CS2040S
Data Structures and Algorithms
(e-learning edition)

Augmented Trees!
Part 3

Today

Three examples of augmenting BSTs

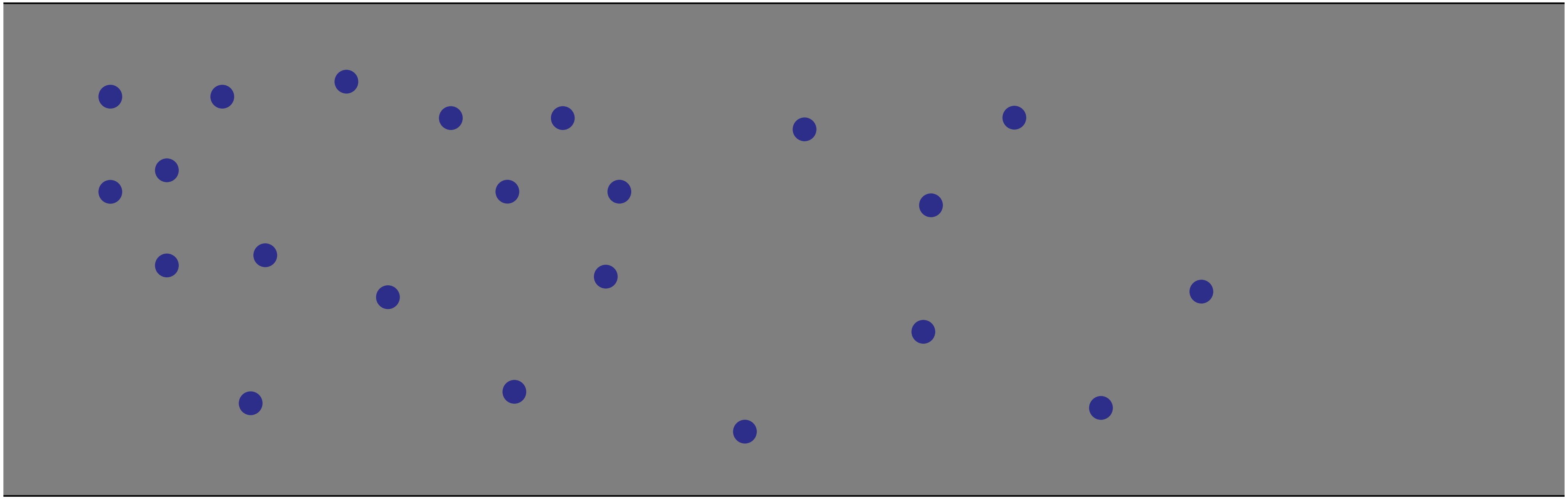
1. Order Statistics

2. Intervals

3. Orthogonal Range Searching

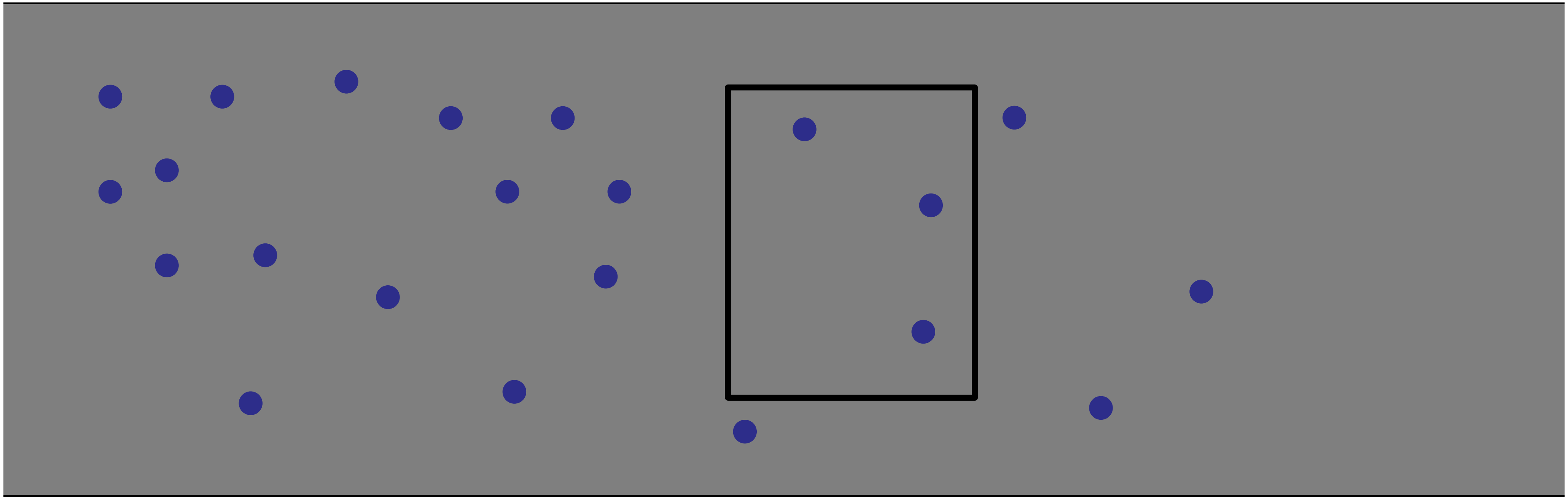
Orthogonal Range Searching

Input: n points in a 2d plane



Orthogonal Range Searching

Input: n points in a 2d plane

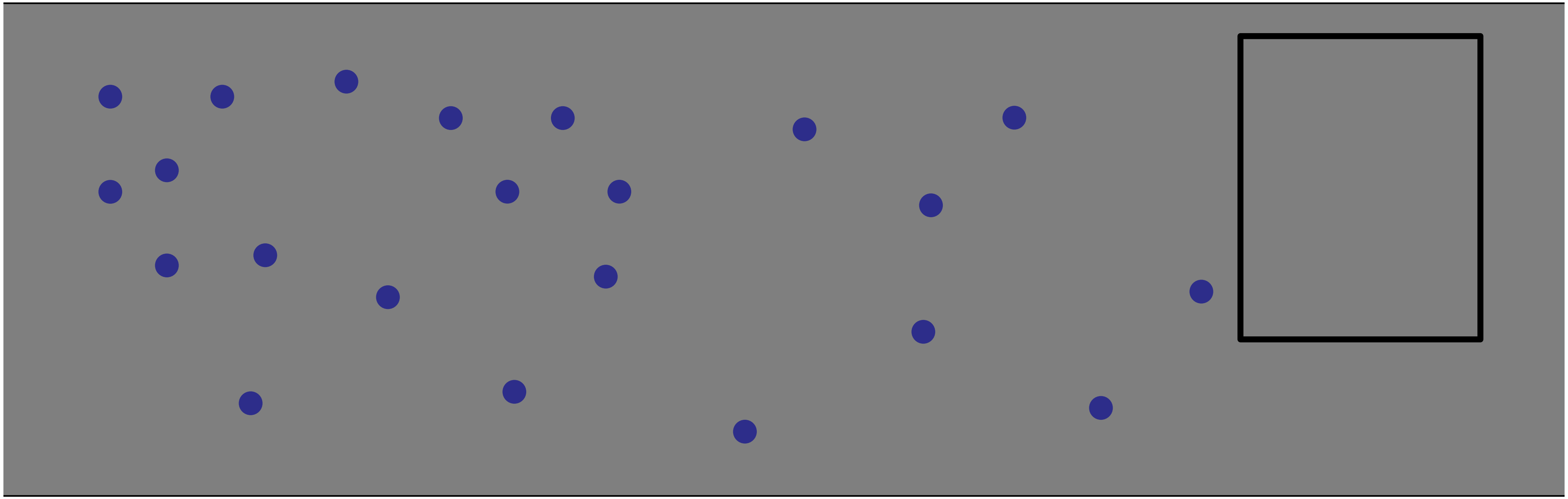


Query: Box

- Contains at least one point?
- How many?

Orthogonal Range Searching

Input: n points in a 2d plane

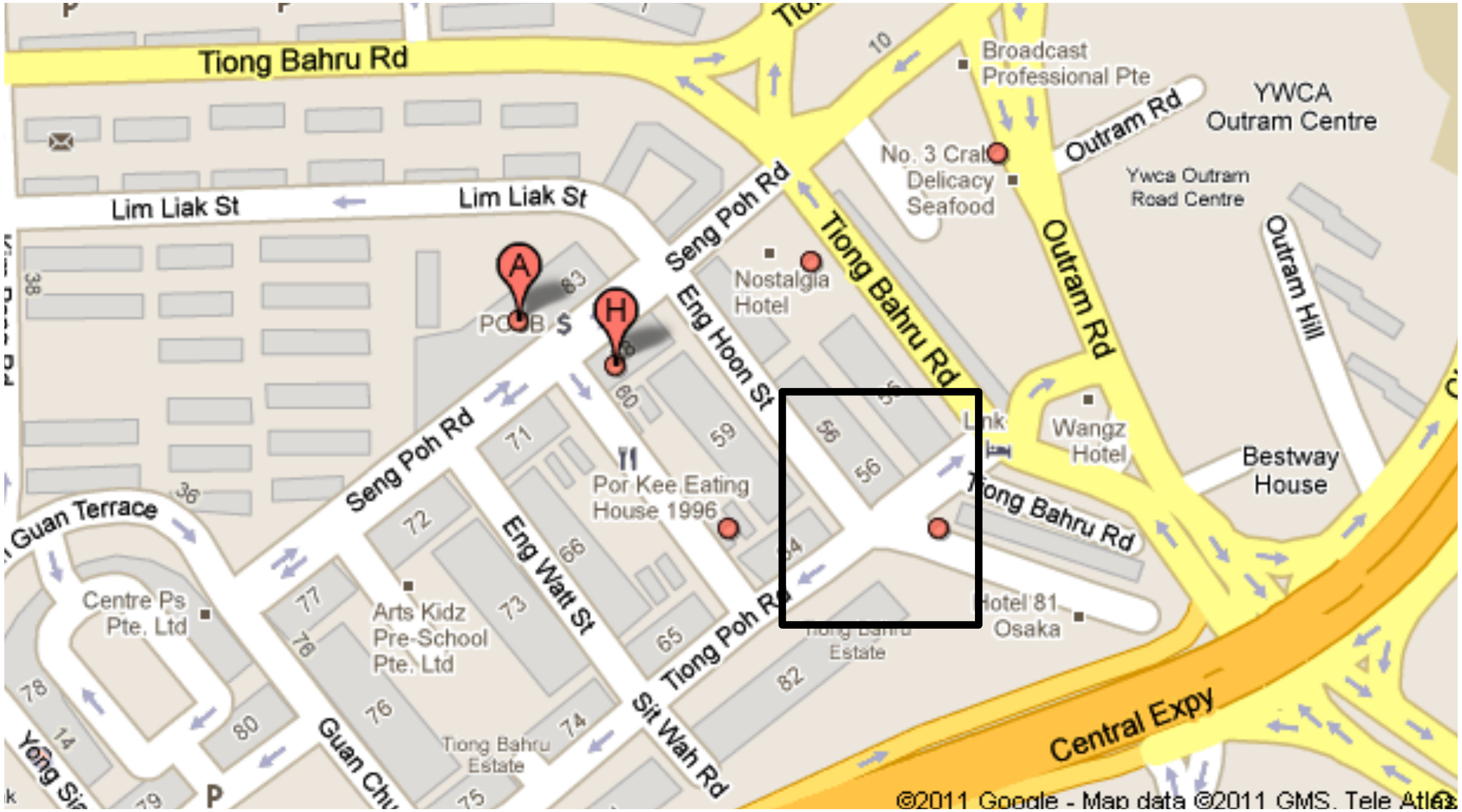


Query: Box

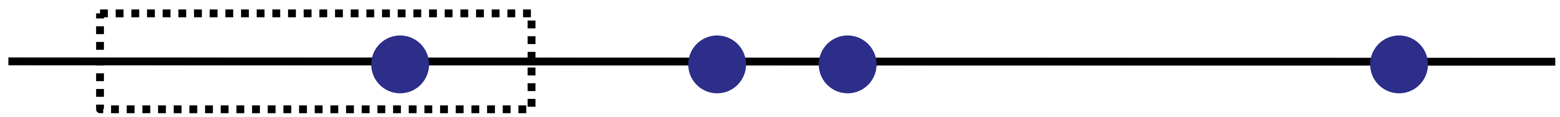
- Contains at least one point?
- How many?

Practical Example

Are there any good restaurants within one block of me?



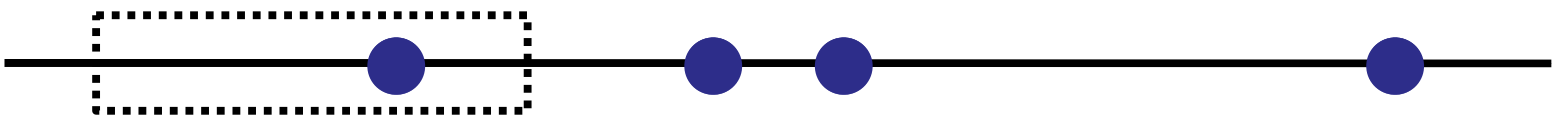
One Dimension



One Dimension

Range Queries

- Important in databases
- "Find me everyone between ages 22 and 27."



One Dimension

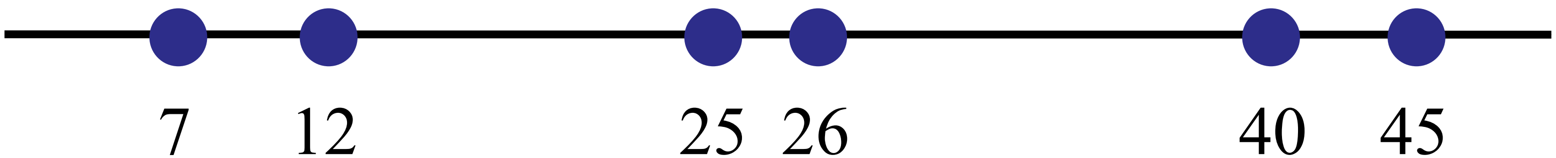
Strategy:

1. Use a binary search tree.

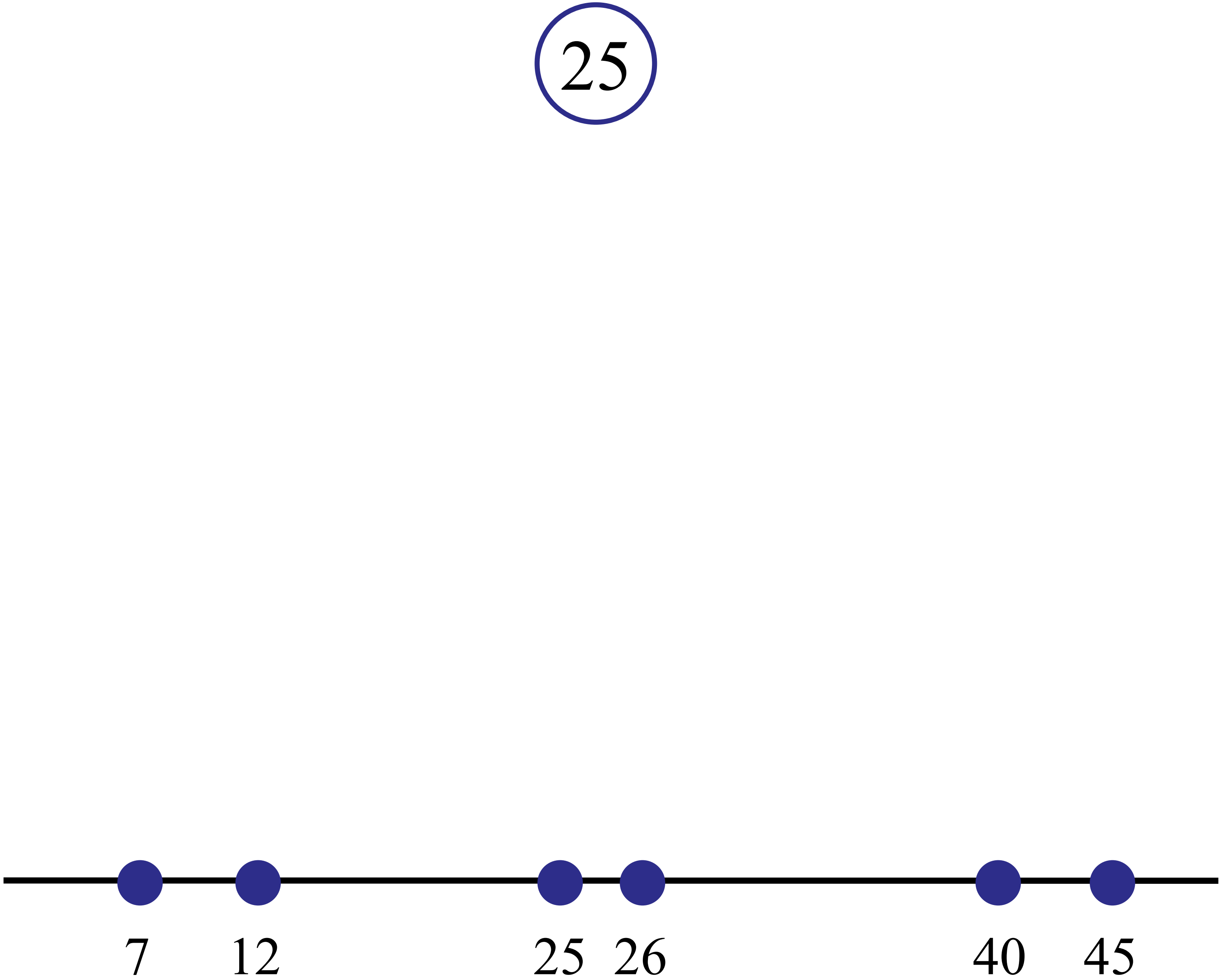
2. Store all points in the <u>leaves</u> of the tree. (Internal nodes store only copies.)

3. Each internal node v stores the MAX of any leaf in the <u>left</u> sub-tree.

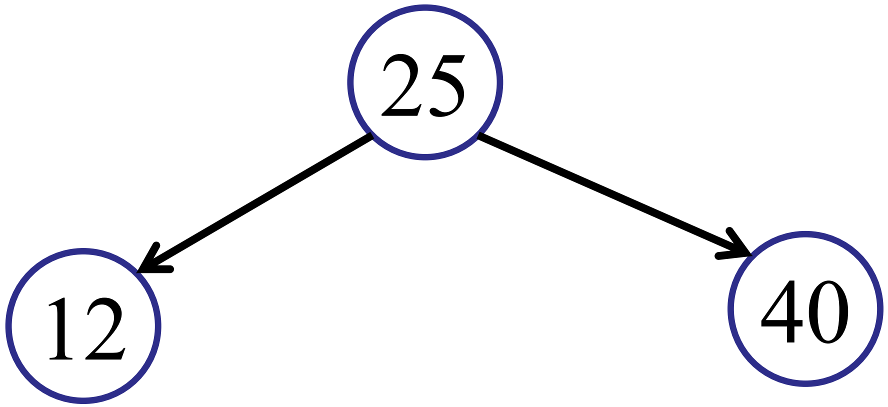
Example

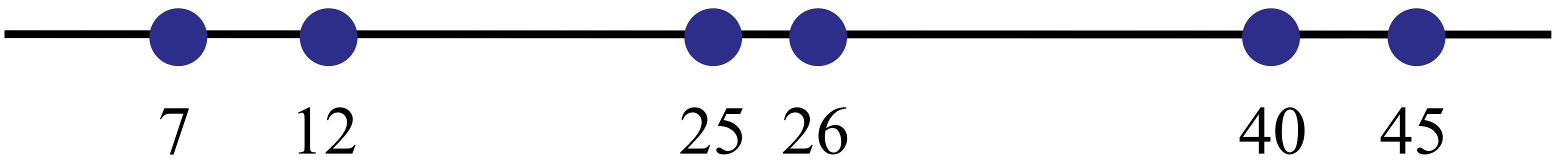


Example

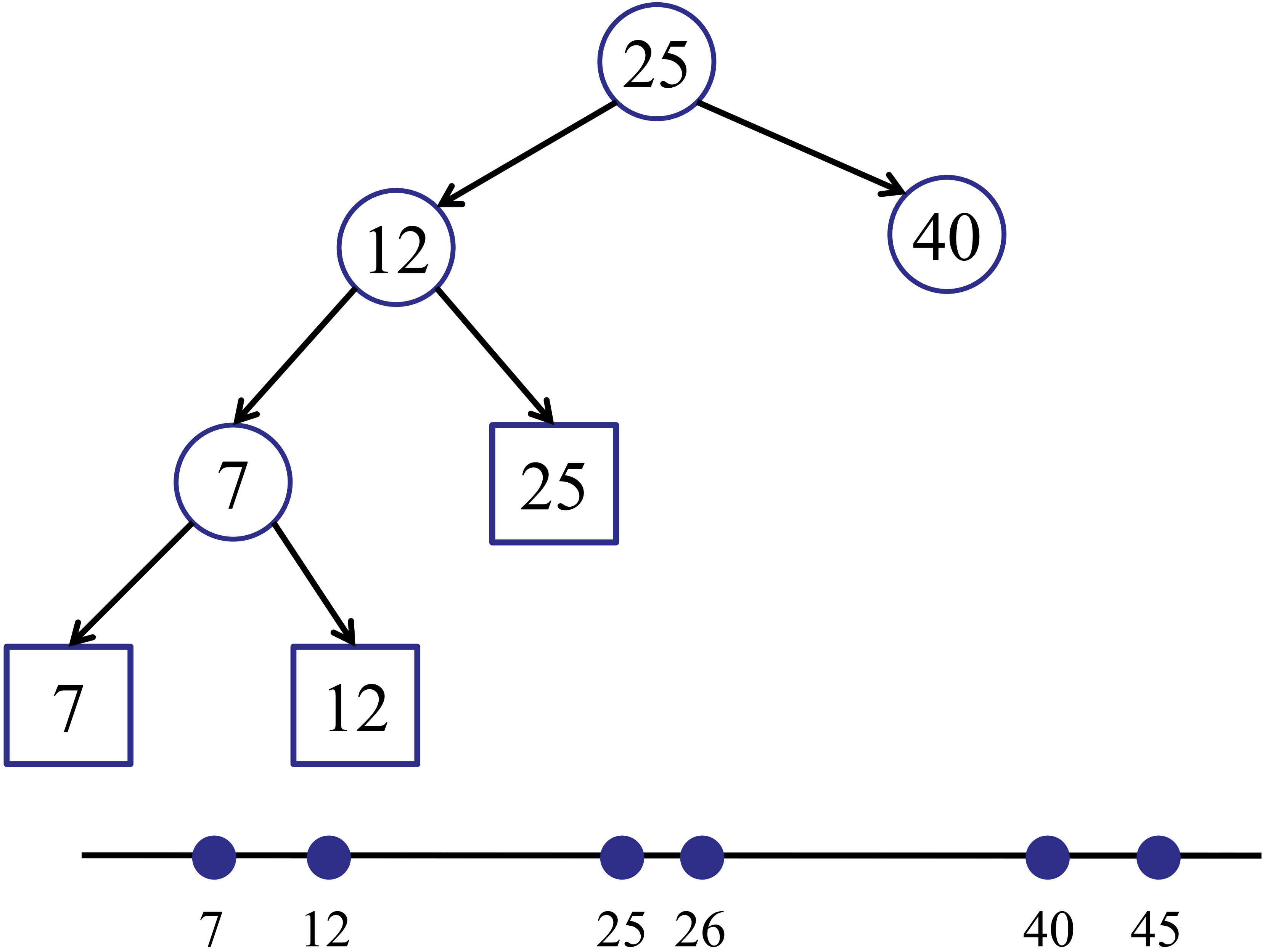


Example

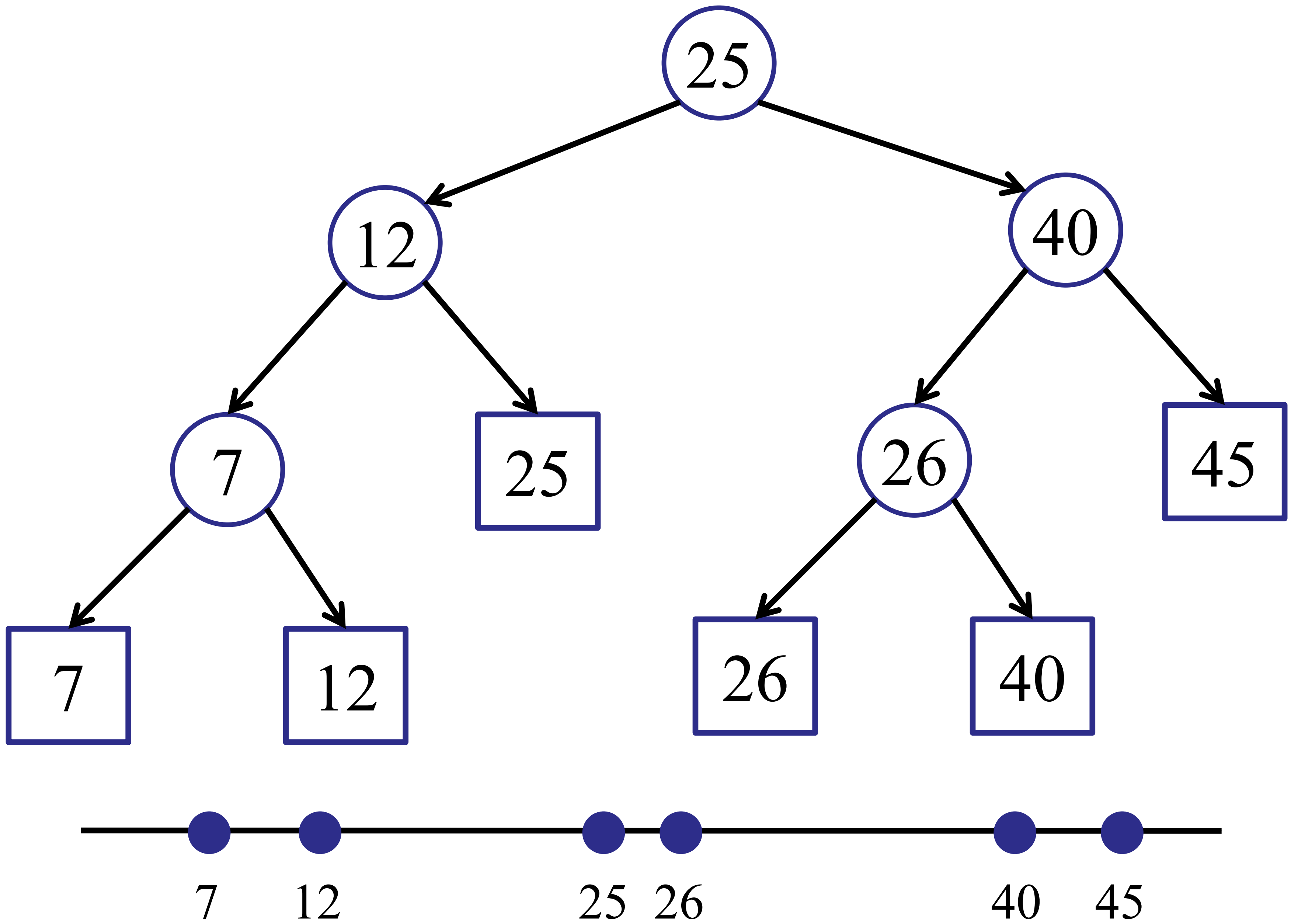




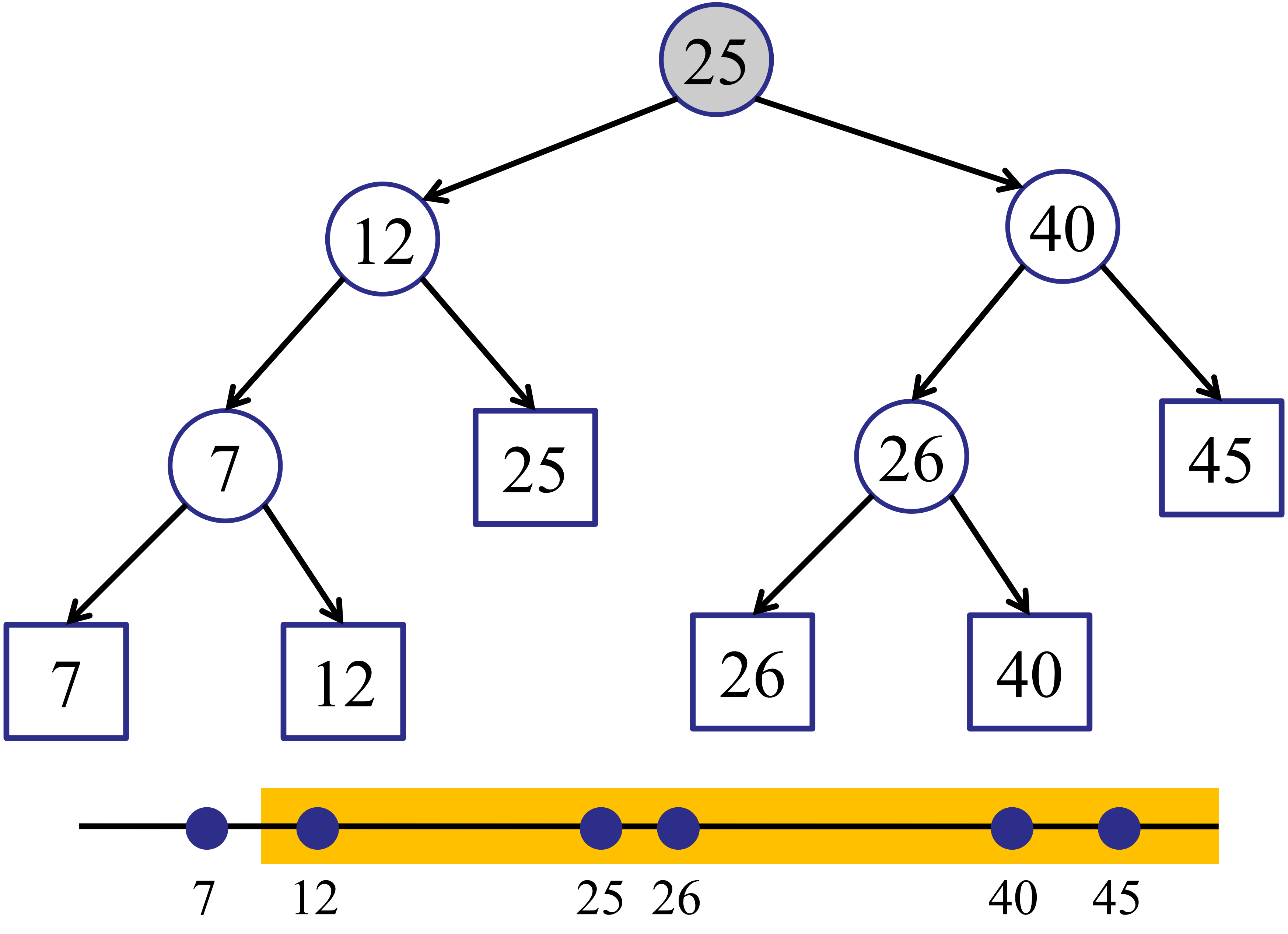
Example



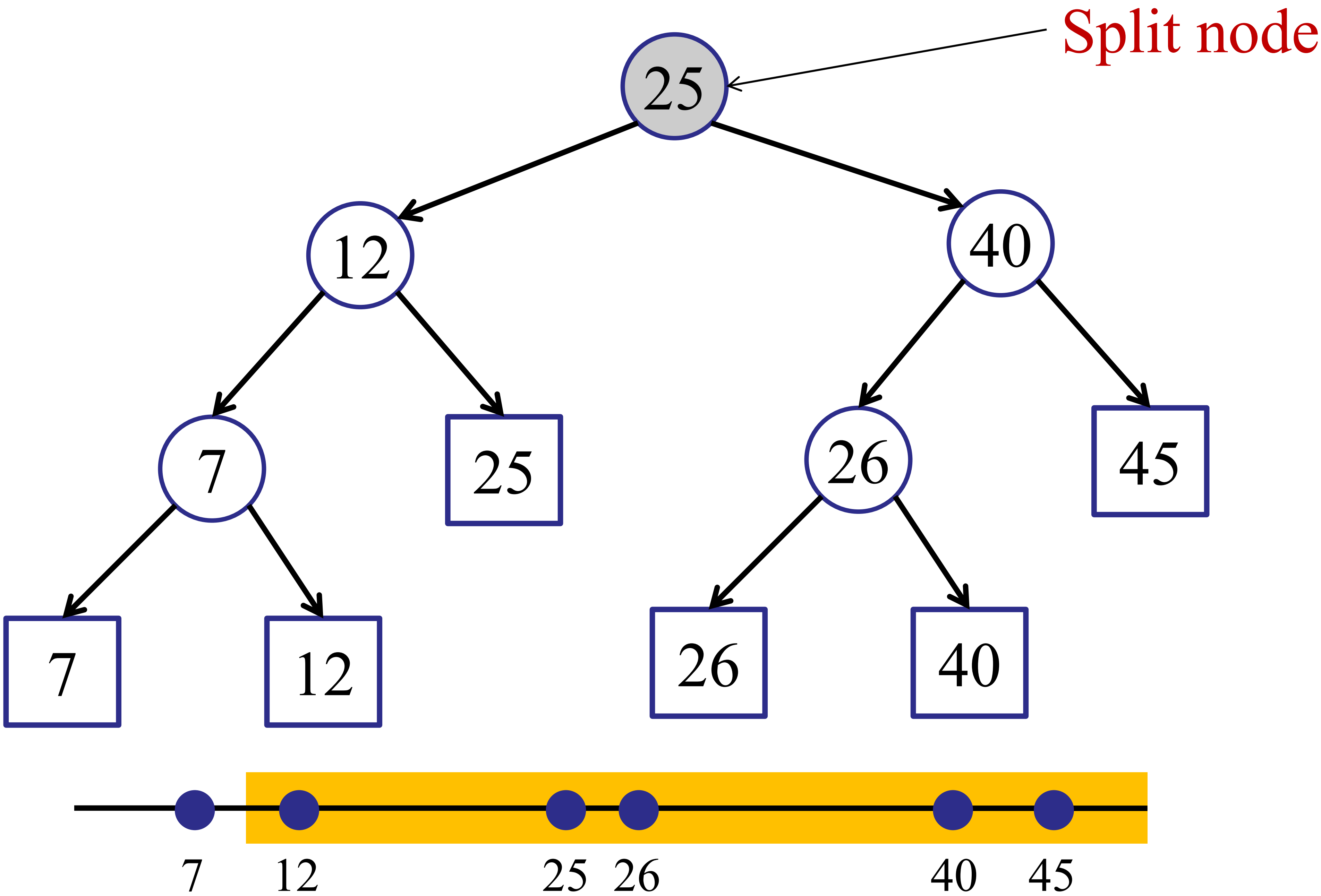
Note: BST Property



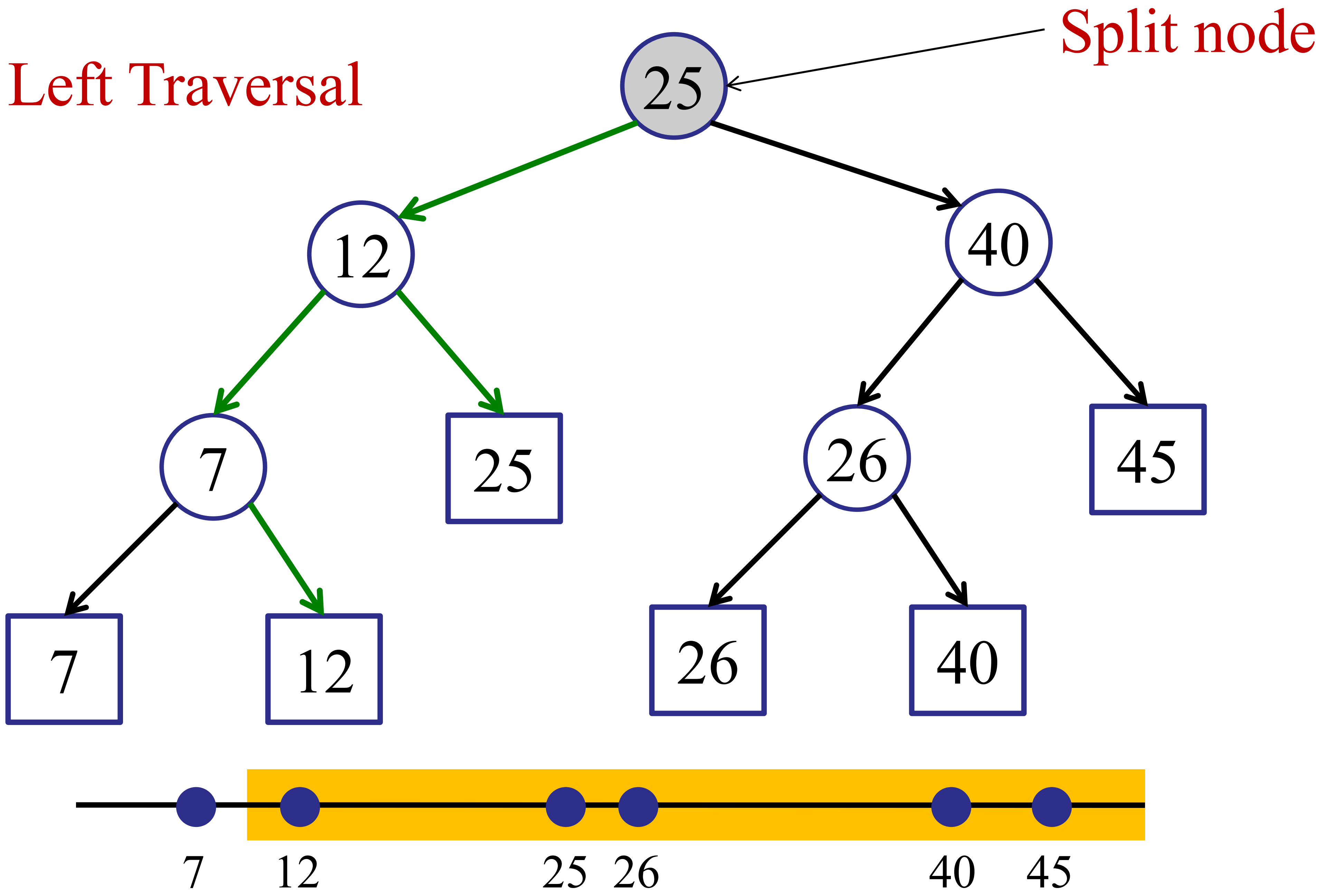
Example: query(10, 50)



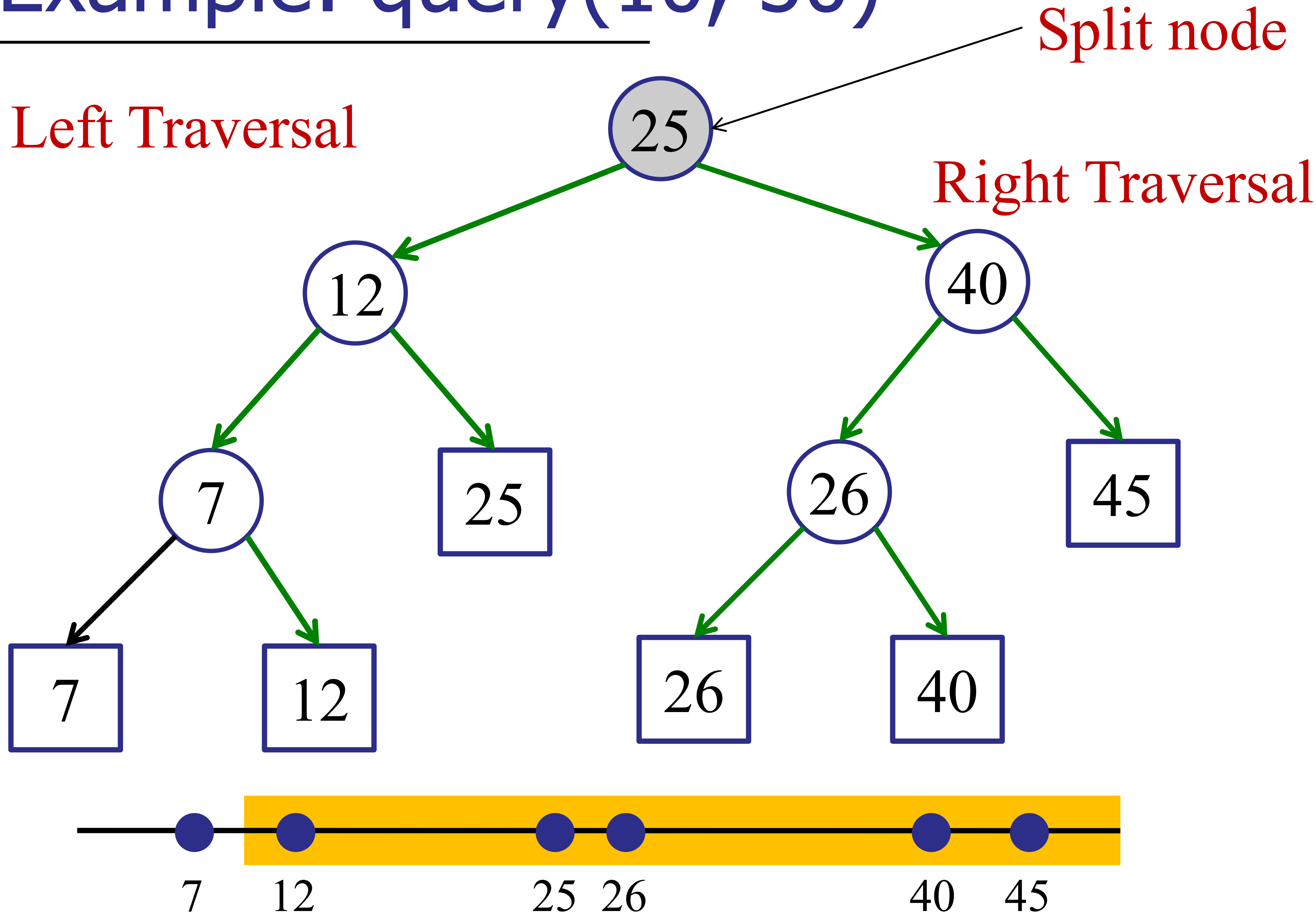
Example: query(10, 50)



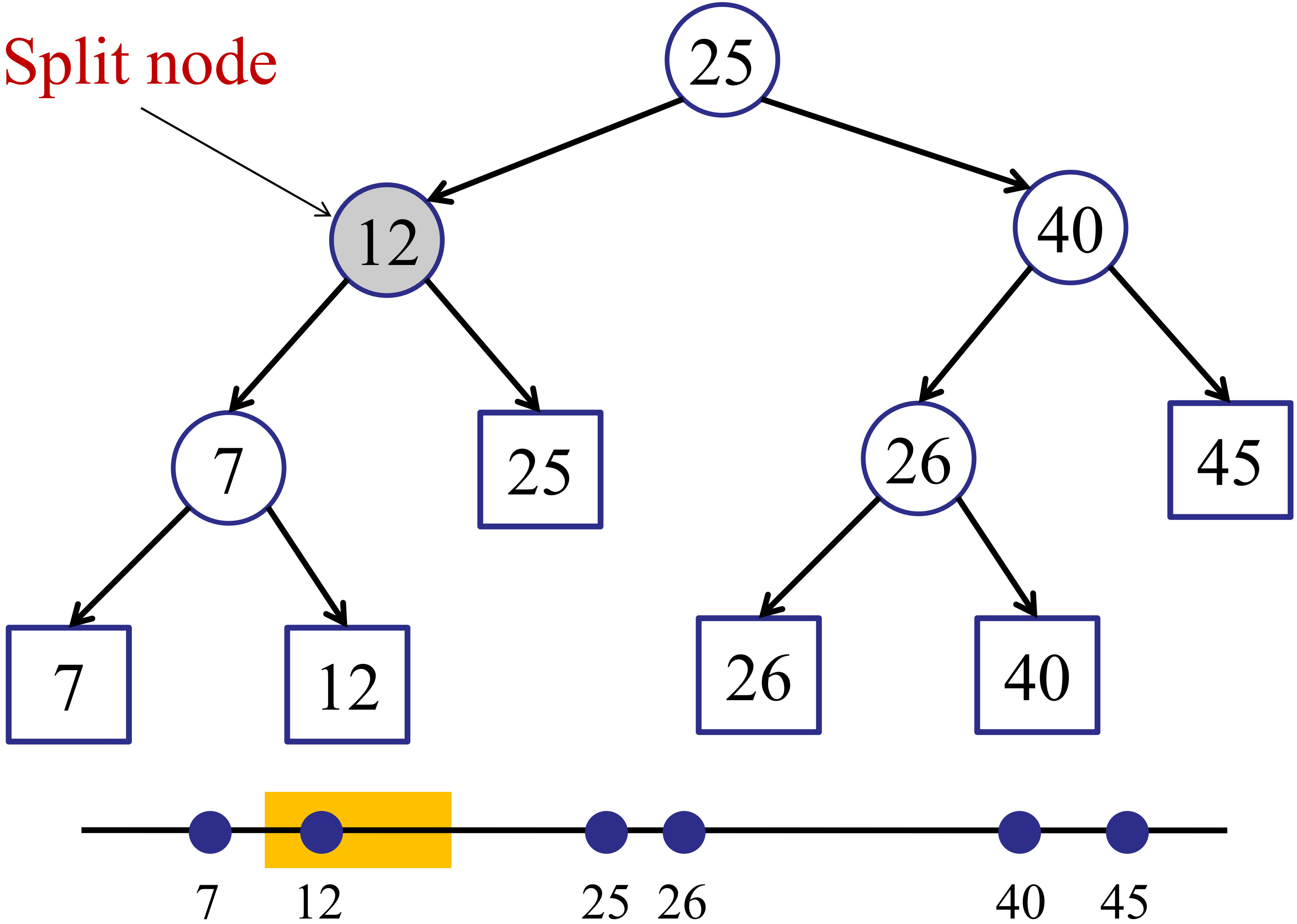
Example: query(10, 50)



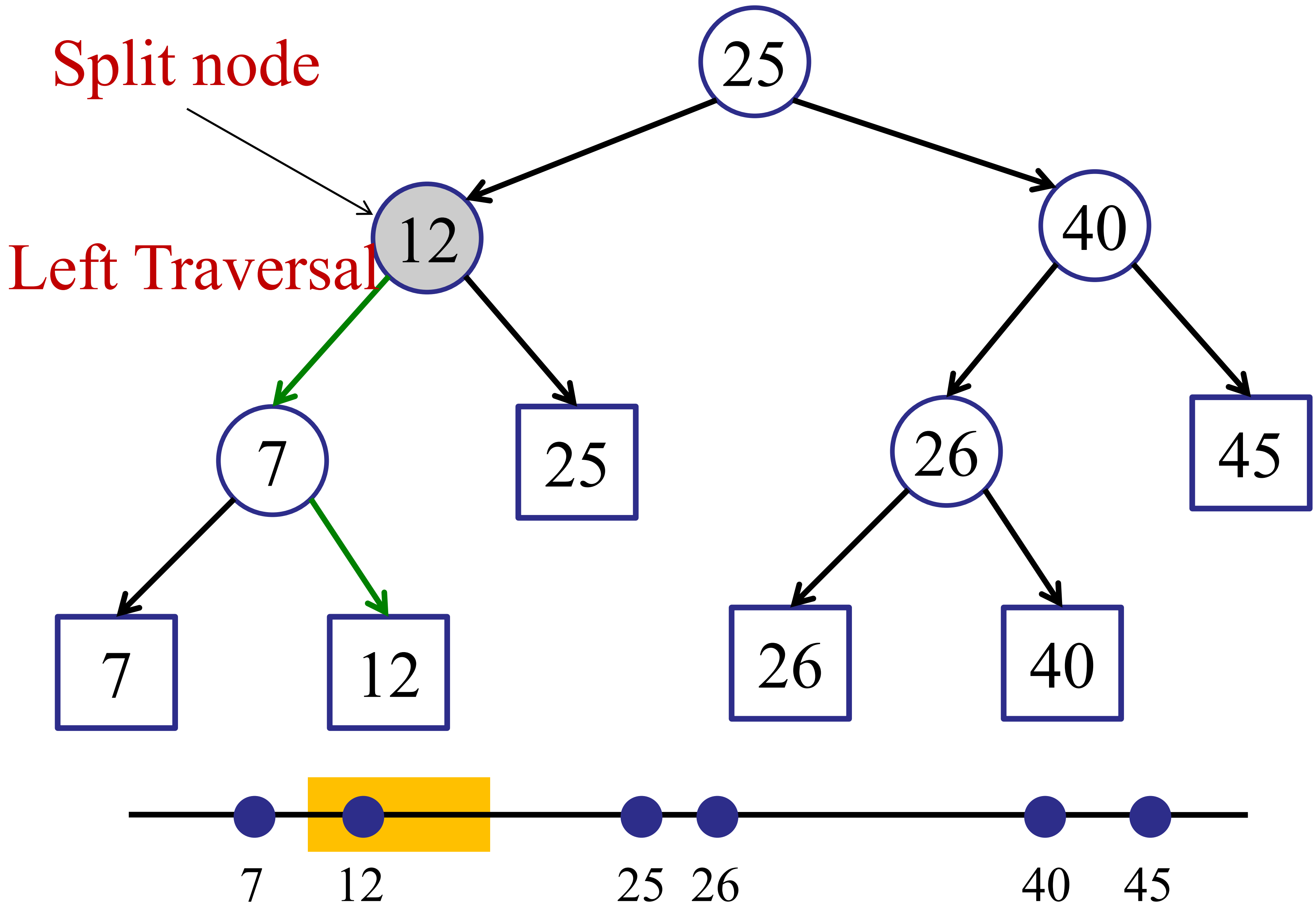
Example: query(10, 50)



Example: query(8, 20)



Example: query(8, 20)



One Dimensional Range Queries

Algorithm:

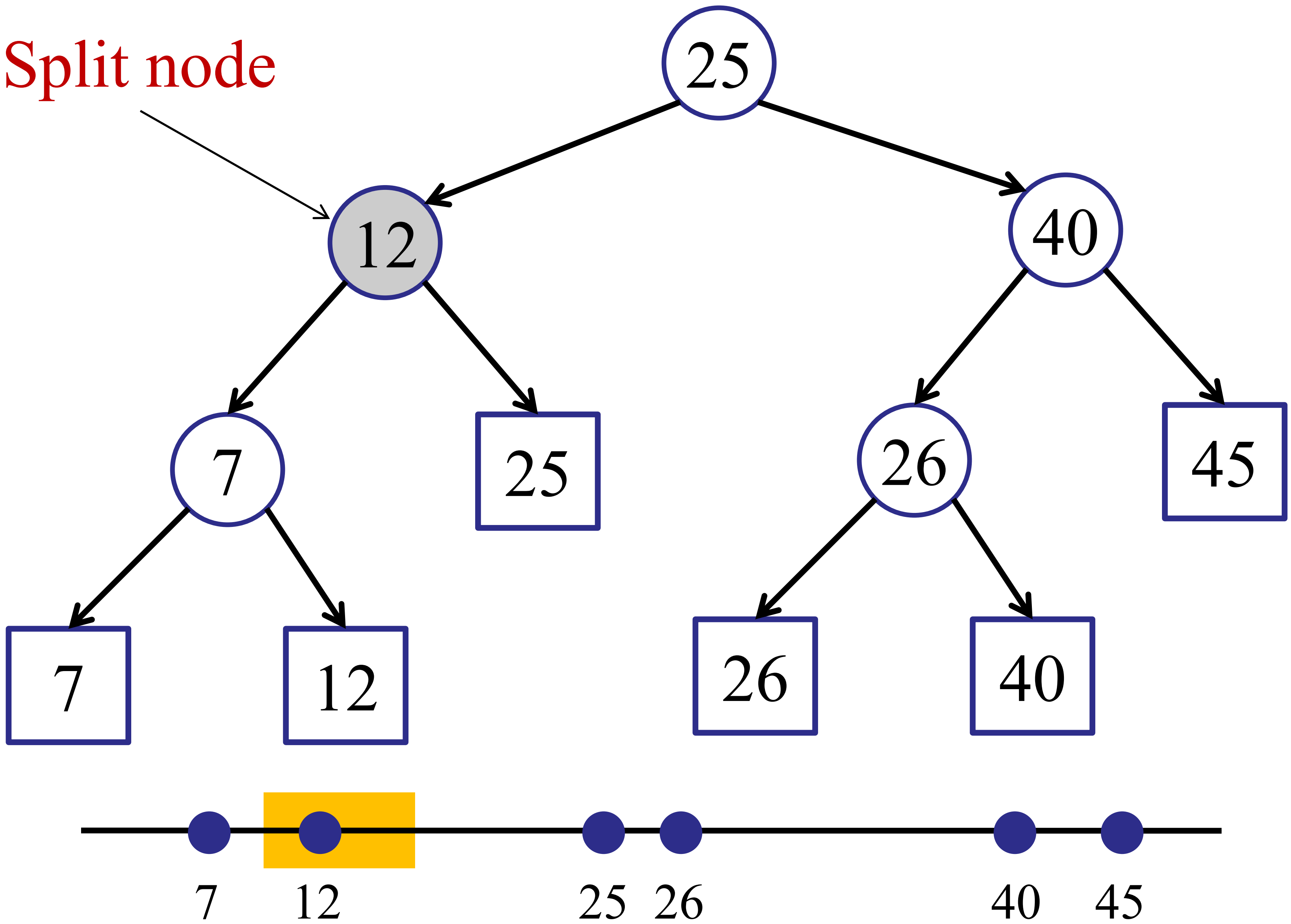
- Find “split” node.
- Do left traversal.
- Do right traversal.

One Dimensional Range Queries

FindSplit(low, high)

```
v = root;
done = false;
while !done {
    if (high <= v.key) then v=v.left;
    else if (low > v.key) then v=v.right;
    else (done = true);
}
return v;
```

Example: query(8, 20)



One Dimensional Range Queries

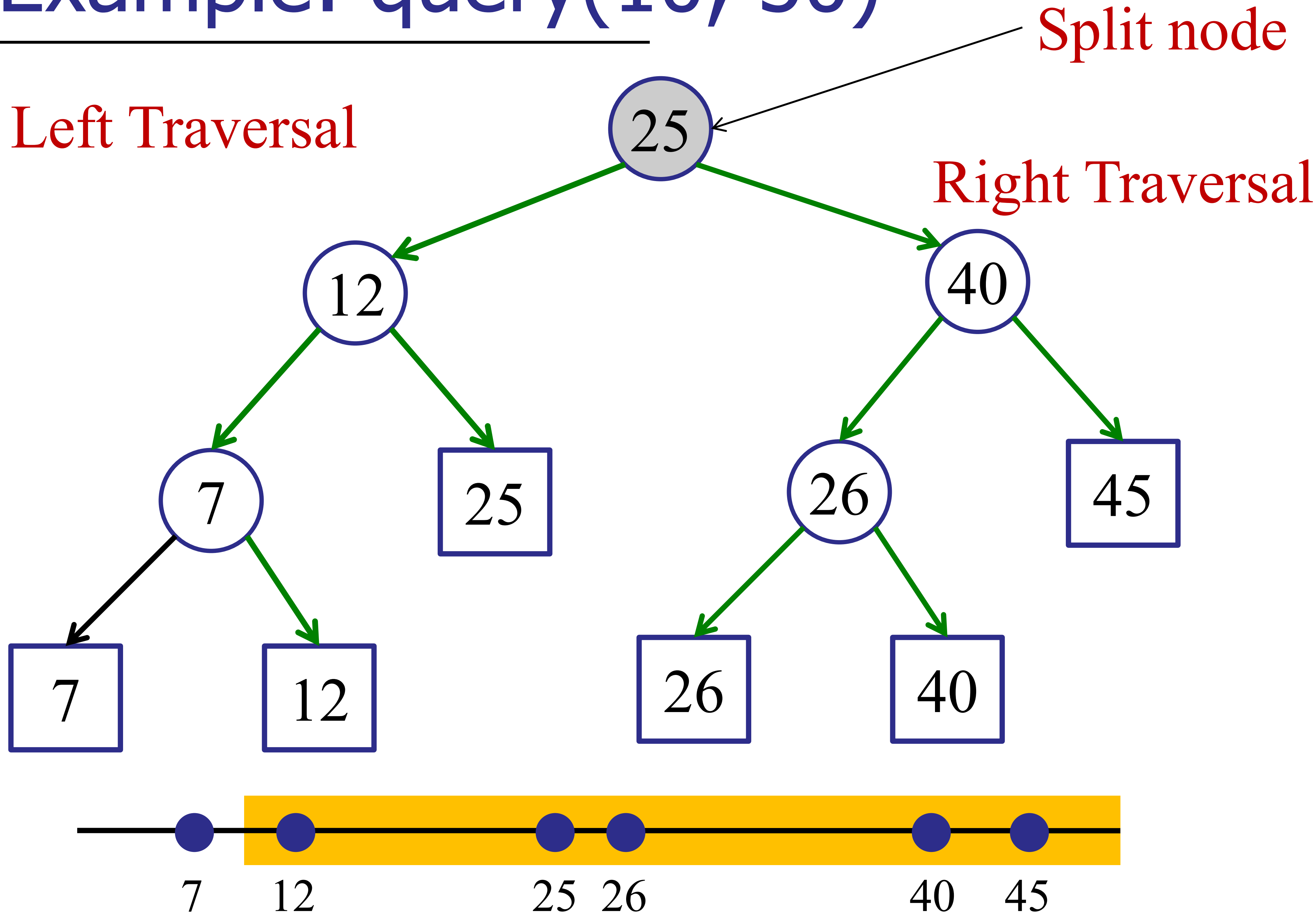
Algorithm:

- v = FindSplit(low, high);
- LeftTraversal(v, low, high);
- RightTraversal(v, low, high);

One Dimensional Range Queries

```
LeftTraversal(v, low, high)
    if (low <= v.key) {
        all-leaf-traversal(v.right);
        LeftTraversal(v.left, low, high);
    }
    else {
        LeftTraversal(v.right, low, high);
    }
}
```

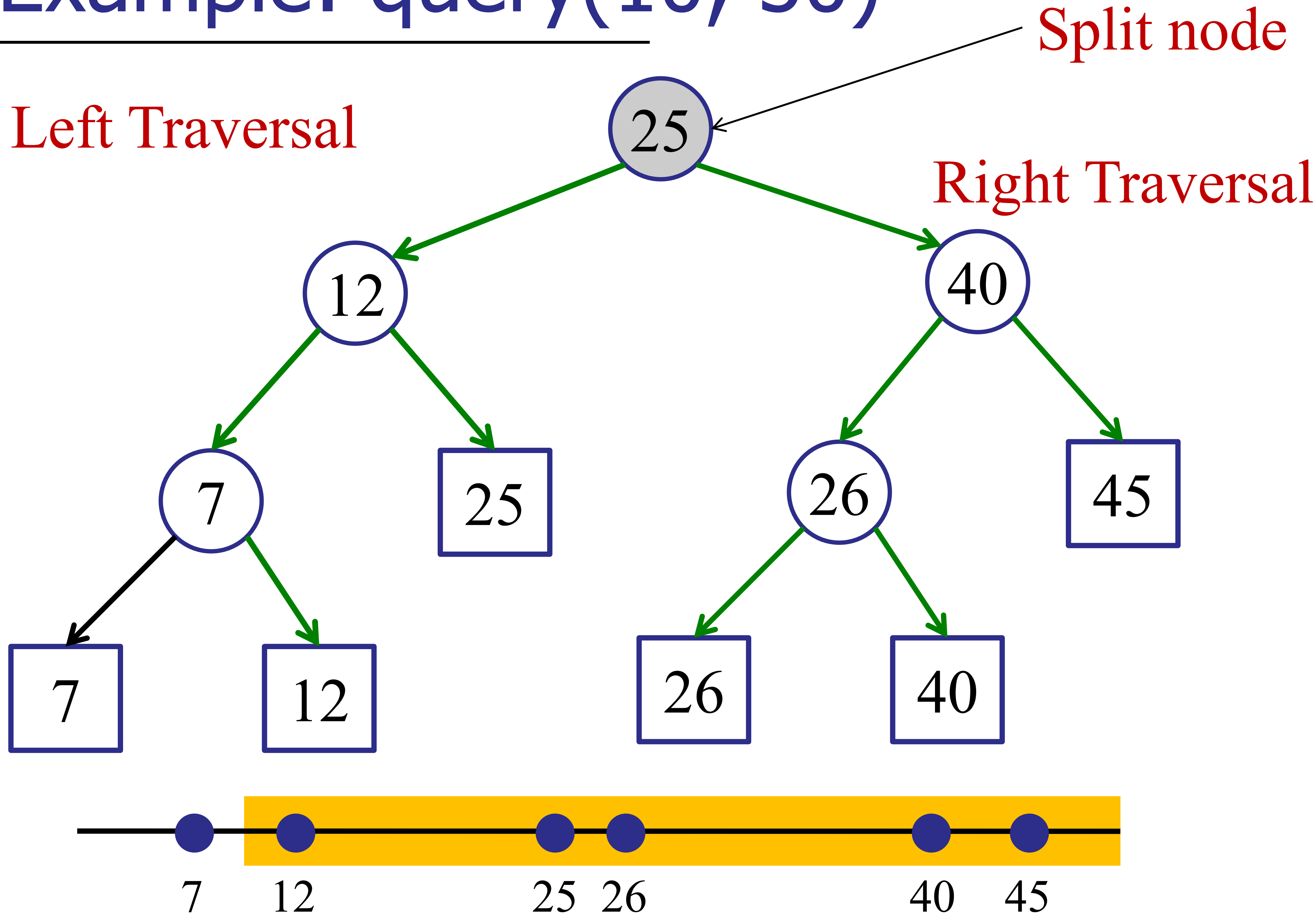
Example: query(10, 50)



One Dimensional Range Queries

```
LeftTraversal(v, low, high)
    if (low <= v.key) {
        all-leaf-traversal(v.right);
        LeftTraversal(v.left, low, high);
    }
    else {
        LeftTraversal(v.right, low, high);
    }
}
```

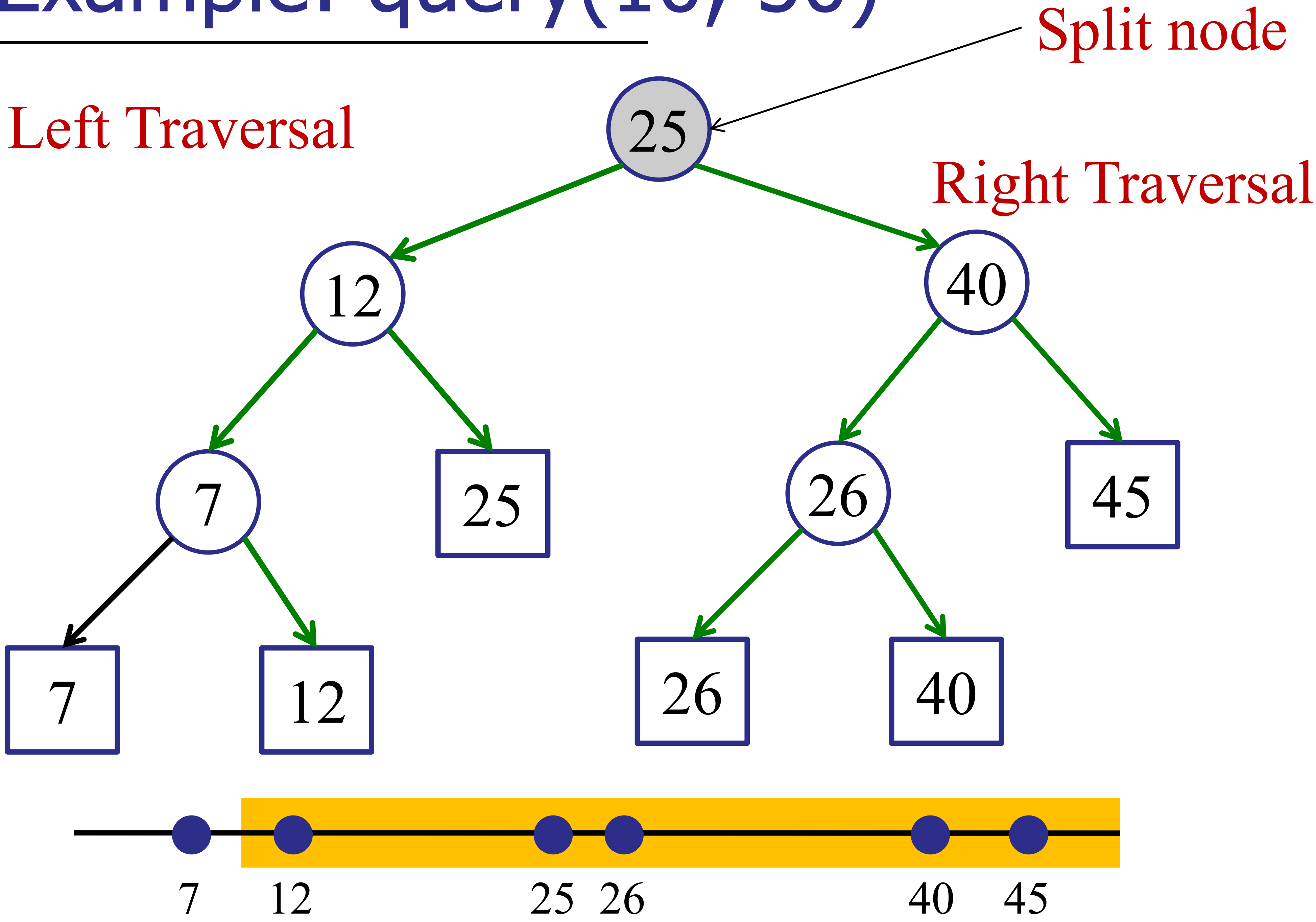
Example: query(10, 50)



One Dimensional Range Queries

```
RightTraversal(v, low, high)
    if (v.key <= high) {
        all-leaf-traverasal(v.left);
        RightTraversal(v.right, low, high);
    }
    else {
        RightTraversal(v.left, low, high);
    }
}
```

Example: query(10, 50)



One Dimensional Range Queries

Algorithm:

- v = FindSplit(low, high);
- LeftTraversal(v, low, high);
- RightTraversal(v, low, high);

Analysis

Query time:

- Finding split node: O(log n)
- Left Traversal:

  At every step, we either:

  1. Output all right sub-tree and recurse left.
  2. Recurse right.
- Right Traversal:

  At every step, we either:

  1. Output all left sub-tree and recurse right.
  2. Recurse left.

Analysis

Left Traversal:

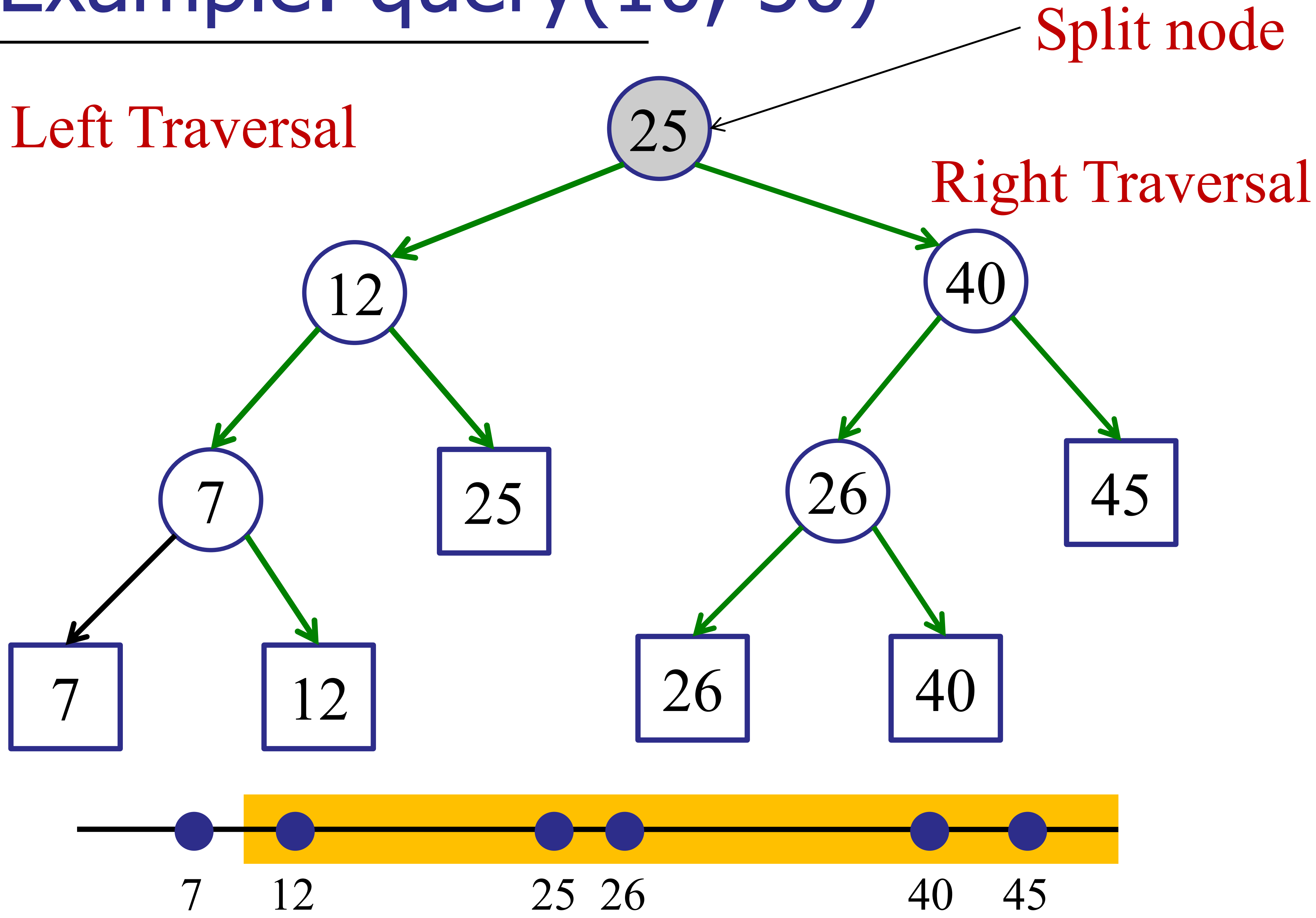
At every step, we either:

1. Output all right sub-tree and recurse left.
2. Recurse right.

Counting:

1. Recurse at most O(log n) times (i.e., option 2).
2. How expensive is “output all sub-tree” (i.e., option 1)?

Example: query(10, 50)



Analysis

Left Traversal:

At every step, we either:

1. Output all right sub-tree and recurse left.
2. Recurse right.

Counting:

1. Recurse at most O(log n) times (i.e., option 2).
2. How expensive is "output all sub-tree" (i.e., option 1)?

→ O(k), where k is number of items found.

Analysis

Query time complexity:

$O(k + \log n)$

where k is the number of points found.

Preprocessing (buildtree) time complexity:

$O(n \log n)$

Total space complexity:

$O(n)$

One Dimensional Range Queries

What if you just want to know *how many* points are in the range?

One Dimensional Range Queries

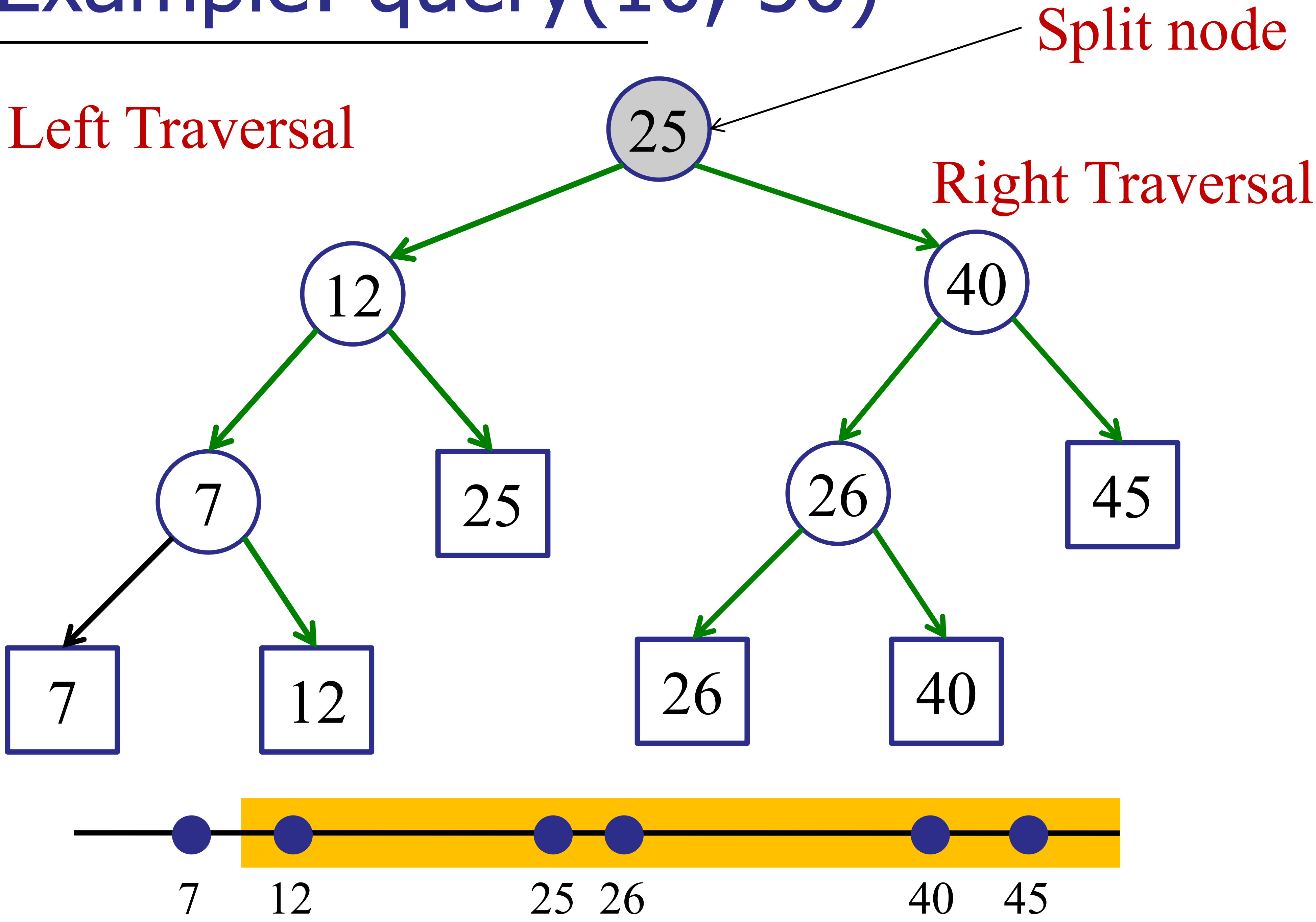
What if you just want to know *how many* points are in the range?

- Augment the tree!
- Keep a count of the number of nodes in each sub-tree.
- Instead of walking entire sub-tree, just remember the count.

One Dimensional Range Queries

```
LeftTraversal(v, low, high)
        if (low <= v.key) {
                all-leaf-traversal(v.right);
                total += v.right.count;
                LeftTraversal(v.left, low, high);
        }
        else {
                LeftTraversal(v.right, low, high);
        }
}
```

Example: query(10, 50)



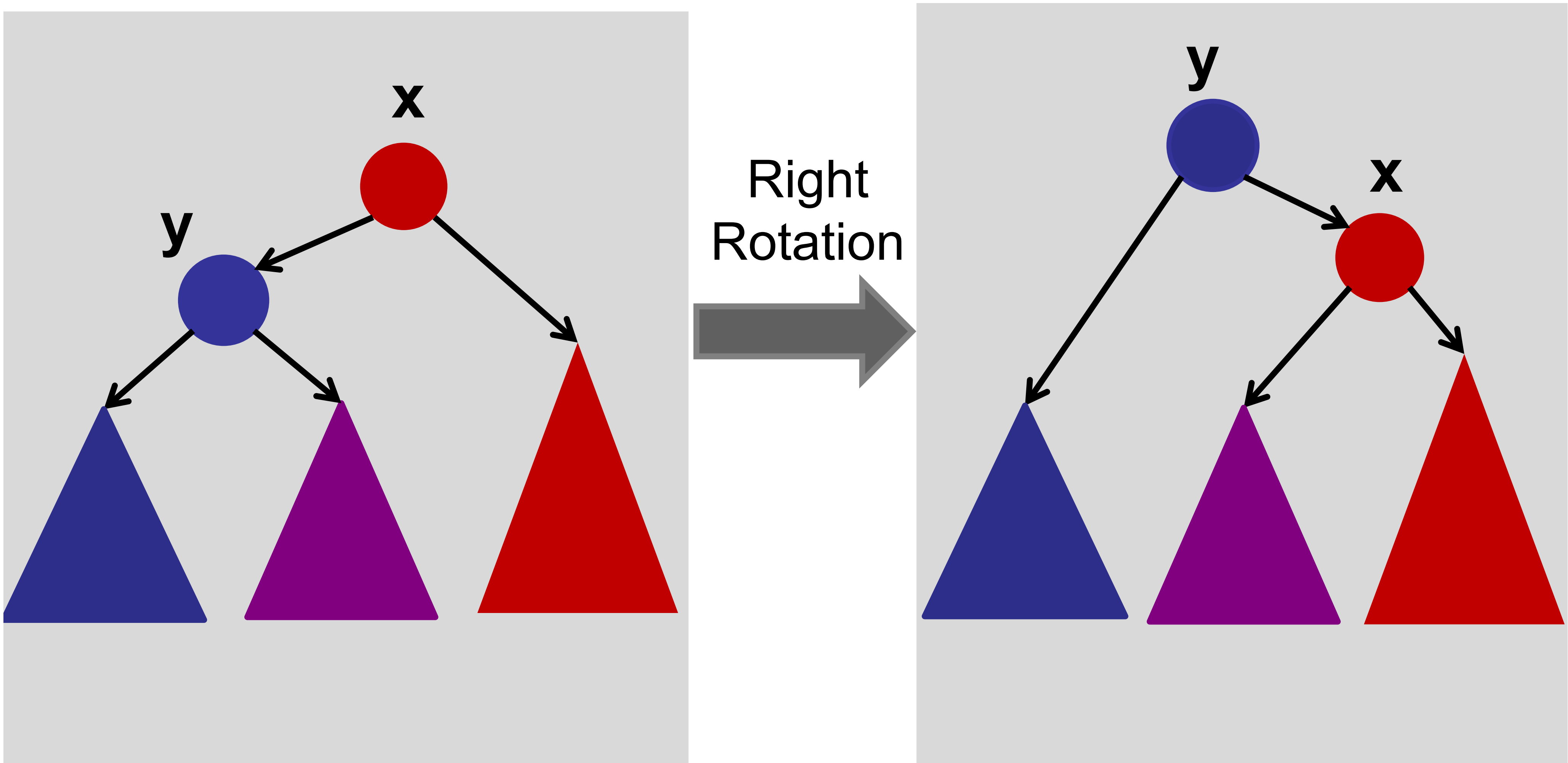
1D Range Tree

Done??

One Dimensional Range Queries

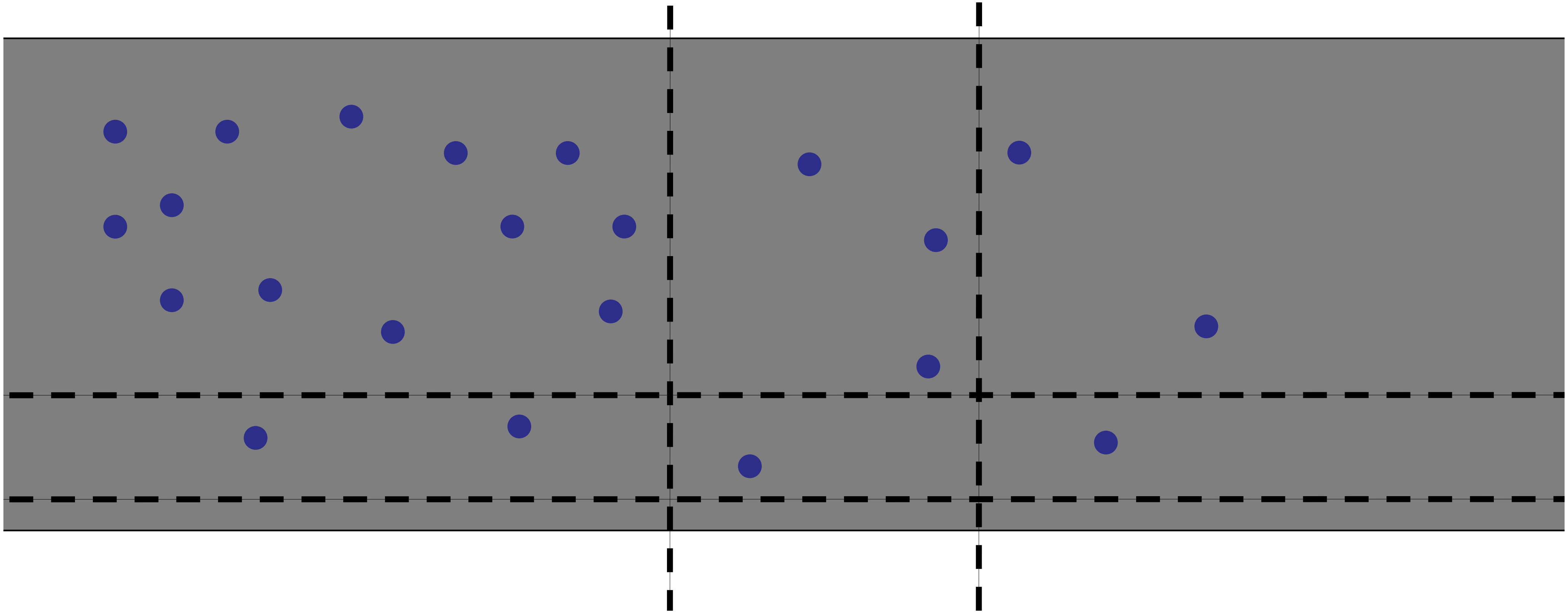
What about dynamic updates?

– Need to verify rotations!



Two Dimensional Range Tree

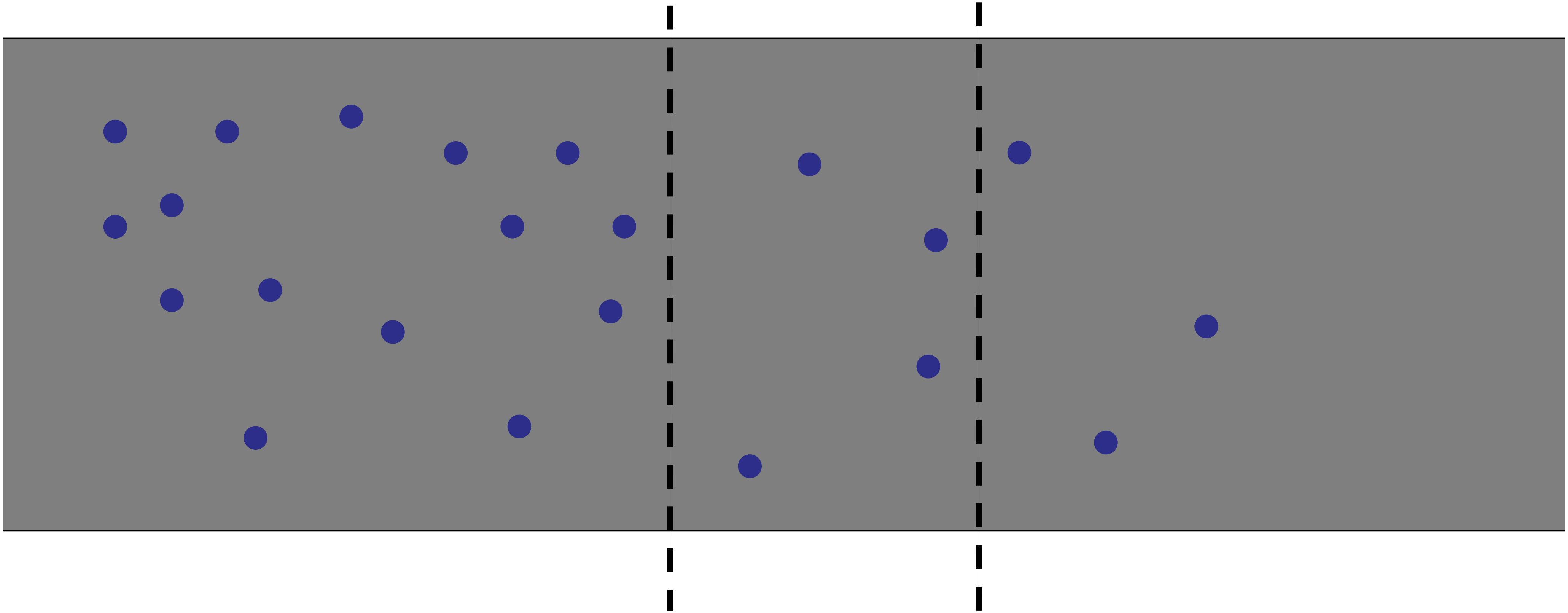
Ex: search for all points between dashed lines.



Two Dimensional Range Tree

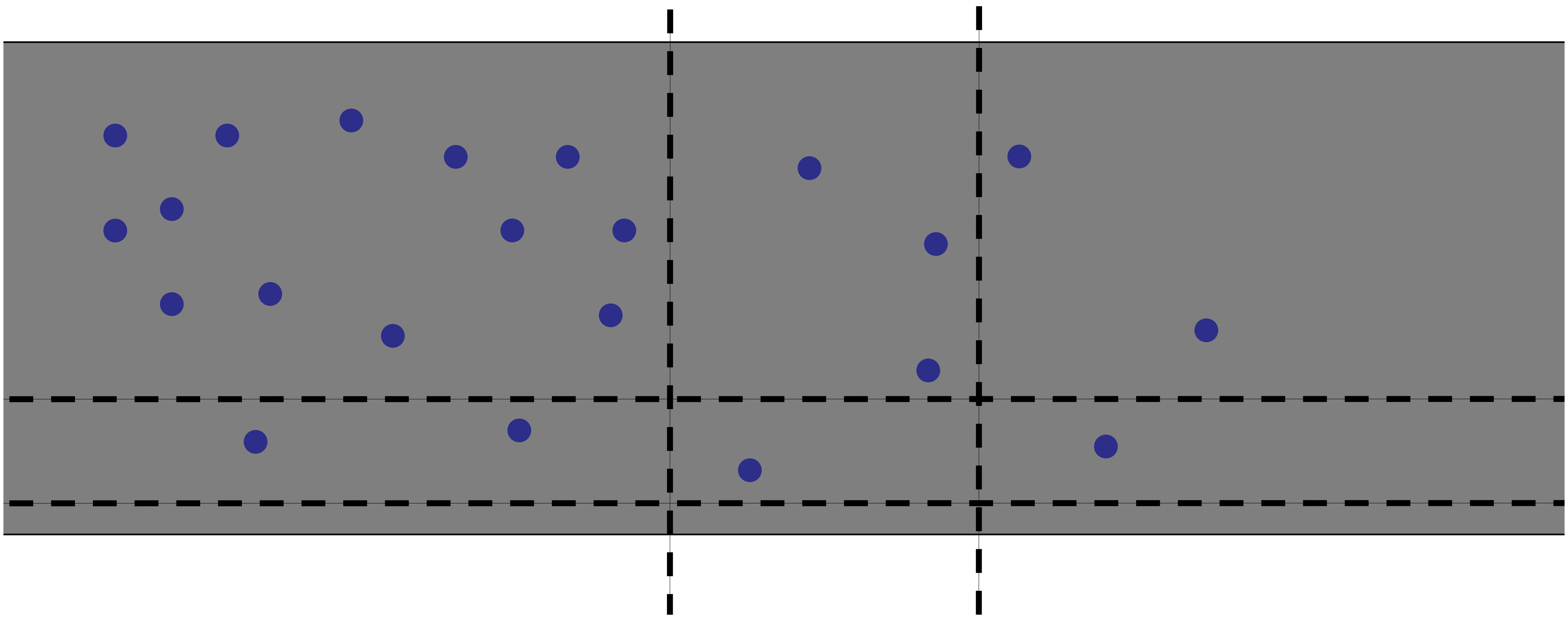
Step 1:

– Create a 1d-range-tree on the x-coords.



Two Dimensional Range Tree

Problem: can't enumerate entire sub-trees, since there may be too many nodes that don't satisfy the y-restriction.



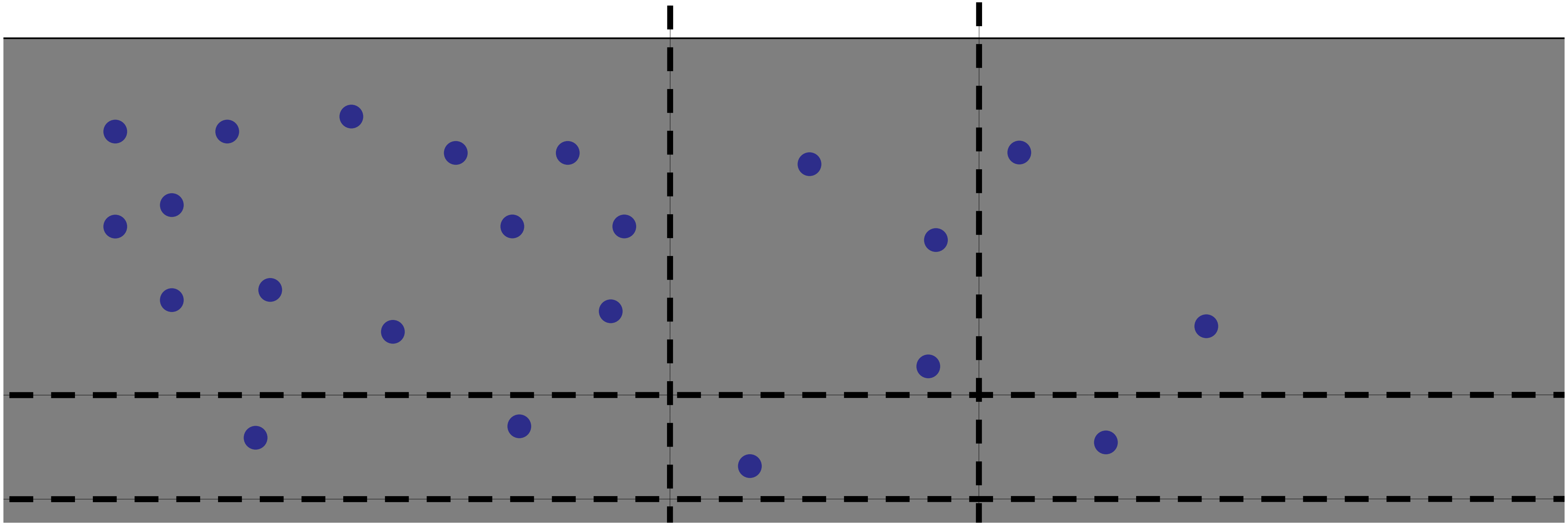
One Dimensional Range Queries

```
LeftTraversal(v, low, high)
    if (v.key >= low) {
            all-leaf-traversal(v.right);
            LeftTraversal(v.left, low, high);
    }
    else {
            LeftTraversal(v.right, low, high);
    }
}
```

Two Dimensional Range Tree

Solution: Augment!

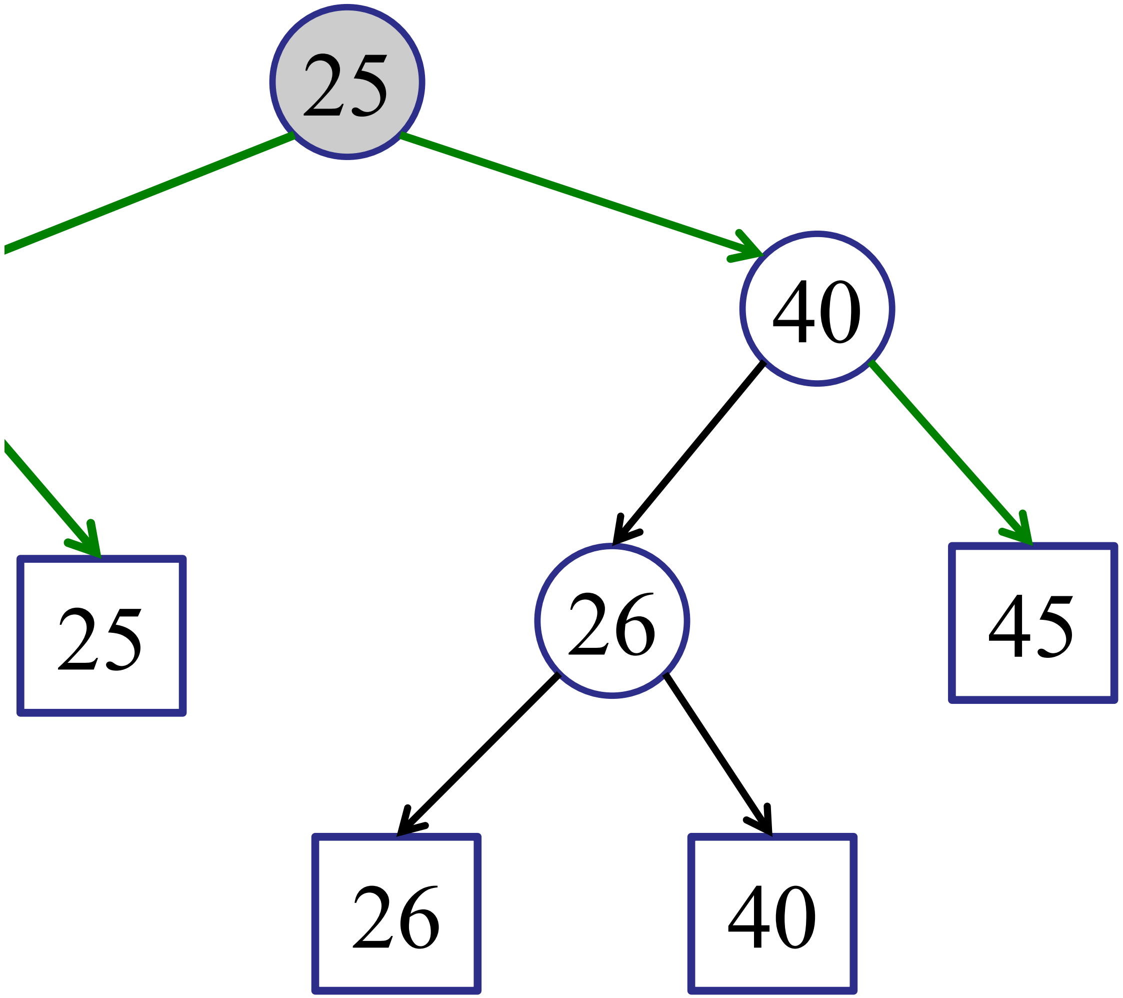
- Each node in the x-tree has a set of points in its sub-tree.
- Store a y-tree at each x-node containing all the points in the sub-tree.



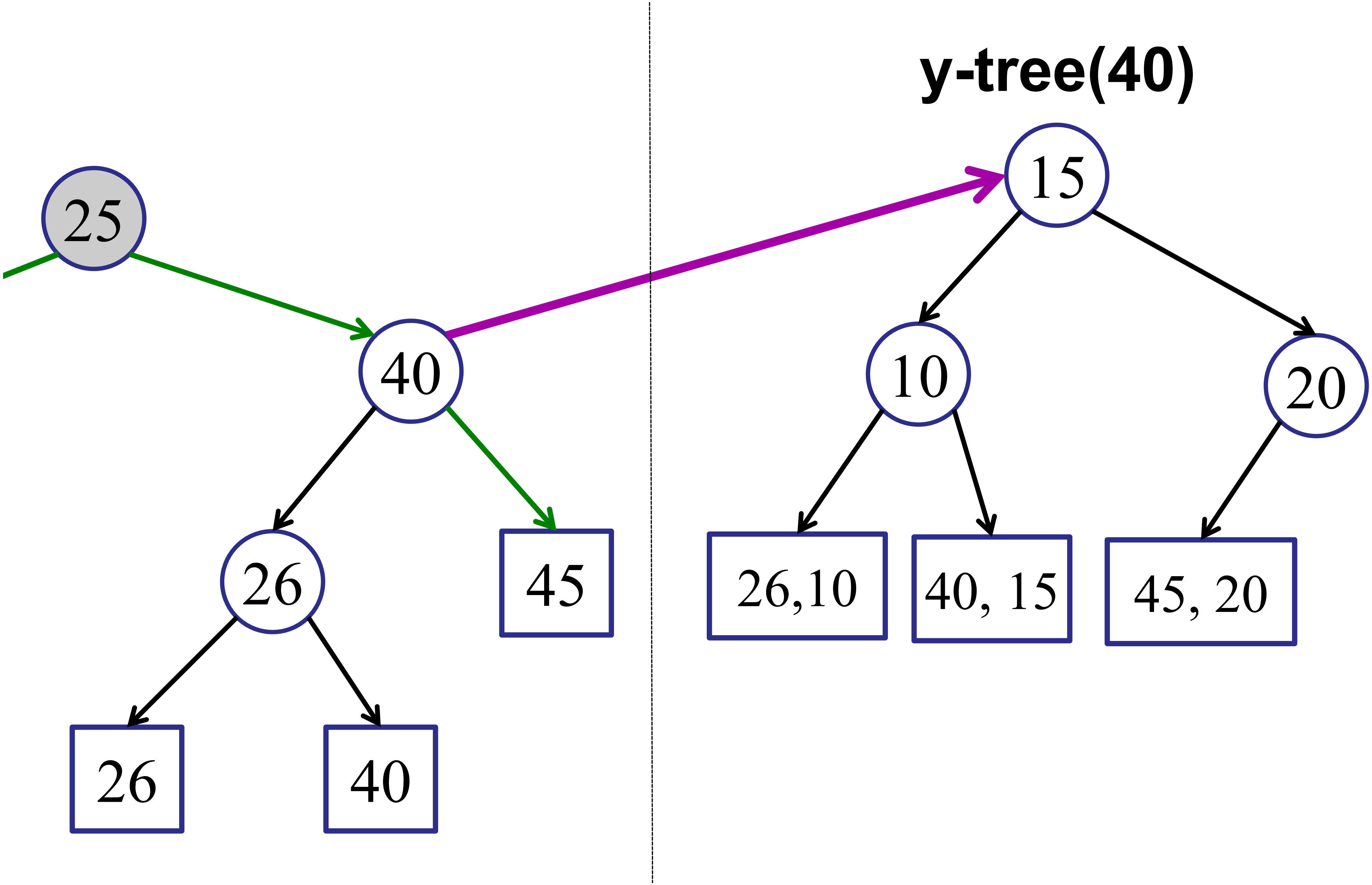
One Dimensional Range Queries

```
LeftTraversal(v, low, high)
    if (v.key.x >= low.x) {
        ytree.search(low.y, high.y);
        LeftTraversal(v.left, low, high);
    }
    else {
        LeftTraversal(v.right, low, high);
    }
}
```

Example:



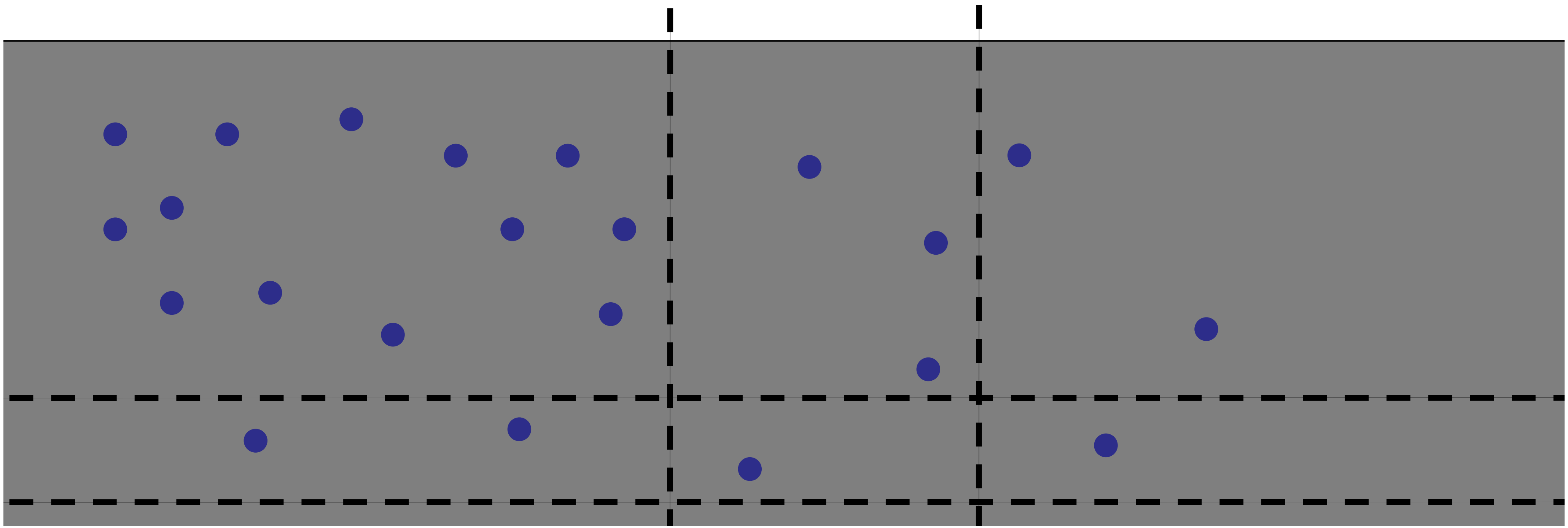
Example:



Two Dimensional Range Tree

Idea:

- Build an x-tree using only x-coordinates.
- For every node in the x-tree, build a y-tree out of nodes in subtree using only y-coordinates.



Analysis

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

- $O(\log n)$ to find split node.
- $O(\log n)$ recursing steps
- $O(\log n)$ y-tree-searches of cost $O(\log n)$
- $O(k)$ enumerating output

Analysis

Space complexity: O(n log n)

- Each point appears in at most one y-tree per level.
- There are O(log n) levels.

➔ Each node appears in at most O(log n) y-trees.

- The rest of the x-tree takes O(n) space.

Analysis

Building the tree: O(n log n)

- Tricky…
- Left as a puzzle… ☺

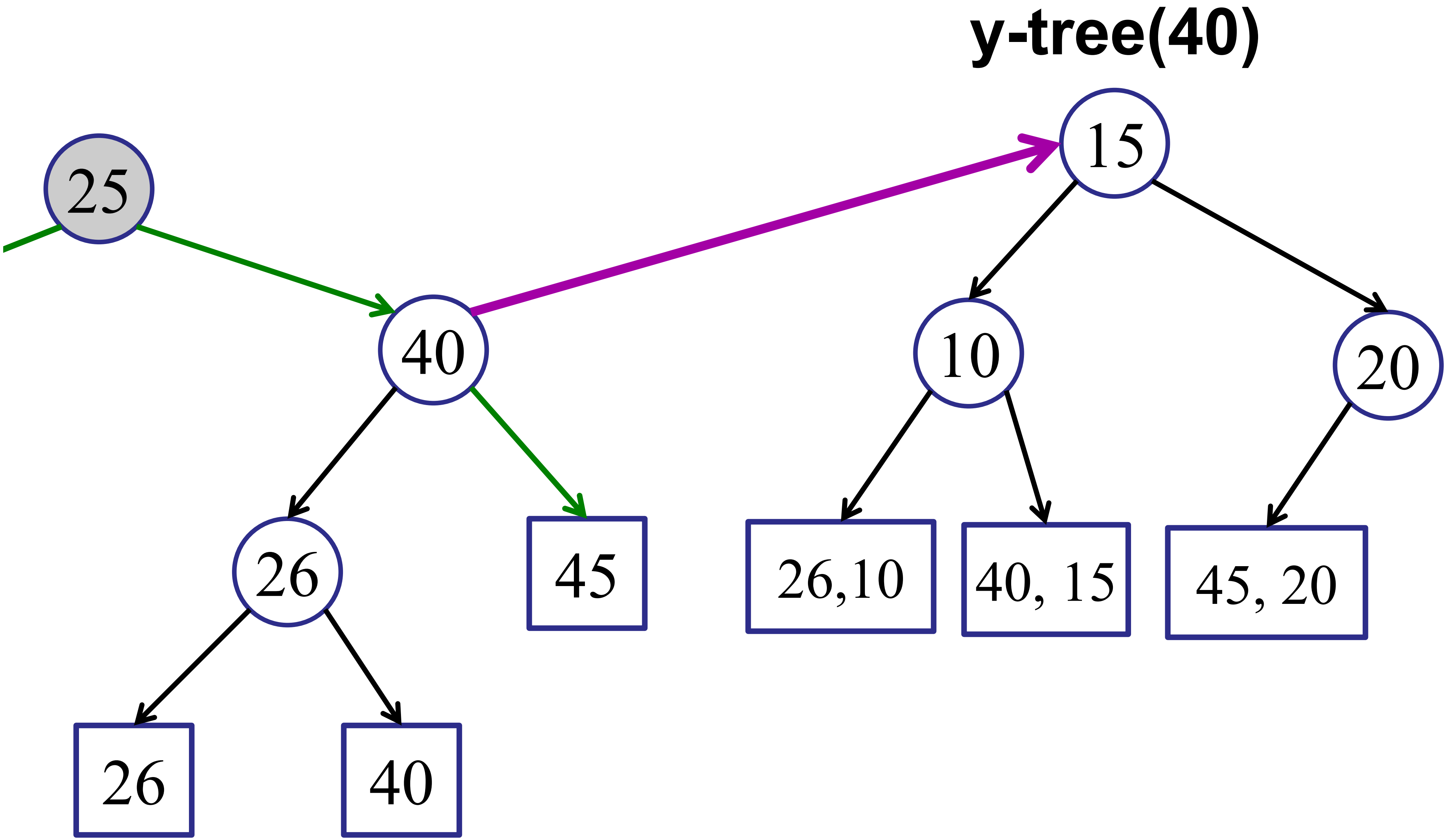
Challenge of the Day…

Dynamic Trees

What about inserting/deleting nodes?

- Hard!
- How do you do rotations?
- Every rotation you may have to entirely rebuild the y-trees for the rotated nodes.
- Cost of rotate: O(n)   !!!!

Example:



d-dimensional

What if you want high-dimensional range queries?

– Query cost: $O(\log^d n + k)$
– buildTree cost: $O(n \log^{d-1} n)$
– Space: $O(n \log^{d-1} n)$

Idea:

– Store d–1 dimensional range-tree in each node of a 1D range-tree.
– Construct the d–1-dimensionsal range-tree recursively.

Curse of Dimensionality

What if you want high-dimensional range queries?

- Query cost: $O(\log^d n + k)$
- buildTree cost: $O(n \log^{d-1} n)$
- Space: $O(n \log^{d-1} n)$

Idea:

- Store d–1 dimensional range-tree in each node of a 1D range-tree.
- Construct the d–1-dimensionsal range-tree recursively.

Real World (aside)

kd-Trees

- Alternate levels in the tree:
  - vertical
  - horizontal
  - vertical
  - horizontal
- Each level divides the points in the plane in half.

Real World (aside)

kd-Trees

- Alternate levels in the tree
- Each level divides the points in the plane in half.
- Query cost: $O(\sqrt{n})$ worst-case
  - Sometimes works better in practice for many queries.
  - Easier to update dynamically.
  - Good for other types of queries: e.g., nearest-neighbor

Today

Three examples of augmenting BSTs

1. Order Statistics

2. Intervals

3. Orthogonal Range Searching