COMP282

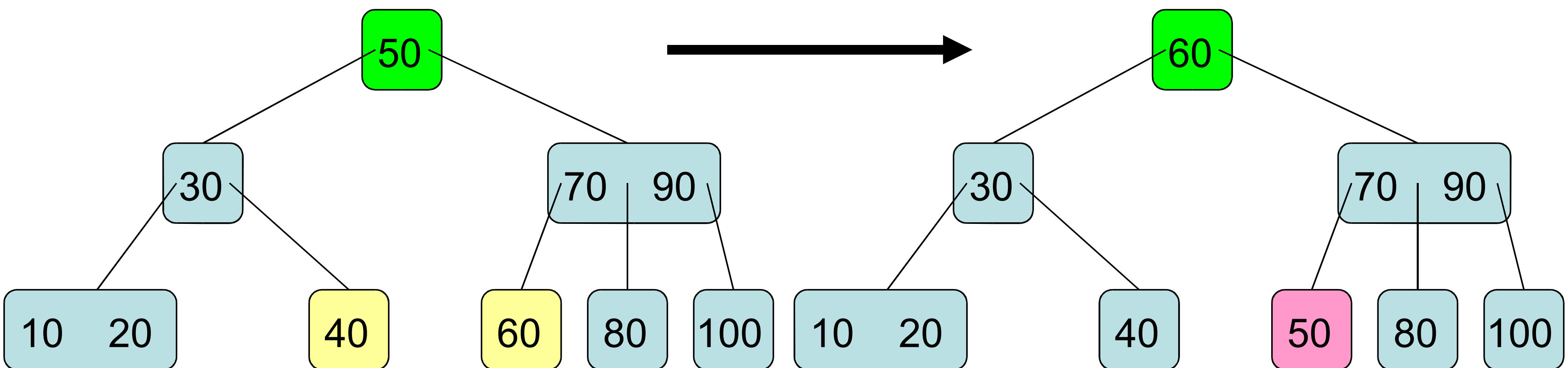
Lecture 16
2-3 Tree Deletion

Deletion

- Deleting a node from a 2-3 tree is only slightly more difficult than insertion.
- Recall that in a binary search tree the easiest nodes to delete are leaves.
- If you want to delete a non-leaf node the simplest method is to swap the node target for deletion with its in-order successor (or predecessor.)
  - This is the same as with normal BSTs.

Simplify by swapping with a leaf

- 2-3 Trees also perform deletions on non-leaf nodes by swapping with successor (or predecessor.)



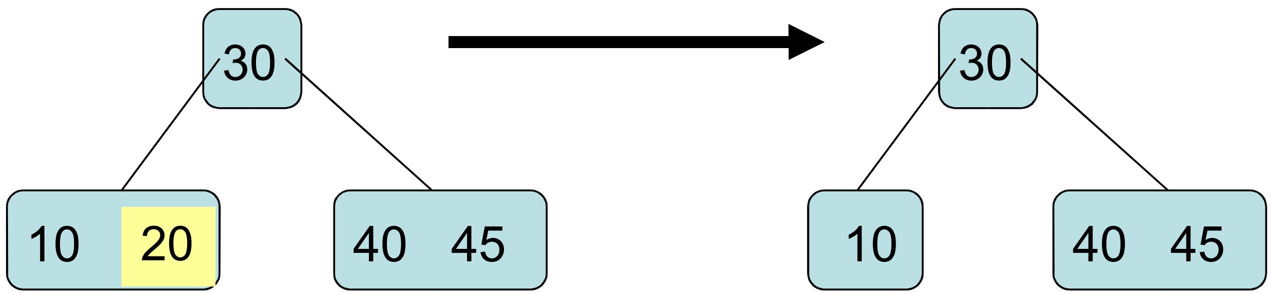
- To delete 50 we swap with 30 or 40 and then delete the leaf we swapped with.

Deletion

- By swapping with successor we are guaranteed to always begin deletion at a leaf.
- There are several distinct cases to consider when attempting to delete this leaf.
- Some are trivial, others more difficult.
- Some will eventually lead to the height of the tree decreasing by one.

The simplest of cases:

- If the item being deleted is held in a 3-node simply delete it.



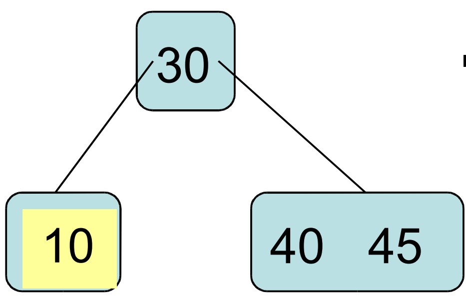
- You don't have to worry about anything because it had no children that could be orphaned.

A little more tricky…

- the node to be deleted is actually a 2-node
  - The node can't simply be deleted because this would leave the parent with a null child reference.
  - This is unacceptable because 2-nodes must have exactly two subtrees and 3-nodes must have exactly three subtrees.
- It may be helpful to think of subtrees as "ordered lists".

Maybe we could “borrow”?

- Since we can’t delete the node outright let’s see if we can delete the item and re-fill the node with a value borrowed from a sibling.



Represents

10, 30, 40, 45

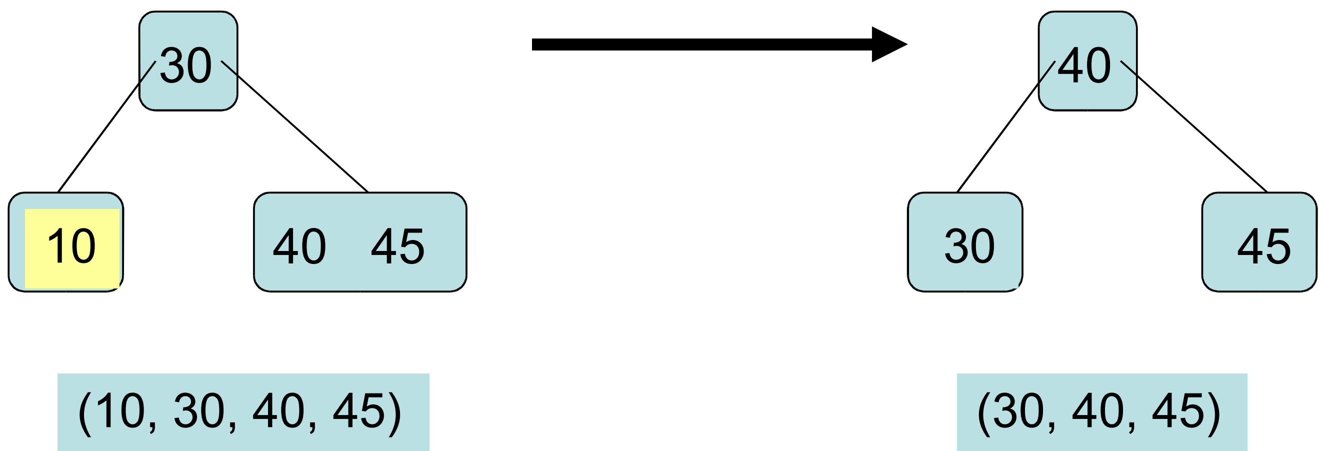
Deletion of 10 yields:

30, 40, 45

We can simply redistribute these values
Among the nodes

Sibling(s) have "extras".

If the sibling is a 3-node (two items) then redistribute the items among the already present nodes:

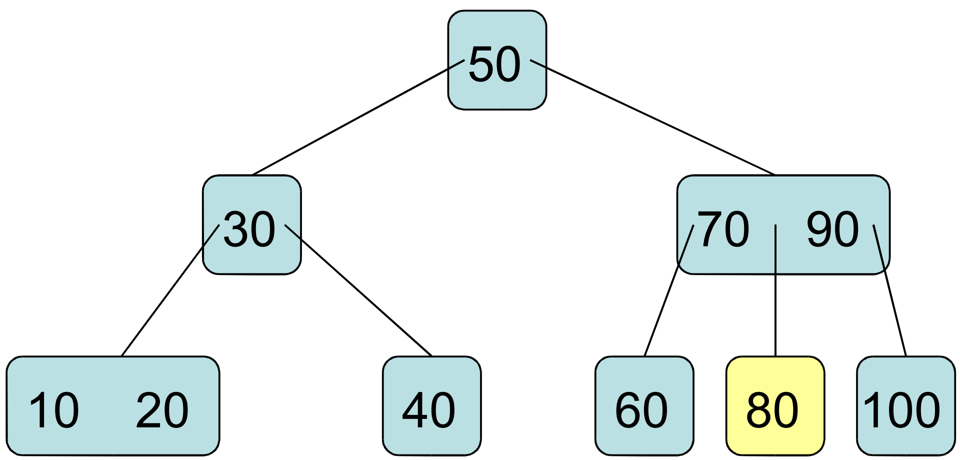


Topology is most important

- Whatever we do the most important point that we must take care of is:
  - Maintain Topology...
    - All nodes must wind up with a proper number of children.
    - The height of the [sub]trees must not change.
    - All keys/values must be accounted for.

Not all siblings want to [can] share

- If we want to delete the 80 neither of its siblings can spare a value.



Choosing a sibling

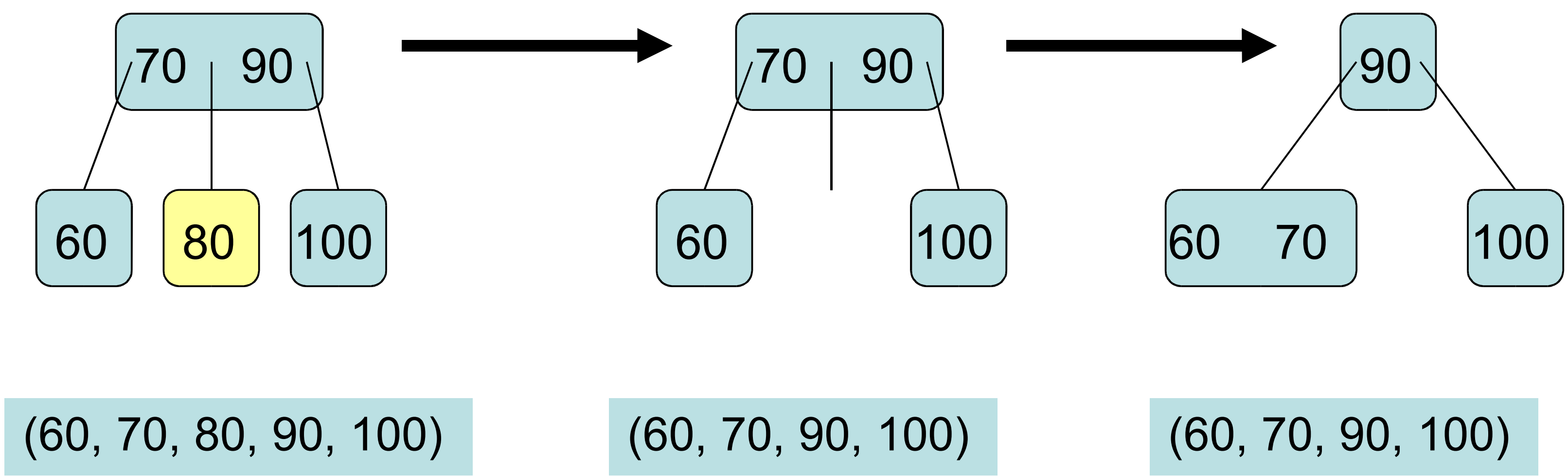
- Note: The simplest definition is: Only use the sibling to your left. (unless you are the left most child then use the sibling to your right.)
- You only care about a sibling because you are trying to identify and choose a proper action to take to restore the topology of the tree.

Procedure

- Actually delete the node.
  - Note: This will leave the parent with one less subtree.
- Move one item from the parent down to one of the remaining siblings (subtrees.)
  - Note: 2-node parents become empty. (This is problematic but can be resolved.)
  - 3-node parents become 2-node and thus can hold exactly two subtrees; Luckily there are exactly two remaining subtrees.

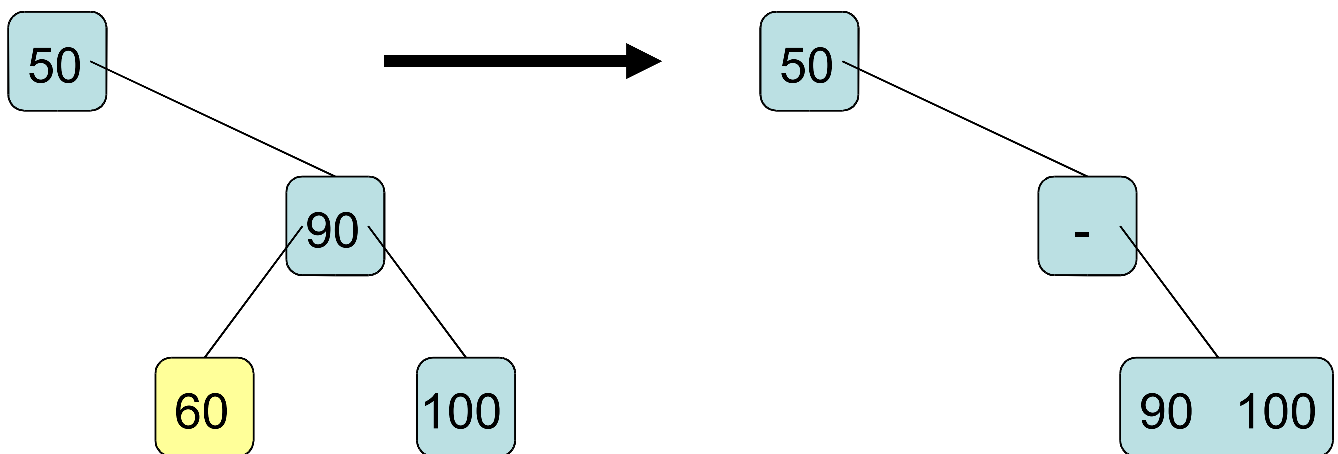
Example

- Parent can spare a value.



2-node parents become empty

- If the parent was a 2-node the push of the parent's single item to a sibling will leave the parent node "empty".



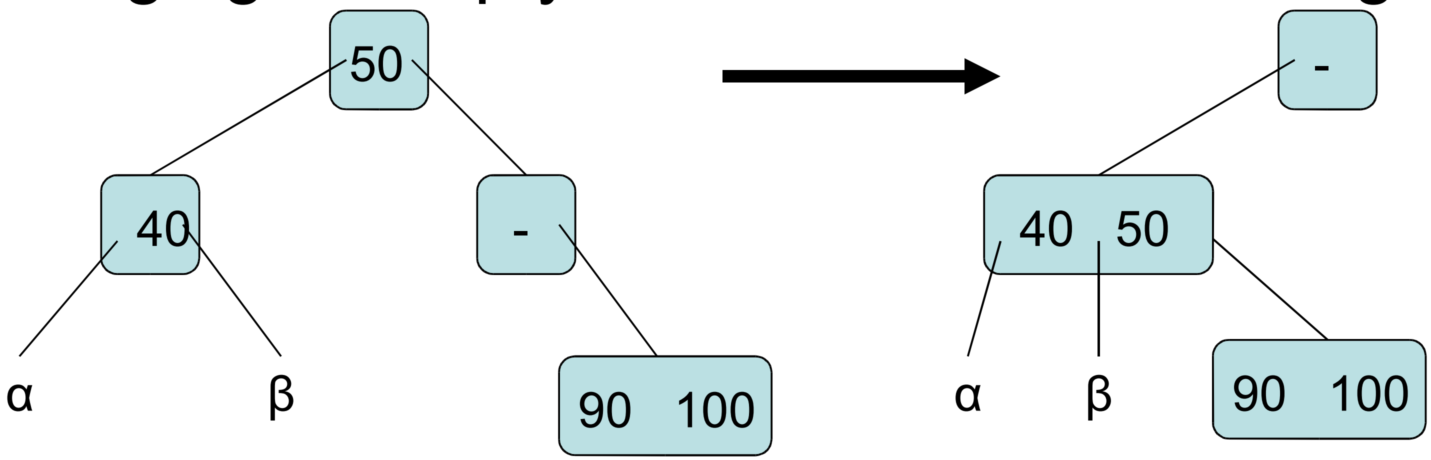
- Solution: Recursively treat this problem as a "leaf" deletion without swapping.

Recursive procedure:

- Resolve an “empty” node by:
  - Check if the sibling is a 3-node.
    - Redistribute the siblings and parent’s items.
    - Re-adopt subrees appropriately.
  - If the sibling is a 2-node:
    - Merge the empty node with its sibling by moving an item from the parent node down to a sibling and moving the child of the empty node over to the sibling.
    - This can cause the parent node to become empty.

Merging Example

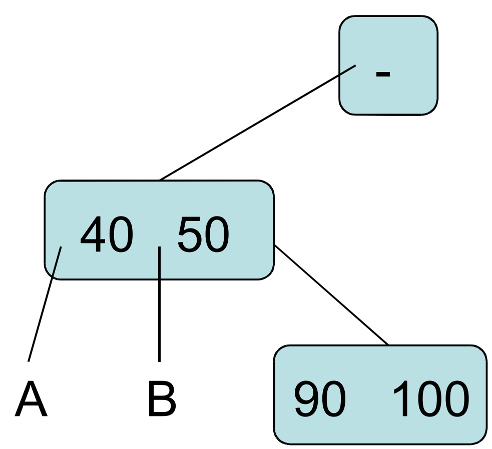
- Merging of empty interior node with sibling.



- If the parent node (50) had been a 3-node it would have simply become a 2-node.
- 2-node parents (as shown here) become empty and need to be recursively resolved.

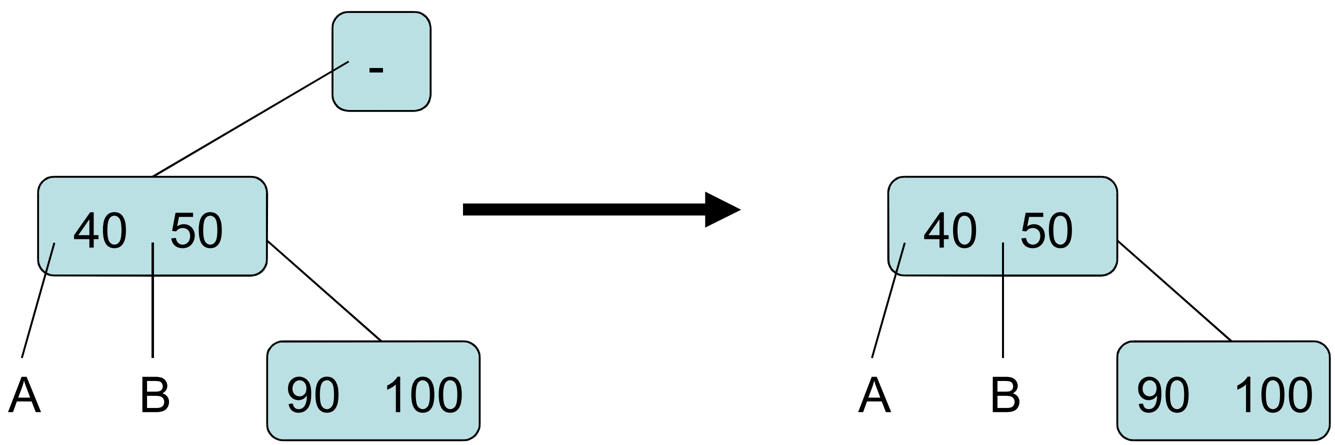
Recursive resolution.

- Simply repeat the procedure until either:
  - the parental node is not empty after the procedure.
  - The empty node is the root node.
    - In this case: The empty root node will have a single subtree that is waiting to be orphaned.



Decrease in height.

- Empty root nodes decrease the height of the tree by 1.



Original height = 3

new height = 2

End