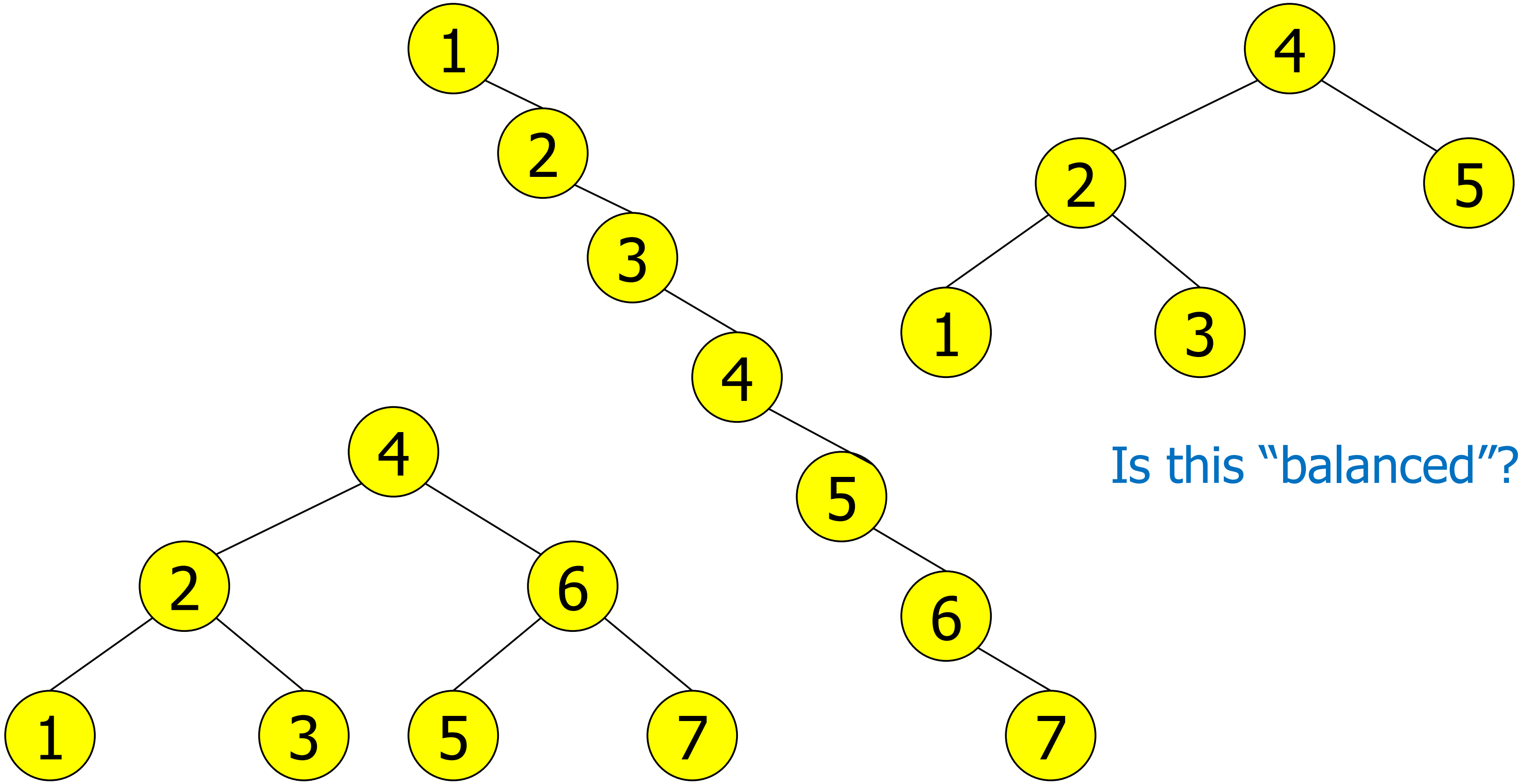
Data Structures
Instructor: Hafiz Tayyeb javed

Week-08-Lecture-01

18. AVL Trees

Balanced and Unbalanced BST

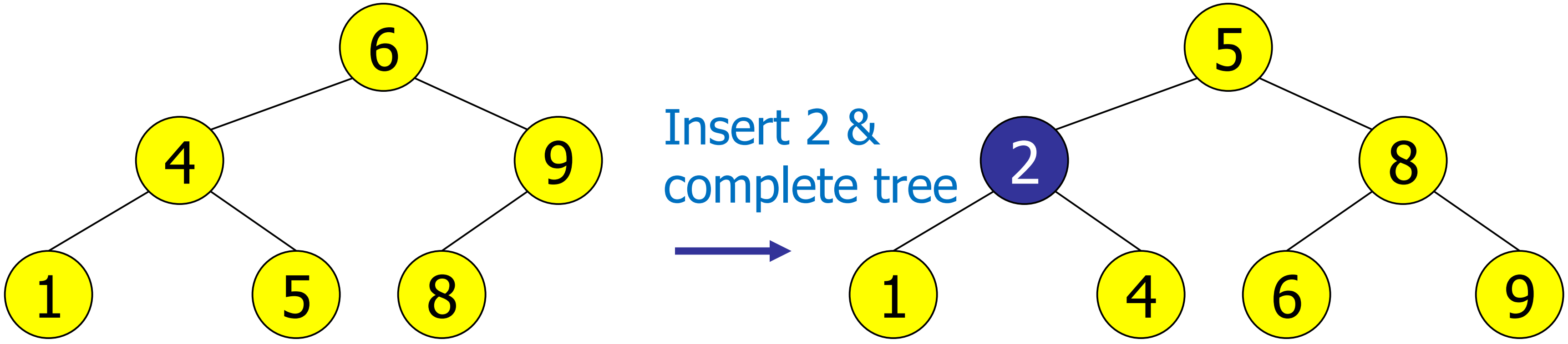


Is this “balanced”?

Balanced Tree

- Want a (almost) complete tree after every operation
  - Tree is full except possibly in the lower right

- Maintenance of such as tree is expensive
  - For example, insert 2 in the tree on the left and then rebuild as a complete tree



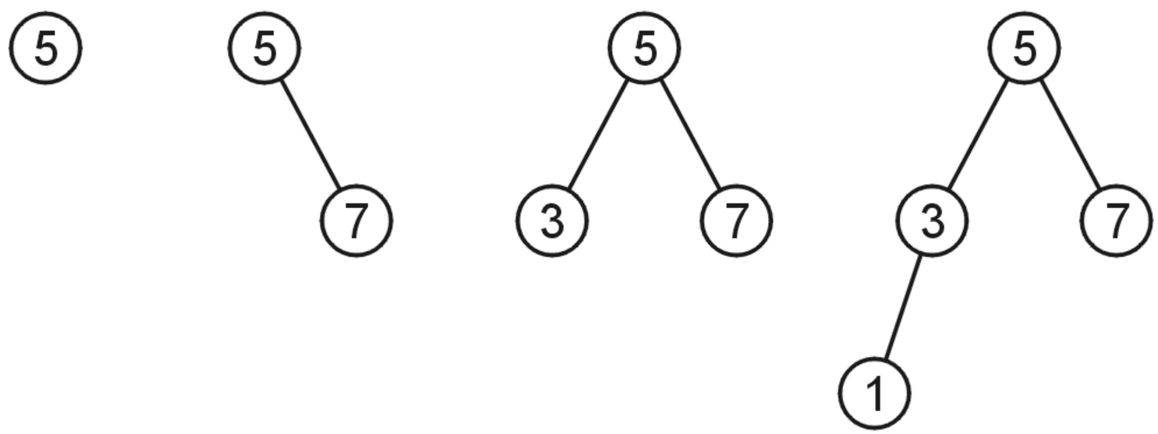
AVL Trees – Good but not Perfect Balance

- Named after Adelson-Velskii and Landis
- Balance is defined by comparing the height of the two sub-trees
- Recall:
  - An empty tree has height –1
  - A tree with a single node has height 0

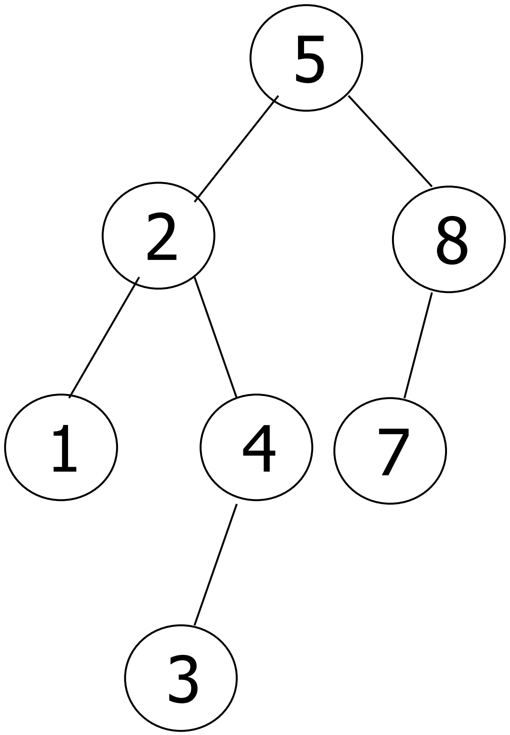
AVL Trees

- A binary search tree is said to be AVL balanced if:
  - The difference in the heights between the left and right sub-trees is at most 1, and
  - Both sub-trees are themselves AVL trees

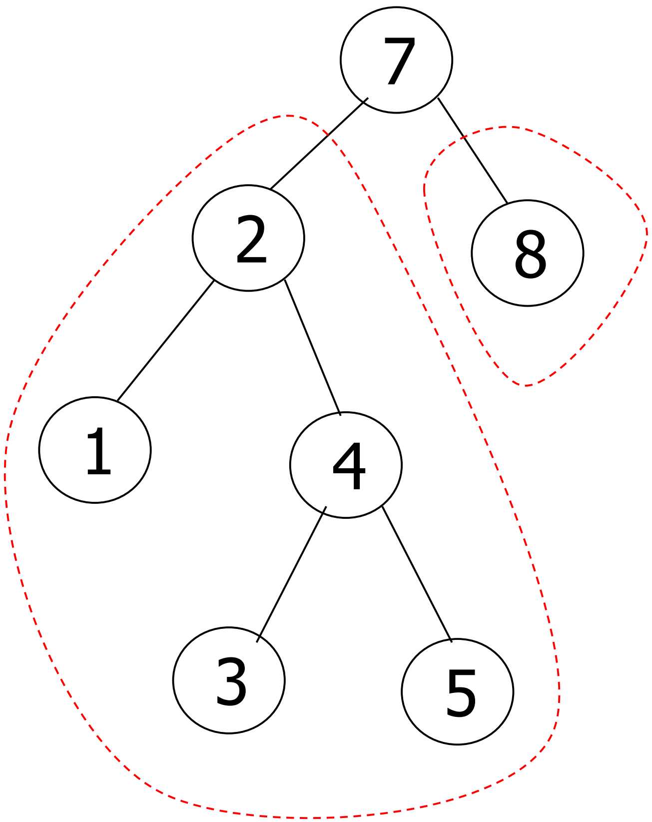
- AVL trees with 1, 2, 3 and 4 nodes



AVL Trees – Example



An AVL Tree

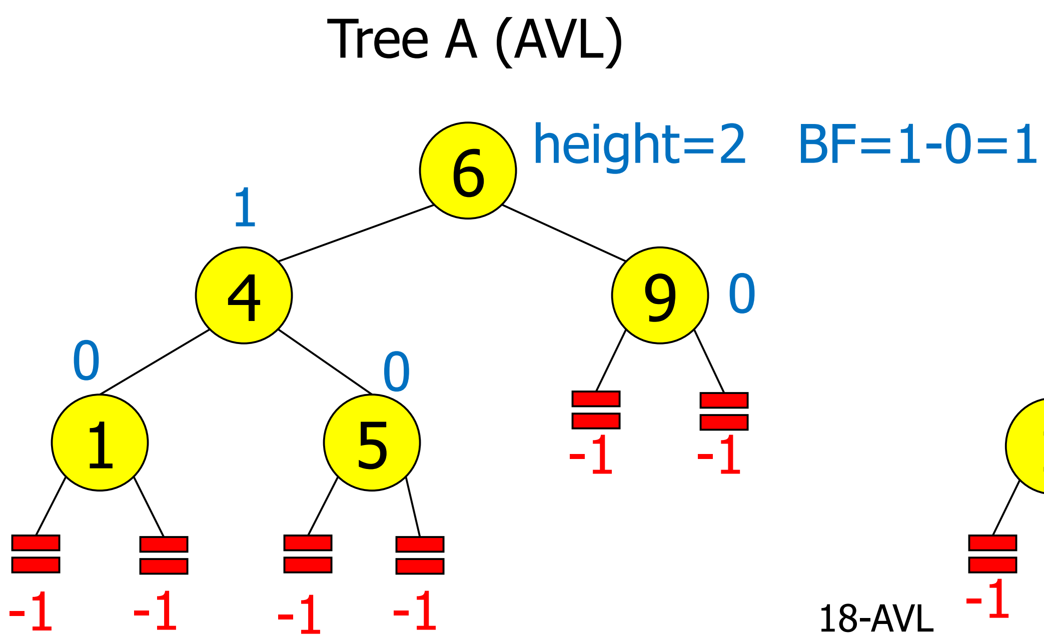


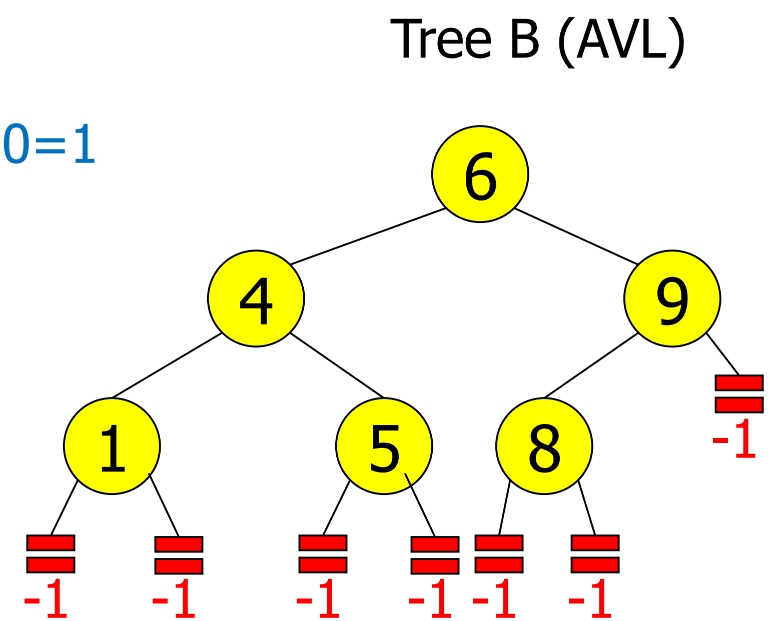
Not an AVL Tree

AVL Trees – Balance Factor

- An AVL tree has balance factor calculated at every node
  - Height of the left subtree minus the height of the right subtree
  - For an AVL tree, the balances of the nodes are always -1, 0 or 1.

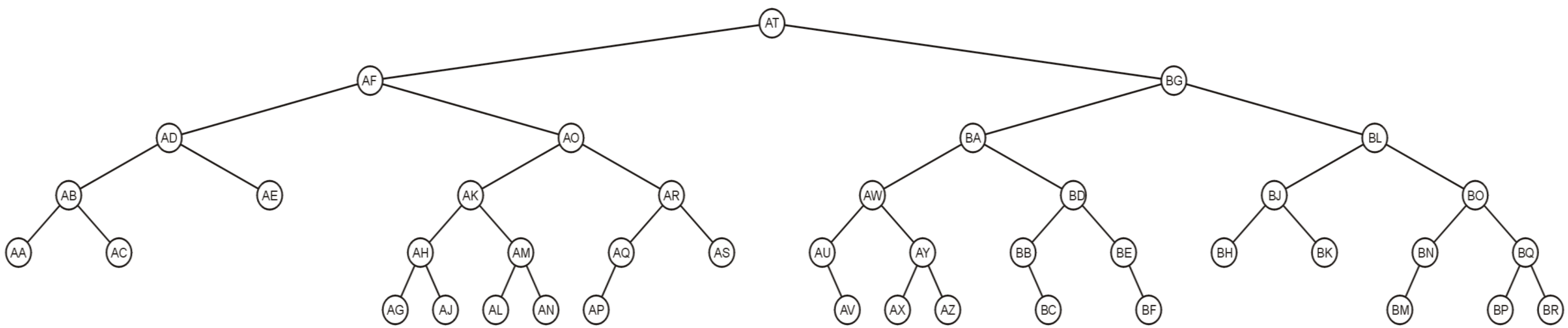
Height of node	=	h
Balance Factor (BF)	=	$h_{left} - h_{right}$
Empty height	=	-1





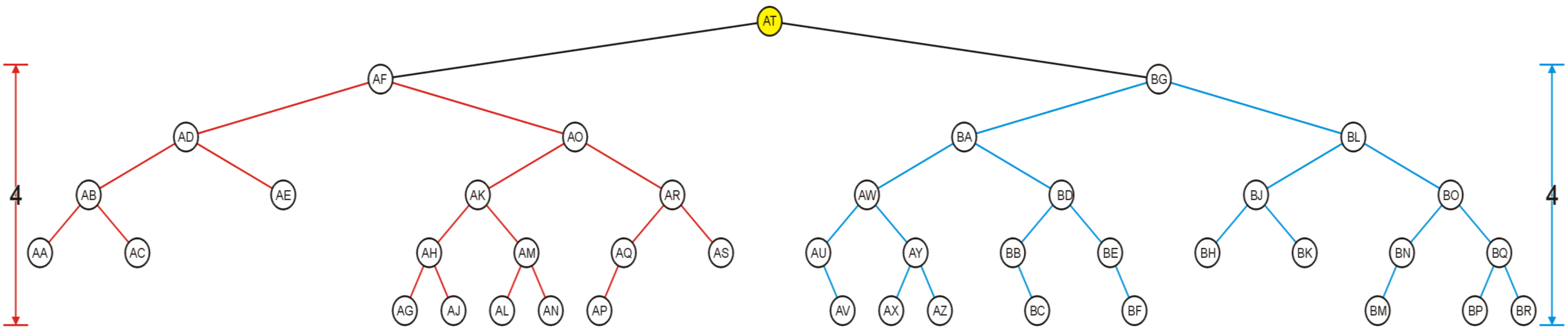
AVL Trees – Example

- Here is a larger AVL tree (42 nodes)



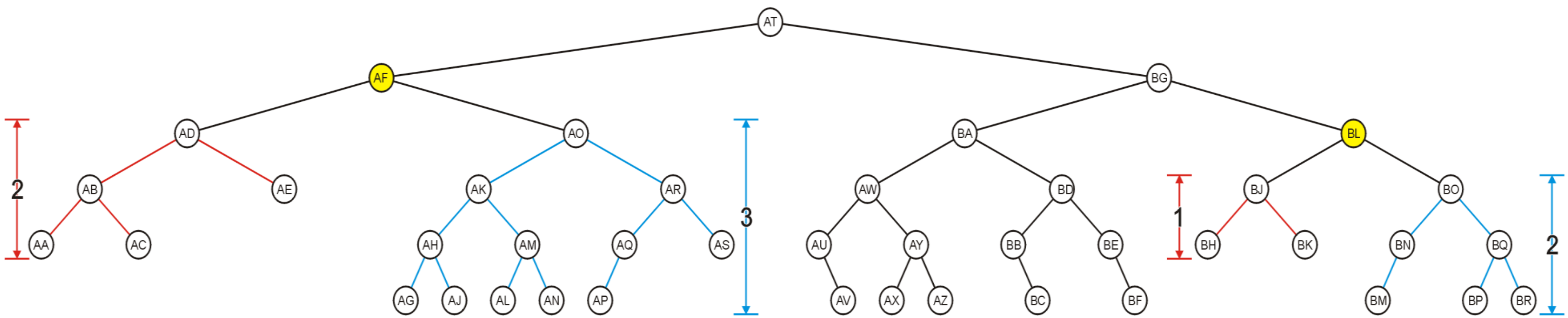
AVL Trees – Example

- The root node is AVL-balanced
  - Both sub-trees are of height 4 (i.e., at root BF = 0)



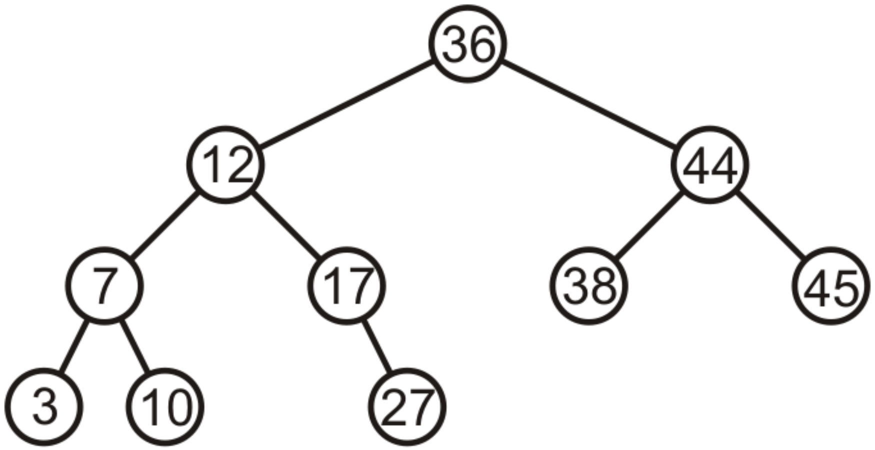
AVL Trees – Example

- All other nodes (e.g., AF and BL) are AVL balanced
  - The sub-trees differ in height by at most one



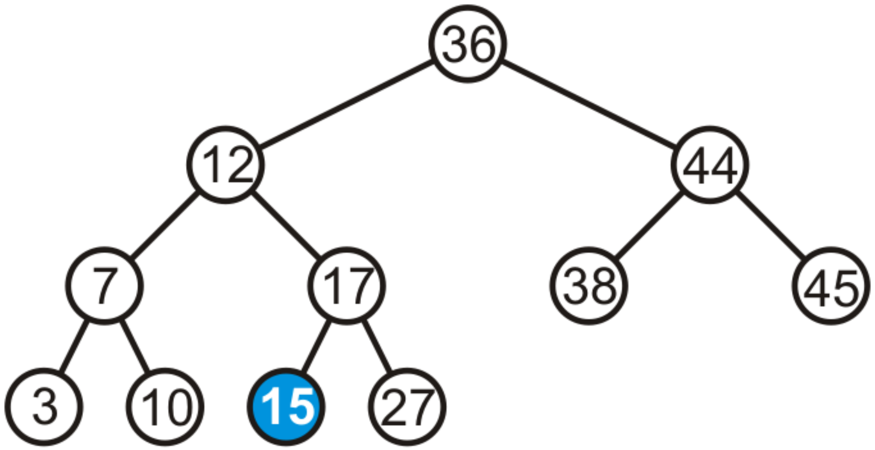
AVL Trees – Example

- Consider this AVL tree



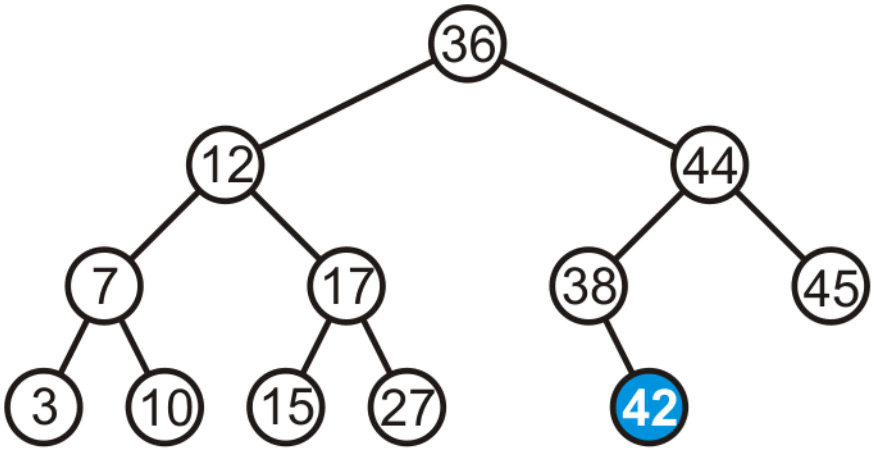
AVL Trees – Example

- Consider inserting 15 into this tree
  - In this case, the heights of none of the trees change
  - Tree remains balanced



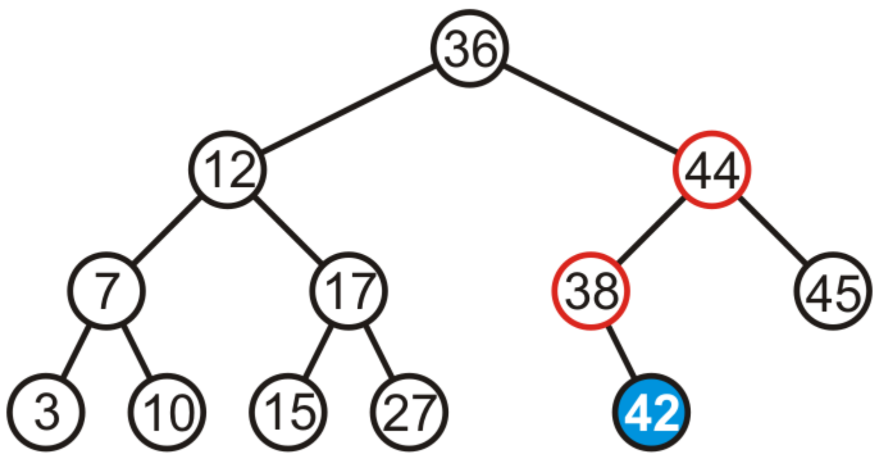
AVL Trees – Example

- Consider inserting 42 into this tree

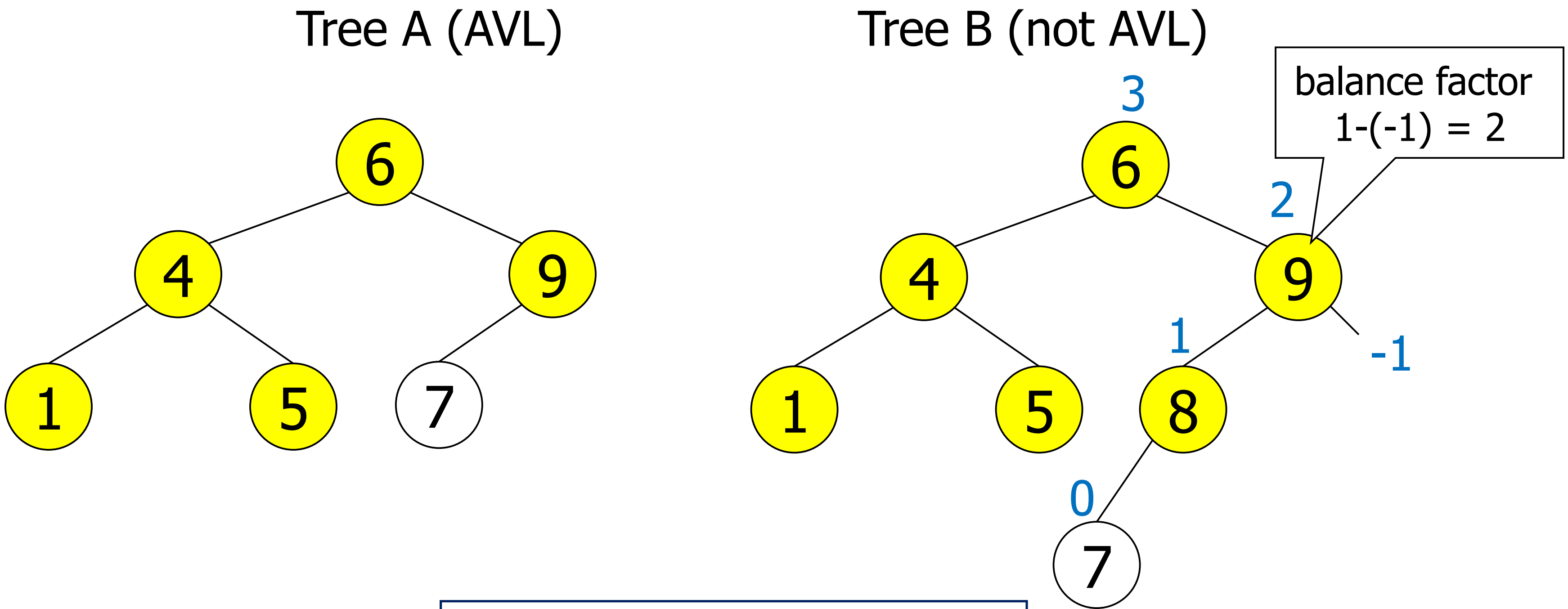


AVL Trees – Example

- Consider inserting 42 into this tree
  - Height of two sub-trees rooted at 44 and 38 have increased by one
  - The tree is still balanced



AVL Trees – Example



Height of node = h
Balance factor = $h_{left}-h_{right}$
Empty height = -1

AVL Trees

To maintain the height balanced property of the AVL tree after insertion or deletion, it is necessary to perform a *transformation* on the tree so that

1) the in-order traversal of the transformed tree is the same as for the original tree (i.e., the new tree remains a binary search tree).

2) the tree after transformation is height balanced.

Transformation (Rotation) of AVL Trees

- Insert operations may cause balance factor to become 2 or –2 for some node

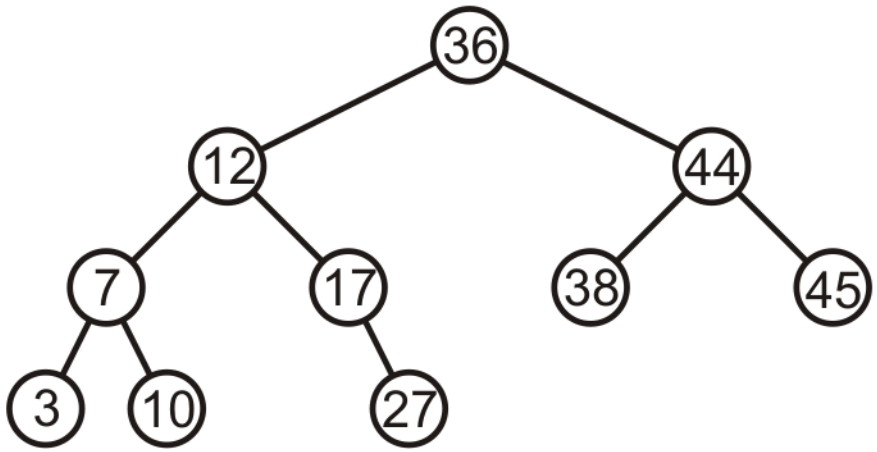
- Only nodes on the path from insertion point to root node have possibly change in height

- Follow the path up to the root, find the first node (i.e., deepest) whose new balance violates the AVL condition
  - Call this node a

- If a new balance factor (the difference h_{left}-h_{right}) is 2 or –2
  - Adjust tree by rotation around the node a

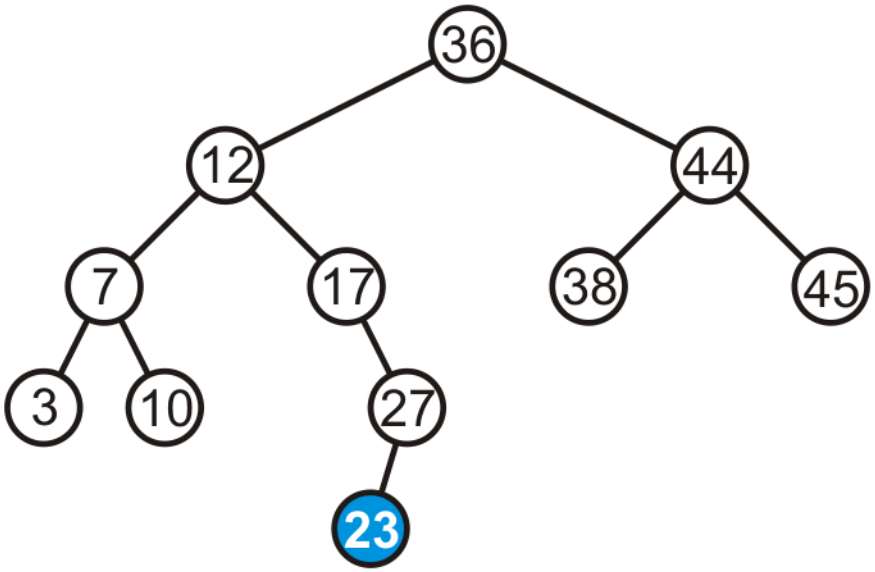
Balancing AVL Trees – Example

- If a tree is AVL balanced, for an insertion to cause an imbalance:
  - The heights of the sub-trees must differ by 1
  - The insertion must increase the height of the deeper sub-tree by 1



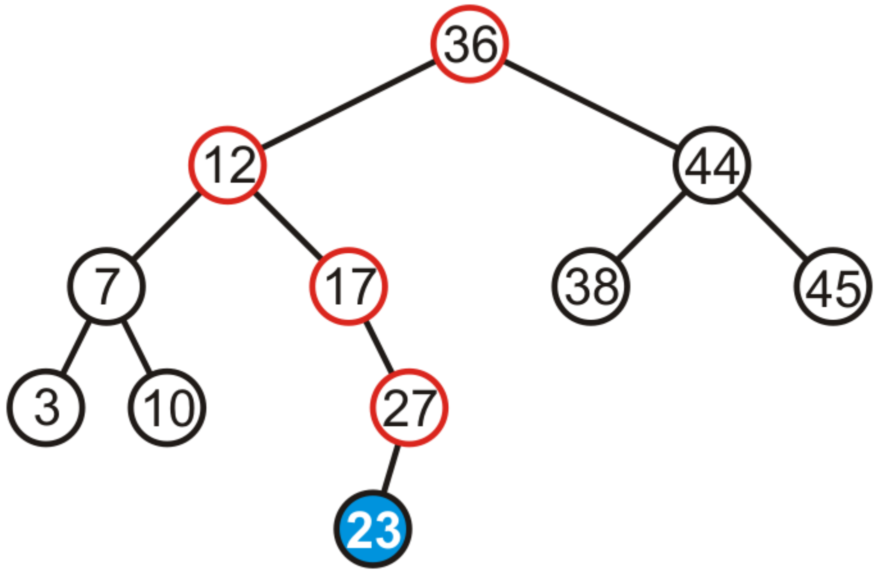
Balancing AVL Trees – Example

- Suppose we insert 23 into our initial tree



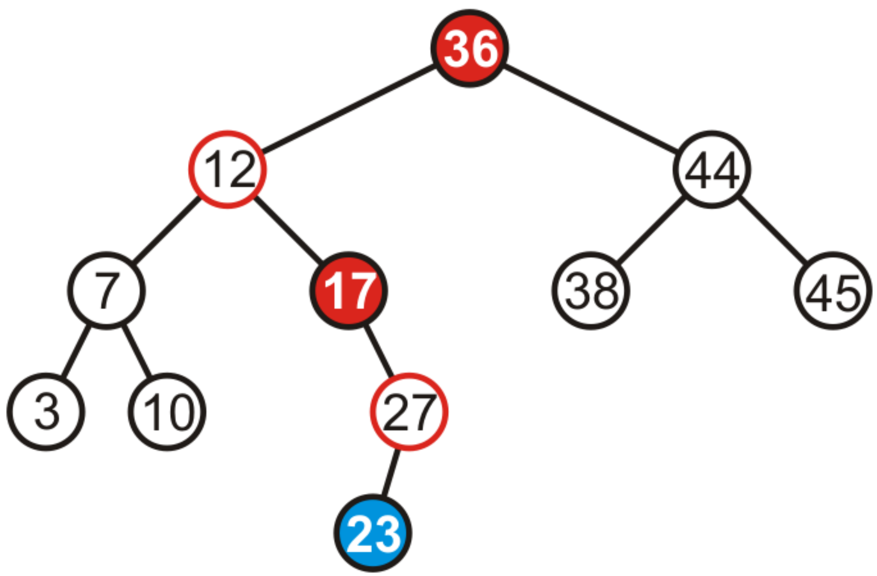
Balancing AVL Trees – Example

- The heights of each of the sub-trees from the insertion point to the root are increased by one



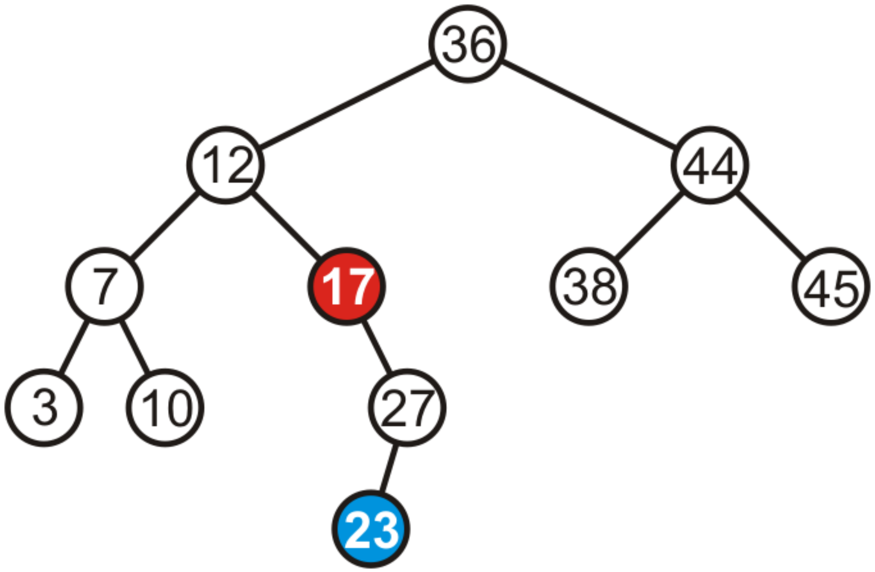
Balancing AVL Trees – Example

- Only two of the nodes are unbalanced, i.e., 17 and 36
  - Balance factor of 17 is -2
  - Balance factor of 36 is 2

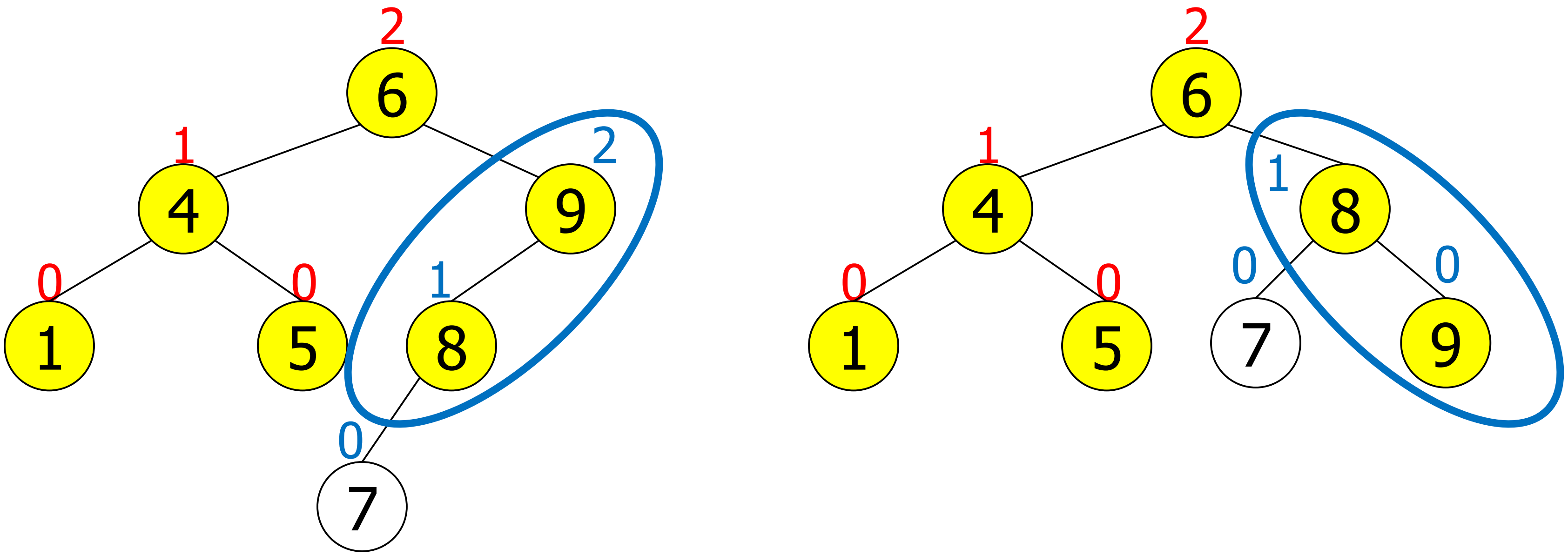


Balancing AVL Trees – Example

- We only have to fix the imbalance at the lowest node



Single Rotation in an AVL Tree

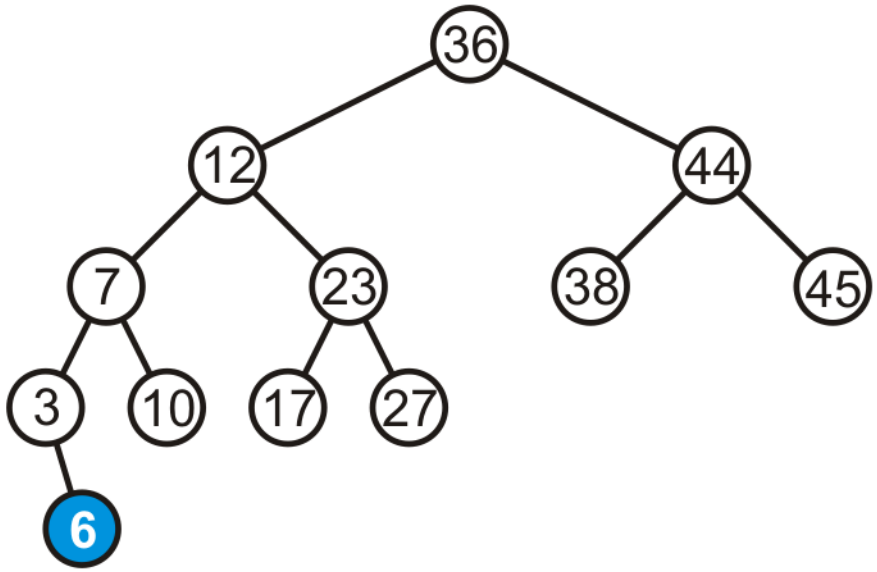


Right Rotation (RR) in an AVL Tree

- Node b becomes the new root
- Node b takes ownership of node a, as it's right child
- Node a takes ownership of node b's right child (or NULL if no child)
  - As left child of node a

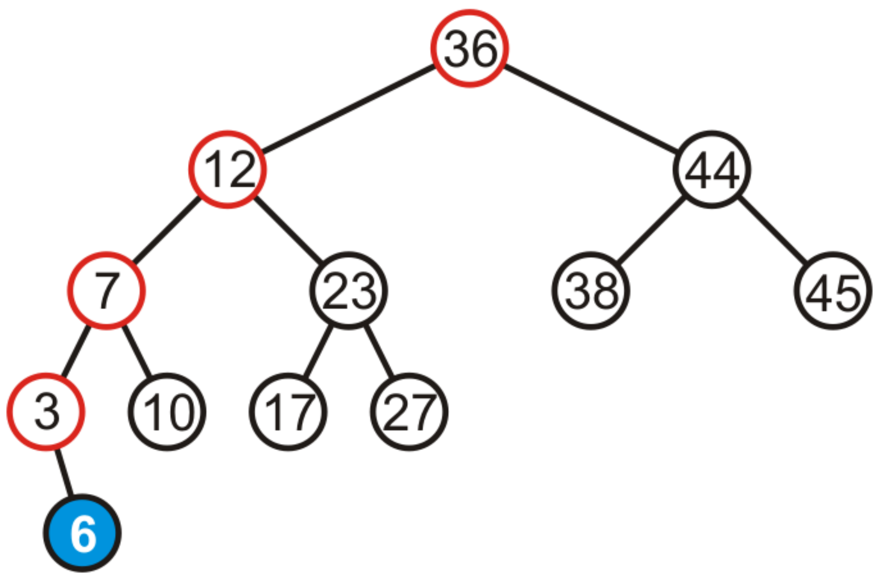
Right Rotation (RR) – Example

- Consider adding 6



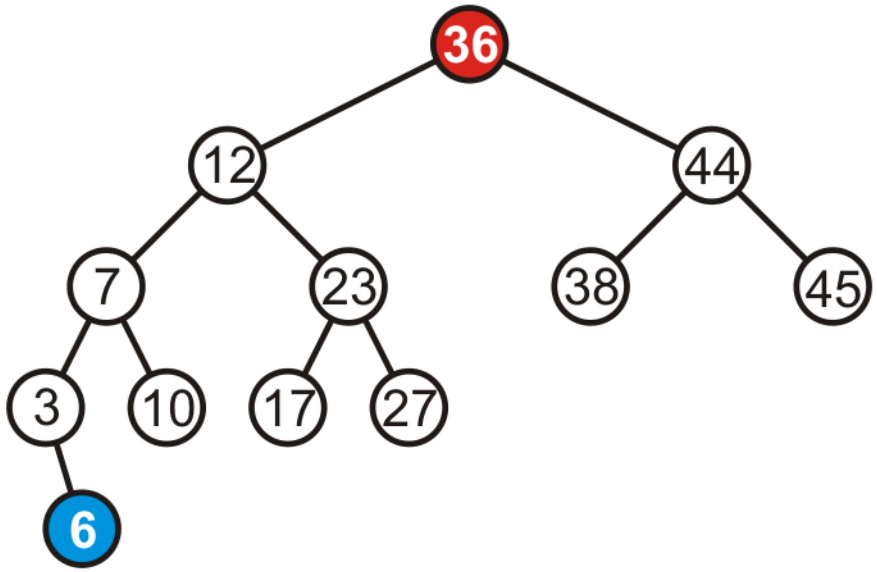
Right Rotation (RR) – Example

- Height of each of the trees in the path back to the root are increased by one



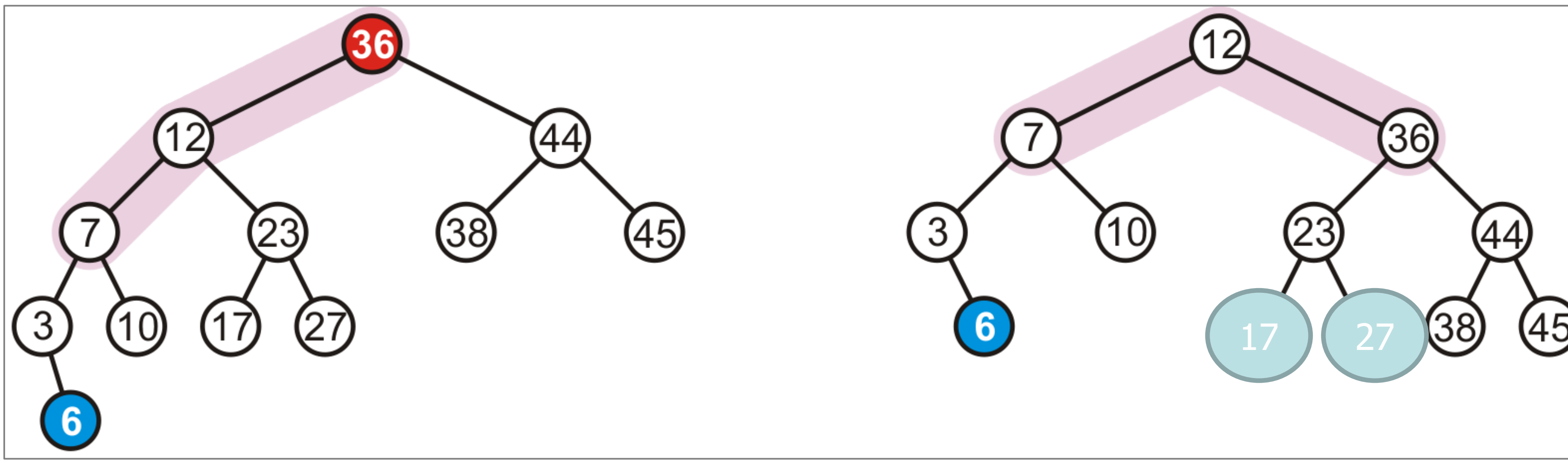
Right Rotation (RR) – Example

- Height of each of the trees in the path back to the root are increased by one
  - Only root node (i.e., 36) violates the balancing factor



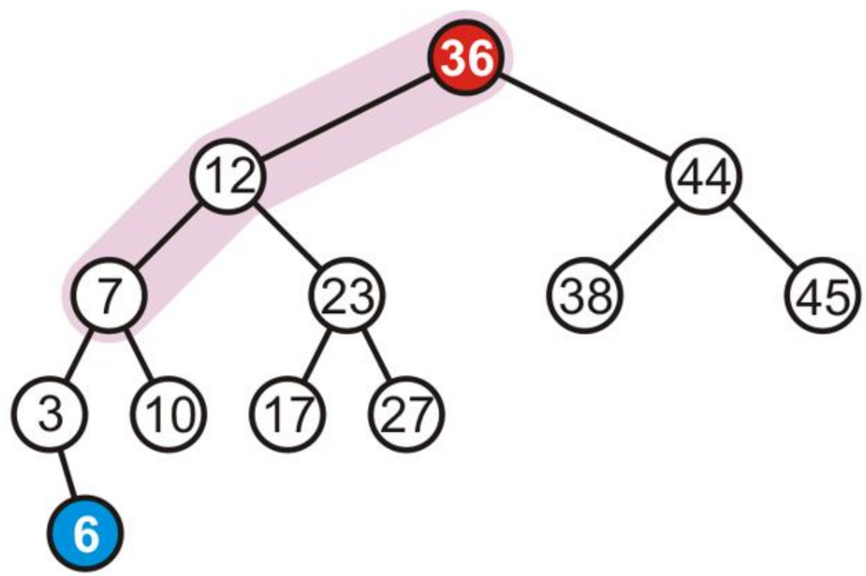
Right Rotation (RR) – Example

- To fix the imbalance, we perform right rotation of root (i.e., 36)



When to Perform Right Rotation (RR)

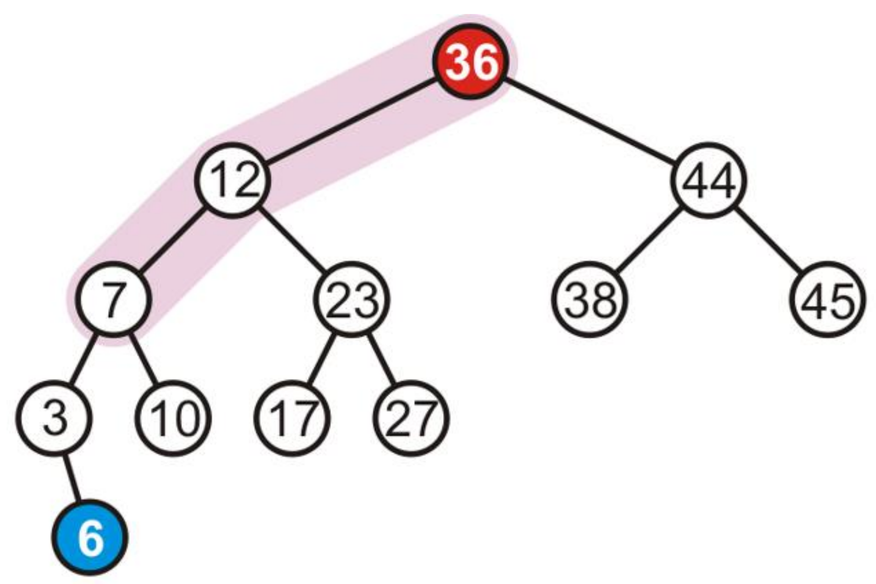
- Let the node that needs rebalancing be a
- Case RR
  - Insertion into left subtree of left child of node a
  - Left tree is heavy (i.e., $h_{left} > h_{right}$)



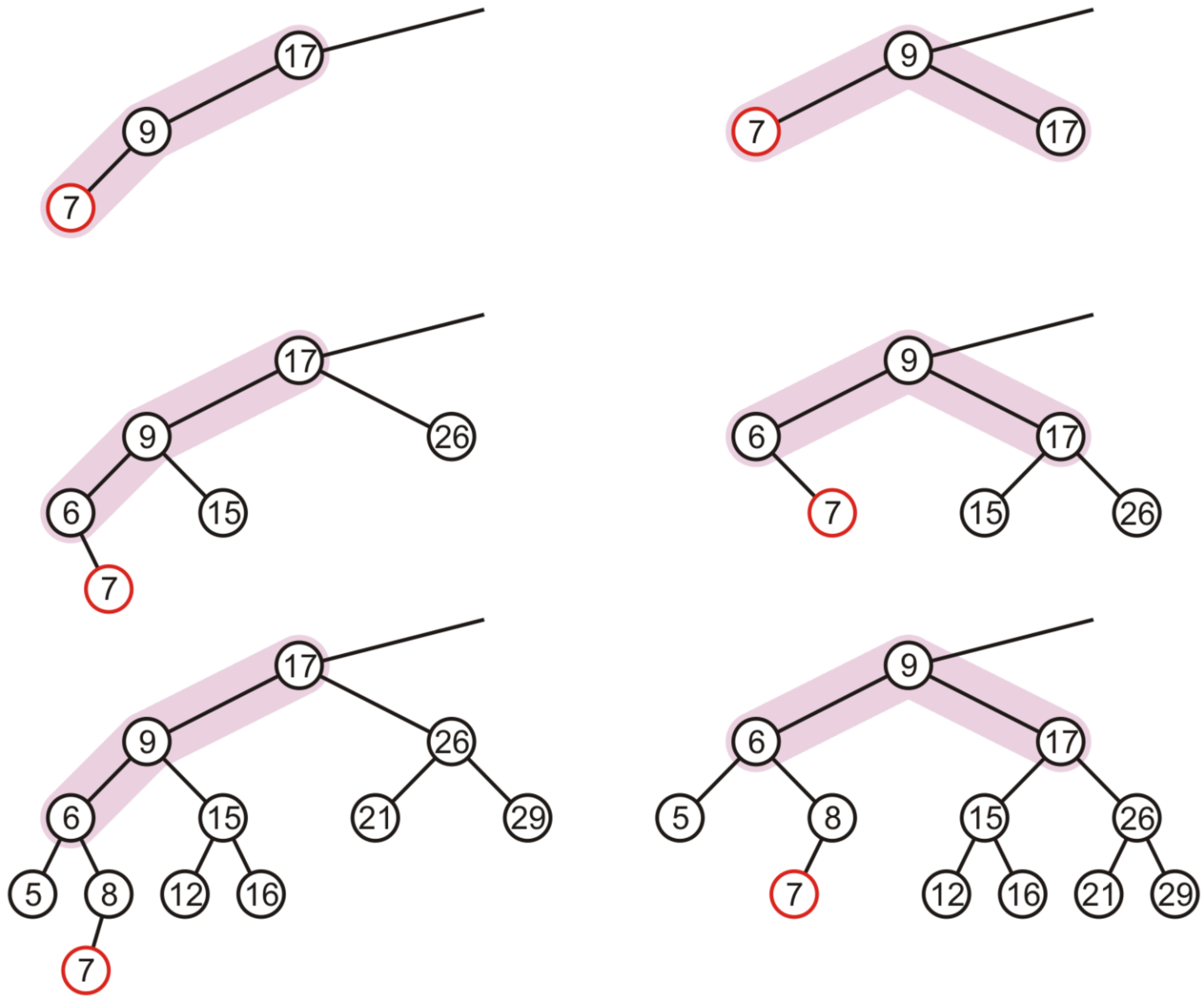
When to Perform Right Rotation (RR)

- Let the node that needs rebalancing be a

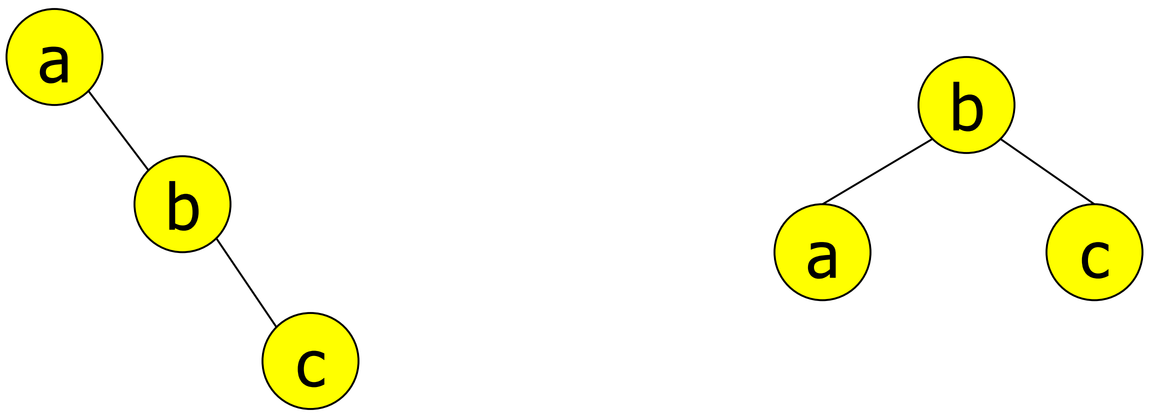
- Case RR
  - Insertion into left subtree of left child of node a (**RR**)
  - Left tree is heavy (i.e., $h_{left} > h_{right}$)



Right Rotation (RR) – Examples



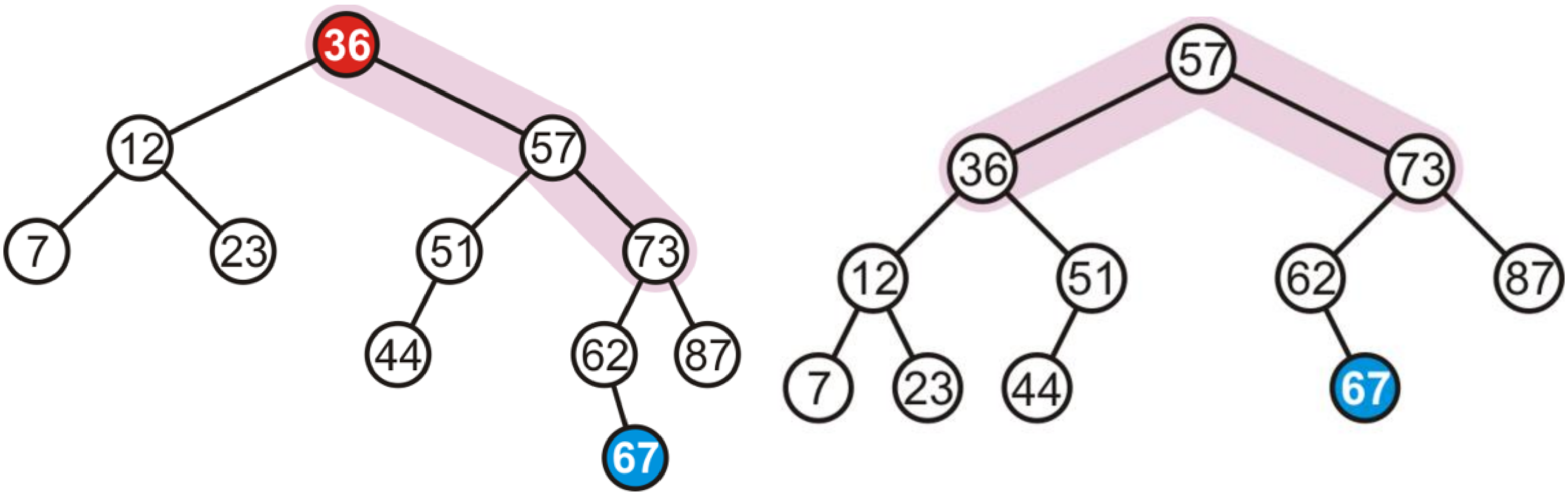
Left Rotation (LL) in an AVL Tree



- Node b becomes the new root
- Node b takes ownership of node a as its left child
- Node a takes ownership of node b's left child (or NULL if no child)
  - As right child of node a

Left Rotation (LL) – Example

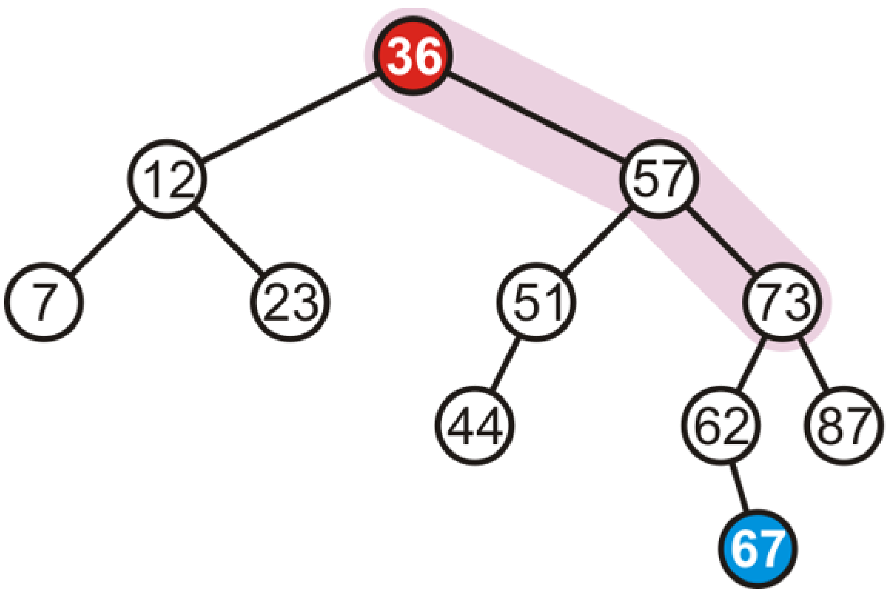
- Consider adding 67
  - To fix the imbalance, we perform left rotation of root (i.e., 36)



When to Perform Left Rotation (LL)

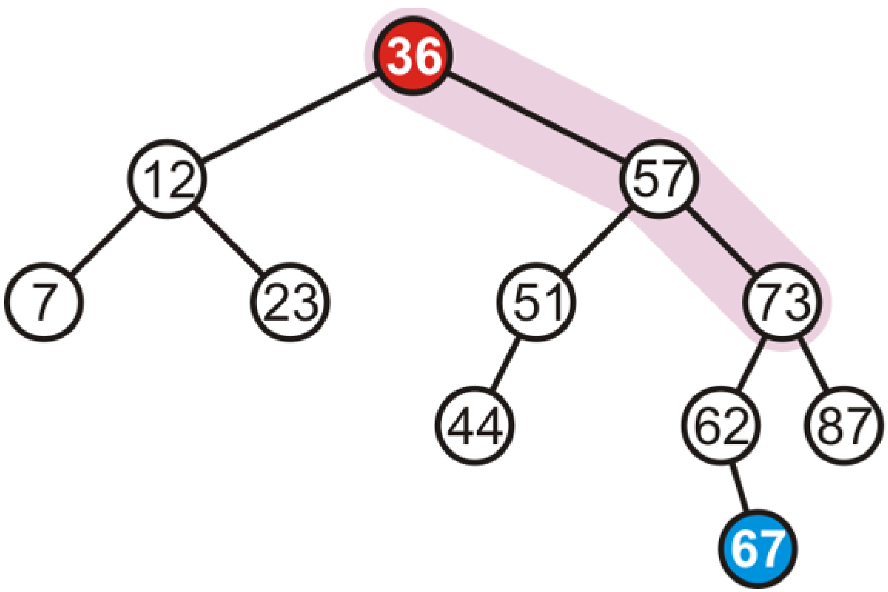
- Let the node that needs rebalancing be a

- Case LL
  - Insertion into right subtree of right child of node a
  - Right tree is heavy (i.e., $h_{left} < h_{right}$)

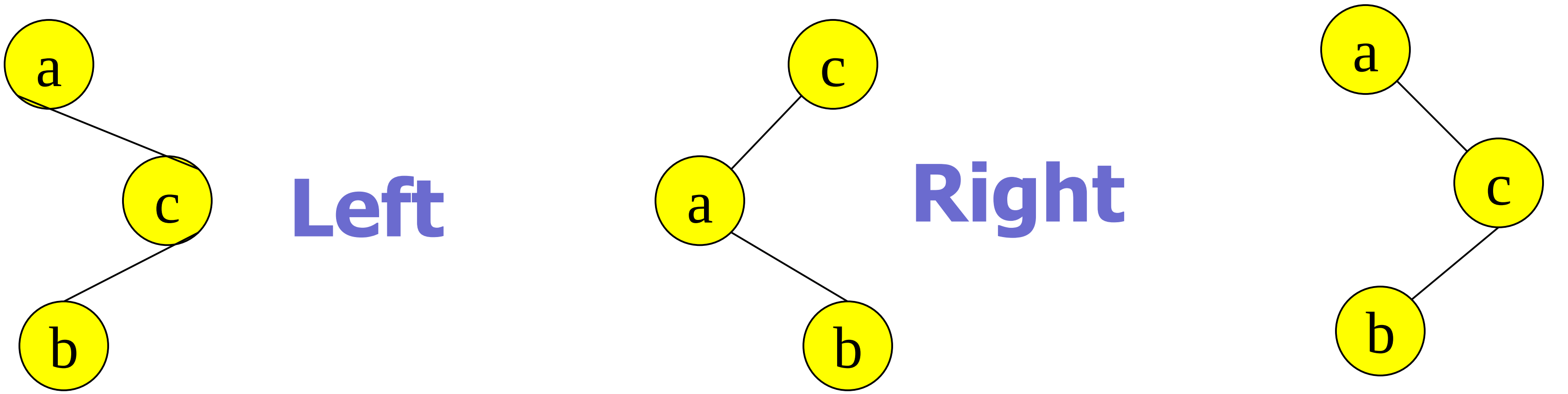


When to Perform Left Rotation (LL)

- Let the node that needs rebalancing be a
- Case LL
  - Insertion into right subtree of right child of node a (LL)
  - Right tree is heavy (i.e., $h_{left} < h_{right}$)



Single Rotation may be Insufficient

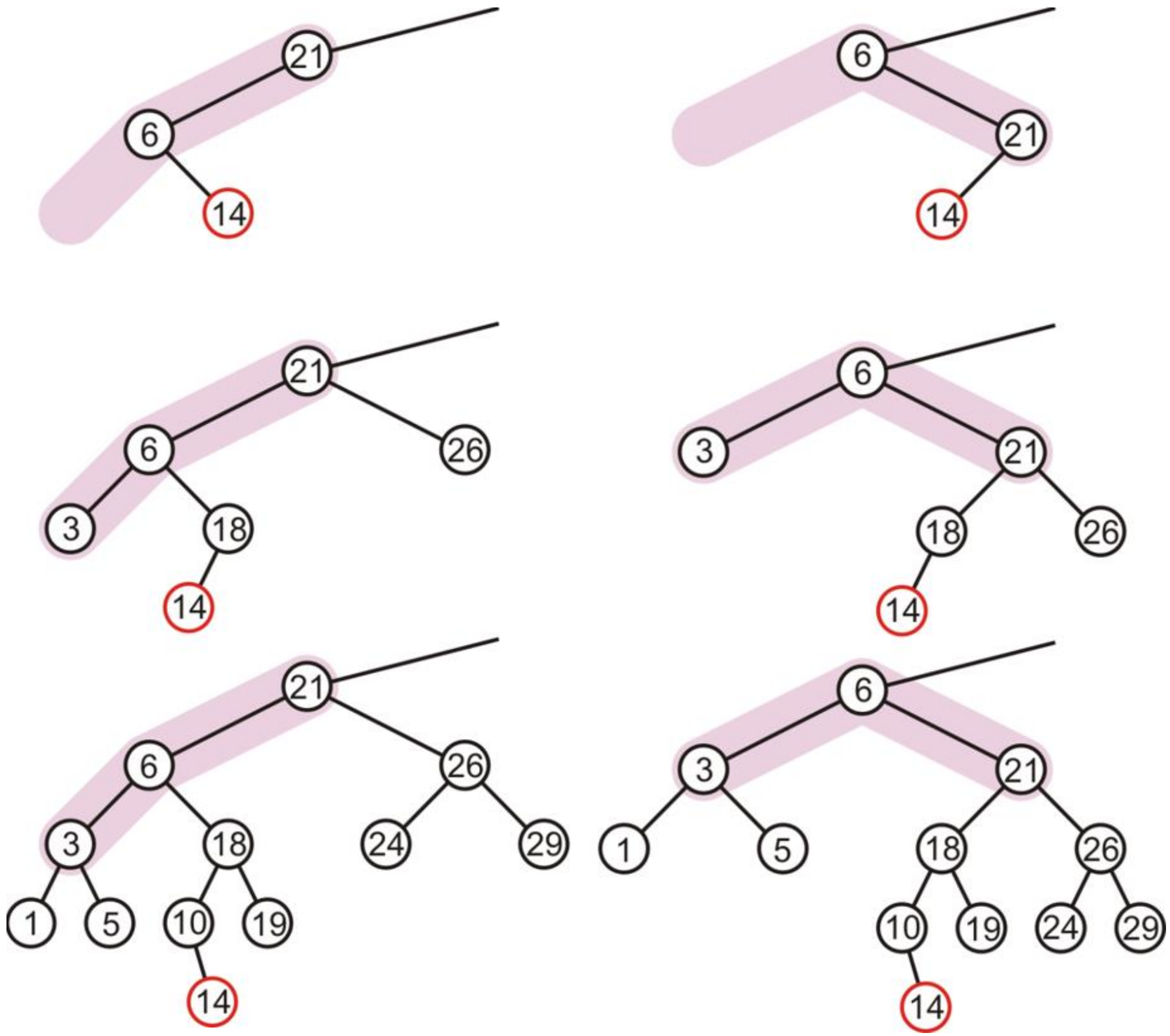


- *c becomes the new root.*
- *a takes ownership of c's left child as its right child, in this case, b.*
- *c takes ownership of a as its left child.*

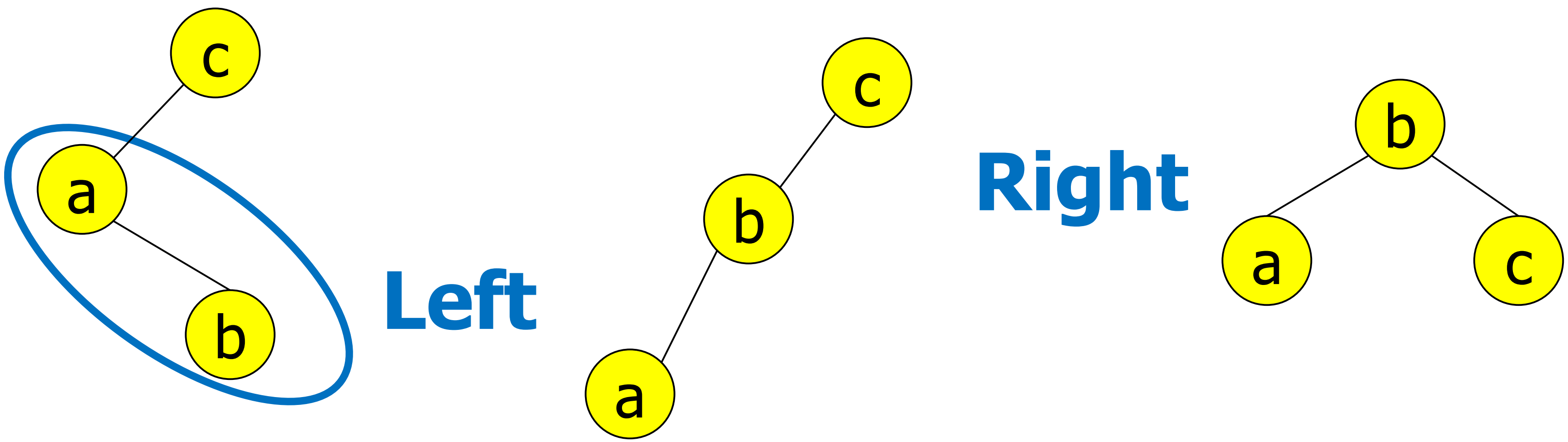
- *a becomes the new root.*
- *c takes ownership of a's right child as its left child, b.*
- *a takes ownership of c as its right child.*

Single Rotation May Be Insufficient

- The imbalance is just shifted to the other side



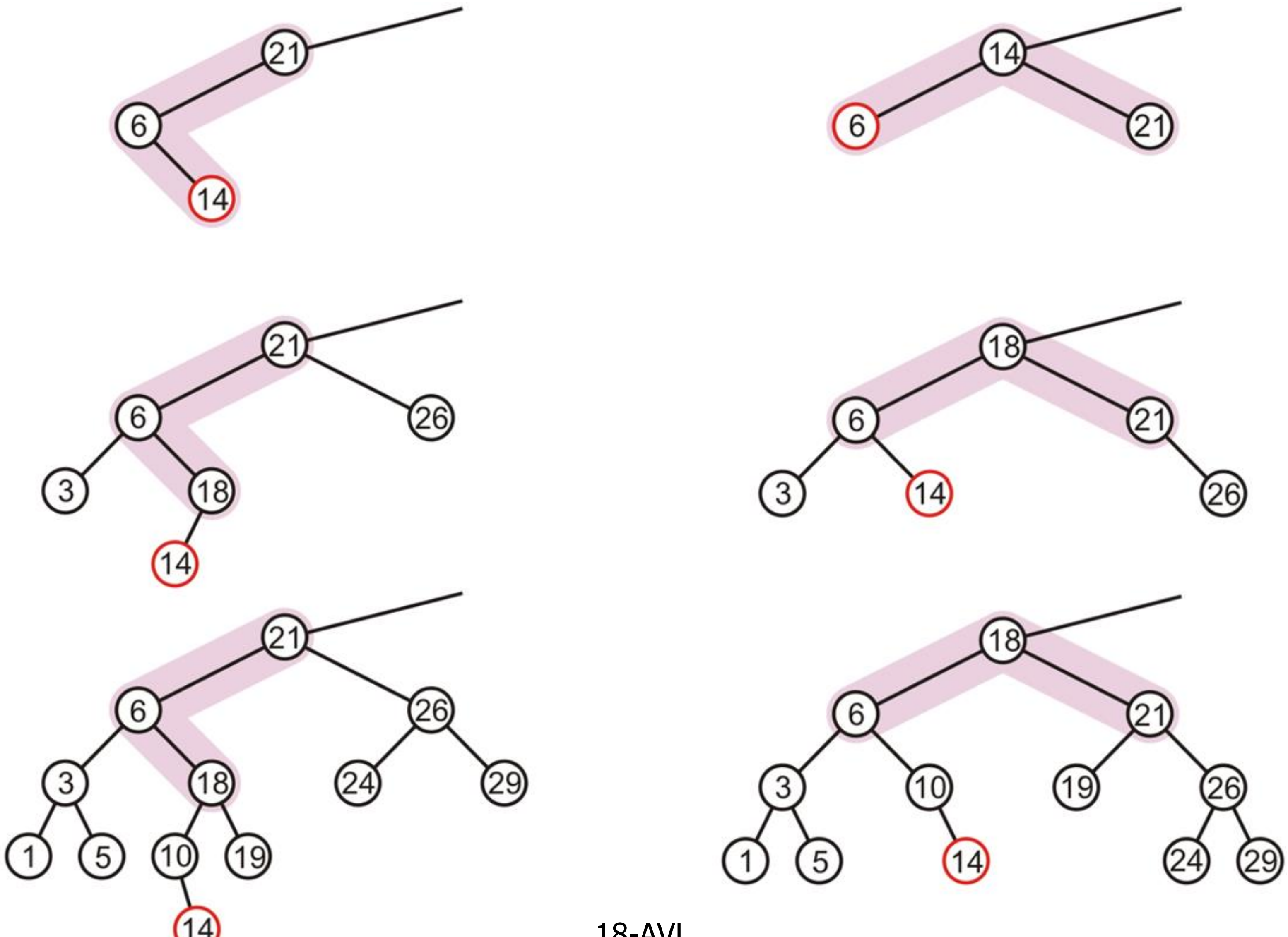
Right-Left Rotation (RL) or "Double Right"



- Perform a left rotation on the left subtree

- Node b becomes the new root
- Node b takes ownership of node c as its right child
- Node c takes ownership of node b's right child
  - As its left child

Right-Left Rotation (RL) or "Double Right" – Example



Continued…

Any Question So Far?