kNN[1] and AkNN[2] Searching in High Dimensions: Foundations, Applications and New Challenges

Prof. Spyros Sioutas

CEID@Upatras

[1] *kNN:k-Nearest Neighbor*

[2] *AkNN: All k-Nearest Neighbor or kNN Join*

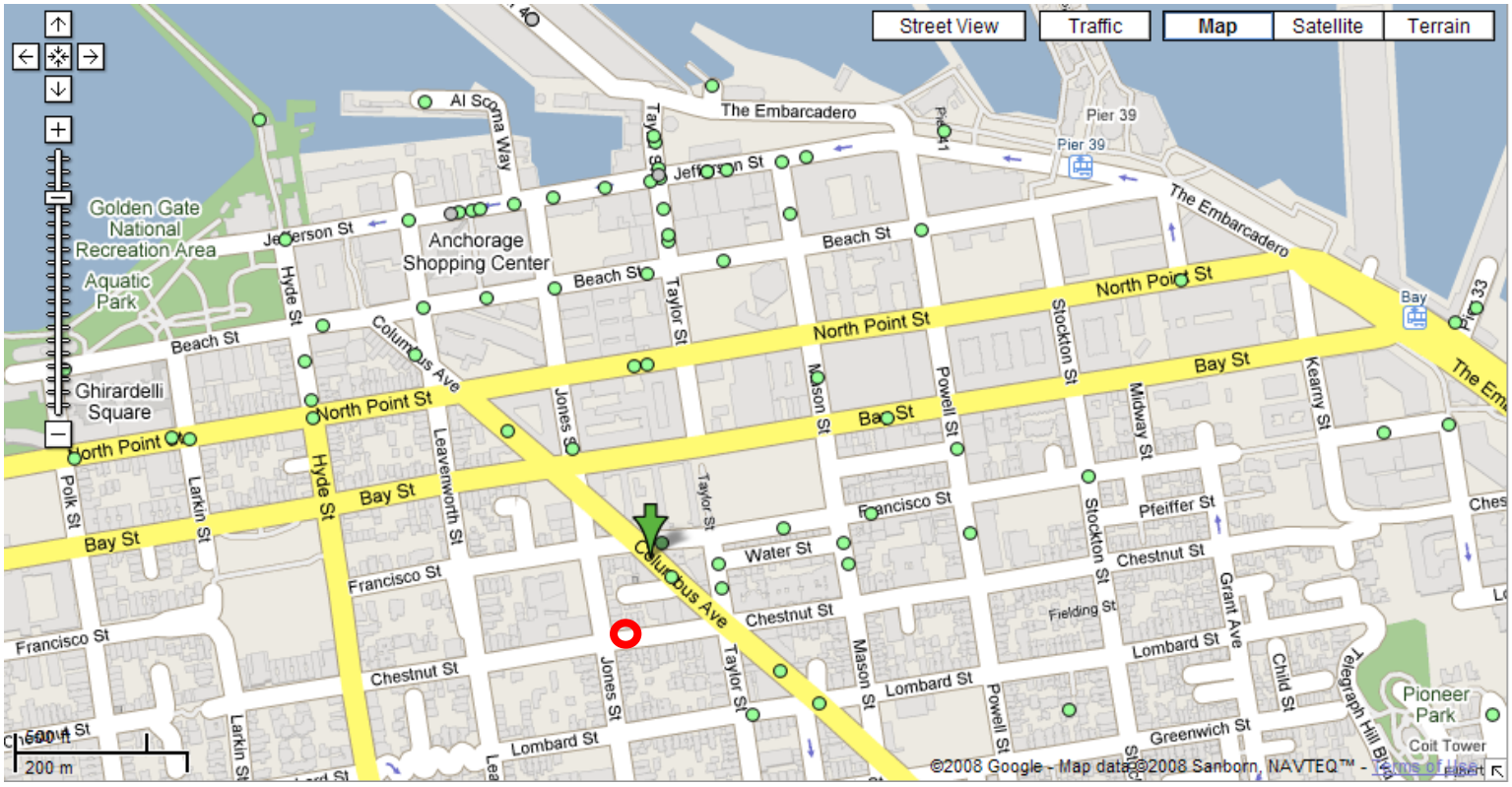
Nearest-Neighbor(s) (NN) Query

- Example:
  - **key=Salary:** retrieve the employee whose salary is closest to $50,000 (i.e., 1-NN).
  - **key=Age:** retrieve the 5 employees whose age is closest to 40 (i.e., k-NN, k=5).

ID	Name	Age	Salary	#Children

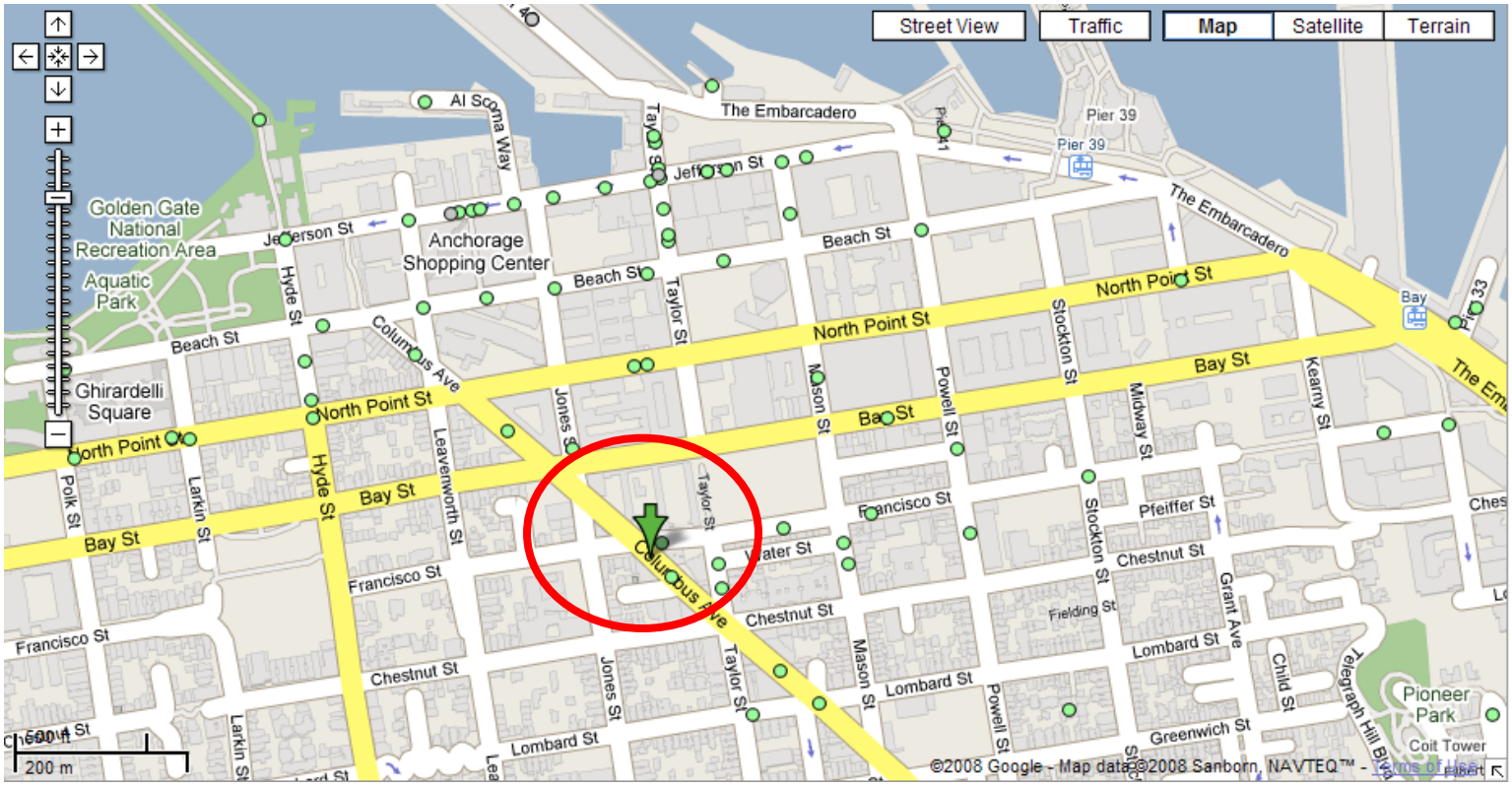
Nearest Neighbor(s) Query

- What is the closest restaurant to my hotel?



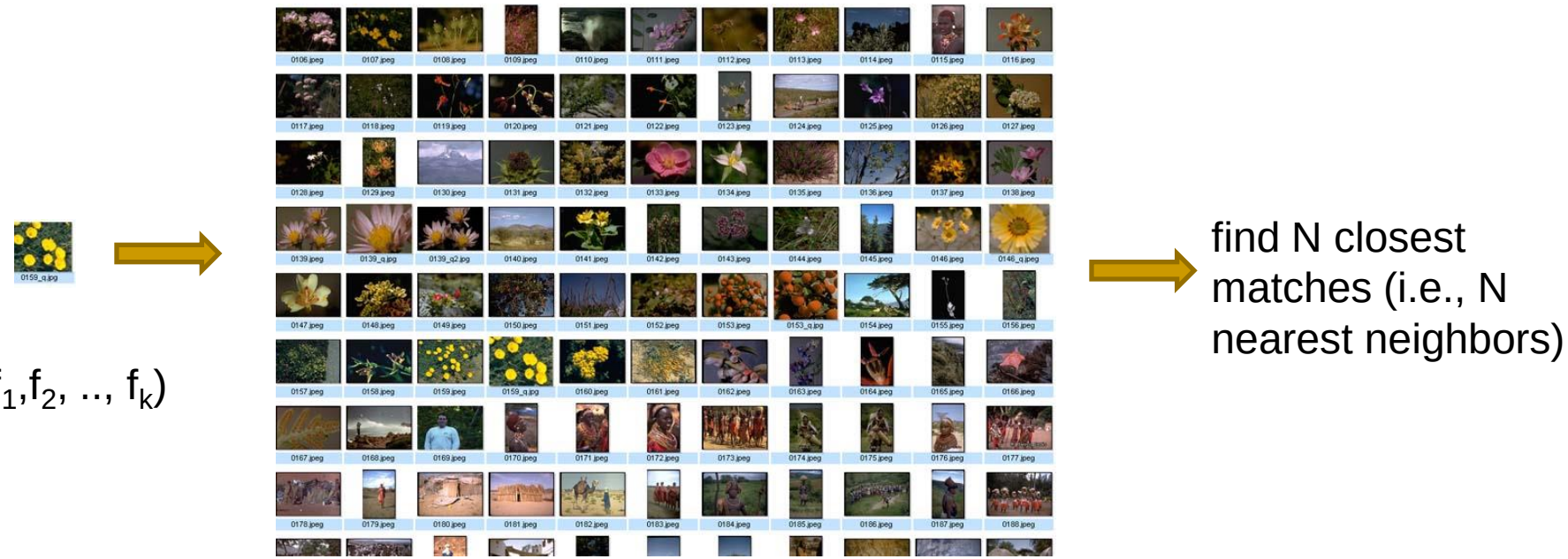
Nearest Neighbor(s) Query (cont'd)

- Find the 4 closest restaurants to my hotel



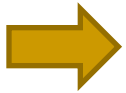
Nearest Neighbor Query in High Dimensions

- Very important and practical problem!
  - Image retrieval



Nearest Neighbor Query in High Dimensions

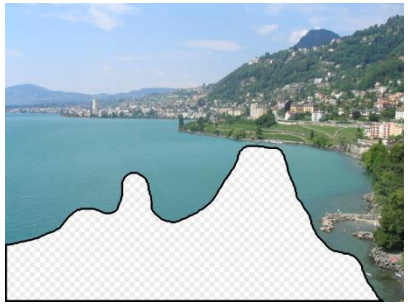
- Face recognition



find closest match
(i.e., nearest neighbor)

Scene Completion Problem

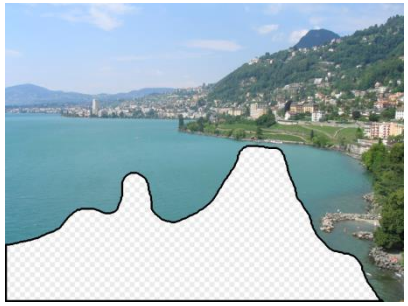
Scene Completion Problem

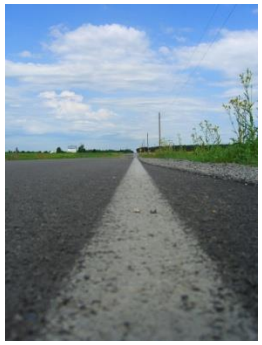


Scene Completion Problem

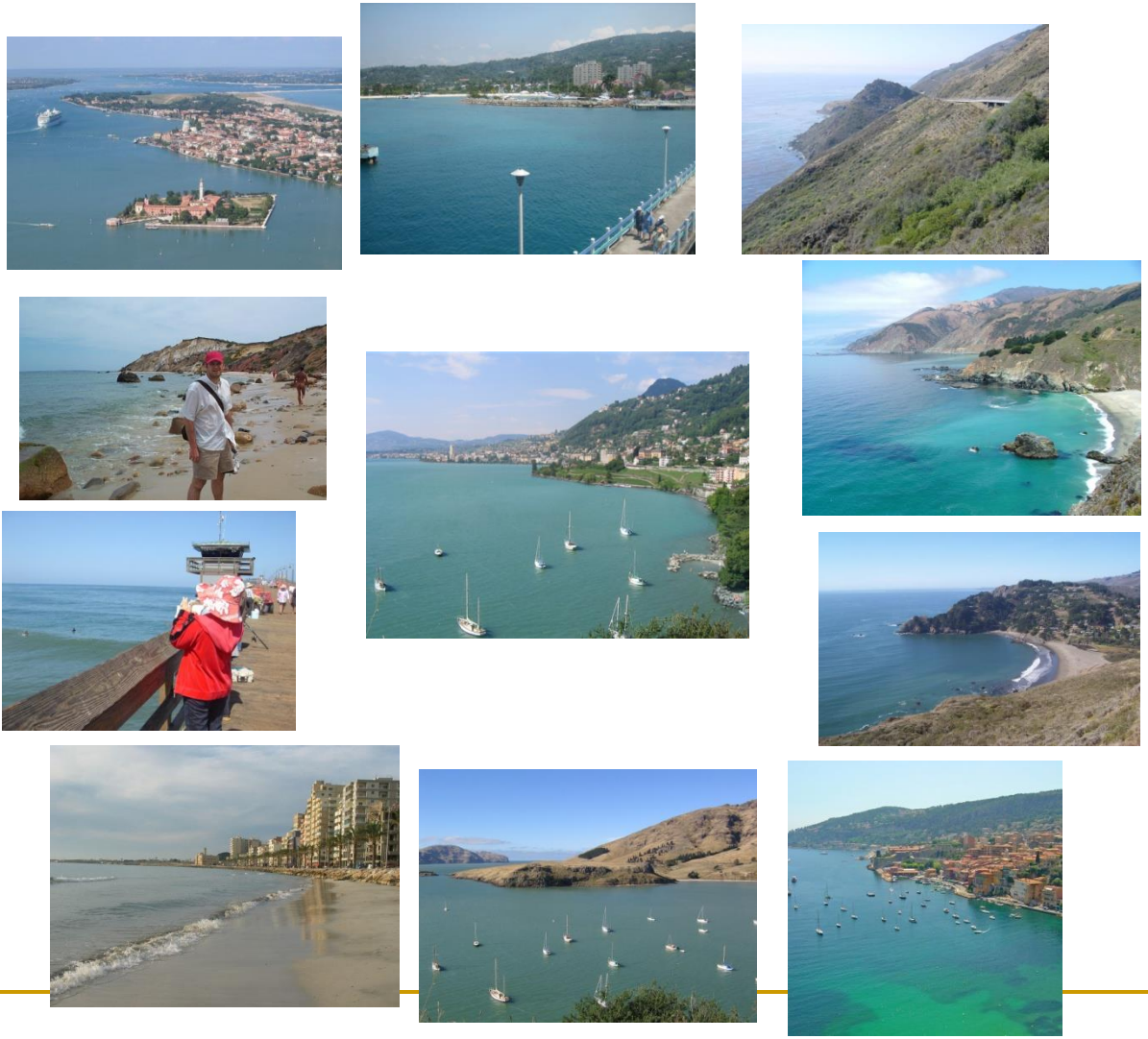




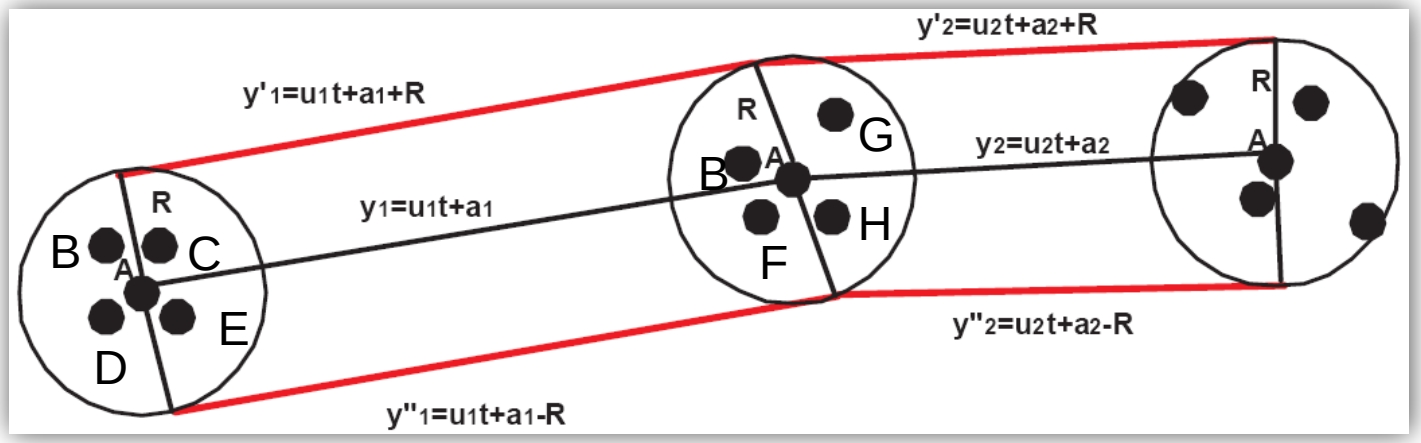



Scene Completion Problem



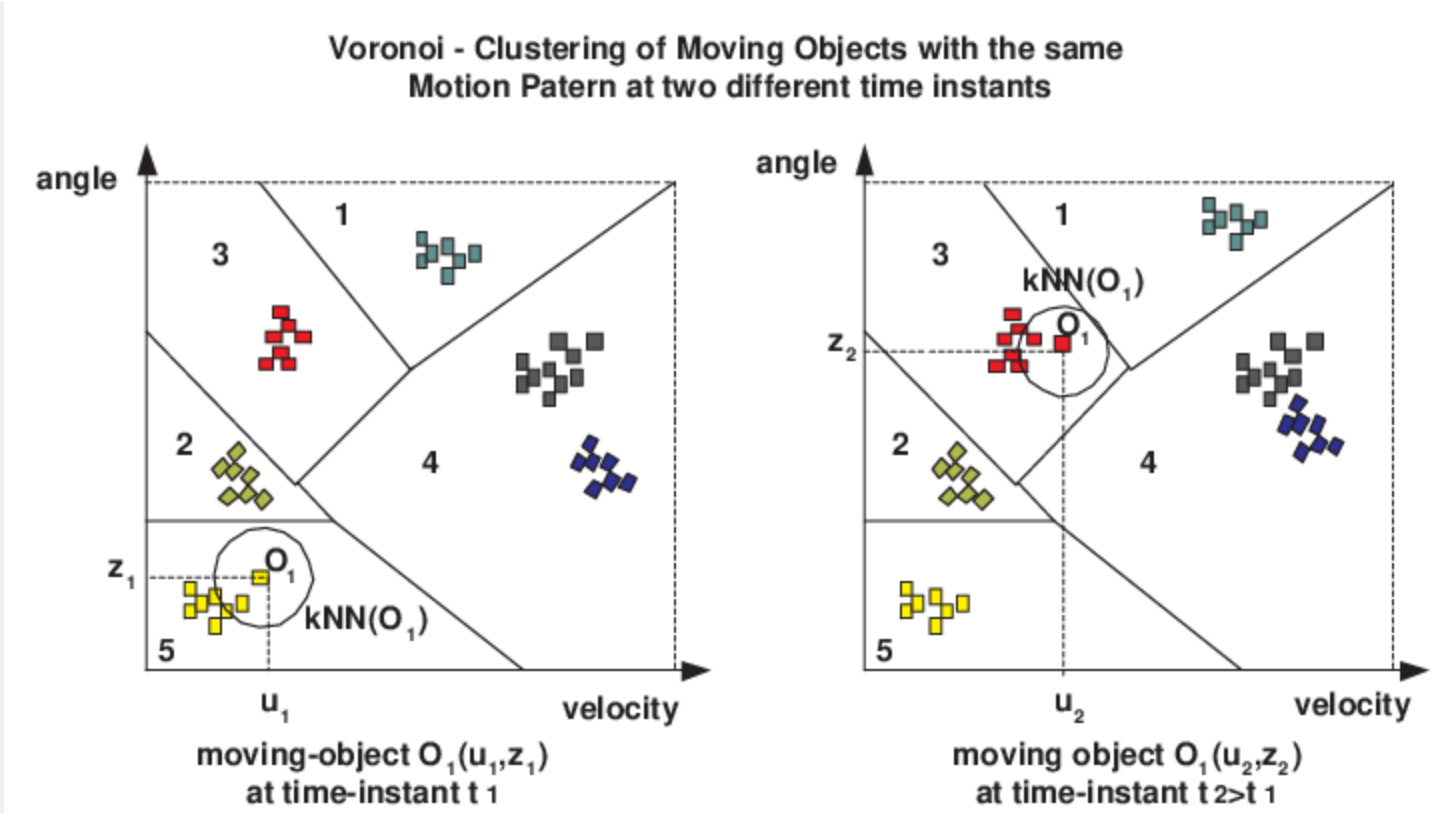
10 nearest neighbors

Trajectory poly-lines and k-Anonimity (Privacy Preserving Spatio-Temporal Databases)

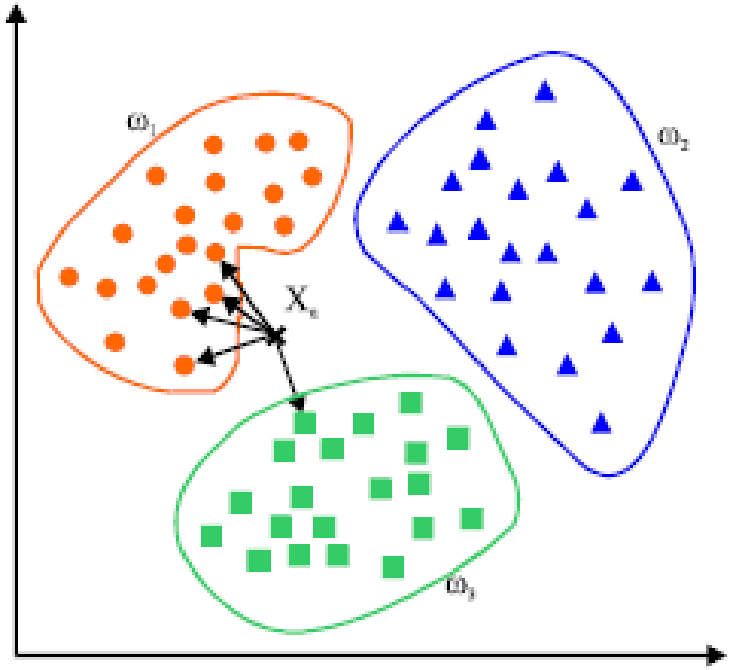


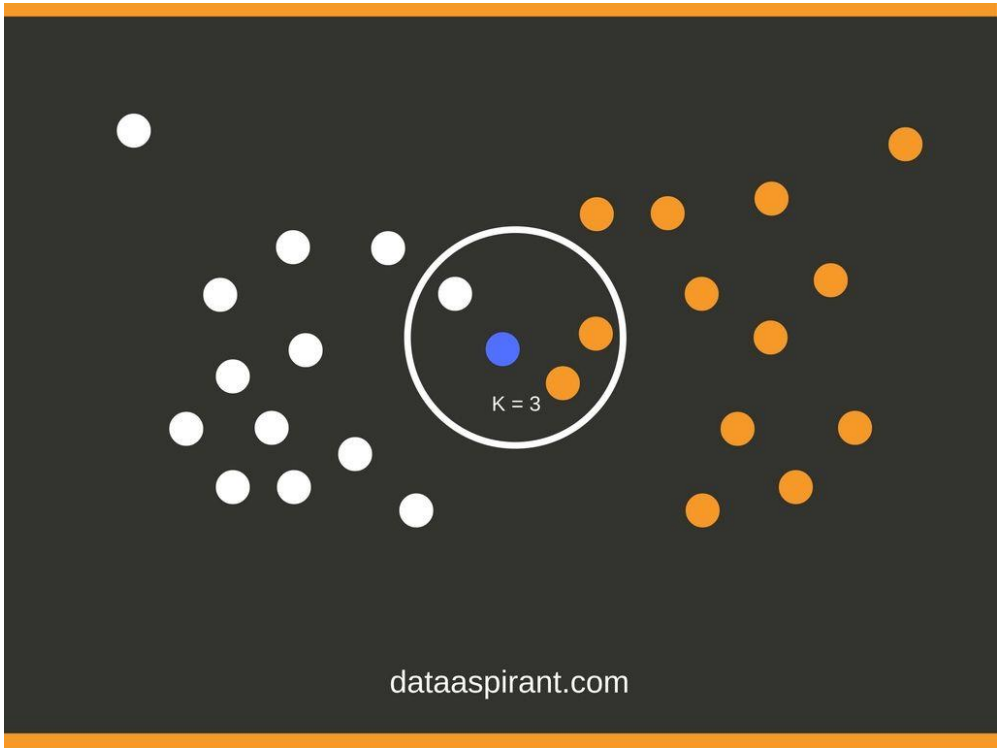
2-D Buffer and Boundary Trajectories y' and y'' of mobile user A

Privacy Preserving Spatio-Temporal Databases



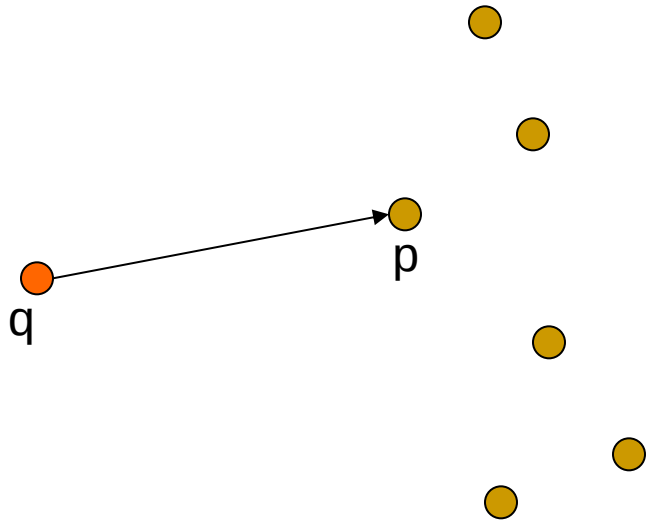
ML, AI and Data Mining: kNN Classifier





Nearest Neighbor (NN) Search

- Given: a set *P* of *n* points in R^d
- Goal: find the *nearest neighbor* *p* of *q* in *P*



d(p, q) ≥ 0 **for all *p* and *q***
d(p, q) = d(q, p) **(Symmetry)**
d(p, r) ≤ d(p, q) + d(q, r)
(Triangular Inequality)

$$p = (x_1, y_1) \qquad q = (x_2, y_2)$$

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Euclidean distance metric

Minkowski distance metric

$$d = \left(\sum_{k=1}^{n} | p_k - q_k |^r\right)^{\frac{1}{r}}$$

r = 1. City block (Manhattan, taxicab, L_1 norm) distance
- f.e. hamming distance

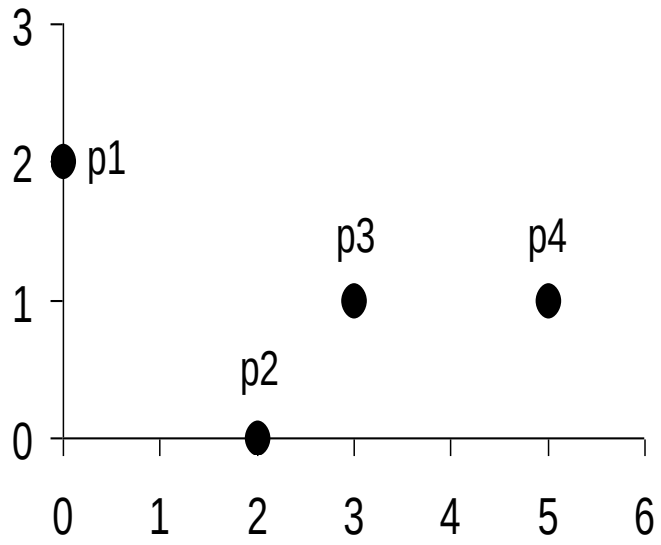
r = 2. Euclidean Distance

$r \to \infty$. "supremum" (L_{max} norm, L_∞ norm) distance.

r parameter

n dimensions

Απόσταση Minkowski



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

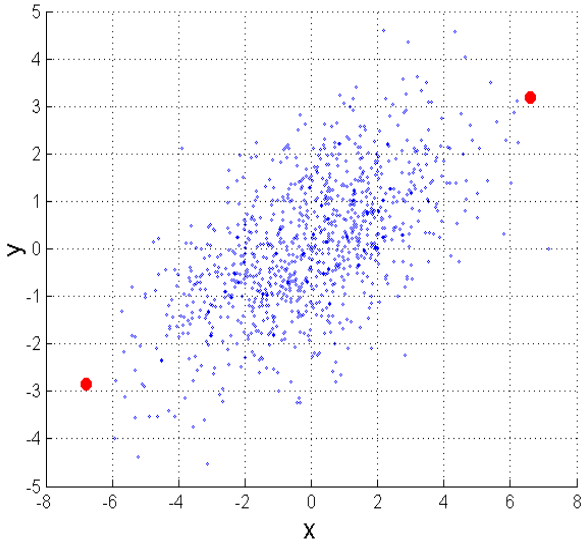
L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

L_∞	p1	p2	p3	p4
p1	0	2	3	5
p2	2	0	1	3
p3	3	1	0	2
p4	5	3	2	0

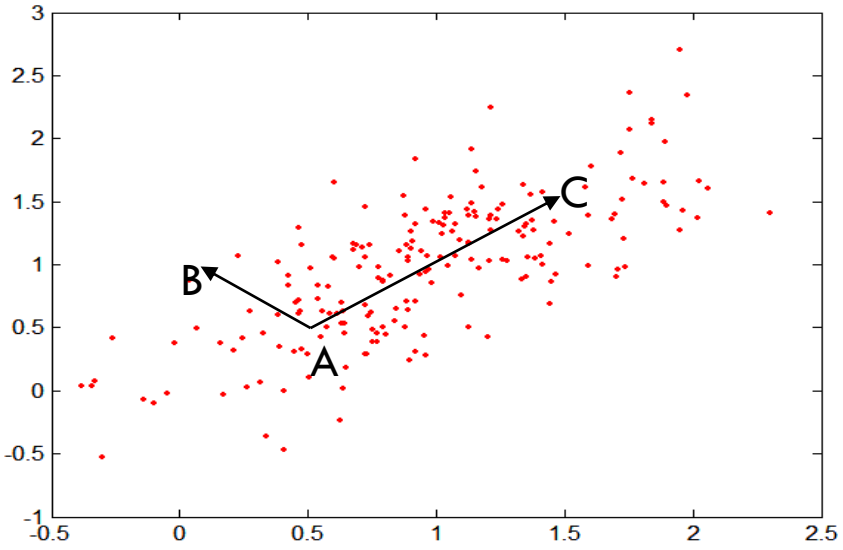
Mahalanobis distance metric

$$mahalanobis(p,q) = (p-q)\Sigma^{-1}(p-q)^T$$



Σ= Covariance Matrix of data X

$$\Sigma_{j,k} = \frac{1}{n-1}\sum_{i=1}^{n}(X_{ij} - \overline{X}_j)(X_{ik} - \overline{X}_k)$$



Covariance Matrix:

$$\Sigma = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$$

A: (0.5, 0.5)

B: (0, 1)

C: (1.5, 1.5)

Mahal(A,B) = 5

Mahal(A,C) = 4

For Spatial kNN we will use …

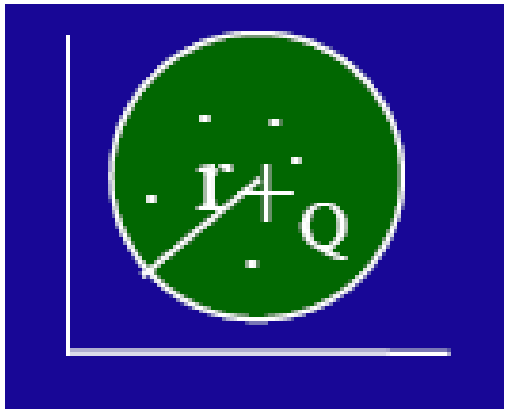
- Quadtrees
- KD-trees
- Range trees (R-trees)

Interpreting Queries Geometrically

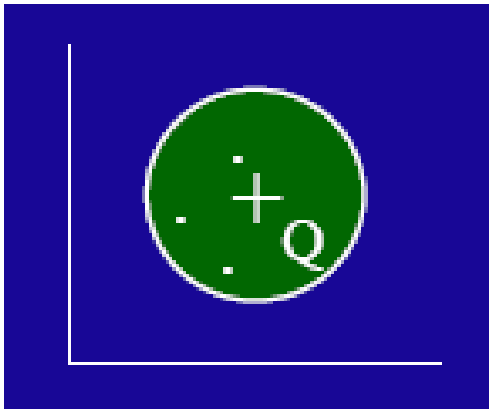
- Multi-dimensional keys can be thought as "points" in high dimensional spaces.

Queries about records → Queries about points

Nearest Neighbor Search -Variations



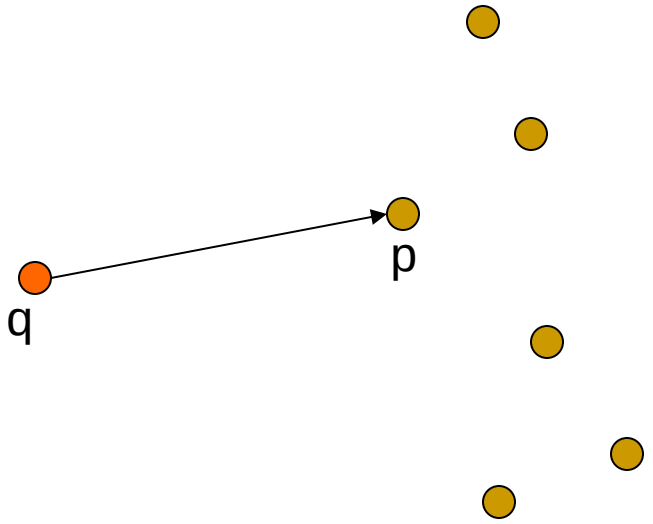
r-search: the distance tolerance is specified.



k-nearest-neighbor-queries: the number of close matches is specified.

Nearest Neighbor (NN) Search

- Naïve approach
  - Compute the distance from the query point to every other point in the database, keeping track of the "best so far".
  - Running time is O(*n*).



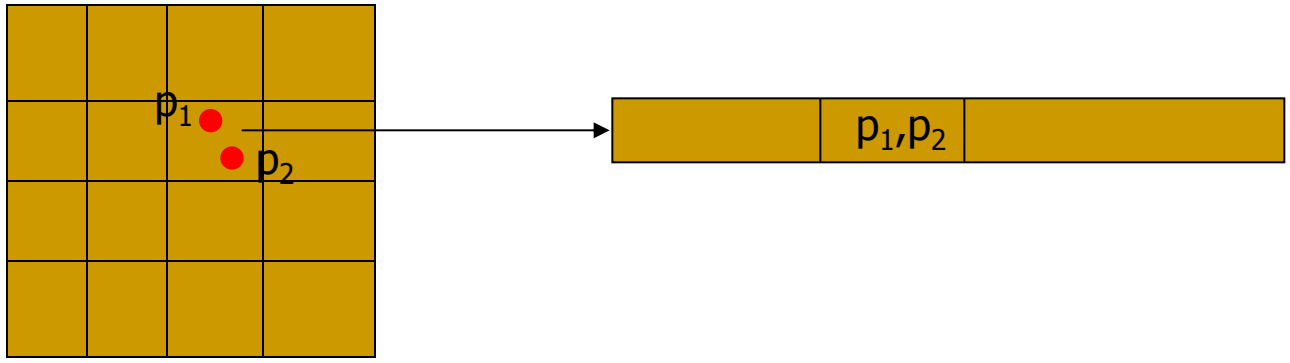
Array (Grid) Structure

(1) Subdivide the plane into a grid of M x N square cells (same size)

(2) Assign each point to the cell that contains it.

(3) Store as a 2-D (or N-D in general) array:

"each cell contains a link to a list of points stored in that cell"

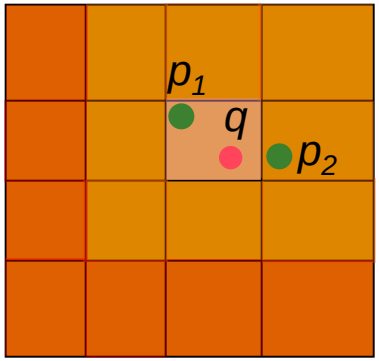


Array (Grid) Structure

Algorithm

* Look up cell holding query point.

* First examine the cell containing the query, then the cells adjacent to the query (i.e., there could be points in adjacent cells that are closer).



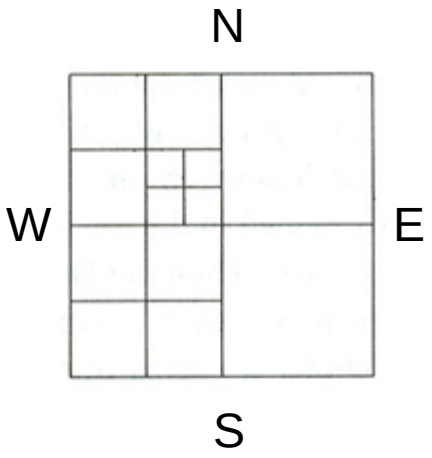
Comments

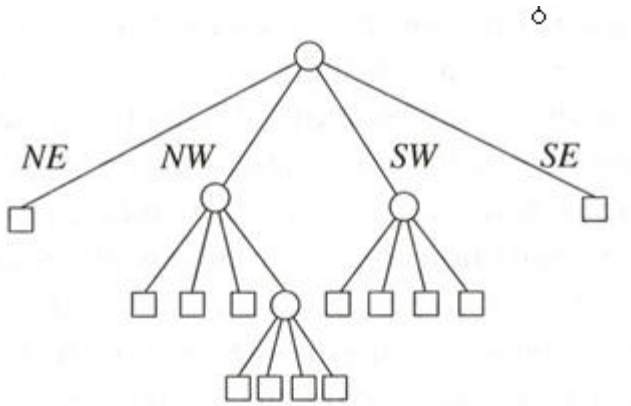
* Uniform grid inefficient if points unequally distributed.
  - Too close together: long lists in each grid, serial search.
  - Too far apart: search large number of neighbors.

* Multiresolution grid can address some of these issues.

Quadtree

- A tree in which each internal node has up to four children.
- Every node in the quadtree corresponds to a square.
- The children of a node v correspond to the four quadrants of the square of v.
- The children of a node are labelled *NE*, *NW*, *SW*, and *SE* to indicate to which quadrant they correspond.





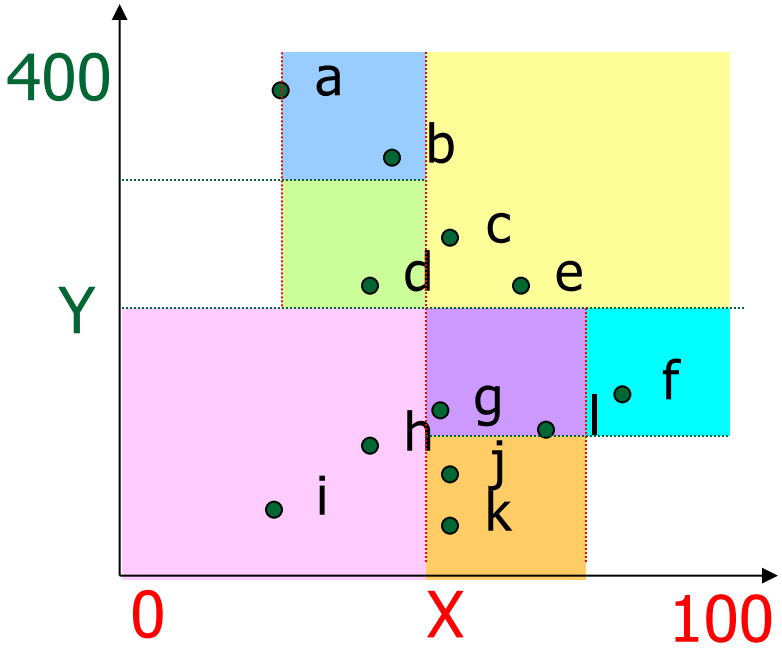
Quadtree Construction (data stored at leaves)

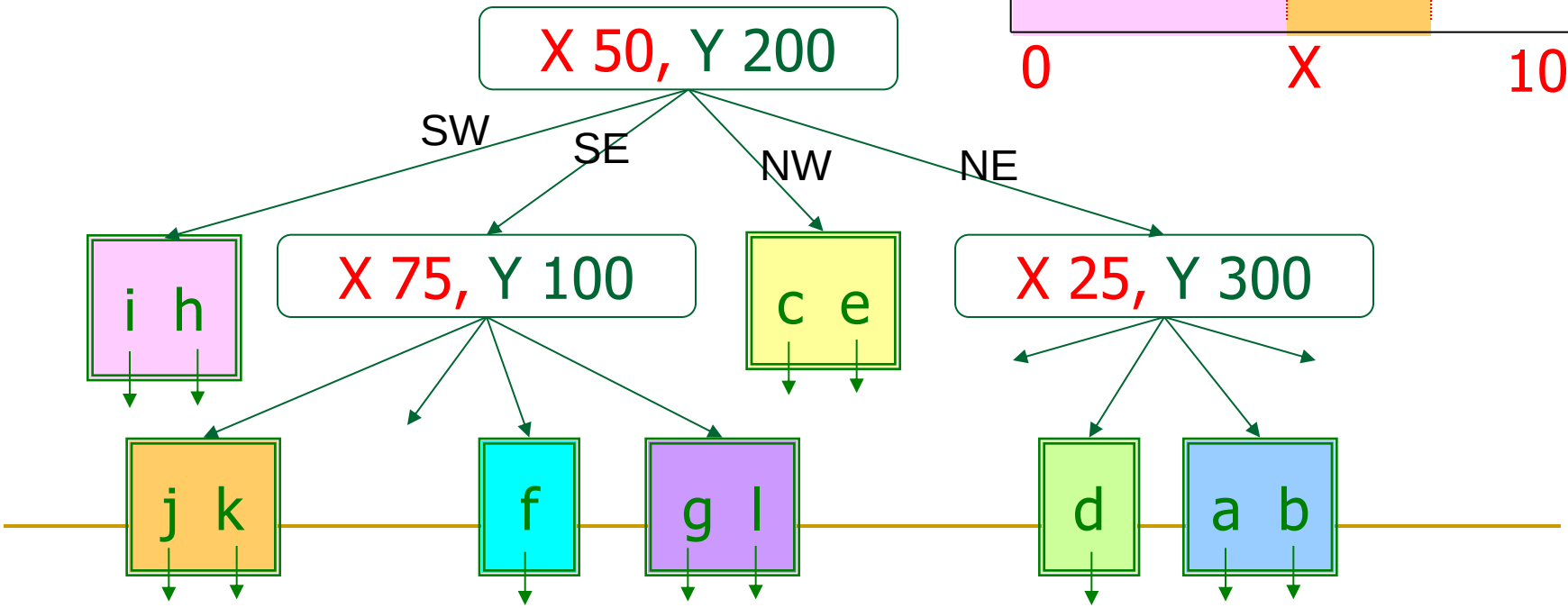
Input: point set P

while Some cell C contains more than “k” points **do**

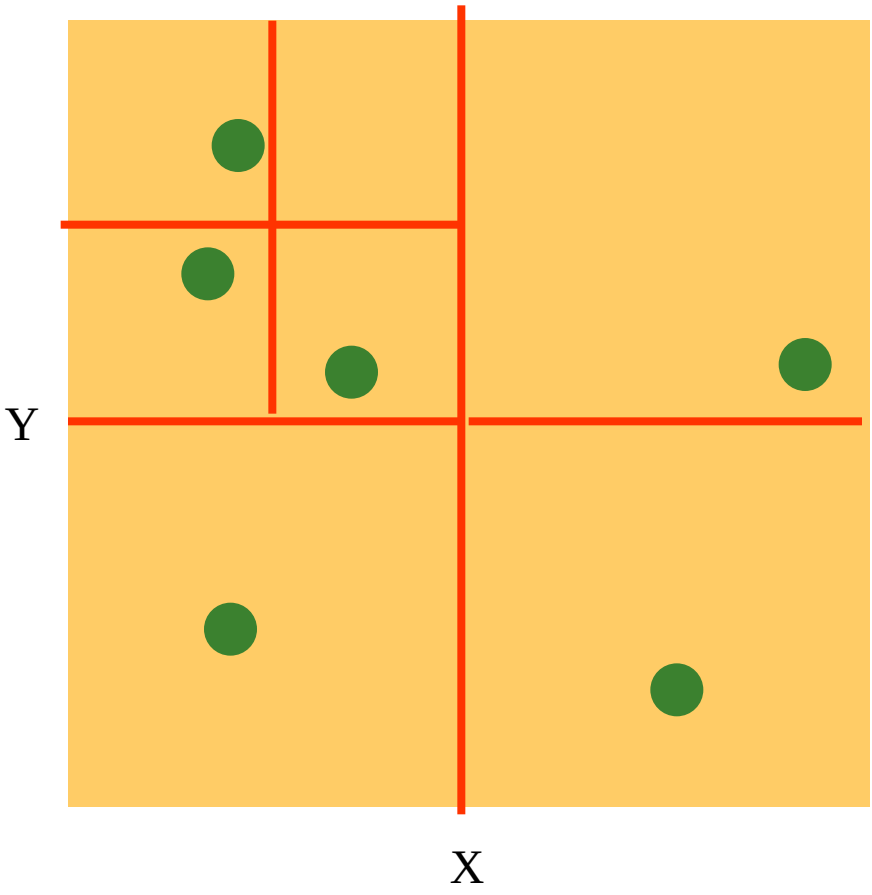
Split cell C

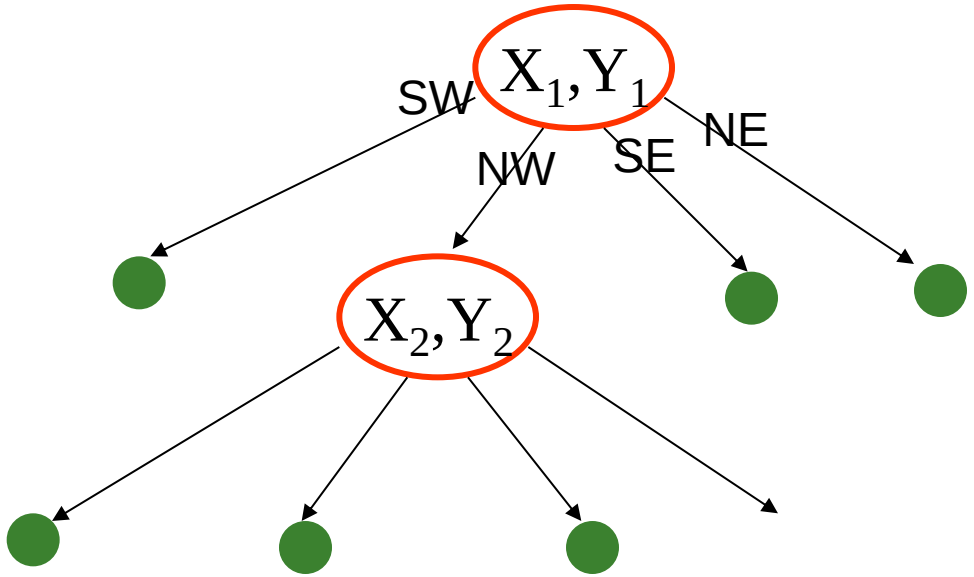
end



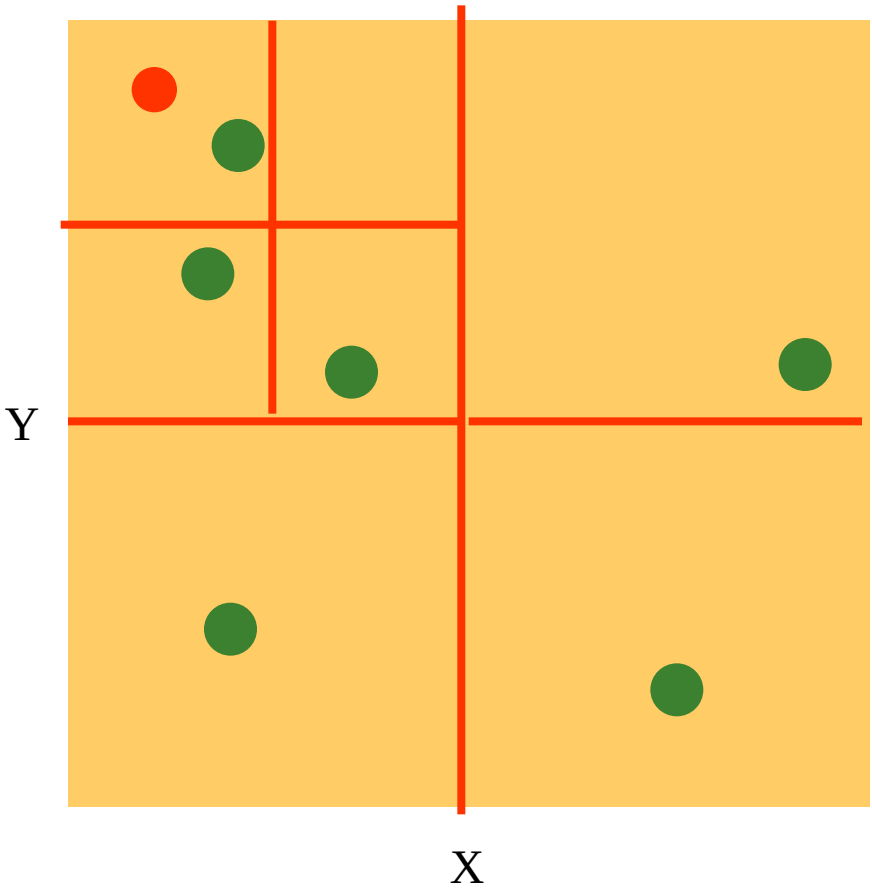


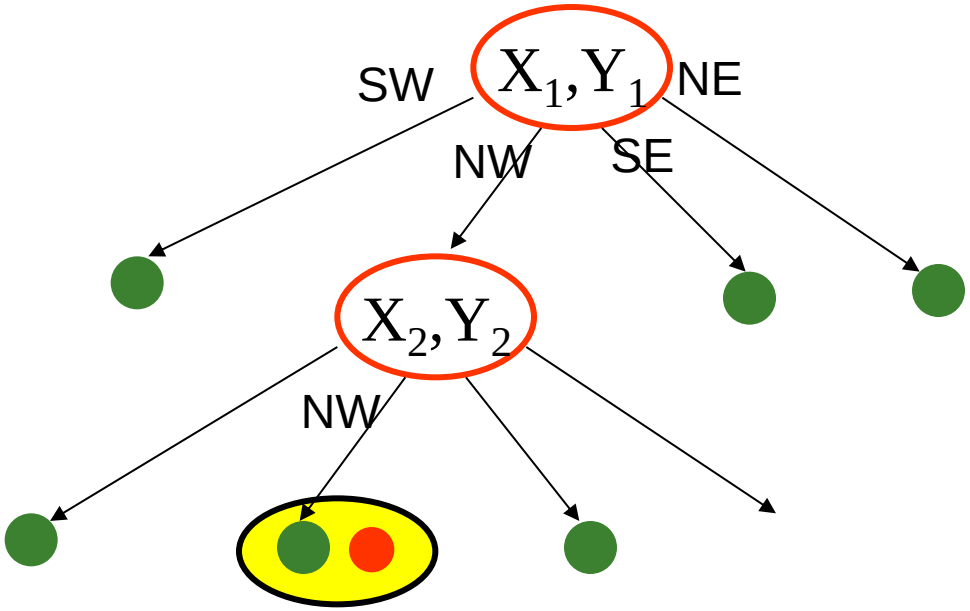
Quadtree – Nearest Neighbor Query



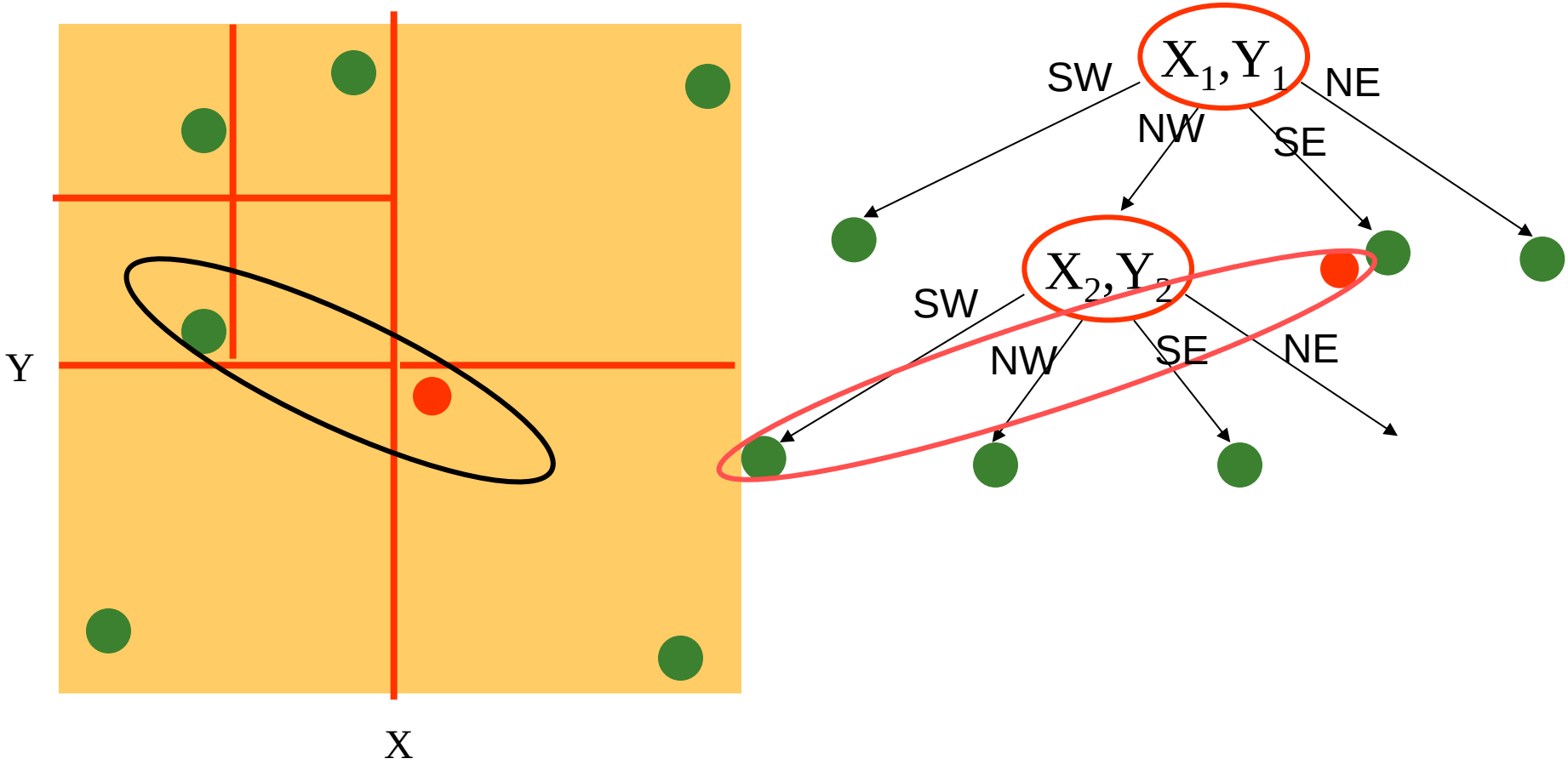


Quadtree – Nearest Neighbor Query

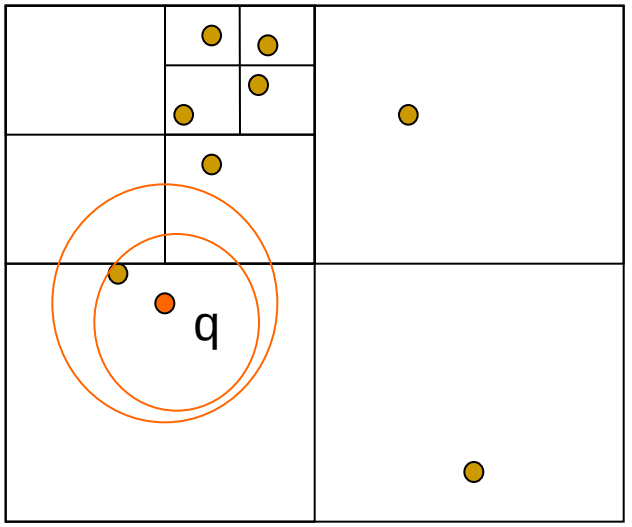




Quadtree – Nearest Neighbor Query



Quadtree – Nearest Neighbor Search



Algorithm

Initialize range search with large r

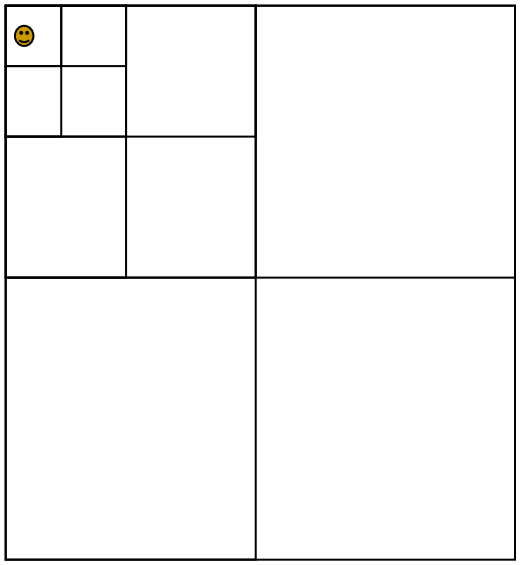
Put the root on a stack

Repeat

- Pop the next node *T* from the stack
- For each child *C* of *T*
  - if *C* intersects with a circle (ball) of radius *r* around *q*, add *C* to the stack
  - if *C* is a leaf, examine point(s) in *C* and update (boost) r

- Whenever a point is found, update r (i.e., current minimum)
- Only investigate nodes with respect to current r.

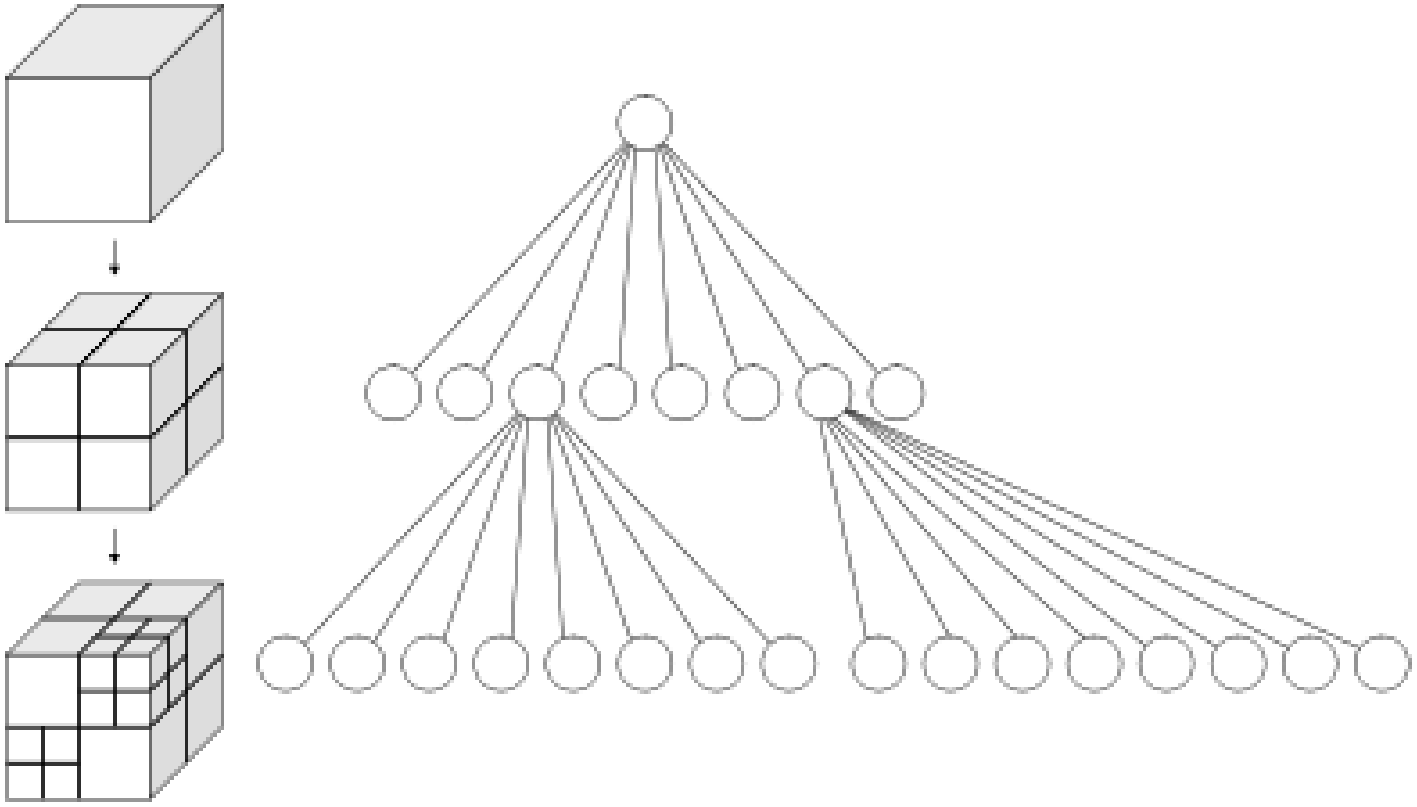
Quadtree (cont'd)



- Simple data structure.
- Easy to implement.
- But, it might not be efficient for skew data:
  - A quadtree could have a lot of empty cells.
  - If the points form sparse clouds, it takes a while to reach nearest neighbors.
- No guaranteed complexity for bad (zipfian or power law) distributions
- Guaranteed complexity for uniform distributions only!!!

$$S(n)=O(n),\ T(n)=O(\log_4 n+ k)$$

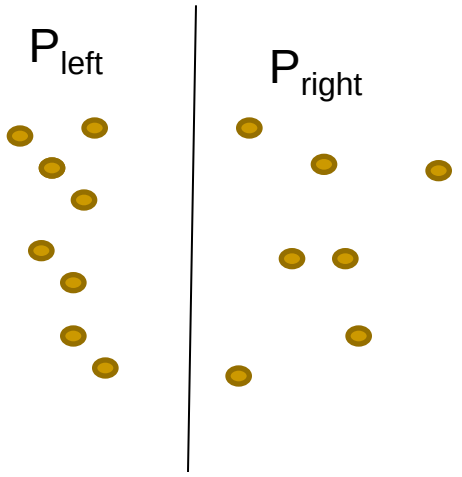
Oct Tree for 3D

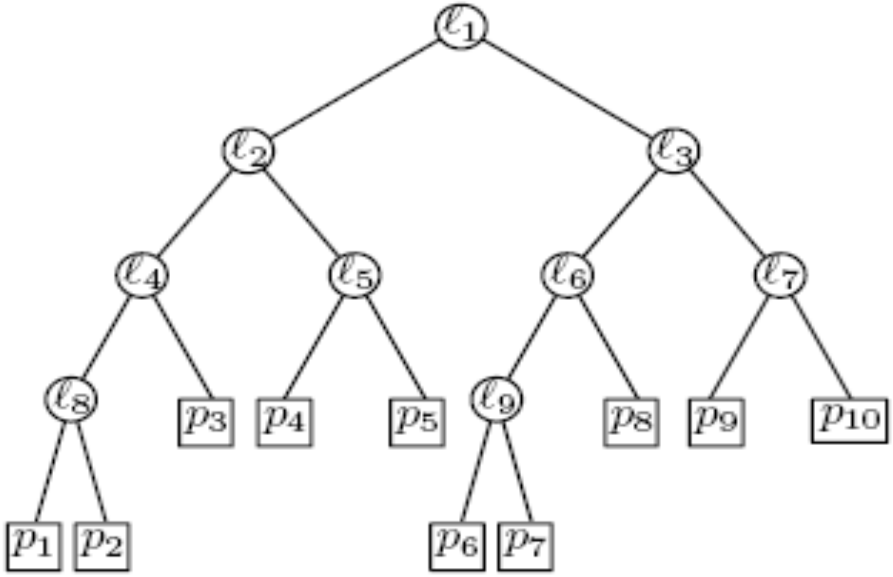


Left: Recursive subdivision of a cube into octants.
Right: The corresponding octree.

KD Tree (NN search – Range Search)

- Data stored at the leaves
- Every node (except leaves) represents a hyperplane that divides the space into two parts.
- Points to the left (right) of this hyperplane represent the left (right) sub-tree of that node.





KD Tree

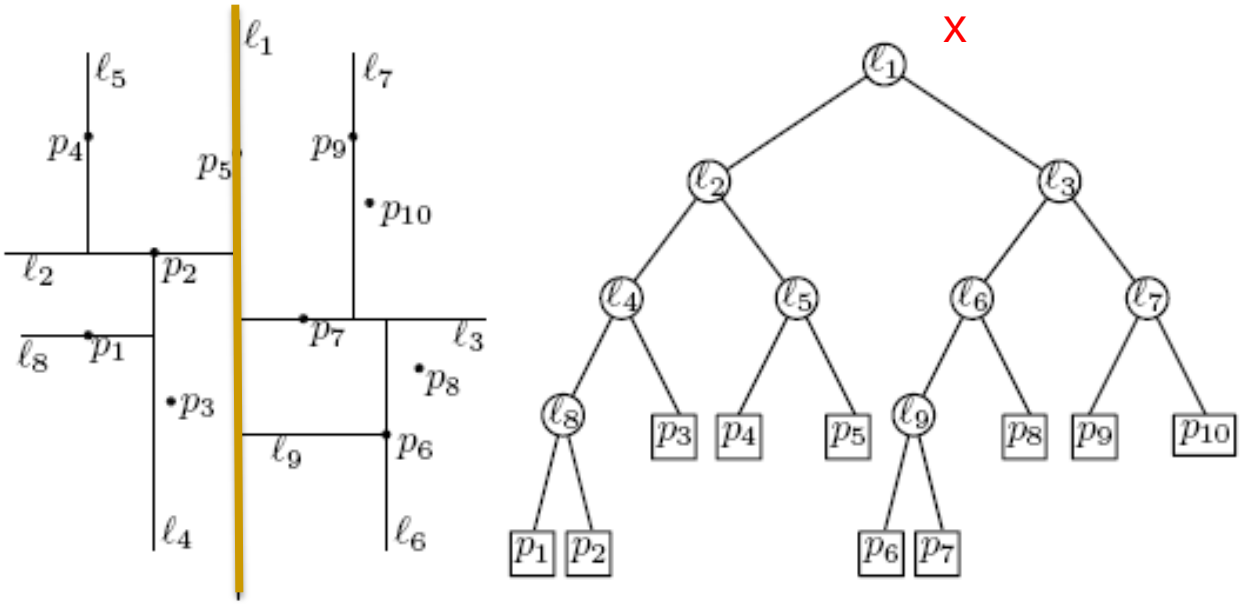
As we move down the tree, we divide the space along alternating (but not always) axis-aligned hyperplanes:

Split by x-coordinate*:* split by a vertical line that has (ideally) half the points left or on, and half right.

Split by y-coordinate*:* split by a horizontal line that has (ideally) half the points below or on and half above.

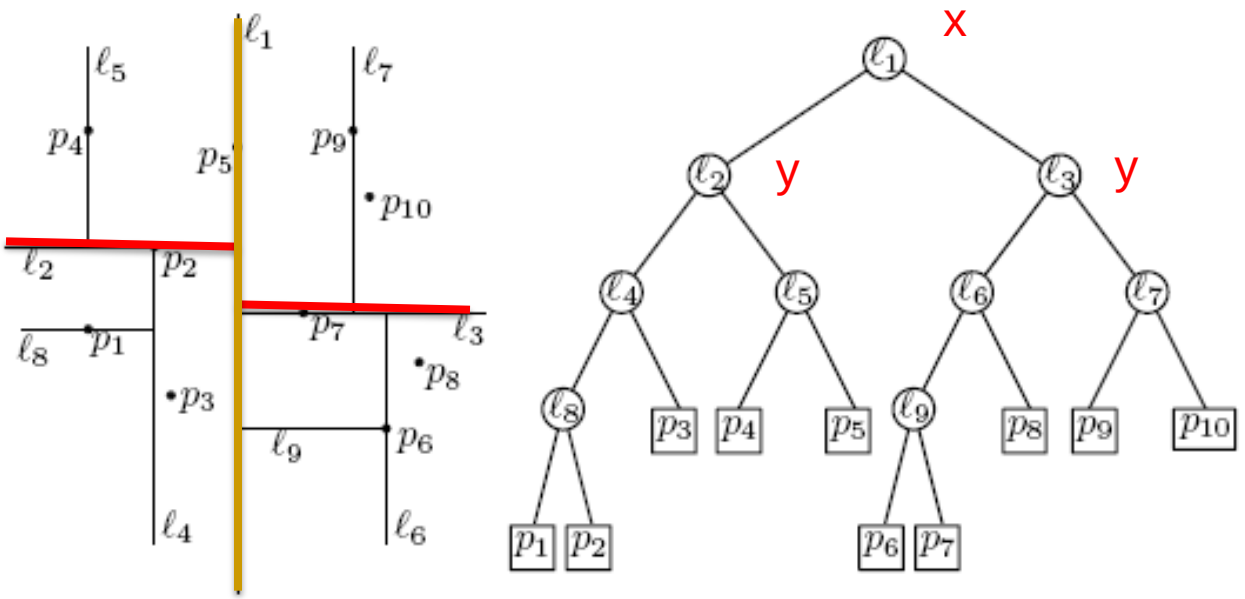
KD Tree - Example

Split by x-coordinate*:* split by a vertical line that has approximately half the points left or on, and half right.



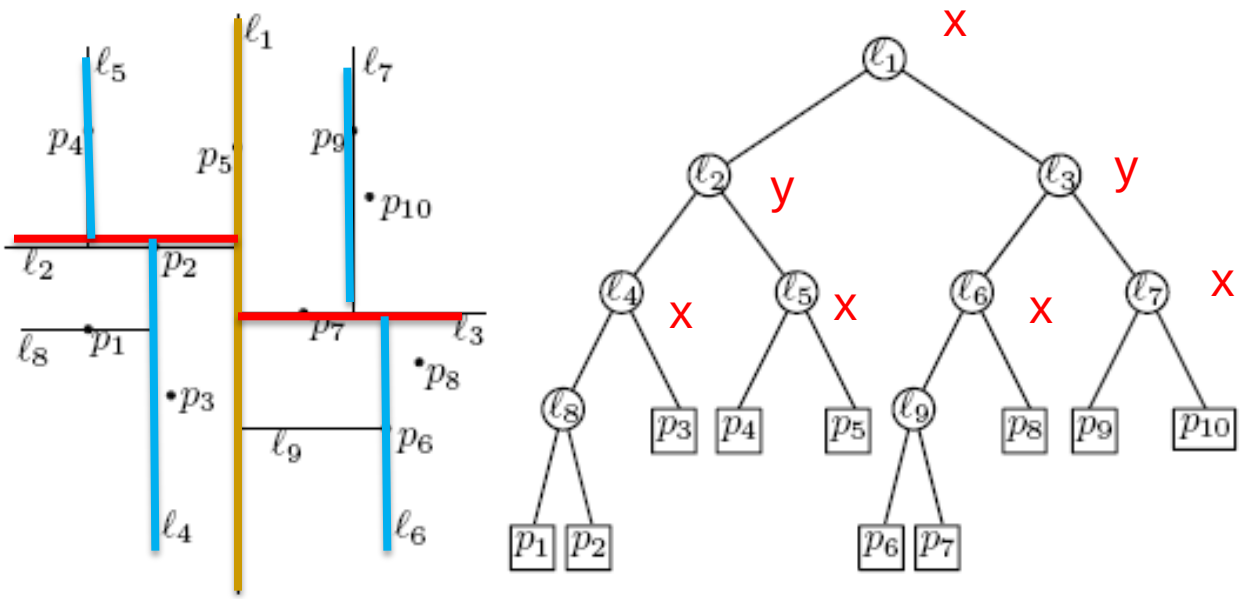
KD Tree - Example

Split by y-coordinate: split by a horizontal line that has half the points below or on and half above.



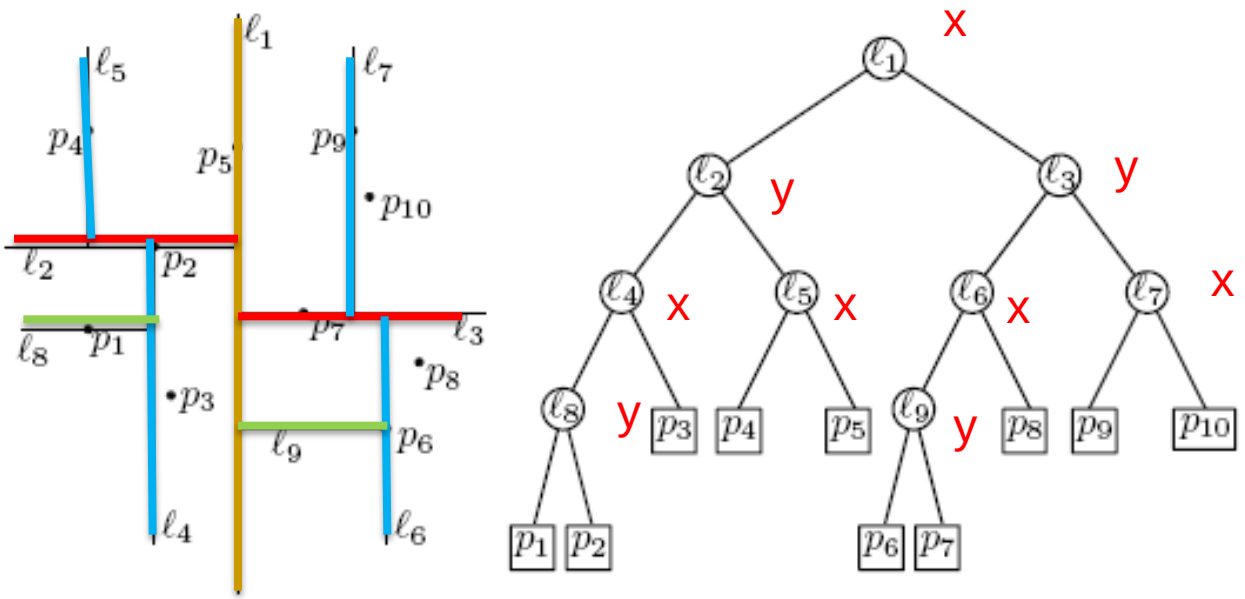
KD Tree - Example

Split by x-coordinate: split by a vertical line that has half the points left or on, and half right.



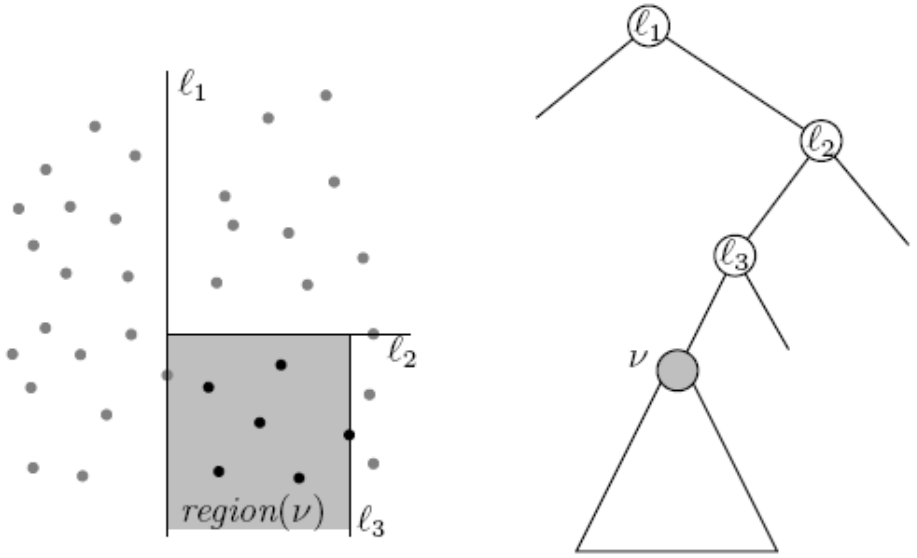
KD Tree - Example

Split by y-coordinate: split by a horizontal line that has half the points below or on and half above.



KD Tree – **Region of a node**

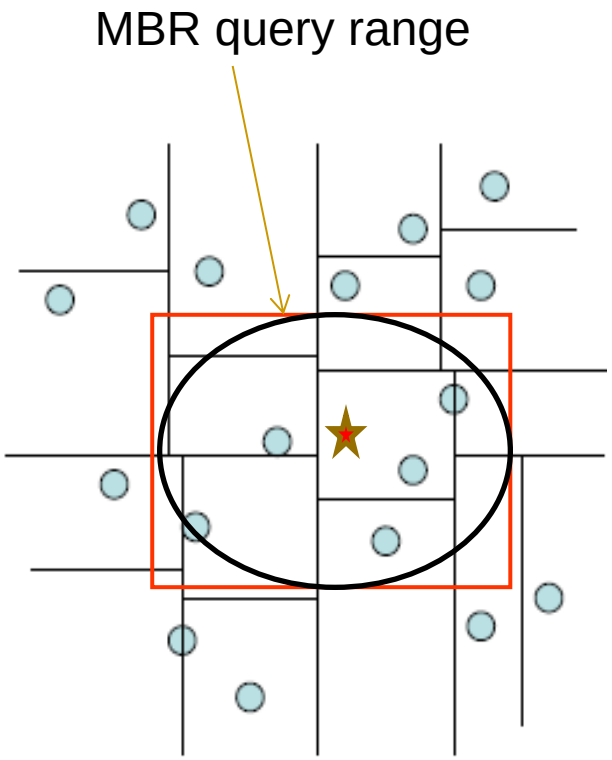
- The region *region(v)* corresponding to a node v is a rectangle, which is bounded by splitting lines stored at ancestors of v.



***A point is stored in the subtree rooted at node v if and only if it lies in *region(v)*.

KD Trees – MBR Range Search

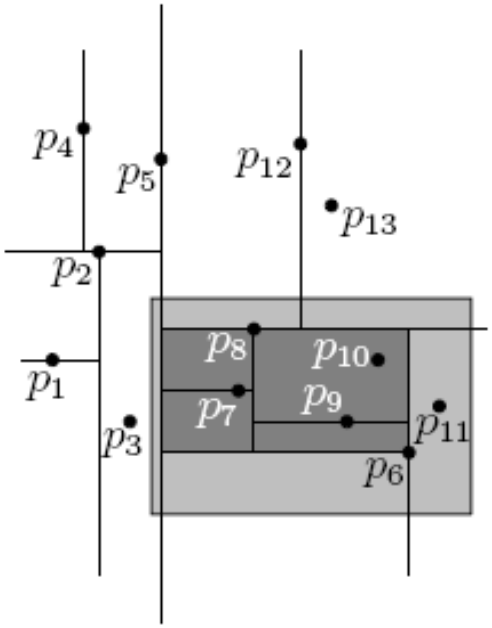
- Need only search nodes whose region intersects query region.
  - Report all points in subtrees whose regions are entirely contained in query range.
  - If a region is partially contained in the query range check points.

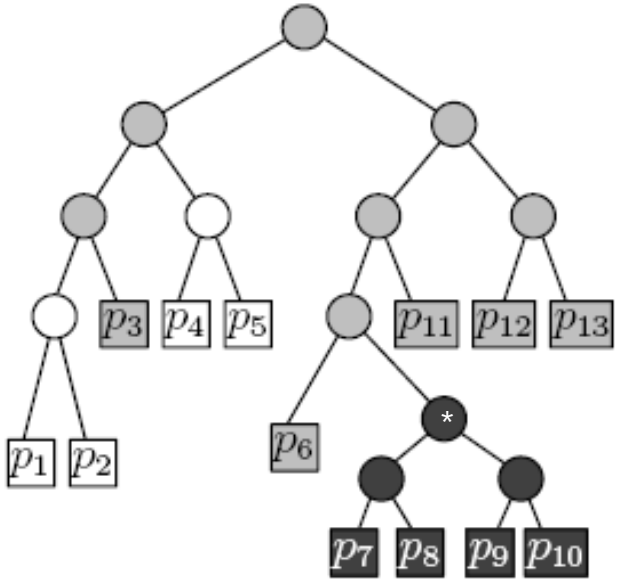


Example – MBR Range Search

Query region: gray rectangle

Gray nodes are the nodes visited in this example.

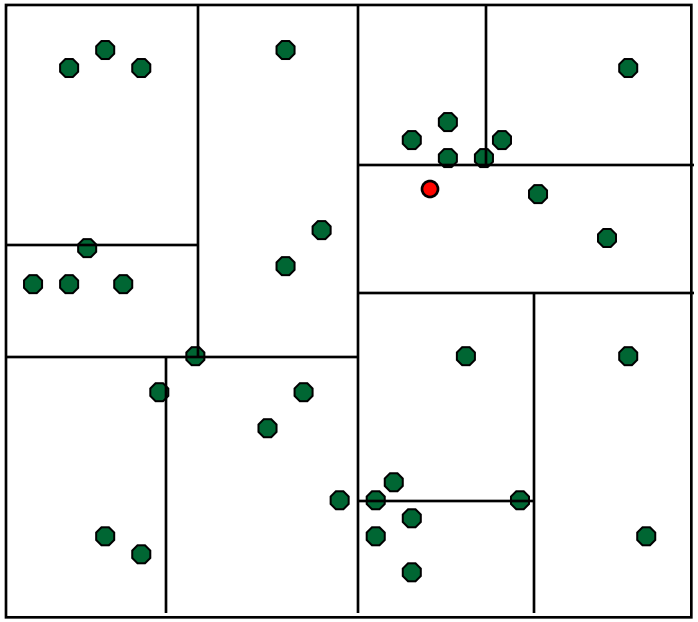


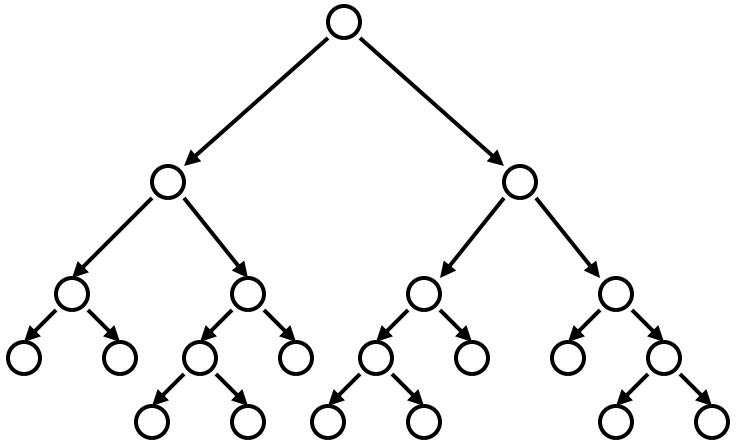


1.Node marked with * corresponds to a region that is entirely inside the query rectangle. Report **all** leaves in this subtree.
2.All other nodes visited (i.e., gray) correspond to regions that are only partially inside the query rectangle.

- Report points p_6 and p_{11} only
- Do not report points p_3, p_{12} and p_{13}

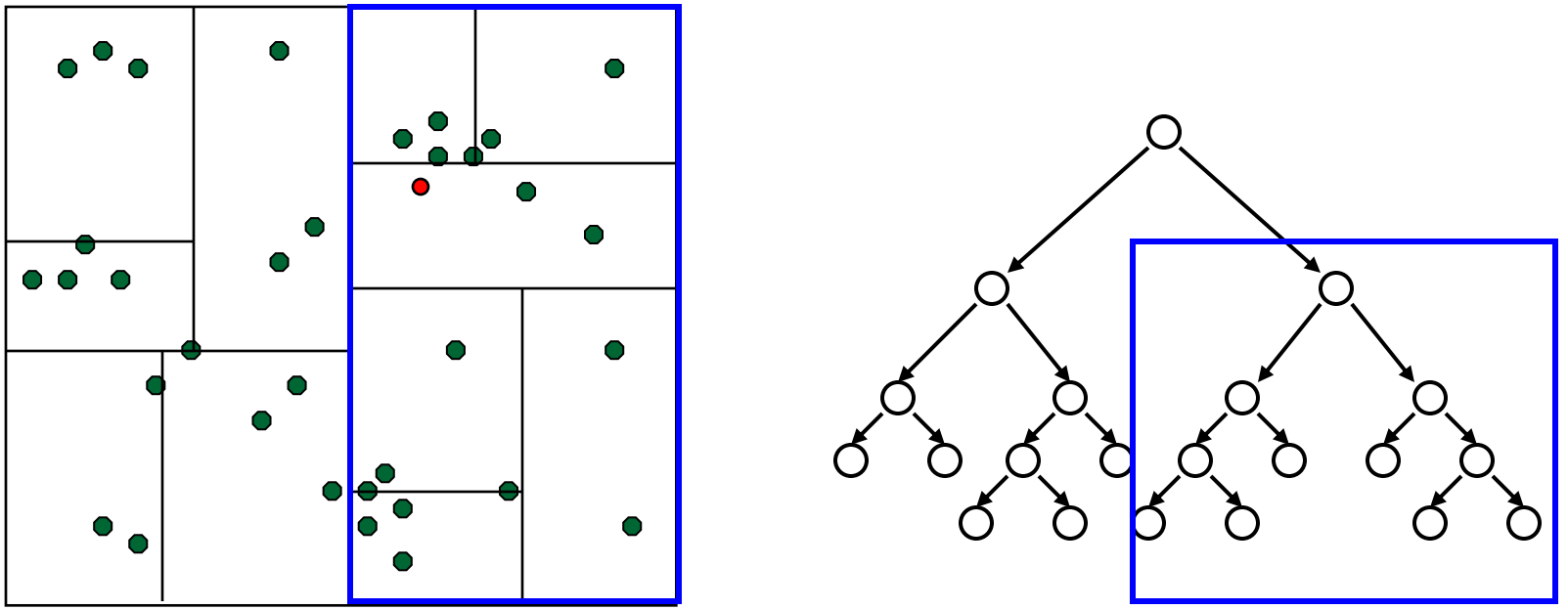
Nearest Neighbor with KD Trees





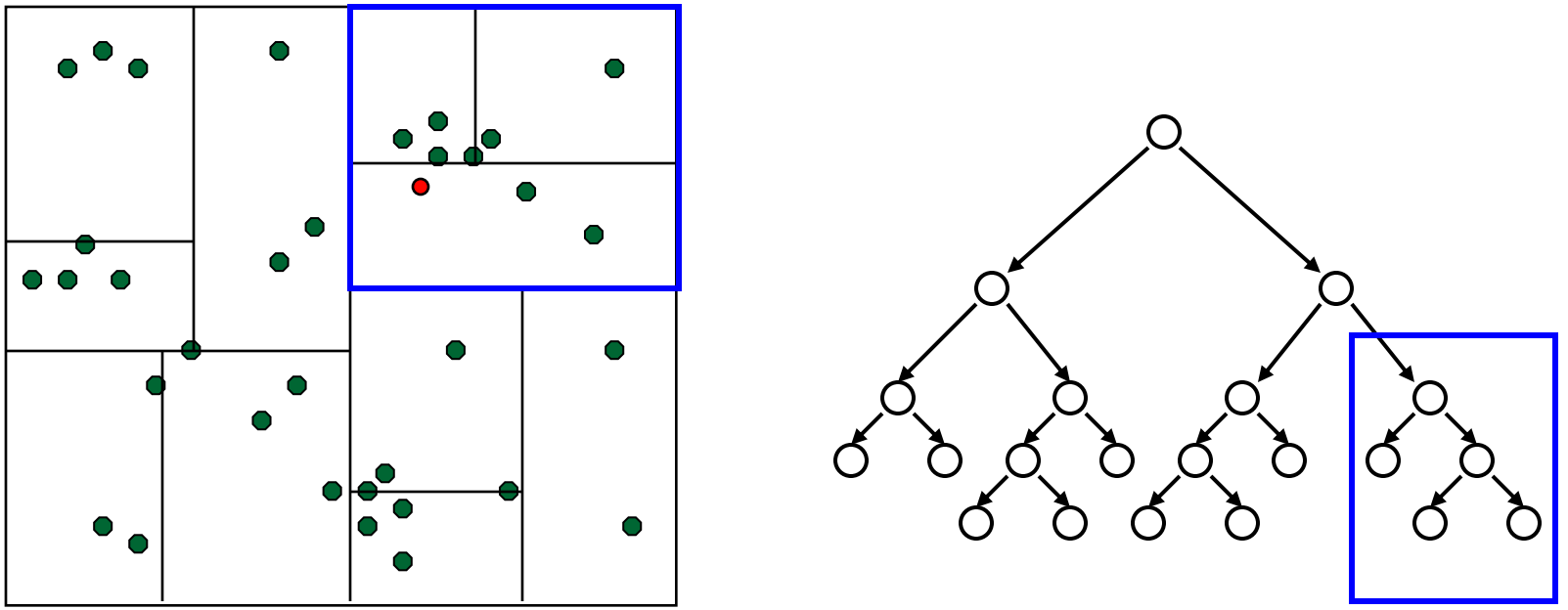
Traverse the tree, looking for the rectangle that contains the query.

Nearest Neighbor with KD Trees



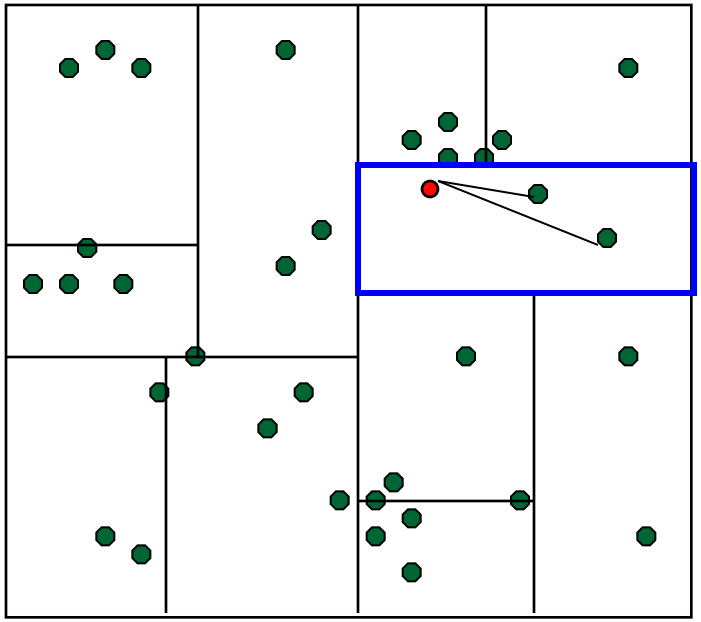
Explore the branch of the tree that is closest to the query point first.

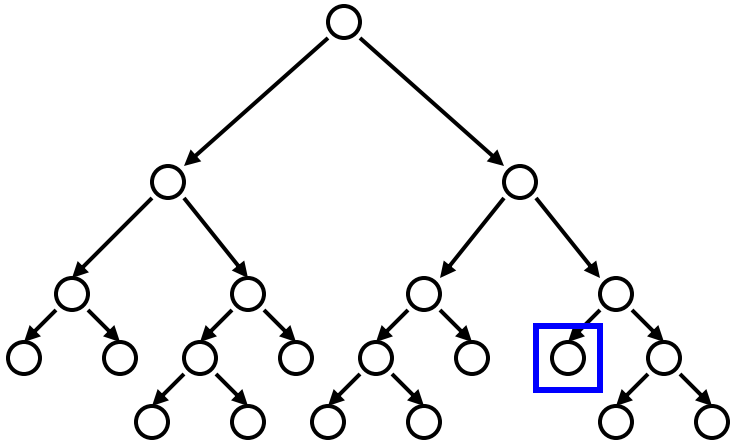
Nearest Neighbor with KD Trees



Explore the branch of the tree that is closest to the query point first.

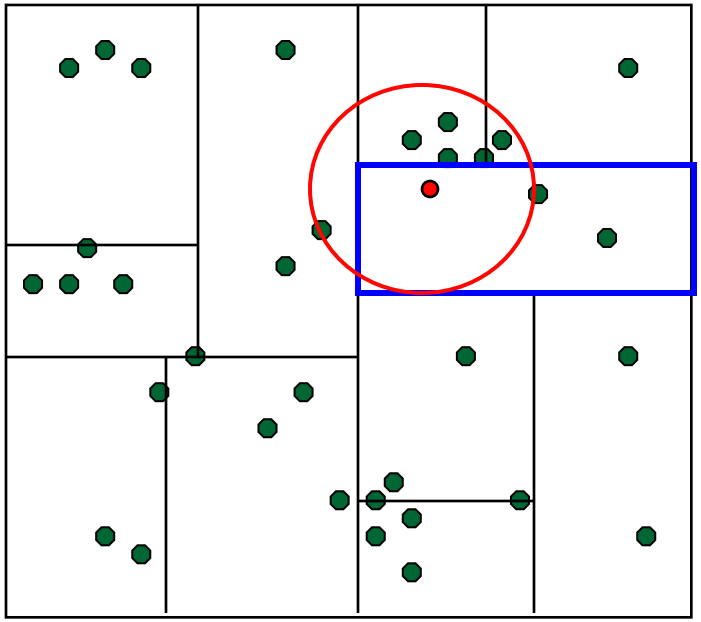
Nearest Neighbor with KD Trees

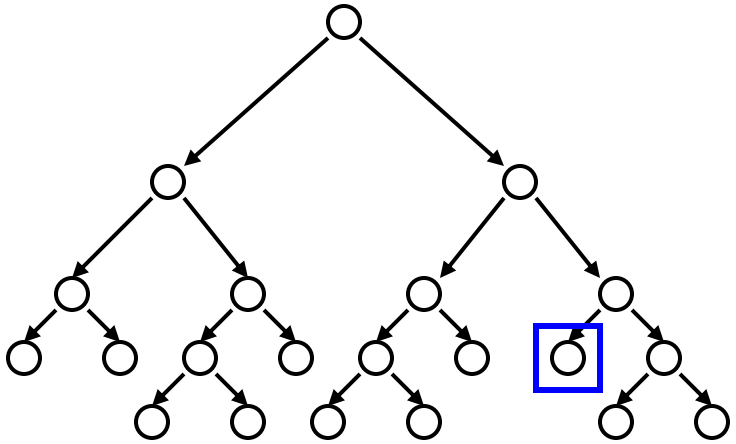




When we reach a leaf, compute the distance to each point in the node.

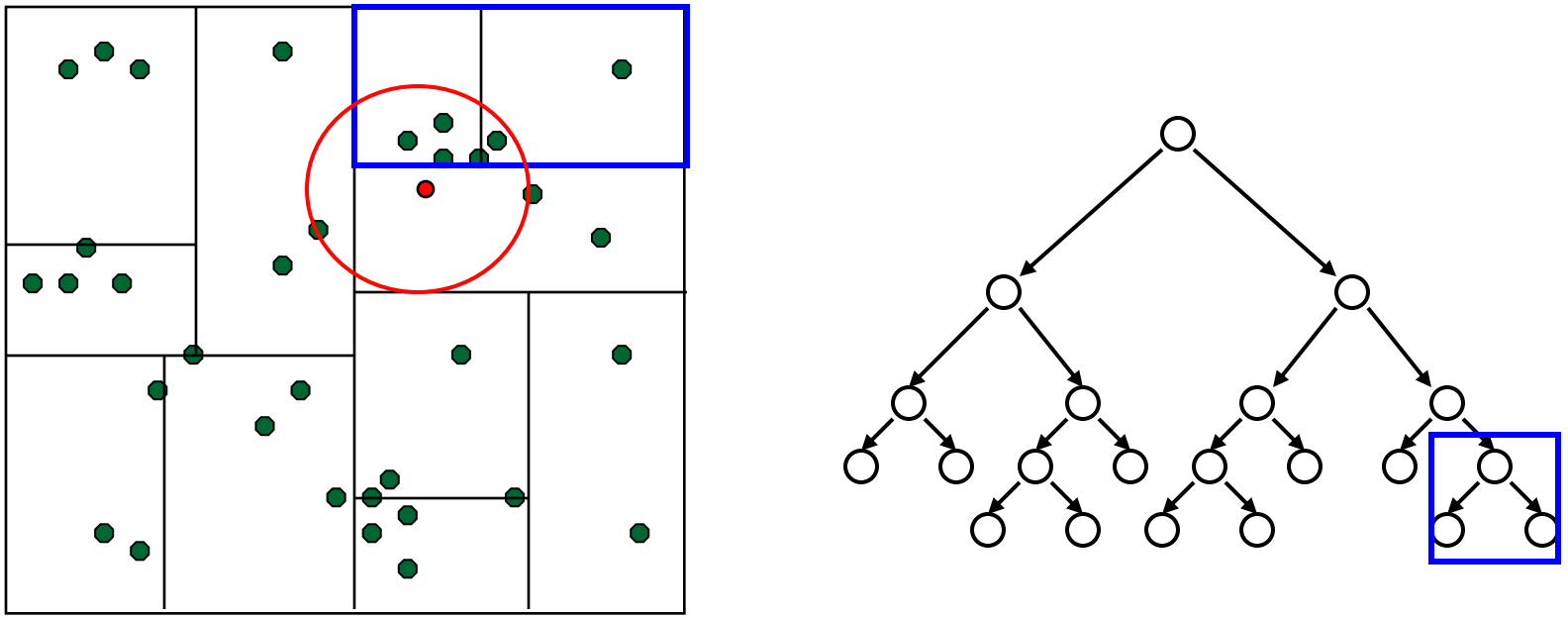
Nearest Neighbor with KD Trees





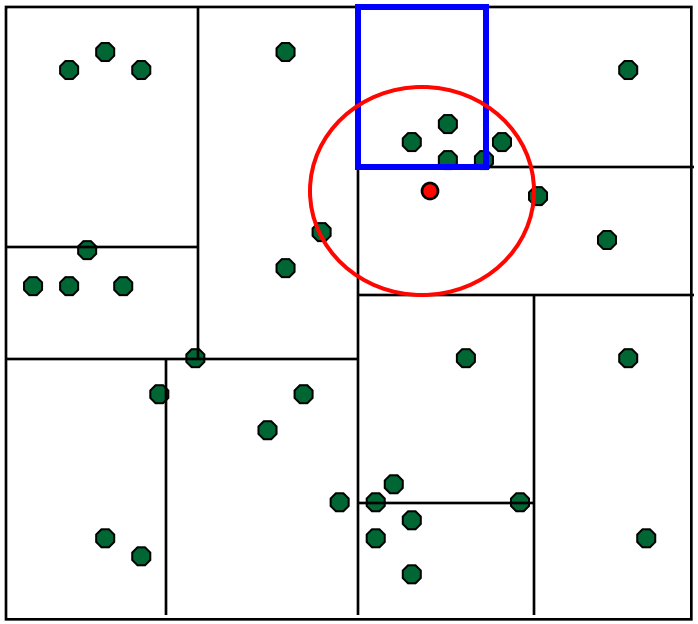
When we reach a leaf, compute the distance to each point in the node.

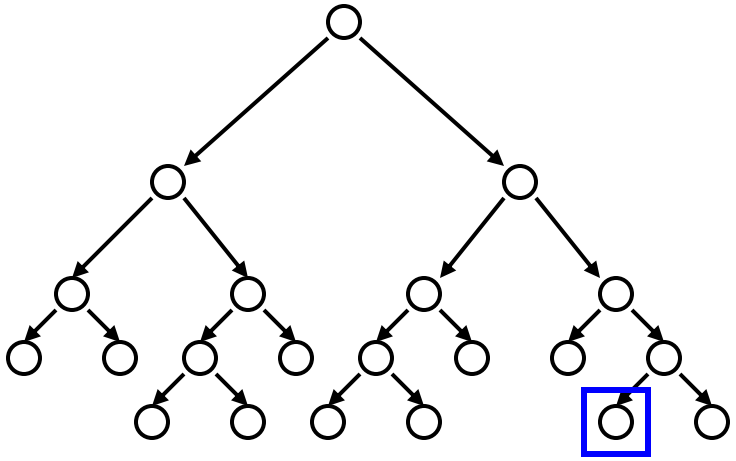
Nearest Neighbor with KD Trees



Then, backtrack and try the other branch at each node visited.

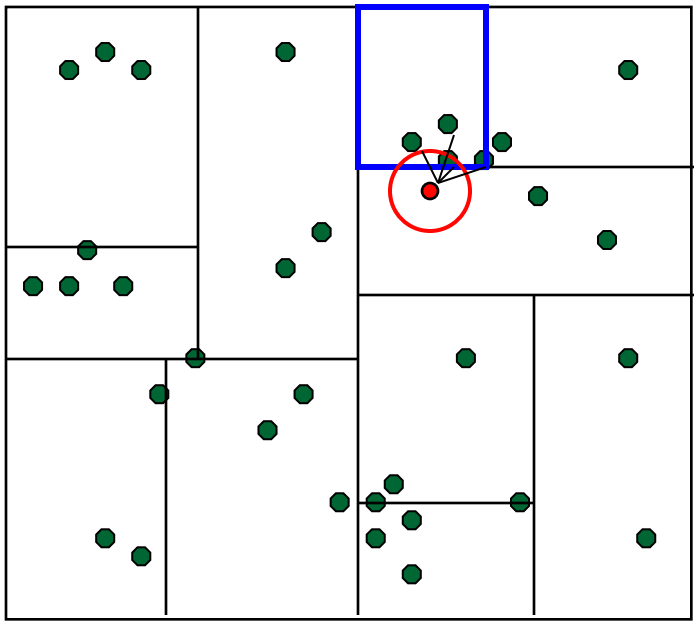
Nearest Neighbor with KD Trees

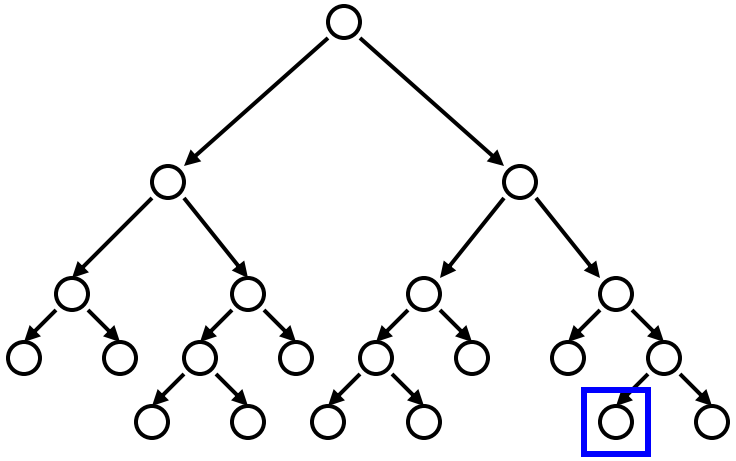




Each time a new closest node is found, we can update the distance bounds.

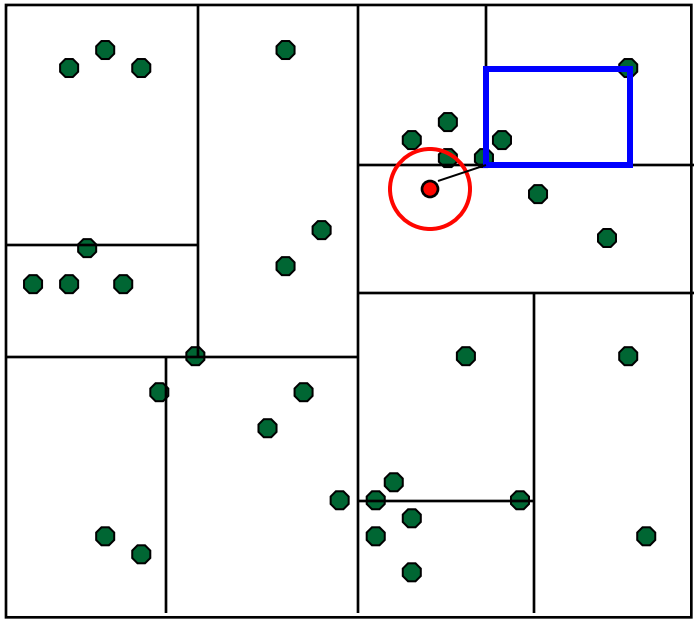
Nearest Neighbor with KD Trees

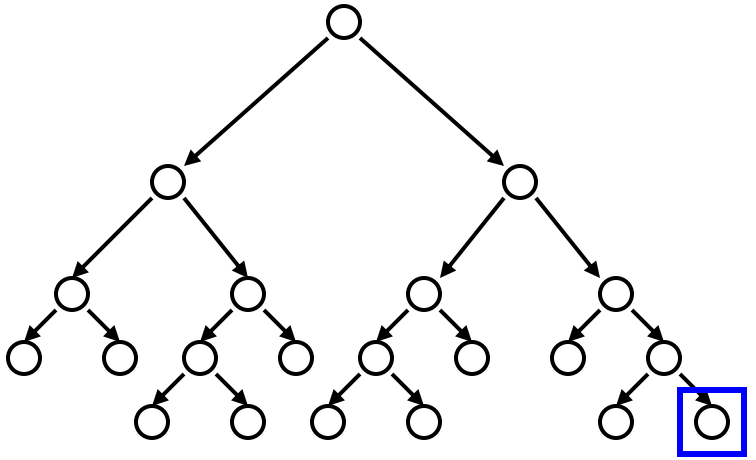




Each time a new closest node is found, we can update the distance bounds.

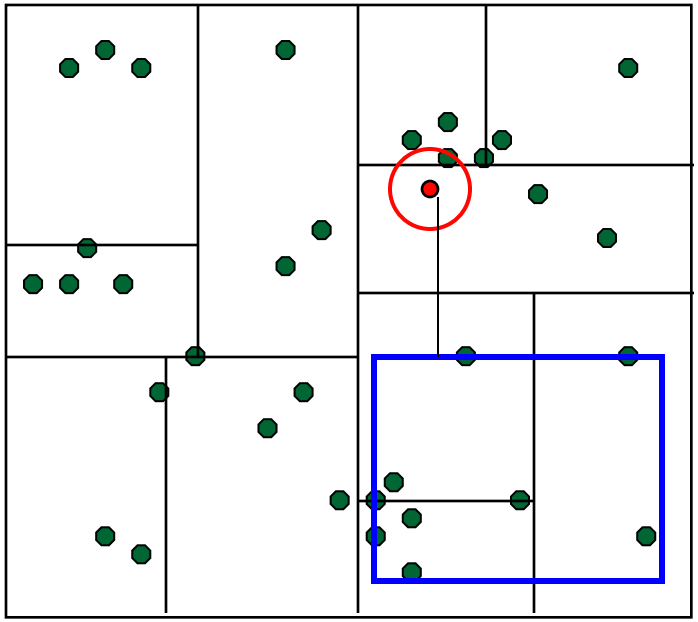
Nearest Neighbor with KD Trees

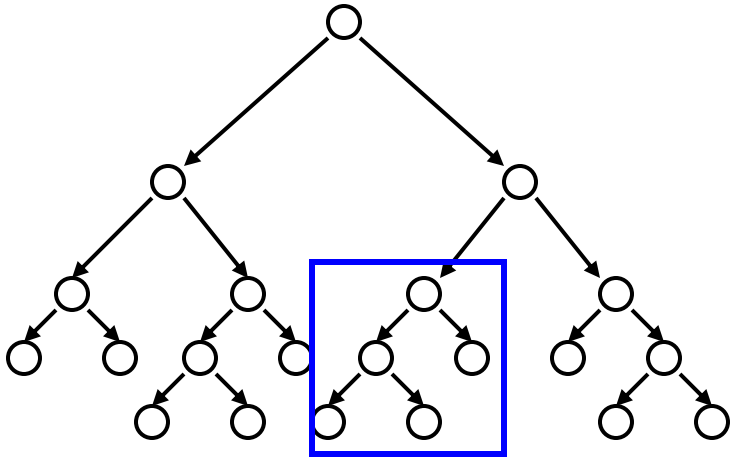




Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Nearest Neighbor with KD Trees



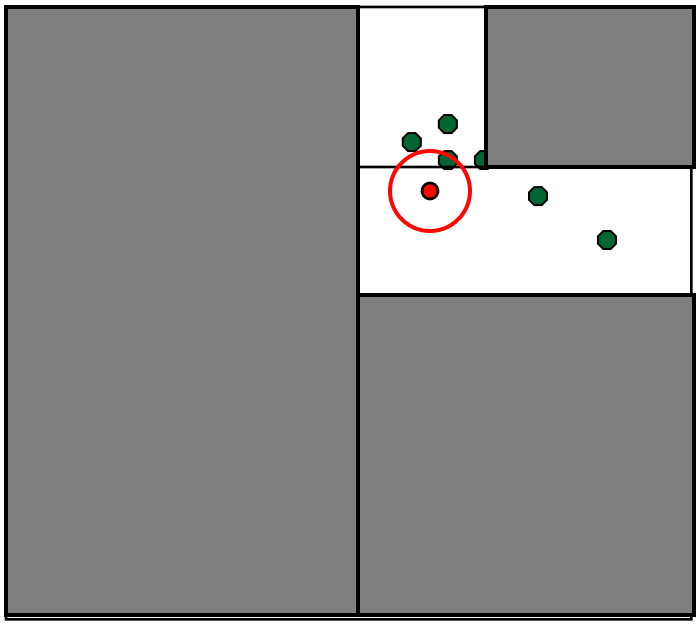


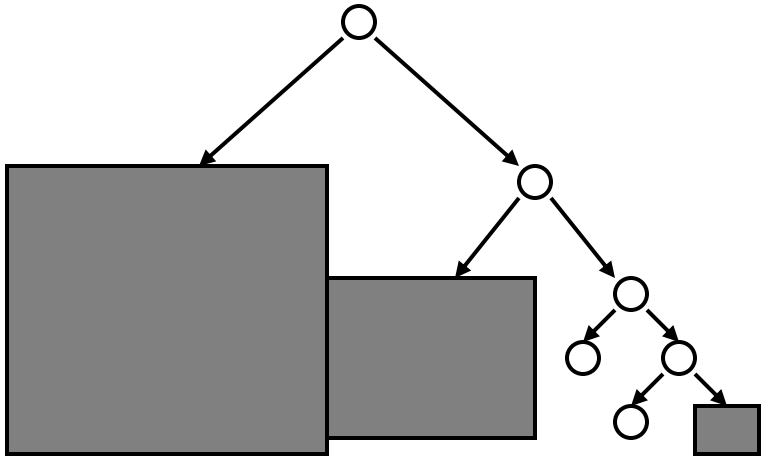
Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Nearest Neighbor with KD Trees

Space requirements: O(n)

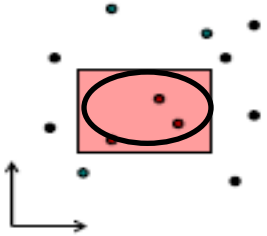
Query requirements: $O(n^{1-1/d} + k)$



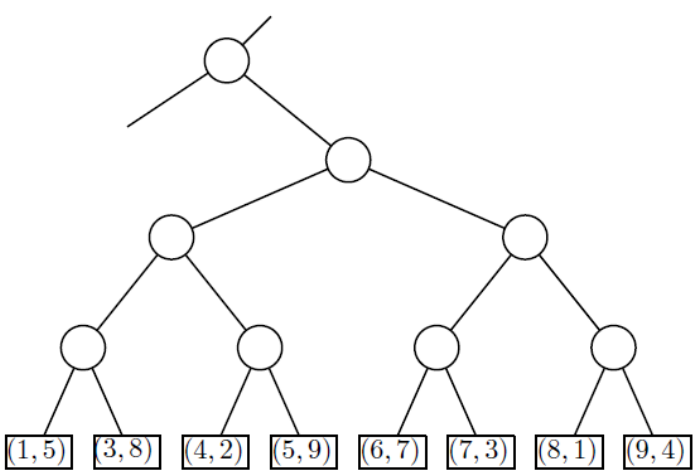


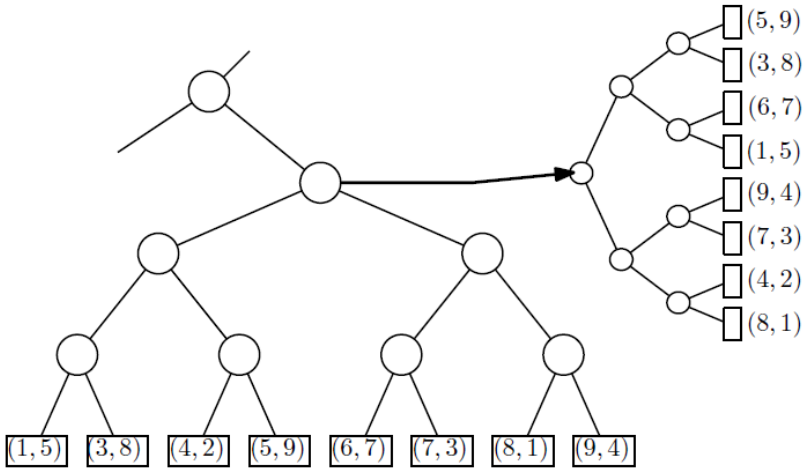
Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

2D MBR Range Tree

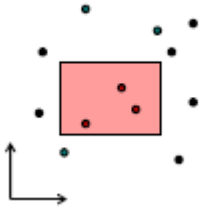


- Store a ***primary 1D range tree*** for all the points based on *x-coordinate.*
- For each node, store a ***secondary 1D range tree*** based on *y-coordinate*.

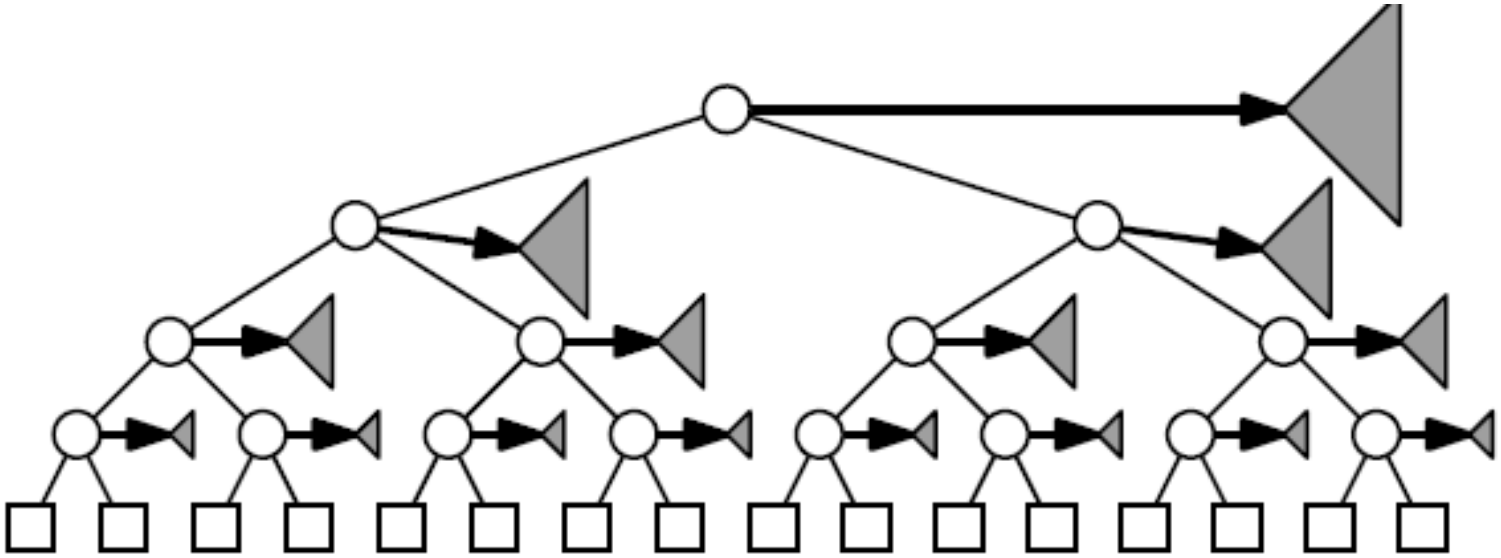




2D MBR Range Search

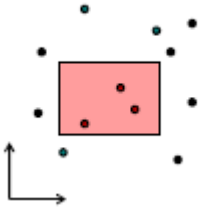


Range Tree

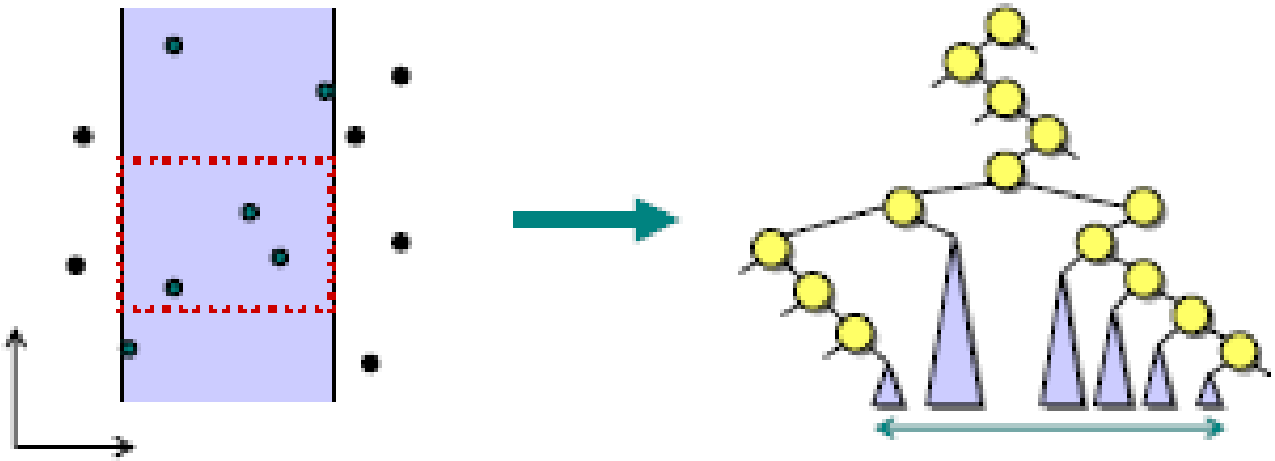


Space requirements: O(nlogn)

2D Range Search (cont'd)

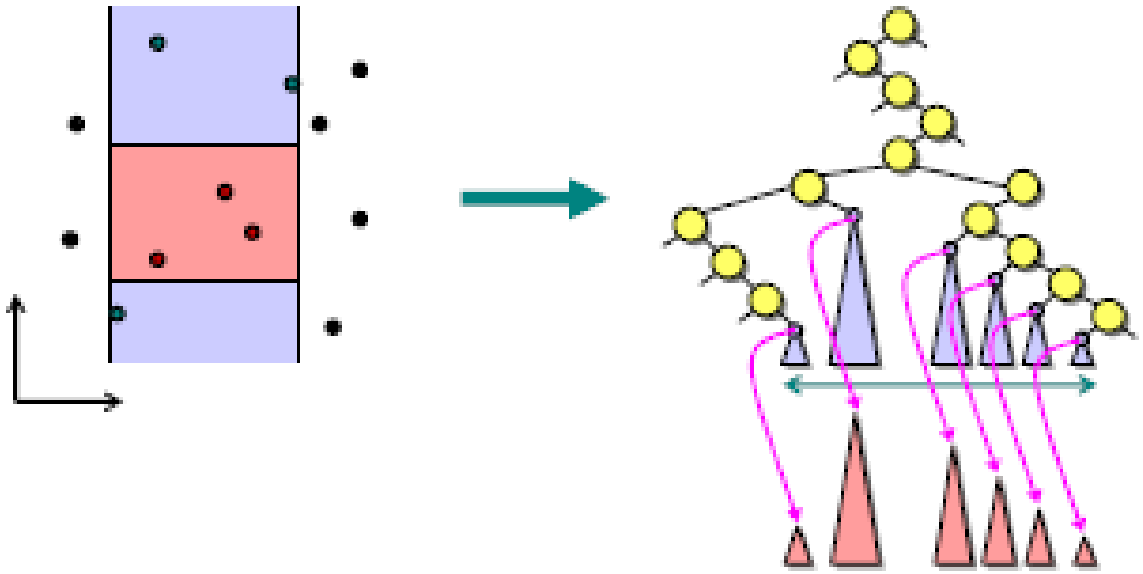


- Search using the x-coordinate only.
- How to restrict to points with proper *y-coordinate?*



2D Range Search (cont'd)

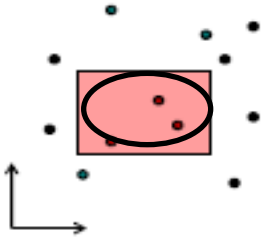
- Recursively search within each subtree using the y-coordinate.



- Query cost: $O(\log^2 n+k)$

MBR Range Search in d dimensions +Filtering on the fly

1D query time: O(logn + k)



2D query time: $O(\log^2 n + k)$

d dimensions:

Query time: $O(k + \log^d n)$ to report k points.
Space: $O(n \log^{d-1} n)$

Fractional Cascading Technique achieves better query time complexity:
$O(k+\log^{d-1} n)$ + filtering on the fly
If needed (less than k neighbors have been reported)
Boost R (or expand the MBR)

Big Data Version: Efficient processing of all-k-nearest-neighbor (aknn) queries in the Map-Reduce programming framework

- Numerous modern applications, like Geographical Information Systems (GIS), facilities management, location-based services, smart-cities services, etc., need efficient processing of queries on big spatial data.
- One such query is the All k Nearest Neighbor Query (AkNNQ, or kNN Join).
- Given two point datasets, Q (Query) and T (Training), the AkNNQ finds the k nearest neighbors of T for each point of Q, according to a certain distance metric.

kNN vs AkNN

Definition 1 (*kNN*)**.** Given a point p, a dataset S and an integer k, the k nearest neighbors of p from S, denoted as $kNN(p, S)$, is a set of k points from S such that, $\forall r \in kNN(p, S)$, $\forall q \in S - kNN(p, S)$, $dist(p, r) \leq dist(p, q)$.

Definition 2 (*AkNNQ*)**.** Given two datasets R and S and an integer k, the result of the All k Nearest Neighbors Query of R from S, denoted as $AkNNQ(R, S)$, is the set of pairs $\{(r, s) : r \in R, s \in kNN(r, S)\}$.

A naive approach to find the k nearest neighbors of two datasets would be to calculate the distances of every point of the one dataset, R, to every point of the other dataset, S, and sort the results by distance. Of course this would be highly inefficient as it would lead to a huge number of calculations, in the order of $O(|R| \times |S|)$.

akNN processing….

Decomposition of data space into small equal hyper-cells and afterwards the merging of some neighboring cells, if they do not contain k points, or more in total…..

For every partition R_i ([∪ R_i = R), find a corresponding partition S_j ([∪ S_j = S).

$kNN(R_i\ join\ S) = kNN(R_i\ join\ S_J)$ and
$kNN(R\ join\ S) = \cup\ kNN(R_i\ join\ S_J)$

R=Q=I
S=T

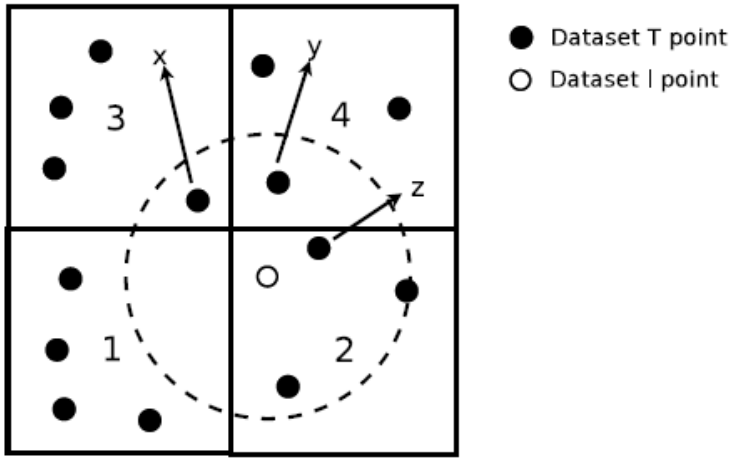
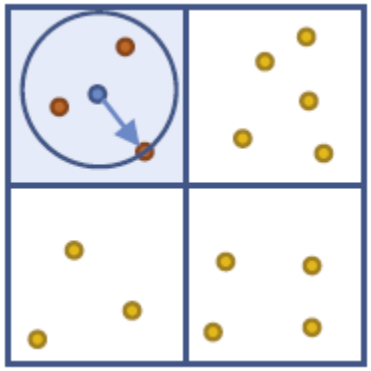
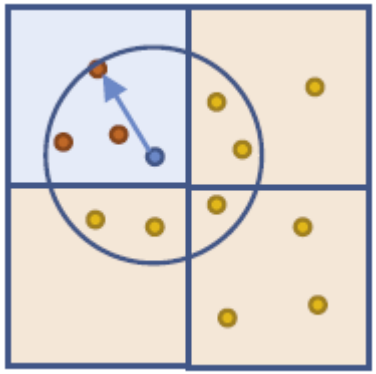
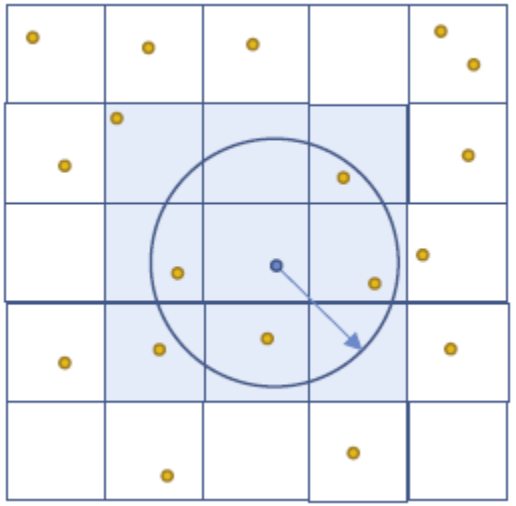


Figure 1: kNN process using cell decomposition ($k = 3$)

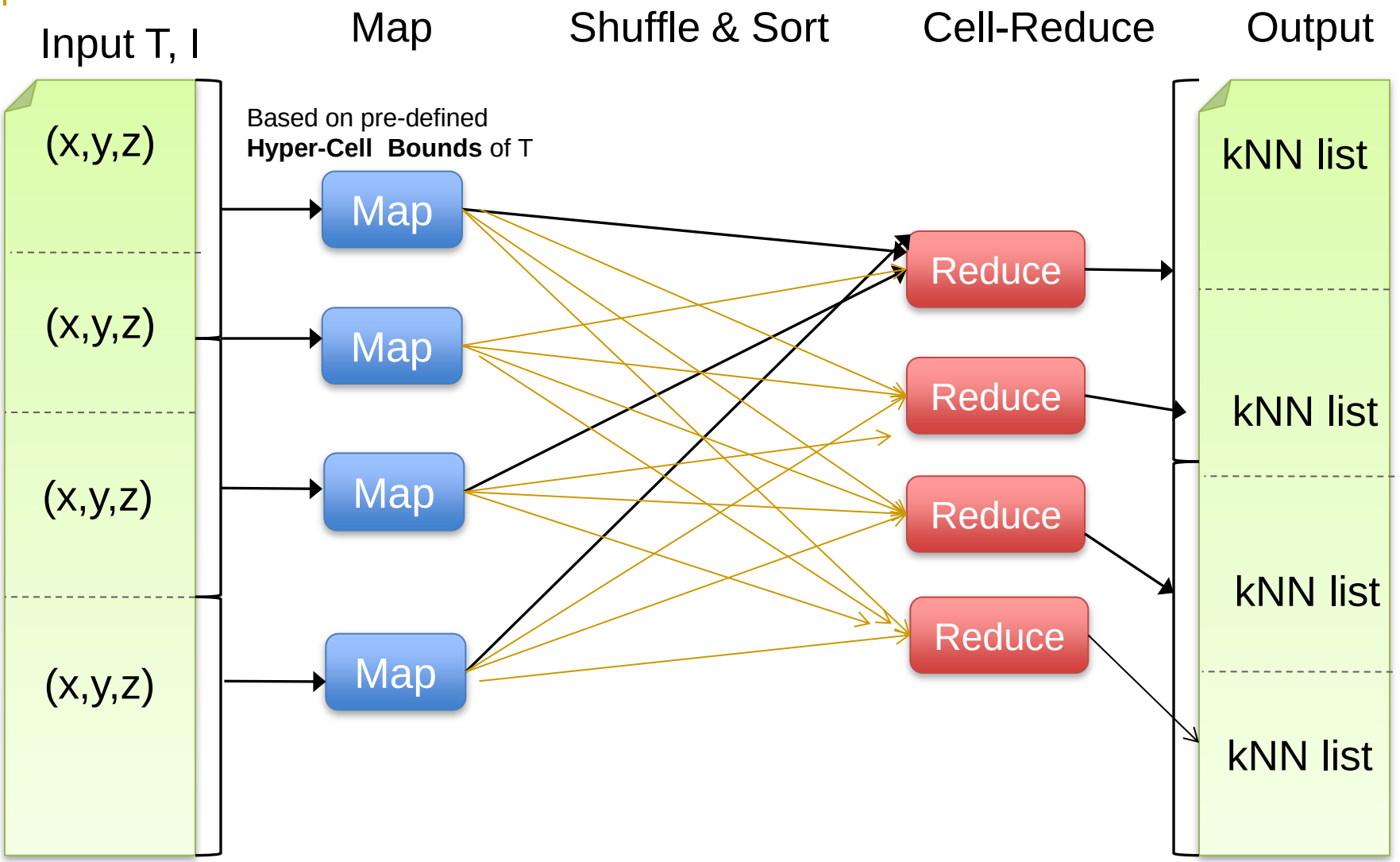
akNN processing….

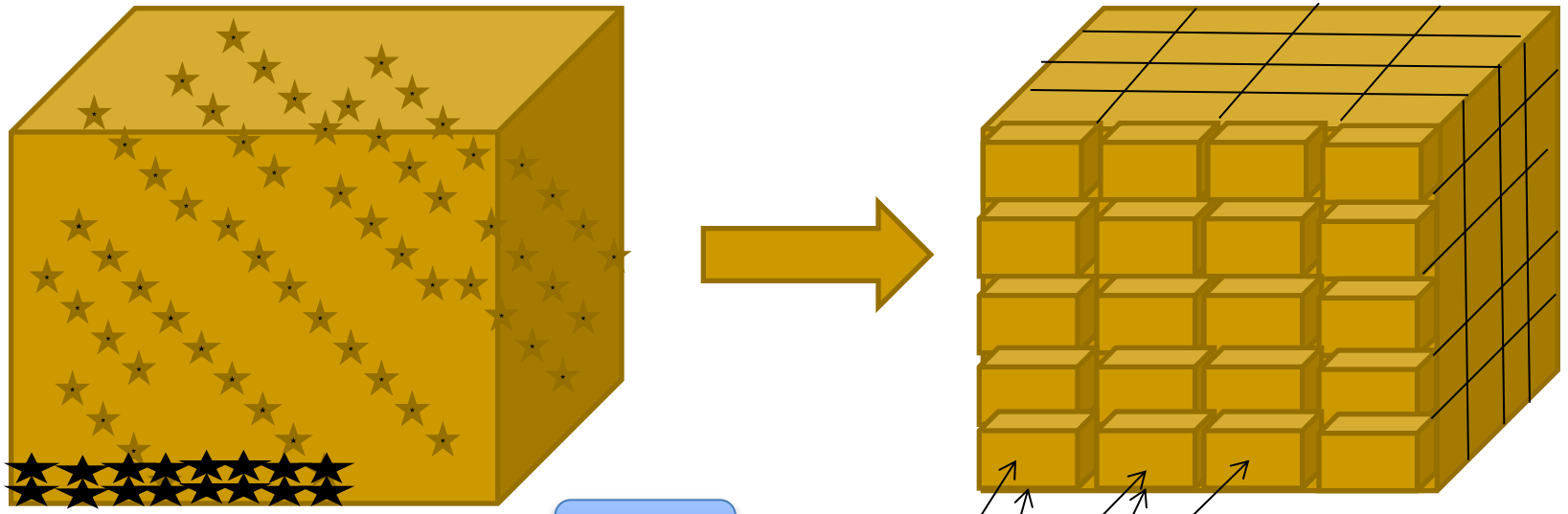






Map-Reduce AkNN Computation (Simple Approach with 4 cells)





Map1

(id1,x1,y1,z1) → (id1,x1,y1,z1, C000)

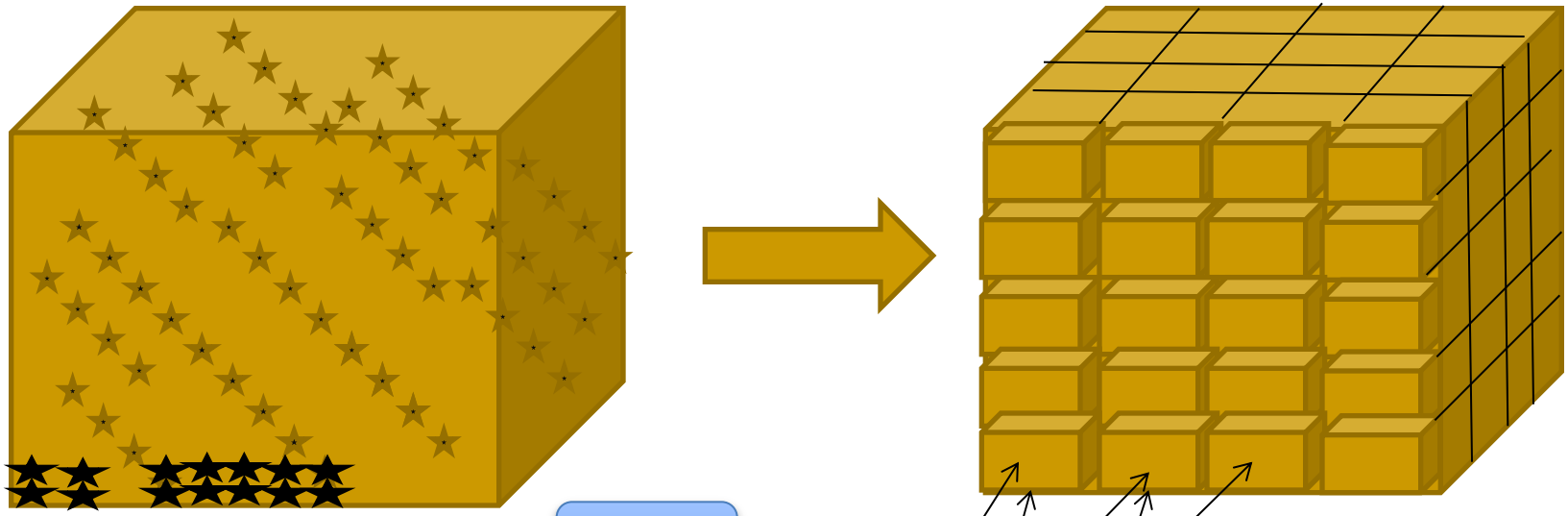
(id2, x2,y2,z2) → (id2, x2,y2,z2, C100)

(id3,x3,y3,z3) → (id3, x3,y3,z3, C200)

(id4, x4,y4,z4) → (id4,x4,y4,z4, C000)

(id5, x5,y5,z5) → (id5, x5,y5,z5,C100)

……



Map2

(id6,x6,y6,z6) → (id6,x6,y6,z6, C000)

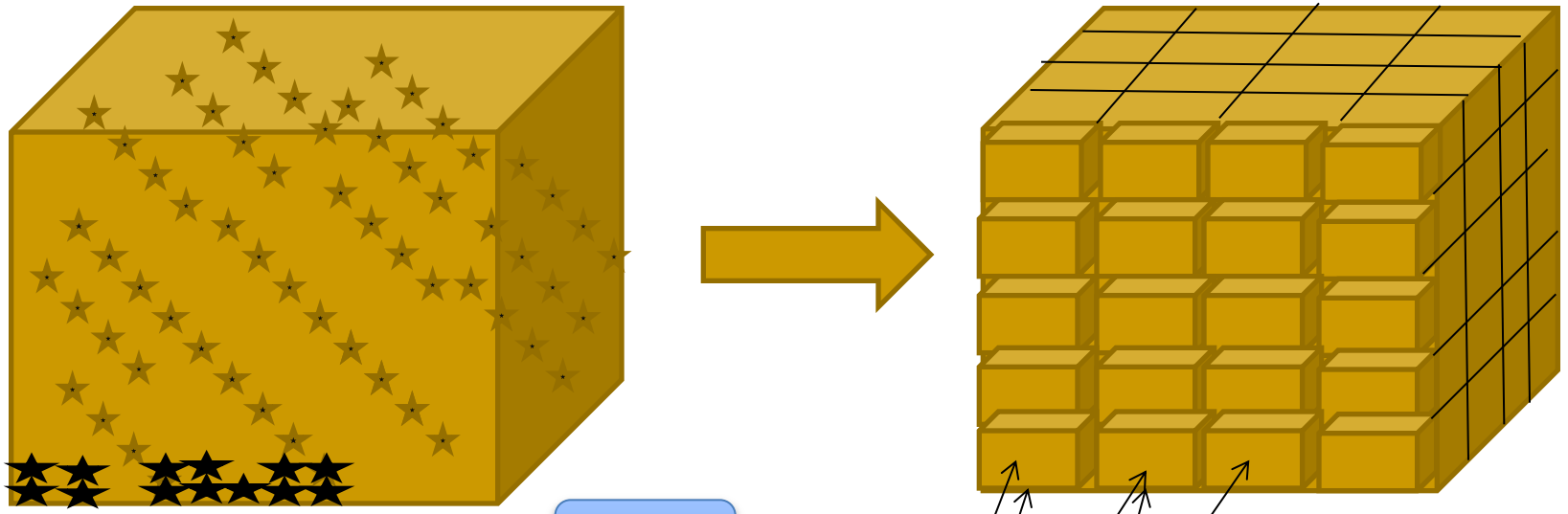
(id7, x7,y7,z7) → (id7, x7,y7,z7, C100)

(id8,x8,y8,z8) → (id8, x8,y8,z8, C200)

(id9, x9,y9,z9) → (id9,x9,y9,z9, C000)

(id10, x10,y10,z10) → (id10, x10,y10,z10,C100)

……



Map3

(id11,x11,y11,z11) → (id11,x11,y11,z11, C000)

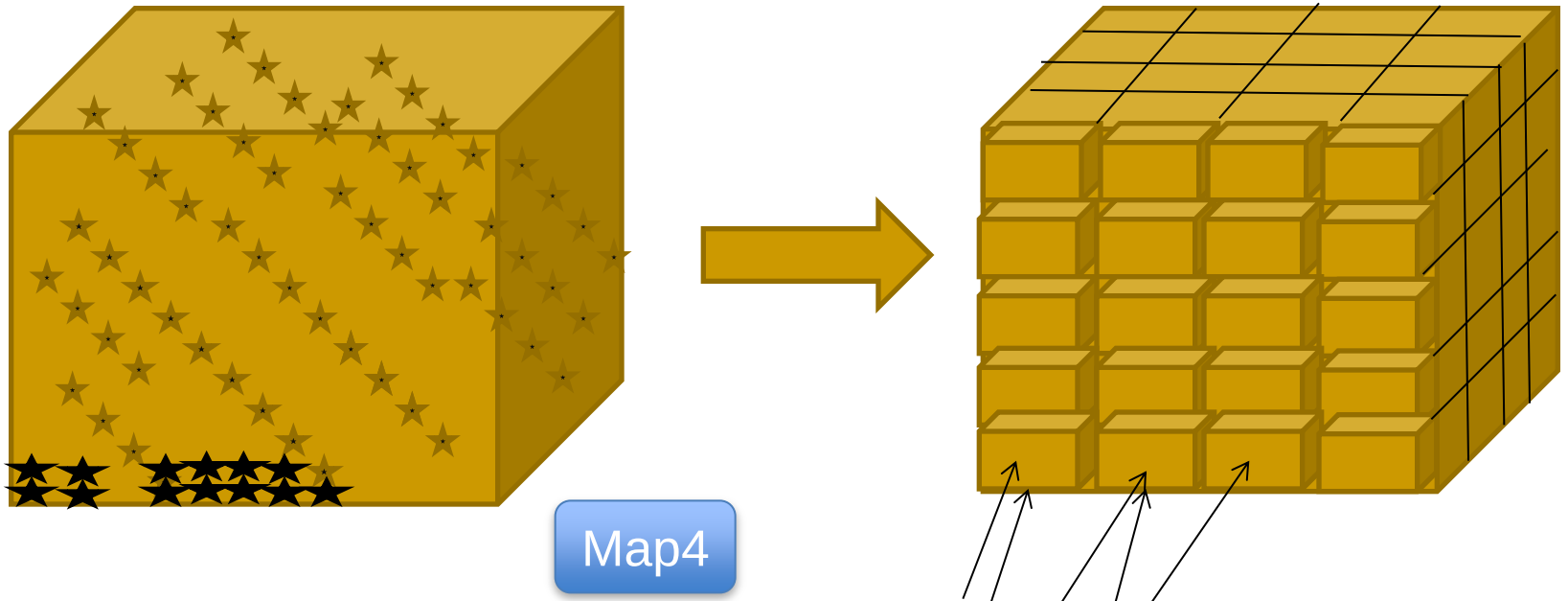
(id12, x12,y12,z12) → (id12, x12,y12,z12, C100)

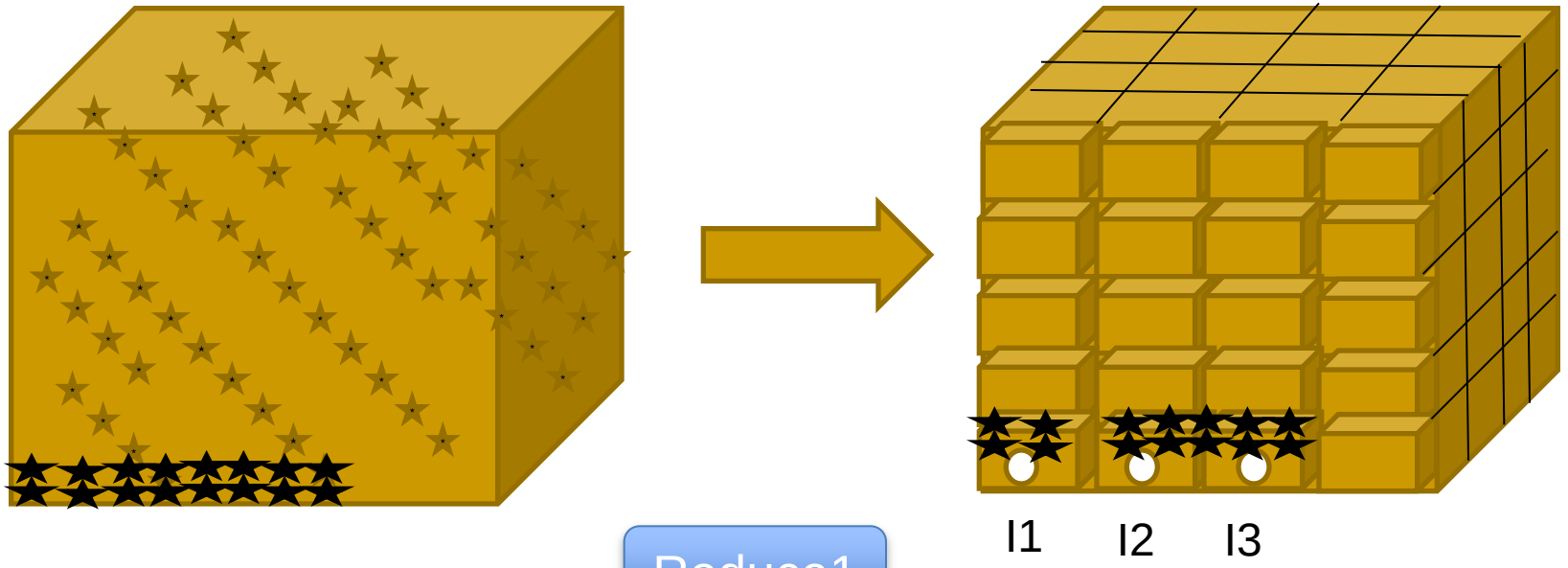
(id13,x13,y13,z13) → (id13, x13,y13,z13, C200)

(id14, x14,y14,z14) → (id14,x14,y14,z14, C000)

(id15, x15,y15,z15) → (id15, x15,y15,z15,C100)

……


Map4
(id16,x16,y16,z16)
(id16,x16,y16,z16, C000)
(id17, x17,y17,z17)
(id17, x17,y17,z17, C100)
(id18,x18,y18,z18)
(id18, x18,y18,z18, C200)
(id19, x19,y19,z19)
(id19,x19,y19,z19, C000)
(id20, x20,y20,z20)
(id20, x20,y20,z20,C100)
……



Reduce1

(id1,id4,id6,id9,id11,id14,id16,id19,C000) → AkNN_Compute(I1,C000)

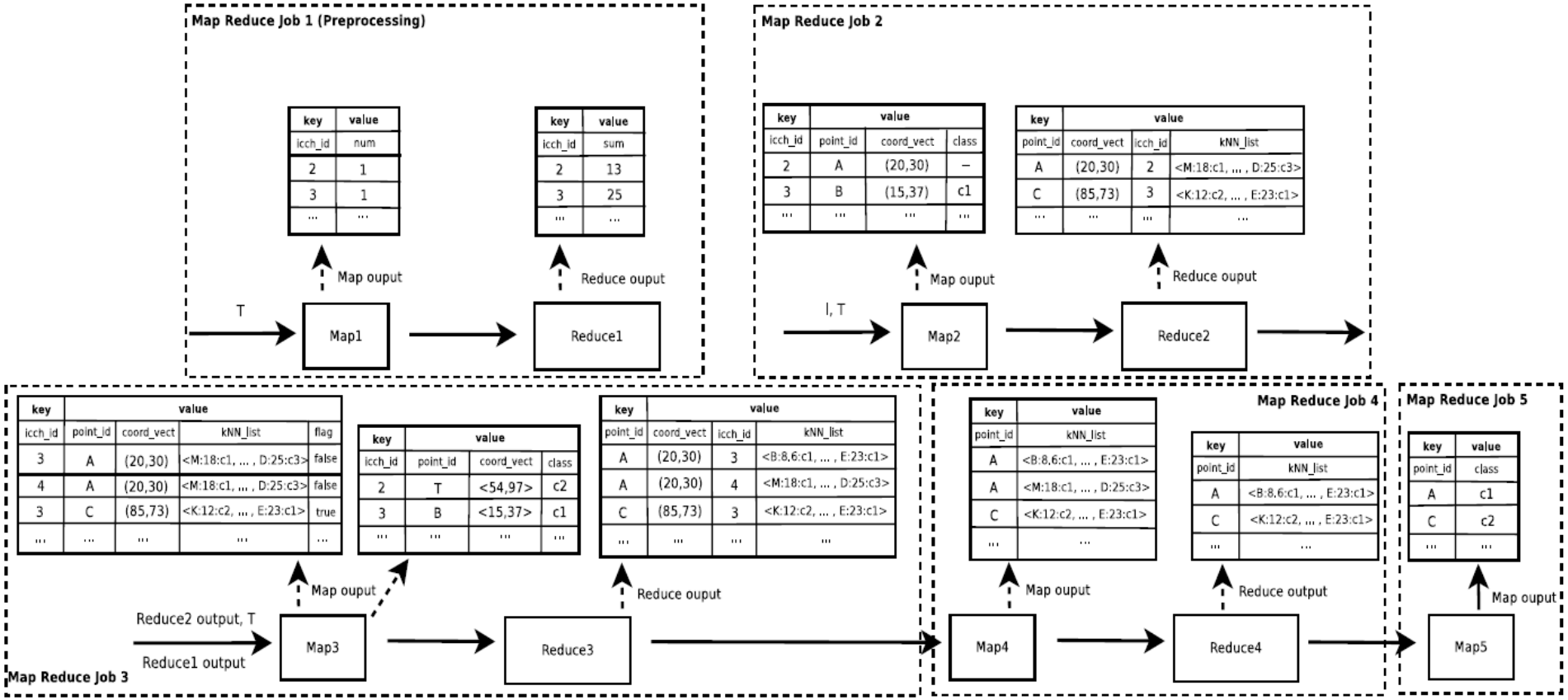
Reduce2

(id2,id5,id7,id10,id12,id15,id17,id20,C100) → AkNN_Compute(I2,C100)

Reduce3

(id3,id8,id13,id18,C200) → AkNN_Compute(I3,C200)

MAP REDUCE JOBS (better approach)



MAP REDUCE JOBS

1. **Distribution Information**. Count the number of points of T that fall into each ICCH.

2. **Primitive Computation Phase**. Calculate possible k-NN points $\forall i \in I$ from T in the same ICCH.

3. **Update Lists**. Draw the boundary ICSH $\forall i \in I$ and increase it, if needed, until it covers at least k-NN points of T. Check for overlaps of neighboring ICCHs and derive updates of k-NN lists.

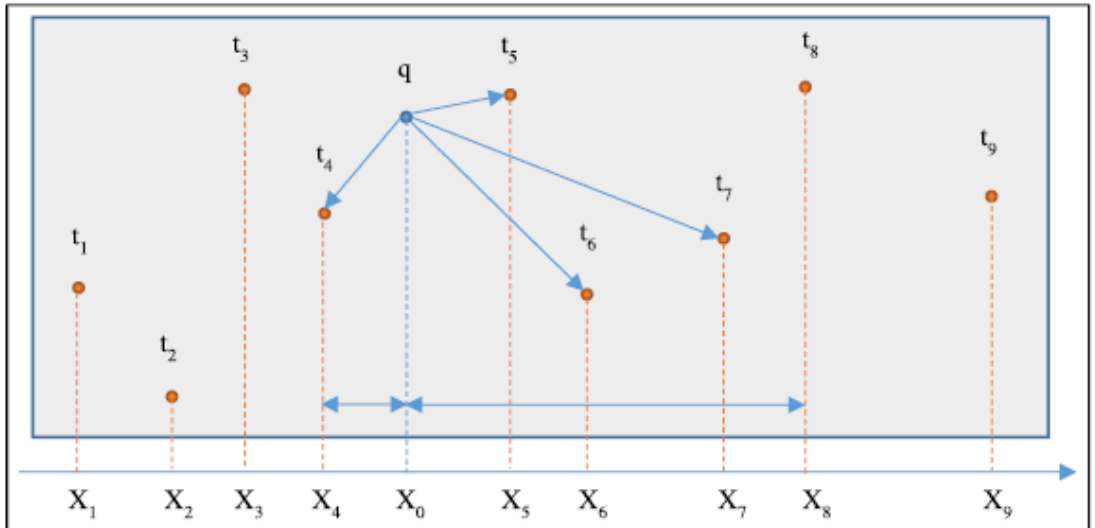
4. **Unify Lists**. Unify the updates of every k-NN list into one final k-NN list $\forall i \in I$.

5. **Classification**. Classify all points of I.

Table 1: Symbols and their meanings

n	granularity of space decomposition
k	number of nearest neighbors
d	dimensionality
D	a d-dimensional metric space
$dist(r, s)$	the distance from r to s
$kNN(r, S)$	the k nearest neighbors of r from S
$AkNNC(R, S)$	$\forall r \in R$ classify r based on $kNN(r, S)$
$ICCH$	interval, cell cube or hypercube
$ICSH$	interval, circle, sphere or hypersphere
I	input dataset
T	training dataset
c_r	the class of point r
C_T	the set of classes of dataset T
S_I	size of input dataset
S_T	size of training dataset
M	total number of Map tasks
R	total number of Reduce tasks

1st Improvement: Sweep Line in each cell….(assume cells with many points)



Max heap (k = 3)

Checking Training points to the right of q.

First k are pushed into the empty heap (t_5, t_6 and t_7), ordered by distance.

t_5	t_6	t_7

Checking t_8 :

t_8 : $|X_0 - X_8| > d(q, 7)$

Skipping points t_8, t_9

Checking tpoints to the left of q (t_4 first).

t_4 : $|X_0 - X_4| < d(q, 7)$

Calculate d(q, 4)

Check if $d(q, 4) < d(q, 7)$ → TRUE

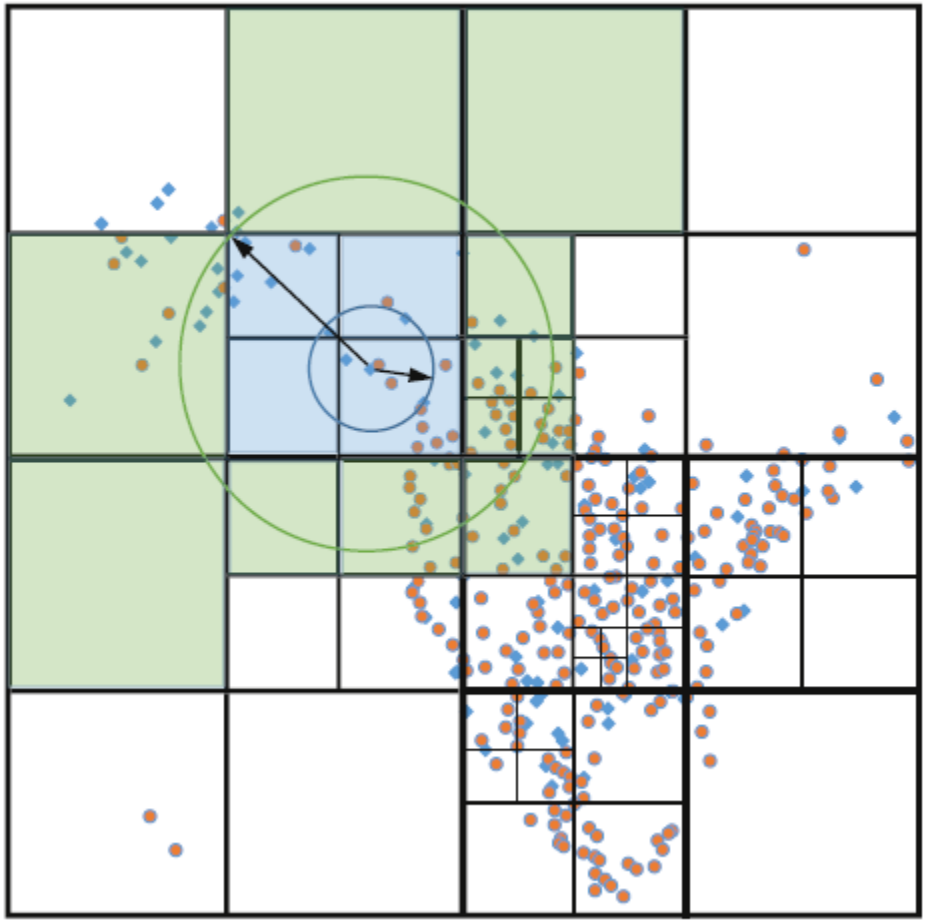
Pop (t_7)

Push (t_4)

Etc.

t_5	t_4	t_6

2nd Improvement: Quad-Tree Mappers (or Oct Tree Mappers)



1. A first sample of T set will be used for quad-tree construction in one machine (VM).

2. The decomposition of each cell to quadrants stops if the number of inside points is <= threshold.

3. Radius boost is driven by the Quad tree structure
4. Better Load Balancing

We have set up a cluster of 9 virtual machines (1 NameNode and 8 DataNodes) running Ubuntu Linux 16.04 64-bit. Each machine is equipped with a Xeon quad core at 2.1 GHz and 16 GB RAM, connected to a 10 Gbit/sec network. Hadoop version is 2.8.0, the replication factor is set to 1, HDFS chunk size is 128 MB and the virtual memory for each Map and Reduce task is set to 4 GB.

We used two real world datasets from OpenStreetMap [12], one consisting of 11.504.035 points (the coordinates of parks around the world) that weighs 373 MB and another consisting of 11.473.662 points (this is a 10% subset of a dataset that contains the coordinates of buildings around the world) that weighs 383 MB. The first one is used as Input and the second one as Training dataset.

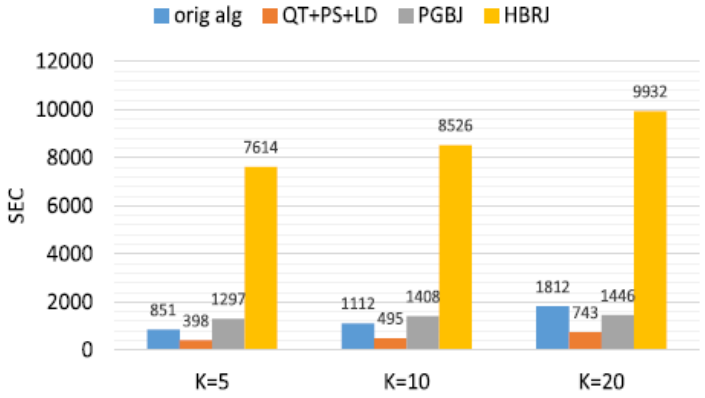


Fig. 58. $2D$ comparison, $11M \times 11M$ datasets.

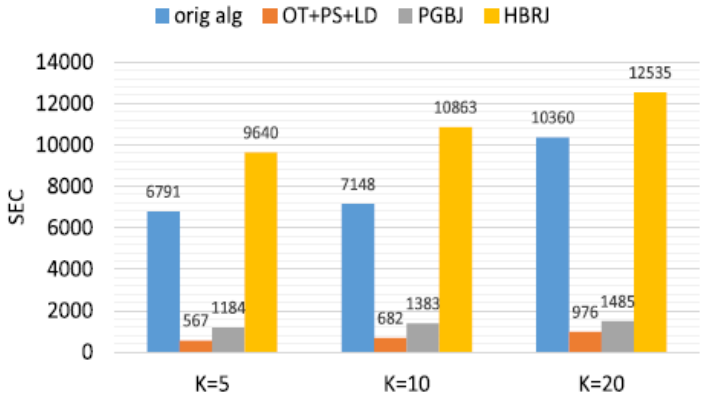


Fig. 59. $3D$ comparison, $11M \times 11M$ datasets.

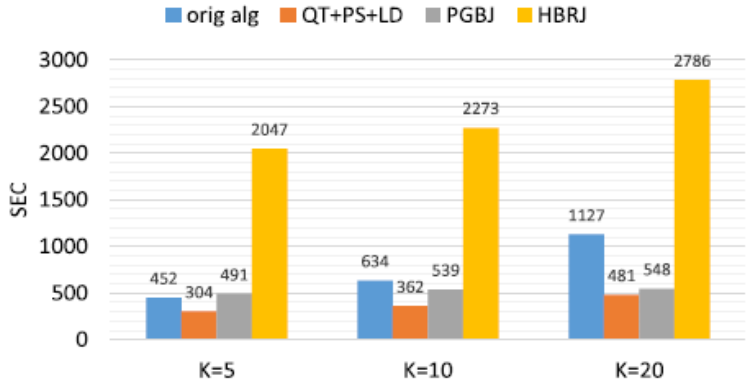


Fig. 60. $2D$ comparison, $5M \times 5M$ datasets.

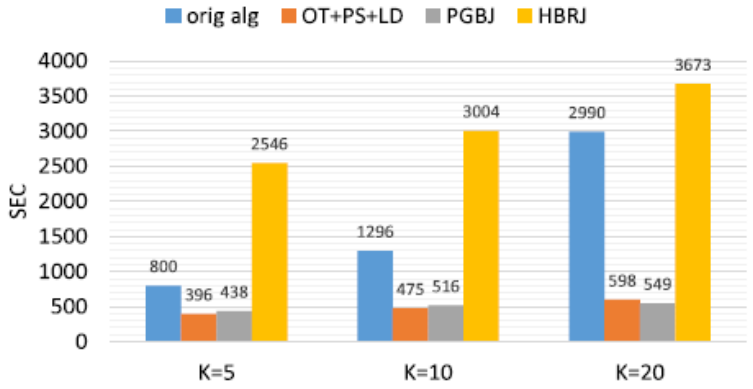


Fig. 61. $3D$ comparison, $5M \times 5M$ datasets.

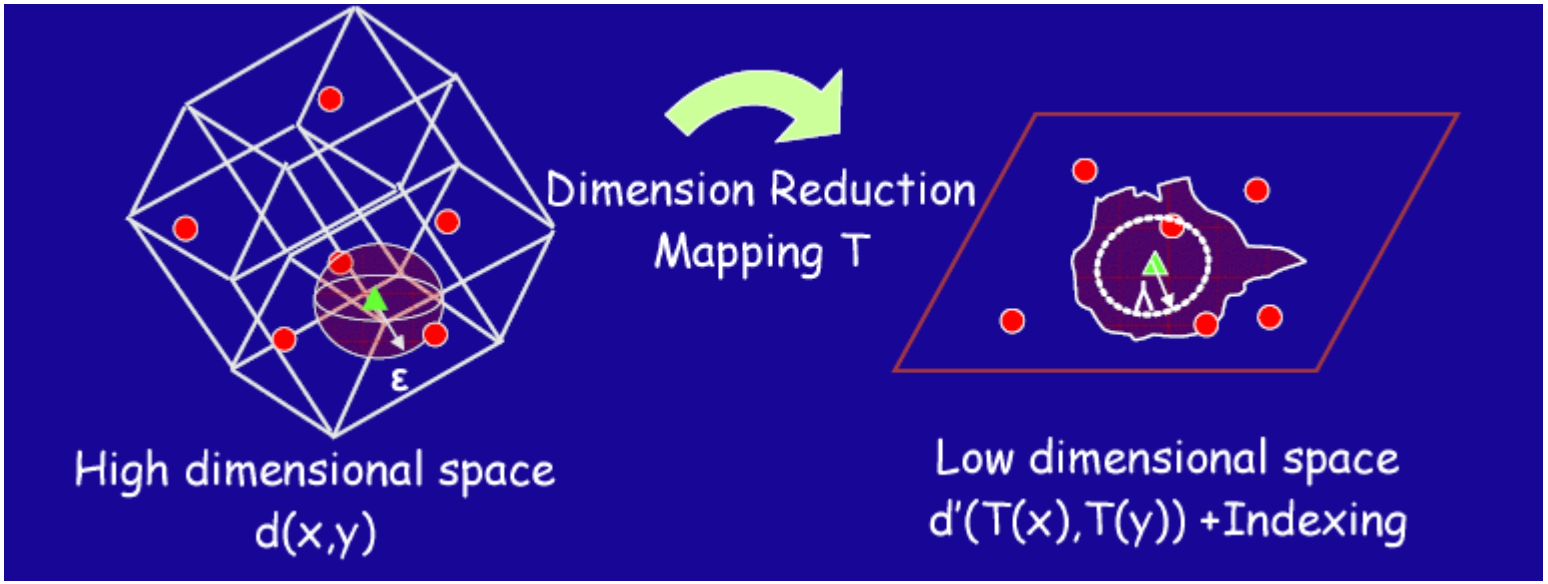
Relevant Publications

- Panagiotis Moutafis, George Mavrommatis, Michael Vassilakopoulos, Spyros Sioutas: Efficient processing of all-k-nearest-neighbor queries in the MapReduce programming framework. Data Knowl. Eng. 121: 42-70,Elsevier (2019)
- Elias Dritsas, Maria Trigka, Panagiotis Gerolymatos, Spyros Sioutas: Trajectory Clustering and k-NN for Robust Privacy Preserving Spatiotemporal Databases. Algorithms 11(12): 207 (2018)
- Nikolaos Nodarakis, Evaggelia Pitoura, Spyros Sioutas, Athanasios K. Tsakalidis, Dimitrios Tsoumakos, Giannis Tzimas: kdANN+: A Rapid AkNN Classifier for Big Data. Trans. Large-Scale Data- and Knowledge-Centered Systems 24: 139-168, Springer (2016)
- Nikolaos Nodarakis, Angeliki Rapti, Spyros Sioutas, Athanasios K. Tsakalidis, Dimitrios Tsolis, Giannis Tzimas, Yannis Panagis: (A)kNN Query Processing on the Cloud: A Survey. ALGOCLOUD 2016: 26-40 (Springer)
- Nikolaos Nodarakis, Spyros Sioutas, Athanasios K. Tsakalidis, Giannis Tzimas: Large Scale Sentiment Analysis on Twitter with Spark. EDBT/ICDT 2016
- Nikolaos Nodarakis, Evaggelia Pitoura, Spyros Sioutas, Athanasios K. Tsakalidis, Dimitrios Tsoumakos, Giannis Tzimas: Efficient Multidimensional AkNN Query Processing in the Cloud. DEXA (1) 2014: 477-491.
- Spyros Sioutas, Emmanouil Magkos, Ioannis Karydis, Vassilios S. Verykios: Uncertainty for Privacy and 2-Dimensional Range Query Distortion. JCSE 5(3): Special Issue of ICDM 2011, 210-222 (2011).
- Spyros Sioutas, Emmanouil Magkos, Ioannis Karydis, Vassilios S. Verykios: Uncertainty for Anonymity and 2-Dimensional Range Query Distortion. Privacy in Statistical Databases 2010: 85-96 (Springer).

Dimensionality Reduction?

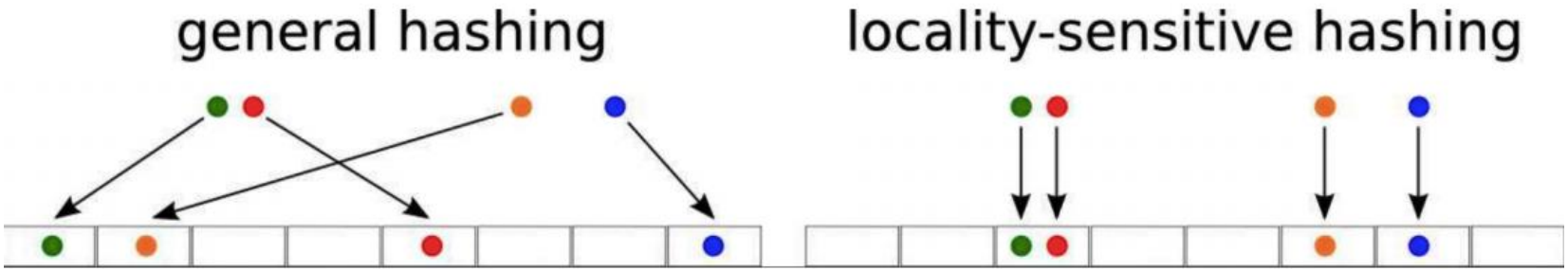
Idea: Find a mapping T to reduce the dimensionality of the data.

Drawback: May not be able to find all similar objects (i.e., distance relationships might not be preserved)



Locality Sensitive Hashing?

- It works well for documents (SET of n-grams)..
- What about spatio-temporal coordinates???



- A family **H** of functions **h: R^d→U** is called **(P_1,P_2,r,cr)**-sensitive if for any p,q:
  if **D(p,q)≤r**, then **Pr[h(p)=h(q)] ≥ P1**
  If **D(p,q)≥ cr**, then **Pr[h(p)=h(q)] ≤ P2**
- **P1 > P2**

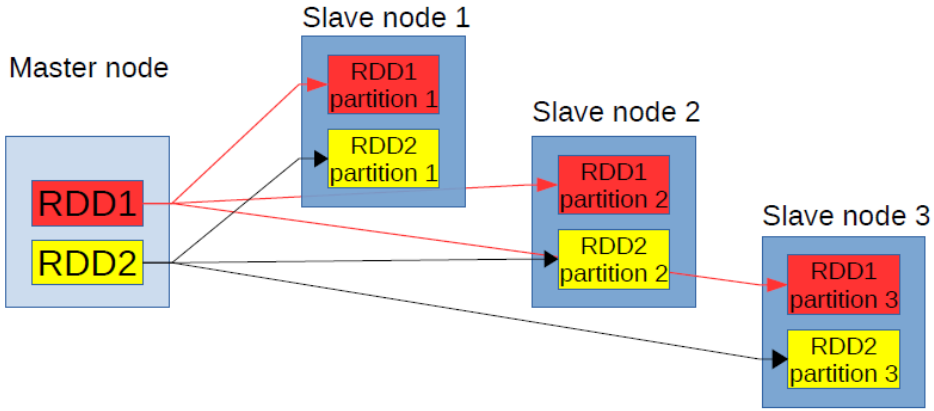
Implementation of optimal high-dimensional main memory data structures in SPARK RDDs ??

f.e.. Optimal
Range Trees?
Optimal M-trees?

SPIS - **SP**ark based **I**nterpolation **S**earch Tree (2)

[Papadopoulos, Sioutas, Zacharatos & Zaroliagis (2019)]

Resilient Distributed Dataset is the main data structure of Spark



- Input dataset is stored in an RDD and its elements are kept sorted across all partitions
- An IST is created on each of the RDD's partitions

Thank you!!!!!

Questions???