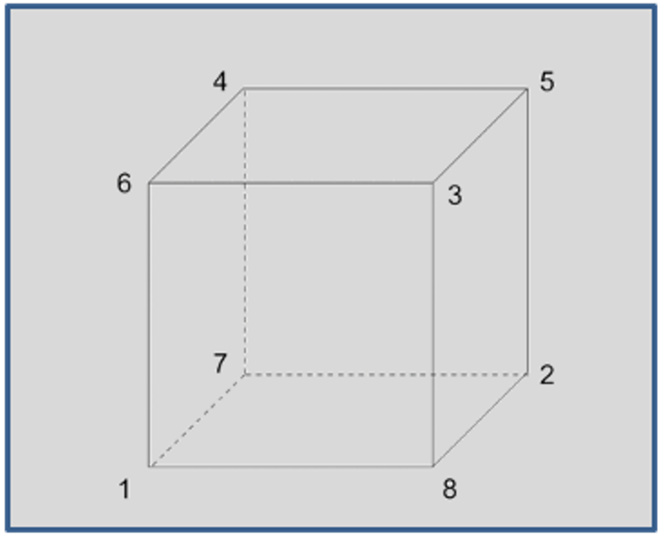
CS2040S
Data Structures and Algorithms

Augmented Trees!

Example: all FaceSums = 18



Puzzle of the Week: Prime Cubes

A cube has eight vertices.

Assign a prime number to each vertex.

A FaceSum is the sum of the four vertices on a face.

Can you find an assignment of prime numbers so that all six FaceSums are equal?

Where we are…

Dictionaries

Binary search trees

Tries

Balanced search trees

- AVL trees
- Scapegoat Trees
- B-trees

Admin

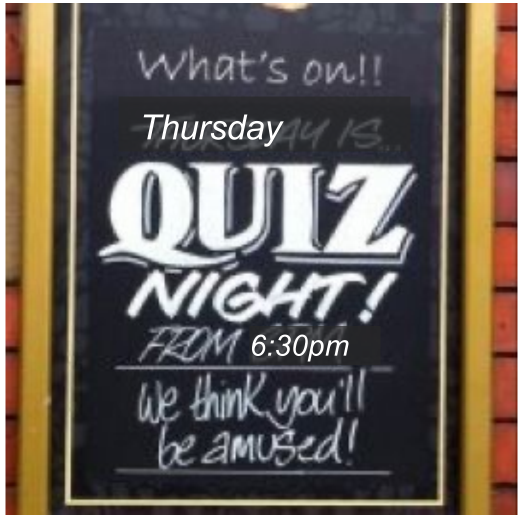
Coursemology Survey

- Anonymous
- Goal: to understand how things are going
- Goal: calibration
- Goal: feedback to TAs and tutors.
- Goal: feedback to me!

Appreciate constructive feedback!

Also appreciate some understanding when things haven't gone perfectly.
Remember, the tutors are *also* students with problem sets, midterms, etc.

Midterm



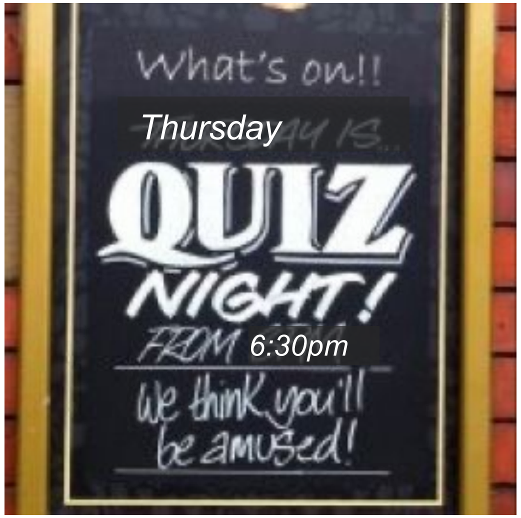
Thurs. March 10 6:30pm

- Location: MPSH
- < 50 students / rooms TBA
- 13+ rooms

Bring to quiz:

- One double-sided sheet of paper with any notes you like.
- Pens/pencils.
- You may not use anything else. (No calculators, no phones, etc.)

Midterm

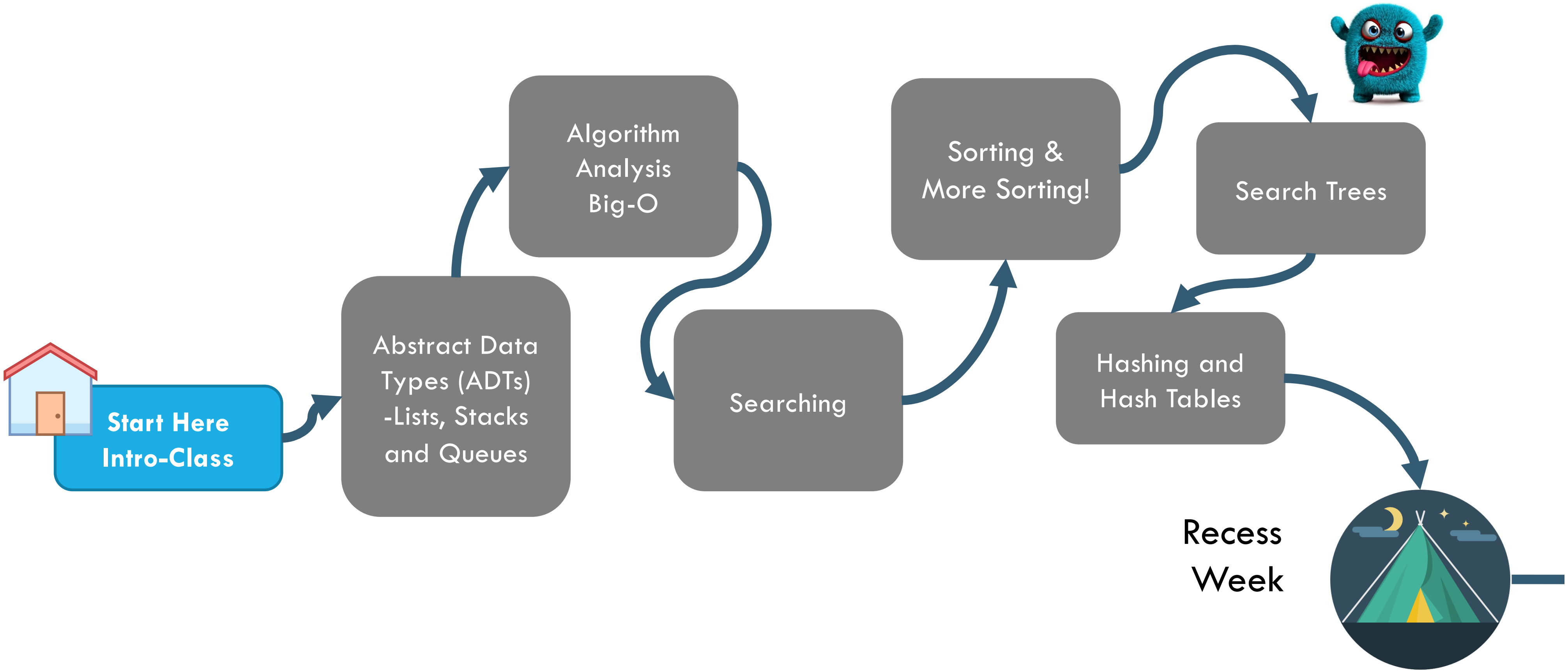


Midterm covers everything through the end of this week.

Practice Material:

- Posted on Coursemology
- Beware previous years were not necessarily the same as this year.

What have we done so far?



Part I: Organizing your Data

Range trees

Interval trees

Order statistics

AVL

B-trees

Binary Search

SelectionSort

So much material?

BubbleSort

InsertionSort

PancakeSort

QuickSelect

QuickSort

ReversalSort

BogoSort

MergeSort

Binary tree

CardFlipTrees

Hash table

Tries

Random Permutations

Range trees

Interval trees

Order statistics

AVL

B-trees

Binary Search

SelectionSort

Three Parts

BubbleSort

InsertionSort

QuickSelect

1) Basic Theory / Algorithm Analysis

2) Major Algorithms

3) Overarching Principles

Basic Theory: Key Topics

- Asymptotic notation (big-O, etc.)
- Simple recurrences
- Asymptotic analysis
- Basic probability (e.g., CS1231 review)

Key Algorithms

- Binary Search
- Sorting
- Balanced binary trees
- Augmented trees
- Hash table (introduction, Wednesday)

Key Ideas

- Basic strategies for solving problems
- Invariants
- Trade-offs: how to choose which data structure
- Augmenting data structures

Basic strategies for solving problems

Try something simple

E.g., naïve search

Reduce-and-conquer

E.g., binary search

Divide-and-conquer

E.g., MergeSort

Maintaining an invariant

E.g., AVL trees

E.g., keep your data sorted

Augment an existing data structure

E.g., AVL trees

Two important questions:

How do you understand what an algorithm is doing?

How do you show that an algorithm is correct?

To understand algorithms:

For every algorithm in the class, identify all of its important invariants.

Range trees

Interval trees

Order statistics

AVL

B-trees

Binary Search

SelectionSort

Three Parts

BubbleSort

InsertionSort

QuickSelect

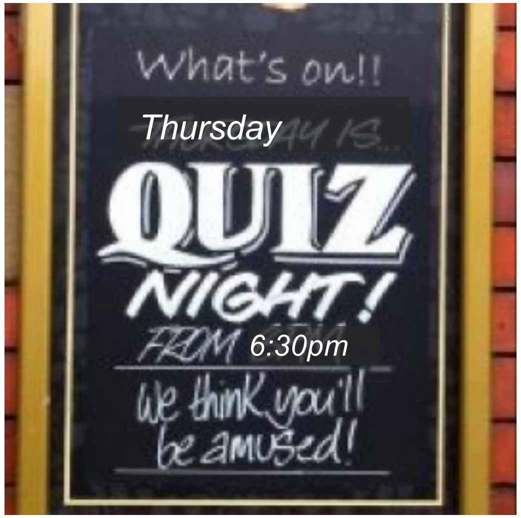
1) Basic Theory / Algorithm Analysis

2) Major Algorithms

3) Overarching Principles

Permutations

Midterm



Thurs. March 10 6:30pm

- Location TBA (MPSH)
- < 50 students / room
- 13+ rooms

Bring to quiz:

- One double-sided sheet of paper with any notes you like.
- Pens/pencils.
- You may not use anything else. (No calculators, no phones, etc.)

Dynamic Data Structures

1. Maintain a set of items

2. Modify the set of items

3. Answer queries.

Big picture idea:

Trees are a good way to store, summarize, and search dynamic data.

Dynamic Data Structures

- Operations that create a data structure
  - build (preprocess)
- Operations that modify the structure
  - insert
  - delete
- Query operations
  - search, select, etc.

Augmented Data Structures

Many problems require storing additional data in a standard data structure.

Augment more frequently than invent…

Plan

Three examples of augmenting balanced BSTs

1. Order Statistics

2. Interval Queries

3. Orthogonal Range Searching

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

    (tree, hash table, linked list, stack, etc.)

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

3. Modify data structure to *maintain* additional info when the structure changes.

   (subject to insert/delete/etc.)

Augmenting data structures

Basic methodology:

1. Choose underlying data structure

   (tree, hash table, linked list, stack, etc.)

2. Determine additional info needed.

3. Modify data structure to *maintain* additional info when the structure changes.

   (subject to insert/delete/etc.)

4. Develop new operations.

Plan

Three examples of augmenting balanced BSTs

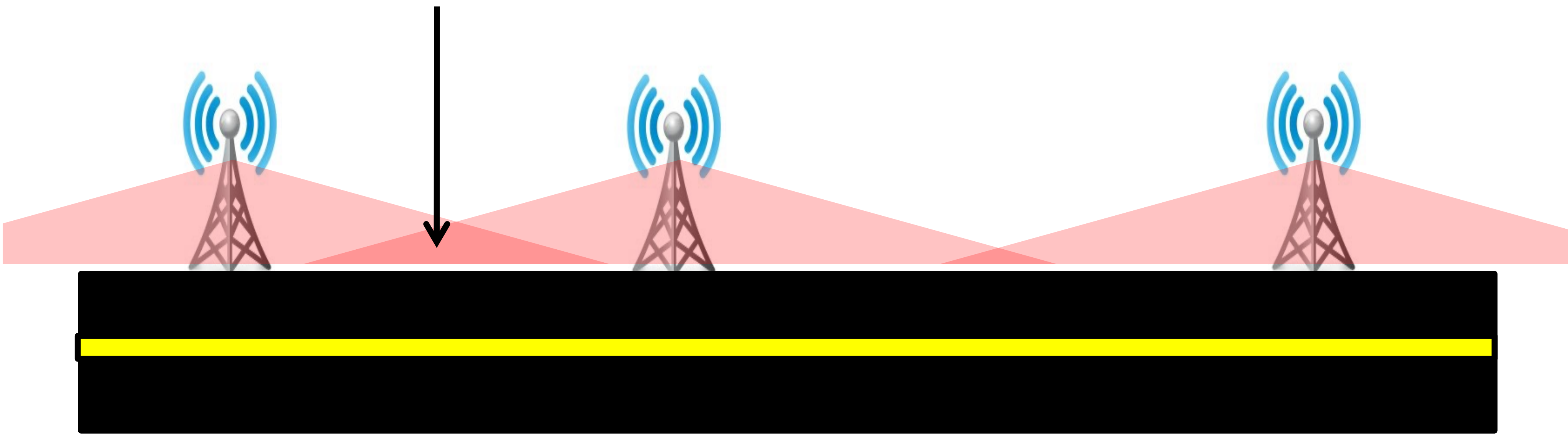
1. ~~Order Statistics~~

2. Interval Queries

3. Orthogonal Range Searching

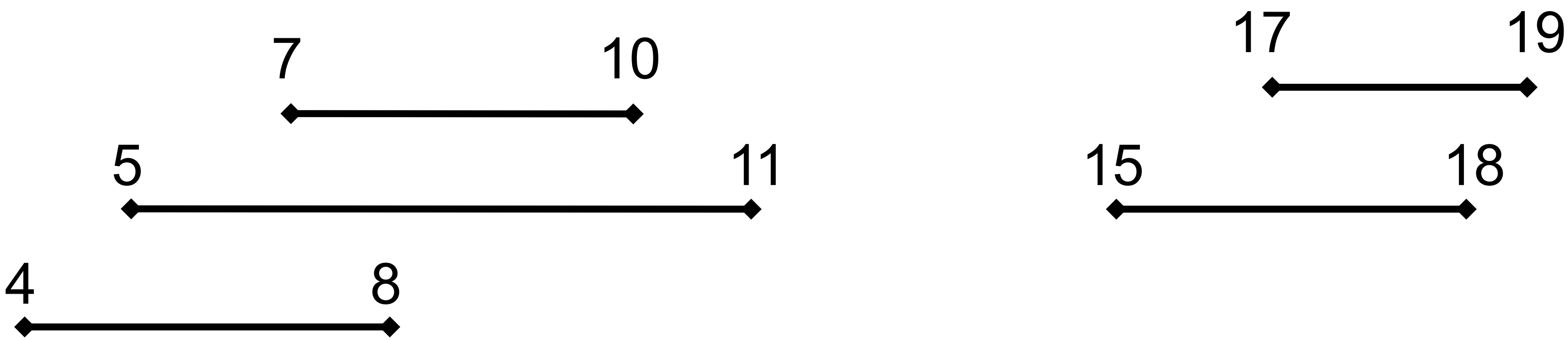
Cell Tower Coverage

Find a tower that covers my location.



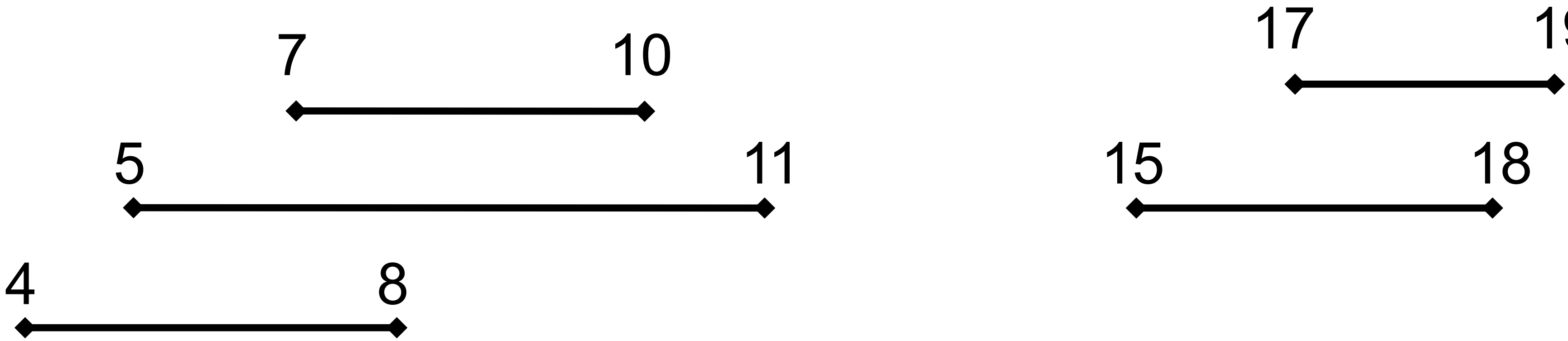
Cell Tower Coverage

Find a tower that covers my location.



Cell Tower Coverage

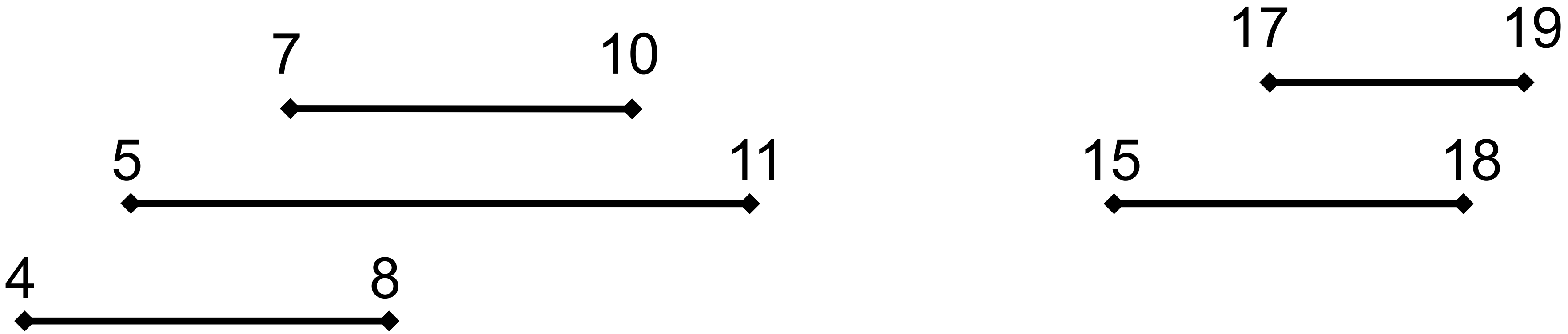
Dynamic data structure: supports new towers.



insert(begin, end)
delete(begin, end)

Cell Tower Coverage

Find a tower that covers my location.

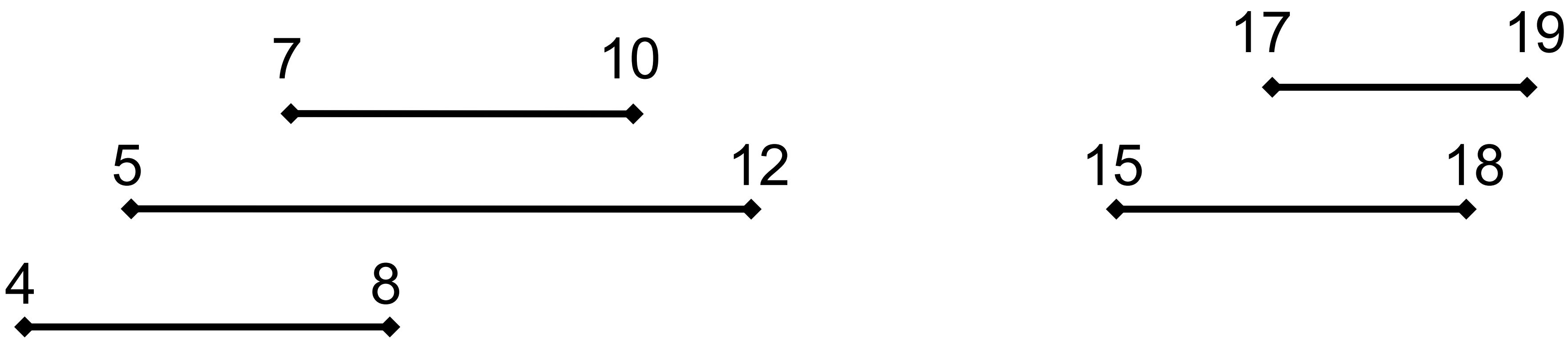


insert(begin, end)
delete(begin, end)

query(x): find an interval that overlaps x.

Cell Tower Coverage

Find a tower that covers my location.



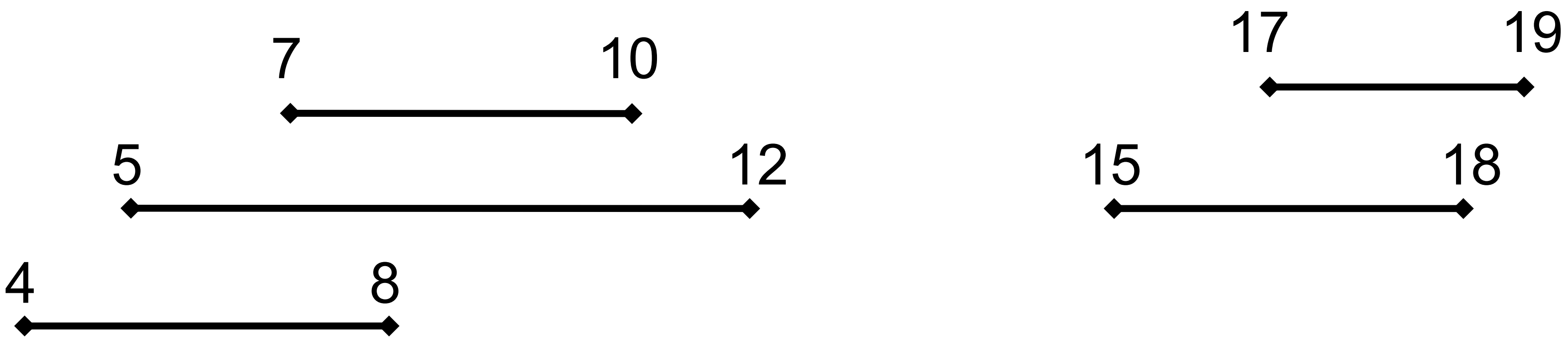
Idea 1: Keep intervals in a list.
Sort by minimum value in interval.
Query: scan entire list.

Does sorting help? Can we binary search?

Cell Tower Coverage

Find a tower that covers my location.

example: query(11)

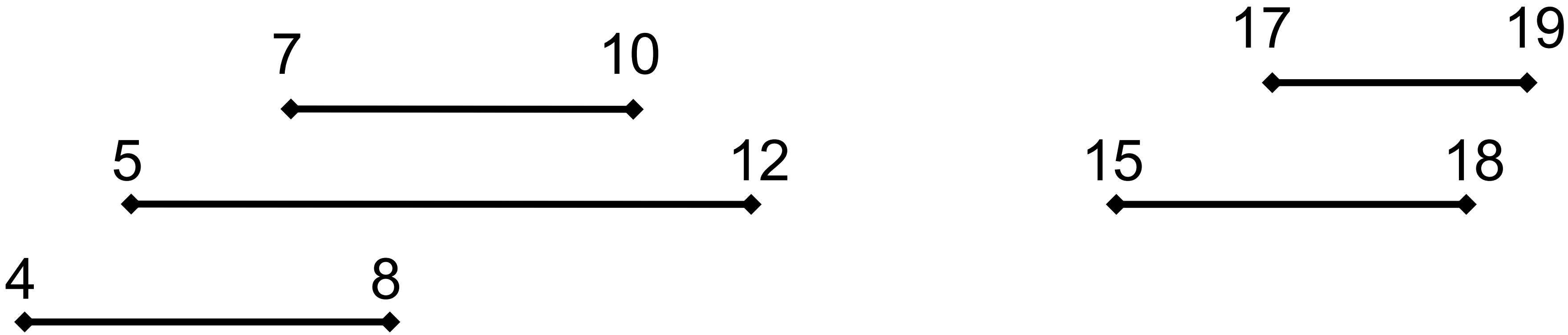


Idea 1: Keep intervals in a list.
Sort by minimum value in interval.
Query: scan entire list.

Does sorting help? Can we binary search?

Cell Tower Coverage

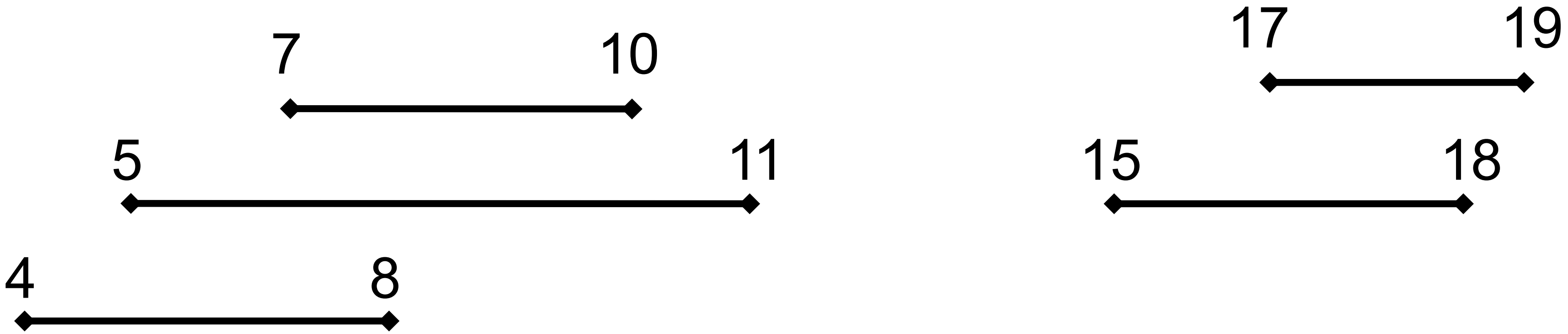
Find a tower that covers my location.



Idea 2: O(1) queries??

Cell Tower Coverage

Find a tower that covers my location.



Idea 2: O(1) queries

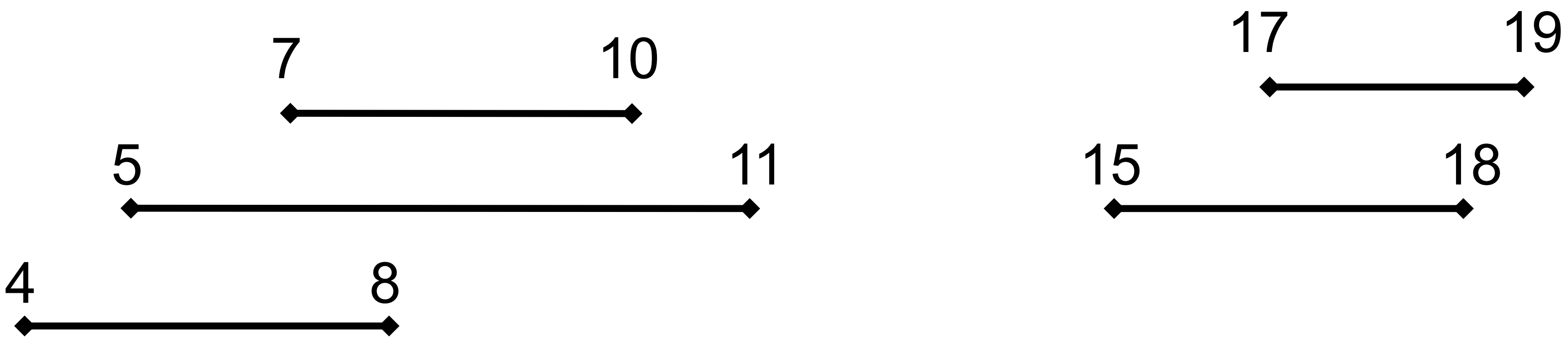
ARCHIPELAGO is open

			A	A	A	A	A	B	B	C				D	D	D	D	E	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Problems??

Cell Tower Coverage

Find a tower that covers my location.



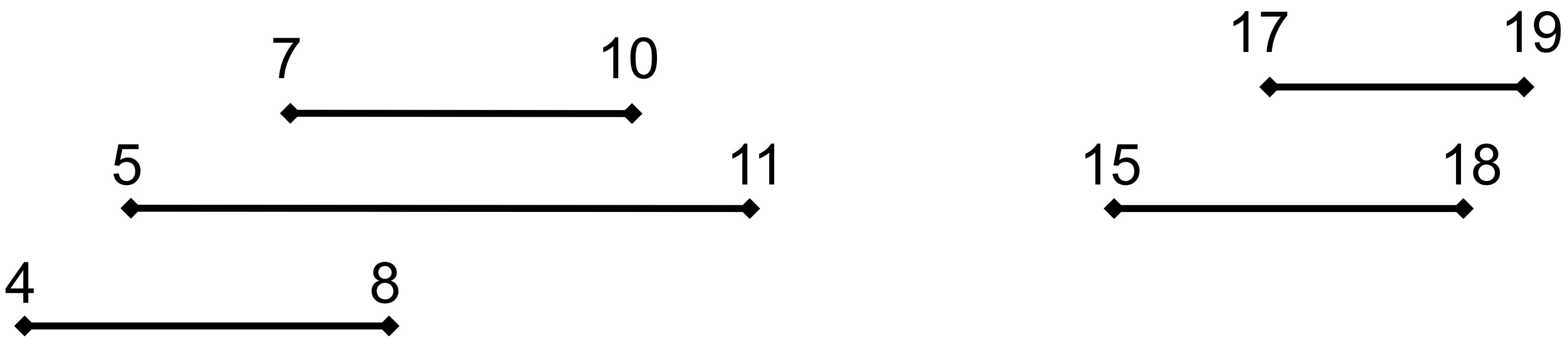
Idea 2: O(1) queries

			A	**A**	**A**	**A**	**A**	**B**	**B**	**C**				**D**	**D**	**D**	**D**	**E**	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Space usage, requires discrete integers, potentially expensive to update.

Cell Tower Coverage

Find a tower that covers my location.

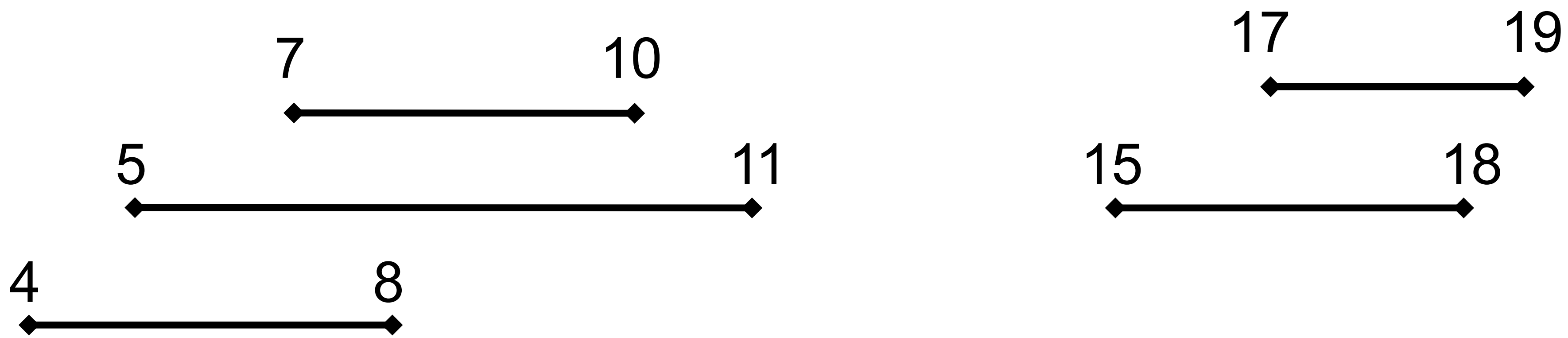


Not ideal solution:

- Space depends on the values stored.
- Time depends on the values stored.

Cell Tower Coverage

Find a tower that covers my location.



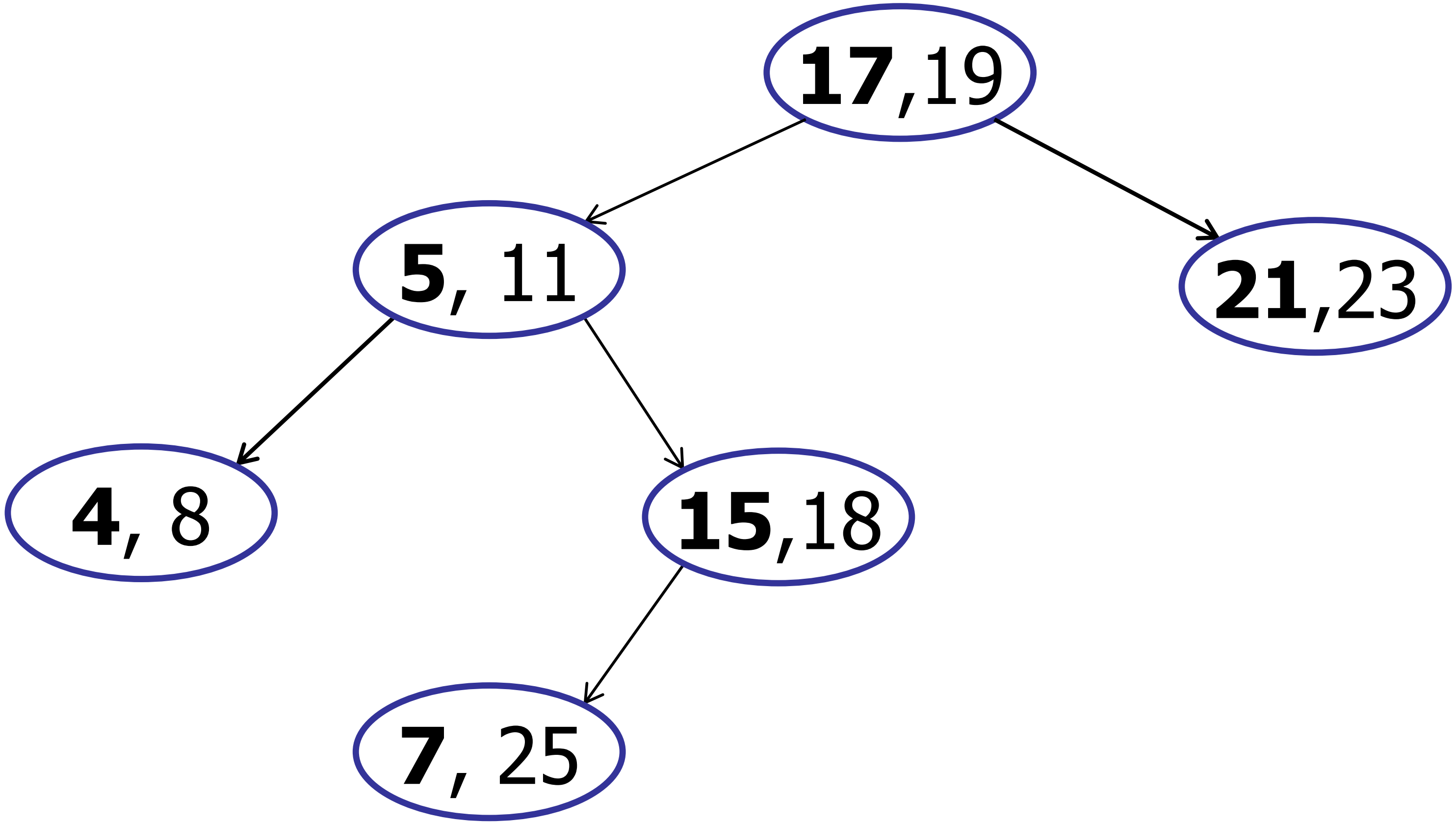
Goal:

- Solutions where space is linear (or near linear) in the number of things stored (i.e., intervals).
- Operations are logarithmic in # of things stored.

Idea 3: Interval Trees

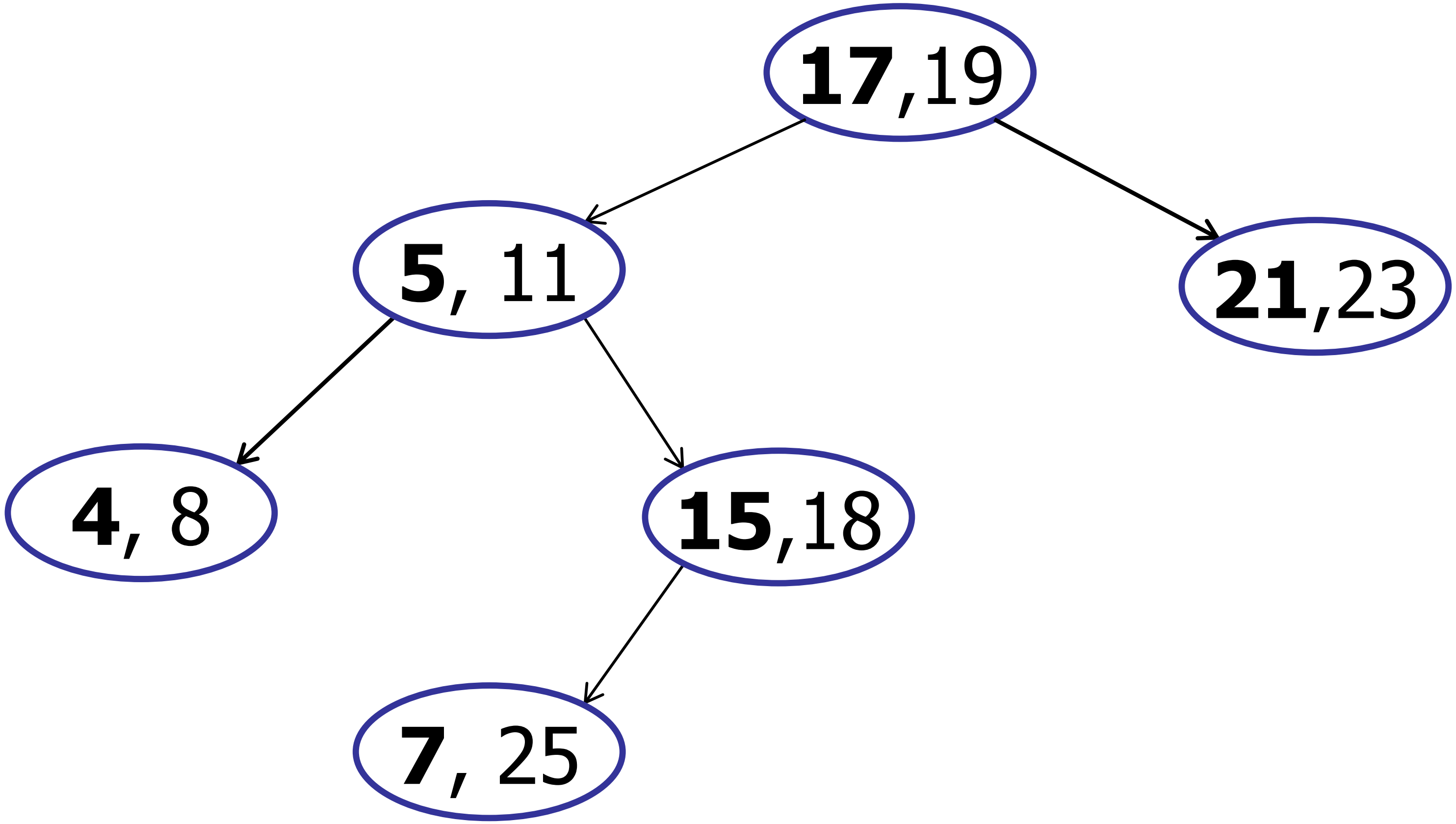
Interval Trees

Each node is an interval



Interval Trees

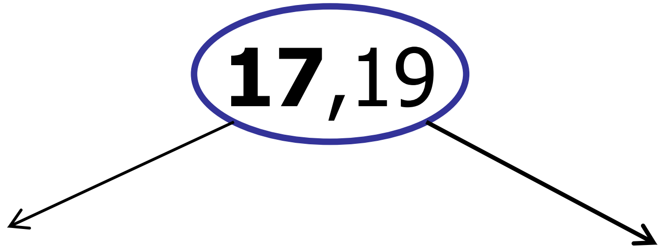
Sorted by left endpoint



Important: always specify what your tree is sorted by!

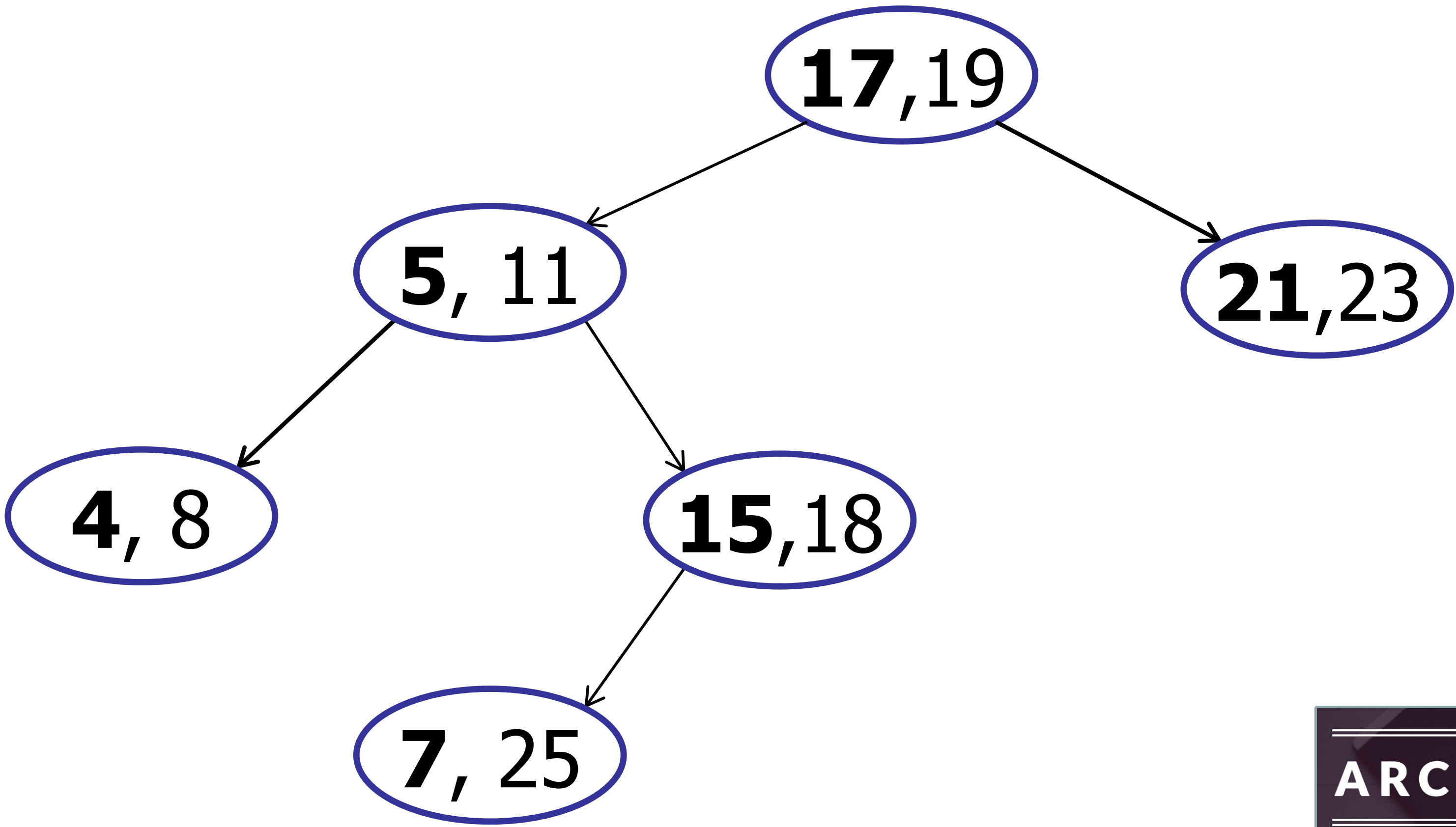
Interval Trees

search-interval(25) = ?



Interval Trees

Augment: ??

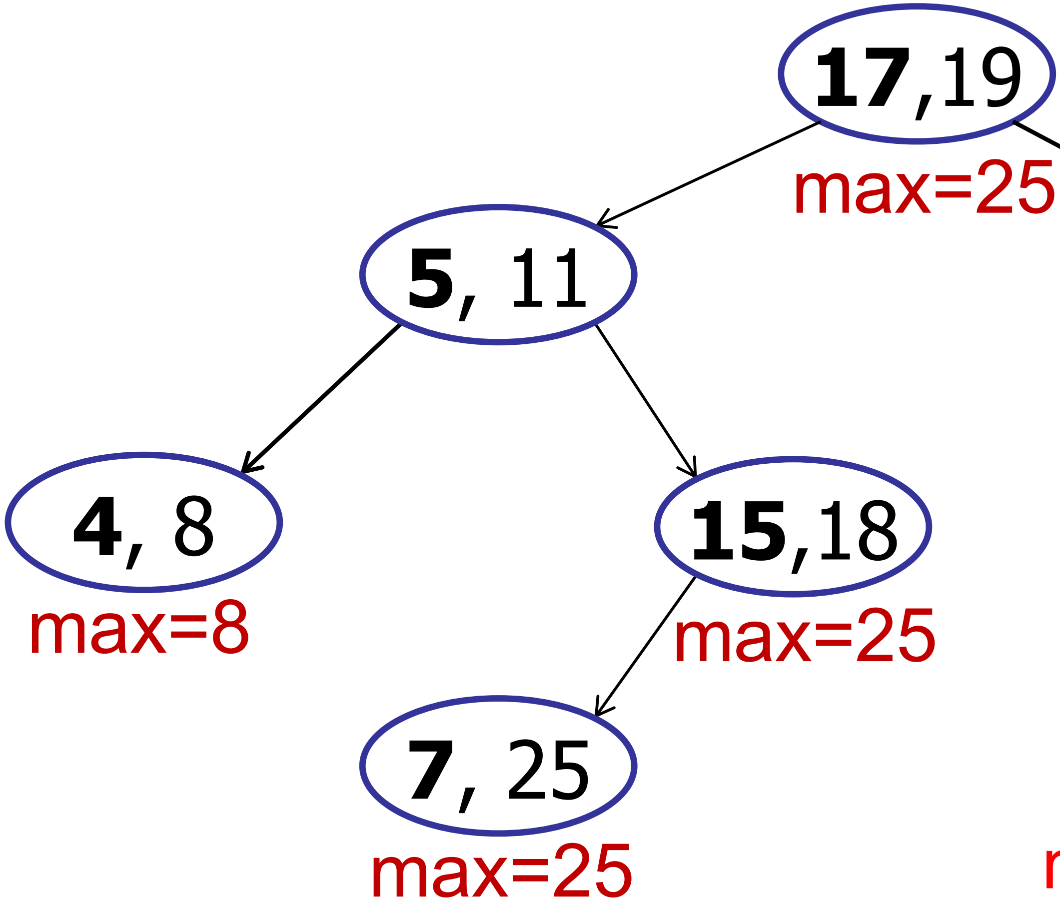


ARCHIPELAGO

is open

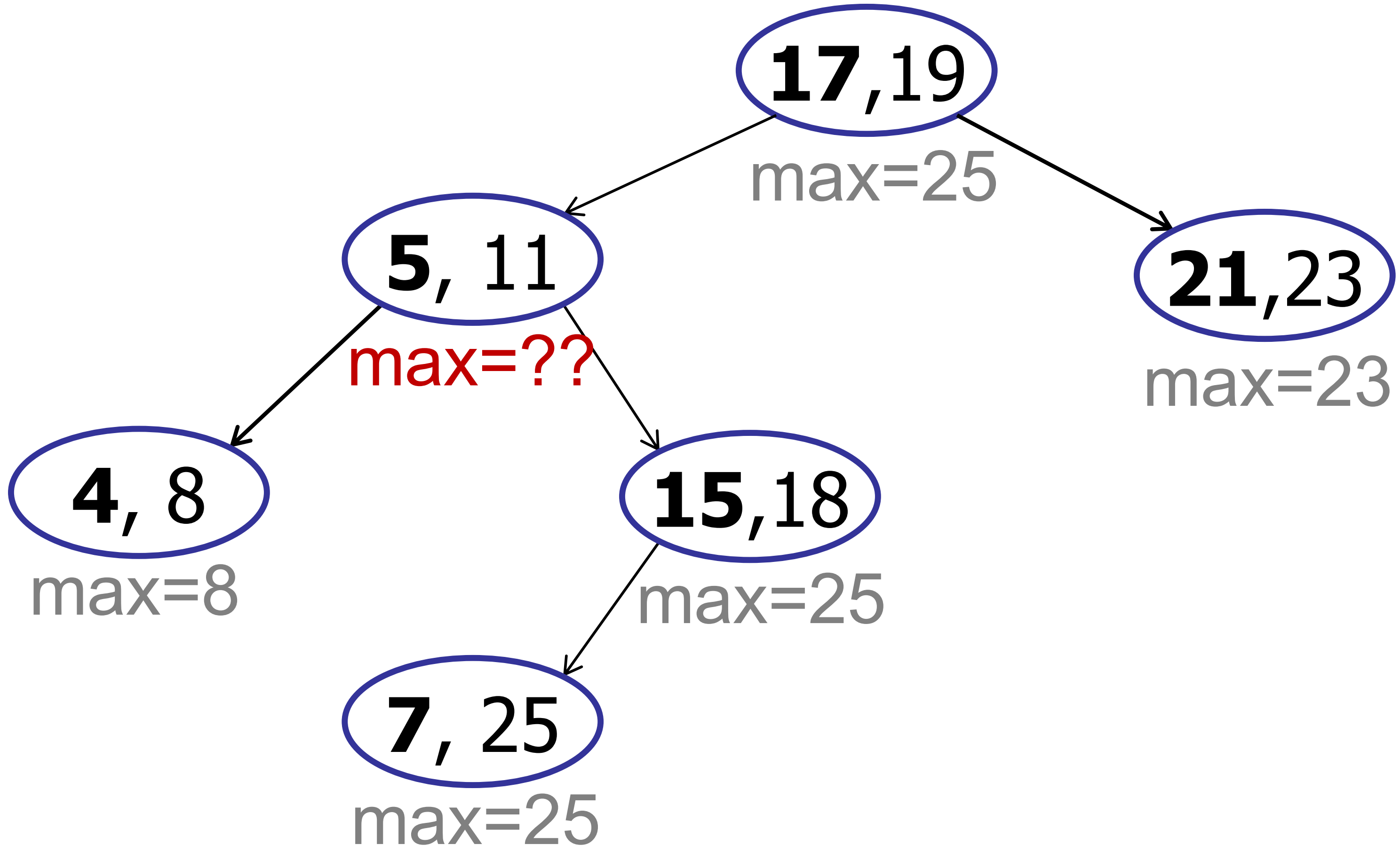
Interval Trees

Augment: maximum endpoint in subtree



$$\text{max} = \begin{cases} \text{node.right} \\ \text{left.max} \\ \text{right.max} \end{cases}$$

max=??



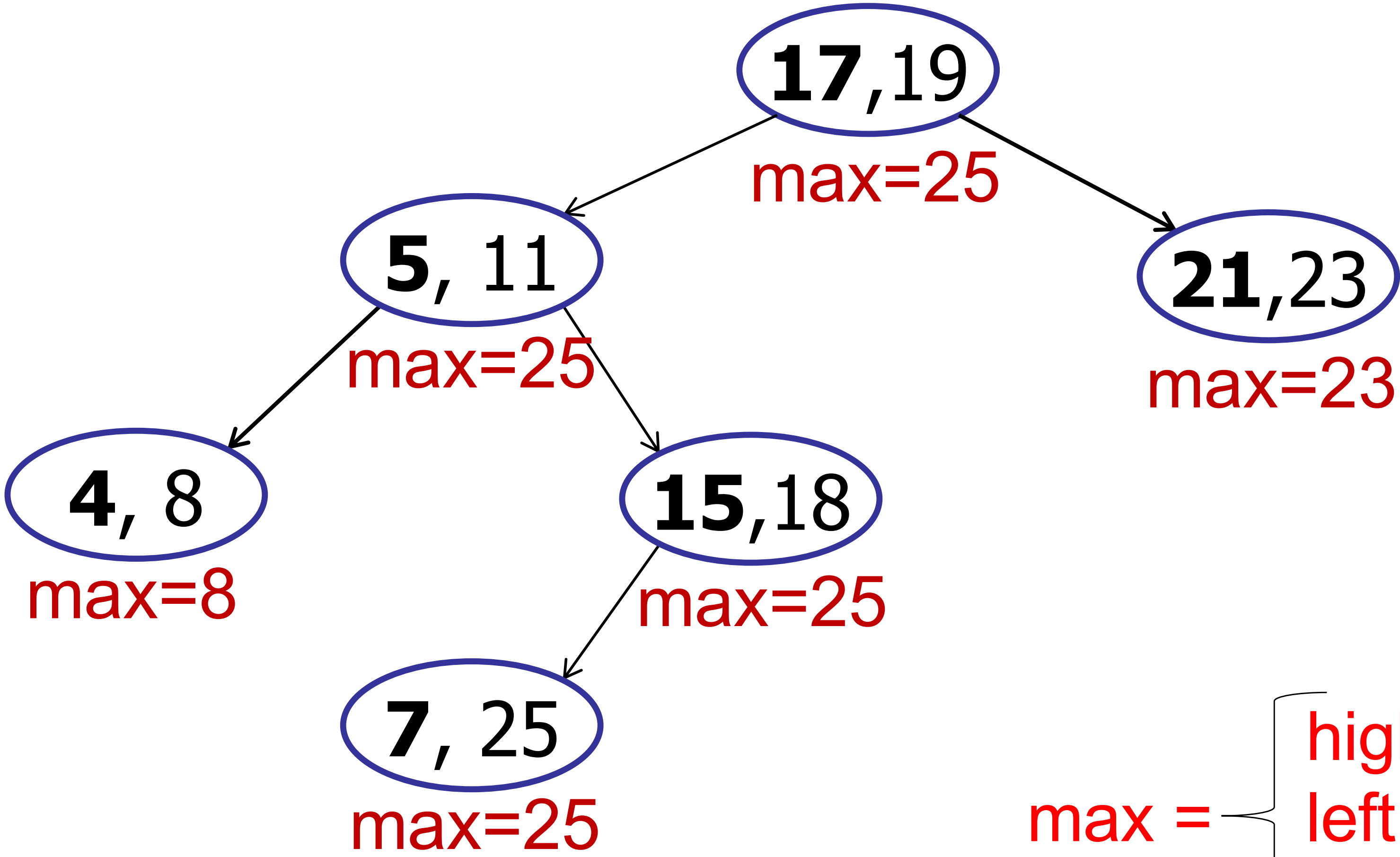
1. 5
2. 8
3. 11
4. 18
5. 25 ✓
6. 19

ARCHIPELAGO

is open

Interval Trees

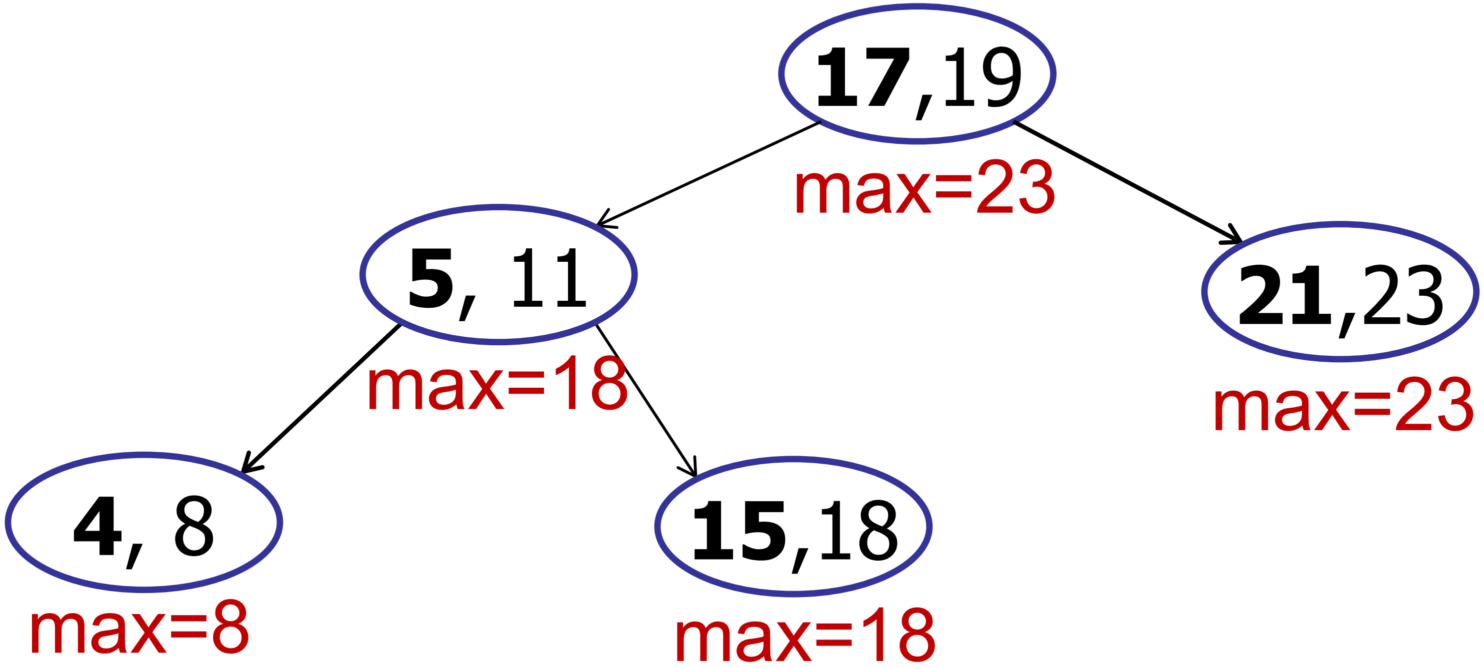
Augment: maximum endpoint in subtree



$$\text{max} = \begin{cases} \text{high} \\ \text{left.max} \\ \text{right.max} \end{cases}$$

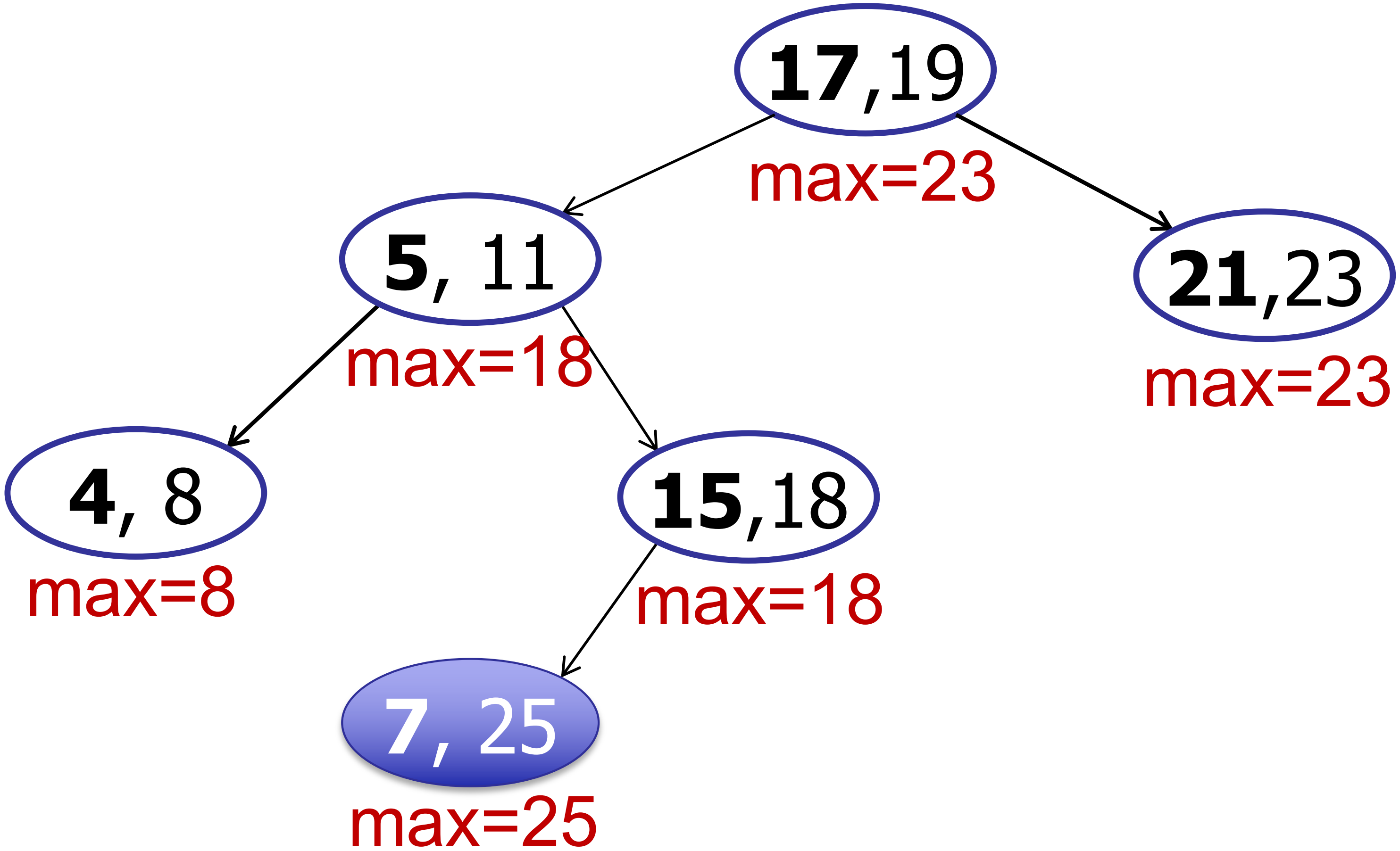
Interval Trees

Insertion: *example* – **insert(7, 25)**



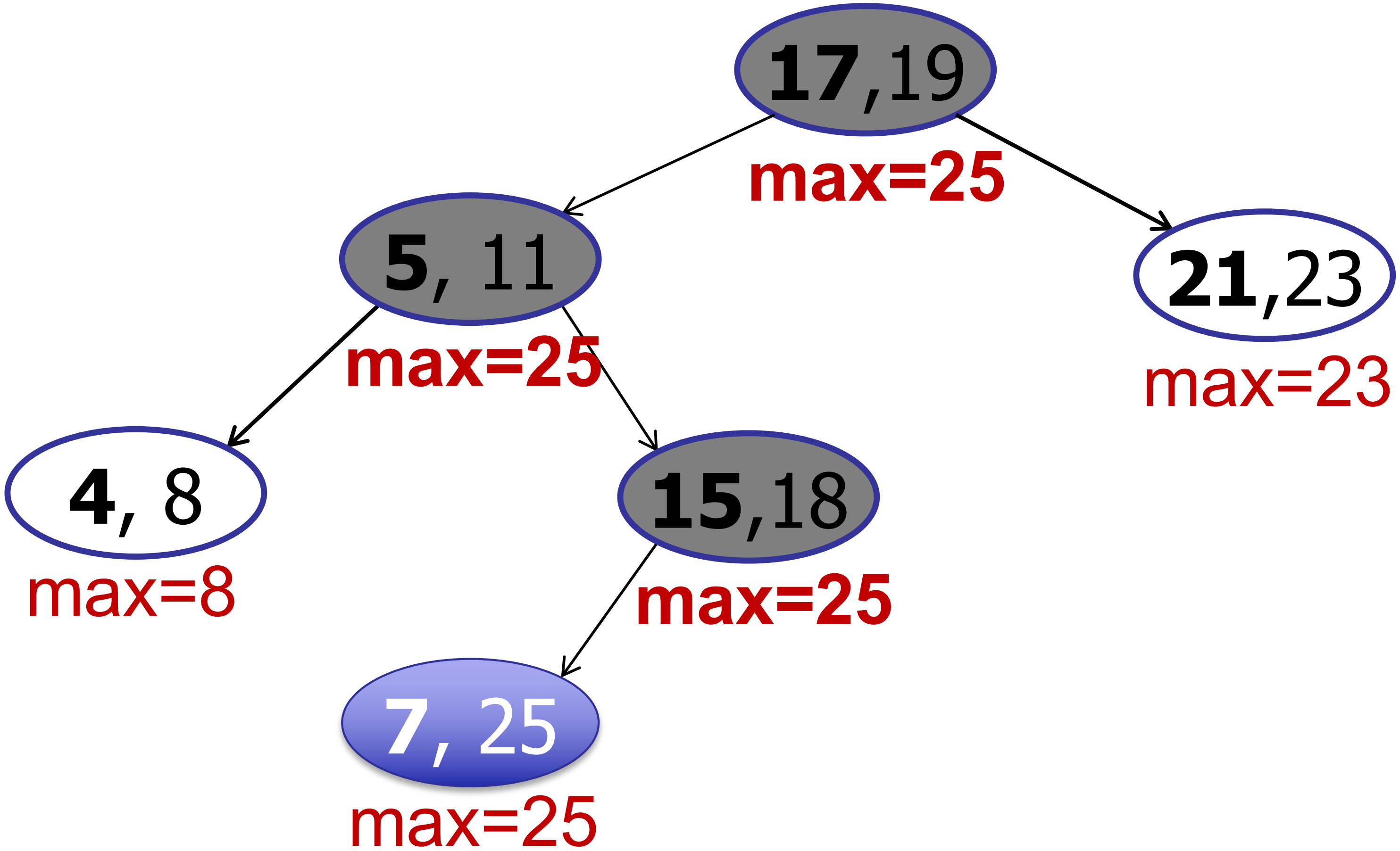
Interval Trees

Insertion: *example* – **insert(7, 25)**



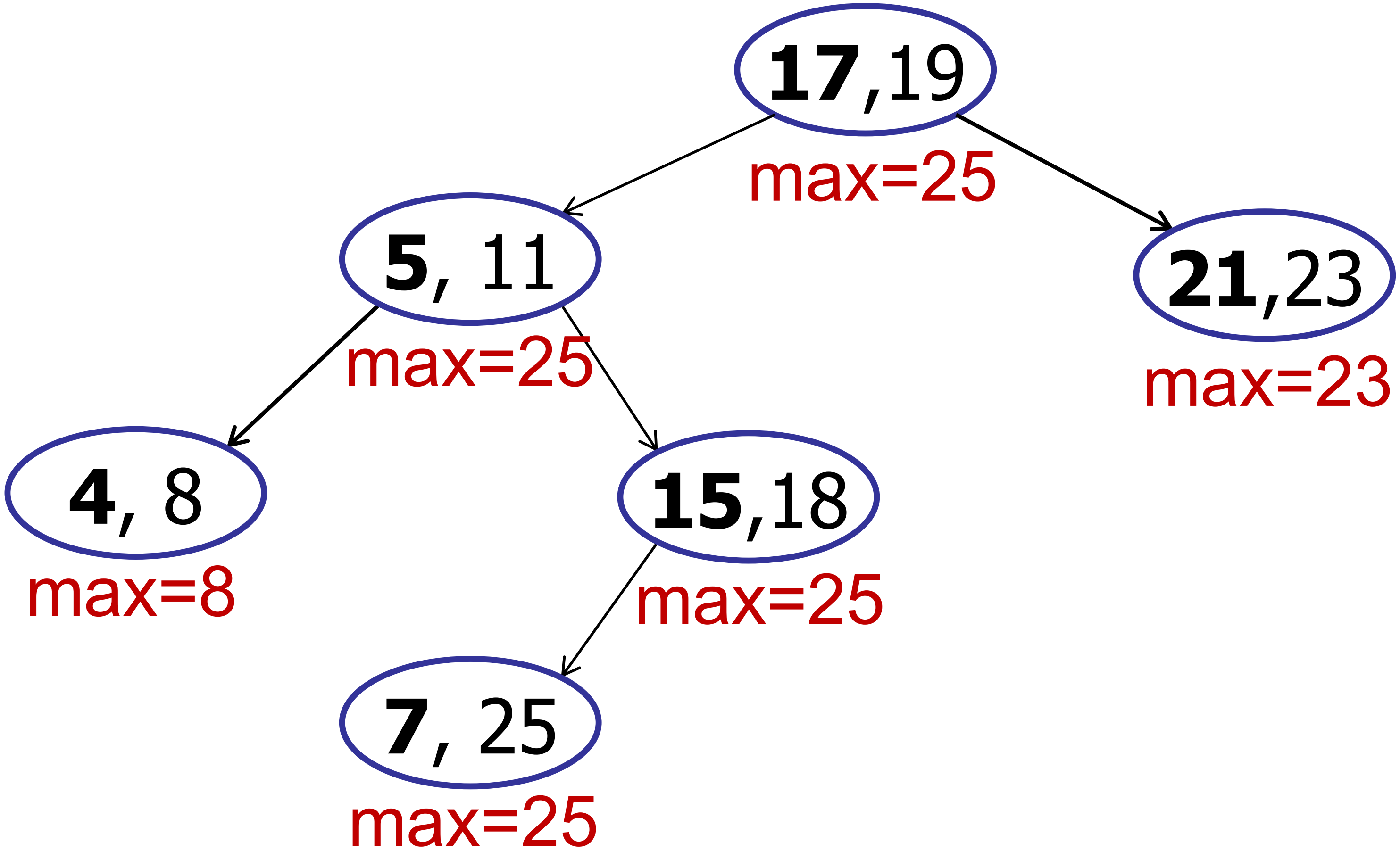
Interval Trees

Insertion: *example* – **insert(7, 25)**



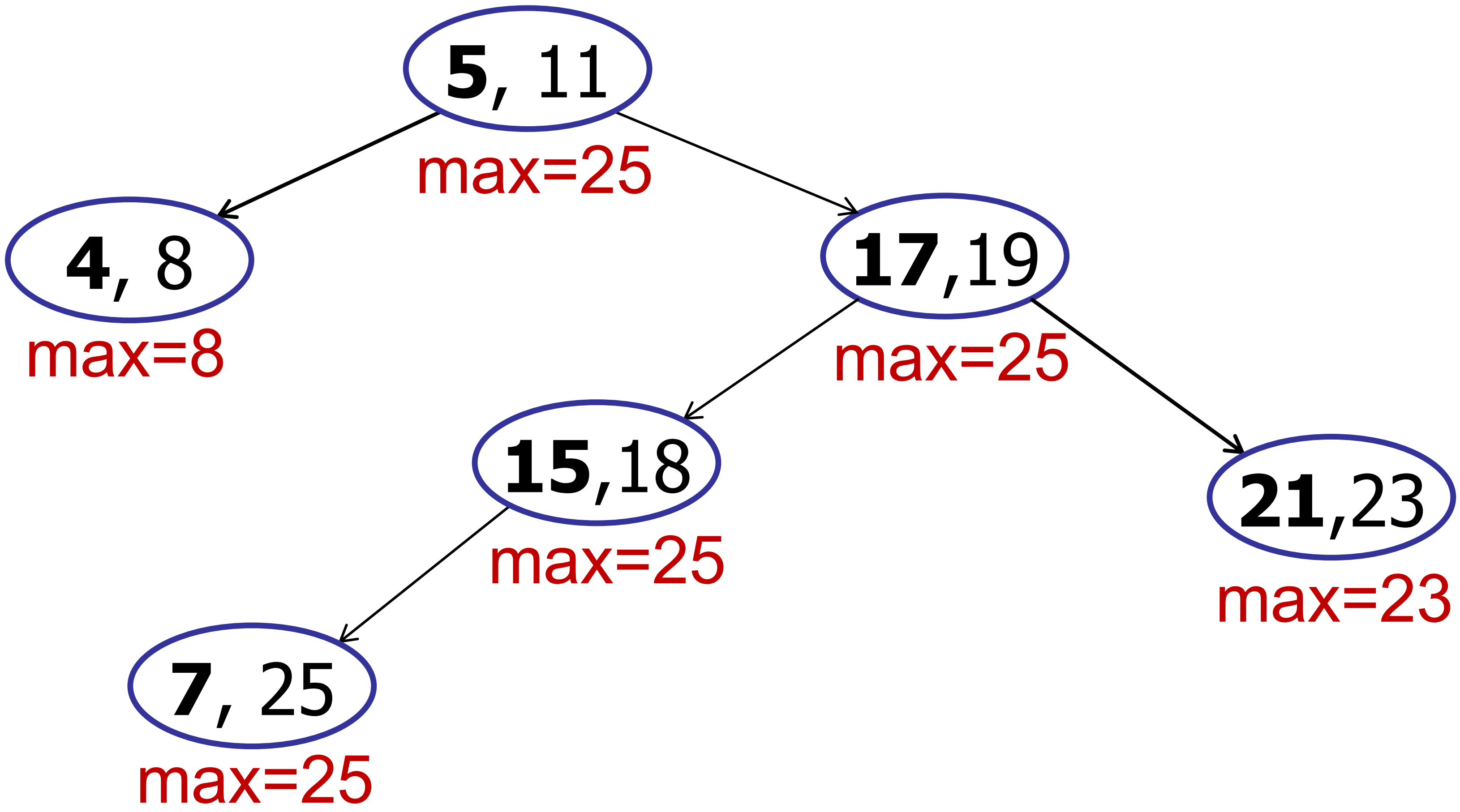
Interval Trees

Insertion: *out-of-balance*



Interval Trees

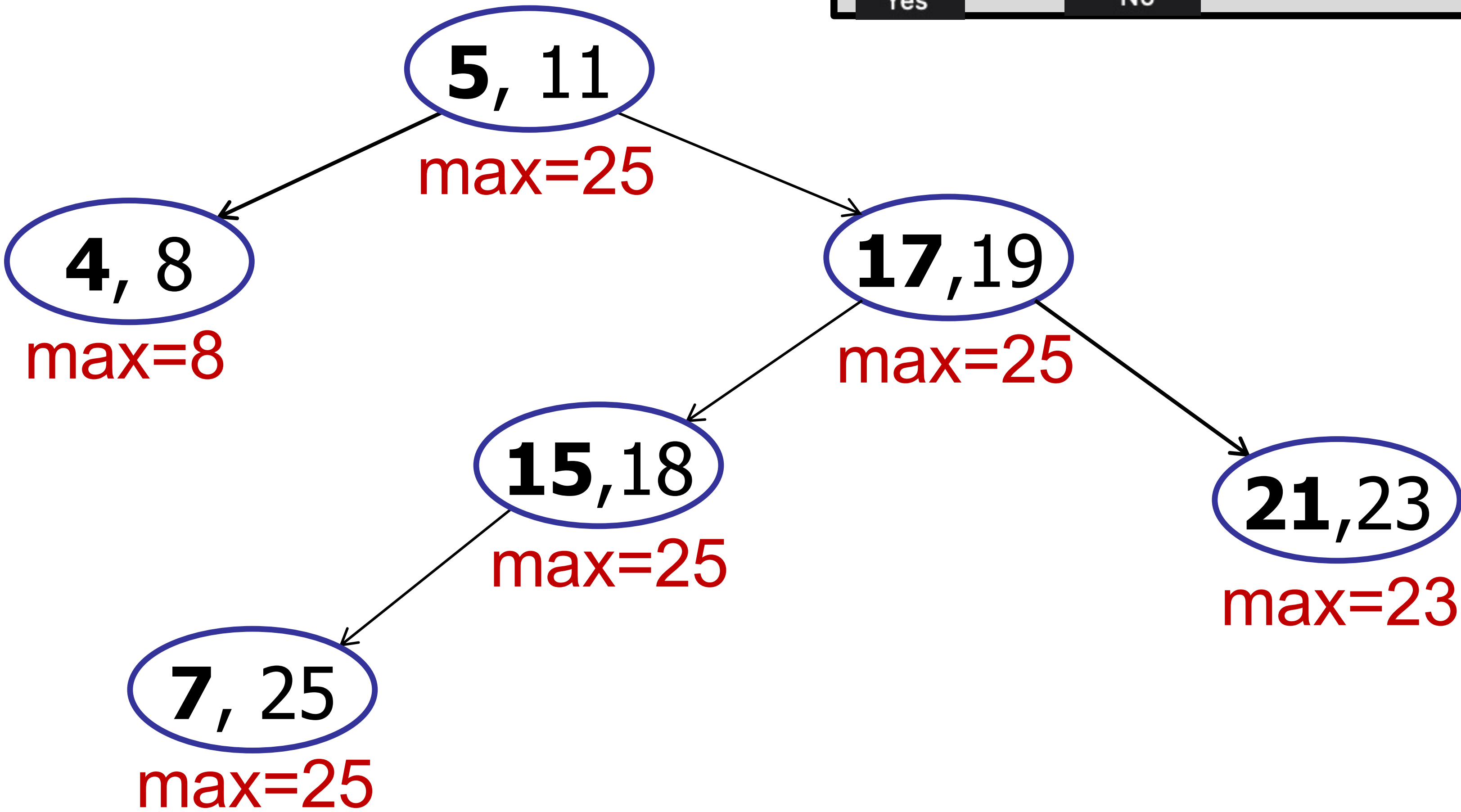
Insertion: right-rotate (17, 19)



Interval Trees

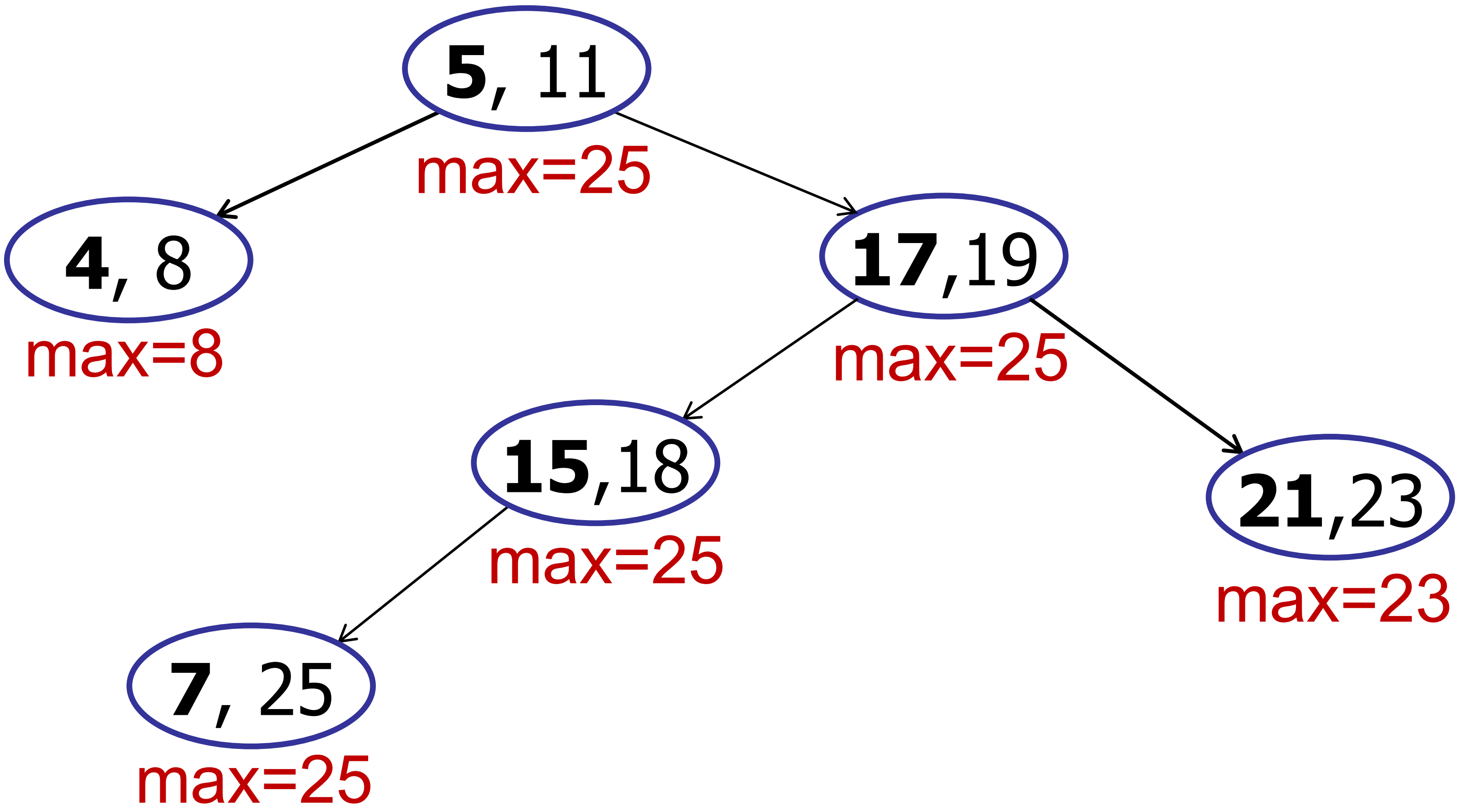
Insertion: right-rotate (17, 19)

Is the tree now balanced?



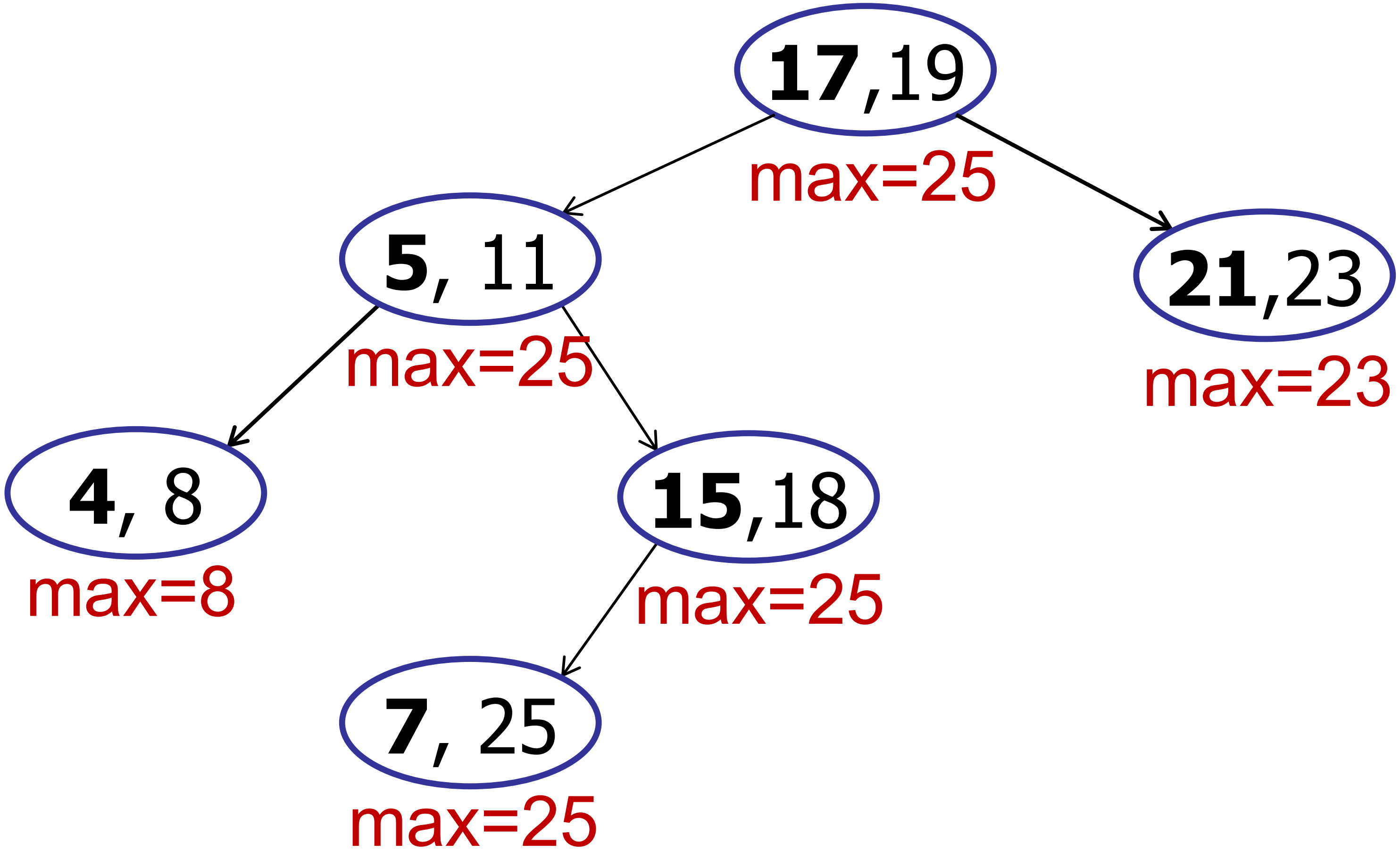
Interval Trees

Insertion: *right-rotate (17, 19)*, OOPS!



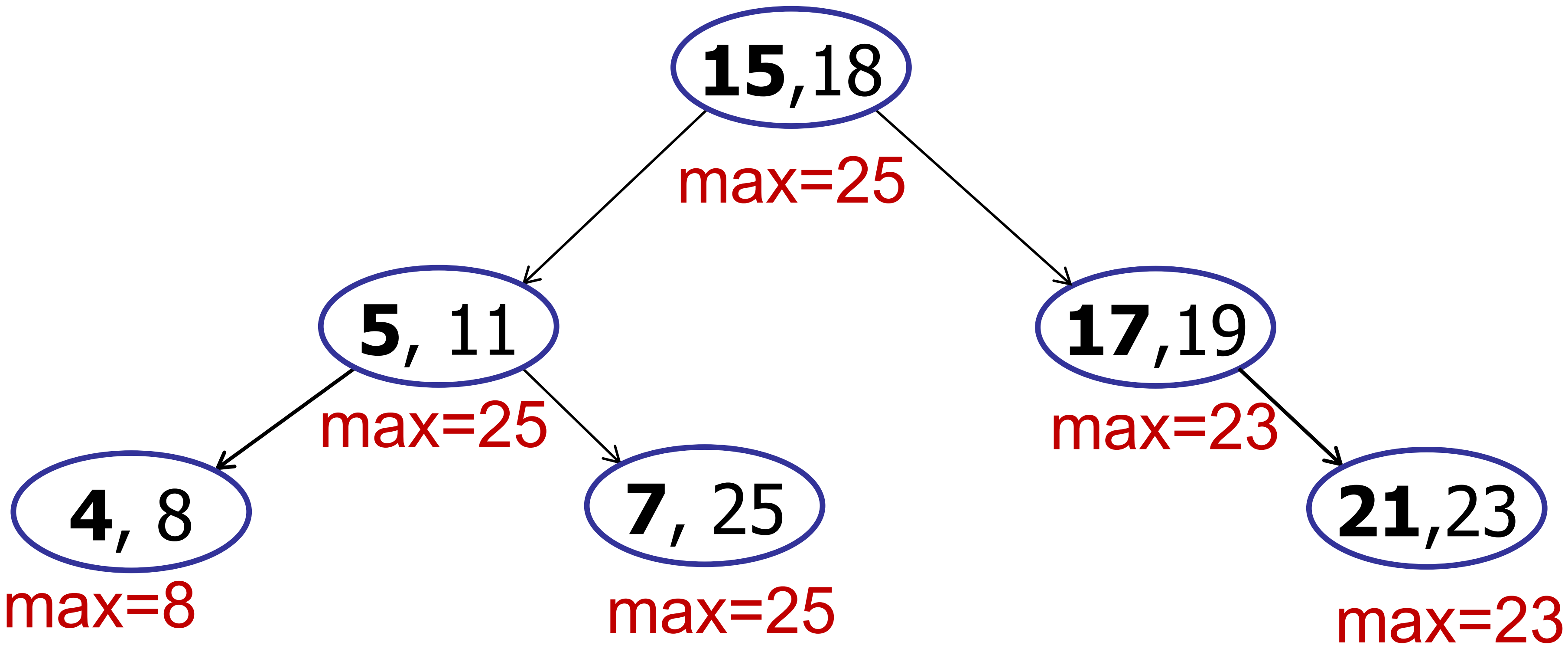
Interval Trees

Insertion: *out-of-balance*



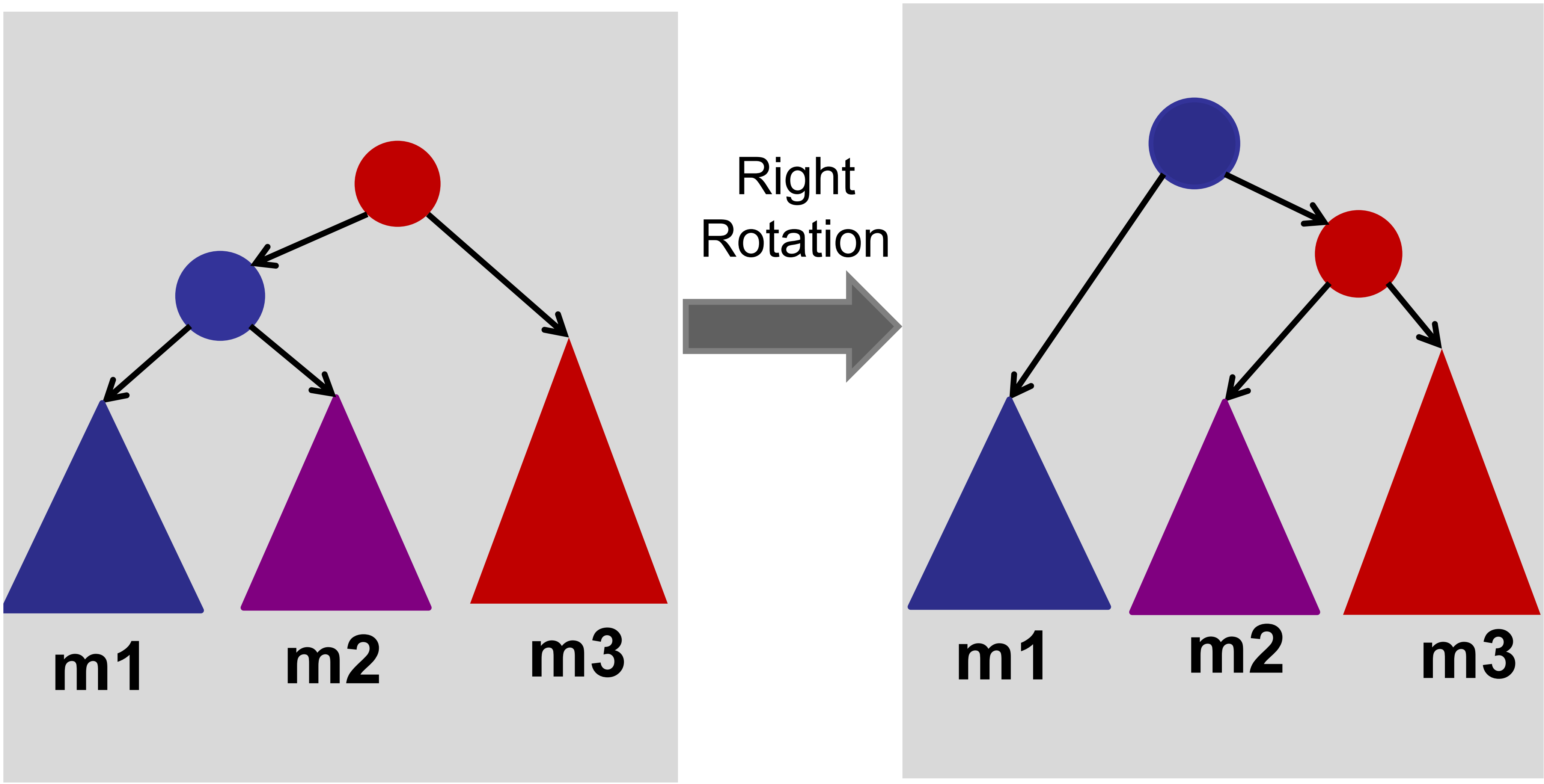
Interval Trees

Insertion: left-rotate, right-rotate



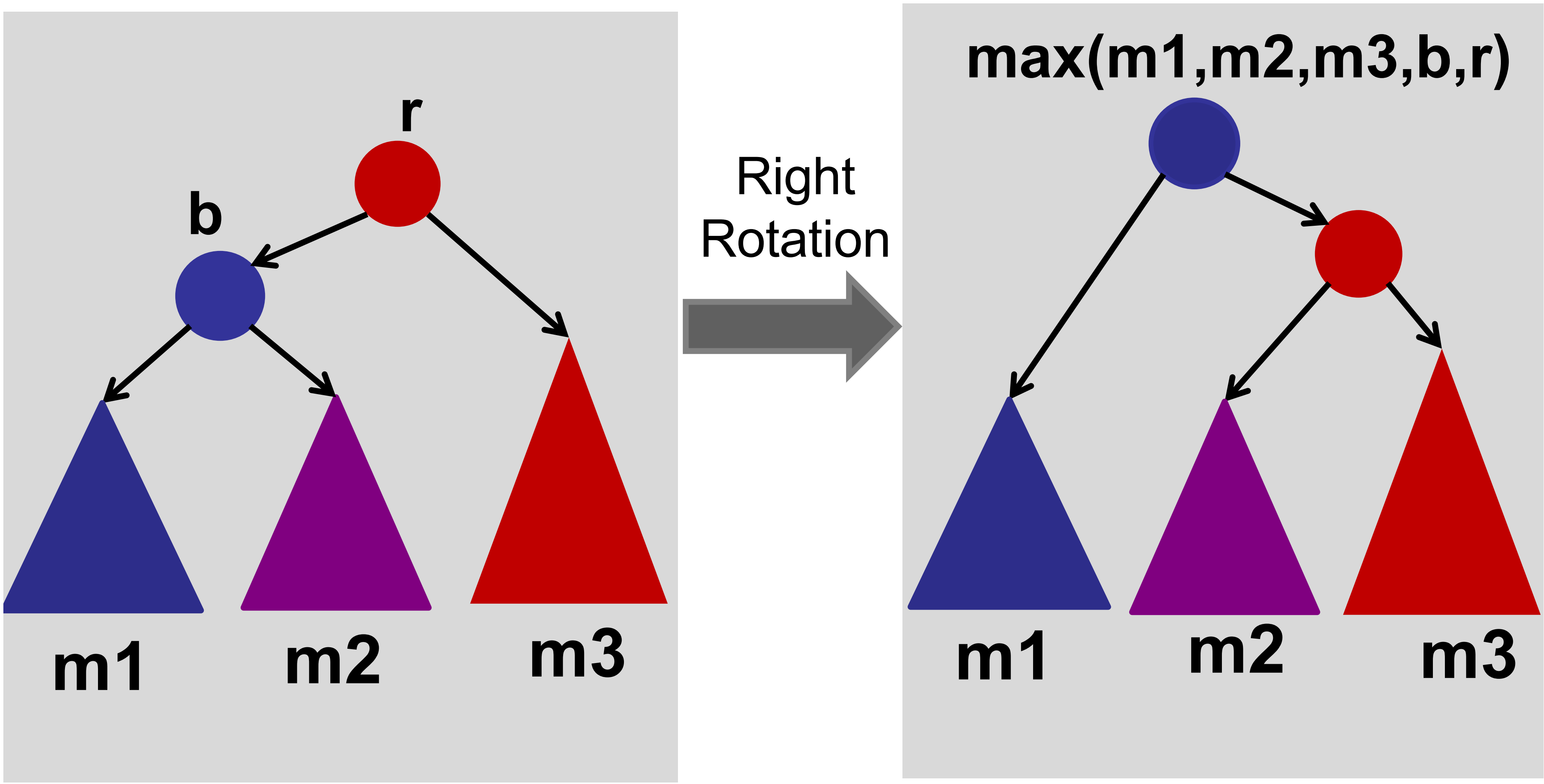
Interval Trees

Maintain MAX during rotations:



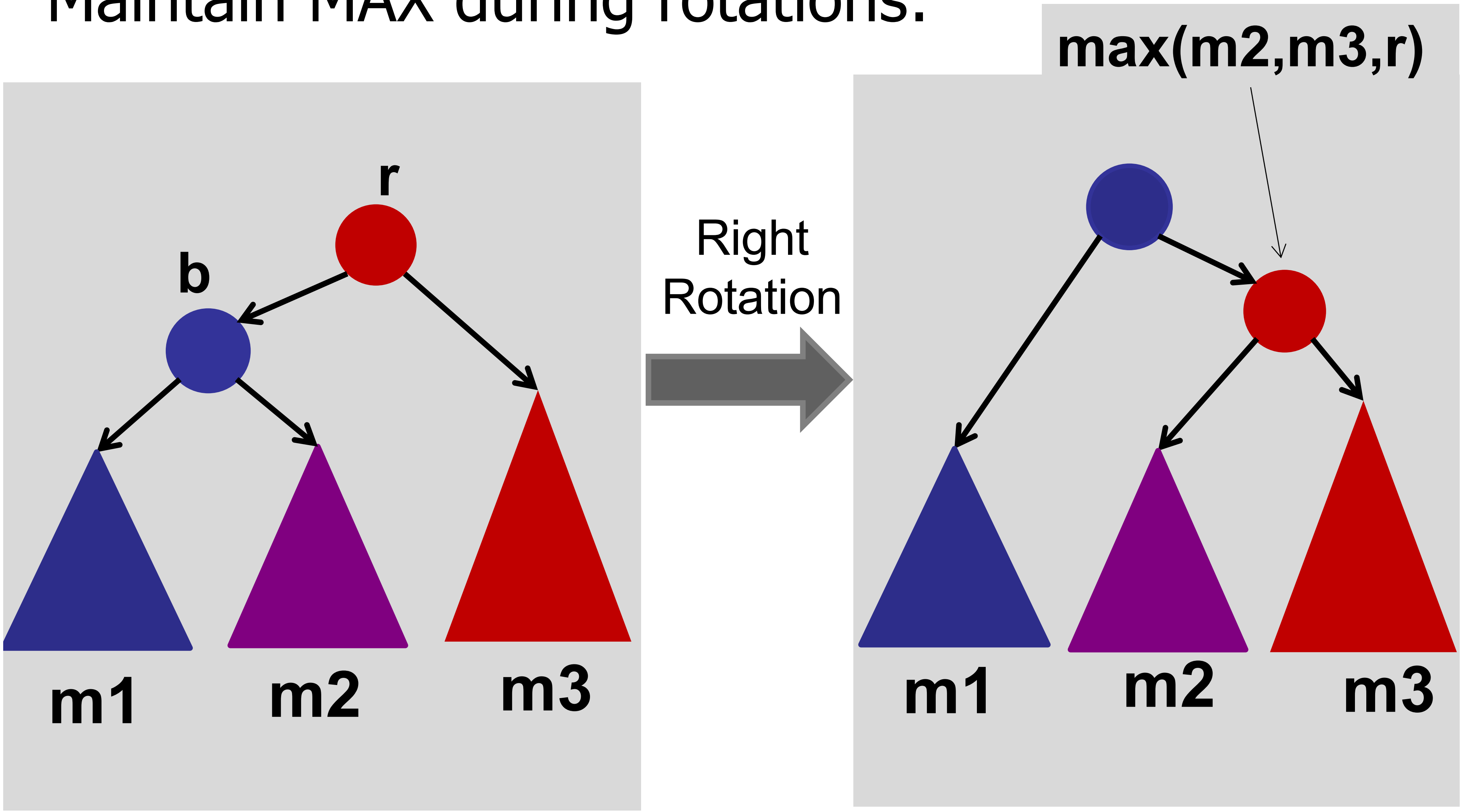
Interval Trees

Maintain MAX during rotations:



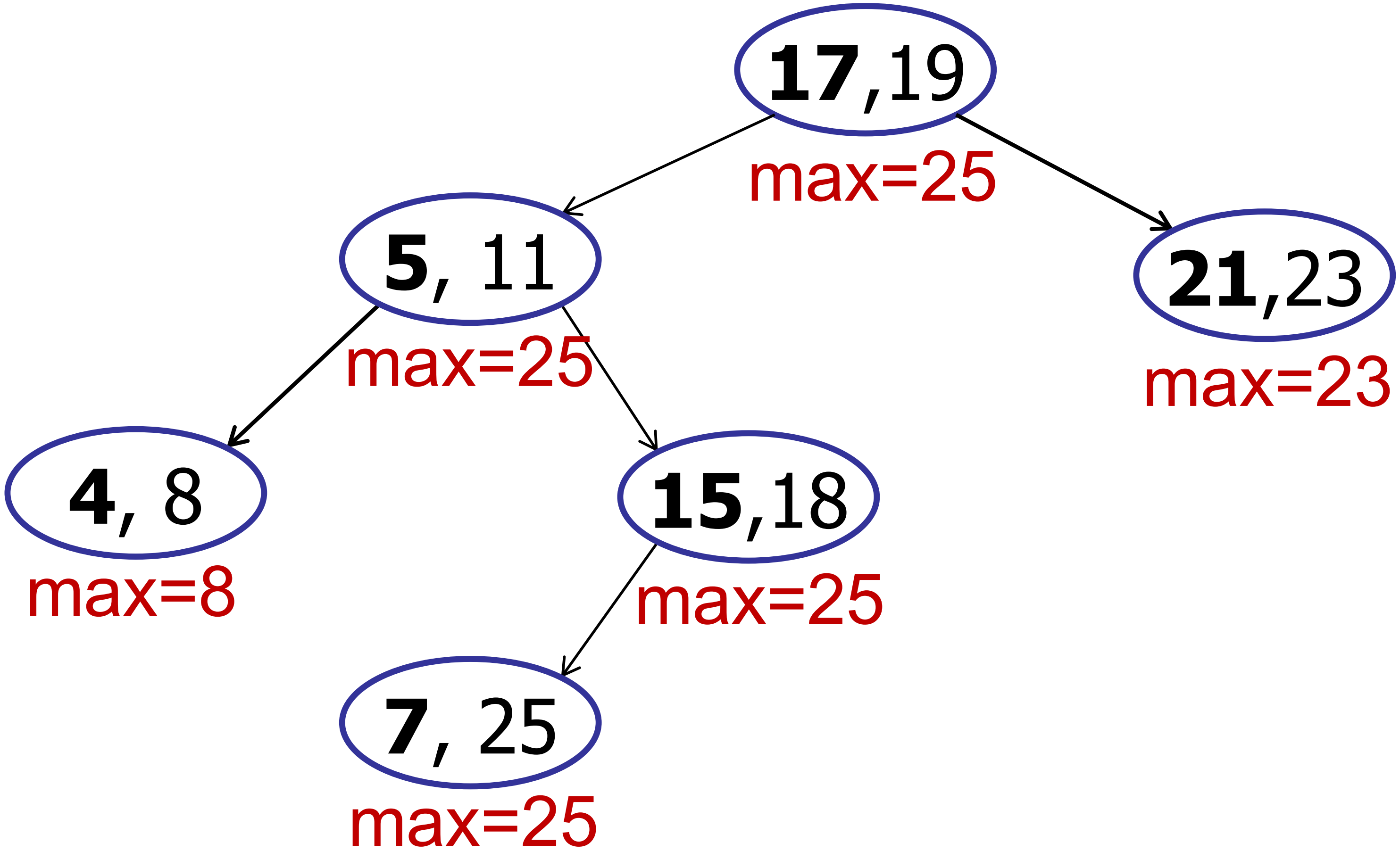
Interval Trees

Maintain MAX during rotations:



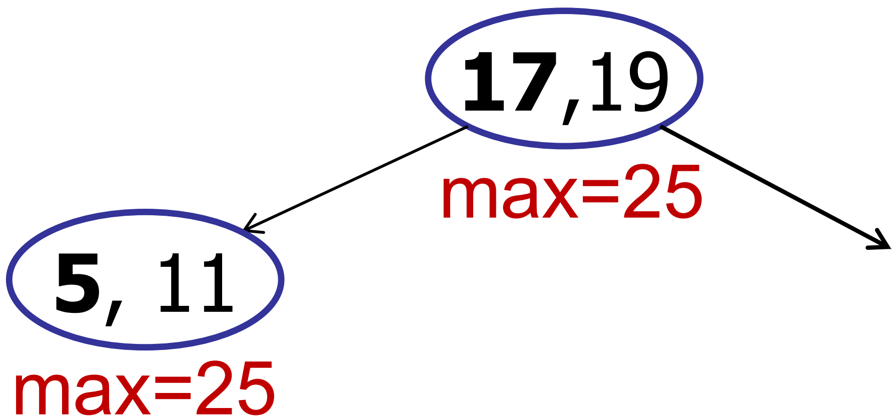
Interval Trees

Searching: interval-search(22)



Interval Trees

Searching: interval-search(22)



It is possible that 22 is covered in the left subtree.

Do we know *for sure* that going left will work?

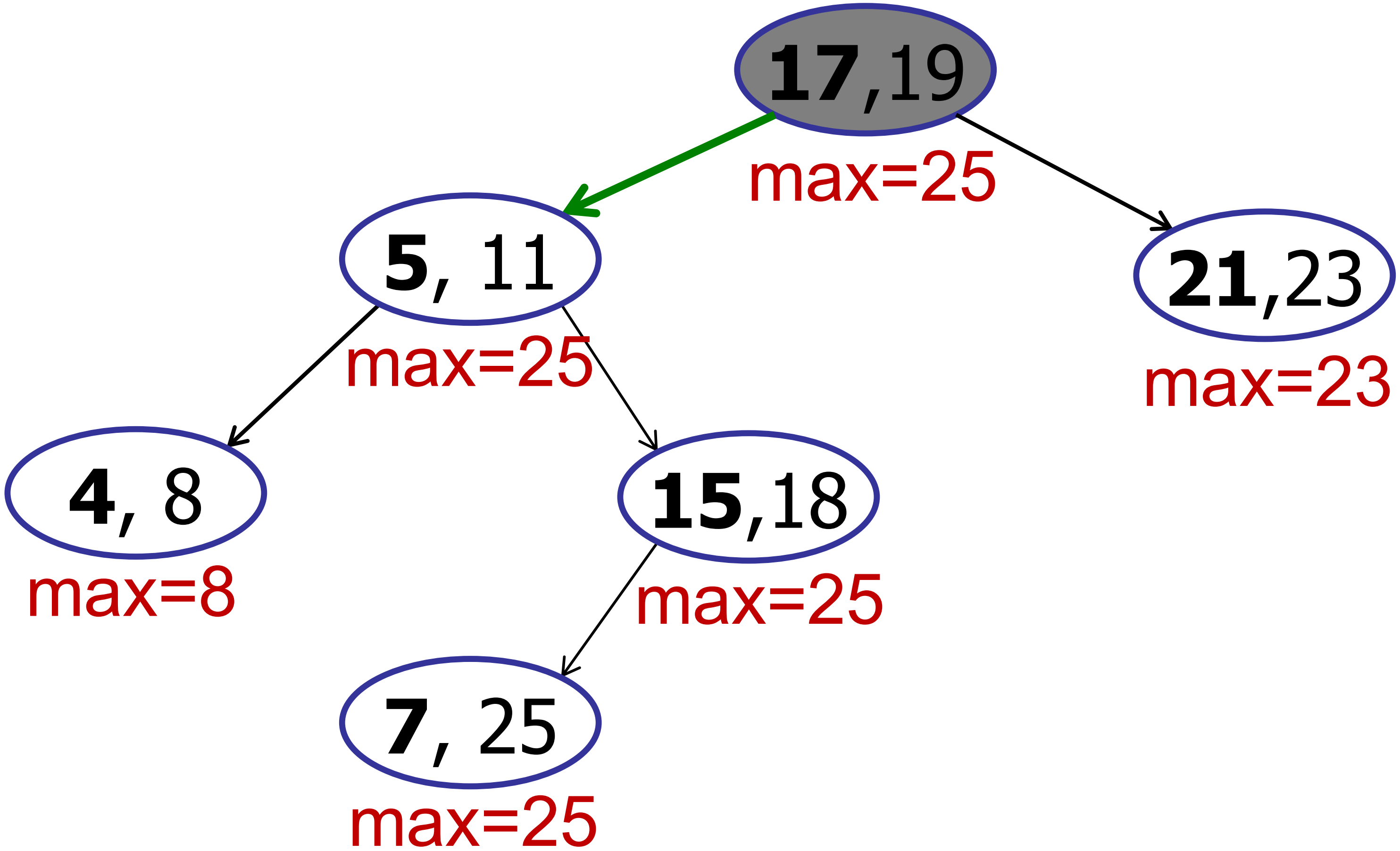
Interval Trees

interval-search(x) : find interval containing x

```
interval-search(x)
    c = root;
    while (c != null and x is not in c.interval) do
        if (c.left == null) then
            c = c.right;
        else if  (x > c.left.max)  then
            c = c.right;
        else c = c.left;
    return c.interval;
```

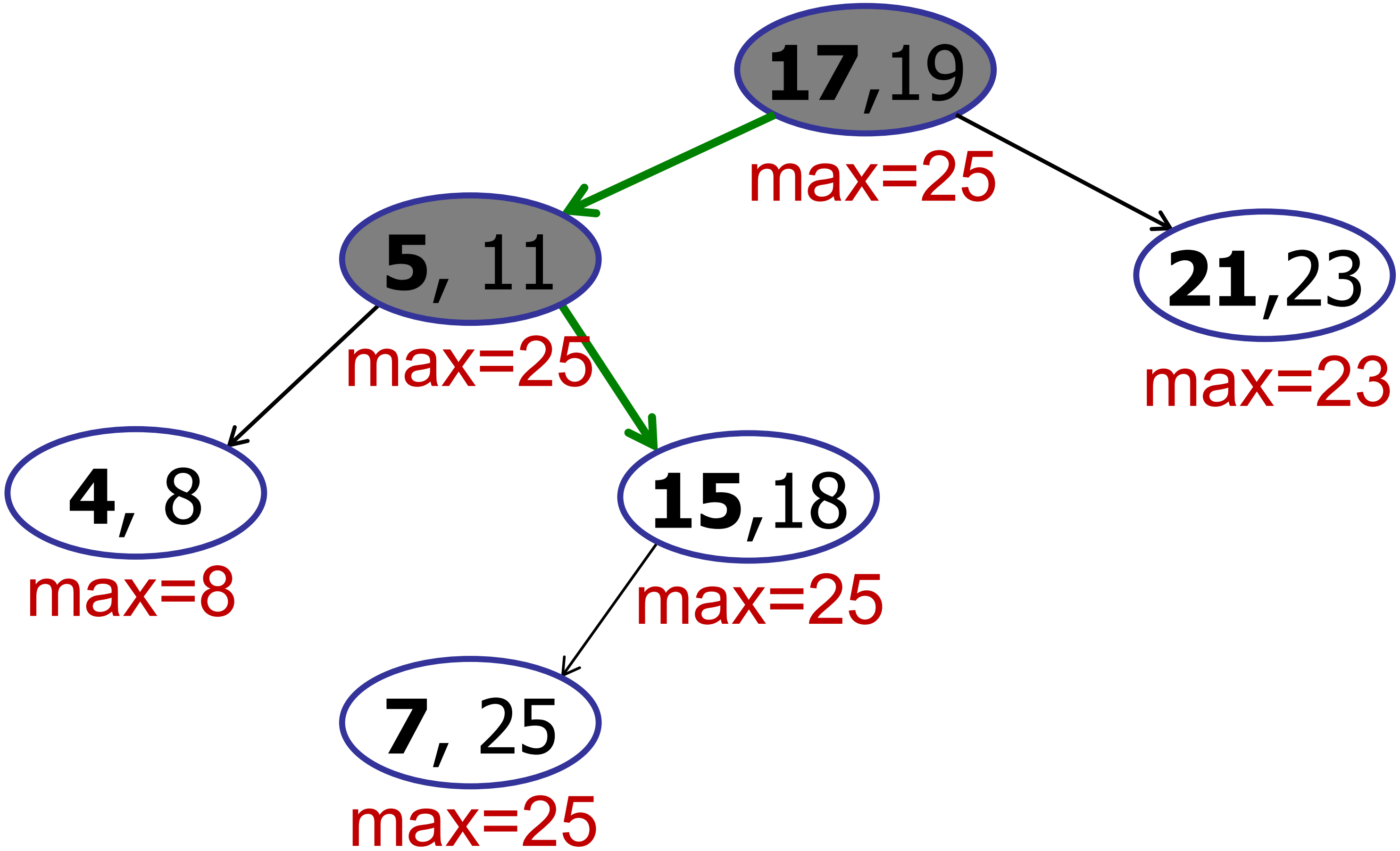
Interval Trees

Searching: interval-search(22)



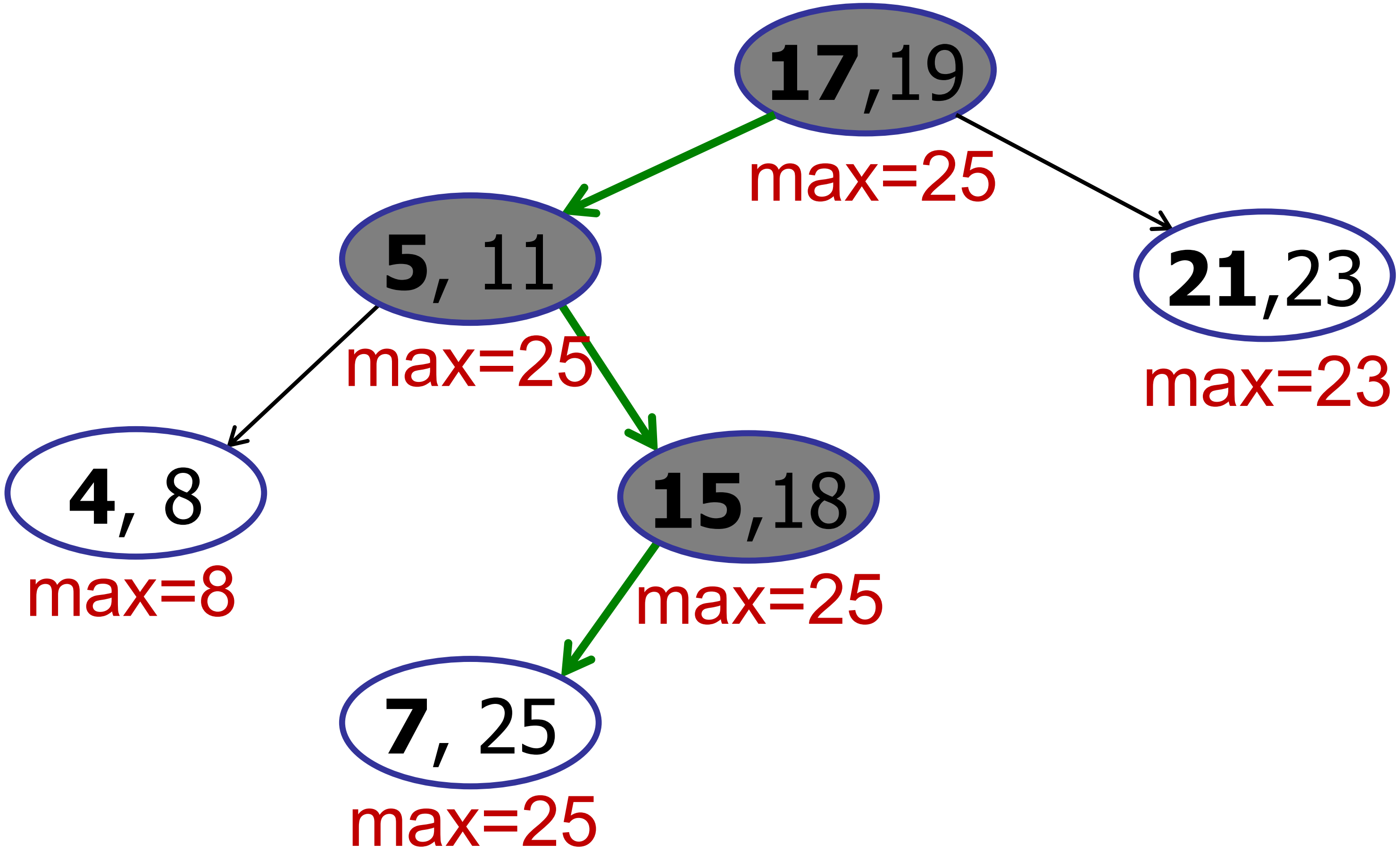
Interval Trees

Searching: interval-search(22)



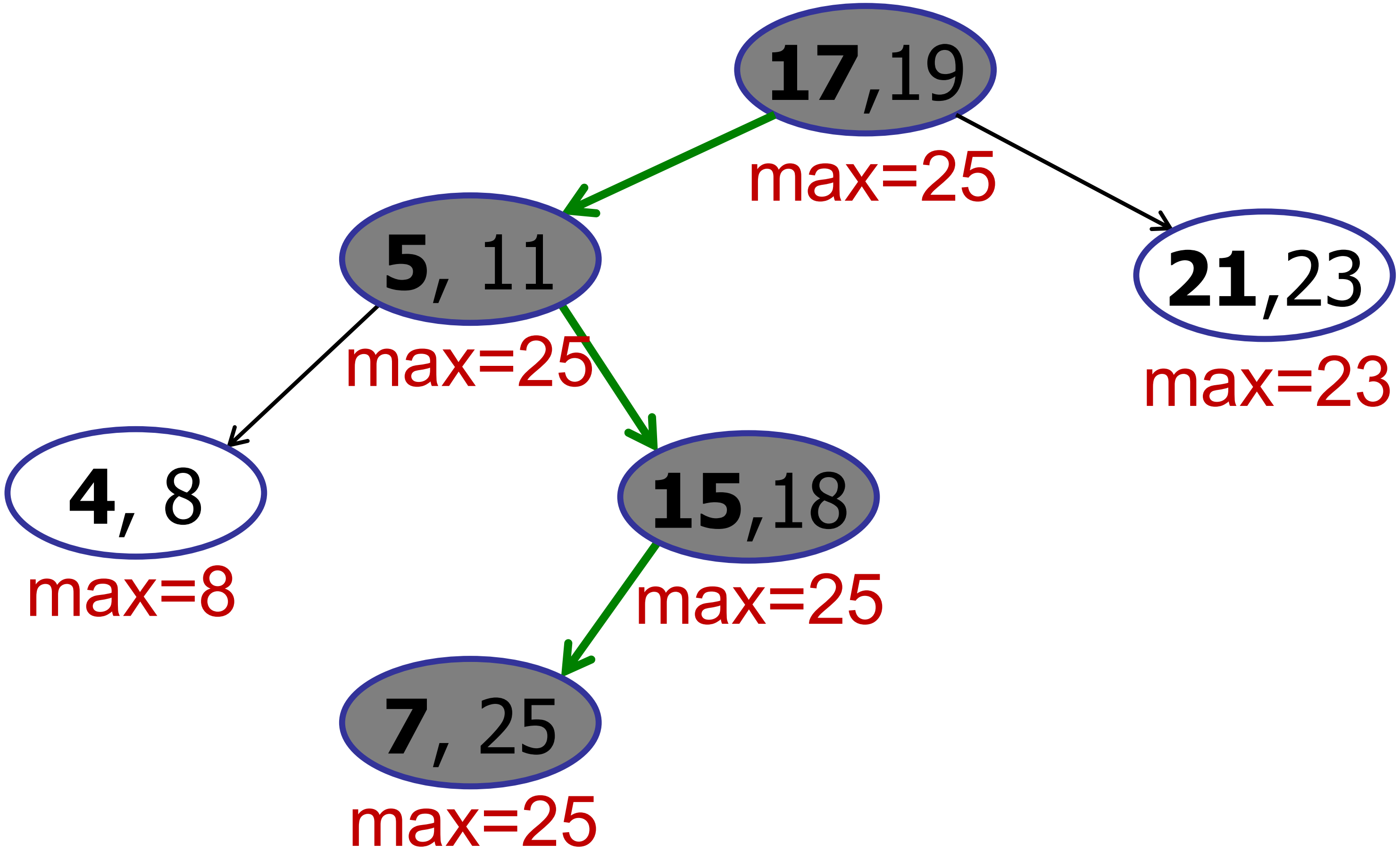
Interval Trees

Searching: interval-search(22)



Interval Trees

Searching: interval-search(22)



Interval Trees

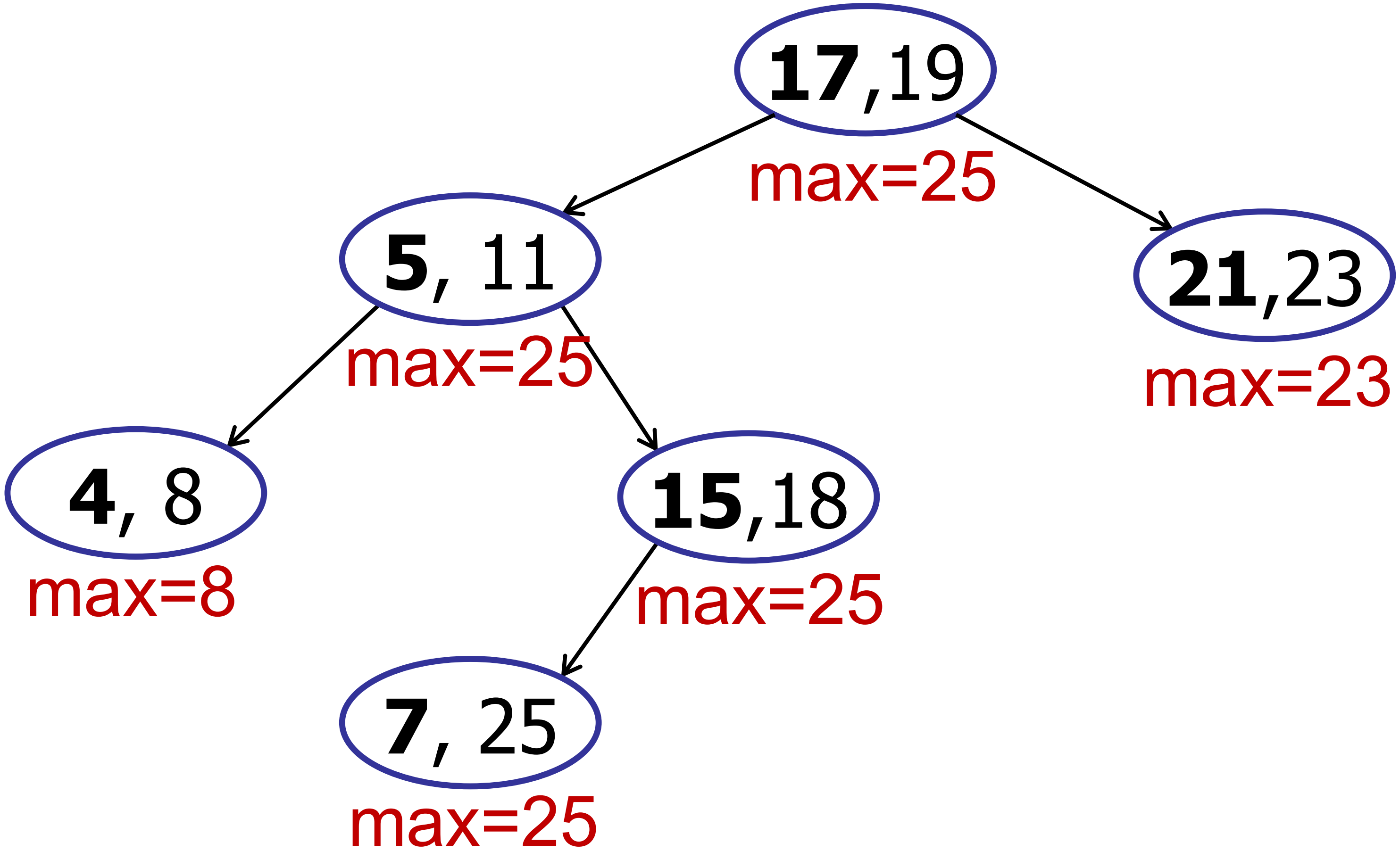
interval-search(x) : find interval containing x

```
interval-search(x)
    c = root;
    while (c != null and x is not in c.interval) do
        if (c.left == null) then
            c = c.right;
        else if  (x > c.left.max)  then
            c = c.right;
        else c = c.left;
    return c.interval;
```

Interval Trees

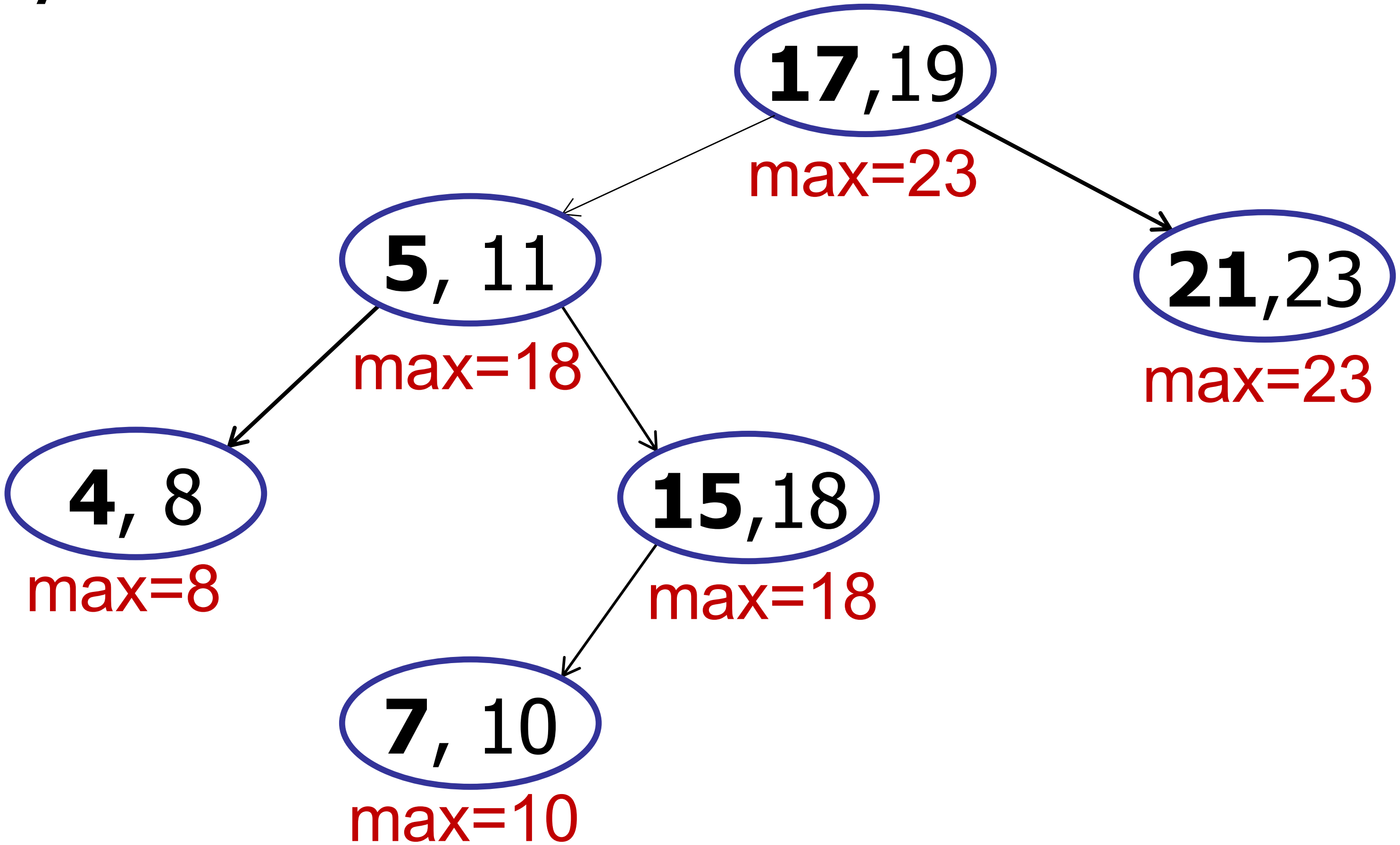
Will any search find (21, 23)?

Yes or No on Zoom.



Interval Trees

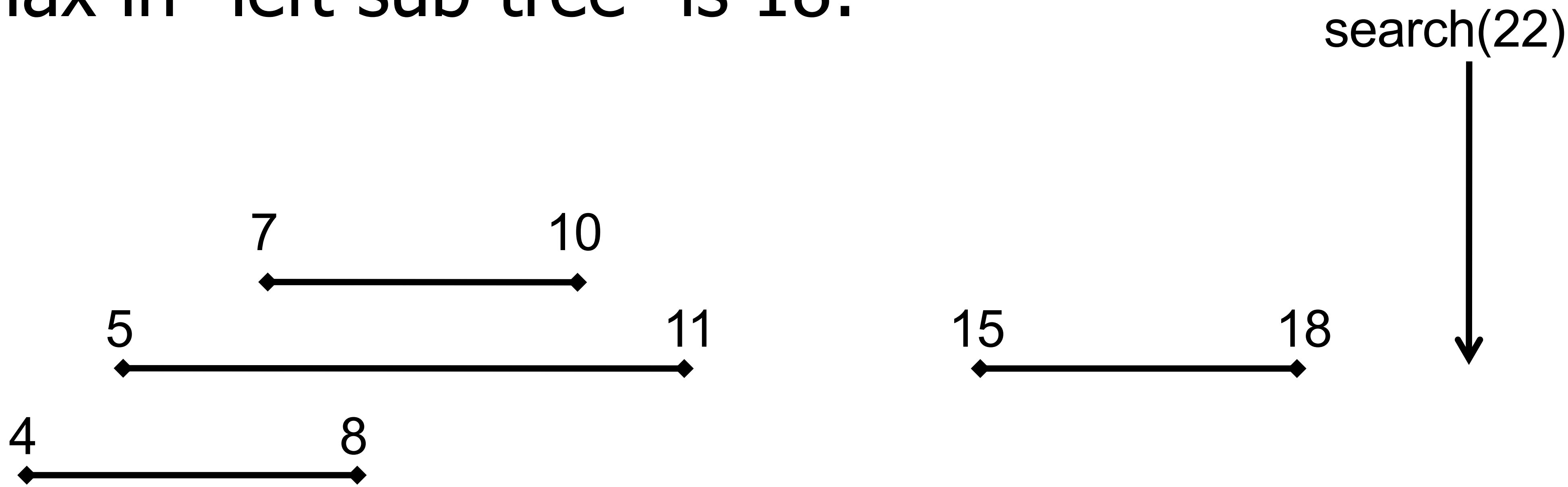
Why does it work?



Claim: If search goes right, then no overlap in left subtree.

Interval Trees

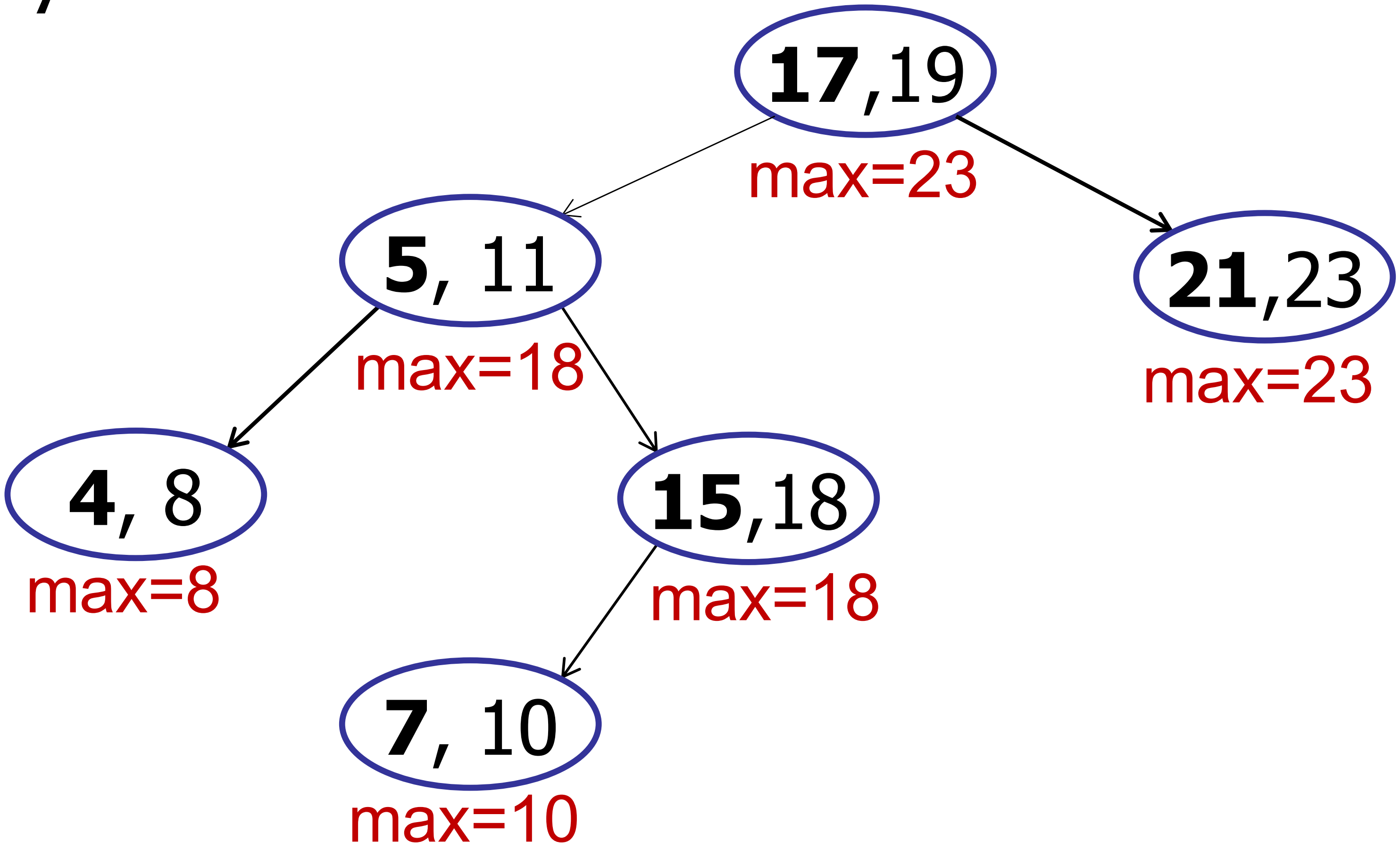
Max in “left sub-tree” is 18:



Safe to go right: 22 is not in the left sub-tree.

Interval Trees

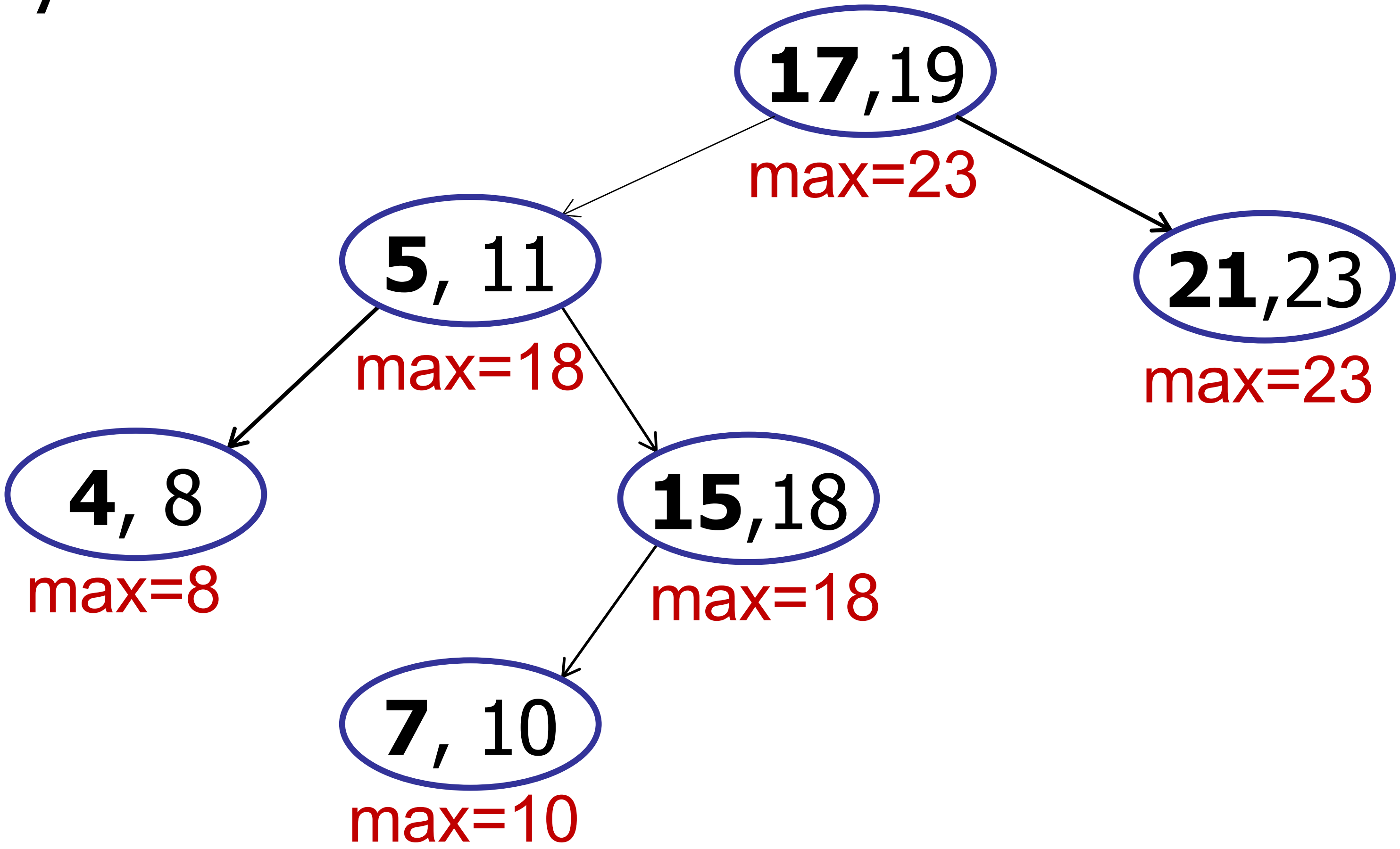
Why does it work?



Claim: If search goes left and there is no overlap in the left subtree…

Interval Trees

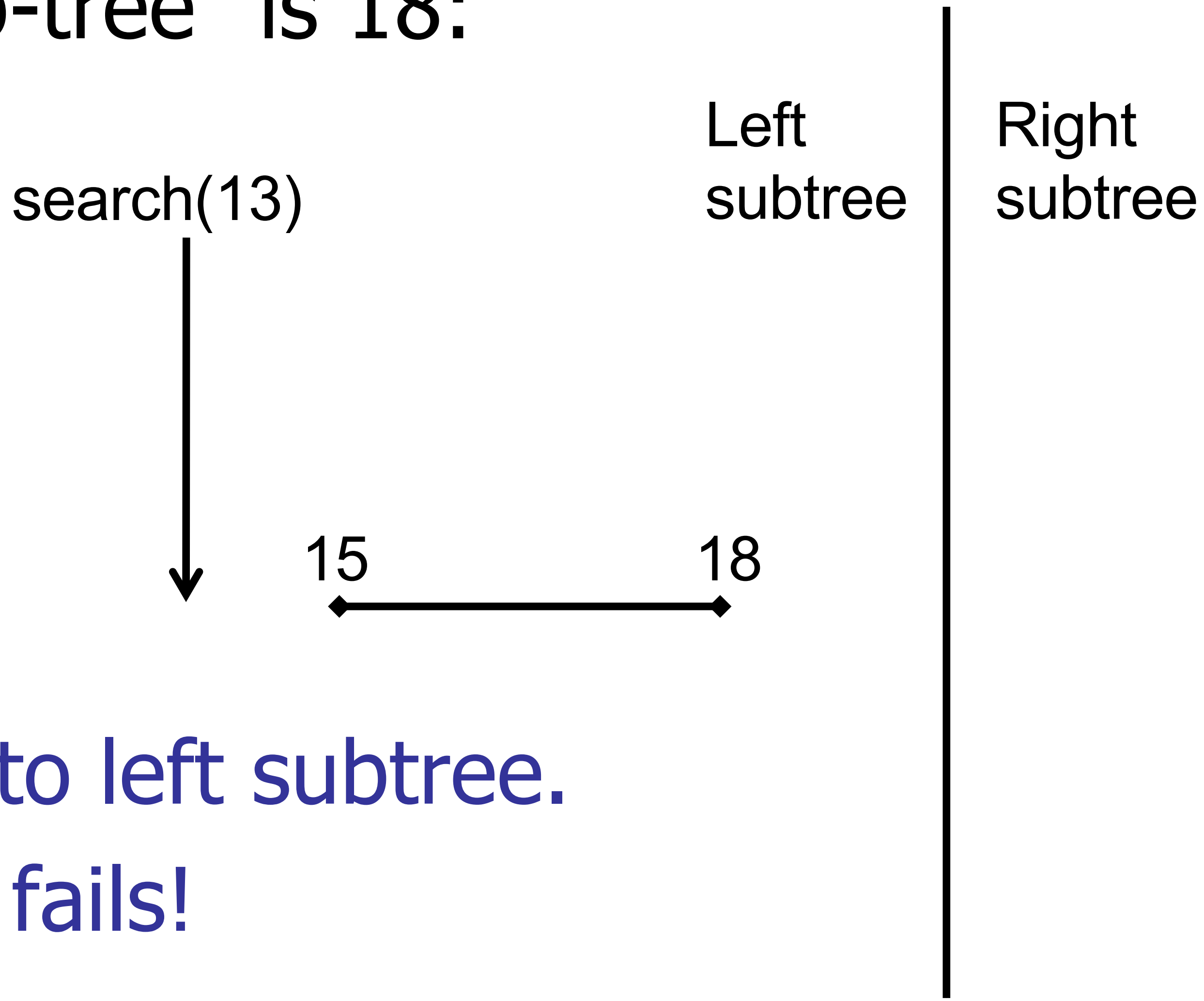
Why does it work?



Claim: If search goes left, then safe to go left.

Interval Trees

Max in “left sub-tree” is 18:

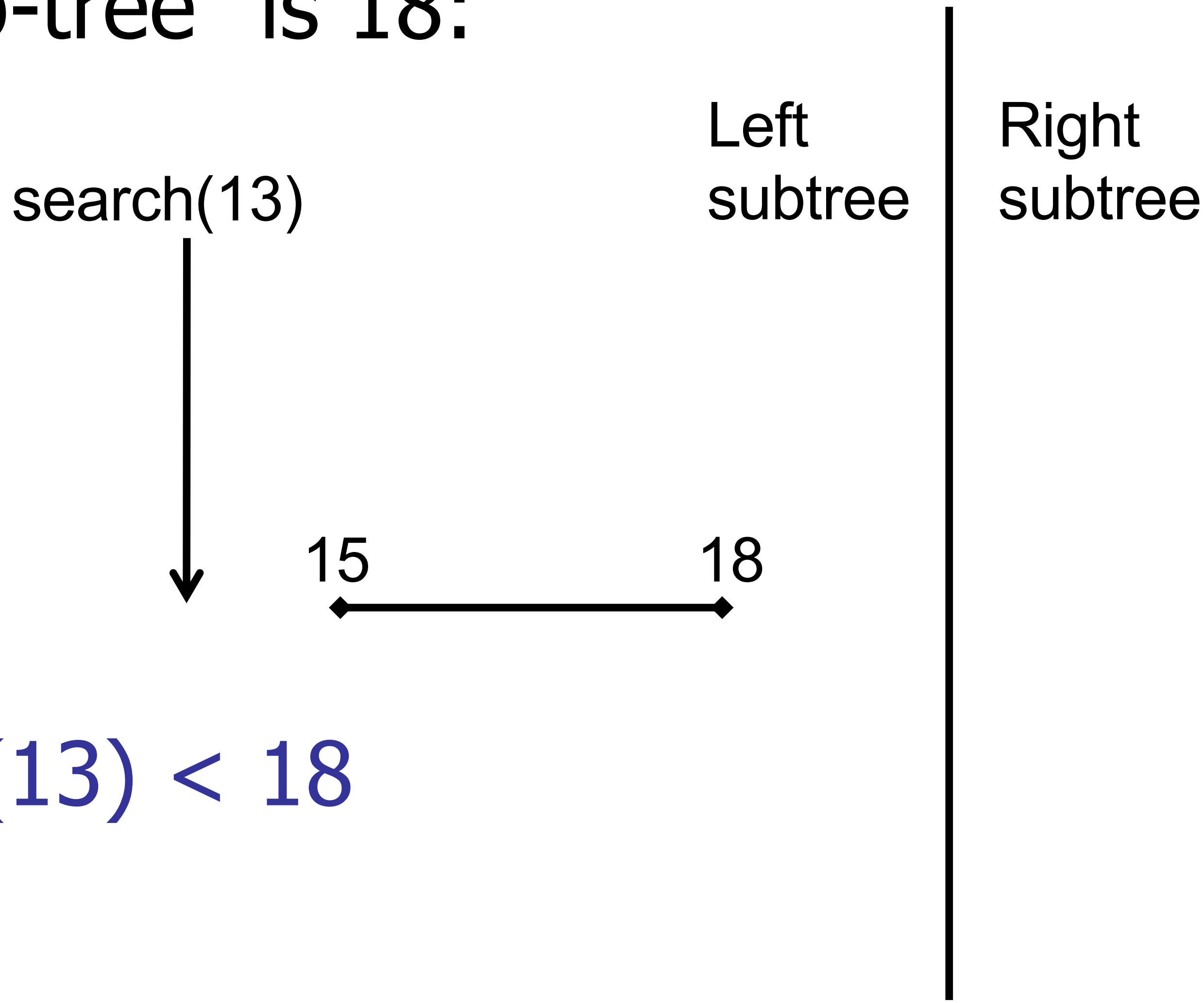


Assume we go to left subtree.

Assume search fails!

Interval Trees

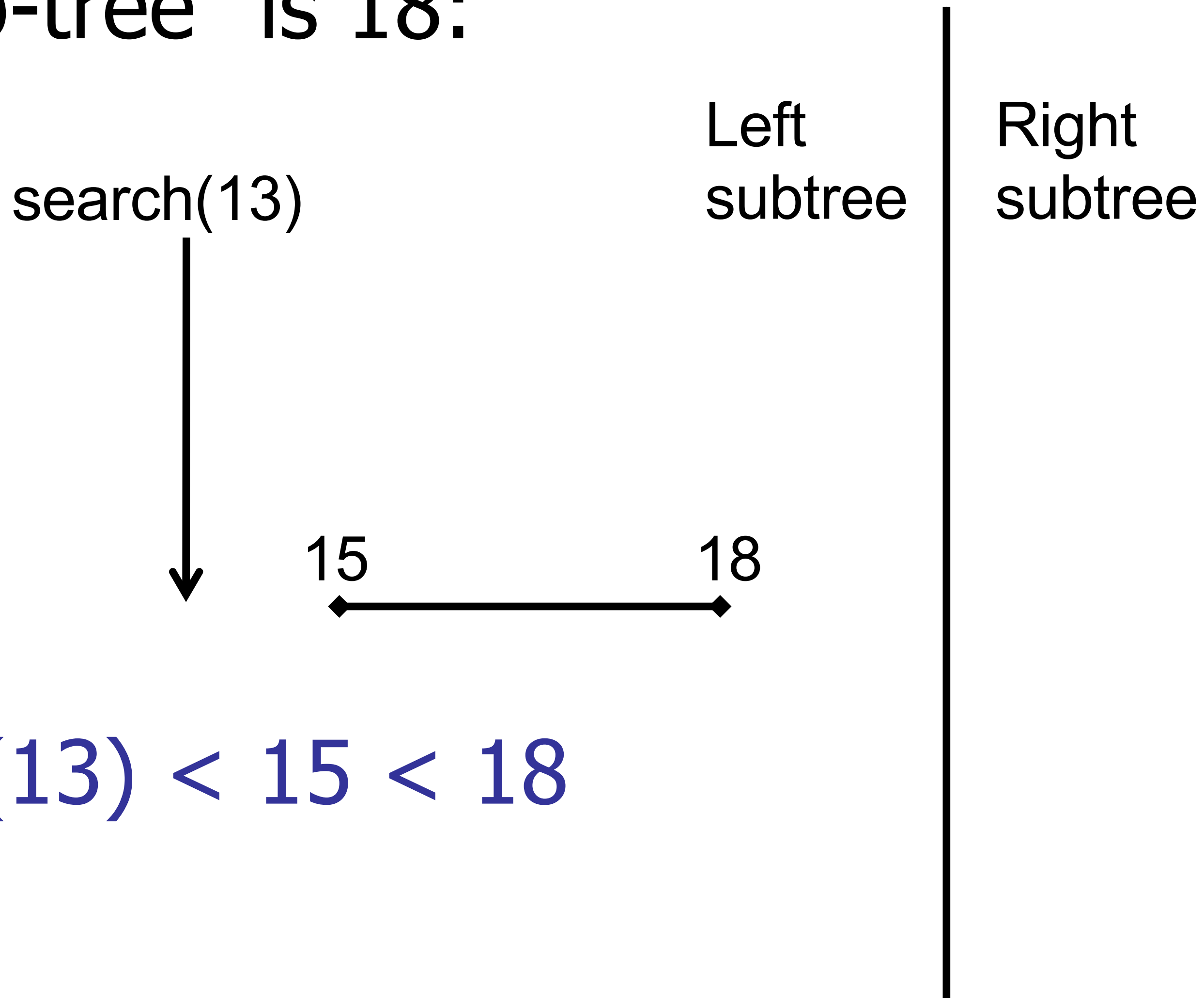
Max in “left sub-tree” is 18:



Go left: search(13) < 18

Interval Trees

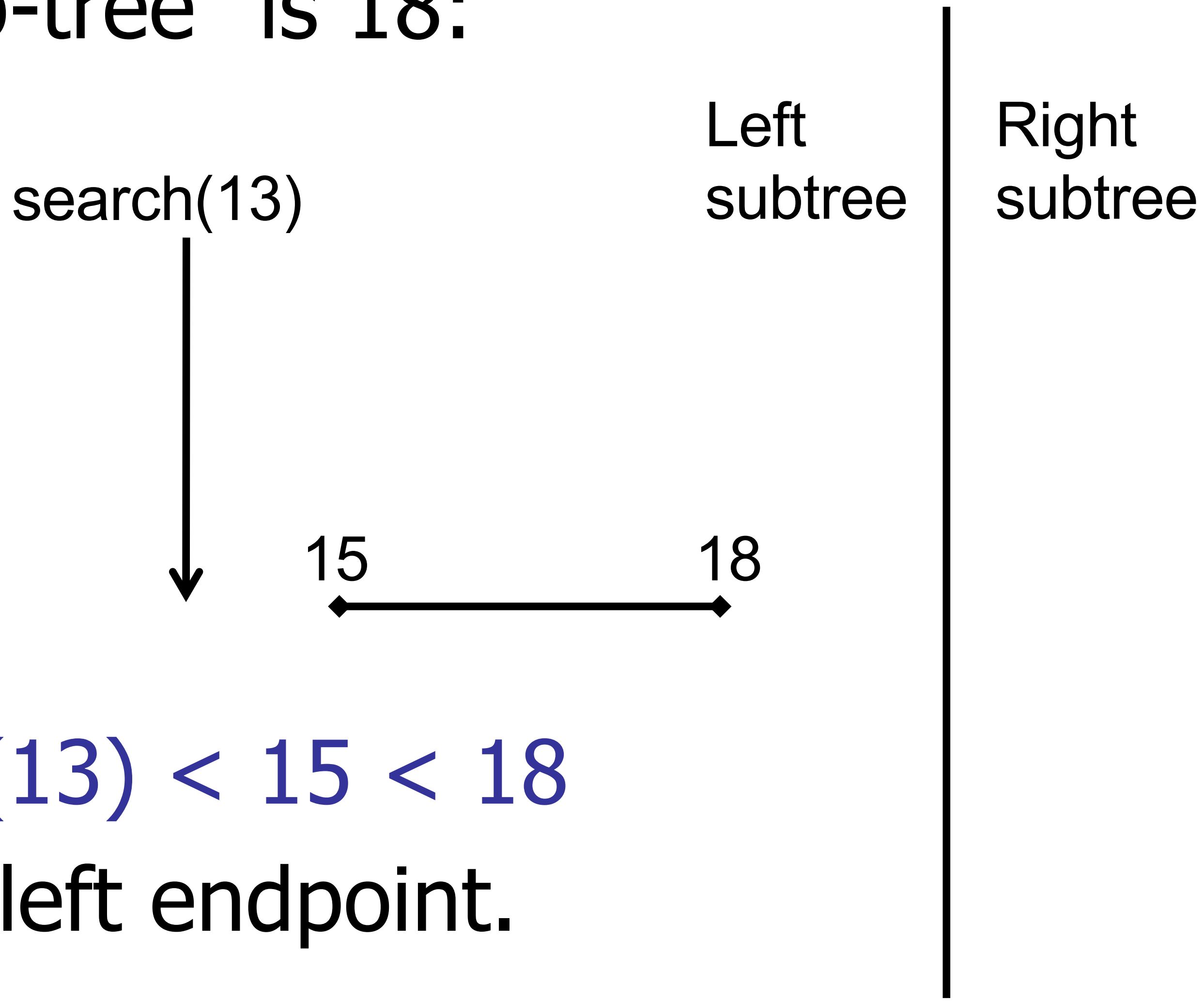
Max in “left sub-tree” is 18:



Go left: search(13) < 15 < 18

Interval Trees

Max in "left sub-tree" is 18:

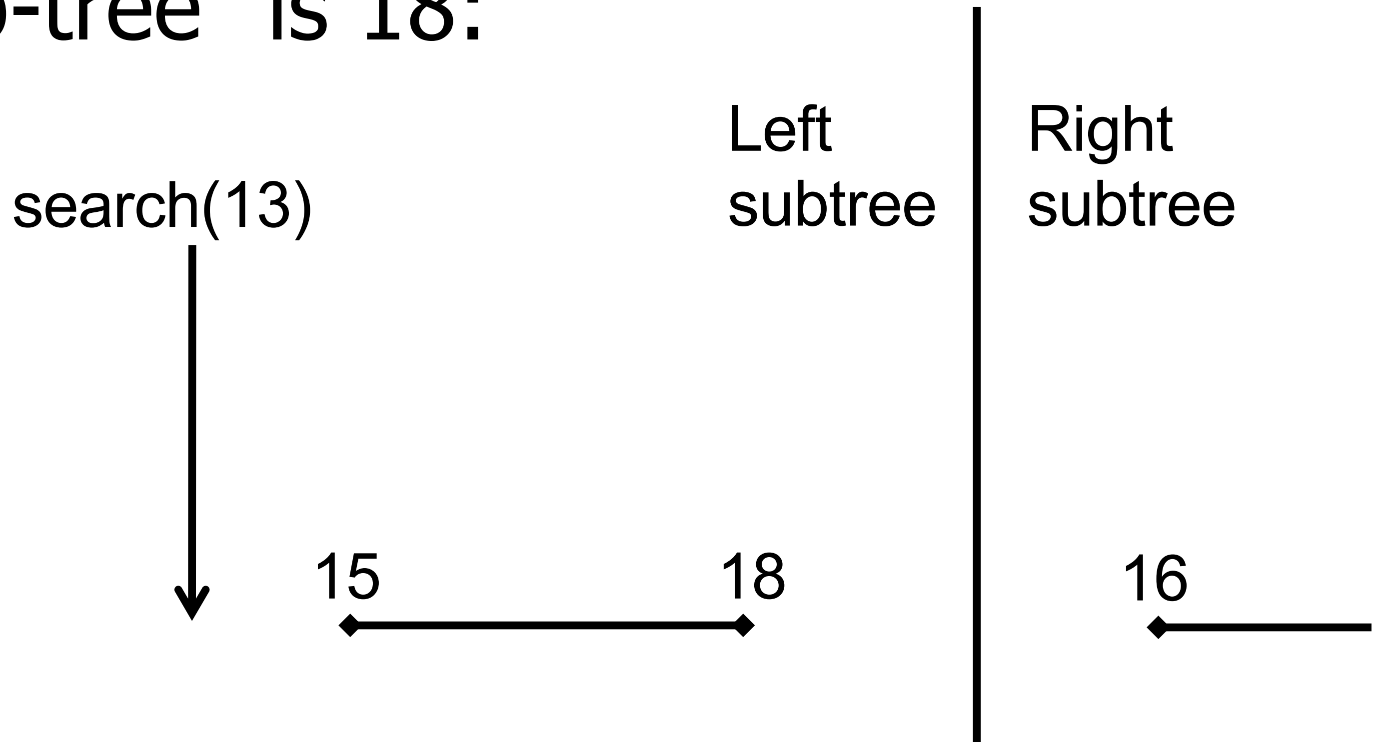


Go left: search(13) < 15 < 18

Tree sorted by left endpoint.

Interval Trees

Max in “left sub-tree” is 18:



Go left: search(13) < 15 < 18

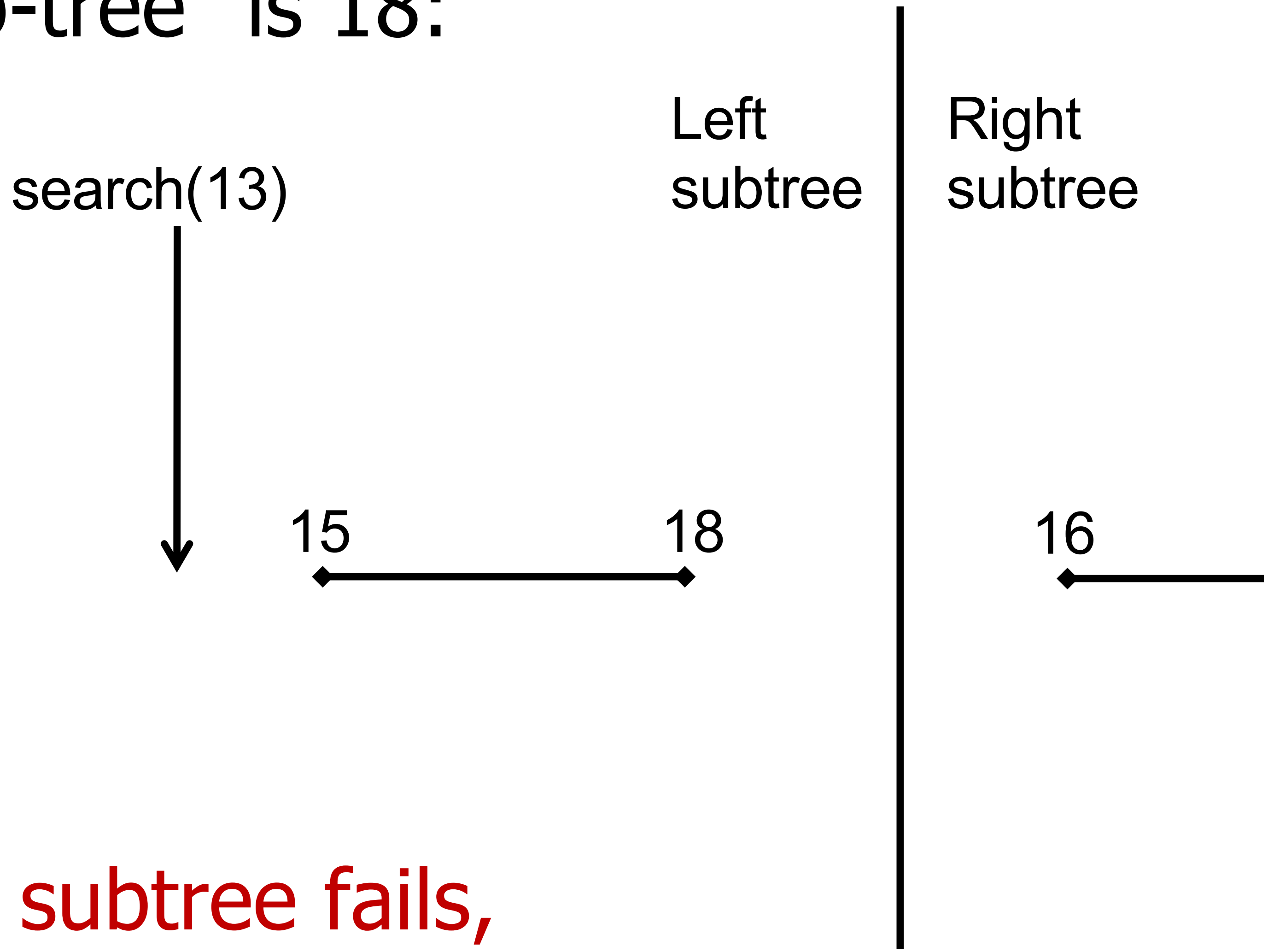
Tree sorted by left endpoint.

13 < every interval in right subtree

➔ Search also would fail in right subtree

Interval Trees

Max in “left sub-tree” is 18:

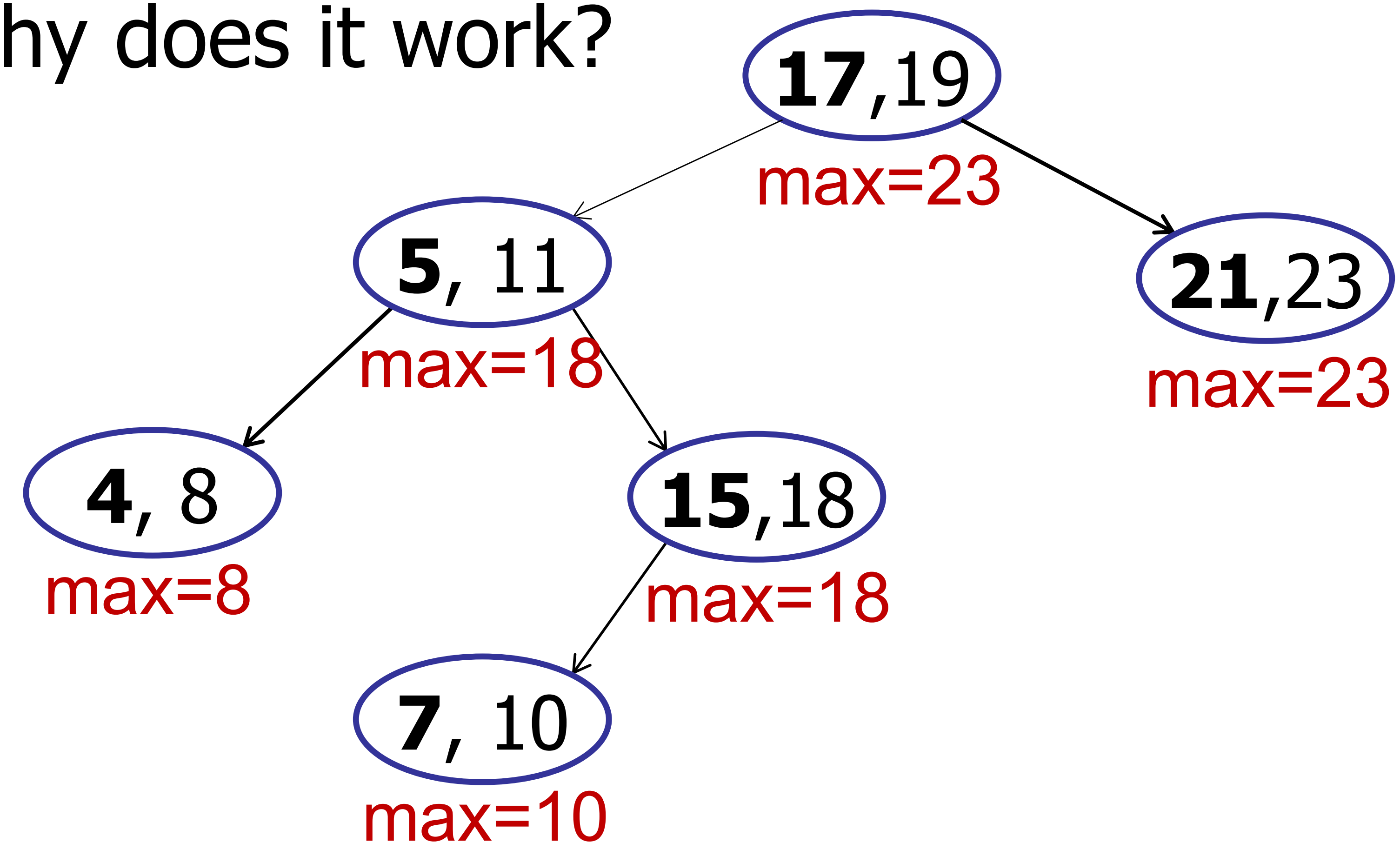


If search in left subtree fails,

Then search also would fail in right subtree!

Interval Trees

Why does it work?



Claim: If search goes left and fails, then key < every interval in right sub-tree.

Interval Trees

If search goes right: then no interval in left subtree.

➔ Either search finds key in right subtree or it is not in the tree.

If search goes left: if there is no interval in left subtree, then there is no interval in right subtree either.

➔ Either search finds key in left subtree or it is not in the tree.

Conclusion: search finds an overlapping interval, if it exists.

The running time of interval-search is:

1. $O(1)$
2. $\boxed{O(\log n)}$
3. $O(n)$
4. $O(n \log n)$
5. $O(n^2)$
6. Can't say.

ARCHIPELAGO

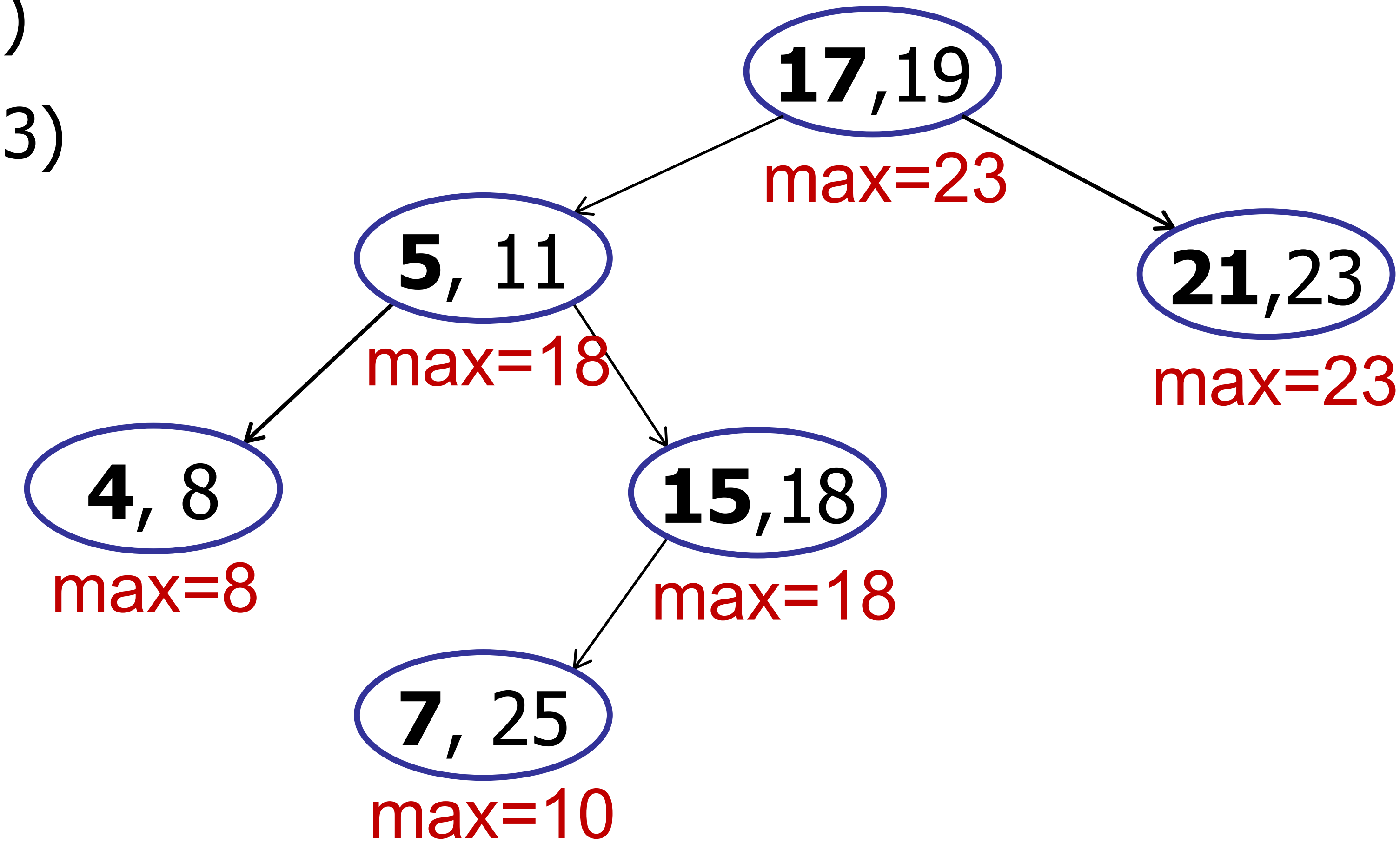
is open

Interval Trees

Extension: List all intervals that overlap with point?

E.g.: search(22) returns:

- (7,25)
- (21,23)



Interval Trees

Extension: List all intervals that overlap with point?

All-Overlaps Algorithm:

Repeat until no more intervals:
- Search for interval.
- Add to list.
- Delete interval.

Repeat for all intervals on list:
- Add interval back to tree.

The running time of All-Overlaps, if there are k overlapping intervals?

1. O(1)
2. O(k)
3. $\boxed{\text{O(k log n)}}$
4. O(k + log n)
5. O(kn)
6. O(kn log n)

ARCHIPELAGO

is open

Interval Trees

Extension: List all intervals that overlap with point?

All-Overlaps Algorithm: O(k log n)

Repeat until no more intervals:

- Search for interval.
- Add to list.
- Delete interval.

Repeat for all intervals on list:

- Add interval back to tree.

Best known solution: O(k + log n)

Today

Three examples of augmenting BSTs

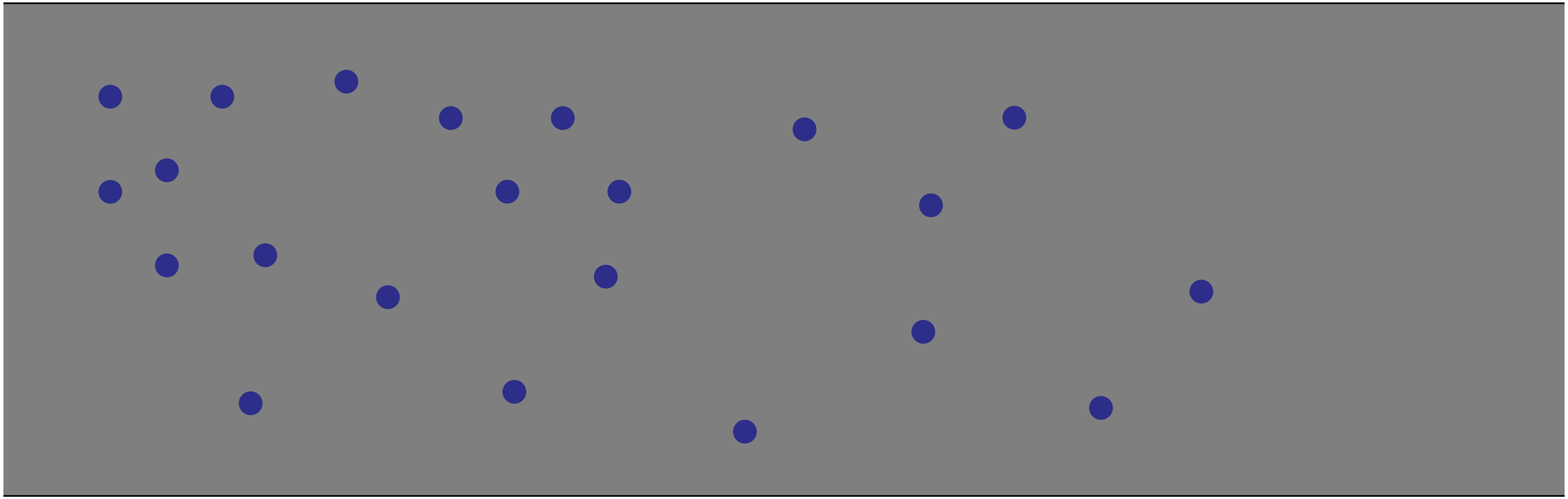
1. ~~Order Statistics~~

2. ~~Intervals~~

3. Orthogonal Range Searching

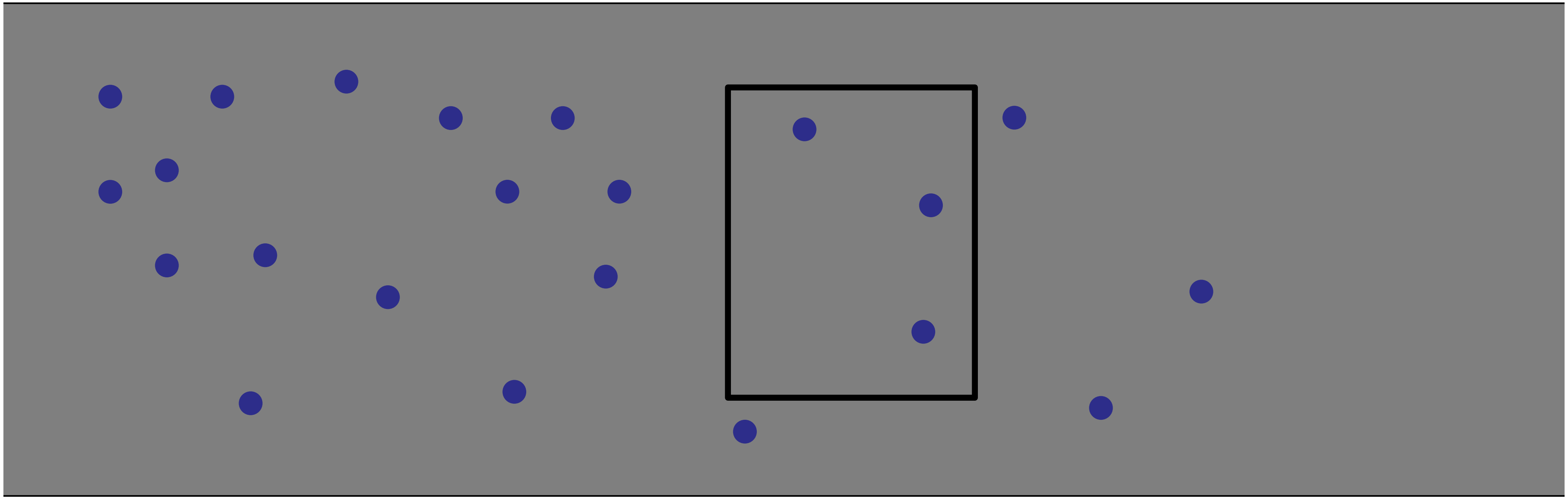
Orthogonal Range Searching

Input: n points in a 2d plane



Orthogonal Range Searching

Input: n points in a 2d plane

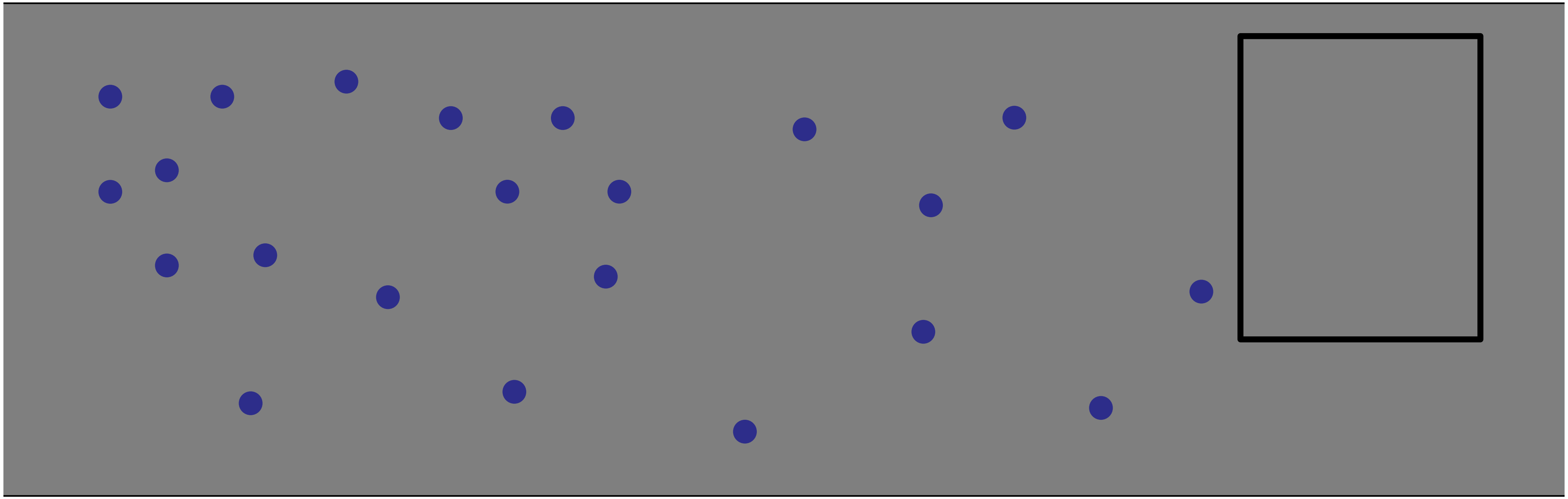


Query: Box

- Contains at least one point?
- How many?

Orthogonal Range Searching

Input: n points in a 2d plane

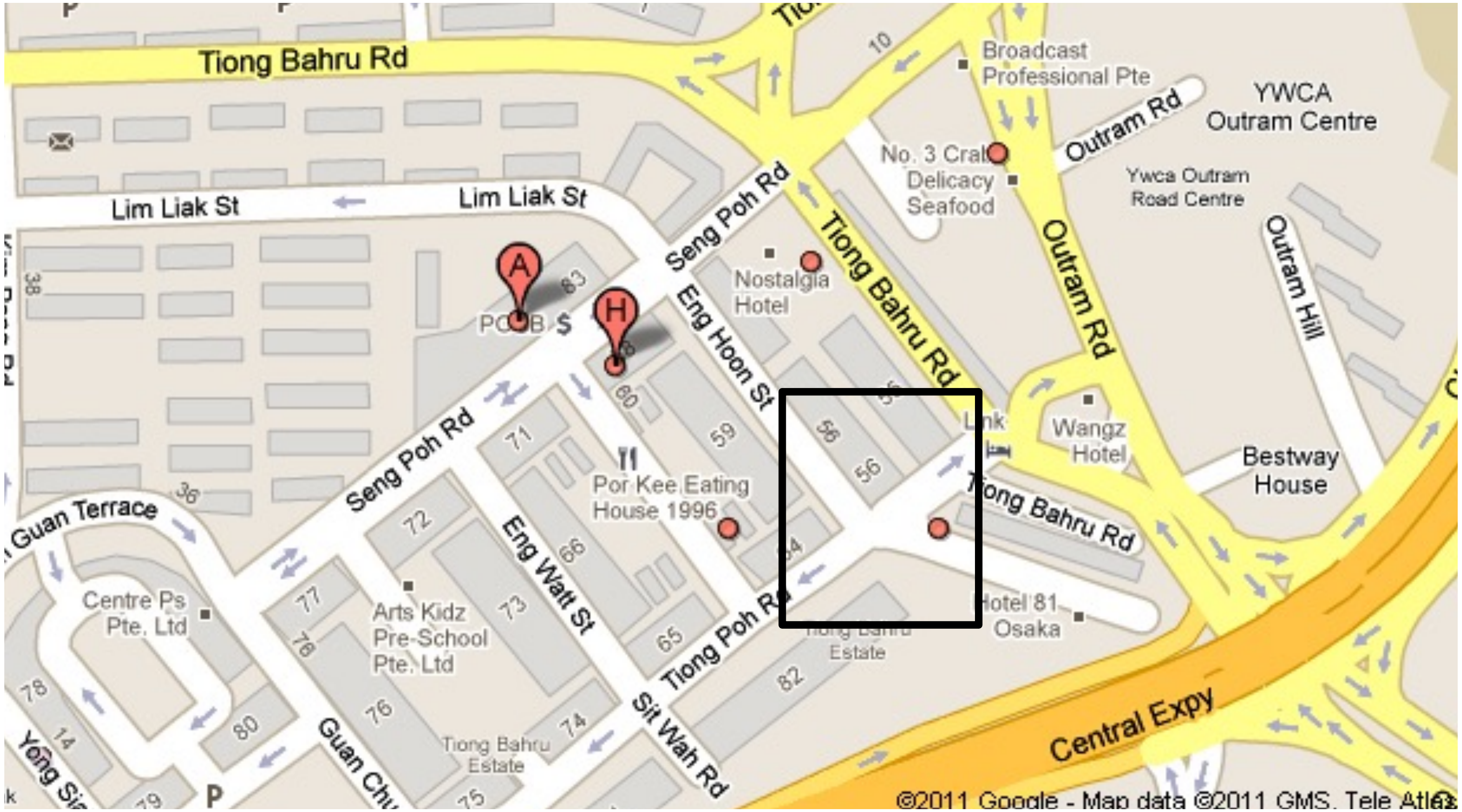


Query: Box

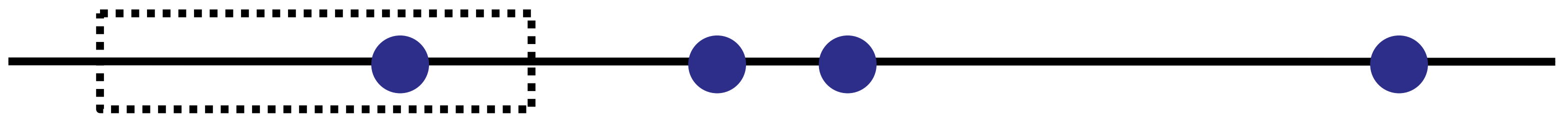
- Contains at least one point?
- How many?

Practical Example

Are there any good restaurants within one block of me?



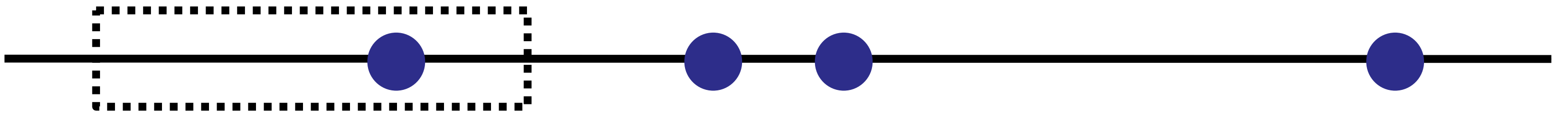
One Dimension



One Dimension

Range Queries

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



One Dimension

Strategy:

1. Use a binary search tree.

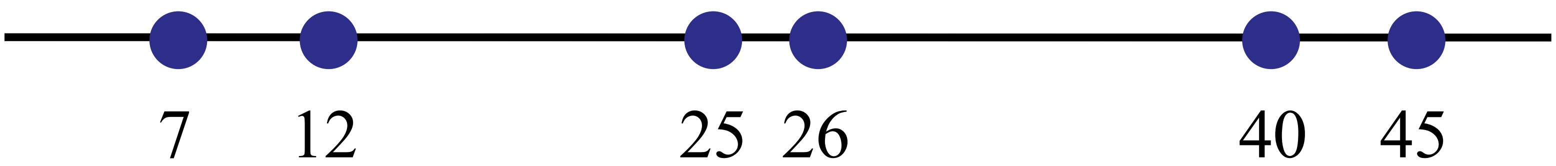
2. Store all points in the leaves of the tree. (Internal nodes store only copies.)

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

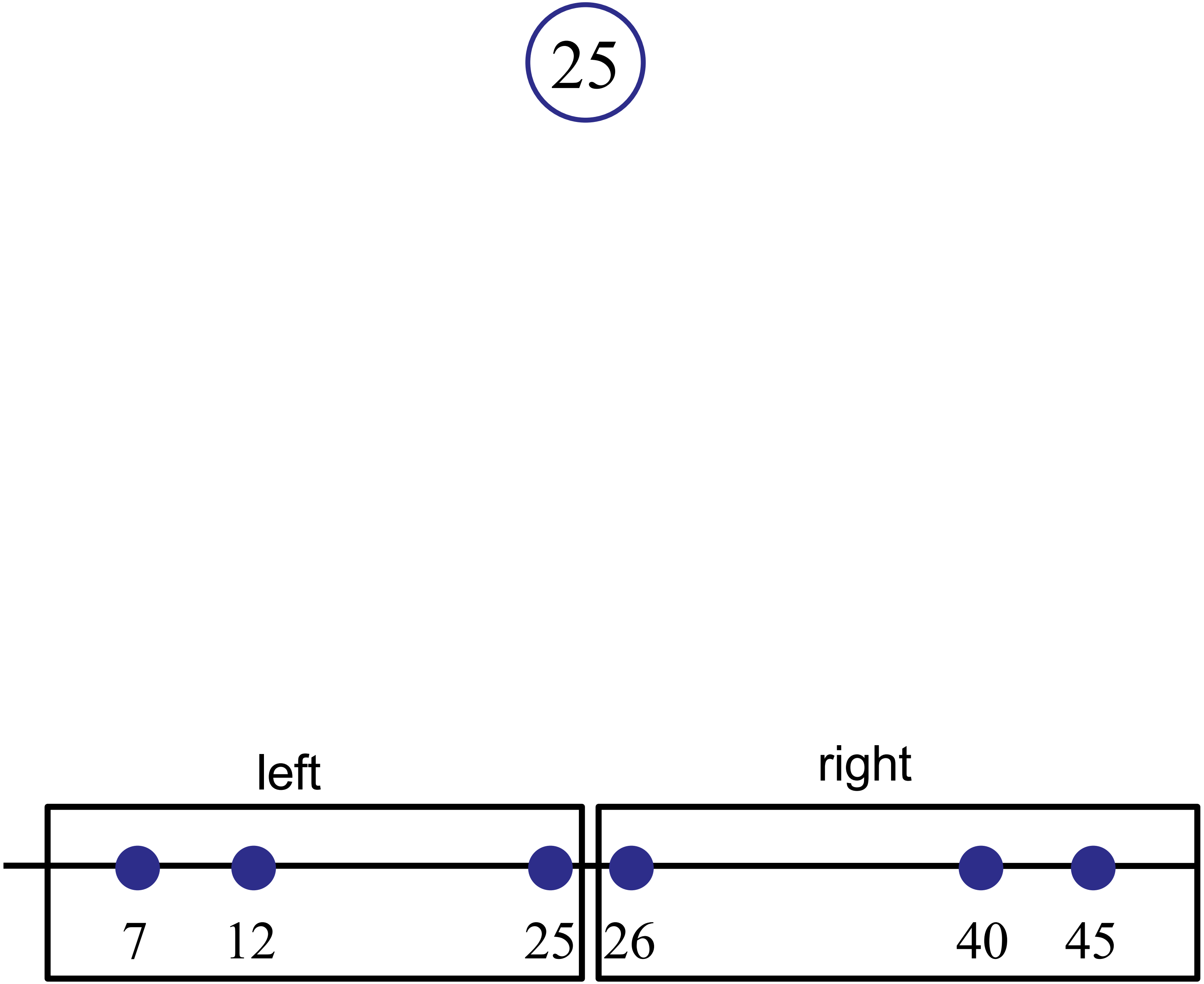
Example: what is the root?

ARCHIPELAGO

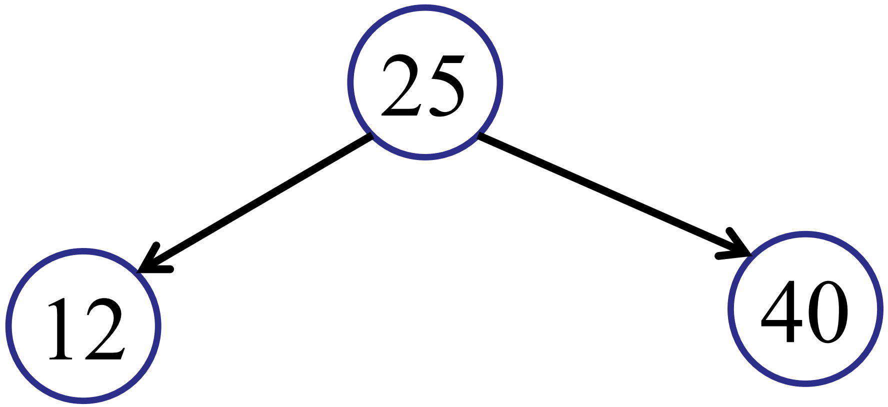
is open

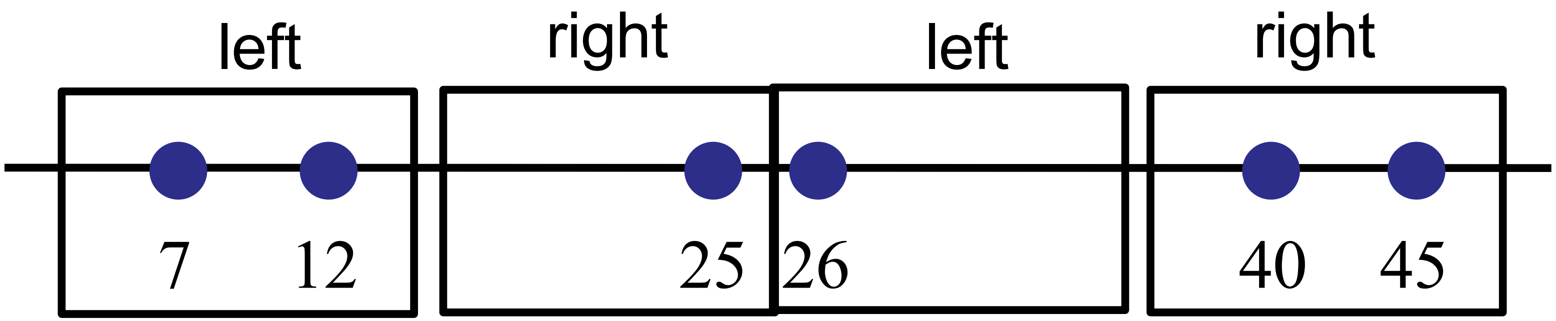


Example

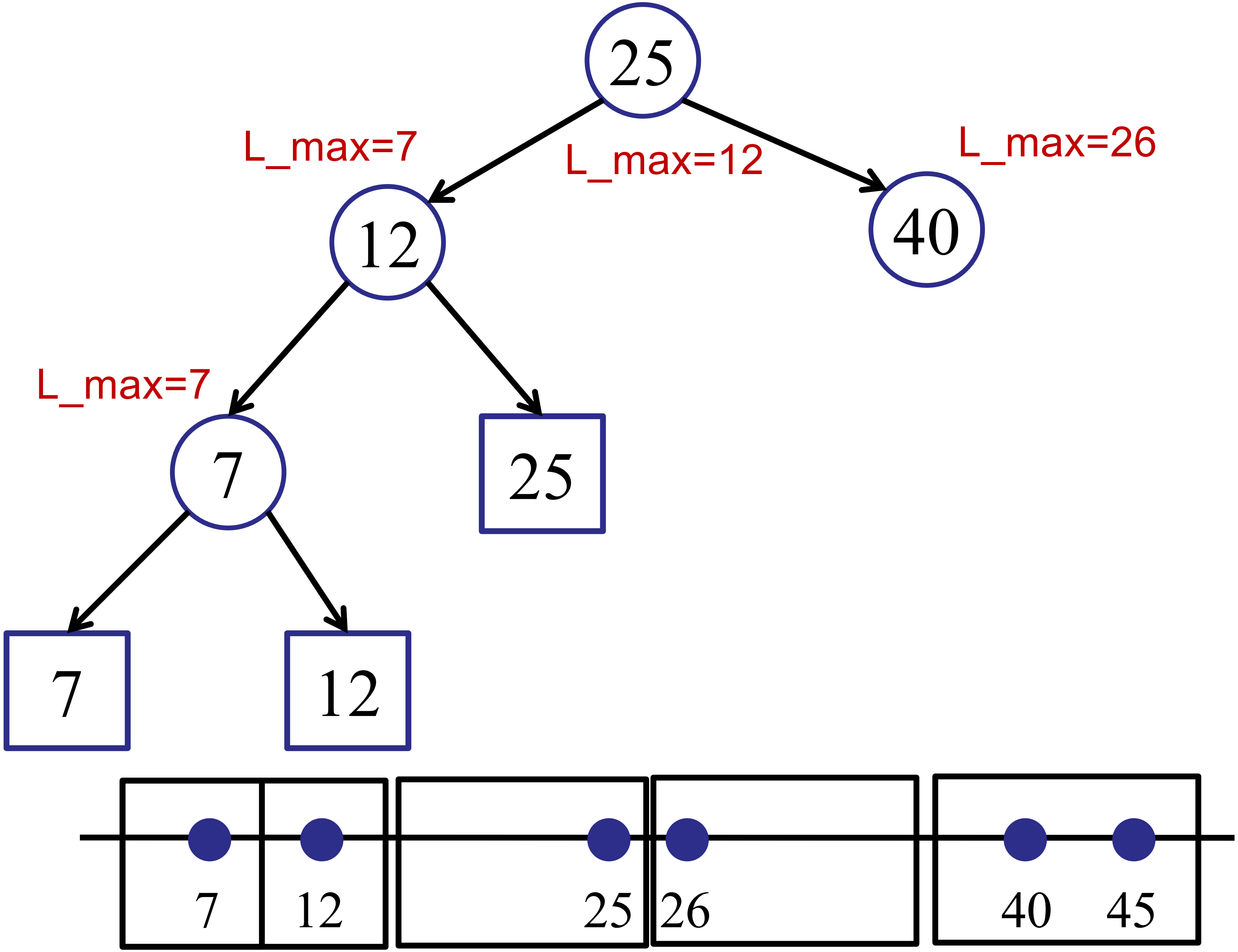


Example

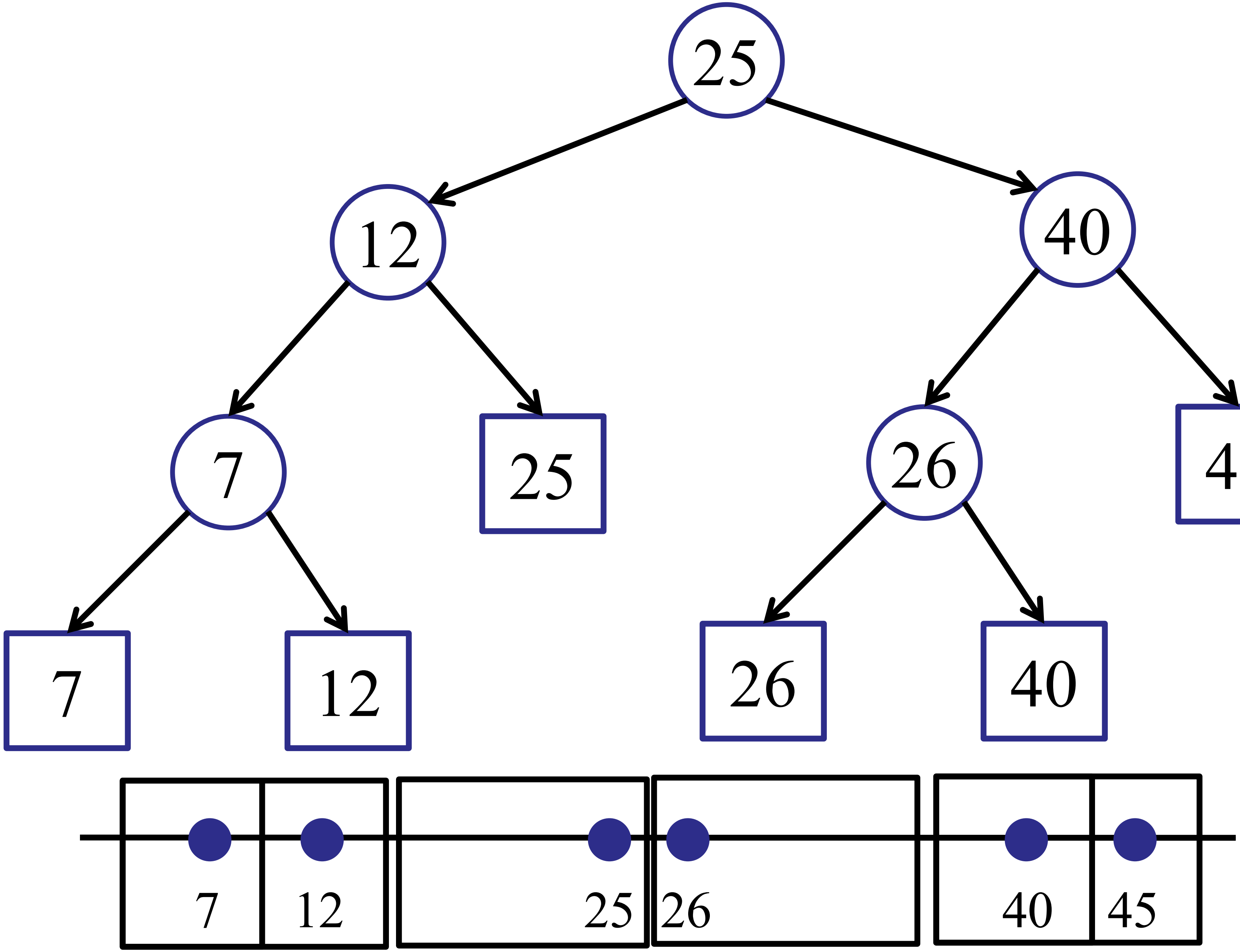




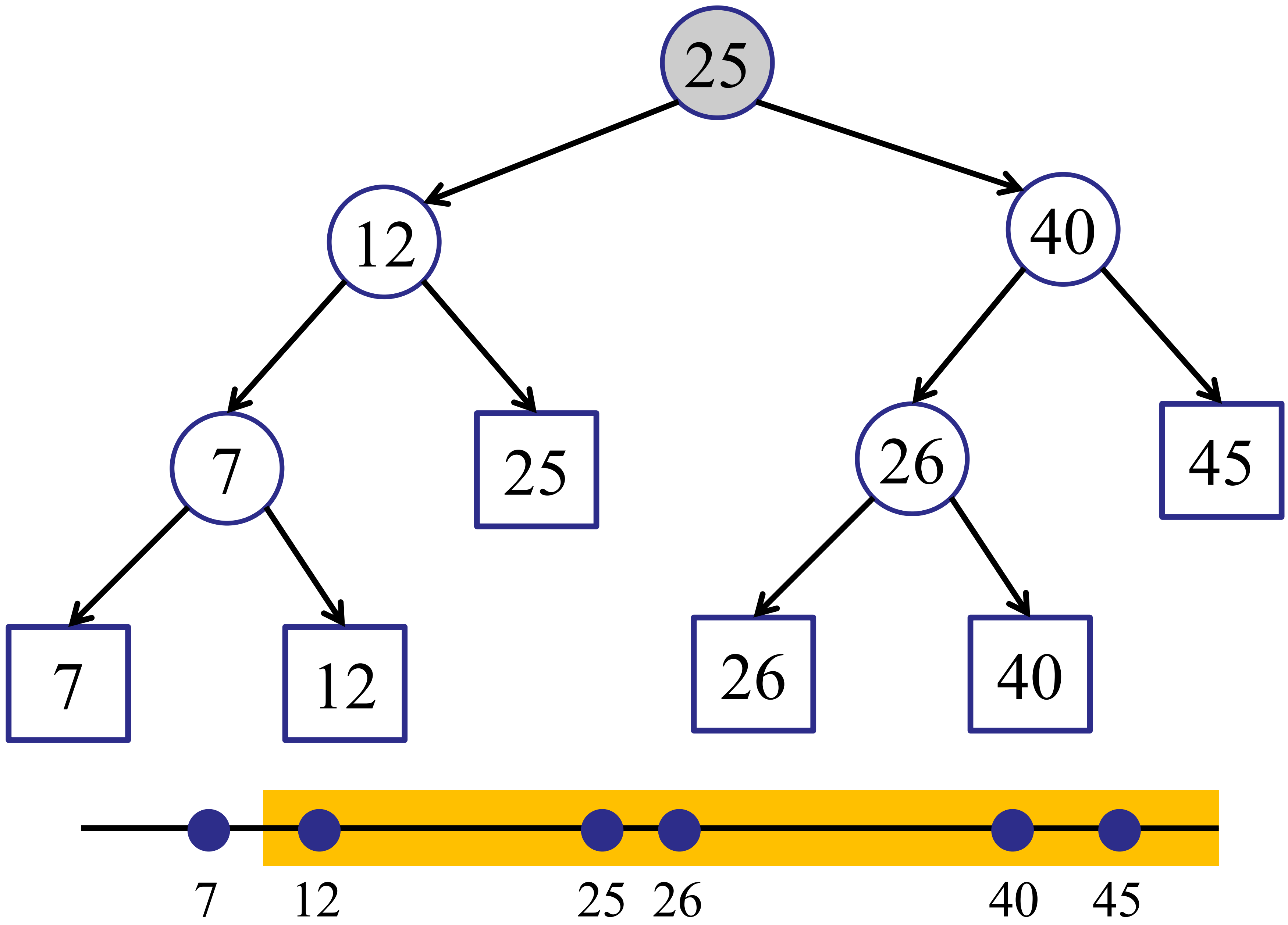
Example



Note: BST Property

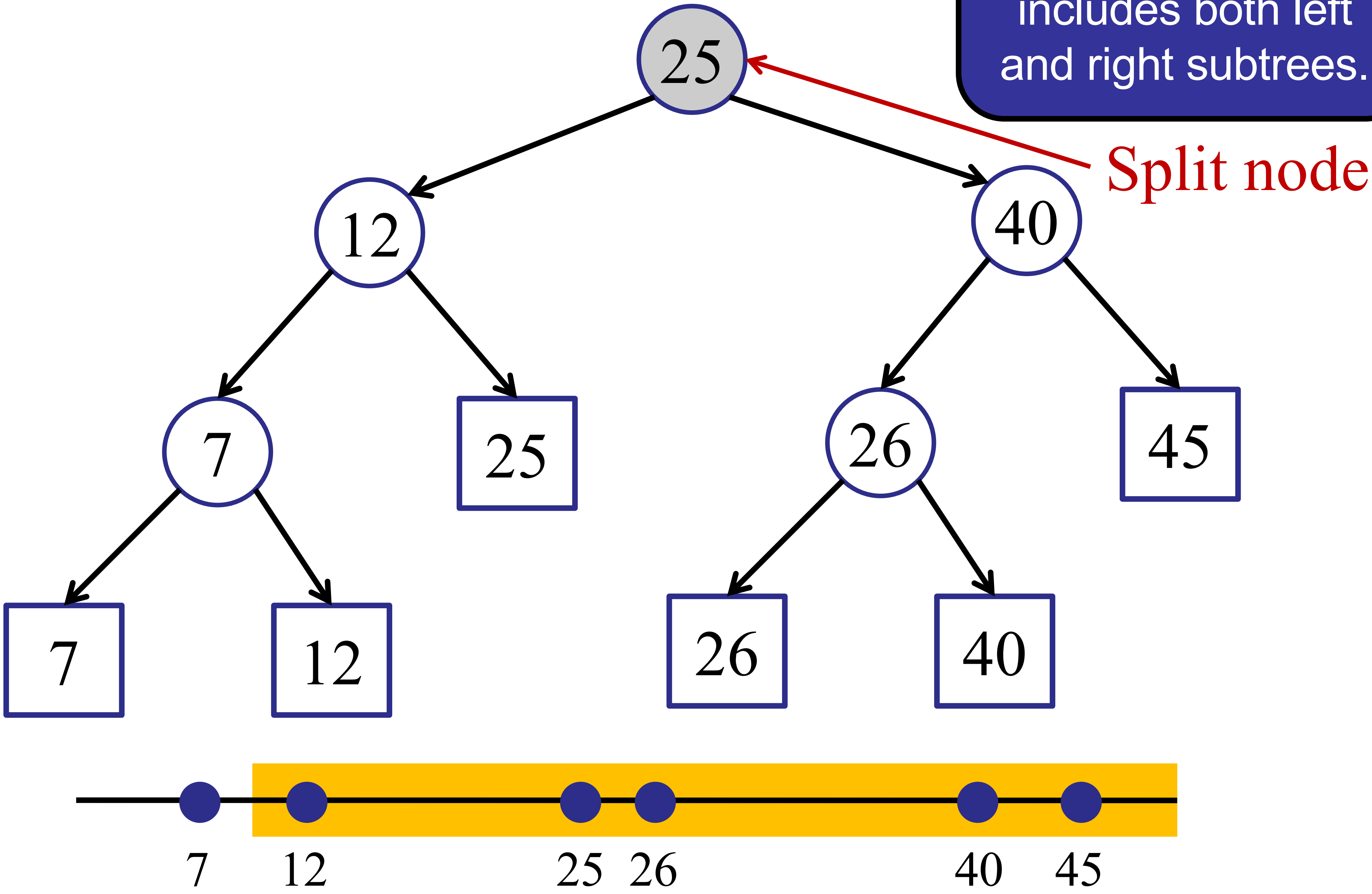


Example: query(10, 50)

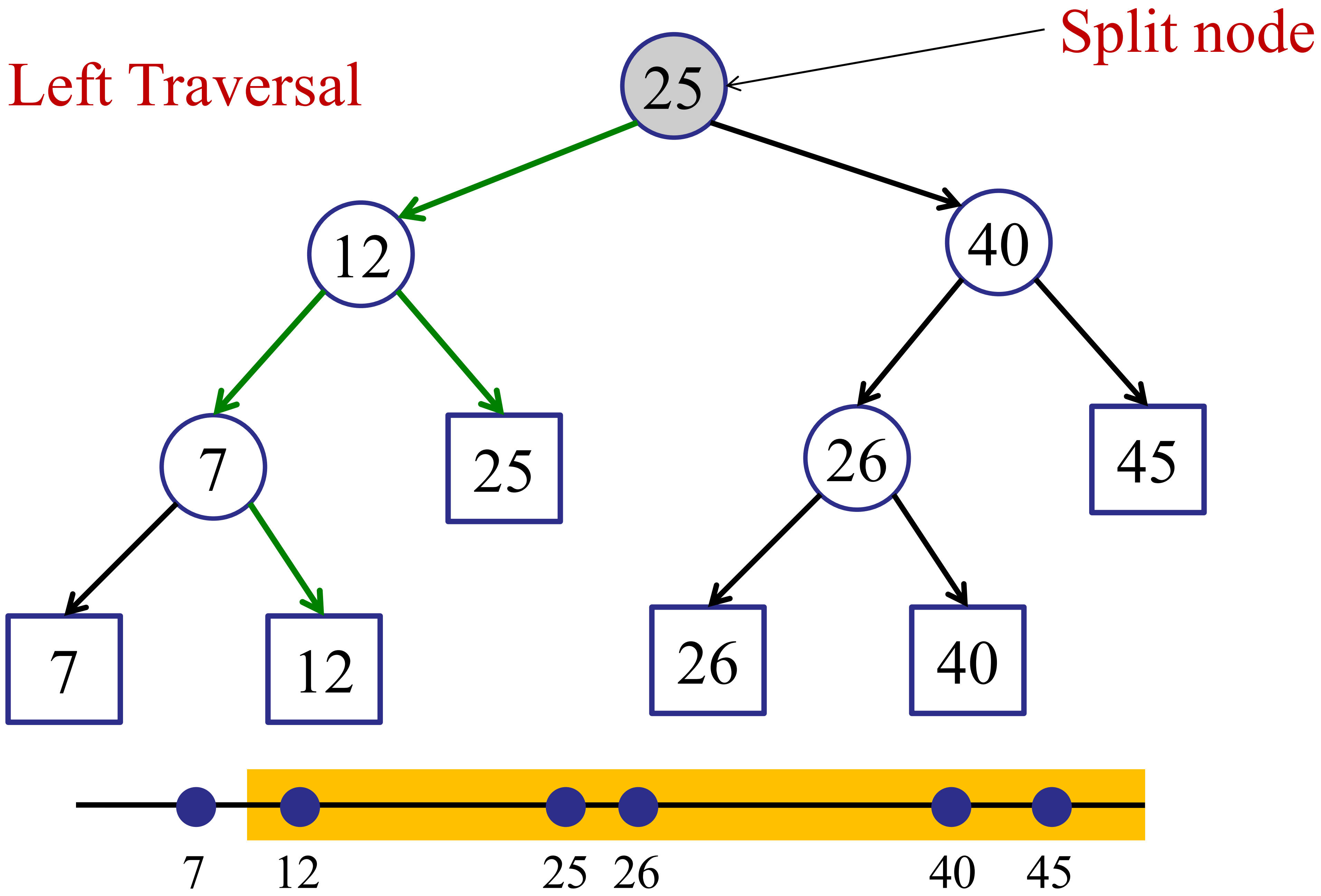


Example: query(10, 50)

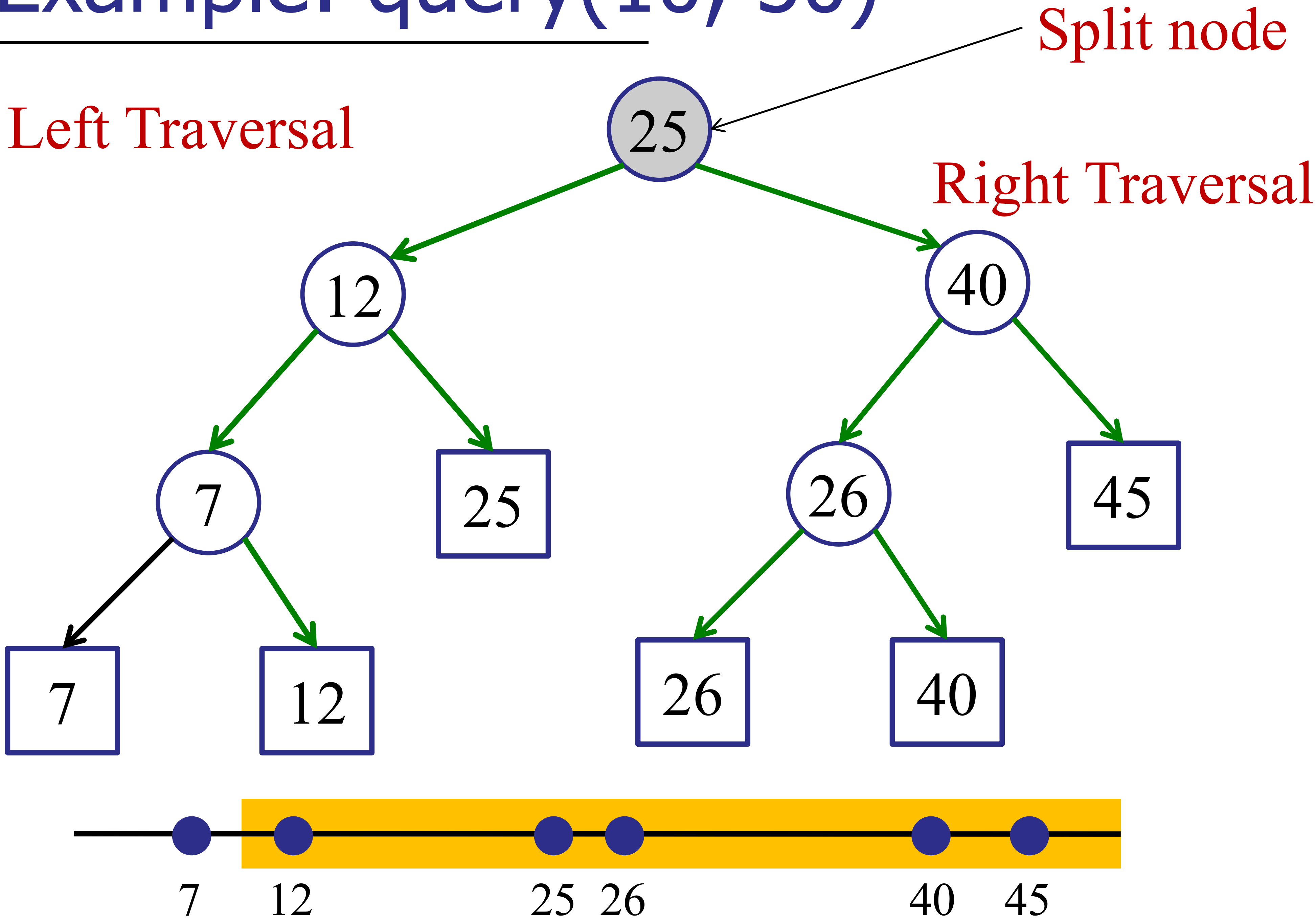
Highest node where search includes both left and right subtrees.



Example: query(10, 50)

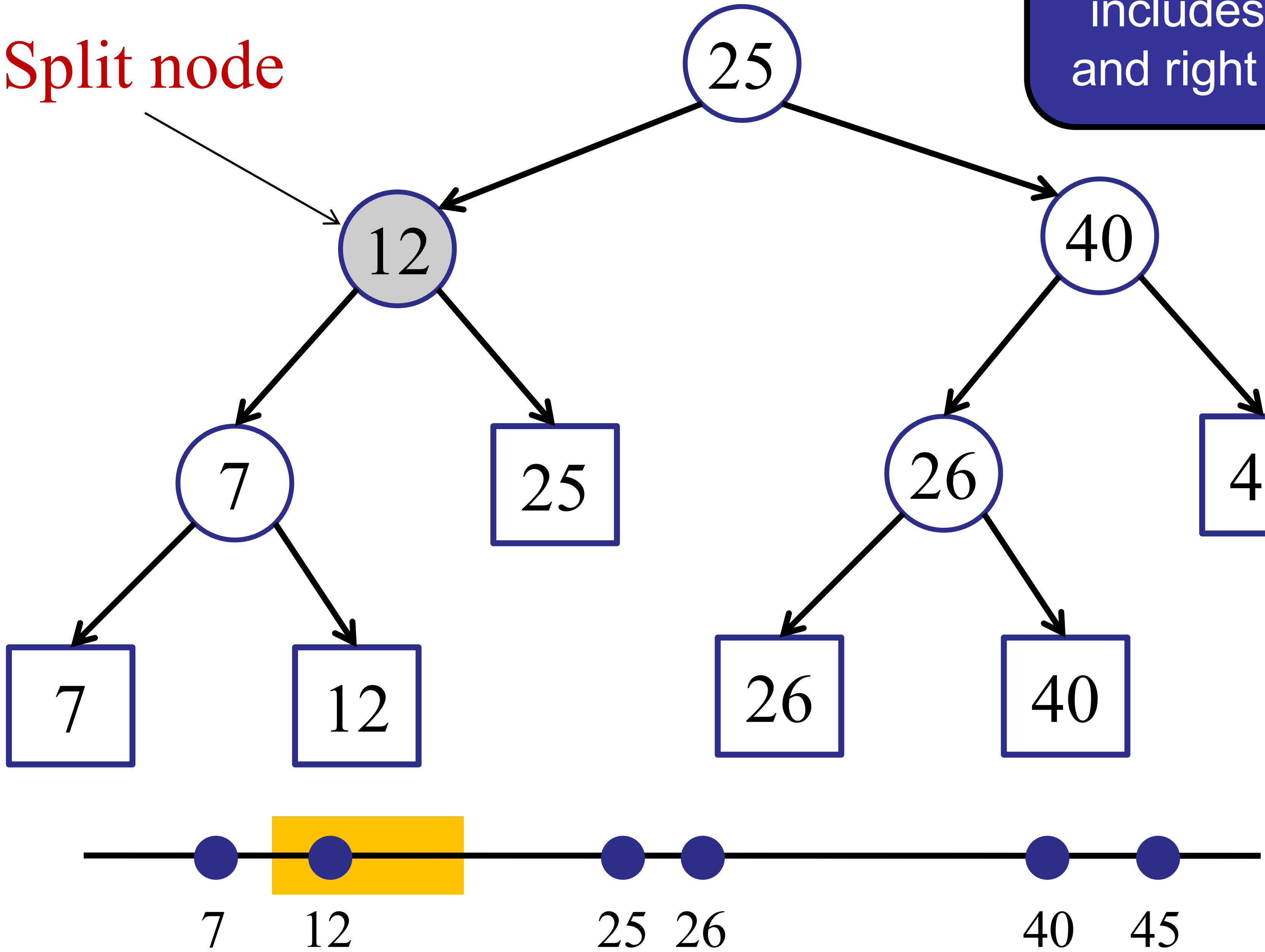


Example: query(10, 50)

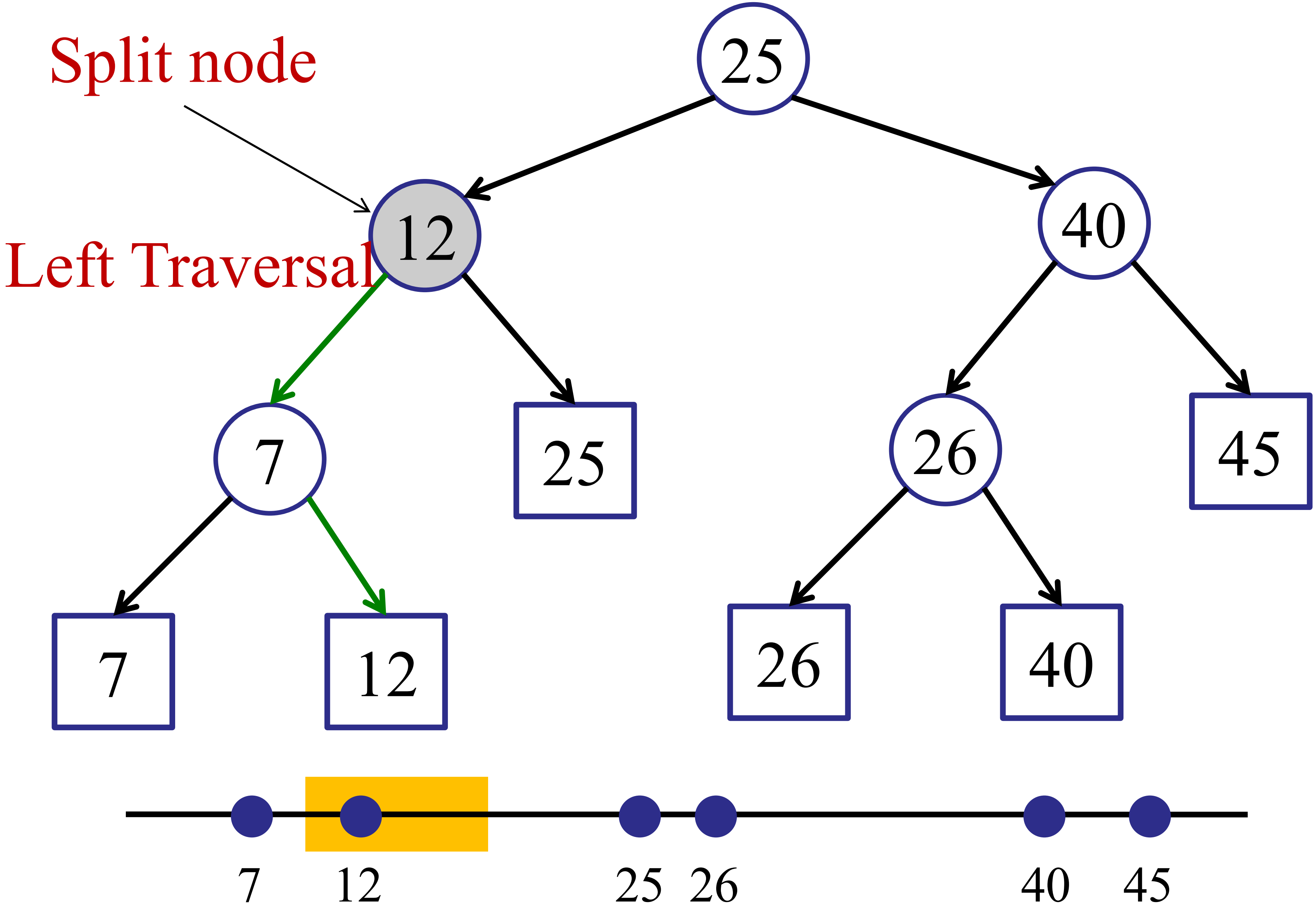


Example: query(8, 20)

Highest node where search includes both left and right subtrees.



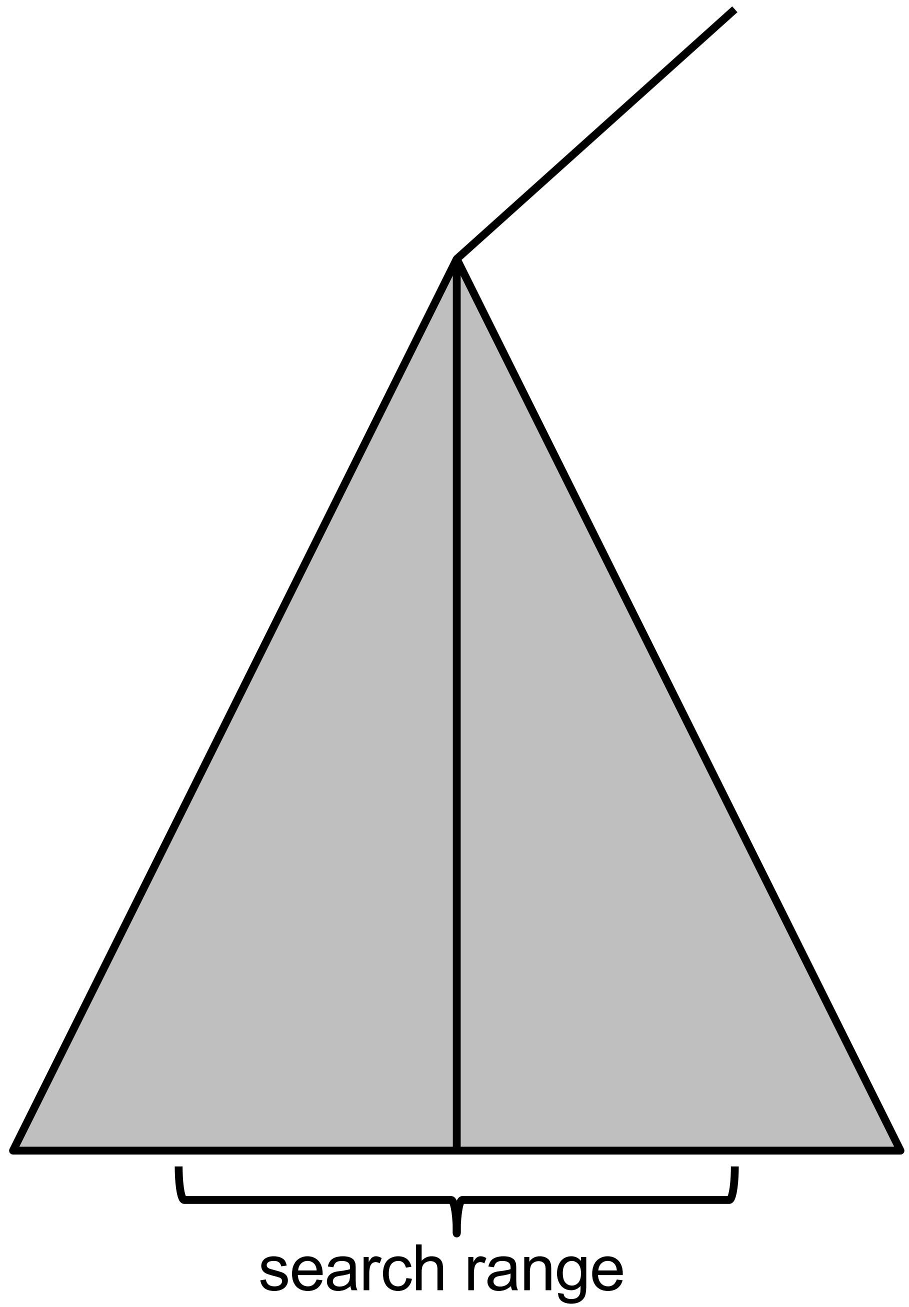
Example: query(8, 20)



One Dimensional Range Queries

Algorithm:

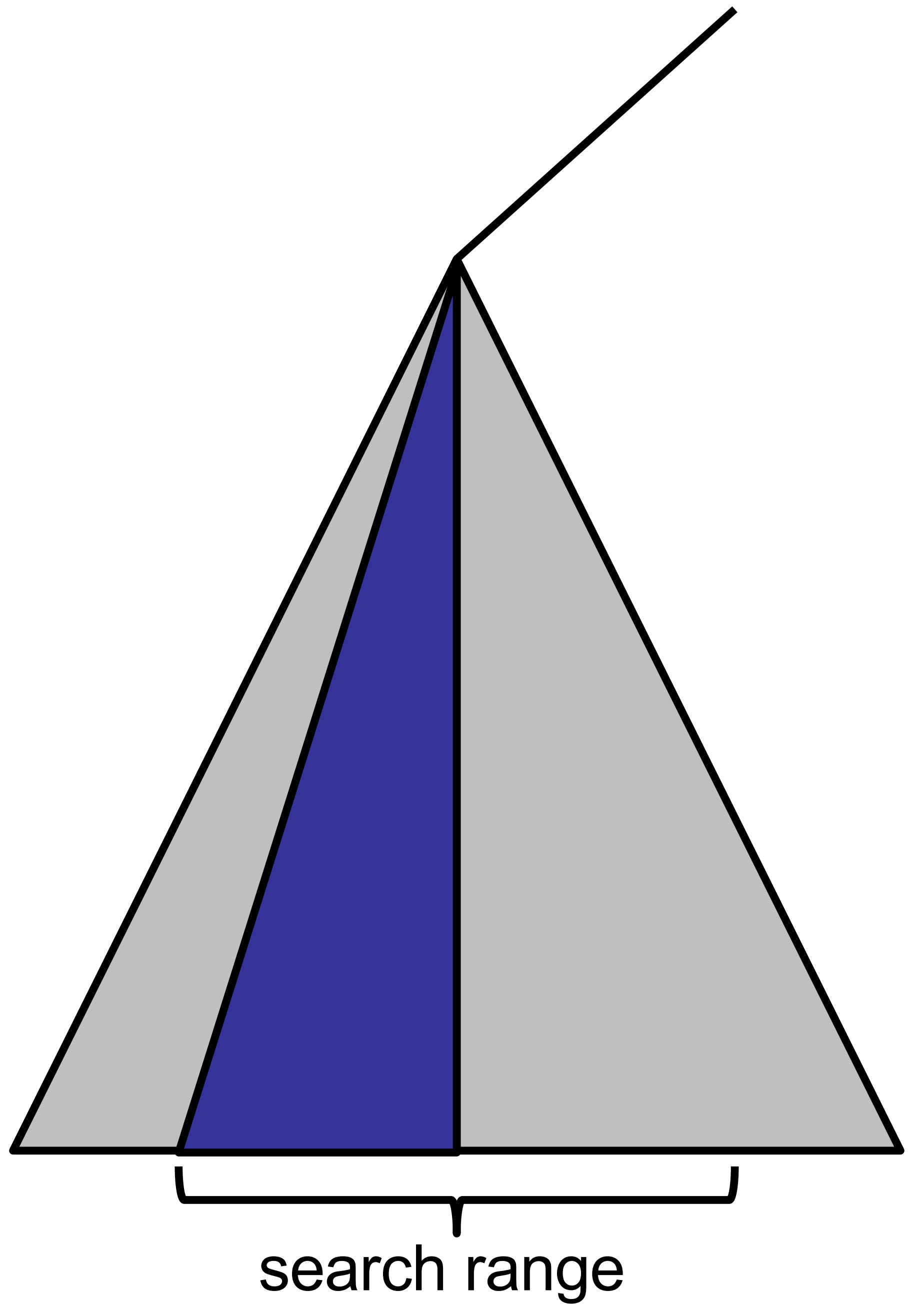
- Find “split” node.
- Do left traversal.
- Do right traversal.



One Dimensional Range Queries

Algorithm:

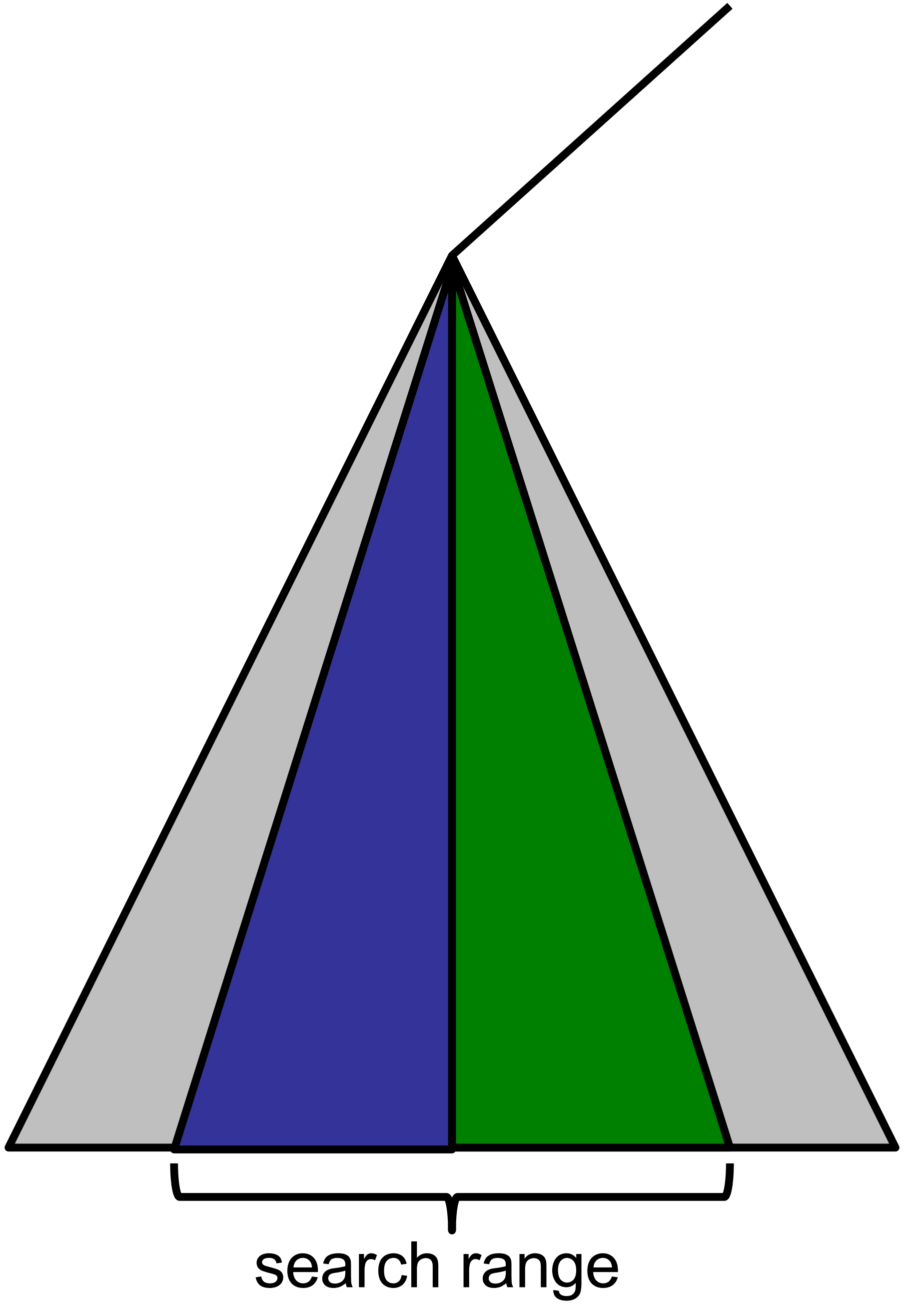
- Find “split” node.
- Do left traversal.
- Do right traversal.



One Dimensional Range Queries

Algorithm:

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

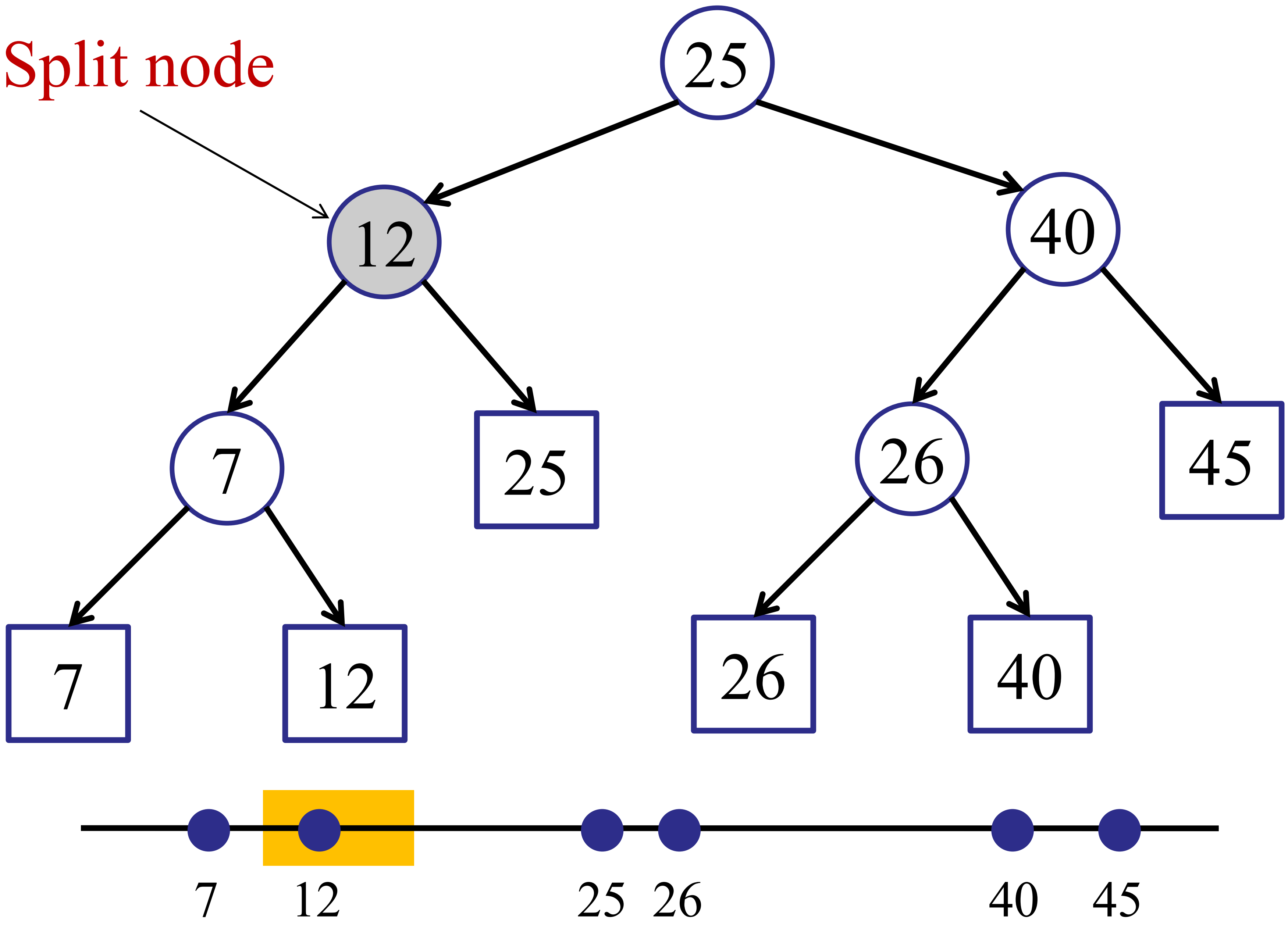


One Dimensional Range Queries

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

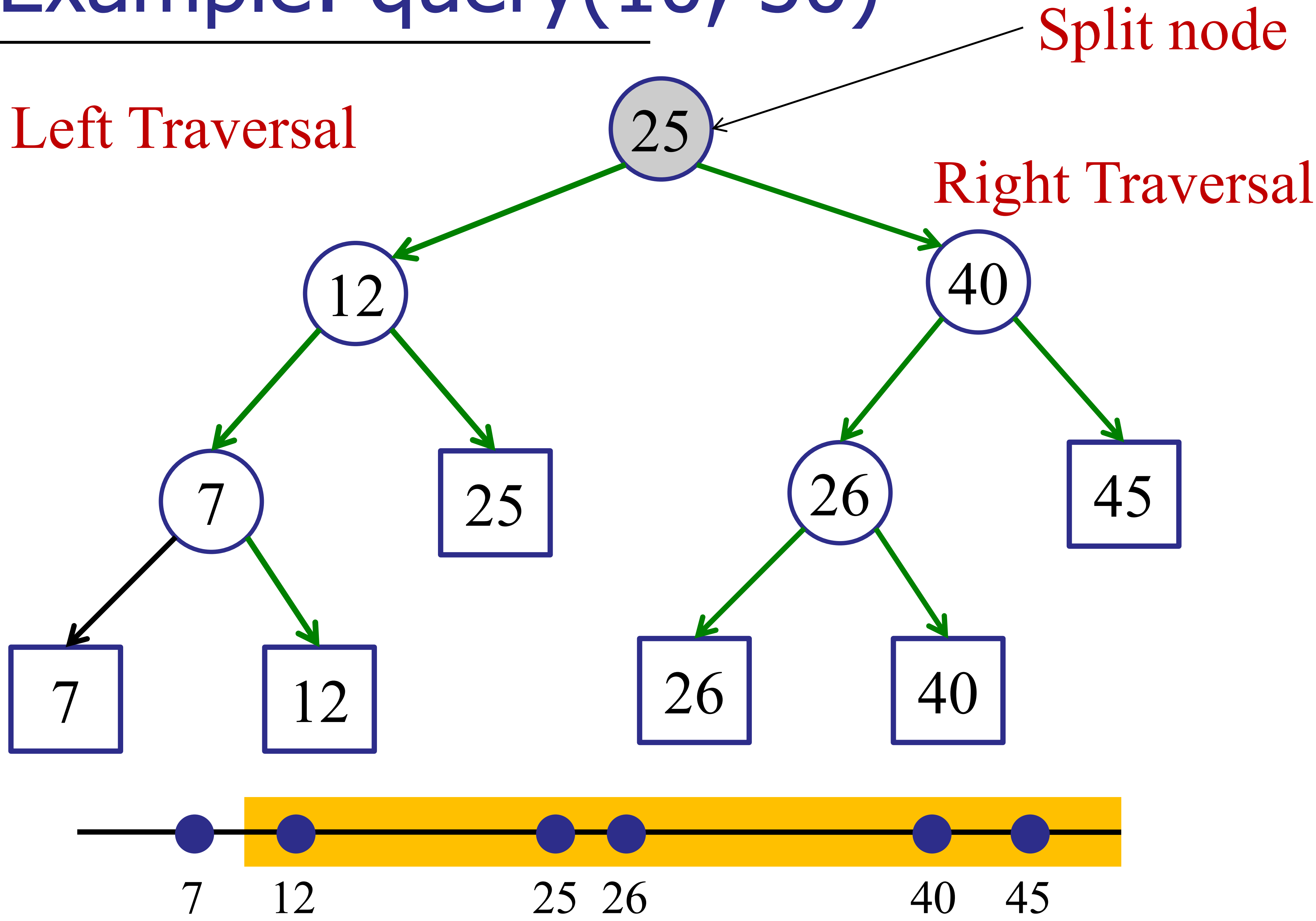
Algorithm:

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

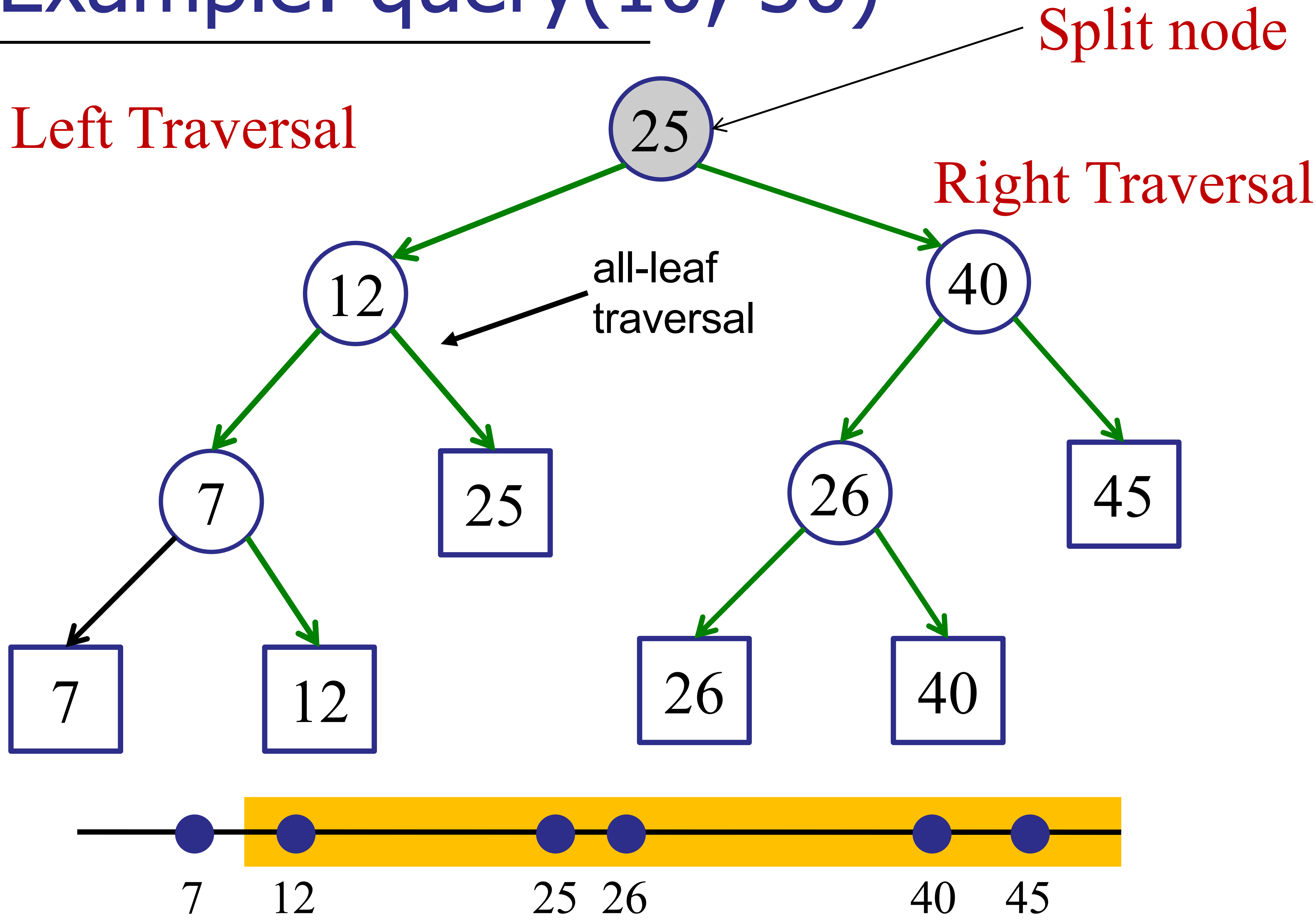
One Dimensional Range Queries

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

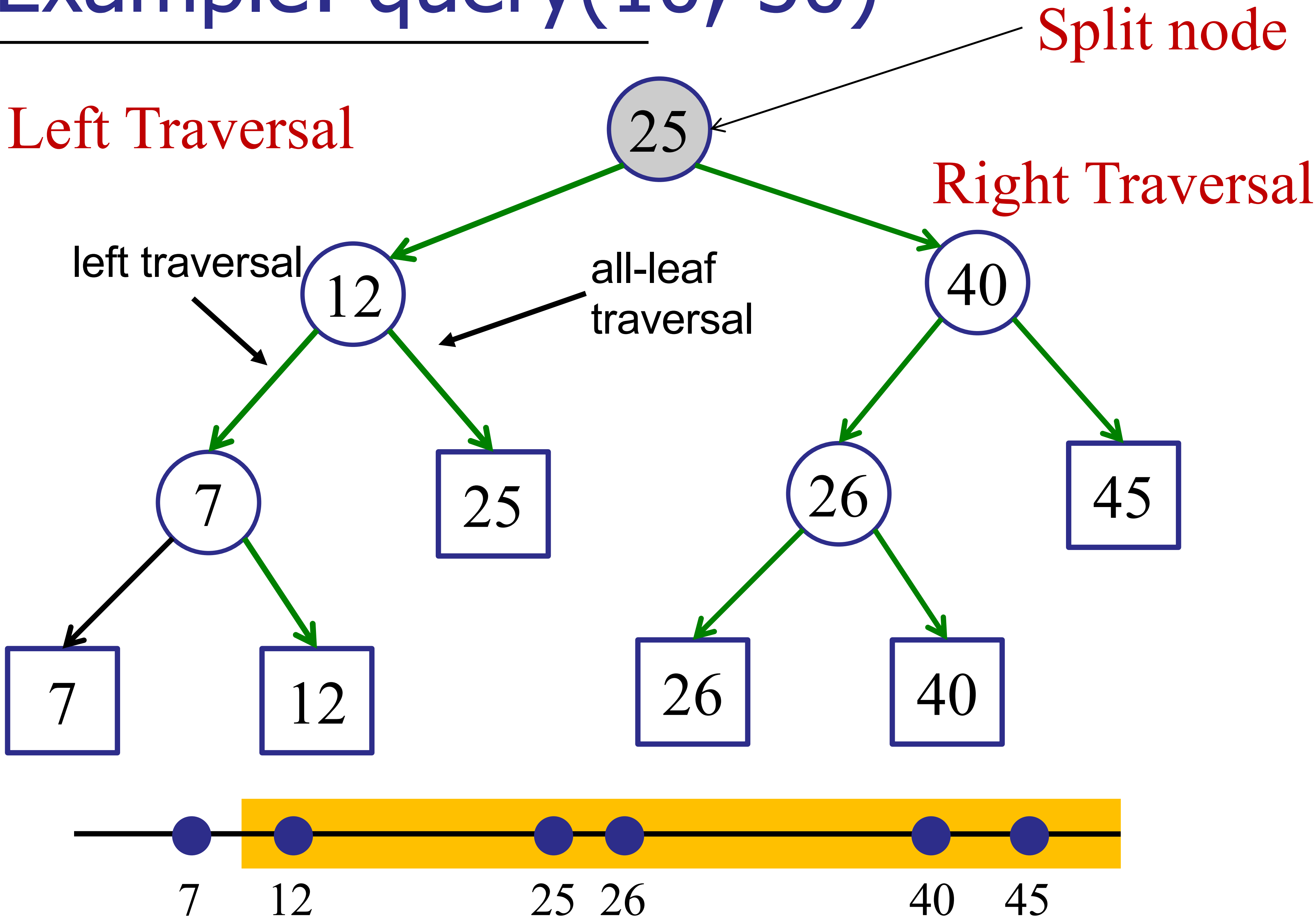
Example: query(10, 50)



Example: query(10, 50)



Example: query(10, 50)

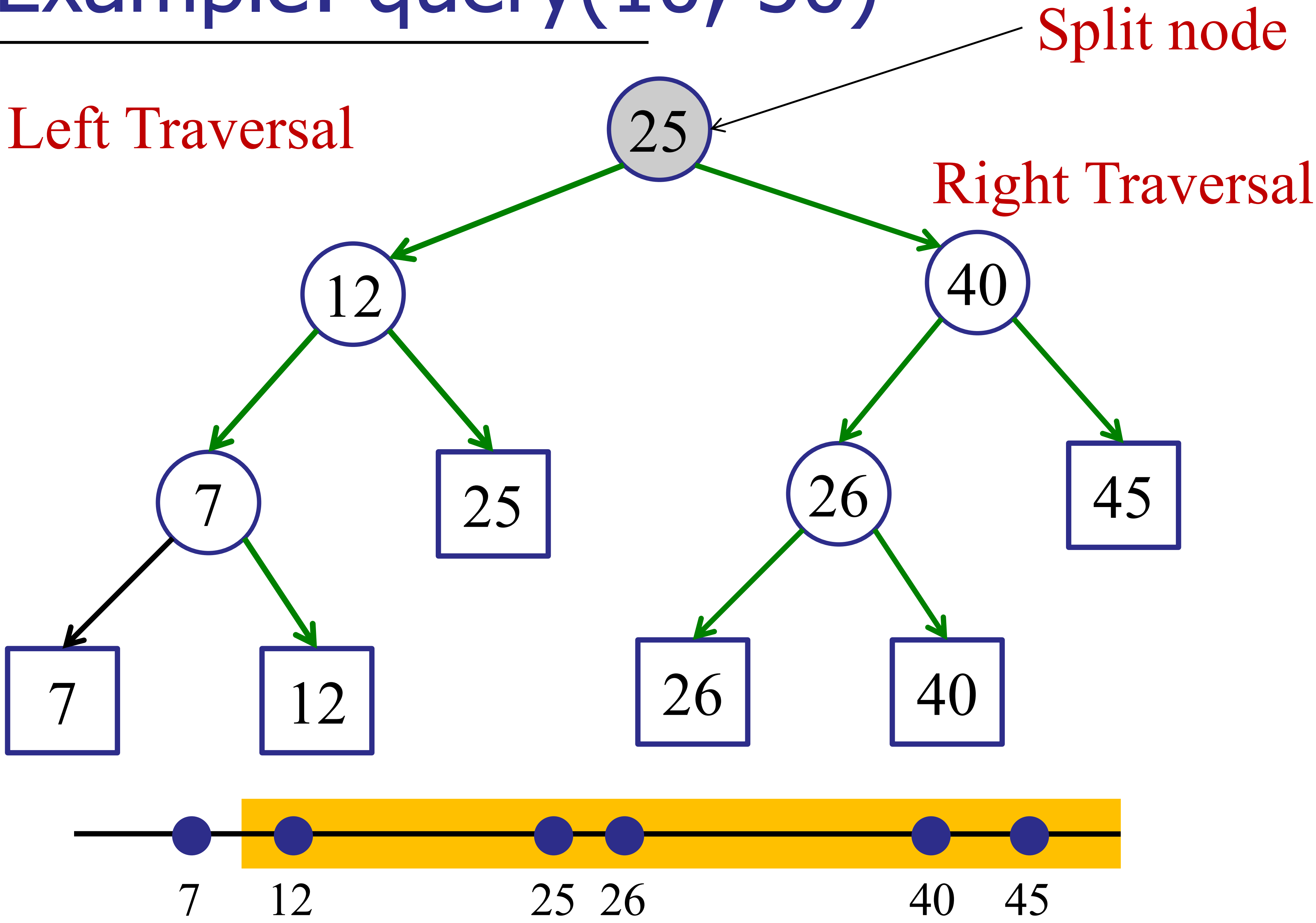


One Dimensional Range Queries

Invariant:

The search interval for a left-traversal at node v includes the maximum item in the subtree rooted at v.

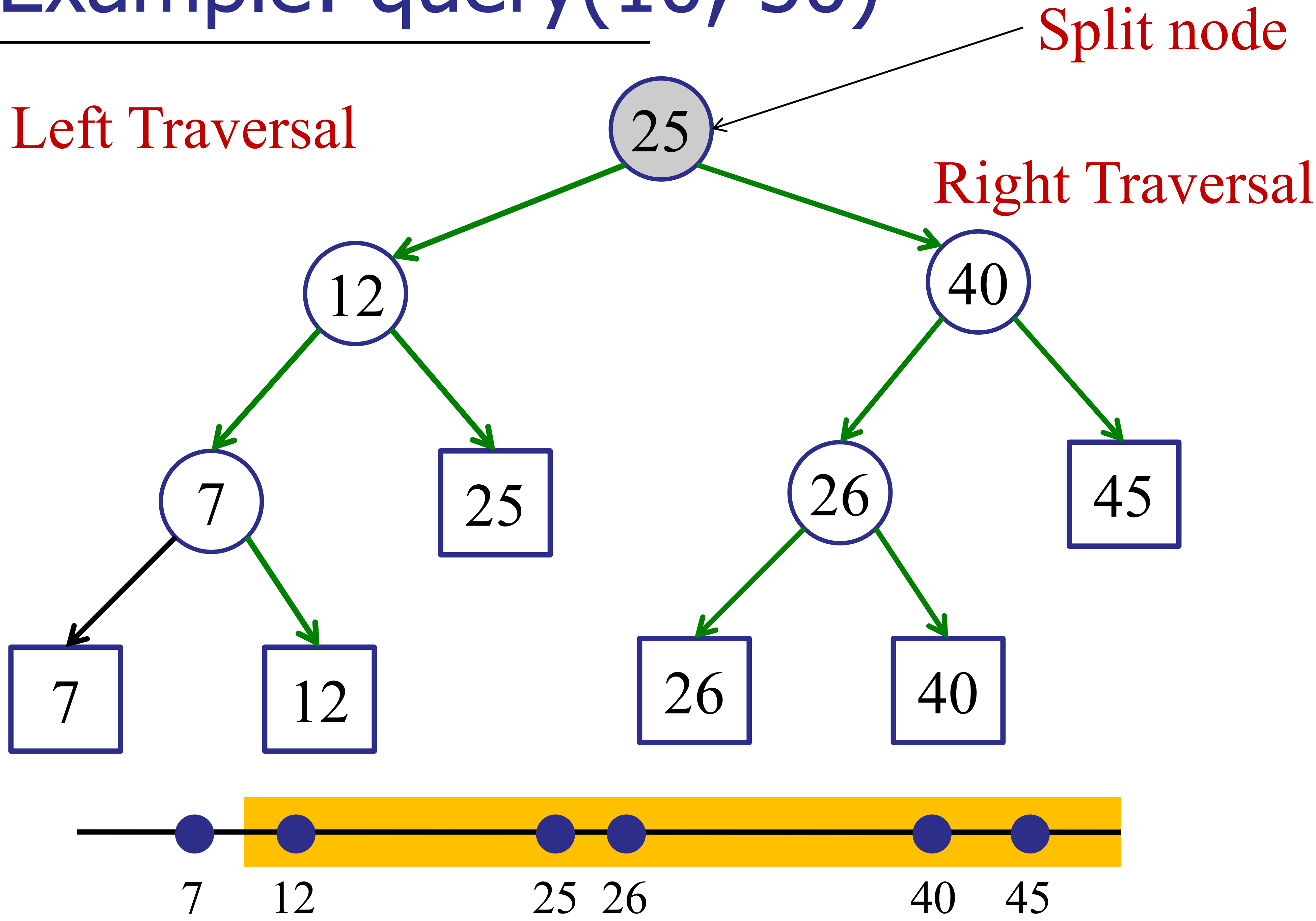
Example: query(10, 50)



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);
    }
}
```

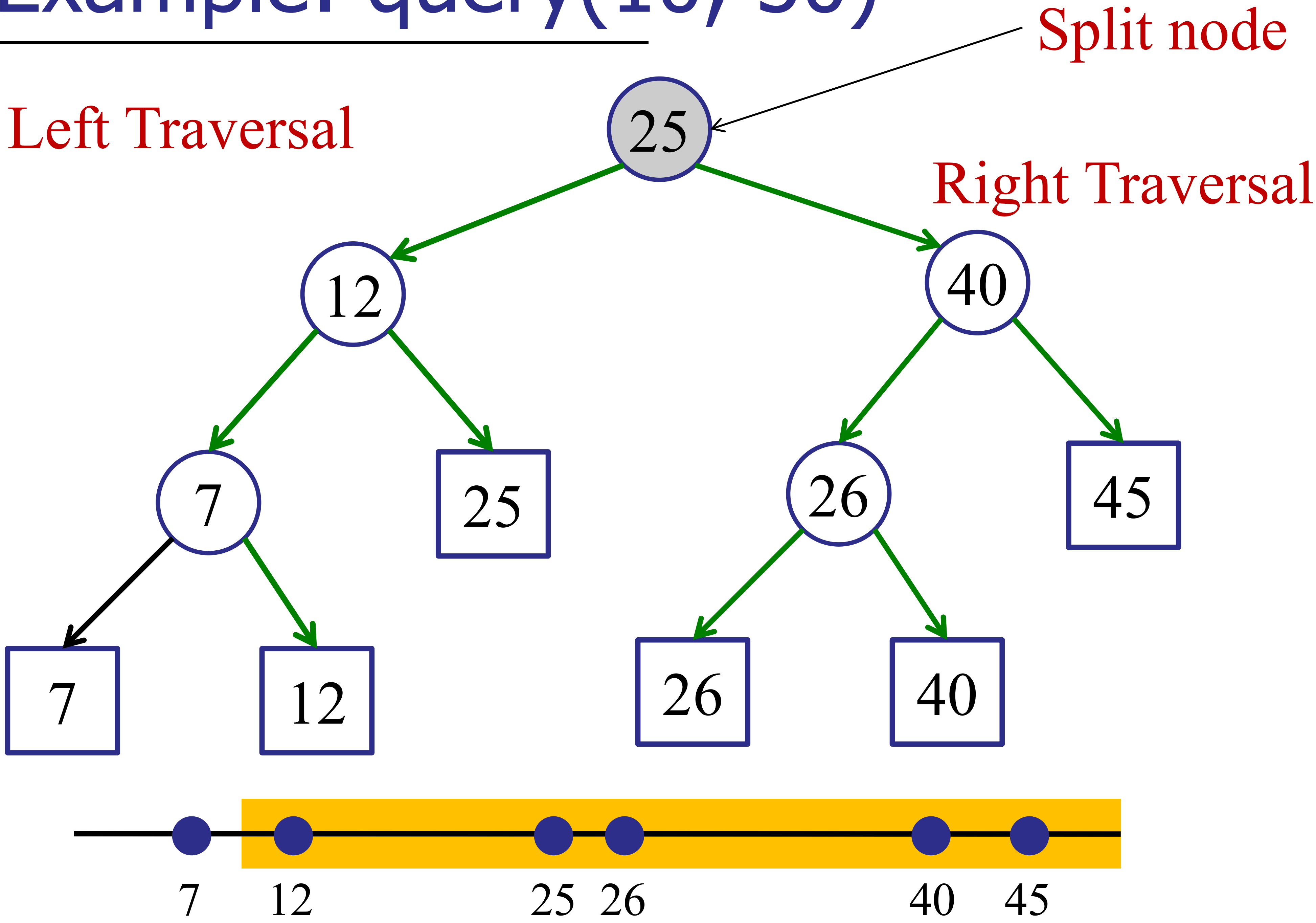
Example: query(10, 50)



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:

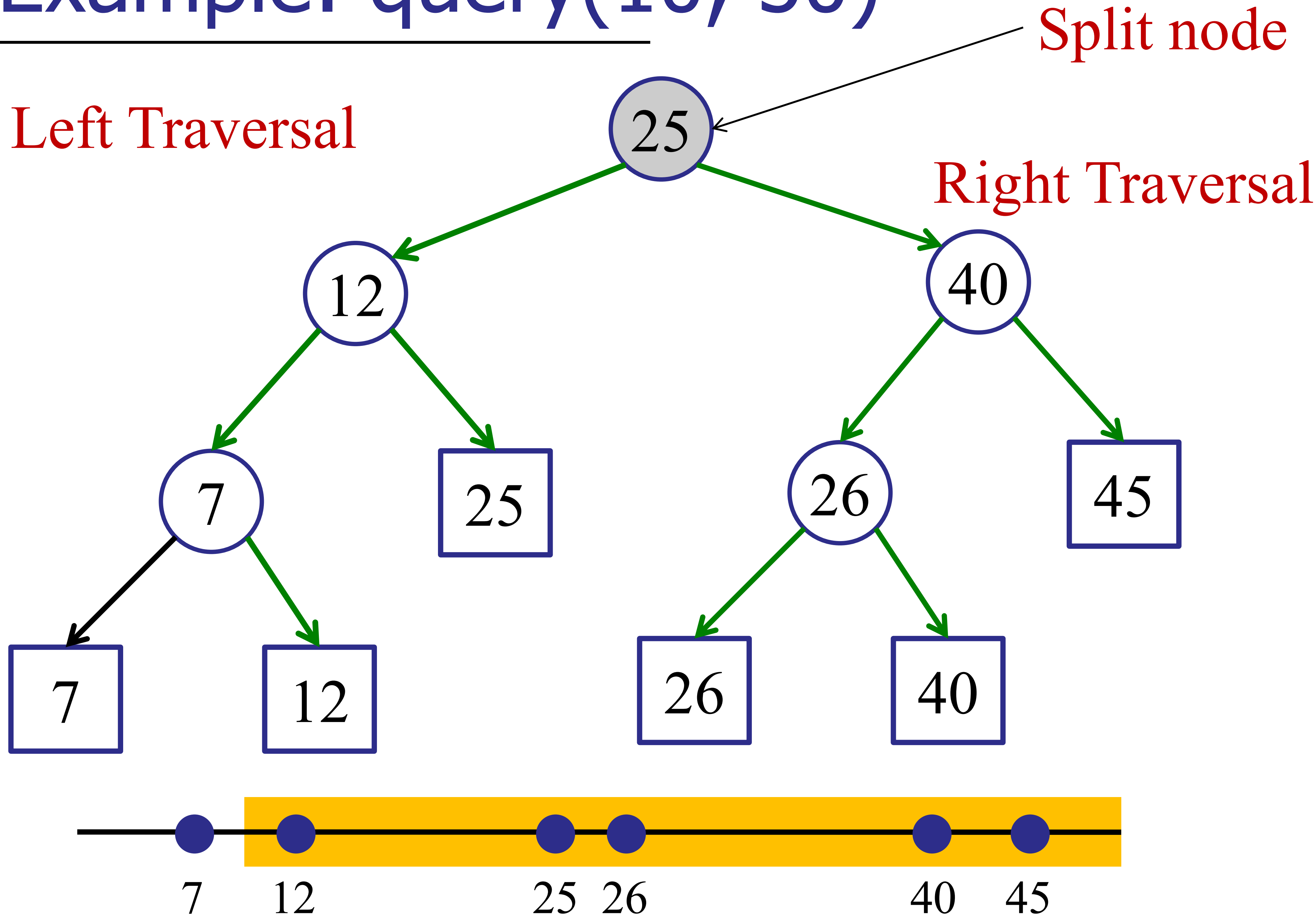
At every step, we either:

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

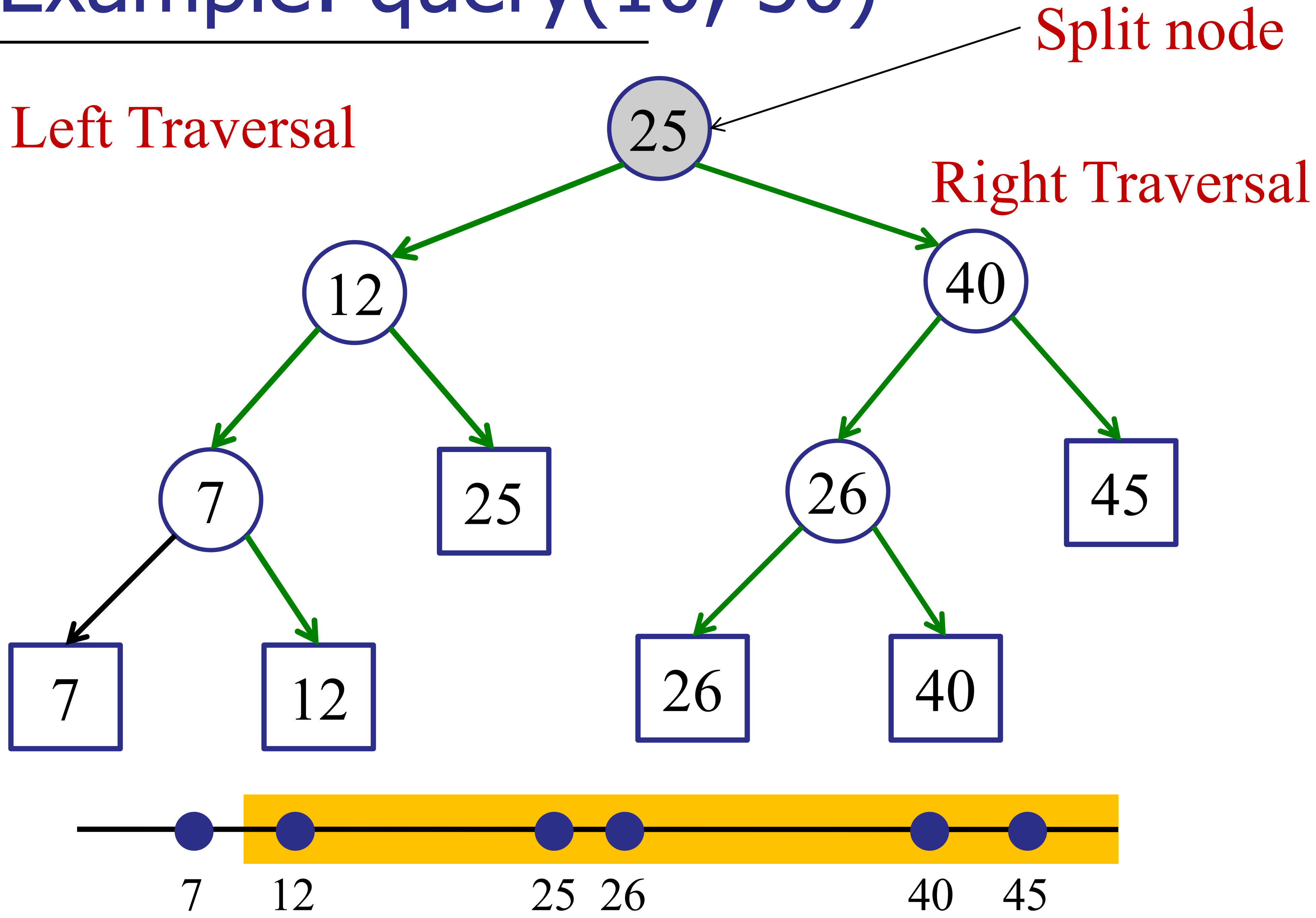
What if you just want to know *how many* points are in the range?

- 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)



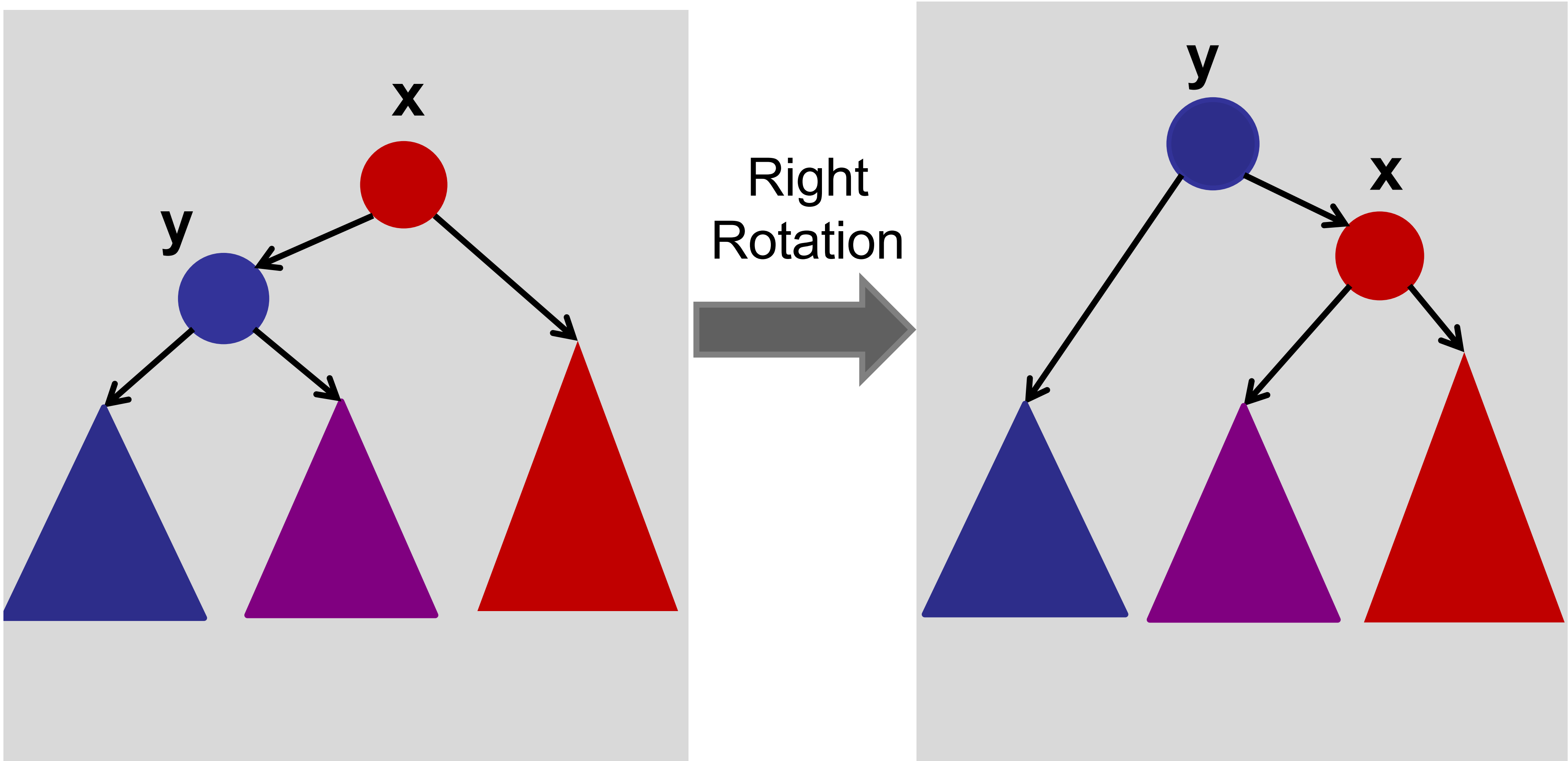
1D Range Tree

Done??

One Dimensional Range Queries

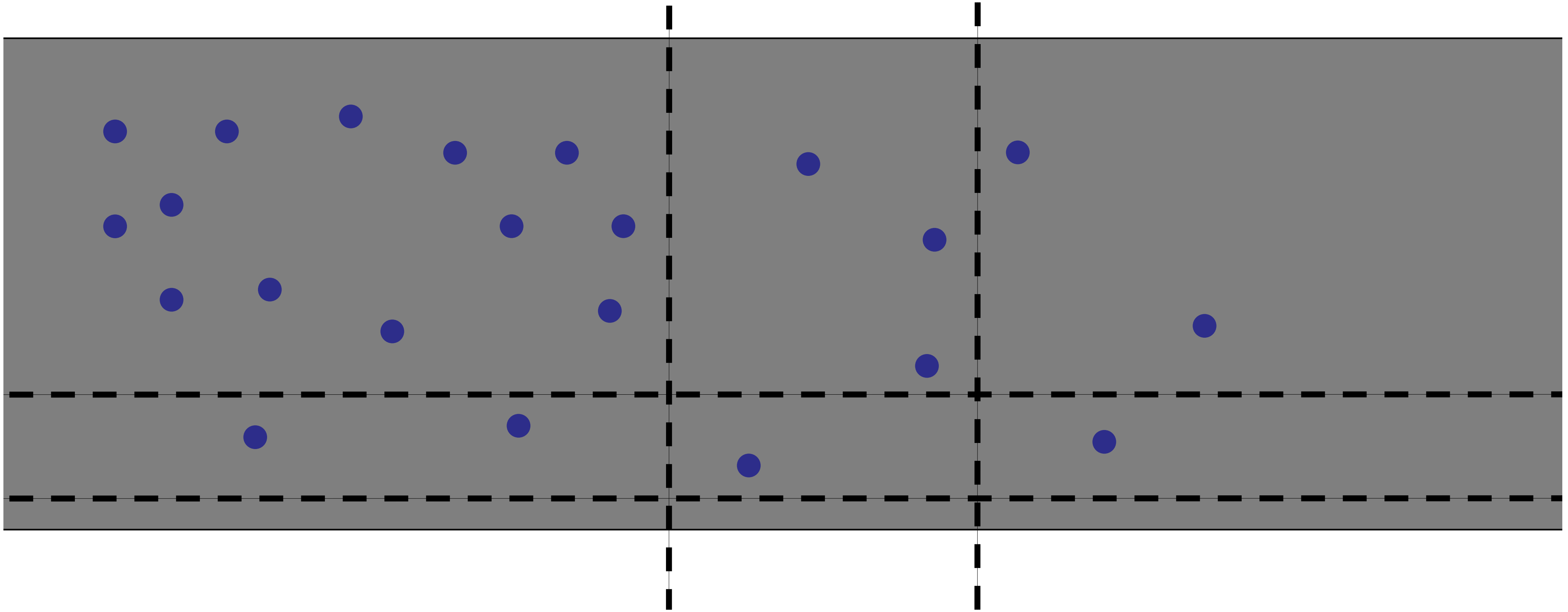
What about dynamic updates?

- Need to fix rotations! Any changes needed?



Two Dimensional Range Tree

Ex: search for all points between dashed lines.



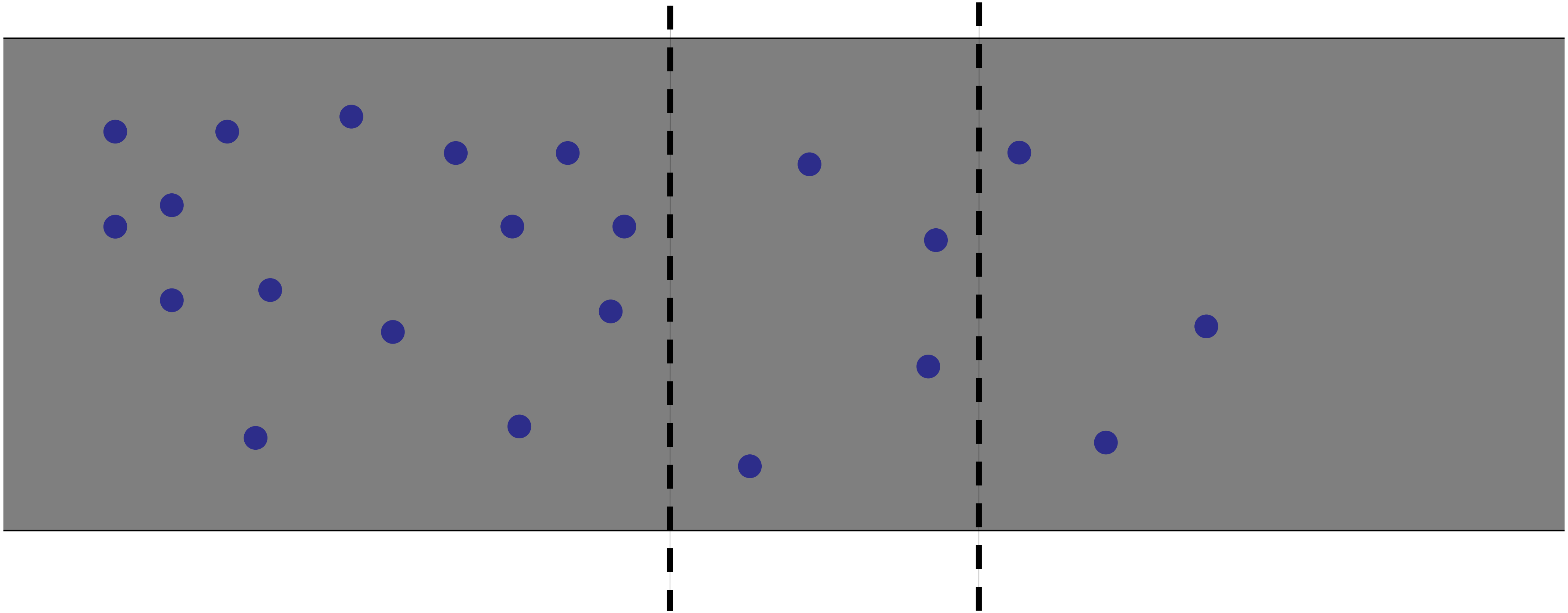
Two Dimensional Range Tree

Idea:

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

Step 2: Enumerate all points in range, sort.

Step 3: Return only points in the y-range.



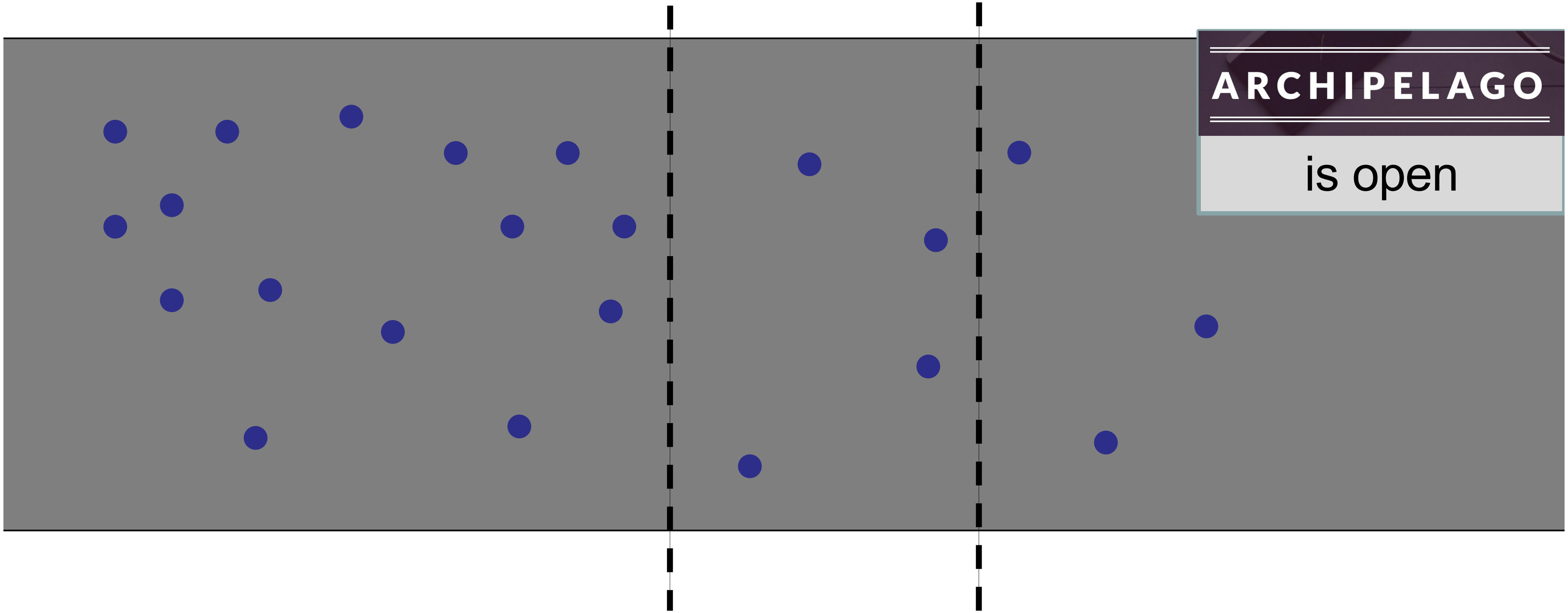
Two Dimensional Range Tree

Idea:

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

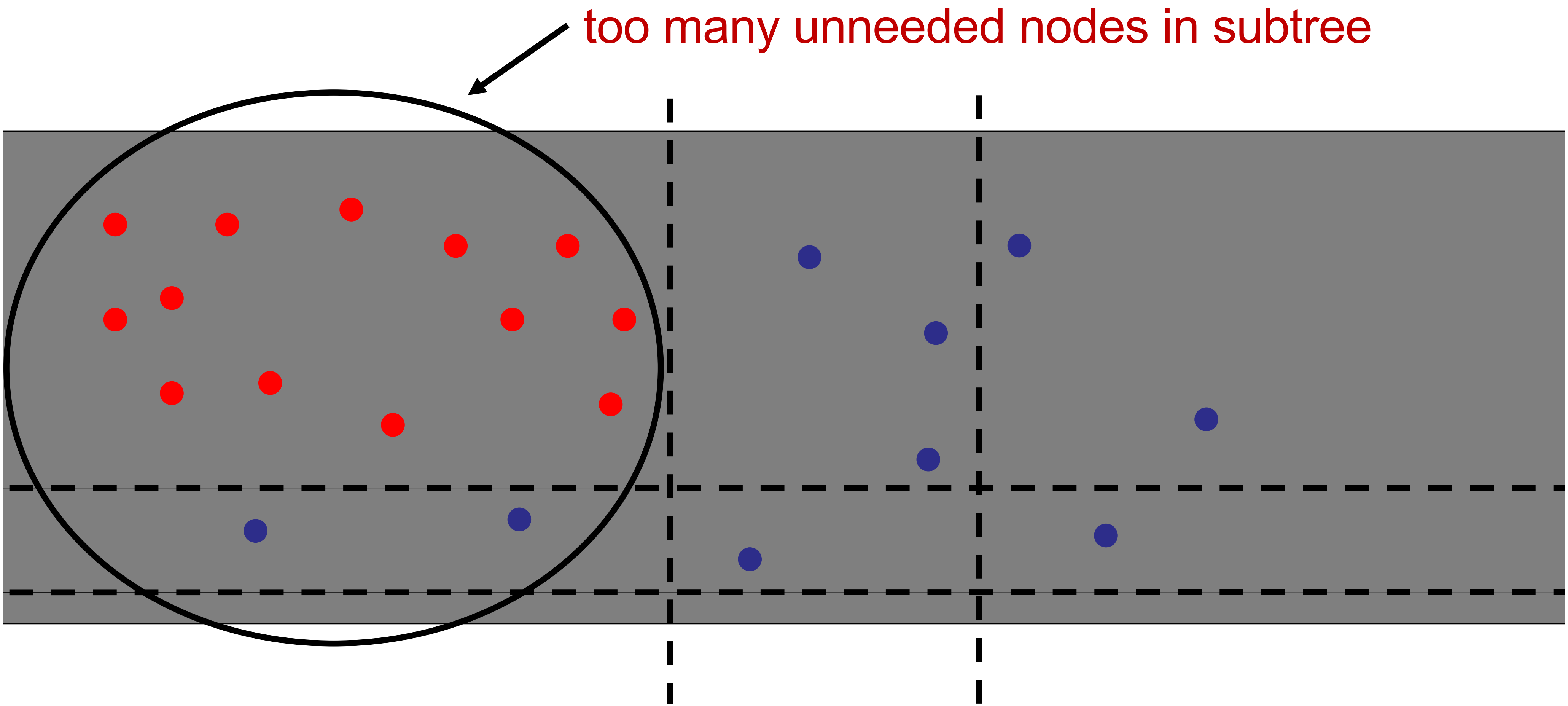
Step 2: Enumerate all points in range, sort.

Step 3: Return only points in the y-range.



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.



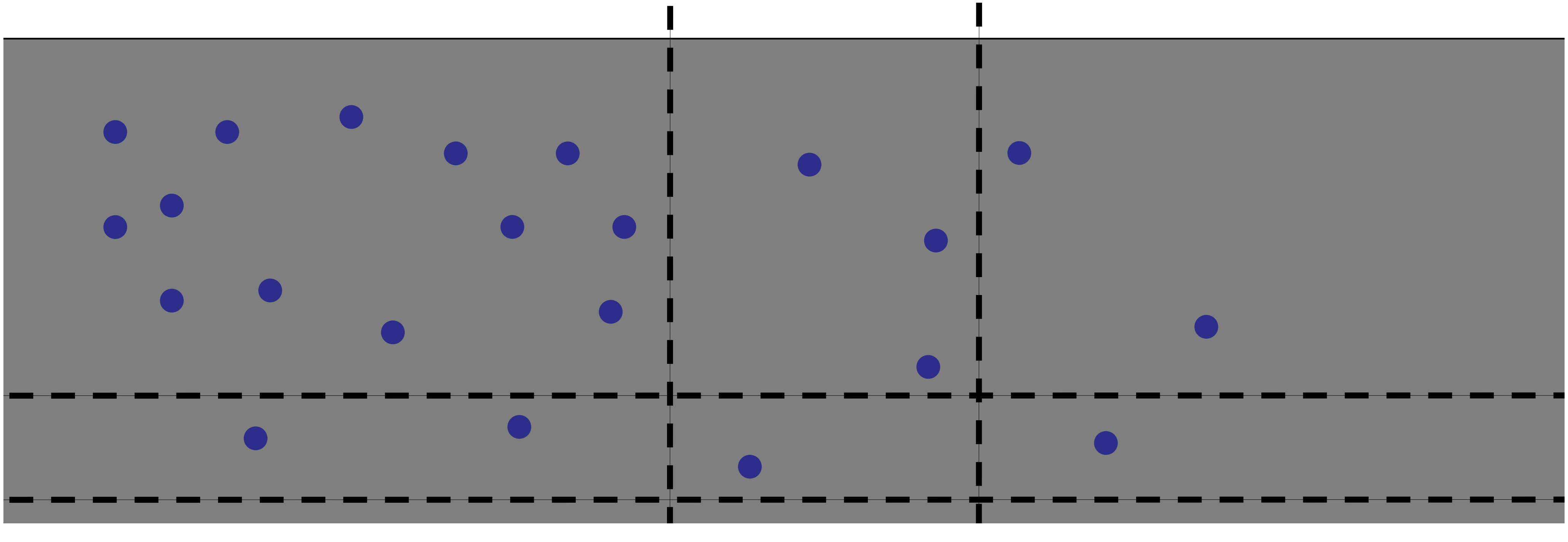
One Dimensional Range Queries

```
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!

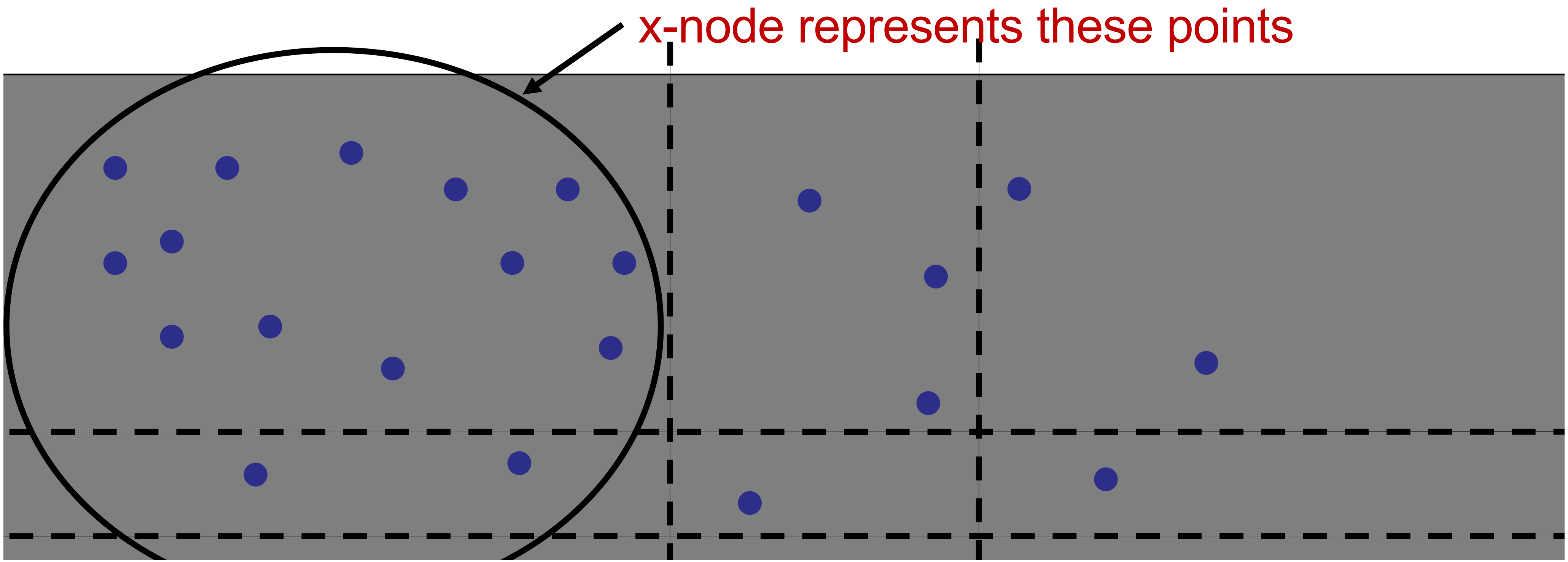
- Each node in the x-tree has a set of points in its sub-tree.
- Store a y-tree at each x-node containing all the points in the sub-tree.



Two Dimensional Range Tree

Solution: Augment!

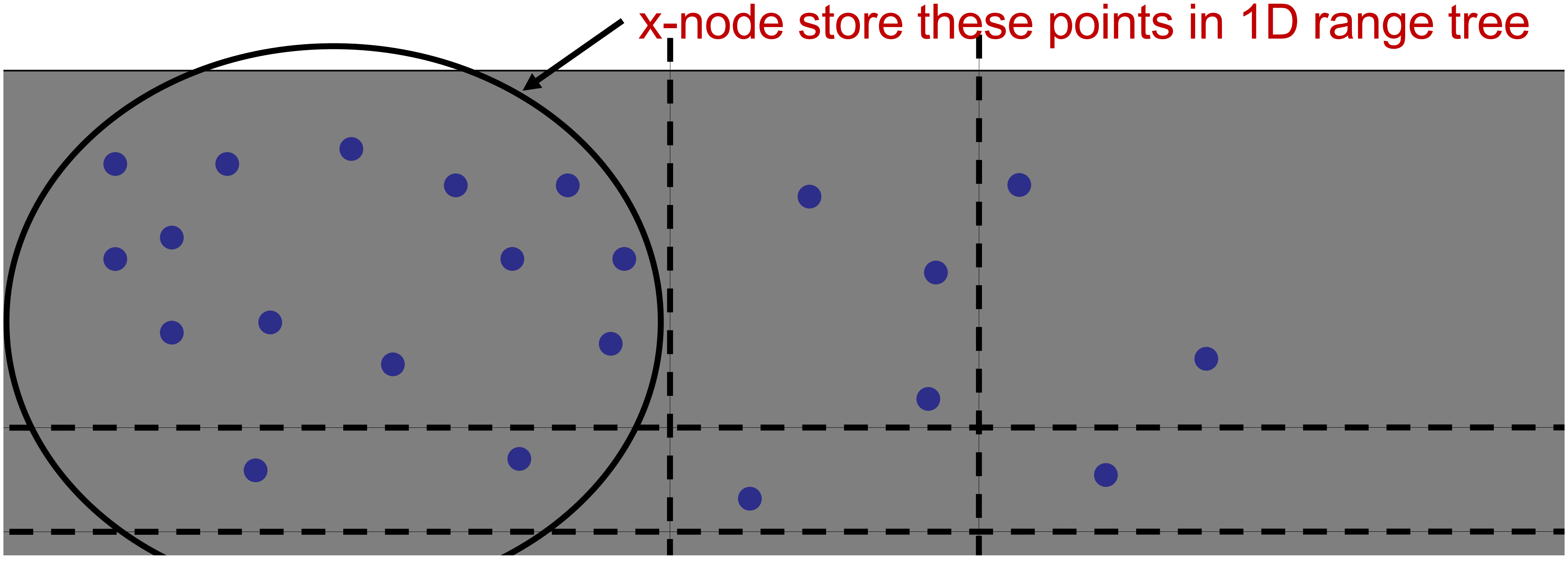
- Each node in the x-tree has a set of points in its sub-tree.
- Store a y-tree at each x-node containing all the points in the sub-tree.



Two Dimensional Range Tree

Solution: Augment!

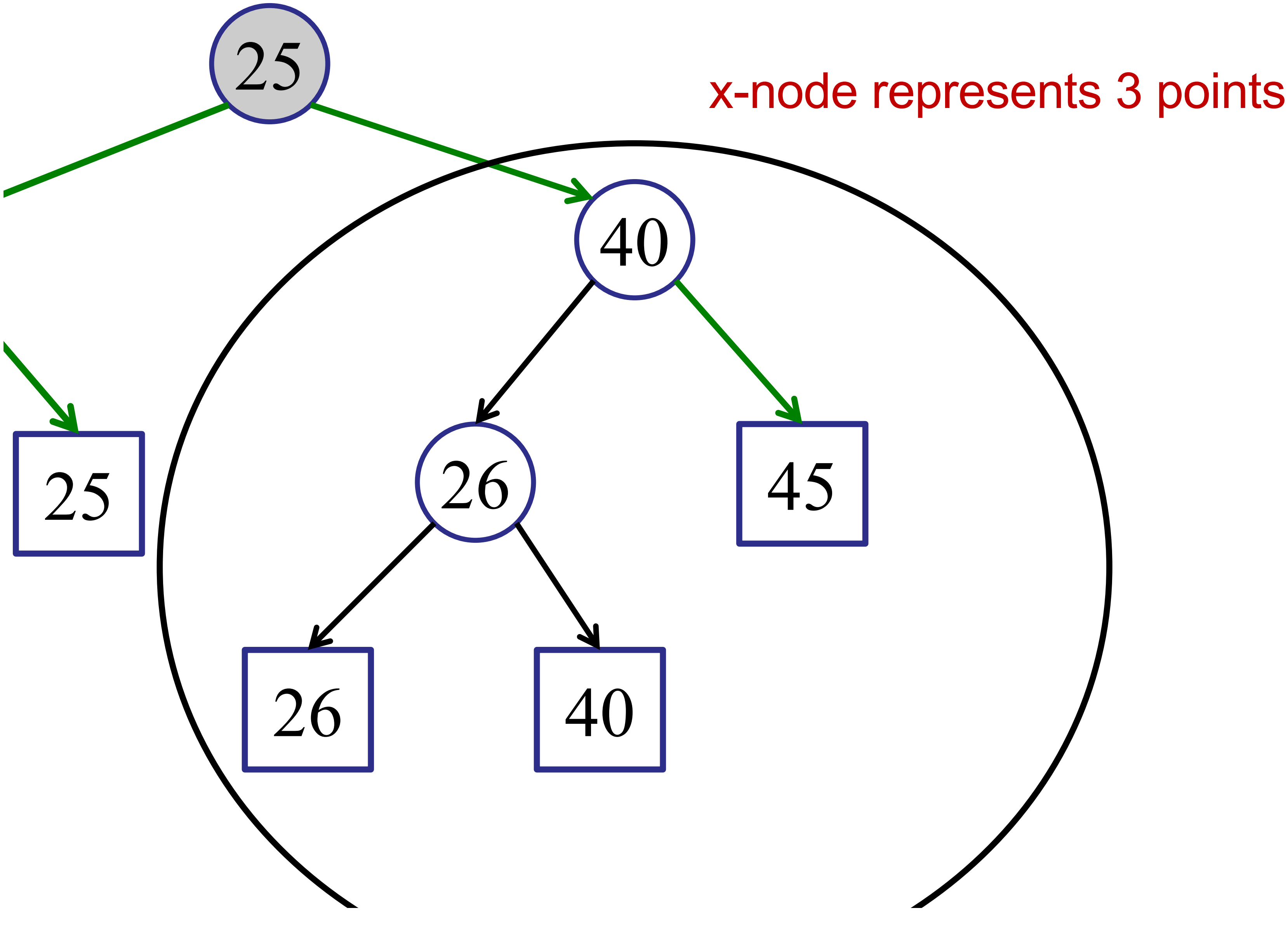
- Each node in the x-tree has a set of points in its sub-tree.
- Store a y-tree at each x-node containing all the points in the sub-tree.



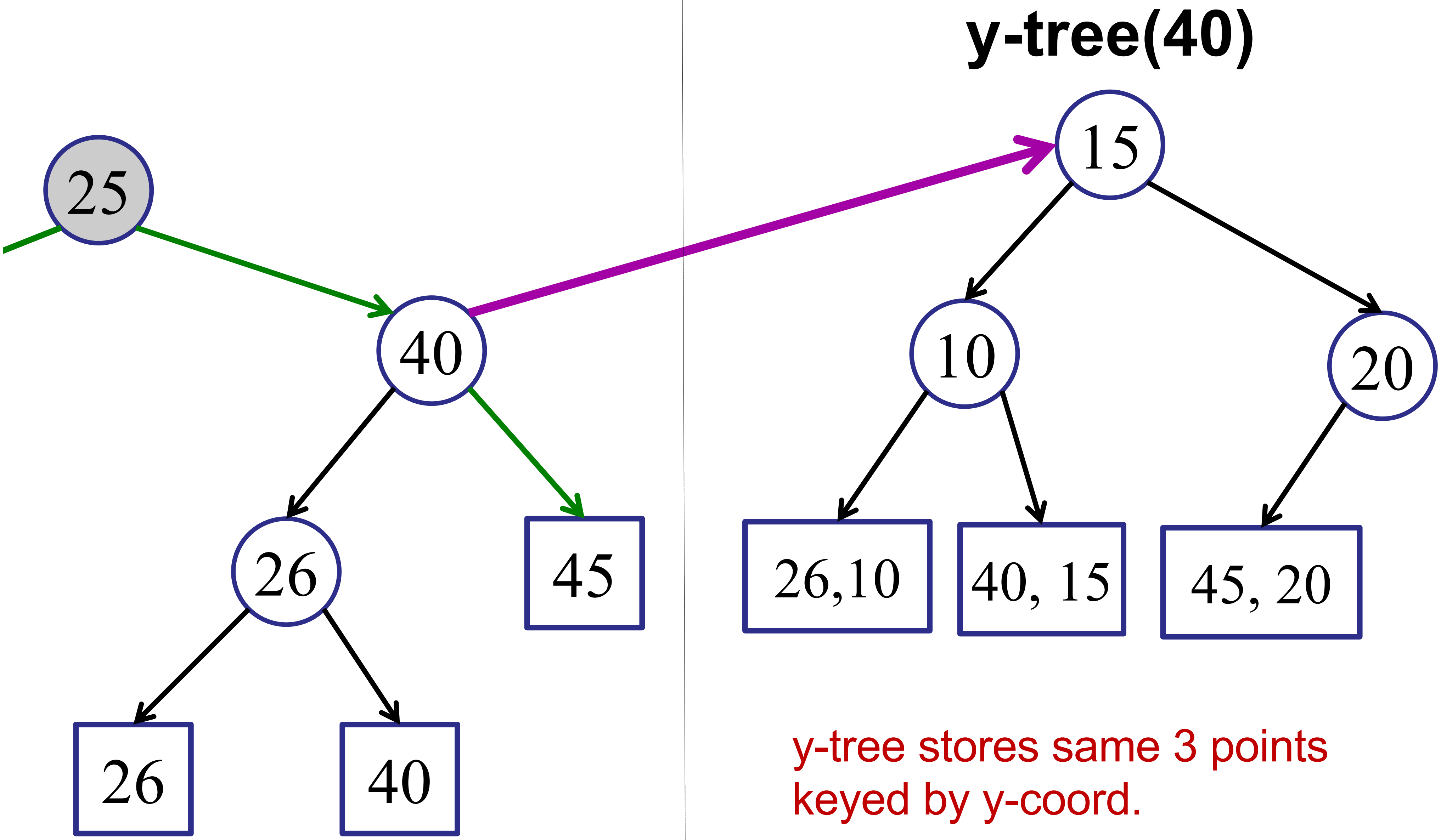
One Dimensional Range Queries

```
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:

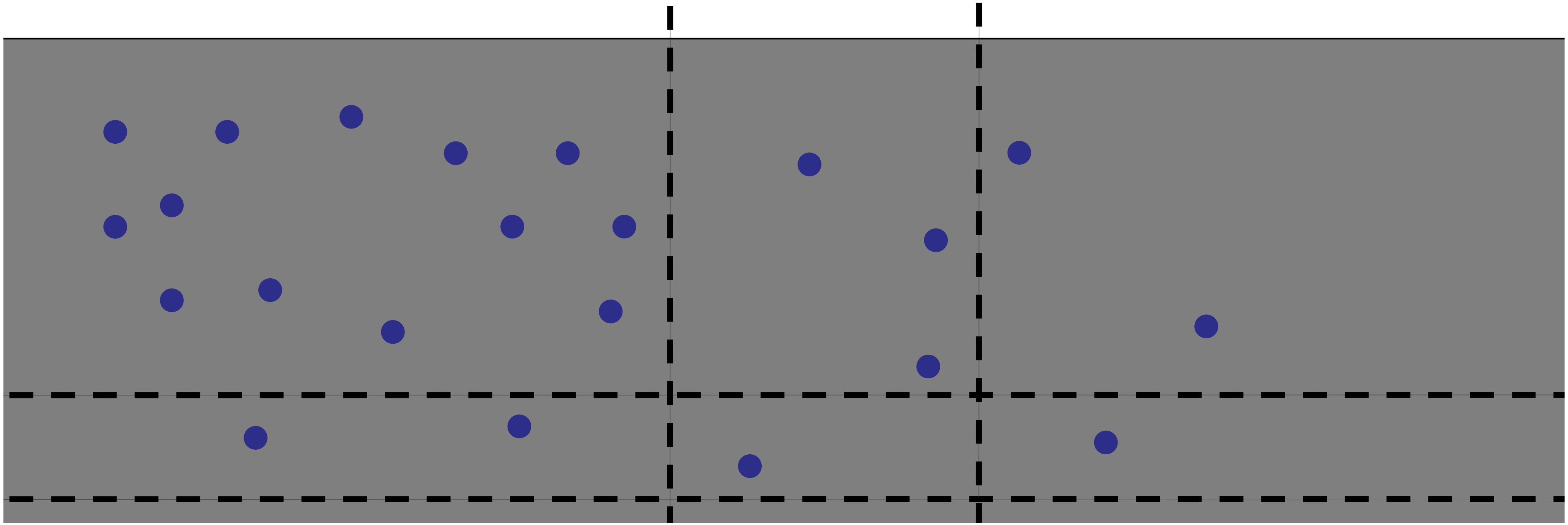


y-tree stores same 3 points keyed by y-coord.

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?

ARCHIPELAGO

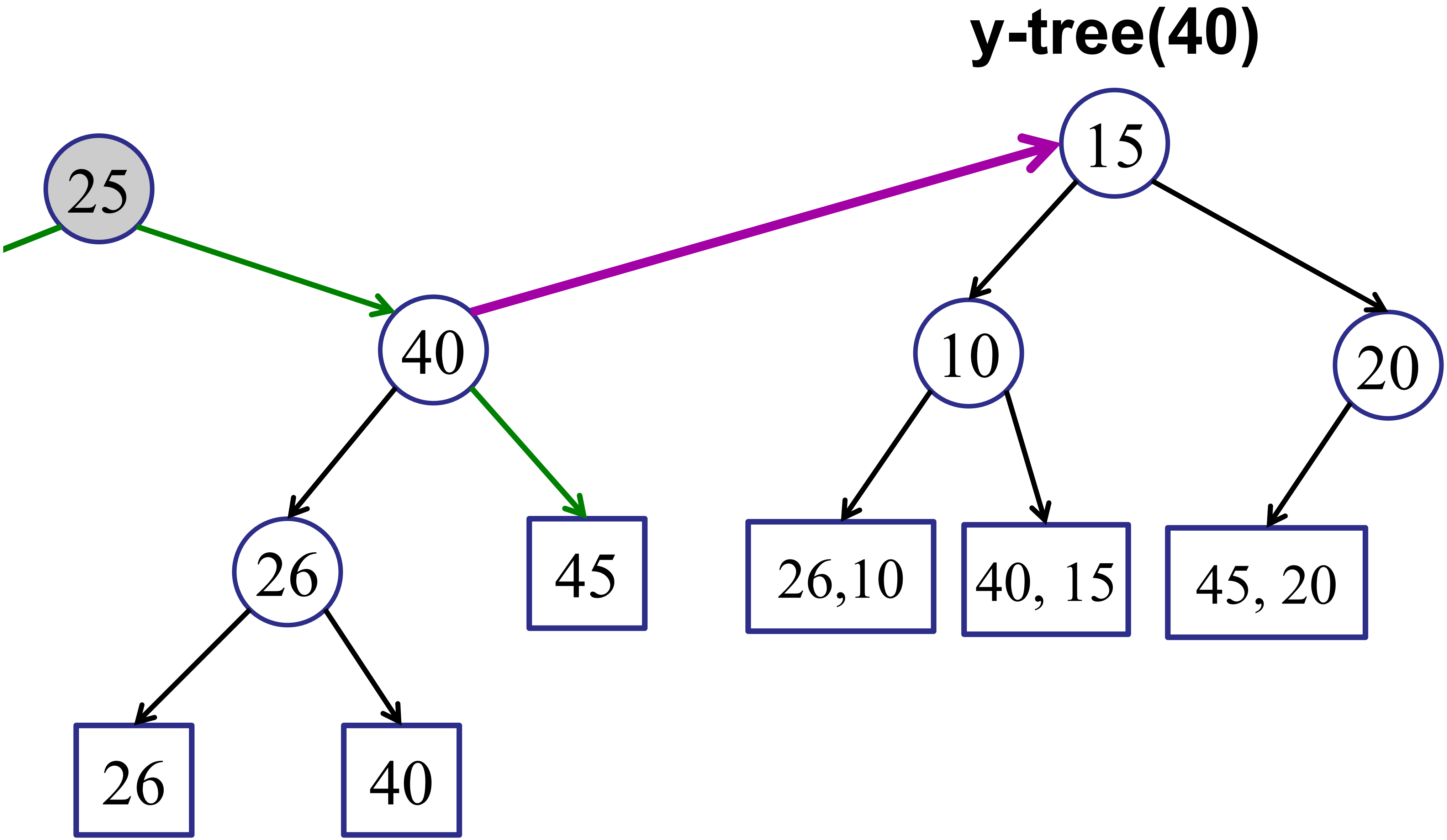
is open

Dynamic Trees

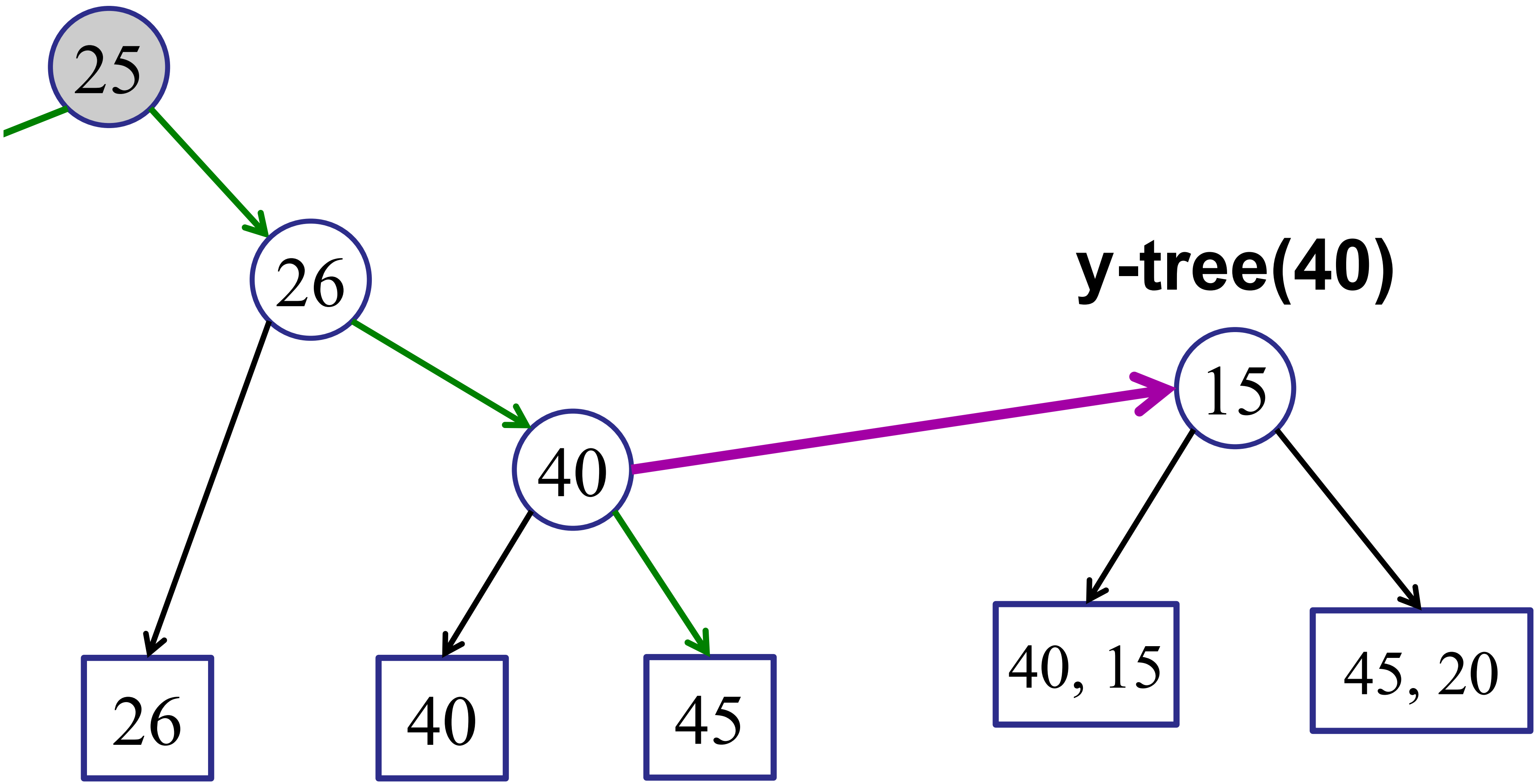
What about inserting/deleting nodes?

- 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:



Example:



Dynamic Trees

Moral:

- Static 2d-range trees support efficient operations.
- We do not support insert/delete operations in 2d-range trees because rotations would be too expensive.

Augmenting data structures has to be done carefully!

Many useful augmentations cannot be supported efficiently!

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.
- Supports more efficient updates
- Often better in practice.
- Good for a variety of queries (e.g., nearest neighbor).

Augmented BSTs

Three examples of augmenting BSTs

1. Order Statistics

2. Intervals

3. Orthogonal Range Searching