Weaviate

# Vector databases:
## The what, why & how

**JP Hwang**
Educator

# Agenda

- **What is a vector database?**
  - What are vectors?
  - Differences vs “traditional” DBs
- **Why use a vector database?**
- **How do vector DBs work?**
- **Demos**

# Demo: A vector DB-driven app

# Vector databases

# ~~Vector~~ databases

# Databases

**Store data:**

- **For** products, customers, financial...
- **In** text, images, videos...

# Databases

**Store data:**

- **For** products, customers, financial…
- **In** text, images, videos…

**Typically allow:**

- Data management (**c**reate, **r**ead, **u**pdate, **d**elete)
- Search & (fast) retrieval

# Databases

**Store data:**

- **For** products, customers, financial…
- **In** text, images, videos…

**Typically allow:**

- Data management (**c**reate, **r**ead, **u**pdate, **d**elete)
- Search & (fast) retrieval

**At scale** (millions / billions of objects)

# Databases

## Established technology

- SQL / Relational DBs: ~50 years

# Databases

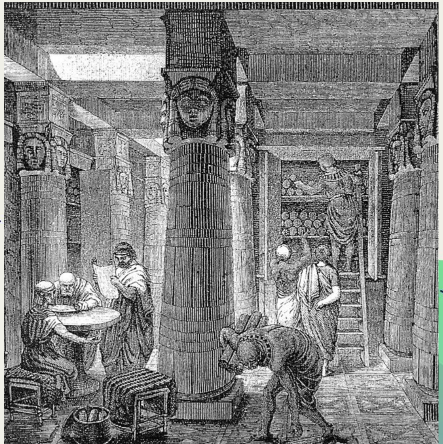
## Established technology

- SQL / Relational DBs: ~50 years
- Library catalogues: rudimentary databases

# Databases

**Established technology**

- SQL / Relational DBs: ~50 years
- Library catalogues: rudimentary databases
    - Library of Alexandria (~300BCE)



# Databases

**Established technology**

- SQL / Relational DBs: ~50 years
- Library catalogues: rudimentary databases
- Solved technology?

# Databases

**Established technology**

- SQL / Relational DBs: ~50 years
- Library catalogues: rudimentary databases
- Solved technology? **No**
    - **New types of DBs** with new features / focusses

# Databases

**Established technology**

- SQL / Relational DBs: ~50 years
- Library catalogues: rudimentary databases
- Solved technology? No
  - New types of DBs with new features / focusses
  - **Why? It's a hard problem!**

# Key challenge:

# **Search speed**

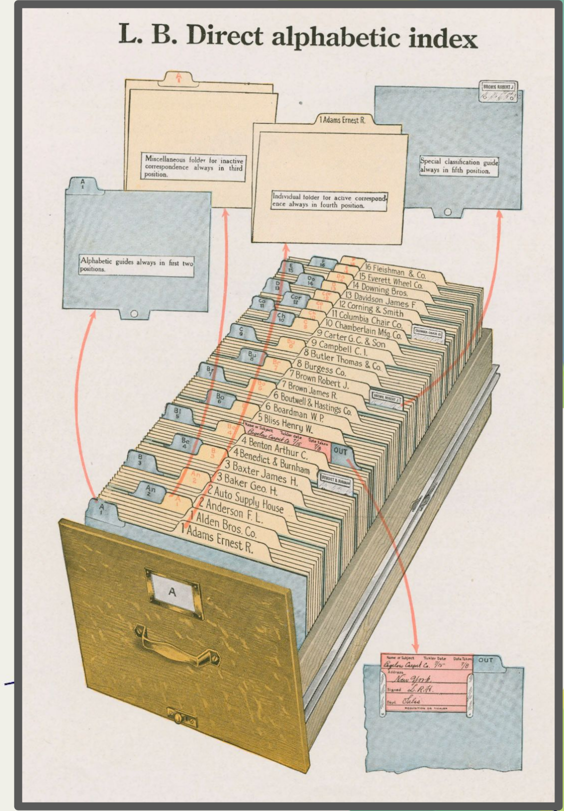
# How long will this take?

```sql
SELECT * FROM Hotels
WHERE Country='Netherlands';
```

# Indexes

Used by databases for speed

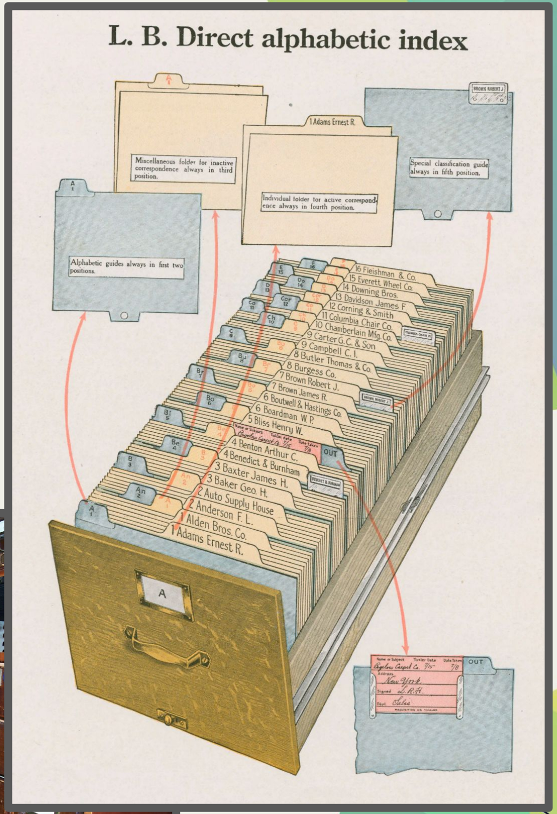
- e.g. Library catalogues



# Indexes

In databases:

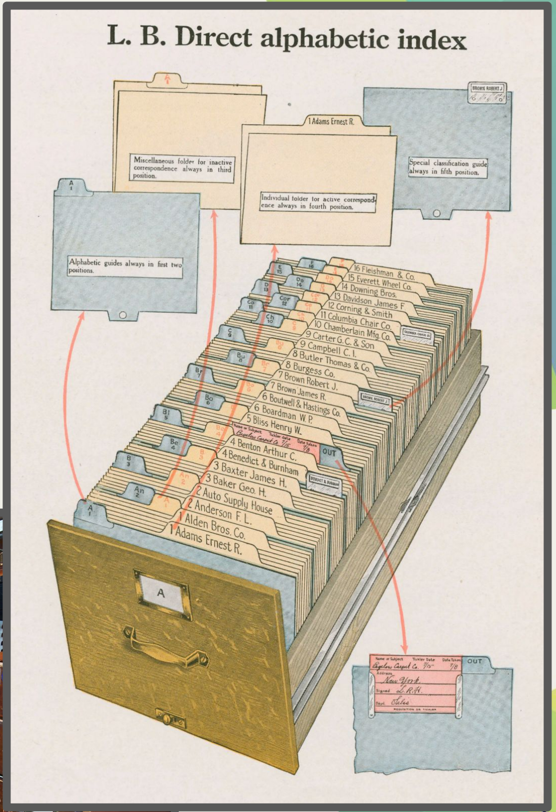
- **Catalogue data**
- **Speed up search & filtering**



# Indexes

Most common type: “**inverted index**”

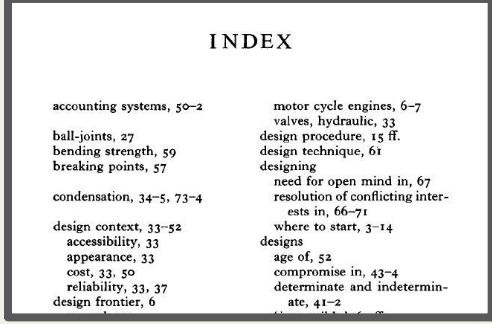
- Catalogued by **keywords**




# Indexes

Most common type: “**inverted index**”

- Catalogued by **keywords**
  - Actually, “tokens”



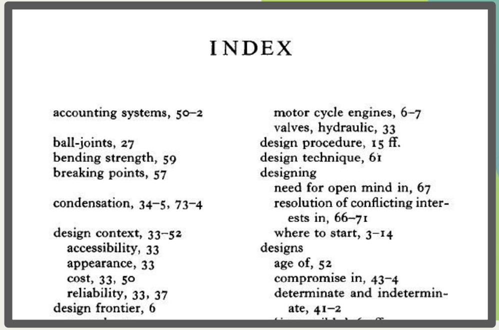
INDEX



# Indexes

Most common type: “**inverted index**”

- Catalogued by **keywords**
  - Actually, “tokens”
- Allows **fast** keyword searches



INDEX



(Another) Key challenge:

**Search quality**

# What are the limitations of this approach?

How would you cover:

- Typos?
- Synonyms?
- Translations?

```sql
SELECT * FROM Hotels
WHERE Country='Netherlands';
```


THERE MUST BE
A BETTER WAY
memegenerator.net

# Vector databases

# A vector is a set of numbers

Like

```
[1, 0]
```

or

```
[0.513, 0.155, 0.983, 0.001, 0.932]
```

or

```
    [0.0009420722, 0.020158706, -0.03939992,
-0.025480185, 0.018441677, 0.0023035712,
-0.012281344, -0.025270471, -0.056622636, …]
```

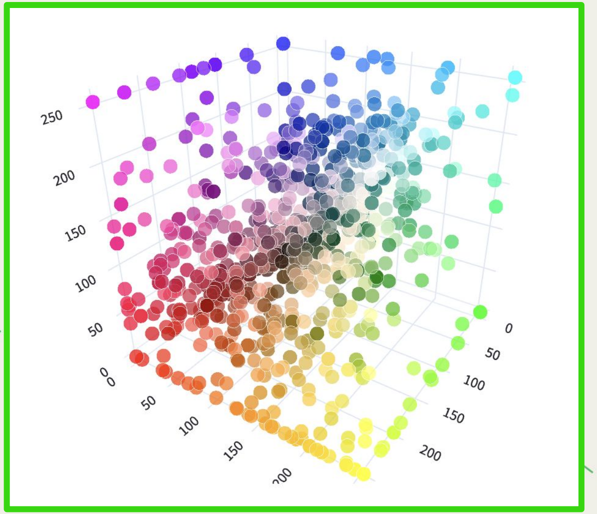
In vector DBs, they're used to represent meaning.

# Numbers represent meaning?

Yes! Here's an example.

RGB *numbers* represent *colors*, like:
(255, 0, 0) = red
(80, 200, 120) = emerald.

Each number is a *dial*
for (red, green, blue) ness.



# Now extend this concept…

To hundreds, or even thousands of these dials.

That’s how vectors represent meaning.

# Example

- "Three people rescued off Australian coast after yacht damaged by multiple shark attacks"

# Vector

[-0.01670855, -0.02290458, 0.01024679, ..., -0.01840662, -0.01677336, 0.00040852]

# Examples

- "Three people rescued off Australian coast after yacht damaged by multiple shark attacks"
- "Tourists taking selfies and feeding dingoes blamed for rise in K'gari attacks"
- "Sam Kerr: Chelsea striker and Matildas captain named runner-up in Uefa's player of the year awards"
- "'She's brilliant': Mary Earps inspires girls to pick up goalkeeper gloves"

# Vectors

[-0.01670855, -0.02290458, 0.01024679, ..., -0.01840662, -0.01677336, 0.00040852]

[-0.01062017, 0.01388064, 0.02811302, ..., -0.01565292, 0.00282415, -0.01064047]

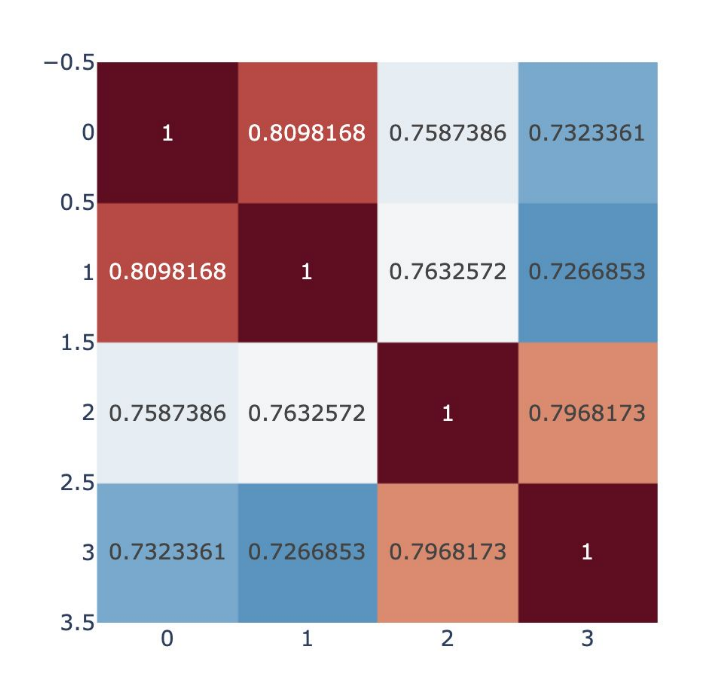
[-0.00067538, -0.00483041, 0.02590884, ..., -0.01845455, -0.01025612, -0.00987435]

[-0.03254206, 0.00462641, 0.00465651, ..., 0.01225011, -0.00032469, -0.01669922]

# Examples

- "Three people rescued off Australian coast after yacht damaged by multiple shark attacks"
- "Tourists taking selfies and feeding dingoes blamed for rise in K’gari attacks"
- "Sam Kerr: Chelsea striker and Matildas captain named runner-up in Uefa’s player of the year awards"
- "‘She’s brilliant’: Mary Earps inspires girls to pick up goalkeeper gloves"

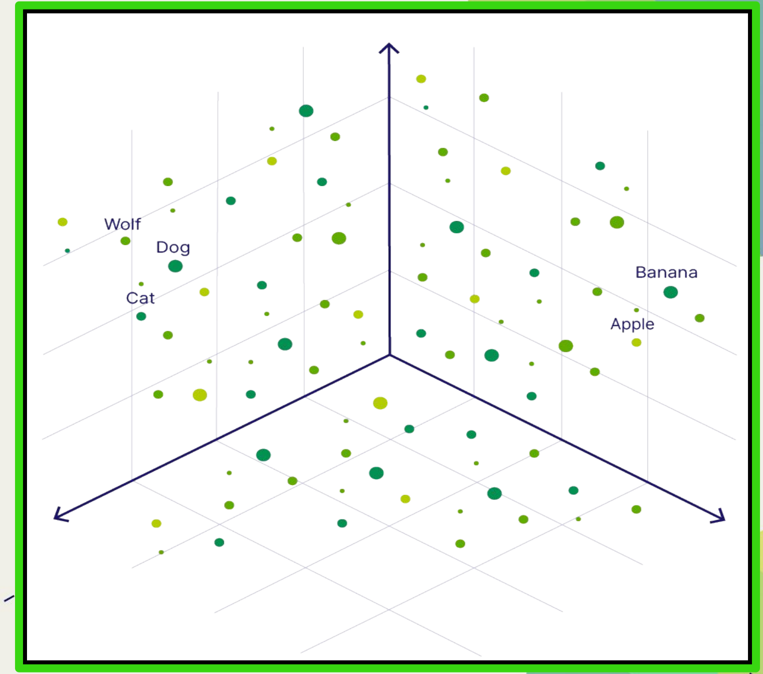
# Similarity matrix



# What is a **Vector?**

**Vector embeddings**:

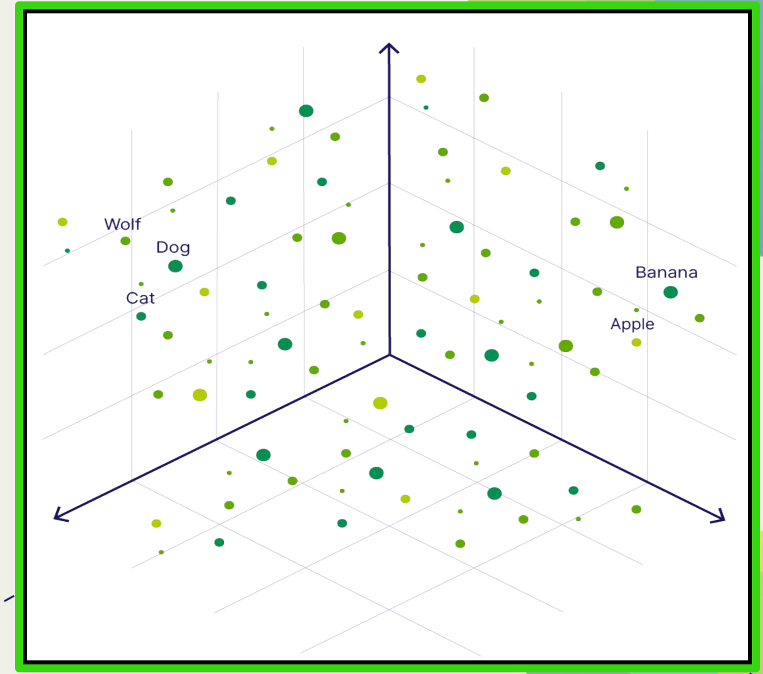
- Text organised by vectors ⇒
- Text with similar meaning are next to each other



# What is a **Vector?**

**Vector embeddings**:

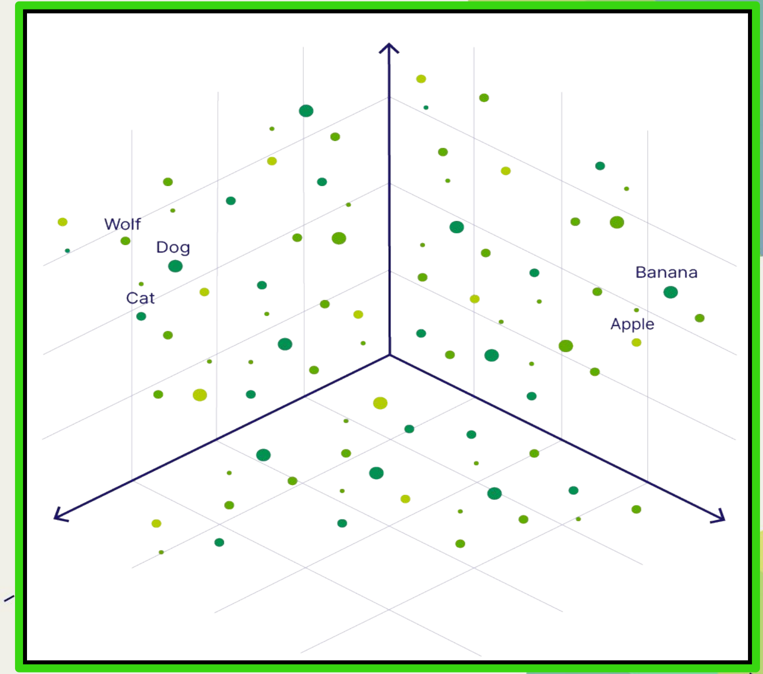
- Text organised by vectors ⇒
- Text with similar meaning are next to each other
- **“AI” (deep learning) models** convert data to vectors



# What is a **Vector?**

**Vector embeddings**:

- Text organised by vectors ⇒
- Text with similar meaning are next to each other
- **“AI” (deep learning) models** convert data to vectors
- Enables **vector search**



# This the key to modern language models

Vector databases like Weaviate uses vectors to:

- Represent the meaning of objects
- Search for similar objects
- Transform objects

And the same core technology is used in LLMs

Vector databases have

**a vector index**

# **Vector** databases

**Vector index**:

- Organised catalogue of data (**index**)
- By meaning (**vector / vector embedding**)

# **Vector** databases

**Vector index**:

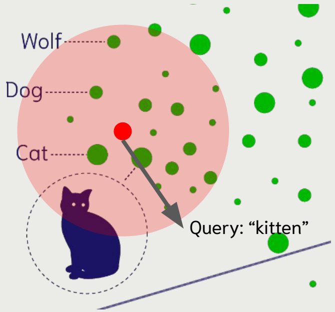
- Organised catalogue of data (**index**)
- By meaning (**vector / vector embedding**)
- Allows fast **similarity searches**

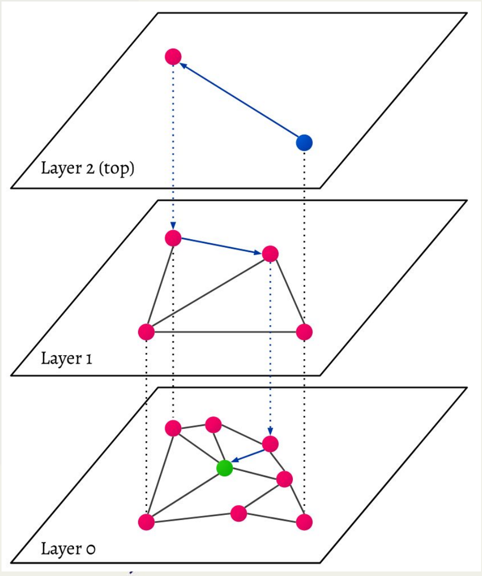
# ANN indexing

Enables scalable search up to billions of vectors.

# ANN indexing

A way to scale search up to billions of vectors.



# Vector index ≠ Vector database

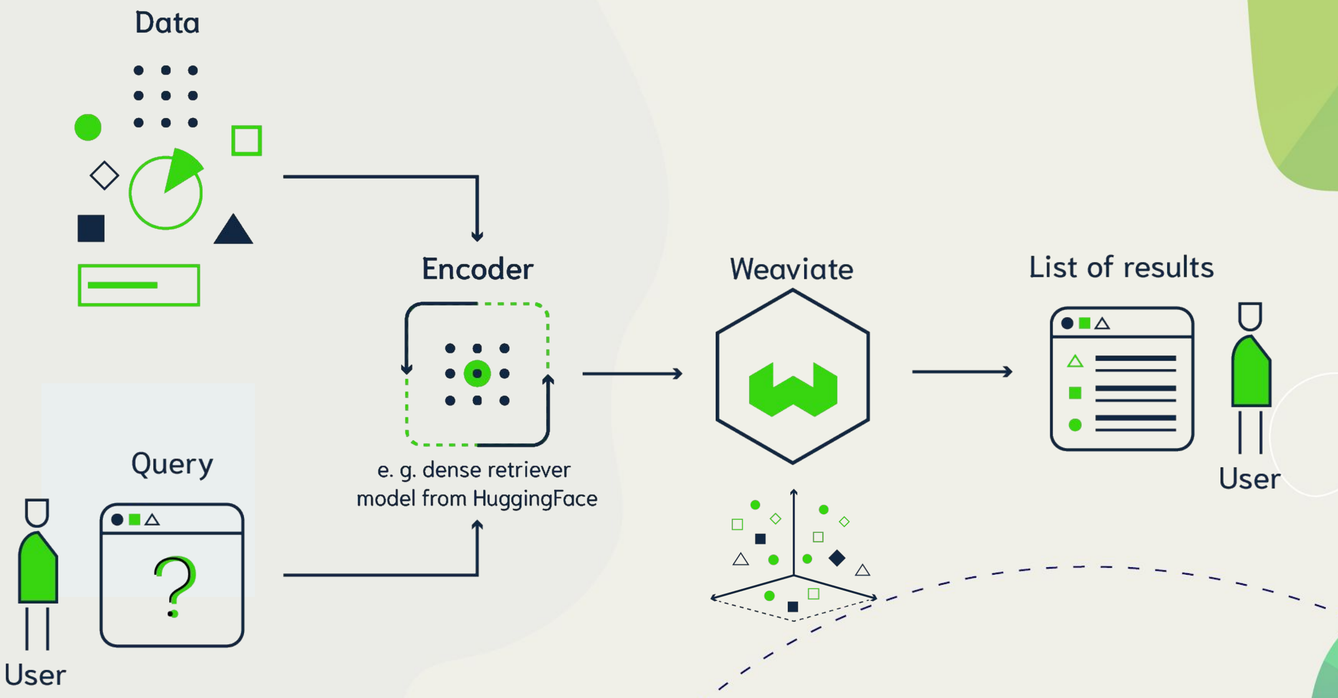
A database **houses and manages** collections of data.

An index **improves** the speed of data retrieval.

(A catalog is not a library.)

# Searches

# Typical Vector DB workflow



# Searches

**Weaviate can perform**

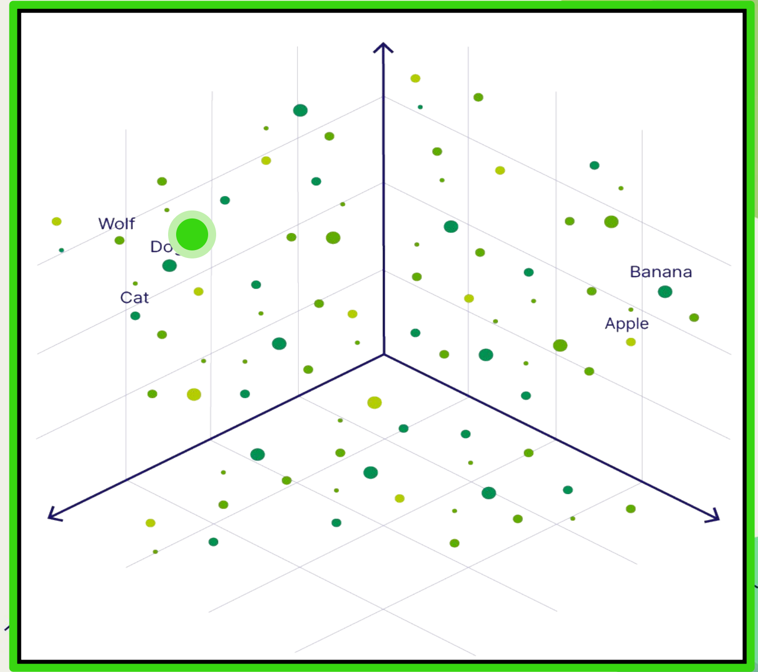
- Vector searches
- Keyword searches
- Hybrid searches
- (+ Filtering)

# Searches

**Weaviate can perform**

- **Vector** searches
- Keyword searches
- Hybrid searches
- (+ Filtering)

<u>Most similar</u> to “puppy”

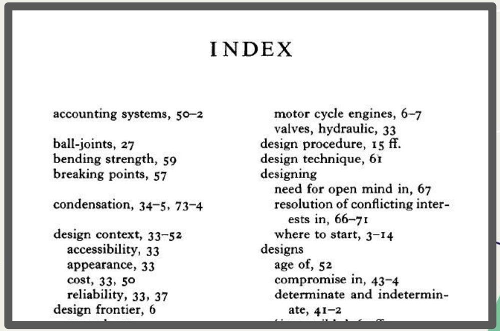


# Searches

**Weaviate can perform**

- Vector searches
- **Keyword** searches
- Hybrid searches
- (+ Filtering)

E.g. Products where “vacuum” <u>most relevant</u>



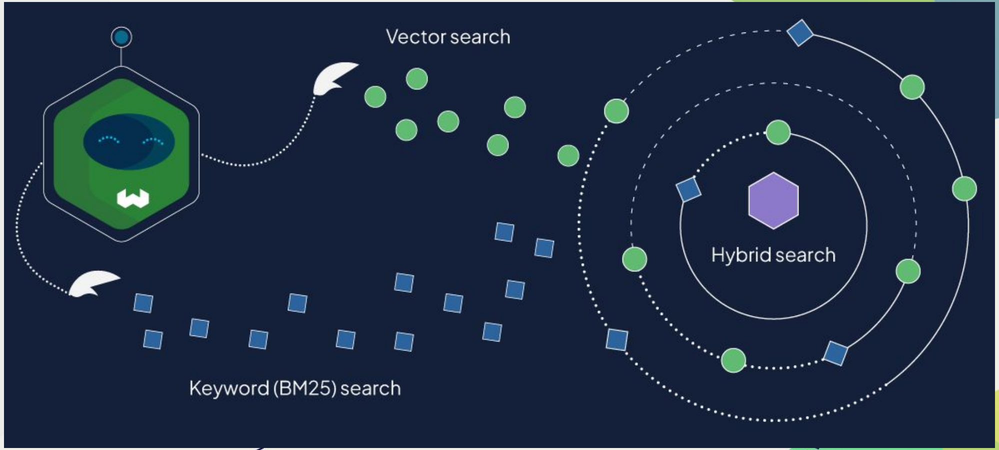
INDEX

# Searches

**Weaviate can perform**

- Vector searches
- Keyword searches
- **Hybrid** searches
- (+ Filtering)

Hybrid search for “vacuum”



# Searches

**Weaviate can perform**

- Vector searches
- Keyword searches
- Hybrid searches
- **(+ Filtering)**

E.g. <u>Only look</u> in products made in the U.K.

# Searches

**Weaviate can perform**

- **Vector** searches
- Keyword searches
- Hybrid searches
- **(+ Filtering)**

E.g. Only look in products made in the U.K.

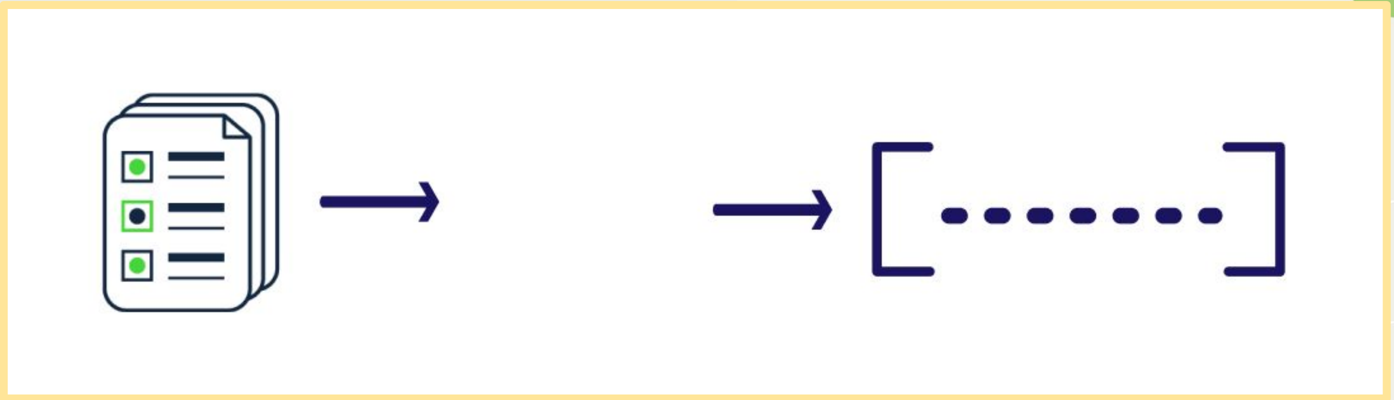
Most similar to "automatic vacuum"

# Demo: Searches

# Where do embeddings come from?

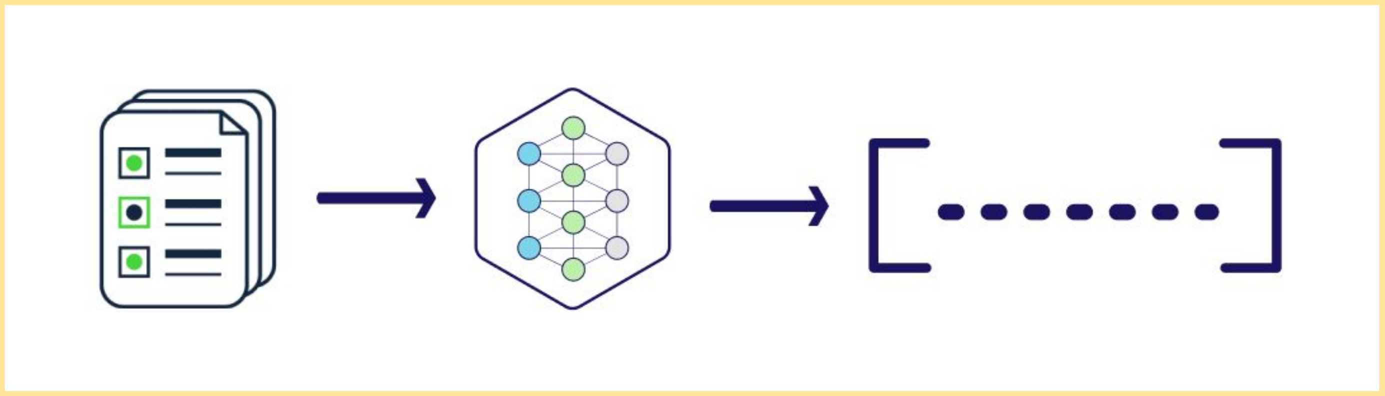
# Objects → Embeddings

How does this happen?

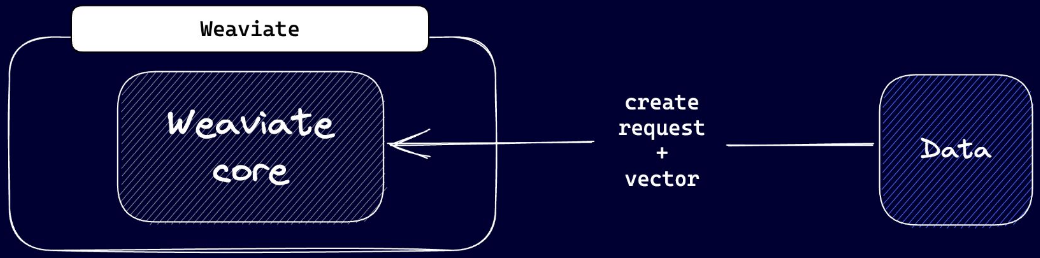


# Objects → Embeddings

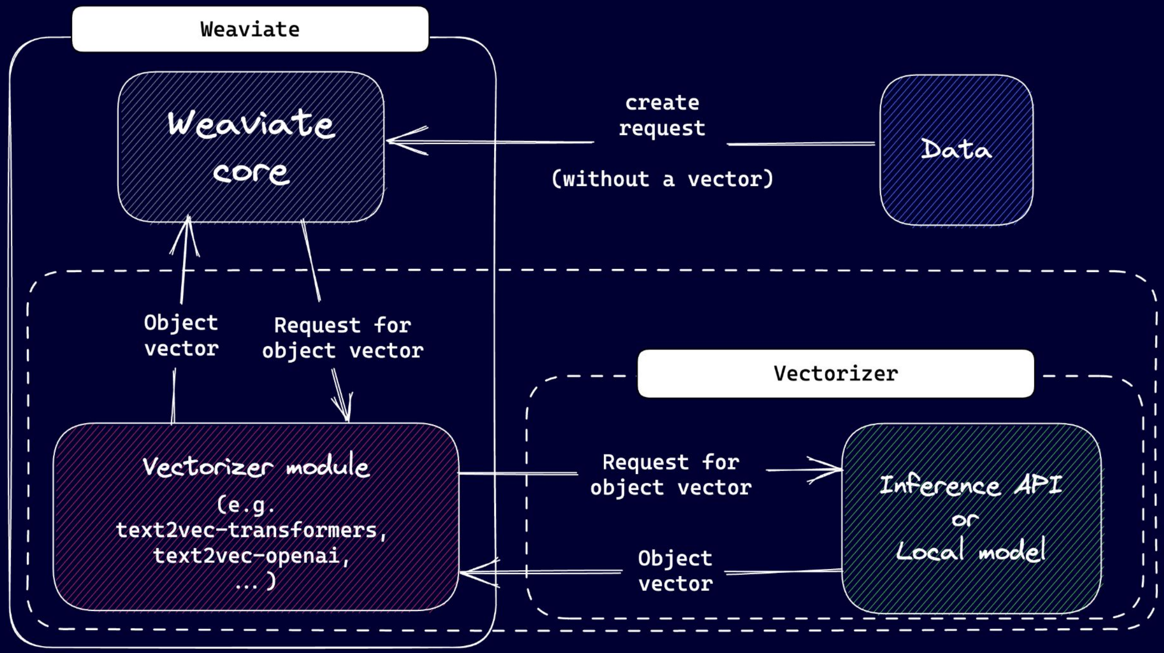
Via deep learning models



# Conceptual diagram - object import process



# Conceptual diagram - object import process



# Objects → Embeddings

**Vectorizer models** translate data into vectors.

**Hundreds** of models are available:

- **Proprietary** models @ Cohere, OpenAI, Google, AWS, etc.
- **Open-source** models from Hugging Face

# Objects → Embeddings

**Vectorizer models** translate data into vectors.

**Hundreds** of models are available:

- **Proprietary** models @ Cohere, OpenAI, Google, AWS, etc.
- **Open-source** models from Hugging Face

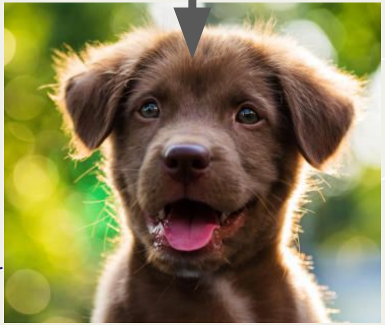
**Why so many?**

# Which two are the *most* similar?

Puppy

Chiot

A cute, furry, brown dog

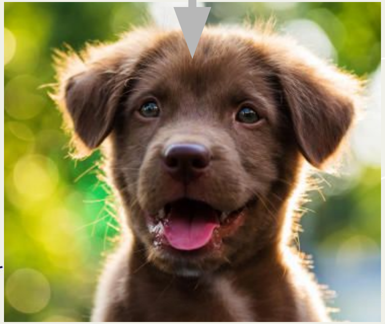


# The only English descriptions

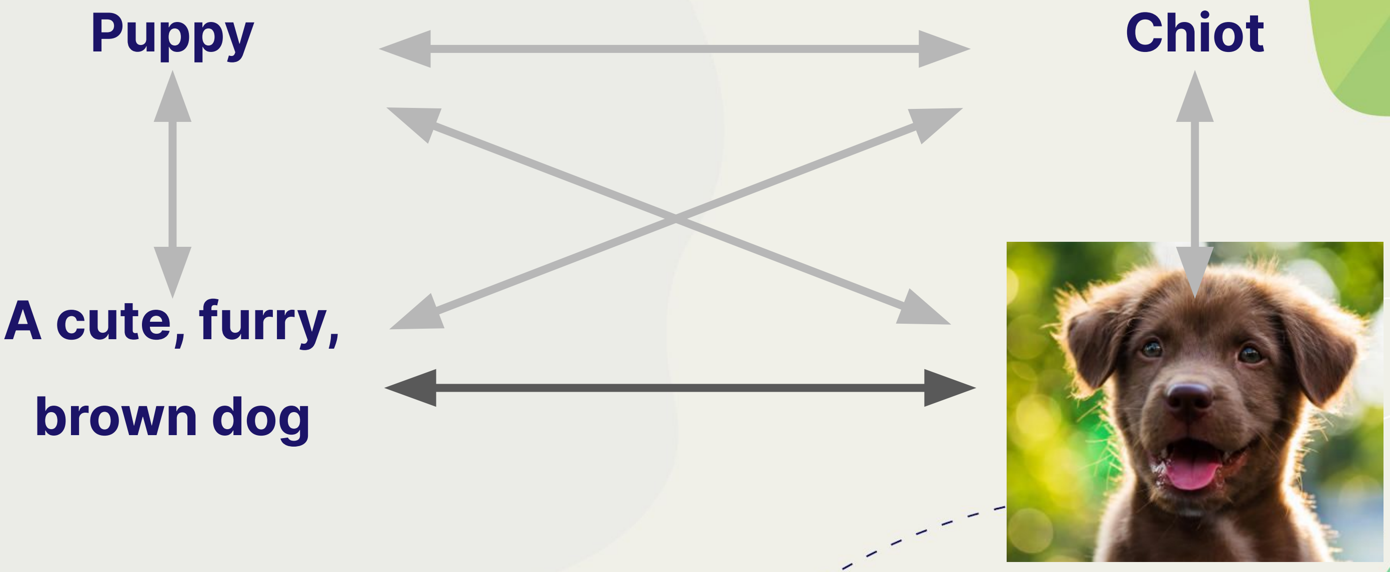
Puppy

Chiot

A cute, furry, brown dog



# Best matching image to text

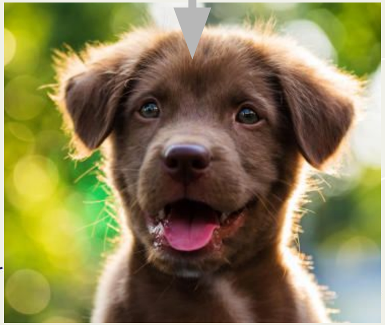


# If you speak English & French

Puppy

Chiot

A cute, furry, brown dog



# Which two are the *most* similar?

Is determined by the **vectorizer model**

# (Some) Significant models

- Word2Vec (2013)
- GloVe (Global Vectors for Word Representation) (2014)
- FastText (2016)
- ELMo (Embeddings from Language Models) (2018)
- BERT (Bidirectional Encoder Representations from Transformers) (2018)
- RoBERTa (Robustly Optimized BERT Pretraining Approach) (2019)
- DistilBERT (2019)
- T5 (Text-To-Text Transfer Transformer) (2019)
- CLIP (Contrastive Language–Image Pretraining) (2020)
- DeBERTa (Decoding-enhanced BERT with Disentangled Attention) (2020)
- Sentence-BERT (SBERT) (2020)
- Ada-002 (2021)
- Embed-multilingual-v2.0 (2022)
- ImageBind (2023)

# Significant Models

## Word2Vec (2023)

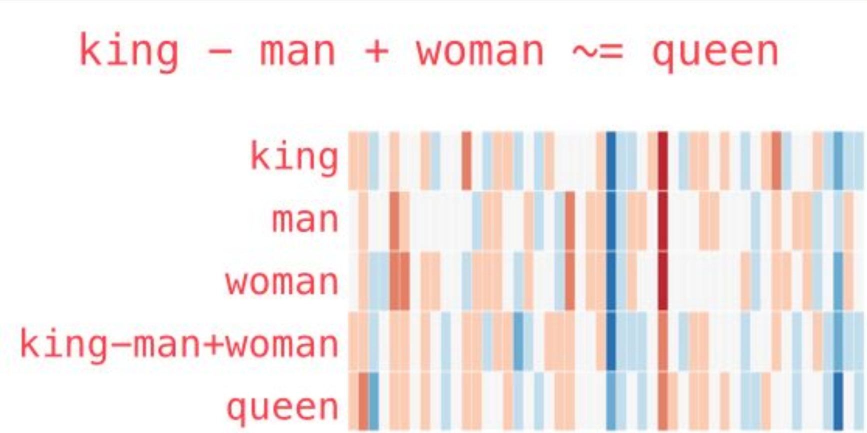
- Convert individual words into vectors.
- Popularised vector maths:

# Significant Models

**Word2Vec (2013)**

- Convert individual words into vectors.
- Popularised vector maths:

(Figure: Jay Alammar blog)



# Significant Models

**Bert (2018)**

- One of the first successful “transformer” architecture implementations.
- Context-aware embeddings

# Significant Models

**Bert (2018)**

- One of the first successful “transformer” architecture implementations.
- Context-aware embeddings
  - (River) **bank ≠ bank** (heist)

# Significant Models

## CLIP (2020)

- A multi-modal model (image & text)

# Significant Models

## CLIP (2020)

- A multi-modal model (image & text)
  - Search images with text & vice versa

**A cute, furry, brown dog**

# Significant Models

## Cohere multilingual (2022)

- A multilingual model

# Significant Models

**Cohere multilingual (2022)**

- A multilingual model
  - ~100 languages supported

**Puppy** ⟷ **Chiot**

# (Some) Significant models

- **Word2Vec** (2013)
- GloVe (Global Vectors for Word Representation) (2014)
- FastText (2016)
- ELMo (Embeddings from Language Models) (2018)
- **BERT** (Bidirectional Encoder Representations from Transformers) (2018)
- RoBERTa (Robustly Optimized BERT Pretraining Approach) (2019)
- DistilBERT (2019)
- T5 (Text-To-Text Transfer Transformer) (2019)
- **CLIP** (Contrastive Language–Image Pretraining) (2020)
- DeBERTa (Decoding-enhanced BERT with Disentangled Attention) (2020)
- Sentence-BERT (SBERT) (2020)
- **Ada-002** (2021)
- **Embed-multilingual-v2.0** (2022)
- **ImageBind** (2023)

# Why vector searches

"To get good results, you shouldn't need to know any magic words. With semantic search, you don't."

- David Haney, David Gibson

*Stackoverflow Blog*

# Vector searches

**Are great because they can:**

- Be **robust** to synonyms, word forms & typos
  - Space vs. intergalactic
  - Puppy vs puppies vs pupppies

# Vector searches

**Are great because they can:**

- Be **robust** to synonyms, word forms & typos
- Work **across languages**
  - Puppies vs chiot vs 강아지

# Vector searches

**Are great because they can:**

- Be **robust** to synonyms, word forms & typos
- Work **across languages**
- Work **across modalities**
  - Puppies vs chiot vs 강아지 vs

# Vector searches

Are **powered** by **models** that **generate vectors**:

- **Robustly** to synonyms, word forms & typos
- **Across languages**
- **Across modalities**
    - Puppies vs chiot vs 강아지 vs 

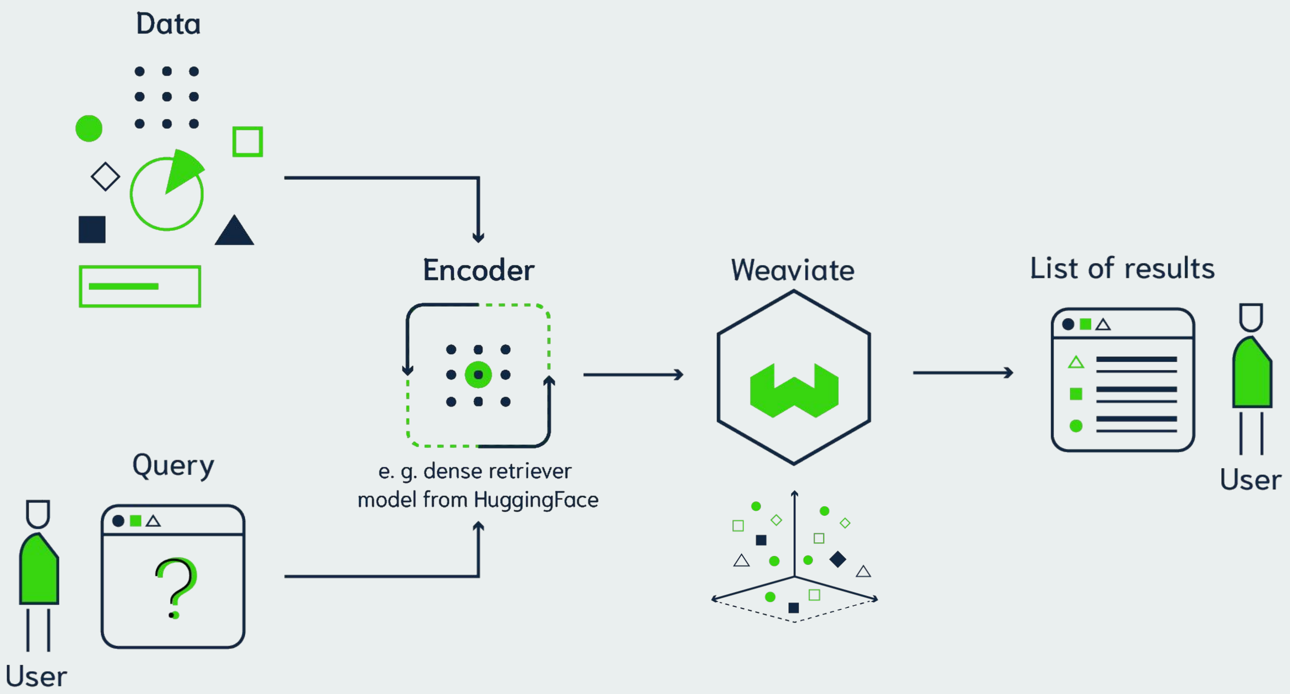
# Vector searches

Are **powered** by **models** that **generate vectors**:

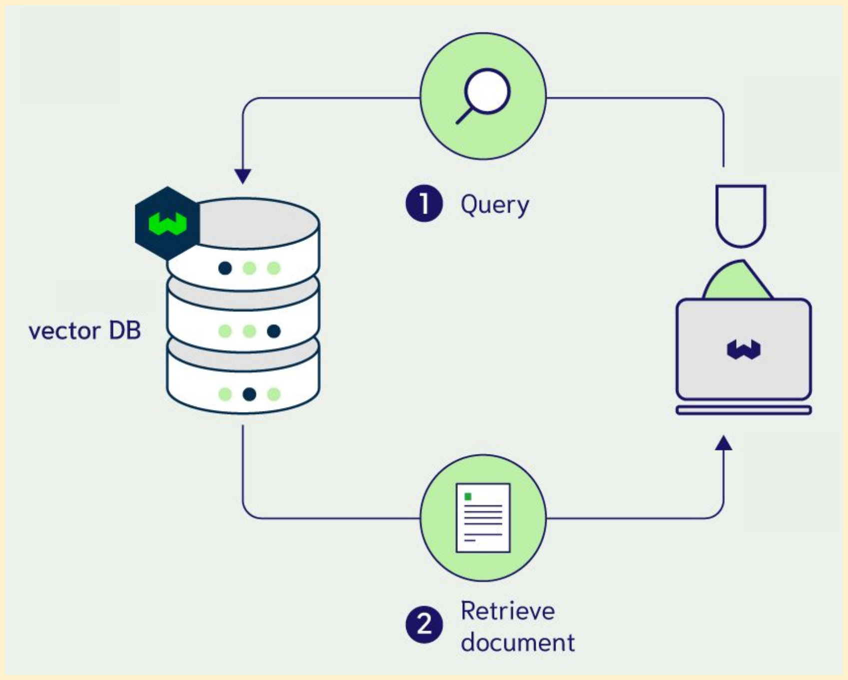
This is why vector DBs are “**AI-native**”.

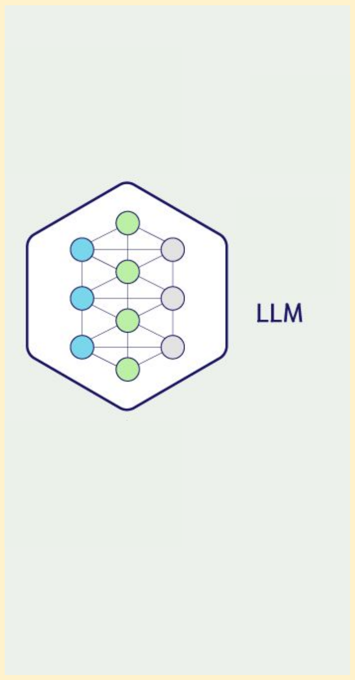
# Retrieval augmented generation

# A vector search pipeline

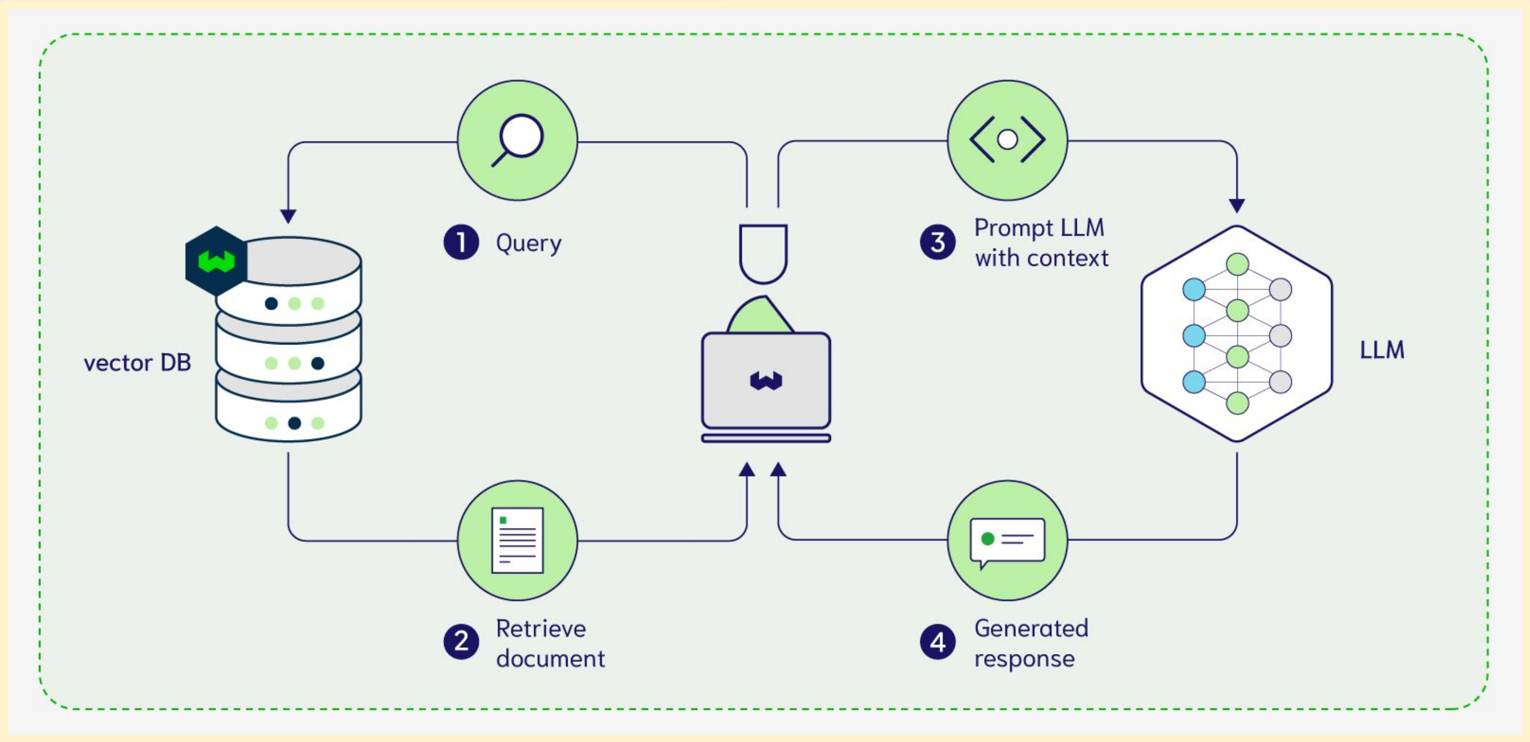


# Vector search + LLM



# Retrieval augmented generation



# Retrieval augmented generation

- Retrieves data
- Sends the data+prompt to an LLM
- Serves data + LLM response

(Some of the served outputs are not in the DB!)

# RAG workflow

- Extract text from source data
- Chunk text
- Add it to Weaviate
- Query with prompt

# RAG workflow

- Extract text

```python
def download_and_parse_pdf(pdf_url: str) → str:
    """
    Get the text from a PDF and parse it
    :param pdf_url:
    :return:
    """
    # Send a GET request to the URL
    response = requests.get(pdf_url)

    # Create a file-like object from the content of the response
    pdf_file = BytesIO(response.content)
    pdf_reader = PdfReader(pdf_file)

    # Initialize a string to store the text content
    pdf_text = ""
    n_pages = len(pdf_reader.pages)

    # Iterate through the pages and extract the text
    for page_num in range(n_pages):
        page = pdf_reader.pages[page_num]
        pdf_text += "\n" + page.extract_text()
    return pdf_text
```

# RAG workflow

- Chunk text

```python
def chunk_text_by_num_words(source_text: str, max_chunk_words: int = 200) → List[str]:
    """
    Chunk text input into a list of strings, using a number of words
    :param source_text: Input string to be chunked
    :param max_chunk_words: Maximum length of chunk, in words
    :return: return a list of words
    """
    sep = " "

    source_text = source_text.strip()
    word_list = source_text.split(sep)
    chunks_list = list()

    n_chunks = ((len(word_list) - 1) // max_chunk_words) + 1
    for i in range(n_chunks):
        window_words = word_list[
                       max(max_chunk_words * i - overlap_words, 0):
                       max_chunk_words * (i + 1)
                       ]
        chunks_list.append(sep.join(window_words))
    return chunks_list
```

# RAG workflow

- Import chunks

```python
def import_chunks(
        self,
        chunks: List[str], source_object_data: SourceData,
        category: str = '',
        chunk_number_offset: int = 0):
    """
    Import text chunks via batch import process
    :param chunks:
    :param source_object_data:
    :param category: Category of the source object (e.g. arxiv)
    :param chunk_number_offset:
    :return:
    """
    counter = 0
    self.client.batch.configure(batch_size=100)
    with self.client.batch as batch:
        for i, chunk_text in enumerate(chunks):
            chunk_object = ChunkData(
                ...
            )
            batch.add_data_object(
                class_name=self.chunk_class,
                data_object=asdict(chunk_object),
                uuid=generate_uuid5(asdict(chunk_object))
            )
            counter += 1
    return counter
```

# RAG workflow

- Perform queries

```python
def generate_on_search(
        client: Client,
        class_name: str, class_properties: List[str],
        prompt: str, search_query: str,
        object_path: str, limit: int = N_RAG_CHUNKS
):
    """
    Perform a search and then a generative task on those search results
    For specific tasks that should be paired with a search (e.g. what does video AA say about topic
BB?)
    """
    where_filter = {
        "path": ["source_path"],
        "operator": "Equal",
        "valueText": object_path
    }
    response = (
        client.query
        .get(class_name, class_properties)
        .with_where(where_filter)
        .with_near_text({'concepts': [search_query]})
        .with_generate(grouped_task=prompt)
        .with_limit(limit)
        .with_sort({
            'path': ['chunk_number'],
            'order': 'asc'
        })
        .do()
    )
    return parse_generative_response(response, class_name)
```

# Search vs RAG workflow

A good search is key for a good RAG system.

# How to get started with vector db / RAG

- Weaviate Cloud Services sandbox (free)
- Quickstart document
- Choose an API vectorizer
  - (e.g. Cohere / OpenAI / HuggingFace)
- Choose a LLM (e.g. Cohere / OpenAI)
- Have fun!