CS2040C Data Structures and Algorithms
AVL Trees

An AVL tree is a balanced binary search tree - named after its inventors **A**delson-**V**elskii and **L**andis

Outline

- AVL tree property
- Rotation
  - right rotation
  - left rotation
- Height of AVL tree
- AVL tree node insertion
  - single rotation
  - double rotation

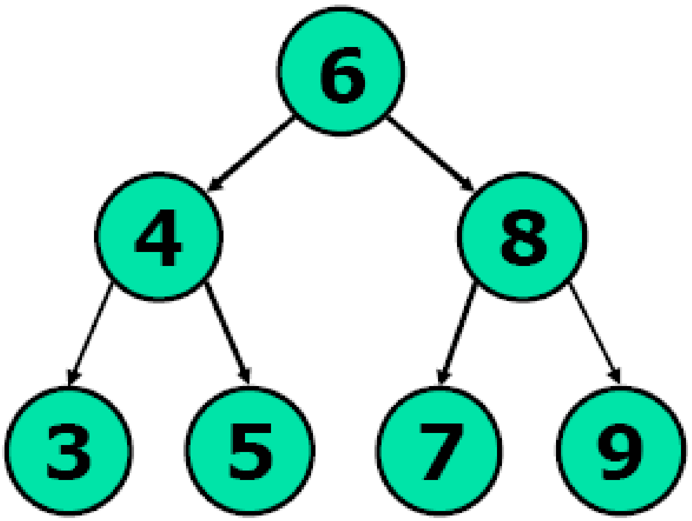
Previously, on BST

- findMin O(h) where h = height of the tree
- search O(h)
- insert O(h)
- delete O(h)

But h is not always $O(\log_2 N)$!

- In the worst case, all BST operations run in O(N) time
- Happens when tree has a linear structure
- Want to maintain additional properties on BST so that it is balanced
- Perfect balance hard to achieve – try to ensure height is always O(log N)

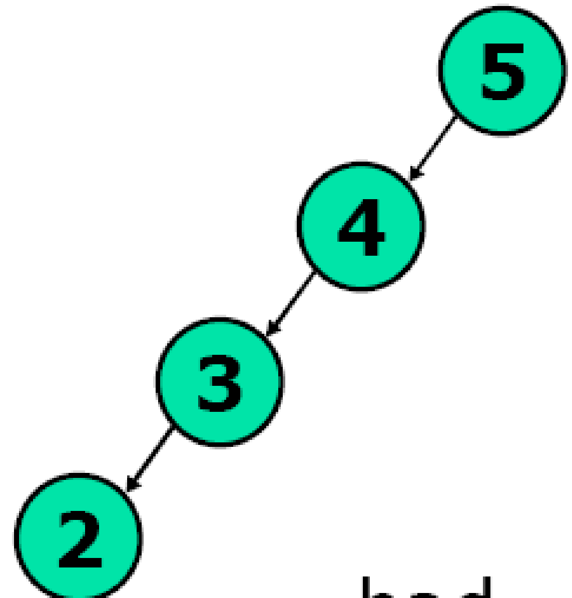
Best case, worst case



good

h = O(log n)

Best case



bad

h = O(n)

worst case

AVL Tree Property

- An AVL tree is a binary search tree that satisfies the AVL tree property:

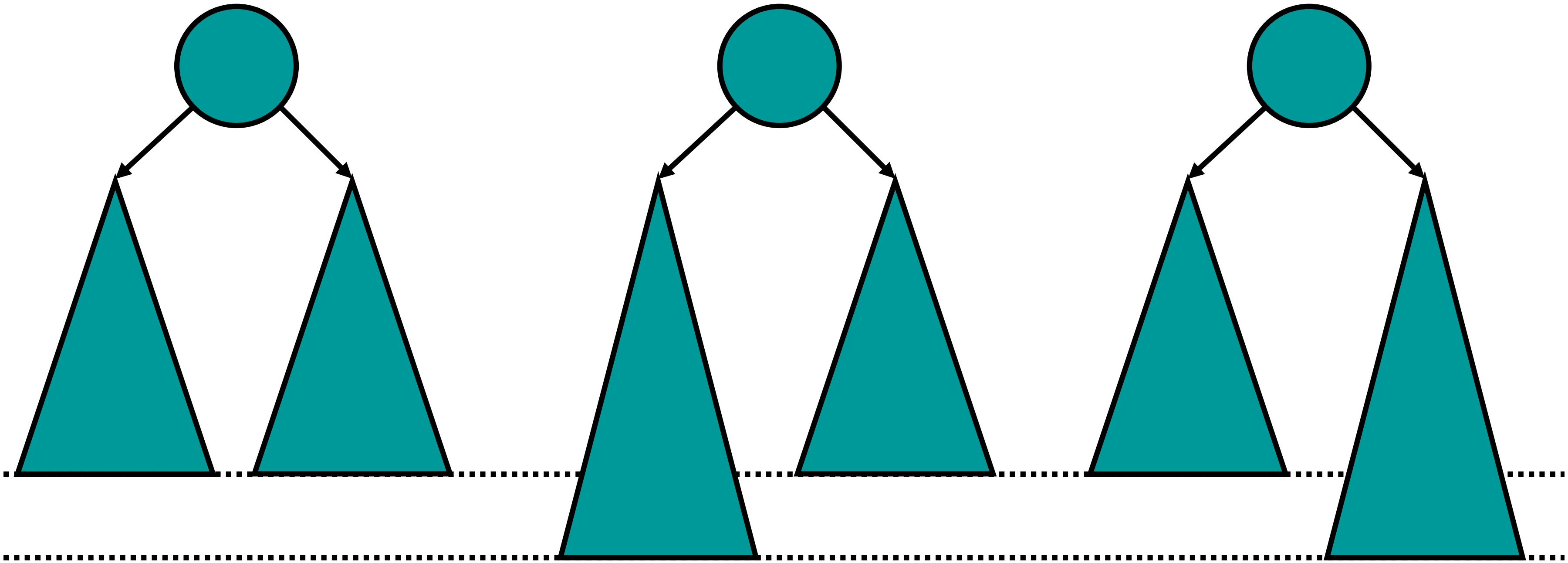
  At any node, the difference in height between left and right subtree is at most one

$$|H_l - H_r| \leq 1$$

Where H_l and H_r are heights of the left and right subtrees of the node

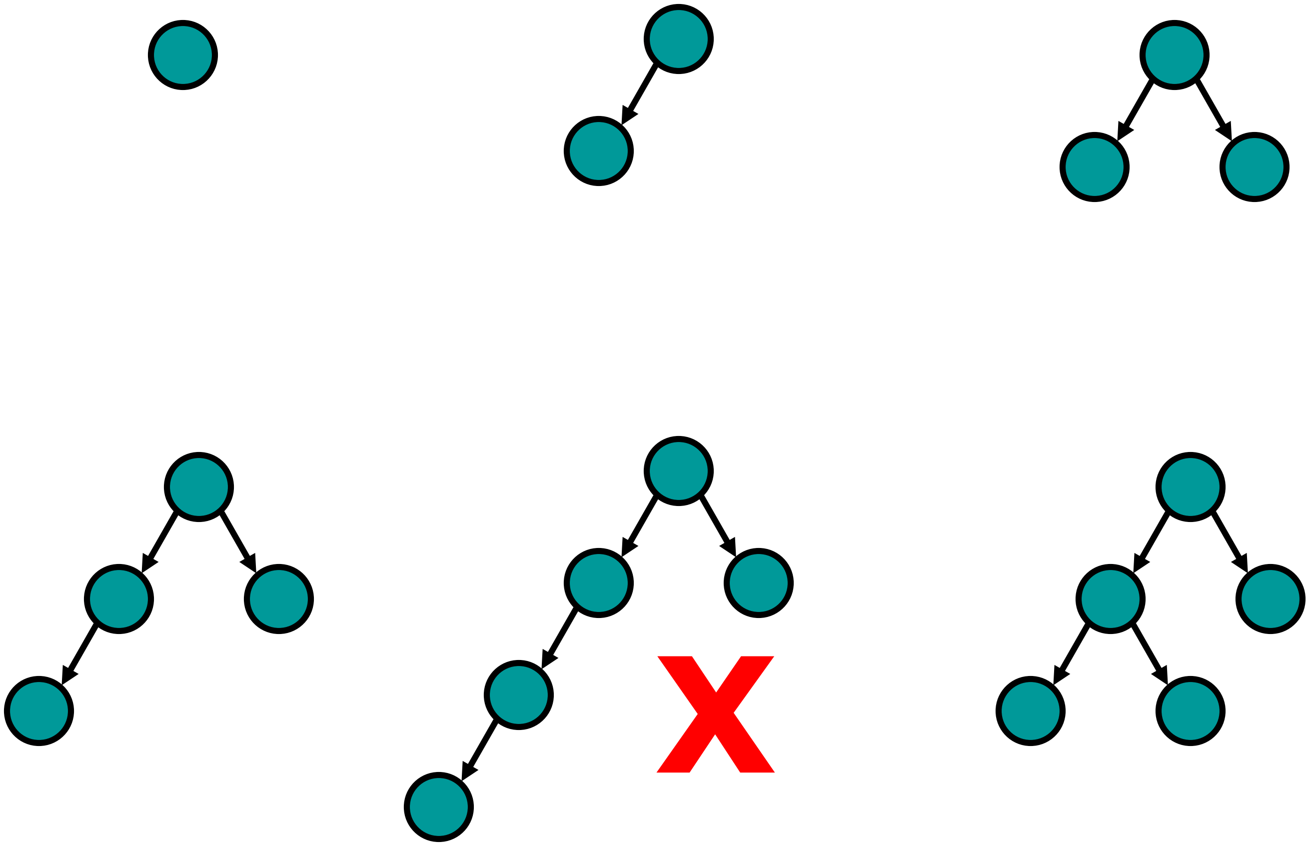
This property must hold recursively for all subtrees.

AVL Tree Property



The difference between the levels of the two dotted lines is one

AVL Tree Examples

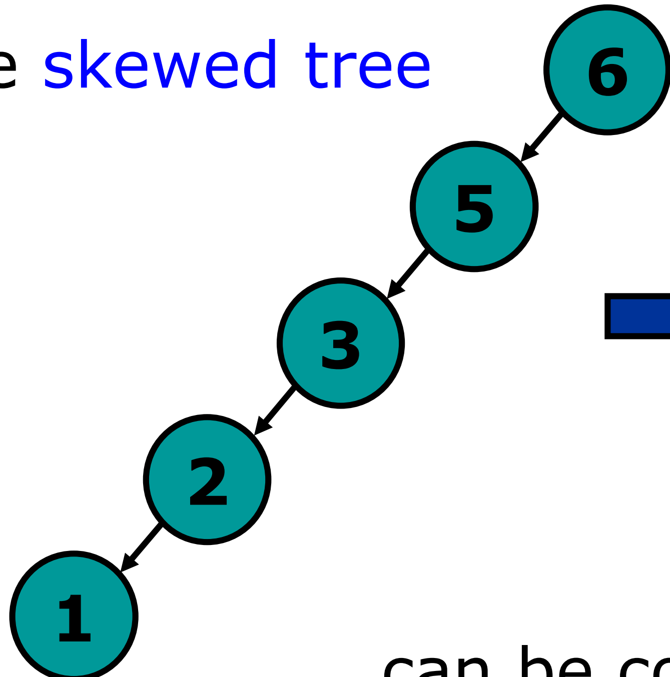


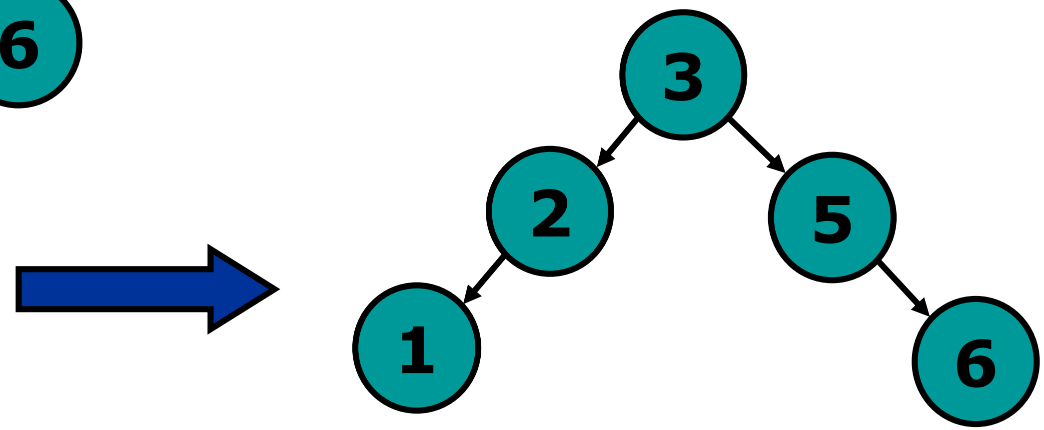
Why?

Rotation

Rotate operation is an important operation for maintaining the balance of a BST

The skewed tree



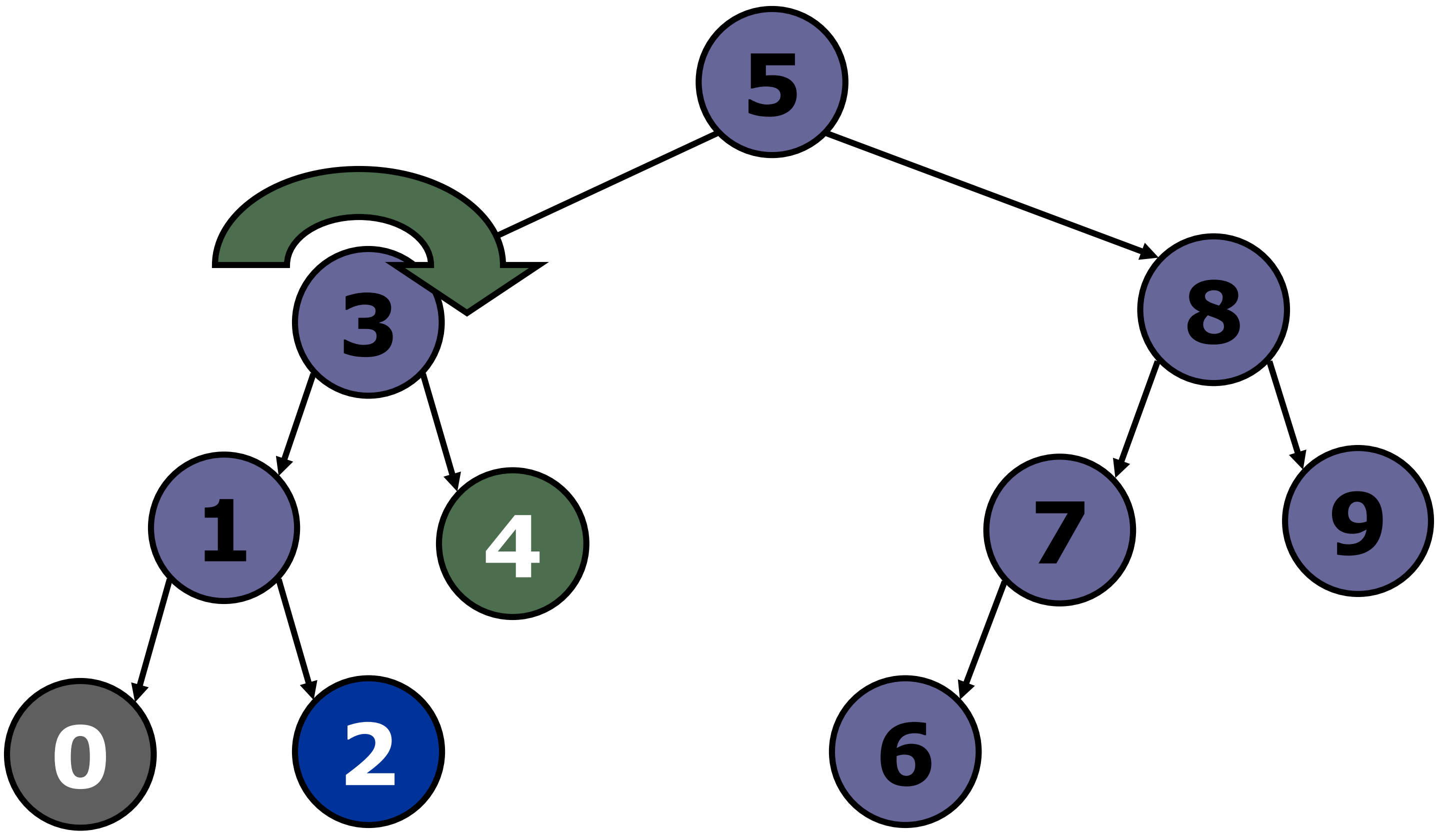


the "**balanced**" tree

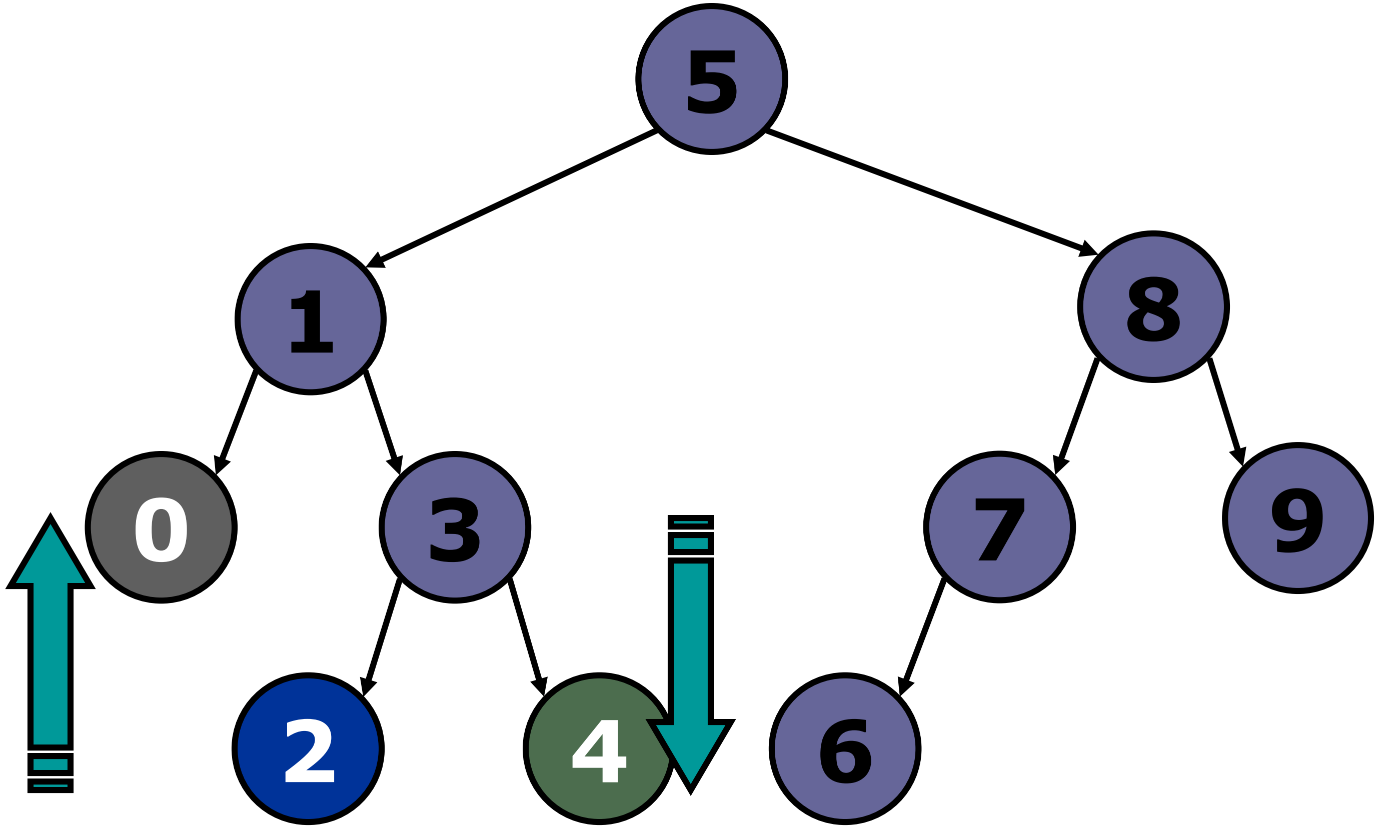
can be converted into

through a series of rotations

Rotate Right at 3

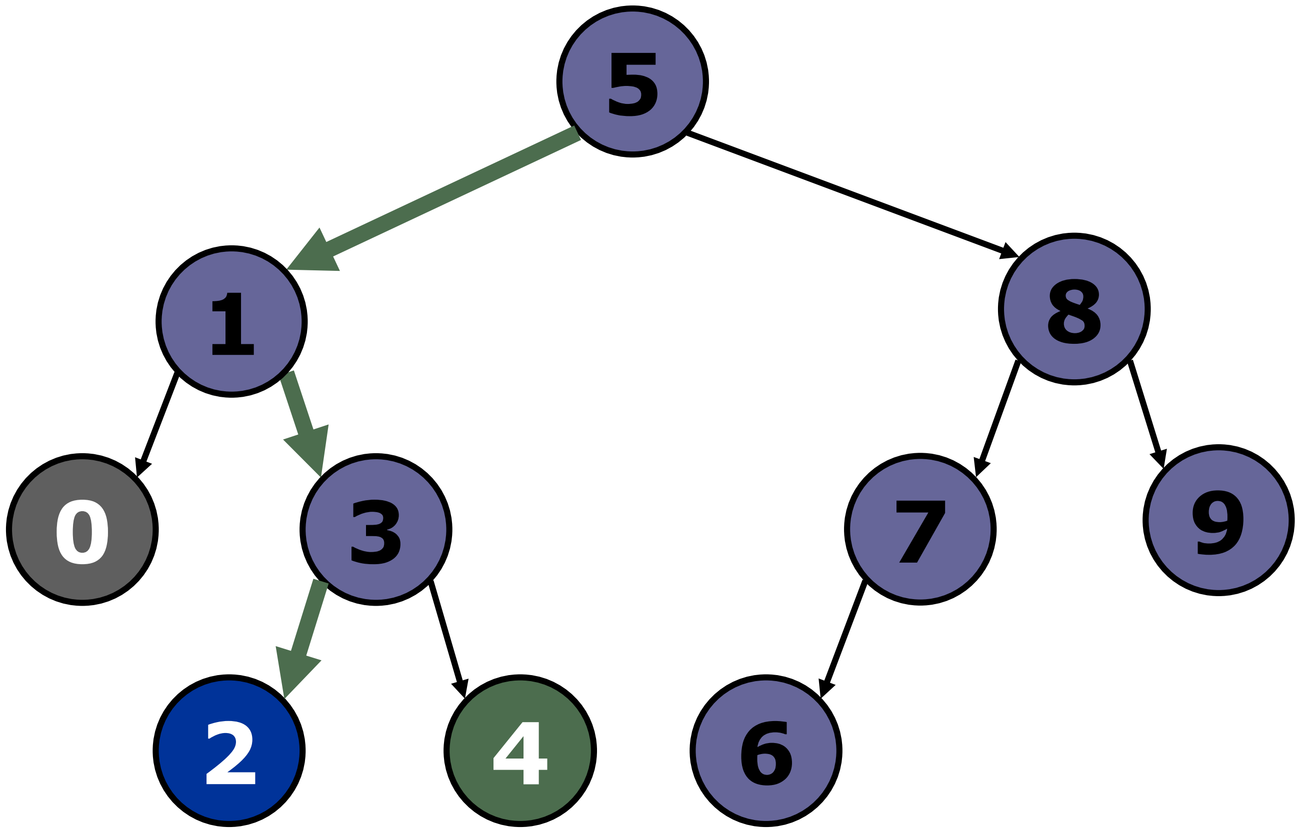


After Rotate Right at 3



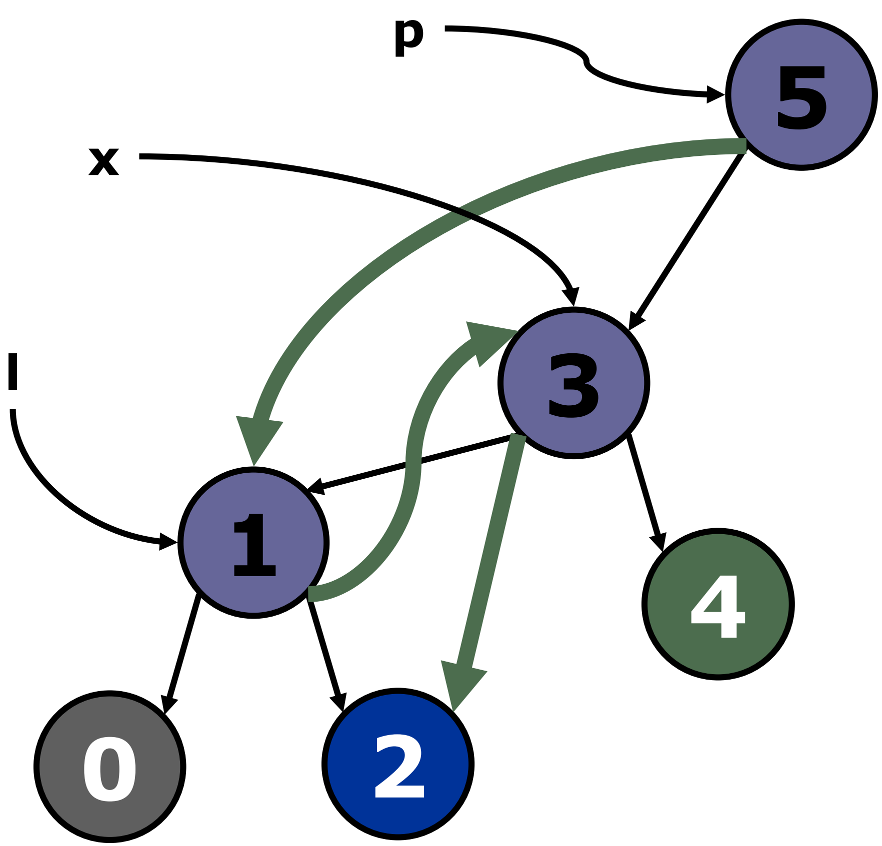
- Rotation changes the **heights** of some nodes
- The depths of nodes 3 and 4 increase by 1
- The depths of nodes 0 and 1 decrease by 1
- The depth of node 2 remains unchanged

After Rotate Right at 3



Rotation modifies the pointers shown in green

Rotate Right at 3



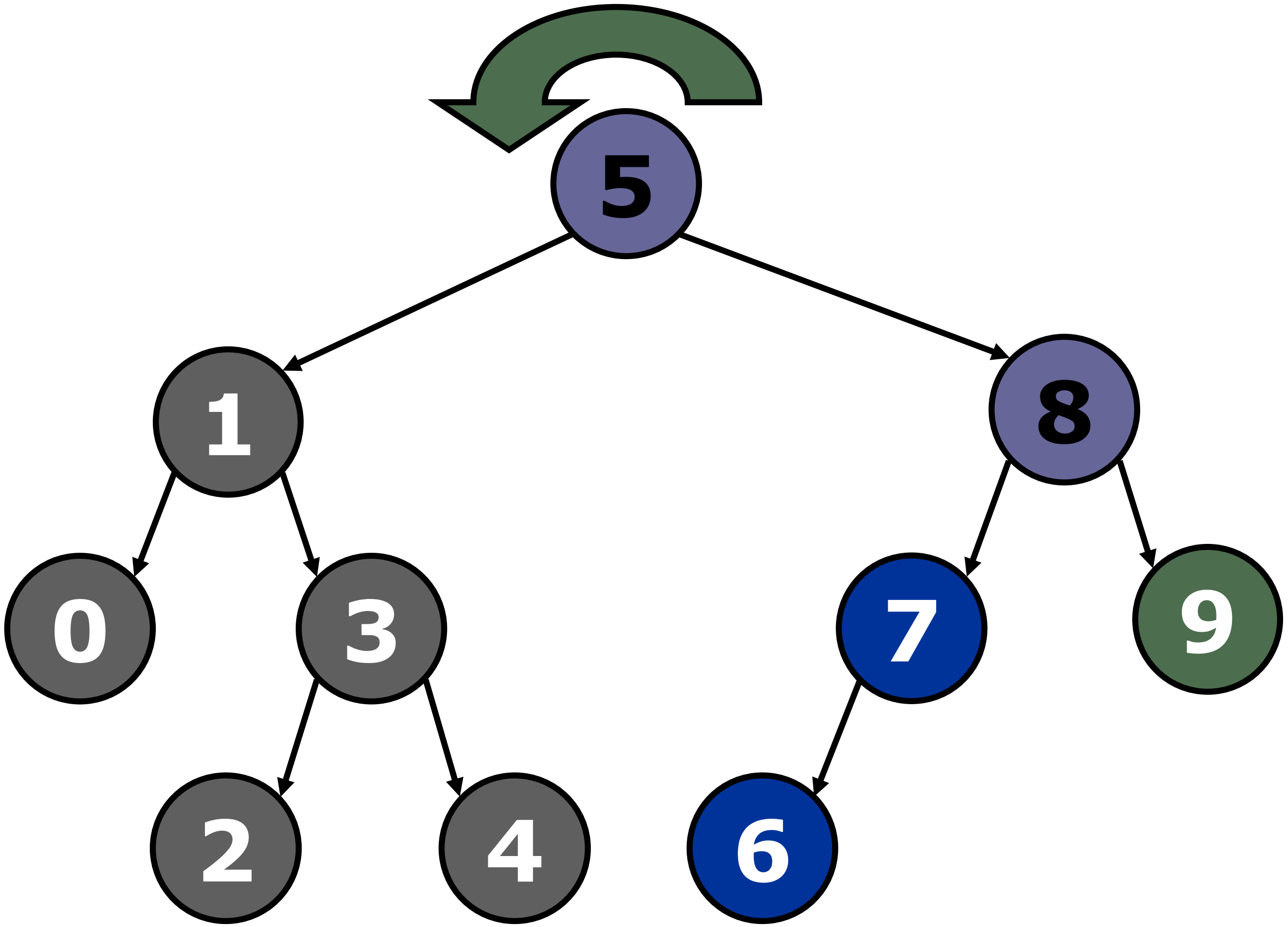

```
rotateRight(x)
    l = x.left
    if l is empty
      return
    x.left = l.right
    l.right = x
    p = x.parent
    if x is a left child
       p.left= l
   else
       p.right = l
```

- The pseudo code on the right shows how we rotate right at x
- The green arrows are the pointers after modification

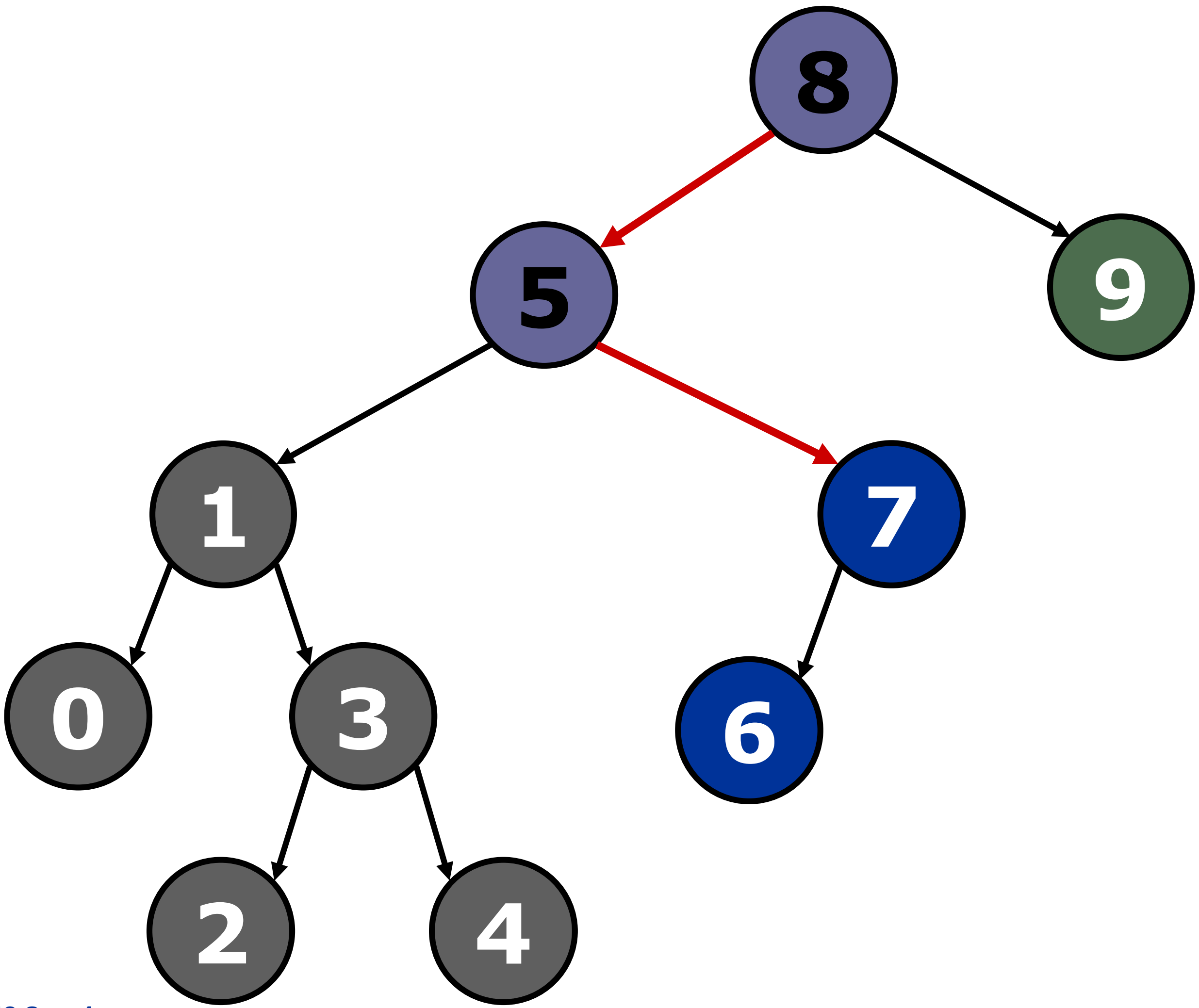
Effect of Rotate Right at x

- l which is x's left child, and l's left subtree, moves up 1 level
- x and x's right subtree move down 1 level
- l's right subtree becomes x's left subtree and remains at the same level
- x's parent becomes l's parent, and x becomes the right child of l

Rotate Left at 5

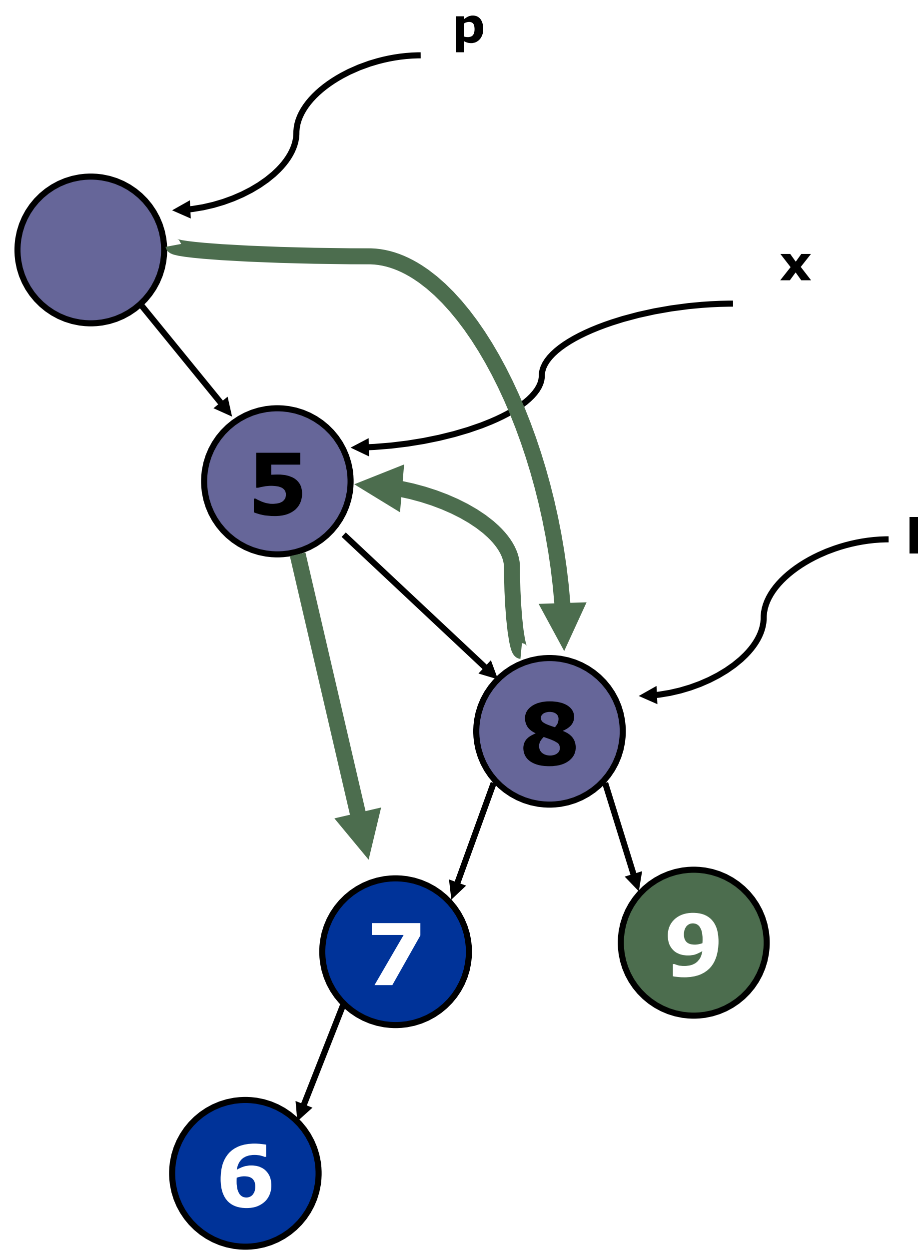


After Rotate Left at 5

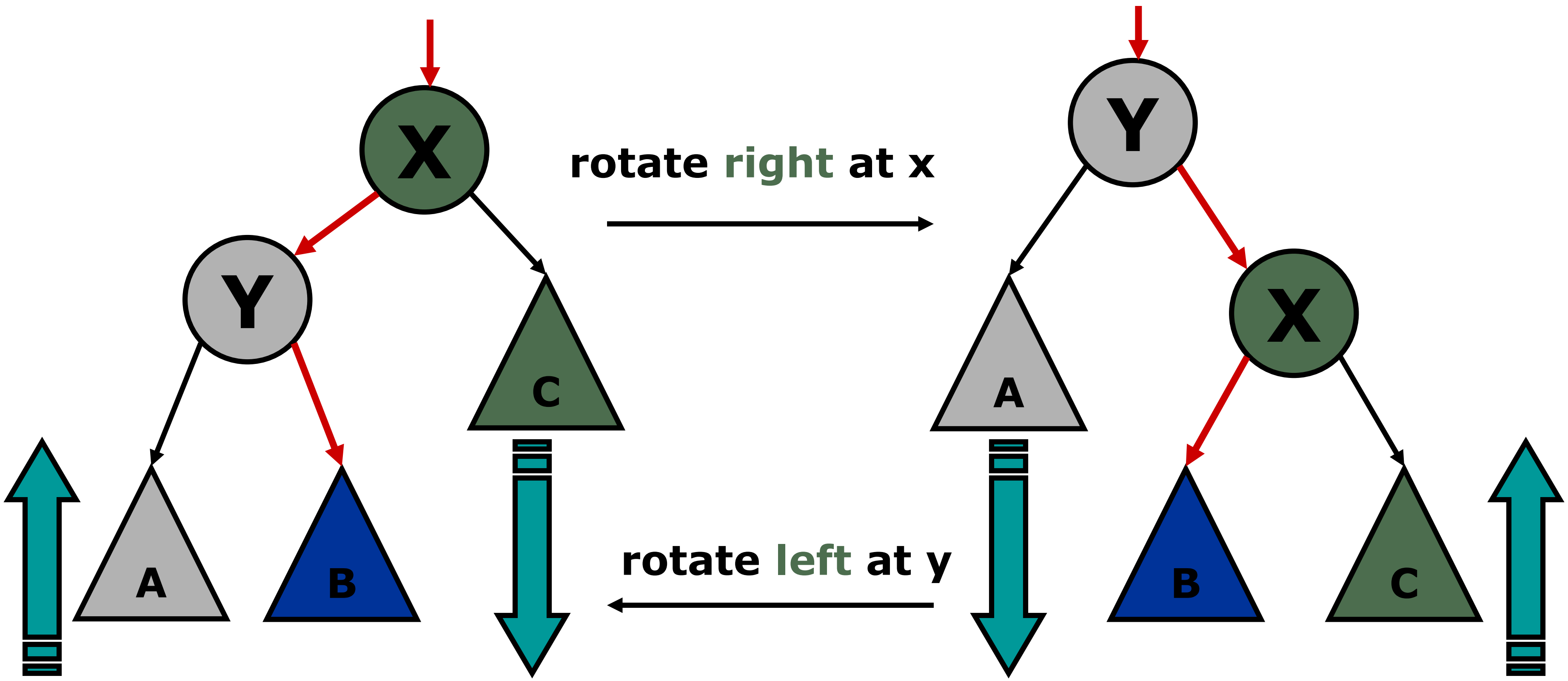


Rotate Left

```
rotateLeft(x)
    l = x.right
    if l is empty
      return
    x.right = l.left
    l.left = x
    p = x.parent
    if x is a right child
       p.right= l
   else
       p.left = l
```

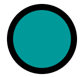


Rotation Summary

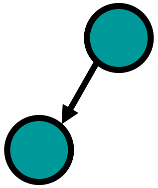


Height of an AVL Tree

- Minimal AVL trees of height h: AVL trees having height h and fewest possible number of nodes
- Minimal AVL tree with height 1



- Minimal AVL tree with height 2

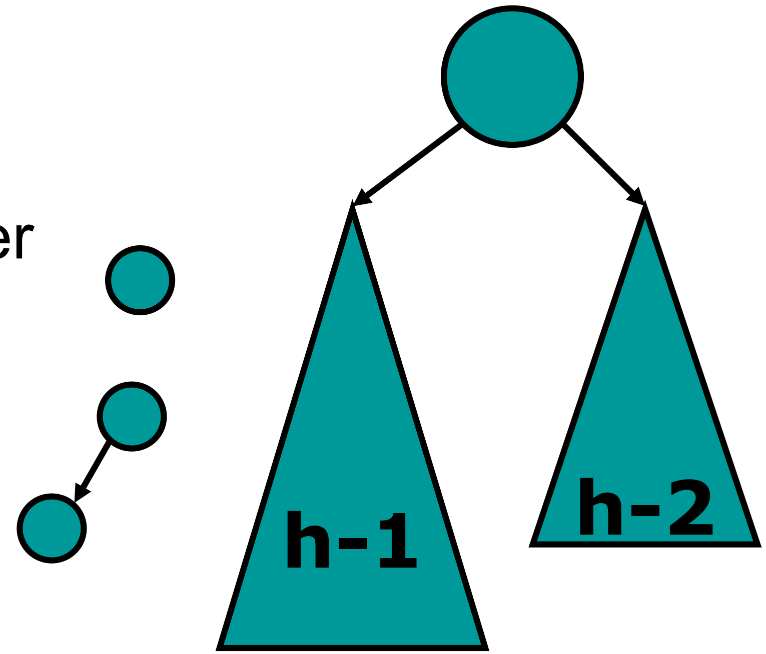


Height of a minimal AVL Tree

- N: number of nodes in a given AVL tree with height h
- n(h): number of nodes in a minimal AVL tree with height h
- n(h) <= N
- Assuming the left subtree is taller
- n(1) = 1
- n(2) = 2
- n(h) = 1 + n(h-1) + n(h-2)

  > 2n(h-2)

  since n(h-1) > n(h-2)



Height of a minimal AVL Tree (cont'd)

$$
\begin{aligned}
n(h) &> 2n(h-2) \\
&> 2 * 2n(h-4) \quad \text{(applying recursively)} \\
&> 2 * 2 * 2n(h-6) \\
&: \\
&> 2^i n(h-2i)
\end{aligned}
$$

when $h - 2i = 1$, $i = (h-1)/2$, (if h is odd)

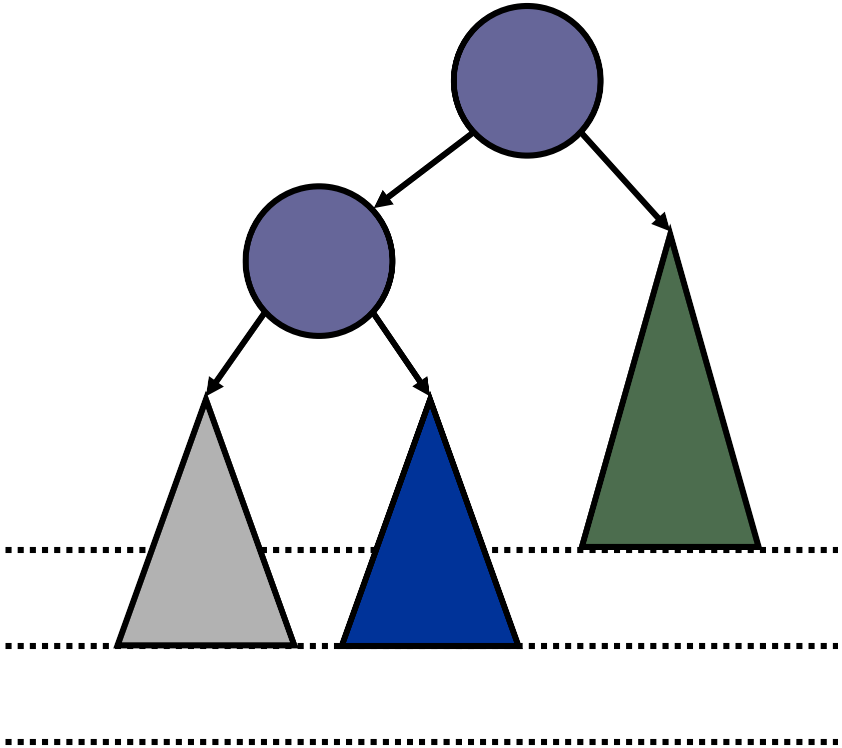
$n(h-2i) = n(1) = 1$

$$
\begin{aligned}
n(h) &> 2^{(h-1)/2} \\
h &< 2 \log n(h) + 1 \\
h &< 2 \log N + 1 \quad \text{(since } N \geq n(h)) \\
h &= O(\log N)
\end{aligned}
$$

AVL Tree Insertion

Idea on Insertion

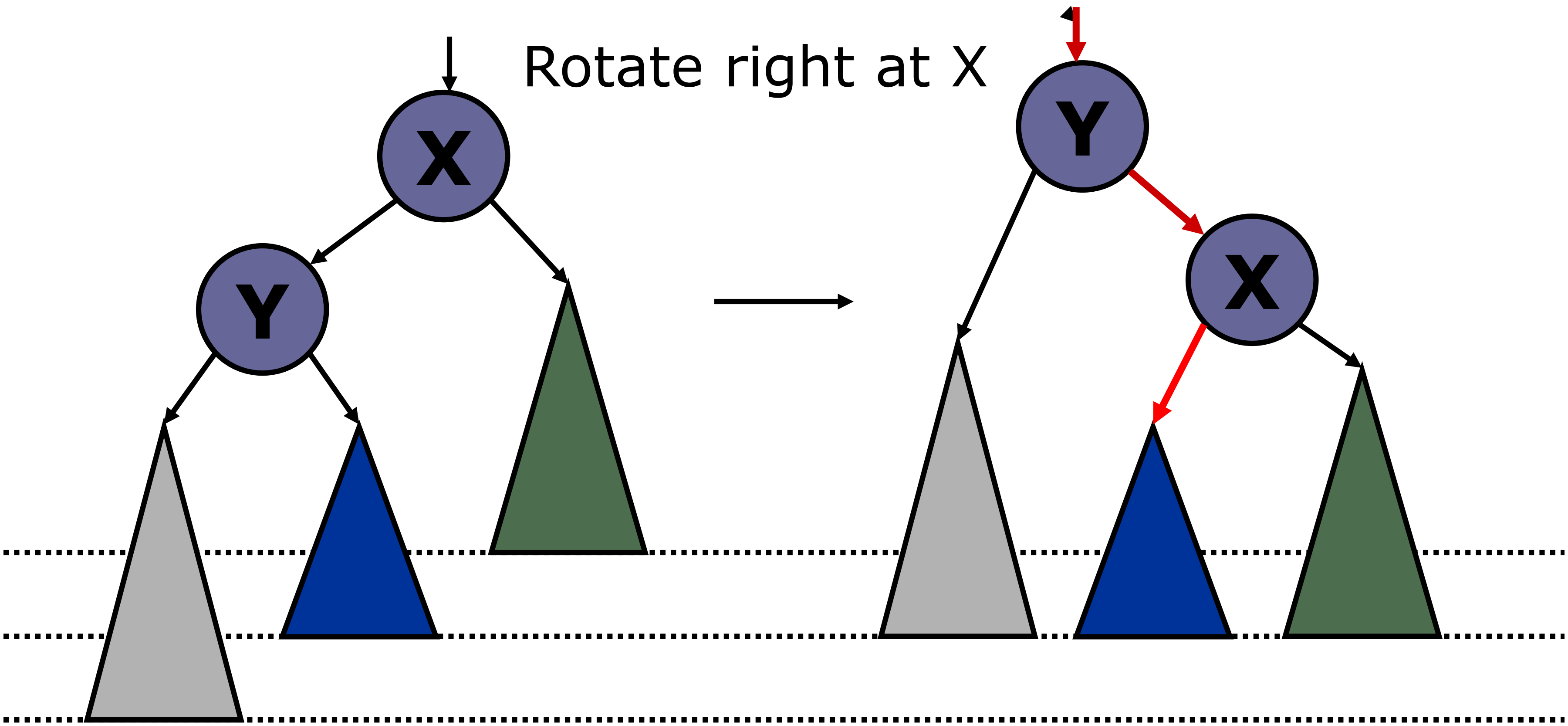
- Insertion in green subtree never violates the AVL tree property
- Insertion into blue and gray subtree may cause a violation



2 passes:
1. Insert the node as usual.
2. After insertion, travel from new node back to the root.
   At each node, check if $| H_l - H_r | \leq 1$.
   If violation occurs, rotate the tree based on the following cases.

Case 1: Insert Outside

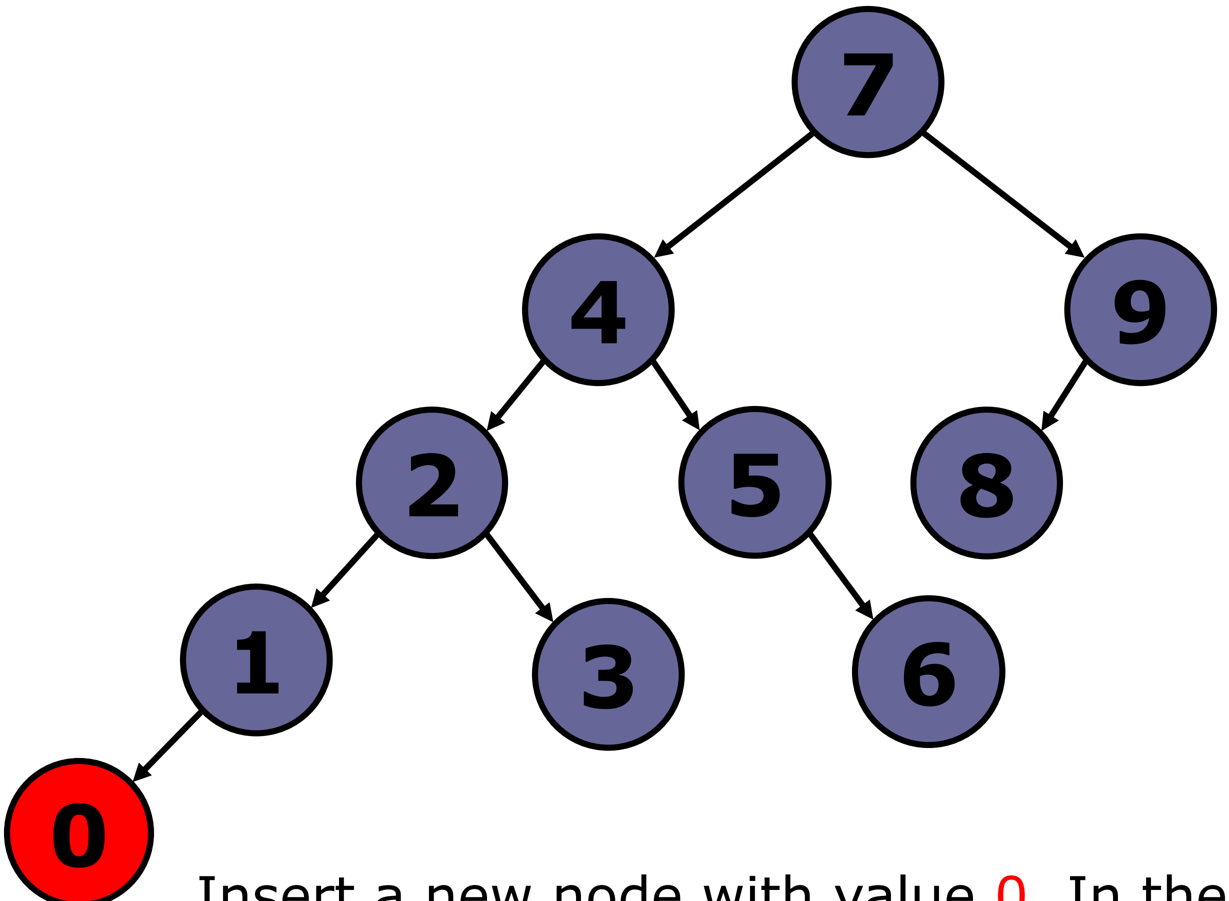
– insert into left subtree of Y



After insertion, if | H_l – H_r | = **2** at X, then Left subtree of Y (left child of X) is taller

Example: Insert Outside

e.g. insert 0



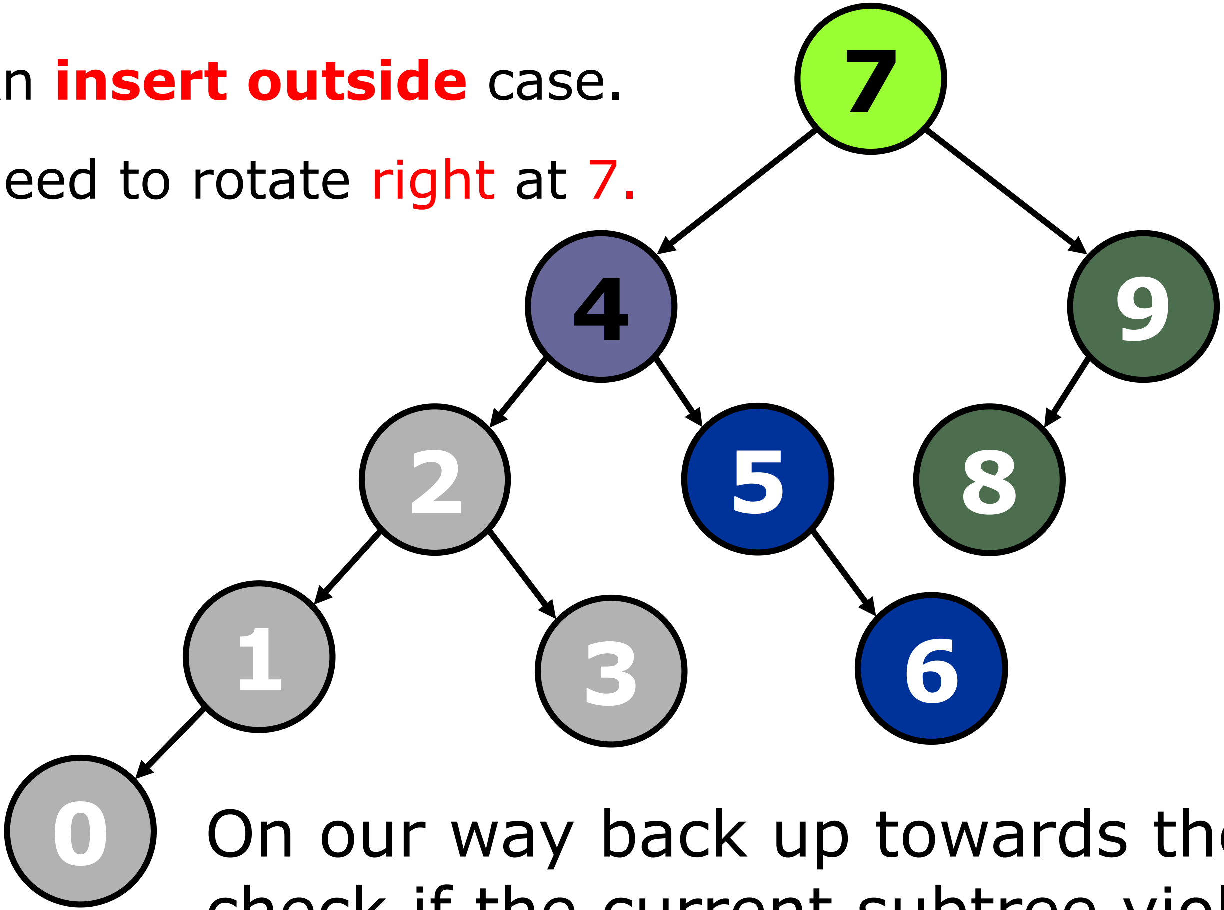
Insert a new node with value 0. In the first pass, we move down the tree just like insertion into a BST.

Example: Insert Outside (cont'd)

Violation at node **7**!

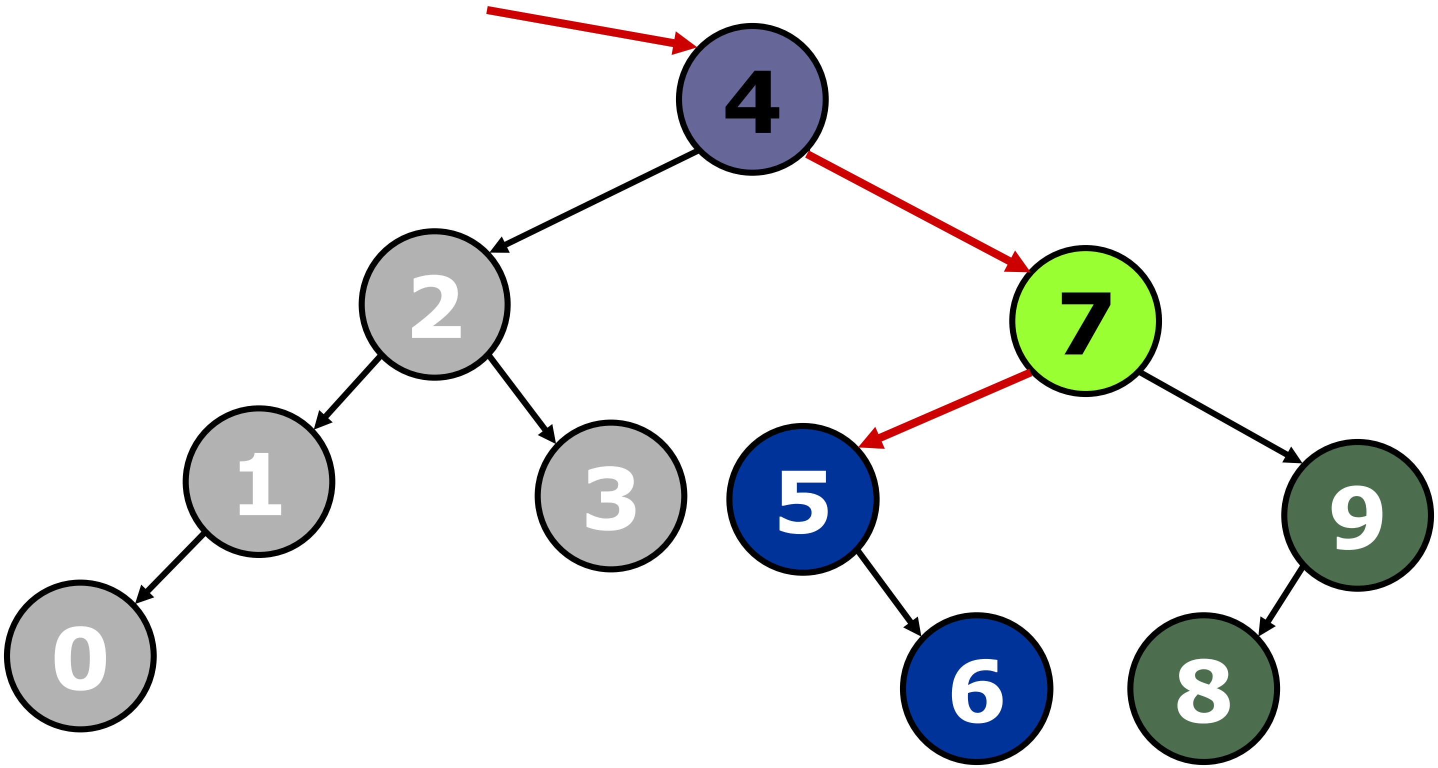
An **insert outside** case.

Need to rotate right at 7.



On our way back up towards the root, we check if the current subtree violates the AVL Tree properties.

Example: Insert Outside (cont'd)

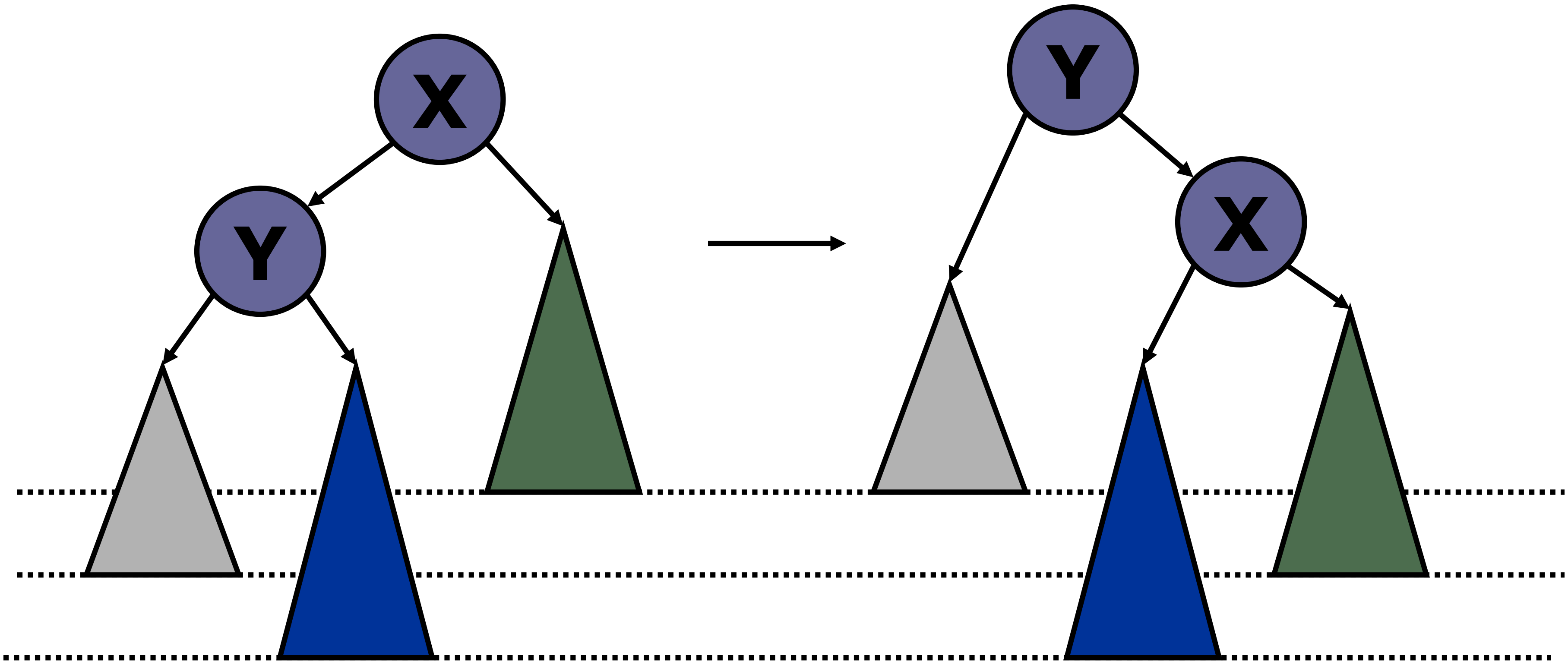


The tree after we perform a single **right rotation at 7 becomes an AVL tree.**

Note the changes in pointers

Case 2: Insert Inside

e.g. insert into blue sub-tree, i.e. the right subtree of Y

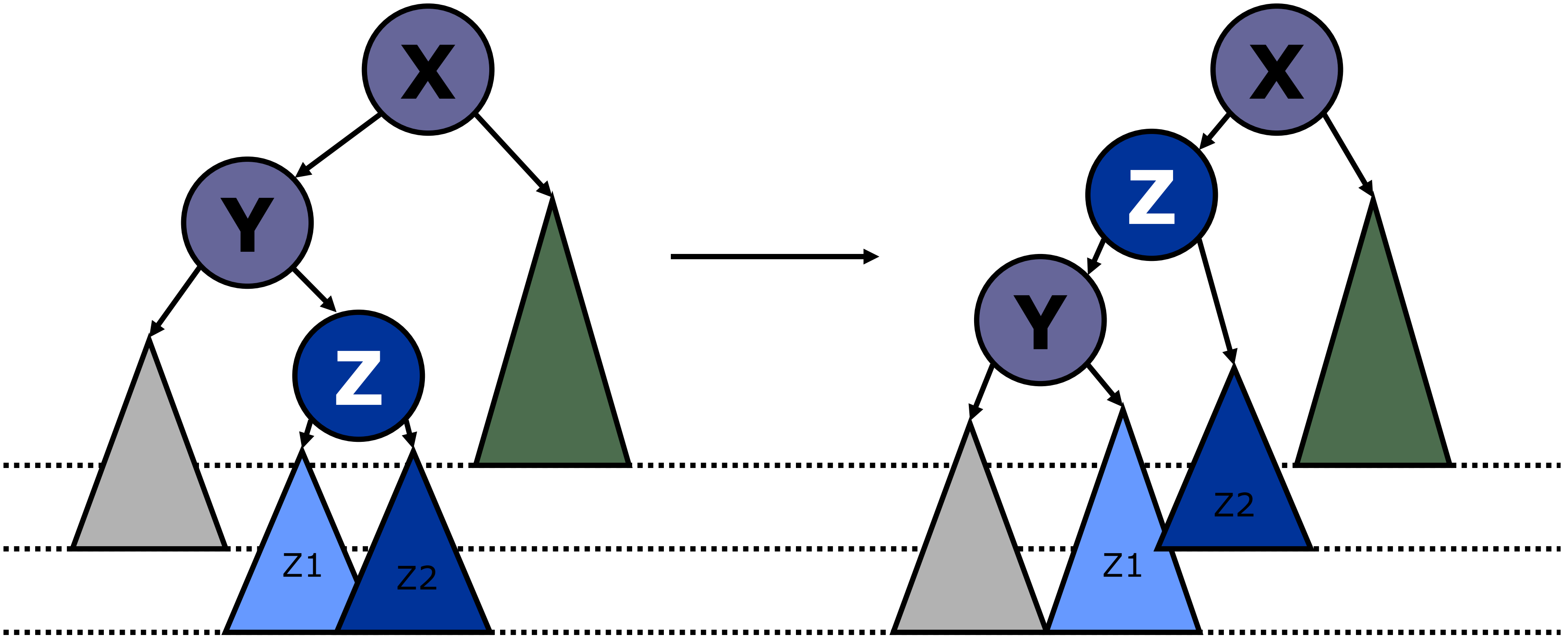


After insertion, when $| H_l - H_r | = \mathbf{2}$ at X, a **single** right rotation at **X** does **not** work.

The height of the blue subtree remains unchanged.

Case 2: Insert Inside

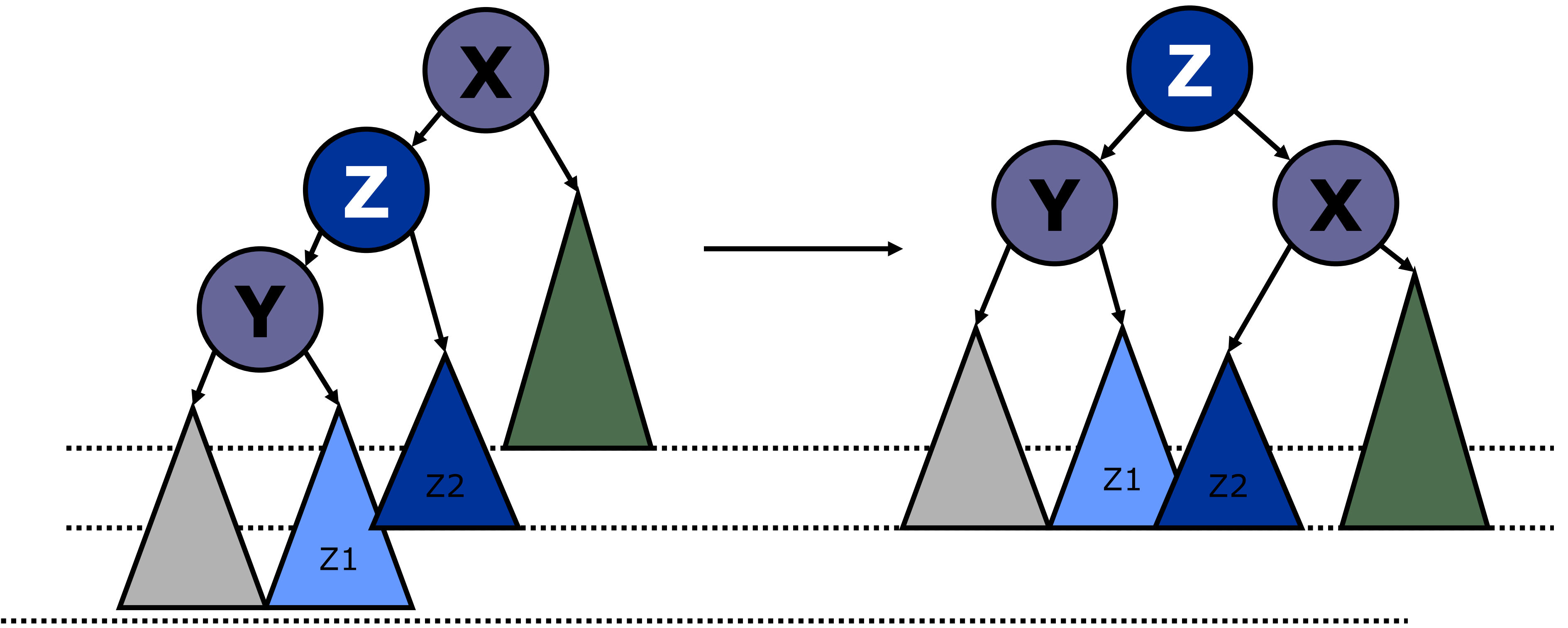
(inserted node into subtree rooted at z)



First rotate left around Y

become case 1

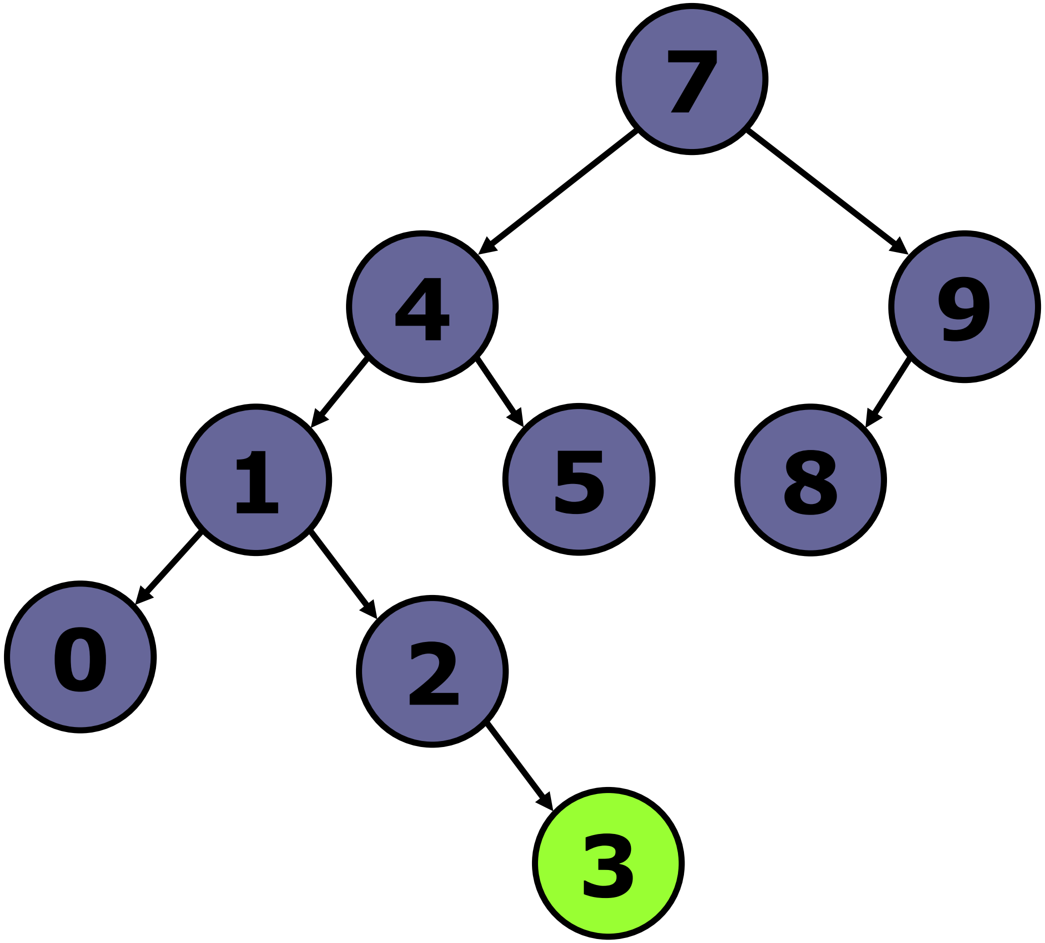
Case 2: Becomes Case 1



Then rotate right around X

Example: Insert Inside

e.g. insert 3

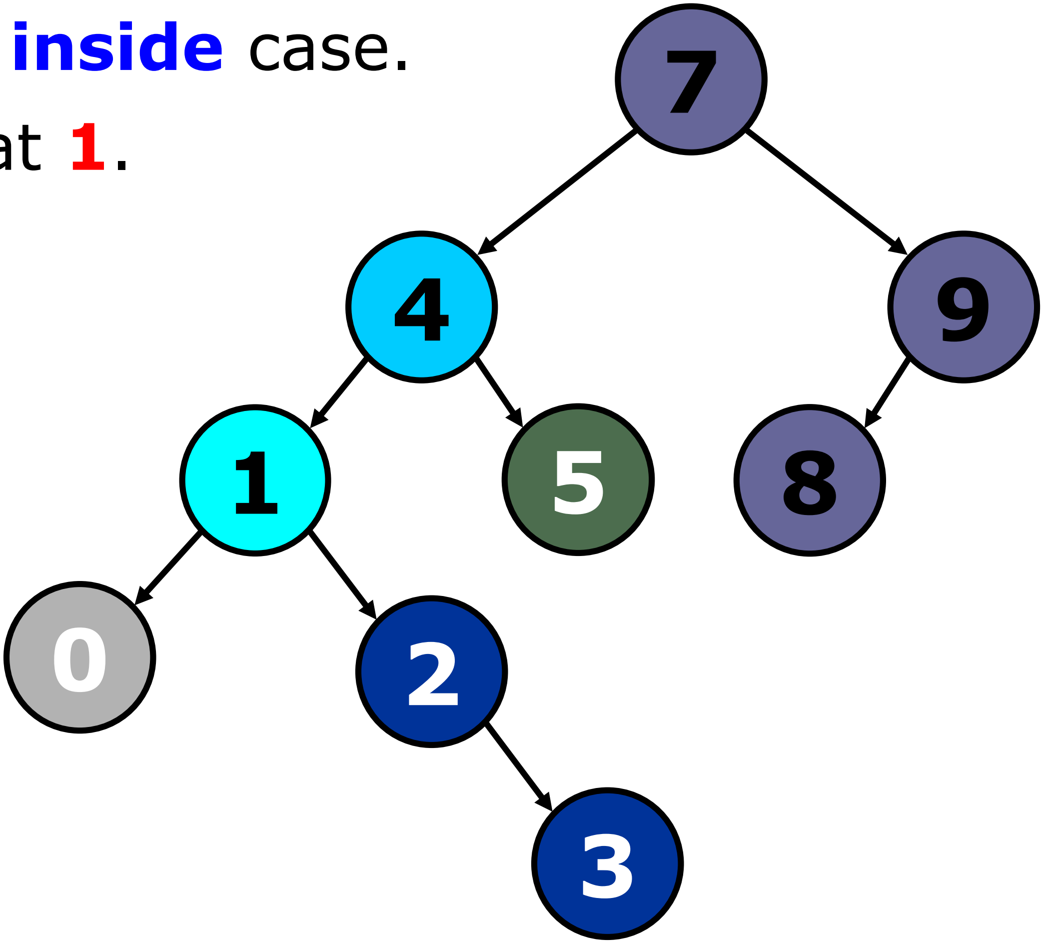


Example: Insert Inside (cont'd)

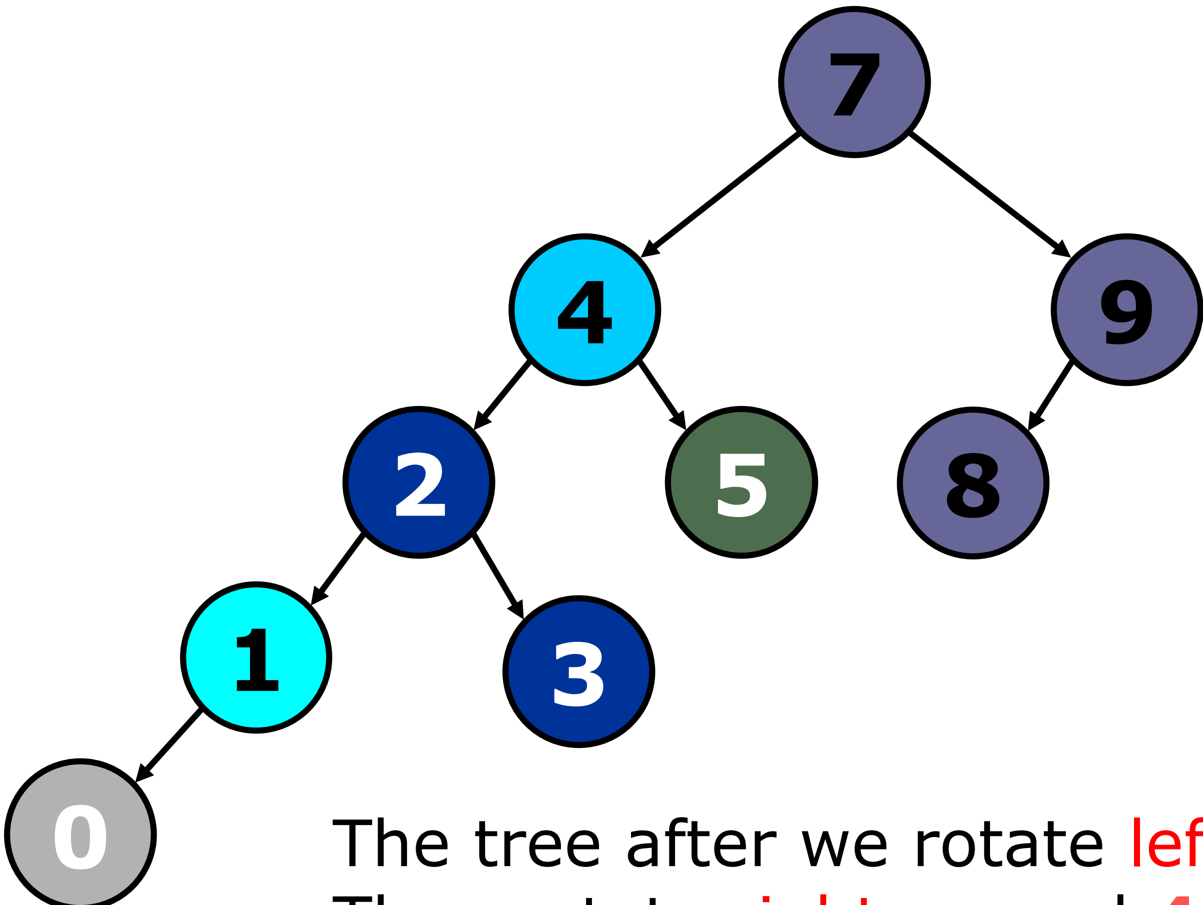
AVL Tree property violated at **4**.

This is an **insert inside** case.

First, rotate left at **1**.

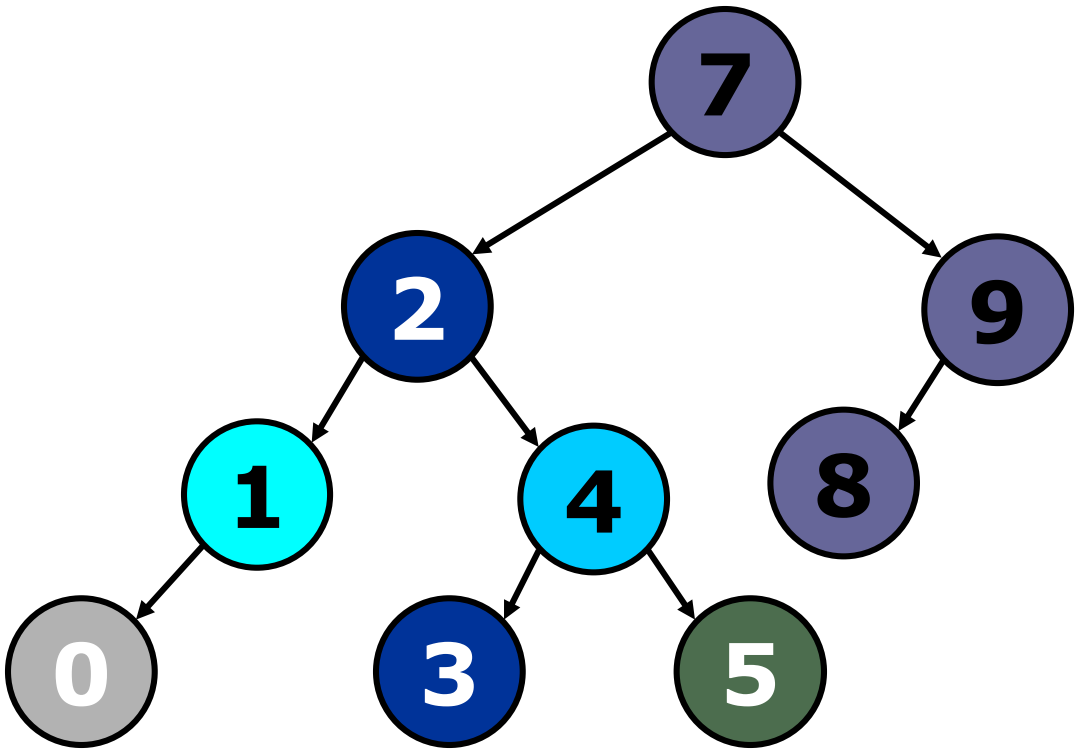


Example: Insert Inside (cont'd)



The tree after we rotate left at **1**.
Then rotate right around **4.**

Example: Insert Inside (cont'd)



After we rotate right around 4, the tree becomes an AVL tree.

Summary

- AVL Tree is a balanced binary search tree
- Balance maintained by AVL Tree property
- Insertion: two passes needed:
  - first pass down to insert, second pass up to fix violation.
- Insert outside: Single Rotation
- Insert inside: Double Rotation

Q: How about deletion of nodes from an AVL tree?