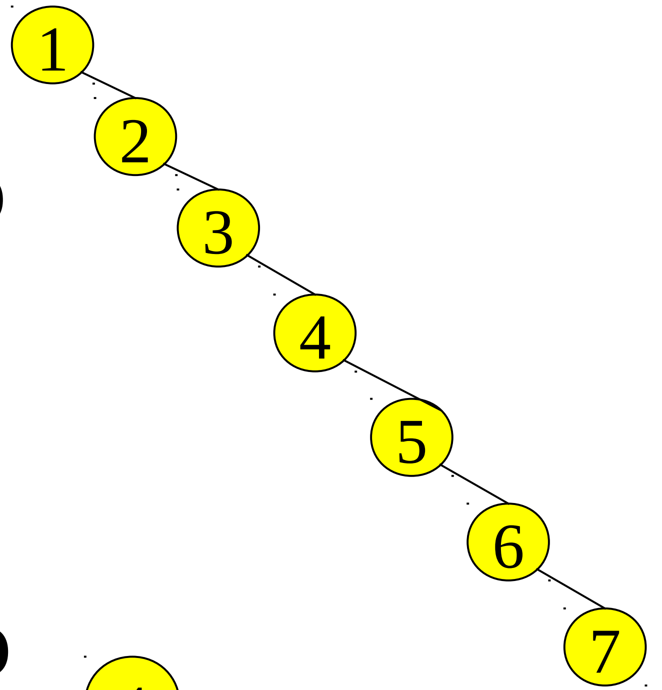
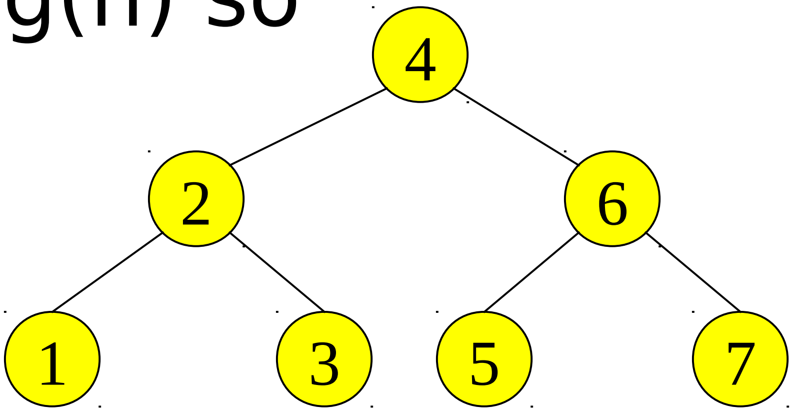
AVL TREE

Need for AVL Trees

- A binary tree can be of height n-1 which leads to complexities of O(n)

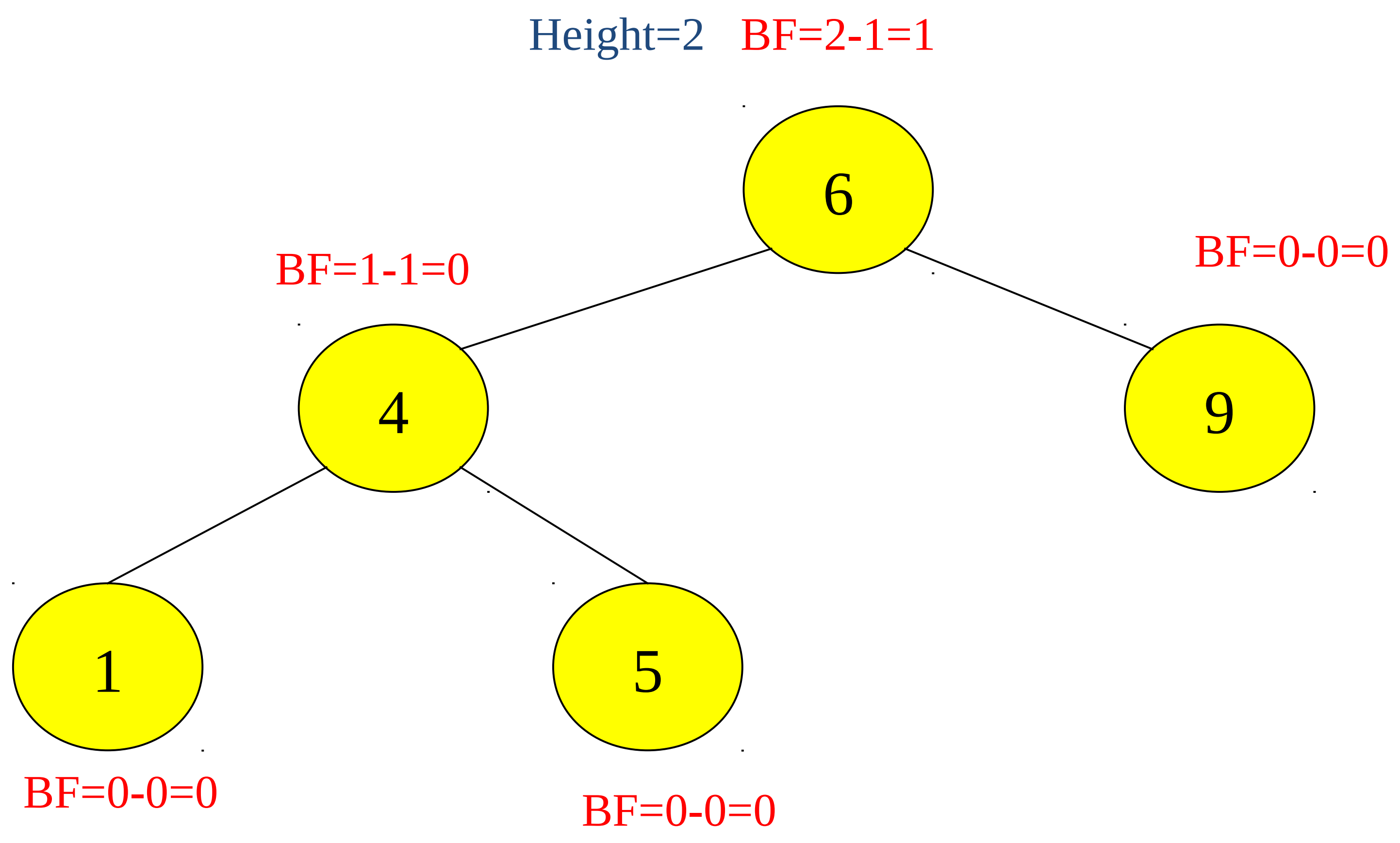


- By balancing the tree, height becomes log(n) so better complexity



What is an AVL Tree?

- Height balanced binary search tree
- Balance factor of node
  - Height(left subtree) – Height(right subtree)
- An AVL tree has balance factor calculated at every node
  - For every node, heights of left and right subtree can differ by no more than 1
  - Store current heights in each node


Height=2
BF=2-1=1
6
BF=1-1=0
4
9
BF=0-0=0
1
5
BF=0-0=0
BF=0-0=0

Insertion/deletion

- Since an insertion/deletion involves adding / deleting a single node, this can only increase / decrease the height of some subtree by 1
- Thus, if the AVL tree property is violated at a node x, it means that the heights of left(x) ad right(x) differ by exactly 2.
- **Rotations** will be applied to x to restore the AVL tree property.

Insertions in AVL Trees

Let the node that needs rebalancing be α.

There are 4 cases:

Outside Cases (require single rotation) :

1. Insertion into left subtree of left child of α.
2. Insertion into right subtree of right child of α.

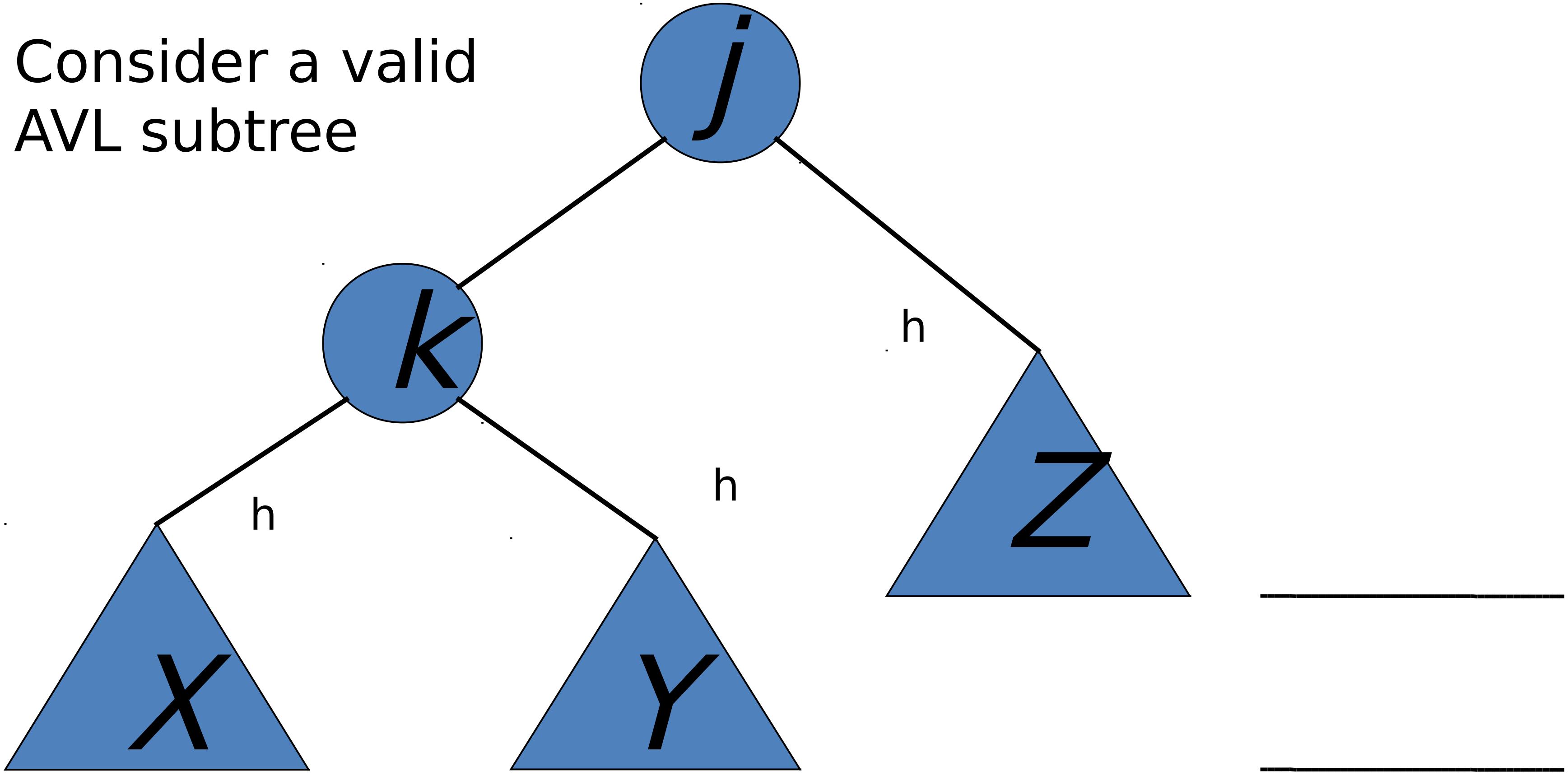
Inside Cases (require double rotation) :

3. Insertion into right subtree of left child of α.
4. Insertion into left subtree of right child of α.

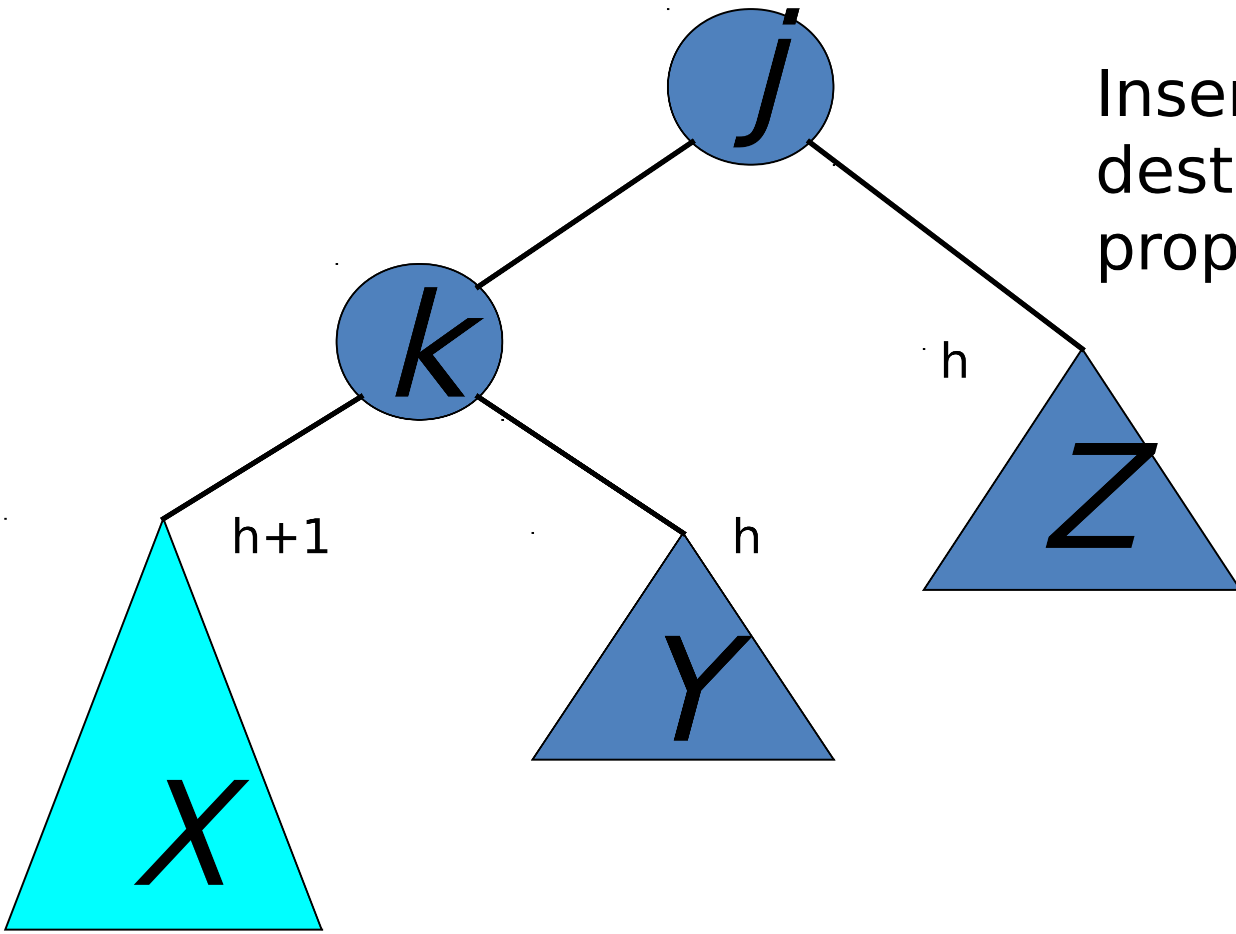
NOTE: Same methods are applicable for deletion also

AVL Insertion: Outside Case

Consider a valid AVL subtree

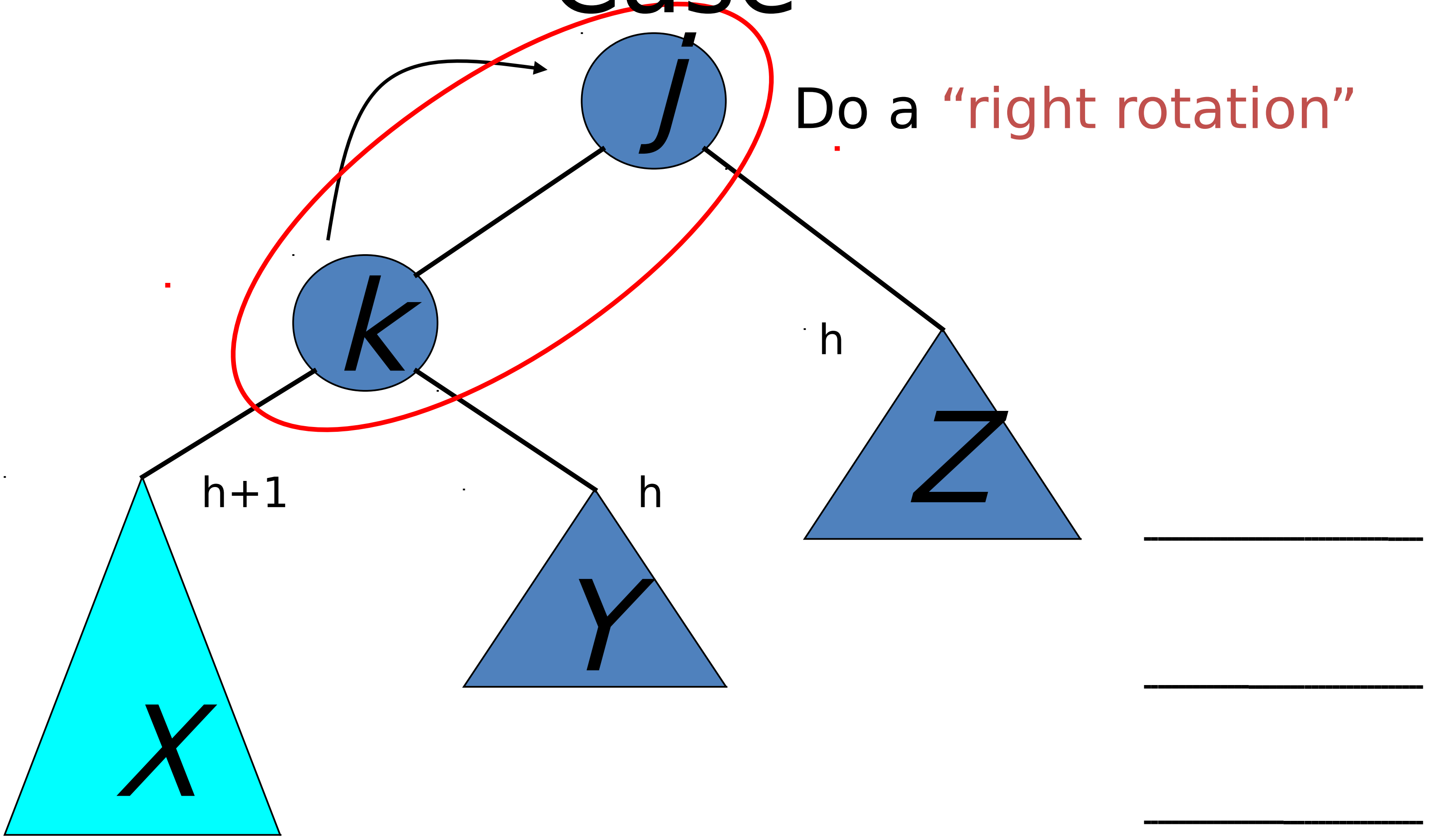


AVL Insertion: Outside Case



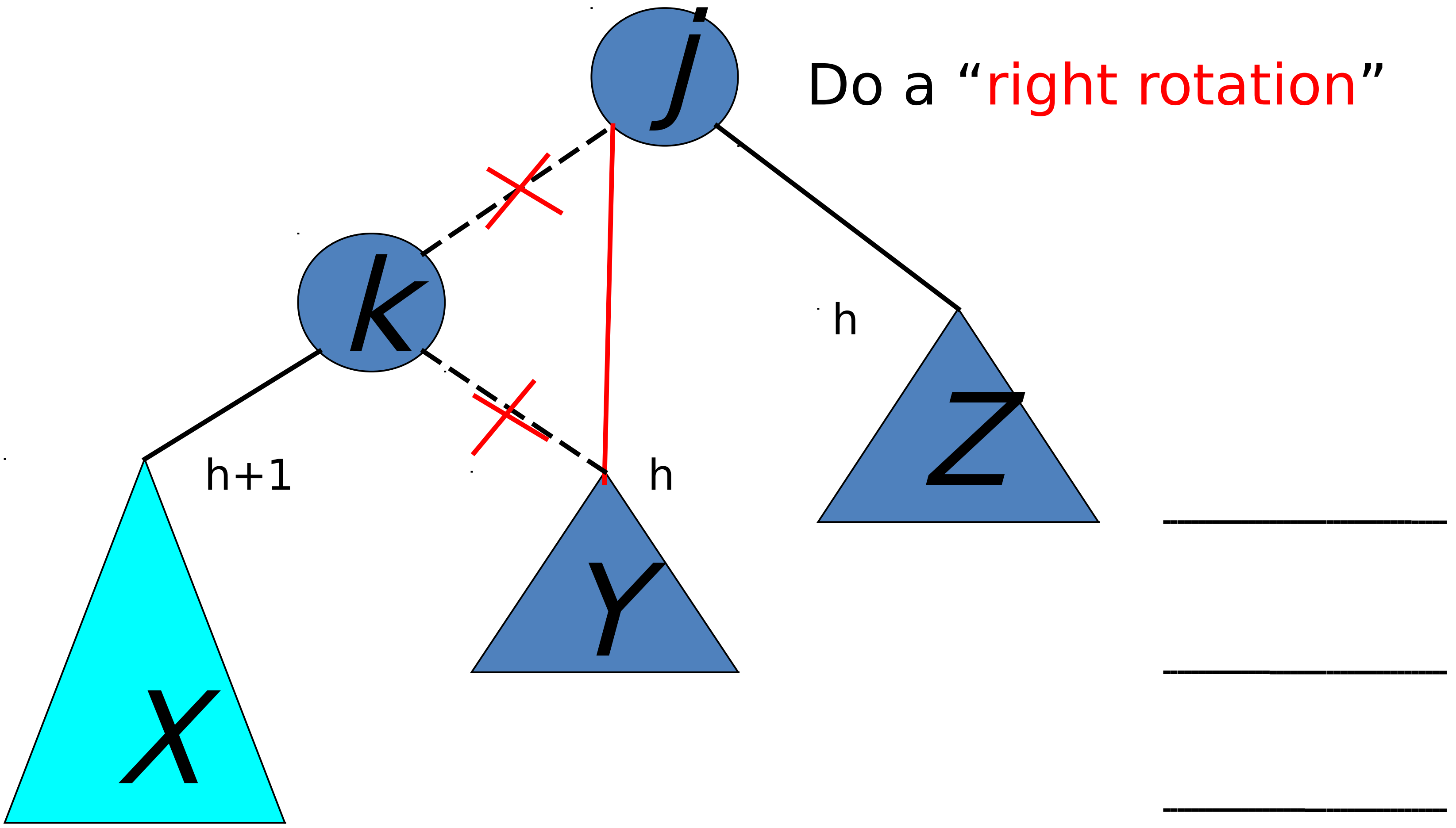
Inserting into X destroys the AVL property at node j

AVL Insertion: Outside Case

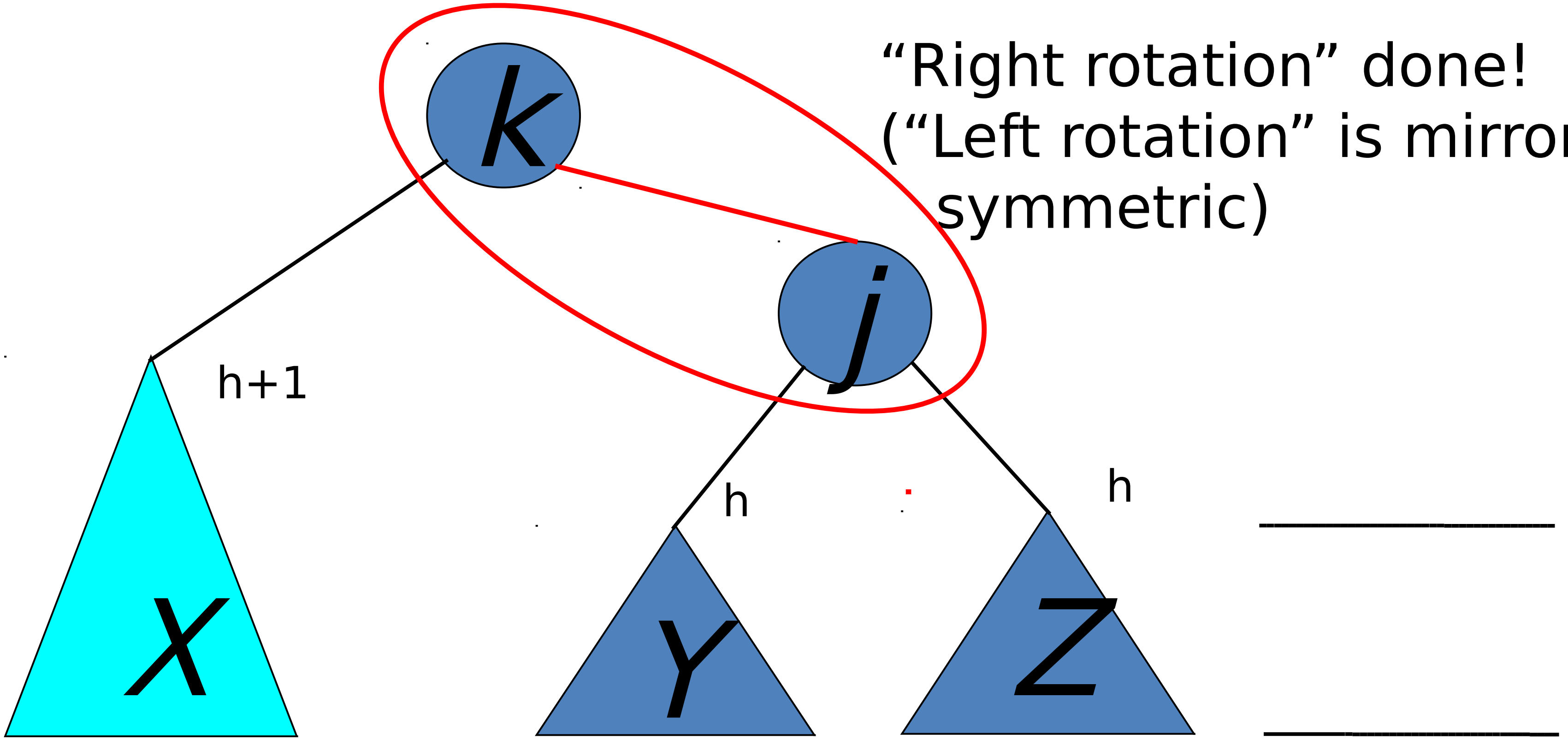


Single right rotation

Do a “right rotation”



Outside Case Completed

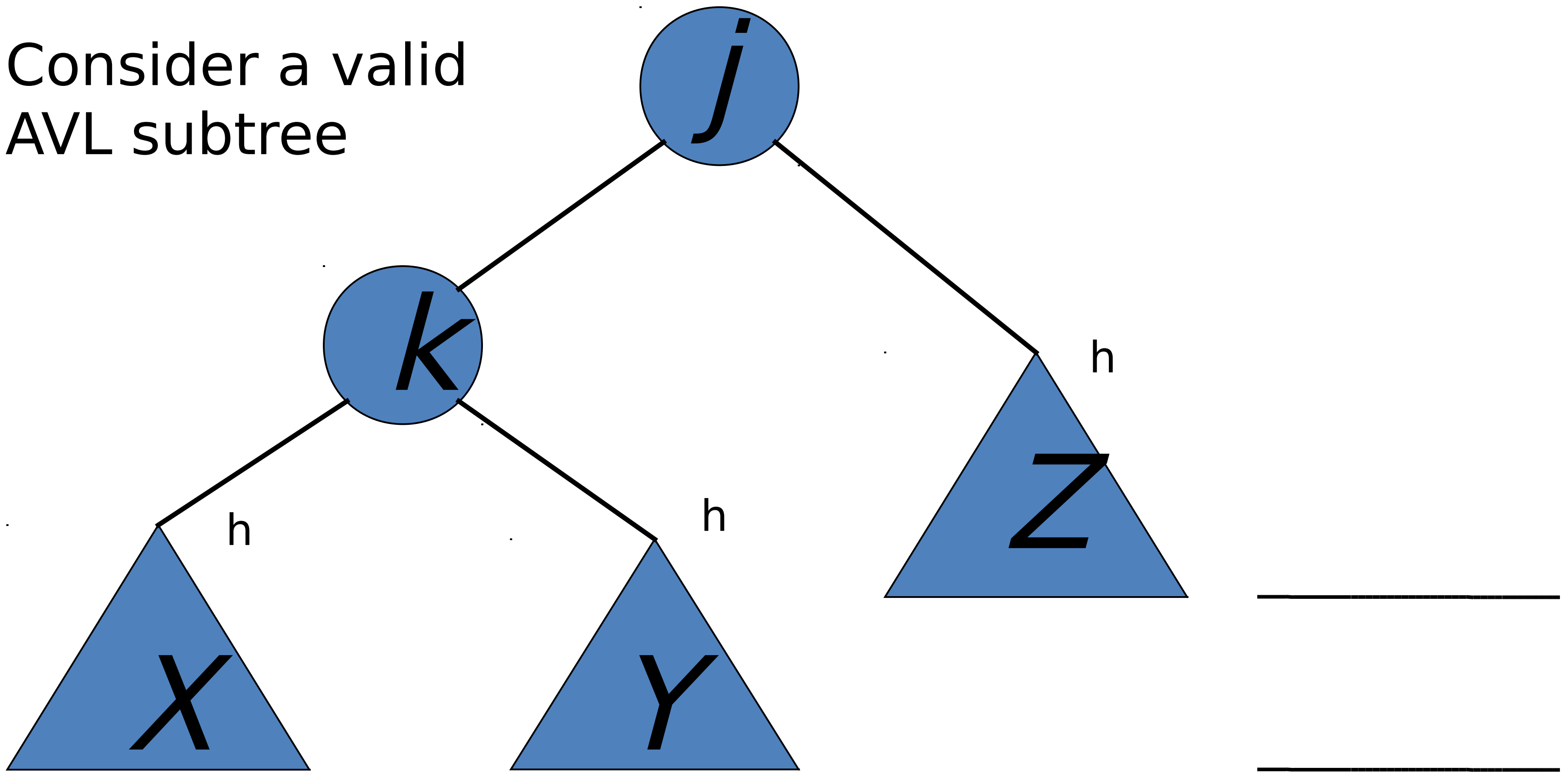


"Right rotation" done!
("Left rotation" is mirror symmetric)

AVL property has been restored!

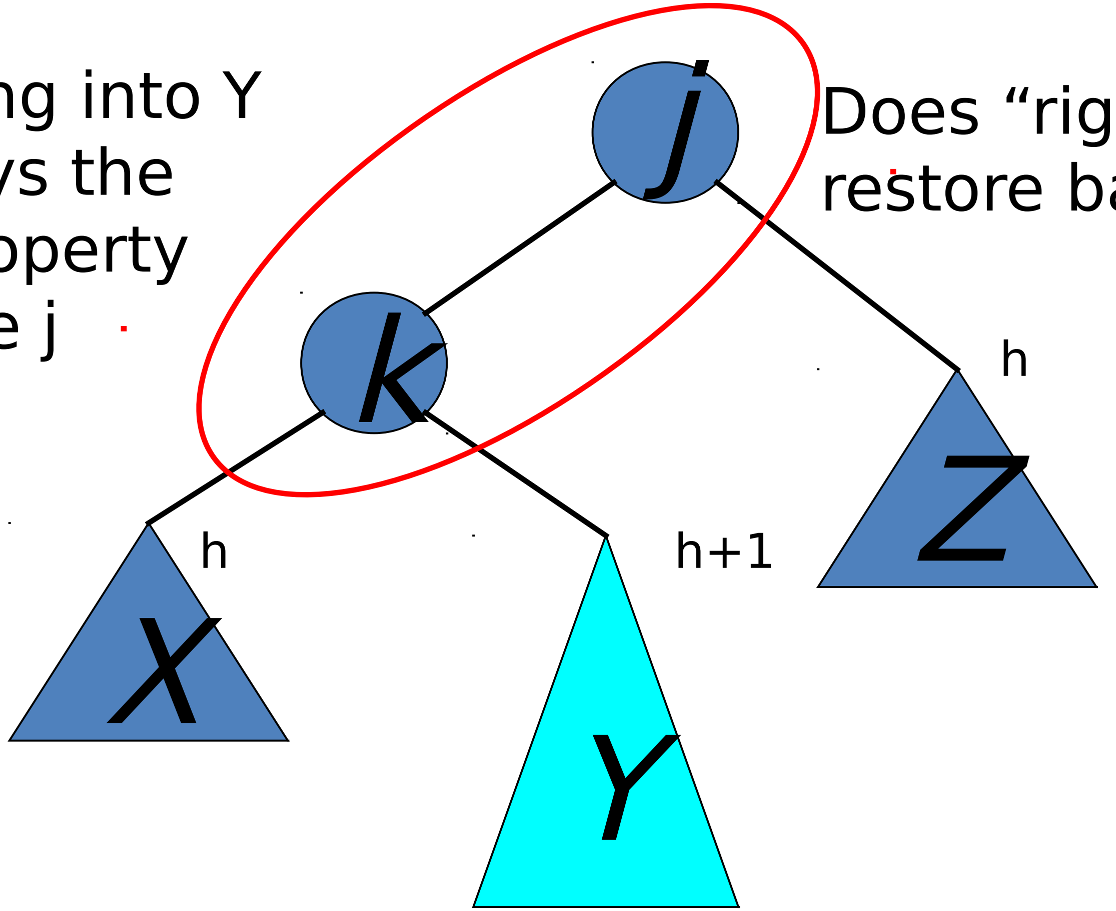
AVL Insertion: Inside Case

Consider a valid AVL subtree



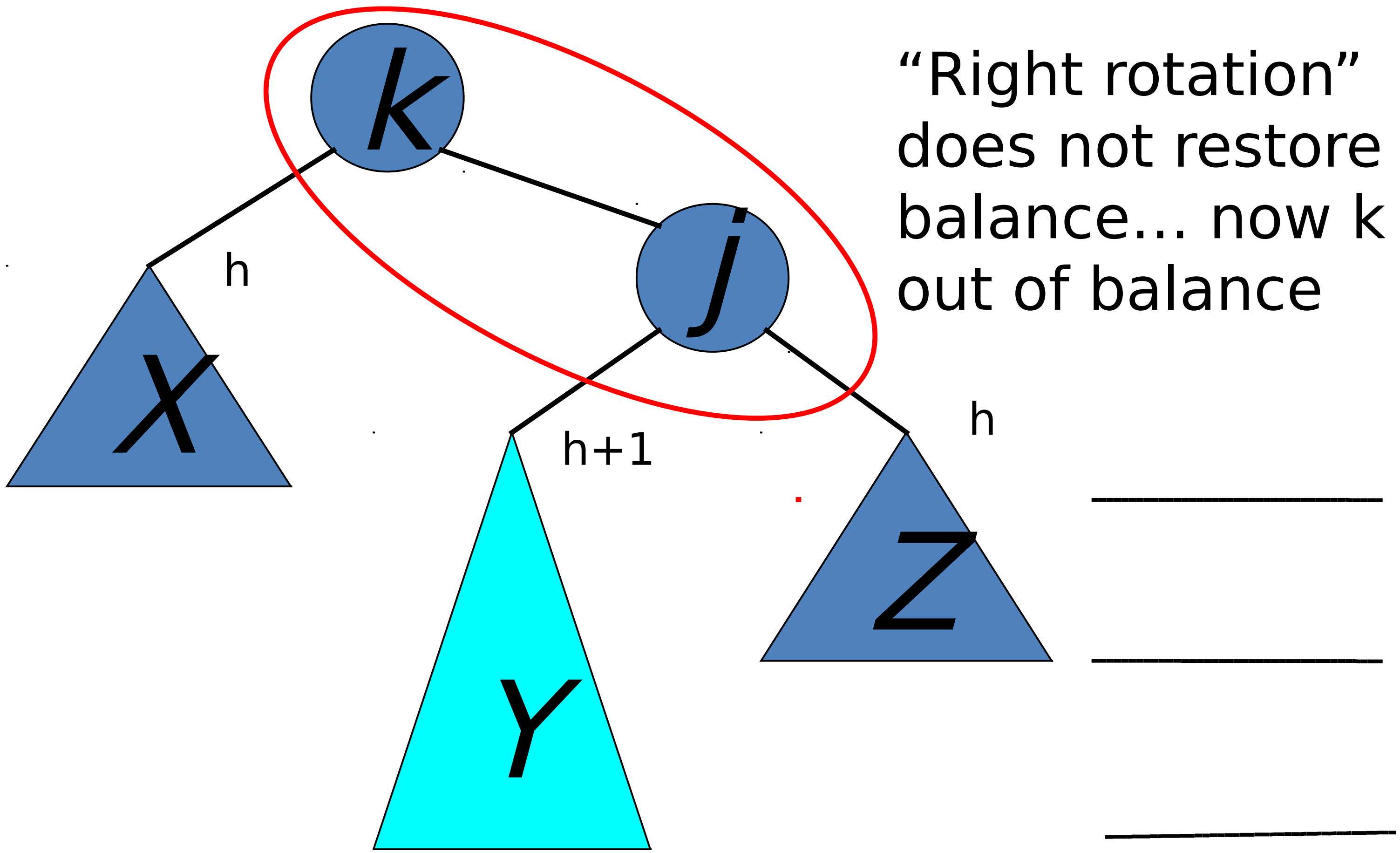
AVL Insertion: Inside Case

Inserting into Y destroys the AVL property at node j



Does "right rotation" restore balance?

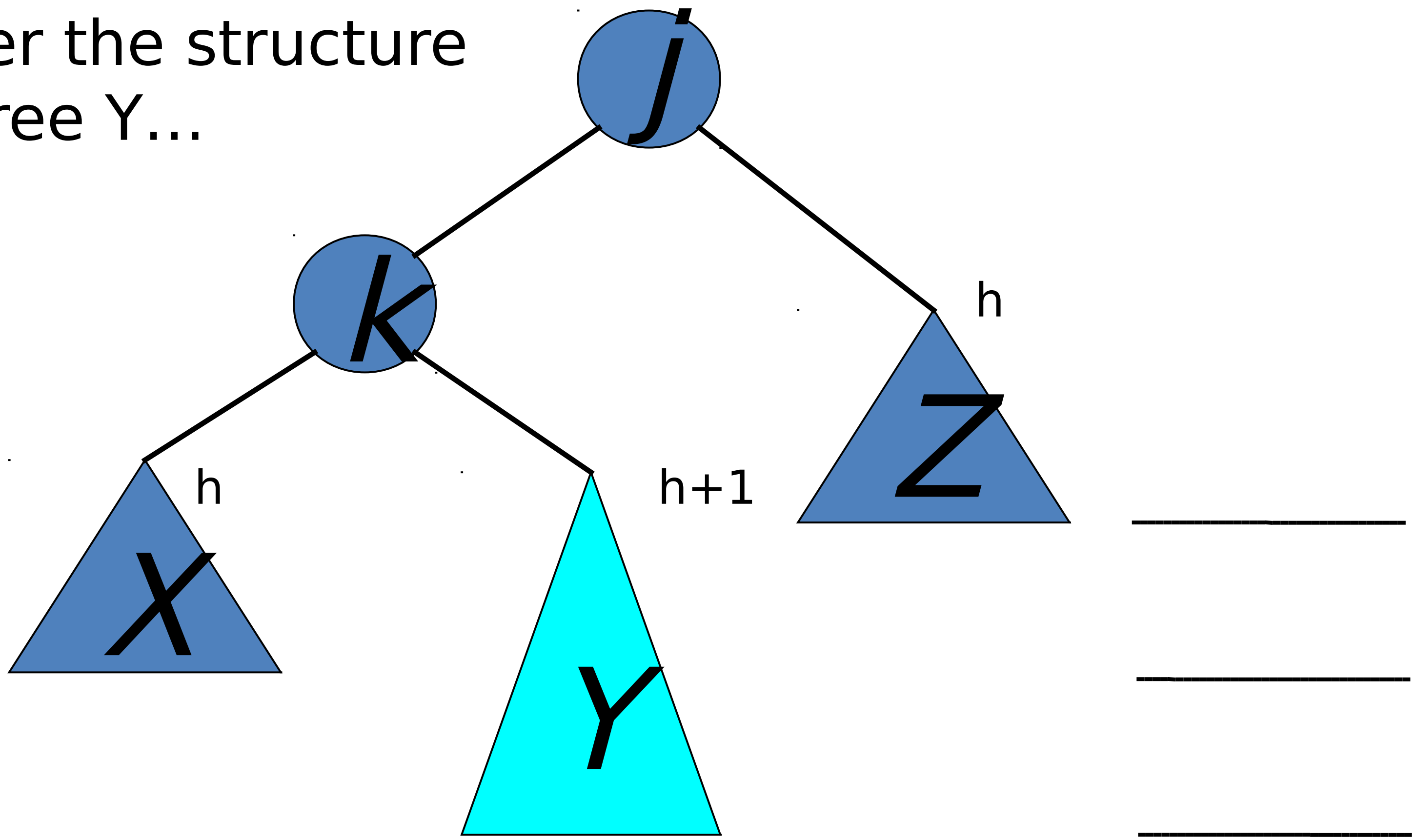
AVL Insertion: Inside Case



“Right rotation” does not restore balance… now k is out of balance

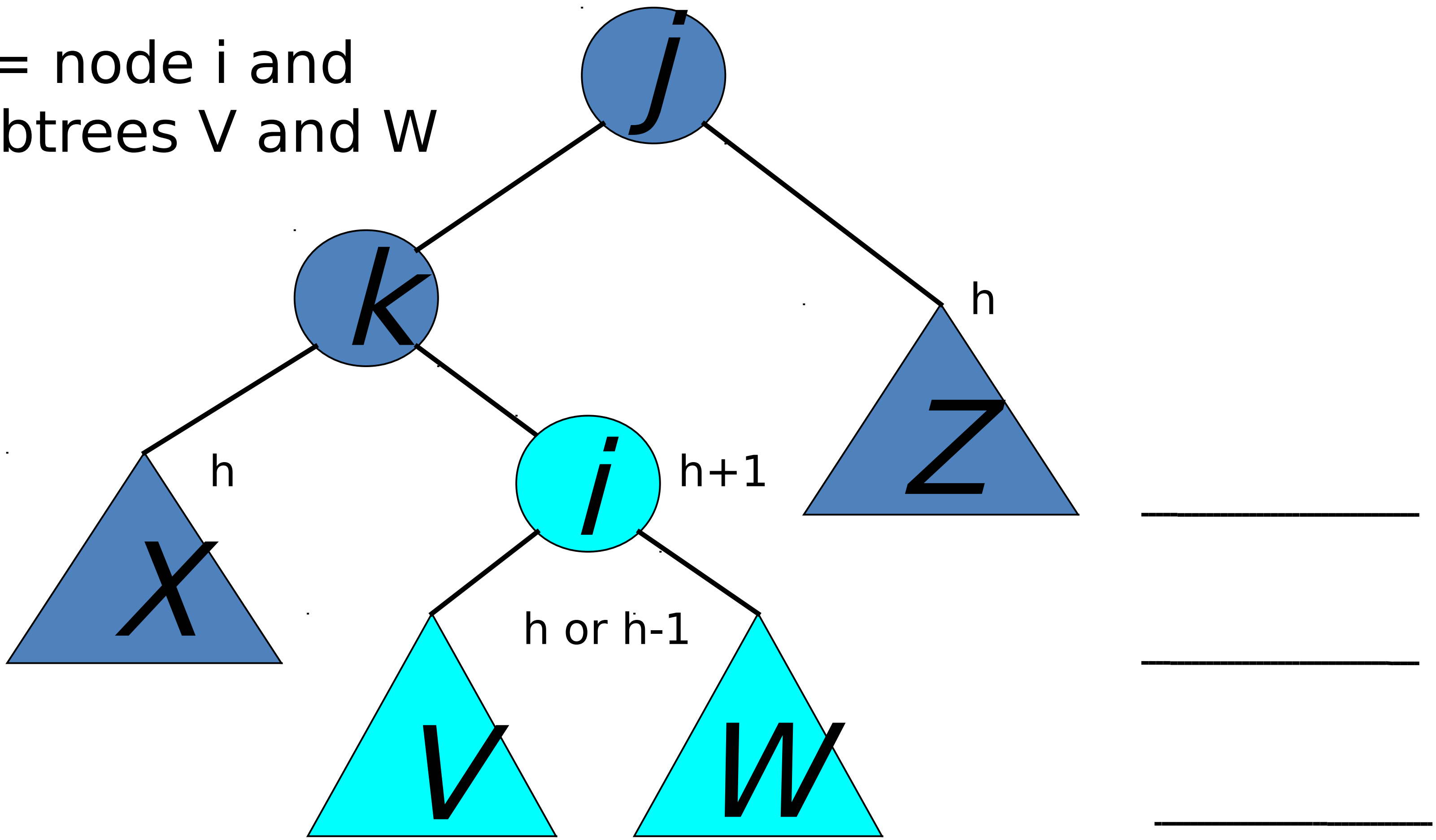
AVL Insertion: Inside Case

Consider the structure of subtree Y…

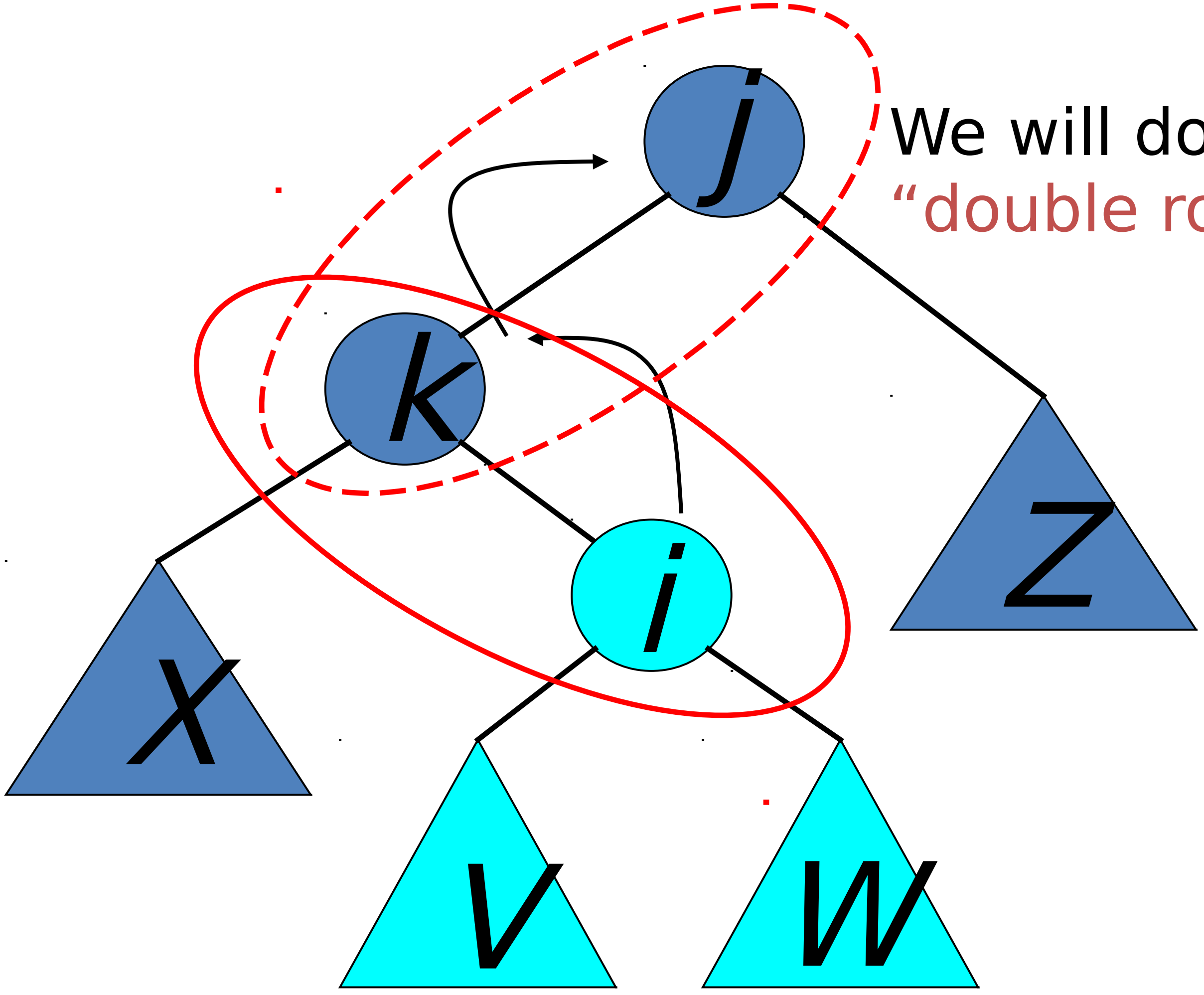


AVL Insertion: Inside Case

Y = node i and subtrees V and W

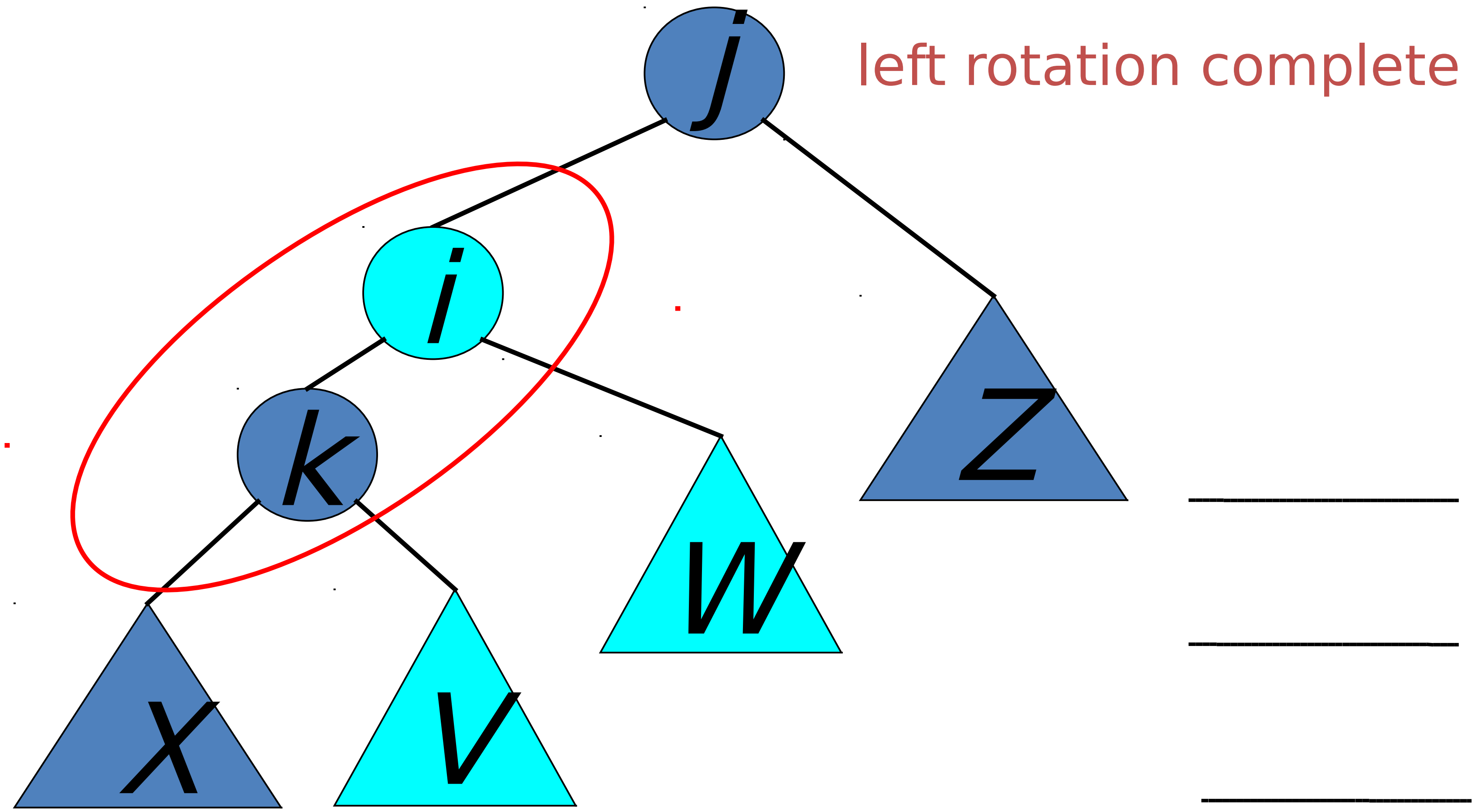


AVL Insertion: Inside Case

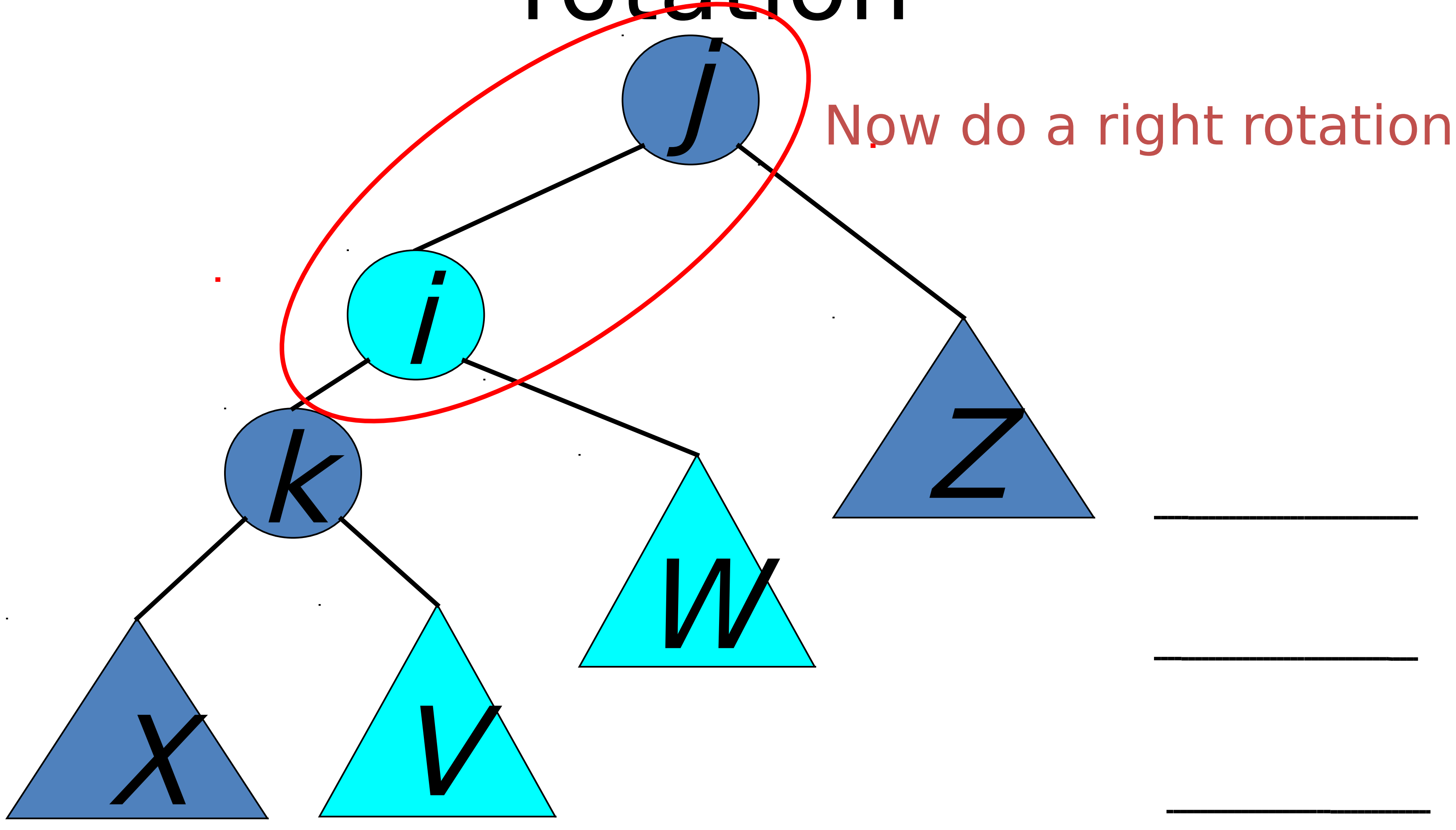


We will do a left-right "double rotation" . . .

Double rotation : first rotation



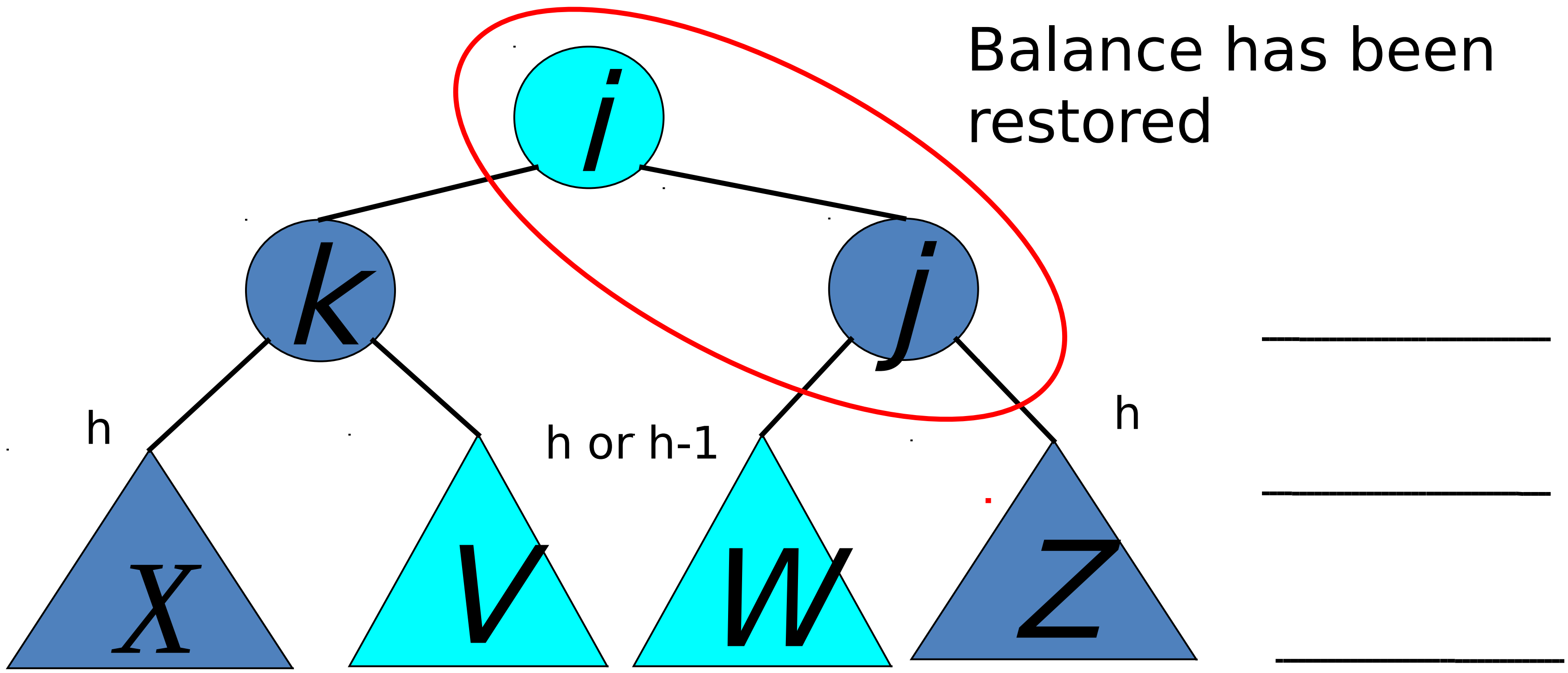
Double rotation : second rotation



Double rotation : second rotation

right rotation complete

Balance has been restored



Exercise

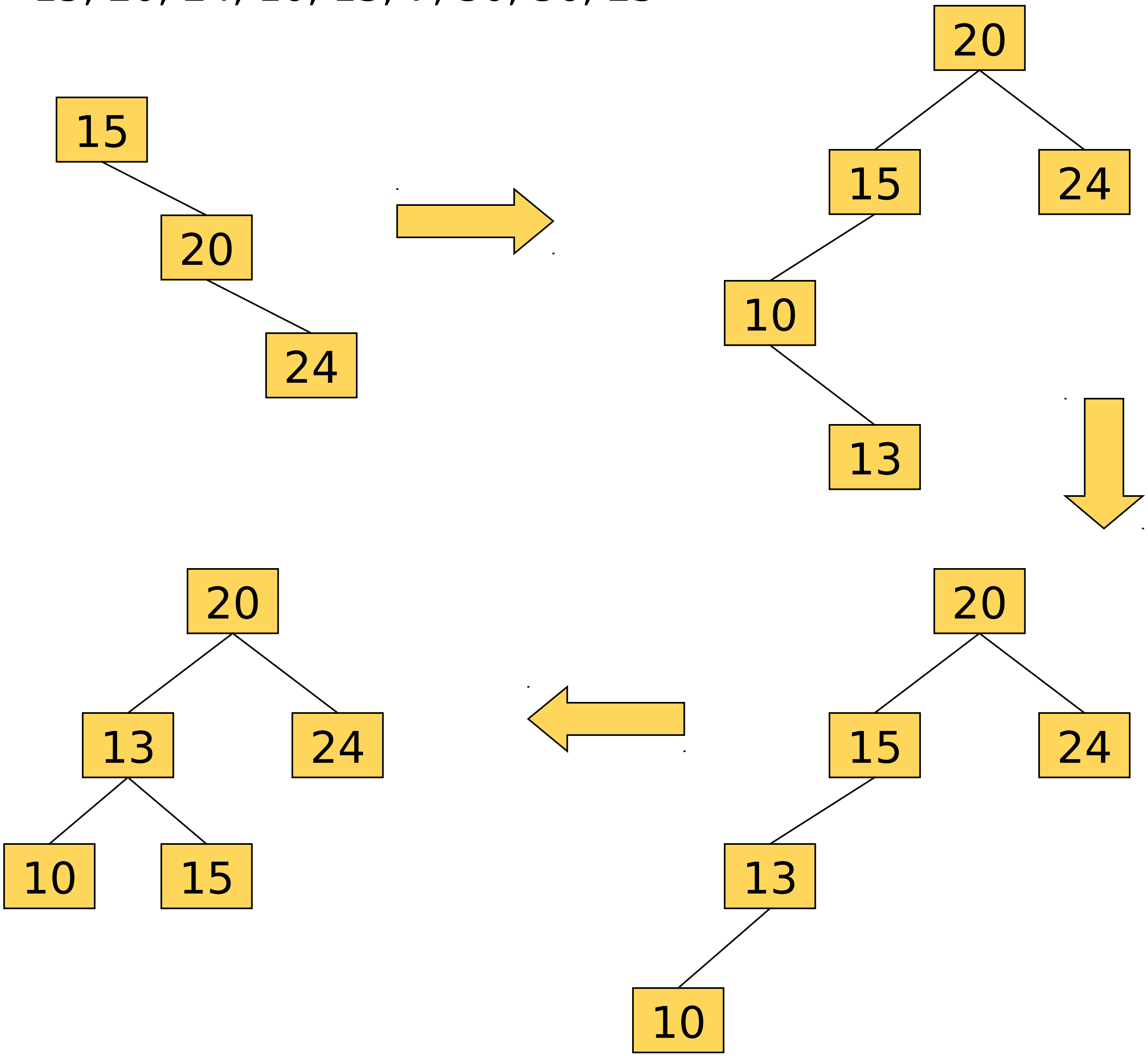
Build an AVL tree with the following values:

15, 20, 24, 10, 13, 7, 30, 36, 25

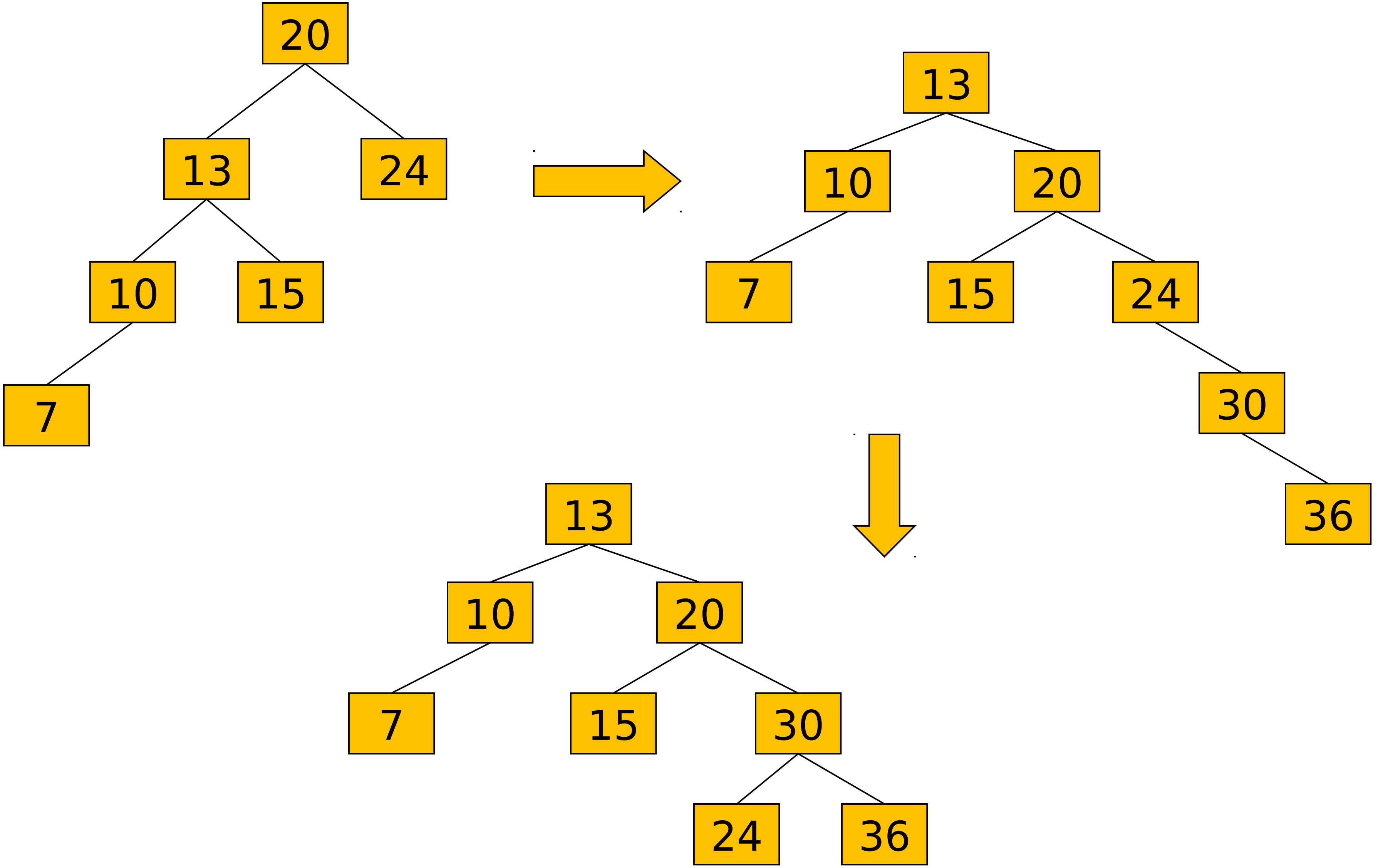
Build an AVL tree with the following values:

15, 20, 24, 10, 13, 7, 30, 36, 25

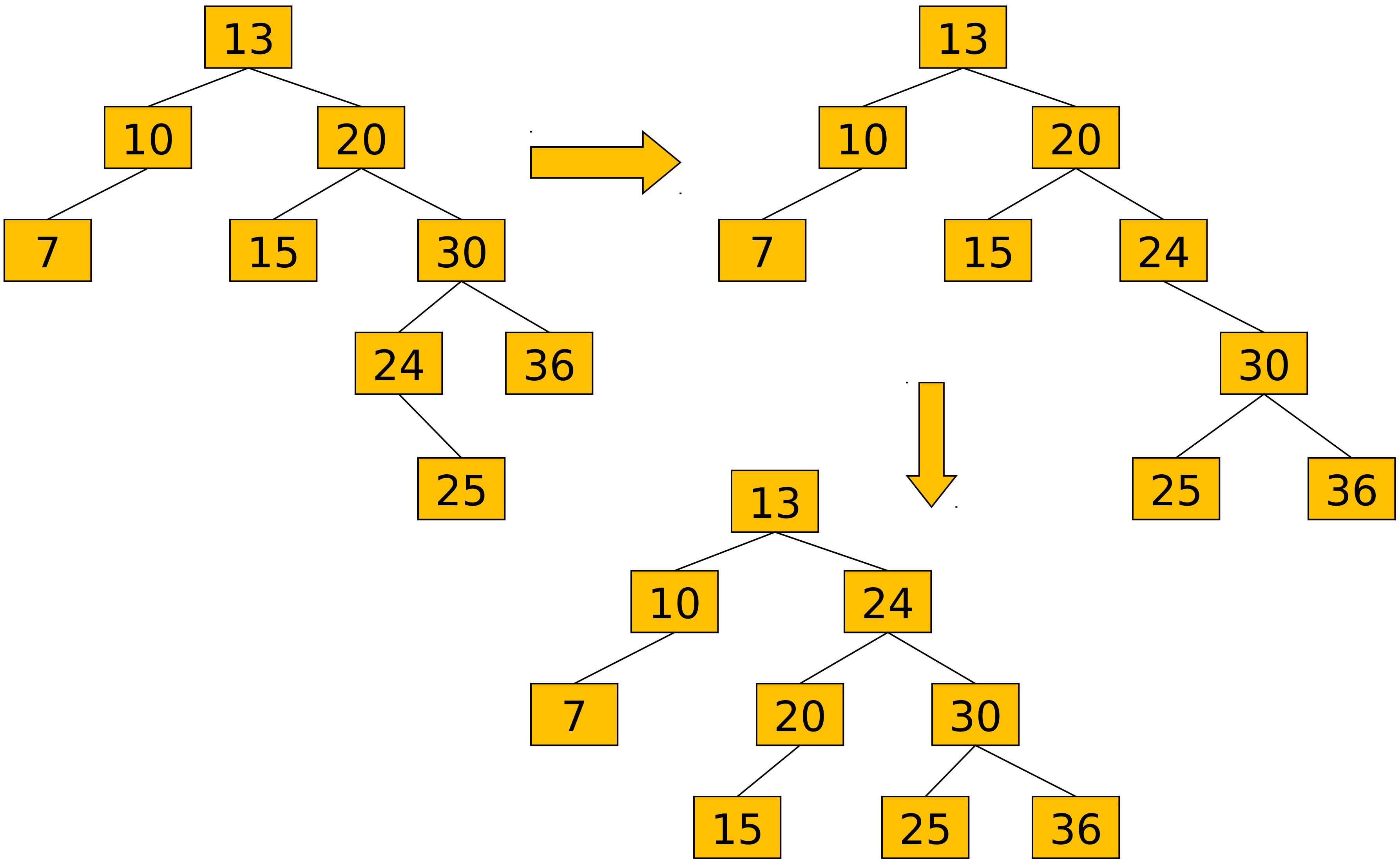
15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25

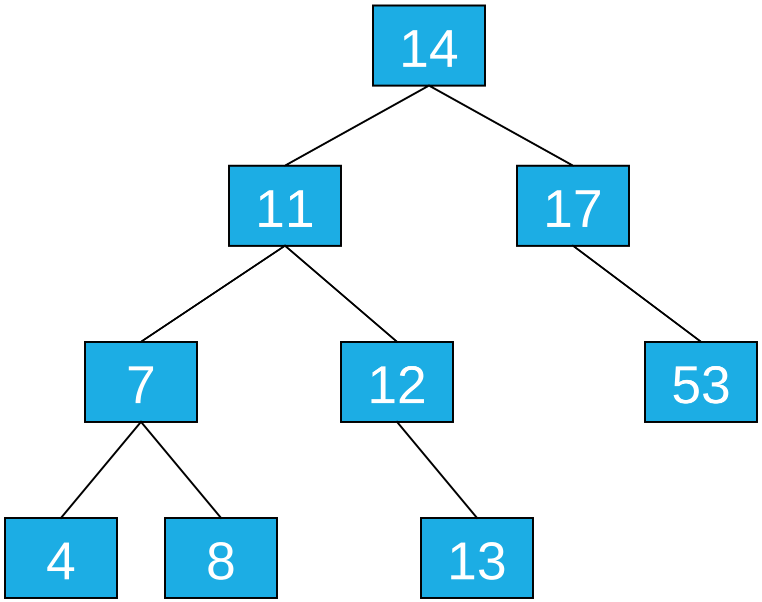


15, 20, 24, 10, 13, 7, 30, 36, 25



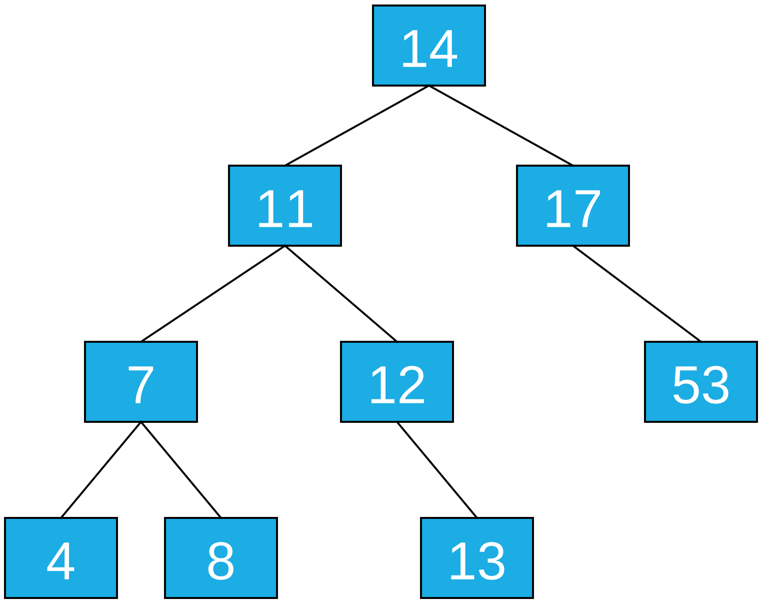
AVL Tree Example:

- **Now the AVL tree is balanced.**



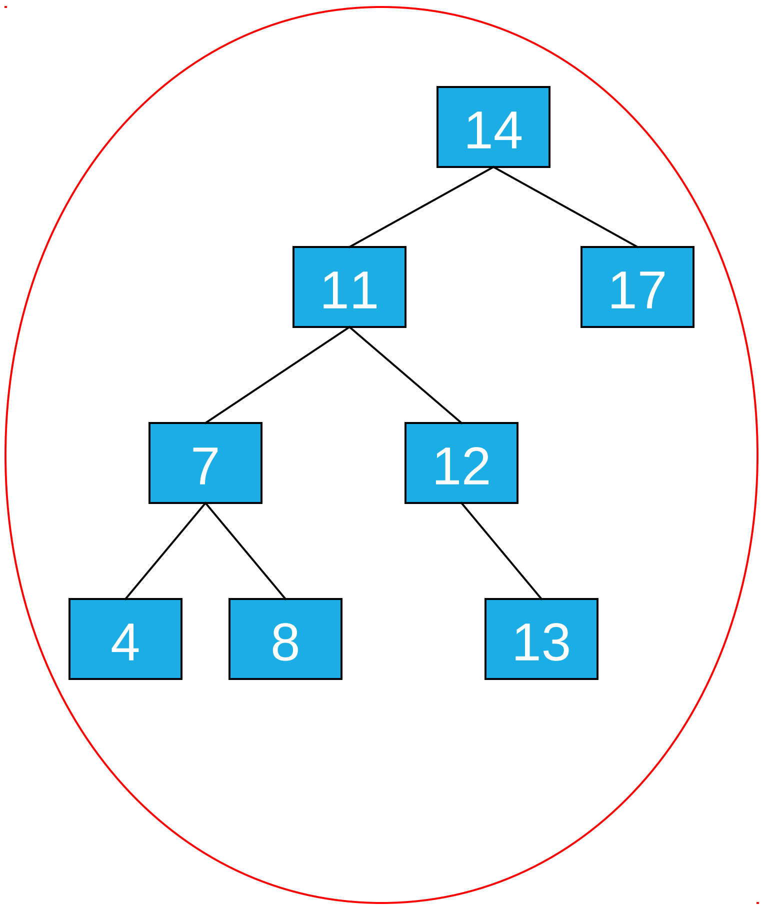
AVL Tree Example:

- **Now remove 53**



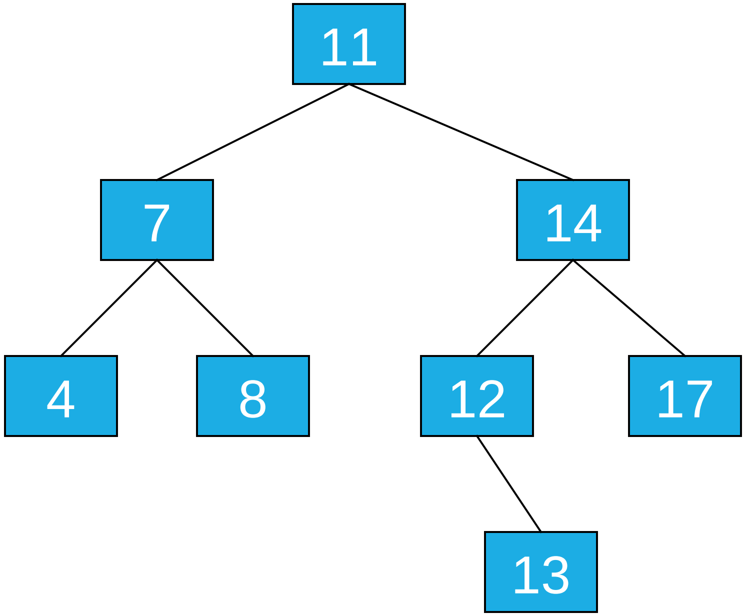
AVL Tree Example:

- **Now remove 53, unbalanced**



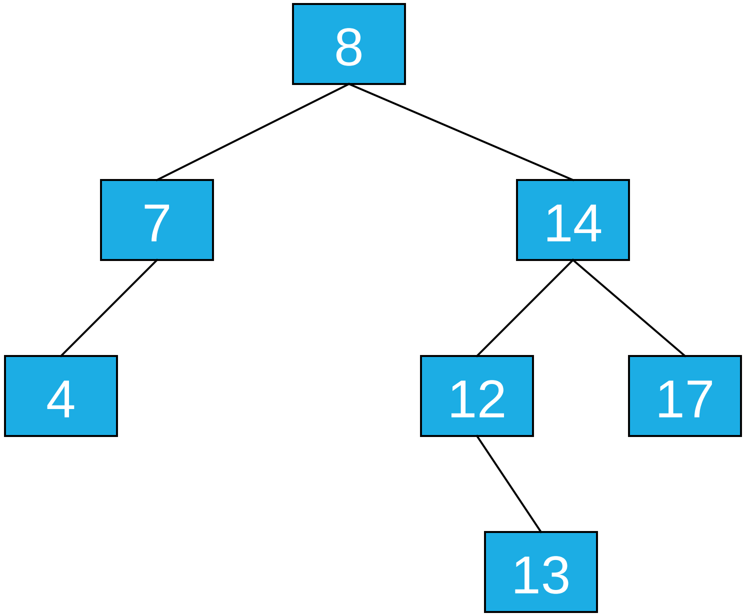
AVL Tree Example:

- **Balanced!   Remove 11**



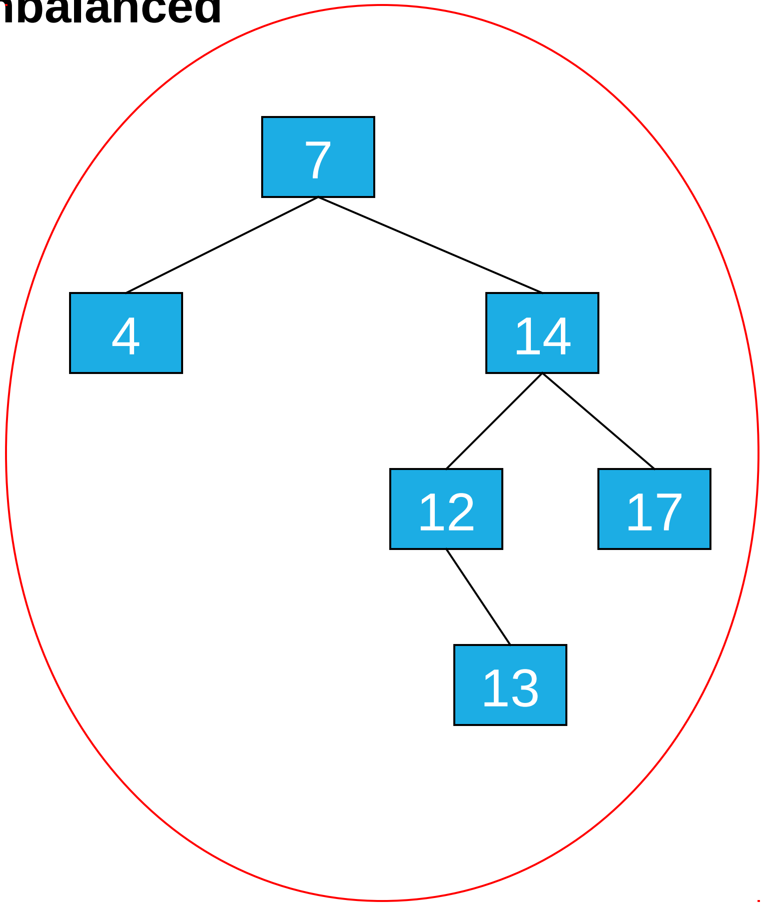
AVL Tree Example:

- **Remove 11, replace it with the largest in its left branch**



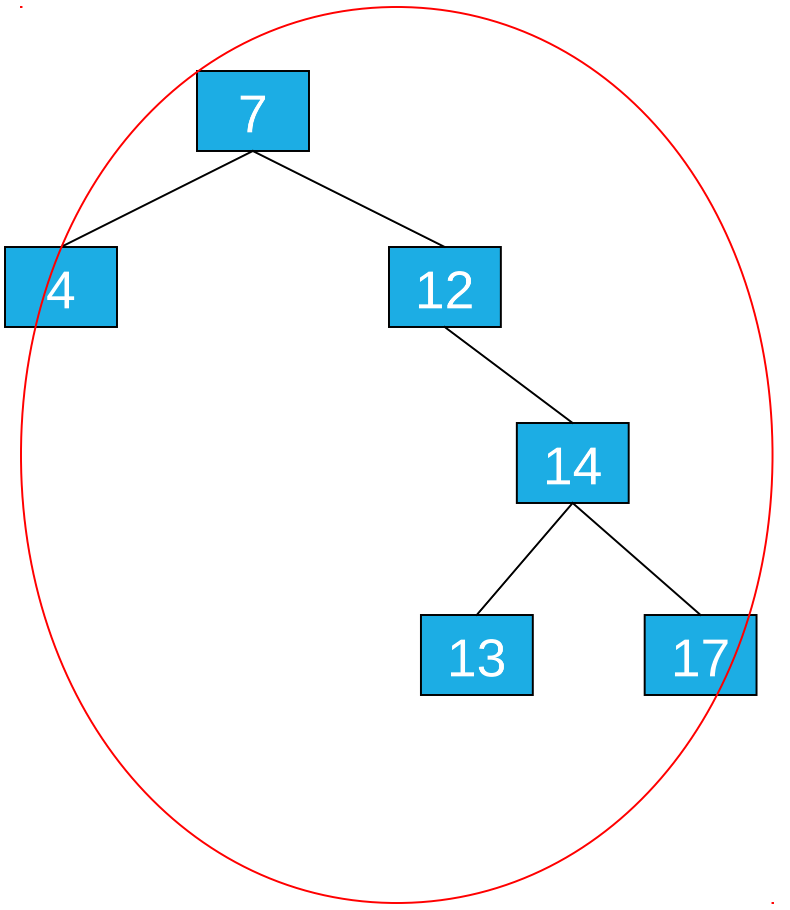
AVL Tree Example:

- **Remove 8, unbalanced**



AVL Tree Example:

- **Remove 8, unbalanced**



AVL Tree Example:

- **Balanced!!**