COMP282

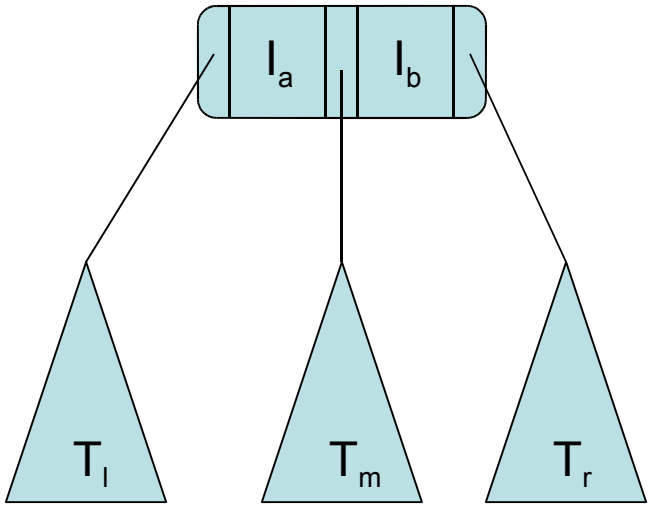
Lecture 13

2-3-4 trees

Insertion

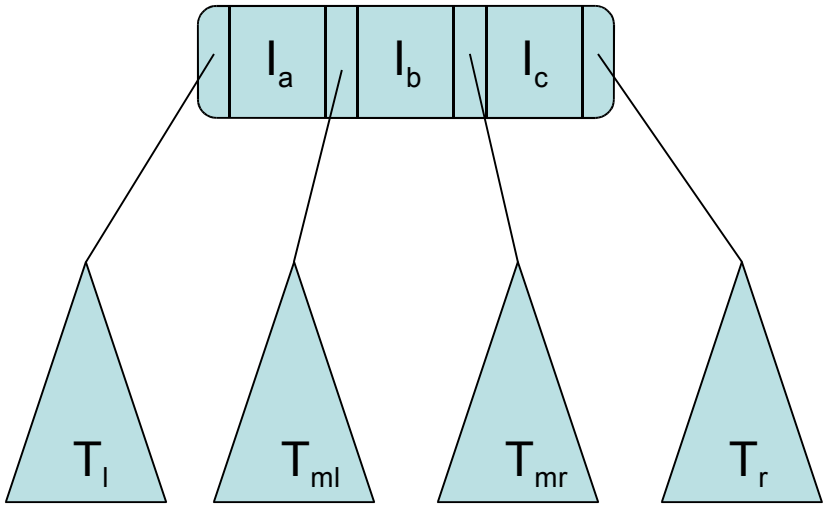
- If 2-3 trees “rock” then…
  - what about 2-3-4 trees?
  - 2-3-4-5-6-7-8-9 trees?
    - (silliness actually)
- 2-3-4 tree though are useful because the steps required to resolve insertion and deletion dilemmas are reduced compared to 2-3 trees.

- Same Math as 2-3 tree, except now nodes can have four subtrees (and 3 items)



$\{T_l\} < I_a < \{T_m\} < I_b < \{T_r\}$

"3" node



$\{T_l\} < I_a < \{T_{ml}\} < I_b < \{T_{mr}\} < I_c < \{T_r\}$

"4" node

Traversal and searching

- Traversal and searching operations are simply extensions of the same procedures for Binary Search Trees (BST) and 2-3 Trees.

Insertion

- In order to insert an item into a 2-3-(4) tree it is first necessary to locate the leaf that the item will be inserted into.
- This requires that we start from the root and make comparison decisions until we arrive at a leaf.
- This effectively traverses a “path” from the root node down to that leaf.

The difficulty with insertions:

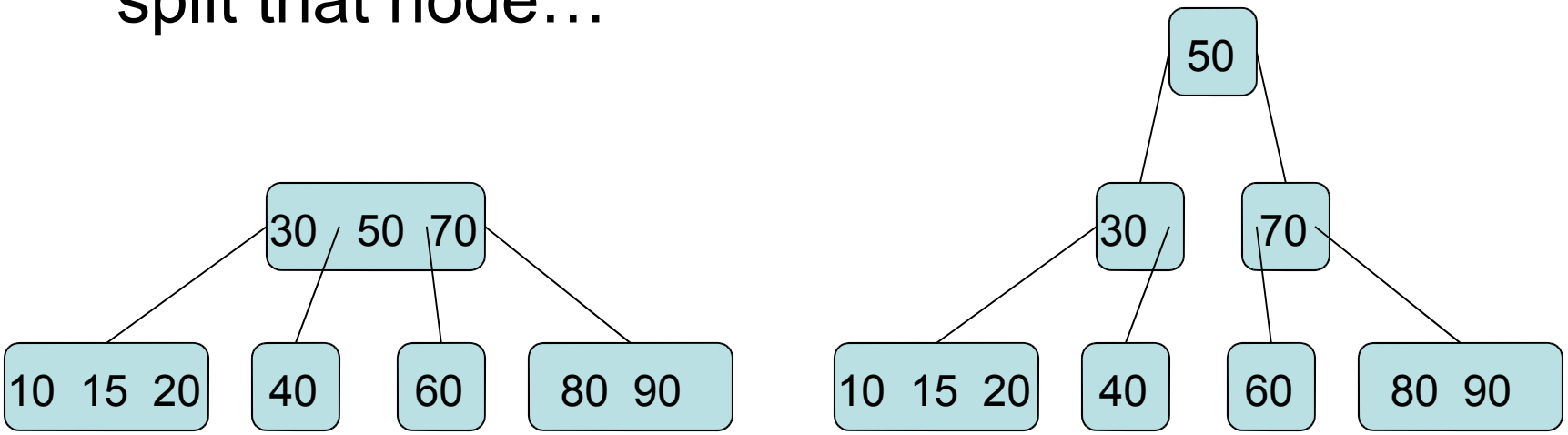
- Recall that in 2-3 trees we blindly followed this path to the leaf. Then we (over)filled nodes to accommodate the new item.
- Overfilled nodes were resolved by pushing items up to the parent.
- This push had the possibility of overfilling the parent as well and the problem was propagated up the tree.

Taking advantage (of descent)

- 2-3-4 Trees make use of the decent from the root to the leaf.
- 2-3 trees blindly descend to the leaf and make modifications to the tree only on the way back up as nodes become overfull.
- 2-3-4 also make modifications to the tree during the descent to find the leaf

The simple modification

- While searching for the leaf on the way down if we encounter a “4” node then we immediately split that node…



- Why would you do such craziness?
  - To make room for items that might be pushed up! (now 50, 30 and 70 can hold at least one more item that might be pushed up from below.)

Reasoning:

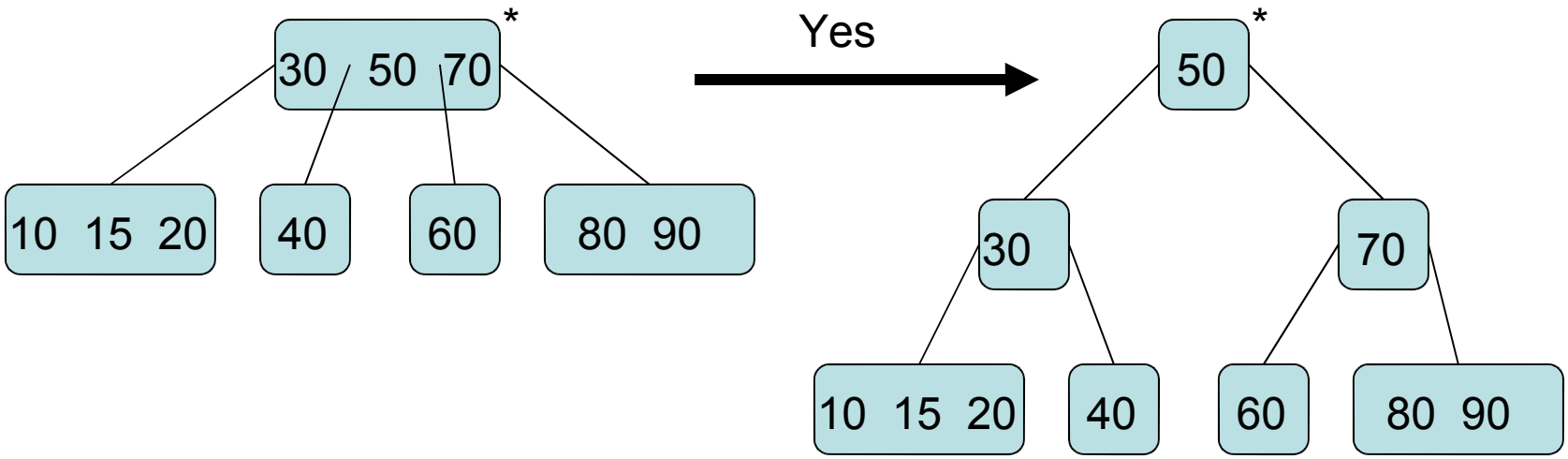
- 2-nodes and 3-nodes can always hold one more item.
- The only nodes that can’t hold one more item are 4-nodes.
- Items are only pushed up to ancestors.
- If we encounter an ancestor on the way down that won’t be able to accept an item we can transform it into 2-nodes that can before the insertion is actually performed.

Guarantee:

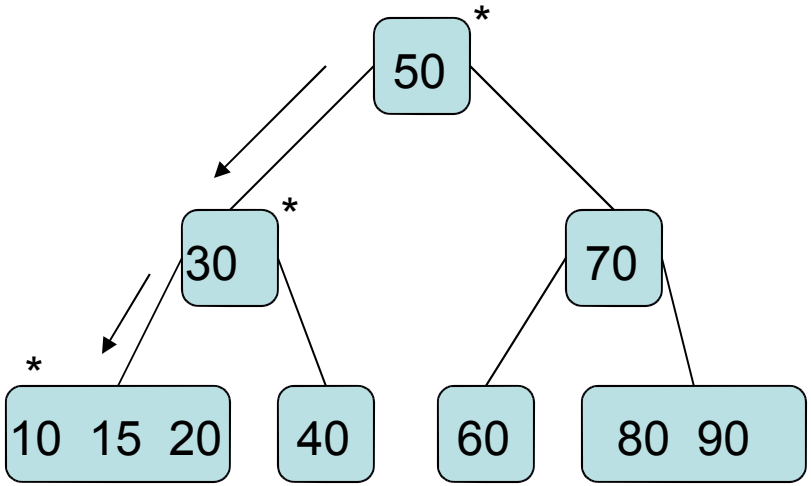
- This guarantees that the exact same insertion routine presented for 2-3 trees can be performed without ever having to worry about “resolving” overfull nodes.

Example:

- Adding 16 to the following tree:
  - Step 1a: (descent) Is current node a 4-node?

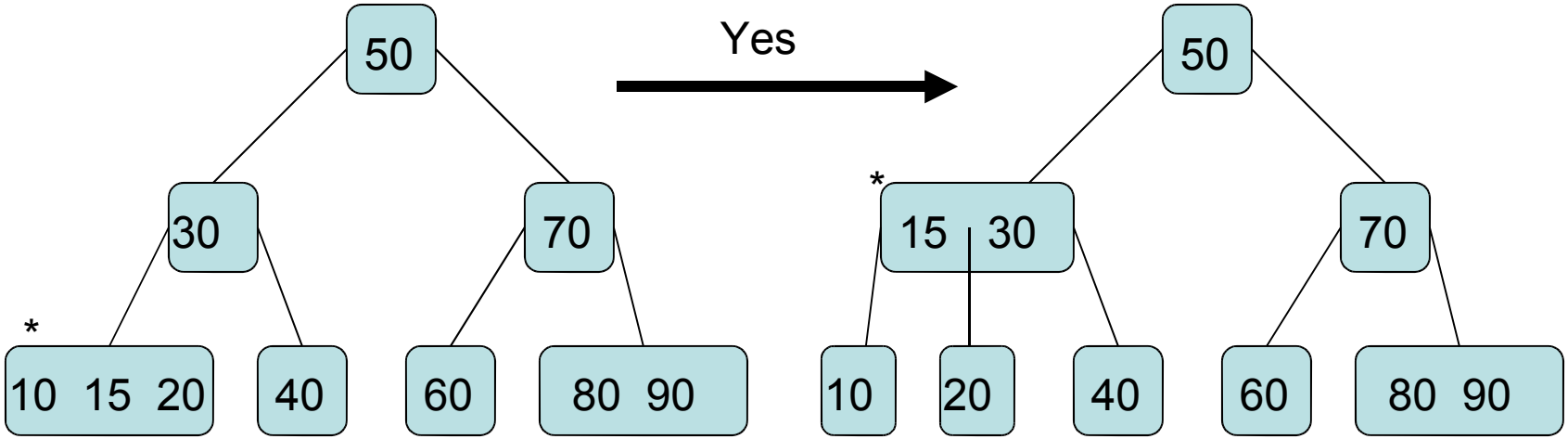


- Step 1b: Compare SearchKey with current node. (16<50)

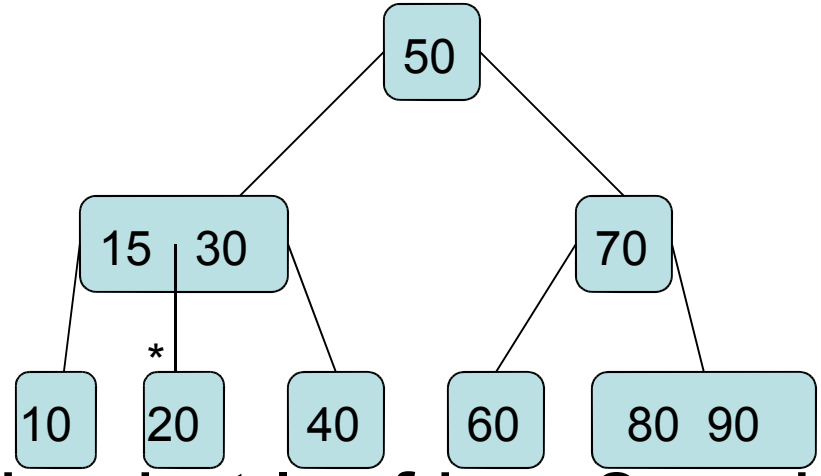


- Step 1a: Is current node (*) a 4-node? No.
- Step 1b: Compare SearchKey with current node (16<30)

- Step 1a: Is current node a 4-node? Yes.

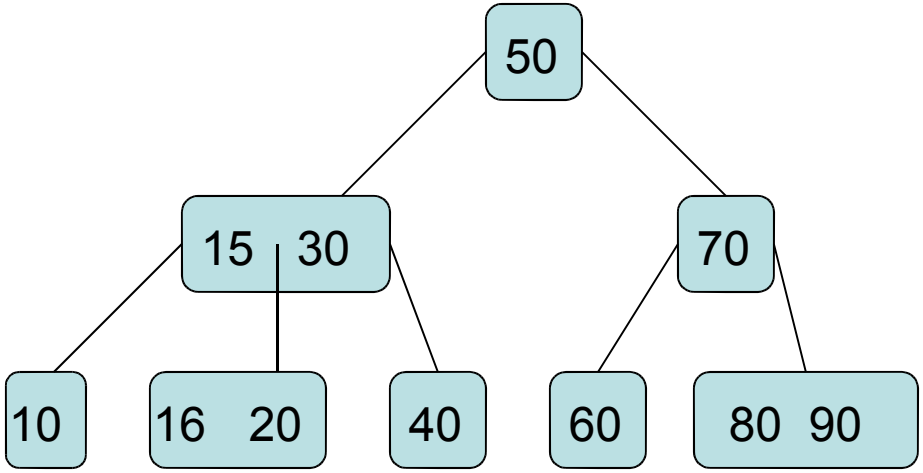


- 16<20 which is the middle branch.



- The arrived at leaf is a 2-node (or maybe a 3-node) so you can simply insert the new item and nothing will have to propagate up

- The insertion…



- Nothing can propagate up because the necessary space was made during the descent to the leaf.