

# **Retrieval-Augmented Generation (RAG)**

# Agenda

- The idea behind RAG and examples of applications
- Document storage methods
- Document retrieval
- Quality assessment
- Tools for implementing RAG

# LLM generation problems

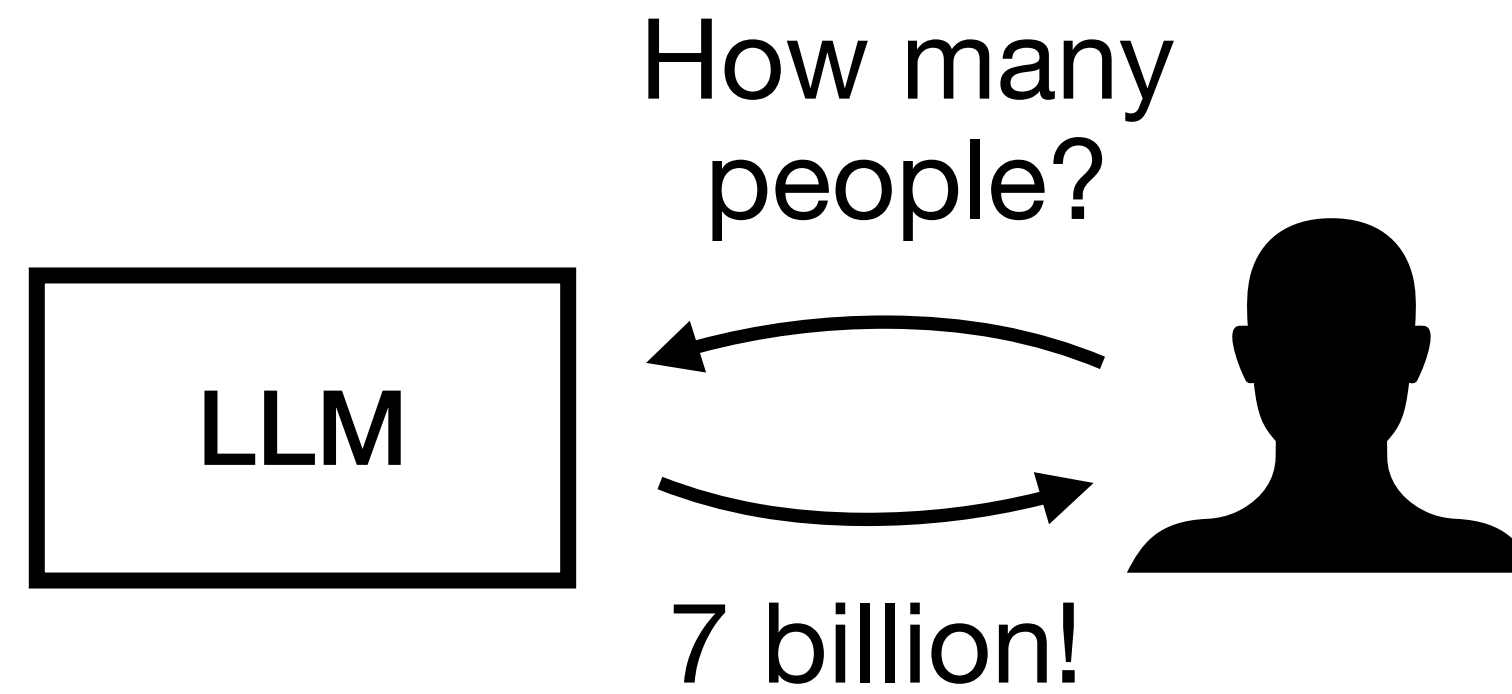
We want to know the answer to the question

"How many people are there on earth?"

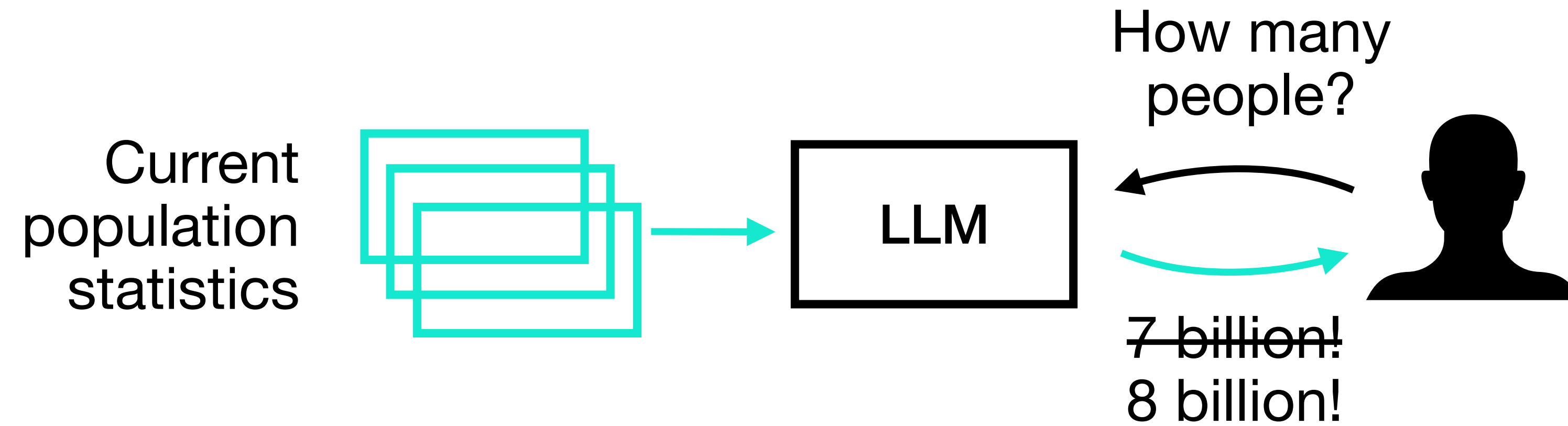
What can go wrong?

- The model might hallucinate
- The information was not presented in the training corpus
- The information is out of date

# LLM generation problems

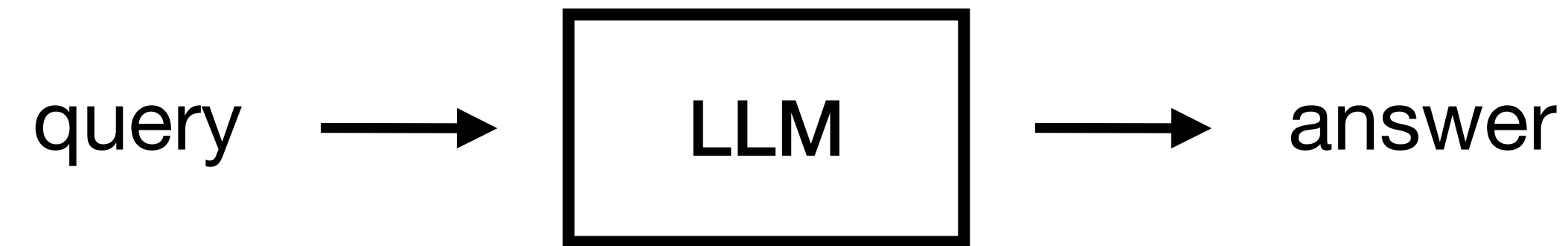


# LLM generation problems

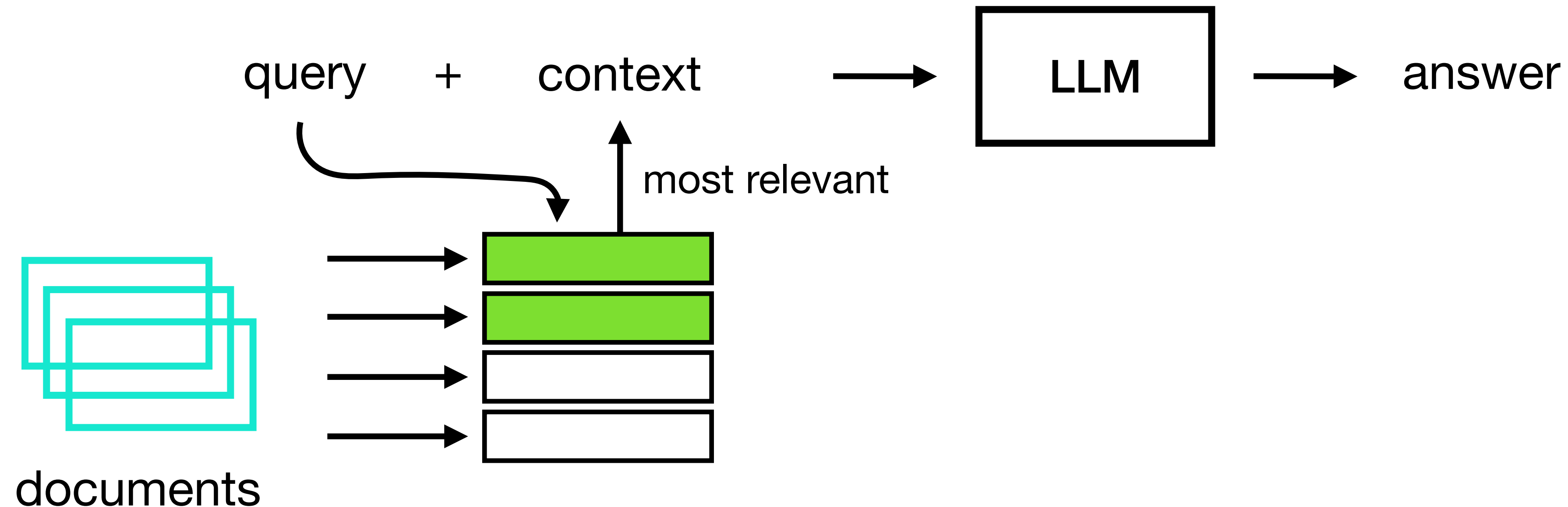


# Retrieval-Augmented Generation (RAG)

## Default application method



## Retrieval-Augmented Generation

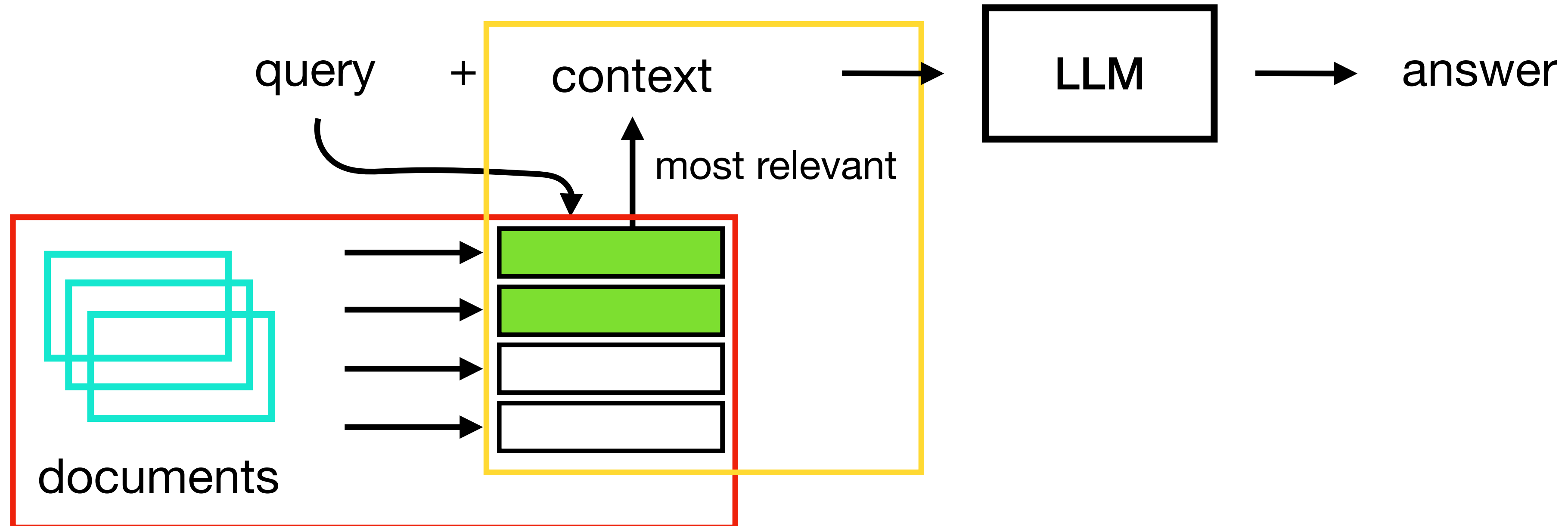


# When to use RAG

- **Documentation search engine**
- **Legal assistance on a specific topic**
- **Educational applications**
  - Adding textbooks to the database
- **Personalised recommendations**
  - Saving all relevant information about the user
- **Customer support**
  - The database stores all information about the company's services

# What next?

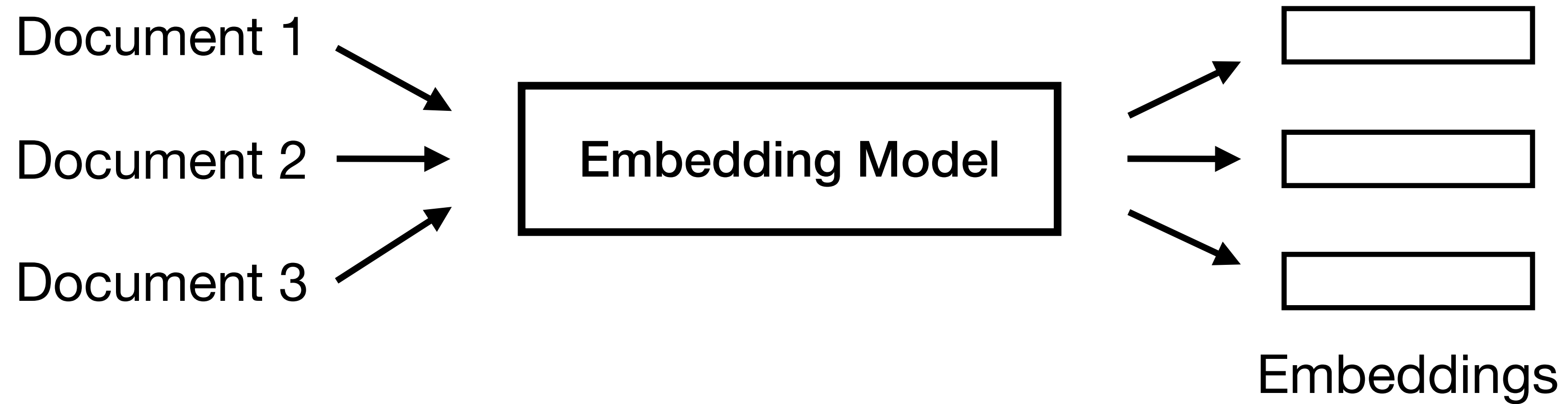
- How to store documents?
- How to select the most relevant ones?





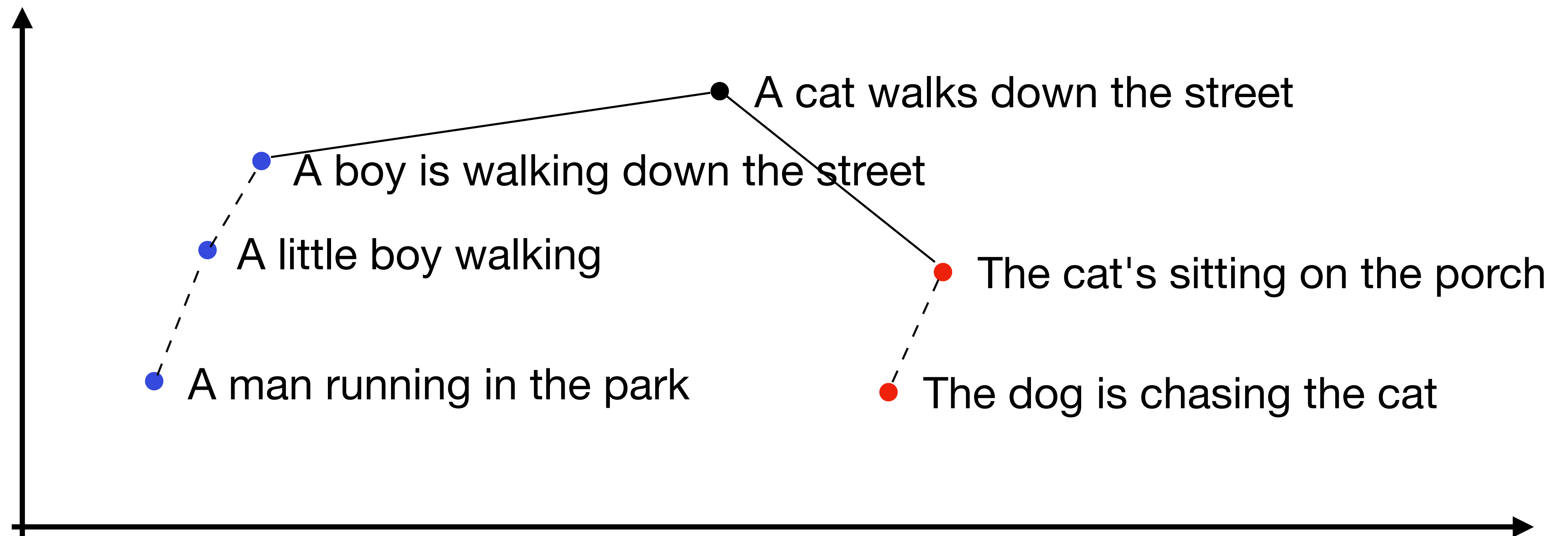
# Vector database

A way of storing texts with semantic search capability



# Vector database

A way of storing texts with semantic search capability



# Document encoding

Encoding the whole document at once is a bad idea!

- You have to feed the whole document into a context, and the context size is limited.
- Speed and memory costs are quadratically dependent on the context size.
- A document may contain conflicting information.
  - The population of the earth for different years.
- Embeddings have limited capacity, so some information can vanish

It is better to break each document into several chunks

# Methods of document chunking

- Fixed size chunking
- Recursive chunking
- Semantic chunking

# Fixed size chunking

Split the document into chunks with a fixed number of characters or tokens.  
Add chunk overlap to prevent important pieces of text from being split into different chunks.

Once upon a time there was an old mother pig who had three little pigs and not enough food to feed them. So when they were old enough, she sent them out into the world to seek their fortunes.

The first little pig was very lazy. He didn't want to work at all and he built his house out of straw. The second little pig worked a little bit harder but he was somewhat lazy too and he built his house out of sticks. Then, they sang and danced and played together the rest of the day.

The third little pig worked hard all day and built his house with bricks. It was a sturdy house complete with a fine fireplace and chimney. It looked like it could withstand the strongest winds.

# Recursive chunking

Takes text structure into account

Introduce a text chunk length limit and several levels of chunking

Standard character set for level division: ["\n\n", "\n", " ", ""].




Diagram illustrating the mapping of standard character sets to text levels:

- ↑ section
- ↑ paragraph
- ↑ word
- ↑ character

If a section doesn't fit into the limit, split it into paragraphs

If a paragraph doesn't fit, we divide it by words.

At the end, we divide it by characters

# Recursive chunking

Once upon a time there was an old mother pig who had three little pigs and not enough food to feed them. So when they were old enough, she sent them out into the world to seek their fortunes.

The first little pig was very lazy. He didn't want to work at all and he built his house out of straw. The second little pig worked a little bit harder but he was somewhat lazy too and he built his house out of sticks. Then, they sang and danced and played together the rest of the day.

The third little pig worked hard all day and built his house with bricks. It was a sturdy house complete with a fine fireplace and chimney. It looked like it could withstand the strongest winds.

# Semantic chunking

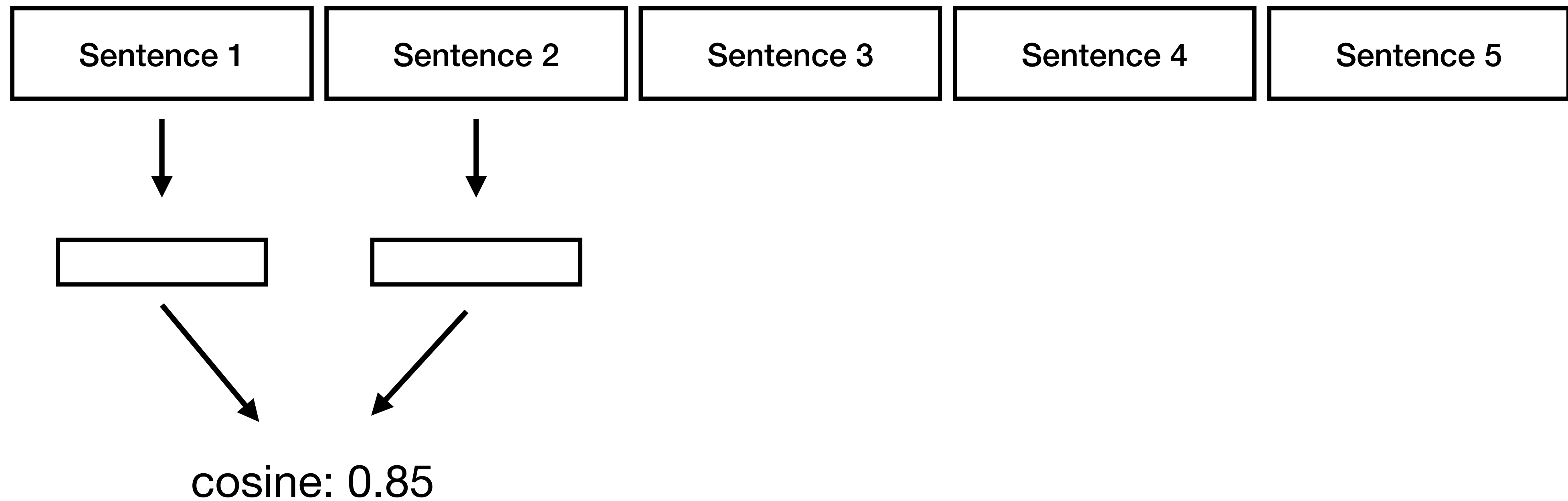
Finds a split by combining similar sentences into one chunk.





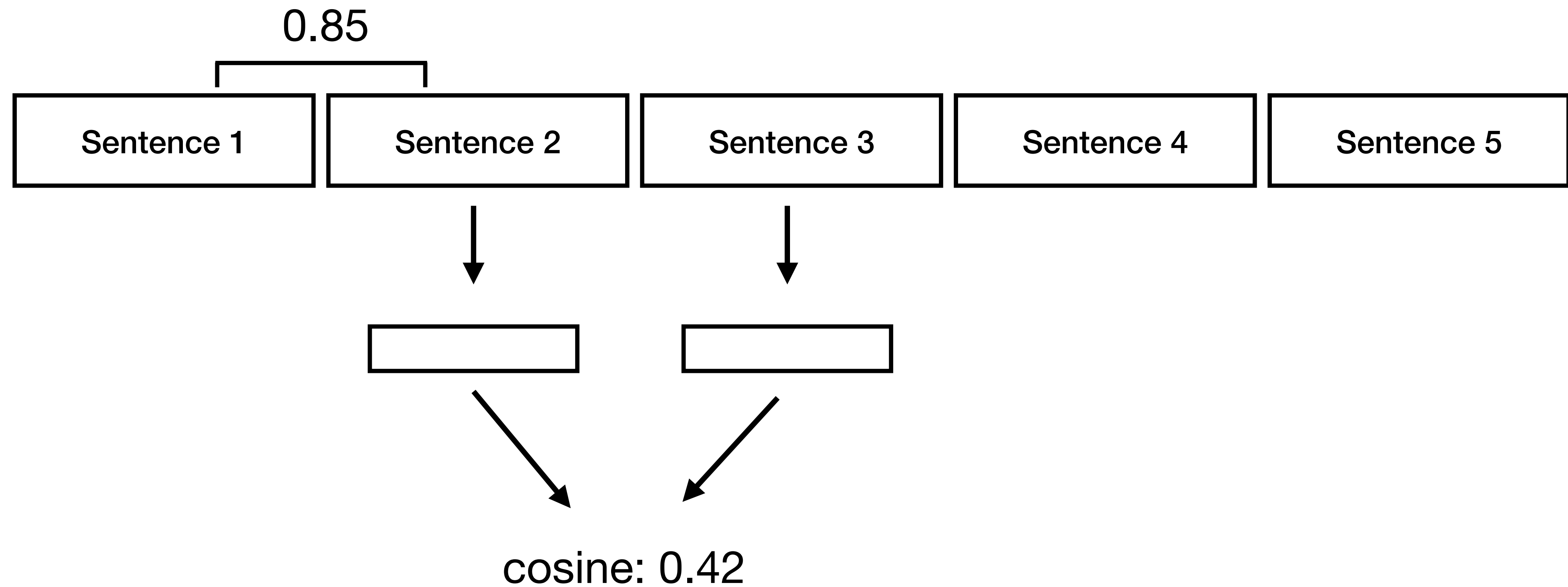
# Semantic chunking

Finds a split by combining similar sentences into one chunk.



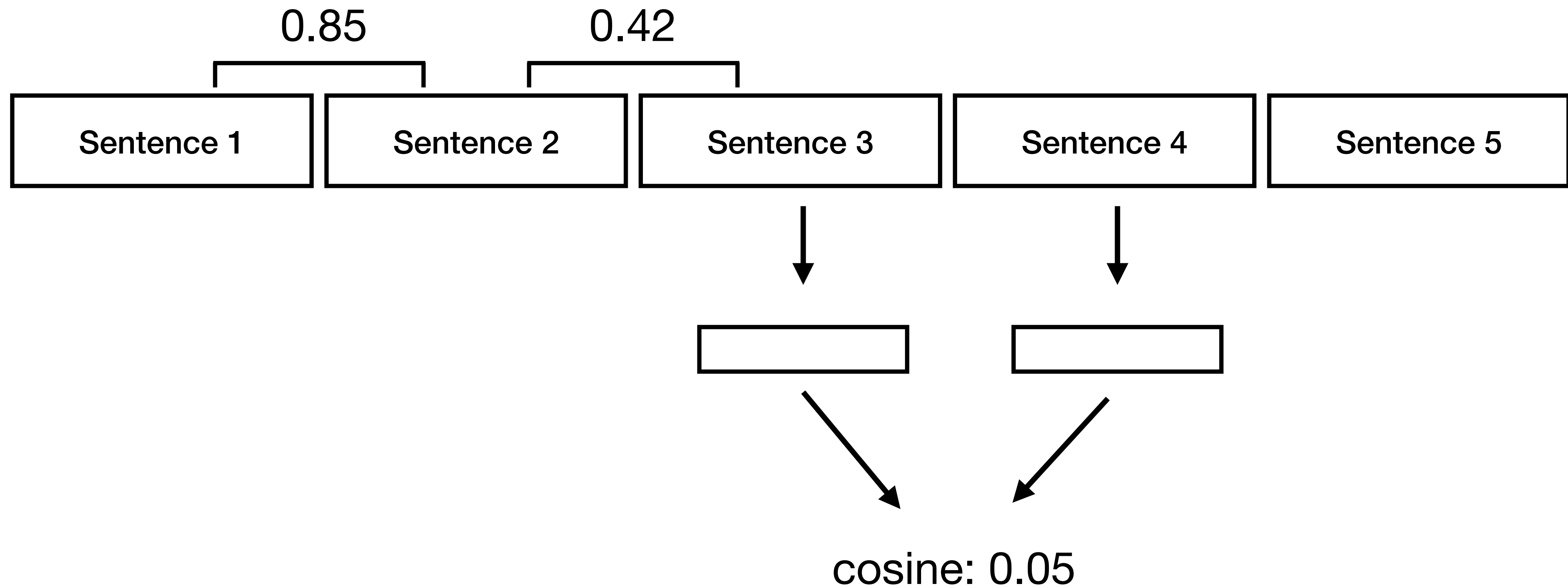
# Semantic chunking

Finds a split by combining similar sentences into one chunk.



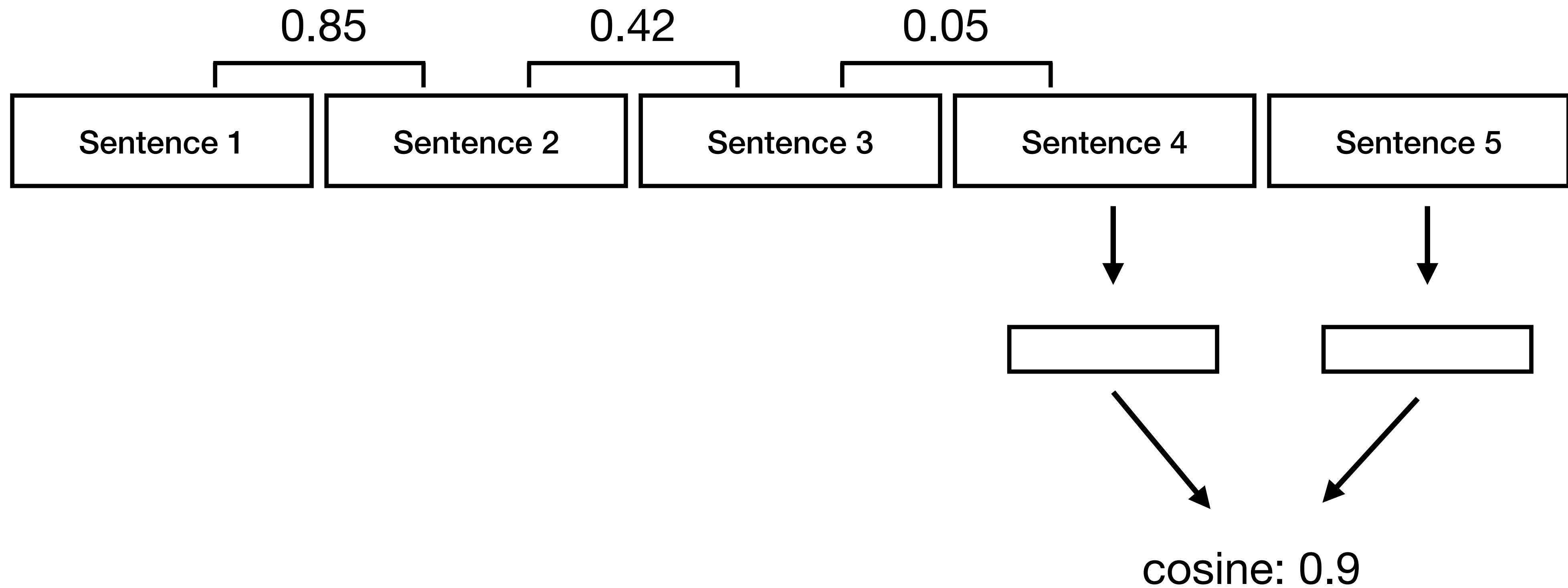
# Semantic chunking

Finds a split by combining similar sentences into one chunk.



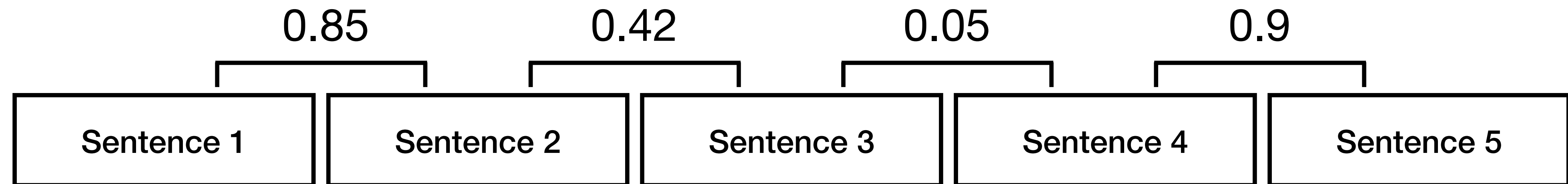
# Semantic chunking

Finds a split by combining similar sentences into one chunk.



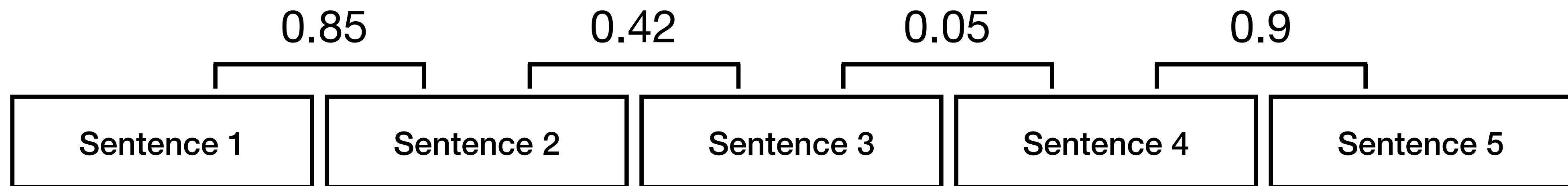
# Semantic chunking

Finds a split by combining similar sentences into one chunk.



# Semantic chunking

Finds a split by combining similar sentences into one chunk.

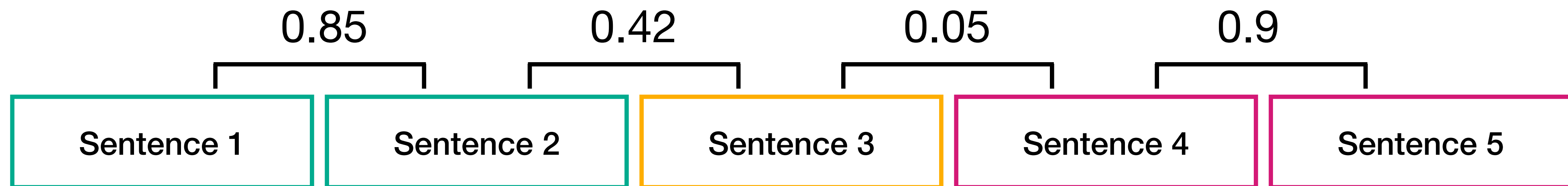


Choose a threshold depending on how many offers we want to combine

Quantile 50% - 0.85

# Semantic chunking

Finds a split by combining similar sentences into one chunk.



Choose a threshold depending on how many offers we want to combine

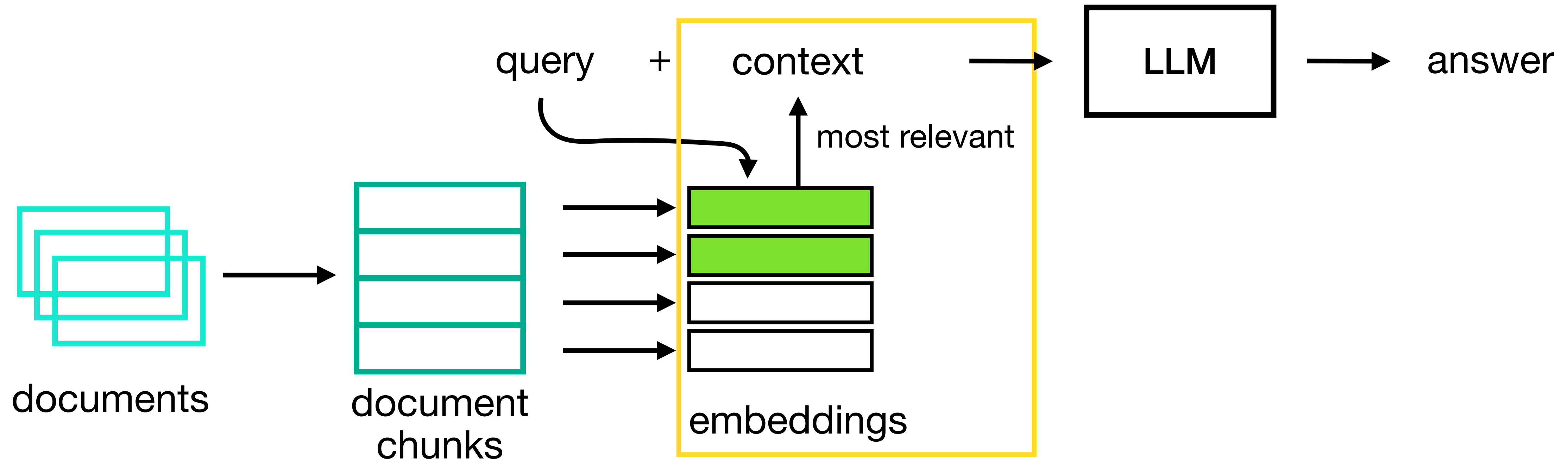
Quantile 50% - 0.85

Combine sentences with a similarity of at least 0.85

# What next?

- We figured out how to store documents
- How to select the most relevant ones?

## Retrieval-Augmented Generation

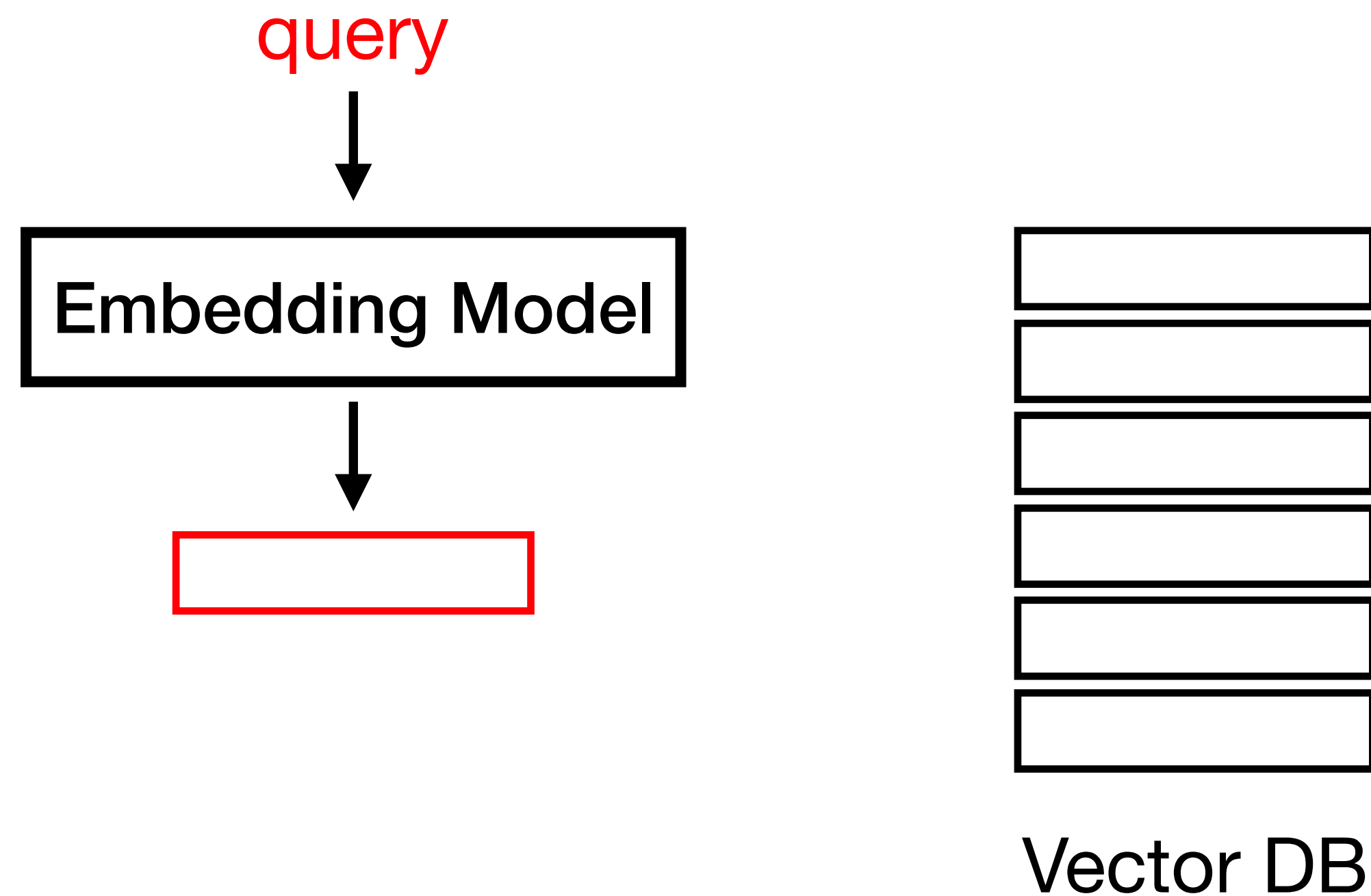




# How to choose relevant chunks?

## Retrieval

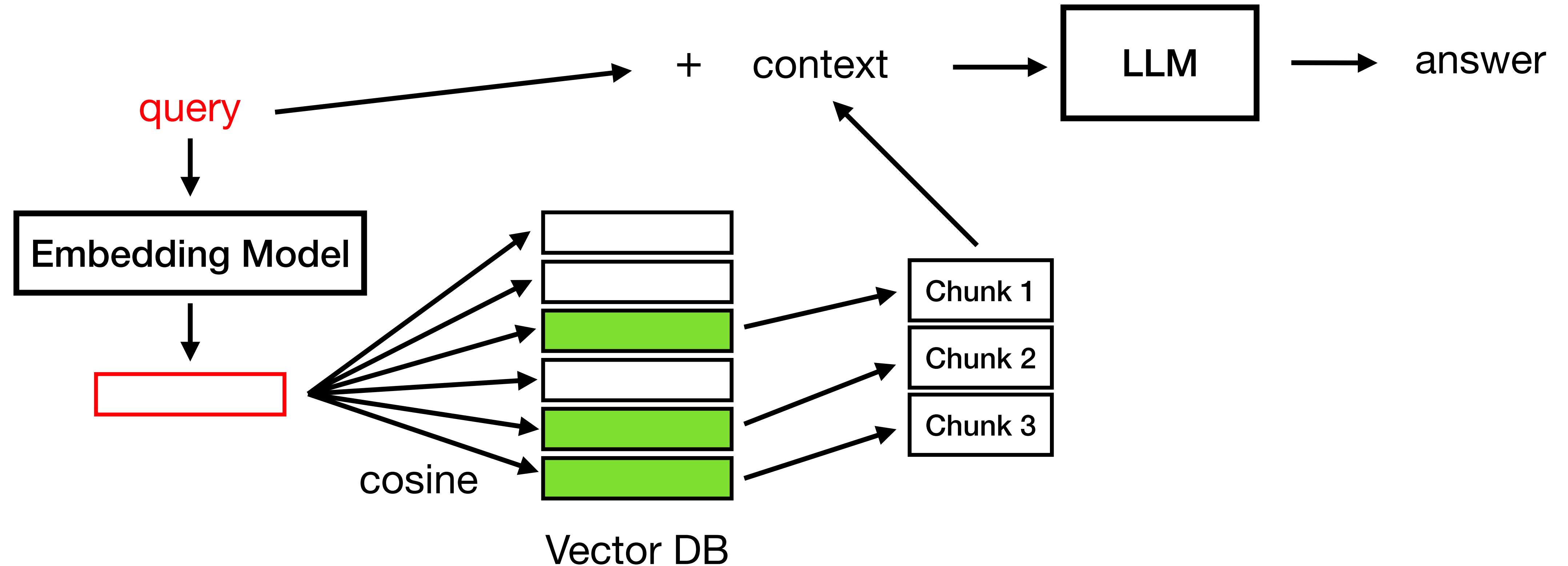
### Naive approach



# How to choose relevant chunks?

## Retrieval

### Naive approach



# Problems of the naive approach

1. It is necessary to consider similarity of each piece of text to a query  
Can take too much time for large databases
2. When text is compressed into embedding, some information gets lost  
Because of this, similarity measurement might be inaccurate
3. Only chunks of documents are extracted from DB  
Therefore context will contain cropped parts of texts
4. If the user's wording is imprecise, the required documents might not be found

# Locality Sensitive Hashing

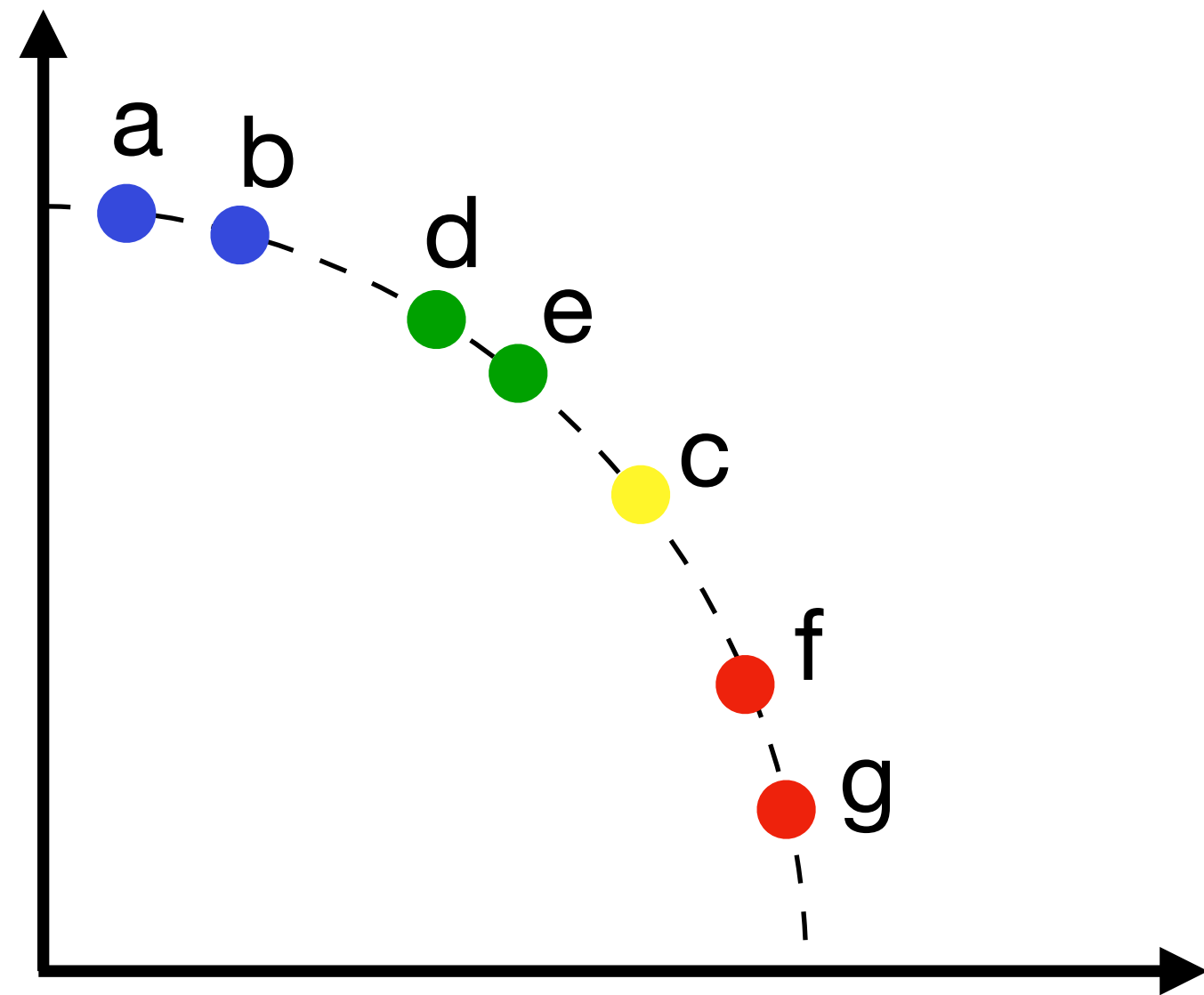
LSH is the most popular method for speeding up the search for relevant documents.

The idea is based on combining similar documents into one group.

If we can find the most relevant group fast, the search will be limited only to this group.

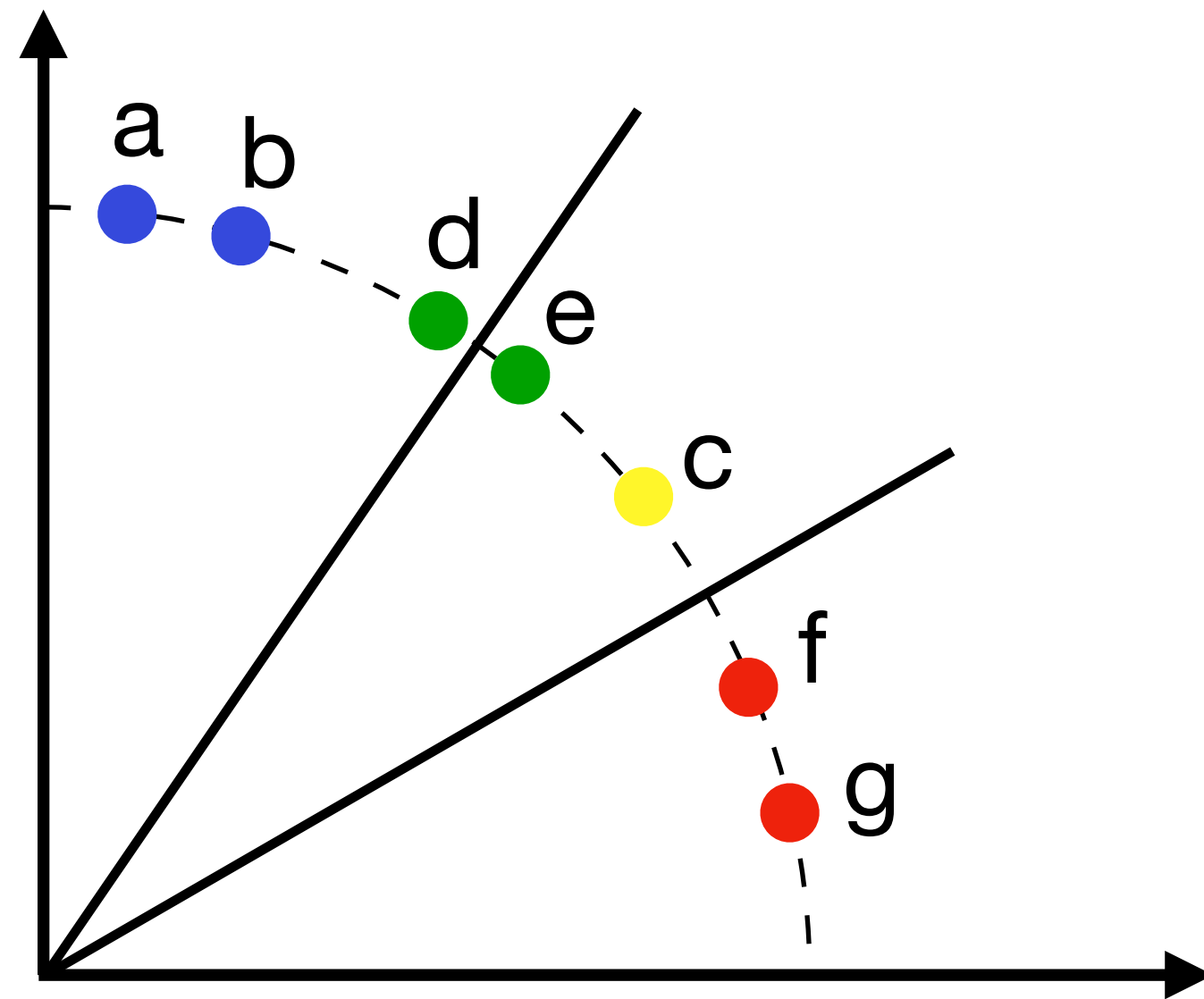
# LSH: how it works

- We have a set of normed vectors from the vector database
- Let's define a hash function that maps each vector to a hash



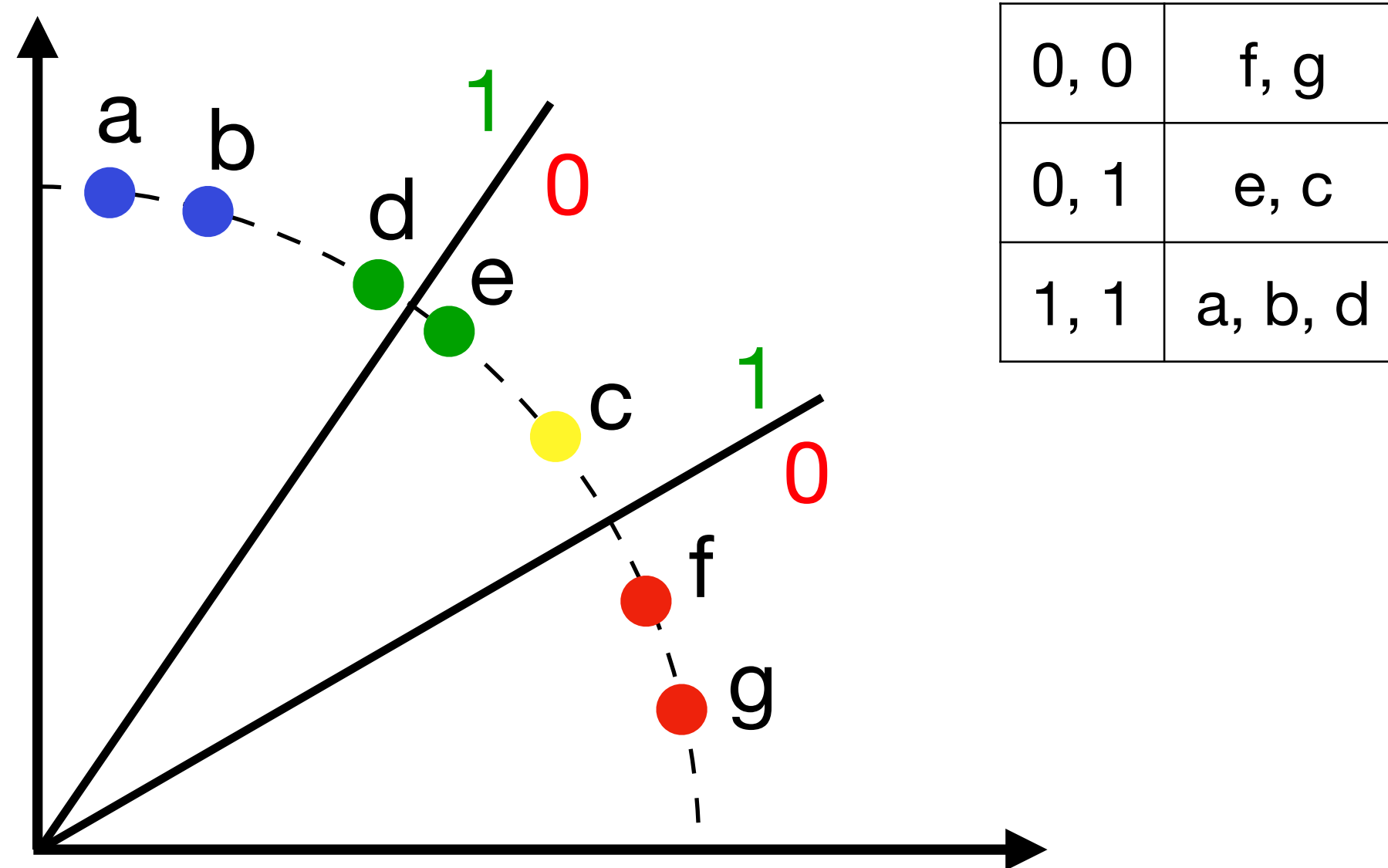
# LSH: how it works

- We have a set of normed vectors from the vector database
- Let's define a hash function that maps each vector to a hash
- Generate  $n$  random separating hyperplanes



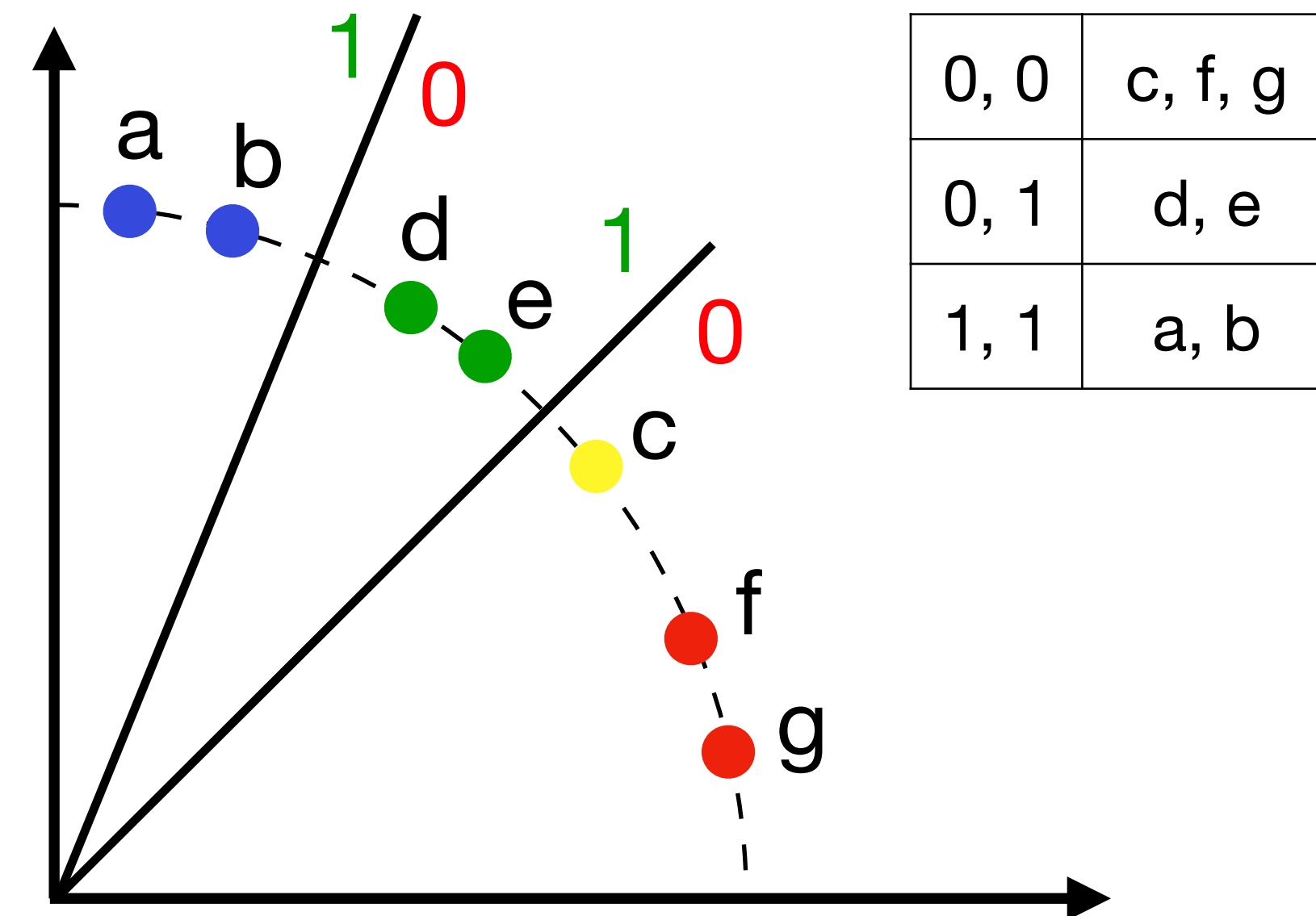
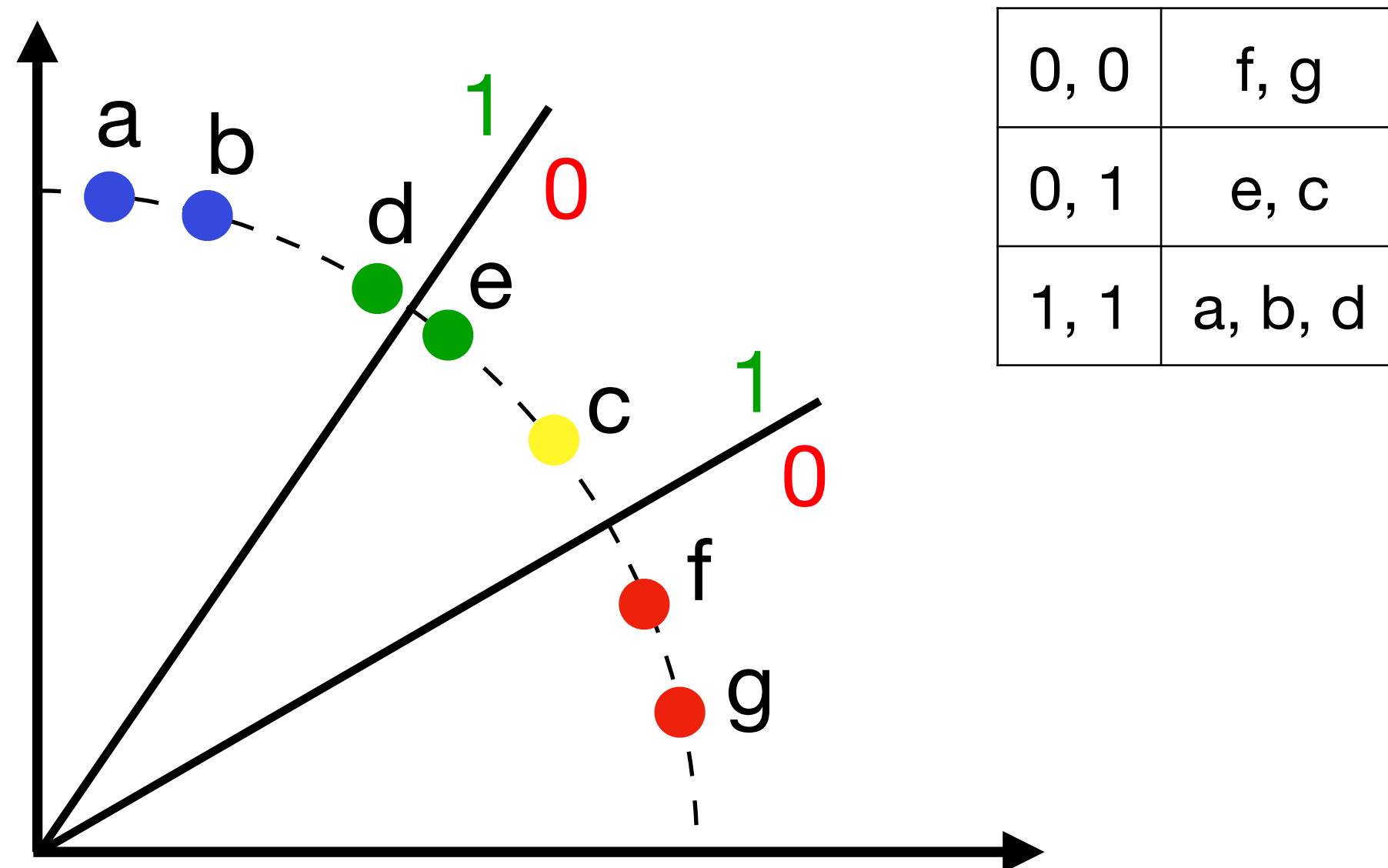
# LSH: how it works

- We have a set of normed vectors from the vector database
- Let's define a hash function that maps each vector to a hash
- Generate  $n$  random separating hyperplanes
- Write to the hash 0 if the point lies below the plane and 1 if above it



# LSH: how it works

- We have a set of normed vectors from the vector database
- Let's define a hash function that maps each vector to a hash
- Generate  $n$  random separating hyperplanes
- Write to the hash 0 if the point lies below the plane and 1 if above it
- Repeat the procedure  $k$  times

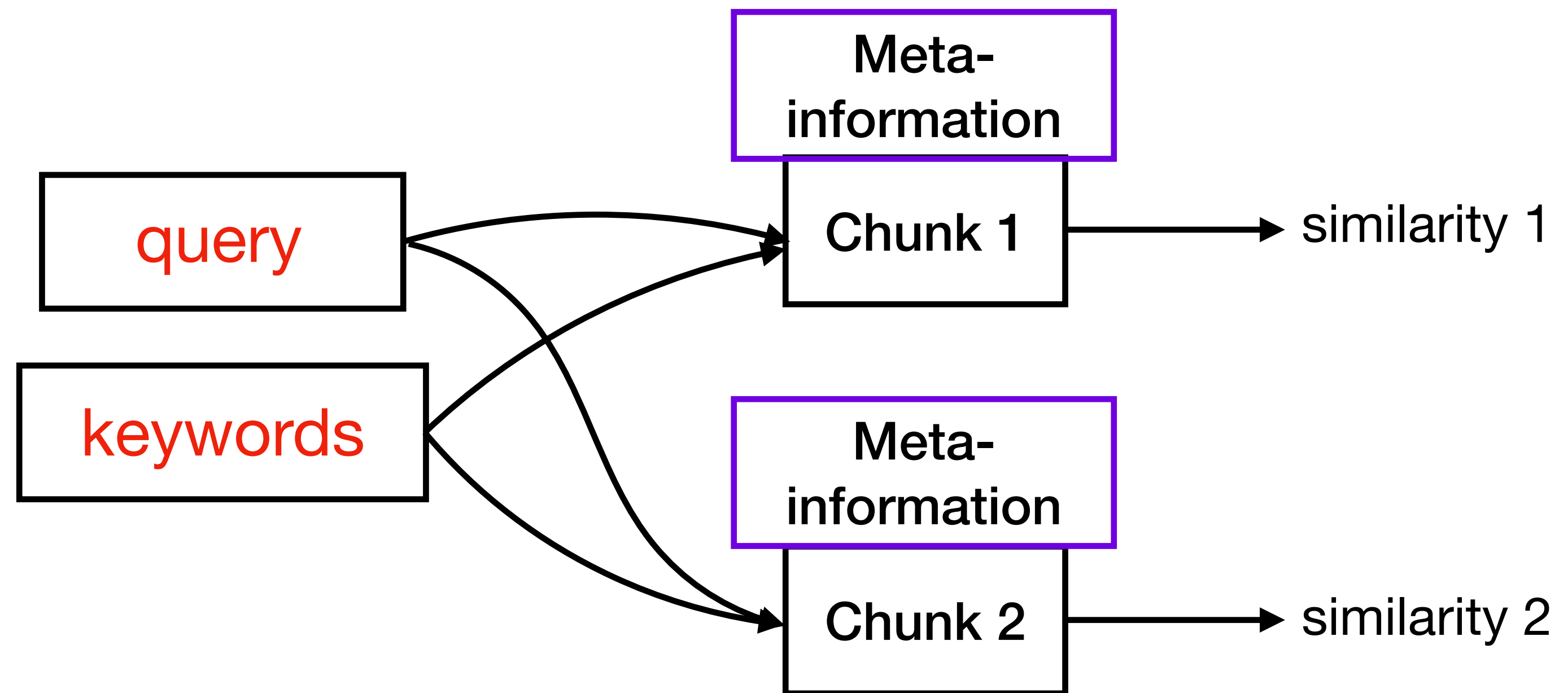




# Additional information

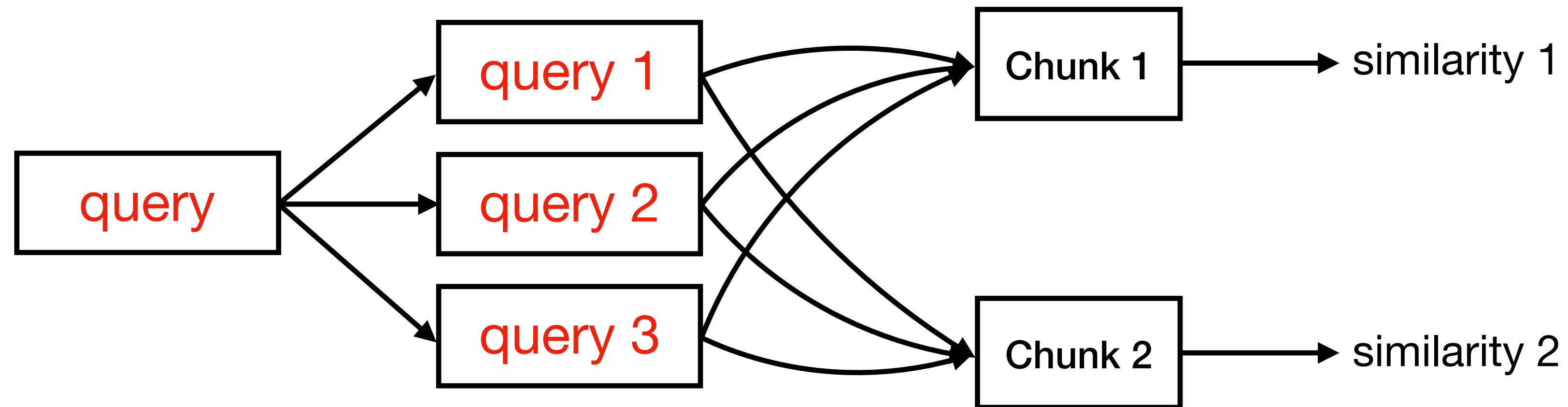
Additional information can be used to estimate similarity:

- Keywords: NER, LLM
- Meta-information at documents:
  - subject title for the textbook
  - film genre
  - product price



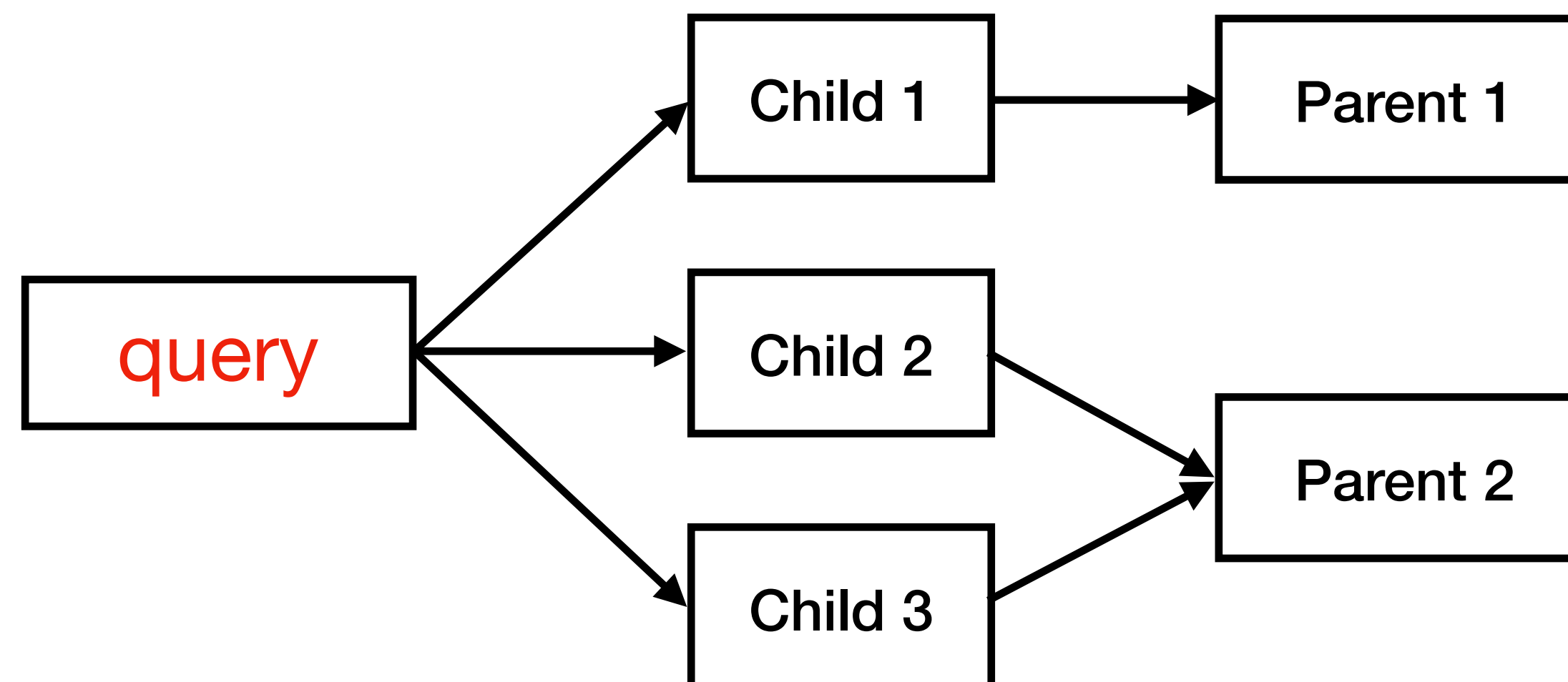
# Multi-Query

- Ask the LLM to reformulate the query several times
- Evaluate the similarities of chunks to each query



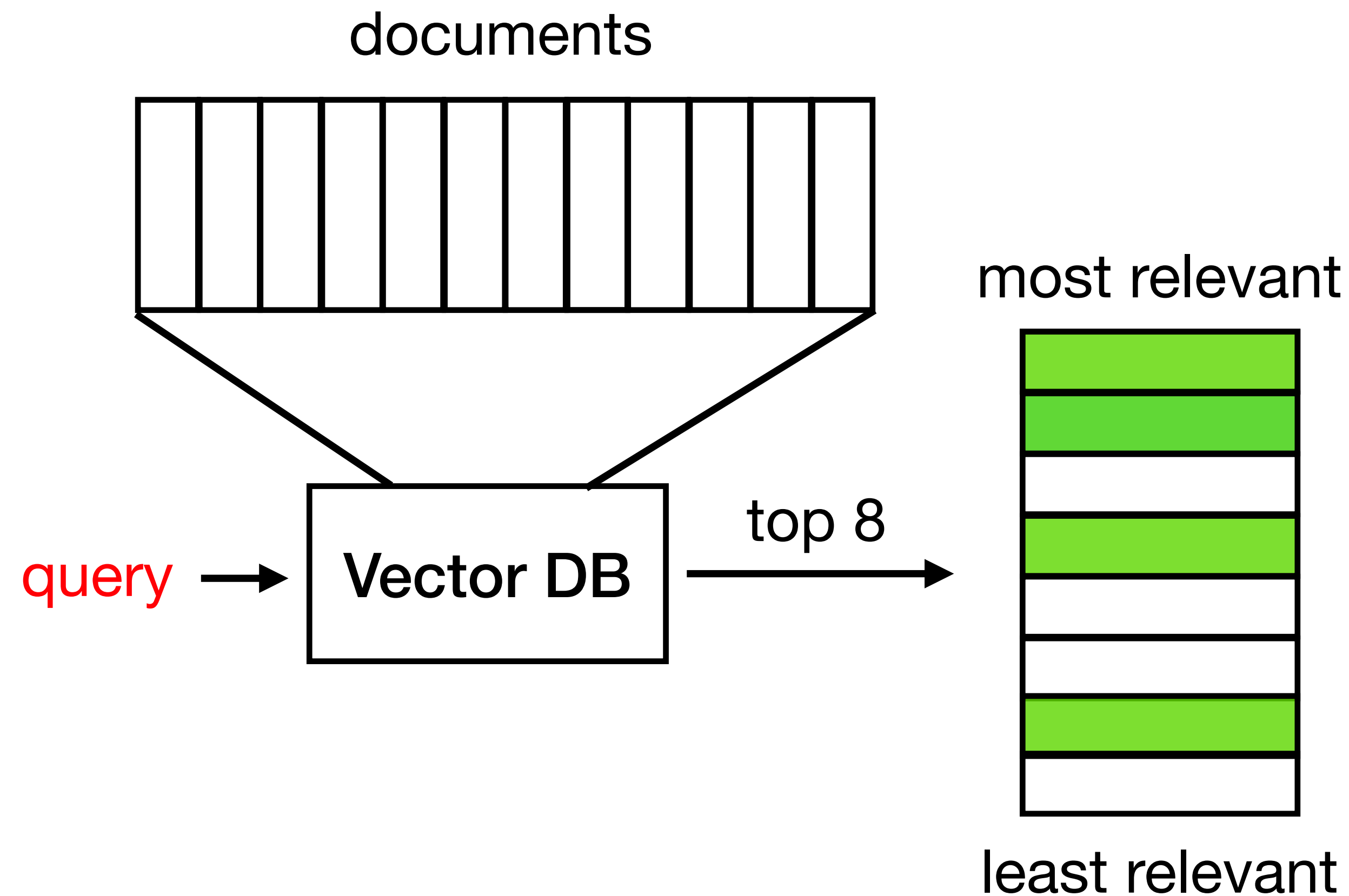
# Parent chunks

- Break the document into large chunks (parents)
- Then split each parent chunk into smaller chunks (children)
- When retrieving documents, calculate similarities with children, but retrieve parents
- This way we don't lose information when compressing the text and more relevant text gets into the LLM context



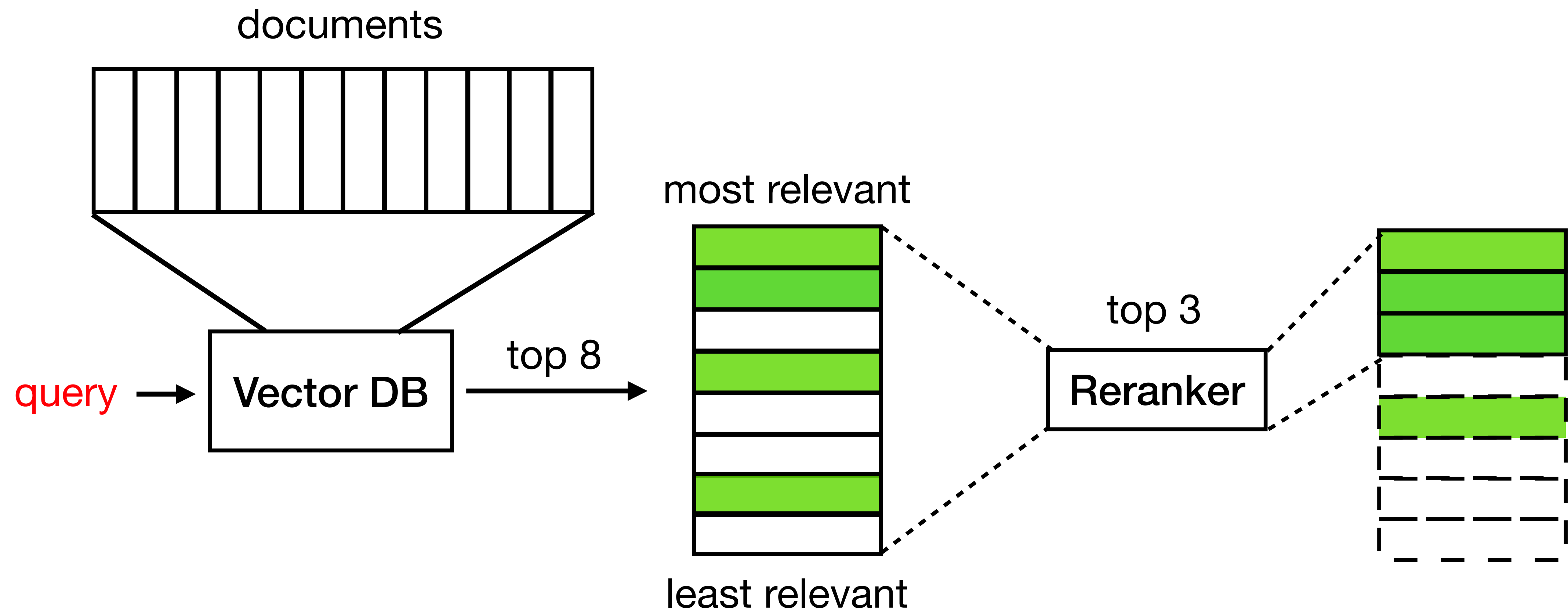
# Re-ranking

When text is compressed into embedding, information is lost  
Because of this, ranking is suboptimal.



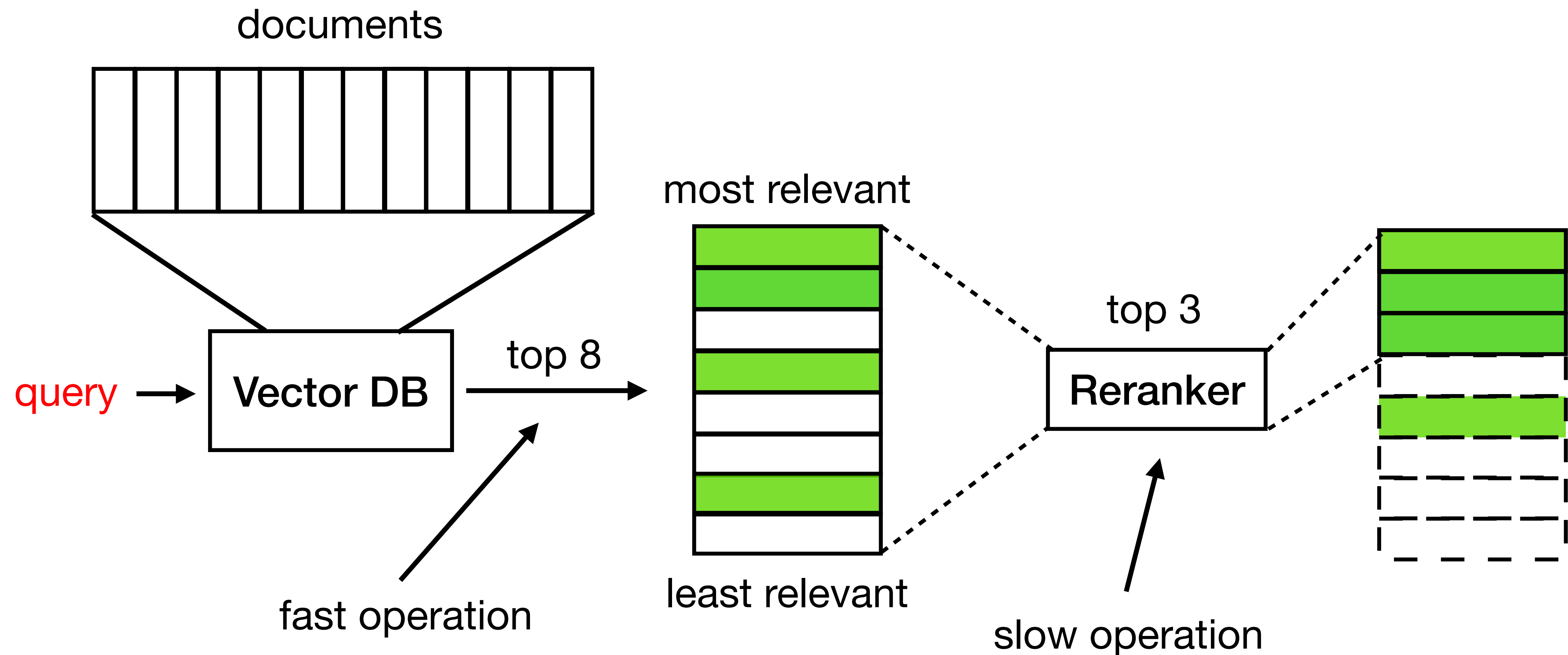
# Re-ranking

When text is compressed into embedding, information is lost  
Because of this, ranking is suboptimal.



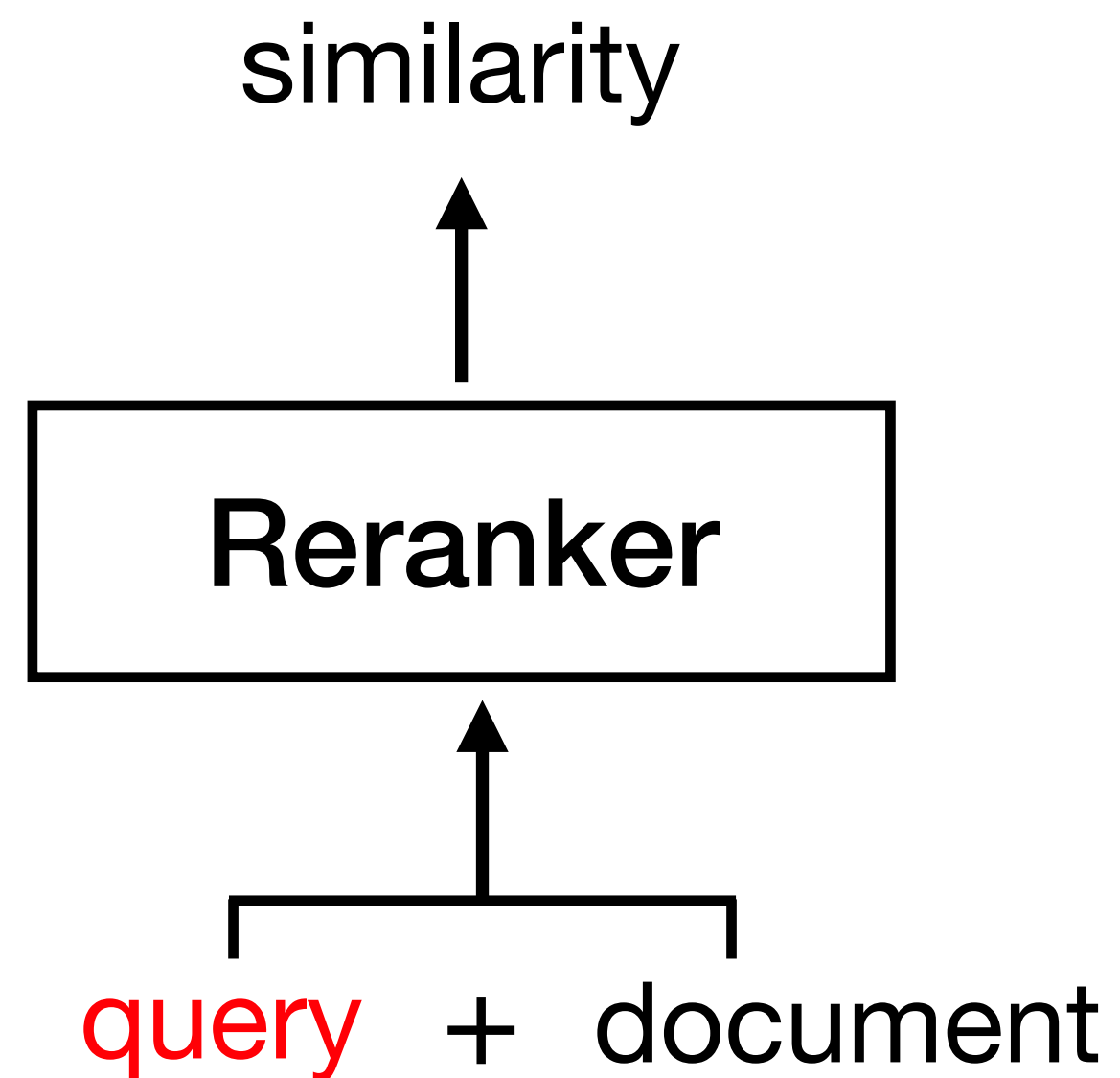
# Re-ranking

When text is compressed into embedding, information is lost  
Because of this, ranking is suboptimal.



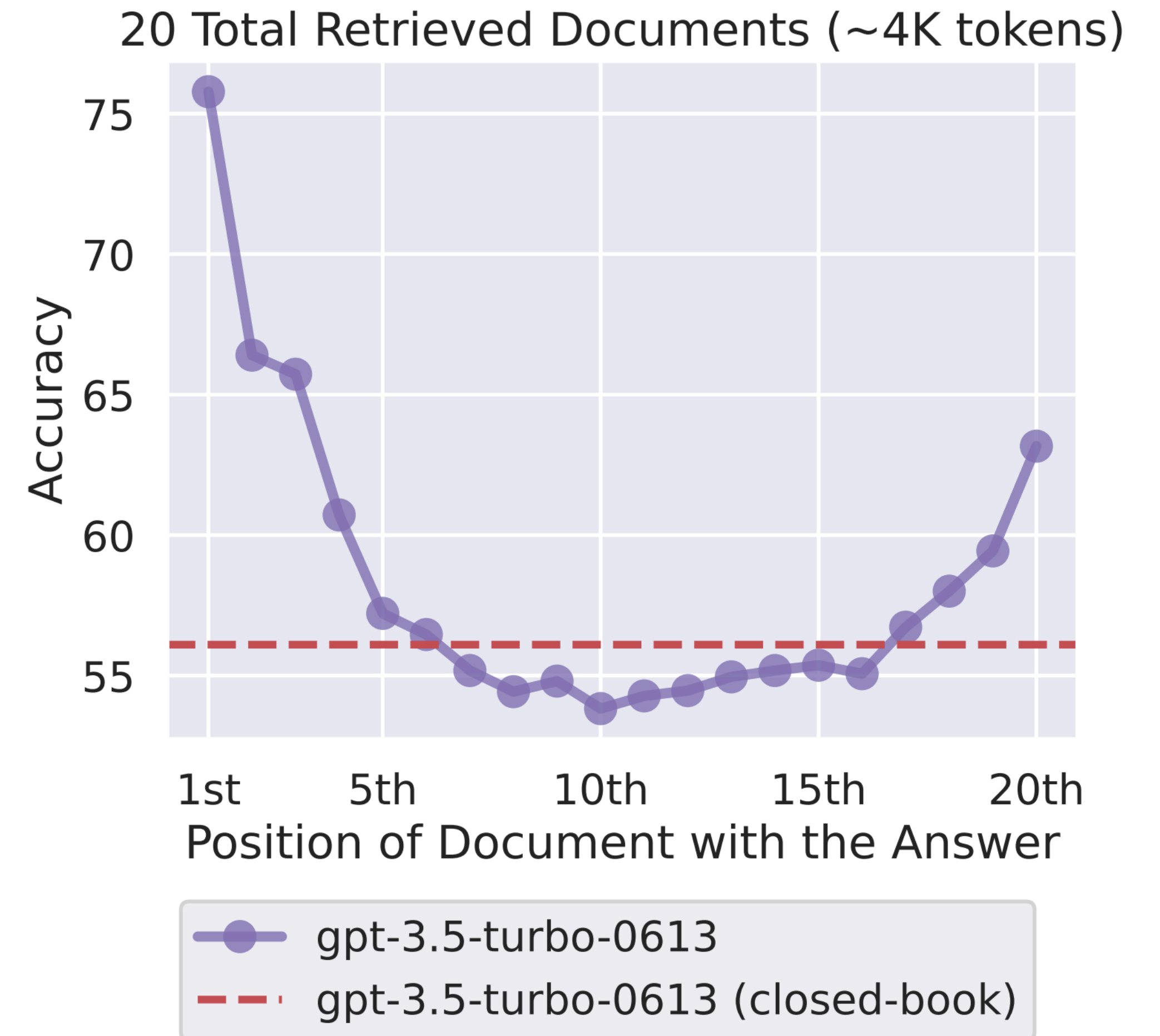
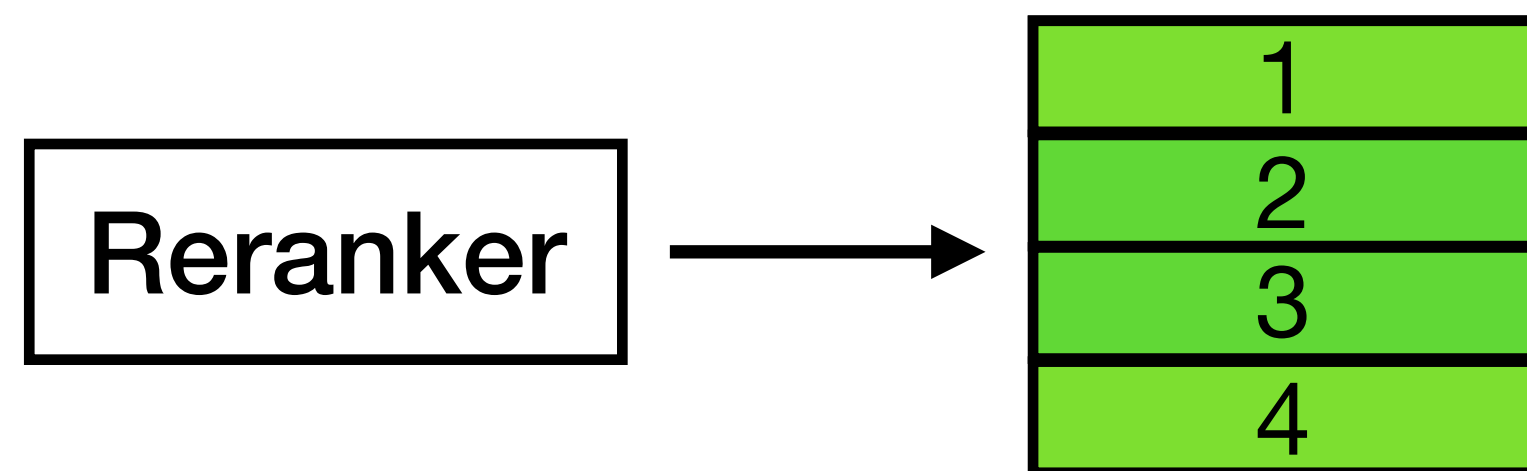
# Reranker

**Reranker** is a Transformer model that evaluates the similarity of two texts  
For example, BERT, which was trained on the Next Sentence Prediction task



# Order of documents in a context

Arrange in rank order is a bad idea!

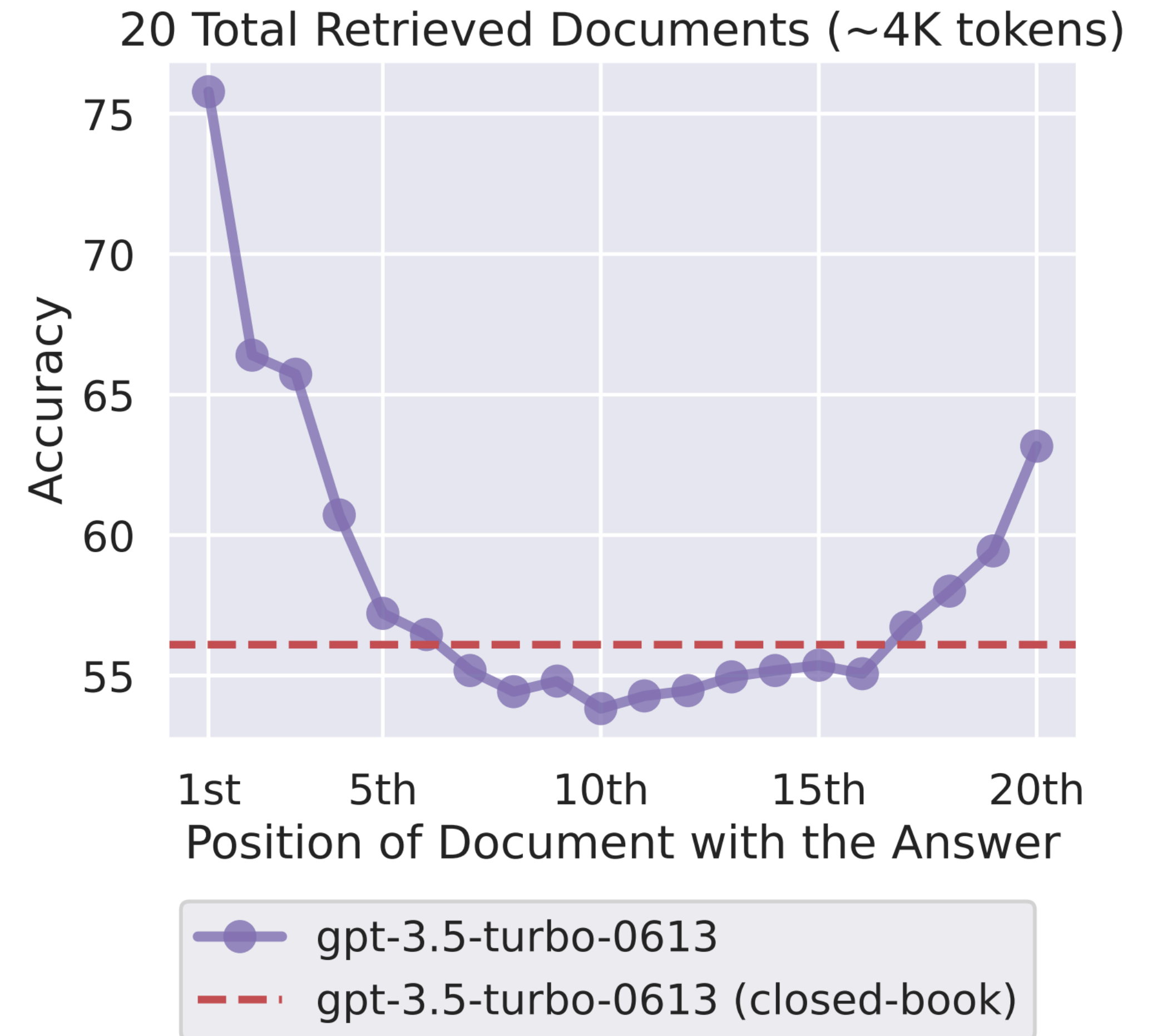
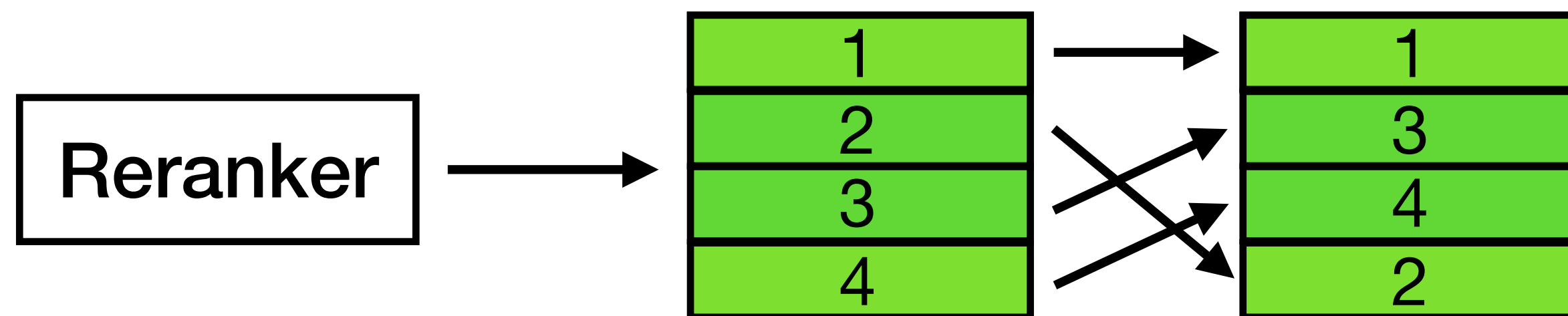




# Order of documents in a context

Arrange in rank order is a bad idea!

The most important information should be located at the beginning and the end of a context



# RAG quality measurement

In quality assessment, all components are tested separately:

- Quality of retrieving relevant chunks of text
- Quality of text generation

# RAG quality measurement

In quality assessment, all components are tested separately:

- Quality of retrieving relevant chunks of text
- Quality of text generation

From RAG system we want:

- Sustainability to irrelevant information in context
- Ability to refuse to answer if context information is insufficient
- Ability to aggregate information from different sources
- Ability to identify irrelevant information in context

# RAG quality measurement

In quality assessment, all components are tested separately:

- Quality of retrieving relevant chunks of text
- Quality of text generation

From RAG system we want:

- Sustainability to irrelevant information in context
- Ability to refuse to answer if context information is insufficient
- Ability to aggregate information from different sources
- Ability to identify irrelevant information in context

All metrics use LLMs

# Quality of retrieval

- **Retrieval Precision** – the proportion of texts matching the query

$$\text{Retrieval Precision} = \frac{\text{Number of relevant texts selected}}{\text{Number of selected texts}}$$

- **Retrieval Recall** – the proportion of correctly selected texts out of all texts matching the query

$$\text{Retrieval Recall} = \frac{\text{Number of relevant texts selected}}{\text{Number of relevant texts}}$$

- Ranking metrics: mAP, MRR

# Quality of text generation

- **Answer Relevancy** – how relevant the answer is to the query

$$\text{Answer Relevancy} = \frac{\text{Number of relevant statements}}{\text{Number of statements}}$$

- **Faithfulness** – the correctness of LLM statements

$$\text{Faithfulness} = \frac{\text{Number of correct statements}}{\text{Number of statements}}$$

# Use of parallel datasets

It is possible to use labelled datasets with correct answers

- SQuAD (Stanford Question Answering Dataset)

Benchmarks:

- CRAG - tasks with 3 difficulty levels with the most diverse questions
- FreshQA - 600 questions divided into 4 categories
- Natural Questions - 7.842 questions based on wikipedia

# RAG python tools

## Collection of LLMs



## Vector databases



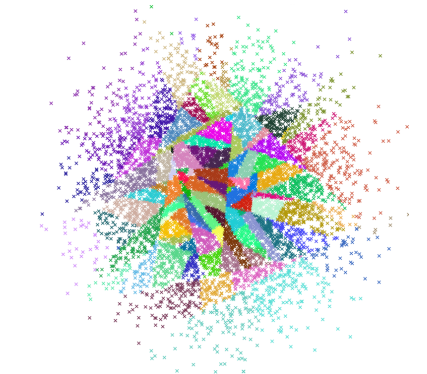
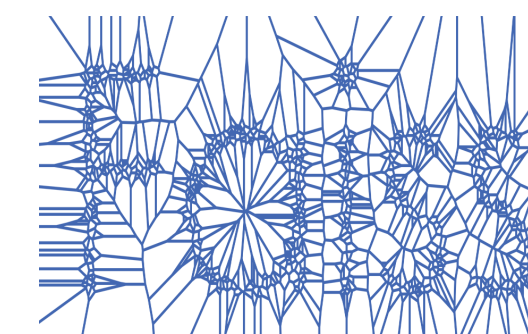
## Database text search



## Libraries for RAG integration



## Nearest vectors search



Annoy