AVL Trees

AVL Trees

- It is named after its inventors Georgy Adelson-Velsky and Evgenii Landis.

Georgy M. Adelson-Velsky
1922-2014

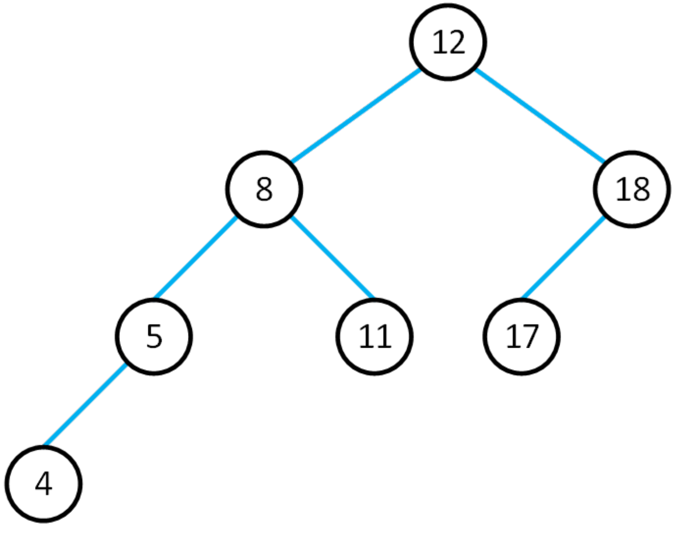
Evgenii Mikhailovich Landis
1921-1997

Outline

- Definition of AVL trees
- AVL Tree Operations
  - Insertion
  - Deletion
- Partial Implementation

Recap: Trees and Binary Tree

- The height of a node is number of edges of the root to the deepest leaf.
- The height of a tree is the height of its root.
- The height of a tree with one node is 0.
- The height of an empty tree is -1.
- In a balanced tree, the height of the left and right subtrees for each node differ by no more than 1 (either 0 or 1).
- Binary search tree (BST):
  - Left subtree $<$ Node $<$ Right subtree



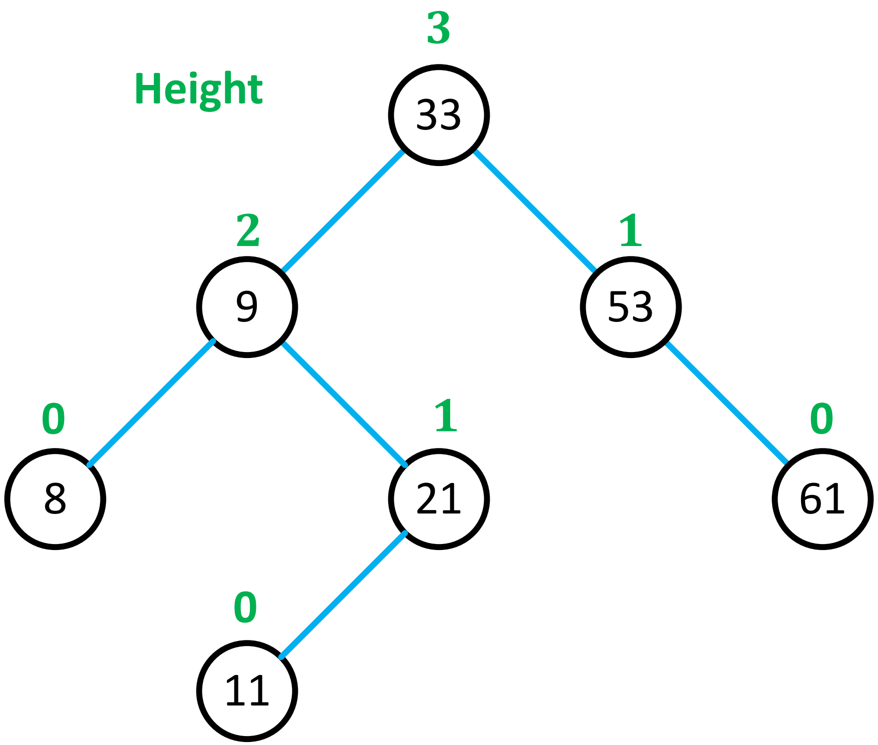
AVL Trees and Balance Factor

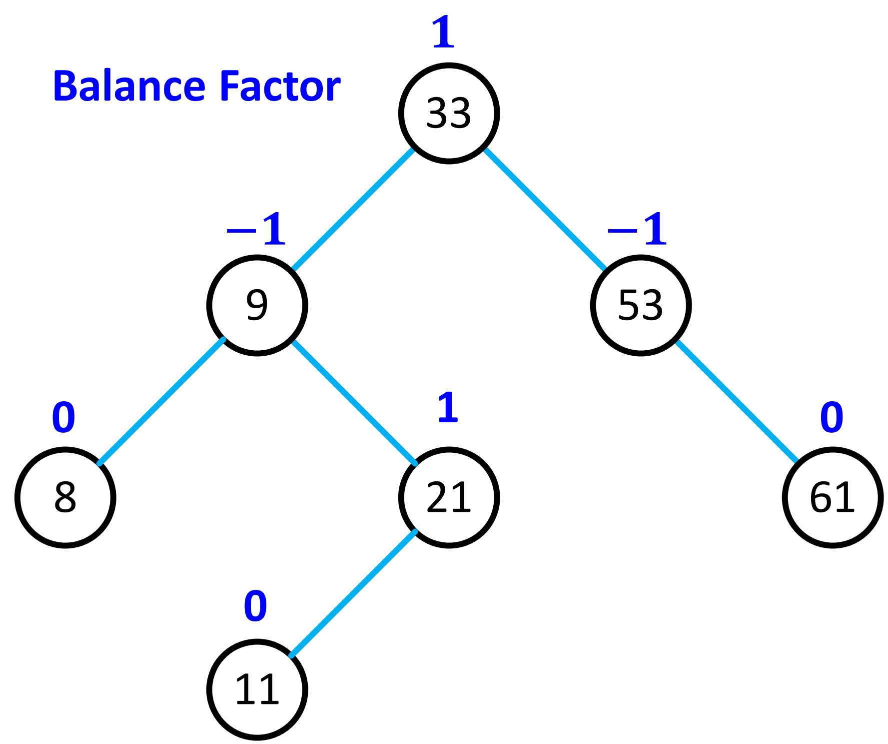
- AVL tree is a balanced BST.
- Balance factor (BF) of a node in a tree

$$BF = h_L - h_R$$

  - h_L: Height of the left subtree
  - h_R: Height of the right subtree
  - $BF > 0$: left-heavy ($h_L > h_R$)
  - $BF < 0$: right-heavy ($h_L < h_R$)
  - The value of the balance factor of an AVL tree should always be $-1, 0$ or 1.

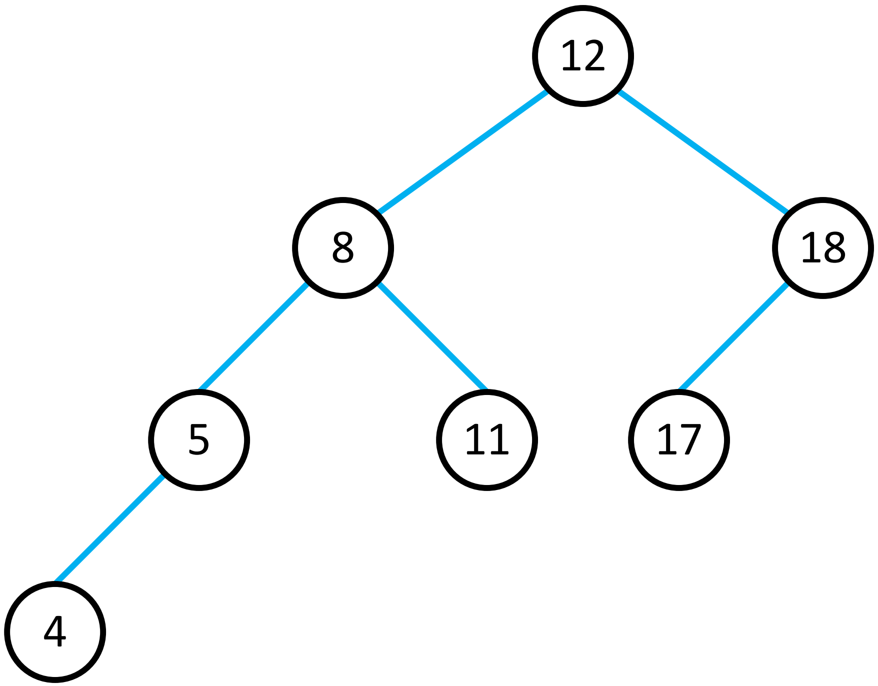
Example: Balance Factor





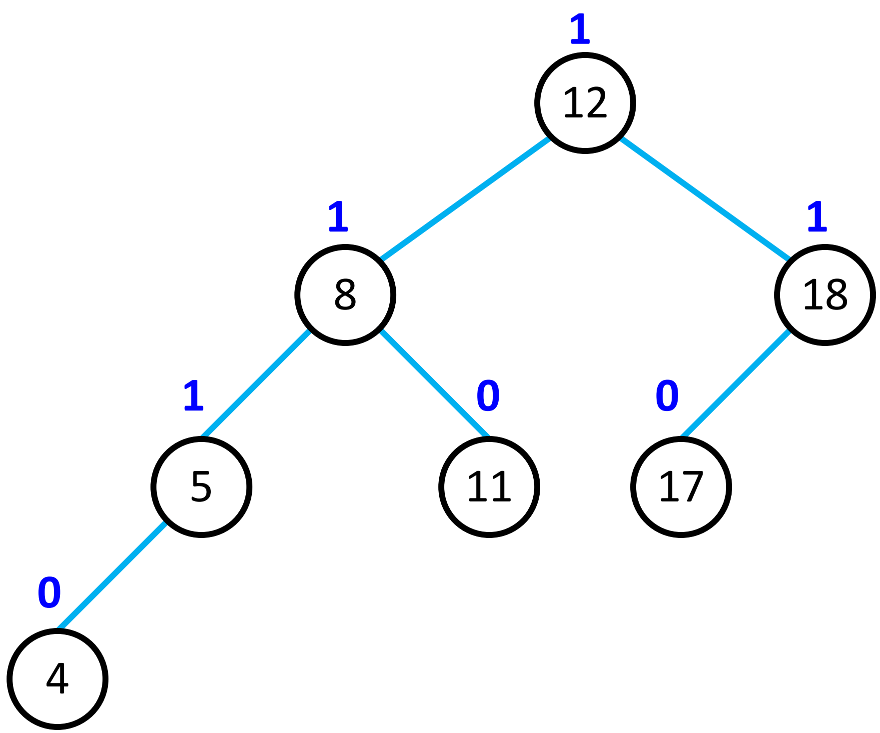
Exercise: AVL Tree

- Check if this is this an AVL tree.



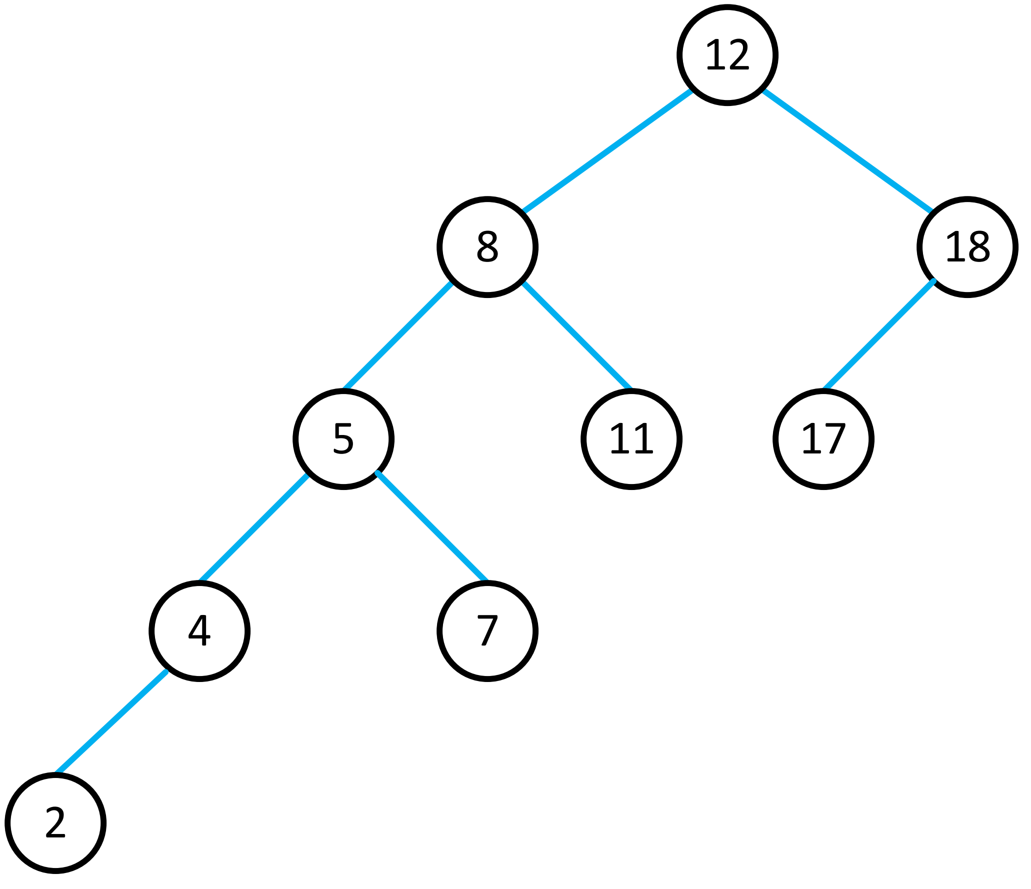
Exercise: AVL Tree

- Check if this is this an AVL tree.



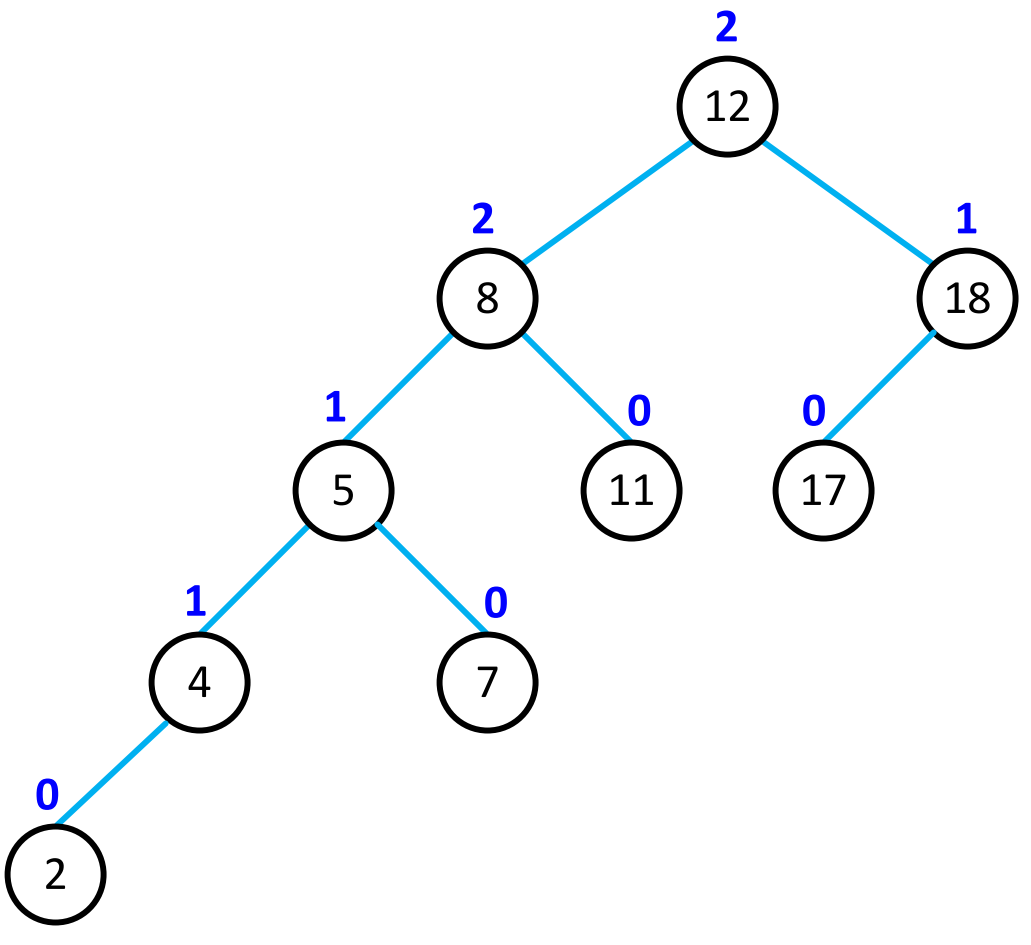
Exercise: AVL Tree

- Check if this is this an AVL tree.



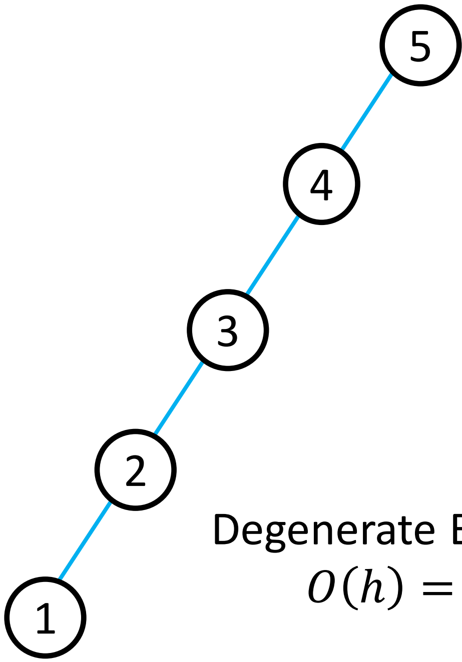
Exercise: AVL Tree

- Check if this is this an AVL tree.

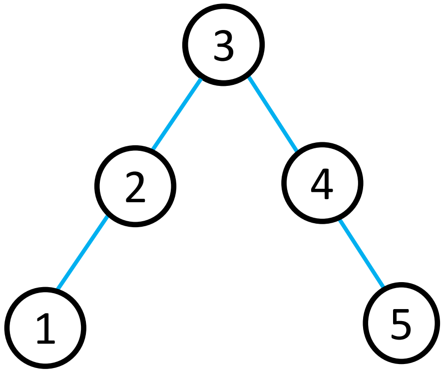


Advantages of AVL Trees

- Recall that the worst-case complexity of the BST operations is $O(h)$ where h is the height of the BST.



Degenerate Binary Tree
$O(h) = O(N)$



Balanced Binary Tree
$O(h) = O(\log(N))$

- The worst-case complexity of the AVL Tree operations (e.g., searching, insertion and deletion) is $O(\log N)$, where N is the number of nodes in the tree.

AVL Tree Operations

- Searching, insertion and deletion of AVL trees are similar to BSTs.
- However, insertion and deletion may violate the balance property.
- Thus, additional adjustment may be required to restore the balance after insertion and deletion.
- This rebalance is achieved through one or more tree rotations.
- The type of rotations depends on the tree orientation.

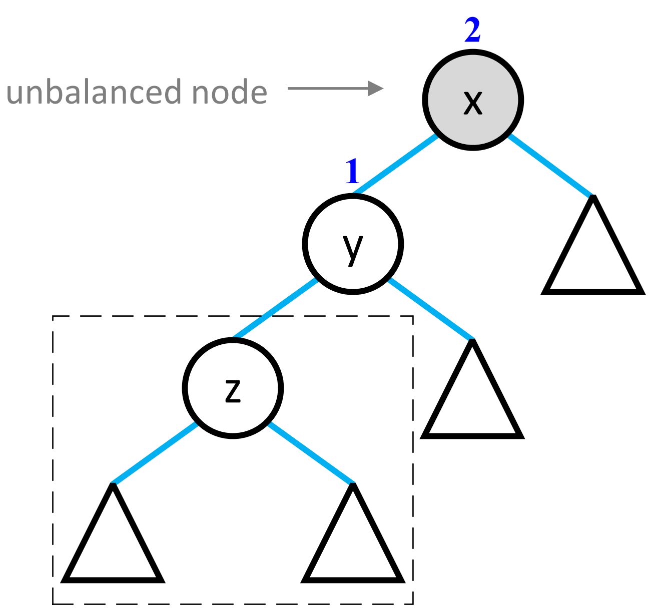
AVL Tree Rebalancing

- There are four possible types of rotations to rebalance an AVL tree after insertion or deletion depending on where is the taller subtree that causes the imbalance.
  - **Case 1 and Case 2**: When imbalance is due to the to the **left** subtree of the unbalanced node
  - **Case 3 and Case 4**: When imbalance is due to the to the **right** subtree of the unbalanced node

AVL Tree Rebalancing

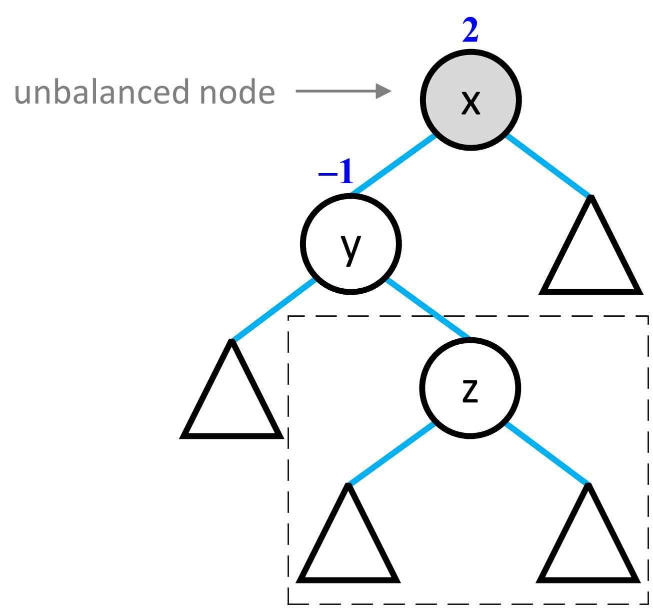
Case 1: When the imbalance is caused by the left child's left subtree of the unbalanced node

⟶ Right rotation



Case 2: When the imbalance is caused by the left child's right subtree of the unbalanced node

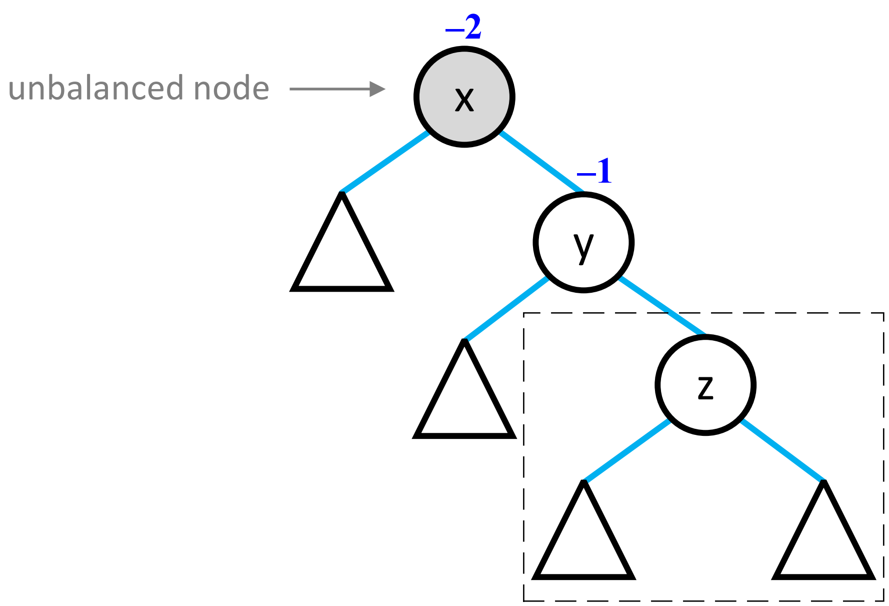
⟶ Left-right rotation



AVL Tree Rebalancing

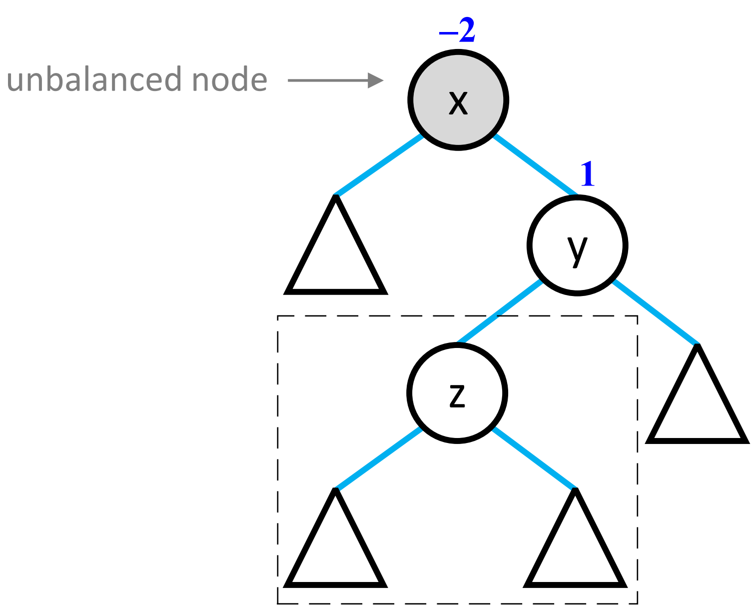
Case 3: When the imbalance is caused by the right child's right subtree of the unbalanced node

⟶ **Left** rotation



Case 4: When the imbalance is caused by the right child's **left** subtree of the unbalanced node

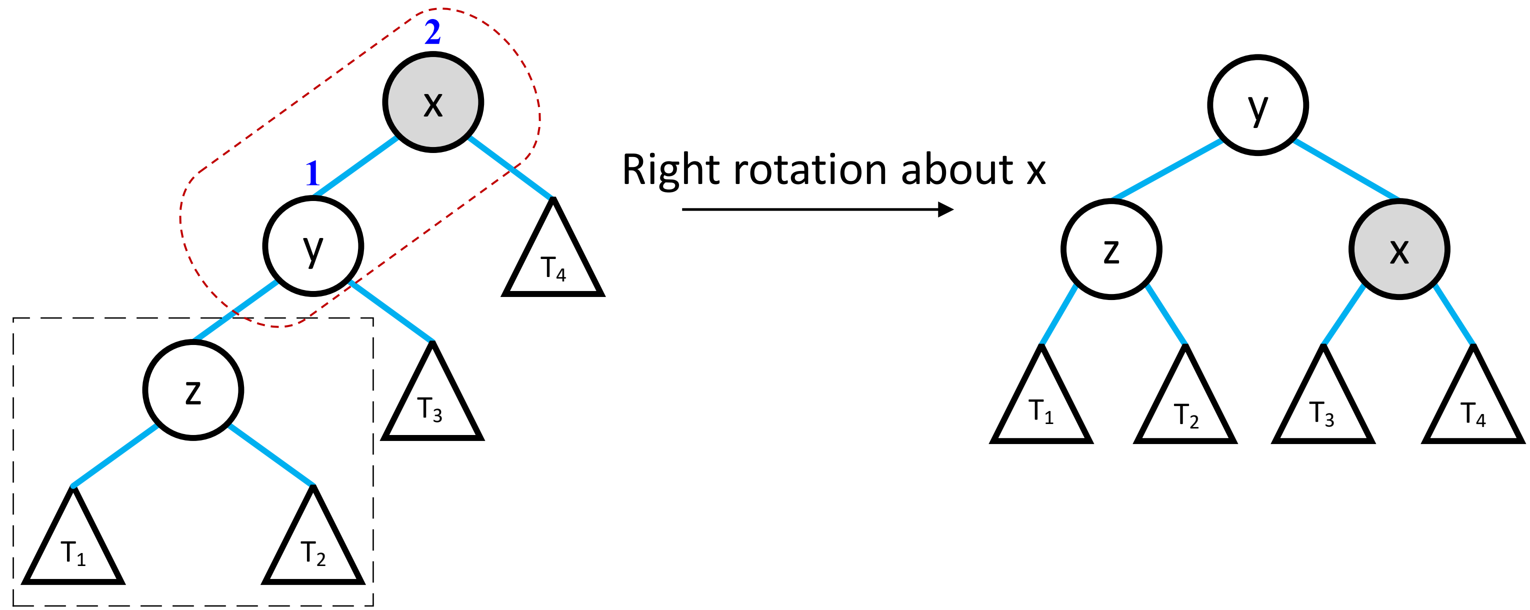
⟶ **Right-left** rotation



AVL Tree Rebalancing: Case 1

Case 1: When the imbalance is caused by the **left** child's **left** subtree of the unbalanced node

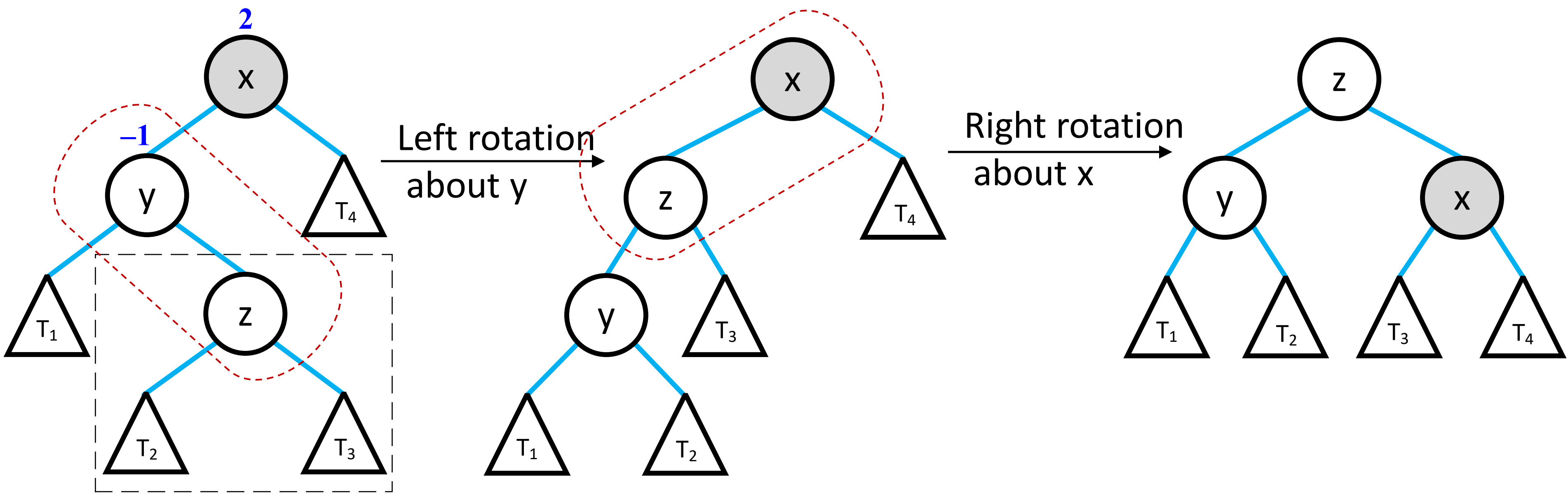
⟶ **Right** rotation



AVL Tree Rebalancing: Case 2

Case 2: When the imbalance is caused by the **left** child's **right** subtree of the unbalanced node

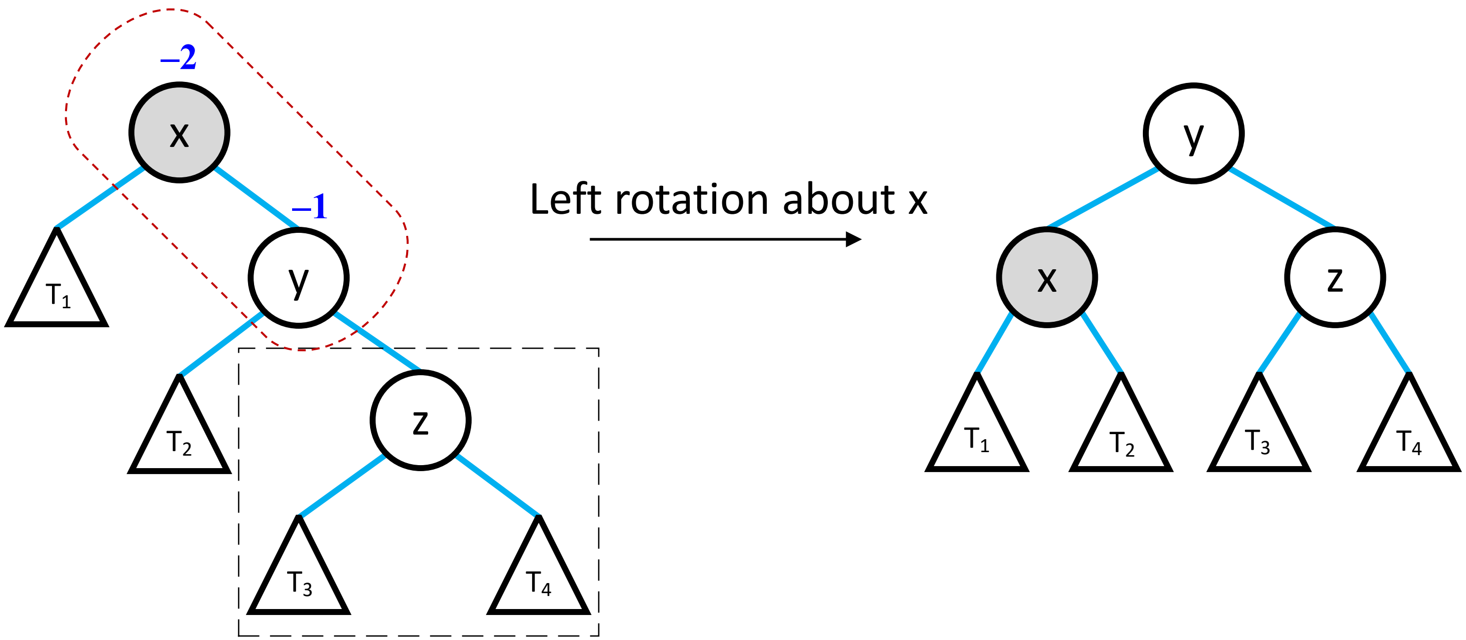
⟶ **Left**-**right** rotation



AVL Tree Rebalancing: Case 3

Case 3: When the imbalance is caused by the **right** child's **right** subtree of the unbalanced node

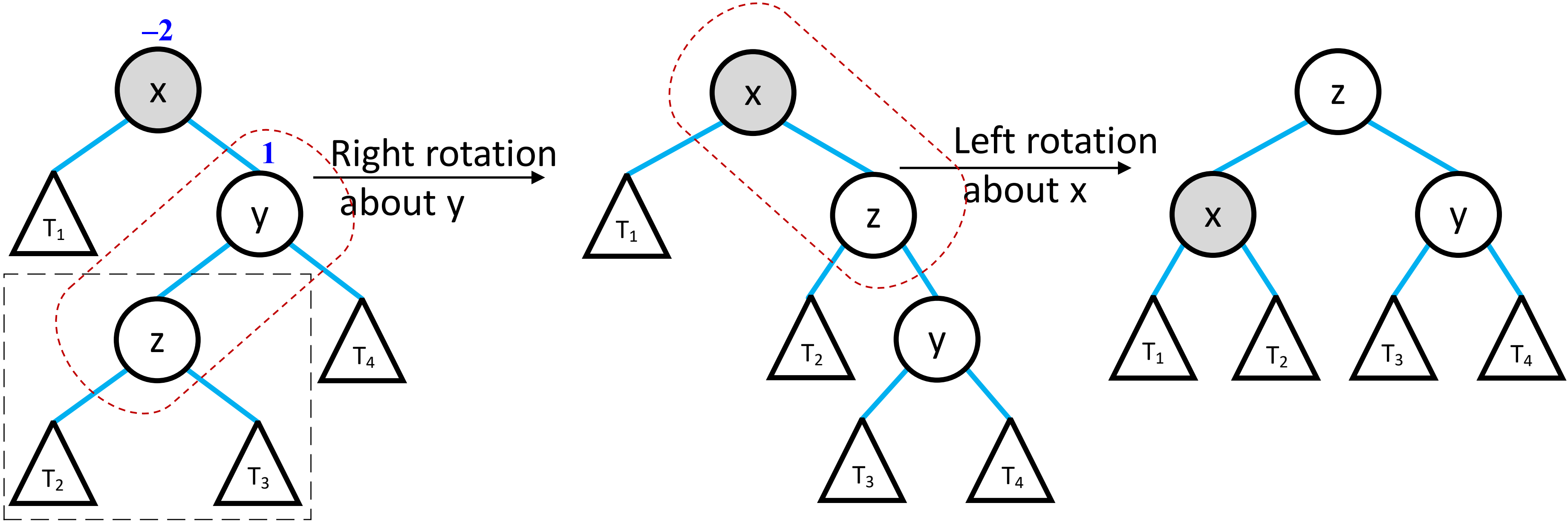
⟶ **Left** rotation



AVL Tree Rebalancing: Case 4

Case 4: When the imbalance is caused by the **right** child's **left** subtree of the unbalanced node

⟶ **Right-left** rotation



AVL Tree Rebalance

- Denote the deepest node that is unbalanced as node x.

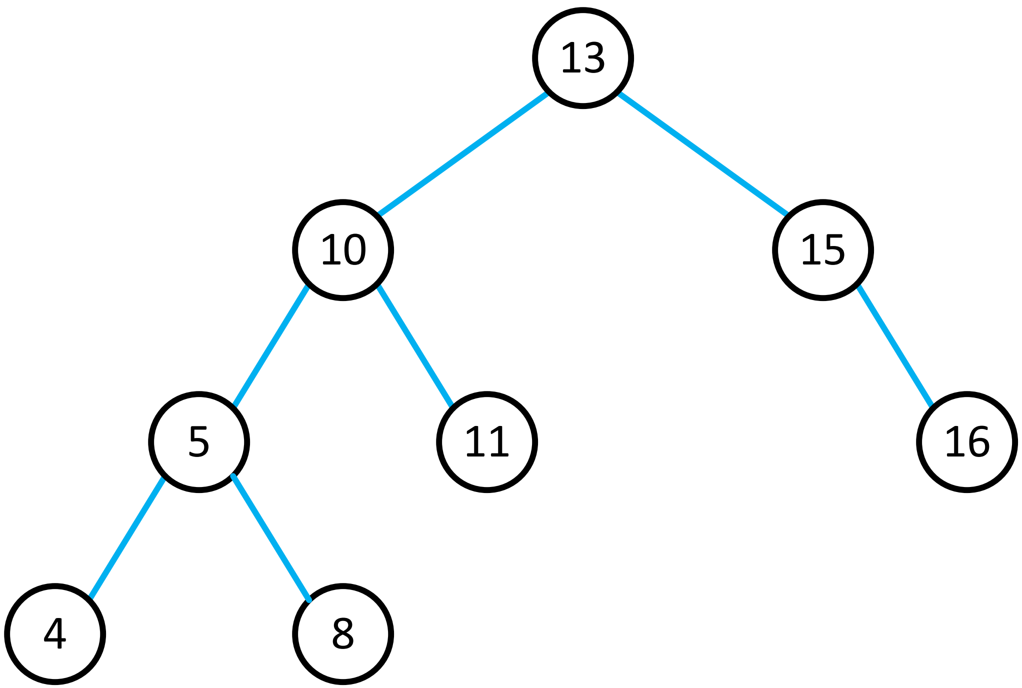
- If $BF(x) > 1$: the left subtree of x is taller
  - If $BF(x \rightarrow left) \geq 0$:
    - Case 1: left-left $\longrightarrow$ right rotation
  - If $BF(x \rightarrow left) < 0$:
    - Case 2: left-right $\longrightarrow$ left-right rotation
- If $BF(x) < -1$, the right subtree of x is taller
  - If $BF(x \rightarrow right) \leq 0$:
    - Case 3: right-right $\longrightarrow$ left rotation
  - If $BF(x \rightarrow right) > 0$:
    - Case 4: right-left $\longrightarrow$ right-left rotation

AVL Tree Insertion

- Insert a node as you would do in a BST.
- Push the visited nodes (nodes that may become unbalanced) onto a stack.
- While the stack is not empty
  - Pop a node from the stack.
  - Update the balance factor of the node.
  - If the node is unbalanced
    - Perform appropriate rotation(s) to rebalance the tree
    - The algorithm is complete after the rotation(s).

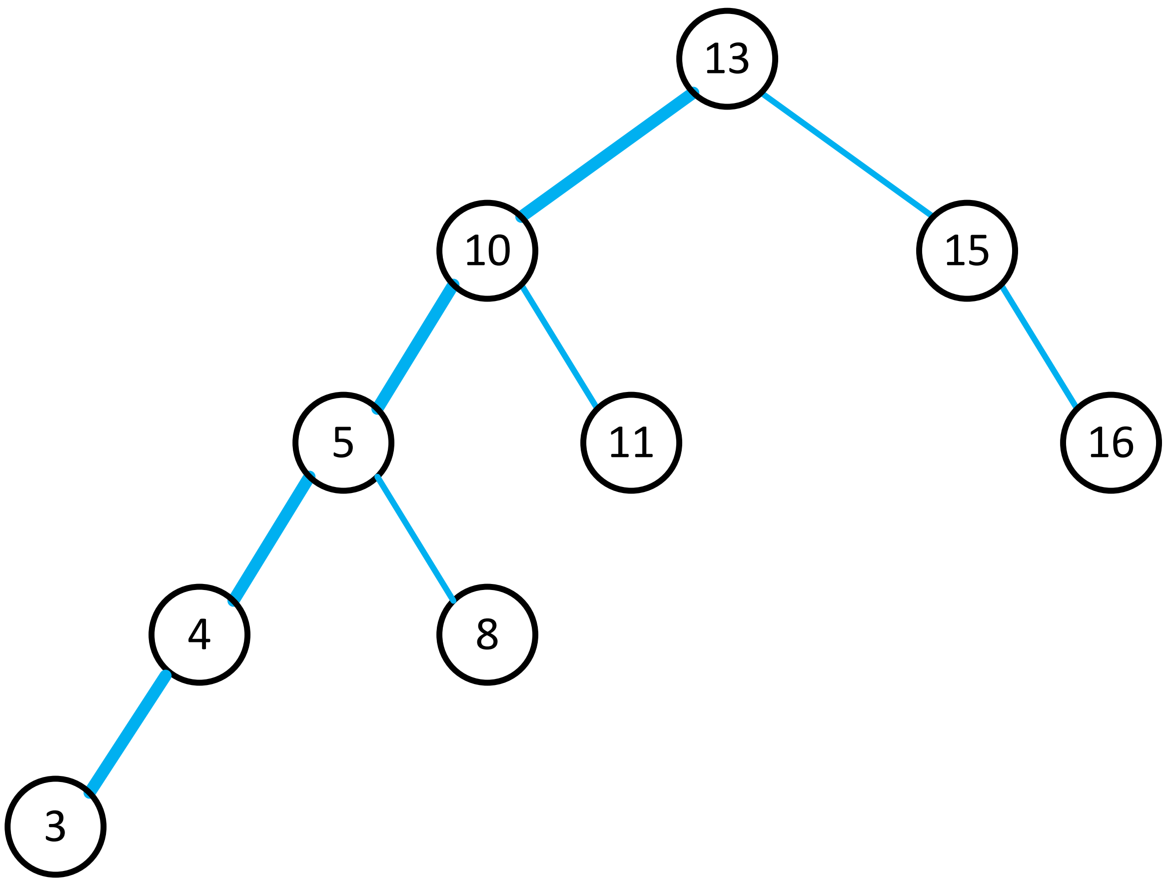
Example 1: AVL Tree Insertion

- Insert 3 into the following AVL tree.



Example 1: AVL Tree Insertion

- Insert 3 and push the visited nodes (13, 10, 5, 4) onto a stack.

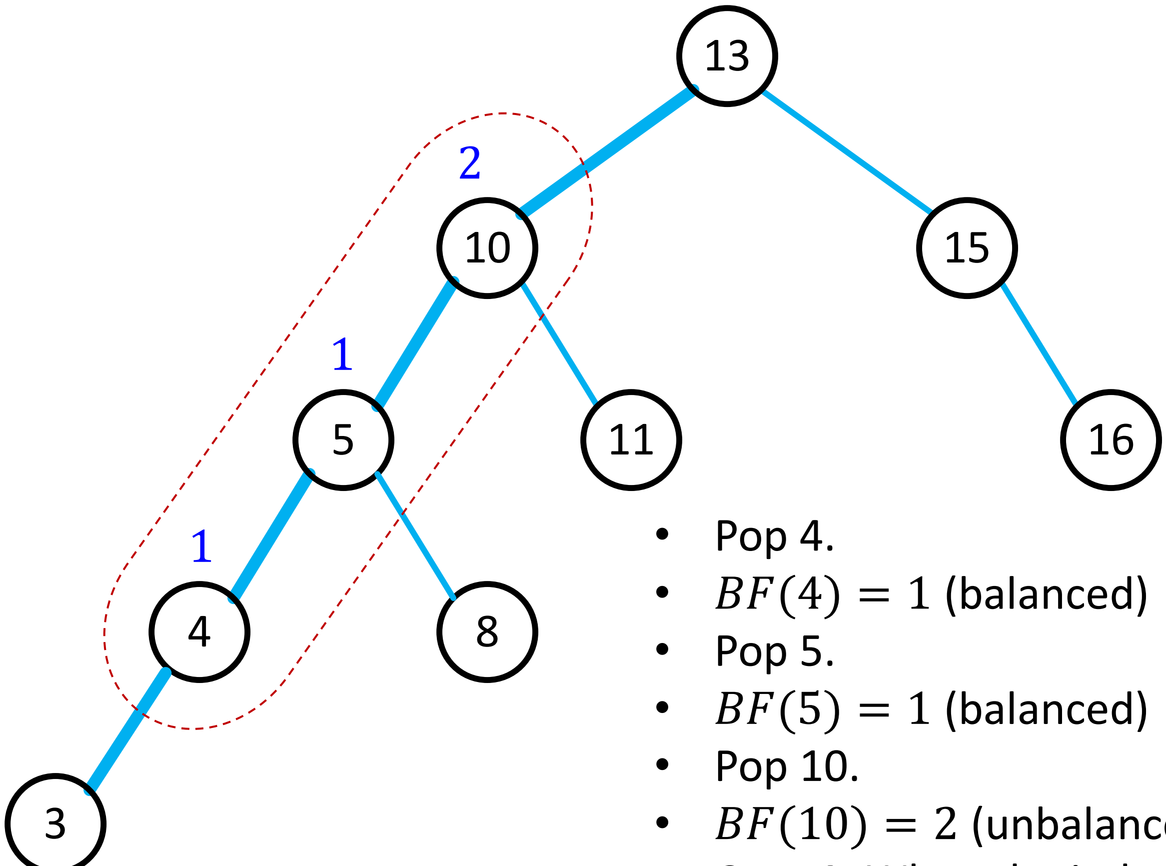


Stack

4
5
10
13

Example 1: AVL Tree Insertion

- Pop a node from the stack, update its balance factor and check if it is unbalanced.



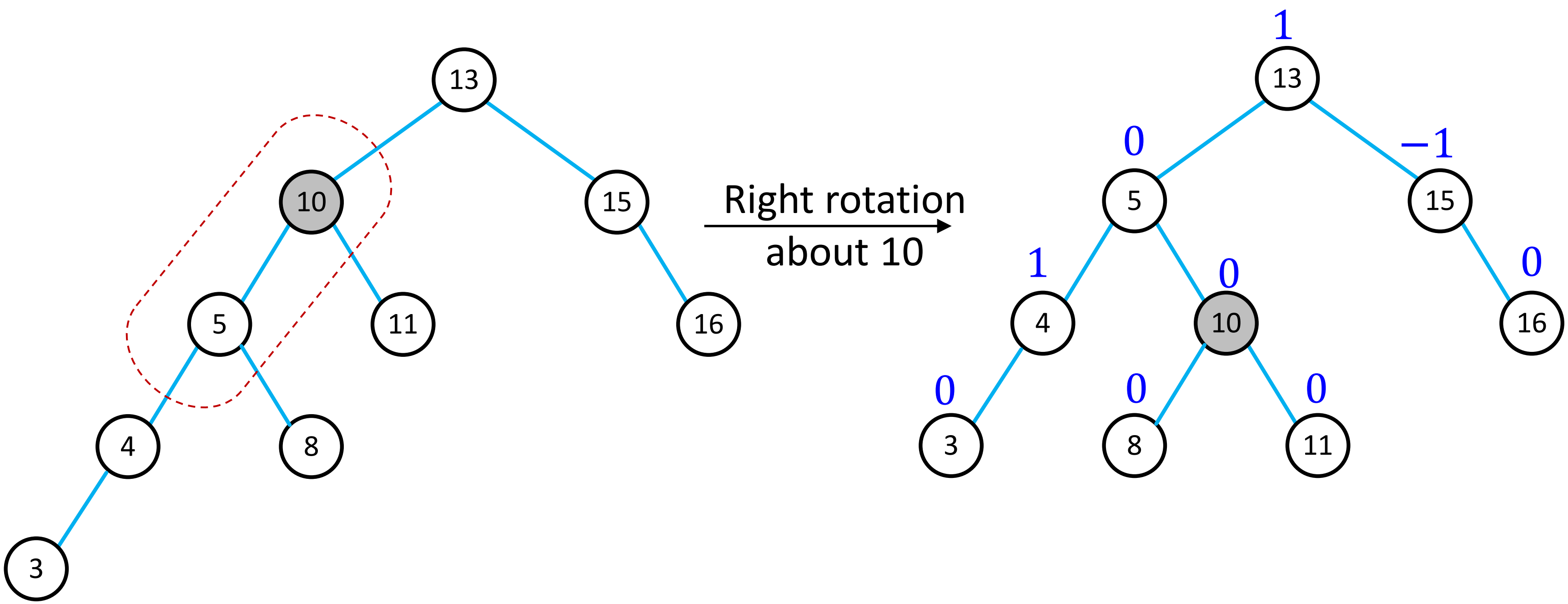
Stack

4
5
10
13

- Pop 4.
- $BF(4) = 1$ (balanced)
- Pop 5.
- $BF(5) = 1$ (balanced)
- Pop 10.
- $BF(10) = 2$ (unbalanced)
- **Case 1**: When the imbalance is caused by the **left** child's **left** subtree $\longrightarrow$ **Right** rotation

Example 1: AVL Tree Insertion

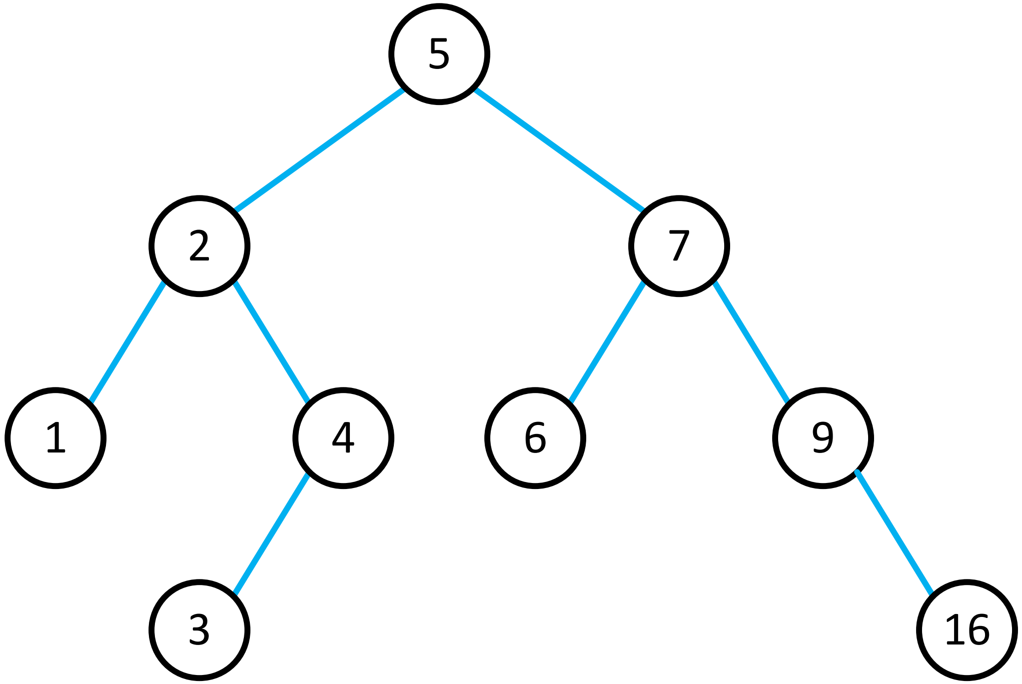
- Rebalance the AVL tree by performing appropriate rotation(s).



Note that the algorithm is complete after performing the rotation even though the stack is not empty.

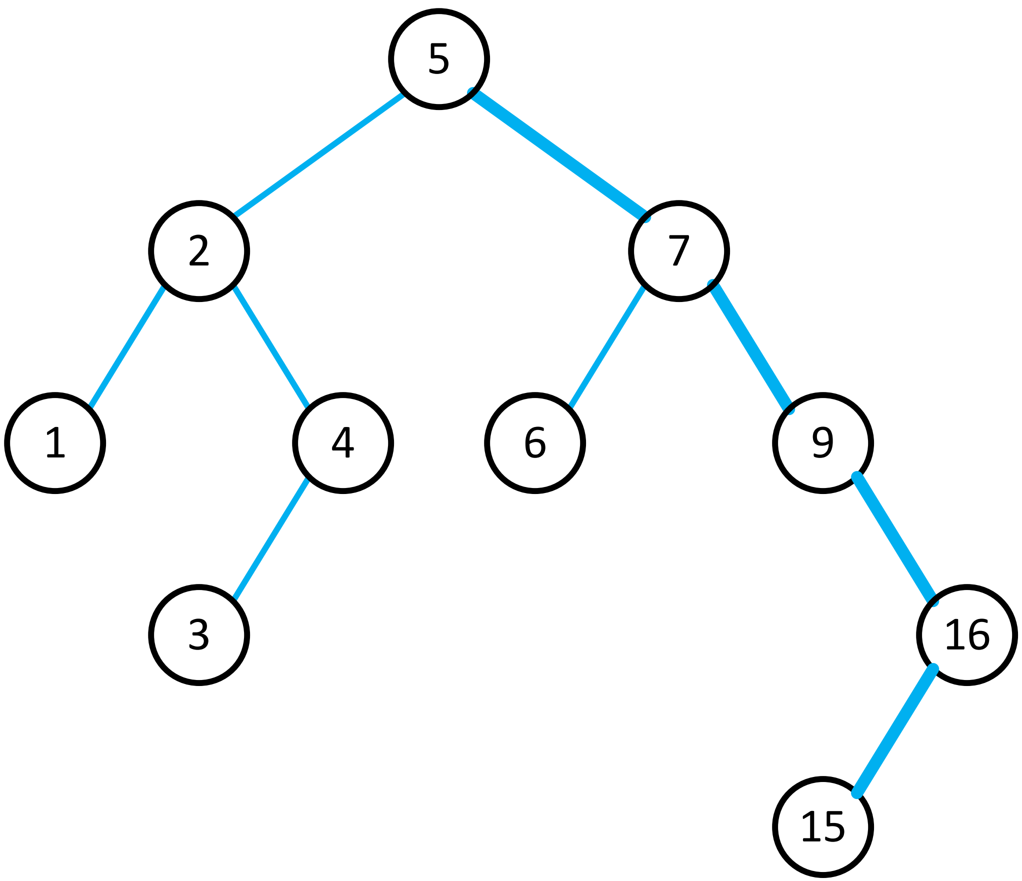
Example 2: AVL Tree Insertion

- Insert 15 into the following AVL tree.



Example 2: AVL Tree Insertion

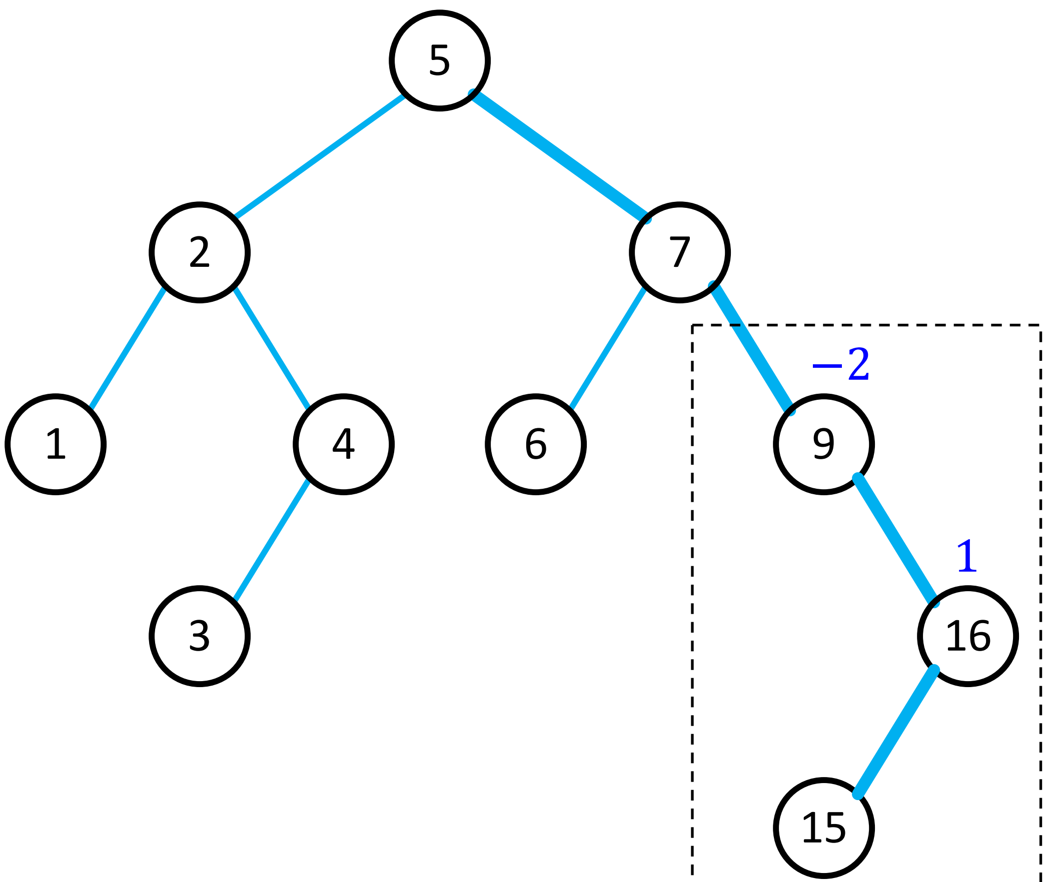
- Insert 15 and push the visited nodes (5, 7, 9, 16) onto a stack.



Stack
16
9
7
5

Example 2: AVL Tree Insertion

- Pop a node from the stack, update its balance factor and check if it is unbalanced.



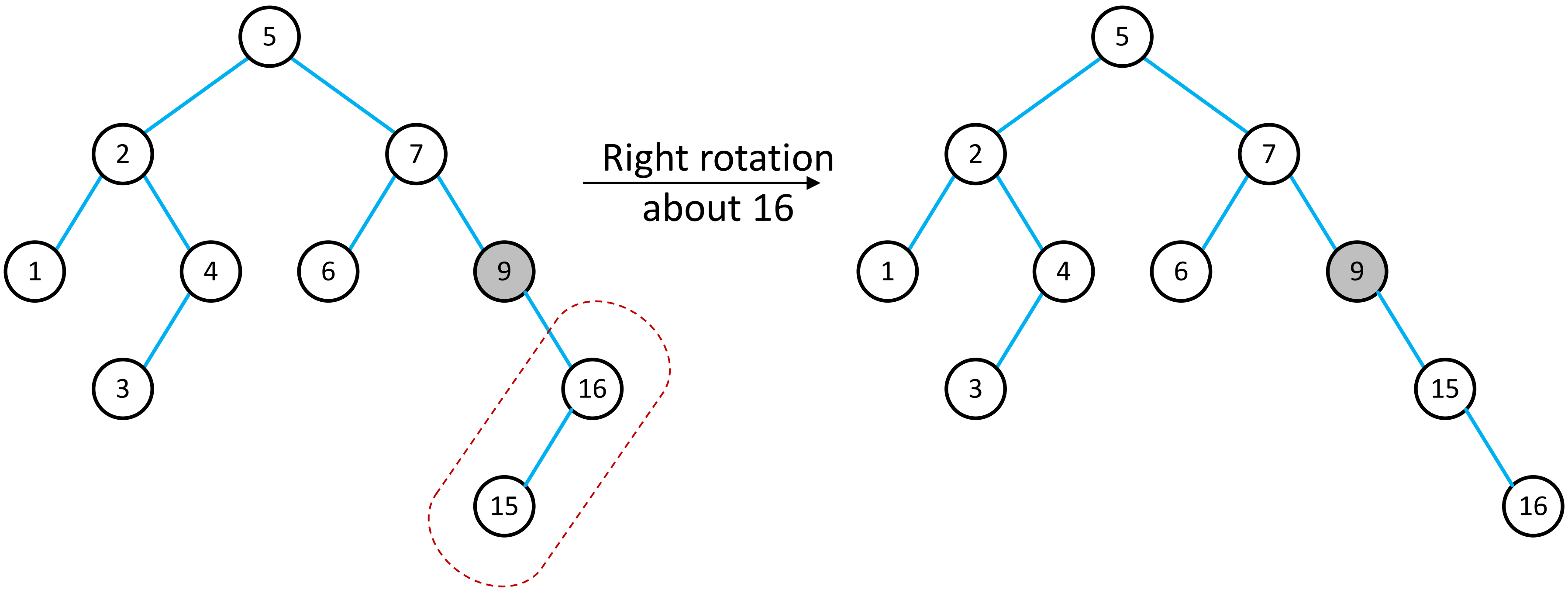
Stack

16
9
7
5

- Pop 16.
- $BF(16) = 1$ (balanced)
- Pop 9.
- $BF(9) = -2$ (unbalanced)
- **Case 4**: When the imbalance is caused by the **right** child's **left** subtree $\longrightarrow$ **Right-left** rotation

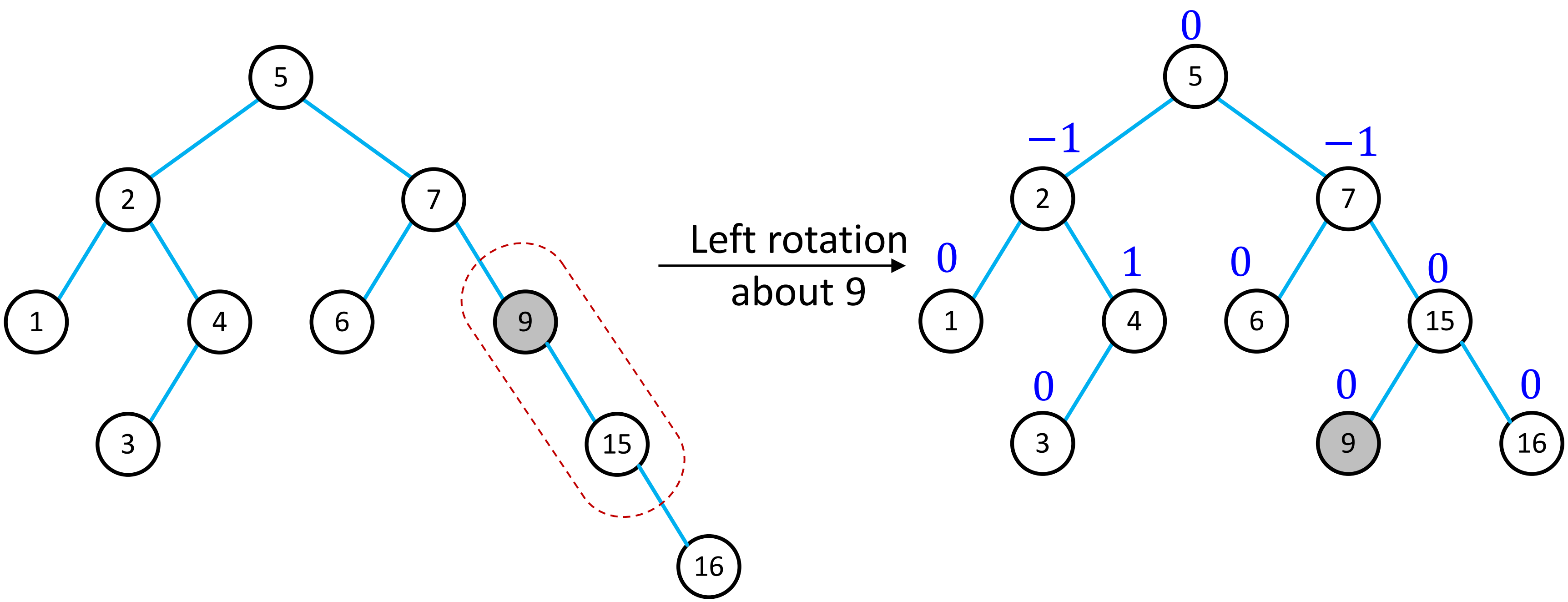
Example 2: AVL Tree Insertion

- Rebalance the AVL tree by performing rotation(s).



Example 2: AVL Tree Insertion

- Rebalance the AVL tree by performing rotation(s).



Note that the algorithm is complete after performing the rotations even though the stack is not empty.

AVL Tree Deletion

- Delete a node as you would do in a BST.
- Push the visited nodes onto a stack.
- While the stack is not empty
  - Pop a node from the stack.
  - Update the balance factor of the node.
  - If the node is unbalanced
    - Perform appropriate rotation(s) to rebalance the tree

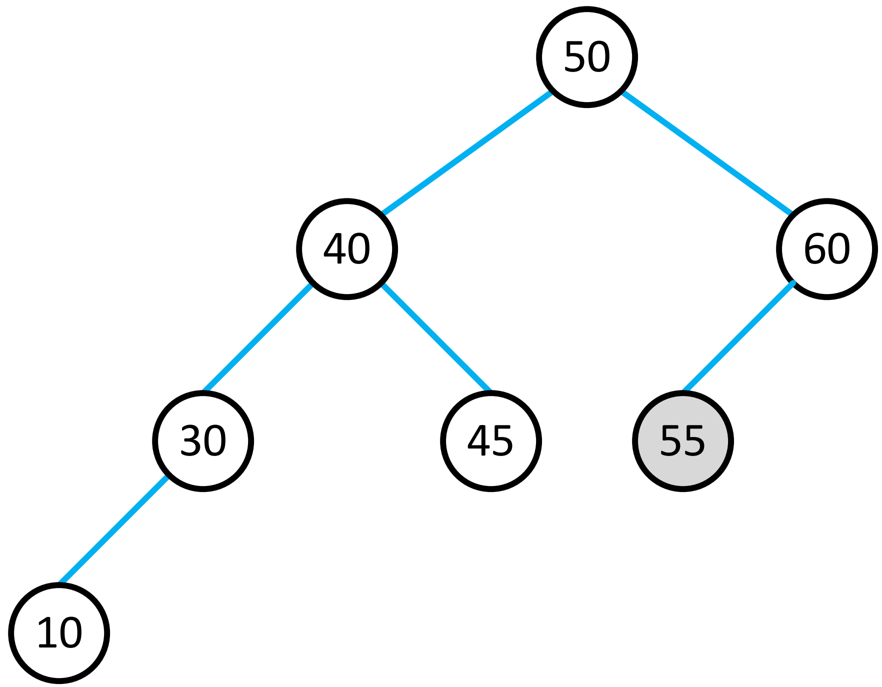
- Note: we need to continue until the stack is empty.

Recall: BST Deletion

- Recall deletion in a BST, there are four cases:
  - If the node to be deleted is a leaf node, then just remove it.
  - If the node to be deleted has only a left child, then replace it with its left child.
  - If the node to be deleted has only a right child, then replace it with its right child.
  - If the node to be deleted has two children, replace its value with that of its in-order predecessor, and remove the node that holds the predecessor.

Example 1: AVL Tree Deletion

- Delete 55 from the following AVL tree.

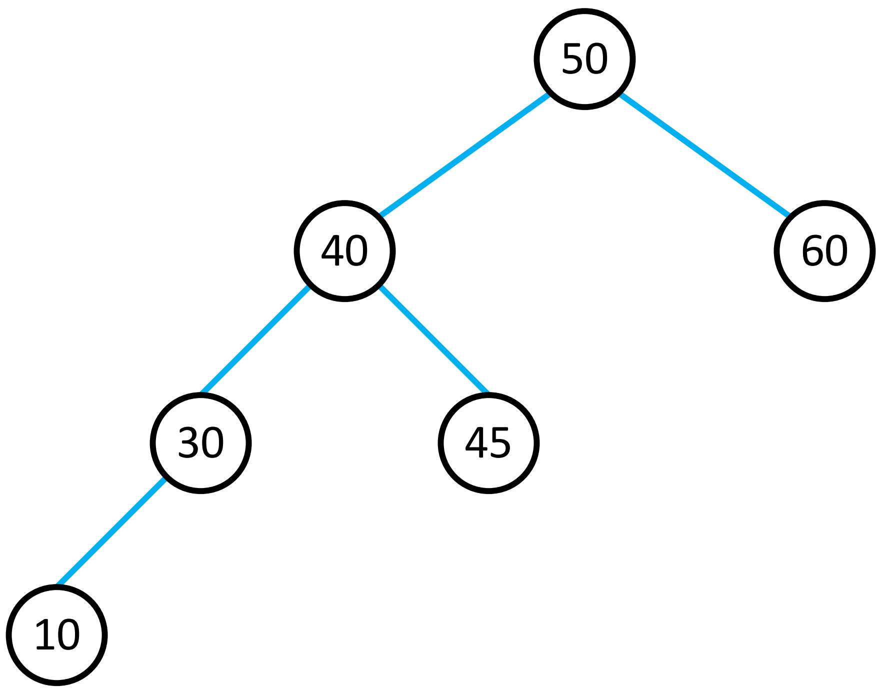


Example 1: AVL Tree Deletion

- Delete 55 (leaf node) and push the visited nodes (50, 60) onto a stack.

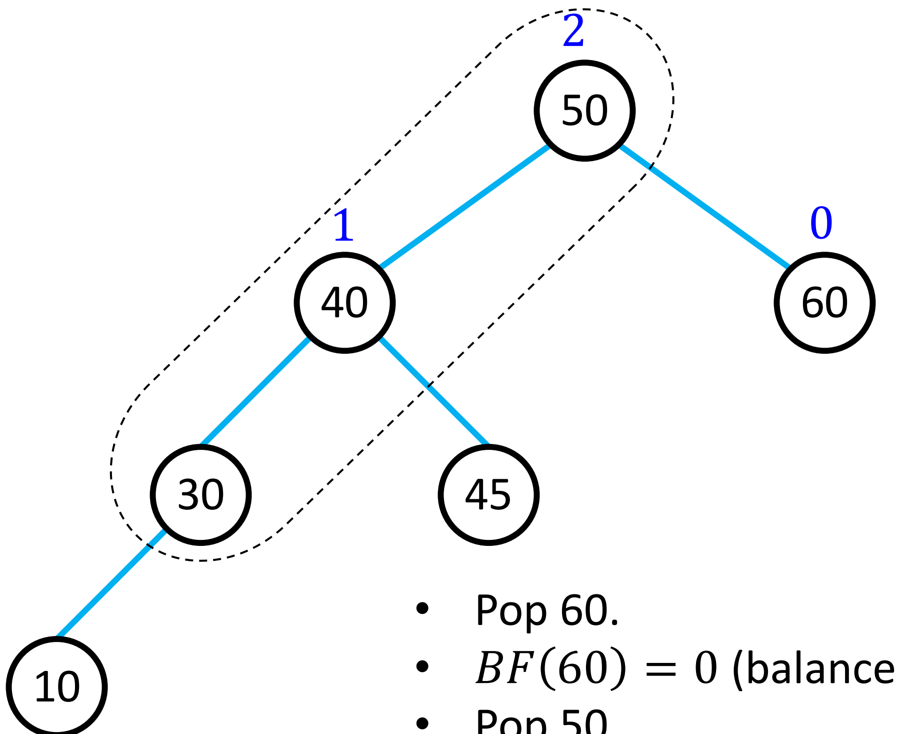
Stack

60
50



Example 1: AVL Tree Deletion

- Pop a node from the stack, update its balance factor and check if it is unbalanced.



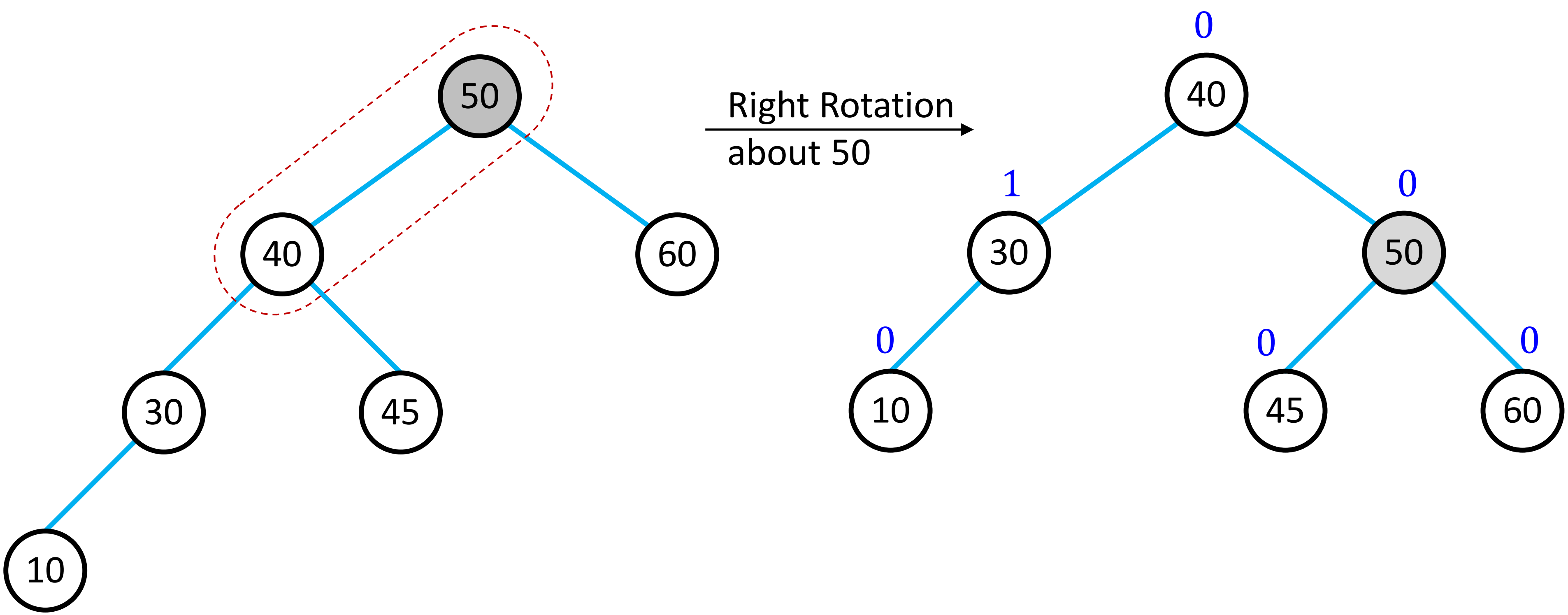
Stack

60
50

- Pop 60.
- $BF(60) = 0$ (balanced)
- Pop 50.
- $BF(50) = 2$ (unbalanced)
- **Case 1**: When the imbalance is caused by the **left** child's **left** subtree. $\longrightarrow$ **Right** rotation

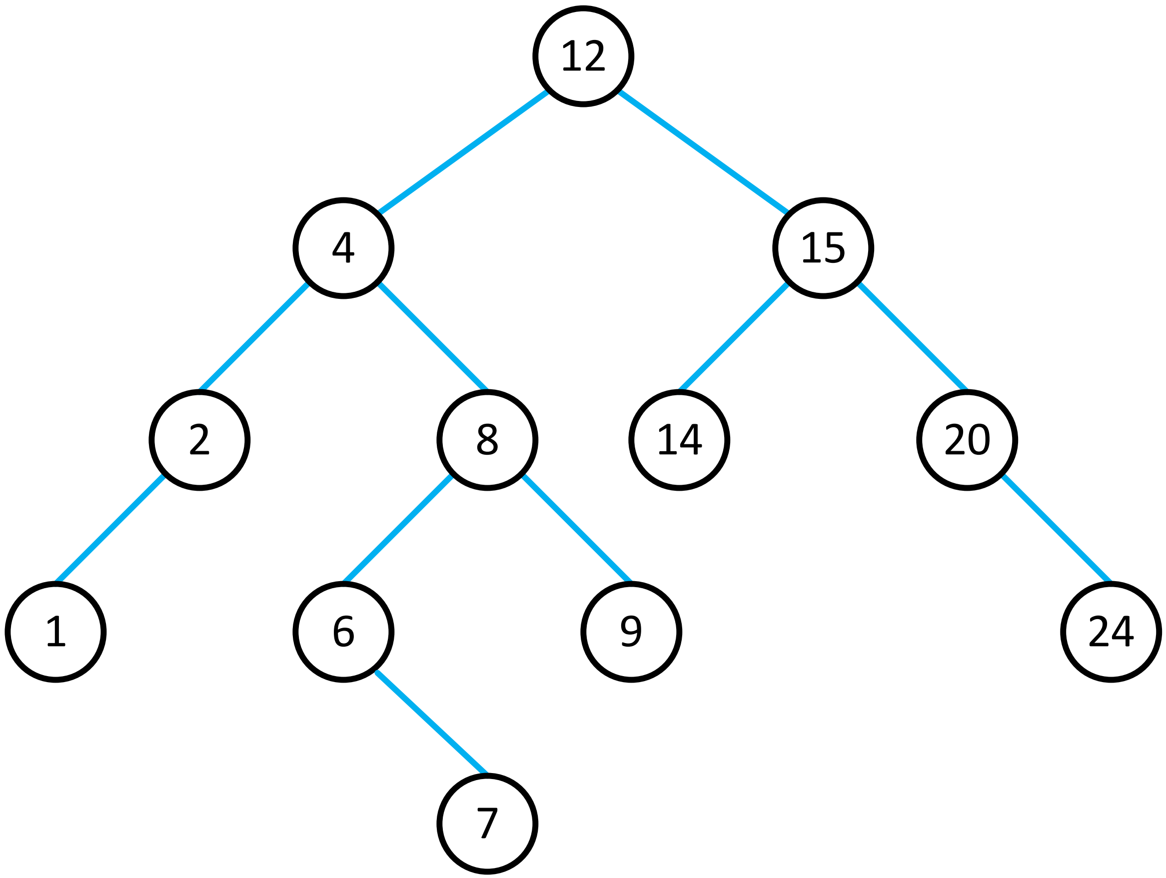
Example 1: AVL Tree Deletion

- Rebalance the AVL tree by performing appropriate rotation(s).



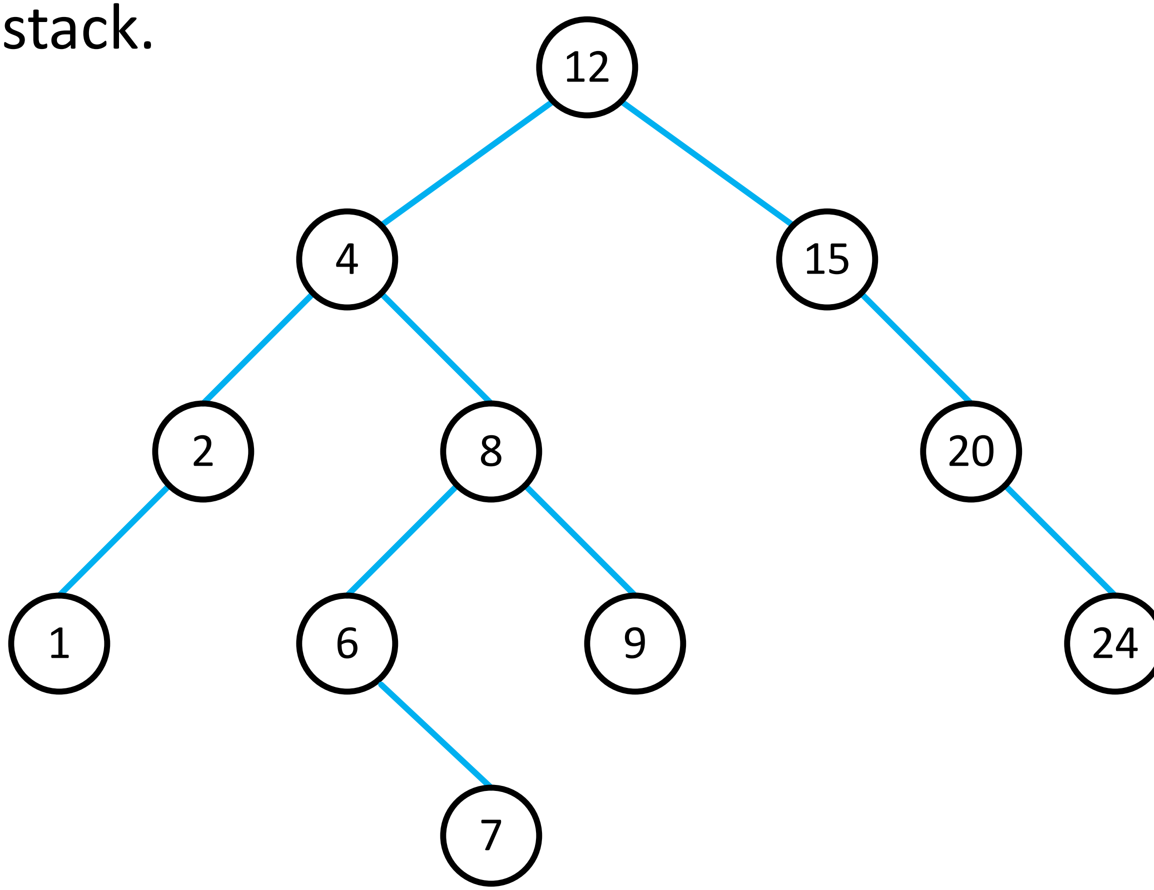
Example 2: AVL Tree Deletion

- Delete 14 from the following AVL tree.



Example 2: AVL Tree Deletion

- Delete 14 (leaf node) and push the visited nodes (12, 15) onto a stack.

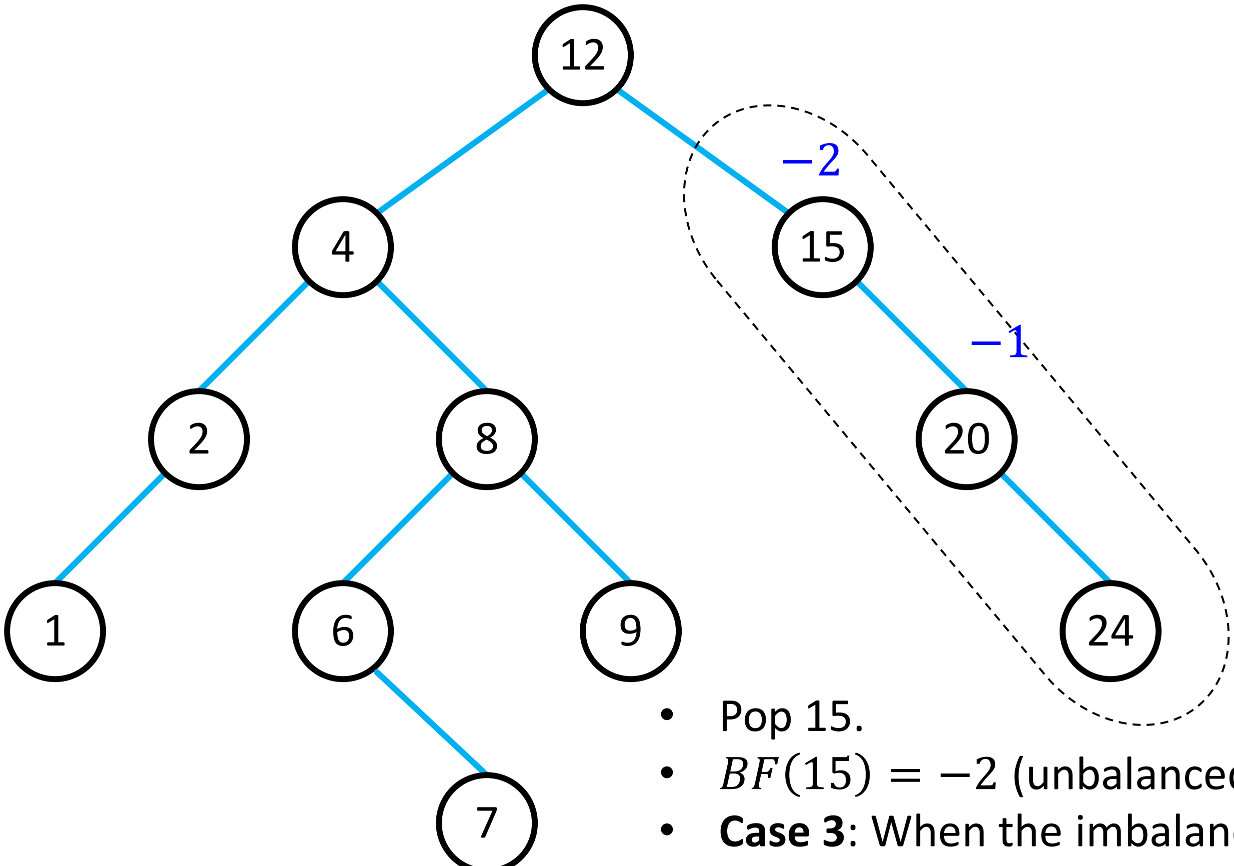


Stack

15
12

Example 2: AVL Tree Deletion

- Pop a node from the stack, update its balance factor and check if it is unbalanced.



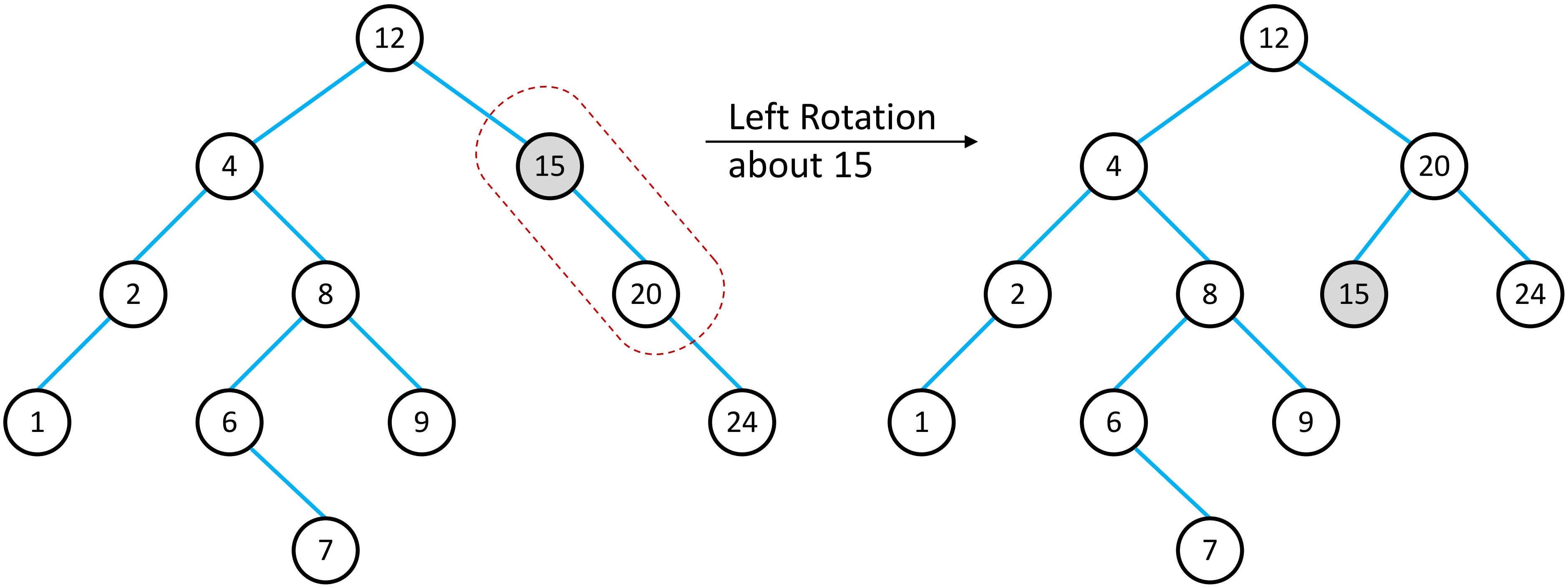
Stack

15
12

- Pop 15.
- $BF(15) = -2$ (unbalanced)
- **Case 3**: When the imbalance is caused by the **right** child's **right** subtree. $\longrightarrow$ **Left** rotation

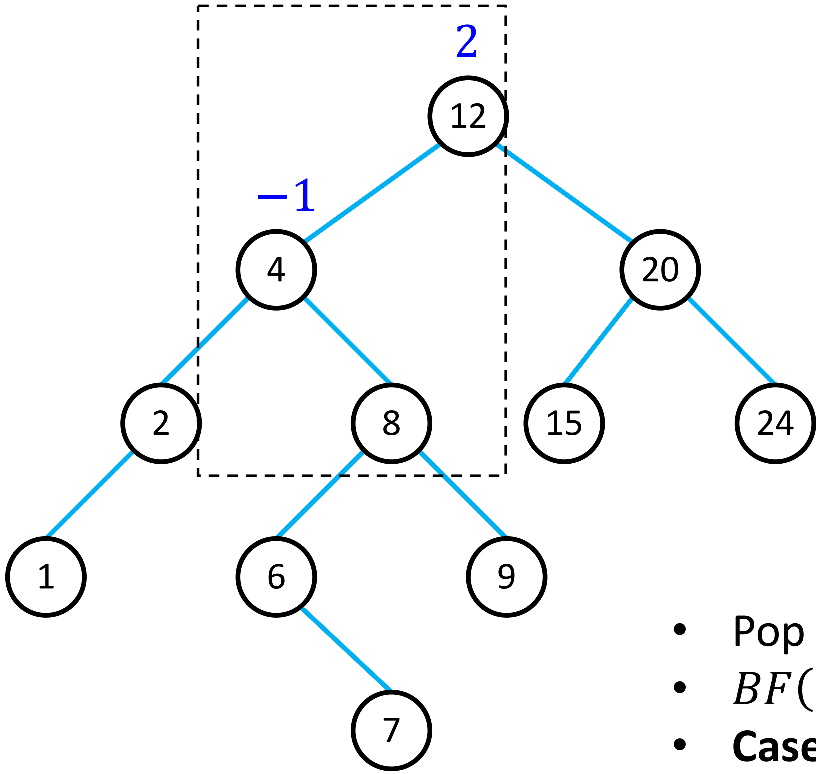
Example 2: AVL Tree Deletion

- Rebalance the AVL tree by performing appropriate rotation(s).

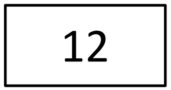


Example 2: AVL Tree Deletion

- Pop a node from the stack, update its balance factor and check if it is unbalanced.



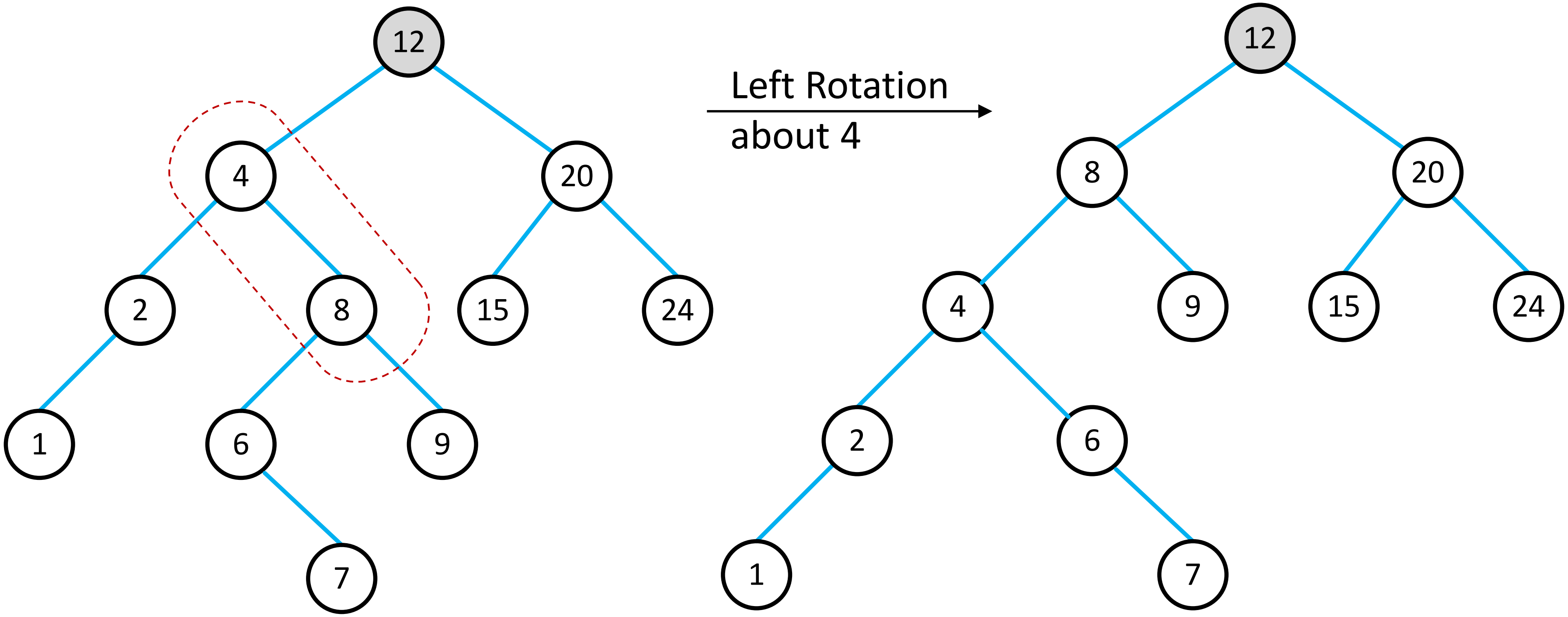
Stack



- Pop 12.
- $BF(12) = 2$ (unbalanced)
- **Case 2**: When imbalance is caused by the **left** child's **right** subtree $\longrightarrow$ **Left-right** rotation

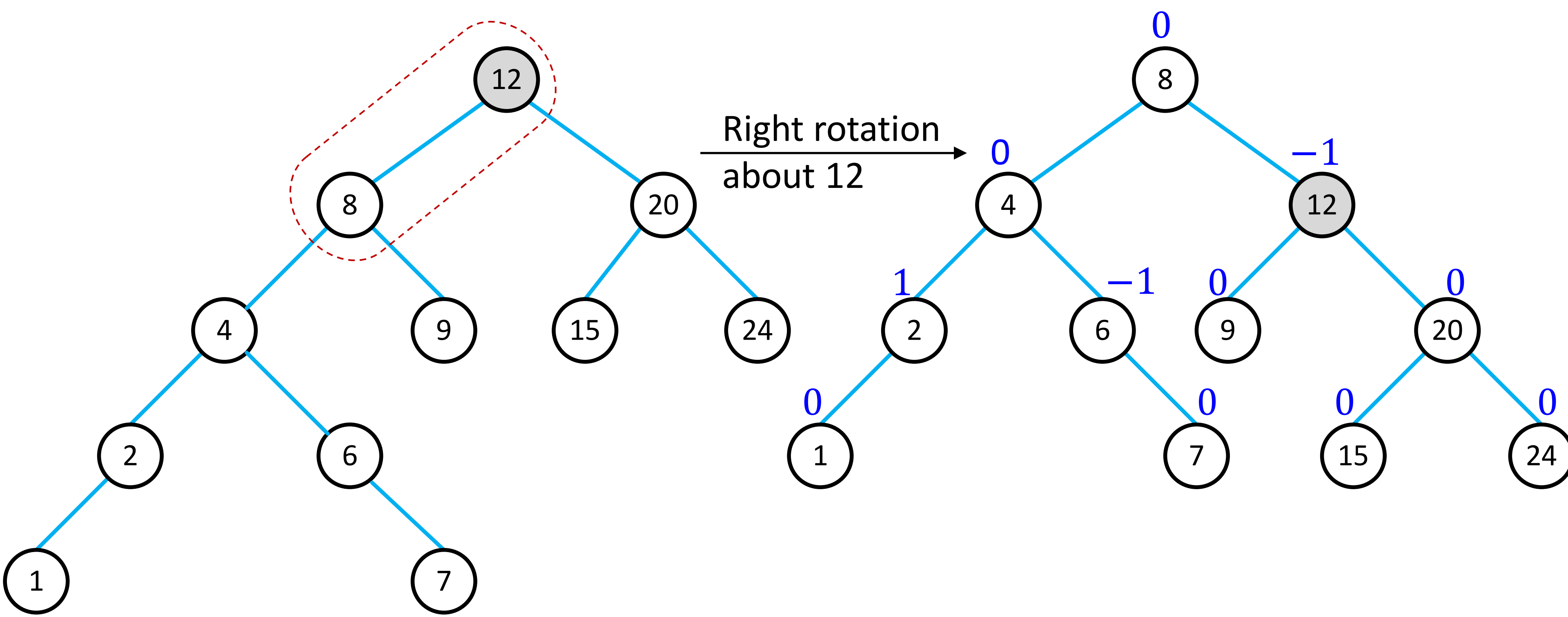
Example 2: AVL Tree Deletion

- Rebalance the AVL tree by performing appropriate rotation(s).



Example 2: AVL Tree Deletion

- Rebalance the AVL tree by performing appropriate rotation(s).



AVL Tree: Insertion

- Recall: Insertion in BST

```
void InsertItem(Tree &tree, int Data){
   if (tree == 0)
      tree = MakeNode(Data);
   else if (Data < tree->data)
      InsertItem(tree->left, Data);
   else if (Data > tree->data)
      InsertItem(tree->right, Data);
   else
      cout << "Error, duplicate item" << endl;
}
```

AVL Tree: Partial Implementation

```
// Client calls this instead of InsertItem
void InsertAVLItem(Tree &tree, int Data){
  stack<Tree> nodes;
  InsertAVLItem2(tree, Data, nodes);
}
```

```
// Auxiliary function with the stack of visited nodes
void InsertAVLItem2(Tree &tree, int Data, stack<Tree> &nodes){
  if (tree == 0){
    tree = MakeNode(Data);
    BalanceAVLTree(nodes); // Balance it now
  }
  else if (Data < tree->data){
    nodes.push(tree); // save visited node
    InsertAVLItem2(tree->left, Data, nodes);
  }
  else if (Data > tree->data){
    nodes.push(tree); // save visited node
    InsertAVLItem2(tree->right, Data, nodes);
  }
  else
    cout << "Error, duplicate item" << endl;
}
```

AVL Tree: Partial Implementation

```
void BalanceAVLTree(stack<Tree> &nodes){
  while (!nodes.empty()){
    Tree node = nodes.top();
    nodes.pop();

    // Implement the algorithm using functions that
    // are already defined (Height, RotateLeft, RotateRight)
}
```

Summary

- Definition of AVL trees
- AVL Tree Operations
  - Insertion
  - Deletion
- Partial Implementation