CS 6400 A

Database Systems Concepts and Design

Lecture 20

11/25/24

Announcements

Upcoming deliverables:

- Paper critique (assignment 4): due tonight
- Project presentation video: Dec 2
- Project demo: Dec 6

Sign up for project demo: https://tinyurl.com/3uukakra

- Email the course staff if you absolutely need to reschedule

Exam 2 solution will be posted on canvas tonight

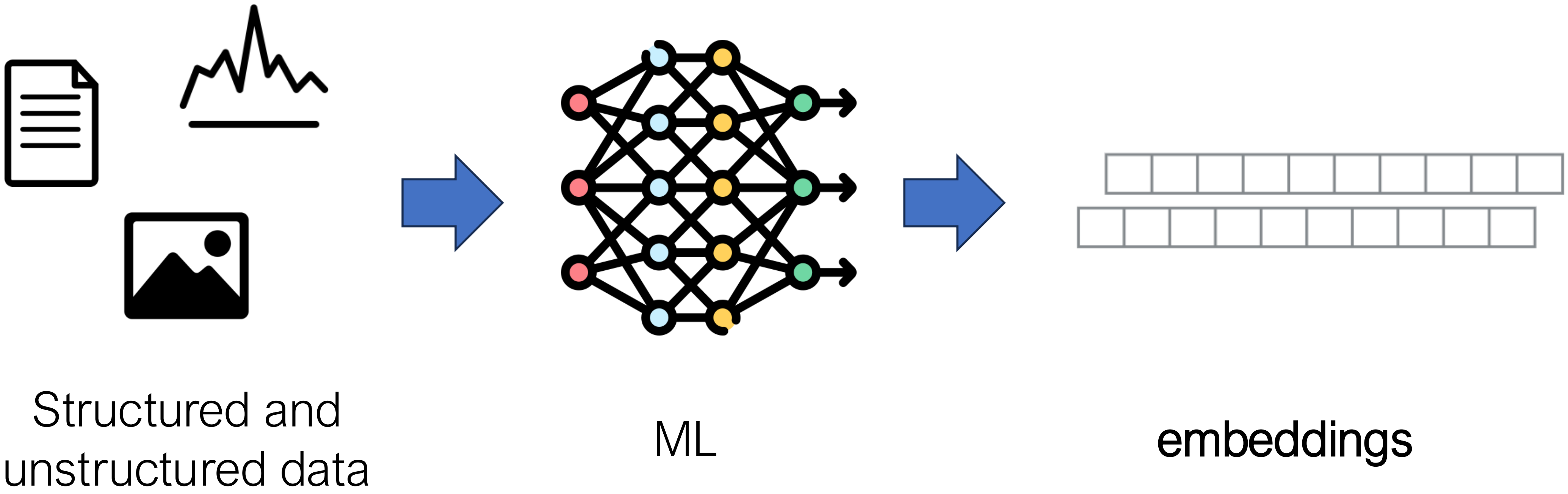
Agenda

1. Vector Search/Database Overview
2. Locality-Sensitive Hashing
3. Product Quantization
4. Graph-based Algorithms

1. Overview

Similarity Search

- Finding the most relevant data points in the database when compared to a specific query point



Scale of Embeddings

Example: OpenAI

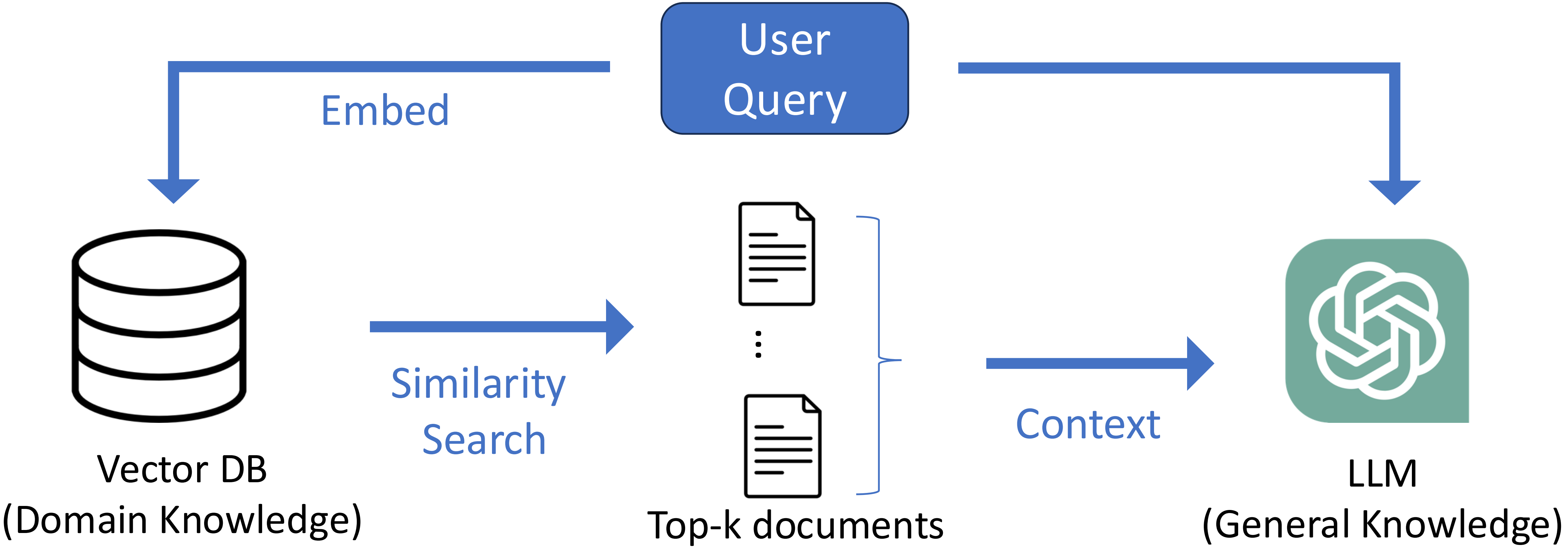
- text-embedding-3-small: 1536 dims
  - 1536 * 4 bytes = 6 KB
  - 6 KB * 1B = 6 TB
  - 6 KB * 1T = 6 PB

- text-similarity-davinci-001: 12288 dims
  - 12288 * 4 bytes = 49 KB
  - 49 KB * 1B = 49 TB
  - 49 KB * 1T = 49 PB

Significant memory requirement for processing billion/trillion scale vector datasets

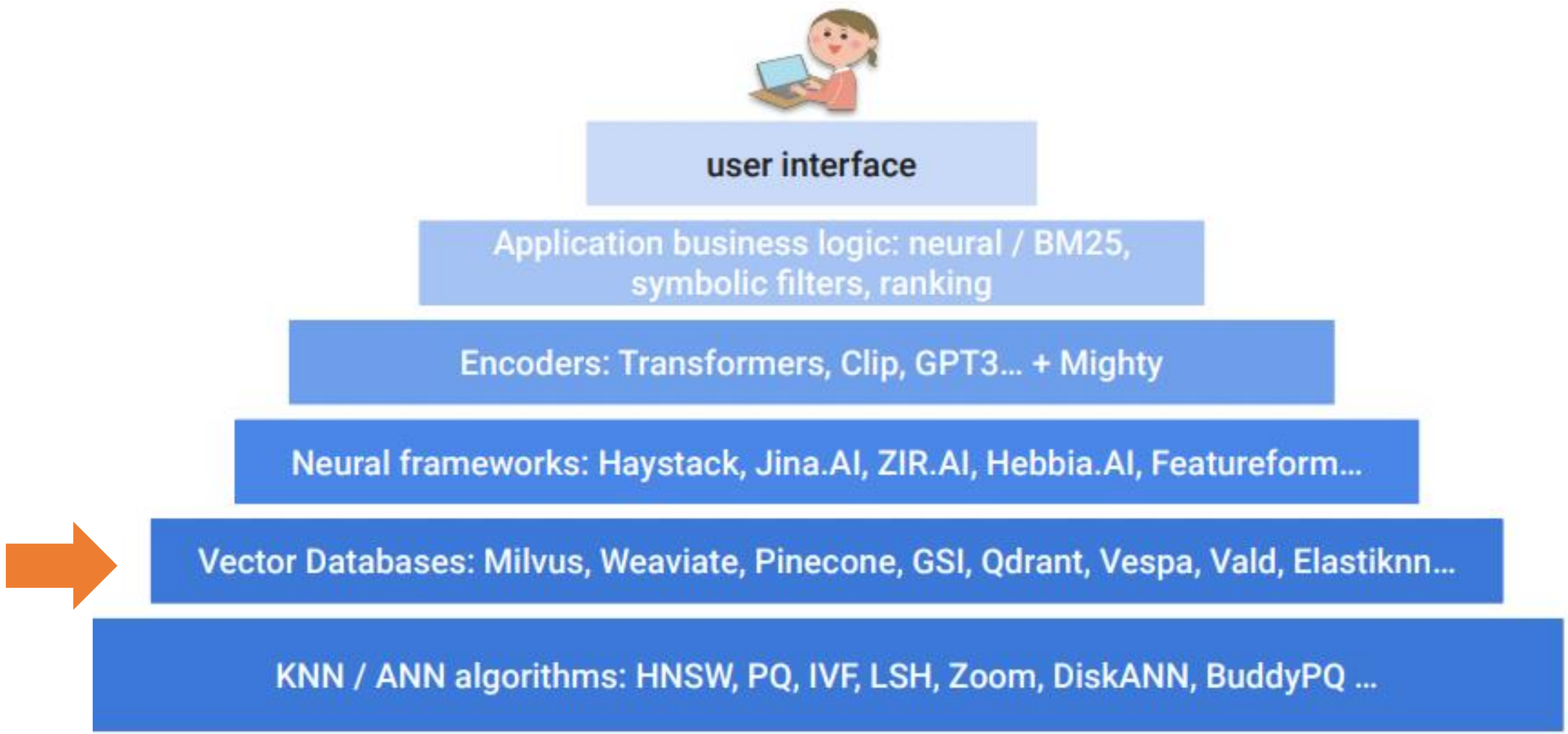
Source: openai.com

Vector Search in LLMs (Retrieval Augmented Generation)



Vector DB is in the critical path of LLM applications – we need them to be performant!

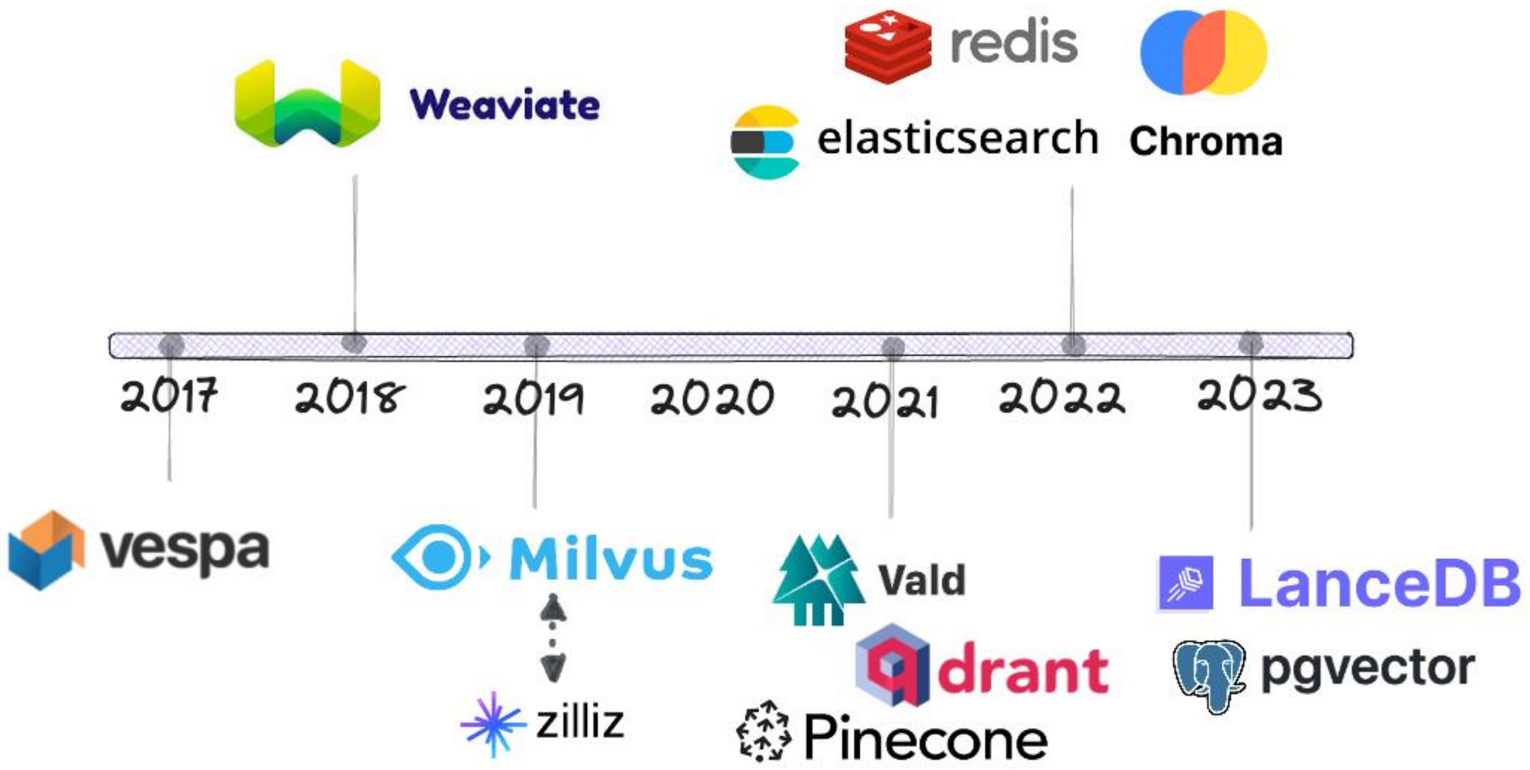
Vector search pyramid



Source: https://www.youtube.com/watch?v=2o8-dX__EgU&ab_channel=OpenSourceConnections

Vector Databases

- Fast similarity searches and retrieval for high-dimensional vectors
- Consistency guarantees, multi-tenancy, cloud-native, CRUD, logging and recovery, serverless, etc



Source: https://thedataquarry.com/posts/vector-db-1/

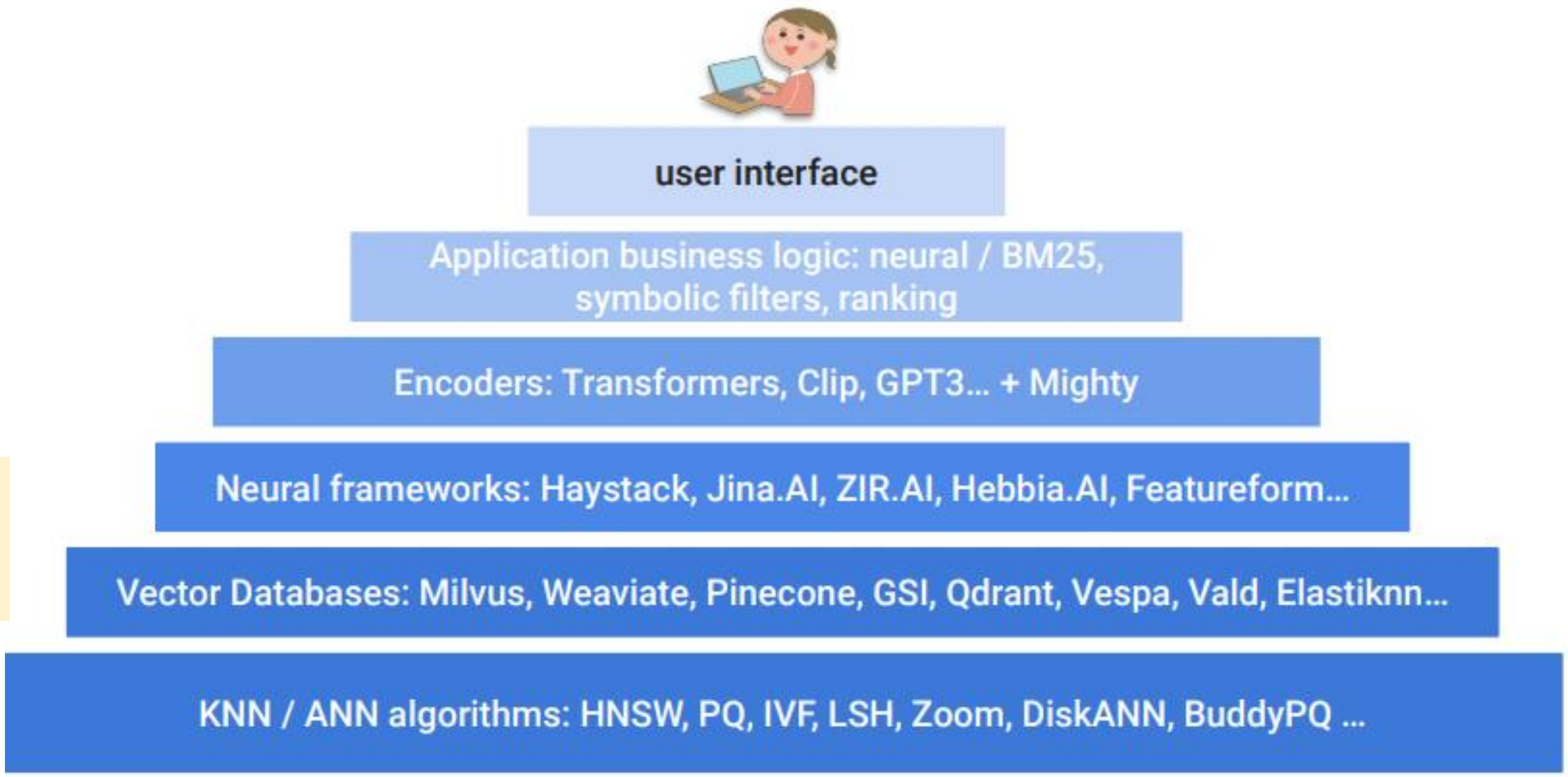
Vector Databases

How do vector databases compare to RDBMS

	RDBMS	Vector Databases
Indexing	B+ Tree, LSM Tree	HNSW, IVF, LSH
Query	Filter, Project, Aggregate, Sort	ANN, Hybrid Search
Performance Metrics	Transactions / Second	Queries / Second, Recall
Performance Metrics	Index loaded page-by-page	Entire index in memory

Vector search pyramid

Our focus today



Source: https://www.youtube.com/watch?v=2o8-dX__EgU&ab_channel=OpenSourceConnections

Indexing Algorithms in Vector Databases

- Pinecone ········ Proprietary composite index
- milvus / zilliz ········ Flat, Annoy, IVF, HNSW/RHNSW (Flat/PQ), DiskANN
- Weaviate ········ Customized HNSW, HNSW (PQ), DiskANN (in progress...)
- Qdrant ········ Customized HNSW
- chroma ········ HNSW
- LanceDB ········ IVF (PQ), DiskANN (in progress...)
- vespa ········ HNSW + BM25 hybrid
- Vald ········ NGT
- elasticsearch ········ Flat (brute force), HNSW
- redis ········ Flat (brute force), HNSW
- pgvector ········ IVF (Flat), IVF (PQ) in progress...

Common indexes: HNSW, IVF(PQ)

Source: https://thedataquarry.com/posts/vector-db-1/

Index Algorithms: Big players in the field

- Meta: FAISS (CPU & GPU)
- Google: ScaNN
- Microsoft (Bing team): DiskANN, SPTAG
- Spotify: ANNOY
- Amazon: KNN based on HNSW in OpenSearch
- Baidu: IPDG (Baidu Cloud)
- Alibaba: NSG (Taobao Search Engine)

Source: https://www.youtube.com/watch?v=2o8-dX__EgU&ab_channel=OpenSourceConnections

Problem: Nearest Neighbor Search (NNS)

Problem definition: given a query object q, we search in a massive high-dimensional dataset $\mathcal{D}$ for one or more objects in $\mathcal{D}$ that are among the closet to q according to some similarity or distance metric.

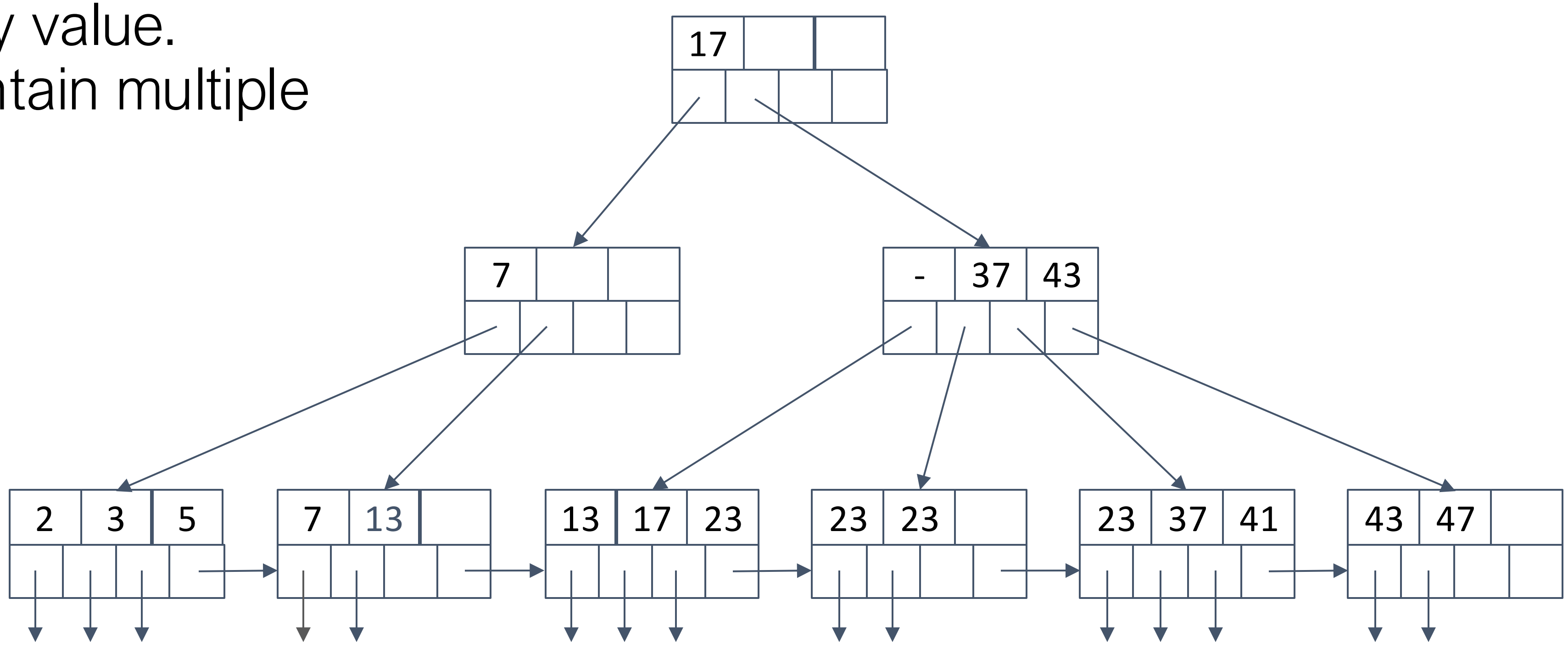
Common similarity metric:

- Euclidean Distance: $||\vec{\mathrm{q}} - \vec{\mathrm{p}}||_2$
- Manhattan Distance: $||\vec{\mathrm{q}} - \vec{\mathrm{p}}||_1$
- Jaccard Similarity: $\frac{|\mathrm{q} \cap \mathrm{p}|}{|\mathrm{q} \cup \mathrm{p}|}$ (q and p are two arbitrary sets)

One-dimensional Indexes

Recall that B-trees are examples of a one-dimensional index

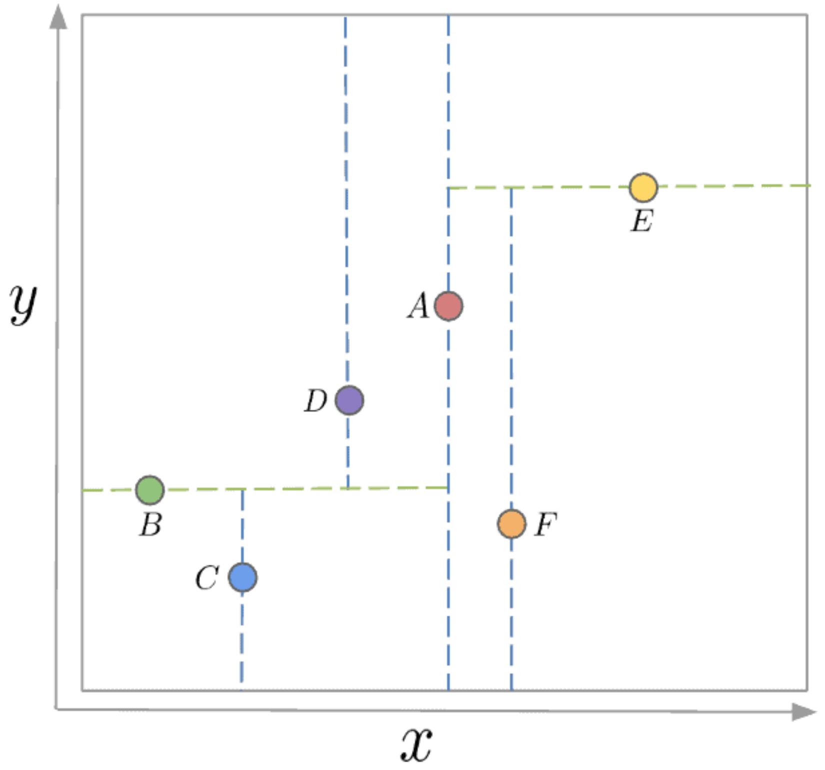
- Assume a single search key, and they retrieve records that match a given search key value.
- The key can contain multiple attributes



Multidimensional Indexes

Multidimensional indexes:

- Specifically designed to partition multi-dimensional data
- Examples: kd-tree, R-tree
    - kd-tree: pick a dimension, find median, split data, repeat



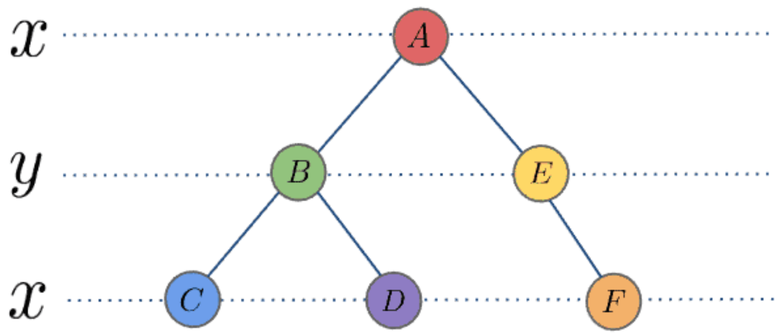


Image source: https://www.baeldung.com/cs/k-d-trees

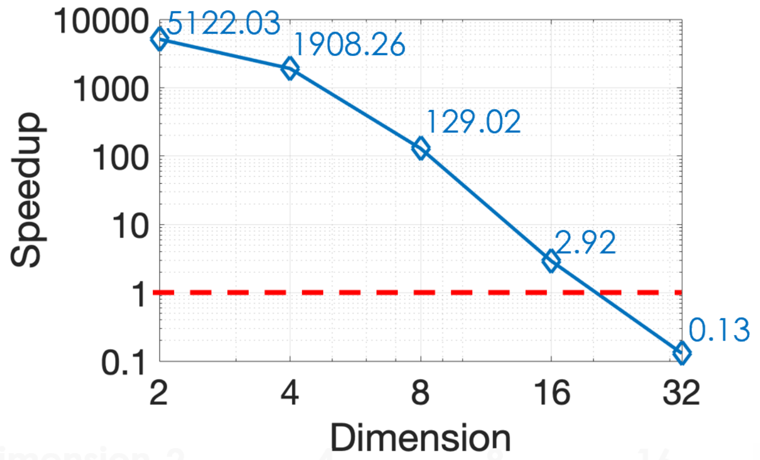
Curse of Dimensionality

Linear scan takes $O(n)$ per query

One of the most popular NNS solutions is the search-tree algorithms, such as kd-tree or R-tree.

However, when the dimension d is very large, search tree performs no better than the linear scan, due to the “curse of dimensionality” [C1994].

Example: k-d tree versus linear scan.



[C1994] K. L. Clarkson. An algorithm for approximate closest-point queries. In Proceedings of the Annual Symposium on Computational Geometry, pages 160–164, 1994.

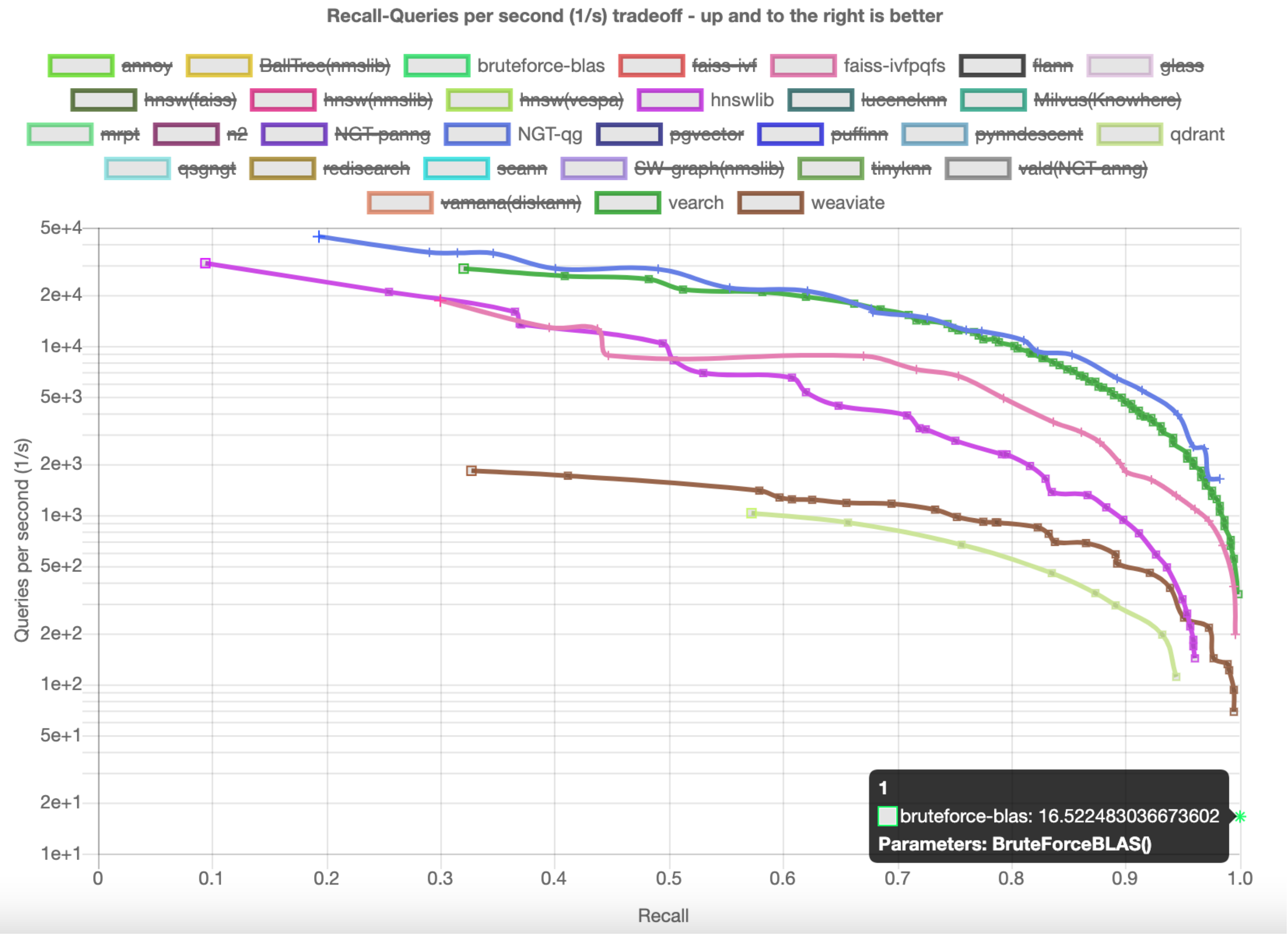
Approximate Nearest Neighbor Search

Problem Definition: Given a query object q, we search in a massive high-dimensional dataset $\mathcal{D}$ for one or more objects in $\mathcal{D}$ that are among the closet to q *with high probability* according to some similarity or distance metric.

ANNS solutions are usually much faster than linear scan with negligible accuracy loss.

- Tradeoff between performance and accuracy

Approximate Nearest Neighbor Search



Exact

This image is downloaded from: http://ann-benchmarks.com/fashion-mnist-784-euclidean_10_euclidean.html

Popular ANNS Algorithms

Locality sensitive hashing (LSH)

Product Quantization (PQ)

Nearest neighbor graph

- KNN graph
- Hierarchical Navigable Small Worlds (HNSW)

VectorDB	ANN library	ANN algorithm
Milvus	Custom FAISS	PQ
Pinecone	Custom FAISS	LSH, PQ
Qdrant	Custom HNSW	NN graph
Pgvector	Custom HNSW	NN graph

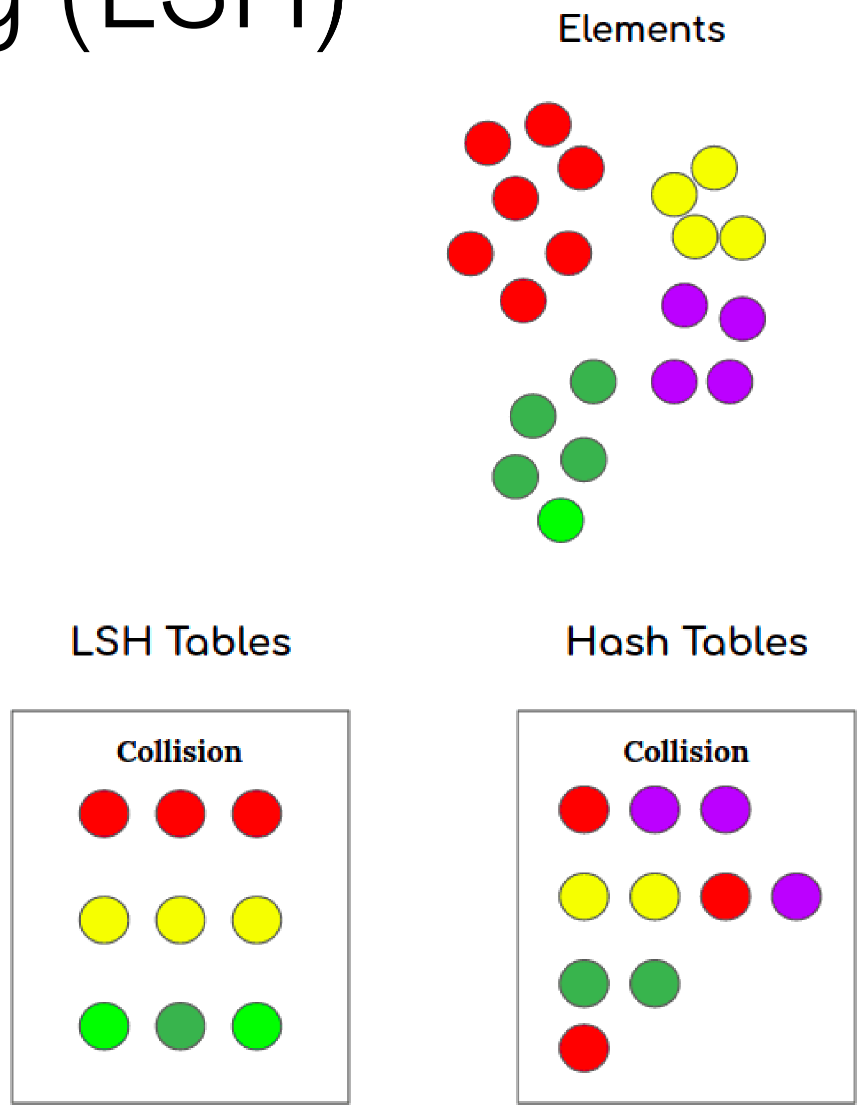
2. Locality Sensitive Hashing

Locality sensitive hashing (LSH)

1. LSH for Cosine Distance

2. Using LSH for ANNS

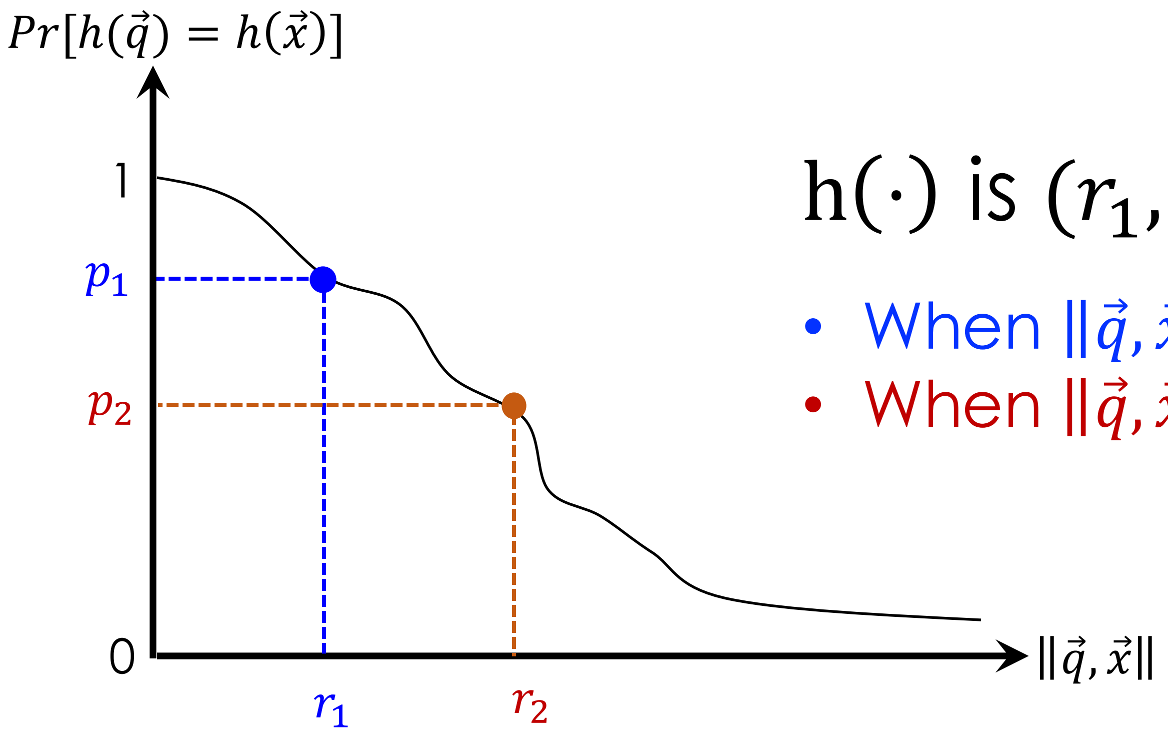
3.Tuning parameters in LSH



Locality sensitive hashing (LSH)

A locality sensitive hashing $\mathbf{h}(\cdot)$ function has the following distance-preserving property:

- The collision probability between two items $Pr[h(\vec{q}) = h(\vec{x})]$ *monotonically decreases* with their distance $\|\vec{q}, \vec{x}\|$

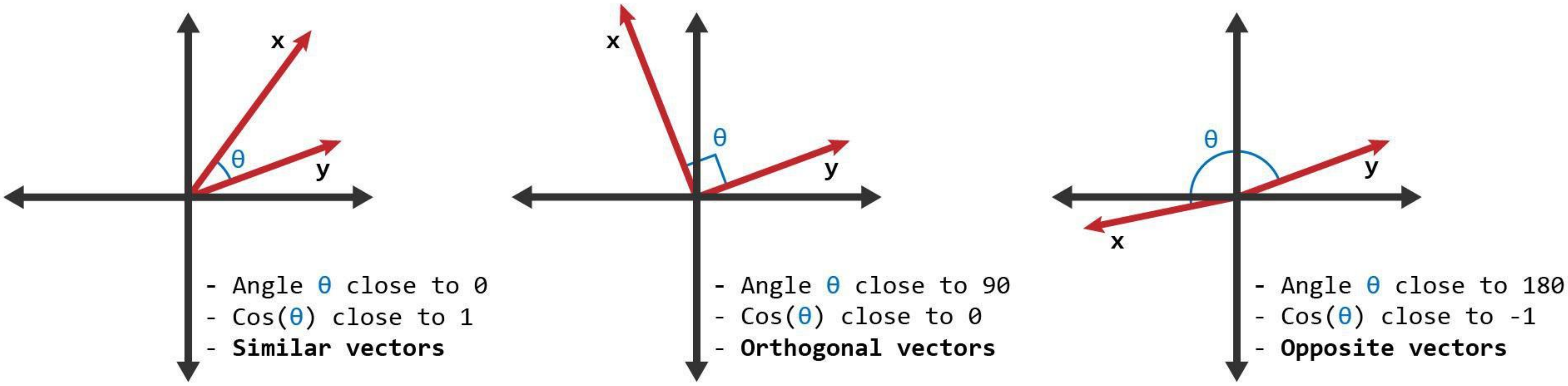


$\mathbf{h}(\cdot)$ is (r_1, r_2, p_1, p_2) –sensitive

- When $\|\vec{q}, \vec{x}\| \leq r_1$, $Pr[h(\vec{q}) = h(\vec{x})] \geq p_1$
- When $\|\vec{q}, \vec{x}\| \geq r_2$, $Pr[h(\vec{q}) = h(\vec{x})] \leq p_2$

LSH for Cosine Distance

- For cosine distance, there is a technique for generating a $\left(d_1, d_2, 1 - \frac{d_1}{180}, 1 - \frac{d_2}{180}\right)$-sensitive family for any d_1 and d_2
- Called *random hyperplanes*.



Random Hyperplanes

Pick a random vector v, which determines a hash function h_v with two buckets:

- $h_v(x) = +1$ if $v \cdot x > 0$
- $h_v(x) = -1$ if $v \cdot x < 0$

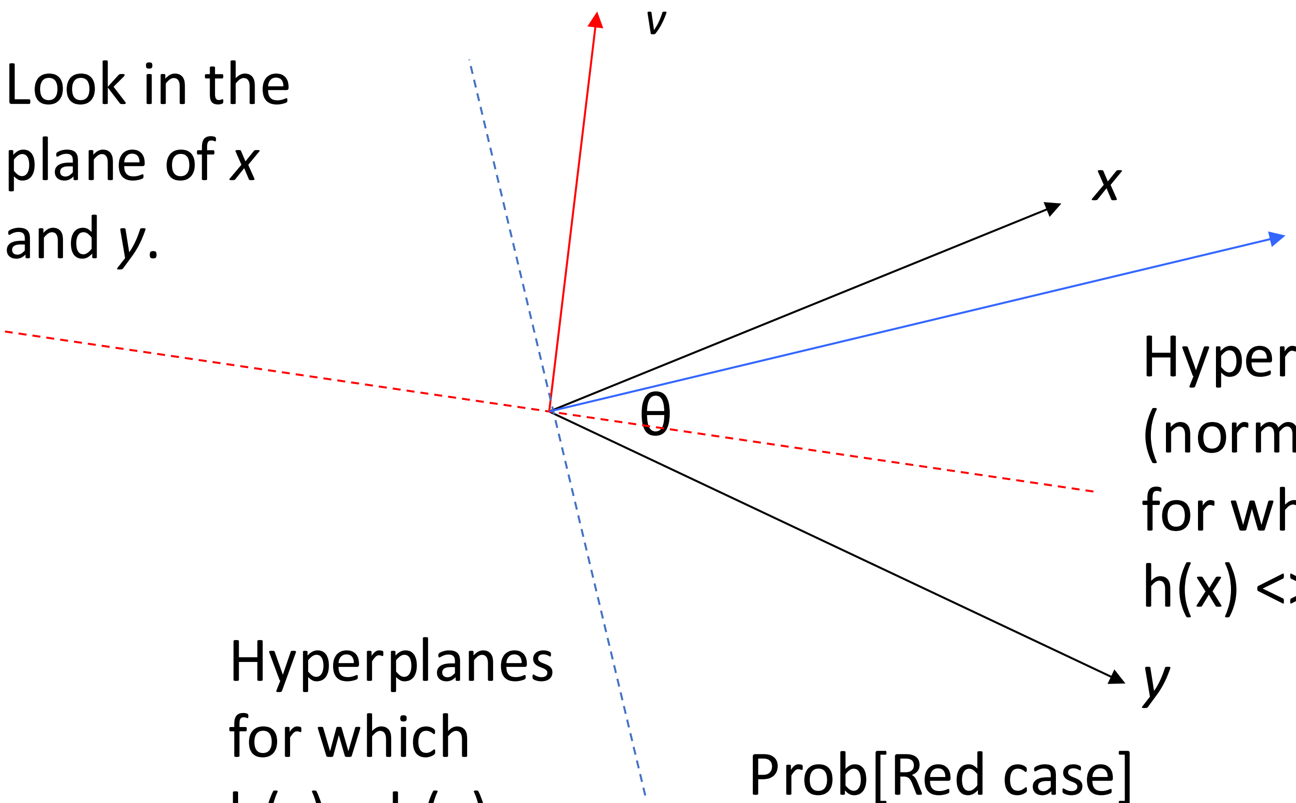
LSH-family **H** = set of all functions derived from any vector.

$$Prob[h(x) = h(y)] = 1 - \frac{\theta}{180}, \cos\theta = \frac{x \cdot y}{|x||y|}$$

Proof of Claim

$$Prob[h(x) = h(y)] = 1 - \frac{\theta}{180}, \cos\theta = \frac{x \cdot y}{|x||y|}$$

Look in the plane of *x* and *y*.

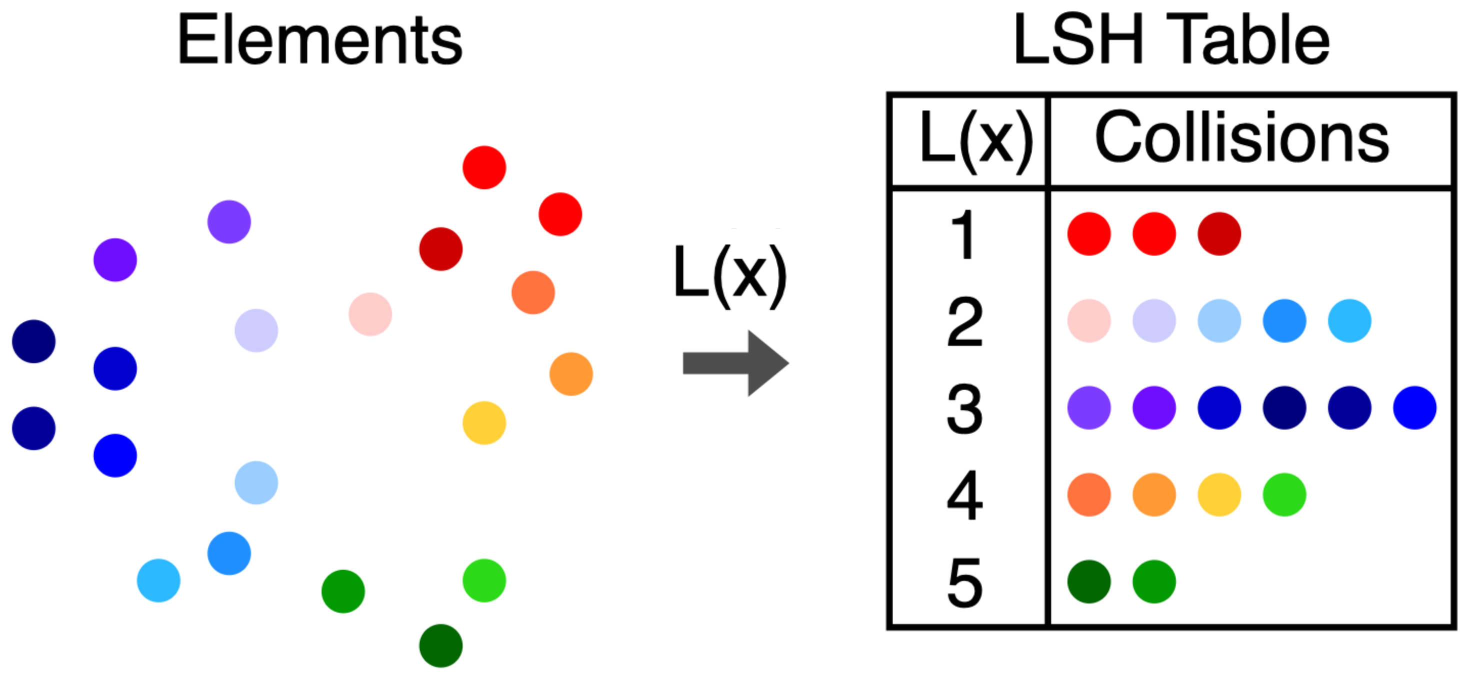


Hyperplanes (normal to *v*) for which h(x) <> h(y)

Hyperplanes for which h(x) = h(y)

Prob[Red case] = θ/180

Using LSH for ANNS



Main idea: Only check data points that *hash collides* with the query (instead of the entire dataset)

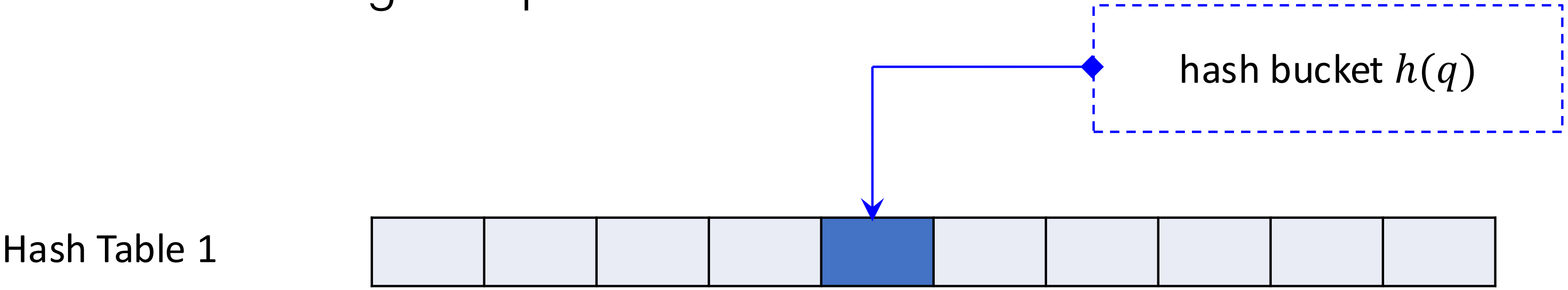
- Two points are close in the projected space are likely to be close in the original space

Image source: https://randorithms.com/2019/09/19/Visual-LSH.html

LSH-based ANNS

Main idea: Only check data points that *hash collides* with the query (instead of the entire dataset)

- Two points are close in the projected space are likely to be close in the original space



For ANNS to be effective, we hope to capture only the true nearest neighbors in $h(q)$

- High precision: low false positives
- High recall: low false negatives

LSH-based ANNS

For ANNS to be effective, we hope to capture only the true nearest neighbors in $h(q)$

- High precision: low false positives
- High recall: low false negatives

Q: What's the probability of two vectors being on the same side of M random hyperplanes?

- Suppose that $P[h_1(x) = h_1(q)] = p$.
- What is $P[h_1(x) = h_1(q) \& h_2(x) = h_2(q) \& \ldots \& h_M(x) = h_M(q)]$?

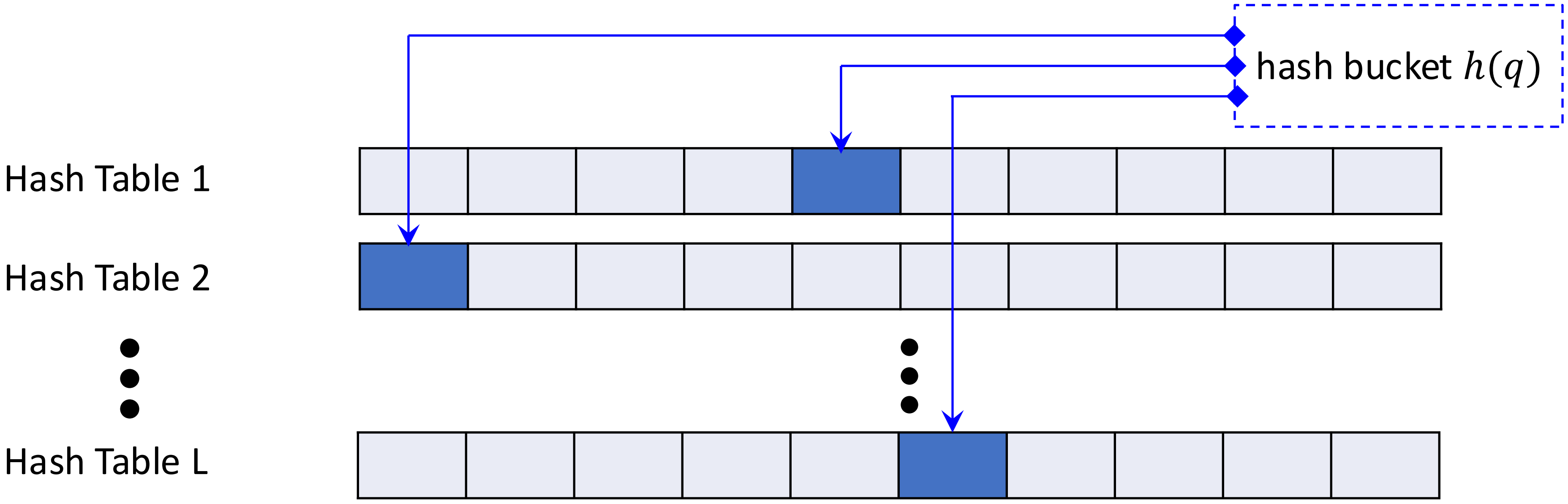
Collision probability reduces to p^M

- Harder for false positives to result in a hash collision

  => increase precision
- Q: What about recall?

LSH-based ANNS

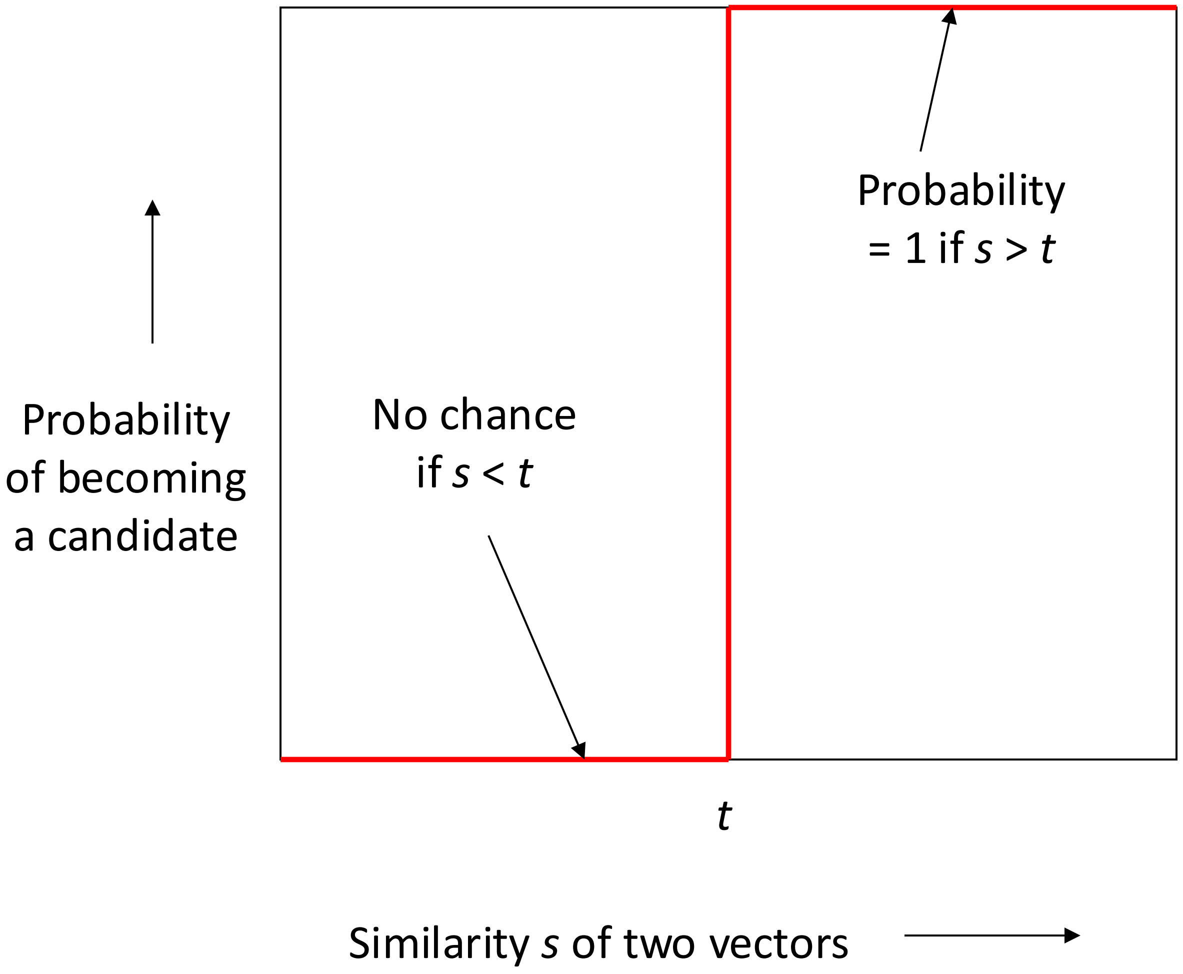
- How to increase recall:
  - Repeat multiple times. Consider a data point an NN candidate if it hash collides with the query in any trial
- Build L hash tables
  - Each table generates hash signatures using M random hyperplanes: $\overrightarrow{\mathrm{h}(\cdot)} \triangleq <\mathrm{h}_1(\cdot), \mathrm{h}_2(\cdot), \cdots, \mathrm{h}_M(\cdot)>$
  - Consider the union of $\overrightarrow{\mathrm{h(q)}}$ buckets from each table



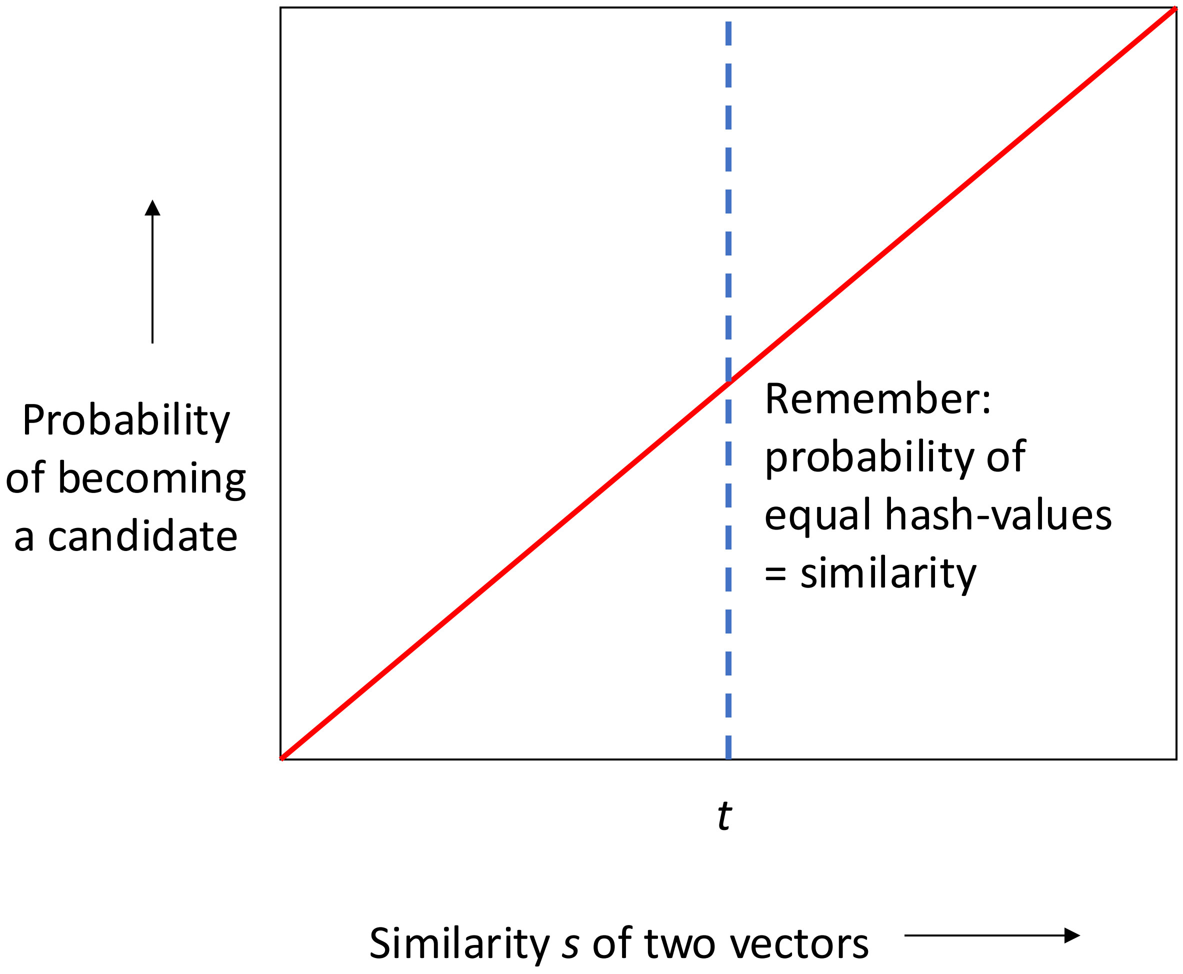
Key Parameters in LSH

- M: number of hash functions (in the hash signature)
    - Larger M increases precision but lowers recall
- L: number of hash tables
    - Larger L increases recall
    - Also at the cost of larger storage overhead
- How to tune these parameters?

Analysis of LSH – What We Want

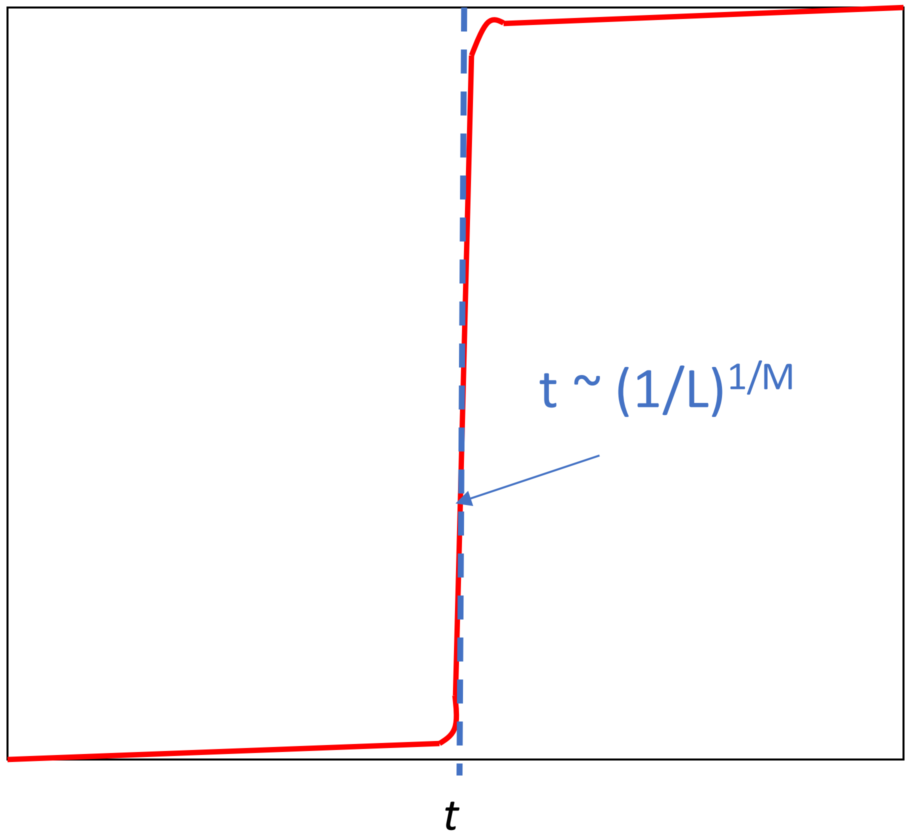


Single hash function (one random hyperplane)



M hash functions, L tables

Probability of becoming a candidate



Similarity s of two vectors

$$1 - (1 - s^{M})^{L}$$

At least one collision

No collision in any of the L tables

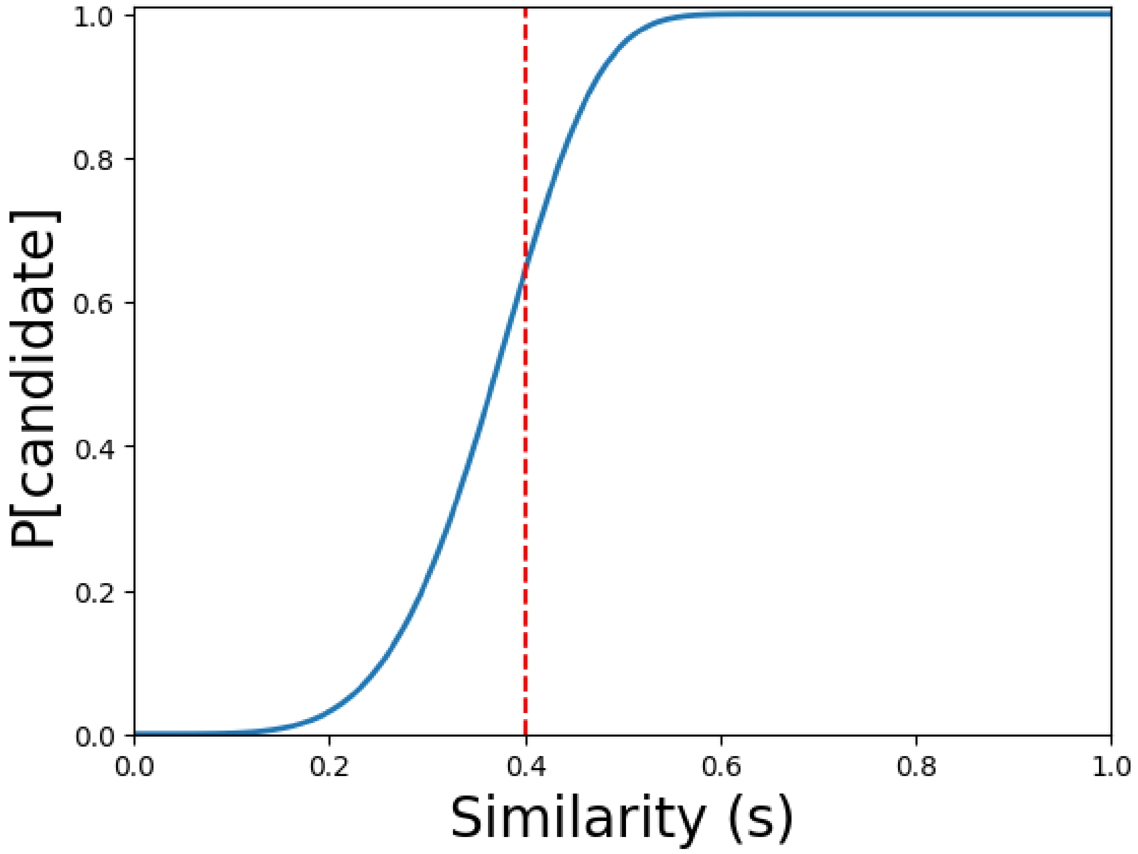
Does not collide in a hash table

All M hash functions are equal (collide in one table)

Example: $L = 20$; $M = 5$

s	$1-(1-s^M)^L$
.2	.03
.3	.22
.4	.64
.5	.96
.6	.9996

$t \approx 0.4$



Multi-probe LSH [VLDB'07]

- Designed to reduce the space requirements of LSH

- In LSH, L can be in the *hundreds* to boost the recall (probability of finding true nearest neighbors in epicenter buckets).

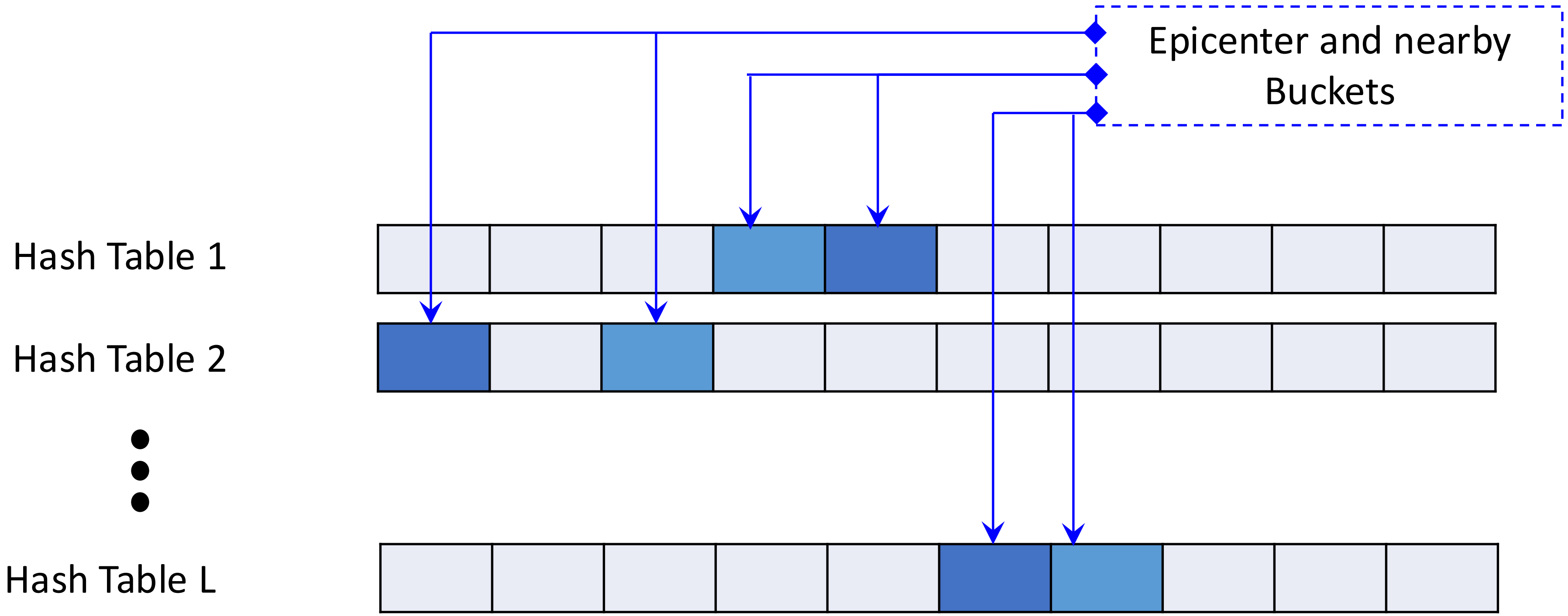
- Multi-probe LSH [Lv2007] was proposed for reducing L when the Gaussian-projection LSH scheme (GP-LSH) is used.

- Main idea: get more information from each hash table

[Lv2007] *Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multiprobe LSH: efficient indexing for high-dimensional similarity search. In PVLDB. 950–961*

Multi-probe LSH [VLDB'07]

- In addition to the epicenter bucket, multi-probe LSH also probes T nearby buckets whose success probabilities (of finding nearest neighbor of $\vec{q}$) are among the T +1 highest.



Significantly reduce L by probing "best" nearby buckets!

3. Product Quantization

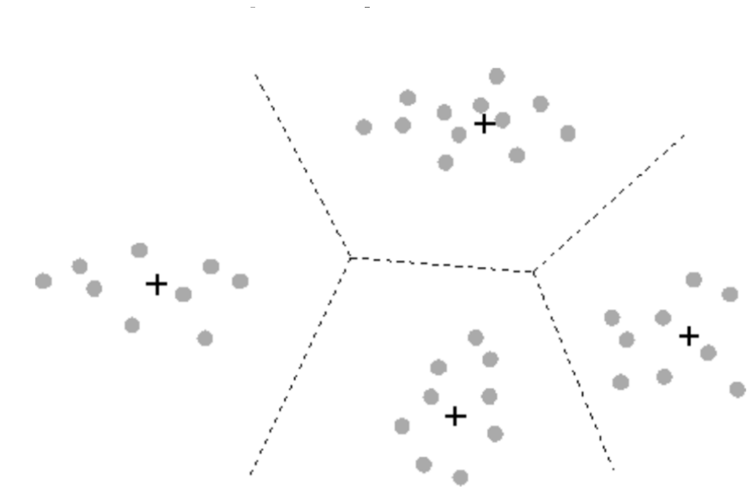
Product Quantization

Winner in [BigANN Competition @ NeurIPS' 21]; a technique for compression high-dimensional vectors, therefore speeding up the similarity search.

Popular implementation: Meta's [faiss library]

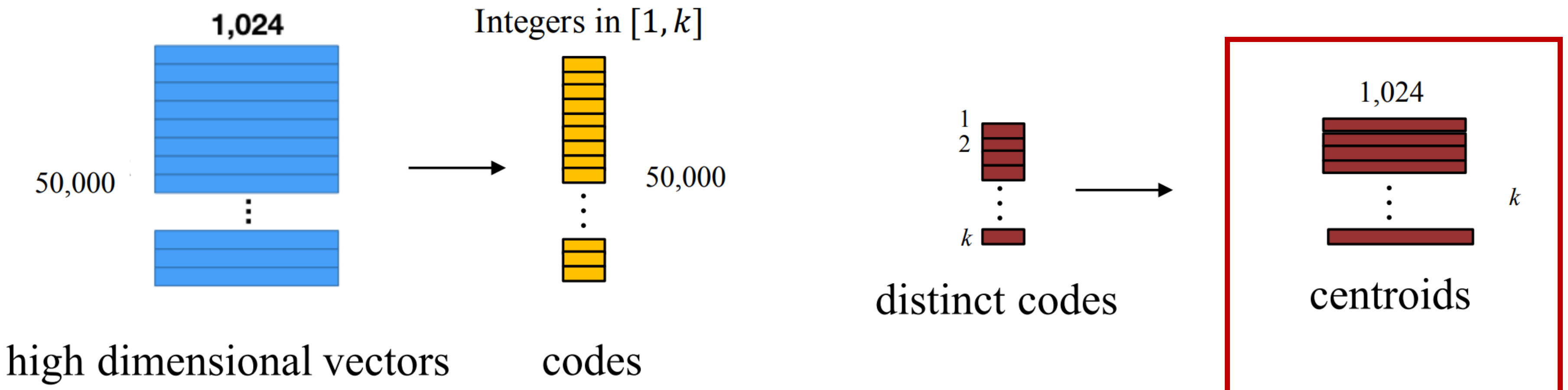
Vector Quantization: use centroids to represent

- distance(query, vector) ~ distance(query, centr



Vector Quantization

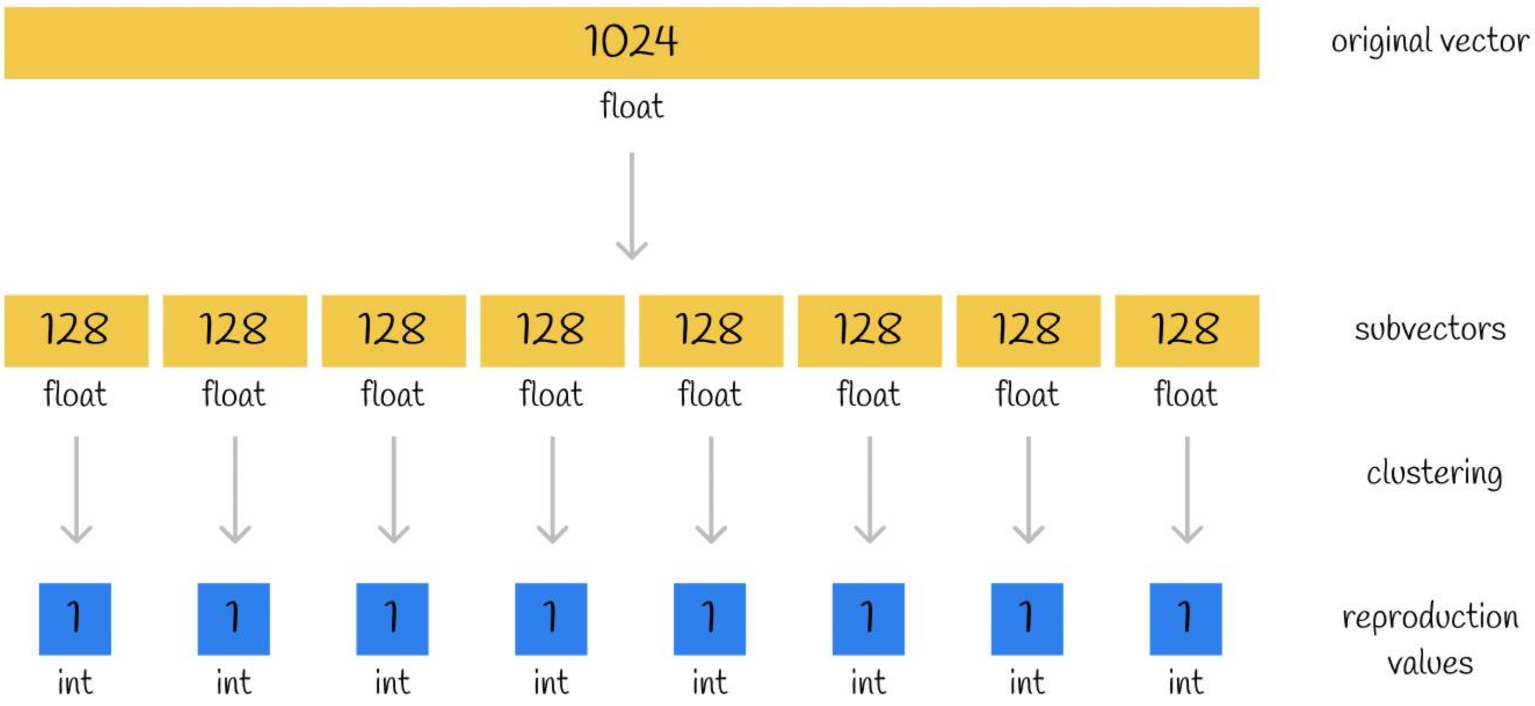
- Map the original dataset by a vector quantizer with k centroids using k-means
- Each code is an integer ranging from 1 to k
- Codebook: a map from code to the centroid



Problem: need a large number of clusters to distinguish vectors

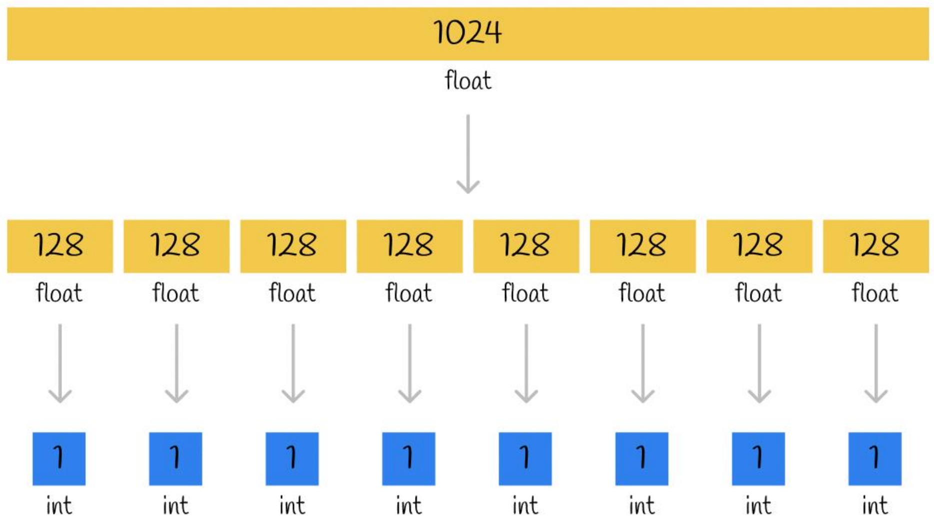
Product Quantization

- Split a high-dimensional vector into equally sized subvectors
- Assigning each of these subvectors to its nearest centroid
- Replacing these centroid values with unique IDs — each ID represents a centroid



https://towardsdatascience.com/similarity-search-product-quantization-b2a1a6397701

Product Quantization



Benefit: Produce a large set of centroids from several small sets of centroids

Suppose we are using 32 bits for each compressed vector

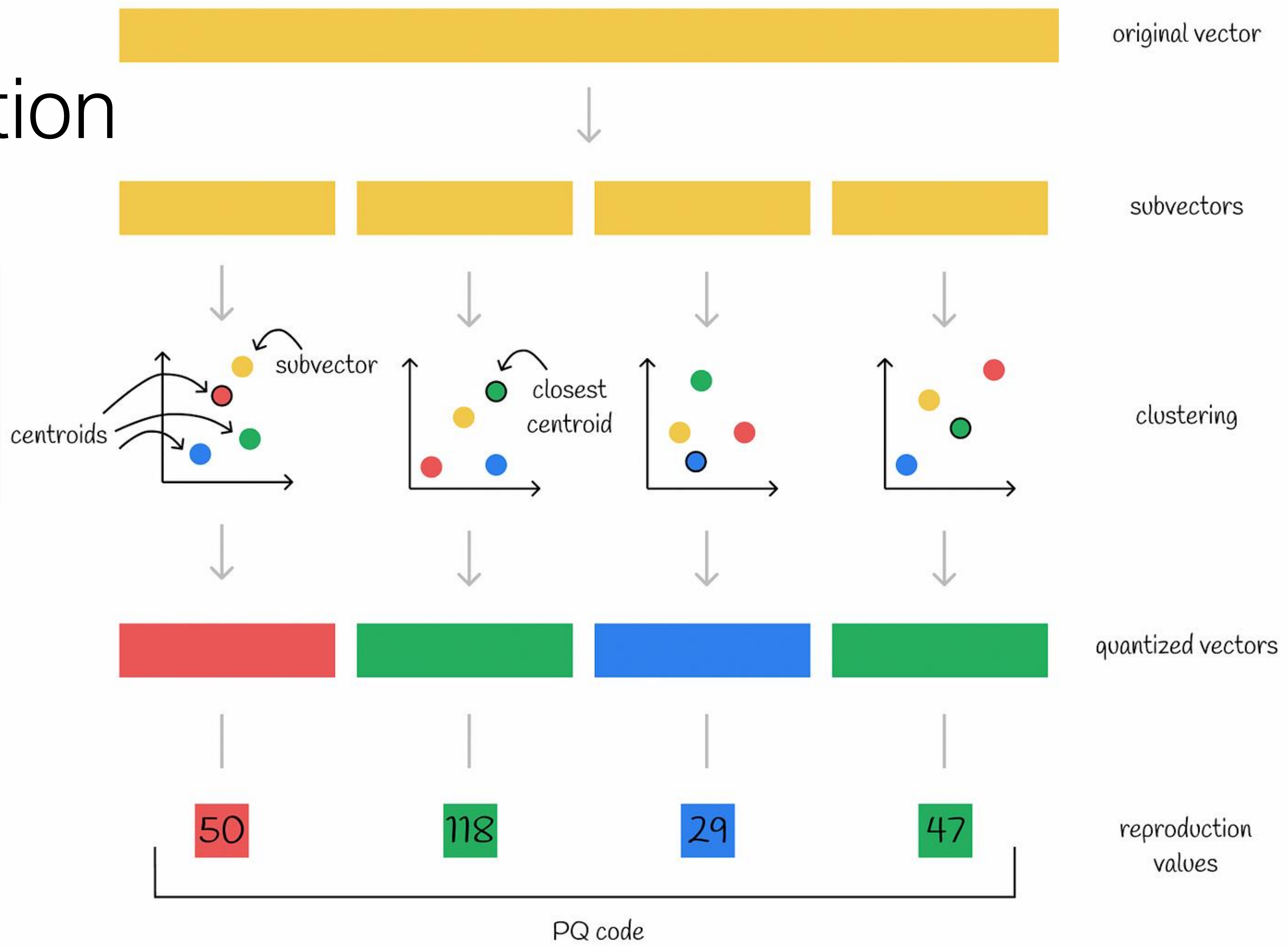
- Vector quantization:
  - $k = 2^{32}$ total centroids
  - Total centroids: $k = 2^{32} = 4{,}294{,}967{,}296$

- Product quantization:
  - $m = 4$ subquantizer
  - $k^* = 2^8$ centroids for each subquantizer
  - Total centroids: $m \cdot k^* = 1024$

$$k = (k^*)^m$$

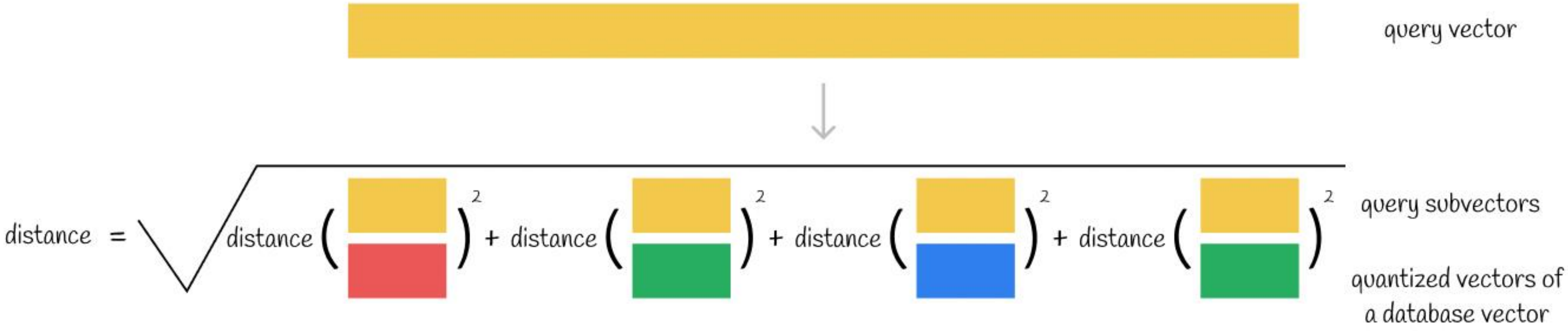
Quantization

Each subvector space has its own set of clusters



https://towardsdatascience.com/similarity-search-product-quantization-b2a1a6397701

Computing Distances with Quantized Codes



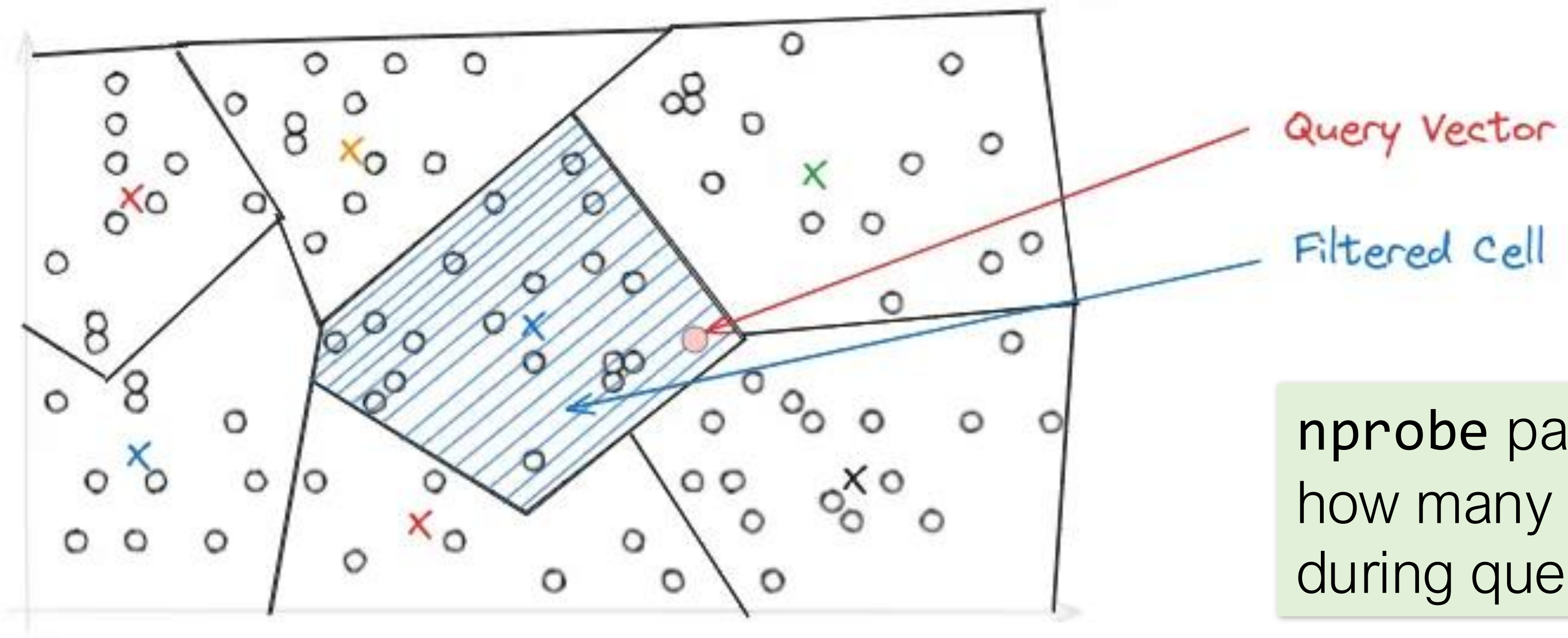
Asymmetric distance computation: The database vector y is represented by $q(y)$, but the query x is not encoded.

$$\tilde{d}(x,y) = \sqrt{\sum_j d(u_j(x), \boldsymbol{q}_j(u_j(y)))^2}$$

https://towardsdatascience.com/similarity-search-product-quantization-b2a1a6397701

IVF-PQ: Search Index

- PQ is just a compression mechanism
- During ANN search, still need an index to avoid exhaustive search
  - IVF: Inverted index of Voronoi cells
  - PQ is usually built on the residuals



`nprobe` parameter controls how many cells to search during query time

https://lancedb.github.io/lancedb/concepts/index_ivfpq/#product-quantization

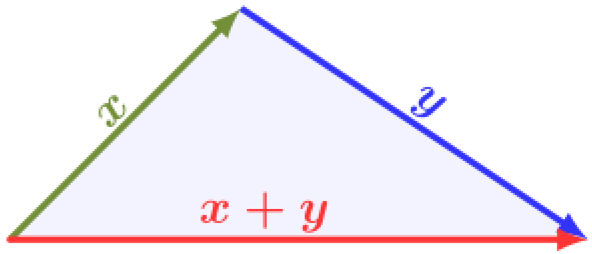
4. Graph-based Algorithms

KNN Graph [WWW'11]

- KNN Graph: for a set of objects V is a directed graph with vertex set V and an edge from each v ∈ V to its K most similar objects in V under a given similarity measure.

- Key intuition: a neighbor of a neighbor is also likely to be a neighbor.

- Triangle inequality:



[Wei2011] Dong, Wei, Charikar Moses, and Kai Li. "Efficient k-nearest neighbor graph construction for generic similarity measures." *Proceedings of the 20th international conference on World wide web*. 2011.

KNN Graph [WWW'11]

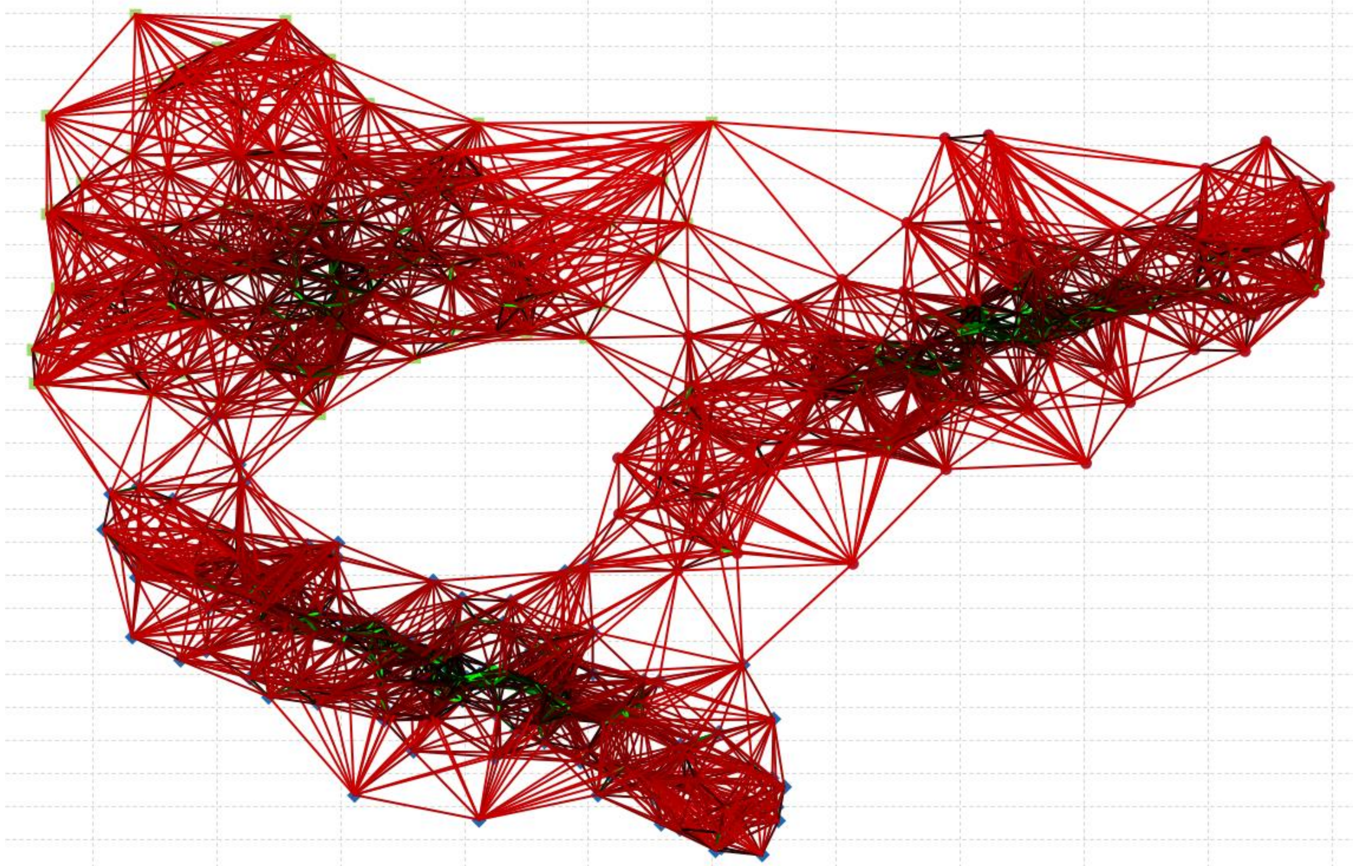
- In the search stage, graph-based algorithms find the candidate neighbors of a query point in some way (e.g., random selection) and then check the neighbors of these candidate neighbors for closer ones iteratively.

- To avoid local optima, we need to traverse over thousands of points to find the nearest neighbors of the query point .

[Wei2011] Dong, Wei, Charikar Moses, and Kai Li. "Efficient k-nearest neighbor graph construction for generic similarity measures." *Proceedings of the 20th international conference on World wide web*. 2011.

KNN Graph [WWW'11]

- The size of KNN graph is usually very large and hard to store in memory.

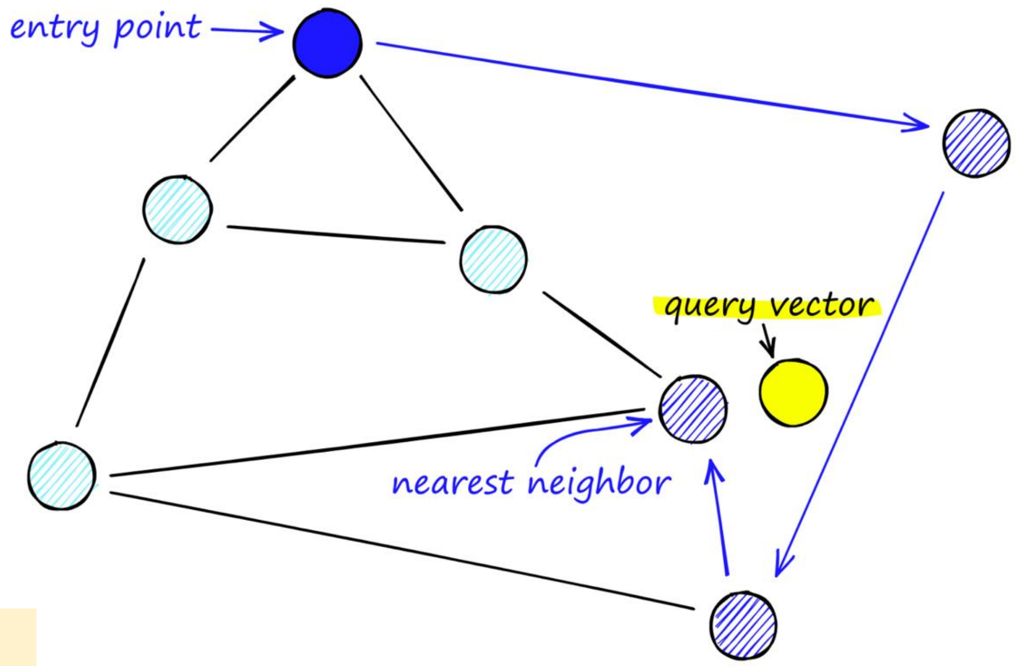


Navigable Small Worlds (NSW)

A KNN graph that has both long-range and short-range links; inspired by the “small-world” phenomenon

Search procedure

- Start from a pre-defined entry point and greedily moves towards the query point
- Stopping condition: find no nearer vertices than our current vertex.



Long-range links help ensure the search doesn’t get stuck in local minima

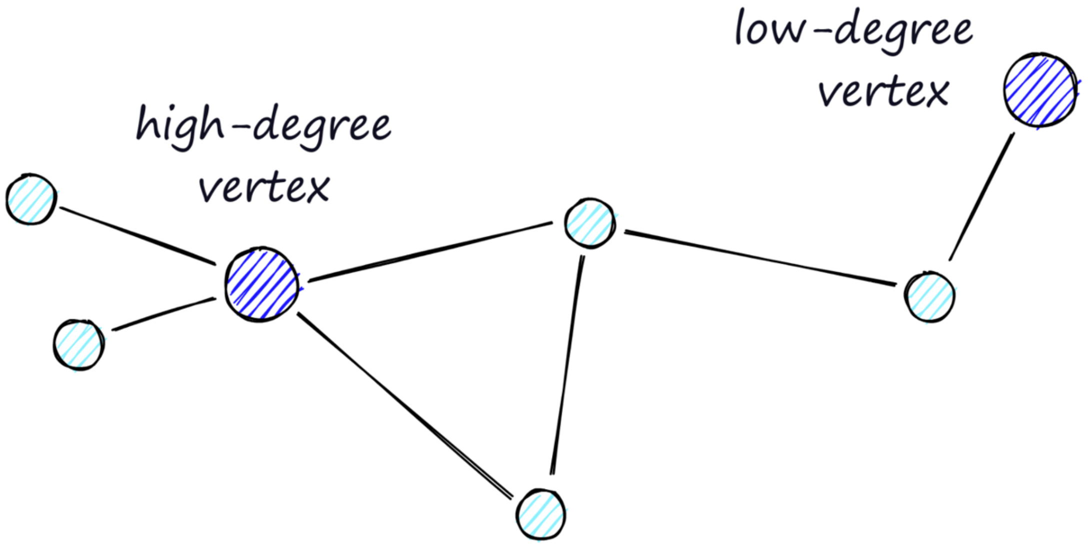
Source: https://www.pinecone.io/learn/series/faiss/hnsw/

Navigable Small Worlds (NSW)

Two phase: start with low-degree vertices ("zoom out") then pass through higher-degree vertices ("zoom in").

- More likely to hit a local minimum and stop too early in the zoom-out phase

Increasing the average degree of vertices would increase search complexity – balance between recall and search speed



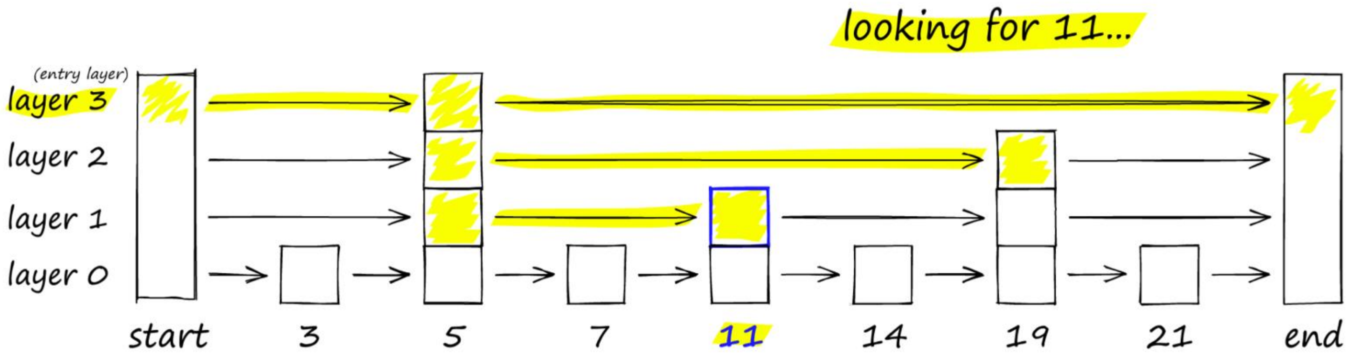
High-degree vertices have *many* links, whereas low-degree vertices have very *few* links.

Source: https://www.pinecone.io/learn/series/faiss/hnsw/

Hierarchical Navigable Small Worlds (HNSW)

Among the top-performing indexes for vector similarity search: fast search speed and good recall

Probability skip list: building several layers of linked lists. On the first layer, we find links that skip many intermediate nodes/vertices. As we move down the layers, the number of 'skips' by each link is decreased.

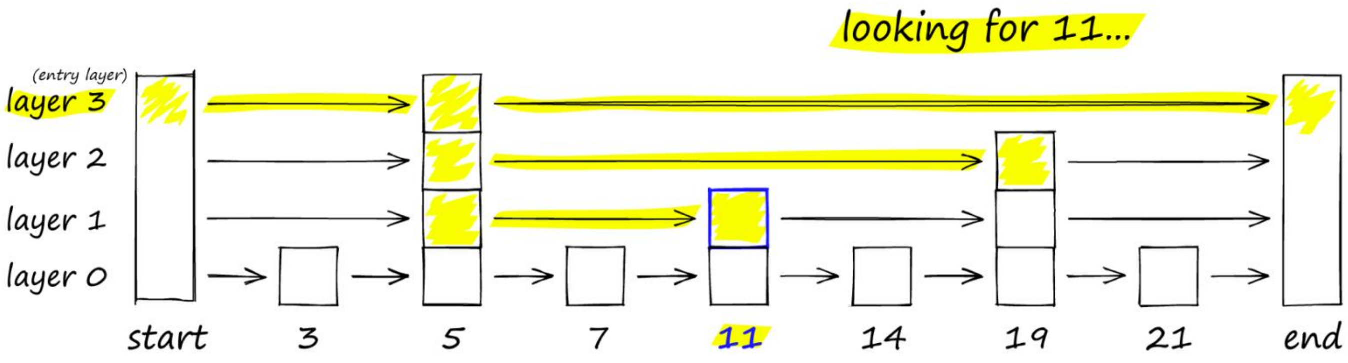


Source: https://www.pinecone.io/learn/series/faiss/hnsw/

Hierarchical Navigable Small Worlds (HNSW)

Search procedure

- Start from the top layer with the longest 'skips'
- If you overshoot, move down to a lower layer

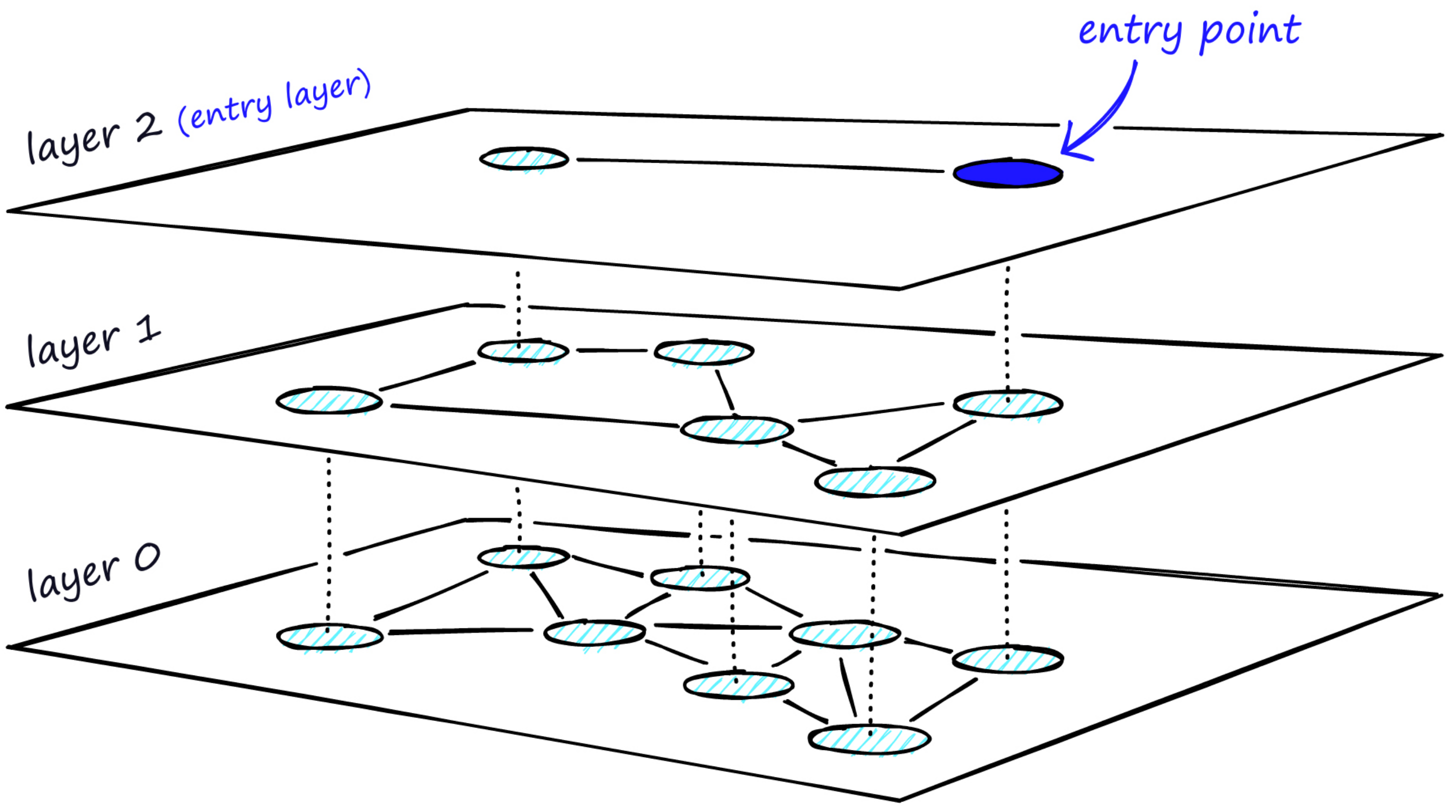


Source: https://www.pinecone.io/learn/series/faiss/hnsw/

Hierarchical Navigable Small Worlds (HNSW)

Main idea: Combine skip list with NSW

- Top layers have longer links and bottom layers have shorter links
- Top layer: fewer vertexes and higher average degree

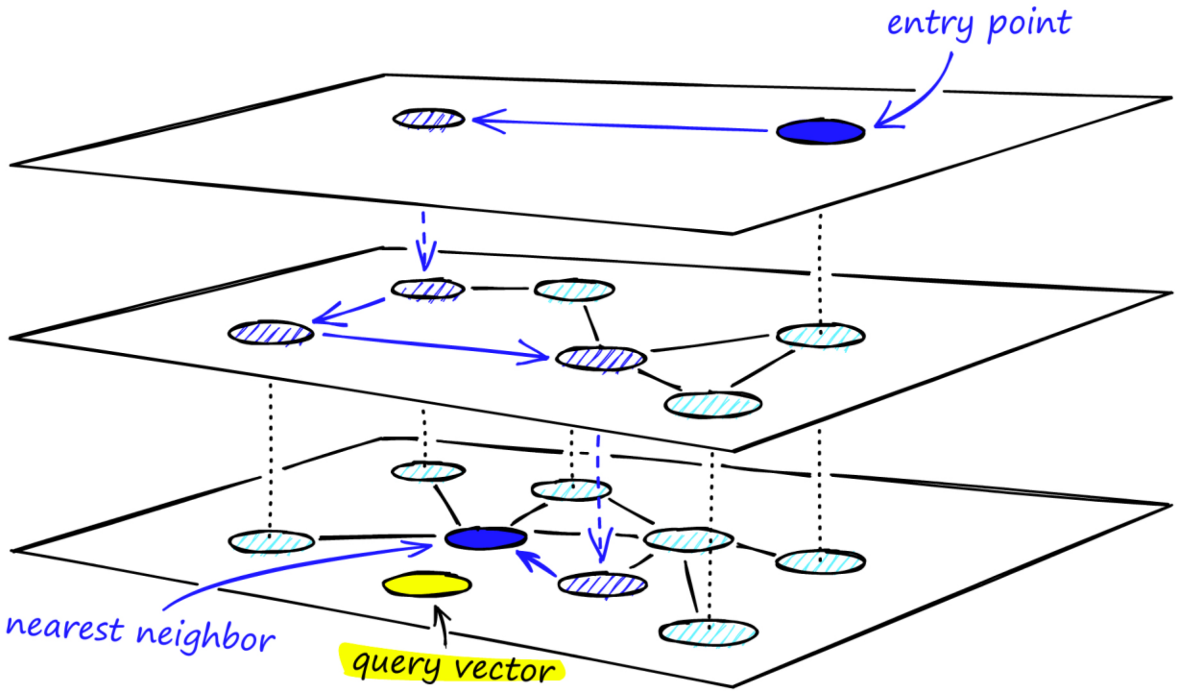


Source: https://www.pinecone.io/learn/series/faiss/hnsw/

Hierarchical Navigable Small Worlds (HNSW)

Search procedure

- Enter from top layer: long links and higher-degree vertices (with links separated across multiple layers)
  - Starting in the “zoom-in” phase
- Upon finding local minimum, move to a lower layer and search again



Source: https://www.pinecone.io/learn/series/faiss/hnsw/

Comparison of ANN algorithms

- Benchmarks:
  - ANN-benchmarks: https://ann-benchmarks.com/
  - Big-ANN benchmarks: http://big-ann-benchmarks.com/neurips23.html

- Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement

Comparison of ANN algorithms

- LSH-based algorithms are easy to index and update and usually have acceptable query performance; not the best fit for high dimensional data and high precision requirement

- Graph-based algorithms have very good query performance with large indexing cost

- Product quantization algorithms are good for very large datasets when memory usage is a concern