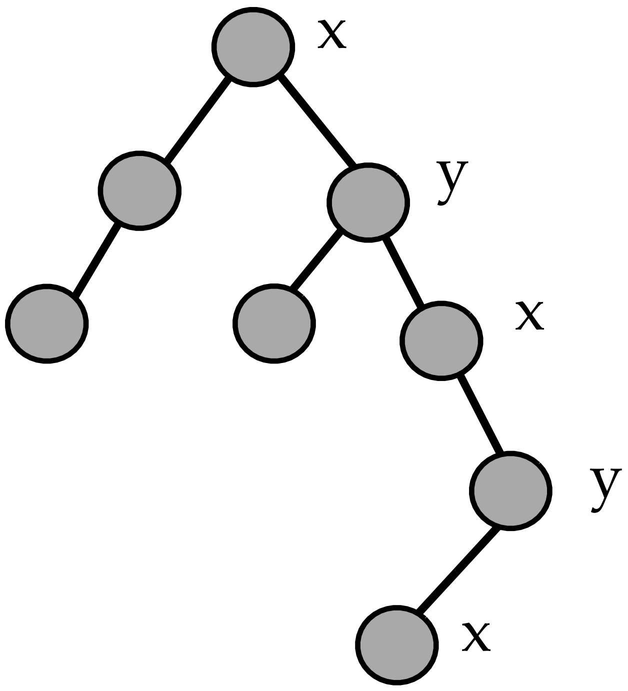
KD Trees

CS223: Data Structures

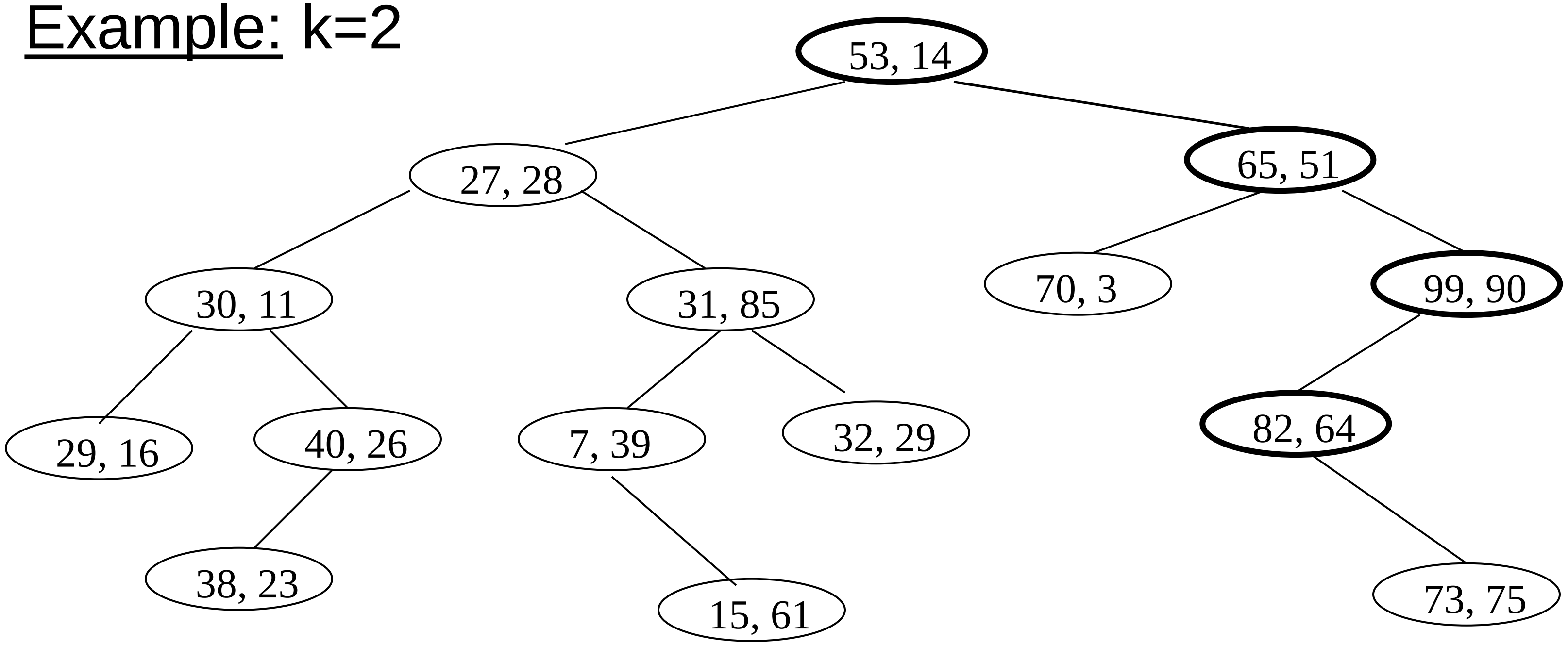
KD Tree

- Name originally meant "3d-trees, 4d-trees, etc" where k was the # of dimensions
- A binary search tree where every node is a k-dimensional point
- It is a space partitioning data structure for organizing points in a K-Dimensional space
- Each level has a "cutting dimension"
- Each node contains a point P = (x,y)
- To find (x',y') you only compare coordinate from the cutting dimension
  - e.g. if cutting dimension is x, then you ask: is x' < x?



KD Tree

Example: k=2



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.
- 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.

Splitting Strategies

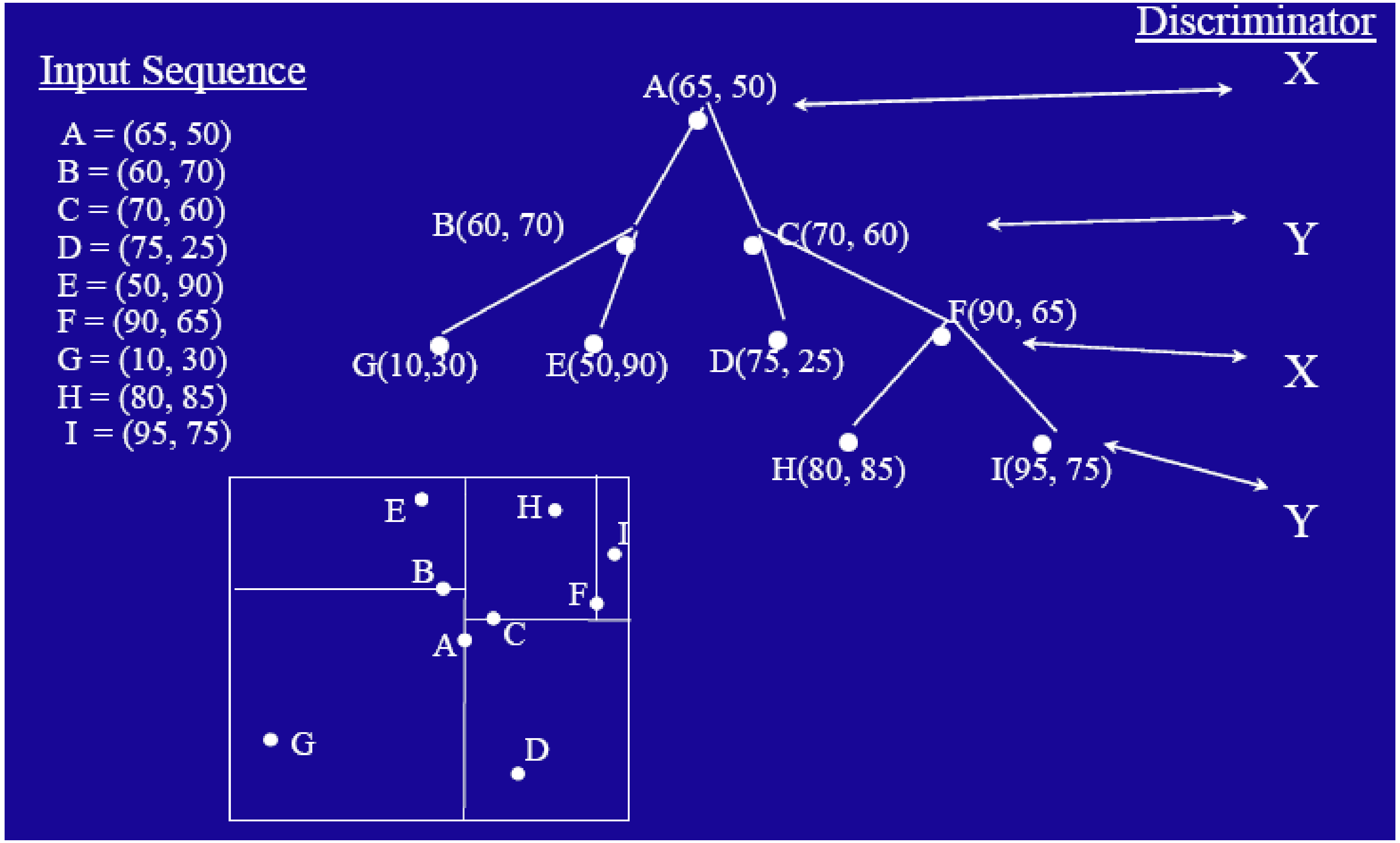
- Divide based on order of point insertion
  - Assumes that points are given one at a time.

- 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

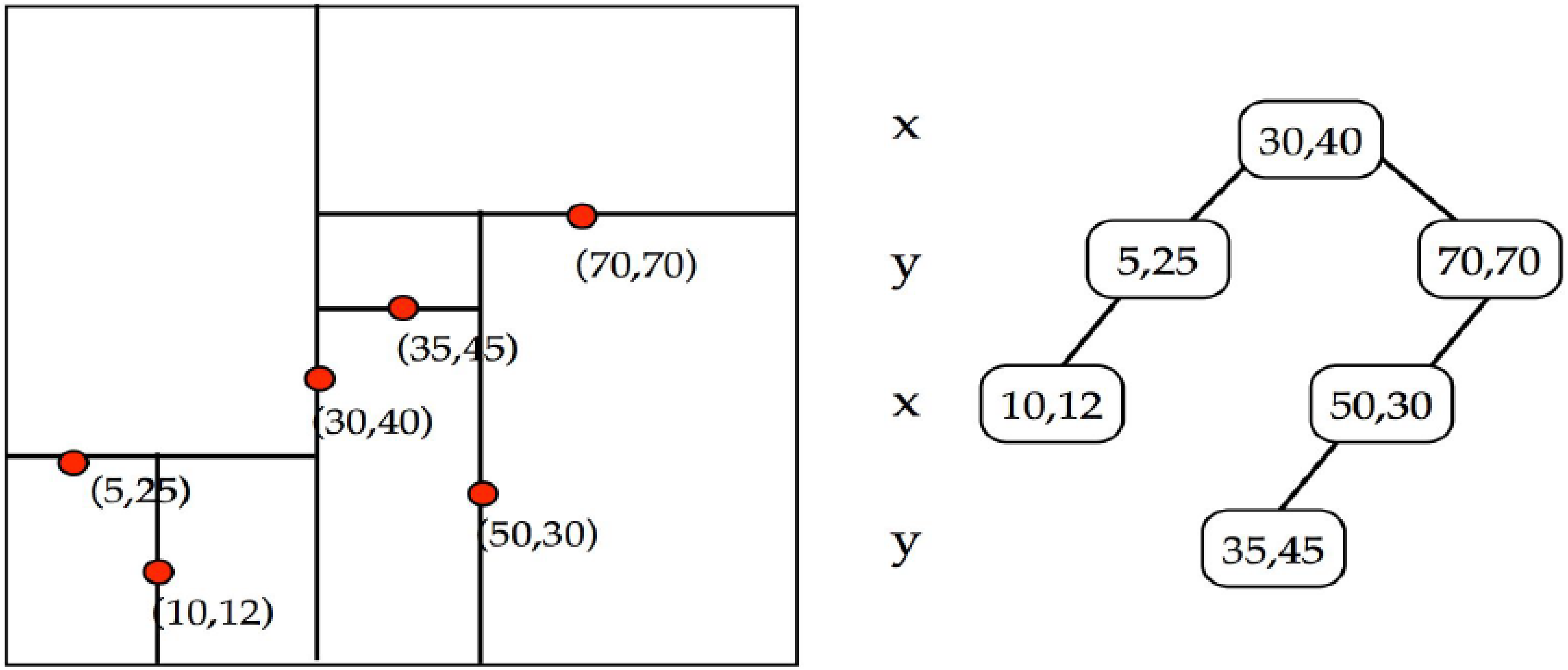
Example – using order of point insertion

(data stored at nodes



Example – using order of point insertion
(data stored at nodes

insert: (30,40), (5,25), (10,12), (70,70), (50,30), (35,45)



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, ...

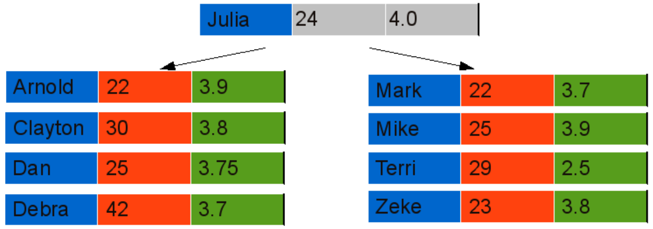
Another Example - using median

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

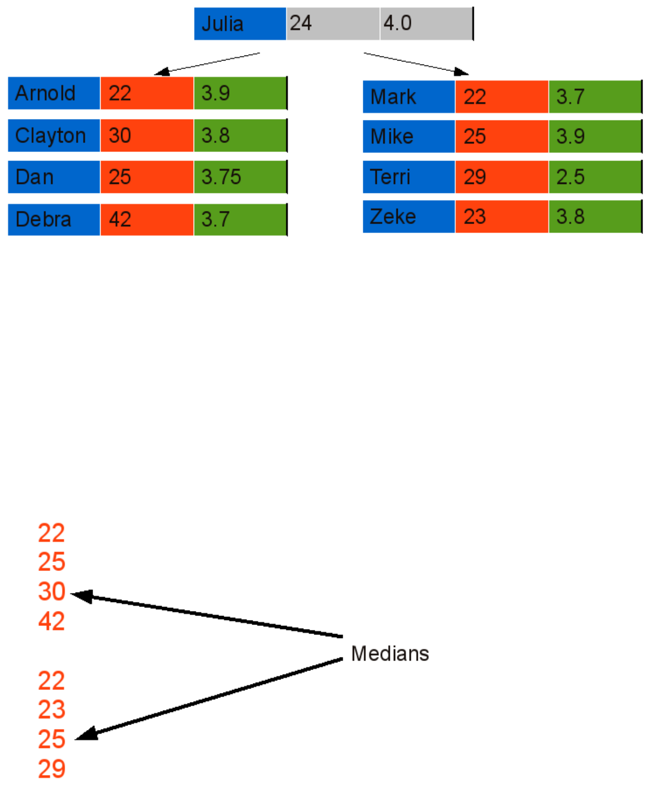
Another Example - using median



Arnold
Clayton
Dan
Debra
Julia ← Median
Mark
Mike
Terri
Zeke

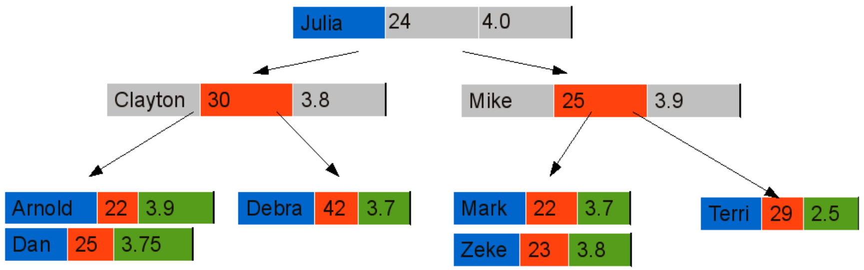
Discriminator order: Name, age, GPA, name, age, GPA, ...

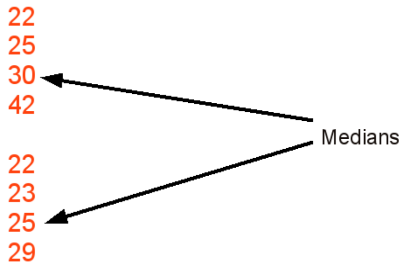
Another Example - using median



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

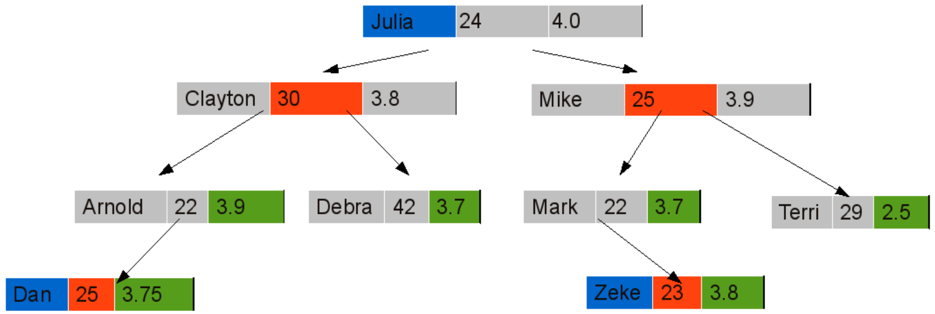
Another Example - using median





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

Another Example - using median



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

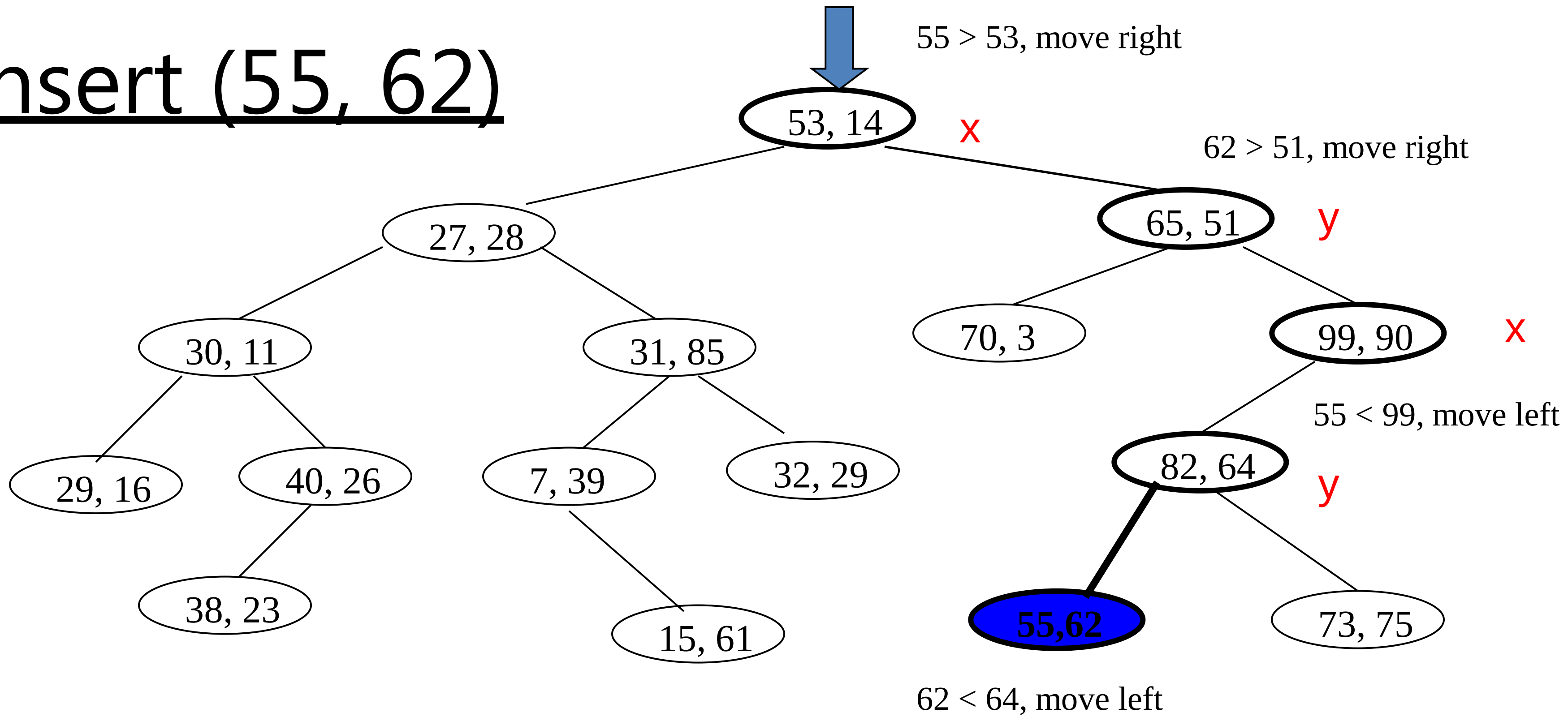
KD Tree

```
insert(Point x, KDNode t, int cd) {
        if t == null t = new KDNode(x)
        else    if (x == t.data)
                // error! duplicate
        else    if (x[cd] < t.data[cd])
                t.left = insert(x, t.left, (cd+1) % DIM)
        else
                t.right = insert(x, t.right, (cd+1) % DIM)
        return t
}
```

```
KDNode
{
    // To store k dimensional point
    int point[k];
    Node *left, *right;
};
```

Insert new data

Insert (55, 62)



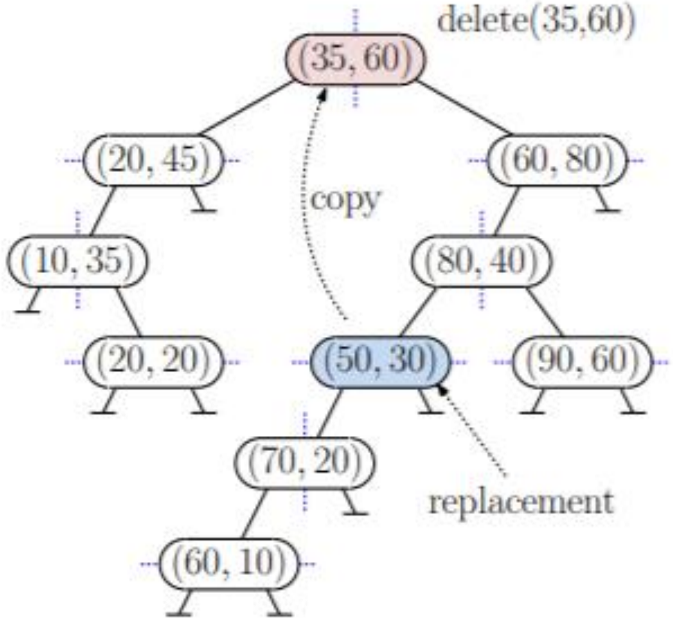
Delete data

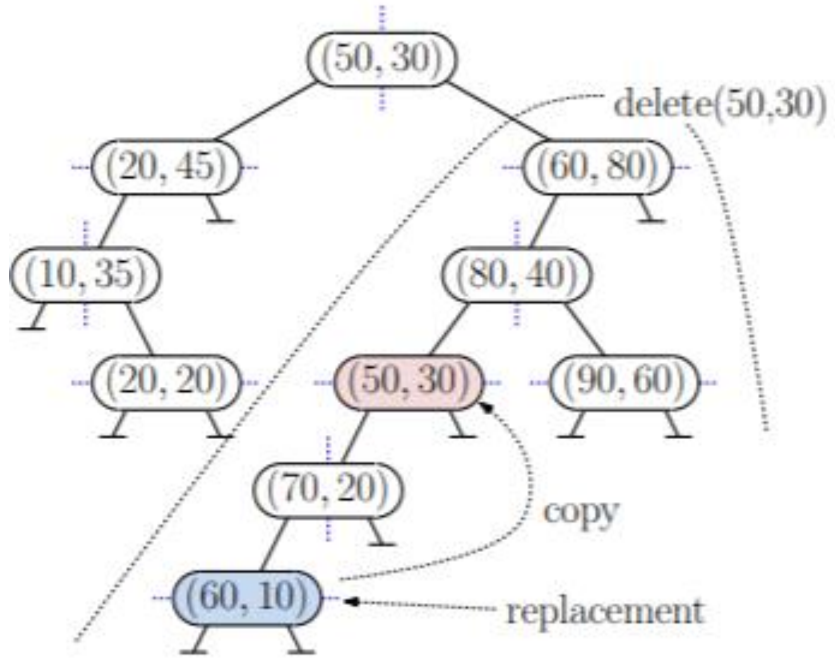
- If current does contain the point to be deleted
  - If node to be deleted is a leaf node, delete it
  - If node to be deleted has right child as not NULL
    - Find minimum of current node’s dimension in right subtree.
    - Replace the node with above found minimum and recursively delete minimum in right subtree
  - Else If node to be deleted has left child as not NULL
    - Find minimum of current node’s dimension in left subtree.
    - Replace the node with above found minimum and recursively delete minimum in left subtree.
    - Make new left subtree as right child of current node.

If current doesn’t contain the point to be deleted

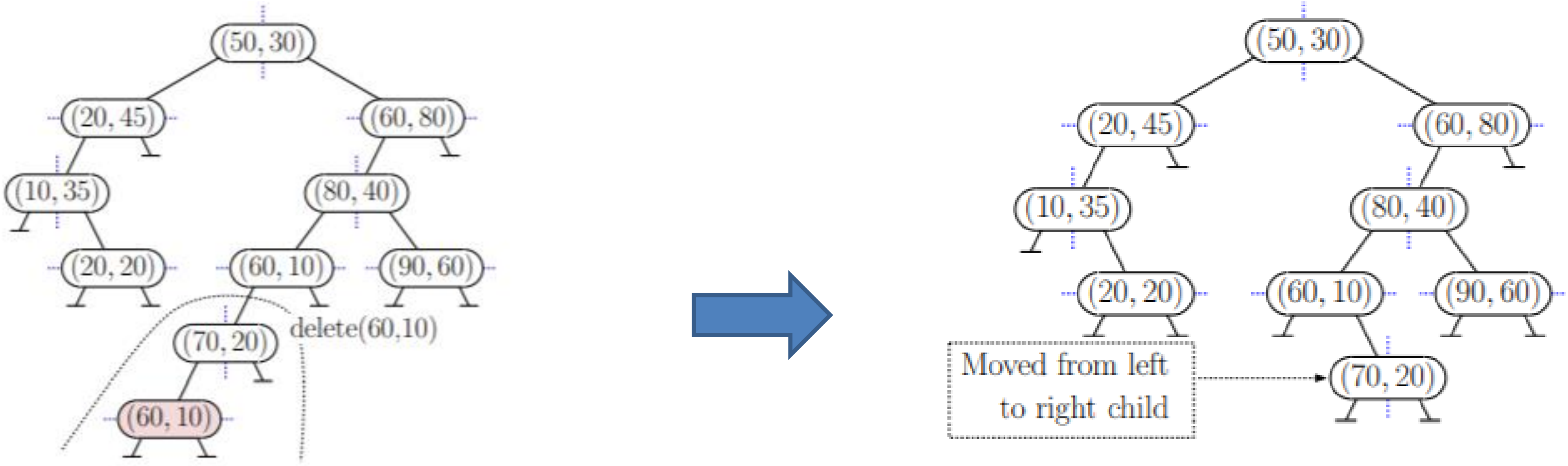
- If node to be deleted is smaller than current node on current dimension, recur for left subtree, else recur for right subtree

Delete data

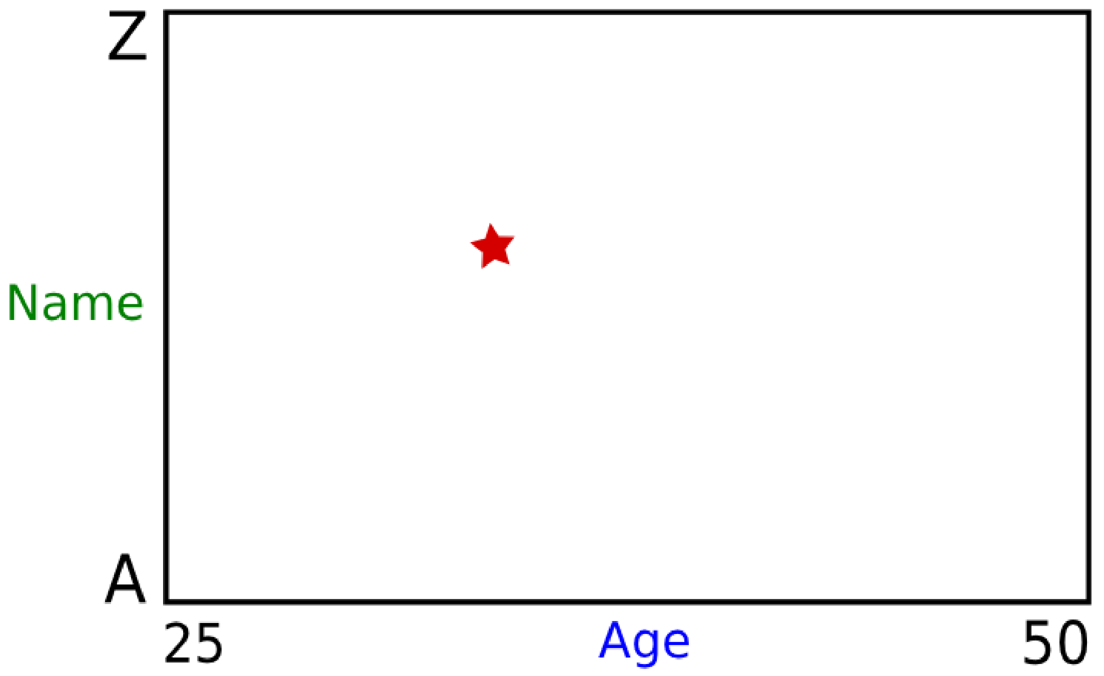

delete(35,60)
(35, 60)
(20, 45)
(60, 80)
copy
(10, 35)
(80, 40)
(20, 20)
(50, 30)
(90, 60)
(70, 20)
replacement
(60, 10)


(50, 30)
delete(50,30)
(20, 45)
(60, 80)
(10, 35)
(80, 40)
(20, 20)
(50, 30)
(90, 60)
(70, 20)
copy
(60, 10)
replacement

Delete data (cont'd)

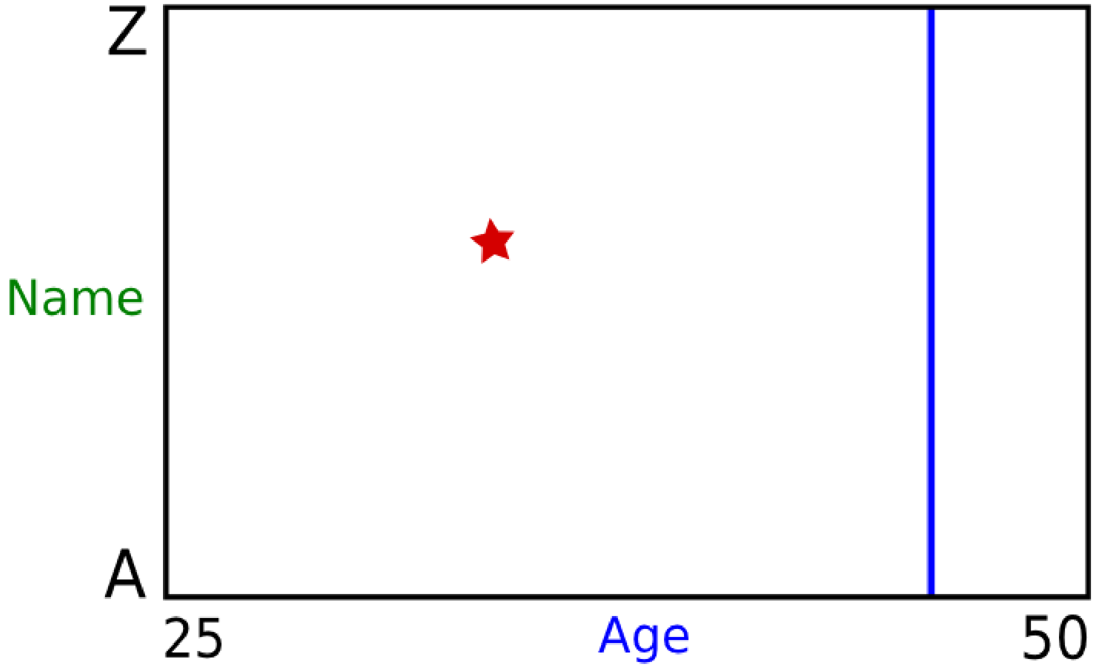


KD Tree – Exact Search



Search key: (32, P)

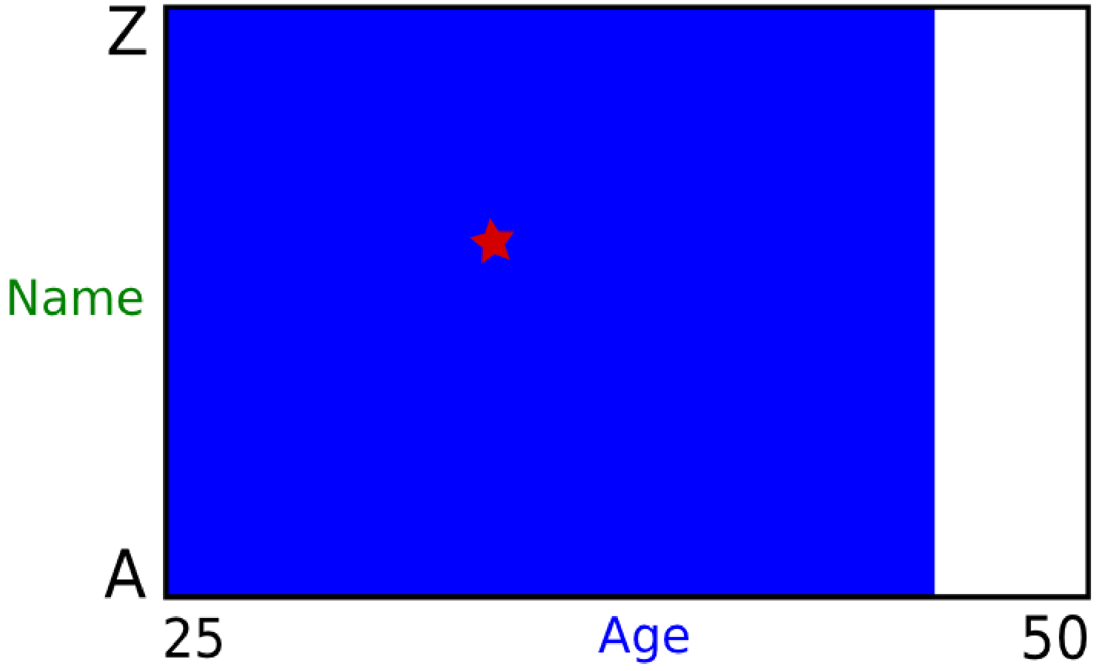
KD Tree – Exact Search



Current node's key: Age=45

Search key: (32, P)

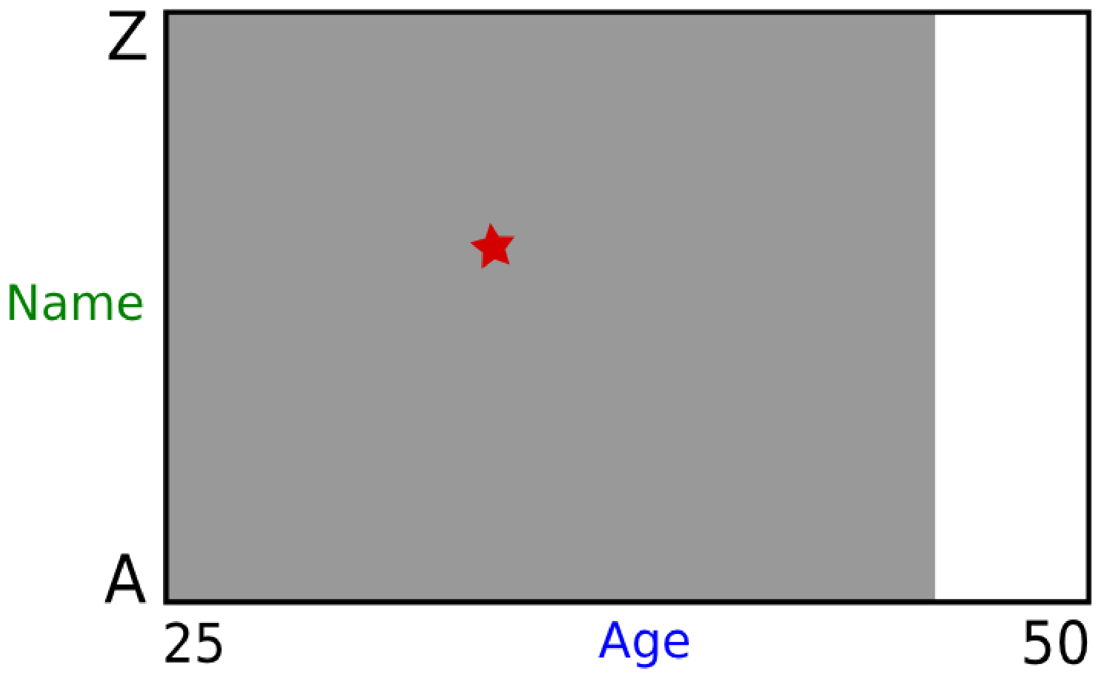
KD Tree – Exact Search



Current node's key: Age=45

Search key: (32, P)

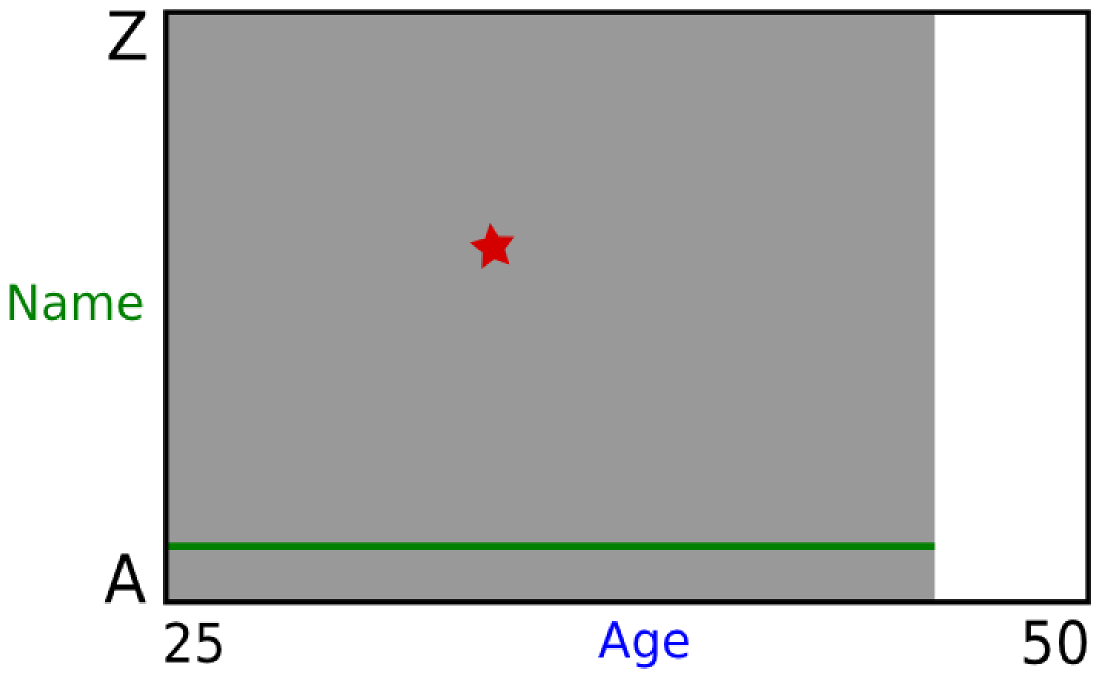
KD Tree – Exact Search



Current node's key: Age=45

Search key: (32, P)

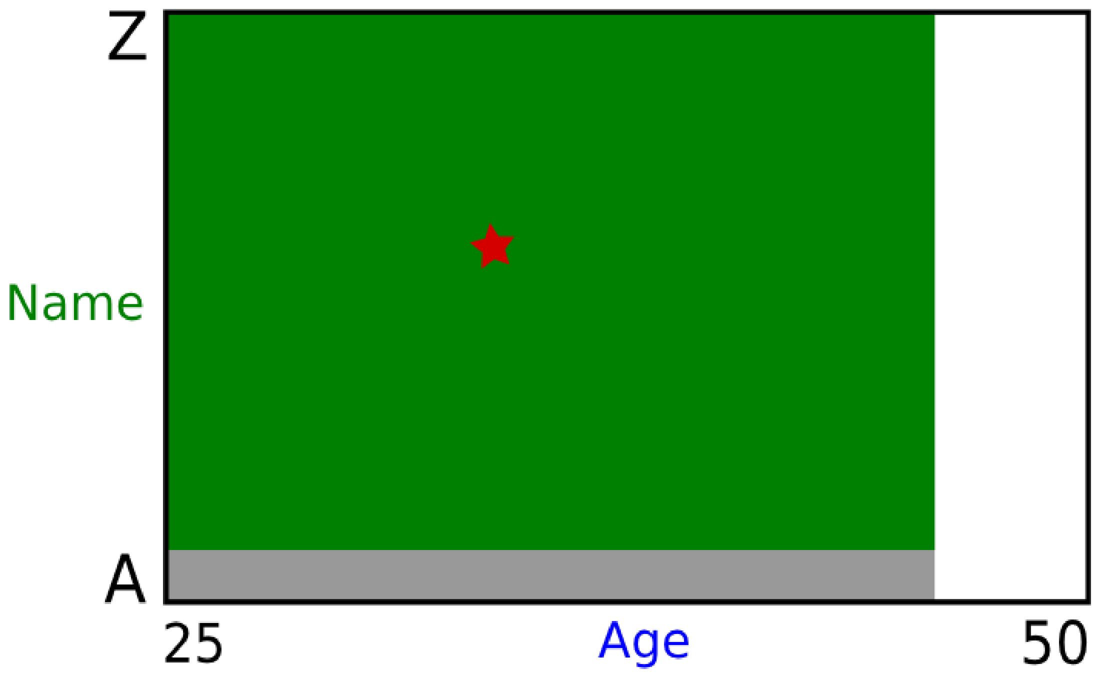
KD Tree – Exact Search



Current node's key: Name=B

Search key: (32, P)

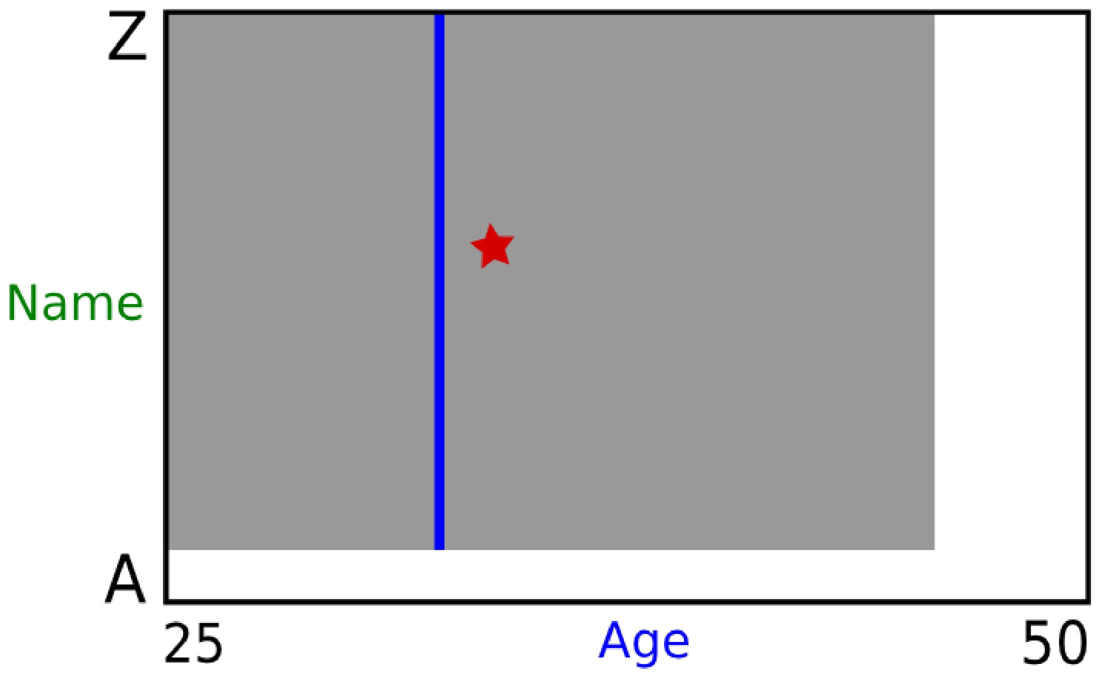
KD Tree – Exact Search



Current node's key: Name=B

Search key: (32, P)

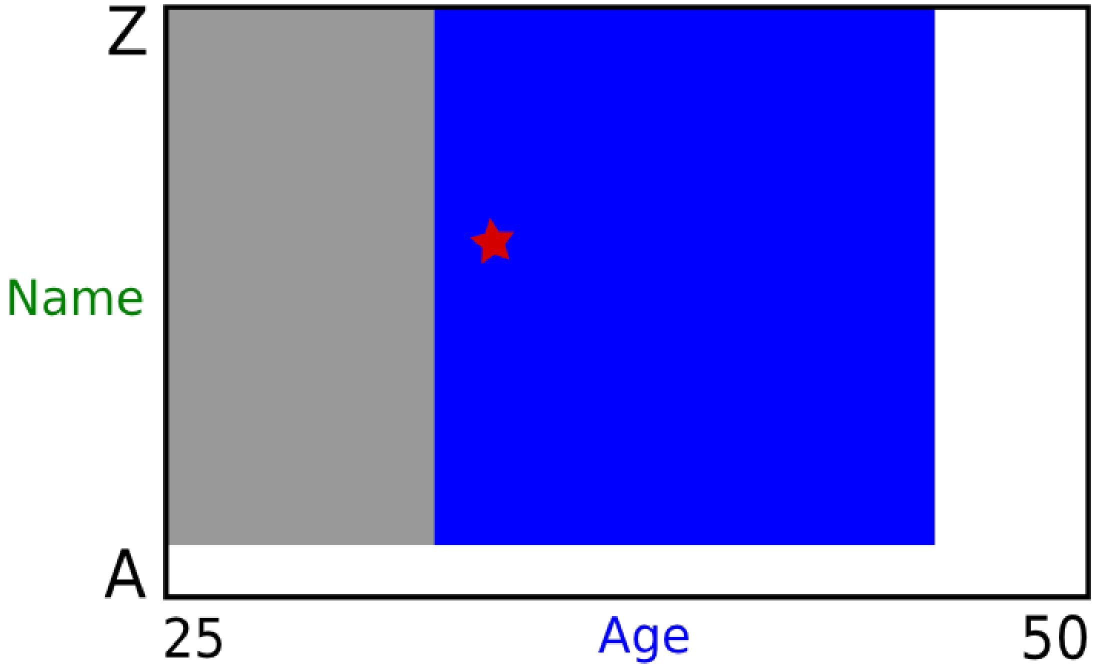
KD Tree – Exact Search



Current node's key: Age=30

Search key: (32, P)

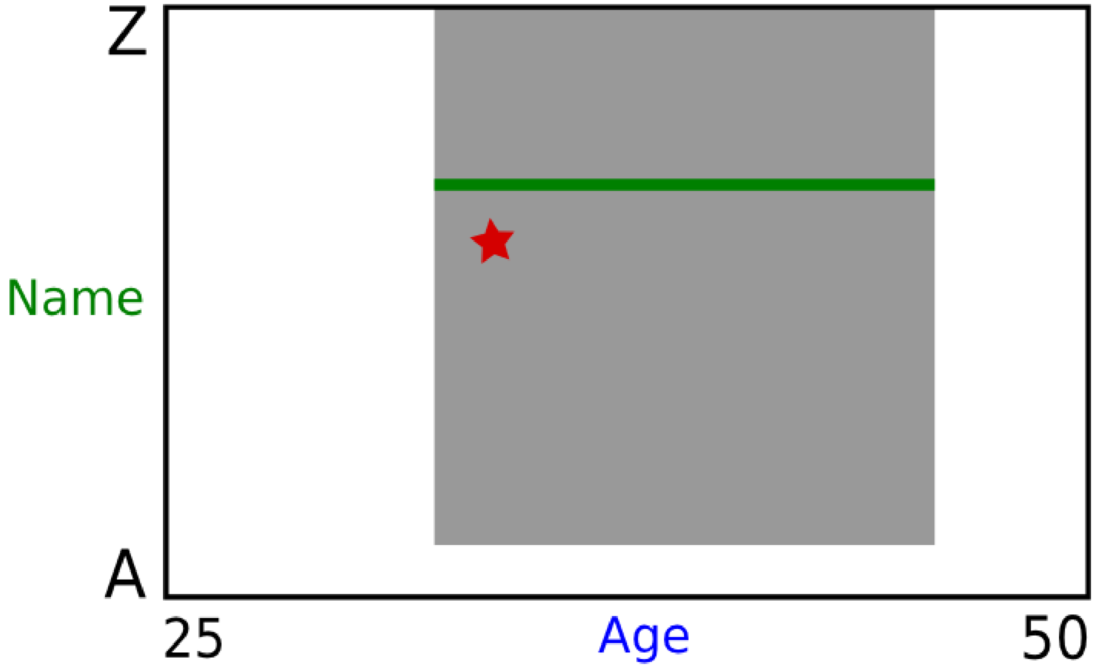
KD Tree – Exact Search



Current node's key: Age=30

Search key: (32, P)

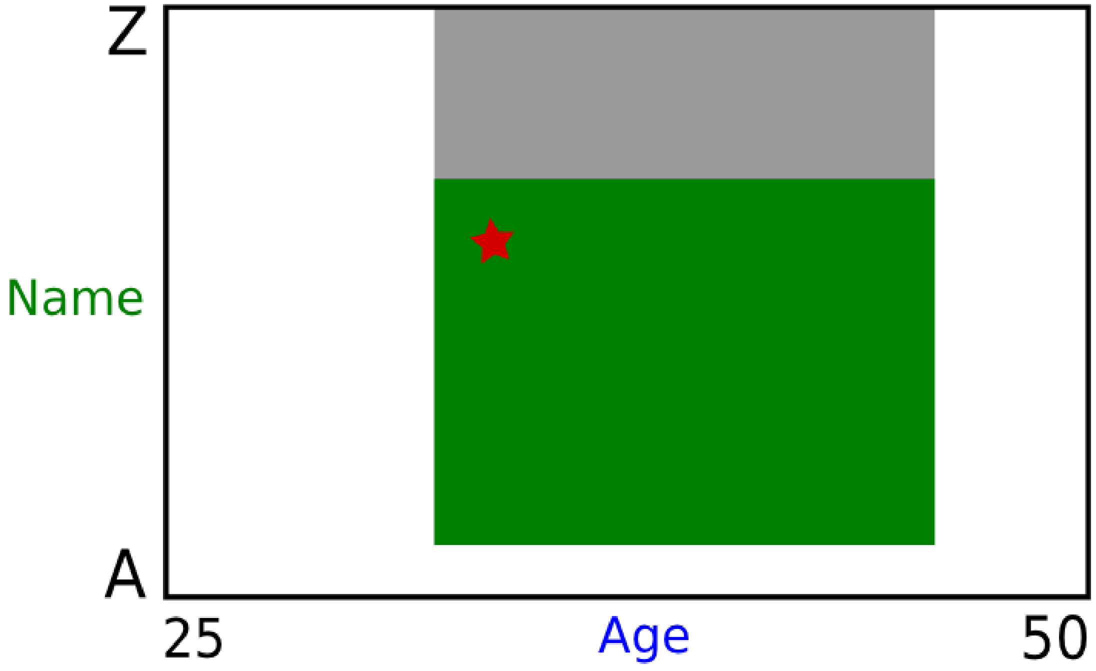
KD Tree – Exact Search



Current node's key: Name=R

Search key: (32, P)

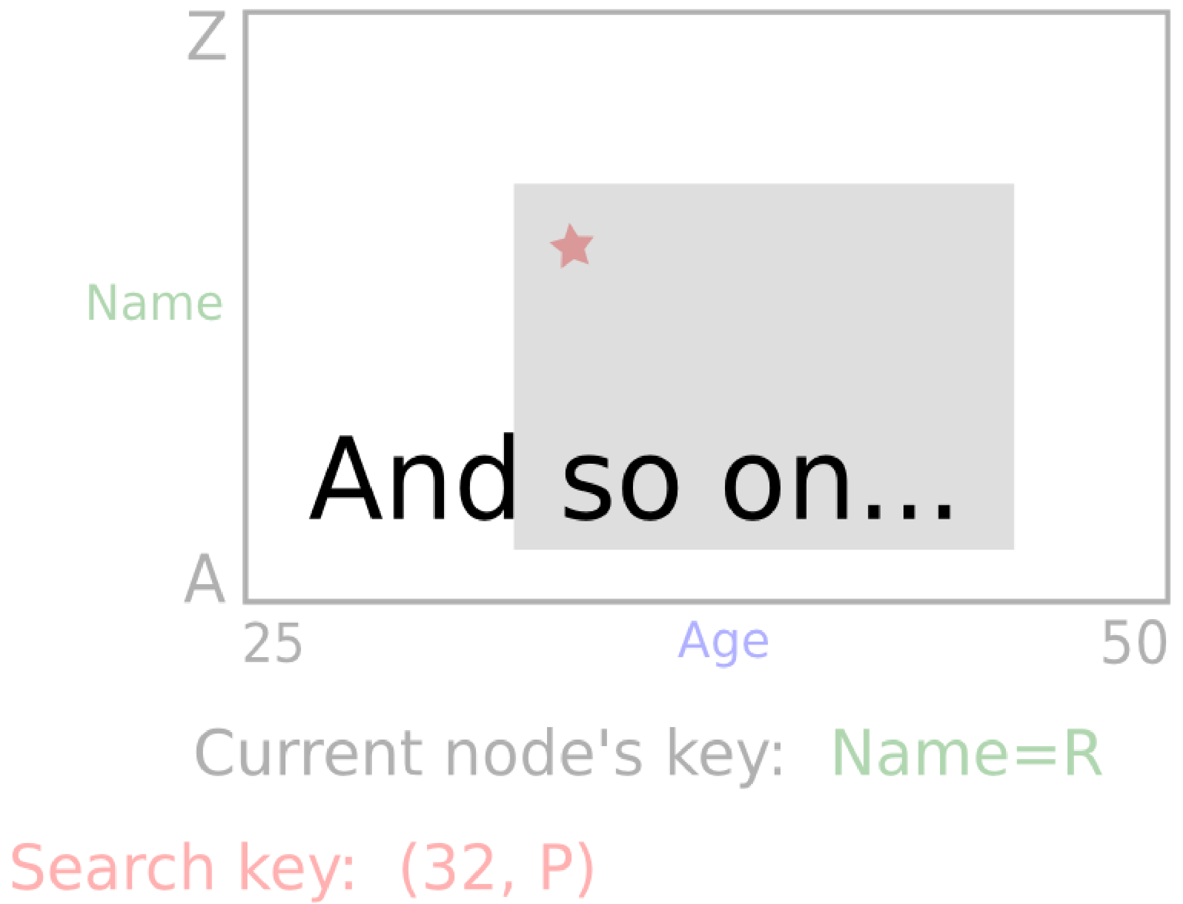
KD Tree – Exact Search



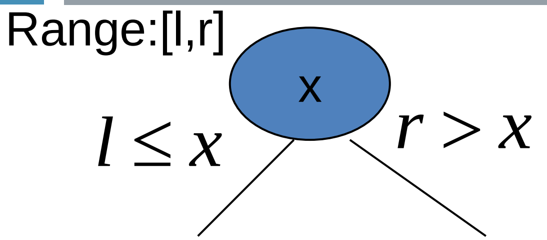
Current node's key: Name=R

Search key: (32, P)

KD Tree – Exact Search



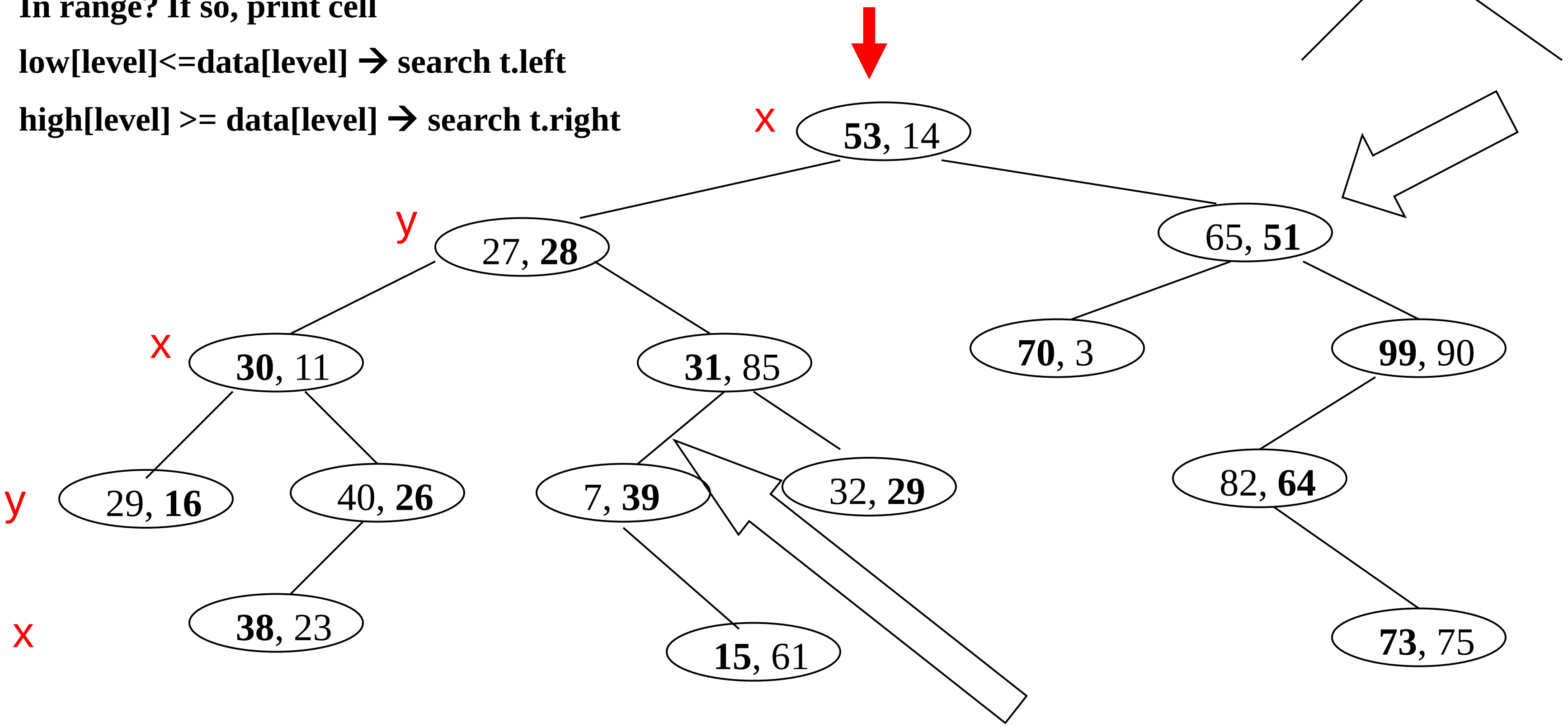
KD Tree - Range Search



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.

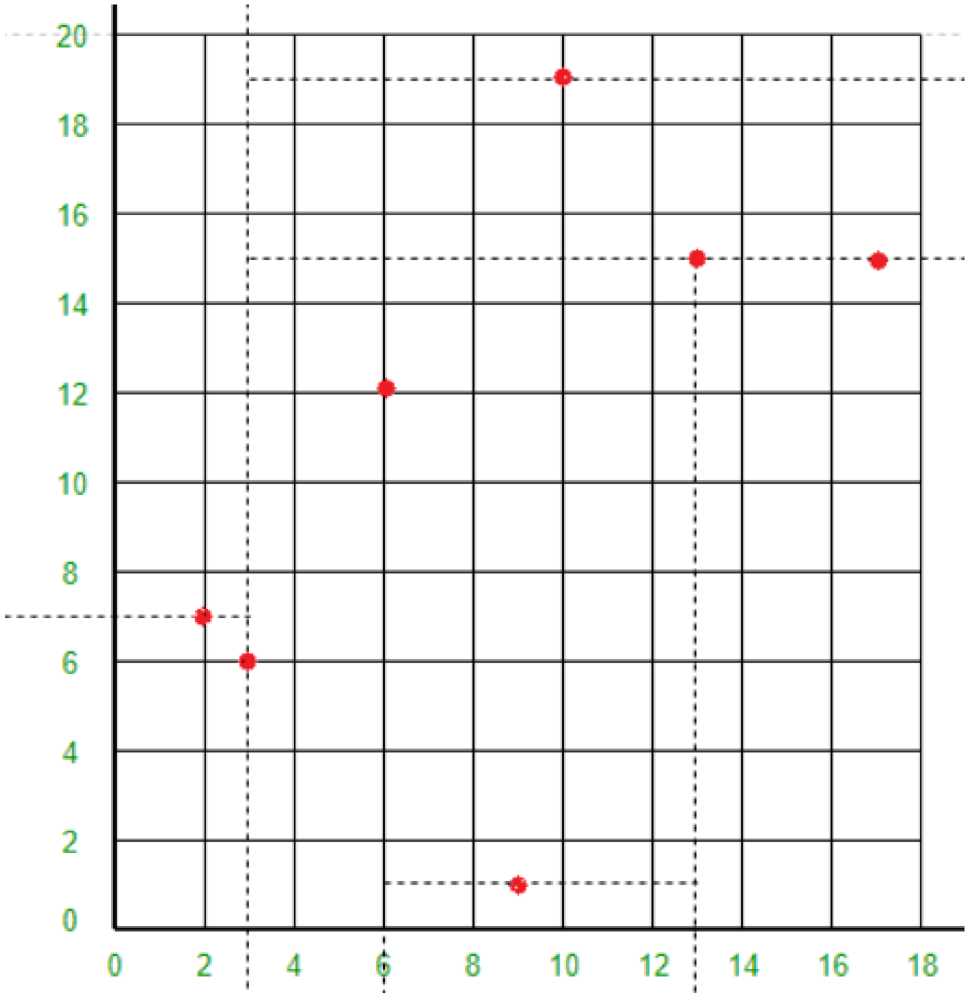
Searching is "preorder". Efficiency is obtained by "pruning" subtrees from the search.

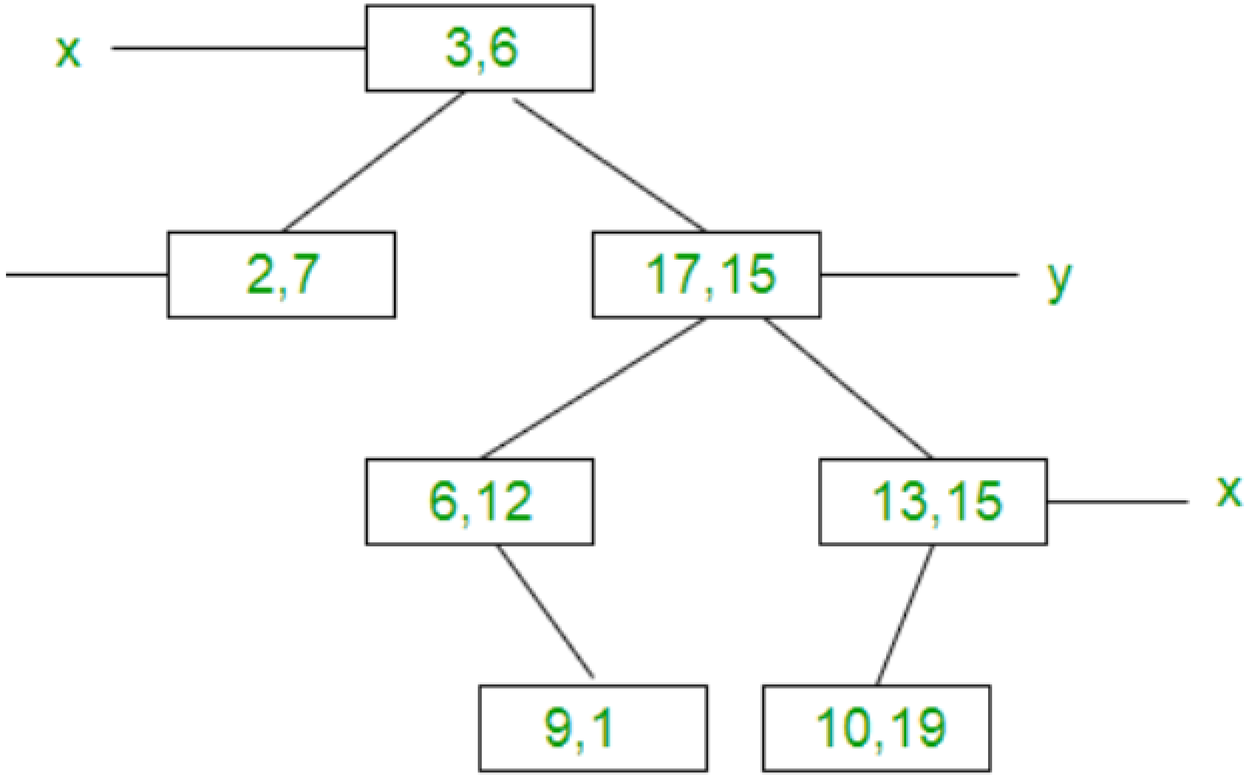
Additional Examples

Example – using order of point insertion

(data stored at nodes

(3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)





Example – using Median

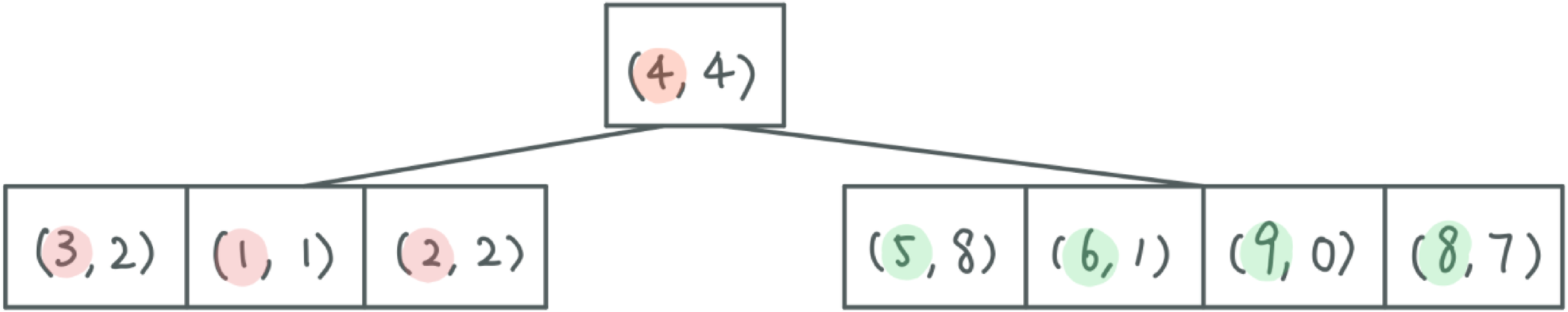
(3, 2)	(5, 8)	(6, 1)	(9, 0)	(4, 4)	(1, 1)	(2, 2)	(8, 7)

Suppose we are given this array to construct a kd-Tree.

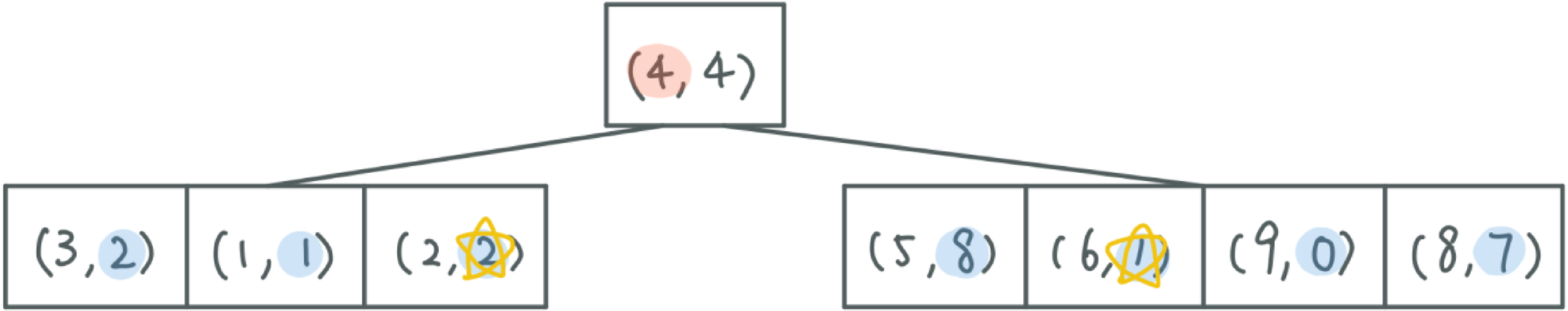
(3, 2)	(5, 8)	(6, 1)	(9, 0)	(4, 4)	(1, 1)	(2, 2)	(8, 7)

(4, 4) is the median in terms of *x* coordinate, make it our subroot.

Example – using Median

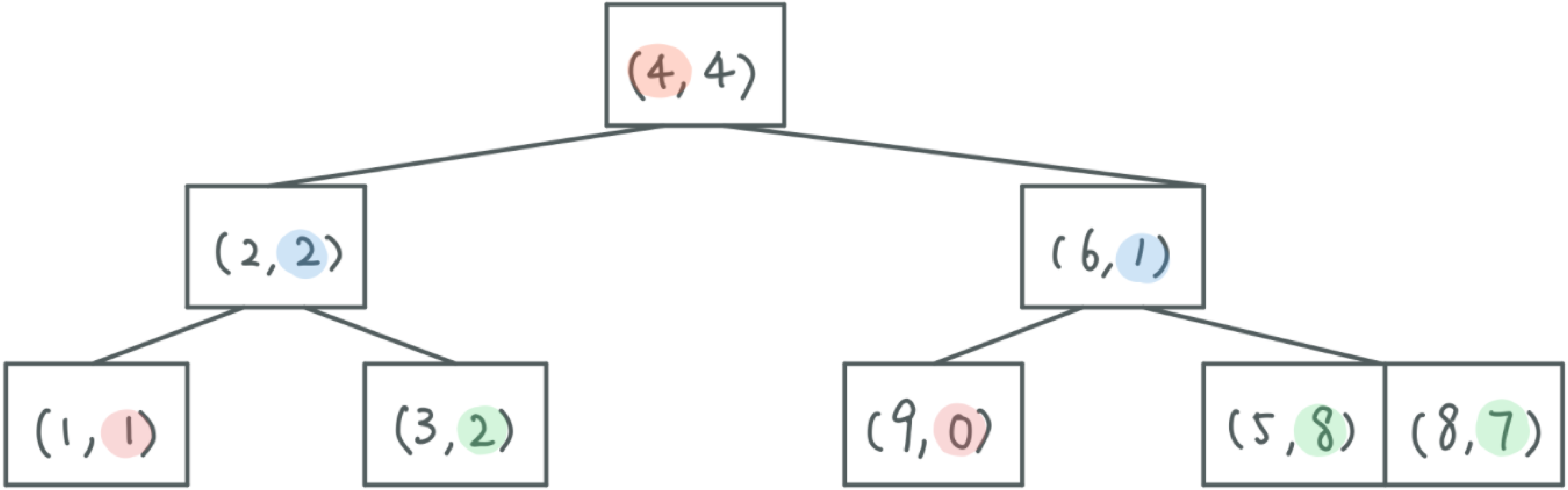


Partition the array so that every point with *x* coordinate smaller than 4 is on the left side of (4, 4), and every point with *x* coordinate larger than 4 is on the right side of (4, 4).



Find the median in terms of *y* coordinate on each of the subarrays.

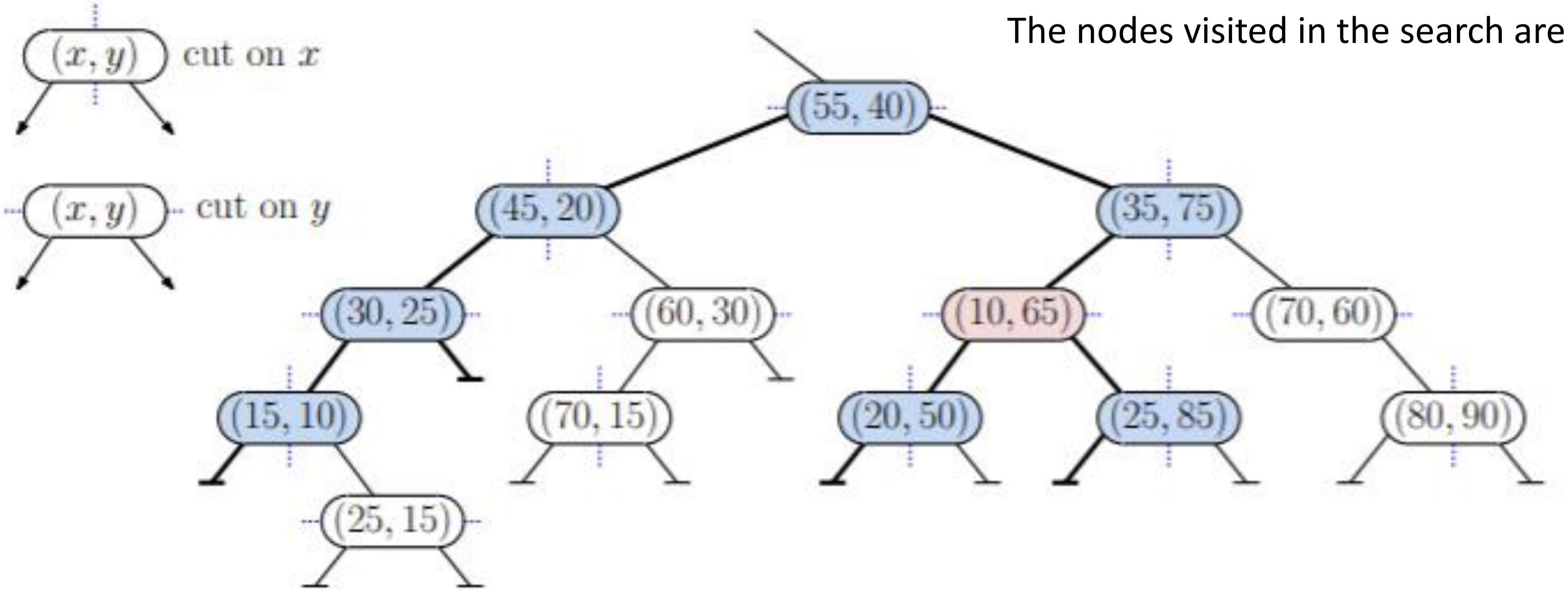
Example – using Median



Partition each subarray by its median, and make the median the subroot. Repeat this process until the array only consists of one node.

Example – Find Minimum

The nodes visited in the search are shaded in blue

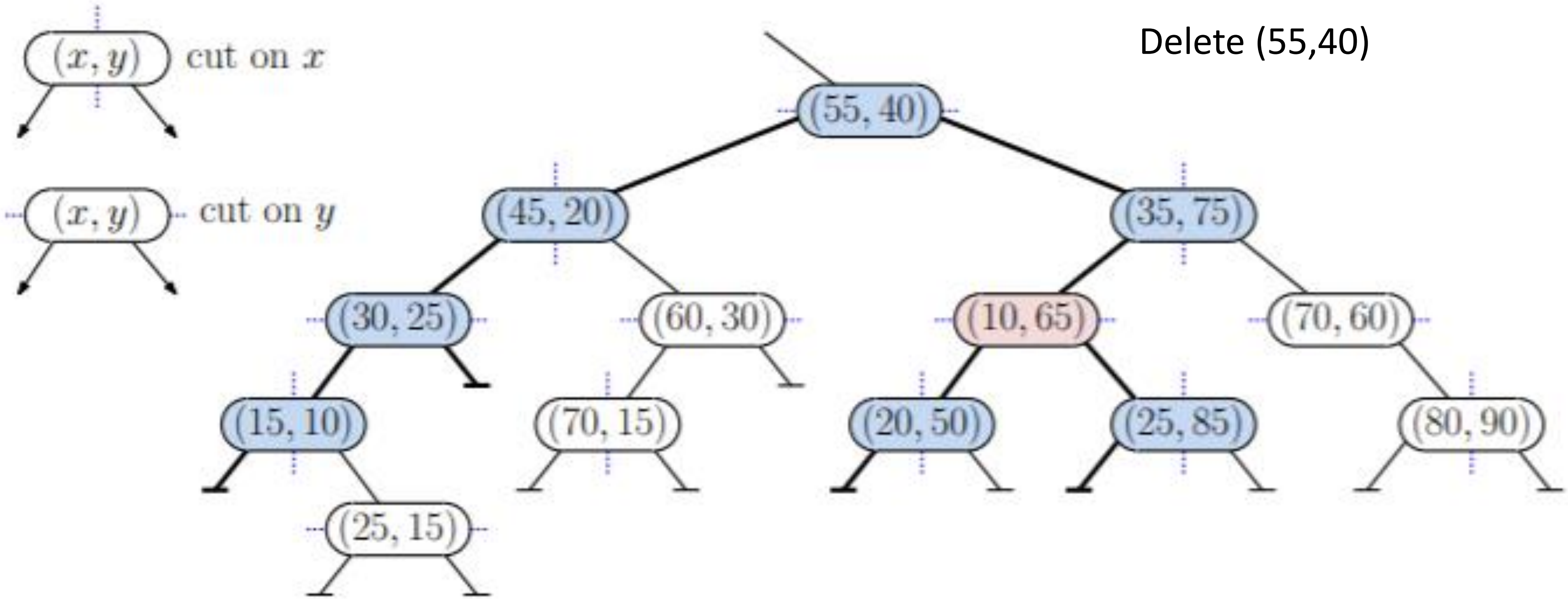


findMin for x-coordinate on the subtree rooted at (55, 40). The function returns (10, 65)

Since this node splits horizontally, we need to visit both of its subtrees

because the subtrees at (45, 20) and (35, 75) both split on the x-coordinate, and we are looking for the point with the minimum x-coordinate, we do not need to search their right subtrees

Example – Delete



Delete (55,40)

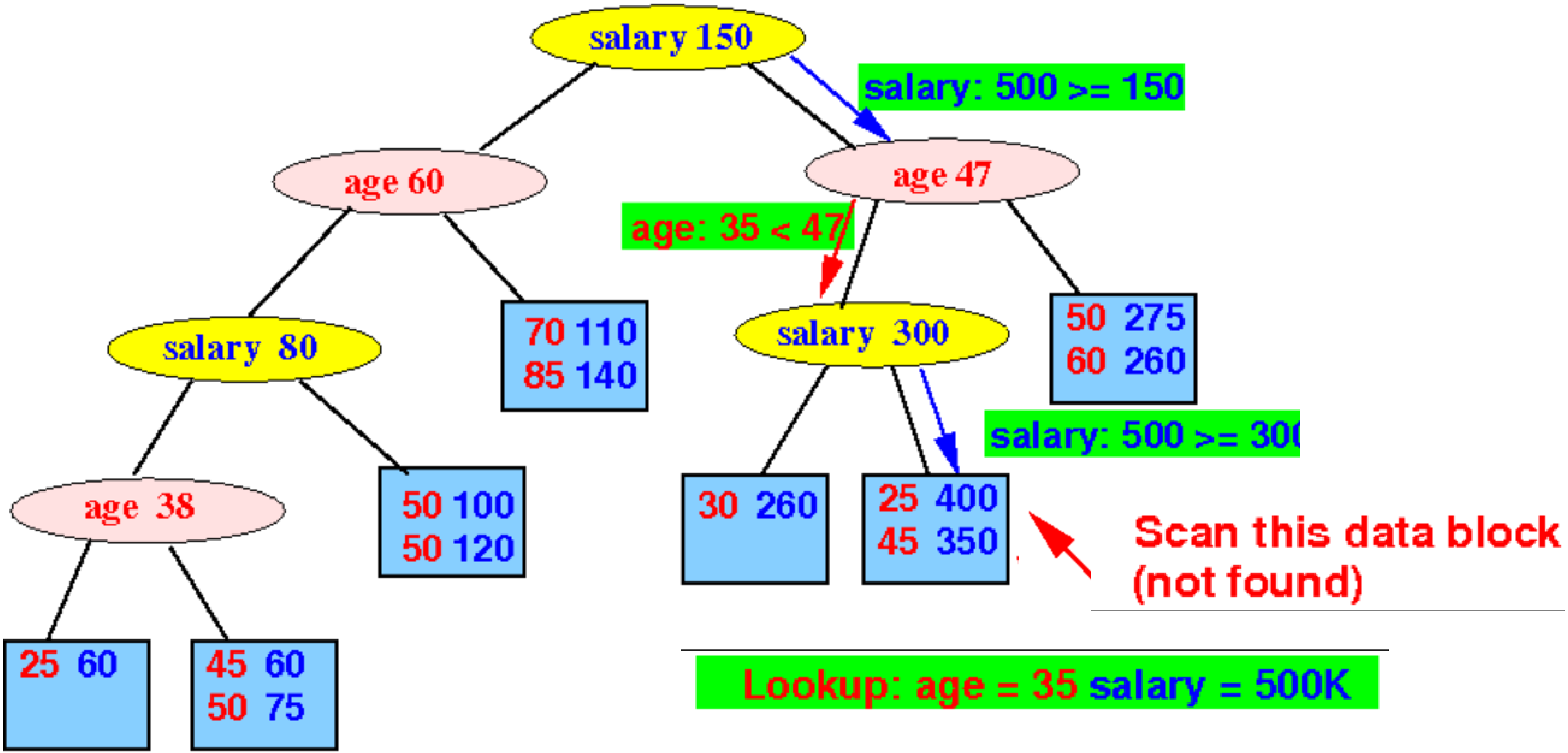
Replace it with minimum in y dimension in right subtree

Because the subtree at (35, 75) split on the x-coordinate, and we are looking for the point with the minimum y-coordinate, we need to search both subtrees

Since both subtrees (10,65) and (70,60) splits horizontally, we need to visit only their left subtrees

Replacement node will be (20,50) which is leaf node so when we recursively delete it, it will be simple case.

Example –Exact Search



Example - Range Search