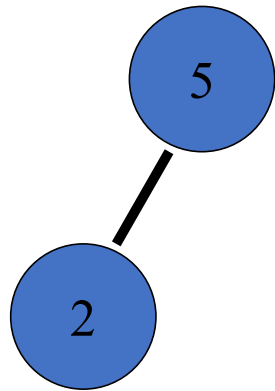


# AVL Trees

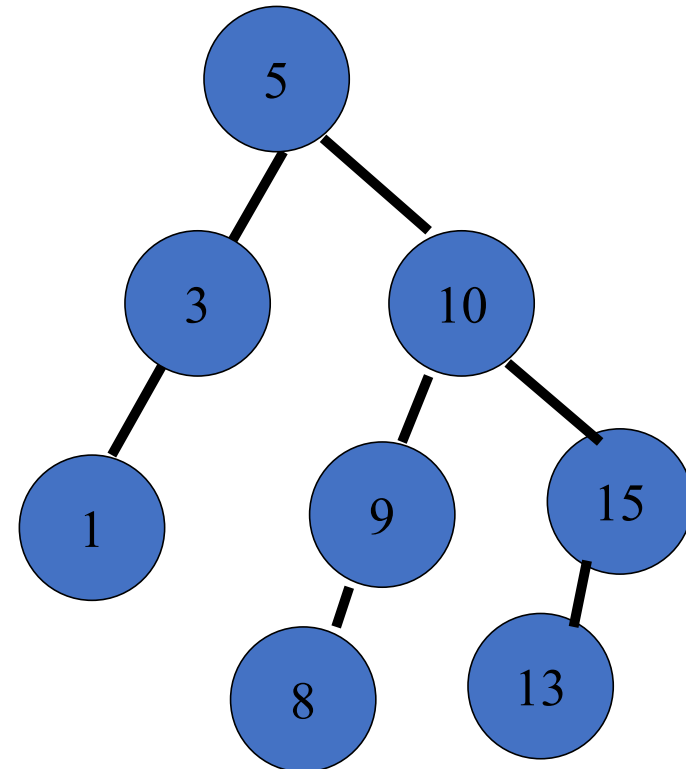
# AVL Trees

- AVL tree is a self-balancing binary search tree invented by G.M. **A**delson-**V**elsky and E.M. **L**andis in 1962
- Definition : An AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right sub-trees can be differ by at most 1.
- *Balance factor = Height (left sub-tree) – Height (right sub-tree)*
- The height of the empty sub-tree is **-1**
- Balance factor = |Height of the left sub-tree – height of the right sub-tree|
- **-1 ≤ balance factor ≤ 1**

# AVL Trees ?

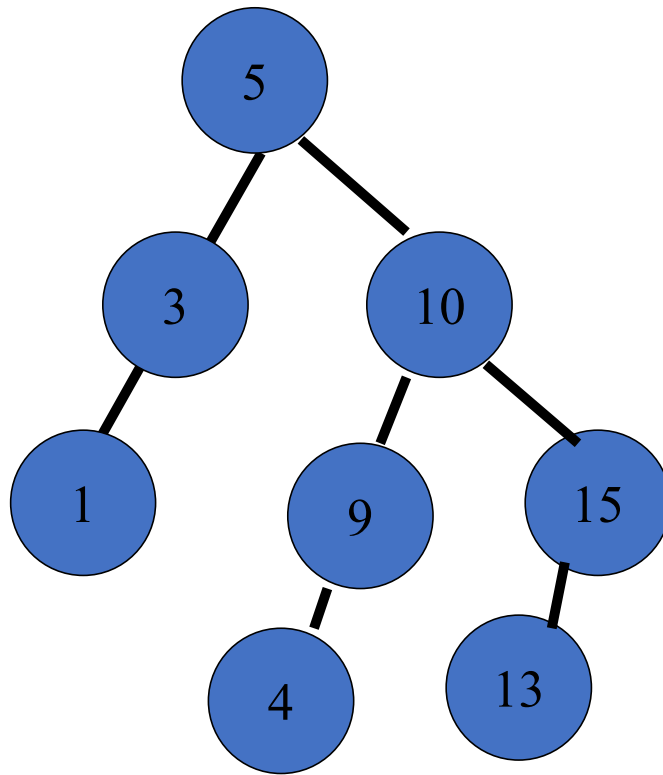


Balance factor = 1

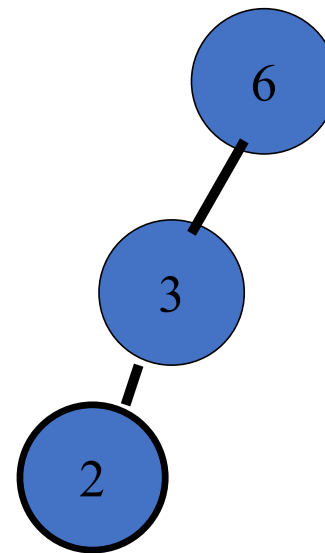


Balance factor = -1

# AVL trees ?



Balance factor = -1



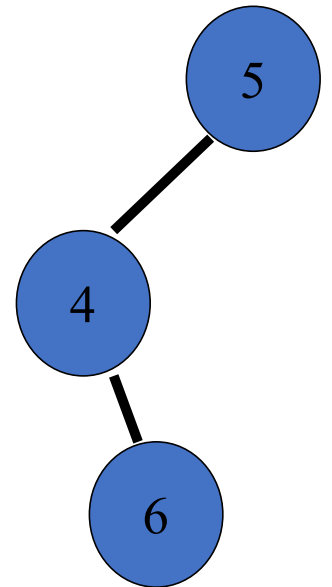
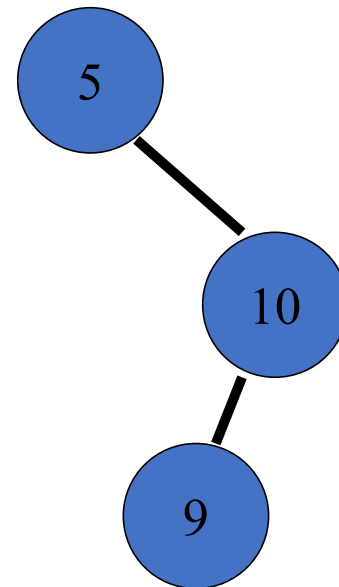
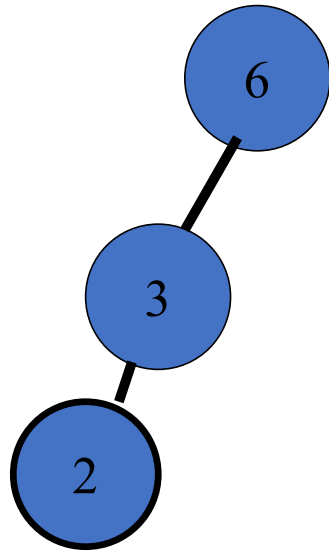
Balance factor = 2

# Algorithm : Inserting new nodes into an AVL trees

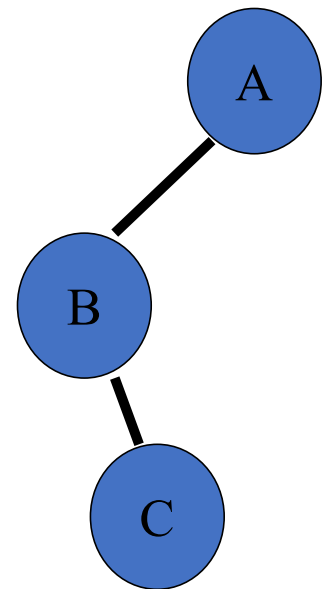
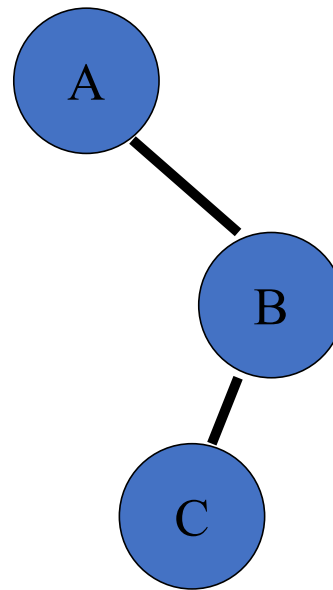
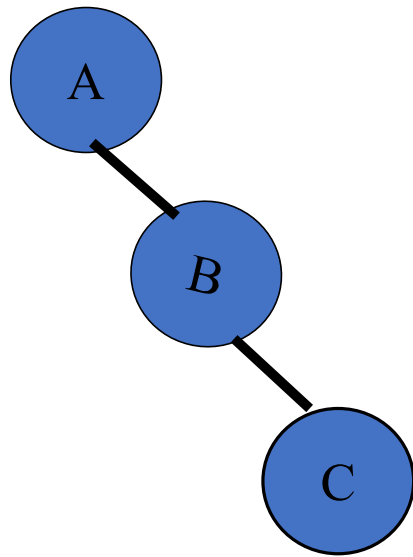
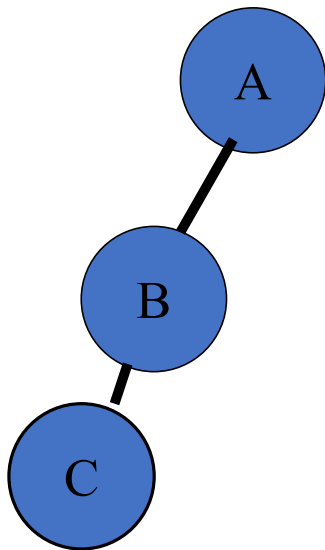
1. Insert the node in the same way as in an ordinary binary search tree.
2. Beginning with the new node, trace a path back towards the root, checking the difference in height of the sub-trees at each node along the way.
3. If you find a node with an imbalance ( a height difference other than 0,+1,-1), stop your trace at this point.
4. Consider the node with the imbalance and two nodes on the layers immediately below this point on the path back to the new node.

# Algorithm : Inserting new nodes into an AVL trees

5. If these three nodes lie in a straight line, apply a **single rotation** to correct the imbalance.
6. If these three nodes lie in a **dog-leg pattern**, apply a double rotation to correct the imbalance.

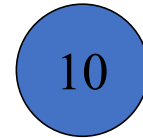


# Single VS. Double Rotations

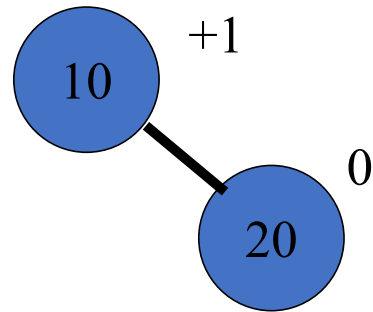


# Construction of an AVL tree.

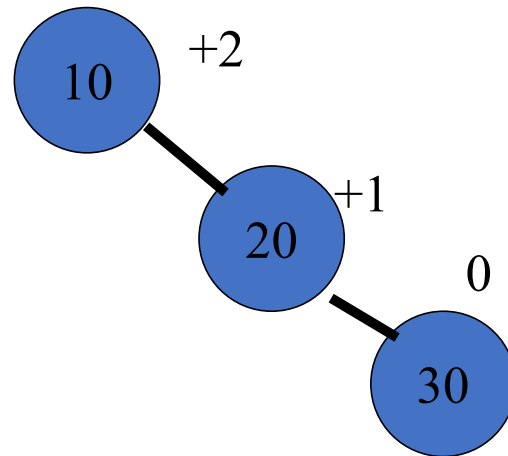
- Eg. 10,20,30,25,27,7,4,23,26,21
- We begin by inserting integer 10 into the root
- This node satisfies the AVL condition.



Now we add another node 20



Now we add third node, 30



The root node 10 has a difference of 2, which violates the AVL conditions.

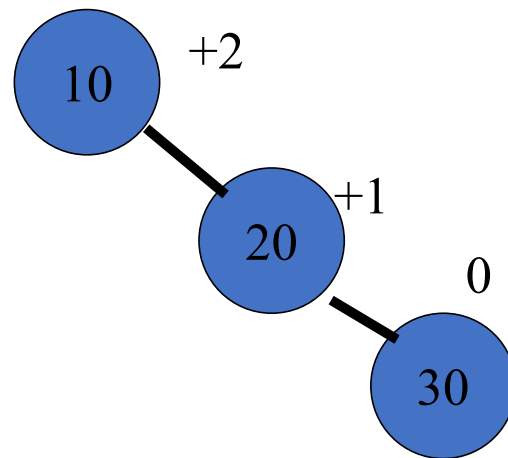
Therefore, we must re-arrange the node to restore the balance in the tree.

We perform an operation known as the rotation when the tree goes out of balance. There are two types of rotation used in AVL trees. **Single rotation and Double rotation.**

The rules for deciding which type of rotations to be used.

1. When you have found the first node is out of balance, restrict your attention to that node and the two nodes in the two layers immediately below it.
2. If these three nodes lie in a **straight line**, **single rotation** is needed to restore the balance.
3. If these three nodes lie in a **“dog-leg”** pattern You need **double rotation** to restore the balance.

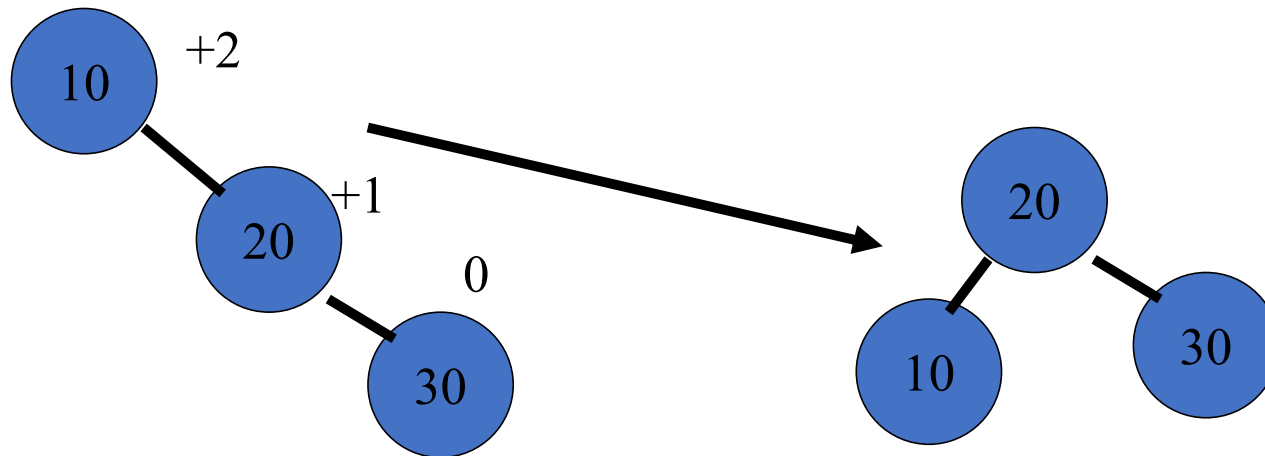
- In our example. Node 10 is imbalance , consider two layers immediately below this node. These nodes lie in a straight line. So we need a single rotation to restore the balance .

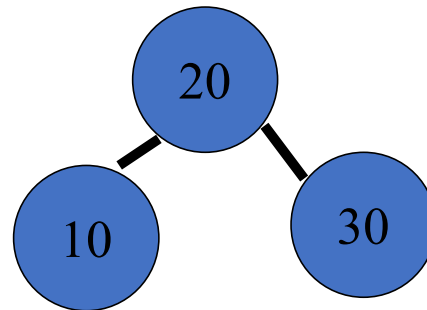


The root node 10 has a deference of 2, which violates the AVL conditions.

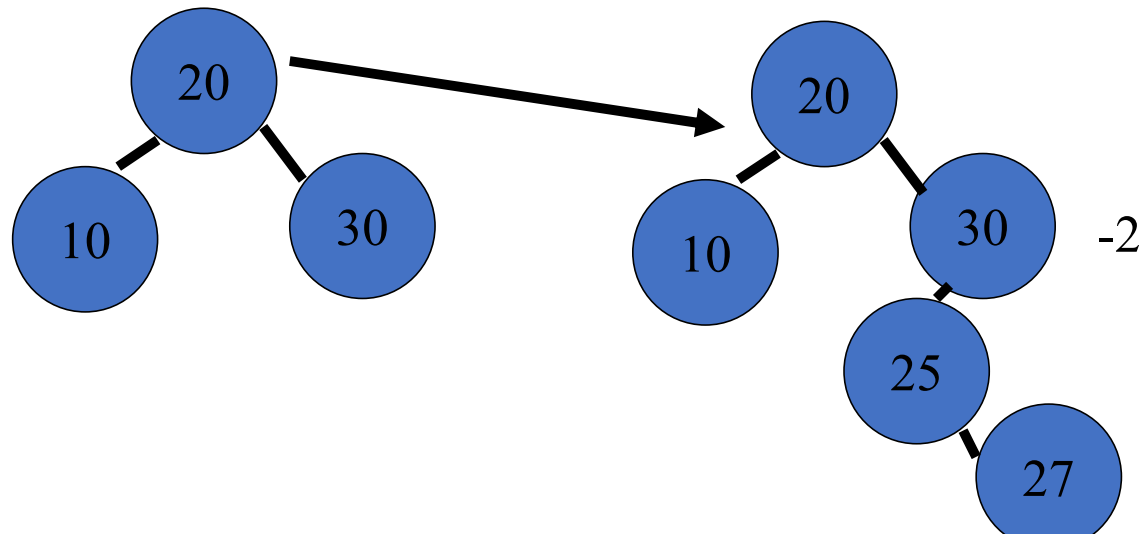
# Single Rotation :

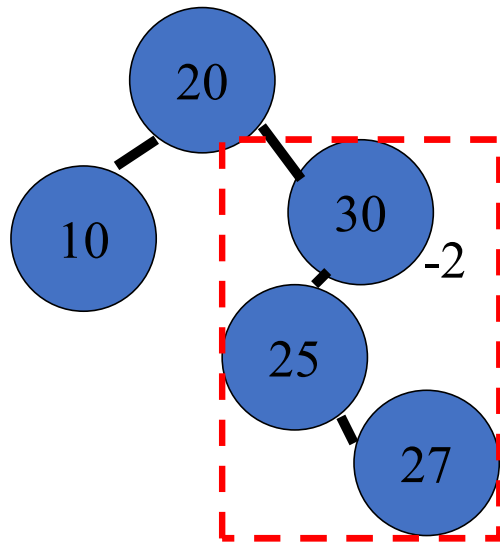
- This involves shifting the middle node up to replace the top node down to become the left child of the middle.





We continue by adding two more nodes 25 & 27



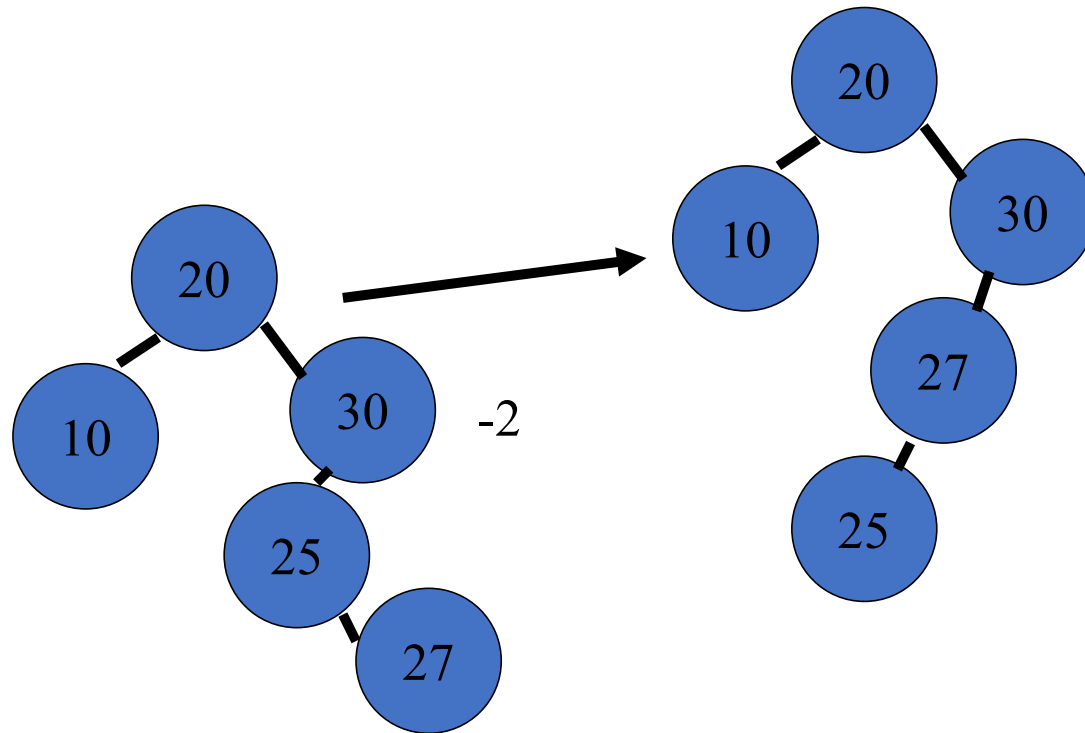


The first imbalance is detected at node 30 these three nodes form a dog-leg pattern.

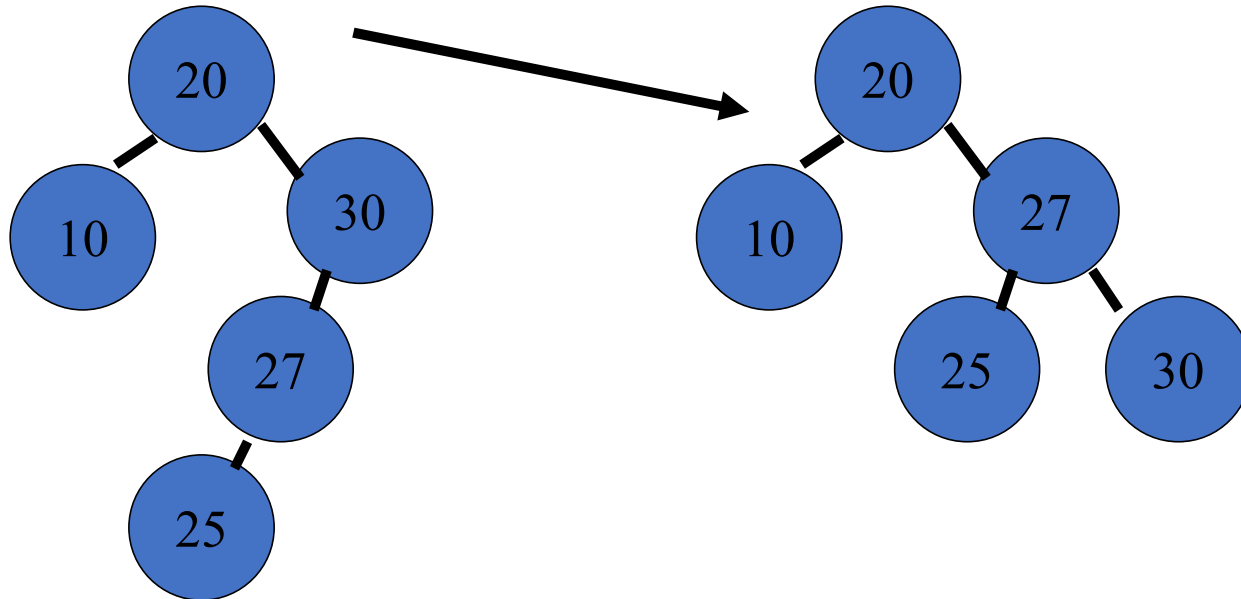
We require a double rotation to correct the balance.

Double rotation consists of two single rotations. These two rotations are in opposite directions.

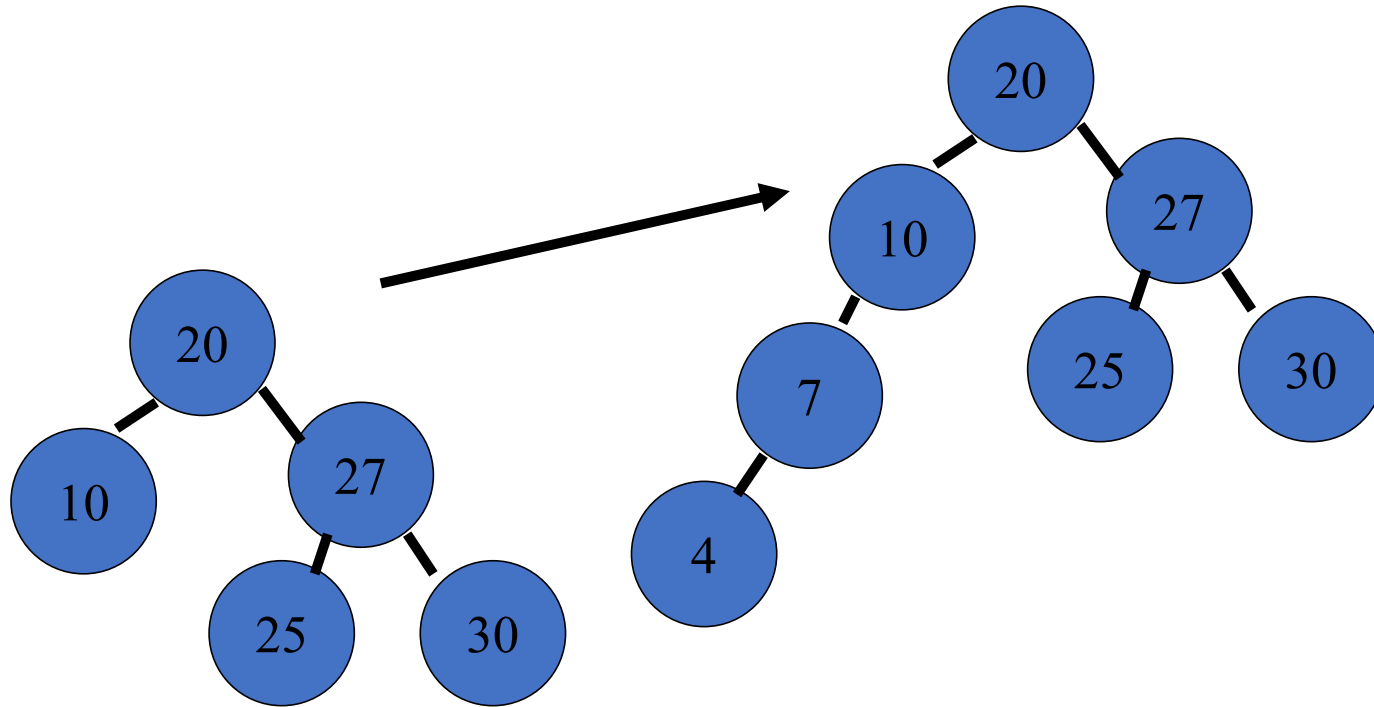
**1<sup>st</sup> Rotation**: we rotate the node 27 up to 25, and 25 down to become the left child of 27.



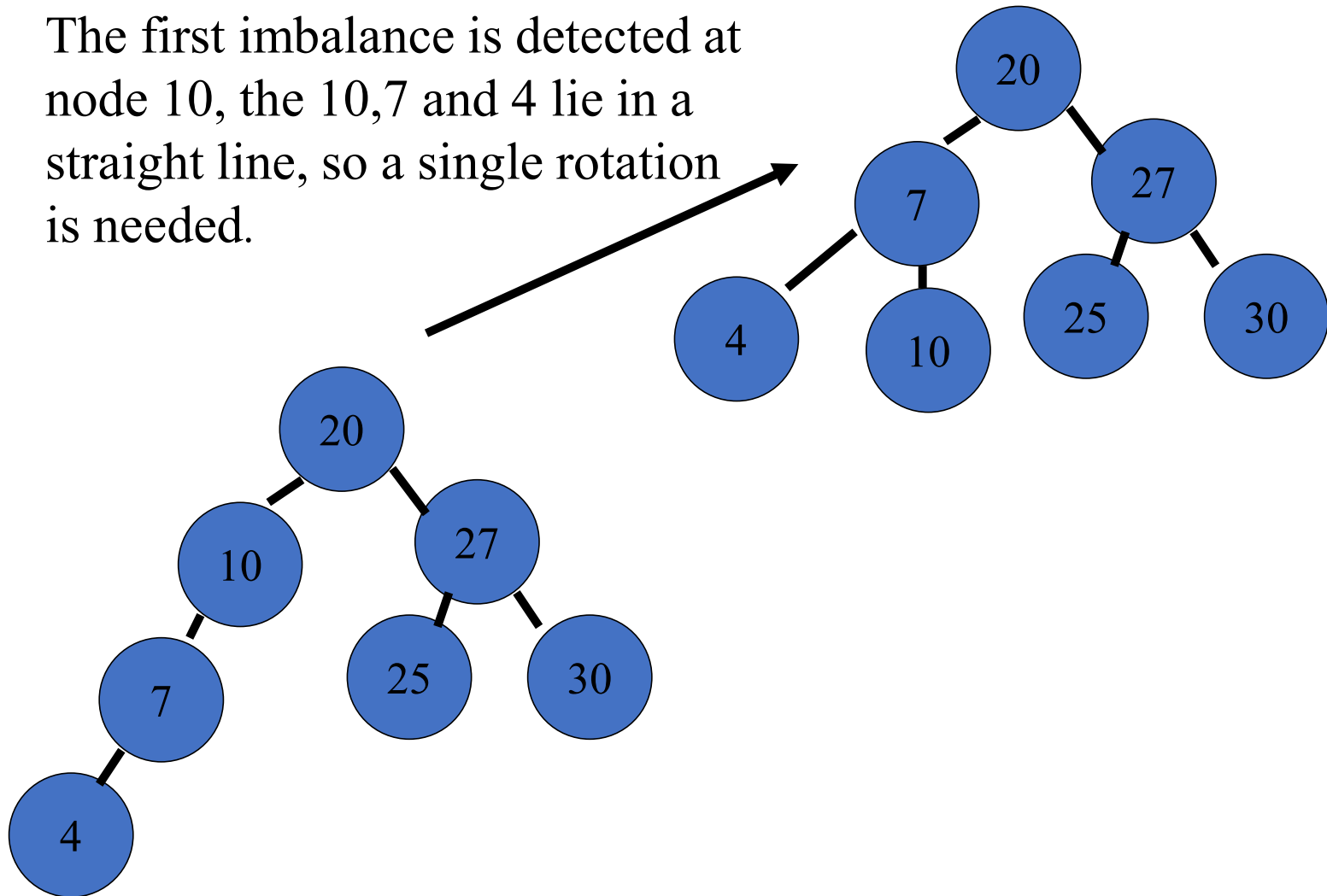
**2<sup>nd</sup> Rotation** : Node 27 rotates up to replace 30 and node 30 rotates down to become the left child of 27



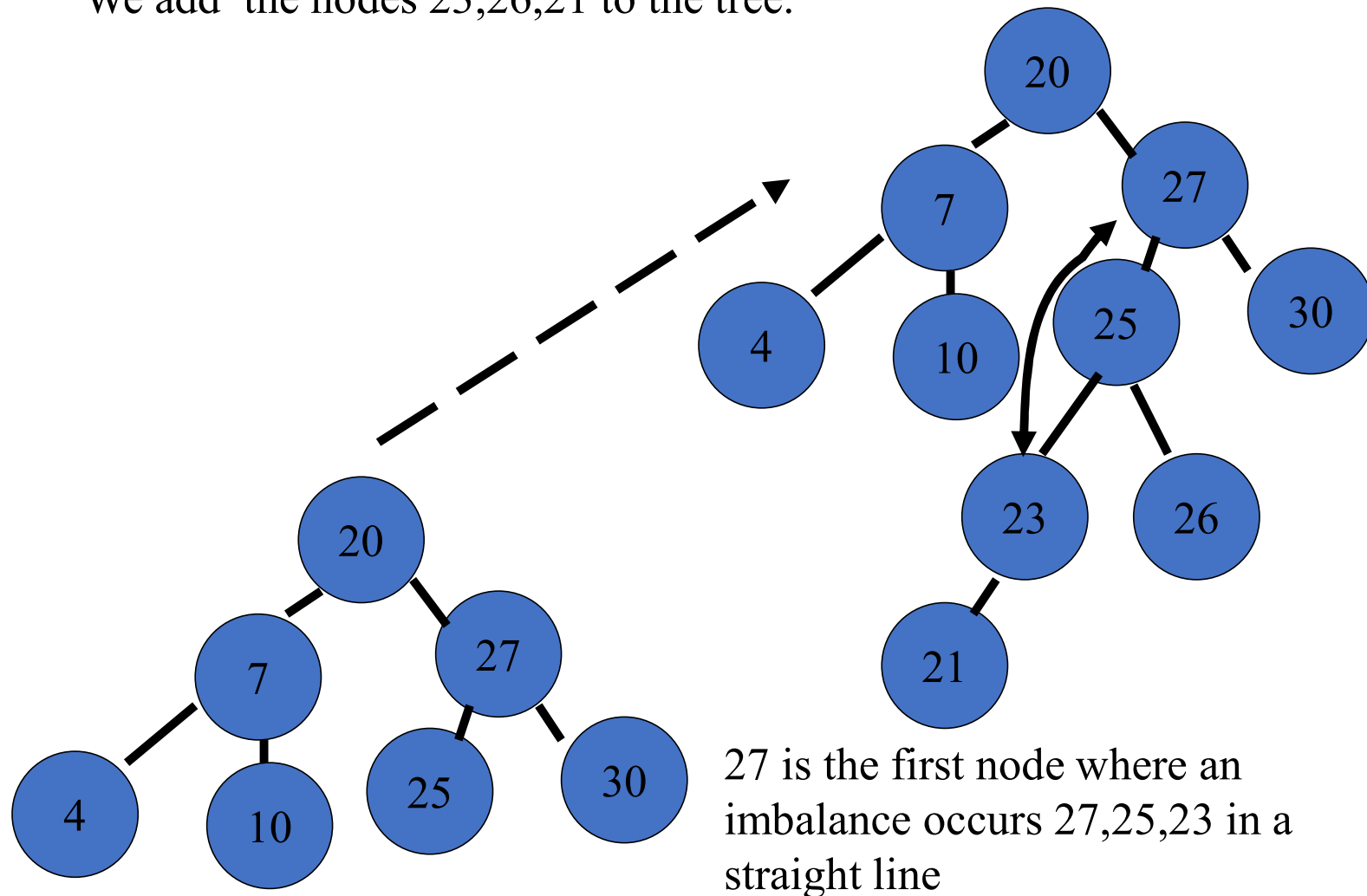
We continue by adding the nodes 7 and 4 to the tree.



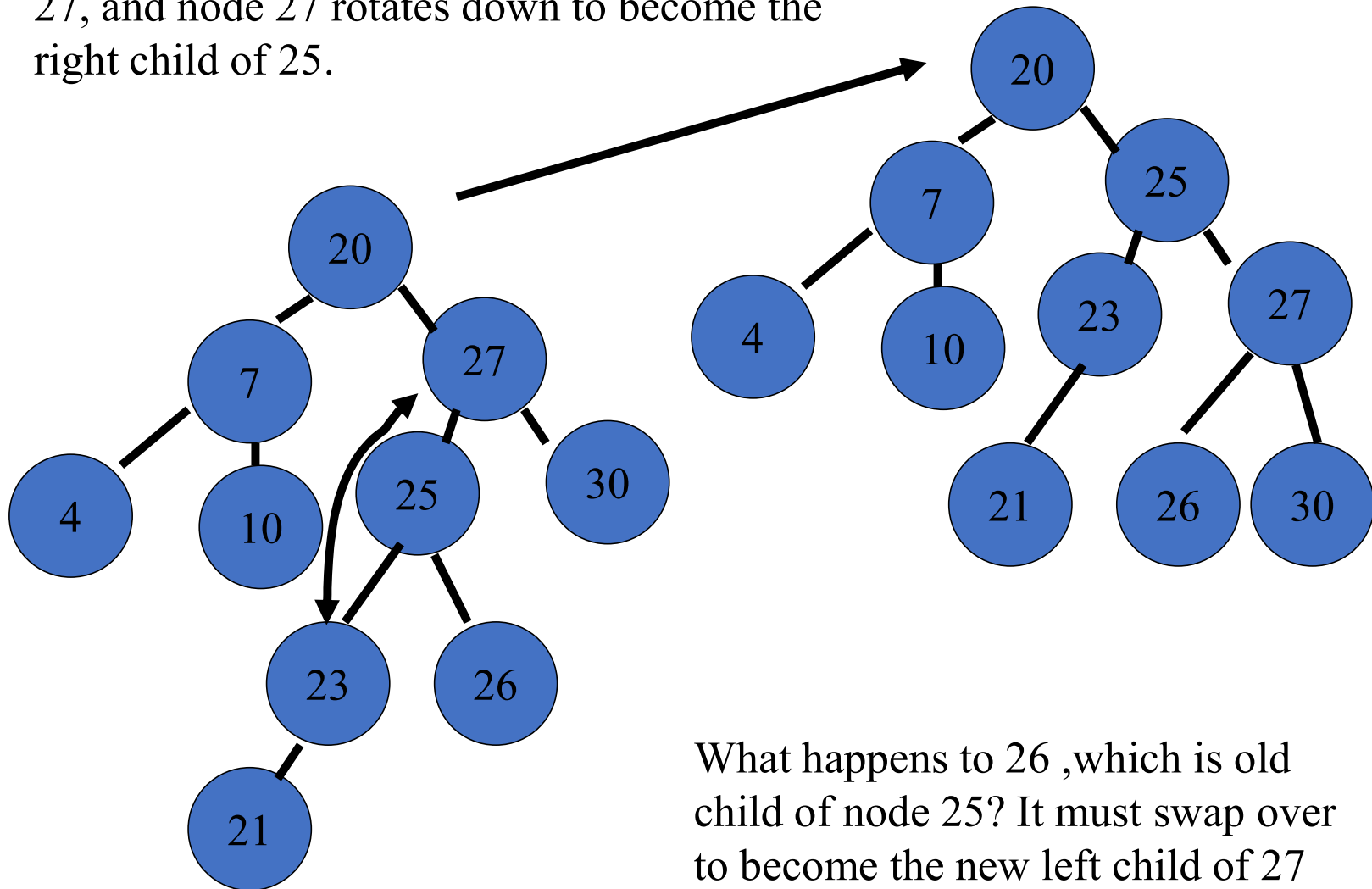
The first imbalance is detected at node 10, the 10,7 and 4 lie in a straight line, so a single rotation is needed.



We add the nodes 23,26,21 to the tree.

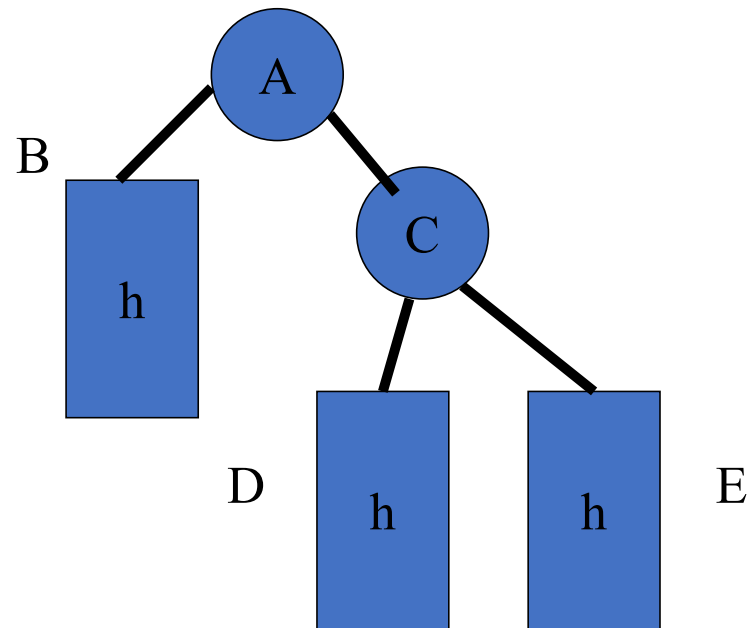


The middle node (25) rotate up to replace node 27, and node 27 rotates down to become the right child of 25.



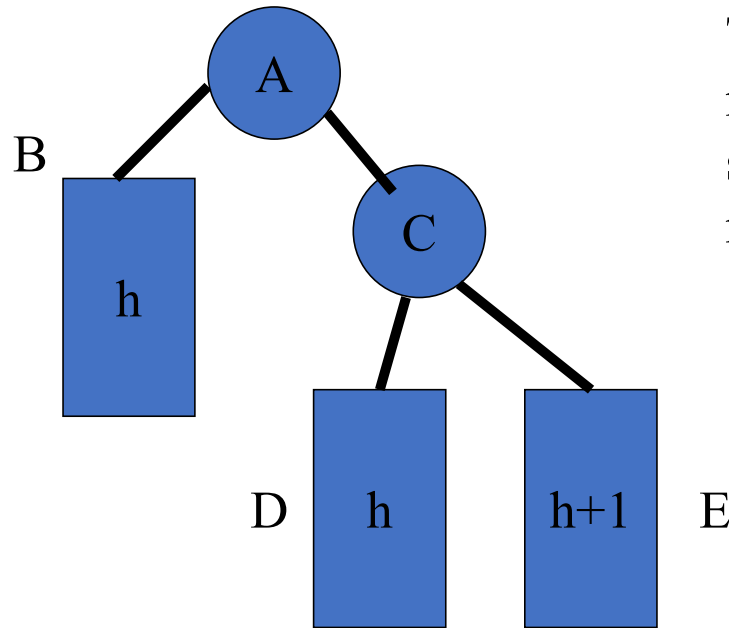
What happens to 26 ,which is old child of node 25? It must swap over to become the new left child of 27

## Single Rotation



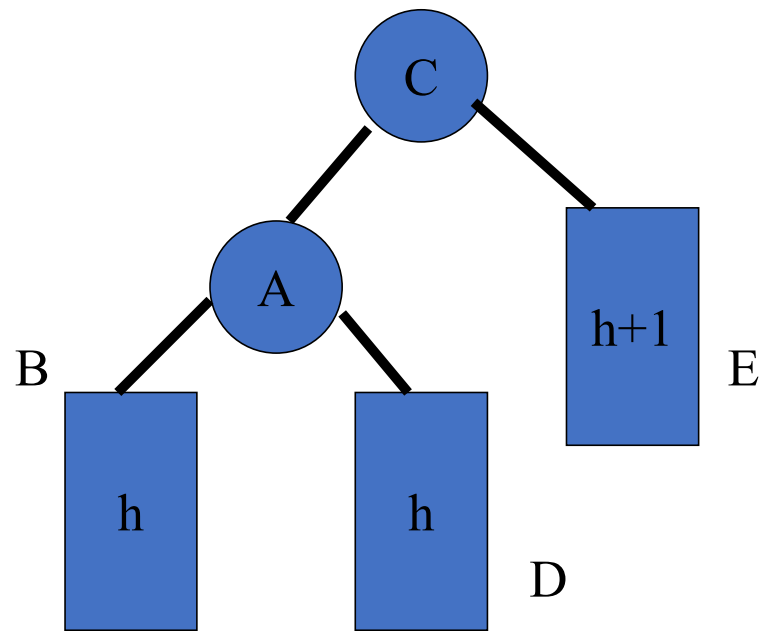
Suppose we want to insert a node into the subtree under node E

The tree is imbalance at node A



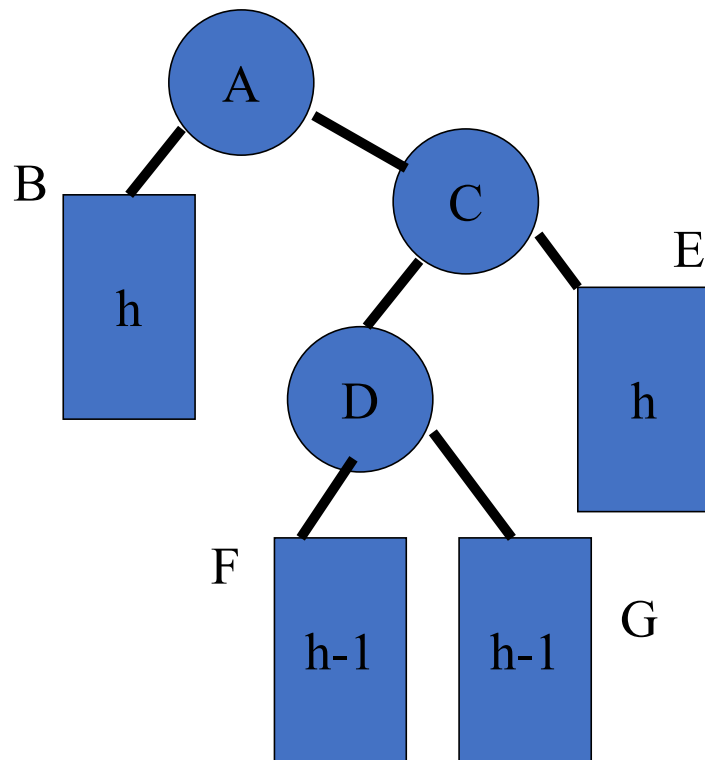
The two nodes immediately below node A. Nodes A,C,E are in the straight line. So we need a single rotation.

We rotate C up to replace A and A down become the left child of C. Node D must swap over to become the new child of A



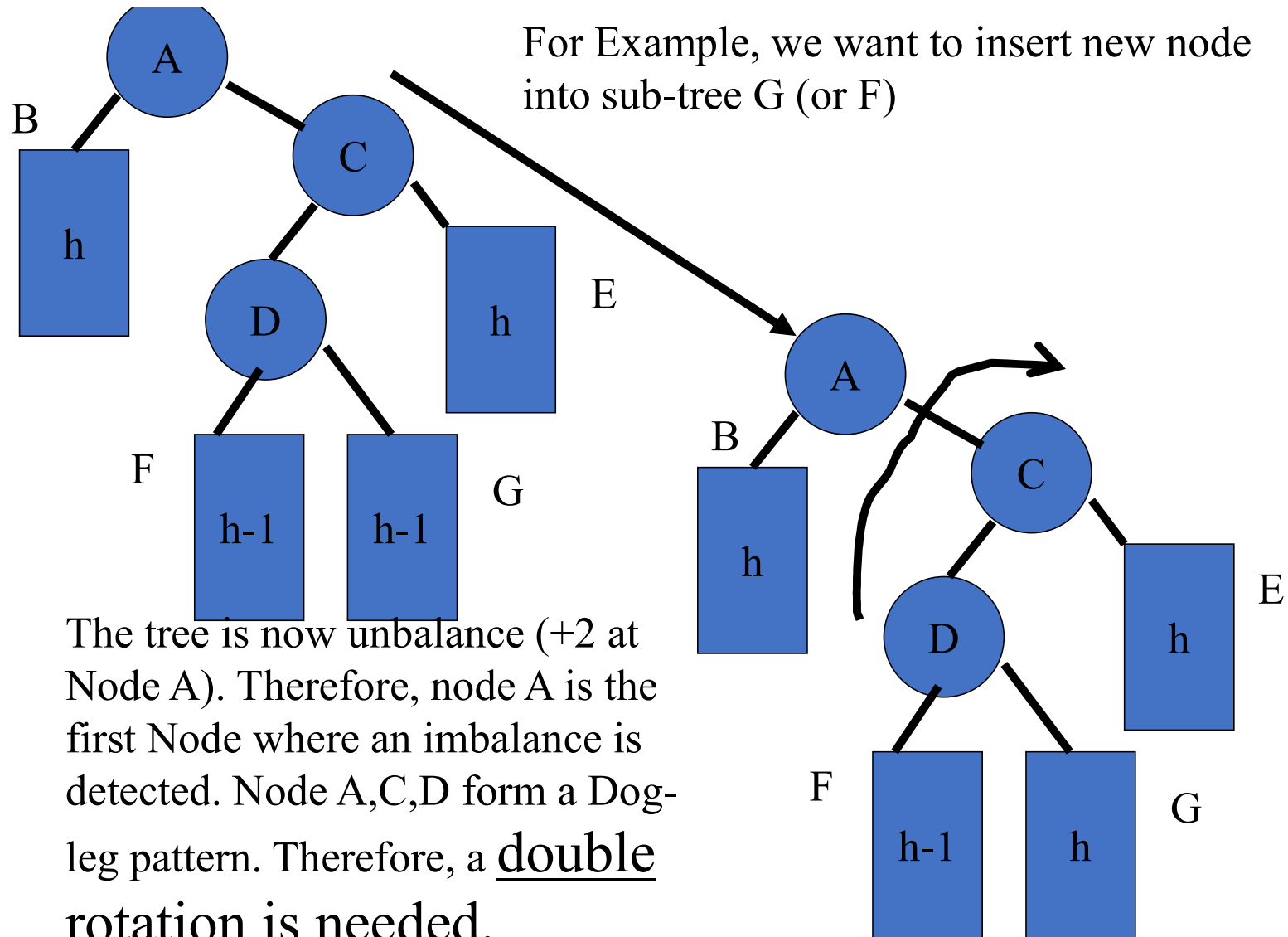
This tree is balanced.

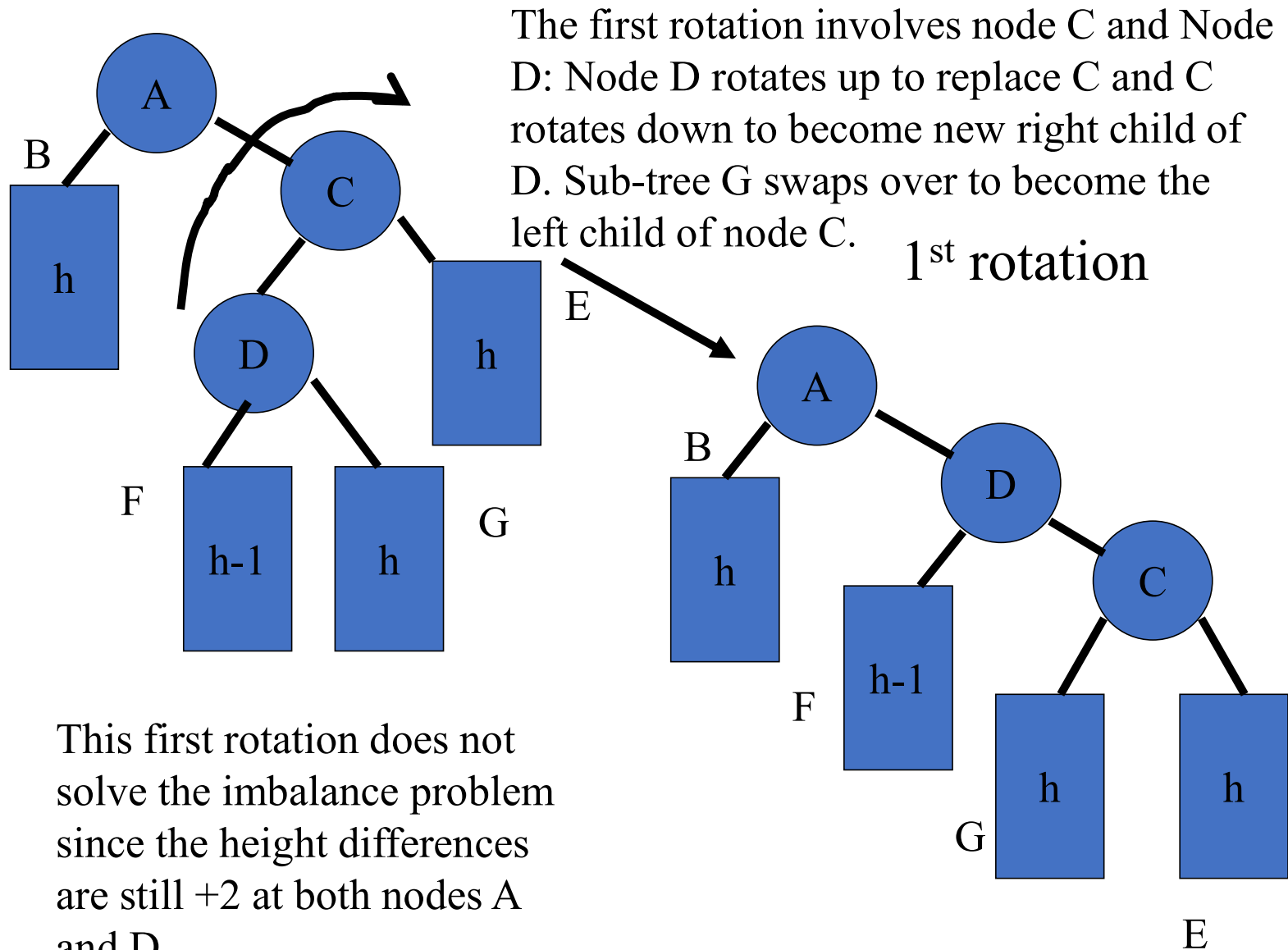
# Double Rotation

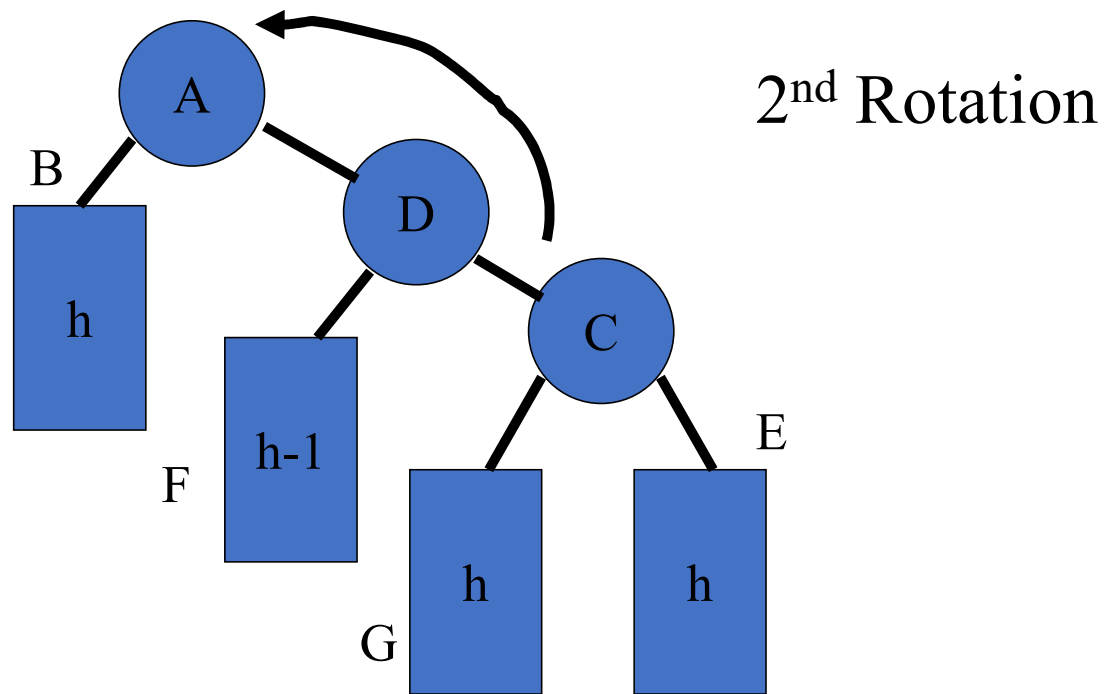


This tree is an AVL tree,  
Assuming all sub-trees in the  
rectangular boxes are AVL  
trees. The height differences  
are 0 at node D & C and +1 at  
Node A.

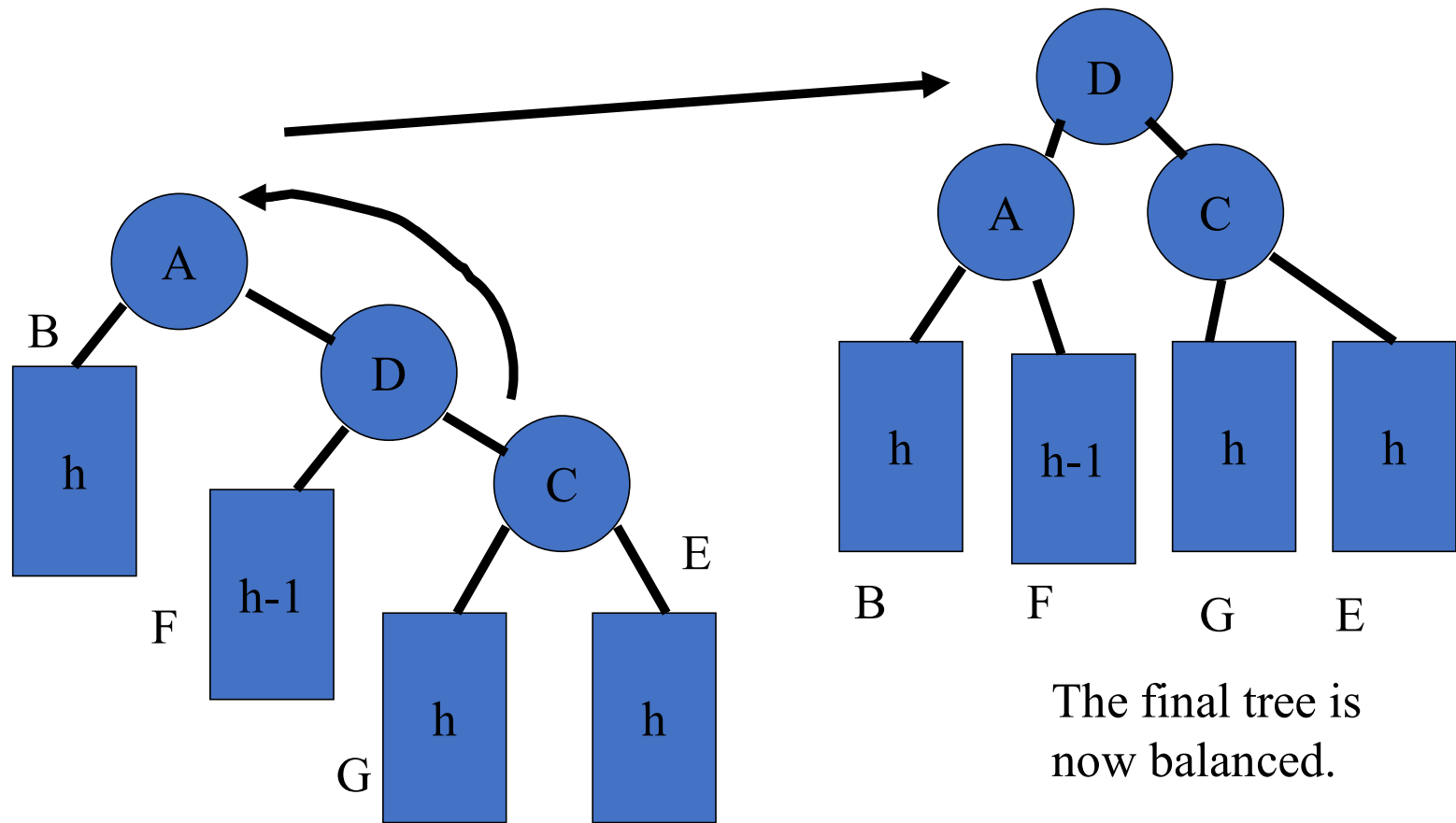
For Example, we want to insert  
new node into sub-tree G (or F)







Therefore, we perform the 2<sup>nd</sup> rotation, which is in the opposite direction to the first one, and which involves nodes A, D and C



Node D rotates up to replace A. Node A rotates down to become the left child of D, and Sub-tree F swaps over to become the new right child of node A.

Thank you