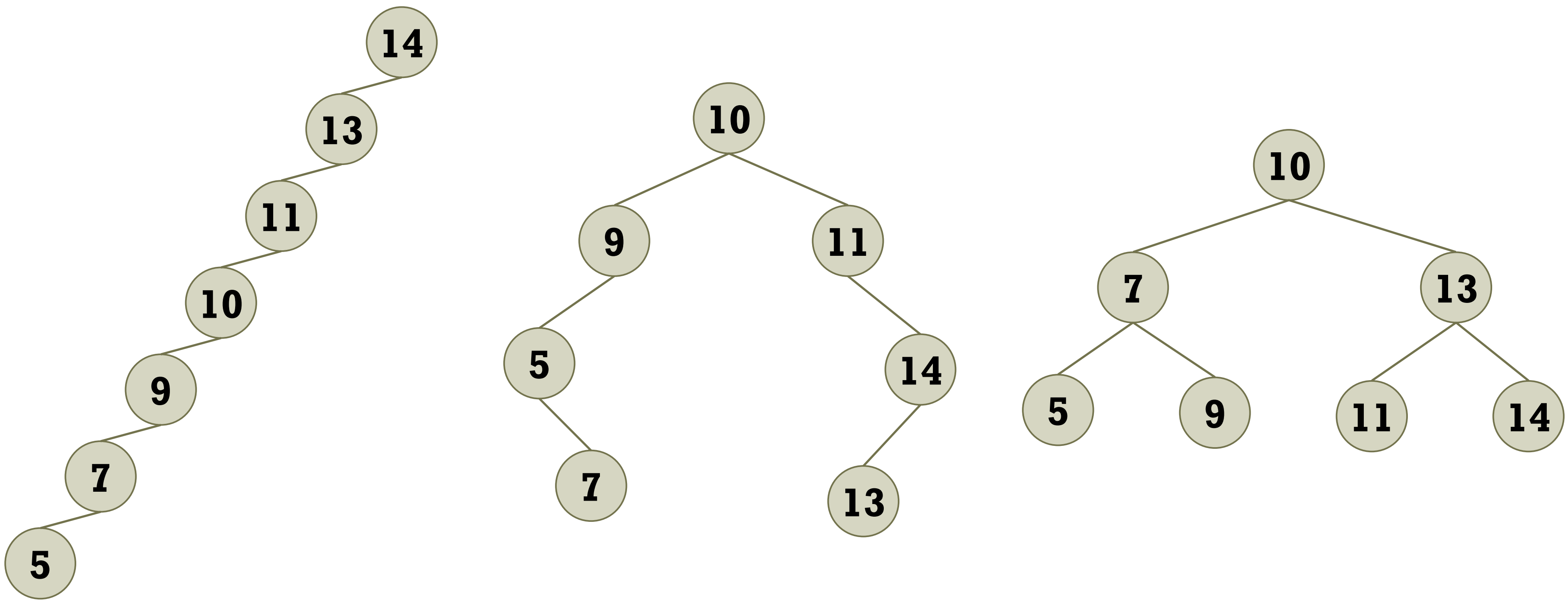
AVL Tree

Motivation: Unbalanced Tree

- Various binary search trees
  - The time complexity of binary search trees is so different.



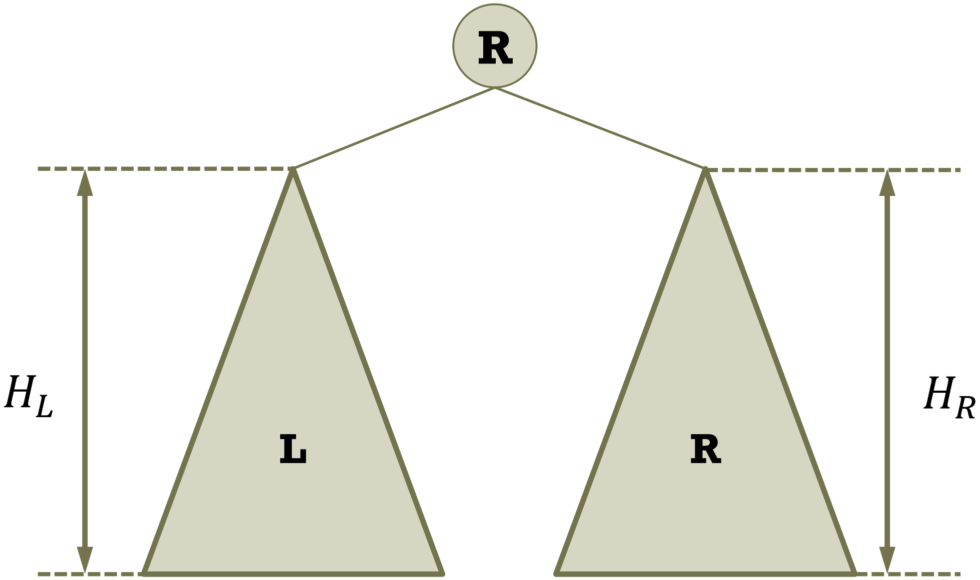
Motivation: Unbalanced Tree

- The time complexity of binary search tree
  - Average case: $O(\log n)$
  - Worst case: $O(n)$
  - Note: The time complexity highly depends on **the order of elements to be inserted**.

- How to improve the worst case of the BST?
  - Maintain the BST as a **balanced** binary tree for every insertion.

What is AVL Tree?

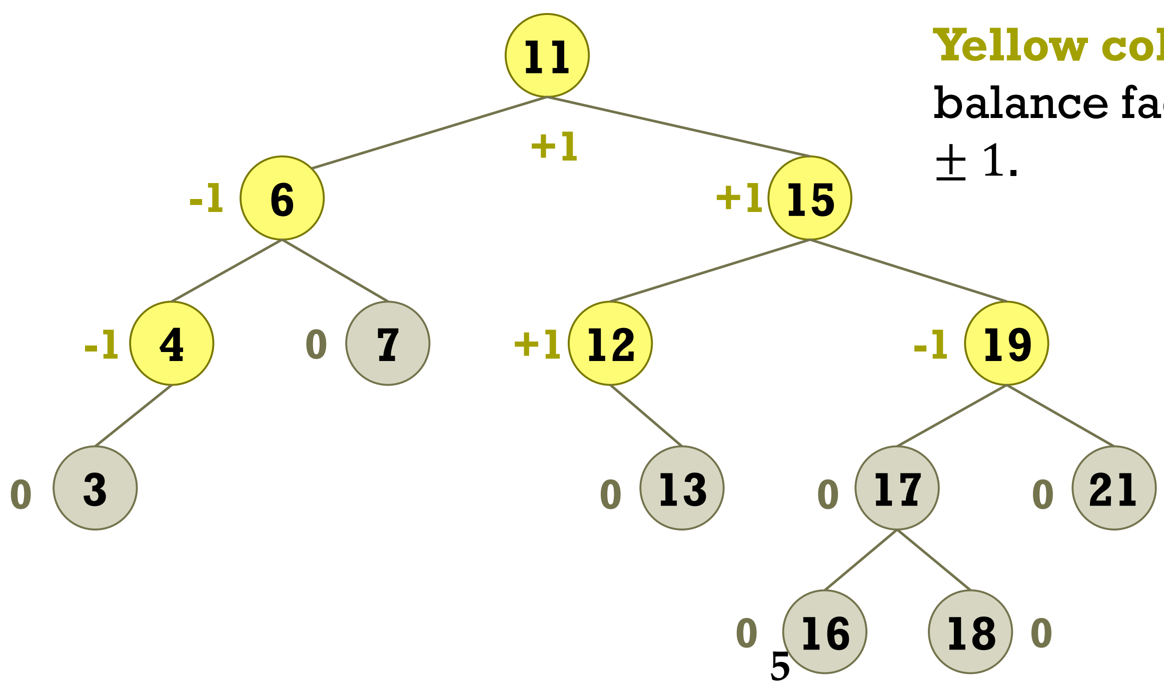
- Description
  - Invented by Georgy Adelson-Velsky and Evgenii Landis in 1962
  - **The first self-balancing tree**
    - It guarantees that the heights of two child subtrees of any node differ by at most one.
    - If the height of two child subtrees differs by more than one, **the rebalancing operation** is performed.



$\boldsymbol{T}$ **is balanced** $\boldsymbol{if}$
$|\boldsymbol{H_L - H_R}| \leq \mathbf{1}$ **for any node** $\boldsymbol{in\ T}$.

What is AVL Tree?

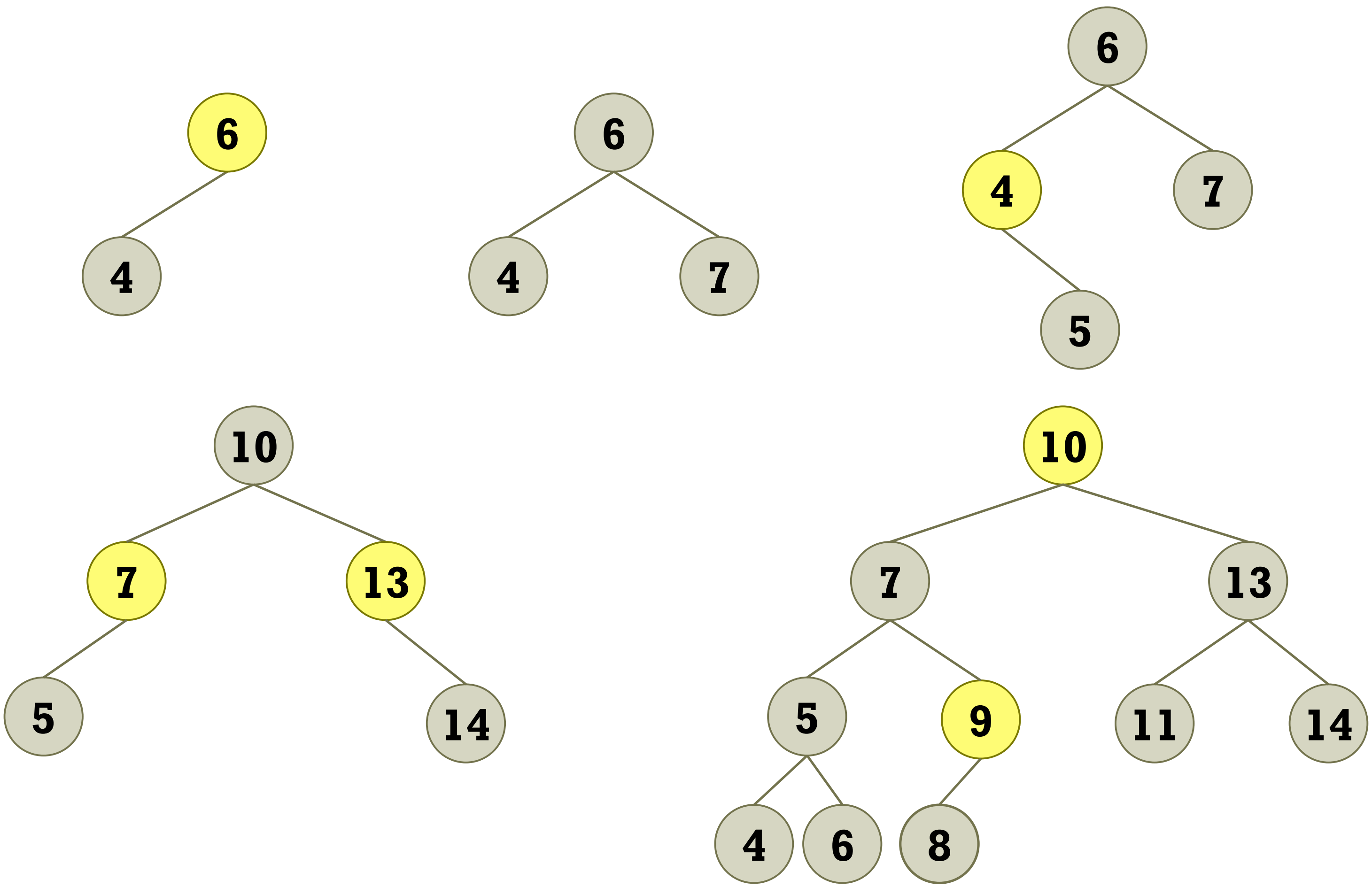
- Definition
  - The balance factor of a node N is defined to be the height difference:
    - $Balance(N) = Height(RightSubtree(N)) - Height\big(LeftSubtree(N)\big)$
    - A binary tree is defined to be an AVL tree if $Balance(N) \in \{-1, 0, +1\}$ holds for every node N in the tree.



Yellow color indicates that the balance factor of the node is ± 1.

What is AVL Tree?

- Examples



Searching in AVL Tree

- Description: This is the same way in the BST.

    1. Begin by examining the root node.

        1.1. If the node is NULL, **the element does not exist**.

    2. Compare the key of the root with the element.

        2.1 If the element is equal to the key, the element is found.

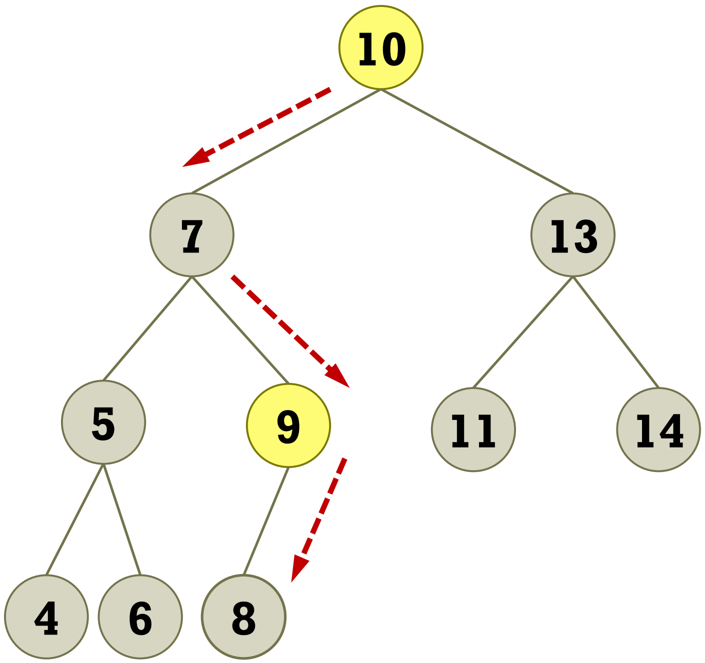
        2.2 If the element is less than the key, **search a left subtree**.

        2.3 If the element is greater than the key, **search a right subtree**.

    3. Repeat steps 1-2 until **the element is found** or **the root is NULL**.

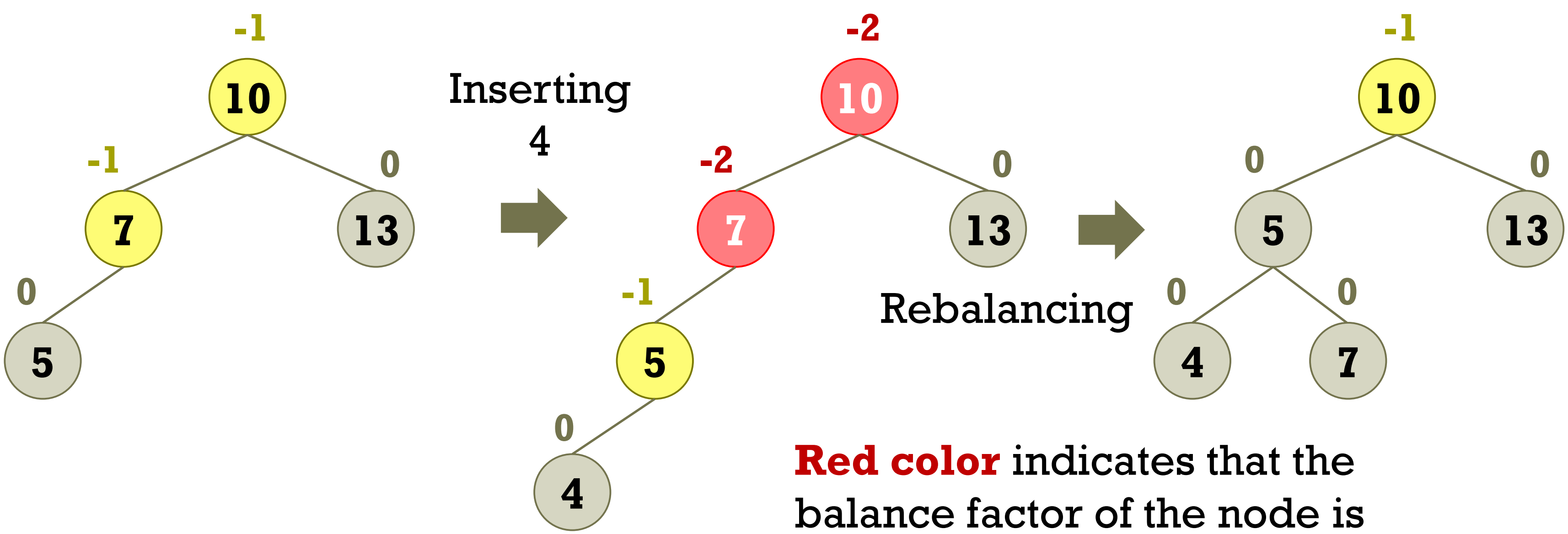
Searching in AVL Tree

- Searching 8 in the AVL Tree
  - Compare 8 with 10. Because 8 < 10, traverse a left subtree.
  - Compare 8 with 7. Because 8 > 7, traverse a right subtree.
  - Compare 8 with 9, Because 8 < 9, traverse a left subtree.



Insertion in AVL Tree

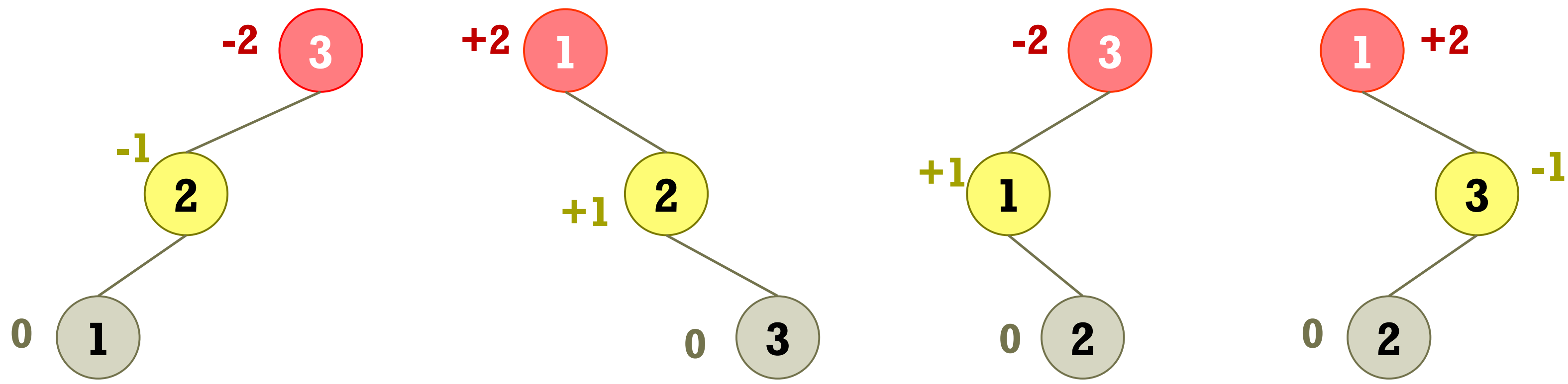
- Description
  - Insert a node in the AVL tree in the same way as the insertion in the BST.
  - Unlike the BST, check the balance factor of every node.
  - If the tree becomes **unbalanced**, the rebalancing operation is performed.



Red color indicates that the balance factor of the node is ± 2.

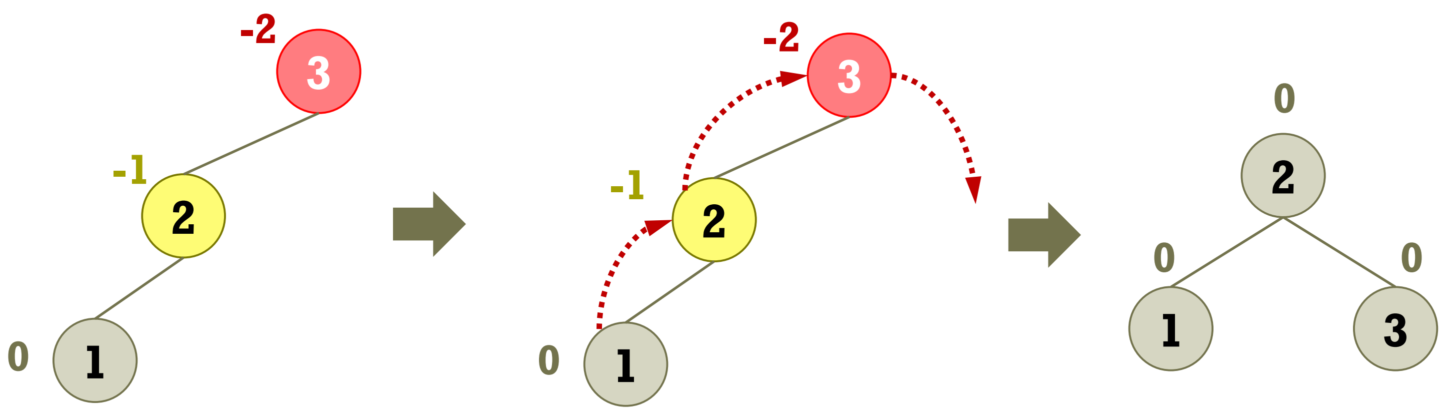
When is Balancing Violated?

- Four possible cases to be unbalanced:
  - Outside cases (requiring **single rotation**):
    - Insertion into **left subtree of left child** of $\boldsymbol{\alpha}$.
    - Insertion into **right subtree of right child** of $\boldsymbol{\alpha}$.
  - Inside cases (requiring **double rotation**):
    - Insertion into **right subtree of left child** of $\boldsymbol{\alpha}$.
    - Insertion into **left subtree of right child** of $\boldsymbol{\alpha}$.



Right-Right (RR) Rotation

- Insertion into left subtree of left child of α



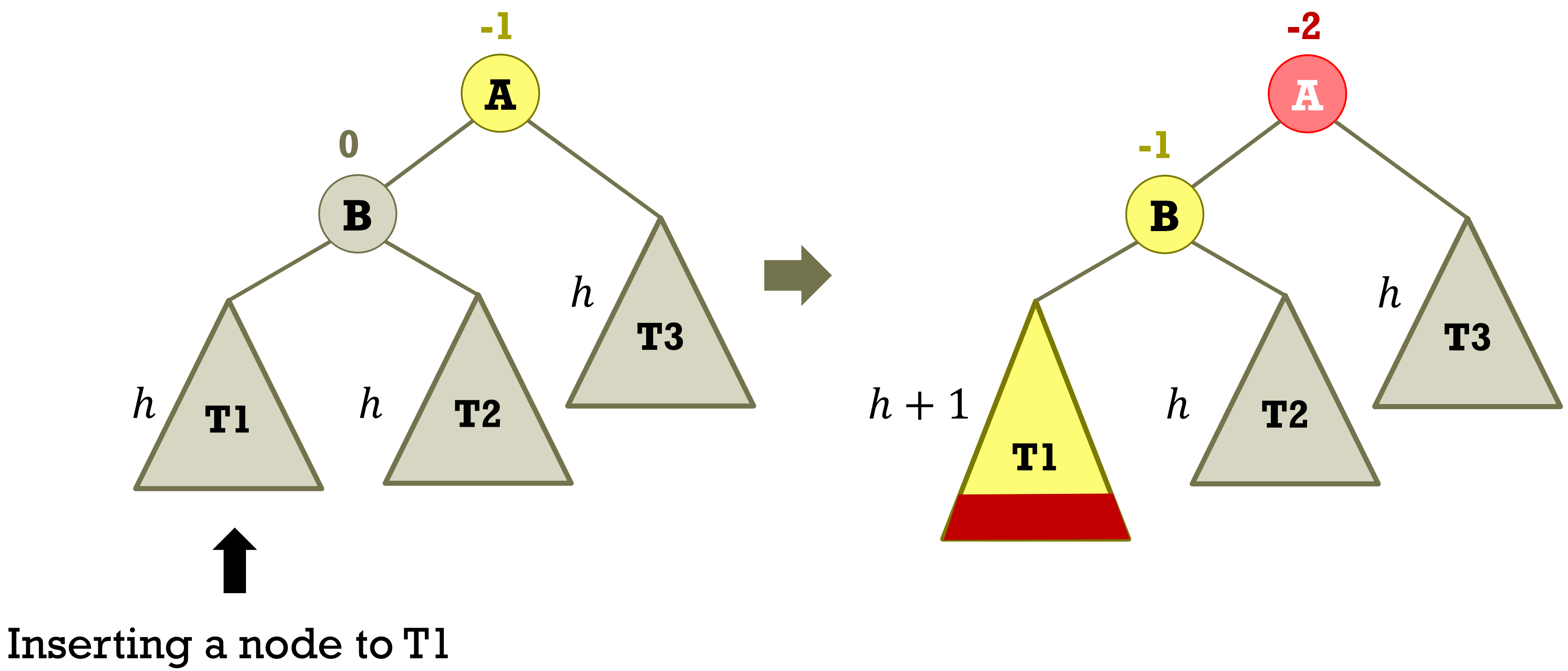
Tree is imbalanced.

To make balanced, we use right-right rotation which moves node one position to right.

After right-right rotation, tree is balanced.

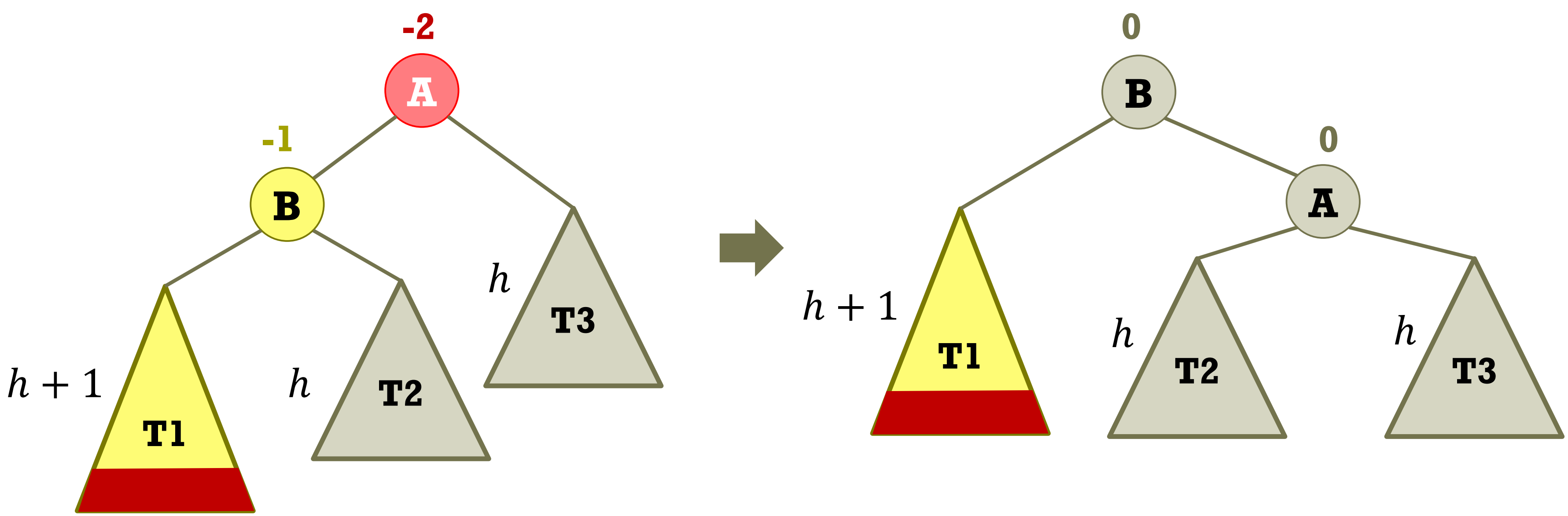
Right-Right (RR) Rotation

- Two nodes A and B is rotated to the right from the current position.

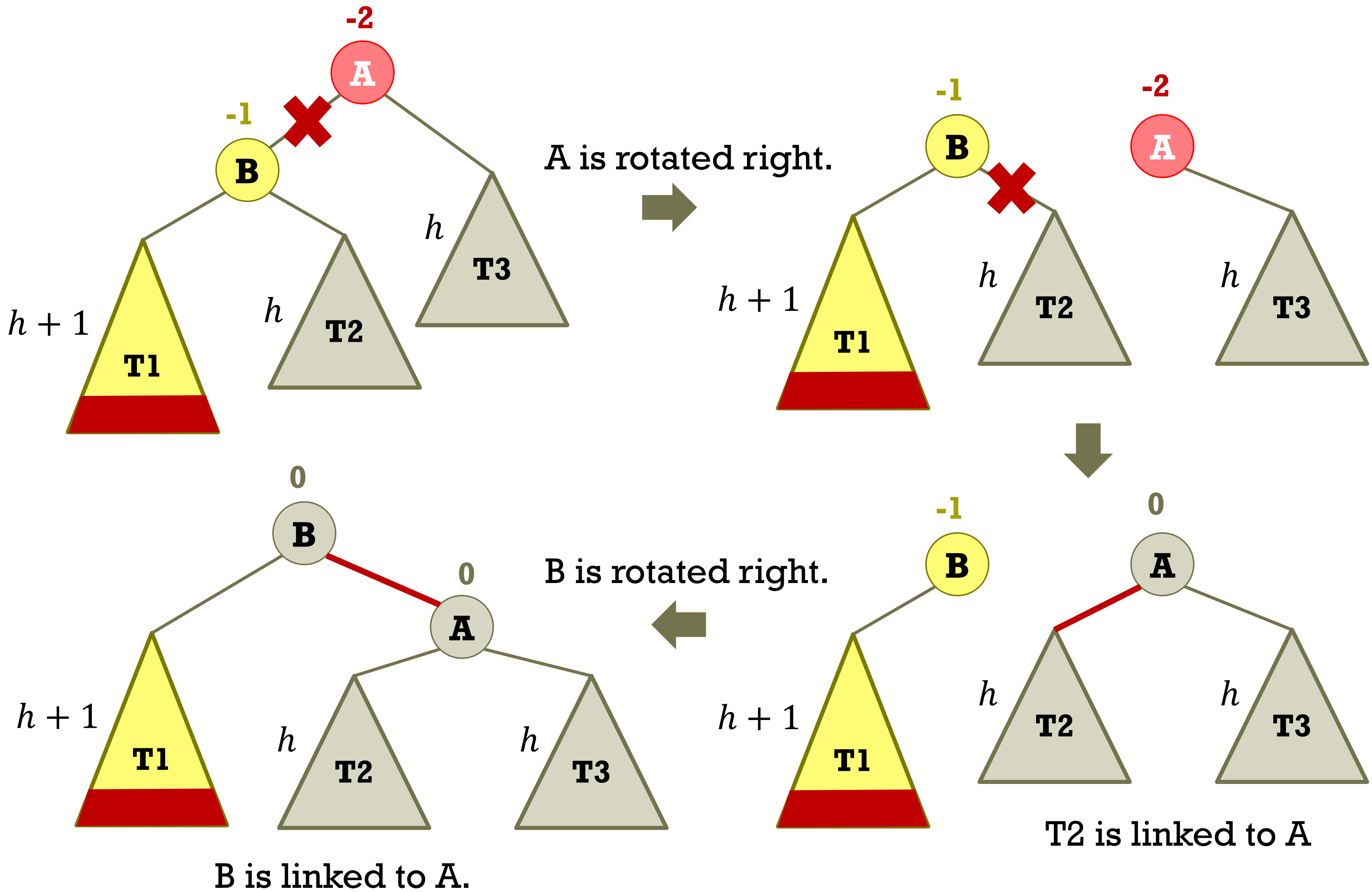


Right-Right (RR) Rotation

- Description: Rotate A and B to the right.
  - To rotate A, the link of its left subtree is disconnected.
  - To rotate B, the link of its right subtree is disconnected.
  - T2 is linked as the left subtree of A.
  - A is linked as the right subtree of B.

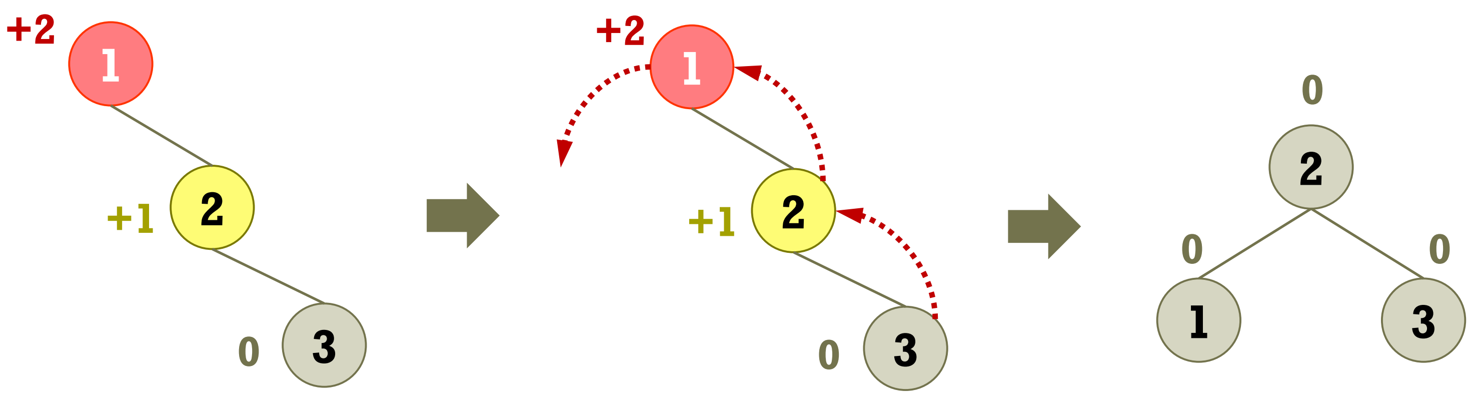


Right-Right (RR) Rotation



Left-Left (LL) Rotation

- Insertion into right subtree of right child of α



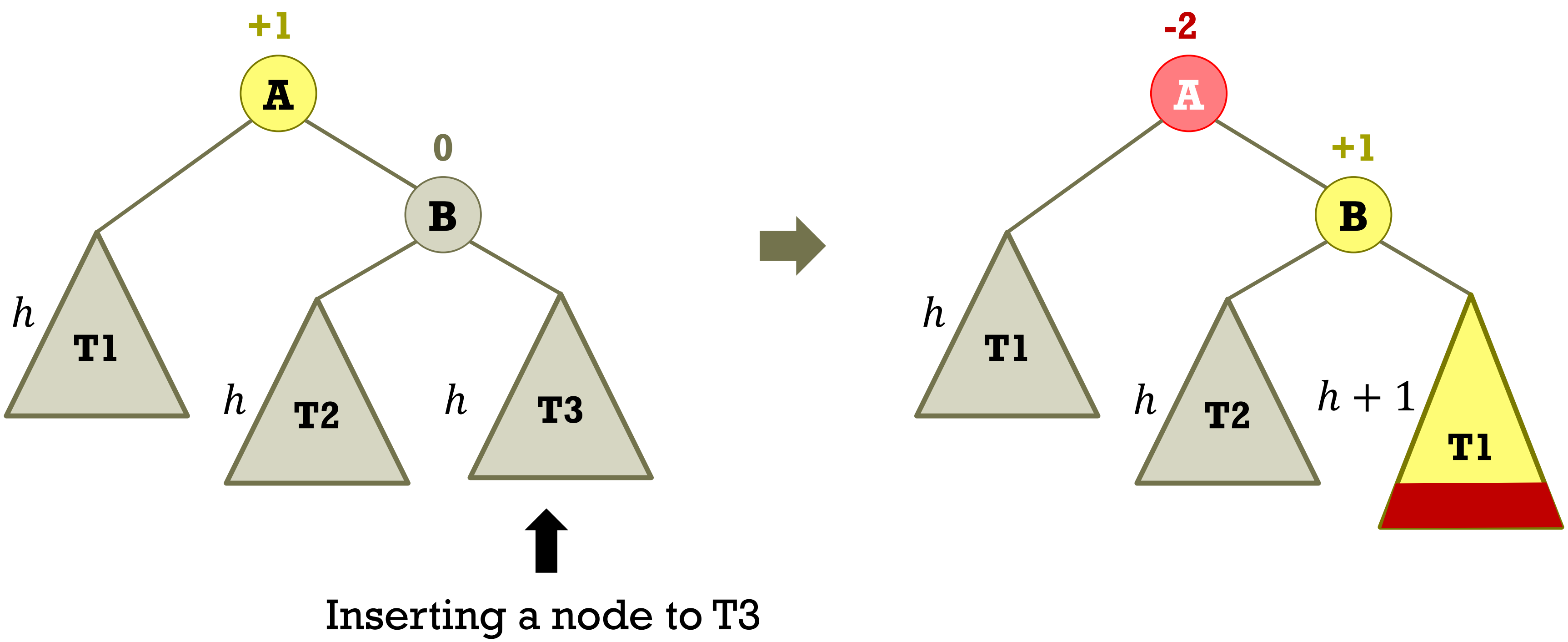
Tree is imbalanced.

To make balanced, we use left-left rotation which moves node one position to left.

After left-left rotation, tree is balanced.

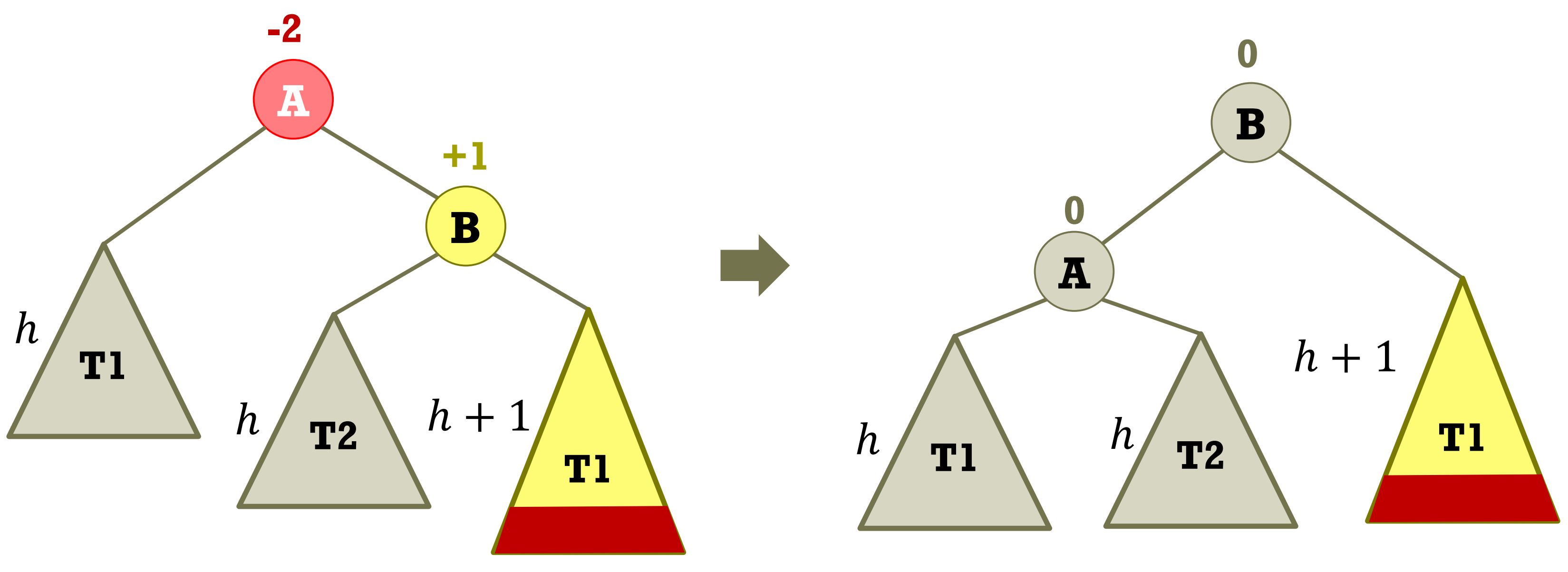
Left-Left (LL) Rotation

- Two nodes A and B is rotated to the left from the current position.

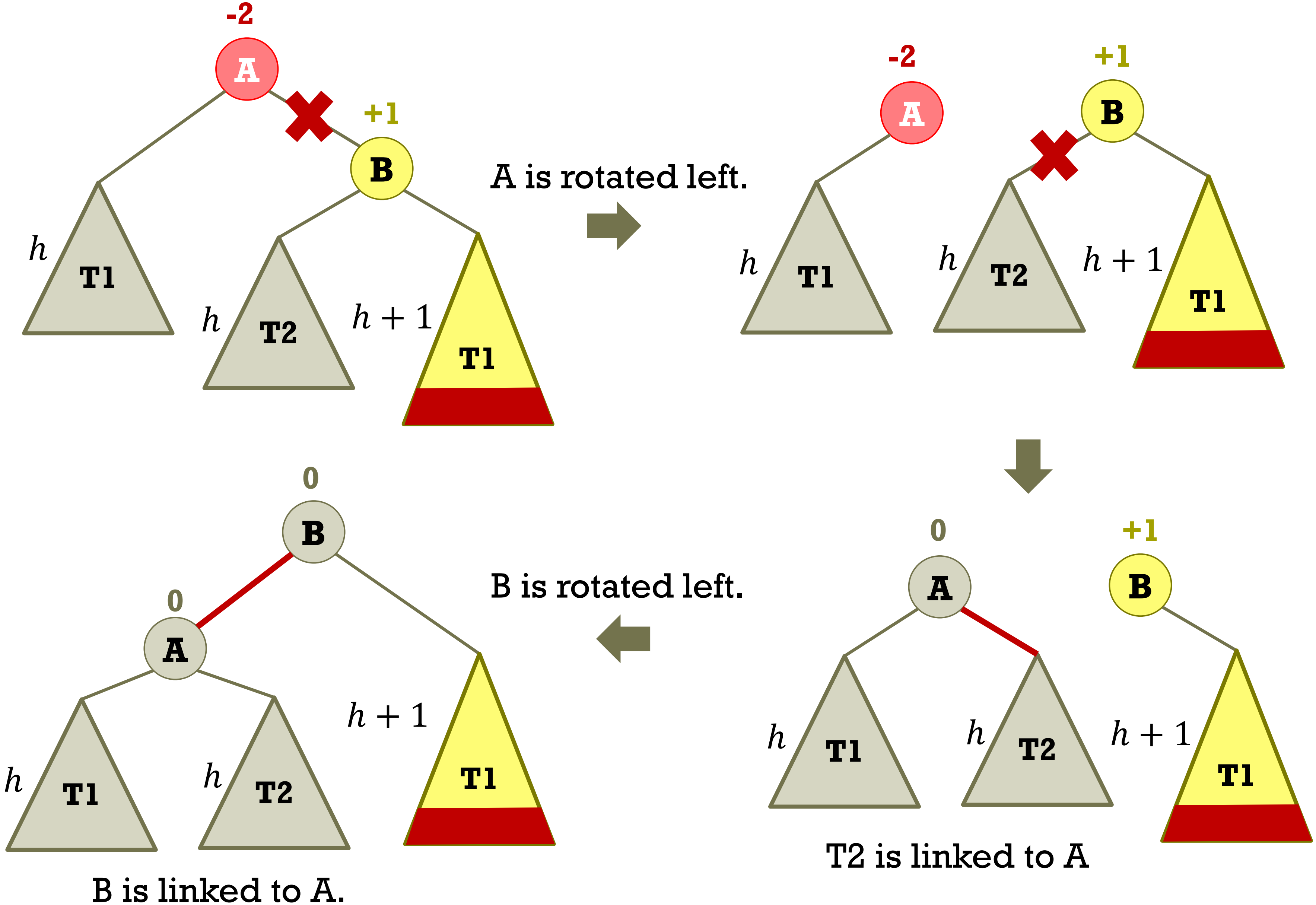


Left-Left (LL) Rotation

- Description: Rotate A and B to the left.
  - To rotate A, the link of its right subtree is disconnected.
  - To rotate B, the link of its left subtree is disconnected.
  - T2 is linked as the right subtree of A.
  - A is linked as the left subtree of B.

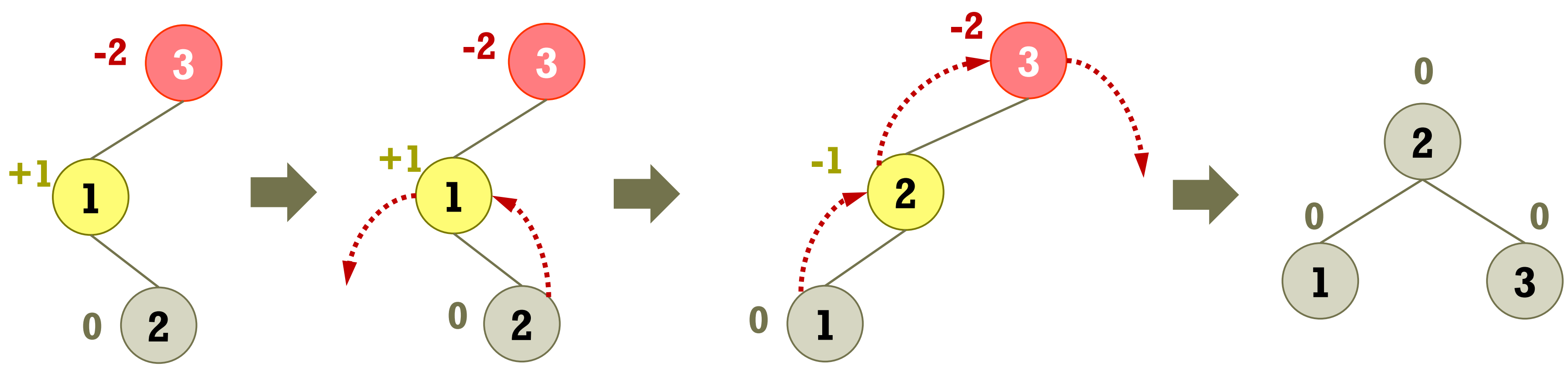


Left-Left (LL) Rotation



Left-Right (LR) Rotation

- Insertion into right subtree of left child of α.



Tree is imbalanced

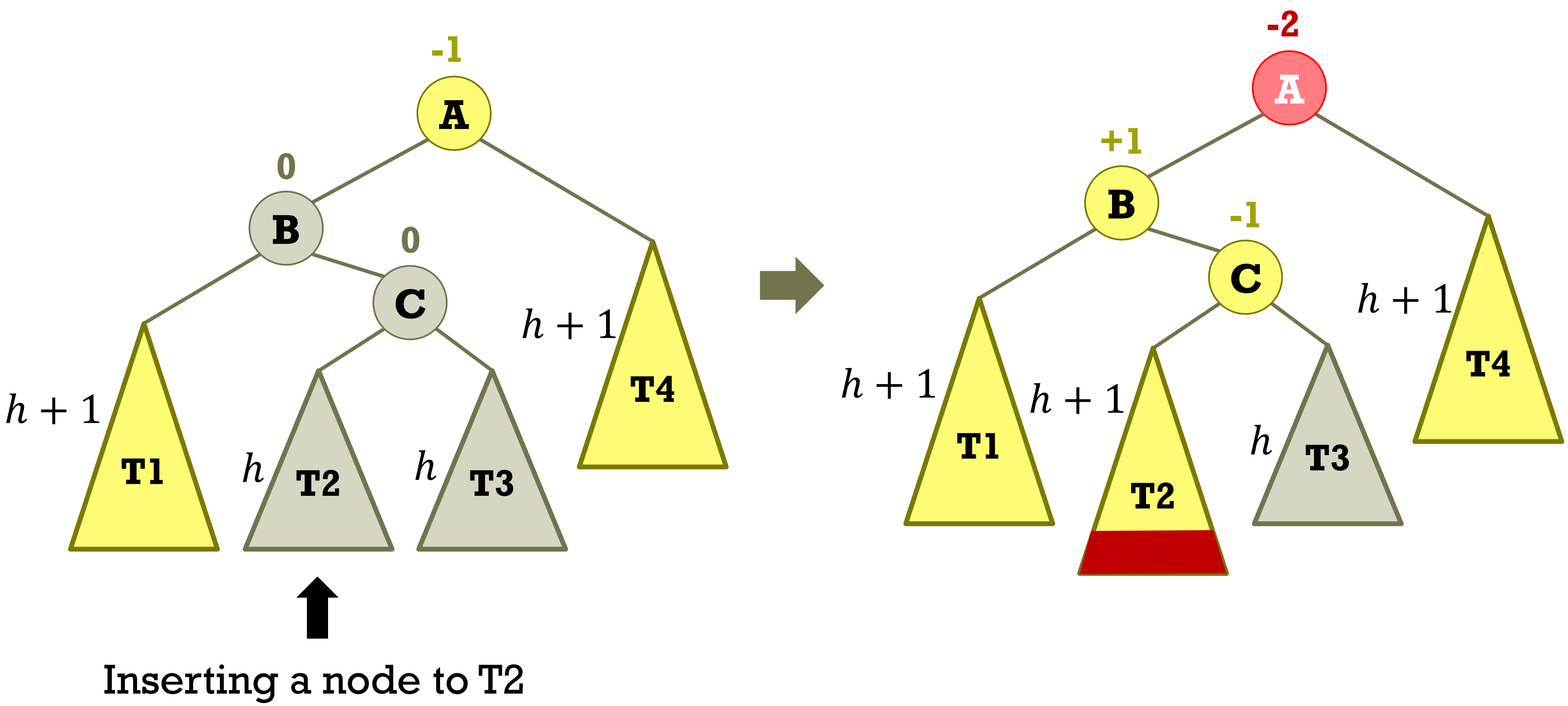
LL rotation

RR rotation

After left-right rotation, tree is balanced.

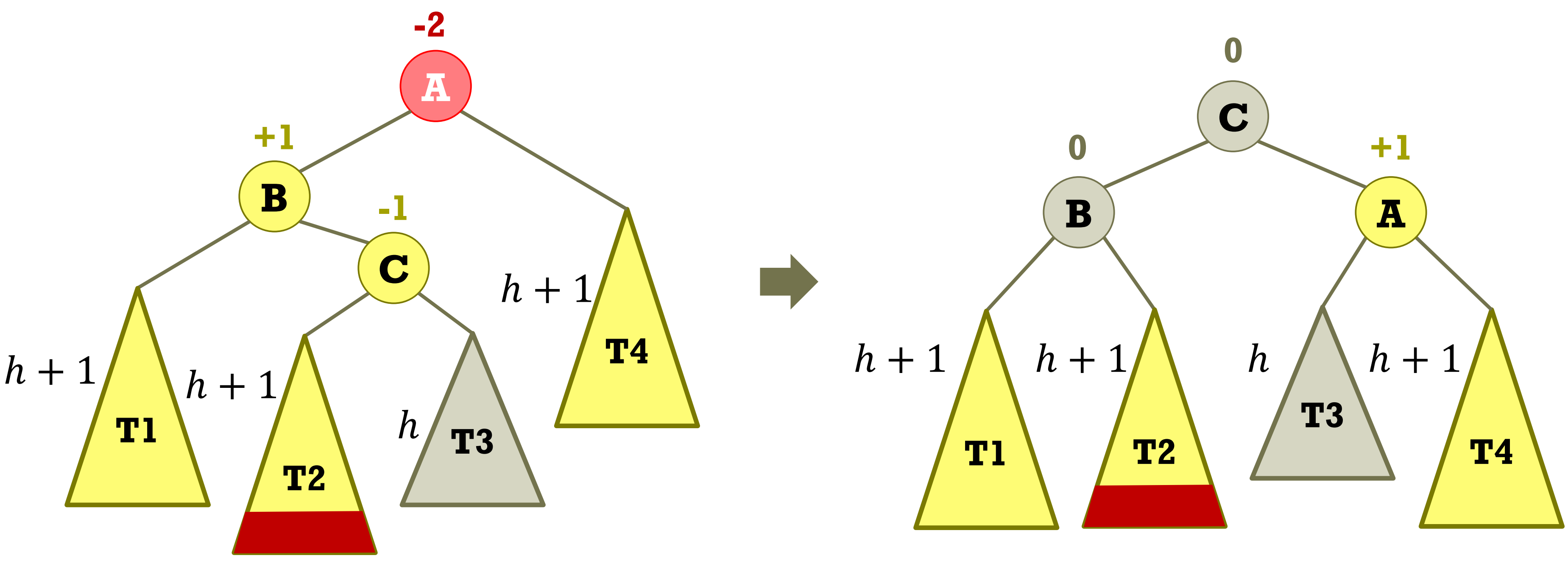
Left-Right (LR) Rotation

- First, rotate B and C to the left from the current position, and then rotate A an C to the right.

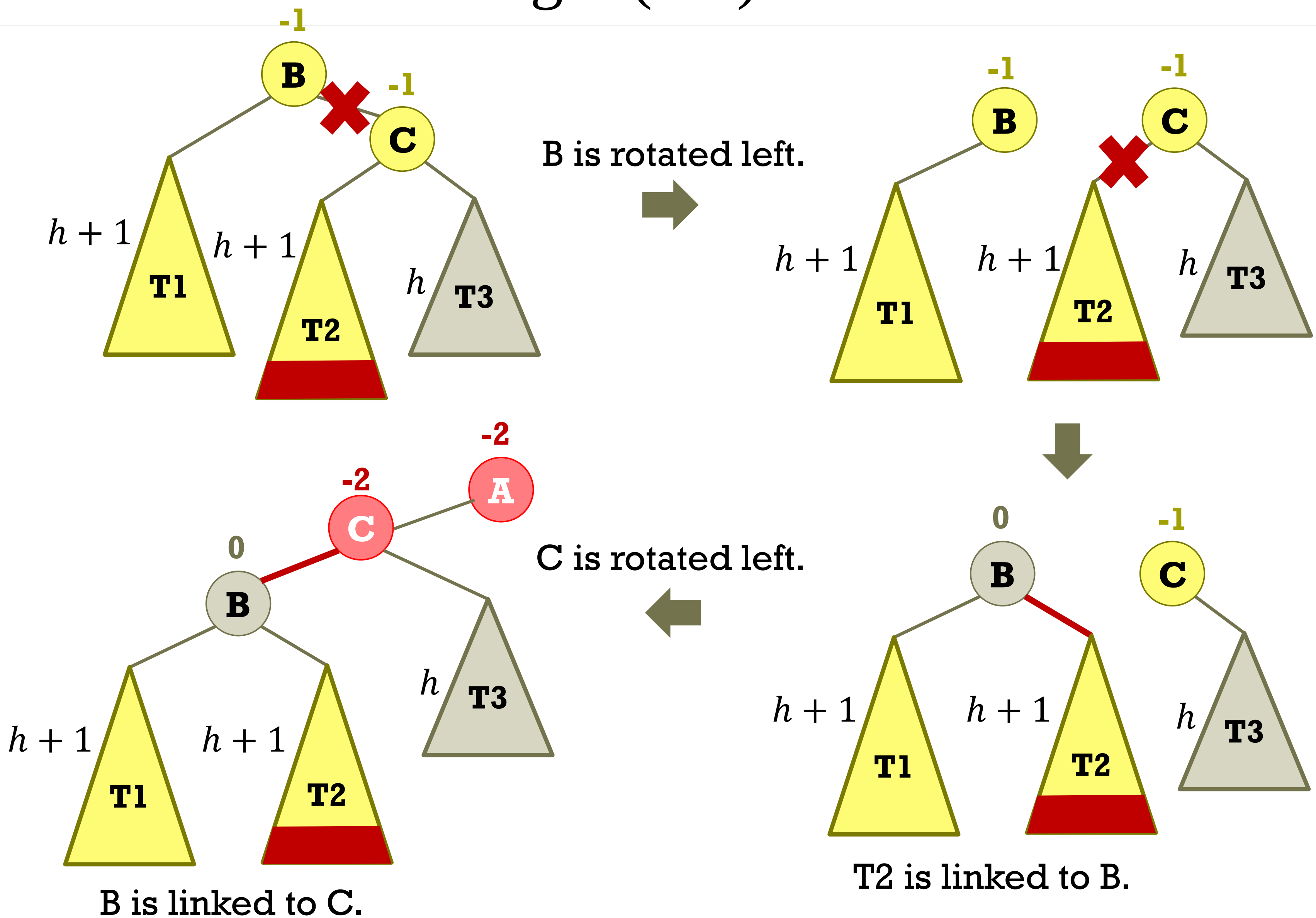


Left-Right (LR) Rotation

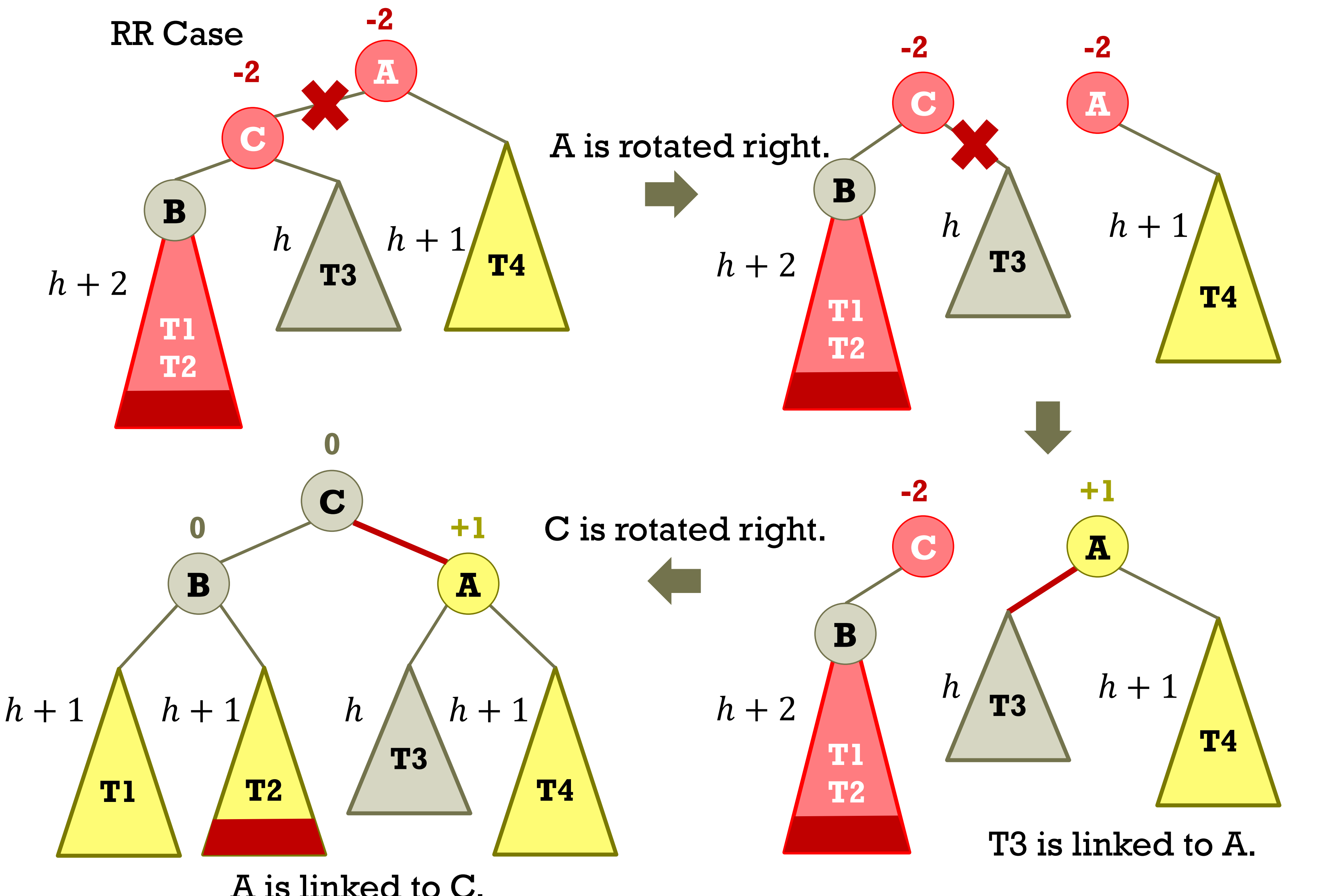
- Description
  - Perform LL rotation for B and C, then you get RR case.
  - Perform RR rotation for A and C.



Left-Right (LR) Rotation

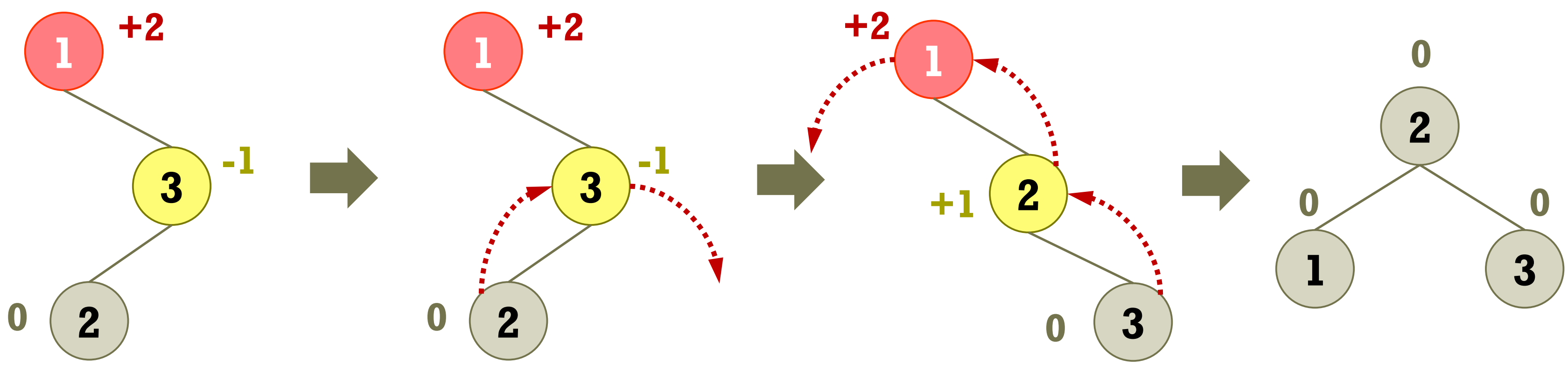


Left-Right (LR) Rotation



Right-Left (RL) Rotation

- Insertion into left subtree of right child of α.



Tree is imbalanced

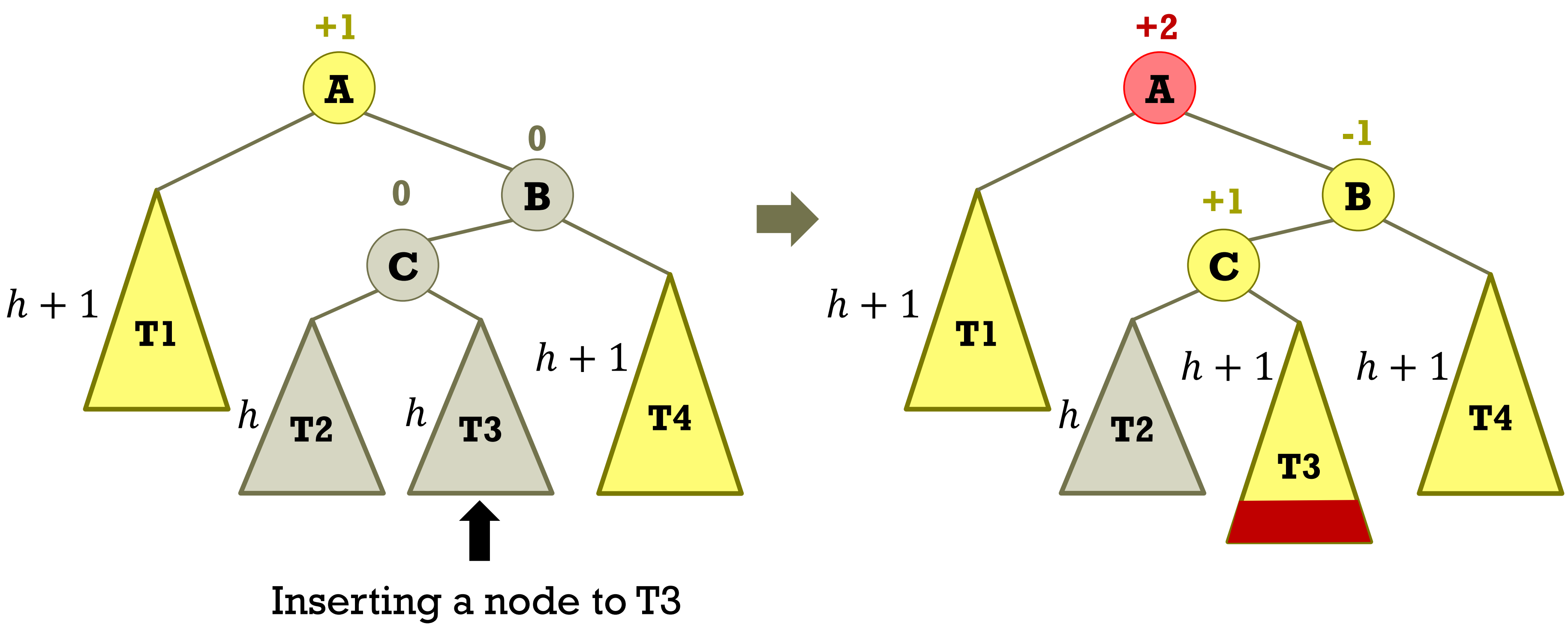
RR rotation

This is an LL problem

After right-left rotation, tree is balanced.

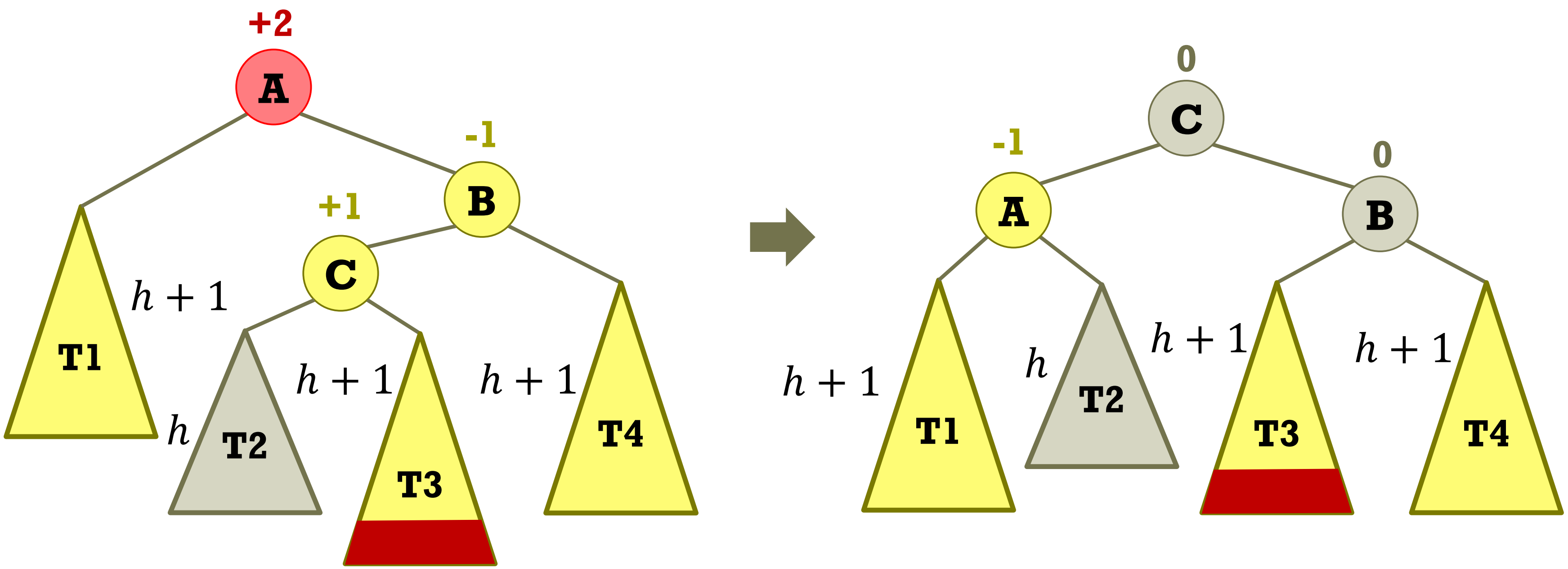
Right-Left (RL) Rotation

- First, rotate B and C to the right from the current position, and then rotate A an C to the left.

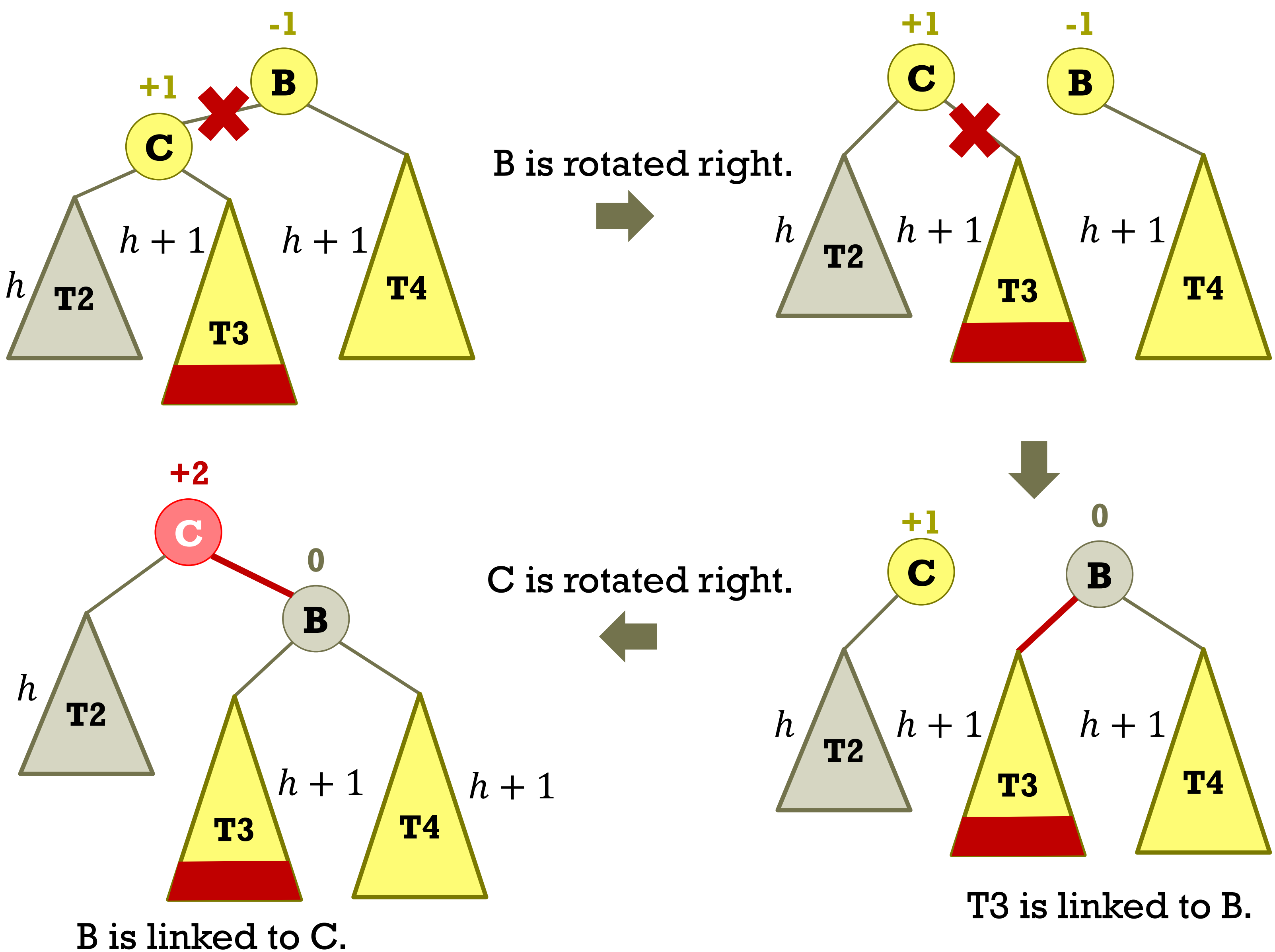


Right-Left (RL) Rotation

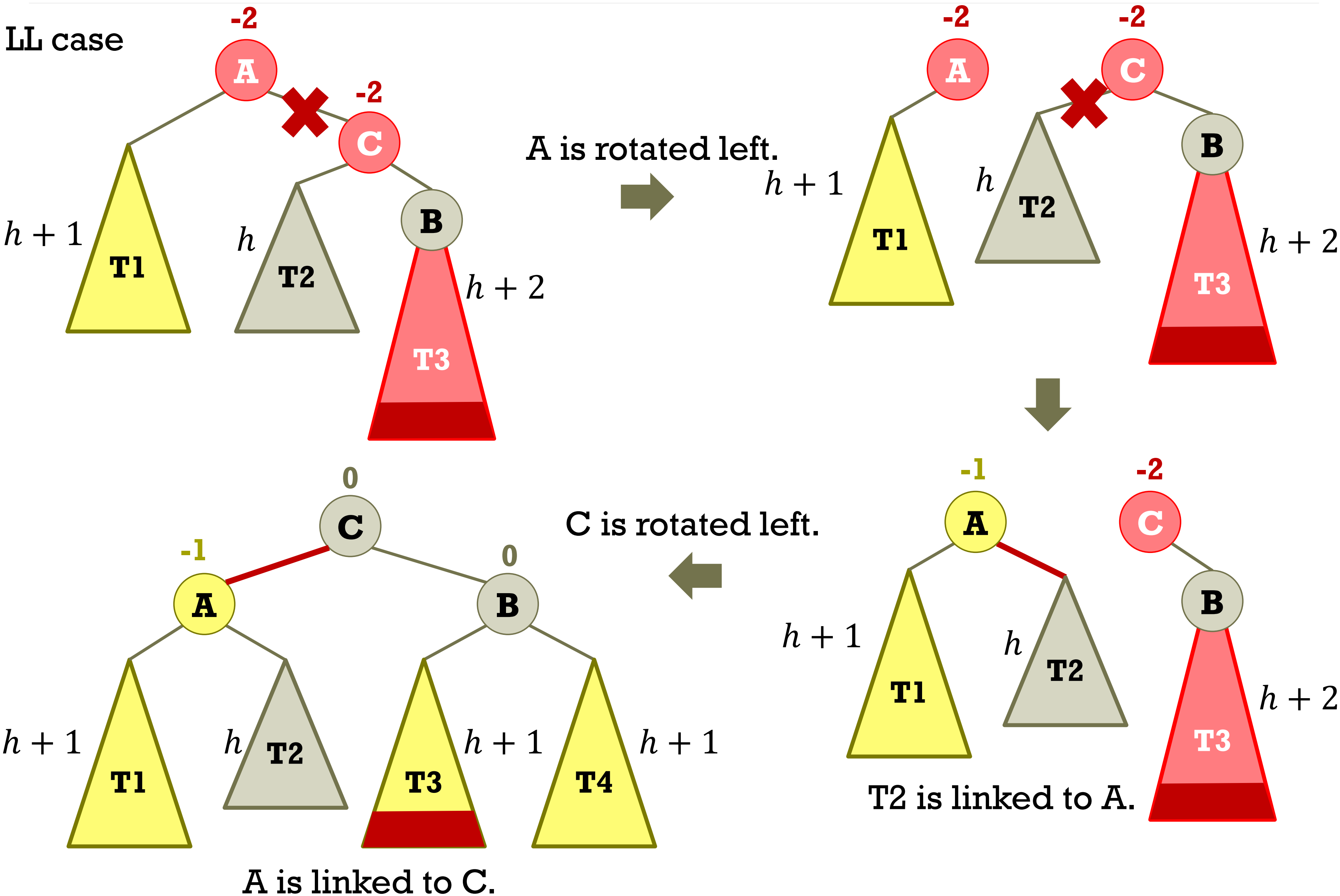
- Description
  - Perform RR rotation for B and C, then you get LL case
  - Perform LL rotation for A and C.



Right-Left (RL) Rotation



Right-Left (RL) Rotation



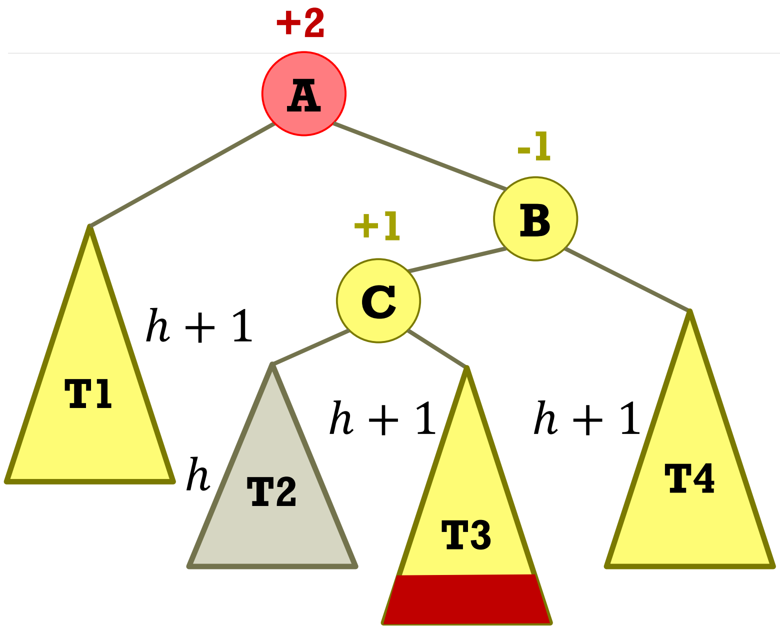
Insertion Example in AVL Tree

- Inserting 1, 2, 3, 4, 5, 6, 7, 8, and 9
  - https://visualgo.net/en/bst
  - https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

Insertion Example in AVL Tree

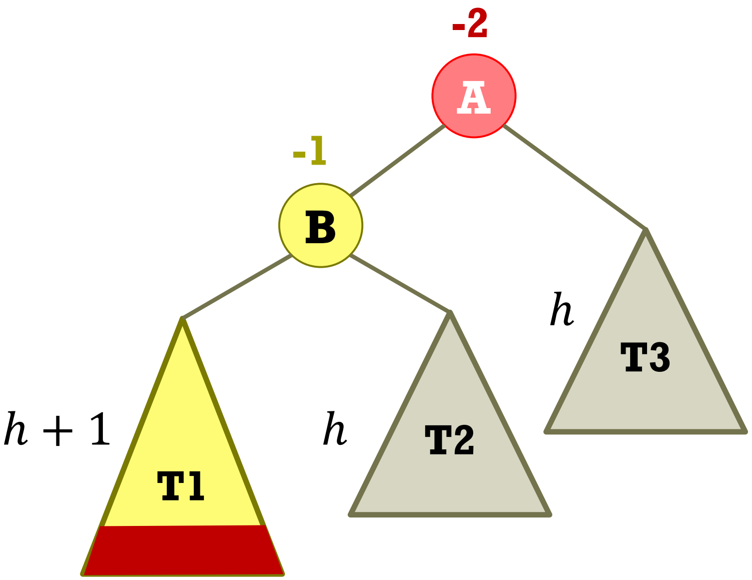
- Inserting 3, 1, 4, 8, 6, 9, 7, and 5
  - https://visualgo.net/en/bst
  - https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

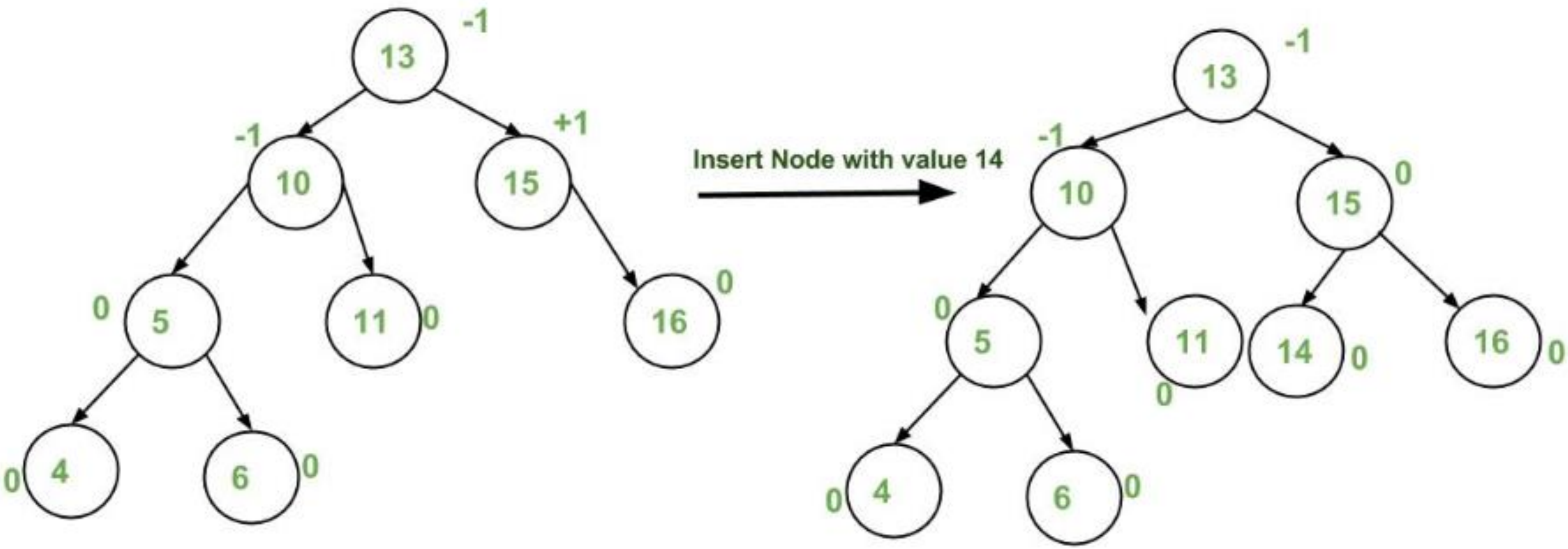
Alternative - RL Case



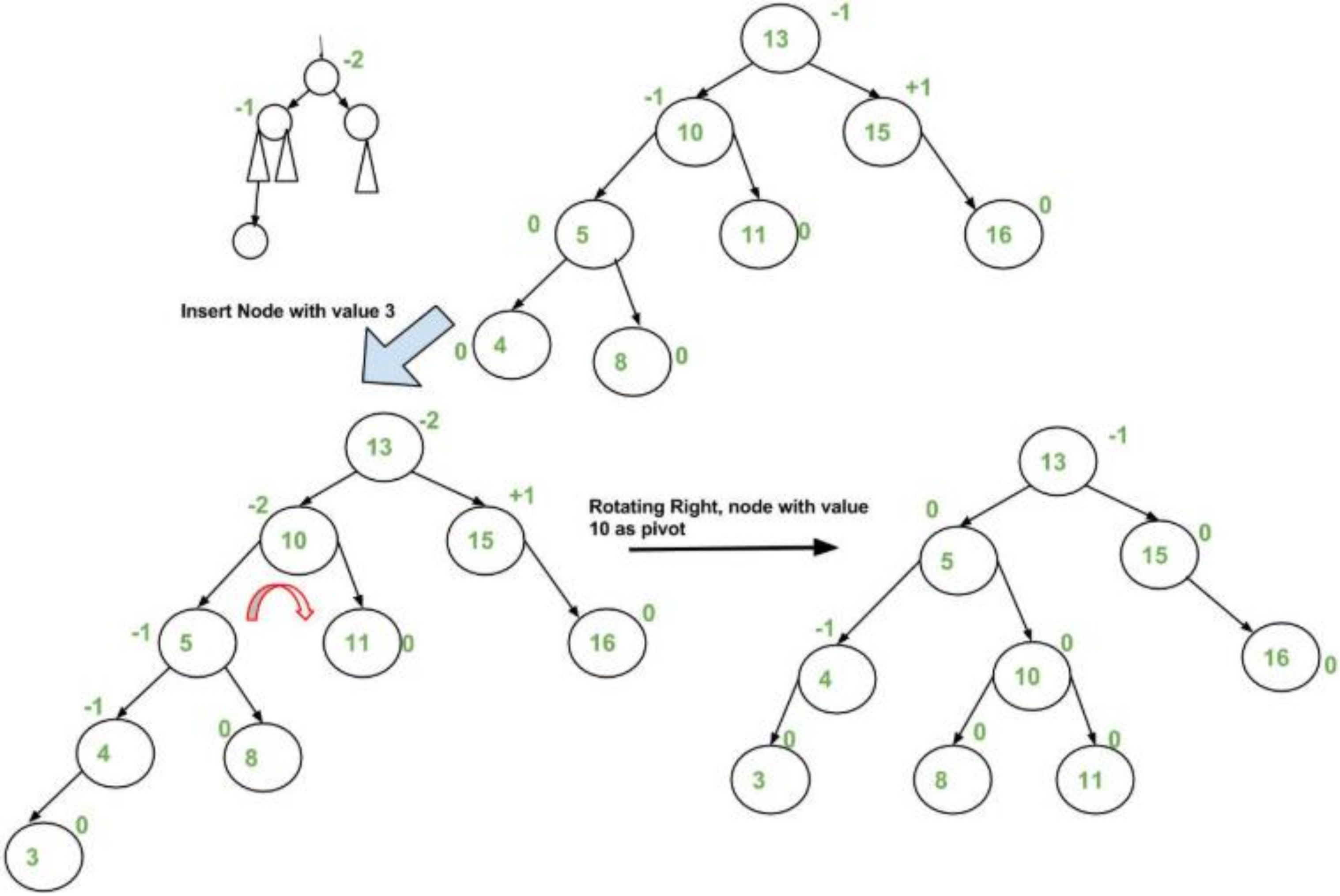
1. Find top three nodes from the unbalancing point, i.e., A, B, C
2. remember the order – T1 < A < T2 < C < T3 < B < T4
3. sort A, B, C
4. attach subtrees

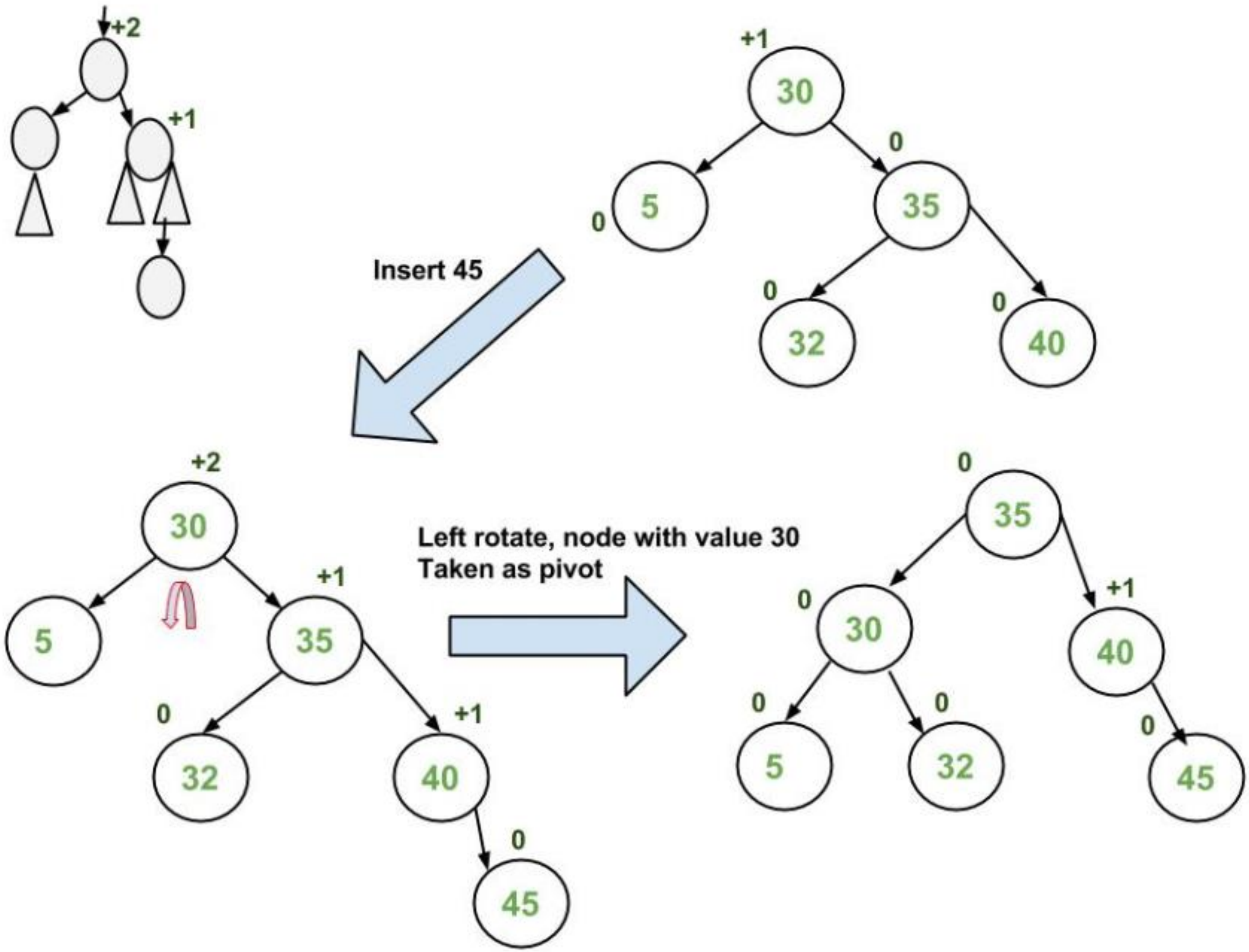
RR Case

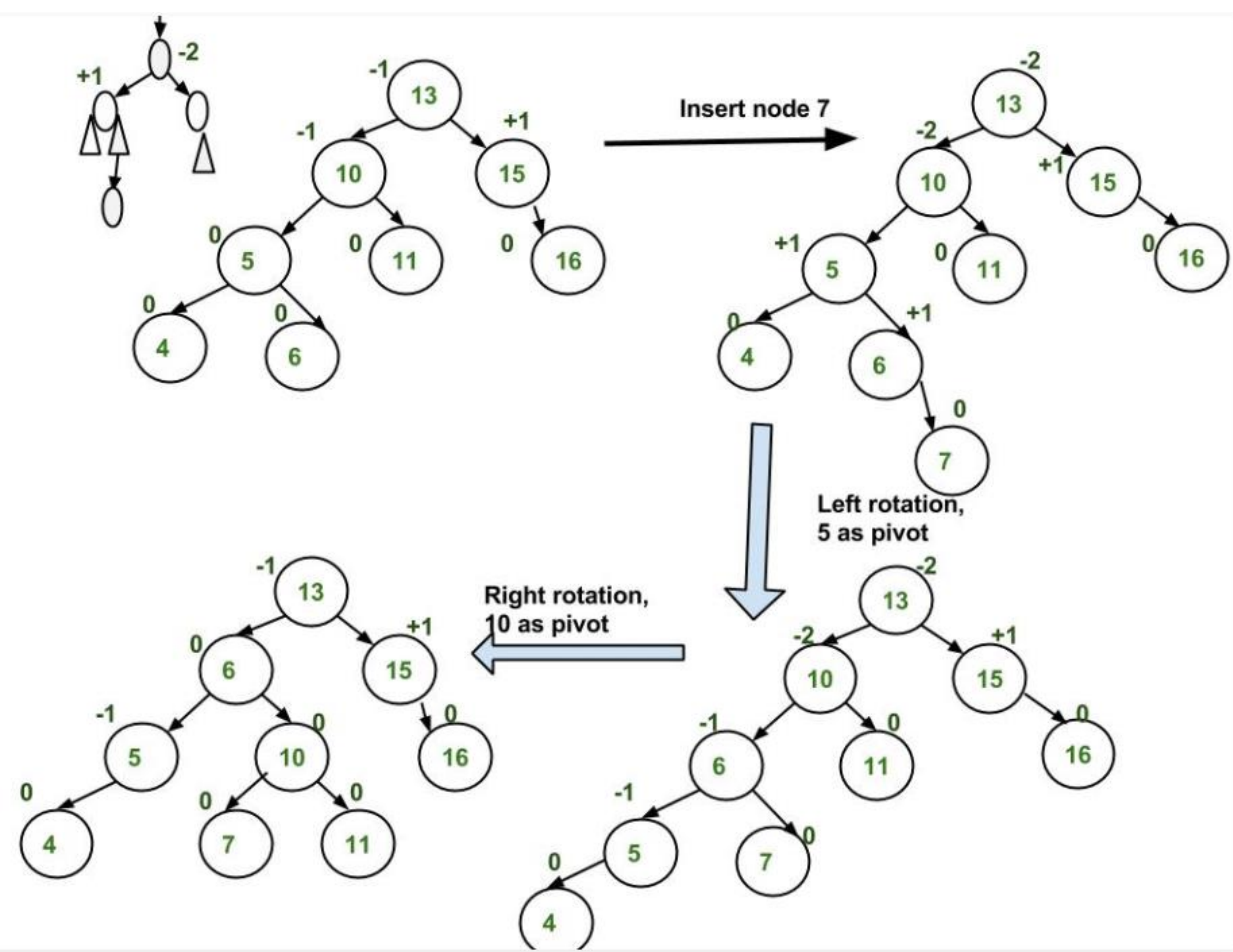


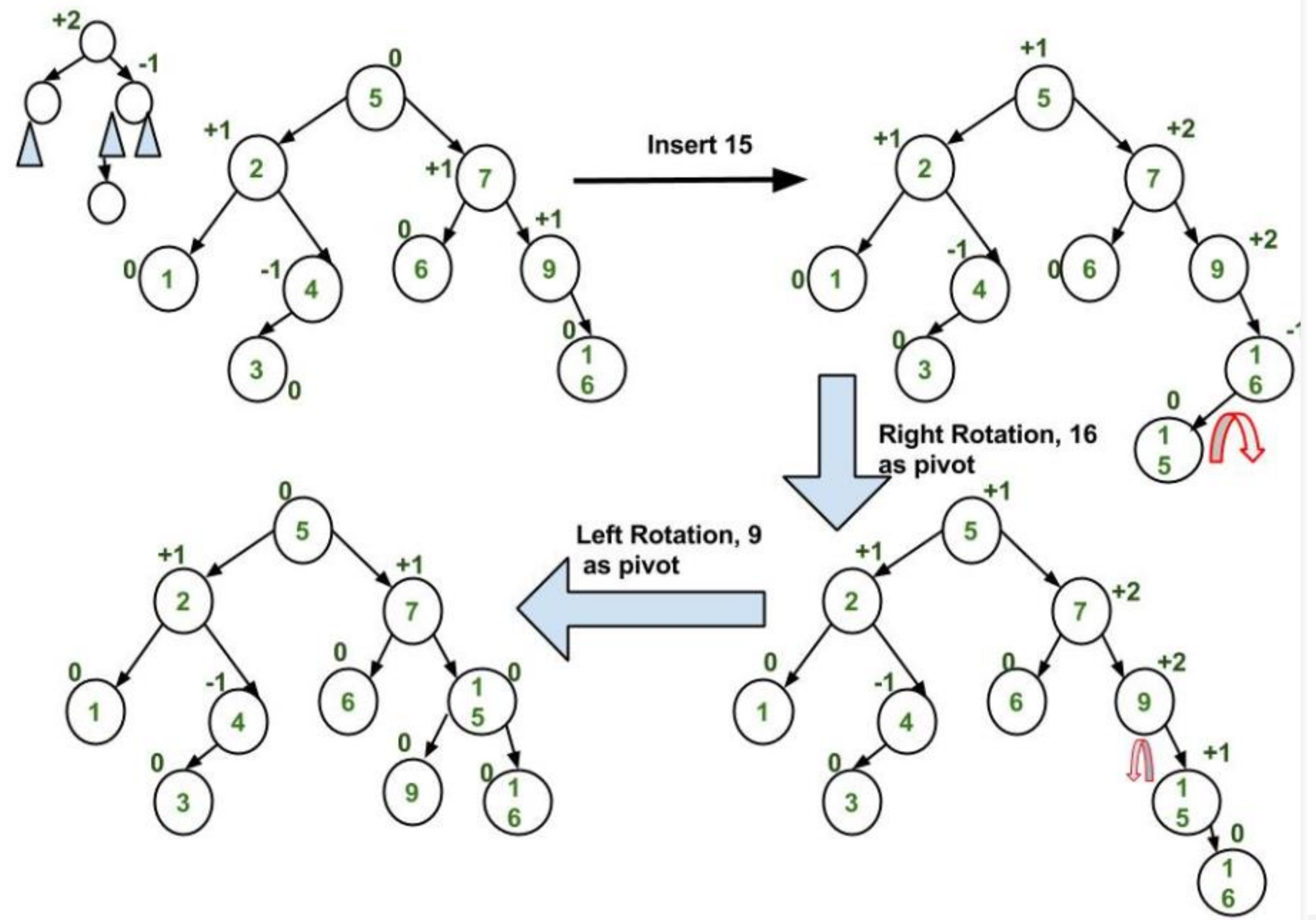


13
-1
10
-1
15
+1
5
0
11
0
16
0
4
0
6
0
Insert Node with value 14
13
-1
10
-1
15
0
5
0
11
0
14
0
16
0
4
0
6
0


Insert Node with value 3
Rotating Right, node with value
10 as pivot

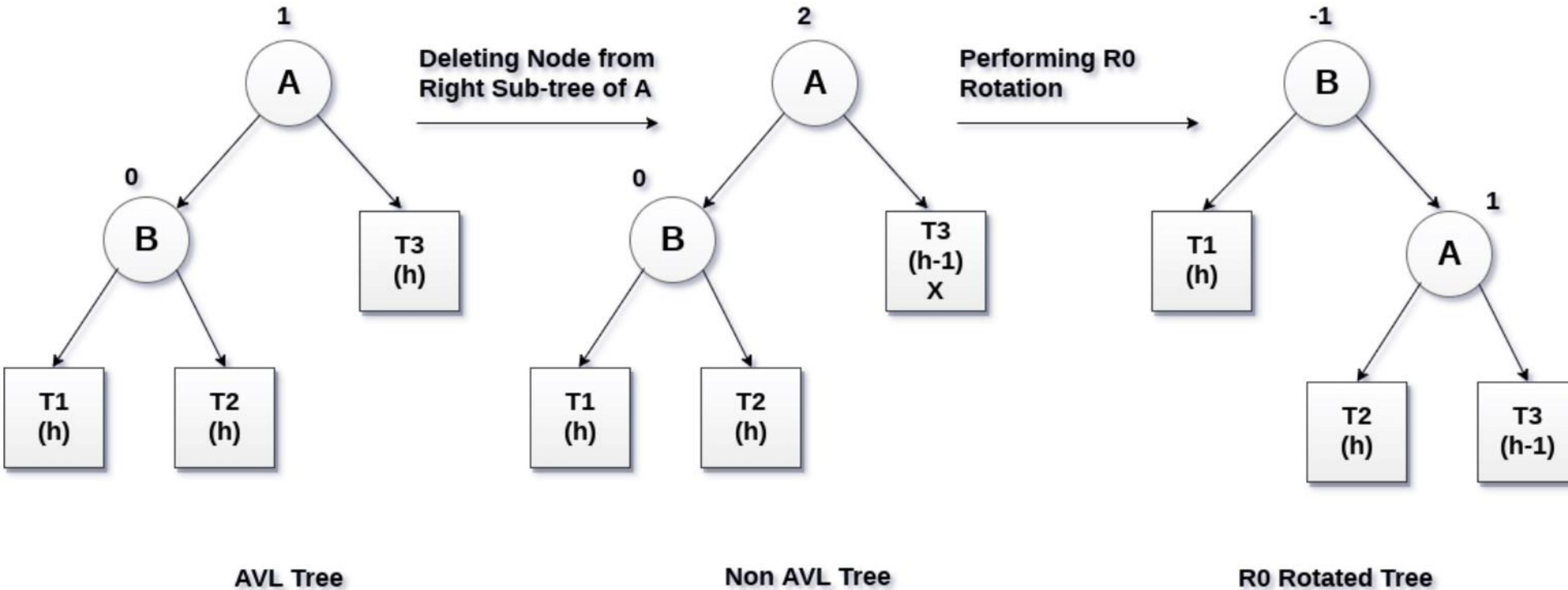

+2
+1
+1
30
0
5
0
35
0
32
0
40
Insert 45
+2
30
Left rotate, node with value 30
Taken as pivot
+1
5
35
0
32
+1
40
0
45
0
35
0
30
+1
40
0
5
0
32
0
45


Insert node 7
Left rotation,
5 as pivot
Right rotation,
10 as pivot


Insert 15
Right Rotation, 16 as pivot
Left Rotation, 9 as pivot

Deletion in AVL Tree – R0 rotation

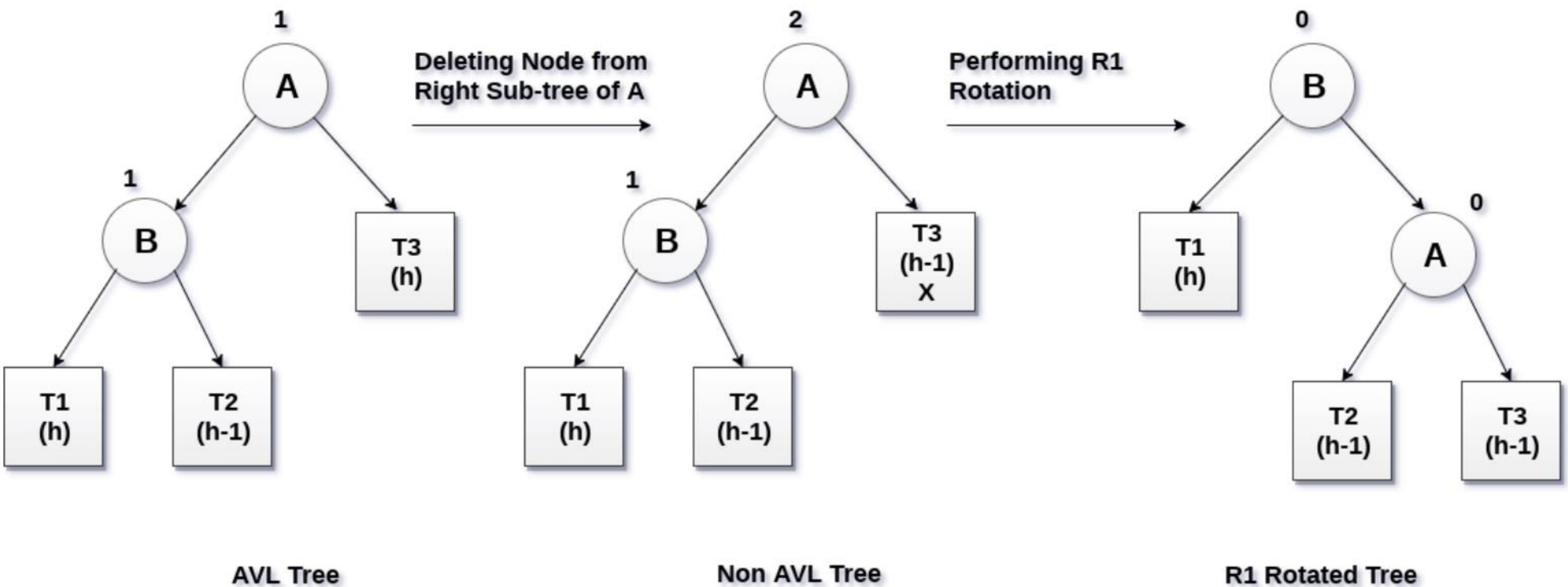
when B has 0 balance factor



This is an RR case when
insertion at T1

Deletion in AVL Tree – R1 rotation

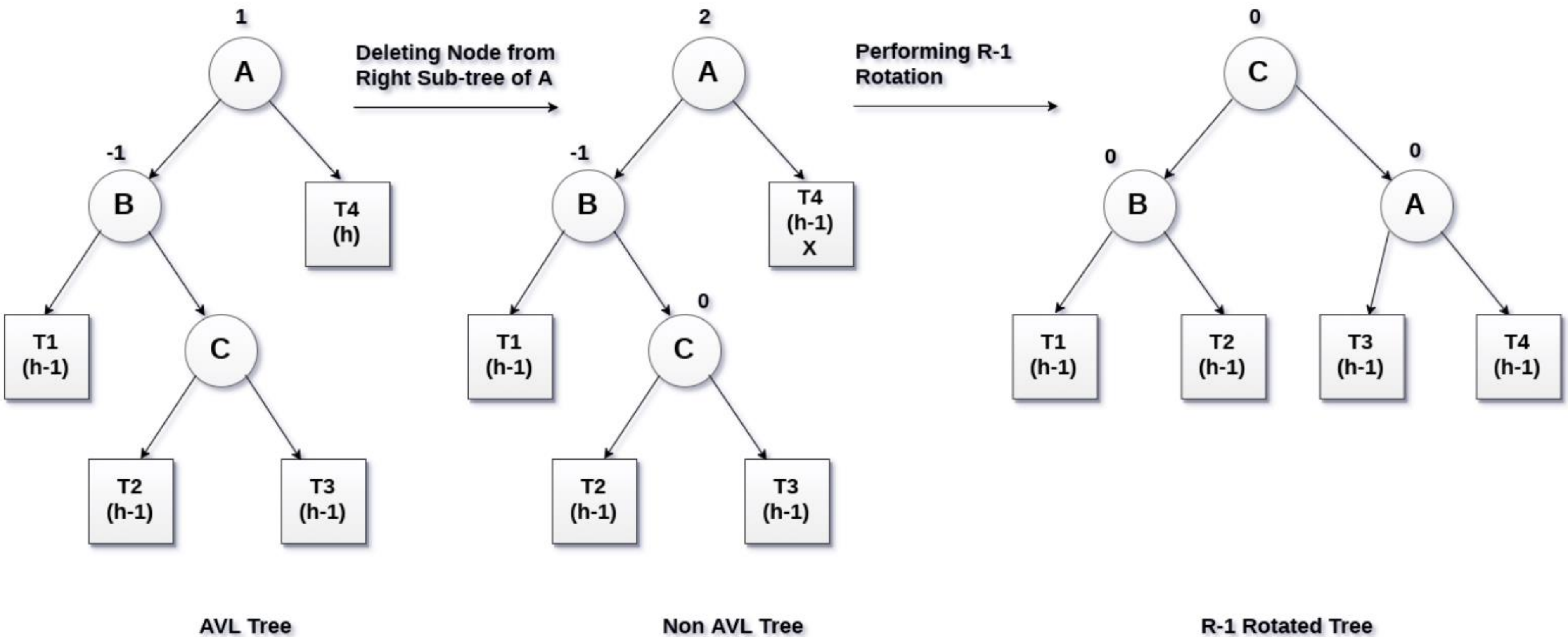
when B has 1 balance factor



This is an RR case when
insertion at T1

Deletion in AVL Tree – R-1 rotation

when B has -1 balance factor



This is an LR case when
insertion at T2

Summary of AVL Tree

- Time complexity of AVL tree
  - The operations for searching, insertion, and deletion are bound by $O(h)$, where h is the height of AVL tree.

Algorithm	Average case	Worst case
Searching	$O(\log n)$	$O(\log n)$
Insertion	$O(\log n)$	$O(\log n)$
Deletion	$O(\log n)$	$O(\log n)$

  - where n is the number of nodes in the AVL tree.