CS 4440 A
Emerging Database Technologies

Lecture 16

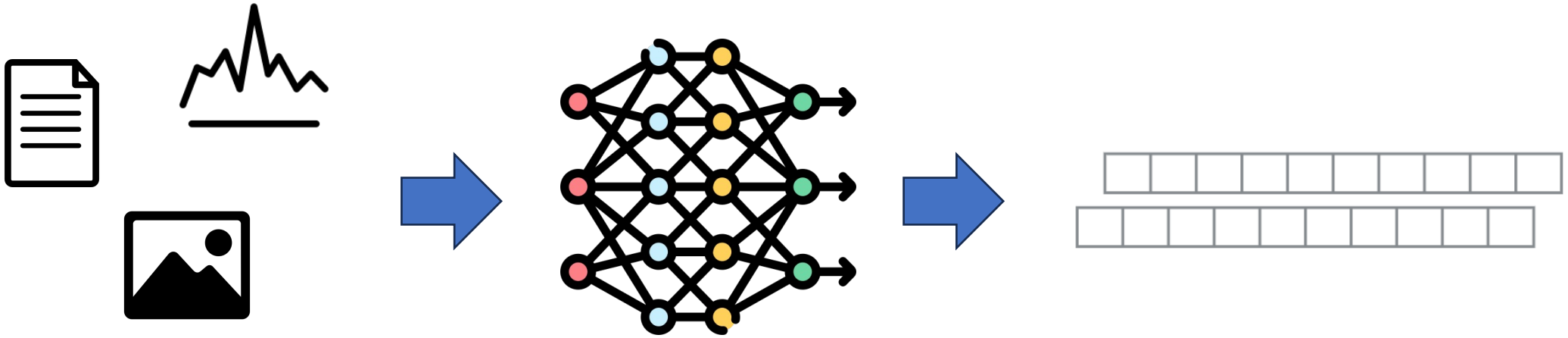
04/01/24

Announcements

- Research paper presentation starts this Wednesday (Apr 3)
  - P1: Map Reduce
  - P2: Big Table
  - P3: C-Store
- If you haven’t yet, start working on your projects!

Similarity Search

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

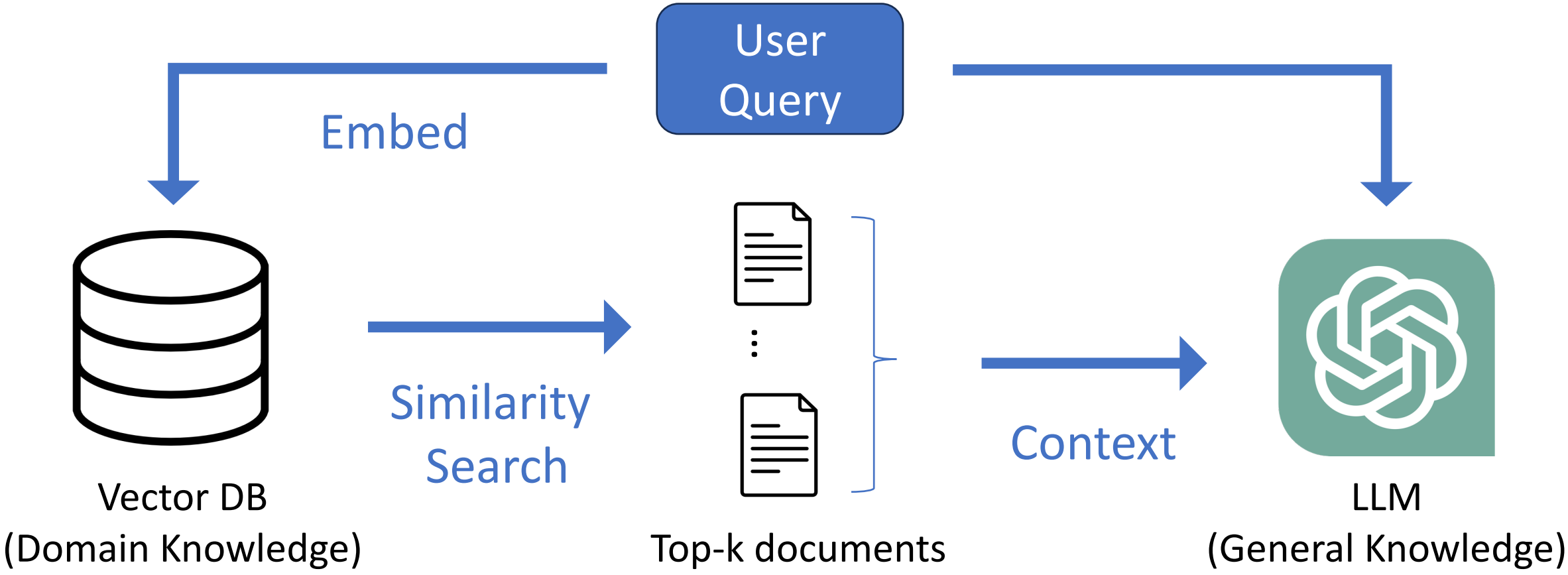


Structured and unstructured data

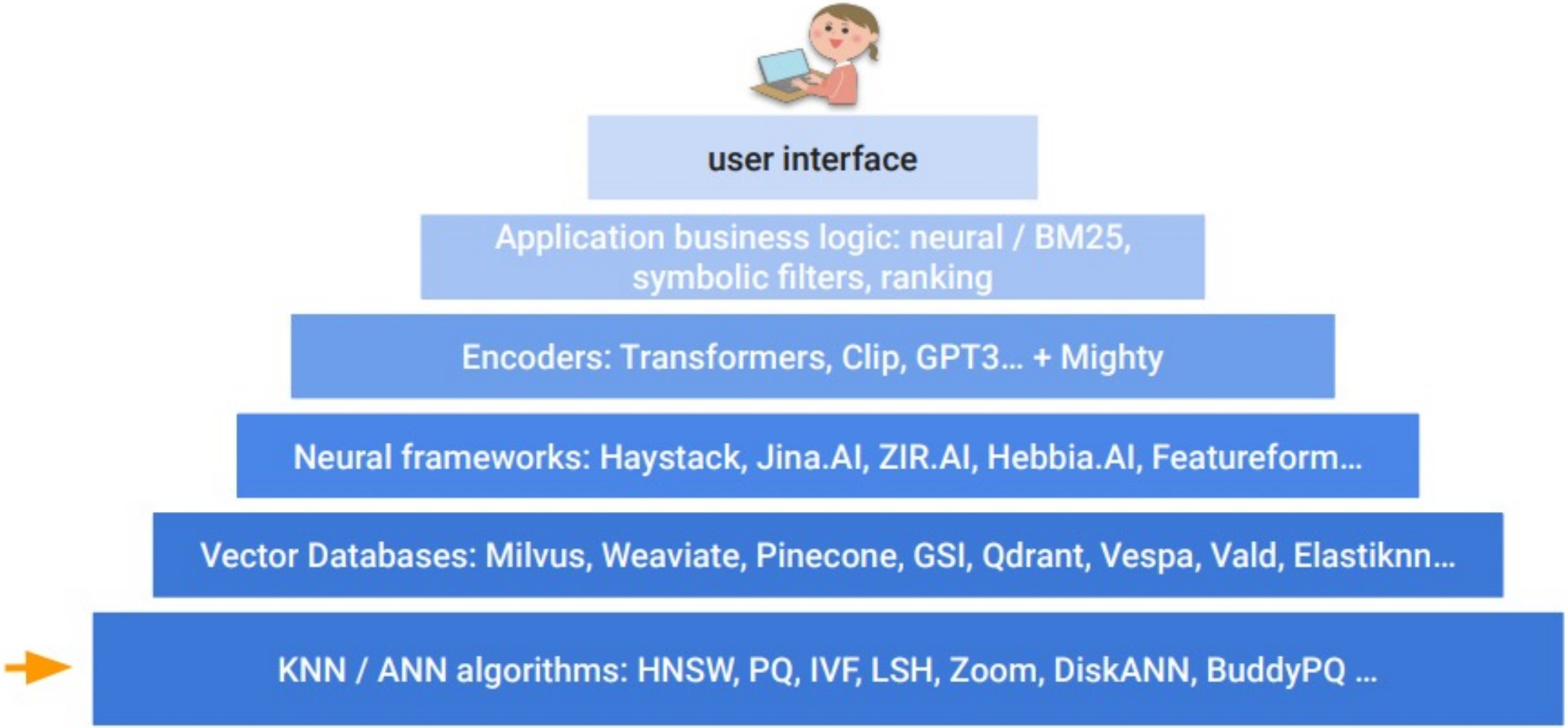
ML

embeddings

Vector Search in LLMs (retrieval augmented generation)



Vector search pyramid



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

Algorithms: Big players in the field

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

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

Nearest Neighbor Search (NNS)

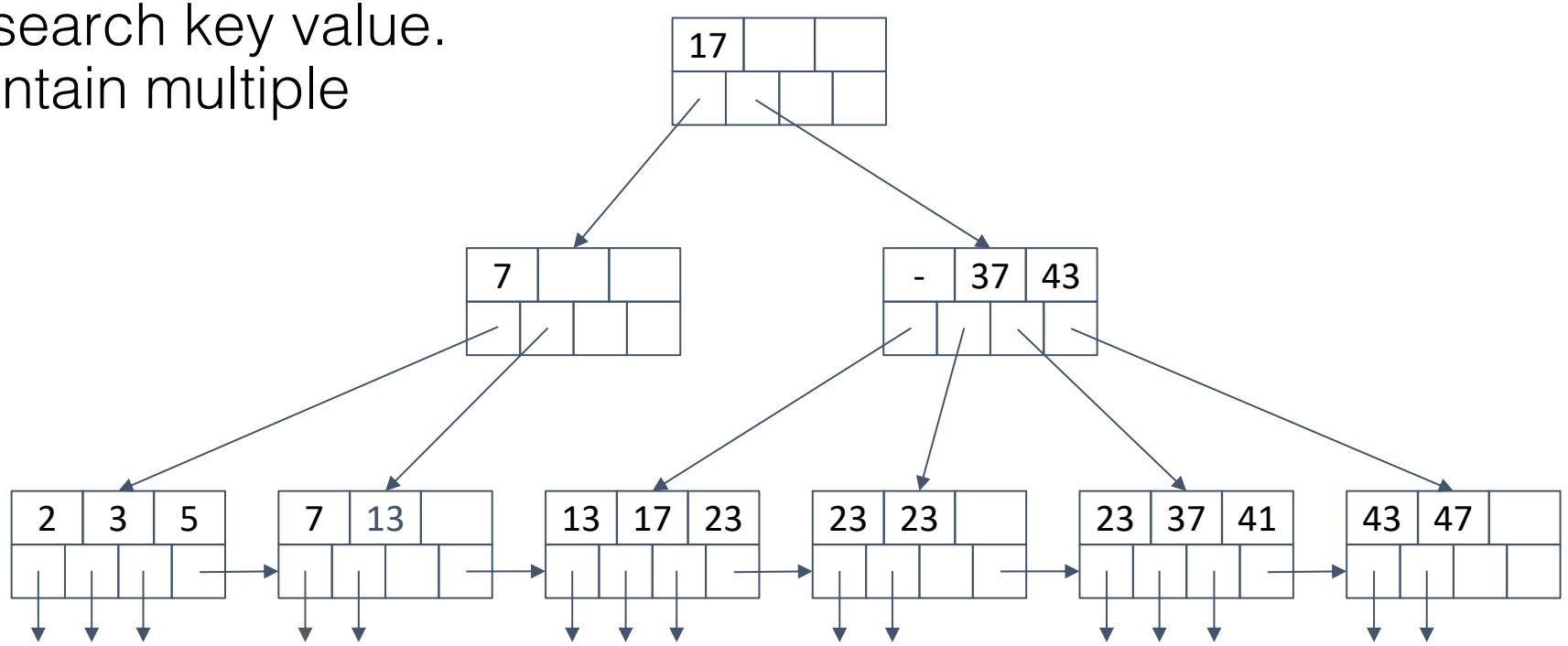
- **Problem definition**: given a query object q, we search in a massive high-dimensional dataset $\mathcal{D}$ for one or more objects in $\mathcal{D}$ that are among the closet to q according to some similarity or distance metric.

- 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

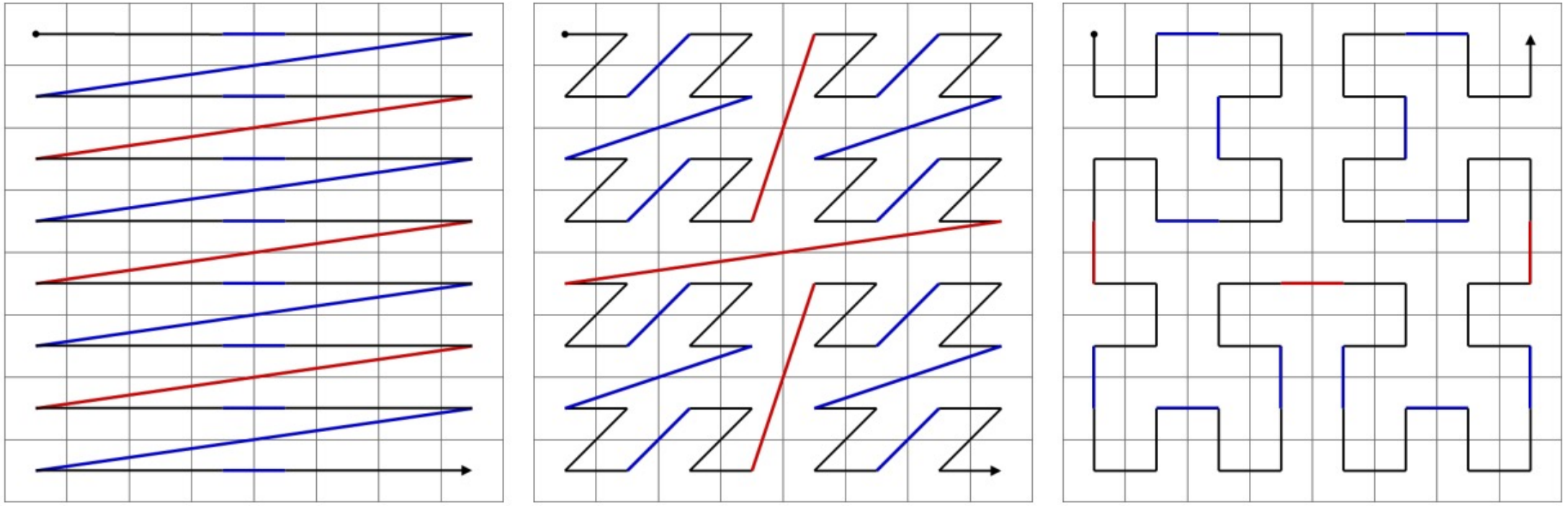
Recall that B-trees are examples of a one-dimensional index

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



One-dimensional Indexes + space-filling curves

- B-trees can be extended to support multiple dimensions using space-filling curves such as Z-order



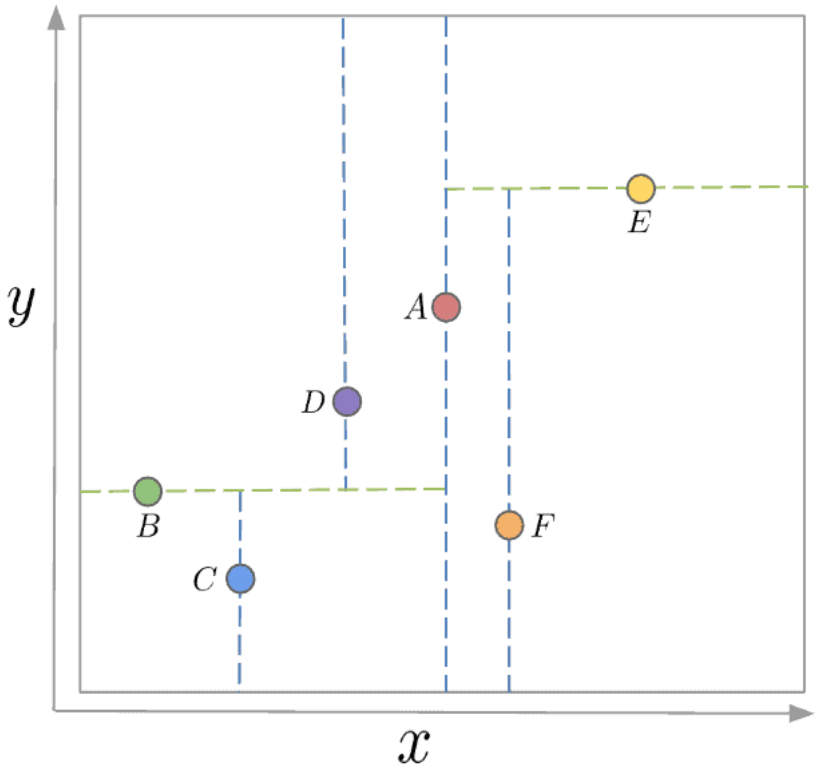
(a) Row order curve (b) Z-order curve (c) Hilbert curve

Image source: https://medium.com/@nishant.chandra/z-order-indexing-for-efficient-queries-in-data-lake-48eceaeb2320

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



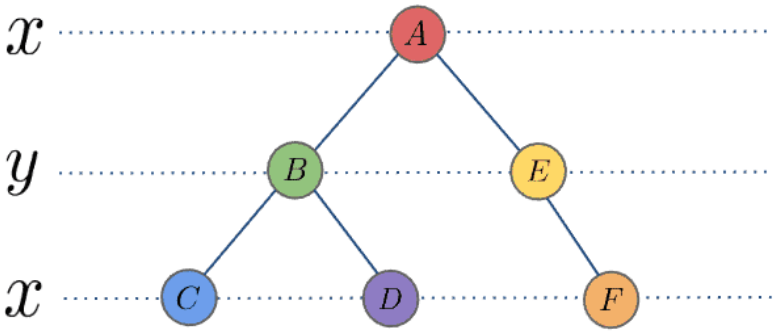
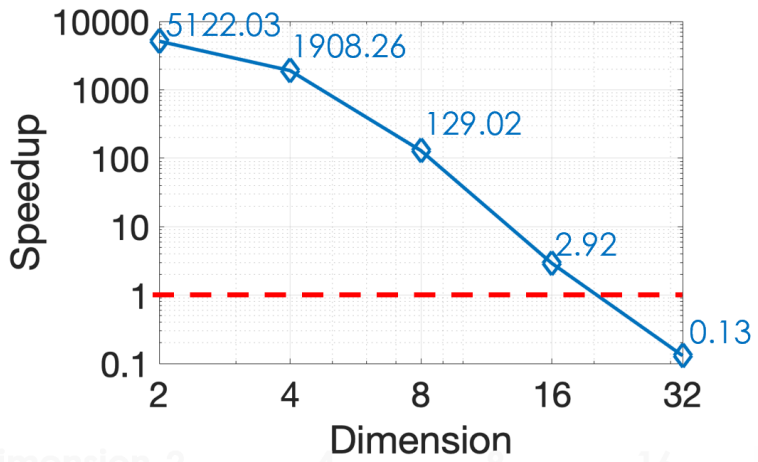


Image source: https://www.baeldung.com/cs/k-d-trees

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.



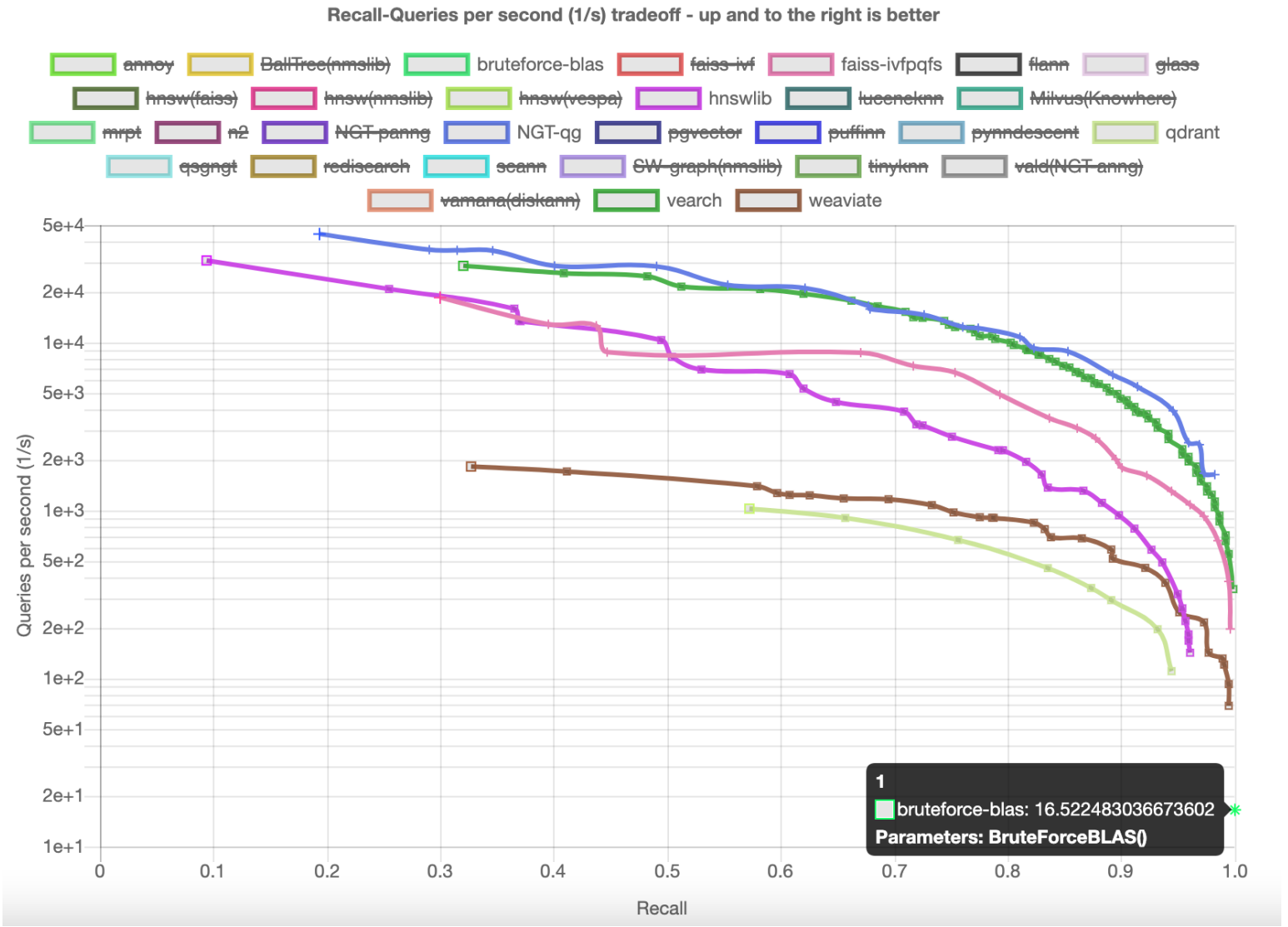
[C1994] K. L. Clarkson. An algorithm for approximate closest-point queries. In Proceedings of the Annual Symposium on Computational Geometry, pages 160–164, 1994.

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.

Approximate Nearest Neighbor Search



Exact

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

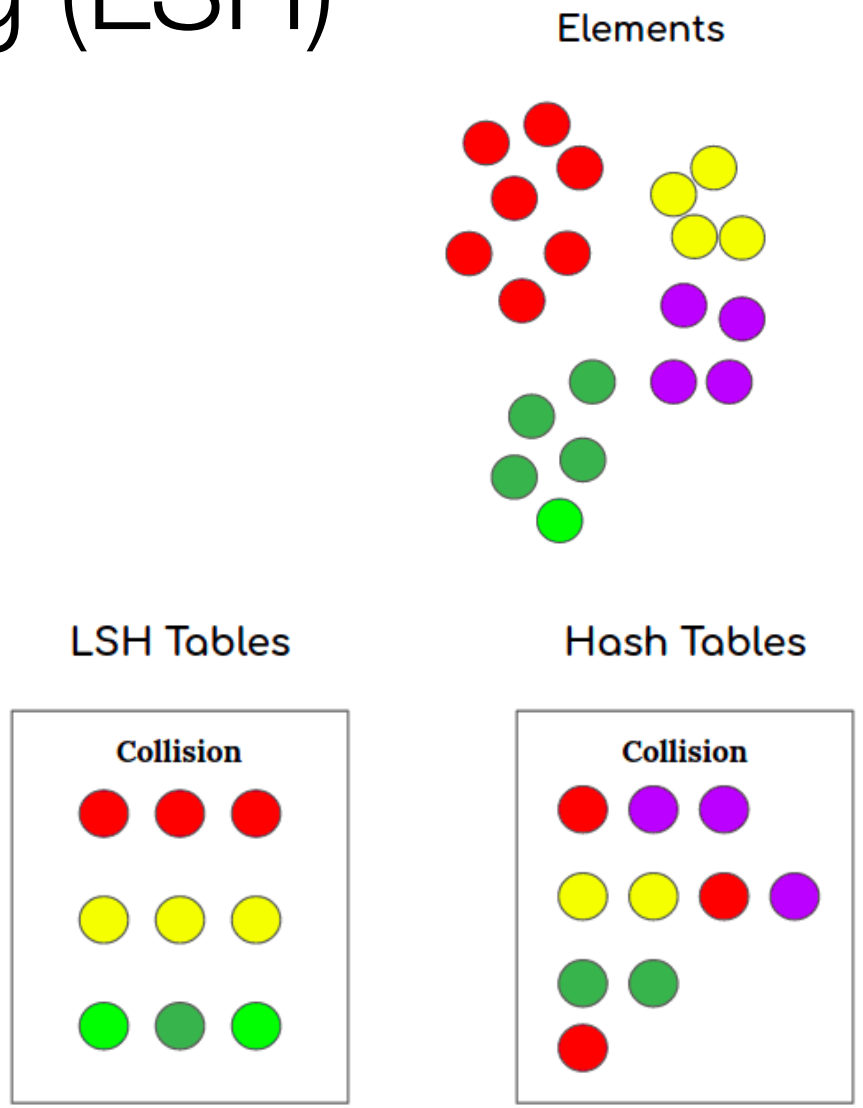
Popular ANNS Algorithms

- Locality sensitive hashing (LSH)
- Nearest neighbor graph
  - KNN graph
  - Hierarchical Navigable Small Worlds (HNSW)
- Product Quantization (PQ)

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

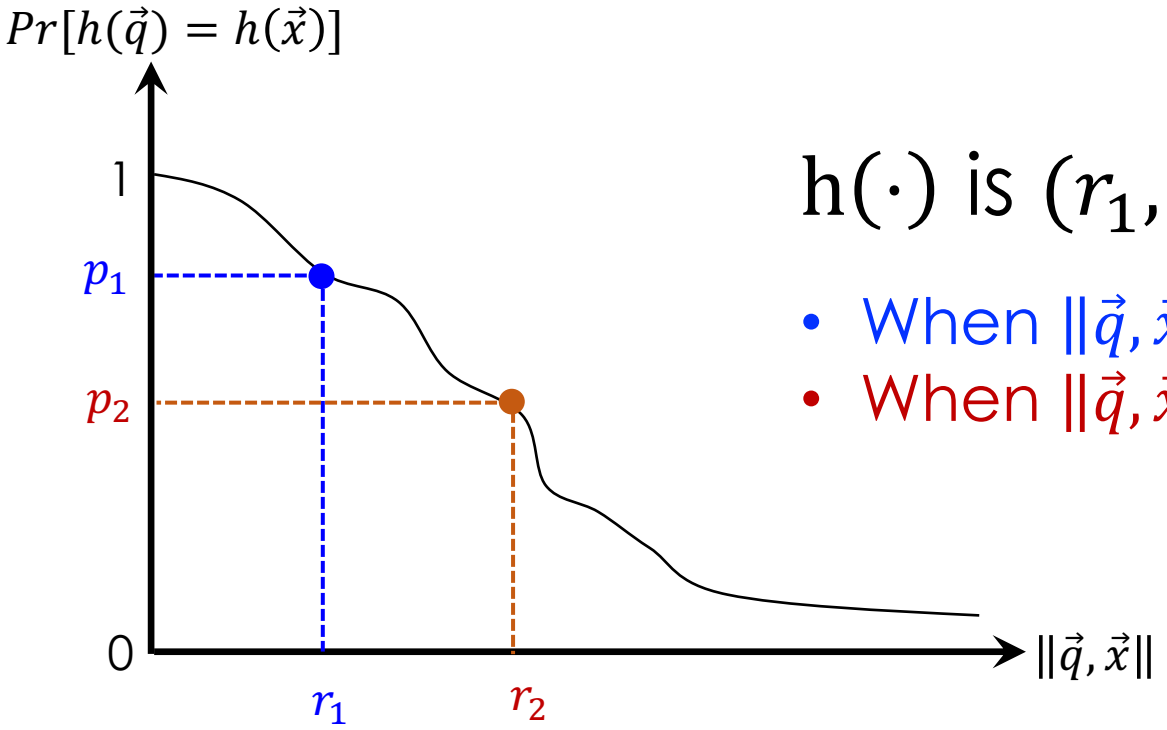
Locality sensitive hashing (LSH)

- LSH for Cosine Distance
- LSH for Jaccard Distance
- Using LSH for ANNS
- 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 $\|\vec{q}, \vec{x}\|$



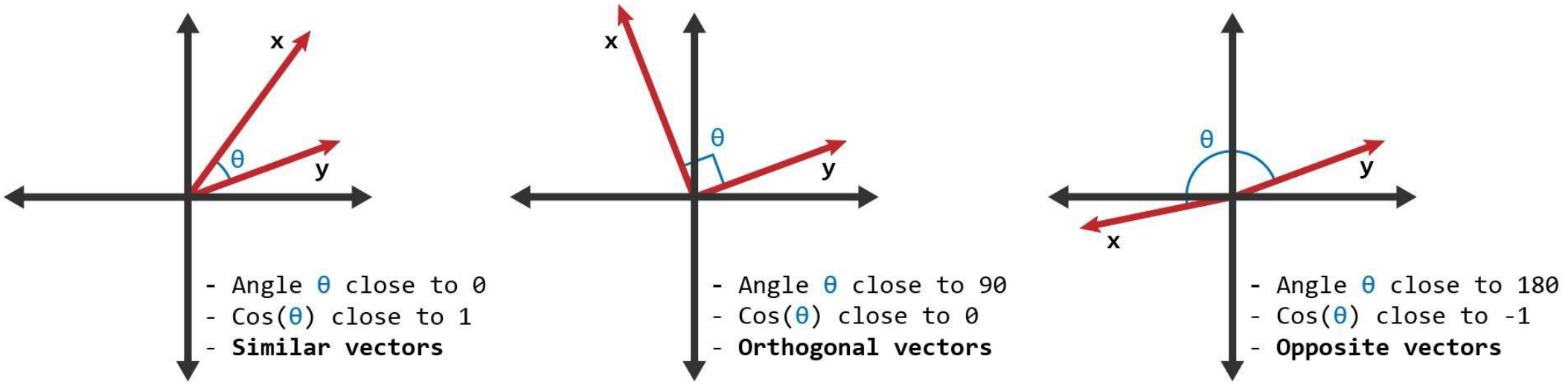
$\mathbf{h}(\cdot)$ is (r_1, r_2, p_1, p_2) –sensitive

- 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*.



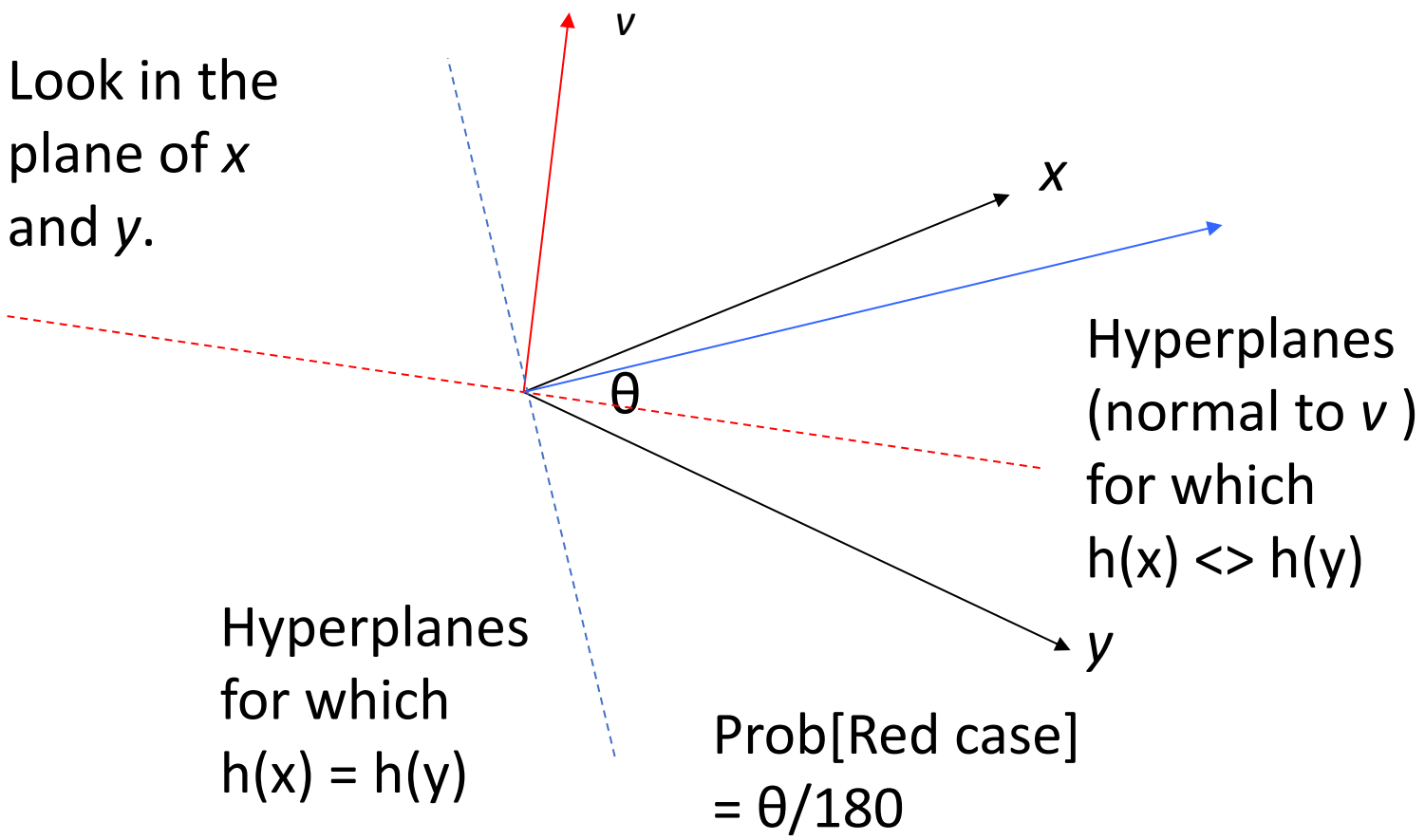
Random Hyperplanes

- 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.

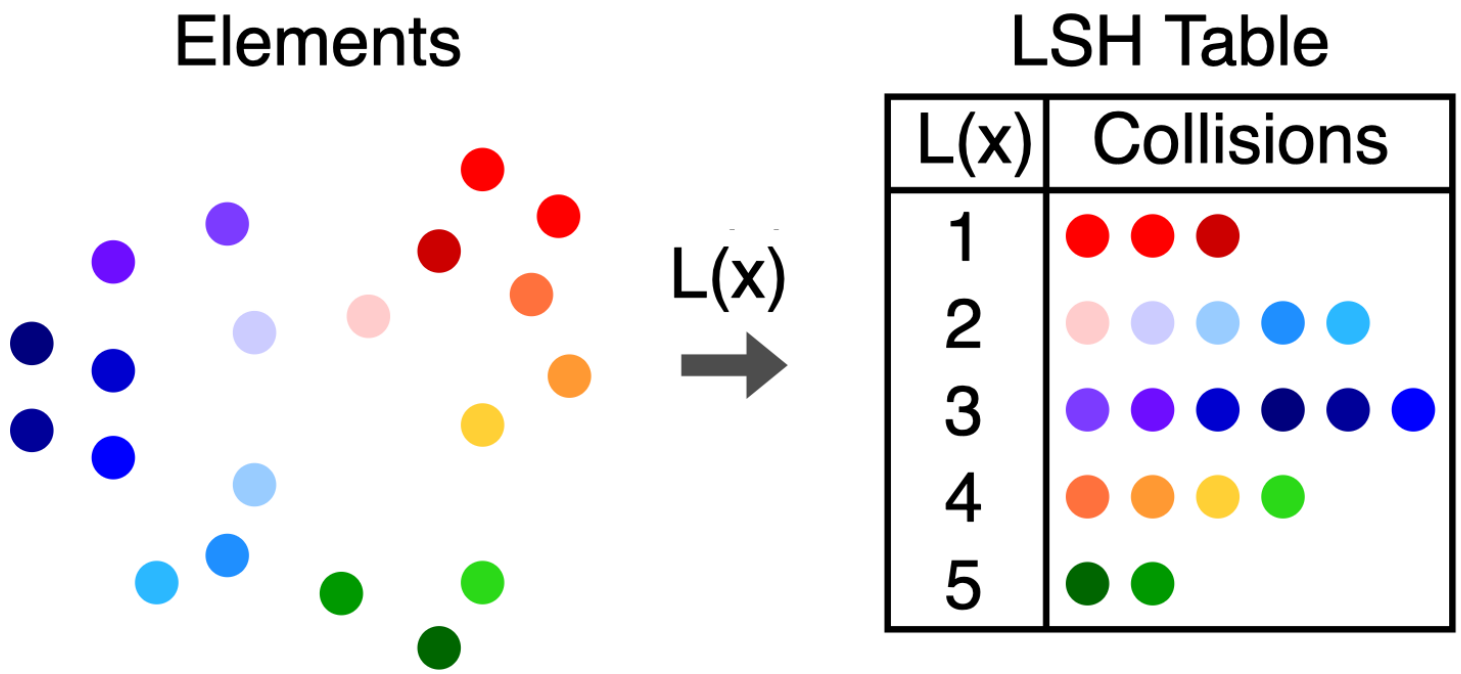
- Claim:
  - $Prob[h(x) = h(y)] = 1 - \frac{\theta}{180}, \cos\theta = \frac{x \cdot y}{|x||y|}$

Proof of Claim

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



How to use LSH in ANNS?

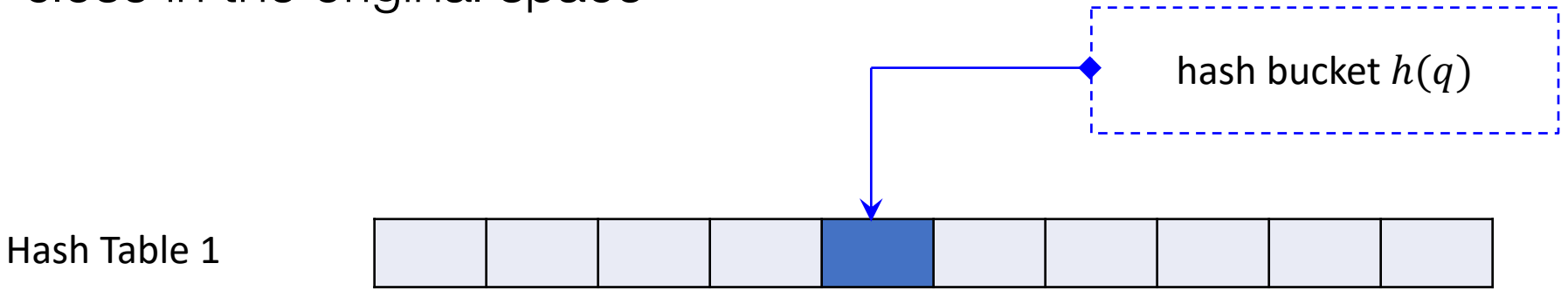


- Main idea: Only check data points that *hash collides* with the query (instead of the entire dataset)
  - Two points are close in the projected space are likely to be close in the original space

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 $P[h_1(x) = h_1(q) \& h_2(x) = h_2(q) \& \ldots \& h_M(x) = h_M(q)]$?

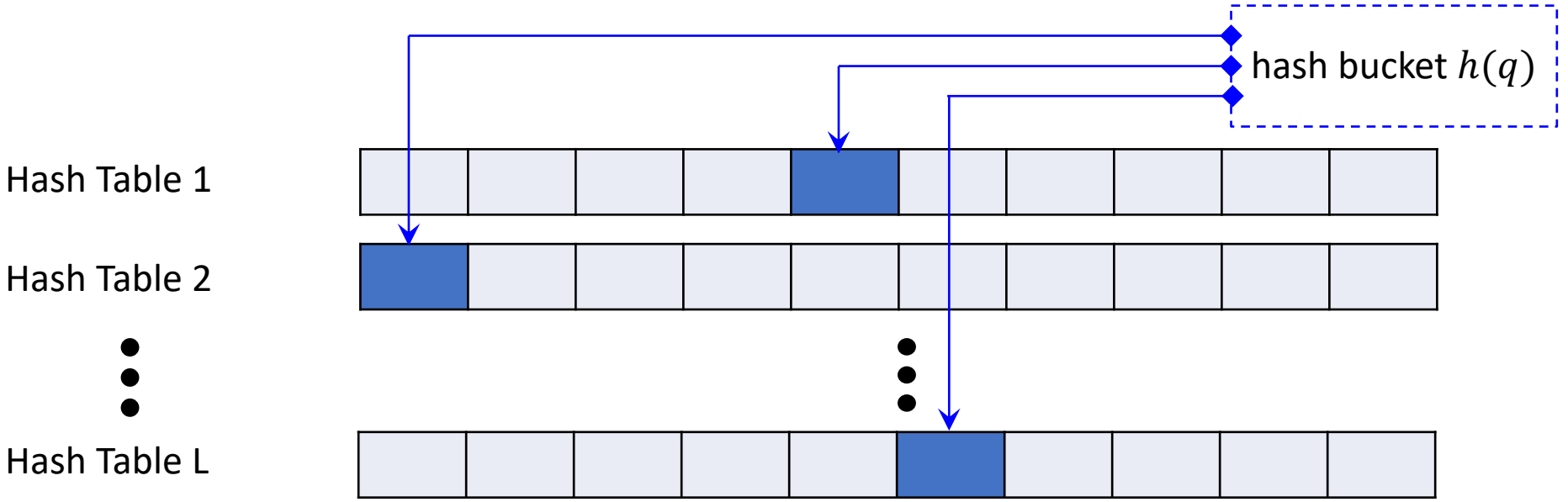
Collision probability reduces to p^M

- Harder for false positives to result in a hash collision

  => increase precision
- Q: What about recall?

LSH-based ANNS

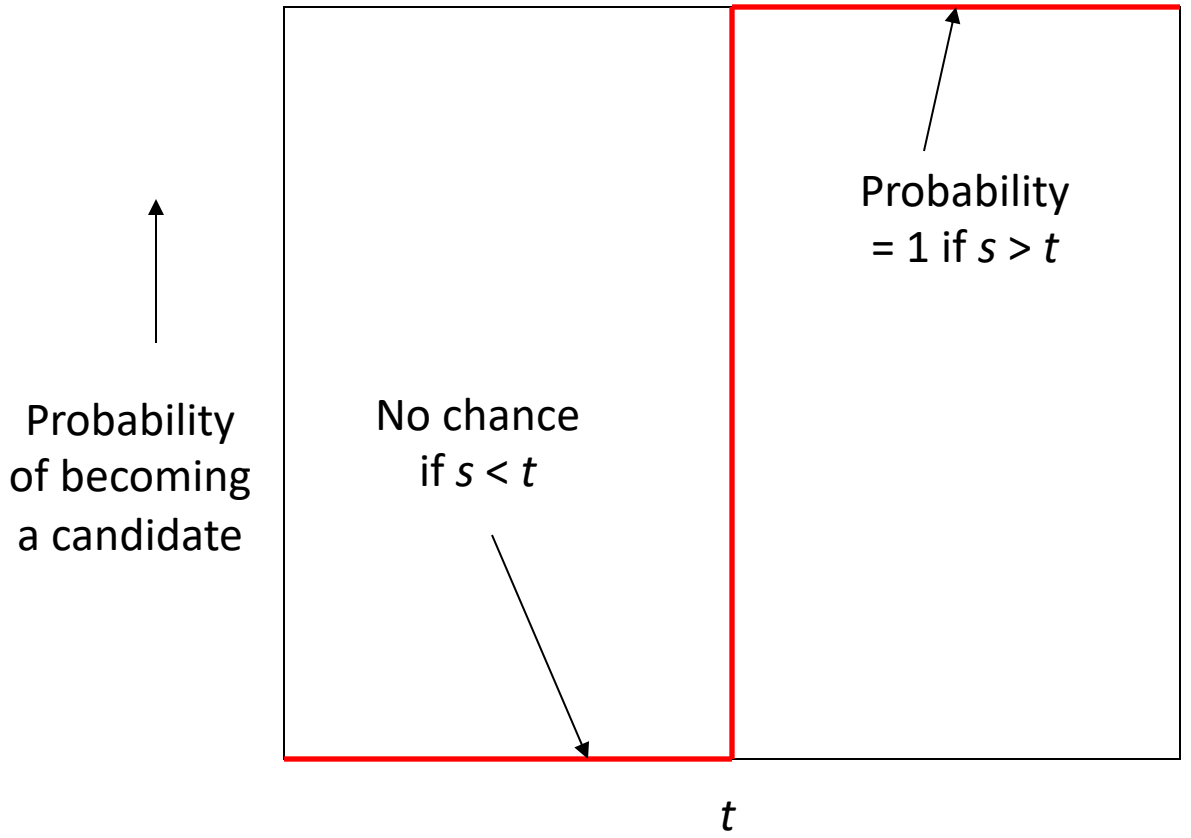
- How to increase recall:
  - Repeat multiple times. Consider a data point an NN candidate if it hash collides with the query in any trial
- Build L hash tables
  - Each table generates hash signatures using M random hyperplanes: $\overrightarrow{\mathbf{h}(\cdot)} \triangleq < \mathbf{h}_1(\cdot), \mathbf{h}_2(\cdot),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



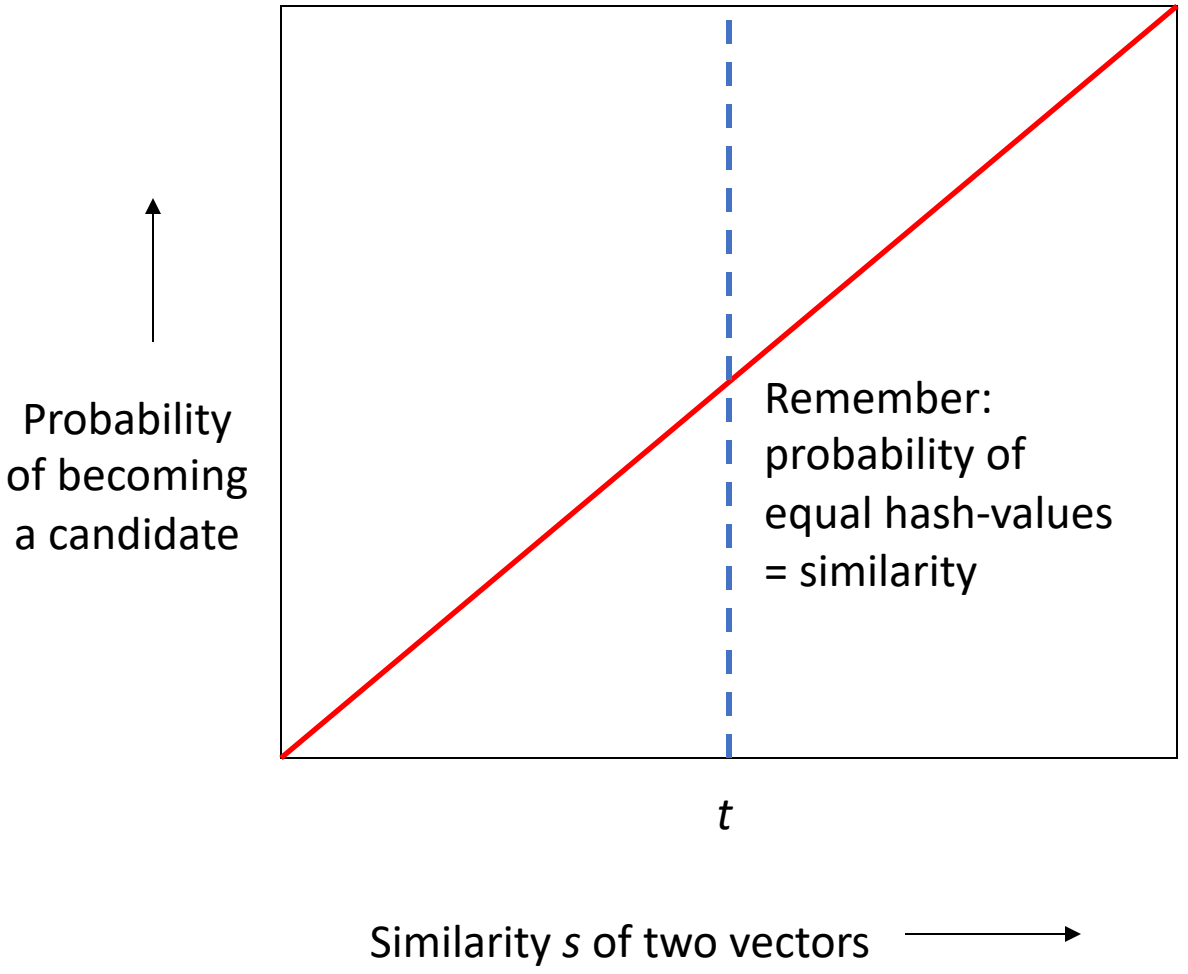
Key Parameters in LSH

- 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?

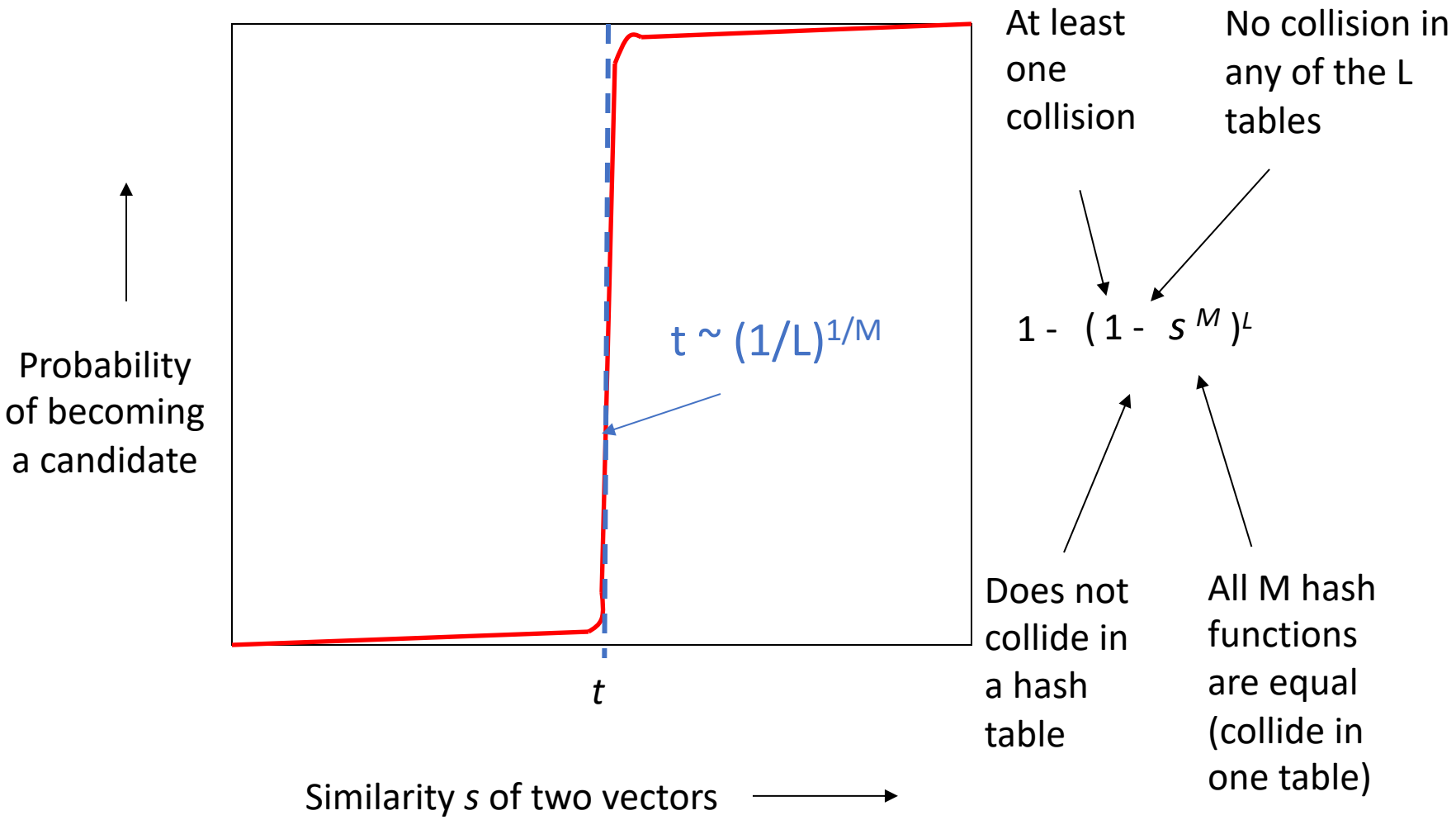
Analysis of LSH – What We Want



Single hash function (one random hyperplane)

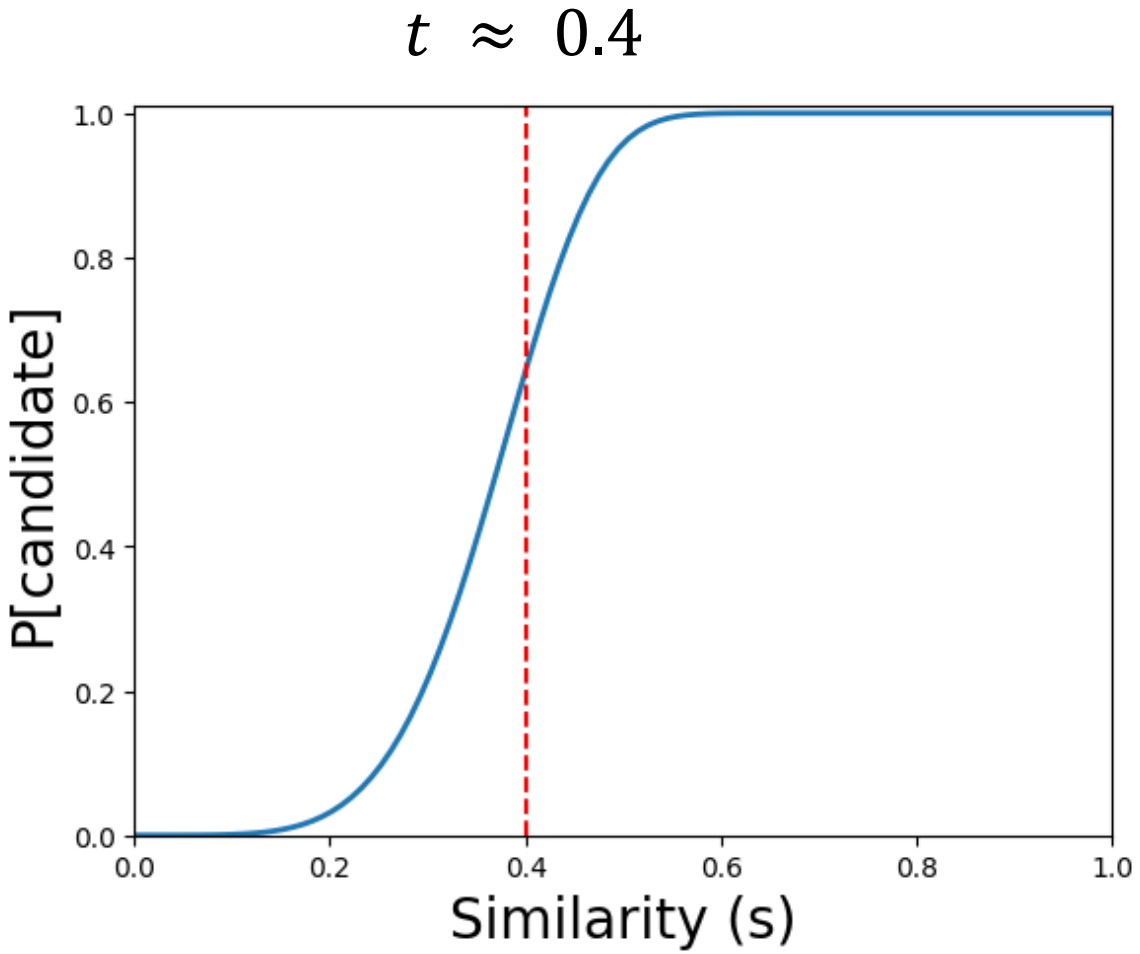


M hash functions, L tables



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

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



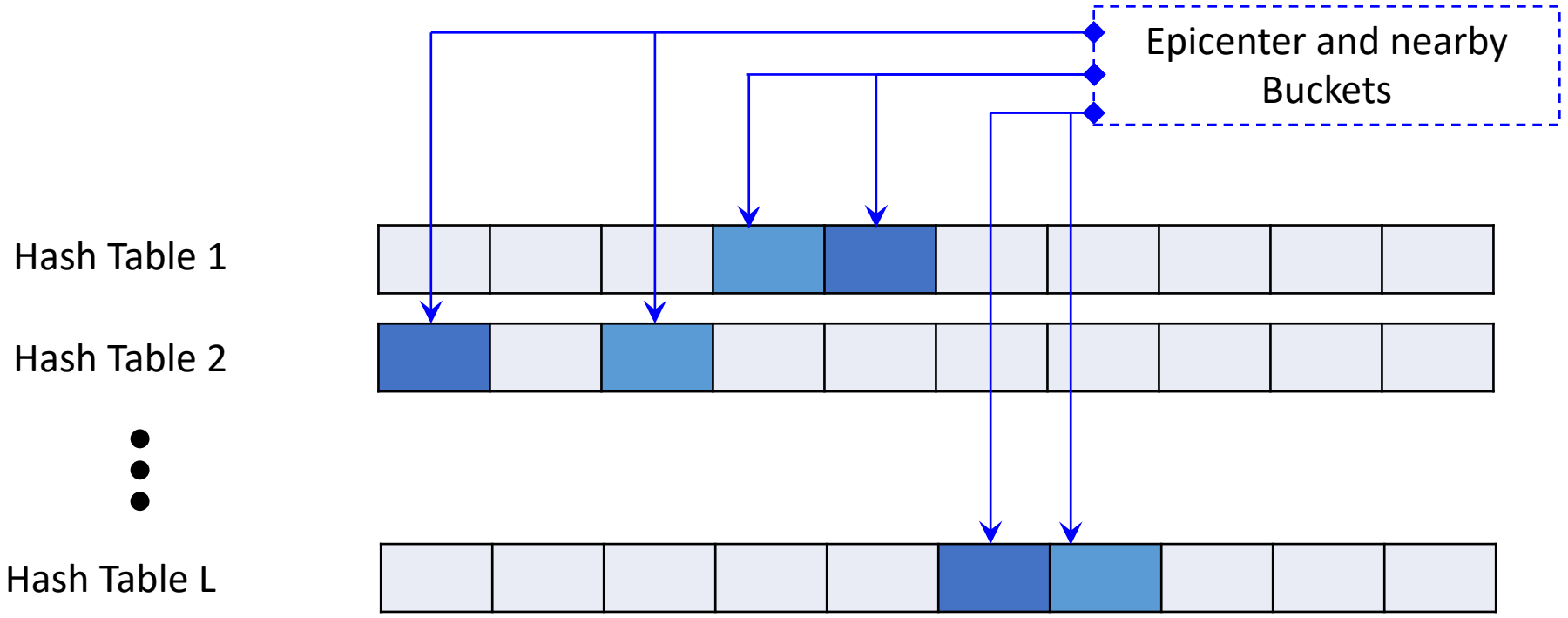
Multi-probe LSH [VLDB'07]

- 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]

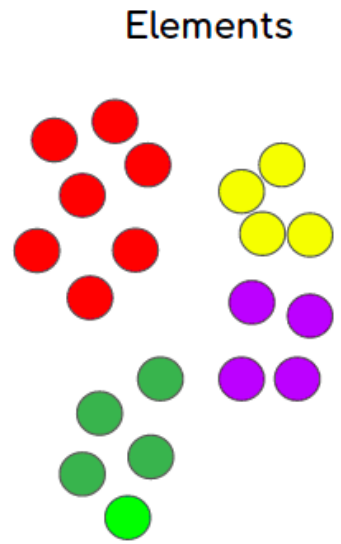
- In addition to the epicenter bucket, multi-probe LSH also probes T nearby buckets whose success probabilities (of finding nearest neighbor of $\vec{q}$) are among the T +1 highest.

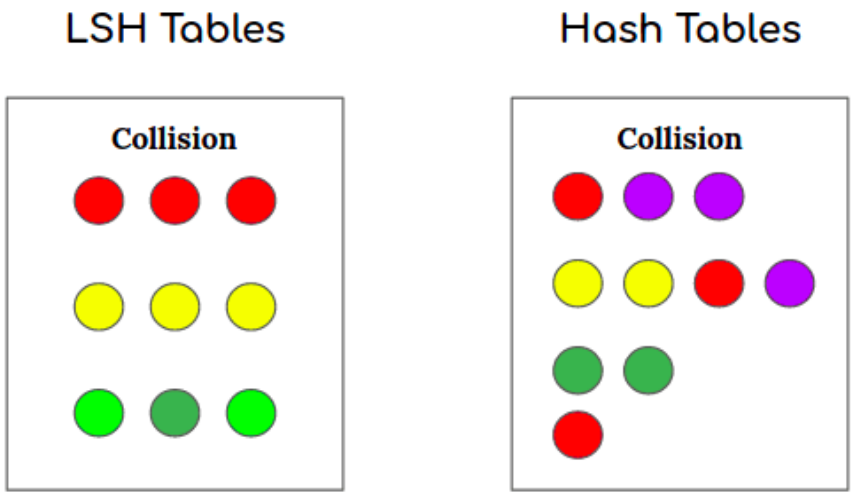


Significantly reduce L by probing "best" nearby buckets!

Locality sensitive hashing (LSH)

- LSH for Cosine Distance
- Using LSH for ANNS
- Tuning parameters in LSH
- LSH for Jaccard Distance





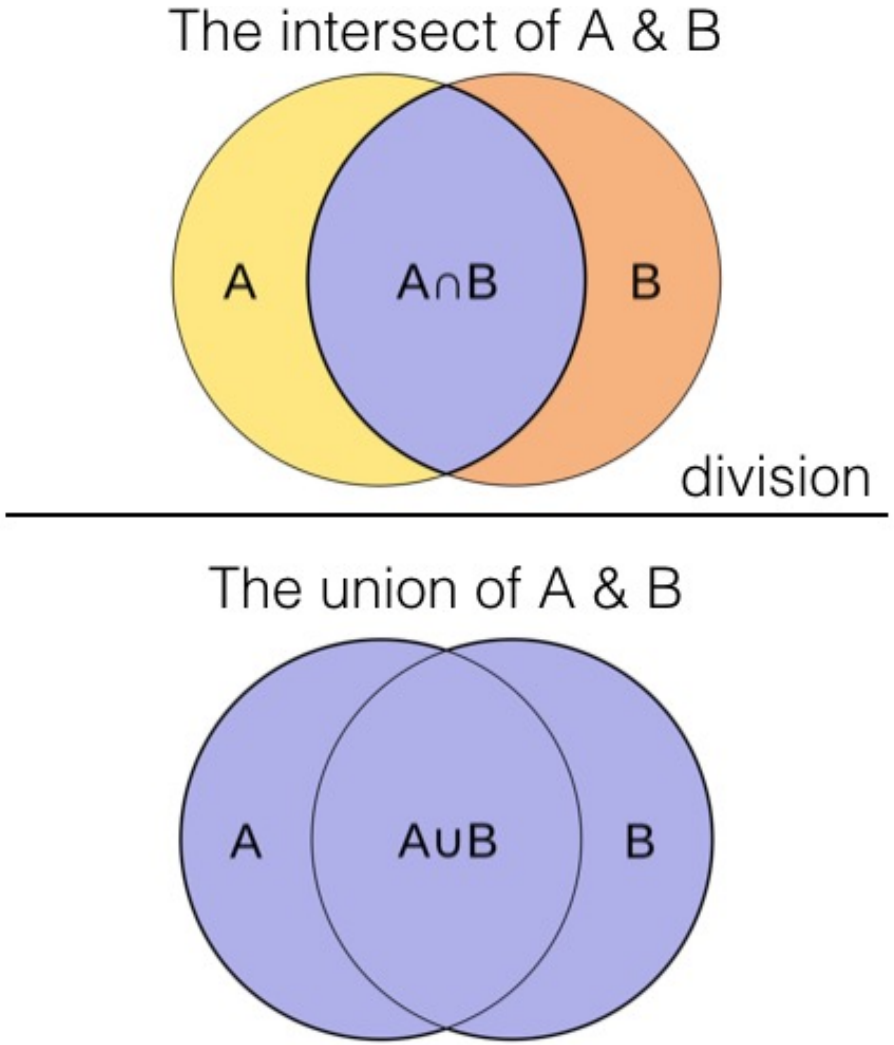
Jaccard Similarity

Given two sets A and B

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

e.g., disjoint sets have similarity 0,

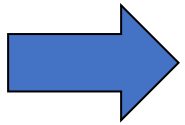
J(A, A) = 1



1 – J(A, B) is a metric (satisfying triangle inequality)

Jaccard Similarity

$S_1 = \{A, B, F, G\}$
$S_2 = \{C, D, E, F\}$
$S_3 = \{A, F, G\}$
$S_4 = \{B, C, D, E\}$



Document

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

Binary Matrix Representation

Q: $J(S_1, S_3)$?

$S_1 \cap S_3 = \{A, F, G\}$
$S_1 \cup S_3 = \{A, B, F, G\}$

Q: Where do we use Jaccard?

* data is not actually stored this way

LSH for Jaccard Similarity

MinHash:

- Given a universe U, pick a permutation π on U uniformly at random
- Hash each set to the minimum value it contains according to π

Example using document deduplication:

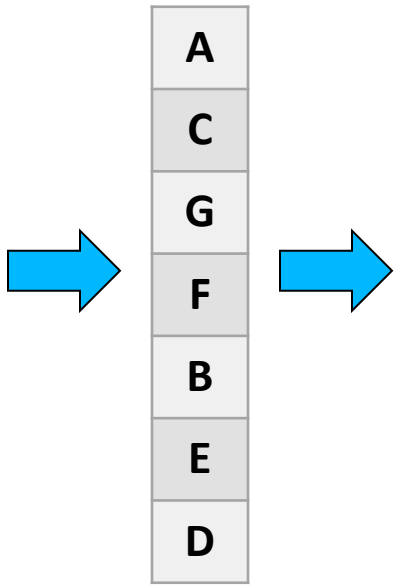
- Randomly order all words.
- Represent each document by the earliest appearing word in the random ordering that the document contains.

Claim: MinHash forms an LSH for Jaccard Similarity

Example: Minhash signatures

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

Input Matrix



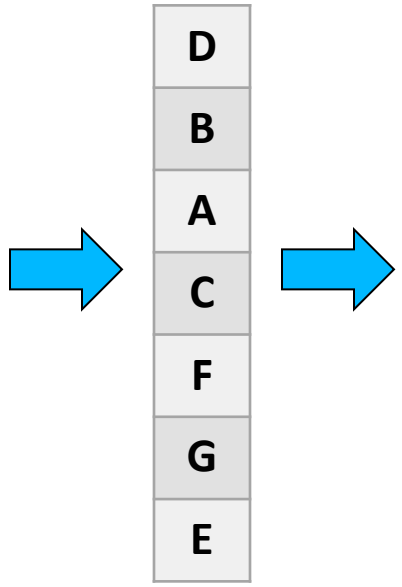
Random permutations

		S_1	S_2	S_3	S_4
1	**A**	1	0	1	0
2	**C**	0	1	0	1
3	**G**	1	0	1	0
4	**F**	1	0	1	0
5	**B**	1	0	0	1
6	**E**	0	1	0	1
7	**D**	0	1	0	1
		1	**2**	**1**	**2**

Signature Matrix

Example: Minhash signatures

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0



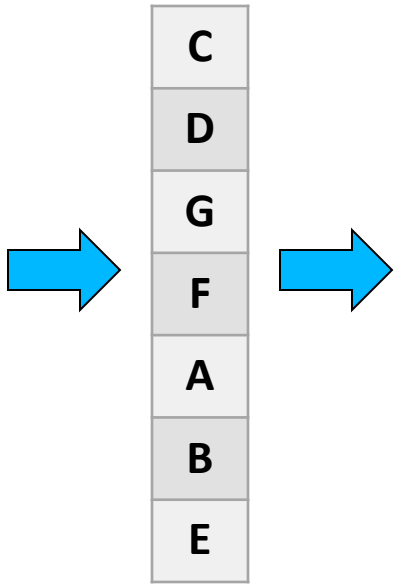
		S_1	S_2	S_3	S_4
1	**D**	0	1	0	1
2	**B**	1	0	0	1
3	**A**	1	0	1	0
4	**C**	0	1	0	1
5	**F**	1	0	1	0
6	**G**	1	0	1	0
7	**E**	0	1	0	1
		2	**1**	**3**	**1**

Random permutations

Example: Minhash signatures

Q: What's the hash signature of S_1?

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0



Random permutations

		S_1	S_2	S_3	S_4
1	**C**	0	1	0	1
2	**D**	0	1	0	1
3	**G**	1	0	1	0
4	**F**	1	0	1	0
5	**A**	1	0	1	0
6	**B**	1	0	0	1
7	**E**	0	1	0	1
		3	**1**	**3**	**1**

Example: Minhash signatures

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

≈

Signature matrix

	S_1	S_2	S_3	S_4
h_1	1	2	1	2
h_2	2	1	3	1
h_3	3	1	3	1

- Sig(S) = vector of hash values
  - e.g., Sig(S_2) = [2,1,1]
- Sig(S,i) = value of the i-th hash function for set S
  - E.g., Sig(S_2,3) = 1

Similarity for Signatures

- The similarity of signatures is the fraction of the hash functions in which they agree.

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

≈

Signature matrix

	S_1	S_2	S_3	S_4
h_1	1	2	1	2
h_2	2	1	3	1
h_3	3	1	3	1

Zero similarity is preserved
High similarity is well approximated

	Actual	Sig
(S_1, S_2)	0	0
(S_1, S_3)	3/5	2/3
(S_1, S_4)	1/7	0
(S_2, S_3)	0	0
(S_2, S_4)	3/4	1
(S_3, S_4)	0	0

- With multiple signatures we get a good approximation

Adapted from https://www.cs.bu.edu/~gkollios/cs660f19/Slides/minhashLSH.pdf

Minhash function property

$$P\left(h(S_i) = h(S_j)\right) = Jaccard(S_i, S_j)$$

where the probability is over all choices of permutations.

Why?

- The first row where one of the two sets has value 1 belongs to the union.
  - Recall that union contains rows with at least one 1.
- We have equality if both sets have value 1, and this row belongs to the intersection

Minhash Implementation

Consider the naïve implementation using permutations:

- Assume a billion rows
- Hard to pick a random permutation of 1…billion
- Even representing a random permutation requires 1 billion entries!

- Accessing rows in permuted order leads to thrashing.

Minhash Implementation

- Instead of permuting the rows we will apply a hash function that maps the rows to a new (possibly larger) space
  - The value of the hash function is the position of the row in the new order (permutation).
  - Each set is represented by the smallest hash value among the elements in the set

- The space of the hash functions should be such that if we select a function at random, each element (row) has equal probability to have the smallest value
  - Min-wise independent hash functions

Minhash Implementation

For each row r

- Compute $h_1(r), h_2(r), \ldots, h_n(r)$
- For each column c do the following:
  - If c has 0 in row r, do nothing.
  - Otherwise, $Sig(i,c) = \min\left(Sig(i,c), h_i(r)\right), i = 1, 2, \ldots, n$

Example

x	Row	S1	S2	$h_1(x)$	$h_2(x)$
0	A	1	0	1	3
1	B	0	1	2	0
2	C	1	1	3	2
3	D	1	0	4	4
4	E	0	1	0	1

$h_1(x) = x+1 \bmod 5$
$h_2(x) = 2x+3 \bmod 5$

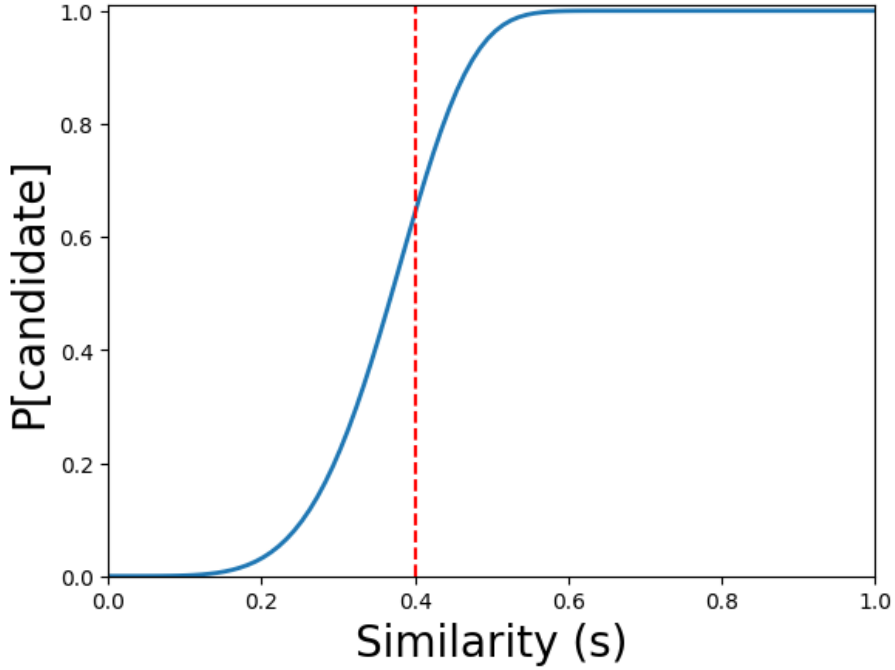
h_1(Row)	Row	S1	S2
0	E	0	1
1	A	1	0
2	B	0	1
3	C	1	1
4	D	1	0

h_2(Row)	Row	S1	S2
0	B	0	1
1	E	0	1
2	C	1	1
3	A	1	0
4	D	1	0

	Sig(S1)	Sig(S2)	
$h_1(0) = 1$	1	-	S2 does not
$h_2(0) = 3$	3	-	contain A
$h_1(1) = 2$	1	2	
$h_2(1) = 0$	3	0	
$h_1(2) = 3$	1	2	
$h_2(2) = 2$	2	0	
$h_1(3) = 4$	1	2	
$h_2(3) = 4$	2	0	
$h_1(4) = 0$	1	0	
$h_2(4) = 1$	2	0	

The same ANNS and parameter tuning procedures apply to Minhash

- 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



Popular ANNS Algorithms

- Locality sensitive hashing (LSH)
- Nearest neighbor graph
  - KNN graph
  - Hierarchical Navigable Small Worlds (HNSW)
- Product Quantization (PQ)

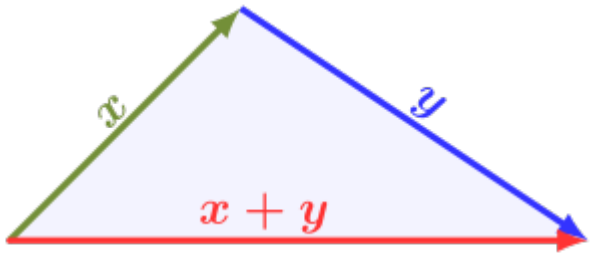
VectorDB	ANN library	ANN algorithm
Milvus	Custom FAISS	PQ
Pinecone	Custom FAISS	LSH, PQ
Qdrant	Custom HNSW	NN graph
Pgvector	Custom HNSW	NN graph

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]

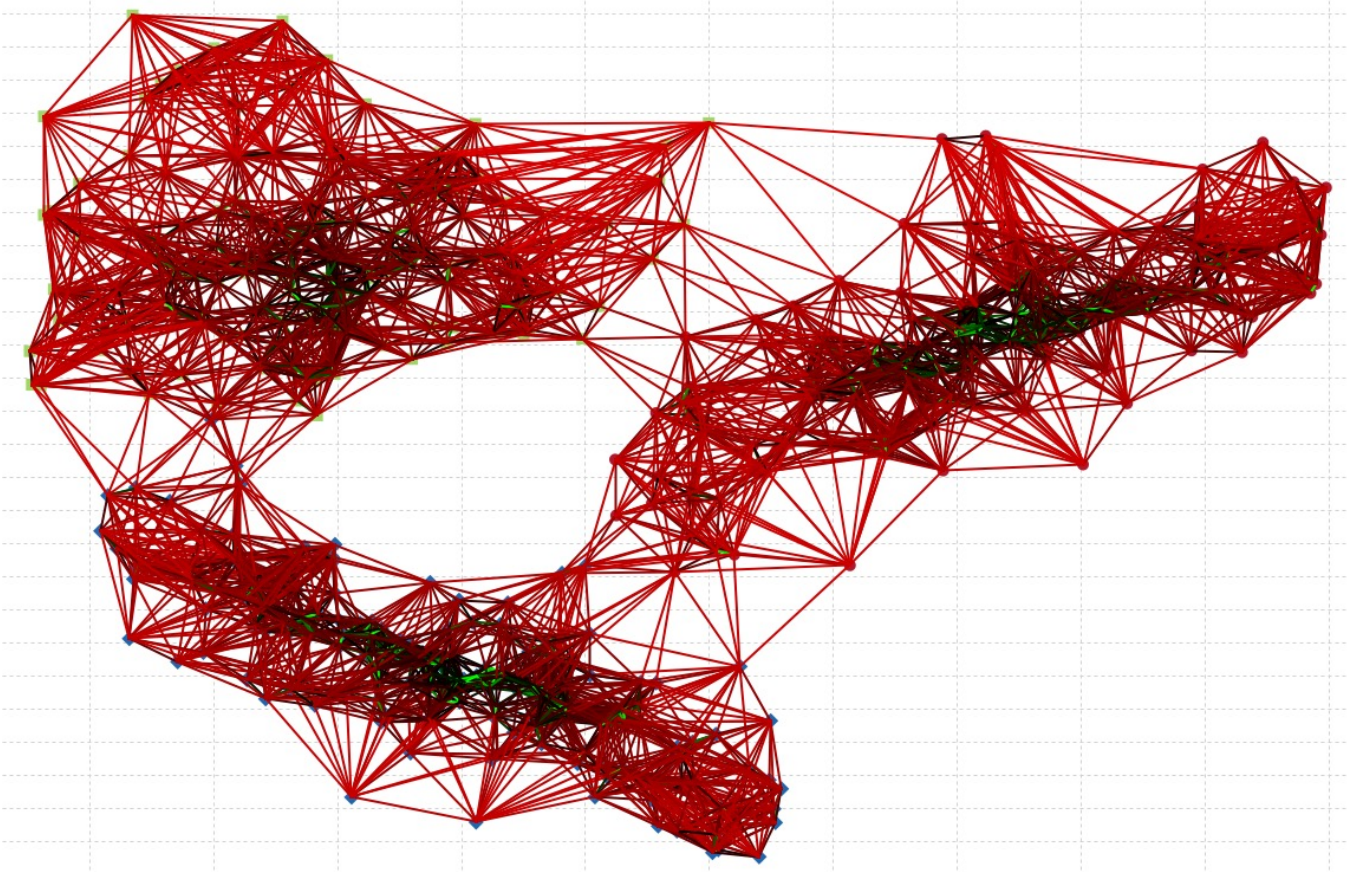
- In the search stage, graph-based algorithms find the candidate neighbors of a query point in some way (e.g., random selection) and then check the neighbors of these candidate neighbors for closer ones iteratively.

- 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.

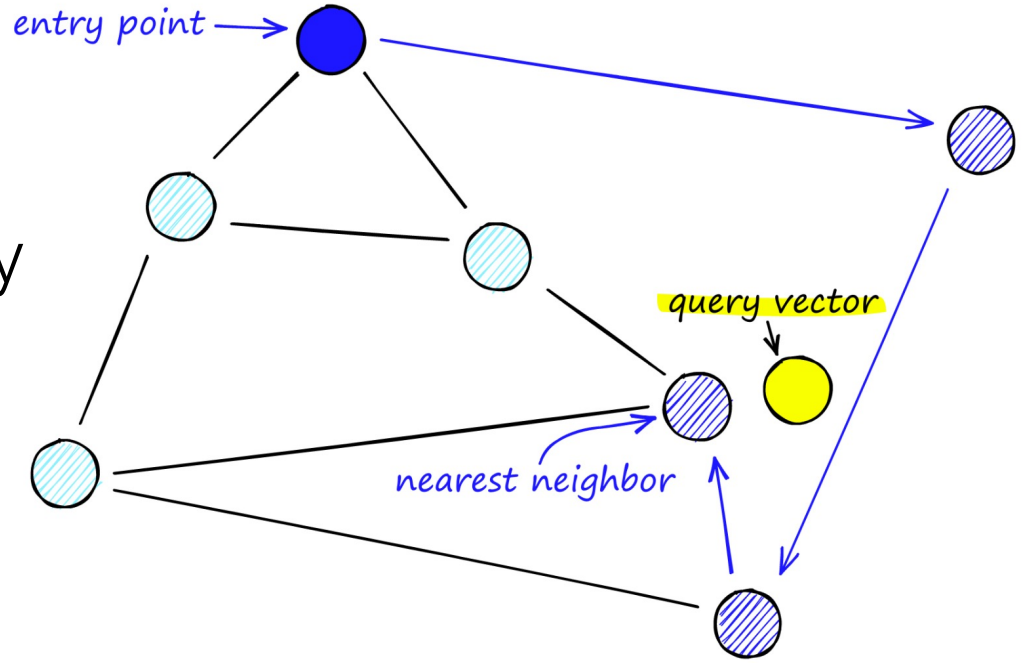


Navigable Small Worlds (NSW)

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

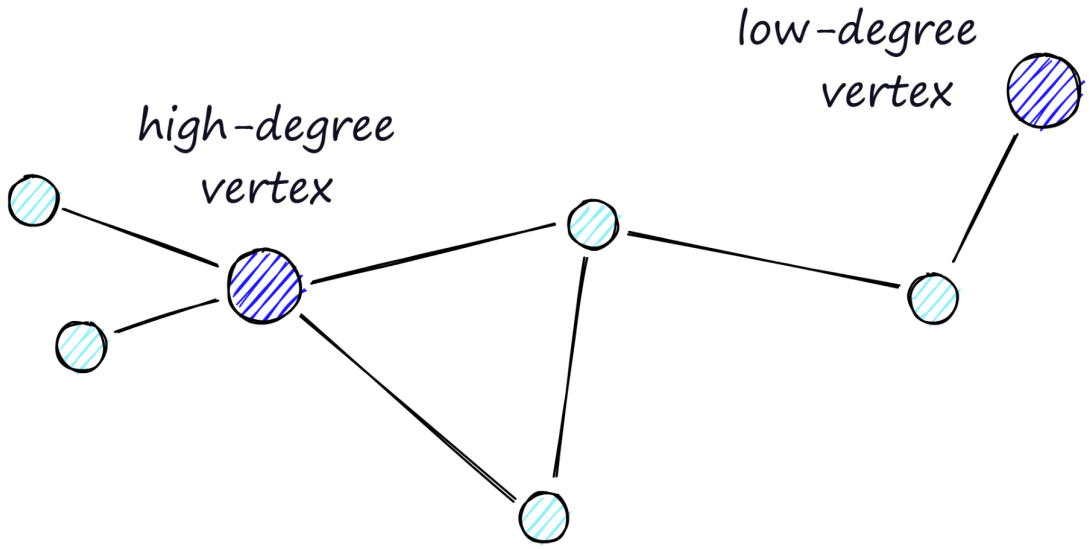
Source: https://www.pinecone.io/learn/series/faiss/hnsw/

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



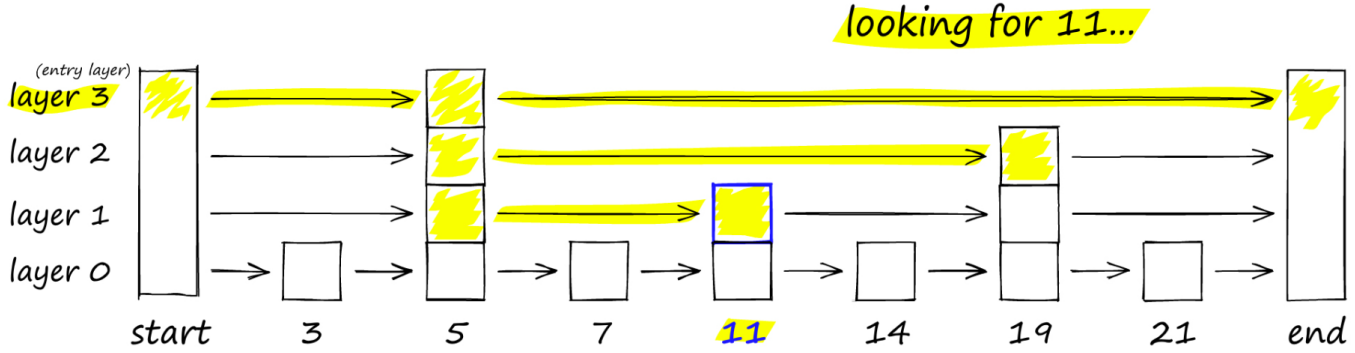
High-degree vertices have *many* links, whereas low-degree vertices have very *few* links.

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.

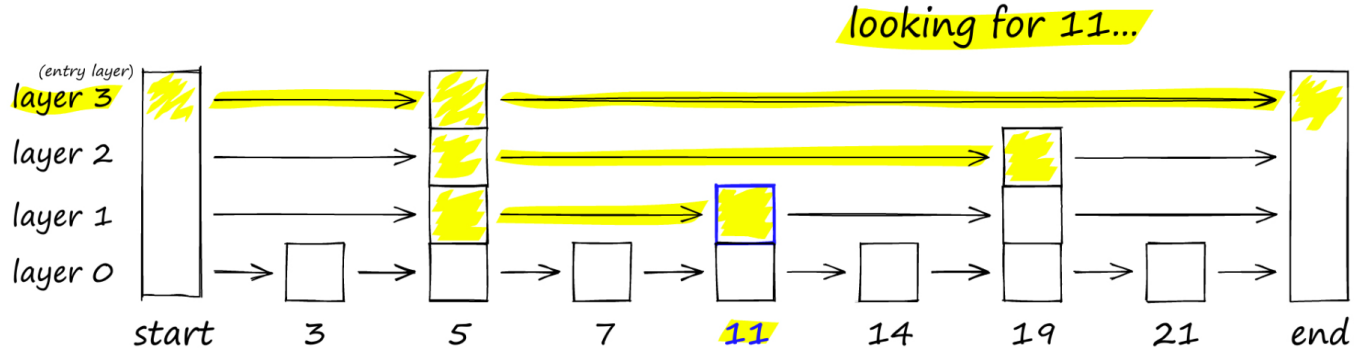


Source: https://www.pinecone.io/learn/series/faiss/hnsw/

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

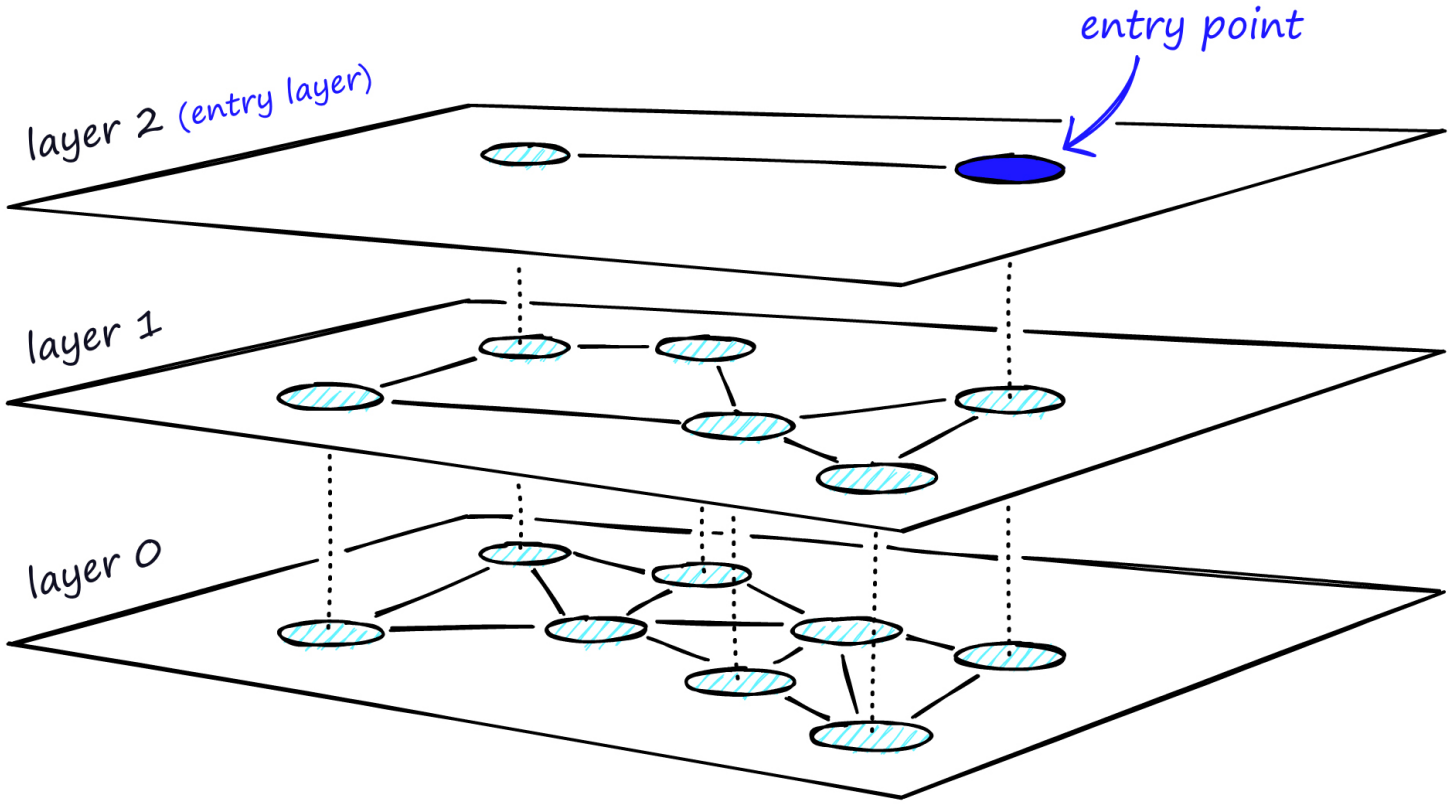


Source: https://www.pinecone.io/learn/series/faiss/hnsw/

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

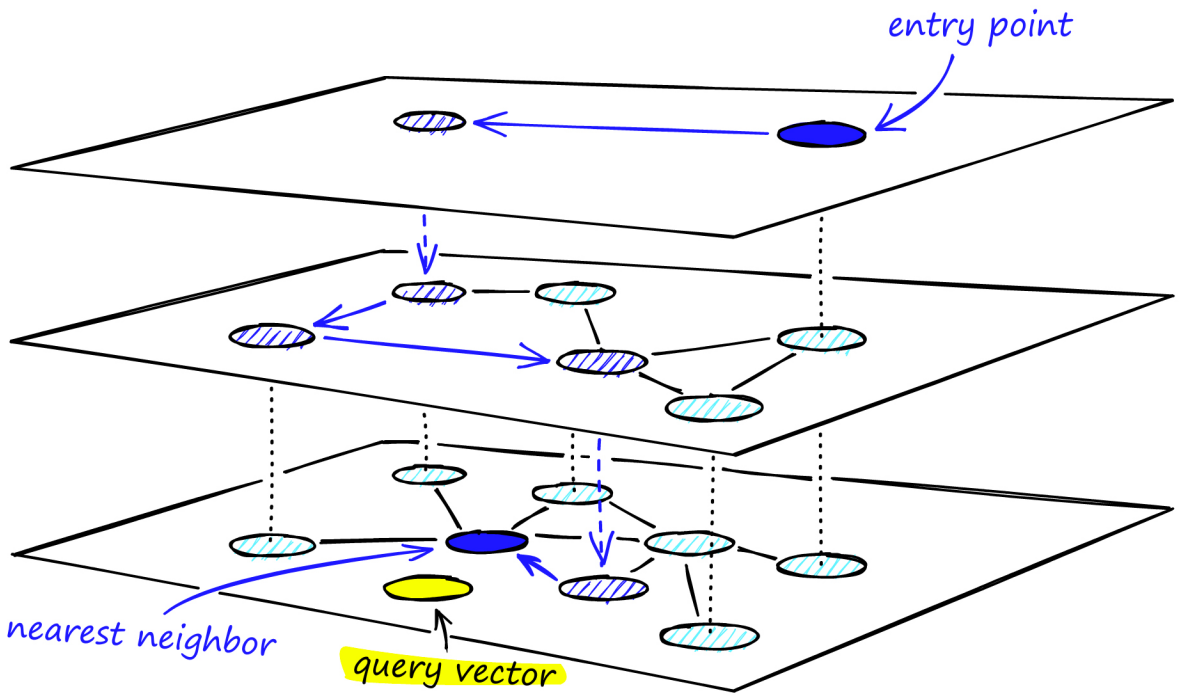


Source: https://www.pinecone.io/learn/series/faiss/hnsw/

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/

Popular ANNS Algorithms

- Locality sensitive hashing (LSH)
- Nearest neighbor graph
  - KNN graph
  - Hierarchical Navigable Small Worlds (HNSW)
- Product Quantization (PQ)

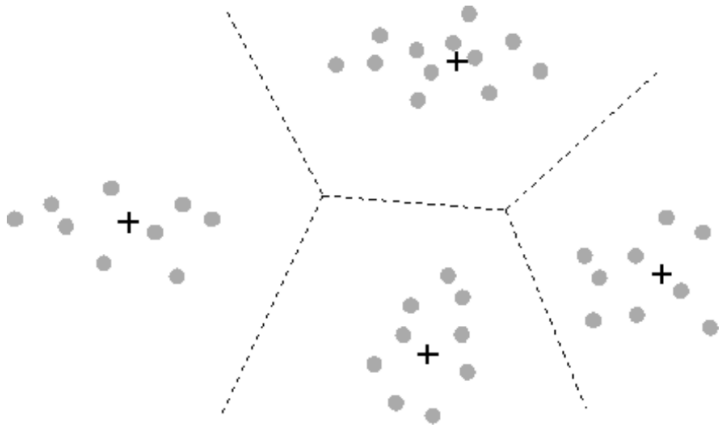
VectorDB	ANN library	ANN algorithm
Milvus	Custom FAISS	PQ
Pinecone	Custom FAISS	LSH, PQ
Qdrant	Custom HNSW	NN graph
Pgvector	Custom HNSW	NN graph

Product Quantization

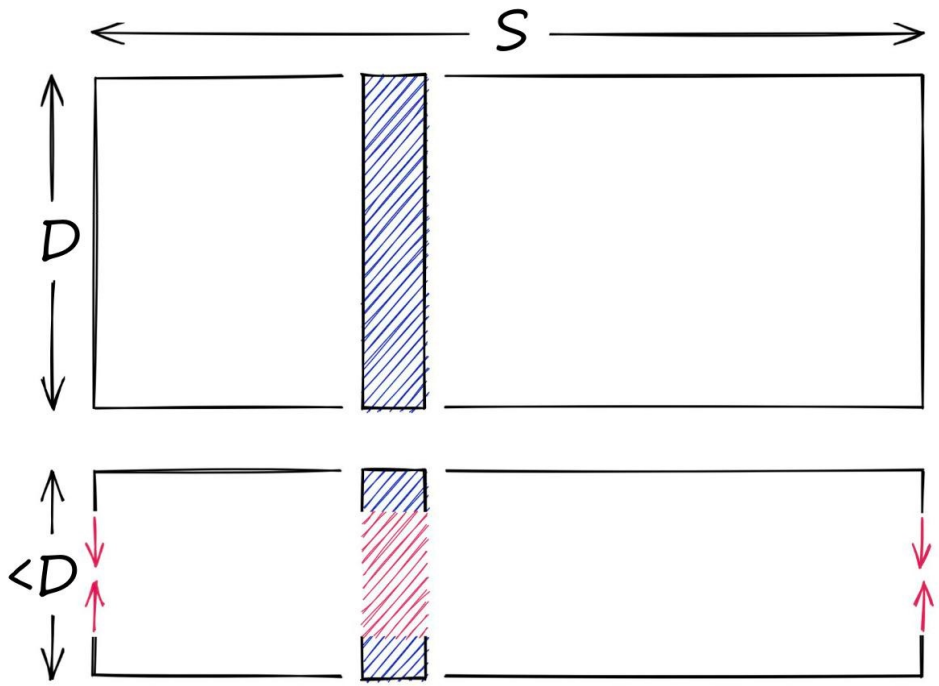
Winner in BigANN Competition @ NeurIPS' 21; a technique for compression high-dimensional vectors, therefore speeding up the similarity search.

Vector Quantization: use centroids to represent vectors in clusters.

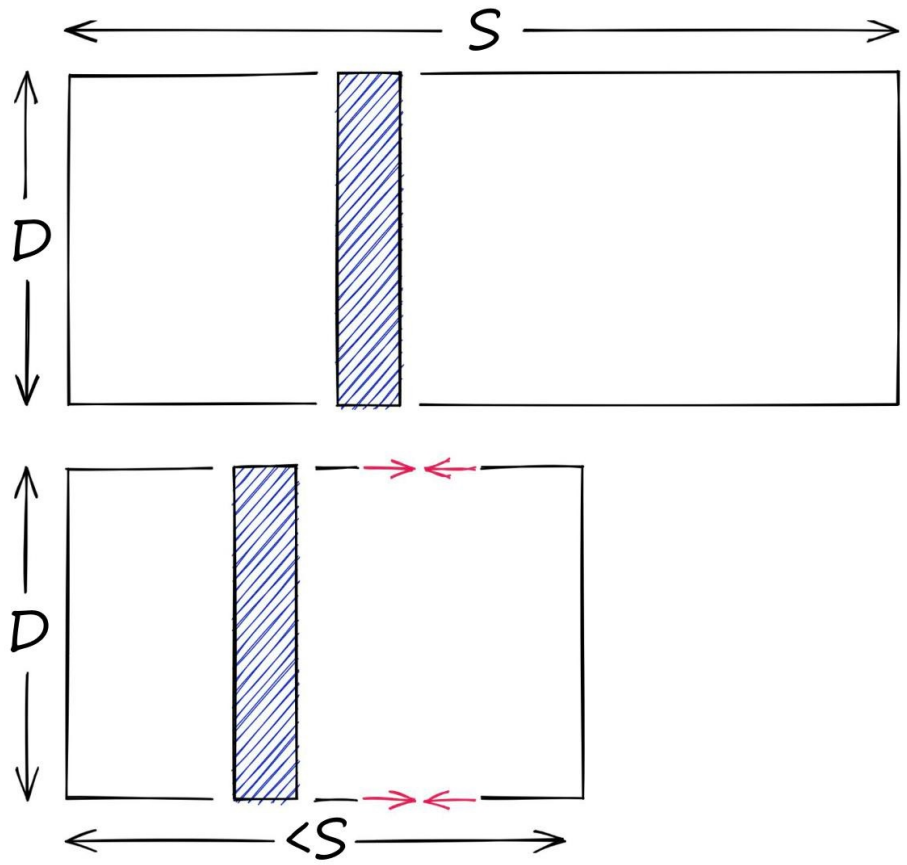
- distance(query, vector) ~ distance(query, centroid)



Dimensionality Reduction vs Quantization



Reducing the dimensionality

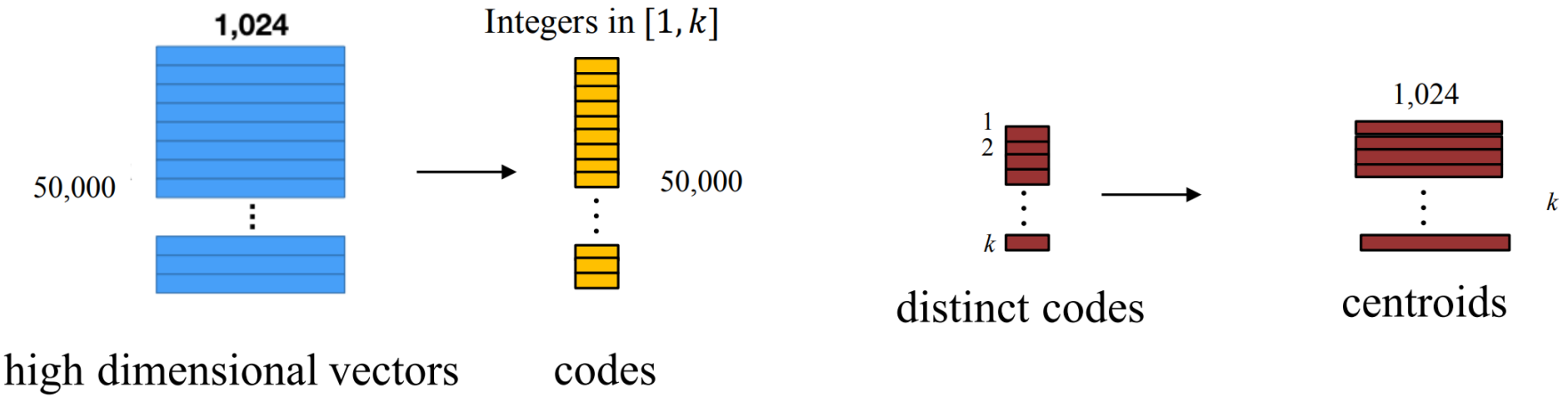


Reducing the range of values

Source: https://www.pinecone.io/learn/series/faiss/product-quantization/

Vector Quantization

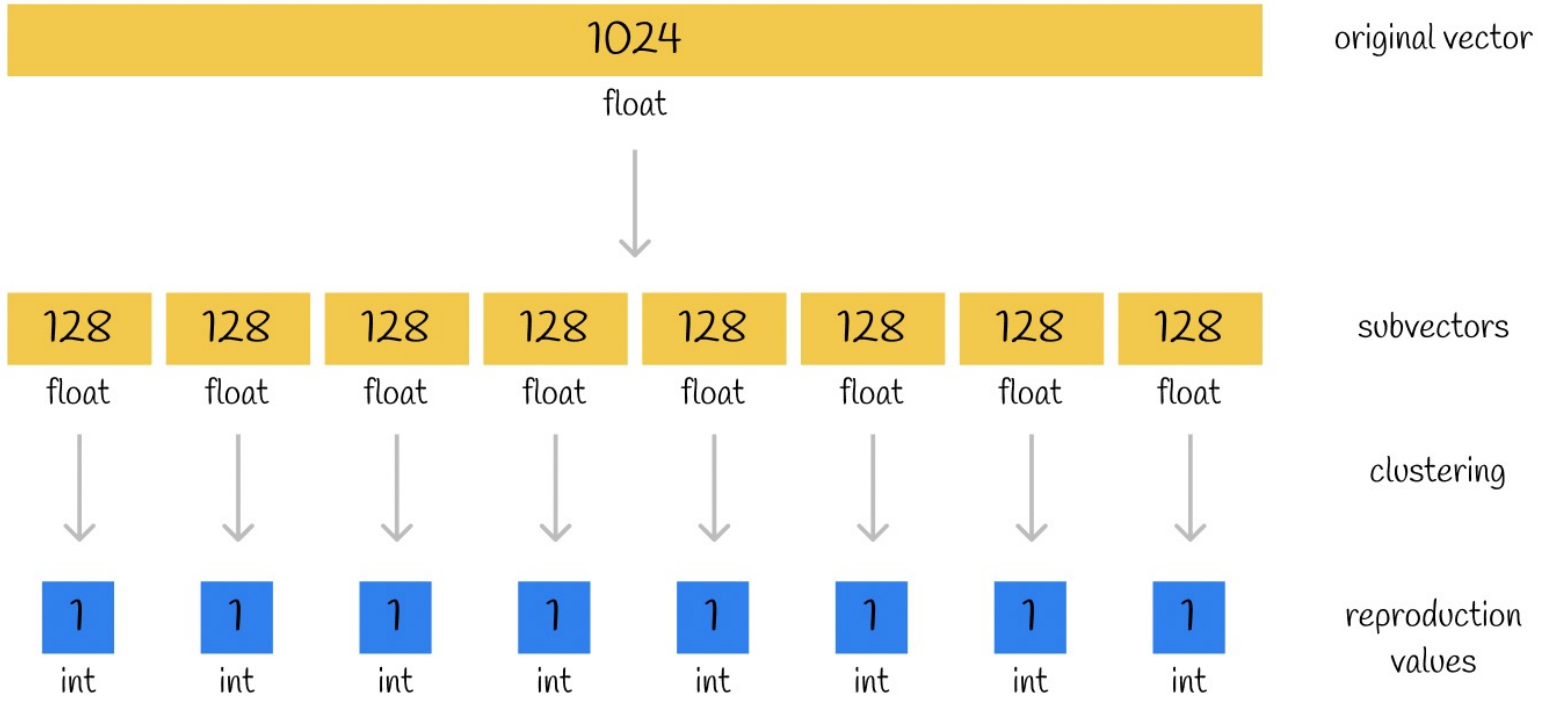
- Map the original dataset by a vector quantizer with k centroids using k-means
- Each code is an integer ranging from 1 to k
- Codebook: a map from code to the centroid



- Problem: need a large number of clusters to distinguish vectors

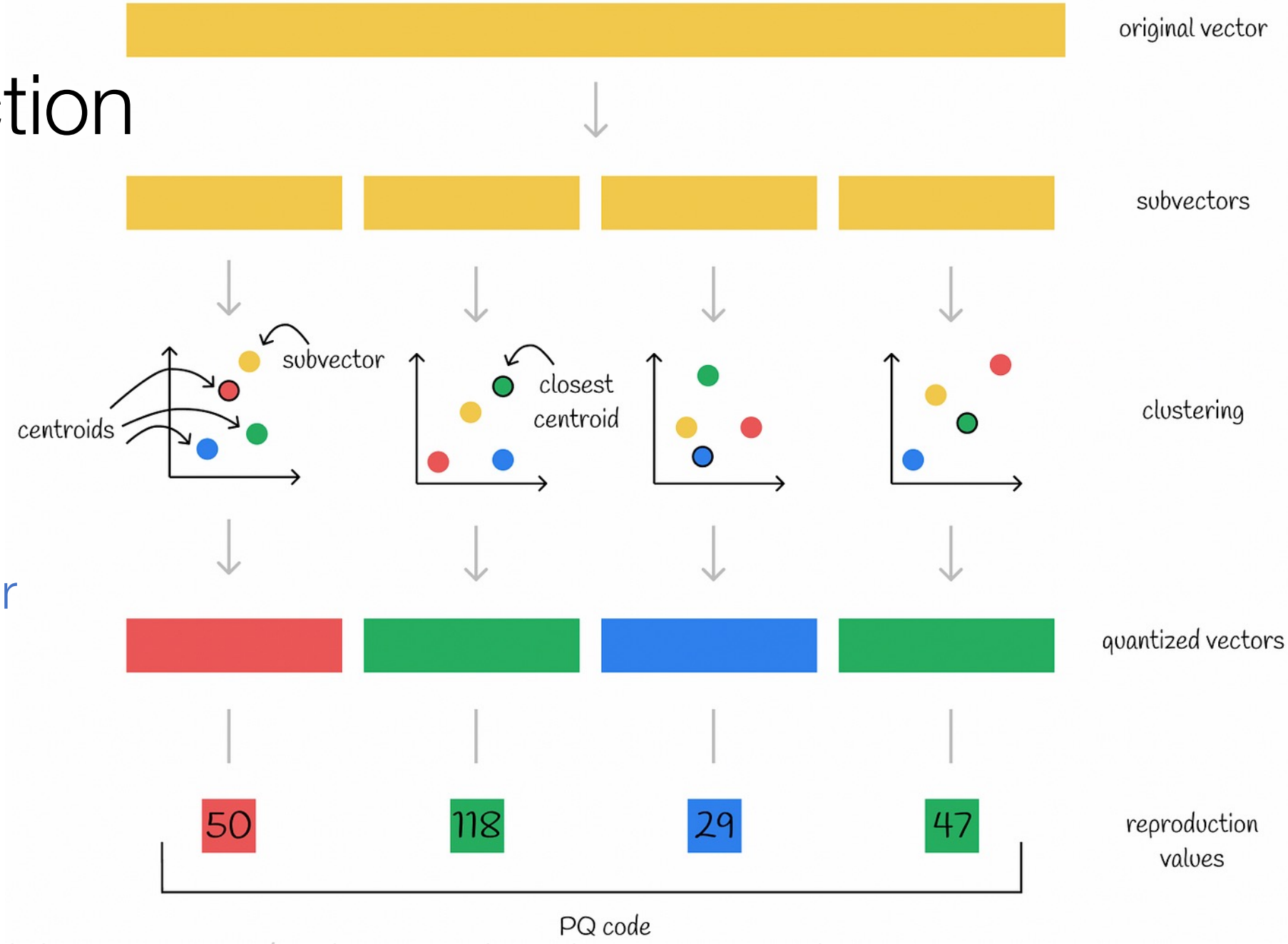
Product Quantizer

- 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

Construction

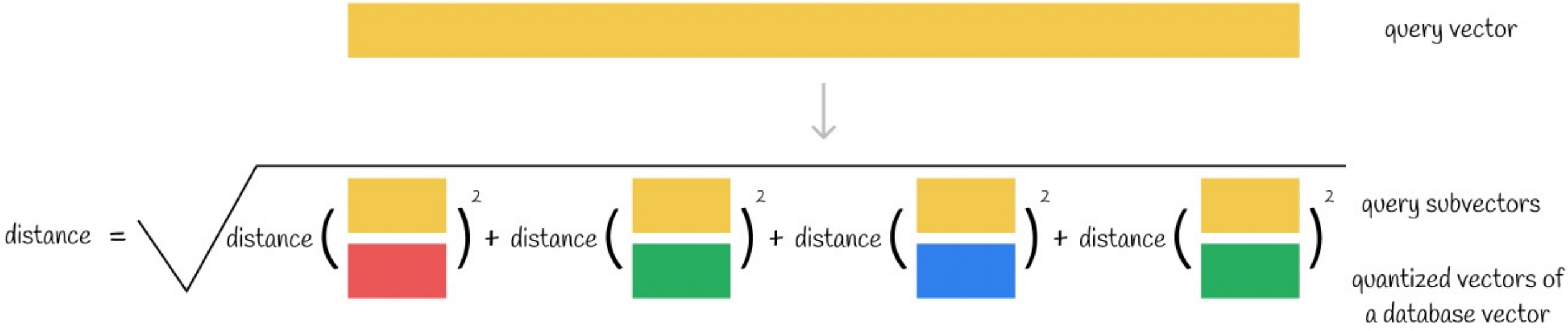


Each subvector space has its own set of clusters

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

Query

- Approximation: sum up the distances between each subvector and its closest centroid.



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

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