BM267 - Introduction to Data Structures

13. Balanced Search Trees

Ankara University

Computer Engineering Department

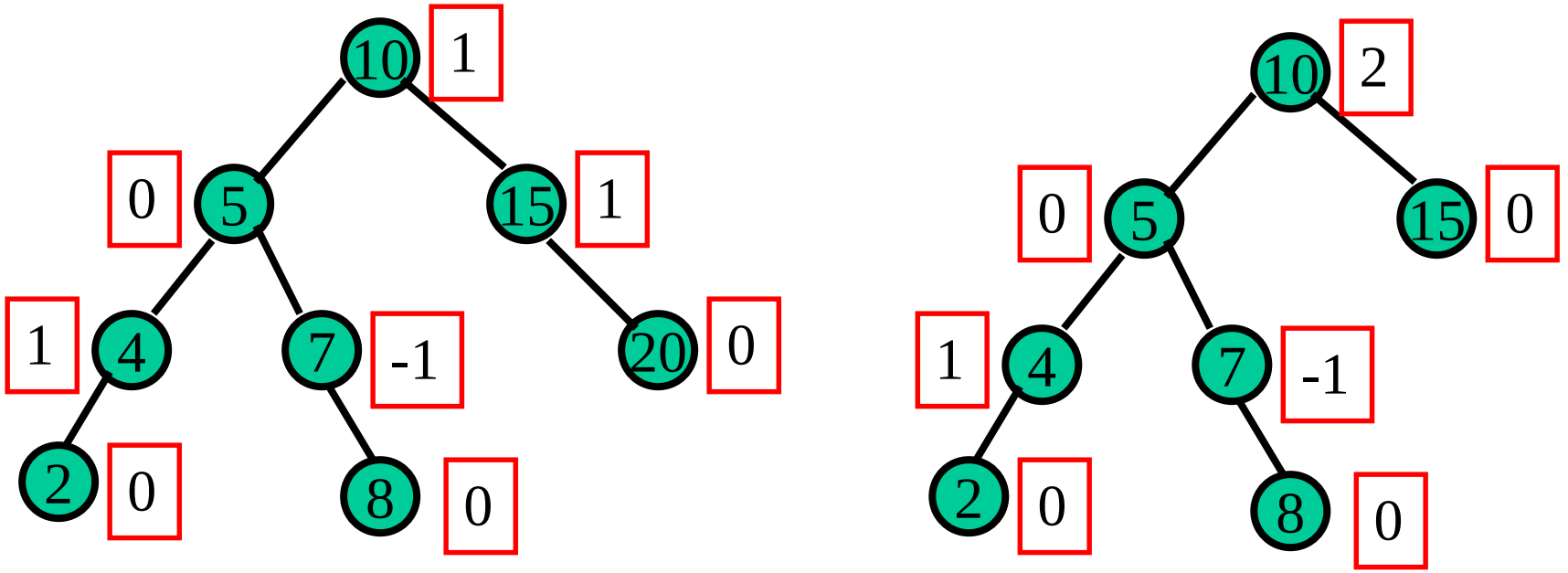
Bulent Tugrul

Balanced Search Trees

- Binary search trees in the average requires ‘logn’ comparison for each operation (searching, deletion and insertion), unfortunately in the worst case they need ‘n’ comparisons.
- Computer scientist come up with some solutions to find a structure that preserves the good properties of the classical binary search tree.
  - AVL Trees
  - 2-3 Trees
  - Red Black Trees

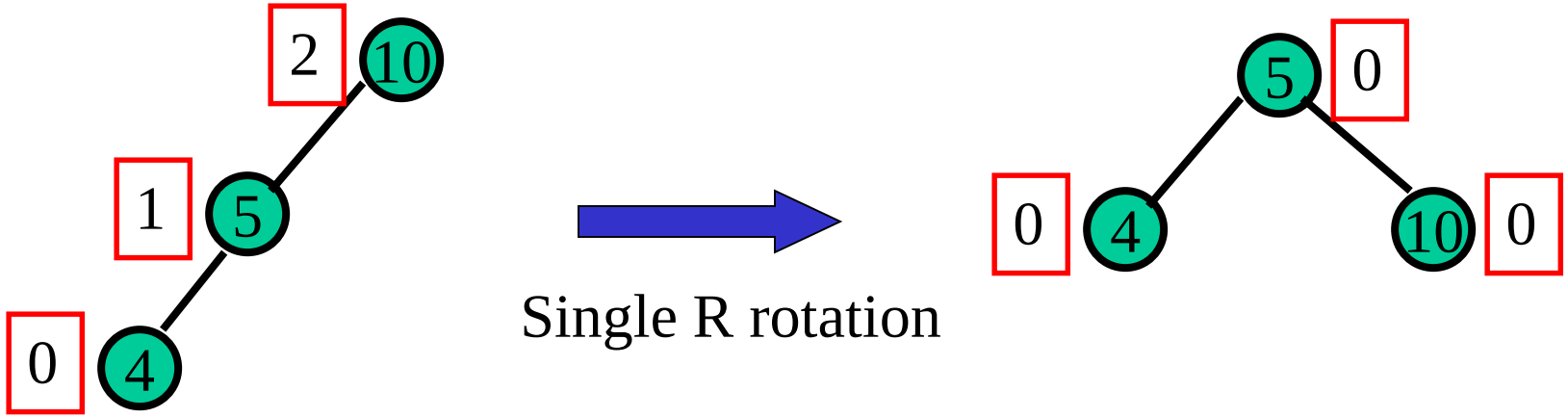
AVL Trees

- Definition: An AVL tree is binary search tree in which the balance factor of every node, which is defined as the difference between the heights of the node's left and right subtrees, is either 0 or +1 or –1.

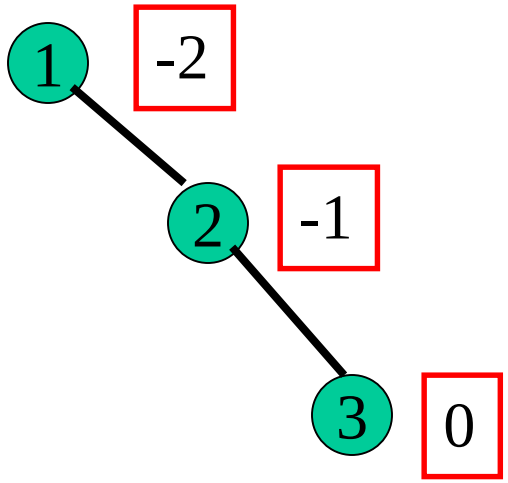


AVL Trees

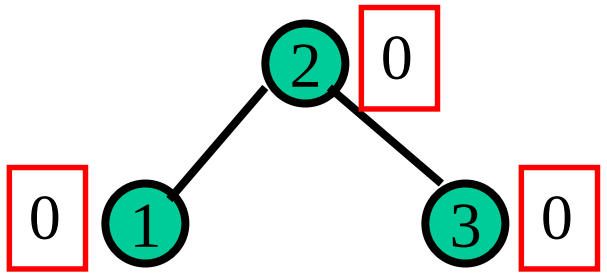
- If an insertion of a new node makes an AVL tree unbalanced, we transform the tree by rotation.
- A rotation in an AVL tree is a local transformations of its subtree rooted at a node whose balance has become either +2 or –2.
- There are only four types of rotations; in fact two of them are mirror images of the other two.

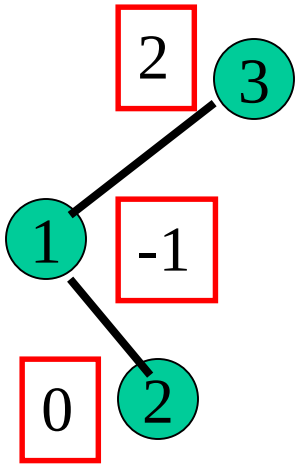


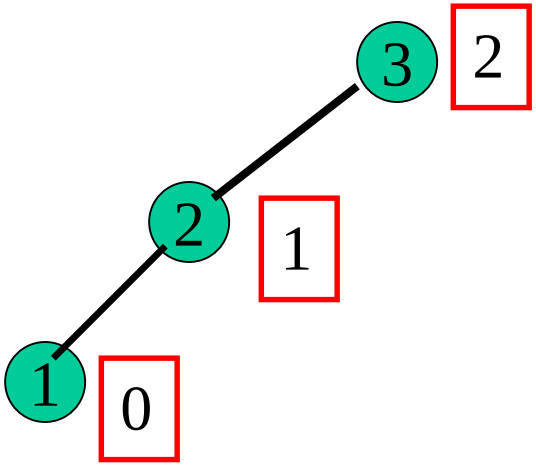
AVL Trees

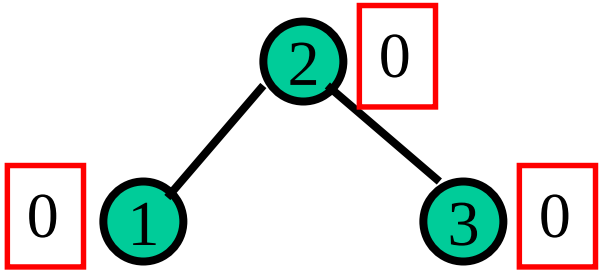


Single L rotation



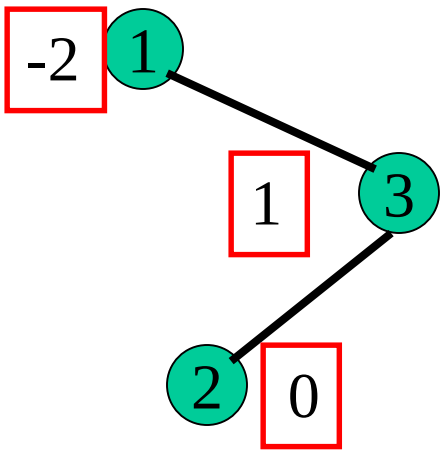


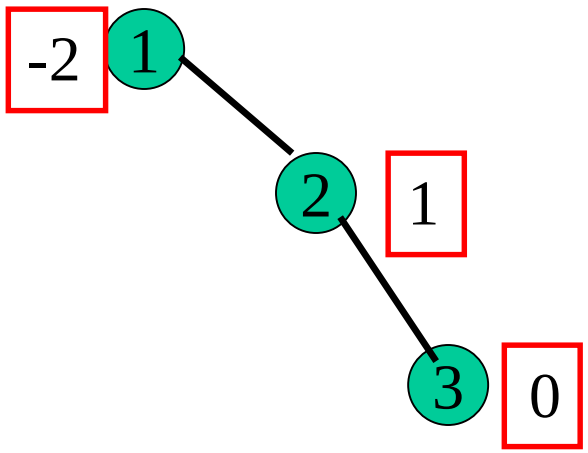


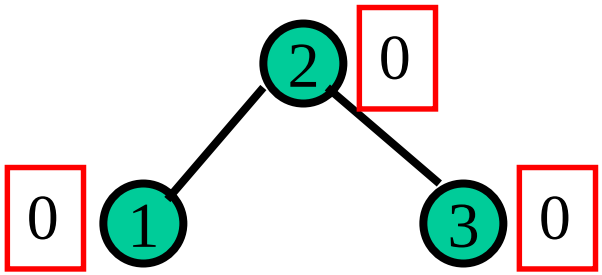


Double LR rotation

AVL Trees

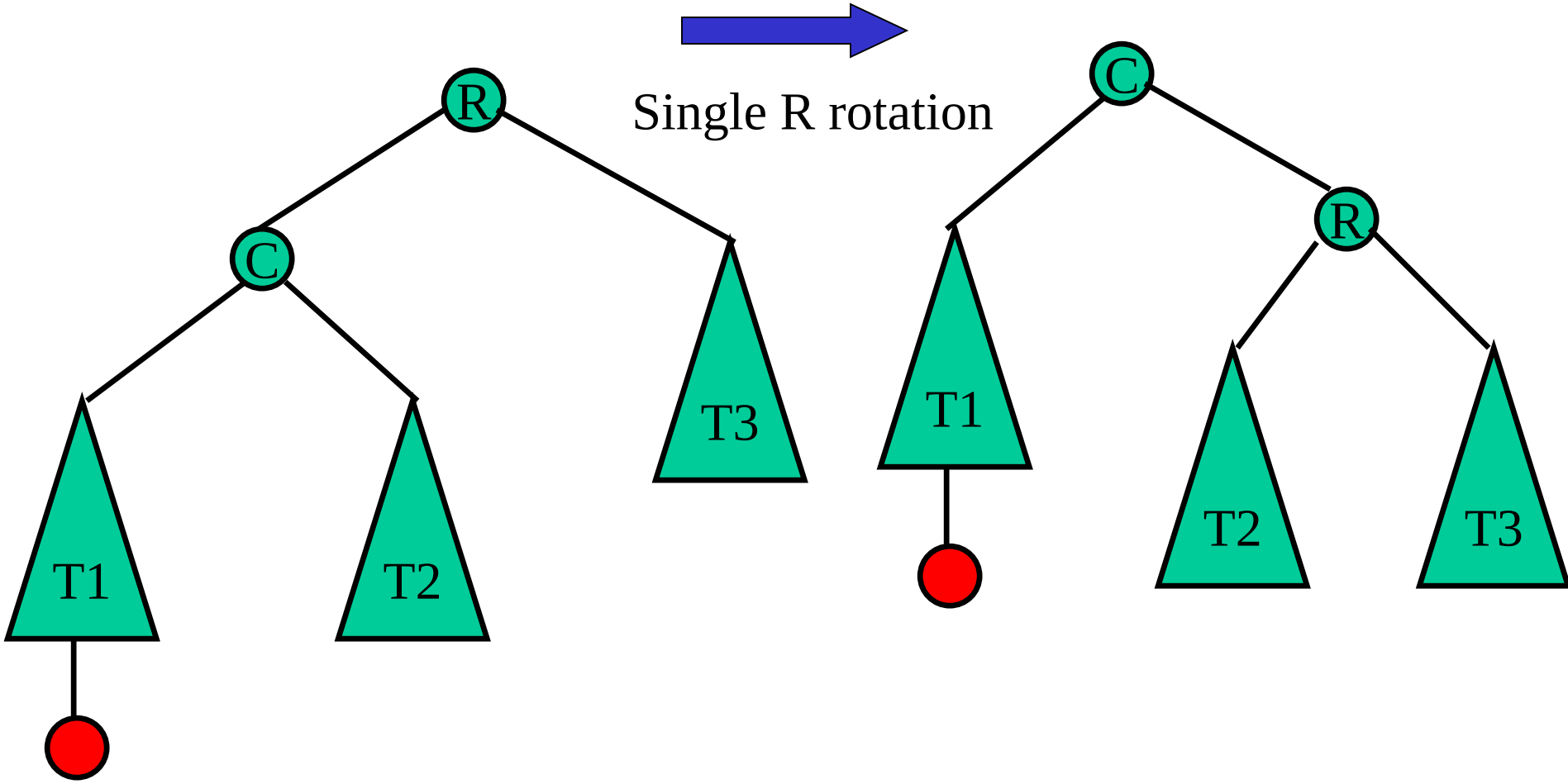






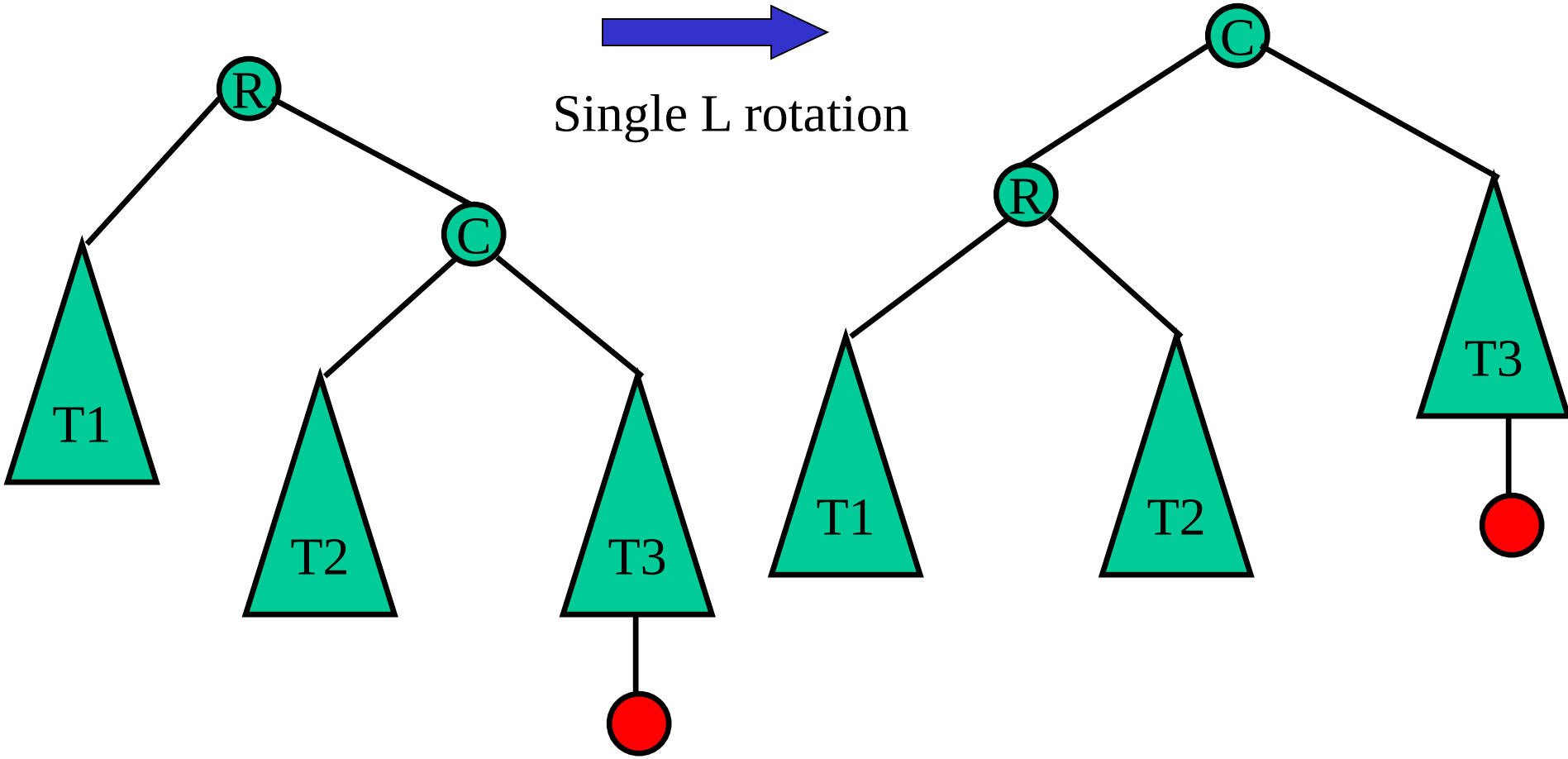
Double RL rotation

AVL Trees



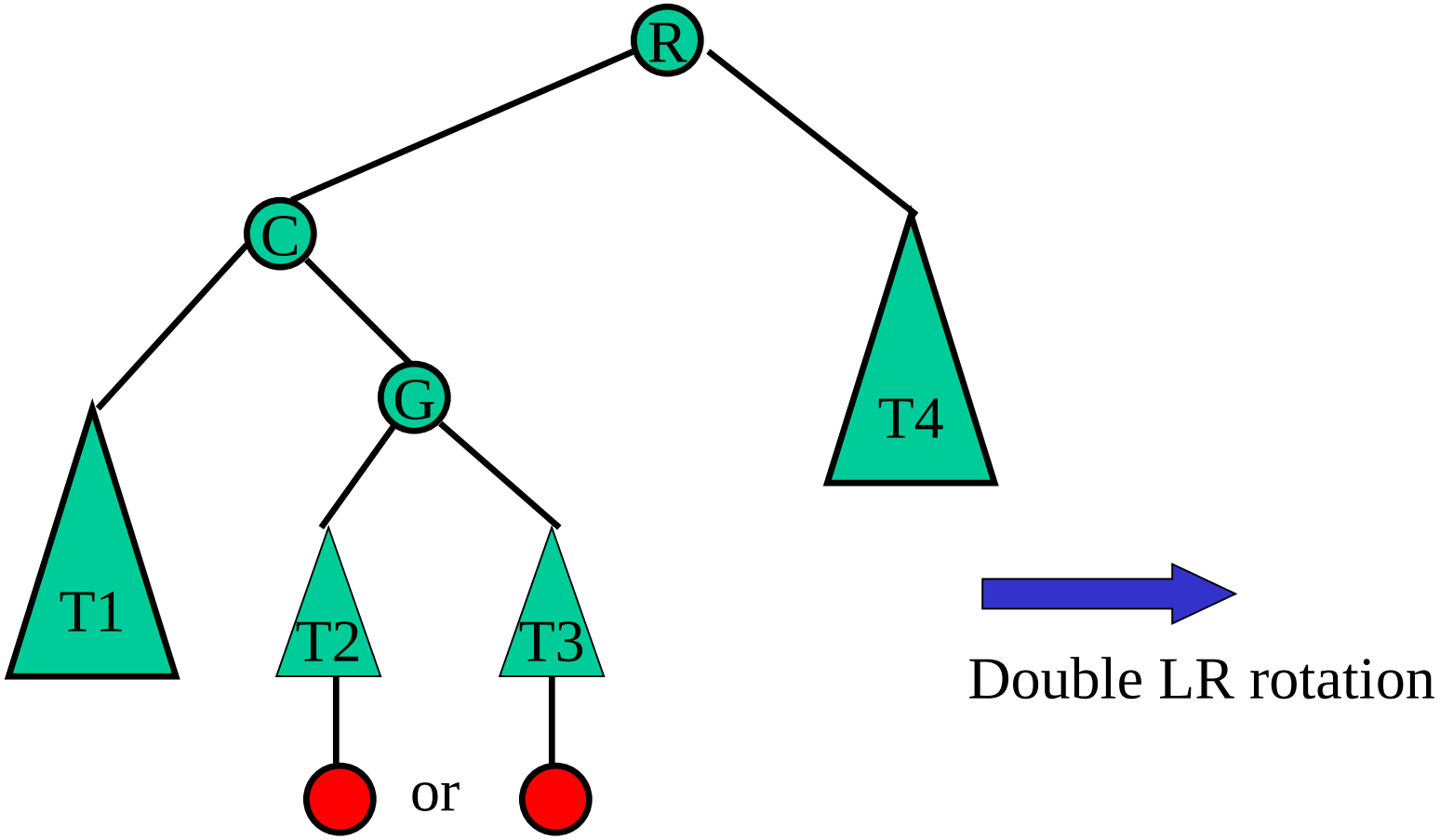
{T1} < C < {T2} < R < {T3}

AVL Trees



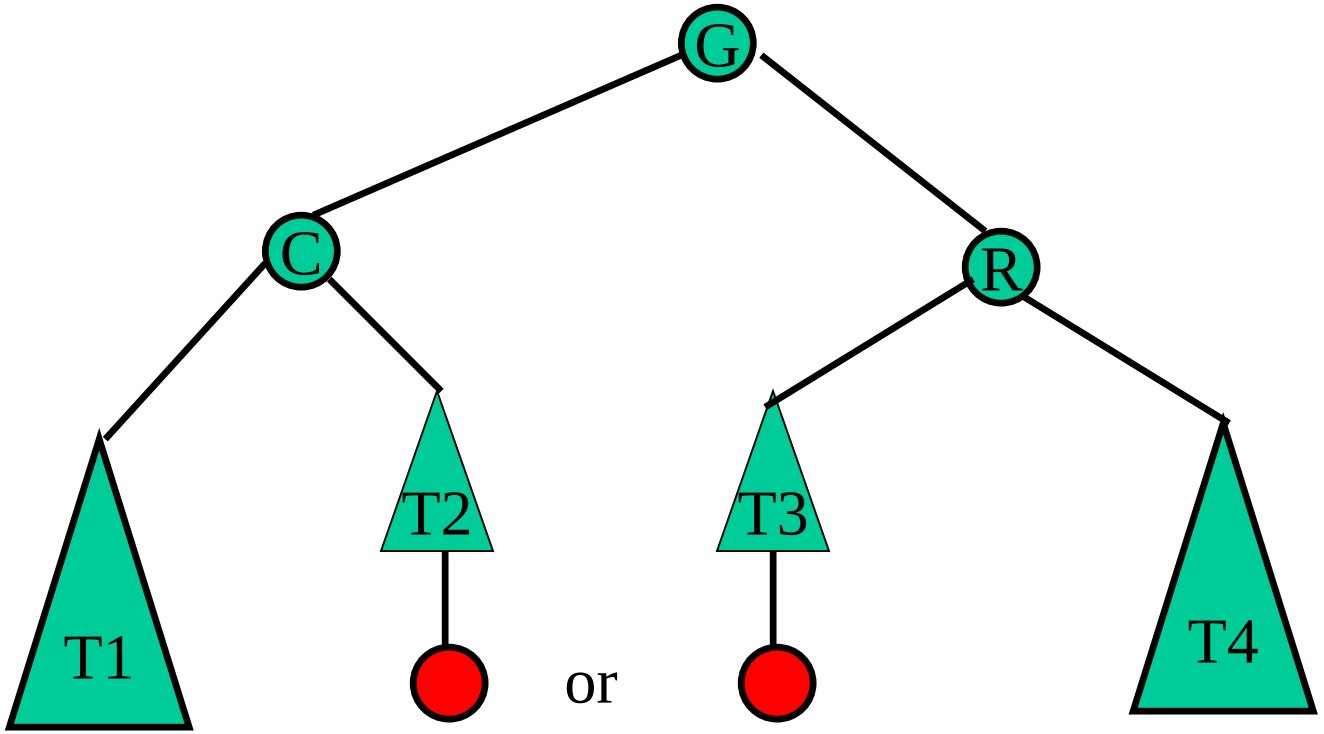
{T1} < R < {T2} < C < {T3}

AVL Trees



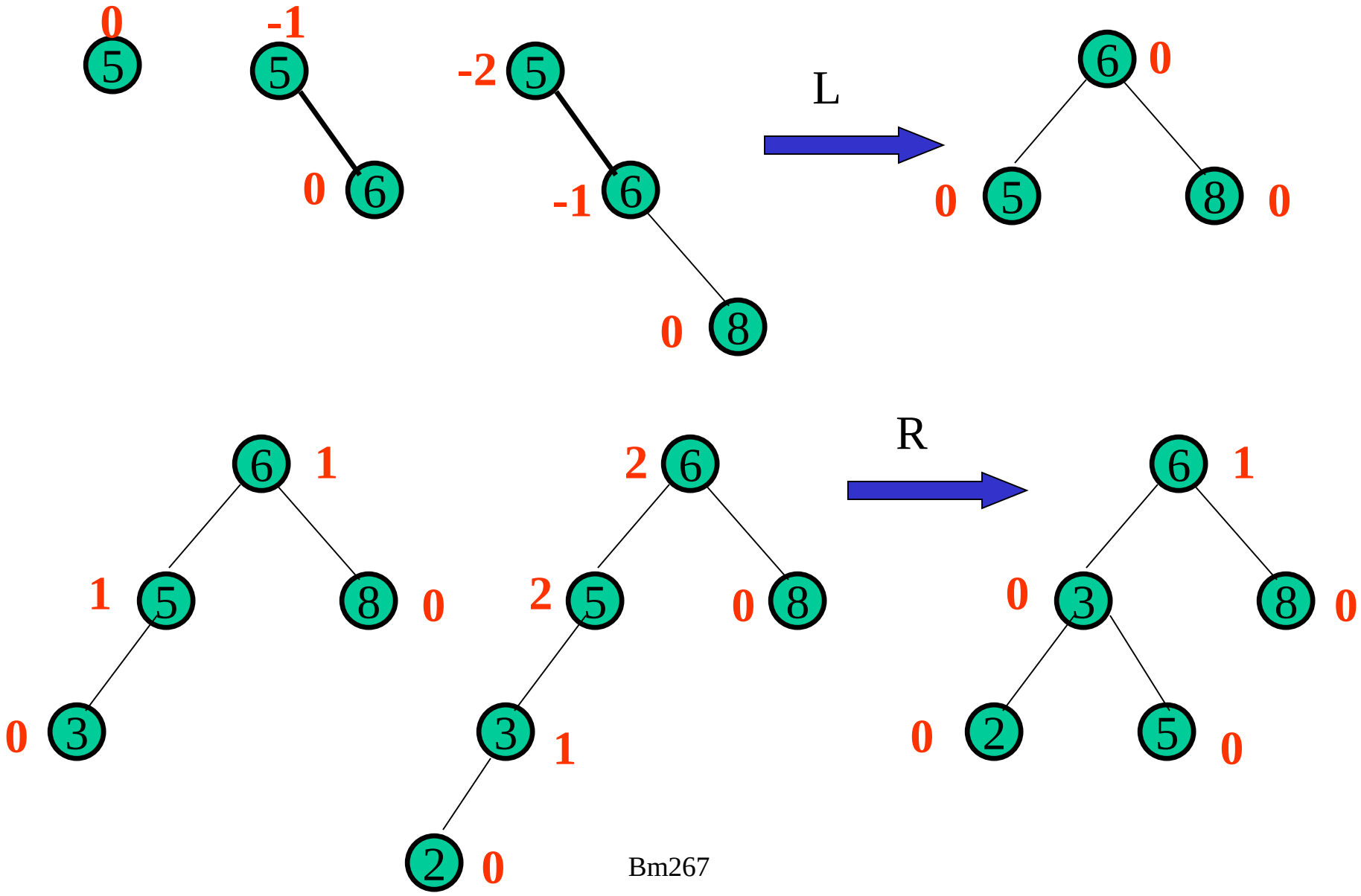
{T1} < C < {T2} < G < {T3} <R < {T4}

AVL Trees

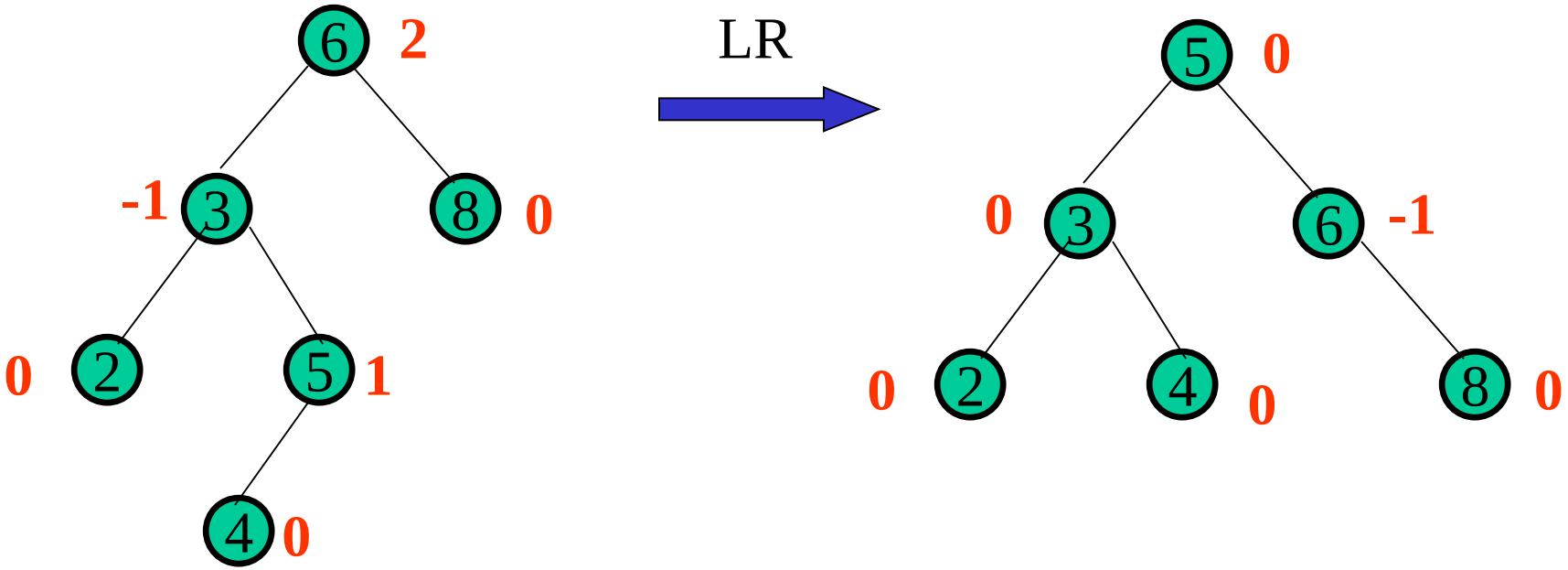


{T1} < C < {T2} < G < {T3} <R < {T4}

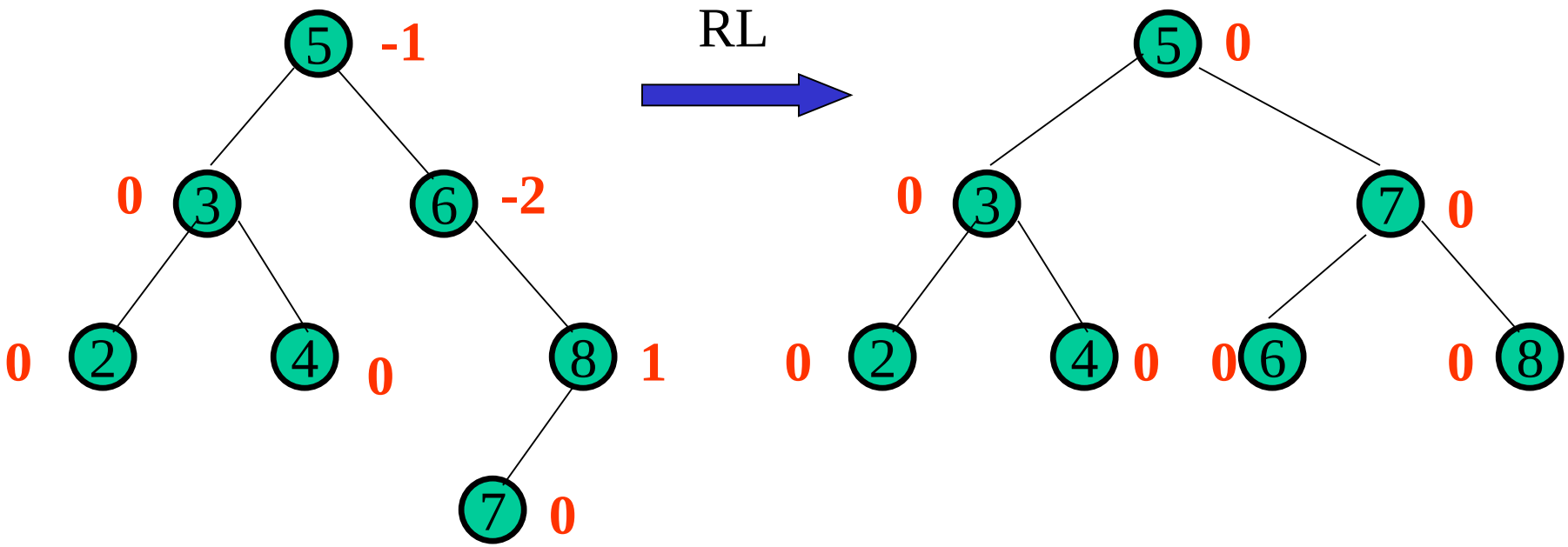
AVL Trees



AVL Trees



AVL Trees



AVL Trees

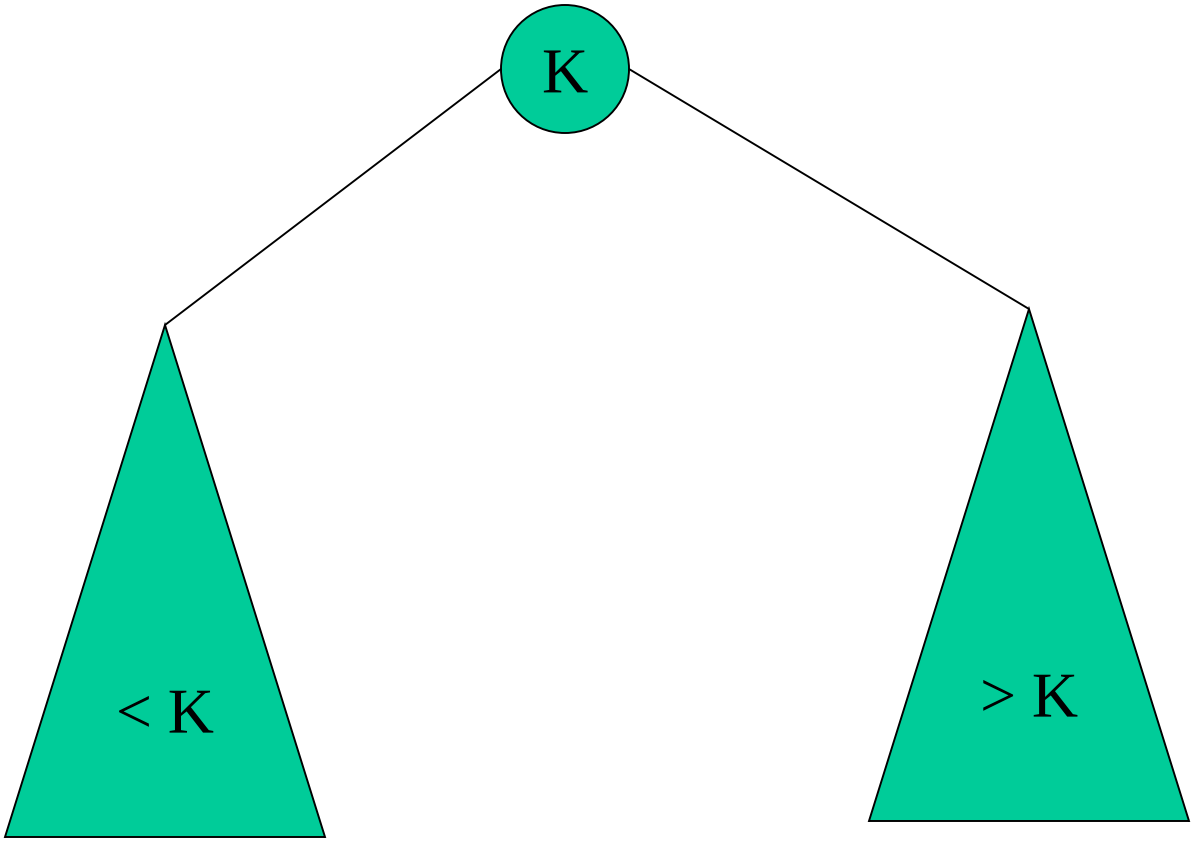
- How efficient is an AVL trees?
- The height h of any AVL tree with n nodes satisfies the inequalities

$$\log_2 n \leq h < 1.4405 \log_2 (n+2) - 1.3277$$

2-3 Search Trees

- The second idea of balancing a search tree is to allow more than one key in the same node.
- The simplest implementation of this idea is 2-3 trees.
- A 2-3 tree is a tree that can have nodes of two kinds: 2-nodes and 3-nodes.
- A 2-node contains a single key K and has two children: the left child serves as the root of a subtree whose keys are less than K and the right child serves as the root of a subtree whose keys are greater than K.

2-3 Search Trees

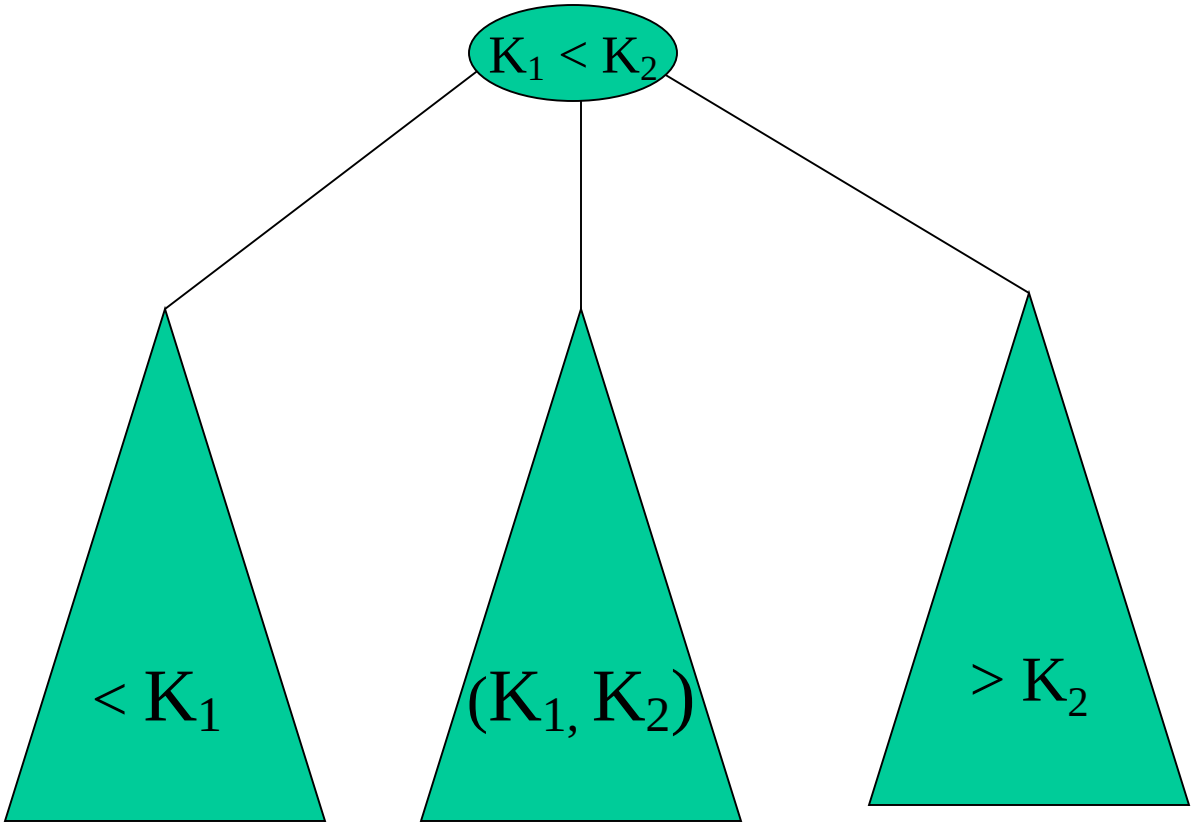


2-node

2-3 Search Trees

- A 3-node contains two ordered keys K_1 and K_2 ($K_1 < K_2$) and has three children.
- The leftmost subtree has keys less than K_1.
- The middle subtree has keys between K_1 and K_2.
- The rightmost subtree has keys greater than K_2

2-3 Search Trees

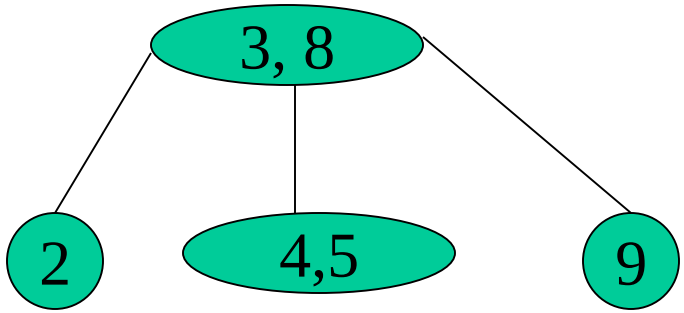


2-3 Search Trees

- The last requirement of the 2-3 tree is that its leaves must be on the same level.
- A 2-3 tree is always height balanced;the length of a path from the root of the tree to a leaf must be same for every leaf.
- Searching for a given key K in a 2-3 tree quite straightforward. We start with the root. If the root is a 2-node, we act as if it were a binary search tree: we either stop if K is equal to the root's key or continue the search in the left or right subtree.

2-3 Search Trees

- If the root is a 3-node, we know after no more than two key comparisons whether the search can be stopped (If K is equal to one of the root's keys) or in which of the root's three subtrees it needs to be continued.



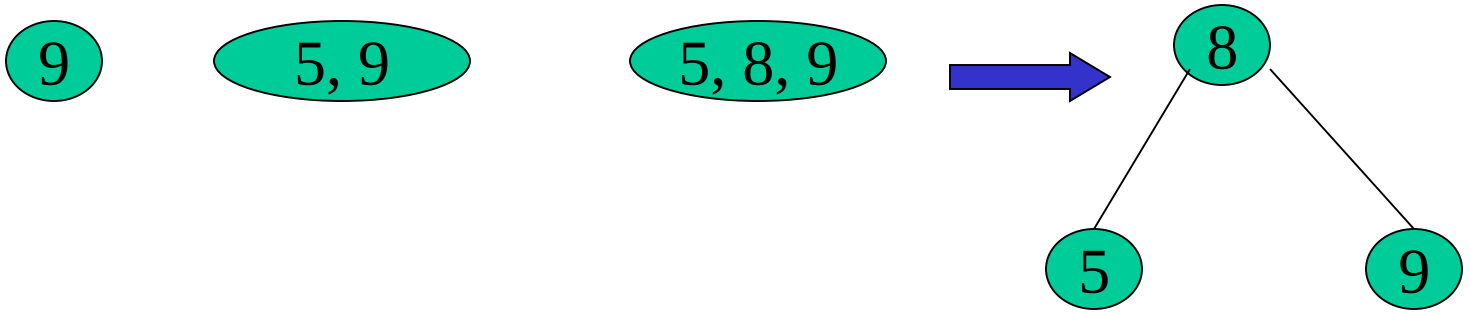
2-3 Search Trees

- Inserting a new key in a 2-3 tree is done as follows:
  - We always insert a new key K at a leaf except for the empty tree.
  - The appropriate leaf is found by performing a search for K.
    - If the leaf in question is a 2-node key, we insert K there as either the first or the second key, depending on whether K is smaller or larger than the node's old key.
    - If the leaf is a 3- node, we split the leaf in two; The smallest of the three is put in the first leaf, the largest key is put in the second leaf , while the middle key is promoted to the old leaf's parent.

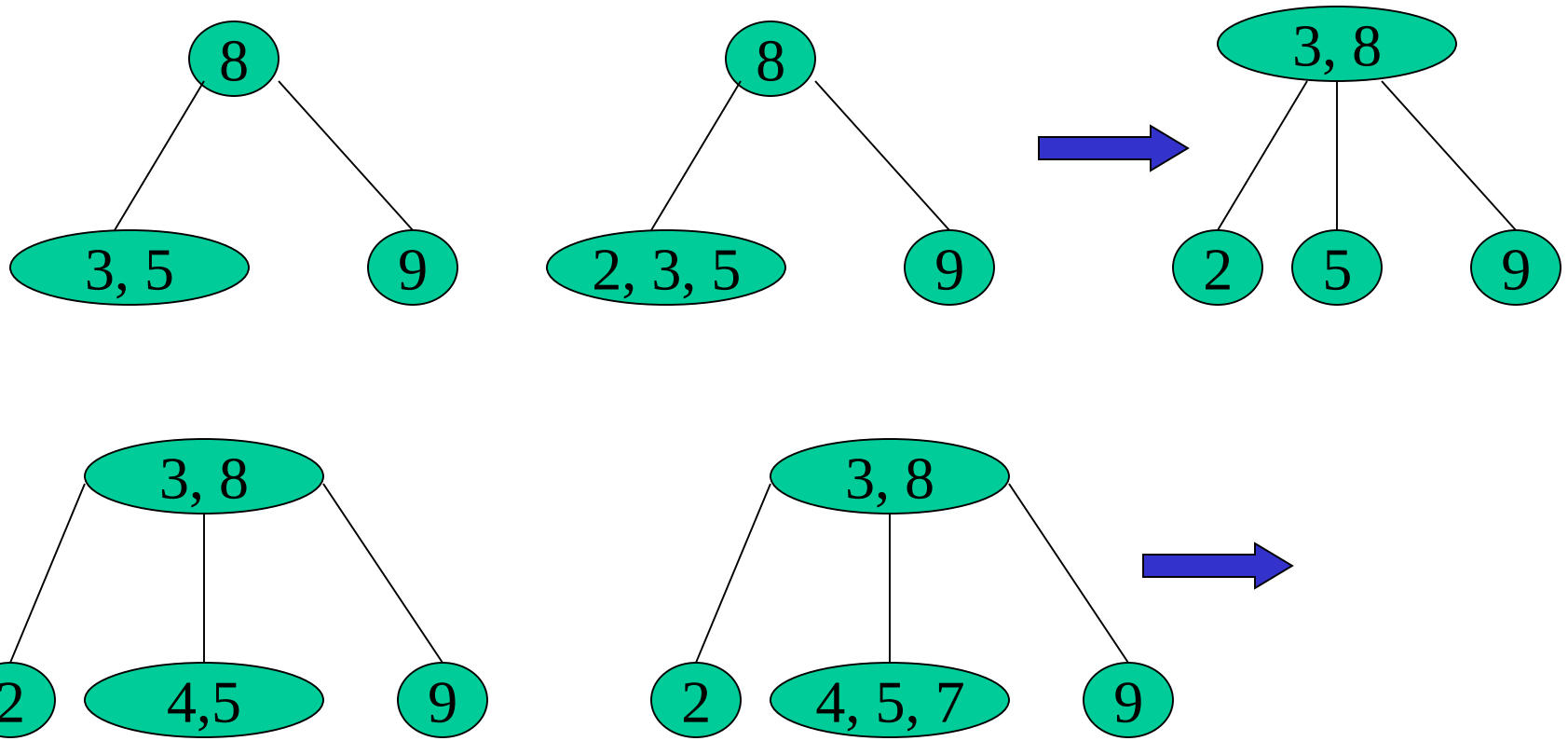
2-3 Search Trees

- If the leaf happens to be the tree's root, a new root is created to accept the middle key.
- Note that promotion of a middle key to its parent can cause the parent's overflow and hence can lead to several node splits along the chain of the leaf's ancestors.

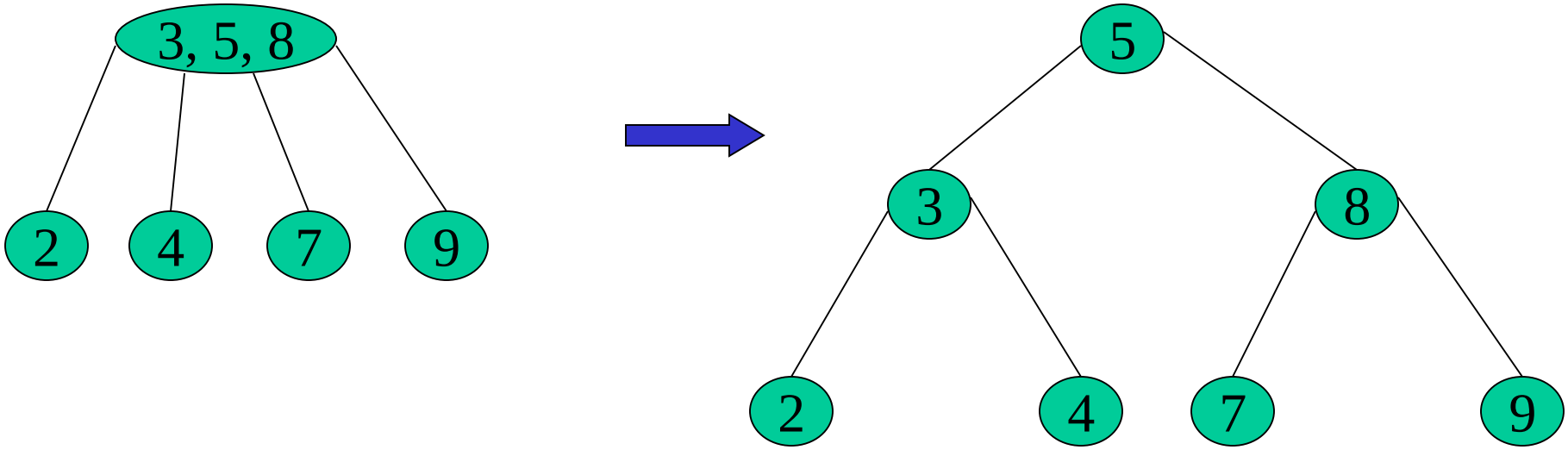
9,5,8,3,2,4,7



2-3 Search Trees



2-3 Search Trees



2-3 Search Trees

- The efficiency of the dictionary operations depends on the tree's height.
- Let's find an upper bound for a 2-3 tree. A 2-3 tree of height h with the smallest number of keys is full tree of 2-nodes.

  $n >= 1+2+...+2^h = 2^{h+1} -1$

  and hence

  $h <= \log_2(n+1) -1$

- On the other hand, a 2-3 tree of height h with the largest number of keys is a full of 3-nodes, each with two keys and three children.

  $n <= 2*1+2*3+...+2*3^h = 3^{h+1} -1$

  $h>=\log_3(n+1) -1$

Therefore: $\mathbf{\log_3(n+1)-1 <= h <= \log_2(n+1)-1}$