VNUHCM - University of Science

fit@hcmus

CSC10004 – Data Structures and Algorithms

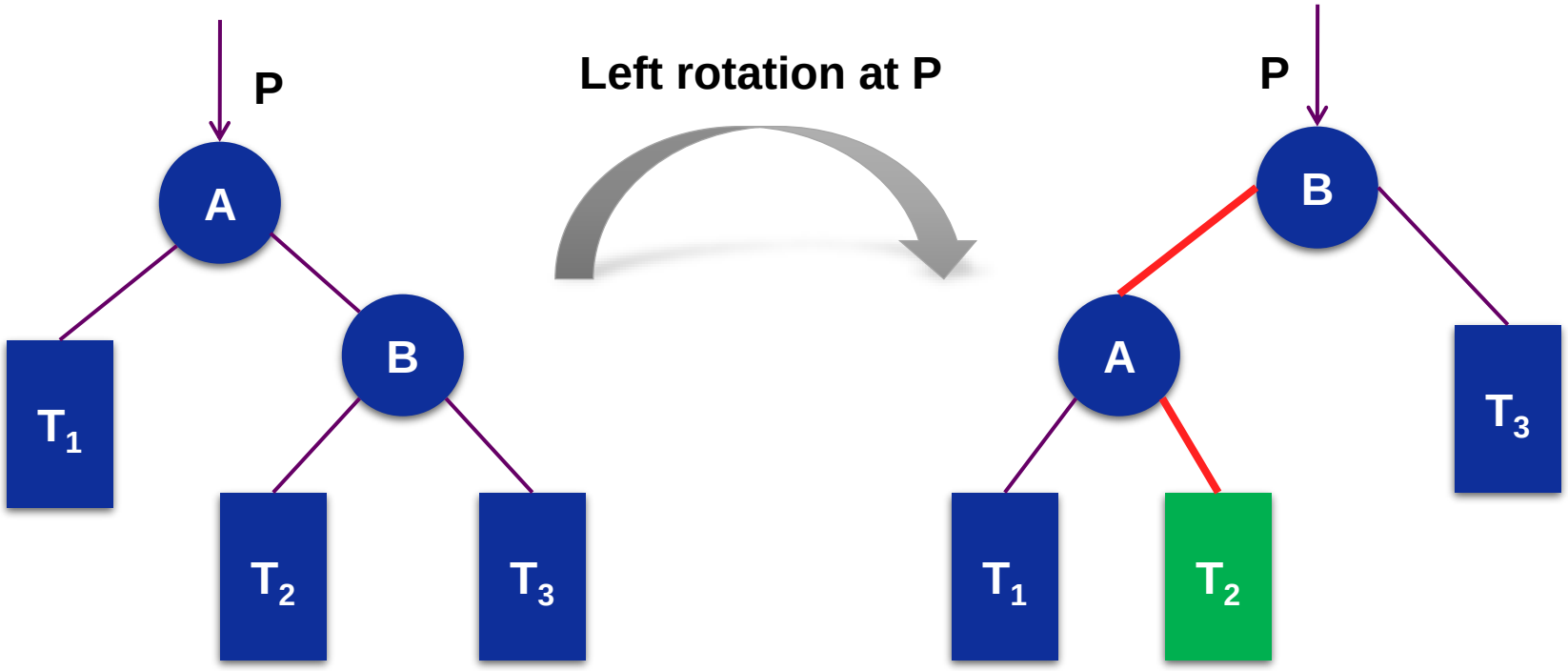
Session 06
Tree Structure

Instructors:
Dr. Lê Thanh Tùng

Content

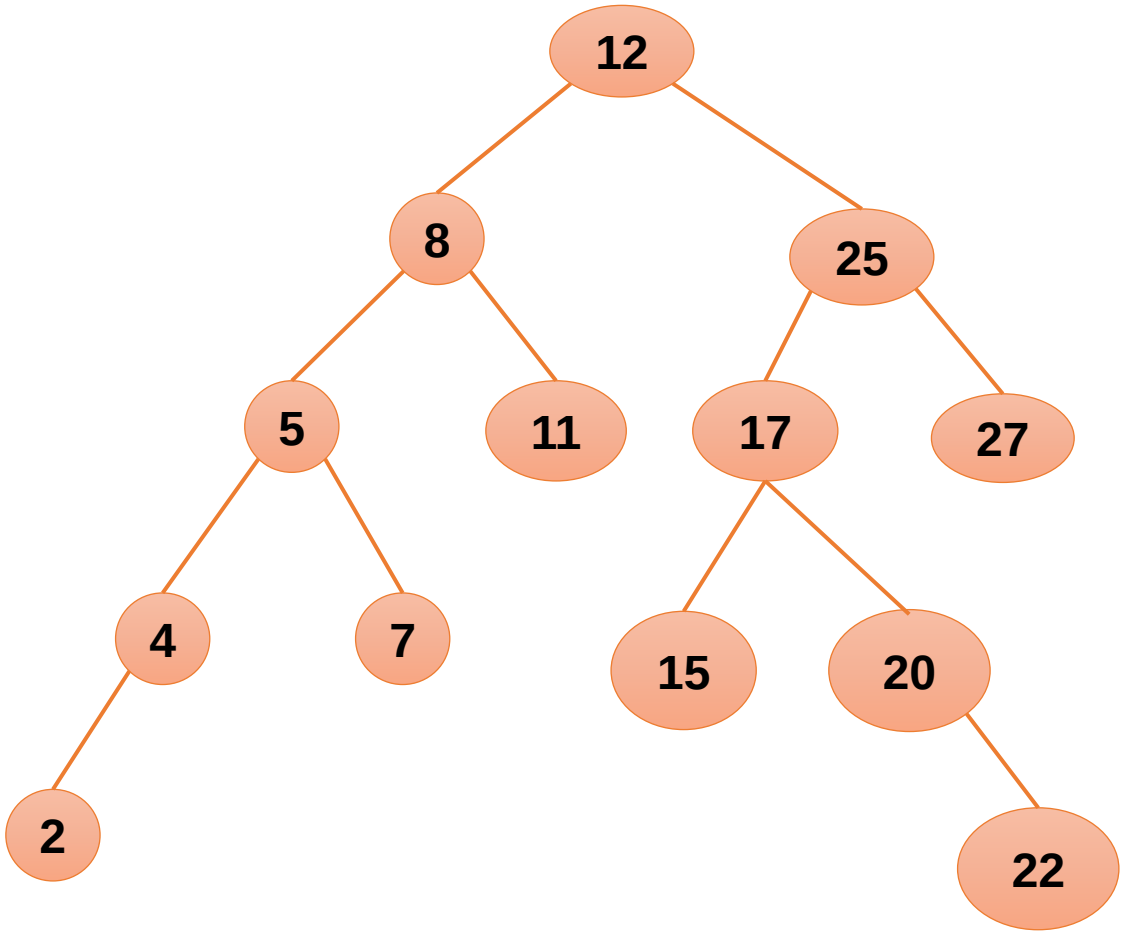
Tree Rotation

Left Rotation



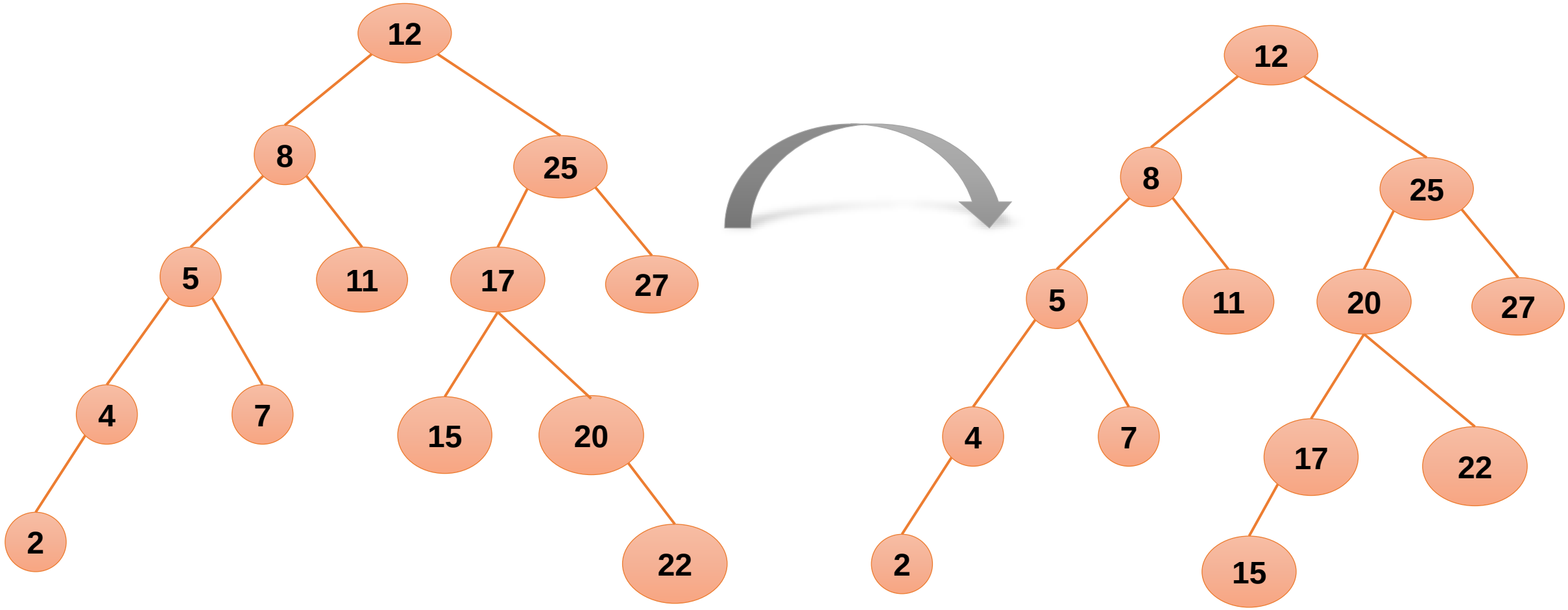
Left Rotation

- Left Rotate the following tree at Node 17

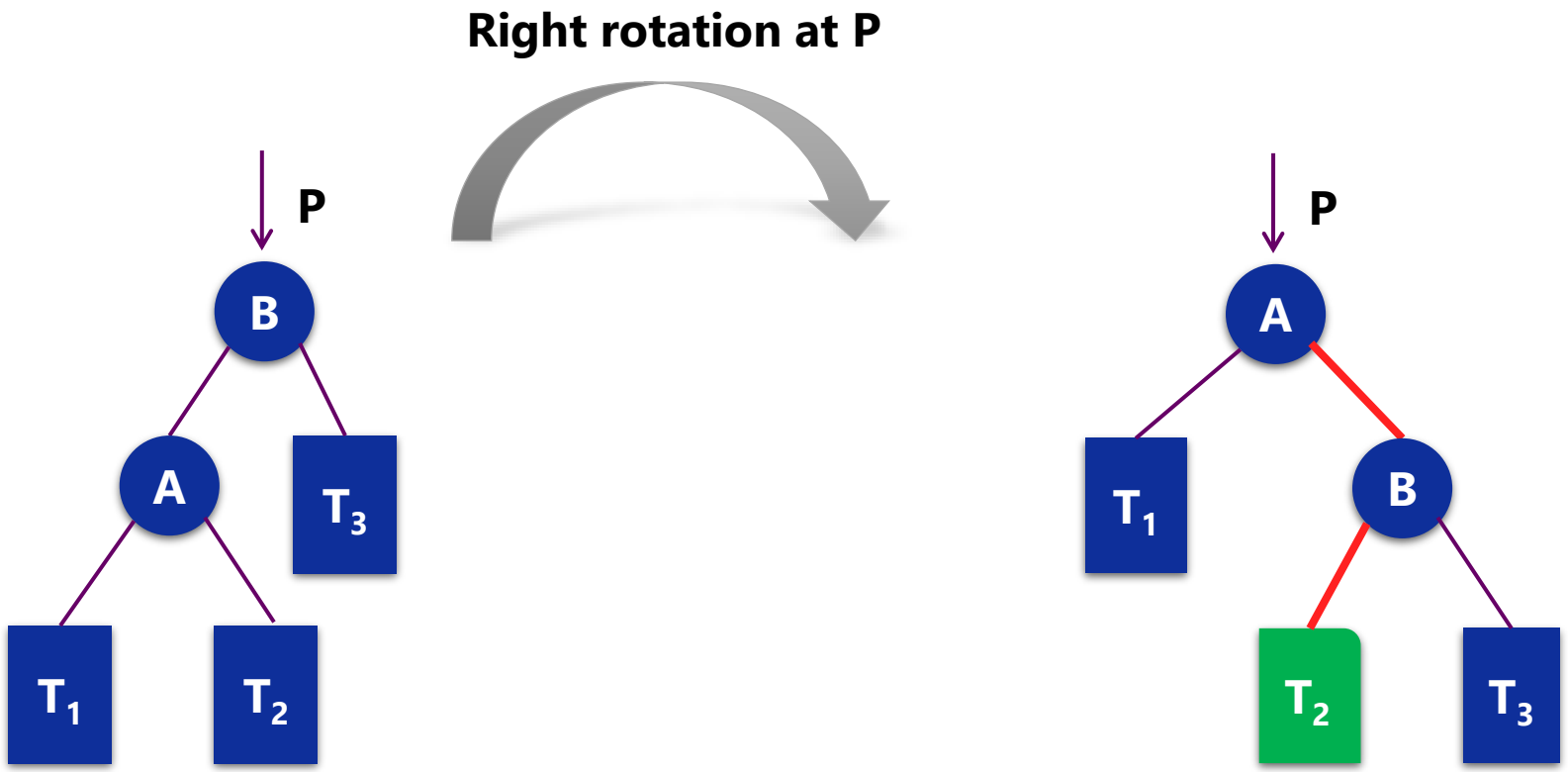


Left Rotation

- Left Rotate the following tree at Node 17

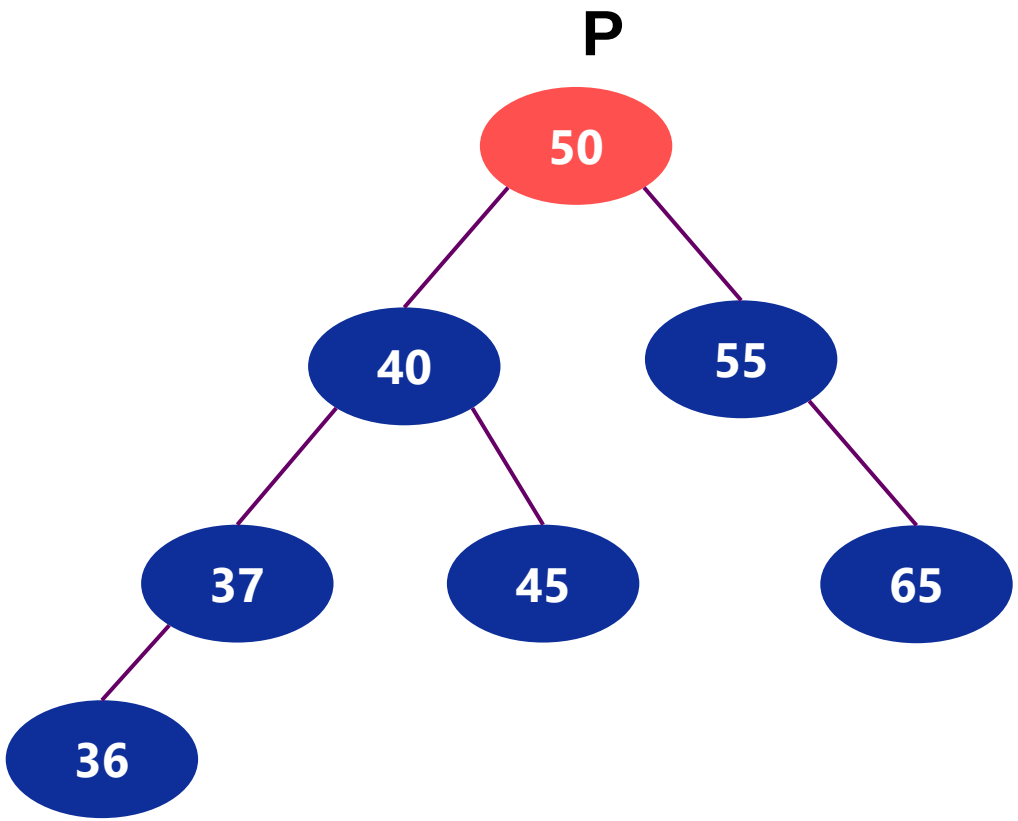


Right Rotation



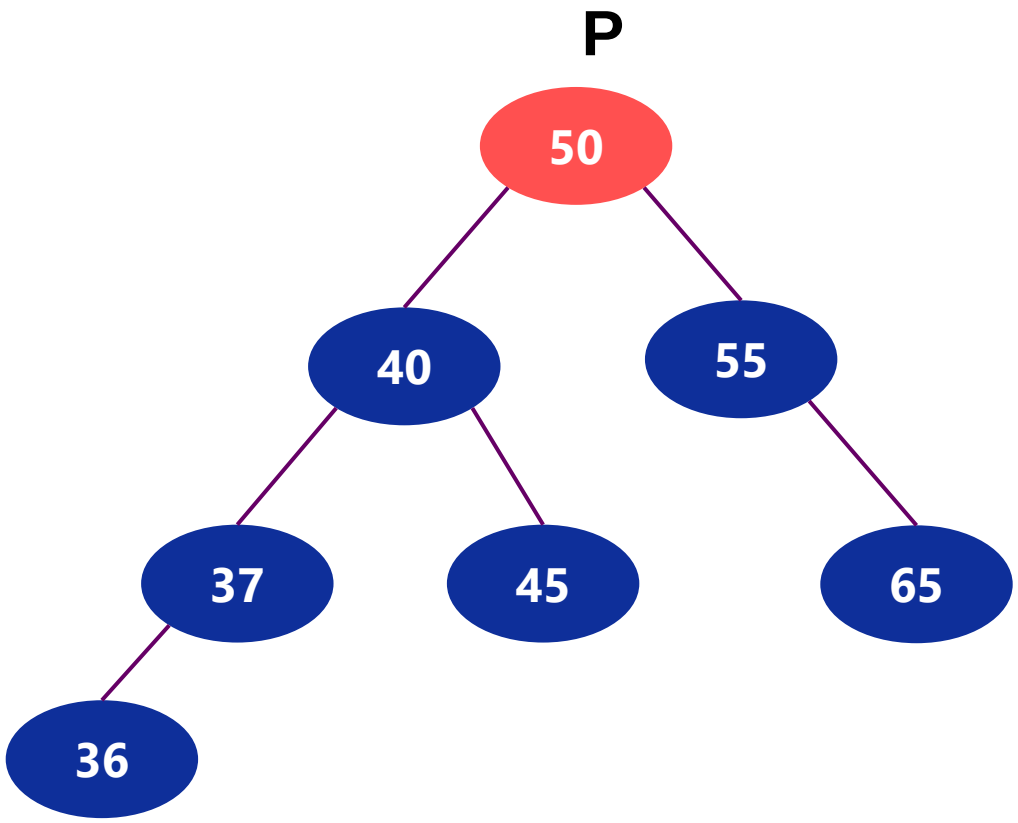
Right Rotation

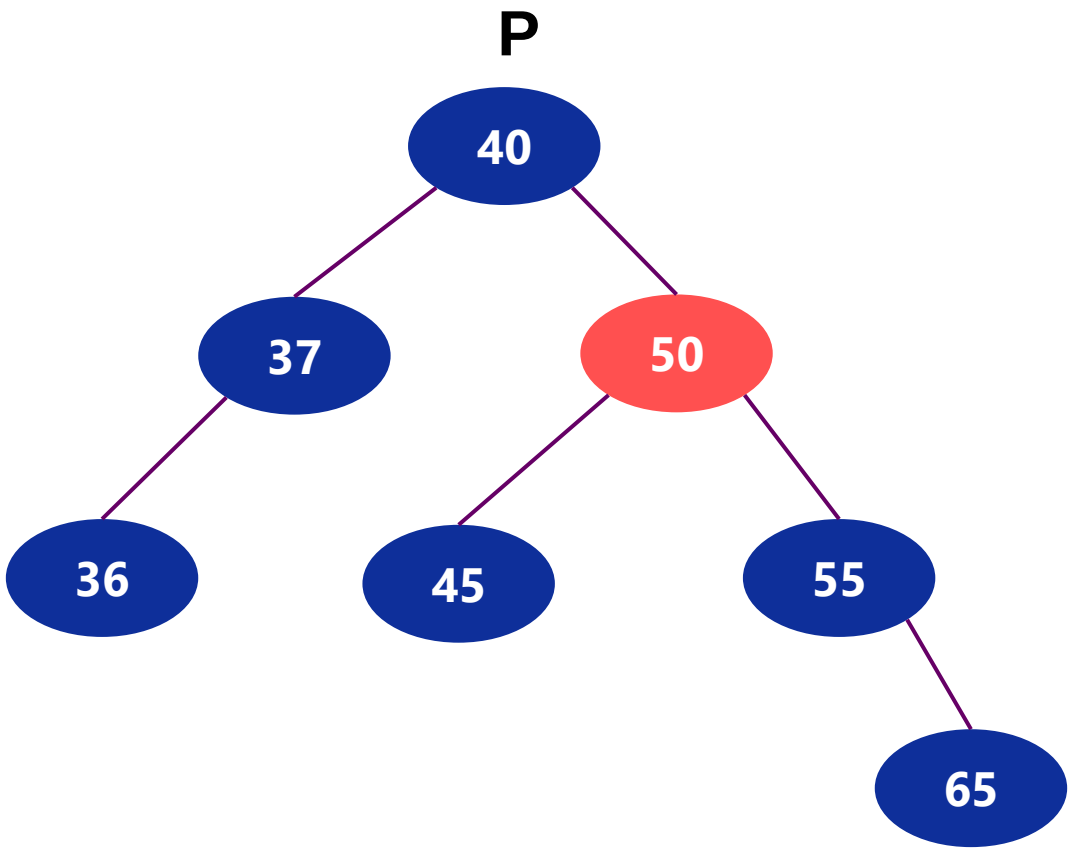
- Right Rotate the following tree at Node P



Right Rotation

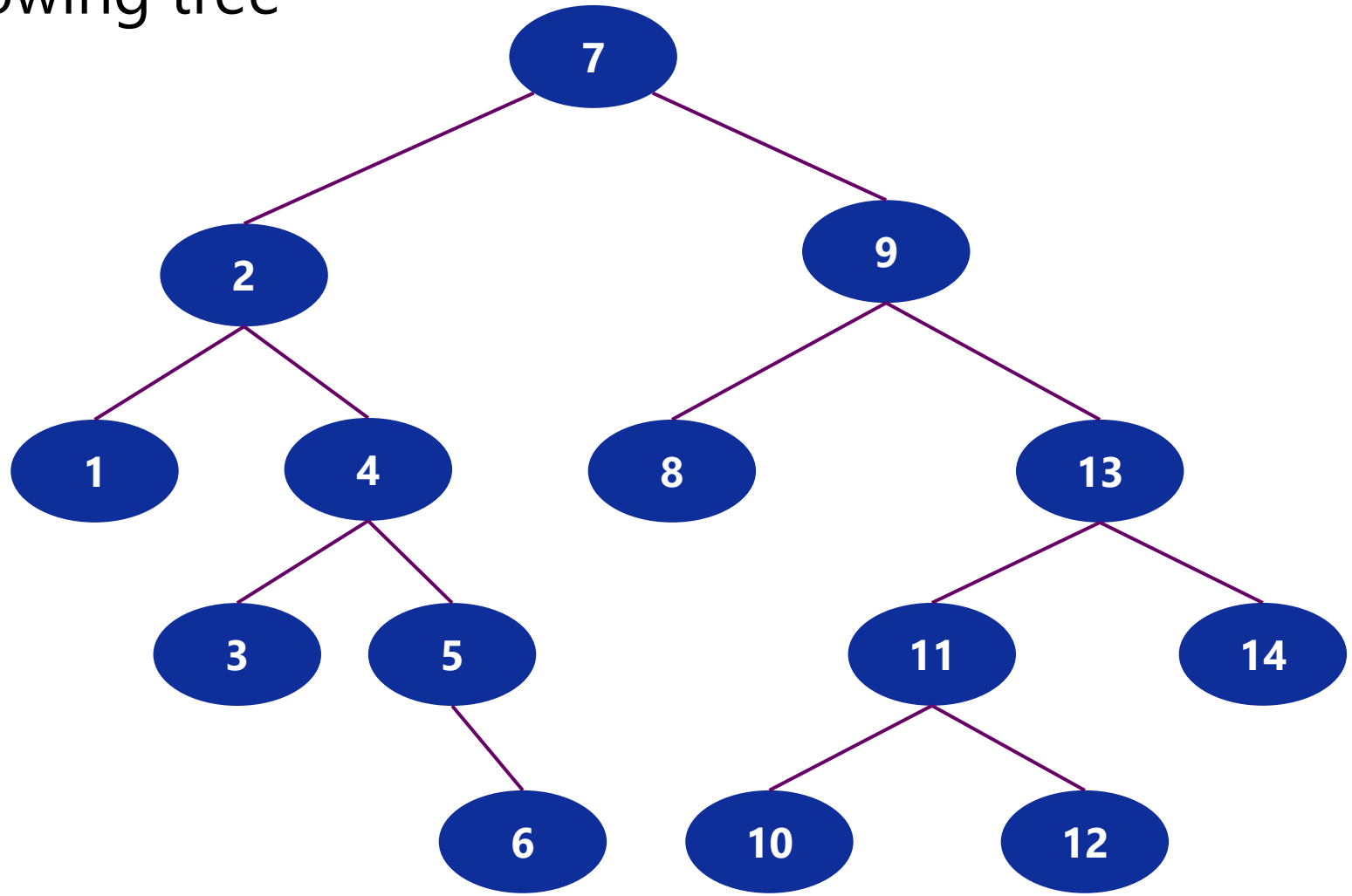
- Right Rotate the following tree at Node P





Exercise

- Give the following tree



AVL Tree

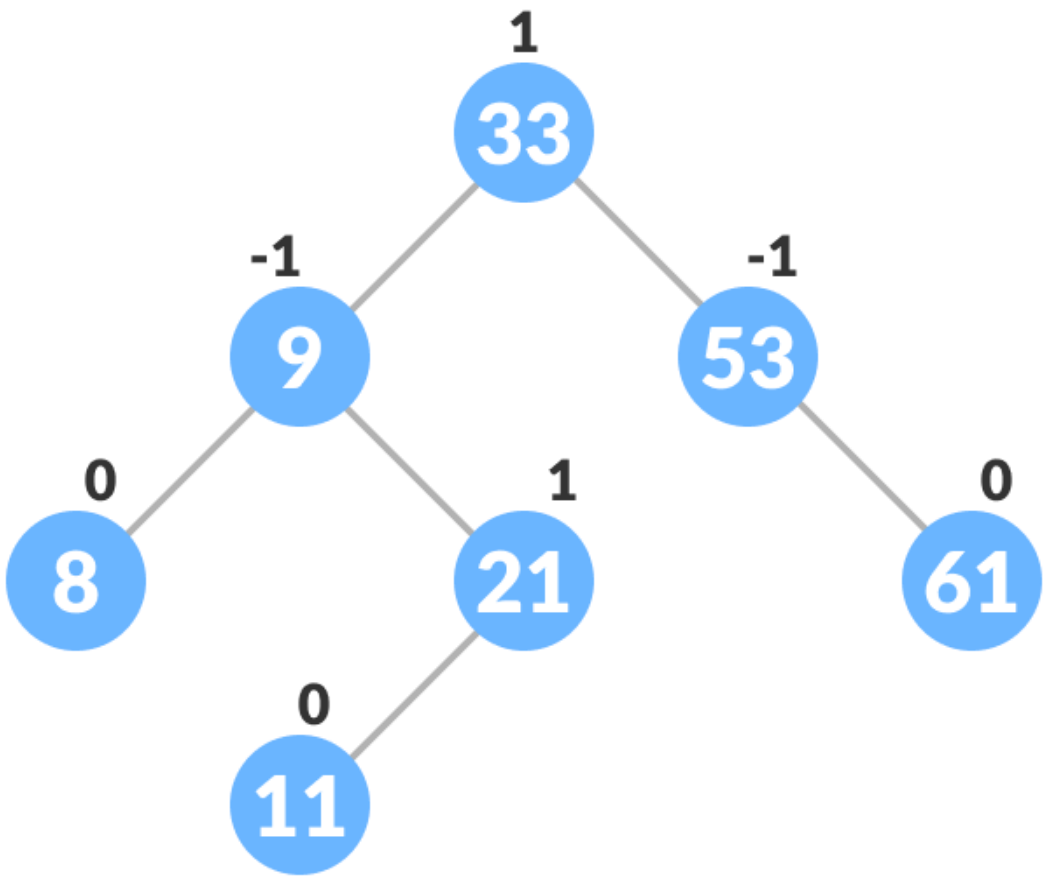
AVL Tree

- Named for inventors, (Georgii) Adelson-Velsky and (Evgenii) Landis
- Invented in 1962 (paper "*An algorithm for organization of information*").
- AVL Tree is a self-balancing binary search tree where
  - for ALL nodes, the difference between height of the left subtrees and the right subtrees **cannot be more than one**. (height invariant, or balance invariant).

AVL Tree

- Balance Factor = (Height of Left Subtree - Height of Right Subtree)

  or = (Height of Right Subtree - Height of Left Subtree)



AVL Tree

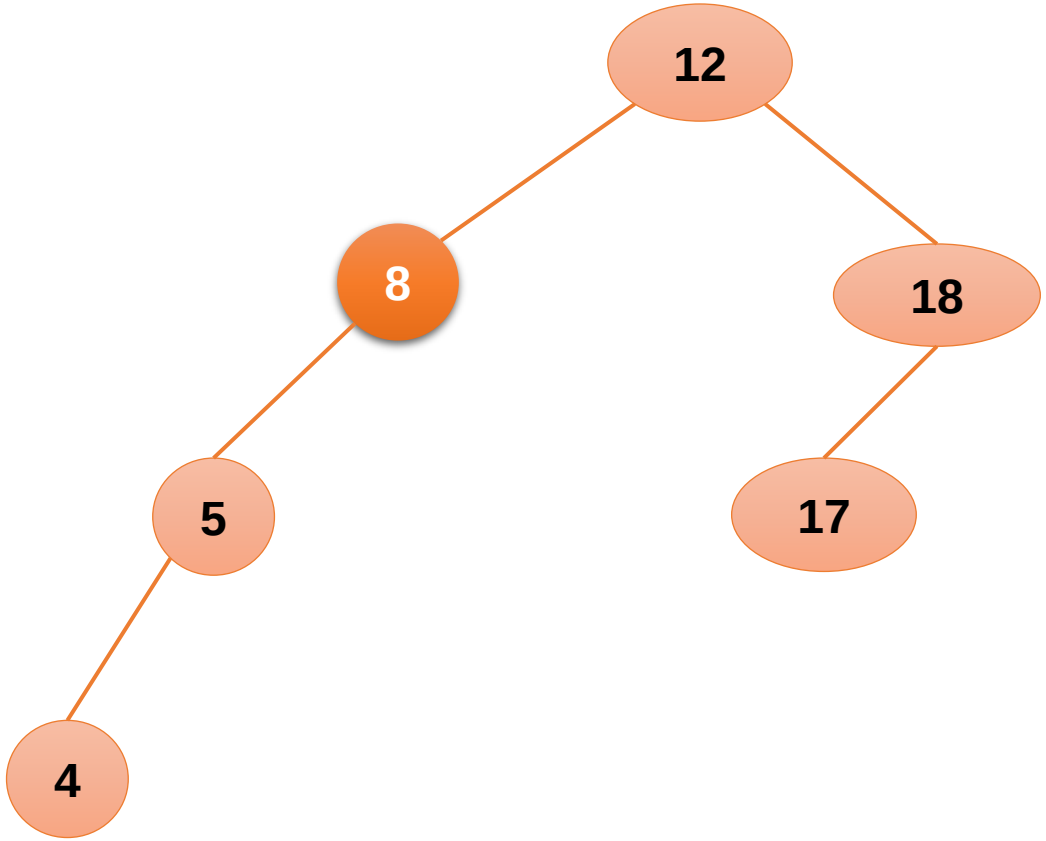
- A balanced binary search tree
  - Maintains height close to the minimum
  - After insertion or deletion, check the tree is still AVL tree – determine whether any node in tree has left and right subtrees whose heights differ by more than 1
- Can search AVL tree almost as efficiently as minimum-height binary search tree.

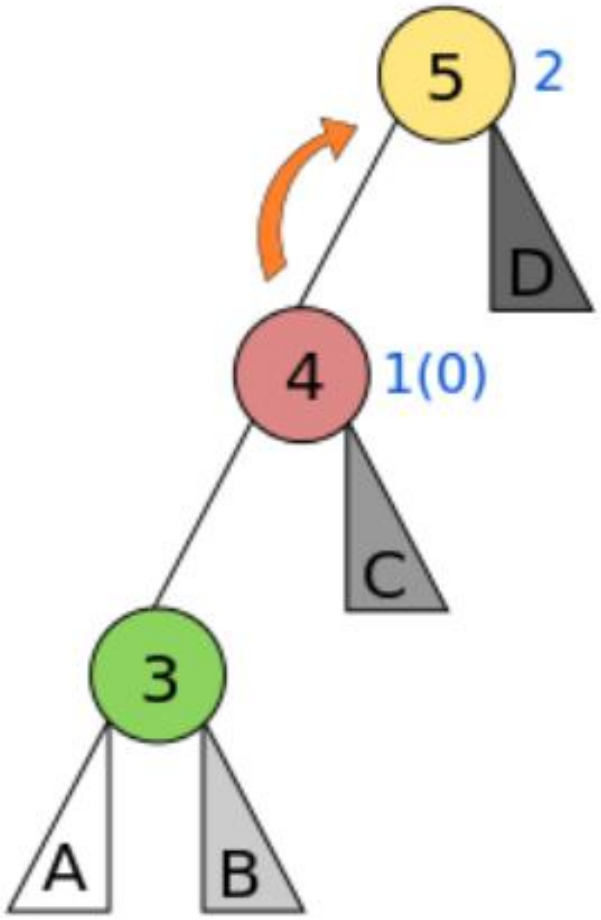
Cases of Height Invariant Violation

- Left-Left case
- Left-Right case
- Right-Right case
- Right-Left case

Cases of Height Invariant Violation

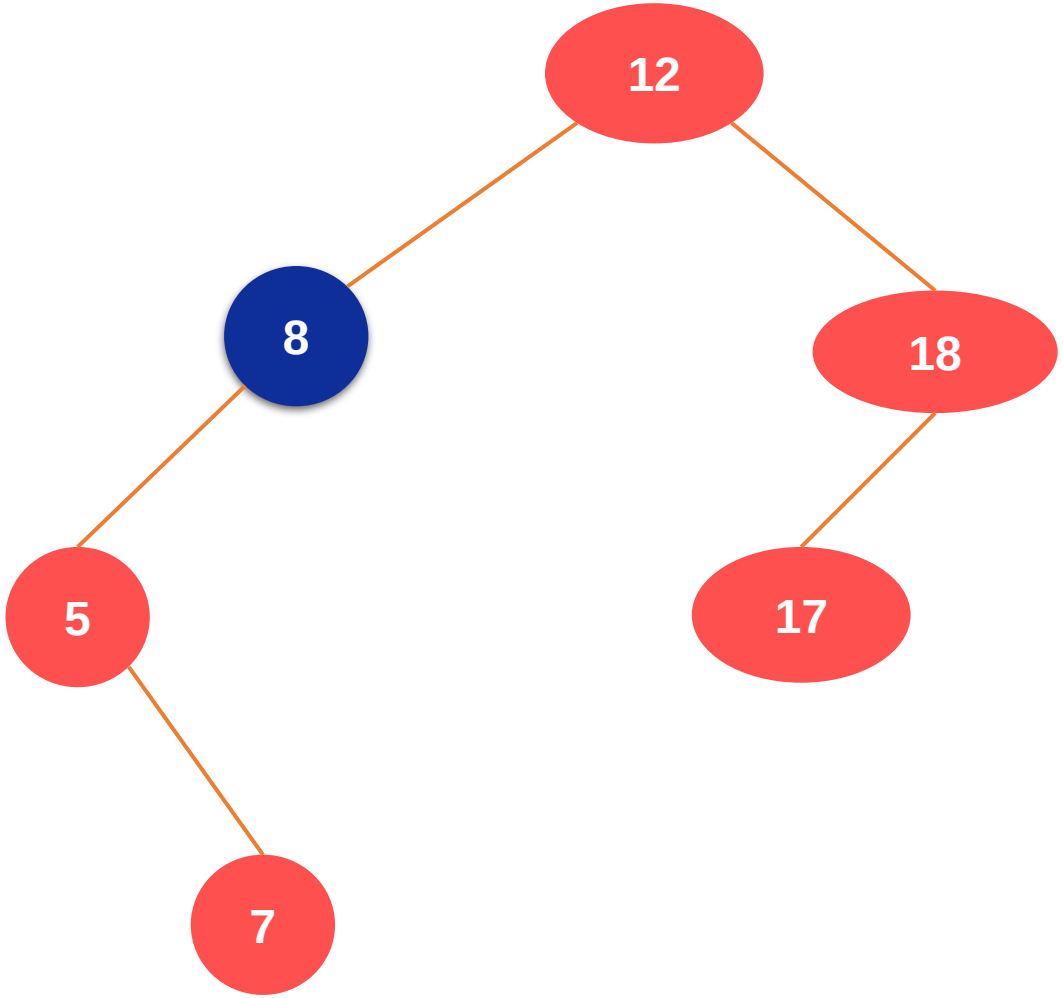
- Left-Left case

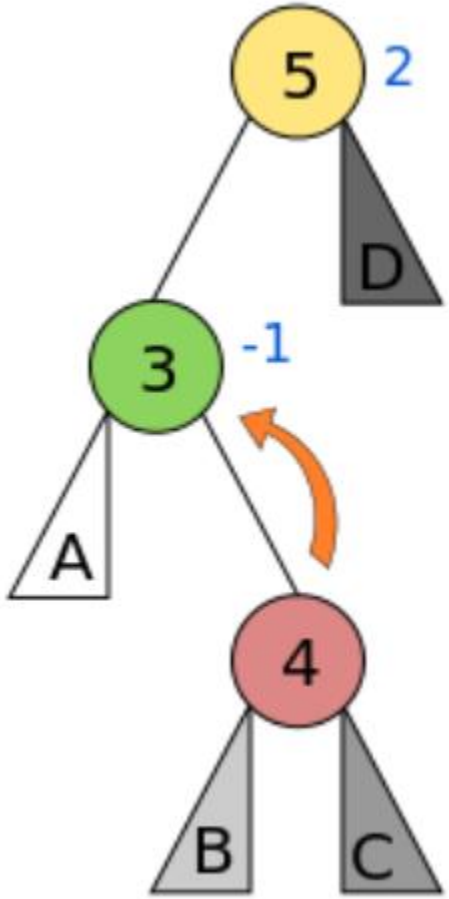




Cases of Height Invariant Violation

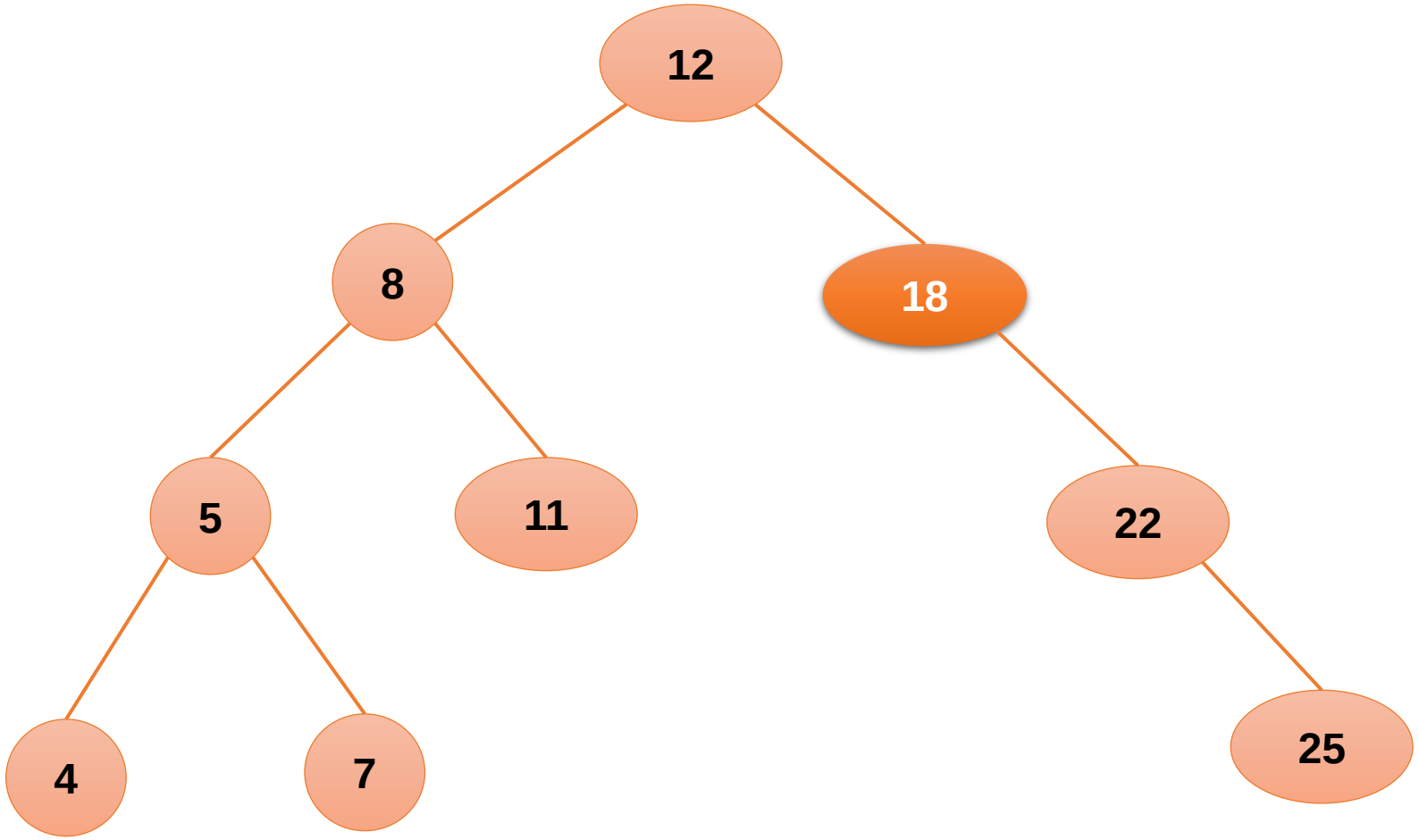
- Left-Right case

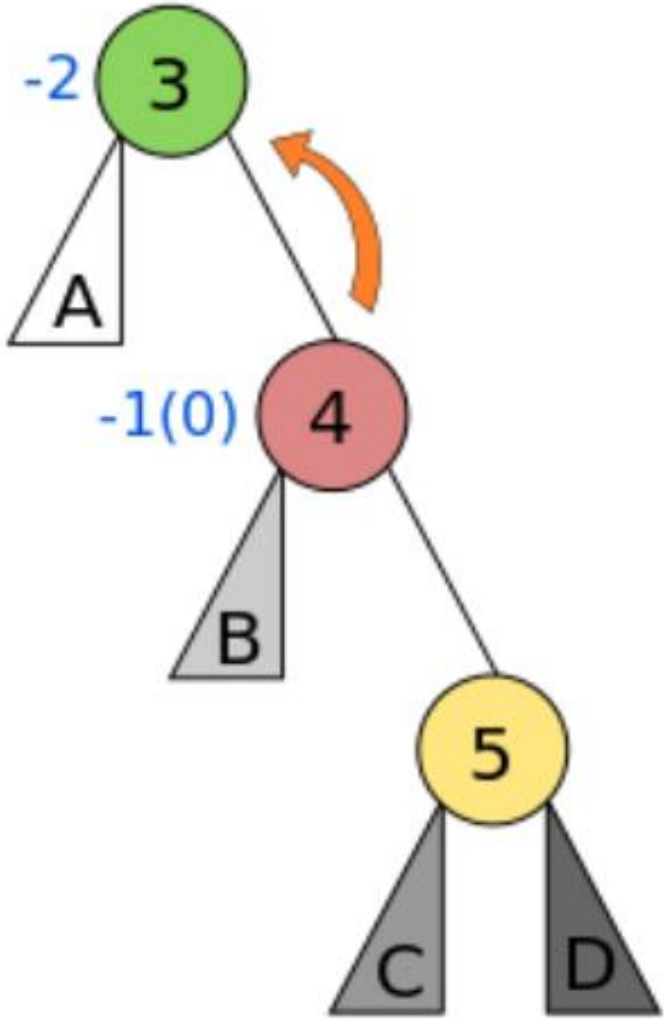




Cases of Height Invariant Violation

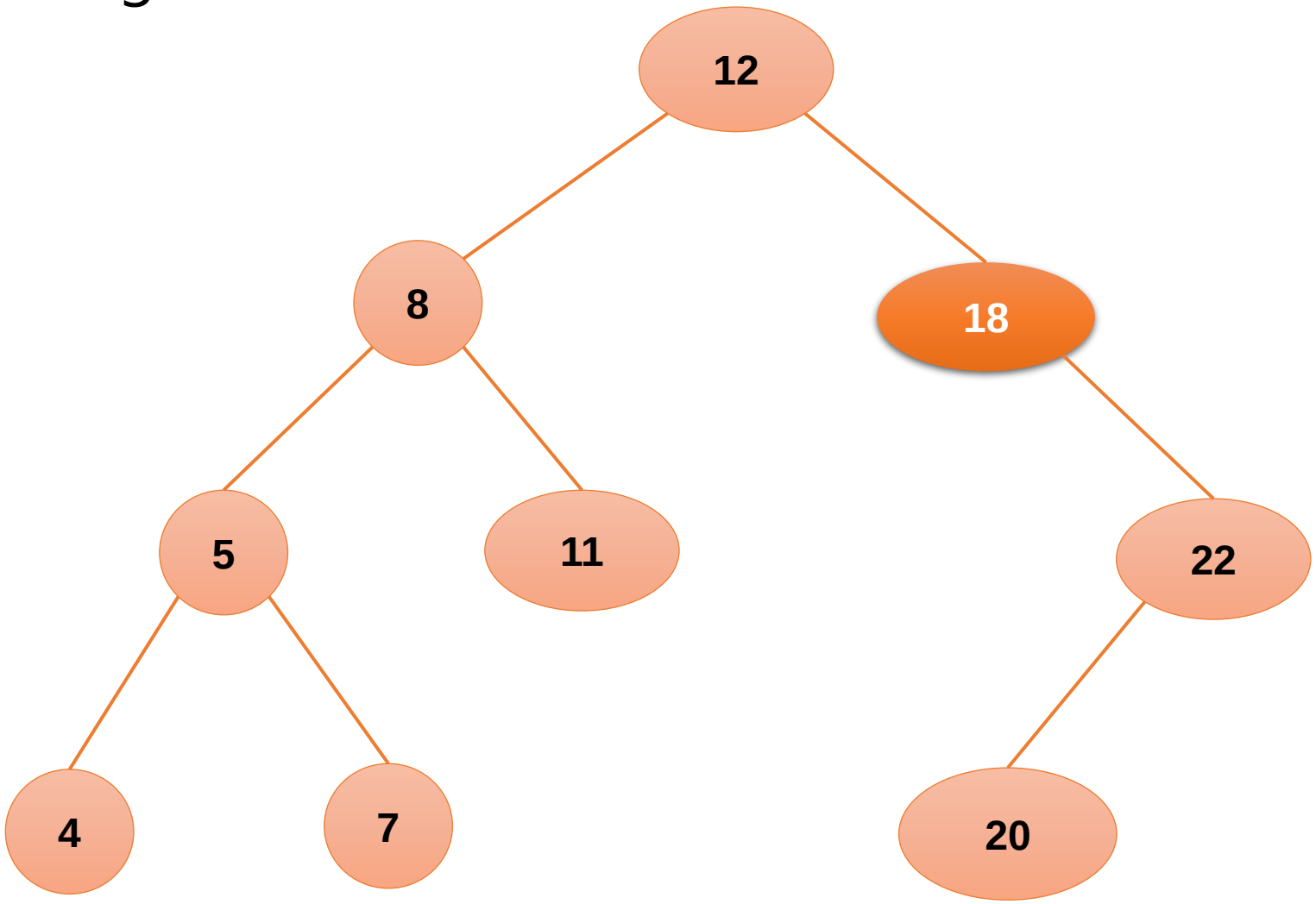
- Right-Right case

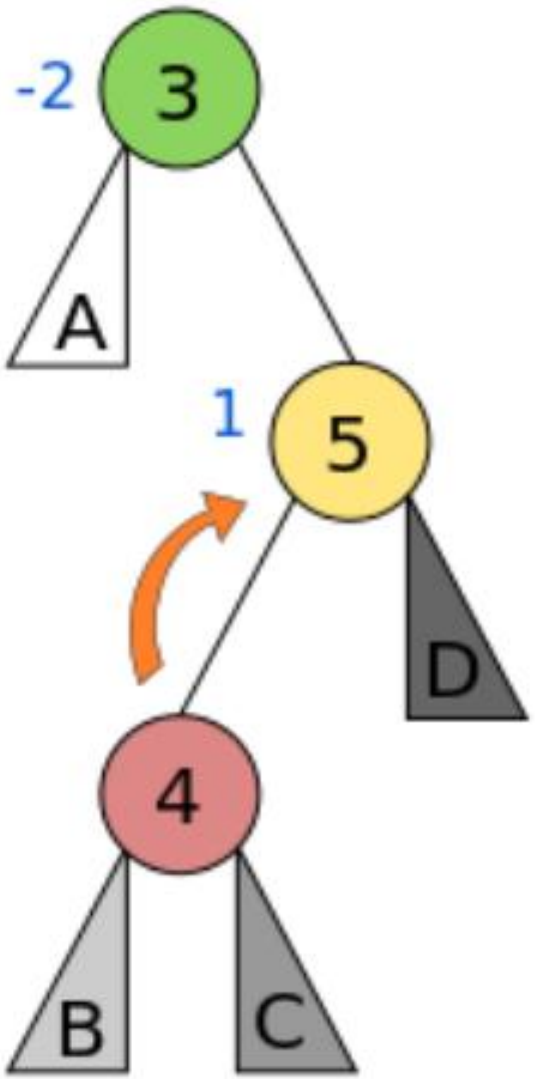




Cases of Height Invariant Violation

- Right-Left case





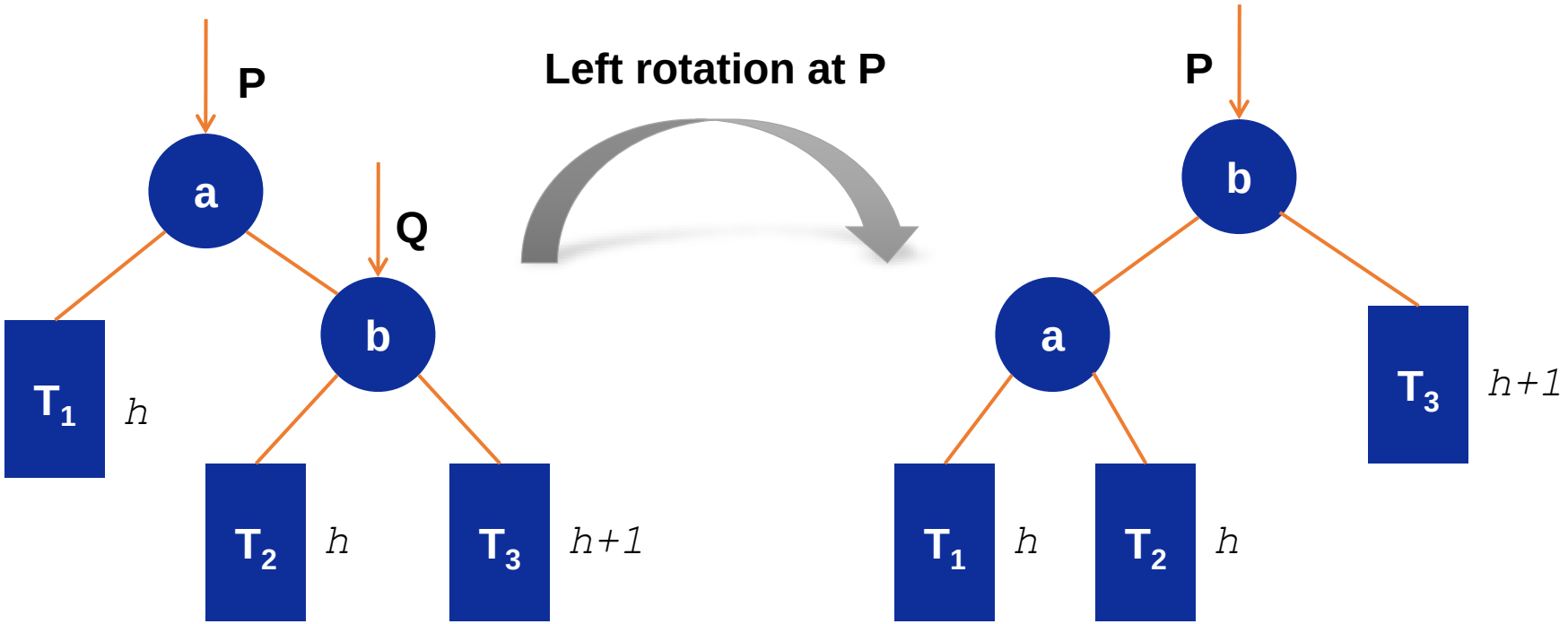
Violation Resolving

- **Right-Right case:**
  - Left rotation at un-balanced node.

- **Right-Left case:**
  - Right rotation at un-balanced node's right child
  - Left rotation at un-balanced node.

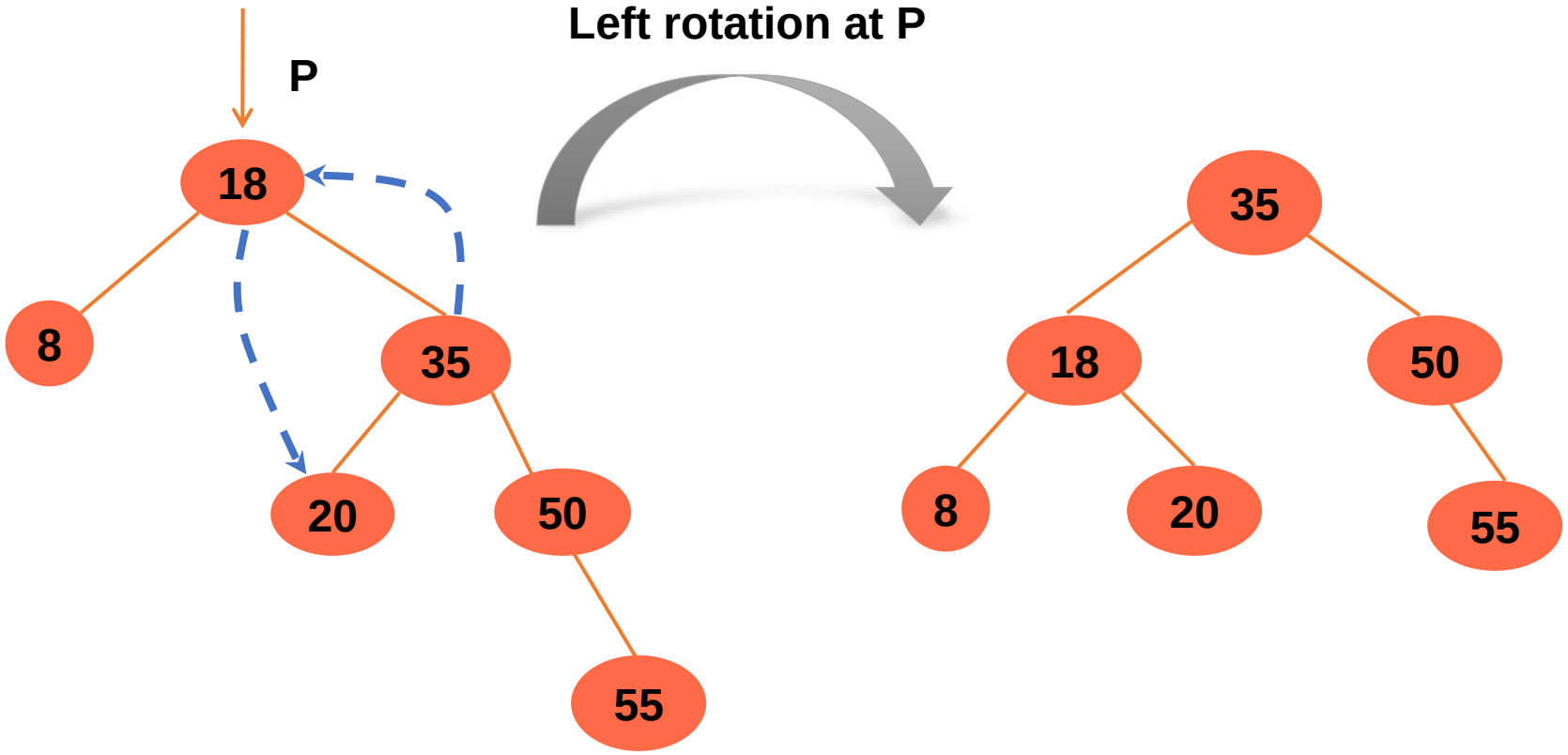
Violation Resolving

- **Right-Right case:**
  - Unbalance at P



Violation Resolving

- **Right-Right case:** example

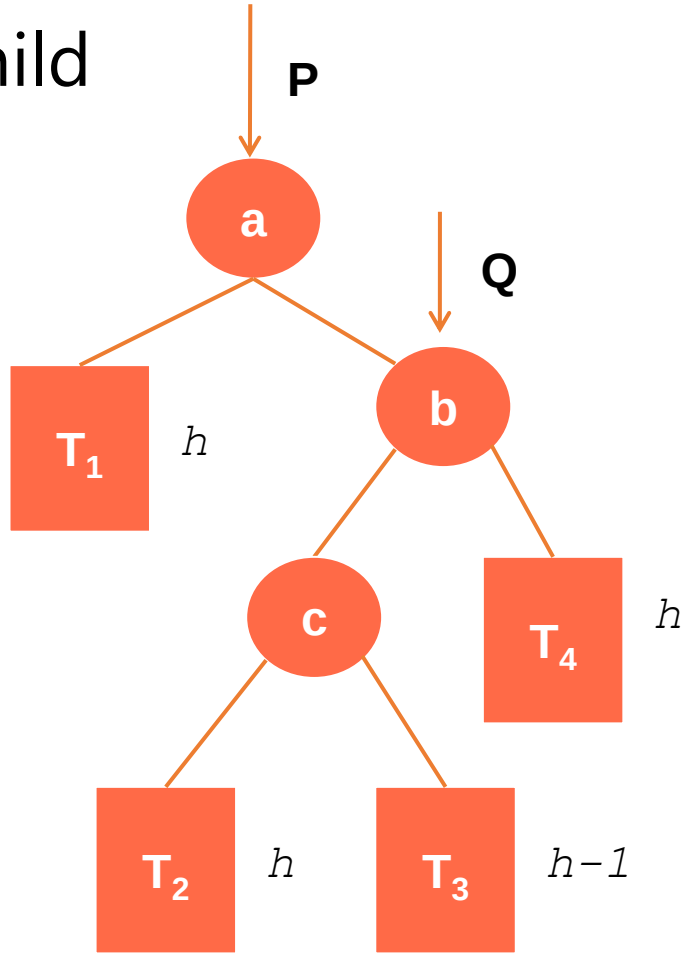


Violation Resolving

- **Right-Left case:**
  - Right rotation at un-balanced node's right child
  - Left rotation at un-balanced node

- In the following tree,

what is the unbalanced node ?



Violation Resolving

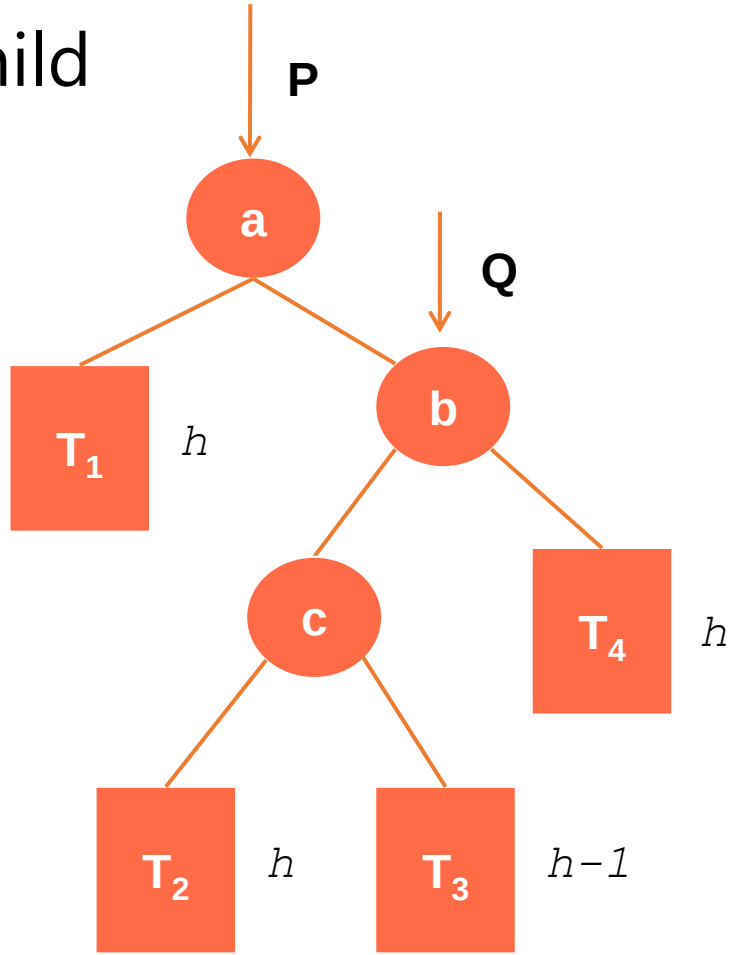
- **Right-Left case:**
  - Right rotation at un-balanced node's right child
  - Left rotation at un-balanced node

- In the following tree,

what is the unbalanced node ?

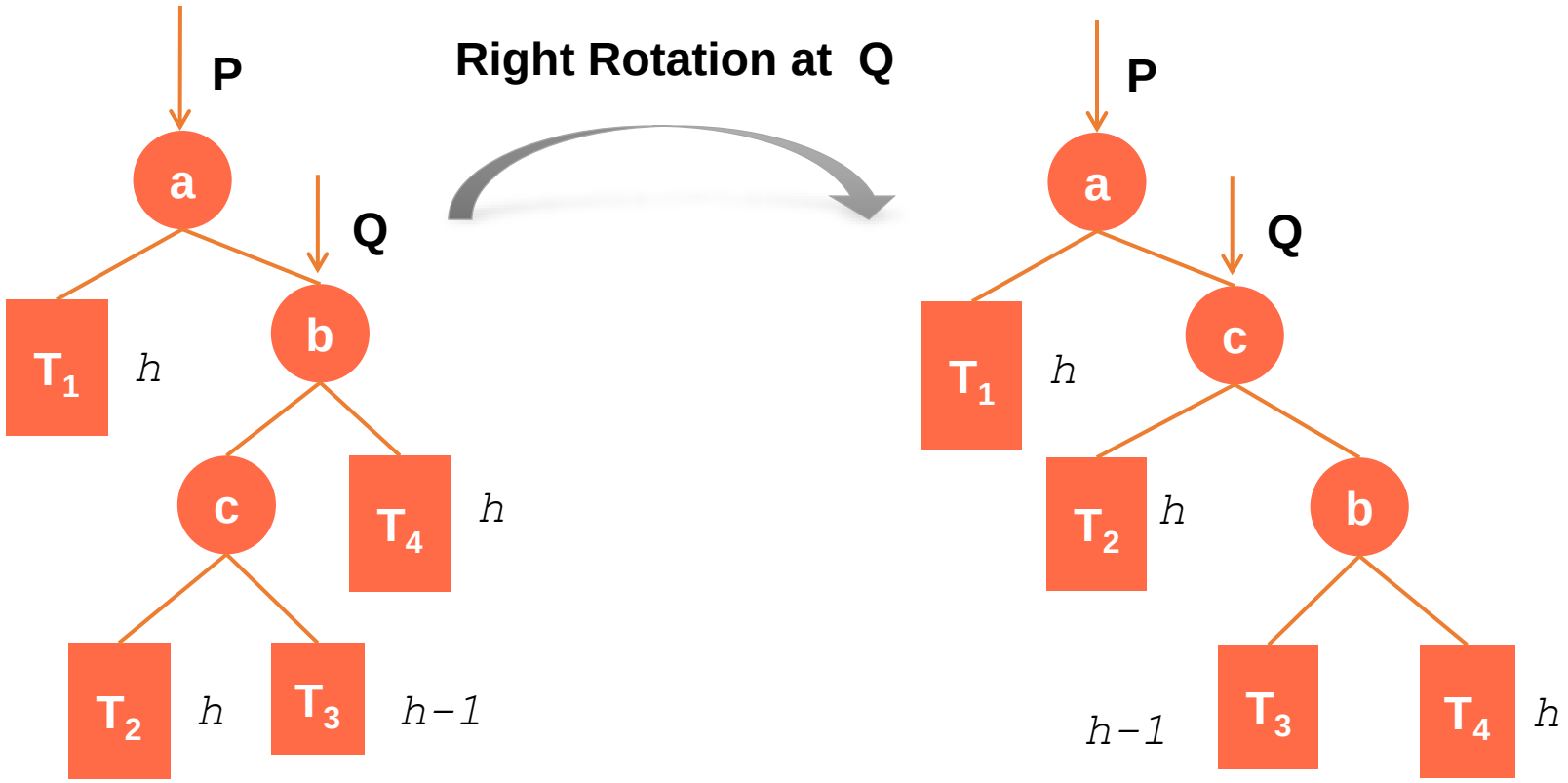
Resolve:

  - Right rotation at Q
  - Left rotation at P



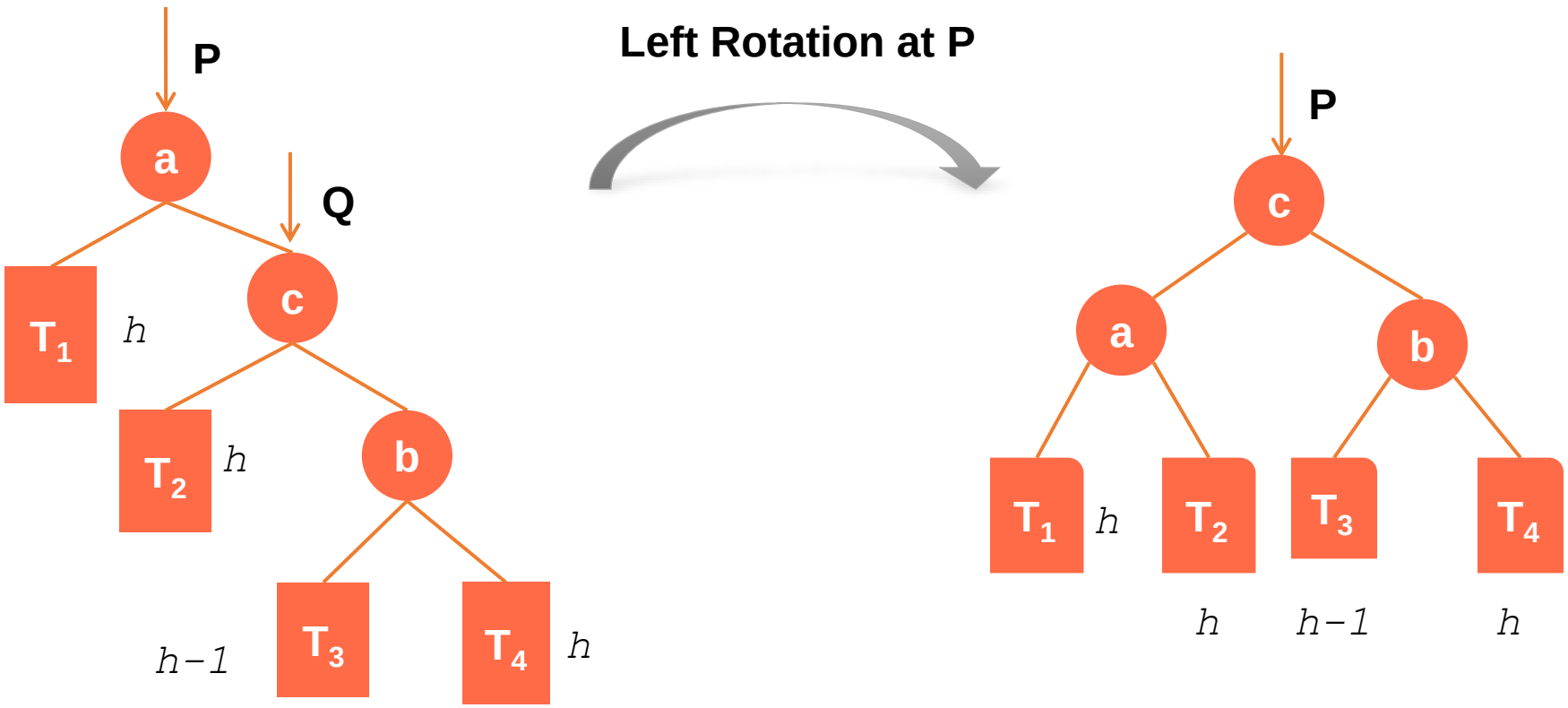
Violation Resolving

- **Right-Left case:**
  - Right rotation at Q



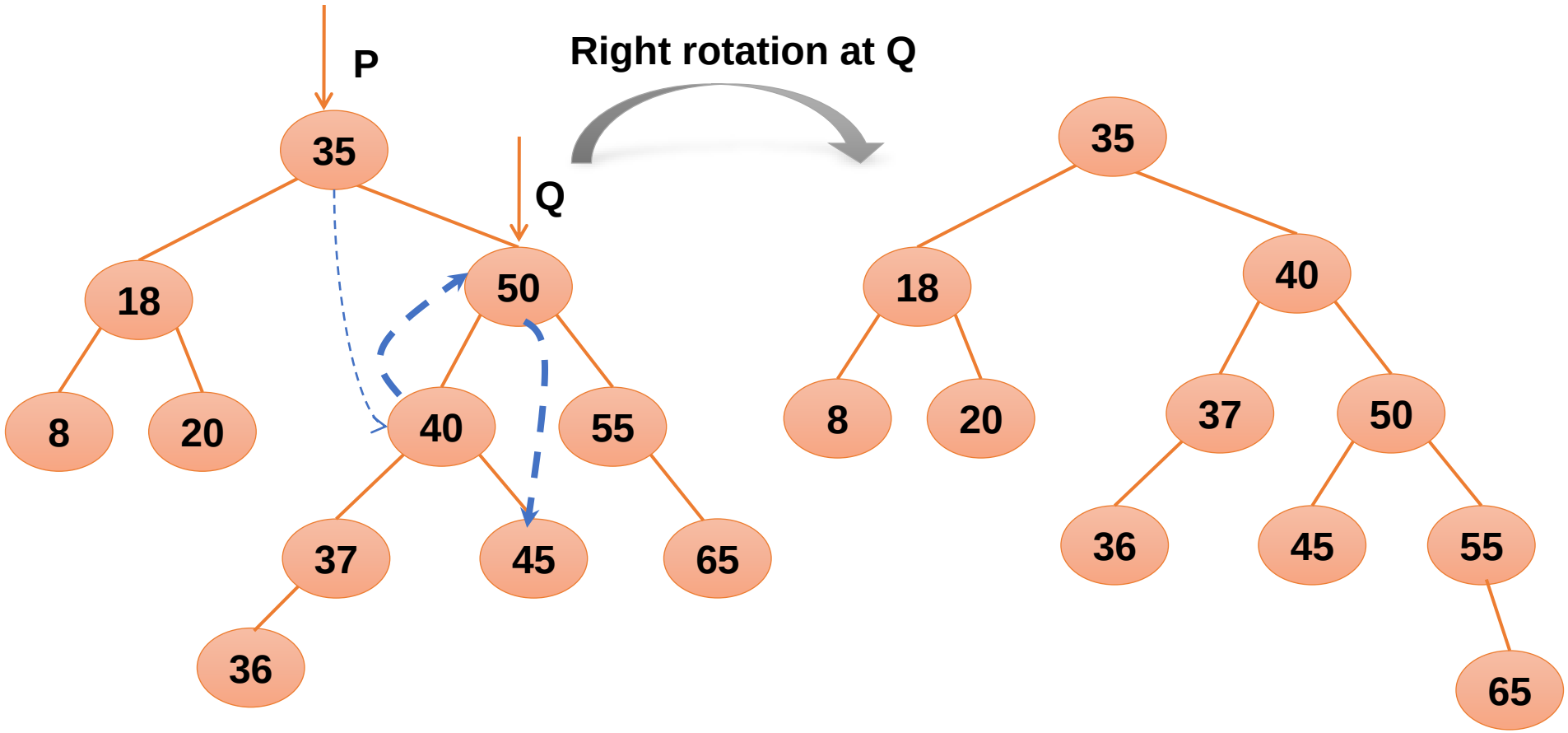
Violation Resolving

- **Right-Left case:**
  - Left rotation at P



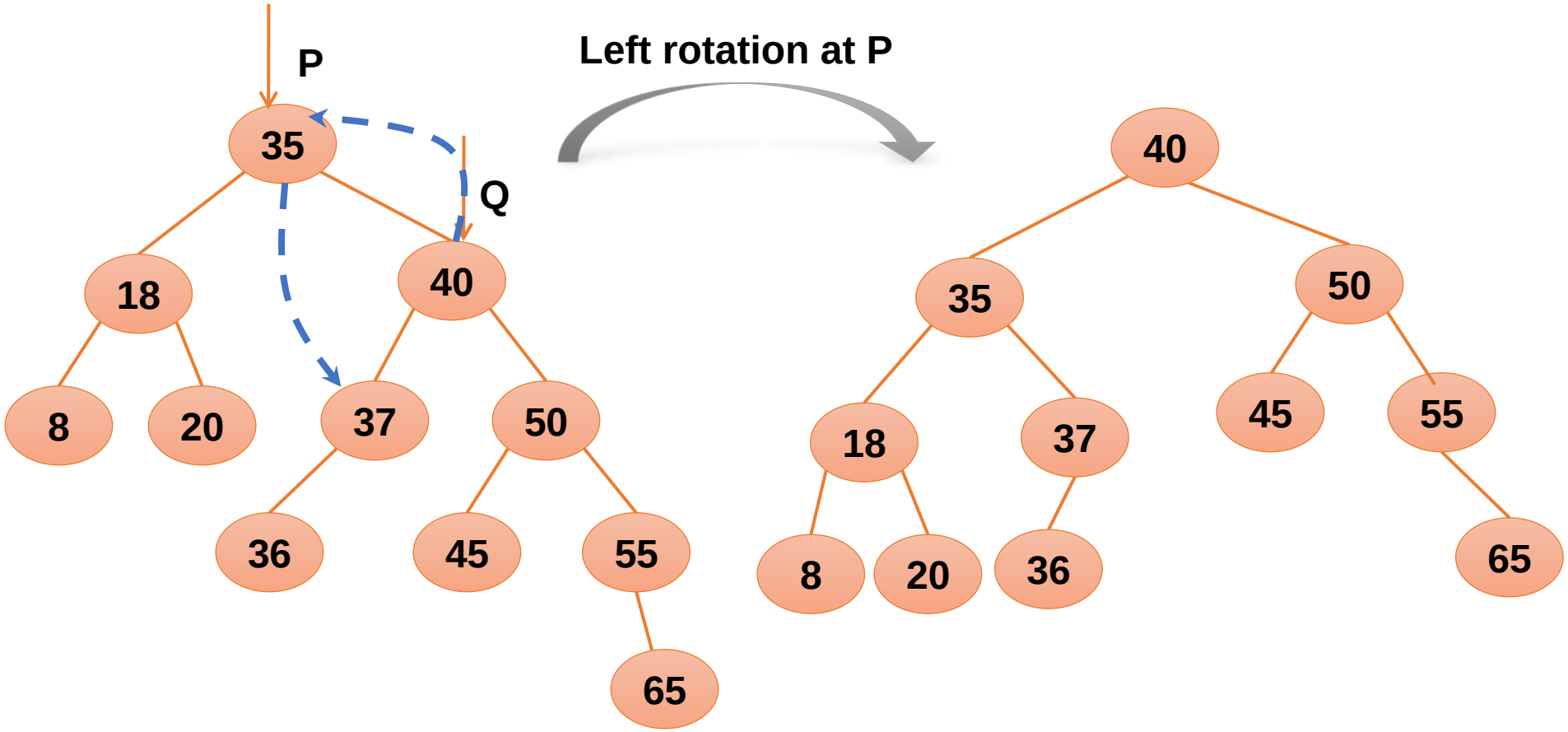
Violation Resolving

- **Right-Left case: example**



Violation Resolving

- **Right-Left case: example**



Violation Resolving

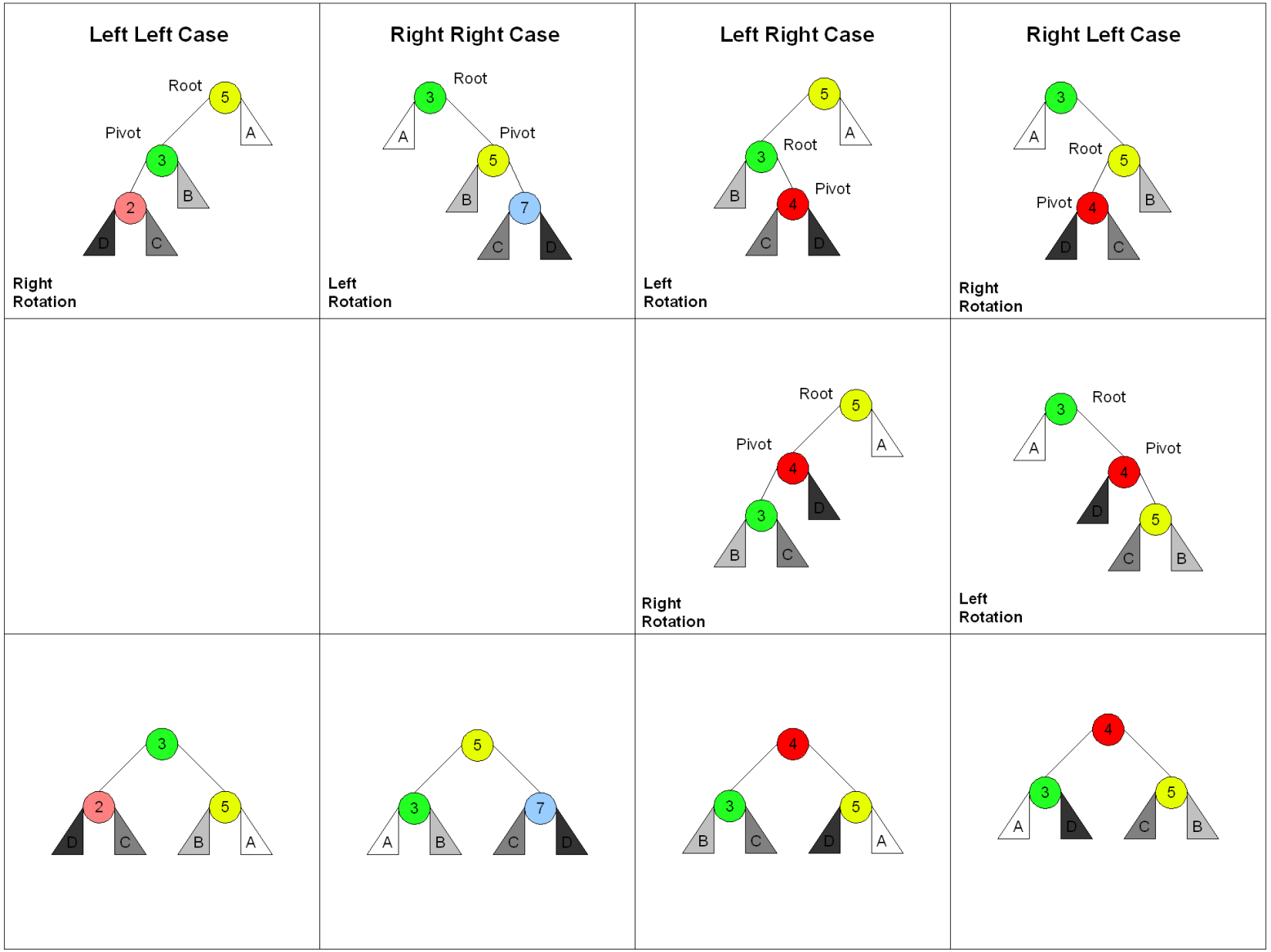
- **Left-Left case:**
  - Right rotation at un-balanced node.

- **Left-Right case:**
  - Left rotation at un-balanced node's left child
  - Right rotation at un-balanced node.

Violation Resolving

There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

Root is the initial parent before a rotation and **Pivot** is the child to take the root's place.



Insertion

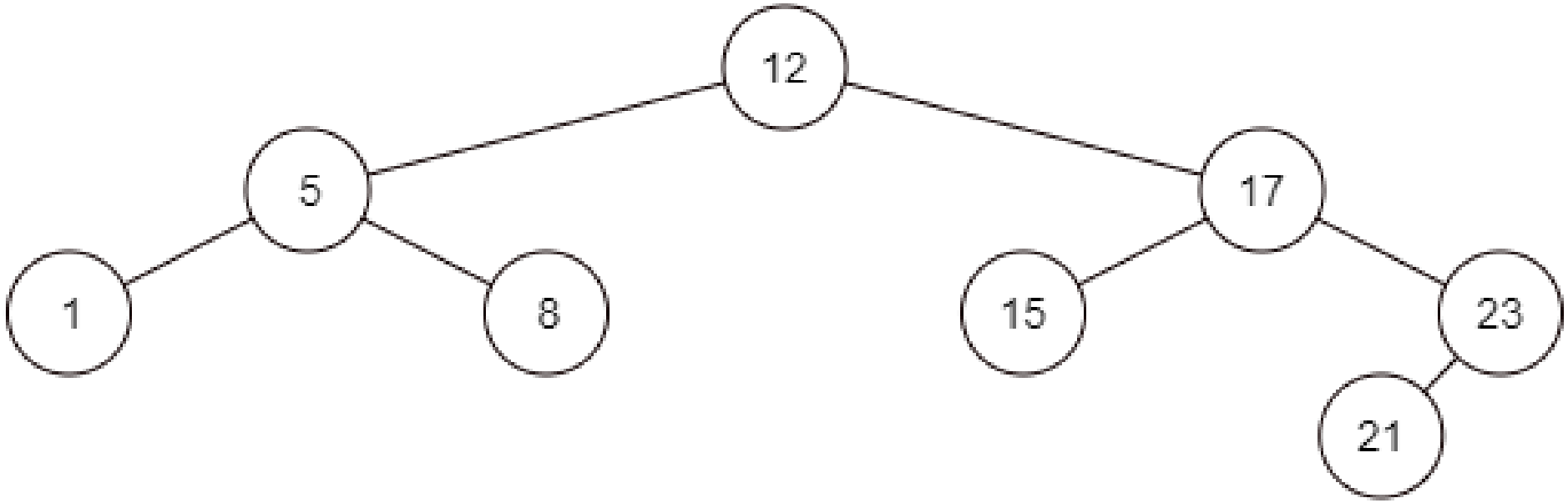
- Beginning with an empty AVL tree, perform step-by-step the insertion of the following values in the order given

15, 5, 12, 8, 23, 1, 17, 21

Insertion

- Beginning with an empty AVL tree, perform step-by-step the insertion of the following values in the order given

15, 5, 12, 8, 23, 1, 17, 21



Insertion

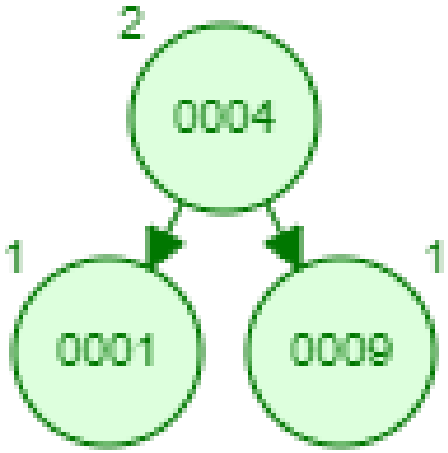
- Beginning with an empty AVL tree, perform step-by-step the insertion of the following values in the order given

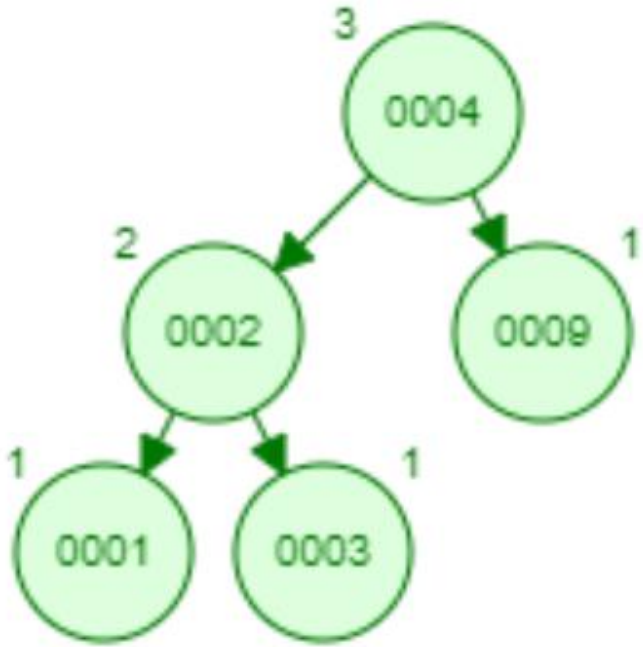
9, 1, 4, 2, 3, 9, 5, 8, 6, 7, 4

Insertion

- Beginning with an empty AVL tree, perform step-by-step the insertion of the following values in the order given

9, 1, 4, 2, 3, 9, 5, 8, 6, 7, 4

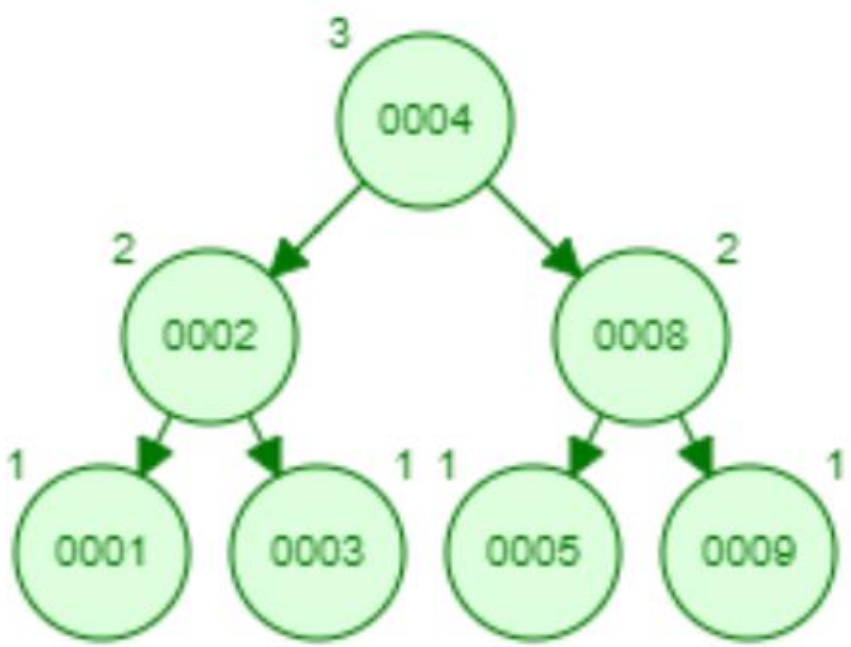


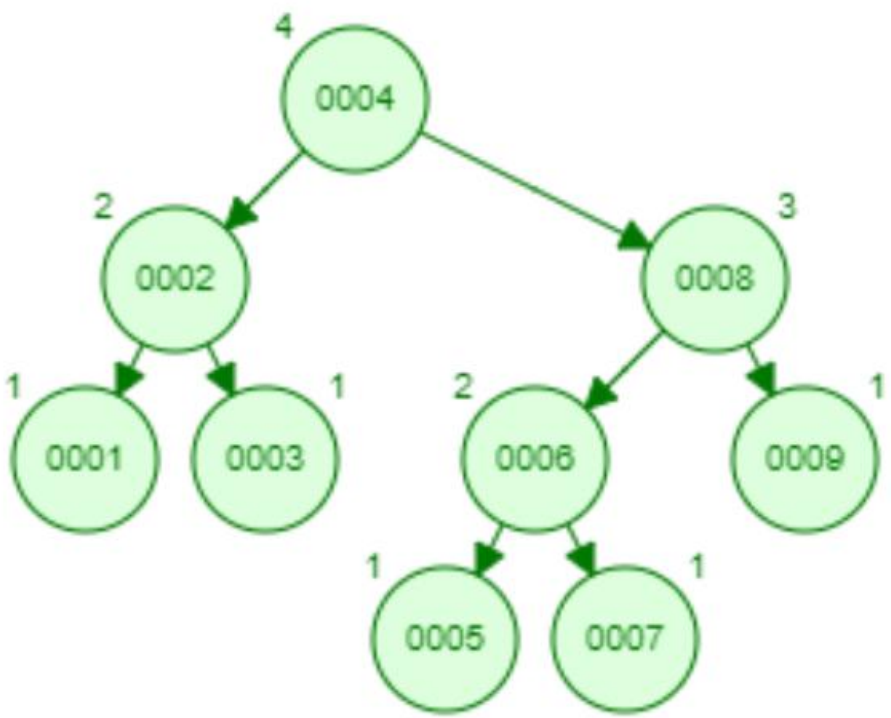


Insertion

- Beginning with an empty AVL tree, perform step-by-step the insertion of the following values in the order given

9, 1, 4, 2, 3, 9, 5, 8, 6, 7, 4

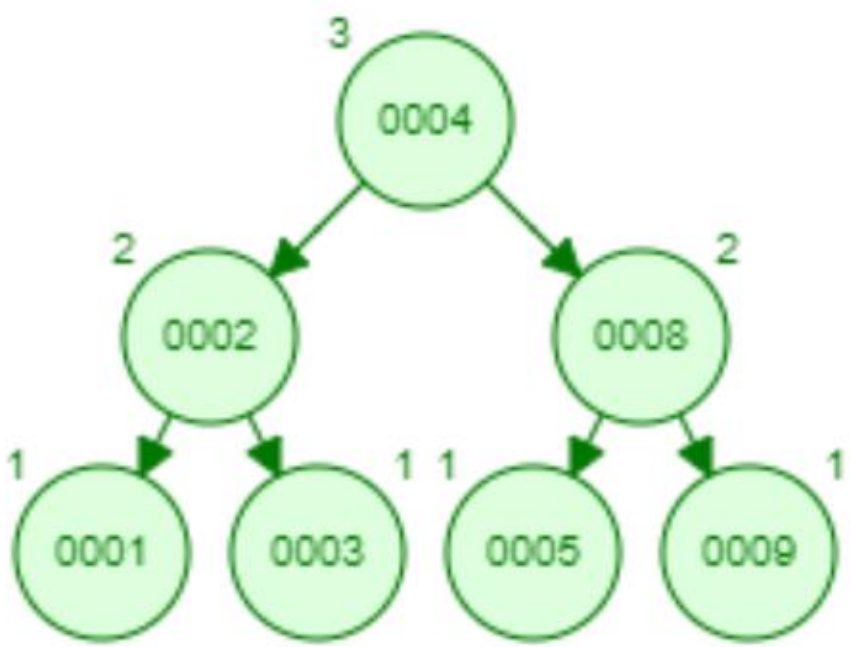


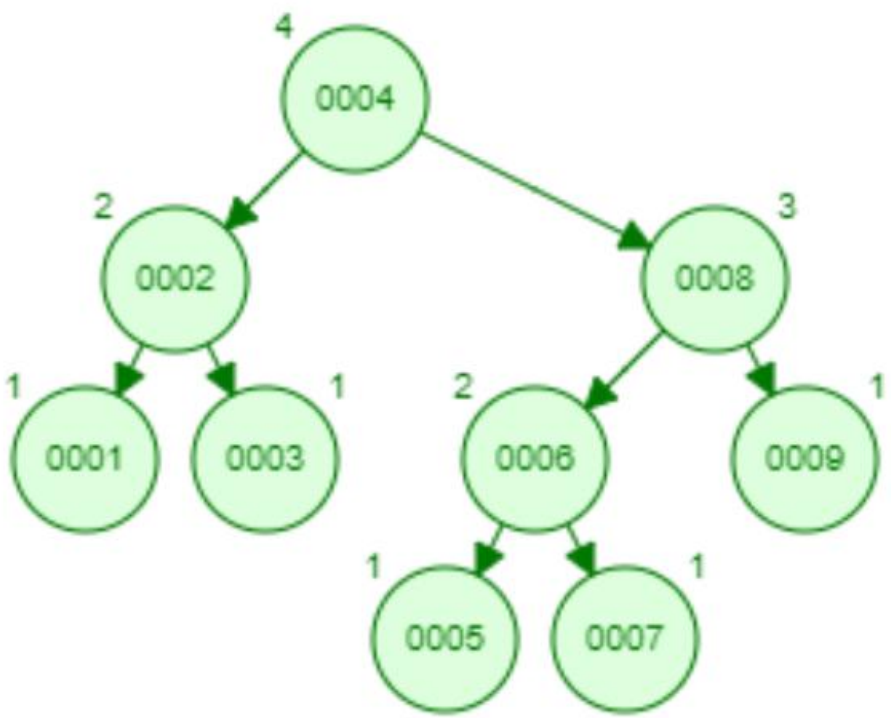


Insertion

- Beginning with an empty AVL tree, perform step-by-step the insertion of the following values in the order given

9, 1, 4, 2, 3, 9, 5, 8, 6, 7, 4



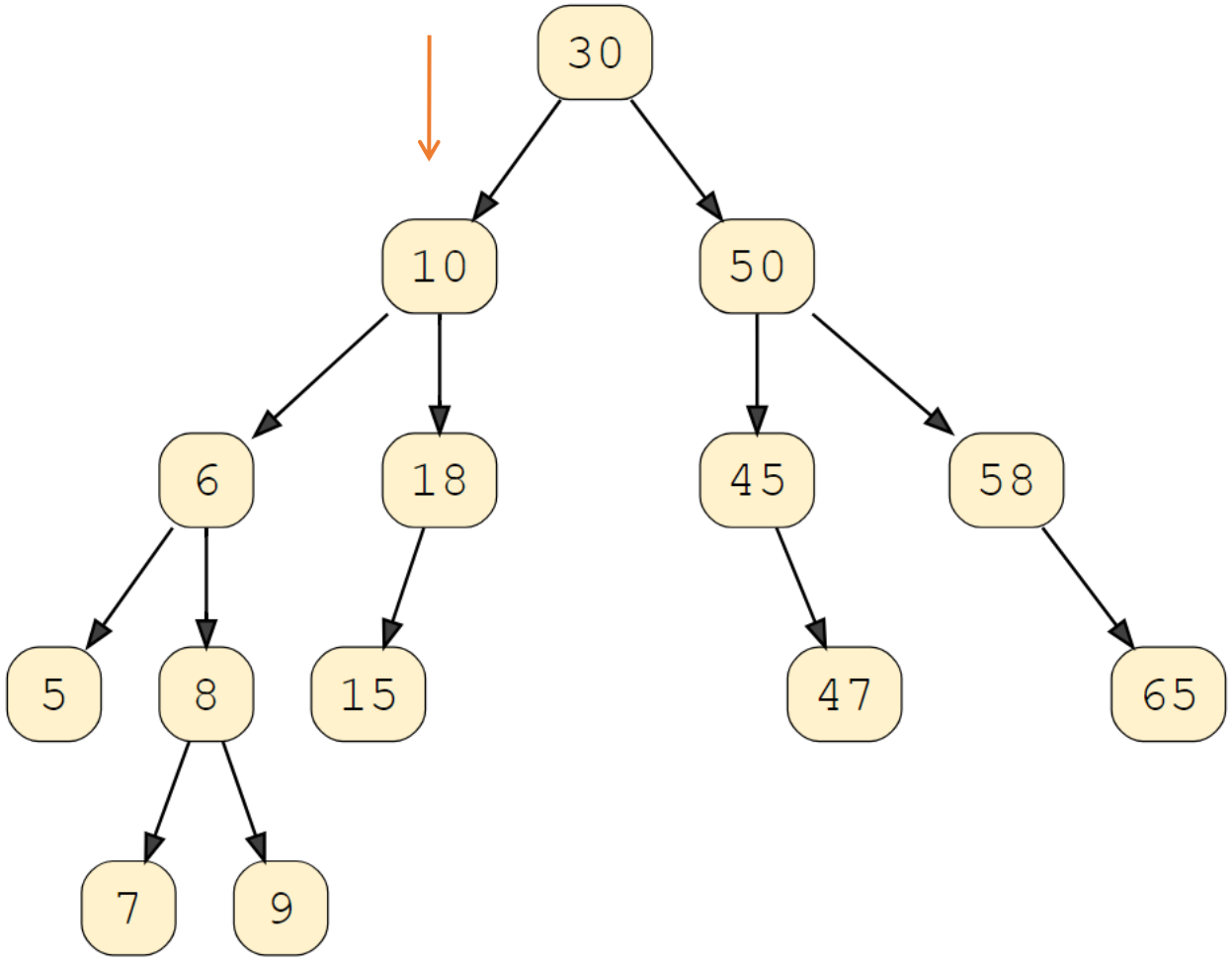


Deletion

- Deleting a node from an AVL tree is similar to that in a binary search tree
  - Delete the node (in 3 case of BST)
  - Rebalance the tree once the node is deleted

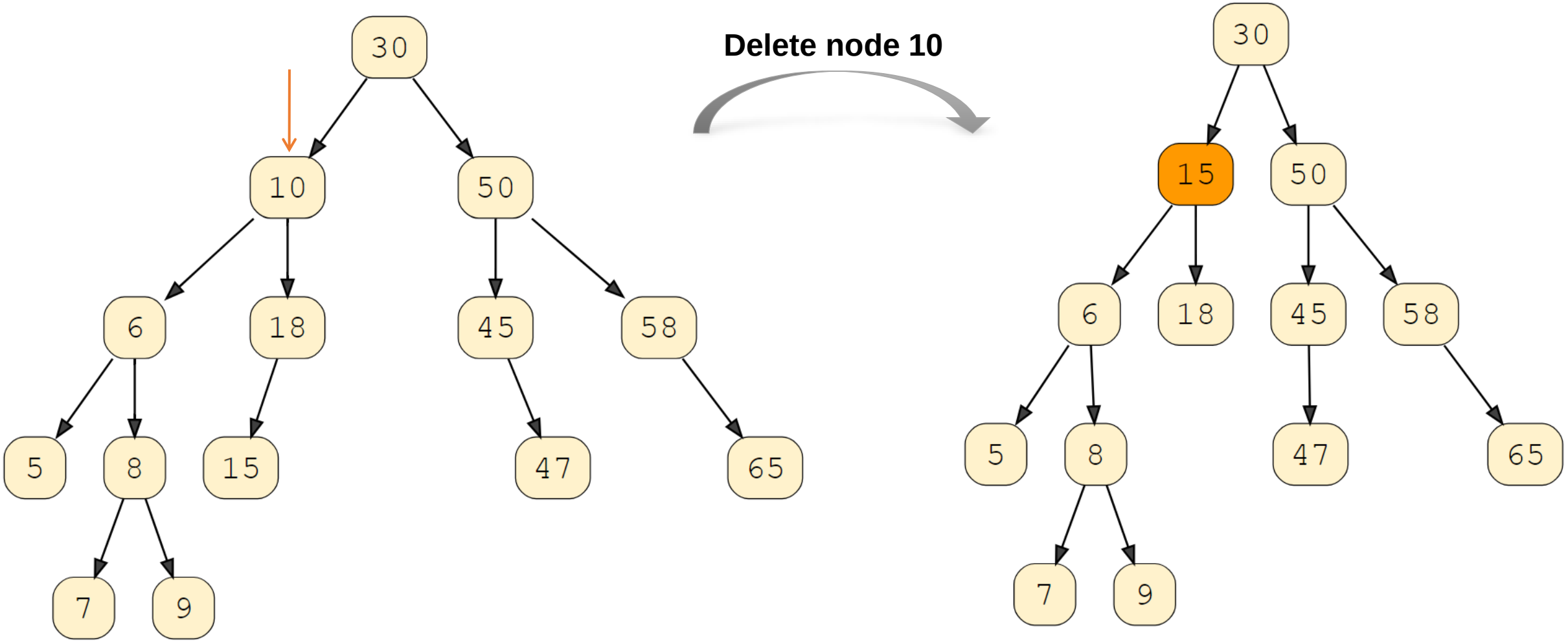
Deletion

- Delete Node 10:



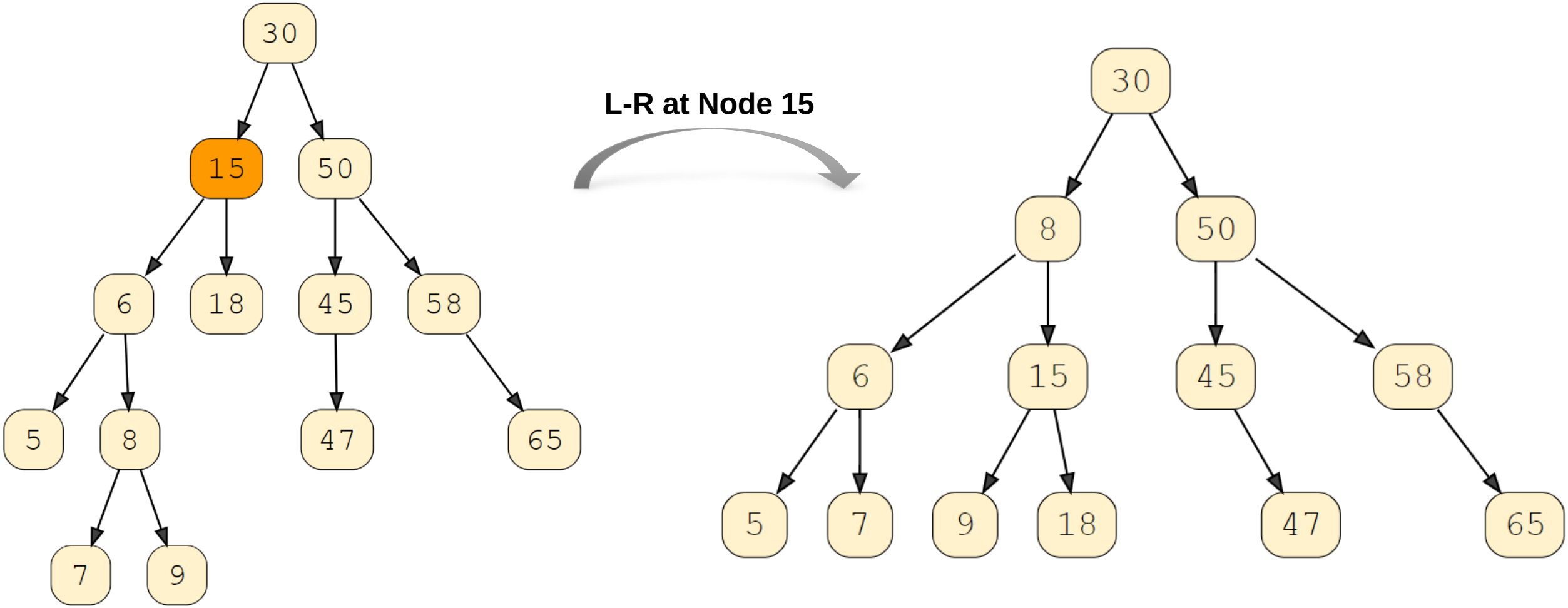
Deletion

- Delete Node 10:



Deletion

- Rebalance the tree



THANK YOU
for YOUR ATTENTION