CSCI203

Algorithms and Data Structures

Multi-dimensional Search Trees

Lecturer: Dr. Fenghui Ren
Room 3.203
Email: fren@uow.edu.au

Query Types

- **Exact match query:** Asks for the object(s) whose key matches query key exactly.

- **Range query:** Asks for the objects whose key lies in a specified query range (interval).

- **Nearest-neighbor query:** Asks for the objects whose key is “close” to query key.

Exact Match Query

- Suppose that we store employee records in a database:

ID	Name	Age	Salary	#Children

- Example:
  - **key=ID:** retrieve the record with ID=12345

Range Query

- Example:
  - **key=Age:** retrieve all records satisfying

    20 < Age < 50
  - **key= #Children:** retrieve all records satisfying

    1 < #Children < 4

ID	**Name**	**Age**	**Salary**	**#Children**

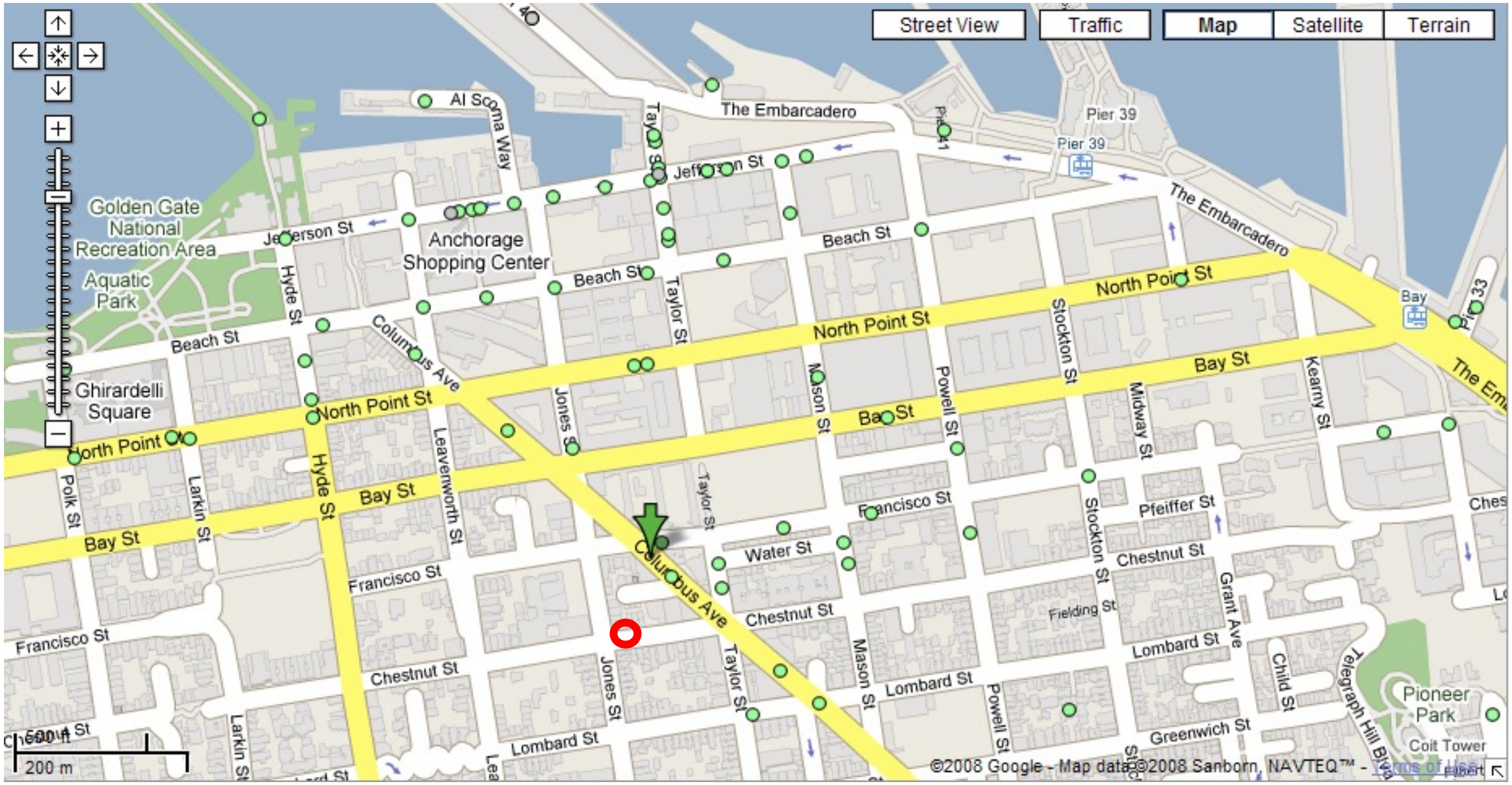
Nearest-Neighbor(s) (NN) Query

- 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**

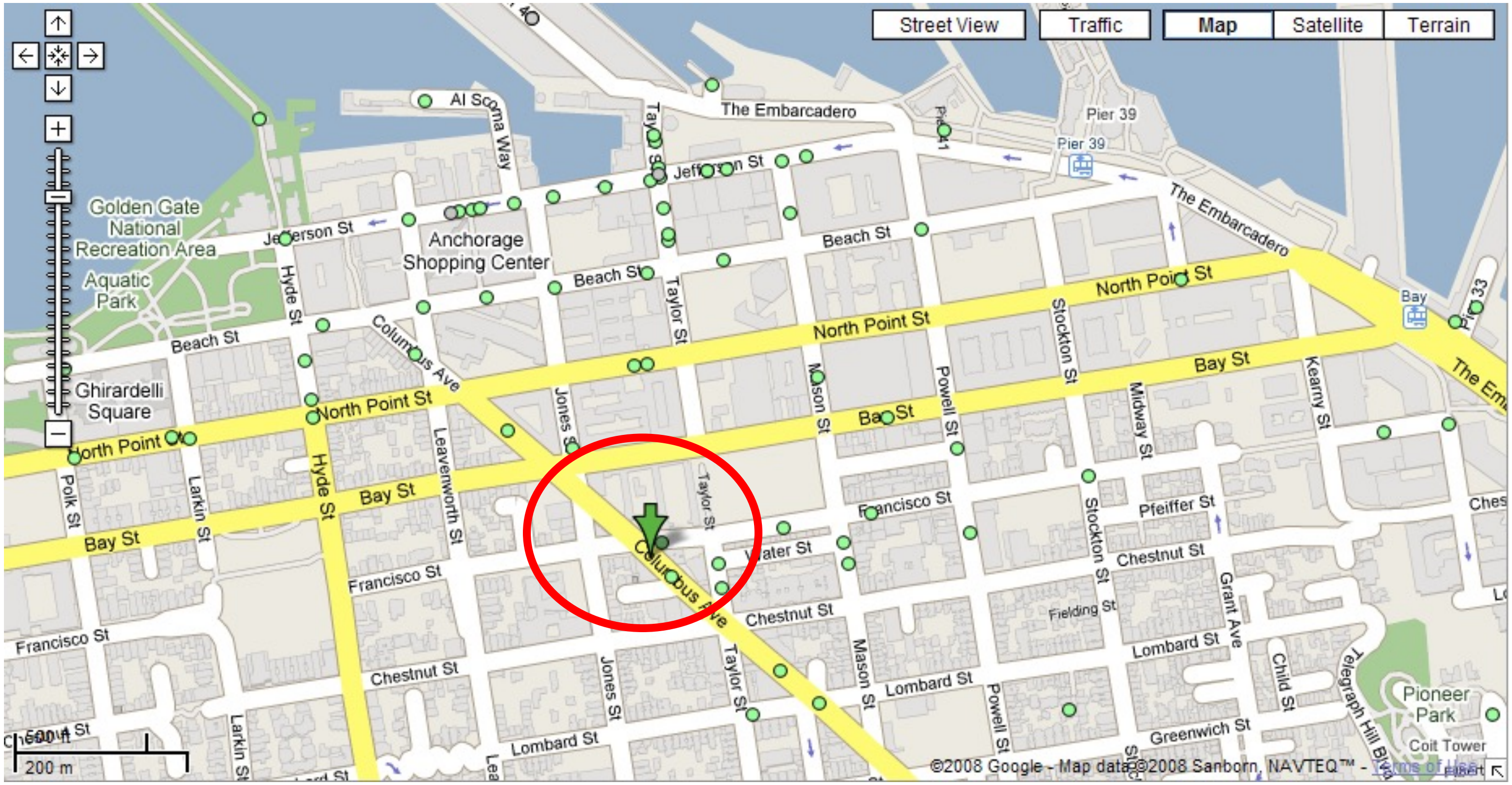
Nearest Neighbor(s) Query

- What is the closest restaurant to my hotel?



Nearest Neighbor(s) Query…

- Find the 4 closest restaurants to my hotel



Multi-dimensional Query

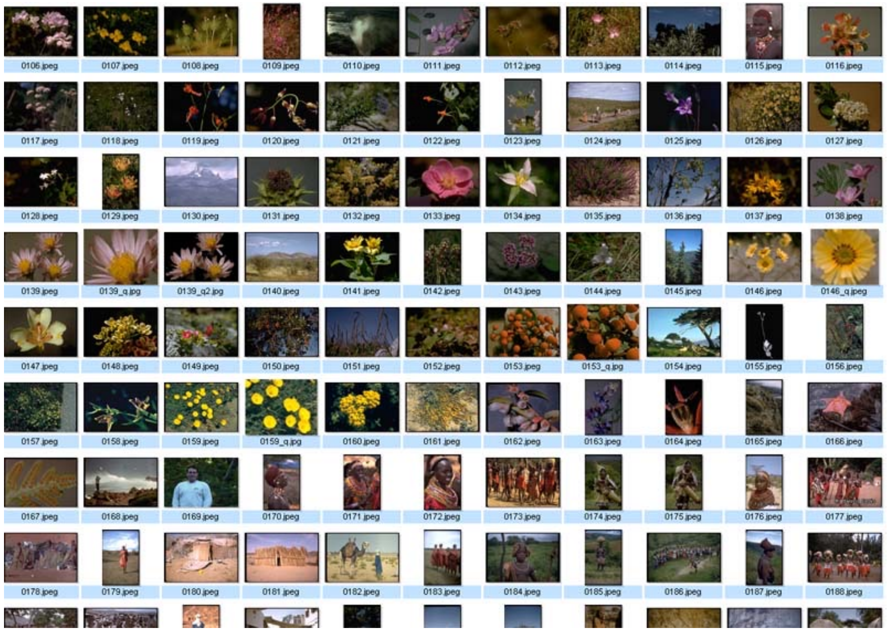
- In practice, queries might involve multi-dimensional keys.
  - **key=(Name, Age):** retrieve all records with Name="George" and "50 <= Age <= 70"

ID	Name	Age	Salary	#Children

Nearest Neighbor Query in High Dimensions

- Very important and practical problem!
  - Image retrieval

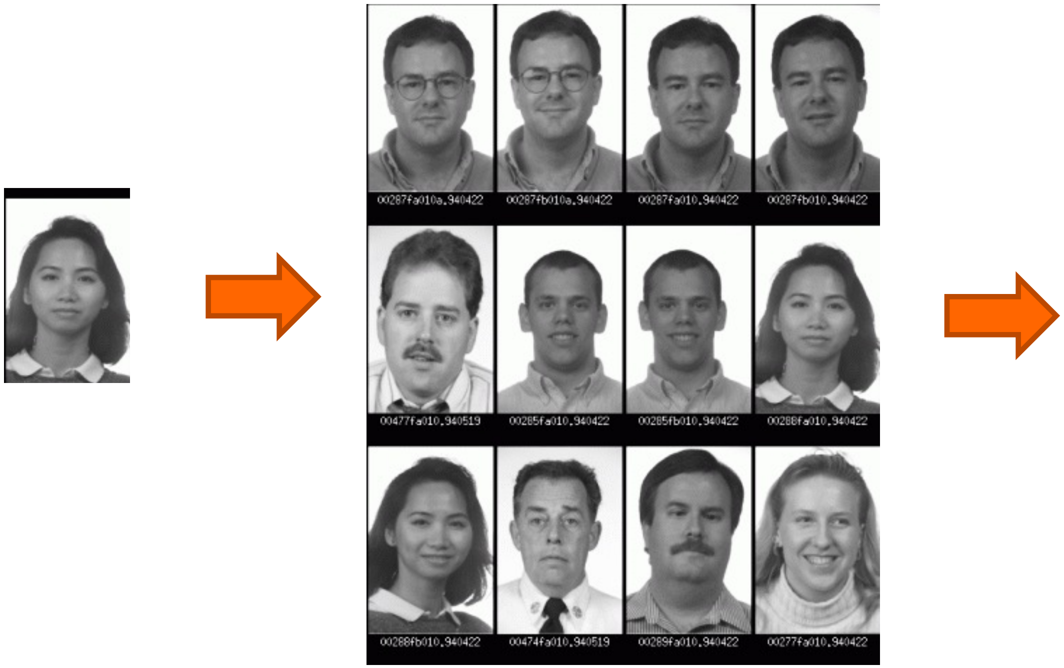
$(f_1, f_2, .., f_k)$



find K closest matches (i.e., K Nearest Neighbors)

Nearest Neighbor Query in High Dimensions

- Face recognition



find closest match (i.e., 1- nearest neighbor)

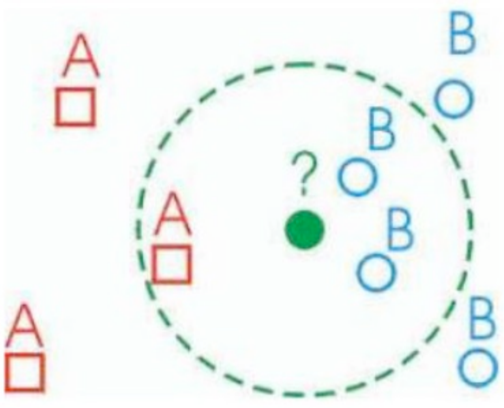
We will discuss …

- Range trees
- KD-trees

Interpreting Queries Geometrically

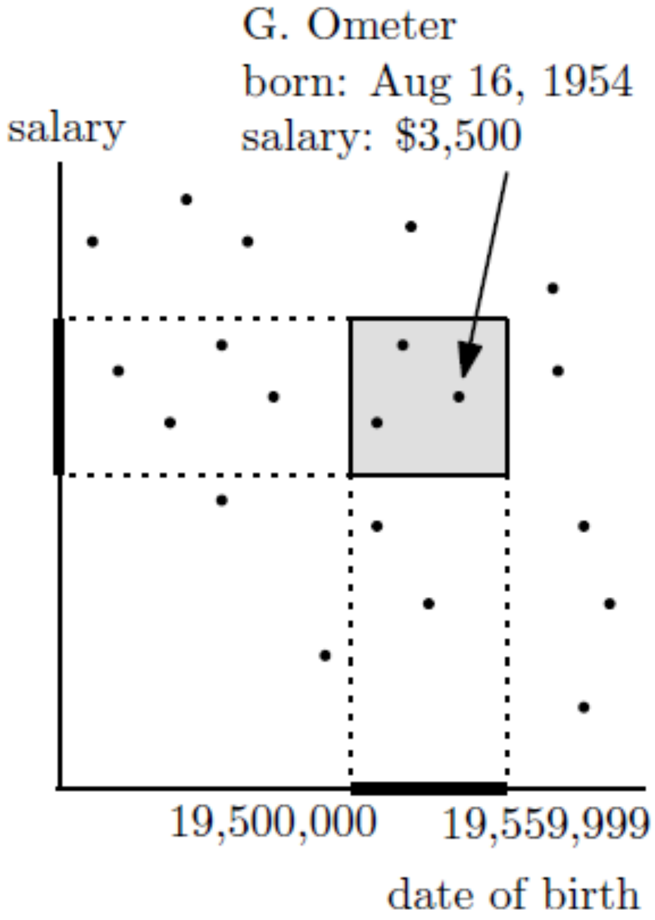
- Multi-dimensional keys can be thought as "*points*" in *high dimensional spaces*.

Queries about records → Queries about points



Example 1- Range Search in 2D

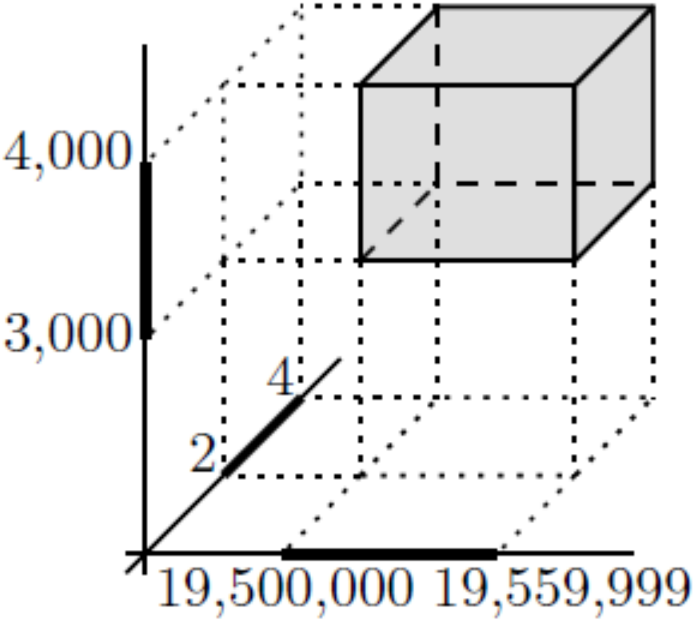
A database query may ask for all employees with age between a_1 and a_2, and salary between s_1 and s_2



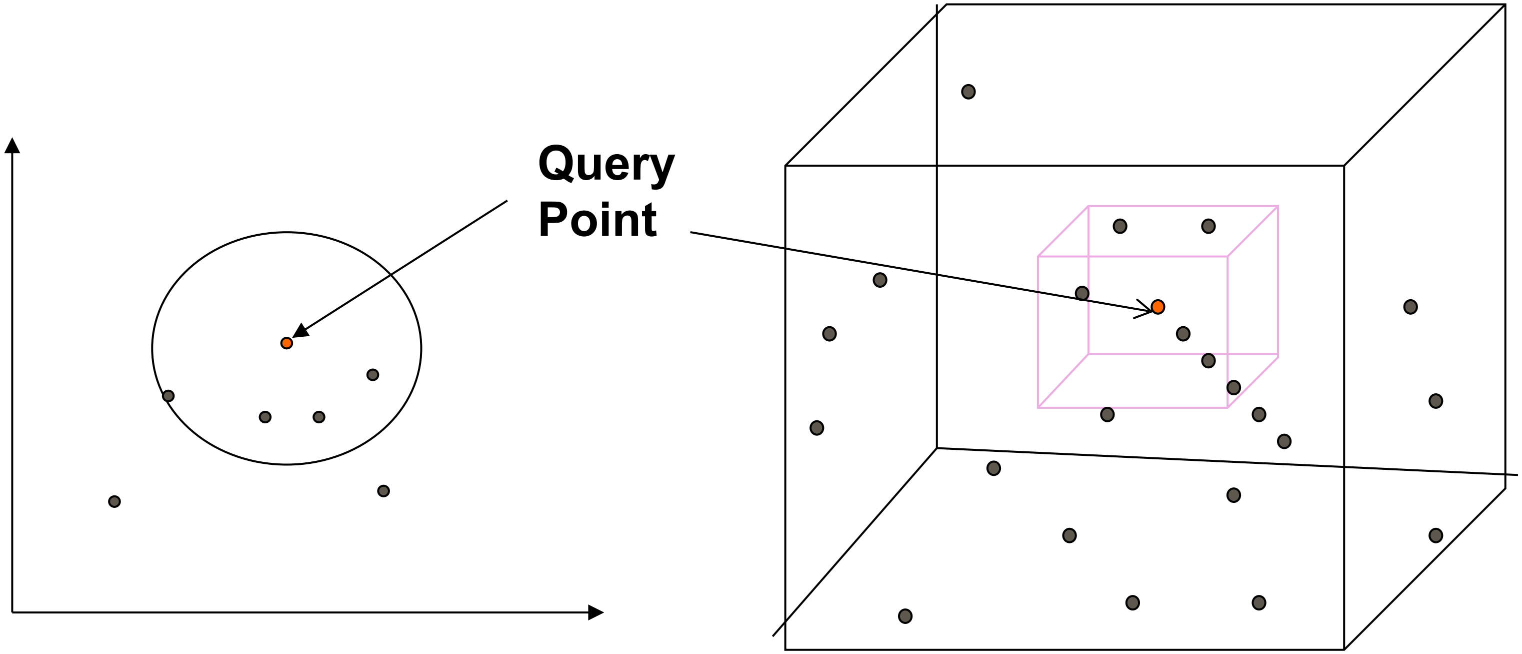
age = 10,000 x year + 100 x month + day

Example 2 - Range Search in 3D

Example of a 3-dimensional (orthogonal) range query: children in $[2, 4]$, salary in $[3000, 4000]$, date of birth in $[19{,}500{,}000\ ,\ 19{,}559{,}999]$

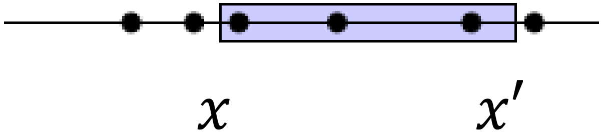


Example 3 - Nearest Neighbors Search



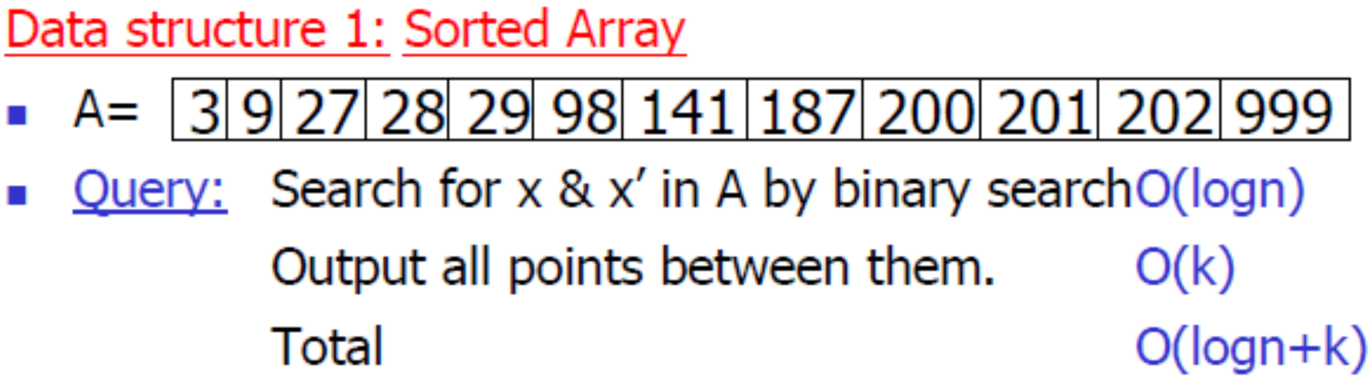
1D Range Search

- Data
  - $P = \{p_1, p_2, \cdots, p_n\}$ in 1D space (a set of real numbers)
- Query
  - Which points are in the interval $[x, x']$



1D Range Search

Range: $[x, x']$



Data structure 1: Sorted Array

- A=

3	9	27	28	29	98	141	187	200	201	202	999

- Query: Search for x & x′ in A by binary search O(logn)

  Output all points between them. O(k)

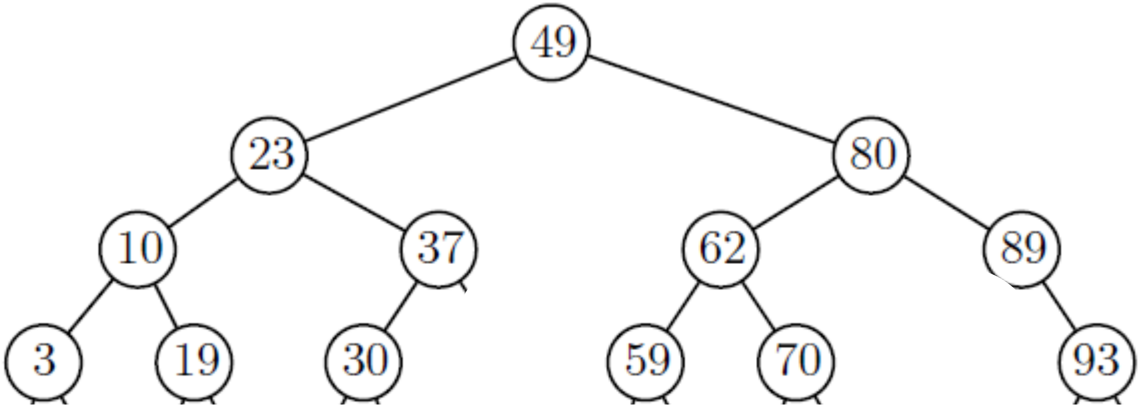
  Total O(logn+k)

Example: retrieve all points in [25, 90]

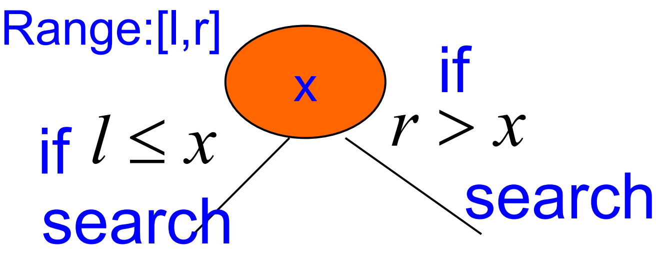
- Does not generalize well to high dimensions.

1D Range Search

- Data Structure 2: BST
  - Search using binary search property.
  - Some subtrees are eliminated during search.



Search using:



Range:[l,r]

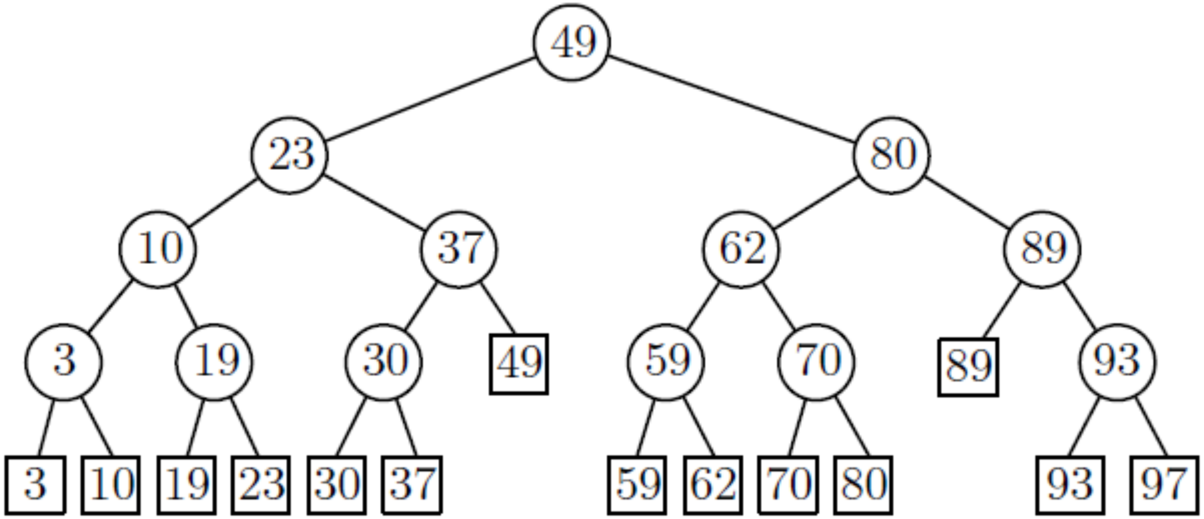
if $l \leq x$ search

if $r > x$ search

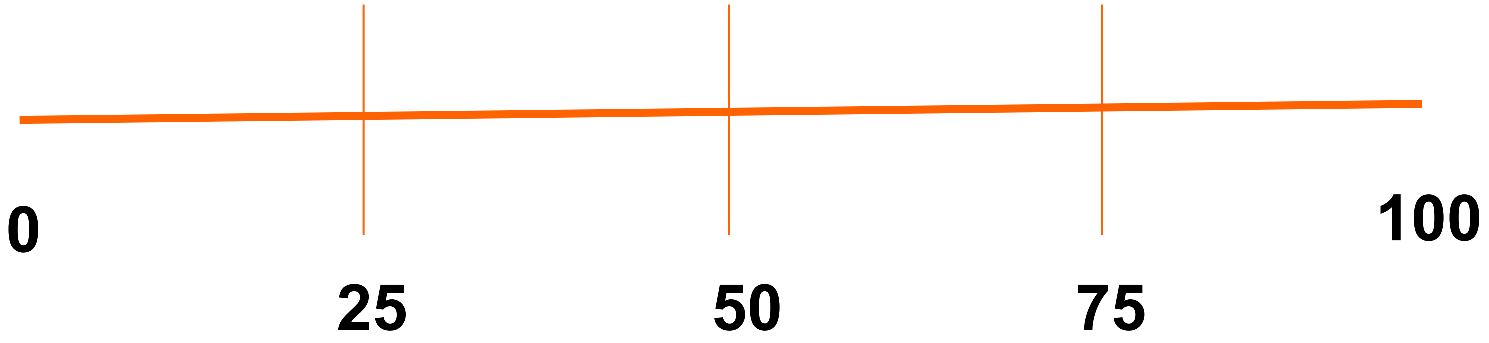
Example: retrieve all points in [25, 90]

1D Range Search

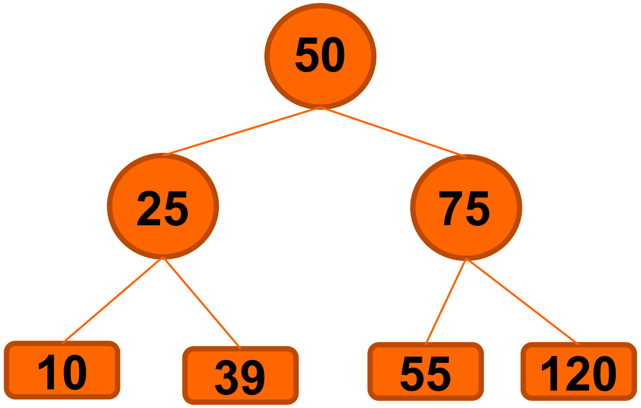
- Data Structure 3: BST with data stored in leaves
  - Internal nodes store *splitting values* (i.e., not necessarily same as data).
  - Data points are stored in the leaf nodes.



BST with data stored in leaves

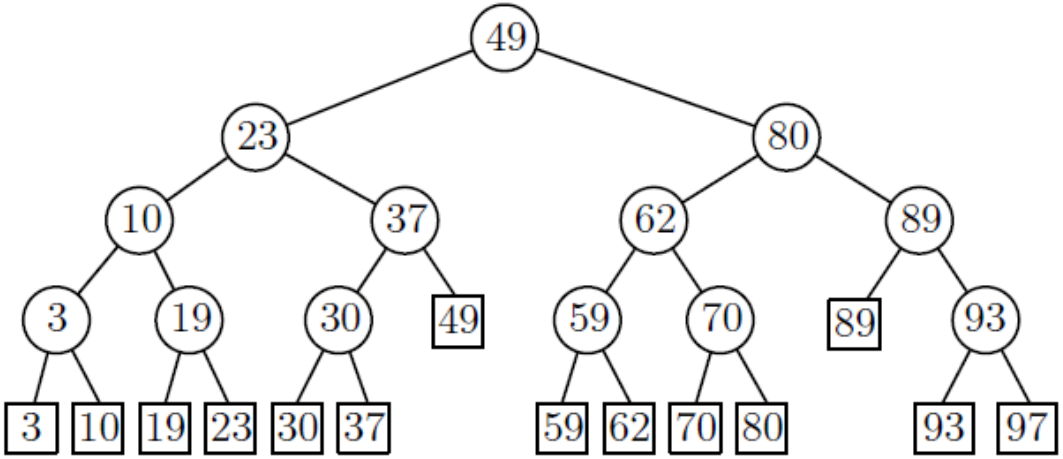


Data: 10, 39, 55, 120



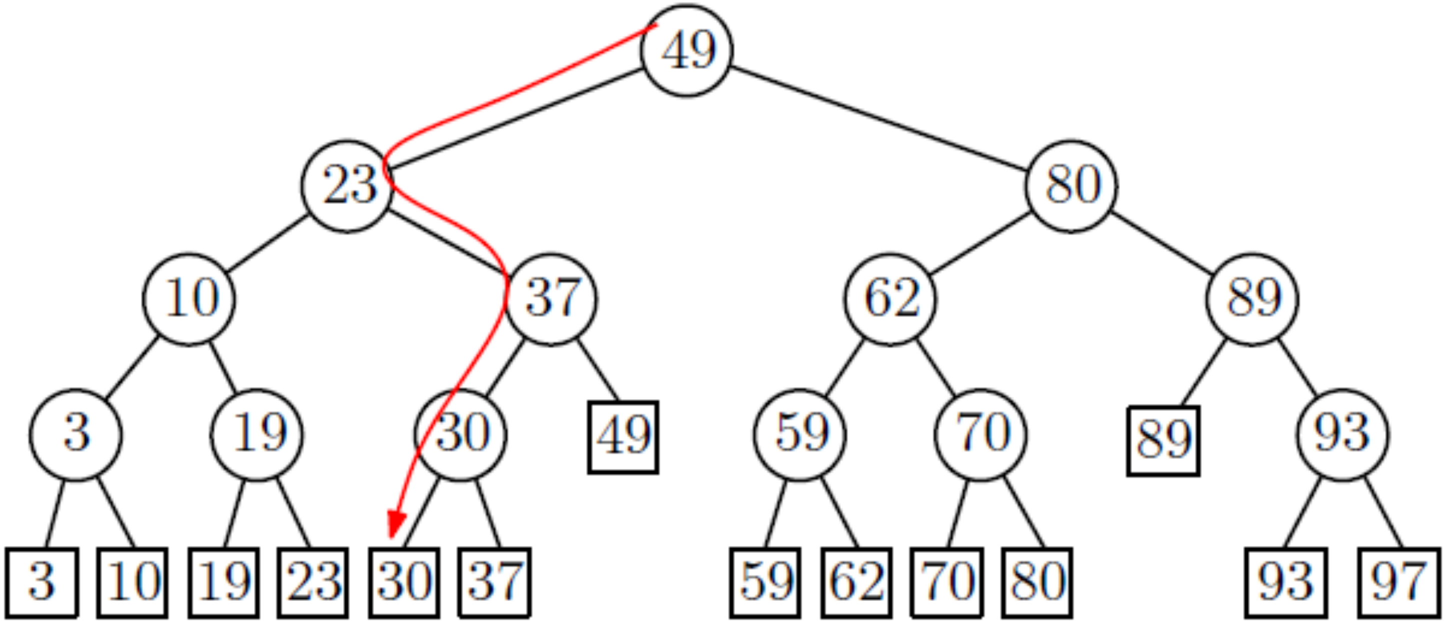
1D Range Search

- Retrieving data in $[x, x']$
  - Perform binary search twice, once using x and the other using x'
  - Suppose binary search ends at leaves l and l'
  - The points in $[x, x']$ are the ones stored between l and l' plus, possibly, the points stored in l and l'



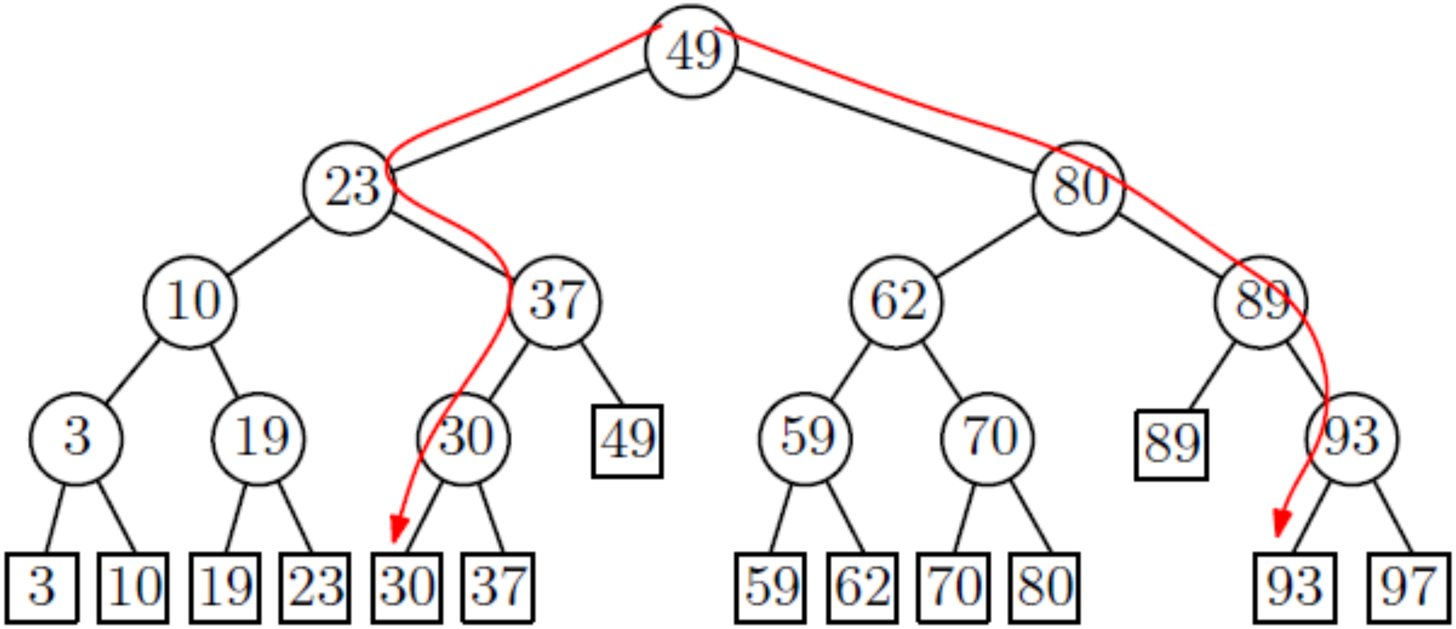
1D Range Search

- **Example:** retrieve all points in $[25, 90]$
  - The search path for 25 is:



1D Range Search

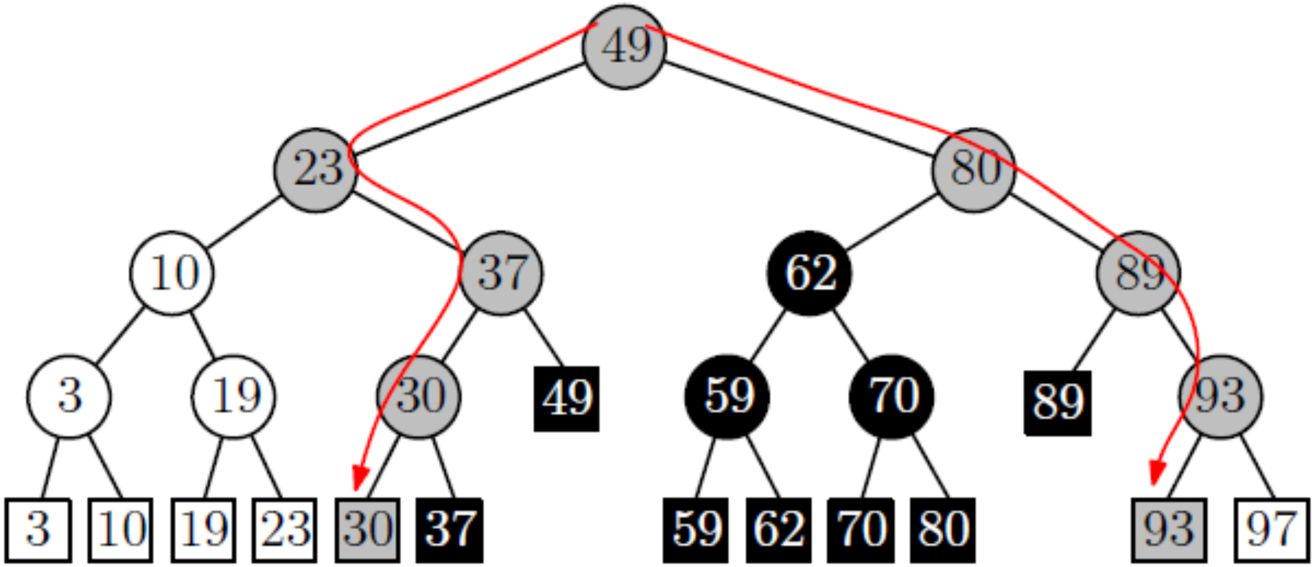
- The search for 90 is:



1D Range Search

- Examine the leaves in the sub-trees between the two traversing paths from the root.

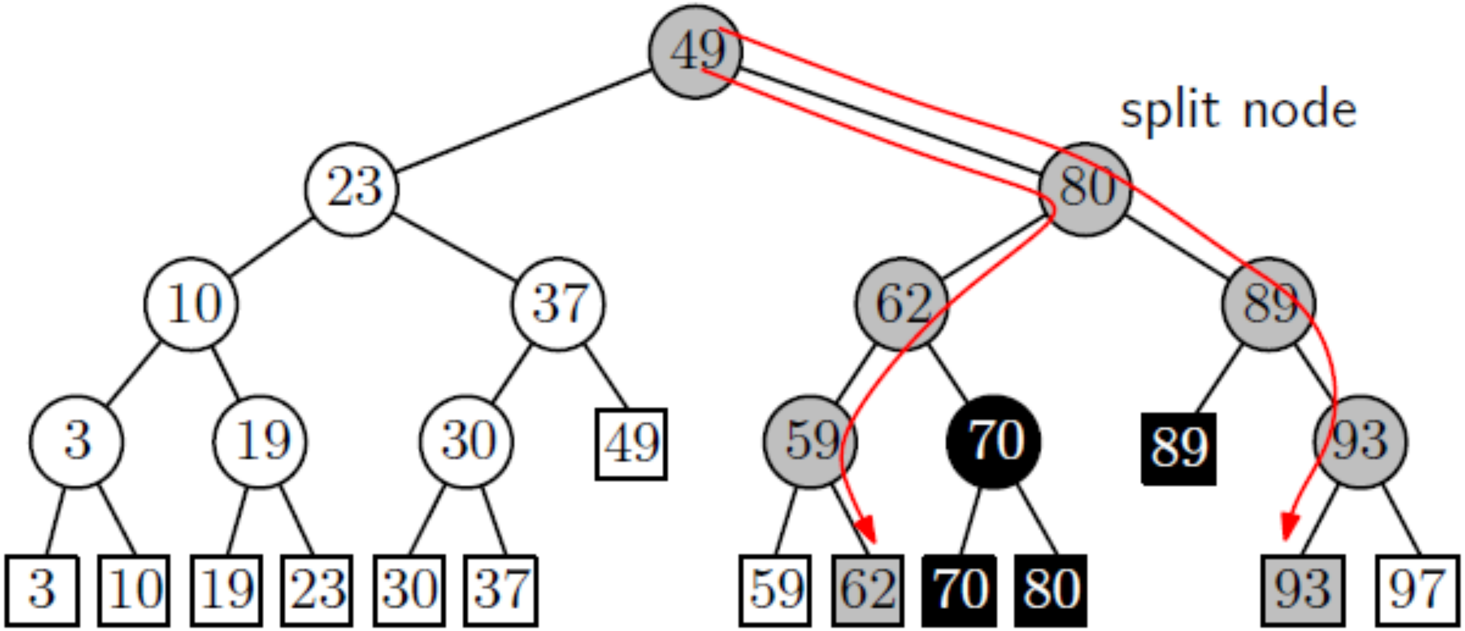
split node



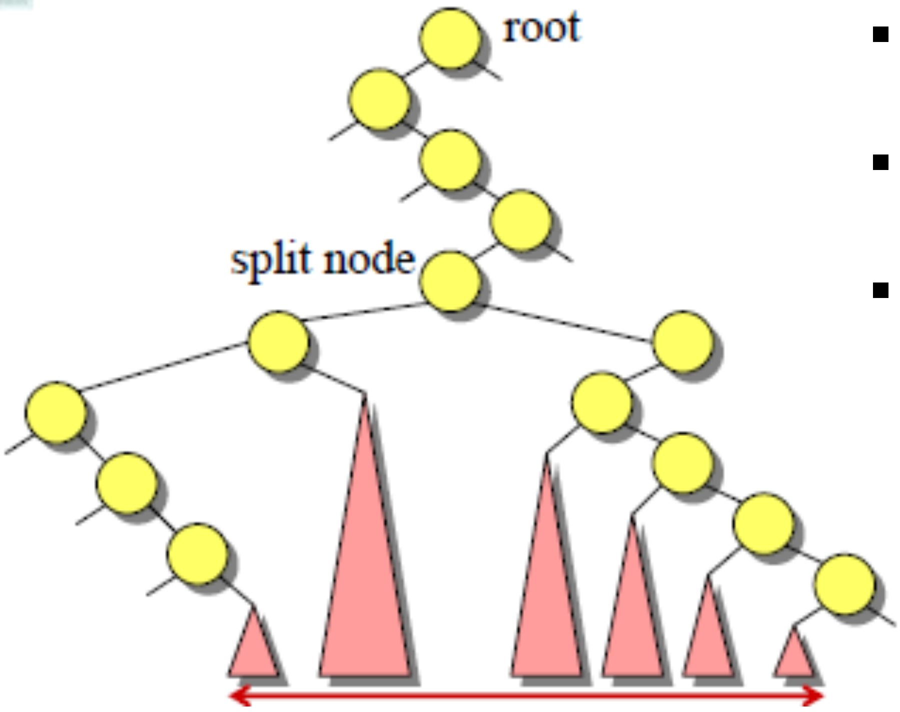
retrieve all points in $[25, 90]$

1D Range Search - Another Example

A 1-dimensional range query with $[61, 90]$



1D Range Search

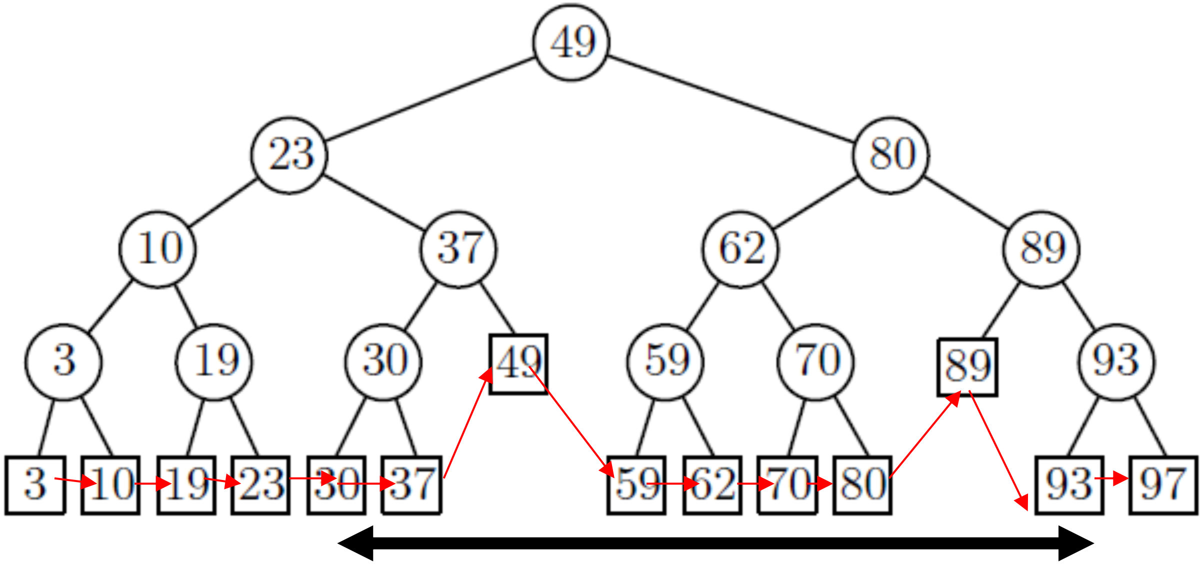


- How do we find the leaves of interest?
  - Find split node (i.e., node where the paths to x and x' split).
  - Left turn: report leaves in right subtrees
  - Right turn: report leaves in left substrees

O(logn + k) time where k is the number of items reported.

1D Range Search

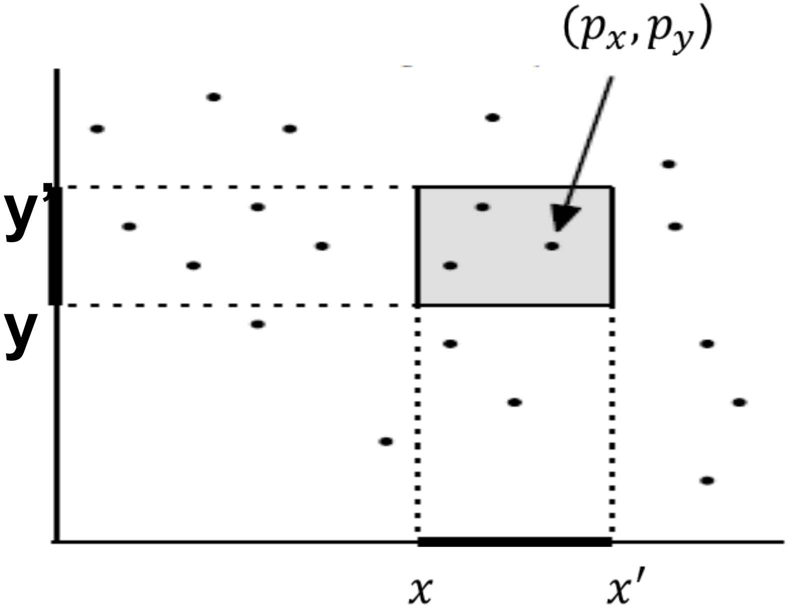
- Speed-up search by keeping the leaves in sorted order using a linked-list.



2D Range Search

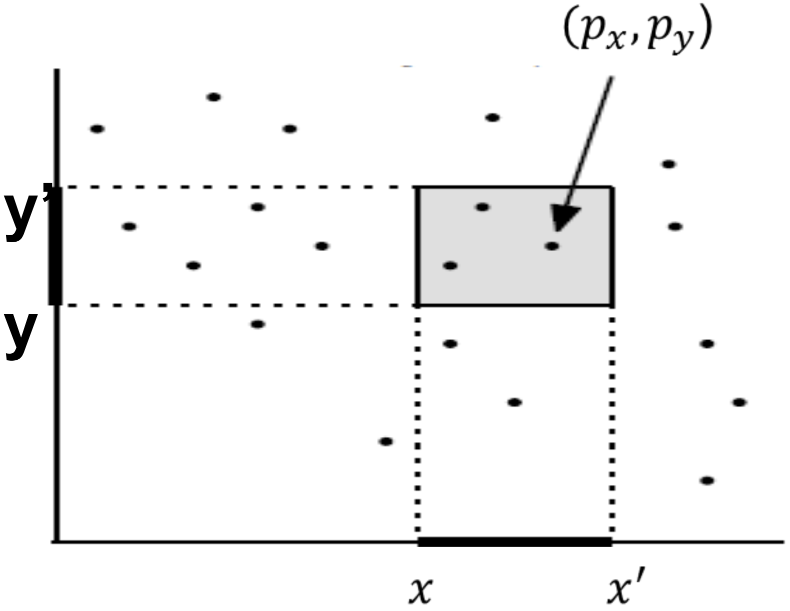
- A 2D range query asks for the points inside a query rectangle $[x, x'] \times [y, y']$
- A point (p_x, p_y) lies in this rectangle if and only if:

$$p_x \in [x, x'] \quad and\ p_y \in [y, y']$$

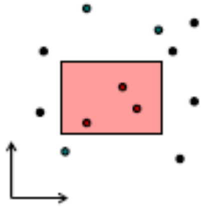


2D Range Search…

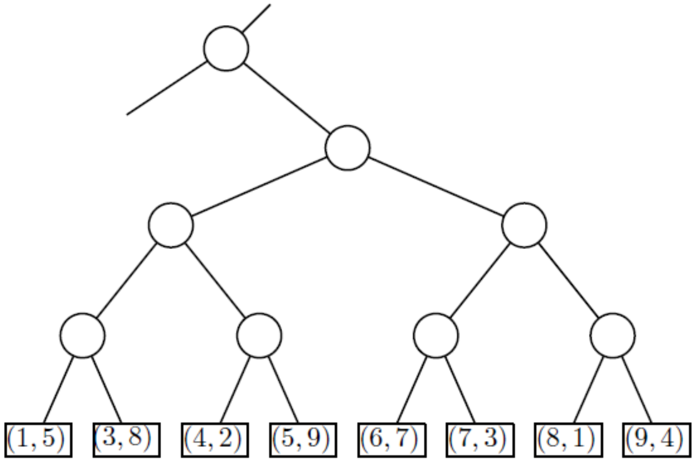
- A 2D range query can be decomposed in two 1D range queries:
  - One on the x-coordinate of the points.
  - The other on the y-coordinates of the points.

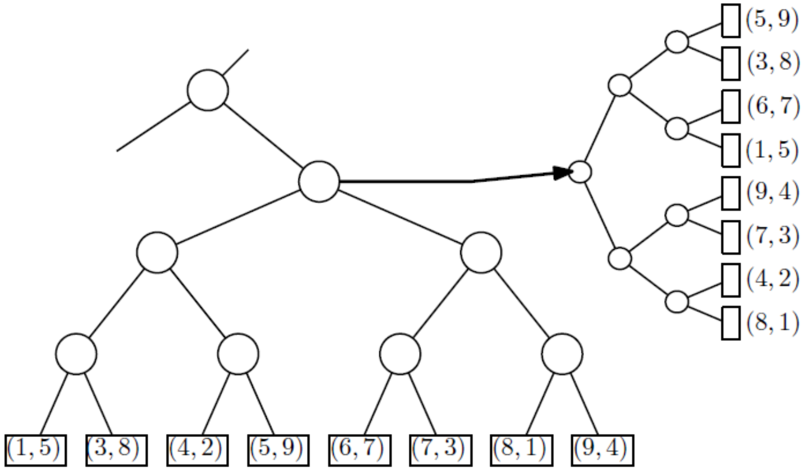


2D Range Search...

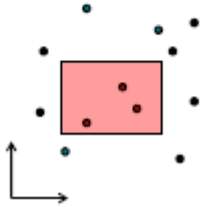


- Store a ***primary 1D range tree*** for all the points based on *x-coordinate.*
- For each node, store a ***secondary 1D range tree*** based on *y-coordinate.*

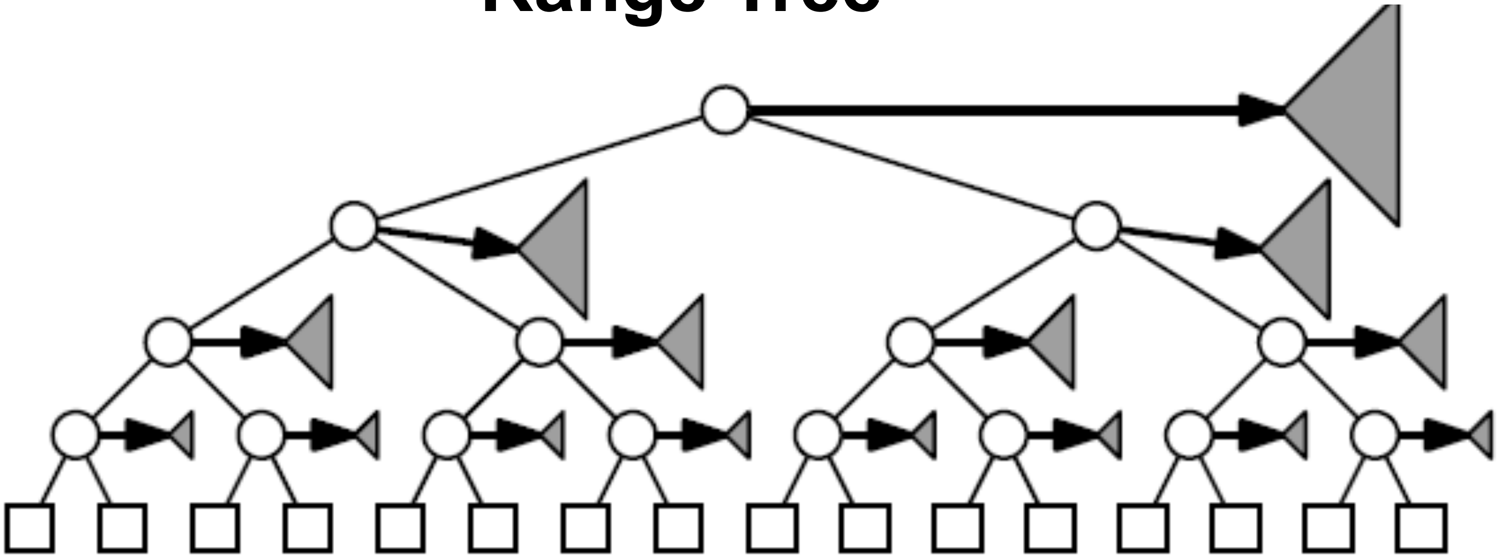




2D Range Search…

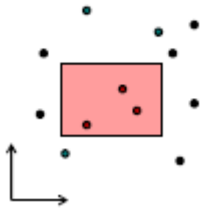


Range Tree

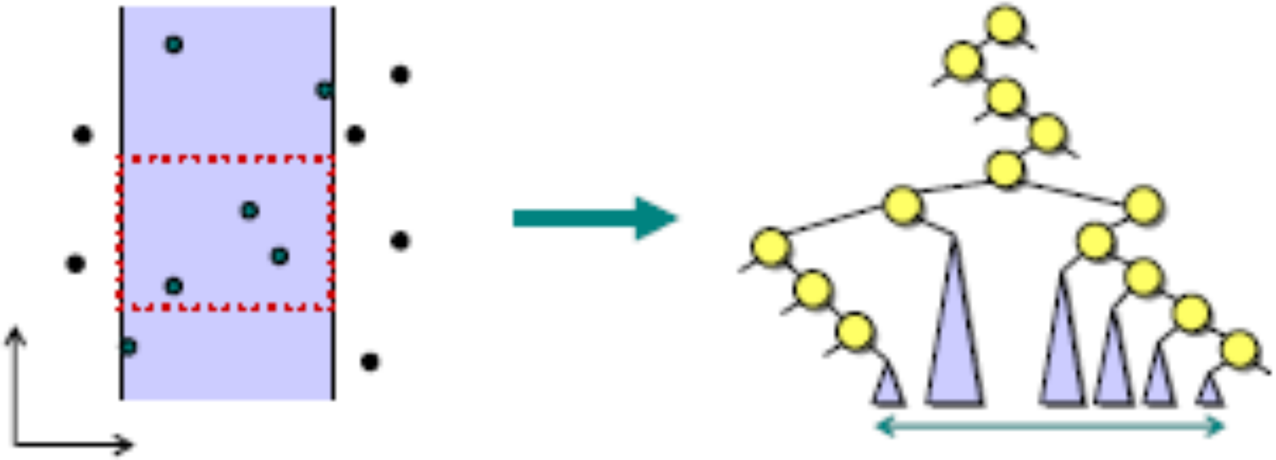


Space requirements: O(nlogn)

2D Range Search…

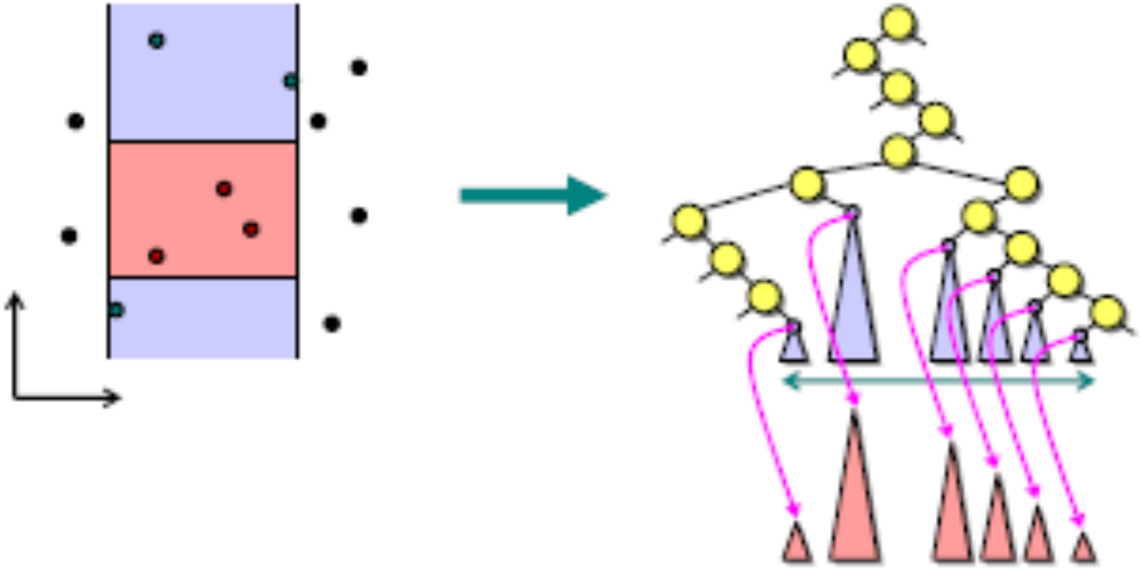


- Search using the x-coordinate only.
- How to restrict to points with proper *y-coordinate?*



2D Range Search…

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



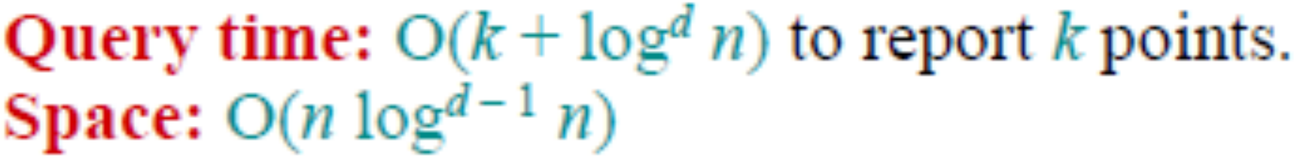
- Query cost: $O(\log^2 n+k)$

Range Search in d dimensions

1D query time: O(logn + k)

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

d dimensions:

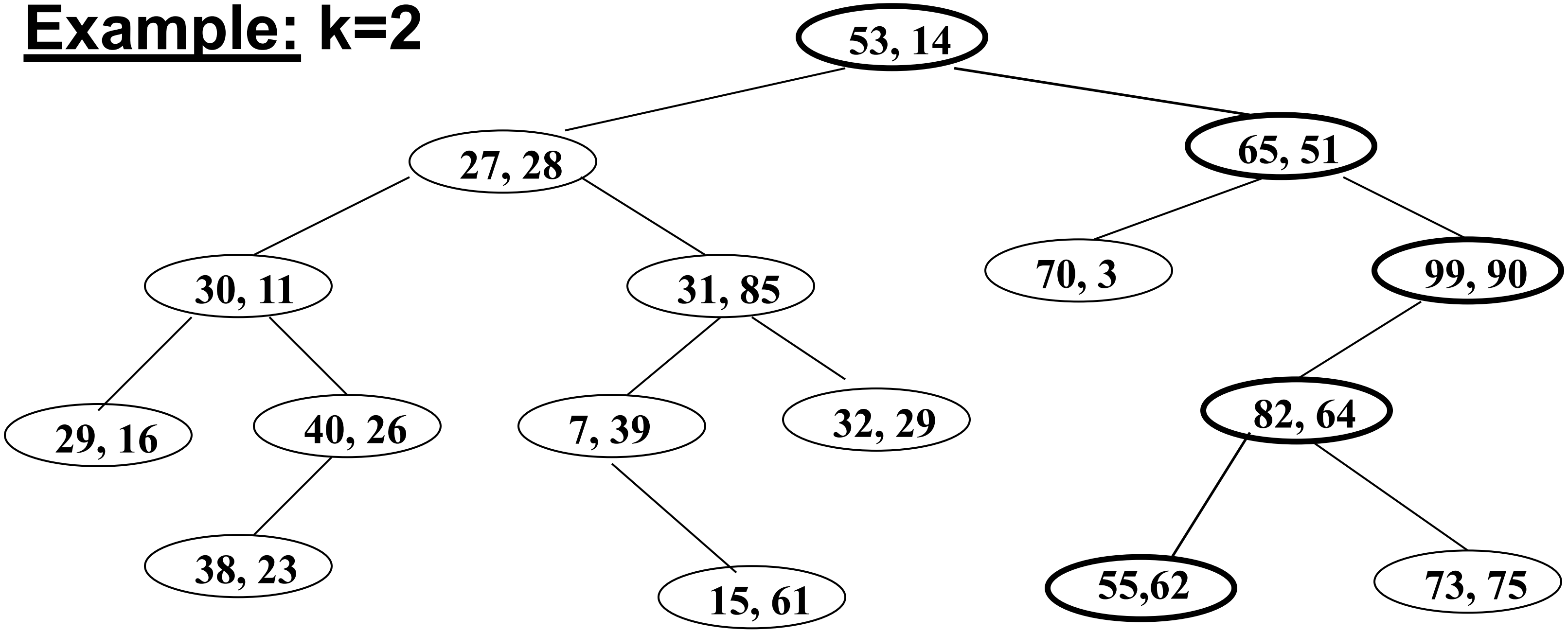


Query time: $O(k + \log^d n)$ to report k points.
Space: $O(n \log^{d-1} n)$

KD Tree

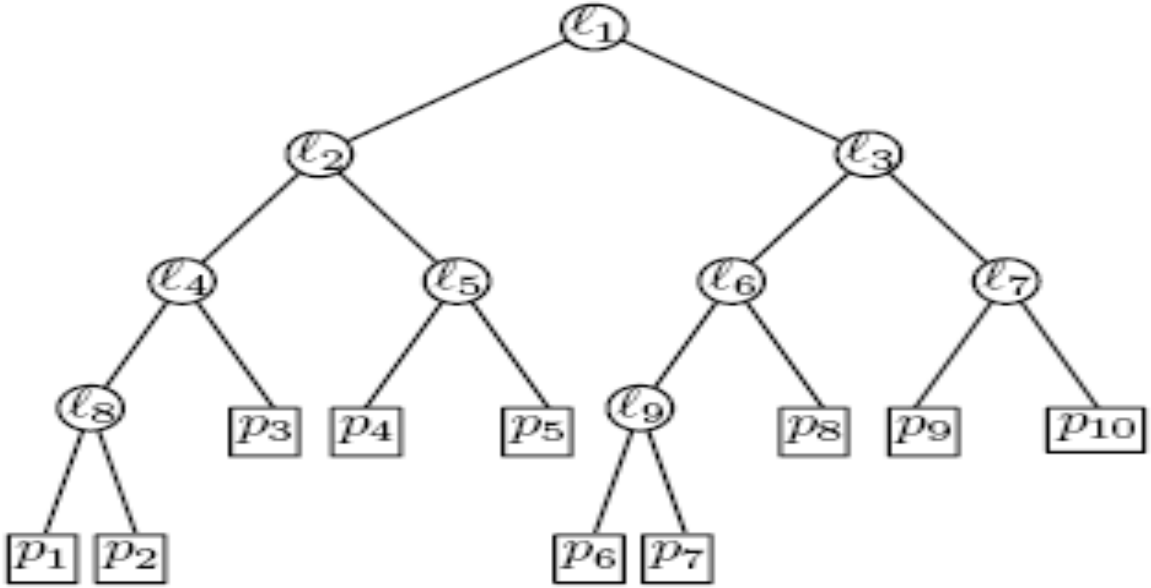
- A binary search tree where every node is a k-dimensional point.

Example: k=2



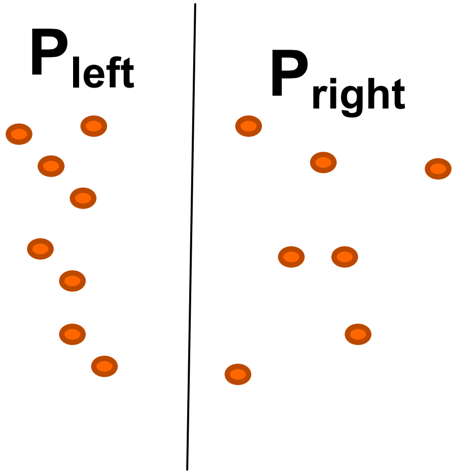
KD Tree…

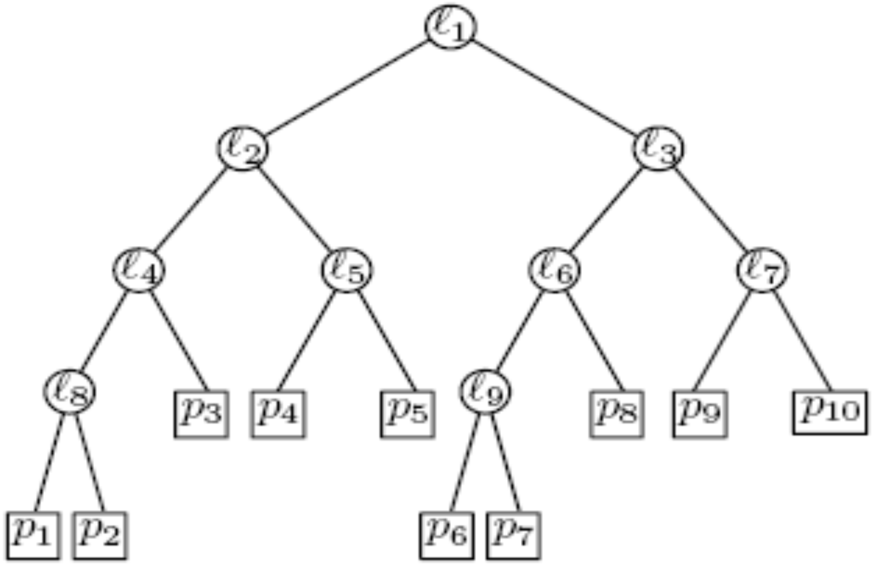
<u>Example:</u> **data stored at the leaves**



KD Tree…

- 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…

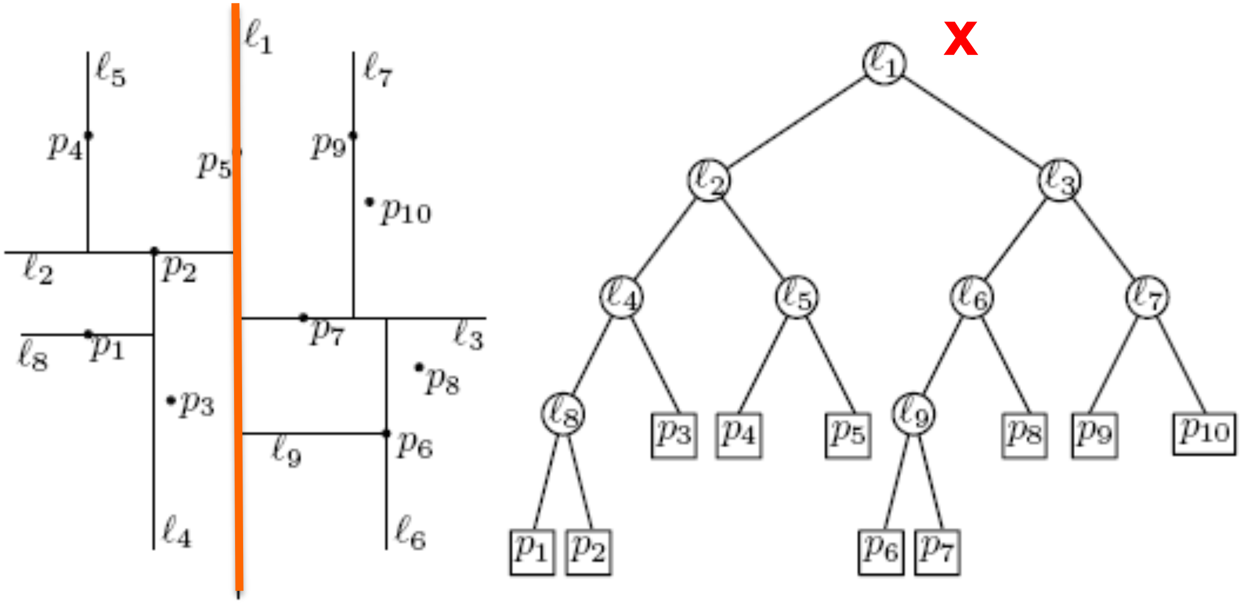
As we move down the tree, we divide the space along alternating (but not always) axis-aligned hyperplanes:

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.

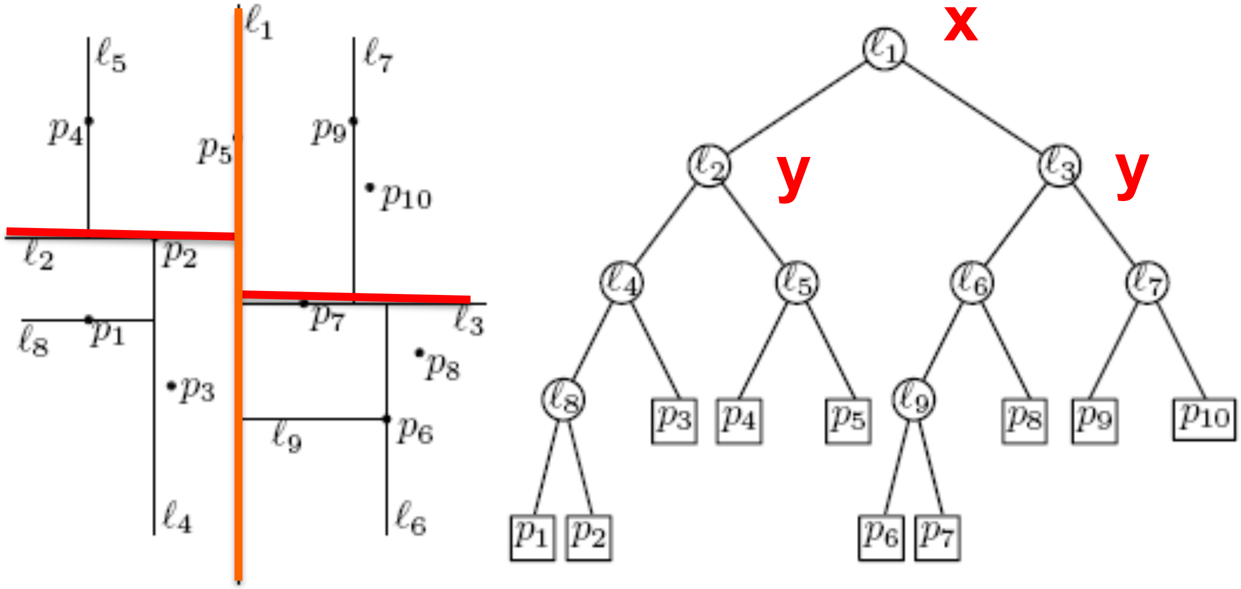
KD Tree - Example

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



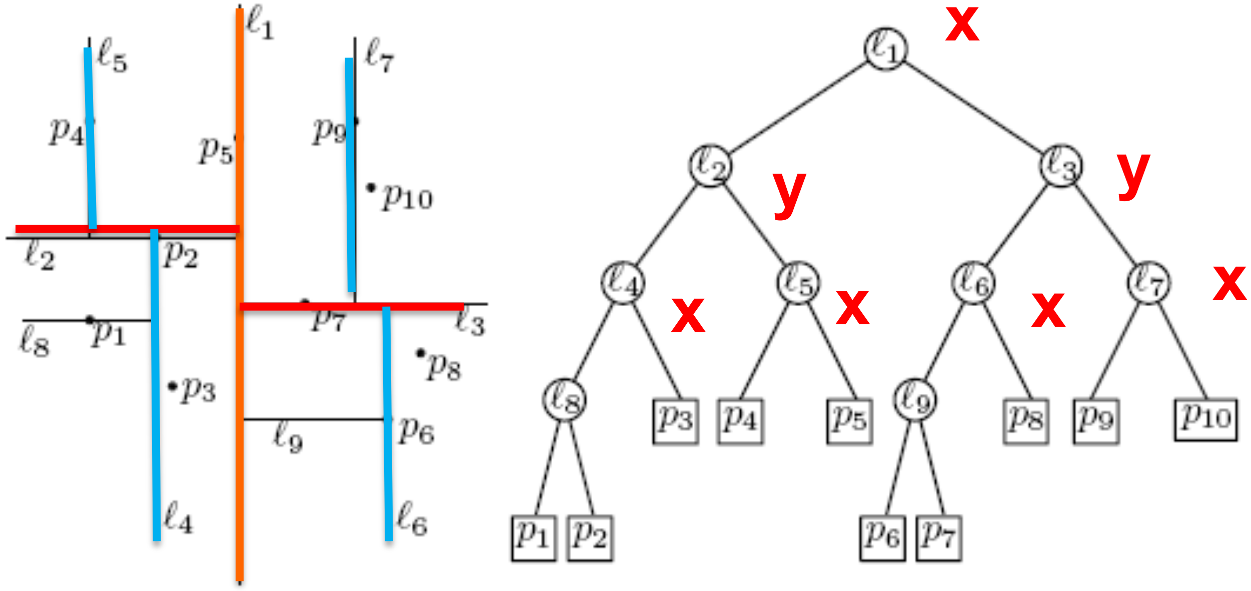
KD Tree - Example

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



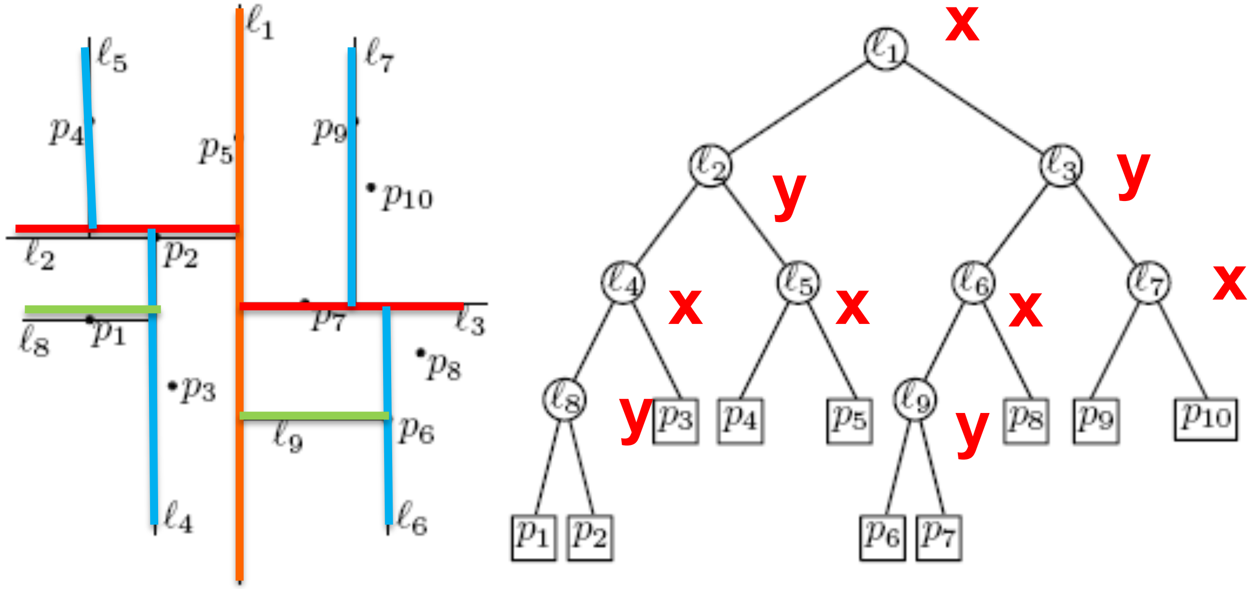
KD Tree - Example

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



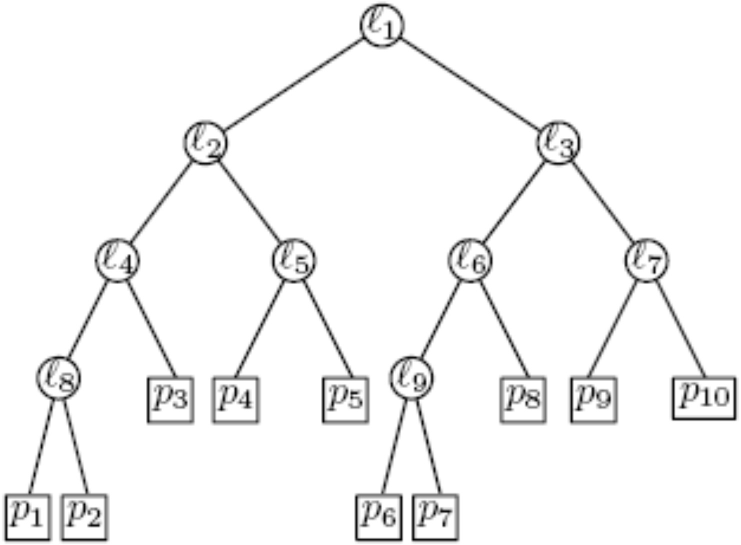
KD Tree - Example

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



Node Structure

- A KD-tree node has 5 fields
  - Splitting axis
  - Splitting value
  - Data
  - Left pointer
  - Right pointer

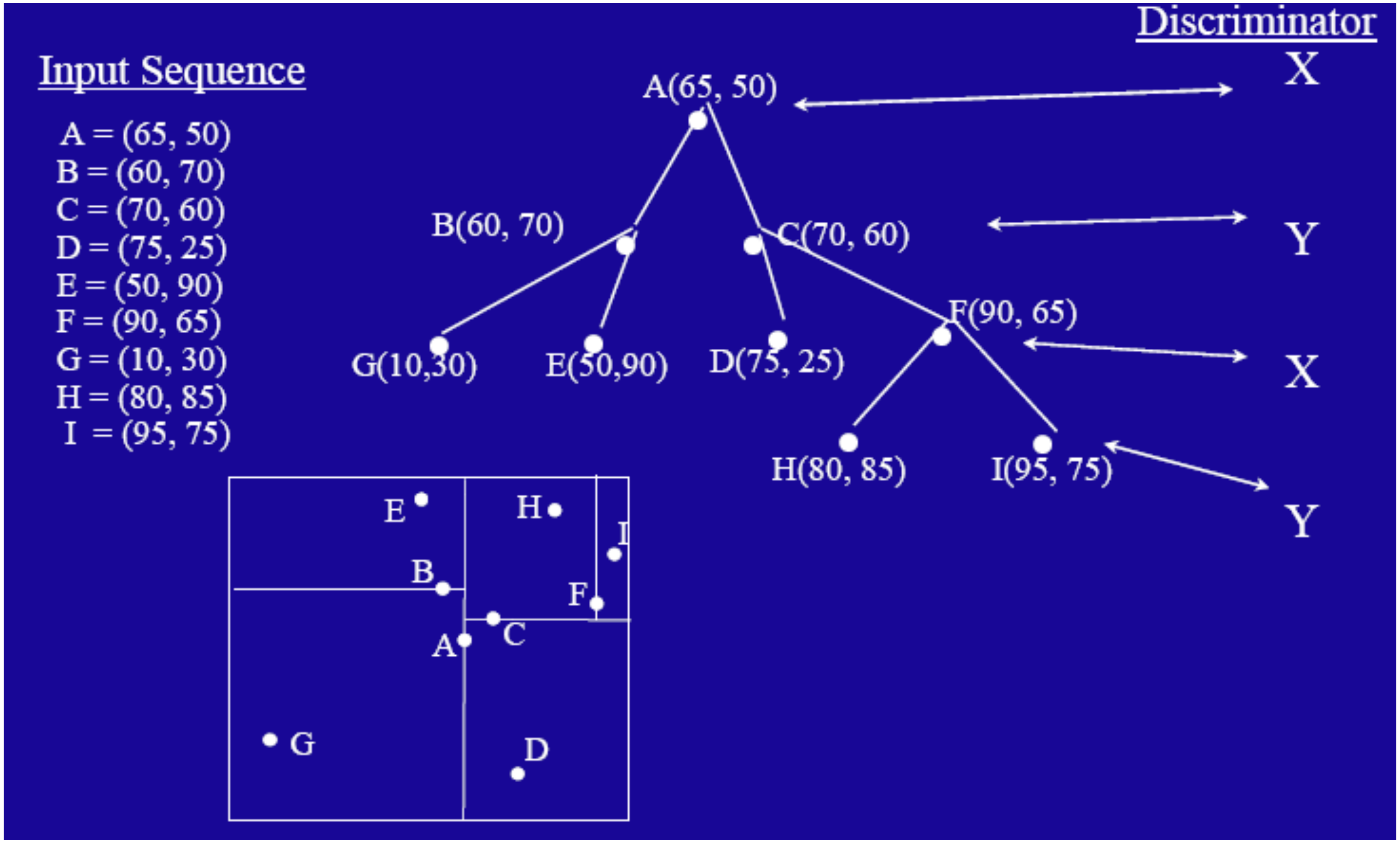


Splitting Strategies

- Divide based on order of point insertion
  - Assumes that points are given one at a time.
- Divide by finding median
  - Assumes all the points are available ahead of time.
- Divide perpendicular to the axis with widest spread
  - Split axes might not alternate

… and more!

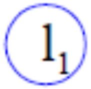
Example - using order of point insertion
(data stored at nodes)

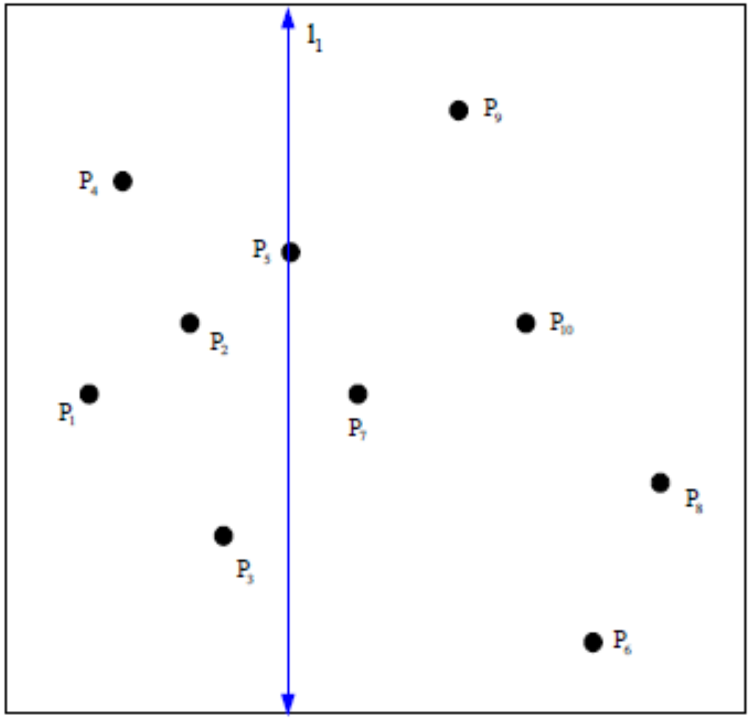


Input Sequence

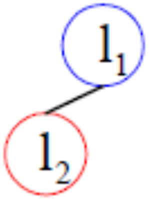
A = (65, 50)
B = (60, 70)
C = (70, 60)
D = (75, 25)
E = (50, 90)
F = (90, 65)
G = (10, 30)
H = (80, 85)
I = (95, 75)

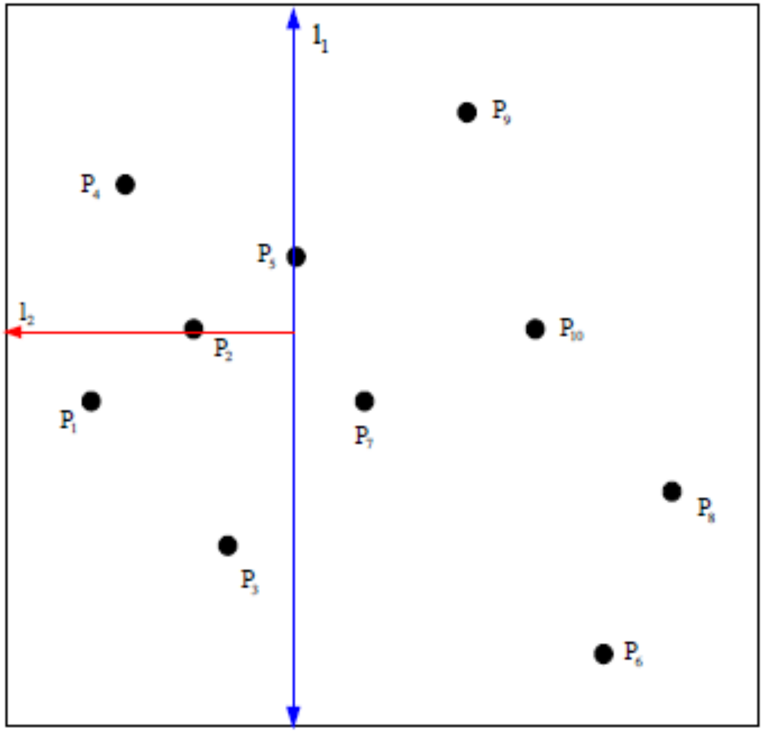
Example - using median (data stored at the leaves)



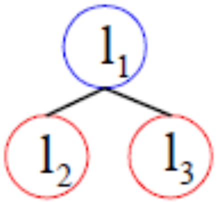


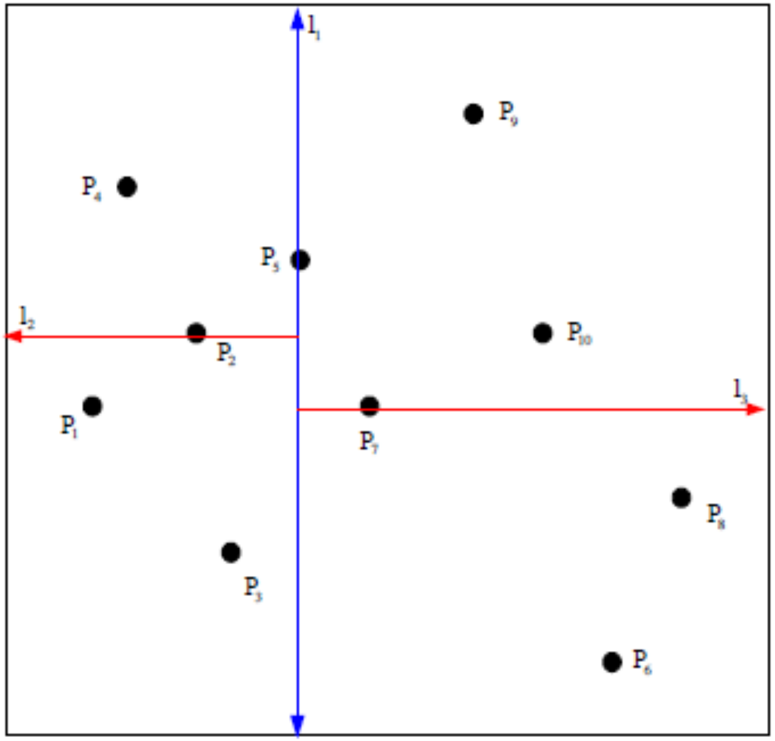
Example - using median (data stored at the leaves)



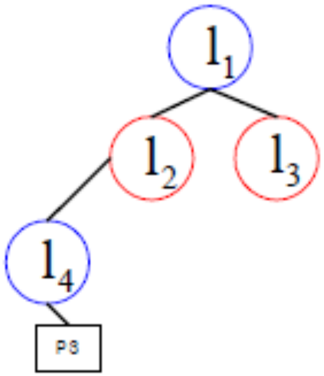


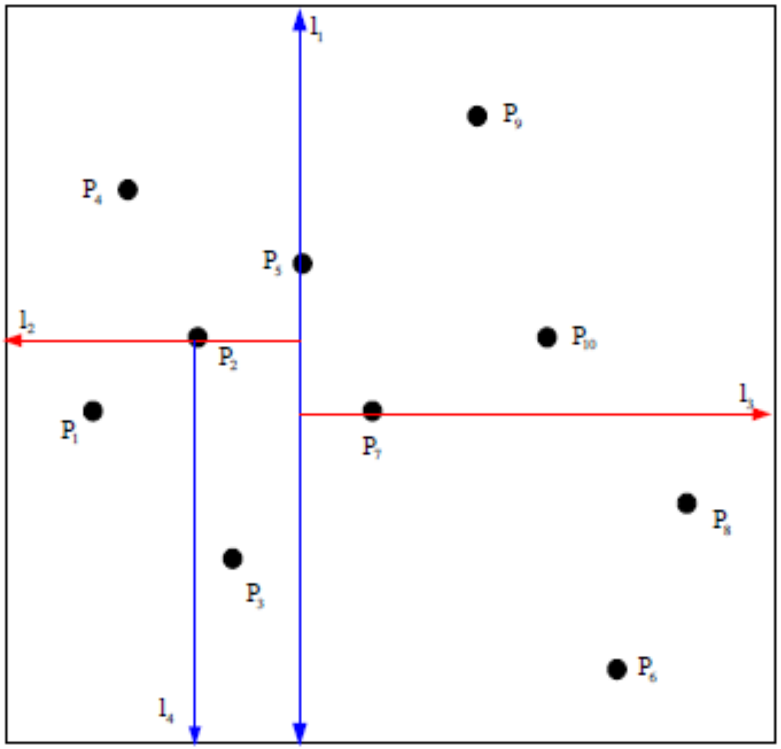
Example - using median (data stored at the leaves)



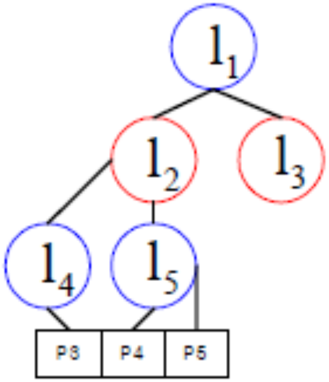


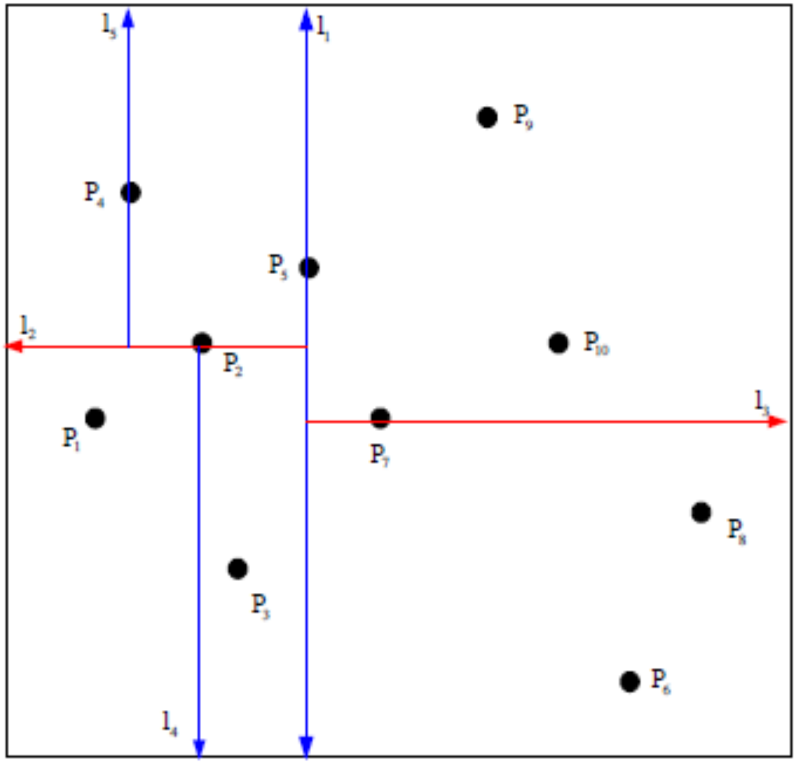
Example - using median (data stored at the leaves)



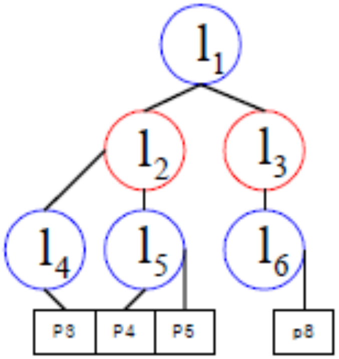


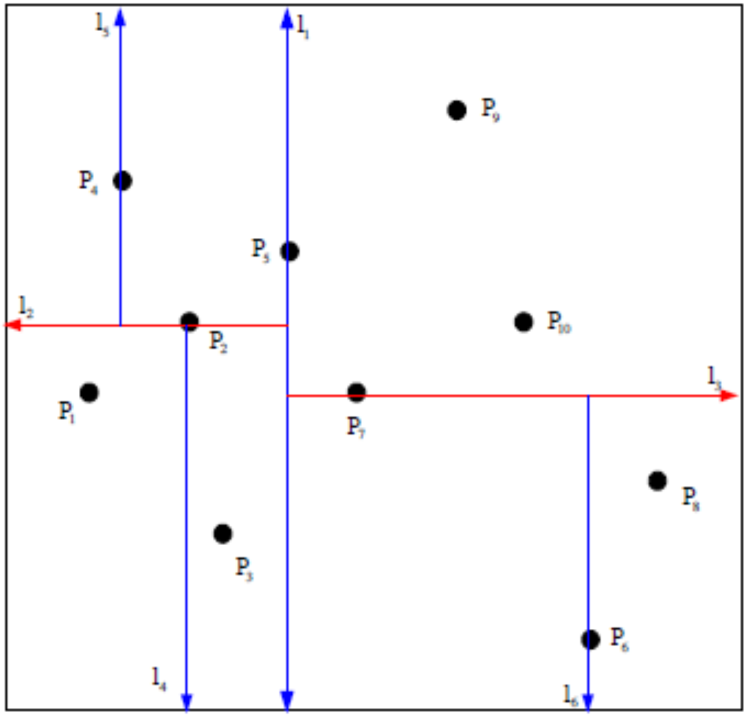
Example - using median (data stored at the leaves)



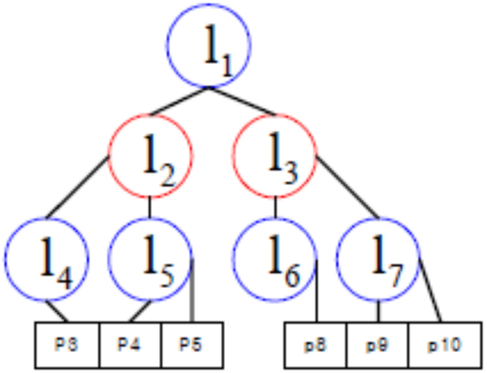


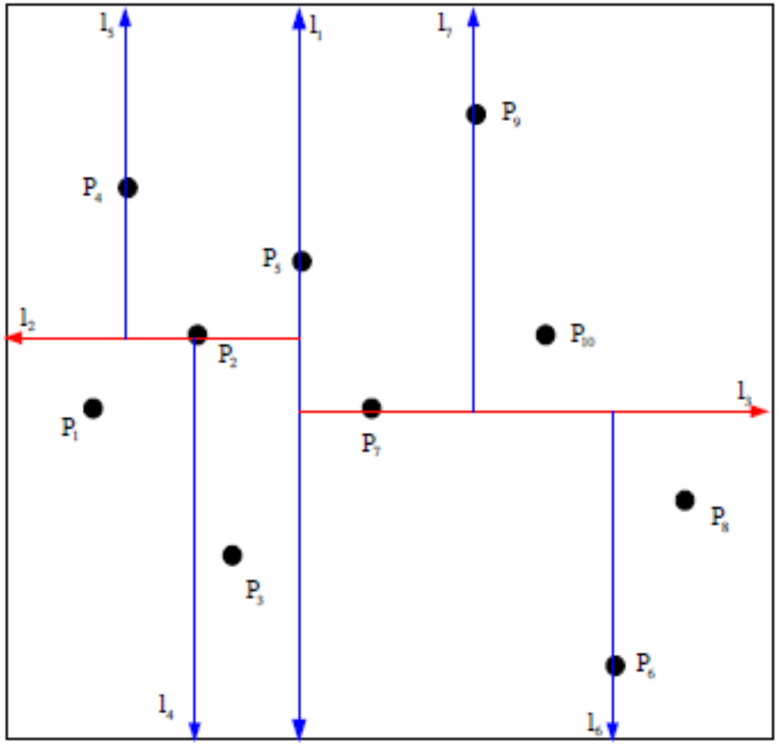
Example – using median (data stored at the leaves)



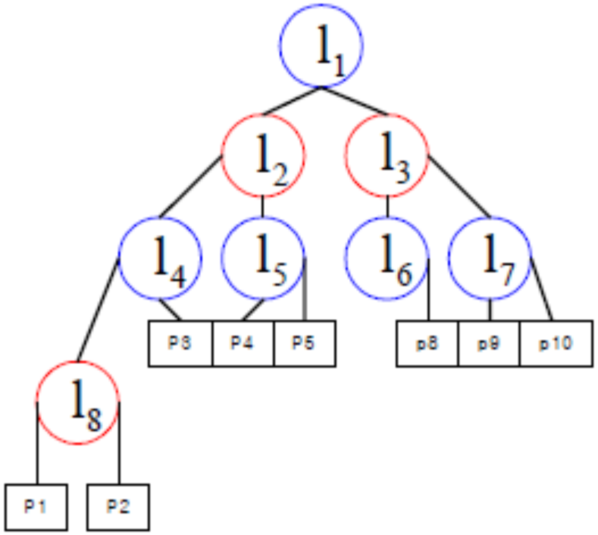


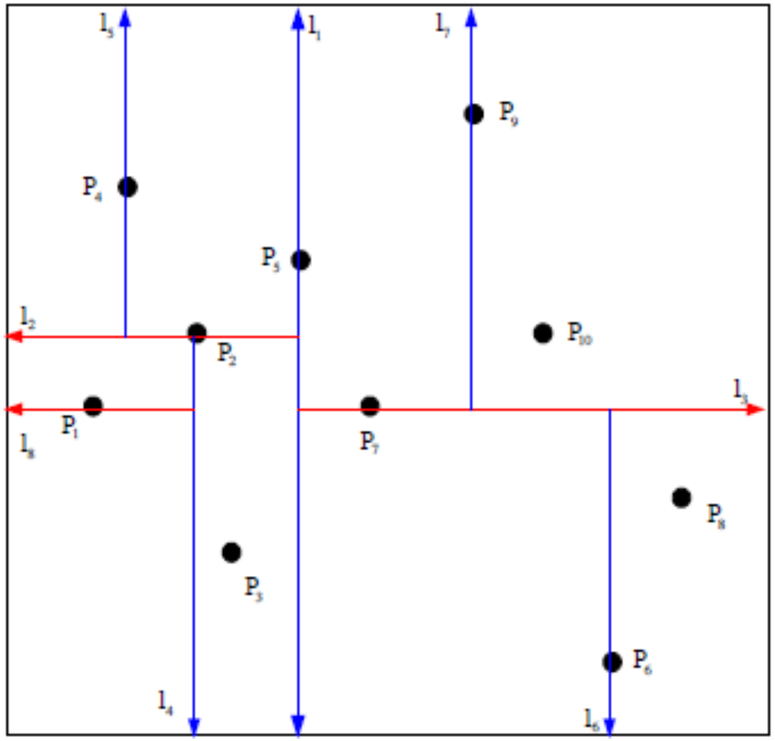
Example - using median (data stored at the leaves)



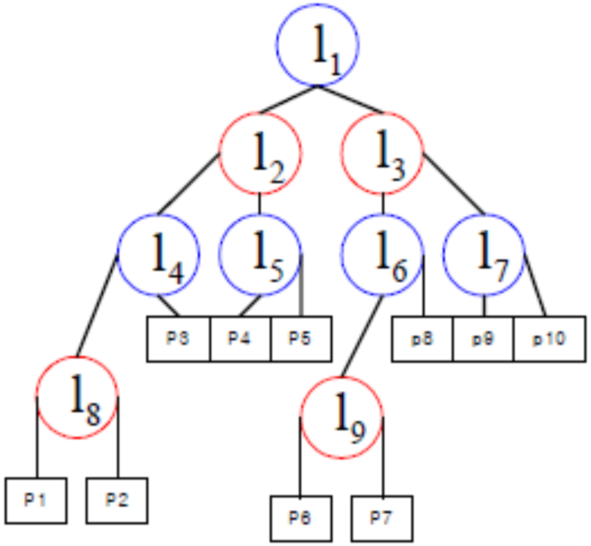


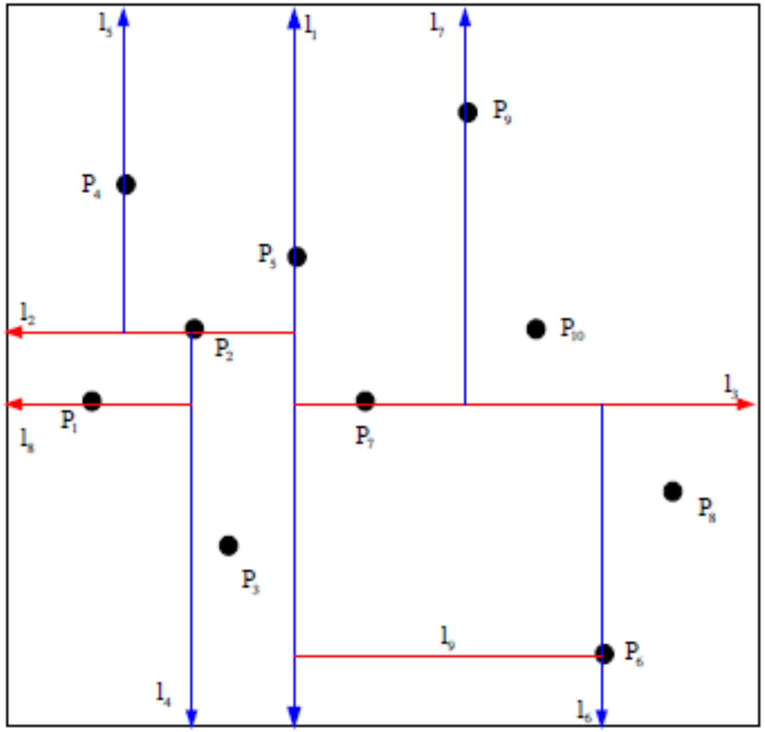
Example - using median (data stored at the leaves)





Example – using median (data stored at the leaves)





Another Example - using median

Students relation (name, age, GPA):
Mike, 25, 3.9
Clayton, 30, 3.8
Terri, 29, 2.5
Debra, 42, 3.7
Dan, 25, 3.75
Mark, 22, 3.7
Julia, 24, 4.0
Arnold, 22, 3.9
Zeke, 23, 3.8

Mike	25	3.9		
Clayton	30	3.8		
Terri	29	2.5		
Debra	42	3.7		
Dan	25	3.75		
Mark	22	3.7		
Julia	24	4.0		
Arnold	22	3.9		
Zeke	23	3.8		

Another Example - using median

Mike	25	3.9		
Clayton	30	3.8		
Terri	29	2.5		
Debra	42	3.7		
Dan	25	3.75		
Mark	22	3.7		
Julia	24	4.0		
Arnold	22	3.9		
Zeke	23	3.8		

Discriminator order: Name, age, GPA, name, age, GPA, ...

Data stored in the node

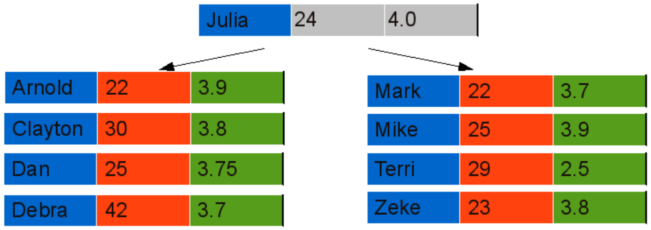
Another Example - using median

Mike	25	3.9		
Clayton	30	3.8		
Terri	29	2.5		
Debra	42	3.7		
Dan	25	3.75		
Mark	22	3.7		
Julia	24	4.0		
Arnold	22	3.9		
Zeke	23	3.8		

Discriminator order: Name, age, GPA, name, age, GPA, ...

Arnold
Clayton
Dan
Debra
Julia ← Median
Mark
Mike
Terri
Zeke

Another Example - using median



Julia	24	4.0

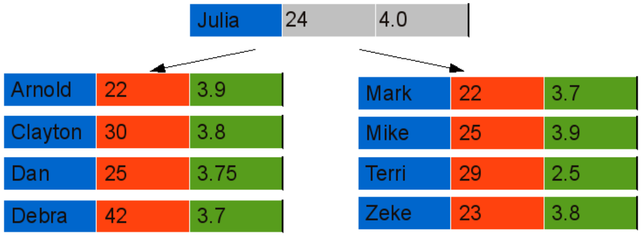
Arnold	22	3.9
Clayton	30	3.8
Dan	25	3.75
Debra	42	3.7

Mark	22	3.7
Mike	25	3.9
Terri	29	2.5
Zeke	23	3.8

Arnold
Clayton
Dan
Debra
Julia ← Median
Mark
Mike
Terri
Zeke

Discriminator order: Name, age, GPA, name, age, GPA, ...

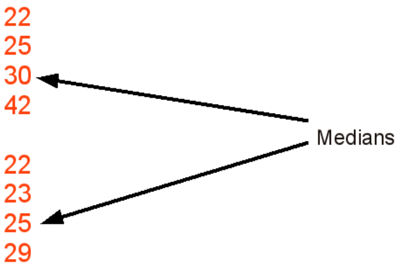
Another Example - using median



Julia	24	4.0

Arnold	22	3.9
Clayton	30	3.8
Dan	25	3.75
Debra	42	3.7

Mark	22	3.7
Mike	25	3.9
Terri	29	2.5
Zeke	23	3.8



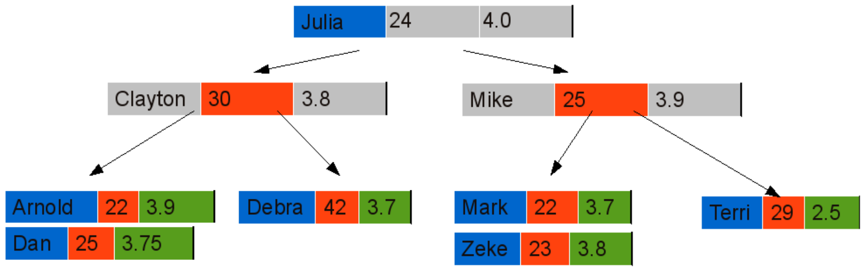
22
25
30
42

22
23
25
29

Medians

Discriminator order: Name, age, GPA, name, age, GPA, ...

Another Example - using median



Julia 24 4.0

Clayton 30 3.8

Mike 25 3.9

Arnold 22 3.9

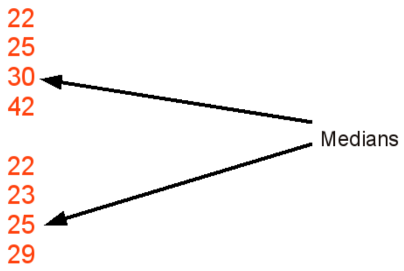
Dan 25 3.75

Debra 42 3.7

Mark 22 3.7

Zeke 23 3.8

Terri 29 2.5



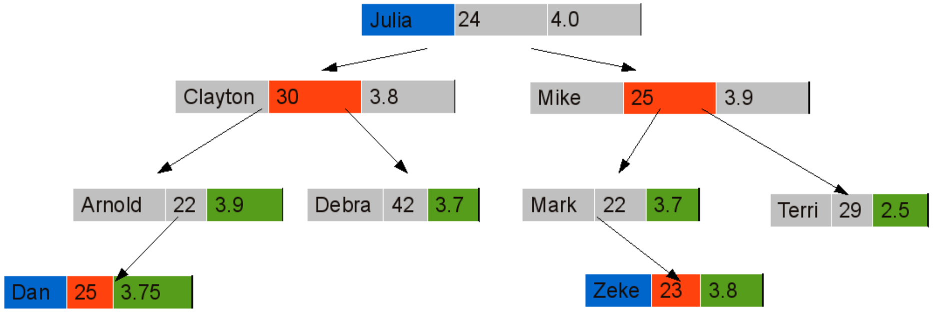
22
25
30
42

22
23
25
29

Medians

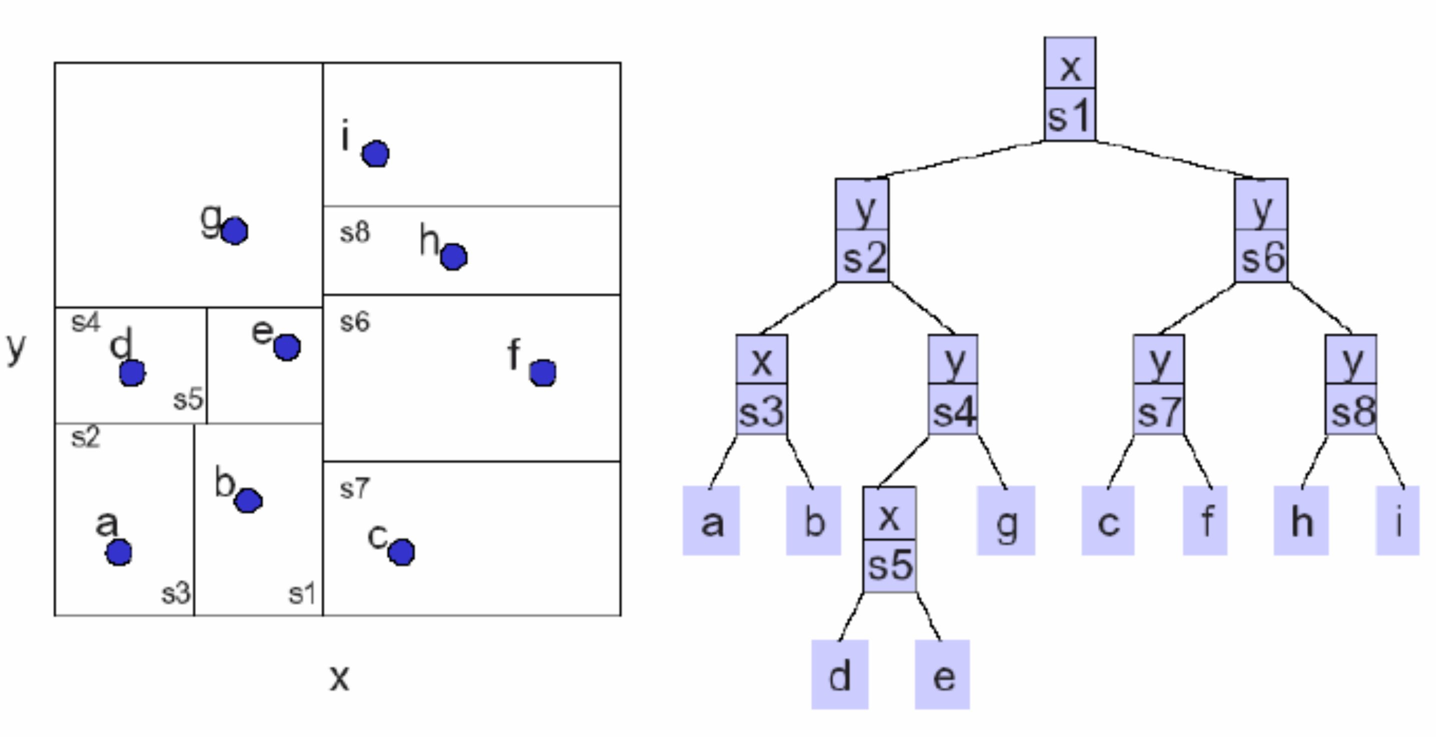
Discriminator order: Name, age, GPA, name, age, GPA, ...

Another Example - using median



Discriminator order: Name, age, GPA, name, age, GPA, ...

Example – Split perpendicular to the axis with widest spread (data stored in the leaves)

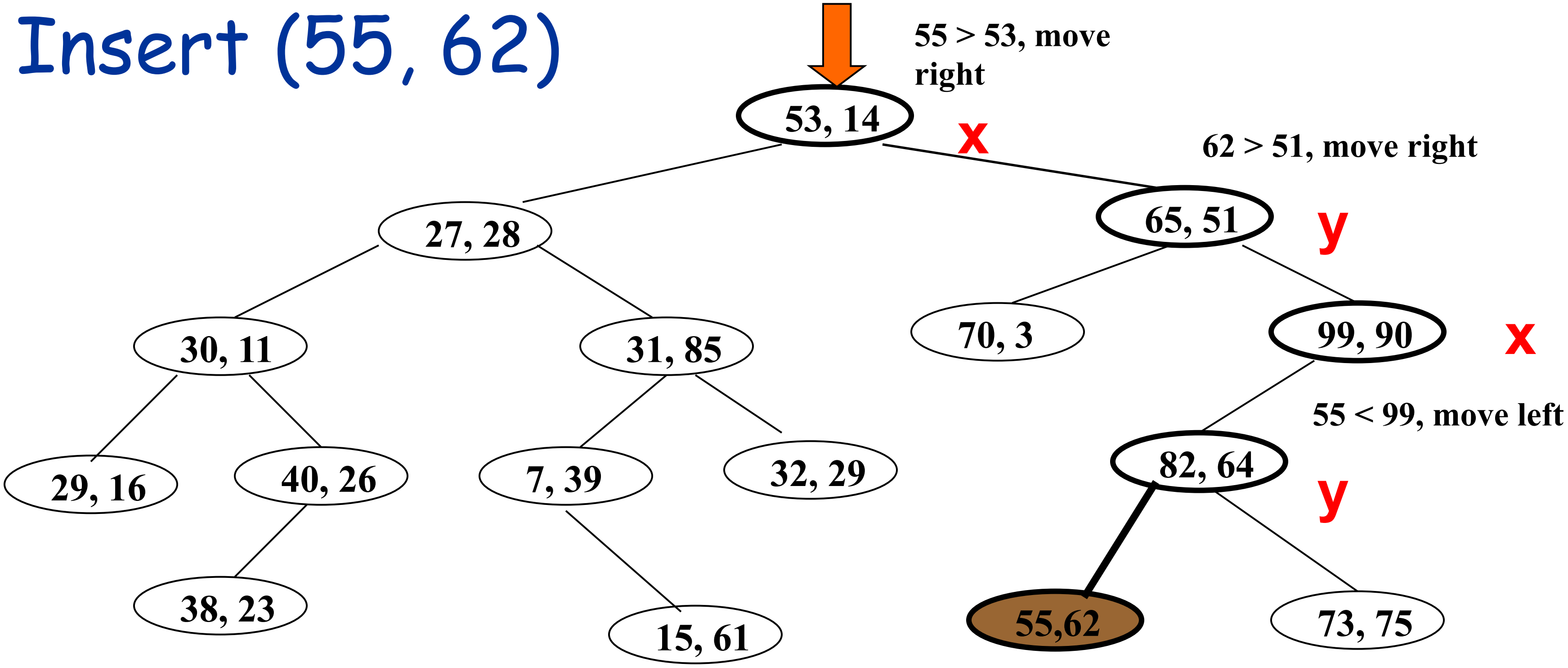


KD Tree…

- Let’s discuss
  - Insert
  - Delete
  - Search

Insert new data

Insert (55, 62)



62 < 64, move left

Null pointer, attach

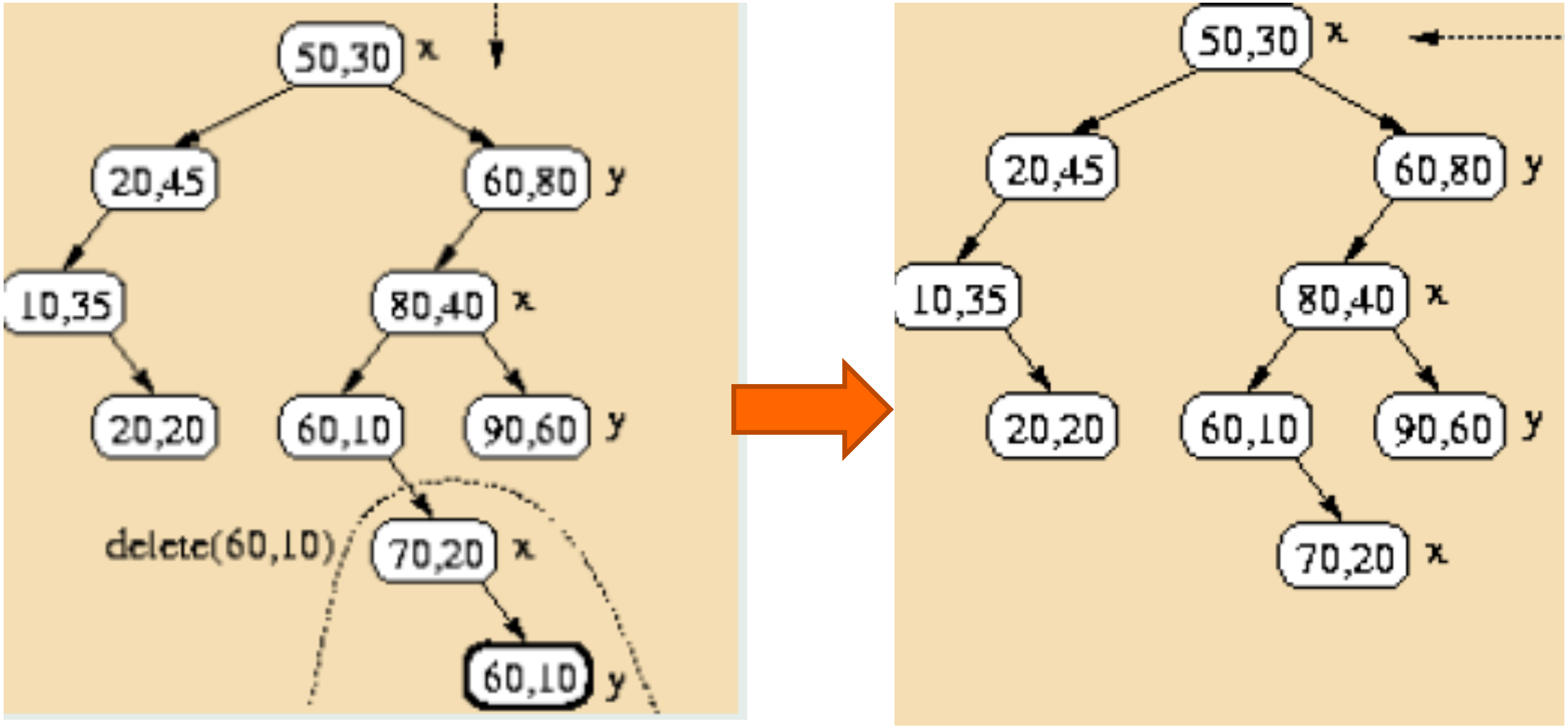
Delete data

- Suppose we need to remove **p** = (a, b)
  - Find node *t* which contains **p**
  - If *t* is a leaf node, replace it by null
  - Otherwise, find a replacement node **r** = (c, d) – see below!
  - Replace (a, b) by (c, d)
  - Remove **r**
- Finding the replacement **r** = (c, d)
  - If **t** has a right child, use the successor*
  - Otherwise, use node with minimum value* in the left subtree

***(depending on what axis the node discriminates)**

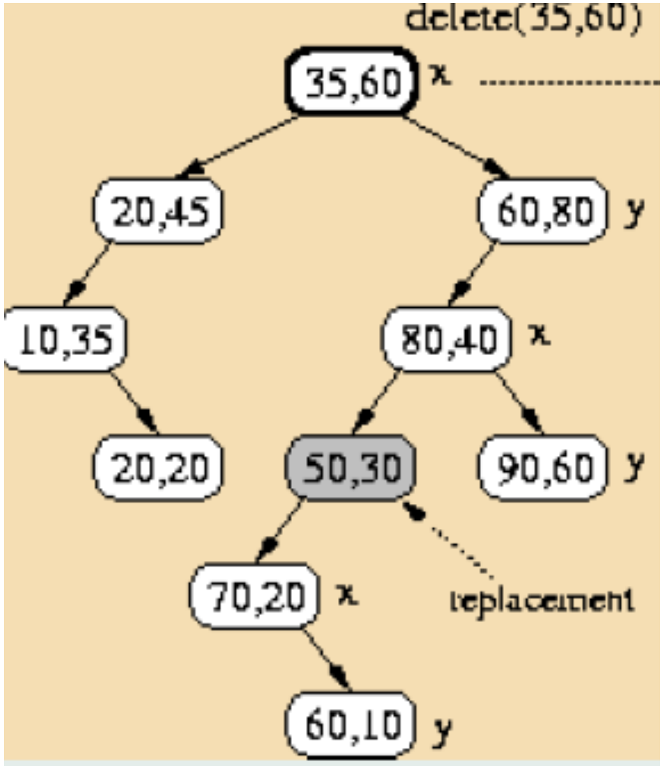
Delete data...

Delete (60,10)

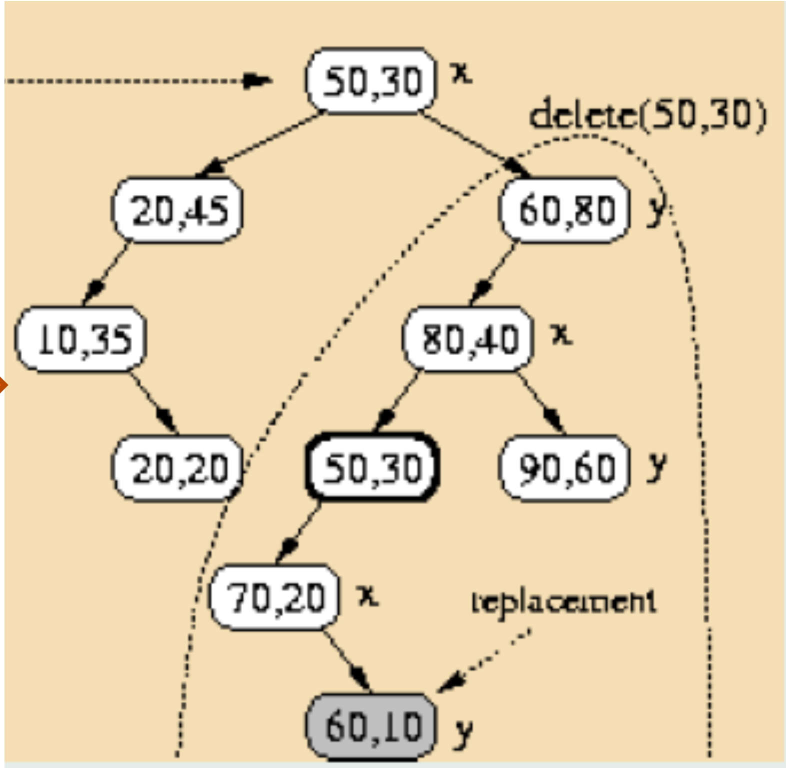


Delete data…

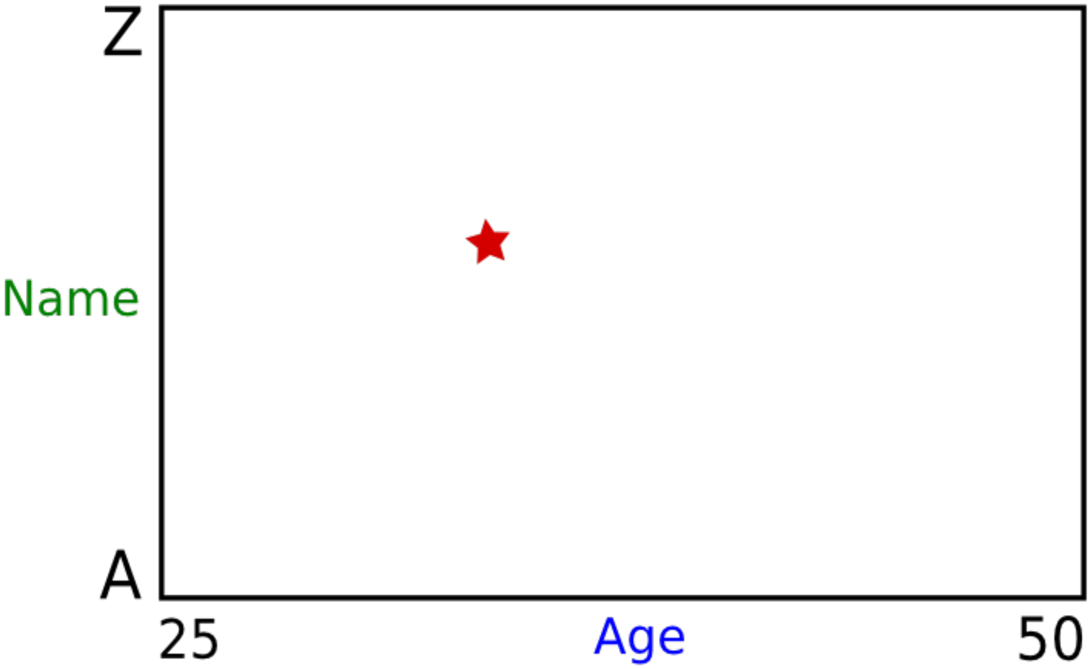
Delete (35, 60)



Delete (50, 30)

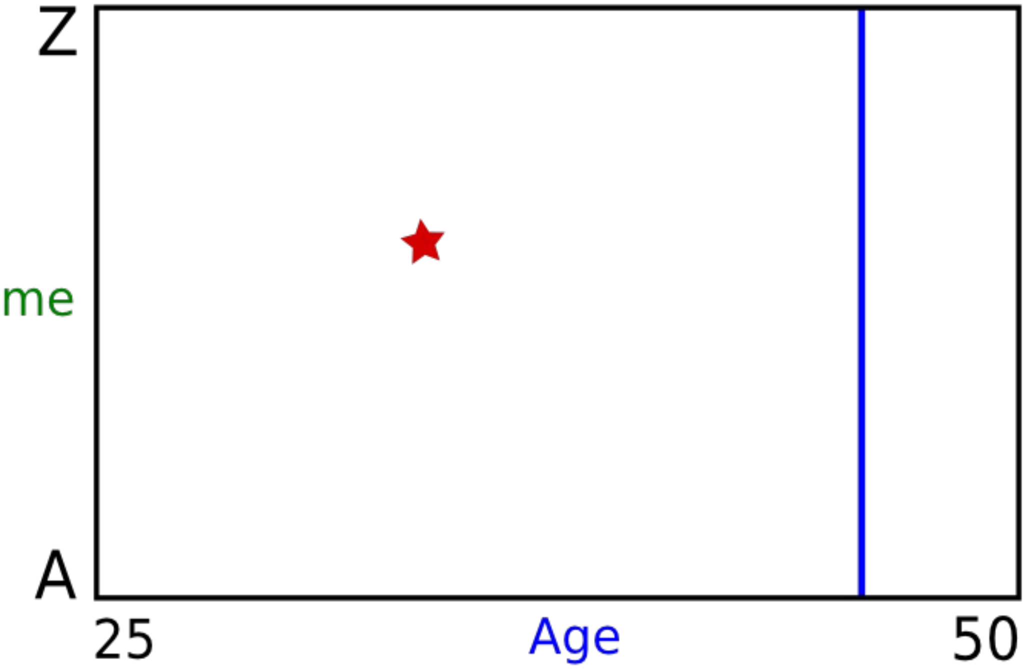


KD Tree - Exact Search



Search key: (32, P)

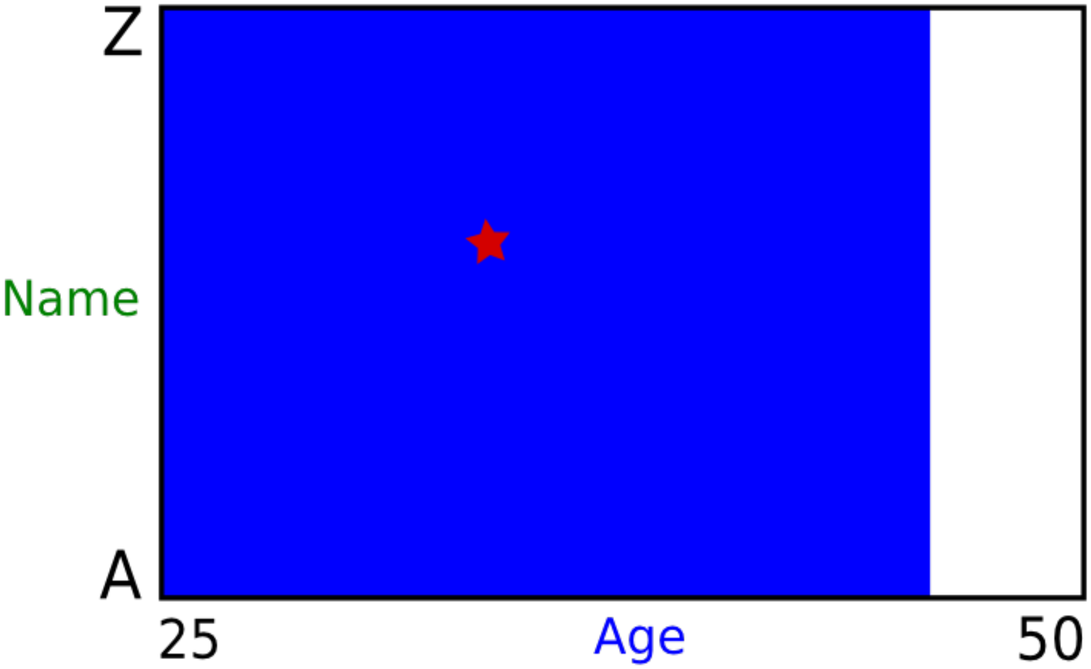
KD Tree - Exact Search



Current node's key: Age=45

Search key: (32, P)

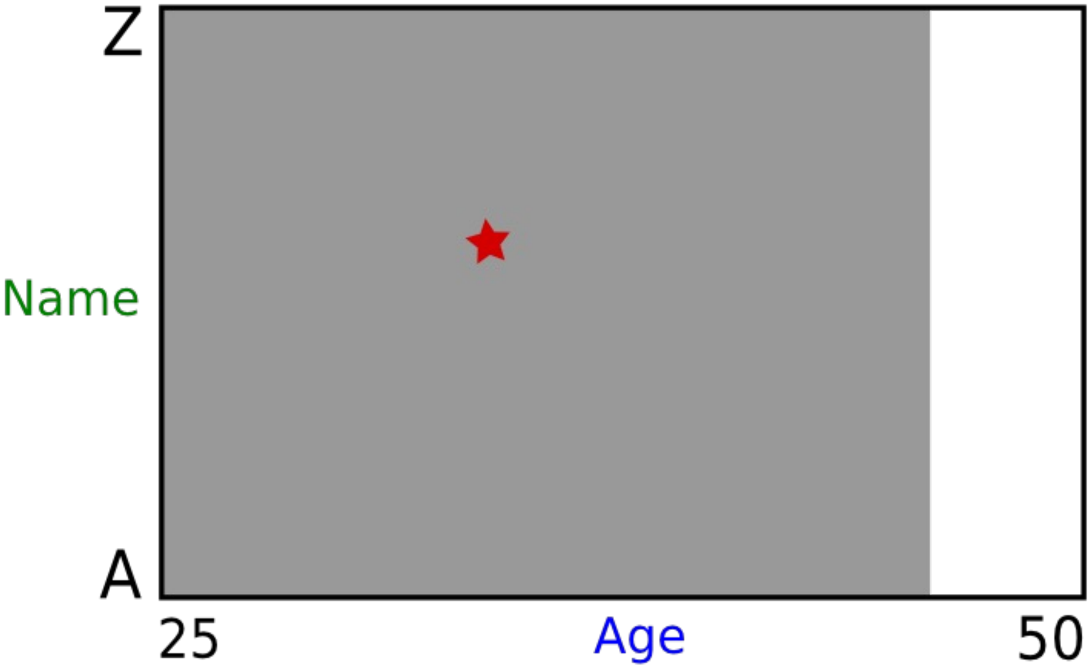
KD Tree - Exact Search



Current node's key: Age=45

Search key: (32, P)

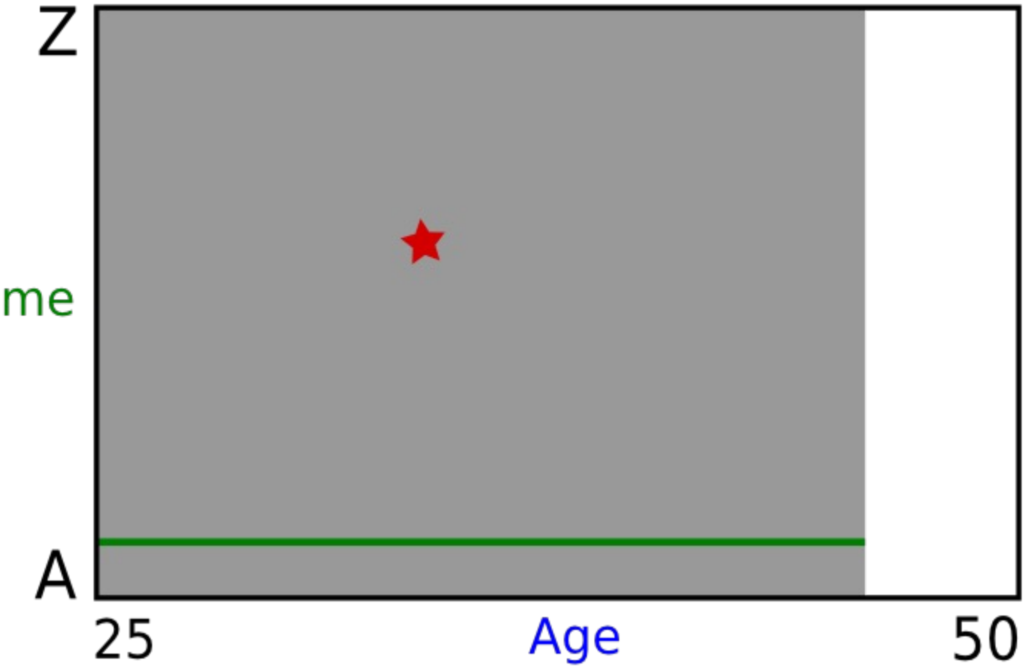
KD Tree - Exact Search



Current node's key: Age=45

Search key: (32, P)

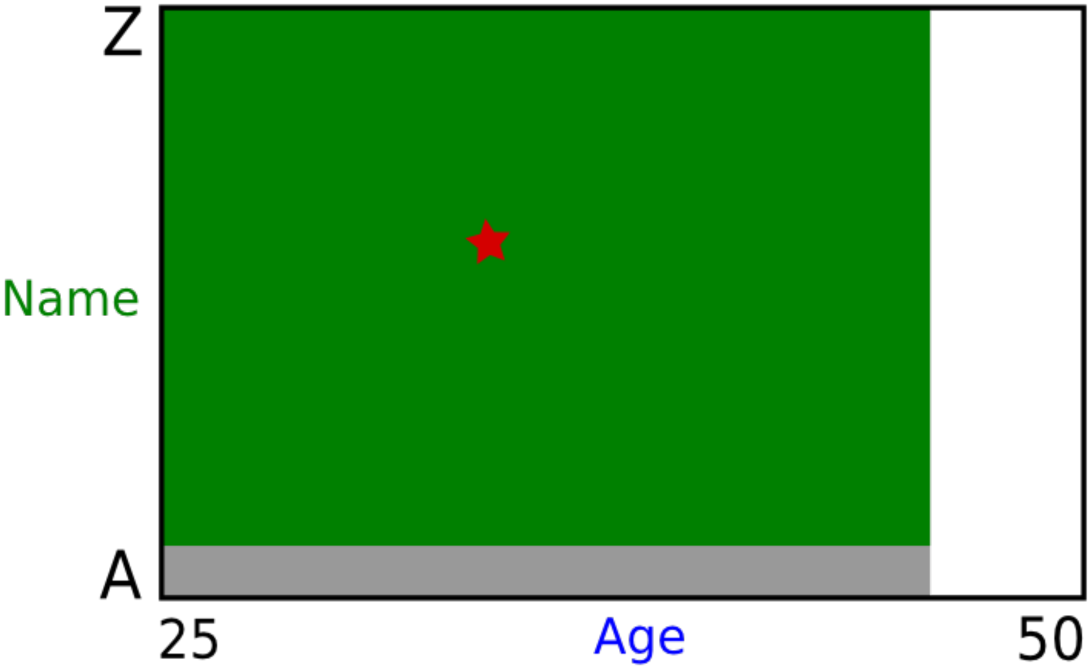
KD Tree - Exact Search



Current node's key: Name=B

Search key: (32, P)

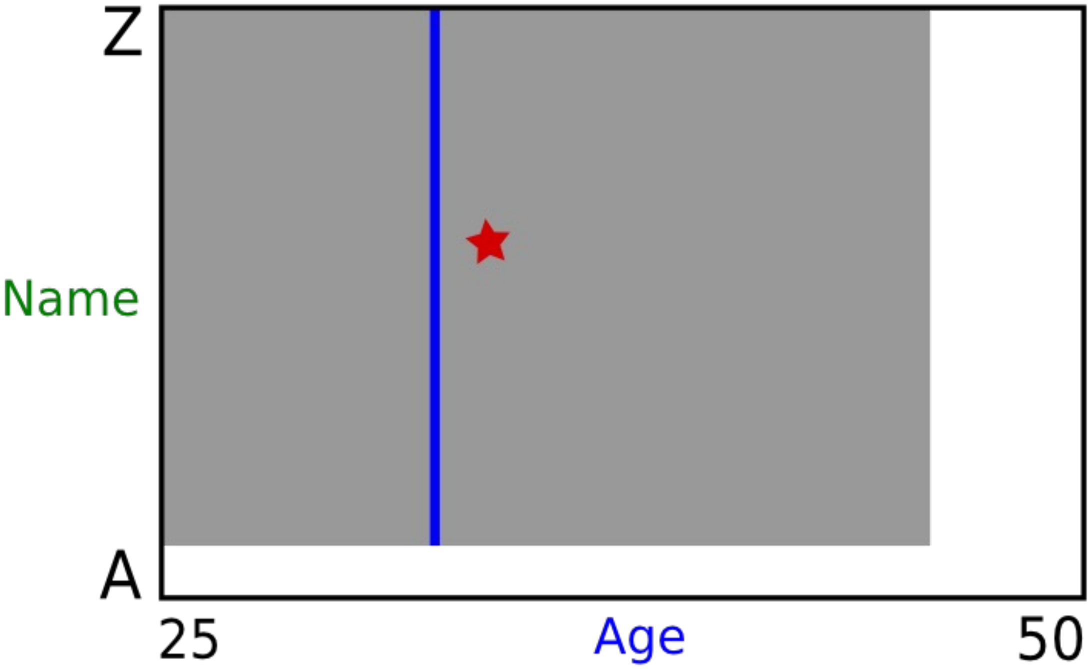
KD Tree - Exact Search



Current node's key: Name=B

Search key: (32, P)

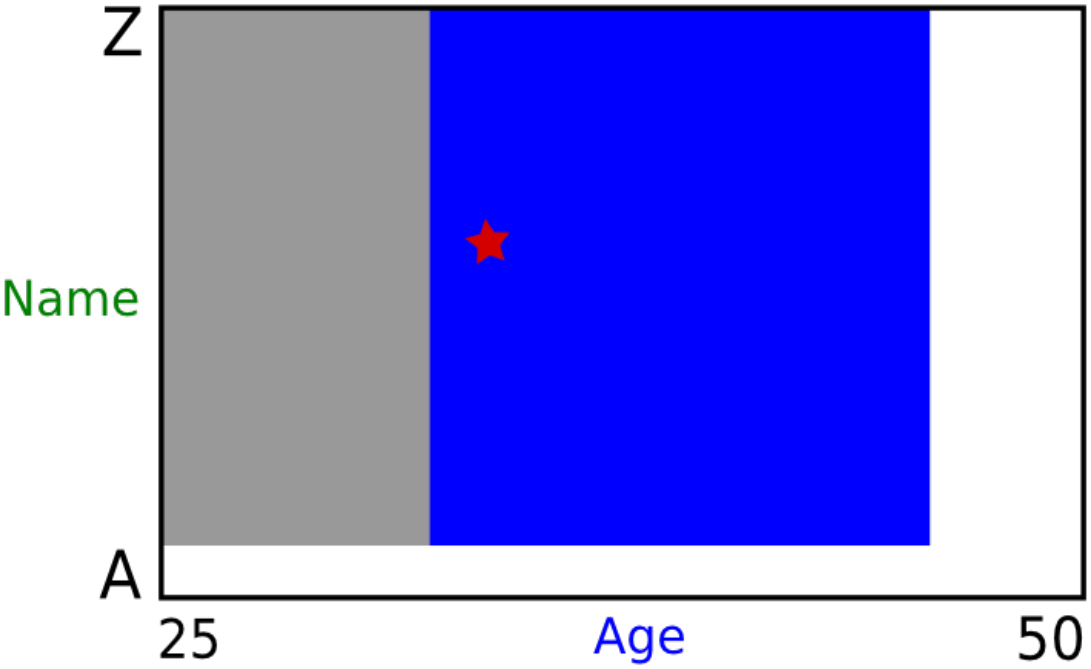
KD Tree - Exact Search



Current node's key: Age=30

Search key: (32, P)

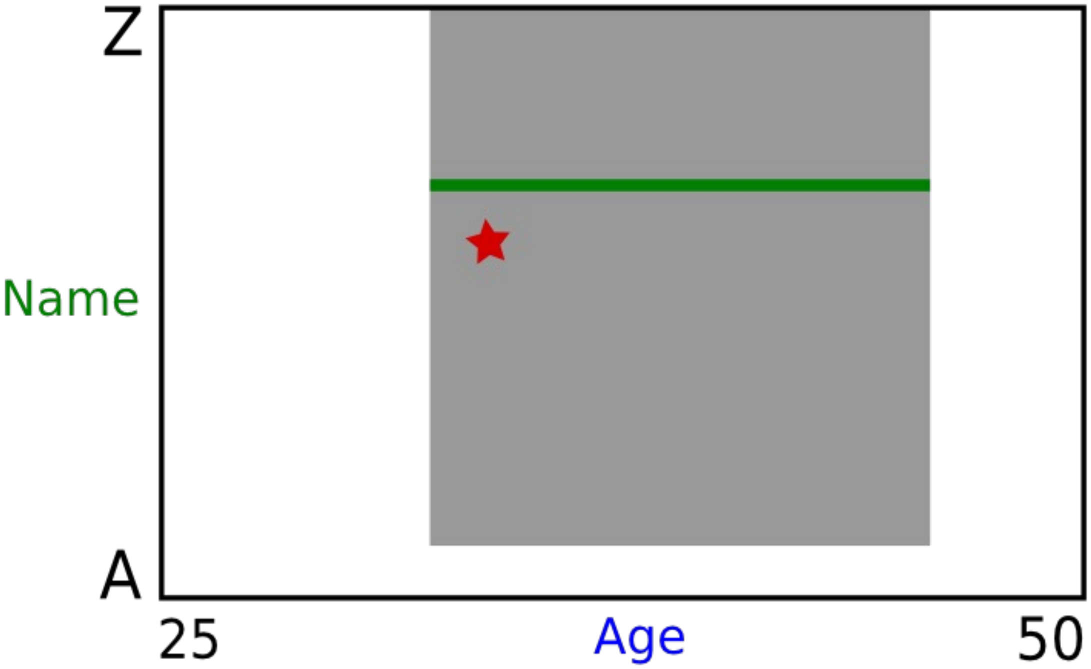
KD Tree - Exact Search



Current node's key: Age=30

Search key: (32, P)

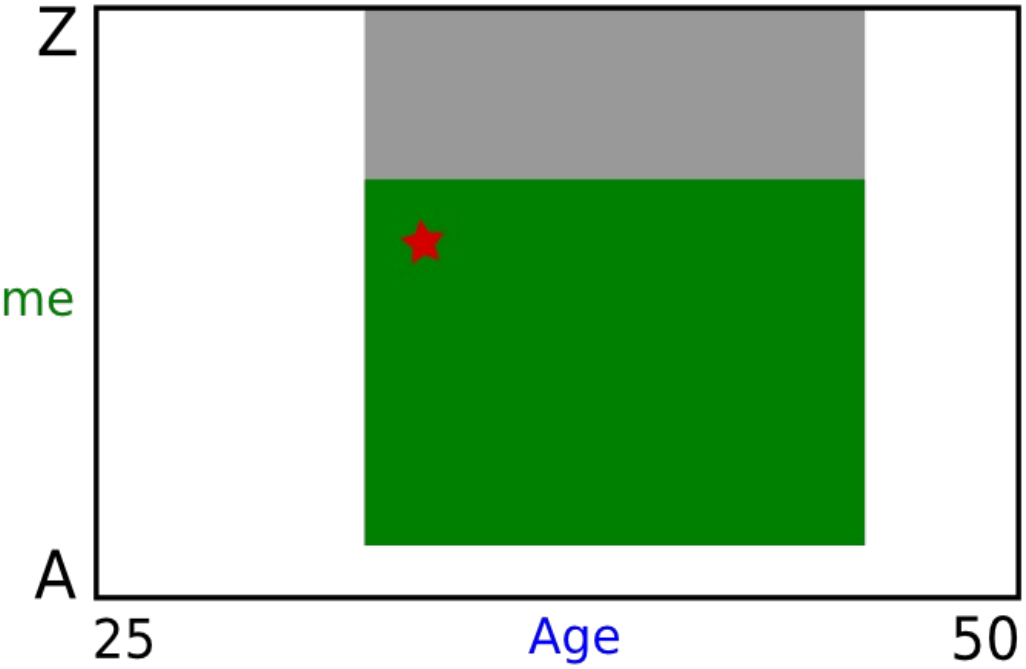
KD Tree - Exact Search



Current node's key: Name=R

Search key: (32, P)

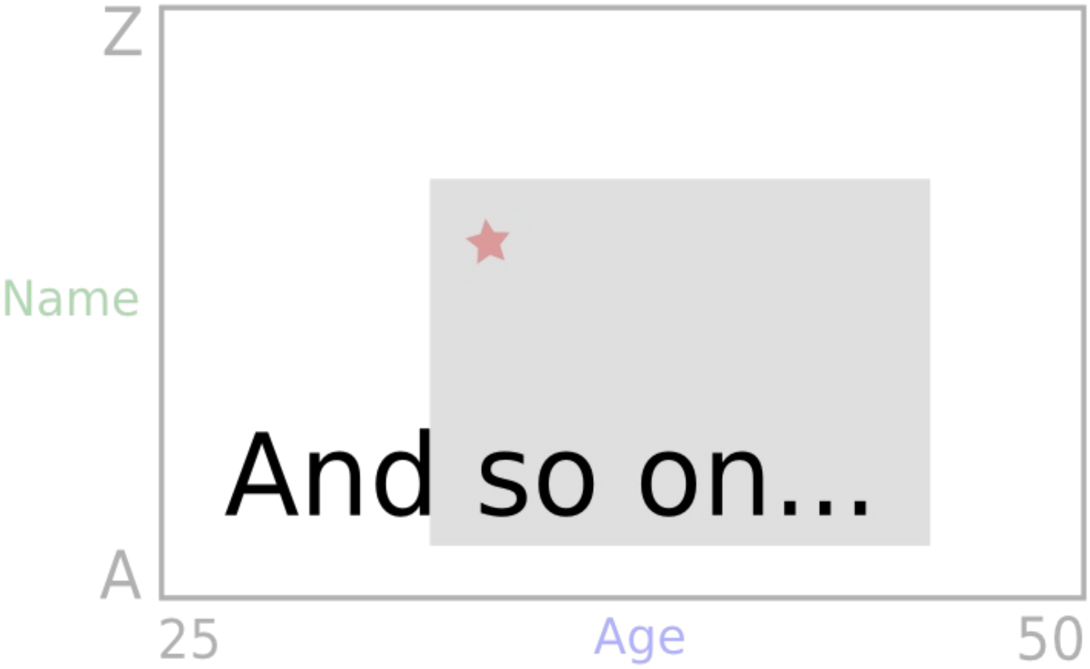
KD Tree - Exact Search



Current node's key: Name=R

Search key: (32, P)

KD Tree - Exact Search



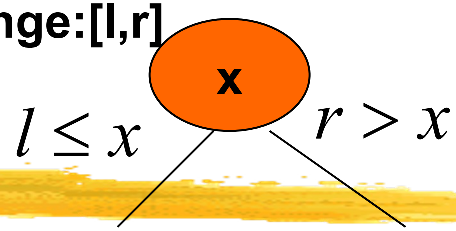
And so on...

Current node's key: Name=R

Search key: (32, P)

KD Tree - Range Search

Range:[l,r]

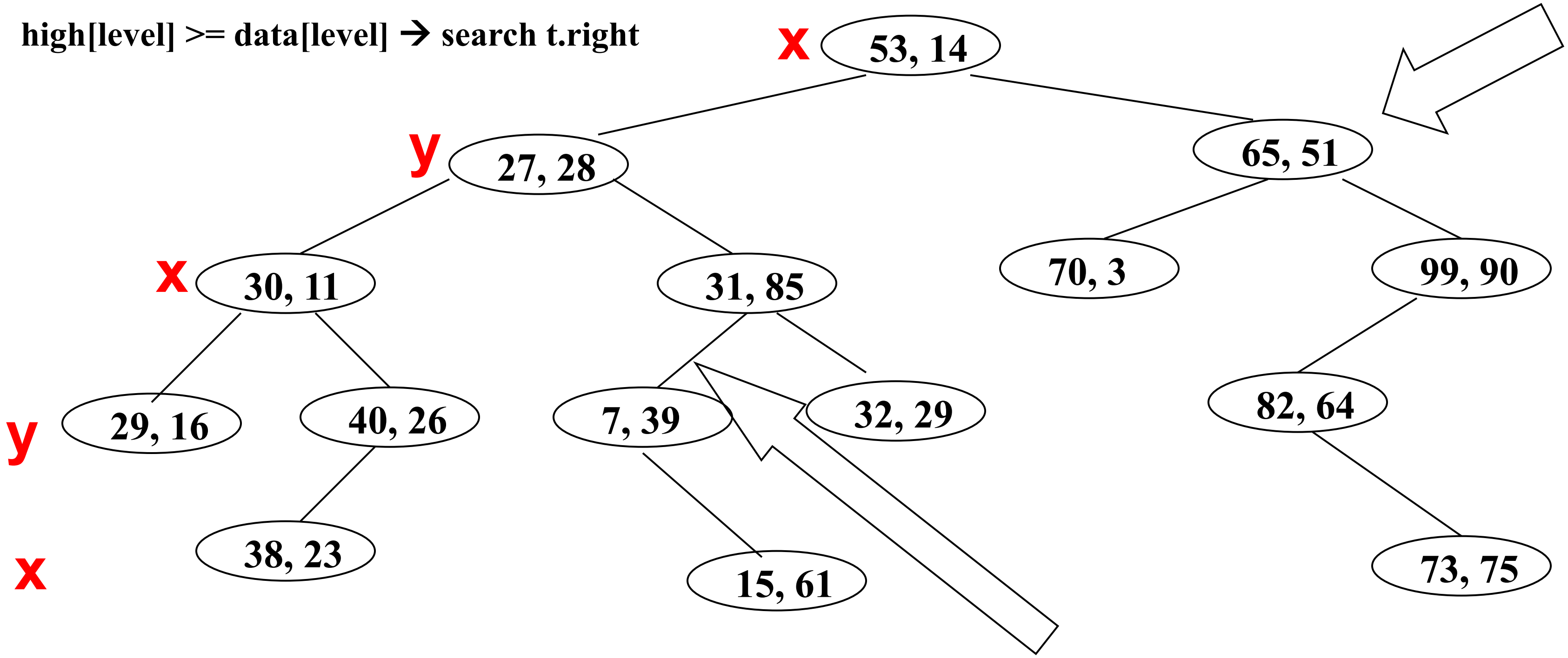


[35, 40] x [23, 30]

In range? If so, print cell

low[level]<=data[level] → search t.left

high[level] >= data[level] → search t.right



low[0] = 35, high[0] = 40;

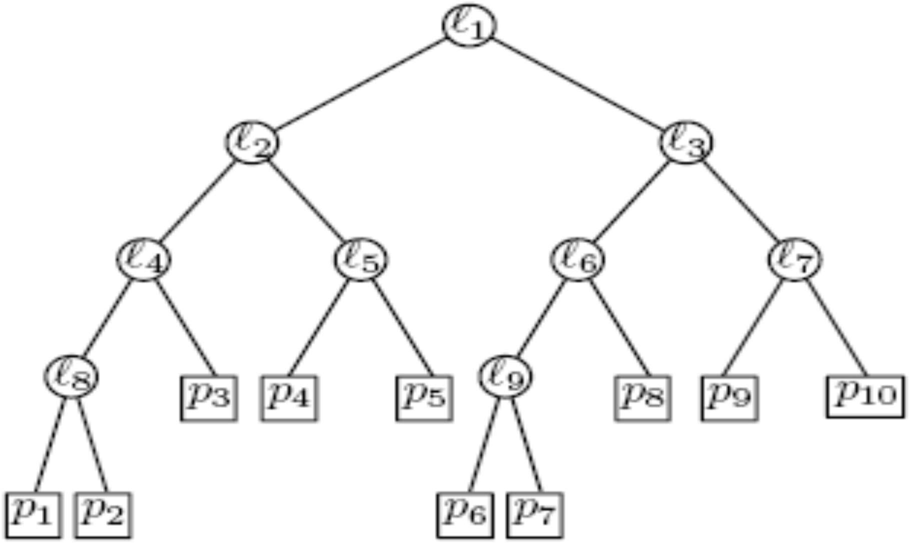
low[1] = 23, high[1] = 30;

This sub-tree is never searched.

Searching is “preorder”. Efficiency is obtained by “pruning” subtrees from the search.

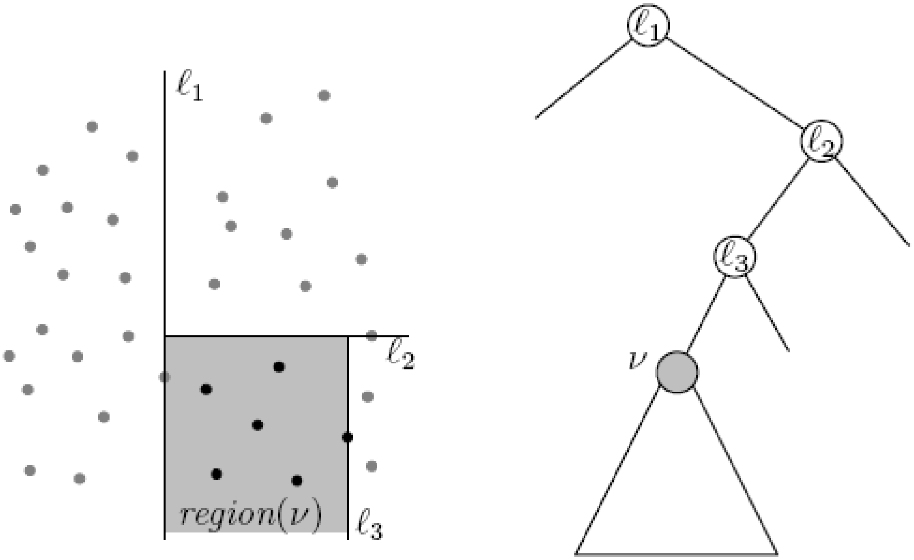
KD Tree - Range Search

- Consider a KD Tree where the data is stored at the leaves, how do we perform range search?



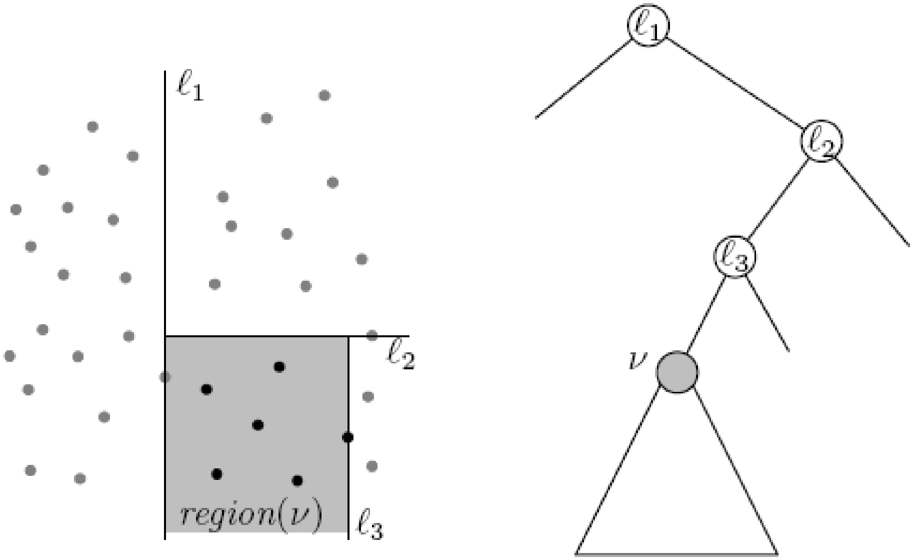
KD Tree - Region of a node

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



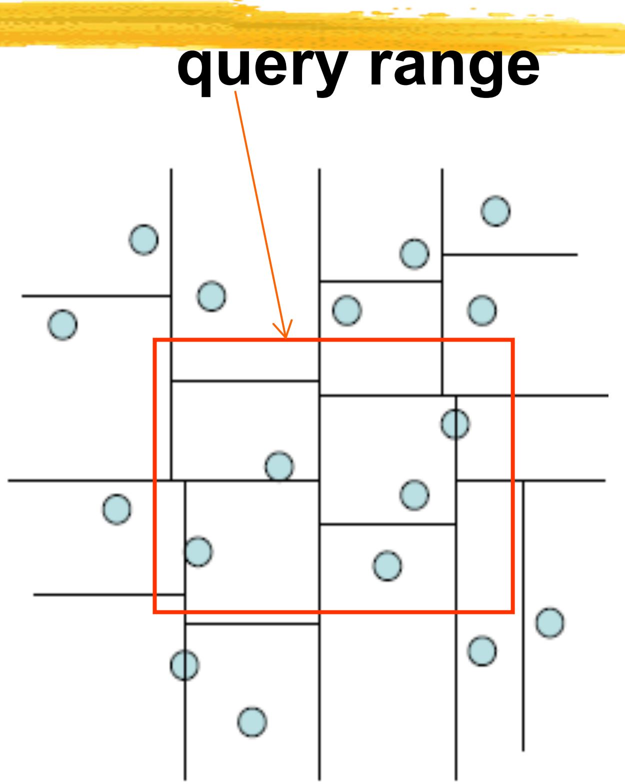
KD Tree - Region of a node

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

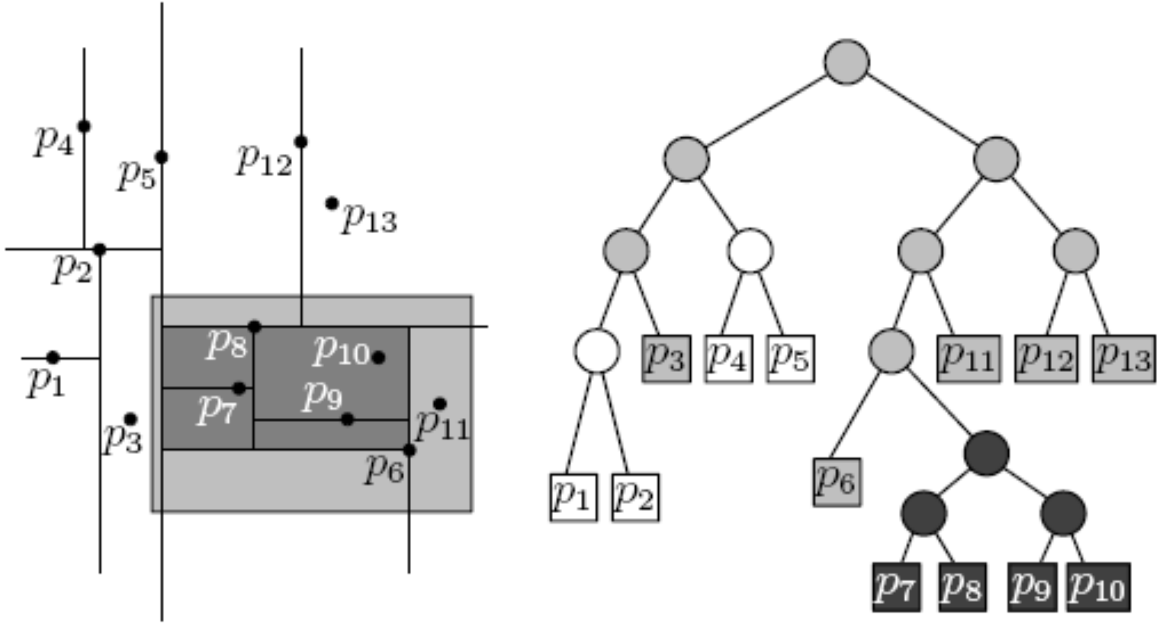


KD Trees - Range Search

- 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.



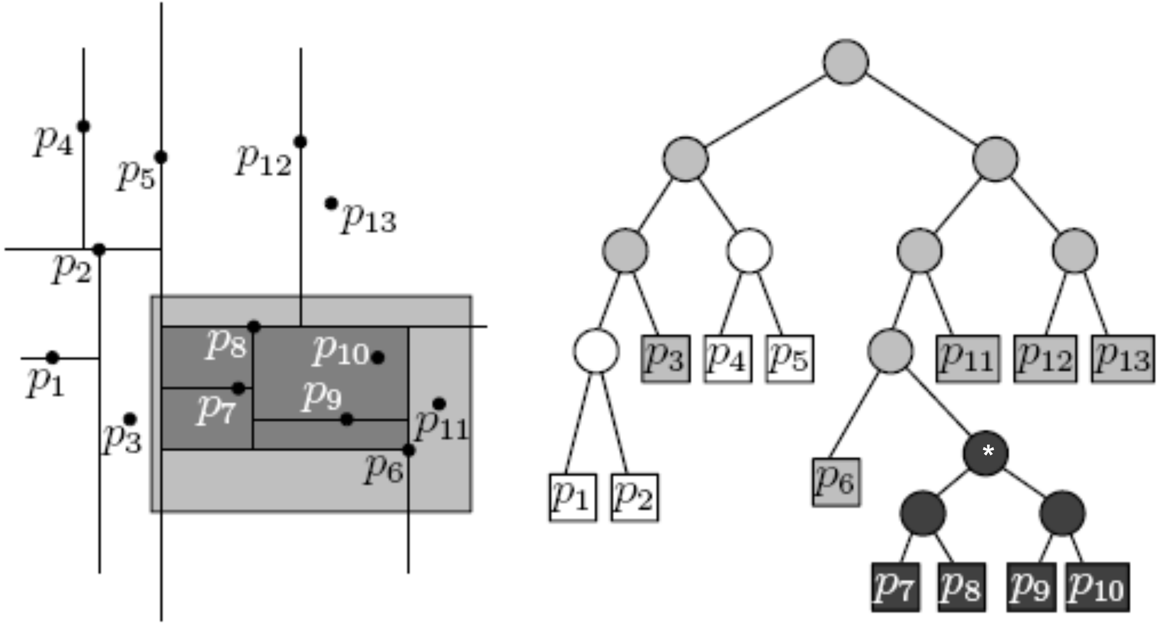
Example - Range Search



Query region: gray rectangle

Gray nodes are the nodes visited in this example.

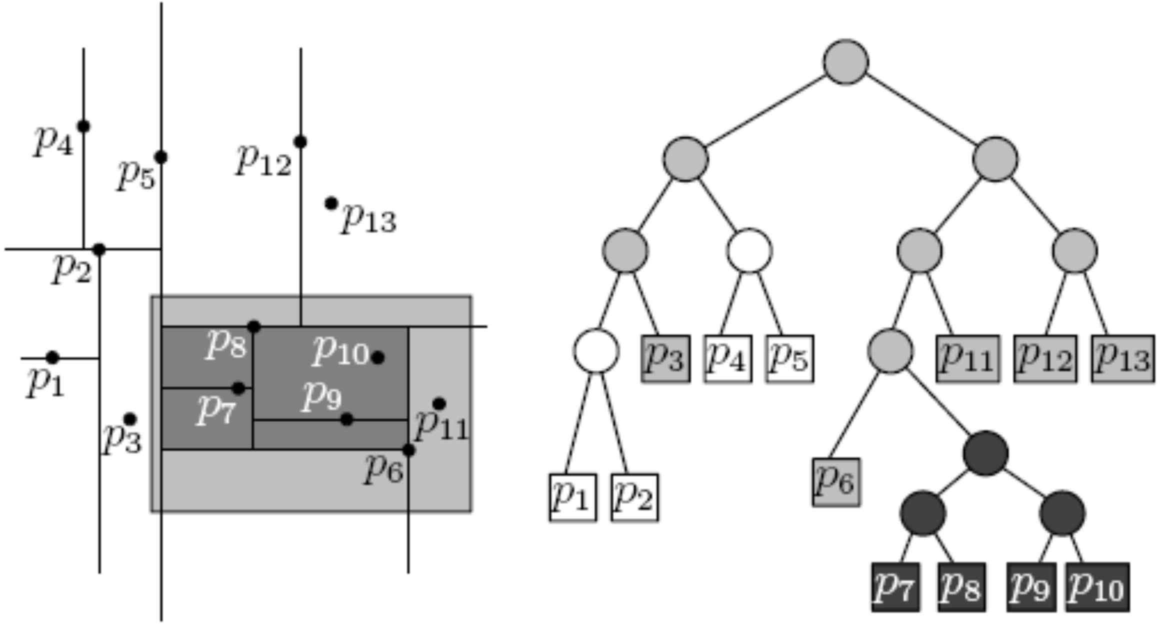
Example - Range Search



Node marked with * corresponds to a region that is entirely inside the query rectangle

Report all leaves in this subtree.

Example - Range Search



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}

KD Tree - Complexity

- Construction $O(dn\log n)$
  - Sort points in each dimension: $O(dn\log n)$
  - Determine splitting line (median finding): $O(dn)$
- Space requirements:
  - $O(n)$
- Query requirements:
  - KD tree: $O(n^{1-\frac{1}{d}} + k)$ → $O(n+k)$ as d increases!

Related References

- Slides are modified based on the slides created by Dr George Bebis from Department of Computer Science & Engineering University of Nevada (UNR) Reno
- https://en.wikipedia.org/wiki/K-d_tree