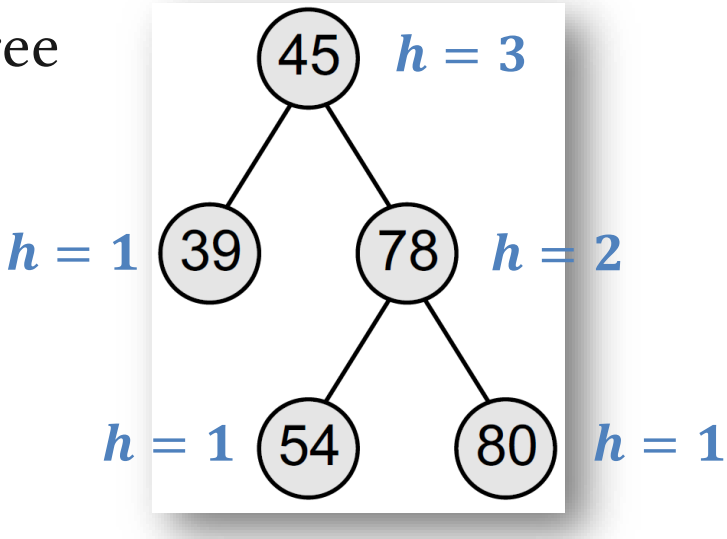
Red-Black Trees

Kuan-Yu Chen (陳冠宇)

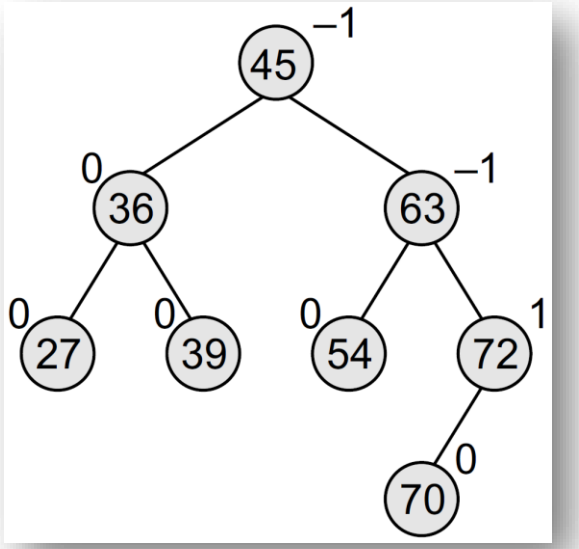
2020/10/26 @ TR-313, NTUST

Review

- Height for a node in a binary search tree
  - The height of the leaf node is 1
  - The height of the internal node is $1 + \max(h_L, h_R)$



- AVL Trees
  - Self-balancing binary search tree
  - Balance Factor
    - Every node has a balance factor of −1, 0, or 1
  - Rotation is used to restore the balance of the tree



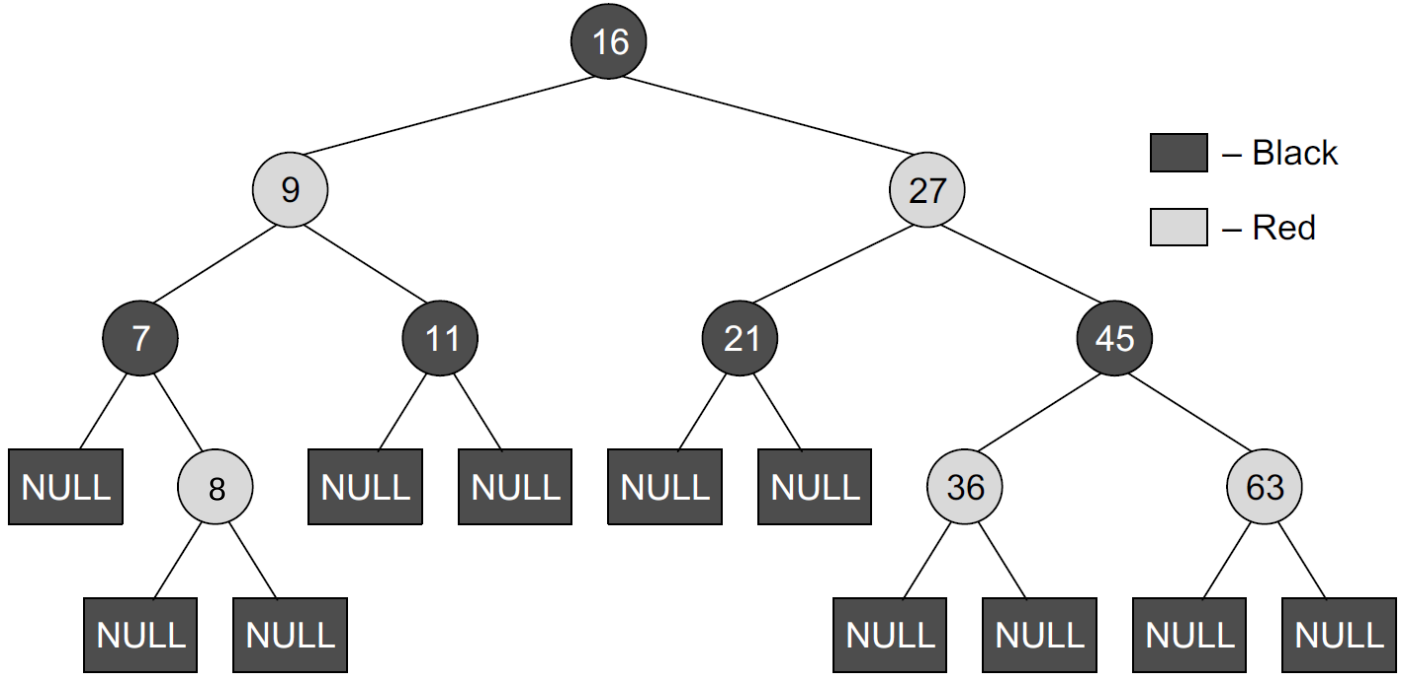
Red-Black Trees.

- A red-black tree is a self-balancing binary search tree that was invented in 1972 by Rudolf Bayer
  - A special point to note about the red-black tree is that in this tree, **no data is stored in the leaf nodes**

- A red-black tree is a binary search tree
  1. The color of a node is either **red** or **black**
  2. The color of the root node is always black
  3. All leaf nodes are black
  4. Every red node has both the children colored in black
  5. Every simple path from a given node to any of its leaf nodes has an equal number of black nodes

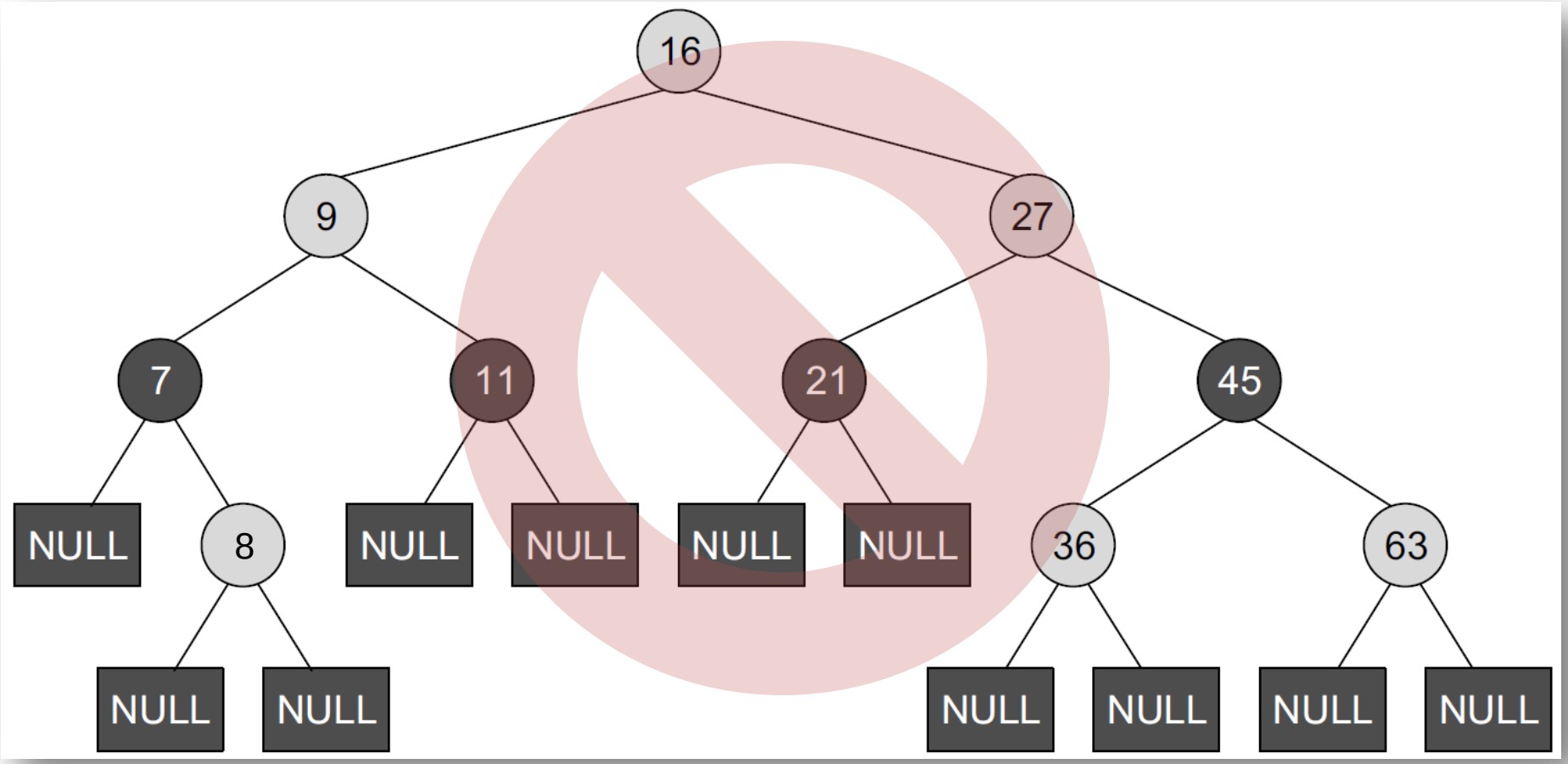
Red-Black Trees..

1. The color of a node is either red or black
2. The color of the root node is always black
3. All leaf nodes are black
4. Every red node has both the children colored in black
5. Every simple path from a given node to any of its leaf nodes has an equal number of black nodes



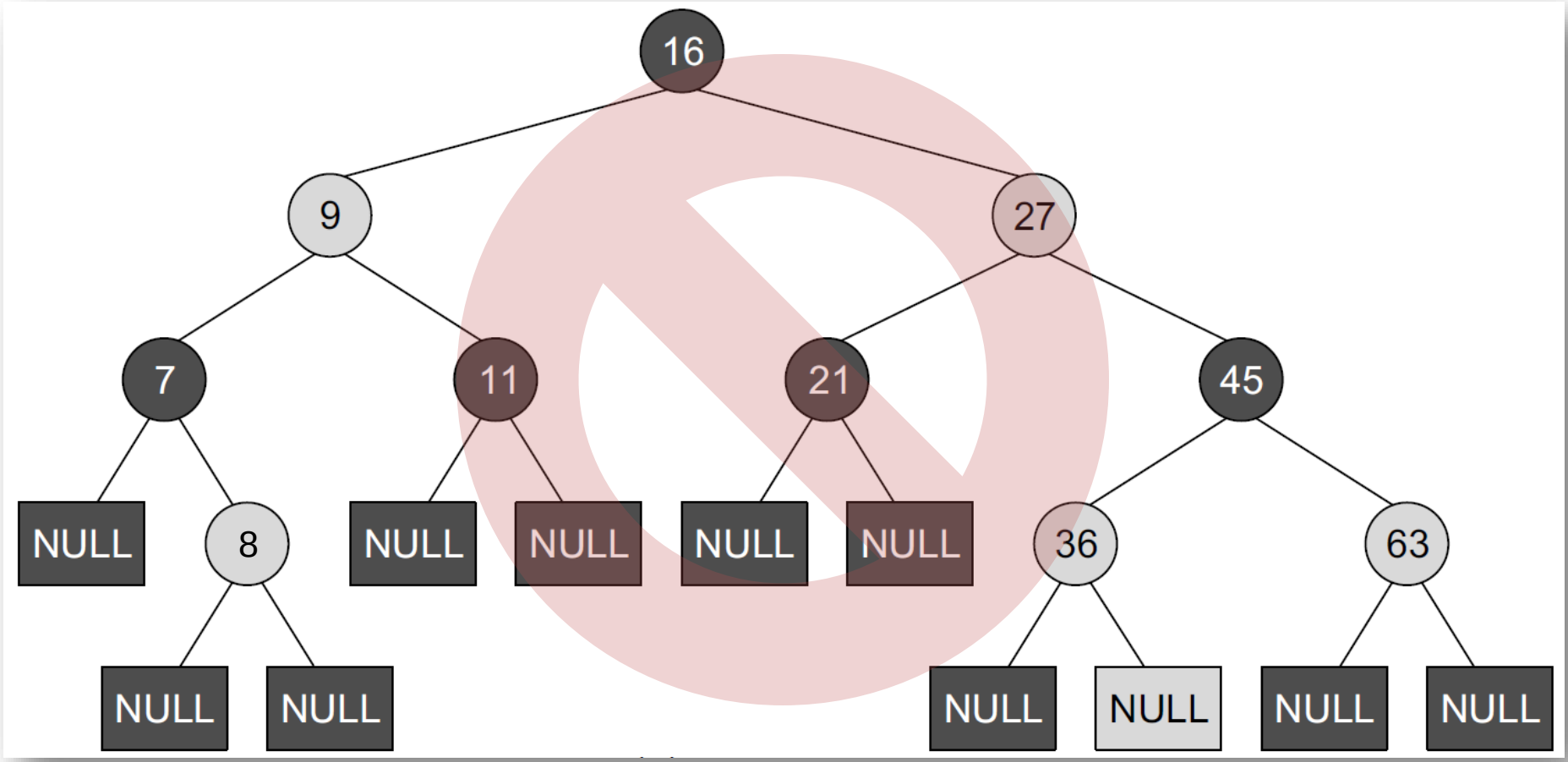
Red-Black Trees...

- Root is red



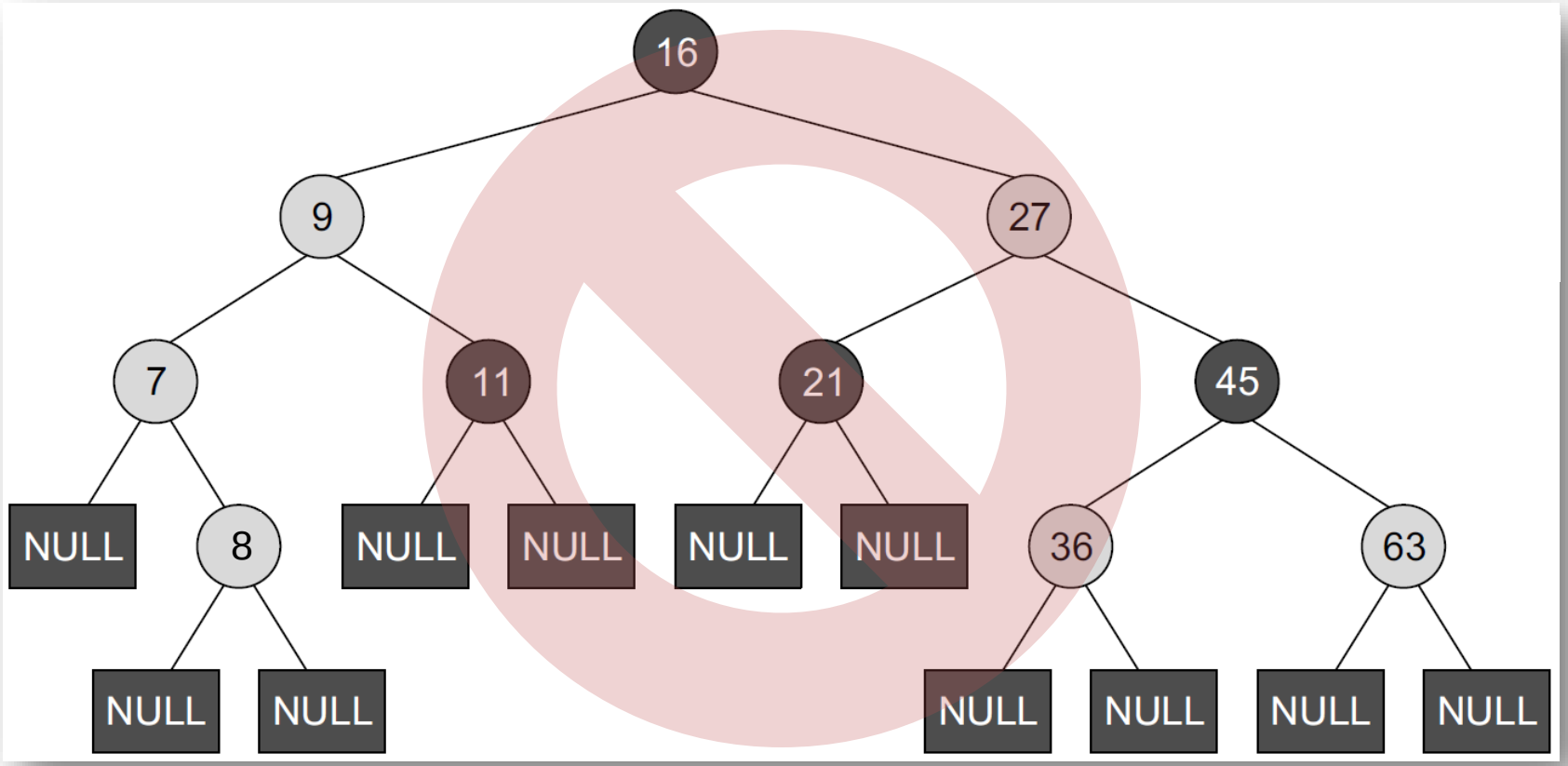
Red-Black Trees….

- A leaf node is red



Red-Black Trees…..

- Every red node does not have both the children colored in black
- Every simple path from a given node to any of its leaf nodes does not have equal number of black nodes



Searching in a Red-Black Tree

- Since red-black tree is a binary search tree, it can be searched using exactly the **same algorithm** as used to search an ordinary binary search tree!

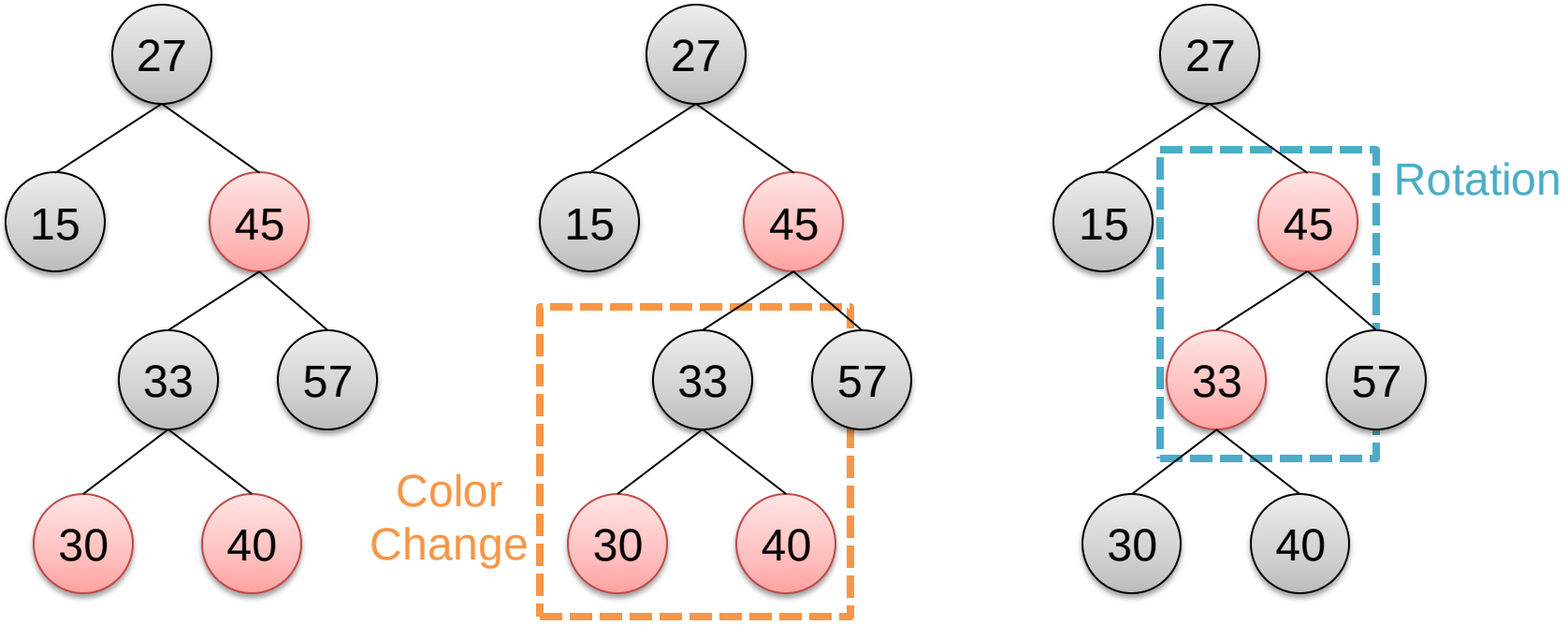
Insertion in a Red-Black Tree

- In a binary search tree, we always add the new node as a leaf, while in a red-black tree, leaf nodes contain no data
  - For a given data
    1. Searching the correct position for the data
    2. In the searching process, if there is a node with two red children
       a) Perform color change algorithm
       b) Check whether there are two consequent red nodes in the path
          ① If yes, do rotation!
    3. Insert the data and set to a red node
    4. Check whether there are two consequent red nodes in the path
       a) If yes, do rotation!
    5. Root should be black

Color Change → Rotation → Insert → Rotation → Check Root

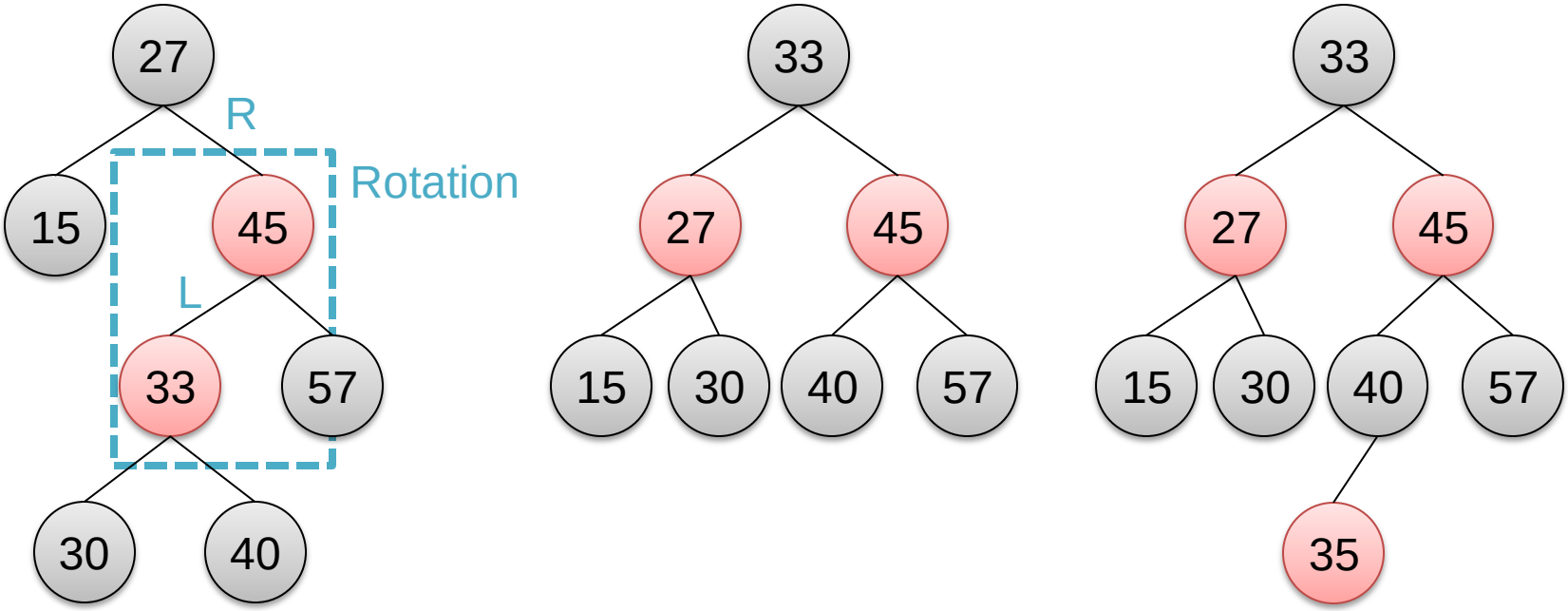
Examples – 1.

- For a given red-black tree, please insert element 35



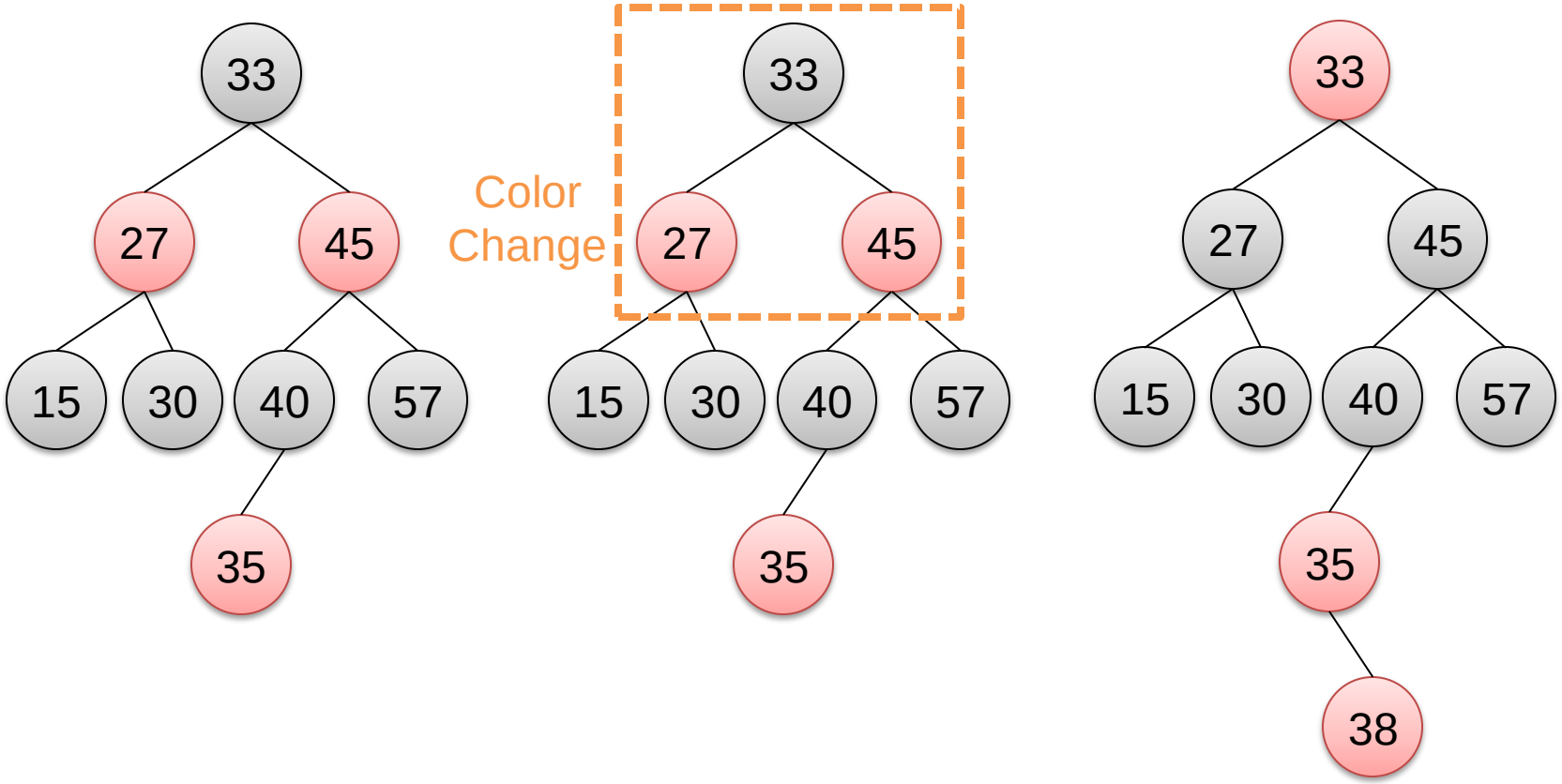
Examples – 1..

- For a given red-black tree, please insert element 35



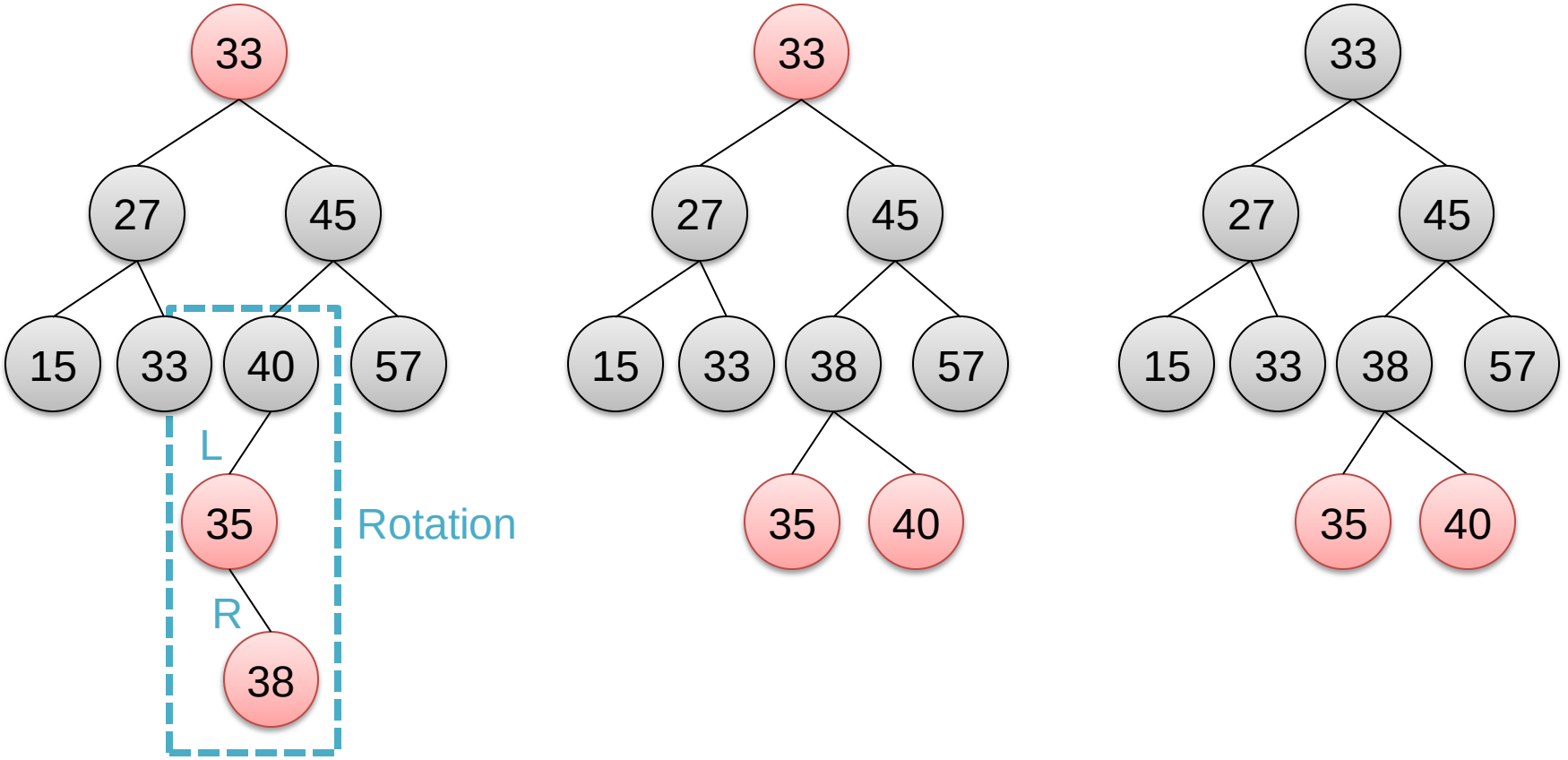
Examples – 2.

- For a given red-black tree, please insert element 38



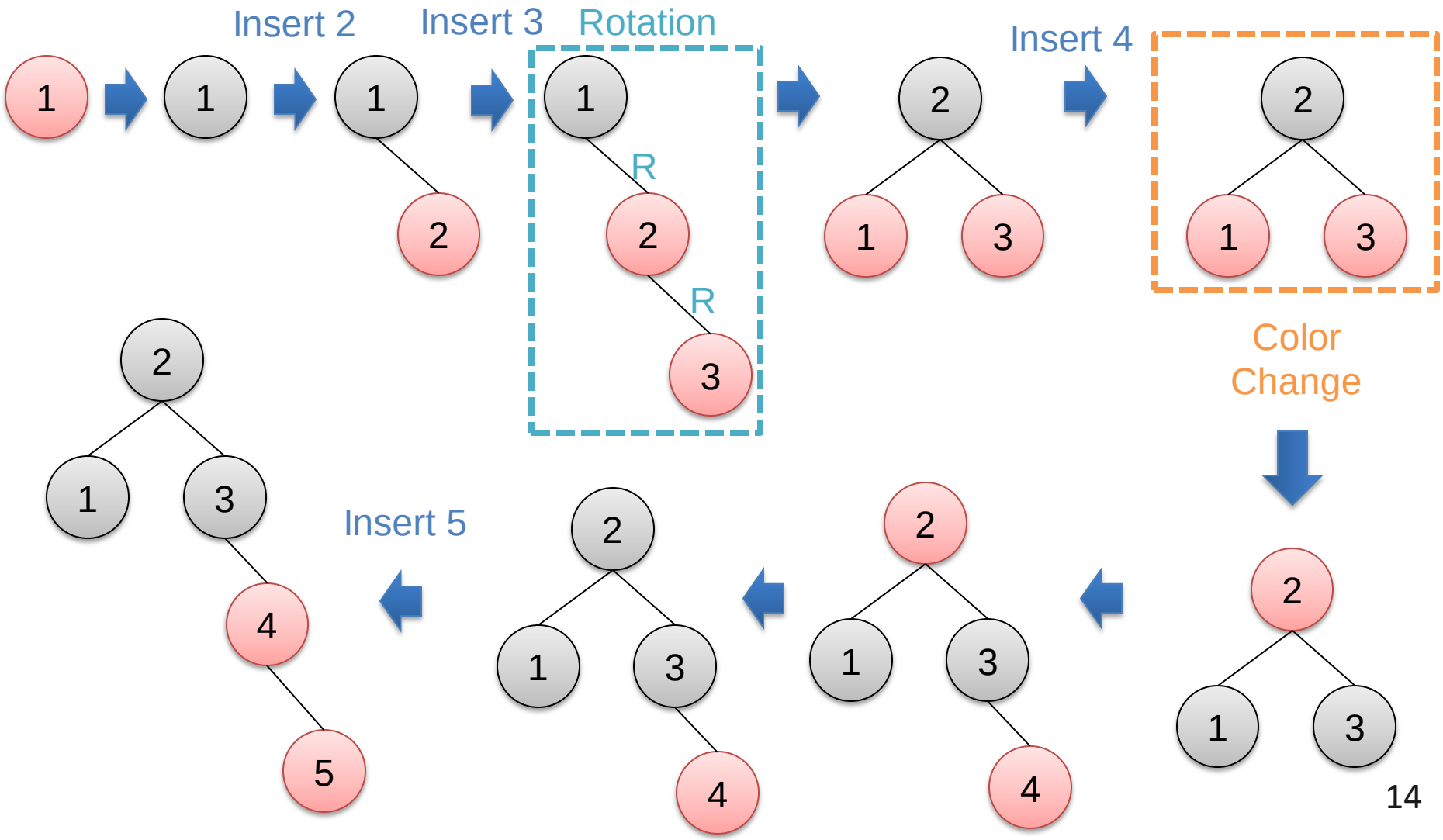
Examples – 2..

- For a given red-black tree, please insert element 38



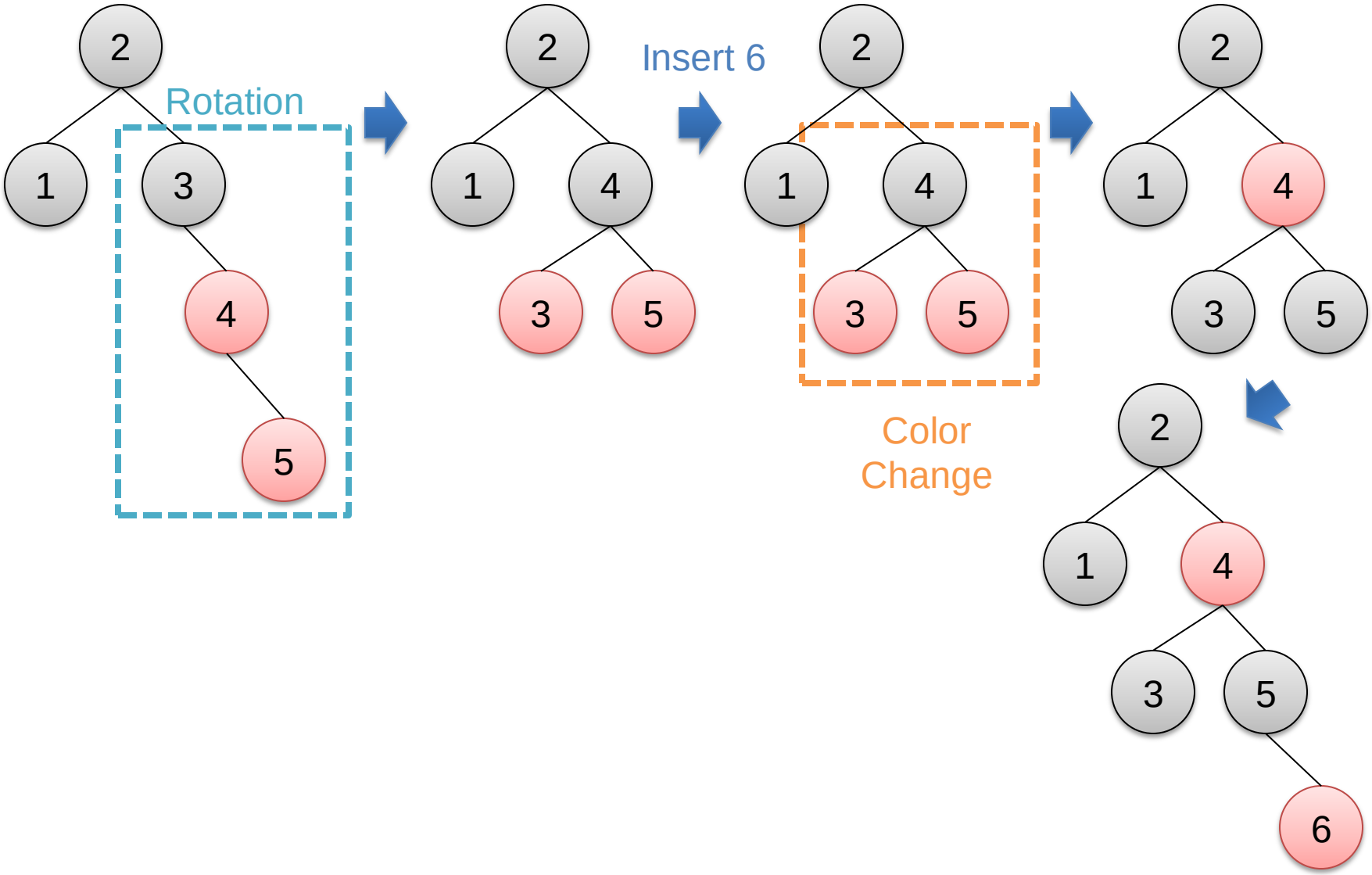
Examples – 3.

- Given 1, 2, 3, 4, 5 and 6, please construct a red-black tree



Examples – 3..

- Given 1, 2, 3, 4, 5 and 6, please construct a red-black tree



Compared with AVL Trees

- Red-black trees are efficient binary search trees, as they offer worst case time guarantee O(log n) for insertion, deletion, and search operations
  - It is roughly a balanced binary search tree

- AVL trees also support O(log n) search, insertion, and deletion operations, but they are more rigidly balanced than red-black trees
  - Thereby, AVL trees are slower insertion and removal but faster retrieval of data

Questions?

kychen@mail.ntust.edu.tw