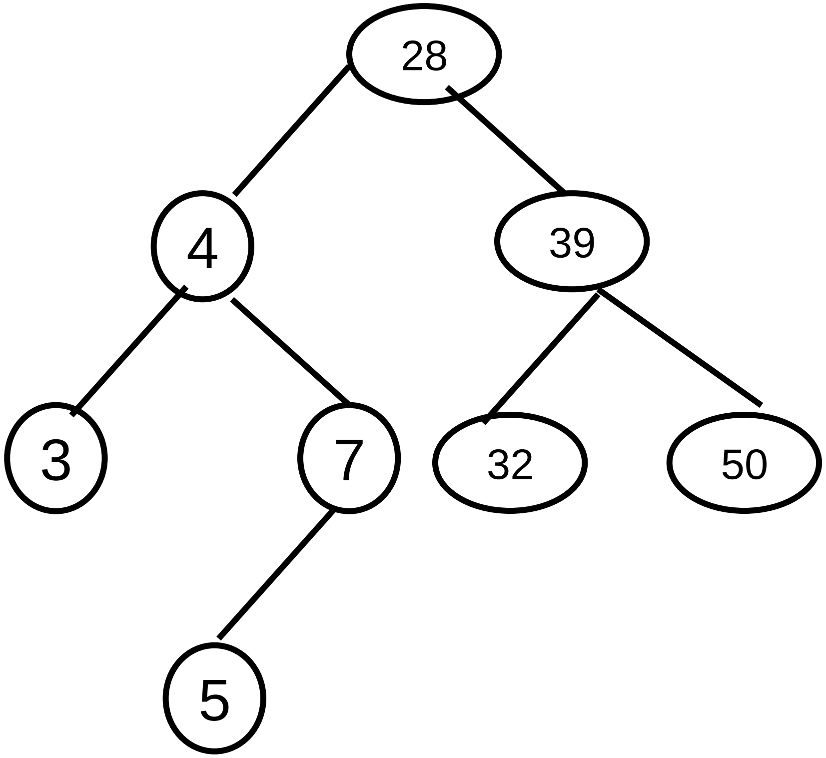
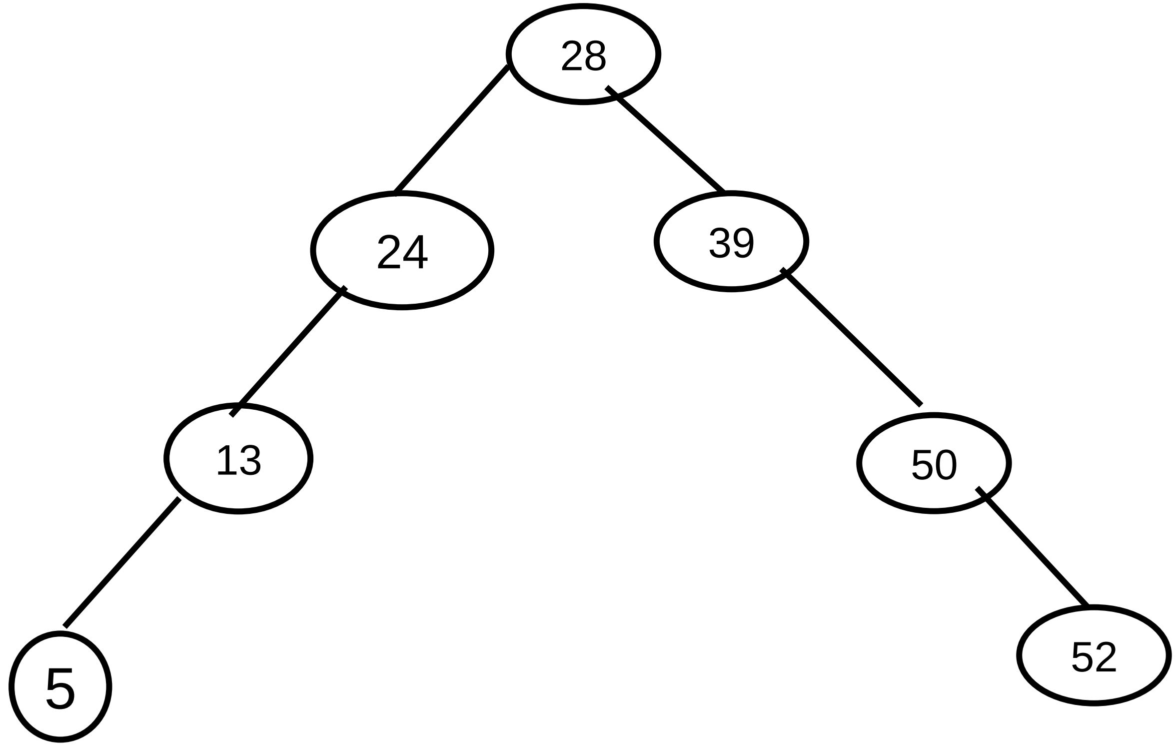
Towards Height Balanced Trees

- How can we control the height of a binary search tree?
  - should still maintain the search invariant
  - additional invariants required.
- What if the root of every subtree is the median of the elements in that subtree?
  - Difficult to maintain as median can change due to insertion/deletion.



Towards Height Balanced Trees



- Take 1: Would it suffice if we say that the root has both a left and a right subtree of equal height?
- Still, the depth of the tree is not O(log n).
- In the above tree, irrespective of values at the nodes, the root has left and right subtrees of equal height.

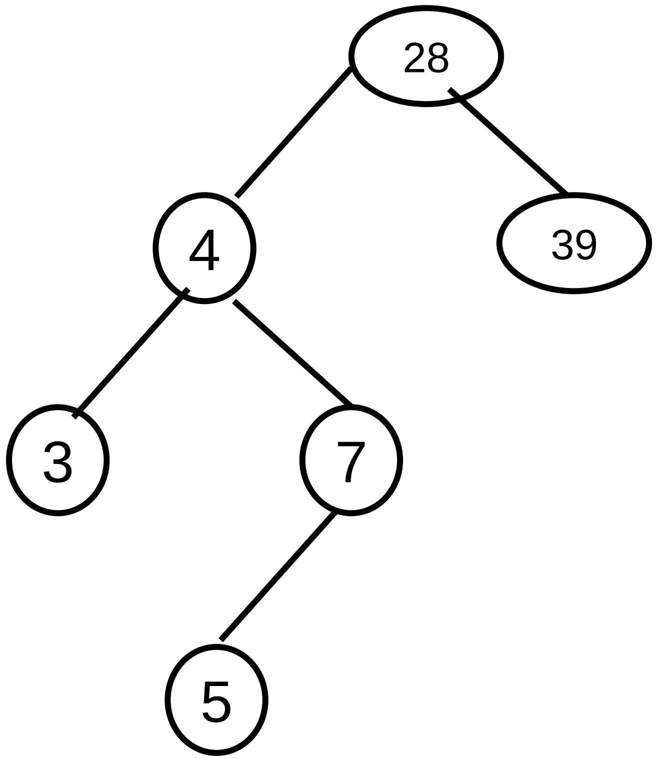
Towards Height Balanced Trees

- Our condition is too simple. Need more strict invariants.
- Consider the following modification.
- Take 2: For every node, its left and right subtrees should be of the same height.
- The condition ensures good balance, but
- The above condition may force us to keep the median as the root of every subtree.
  - Fairly difficult to maintain.

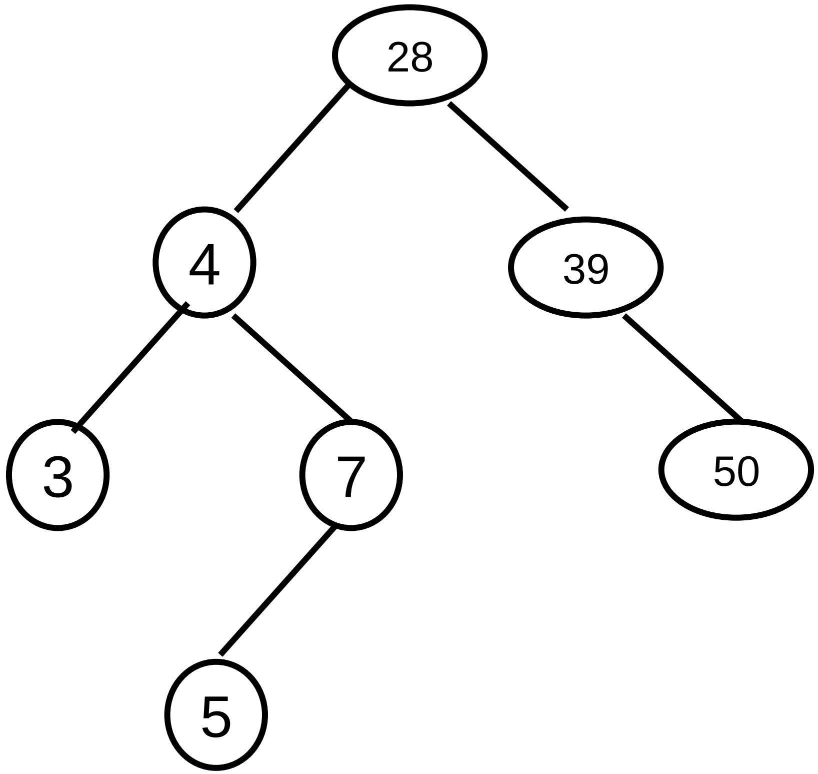
Towards Height Balanced Trees

- a small relaxation to Condition 2 works surpisingly well.
- The relaxed condition, Condition 3, is stated below.
- Height Invariant: For every node in the tree, its left and the right subtrees can have heights that differ by **at most** 1.

Example Height Balanced Trees



Not a Height Balanced Tree



Height Balanced Tree

The AVL Tree

- A binary tree satisfying the
  - search invariant, and
  - the height invariant

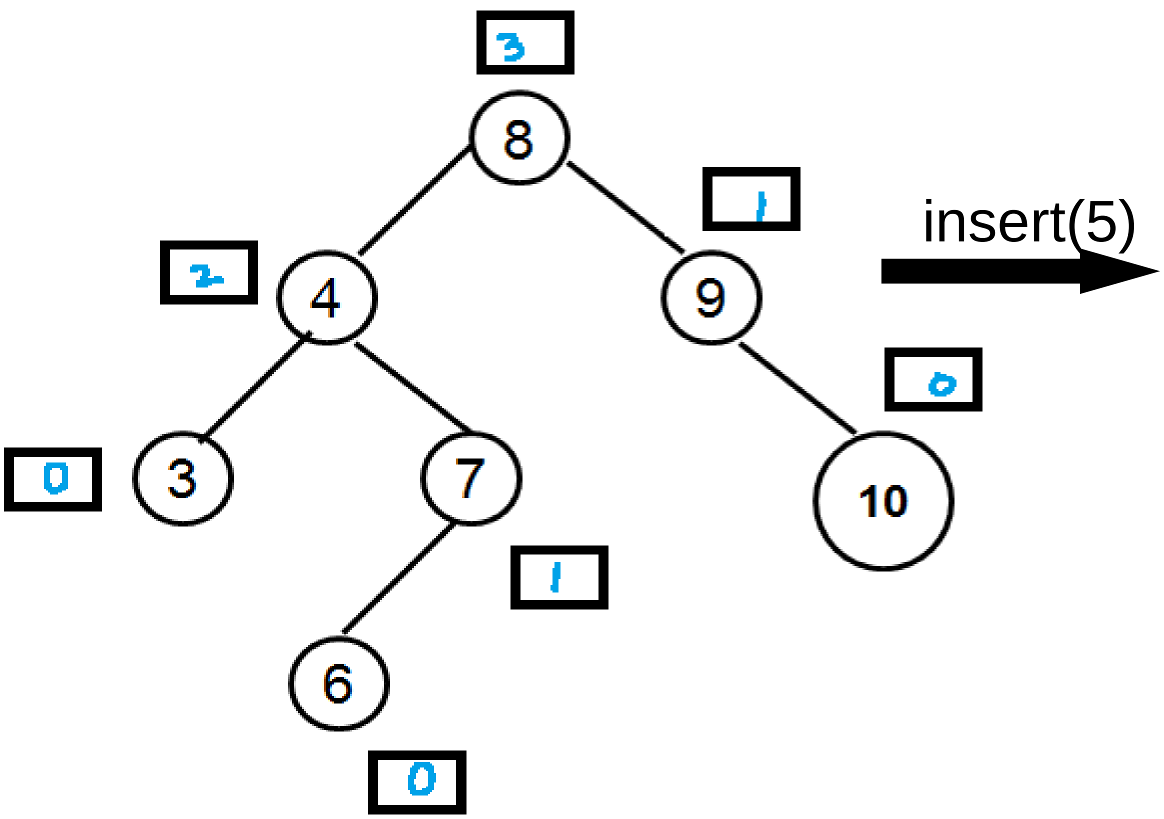
  is called an AVL tree.
- Named after its inventors, Adelson–Velskii and Landis.
- Throughout, let us define the height of an empty tree to be -1.

Operations on an AVL Tree

- An insertion/removal can violate the height invariant.
- We'll show how to maintain the invariant after an insert/remove.

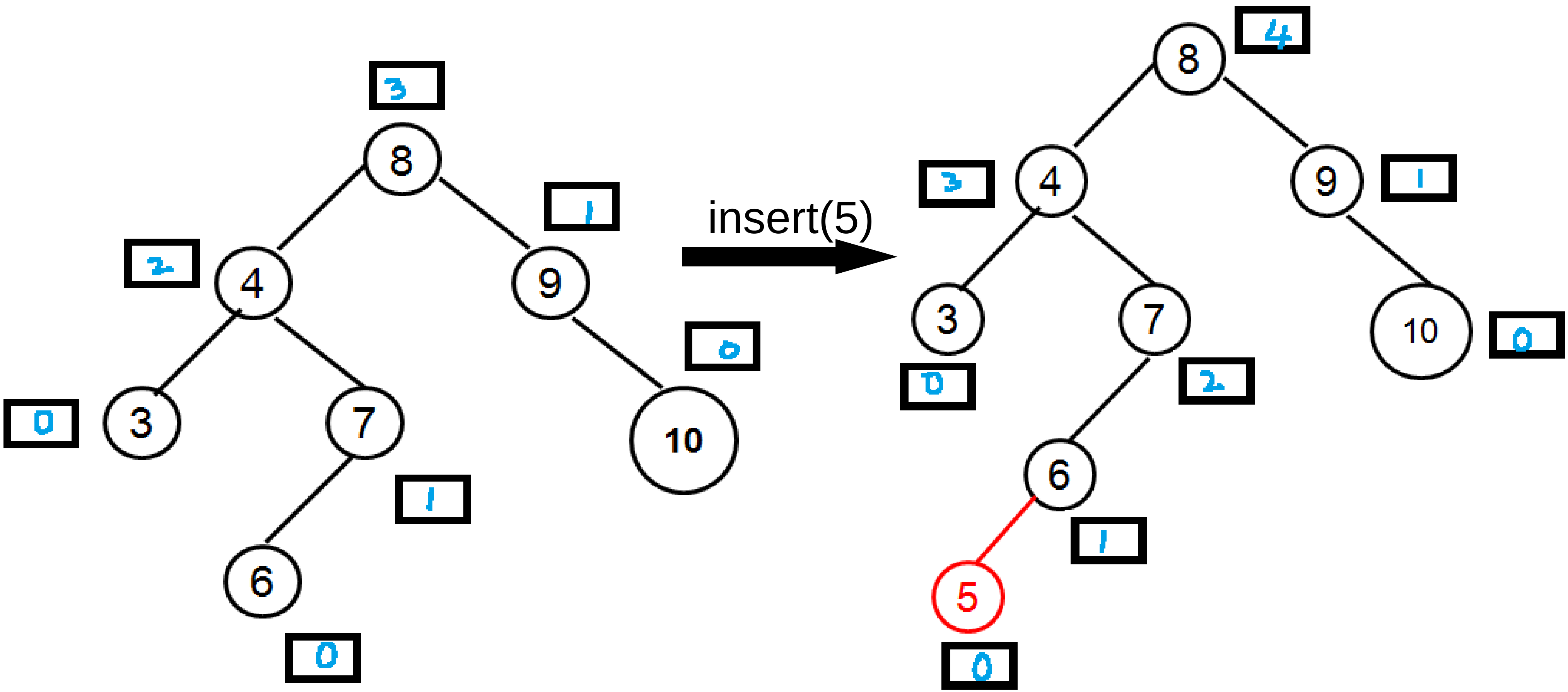
Insert in an AVL Tree

- Proceed as insertion into a search tree.
  - At least satisfies the search invariant.
- It may violate the height invariant as follows.



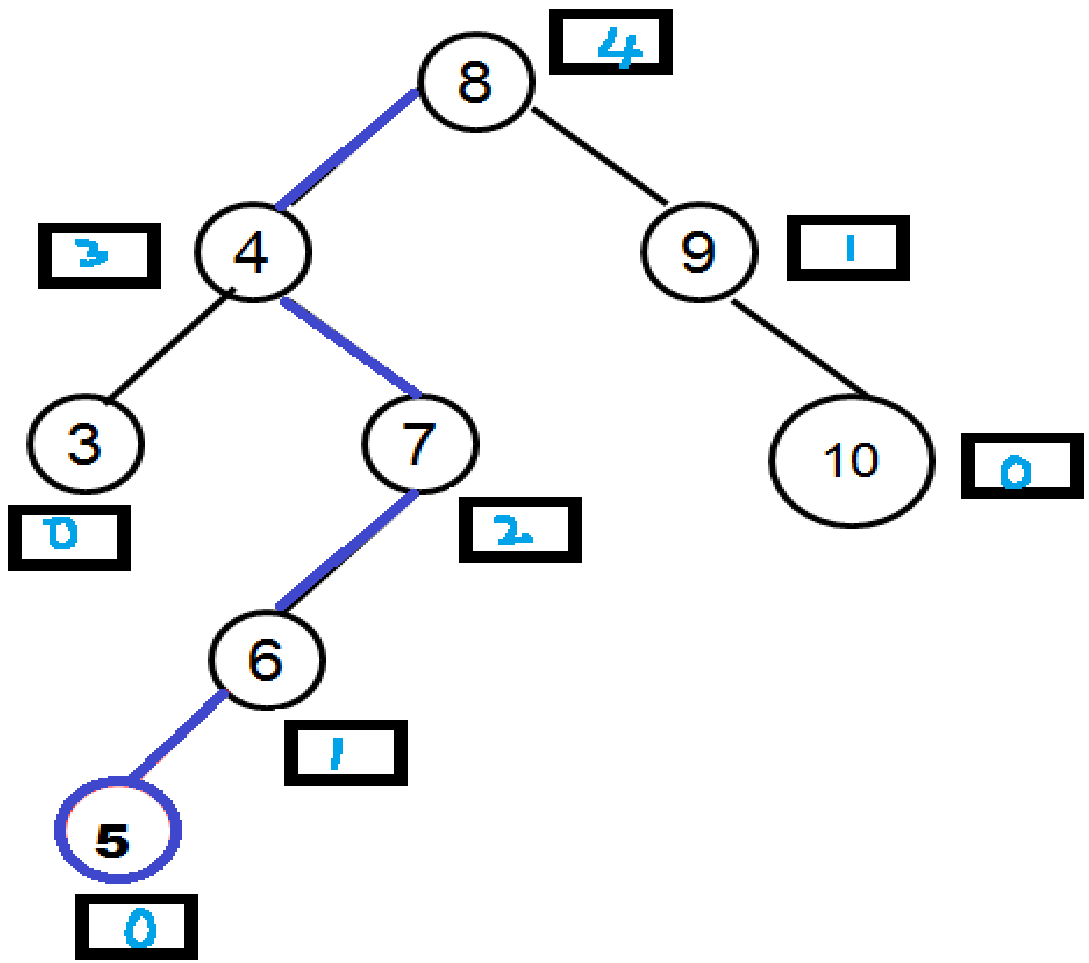
Insert in an AVL Tree

- Proceed as insertion into a search tree.
  - At least satisfies the search invariant.
- It may violate the height invariant as follows.



Insert in an AVL Tree

- After inserting as in a binary search tree, notice that all the nodes in the path along the insert may now violate the height invariant.



Insert in an AVL Tree

- How to restore balance?
- Notice that node 7 was in height balance before the insert, but now lost balance.
- Let us try to fix balance at that node.
- Node 7 has a left subtree of height 2 and a right subtree of height 0.
- If node 6 were the root of that subtree, then that subtree will have a left and right subtree of height 1 each.

Insert in an AVL Tree

- Making that change at node 7, would also fix the height violations in all other places too.
- Suggests that fixing the height violation at one node can be of great help.
- Holds true in general.
- So, need to formalize this notion.

Insert in an AVL Tree

- What is the deepest node that may violate the height invariant?
  - The leaf/deficient node at which an insert happens?
  - Or, the parent of such a node?
  - Or some node higher up?
  - Why?

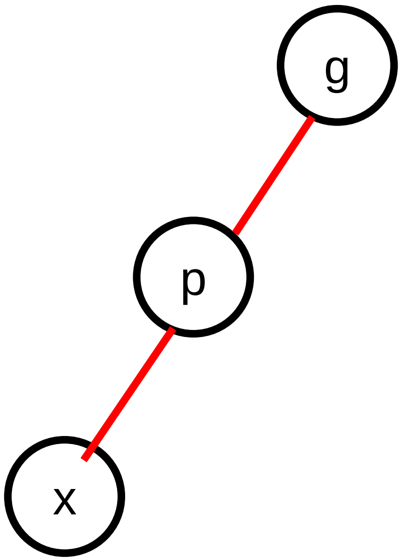
Insert in an AVL Tree

- Let node t be the deepest node that violates the height condition.
- Such a violation can occur due to the following reasons:
  - An insertion into the left subtree of the left child of t.
  - An insertion into the right subtree of the left child of t.
  - An insertion into the left subtree of the right child of t, and
  - An insertion into the right subtree of the right child of t.

Insert into an AVL Tree

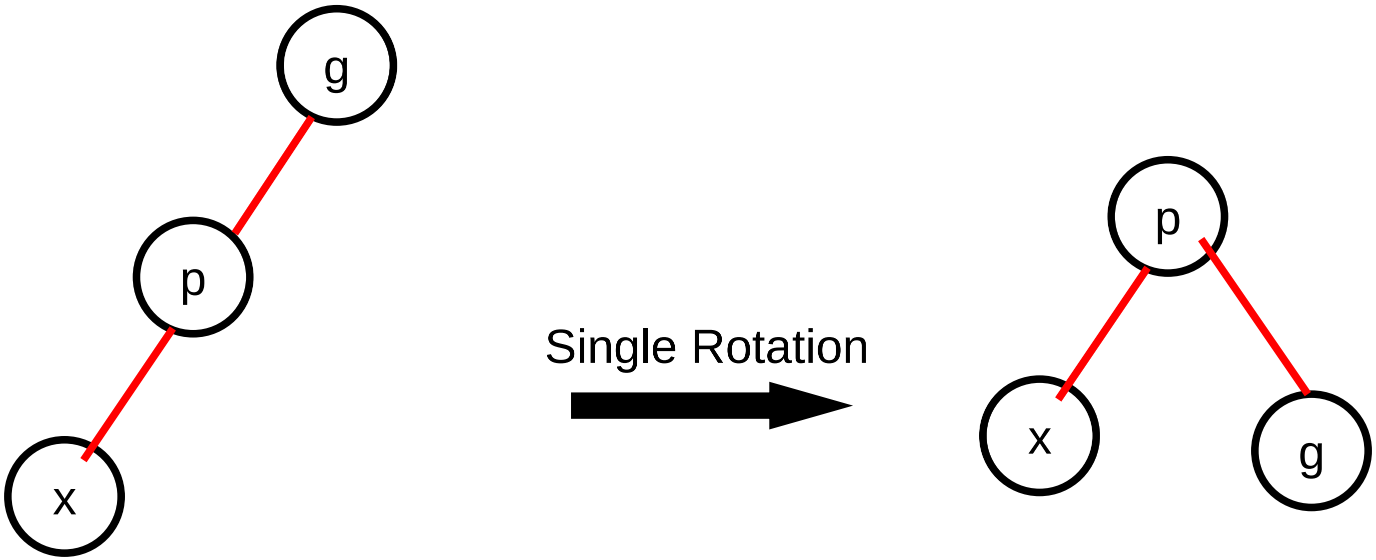
- Notice that cases 1 and 4 are symmetric.
- Similarly, cases 2 and 3 are symmetric.
- So, let us treat cases 1 and 2.

Insert into an AVL Tree



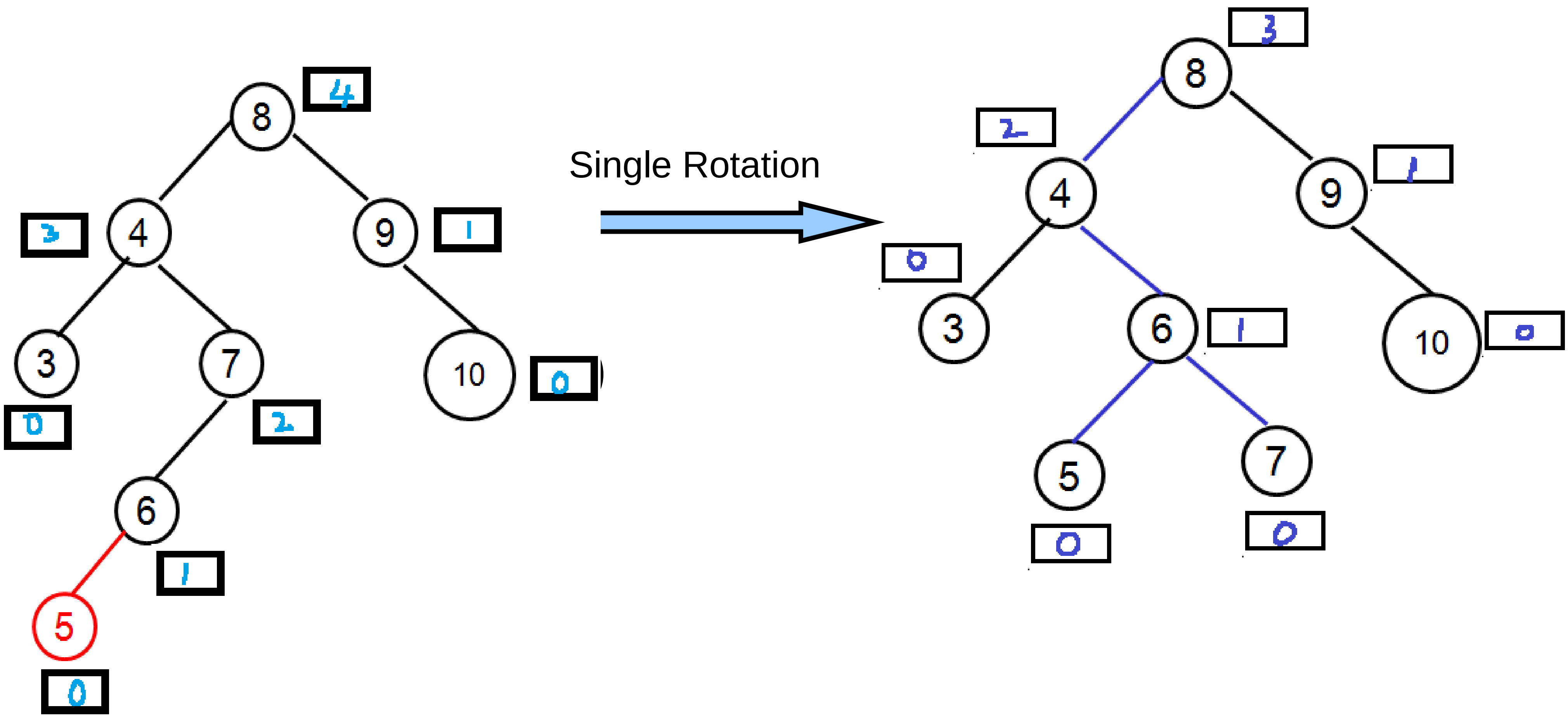
- Recall the earlier fix at node 7.
- We call that operation a single rotation.
  - In a single rotation, we consider a node x, its parent p, and its grandparent g.
  - Let x be a left child of p, and p a left child of g.
  - After rotation, we make p the root of the subtree.
  - To satisfy the search invariant, g should now be the right child of p and x the left child of p.

Single Rotation Example



- Recall the earlier fix at node 7.
- We call that operation a single rotation.
  - In a single rotation, we consider a node x, its parent p, and its grandparent g.
  - Let x be a left child of p, and p a left child of g.
  - After rotation, we make p the root of the subtree.
  - To satisfy the search invariant, g should now be the right child of p and x the left child of p.

Single Rotation Example

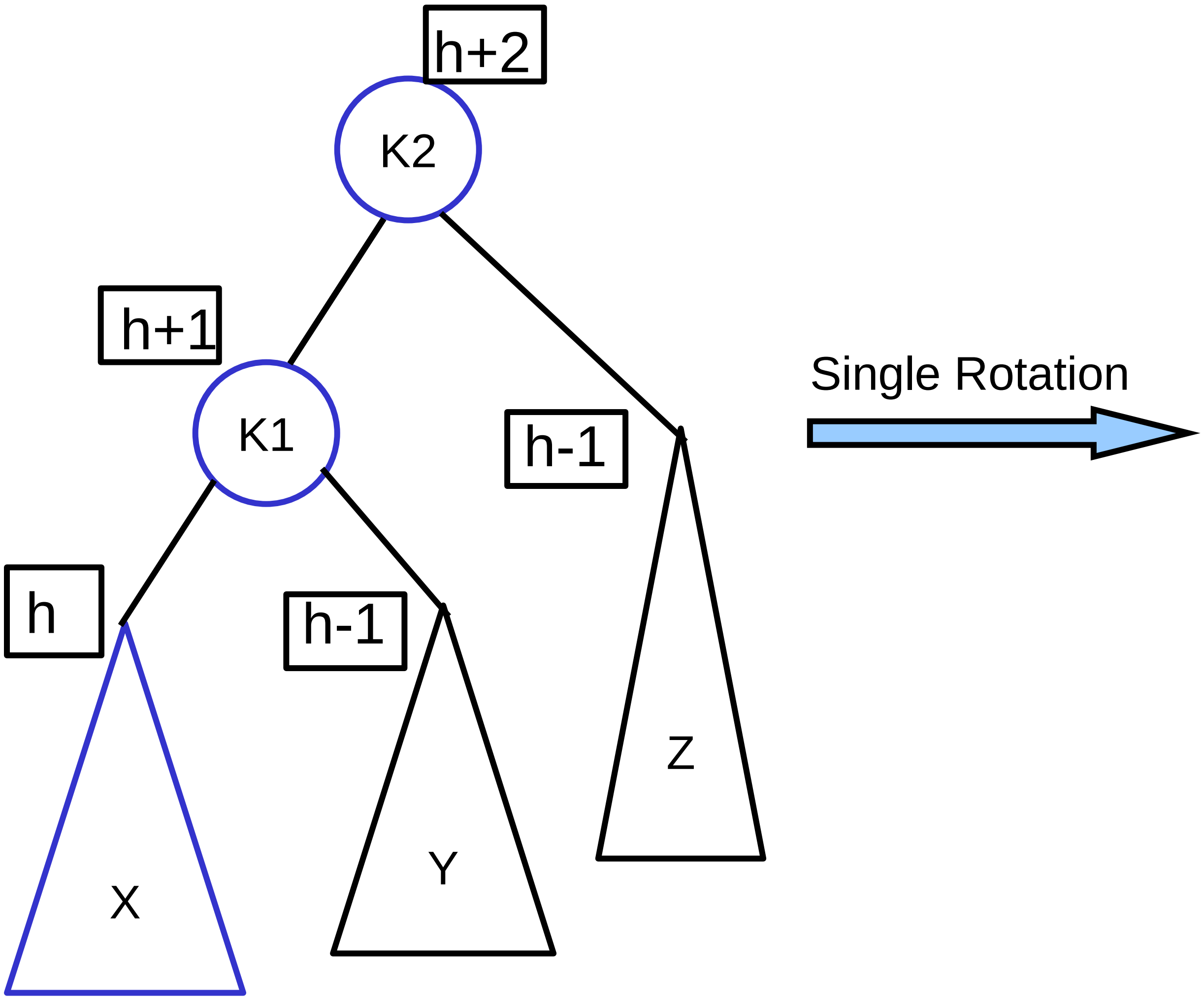


Practice Problem

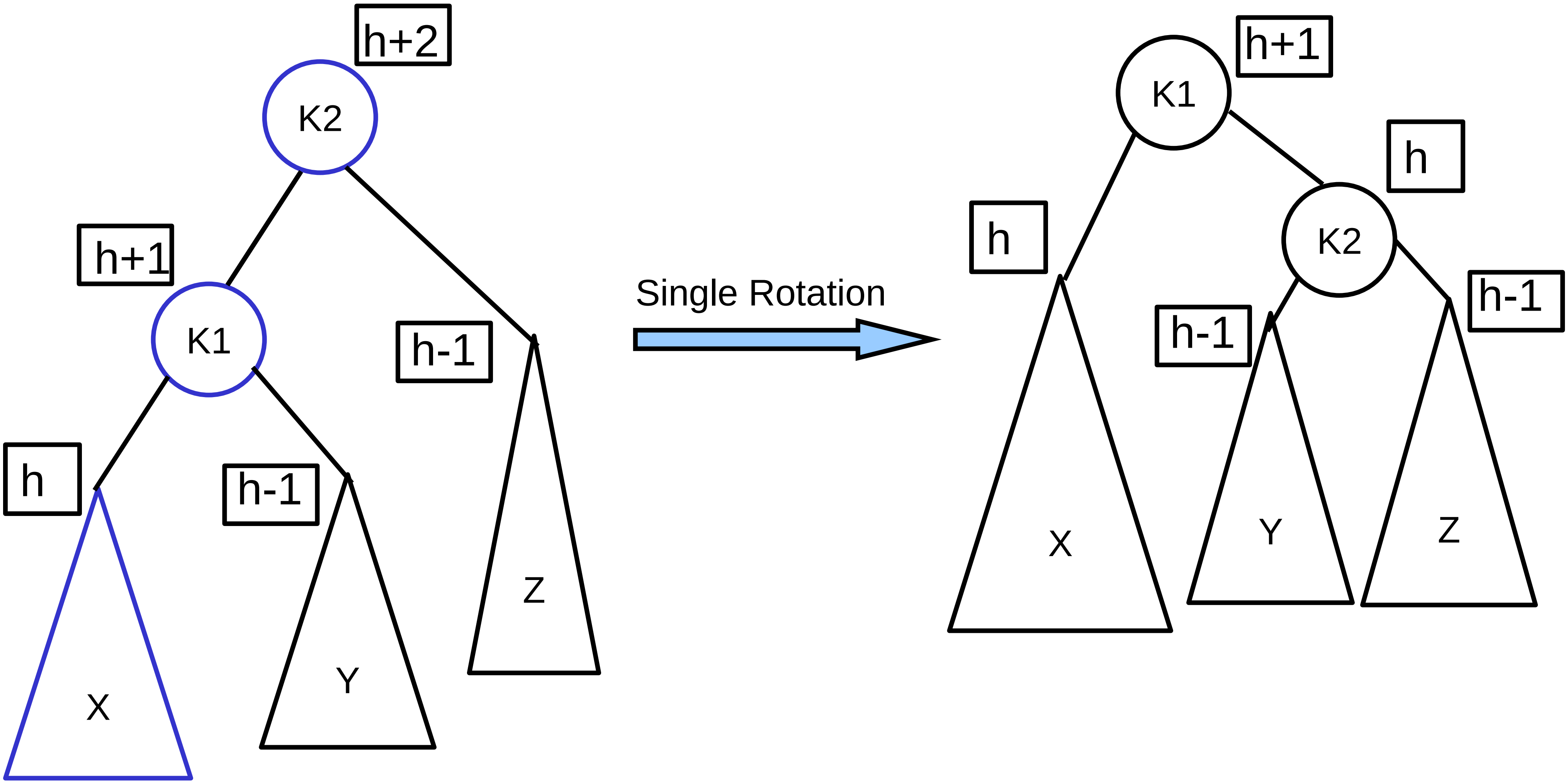
- Insert the following values in that order into an initially empty AVL tree.

12, 16, 19, 8, 6, 22, 26

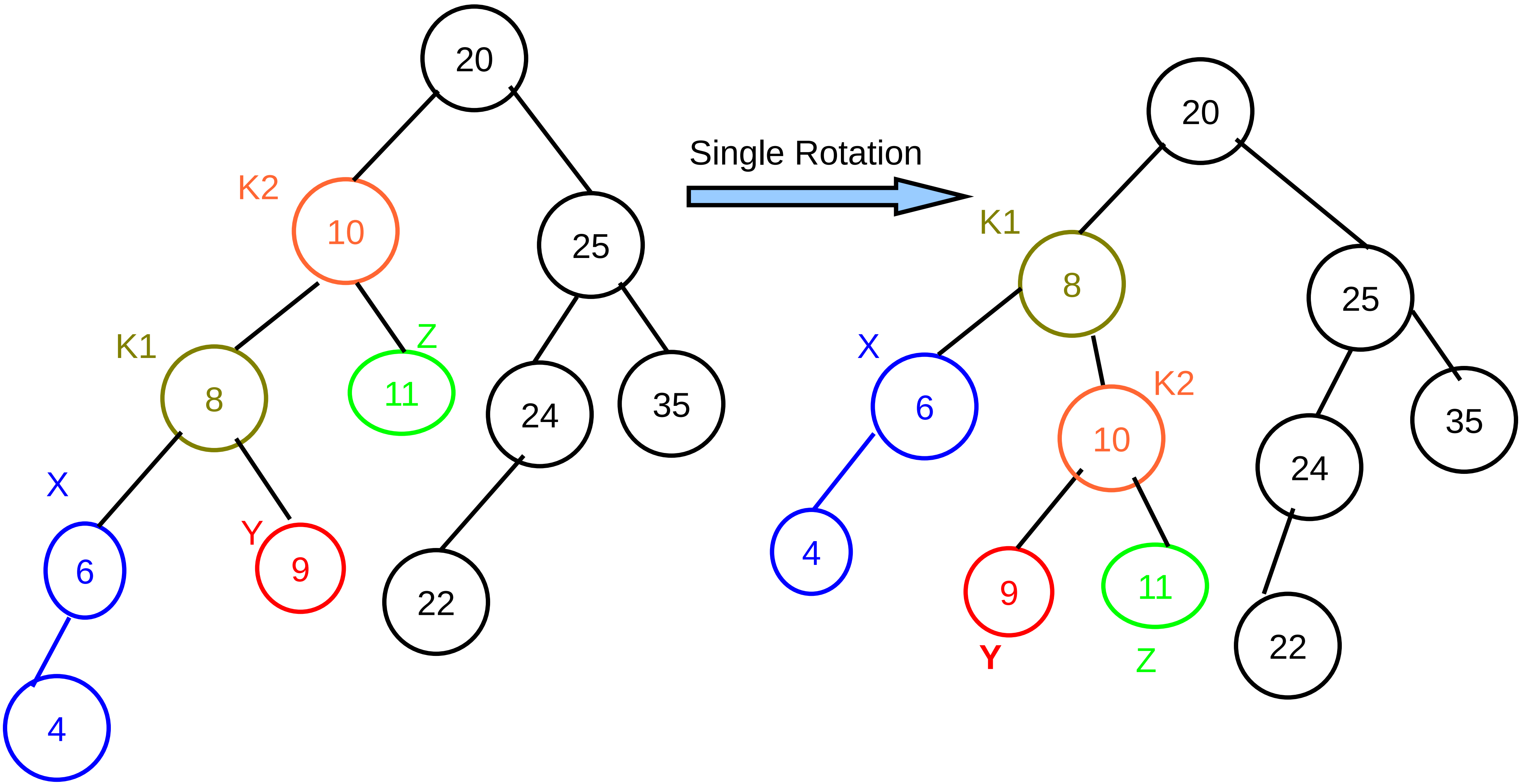
Single Rotation – Generalization



Single Rotation – Generalization



Single Rotation – Example

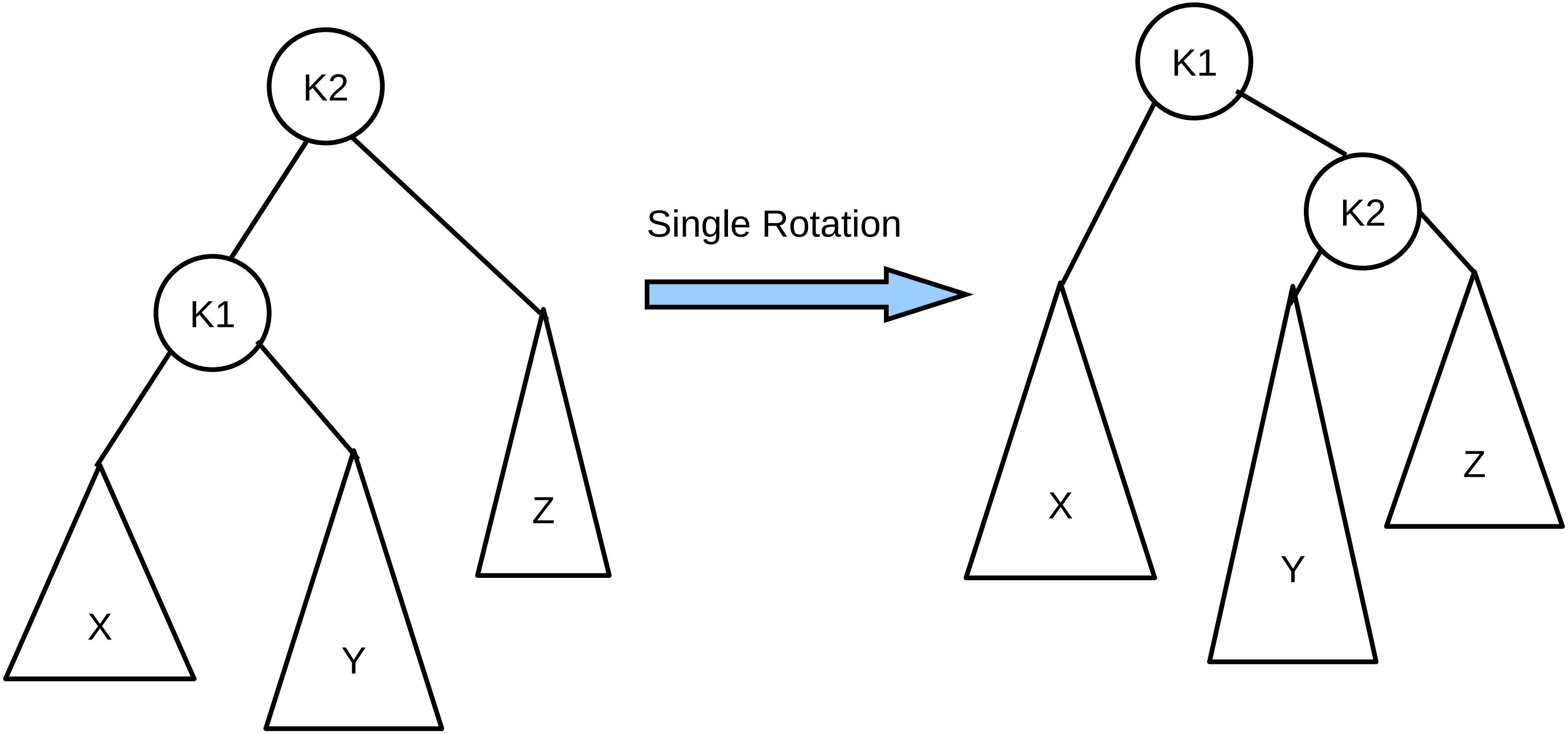


Single Rotation

- Why does it help?
- If K2 is out of balance after the insert, the height difference between Z and K1 is 2.
  - Why can't it be more than 2?
- Now, the height of Z increases by 1 after the rotate
- Also, the height of X and Y decrease by 1.
- So, the subtree at K1 now has the same height as K2 had before the insert.

Case 2 of the Insert

- Single rotation may not help here.

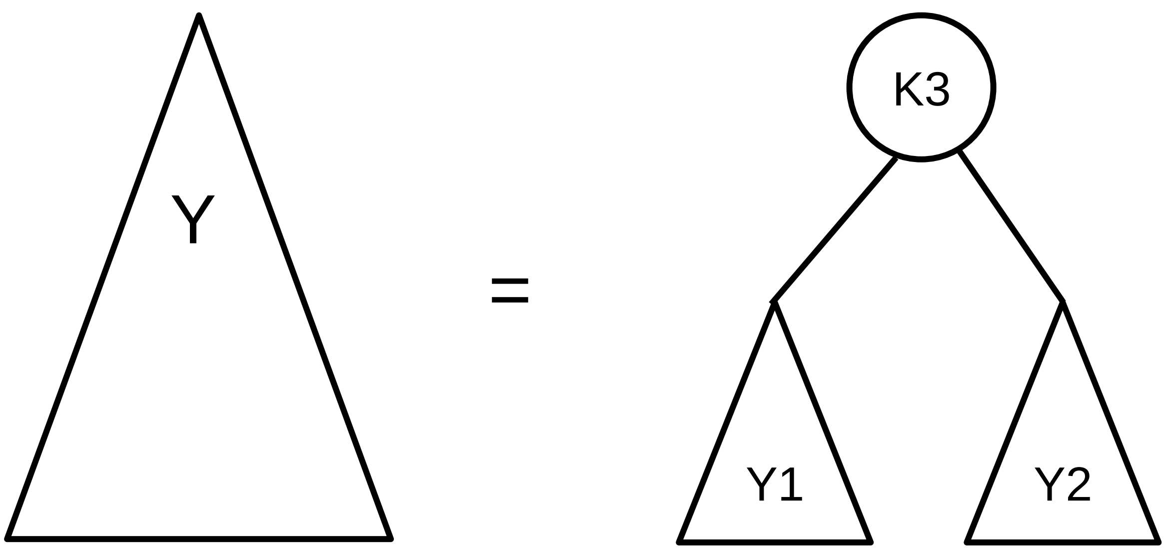


Case 2 of Insert

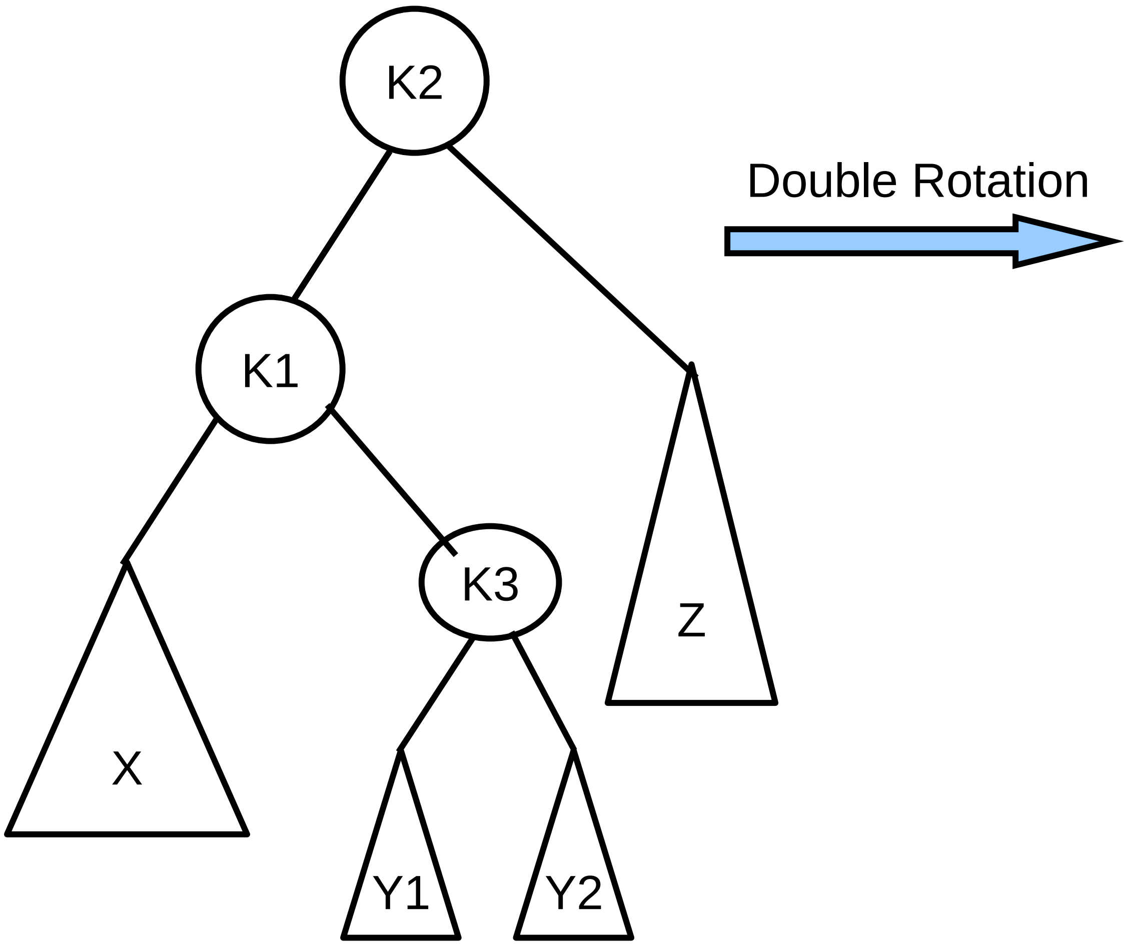
- Why single rotation did not help?
- Height of Y increased, resulting in increase of height of K2.
- After rotate also, height of Y is same as earlier.
- So, does not help fix the height imbalance.

Case 2 of Insert

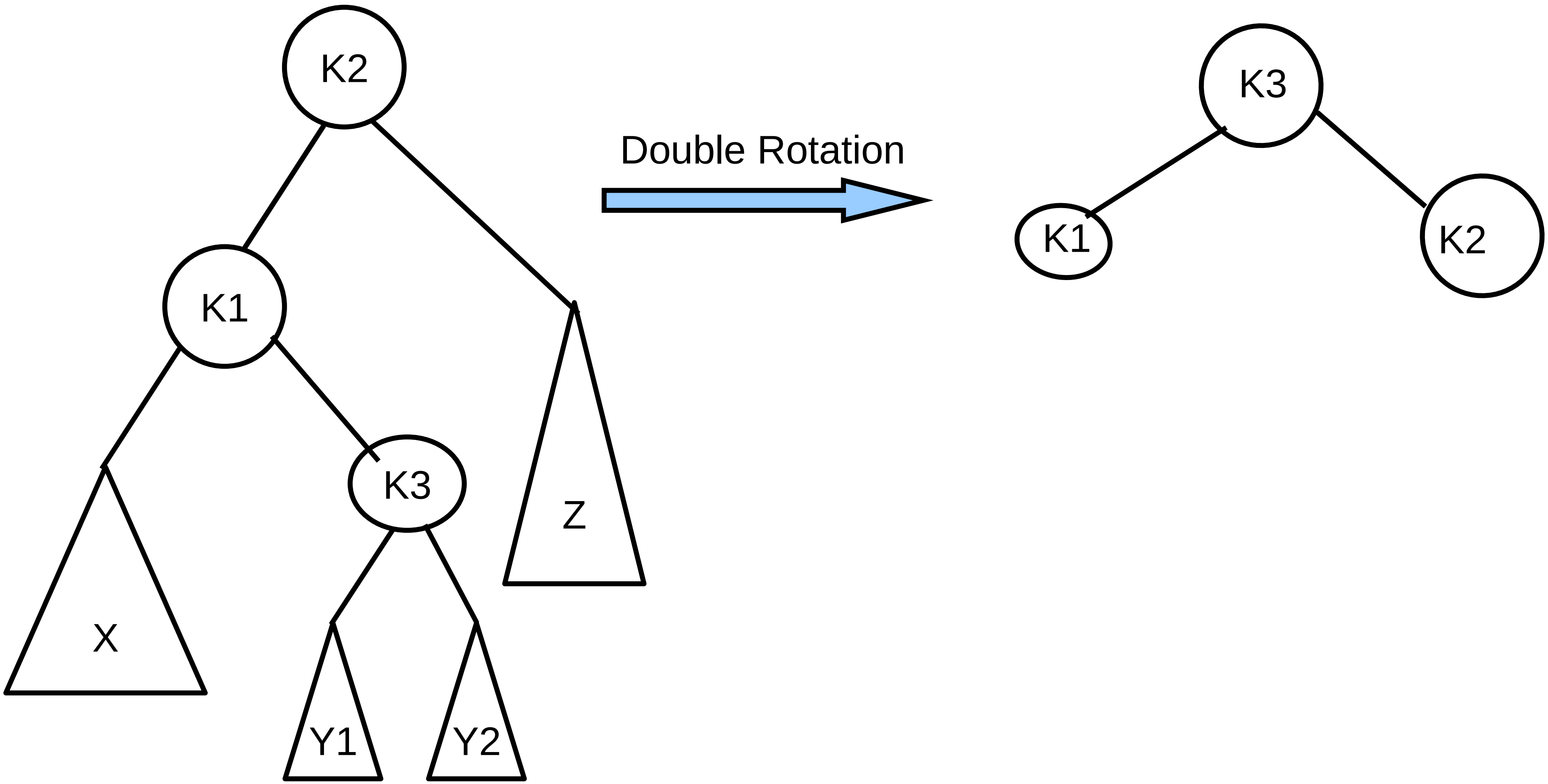
- Need more fixes.
- Idea : Y should reduce height by 1.
- We hence introduce double rotation.
- Would be helpful to view as follows.



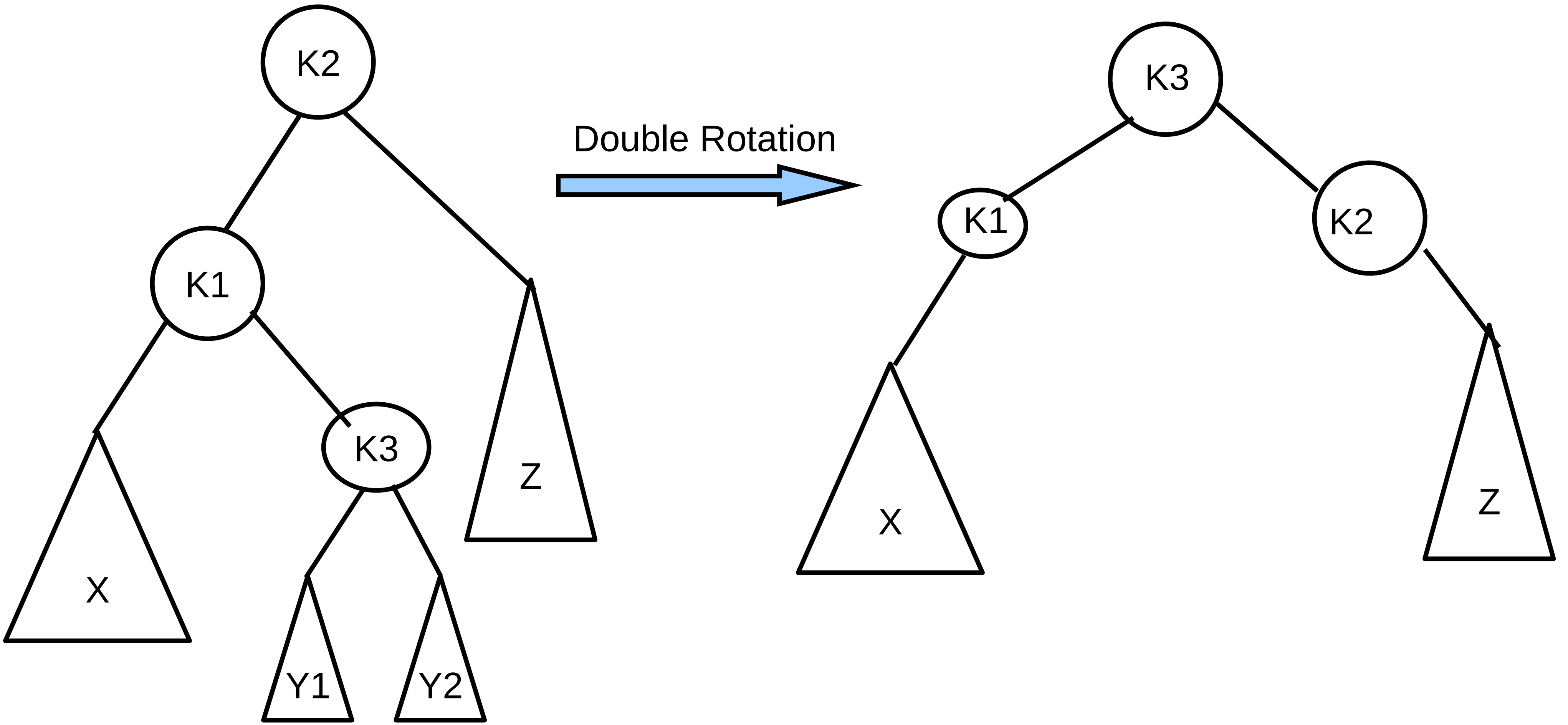
Double Rotation Generalization



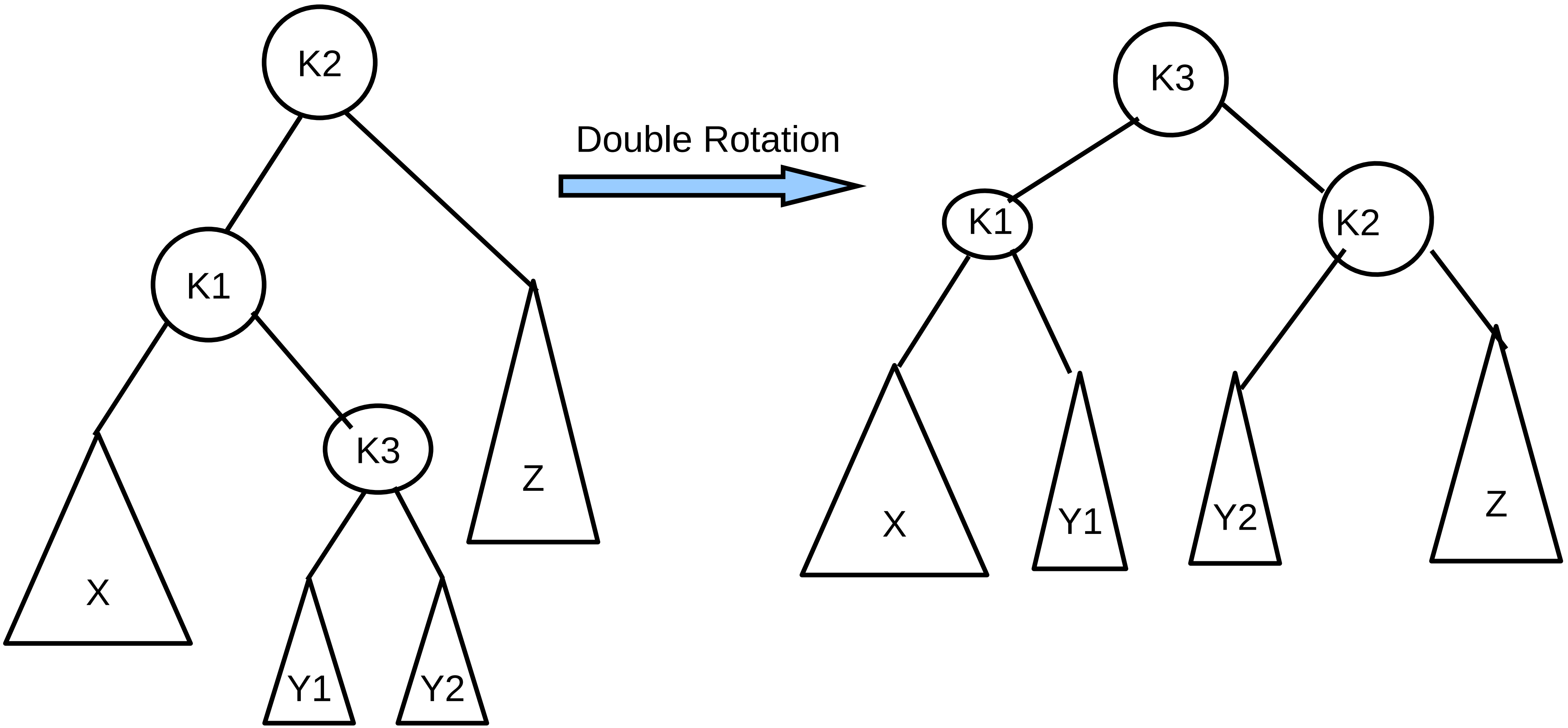
Double Rotation Generalization



Double Rotation Generalization



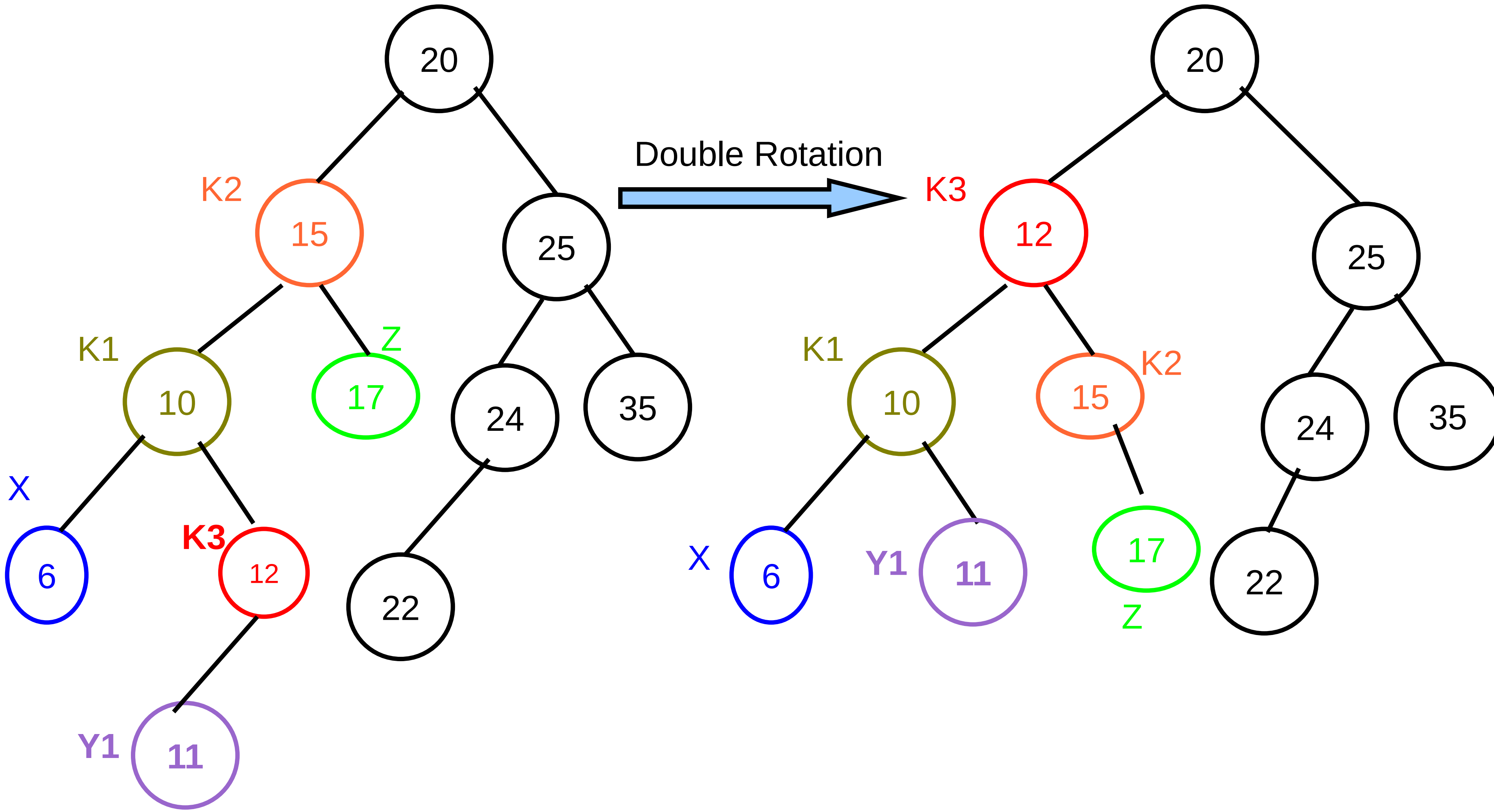
Double Rotation Generalization



Double Rotation

- Any of X, Y1, Y2, and Z can be empty.
- After the rotation, one of Y1 and Y2 are two levels deeper than Z.
- Though we cannot say which is deeper among Y1 and Y2, it turns out that fortunately, it does not matter.
- The resulting tree satisfies search invariant also.
  - Hence the placement of Y1, Y2, etc.

Double Rotation Example



Practice Problem

- Insert the following numbers into an initially empty AVL tree in that order.

29, 52, 76, 25, 45, 41, 17, 37, 32