Similarity Search - PQ

Lin Guosheng
School of Computer Science and Engineering
Nanyang Technological University

Outline

- Nearest Neighbour Search (NNS) Problems

- Local Sensitive Hashing (LSH) – Random Projection

- **Product Quantization (PQ)**

- Inverted File Index **(non-examinable!!)**

Product Quantization (PQ)

- Product Quantization (PQ)
  - A clustering based method
  - Compress the data
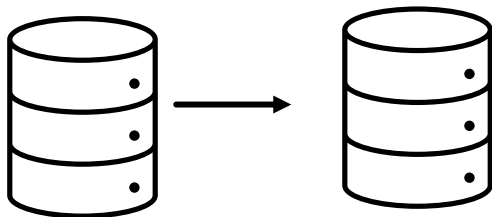 - Speed up searching by using precomputed distances

Most materials are from
https://www.pinecone.io/learn/product-quantization/

Paper: Product quantization for nearest neighbor search, 2011

Overview of PQ for similarity search

Step 2: process query requests
Use efficient approximate distance
(symmetric or asymmetric)
calculation to get top-k retrieved results

Step1: convert the database
into compressed vectors (offline step)

When using symmetric distance:
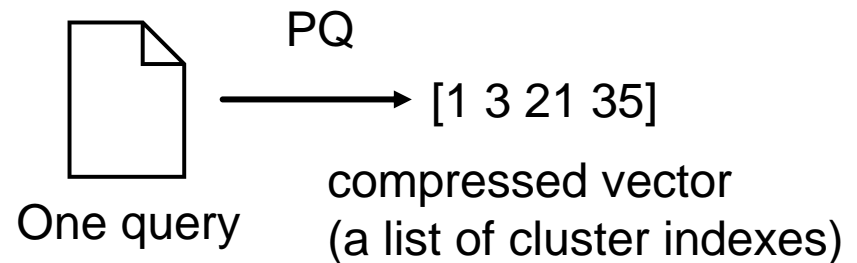For each query, we need to
convert the query to a compressed vector

Compressed vectors

When using symmetric distance:
Precompute the
centroid distances tables
for each subspace

One query

PQ

[1 3 21 35]

compressed vector
(a list of cluster indexes)

The process of PQ for compression

Pre-processing:

1. Define the number of subspaces (subvectors).

2. For each subspace, generate k cluster centroids using the samples in the database.

Generating compressed vectors using PQ

Input: the input feature vector (high-dimensional)

Steps:

1. Equally split the input into M sub-vectors.
Each sub-vector corresponds to one subspace.

2. Assigning each of these sub-vectors to its nearest cluster centroid (also called reproduction/reconstruction values)

3. Use the assigned cluster IDs (indexes) of all sub-vectors to construct the compressed vector (or called quantized vector).

Example

We define M=4 subspaces.
-> We generate 4 sub-vectors for each input.
(equally split the input vector into M sub-vectors)
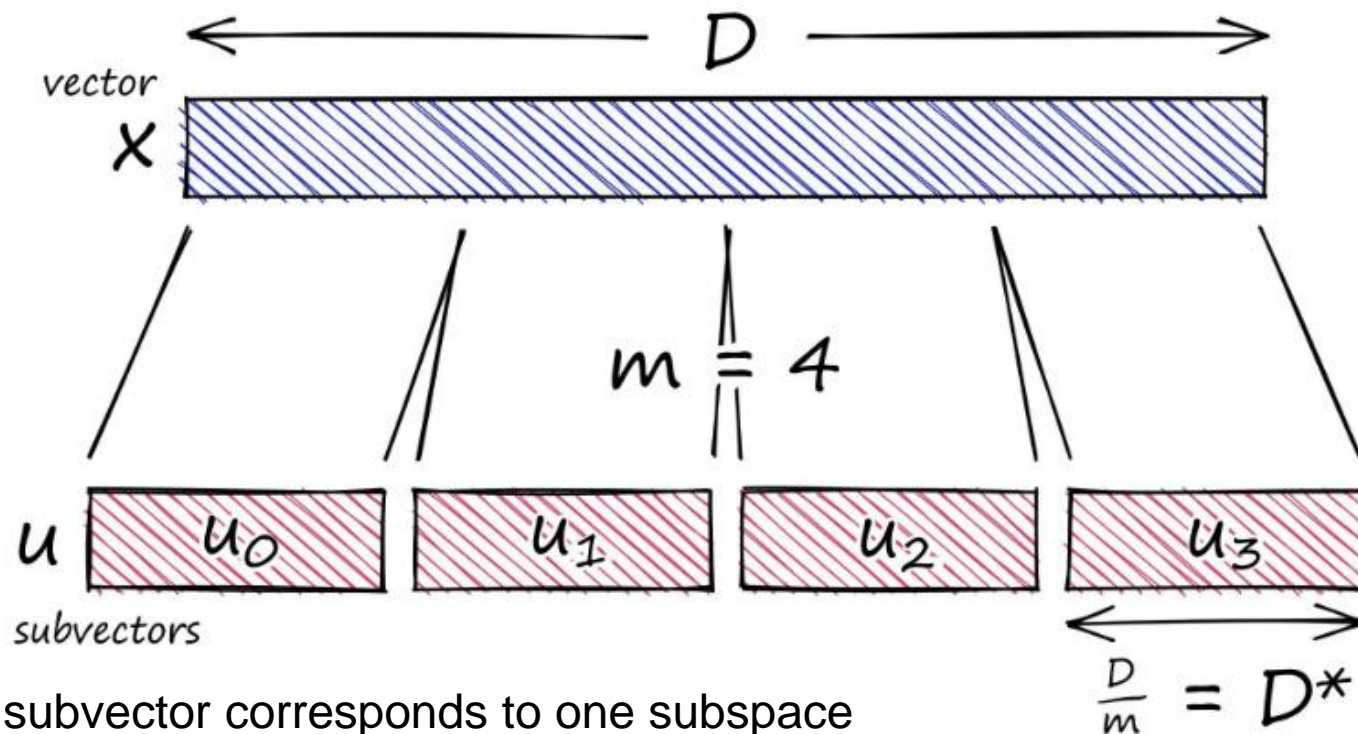
One subvector corresponds to one subspace

E.g., if the input vector dimension D=100,
the length of each sub-vector is 100/4 = 25

Sub-vector generation example

Compressed vector

Define 4 subspaces

-> 4 sub-vectors

Input vector

Equally split

Find the nearest centroid in subspace 1

Find the nearest centroid in subspace 2

Find the nearest centroid in subspace 3

Find the nearest centroid in subspace 4

Assigned cluster ID (S1)

Assigned cluster ID (S2)

Assigned cluster ID (S3)

Assigned cluster ID (S4)

Compressed vector:
[S1, S2, S3, S4]

Example

- Product Quantization example

    Define 2 Subspaces
    Each subspace has 4 centroids.

    Input vectors:
    A: [1.82, 5.08, 2.21, 4.21]
    B: [4.96, 4.46, 4.1, 1.3]

Example

Subspace 1, We have 4 centroids: C1 to C4.

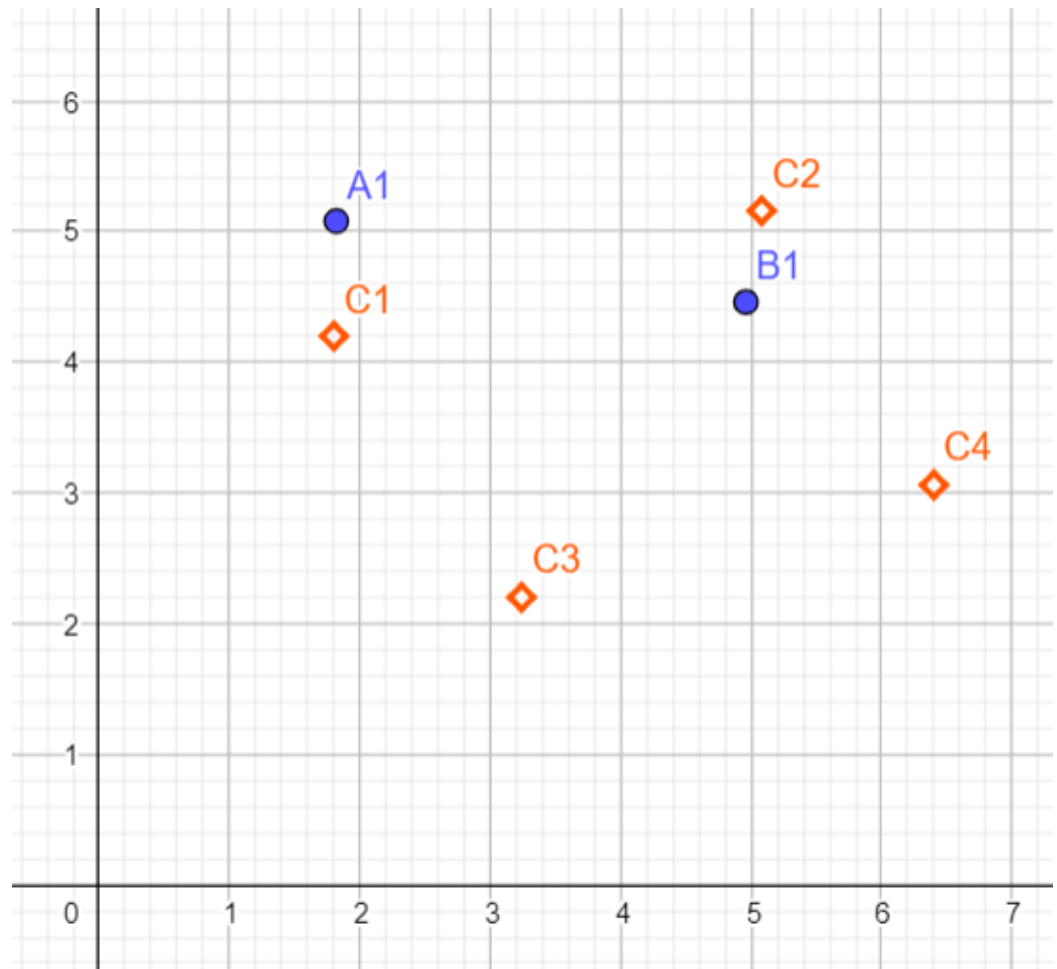A1, B1: the sub-vectors for datapoint A and B, respectively

$A1 = (1.82,\ 5.08)$

$B1 = (4.96,\ 4.46)$

$C1 = (1.8,\ 4.2)$

$C2 = (5.08,\ 5.16)$

$C3 = (3.24,\ 2.2)$

$C4 = (6.4,\ 3.06)$

Example

Subspace 2, We have 4 centroids: C1 to C4.

A2, B2: the sub-vectors for datapoint A and B, respectively

$A2 = (2.01, 4.21)$
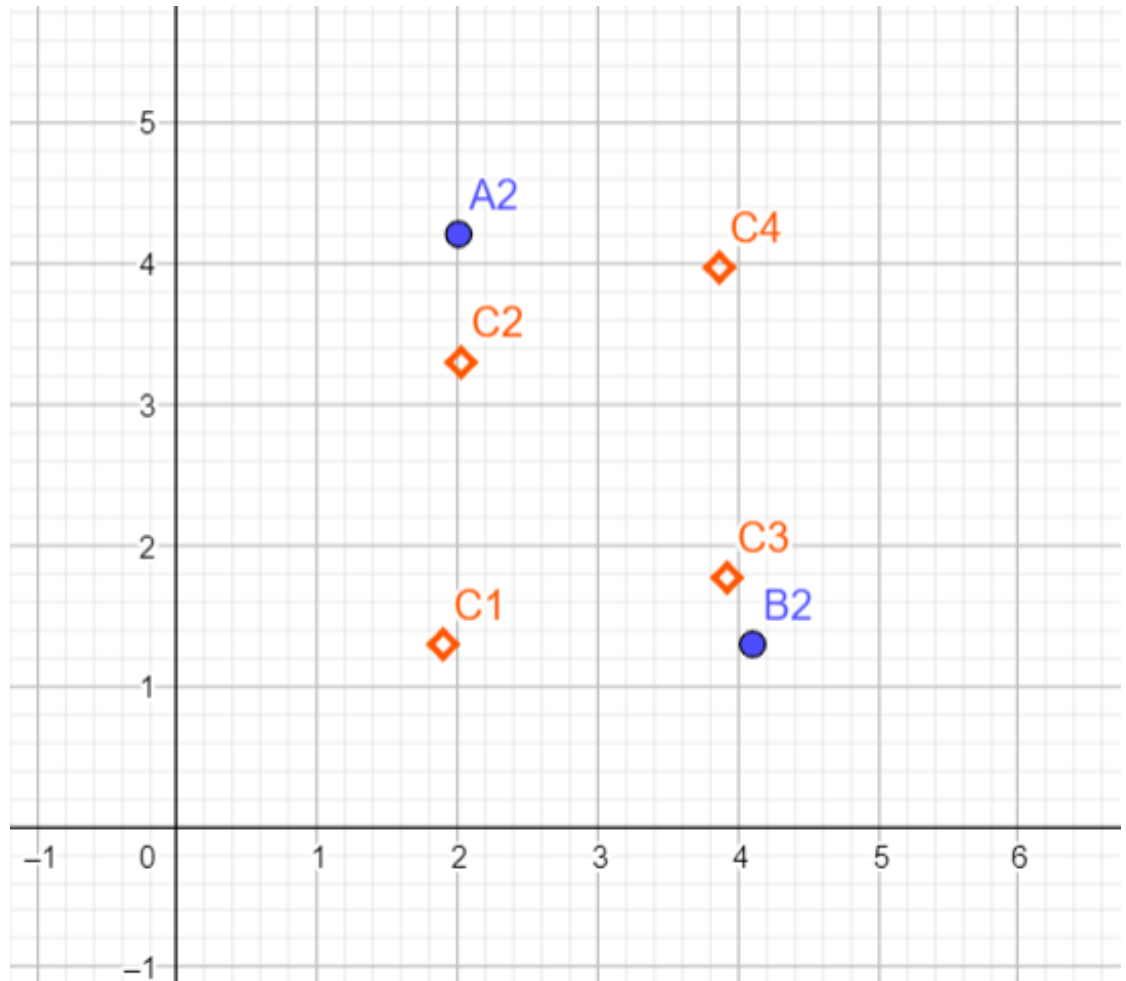
$B2 = (4.1, 1.3)$

$C1 = (1.9, 1.3)$

$C2 = (2.02, 3.3)$

$C3 = (3.92, 1.77)$

$C4 = (3.87, 3.98)$

Example

- Q: generate the compressed vector for datapoint A and B:

Answer:
In subspace 1, A is assigned to the centroid C1
(1st dimension of the compressed vector)

In subspace 2, A is assigned to the centroid C2
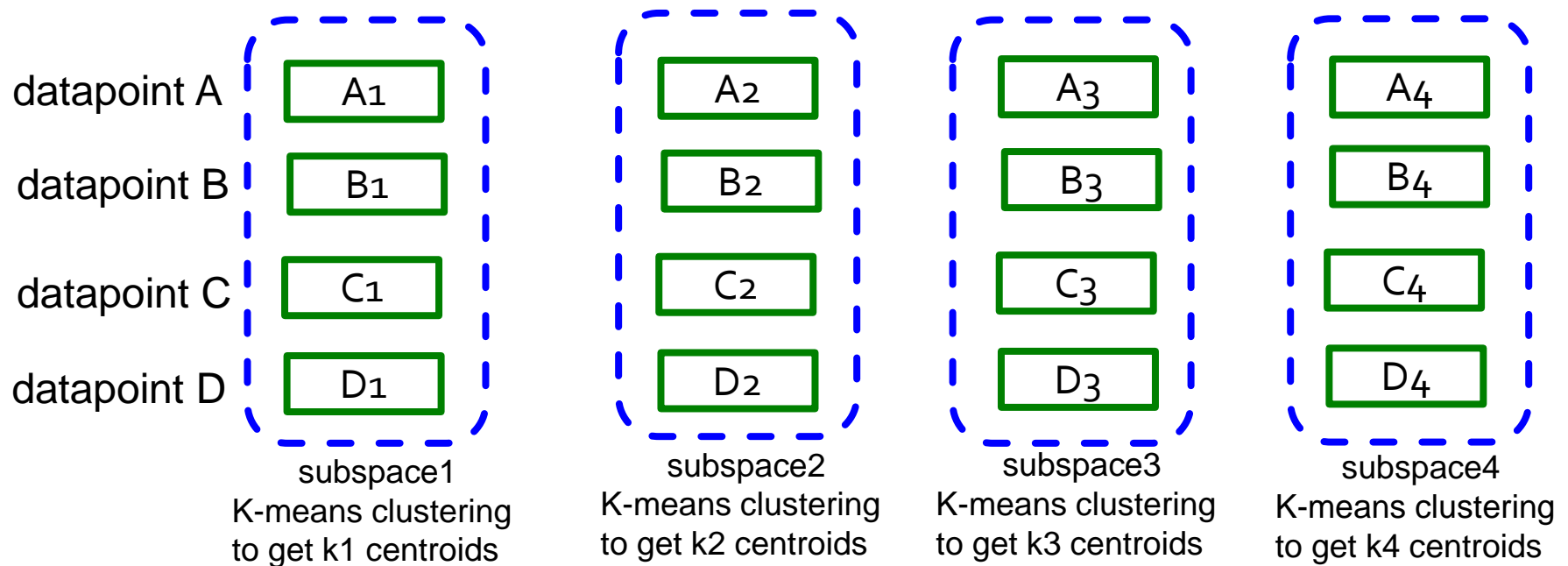(2nd dimension of the compressed vector)

-> A: [1, 2]

In subspace 1, B is assigned to the centroid C2
In subspace 2, B is assigned to the centroid C3
- -> B: [2, 3]

- How to generate cluster centroids?
  - Generate sub-vectors for all data examples in the database
    - Define M subspaces -> Generate M sub-vectors for one input.
  - Perform K-means in each subspace to generate k centroids.
   - For example, we can generate k=256 centroids for each sub space

Use datapoints in the database to generate centroids for each subspace

	subspace1	subspace2	subspace3	subspace4
datapoint A	A_1	A_2	A_3	A_4
datapoint B	B_1	B_2	B_3	B_4
datapoint C	C_1	C_2	C_3	C_4
datapoint D	D_1	D_2	D_3	D_4

subspace1
K-means clustering
to get k1 centroids

subspace2
K-means clustering
to get k2 centroids

subspace3
K-means clustering
to get k3 centroids

subspace4
K-means clustering
to get k4 centroids

Example: Subspace clustering

Database: 50K samples,
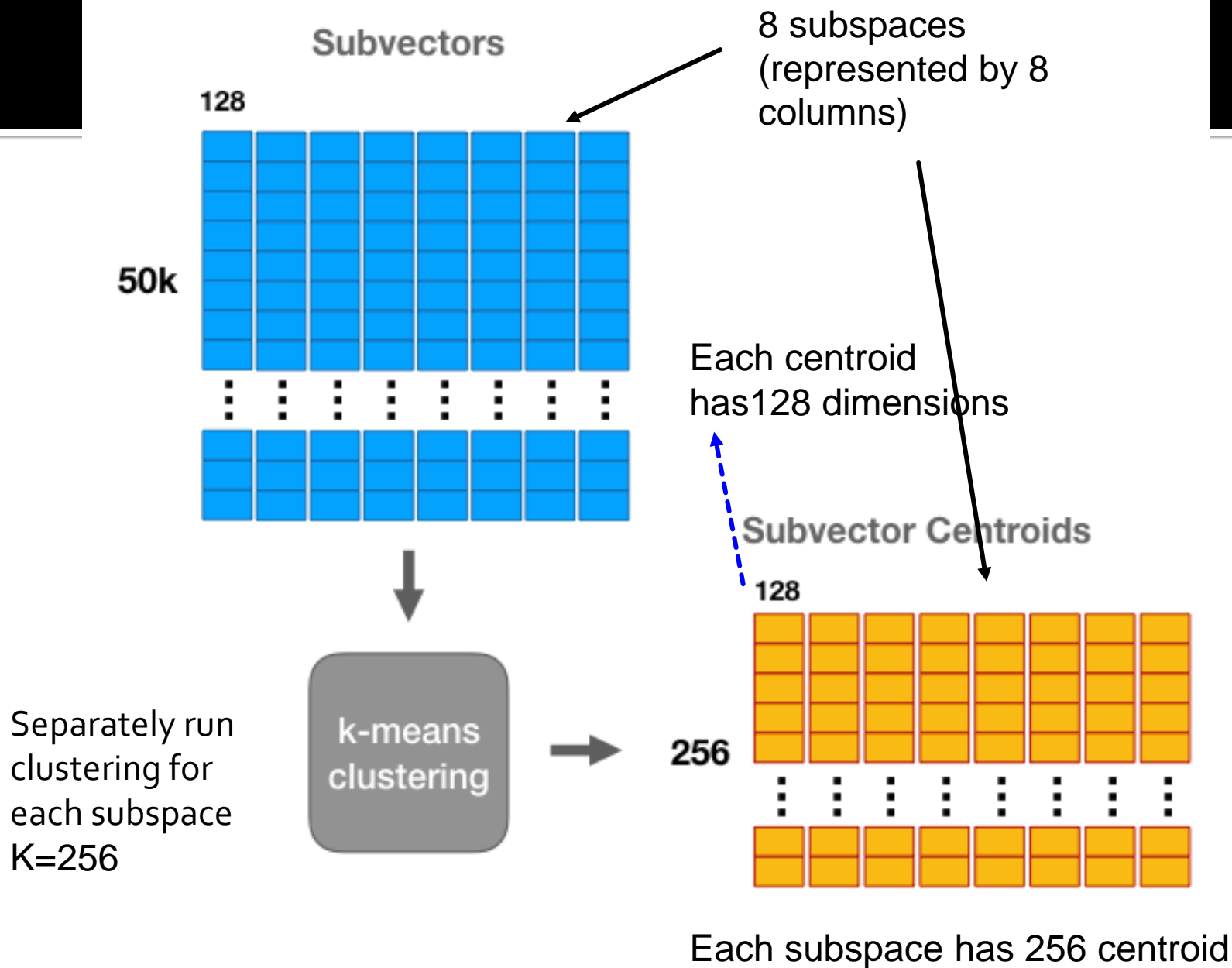each has 1024 dimensions:

8 sub-vectors for each sample:
(each sub-vector's dimension: 128)
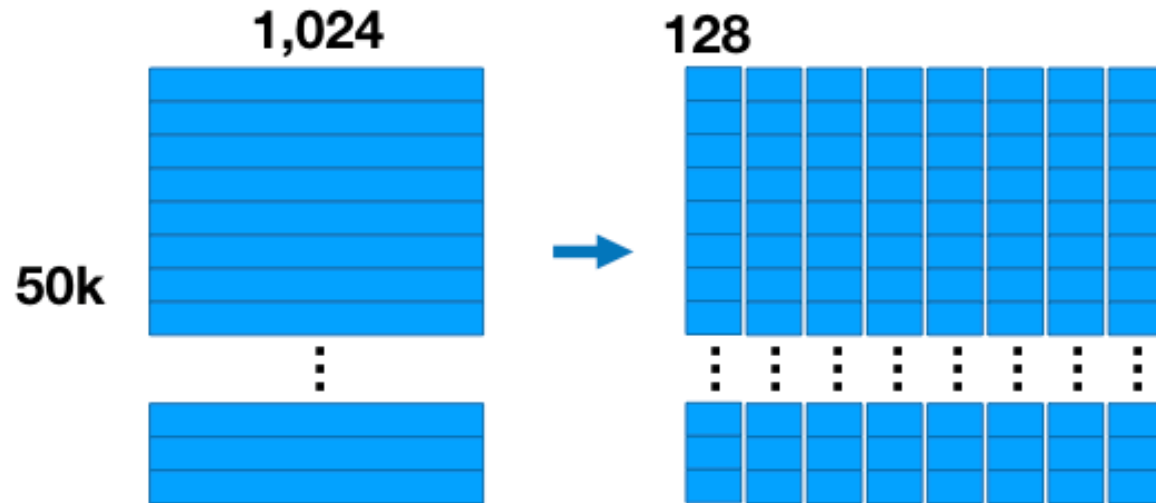
Each row indicates one data sample.

Define 8 subspaces. Each vector is spitted into 8 sub-vectors, each of
length 128 (8 sub vectors x 128 dimensions = 1,024 dimensions).

https://mccormickml.com/2017/10/13/product-quantizer-
tutorial-part-1/#nearest-neighbor-search

Subvectors

128

50k

8 subspaces (represented by 8 columns)

Each centroid has128 dimensions

Subvector Centroids

128

256

Separately run clustering for each subspace K=256

k-means clustering

Each subspace has 256 centroid

14

Compression analysis

1 float value requires 4 bytes (4*8=32 bits);
1 integer value requires 1 byte (here we use 8-bit integer, value range: 0-255)

1024 x (4 bytes)=4096 bytes

8 sub-vectors

The final compression is 512 times!

8 x (1 bytes)= 8 bytes
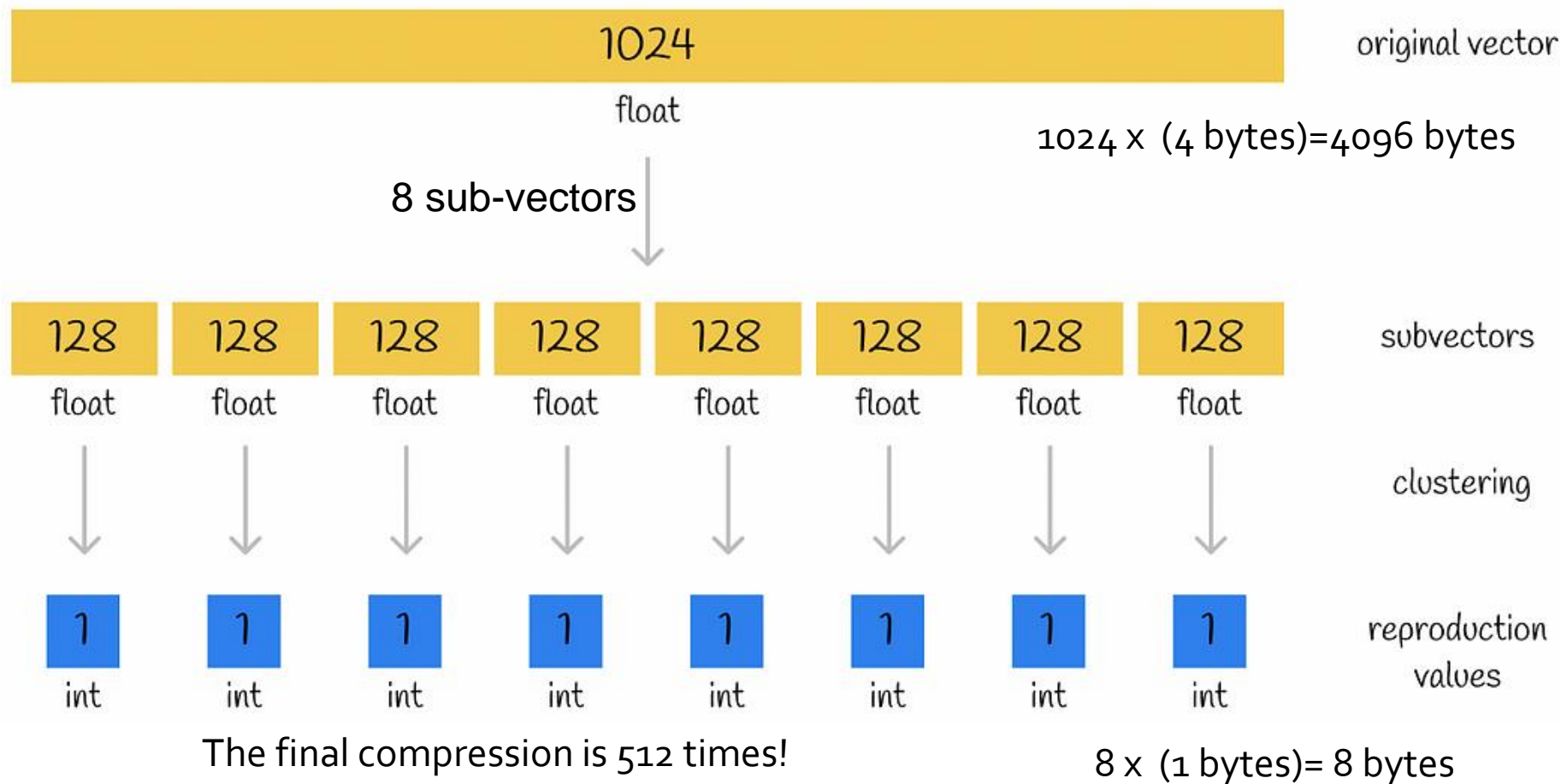
One centroid ID is an integer value. In this example, we only have 256 centroids, so we can use one 8-bit integer ($2^{8=256}$) to store one cluster ID.

Credit: Vyacheslav Efimov, https://towardsdatascience.com/similarity-search-product-quantization-b2a1a6397701

Product Quantization

- Codebook based compression
  - the codebook is the centroids
  - Separate codebook (centroids) for each subspace

- Compress high-dimensional vector to a tiny vector of IDs that only requires very little memory/storage space.

Product Quantization

- ## Product Quantization (PQ)

  - ### A data dependent method
    - Need to use the data in the database to determine the parameters (cluster centres) of the algorithm.

  - ### LSH is a data independent method
    - The hash functions are randomly generated, not using the data in the database.

PQ for similarity search

- ## PQ for Similarity Search
  - ### Use liner search (exhaustive search)
    - Use approximate distance to speed up distance calculation

  - ### Calculating approximate distance
    - Symmetric distance
      - Use pre-computed distance look-up table to speed up the calculation.

    - Asymmetric distance

Symmetric distance

- Approximate distance: Symmetric distance

$$\hat{d}(x, y) = d(q(x), q(y)) = \sqrt{\sum_j d(q_j(x), q_j(y))^2},$$
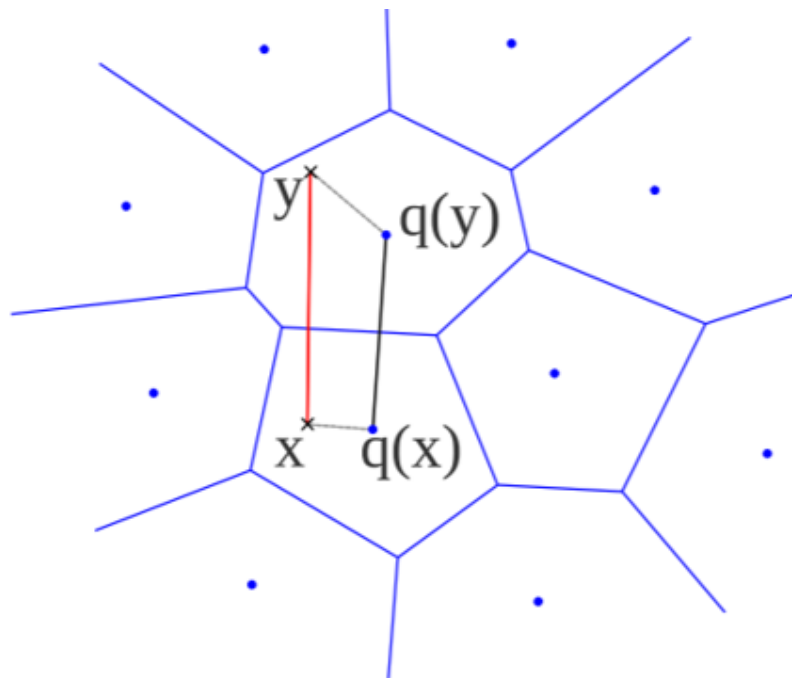
symmetric case

x, y are two input vectors

q(x) and q(y) are the
assigned cluster centroids

The distance of (x, y) is approximated
by the distance of cluster centroids

Recap

Overview of PQ for similarity search

Step1: convert the database
into compressed vectors (offline step)
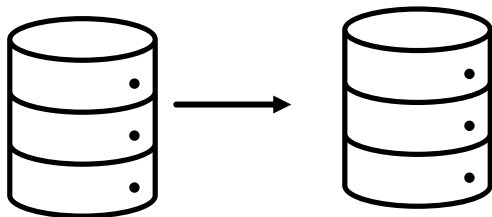
Step 2: process query requests
Use efficient approximate distance
(symmetric or asymmetric)
calculation to get top-k retrieved results

When using symmetric distance:
For each query, we need to
convert the query to a compressed vector

Compressed vectors

When using symmetric distance:
Precompute the
centroid distances tables
for each subspace

PQ

[1 3 21 35]

One query

compressed vector
(a list of cluster indexes)
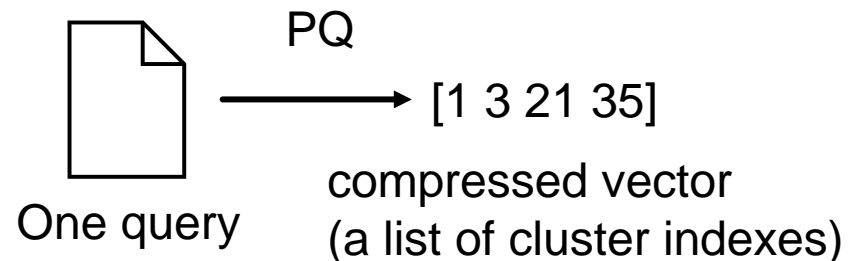
Symmetric distance example

Subspace 1, We have 4 centroids: C1 to C4.

A1, B1: the sub-vectors for datapoint A and B, respectively

$A1 = (1.82, 5.08)$

$B1 = (4.96, 4.46)$

$C1 = (1.8, 4.2)$

$C2 = (5.08, 5.16)$

$C3 = (3.24, 2.2)$

$C4 = (6.4, 3.06)$
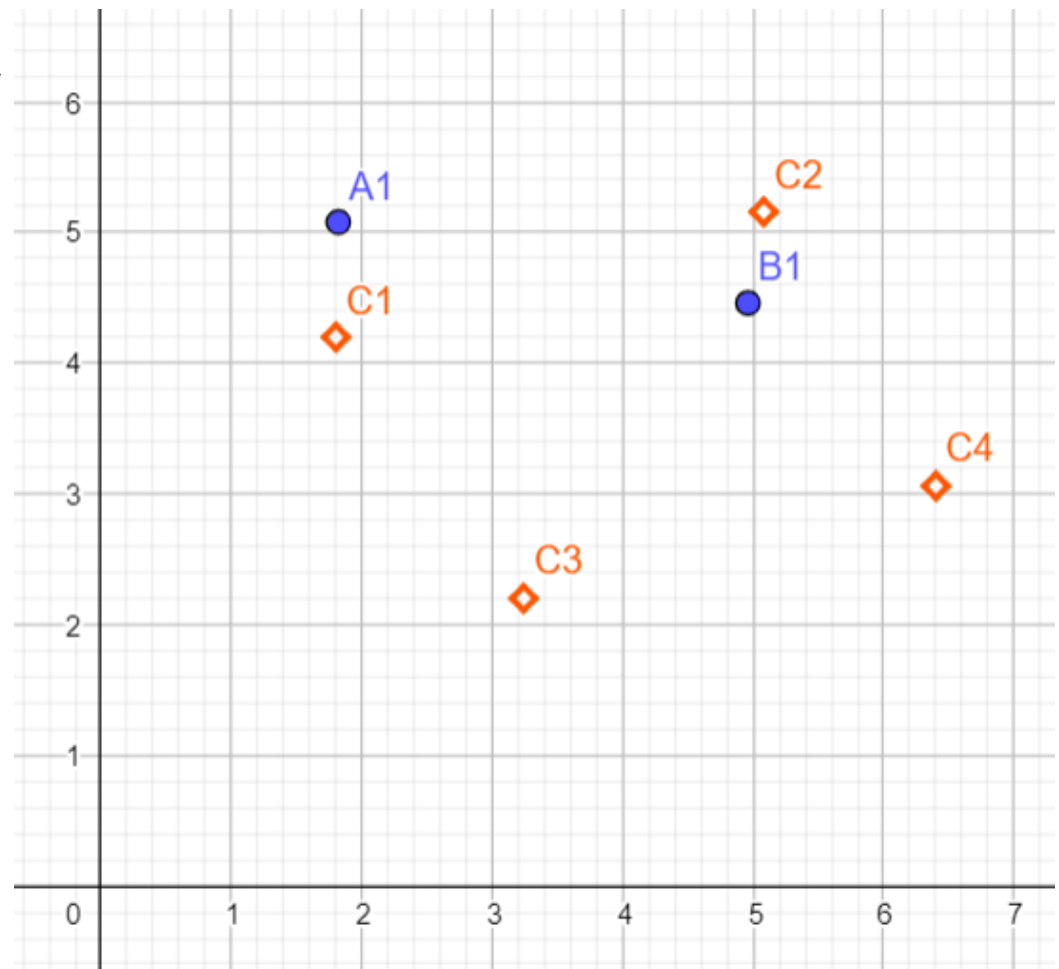
Symmetric distance example

Subspace 2, We have 4 centroids: C1 to C4.

A2, B2: the sub-vectors for datapoint A and B, respectively
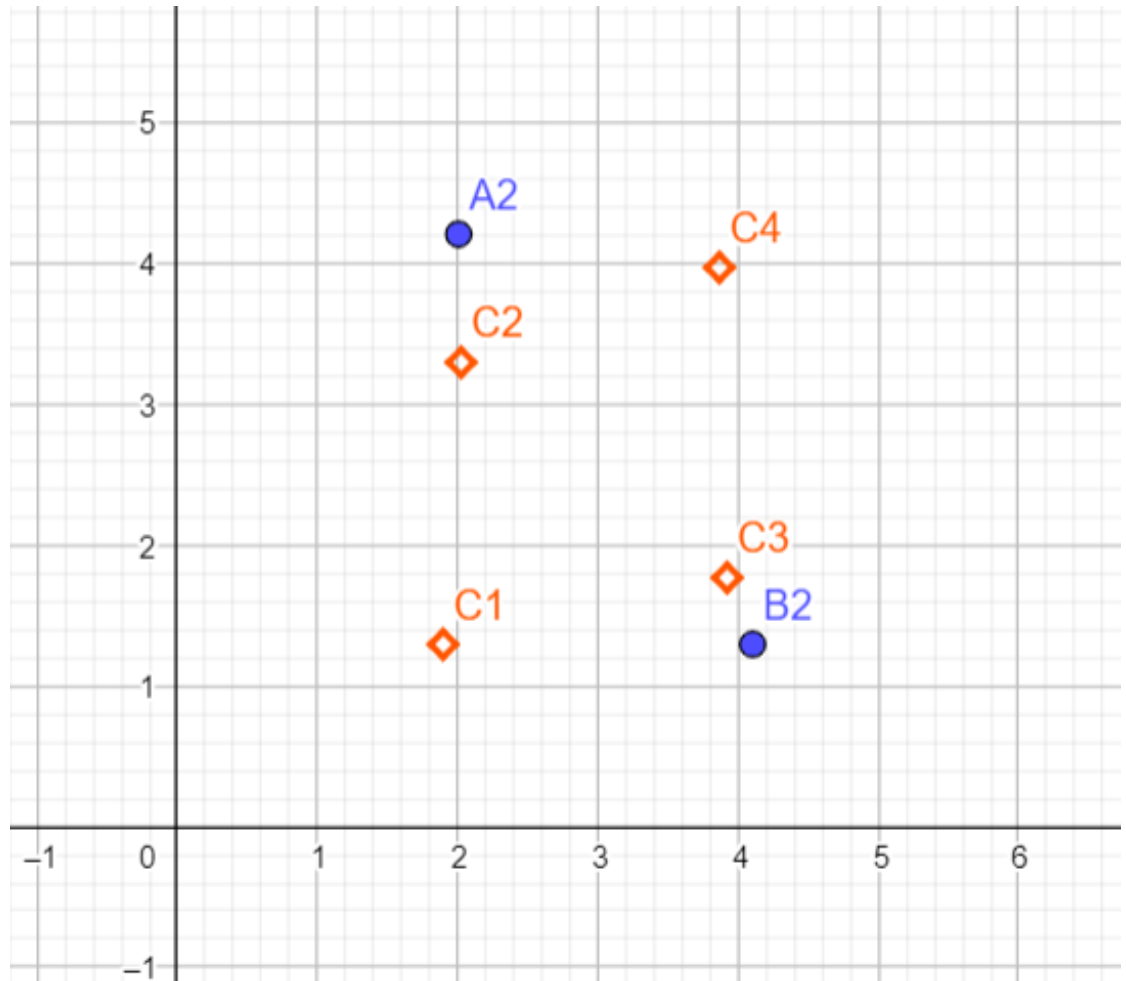
$A2 = (2.01, 4.21)$

$B2 = (4.1, 1.3)$

$C1 = (1.9, 1.3)$

$C2 = (2.02, 3.3)$

$C3 = (3.92, 1.77)$

$C4 = (3.87, 3.98)$

Symmetric distance example

Define 2 subspaces. The centroids and datapoints are given in the above slides.

Question:
1. Construct centroid distance look up tables and

2. compute the approximate Squared L2 distance (symmetric case) between A and B.

Symmetric distance example

- Solution to Q1:

Subspace 1

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	11.68	6.07	22.46
Centroid 2	11.68	0	12.15	6.15
Centroid 3	6.07	12.15	0	10.73
Centroid 4	22.46	6.15	10.73	0

Table 1: squared L2 distance table for Subspace 1 (D1)

Distance calculation example:
C1=[1.8, 4.2], C2=[5.08, 5.16],
D1(C1, C2) = (1.8-5.08)^2 + (4.2-5.16)^2 = 10.76 + 0.92 =11.68

Subspace 1, We have 4 centroids: C1 to C4.

A1, B1: the sub-vectors for datapoint A and B, respectively

$A1 = (1.82,\ 5.08)$

$B1 = (4.96,\ 4.46)$

$C1 = (1.8,\ 4.2)$

$C2 = (5.08,\ 5.16)$

$C3 = (3.24,\ 2.2)$

$C4 = (6.4,\ 3.06)$
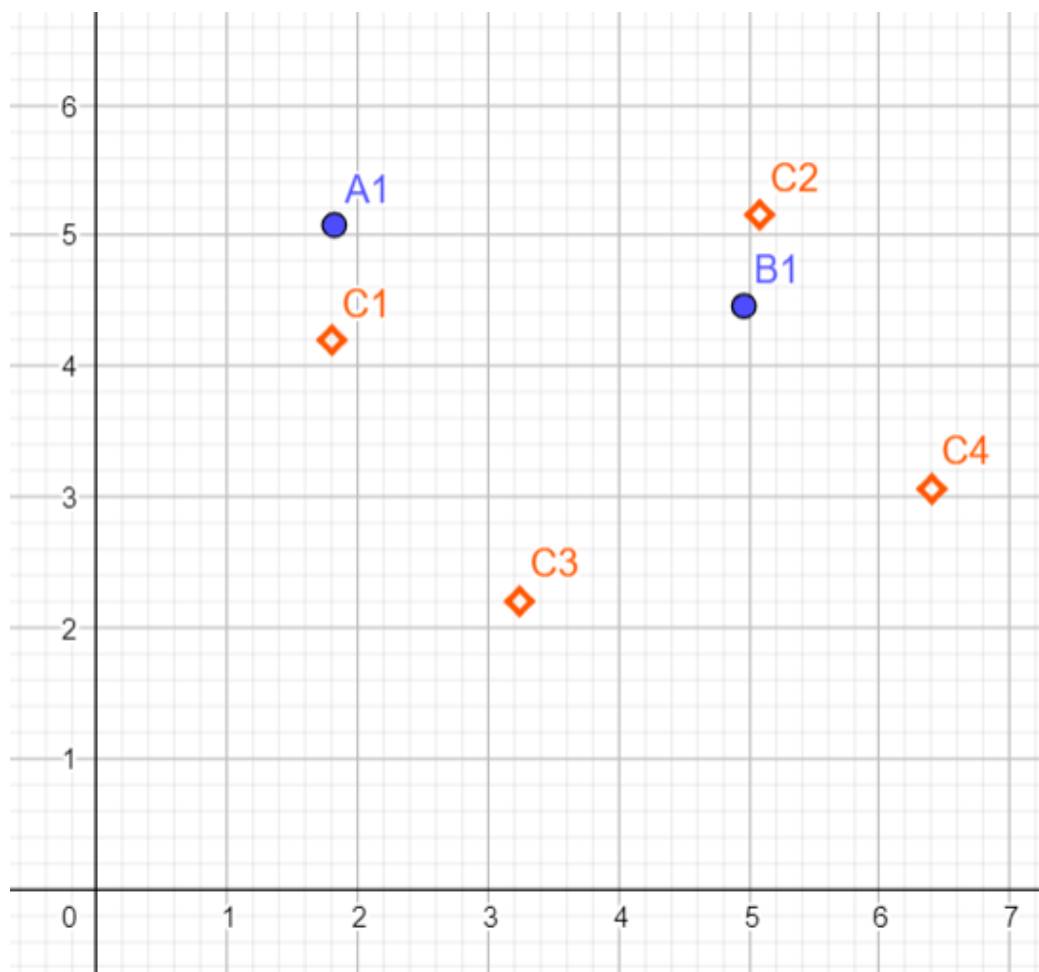
Subspace 2

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	4.01	4.30	11.06
Centroid 2	4.01	0	**5.95**	3.88
Centroid 3	4.3	5.95	0	4.89
Centroid 4	11.06	3.88	4.89	0

Table 2: squared L2 distance table for Subspace 2 (D2)

Subspace 2, We have 4 centroids: C1 to C4.

A2, B2: the sub-vectors for datapoint A and B, respectively

$A2 = (2.01, 4.21)$

$B2 = (4.1, 1.3)$

$C1 = (1.9, 1.3)$

$C2 = (2.02, 3.3)$

$C3 = (3.92, 1.77)$

$C4 = (3.87, 3.98)$
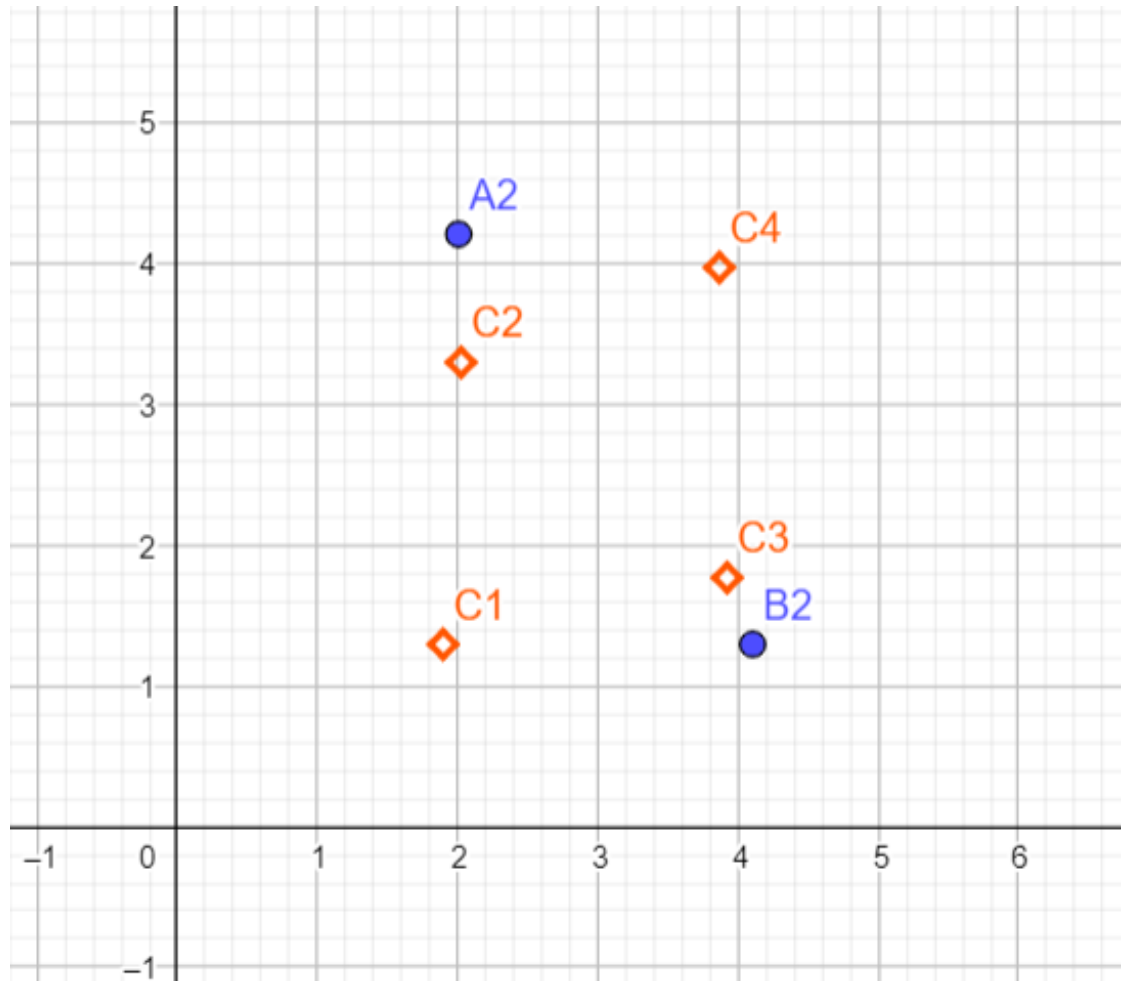
Question:

2. compute the approximate Squared L2 distance (symmetric case) between A and B.

- ## Solution to Q2:

  Compressed vectors: A: [1, 2];   B: [2, 3]

  Look up the distance values in table D1 and D2 using the cluster IDs:

  D (A1, B1) = D1(Centroid 1, Centroid 2) = 11.68
  D (A2, B2) = D2(Centroid 2, Centroid 3) = 5.95

  D(A, B) ~= D (A1, B1) + D (A2, B2) + = 11.68 + 5.95 = 17.63

Subspace 1

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	11.68	6.07	22.46
Centroid 2	11.68	0	12.15	6.15
Centroid 3	6.07	12.15	0	10.73
Centroid 4	22.46	6.15	10.73	0

Subspace 2

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	4.01	4.30	11.06
Centroid 2	4.01	0	5.95	3.88
Centroid 3	4.3	5.95	0	4.89
Centroid 4	11.06	3.88	4.89	0

Compressed vectors: A: [1, 2];   B: [2, 3]

D (A1, B1) = D1(Centroid 1, Centroid 2) = 11.68
D (A2, B2) = D2(Centroid 2, Centroid 3) = 5.95

D(A, B) ~= D (A1, B1) + D (A2, B2) + = 11.68 + 5.95 = 17.63

Symmetric distance

- Approximate distance calculation

  - we can calculate the distances much more efficiently using just table look-ups and some addition.

  - Using centroid pre-computed distances.
    Pre-compute and the distances between centroids for each subspace, stored the distances in a matrix or lookup table, use the centroid index pair to retrieve the partial distances for each subspace.

  - Sum-up the partial distances from all subspaces as the final distance output

Asymmetric distance

Asymmetric distance
 More accurate for distance calculation,
as we only quantize one input.
(compared to symmetric distance)

$$\tilde{d}(x, y) = d(x, q(y))$$

x: a query example,
y: an example in the database.

Use q(y) to replace y
for distance calculation
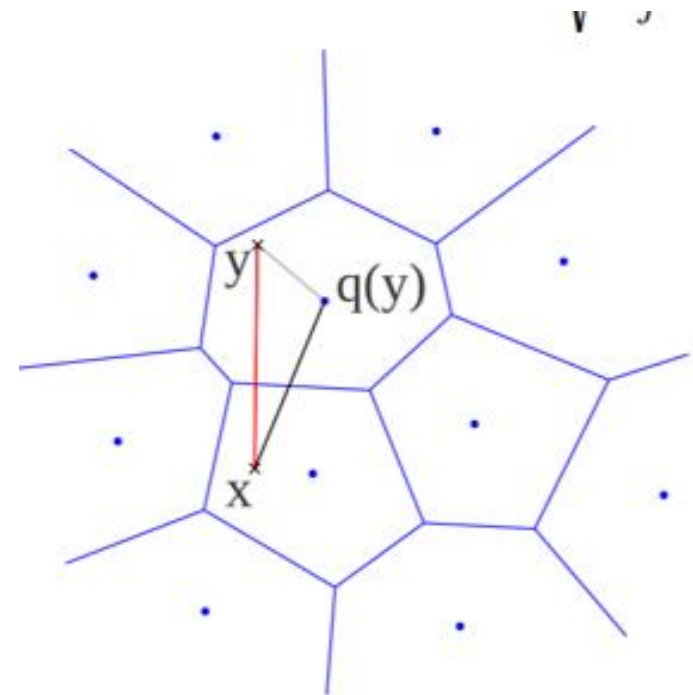
asymmetric case

Example for asymmetric distance

Traditional
linear search
For top-k retrieval

Database
All datapoints
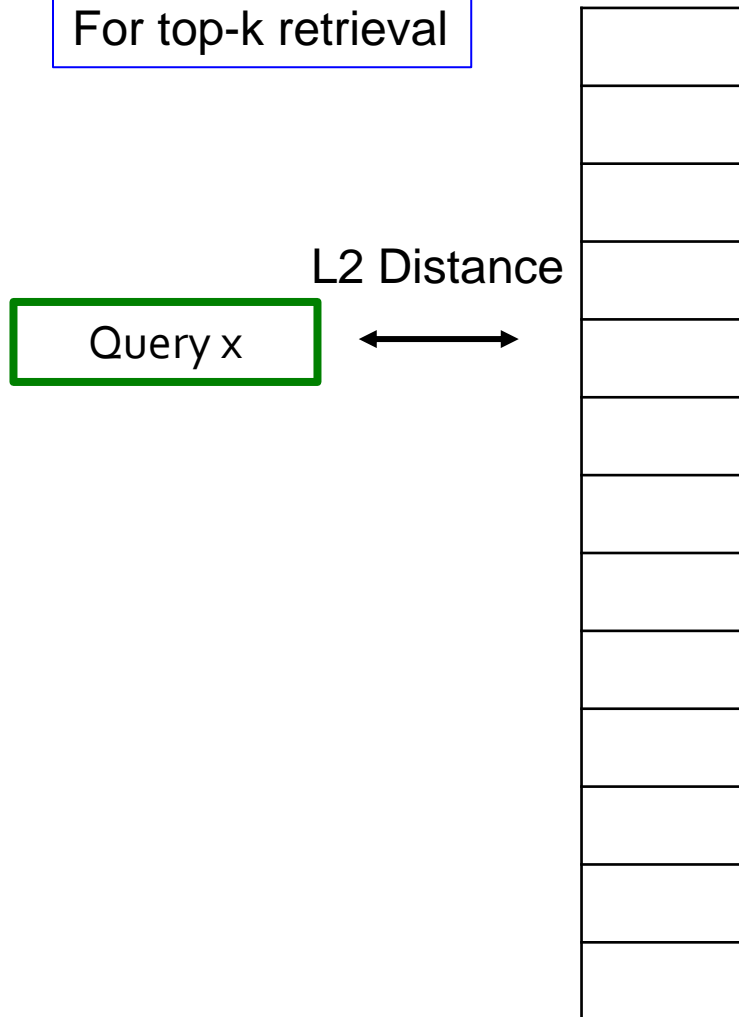
PQ + asymmetric distance
for top-k retrieval

Database
Centroids
(subspace1)

L2 Distance

Query x

L2 Distance

Query x

L2 Distance

Database
Centroids
(subspace2)
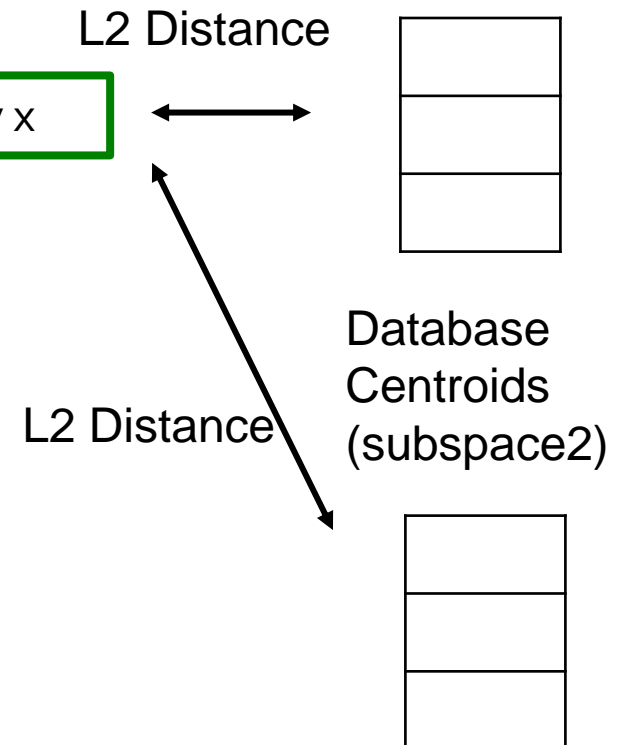
Asymmetric distance

1. Given a query in a top-k retrieval task, we don't need to compute the distance between the query and all the samples in the database.

   We only need to calculate the distance between the query and all the centroids in each subspace.

$$\tilde{d}(x, y) = d(x, q(y))$$

asymmetric case

2. The number of centroids are much less than the number of samples in the database, so we can speed up the retrieval process.

3. We don't need to quantize x, which reduces computation compared to the symmetric case.
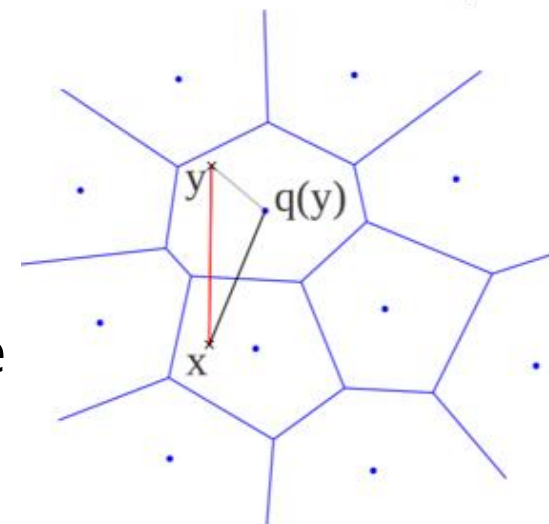
Asymmetric distance

4. For each subspace, we compute the squared L2 distances between the x and all centroids.
This bring extra computation compared to the symmetric case.

5. The above extra computation cost is similar with the quantization cost for x. (cluster assignment requires distance calculation between x and all centroids)

That means symmetric and asymmetric distance have the same computation complexity.
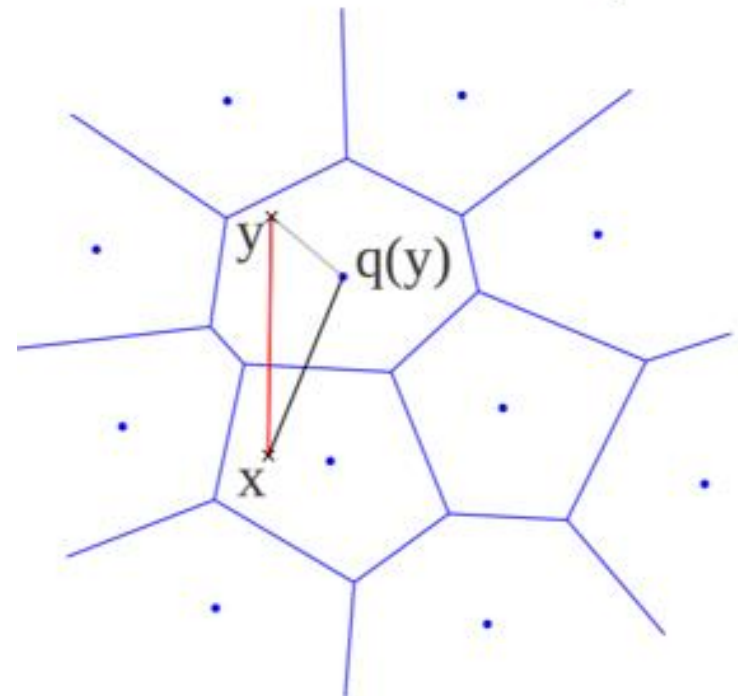
$$\tilde{d}(x, y) = d(x, q(y))$$
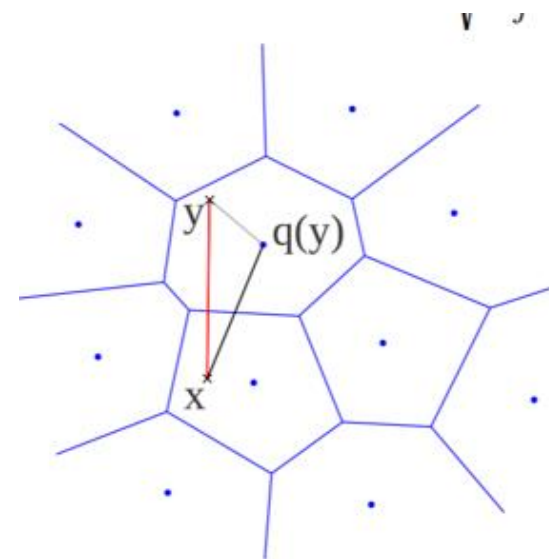
asymmetric case

Comparison

$$\hat{d}(x,y) = d\big(q(x), q(y)\big) = \sqrt{\sum_j d\big(q_j(x), q_j(y)\big)^2},$$
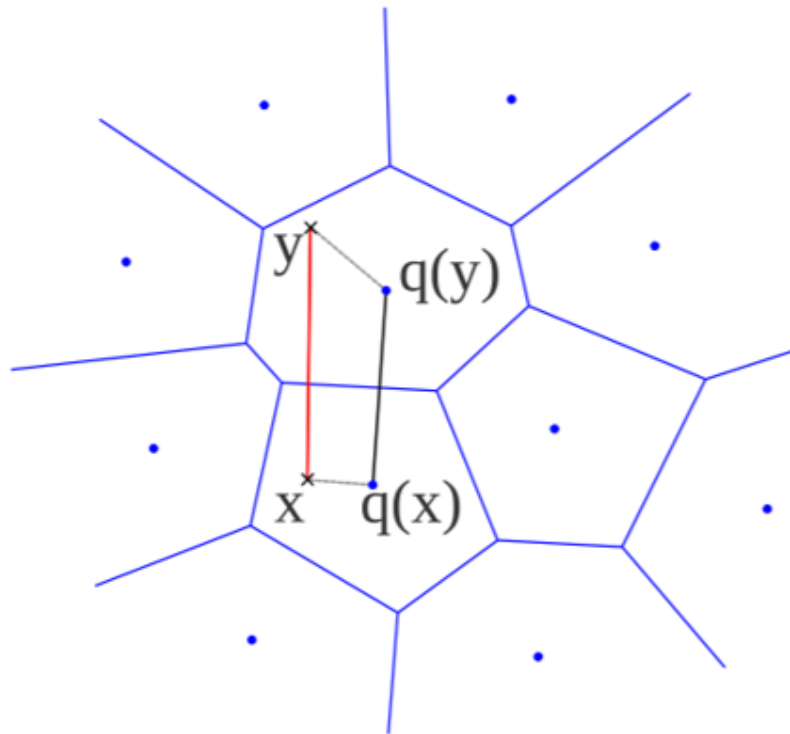
$$\tilde{d}(x,y) = d\big(x, q(y)\big)$$

symmetric case

asymmetric case

Comparison for top-k search

PQ+
Symmetric distance

Database
All datapoints

PQ + asymmetric distance
for top-k retrieval

Query x

Distance lookup

Compressed vector

1. Distance calculation using centroid distance lookup table

Cost:
 Need to compress the query to compressed vector (it requires distance calculation between the query and all centroids)
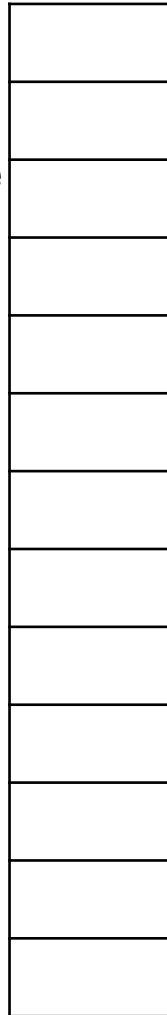
Database
Centroids
(subspace1)

L2 Distance

Query x

L2 Distance

Database
Centroids
(subspace2)

Cost:
require distance calculation between the query and all centroids
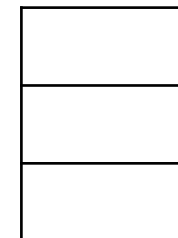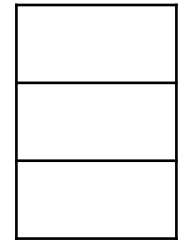
Outline

- Nearest Neighbour Search (NNS) Problems

- Local Sensitive Hashing (LSH) – Random Projection

- Product Quantization (PQ)

- **Inverted File Index (non-examinable!!)**

Inverted File Index

Non-examinable!

- Inverted File Index
  - Clustering based method
 - Data dependent
  - Reduce the search scope

Non-examinable!

- Inverted File Index (IVT)
  - An inverted index here means a mapping (a lookup table) from cluster centroids (words) to the cluster members.
- Steps:
  - 1. Clustering of the samples in the database, generate the centroids and cells
  - 2. Given a query vector, identify the cells for search scope
  - 3. Use linear search (exhaustive search) to retrieve results within the selected cells

xq: the query vector

search scope
nprobe = 1

xq
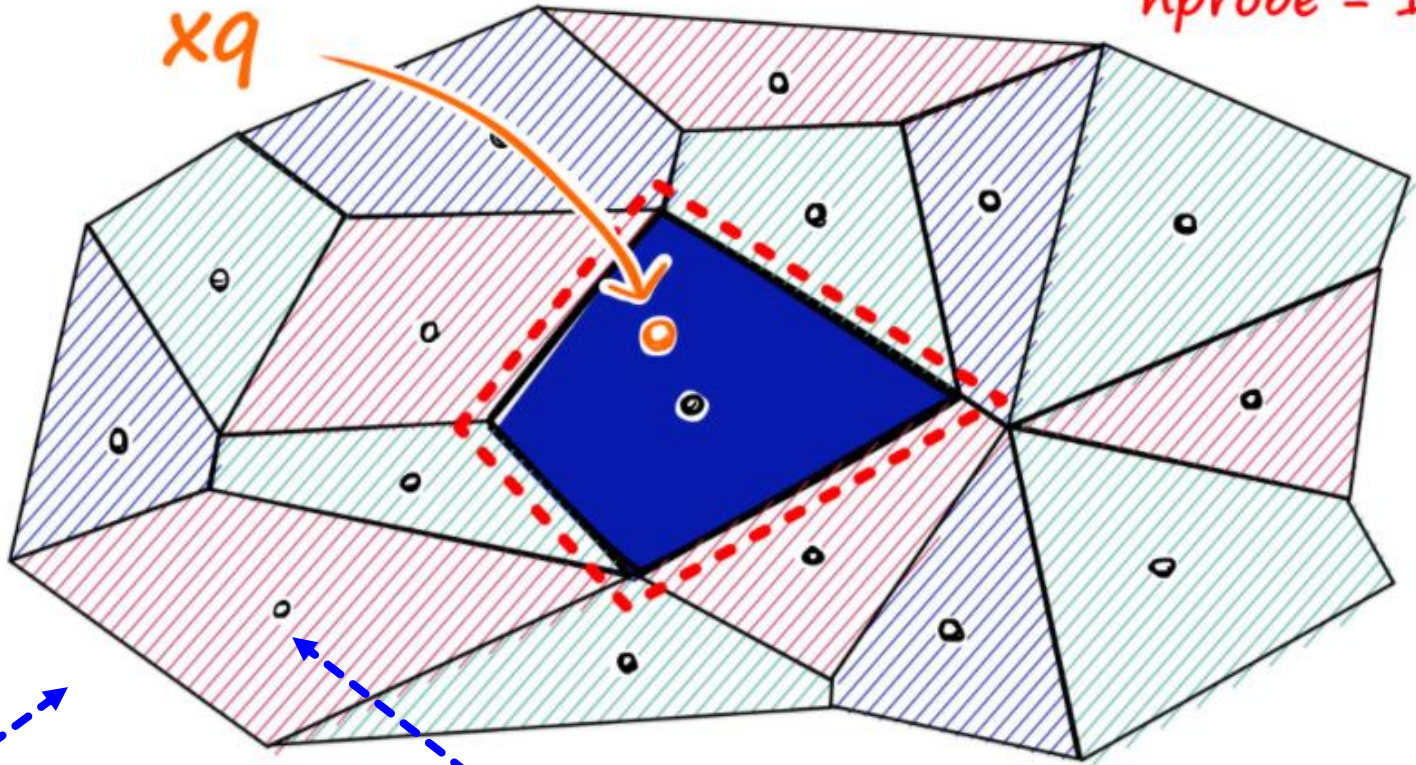
Perform k-means clustering to group the data points in the database (cluster centroids and assignments)

Cluster centroid in one cell.
One cell represents one cluster
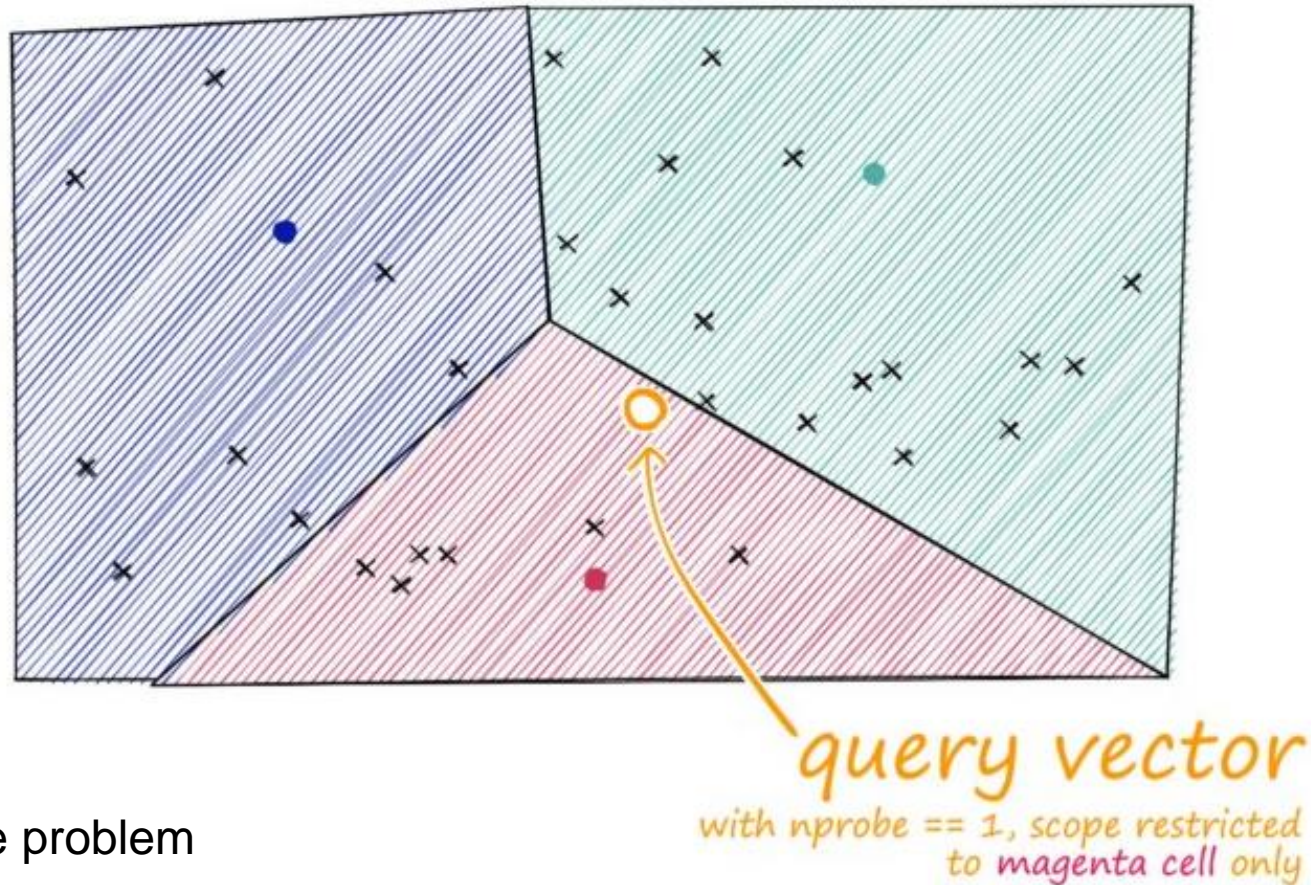
xq: the query vector

search scope
nprobe = 1

xq
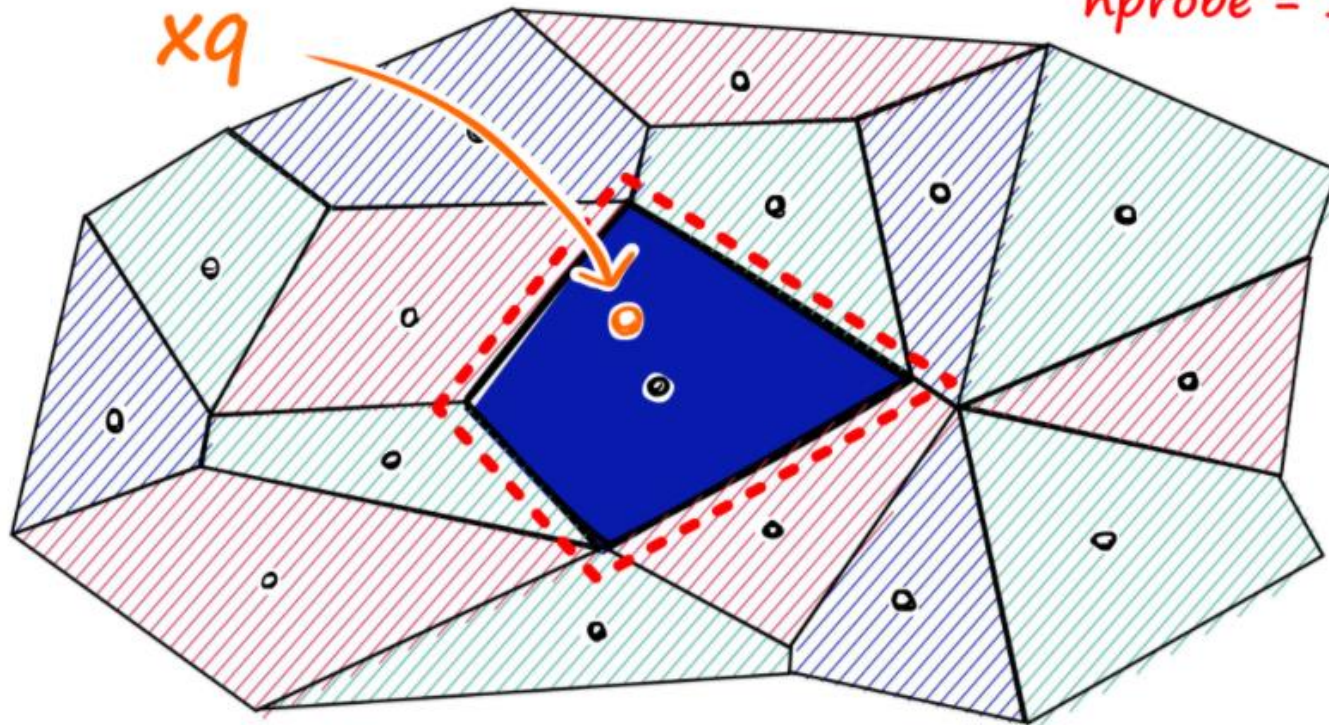
nprobe: select the top k nearest clusters (cells) as the search scope
nprobe=1: only select 1 cell for searching

There are 2 parameters in IVT:
1. nprobe: the number of cells to search
2. The number of cells (clusters) to create.

query vector
with nprobe == 1, scope restricted
to magenta cell only

Edge problem

Our query vector **xq** lands on the edge of the magenta cell. Despite being closer to datapoints in the teal cell, we will not compare these if **nprobe == 1** — as this means we would restrict search scope to the magenta cell only.
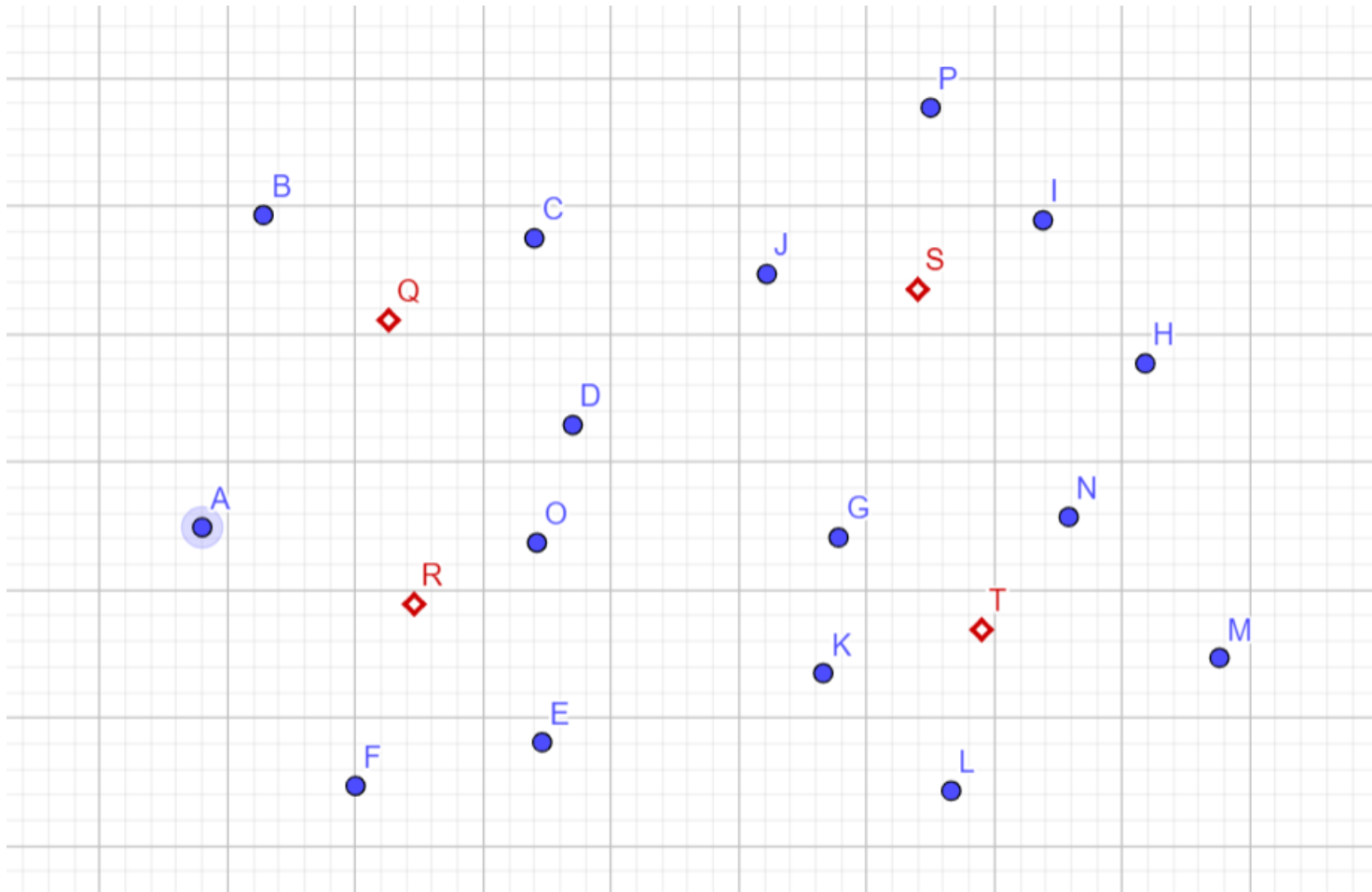
xq: the query vector

search scope
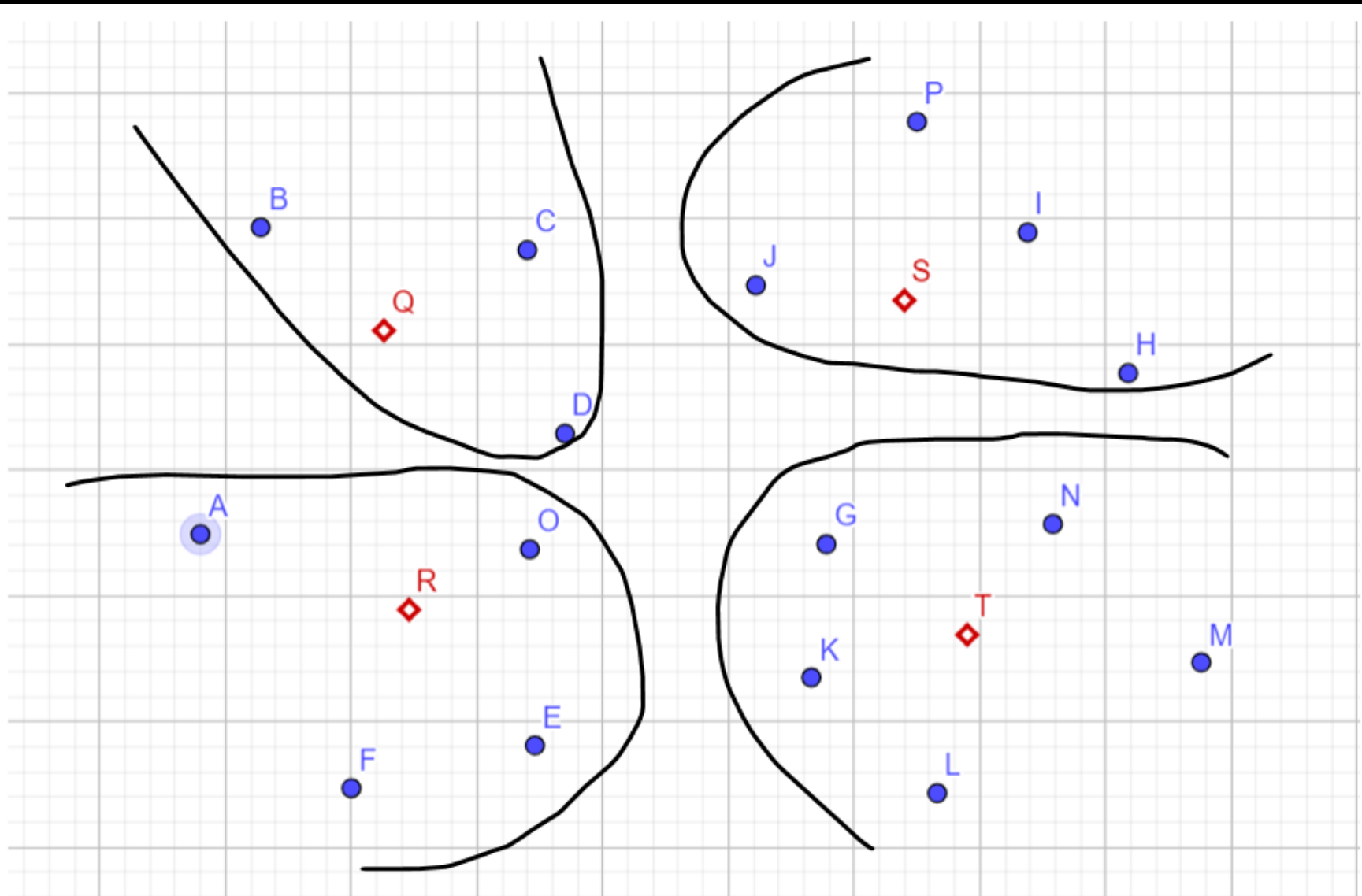nprobe = 8
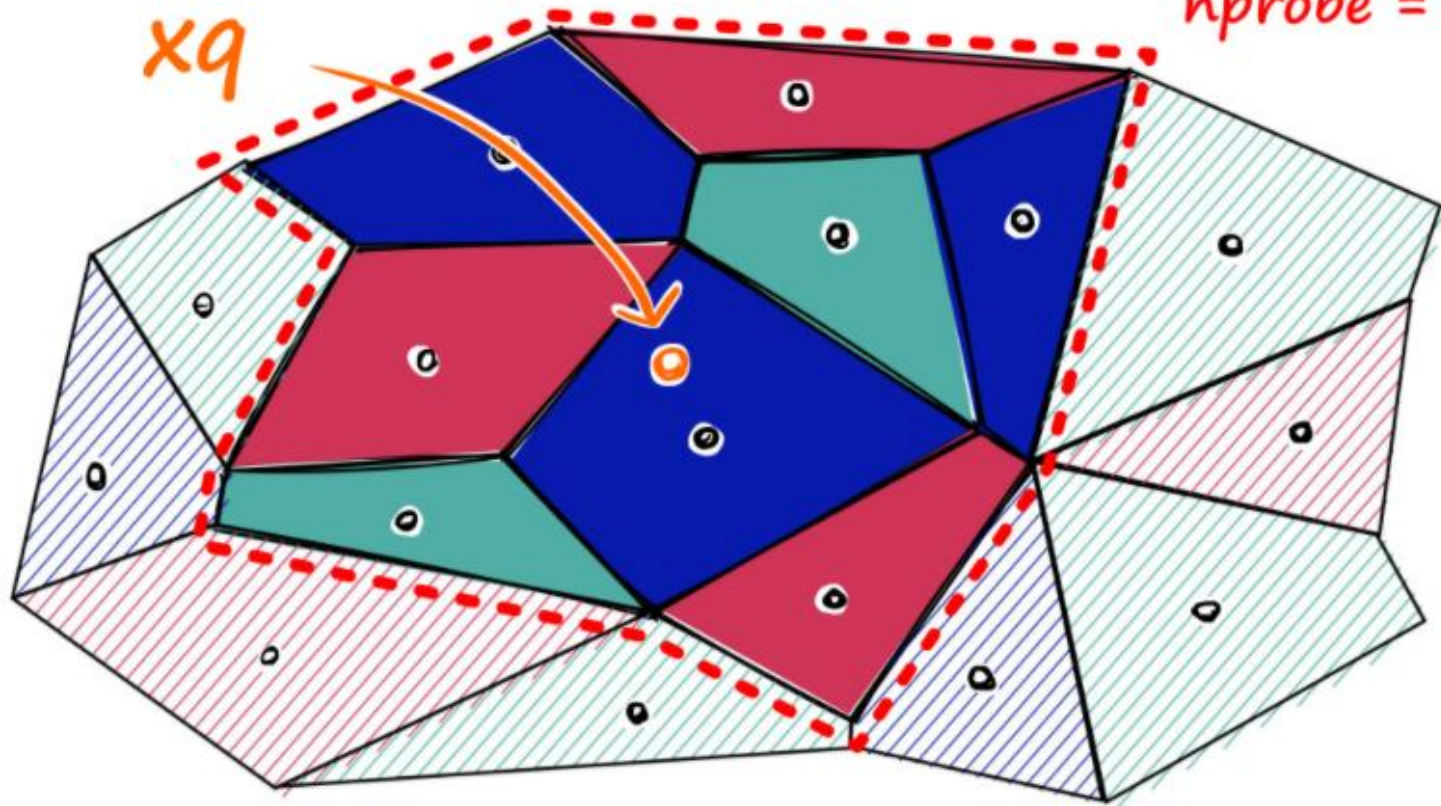
Increasing **nprobe** increases our search scope.

nprobe=8: select the top 8 nearby cells for searching.
(ranked by the distance between the query and the centroids)
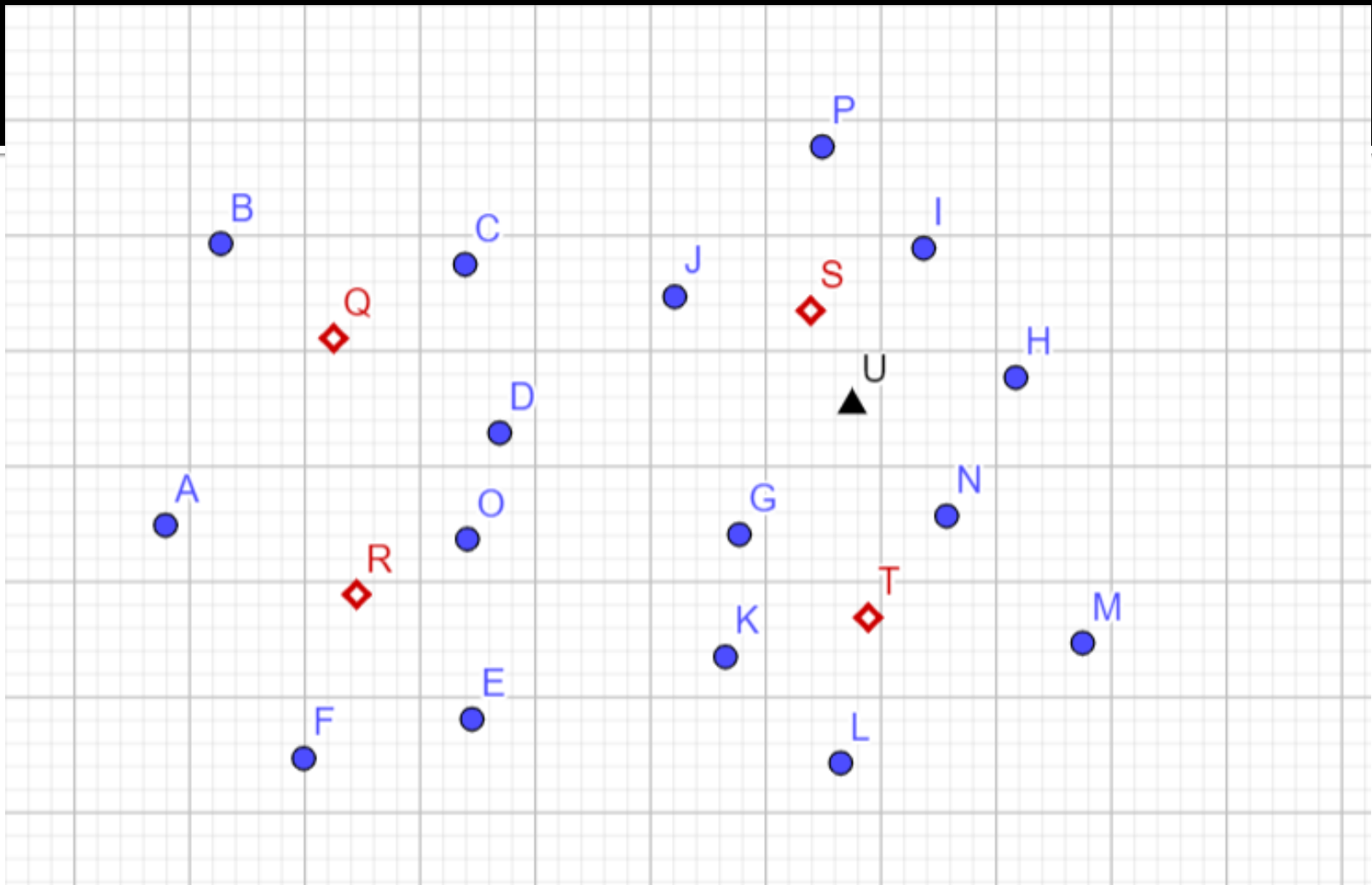
Inverted File Index Example

Q, R, S, T are centroids, all other points are the
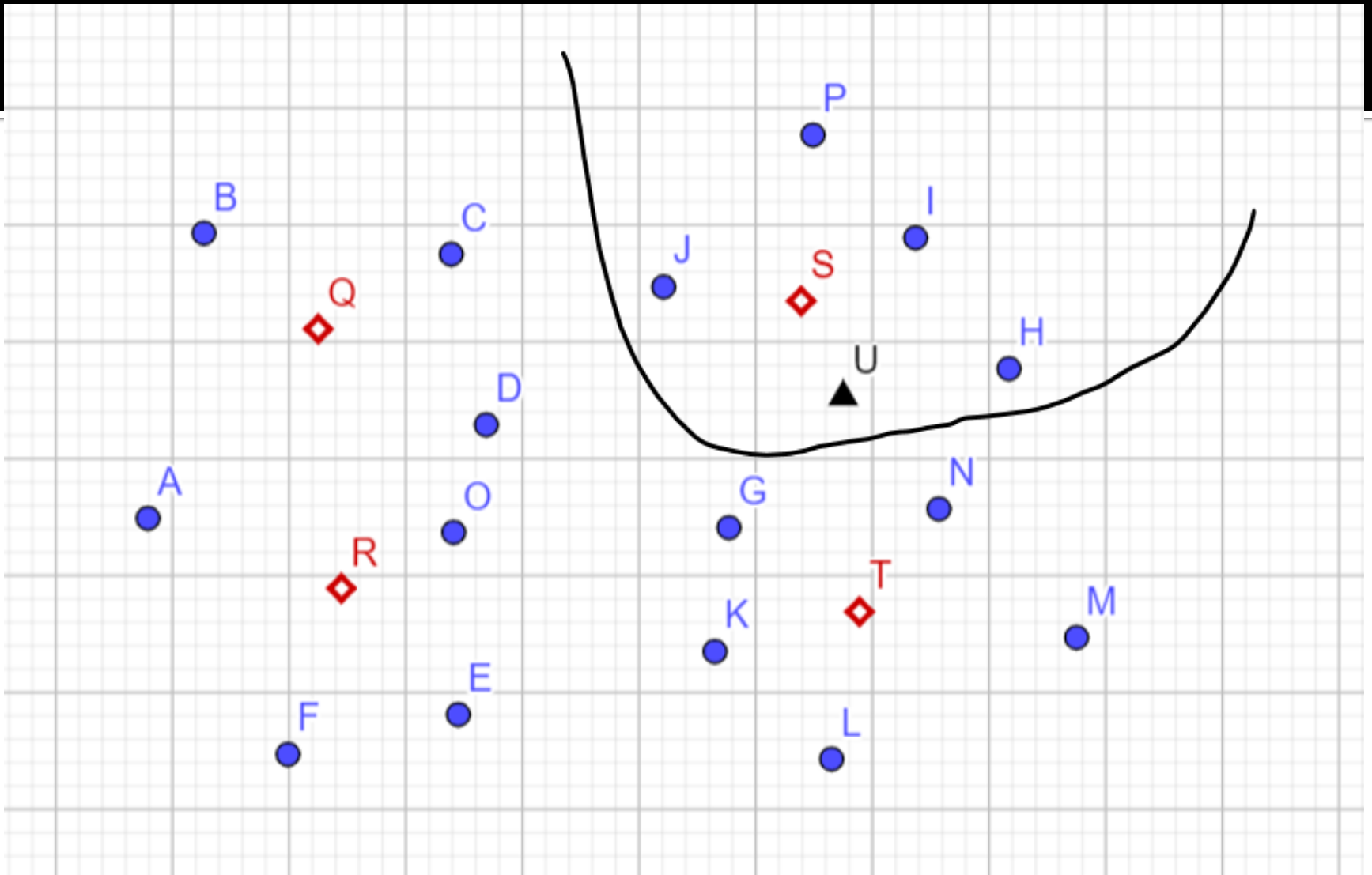data samples in the database.

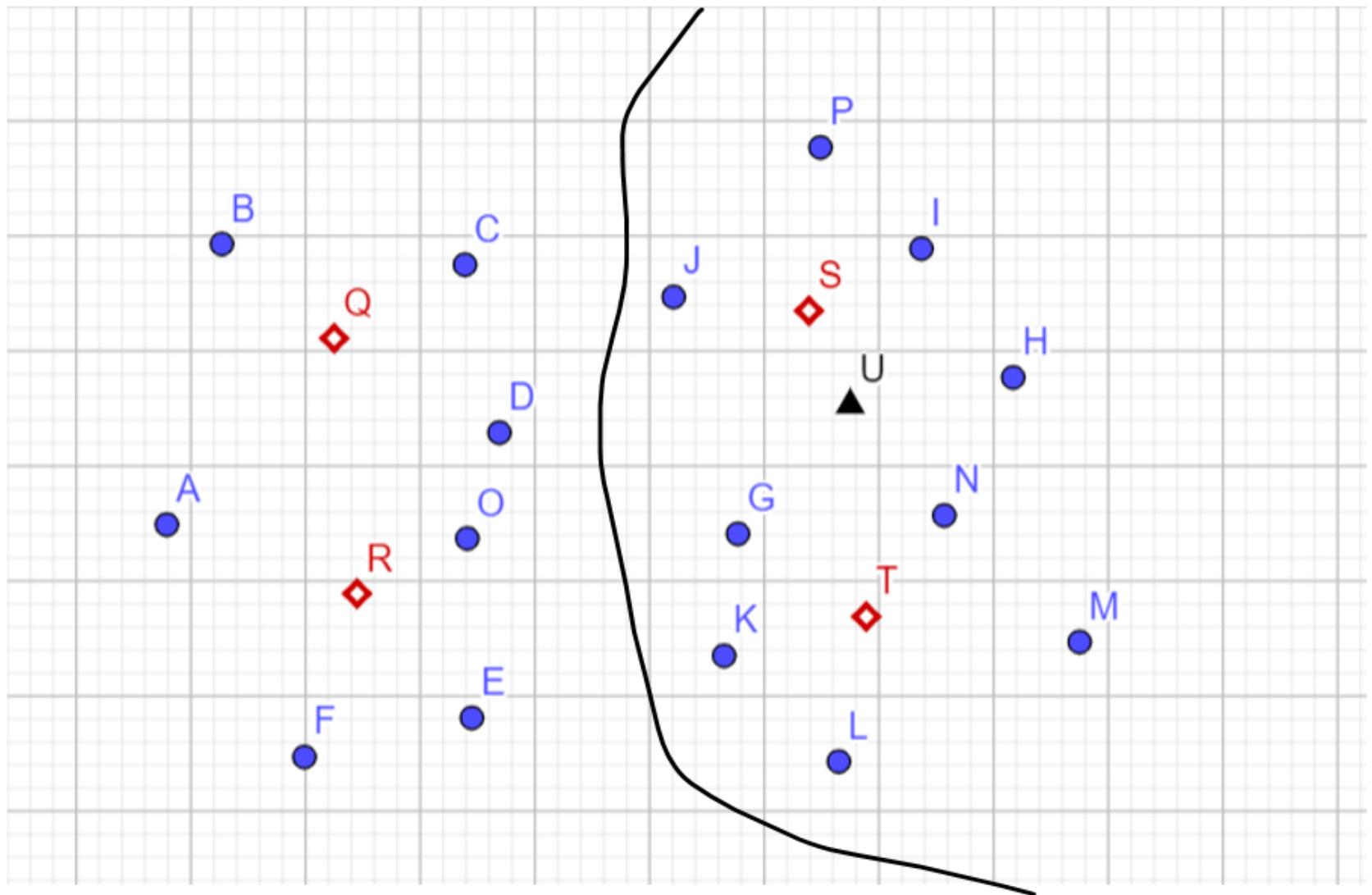- Q(a): assign the samples to the cells defined by the centroids.

one data point is assigned to its nearest centroid based on L2 distance

Q(b) Given the query sample U,
list the candidate samples in the database for similar search
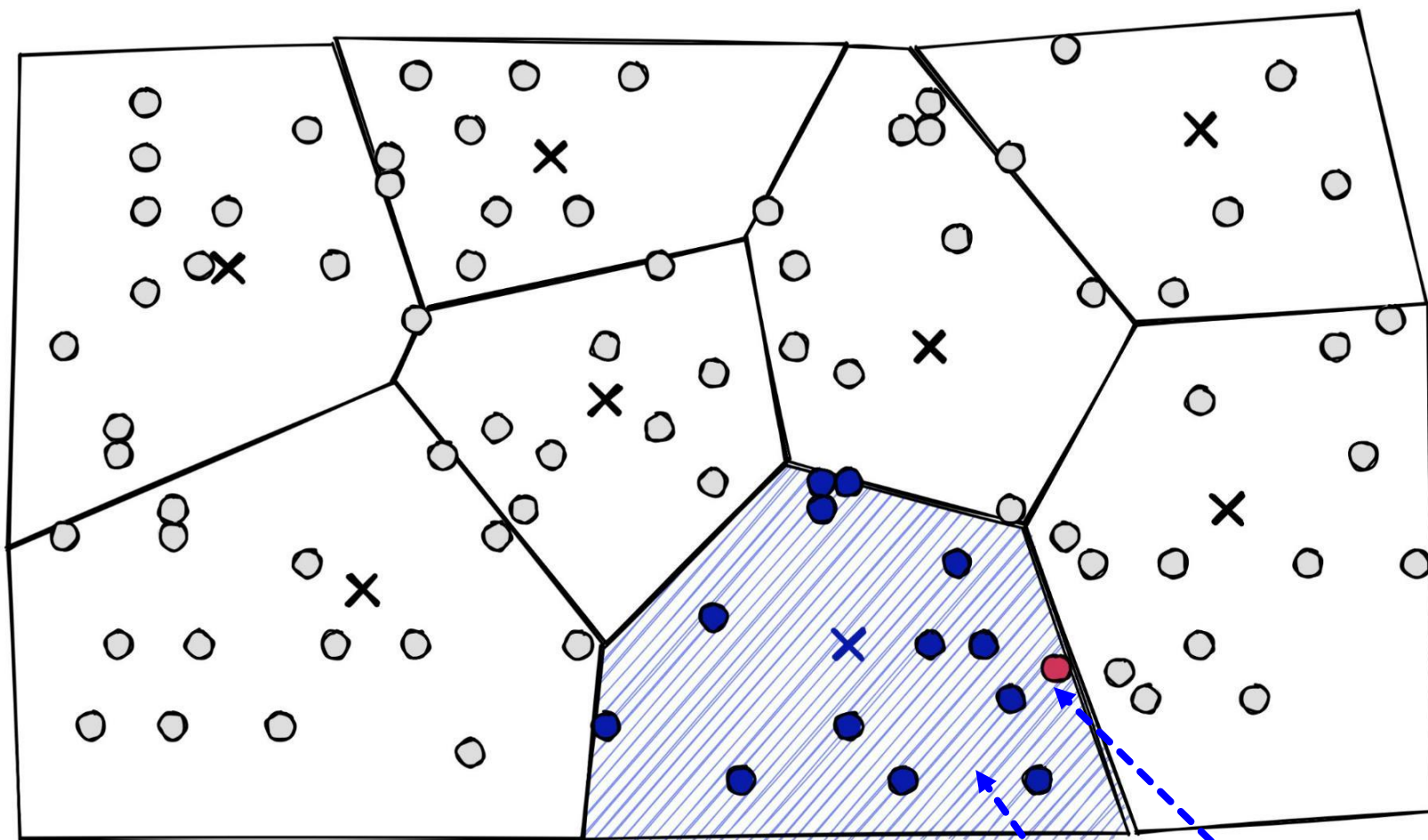when nProbe=1 and nProbe=2, respectively

nProbe=1: only search in 1 cell

nProbe=2: search in 2 cells (the top 2 closest cells)

PQ+IVF

Non-examinable!

- PQ+IVF
  - Product Quantization + Inverted File Index
  - A hierarchical solution
  - Much better performance in searching
  - Reduce search scope by using IVF

Step1: use IVF to identify search scope (cells).
IVF allows us to restrict our search in the target cells only.
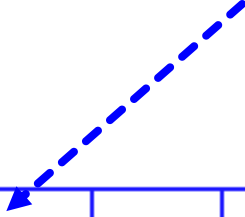
Target cell

Step2: perform PQ based linear search in the target cells. PQ generates
compresses vectors and uses fast distance calculation

query

51

Experiment:
Sift1M dataset, 2048 centroids for each subspace

L2 distance base linear search

	FlatL2	PQ	IVFPQ
Recall (%)	100	50	52
Speed (ms)	8.26	1.49	0.09
Memory (MB)	256	6.5	9.2

https://www.pinecone.io/learn/product-quantization/

Online resources

- FAISS
  - Faiss is a library for efficient similarity search and clustering of dense vectors.

  - https://github.com/facebookresearch/faiss/wiki