_filename, 'r') as infile:
    my_text = infile.read()

document = llama_index.Document(text=my_text, id_=my_filename)

Document(
    id_='somefile.txt',
    metadata={},
    text='The contents of a file on my computer',
)
```

Step 2: creating Documents from data

Add metadata as dictionary

```
simple_text = "Hello world"
some_metadata = { "author" : "Judith", "created_at": datetime.now() }

document = llama_index.Document(text=simple_text, metadata=some_metadata)

Document(
  id_='70b4c14c-18e9-49c8-9265-01760b8d5c3d',
  metadata={
    'author': 'Judith',
    'created_at': datetime.datetime(2023, 9, 8, 11, 7, 55, 281502)
  },
  text='Hello world'
)
```

Step 3: creating vector index from data

```
documents = [Document(text=datapoint) for datapoint in data]
```

```
index = VectorStoreIndex.from_documents(documents)
```

And we're done!

Now we can create a `retriever` or `query_engine` and do everything we saw in part 1.

Step 3: creating vector index from data

We can use a `ServiceContext` to change:

- the size of text in nodes, or chunk size
- the overlap between nodes
- the embeddings model for transforming text to numbers
- the language model used for querying

And lots of other things as well!

Step 3: creating vector index from data

```
my_service_context = ServiceContext.from_defaults()
my_documents = [Document(text=datapoint) for datapoint in data]

index = VectorStoreIndex.from_documents(documents=my_documents,
service_context=my_service_context)
```

In part 1 of this tutorial, we used a ServiceContext to

- use the smallest sentence-transformers embeddings model
- remove the default LLM so llama-index won't call OpenAI's GPT

Step 3: creating vector index from data

```
service_context = ServiceContext.from_defaults()

service_context = ServiceContext.from_defaults(llm=None)

my_model = "local:sentence-transformers/all-minilm-l6-v2" # llama-index will
download it for us
service_context = ServiceContext.from_defaults(embed_model=my_model)

service_context = ServiceContext.from_defaults(chunk_size=10)
```


Time for a notebook!

`index` is a vector store index

```
node.text
node.metadata
```

```
retriever = index.as_retriever()
search_results = retriever.retrieve("your query here")
```

```
query_engine = index.as_query_engine()
search_results = query_engine.query("your query here")
```

```
result.node
Result.score
```

```
document = Document(text="hello", id_="hello.txt", metadata={"author":"Judith"})
```

```
documents = [Document(text=datapoint) for datapoint in data]
```

```
from llama_index.llms import OpenAI
my_llm = OpenAI(temperature=0.1, model="gpt-4")
my_embed_model =
service_context = ServiceContext.from_default(llm=my_llm, embed_model=my_embed_model, chunk_size=my_chunk_size)

index = VectorStoreIndex.from_documents(documents=documents, service_context=my_service_context)
```

Time for a notebook!

`index` is a vector store index with the PyData schedule

```
node.text
```

```
node.metadata
```

```
retriever = index.as_retriever()
```

```
search_results = retriever.retrieve("your query here")
```

```
query_engine = index.as_query_engine()
```

```
search_results = query_engine.query("your query here")
```

```
result.node
```

```
result.score
```

Time for a notebook!

```
document = Document(text="hello", id_="hello.txt", metadata={"author": "Judith"})

documents = [Document(text=datapoint) for datapoint in data]

service_context = ServiceContext.from_default(llm=my_llm, embed_model=my_embed_model, chunk_size=my_chunk_size)

index = VectorStoreIndex.from_documents(documents=documents, service_context=my_service_context)
```