AVL tree

(**A**delson-**V**elsky and **L**andis)

Invented in 1962 by two Russian mathematicians - Adelson-Velsky and Landis

AVL tree is a **self-balancing Binary Search Tree (BST)**

An AVL tree is a binary search tree that also meets the following rule

AVL condition: For every node, the height of its left subtree and right subtree differ by at most 1

Height is always O(log N)

For every node, maintain its height

- Leaves have height 0
- NULL has “height” -1

Height of a tree: Maximum number of edges on a path from the root to a leaf.

Ordering Invariant

For binary search trees, at any node with key k in a binary search tree, all keys of the elements in the left subtree are strictly less than k, while all keys of the elements in the right subtree are strictly greater than k.

Height Invariant

At any node in the tree, the heights of the left and right subtrees differs by at most 1.

AVL is height invariant

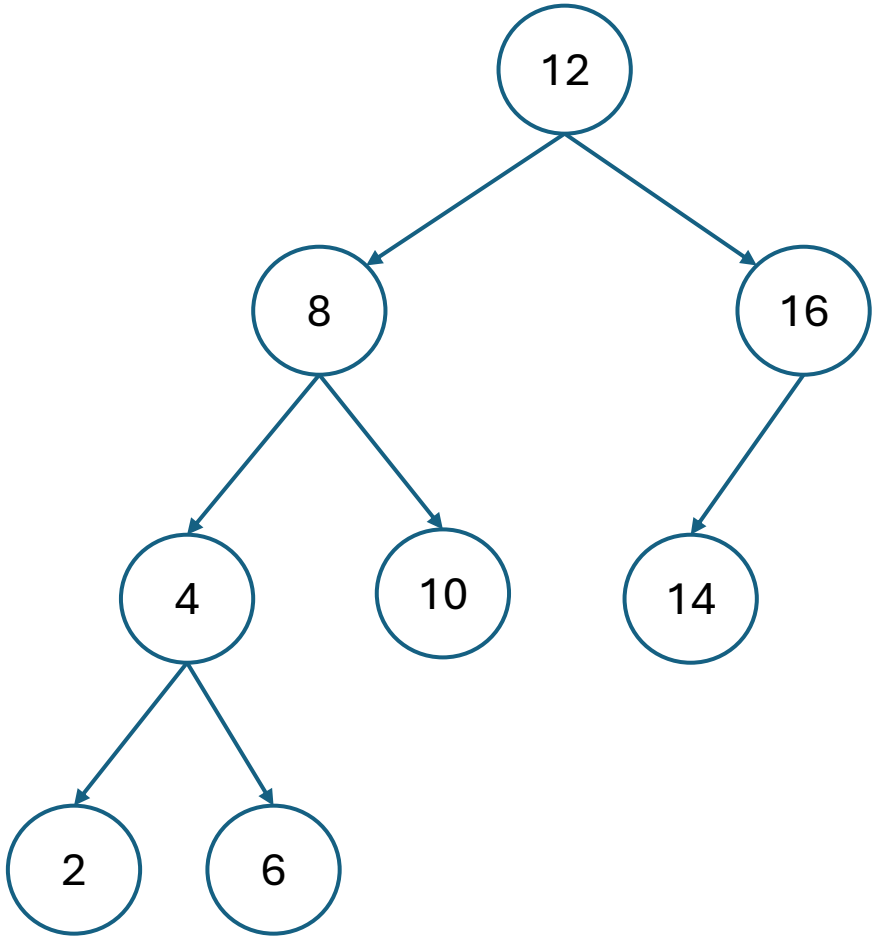
Basic idea:

When nodes are added to / removed from the tree, if the tree becomes unbalanced, repair the tree until balance is restored.

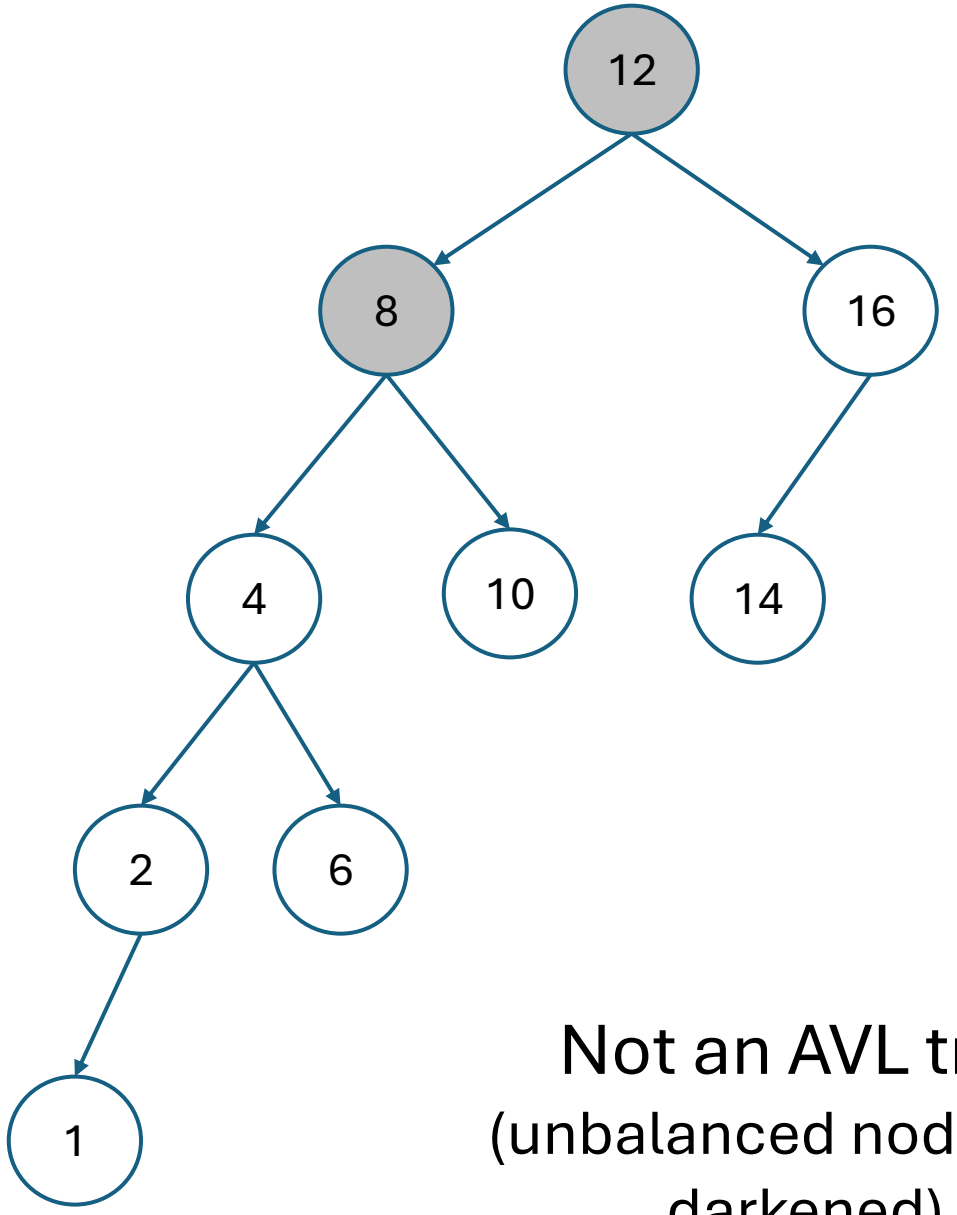
- Rebalancing operations are relatively efficient (O(1))
- Overall tree maintains a balanced O(log N) height

Balance factor

- Balance factor, for a tree node T:
  - = height of T's right subtree minus height of T's left subtree
  - ***BF(T) = Height(T.right) - Height(T.left)***
- An AVL tree maintains a "balance factor"in each node of **0, 1, or -1**
  - i.e. no node's two child subtrees differ in height by more than 1
- Rebalancing required after each insertion and deletion operation
  - To maintain the AVL condition or height invariant property
    - Rotation operations – perform rotation at the node where there is an imbalance



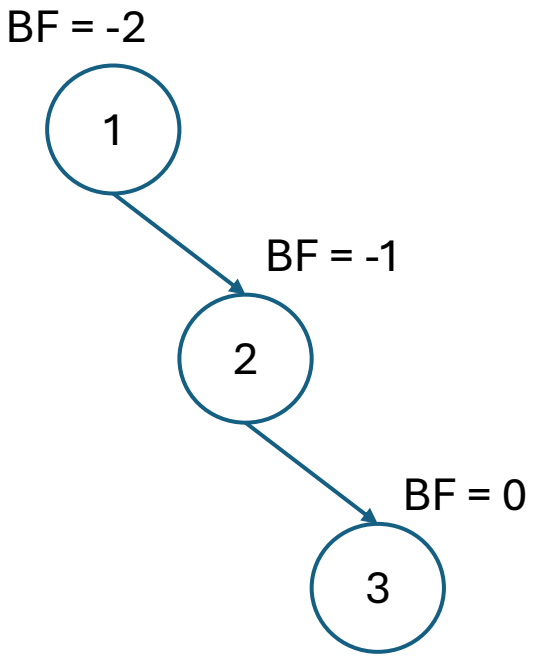
AVL tree

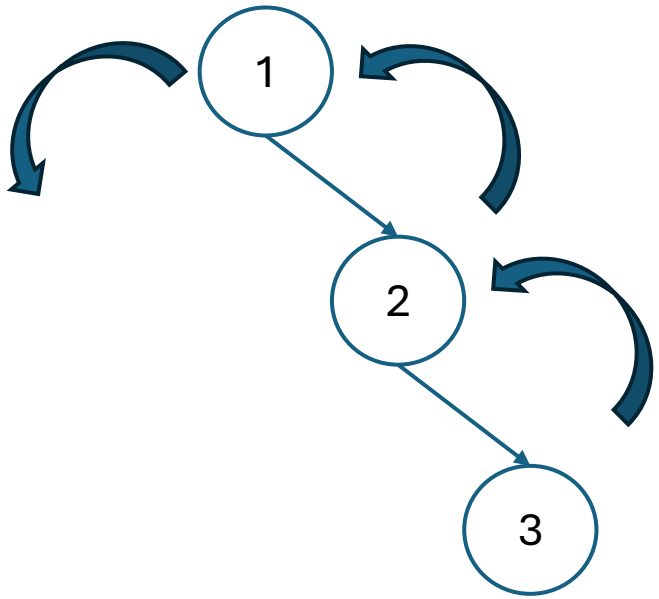


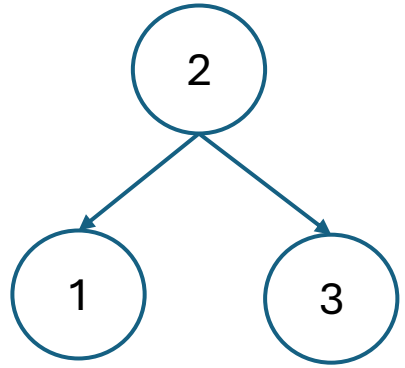
Not an AVL tree
(unbalanced nodes are darkened)

AVL Rotations

Left-Left rotation



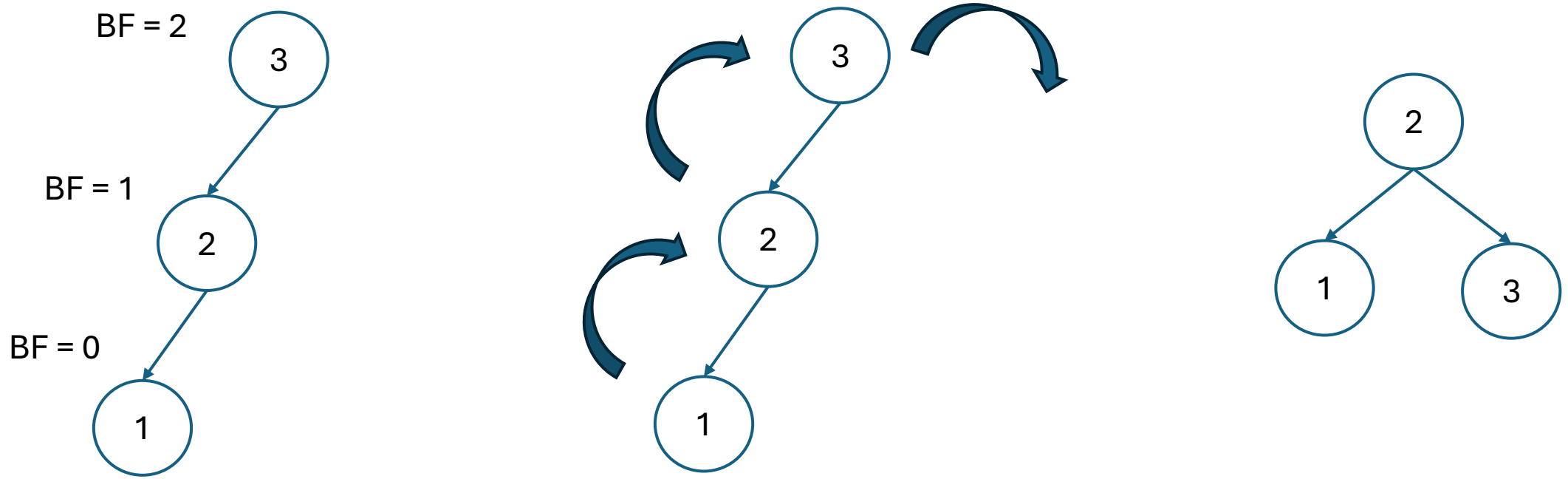




Tree is imbalanced

AVL Rotations

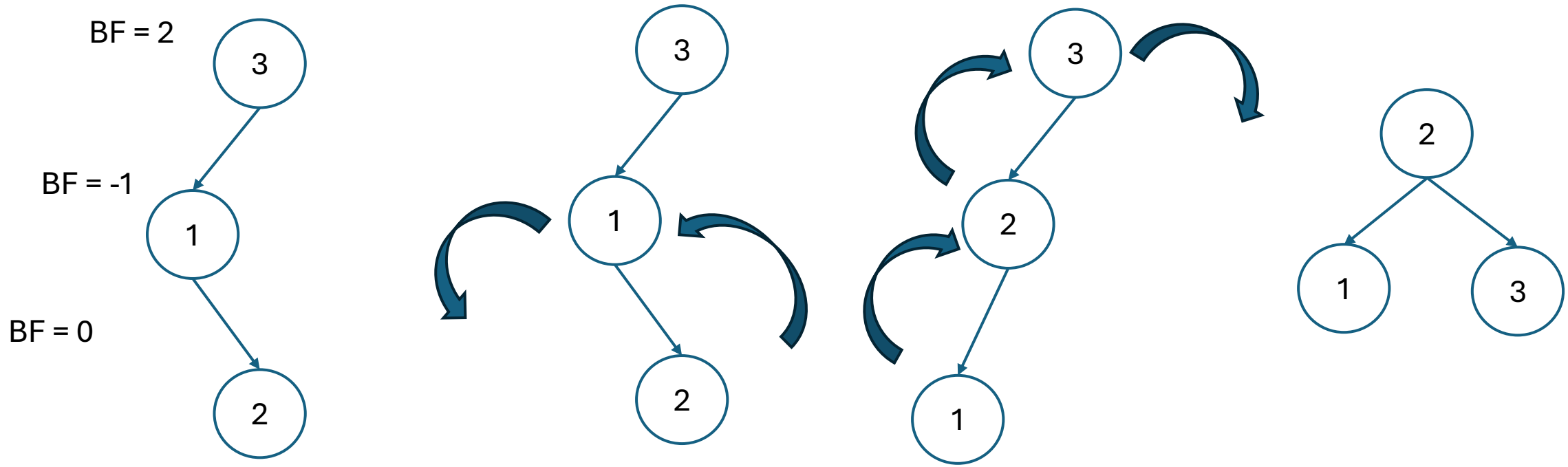
Right-Right rotation



Tree is imbalanced

AVL Rotations

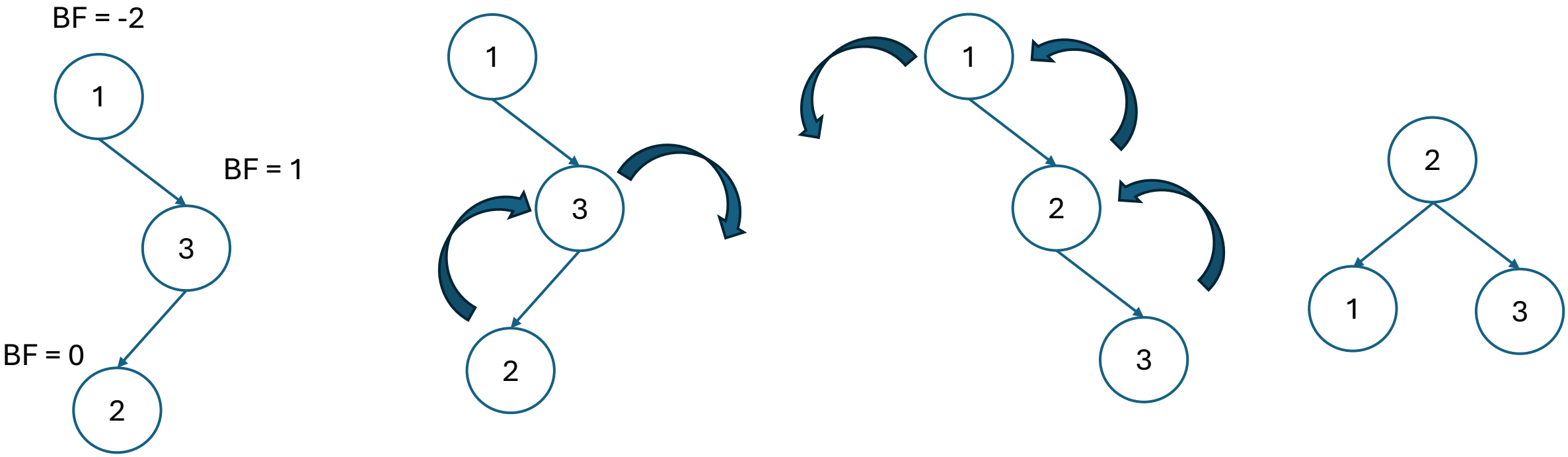
Left - Right rotation



Tree is imbalanced

AVL Rotations

Right - Left rotation

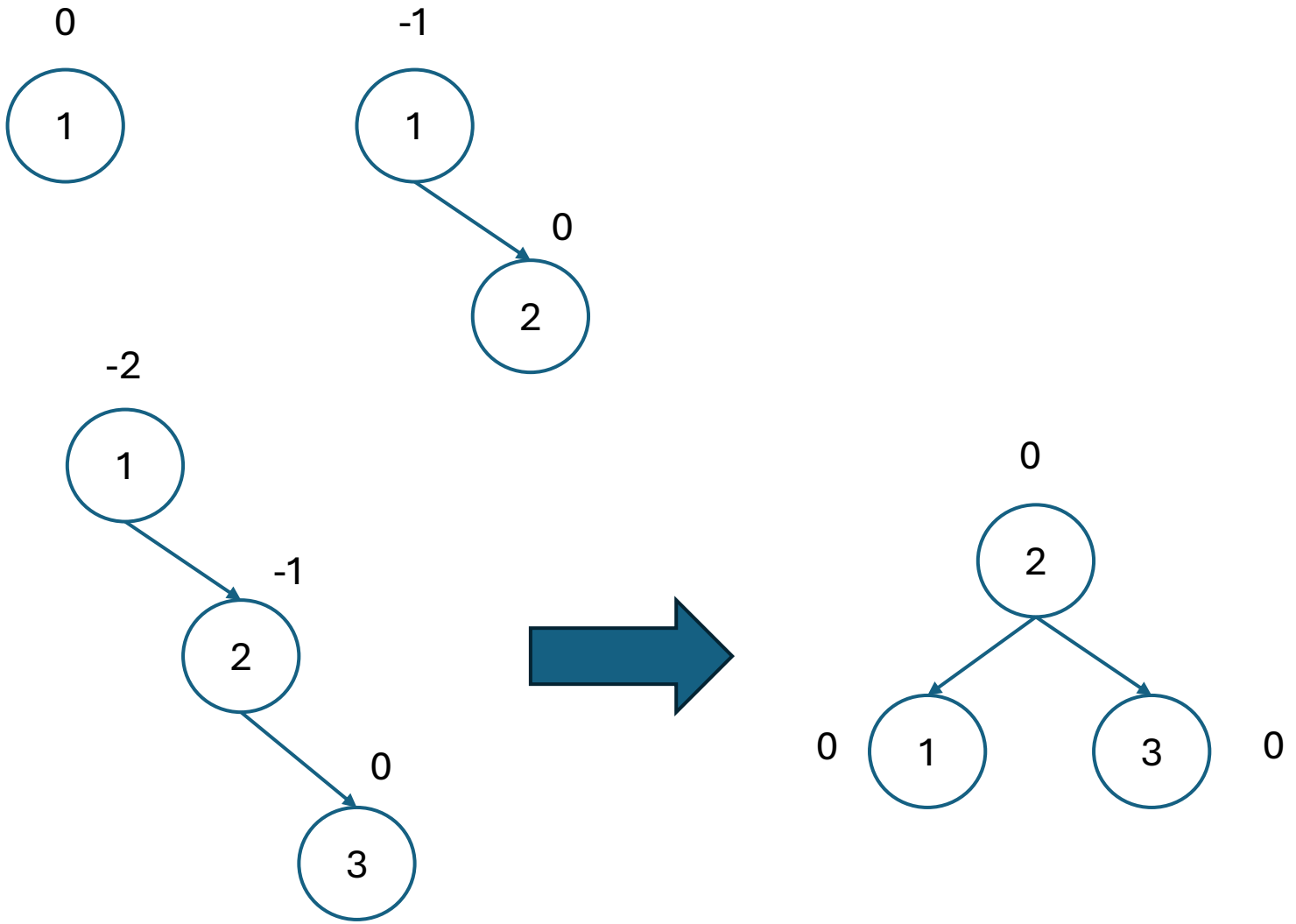


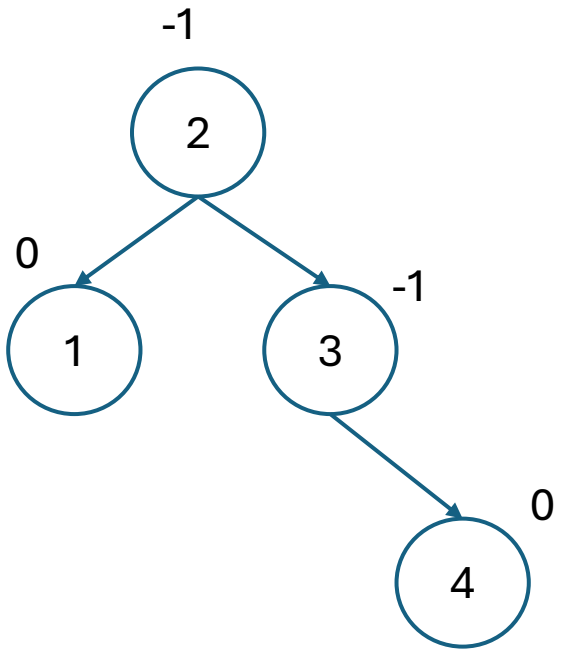
Tree is imbalanced

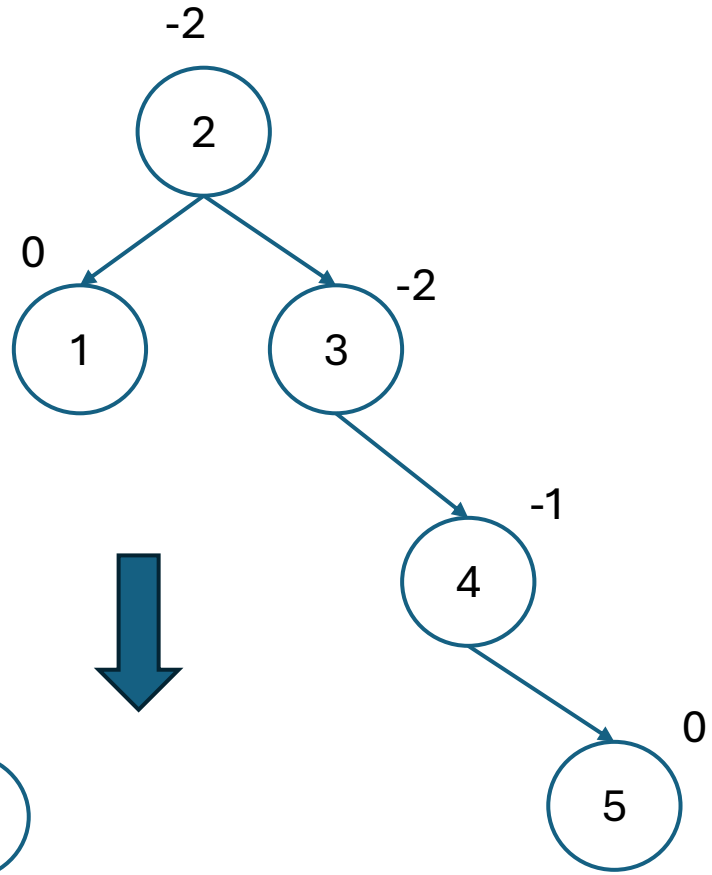
Insertion

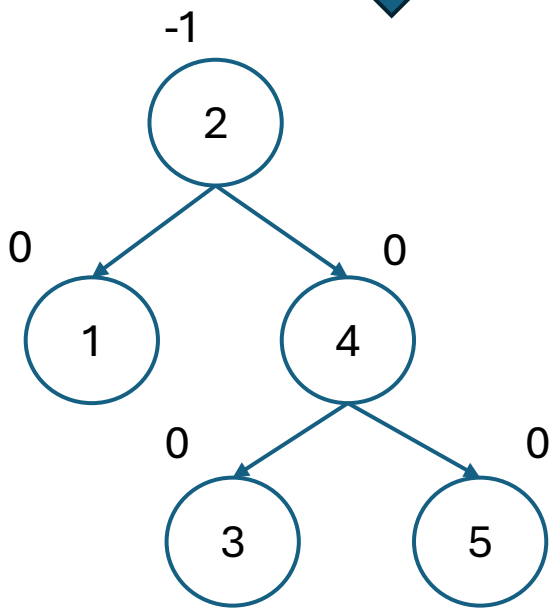
1. Insert the new element into the tree using BST insertion logic
2. Check the balance factor for every node
3. If the balance factor of every node is 1, 0 or -1, go for the next operation
4. If the balance factor of any node is other than 1, 0 or -1, then the tree is imbalanced and hence go for the respective rotation operation to make it balanced

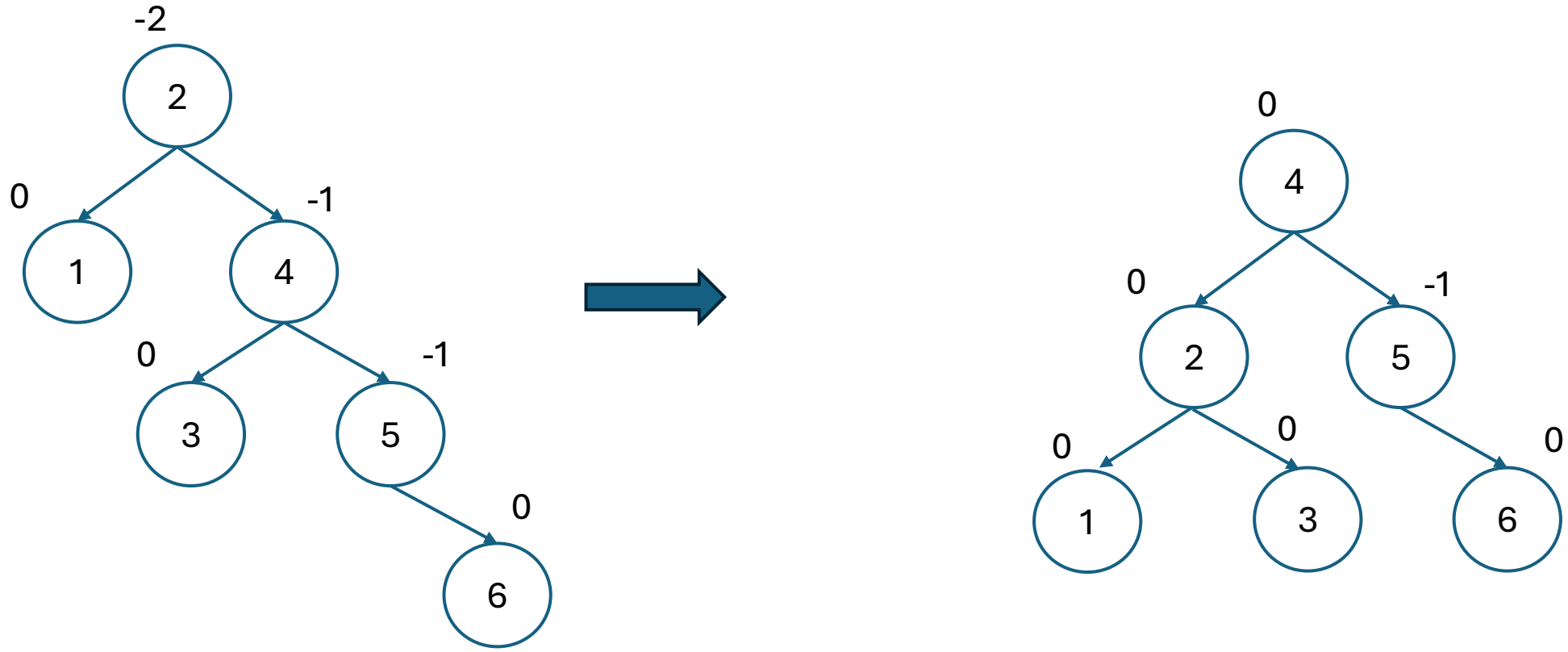
- Insert the elements 1, 2, 3, 4, 5, 6




-1
2
0
1
-1
3
0
4


-2
2
0
1
-2
3
-1
4
0
5


-1
2
0
1
0
4
0
3
0
5


-2
2
0
1
-1
4
0
3
-1
5
0
6
0
4
0
2
-1
5
0
1
0
3
0
6