National University of Computer & Emerging Sciences

AVL Trees

Georgy **Adelson-Velsky and Landis**' tree

AVL Trees: Implementation

```
struct AvlNode;
typedef struct AvlNode *Position;
typedef struct AvlNode *AvlTree;
AvlTree MakeEmpty( AvlTree T );
Position Find( ElementType X, AvlTree T );
Position FindMin( AvlTree T );
Position FindMax( AvlTree T );
AvlTree Insert( ElementType X, AvlTree T );
AvlTree Delete( ElementType X, AvlTree T );
ElementType Retrieve( Position P );
```

AVL Trees: Implementation

```
struct AvlNode
{
        ElementType Element;
        AvlTree  Left;
        AvlTree  Right;
        int      Height;
};
```

AVL Trees: Implementation

```
int Height(Position P )
{
        if( P == NULL )
            return -1;
        else
            return P->Height;
}
```

AVL Trees: Implementation

```
AvlTree Insert( ElementType X, AvlTree T )
{     if ( T == NULL )
      {         /* Create and return a one-node tree */
                T = new AvlNode;
                T->Element = X;
                T->Left = T->Right = NULL;
       }
       else
```

AVL Trees: Implementation

```
if( X < T->Element )
{
    T->Left = Insert( X, T->Left );
    if( Height( T->Left ) - Height( T->Right ) == 2 )
        if( X < T->Left->Element )
            T = SingleRotateWithLeft( T );
        else
            T = DoubleRotateWithLeft( T );
}
```

AVL Trees: Implementation

```
else
        if( X > T->Element )
        {
            T->Right = Insert( X, T->Right );
            if( Height( T->Right ) - Height( T->Left ) == 2 )
                if( X > T->Right->Element )
                    T = SingleRotateWithRight( T );
                else
                    T = DoubleRotateWithRight( T );
        }
```

AVL Trees: Implementation

```
    /* Else X is in the tree already; we'll do nothing */
    T->Height = Max( Height( T->Left ), Height( T->Right ) ) + 1;
    return T;
}
```

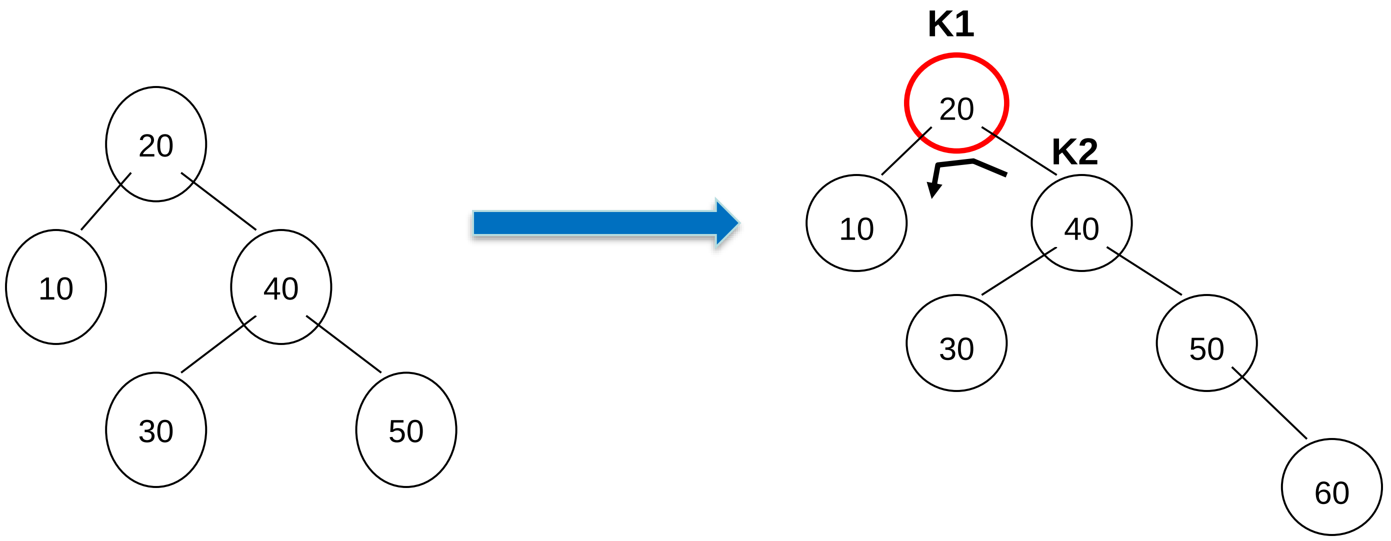
AVL Trees: Implementation

```
Position SingleRotateWithRight( Position K1 )
{
    Position K2;

    K2 = K1->Right;
    K1->Right = K2->Left;
    K2->Left = K1;
    K1->Height = Max( Height(K1->Left), Height(K1->Right) ) + 1;
    K2->Height = Max( Height(K2->Right), K1->Height ) + 1;
    return K2;  /* New root */
}
```

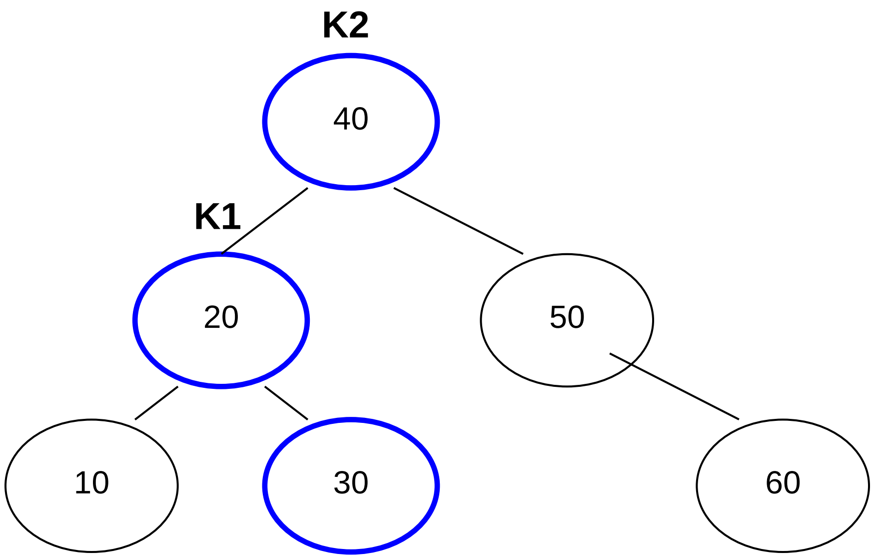
Case 1: Right heavy (RR)

- Add value 60
- Left rotation is performed



Case 1: Right heavy (RR)

- Resulting Tree after left Rotation



AVL Trees: Implementation

```
Position SingleRotateWithLeft( Position K1 )
{
   Position K2;

   K2 = K1->Left;
   K1->Left = K2->Right;
   K2->Right = K1;
   K1->Height = Max( Height(K1->Left), Height(K1->Right) ) + 1;
   K2->Height = Max( Height(K2->Left), K1->Height ) + 1;
   return K2;  /* New root */
}
```

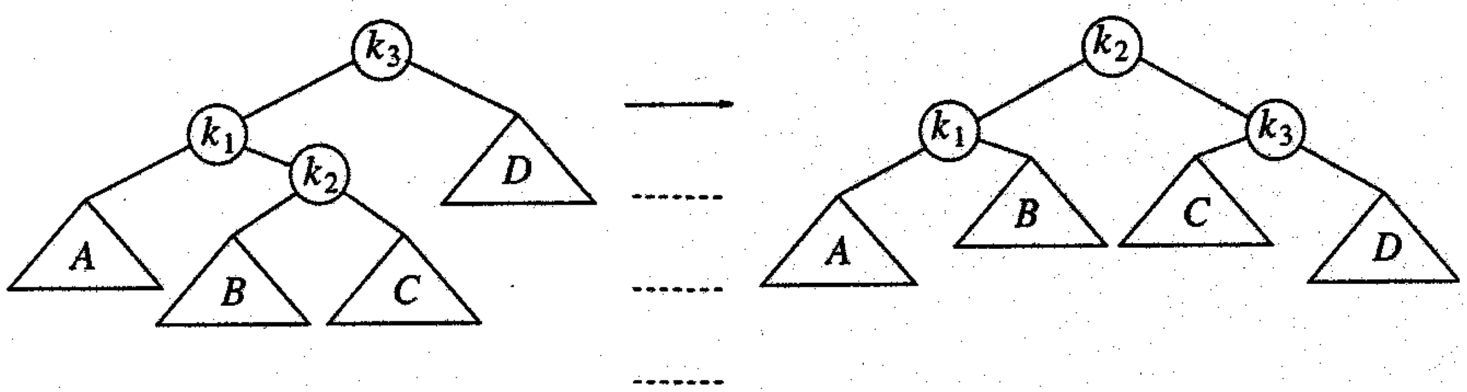
AVL Trees: Implementation

```
Position DoubleRotateWithLeft( Position K3 )
{
    /* Rotate between K1 and K2 */
    K3->Left = SingleRotateWithRight( K3->Left );
    /* Rotate between K3 and K2 */
    return SingleRotateWithLeft( K3 );
}
```

Single Left Rotation at K1

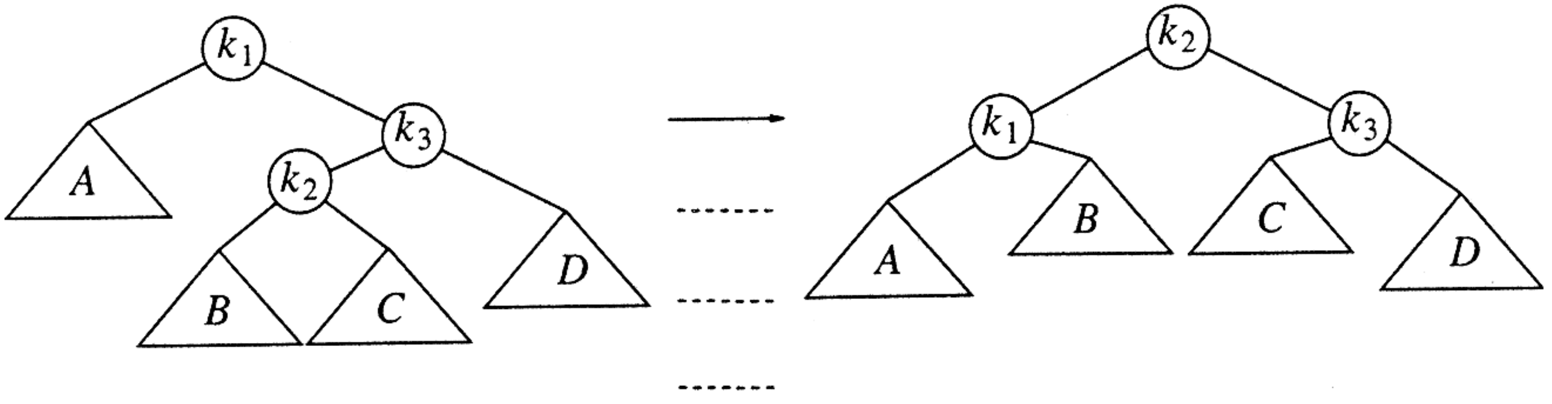
Single Right rotation at K3

Figure 4.35 Left–right double rotation to fix case 2



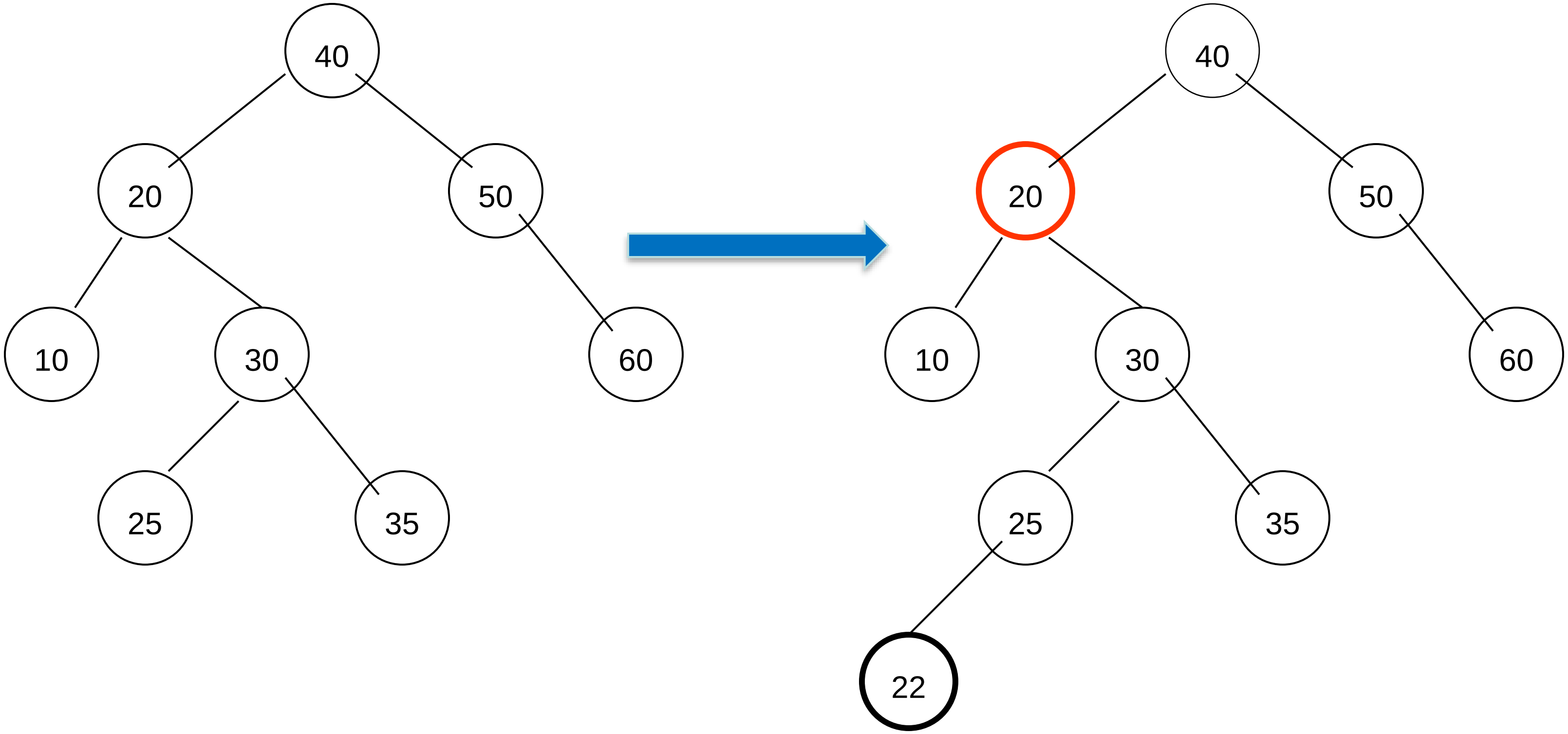
AVL Trees: Implementation

```
Position DoubleRotateWithRight( Position K1)
{
    /* Rotate between K3 and K2 */
    K1->Right = SingleRotateWithLeft( K1->Right );
    /* Rotate between K1 and K2 */
    return SingleRotateWithRight( K1 );
}
```

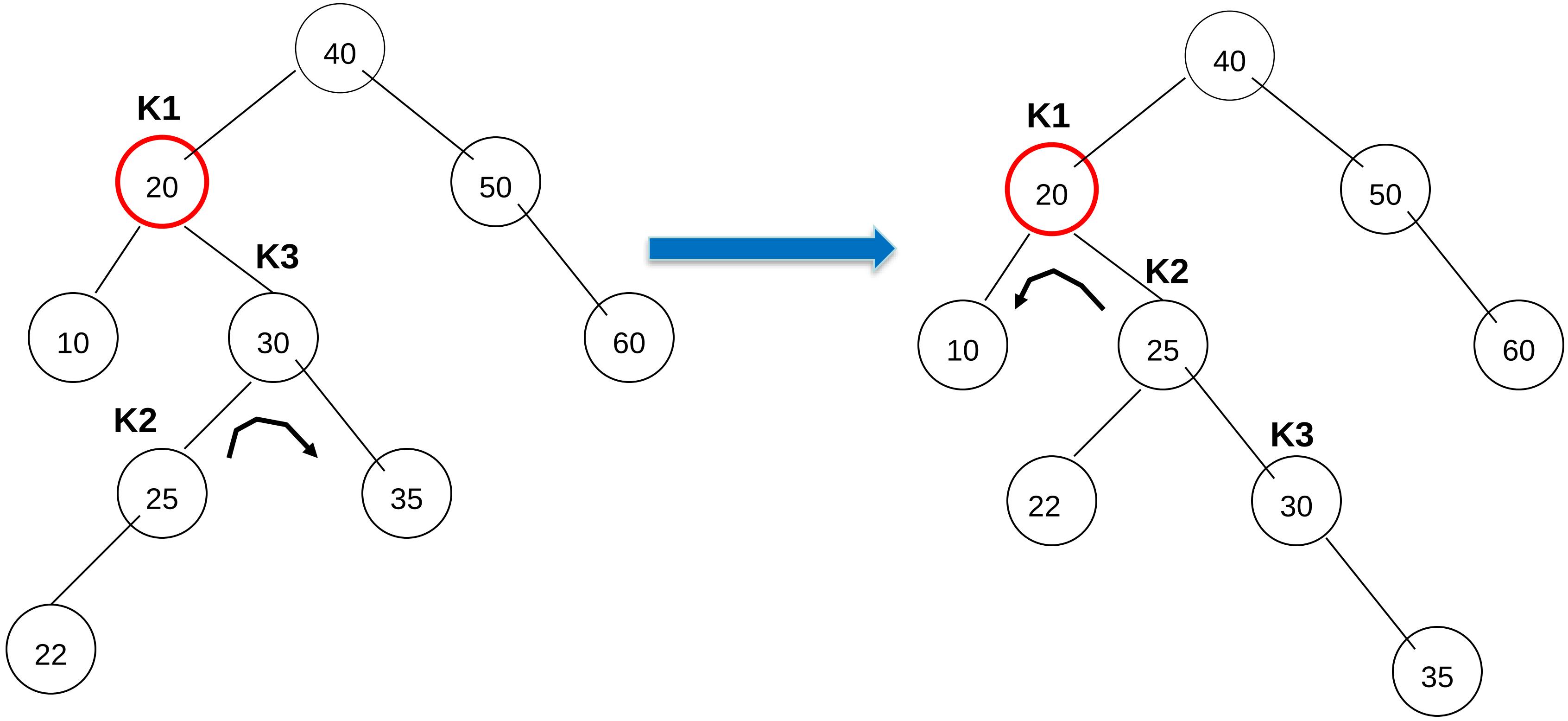


Case: Left heavy (LR)

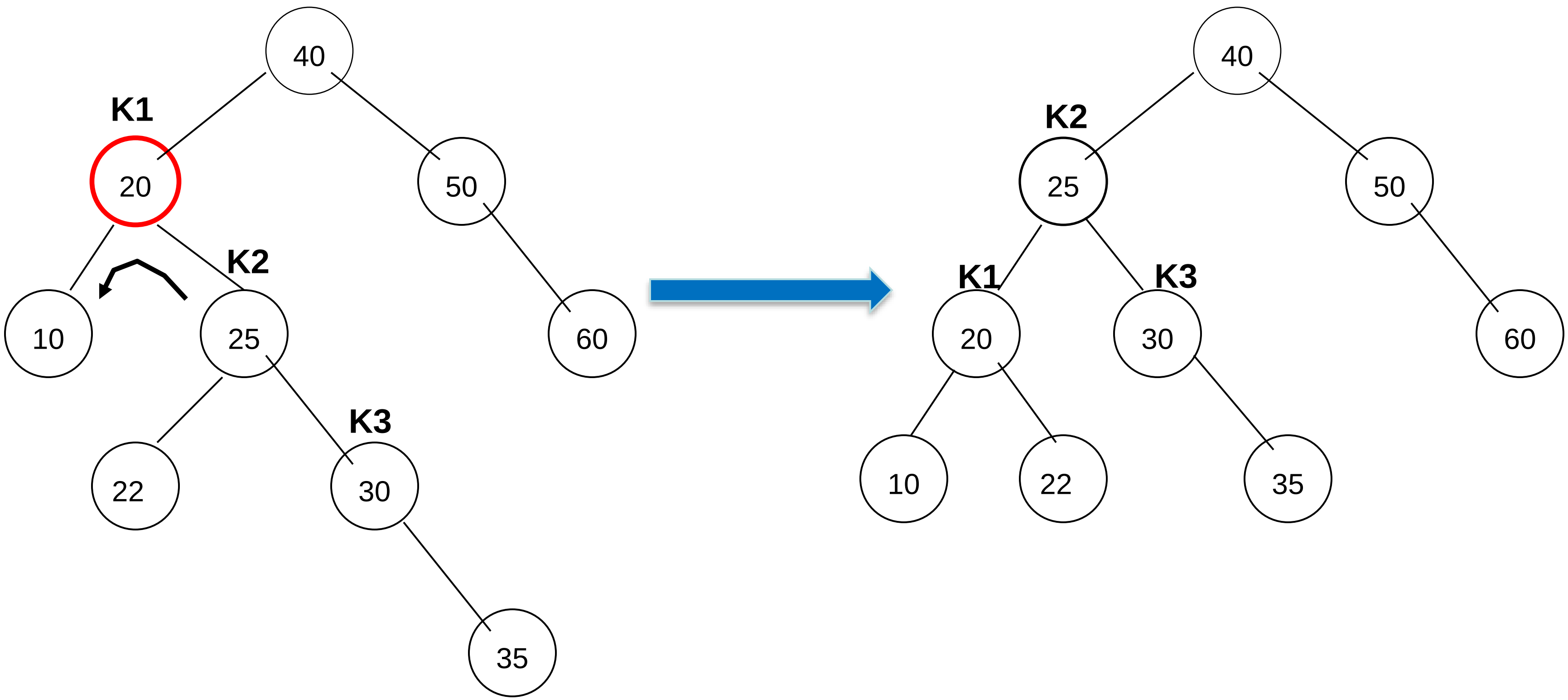
- Adding node 22
  - Requires double rotation!



Case 2:Left heavy



Case 2:Left Subtree is higher



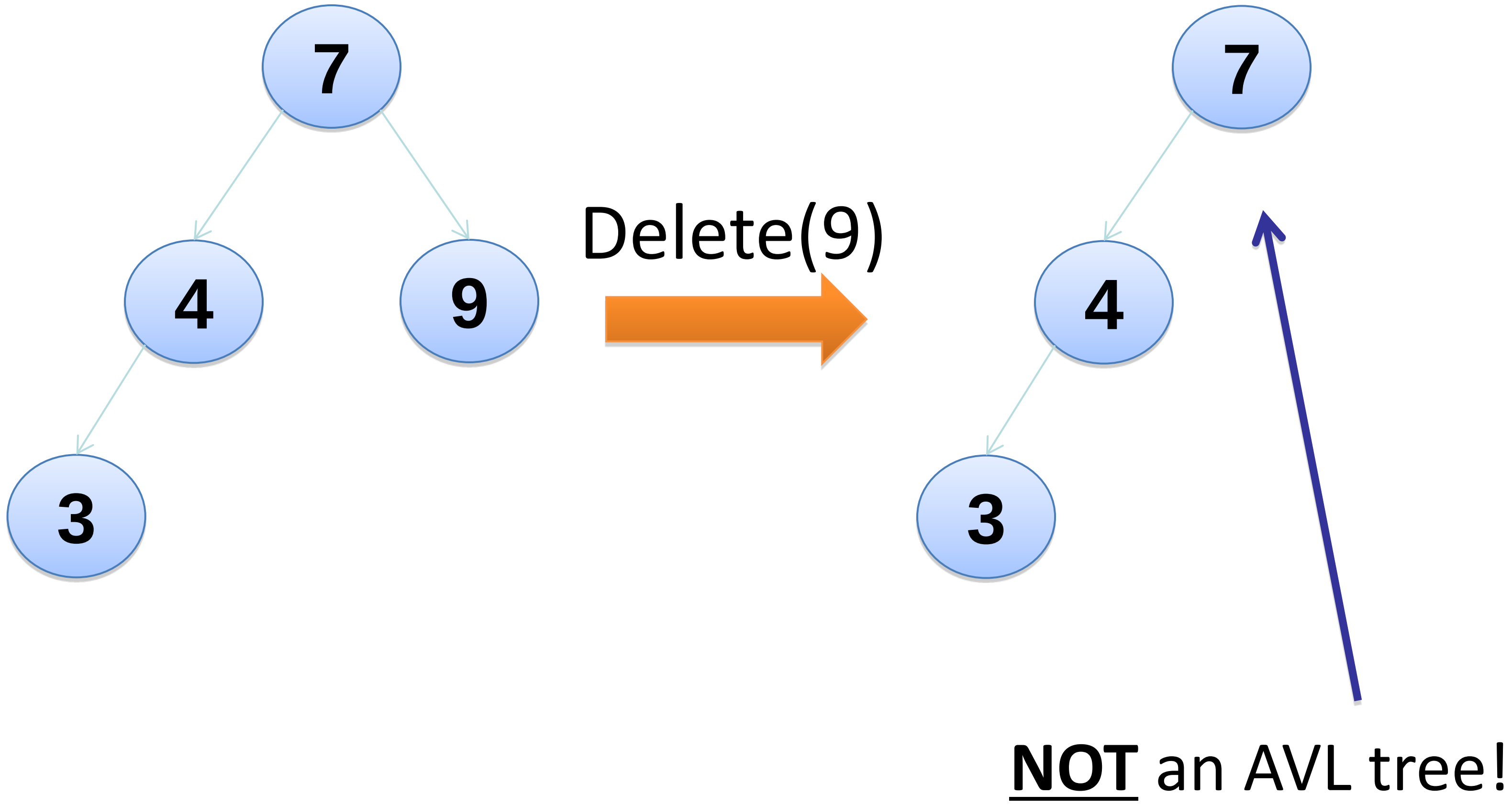
AVL TREE DELETION

AVL tree - deletion

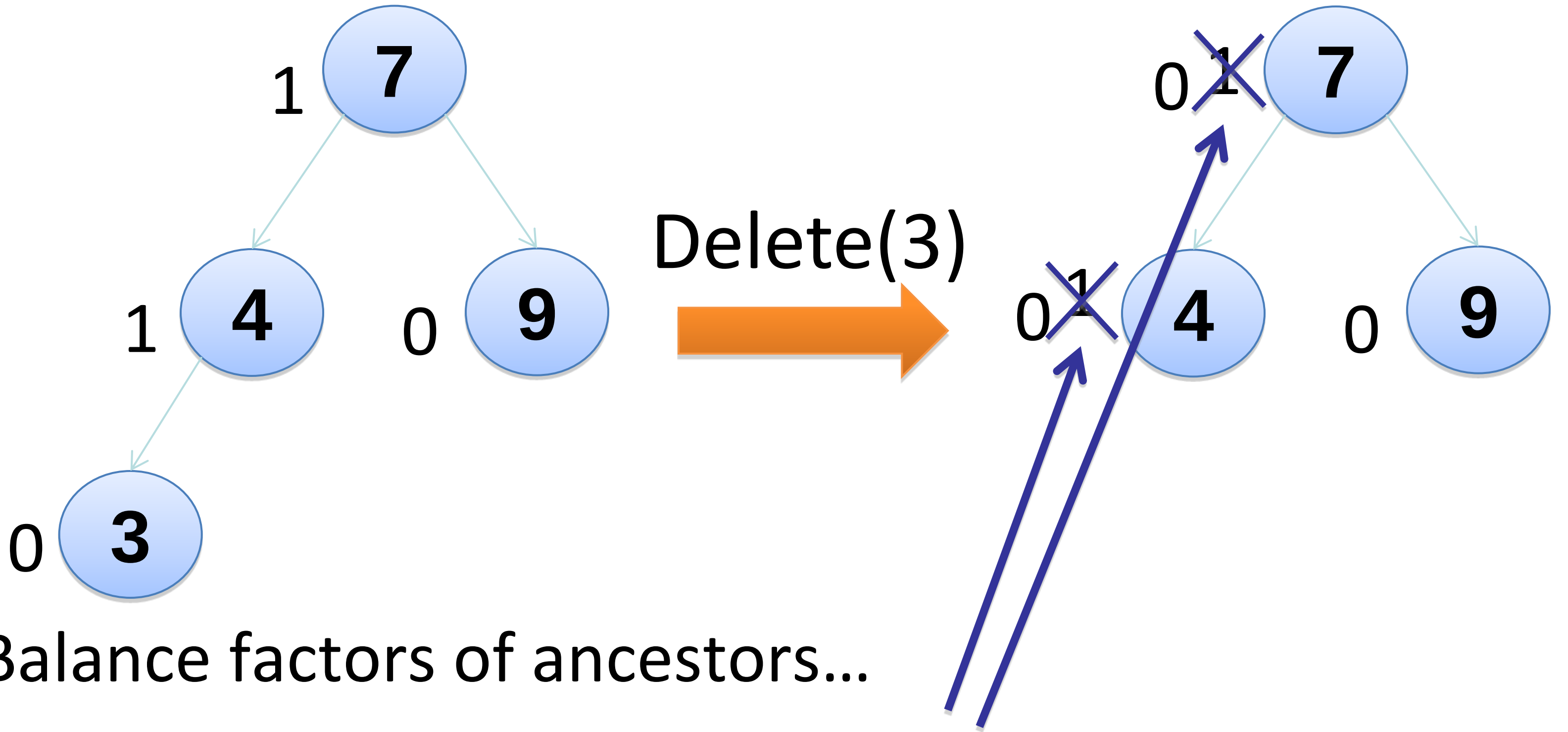
- To be an AVL tree, must **always:**
  (1) Be a ***binary search tree***
  (2) Satisfy the ***height constraint***

- Suppose we start with an AVL tree, then delete as a regular BST.
- Will the tree be an AVL tree after the delete?
  (1) It will still be a BST… that’s one part.
  (2) Will it satisfy the ***height constraint***?

BST Delete breaks an AVL tree



What else can BST Delete break?

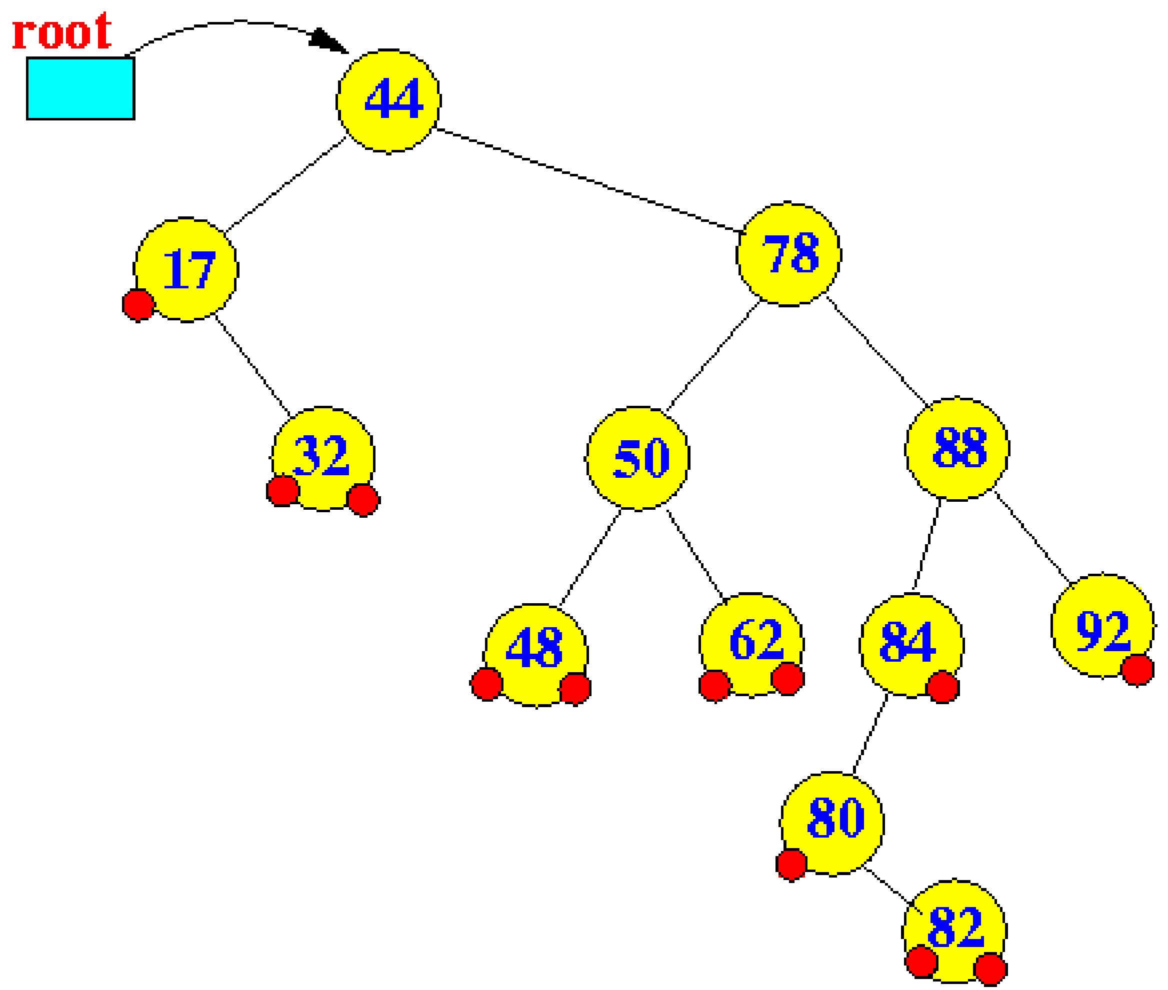


- Balance factors of ancestors…

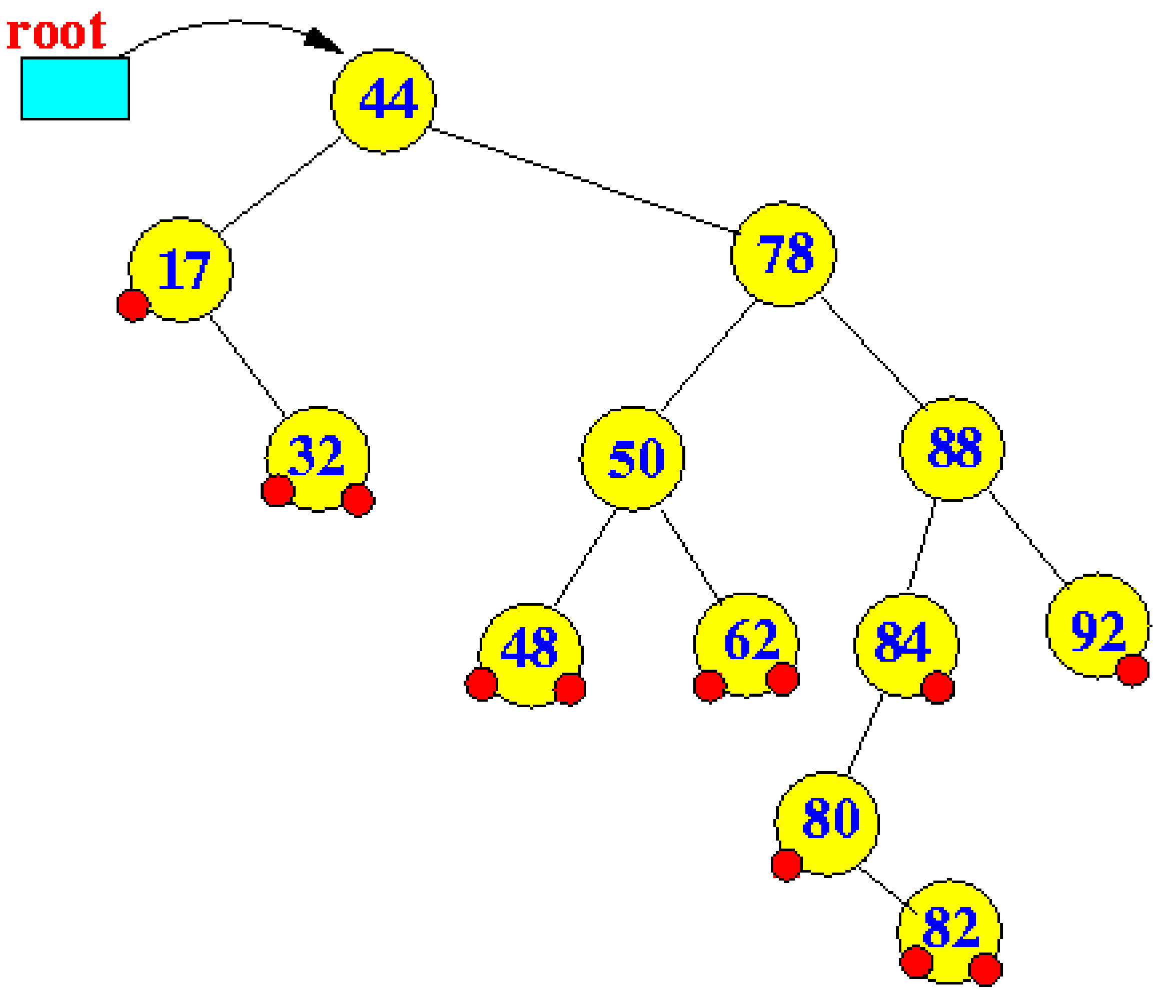
Need a new Delete algorithm

- We are starting to see what our delete algorithm must look like.
- **Goal:** if tree is AVL before Delete, then tree is AVL after Delete.
- **Step 1:** do BST delete.
  - This maintains the *BST property*, but can BREAK the *balance factors of ancestors!*
- **Step 2:** fix the balance constraint.
  - Do something that maintains the BST property, but fixes any balance factors that are < -1 or > 1.

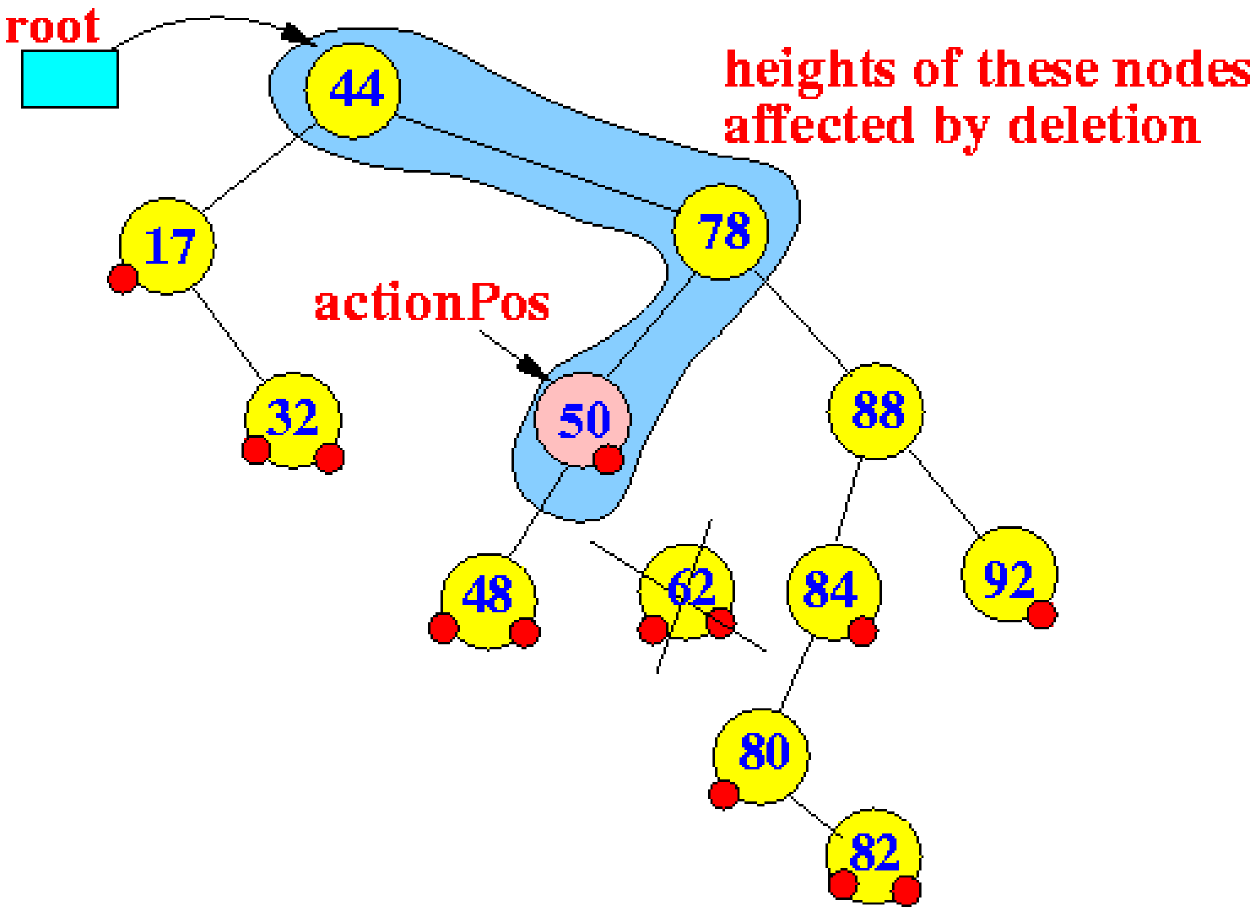
BST delete example (Review)



Deleting a *leaf* node (no children nodes): easy, just delete away....



AVL Trees - Implementation

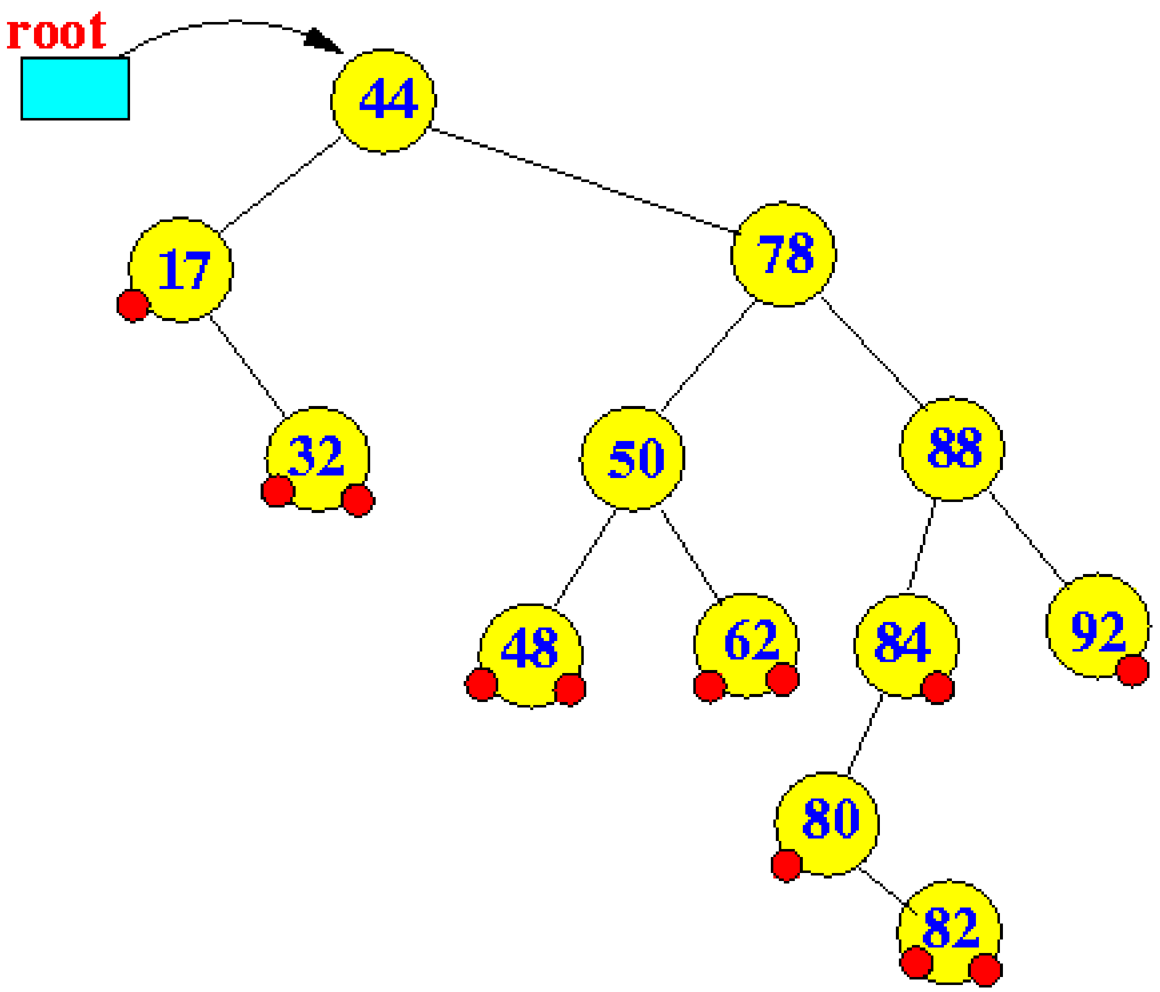


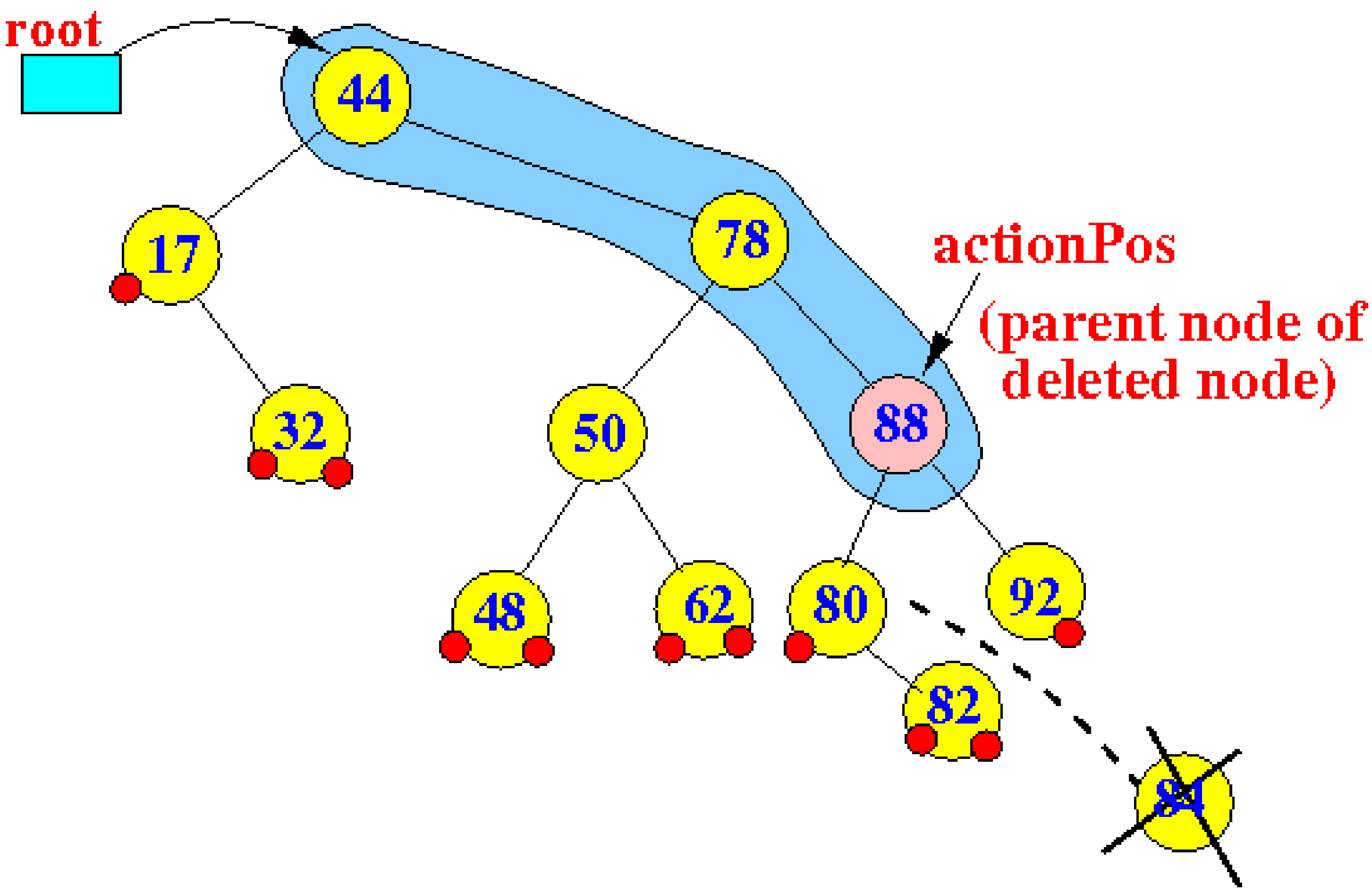
Note: action position

- The **action position** is a **reference** to the **parent node** from which a **node** has been ***physically* removed**
- The **action position** indicate the **first node** whose **height** has been **affected (possibly changed)** by the **deletion**

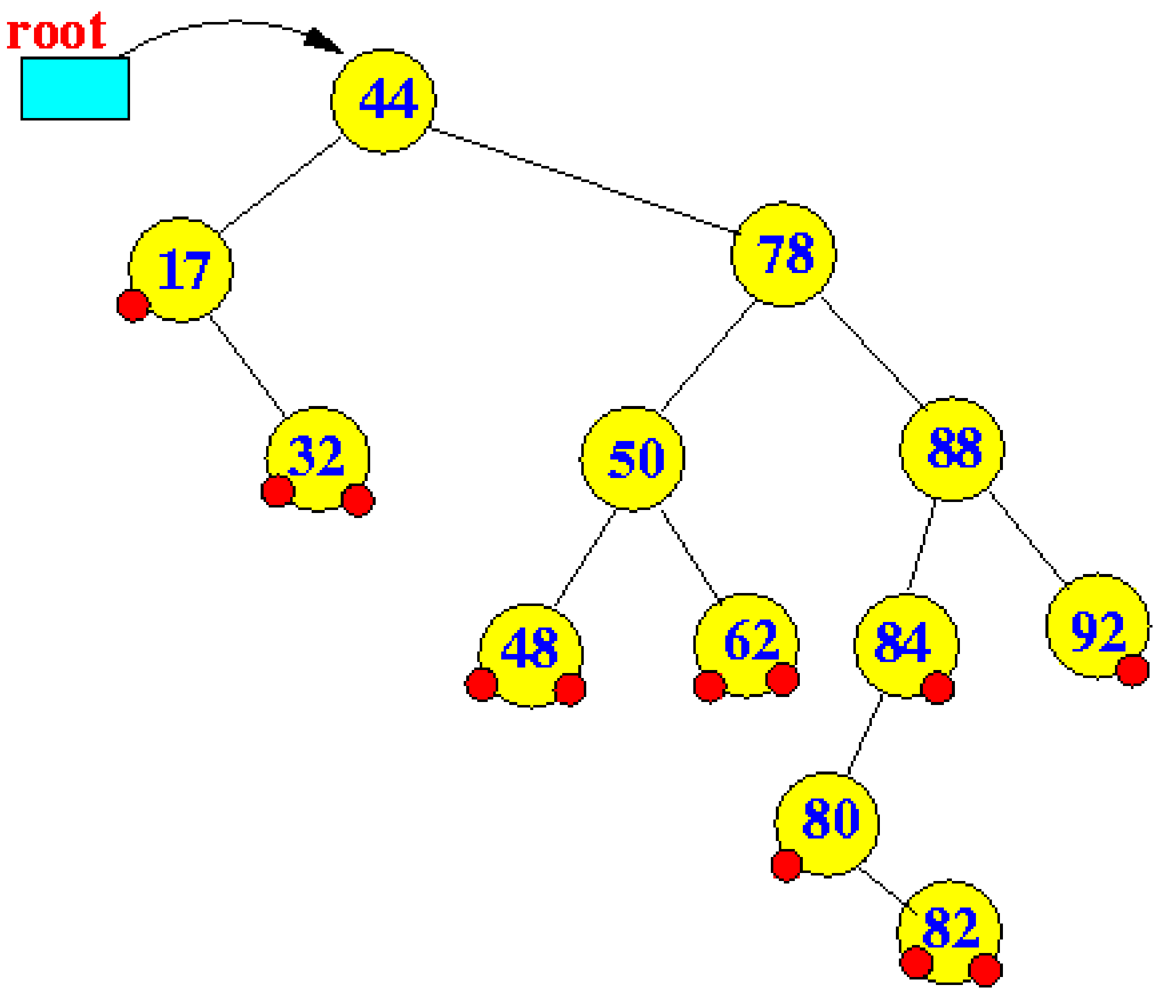
  (This will be **important in the re-balancing phase to adjust the tree back to an AVL tree**)

Deleting a node with *1 child node*: easy, connect its parent and child....

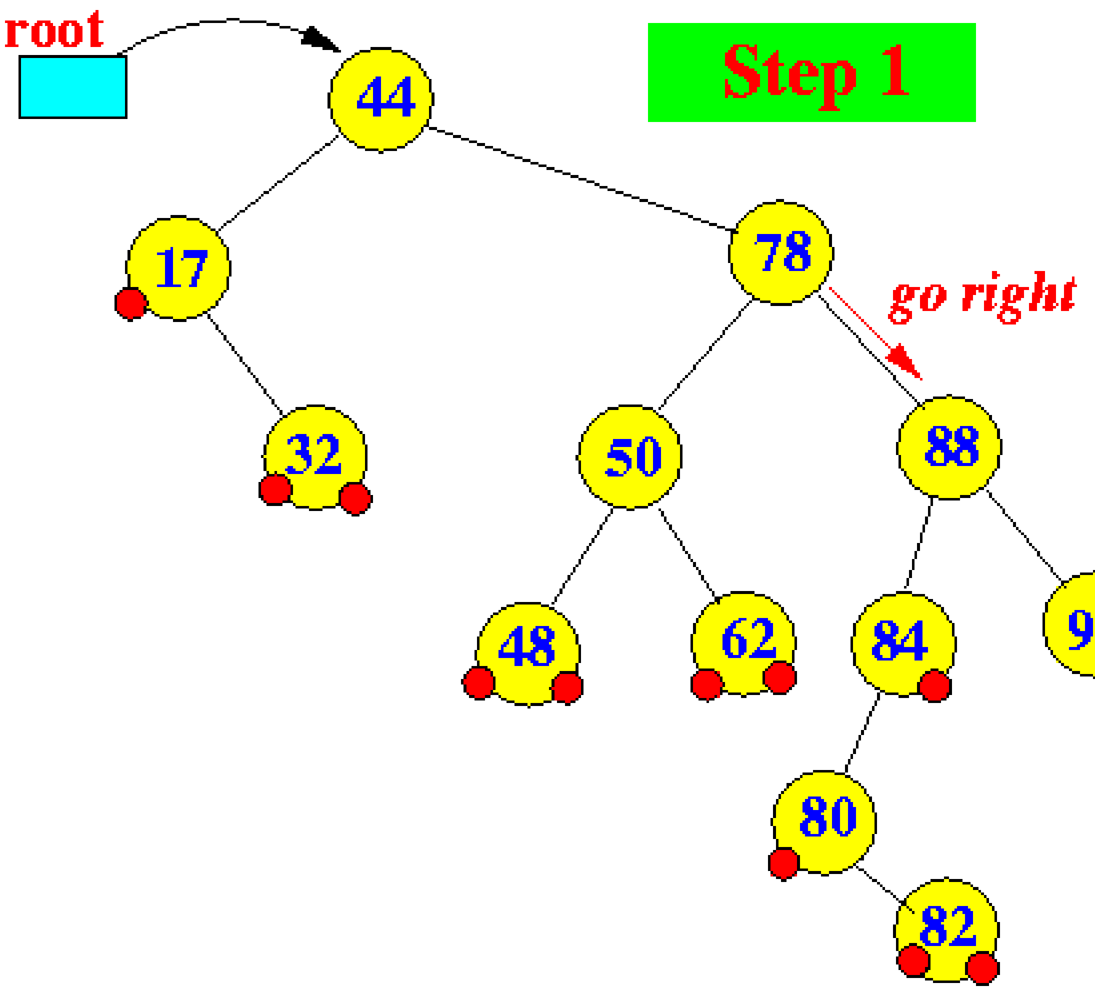


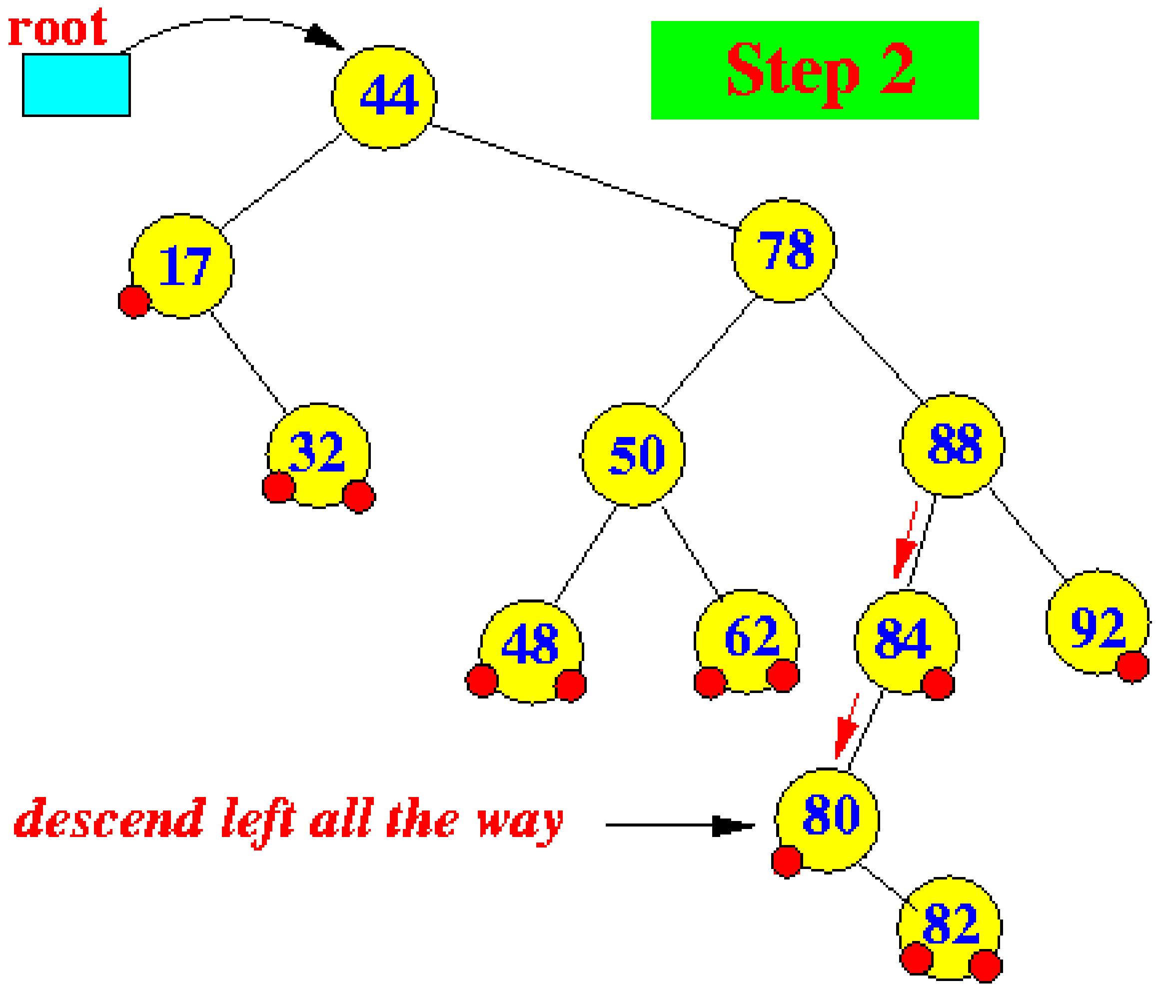

root
44
17
78
32
50
88
actionPos
(parent node of
deleted node)
48
62
80
92
82
84

Deleting a node with *2 children nodes*

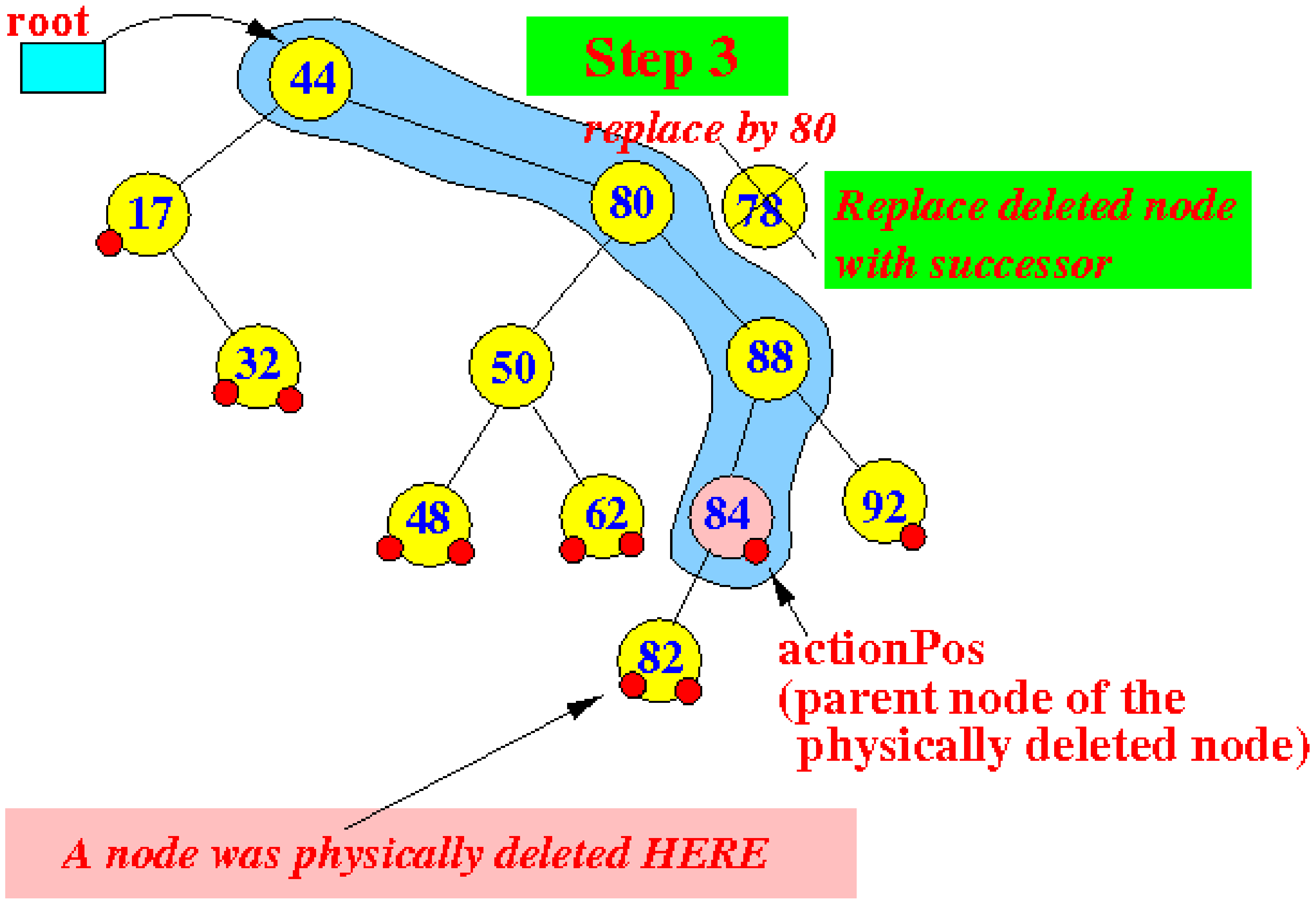


Deleting a node with *2 children nodes*

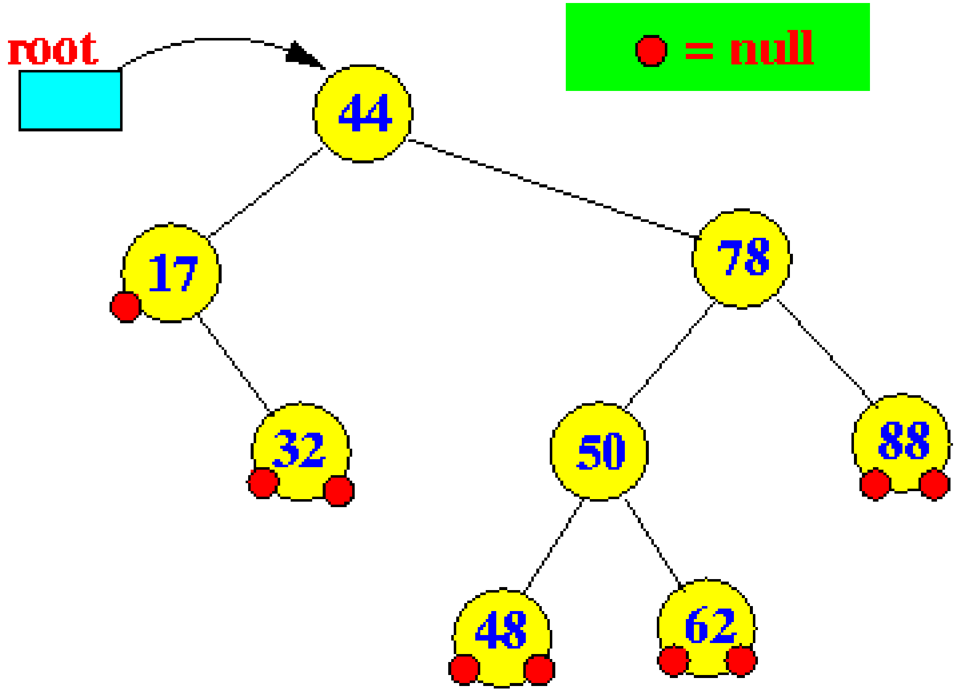



root
Step 2
44
17
78
32
50
88
48
62
84
92
descend left all the way
80
82

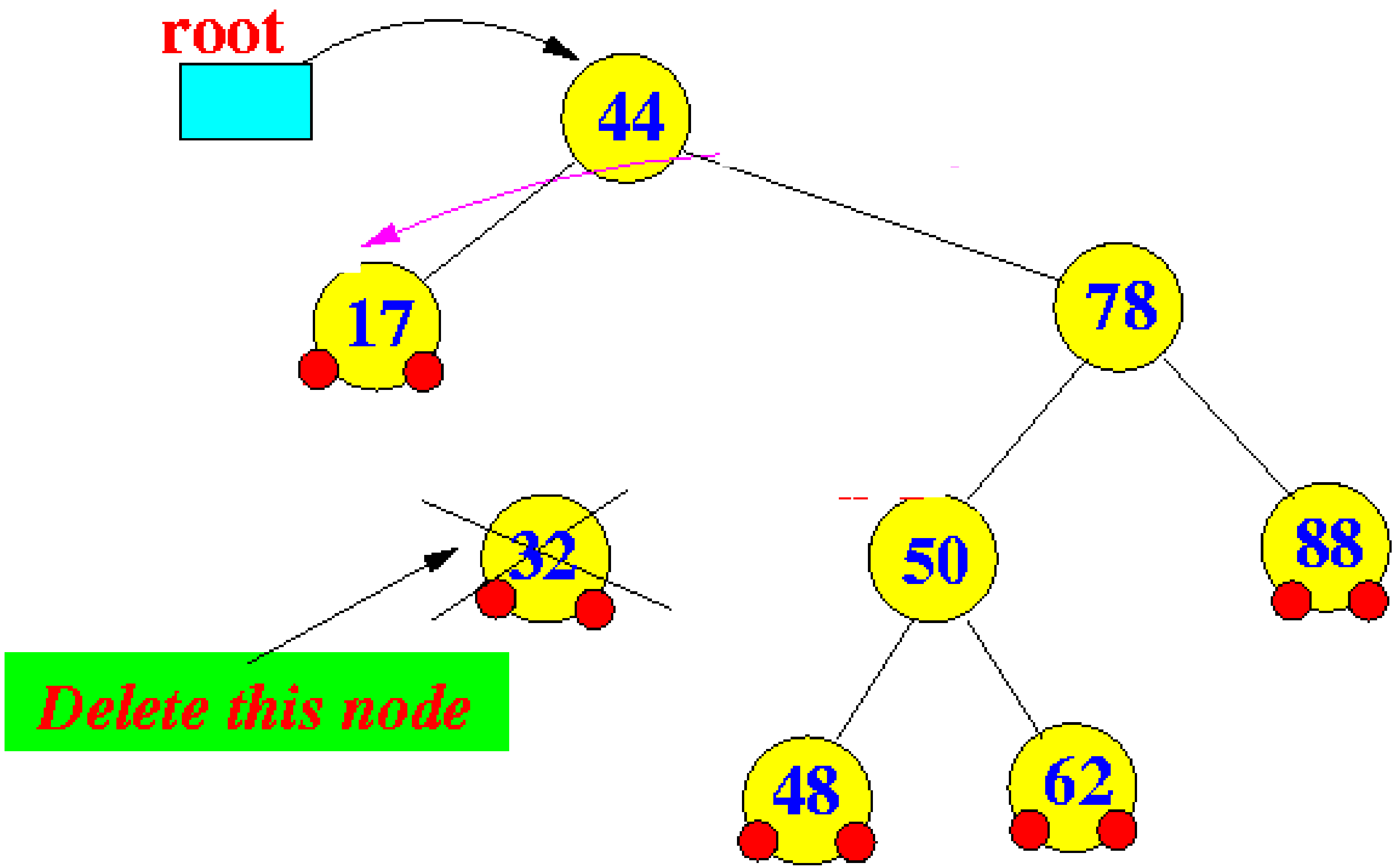
AVL Trees - Implementation



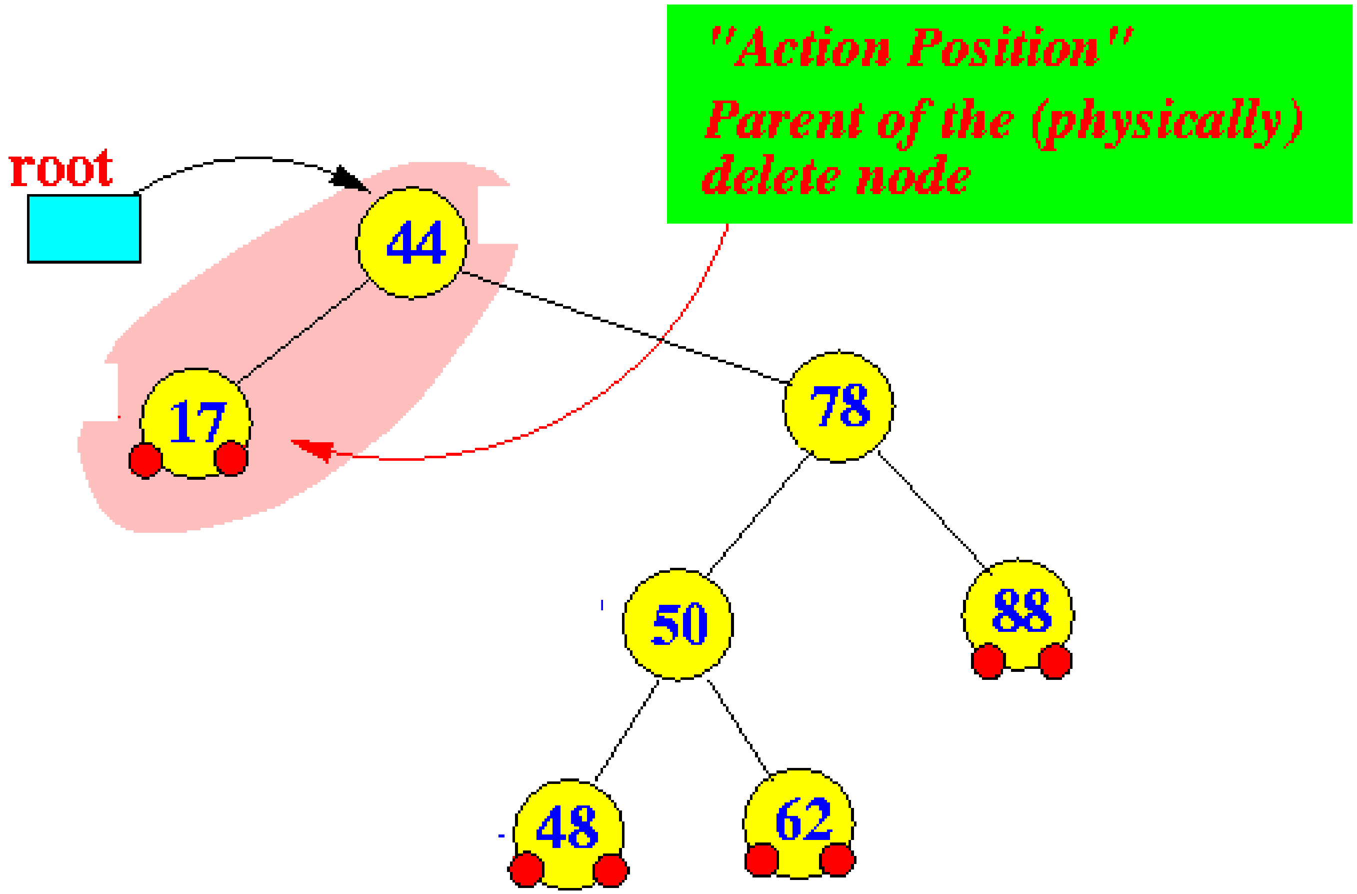
Deletion in an AVL tree can *also* cause imbalance



Deleting an entry (node) can also cause an AVL tree to become height unbalanced:



The height changes at *only* nodes between the root and the parent node of the *physically* deleted node



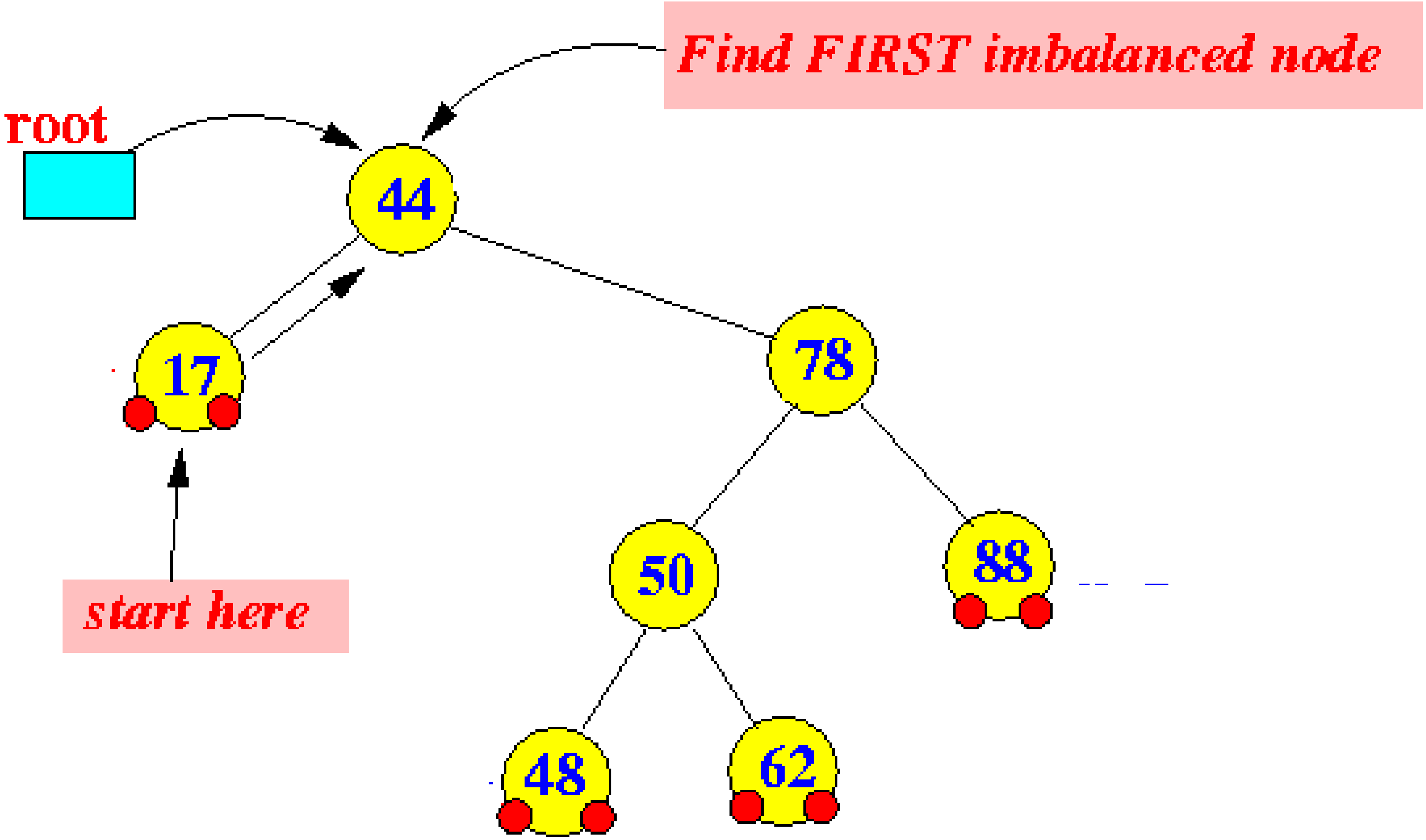
Re-balancing the AVL tree after a delete operation:

- Just like **insert operation**, we can use the **rotations** to **re-balance** an **out-of-balanced** AVL tree.

- ***How*** to **apply** the **rotations** is a bit ***tricky*** in the **delete operation**.

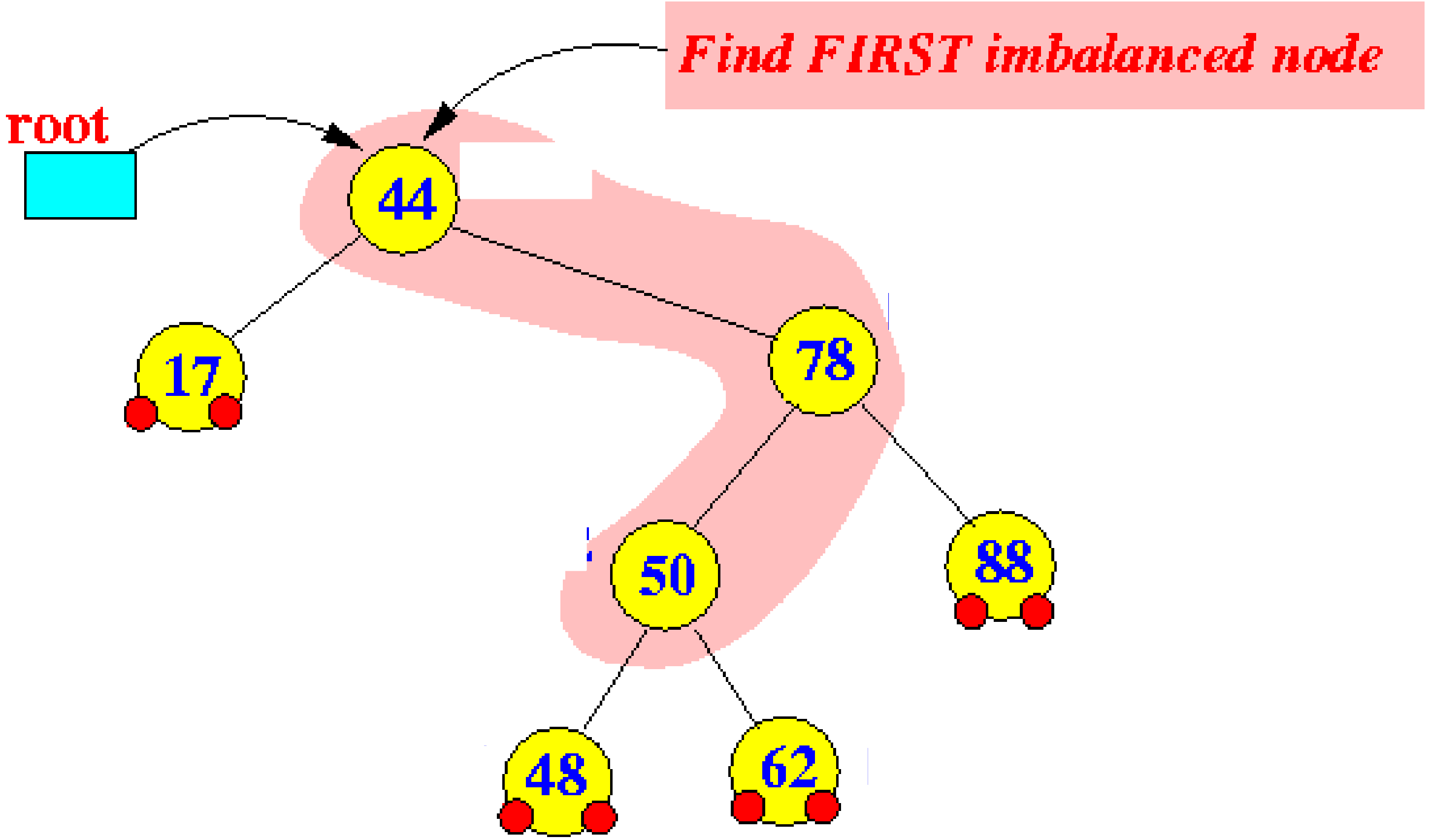
Re-balancing an AVL tree after deletion:

Starting at the action position (= parent node of the *physically* deleted node), find the *first* imbalanced node (This step is exactly the same as in insert)



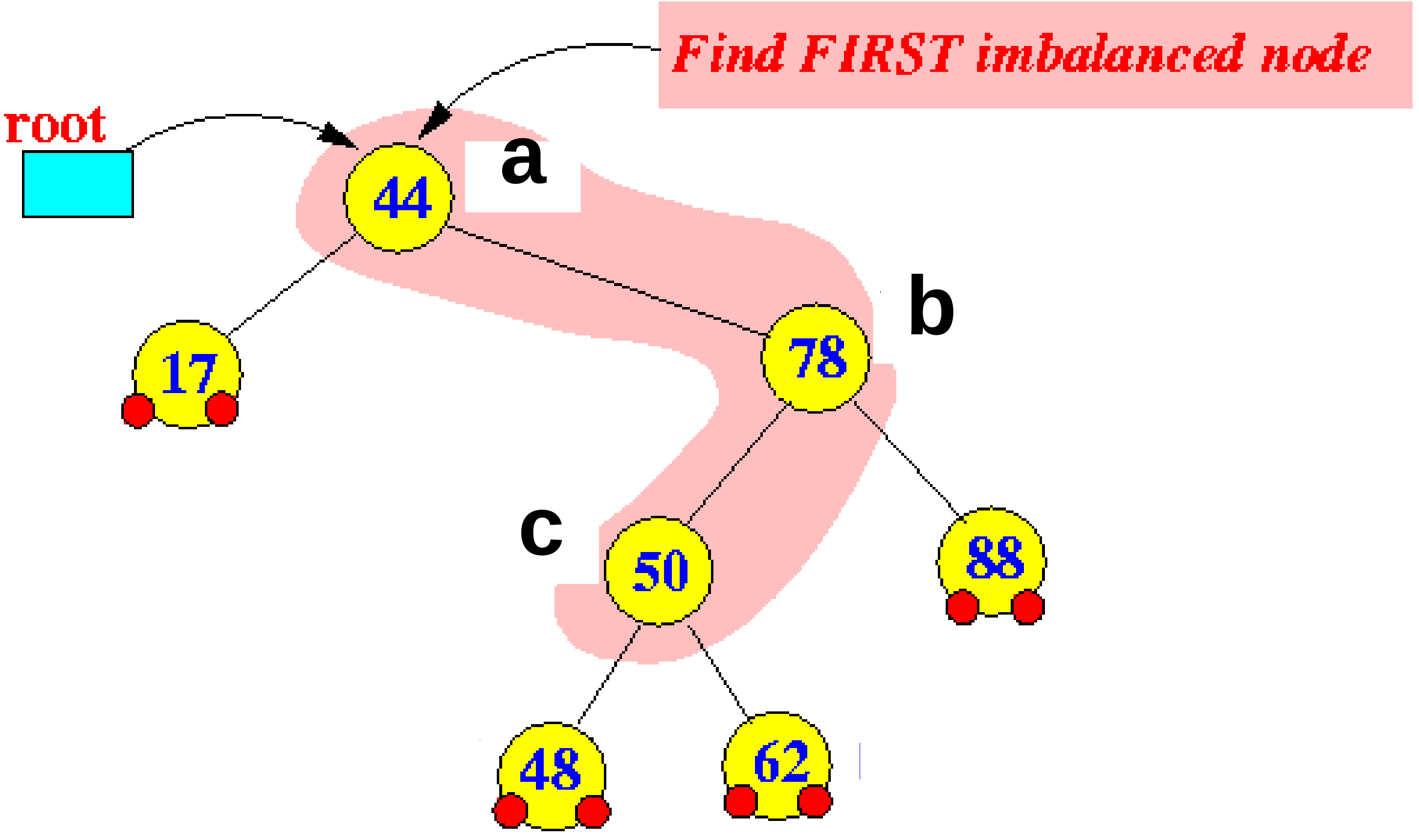
Re-balancing an AVL tree after deletion:

Perform a rotation using *these* 3 nodes (shaded):



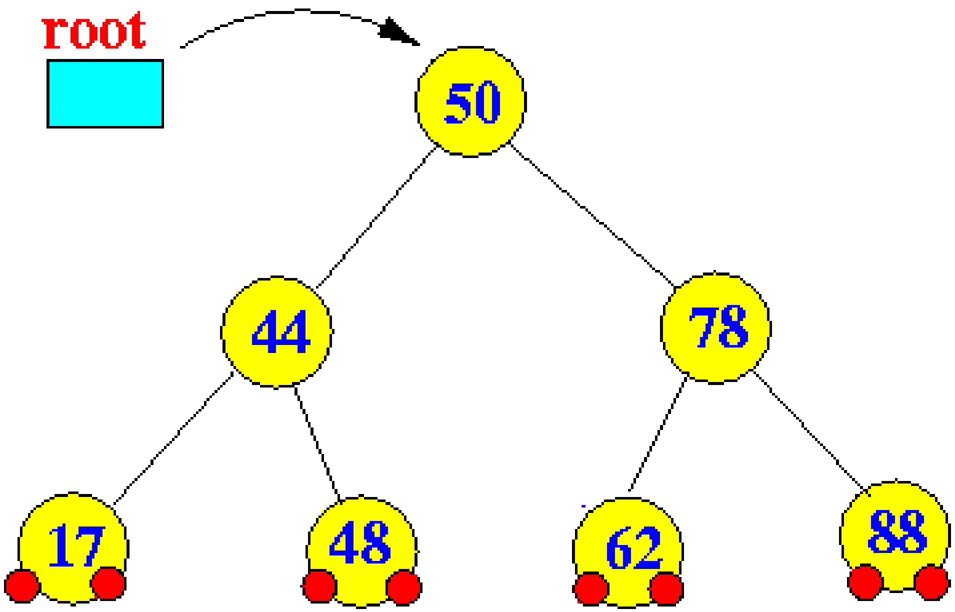
Re-balancing an AVL tree after deletion:

Perform a rotation using *these* 3 nodes (shaded):



Re-balancing an AVL tree after deletion:

The tree *after* the rotation is:

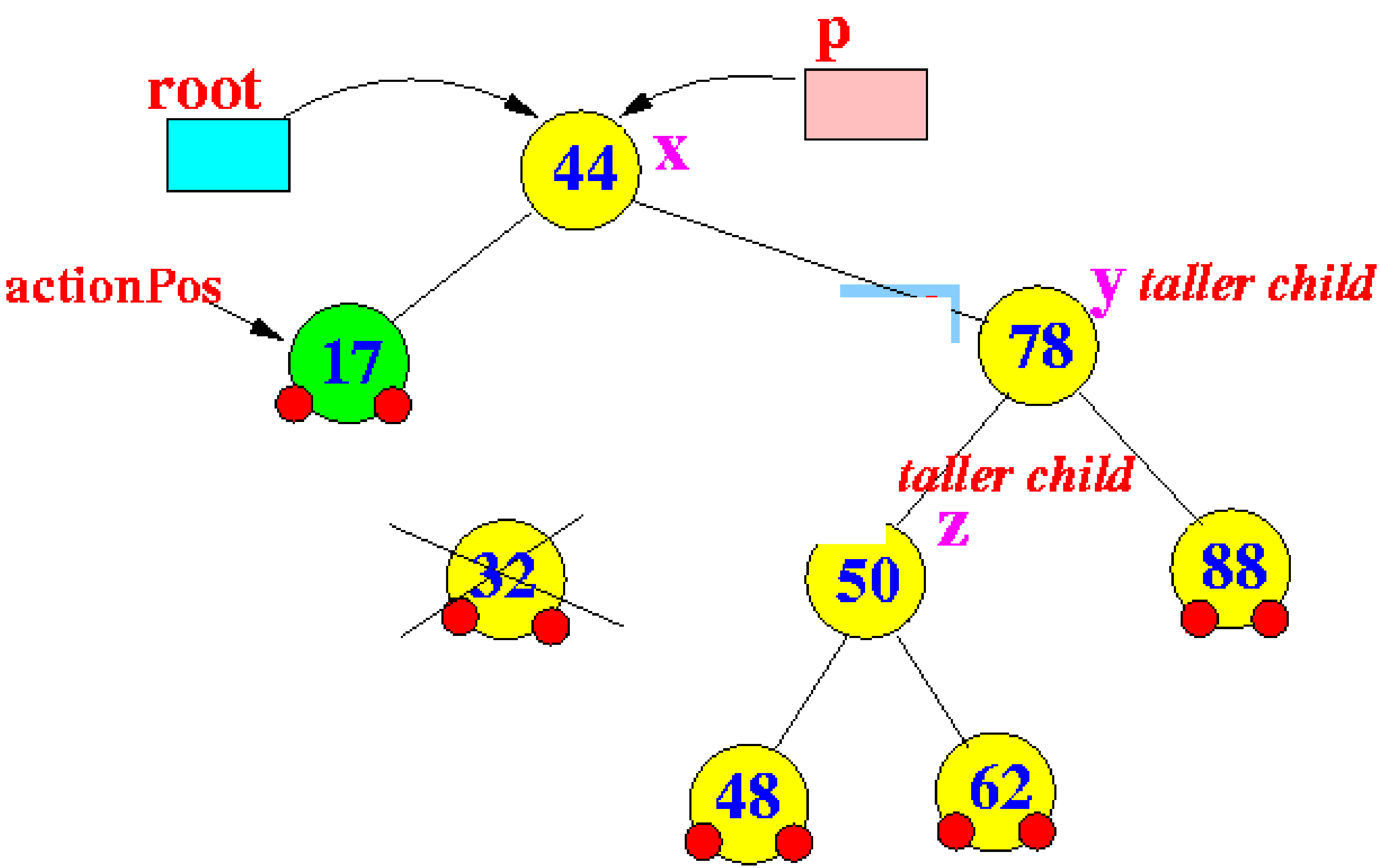


Pre-conditions for applying rotations to re-balance an AVL tree:

- **Node a or x** = the ***first*** **imbalanced node** from the **action position** (= **parent** of the ***physically*** **inserted/deleted node**) to the **root**.

- **Node b or y** = the **child node** of **node a** that has the ***higher*** **height**

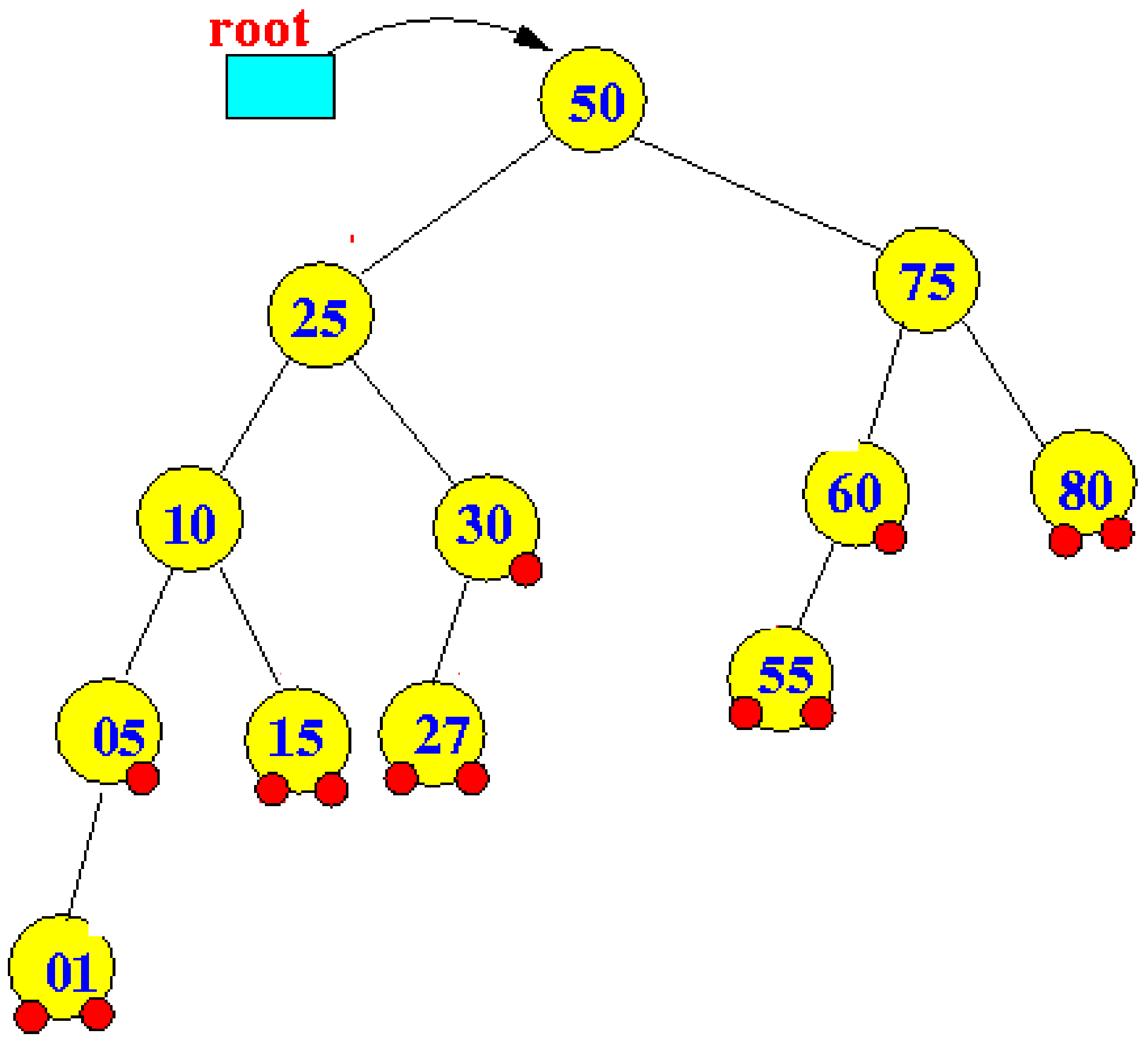
- **Node c or z** = the **child node** of **node b** that has the ***higher*** **height**

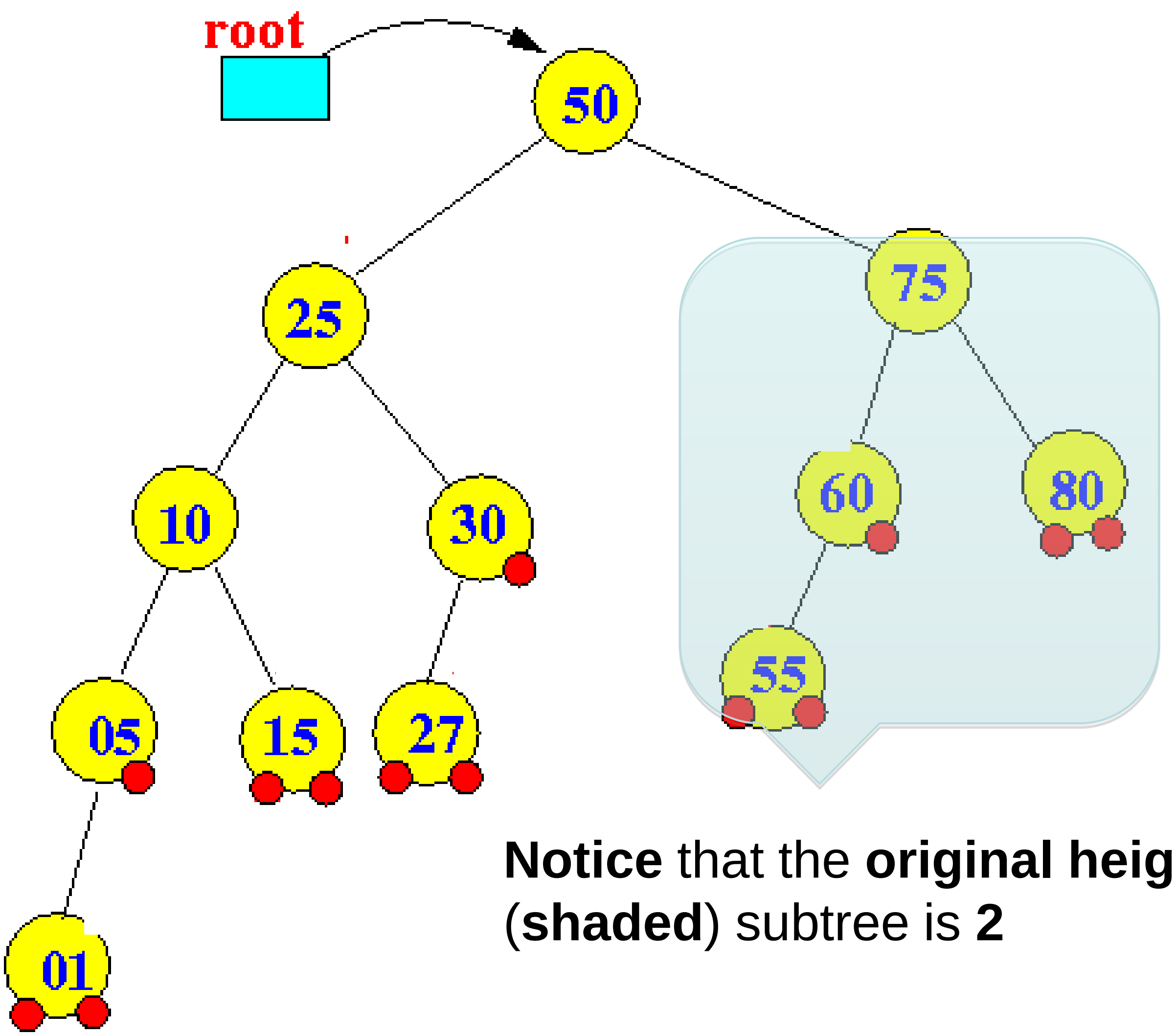

p
root
44
x
actionPos
17
78
y
taller child
taller child
32
50
z
88
48
62

Further re-balancing required for the *delete* operation

- We just saw that:
  - The **imbalance** at the ***first*** **imbalance node** due to a **deletion operation** ***can be restored*** *using rotation*
- **However**:
  - The ***resulting*** **subtree does** ***not*** have the ***same*** **height** as the ***original*** **subtree** !!!
- **Consequently**:
  - **Nodes** that are ***further*** **up the tree** are **re-balanced** by the **re-balancing of the** ***first*** **imbalanced node** !!!

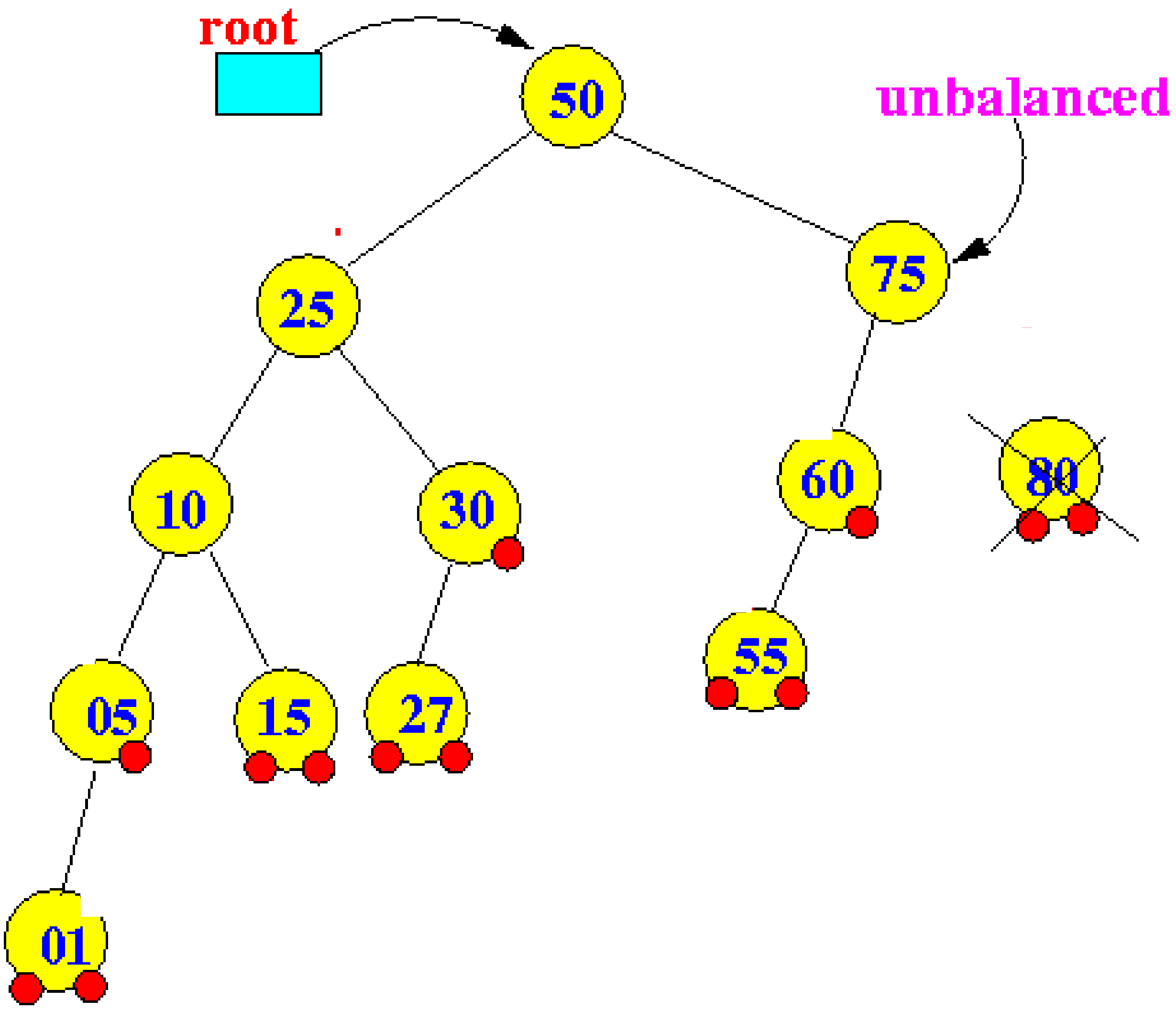
AVL Trees - Implementation



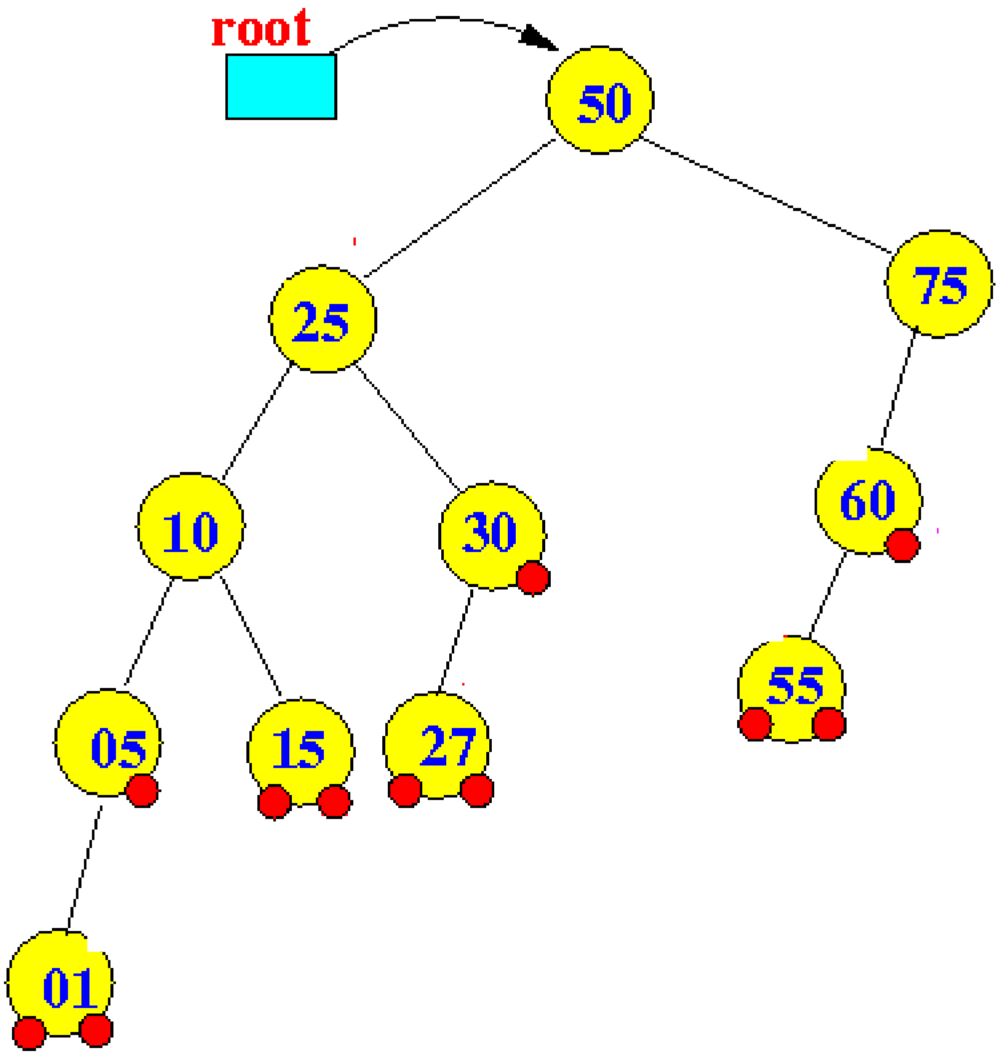


Notice that the **original height** of this (**shaded**) subtree is **2**

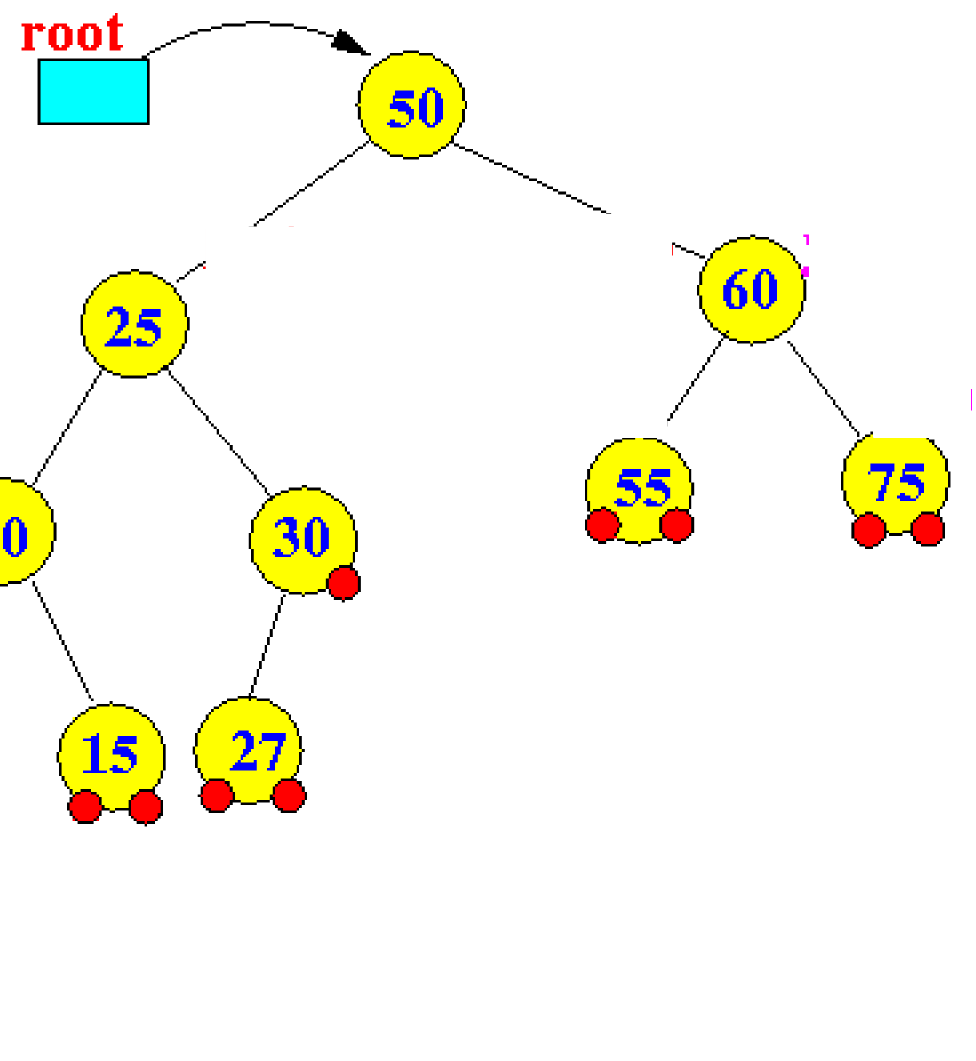
Delete the node 80

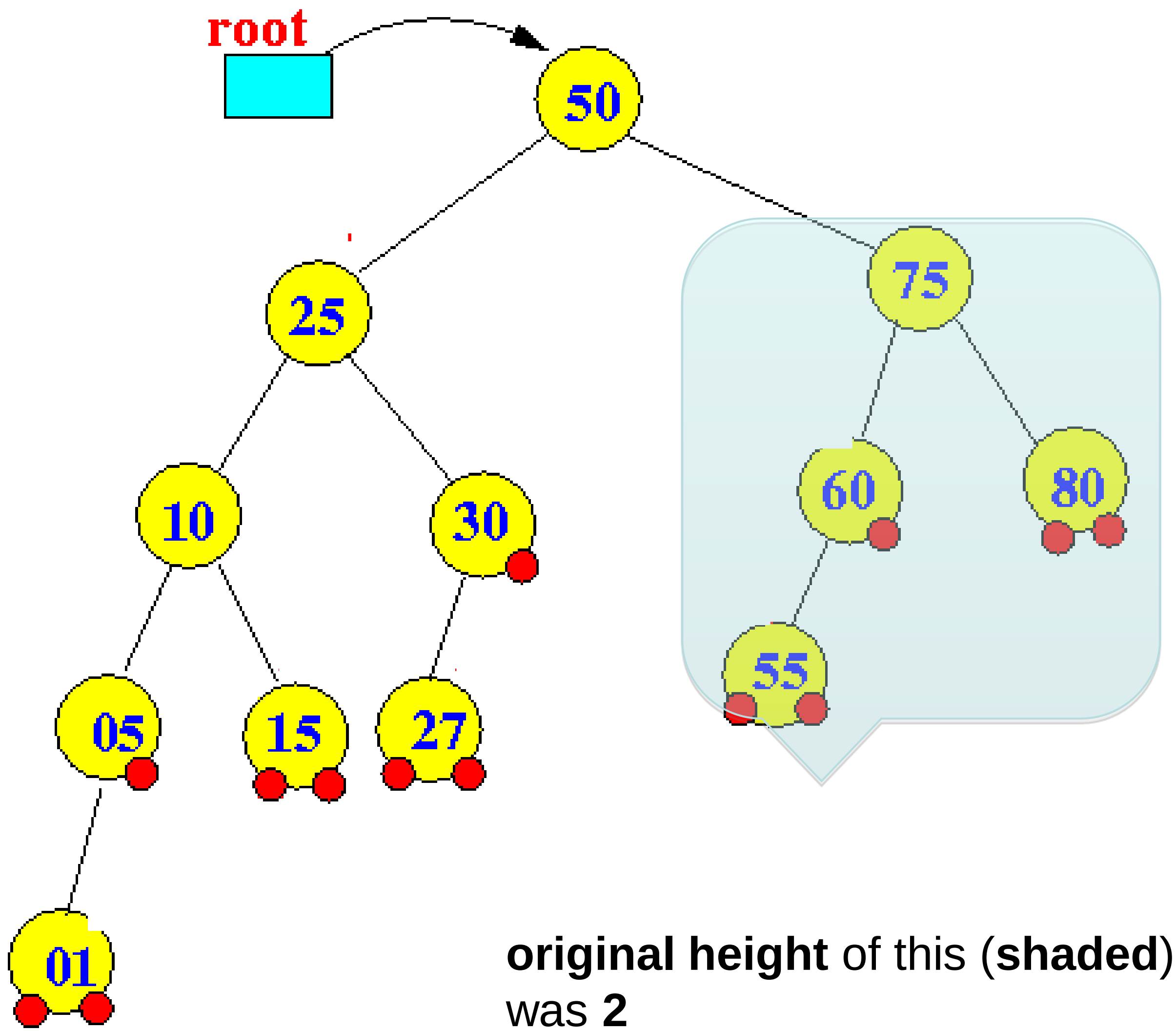


Delete the node 80



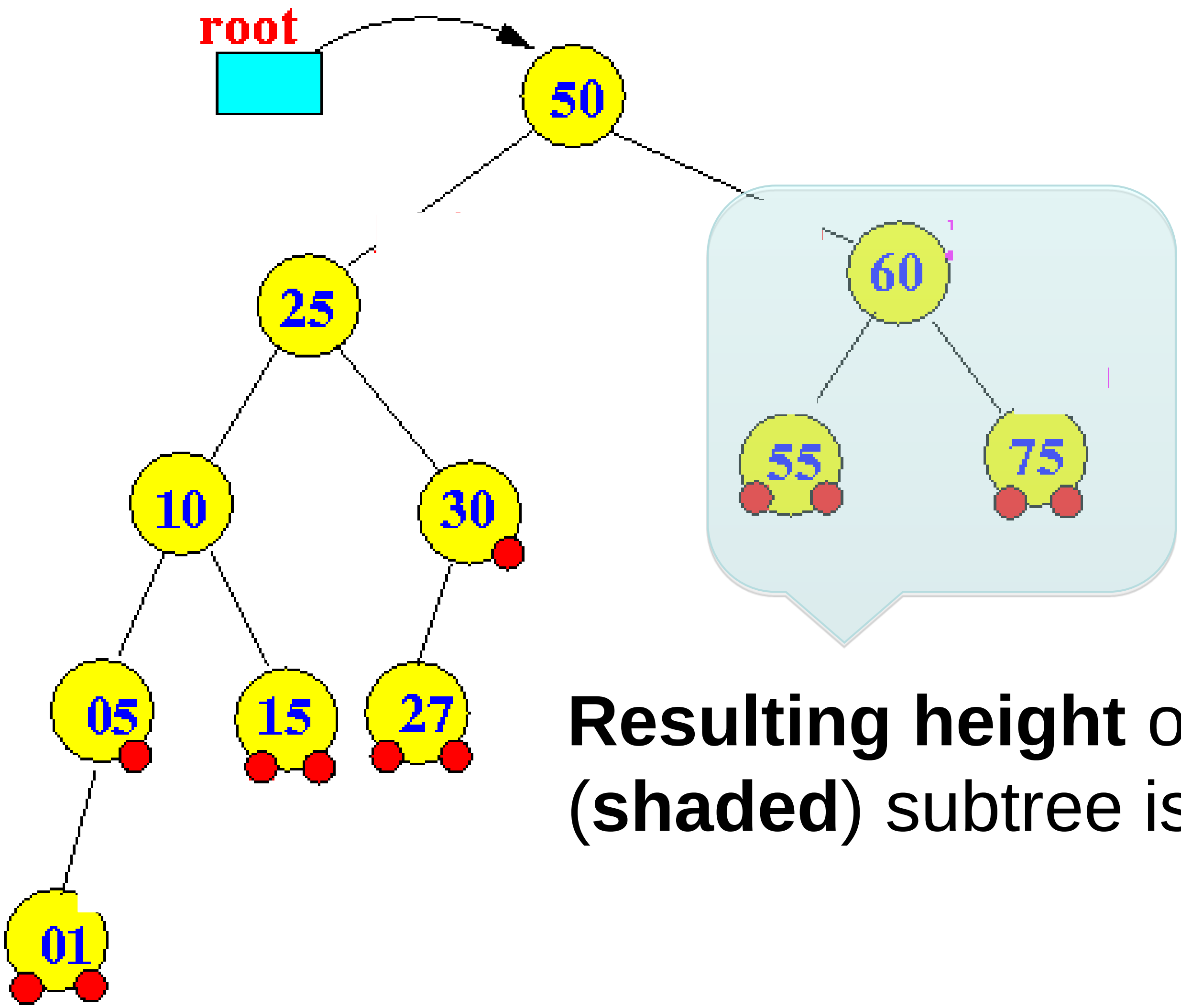
Delete the node 80





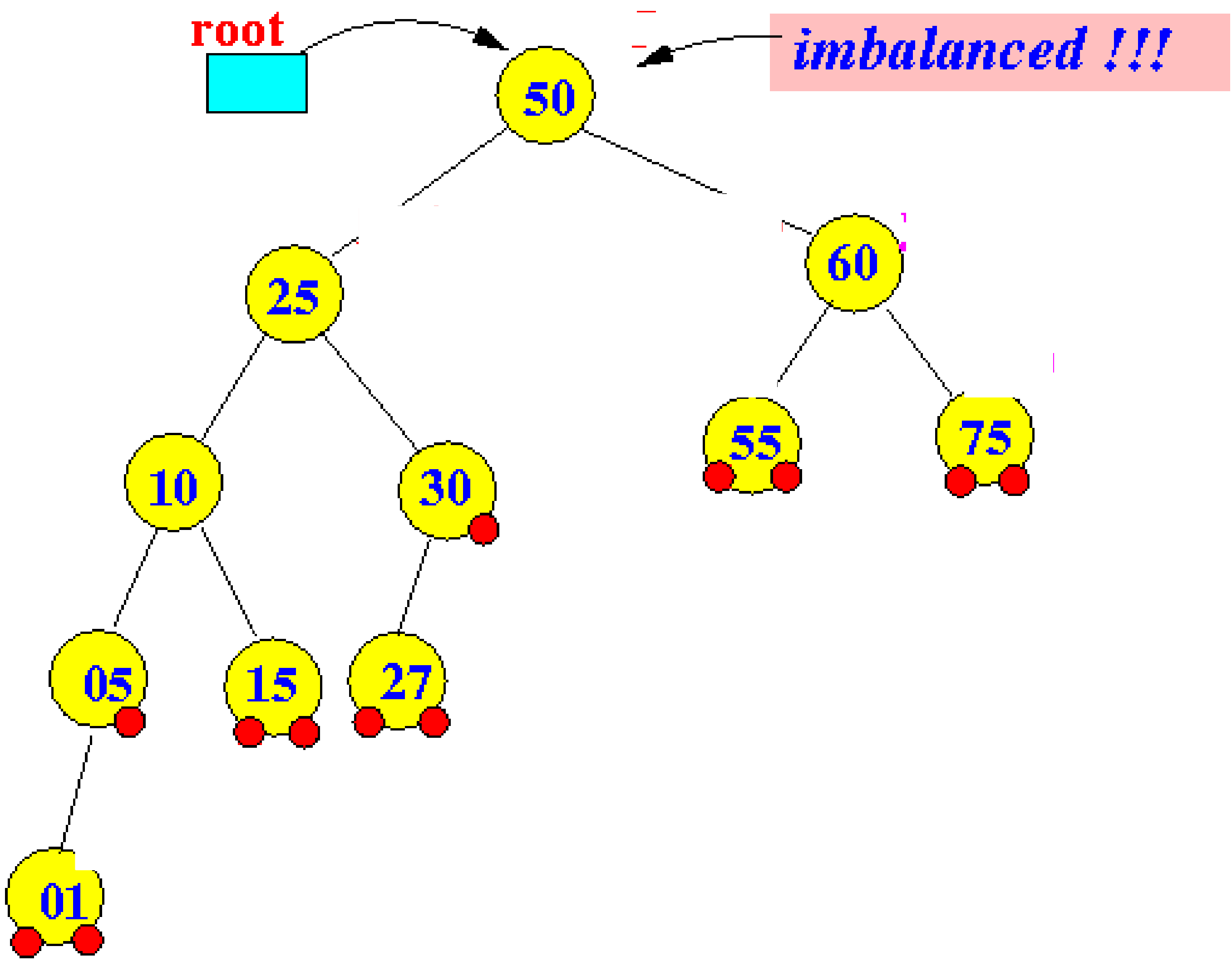
original height of this (**shaded**) subtree was **2**

Delete the node 80



Resulting height of this (**shaded**) subtree is **1**

Delete the node 80

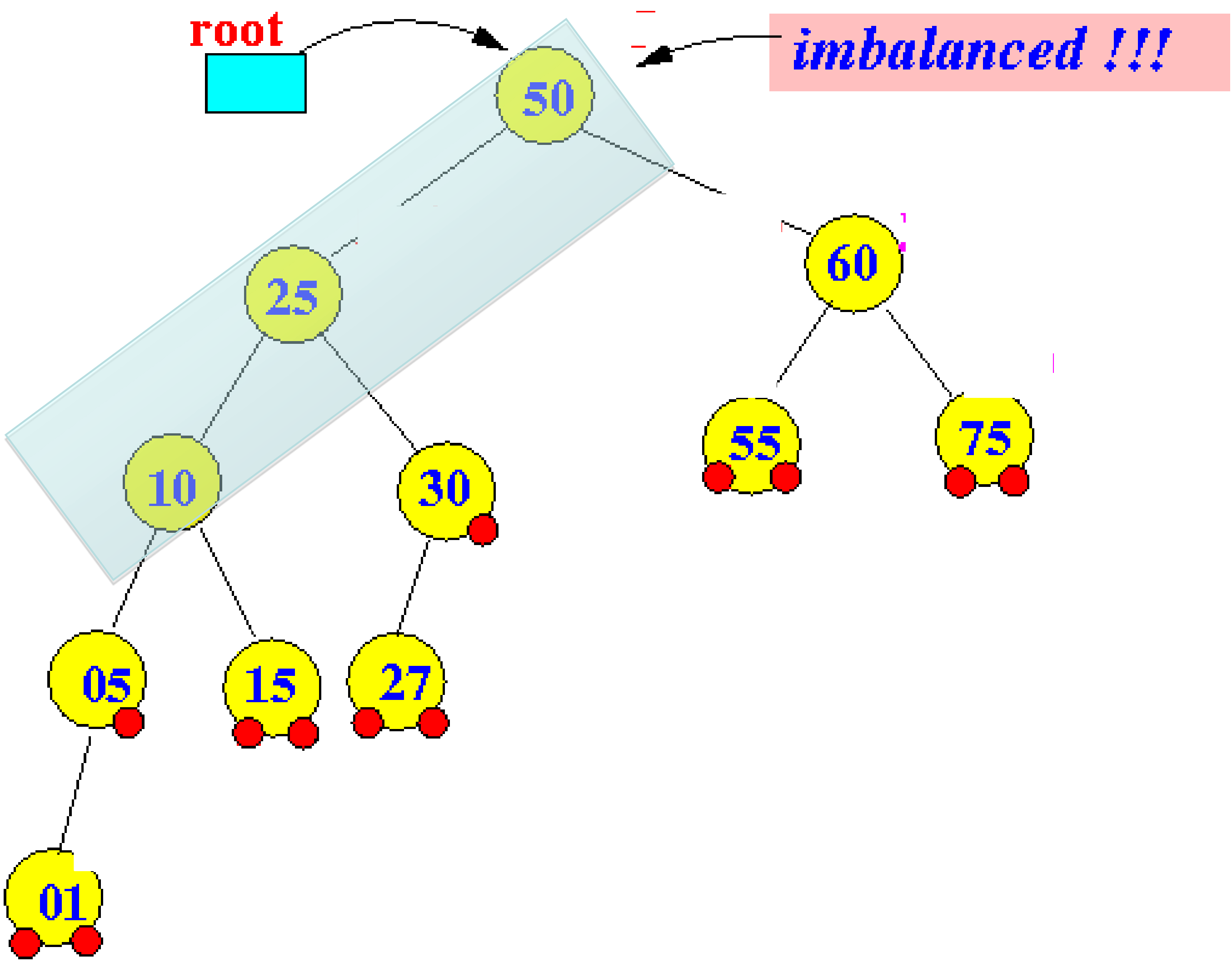


Node a or x = the ***first*** **imbalanced node** from the **action position** (= **parent** of the ***physically*** **inserted/deleted node**) to the **root**.

Node b or y = the **child node** of **node a** that has the ***higher*** **height**

Node c or z = the **child node** of **node b** that has the ***higher*** **height**

Delete the node 80



Homework

- Write C++ code for:
  - AVL Delete
  - Finding Min
  - Finding Max
  - Finding Height of a node if we do not have height member in structure definition
  - Finding Depth of a node
  - Checking if a given BST is AVL
  - Checking if a given binary tree is AVL

Questions?

"He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever"
Chinese Proverb

"The wise man doesn't give the right answers, he poses the right questions."
Claude Levi-Strauss

"A wise man can learn more from a foolish question than a fool can learn from a wise answer."
Bruce Lee