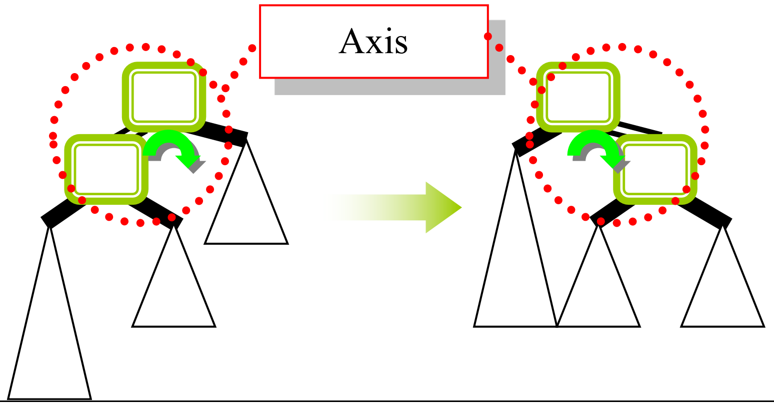
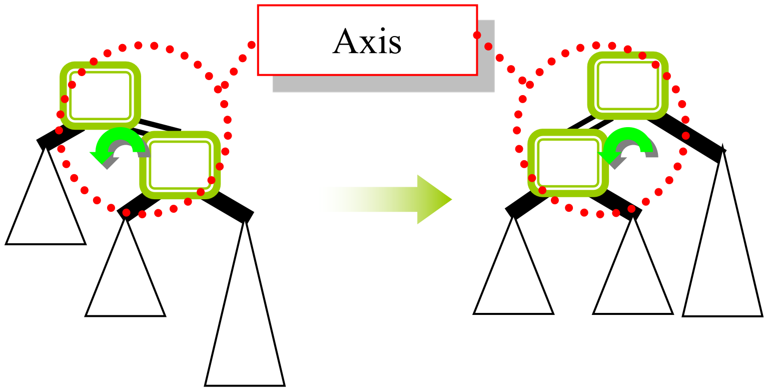
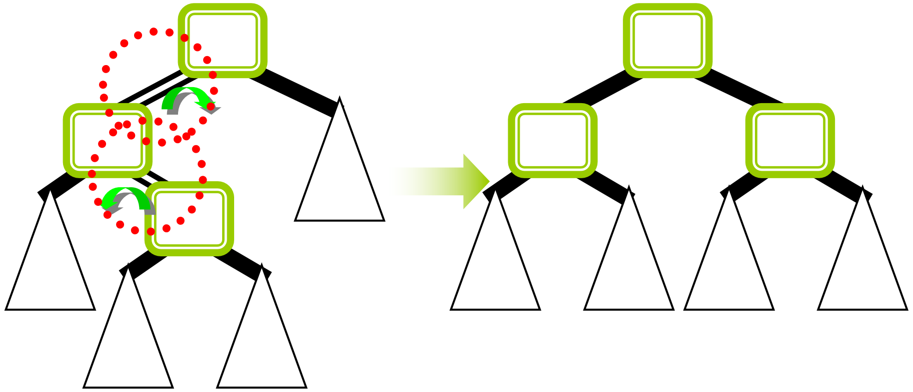
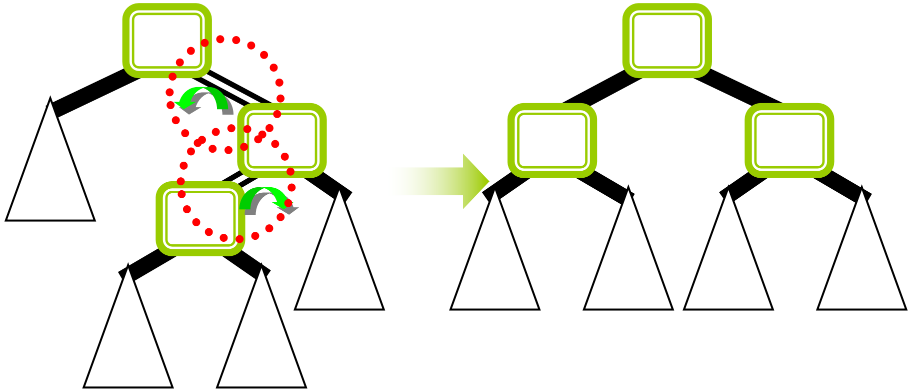
# Rebalancing AVL Trees

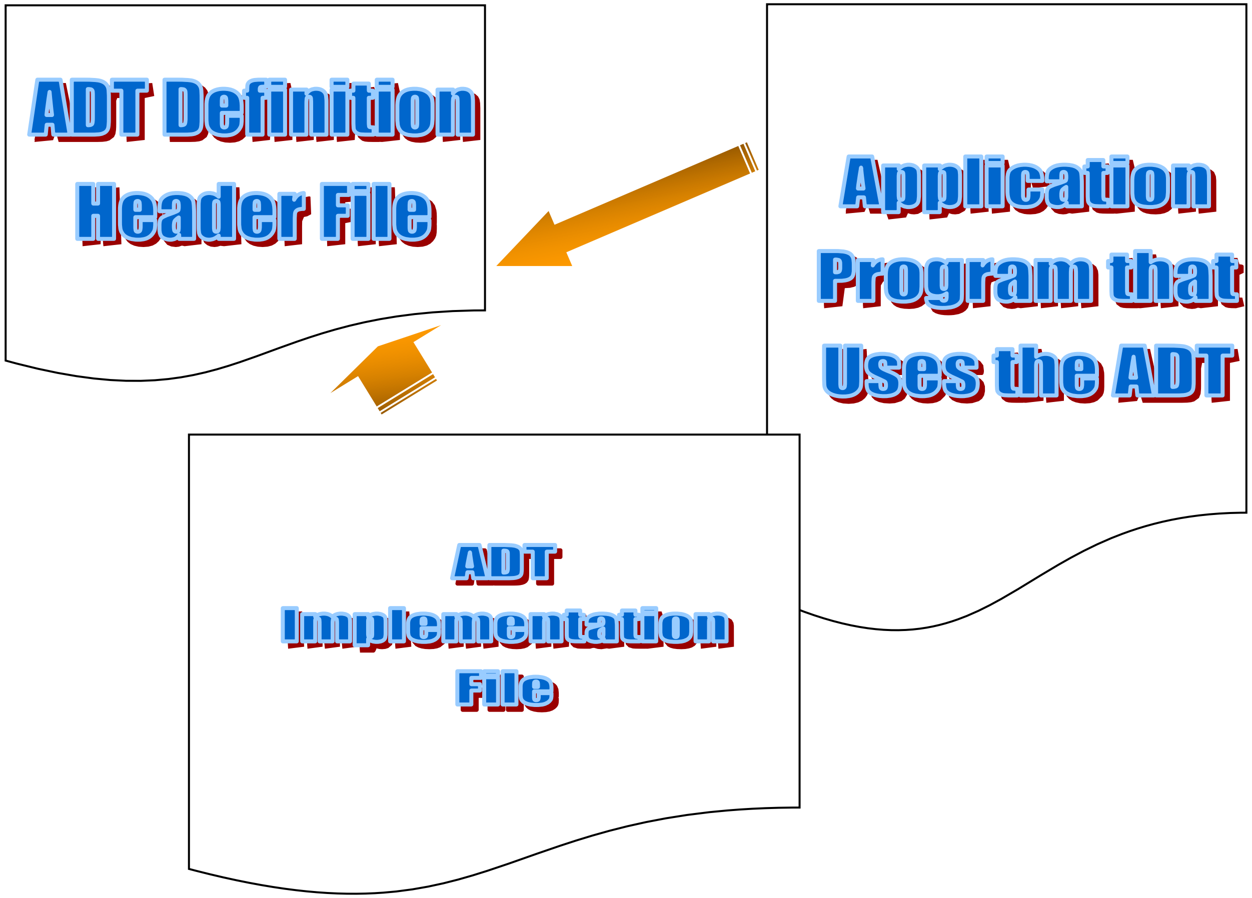
| | |
|---|---|
| **Case 1:**<br>**Right Single Rotation**<br>A new node is inserted into the left subtree of the left child. | **Case 2**<br>**Left Single Rotation**<br>A new node is inserted into the right subtree of the right child. |
|  |  |

# Rebalancing AVL Trees

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

# The AVL Tree ADT

As we always did, we consider AVL trees as an ADT. We shall first show the ADT definition header file.

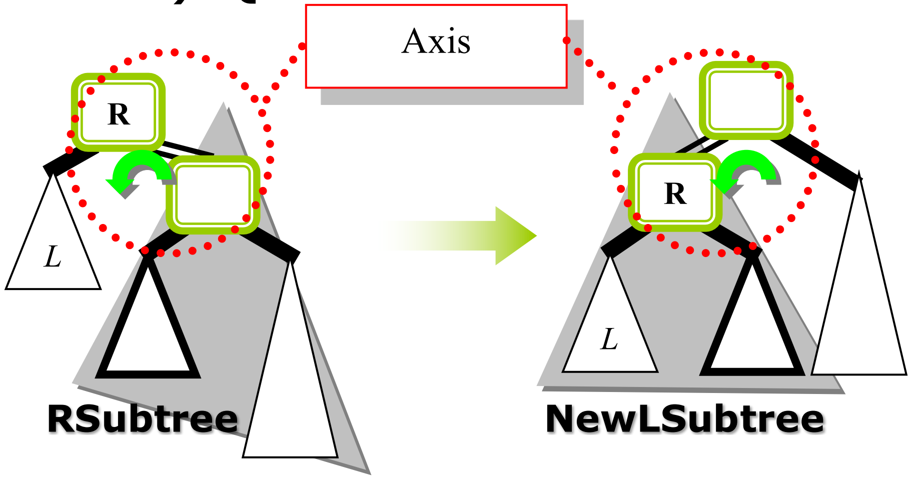

ADT Definition Header File
Application Program that Uses the ADT
ADT Implementation File

```
/* File: AVLTreeADT.h */
#include <stdio.h>
typedef struct AVLTreeCDT *AVLTreeADT;
typedef struct TreeNodeCDT *TreeNodeADT;
#define SpecialErrNode (TreeNodeADT) NULL
AVLTreeADT NonemptyAVLTree(TreeNodeADT, AVLTreeADT, AVLTreeADT);
AVLTreeADT EmptyAVLTree(void);
AVLTreeADT LeftAVLSubtree(AVLTreeADT);
AVLTreeADT RightAVLSubtree(AVLTreeADT);
int AVLTreeIsEmpty(AVLTreeADT);
int AVLTreeHeight(AVLTreeADT);
TreeNodeADT AVLRoot(AVLTreeADT);
TreeNodeADT NewTreeNode(char*, int);
char* GetNodeKey(TreeNodeADT);
int GetNodeData(TreeNodeADT);
```

**Now, let’s write functions that perform rotations.**

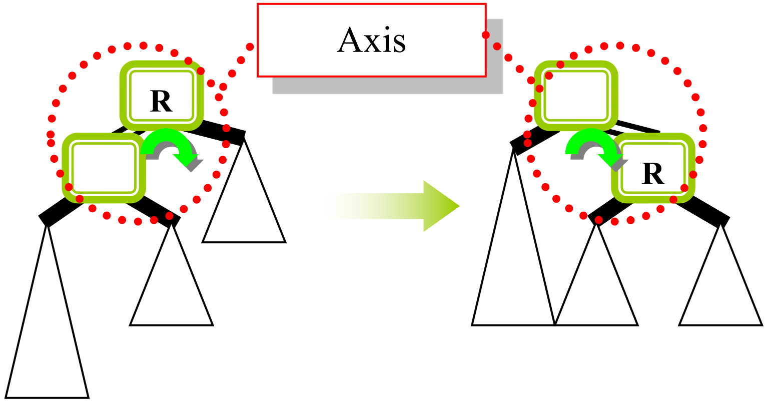
**First, we write functions that perform SINGLE rotations.**

```
AVLTreeADT LeftRotate(AVLTreeADT t) {

  AVLTreeADT RSubtree;
  AVLTreeADT NewLSubtree;


  RSubtree = RightAVLSubtree(t);
  NewLSubtree = NonemptyAVLTree(AVLRoot(t),
              LeftAVLSubtree(t), LeftAVLSubtree(RSubtree));

  return NonemptyAVLTree(AVLRoot(RSubtree),
              NewLSubtree, RightAVLSubtree(RSubtree));
}
```

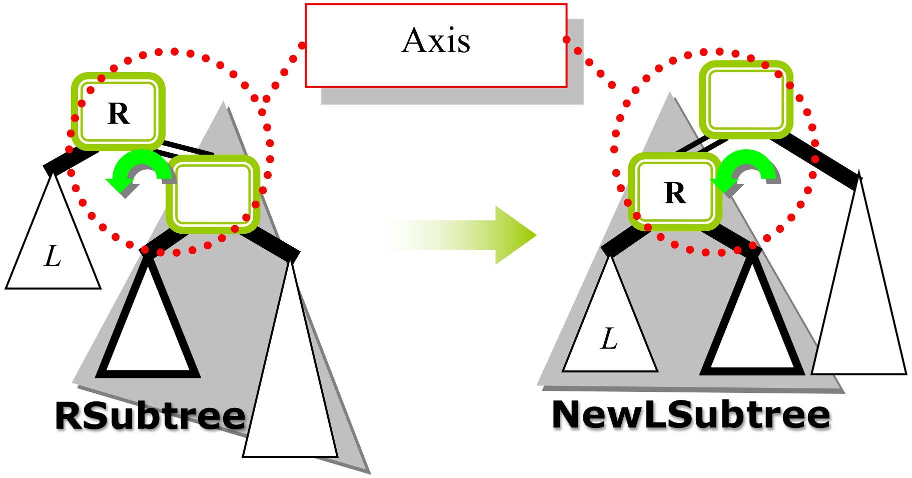


## This is pretty simple (and very logical)!

Note: I hope all of you can now write the RightRotate function, which I do not write.

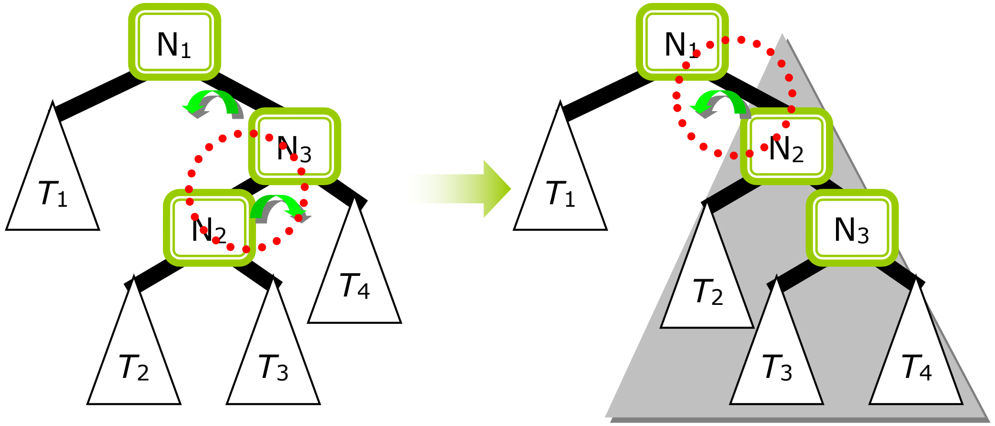


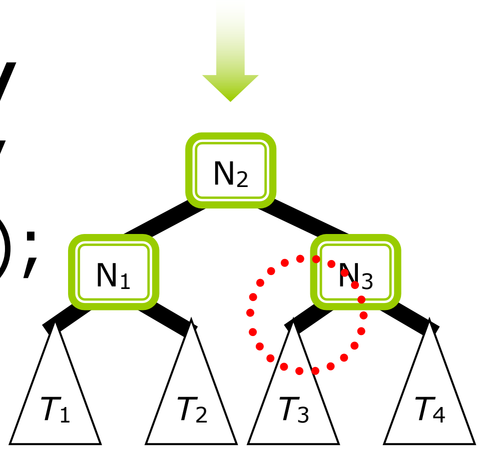
```
AVLTreeADT LeftRotate(AVLTreeADT t) {
   return NonemptyAVLTree(
                        AVLRoot(RightAVLSubtree(t)),
                        NonemptyAVLTree(
                              AVLRoot(t),
                              LeftAVLSubtree(t),
                              LeftAVLSubtree(RightAVLSubtree(t))),
                        RightAVLSubtree(RightAVLSubtree(t)));
}
```



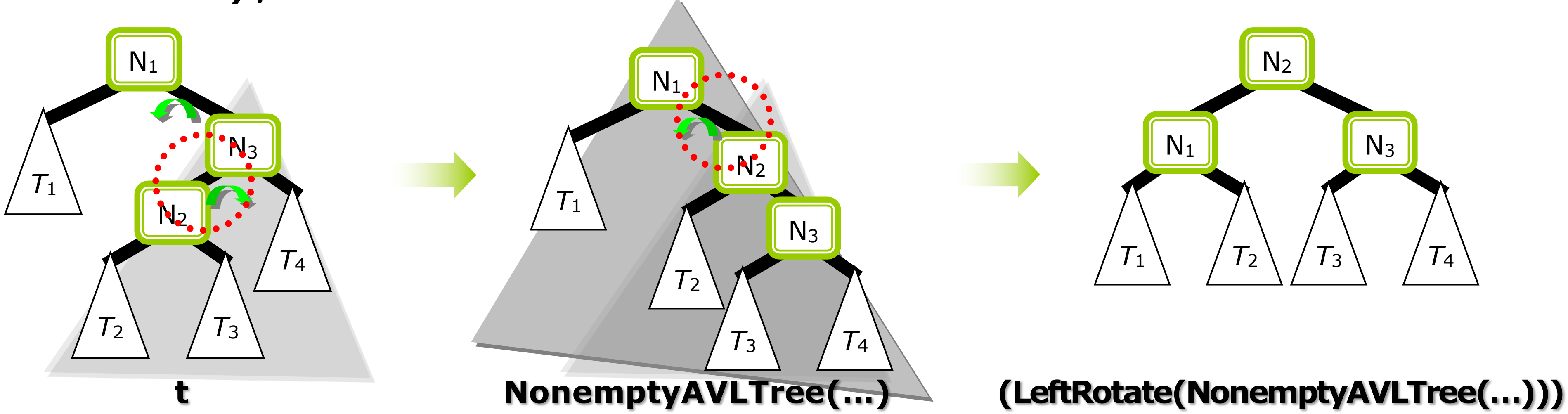
# Next, let’s write functions that perform DOUBLE rotations.

```
AVLTreeADT RightLeftRotate(AVLTreeADT t) {

  AVLTreeADT t1;

  t1 = NonemptyAVLTree(
            AVLRoot(t),                /*N1*/
            LeftAVLSubtree(t),         /*T1*/
            RightRotate(RightAVLSubtree(t)));

  return LeftRotate(t1);
}
```



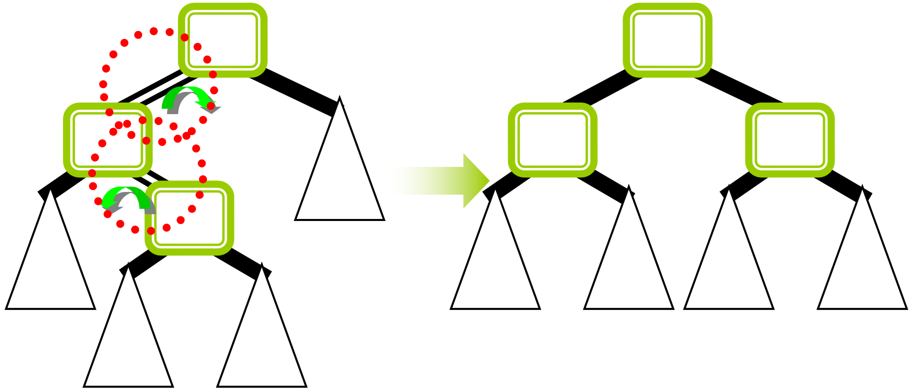


```
AVLTreeADT RightLeftRotate(AVLTreeADT t) {
  return LeftRotate(
                NonemptyAVLTree(
                    AVLRoot(t),
                    LeftAVLSubtree(t),
                    RightRotate(RightAVLSubtree(t))
                )
           );
}
```



## This is really even simpler (and again very logical)!

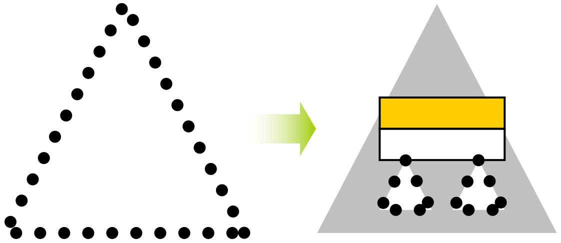
Note: I hope all of you can now write the LeftRightRotate function, which I do not write.



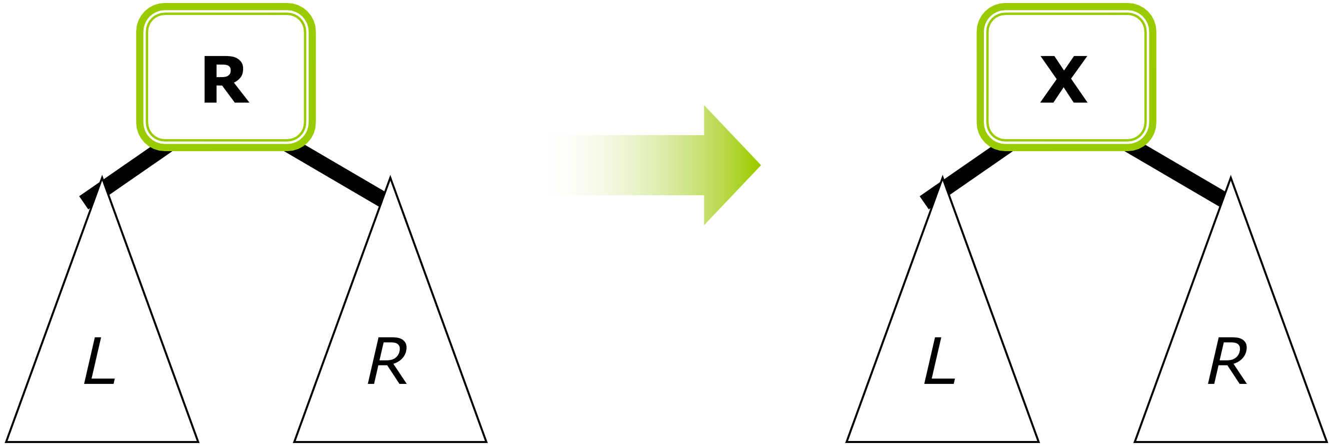
**We are now ready to write the AVLInsertNode function for AVLTreeADT.**

```
AVLTreeADT AVLInsertNode(TreeNodeADT X, AVLTreeADT T)

   if (AVLTreeIsEmpty(T))
      return NonemptyAVLTree(X, EmptyAVLTree(), EmptyAVLTree());
```

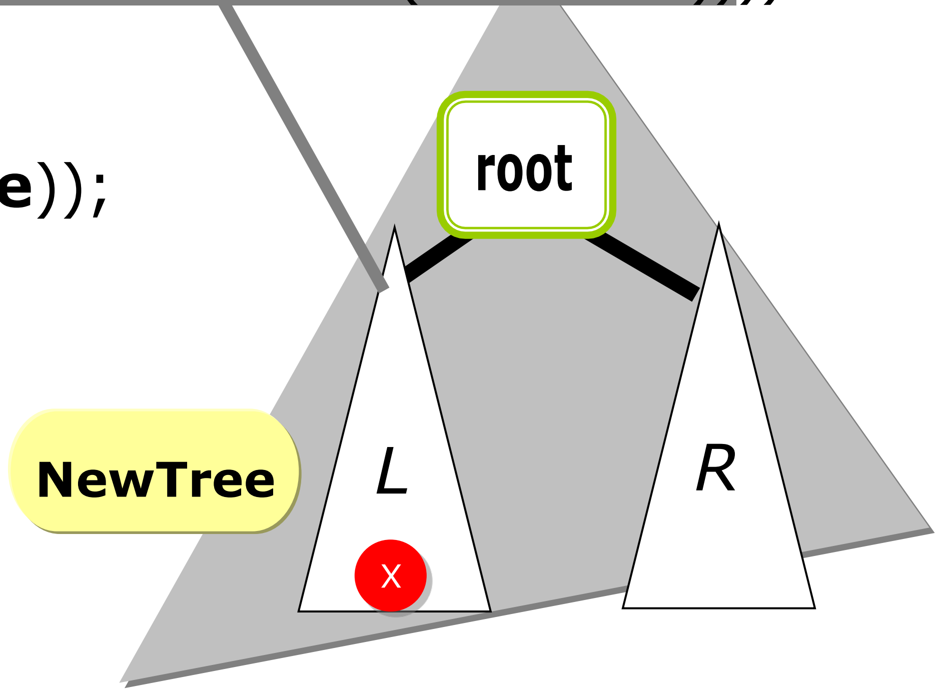
**/* to be continued on next page ... */**

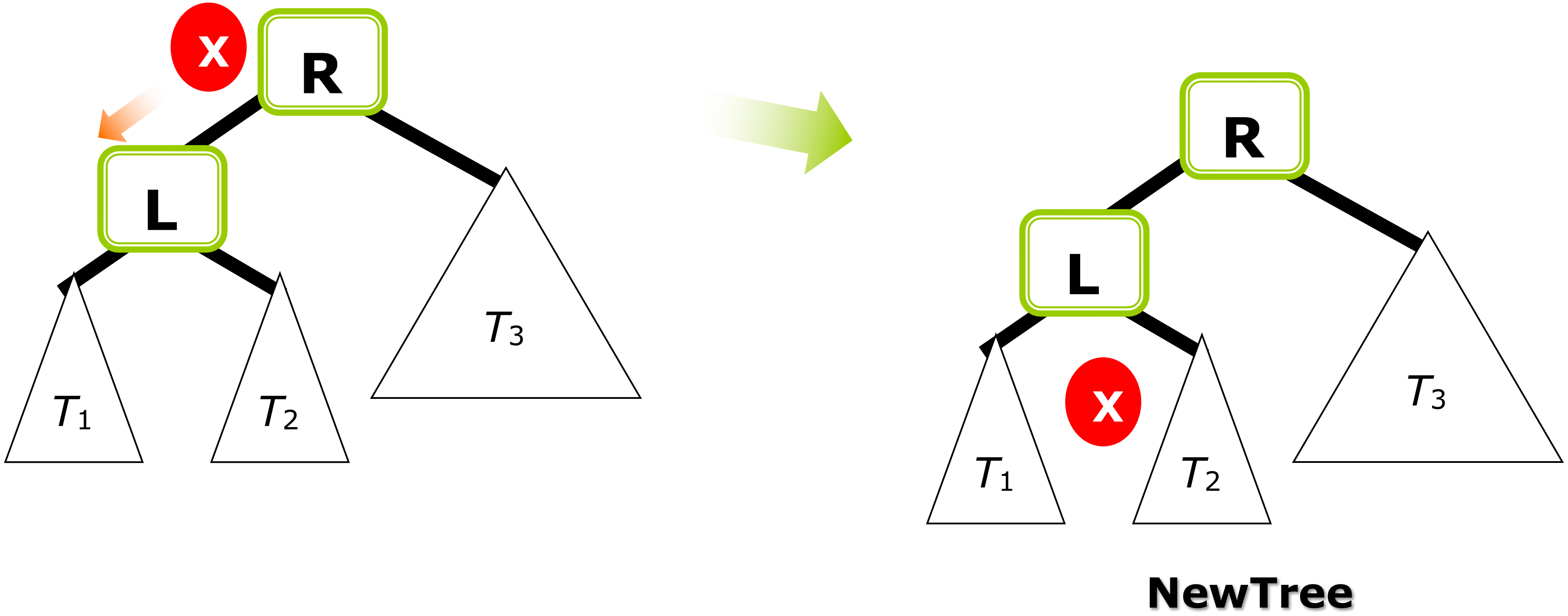


```
int sign = strcmp(GetNodeKey(X), GetNodeKey(AVLRoot(T)));

if (sign == 0)
   return NonemptyAVLTree(
         X, LeftAVLSubtree(T), RightAVLSubtree(T));

/* to be continued on next page … */
```

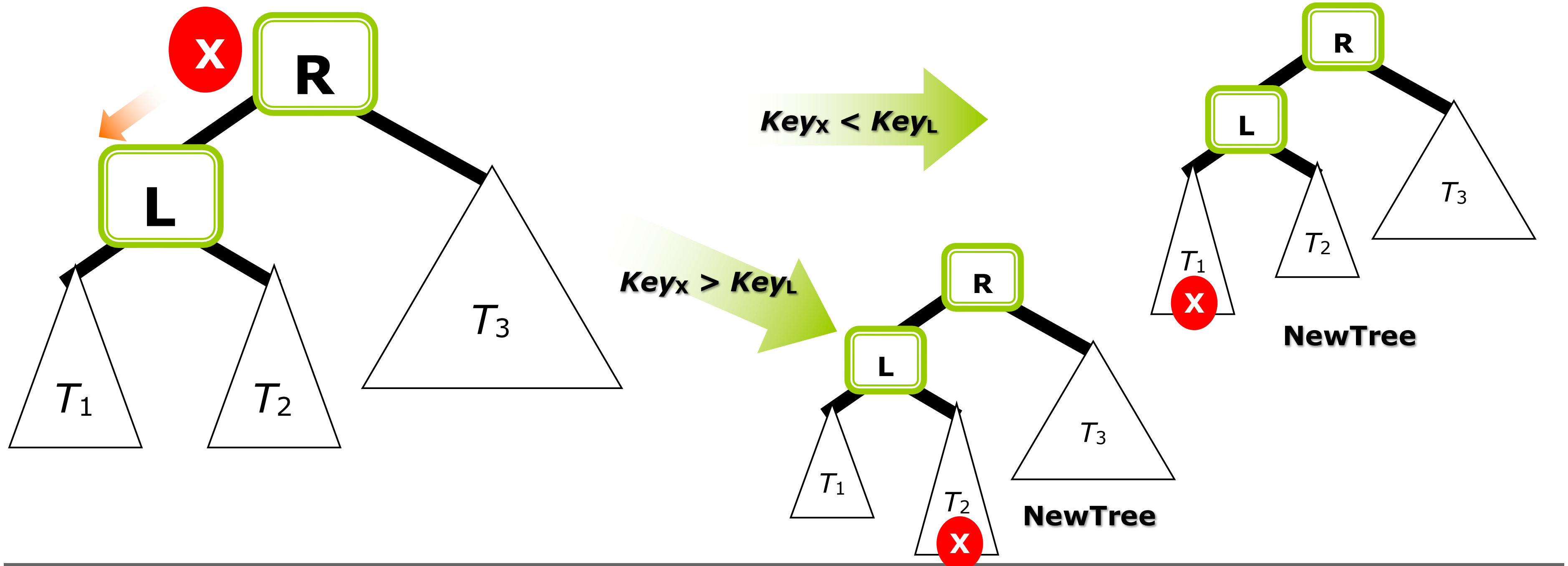


```
if (sign < 0) {
  AVLTreeADT NewTree = NonemptyAVLTree(AVLRoot(T),
                          AVLInsertNode(X, LeftAVLSubtree(T)),
                          RightAVLSubtree(T));

  if (AVLTreeHeight(LeftAVLSubtree(NewTree))
        - AVLTreeHeight(RightAVLSubtree(NewTree)) == 2)
    return (strcmp(GetNodeKey(X),
                 GetNodeKey(AVLRoot(LeftAVLSubtree(NewTree))))
                 < 0 ?
            RightRotate(NewTree) :
            LeftRightRotate(NewTree));
  return NewTree;
  };
```



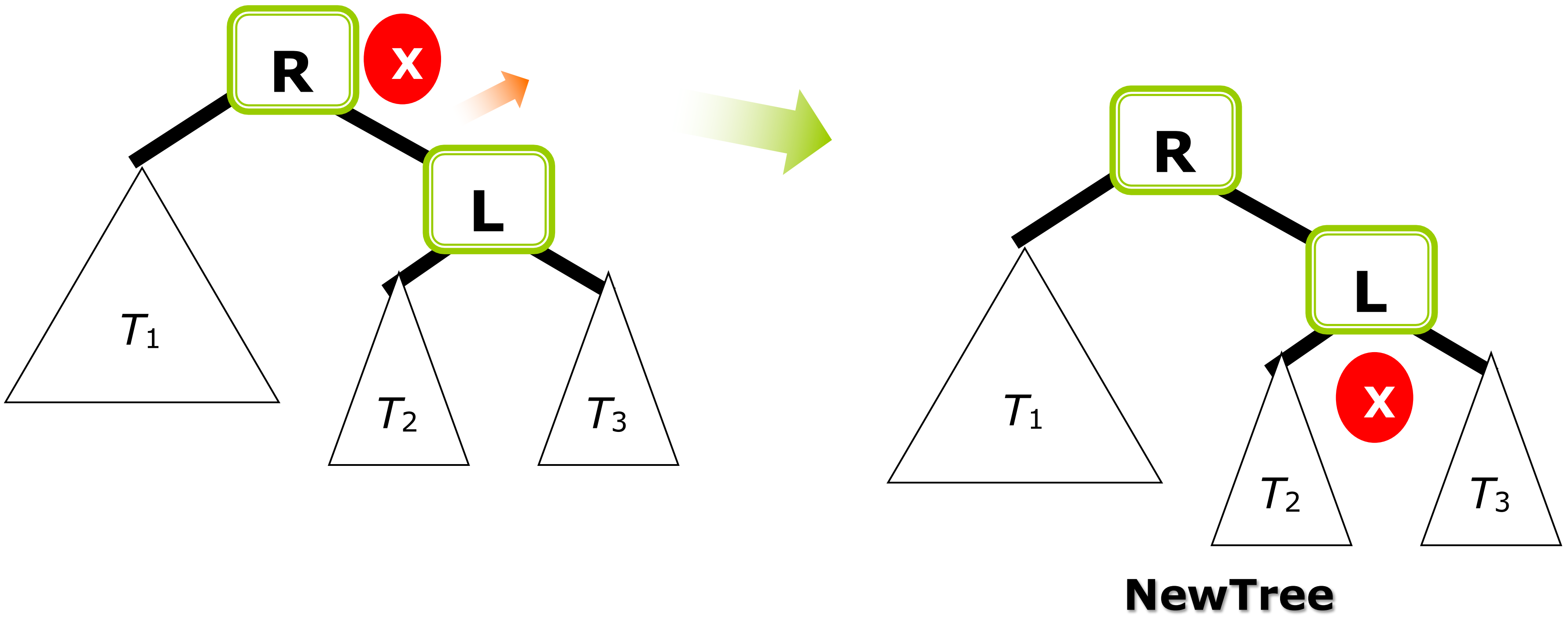


```
• if ($height_{\text{LeftSubtree(NewTree)}} - height_{\text{RightSubtree(NewTree)}}$ == 2)
        return …
  return NewTree;
```

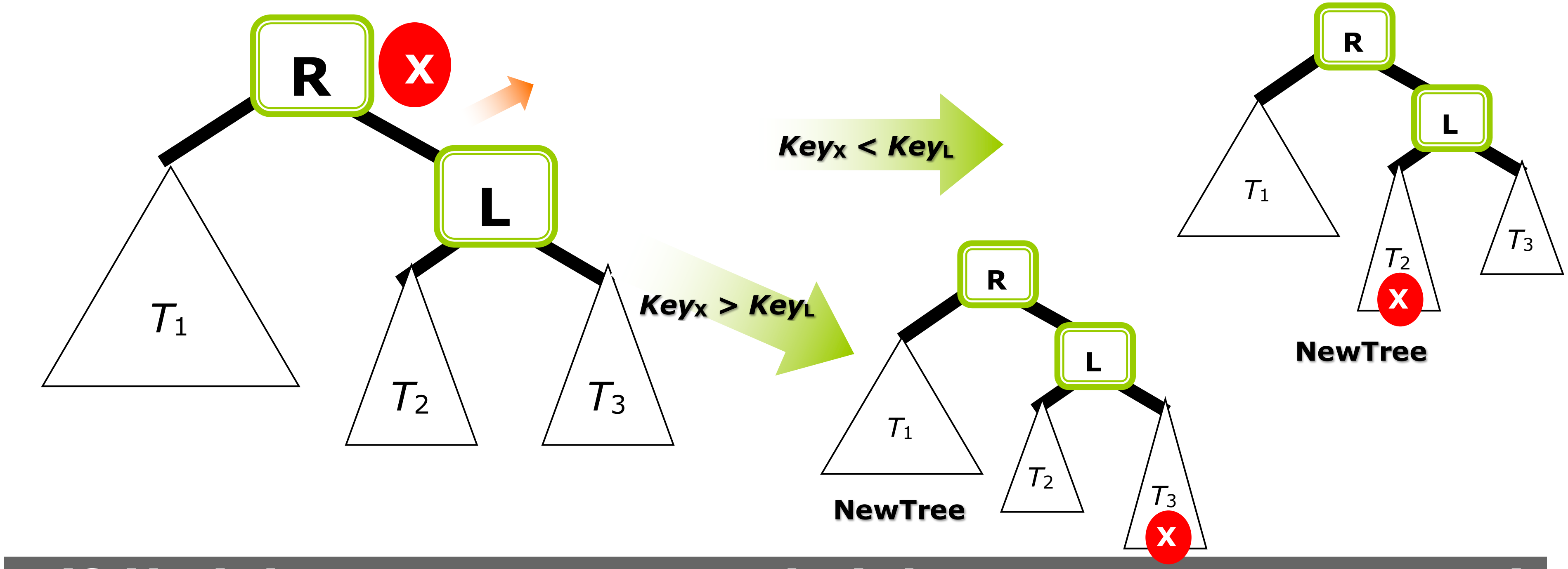


```
• if (height_LeftSubtree(NewTree) − height_RightSubtree(NewTree) == 2)
        return (Key_X < Key_L ?
                    RightRotate(NewTree) :
                    LeftRightRotate(NewTree));
  return NewTree;
```

```
if (sign > 0) {
   AVLTreeADT NewTree = NonemptyAVLTree(AVLRoot(T),
                           LeftAVLSubtree(T),
                           AVLInsertNode(X, RightAVLSubtree(T)));

   if (AVLTreeHeight(RightAVLSubtree(NewTree))
            – AVLTreeHeight(LeftAVLSubtree(NewTree)) == 2)
      return (strcmp(GetNodeKey(X),
                     GetNodeKey(AVLRoot(RightAVLSubtree(NewTree))))
                     > 0 ?
               LeftRotate(NewTree) :
               RightLeftRotate(NewTree));
   return NewTree;
   }
}
```



```
• if ($height_{RightSubtree(NewTree)} - height_{LeftSubtree(NewTree)}$ == 2)
        return …
  return NewTree;
```



```
• if ($height_{RightSubtree(NewTree)} - height_{LeftSubtree(NewTree)}$ == 2)
      return ($Key_X > Key_L$ ?
                LeftRotate(NewTree) :
                RightLeftRotate(NewTree));
  return NewTree;
```

```
AVLTreeADT AVLInsertNode(TreeNodeADT X, AVLTreeADT T)
    if (AVLTreeIsEmpty(T)) return NonemptyAVLTree(X, EmptyAVLTree(), EmptyAVLTree());
    int sign = strcmp(GetNodeKey(X), GetNodeKey(AVLRoot(T)));

    if (sign == 0) return NonemptyAVLTree(X, LeftAVLSubtree(T), RightAVLSubtree(T));
    if (sign < 0) {
        AVLTreeADT NewTree = NonemptyAVLTree(AVLRoot(T),
                                        AVLInsertNode(X, LeftAVLSubtree(T)),
                                        RightAVLSubtree(T));
        if (AVLTreeHeight(LeftAVLSubtree(NewTree)) – AVLTreeHeight(RightAVLSubtree(NewTree)) == 2)
            return (strcmp(GetNodeKey(X),  GetNodeKey(AVLRoot(LeftAVLSubtree(NewTree)))) < 0 ?
                    RightRotate(NewTree) :
                    LeftRightRotate(NewTree));
        return NewTree;
        };
    if (sign > 0) {
        AVLTreeADT NewTree = NonemptyAVLTree(AVLRoot(T),
                                        LeftAVLSubtree(T),
                                        AVLInsertNode(X, RightAVLSubtree(T)));
        if (AVLTreeHeight(RightAVLSubtree(NewTree)) – AVLTreeHeight(LeftAVLSubtree(NewTree)) == 2)
            return (strcmp(GetNodeKey(X), GetNodeKey(AVLRoot(RightAVLSubtree(NewTree)))) > 0 ?
                    LeftRotate(NewTree) :
                    RightLeftRotate(NewTree));
        return NewTree;
        }
}
```