AVL Trees

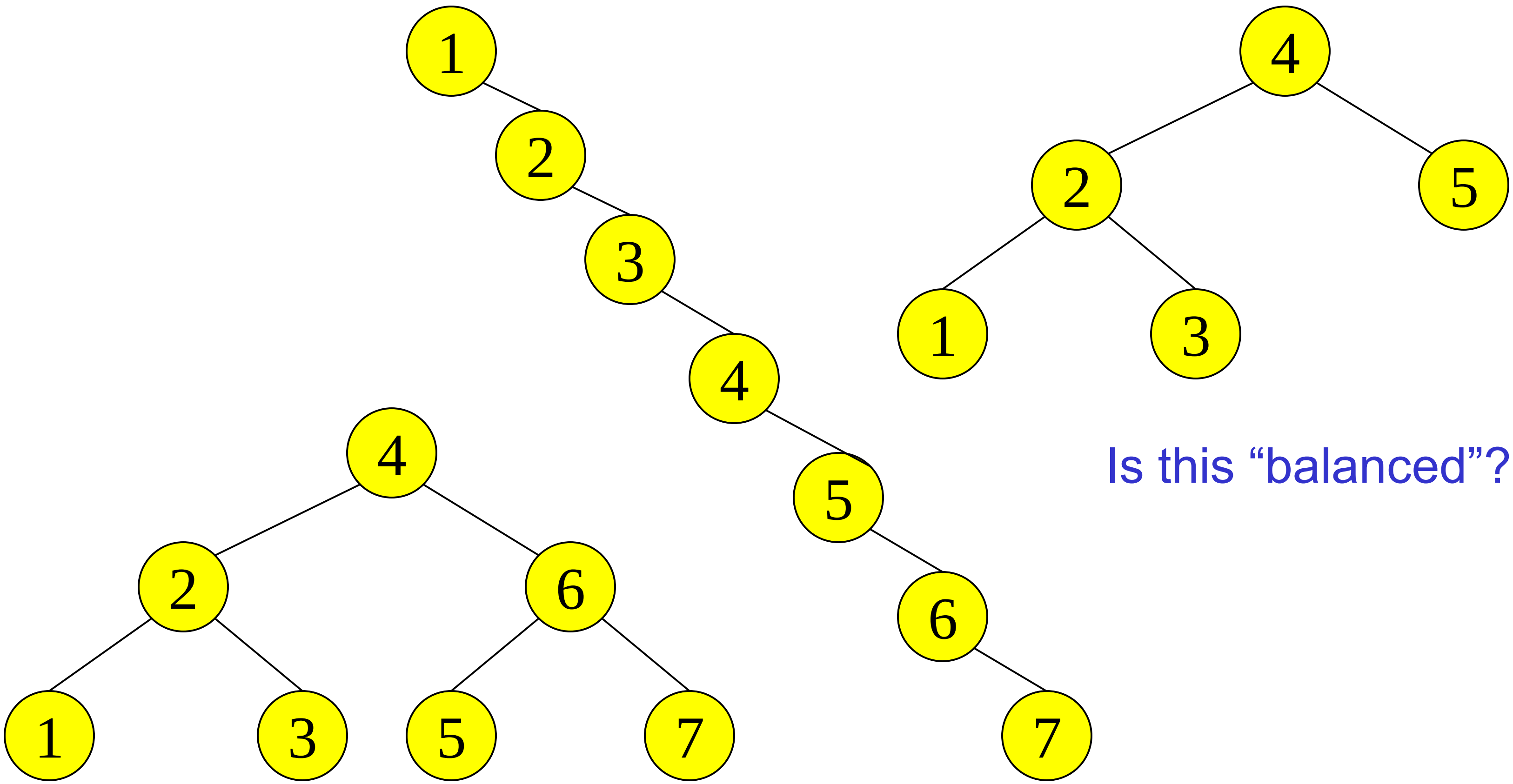
Binary Search Tree - Best Time

- All BST operations are O(h), where h is tree height
- minimum h is $\log_2 N$ for a binary tree with N nodes
  - What is the best case tree?
  - What is the worst case tree?
- So, best case running time of BST operations is O(log N)

Binary Search Tree - Worst Time

- Worst case running time is O(N)
  - › What happens when you Insert elements in ascending order?
    - Insert: 2, 4, 6, 8, 10, 12 into an empty BST
  - › Problem: Lack of "balance":
    - compare height of left and right subtree
  - › Unbalanced degenerate tree

Balanced and unbalanced BST



Balancing Binary Search Trees

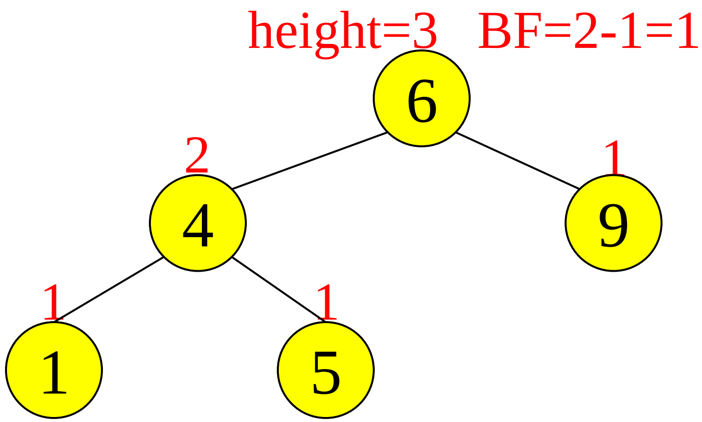
- Many algorithms exist for keeping binary search trees balanced
  - Adelson-Velskii and Landis (AVL) trees (height-balanced trees)
  - Splay trees and other self-adjusting trees
  - B-trees and other multiway search trees

AVL Trees

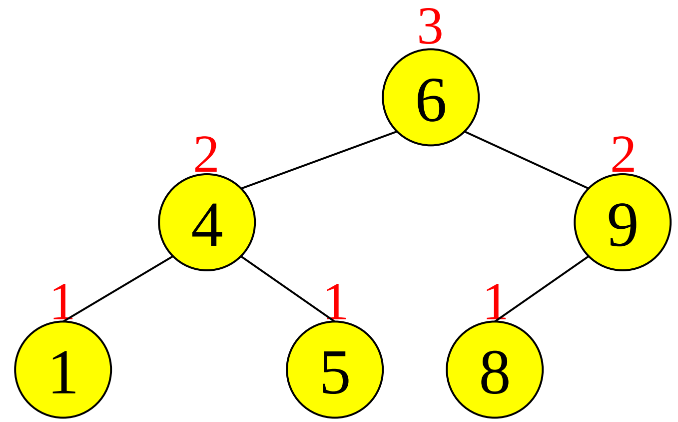
- AVL trees are height-balanced binary search trees
- Balance factor of a node
  - height(left subtree) - height(right subtree)
- An AVL tree has balance factor calculated at every node
  - For every node, heights of left and right subtree can differ by no more than 1
  - Store current heights in each node

Node Heights

Tree A (AVL)

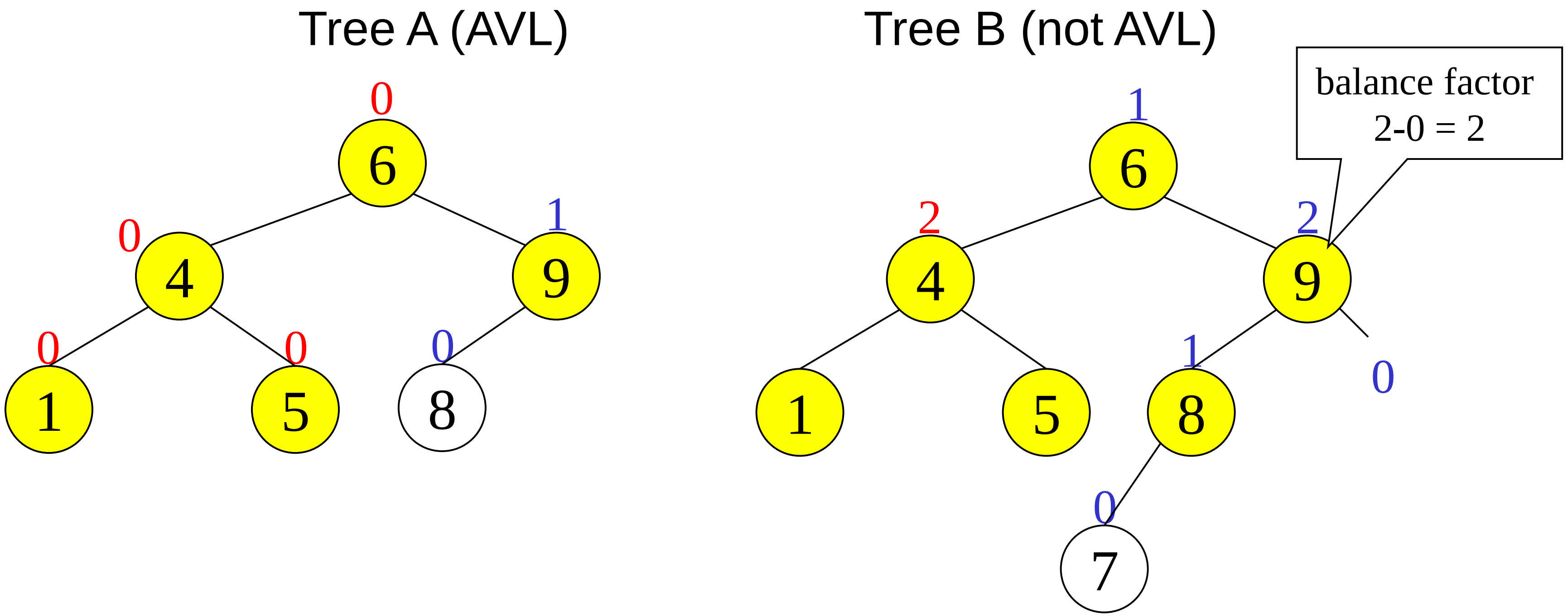


Tree B (AVL)



height of node = h
balance factor = h_{left}-h_{right}
empty height = 0

Node Heights after Insert 7



Insert and Rotation in AVL Trees

- Insert operation 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 changed in height
  - So after the Insert, go back up to the root node by node, updating heights
  - If a new balance factor (the difference h_{left}-h_{right}) is 2 or –2, adjust tree by *rotation* around the node

Insertions in AVL Trees

Let the node that needs rebalancing be A
(with a balance factor > 1 or < -1)

There are 4 cases:

Single rotation (LL / RR)

1. LL- Insertion into left subtree of left child of A
2. RR- nsertion into right subtree of right child of A.

Double Rotation(LR/RL)

3. LR -Insertion into right subtree of left child of A
4. RL -Insertion into left subtree of right child of A

The rebalancing is performed through four separate rotation algorithms

The rebalancing is performed through four separate rotation algorithms

If **balance factor > 1**, then the current node is unbalanced..

We are either in **Left-Left [LL] case** or **Left-Right [LR] case**.

To check whether it is Left-Left case or not, compare the newly inserted key with the key in left sub-tree of the root.

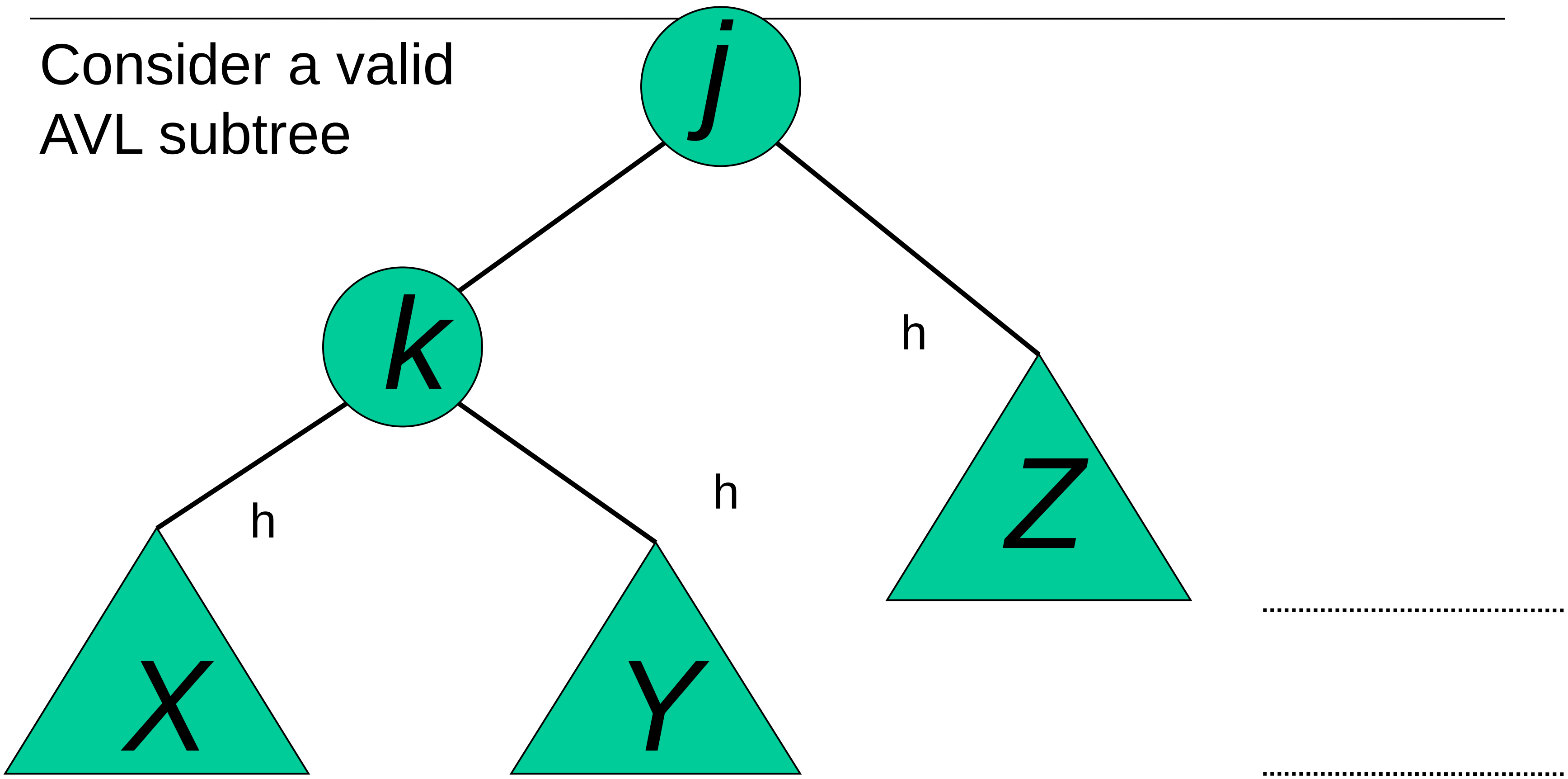
If **balance factor < -1**, then the current node is unbalanced ..

We are either in **Right-Right [RR] case** or **Right-Left [RL] case**.

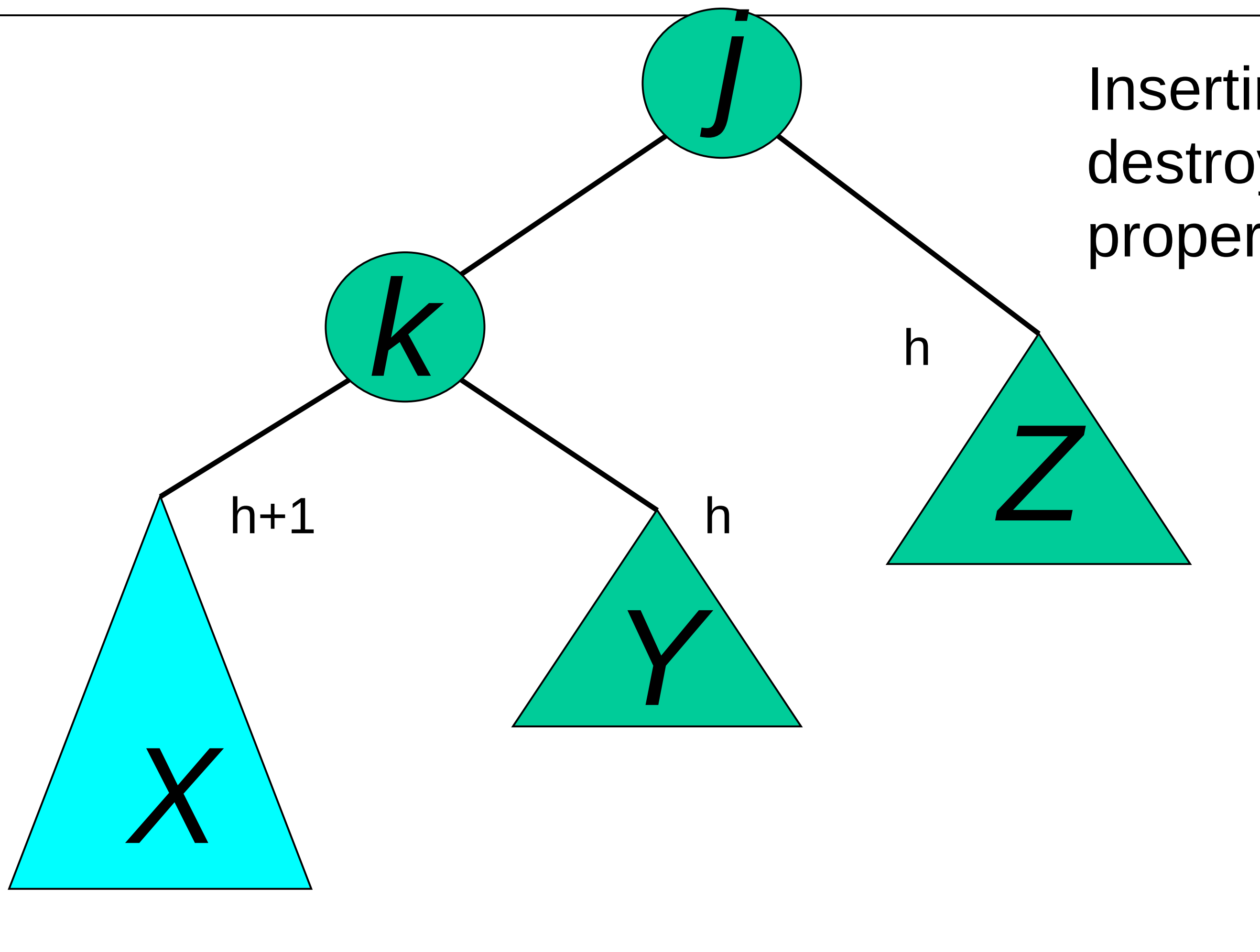
To check whether it is Right-Right case or not, compare the newly inserted key with the key in right sub-tree of the root.

AVL Insertion: LL Case

Consider a valid AVL subtree

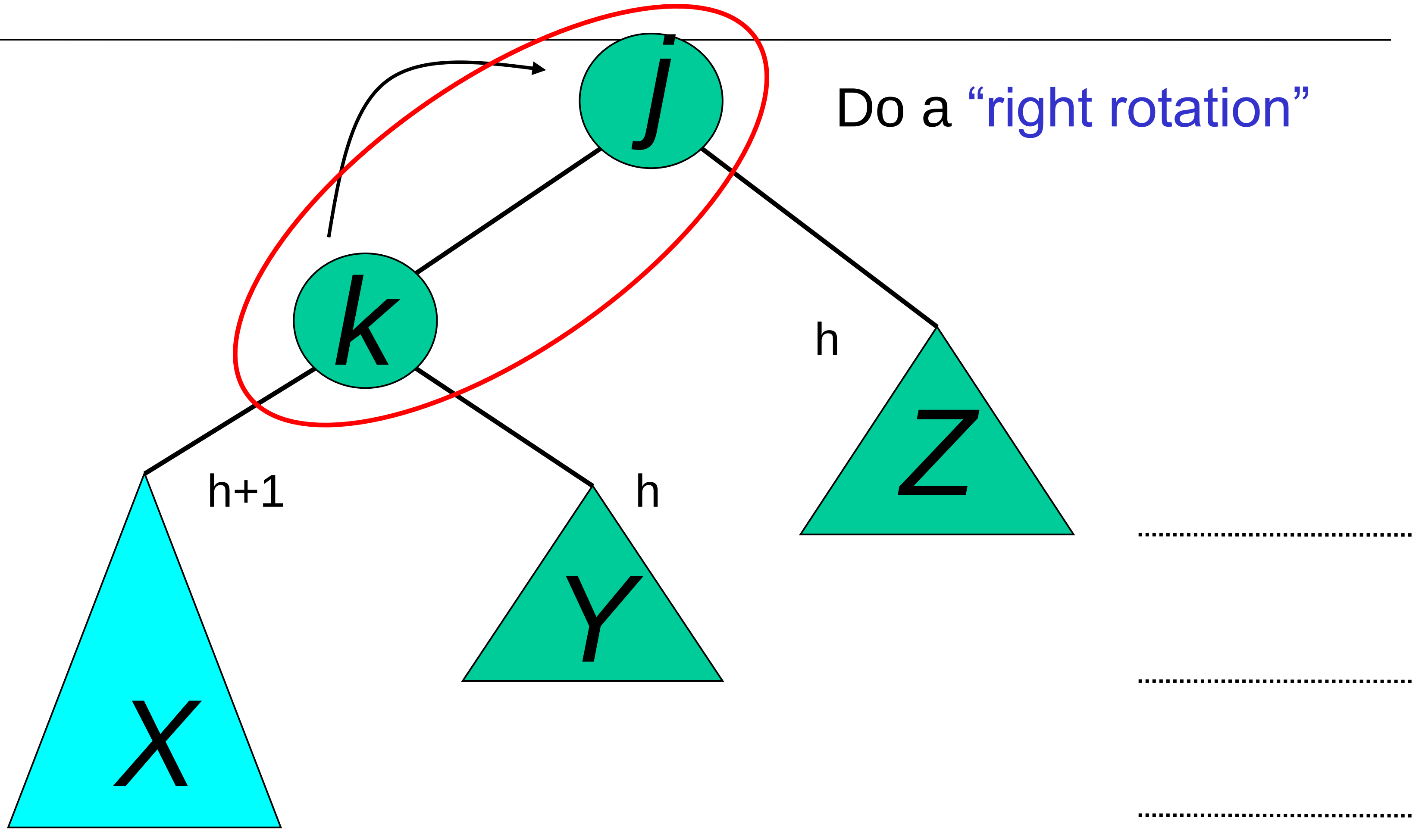


AVL Insertion: LL Case

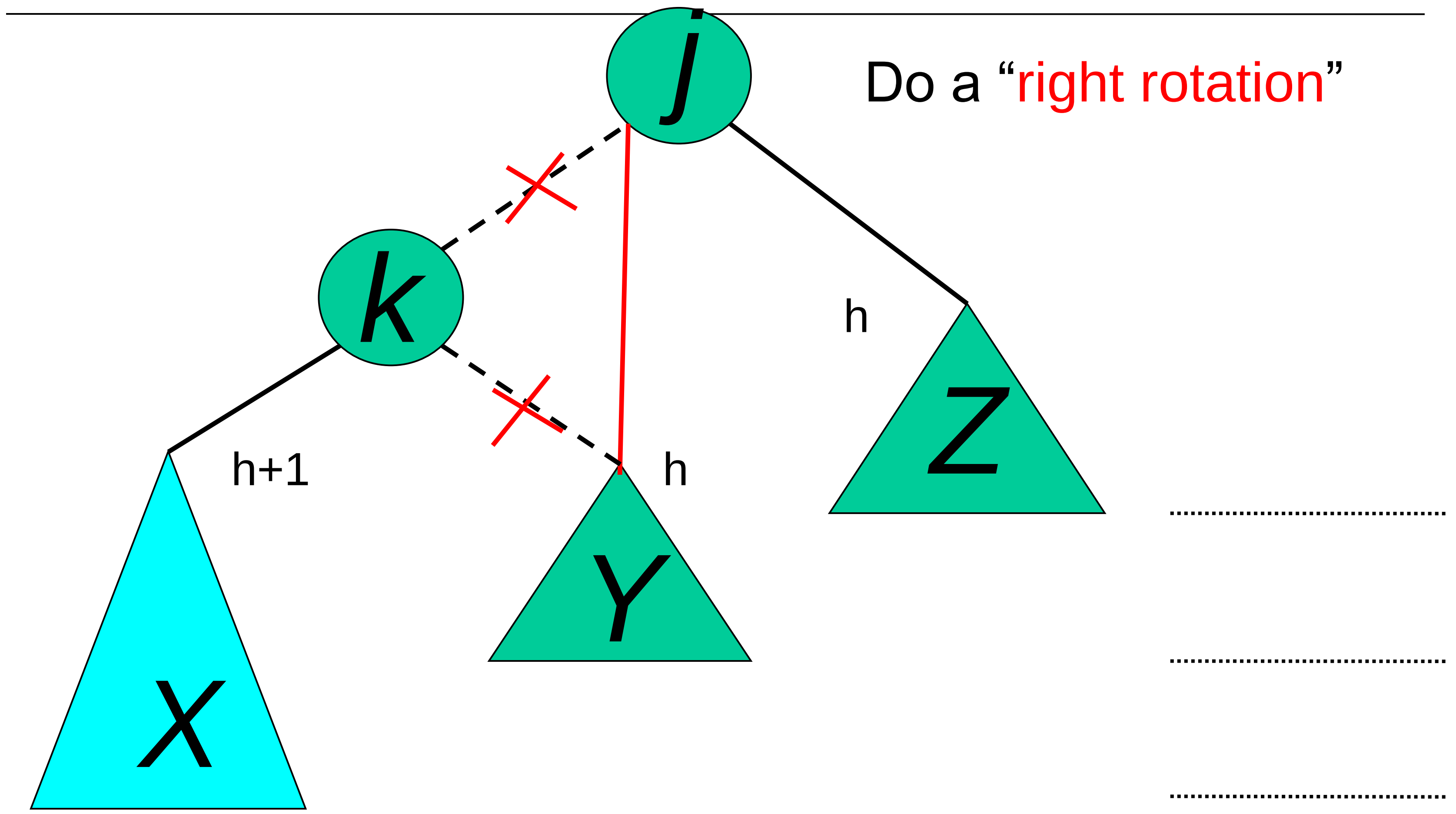


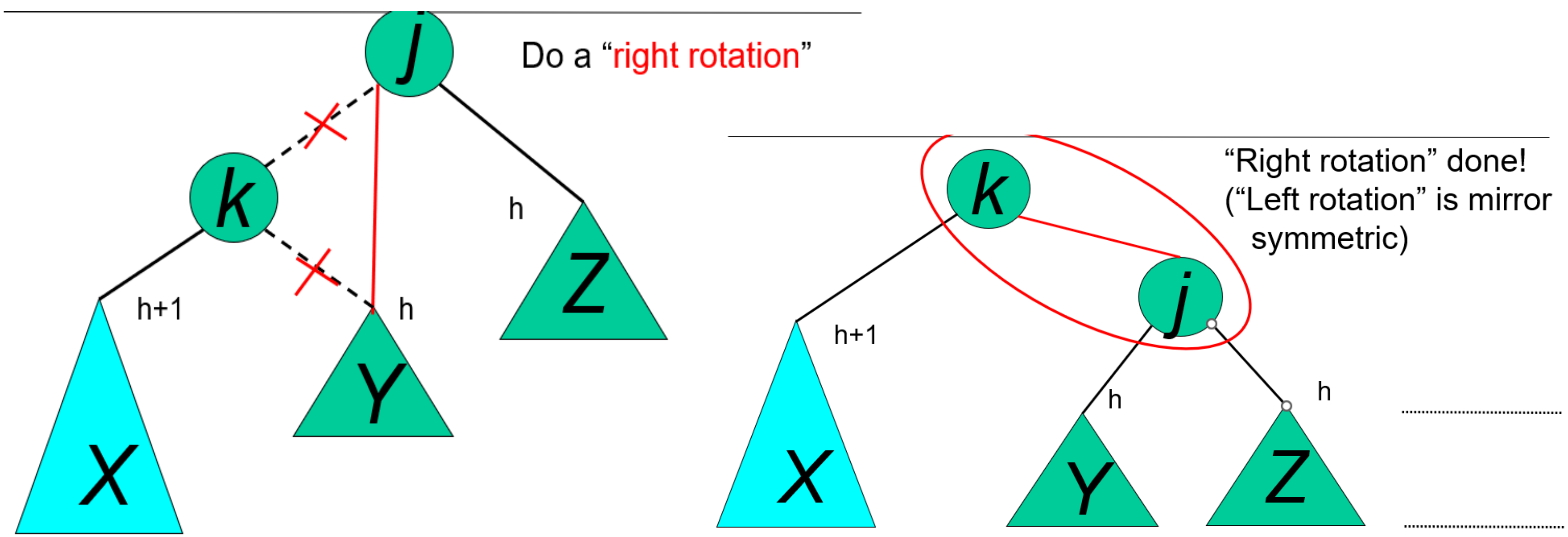
Inserting into X destroys the AVL property at node j

AVL Insertion: LL Case

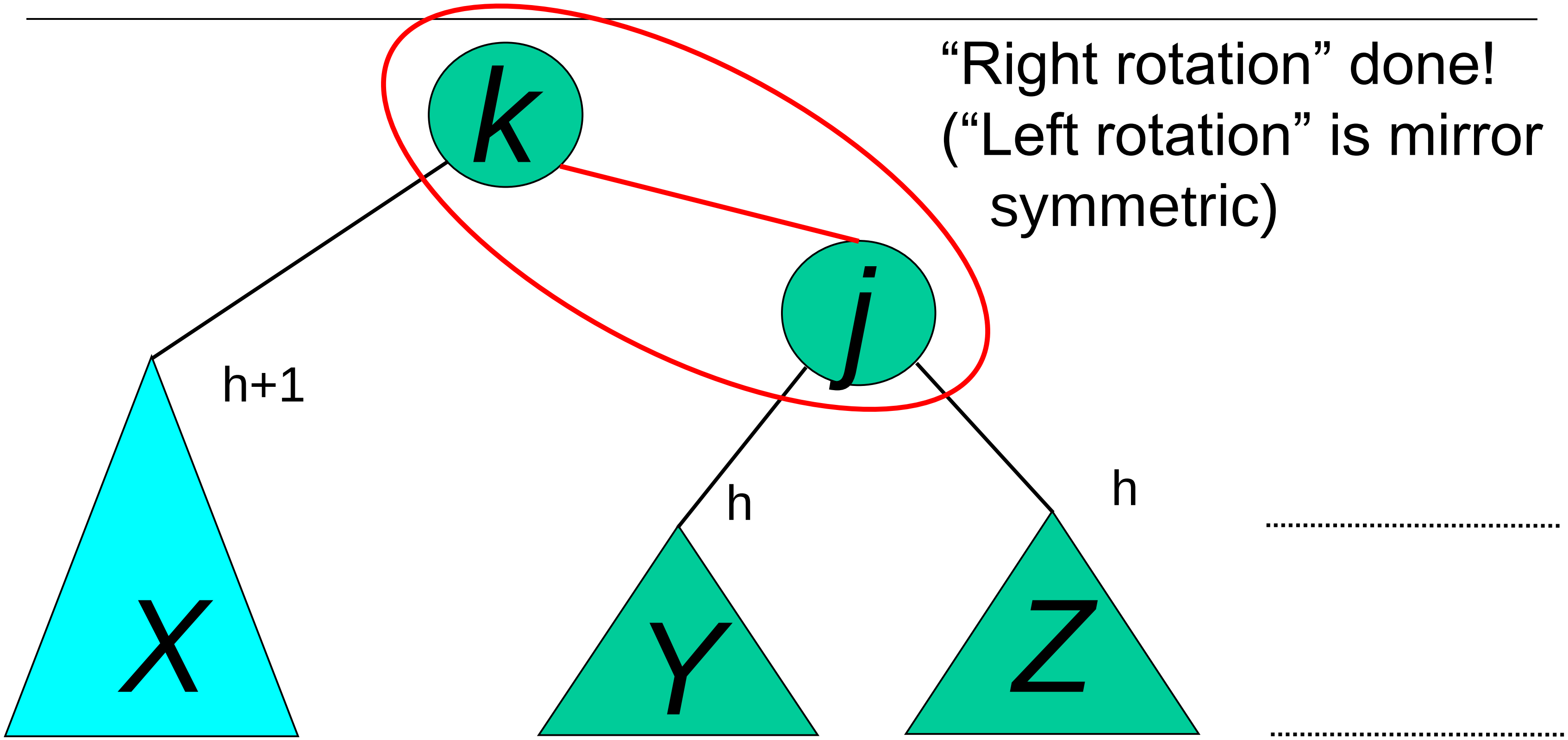


Single right rotation




j
Do a "right rotation"
k
h
h+1
h
Z
Y
X
"Right rotation" done!
("Left rotation" is mirror
symmetric)
k
j
h+1
h
h
X
Y
Z

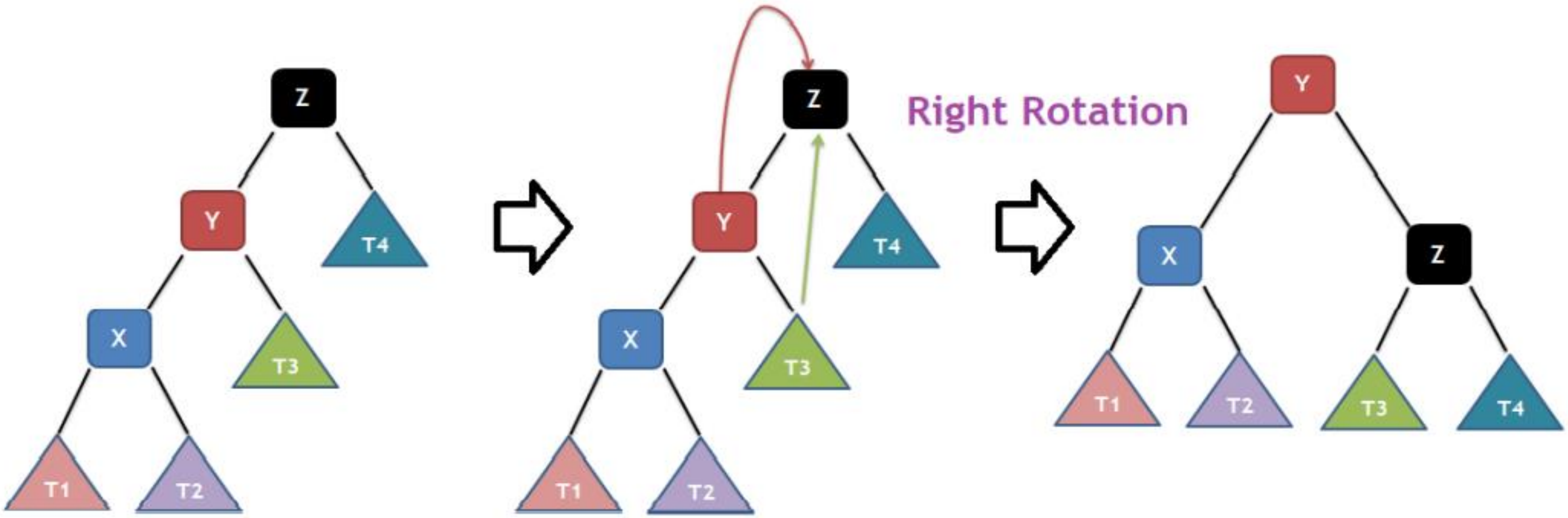
LL Case Completed



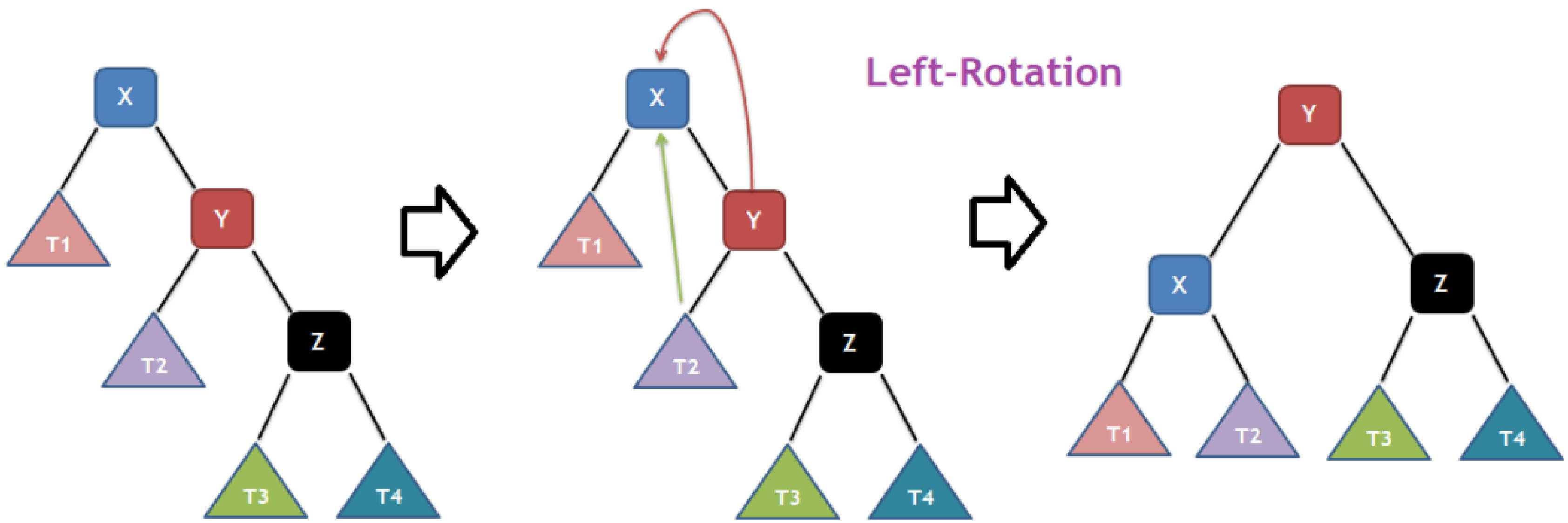
"Right rotation" done!
("Left rotation" is mirror symmetric)

AVL property has been restored!

LEFT-LEFT CASE [LL CASE]

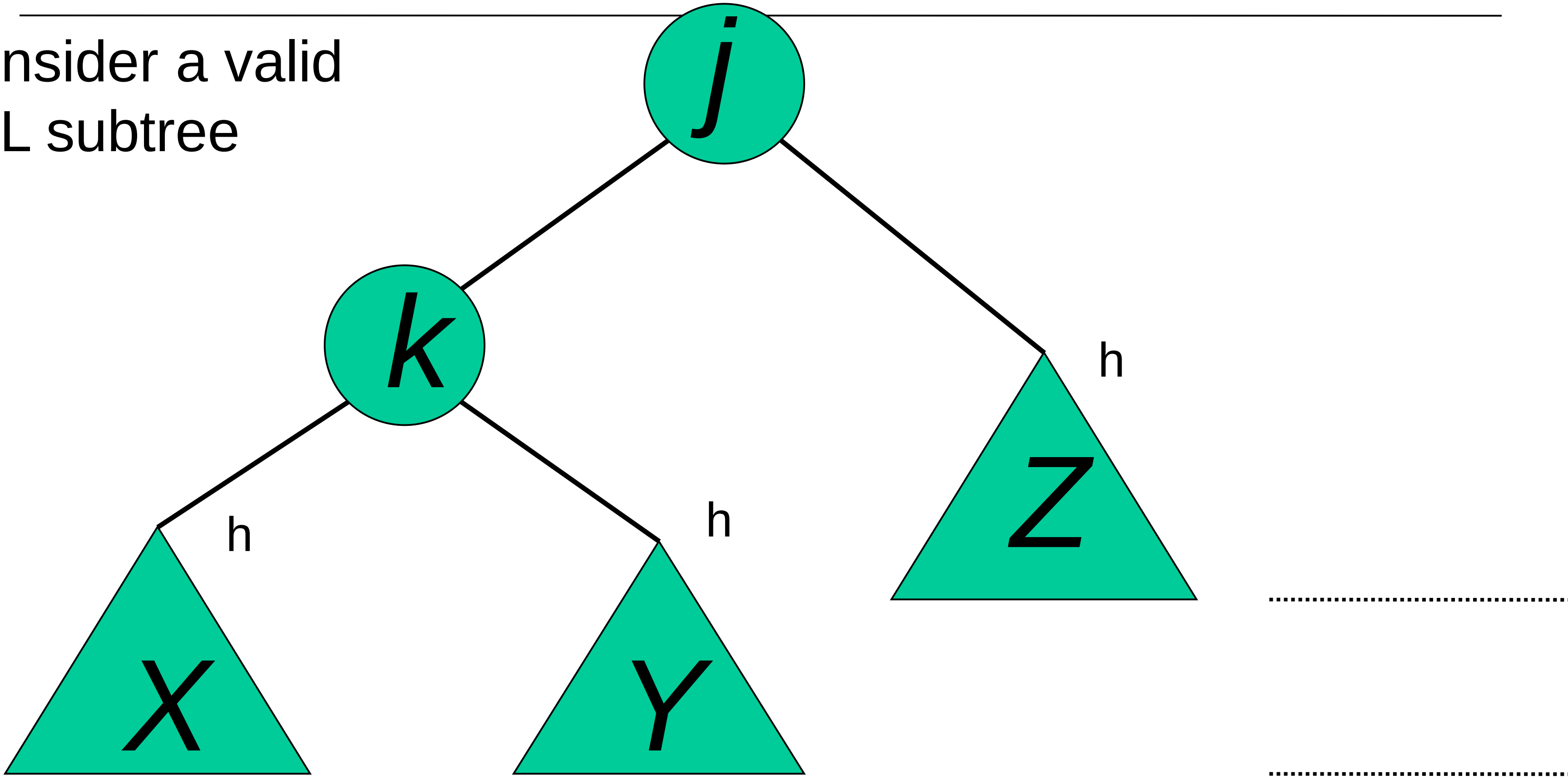


RIGHT-RIGHT CASE [RR-CASE]



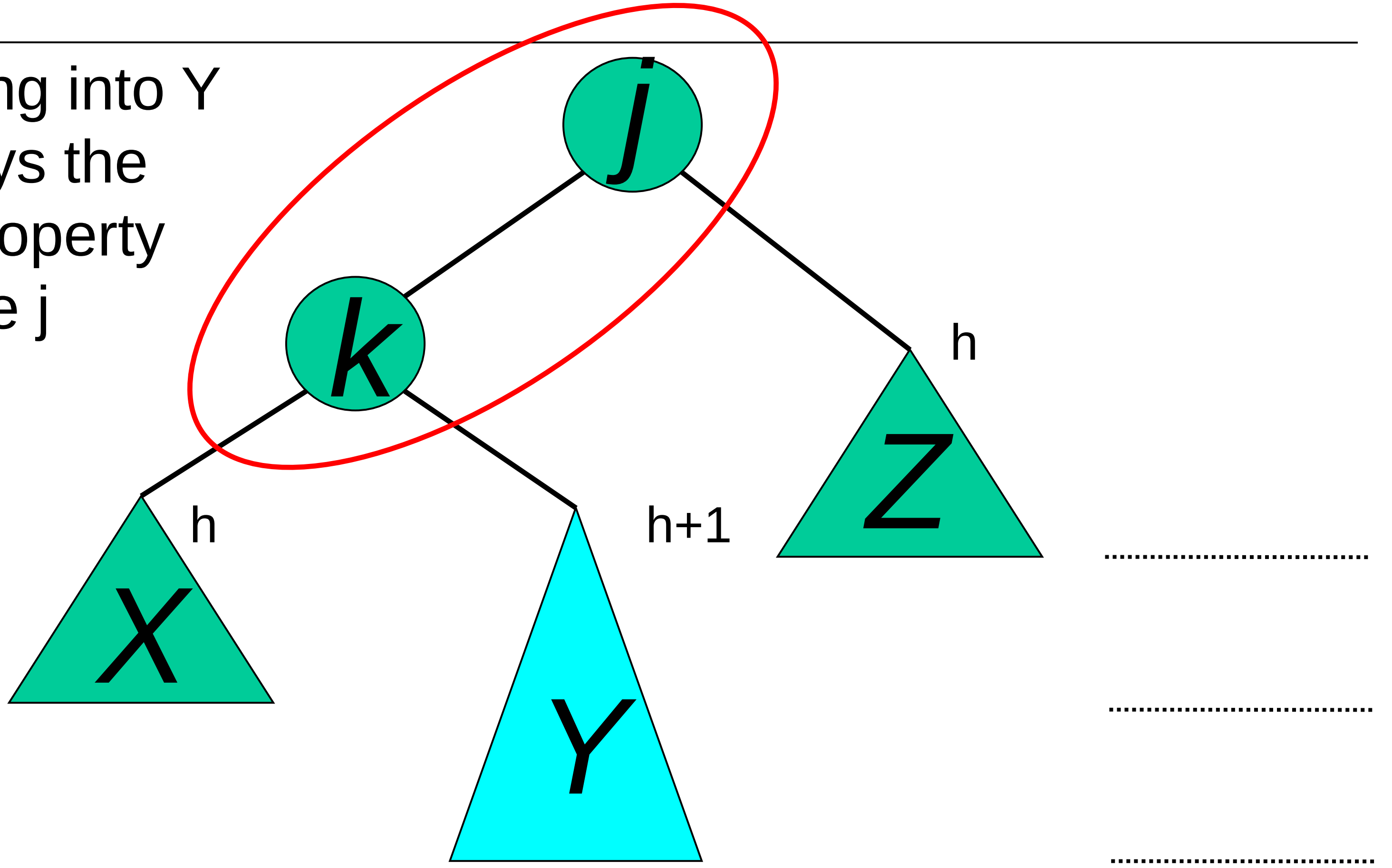
A valid AVL Tree

Consider a valid
AVL subtree



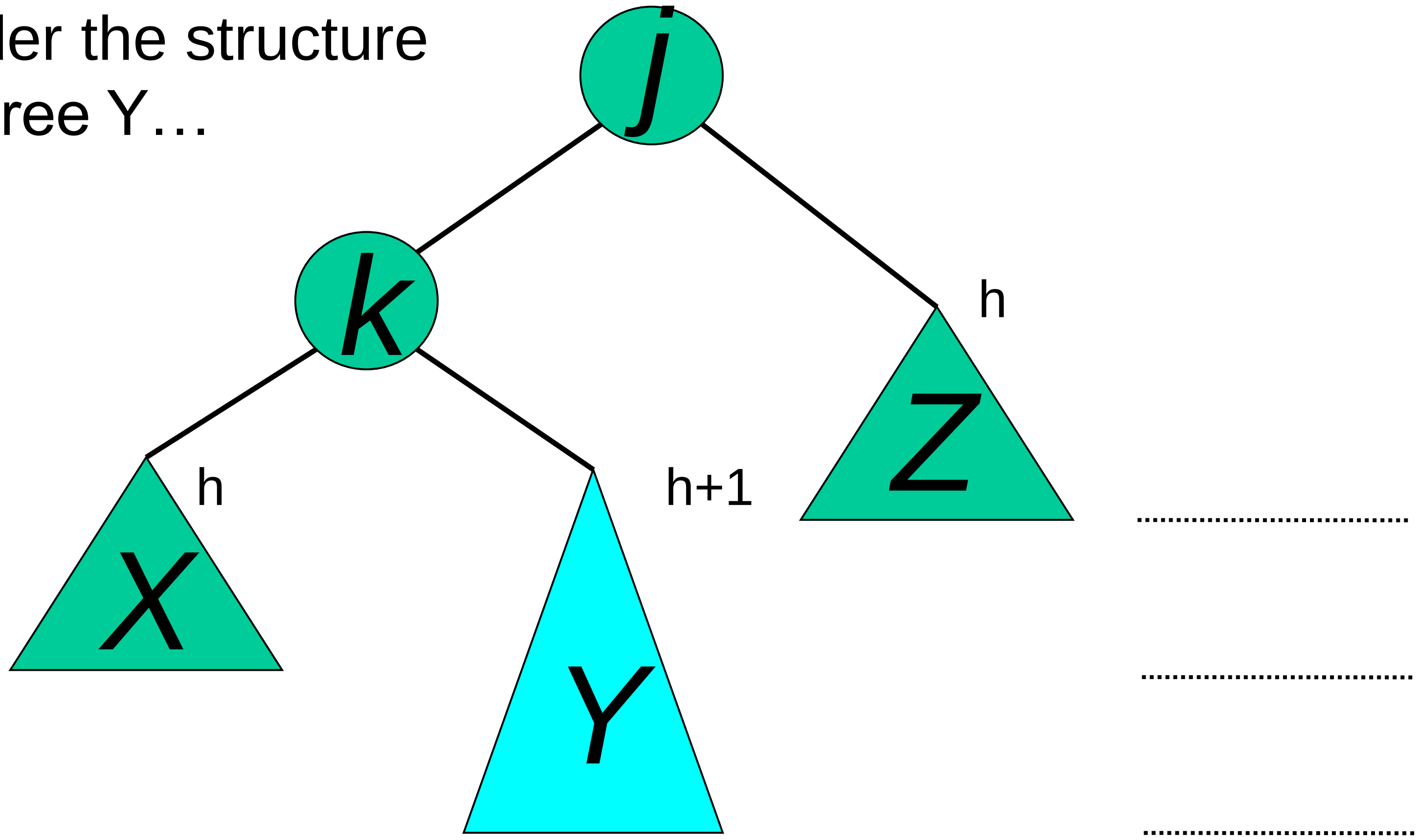
AVL Insertion: LR Case

Inserting into Y destroys the AVL property at node j



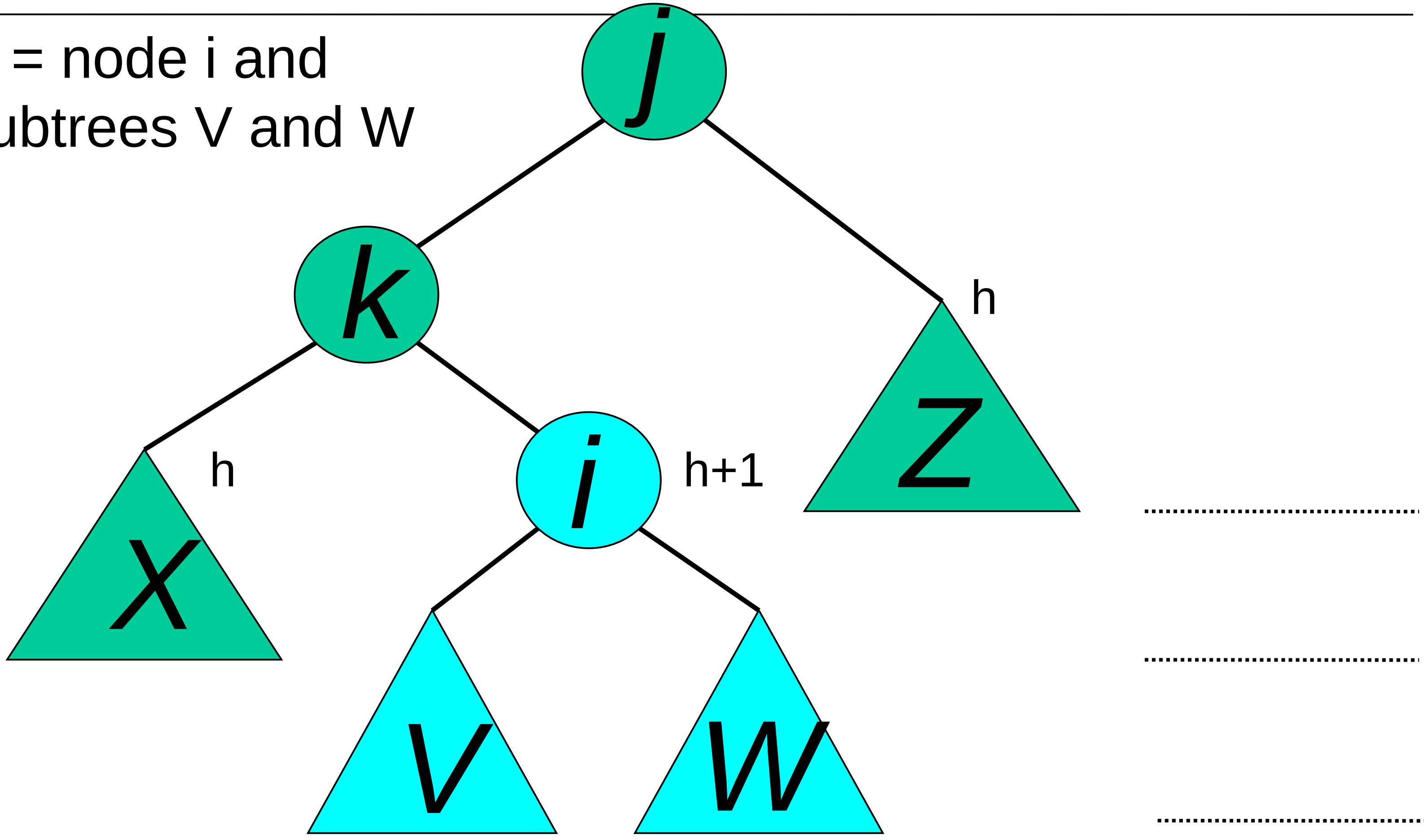
AVL Insertion: LR Case

Consider the structure of subtree Y…

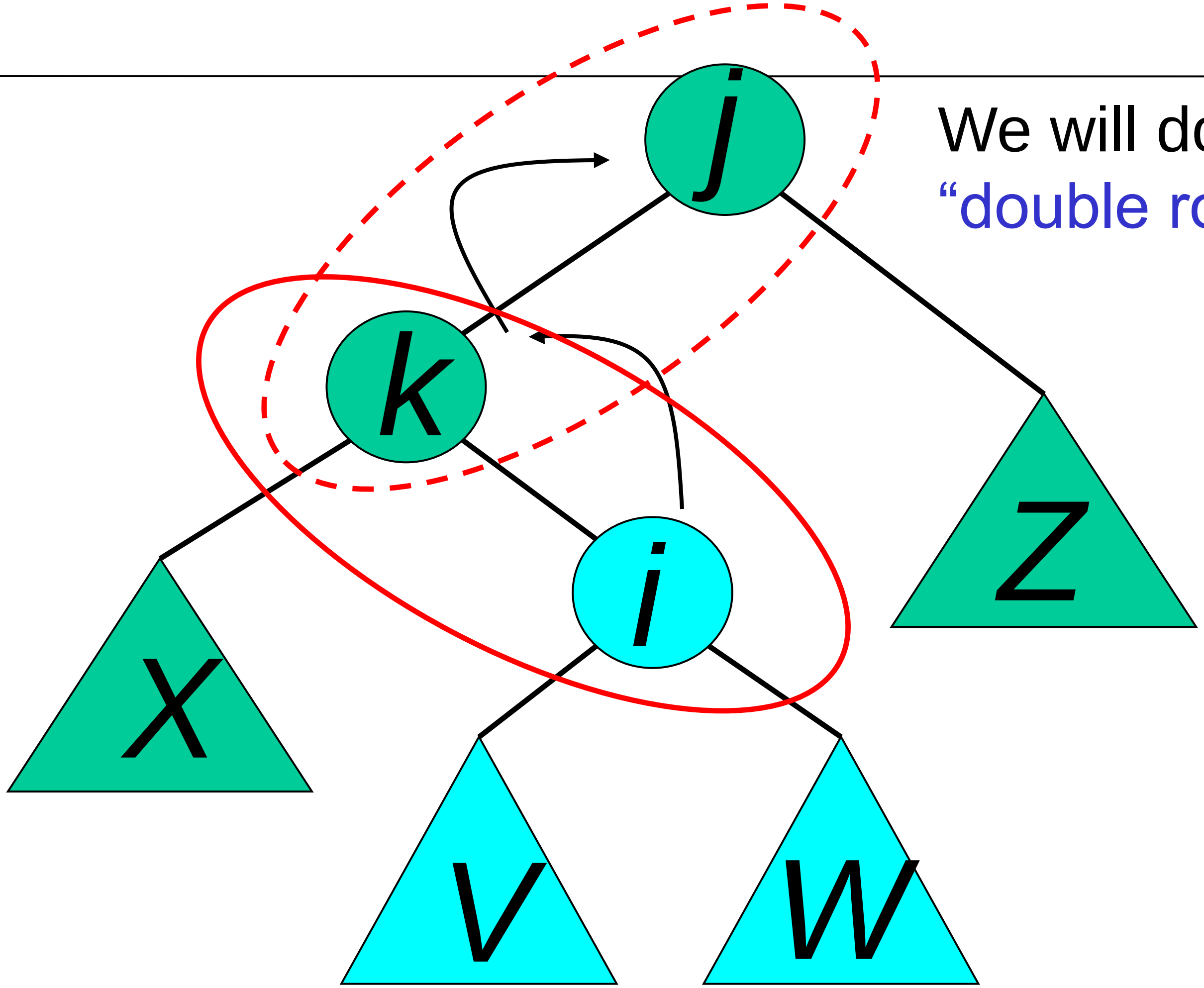


AVL Insertion: LR Case

Y = node i and subtrees V and W



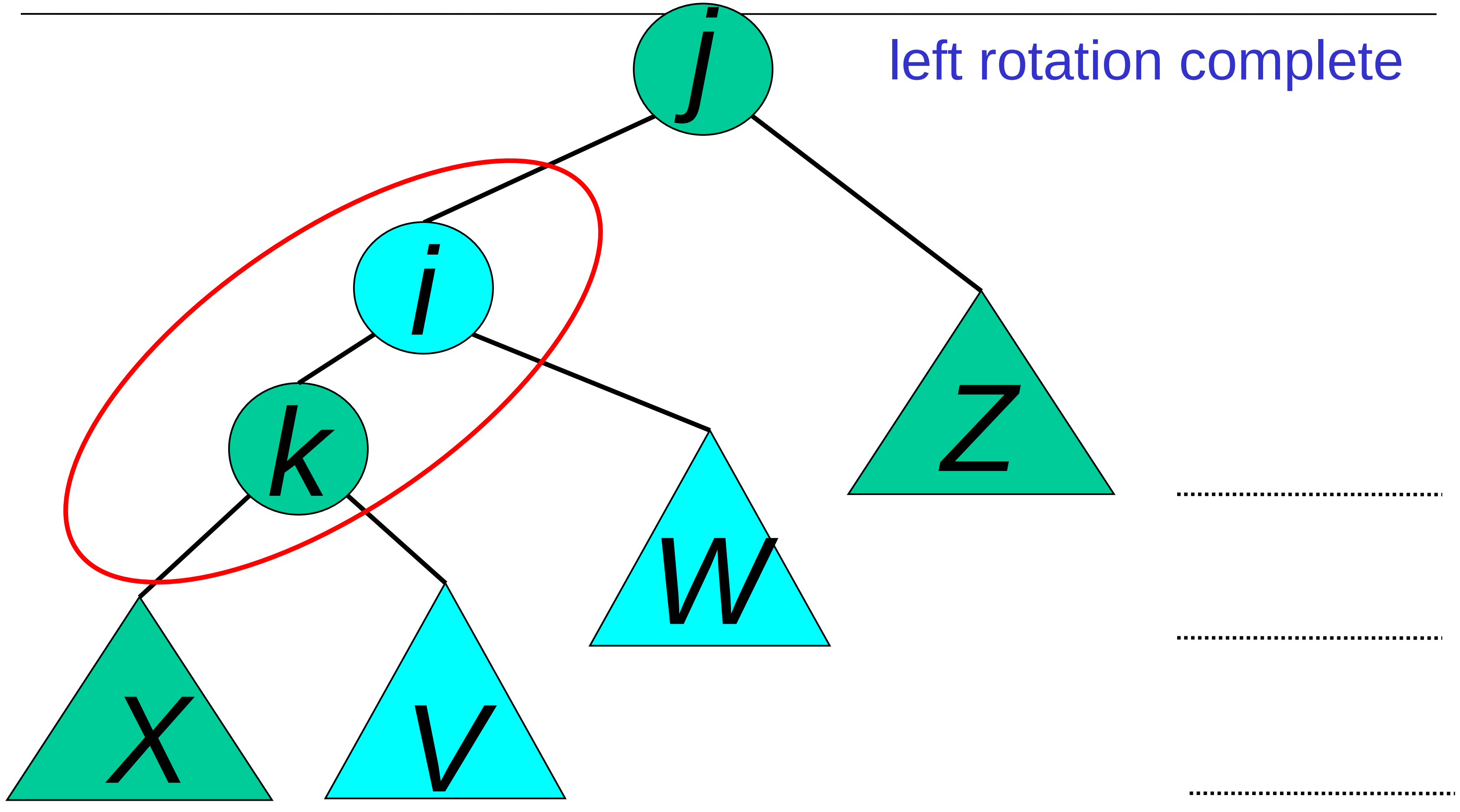
AVL Insertion: LR Case



We will do a left-right "double rotation" . . .

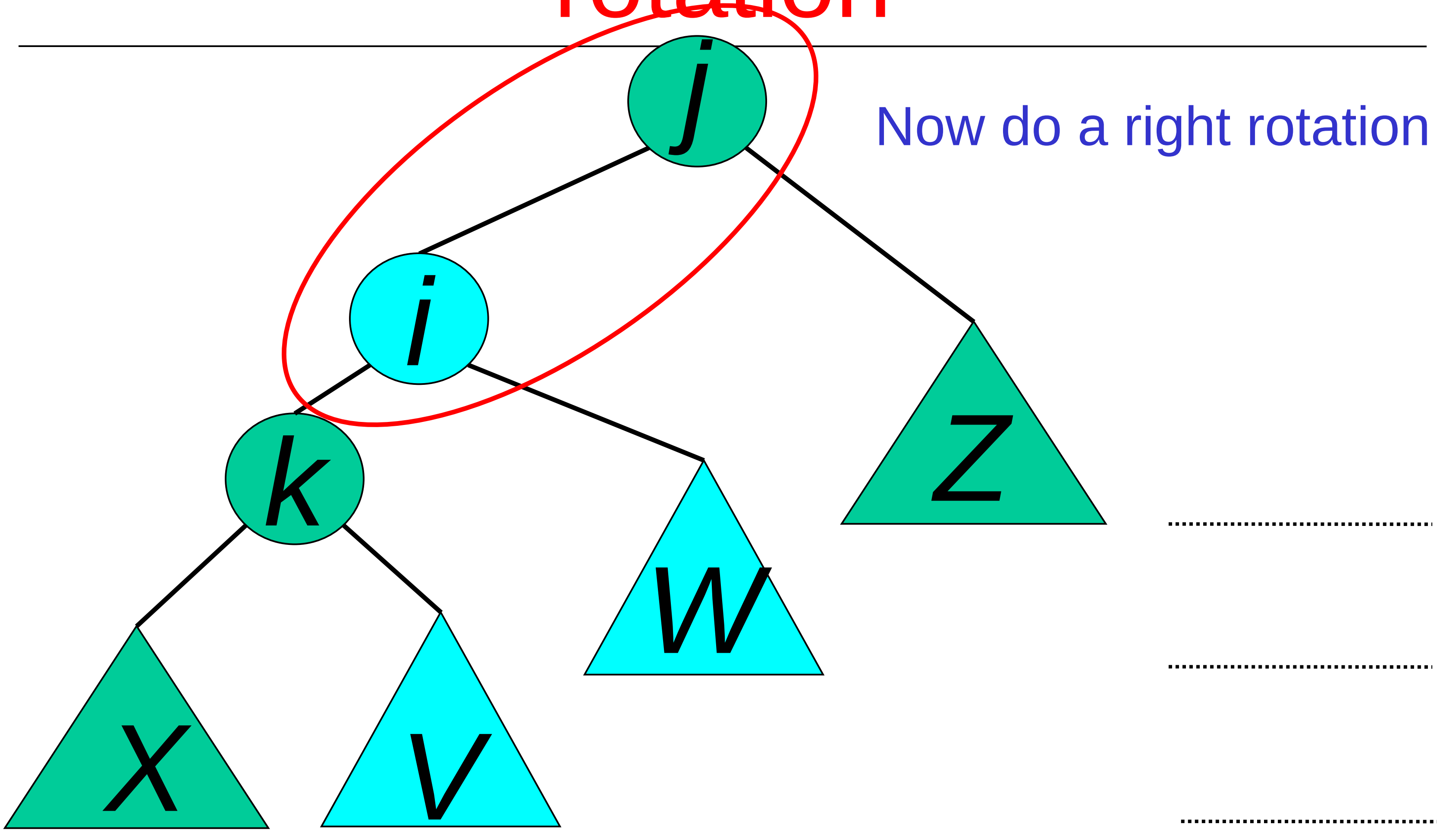
Double rotation : first rotation

left rotation complete



Double rotation : second rotation

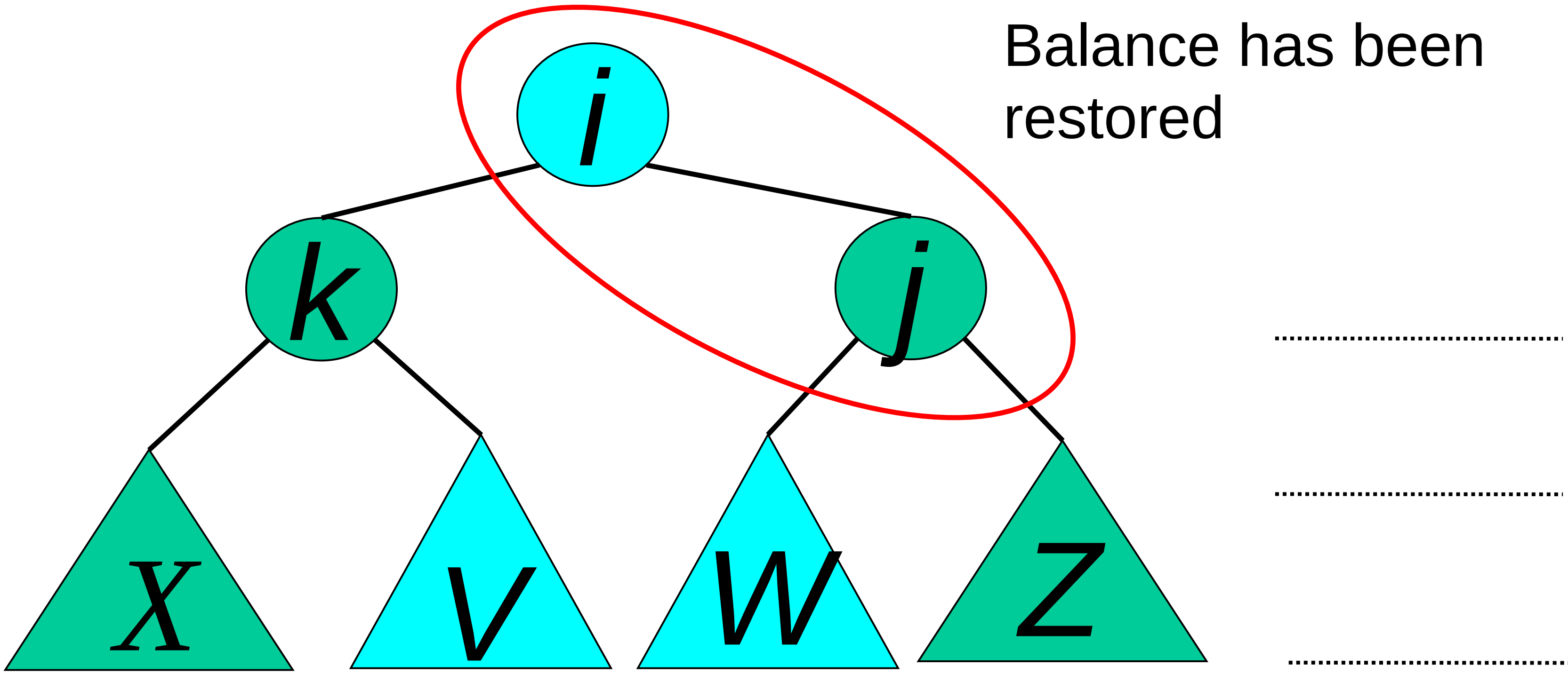
Now do a right rotation



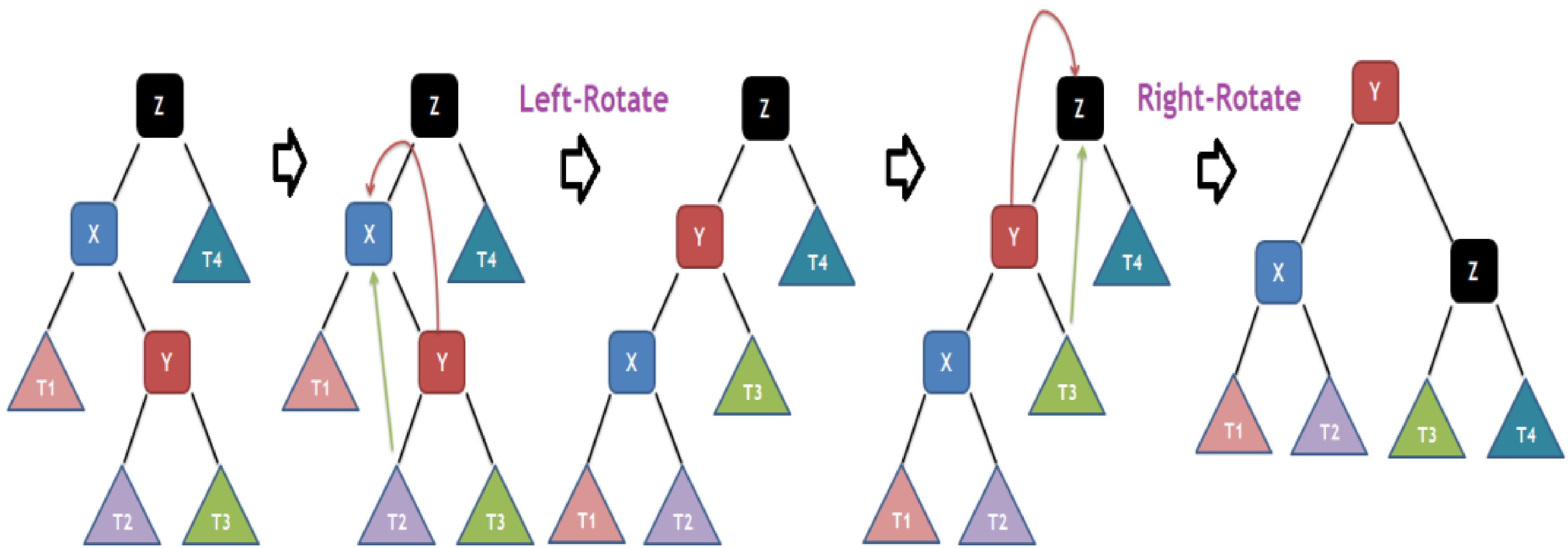
Double rotation : second rotation

right rotation complete

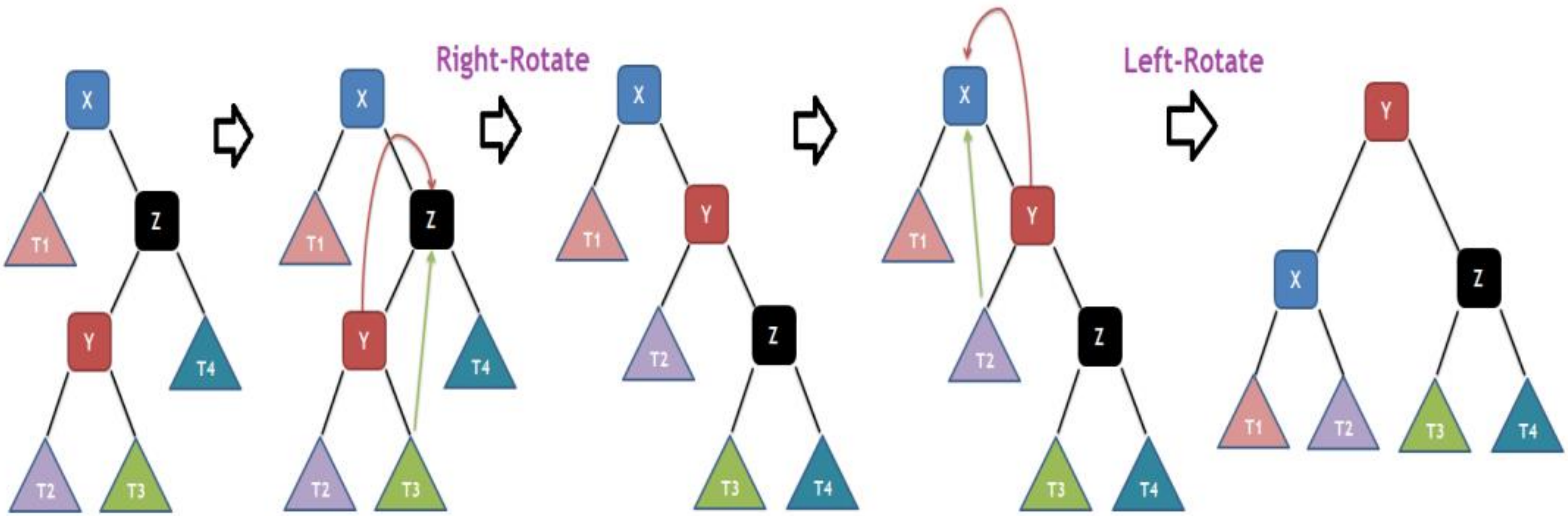
Balance has been restored



LEFT-RIGHT CASE [LR CASE]

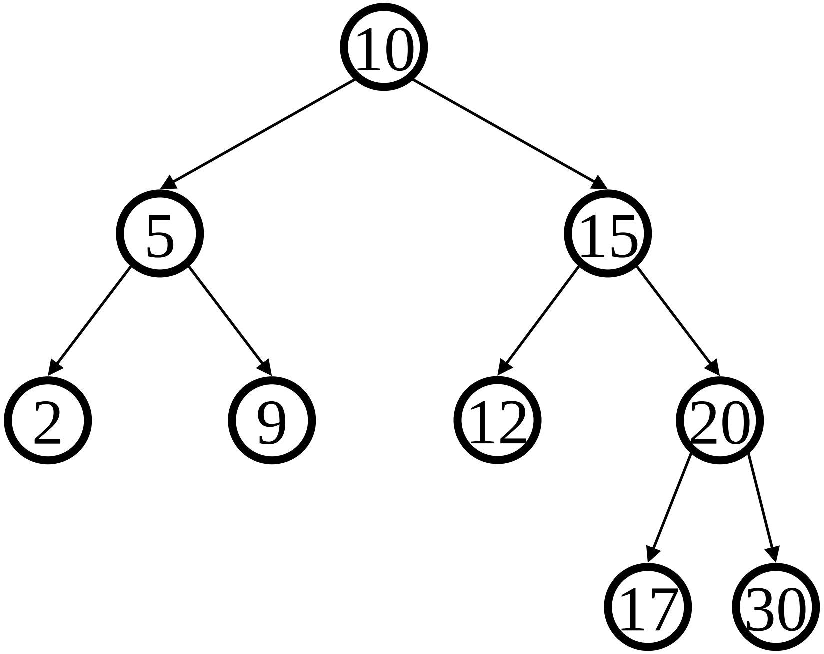


RIGHT-LEFT CASE [RL CASE]

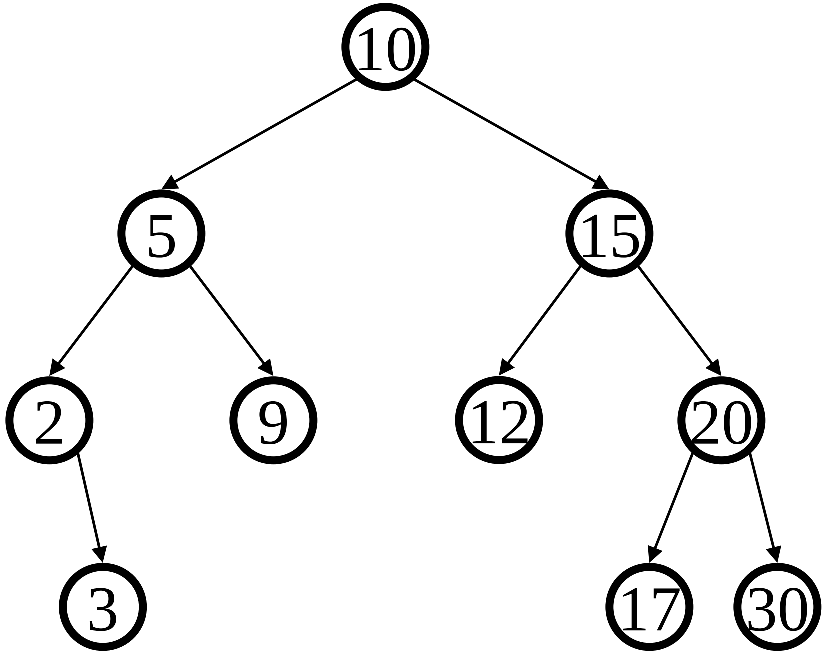


Examples: Insert

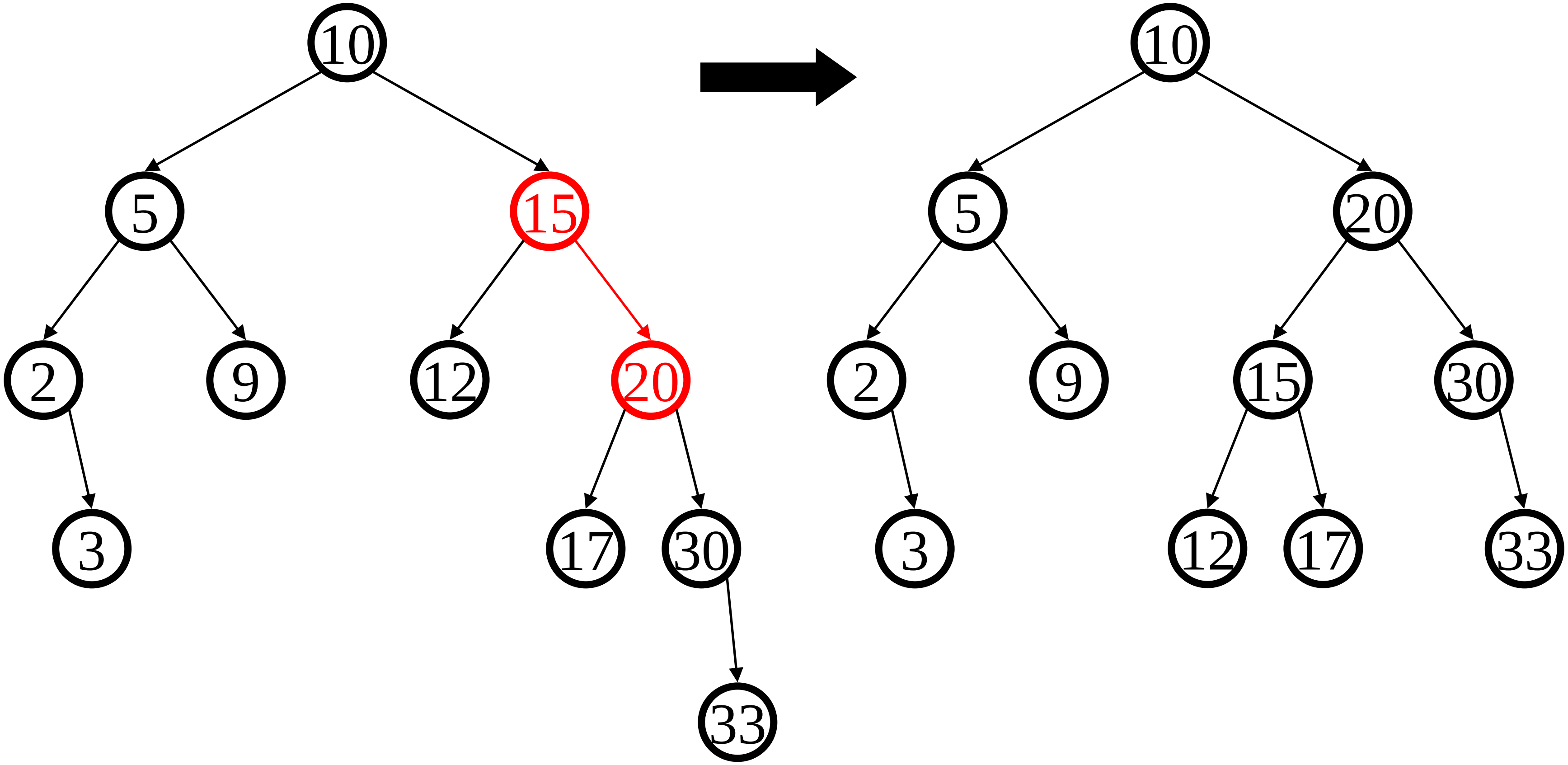
Insert(3)



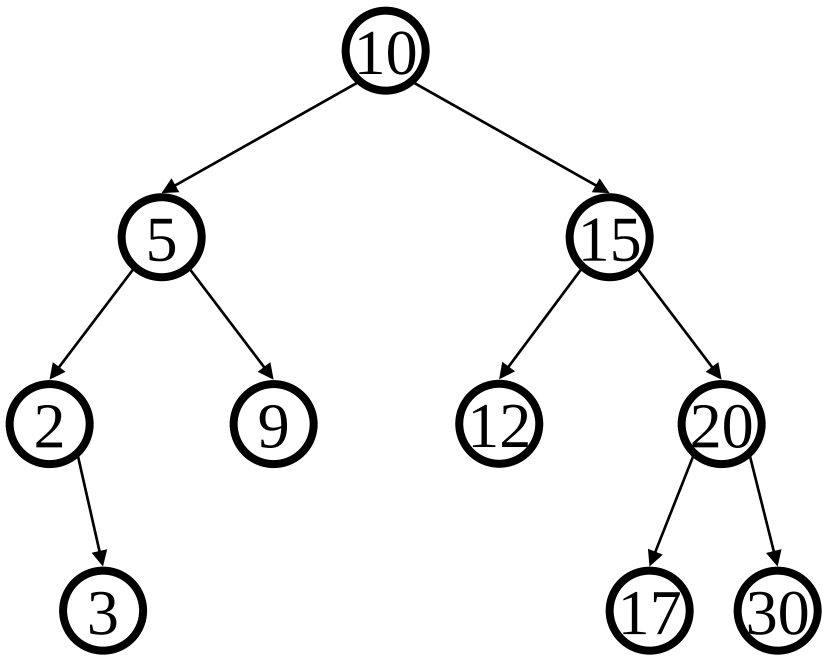
Insert(33)



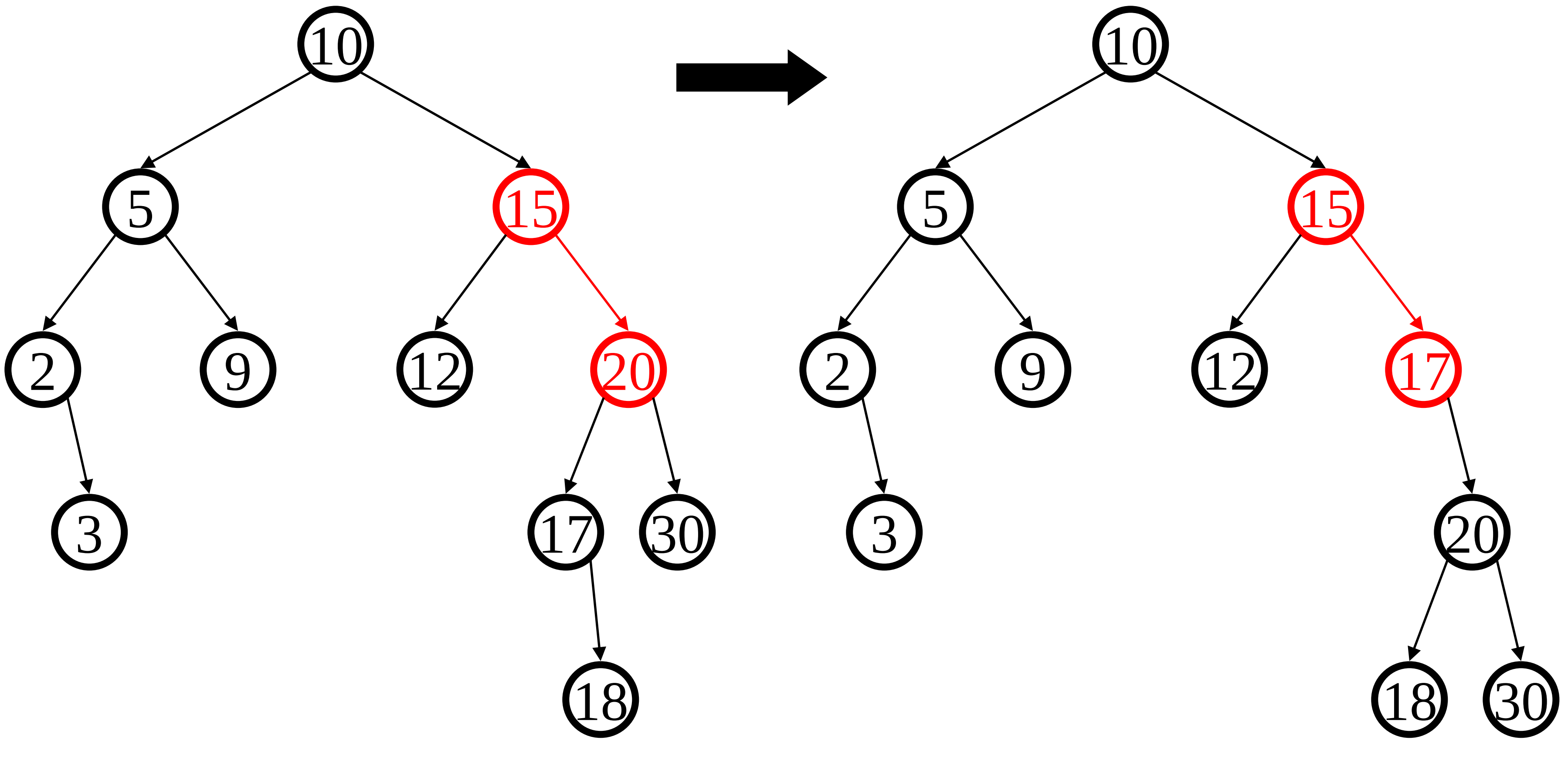
Single Rotation



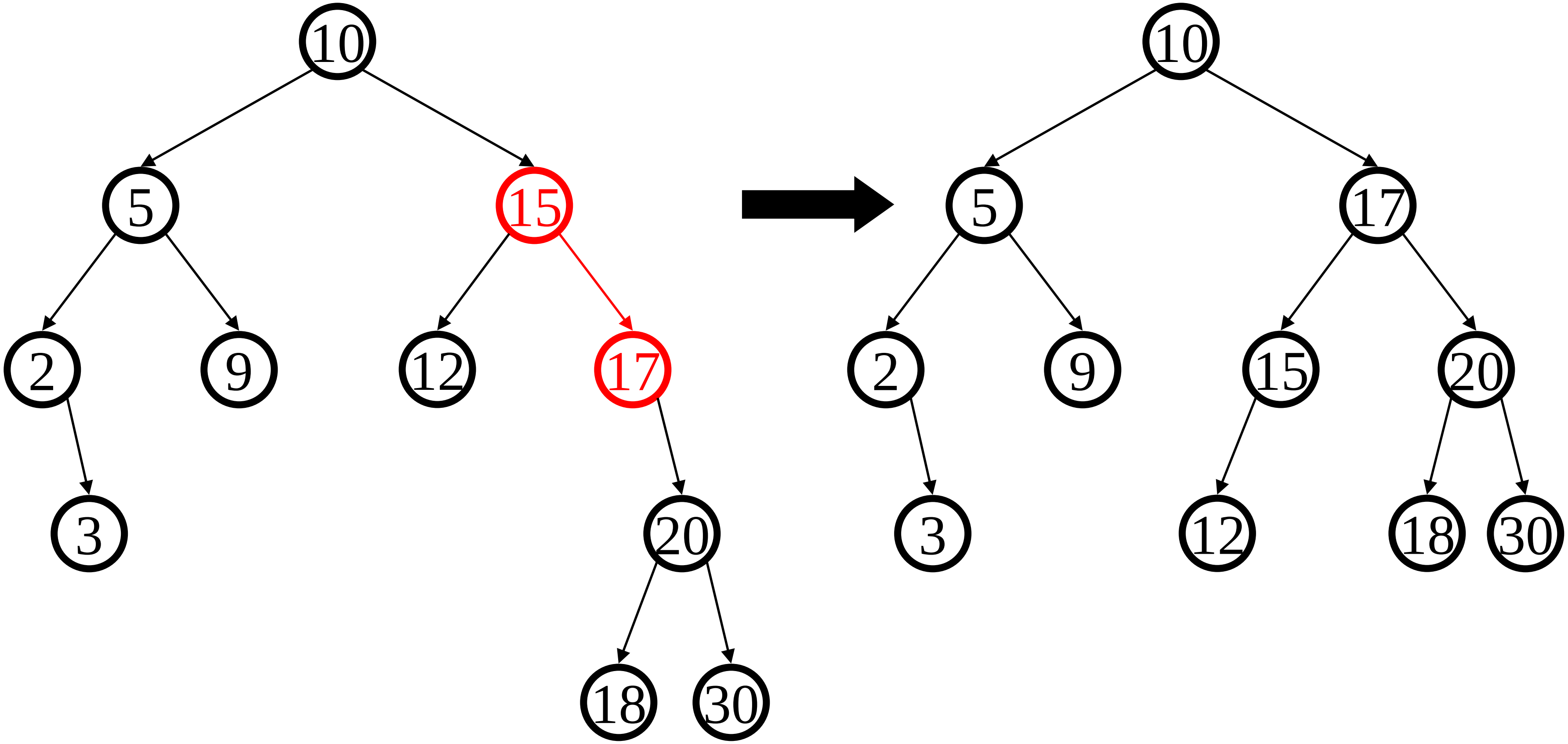
Insert(18)



Double Rotation (Step #1)



Double Rotation (Step #2)



Balance Factor

Function balance_factor(ROOT)

Given a rooted AVL tree denoted by ROOT, this function returns the balance factor of the AVL node.

1. Is the tree empty??

    If ROOT = NULL

        Return 0

2. Otherwise, return balance factor

    Return Height (ROOT->LEFT) - Height(ROOT->RIGHT)

createNode()

Function Create_AVL_Node(KEY)

This function creates an AVL node and initializes its DATA field to the value contained in KEY. It returns address of the created node. NEWW is the local tree pointer

1. Create a node

   ROOT <= AVL_NODE (Ex. malloc() in C)

2. Is the node usable??

   If ROOT = NULL

   Write('AVAIL underflow, creation failed')

   Return NULL

3. Initialize the node

   NEWW->DATA = KEY

   NEWW->LEFT= NEWW->RIGHT = NULL

   NEWW-> HEIGHT = 1

4. Return node address Return NEWW

AVL Insert Algorithm

Function insert_AVL(ROOT, KEY)

Given an AVL tree rooted at ROOT, this function inserts an AVL node with DATA value contained in KEY and returns the updated tree pointer to the height balanced tree. BAL is local integer variable

1. Is the tree empty??

    If ROOT = NULL

        ROOT = createNode()

        return ROOT

2. Insert the node appropriately [recursively]

    If KEY < ROOT->DATA

        ROOT->LEFT = Insert_AVL(ROOT->LEFT,KEY)

    Else If **KEY > ROOT->DATA**

        ROOT->RIGHT = Insert_AVL(ROOT->RIGHT,KEY)

    Else **Return ROOT**

3. **Update the height of ROOT**

    ROOT->HEIGHT = MAX(Height (ROOT->LEFT), Height (ROOT->RIGHT)) + 1

4. **Validate balance factor of ROOT**

    BAL = balance_factor(ROOT)

5. Balance AVL tree, **LL**-Case

If BAL > 1 AND KEY < ROOT->LEFT->DATA

Return (Right_Rotate_AVL(ROOT))

6. Balance AVL tree, **RR**-Case

If BAL < -1 AND KEY > ROOT->RIGHT->DATA

Return (Left_Rotate_AVL(ROOT))

7. Balance AVL tree, **LR**-Case

If BAL > 1 AND KEY > ROOT->LEFT->DATA

ROOT->LEFT = Left_Rotate_AVL(ROOT->LEFT)

Return (Right_Rotate_AVL(ROOT))

8. Balance AVL tree, **RL**-Case

If BAL < -1 AND KEY < **ROOT->RIGHT->DATA**

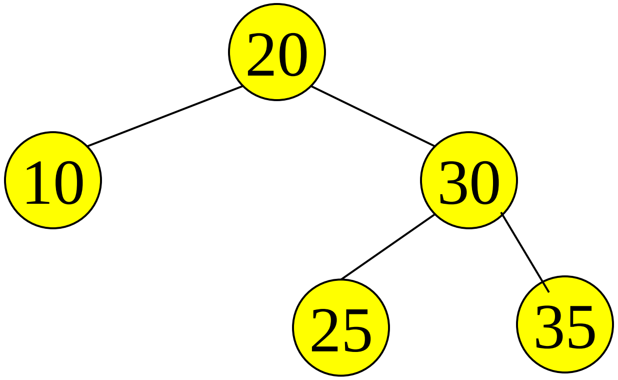
ROOT->RIGHT = Right_Rotate_AVL(ROOT->RIGHT)

Return (Left_Rotate_AVL(ROOT))

9. Return AVL tree, if no balancing required

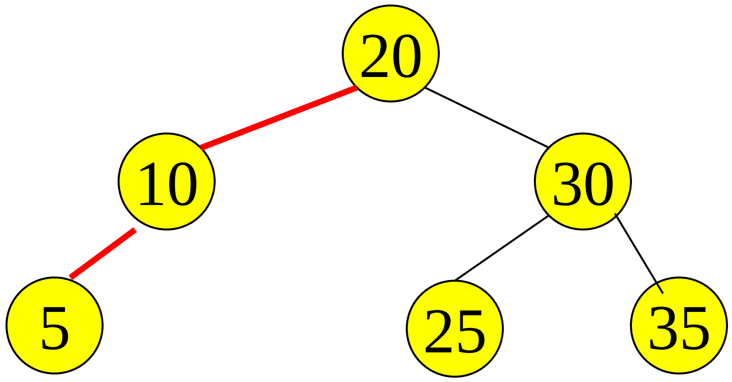
Return ROOT

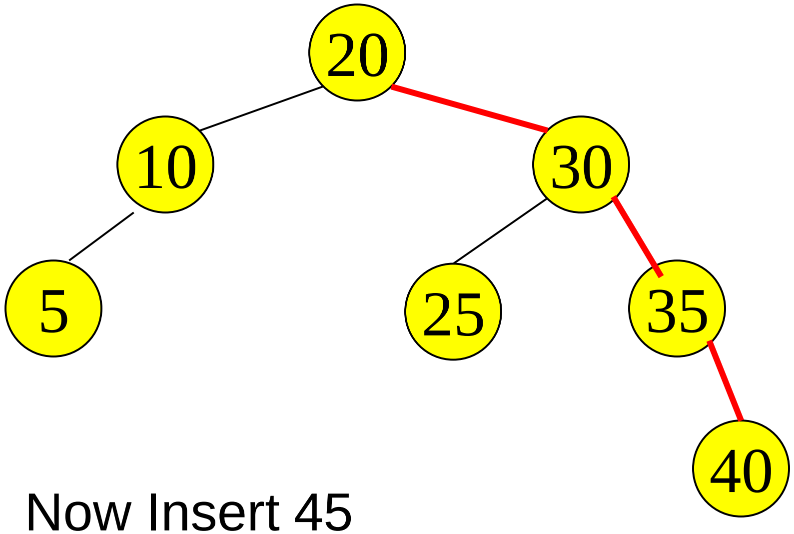
Few more Examples of Insertions in an AVL Tree



Insert 5, 40

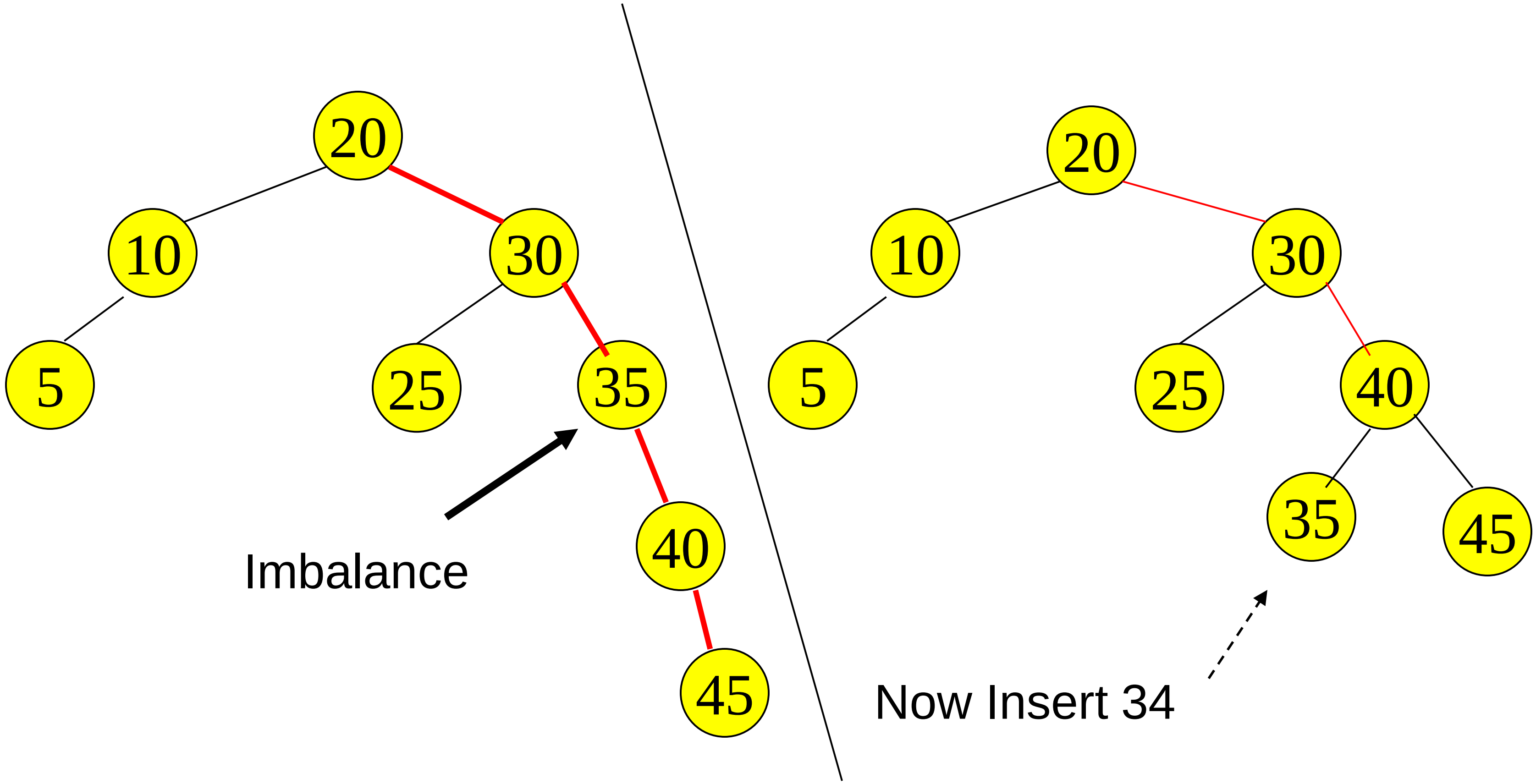
Example of Insertions in an AVL Tree



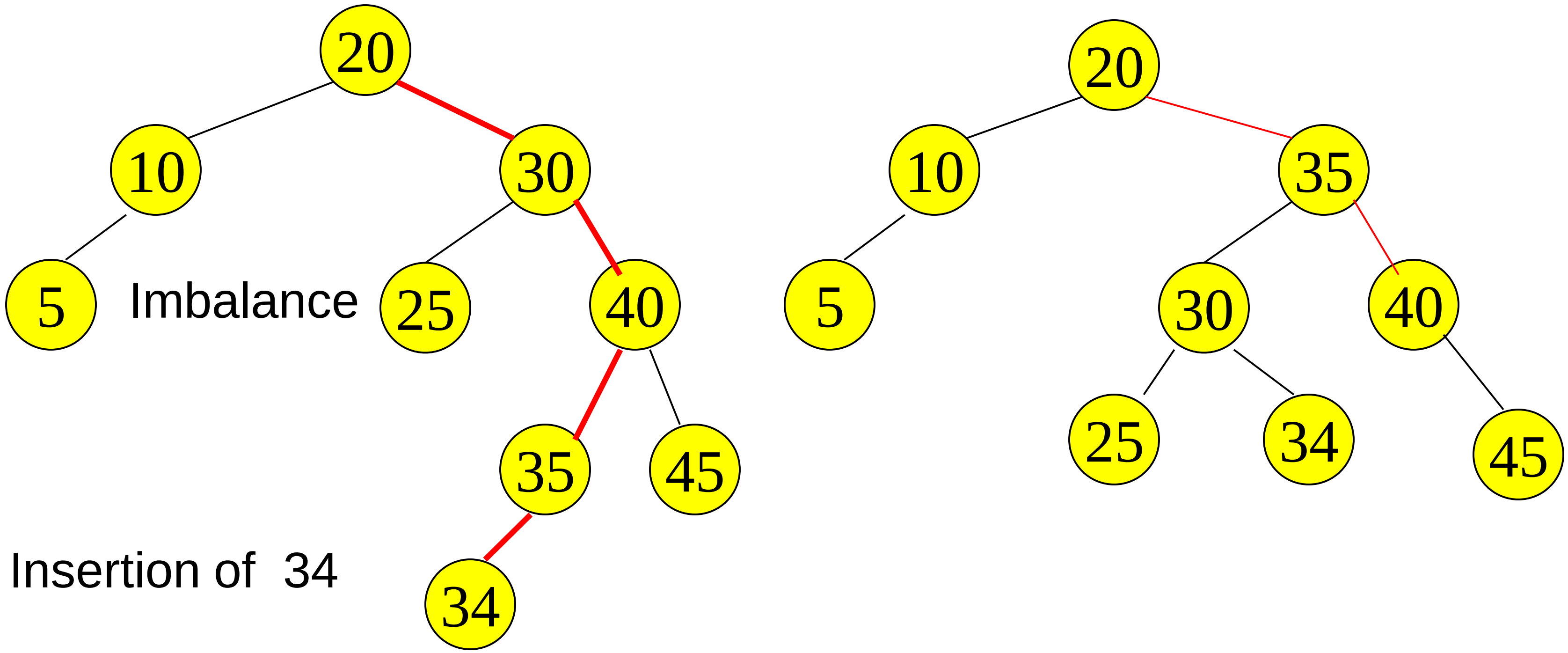


Now Insert 45

Single rotation (RR case)



Double rotation (RL case)



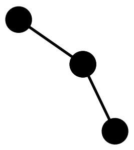
AVL Insert Algorithm Summary

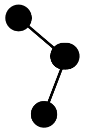
Find spot for value

Add new node

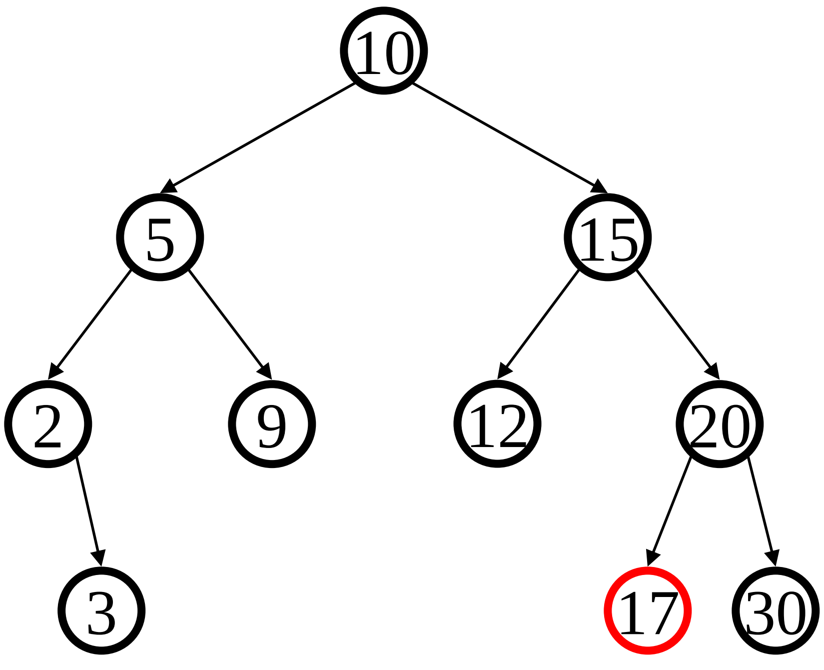
Search back up looking for imbalance

If there is an imbalance:

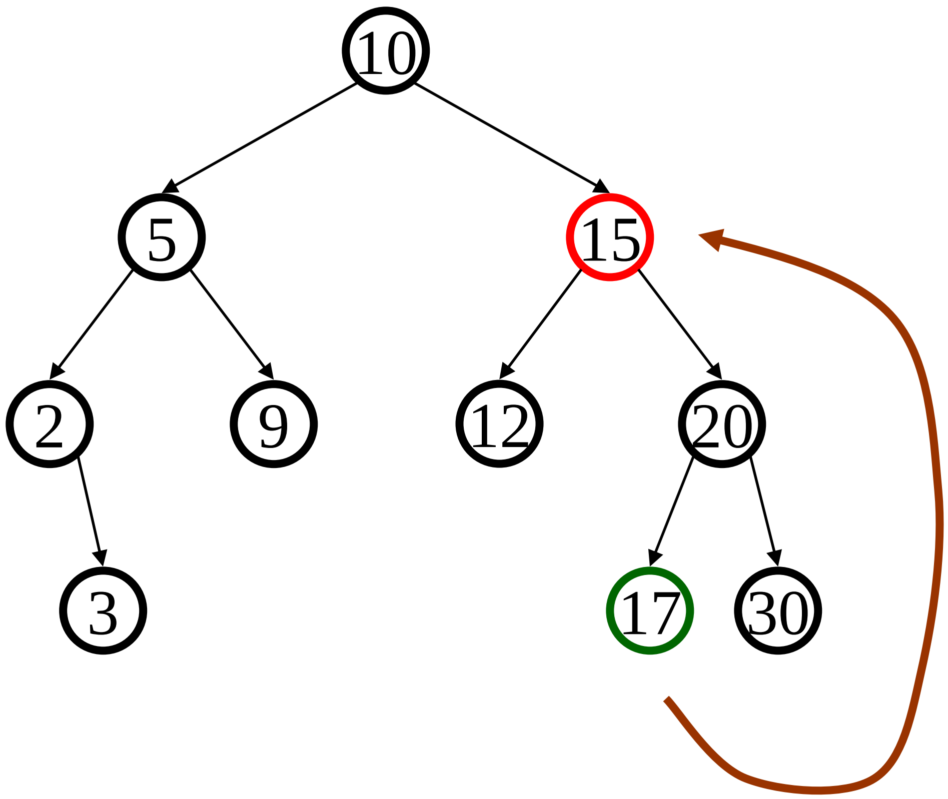
case #1: Perform single rotation 

case #2: Perform double rotation 

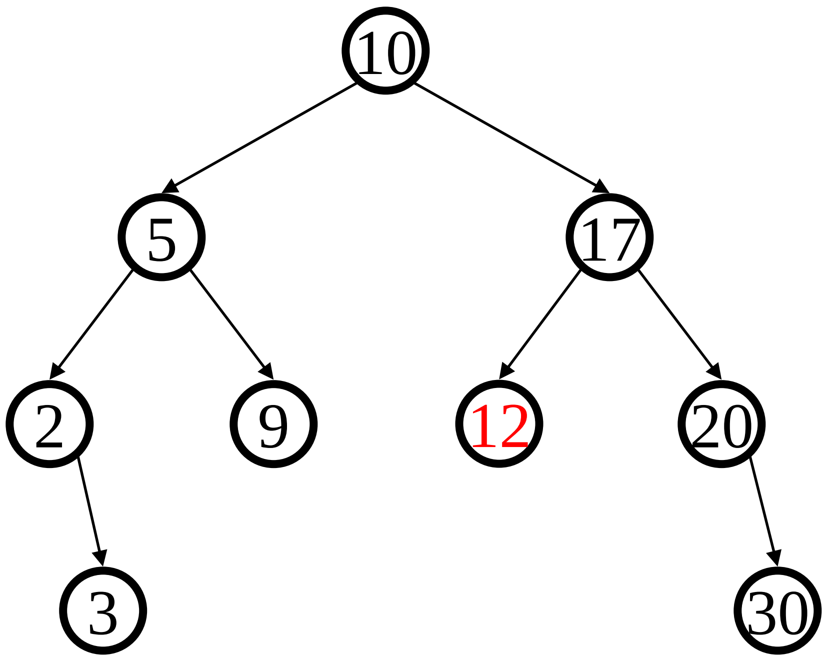
Deletion : 17



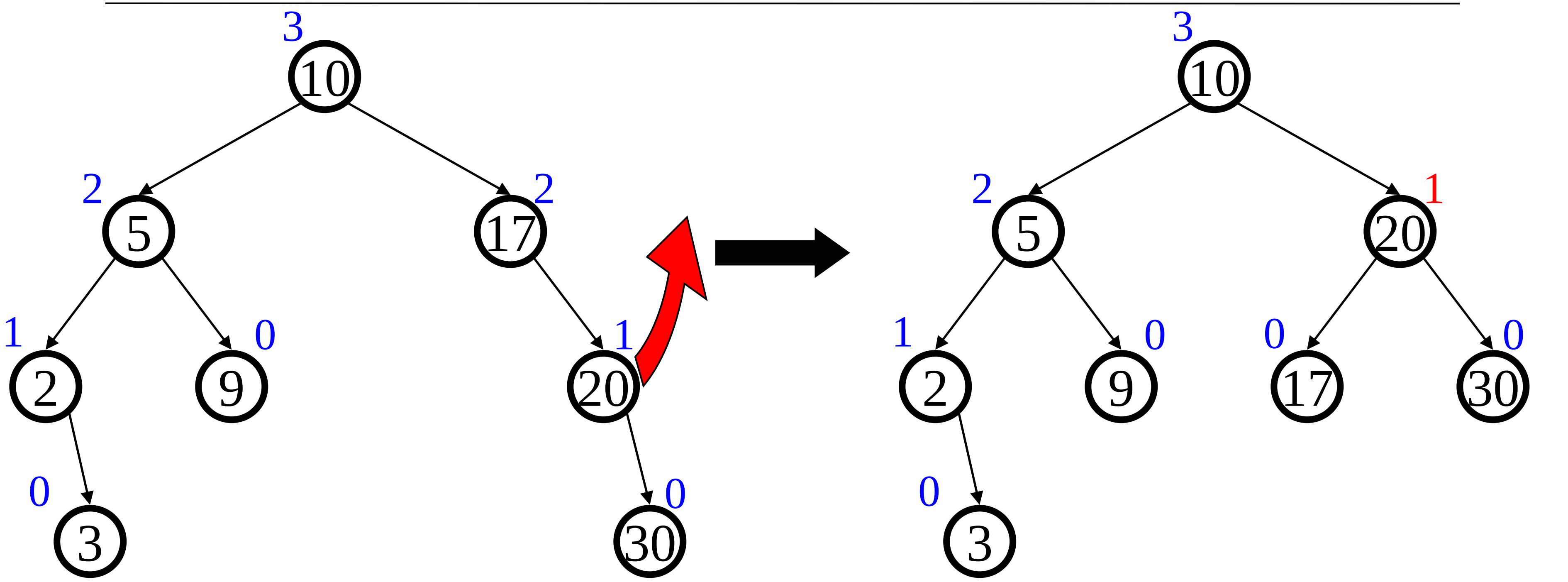
Delete(15)



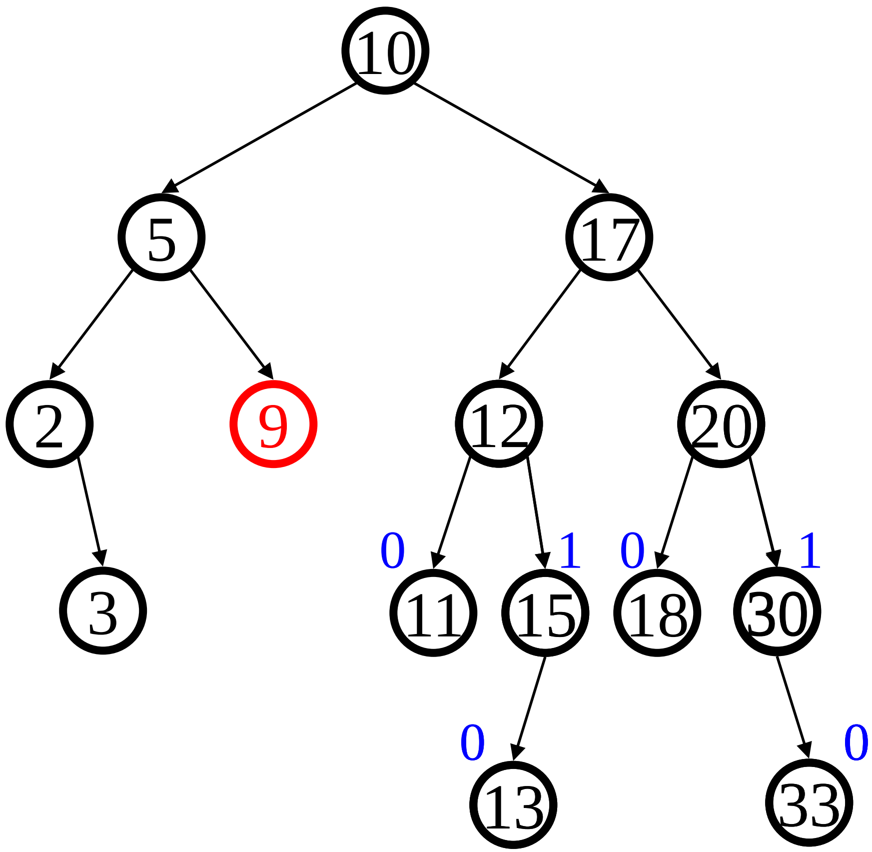
Delete(12)



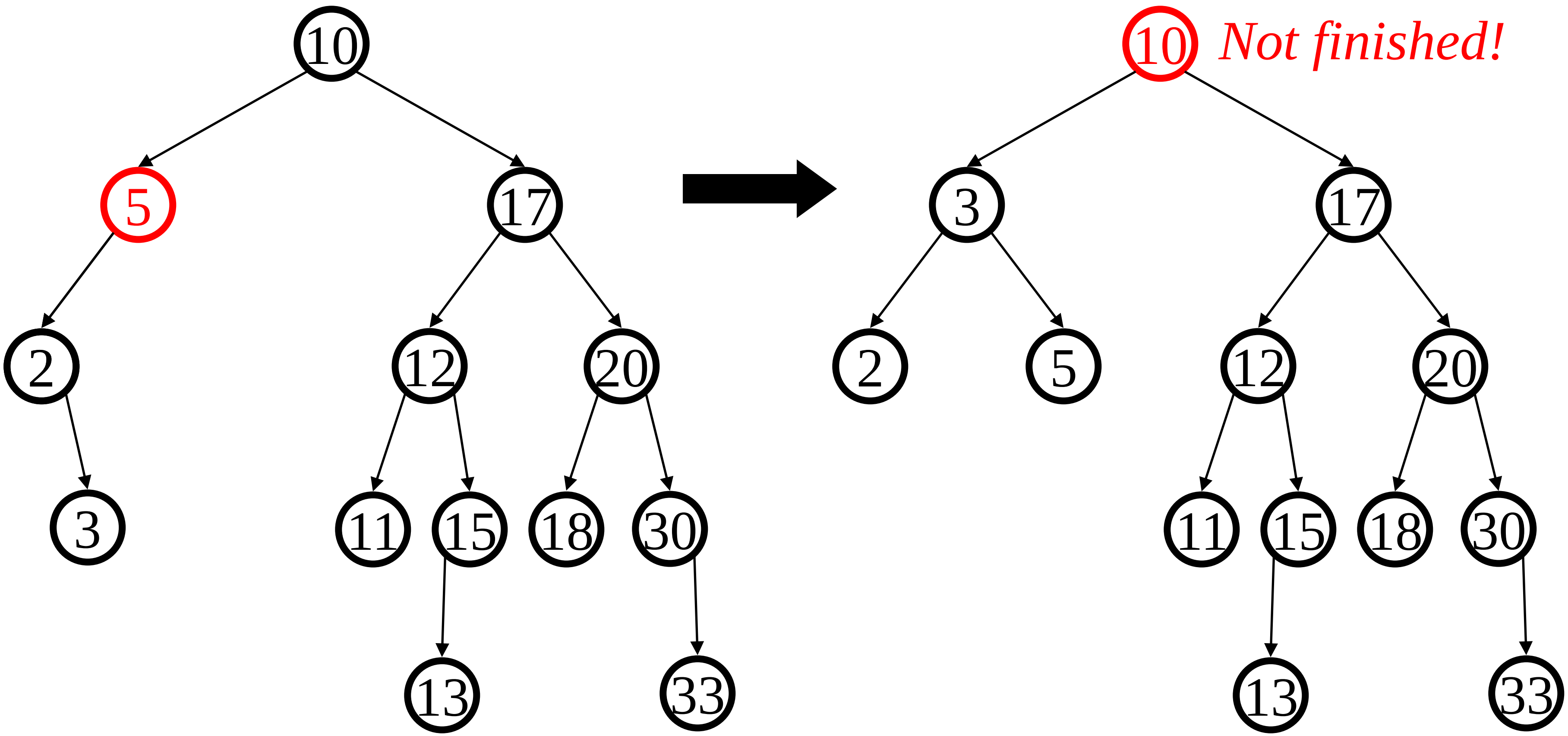
Single Rotation on Deletion



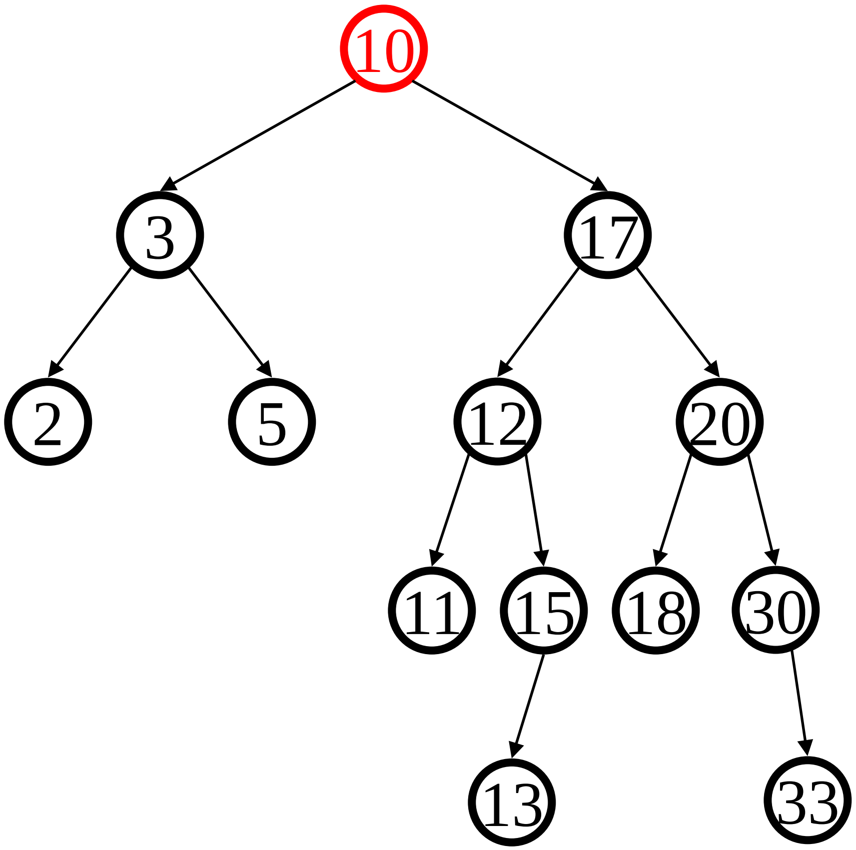
Delete(9)



Double Rotation on Deletion

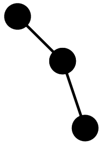


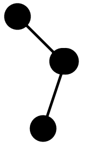
Deletion with Propagation



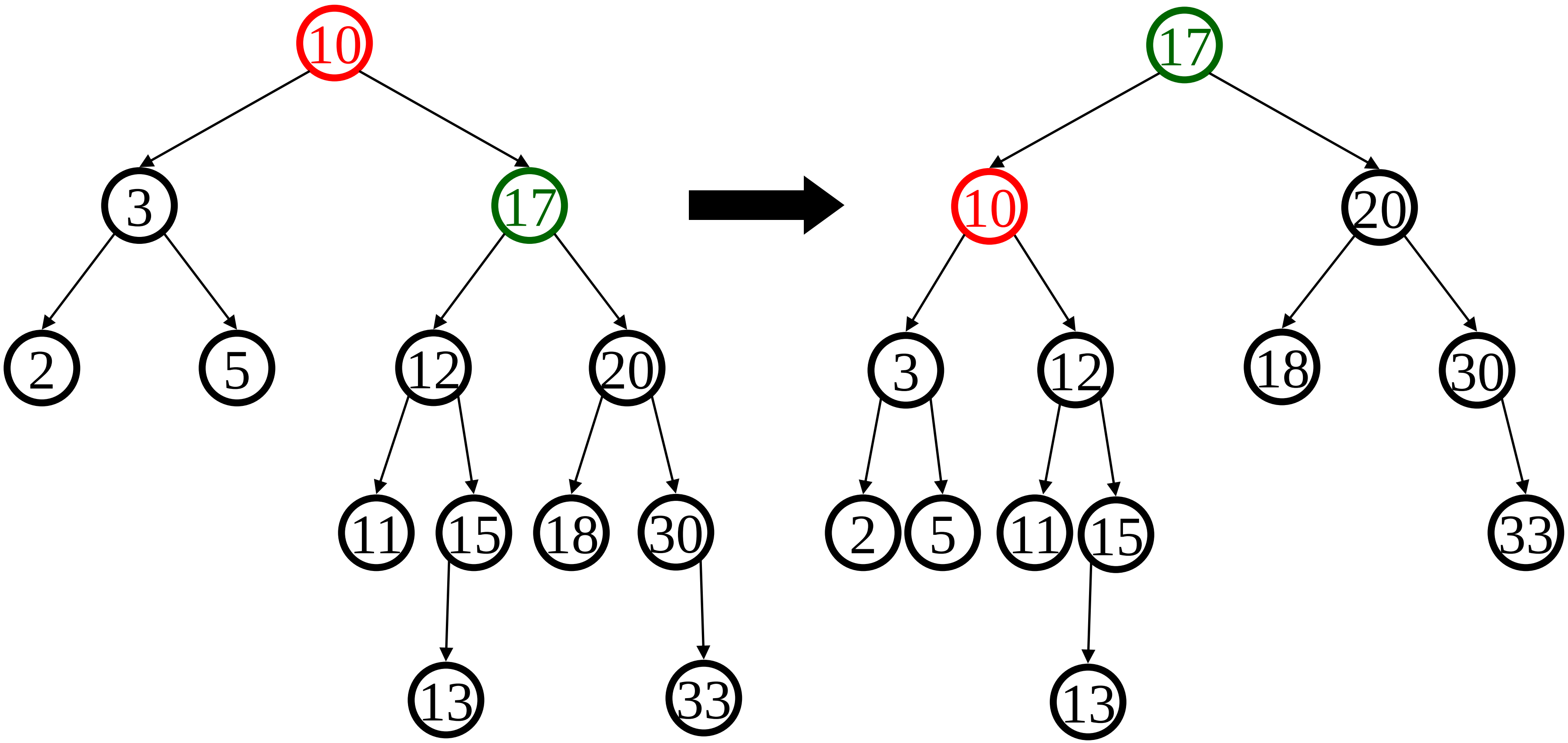
What different about this case?

We get to choose whether to single or double rotate!





Propagated Single Rotation



AVL Tree Deletion

- Similar but more complex than insertion
  - Rotations and double rotations needed to rebalance
  - Imbalance may propagate upward so that many rotations may be needed.

Pros and Cons of AVL Trees

Advantages of AVL trees:

1. Search is O(log N) since AVL trees are always balanced
2. Insertion and deletions are also O(logn)
3. The height balancing adds no more than a constant factor to the speed of insertion

Problems in AVL trees:

1. Difficult to program & debug; more space for balance factor
2. Asymptotically faster but rebalancing costs time
3. Most large searches are done in database systems on disk and use other structures (e.g. B-trees)