RECAP

MODERN APPLICATIONS NEED:

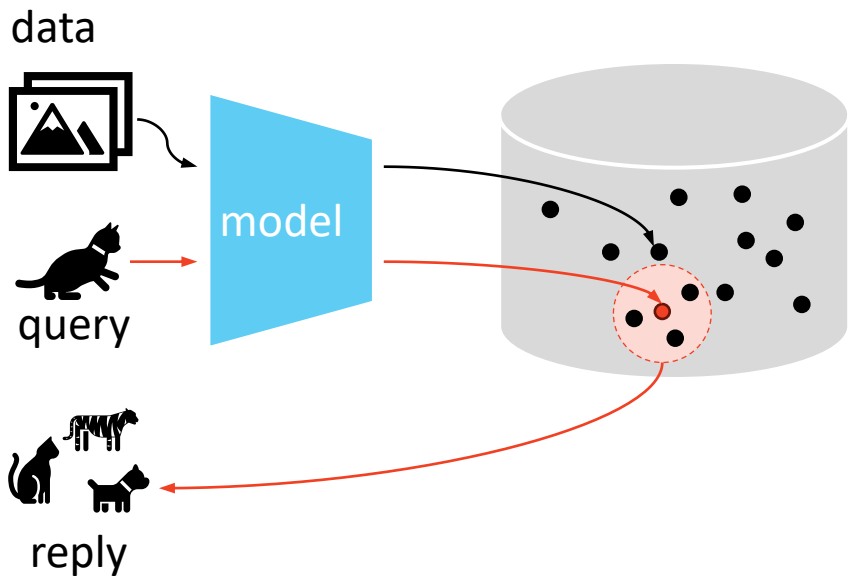
VECTOR DATABASE = VECTOR STORAGE + SEMANTIC SEARCH

Insert lots of vectors:

1. Create embedding vector *x*
2. Store *x* in specialized DB
   (with associated attributes)

Query data quickly:

3. Embed query as vector *q*
4. Find nearest neighbours to *q*

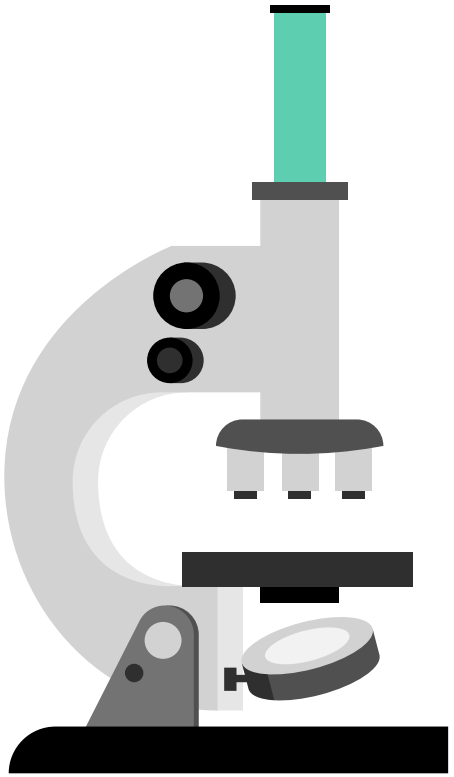


RDBMS  *VDBMS*

- Different requirements → different design!
- Main differences:
  - **Records** vs **vectors**
  - **SQL** vs **NN**
  - **ACID** vs **eventual**
- Key challenges:
  - Large vectors
  - No structure
  - Slow updates

	Traditional DB (RDBMS)	Vector DB (VDBMS)
Data	Records	Vectors
Queries	Relational algebra	Nearest neighbors + filtering
Advanced query features	Join, group, FK, cursors	None of those*
Updates	To part of record To multiple records	On whole vector Insert/delete/replace
Consistency	Strong + transactions	Eventual, tunable
Index updates	Fast	Slow
Storage	Row/column based, LSM	Vector is opaque blob
Hardware , scaling cost	Uniform , moderate	Diverse , expensive (GPUs)
Architecture	More monolithic	More disaggregated

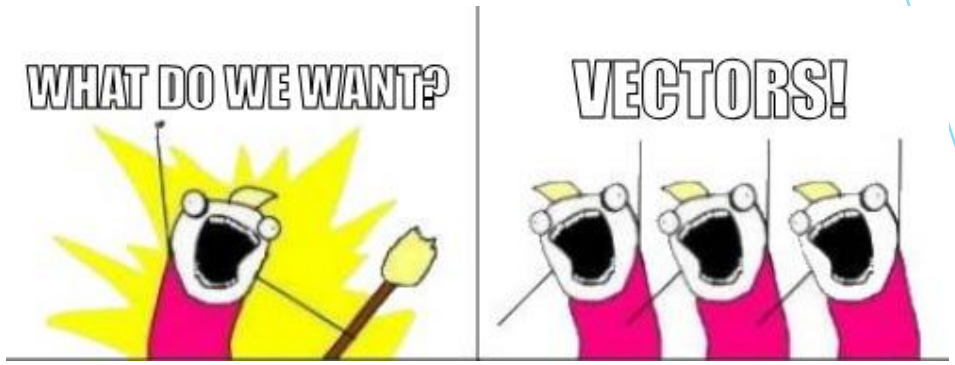
II. QUERYING



QUERYING

- **Have**: query vector *q*
- **Want**: “some” set of vectors “near” *q*

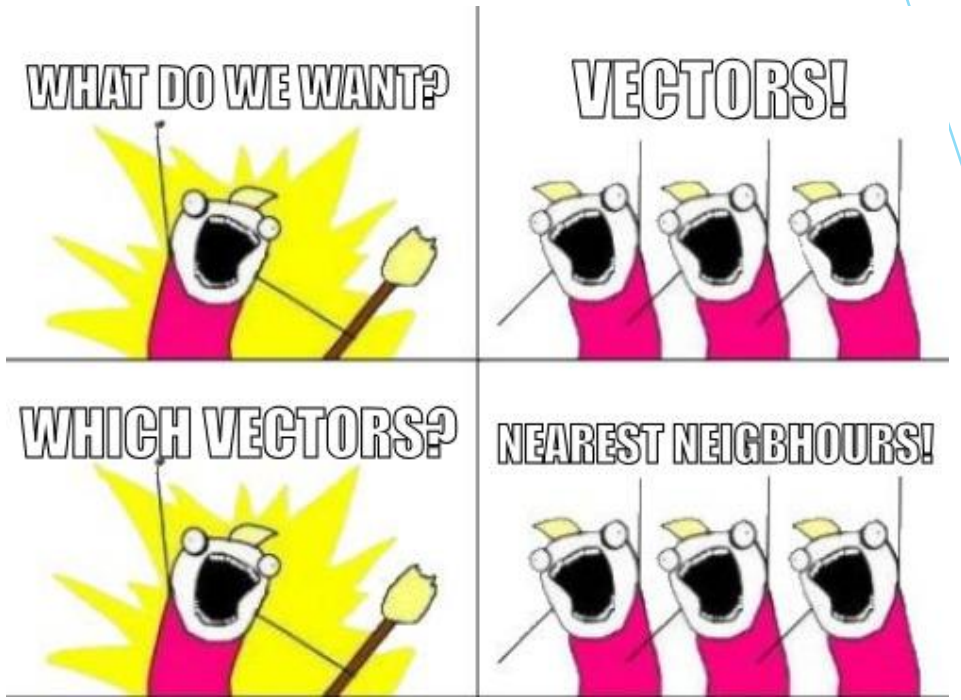
- What is “near”? → similarity **score**



QUERYING

- **Have**: query vector *q*
- **Want**: “some” set of vectors “near” *q*

- What is “near”? → similarity **score**
- Which “some”? → various **query types**



QUERYING

- **Have**: query vector *q*
- **Want**: “some” set of vectors “near” *q*

- What is “near”? → similarity **score**
- Which “some”? → various **query types**
- When? → as fast as possible

DISTANCE/SIMILARITY SCORES

- Given vectors v, q → number expressing similarity/distance of v to q
- $d(v,q) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

Name	Function $d(v,q)$	Range
Euclidean distance	$\|v-q\| = \sqrt{\sum(v_i - q_i)^2}$	$\mathbb{R}^+$
Inner product* , MIPS	$v \cdot q = \mathrm{v^T q} = \sum v_i q_i$	$\mathbb{R}$
Cosine similarity*	$\frac{v \cdot q}{\|v\|\|q\|}$	$[-1,1]$

Used by most embedding models
Implemented in most VDBMS

MIPS = Maximum Inner Product Search

These are similarity functions: high $d(v,q)$ is "closer" → must "reverse" score if we want distance

CHANGE SIMILARITY TO DISTANCE

Ways to "reverse" similarity score d?

- Simple inversion: just use $-d$
  - Problems: $d(v, v) \neq 0$ and $d(q, v)$ can be negative
- For cosine distance: $\frac{1-d}{2}$
- For inner product:
  - Use –d for graph-based index [Morozov and Babenko, NeurIPS'18]
  - Add one dimension to vectors, then use Euclidean/cosine [Bachrach, RecSys'14]

    $\hat{v} = \left[v, \sqrt{\phi^2 - \|v\|^2}\right], \hat{q} = [q, 0] \Rightarrow \operatorname{argmin} \|\hat{v} - \hat{q}\| = \operatorname{argmax} v \cdot q$
  - To limit range, transform: $\frac{1}{1+e^d}$ (numerically problematic, not often done)

DISTANCE/SIMILARITY SCORES

- Given vectors v, q → number expressing similarity/distance of v to q
- $d(v,q) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

Name	**Function** $d(v,q)$	**Range**
Euclidean distance	$\|v-q\| = \sqrt{\sum(v_i - q_i)^2}$	$\mathbb{R}^+$
Inner product* , MIPS†	$v \cdot q = \mathrm{v^T q} = \sum v_i q_i$	$\mathbb{R}$
Cosine similarity*	$\frac{v \cdot q}{\|v\|\|q\|}$	$[-1,1]$
Mahalanobis distance	$\sqrt{(v-q)^T M (q-\mathrm{q})}$	$\mathbb{R}$
Hamming distance	Num of $v_i \neq q_i$	$\mathbb{N}$
Manhattan distance	$\|v-q\|_1 = \sum\|q_i - v_i\|$	$\mathbb{R}^+$

Used by most embedding models
Implemented in most VDBMS

← Can use for **learned** score: learn M from data

* Similarity: high $d(v,q)$ is "closer" → must "reverse" score for distance

† MIPS = Maximum Inner Product Search

kNN QUERY – THE WORKHORSE

- Find k nearest neighbours (kNN)
- Denote:
  S = stored vectors
  q = query
  R = result set
- Find set of k vectors in S closest to q
- Generally: returned vectors sorted by distance from q
- Formally:
  return $\mathrm{R} = \{r_i\} \subseteq S$ such that $\forall v \in R\colon\ d(q,v) \leq \min_{x \in S \setminus R} d(q,x)$ and
  $|R| = k$ and
  $d(r_i, q) < d(r_{i+1}, q)$

In pgvector

```
SELECT * FROM items
ORDER BY vec <-> '[1,6.4,-2.1]'
LIMIT 5;
```

RANGE QUERY

- **Range**: get all vectors within range t
  - $R = \{v \in S \mid d(q, v) \leq t\}$

- Not as common but is used
  - Hard to determine t from application

In pgvector

```
SELECT * FROM items
WHERE vec <-> '[1,6.4,-2.1]' < 4;
```

PREDICATED QUERIES

- Other names: attribute filtering, hybrid queries, mixed queries.
- kNN/range query + predicate on associated attributes.
- Return nearest neighbours **that satisfy predicate**.
- Sounds simple?
- An active research problem!
  - ANNS indexes do not like predicates.
  - Challenging to implement quickly!
  - Several recent papers inlucding in 2024.

Example: find 3 nearest **green** products with **price under 100$** in Pinecone

```
index.query(
    namespace="products",
    vector=[0.81, 0.46, 0.41, 0.64, 0.11],
    filter={
        "price": {"$lt": 100},
        "color": {"$eq": "green"}
    },
    top_k=3,
    include_metadata=True
)
```

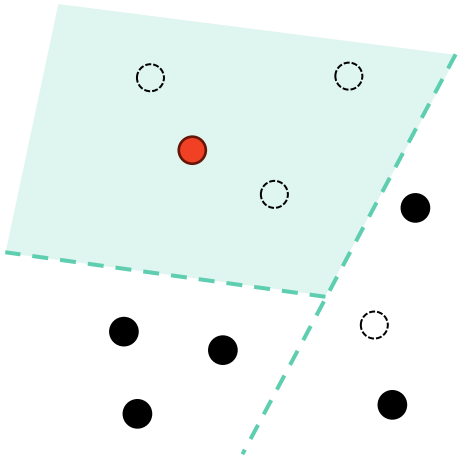
RUNNING PREDICATED QUERIES

- "Find 5 **nearest neighbours** with 50 < **price** < 100 and **color** is green or red"

ANNS index (clustering) | int index (B-tree) | String equality index (hash)

- **Prefiltering**: first filter price, color → then kNN?
  - **Cannot use ANNS index** (lost structure due to filter)
  - Filter very selective? → result is small → flat scan is fast.
  - Filter not selective? → result is huge → **flat scan too slow!**
- **Postfiltering**: first kNN → then price, color?
  - May result in **less than k vectors**, or even **zero**.

  → **Low accuracy**



RUNNING PREDICATED QUERIES

- "Find 5 **nearest neighbours** with 50 < **price** < 100 and **color** is green or red"

ANNS index (clustering) | int index (B-tree) | String equality index (hash)

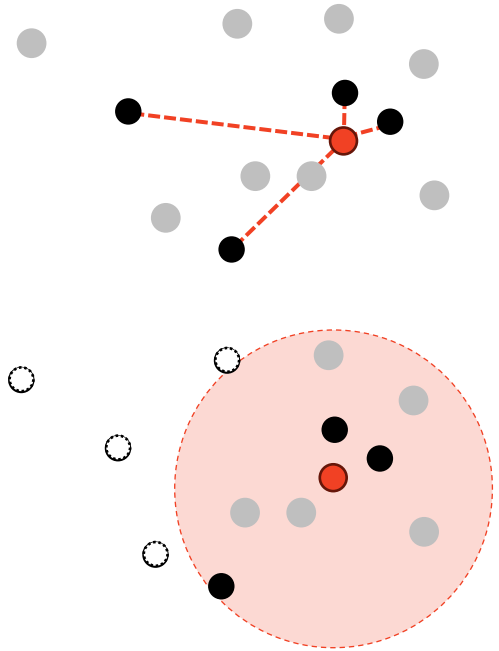
- **Prefiltering**: first filter price, color → then kNN?
  - **Cannot use ANNS index** (lost structure due to filter)
  - Filter very selective? → result is small → flat scan is fast.
  - Filter not selective? → result is huge → **flat scan too slow!**
- **Postfiltering**: first kNN → then price, color?
  - May result in **less than k vectors**, or even **zero**.

  → **Low accuracy**

ANNS index and attribute index **do not mix**!

THREE FILTERING STRATEGIES

1. **Prefilter** then scan:
   - Example: exhaustive scan
   - Good with highly selective predicate.
2. **Postfilter** with larger k
   - First kNN with $k' = f \cdot k$, $f > 1$
   - Then filter top k' to select top k
   - Good with low-selectivity, small k
3. **Single stage scan**
   - Combine index traversal with filtering
   - Holy grail!
   - We will see some papers



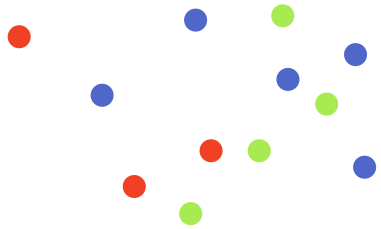
Choose using statistics + cost model

EXAMPLE IMPLEMENTATIONS

- **Block-first scan** (prefilter)**:**
  1. Build bitmap = predicate output for each vector
  2. Use bitmap during kNN
  - Milvus, AnalyticsDB-V
  - Problem: breaks graph-based index

  → Use attributes when building graph (Filtered-DiskANN)
- **Visit-first Scan** (single-stage):
  1. Start with neareast neighbour
  2. Incrementally add next neighbour that pass filter.
  3. Stop when have *k*.
  - Good for low-selectivity filter
  - Done by Timescale (pgvectorscale/StreamingDiskANN)

- **Pre-partition by attribute range**
  - Partition data based on attribute value.

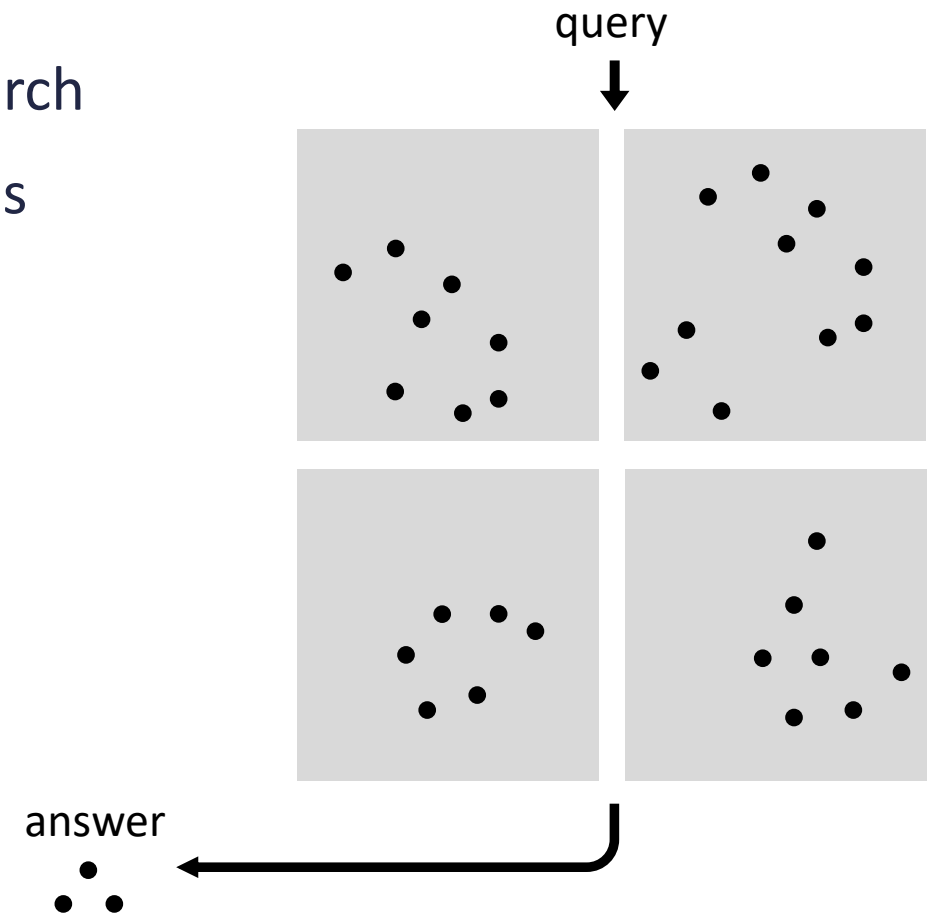


  - Query multiple partitions and combine.
  - Used in Milvus

Hard!
→ ANNS index search is not incremental (no backtracking).

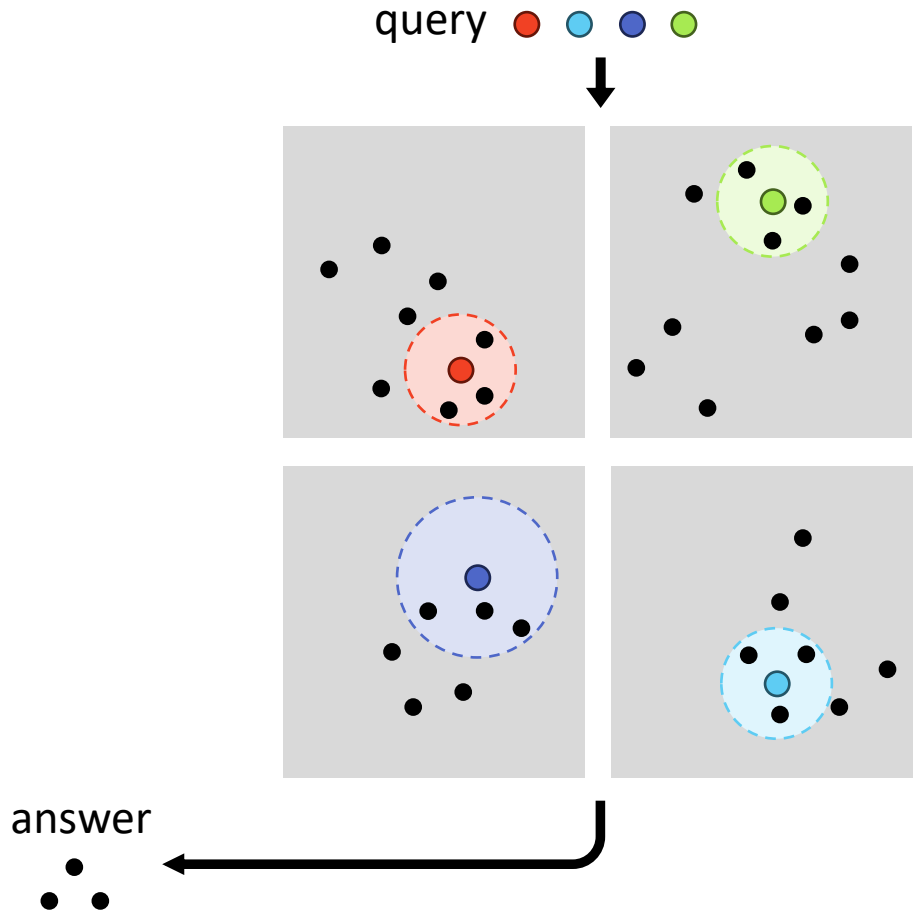
MULTI-VECTOR QUERIES

- Other names: hybrid search, multi-modal search
- Simultaneous search in multiple vector spaces
- Or single entity has multiple vectors
- Why?
  - Combine text and audio search
  - Encode faces from several angles
  - Encode product as text + image
  - Multiple embedding models
  - Encode data at multiple scales



MULTI-VECTOR QUERIES

- Find 3 nearest neighbors to face, fingerprint, voice print, gait analysis
- Idea:
  - Issue 3-NN in all spaces
  - Combine somehow
- Problem: potentially no overlap
  - Up to 12 different people
  - **How do we merge scores**?
  - Not easily!



MULTI-VECTOR QUERIES

- Naïve method:
  1. Run m kNN searches → mk vectors (from up to mk items)
  2. Combine scores across items
     - Min, weighted average, max, learned, …
  3. Select top k

  → **Can miss true neighbours**! (due to how we combine scores)
- Classic top-k algorithms?
  - Require "get next candidate" operation

  → **Not supported by most vector indices!**

MULTI-VECTOR QUERIES

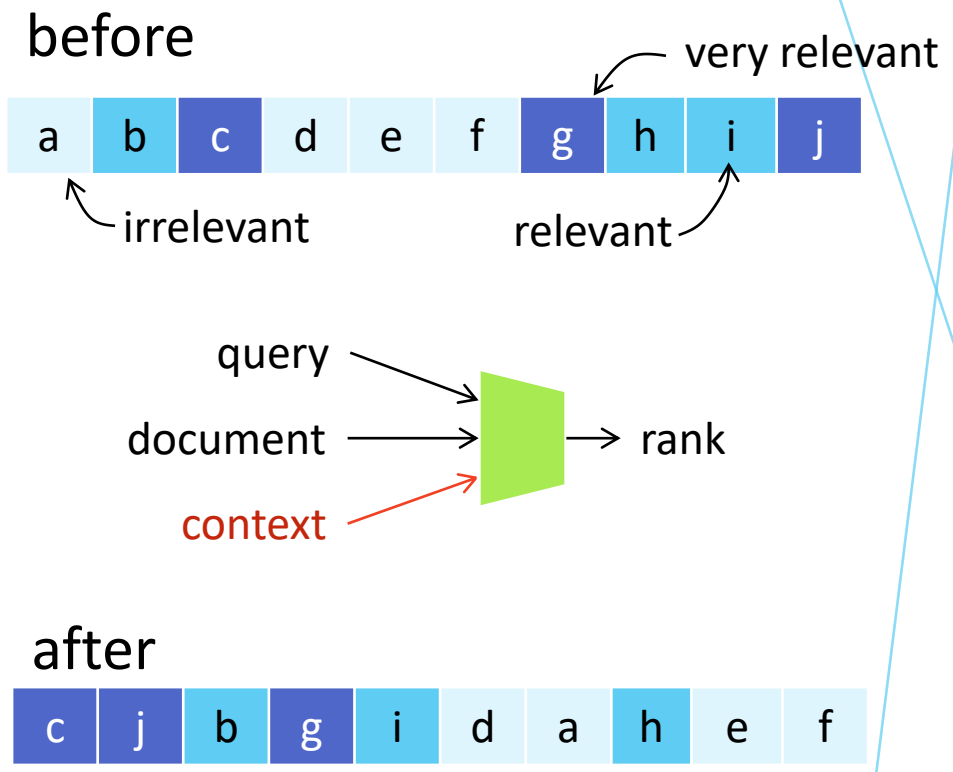
- Vector fusion
  - Store concatenated vectors as single long *v*
  - Single kNN search over concatenated *q*
  - Math works for inner product, **not for Euclidean**
- Iterative merging (Milvus)

  Based on classic top-k algorithm NRA

  1. Issue kNN queries with k'
  2. Run NRA on result sets, stop if finished.
  3. Double k', go to 1
- MUST [Wang, ICDE'24]
- Recent work by Timescale on streaming retrieval index

RERANKING STEP AFTER QUERYING

- Many VDBMS provide **re-ranking**
- Reorder kNN result to improve relevance:
  1. Get kNN result set
  2. Apply **re-ranking model** to result
- Common in RAG, search, recommendation
- Why?
  - Distance scores measure **similarity**, not **relevance**
  - They are simplistic
  - Approximation introduces errors
  - Can afford to be smarter in rerank
  - Can **incorporate context** in re-ranking model



EXACT kNN SEARCH IS SLOW!

- Exhaustive search is **very slow**:
  - Compute $d(q, v)$ for all v, then sort / use priority queue / top-k
  - Supports exact kNN, range queries, predicated queries, everything
  - Time: O(ND) + top k time
- Solution: **approximate nearest neighbour search (ANNS)**
  - Return k vectors, not guaranteed to be nearest
- Key enabler: **ANNS index**
  - Quick but potentially inaccurate queries
  - Slower inserts, potential memory, storage costs

SEARCHING WITH ANNS INDEX

- Given *q*, quickly find potential neighbours
- The index deal:
  - Faster search ☺
  - **Less accuracy**, more memory, slower updates ☹
- Example
  - Building: cluster vectors, associate with nearest centroid
  - Querying: find nearest centroid to q, search its list
  - Errors: q near edge → may miss nearer neighbours!
- **Let's talk about indexing!**