Graduate Algorithms

CS673-2016F-08

Extending Data Structures

David Galles

Department of Computer Science
University of San Francisco

08-0: Dynamic Order Statistics

- Data structure with following operations:
  - Insert / Remove / Find $\Theta(\lg n)$
  - Find nth smallest element
  - Find the rank of any element (is it smallest, second smallest, etc)

How can we do this with red/black trees?

- How to find the rank of any element in a red/black tree
- How to find the nth element in a red/black tree

08-1: Dynamic Order Statistics

- Addinging functionality to red/black trees
  - Finding the nth element in a red/black tree
  - Finding the rank of any element in a red/black tree
- What if we could add some other data to a red/black tree, to make this easier?
  - What should we add?
  - How could we use it?

08-2: Size Field

- Add a "size" field to each node in a red/black tree
  - How can we use this size field to find the nth element in the tree?
  - How can we use this size field to find the rank of any element?
  - Can we maintain the size field, and still have insert/remove/find take time $O(\lg n)$?

08-3: Using Size Field

- To find the kth element in a red/black tree with size field:
  - If # of elements in left subtree $= k - 1$, then root is the kth element
  - if # of elements in left subtree $> k - 1$, then the kth element is the kth element in the left subtree
  - if # of elements in the left subtree $= n < k - 1$, then the kth element is the $(k - (n + 1))$th element in the right subtree

08-4: Updating Size: Insert

- Updating size field on insert:

08-5: Updating Size: Insert

- Updating size field on insert:
  - As we go down the tree looking for the correct place to insert an element, add one to the size field of every node along the path from the root to the inserted element
    - (examples on board)

08-6: Updating Size: Delete

- Updating size field on delete:

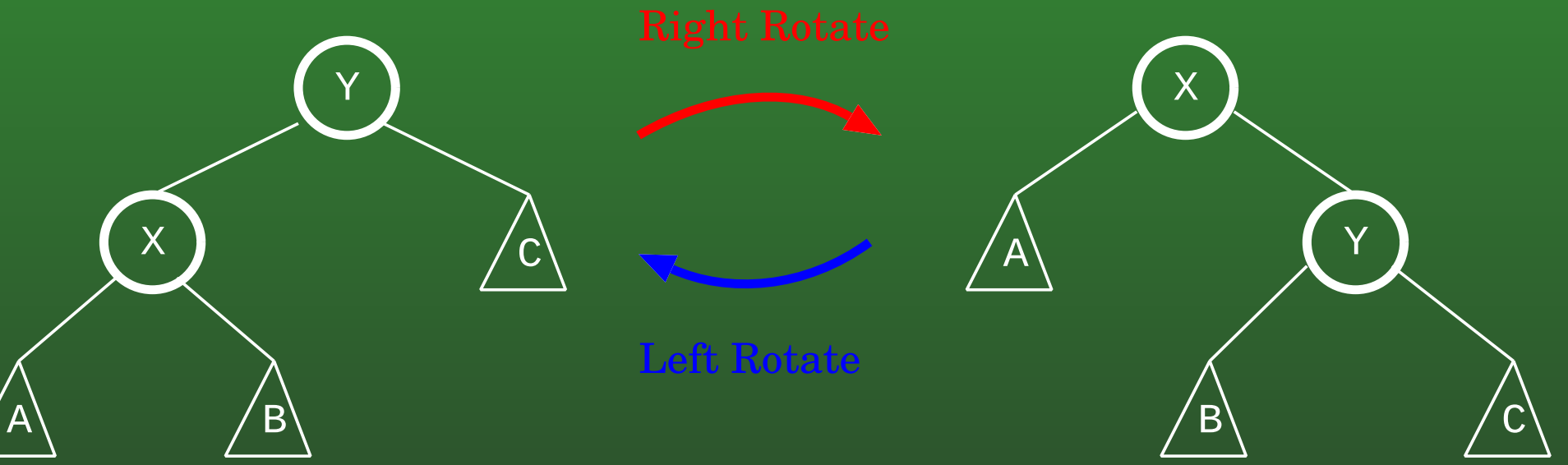
08-7: Updating Size: Delete

- Updating size field on delete:
  - As we go down the tree looking for the element to delete, delete one from the size of evey node along the path from the root to the deleted element
    - (examples on board)
  - Need to be careful about trying to delete elements that aren't in the tree

08-8: Updating Size Field: Rotate

- Updating size on rotations
  - How should sizes be updated on rotations?
    - (Picture on board)

08-9: Updating Size Field: Rotate



zes of A,B,C not changed
ze(X) = Size(A) + Size(B) + 1
ze(Y) = Size(X) + Size(C) + 1

Sizes of A,B,C not changed
Size(Y) = Size(B) + Size(C) + 1
Size(X) = Size(A) + Size(Y) + 1

08-10: Augmenting Data Structures

- Decide what extra information to add to each element of the data structure
- Make sure we can update this extra information for each operation on the data structure
- Add operations that use this extra information
  - New operations
  - Do old operations more efficiently

(Finding rank example)

08-11: Augmenting Data Structures

- For Red/Black trees:
  - If extra information in a node is dependent only on the node itself, and values in left & right children
  - Then, we can always update this information during insert and delete in time $O(\lg n)$

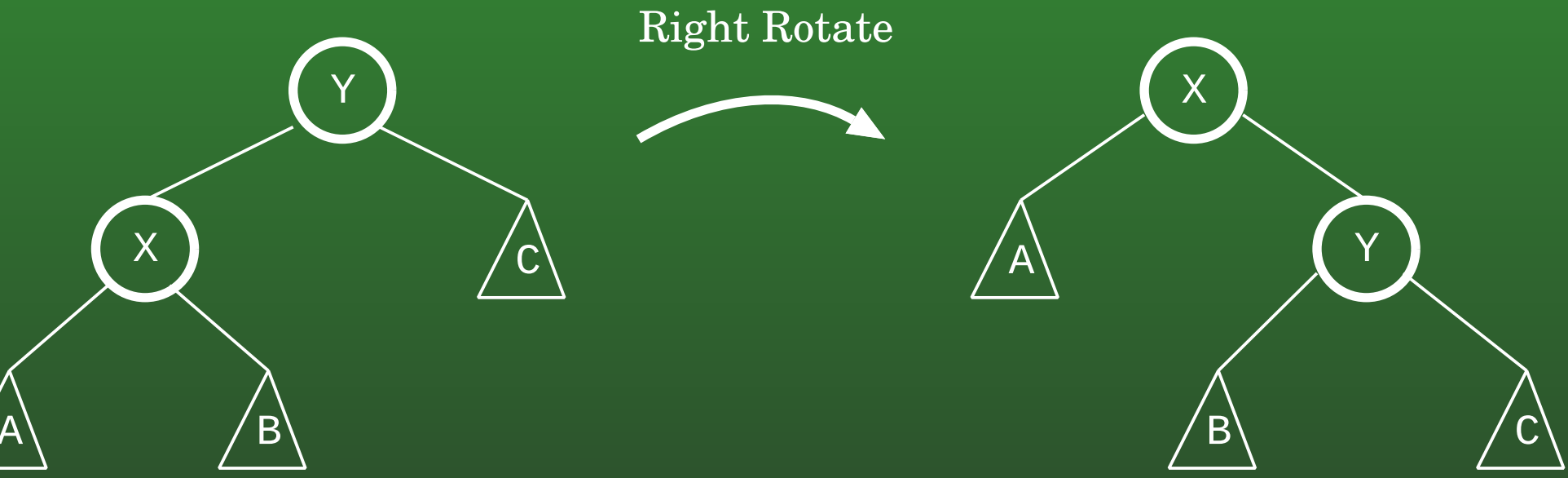
08-12: Augmenting Data Structures

- On an insert:
  - Add the leaf
  - Update information on path from leaf to root after the insertion
    - Extra time: $O(\lg n)$
  - Rotate as necessary

08-13: Augmenting Data Structures

- On a delete:
  - Delete the node
  - Update information on path from deleted node to root after deletion is completed
  - (also works for deletion of node w/ 2 children, do example)
    - Extra time: $O(\lg n)$
  - Rotate as necessary

08-14: Augmenting Data Structures



- Values in A,B,C don't need to change
- Values in X,Y can be changed by looking at A,B,C
- Might need to propagate change up the tree (time $O(\lg n)$)

08-15: Intervals

- Closed intervals
  - $i = [t_1, t_2]$
  - All values between t_1 and t_2, including t_1 and t_2
- Open / Half-Open intervals
  - $i = (t_1, t_2)$, $i' = (t'_1, t'_2]$
  - All values between t_1 and t_2, not including t_1 or t_2
  - All values between t_1 and t_2, including t_2 but not t_1

08-16: Intervals

- Given two intervals $i = [t_1, t_2]$ and $i' = [t'_1, t'_2]$
- Three cases
  - i and i' overlap (including one is a subset of the other)
  - i is to the left of i'
  - i is to the right if i'

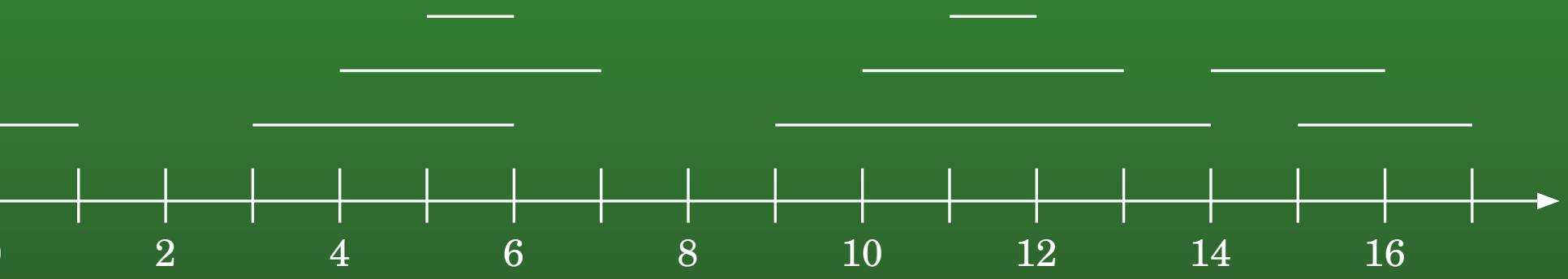
08-17: Intervals

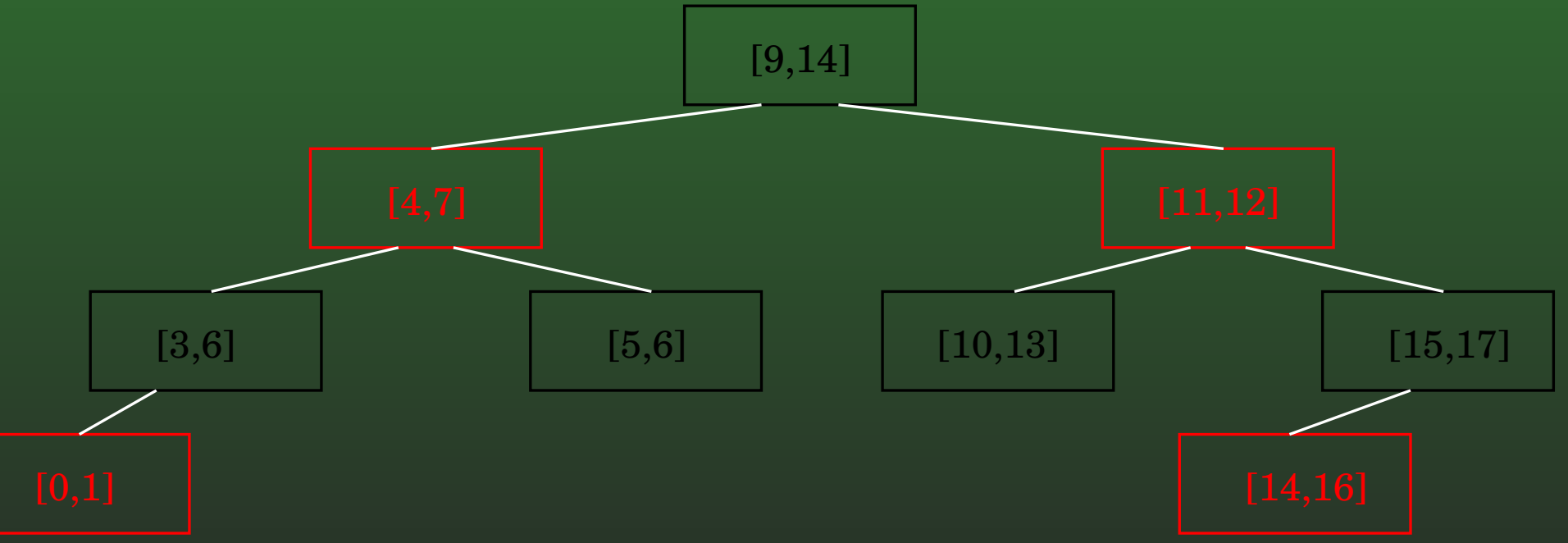
- Data structure that manipulates intervals
  - Insert a closed interval (using endpoints)
  - Delete a closed interval (using endpoints)
  - Find an interval
    - Given an interval i, find an interval that overlaps i
    - If > 1 interval overlaps i, return one arbitrarily

08-18: Interval Trees

- First Try:
  - Each node stores low/high endpoint of interval
  - Sorted based on low endpoint
  - No extra information (yet!)

08-19: Interval Trees

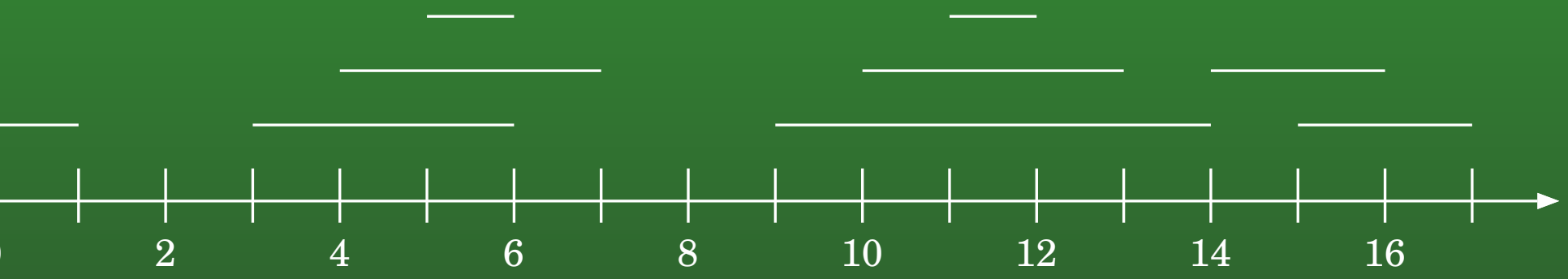


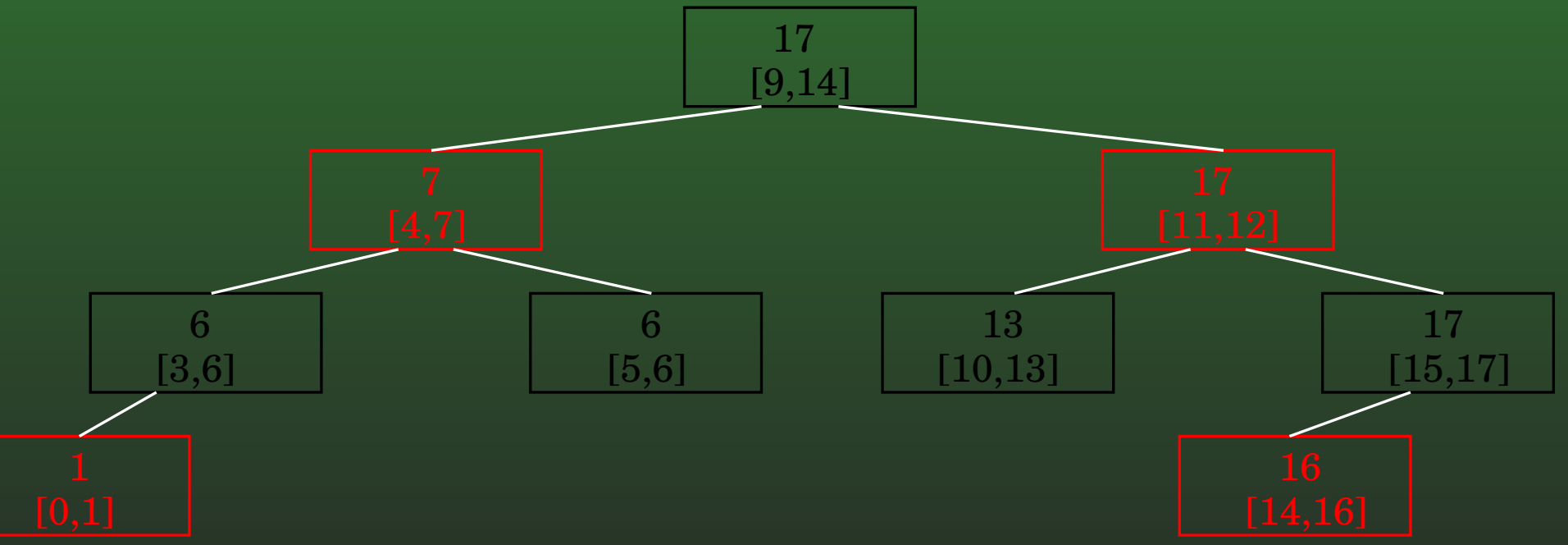


08-20: Interval Trees

- Data Structure
  - Each node stores low/high endpoint of interval
  - Sorted based on low endpoint
  - Extra information: Maximum value of any interval stored in the subtree rooted at this node

08-21: Interval Trees





08-22: Interval Trees

- Extra Information:
  - Maximum value of any interval stored in the subtree
- Can we maintain this extra information through inserts and deletes?

08-23: Interval Trees

- Extra Information:
  - Maximum value of any interval stored in the subtree
- Can we maintain this extra information through inserts and deletes?
  - Yes: Maximum value in a subtree only depends upon:
    - High endpoint stored in current node
    - Maximum value in left subtree
    - Maximum value in right subtree

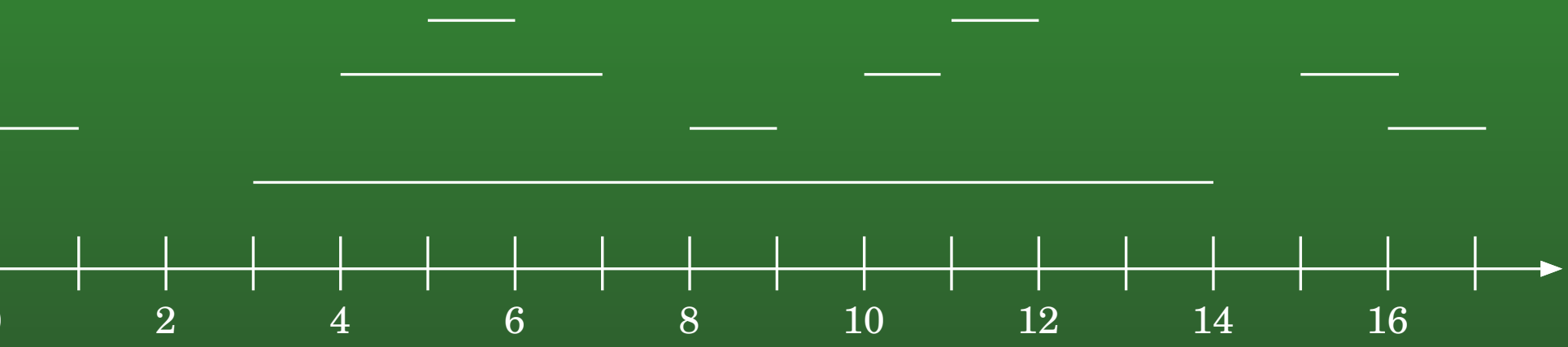
(Example updates on board)

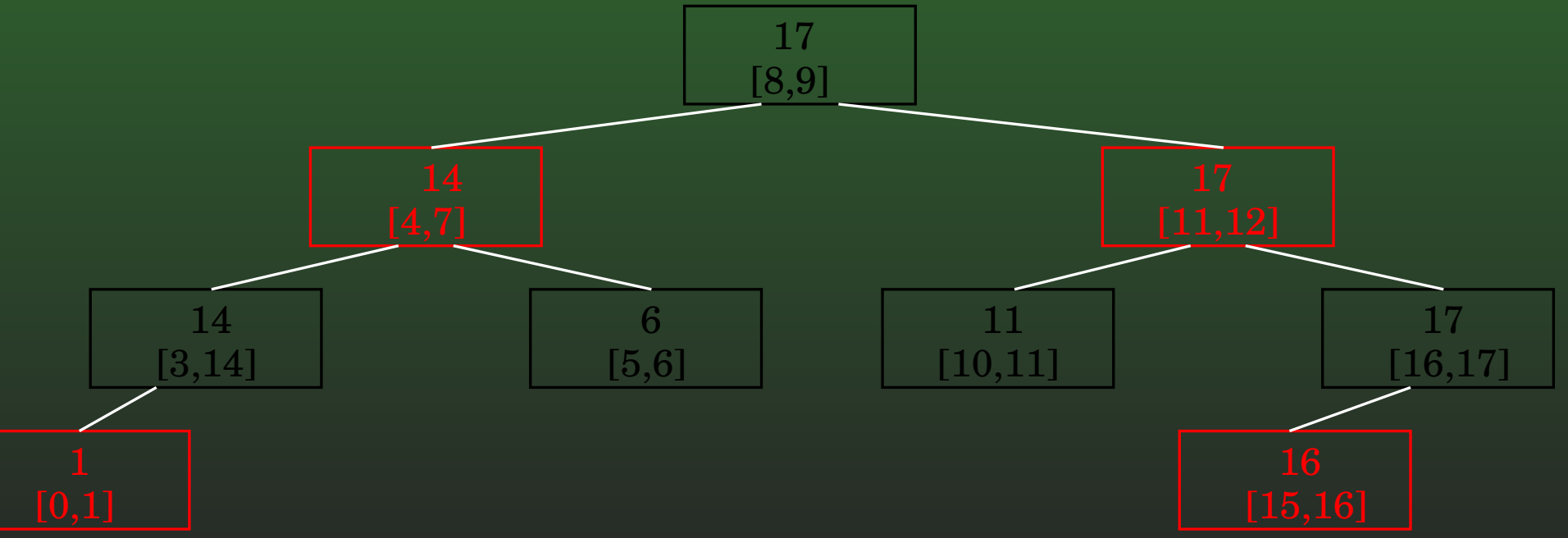
08-24: Interval Trees

- How do we find an interval?
  - Given an interval i, find an interval that overlaps i.
  - If > 1 interval overlaps i, return any one of the intervals that overlap i

(examples in next slide)

08-25: Interval Trees





08-26: Interval Trees

- How do we find an interval?

```
IntervalSearch(T,i)
    if (T == null) return null
    if (i overlaps interval at root)
        return T
    if (T.left != null && T.left.max > i.low)
        return IntervalSearch(T.left,i)
    else
        return IntervalSearch(T.right,i)
```

Can we remove recursion?

08-27: Interval Trees

- How do we find an interval?

```
IntervalSearch(T,i)
   while ((T != null) && (i does not
            overlap interval stored at t))
        if (T.left != null && T.left.max > i.low)
            T = T.left
        else
            T = T.right
   return T
```