CSC 162
DATA STRUCTURES

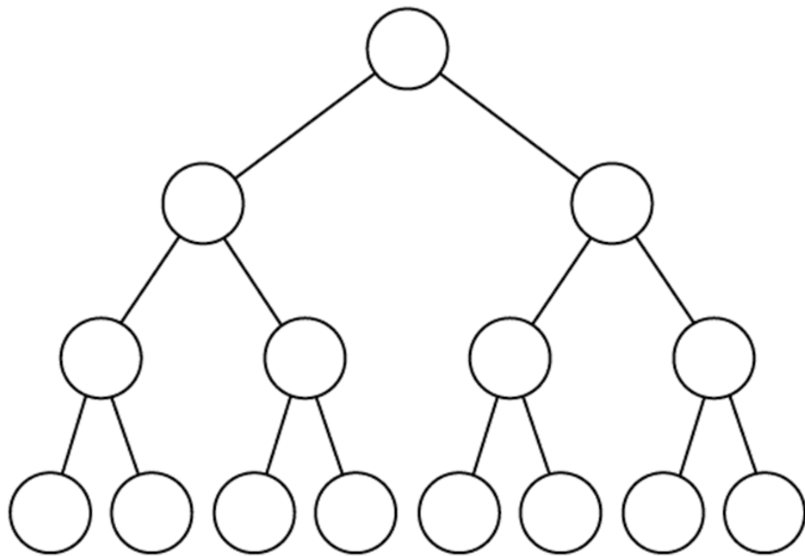
A PROBLEM WITH BSTs

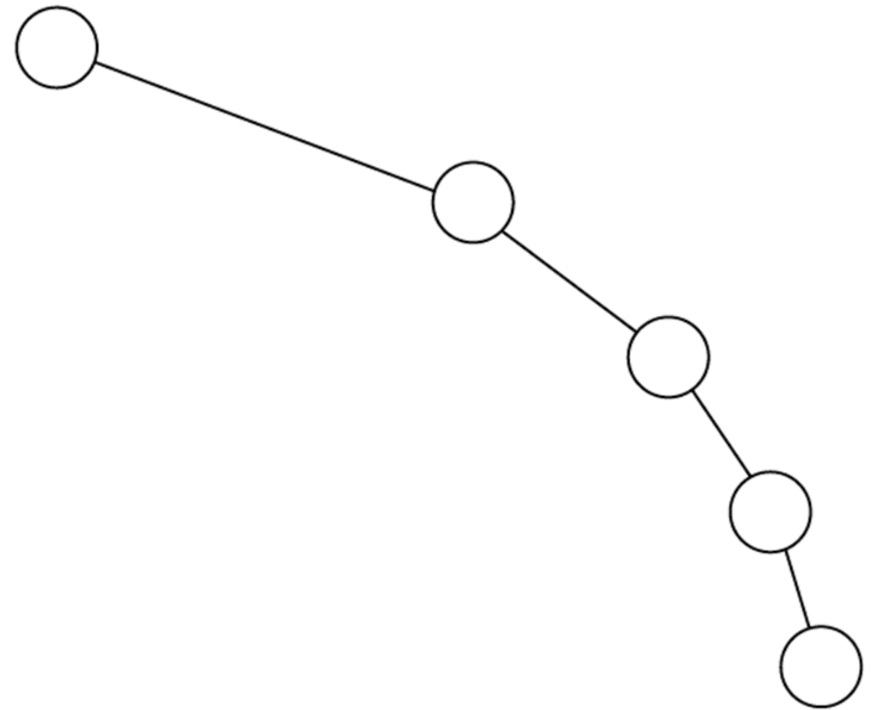
Common operations on *balanced* BST are O(log(n))

Alas, when the tree goes out of balance, performance degrades (worst case : chain O(n))

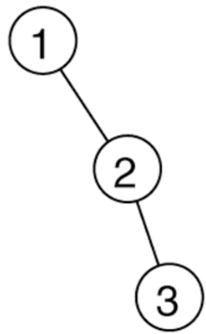
There are several data structures that modify the BST to maintain balance.

BST Structure : Log & Linear

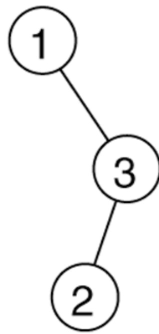




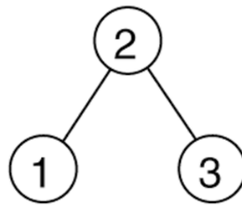
Permutations of 1,2,3



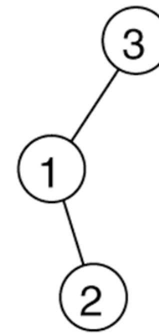
(a)



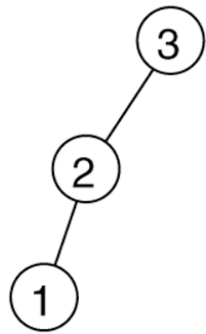
(b)



(c)



(d)



(e)

AVL Trees

The first balanced binary search tree

Named after discoverers

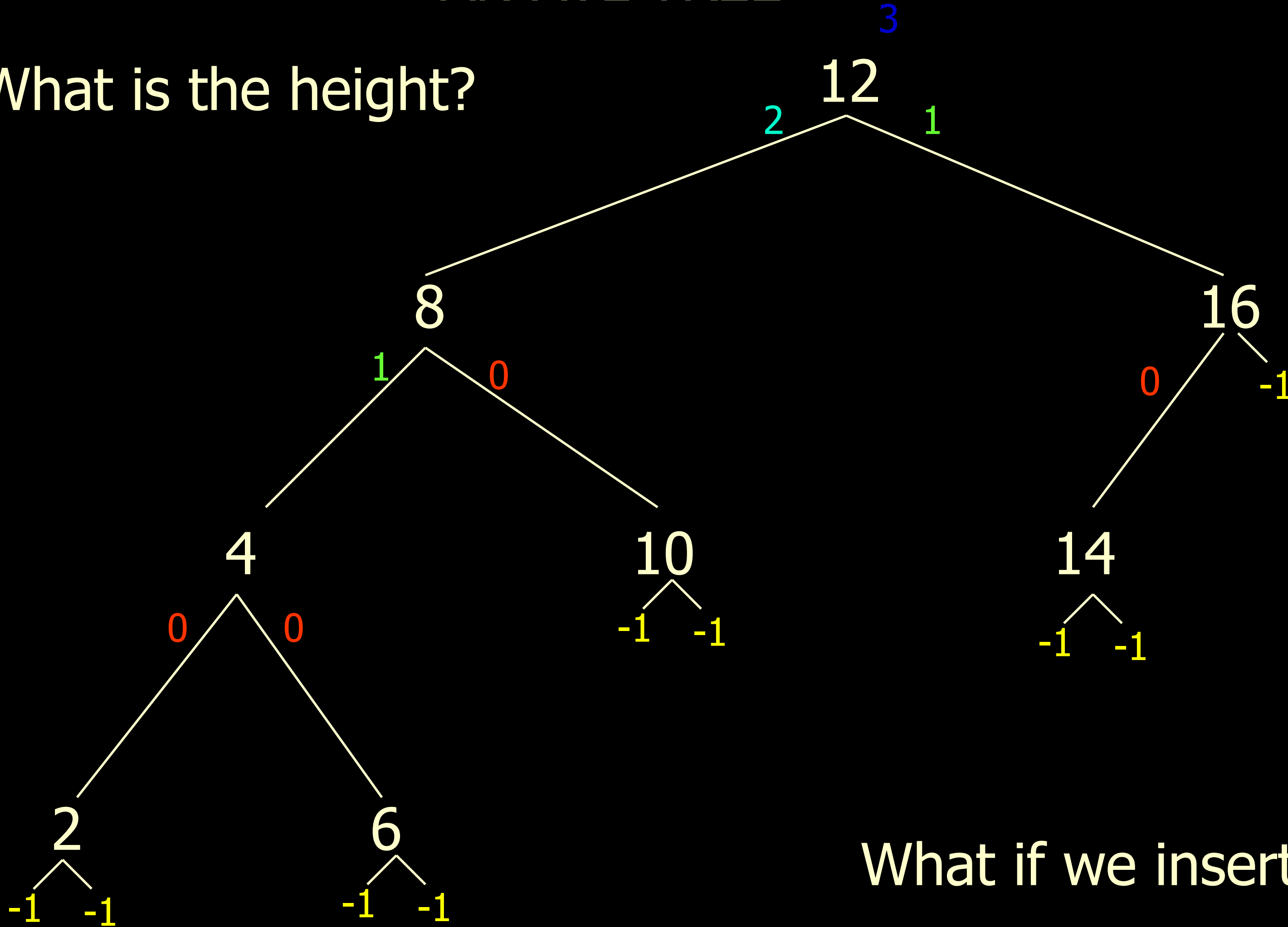
- Adelson-Velskii and Landis.

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. (Height of an empty subtree is –1).

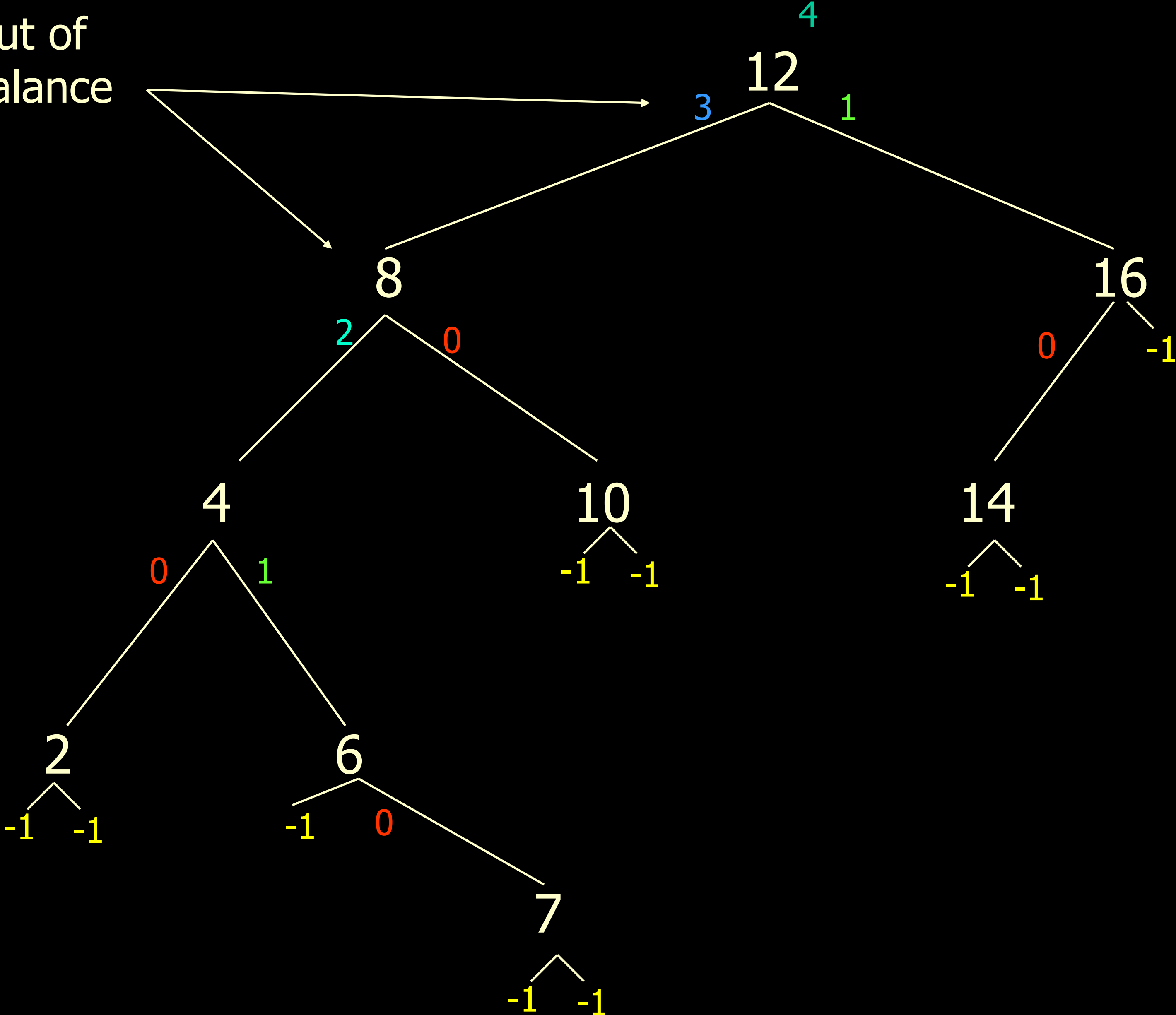
AN AVL TREE

What is the height?



What if we insert “7”?

NOT AN AVL TREE



AVL THEOREM

An AVL tree of n element has height O(log n)

In fact, An AVL tree of height H has at least $F_{H+3} - 1$ nodes, where F_i is the ith Fibonacci number

AVL THEOREM

Let S_H be the size of smallest AVL tree of height H.

Clearly $S_0=1$ and $S_1=2$

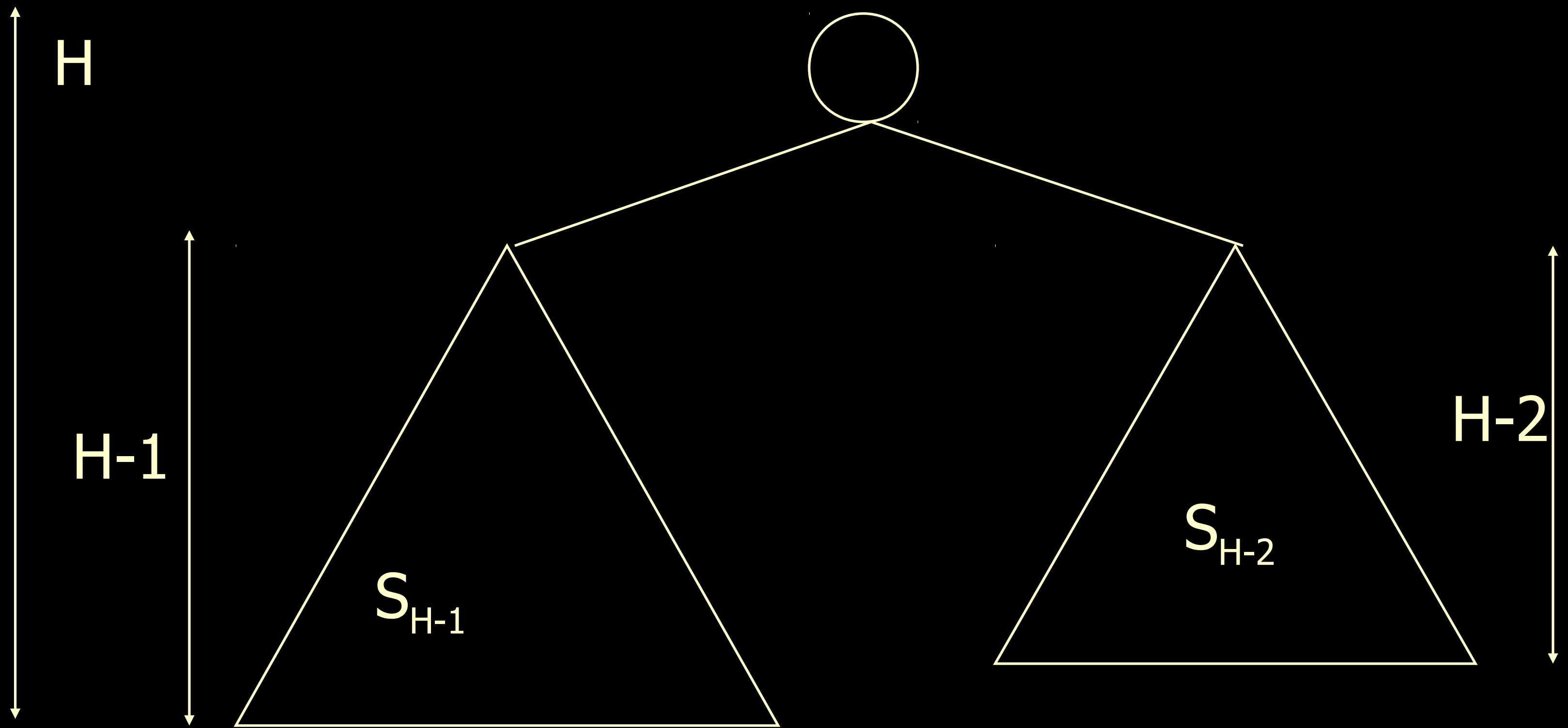
S_H must have subtrees of height H-1 and H-2

These subtrees must have fewest nodes for height

So, $S_H = S_{H-1} + S_{H-2} + 1$

Minimum number AVL trees are *Fibonacci trees*

Smallest AVL Tree of height H



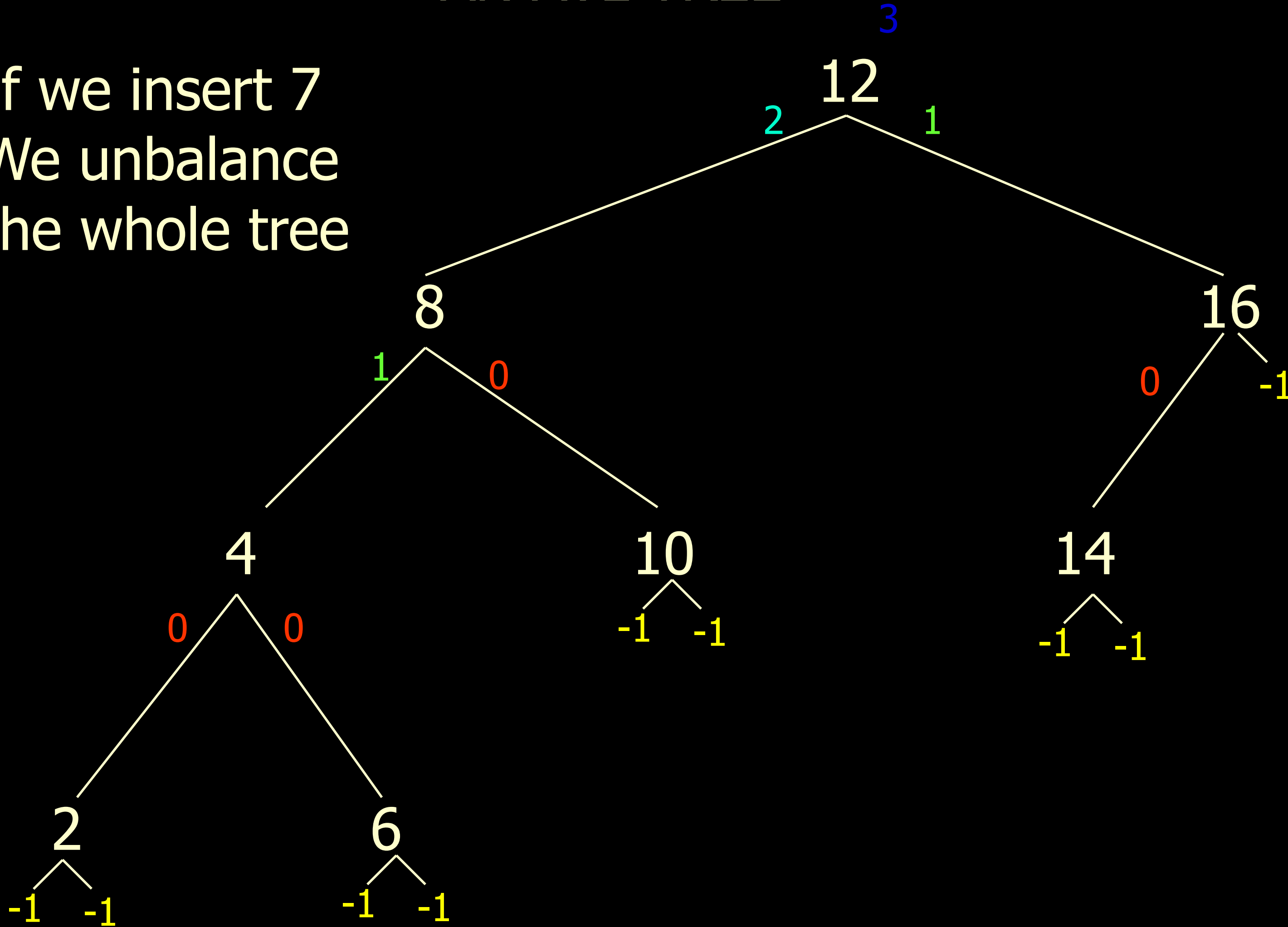
Adds can unbalance a BST

Only nodes on the path from the root to the insertion point can have their balances altered

If we restore the unbalance node, we balance the tree

AN AVL TREE

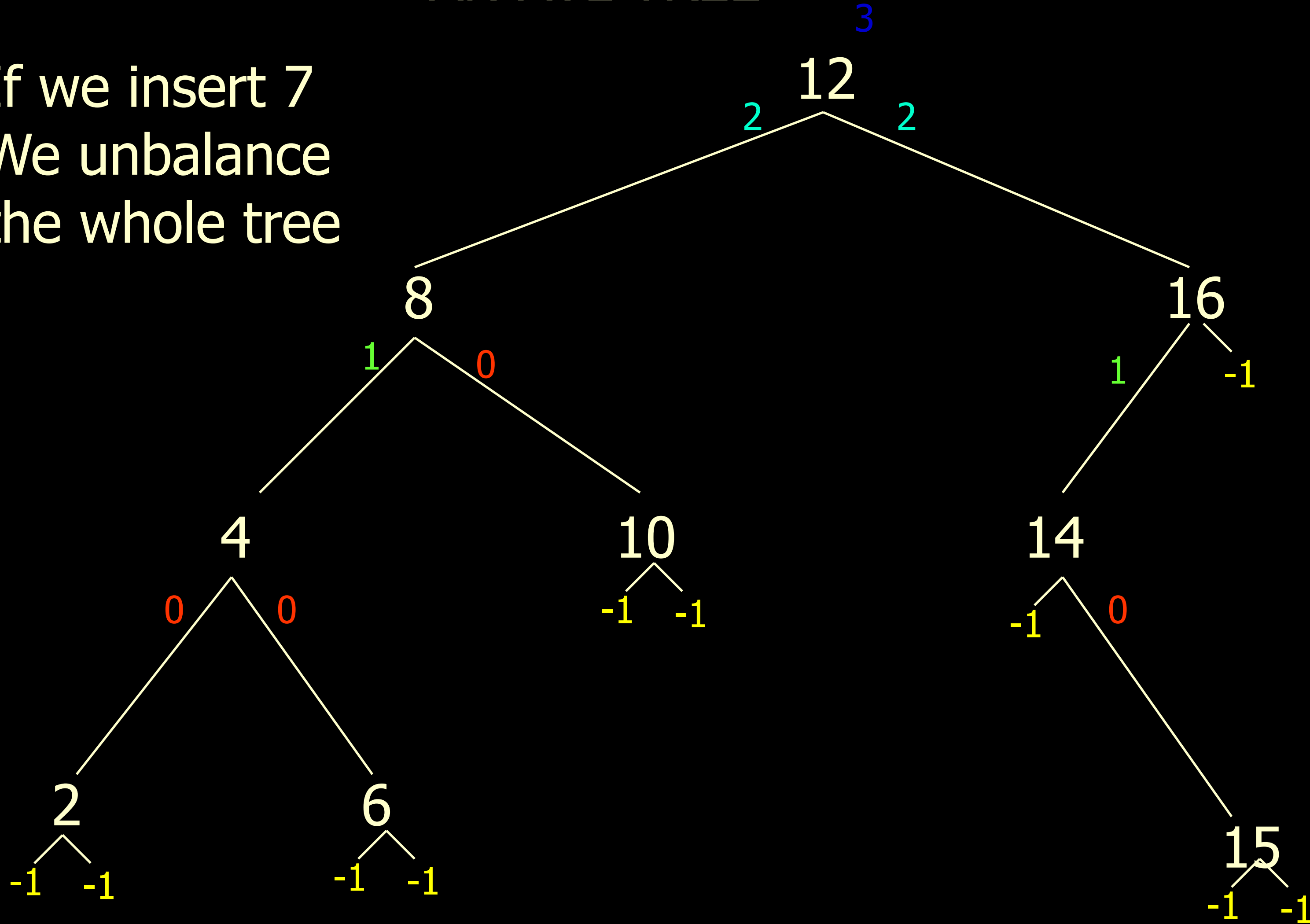
If we insert 7
We unbalance
the whole tree



What if we insert “15”?

AN AVL TREE

If we insert 7
We unbalance
the whole tree



What if we insert “15”?

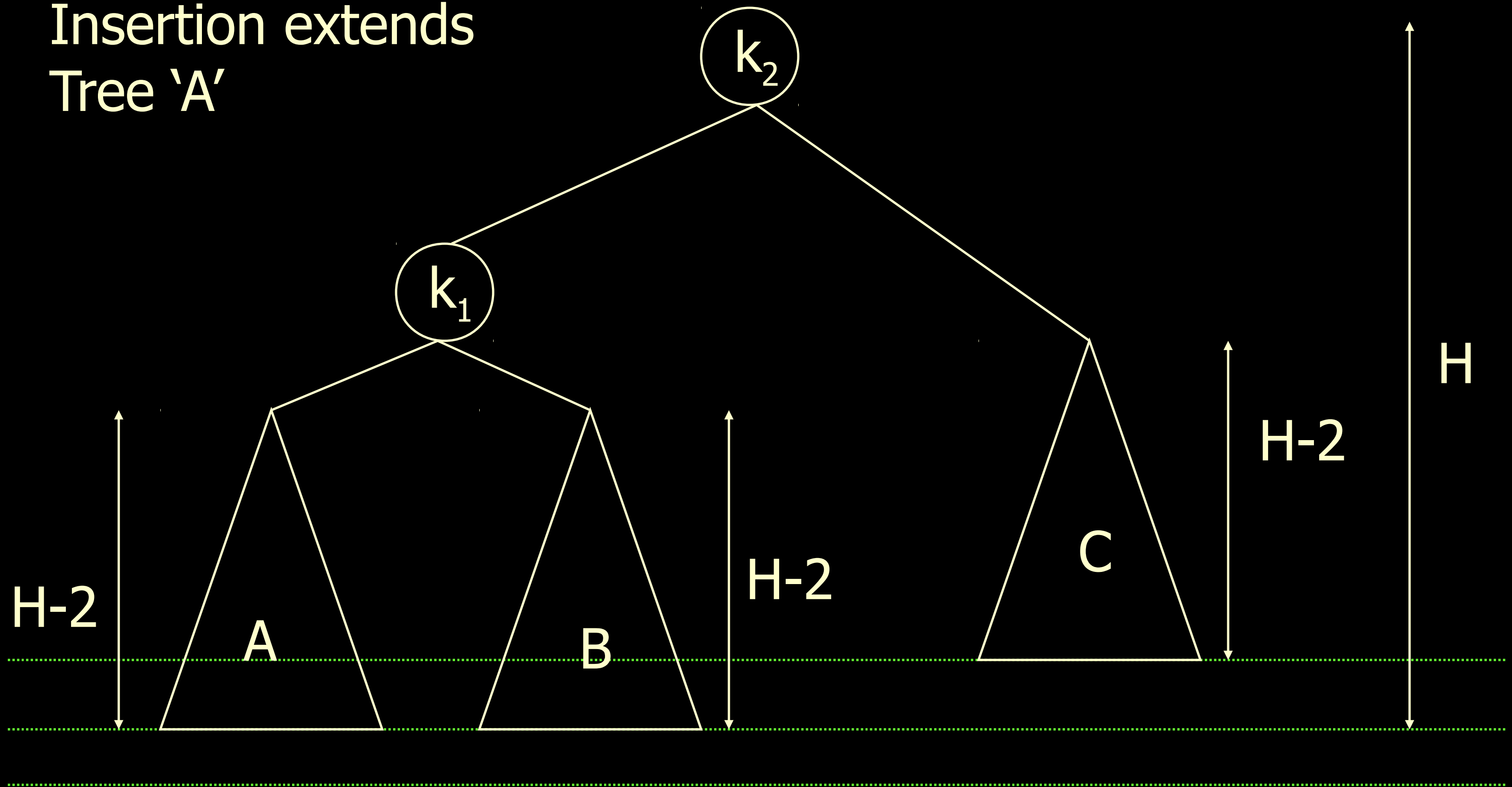
4 cases

1. Insertion in left sub-tree of left child
2. Insertion in right sub-tree of left child
3. Insertion in left sub-tree of right child
4. Insertion in right sub-tree of right child

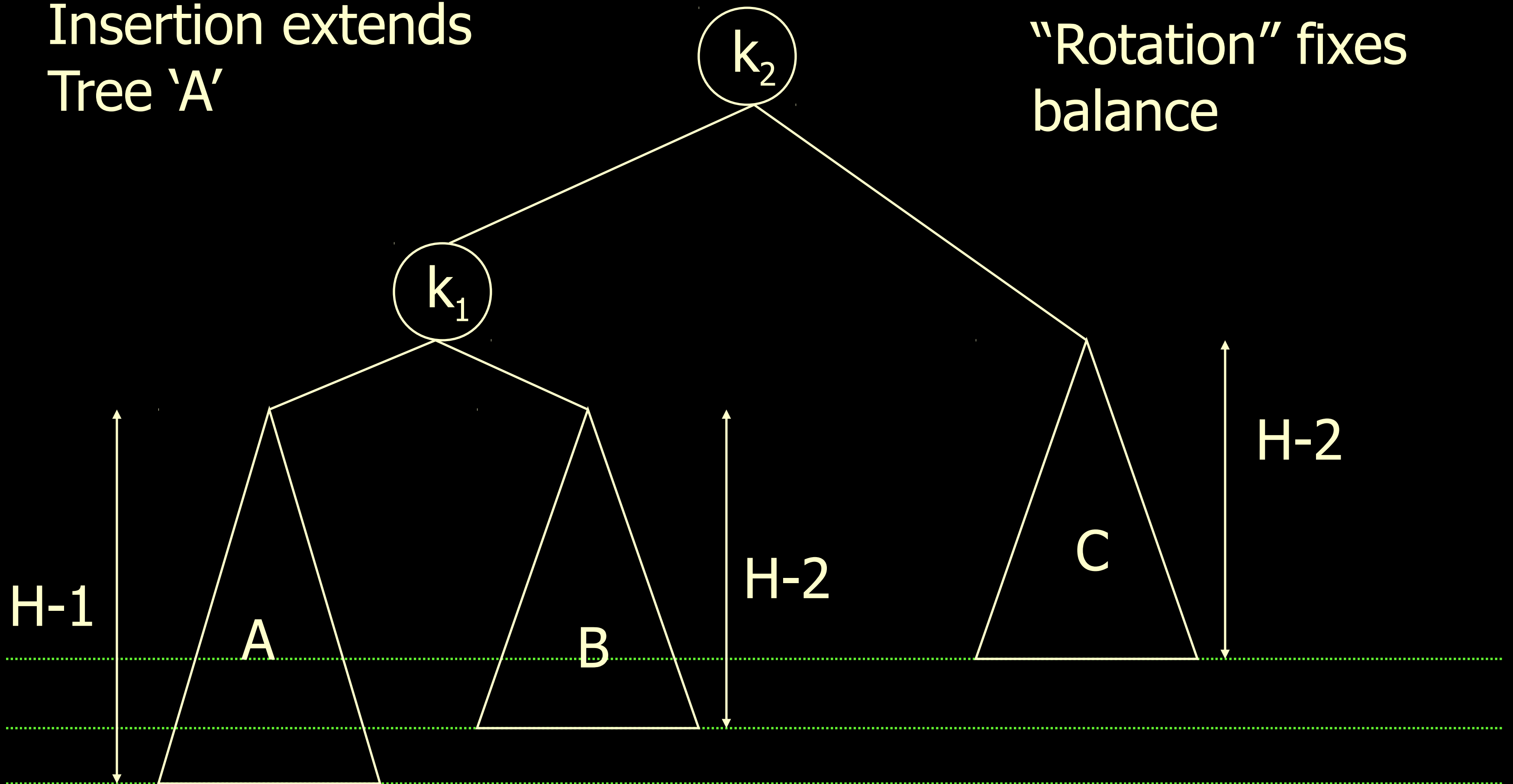
1 & 4 are symmetric

2 & 3 are symmetric

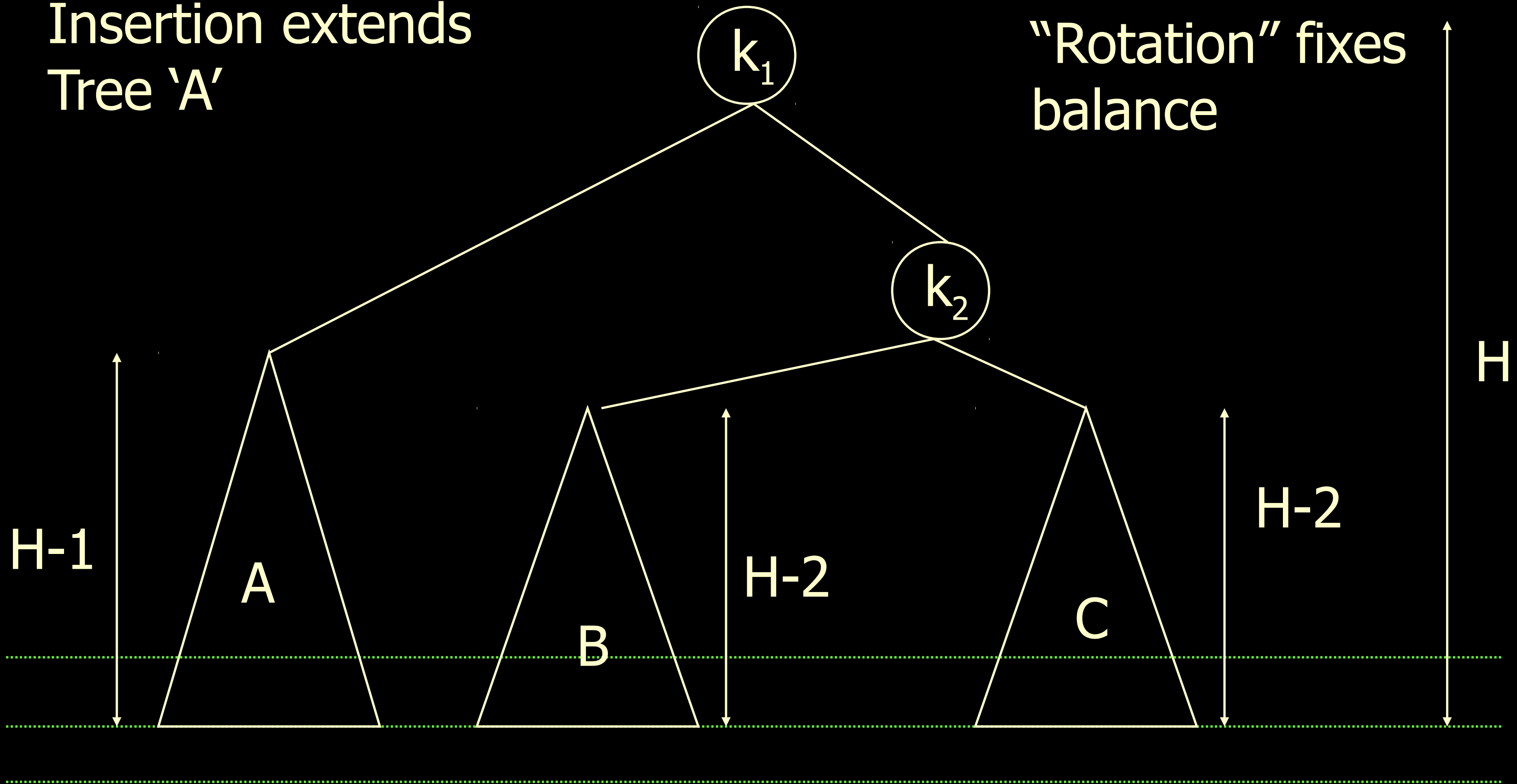
Cases 1 & 4



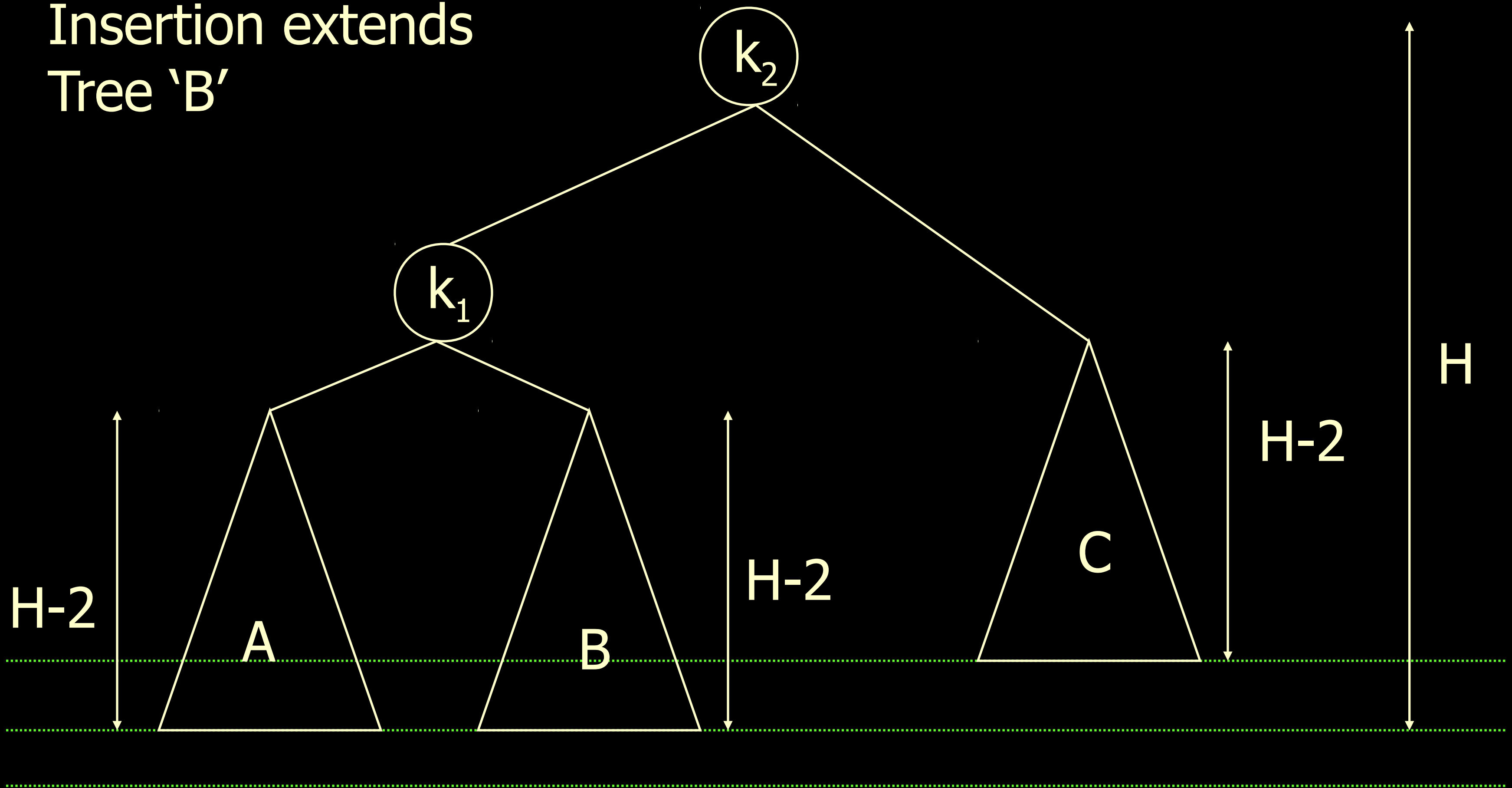
Cases 1 & 4



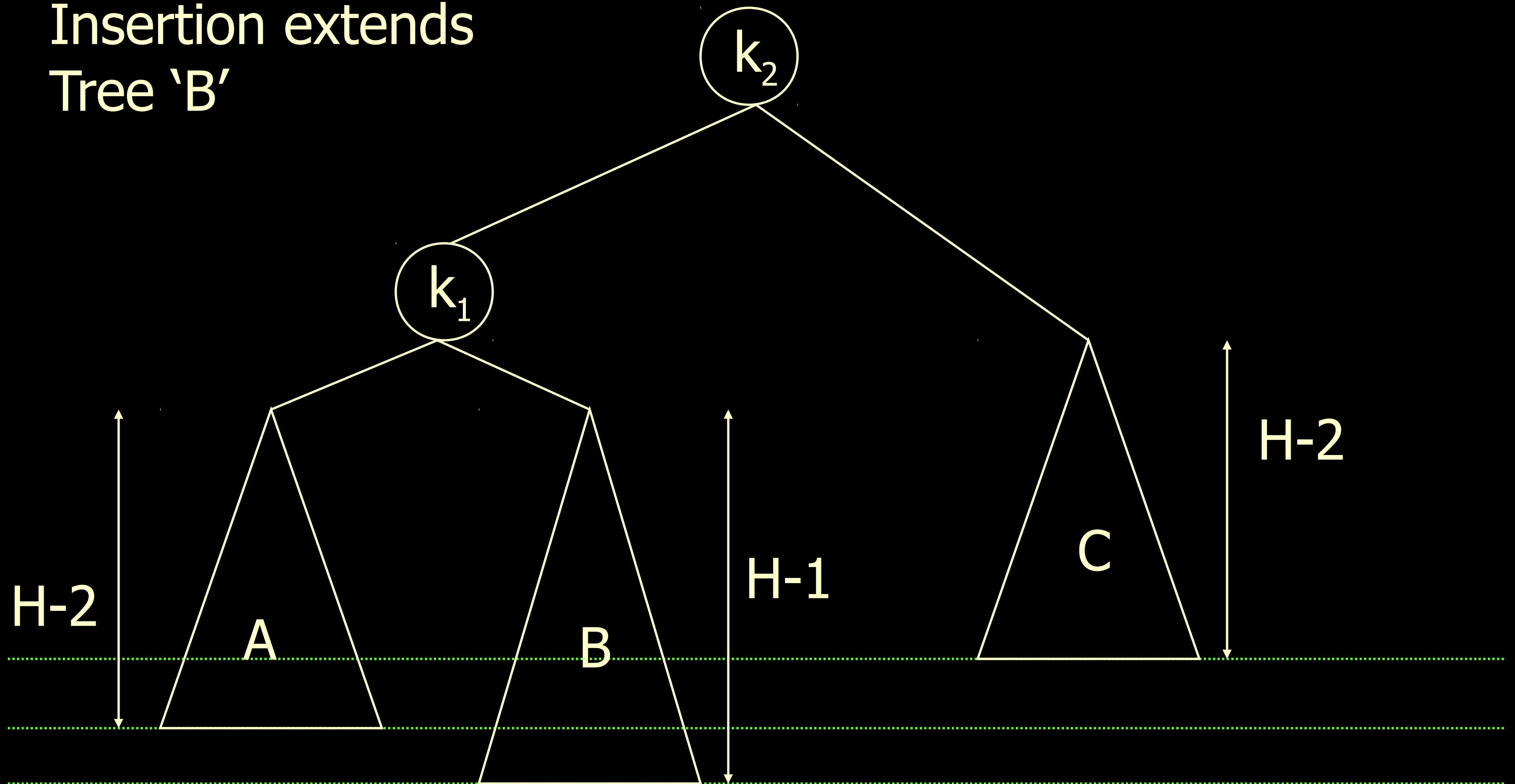
Cases 1 & 4



Cases 2 & 3



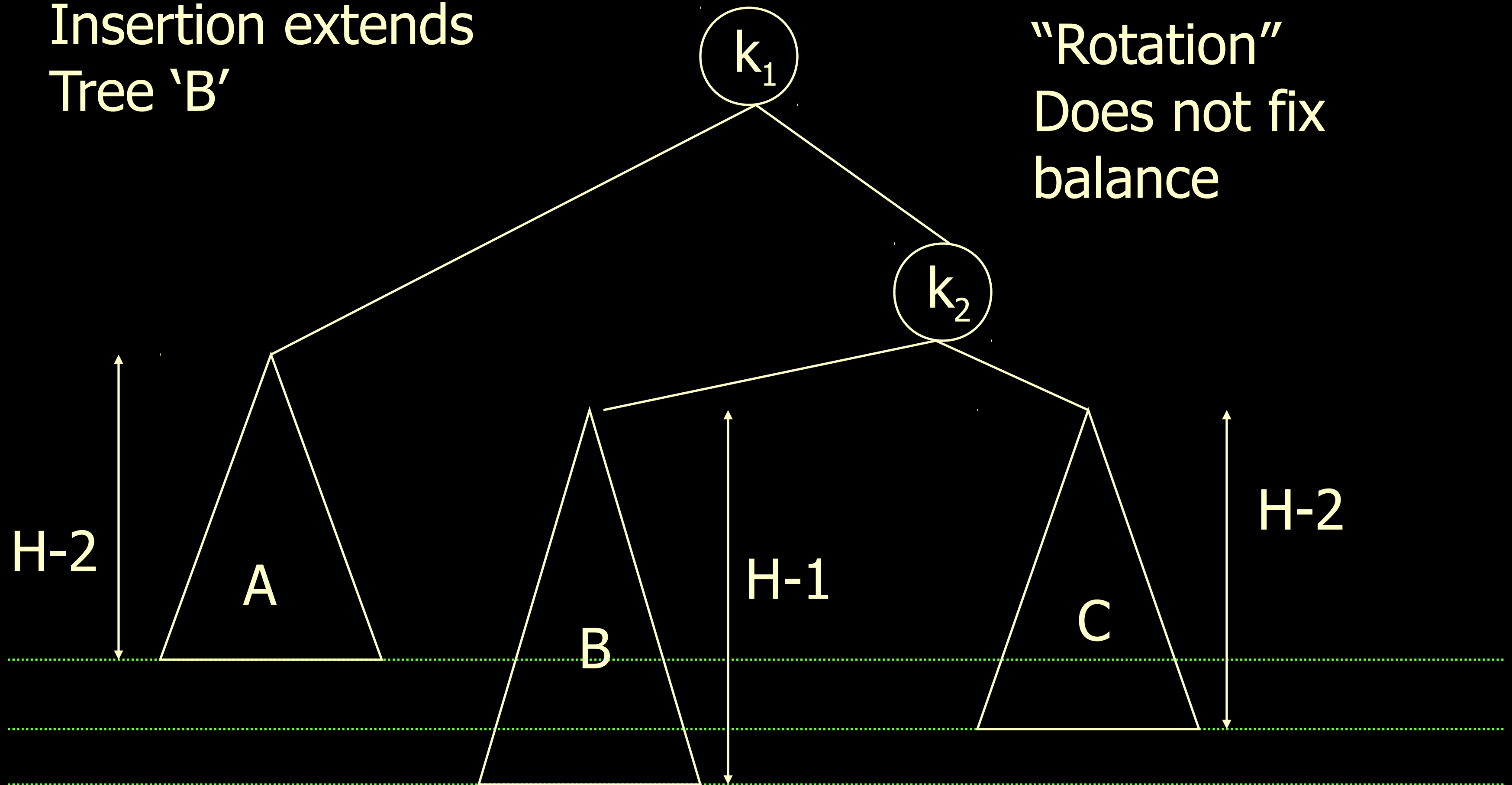
Cases 2 & 3



Cases 2 & 3

Insertion extends Tree 'B'

"Rotation" Does not fix balance

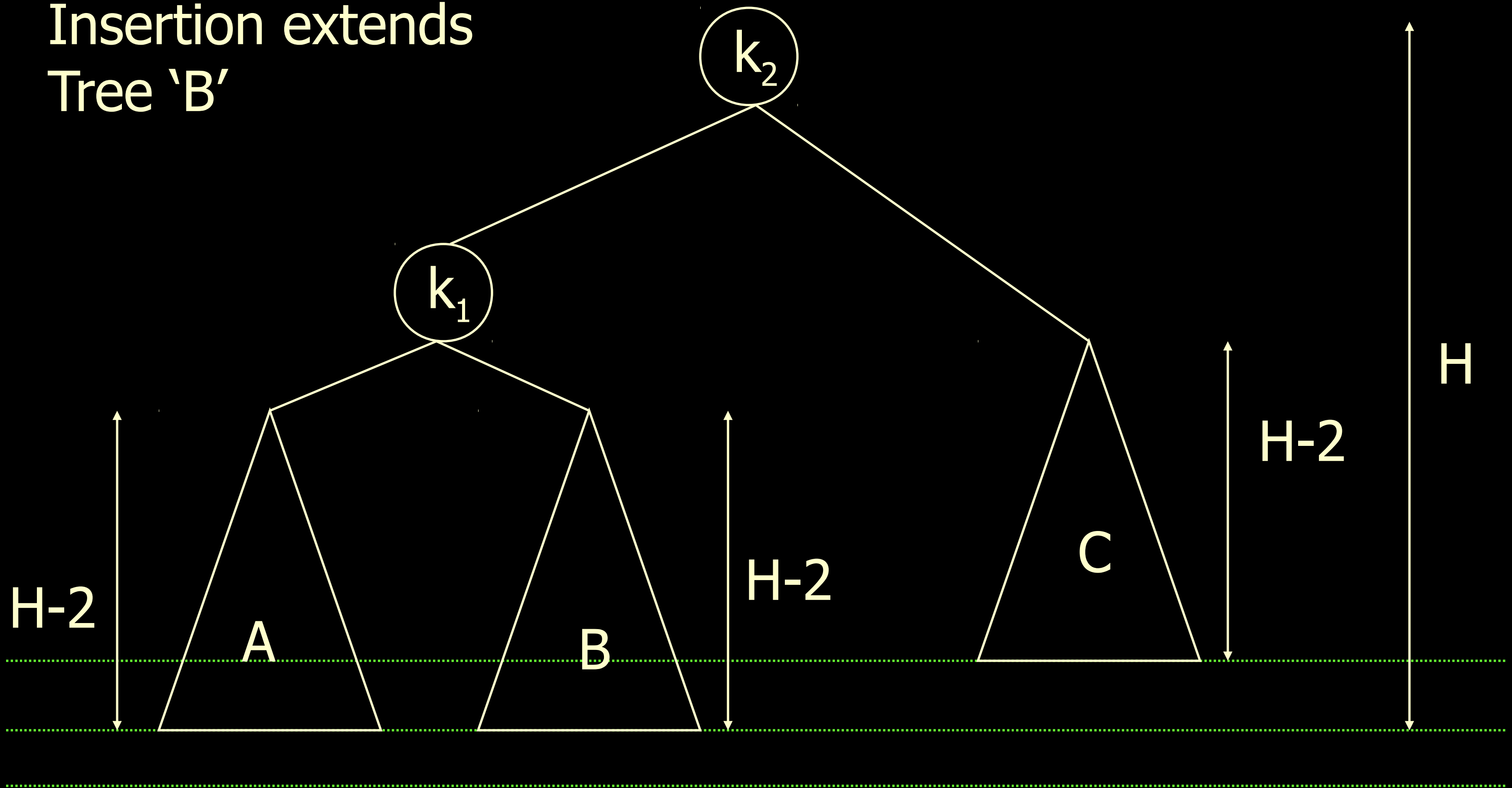


Double Rotation

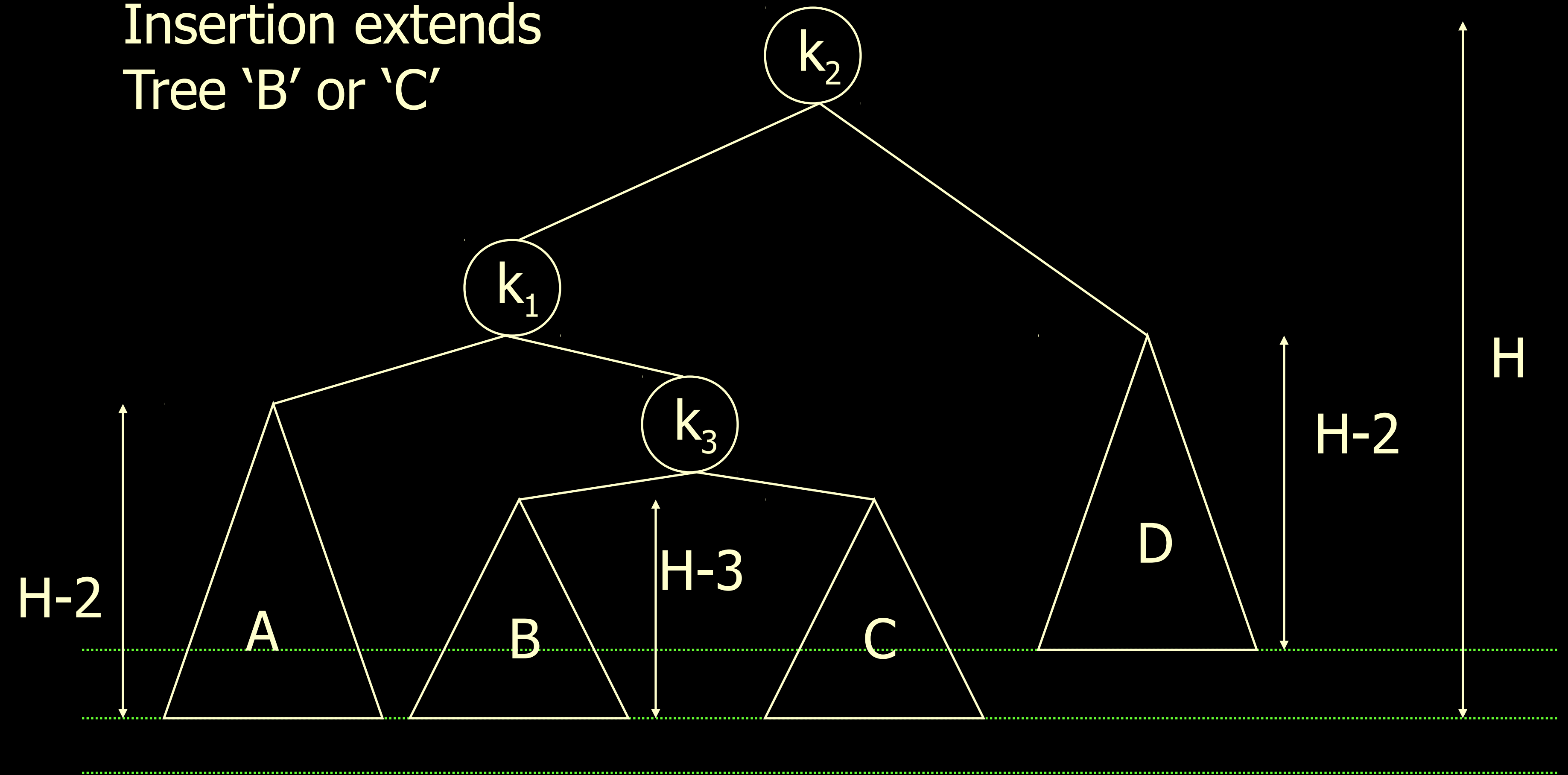
If x is out of balance for cases 2&3

1. Rotate between X's child and grandchild
2. Rotate between X and it's new child

Cases 2 & 3

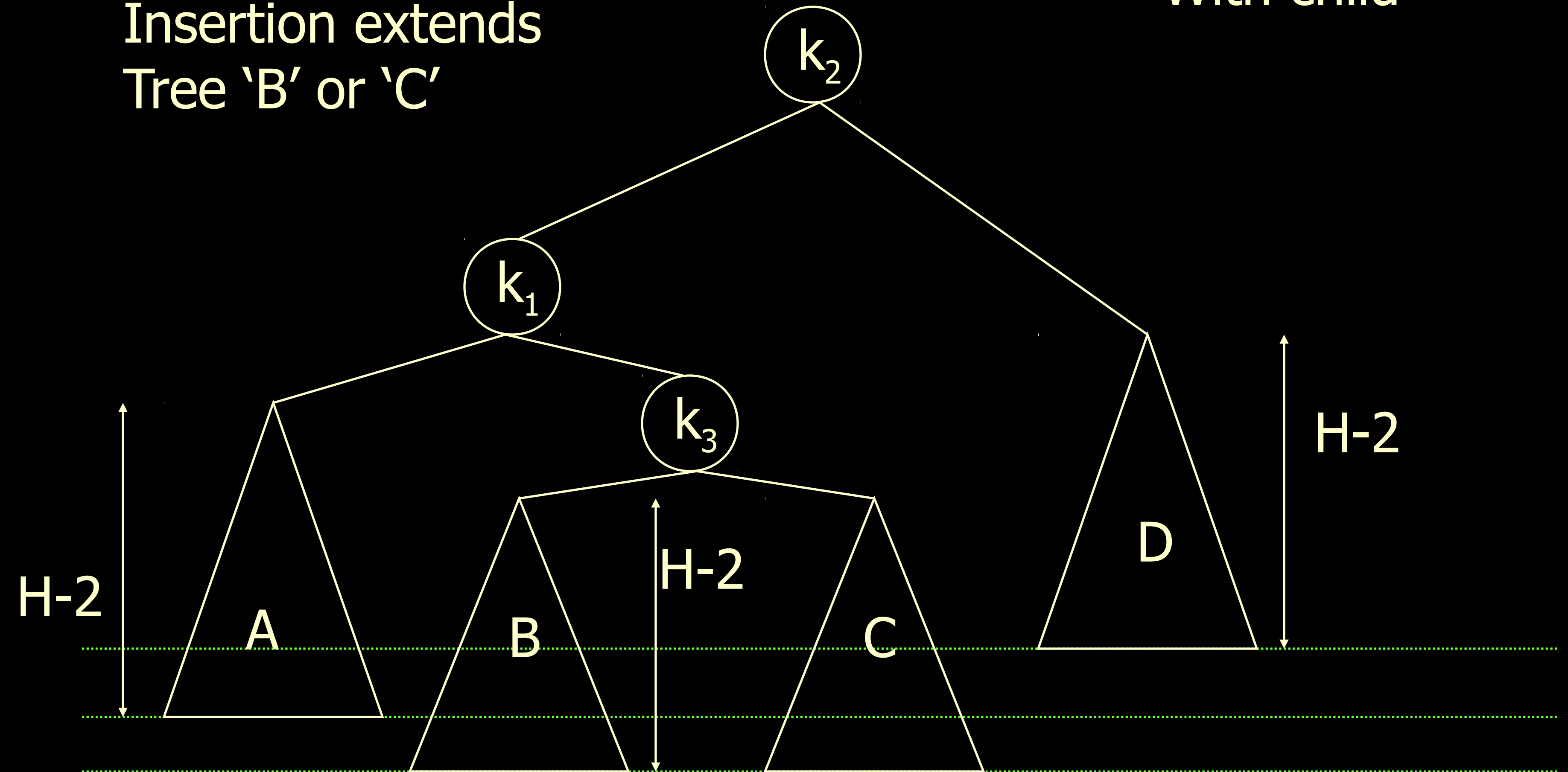


Cases 2 & 3



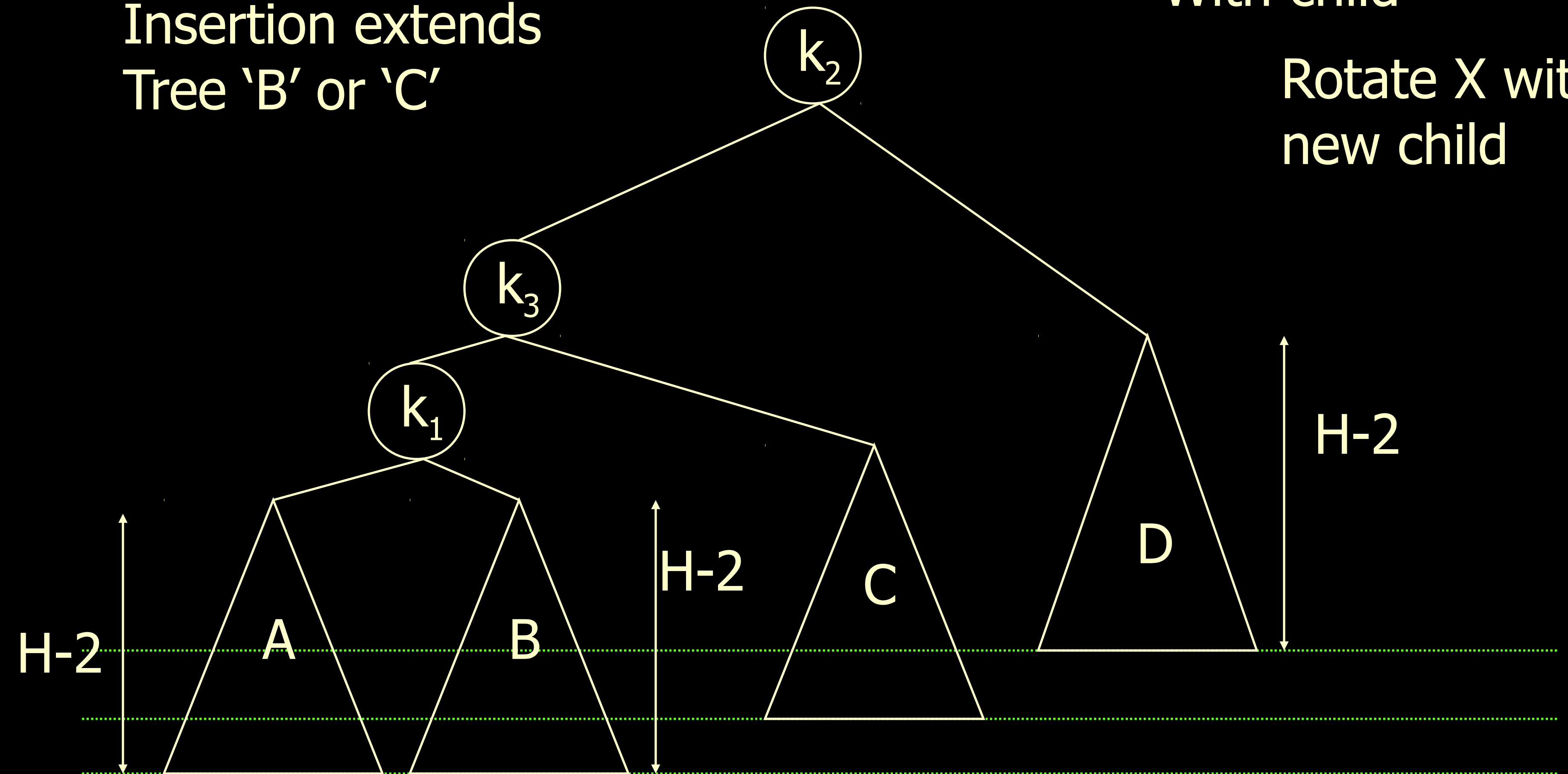
Cases 2 & 3

Rotate grandchild
With child



Cases 2 & 3

Rotate grandchild
With child

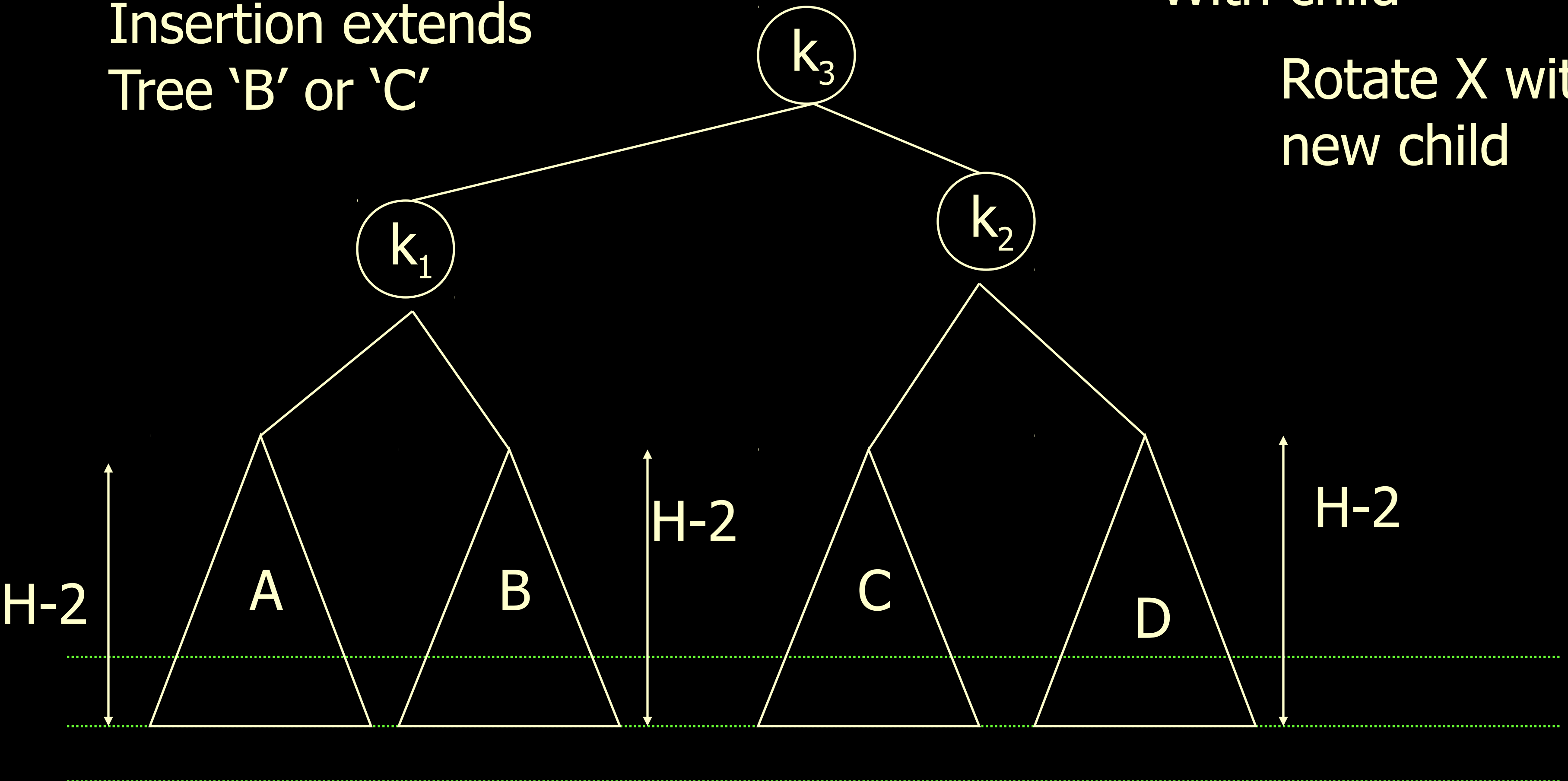


Cases 2 & 3

Rotate grandchild
With child

Rotate X with
new child

Insertion extends
Tree 'B' or 'C'



Implementation
Do this in lab

Insert

Fixup

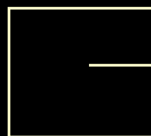
Rotate

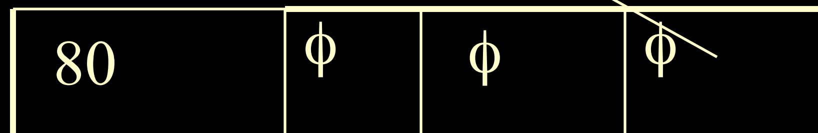
Need to keep track of balance

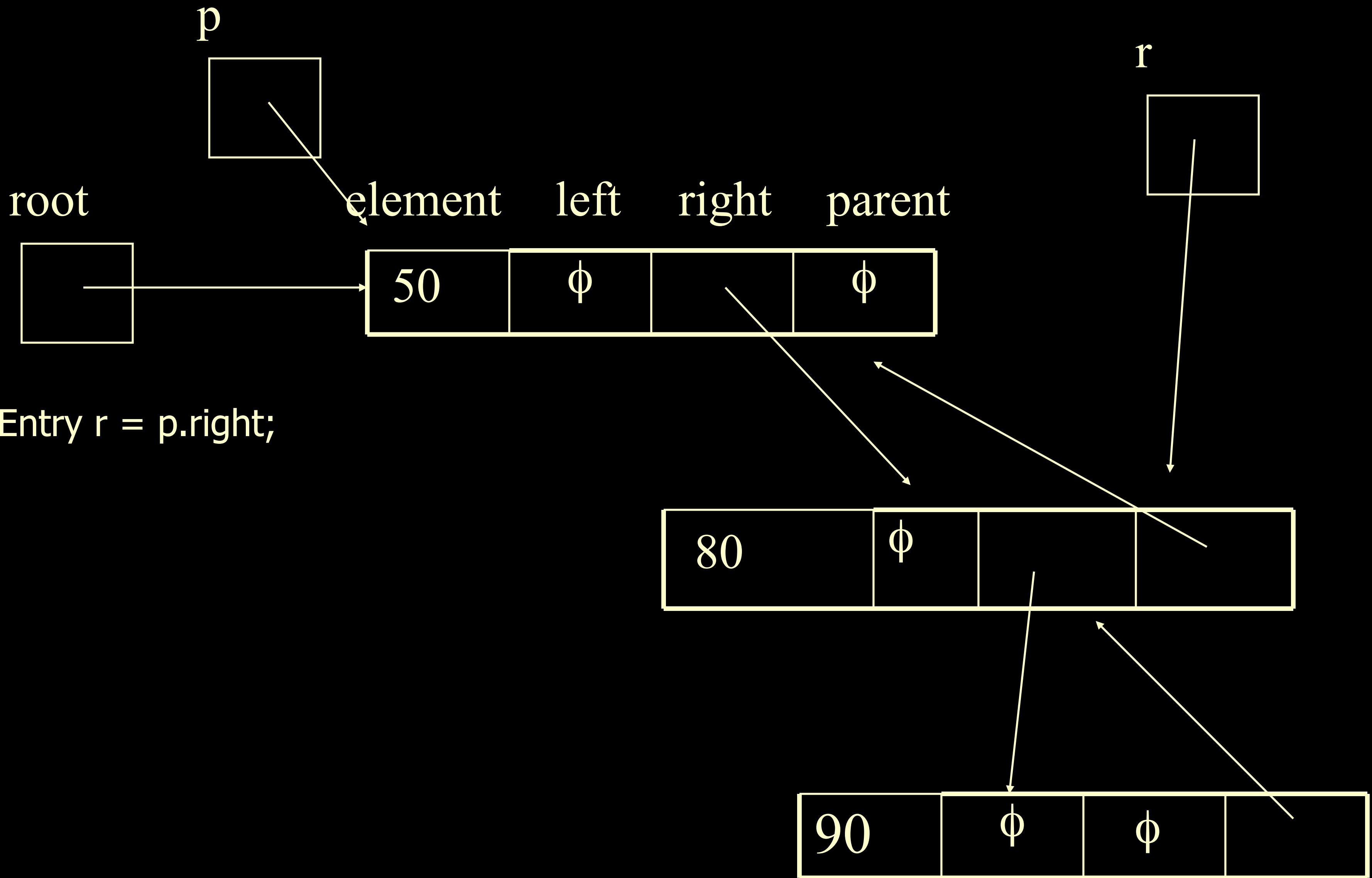
Rotation

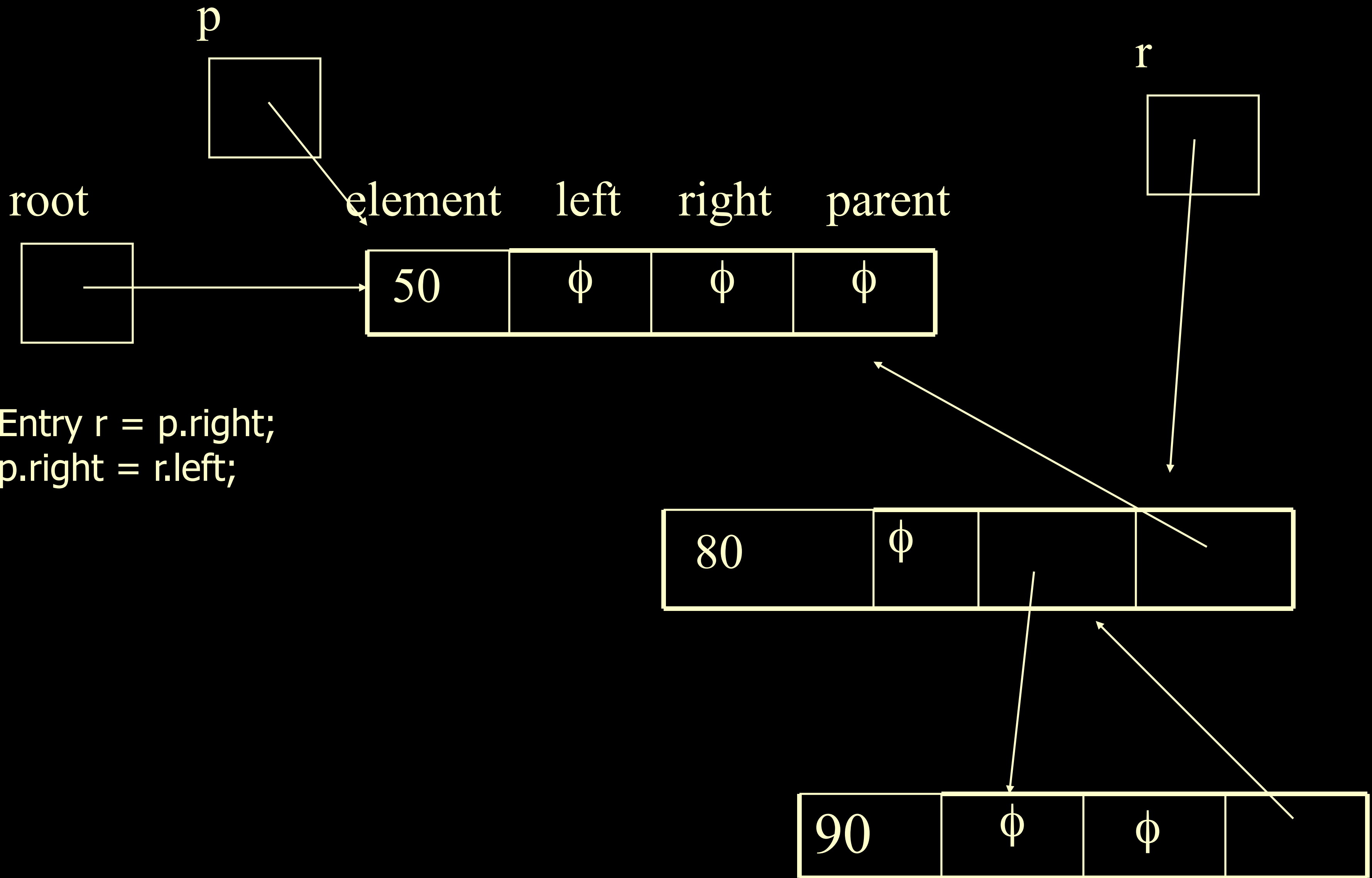
```
private void rotateLeft(Node p) {
    Node r = p.right;
    p.right = r.left;
    if (r.left != null) r.left.parent = p;
    r.parent = p.parent;
    if (p.parent == null) {root = r; r.parent = null;}
    else if (p.parent.left == p) p.parent.left = r;
    else p.parent.right = r;
    r.left = p;
    p.parent = r;
}
```

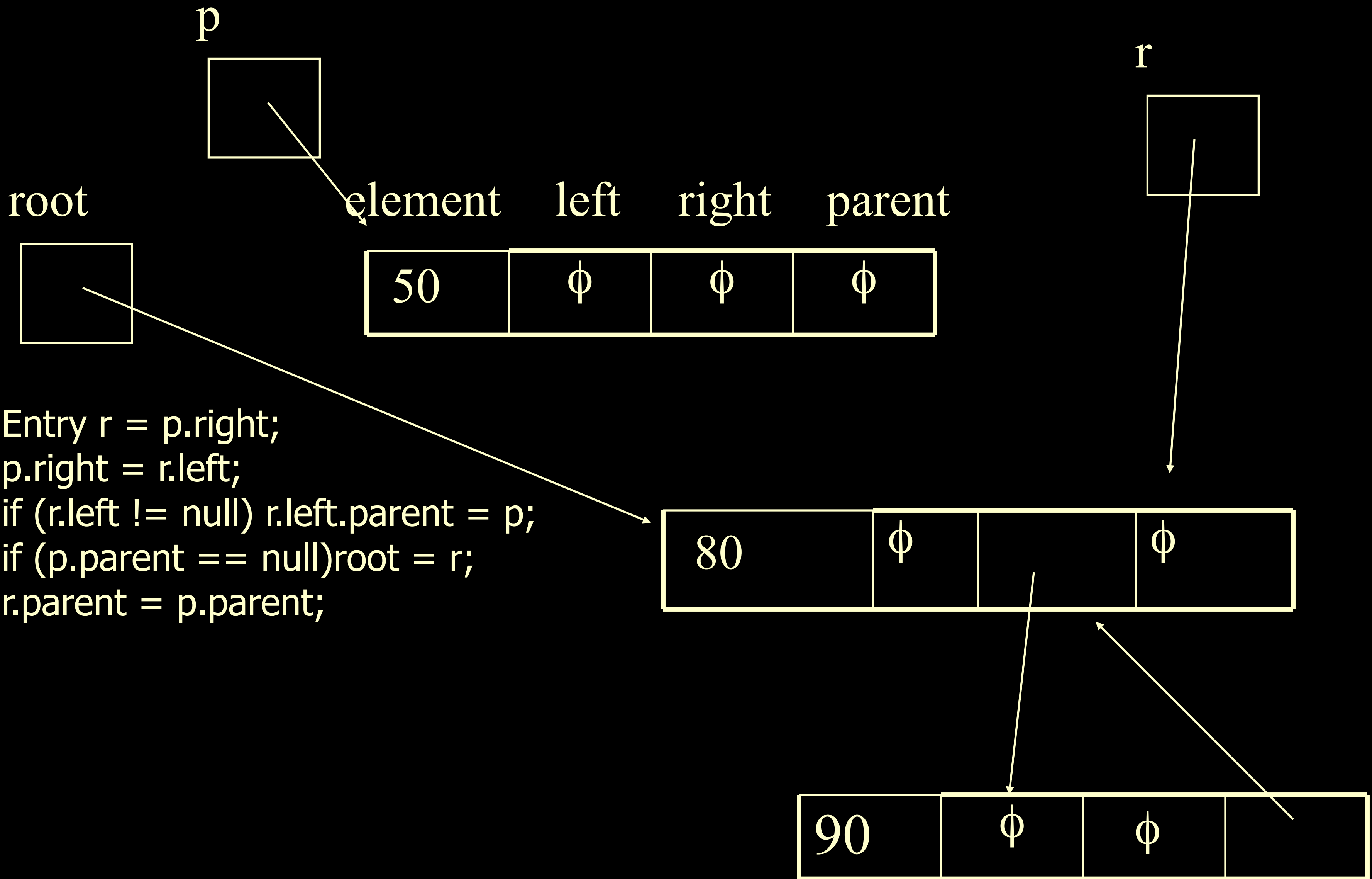
root element left right parent

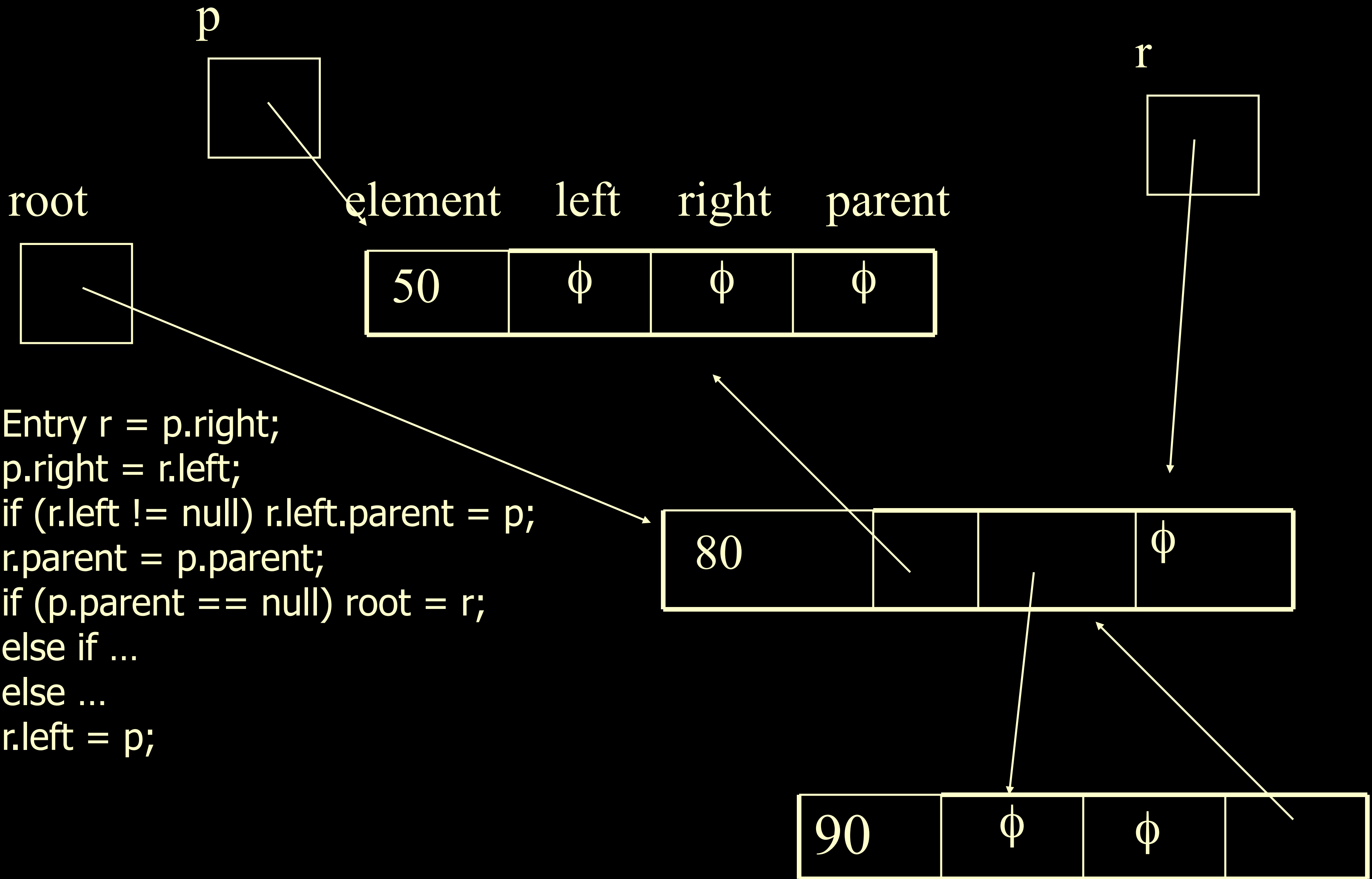


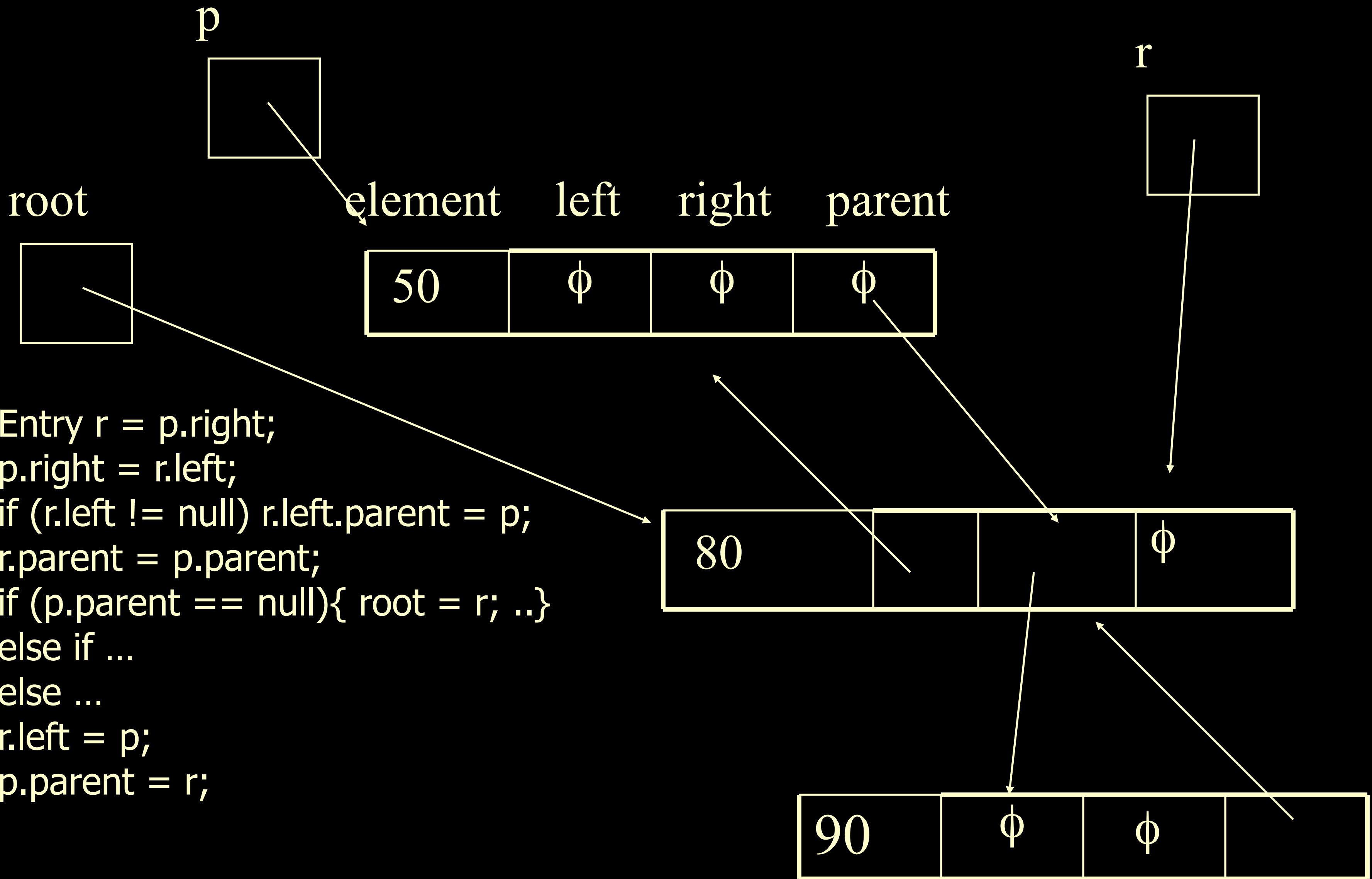



p
r
root
element
left
right
parent
50
ϕ
ϕ
Entry r = p.right;
80
ϕ
90
ϕ
ϕ


p
r
root
element
left
right
parent
50
ϕ
ϕ
ϕ
Entry r = p.right;
p.right = r.left;
80
ϕ
90
ϕ
ϕ


p
r
root
element
left
right
parent
50
ϕ
ϕ
ϕ
Entry r = p.right;
p.right = r.left;
if (r.left != null) r.left.parent = p;
if (p.parent == null)root = r;
r.parent = p.parent;
80
ϕ
ϕ
90
ϕ
ϕ

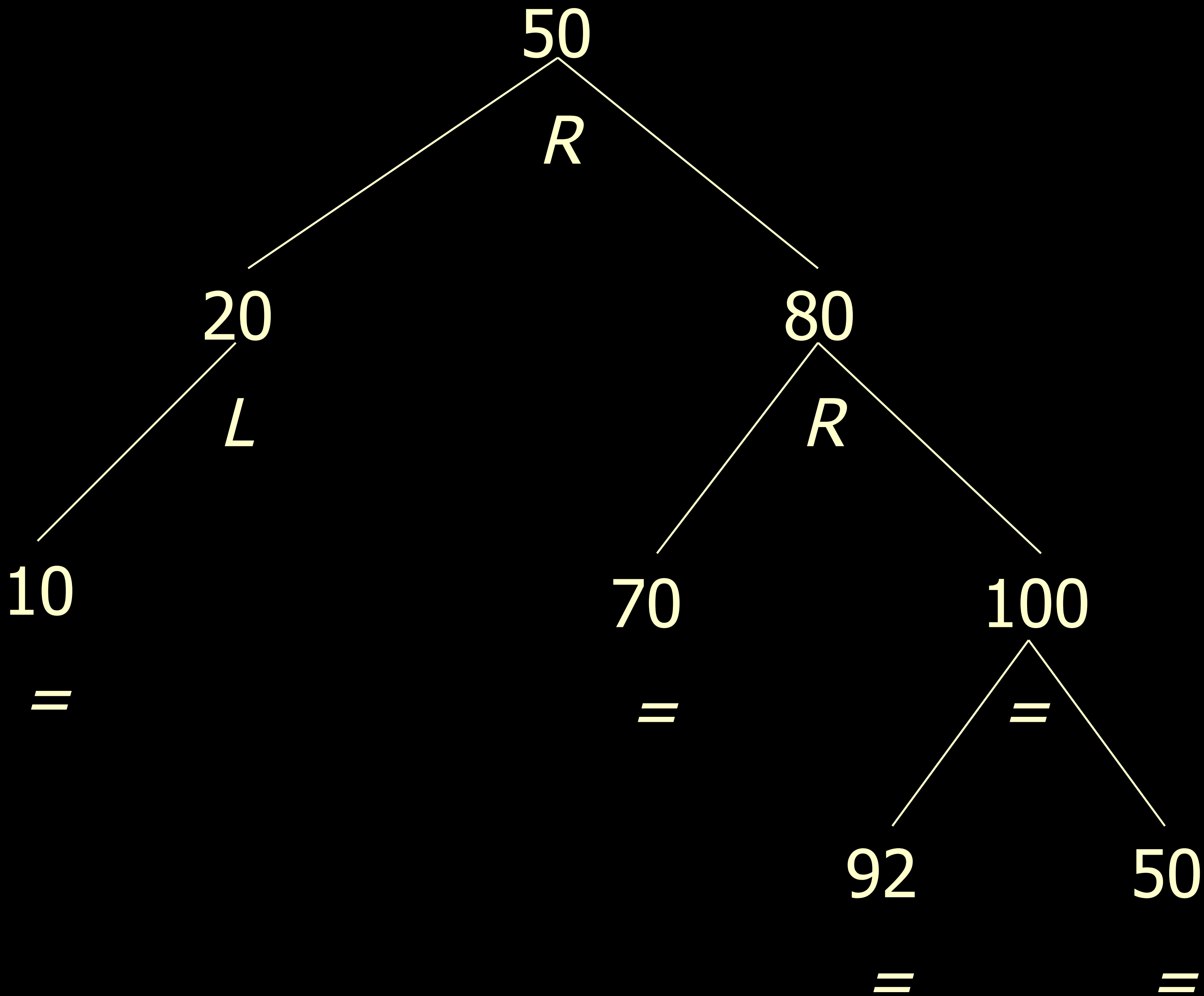

p
r
root
element
left
right
parent
50
ϕ
ϕ
ϕ
Entry r = p.right;
p.right = r.left;
if (r.left != null) r.left.parent = p;
r.parent = p.parent;
if (p.parent == null) root = r;
else if …
else …
r.left = p;
80
ϕ
90
ϕ
ϕ



p
r
root
element left right parent
50 ϕ ϕ ϕ
80 ϕ
90 ϕ ϕ
Entry r = p.right;
p.right = r.left;
if (r.left != null) r.left.parent = p;
r.parent = p.parent;
if (p.parent == null){ root = r; ..}
else if …
else …
r.left = p;
p.parent = r;

Node class

```
private static class Node {
    Object element;
    char balanceFactor = ‘=‘; // new nodes are balanced
    // we could set this to R or L indicating child with > height
    Node left = null, right = null, parent;
    Node (Object element, Node parent) {
        this.element = element;
        this.parent = parent;
    }
}
```

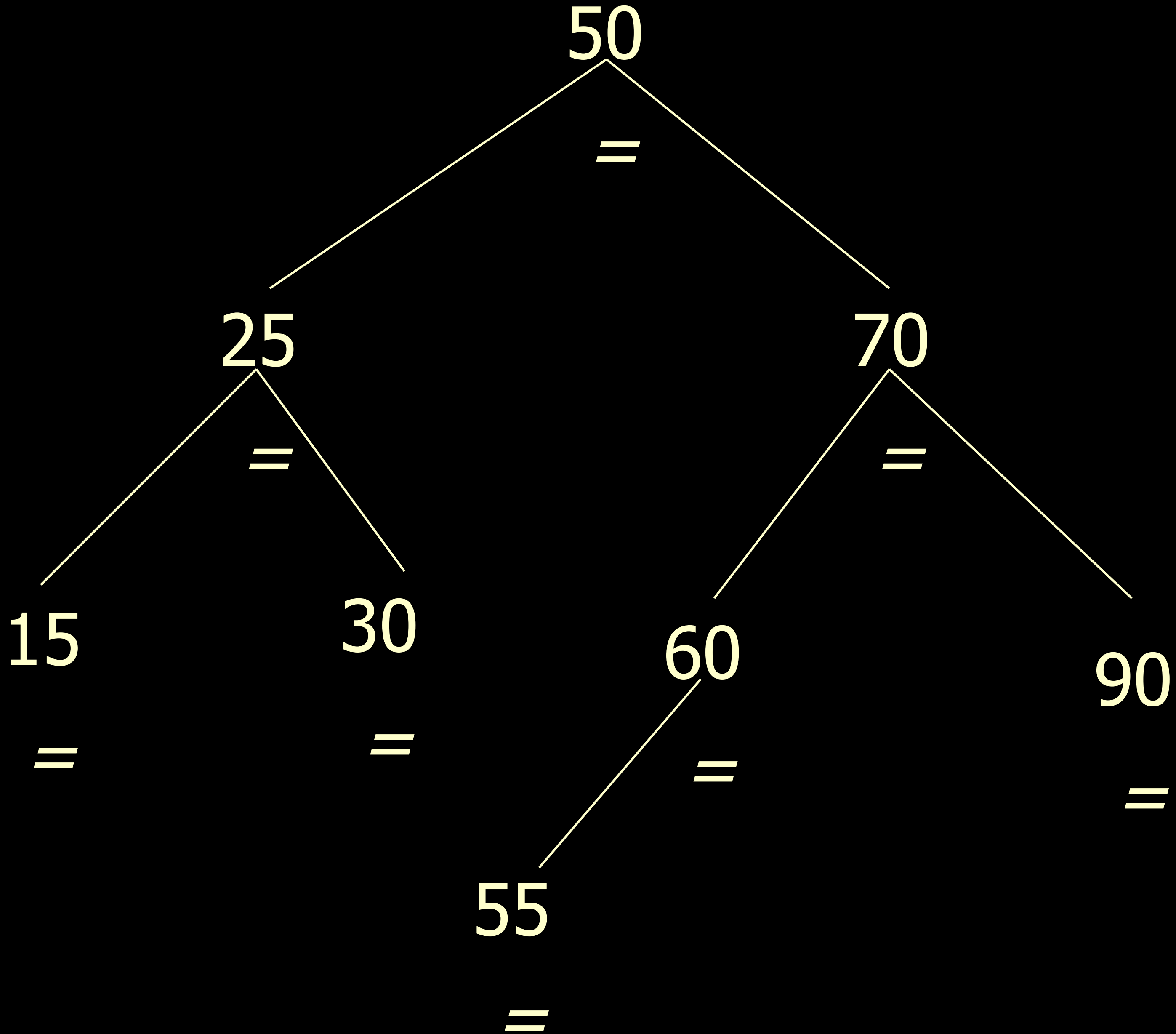

50
R
20
80
L
R
10
70
100
=
=
=
92
50
=
=

```
public boolen add(Object o){
    if (root == null) {
        root = new Node(o,null);
        size++;
        return true;
    }// empty tree
    else {
        Node temp = root,
        ancestor = null; // we keep track of nearest unbalanced ancestor
        int comp;
        while (true) {
            comp = ((Comparable)o).compareTo(temp.element);
            if (comp == 0) return false;
```

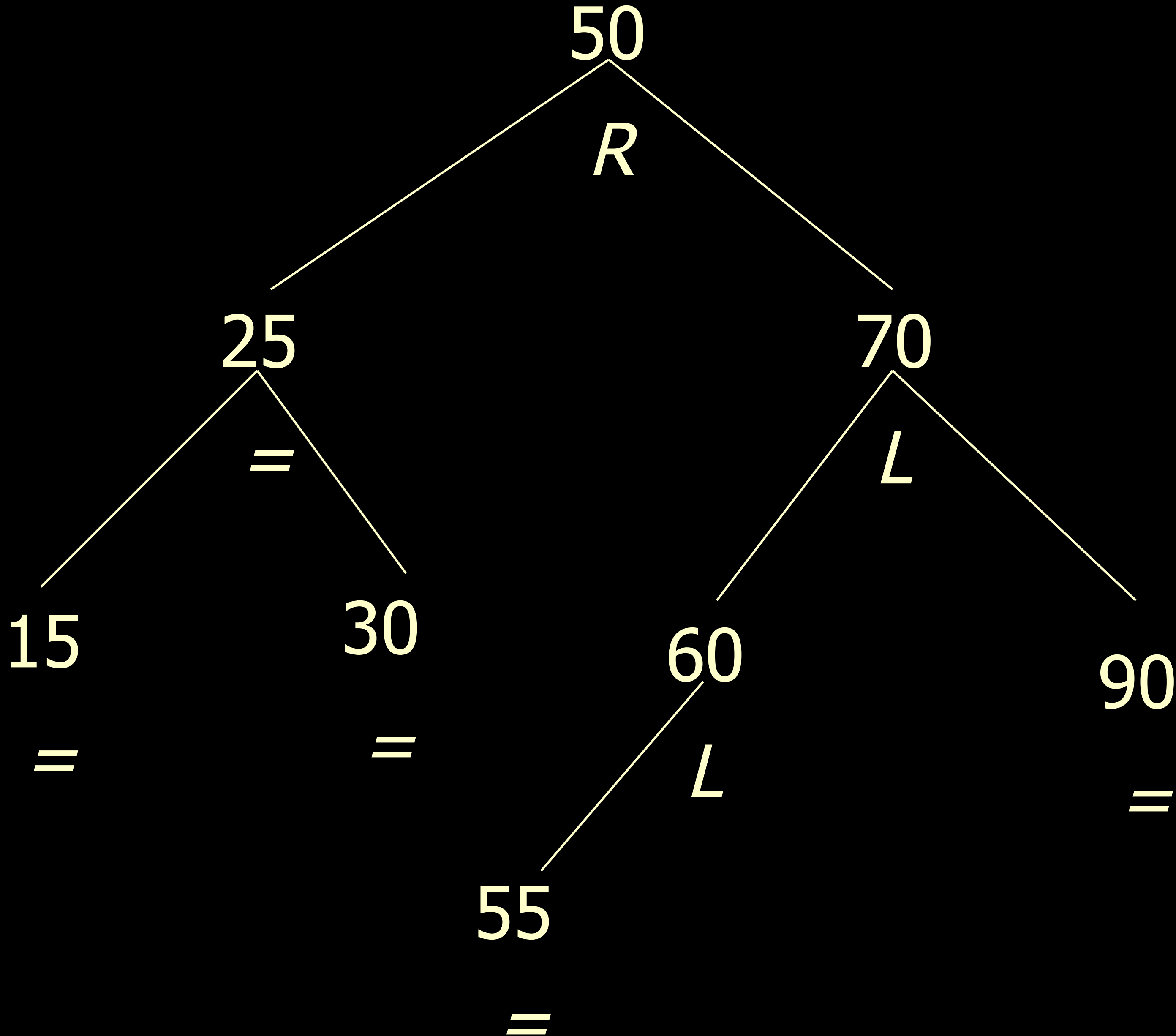
```
if (comp < 0) {
        if (temp.balanceFactor != ‘=‘) ancestor = temp;
        if (temp.left != null) temp = temp.left;
        else {
                temp.left = new Node(o,temp);
                fixAfterInsertion(ancestor,temp.left);
                size++;
        }
}// comp < 0
```

```
            else { // comp > 0
                  if (temp.balanceFactor != ‘=‘) ancestor = temp;
                  if (temp.right != null) temp = temp.right;
                  else {
                        temp.rig = new Node(o,temp);
                        fixAfterInsertion(ancestor,temp.right);
                        size++;
                  }
            }// comp < 0
      }// while
   }// root not null
}//method add
```

Adjusting paths



Adjusting paths



```
protected void adjustPath(Entry to, Entry inserted) {
    Object o = inserted.element;
    Node temp = inserted.parent;
    while (temp != to) {
        if (((Comparable)o).compareTo(temp.element)<0)
                temp.balanceFactor = ‘L’;
        else
                temp.balanceFactor = ‘R’;
        temp= temp.parent
    }// while
} //adjust path
```

```
protected void fixAfterInsertion(Node ancestor, Entry inserted) {
    Object o = inserted element;
    if (ancestor == null) {
        if (((Comparable)o).compareTo(root.element)<0)
                root.balanceFactor = ‘L’;
        else
                root.balanceFactor = ‘R’;
        adjustPath(root,inserted);
    } // Case 1: all ancestor of inserted element have ‘=‘ balanceFactor
```

```
if ((ancestor.balanceFactor == ‘L’ &&
      ((Comparable)o).compareTo(ancestor.element)>0)||
      (ancestor.balanceFactor == ‘R’ &&
      ((Comparable)o).compareTo(ancestor.element)<0)){
              ancestor.balanceFactor = ‘=’;
              adjustPath(ancestor,inserted);
} // Case 2: insertion causes ancestor’s balanceFactor to ‘=‘
```

```
if ((ancestor.balanceFactor == 'R' &&
     ((Comparable)o).compareTo(ancestor.right.element)>0){
          ancestor.balanceFactor = '=';
          rotateLeft(ancestor);
          adjustPath(ancestor.parent,inserted);
} // Case 3: ancestor's balance factor = 'R'
  //         and o > ancestor's right child
```

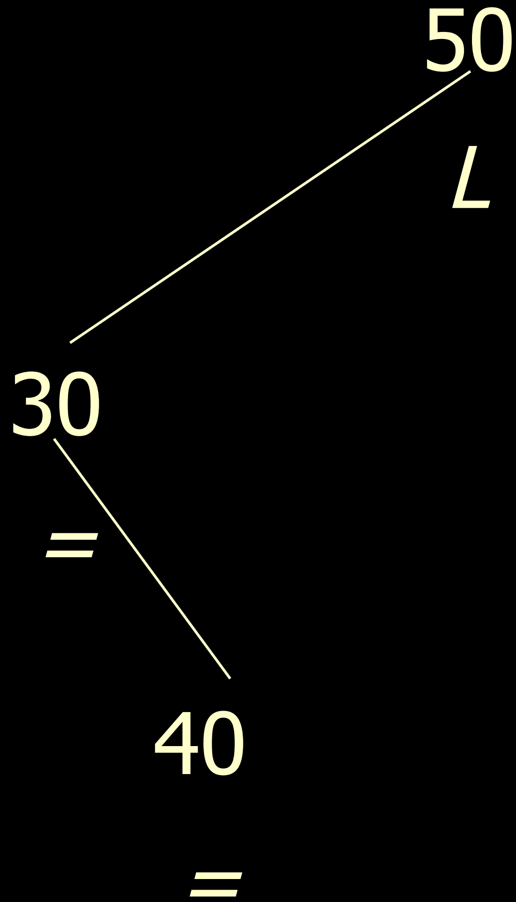
```
        if ((ancestor.balanceFactor == ‘L’ &&
        ((Comparable)o).compareTo(ancestor.left.element)<0){
                ancestor.balanceFactor = ‘=’;
                rotateRight(ancestor);
                adjustPath(ancestor.parent,inserted);
} // Case 4: ancestor’s balance factor = ‘L’
  //            and o < ancestor’s right child
```

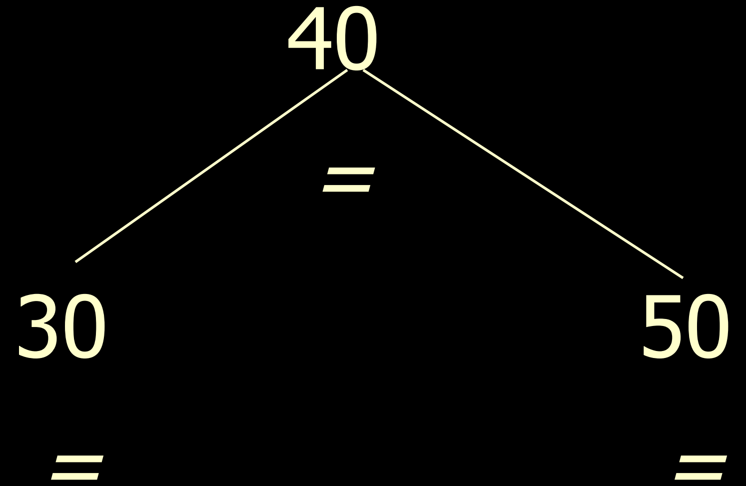
```
if (ancestor.balanceFactor == 'L' &&
    ((Comparable)o).compareTo(ancestor.left.element)>0){
        rotateLeft(ancestor.left);
        rotateRight(ancestor);
        adjustLeftRight(ancestor,inserted);
} // Case 5: ancestor's balanceFactor = 'L'
  //     and o > ancestor's left child
```

```
        else{
                rotateRight(ancestor.right);
                rotateLeft(ancestor);
                adjustRightLeft(ancestor,inserted);
    } // Case 6: ancestor's balanceFactor = 'R'
      //      and o < ancestor's right child
}// fixAfterInsertion
```

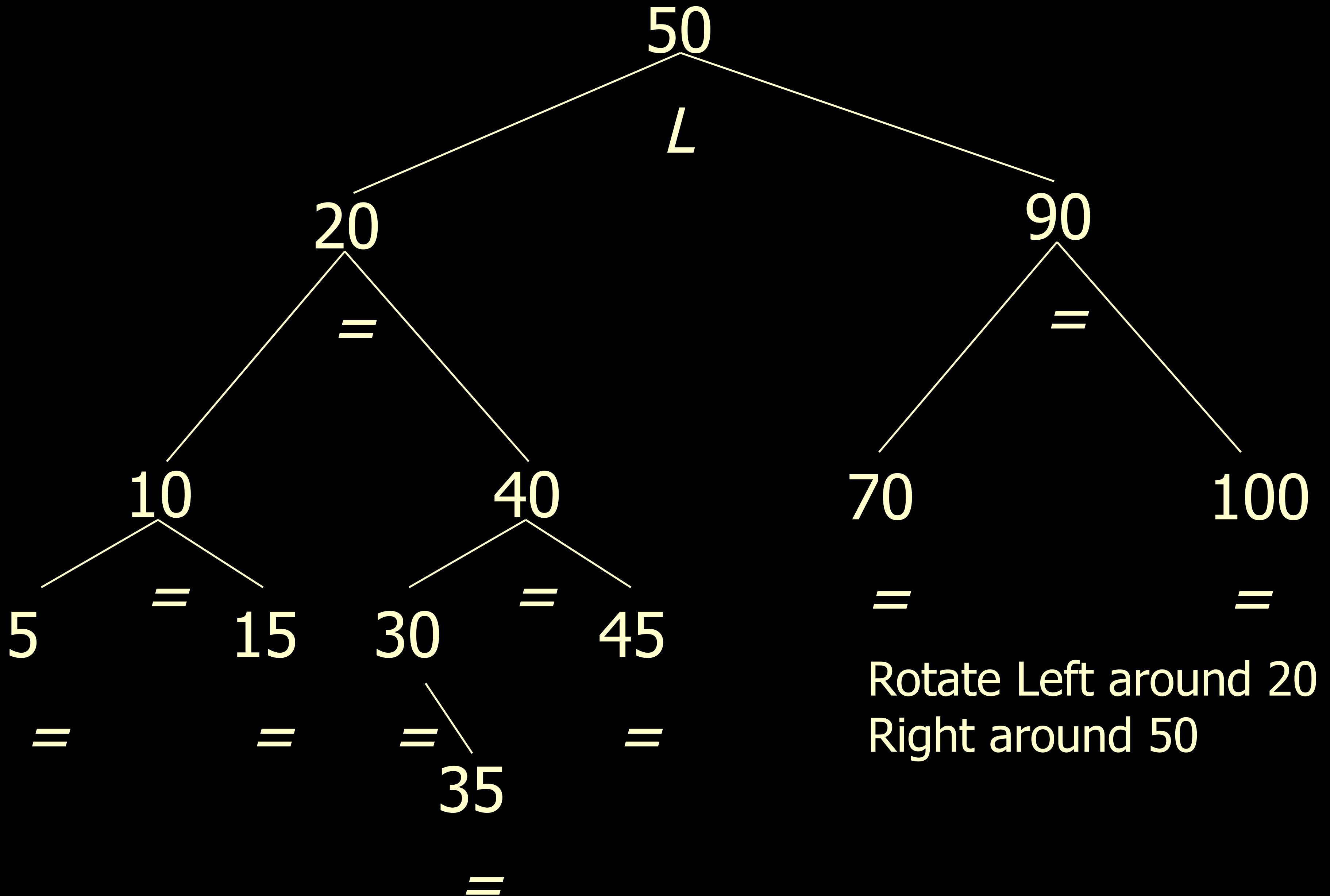
```
protected void adjustLeftRight(Entry ancestor, Entry inserted) {
    Object o = inserted.element;
    if (ancestor.parent == inserted) ancestor.balanceFactor = '=';
    else if (((Comparable)o).compareTo(ancestor.parent.element)<0){
                    ancestor.balanceFactor = 'R';
                    adjustPath(ancestor.parent.left,inserted);
    }// o < ancestor's parent
    else  {
          ancestor.balanceFactor = '=';
          ancestor.parent.left.balanceFactor = 'L';
          adjustPath(ancestor,inserted);
    }// while
} //adjustLeftRight
```

AdjustLeftRight Case 1

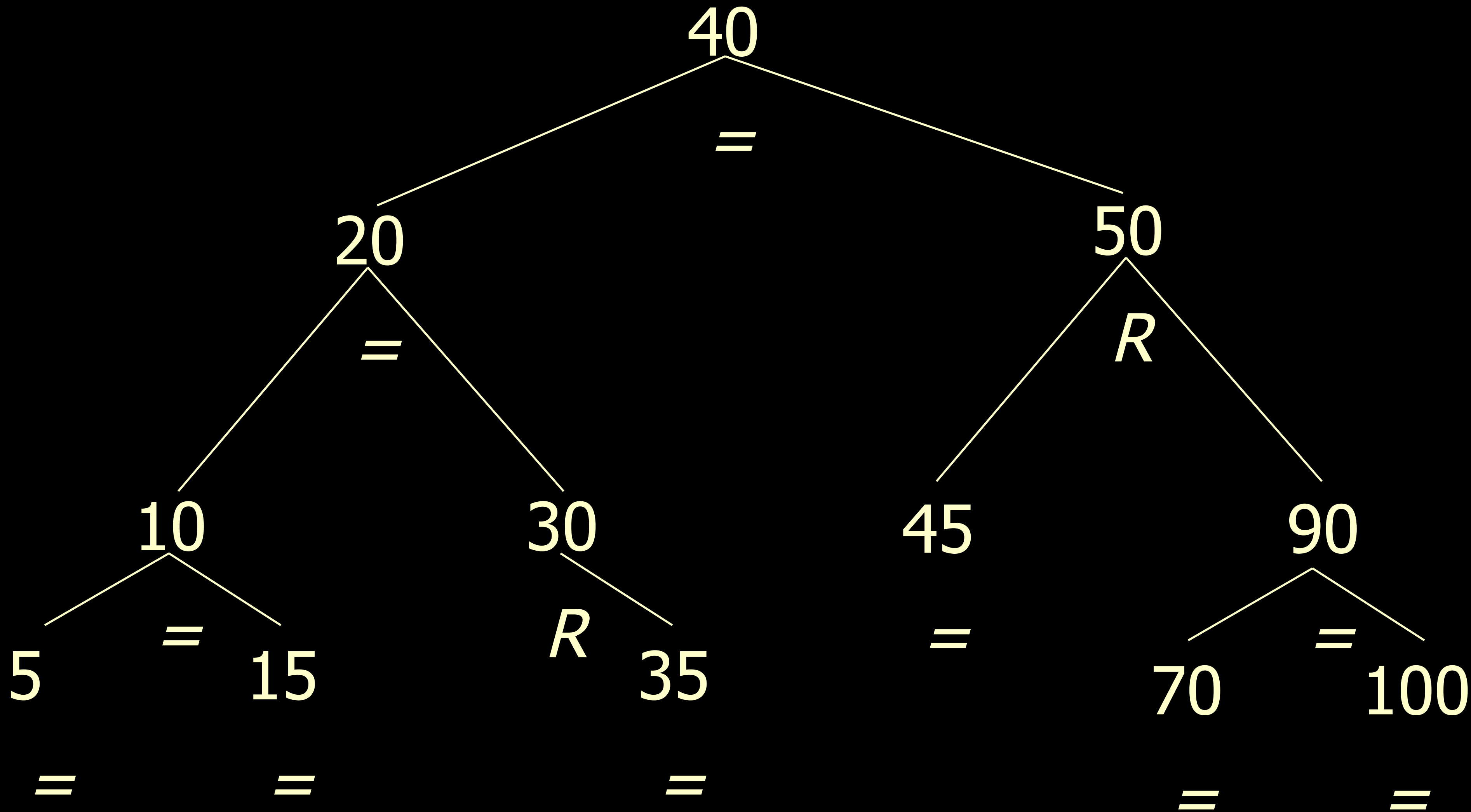




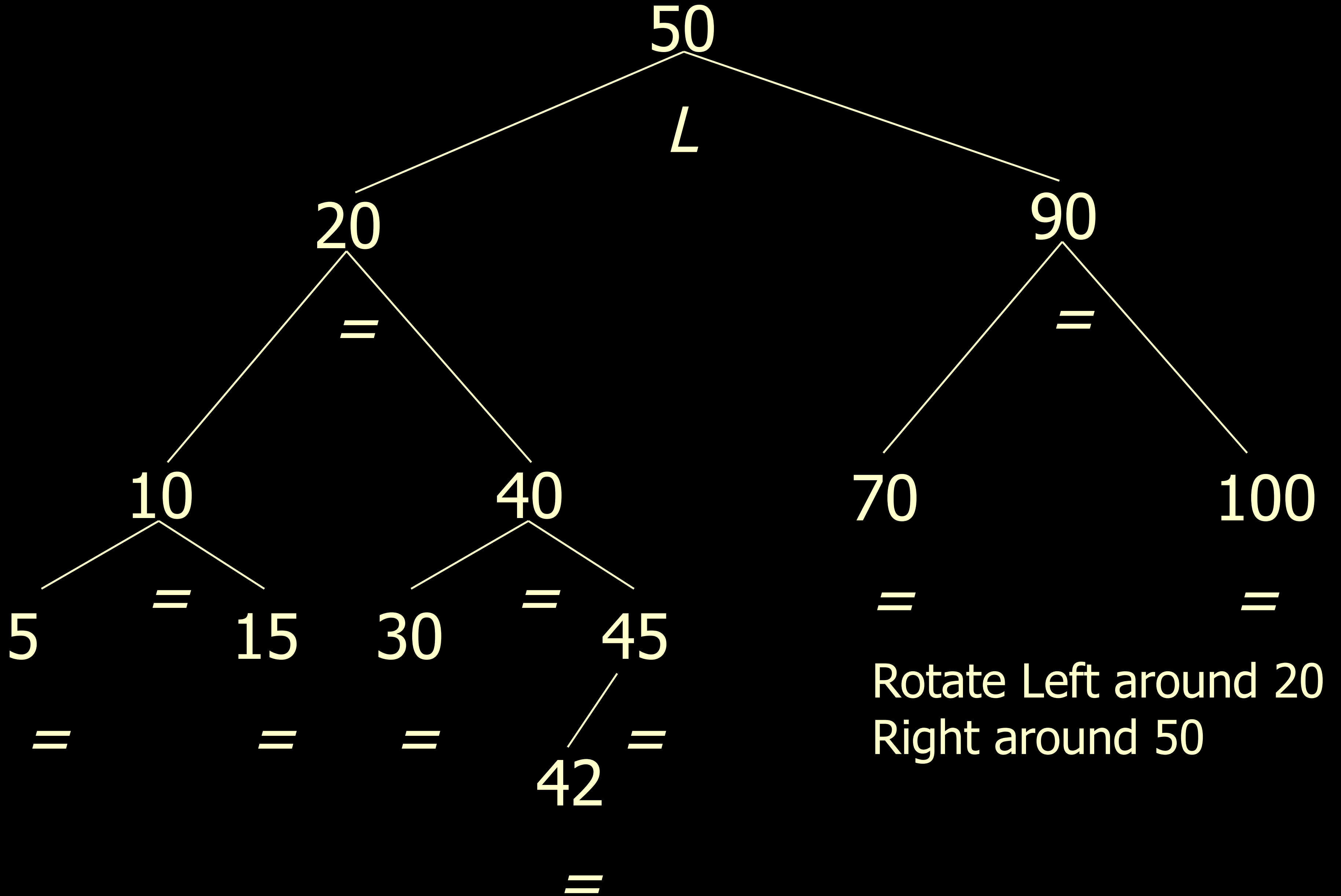
AdjustLeftRight Case2



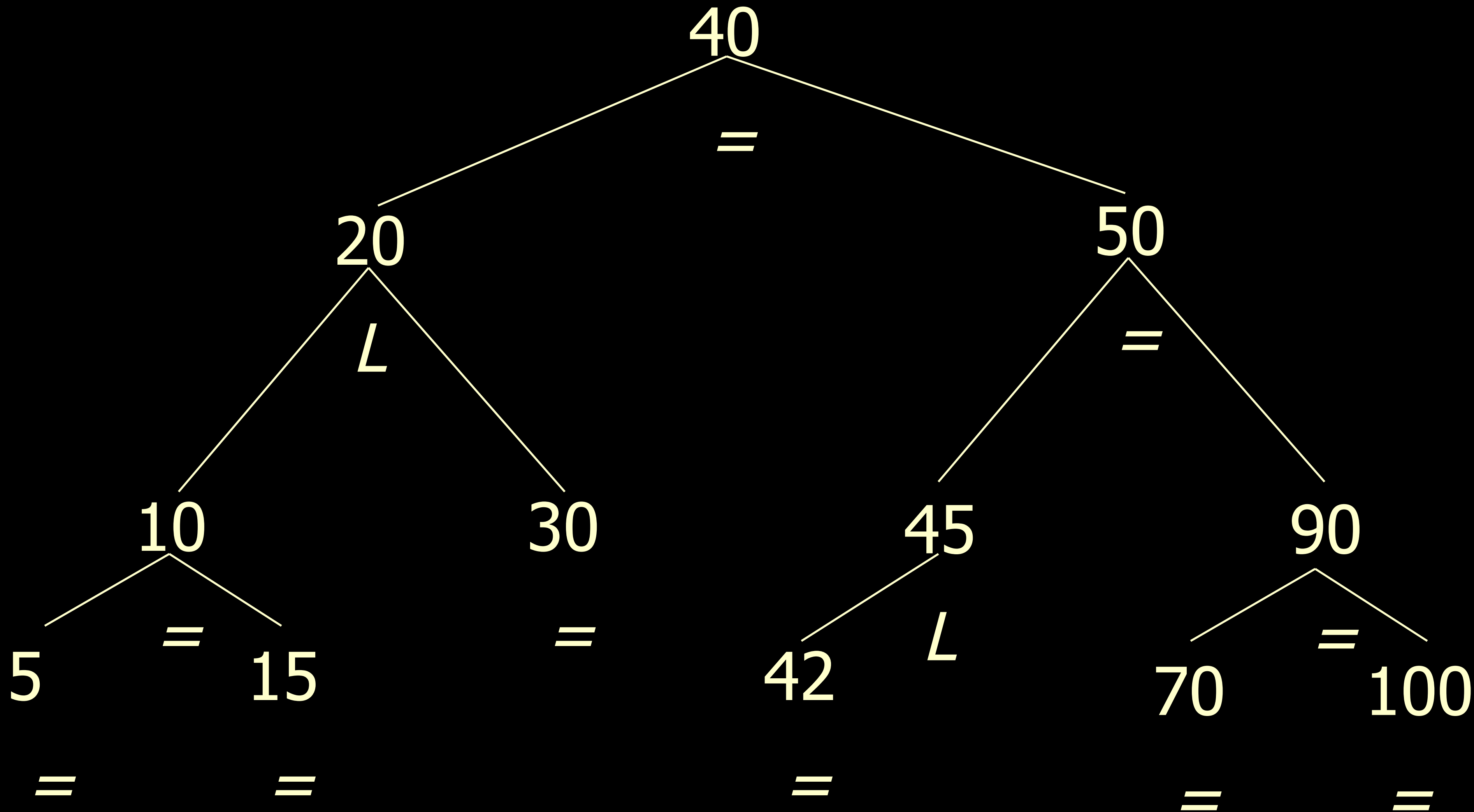
AdjustLeftRight Case2



AdjustLeftRight Case3



AdjustLeftRight Case3



```
protected void adjustRightLeft(Entry ancestor, Entry inserted) {
    Object o = inserted.element;
    if (ancestor.parent == inserted) ancestor.balanceFactor = '=';
    else if (((Comparable)o).compareTo(ancestor.parent.element)>0){
                    ancestor.balanceFactor = 'L';
                    adjustPath(ancestor.parent.right,inserted);
    }// o < ancestor's parent
    else  {
            ancestor.balanceFactor = '=';
            ancestor.parent.right.balanceFactor = 'R';
            adjustPath(ancestor,inserted);
    }// while
} //adjustRightLeft
```