Struktur Data & Algoritma
(*Data Structures & Algorithms*)

Red Black Tree

Denny

Fakultas Ilmu Komputer
Universitas Indonesia

Version 3.0 - Internal Use Only

Motivation

- **AVL Trees take top down insertion and bottom up balancing**
  - ➔ Need recursive implementation
- **Red-Black Tree can be top-down balancing and insertion**
  - ➔ Can be implemented iteratively

- **Red-Black Tree is used as the implementation of java.util.TreeMap and java.util.TreeSet**

Objectives

- **Understand the definition, properties and operations of Red-Black Trees.**

Outline

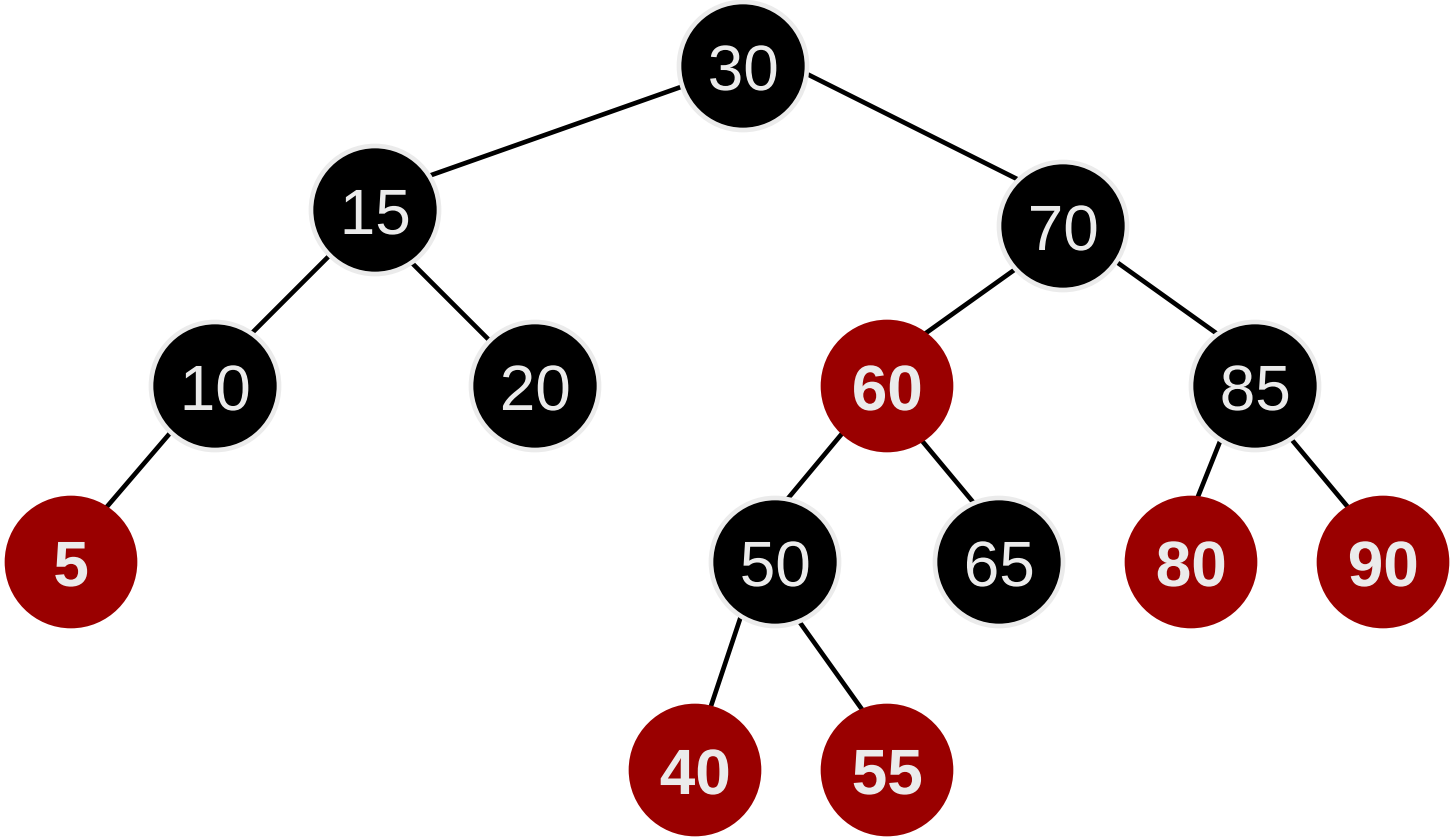
- **Red-Black Trees**
  - Definition
  - Operation

Red-Black Trees: Definition

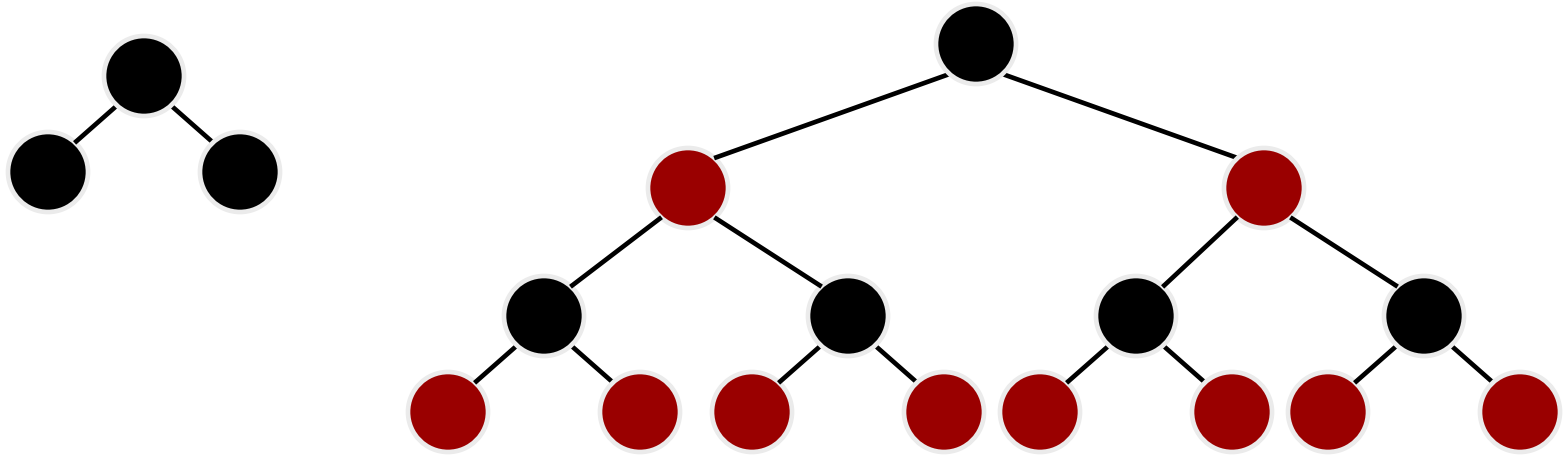
1. **Every node is colored either red or black**
2. **The root is black**
3. **If a node is red, its children must be black**
   - consecutive red nodes are disallowed
4. **Every path from a node to a null reference must contain the same number of black nodes**

Red-Black Trees

- **The insertion sequence is 10, 85, 15, 70, 20, 60, 30, 50, 65, 80, 90, 40, 5, 55**



Red-Black Trees: properties



$$2^B - 1 \le N \le 2^{2B} - 1$$

$$2^B \le N + 1 \le 2^{2B}$$

$$\log 2^B = B \qquad \log 2^{2B} = 2B$$

$$B \le \log(N+1) \qquad \log(N+1) \le 2B$$

$$\frac{1}{2}\log(N+1) \le B \le \log(N+1)$$

$$\log(N+1) \le H \le 2\log(N+1)$$

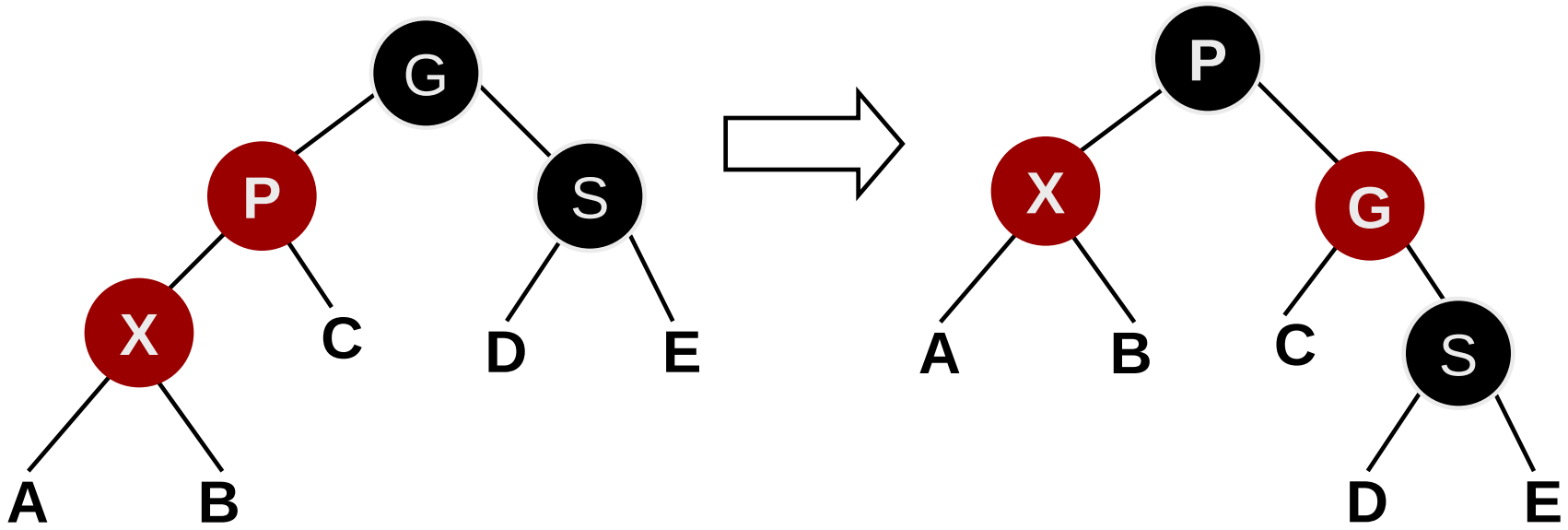
- **B = total black nodes from root to leaf**
- **N = total all nodes**
- **H = height**

All operation guaranteed logarithmic.

Insertion

- **A new node must be colored red**
  - why?
    - new item is always inserted as a leaf in the tree
    - if we color a new item black, then the number of black nodes from root would be different (violate property #4)
  - if the parent is black, no problem!
  - if the parent is red, we create two consecutive red nodes
- **Convention: null nodes are black**

Single Rotation



- **Case after insertion:**
  - Consecutive red
  - Sibling of parent is black
  - Outer node (left-left or right-right)

X: new node

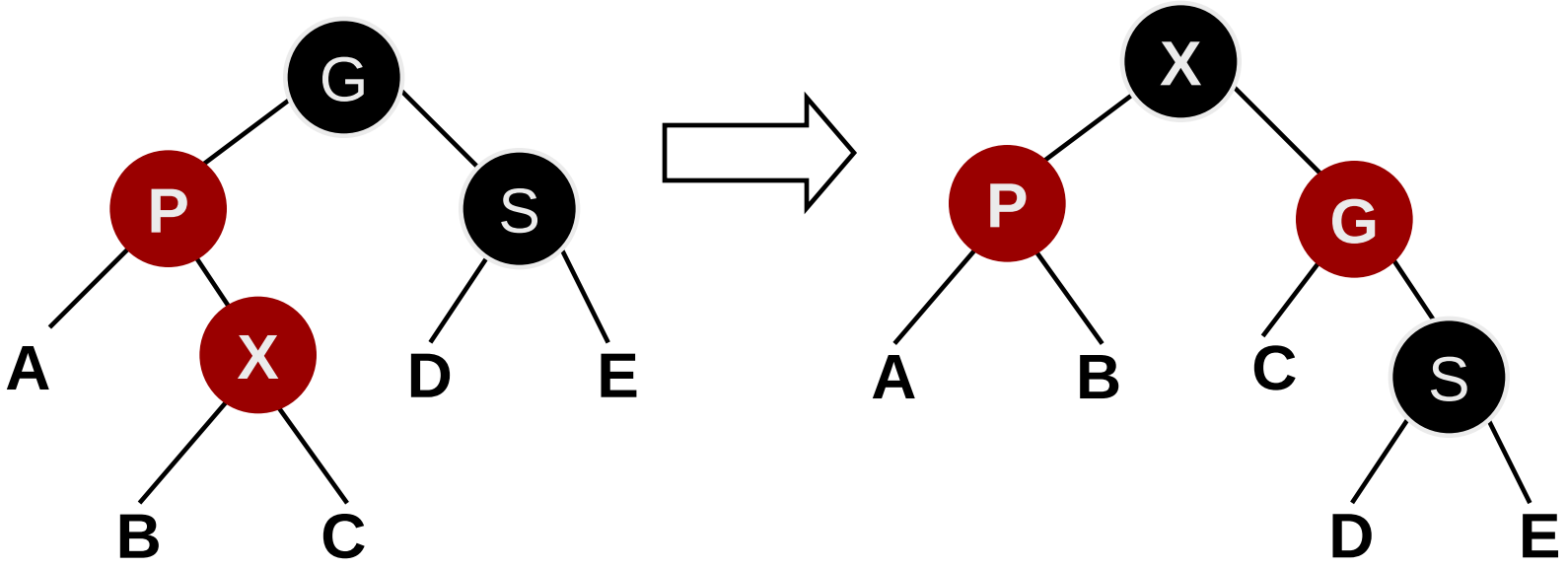
P: parent

S: sibling

G: Grandparent

Maintain number of black nodes in every path

Double Rotation



- **Case after insertion:**
  - Consecutive red
  - Sibling of parent is black
  - Inner node (left-right or right-left)

X: new node

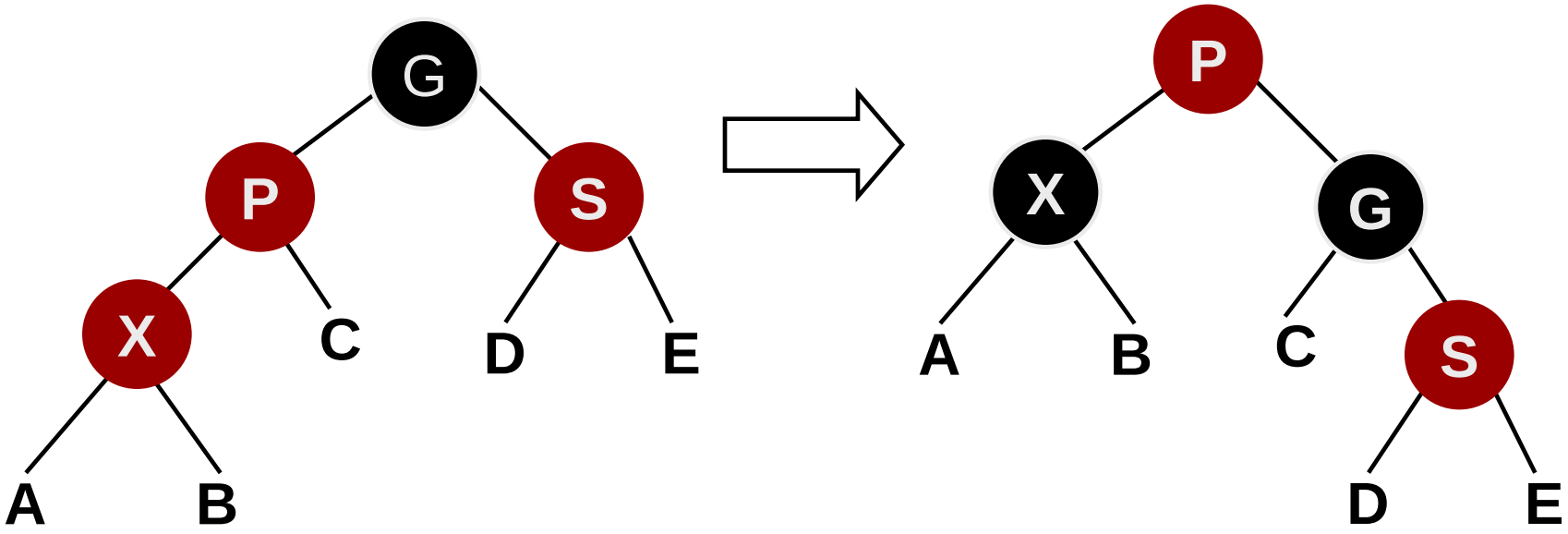
P: parent

S: sibling

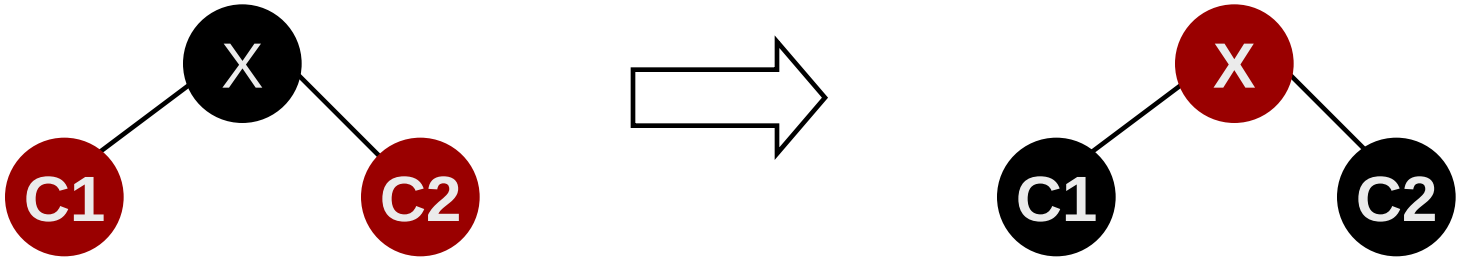
G: Grandparent

Single Rotation

- **Case after insertion:**
  - Consecutive red
  - Sibling of parent is red
  - Outer node (left-left or right-right)

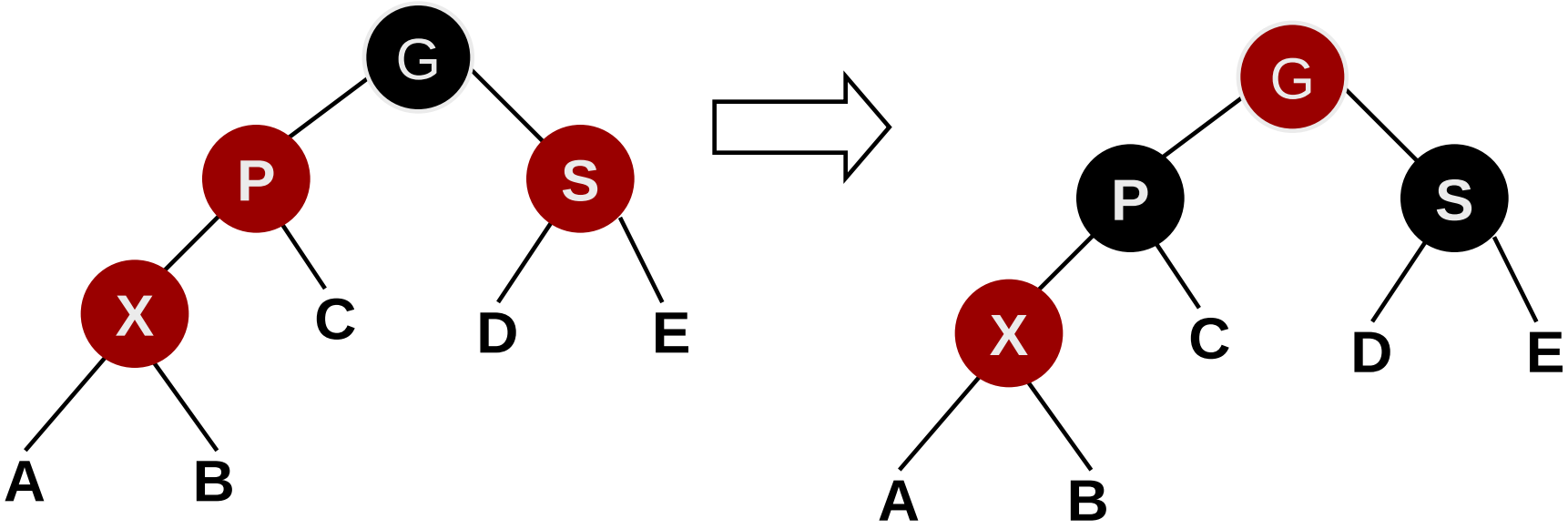


Color Flip (1)



- **During the insertion process:**
  - if we fall into black node with two red children, flip the color
  - do rotation if consecutive red occurs
  - set the root to black (If the flip process involves root, root becomes red.)

Color Flip (2)

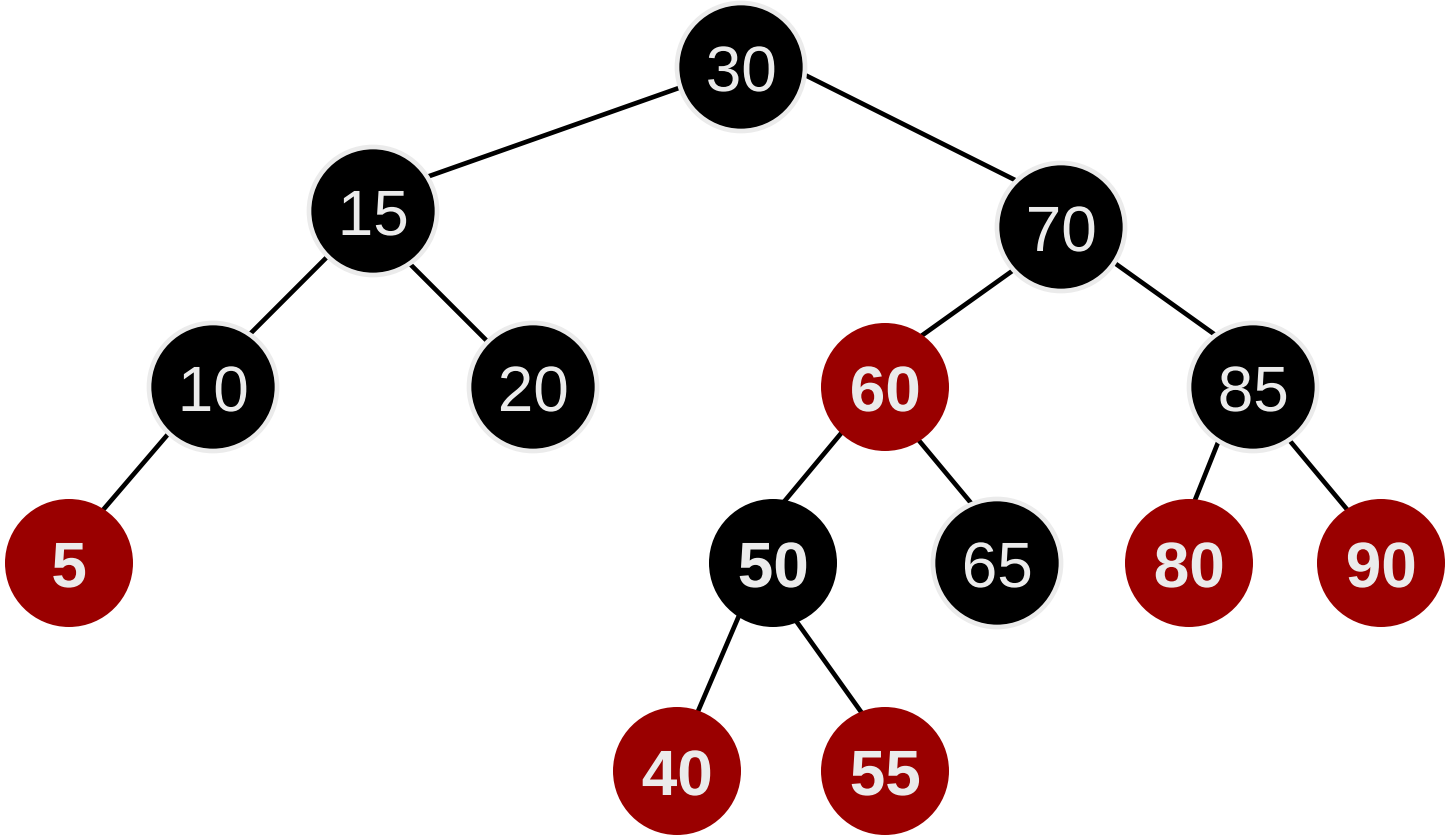


Insertion

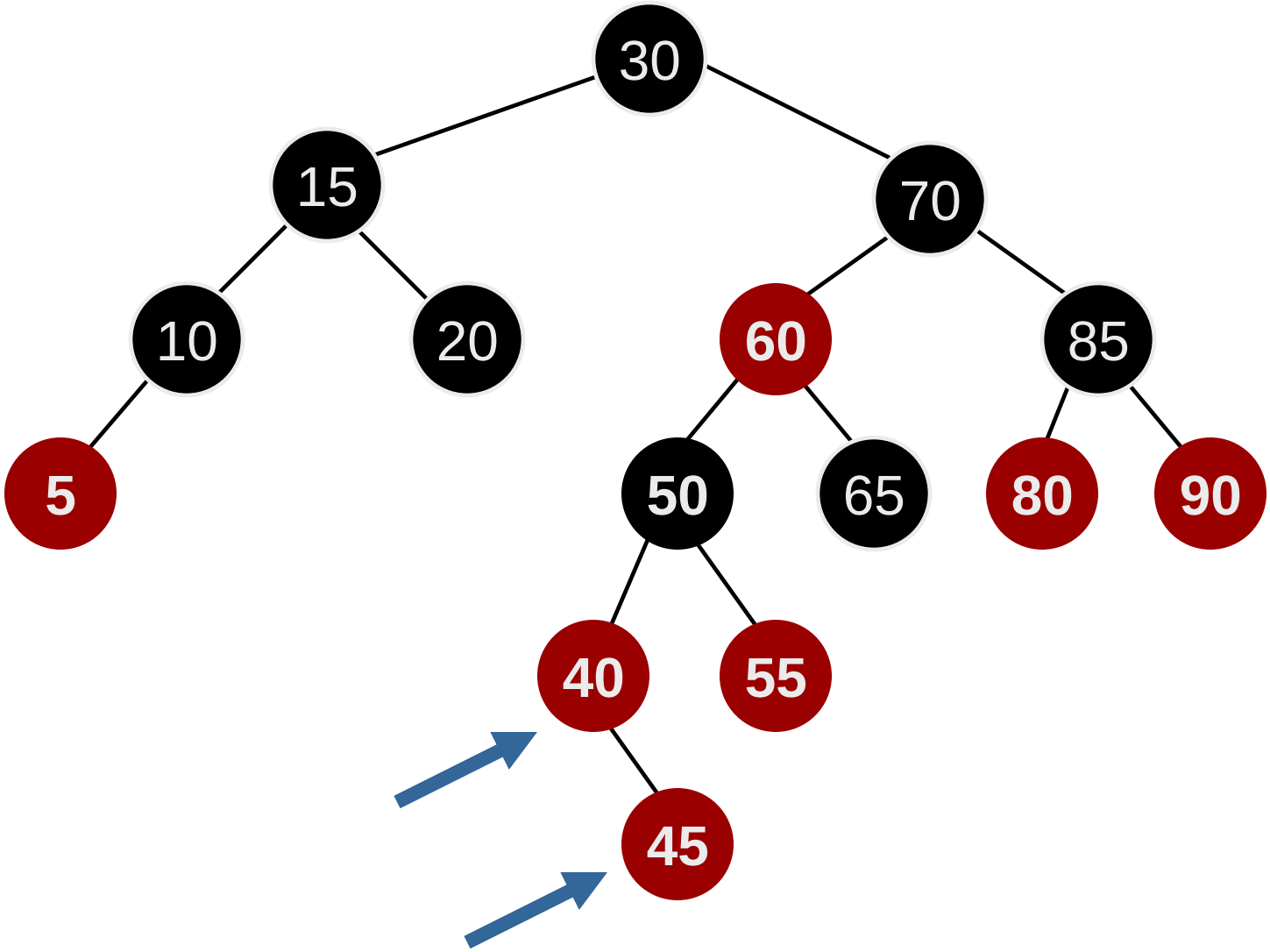
- **Top down insertion: as BST**
- **Bottom-up:**
  - check for two consecutive red nodes
    - if sibling is red ➔ color flip
    - if sibling is black ➔ rotation

- **Implementation: quite straight forward, similar to AVL tree**

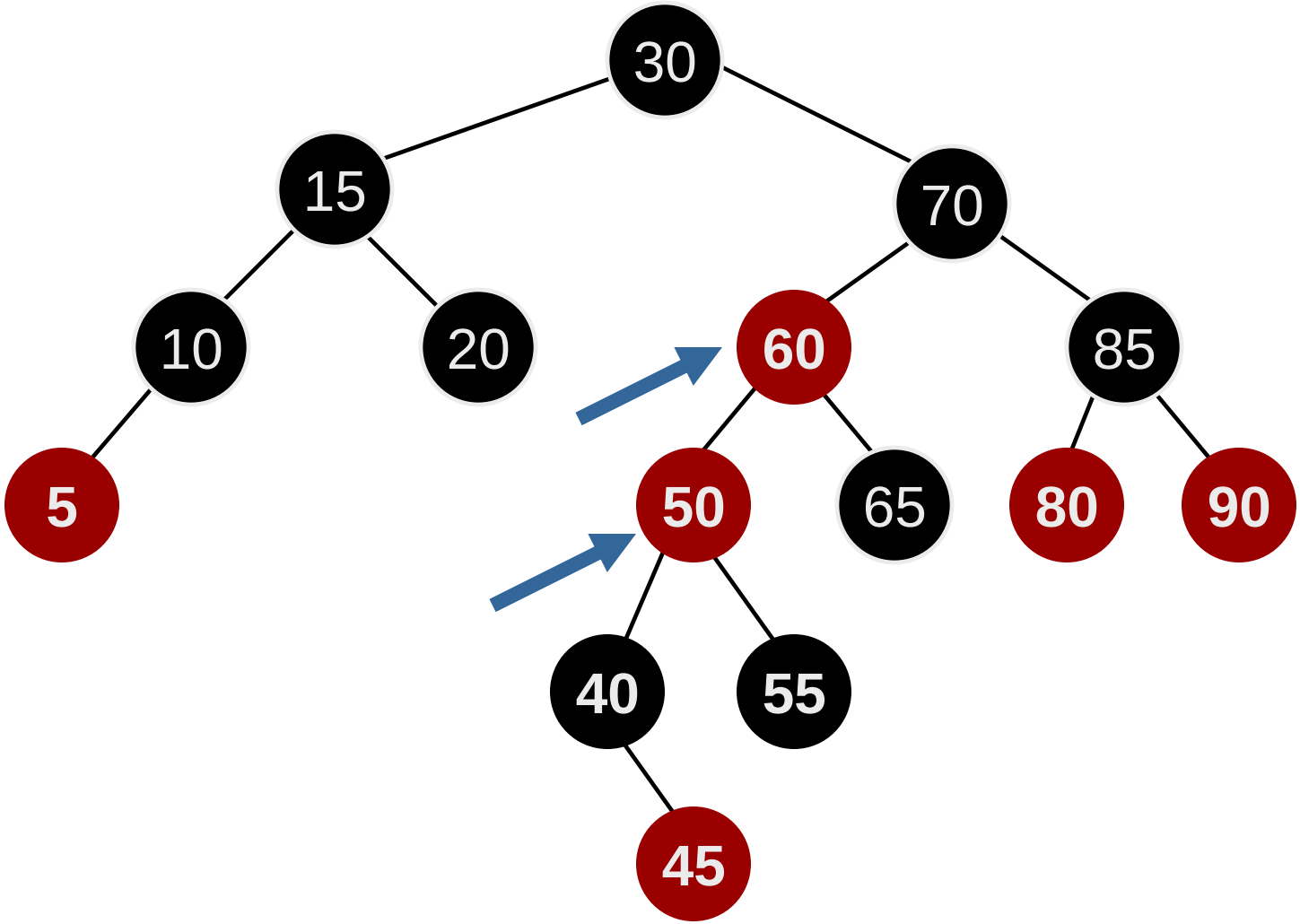
Insert 45 (original)



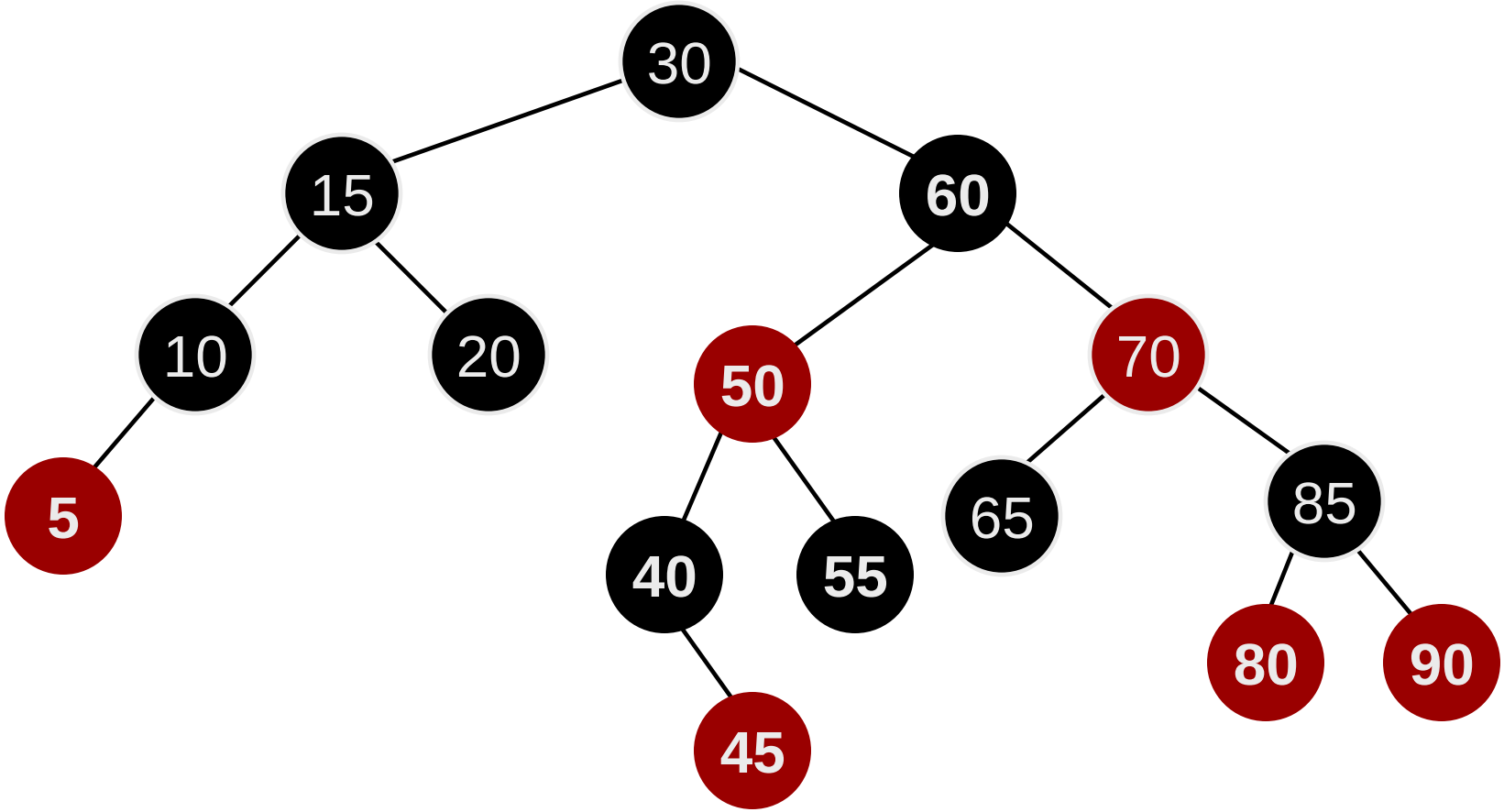
Insert 45



Insert 45 (after color flip)



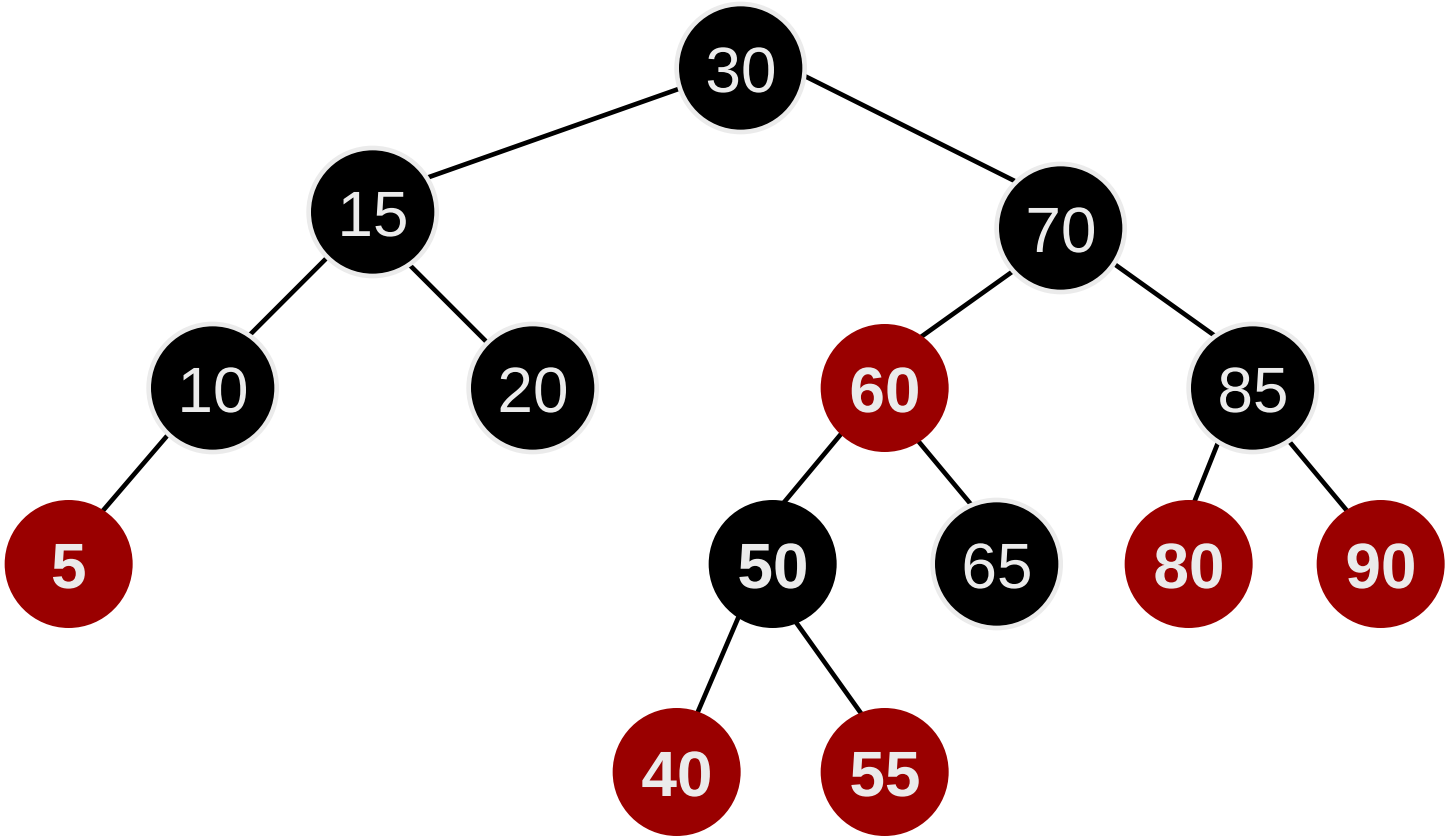
Insert 45 (single rotation)



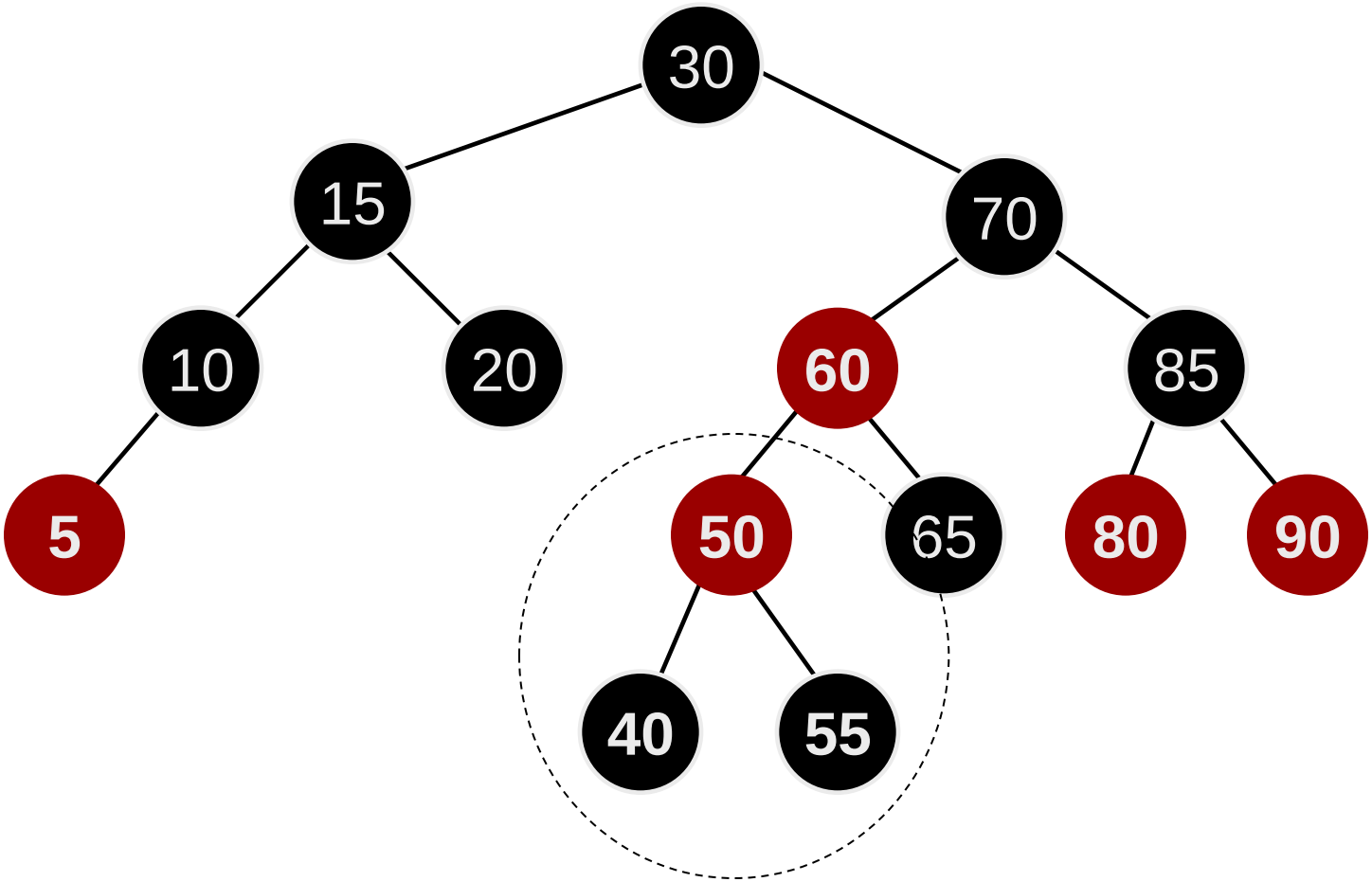
Top Down Insertion

- **Top down insertion, one pass:**
  - check if a node has both children red ➔ color flip
    - color flip might cause two consecutive red nodes
    - if after a color flip cause two consecutive red nodes ➔ perform a single or double rotation
  - insert the new red node at the leaf
    - check for two consecutive red nodes
- **No need to fix on the way back to root**
- **Implementation:**
  - need to store current, parent, grand parent, great grand parent.

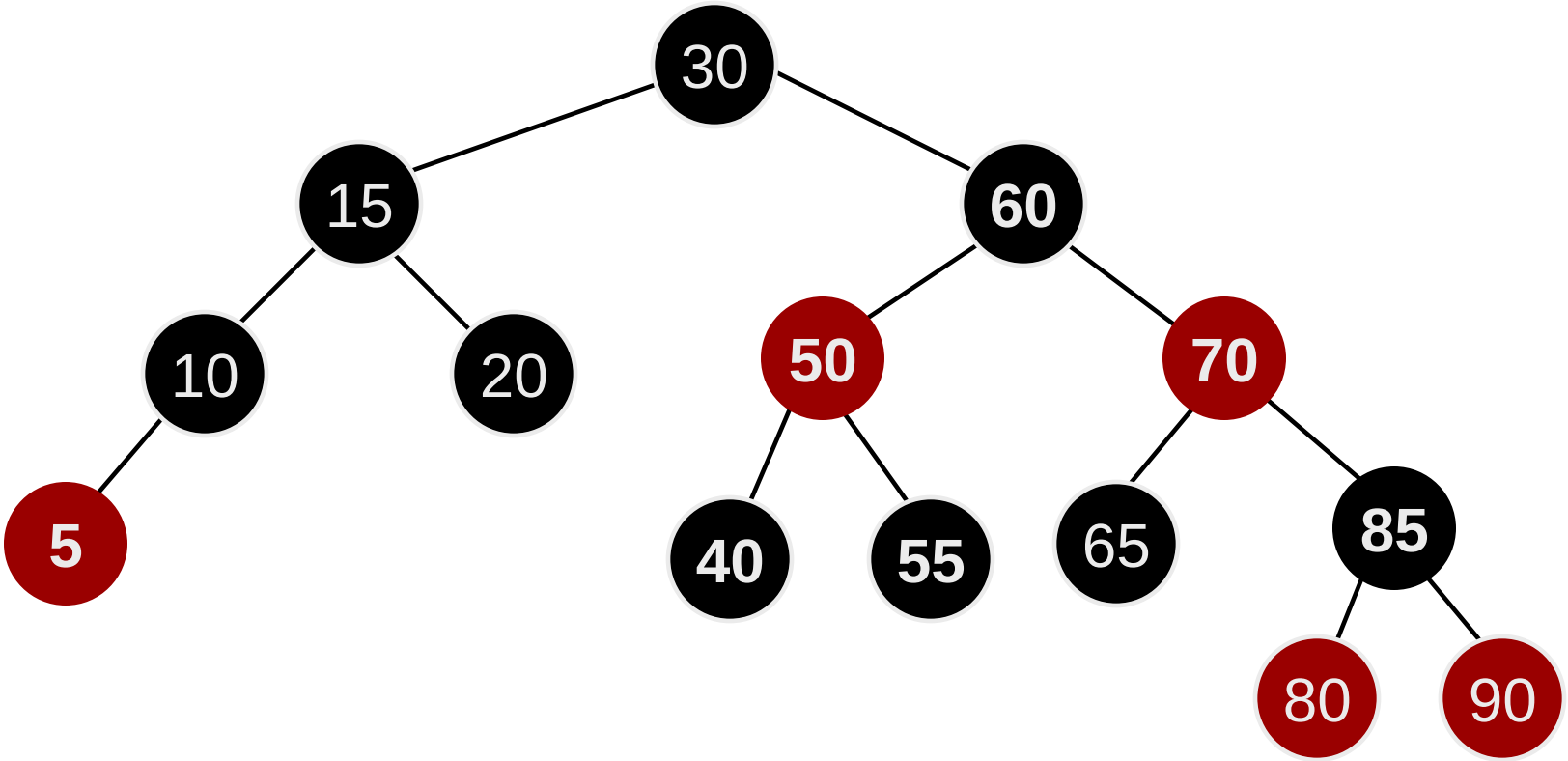
Insert 45 (original)



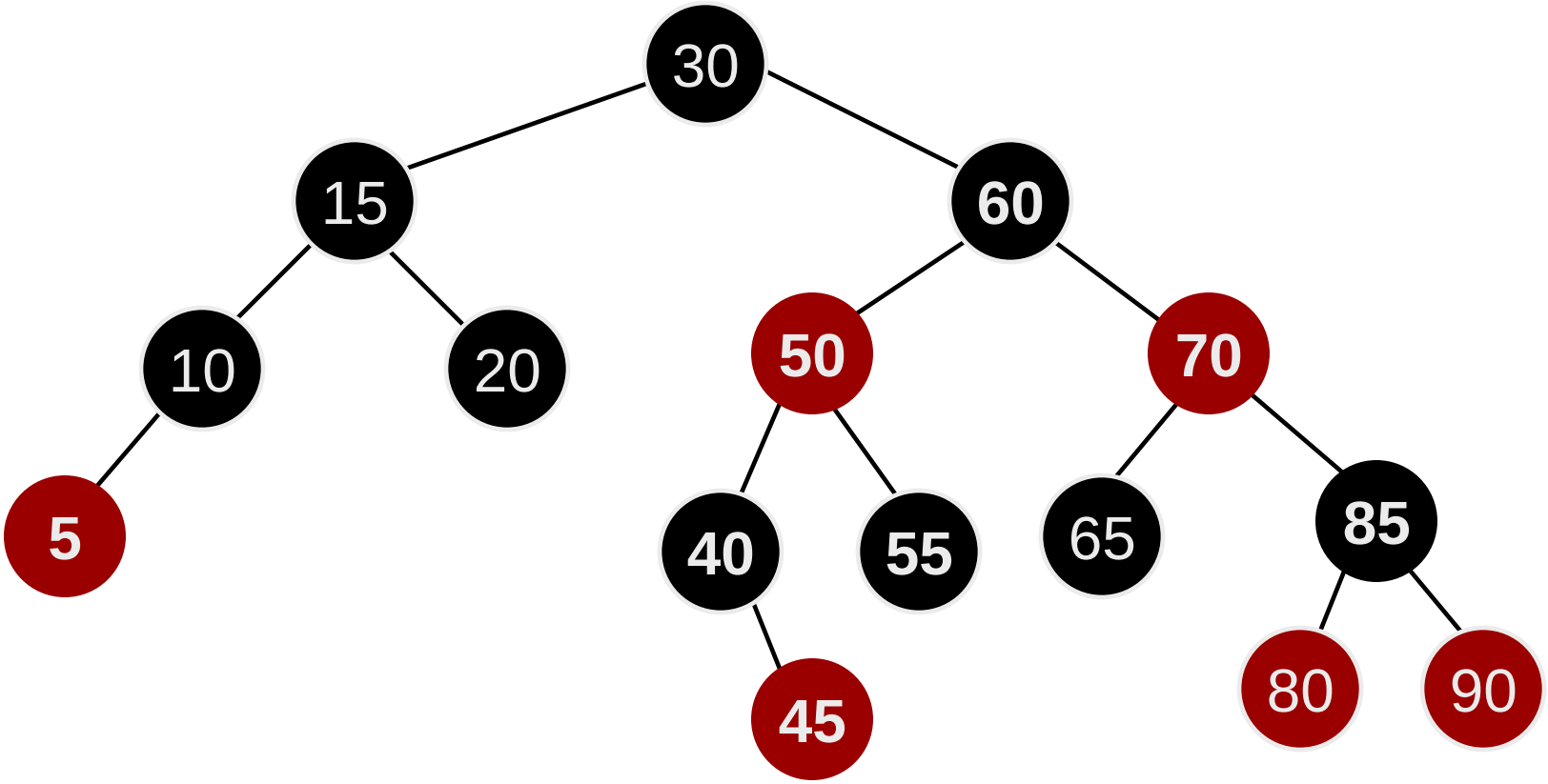
Insert 45 (after color flip)



Insert 45 (single rotation)



Insert 45



Summary

- **Red-Black trees use color as balancing information instead of height in AVL trees.**
- **An insertion may cause a local perturbation (two consecutive red nodes)**
- **The pertubation is either**
  - resolved locally (rotations), or
  - propagated to a higher level in the tree by recoloring (color flip)
- **O(1) for a rotation or color flip**
- **At most one restructuring per insertion.**
- **O(log n) color flips**
- **Total time: O(log n)**

Further Reading

- **Red-Black Tree Applet**
  http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html