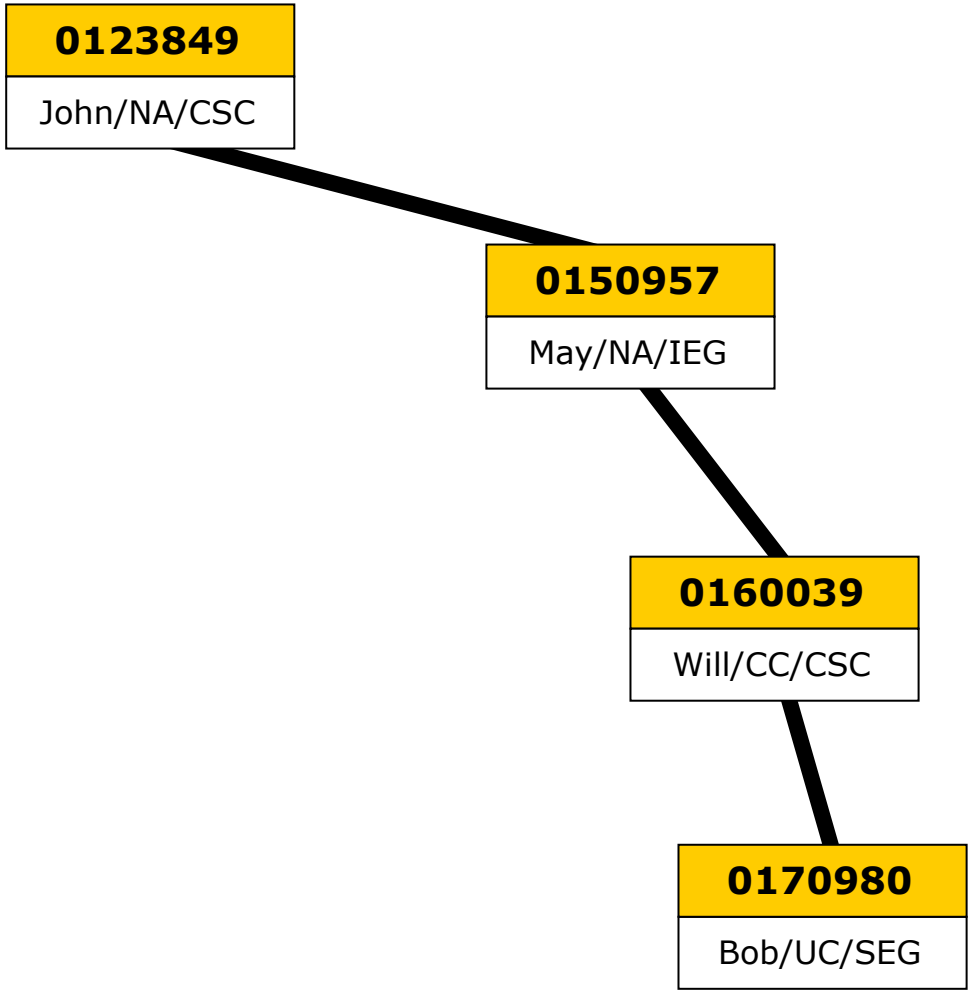
Binary Search Tree

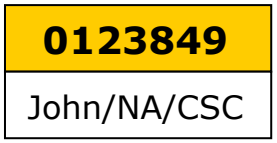


This is an example of bad binary search trees.

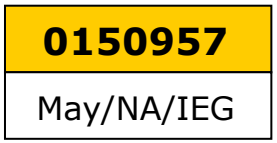
Balanced Binary Search Tree

Trees created by consecutive **InsertNode** calls are in general unbalanced.

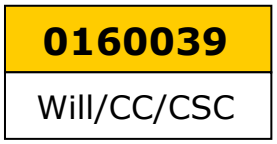
t = EmptyBinaryTree();

t = InsertNode(t, 

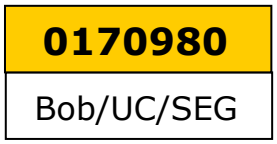
);

t = InsertNode(t, 

);

t = InsertNode(t, 

);

t = InsertNode(t, 

);

```
BinaryTreeADT InsertNode(BinaryTreeADT t, TreeNodeADT n) {

   if (TreeIsEmpty(t)) return NonemptyBinaryTree(n,
         EmptyBinaryTree(), EmptyBinaryTree());
   else {
      int sign = strcmp(GetNodeKey(n), GetNodeKey(Root(t)));
      if (sign == 0) return NonemptyBinaryTree(n,
         LeftSubtree(t), RightSubtree(t));
      if (sign < 0) return NonemptyBinaryTree(Root(t),
         InsertNode(LeftSubtree(t),n), RightSubtree(t));
      return NonemptyBinaryTree(Root(t),
         LeftSubtree(t), InsertNode(RightSubtree(t),n));
      }
}
```

Balanced Binary Search Tree

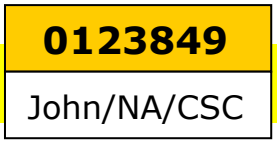
Trees created by consecutive **InsertNode** calls are in general unbalanced.

```
t = EmptyBinaryTree();
t = InsertNode(t, 0123849 John/NA/CSC );
t = InsertNode(t, 0150957 May/NA/IEG );
t = InsertNode(t, 0160039 Will/CC/CSC );
t = InsertNode(t, 0170980 Bob/UC/SEG );
```

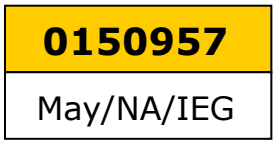
Balanced Binary Search Tree

Trees created by consecutive **InsertNode** calls are in general unbalanced.

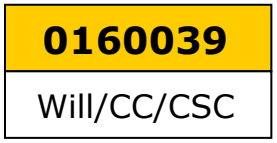
t = EmptyBinaryTree();

t = InsertNode(t, 

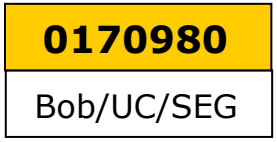
);

t = InsertNode(t, 

);

t = InsertNode(t, 

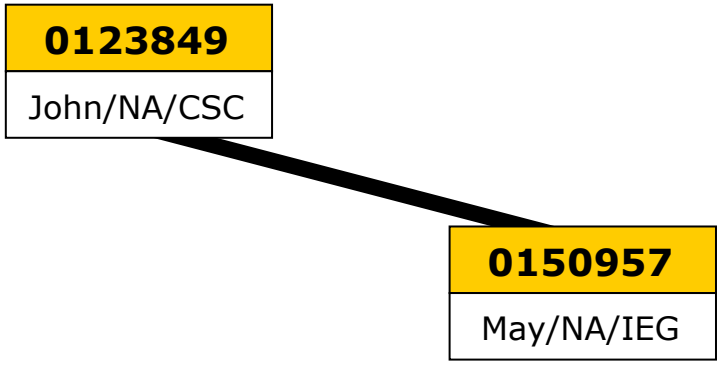
);

t = InsertNode(t, 

);

Balanced Binary Search Tree

Trees created by consecutive **InsertNode** calls are in general unbalanced.



t = EmptyBinaryTree();

t = InsertNode(t, 0123849 John/NA/CSC);

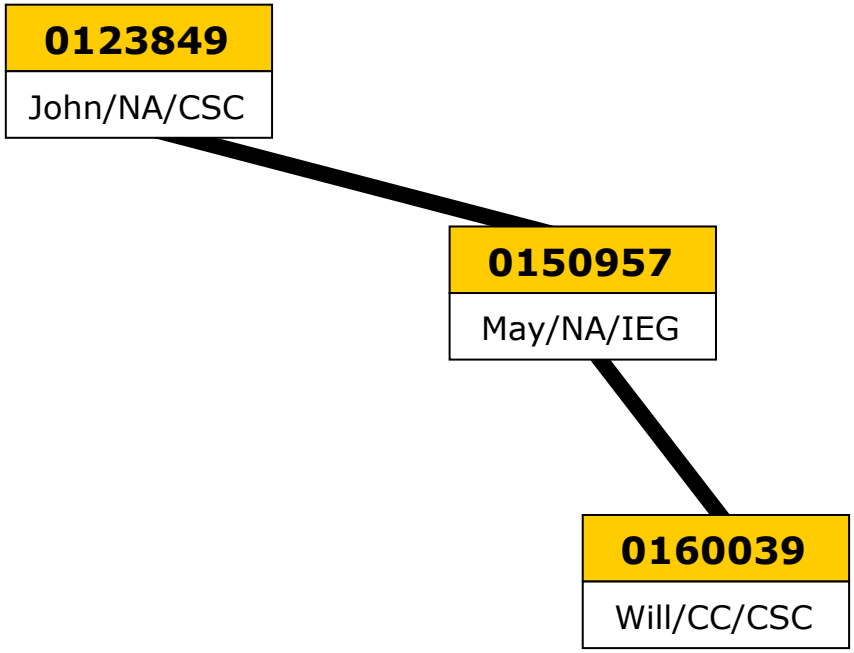
t = InsertNode(t, 0150957 May/NA/IEG);

t = InsertNode(t, 0160039 Will/CC/CSC);

t = InsertNode(t, 0170980 Bob/UC/SEG);

Balanced Binary Search Tree

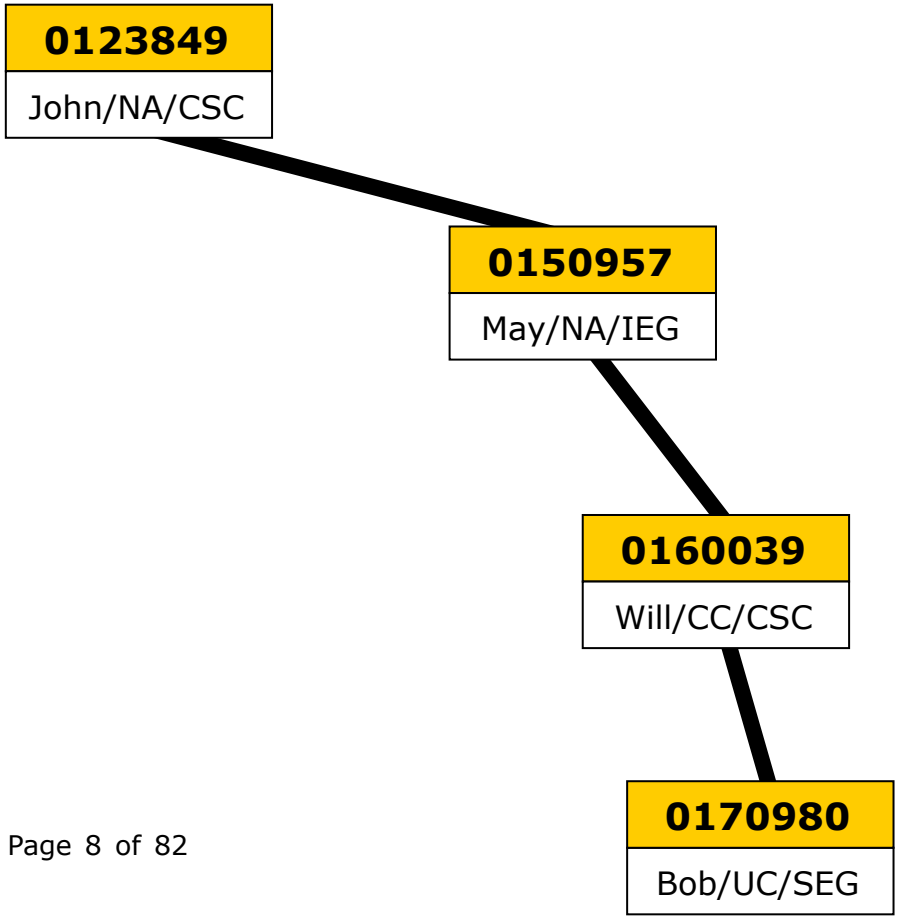
Trees created by consecutive **InsertNode** calls are in general unbalanced.




```
t = EmptyBinaryTree();
t = InsertNode(t, 0123849 John/NA/CSC );
t = InsertNode(t, 0150957 May/NA/IEG );
t = InsertNode(t, 0160039 Will/CC/CSC );
t = InsertNode(t, 0170980 Bob/UC/SEG );
```

Balanced Binary Search Tree

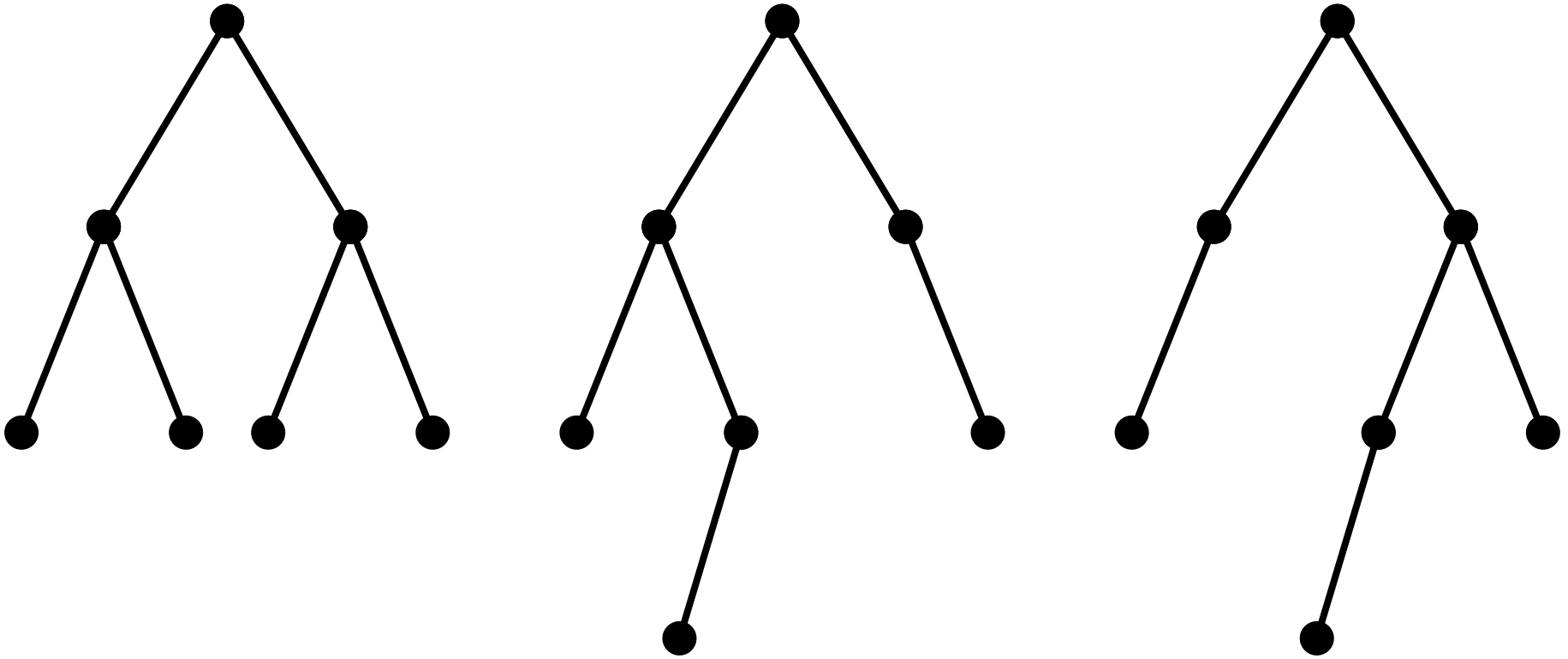
Trees created by consecutive **InsertNode** calls are in general unbalanced.



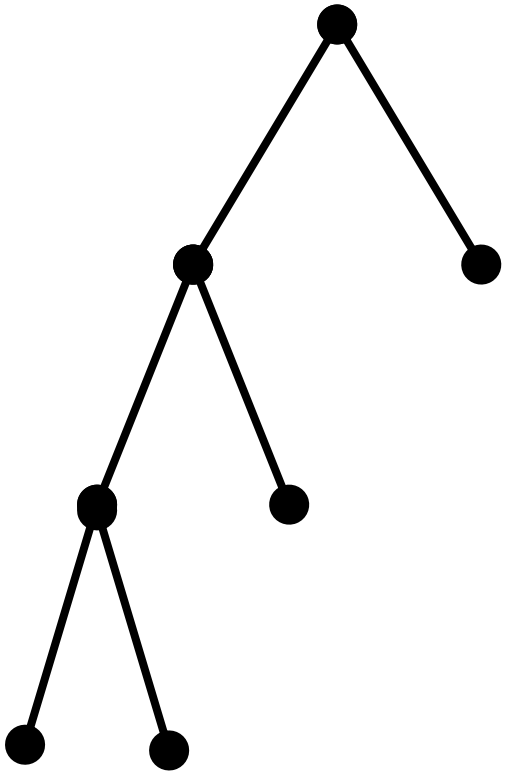

```
t = EmptyBinaryTree();
t = InsertNode(t, 0123849 John/NA/CSC );
t = InsertNode(t, 0150957 May/NA/IEG );
t = InsertNode(t, 0160039 Will/CC/CSC );
t = InsertNode(t, 0170980 Bob/UC/SEG );
```

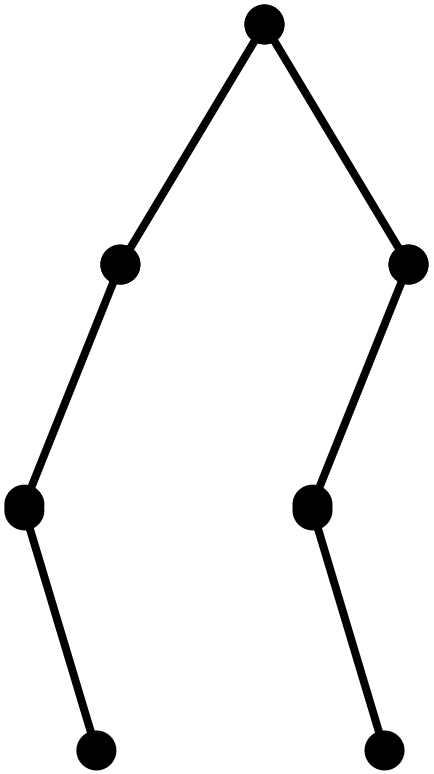
Balanced Binary Search Tree

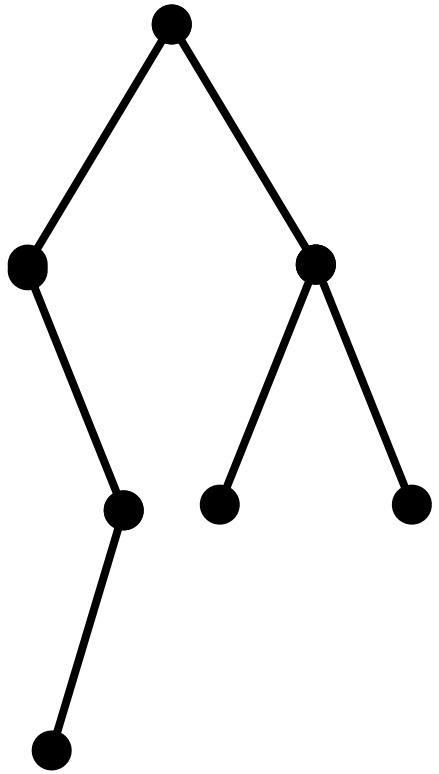
A tree is balanced if, at each node, the heights of the left and right subtrees differ by at most one.



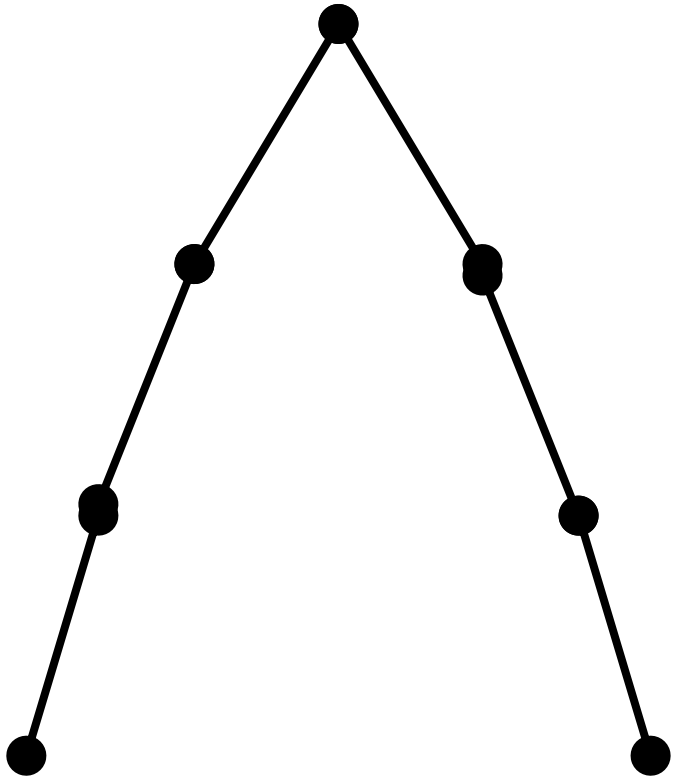
Are the following Balanced Binary Search Trees?



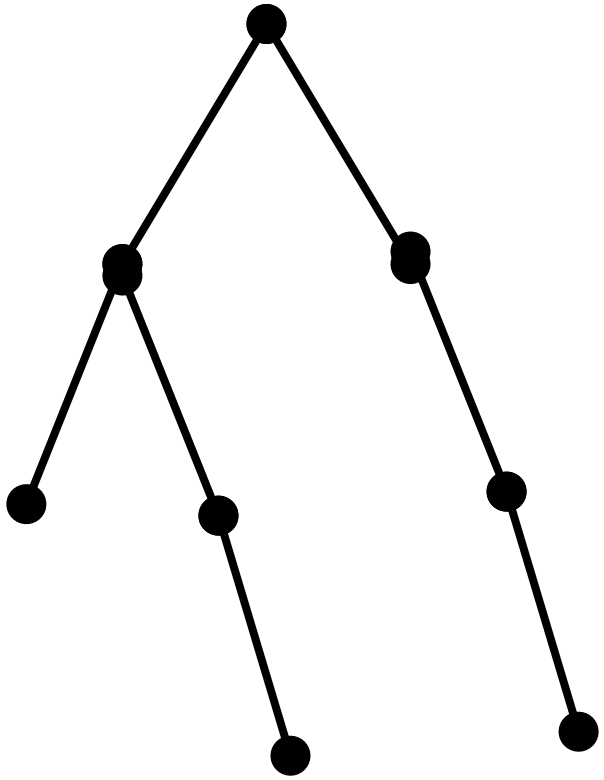




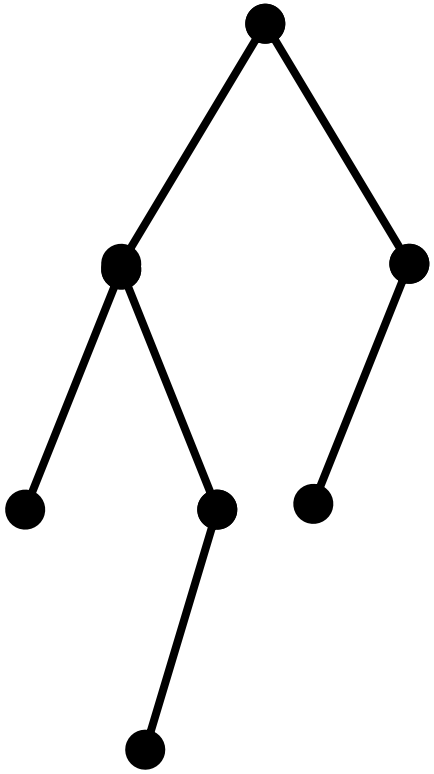
Is the following a Balanced Binary Search Tree?

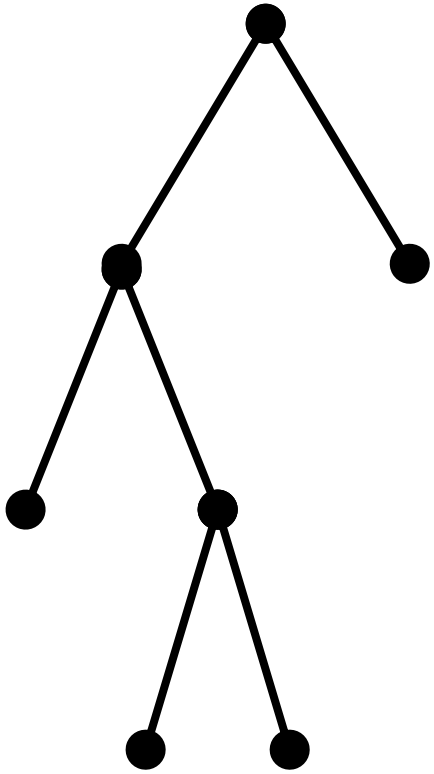


Is the following a Balanced Binary Search Tree?



Question: Are the following trees balanced?



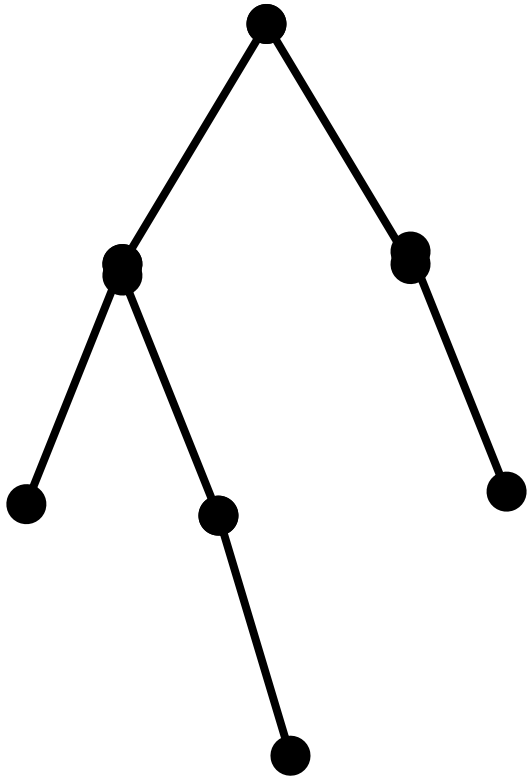


AVL Tree

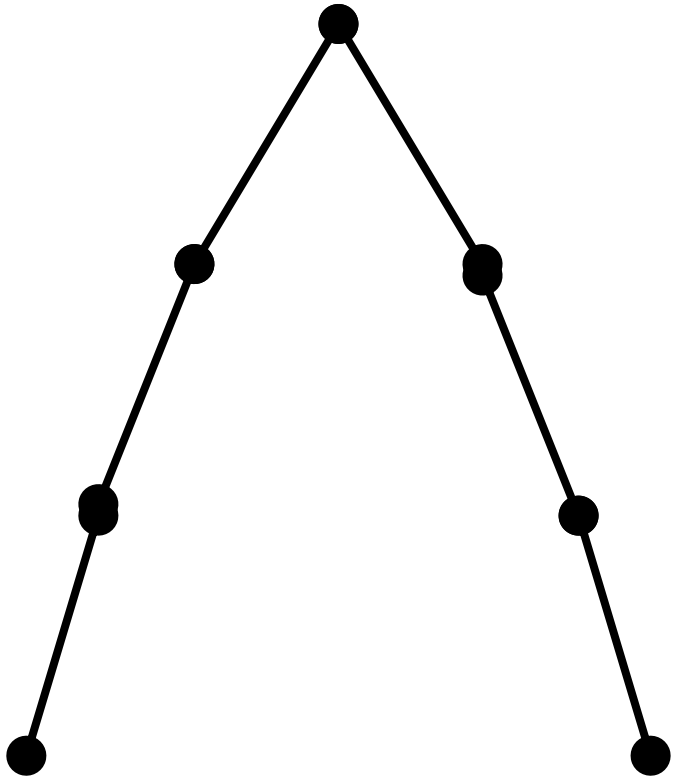
An **AVL** (Adelson-Velskii and Landis) tree is a binary search tree with a balance condition.

An AVL tree is identical to a binary search tree, except that for every node in the tree, the height of the left and right subtrees can differ by at most 1.

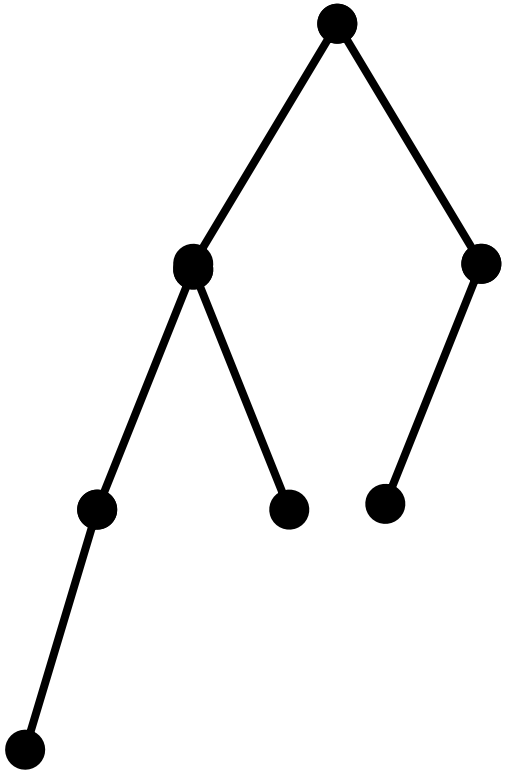
Is the following an AVL Tree?

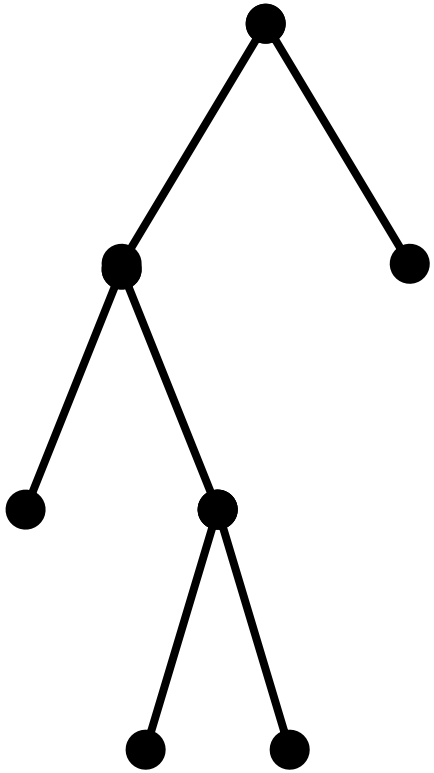


Is the following an AVL Tree?

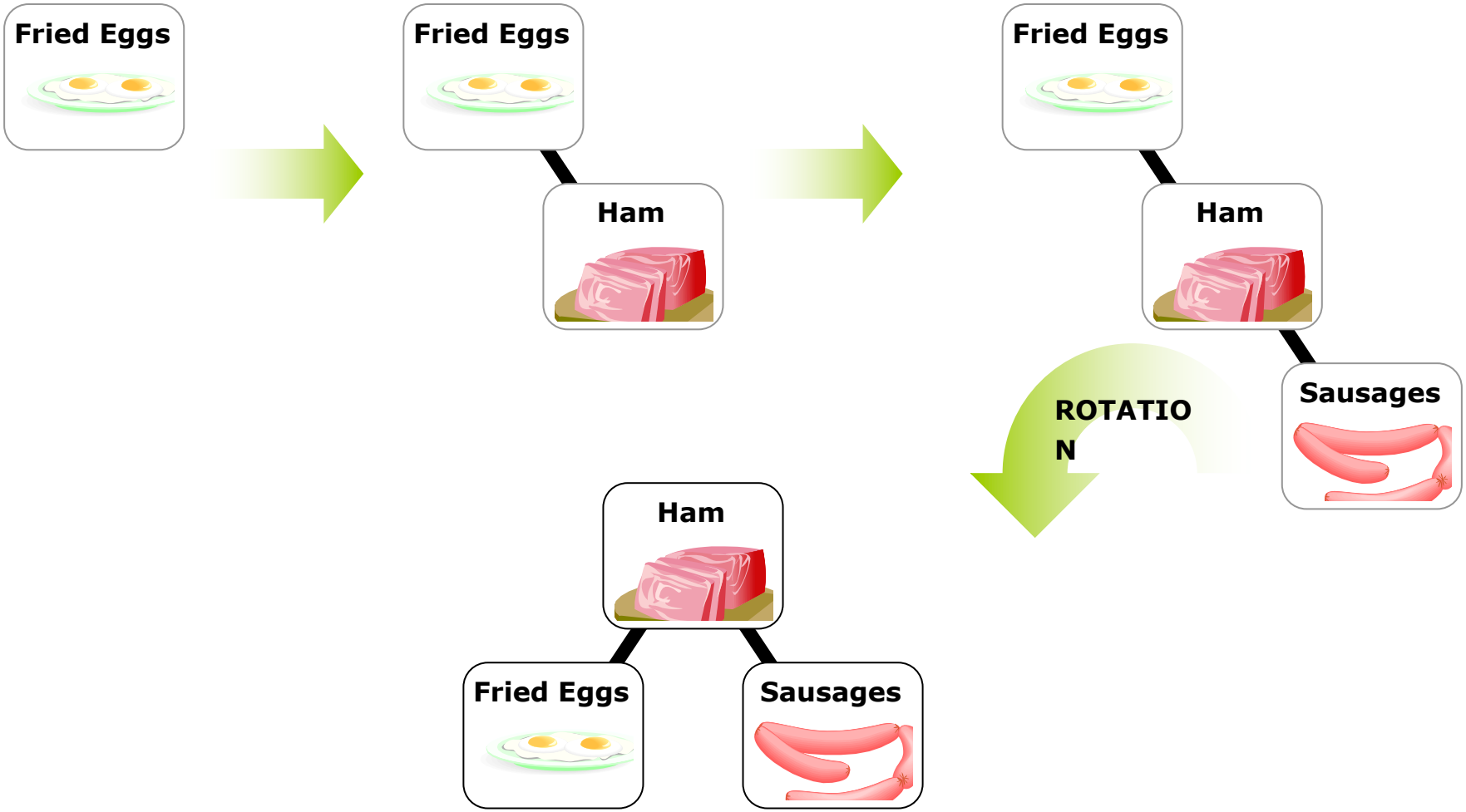


Question: Are the following AVL Trees?





Binary Search Tree Balancing Strategy: Rotation



Single Rotation

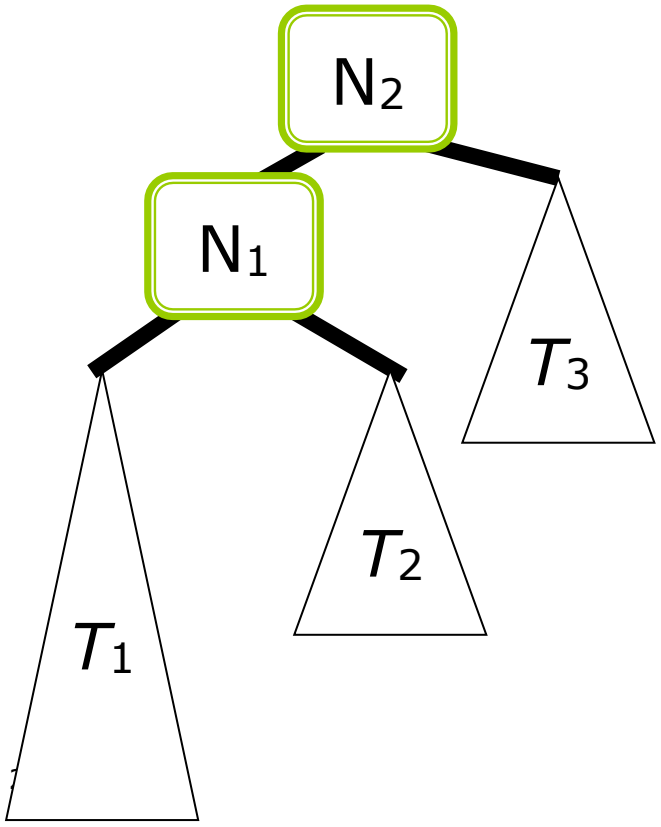
Observation

A rotation is needed **only when** a new node is inserted into an AVL Tree.

WHY?

Single Rotation

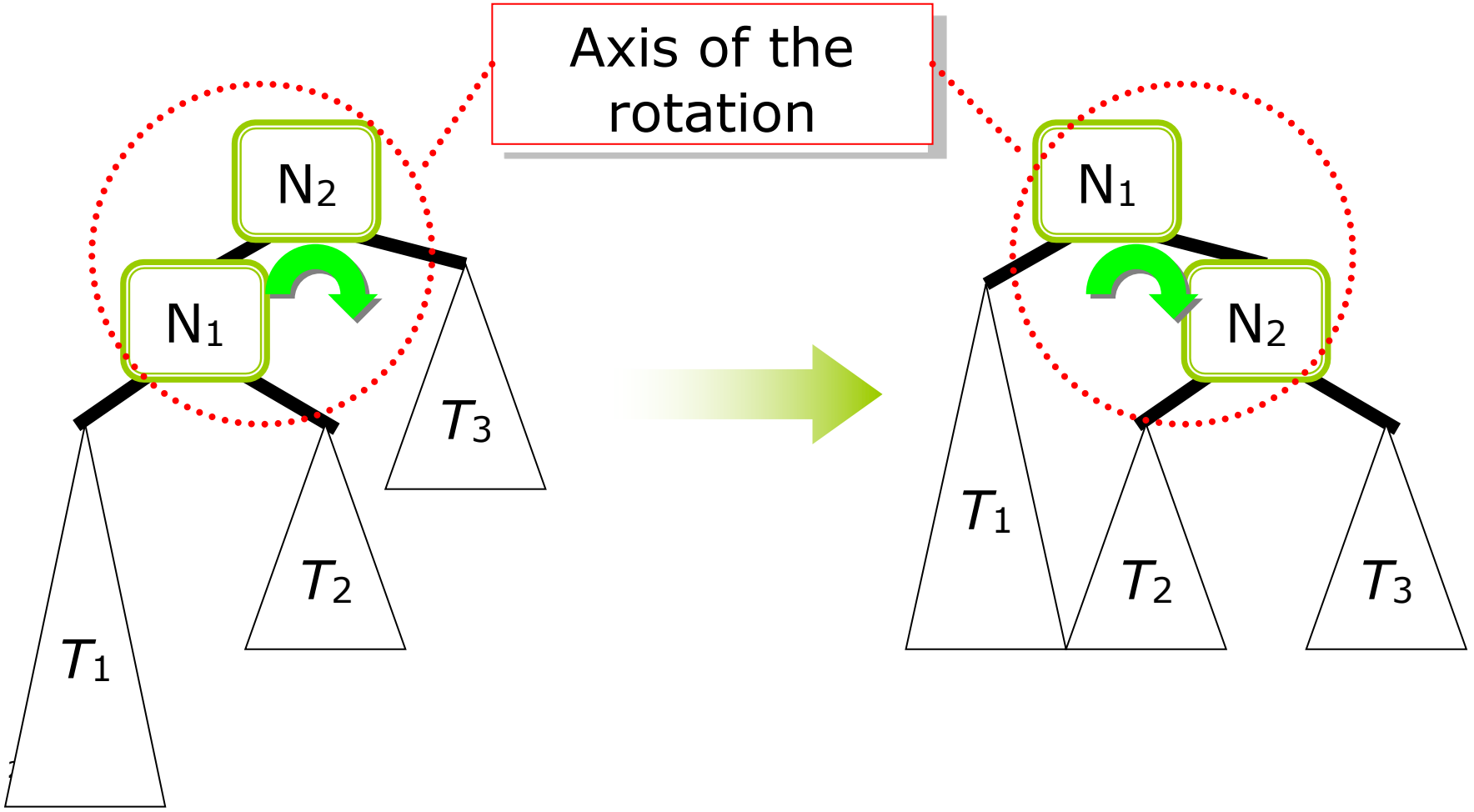
First, we consider when a new node is inserted into the left subtree of the left child.



Node N_2 violates the AVL balance property after a new node is inserted into T_1.

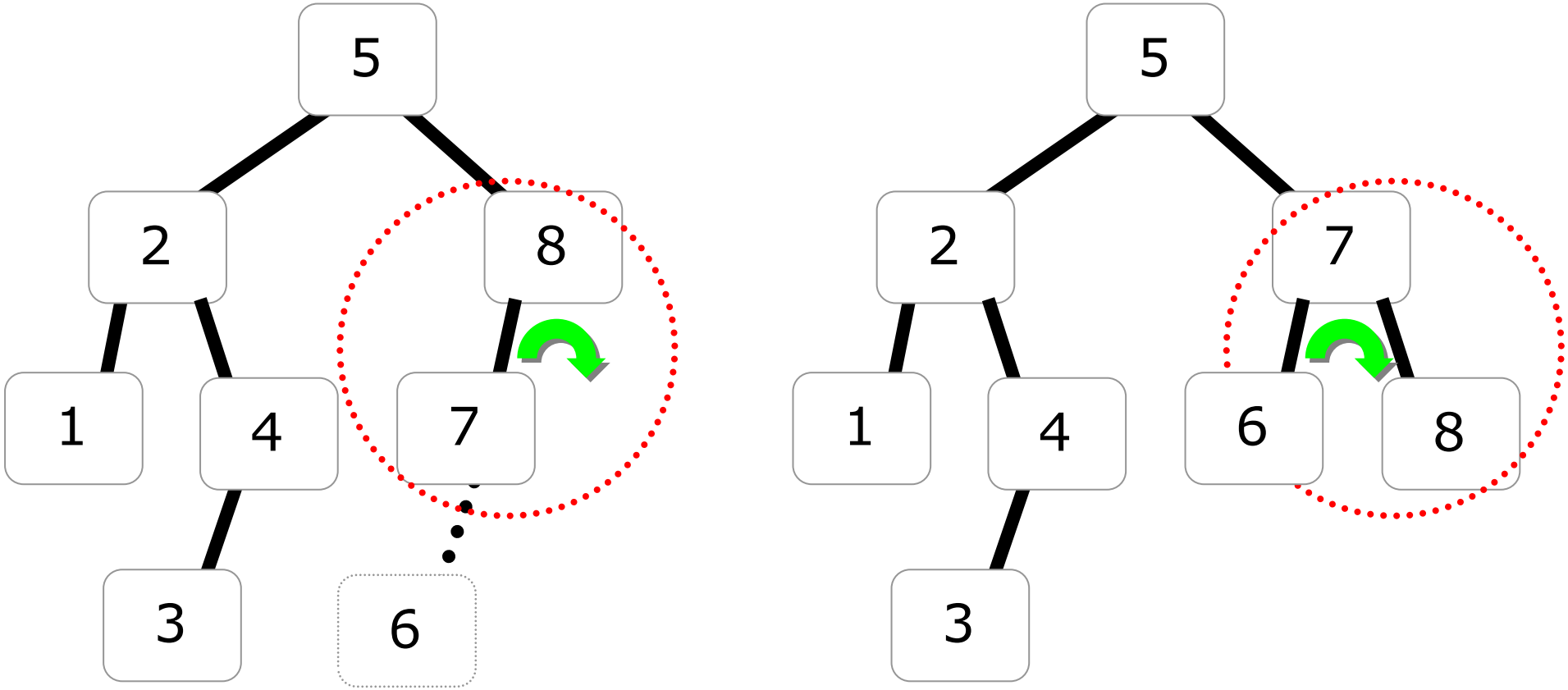
Single Rotation

The tree is rebalanced by a single right rotation.



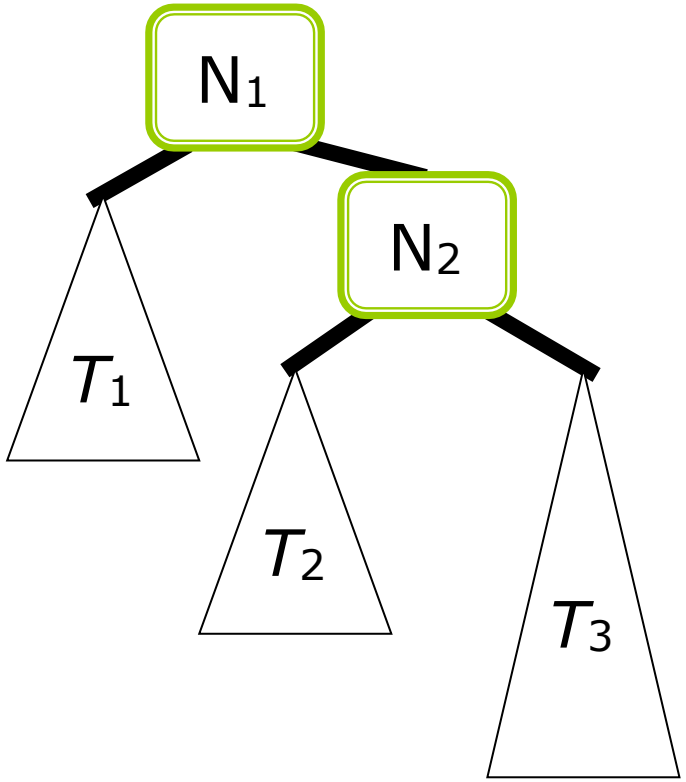
Single Rotation

Example



Single Rotation

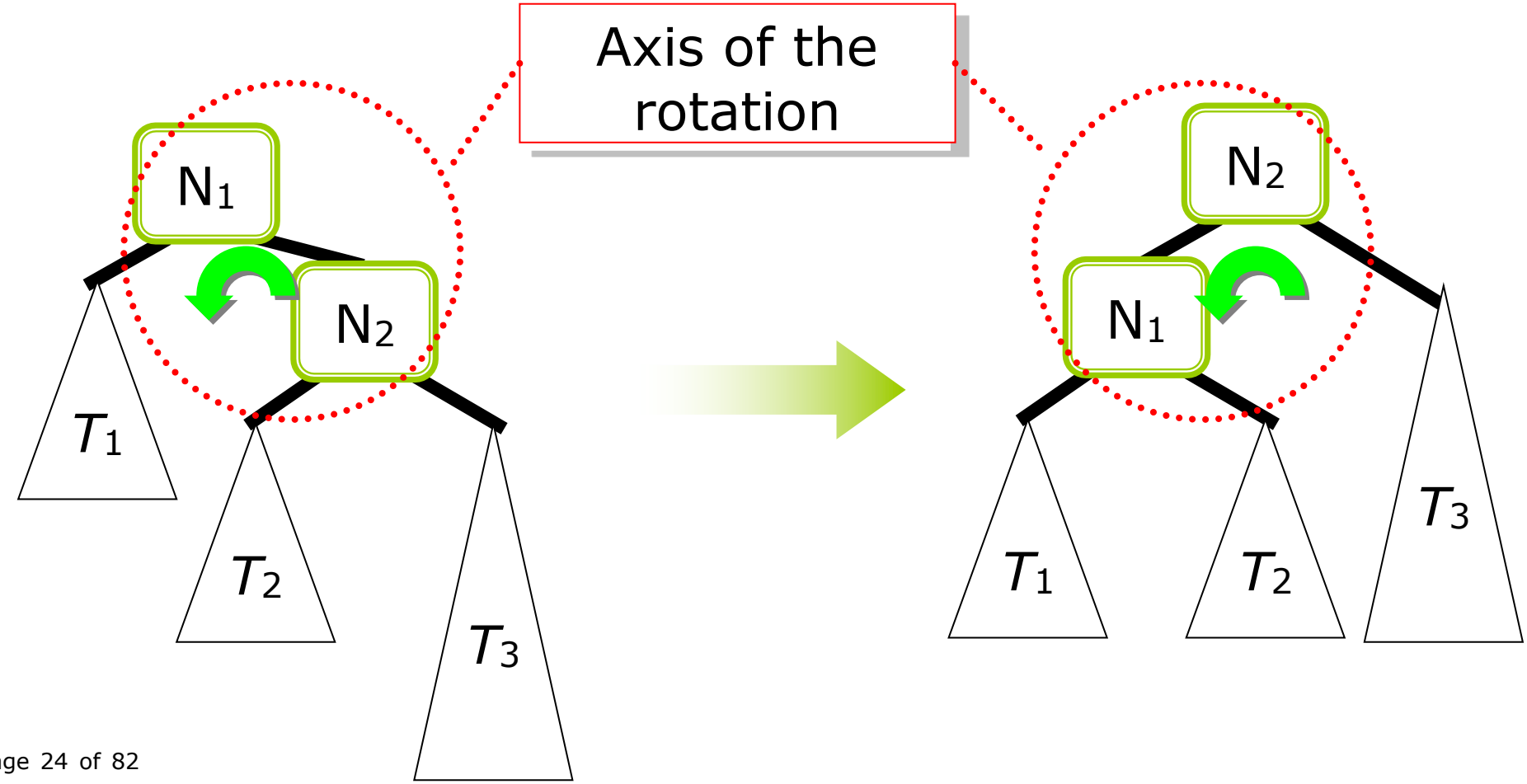
Second, we consider the symmetric case when a new node is inserted into the right subtree of the right child.



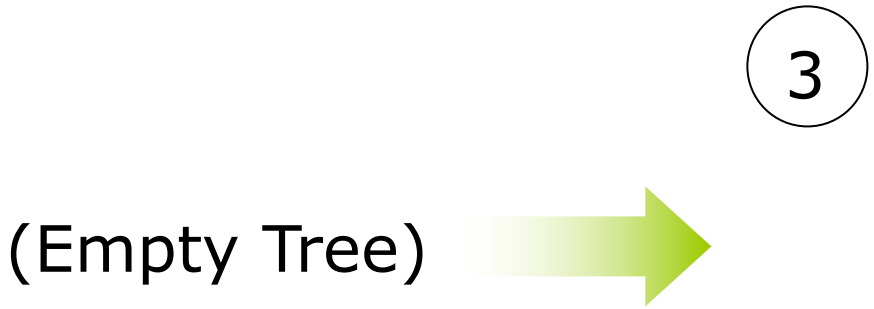
Node N_1 violates the AVL balance property after a new node is inserted into T_3.

Single Rotation

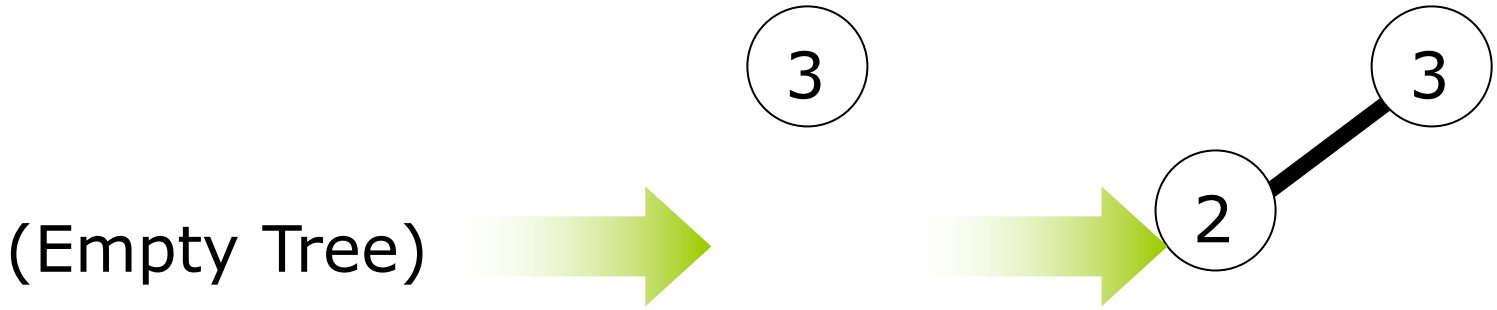
The tree is rebalanced by a single right rotation.



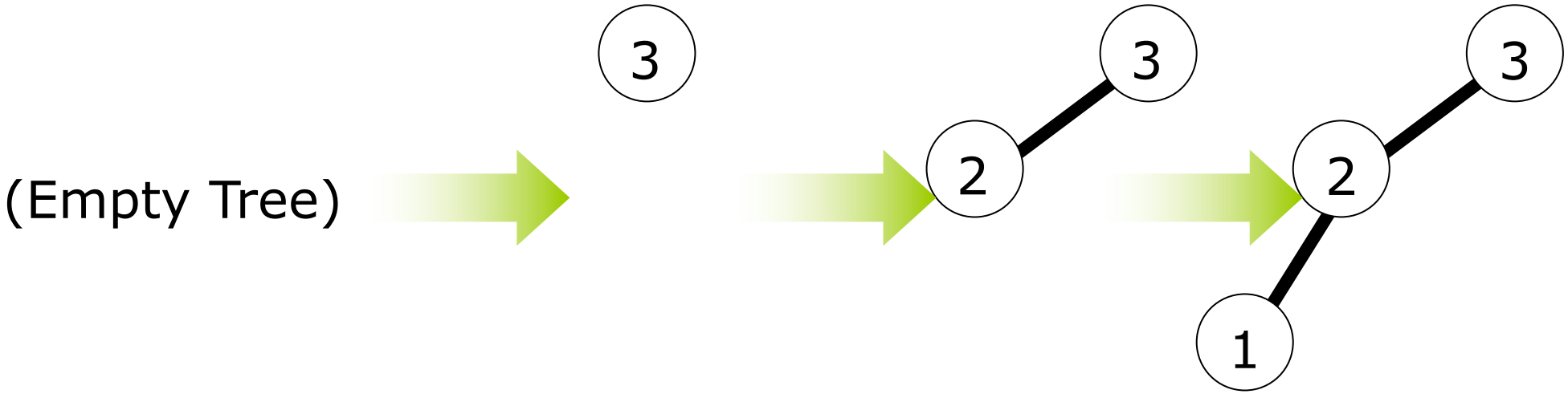
We now consider a longer example. We first insert the nodes with keys 3, 2 and 1.



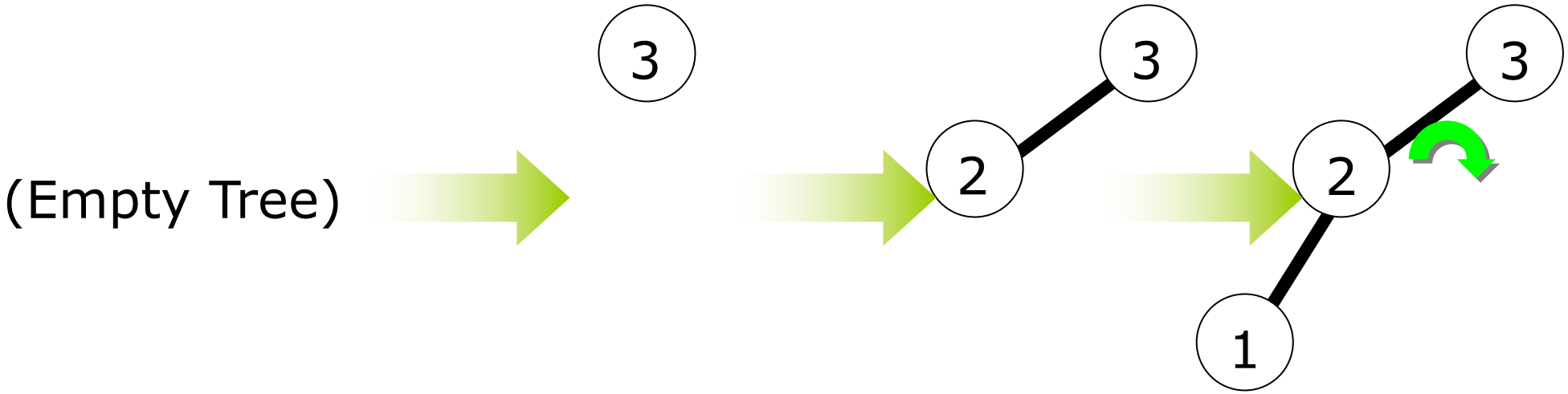
We now consider a longer example. We first insert the nodes with keys 3, 2 and 1.

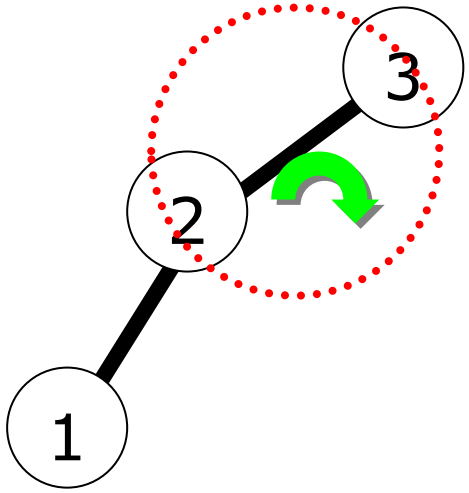


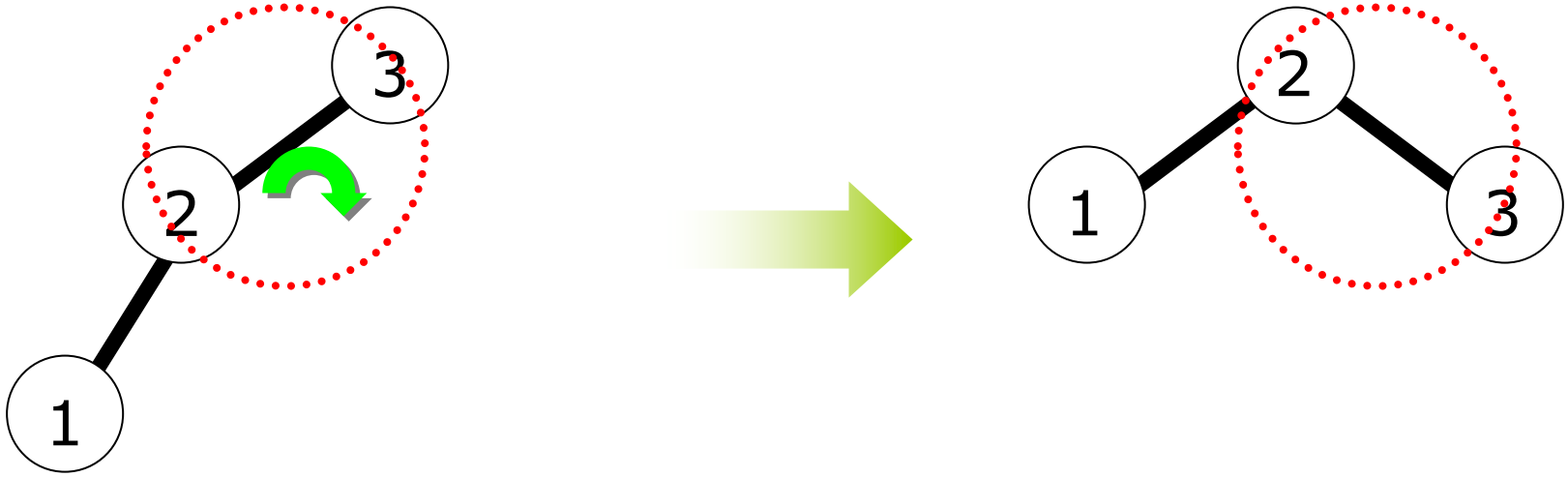
We now consider a longer example. We first insert the nodes with keys 3, 2 and 1.

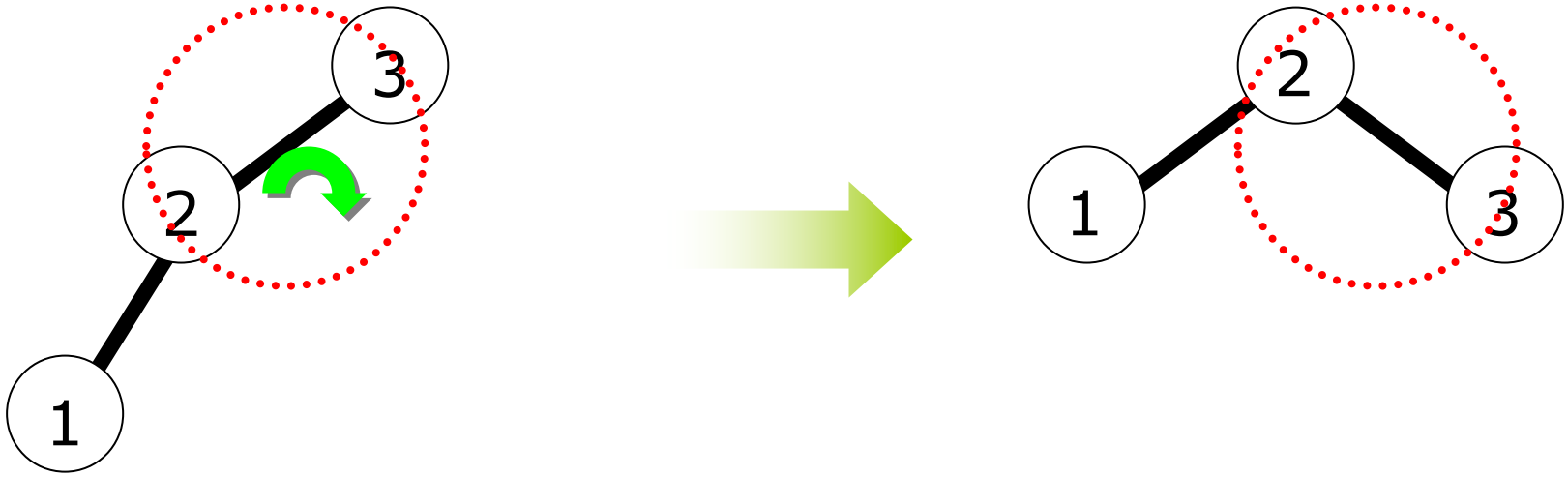


We now consider a longer example. We first insert the nodes with keys 3, 2 and 1.

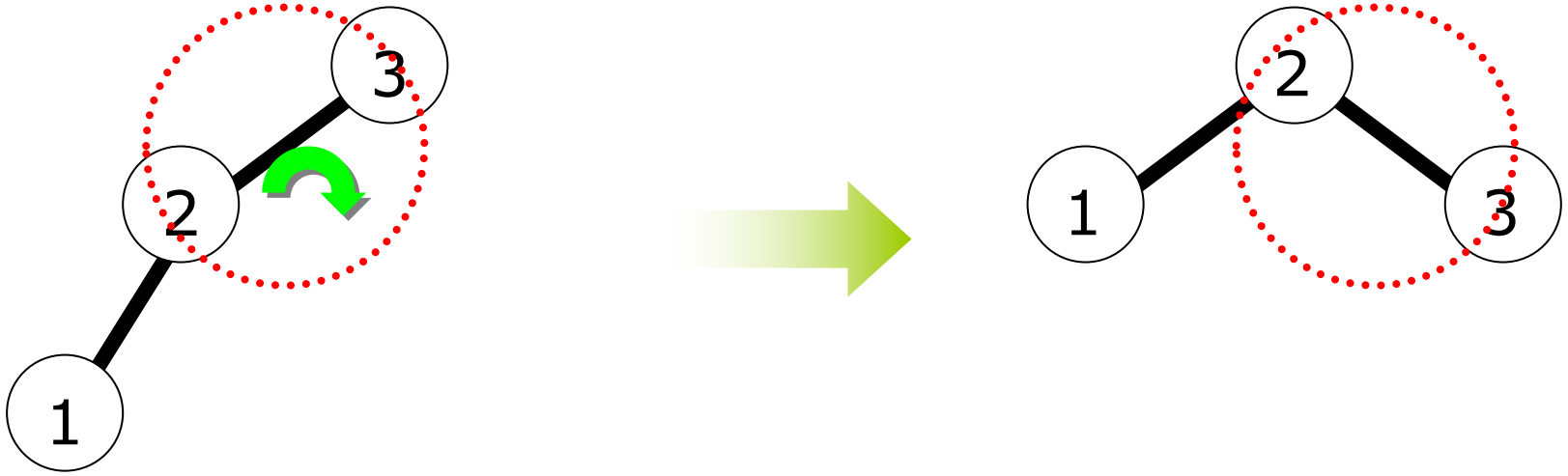



3
2
1

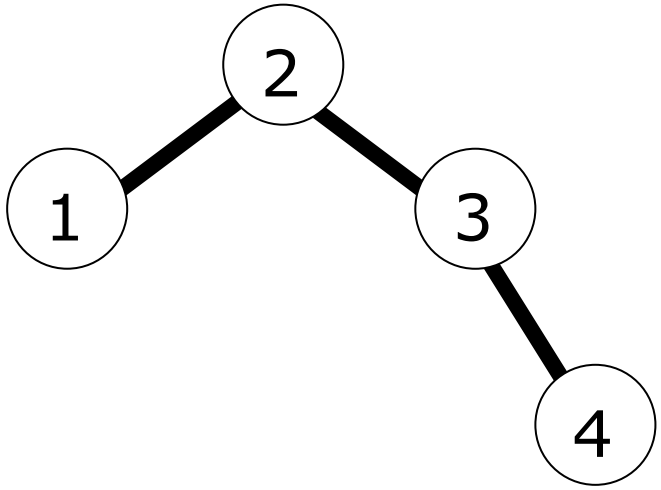

3
2
1
2
1
3

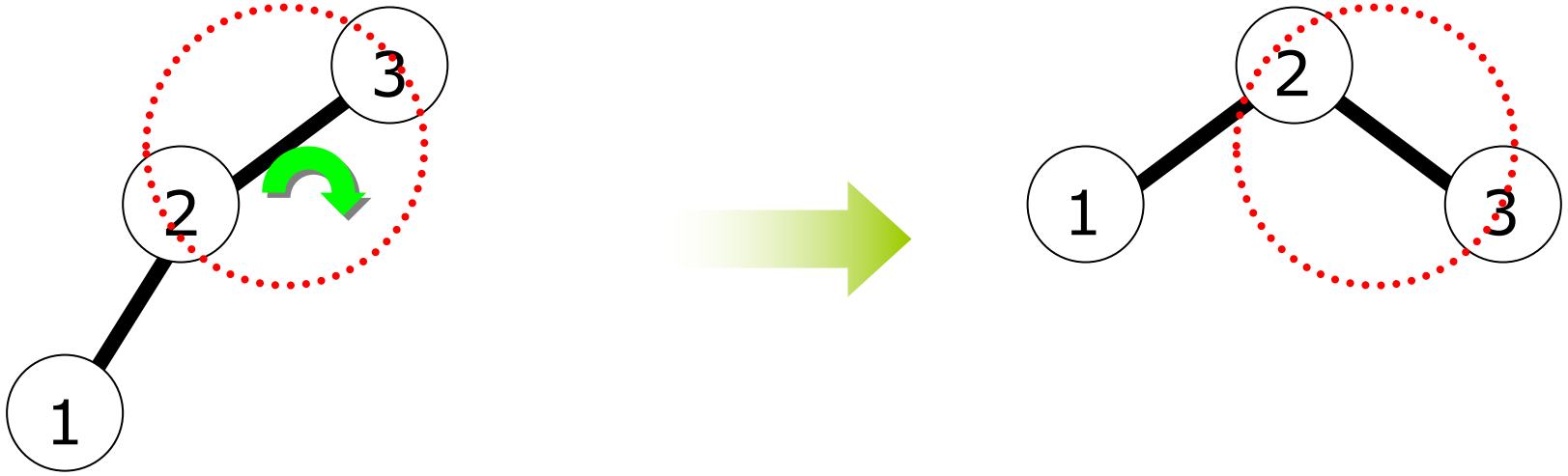


Next, we insert nodes with keys 4 and 5.

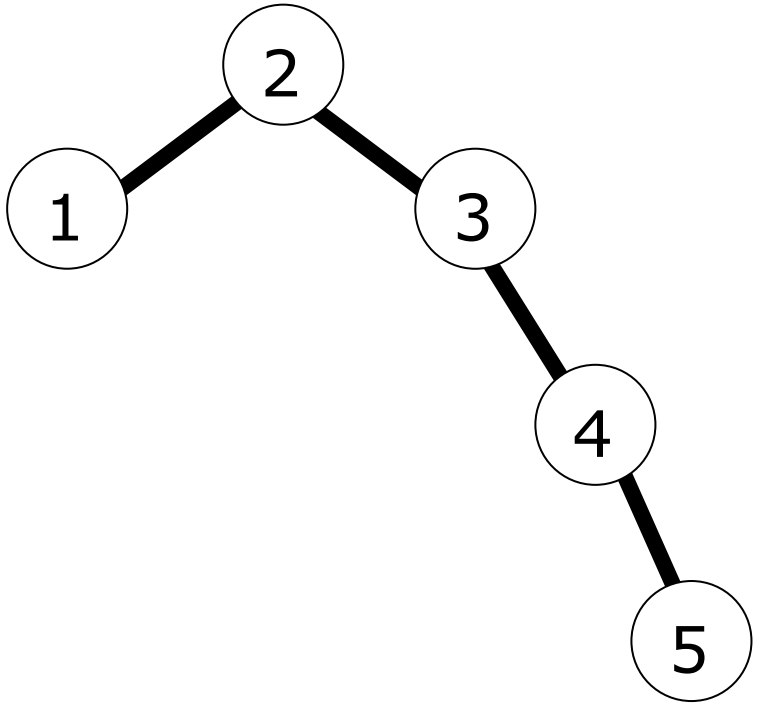


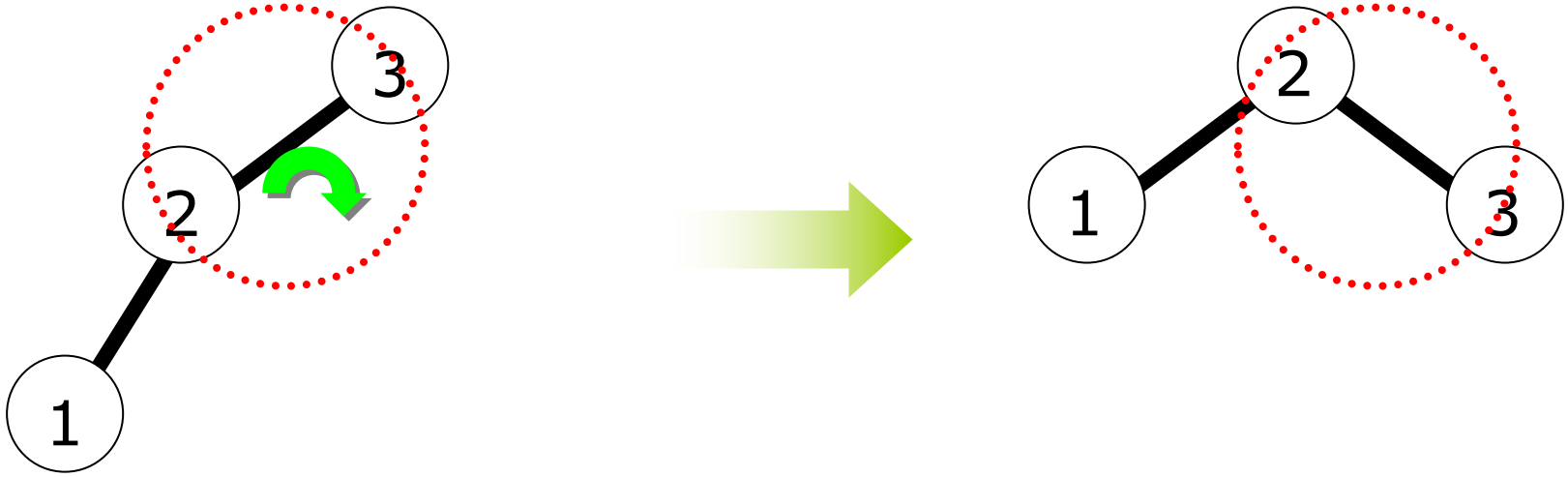
Next, we insert nodes with keys 4 and 5.



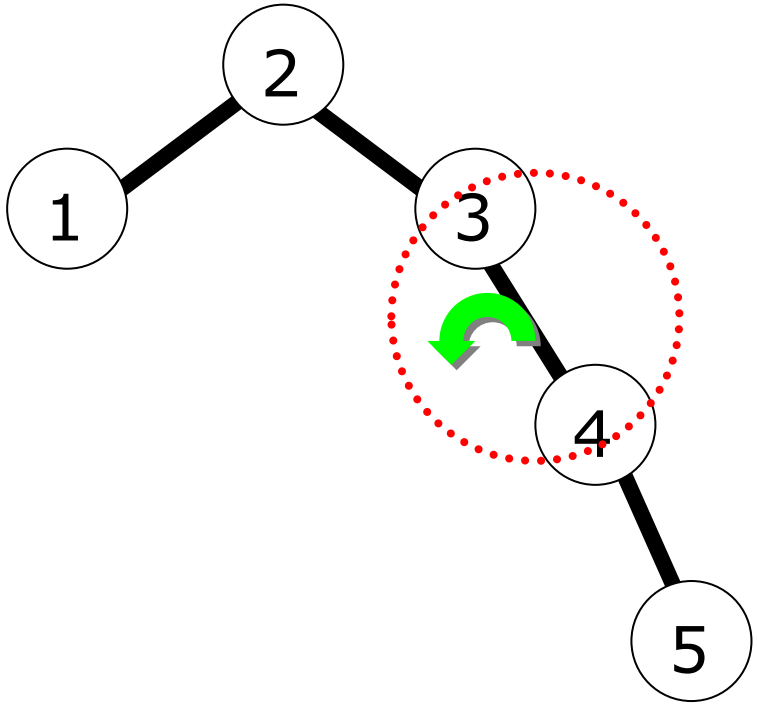


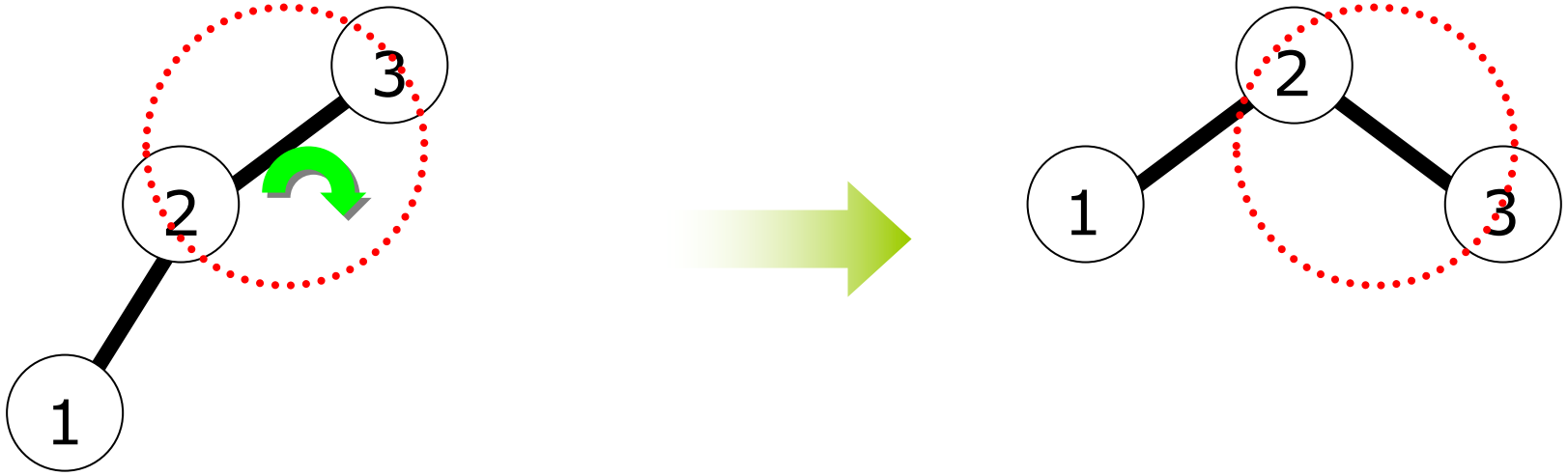
Next, we insert nodes with keys 4 and 5.



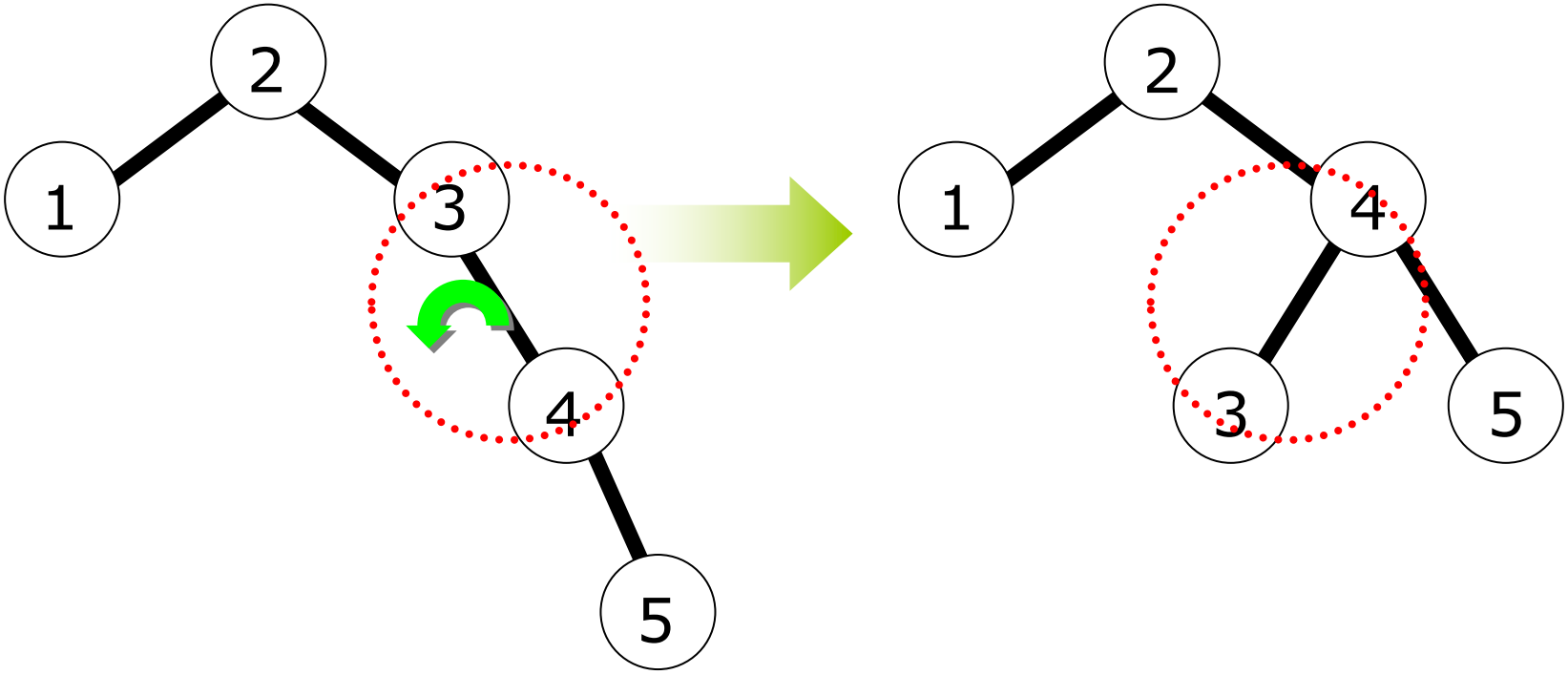


Next, we insert nodes with keys 4 and 5.

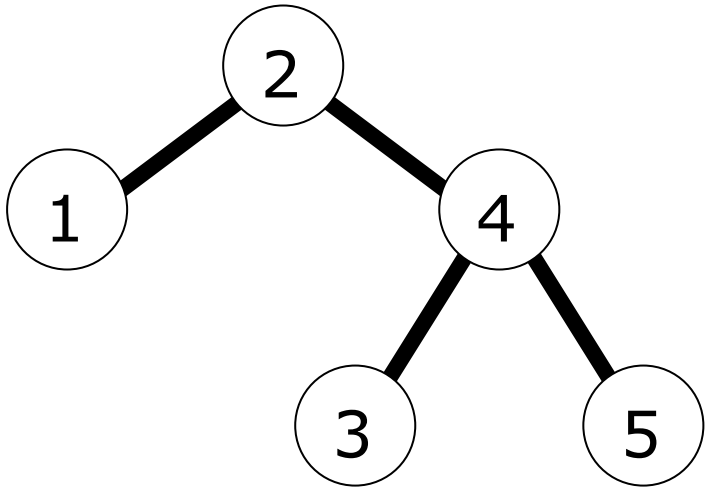




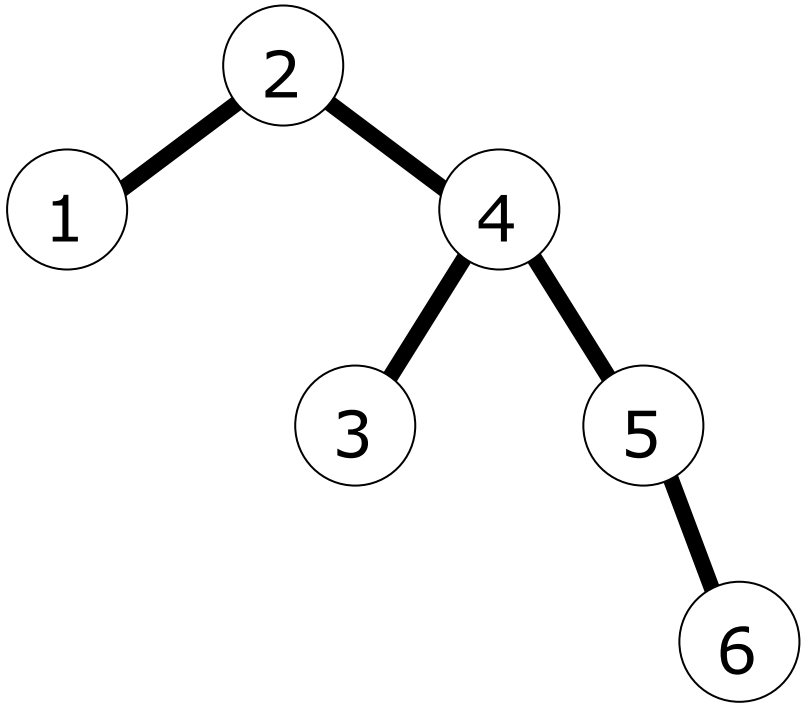
Next, we insert nodes with keys 4 and 5.



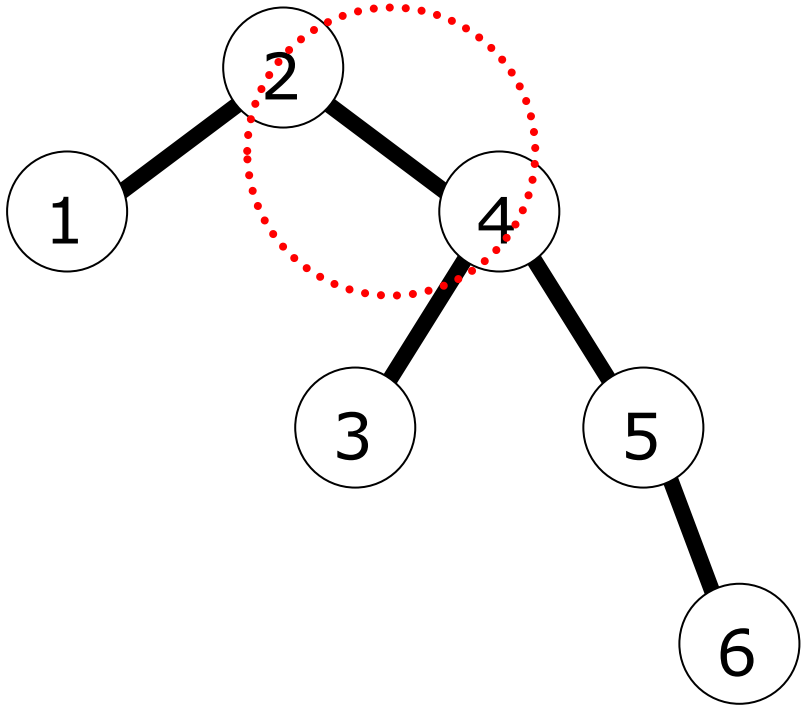
Next, we insert a node with key 6.



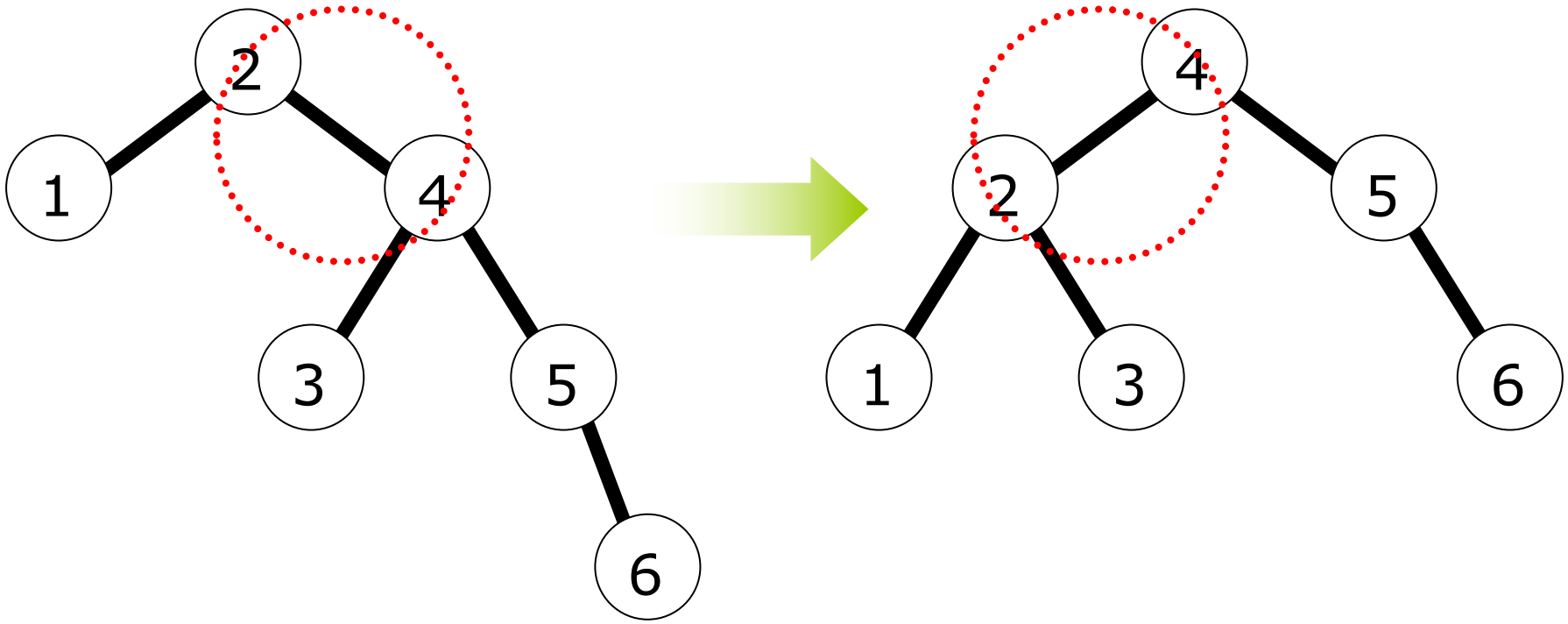
Next, we insert a node with key 6.



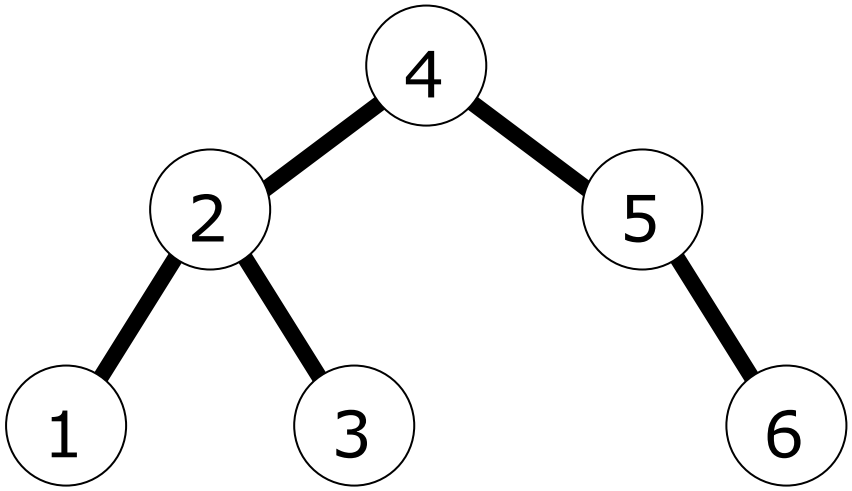
Next, we insert a node with key 6.



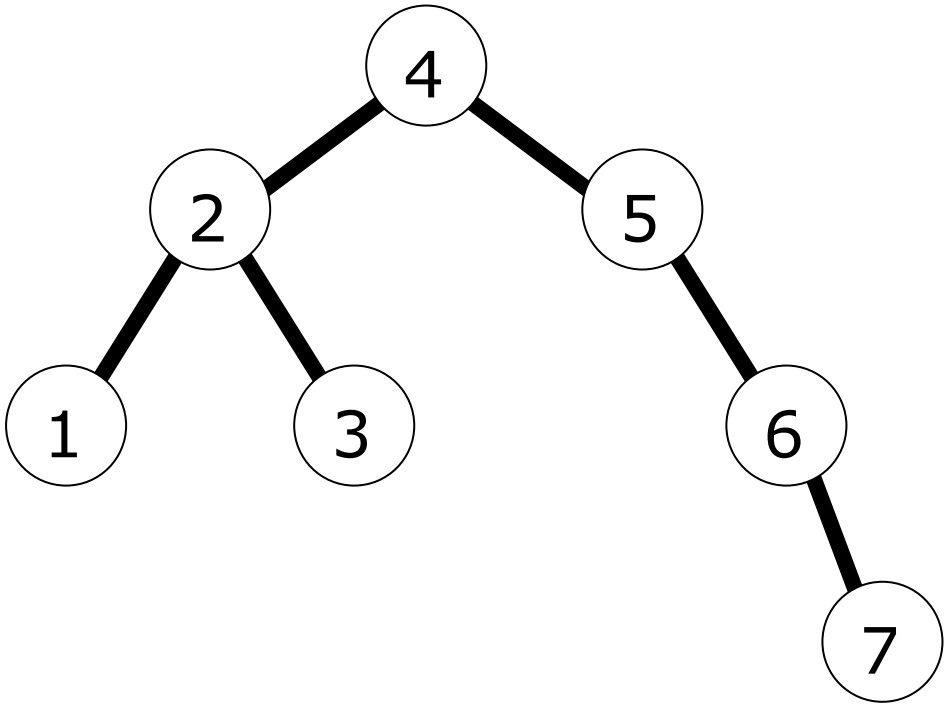
Next, we insert a node with key 6.



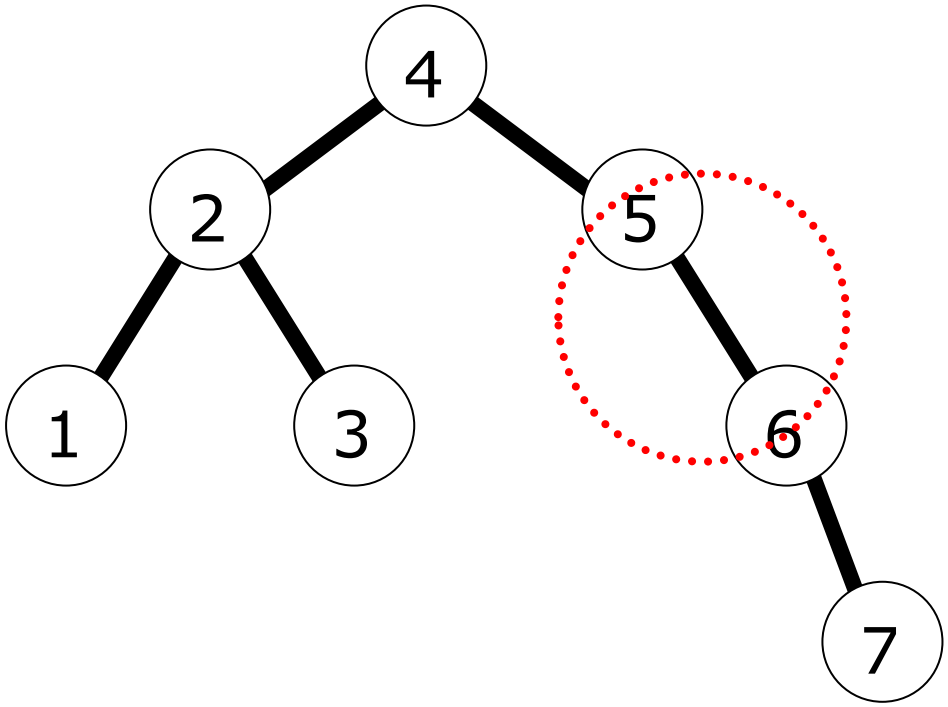
Finally, we insert a node with key 7.



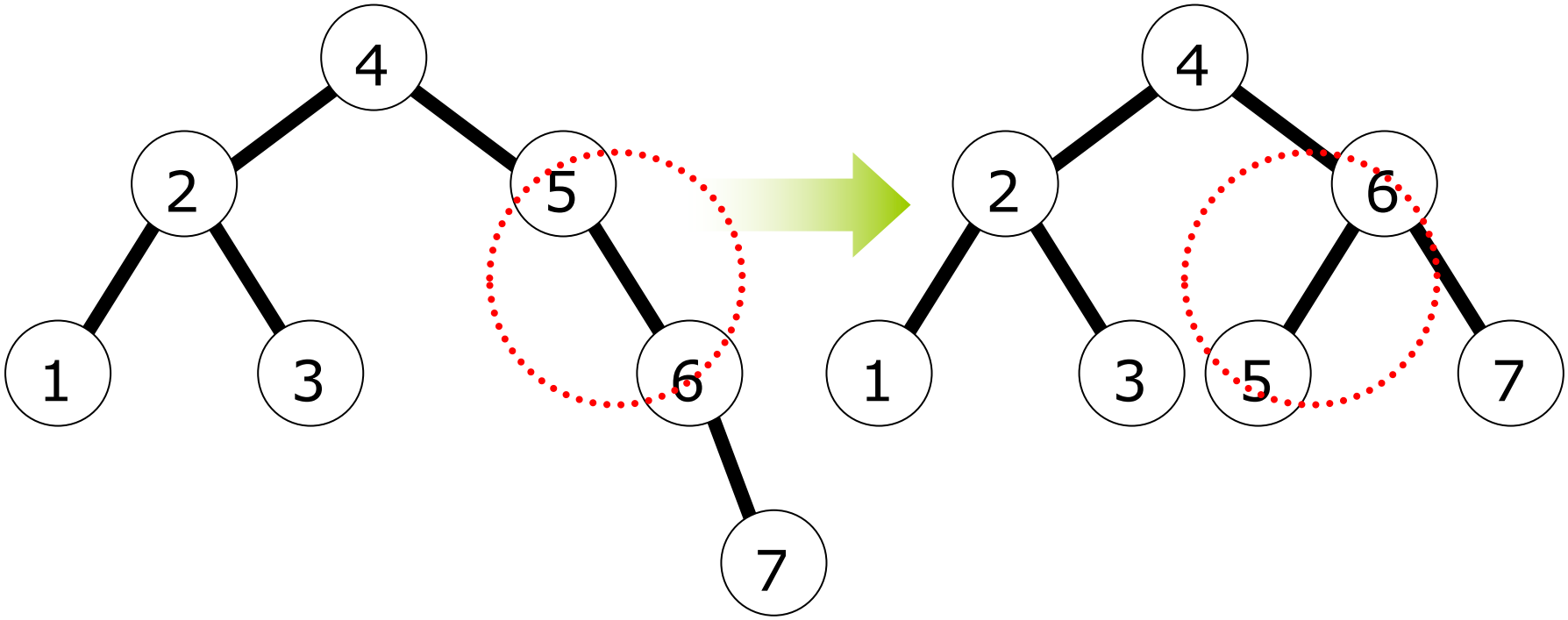
Finally, we insert a node with key 7.



Finally, we insert a node with key 7.

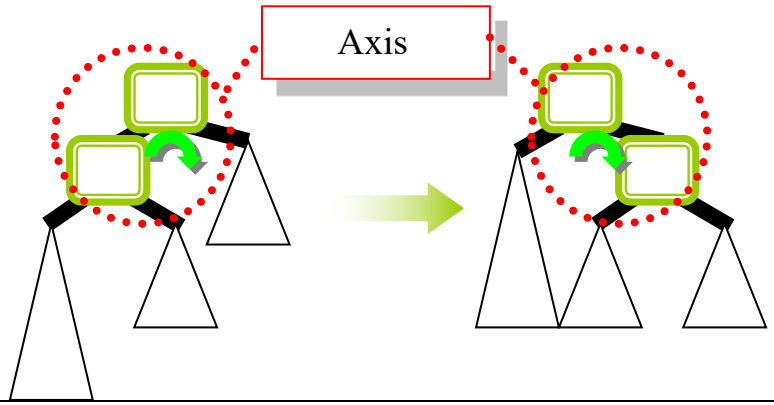
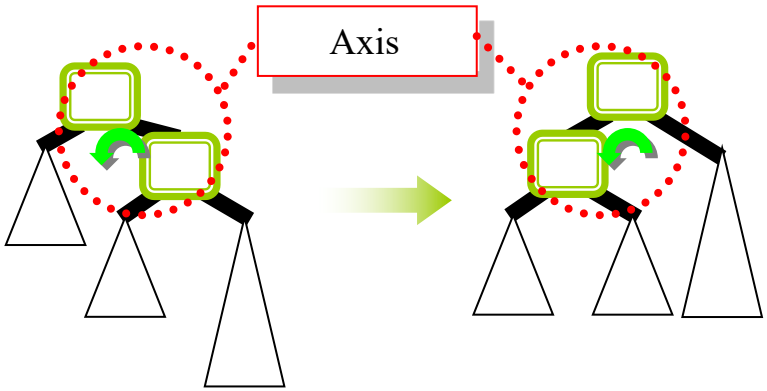


Finally, we insert a node with key 7.

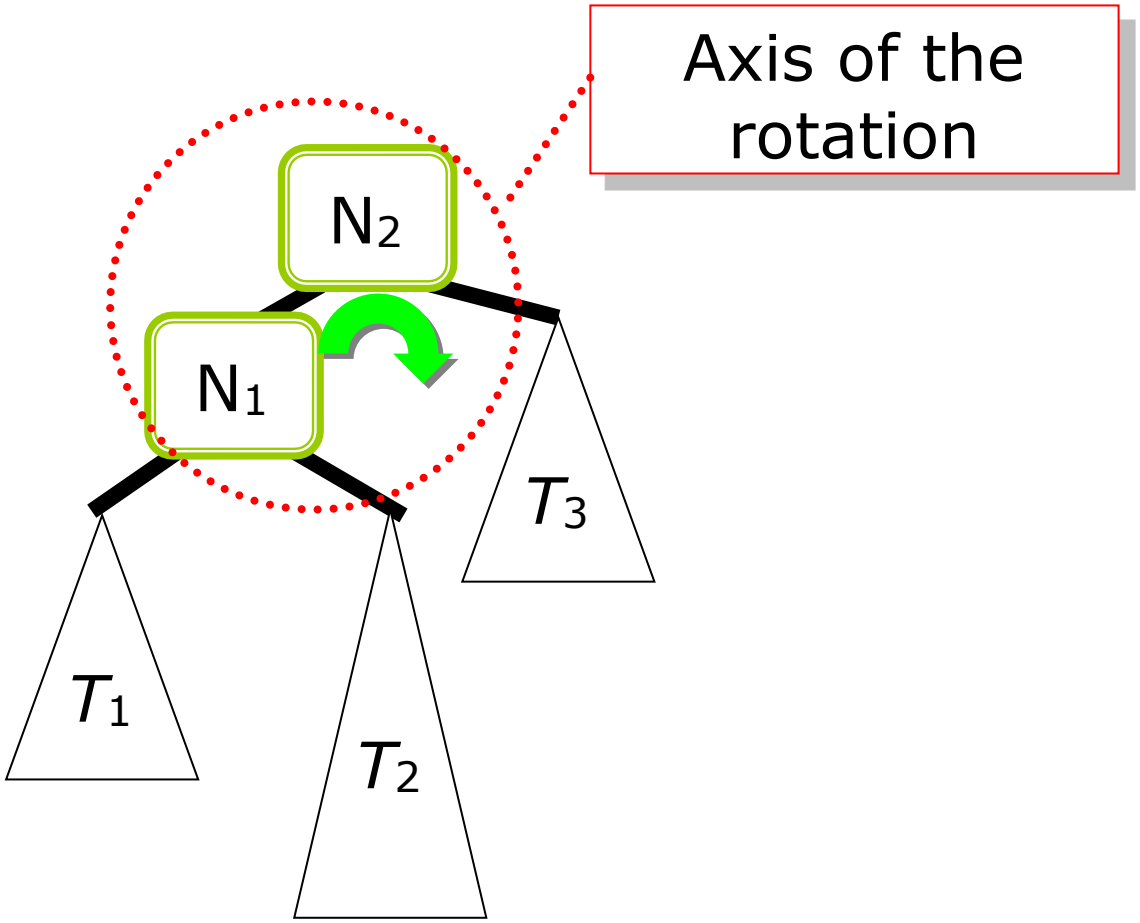


So far so good!

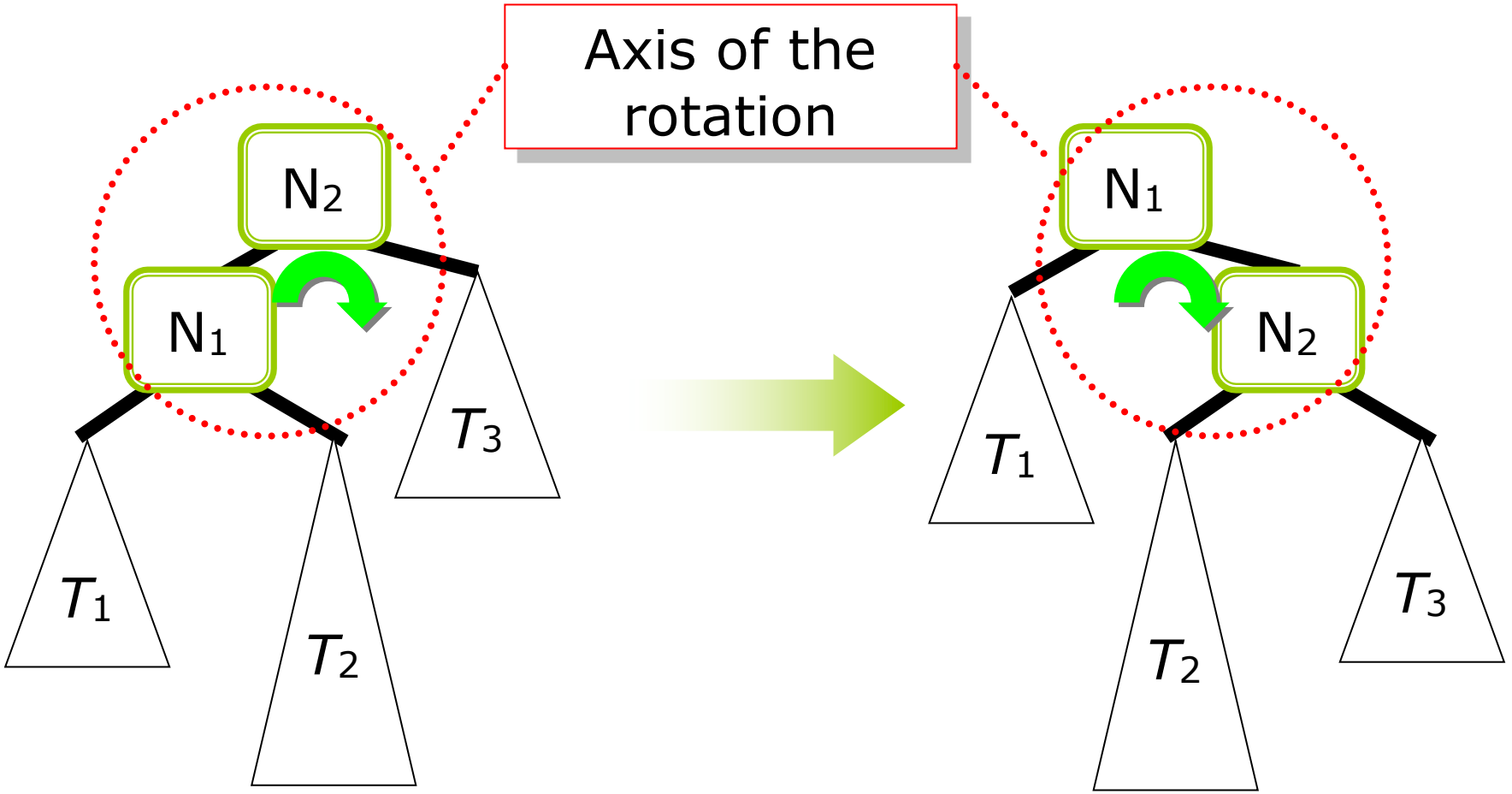
We have so far considered two cases:

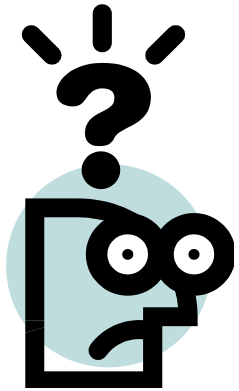
Case 1	**Case 2**
A new node is inserted into the left subtree of the left child.	A new node is inserted into the right subtree of the right child.
	

Now we consider when a new node is inserted into the right subtree of the left child.

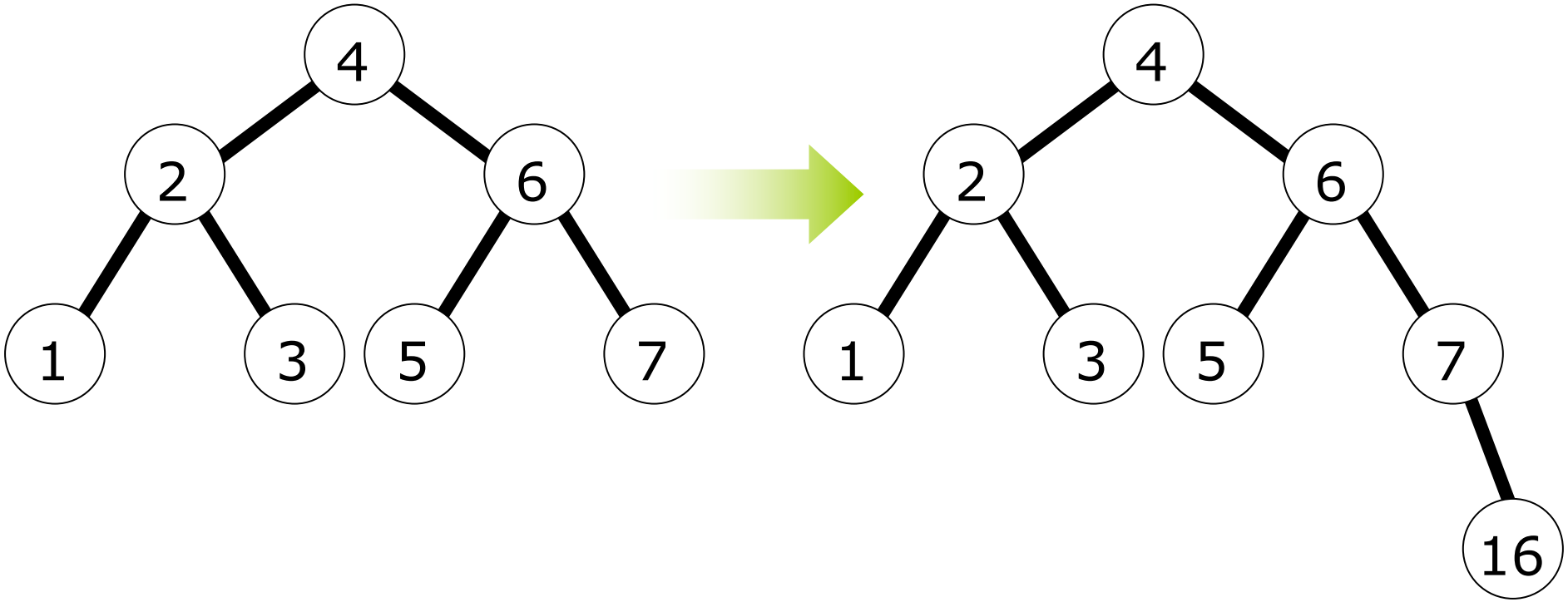


Now we consider when a new node is inserted into the right subtree of the left child.

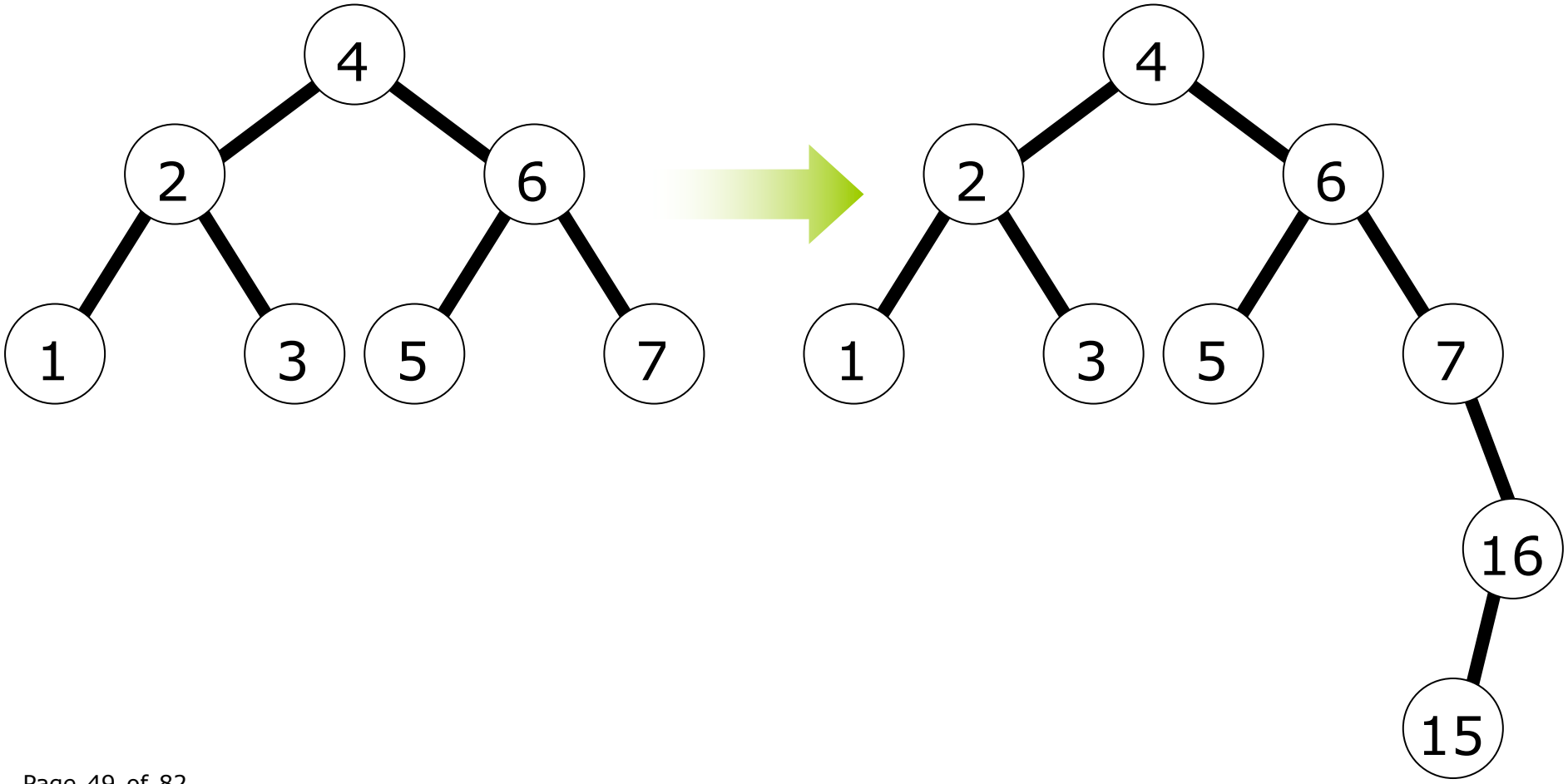




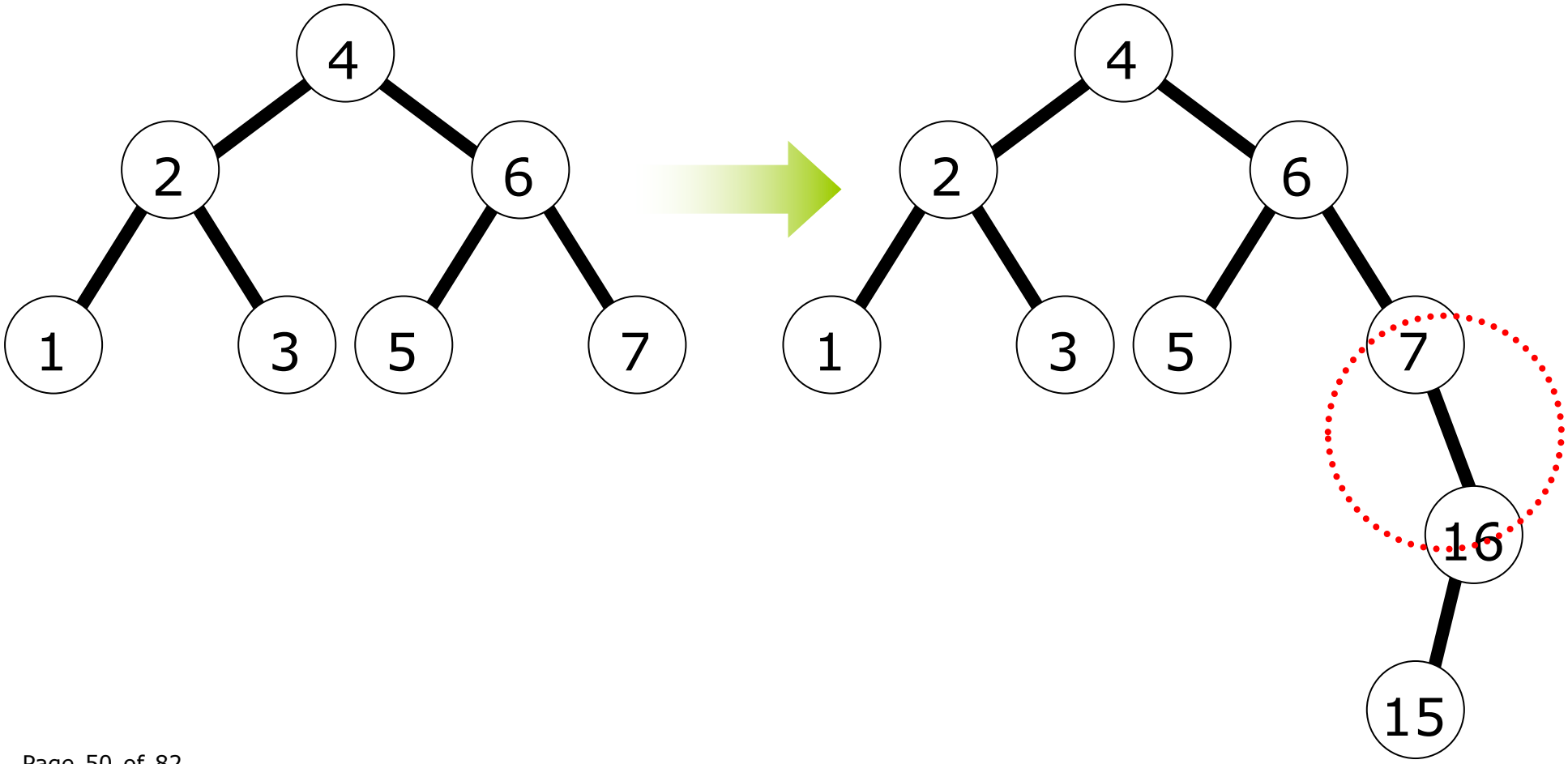
Let’s continue with the previous example, and insert nodes with keys 16 and 15.



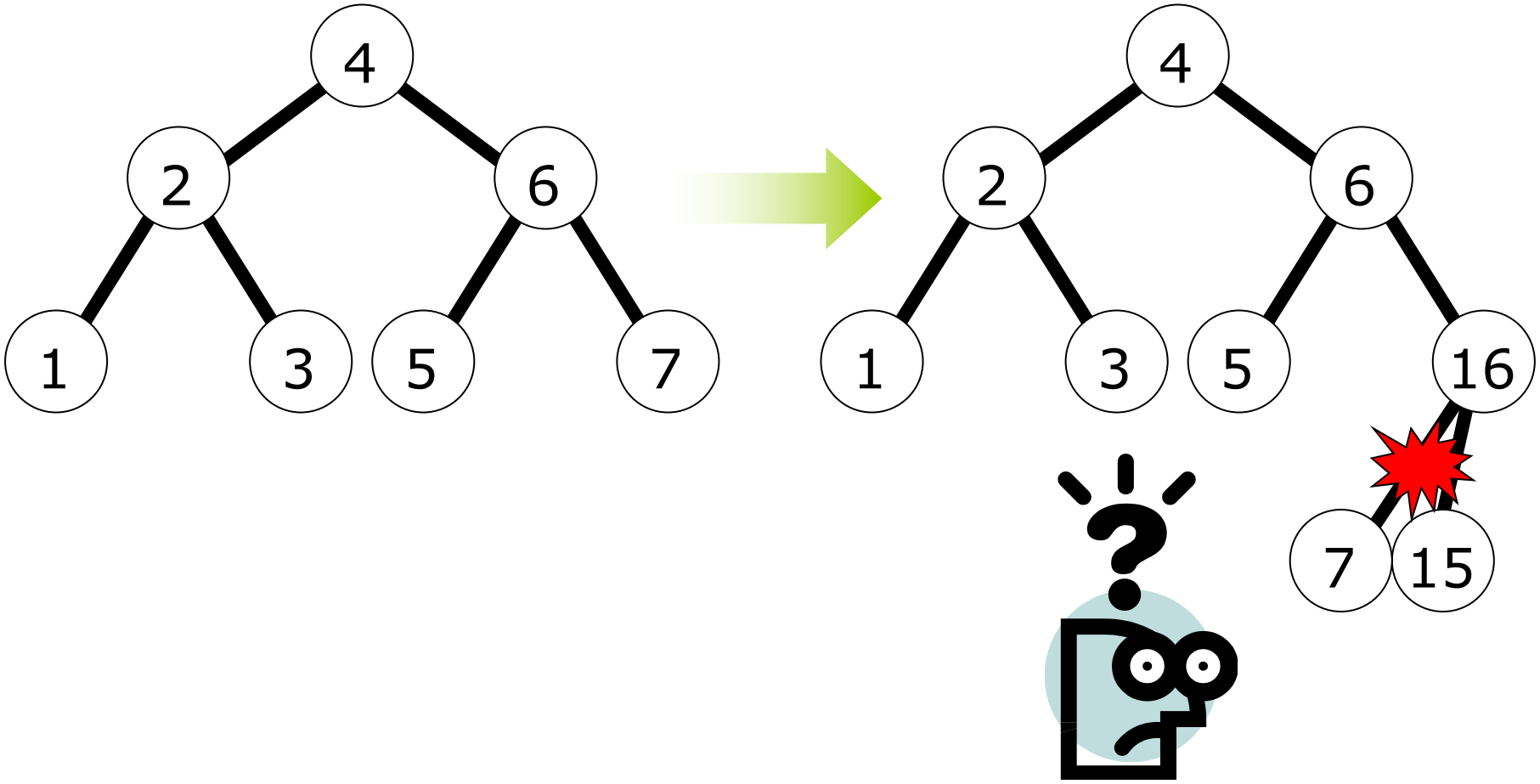
Let’s continue with the previous example, and insert nodes with keys 16 and 15.



Let’s continue with the previous example, and insert nodes with keys 16 and 15.

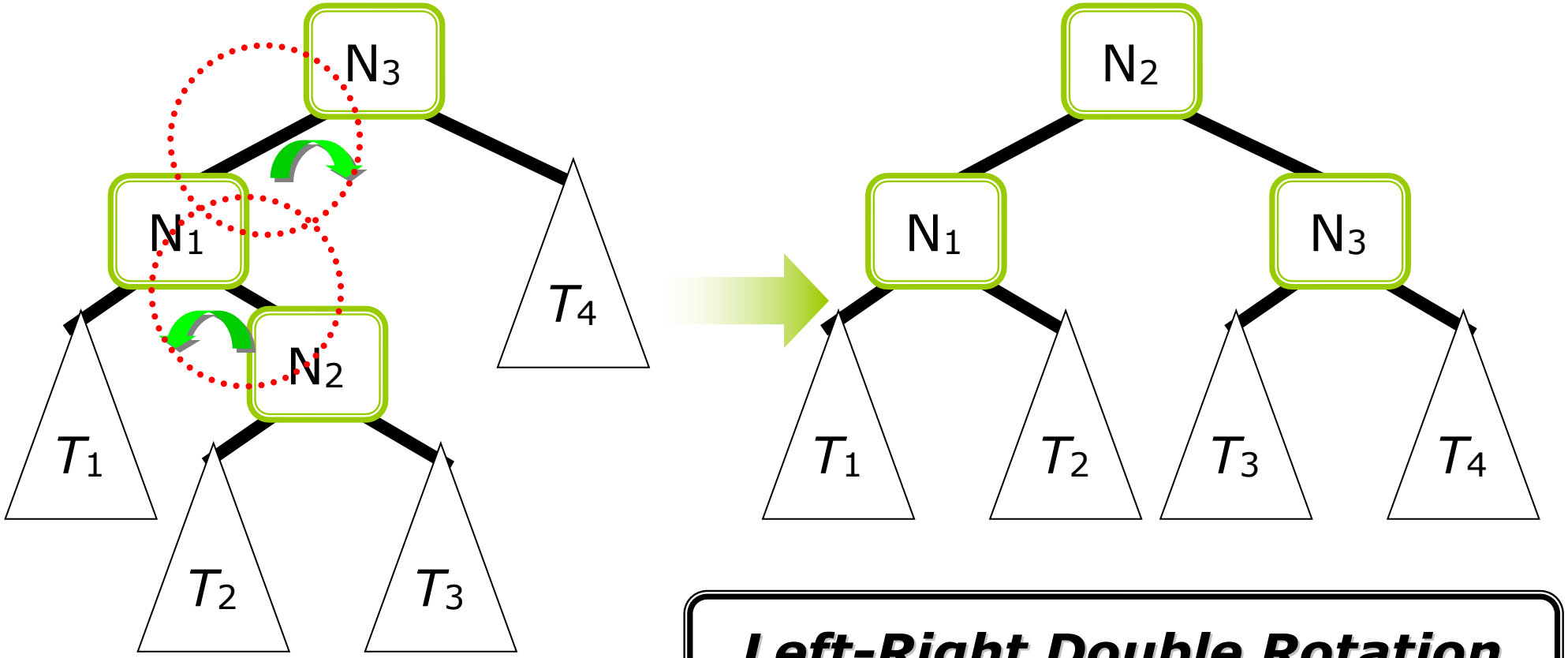


We see that single rotation does not work!



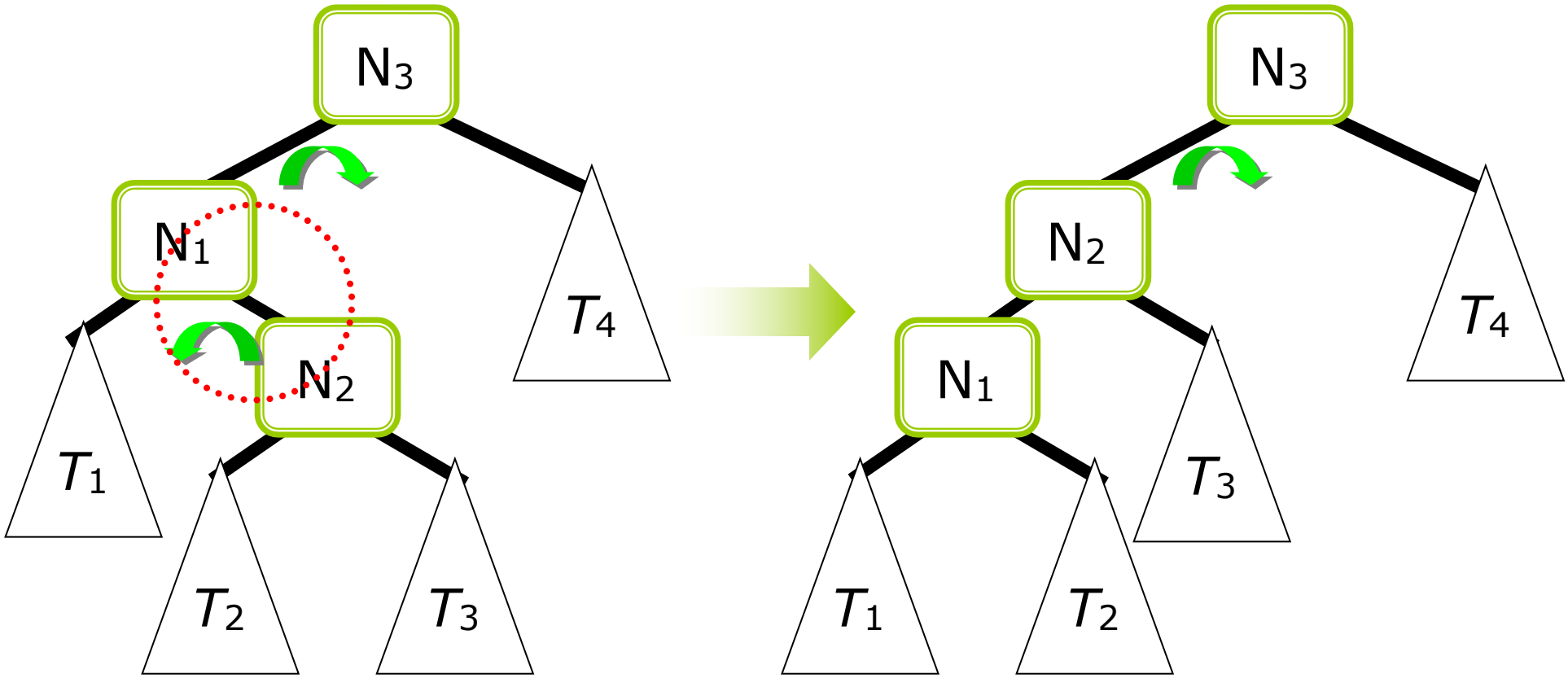
Double Rotation

We consider again when a new node is inserted into the right subtree of the left child.

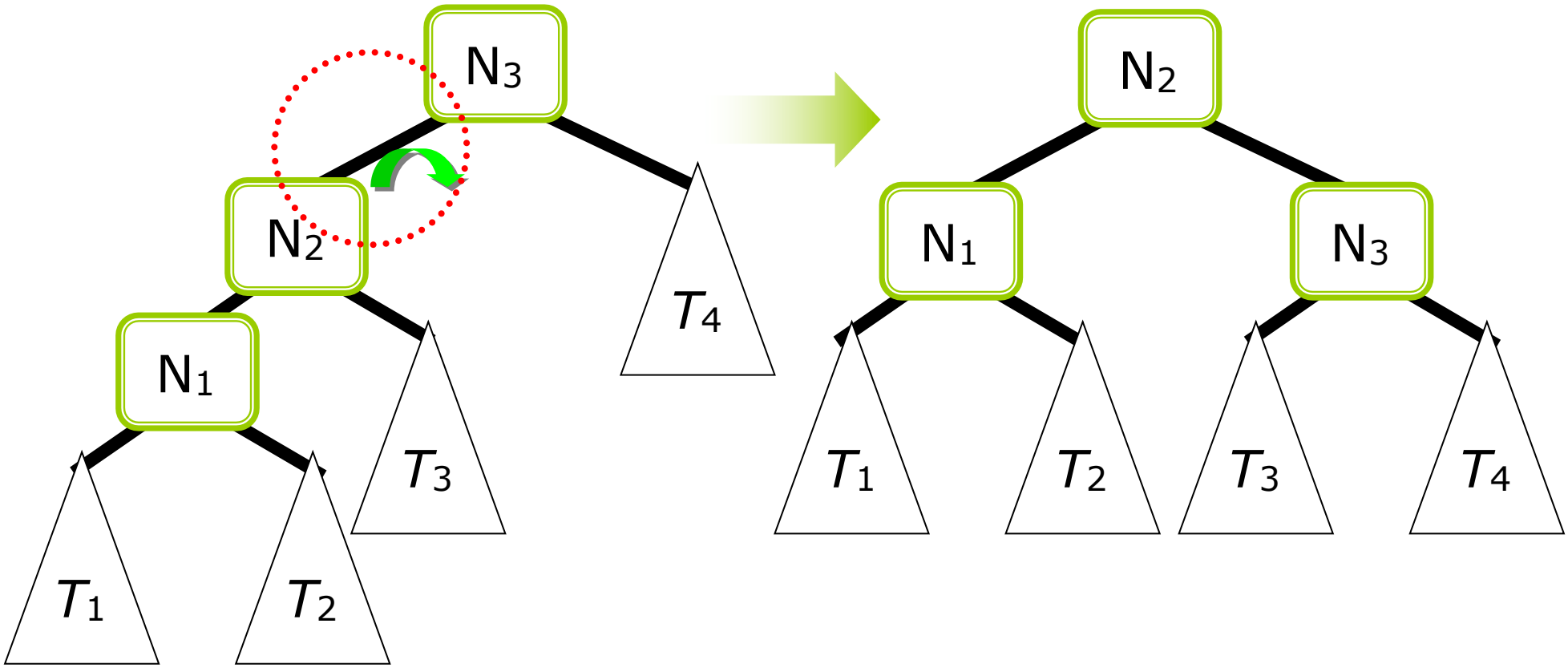


Left-Right Double Rotation

Step 1 of a **Left**-Right Double Rotation:



Step 2 of a Left-**Right** Double Rotation:

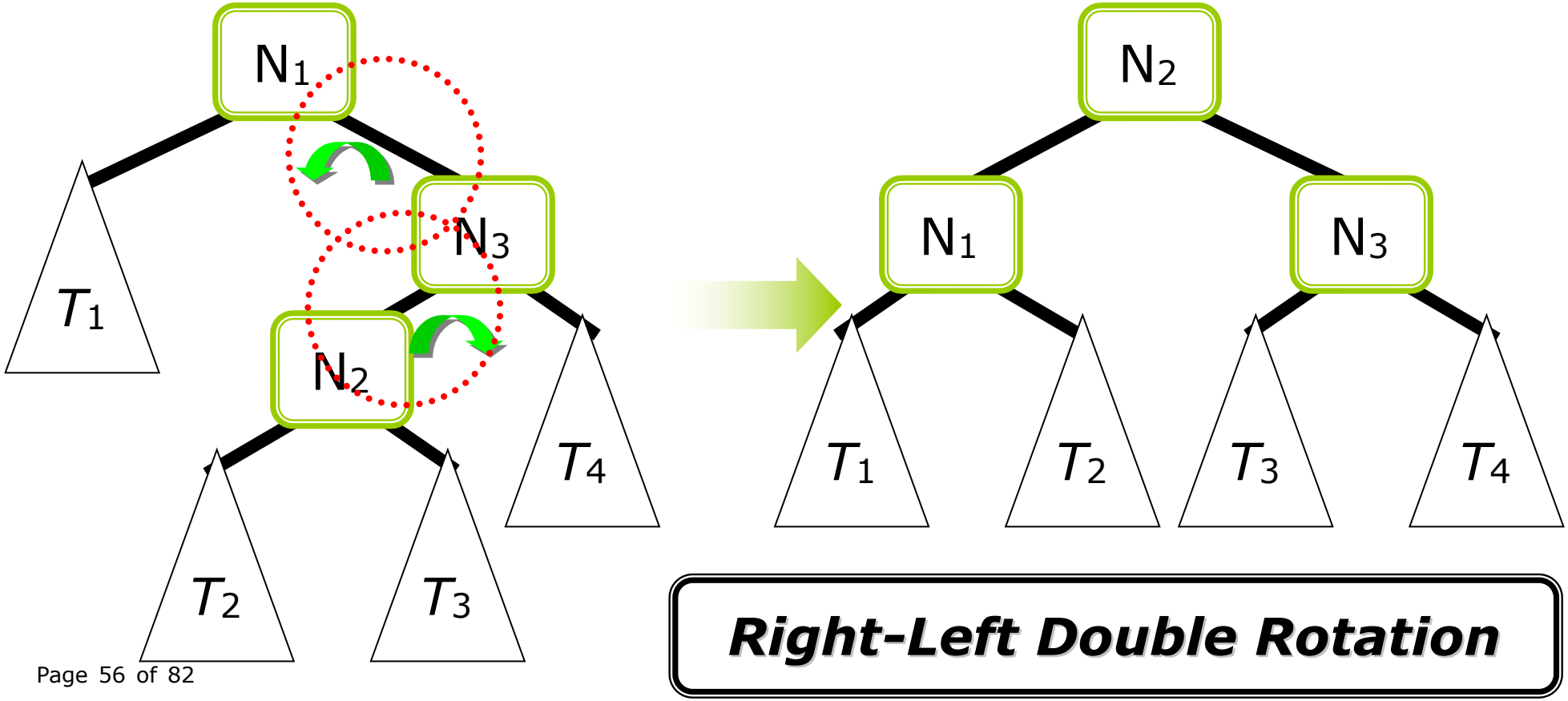


So, a Left-Right Double Rotation means a Left Single Rotation followed by a Right Single Rotation on one level higher.

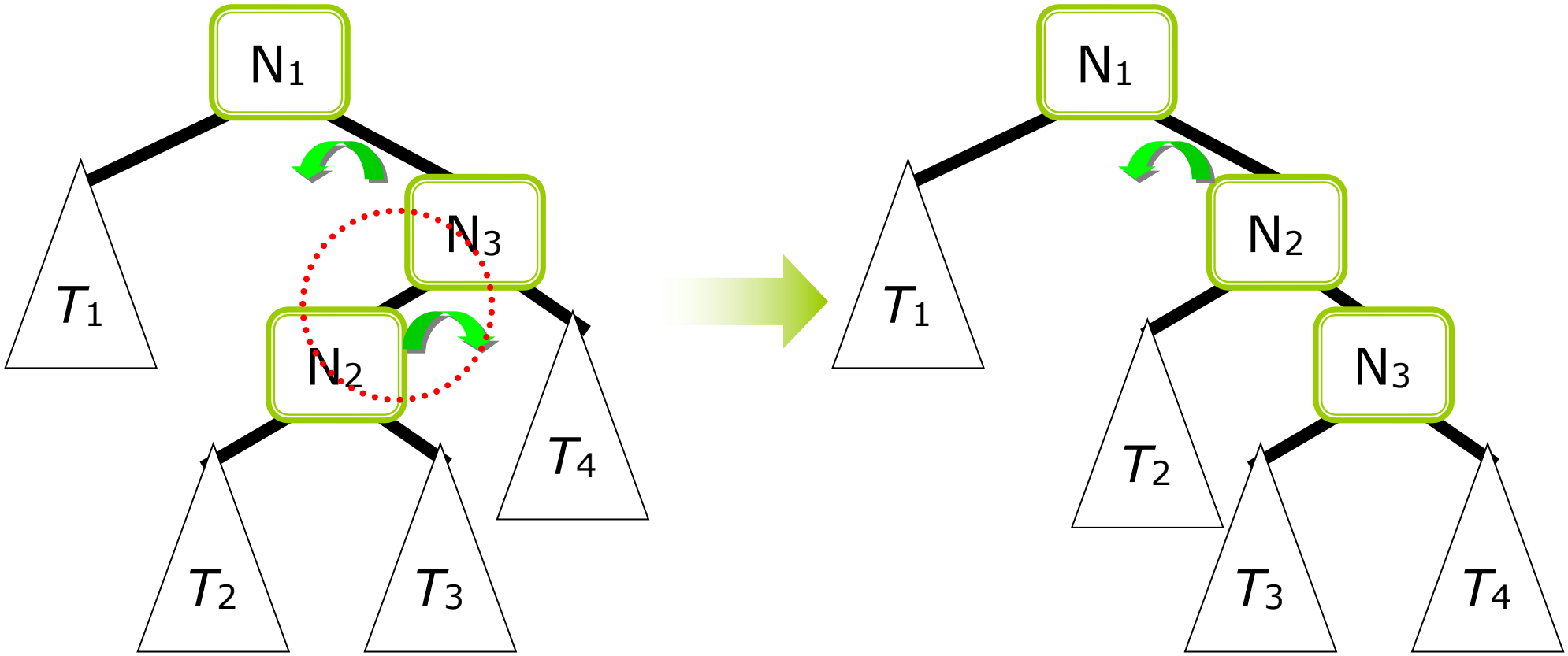
Just like this. No other meaning. No trick. Nothing really special.

Double Rotation

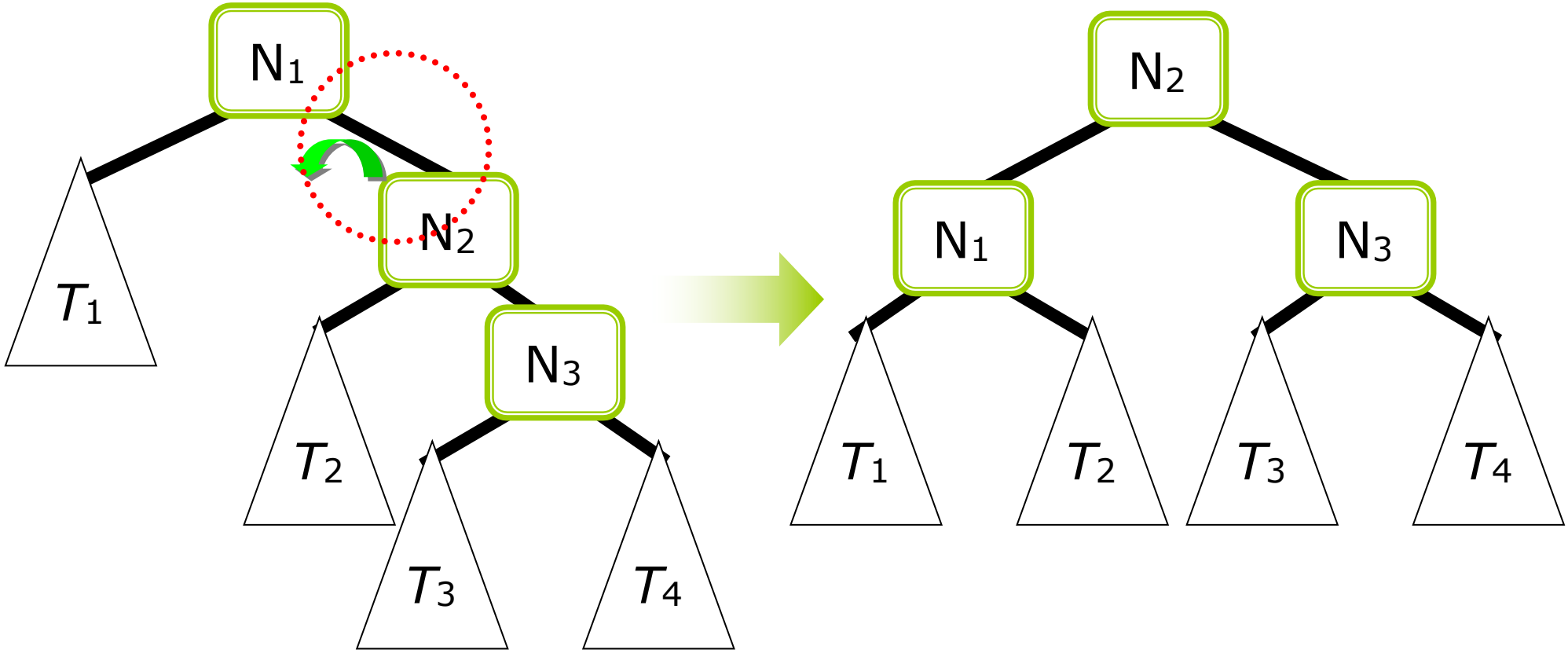
The case when a new node is inserted into the left subtree of the right child is similar.



Step 1 of a **Right**-Left Double Rotation:



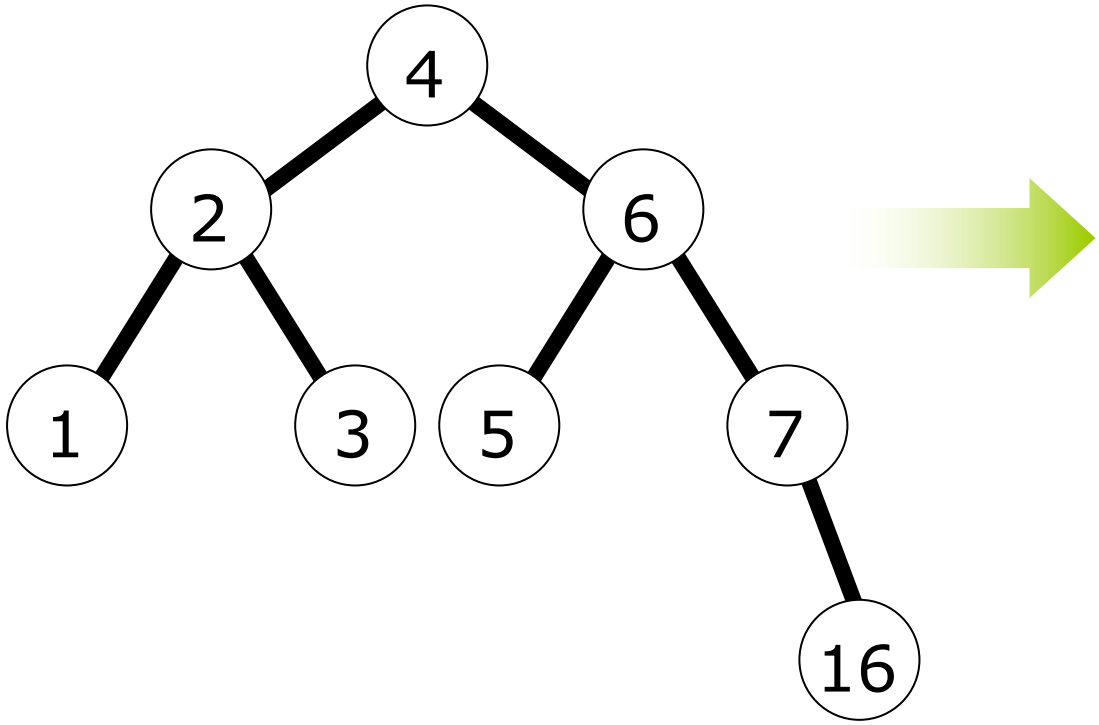
Step 2 of a Right-**Left** Double Rotation:



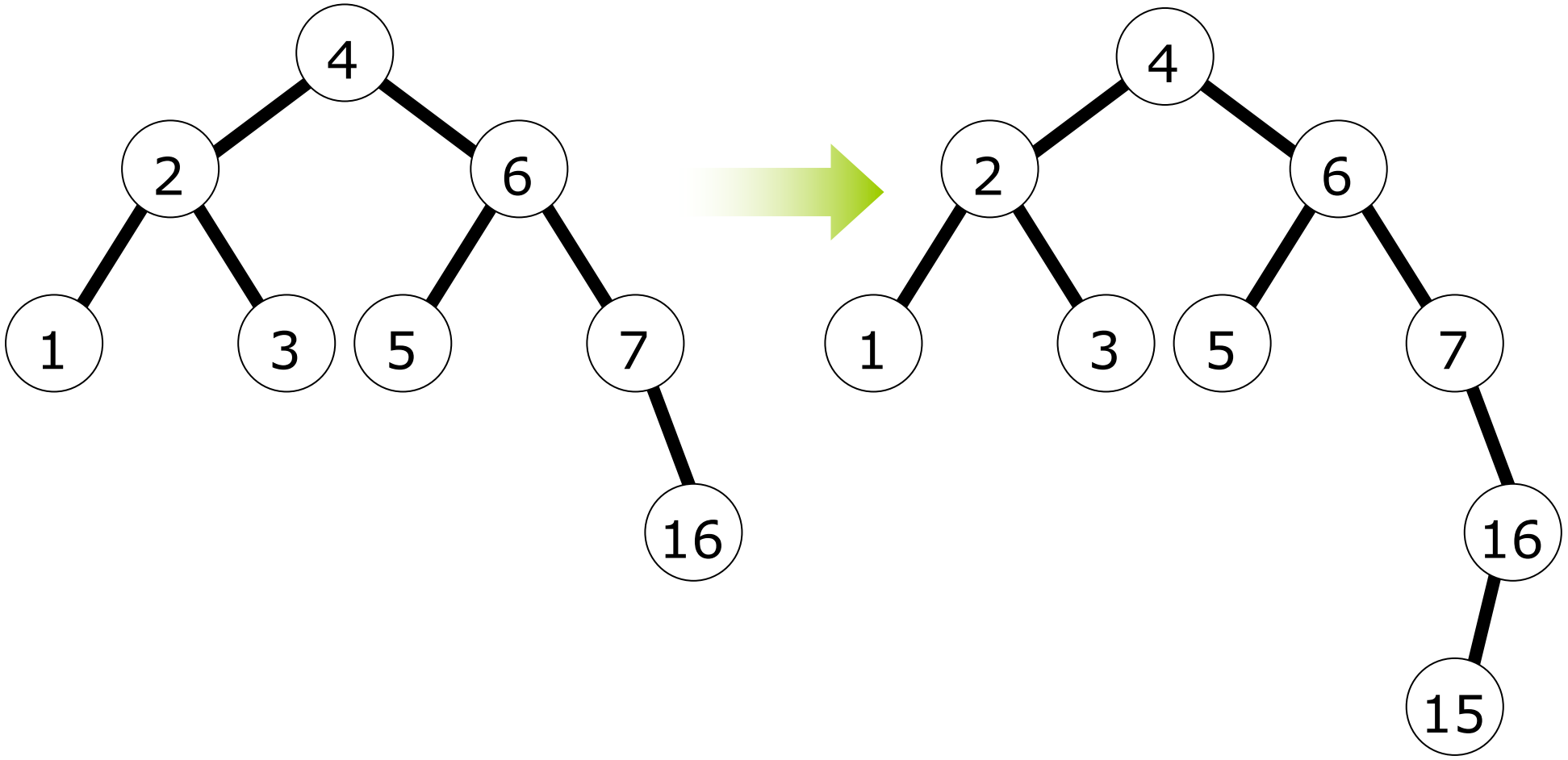
So, a Right-Left Double Rotation means a Right Single Rotation followed by a Left Single Rotation on one level higher.

Just like this. No other meaning. No trick. Nothing really special.

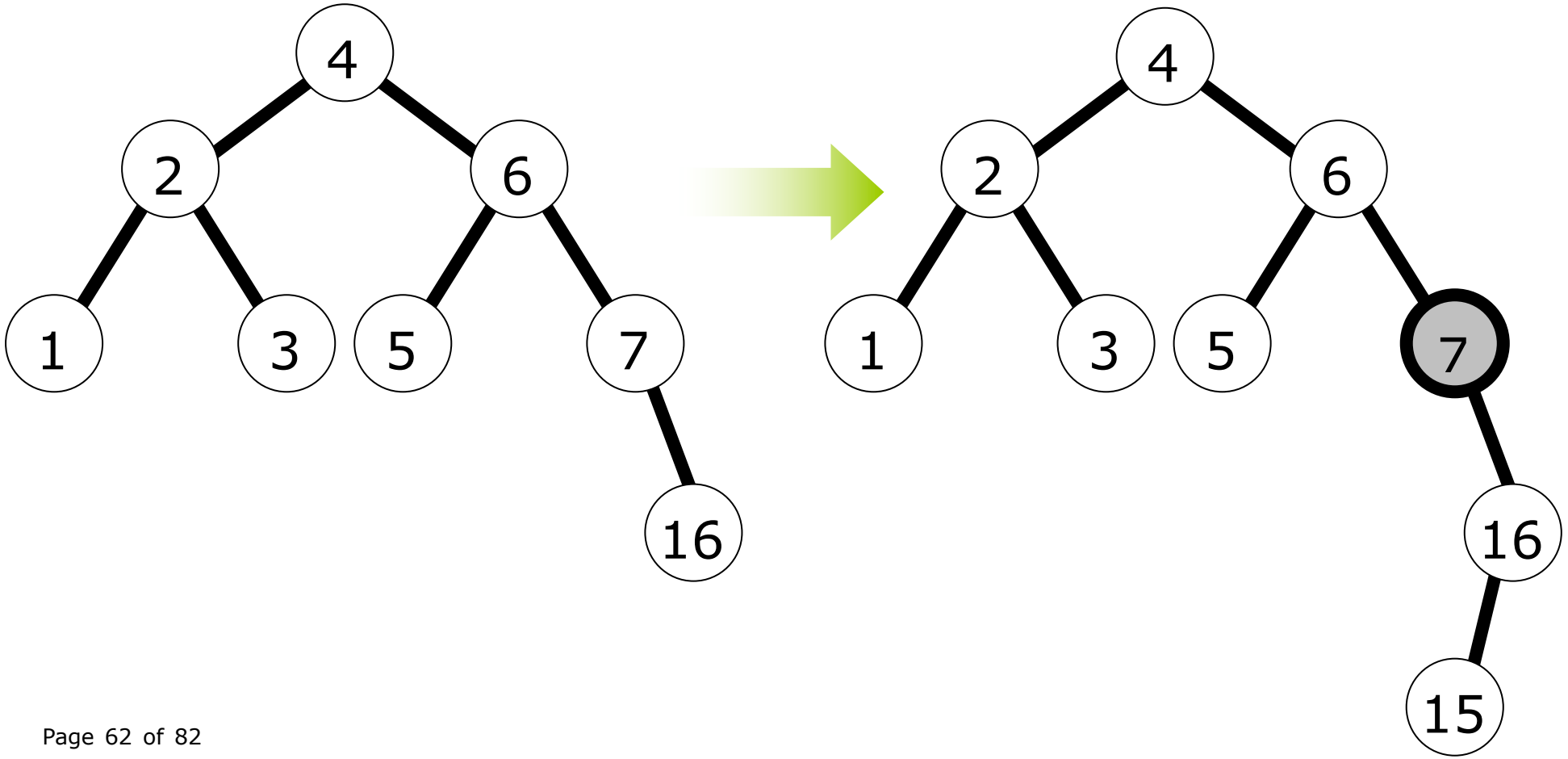
Example: a node 15 is inserted into the left subtree of the right child of **7**.



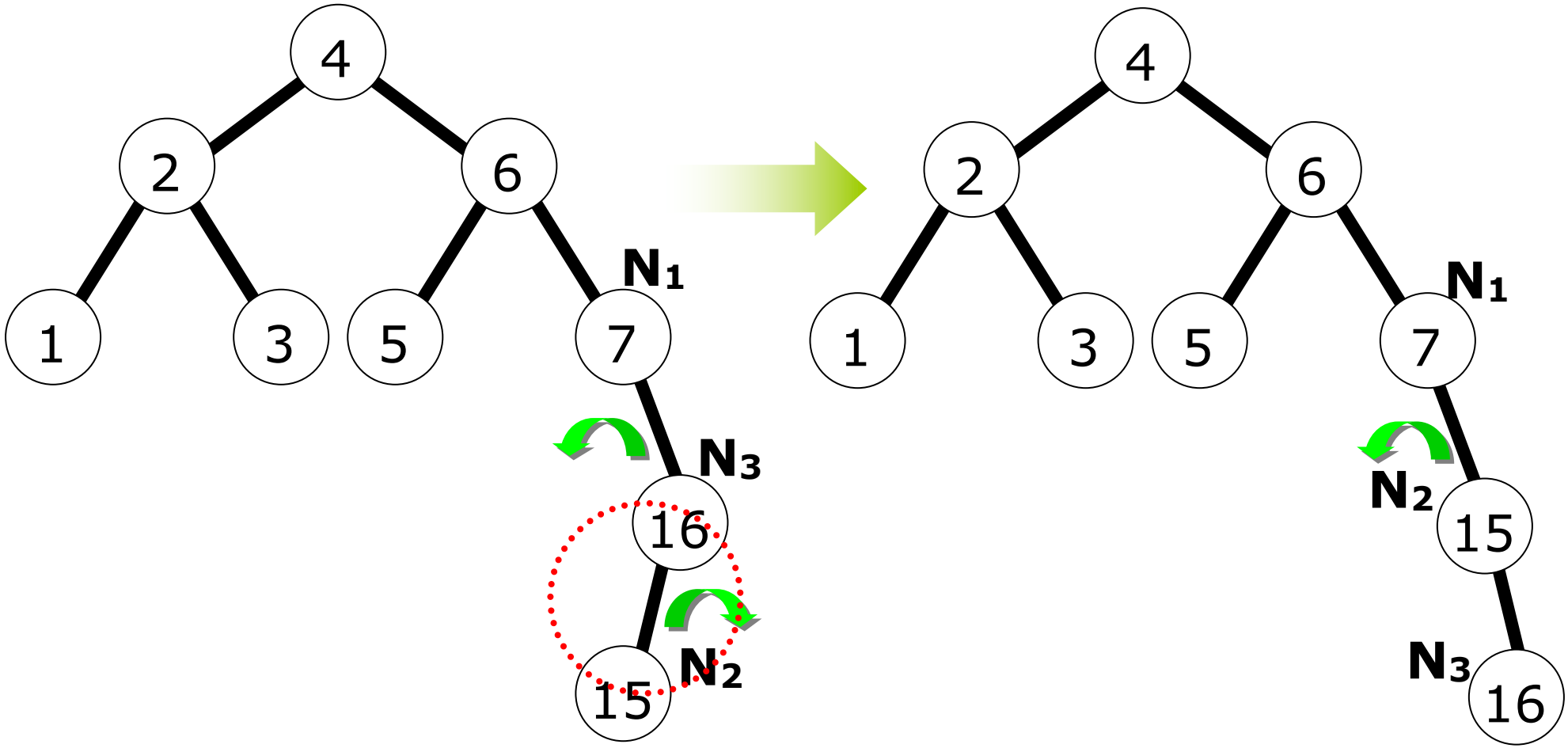
Example: a node 15 is inserted into the left subtree of the right child of **7**.



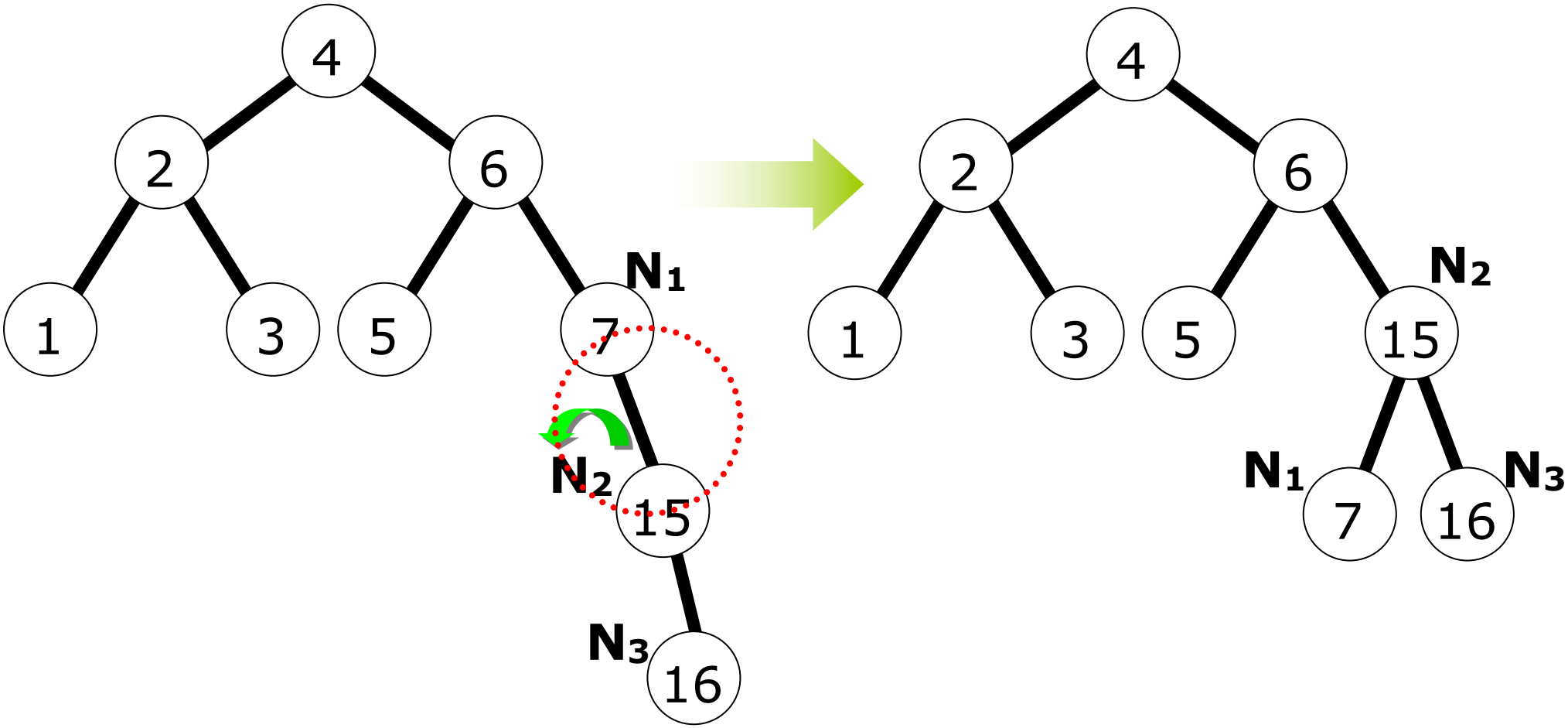
Example: a node 15 is inserted into the left subtree of the right child of **7**. So we need a **right-left double rotation**.



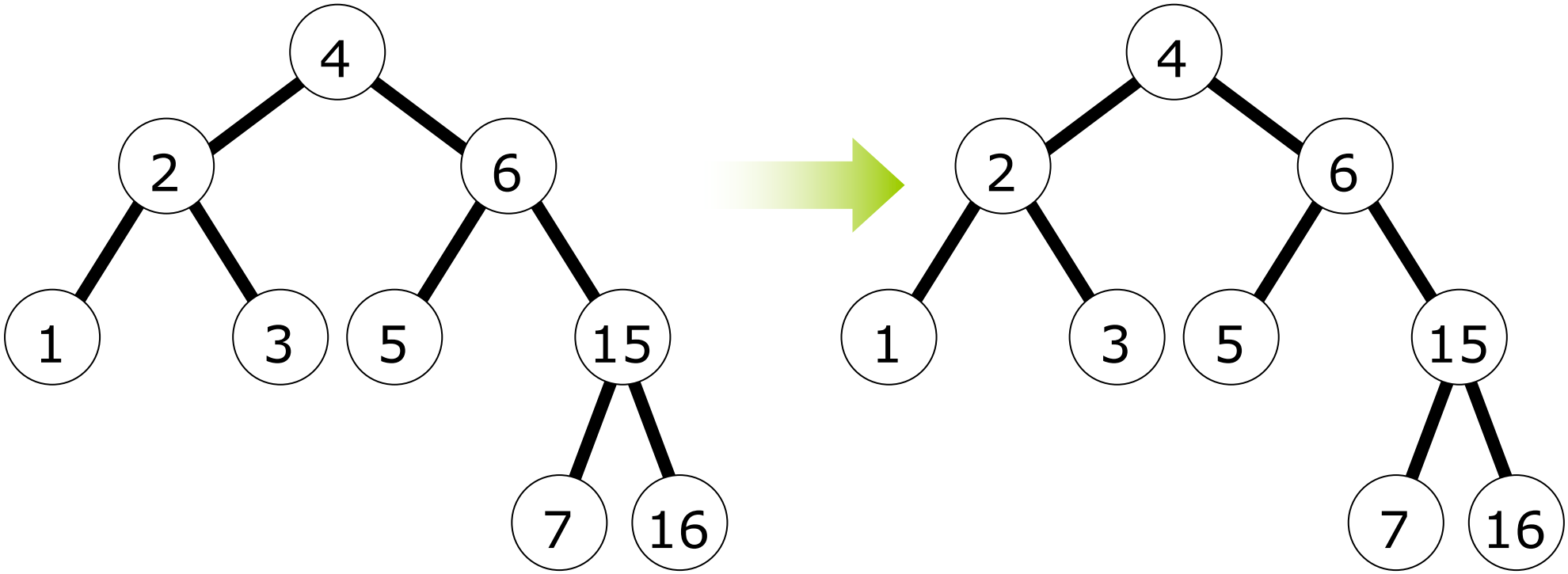
Example: Step 1 of the **right-left double rotation**.



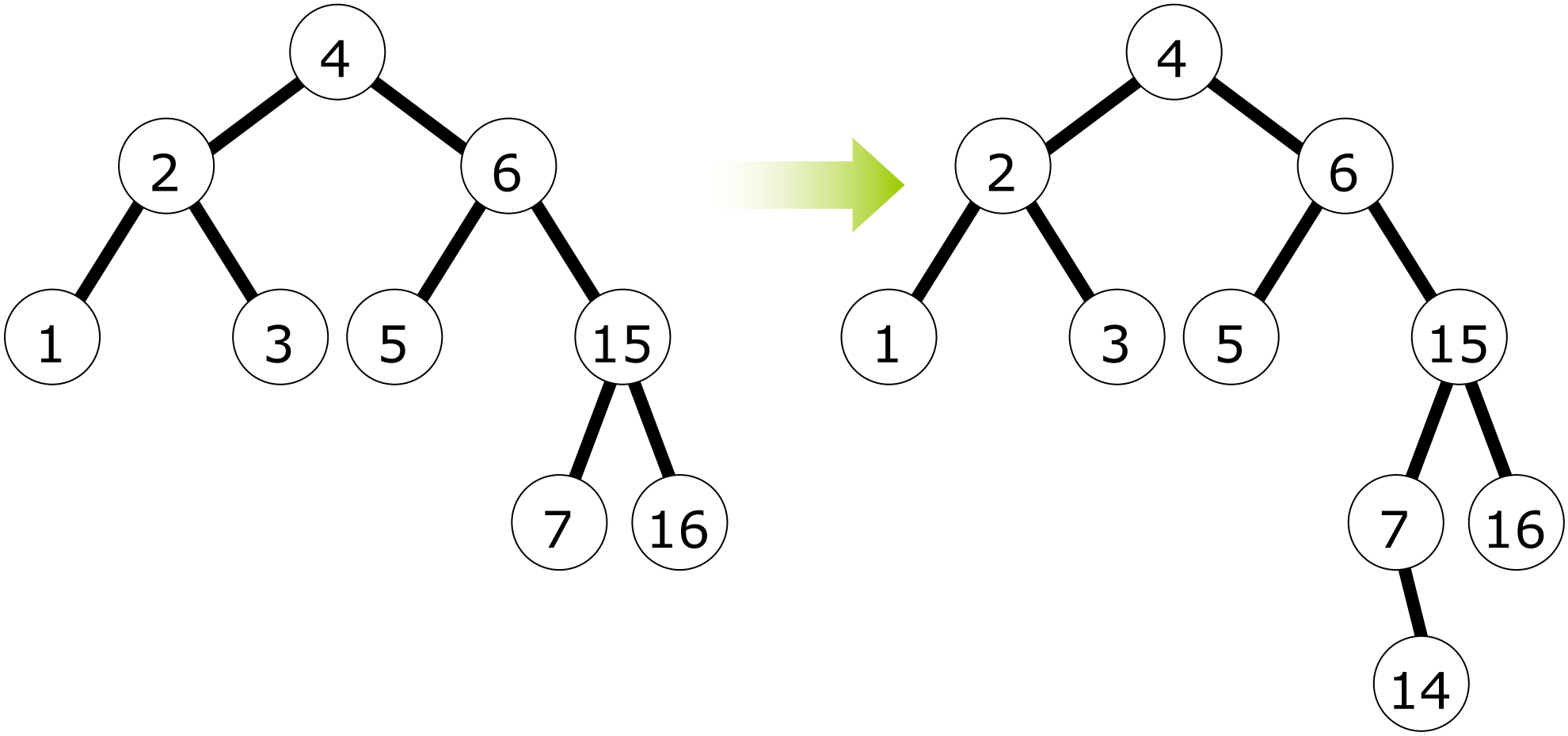
Example: Step 2 of the **right-left double rotation**.



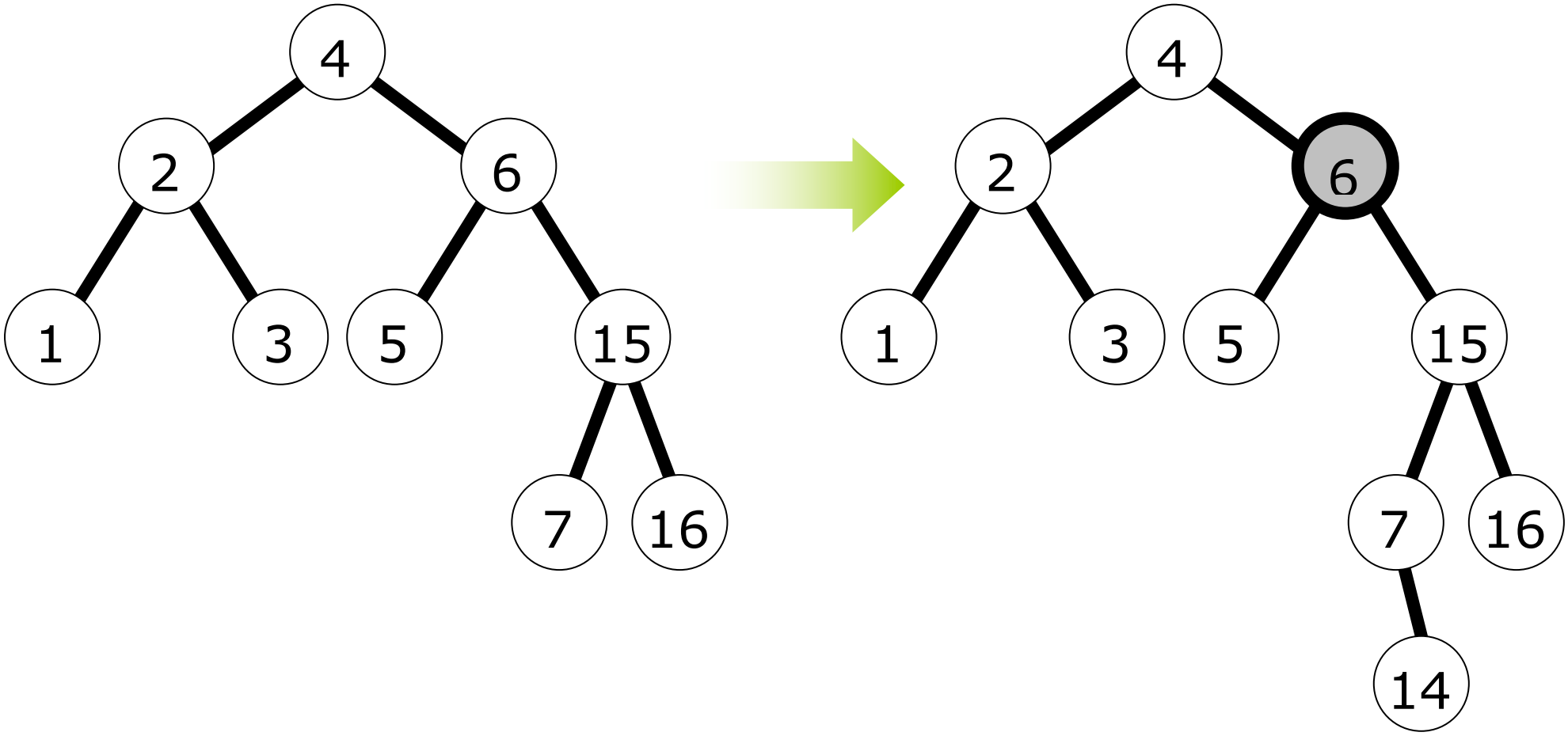
Next, we insert a node with key 14.



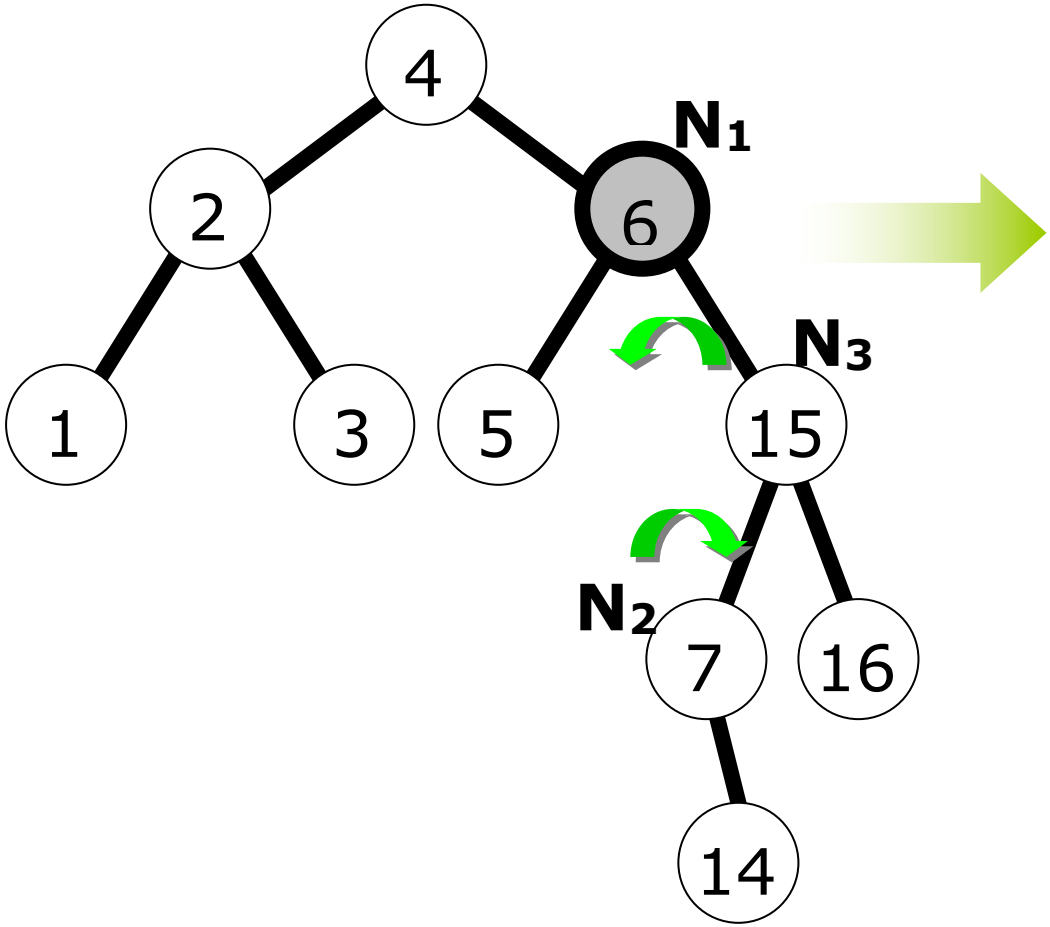
Next, we insert a node with key 14.



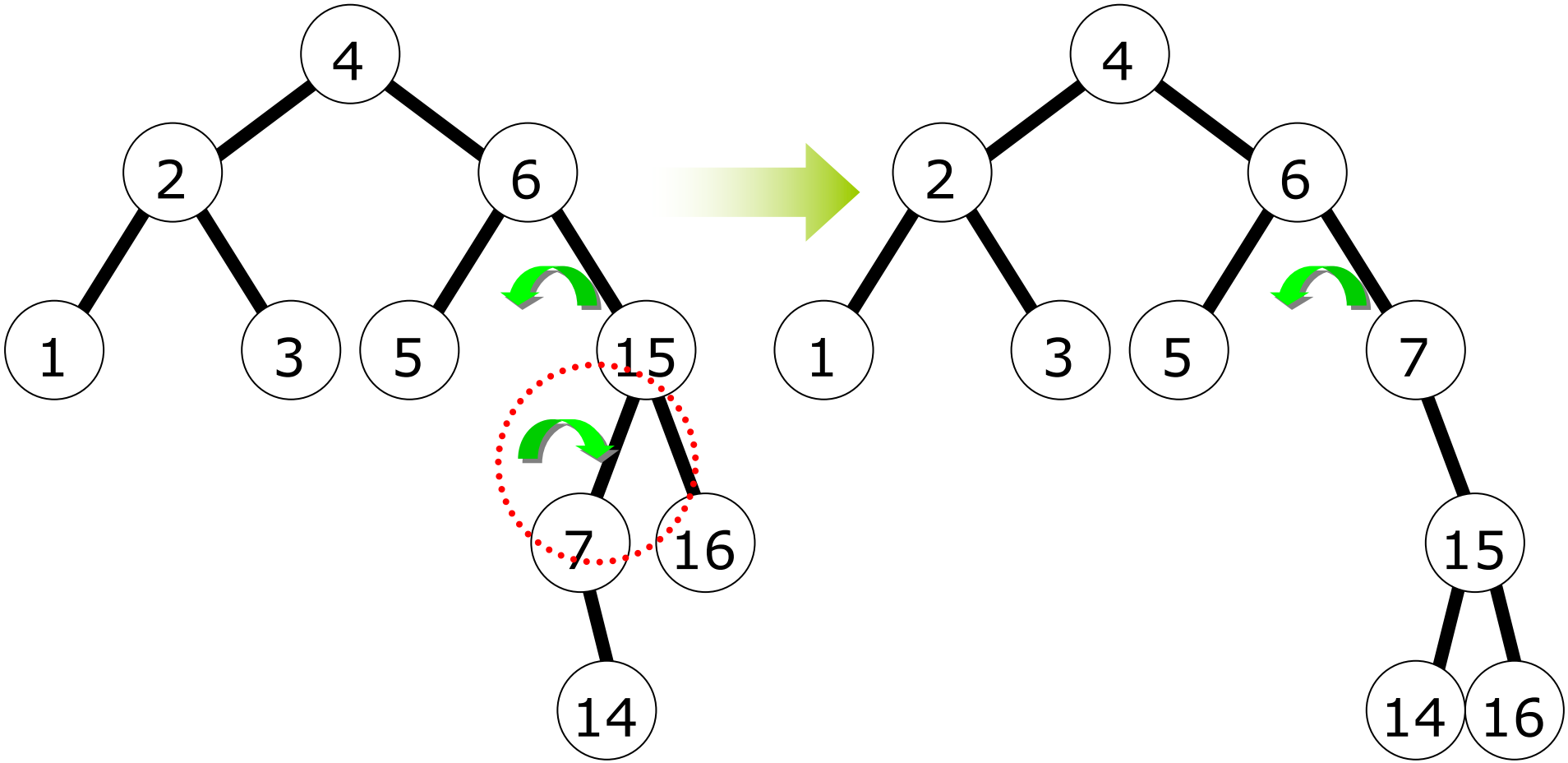
Next, we insert a node with key 14.



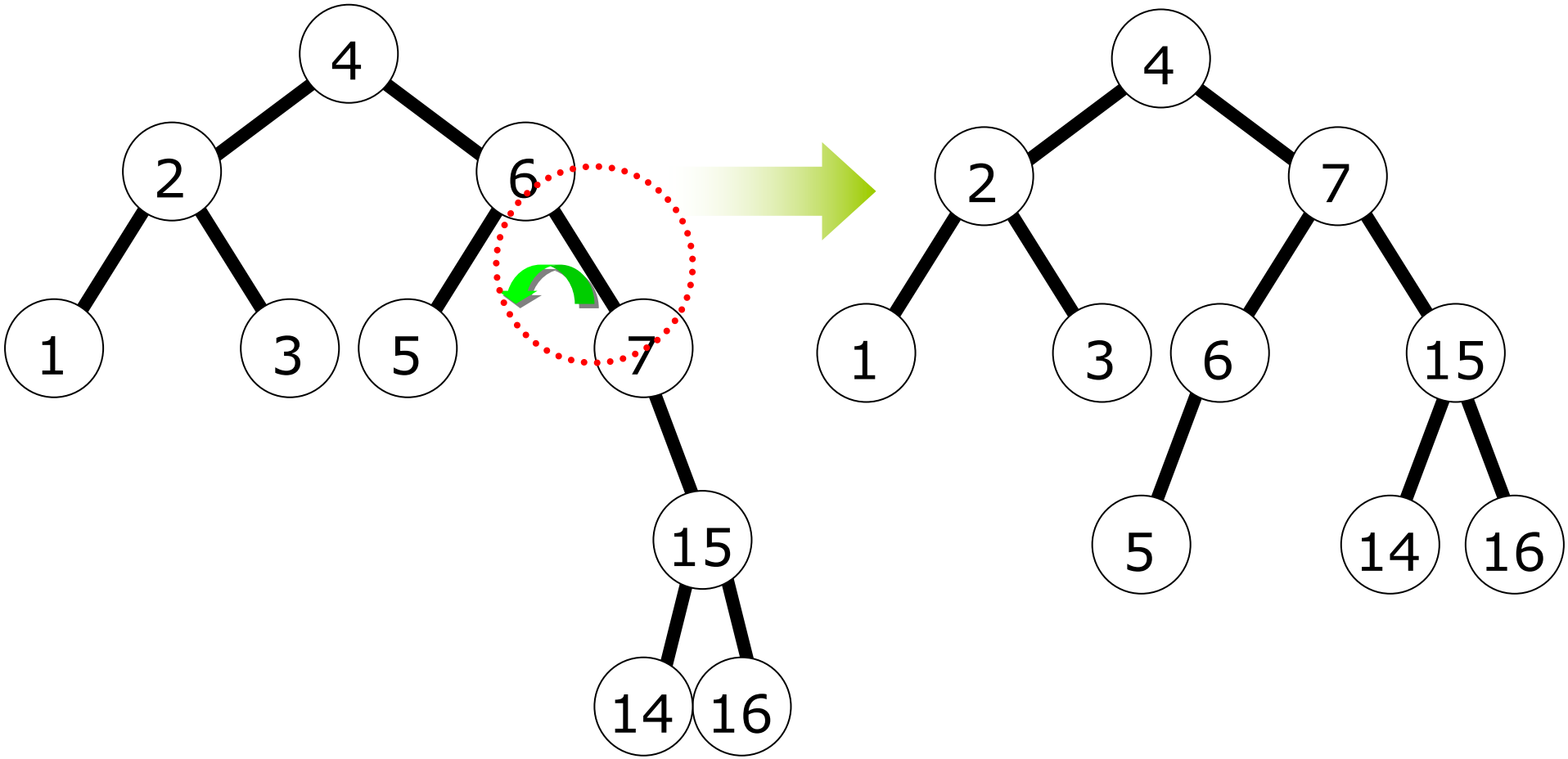
The node is inserted into the left subtree of a right child of **6**, so we need a **right-left double rotation**.



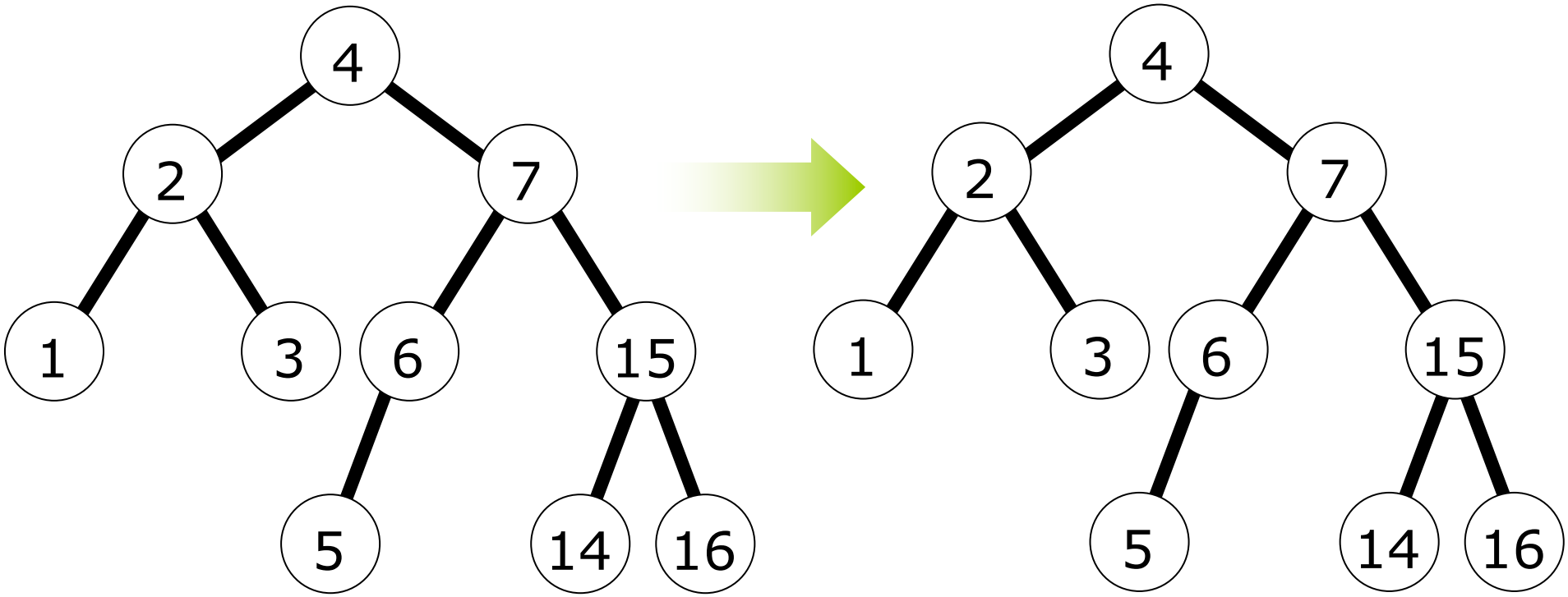
Step 1 of the **right-left double rotation**:



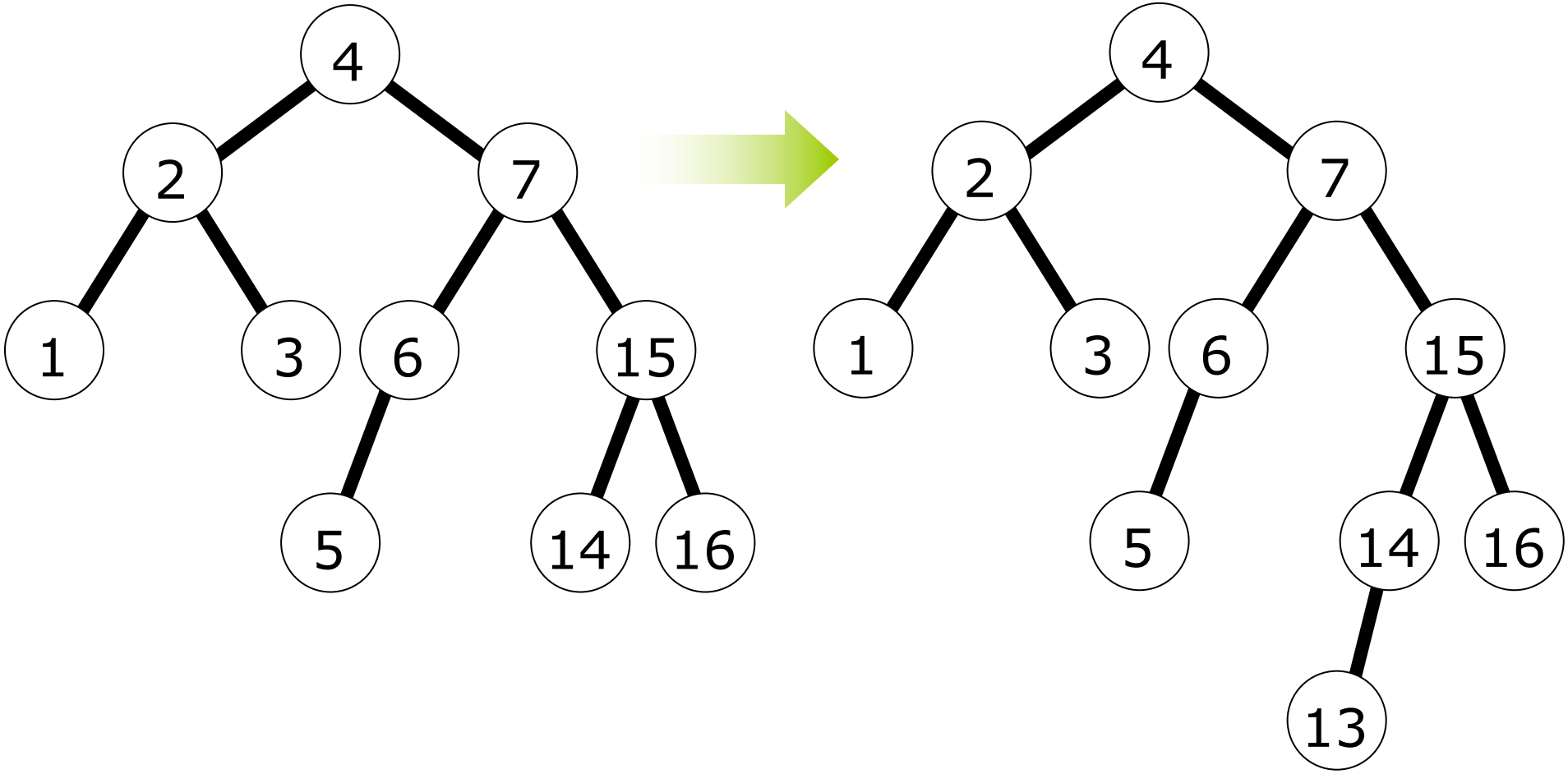
Step 2 of the **right-left double rotation**:



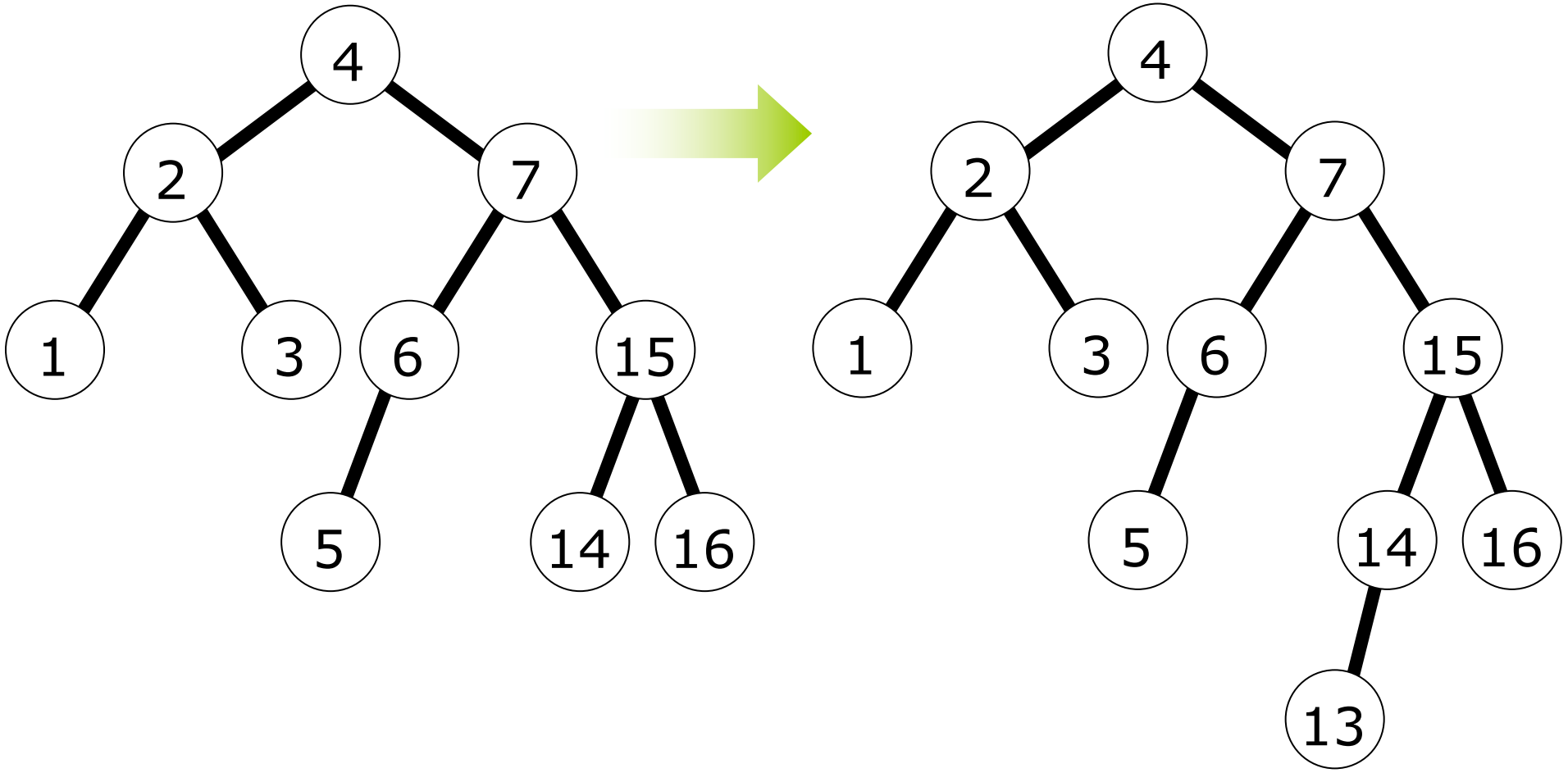
Next, we insert a node with key 13.



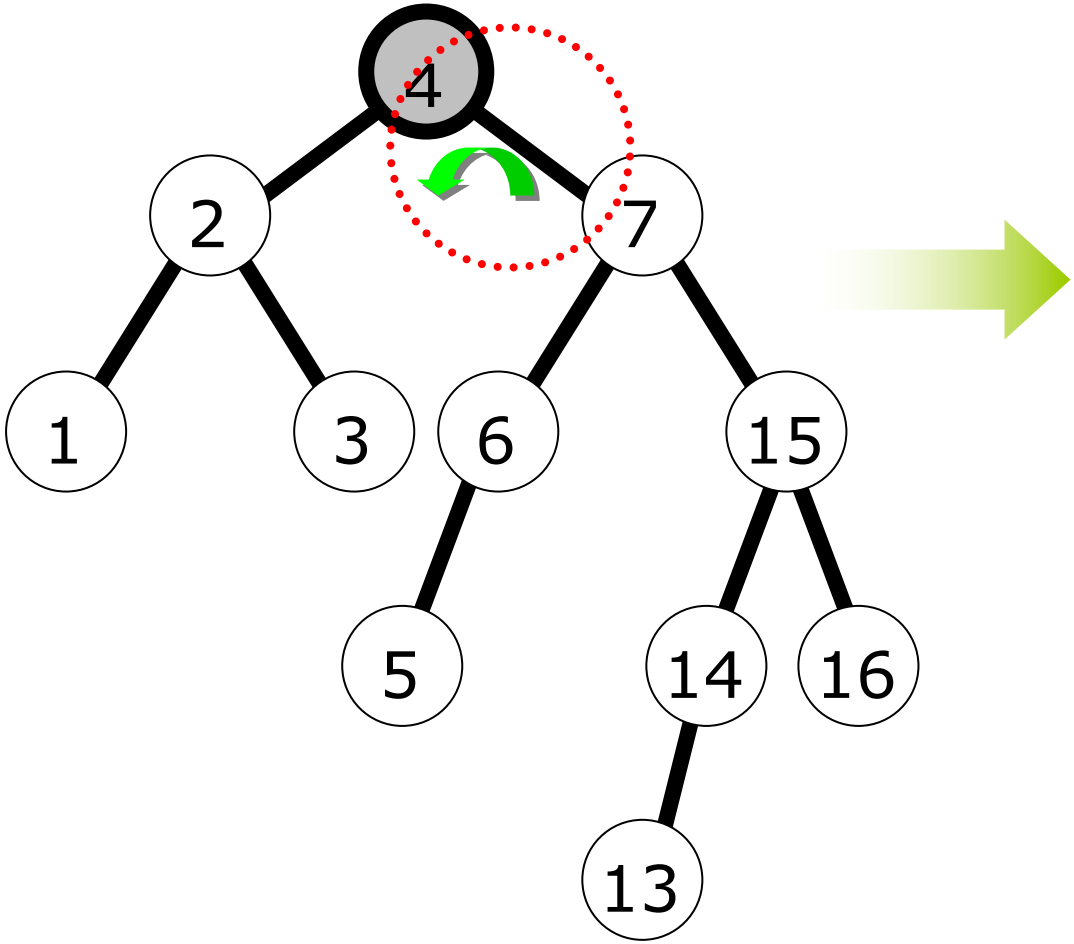
Next, we insert a node with key 13.



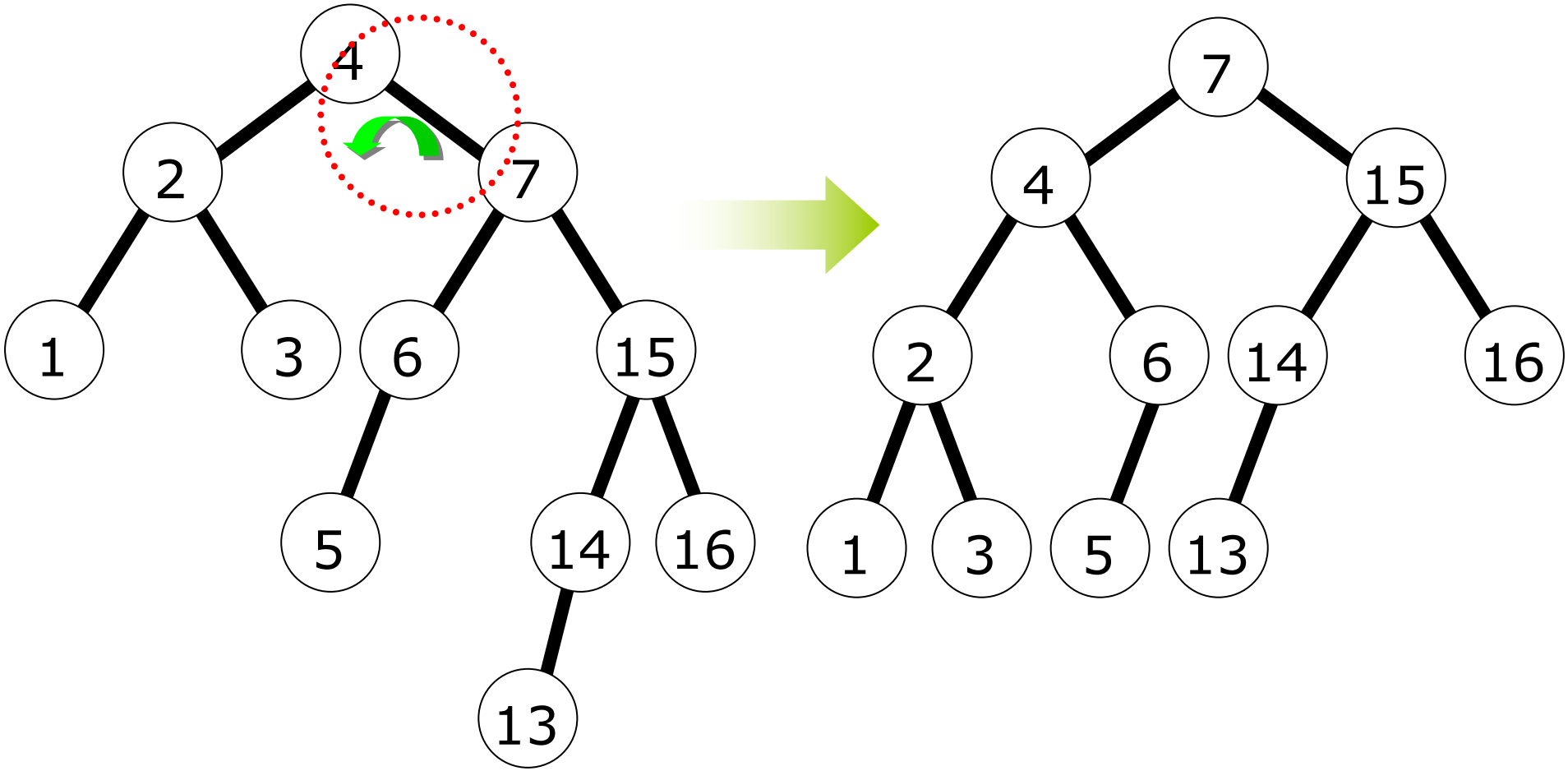
How to rotate?



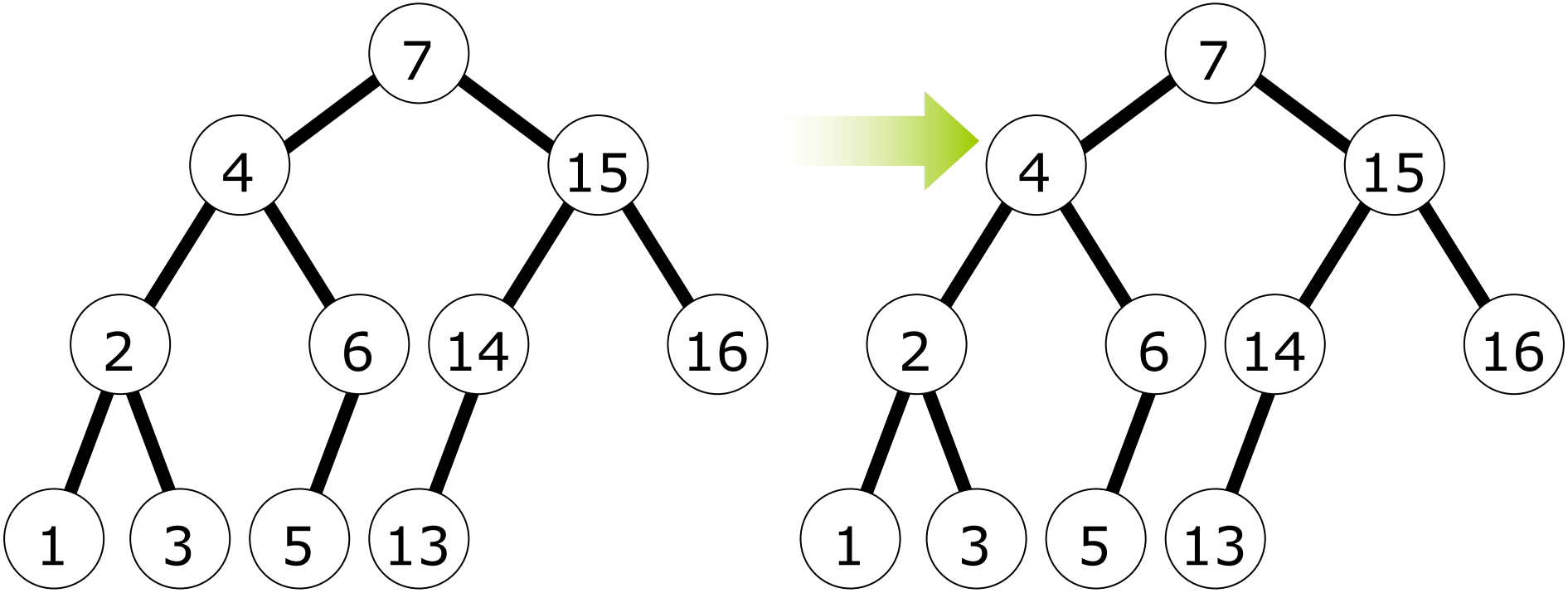
The node is inserted into the right subtree of a right child of **4**, so we need a **left single rotation**.



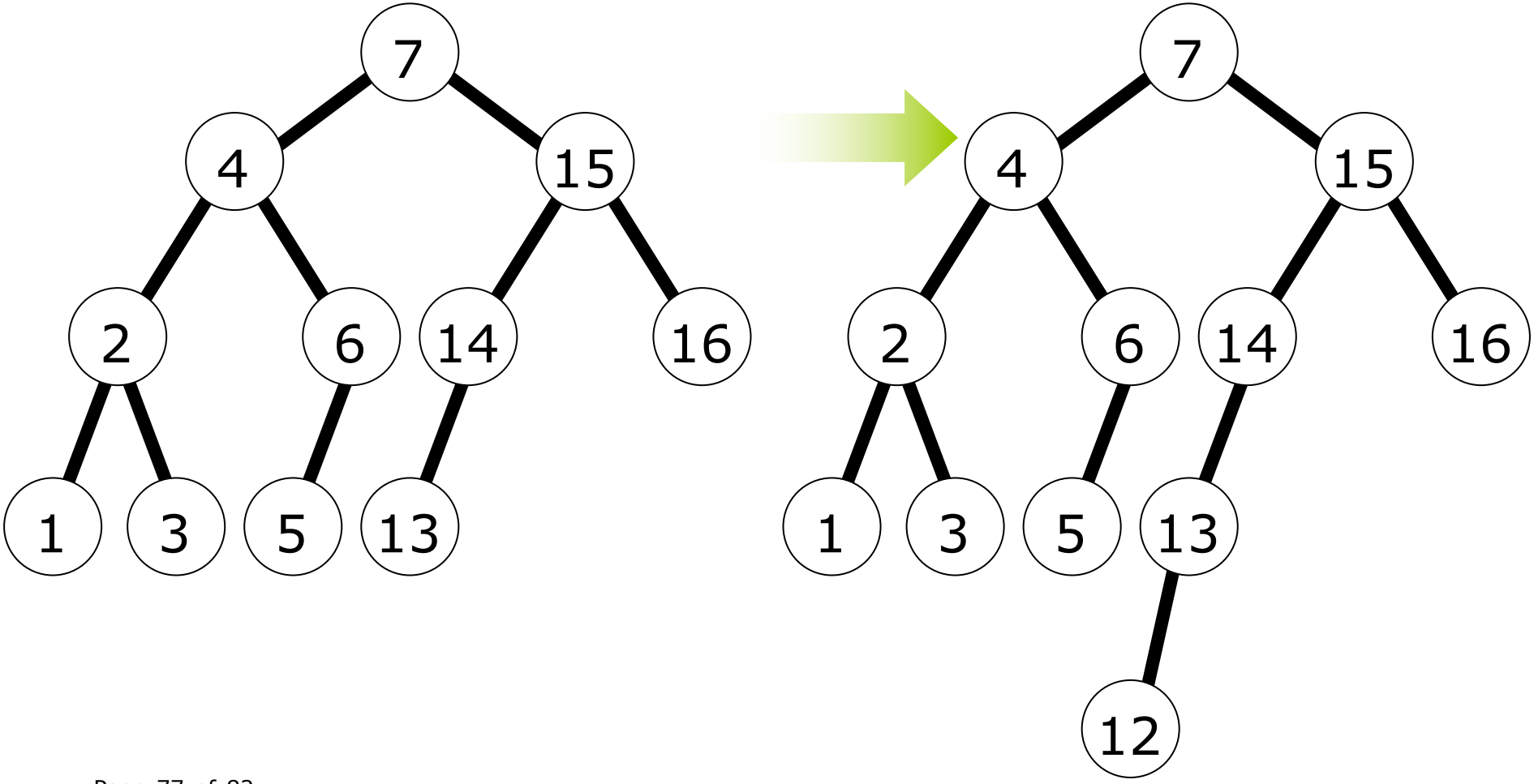
The node is inserted into the right subtree of a right child of **4**, so we need a **left single rotation**.



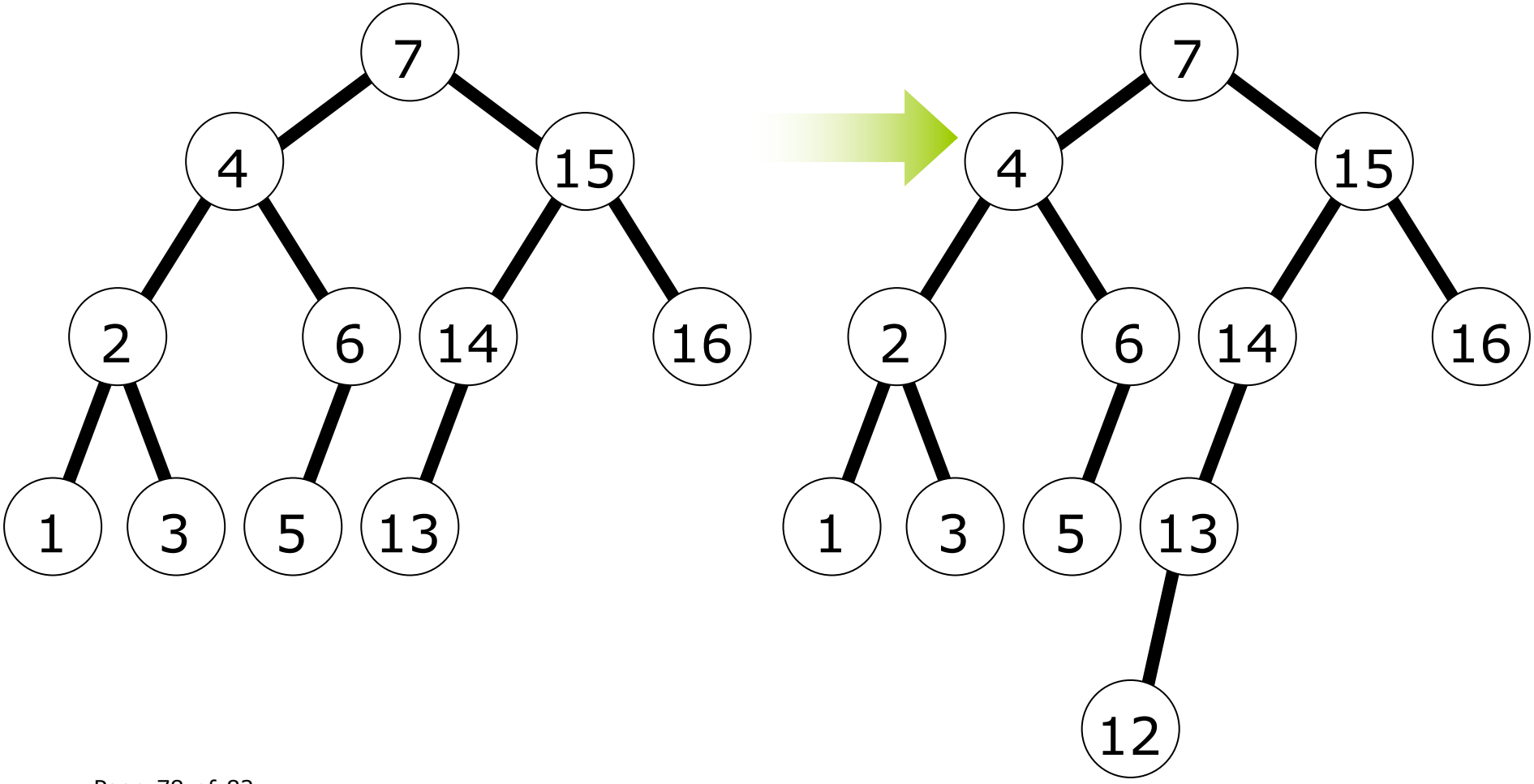
Next, we insert a node with key 12.



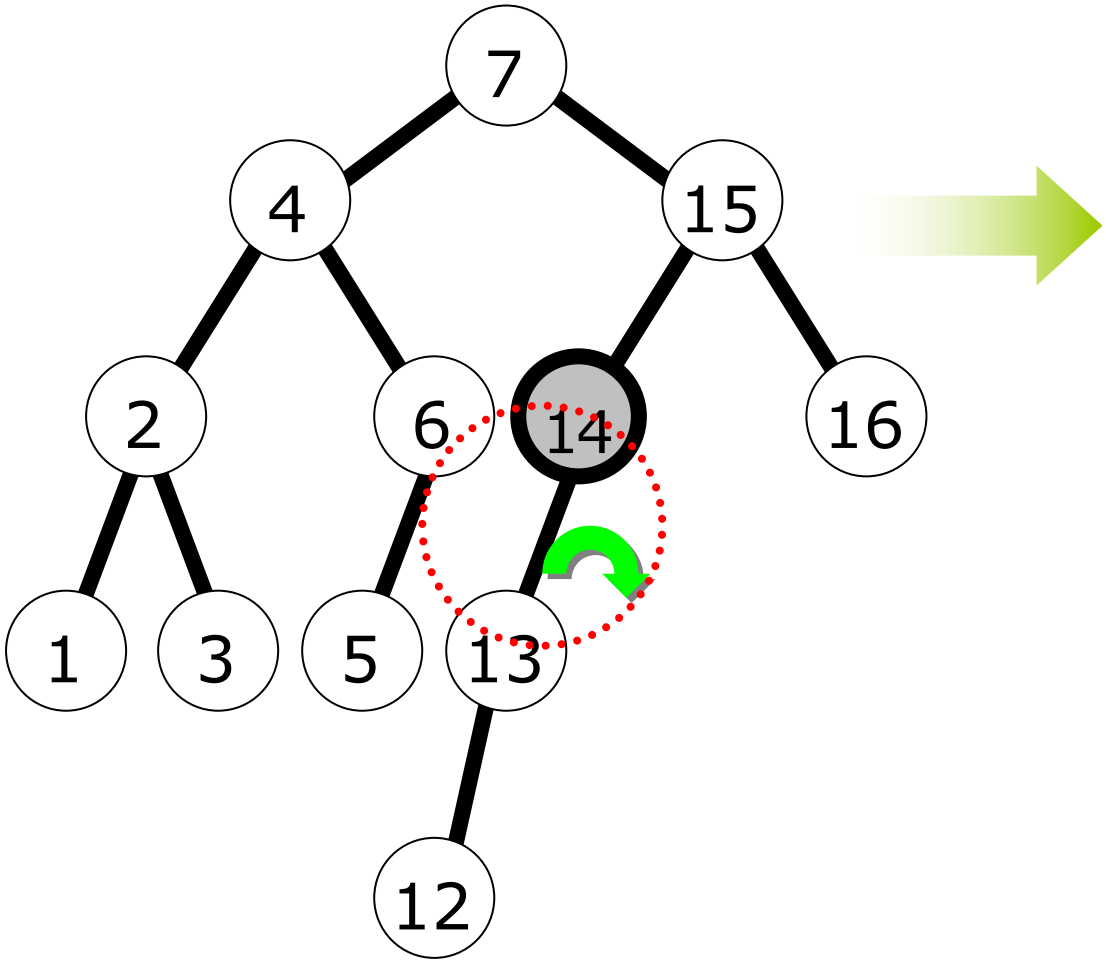
Next, we insert a node with key 12.



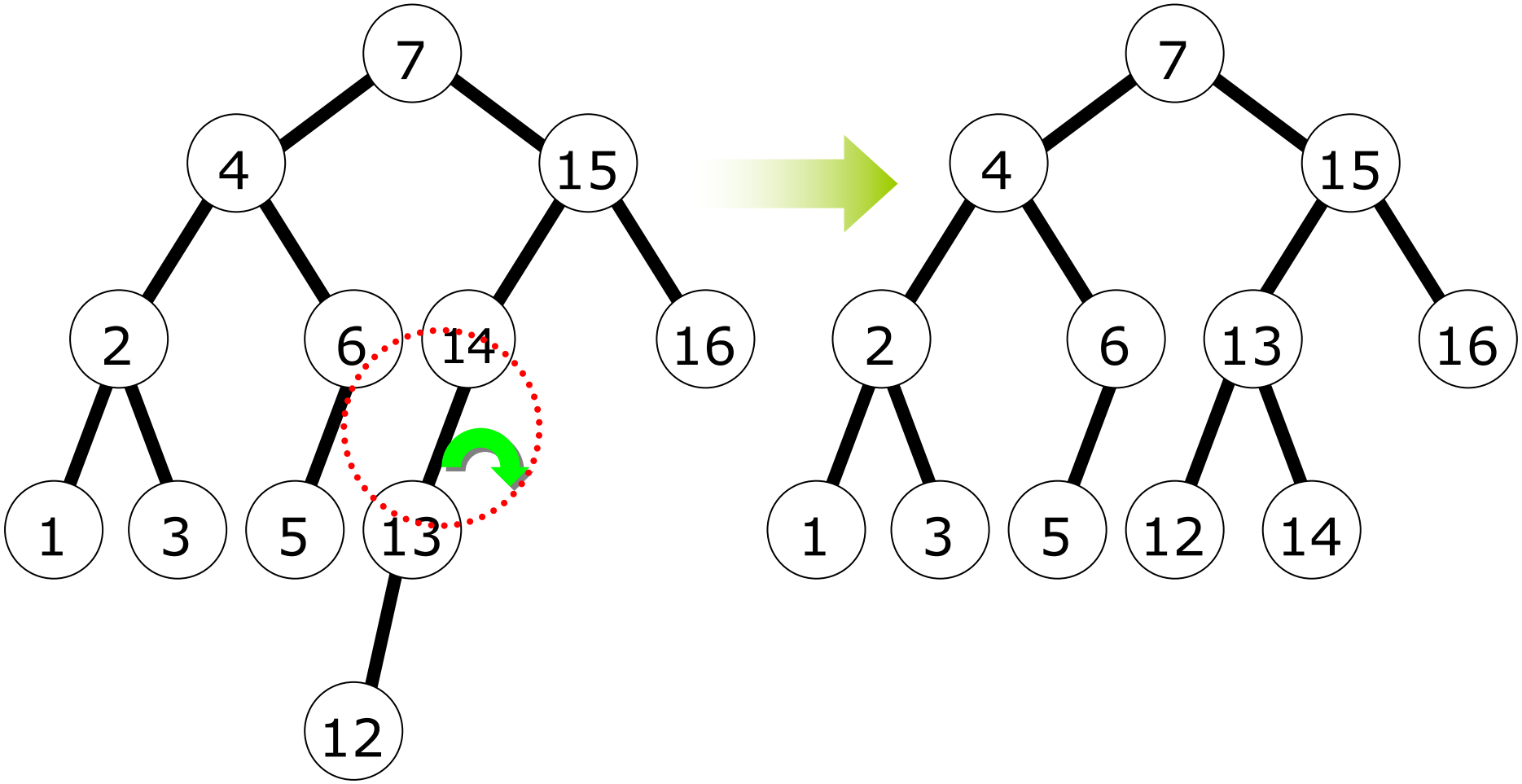
How to rotate?



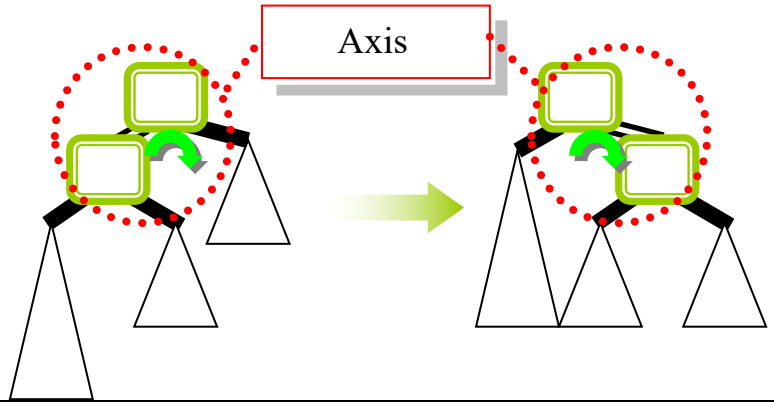
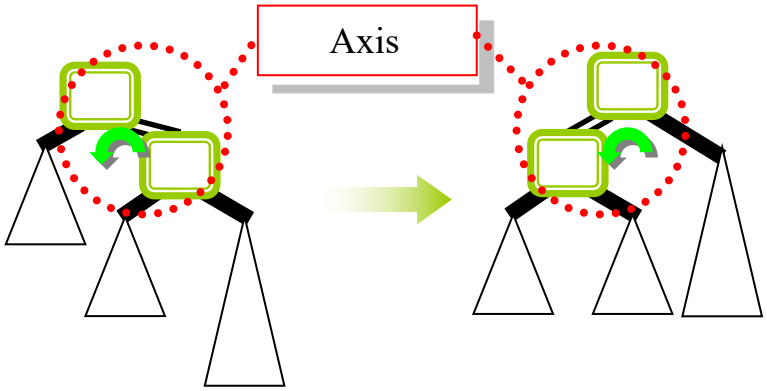
The node is inserted into the left subtree of a left child of **14**, so we need a **right single rotation**.



The node is inserted into the <u>left subtree of a left child of **14**</u>, so we need a **right single rotation**.



Rebalancing AVL Trees

Case 1: **Right Single Rotation**	**Case 2** **Left Single Rotation**
A new node is inserted into the left subtree of the left child.	A new node is inserted into the right subtree of the right child.
	

Rebalancing AVL Trees

Case 3: **Left-Right Double Rotation**	**Case 4:** **Right-Left Double Rotation**
A new node is inserted into the right subtree of the left child.	A new node is inserted into the left subtree of the right child.