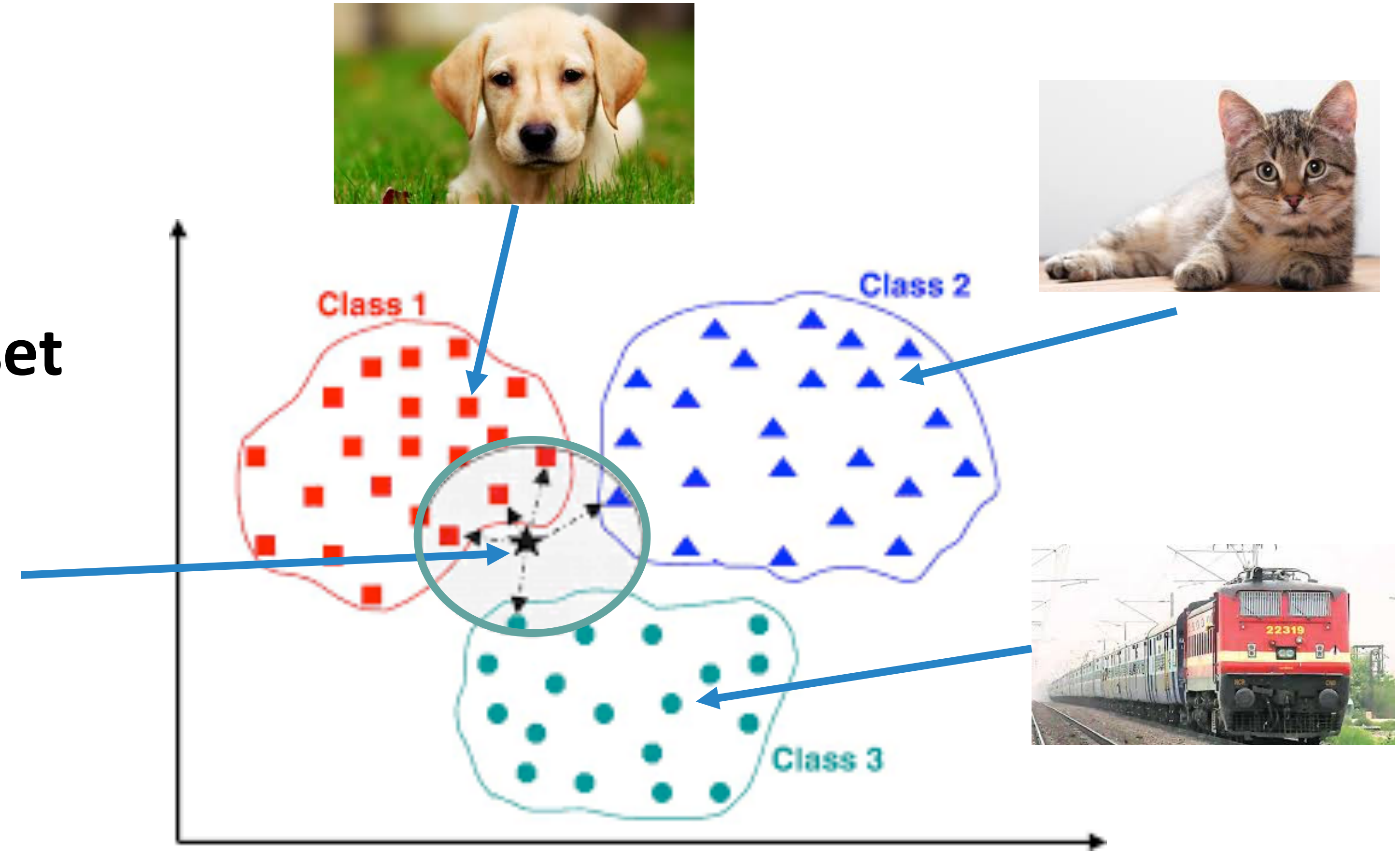
Nearest Neighbor Search

M1 Keisuke Fukuta

Nearest Neighbor Search

Find the nearest samples in dataset



Nearest Neighbor Search

Application

- Similar Image retrieval
- Feature matching

Nearest Neighbor Search

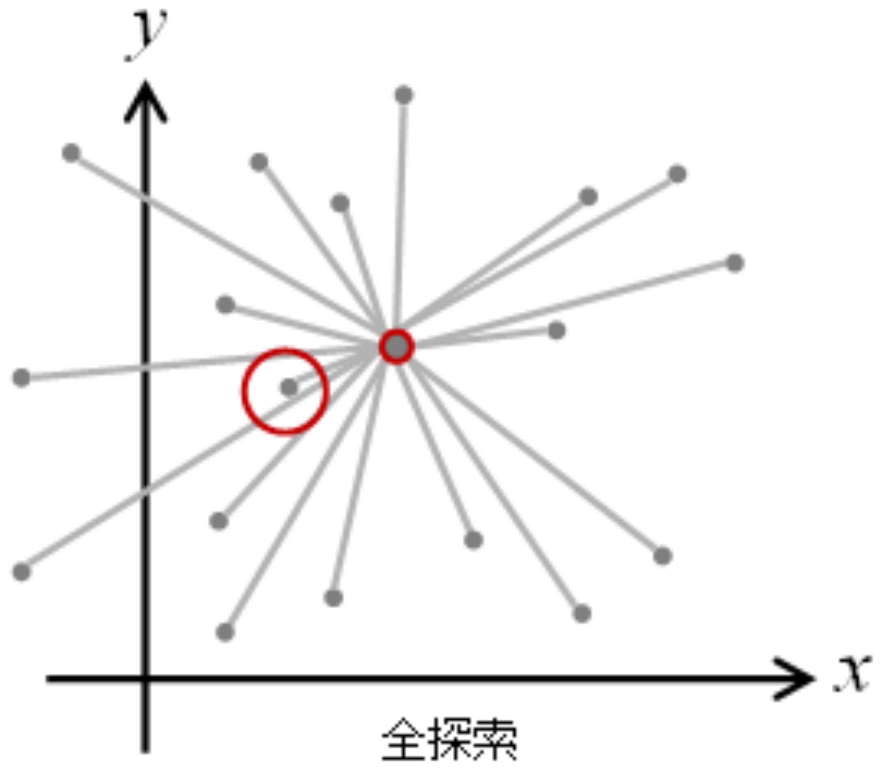
- Formulation
  - Data points $X \subset R^D$, input query $q \in R^D$
  - $NN(q) = arg\min_{x \in X} d(x, q)$
  - Data num : N, data dim: D

- Distance $d(x, y)$
  - Minkowski distance $d_P(x, y) = \left(\sum_{i=0}^{D} |x_i - y_i|^p\right)^{\frac{1}{p}}$
    - d=2 -> Euclid Distance, d=1 -> Manhattan Distance
  - Cosine Similarity
  - Hamming Distance

Linear Search

- Brute force search
- Calculate distances between query and all samples
- $O(DN)$

If N >> 1, time-consuming or intractable
Curses of dimentionality



Approximate Nearest Neighbor Search

- To find the exact nearest sample is time-consuming (curses of dimensionality)

→ **Approximate Nearest Neighbor**

$$x \in X \ \ s.t.\ d(x,q) \le (1+\epsilon)d(v',q)\ for\ any\ v' \in X$$

- Memory, Speed, Accuracy is trade-off

Approximate Nearest Neighbor Search

Tree-based

- kd-tree
  - Approximate kd-tree
  - Randomized kd-tree
- Hierarchial k-means
- FLANN (RKD & HKM)
- Ball-tree
- Vp-tree

Hash-based

- Locality sensitive hash (LSH)
- Spectral Hashing (SH)

Quantization

- LVQ
- Product Quantization (PQ)
- IDFADC
- LOPQ

KD-tree (k-dimensional tree)

Idea

- Make good data structure beforehand

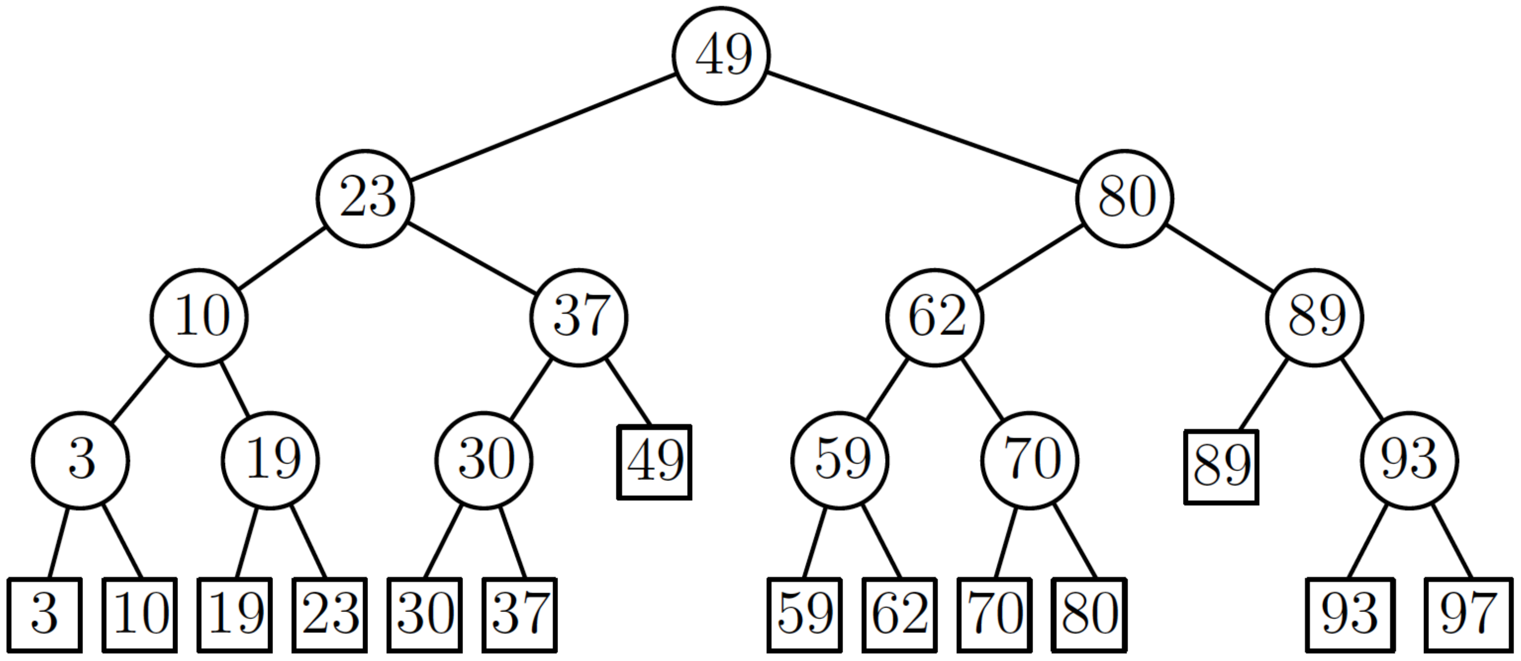
→ Utilize *binary tree*

- Guess samples (region) which is likely to be the nearest

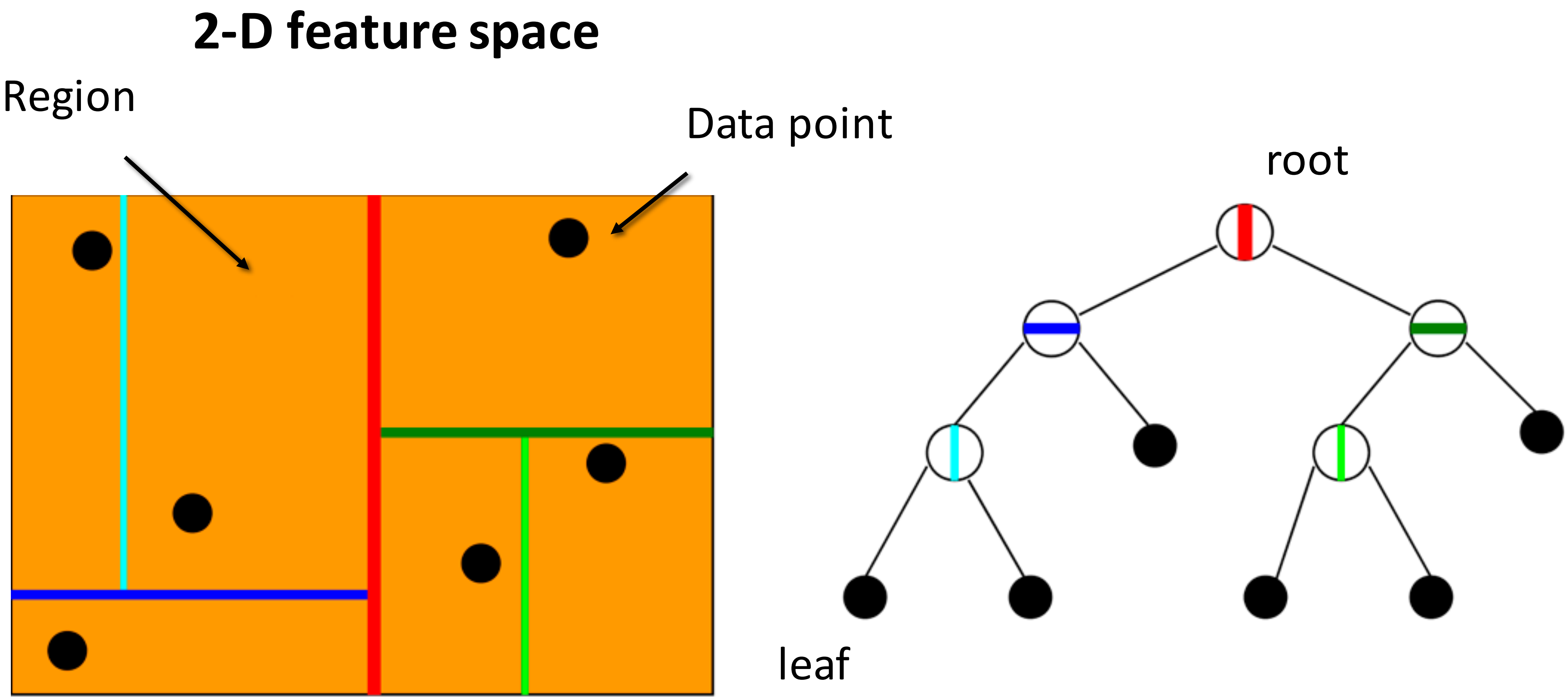
- Verification to find the exact nearest

Binary Tree

- Left node is smaller than parent
- Right node is bigger than parent



KD-tree

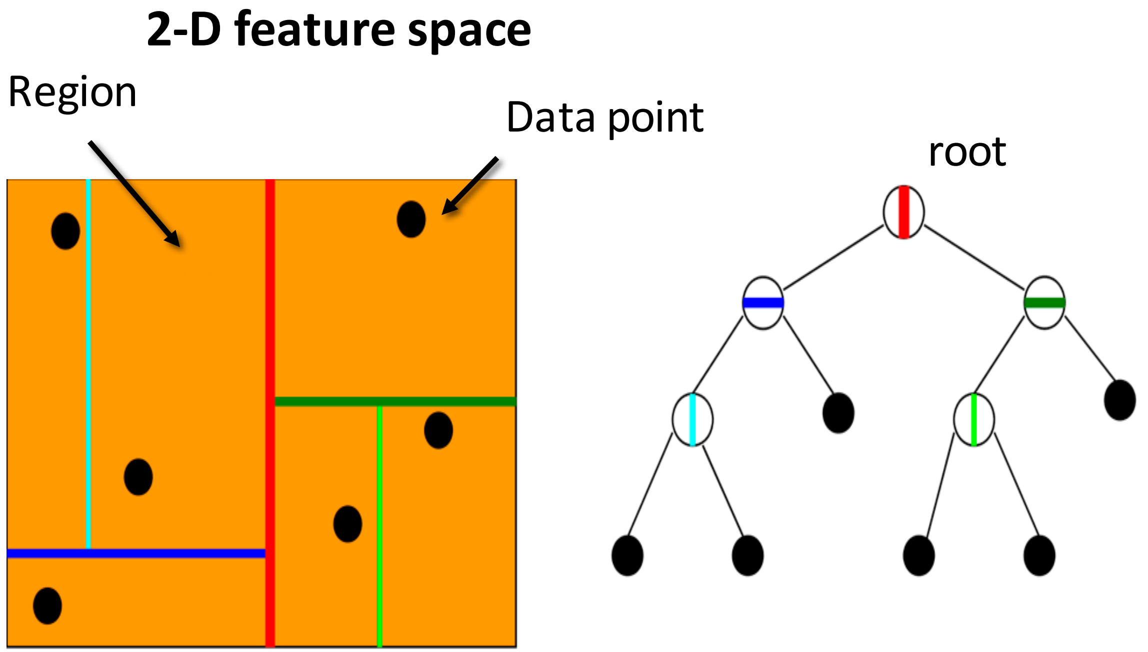


KD-tree

How to build tree ??

1. Choose axis whose variance is the largest
2. Split region with median
3. Repeat 1,2 in leaf nodes

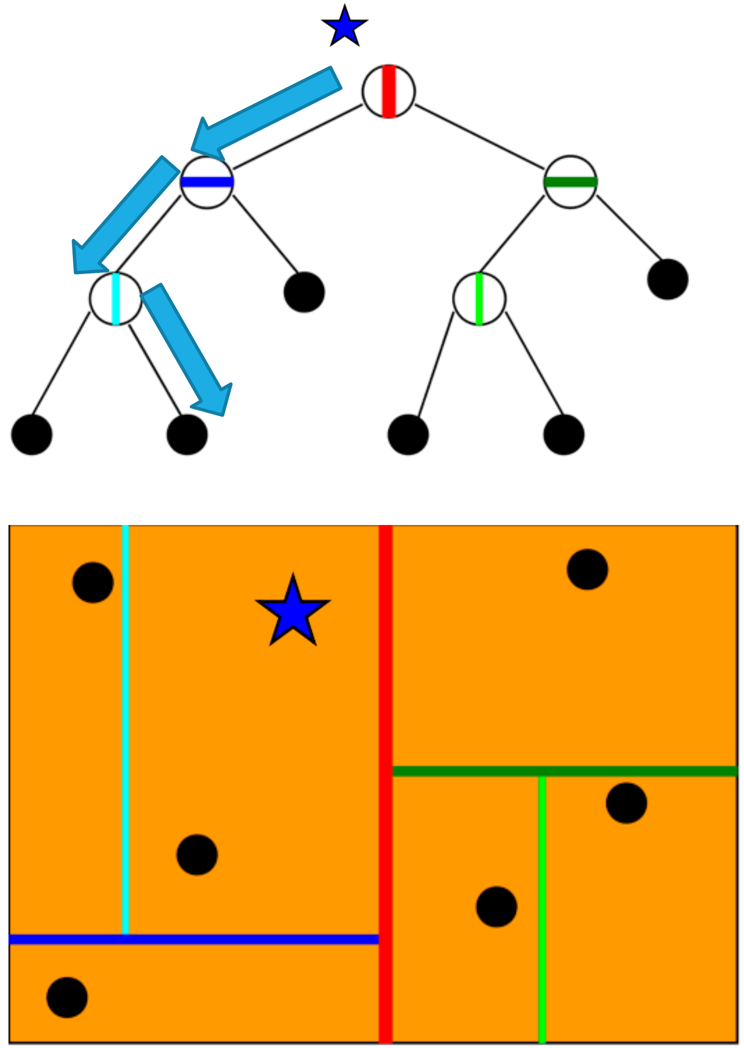
(There are more smart splitting rule)



KD-tree

How to search?

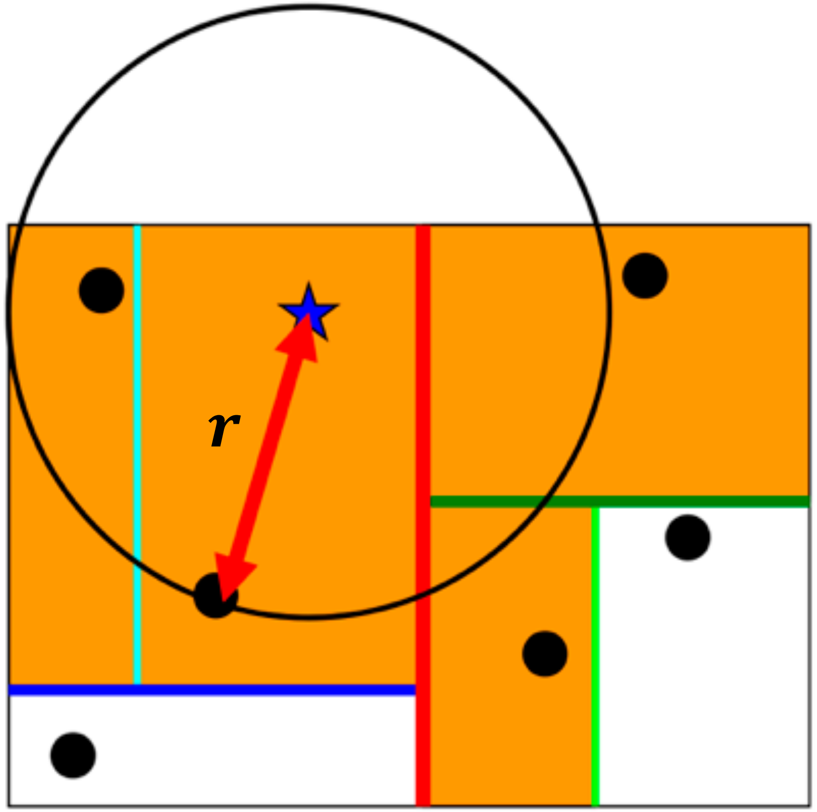
1. Binary Search
   - Find the region that contains input query with binary search



KD-tree

How to search?

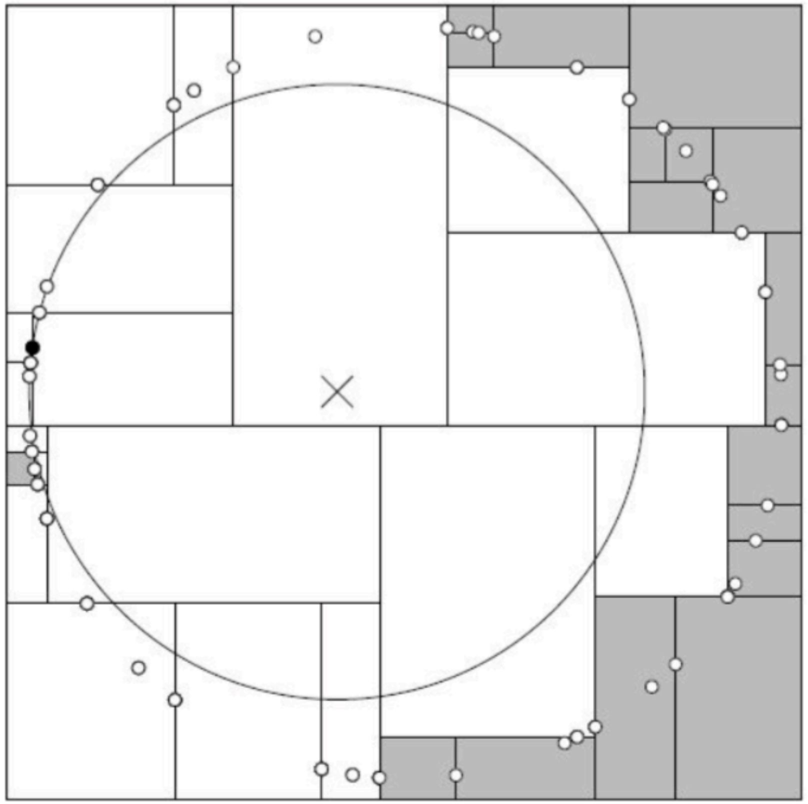
1. Binary Search
   - Find the region that contains input query with binary search
2. Back track
   - Find temporal solution within the region
   - Find the region within r, and compare with temporal solution
   - Traverse every region



KD-tree

- $O(D\log_2 N)$ (for binary search)

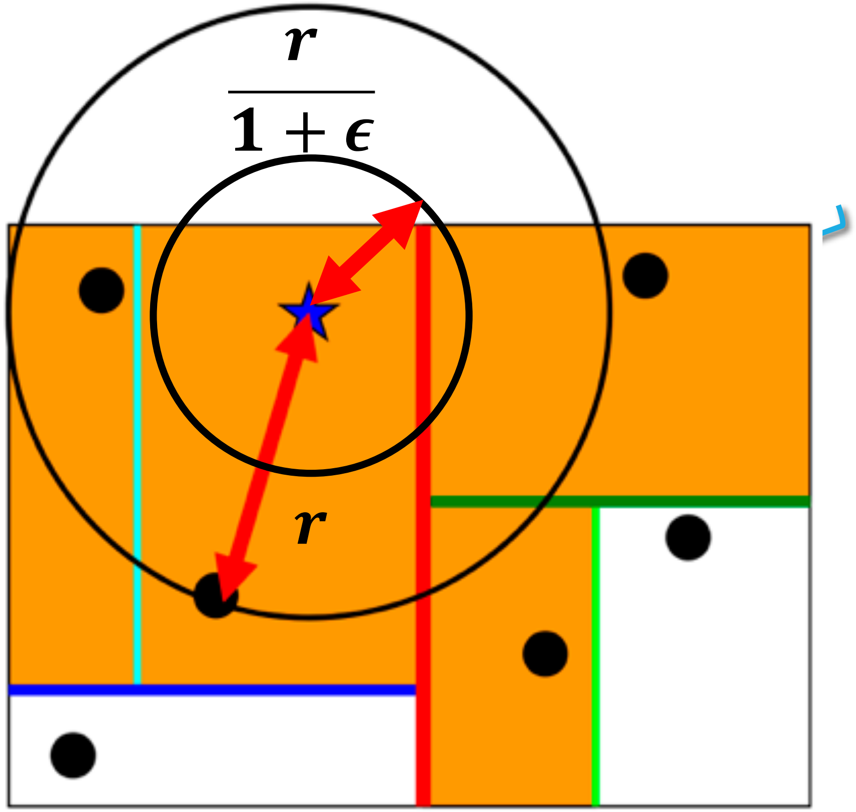
- In some case, almost same as linear search (curses of dimensionality)
  - Worst case -> $O(D\,N^{1-\frac{1}{D}})$
  - N >> 2^D is required



Kd-tree

Approximate kd-tree

- Introduce approximate parameter ϵ
  - if $\epsilon = 0$ -> equivalent to exact neighbor
- $d(x, q) \leq (1 + \epsilon)\, r_n$ (r_n : the true nearest)



Randomized kd-tree

- Make m different trees

KD-tree

- Pros
  - Good accuracy
  - High speed
- Cons
  - Very slow in high dimension

Approximate Nearest Neighbor Search

Tree-based	Hash-based	Quantization
• kd-tree • Approximate kd-tree • Randomized kd-tree • Hierarchial k-means • FLANN (RKD & HKM)	• Locality sensitive hash (LSH) • Spectral Hashing (SH)	• LVQ • Product Quantization (PQ) • IDFADC • LOPQ

Locality Sensitive Hash

- Hash
  - Hash function h(x)
  - $O(1)$
  - Ex. Str -> int

```
int hash(char *s) {
    int i= 0;
    while (*s) i += *s++;
    return i%100;
}
```

one	22	six	40
two	46	seven	45
three	36	eight	29
four	44	nine	26
five	26	ten	27

Collision

- Locality Sensitive

$$d(v,q) \leq r_1 \Rightarrow \Pr[h(q) = h(v)] \geq p_1$$
$$d(v,q) > r_2 \Rightarrow \Pr[h(q) = h(v)] < p_2$$

Locality Sensitive Hash

Hash function h(x) depends on distance metrics

- Hamming distance
- L_p distance
- Jaccard index
- Cosine similarity

Locality Sensitive Hash

Example : Hamming Distance

- $q = 10101$, $p1 = 10001$, $p2 = 00111$

-> Hamming distance $\mathrm{d}(p_1; q) = 1$, $d(p_2; q) = 2$

- h(x): randomly choose i and return i-th bit of input

-> h(x) is locally sensitive

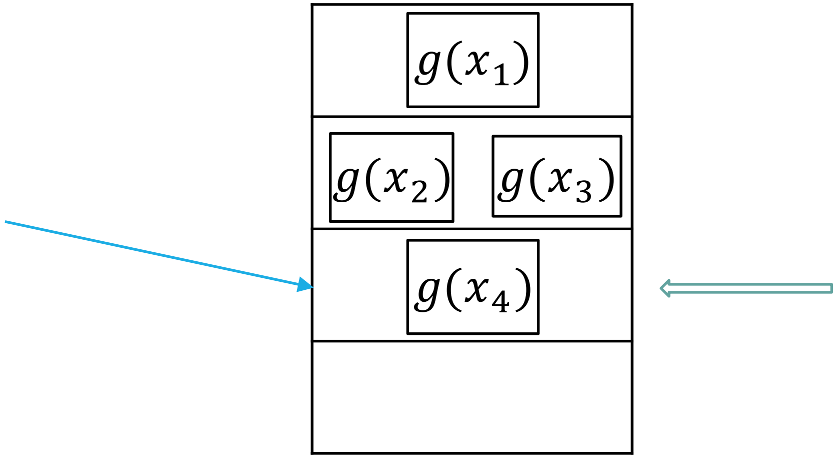
$$P[h(p) = h(q)] = 1 - \frac{d(p,\ q)}{D}$$

Locality Sensitive Hash

- Prepare $\mathrm{g(v)} = \begin{bmatrix} h_1(v) \\ h_2(v) \\ \vdots \\ h_k(v) \end{bmatrix}$

- Translate data points to hash X -> g(X)

$q \; -> \; g(q)$



Search only in this backet

Hash table

Locality Sensitive Hash

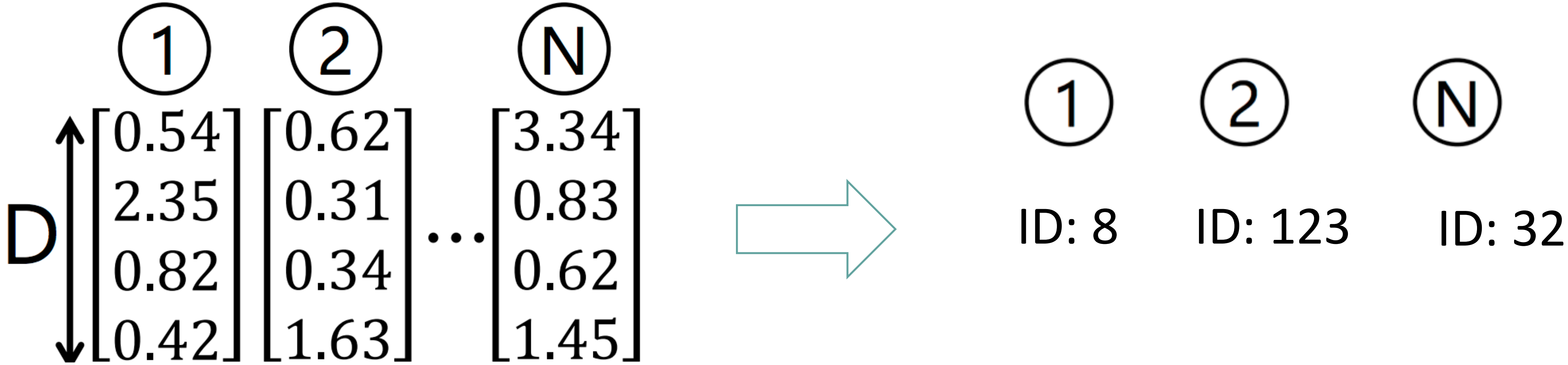
- Pros
  - High-speed
  - Able to determine approximation or performance theoretically

- Cons
  - Probabilistic search -> low accuracy

Approximate Nearest Neighbor Search

Tree-based	Hash-based	Quantization
• kd-tree • Approximate kd-tree • Randomized kd-tree • Hierarchial k-means • FLANN (RKD & HKM)	• Locality sensitive hash (LSH) • Spectral Hashing (SH)	• LVQ • Product Quantization (PQ) • IDFADC • LOPQ

Quantization

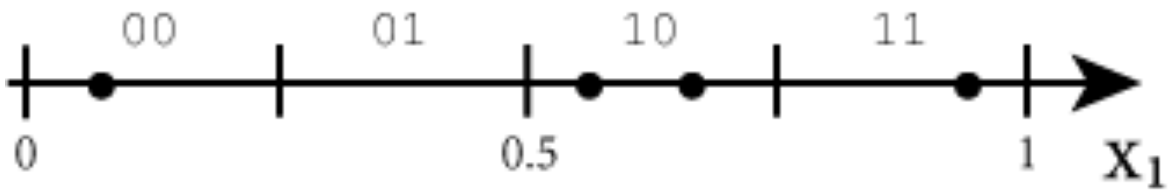


Ex. $N = 10^9,\ D = 128,\ float32 \rightarrow 512GB$

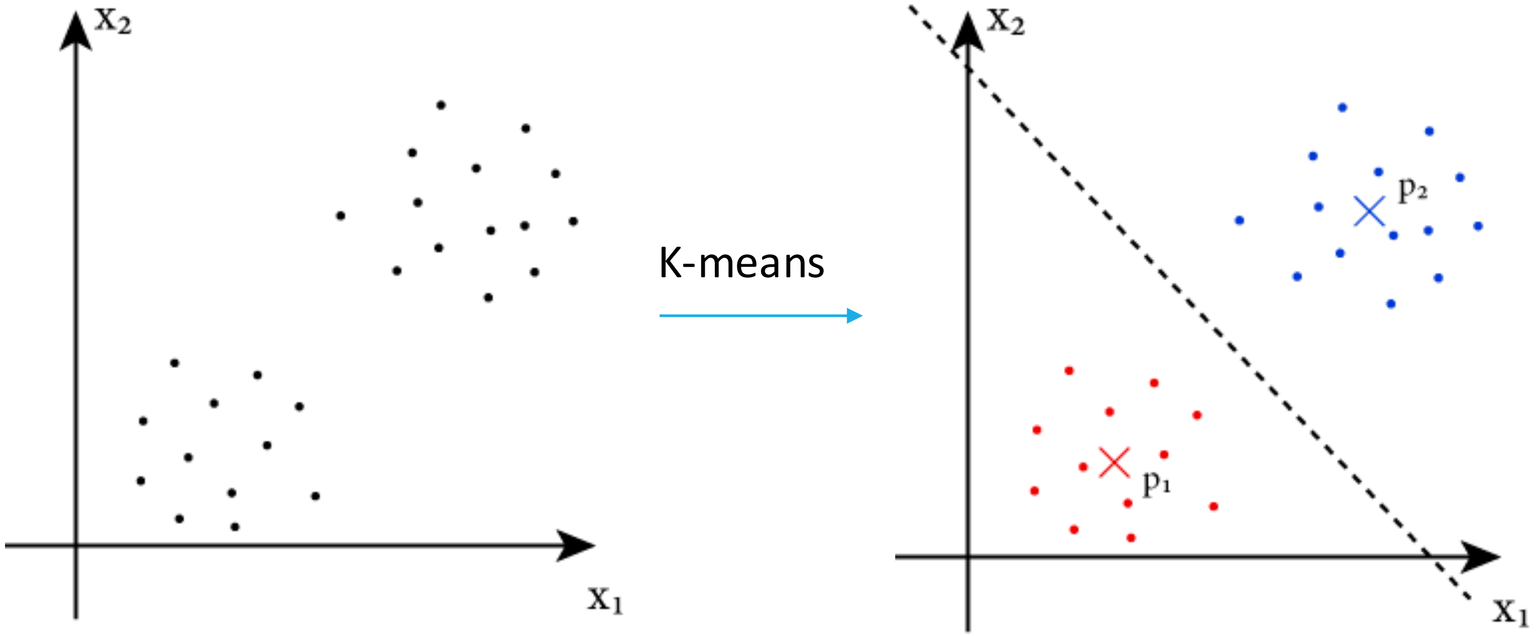
Ex. $N = 10^9,\ 32bit \rightarrow 4GB$

Quantization

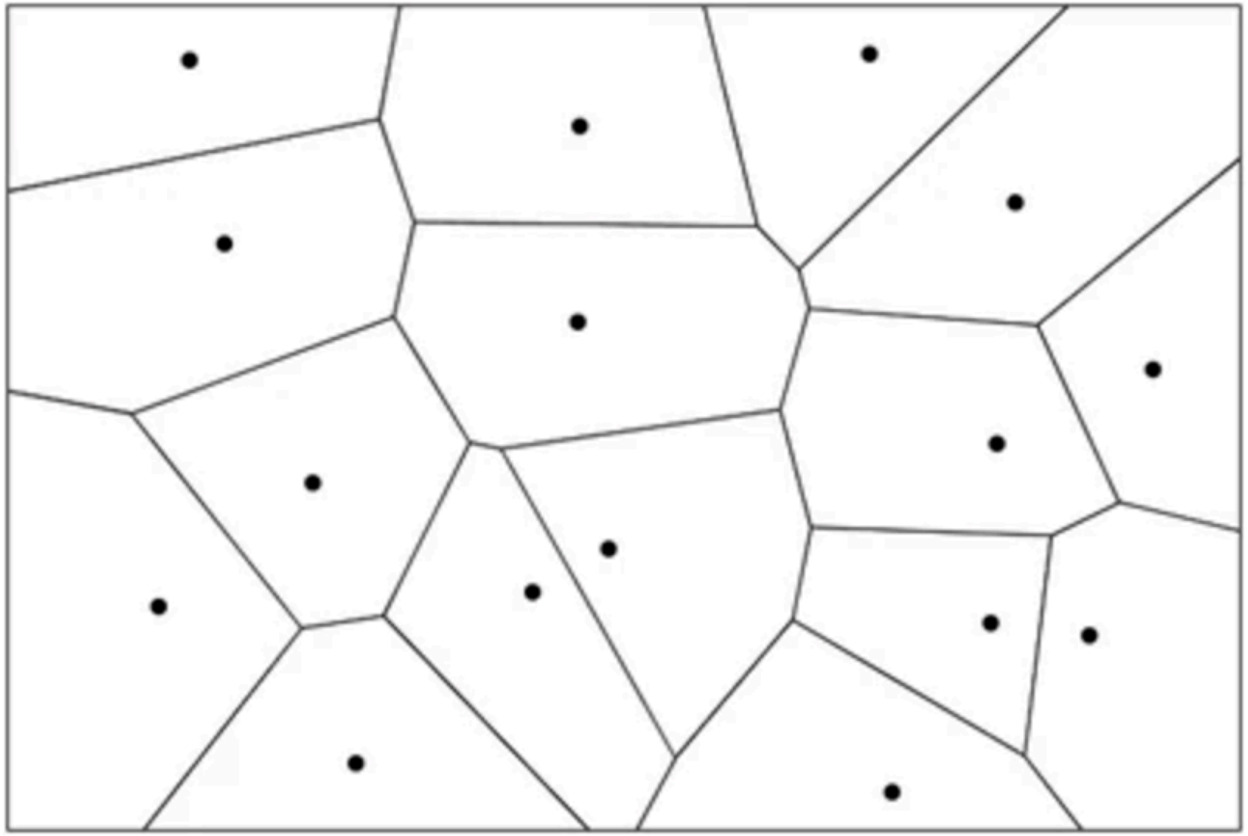
- Scalar Quantization
  - Ex. Float 32 bit -> 2bit



- Vector Quantization
  - Representative vectors
  - (float, float) -> 1bit
  - Code book size k
    -> $\log_2 k$ bit



Voronoi diagram

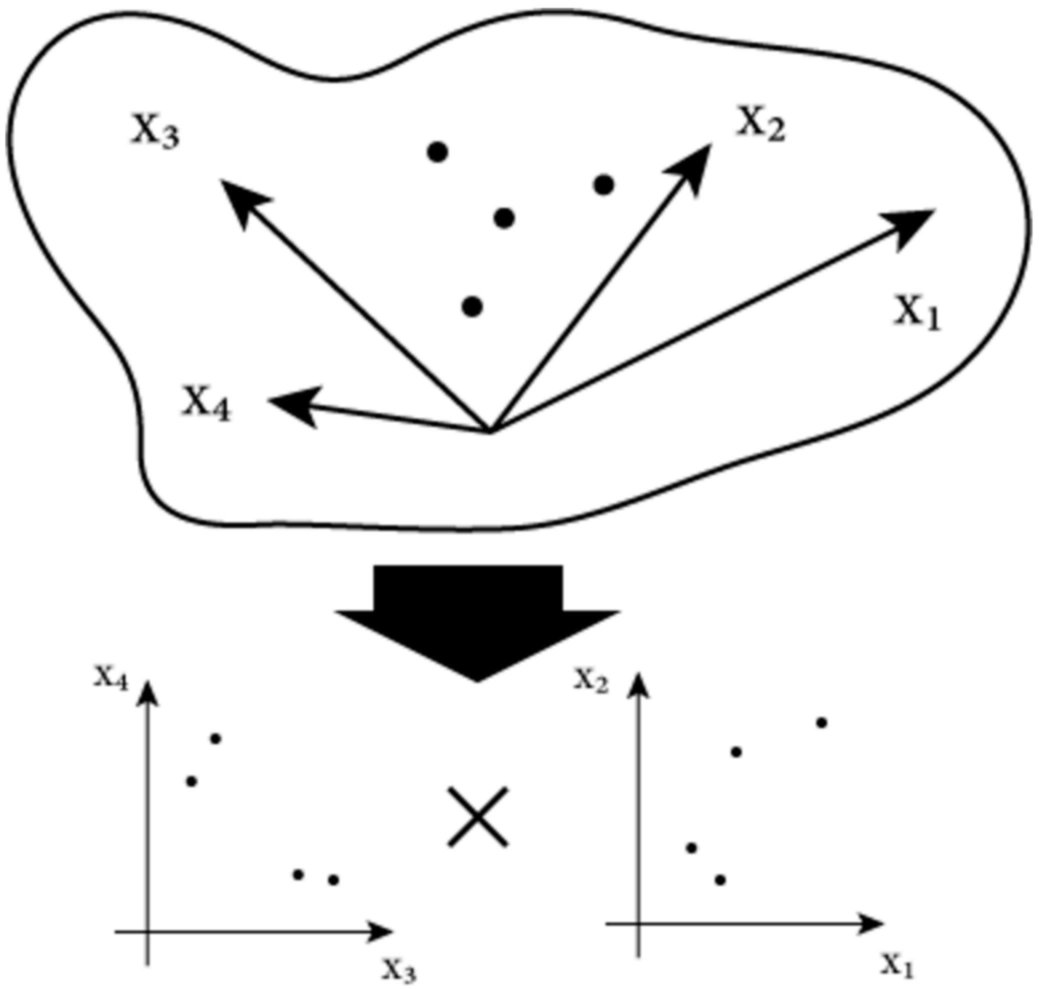


$\mathbf{k} \propto \boldsymbol{exp(D)}$

Cannot apply to high dimension

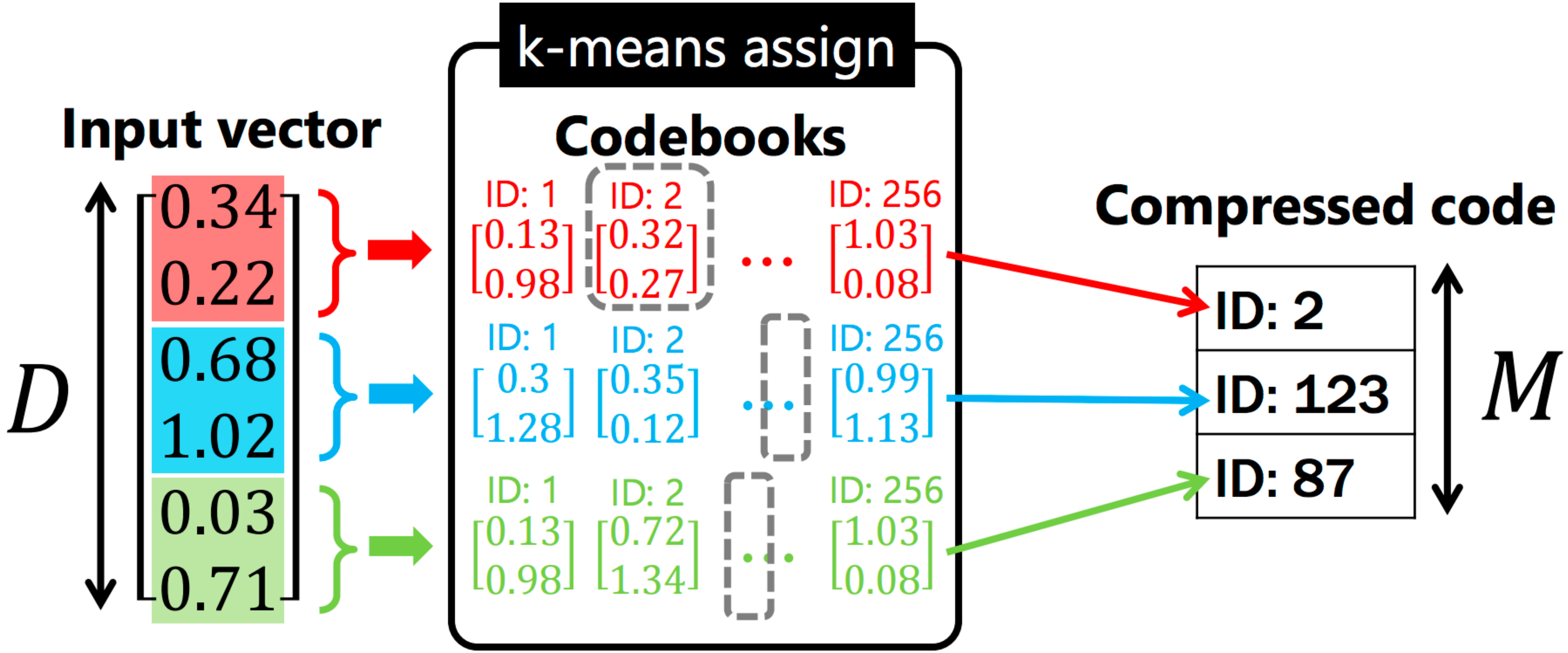
Product Quantization

- Divide into sub-vector
- Product (not tensor-product)



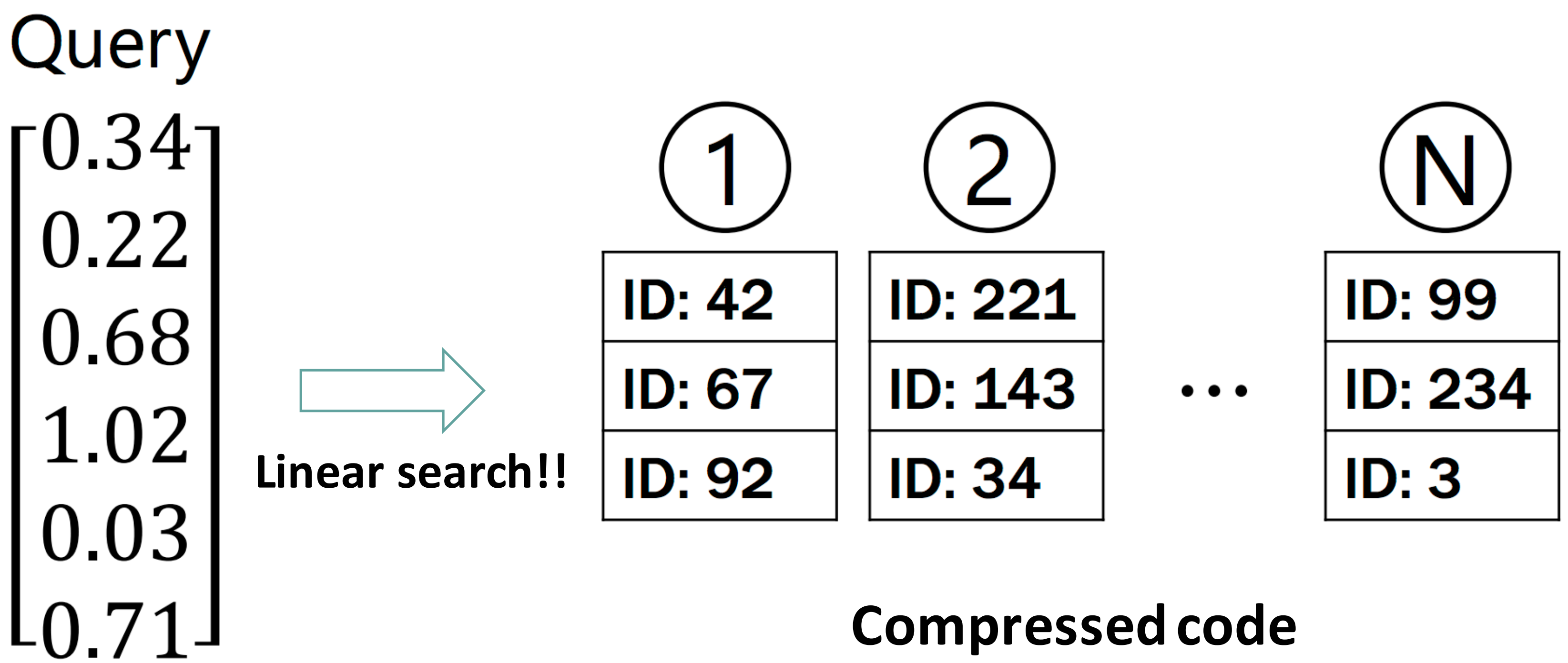
Product Quantization

Divide into sub-vector and k-means in each sub-vectors

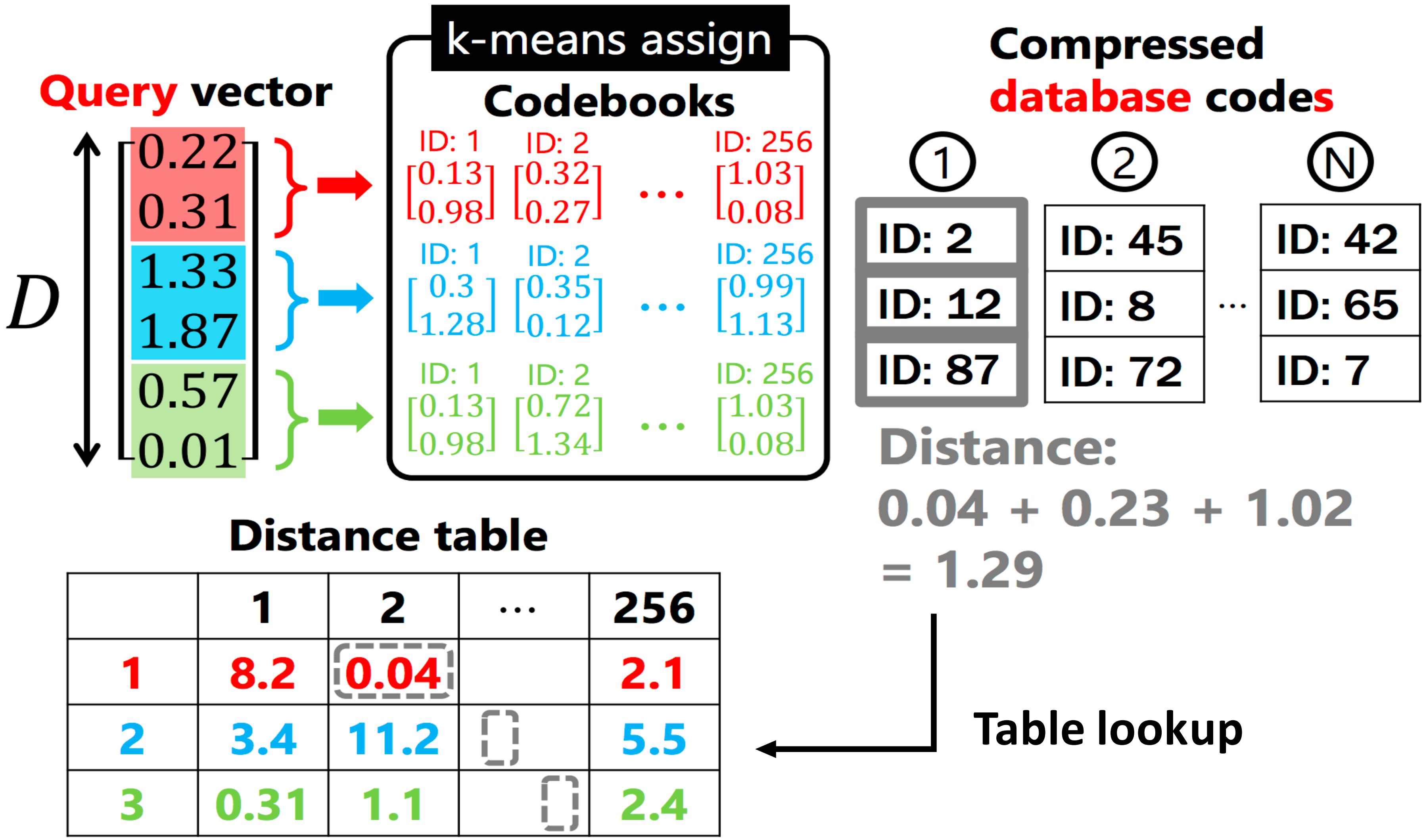


http://yusukematsui.me/project/survey_pq/doc/ann_lecture_20161202.pdf

Product Quantization



Product Quantization



Product Quantization

- Simple
- Memory efficient
- Precise Approximation
  - divide into 256^M (code book size: 256 * D, VQ : $256^M * D$)

- But, still O(N) (linear search)

-> combine with coarse quantization beforehand

-> apply PQ to residual of representative vector and input vector

Approximate Nearest Neighbor Search

Tree-based	Hash-based	Quantization
• kd-tree • Approximate kd-tree • Randomized kd-tree • Hierarchial k-means • FLANN (RKD & HKM)	• Locality sensitive hash (LSH) • Spectral Hashing (SH)	• LVQ • Product Quantization (PQ) • IDFADC • LOPQ

Reference