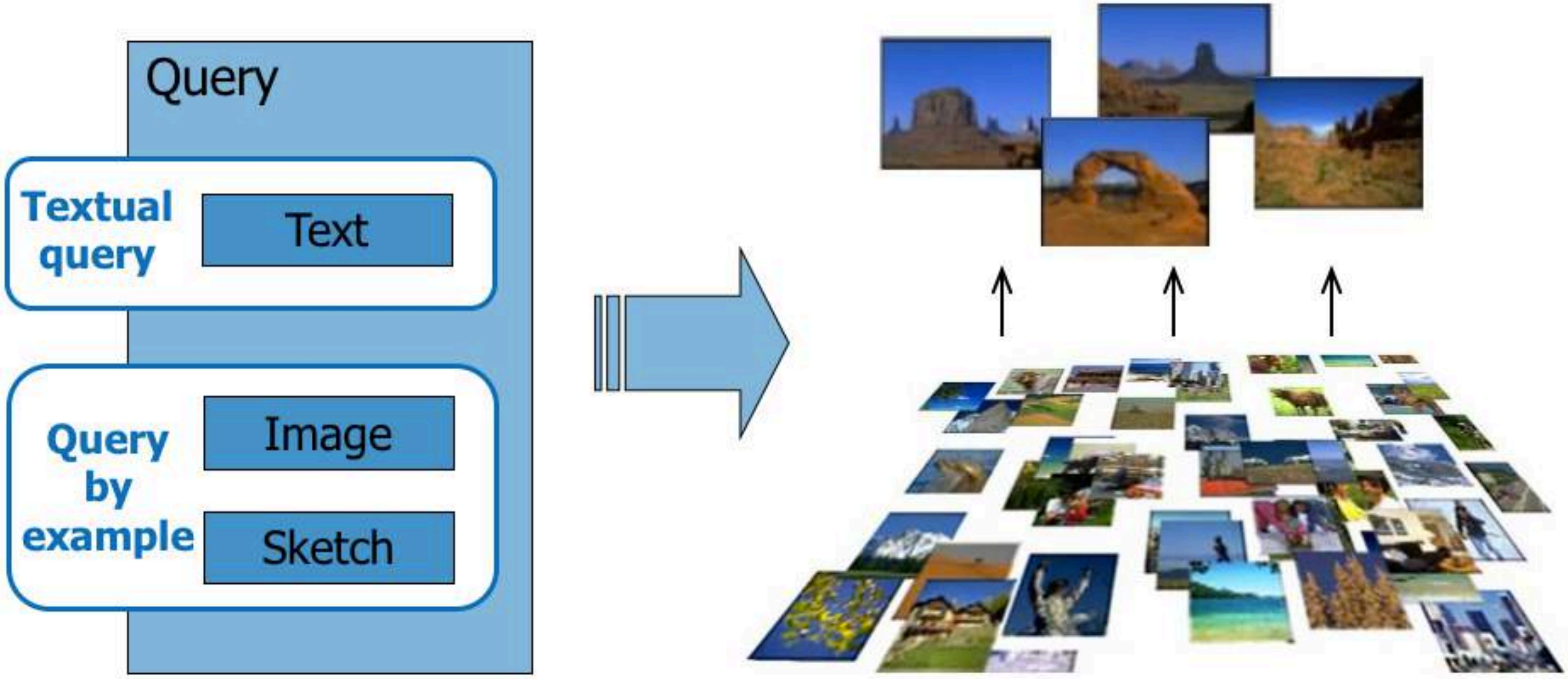
Image Indexing

Eduardo Tavares

Image Retrieval

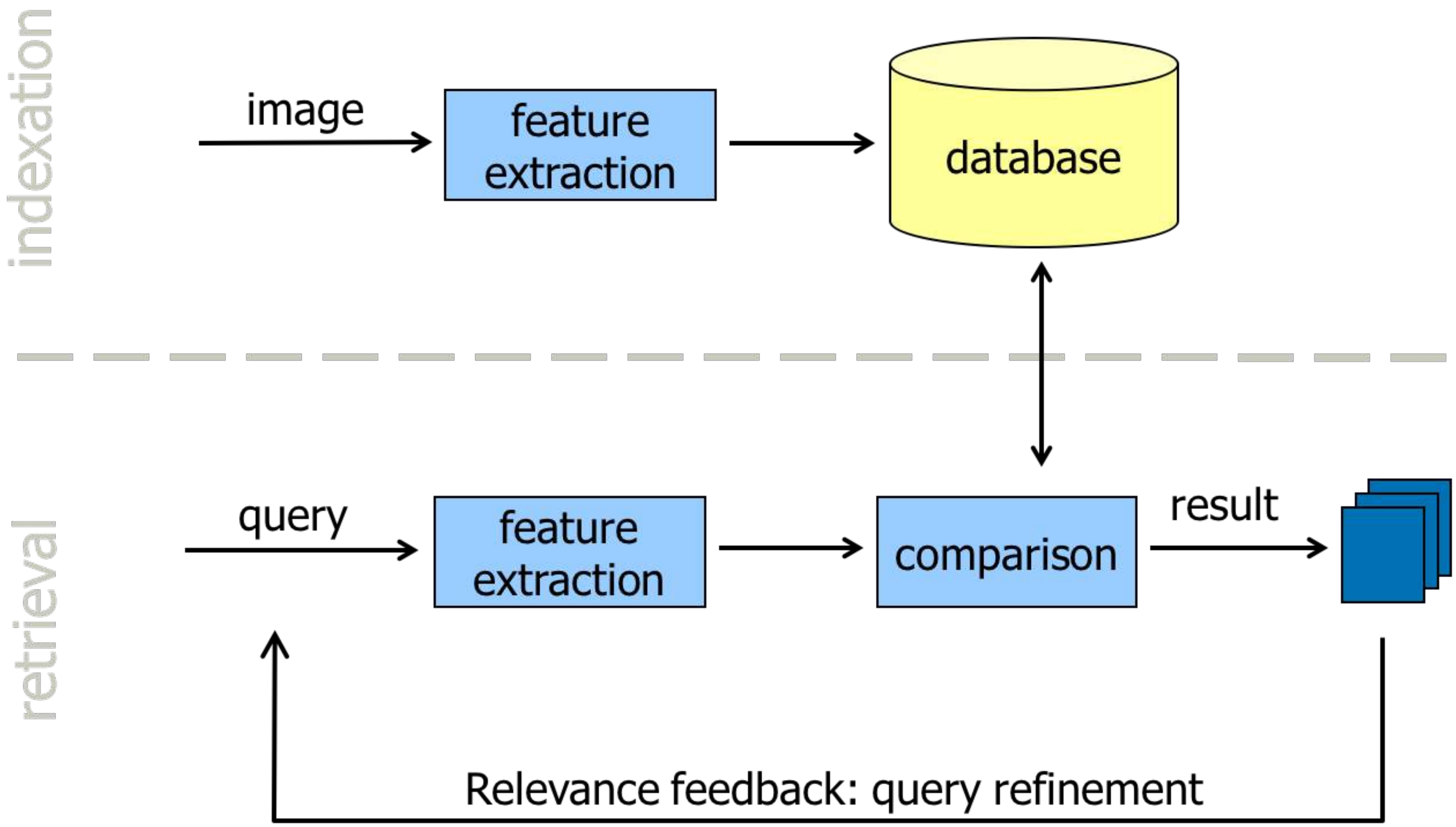
- ✓ Description Based Image Retrieval (DBIR)
- ✓ Content Based Image Retrieval (CBIR)



Levels of image retrieval

- Level 1: Based on color, texture, shape features
  - Images are compared based on low-level features, no semantics involved
- Level 2: Bring semantic meanings into the search
  - E.g. identifying human beings, horses, trees, beaches
  - Requires retrieval techniques of level 1
- Level 3: Retrieval with abstract and subjective attributes
  - Find pictures of a particular birthday celebration
  - Find a picture of a happy beautiful woman
  - Requires retrieval techniques of level 2 and very complex logic

Common components of CBIR system



Problems and directions

- ✓ Low-level feature extraction
  - ✓ How to represent an image in a compact and descriptive way?
  - ✓ How to compare features, and, thus, images?
- ✓ High dimensional indexing
  - ✓ How to index huge amounts of high dimensional data?
- ✓ Visual interface for image browsing
  - ✓ How to visualize the results?

Image features

Textual/metadata features

Levels of image content

- Semantics
- Shape
- Texture
- Color, lightness

Low-level features / visual features
(signatures, descriptors)

Image features

Image features

Textual

Visual (low-level)

Annotations and metadata:

- tags/keywords;
- creation date;
- geo tags;
- name of the file;
- photography conditions (exposition, aperture, flash...).

Features extracted from pixel values:

- color descriptors;
- texture descriptors;
- shape descriptors;
- spatial layout descriptors.

Image features

Low-level features

Global

Describes the whole image:

- average intensity;
- average amount of red;
- ...

All pixels of the image are processed.

Local

Describes one part of the image:

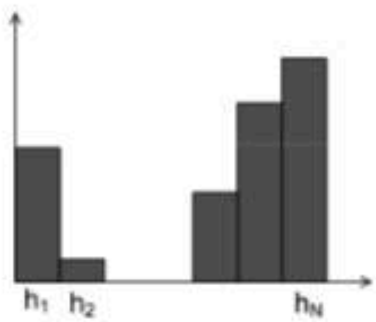
- average intensity for the left upper part;
- average amount of red in the center of the image;
- ...

Segmentation of the image is performed, pixels of a particular segment are processed to extract features.

Color features

Color features

Color histograms



$$F(I) = (h_1^I, h_2^I, \ldots, h_N^I)$$

Metrics: L_1, L_2, L_∞

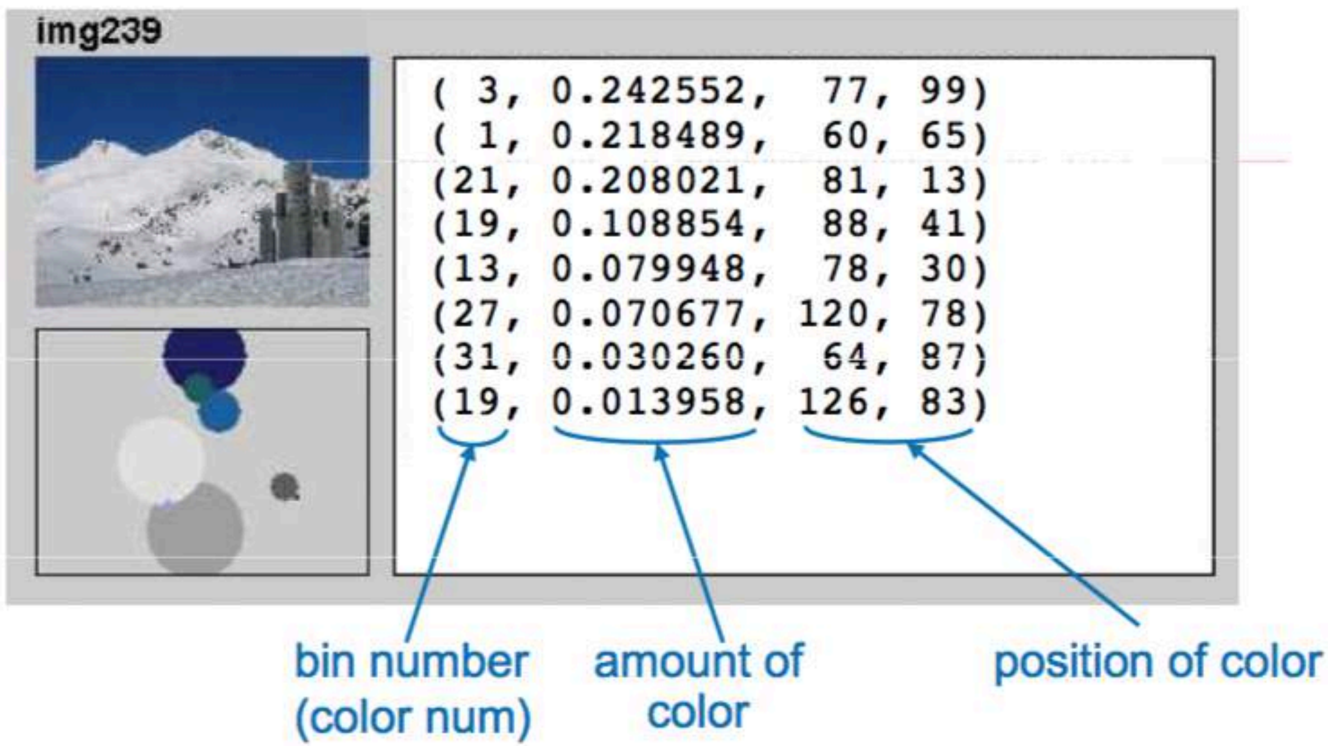
Color moments

Statistical moments for every color channel

$$F(I) = (E_1^I, E_2^I, E_3^I, \sigma_1^I, \sigma_2^I, \sigma_3^I, s_1^I, s_2^I, s_3^I)$$

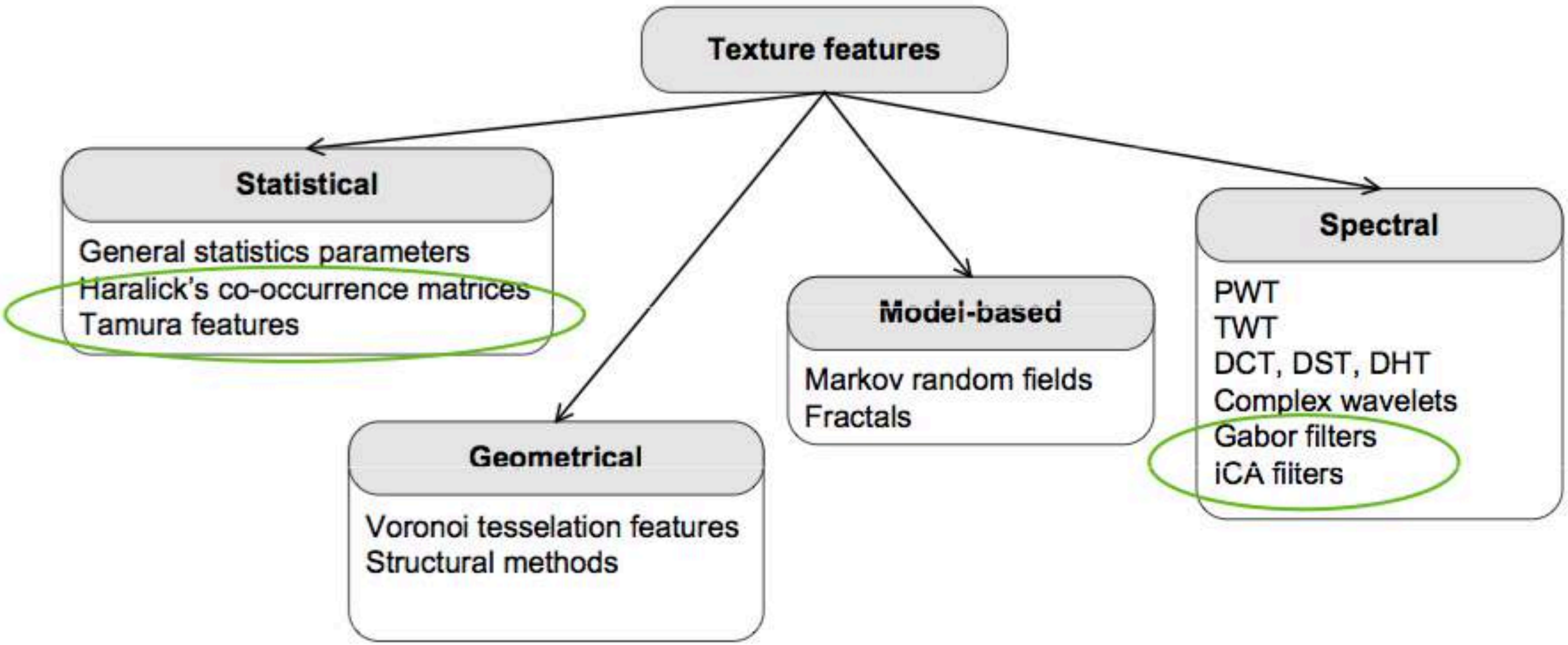
Metrics: $\sim L_1$

Color features

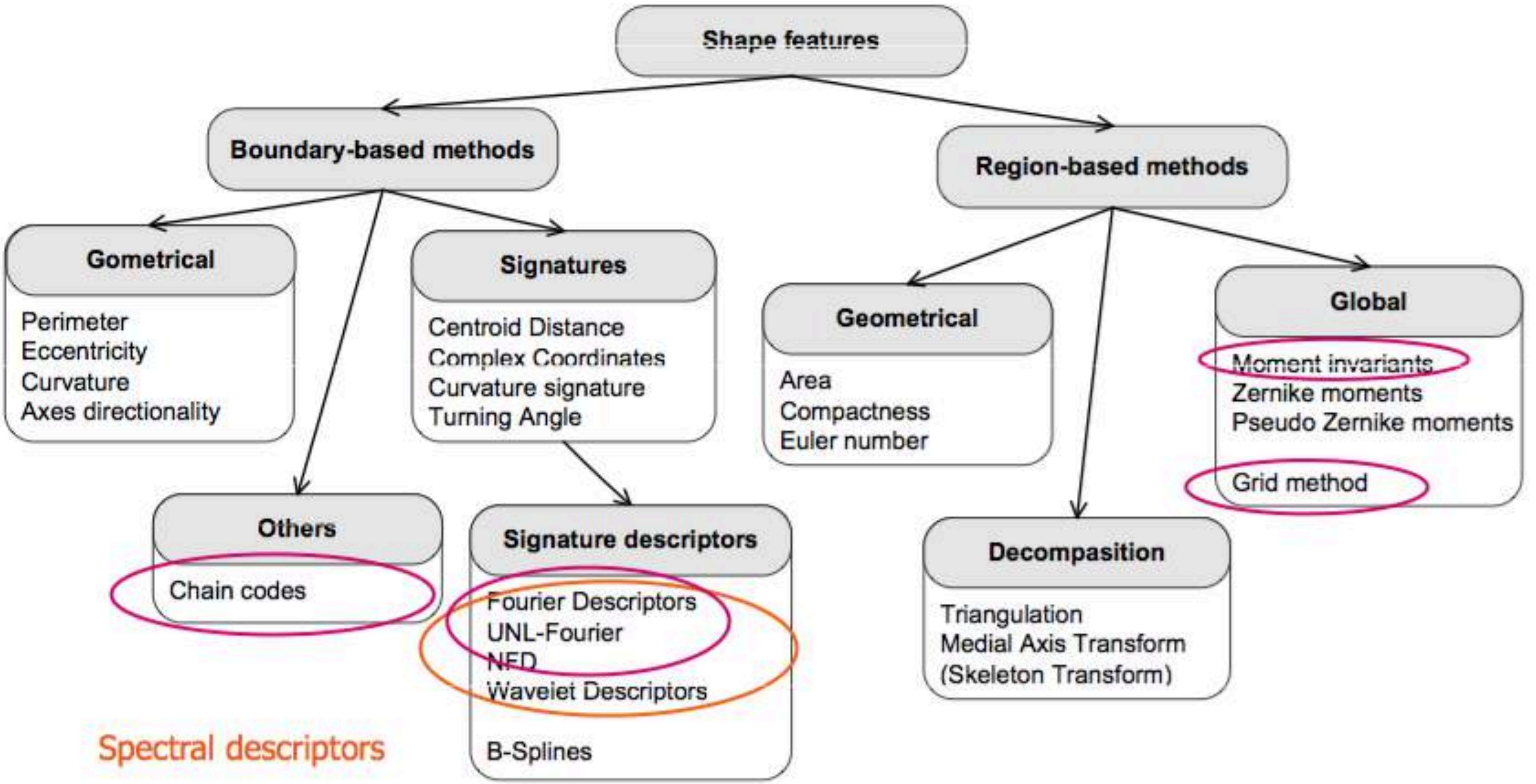


- Disadvantage of histogram: spatial color layout is not considered.
- On heterogeneous collections moments are slightly better.
- Fusion of histograms and moments can lead to better results

Texture features

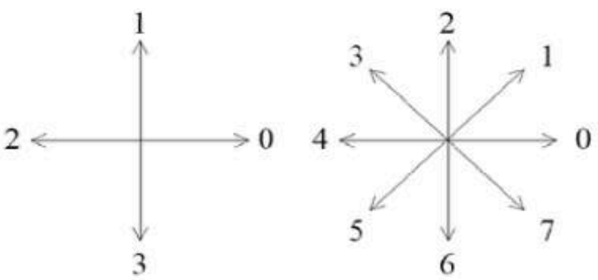


Shape features



Chain codes

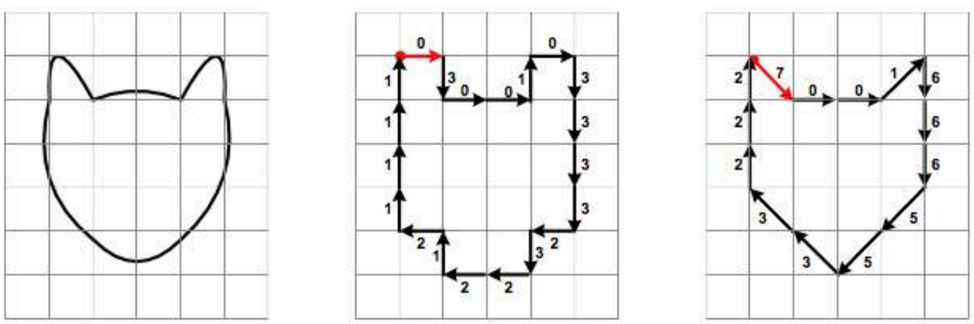
Directions for 4-connected and 8-connected chain codes:



A: 03001033332322121111

B: 70016665533222

Example:



A

B

Starting point invariance: minimal code

70016665533222 -> 00166655332227

Rotation invariance: codes subtraction

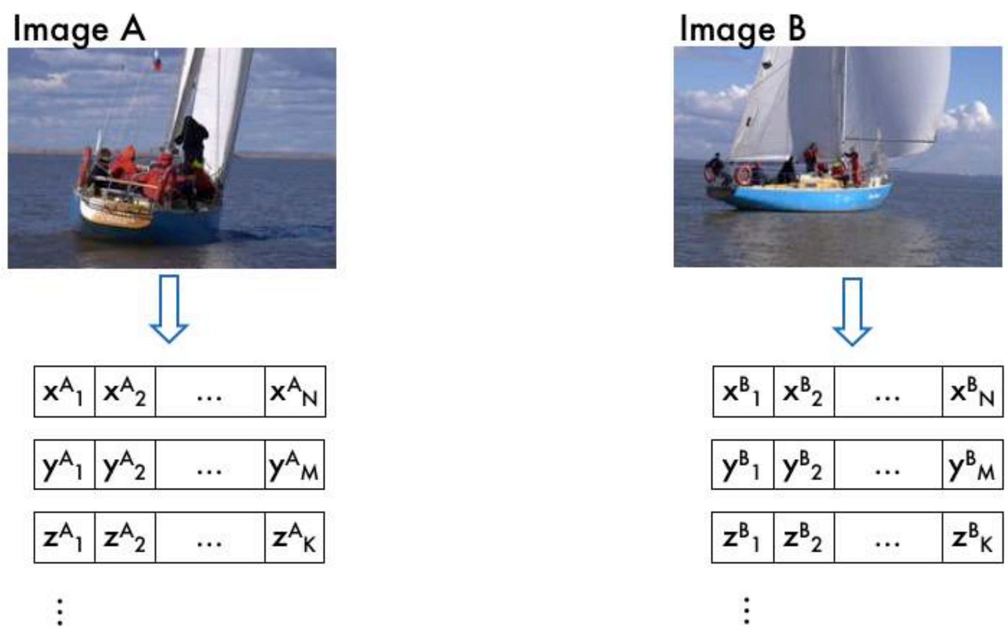
00166655332227 -> 01500706070051

Local Descriptors

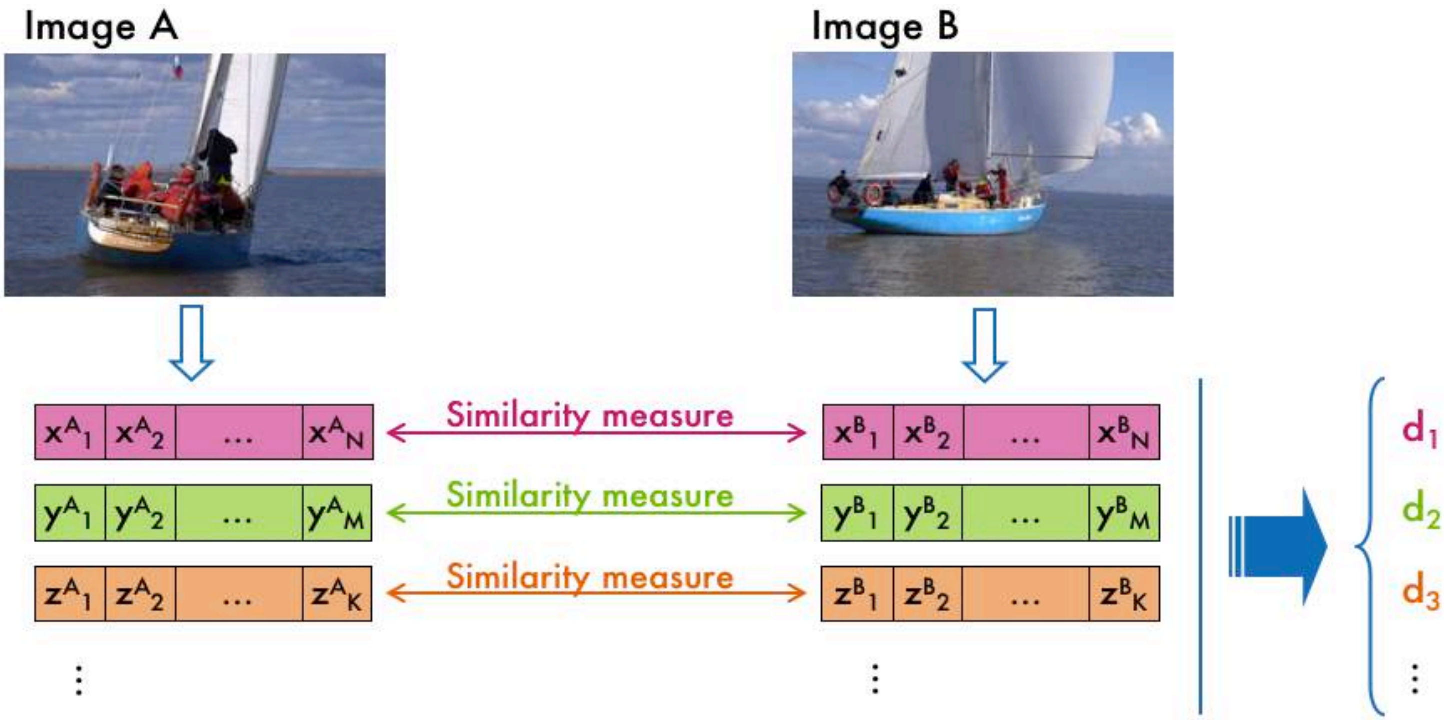
- ✓ Features for local regions in the image
  - ✓ Regions obtained by segmentation
  - ✓ Regions of interest (RoI) – around interest points (keypoints)
- ✓ Interest points: corners, edges and others
- ✓ Keypoints: points invariant to image translation, scale and rotation, and minimally affected by noise and small distortions

Feature Spaces

- **Feature vector** – a vector of features, representing one image.
- **Feature space** – the set of all possible feature vectors with defined similarity measure.

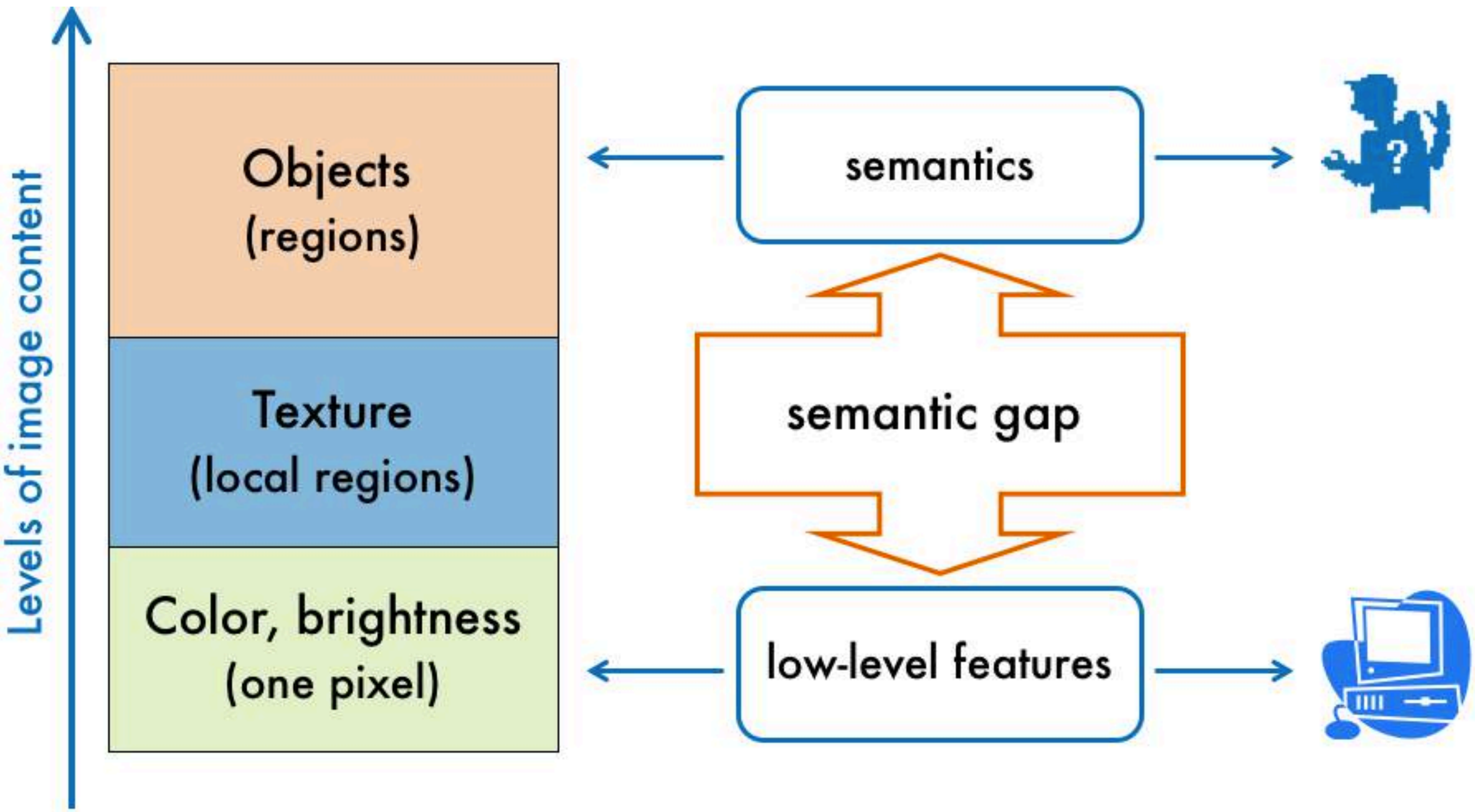


How to compare?



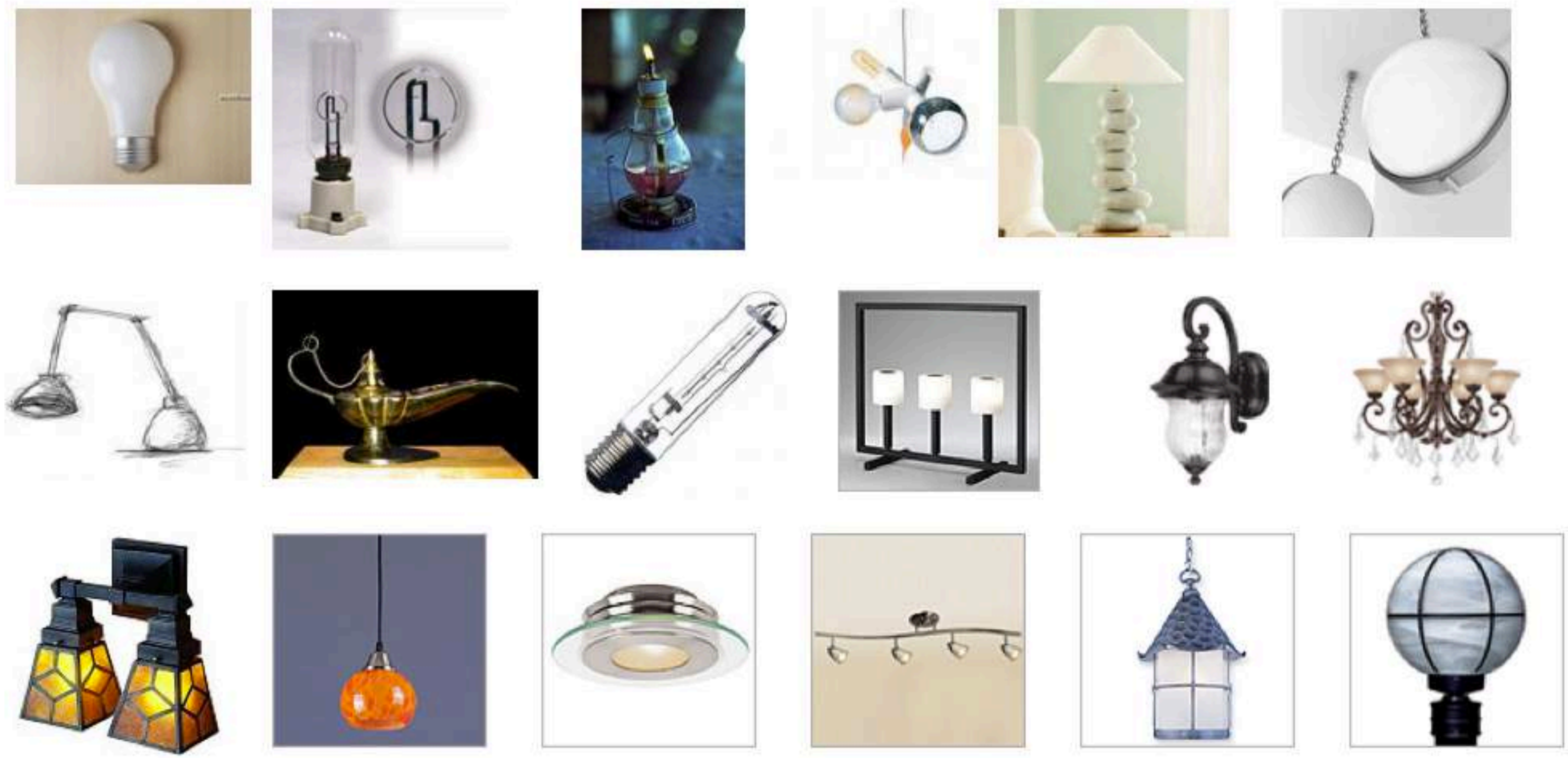
$$D = \sum_i c_i d_i$$

Problems: semantic gap



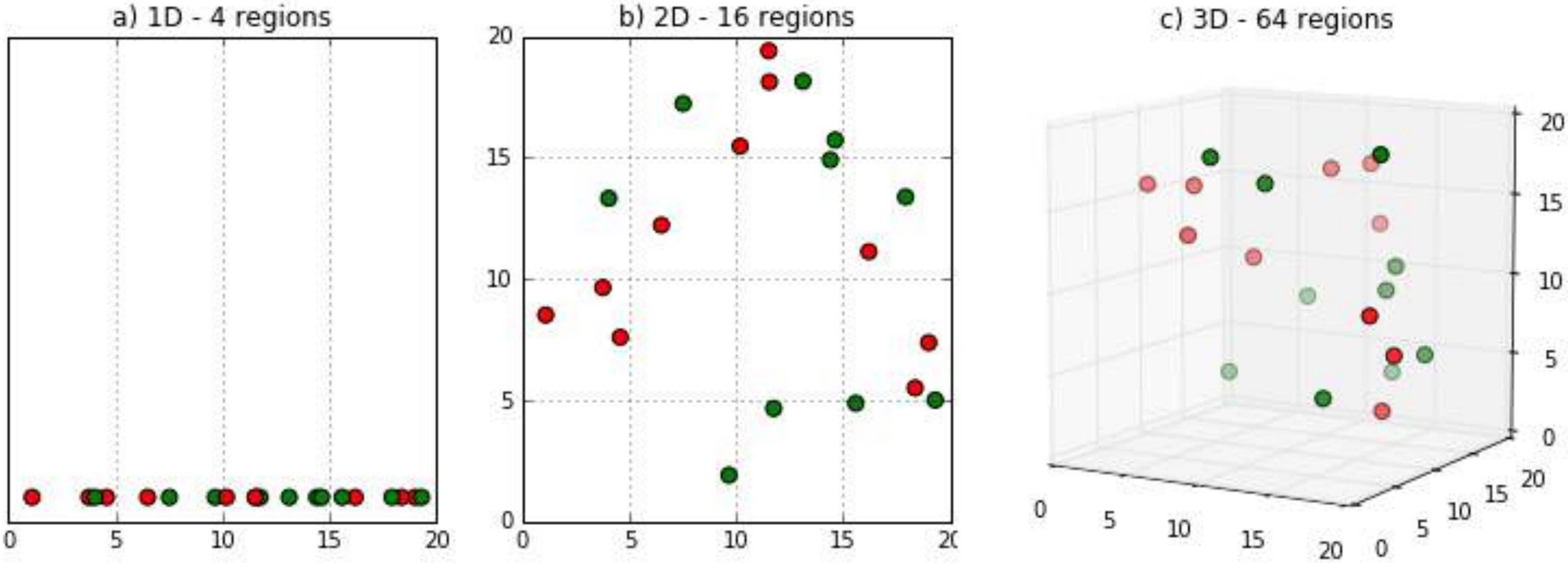
How to understand what is on the images?

Problems: semantic gap



How do we know that all these objects are lamps?

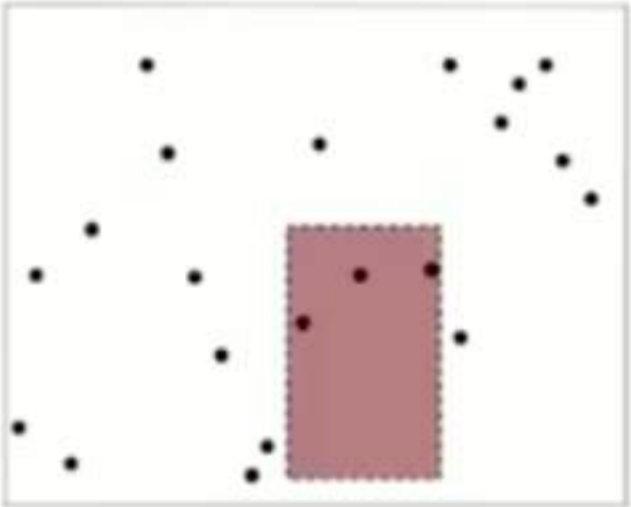
Problems: curse of dimensionality



Indexing techniques

2D Orthogonal Range Search

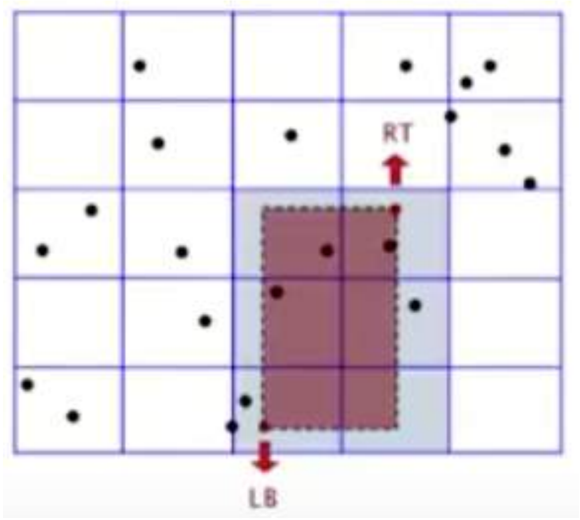
- ✓ Search for a 2D key.
- ✓ Range search: find all keys that lie in a 2D range.
- ✓ Range count: number of keys that lie in a 2D range.



2D Orthogonal Range Search

Grid implementation:

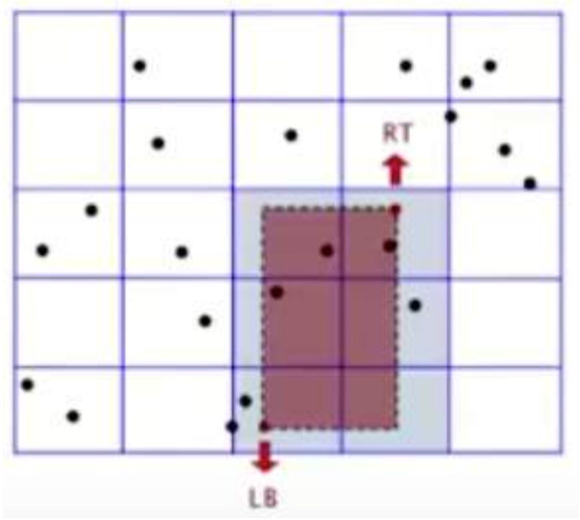
- ✓ Divide space into M-by-M grid of squares.
- ✓ Create list of points contained in each square.
- ✓ Range search: examine only squares that intersect 2D range query.



2D Orthogonal Range Search

Choose grid square size to tune performance.

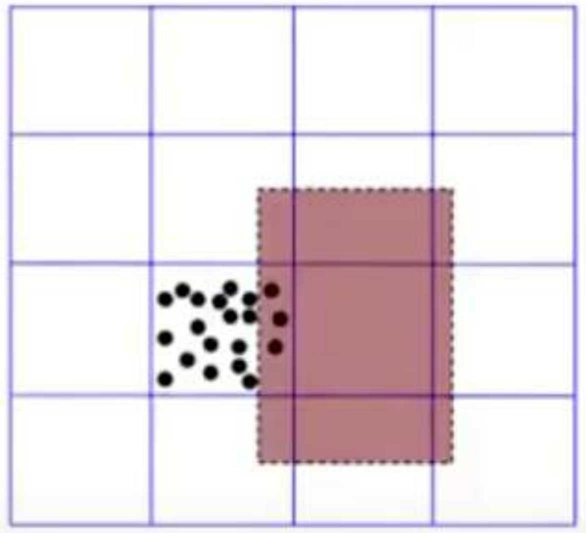
- ✓ Too small: wastes space
- ✓ Too large: too many points per square

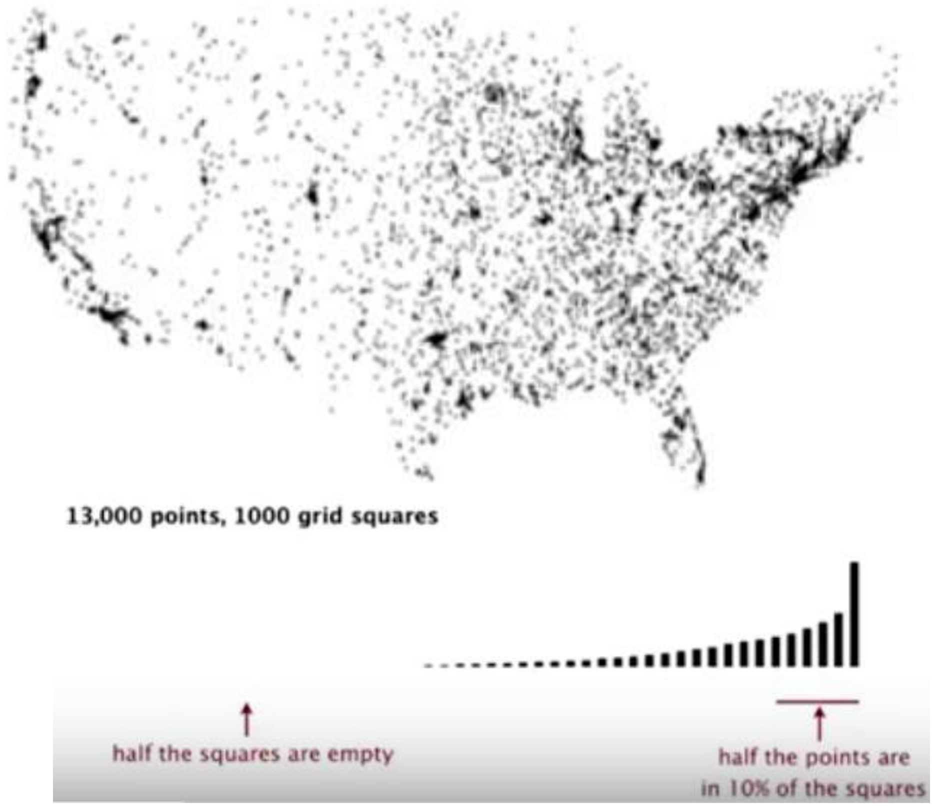


2D Orthogonal Range Search

Problem: clustering, a well-known phenomenon in geometric data.

Need a method that adapts gracefully to data.





Space-partitioning trees

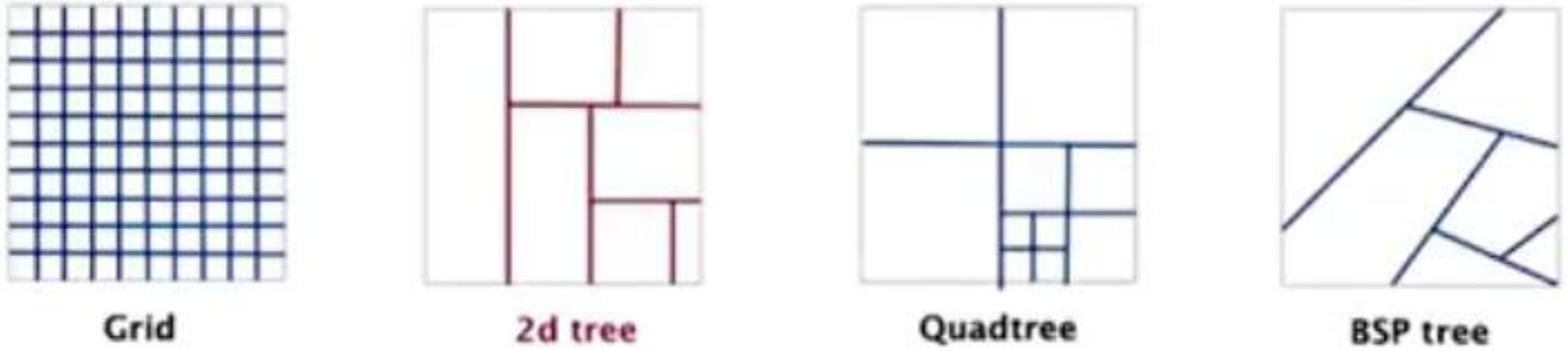
Use a tree to represent a recursive subdivision of 2D space.

Grid. Divide space uniformly into squares.

2D tree. Recursively divide space into two halfplanes.

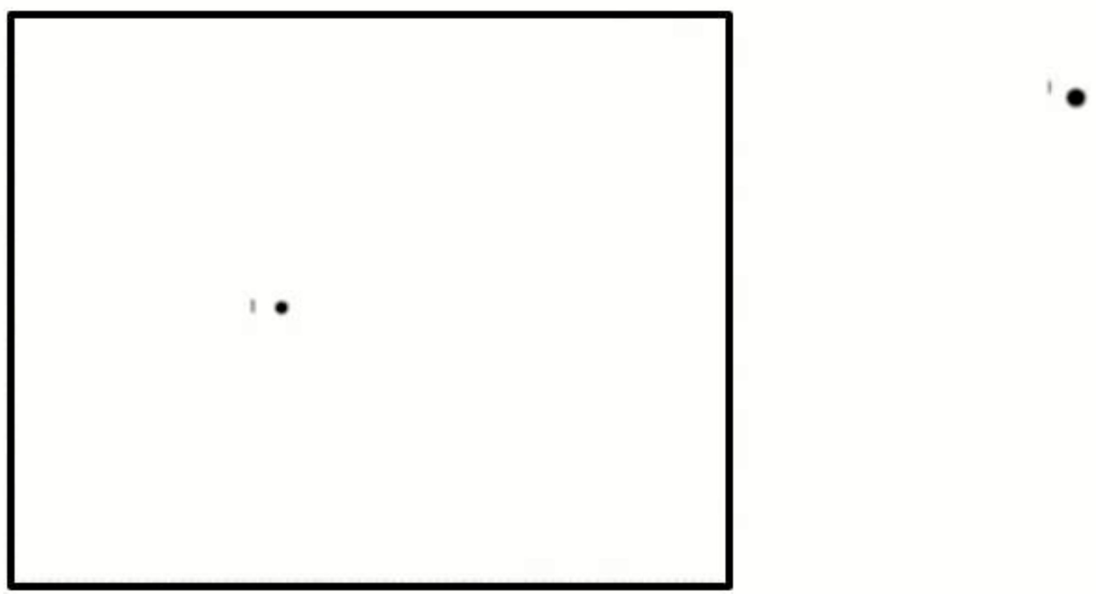
Quadtree. Recursively divide space into four quadrants.

BSP tree. Recursively divide space into two regions.



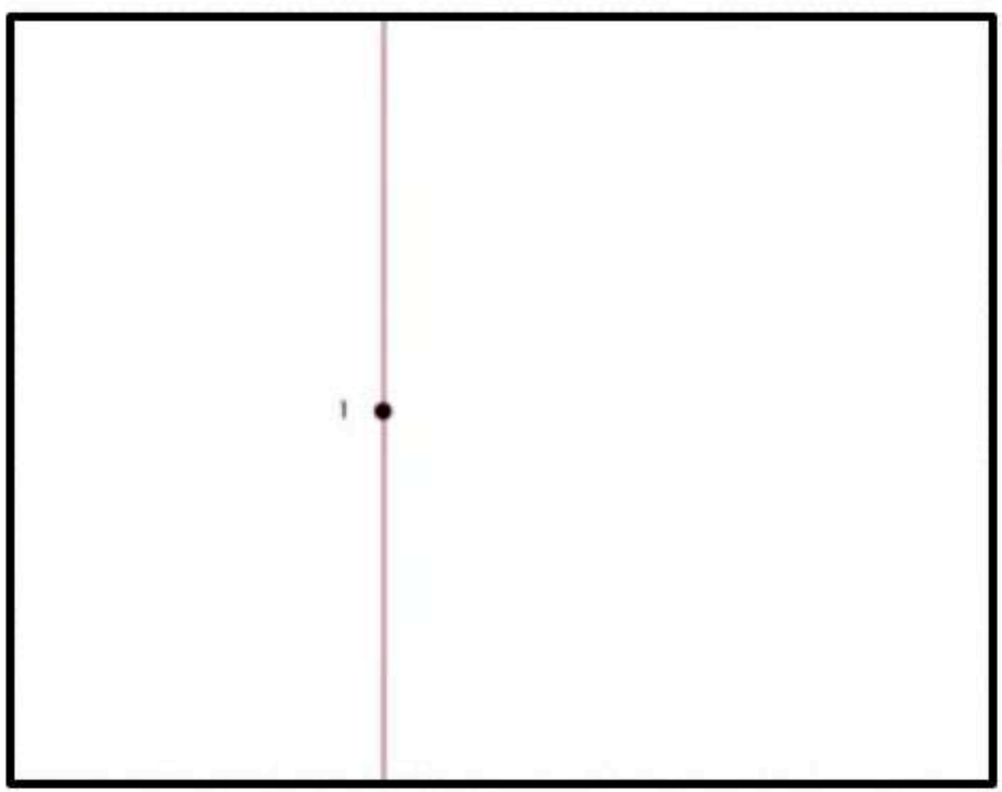
2d-tree construction

Recursively partition plane into two half-planes



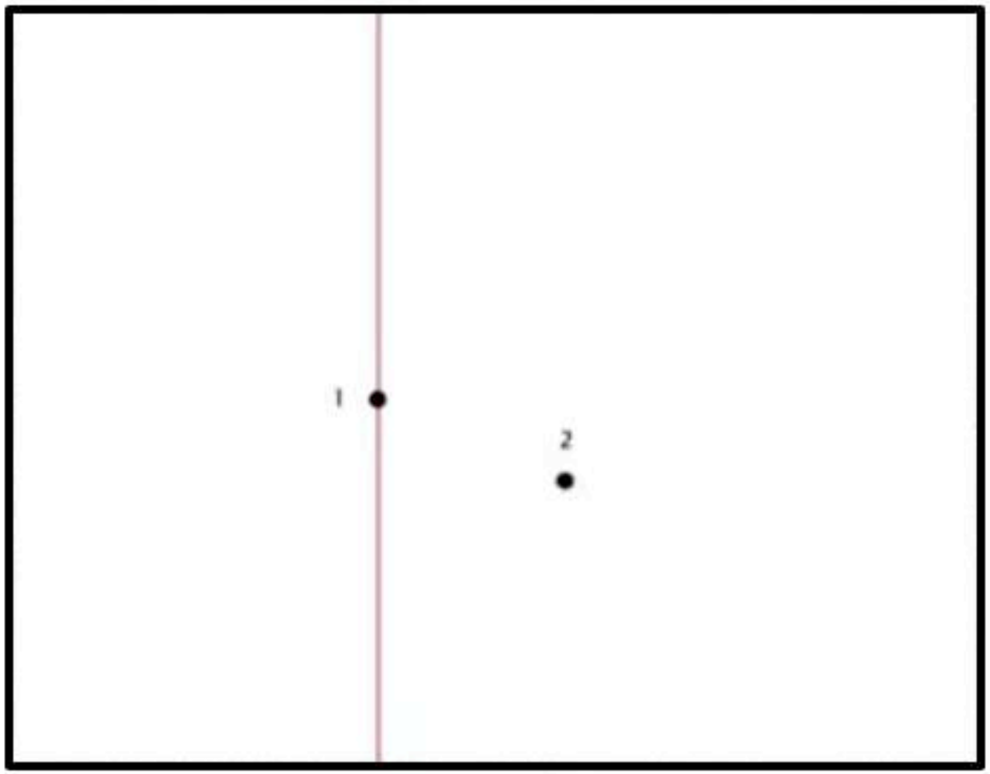
2d-tree construction

Recursively partition plane into two half-planes



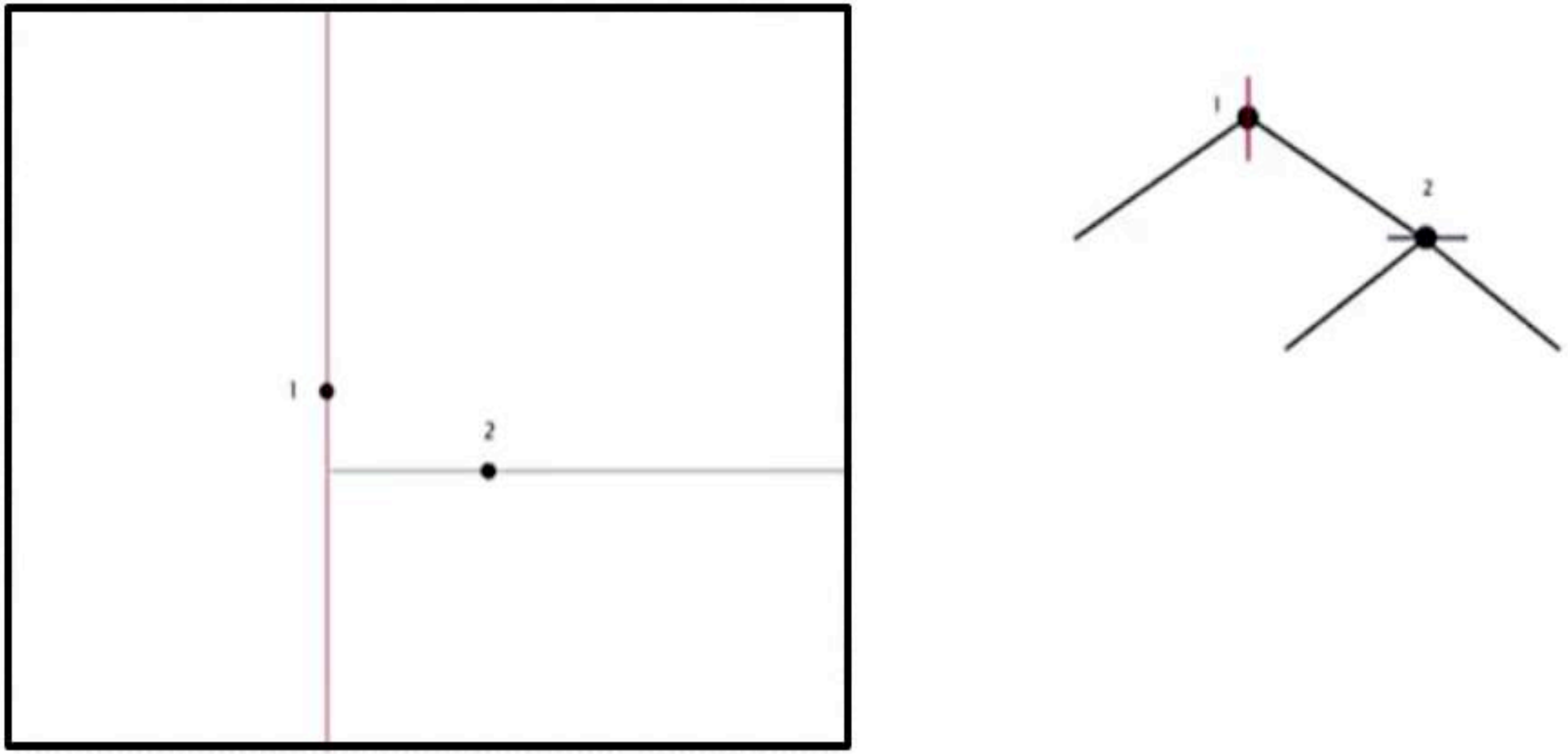
2d-tree construction

Recursively partition plane into two half-planes



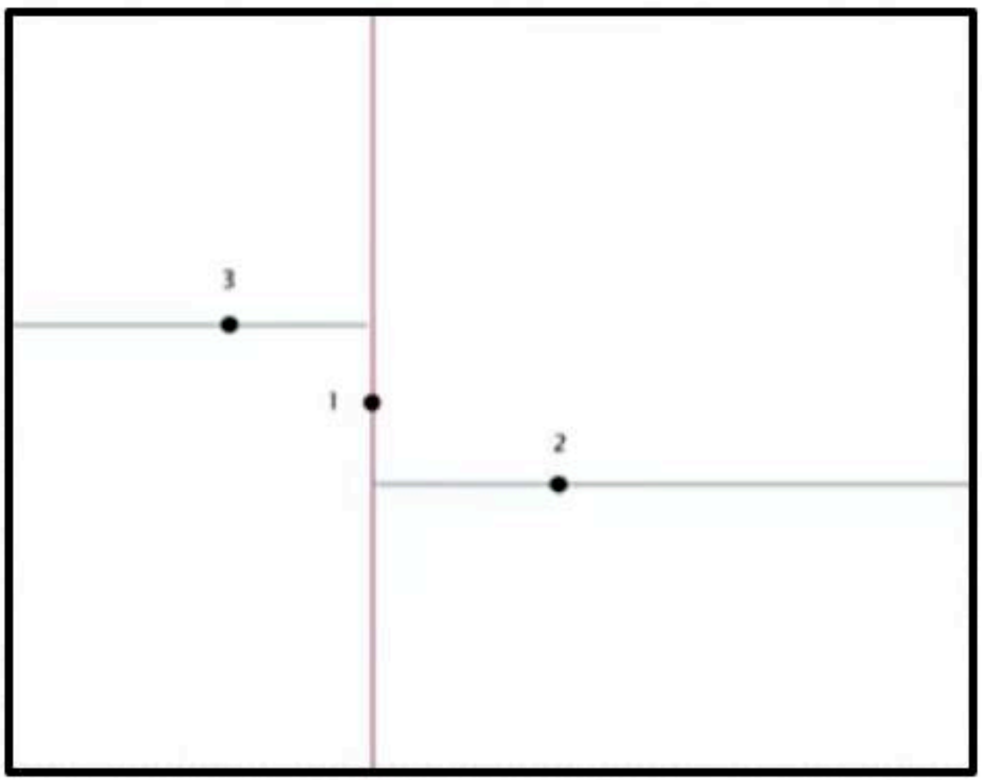
2d-tree construction

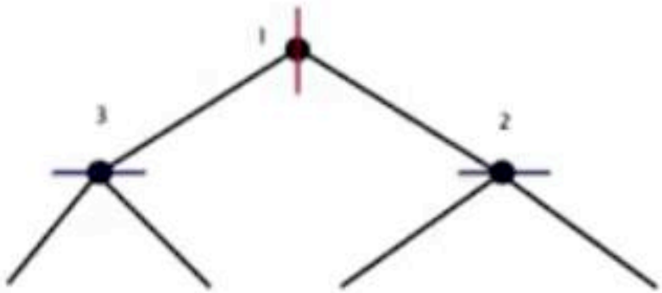
Recursively partition plane into two half-planes



2d-tree construction

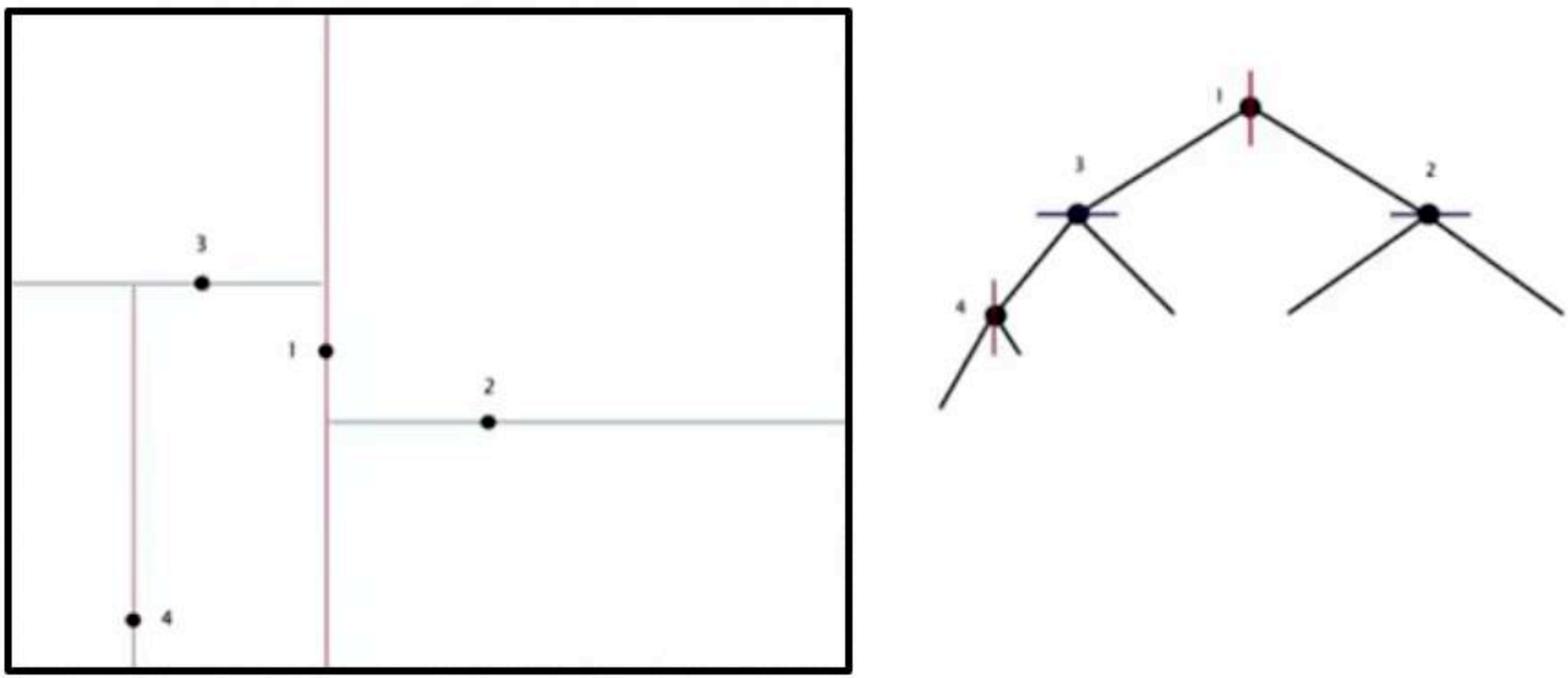
Recursively partition plane into two half-planes





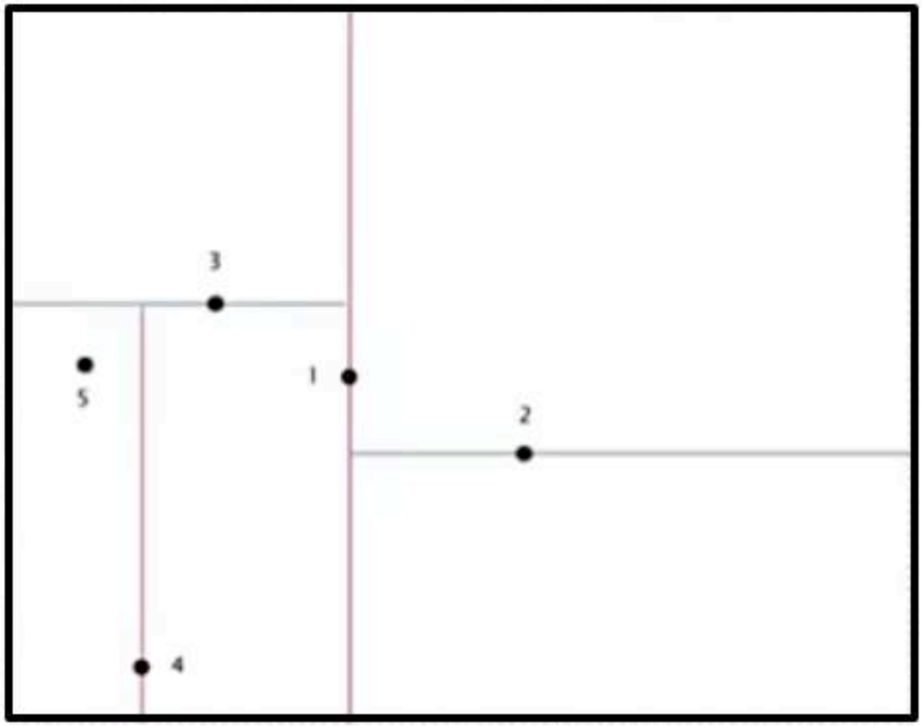
2d-tree construction

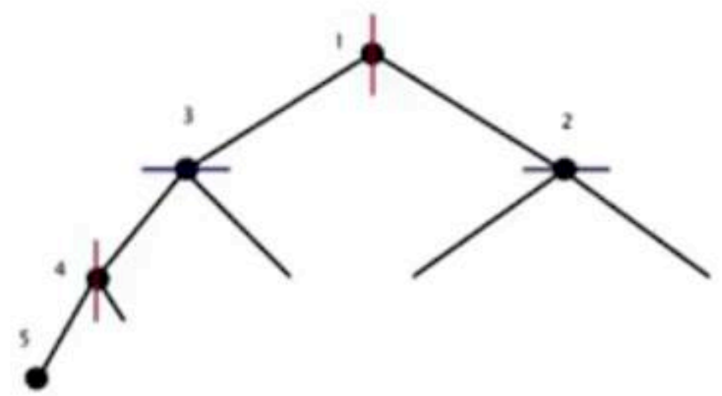
Recursively partition plane into two half-planes



2d-tree construction

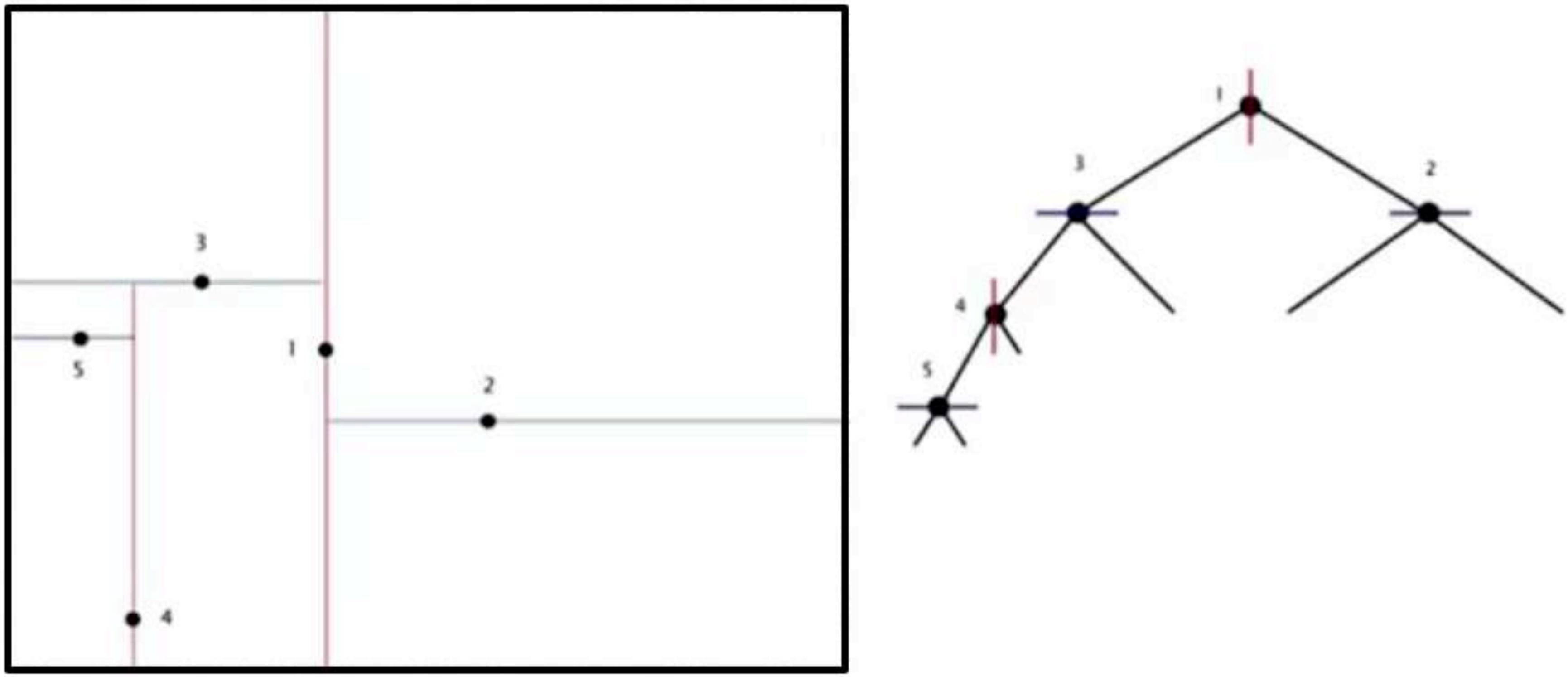
Recursively partition plane into two half-planes





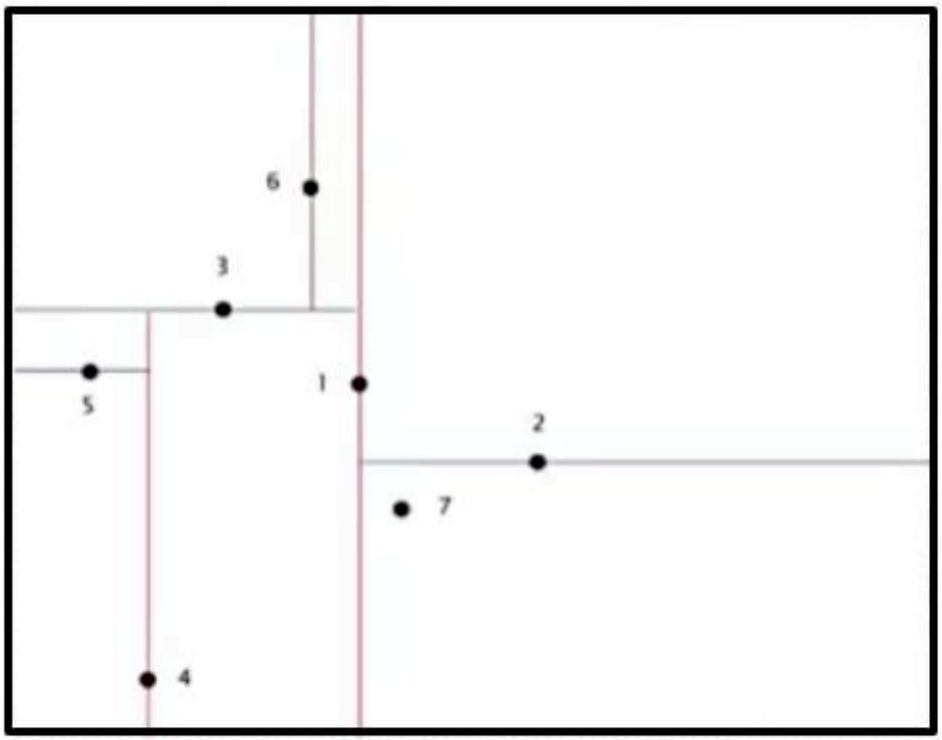
2d-tree construction

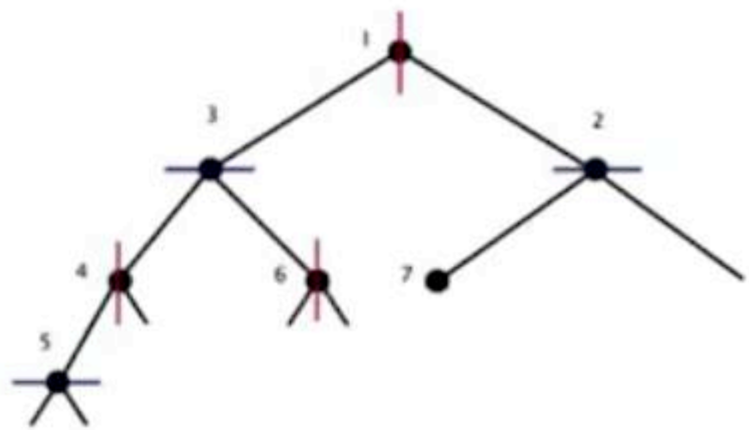
Recursively partition plane into two half-planes



2d-tree construction

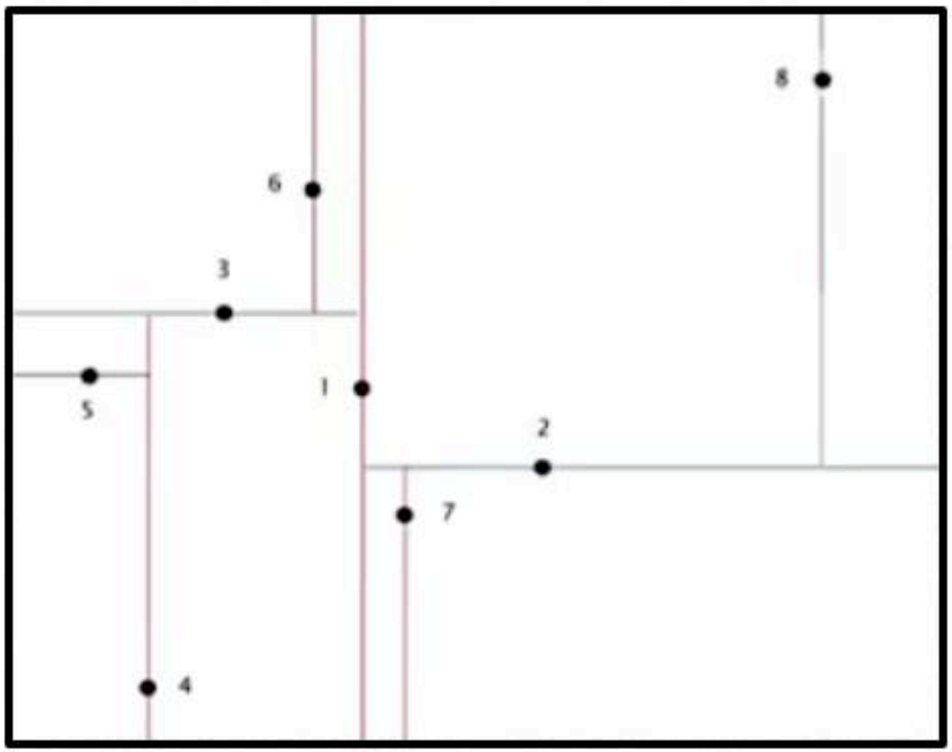
Recursively partition plane into two half-planes

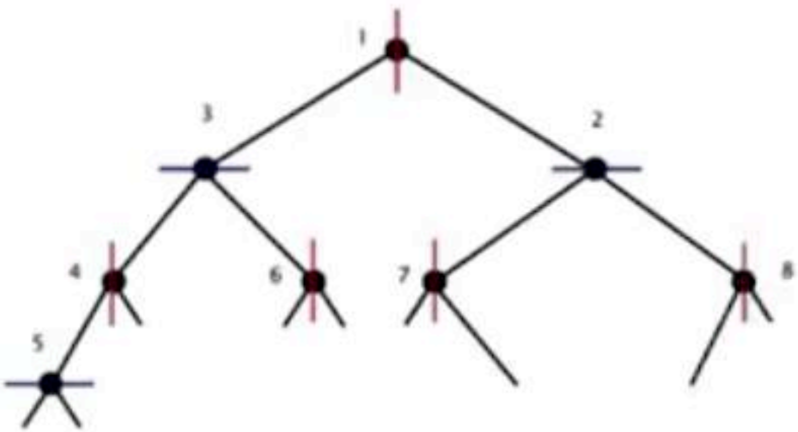




2d-tree construction

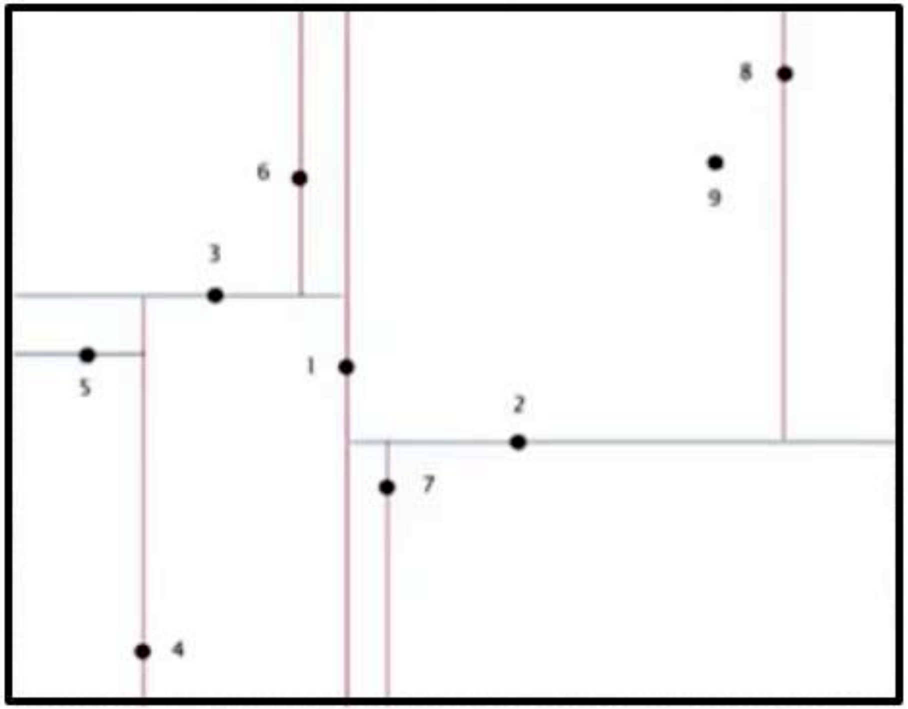
Recursively partition plane into two half-planes

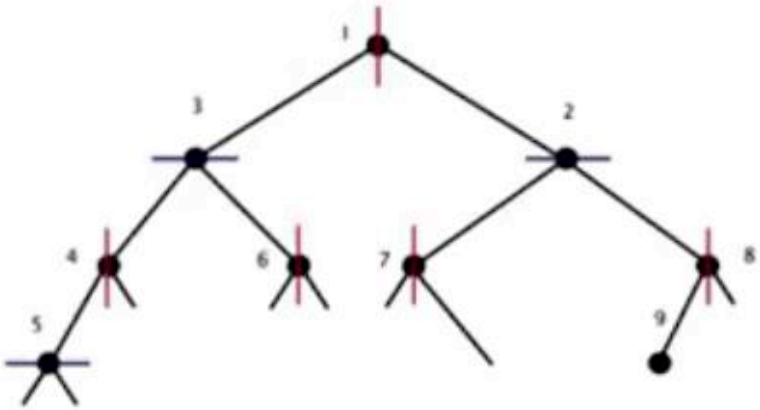




2d-tree construction

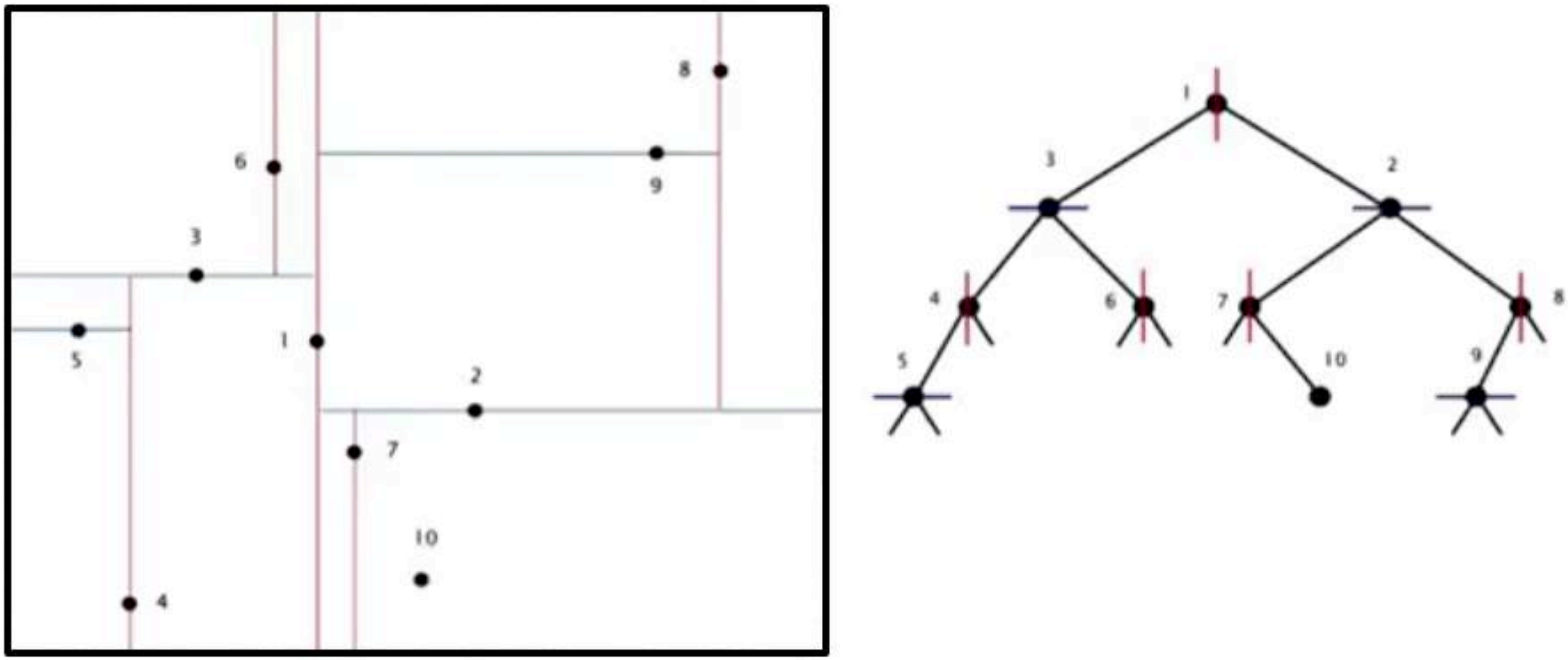
Recursively partition plane into two half-planes





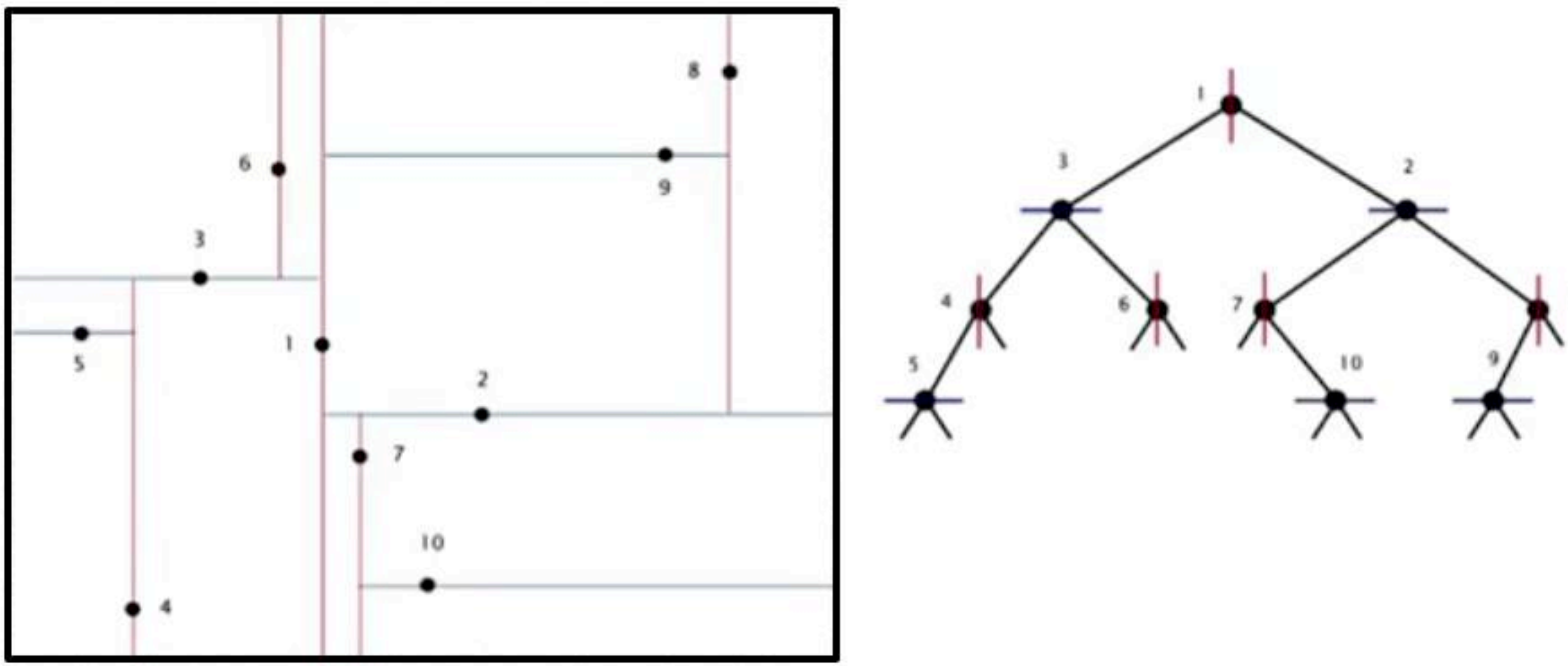
2d-tree construction

Recursively partition plane into two half-planes



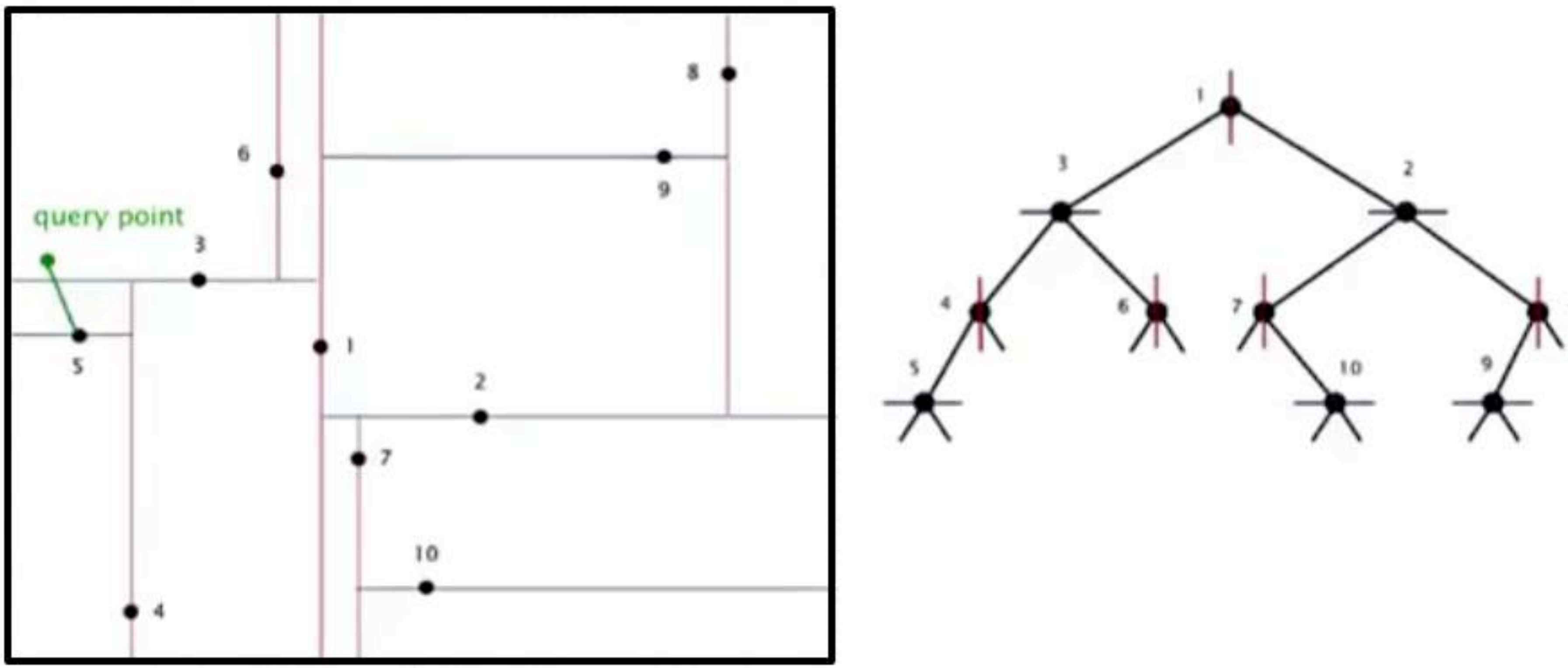
2d-tree construction

Recursively partition plane into two half-planes



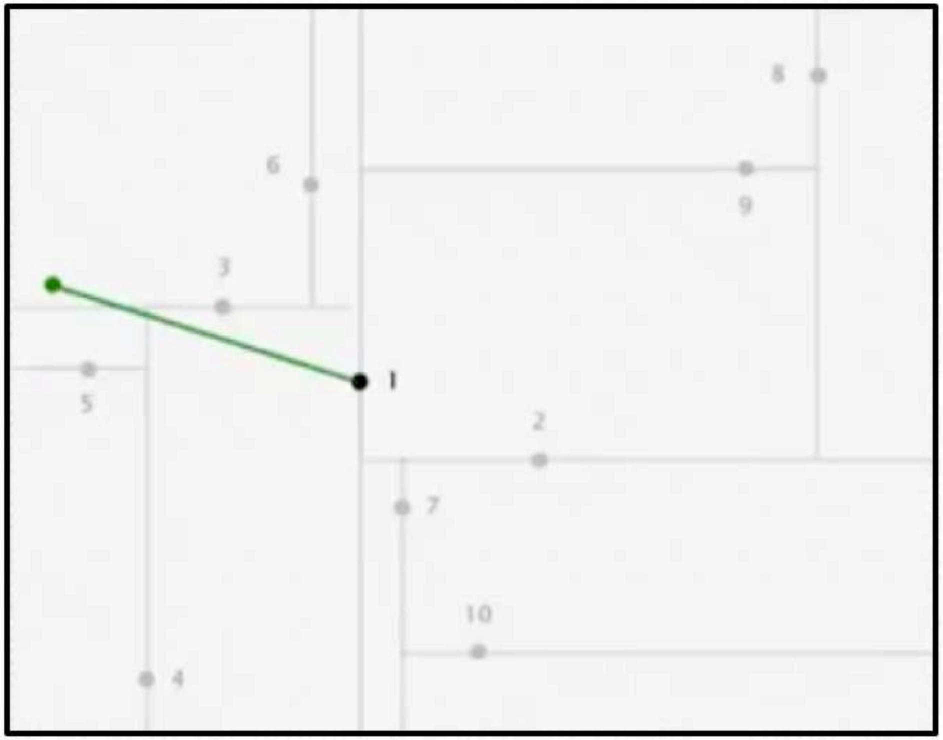
Nearest Neighbor Search in a 2d tree

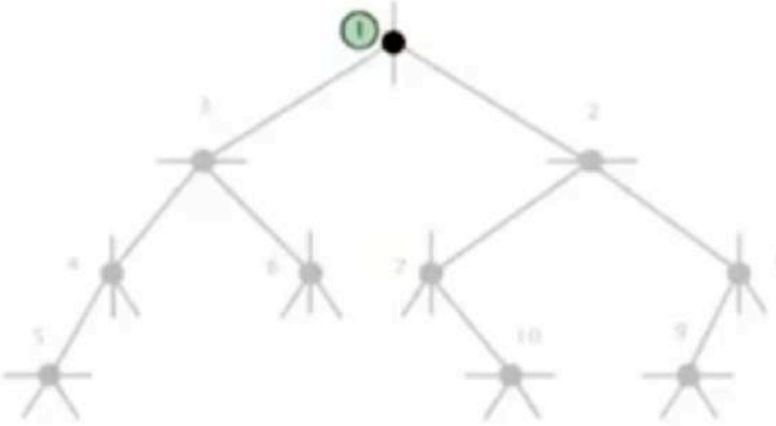
- ✓ Goal: find closest point to query point



Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

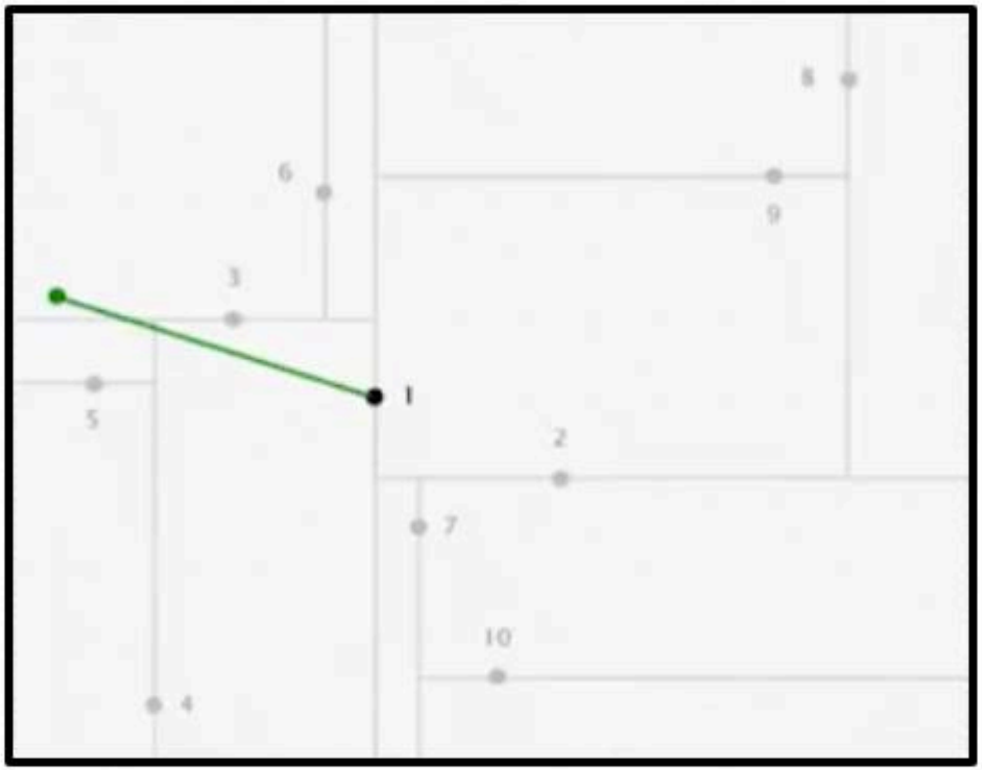


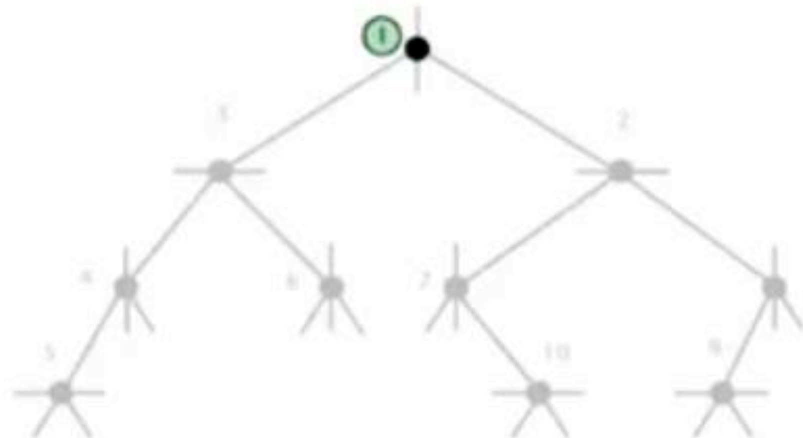


search root node
compute distance from query point to 1
(update champion nearest neighbor)

Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

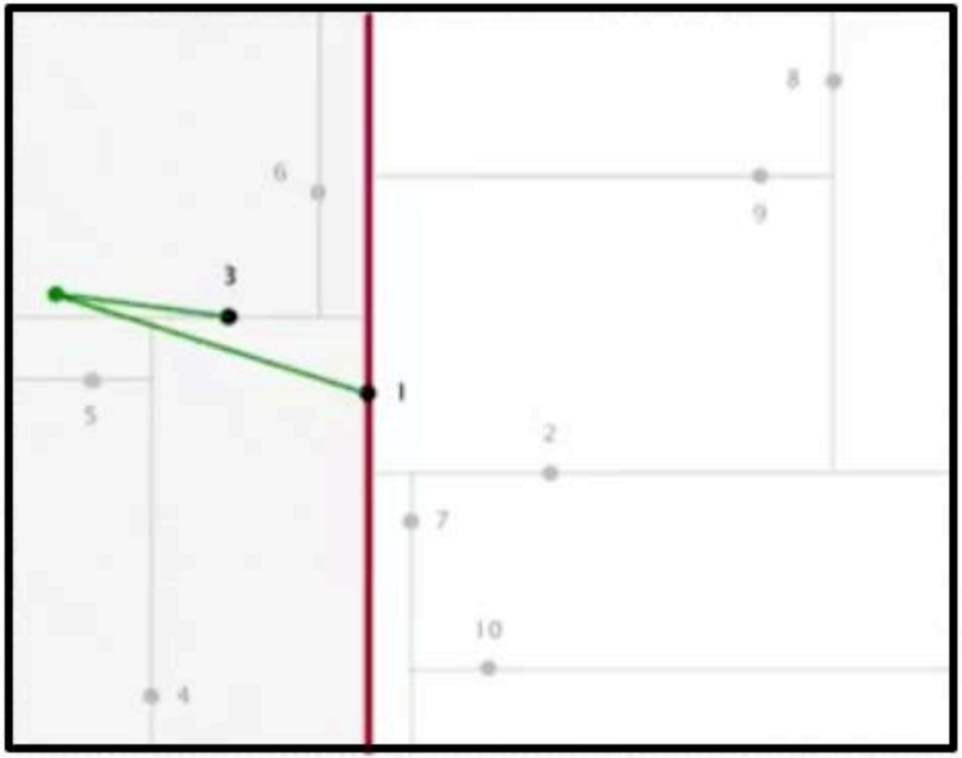


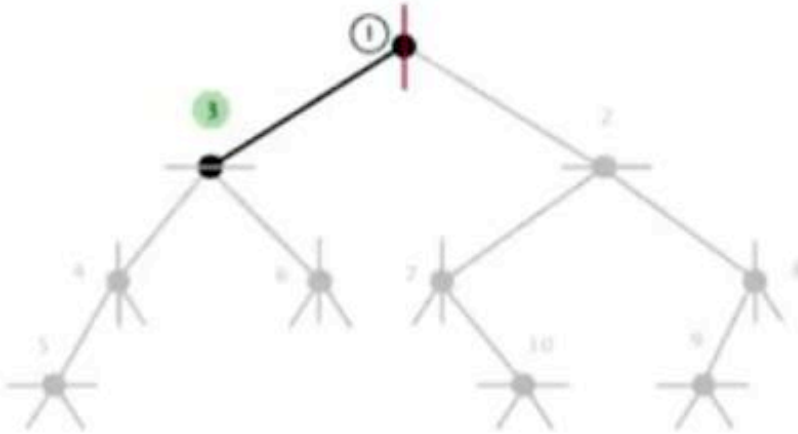


query point is to the left of splitting line
search left subtree first

Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

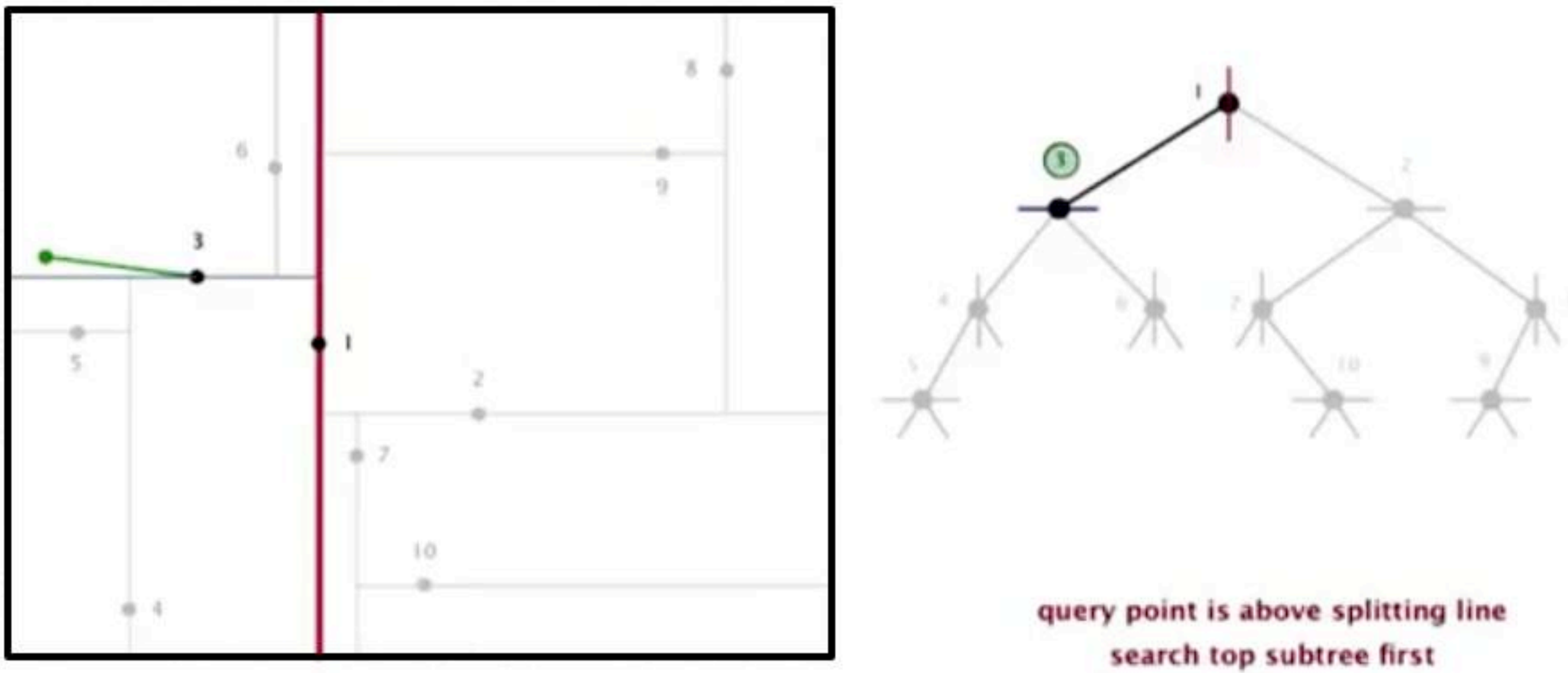




search left subtree
compute distance from query point to 3
(update champion)

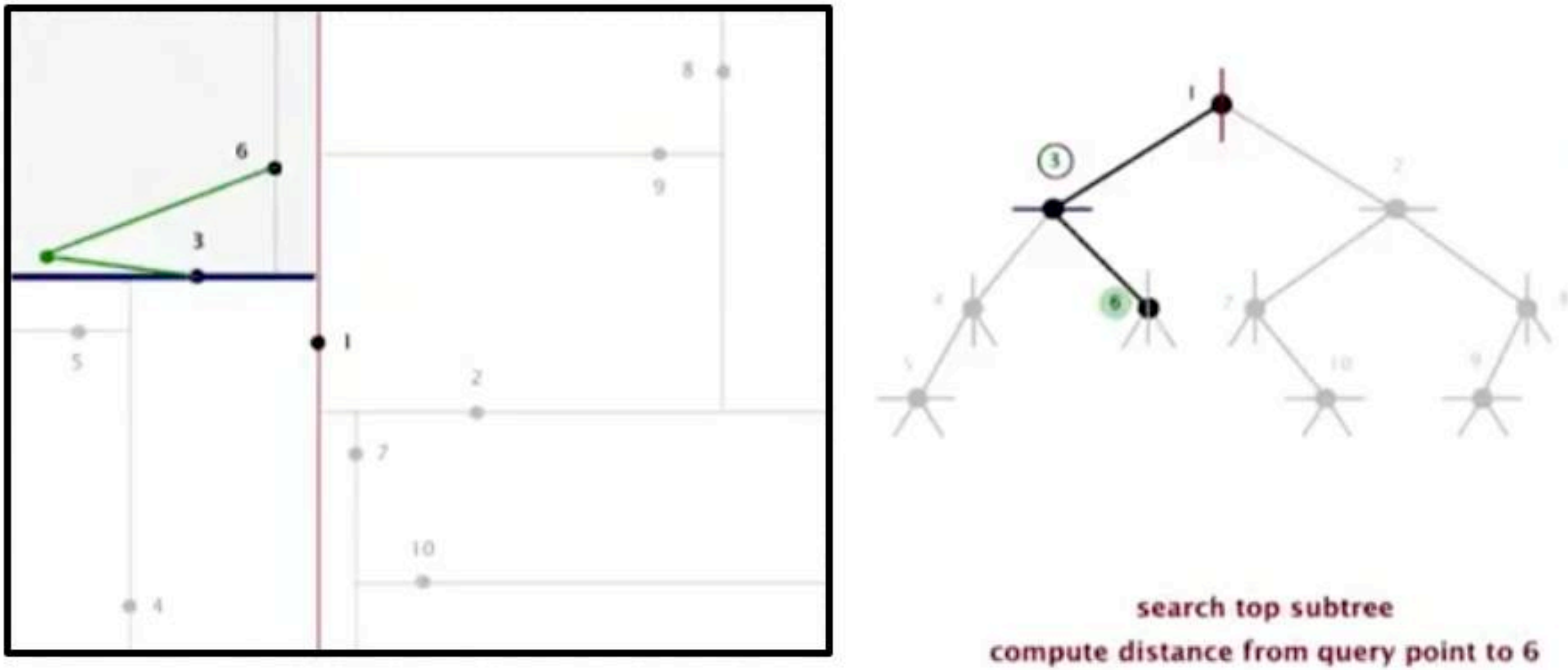
Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



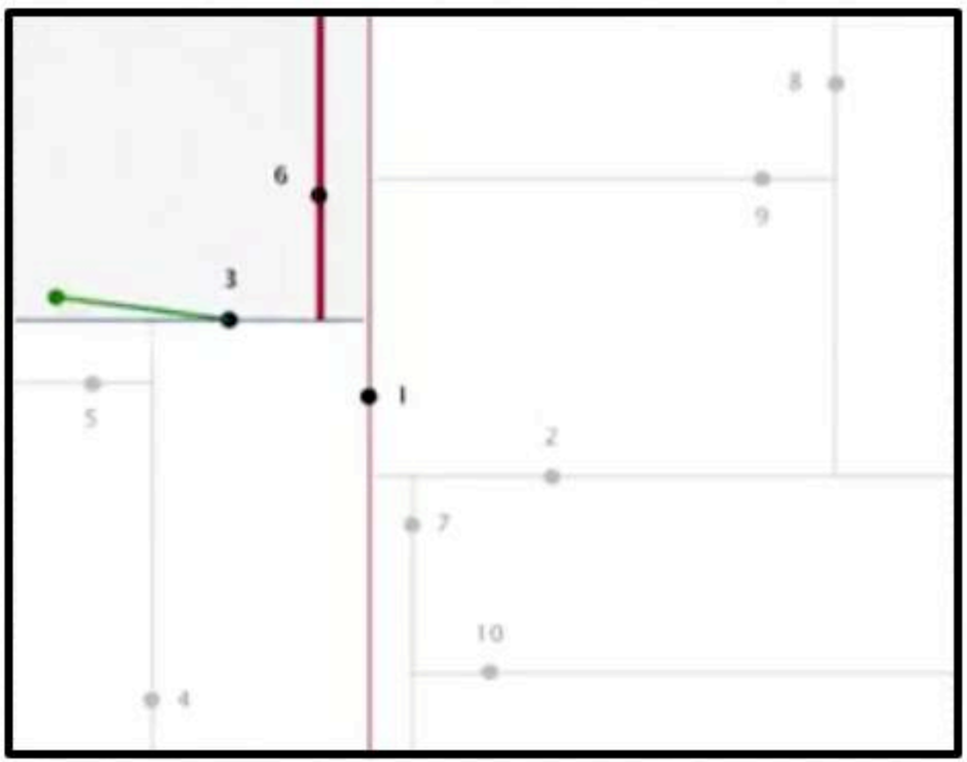
Nearest Neighbor Search in a 2d tree

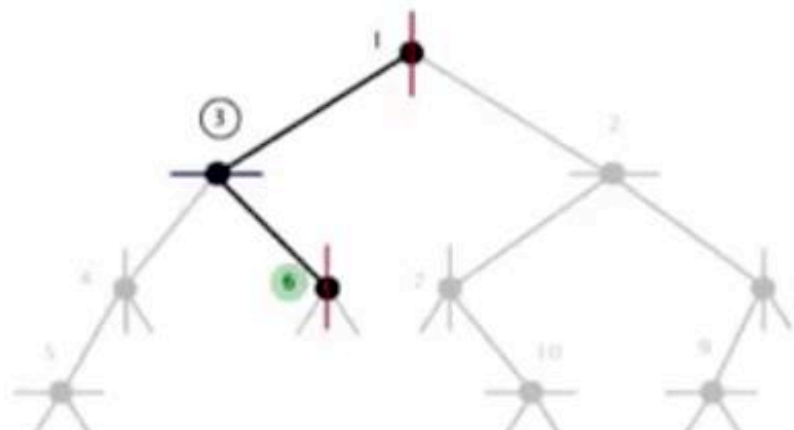
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

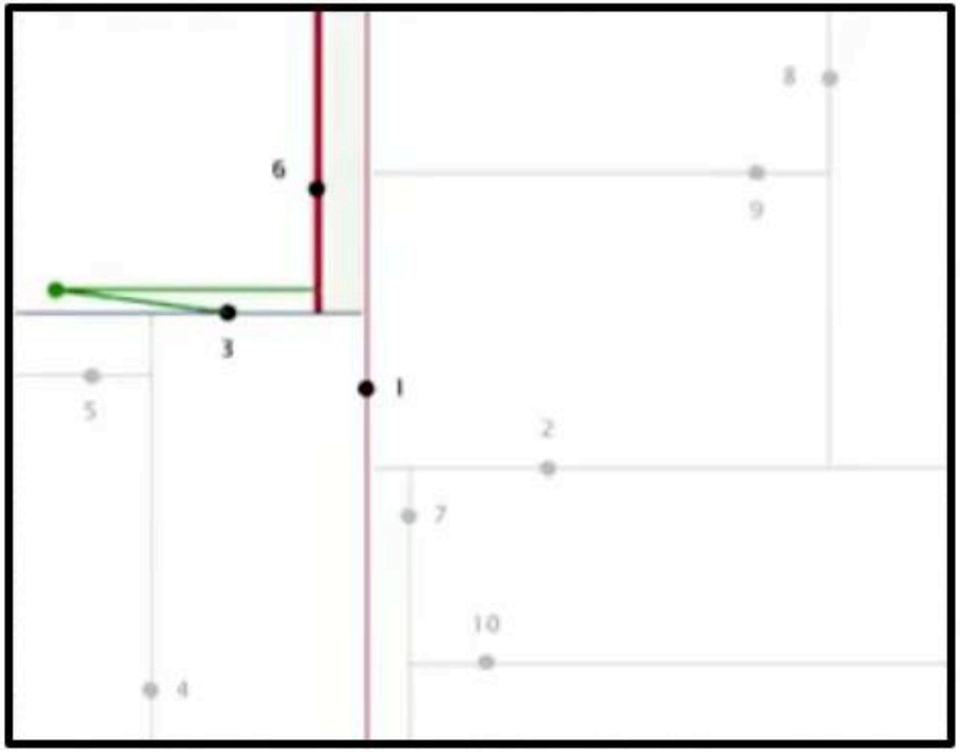


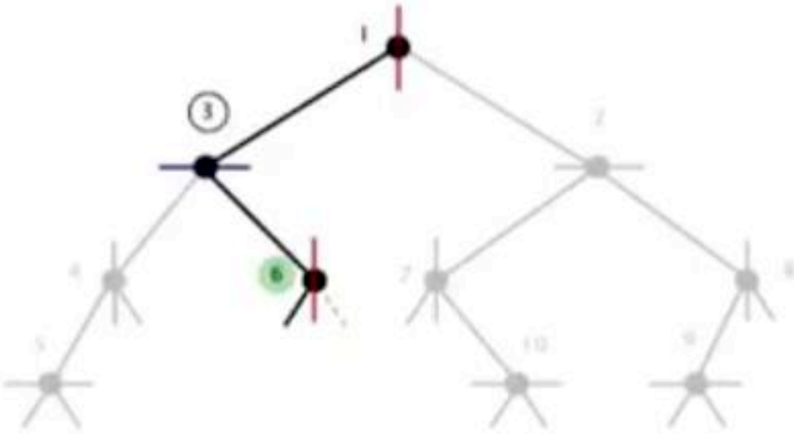


query point is to left of splitting line
search left subtree first

Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

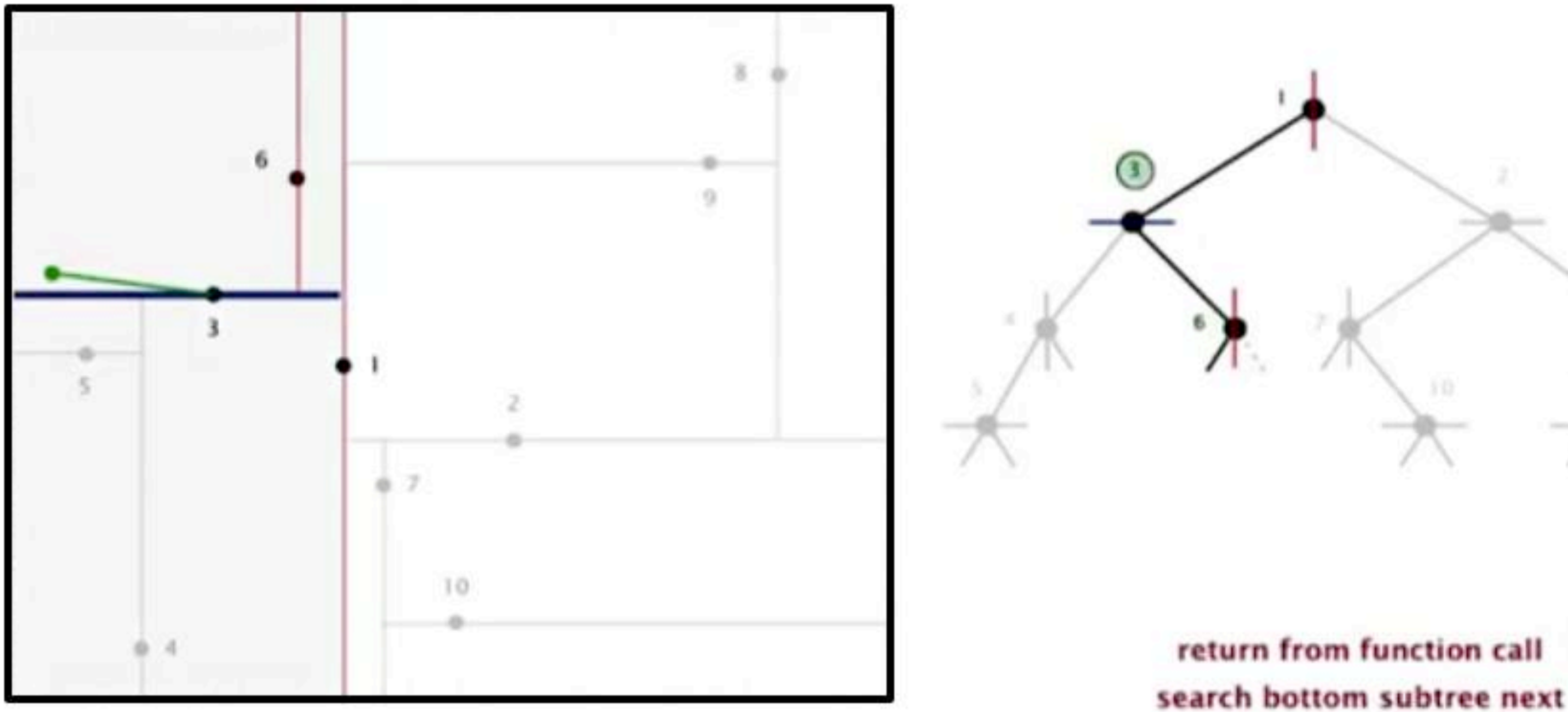




search right subtree
prune since nearest neighbor
can't be in

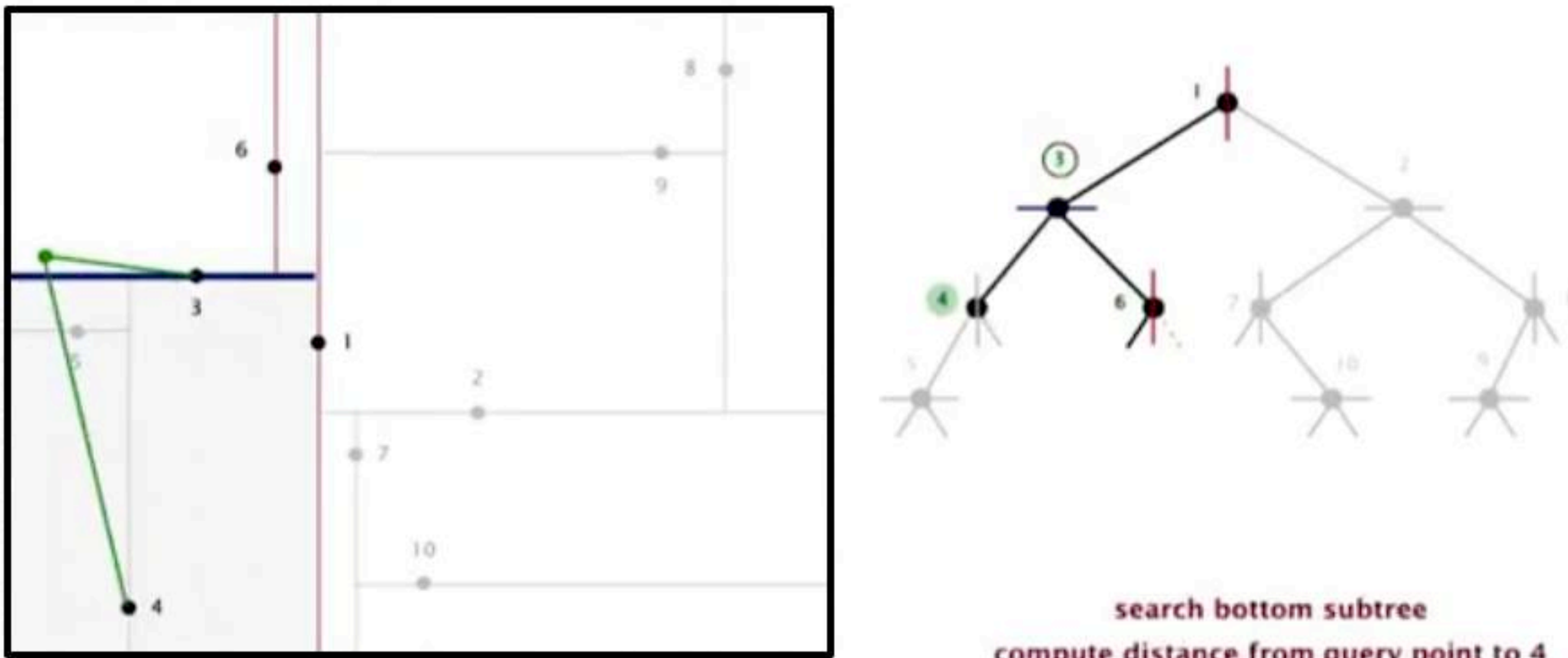
Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



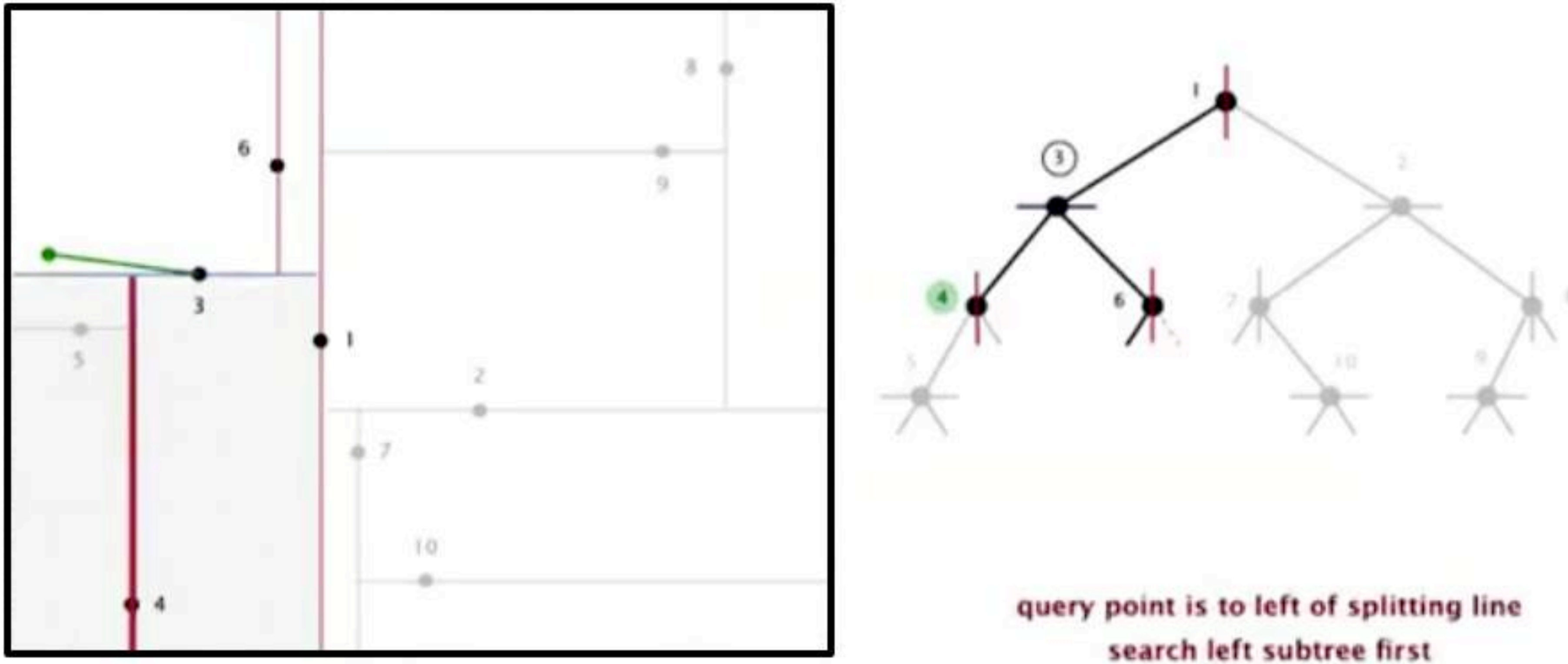
Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



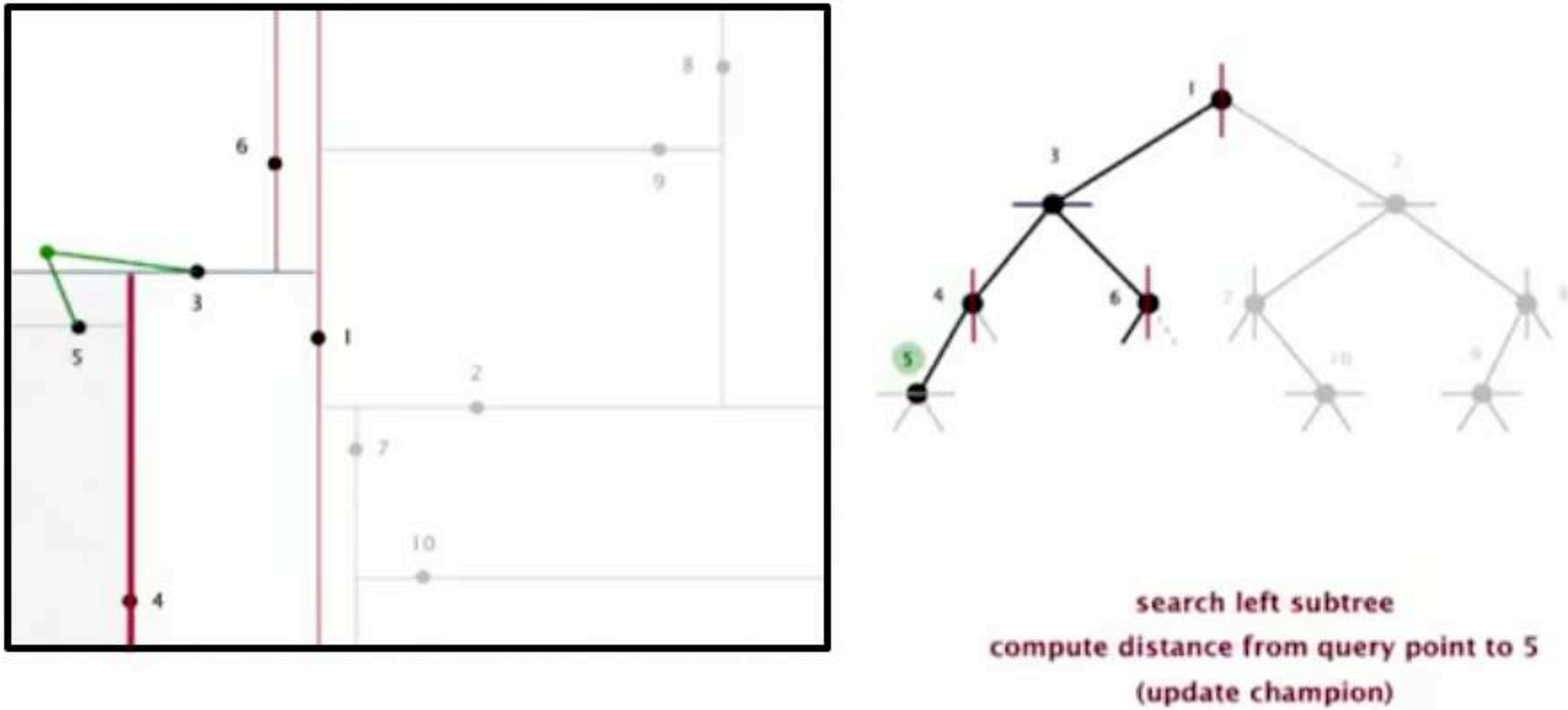
Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



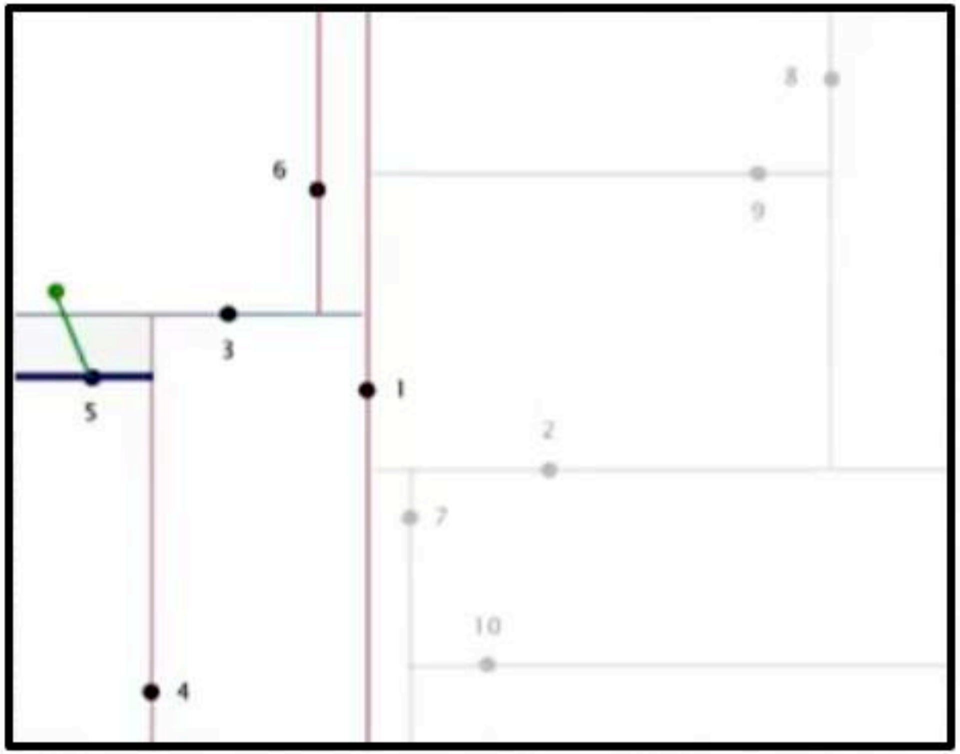
Nearest Neighbor Search in a 2d tree

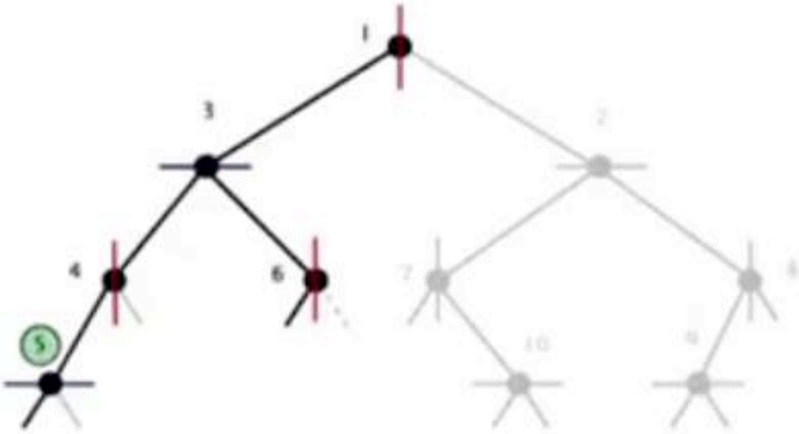
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

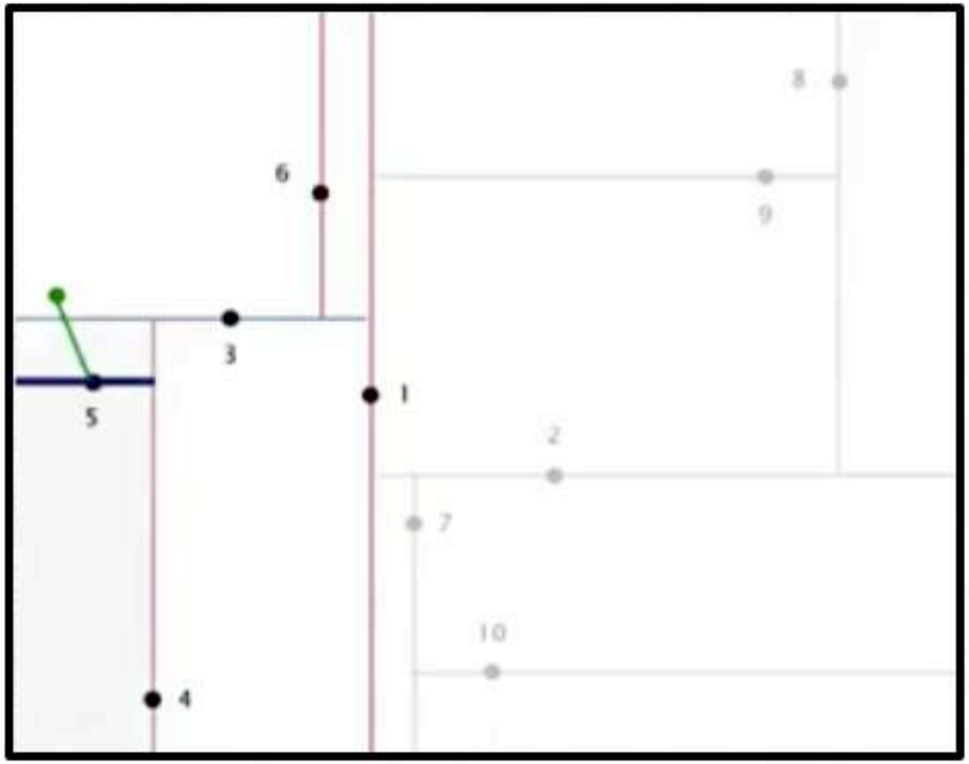


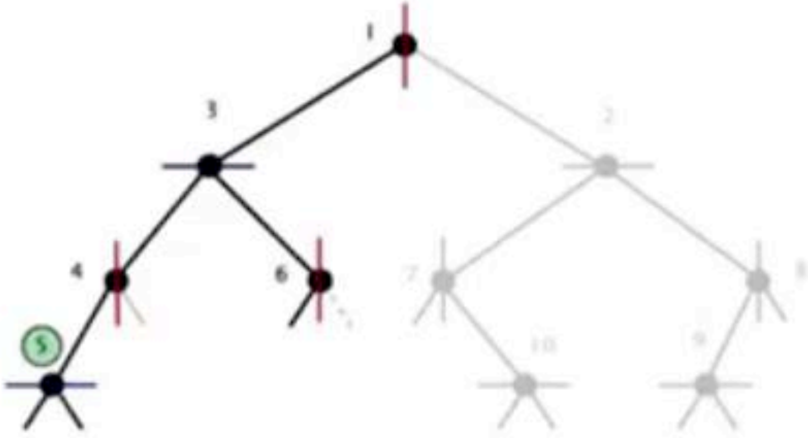


search top subtree
return since empty

Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).

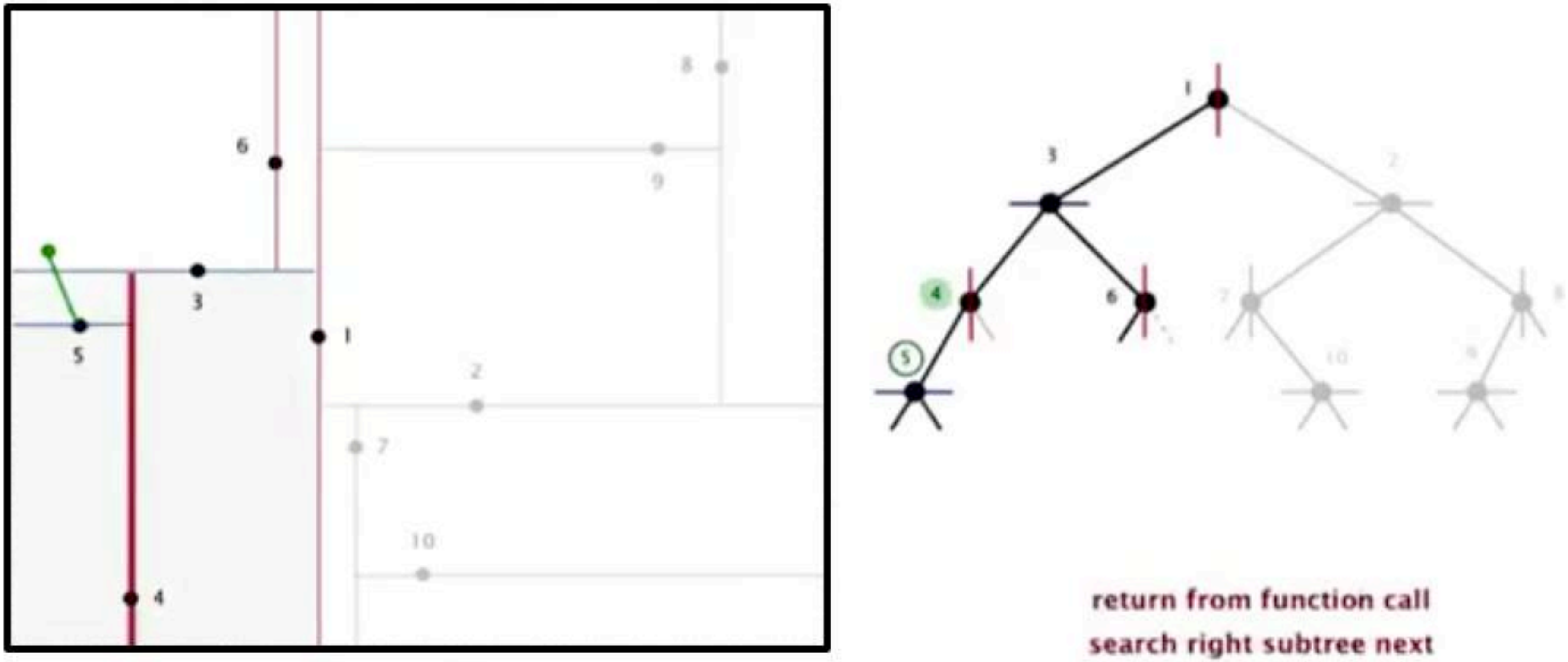




search bottom subtree
return since empty

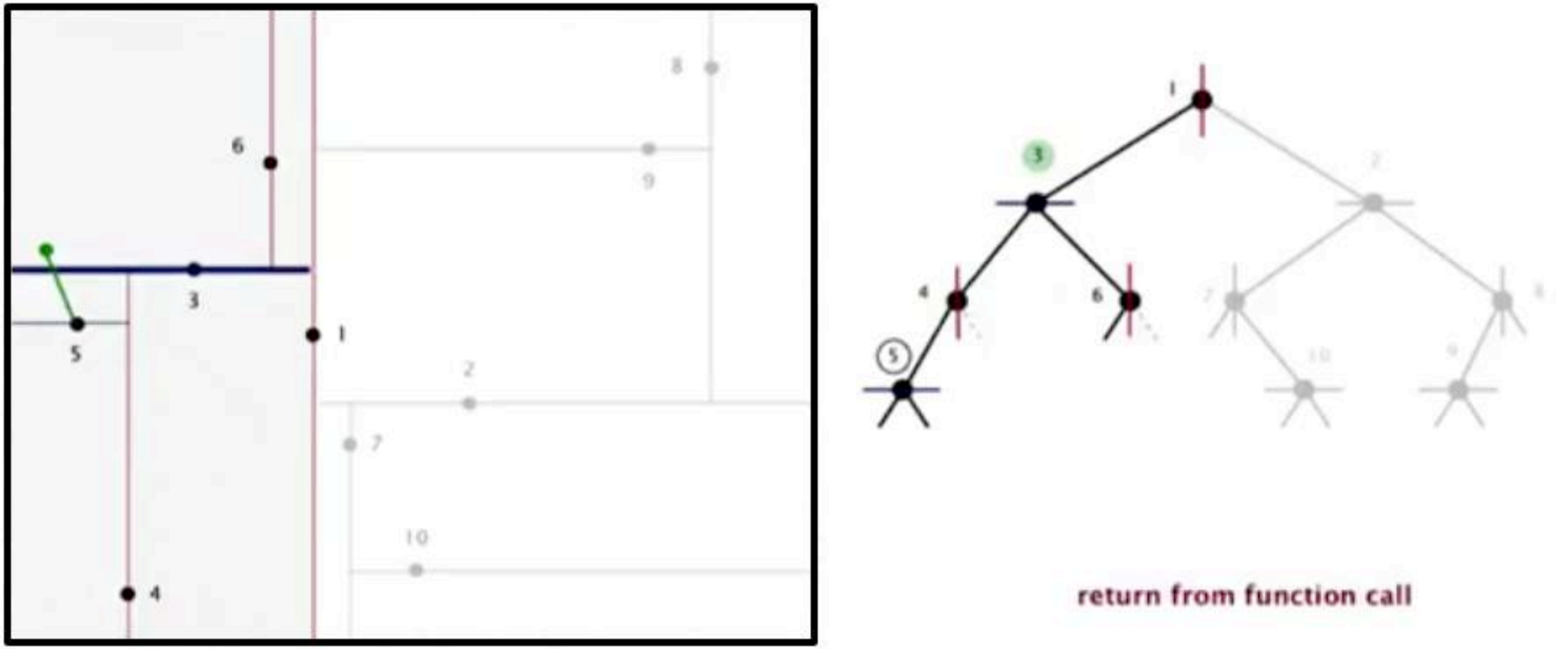
Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



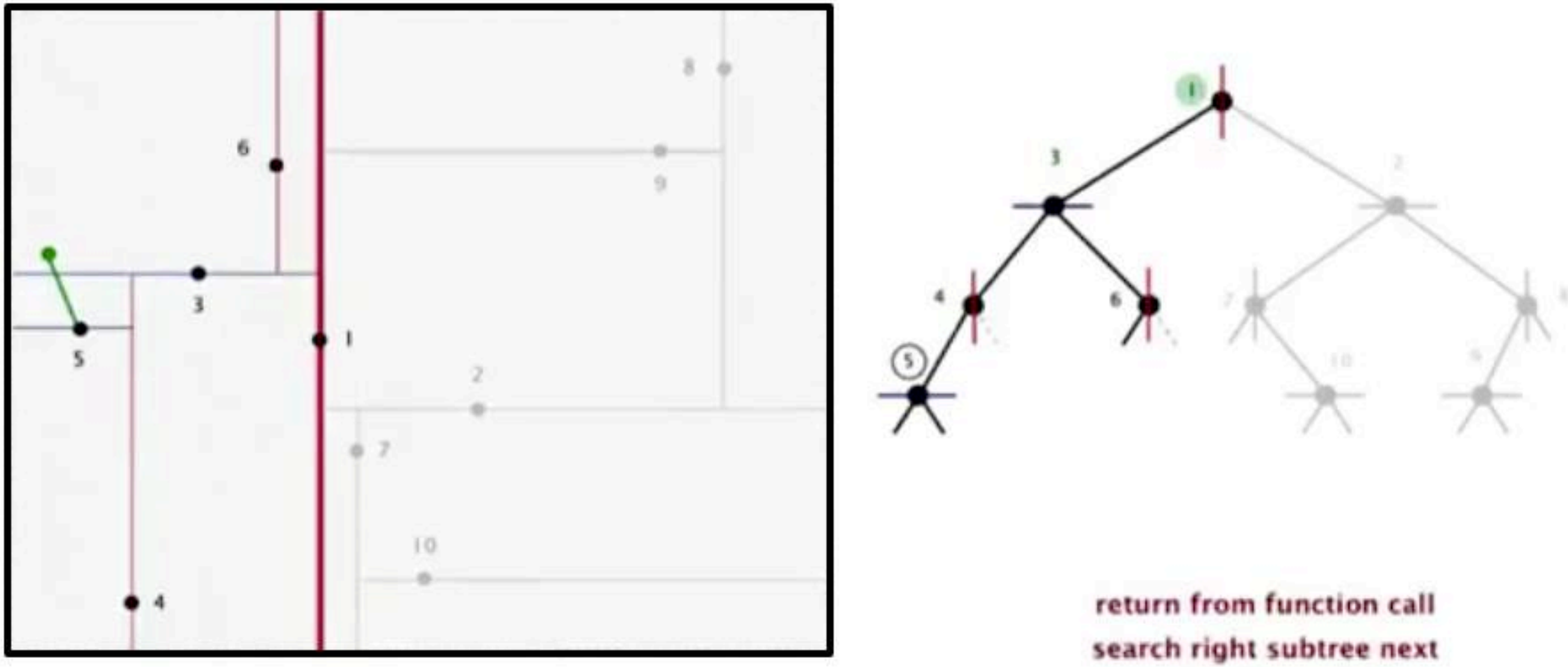
Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



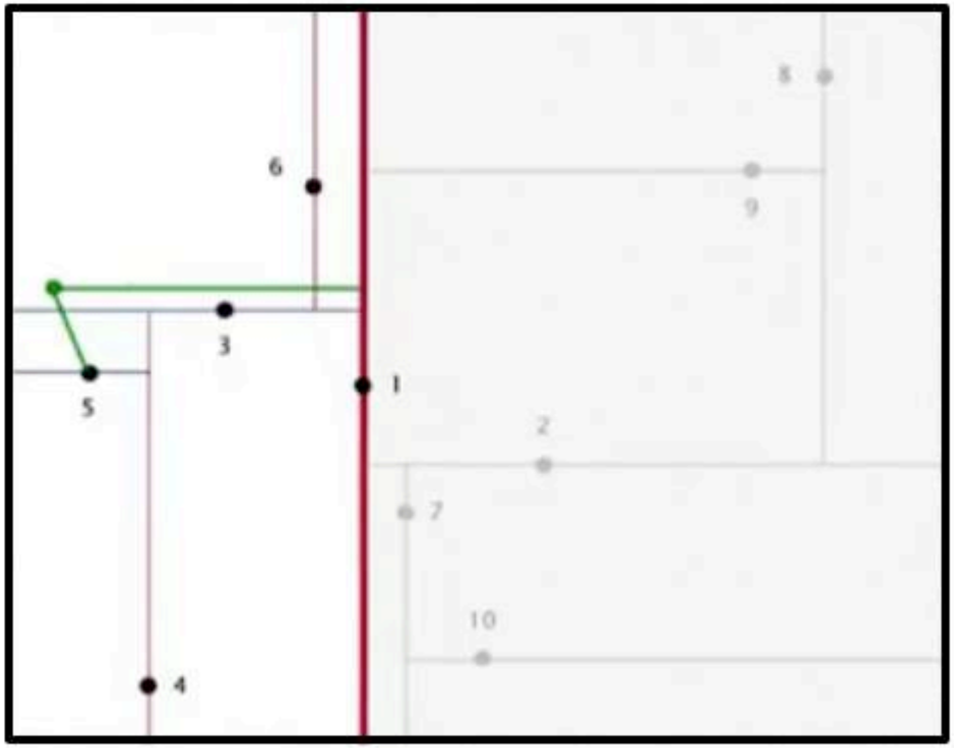
Nearest Neighbor Search in a 2d tree

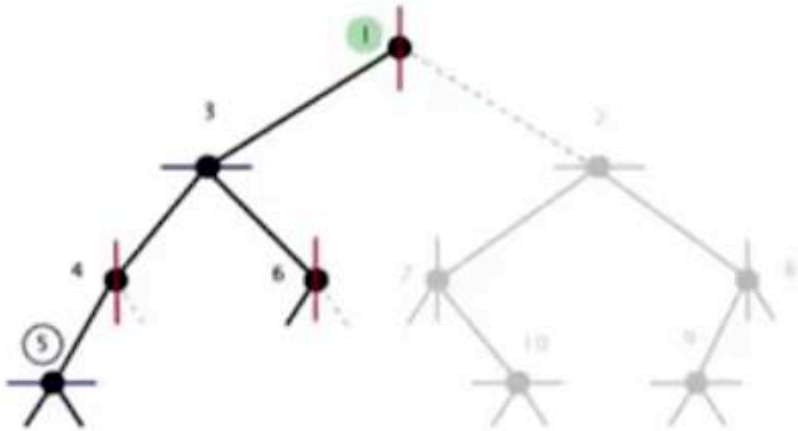
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



Nearest Neighbor Search in a 2d tree

- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).



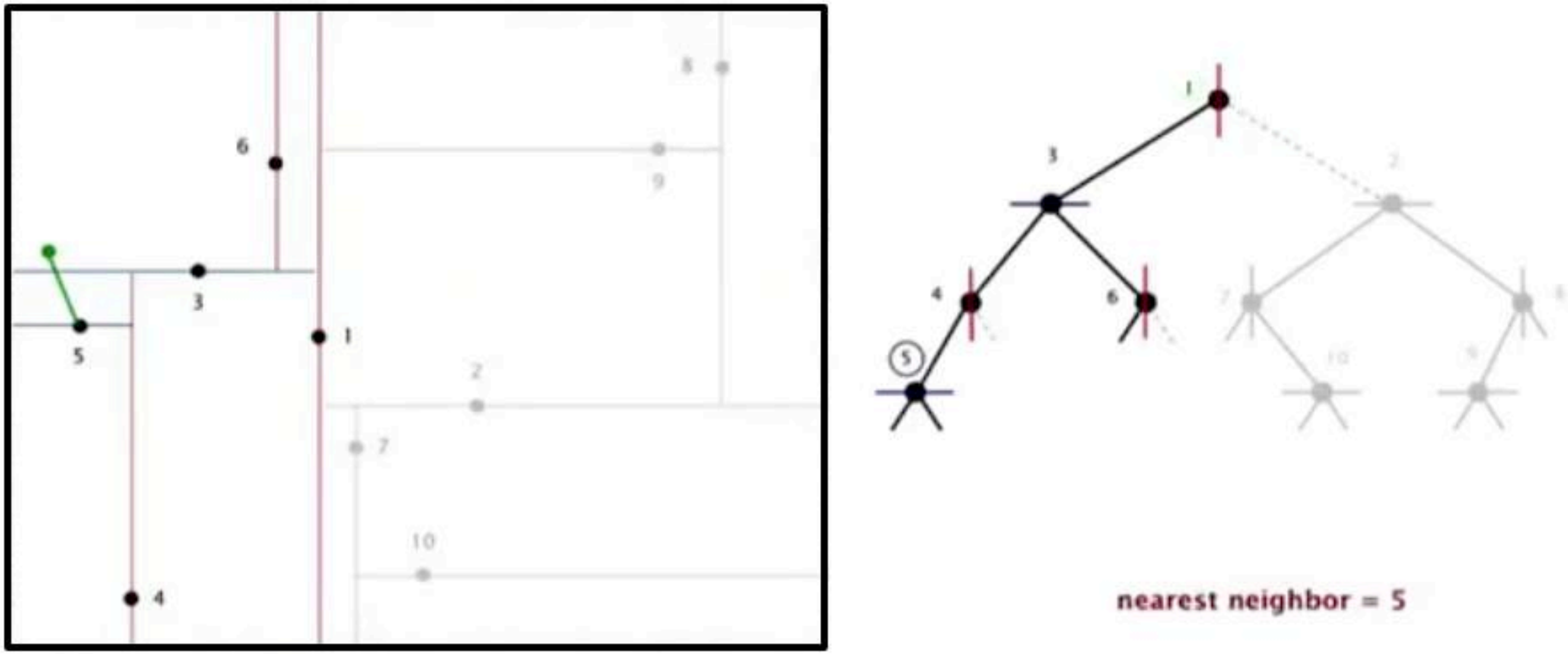


search right subtree
prune since nearest neighbor
can't be in

Nearest Neighbor Search in a 2d tree

Typical case. $\log N$.

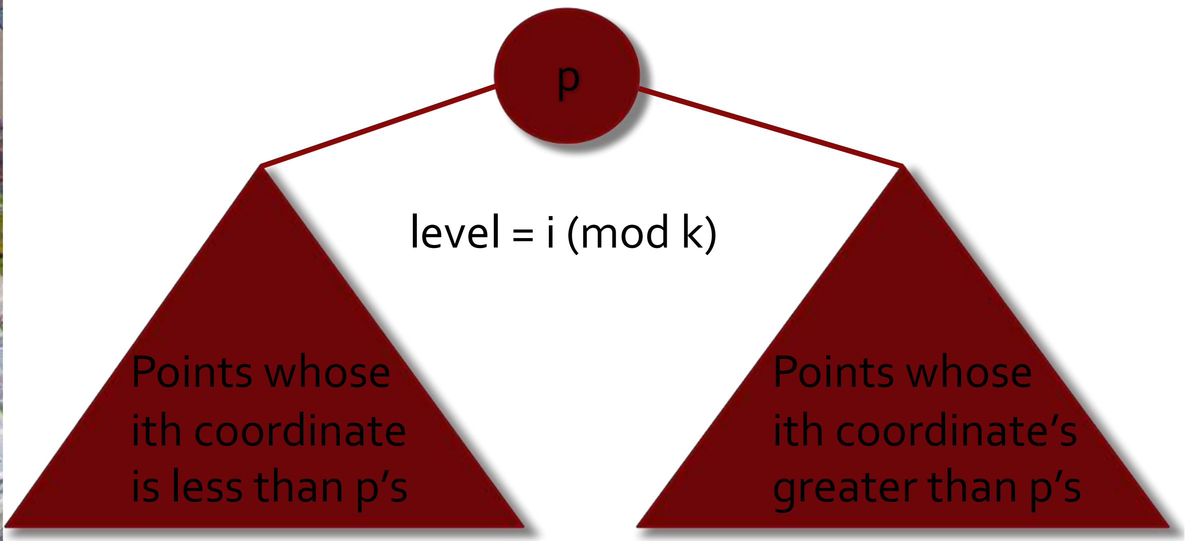
Worst case (even if tree is balanced). N.

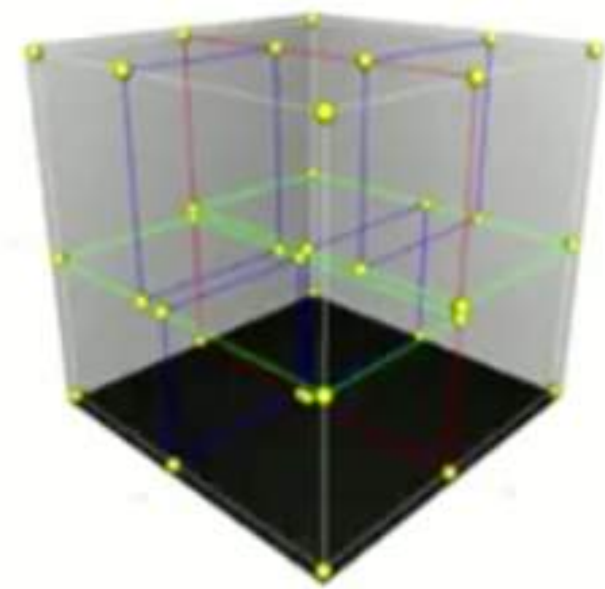


Kd-tree

Recursively partition k-dimensional space into 2 halfspaces.

Implementation: cycle trough dimensions à la 2d trees.





Adapts well to high-dimensional and clustered data
Discovered by an undergrad in an algorithm class.

Locality Sensitive Hashing (LSH)

Detection of near-duplicates

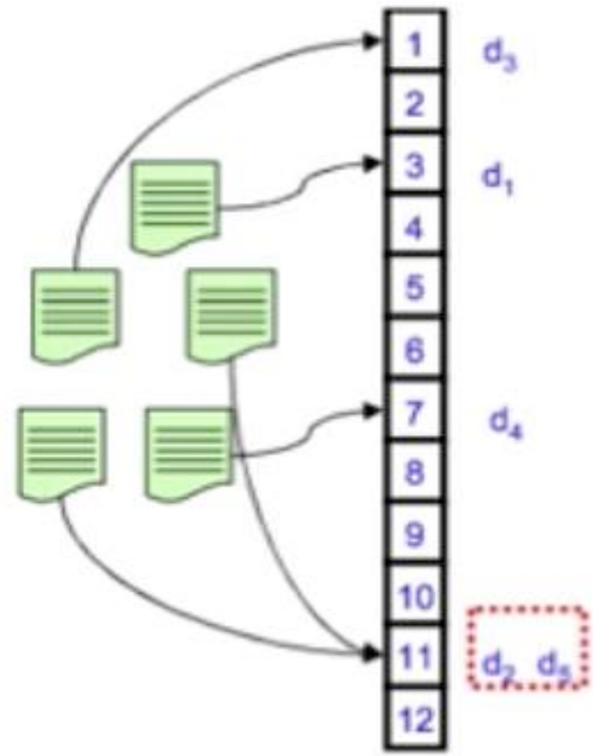
- ✓ Similar files -> similar hash-code

For each file d:

- ✓ Generate K-bit hash-code
- ✓ Insert file into hash-table
- ✓ Collision -> possible duplicate
  - ✓ Compare files in the same bucket

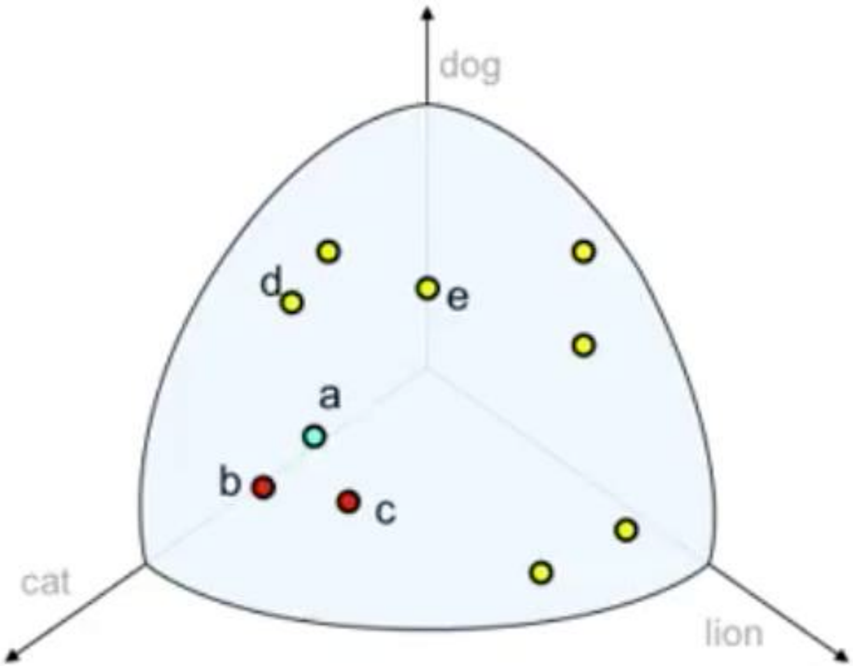
Can miss near-duplicates:

- ✓ Similar hash-codes ≠ same bucket
- ✓ Repeat L times with different hash-tables (randomized)



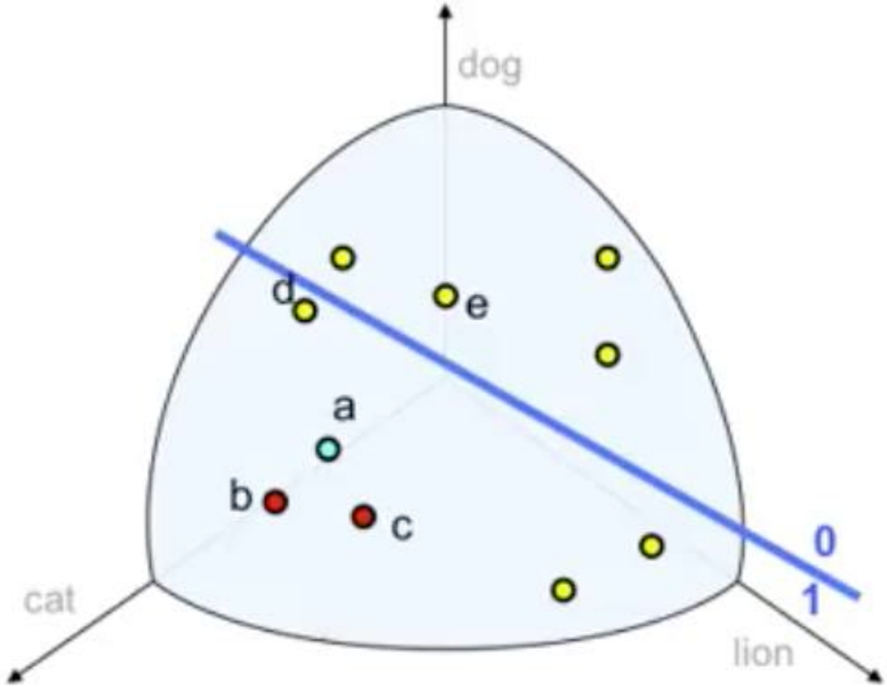
Locality Sensitive Hashing (LSH)

1. Want: similar hash-codes for nearby points
2. Generate random hyperplanes: h_1 h_2 h_3



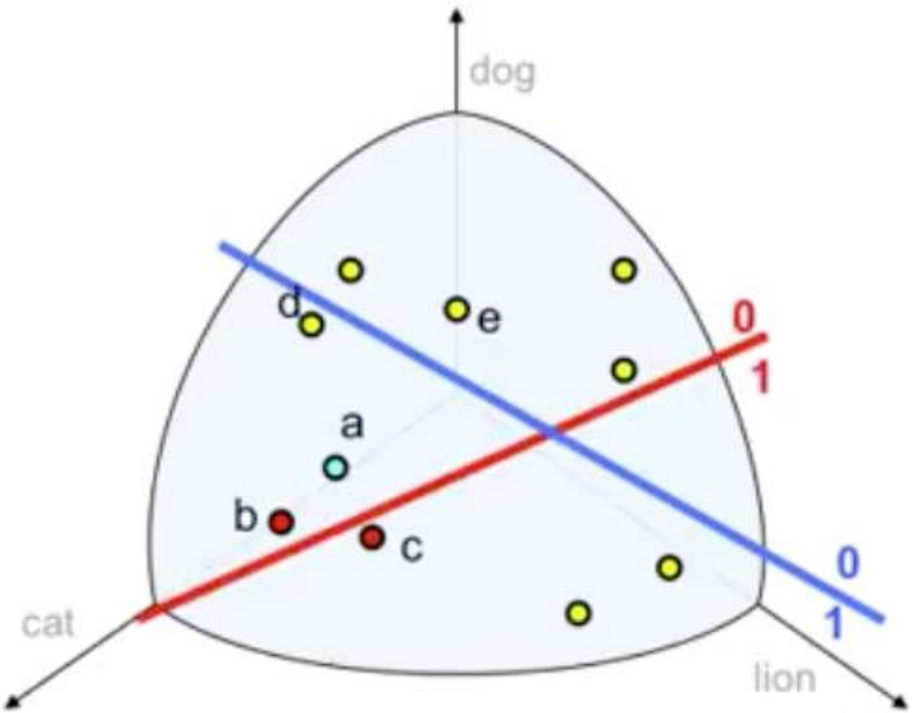
Locality Sensitive Hashing (LSH)

1. Want: similar hash-codes for nearby points
2. Generate random hyperplanes: h_1 h_2 h_3



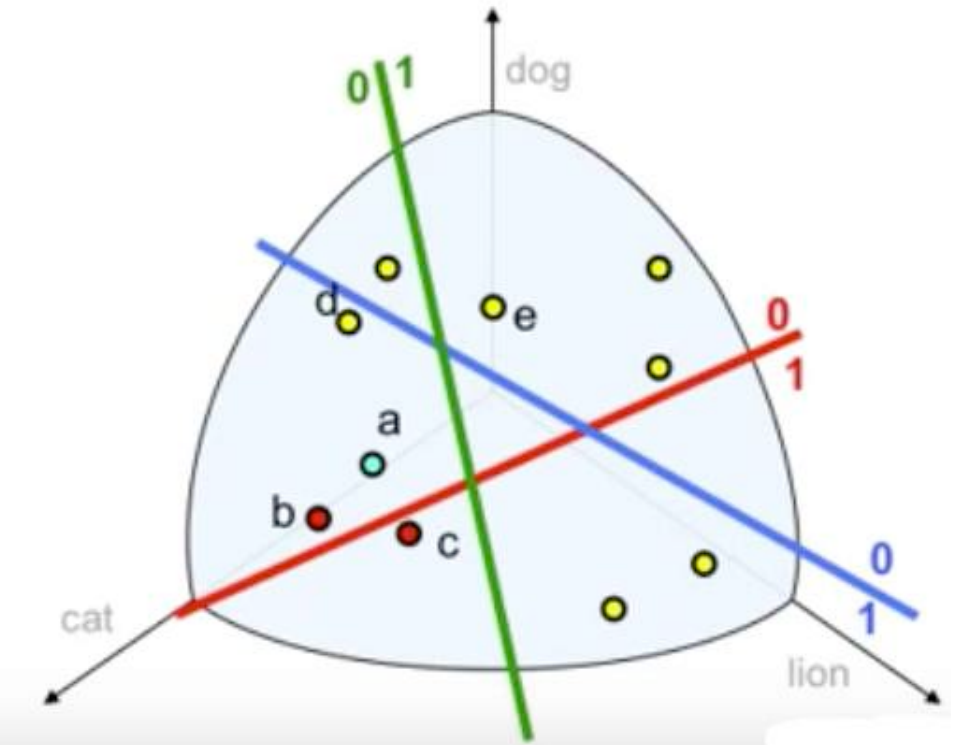
Locality Sensitive Hashing (LSH)

1. Want: similar hash-codes for nearby points
2. Generate random hyperplanes: h_1 h_2 h_3



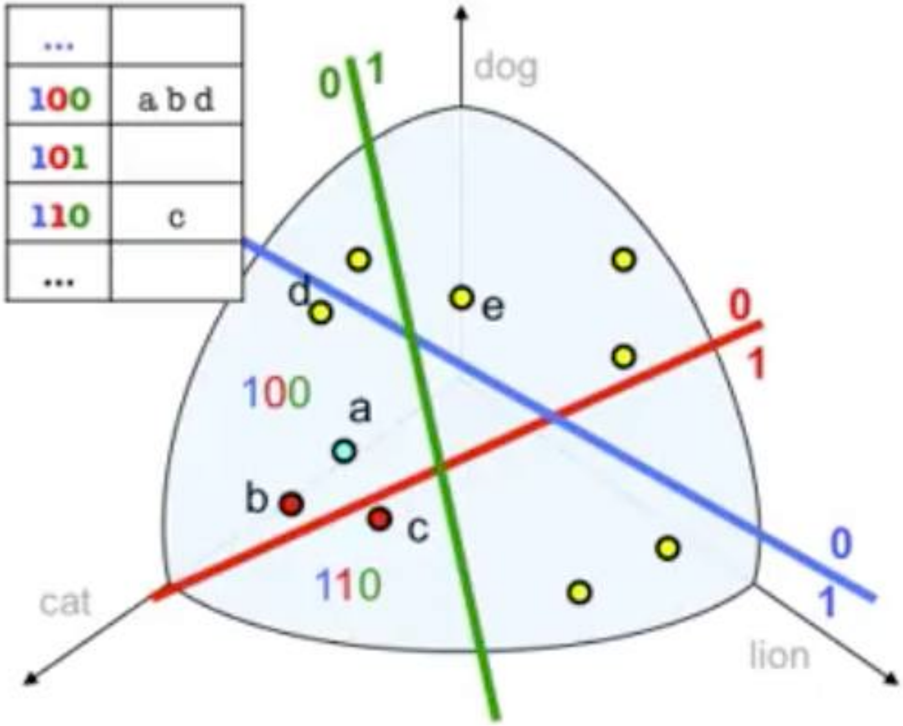
Locality Sensitive Hashing (LSH)

1. Want: similar hash-codes for nearby points
2. Generate random hyperplanes: h_1 h_2 h_3



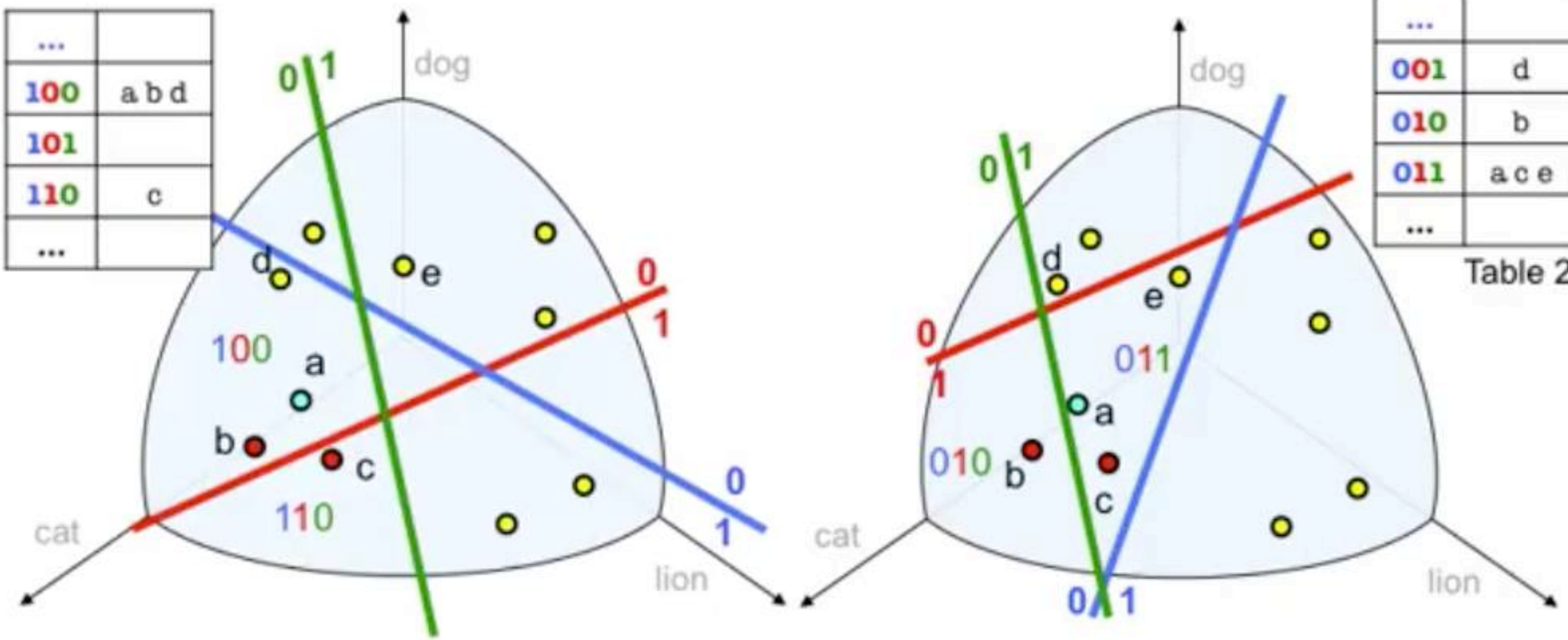
Locality Sensitive Hashing (LSH)

1. Want: similar hash-codes for nearby points
2. Generate random hyperplanes: h_1 h_2 h_3
3. Compare **a** to points with same hash-code
   - **b** … indeed similar to a
   - **d** … false positive, will be eliminated
   - **c** … different hash-code, will miss it



Locality Sensitive Hashing (LSH)

5. Repeat with different hyperplanes: h_4 h_5 h_6



...	
100	a b d
101	
110	c
...	

...	
001	d
010	b
011	a c e
...	

Table 2

Acknowledgments

PATREO
UFMG
30 ANOS FAPEMIG 1986 - 2016
CAPES
CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

References

Jin, J. S. (2003). Indexing and Retrieving High Dimensional Visual Features, pages 178–203. Springer Berlin Heidelberg, Berlin, Heidelberg.

Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., and Equitz, W. (1994). Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3):231–262.

Idris, F. and Panchanathan, S. (1997). Review of image and video indexing techniques. *J. Vis. Comun. Image Represent.*, 8(2):146–166.

Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for similarity search: A survey. *CoRR*, abs/1408.2927.