CME 2001
Data Structures and Algorithms

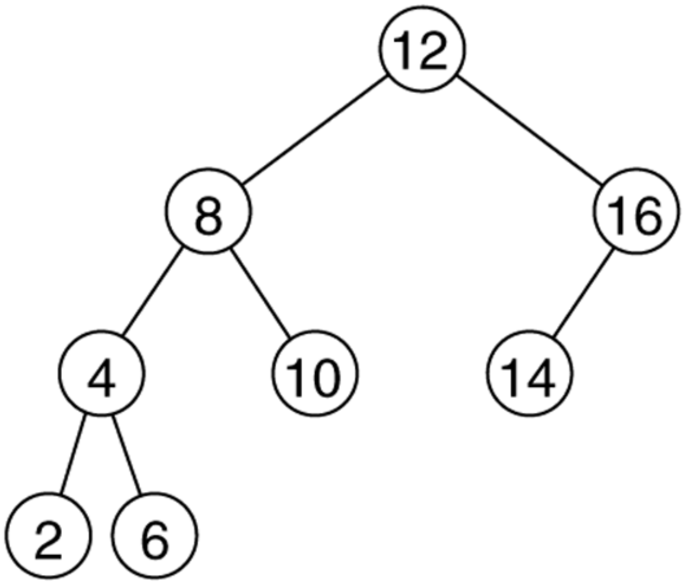
Zerrin Işık
zerrin@cs.deu.edu.tr

AVL Trees

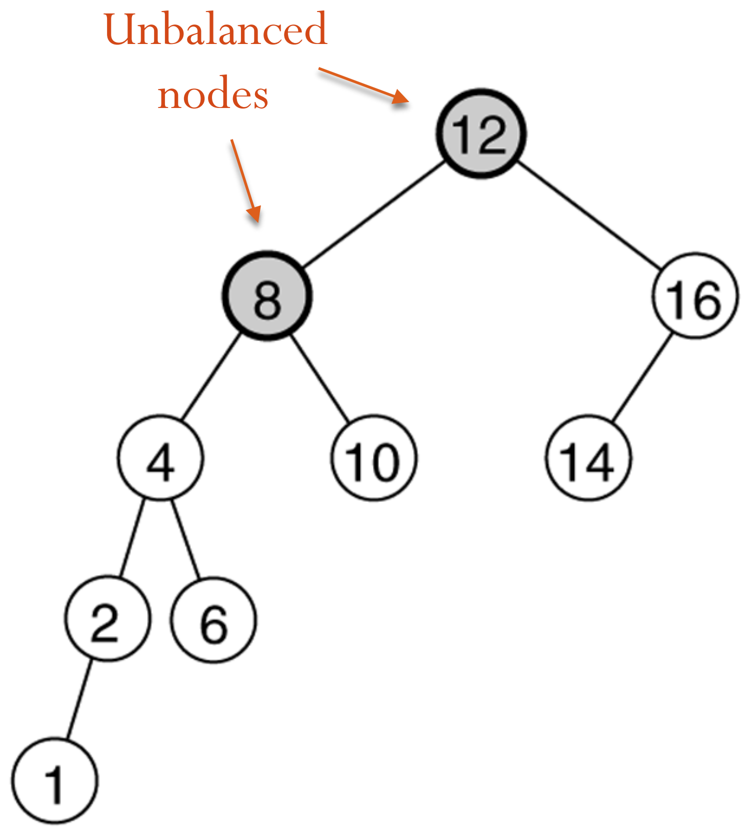
What is an AVL tree?

- A binary search tree with a ***balance*** condition.
- AVL is named for its inventors: **A**del'son-**V**el'skii and **L**andis
- *Approximates* the ideal tree : completely balanced tree
- Maintains a height close to the minimum.

Definition: An AVL tree is a BST such that for any node in the tree, the height of the left and right sub-trees can differ by at most 1.

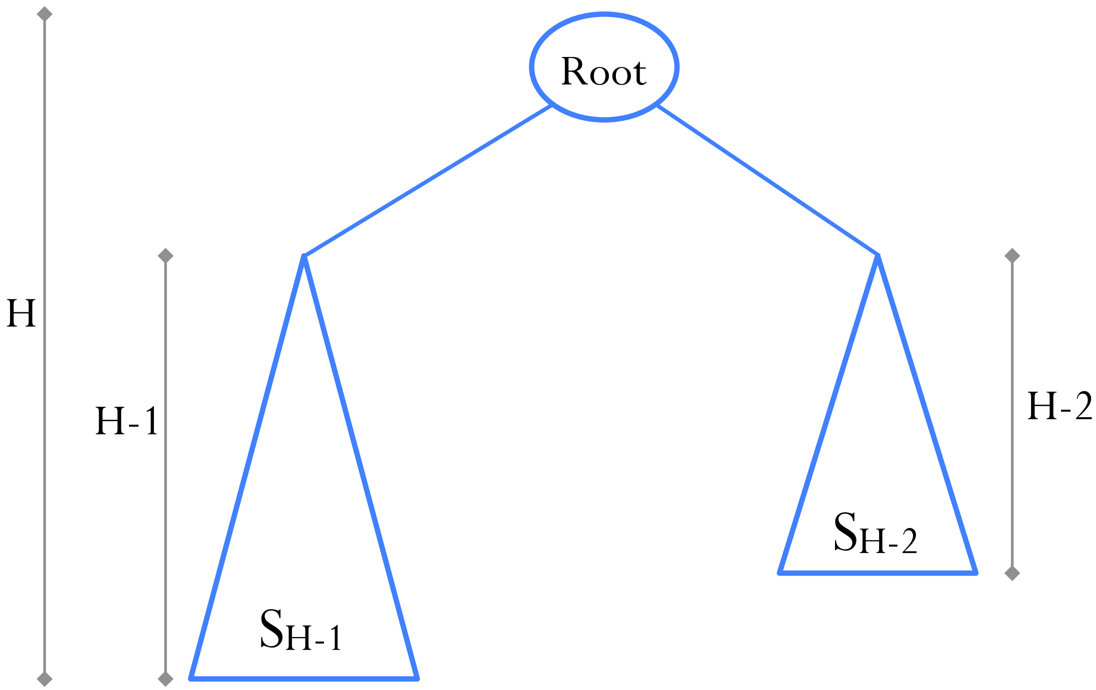


AVL Tree



Binary-Search-Tree

Balance property



Balance(tree) = | height(left subtree) - height(right subtree) | ≤ 1

Properties

- The depth of an AVL tree is very close to the optimal *lgN*.
- So, all searching operations in an AVL tree have *logarithmic* worst-case bounds.
- An update (insert or delete) could destroy the balance. It must then be rebalanced.
- After an insertion, only nodes that are on the path from the insertion point to the root can have their balances altered.

Rebalance AVL trees

- Suppose the node to be rebalanced is X:

  Case 1: An insertion in the left subtree of the left child of X,

  Case 2: An insertion in the right subtree of the left child of X,

  Case 3: An insertion in the left subtree of the right child of X, or

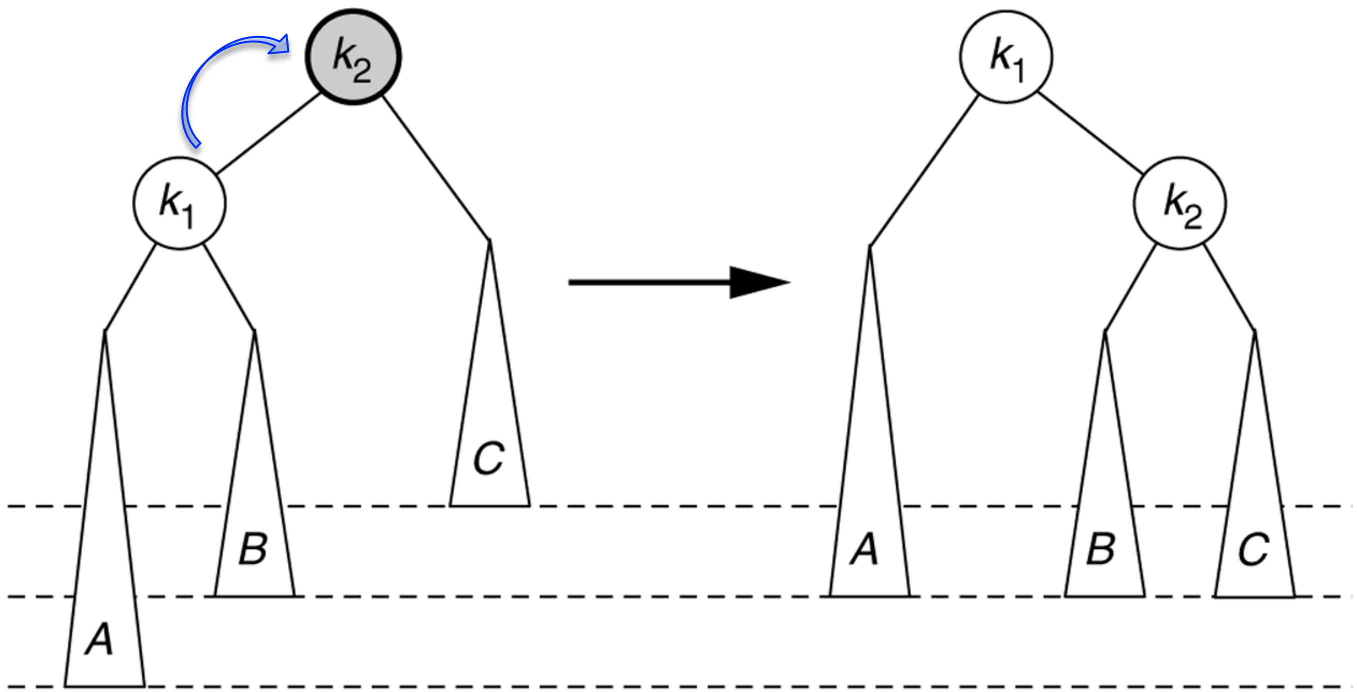
  Case 4: An insertion in the right subtree of the right child of X.

- Balance is restored by ***tree rotations***:
  - Case 1 and 4 are symmetric and performed by a *single rotation*.
  - Case 2 and 3 are symmetric and performed by a *double rotation*.

Single Rotation

- It switches the roles of the parent and child while maintaining the search order.
- Single rotation handles the outside cases (Case 1, Case 4).
- The result is a new binary search tree that satisfies the height property.

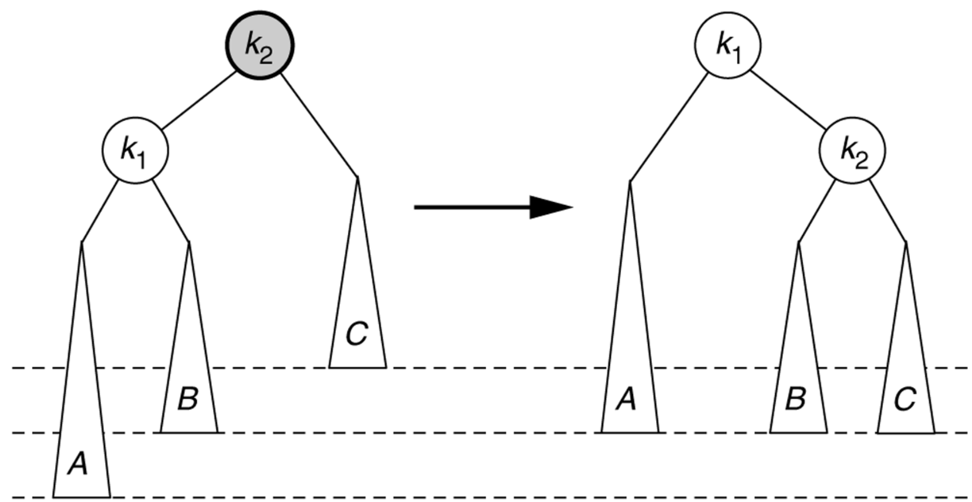
Rotate Right – Fix Case 1



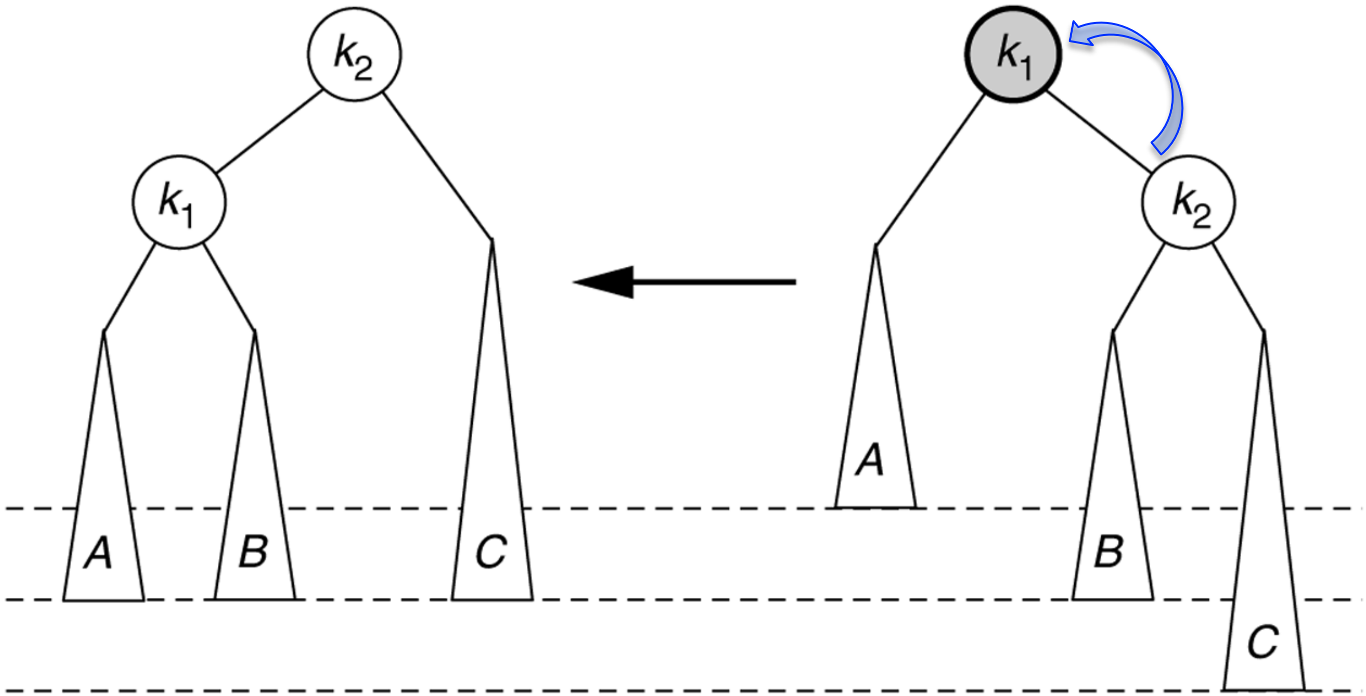
Before rotation After rotation

Rotate Right - Code

```
Node ROTATE_RIGHT(Node k2)
  k1 = k2.left
  k2.left   = k1.right
  k1.right  = k2
  k1.parent = k2.parent
  k2.parent = k1

  if (k2.left != NULL)
    k2.left.parent = k2

  Arrange-new-heights

  return (k1)
```



Rotate Left – Fix Case 4

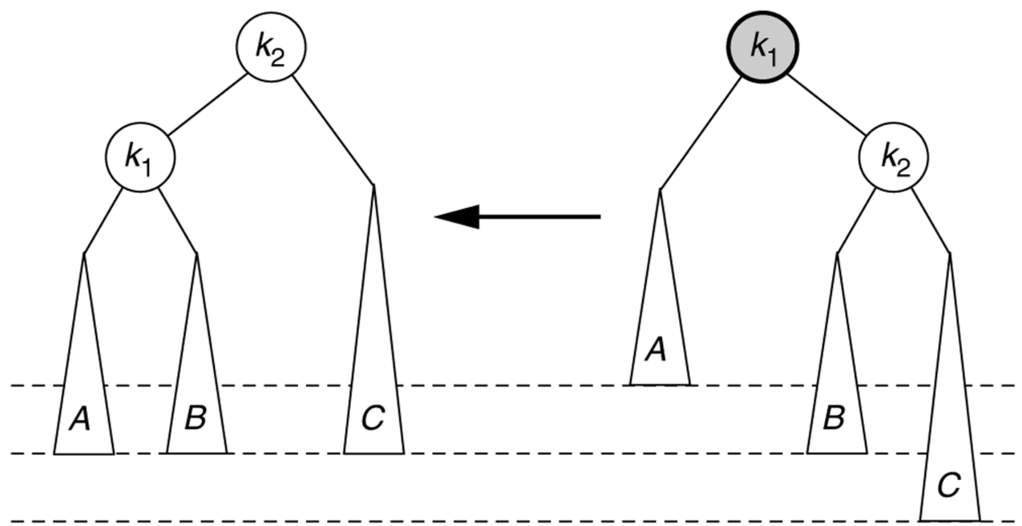


After rotation

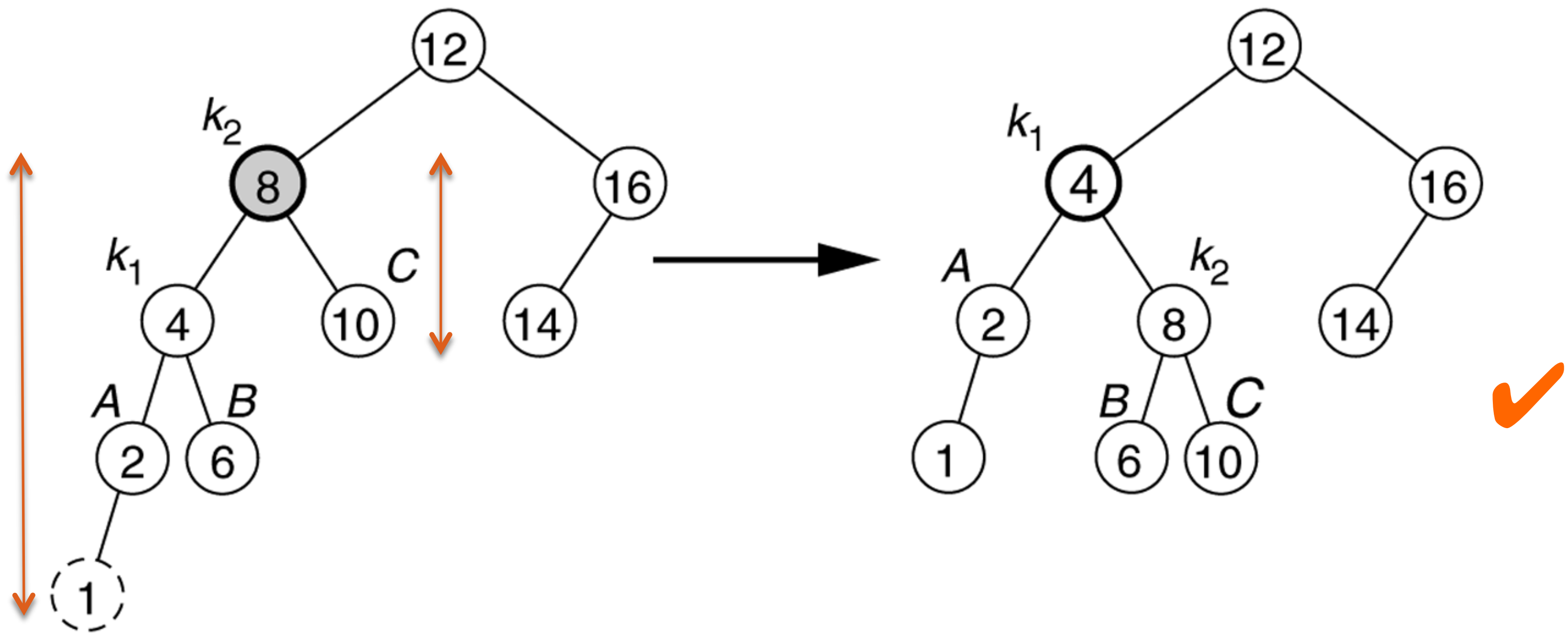
Before rotation

Rotate Left - Code

```
Node ROTATE_LEFT(Node k1)
  k2 = k1.right
  k1.right  = k2.left
  k2.left   = k1
  k2.parent = k1.parent
  k1.parent = k2

  if (k1.right != NULL)
    k1.right.parent = k1

  Arrange-new-heights

  return (k2)
```



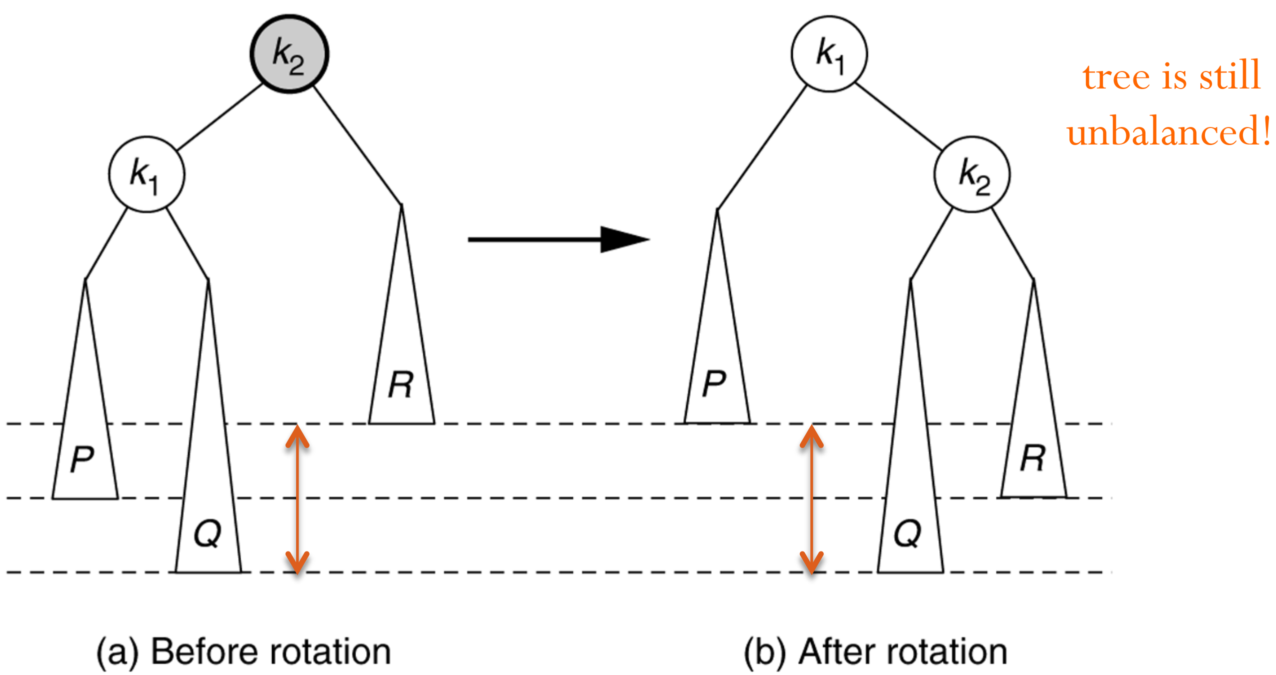
Single Rotation Example



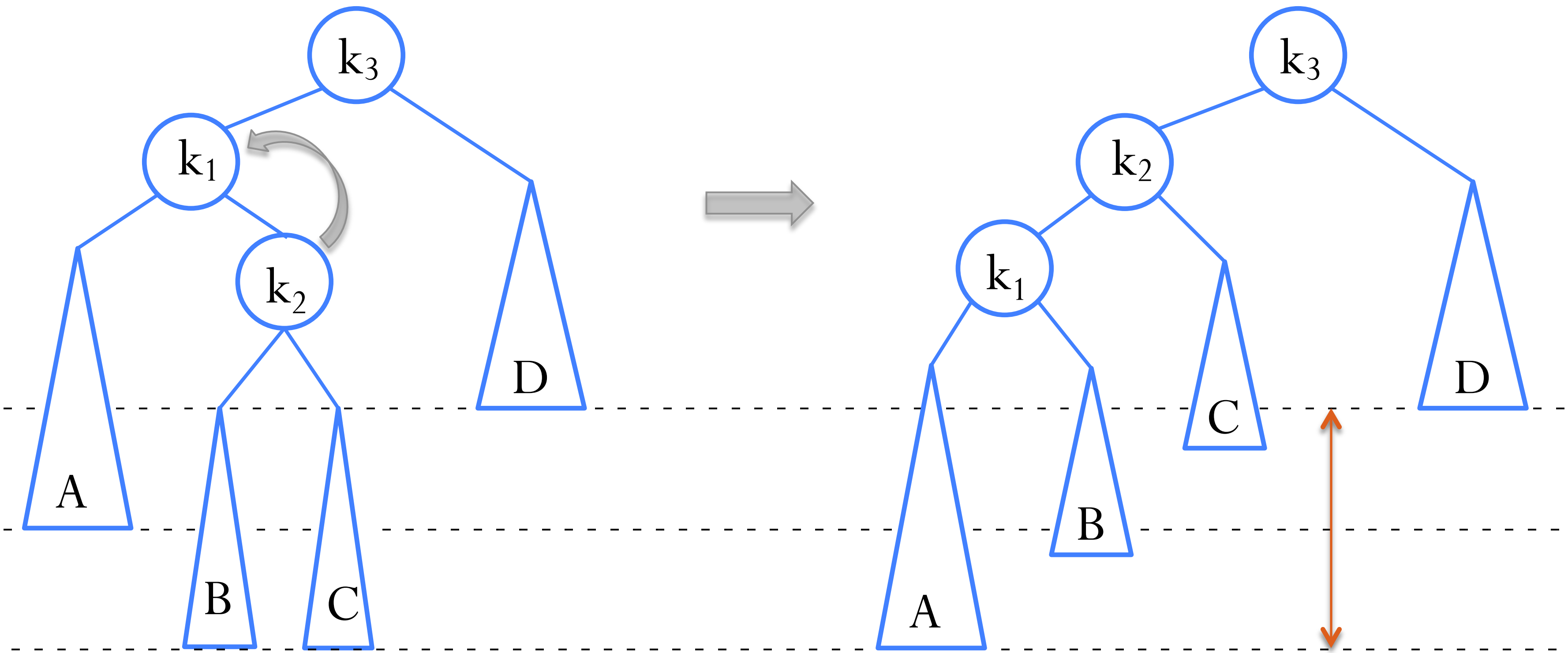
- Right-rotation is required.

Double Rotation

- It fixes Case 2 and Case3, because they involve tree nodes and four subtrees.
- Single rotation can not fix Case 2:



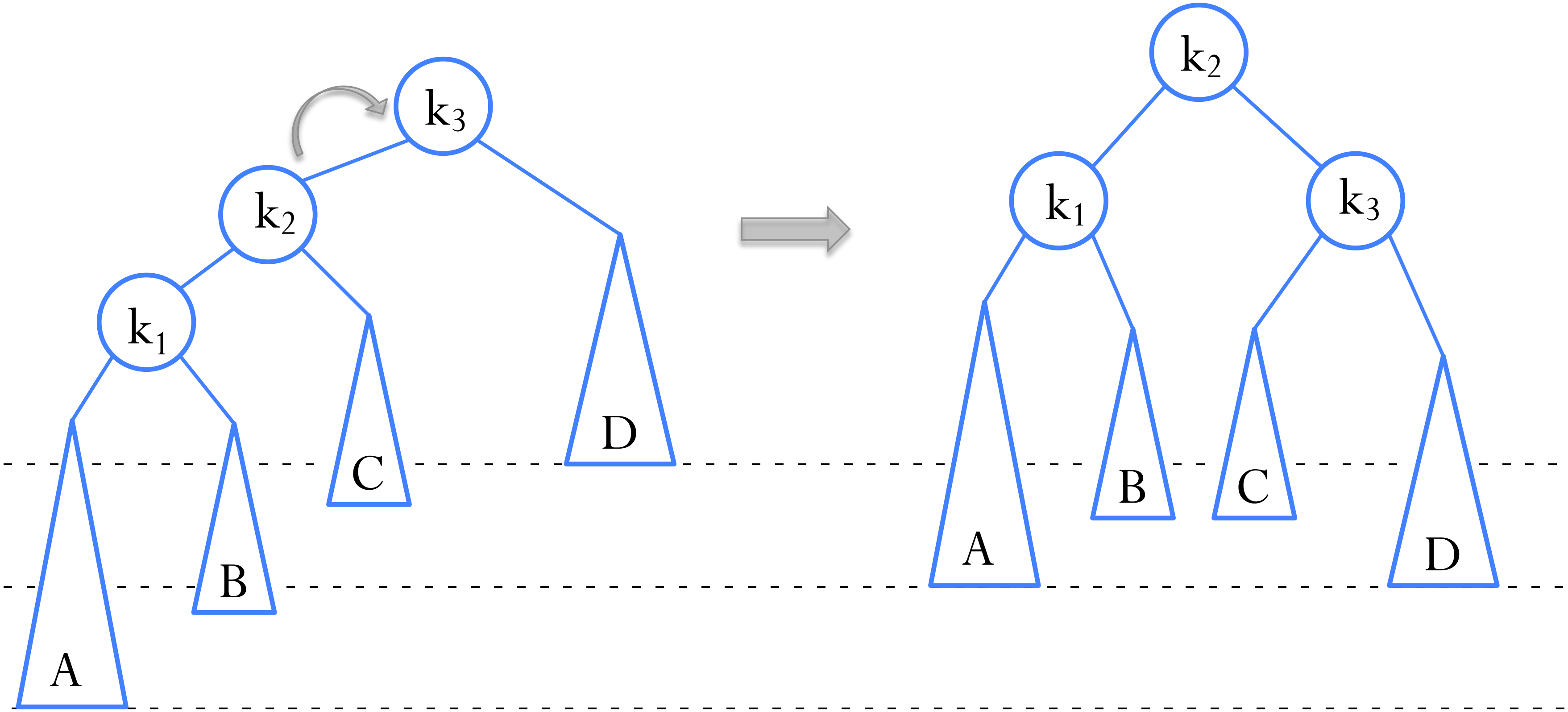
Left-right double rotation : Case 2



Before rotation

After left rotation

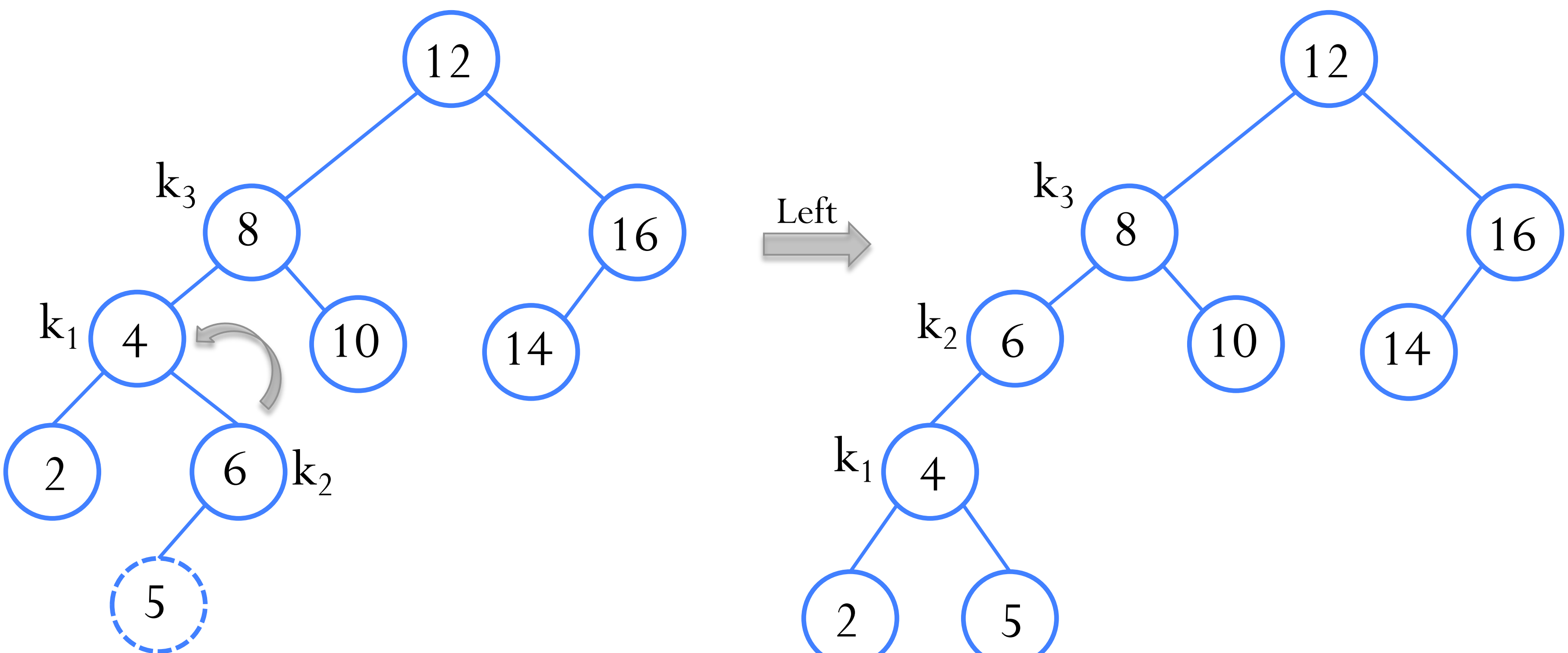
Left-right double rotation …



After left rotation

After right rotation ✔

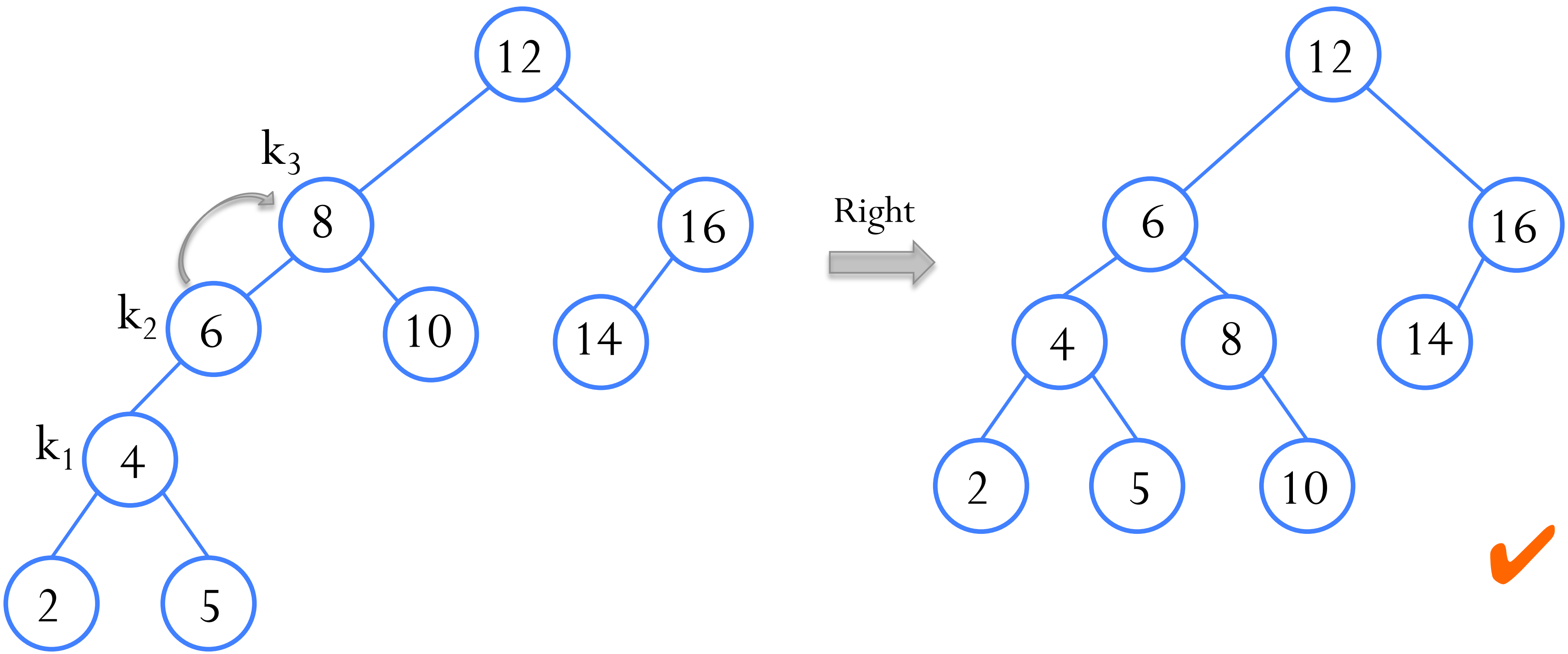
Example: Left-right double rotation



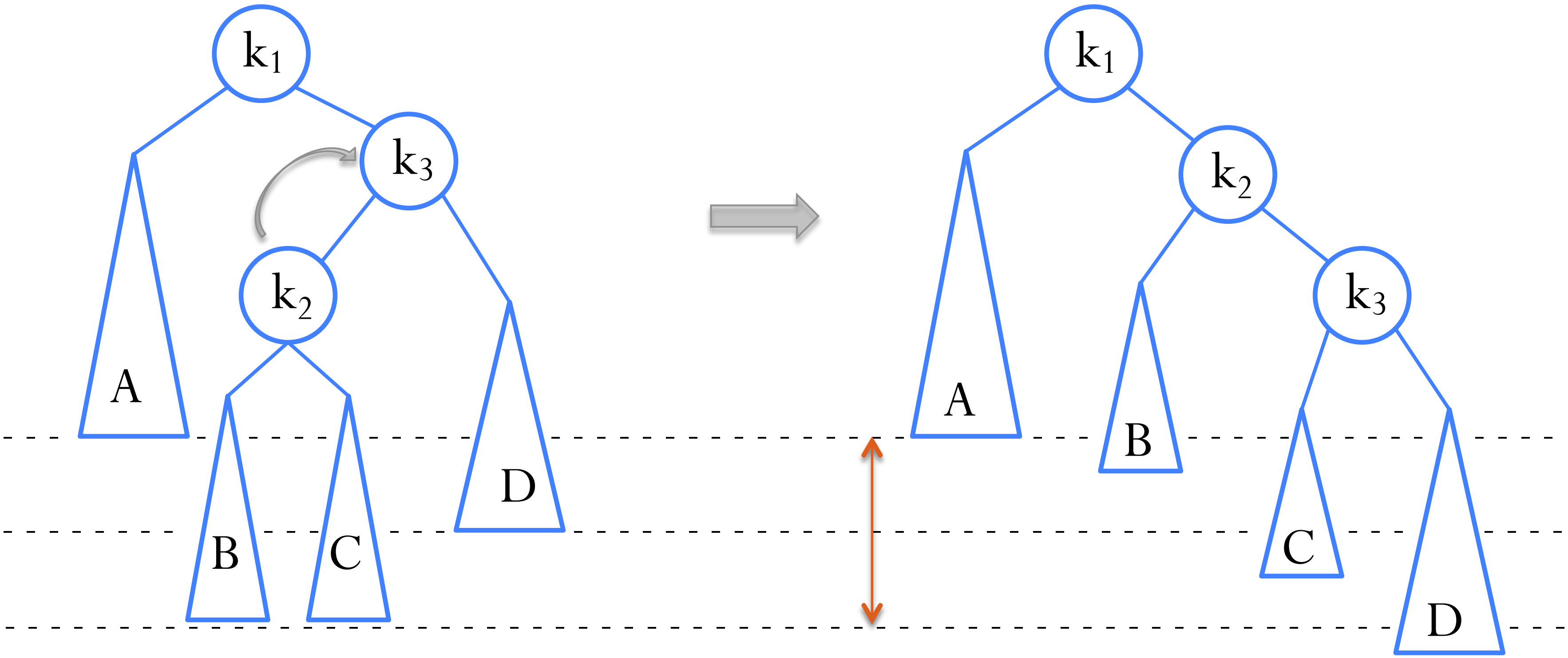
After the insertion of 5, tree became unbalanced

After left rotation, tree is still unbalanced

Example: Left-right double rotation …



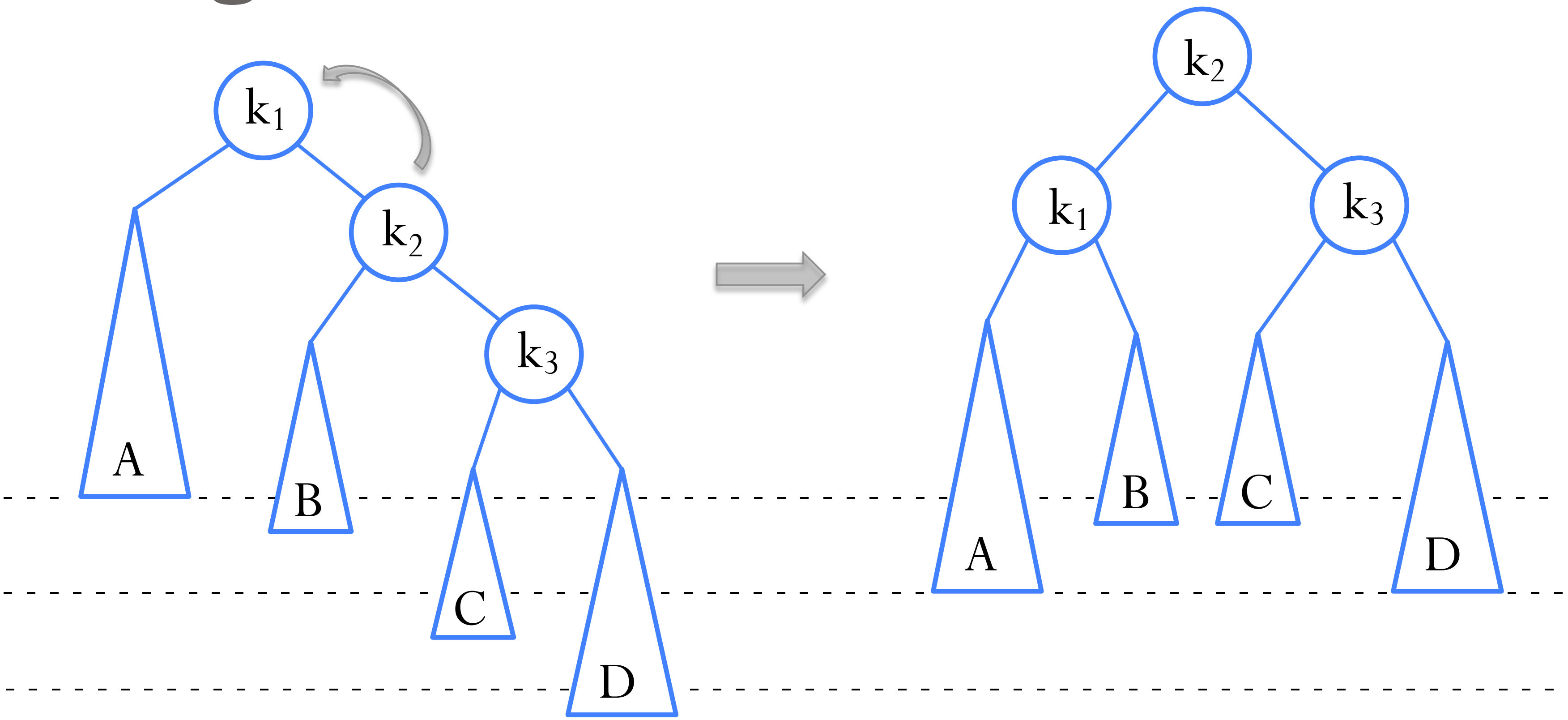
Right-left double rotation: Case 3



Before rotation

After right rotation

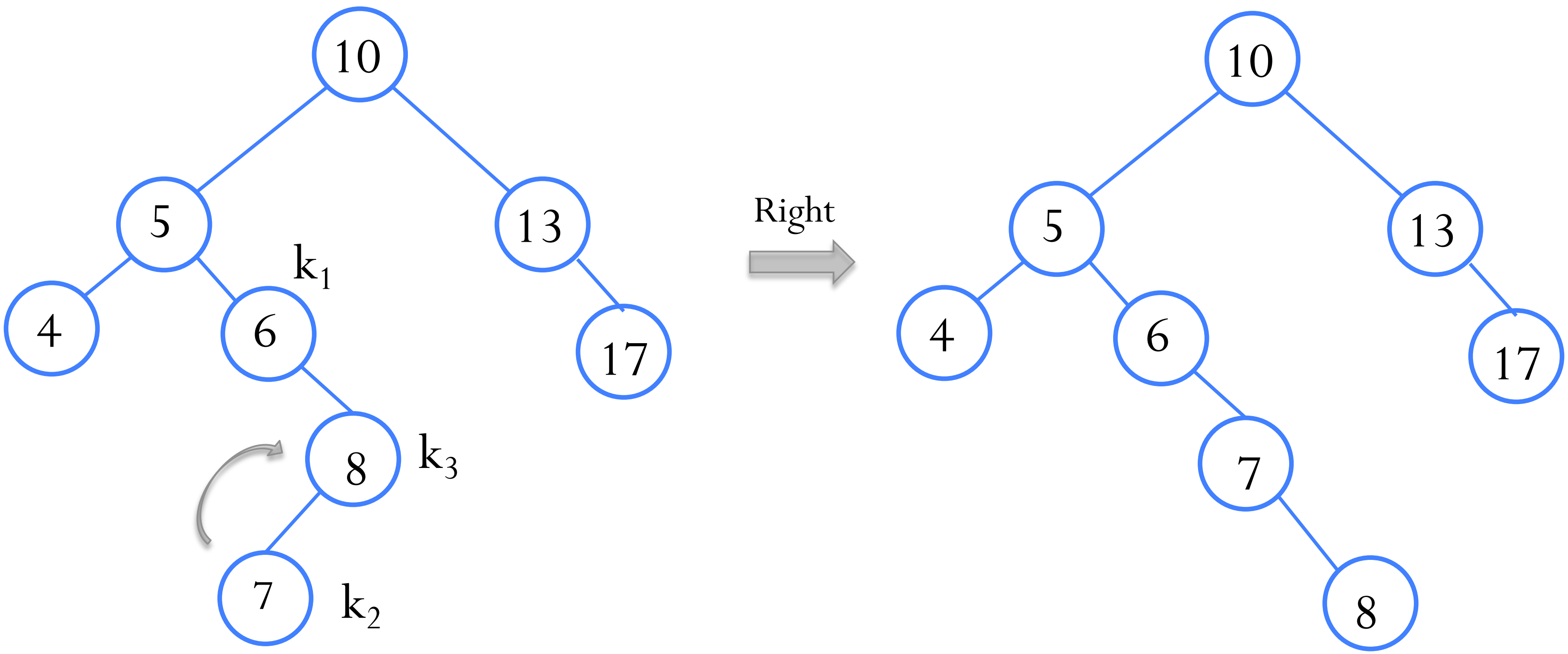
Right-left double rotation …



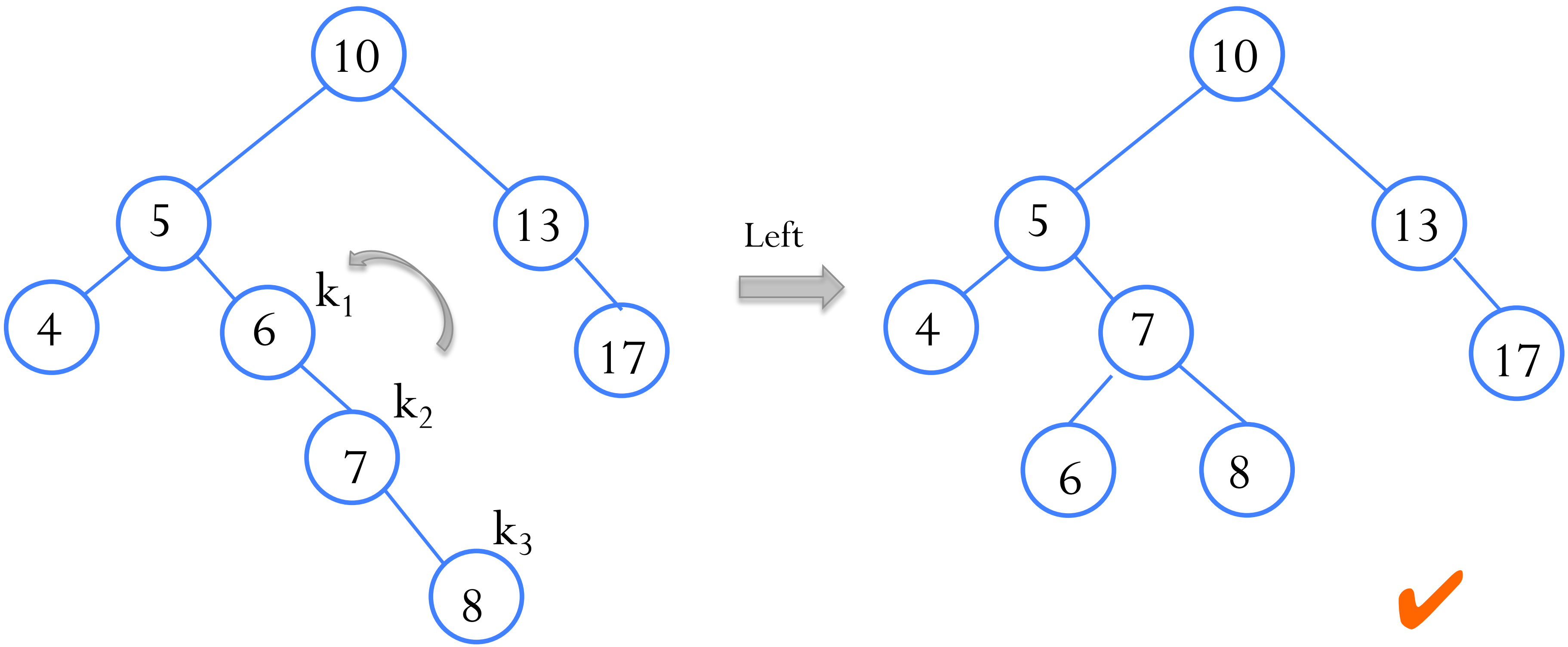
After right rotation

After left rotation ✔

Example: Right-Left double rotation …



Example: Right-Left double rotation …



Node Deletion

- Deletion of a node ***x*** from an AVL tree requires the same operations (e.g., single and double rotations), which are used for insertion.

- Each node is associated with a ***balance factor:***

$$| \text{ height(left subtree) - height(right subtree) } | \leq 1$$

Node Deletion - Method

1. Reduce the problem to the case when the node x to be deleted has at most one child.
   - If x has two children replace it with its immediate predecessor y under inorder traversal.
   - Delete y from its original position, by using y in place of x in each of the following steps.

Node Deletion – Method ...

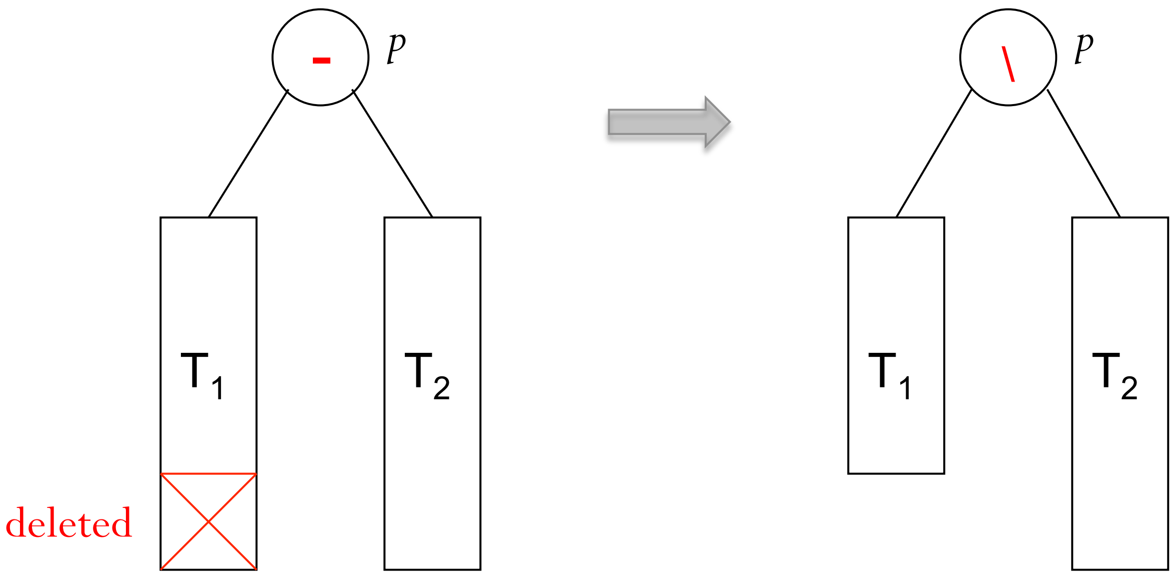
2. Delete the node *x* from the tree.
   - Trace the effects of this change on height through all the nodes on the path from *x* back to the root.
   - Use a boolean variable **`shorter`** to show if the height of a subtree has been shortened.
   - The action to be taken at each node depends on
     - the value of **`shorter`**
     - balance factor of the node
     - or the balance factor of a child of the node.
3. Initially **`shorter`** is set true. While it is true, apply for each node ***p*** on the path from the parent of *x* to the root, the following steps:

Node Deletion – Method ...

Case 1:

The current node ***p*** has balance factor equal.

- Change the balance factor of ***p***.
- **`shorter`** becomes false.



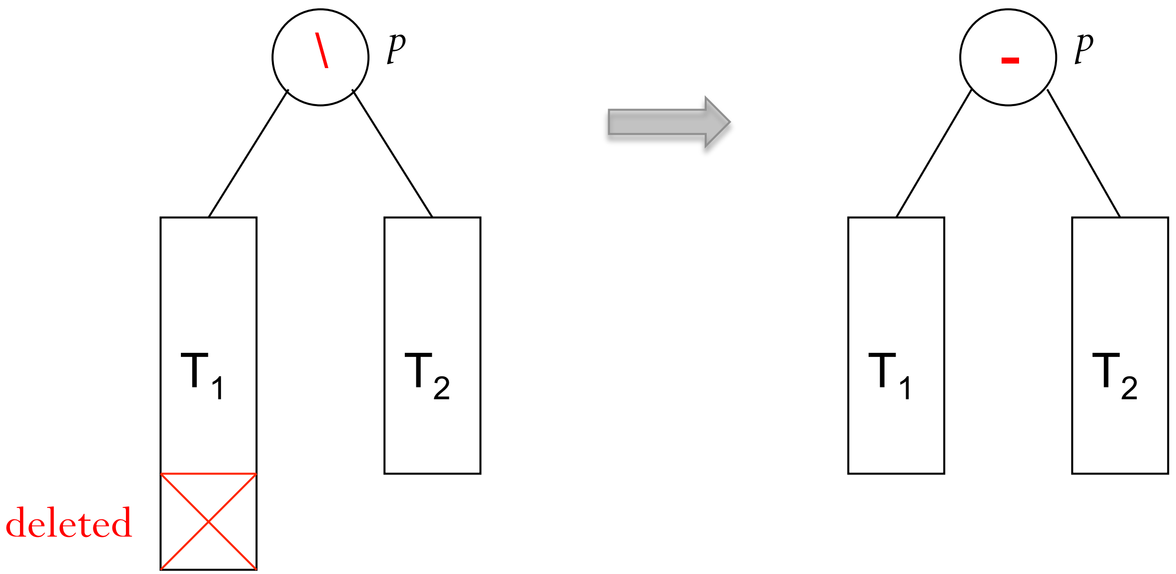
- No rotations
- Height unchanged

Node Deletion – Method ...

Case 2:

The current node ***p*** has balance factor is not equal and the longer subtree was shortened.

- Change the balance factor to equal.
- Leave **`shorter`** as true.



- No rotations
- Height reduced

Node Deletion – Method …

Case 3:

The current node ***p*** has balance factor is not equal, and the shorter subtree was shortened.

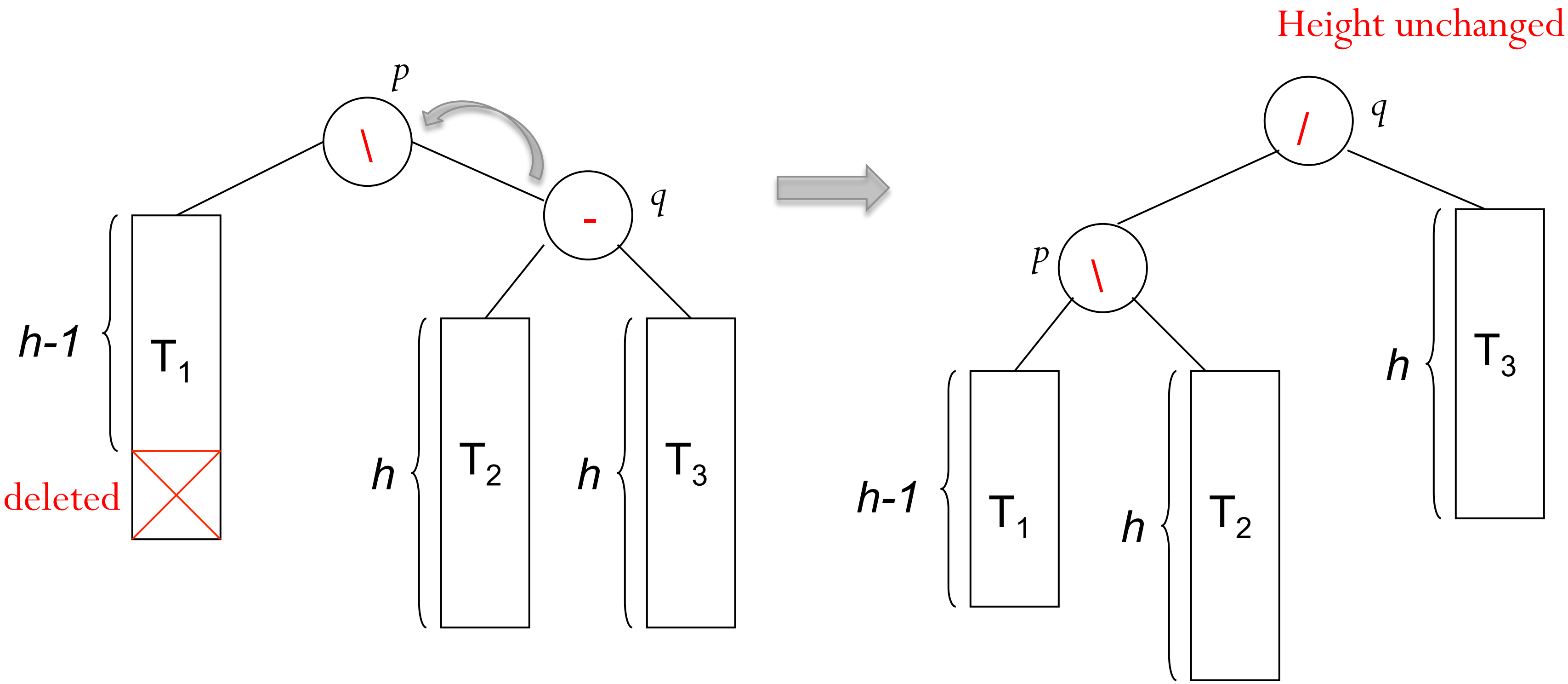
- Rotation is required.
- Let ***q*** be the root of the longer subtree of ***p***.
- There are three cases according to the balance factor of ***q***.

Node Deletion – Method …

Case 3a:

The balance factor of ***q*** is equal

- Apply a single rotation.
- `shorter` becomes false.

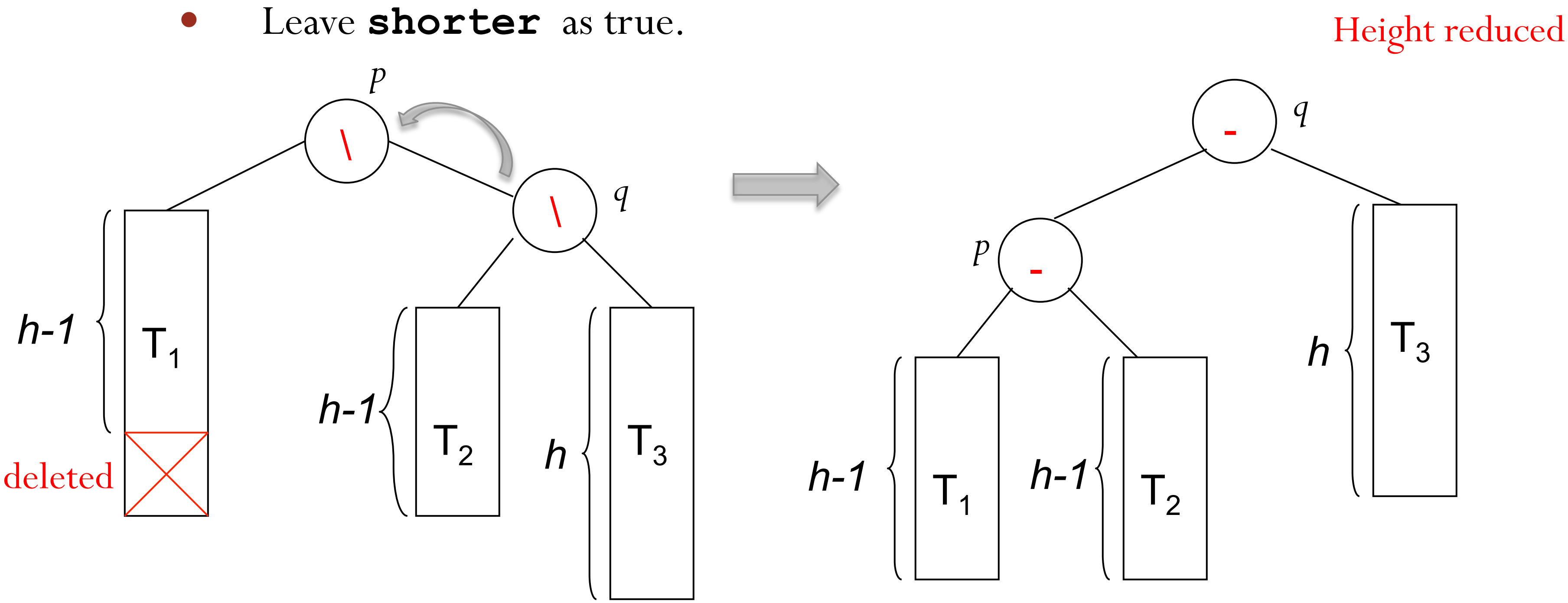


Node Deletion – Method ...

Case 3b:

The balance factor of ***q*** is the same with that of ***p*.**

- Apply a single rotation.
- Set the balance factors of ***p*** and ***q*** to equal.
- Leave **`shorter`** as true.

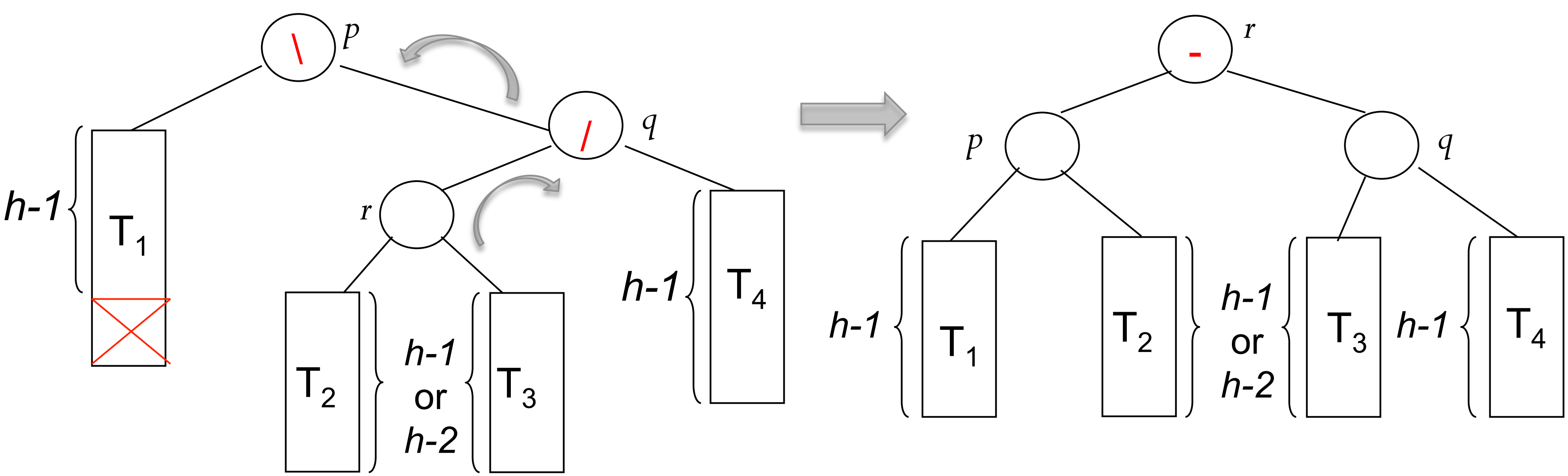


Node Deletion – Method …

Case 3c:

The balance factor of ***q*** are opposite**.**

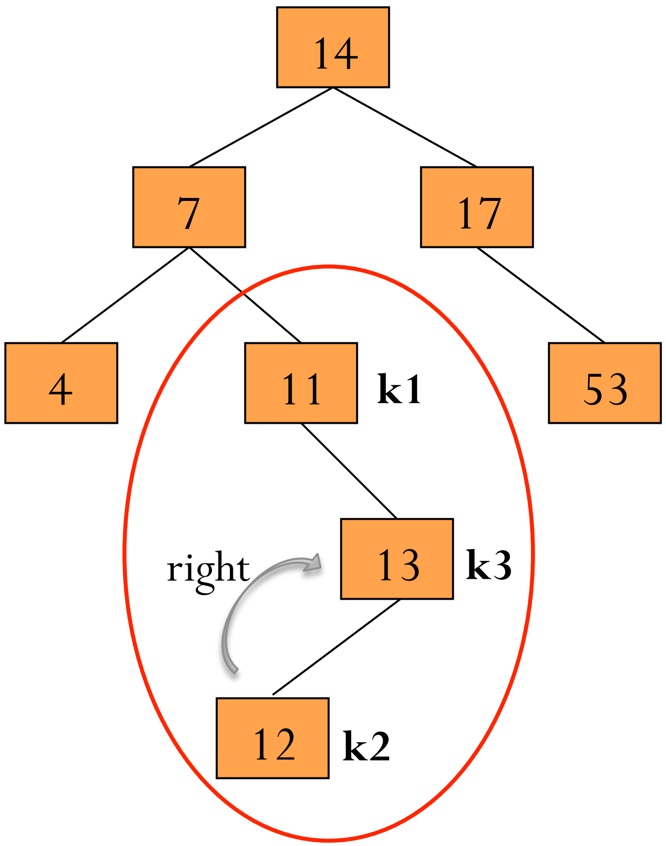
- Apply a double rotation.
- Set the balance factors of the new root to equal.
- Leave **`shorter`** as true.



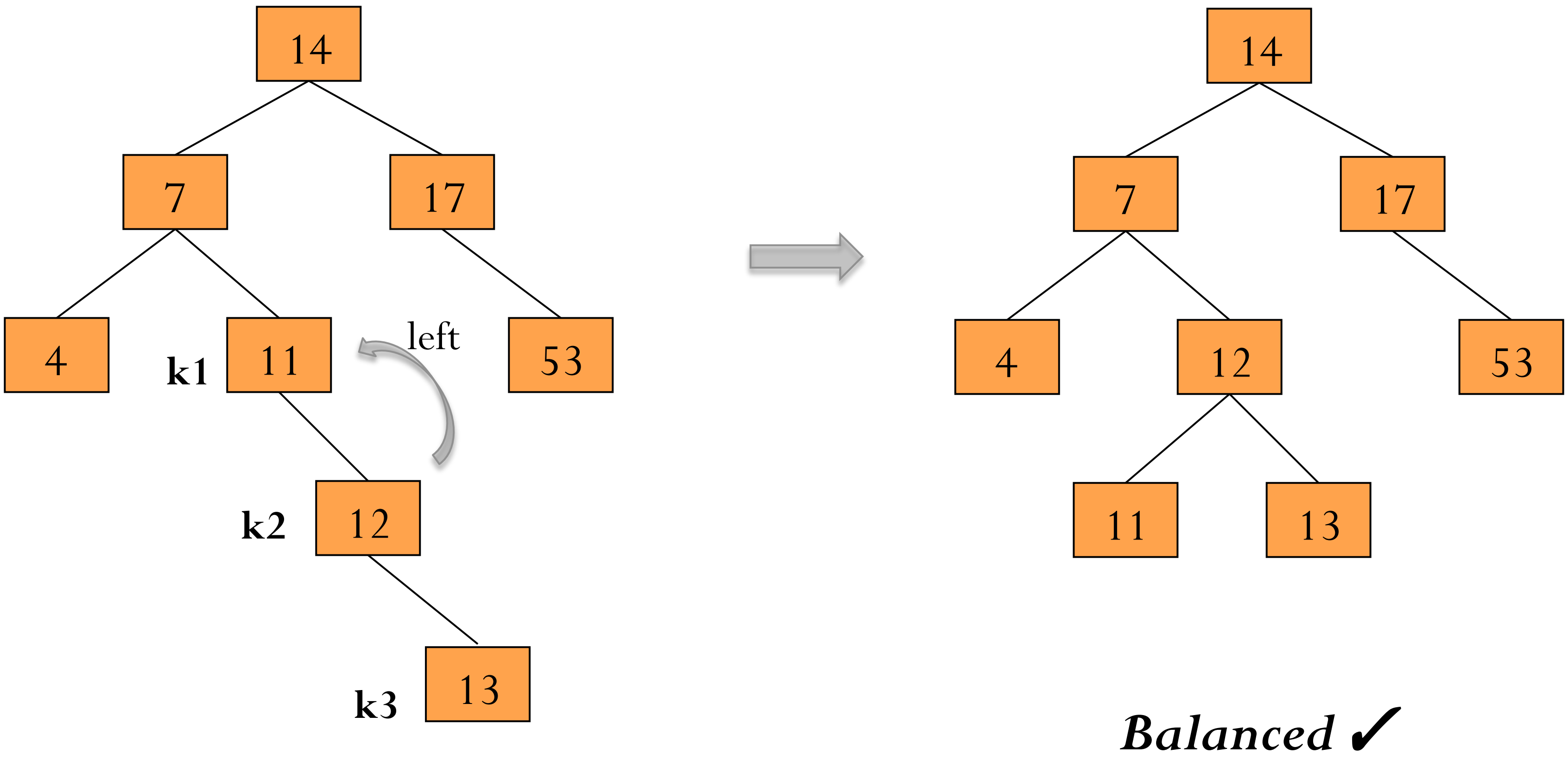
AVL Tree examples

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

- Insert 12.

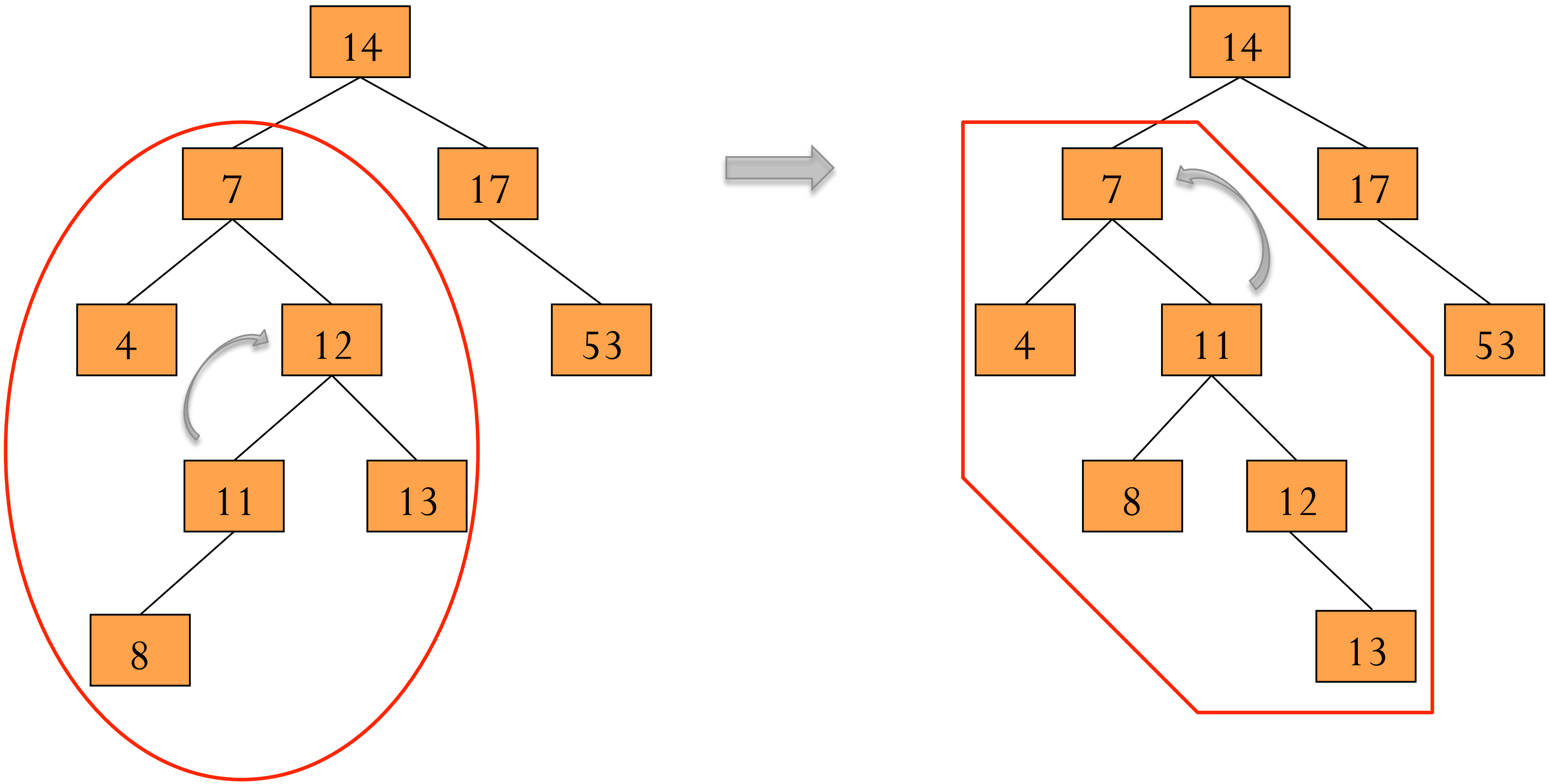


AVL Tree examples

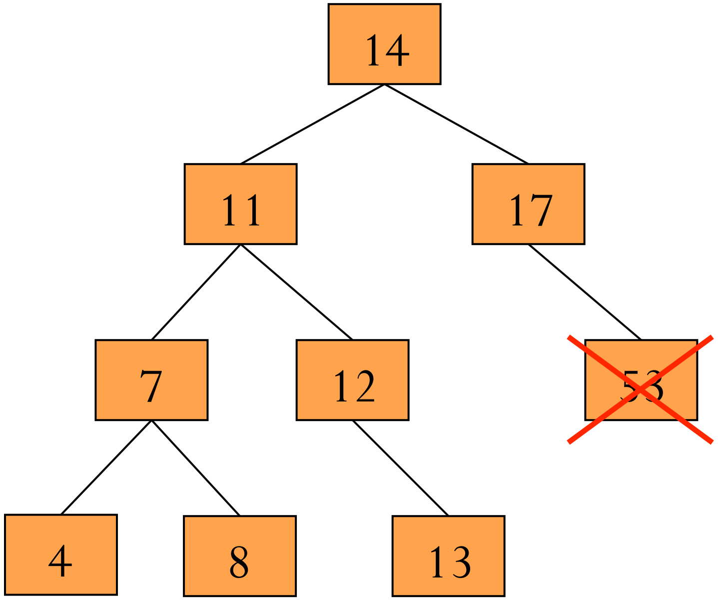


AVL Tree examples

- Insert 8



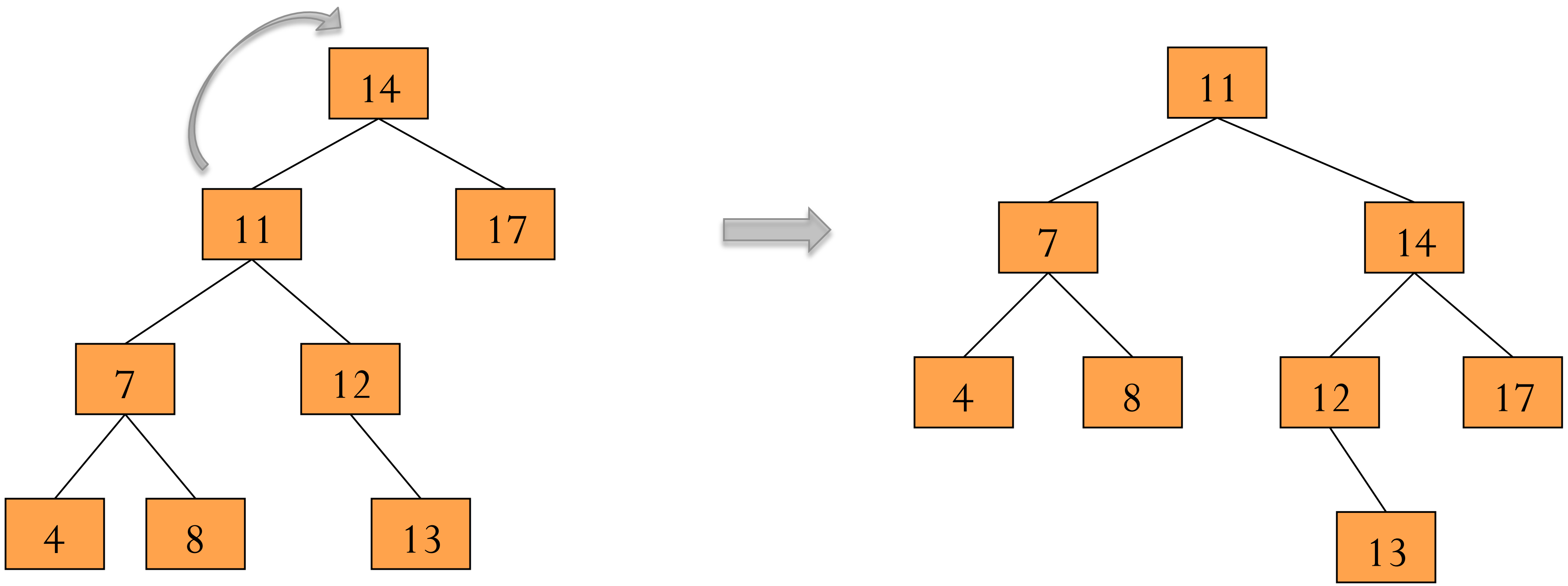
AVL Tree examples



Remove 53

Balanced ✓

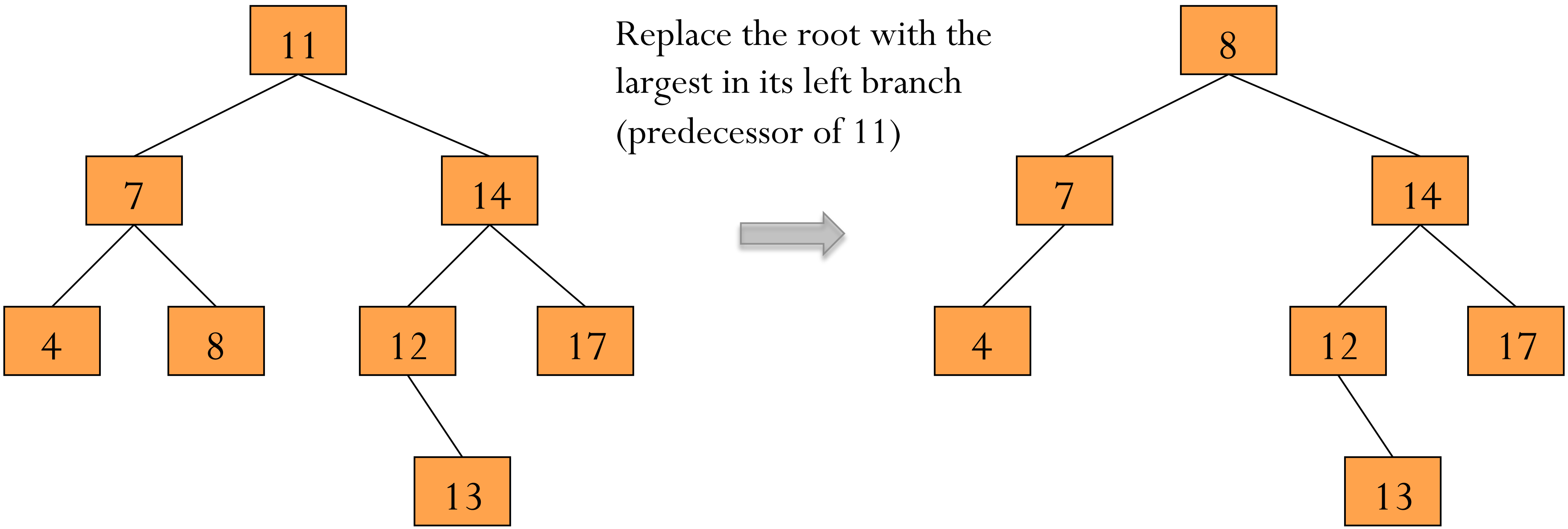
AVL Tree examples …



Unbalanced
Case 3a

Balanced ✓

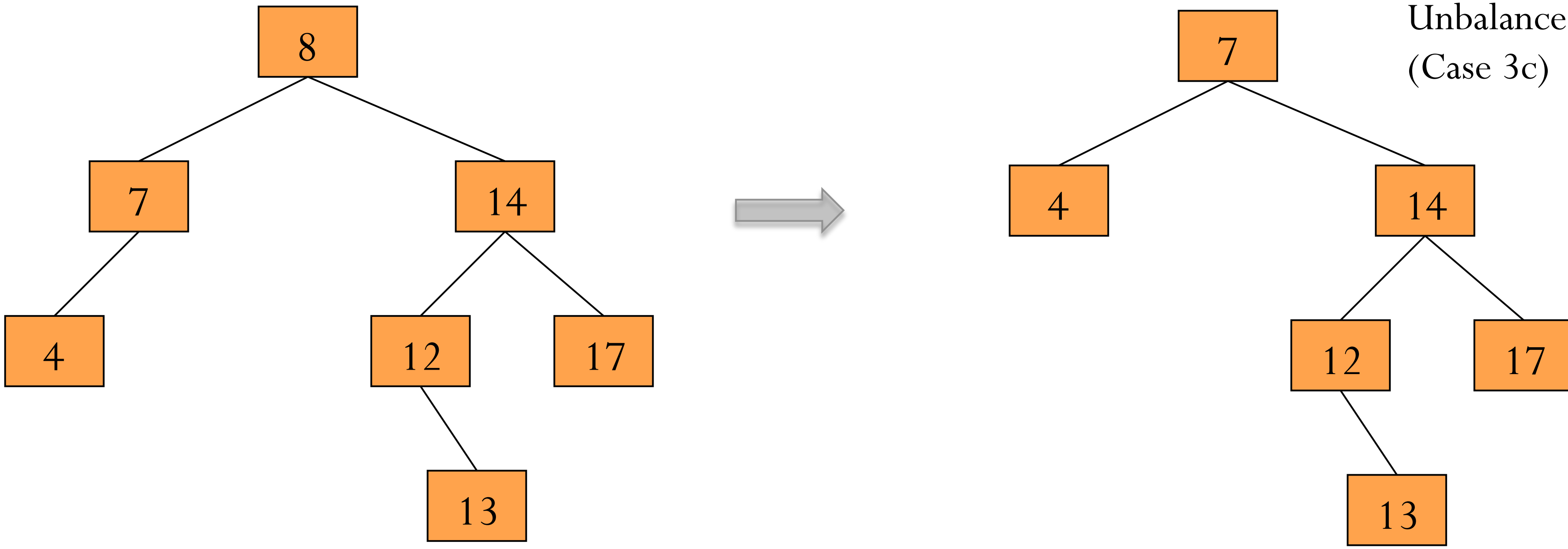
AVL Tree examples …



Replace the root with the largest in its left branch (predecessor of 11)

Remove *11*

AVL Tree examples …



Unbalanced
(Case 3c)

Remove *8*

AVL Tree examples …