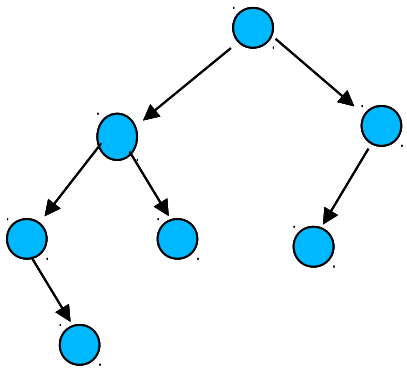
Computer Science 112
Data Structures

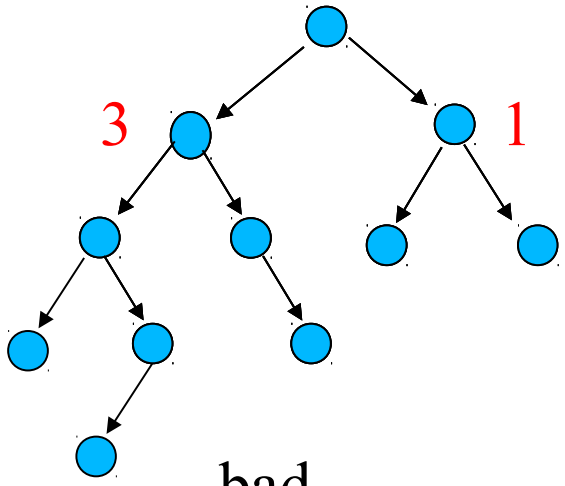
Lecture 13:
More AVL Trees

Review: AVL Trees

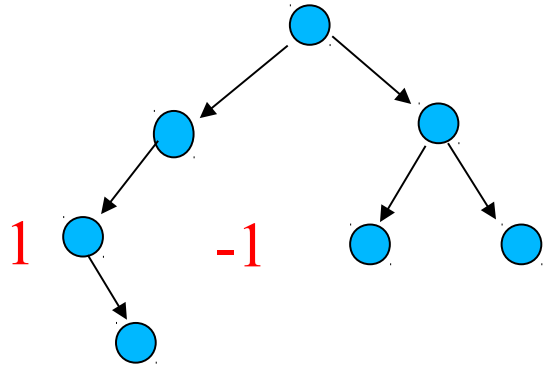
- **Binary Search Tree**
- **Almost balanced**
  - **At every node, subtree heights same +/- 1**



good



bad

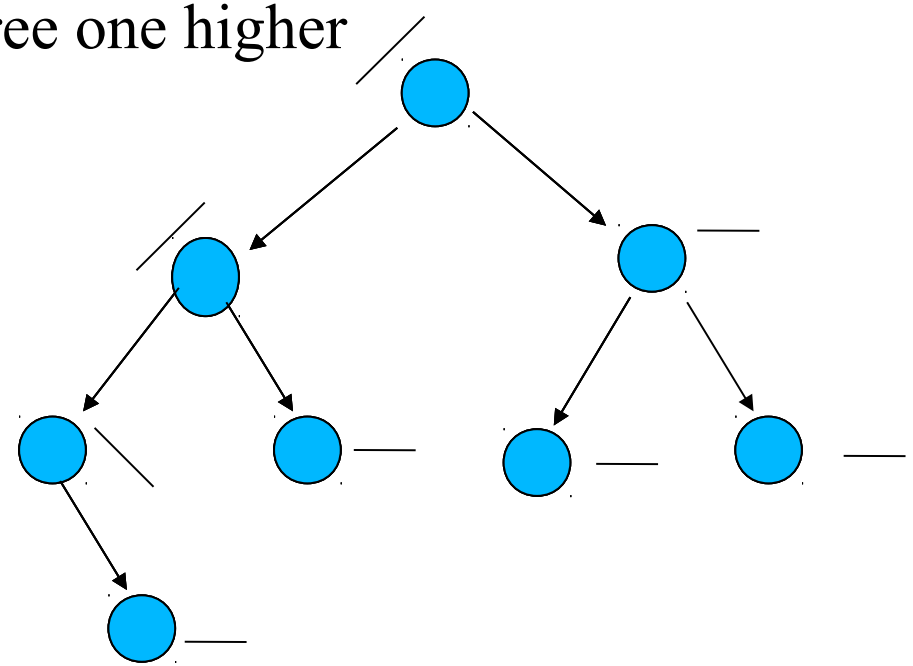


bad

Labeling an AVL Tree

- **Label each node as**

  - left & right subtrees equally high
  \ right subtree one higher
  / left subtree one higher

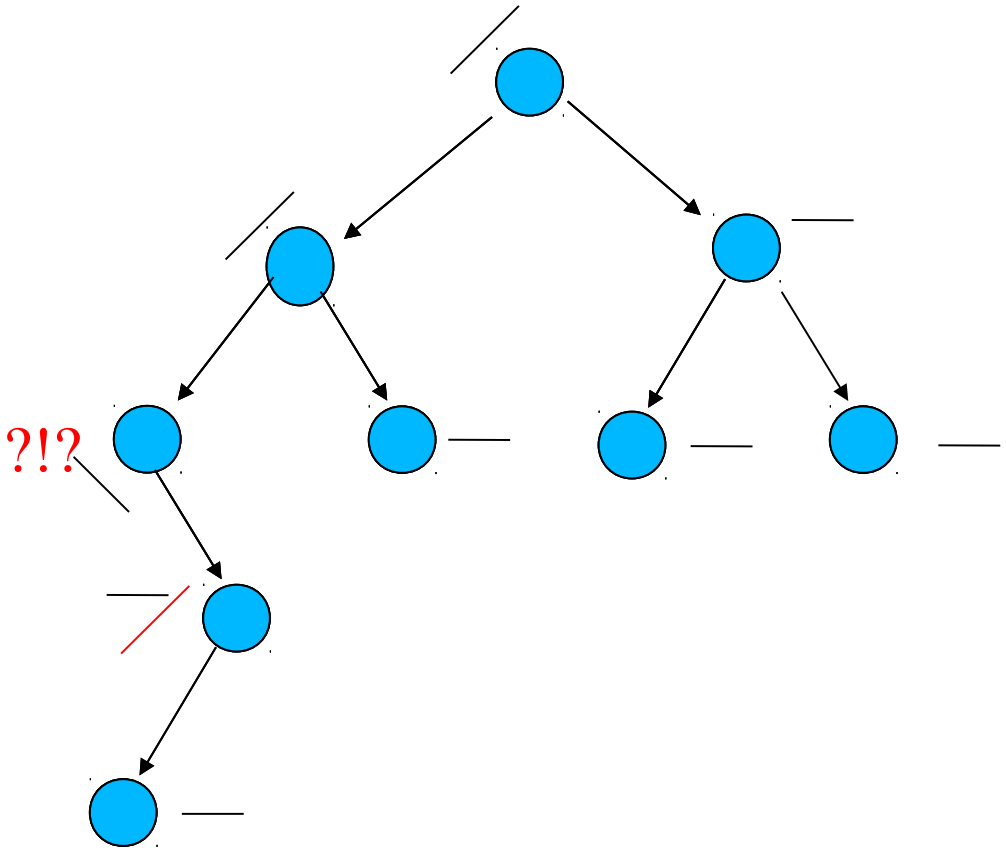


AVL Node

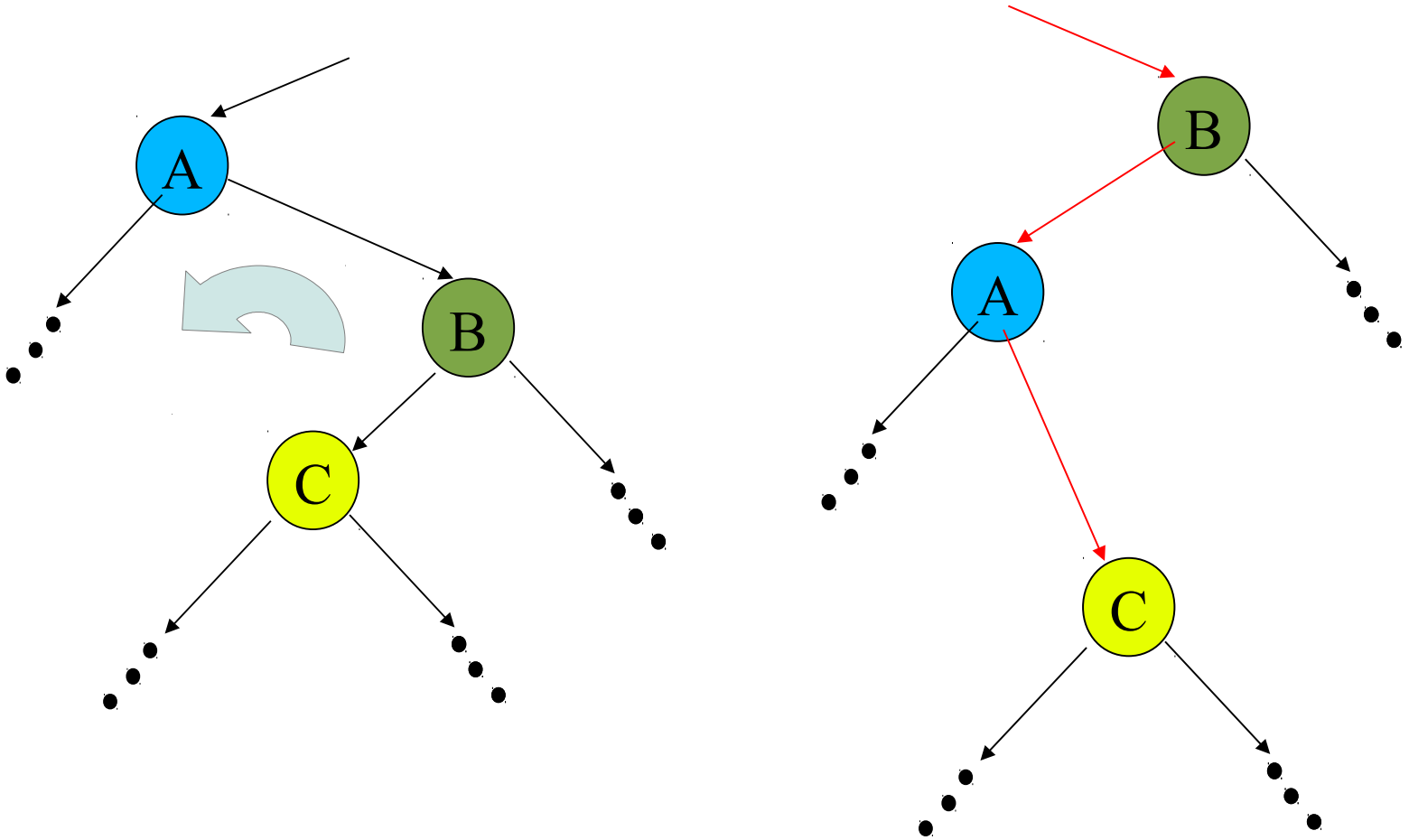
- As in BSTNode
  - left subtree, right subtree, data
- Also:
  - label
  - parent (may be convenient)
  - height (may be convenient)

Rebalancing

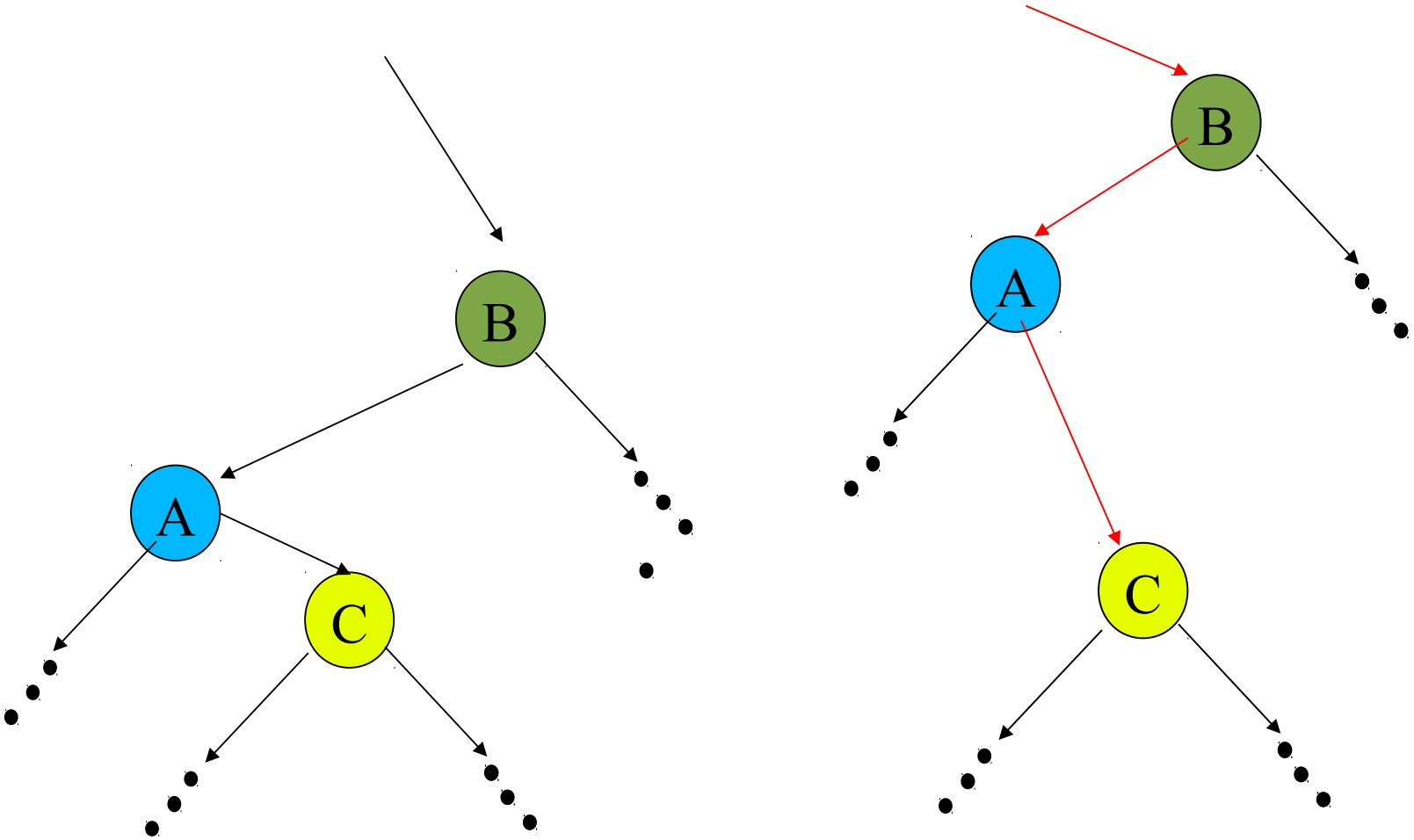
- **Problem: insert/delete -> not balanced**



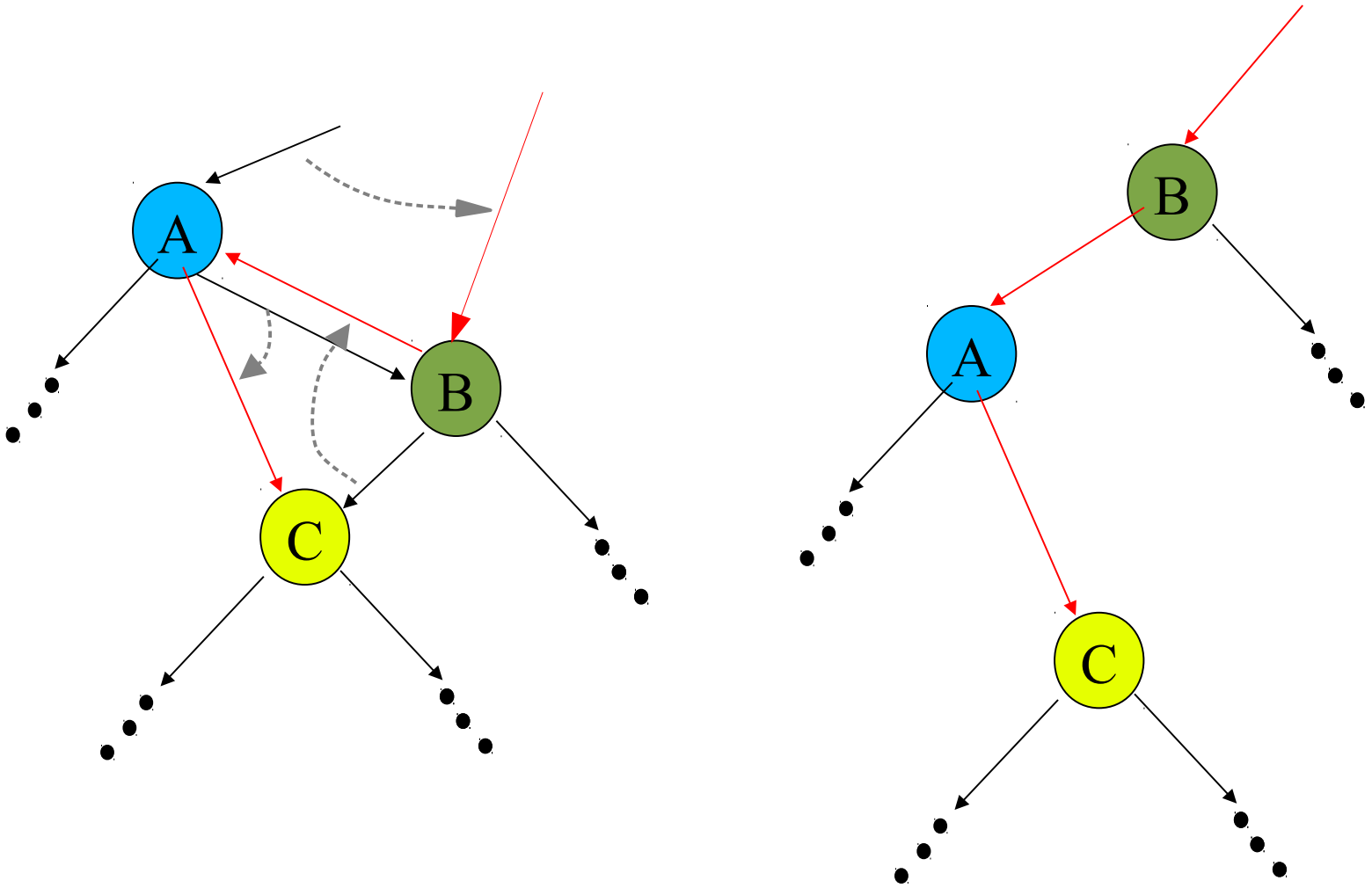
Rotation



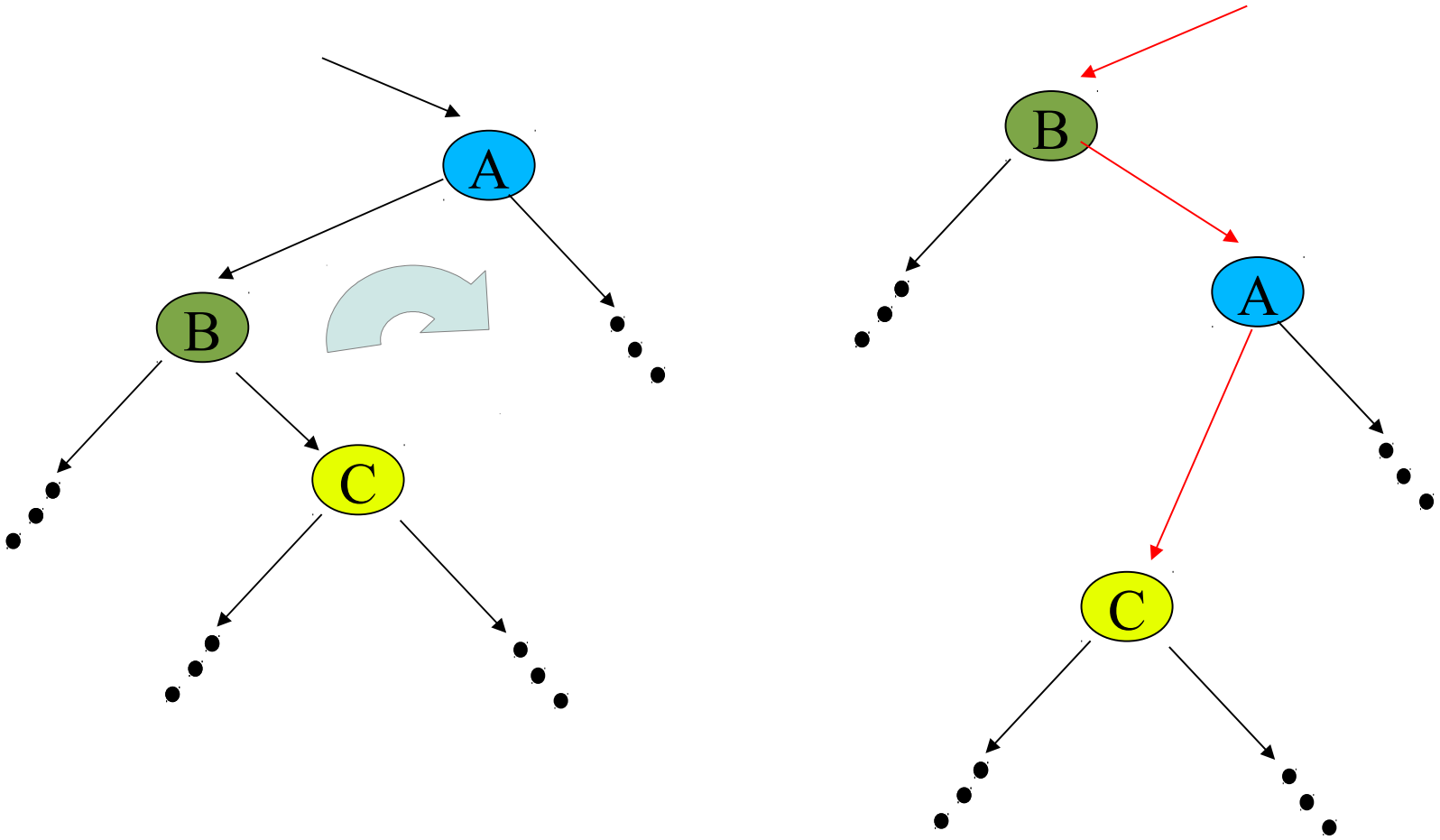
Rotation



Rotation

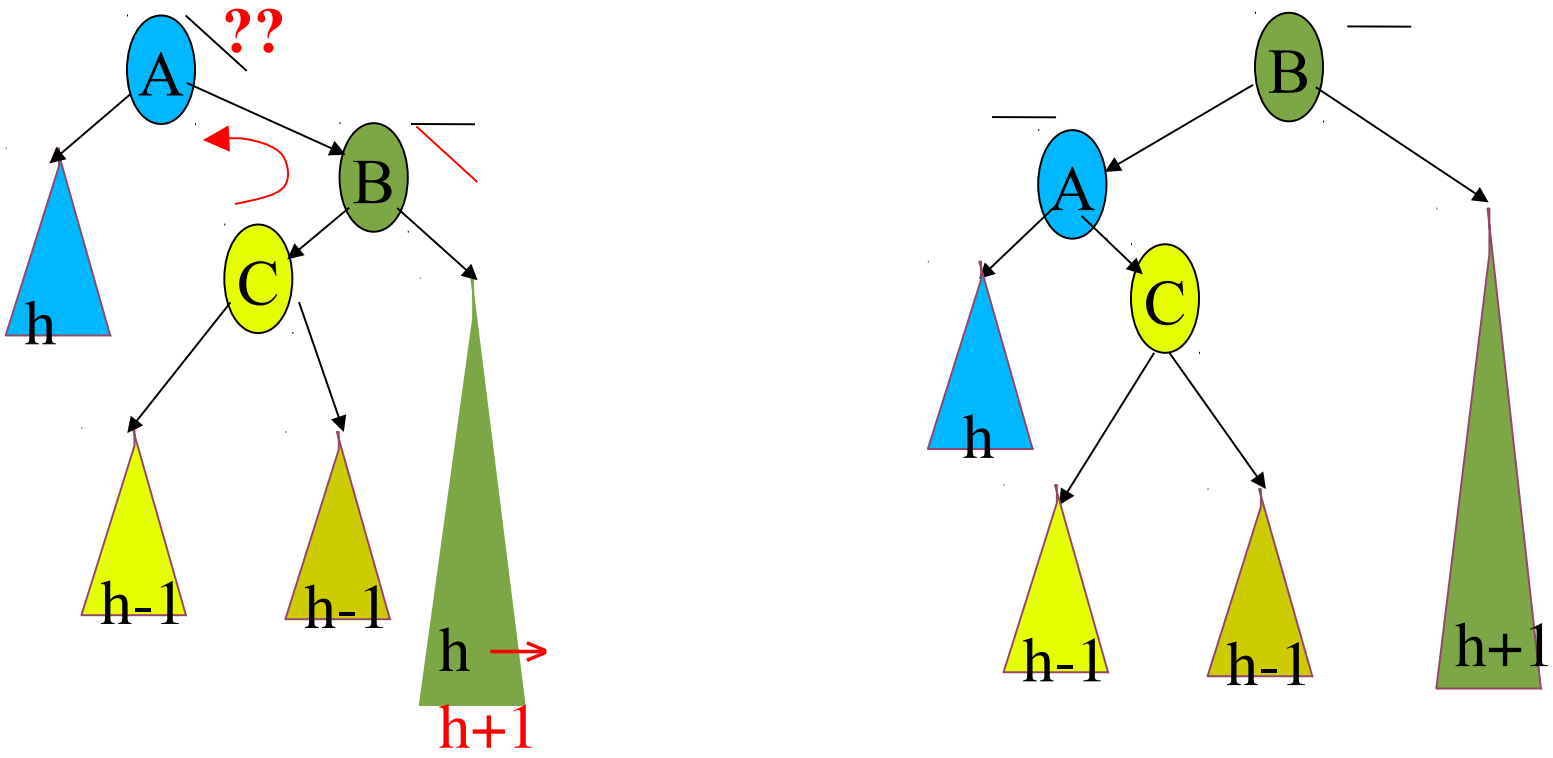


Mirror Image Rotation



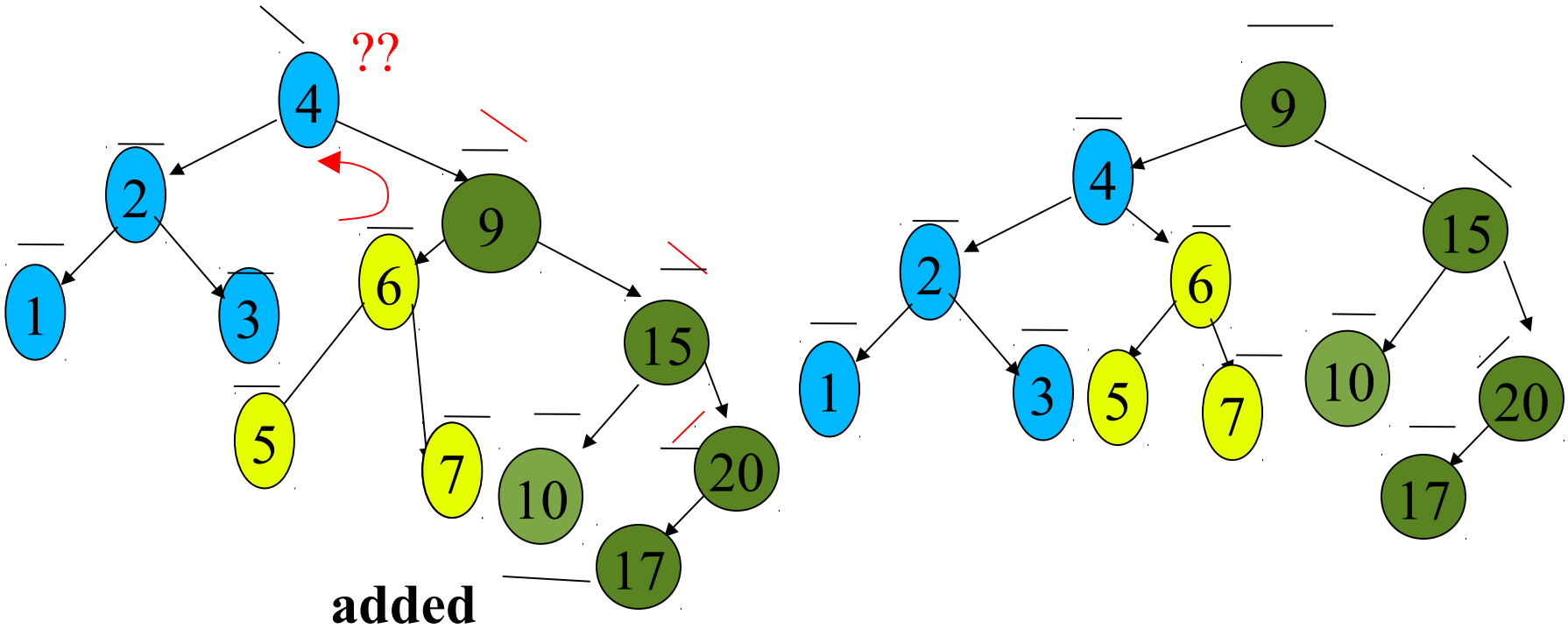
Rebalancing Using Rotation

- **Case 1: Highside child of A has same label as A**



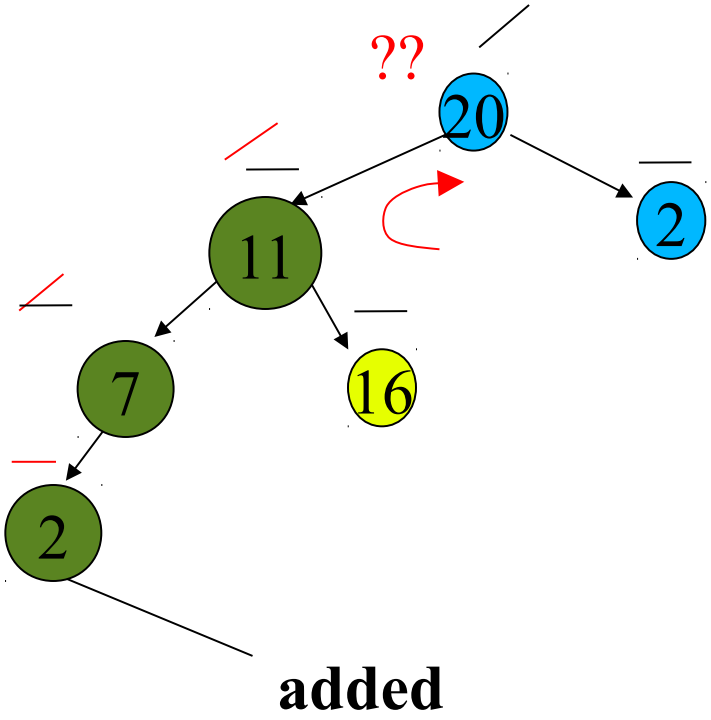
Rebalancing Using Rotation

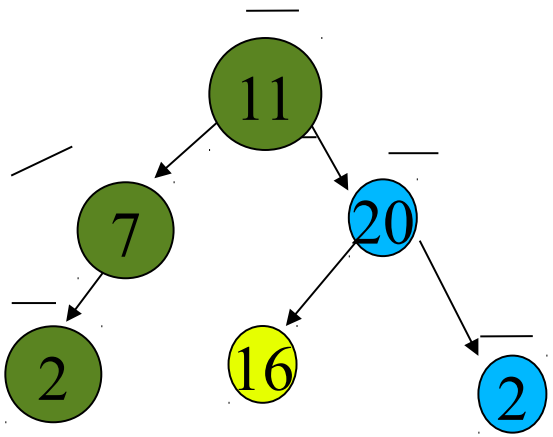
- **Case 1**



Rebalancing Using Rotation

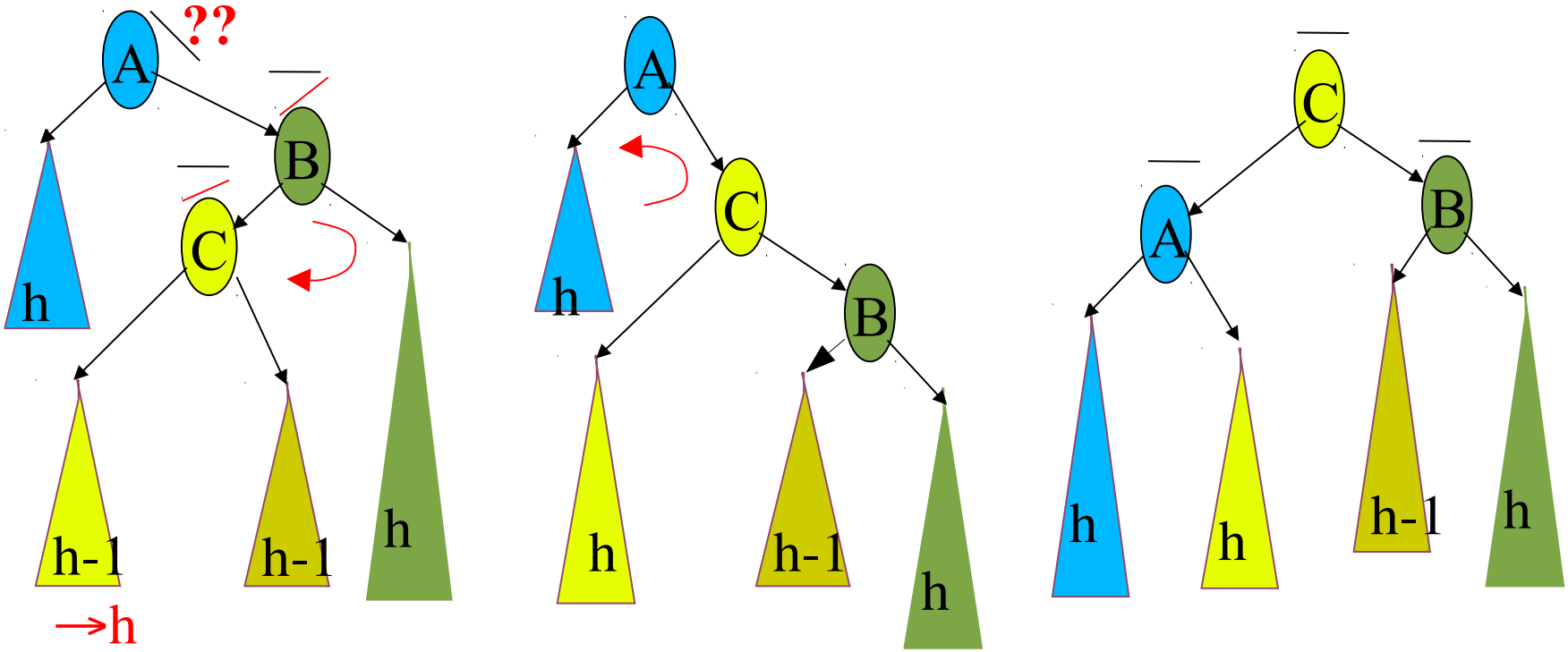
- **Case 1, mirror image**





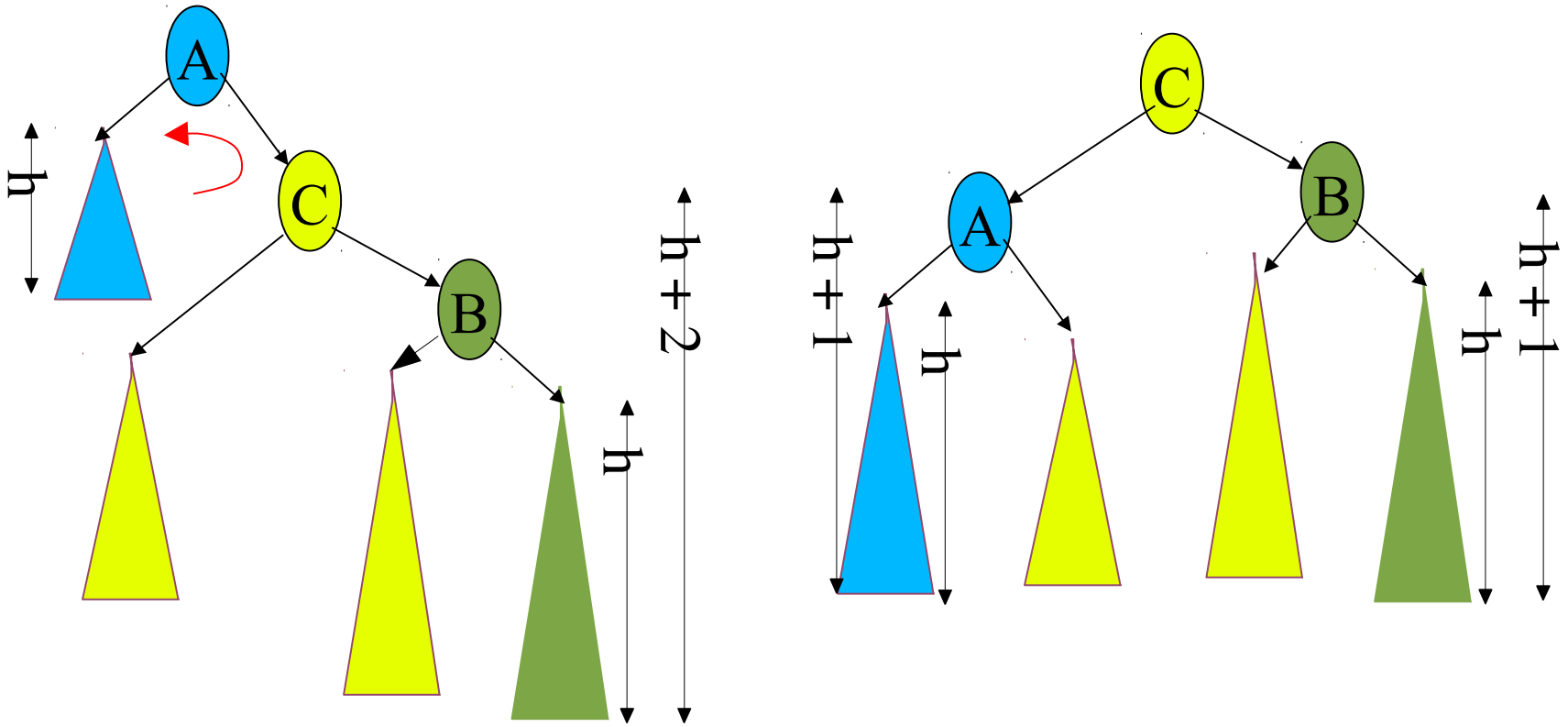
Rebalancing Using Rotation

- **Case 2: Highside child of A has opposite label from A**
  - **Two rotations: BC, then AC**



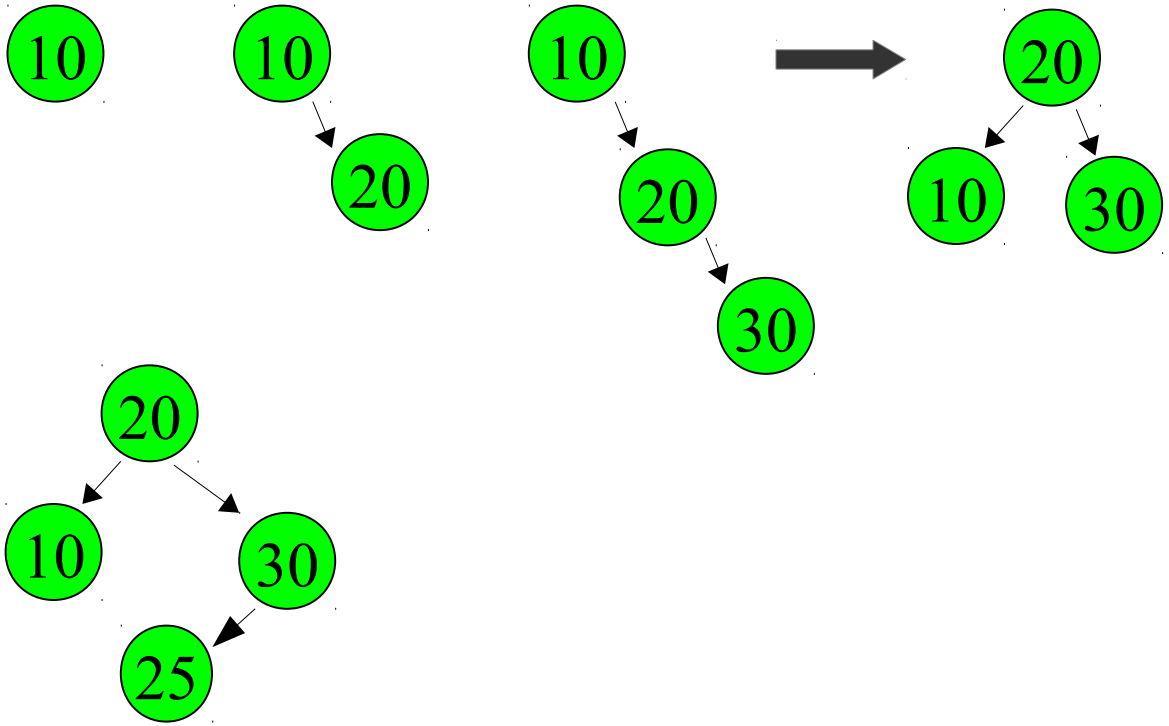
Rebalancing

- **Case 2: Highside child of A has opposite label from A**
  - **Two rotations: BC, then AC**



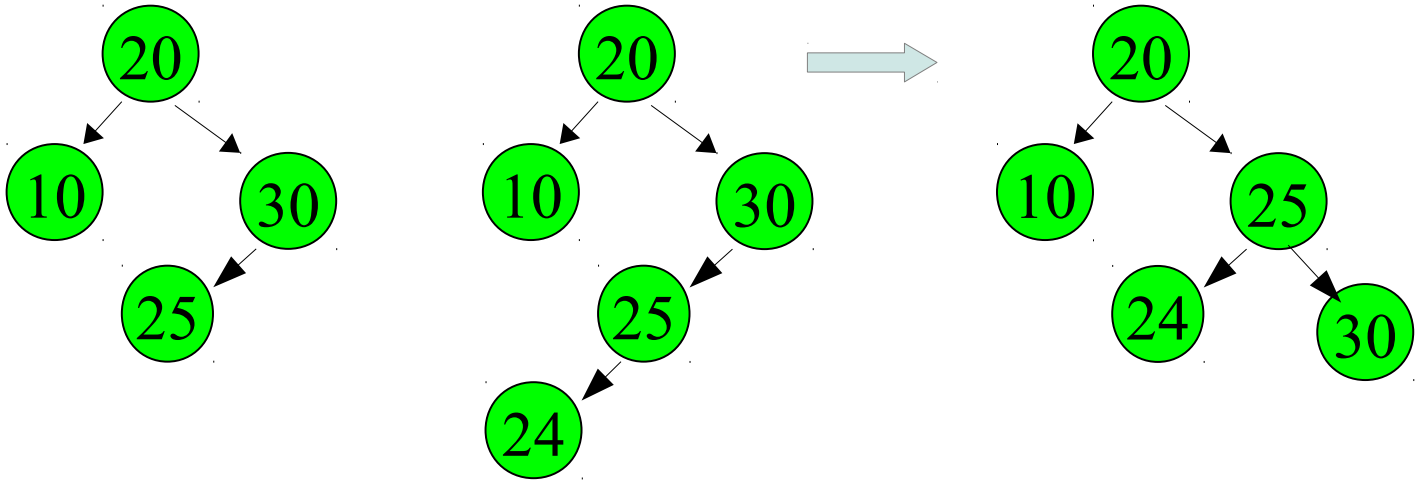
Example

- **insert 10, 20, 30, 25, ...**



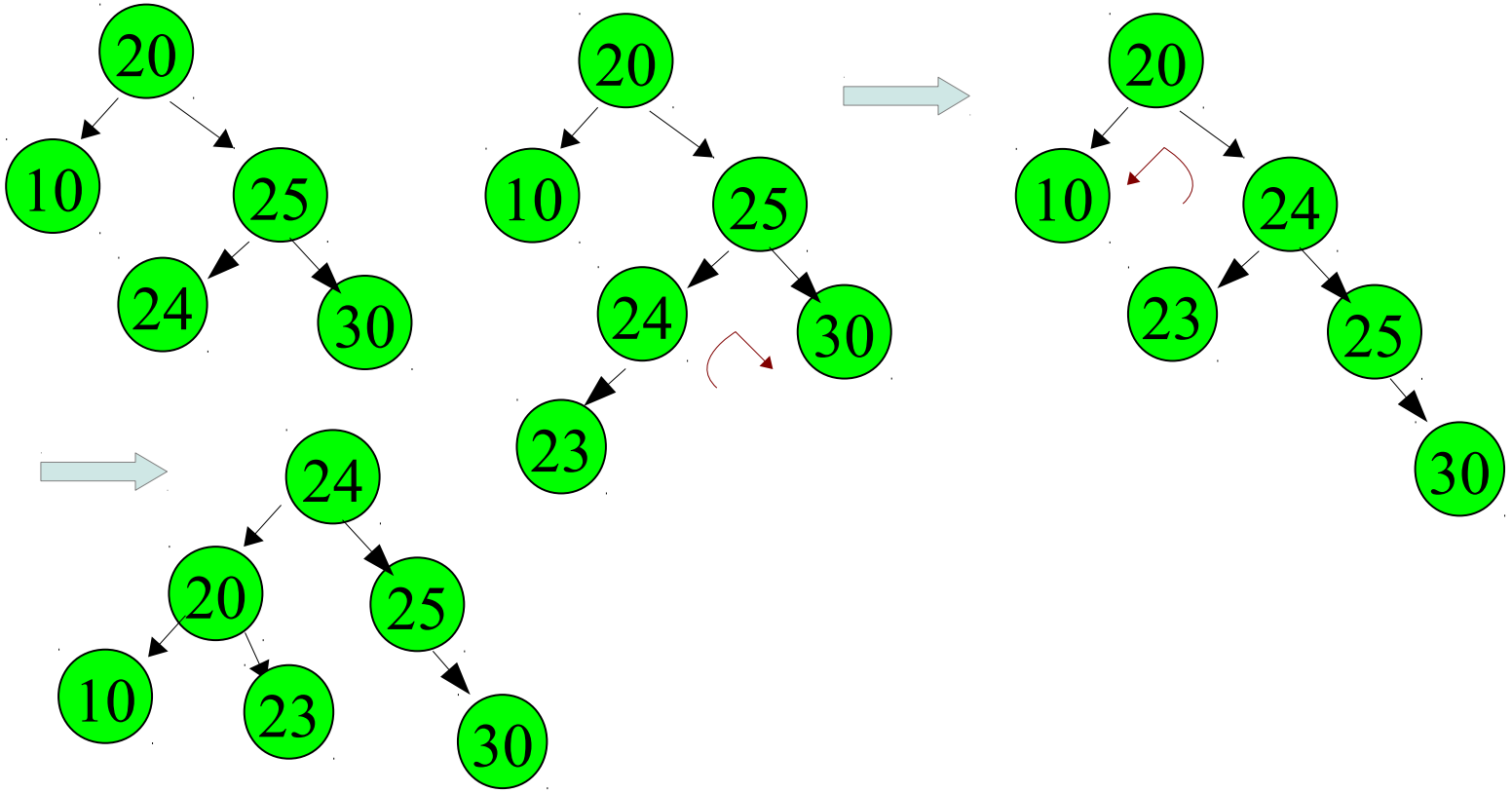
Example

- **insert …, 24, ...**



Example

- **insert …, 23, ...**



Big-O for insert

- **climb tree adjusting balance factors:**
  - **worst case: O(height of tree)**
- **rotate once or twice**
  - **worst case: O(1)**
- **worst case total: O(height)**
- **worst case height: O(log(number of nodes))**
- **worst case insert: O(log(number of nodes))**

Big-O for AVL Tree

- **insert: as in BST then rebalance**

  O(log(n))

- **search: just like Binary Search Tree**

  O(log(n))

- **Delete: delete as in BST, then rebalance**

  O(log(n))