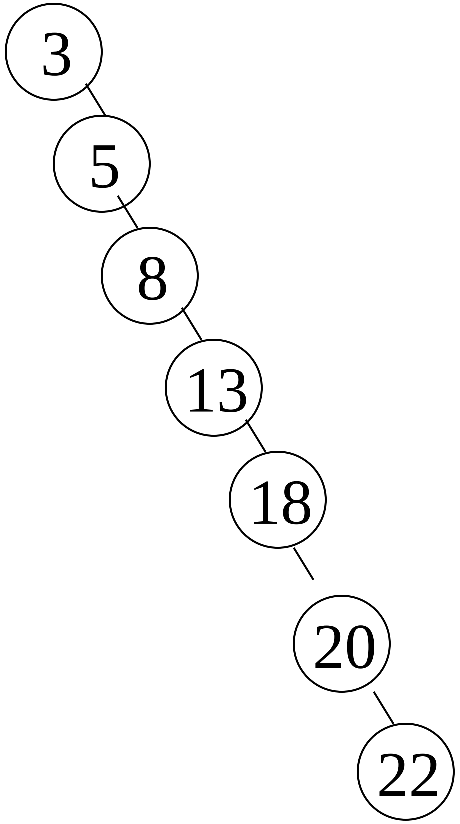
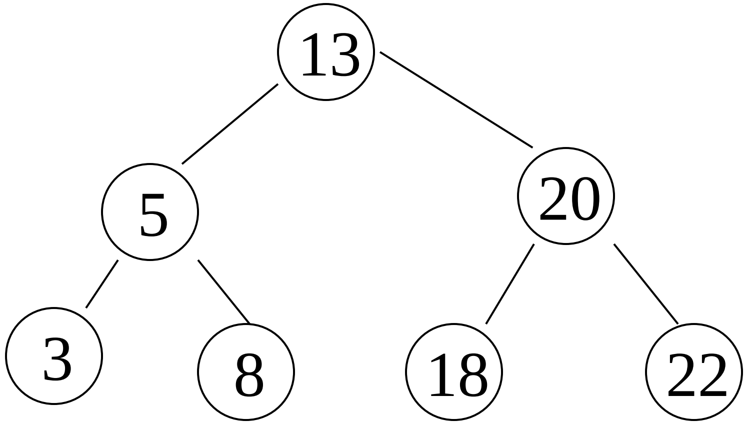
AVL TREES

Self Balancing Binary Search Tree

MOTIVATION

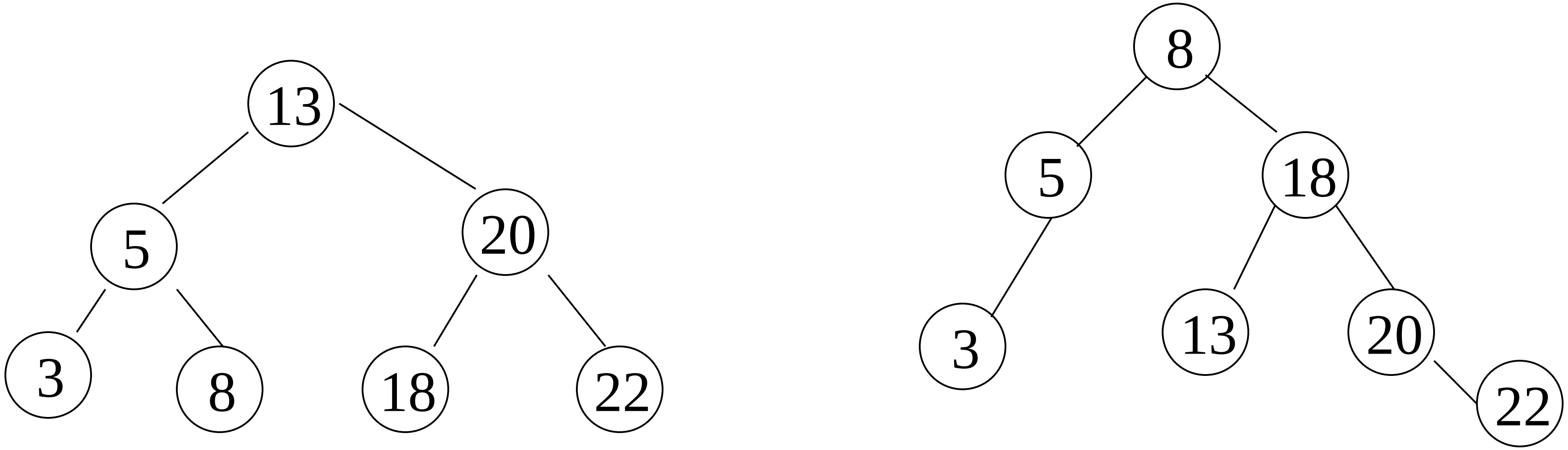
When building a binary search tree, what type of trees would we like? Example: 3, 5, 8, 20, 18, 13, 22





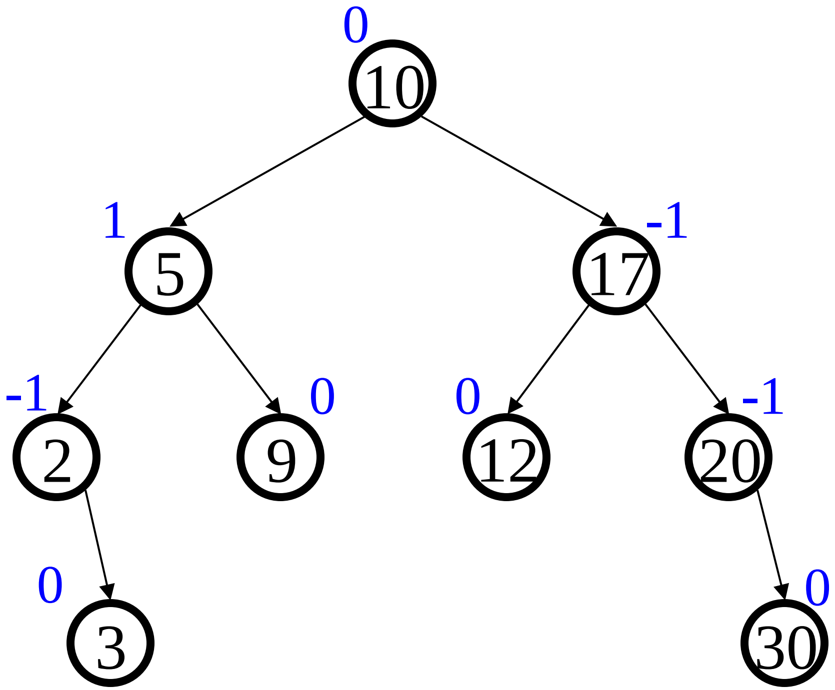
MOTIVATION

- We want a tree that has the following properties
  - Tree height = O(log(N))
  - allows dynamic insert and remove with O(log(N)) time complexity.
- The AVL tree is one of this kind of trees.



AVL (ADELSON-VELSKII AND LANDIS) TREES

- An AVL Tree is a ***binary search tree*** such that for every internal node v of T, the *heights of the children of v can differ by at most 1.*



An example of an AVL tree where the heights are shown next to the nodes:

AVL TREE PROPERTIES

- AVL tree is a binary search tree with balance condition
  - To ensure depth of the tree is O(log(N))
  - And consequently, search/insert/remove complexity bound O(log(N))
- Balance condition
  - For every node in the tree, height of left and right subtree can differ by at most 1
  - Balance factor of a node = Height of its left subtree – Height of its right subtree
  - Balance factor of a node can be -1,0 or 1

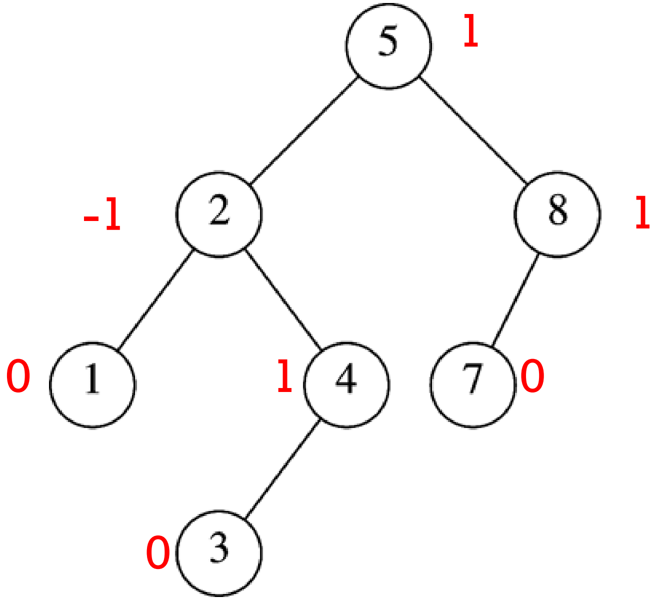
BALANCE FACTOR OF A NODE IN AN AVL TREE

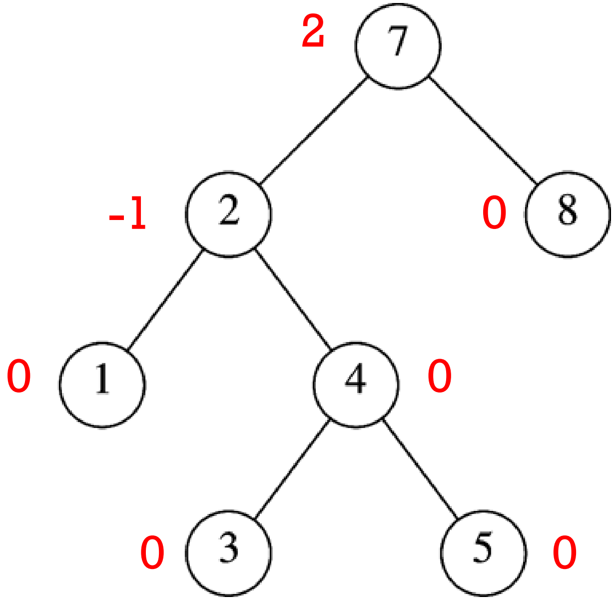
- Height of left subtree > height of right subtree
  - Balance Factor is 1
  - Tree is Left Heavy

- Height of left subtree < height of right subtree
  - Balance Factor is -1
  - Tree is Right Heavy

- Height of left subtree = height if right subtree
  - Balance Factor is 0
  - Tree is balanced

WHICH IS AN AVL TREE?





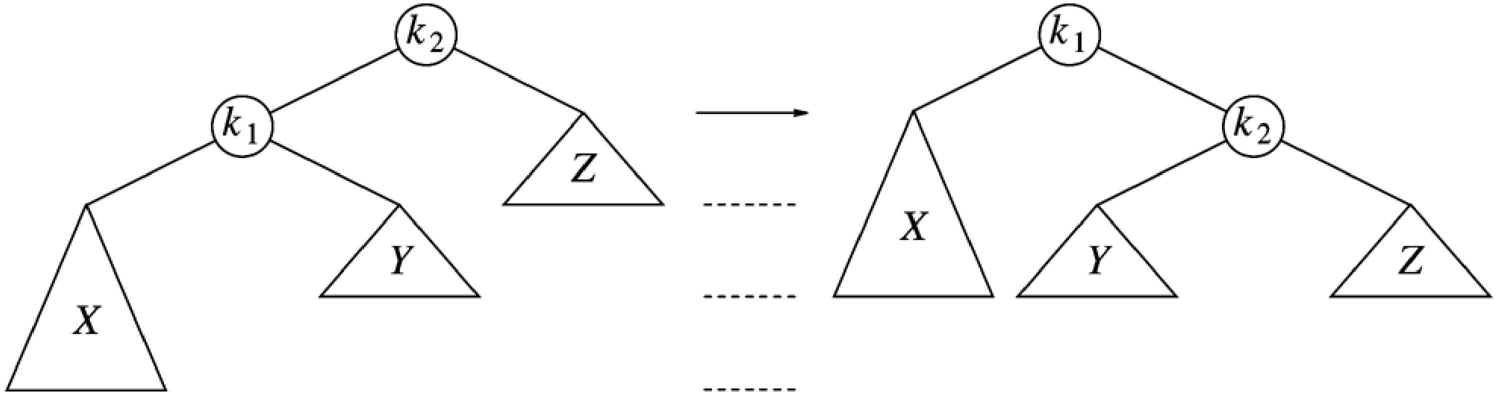
AVL TREE INSERT AND REMOVE

- Do binary search tree insert and remove
- The balance condition can be violated sometimes
  - Do something to fix it : rotations
  - After rotations, the balance of the whole tree is maintained

BALANCE CONDITION VIOLATION

- If condition violated after a node insertion
  - Which nodes do we need to rotate?
  - Only nodes on path from insertion point to root may have their balance altered
- Rebalance the tree through rotation at the deepest node with balance violated
  - The entire tree will be rebalanced
- Violation cases at node k (deepest node)
  1. An insertion into left subtree of left child of k
  2. An insertion into right subtree of left child of k
  3. An insertion into left subtree of right child of k
  4. An insertion into right subtree of right child of k
  - Cases 1 and 4 equivalent
    - Single rotation to rebalance
  - Cases 2 and 3 equivalent
    - Double rotation to rebalance

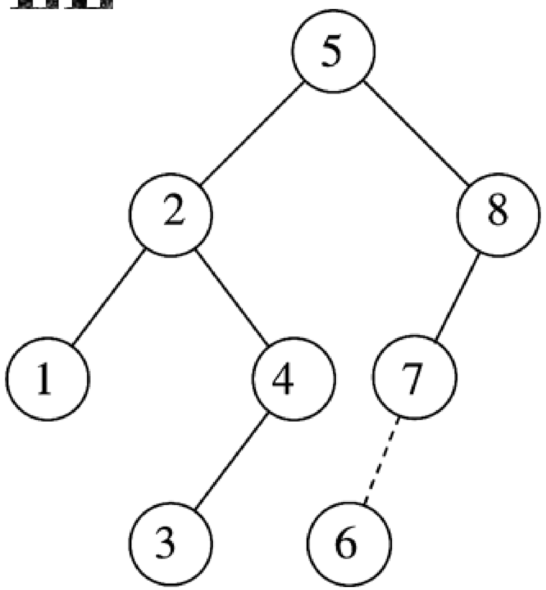
SINGLE RIGHT ROTATION

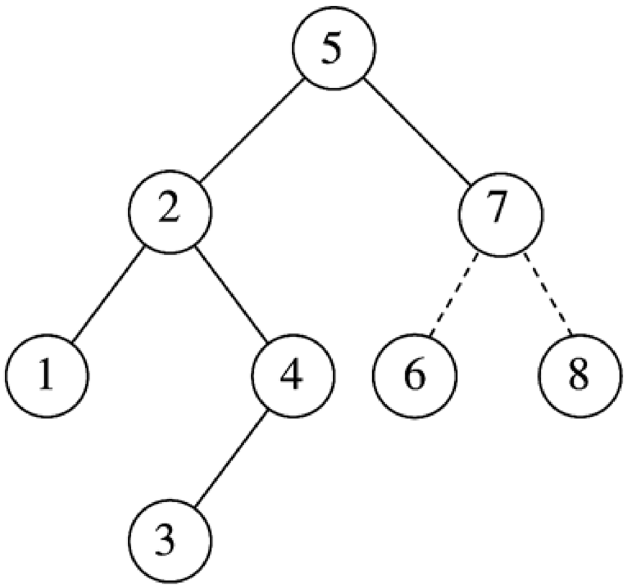


- ***Single rotation:*** The basic operation we'll use to rebalance
  - Move child of unbalanced node into parent position
  - Parent becomes the "other" child (always okay in a BST!)
  - Other subtrees move in only way BST allows (we'll see in generalized example)

- Replace node k_2 by node k_1
- Set node k_2 to be right child of node k_1
- Set subtree Y to be left child of node k_2
- Case 4 is similar

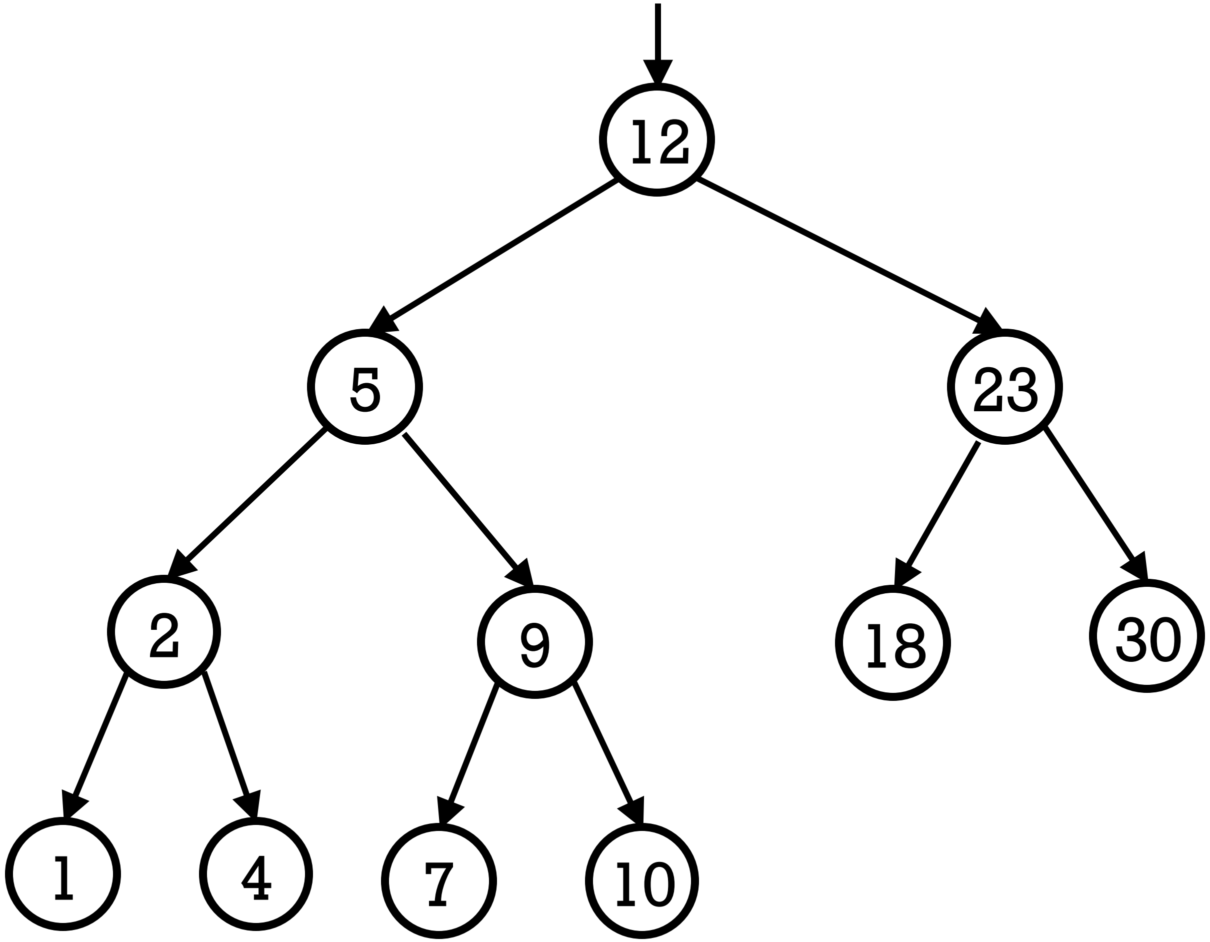
EXAMPLE



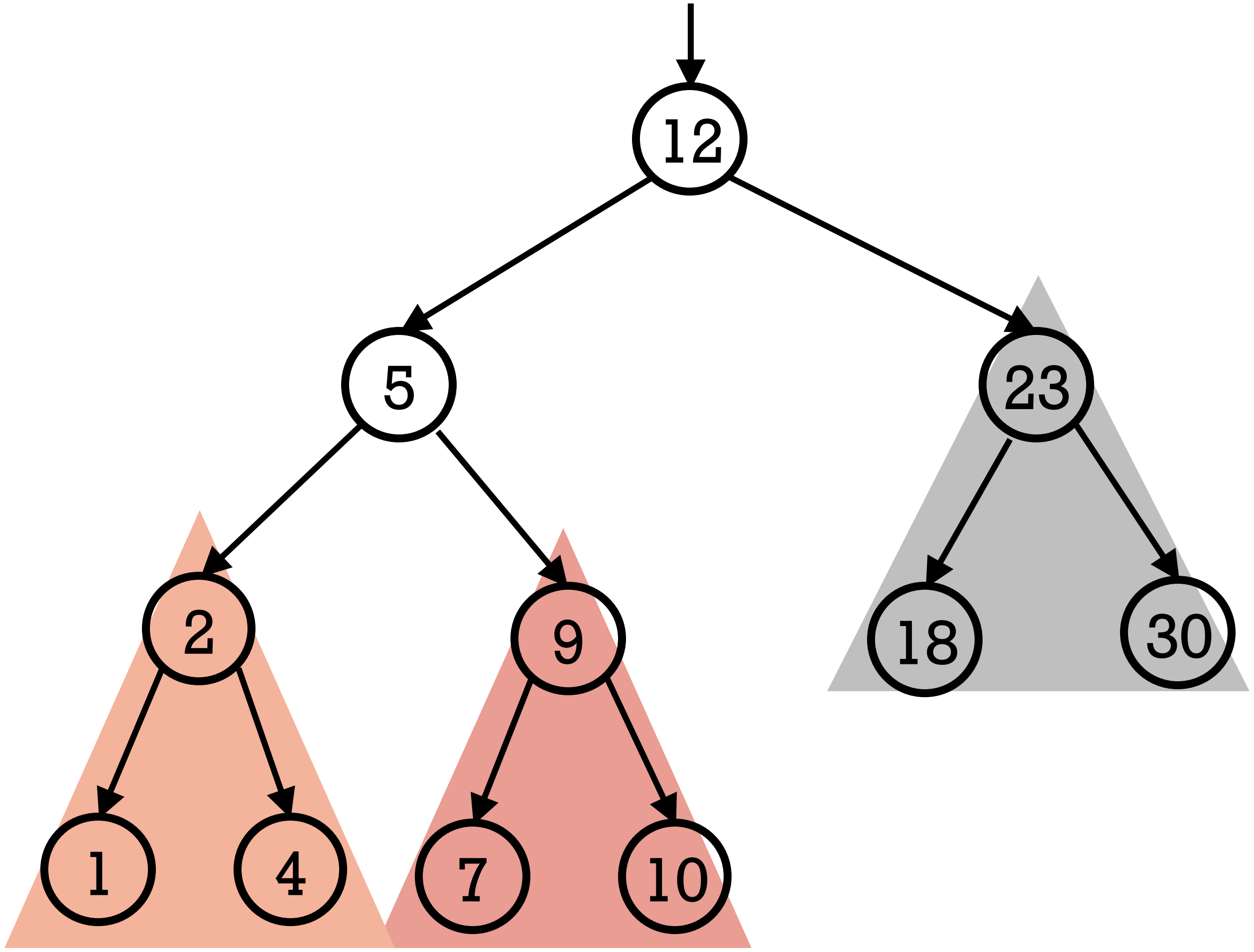


- After inserting 6
  - Balance condition at node 8 is violated
  - Perform Right Rotation on 8

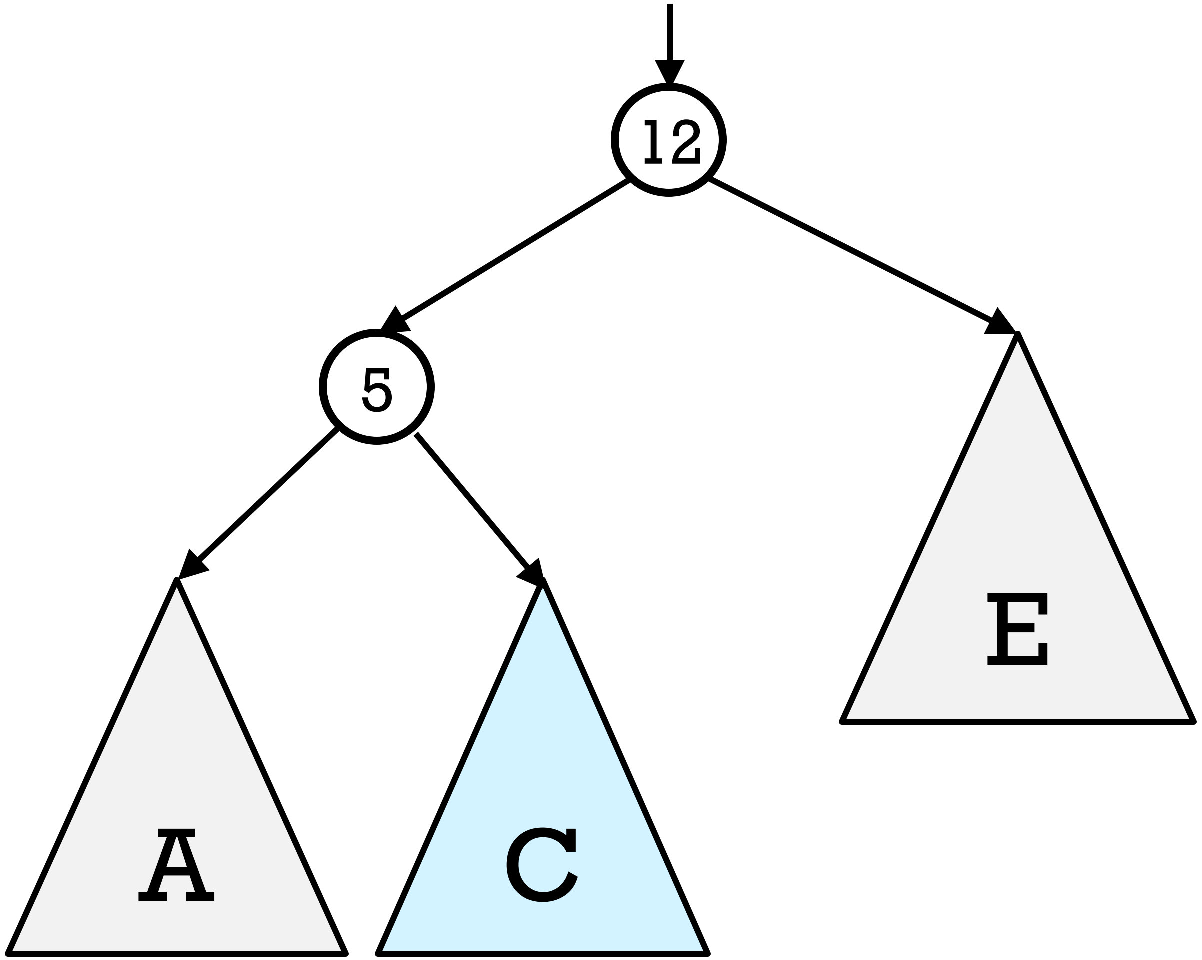
GENERALIZING...



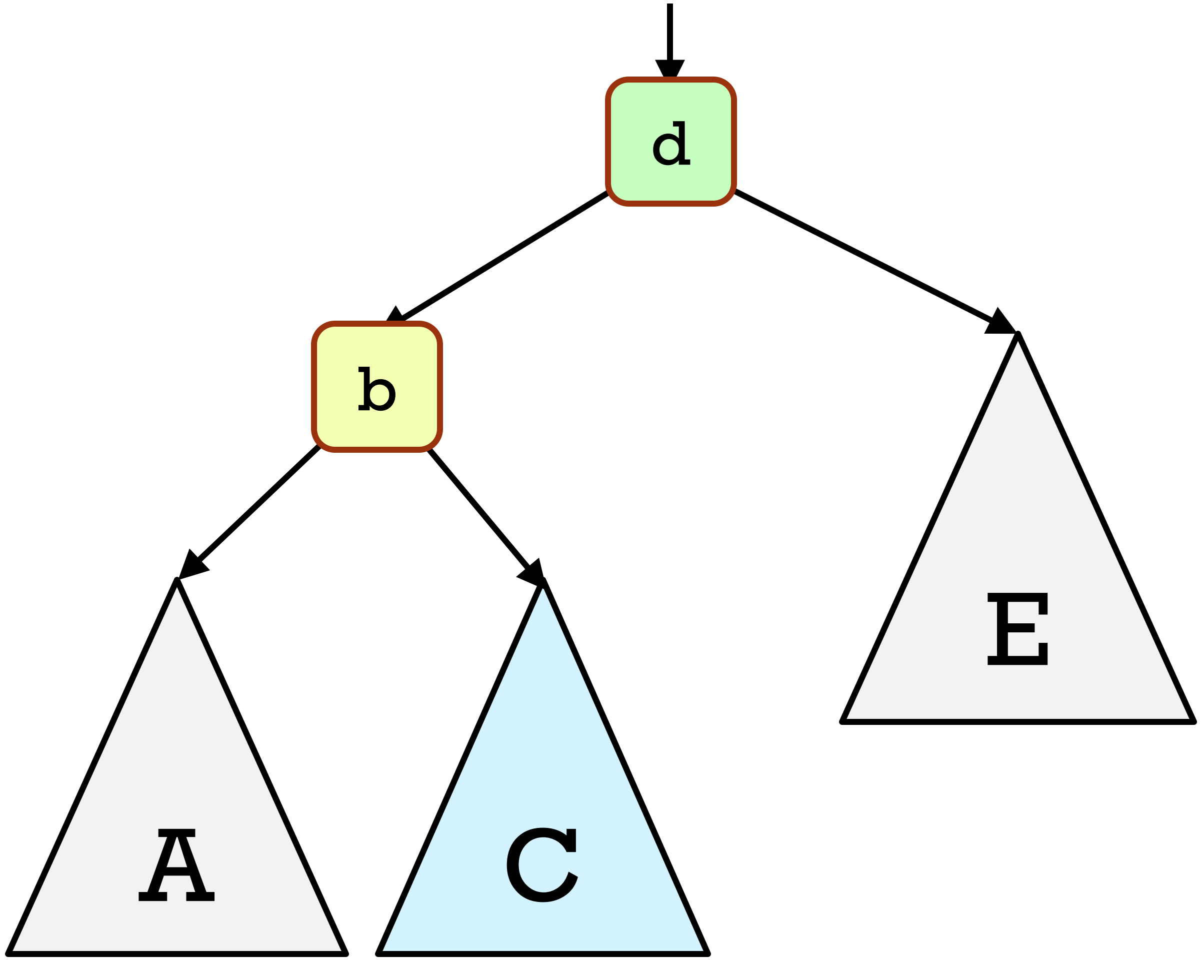
GENERALIZING OUR EXAMPLES...



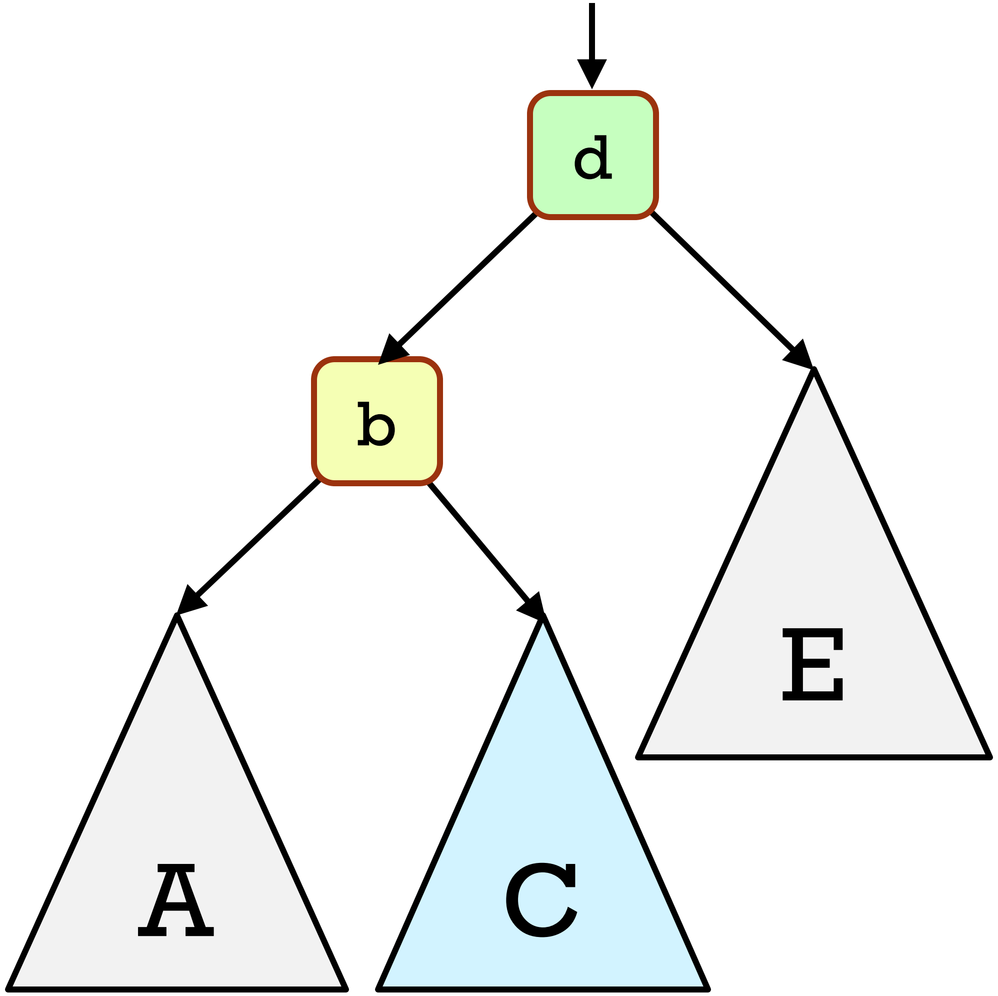
GENERALIZING OUR EXAMPLES...



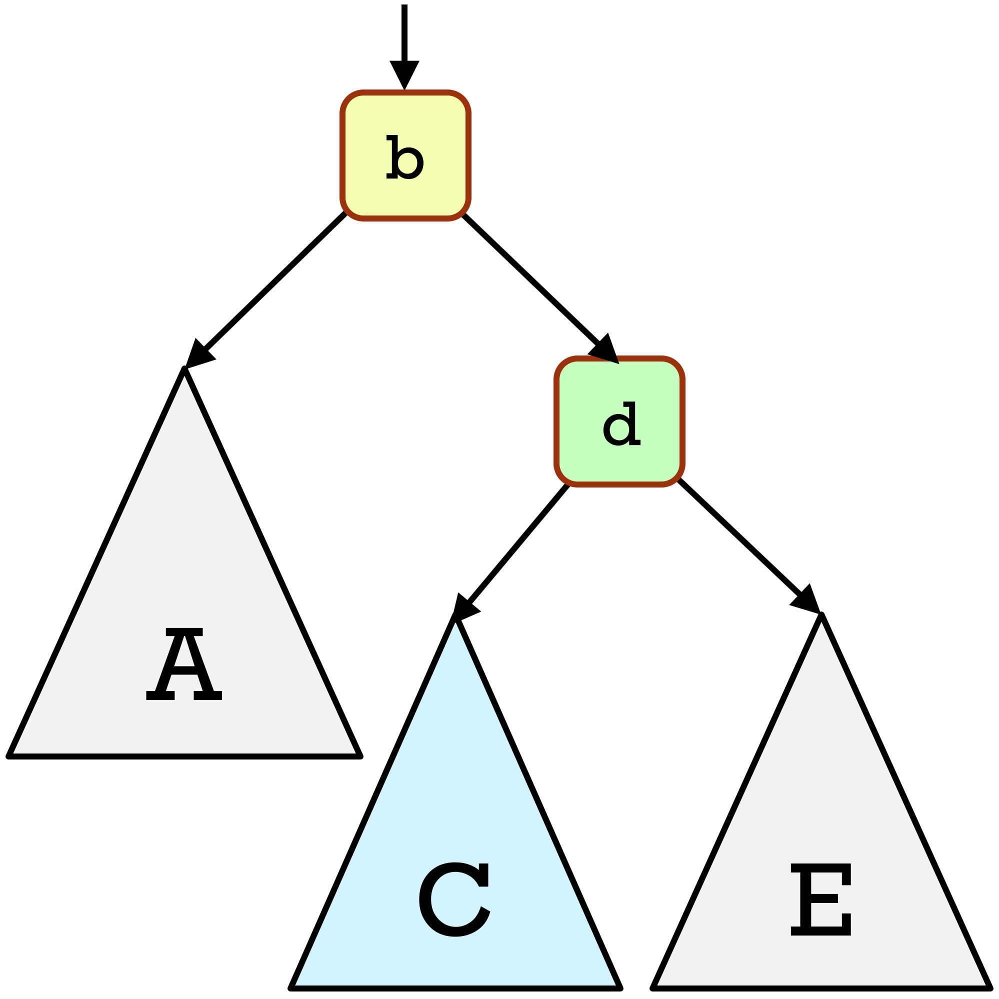
GENERALIZING OUR EXAMPLES...



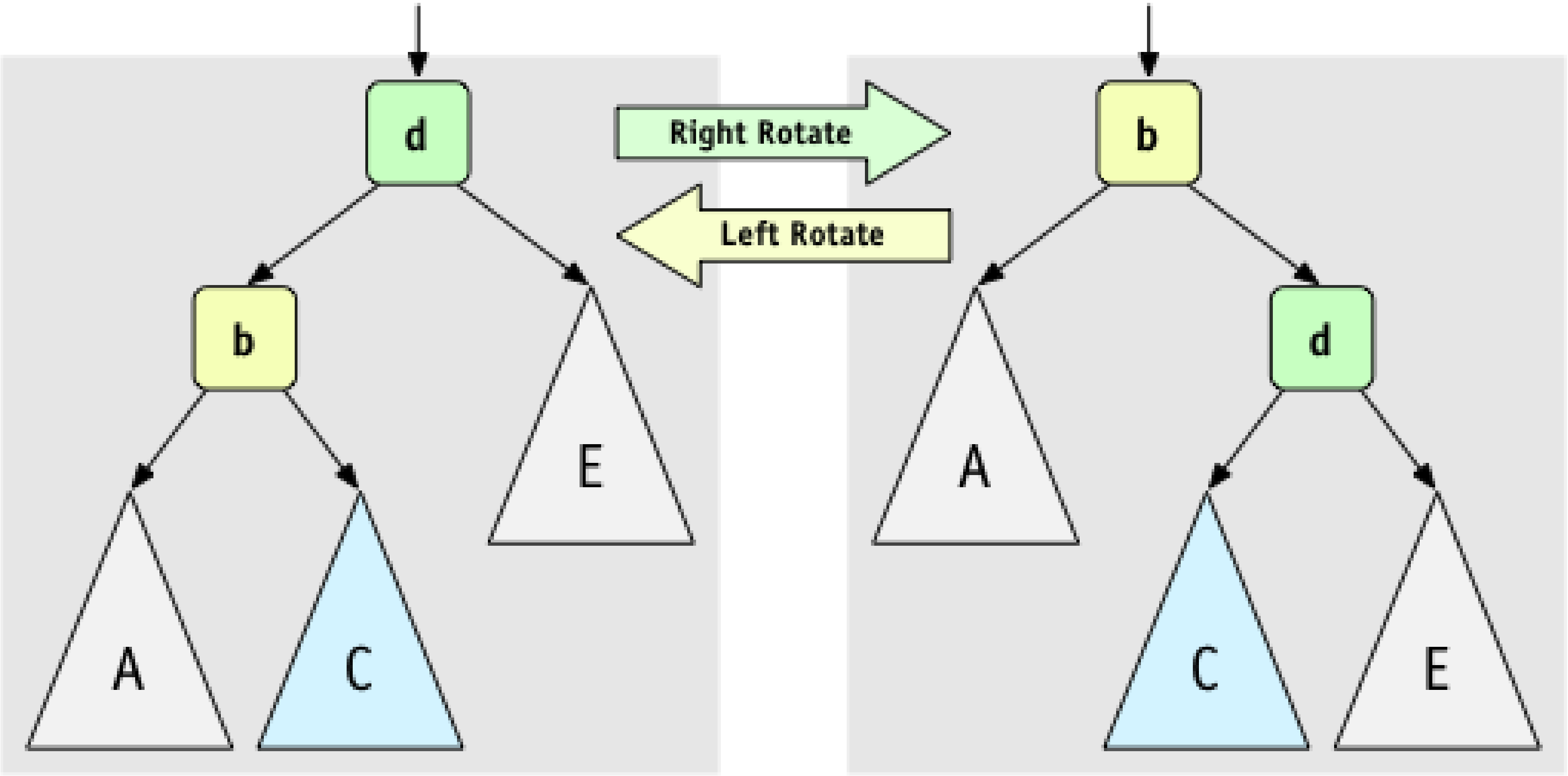
GENERALIZED SINGLE ROTATION



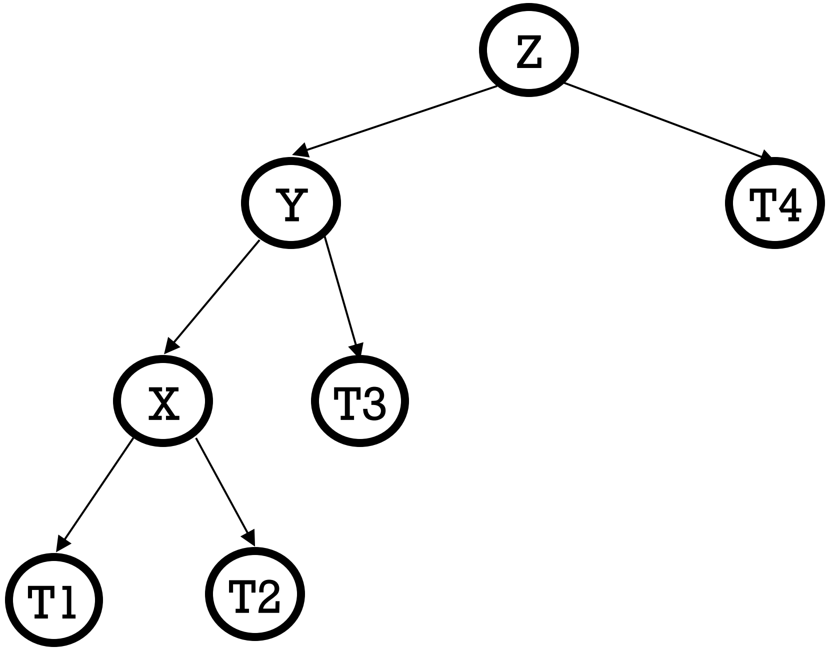
GENERALIZED SINGLE ROTATION

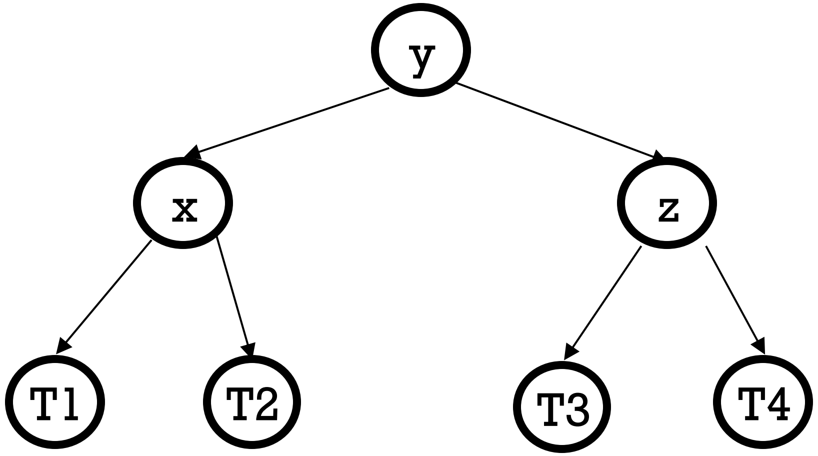


SINGLE ROTATIONS



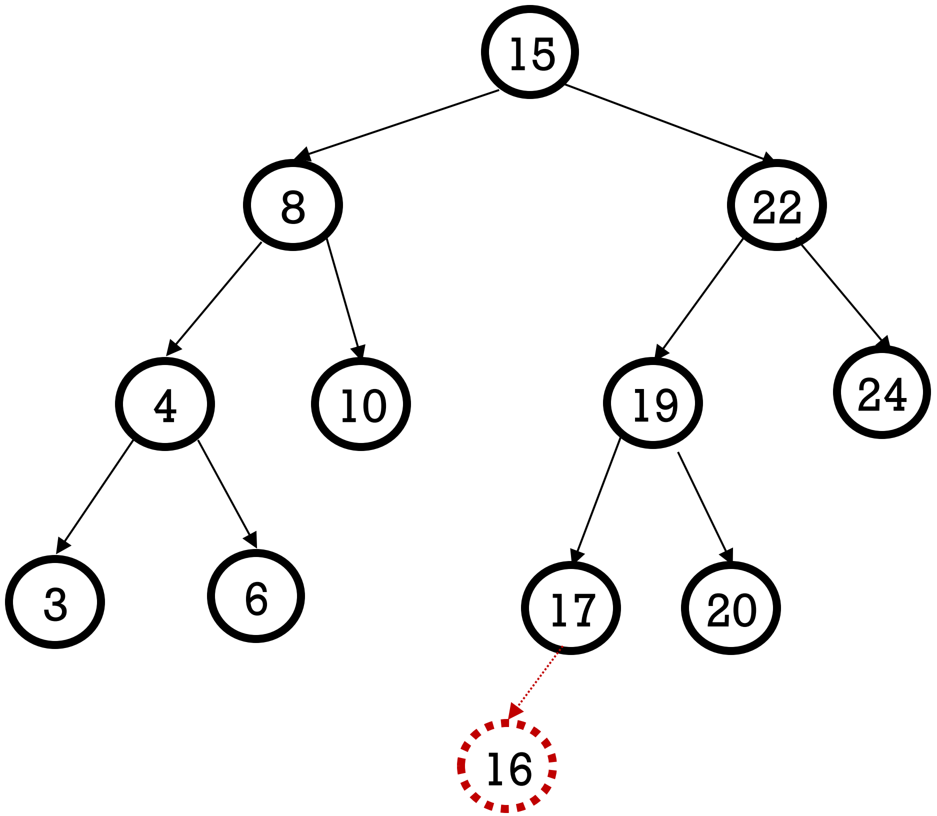
INSERTION CASE #1 LEFT LEFT

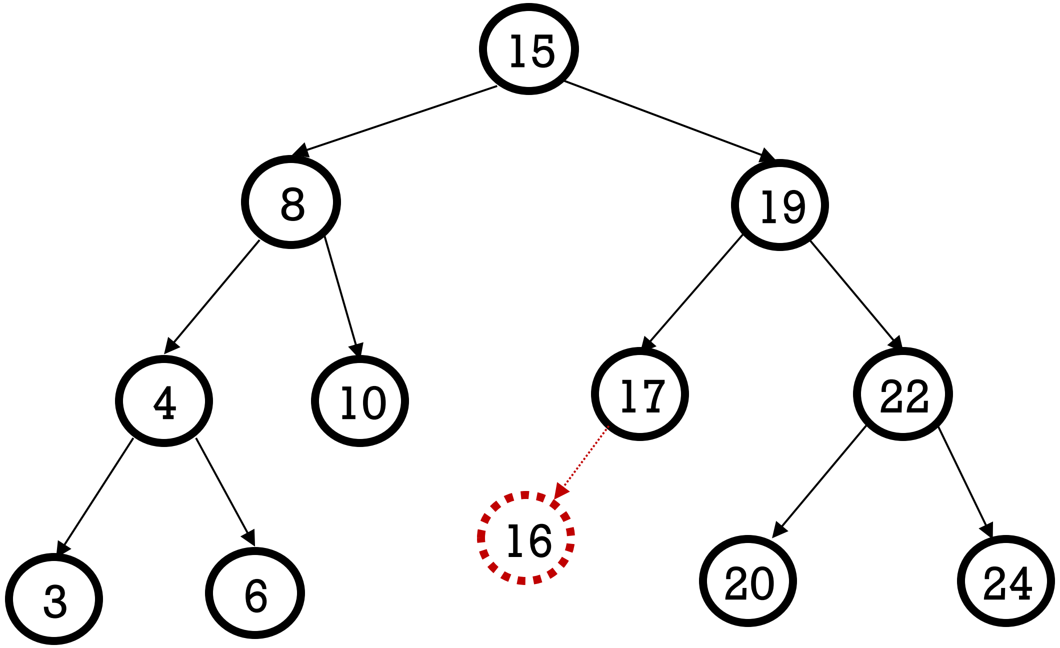




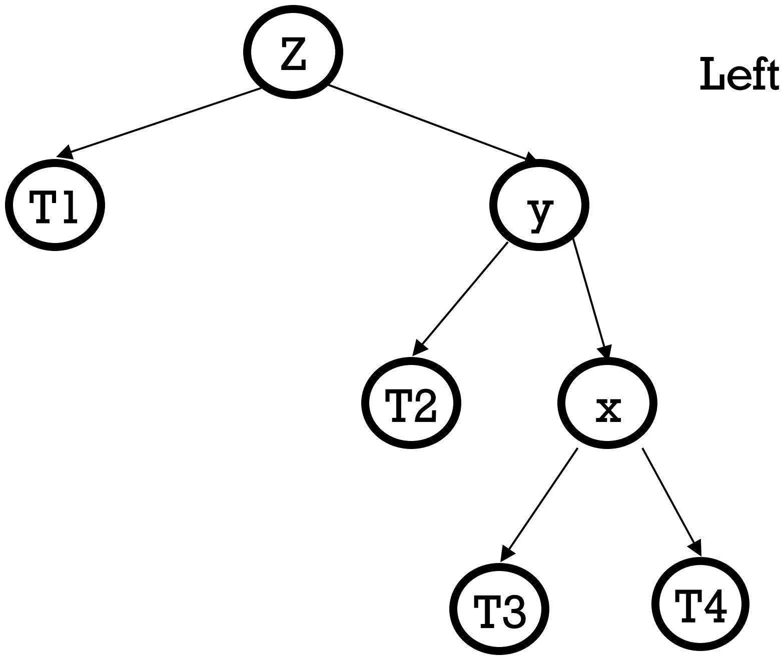
Right Rotate on z

INSERT CASE # 1 EXAMPLE

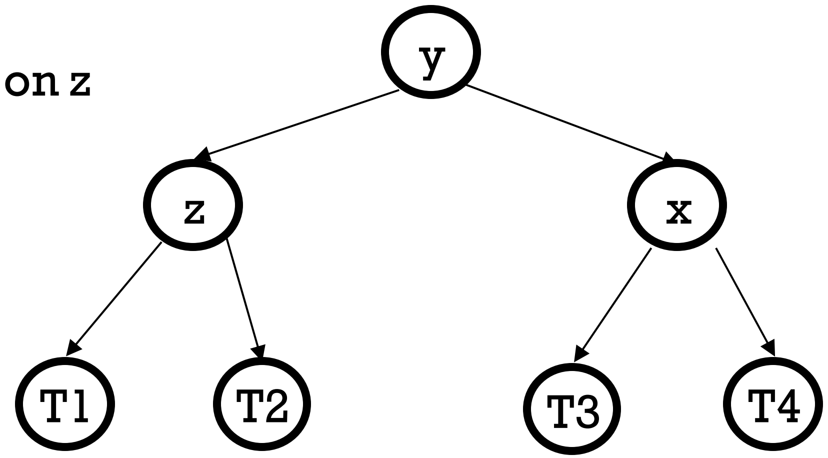




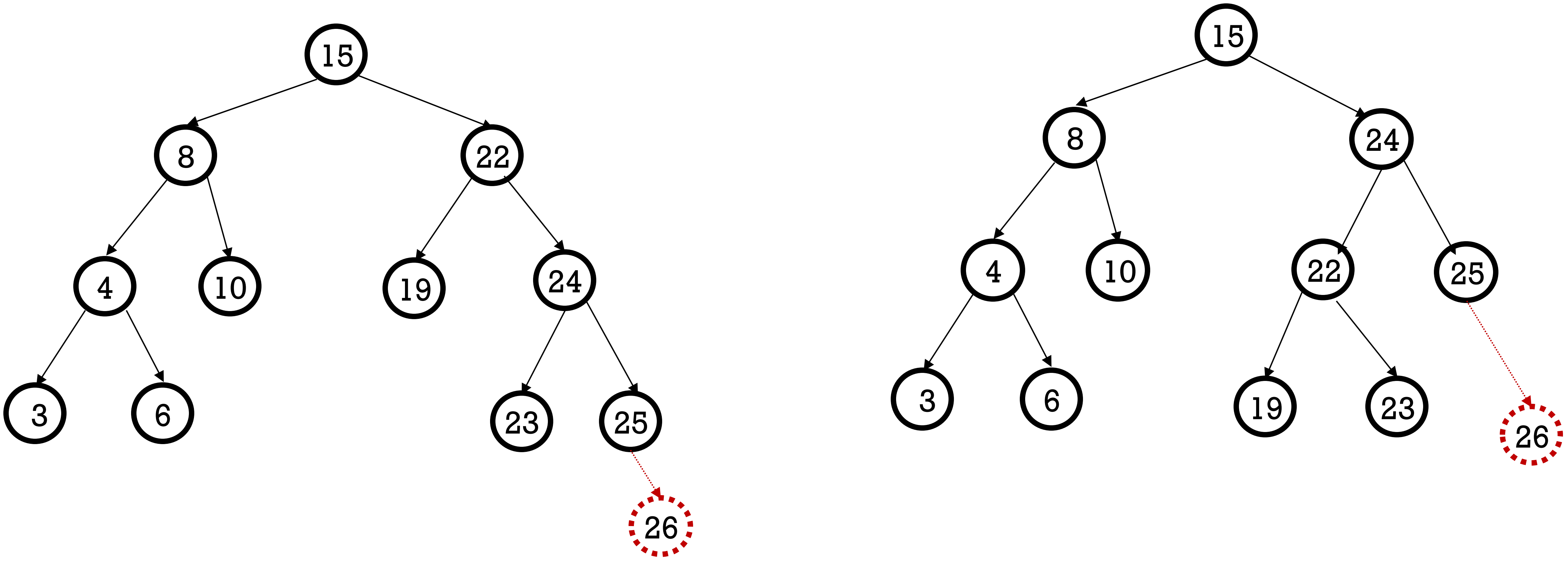
INSERT CASE #2 RIGHT RIGHT



Left Rotate on z

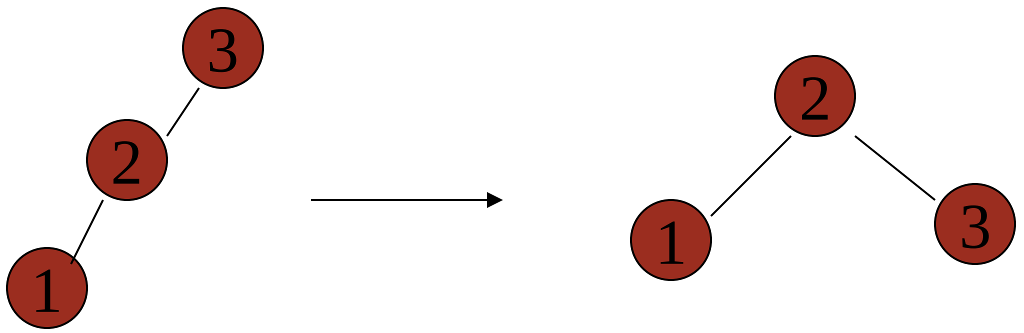


INSERT CASE # 2 EXAMPLE



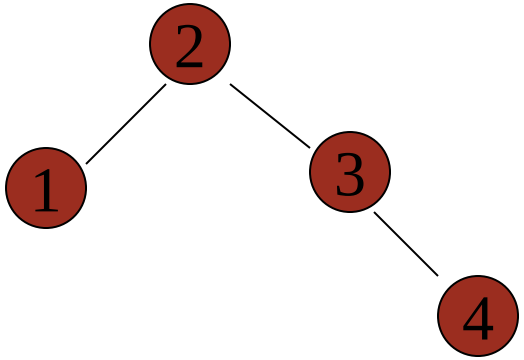
EXAMPLE

- Inserting 3, 2, 1, and then 4 to 7 sequentially into empty AVL tree

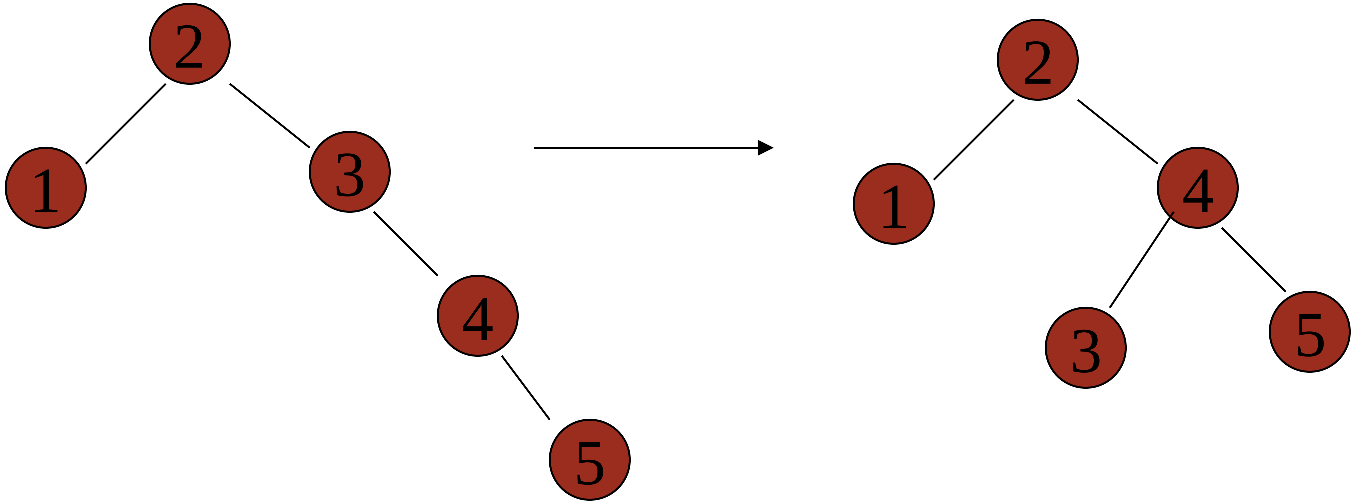


EXAMPLE

- Inserting 4

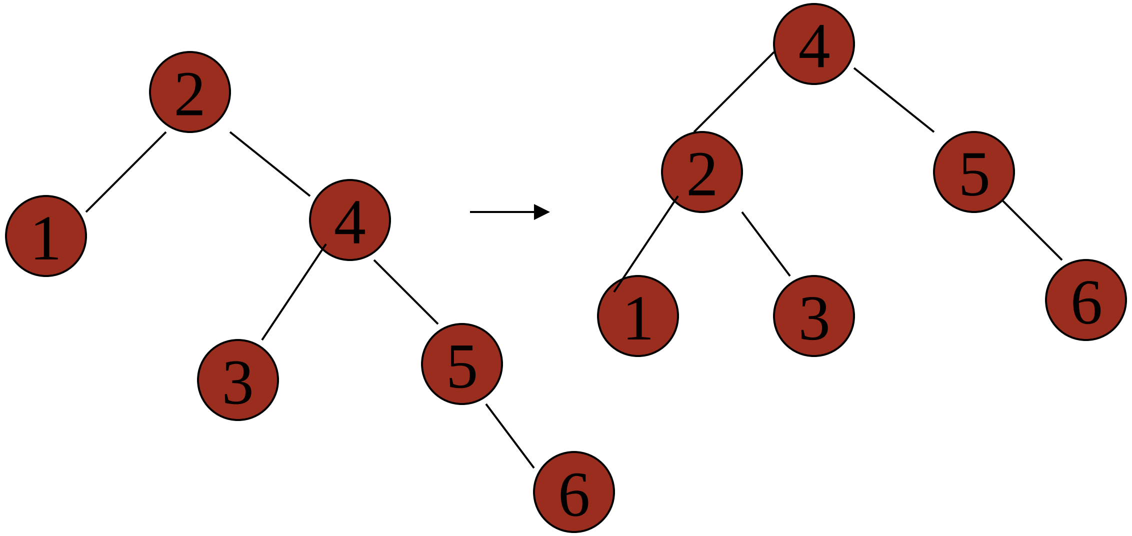


- Inserting 5

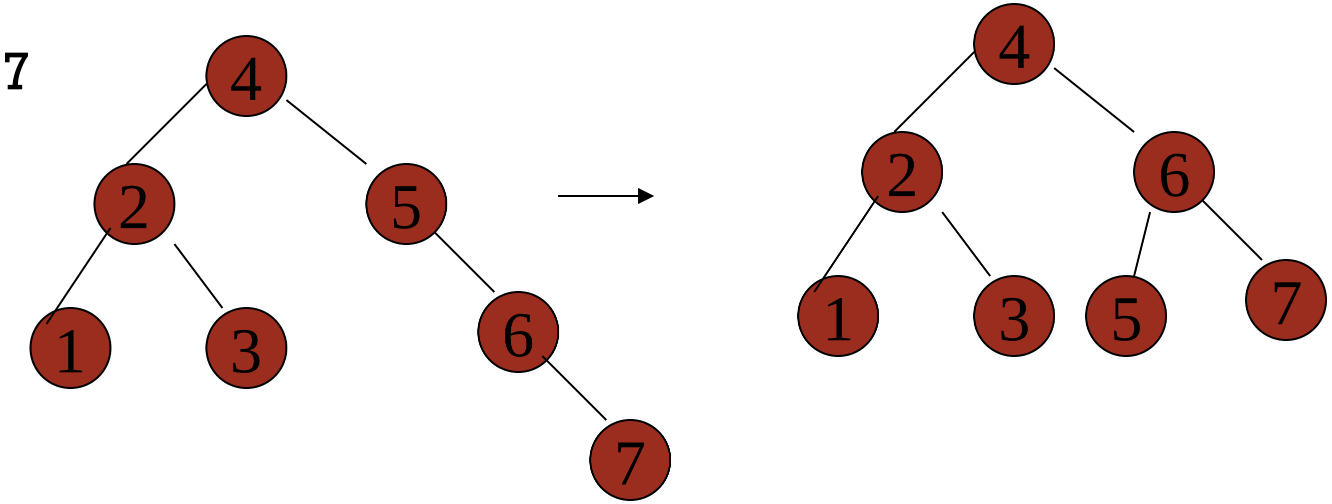


EXAMPLE

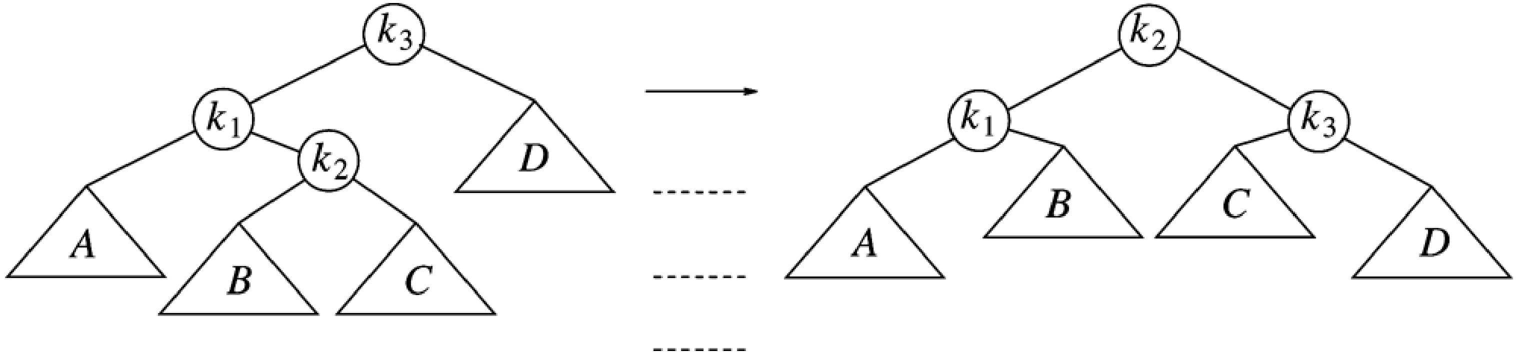
- Inserting 6



- Inserting 7

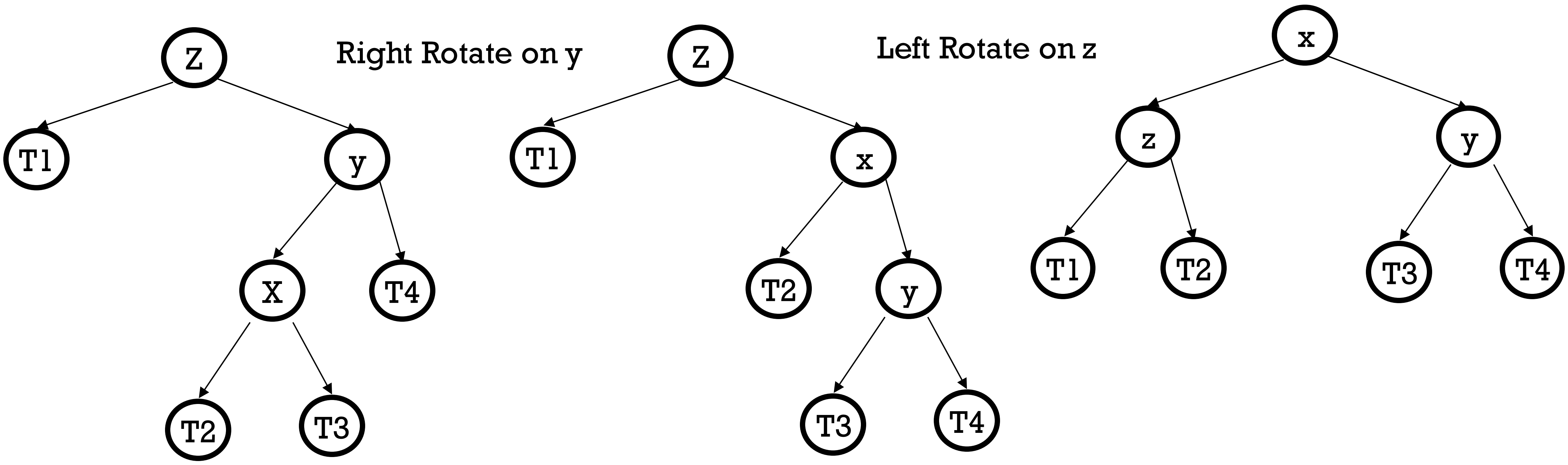


DOUBLE ROTATION

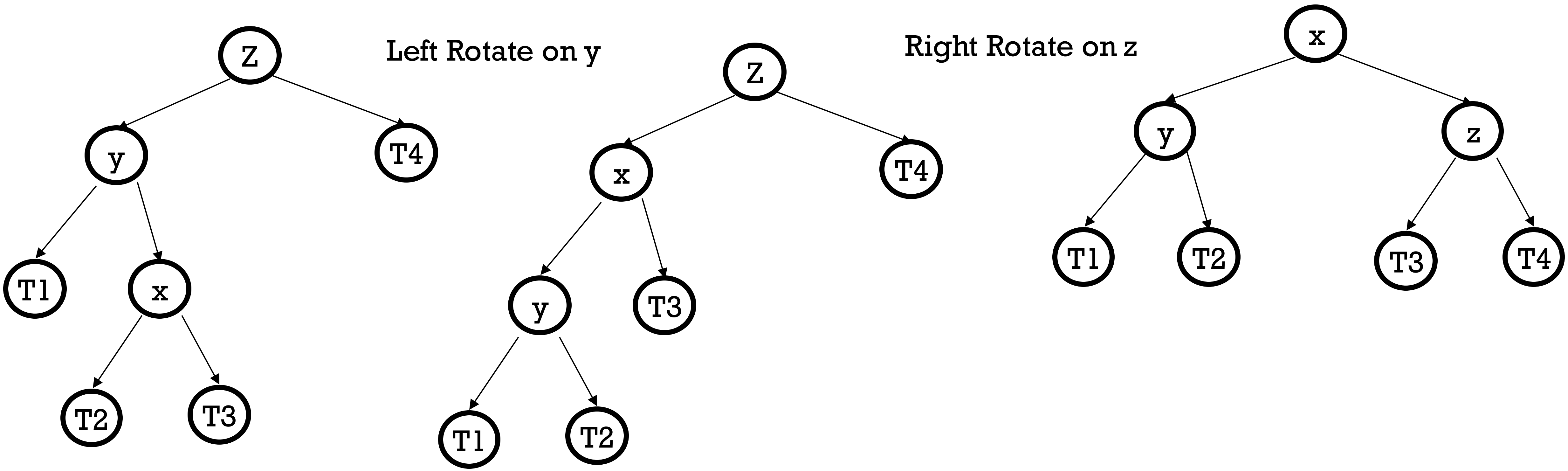


- Left-right double rotation to fix
- First rotate between $\mathbf{k_1}$ and $\mathbf{k_2}$
- Then rotate between $\mathbf{k_2}$ and $\mathbf{k_3}$

INSERT CASE # 3 RIGHT LEFT



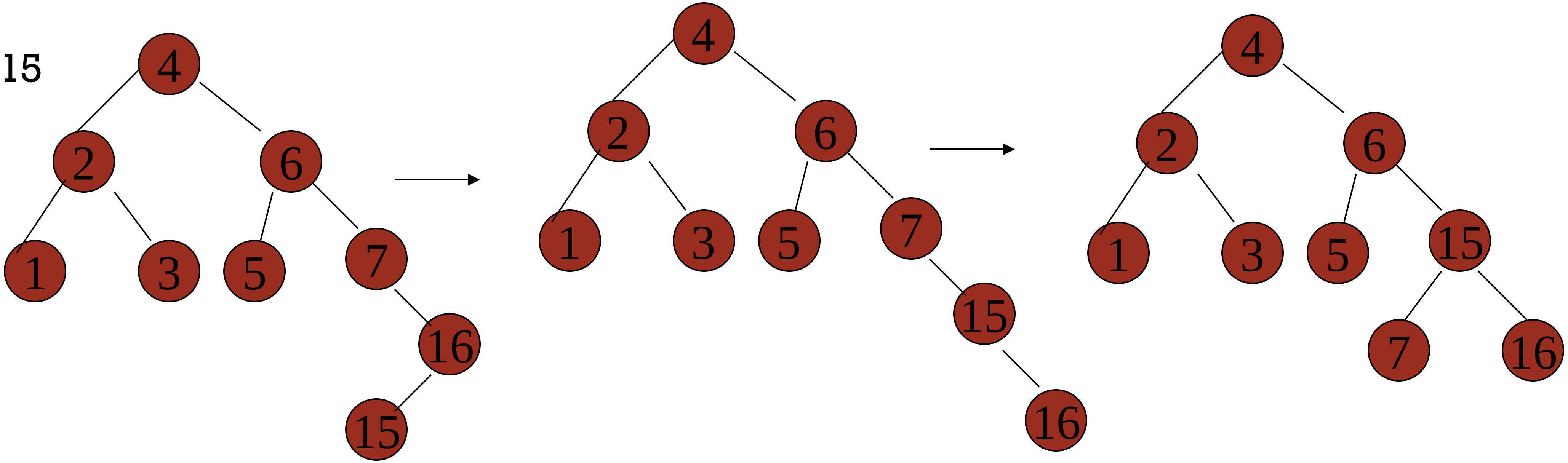
INSERT CASE #4 LEFT RIGHT



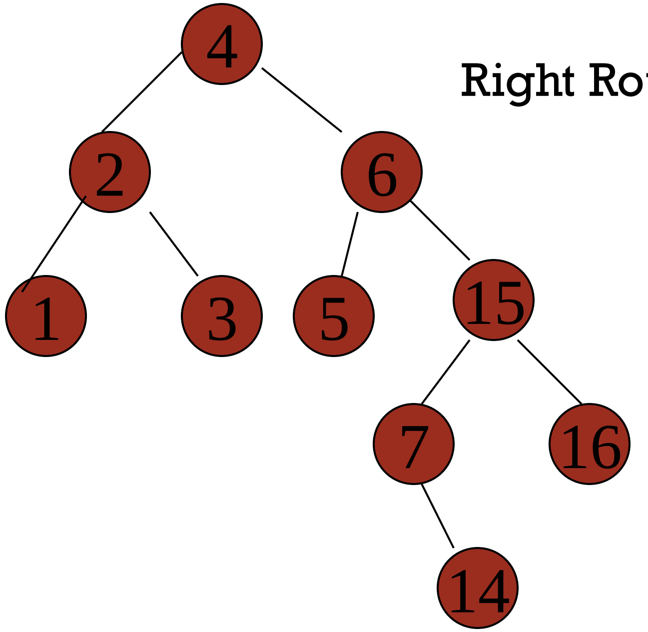
EXAMPLE

- Continuing the previous example by inserting
  - 16 down to 10, and then 8 and 9

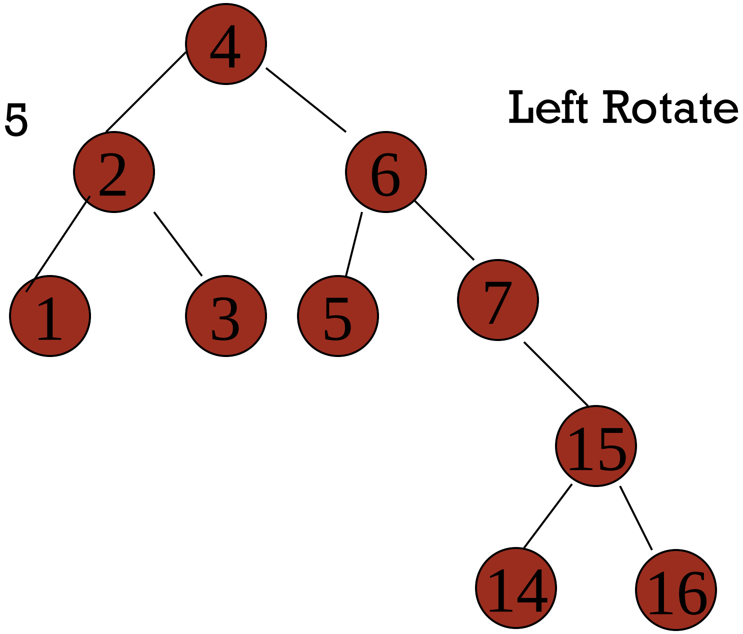
- Inserting 16 and 15



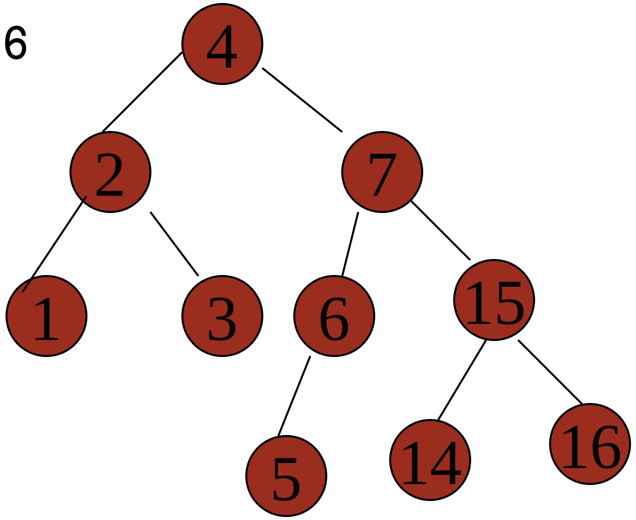
- Inserting 14



Right Rotate on 15



Left Rotate on 6



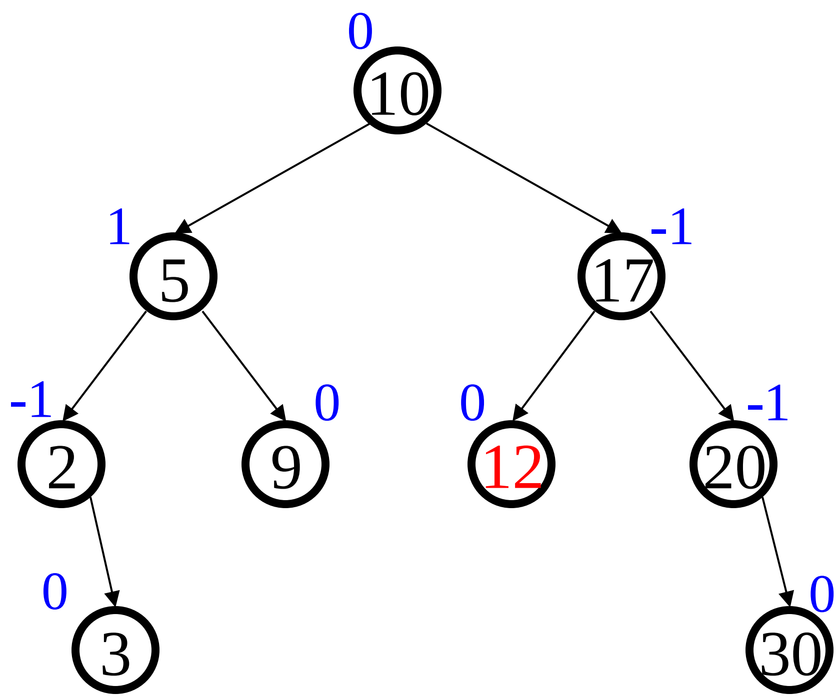
AVL DELETION ALGORITHM

Recursive

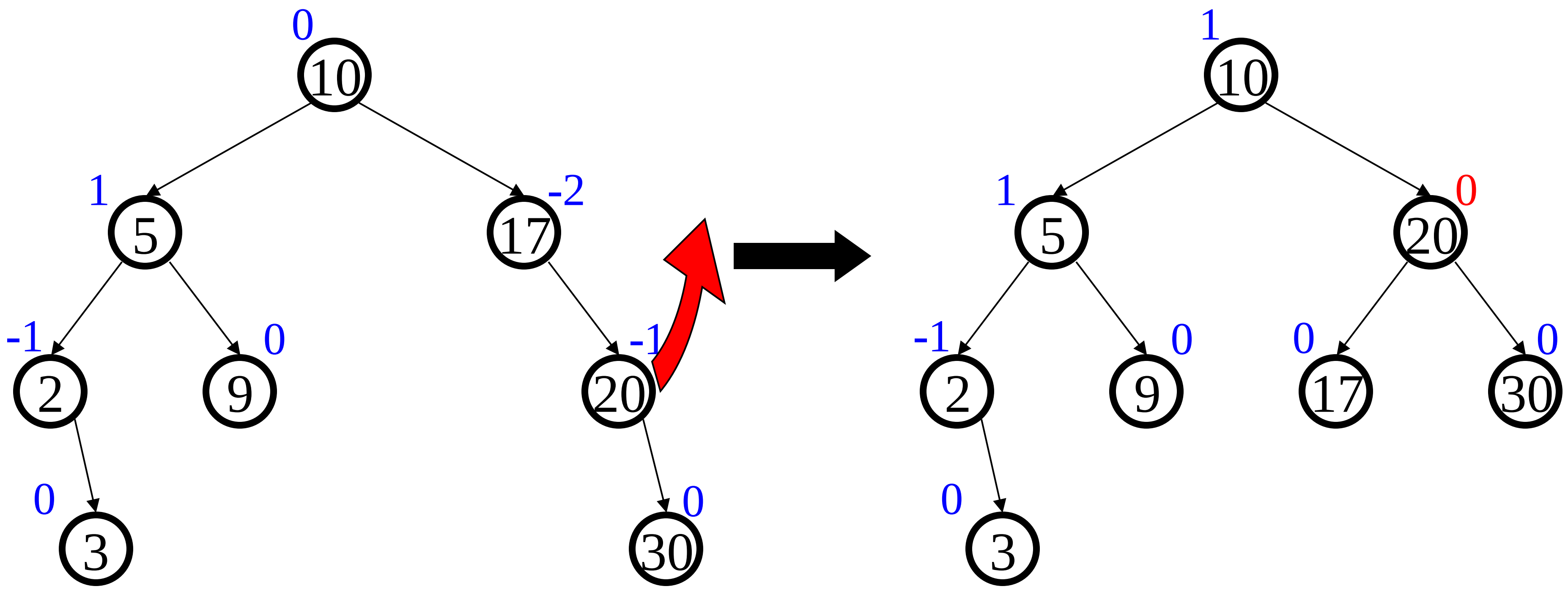
1. Search downward for node
2. Delete node
3. Unwind, correcting heights as we go
   a. If imbalance is left-left or right-right, single rotate
   b. If imbalance right-left or left-right, double rotate

DELETION CASE #1

Delete(12)



SINGLE ROTATION ON DELETION

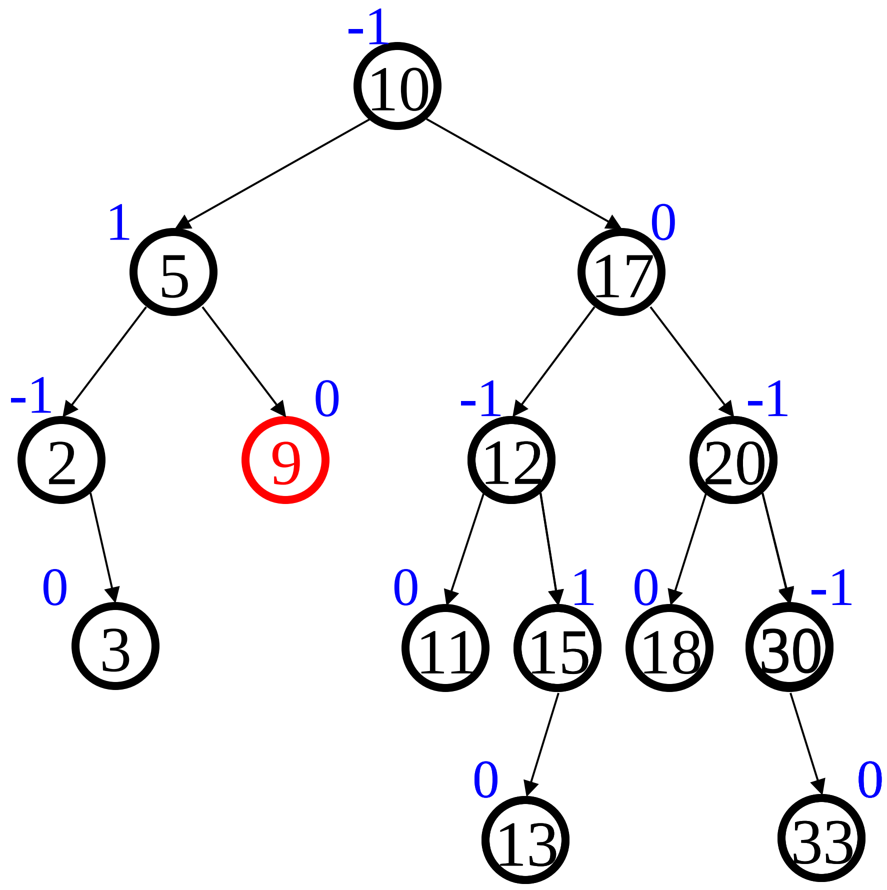


Deletion can differ from insertion – ***How?***

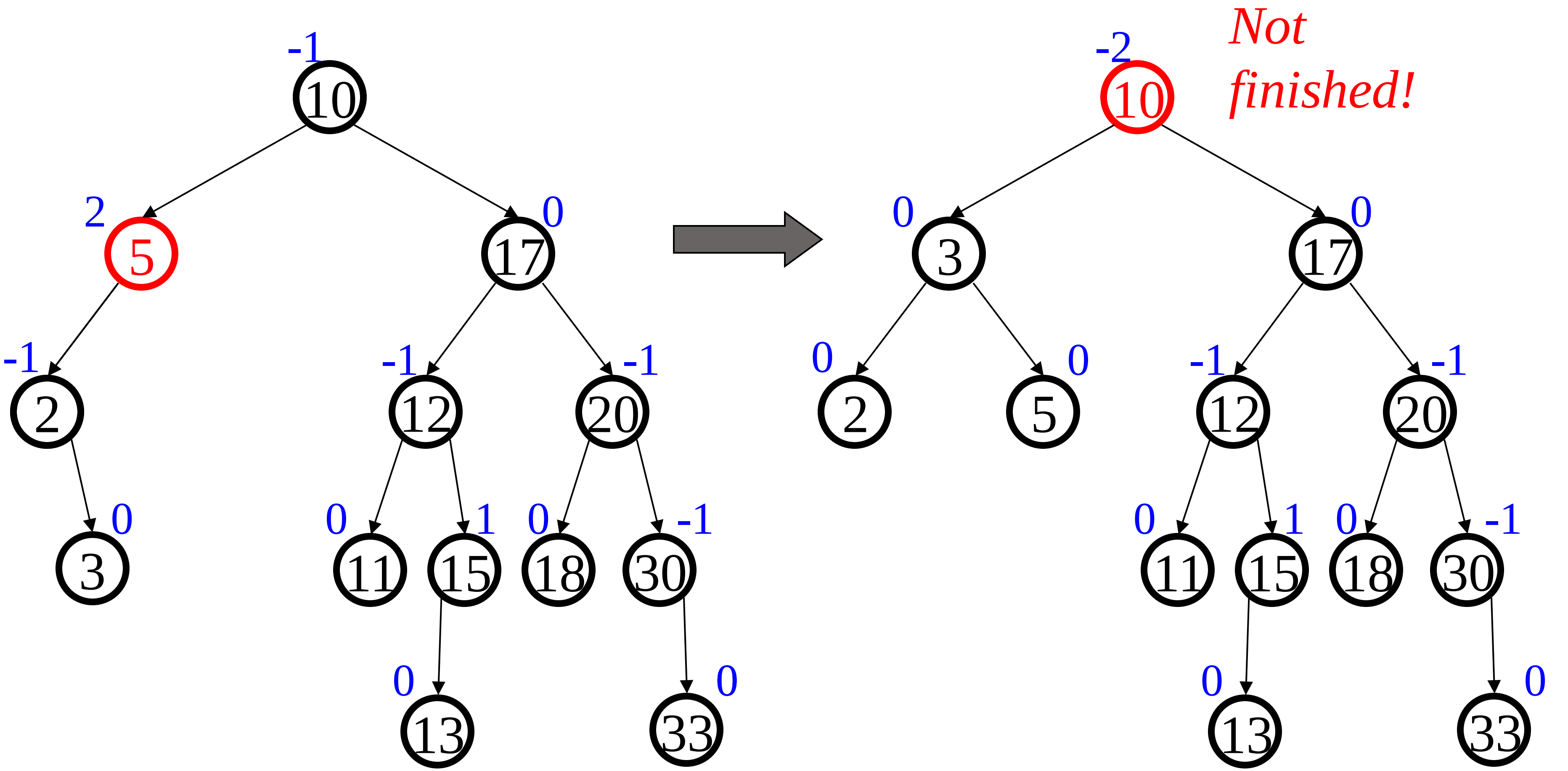
Deletion can propogate

DELETION CASE #2 - PROPOGATION

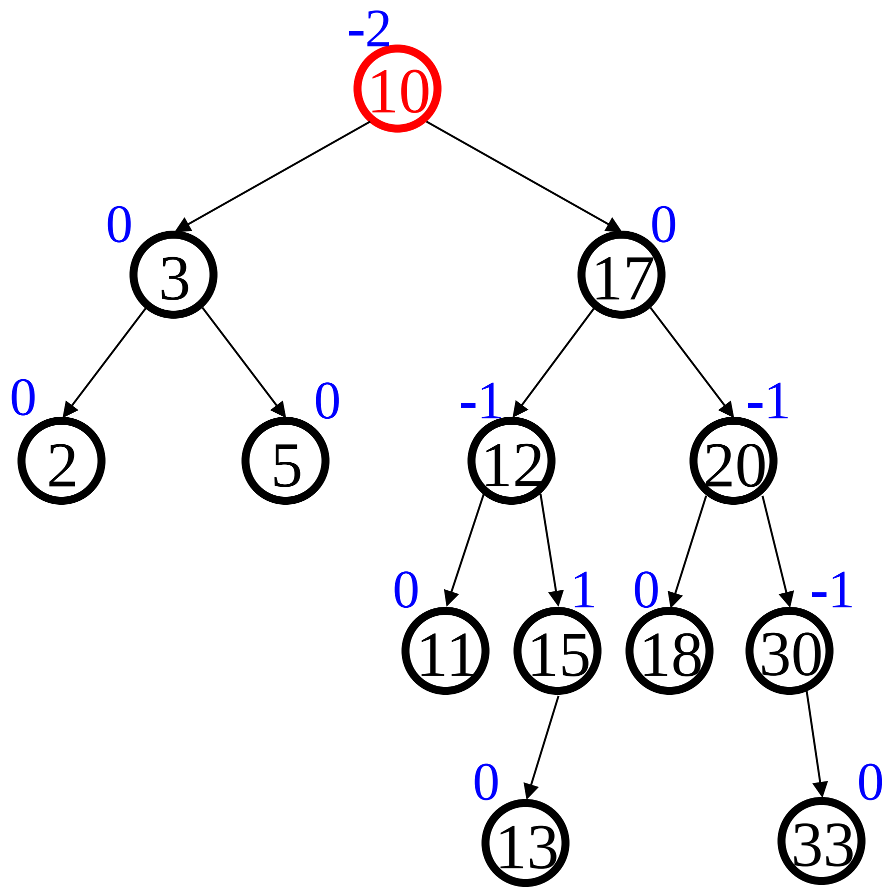
Delete(9)



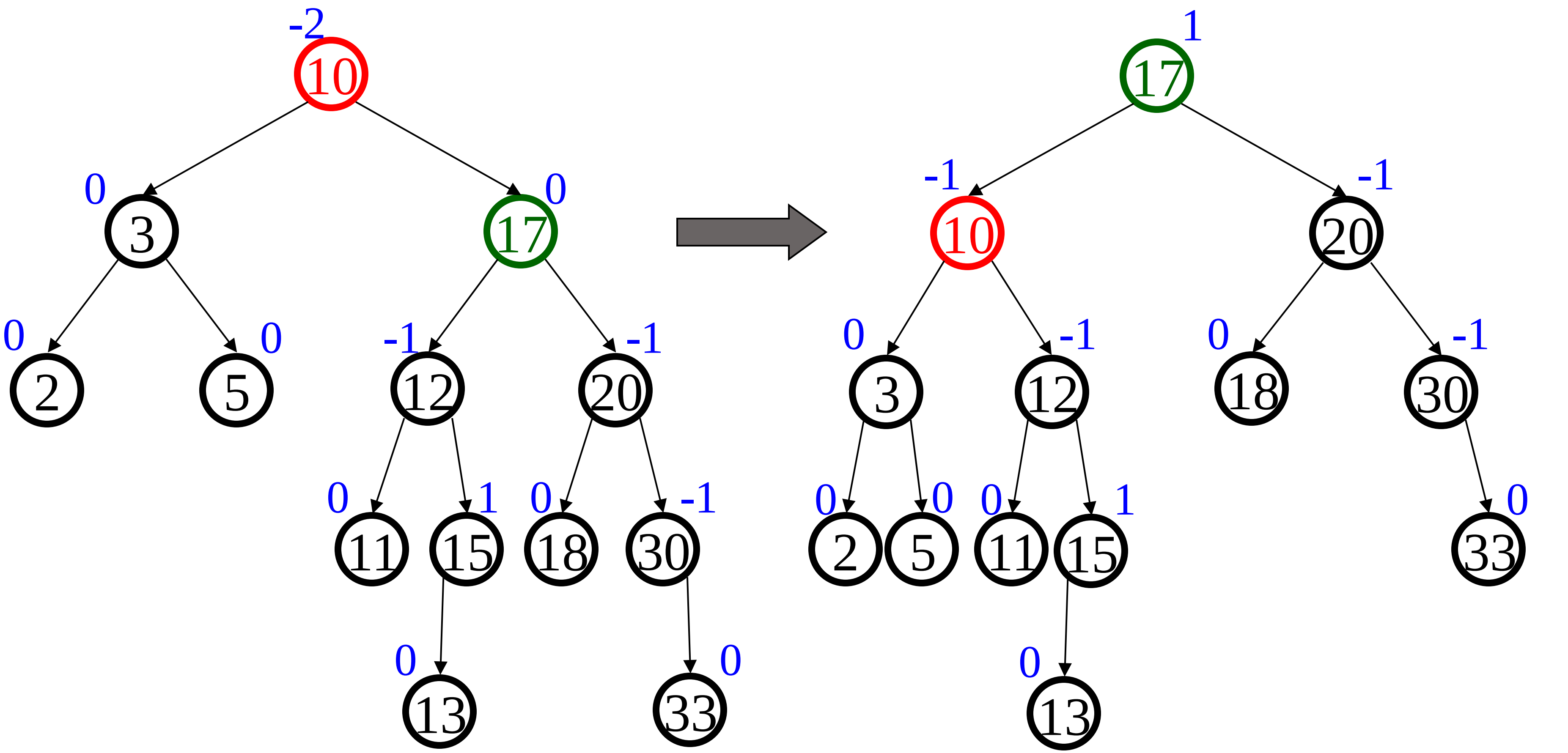
DOUBLE ROTATION ON DELETION



DELETION WITH PROPAGATION



PROPAGATED SINGLE ROTATION



PROS & CONS

Arguments for AVL trees:

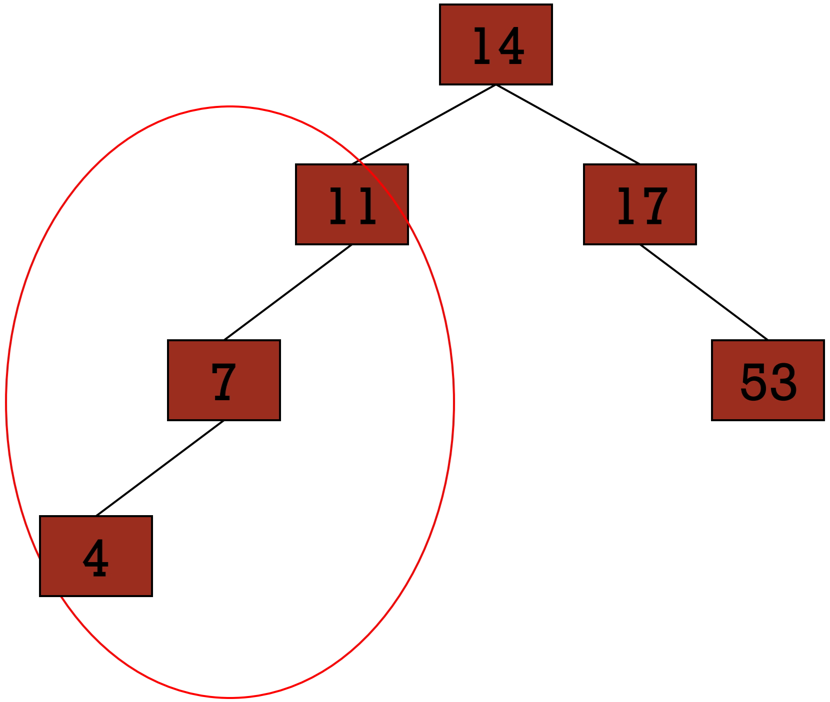
- All operations logarithmic worst-case because trees are *always* balanced
- Height balancing adds no more than a constant factor to the speed of insert and delete

Arguments against AVL trees:

- Difficult to program & debug
- More space for height field
- Asymptotically faster but rebalancing takes time

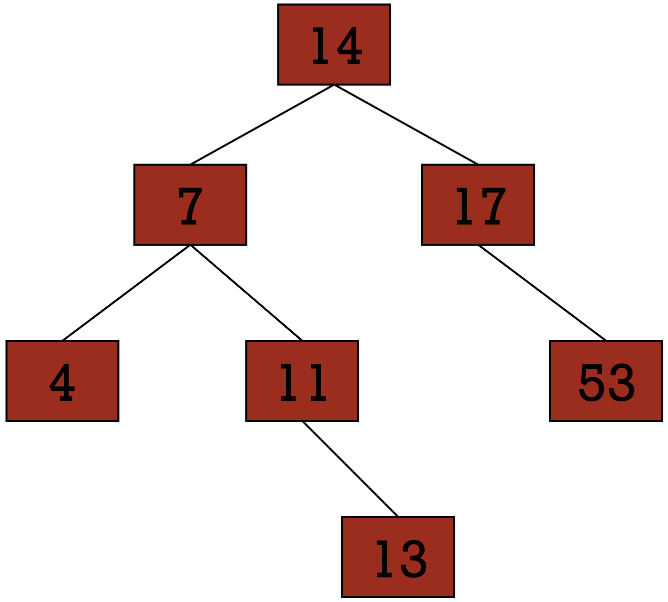
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



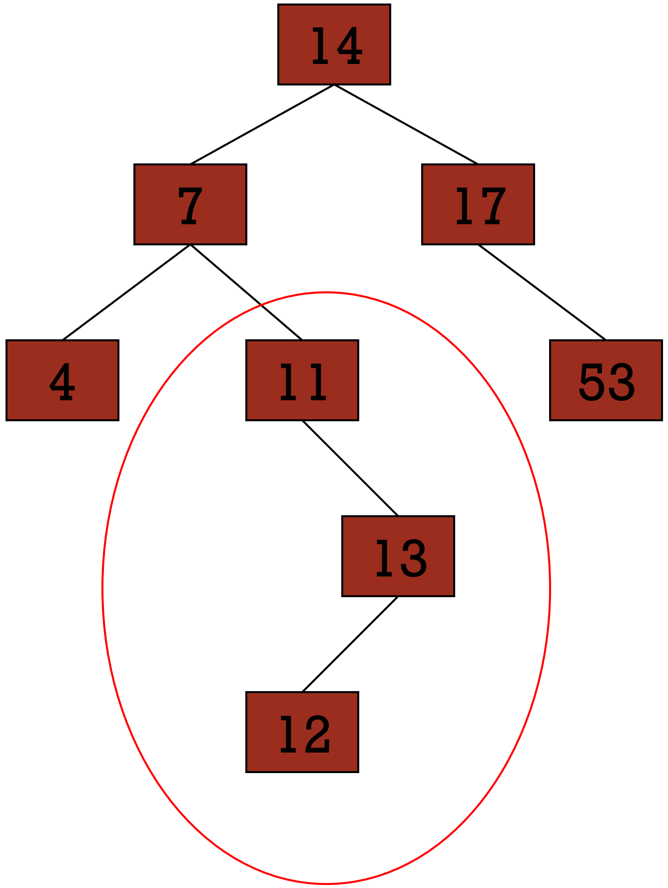
AVL Tree Example:

- **Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree**



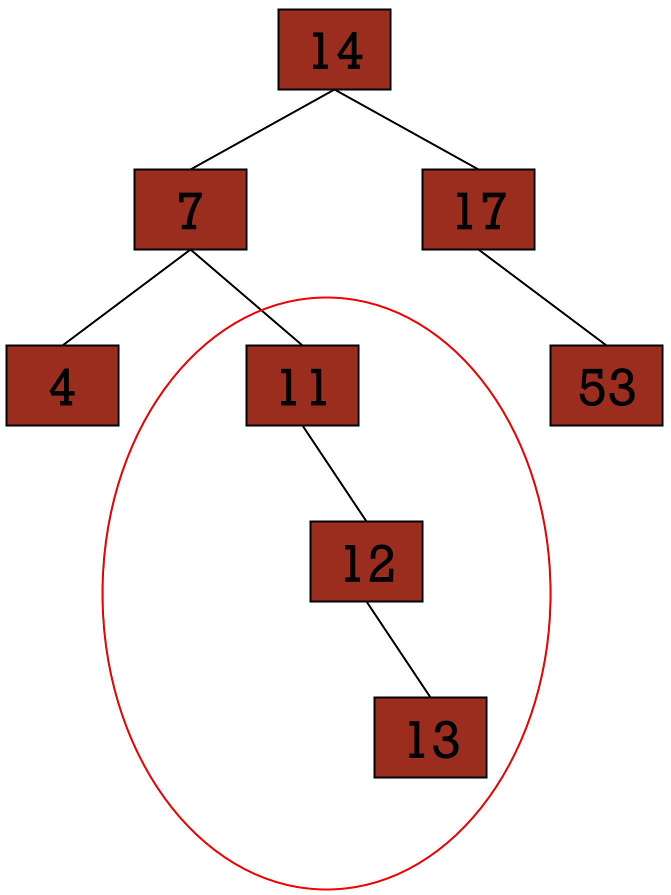
AVL Tree Example:

- Now insert 12



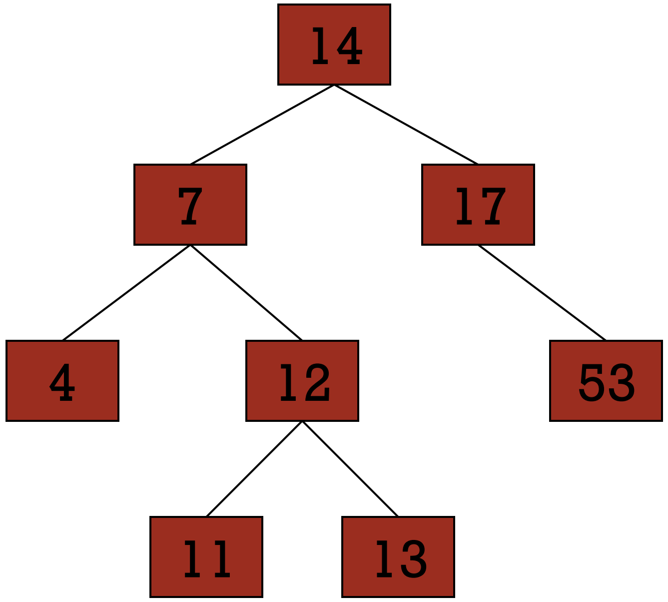
AVL Tree Example:

- **Now insert 12**



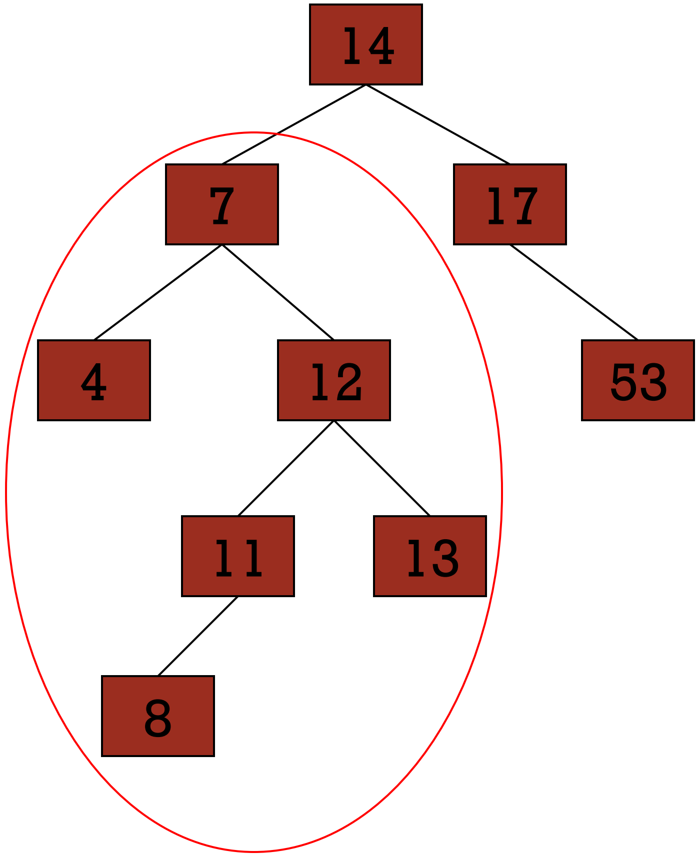
AVL Tree Example:

- **Now the AVL tree is balanced.**



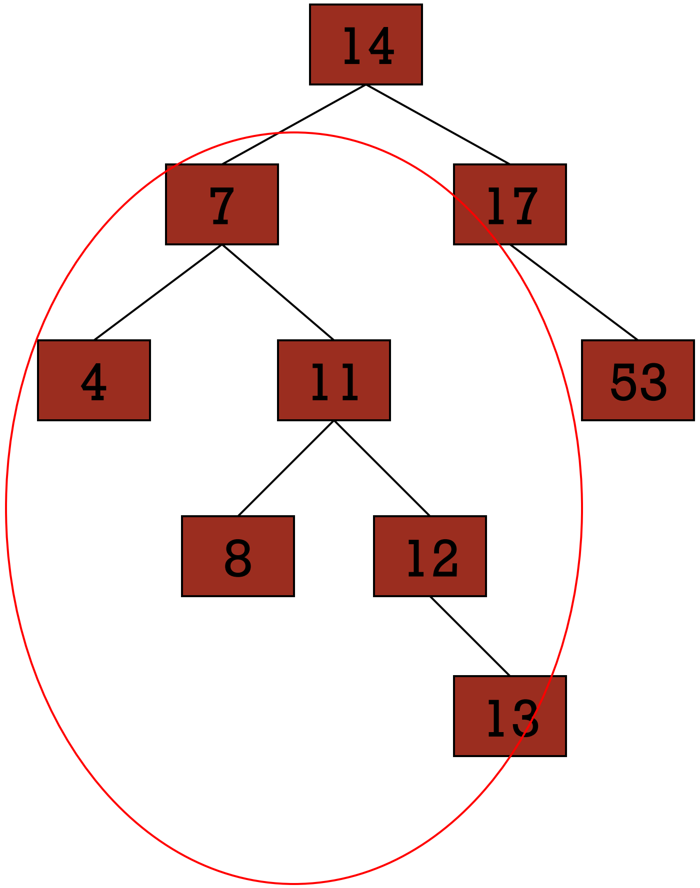
AVL Tree Example:

- **Now insert 8**



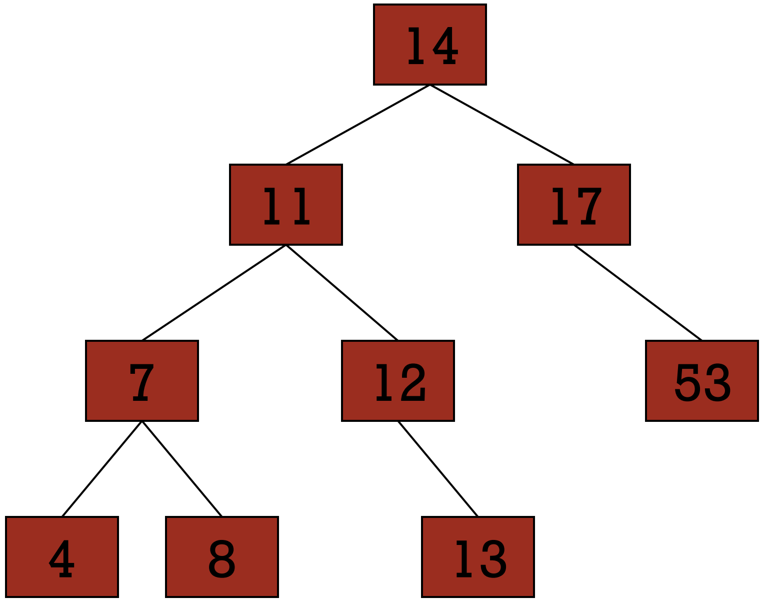
AVL Tree Example:

- **Now insert 8**



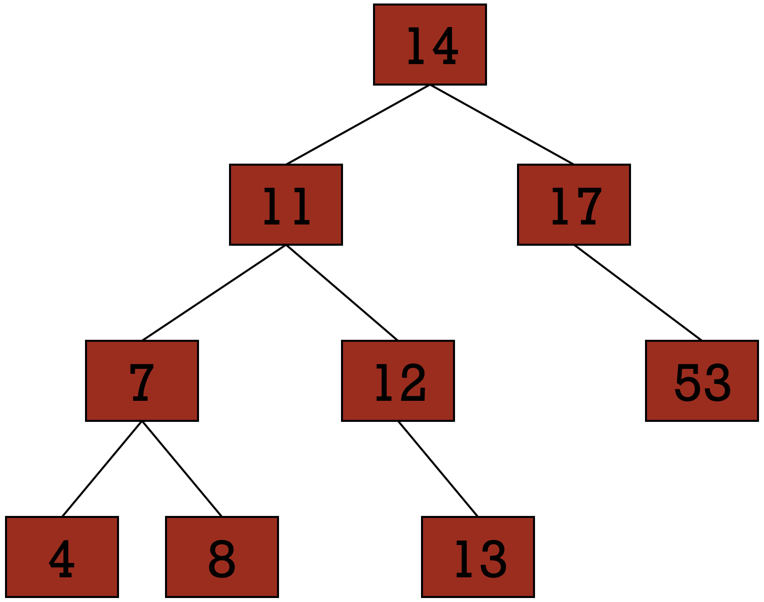
AVL Tree Example:

- **Now the AVL tree is balanced.**



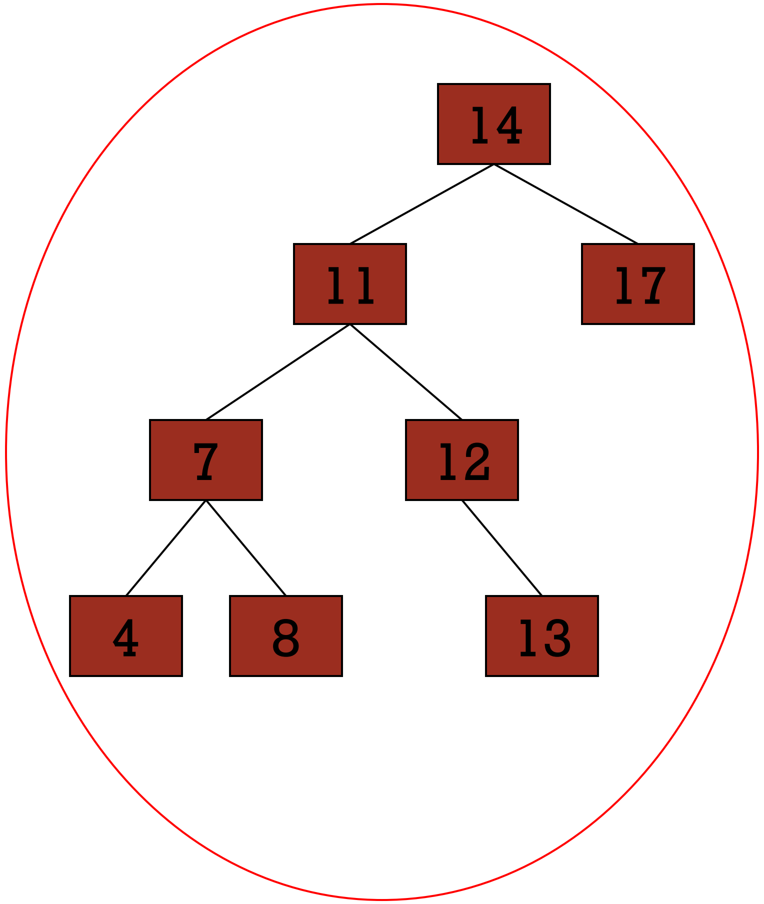
AVL Tree Example:

- **Now remove 53**



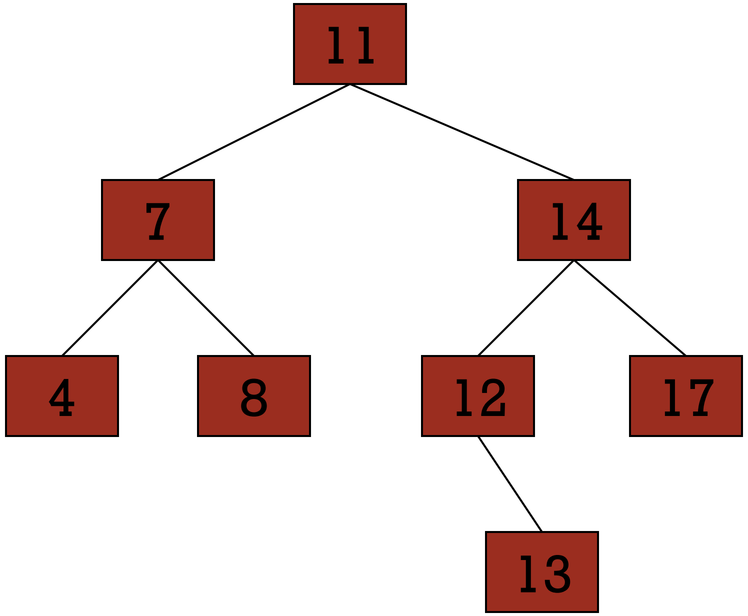
AVL Tree Example:

- **Now remove 53, unbalanced**



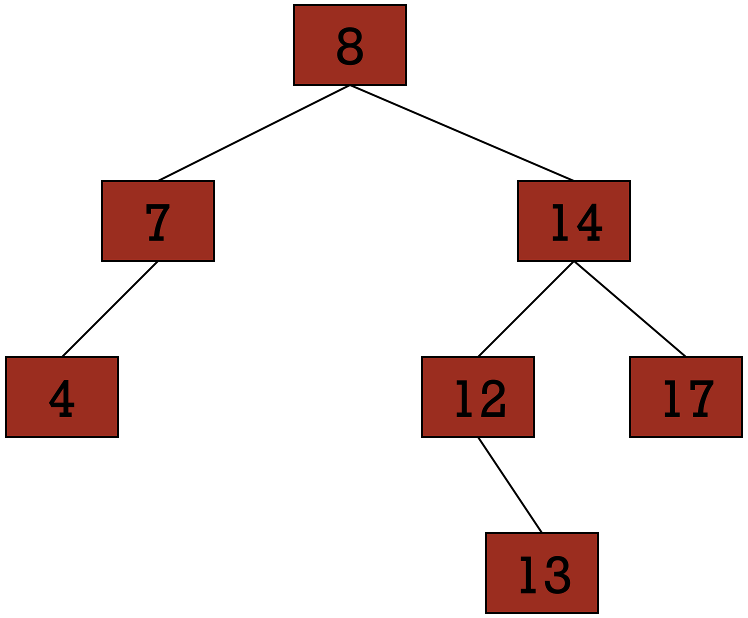
AVL Tree Example:

- **Balanced!   Remove 11**



AVL Tree Example:

- **Remove 11, replace it with the largest in its left branch**



AVL Tree Example:

- **Remove 8, replace with 12. Balanced.**