CS350: Data Structures

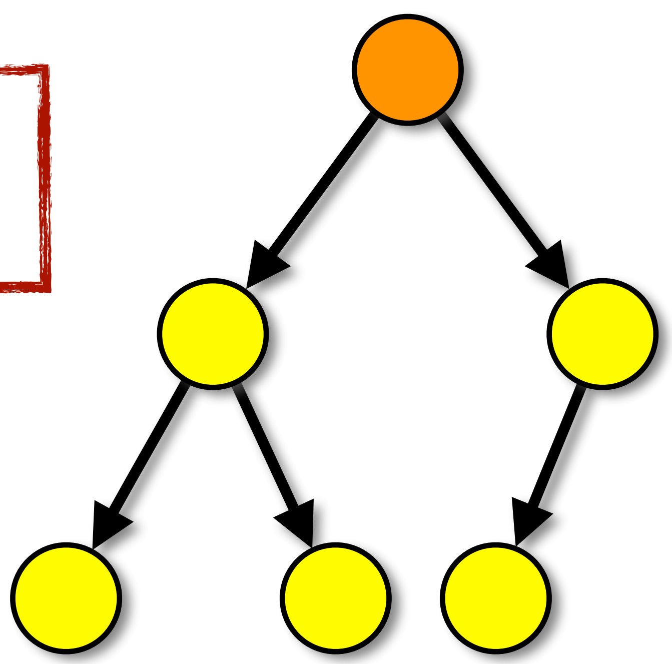
AVL Trees

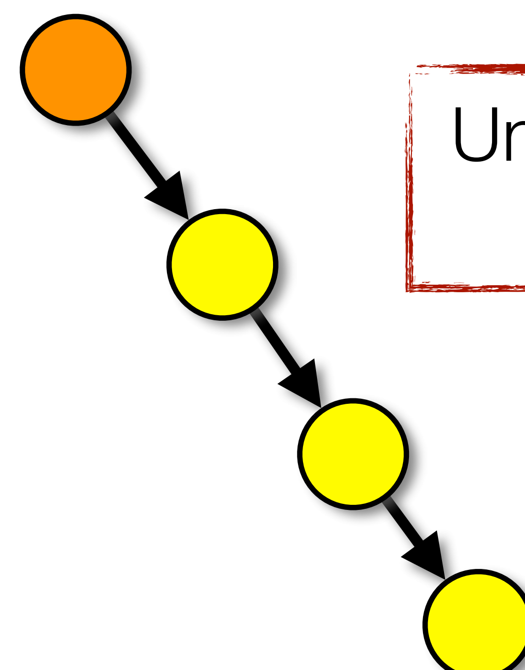
James Moscola
Department of Physical Sciences
York College of Pennsylvania

Balanced Search Trees

- **Binary search trees are not guaranteed to be balanced given random insertions and deletions**
  - Inserting a sorted lists of elements into a BST produces the worst case -- O(N)
  - Performance of an unbalanced tree can degrade as more elements are inserted
- **Balanced search tree operations, such as insert, insure that the a tree always remains balanced**
  - An operation is not complete until it returns the tree to a balanced state

Balanced tree





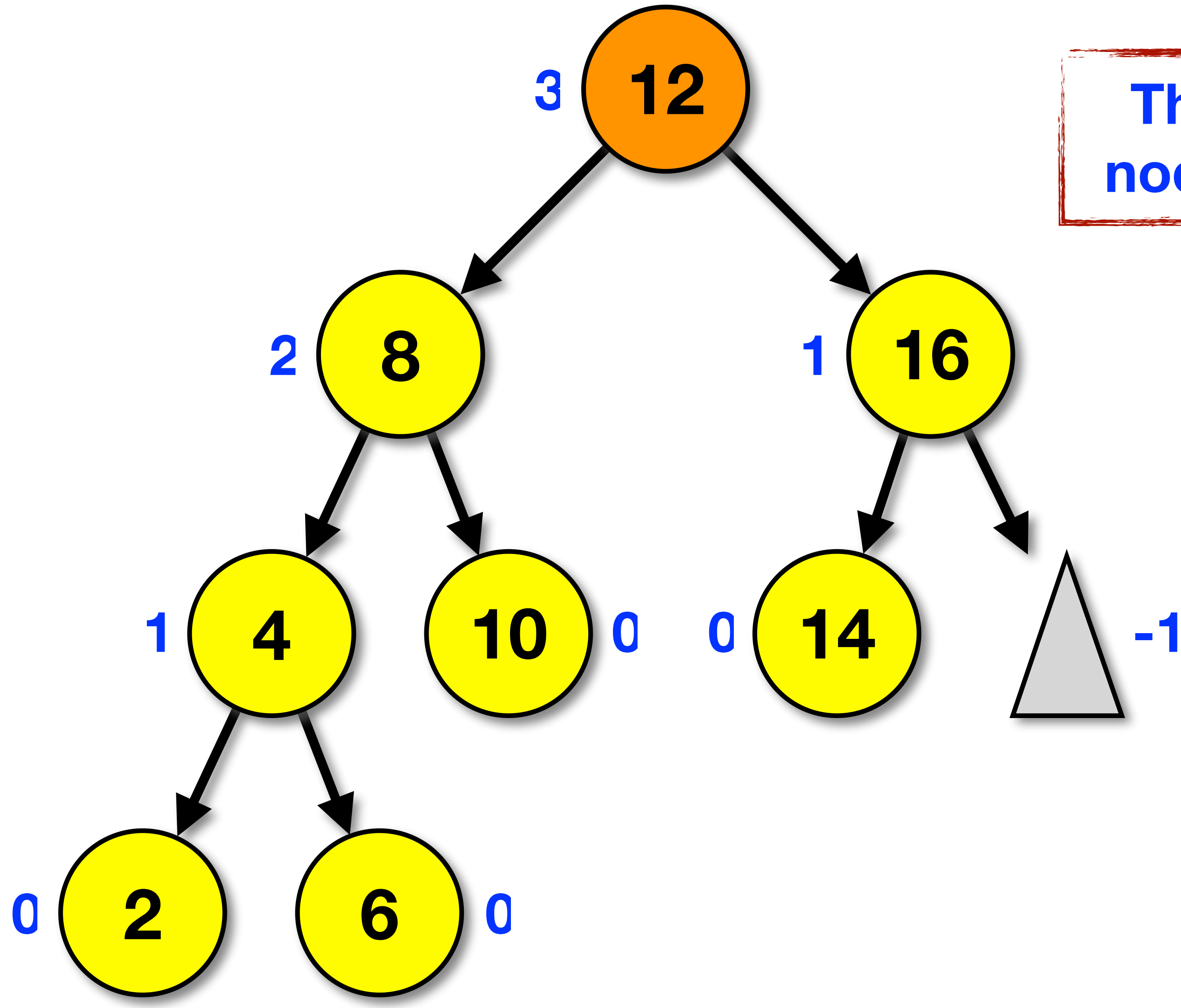
Unbalanced tree

AVL Trees

- **A type of balanced binary search tree**
- **Named for its discoverers -- Adelson, Velskii, and Landis**
- **Definition:**

> An AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right subtrees can differ by at most 1. The height of an empty subtree is -1.

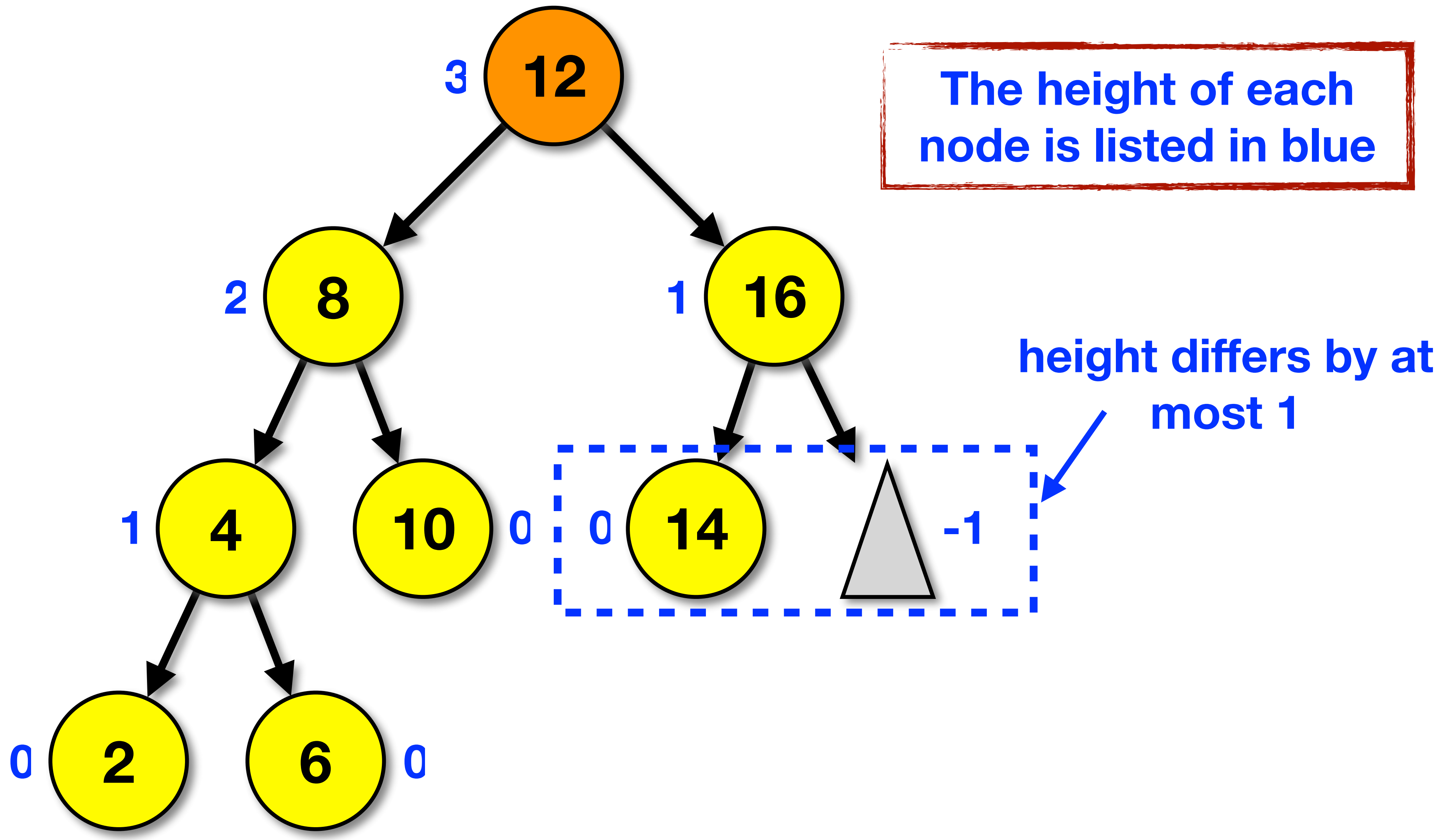
AVL Trees



The height of each node is listed in blue

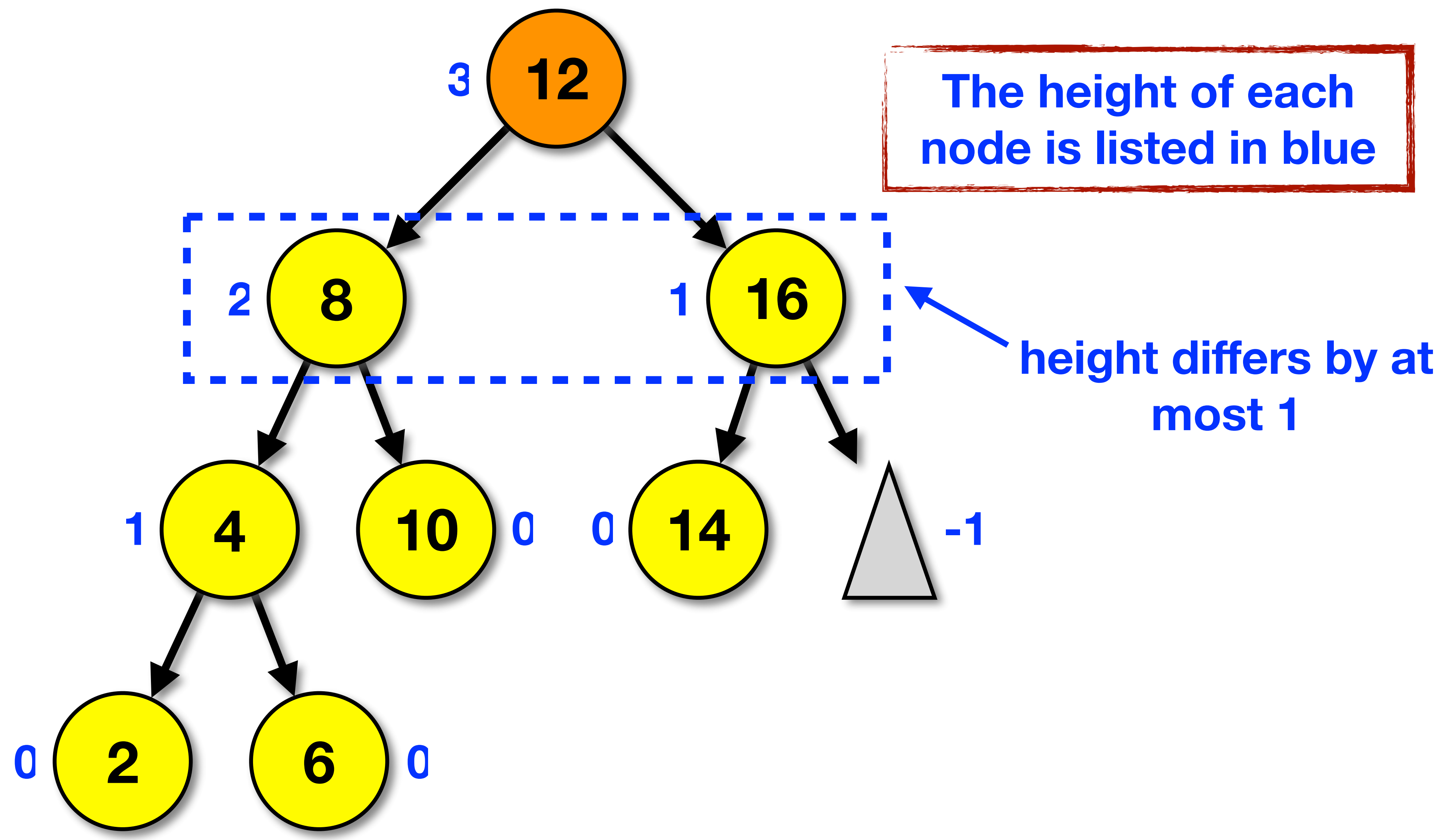
An AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right subtrees can differ by at most 1. The height of an empty subtree is -1.

AVL Trees



An AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right subtrees can differ by at most 1. The height of an empty subtree is -1.

AVL Trees

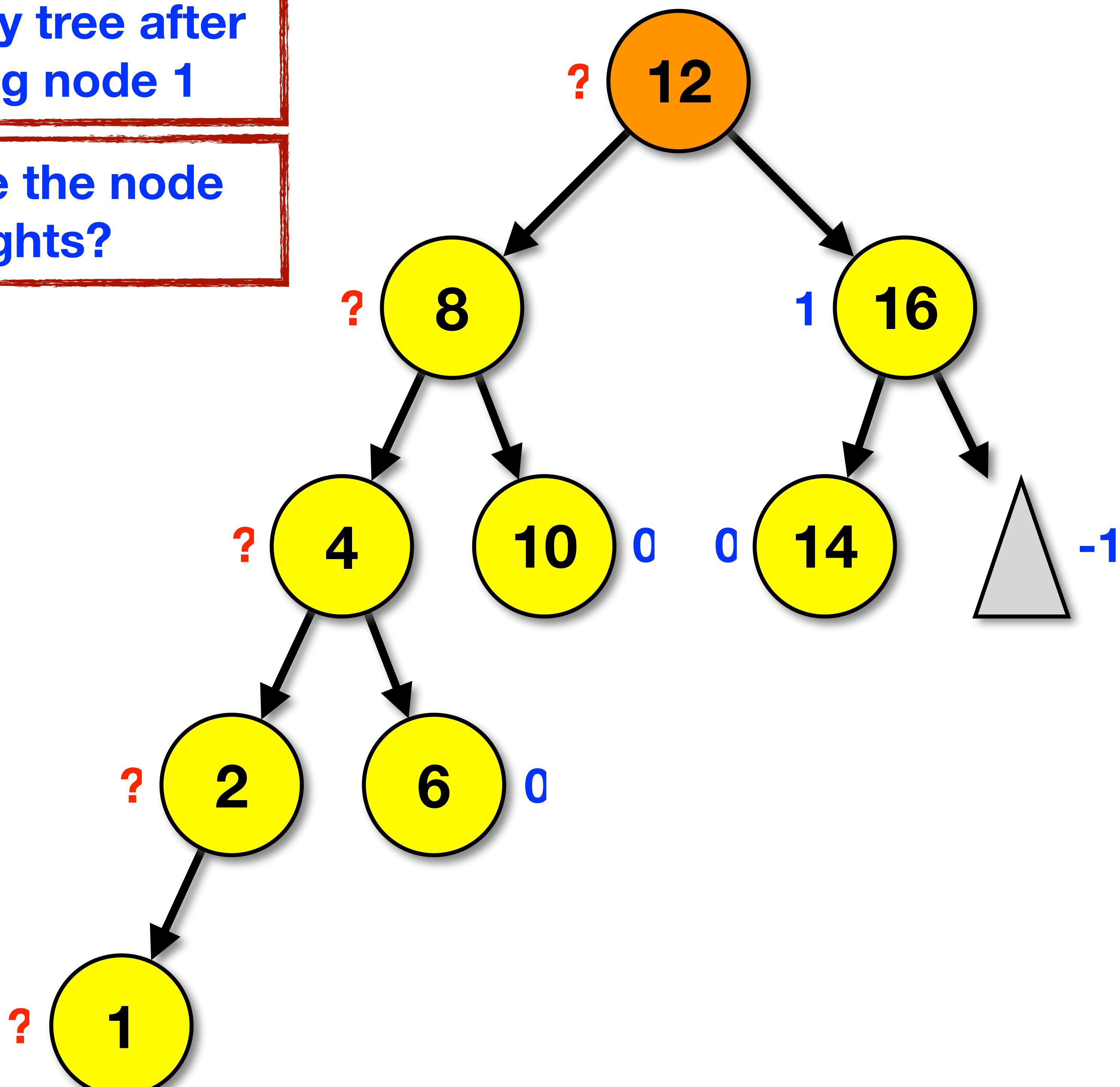


An AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right subtrees can differ by at most 1. The height of an empty subtree is -1.

AVL Trees

The binary tree after inserting node 1

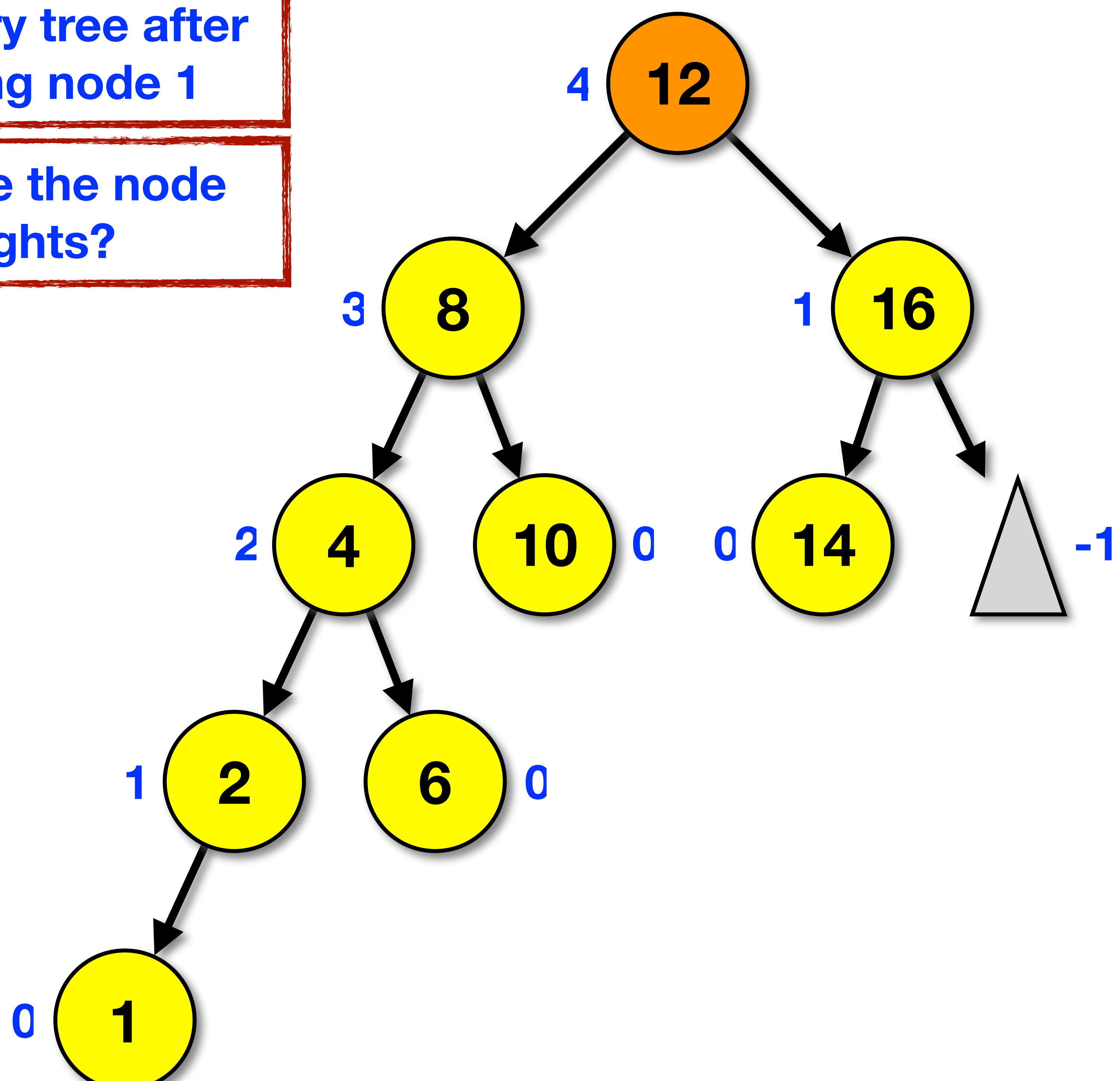
What are the node heights?



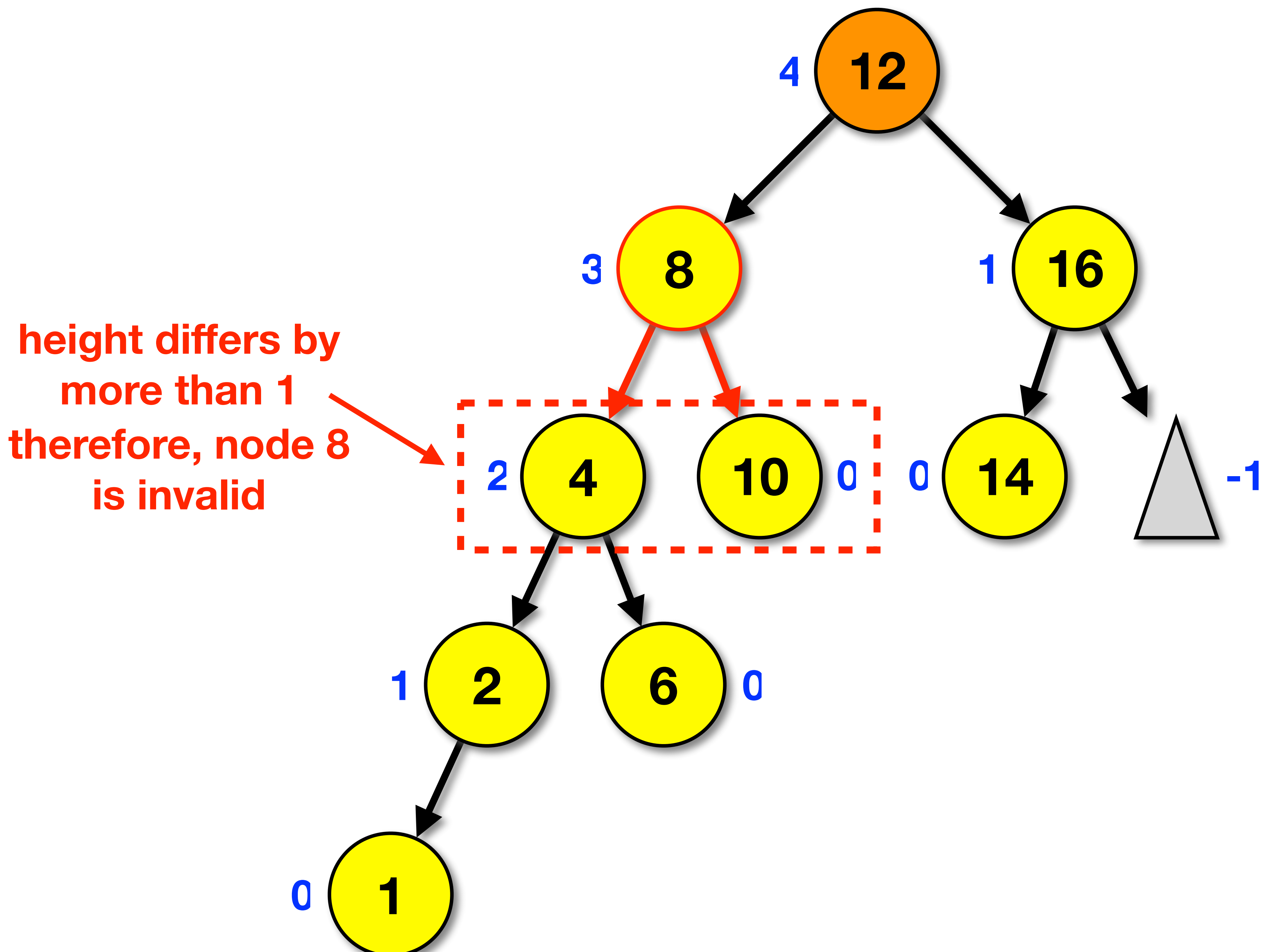
AVL Trees

The binary tree after inserting node 1

What are the node heights?

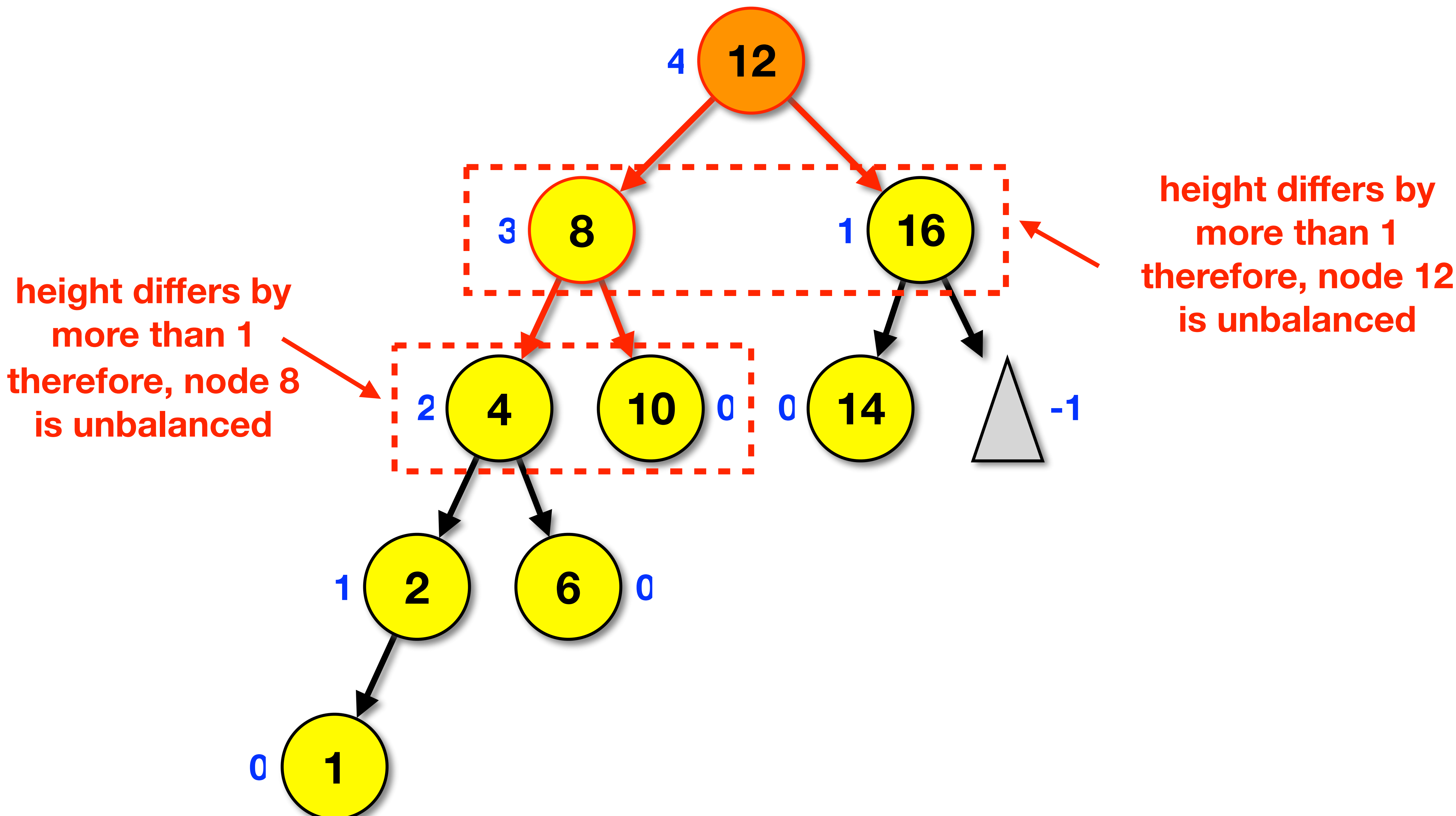


AVL Trees



4
12
3
8
1
16
height differs by
more than 1
therefore, node 8
is invalid
2
4
10
0
0
14
-1
1
2
6
0
0
1

AVL Trees



4
12
3
8
1
16
2
4
10
0
0
14
-1
1
2
6
0
0
1
height differs by more than 1 therefore, node 12 is unbalanced
height differs by more than 1 therefore, node 8 is unbalanced

AVL Trees

- **When a node in the tree no longer satisfies the invariant required to be an AVL tree the tree must be rebalanced around that node**

- **There are four ways in which an insertion into a tree may cause an imbalance to occur at some node X:**

  (1) An insertion in the left subtree of the left child of X

  (2) An insertion in the right subtree of the left child of X

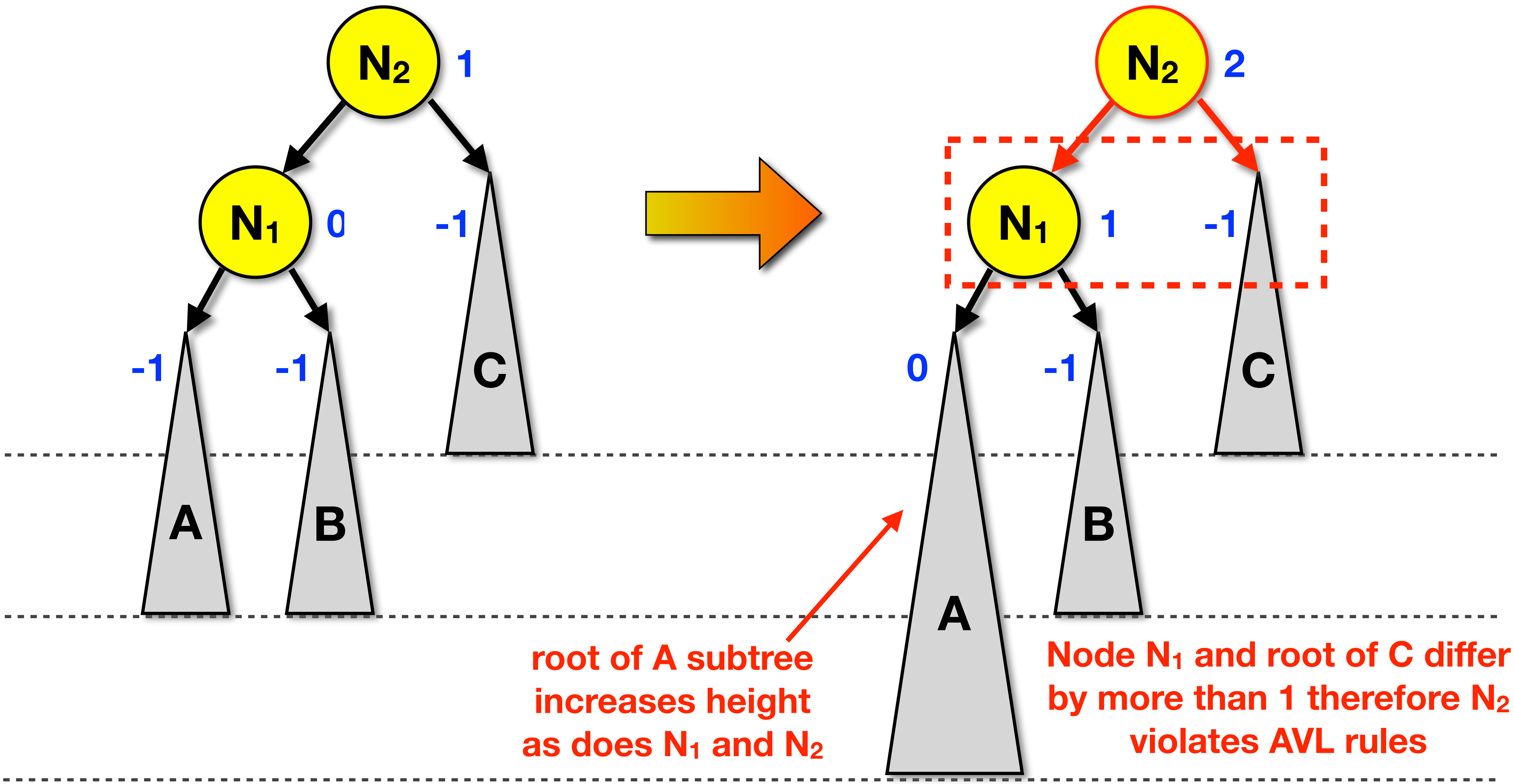
  (3) An insertion in the left subtree of the right child of X

  (4) An insertion in the right subtree of the right child of X

symmetric

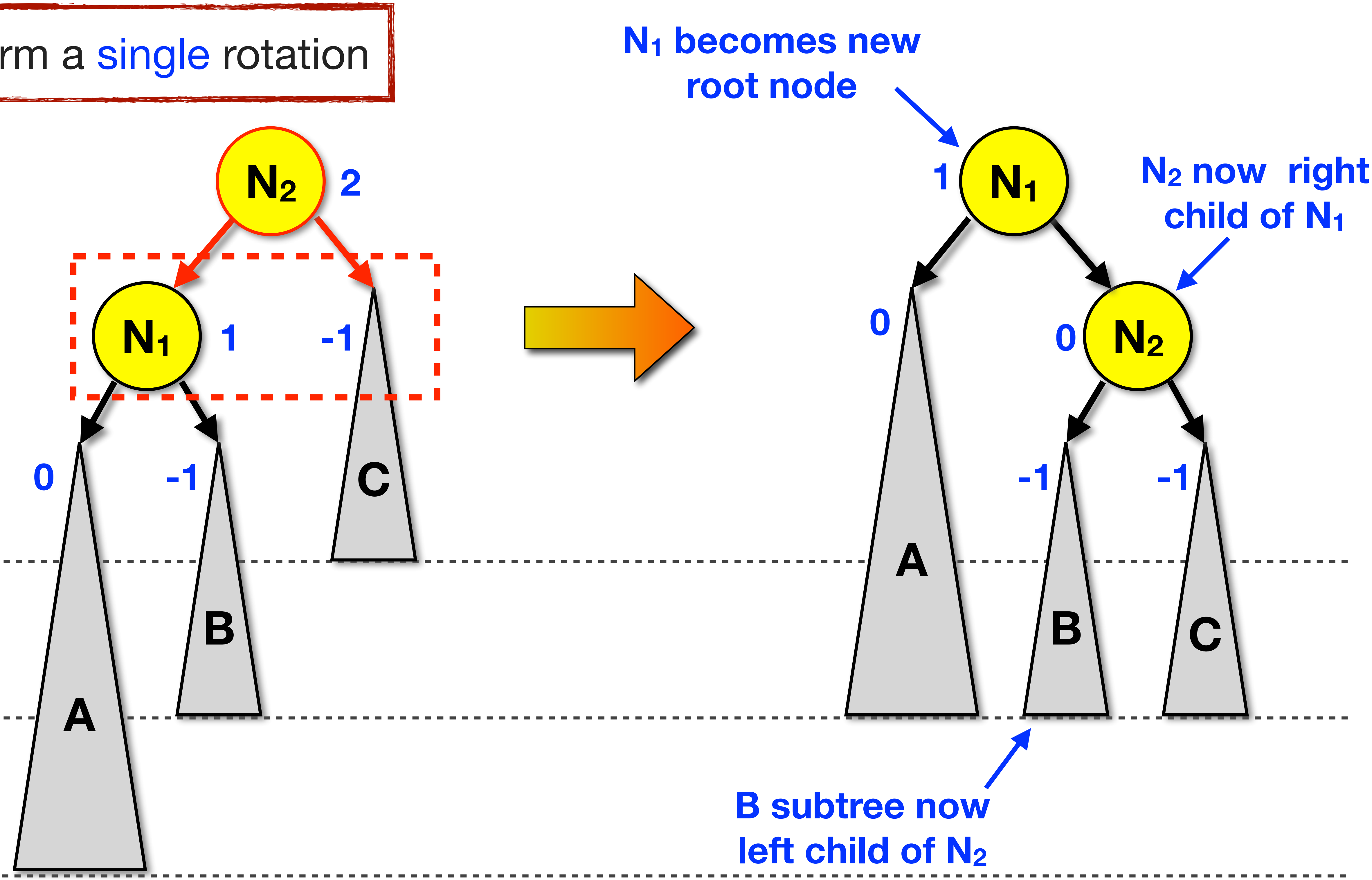
Creating an Imbalance -- **Case #1**

(1) An insertion in the left subtree of the left child of X



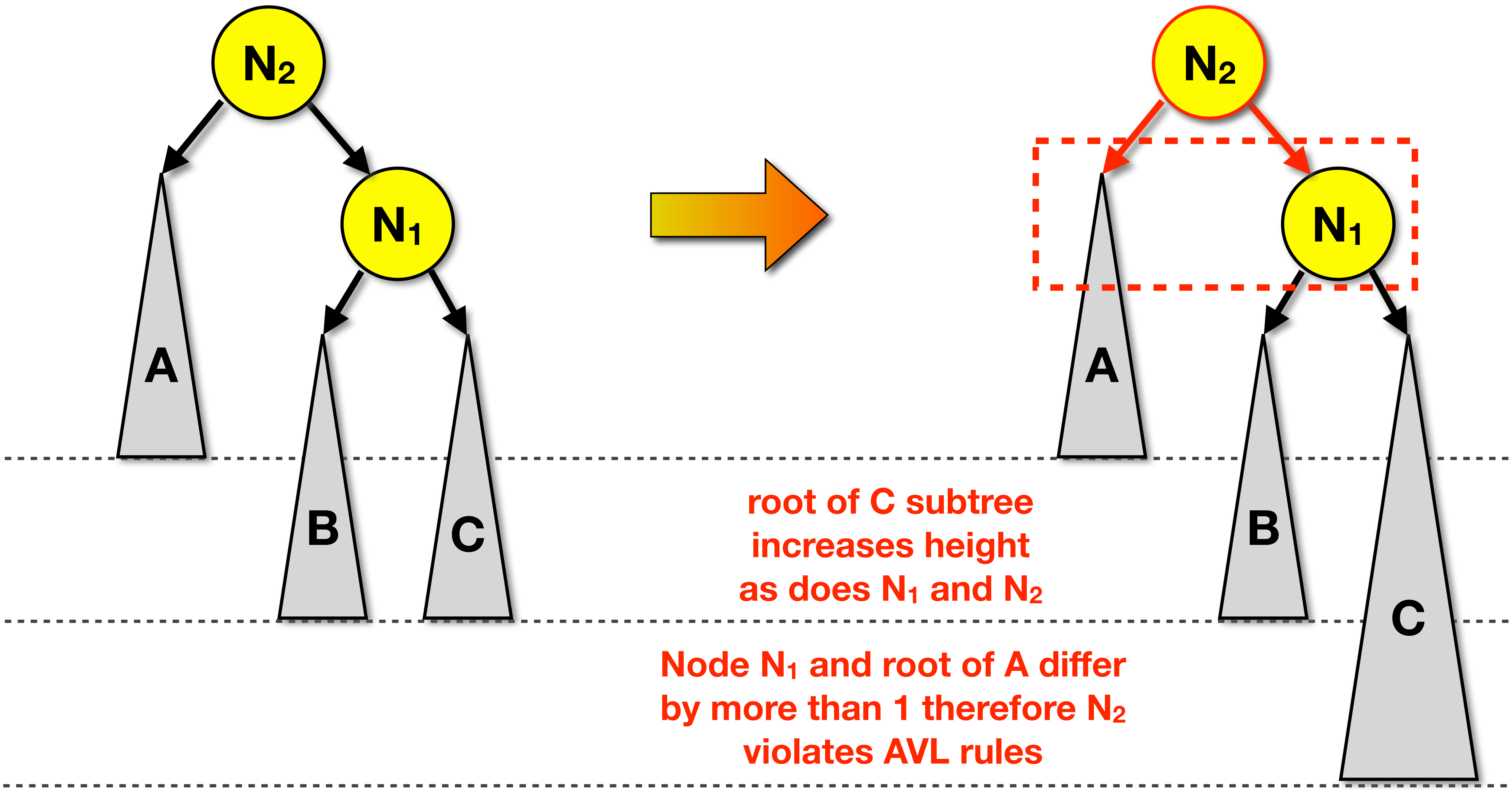
Fixing the Imbalance -- **Case #1**

Perform a single rotation



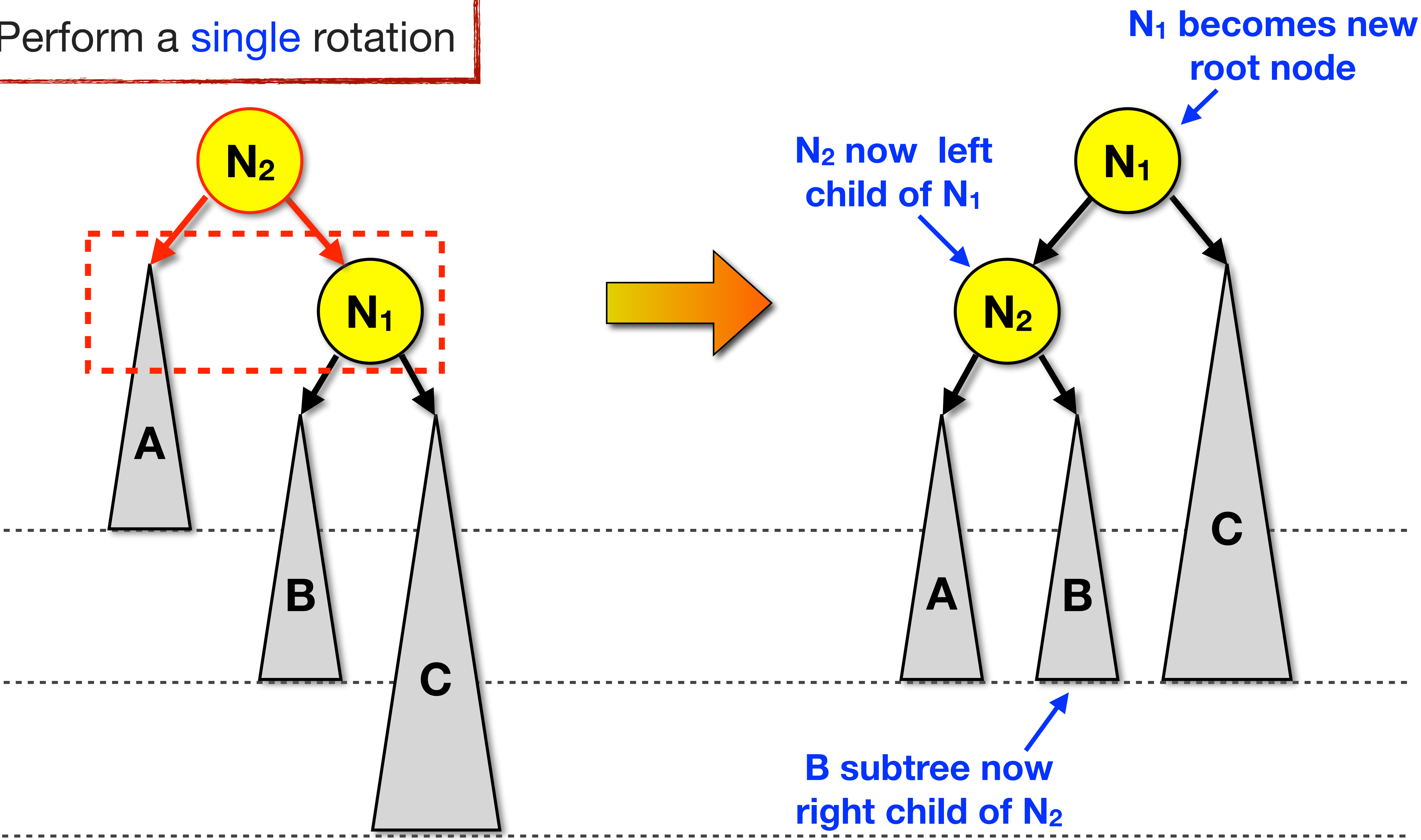
Creating an Imbalance -- **Case #4** **(symmetric with 1)**

(4) An insertion in the right subtree of the right child of X



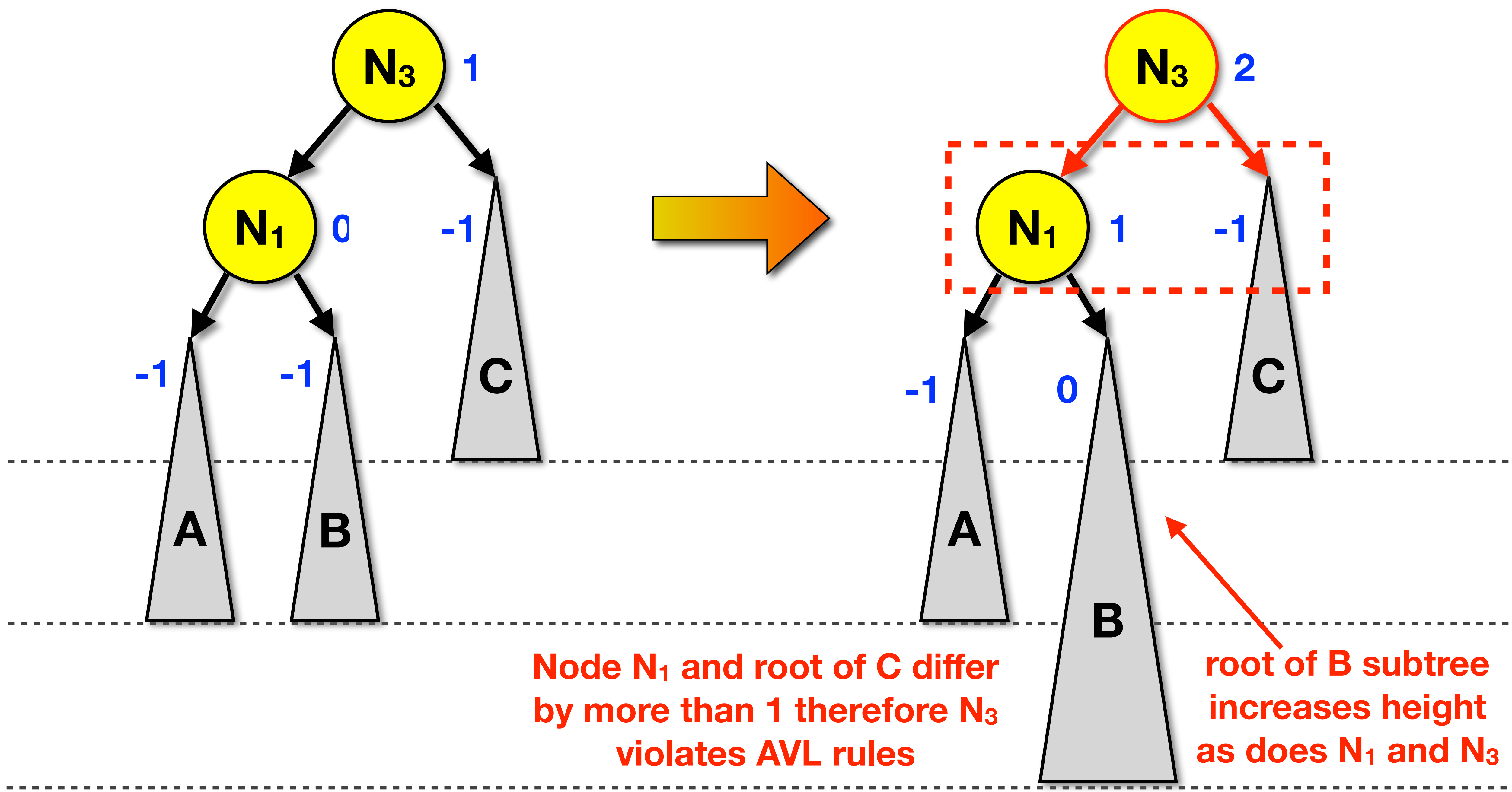
Fixing the Imbalance -- **Case #4 (symmetric with 1)**

Perform a single rotation



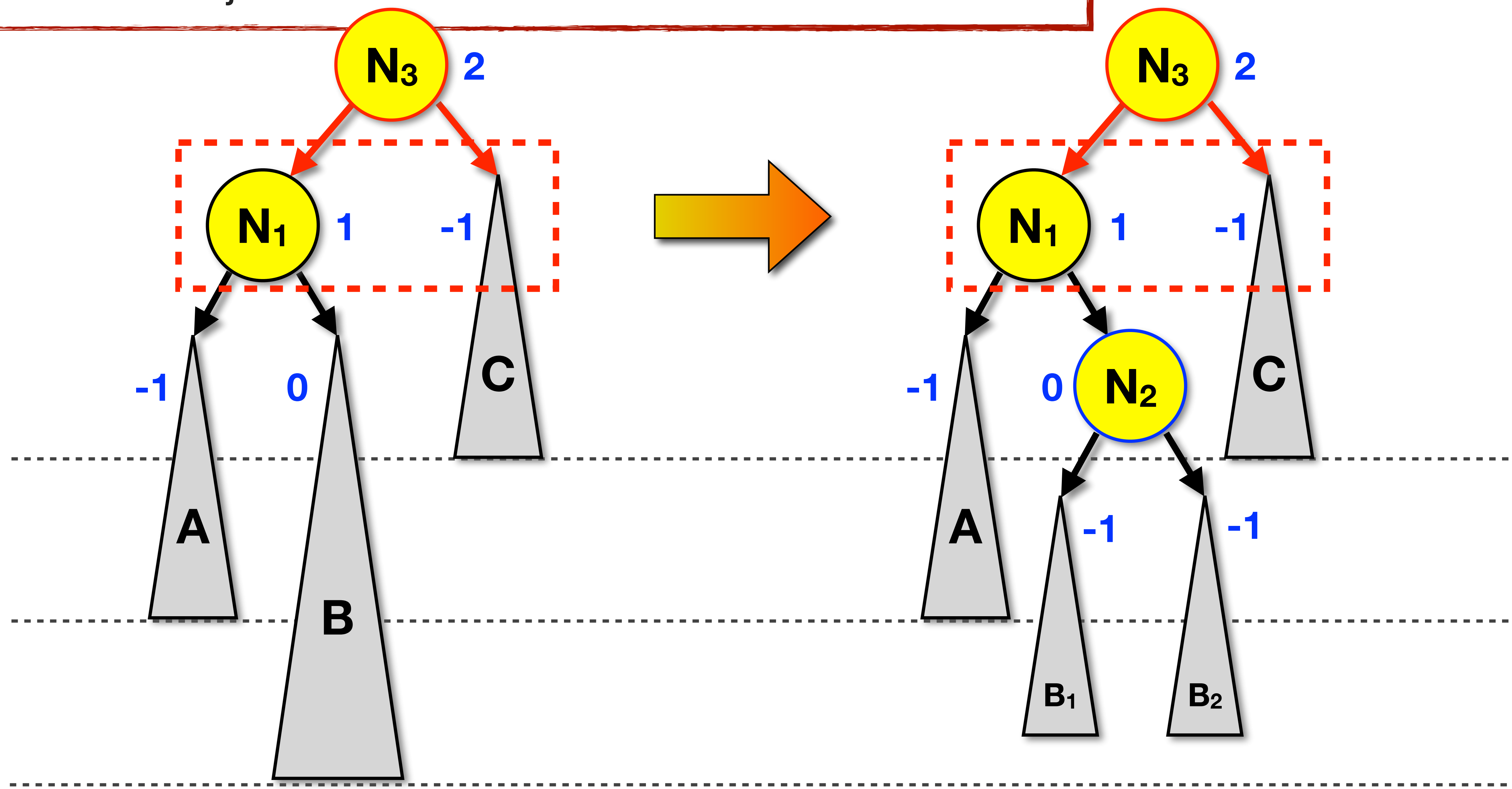
Creating an Imbalance -- **Case #2**

(2) An insertion in the right subtree of the left child of X



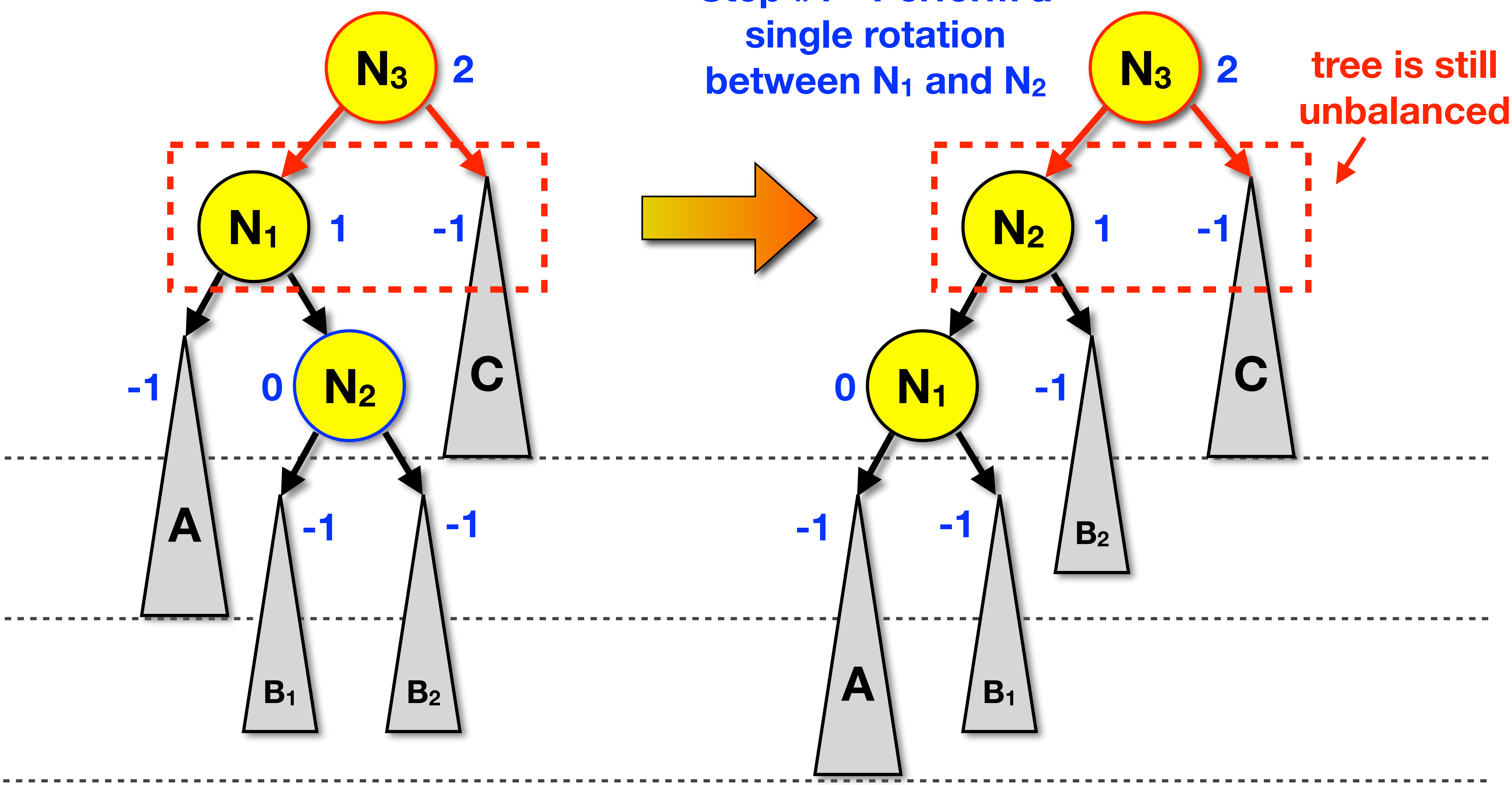
Fixing the Imbalance -- **Case #2**

We know the node N_2 exists because a node was just inserted into the B subtree



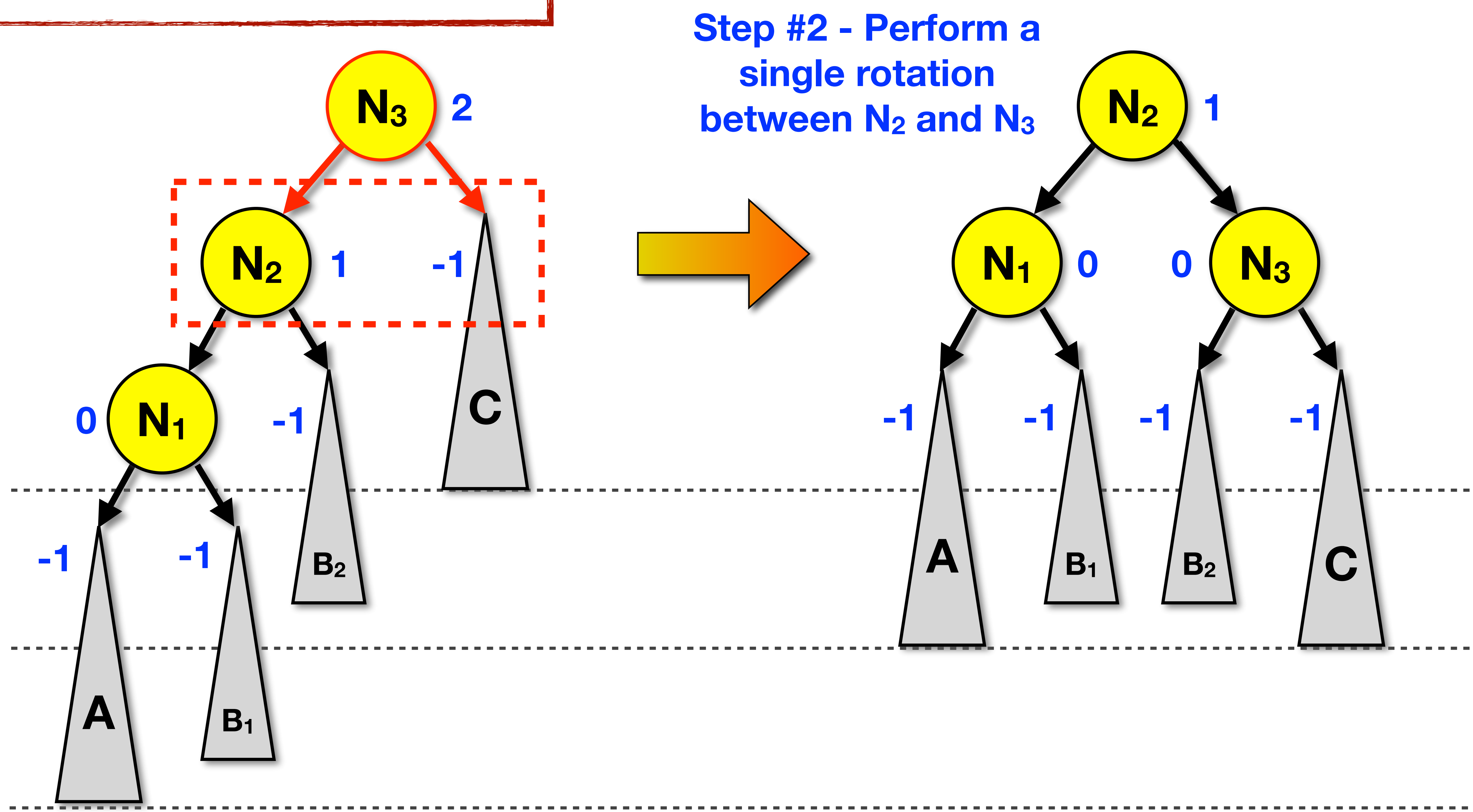
Fixing the Imbalance -- **Case #2**

Perform a double rotation



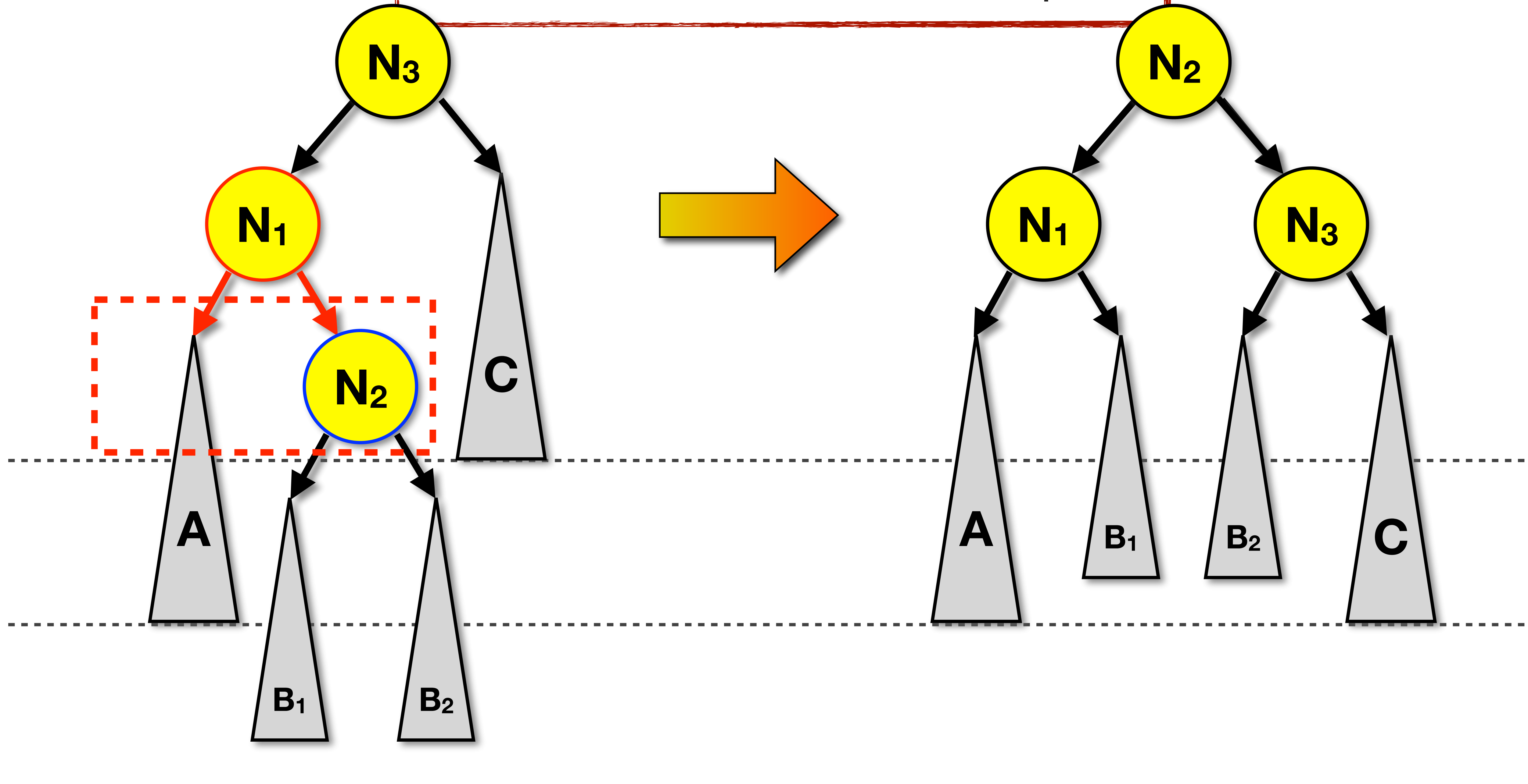
Fixing the Imbalance -- **Case #2**

Perform a double rotation



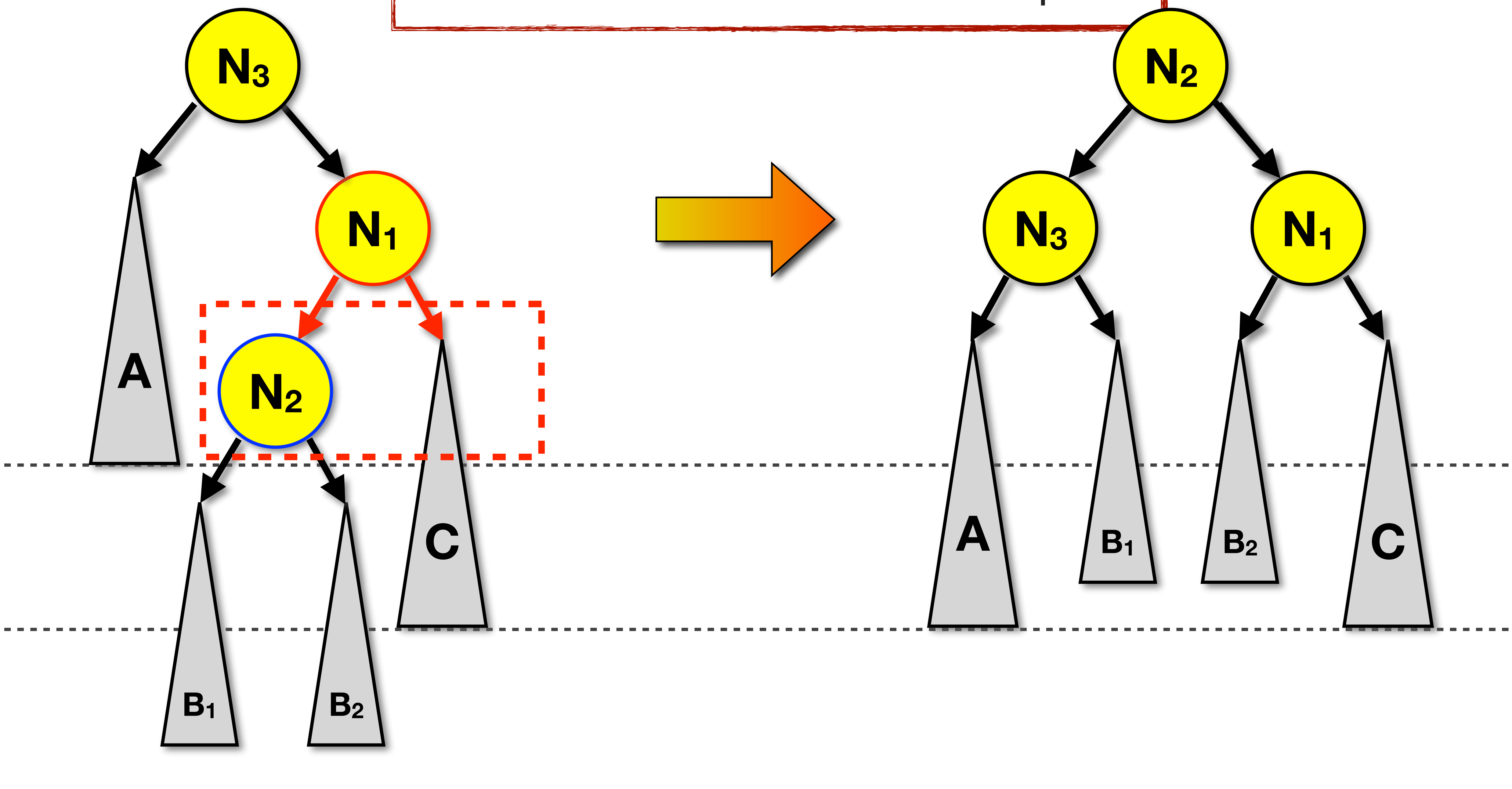
Fixing the Imbalance -- **Case #2**

Another look at the double rotation without the intermediate steps

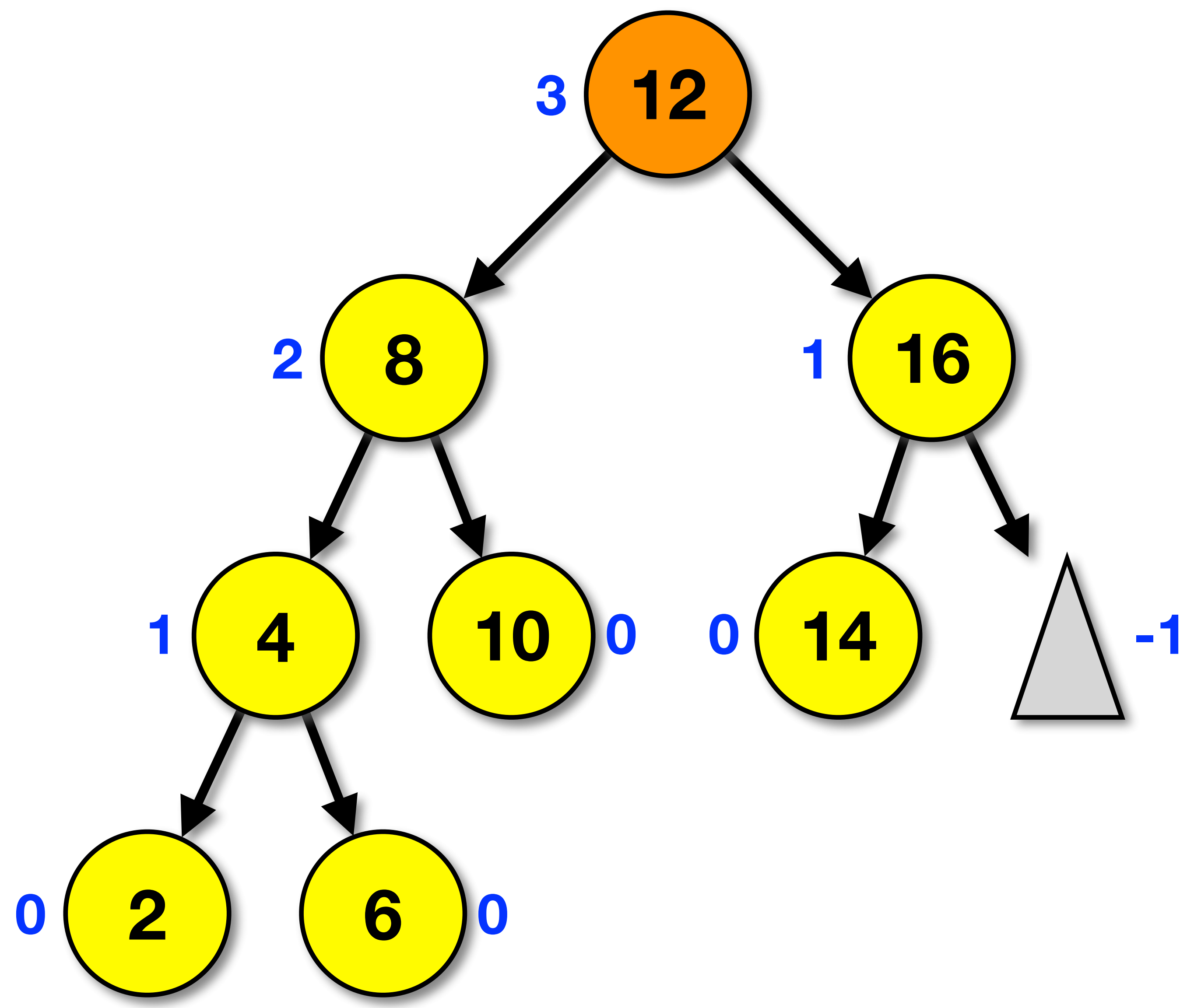


Fixing the Imbalance -- **Case #3** **(symmetric with 2)**

Another look at the double rotation without the intermediate steps

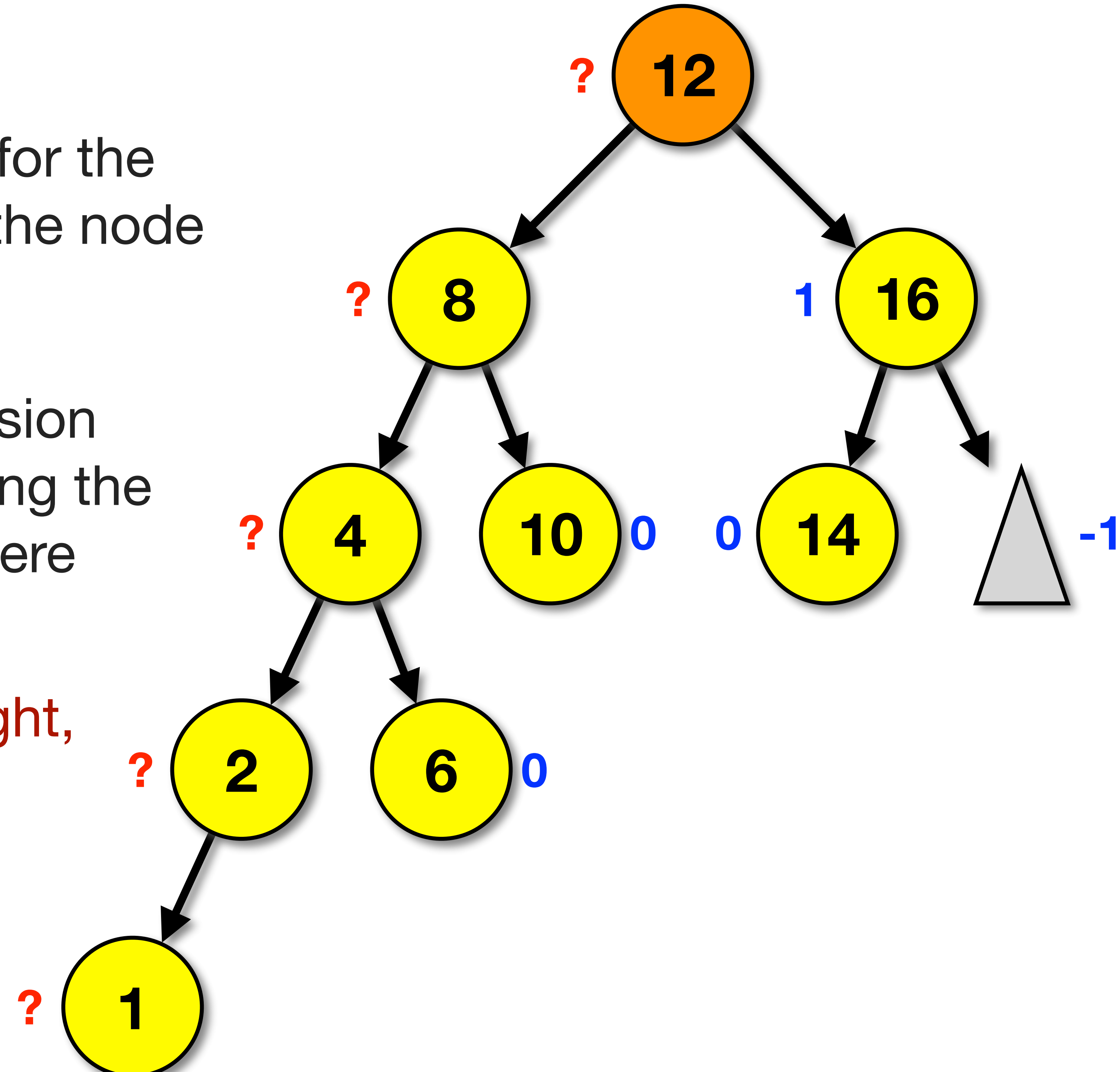


AVL Tree Insertion Example

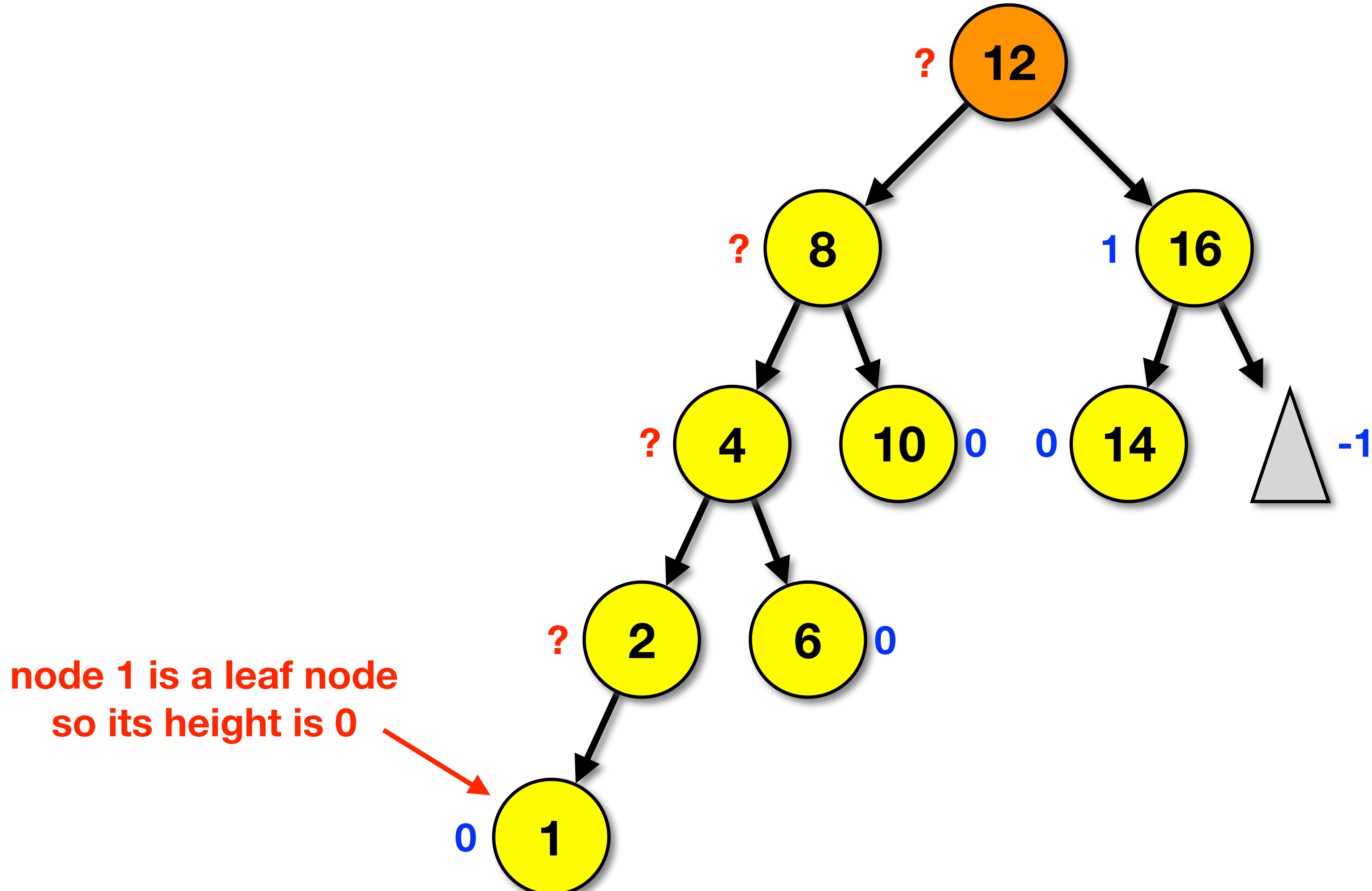


AVL Tree Insertion Example

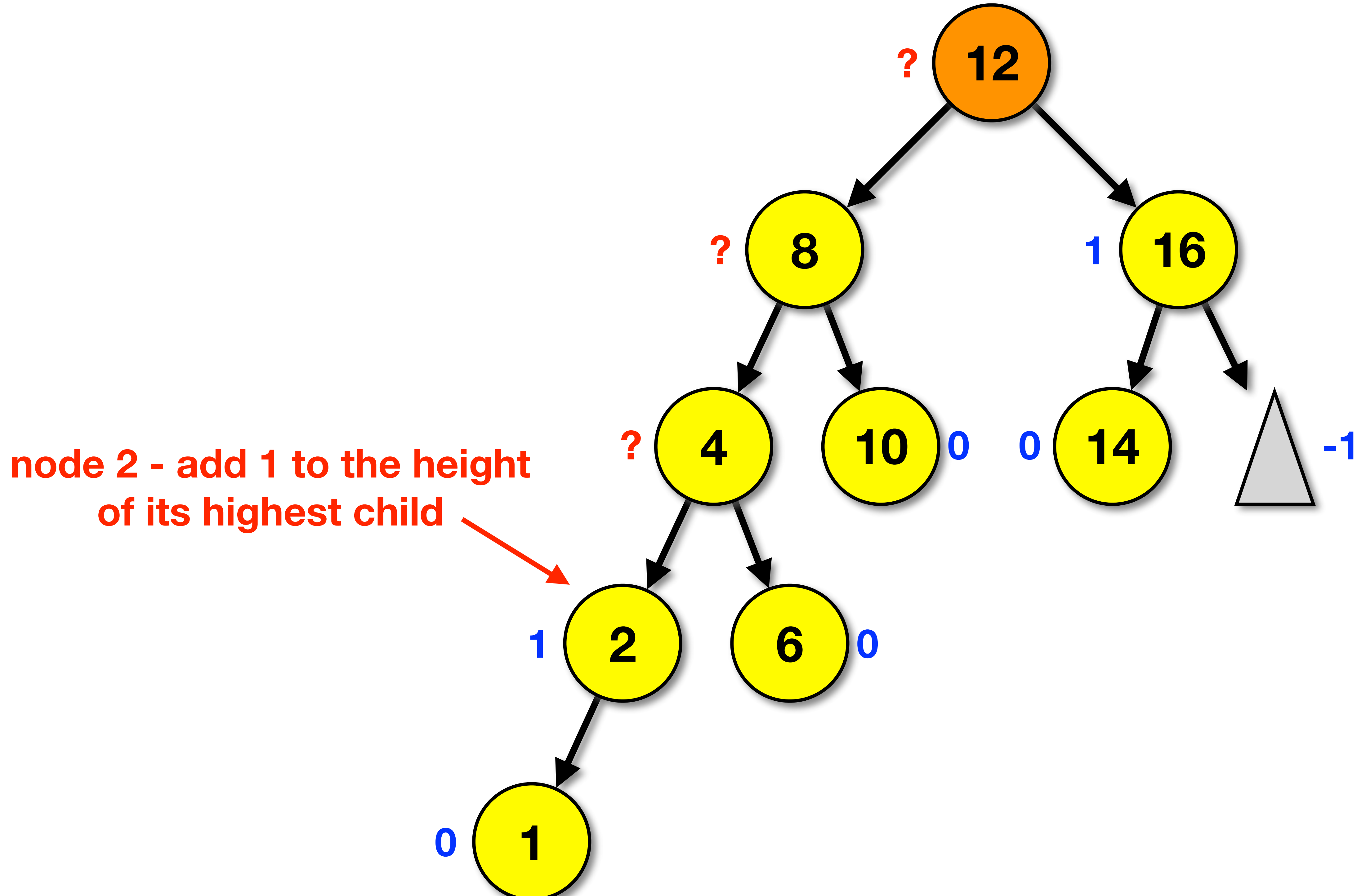
- **Inserting node 1**
  - First, recursively search for the location which to insert the node
  - Next, insert the node
  - Finally, unwind the recursion update node heights along the way. Rebalance tree where necessary
    - To update node's height, check its children



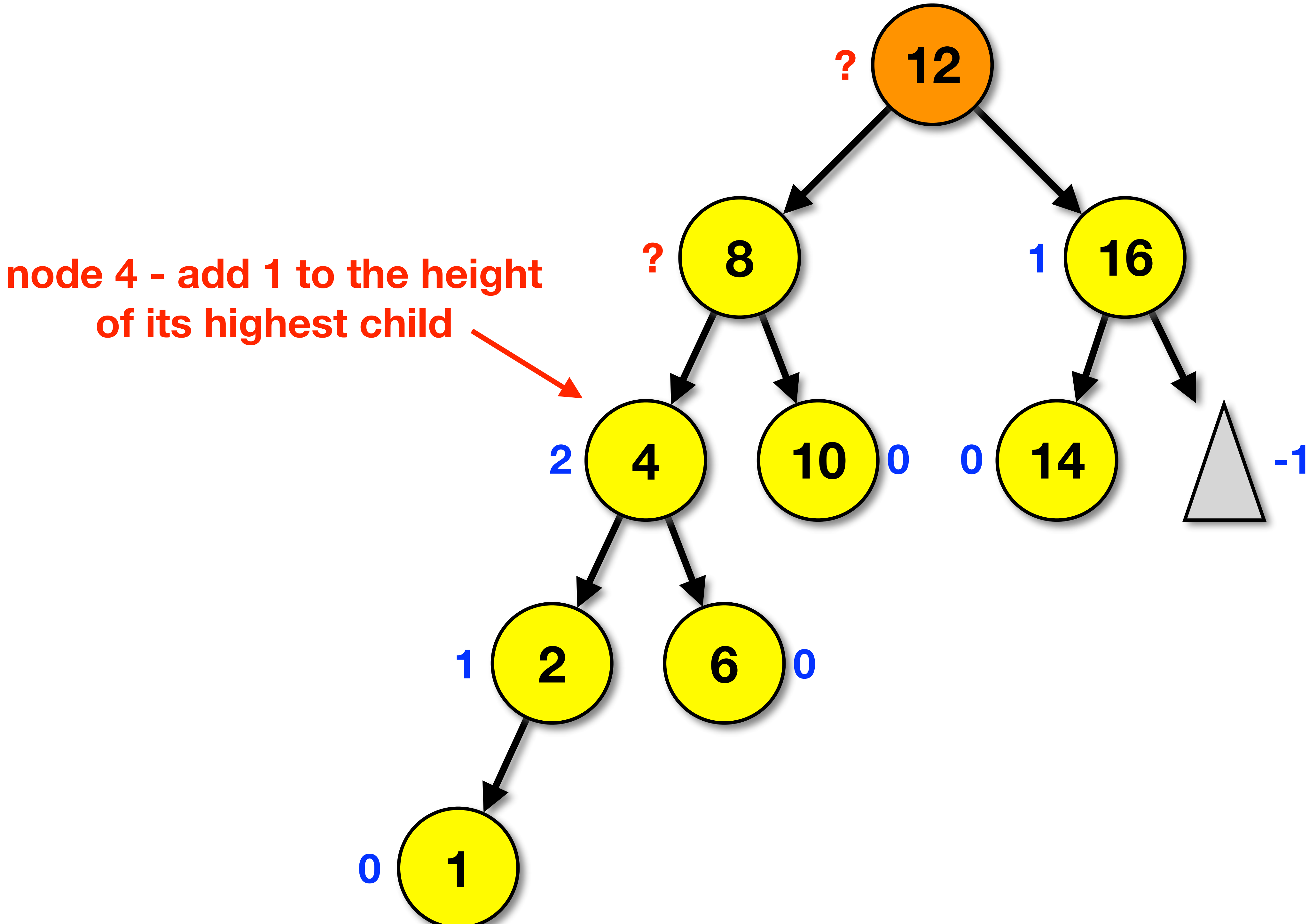
AVL Tree Insertion Example



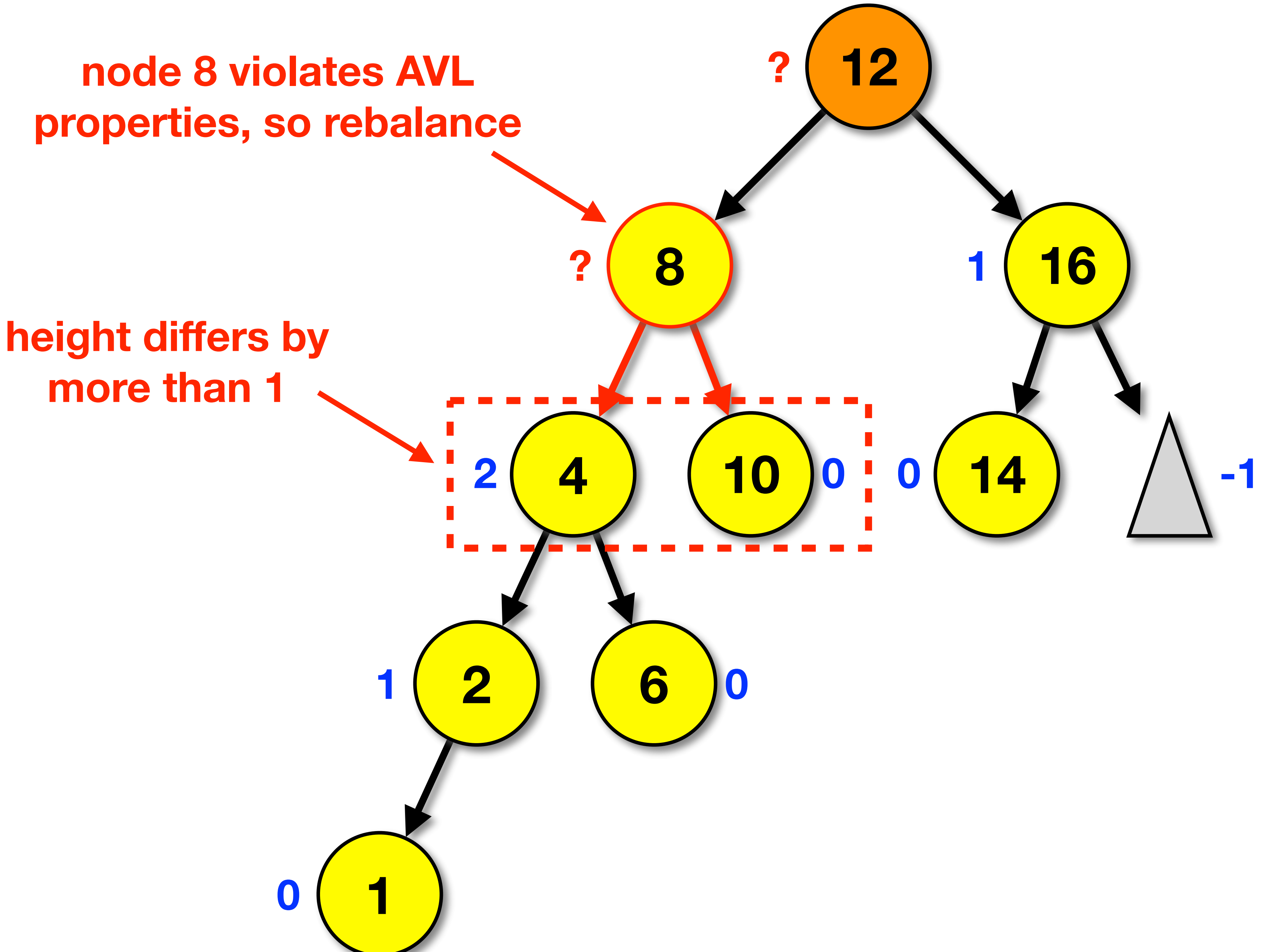
AVL Tree Insertion Example



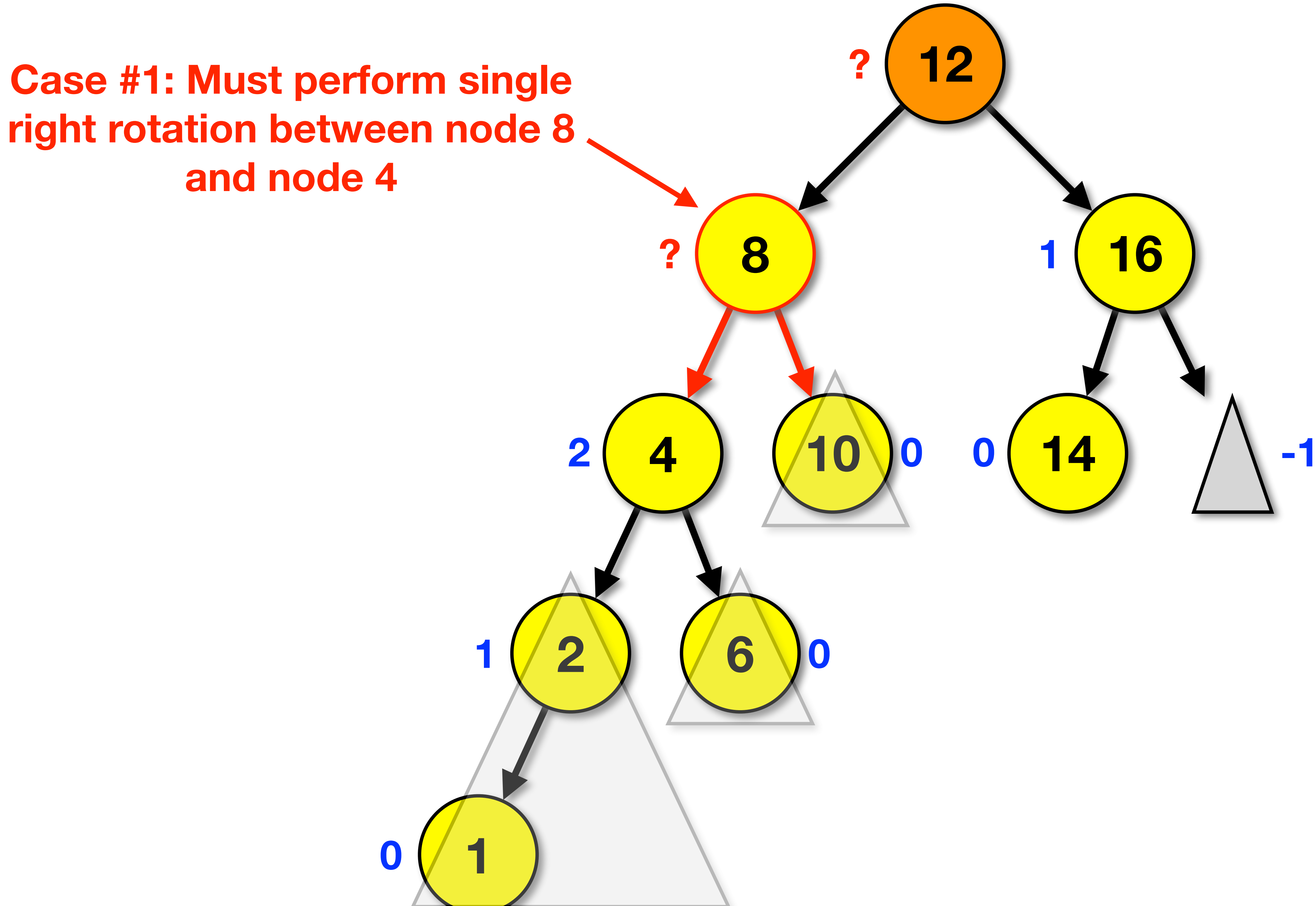
AVL Tree Insertion Example



AVL Tree Insertion Example

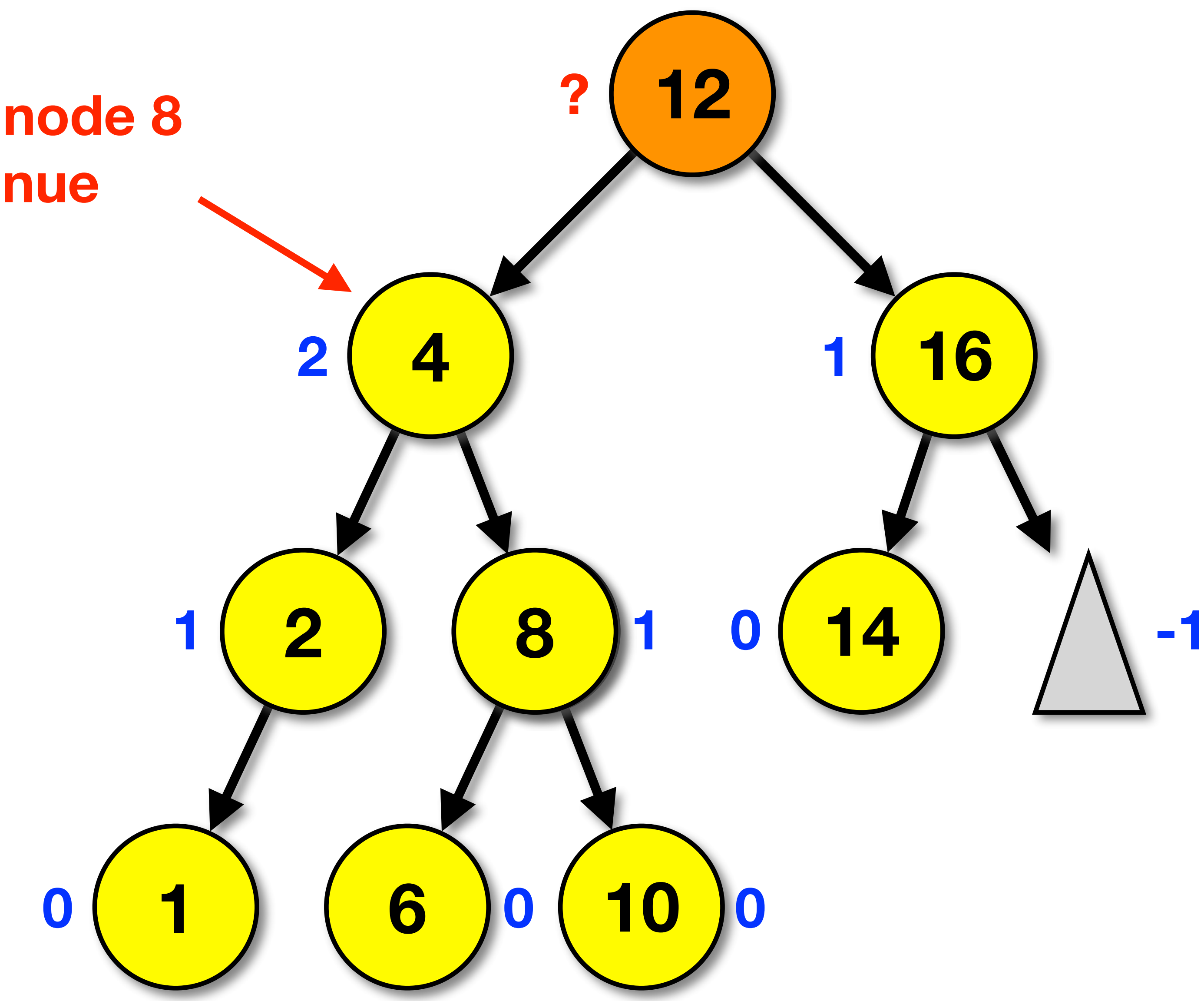


AVL Tree Insertion Example

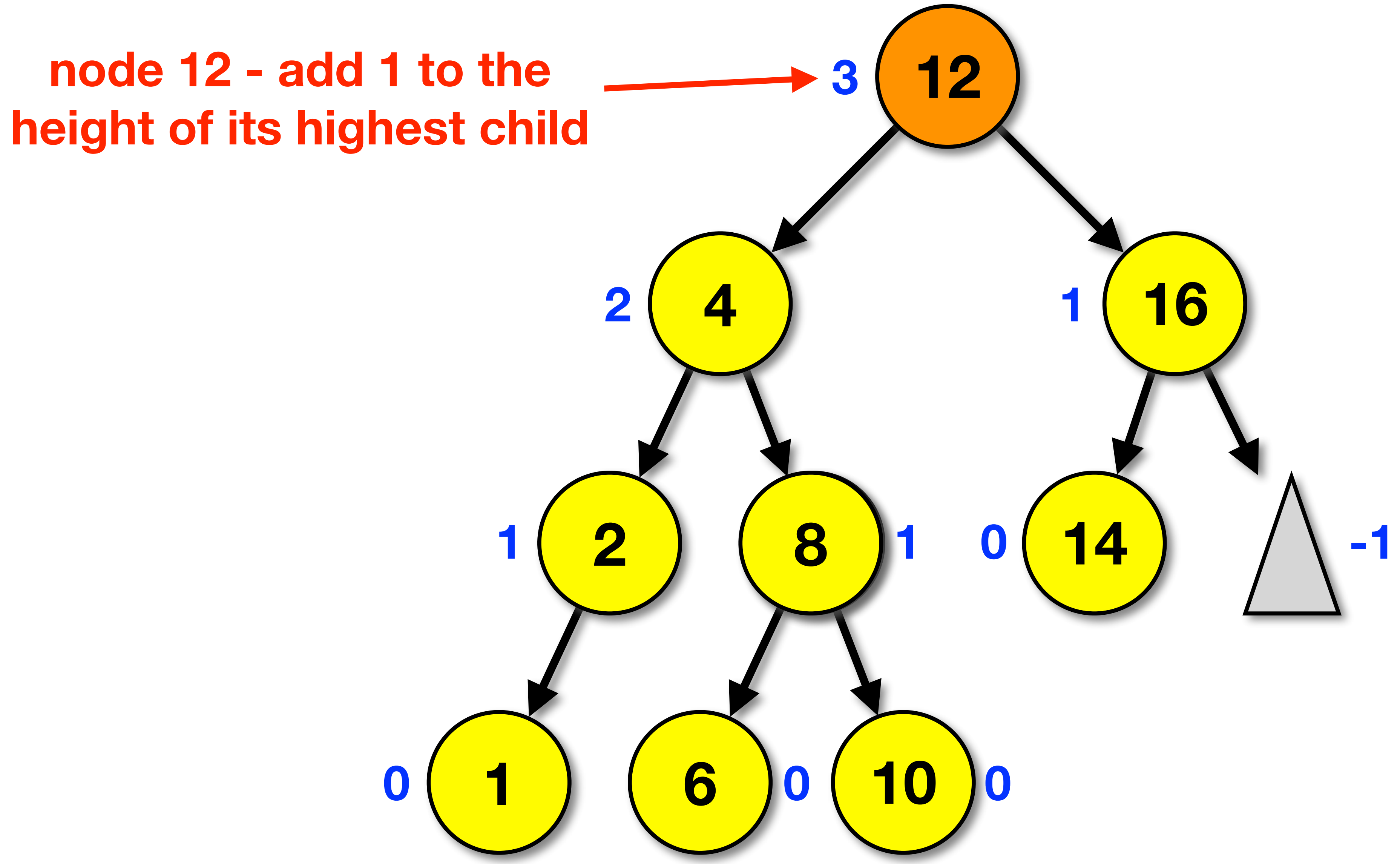


AVL Tree Insertion Example

After rotation, update height of node 8
Node 4 is balanced, so continue
unwinding recursion



AVL Tree Insertion Example



Other Operations: `find` / `remove`

- **The `find` operation is the same as the unbalanced binary search tree**
- **The `remove` operation works similarly to the `remove` operation from the unbalanced binary search tree with a few modifications**
  - When a node is removed, the heights of its ancestors may need to be updated as the recursion is unwound -- fix imbalances as they are encountered just like with insertion

Analysis of AVL Tree Operations

- **Time complexity of AVL Tree operations**

	worst case	average
`find`	O(log N)	O(log N)
`insert`	O(log N)	O(log N)
`remove`	O(log N)	O(log N)