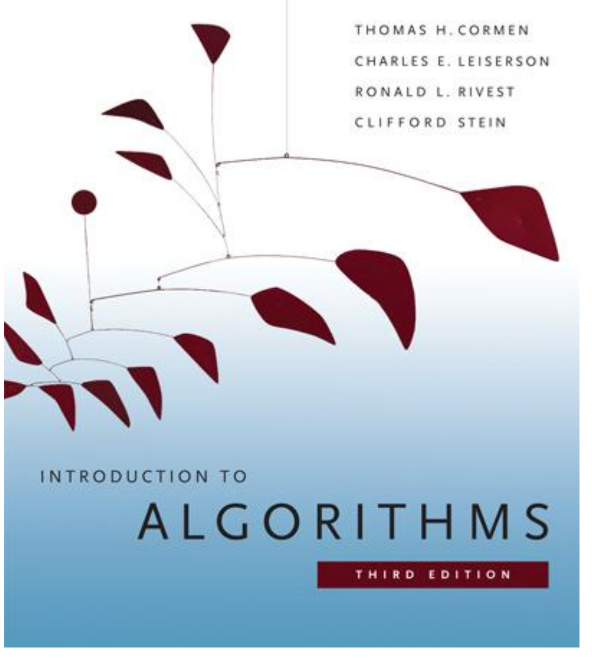
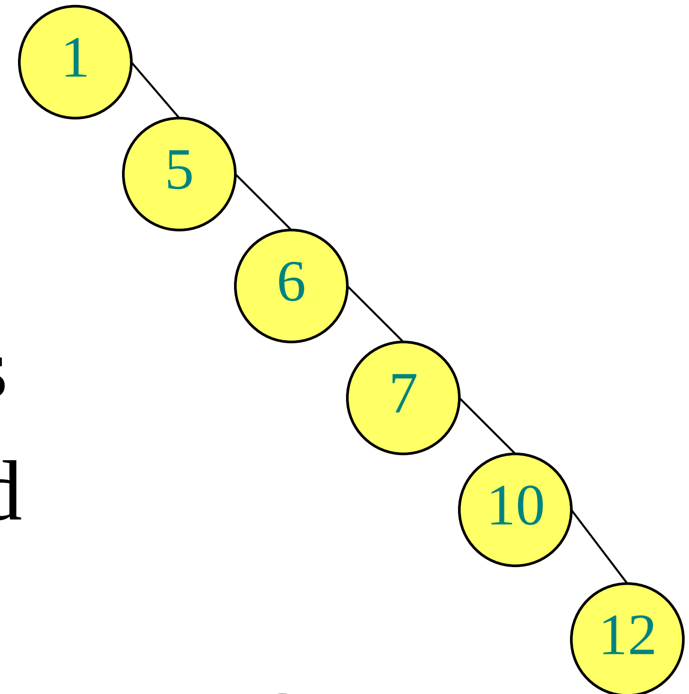
6.006- *Introduction to Algorithms*

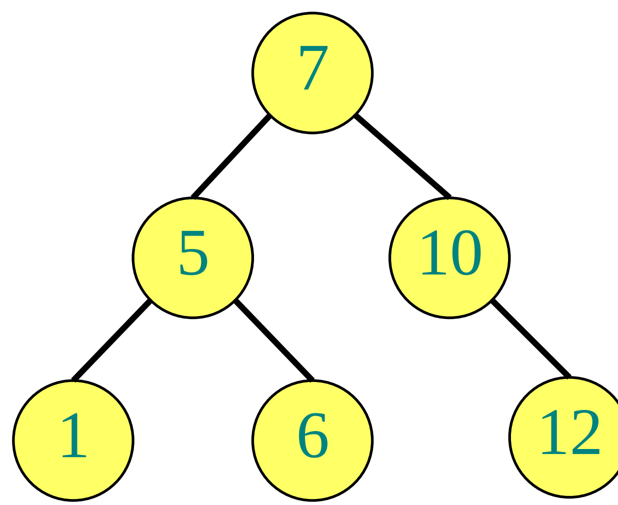


Lecture 6

Lecture Overview



- Review: Binary Search Trees
- Importance of being balanced
- Balanced BSTs
  – AVL trees
    - definition
    - rotations, updates



Binary Search Trees (BSTs)

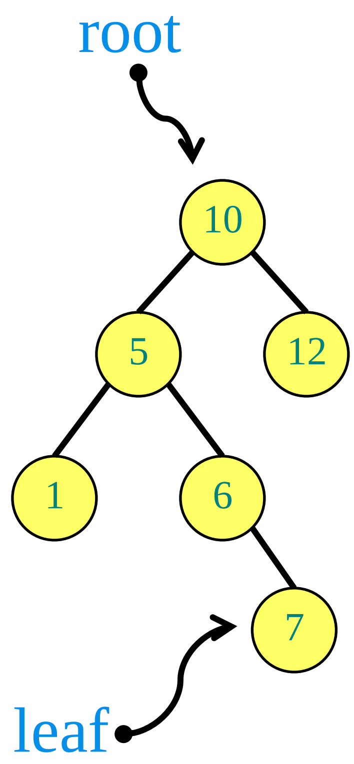
- Each node x has:
  - key[x]
  - Pointers: left[x], right[x], p[x]

- Property: for any node x:
  - For all nodes y in the left subtree of x:

    $$key[y] \leq key[x]$$

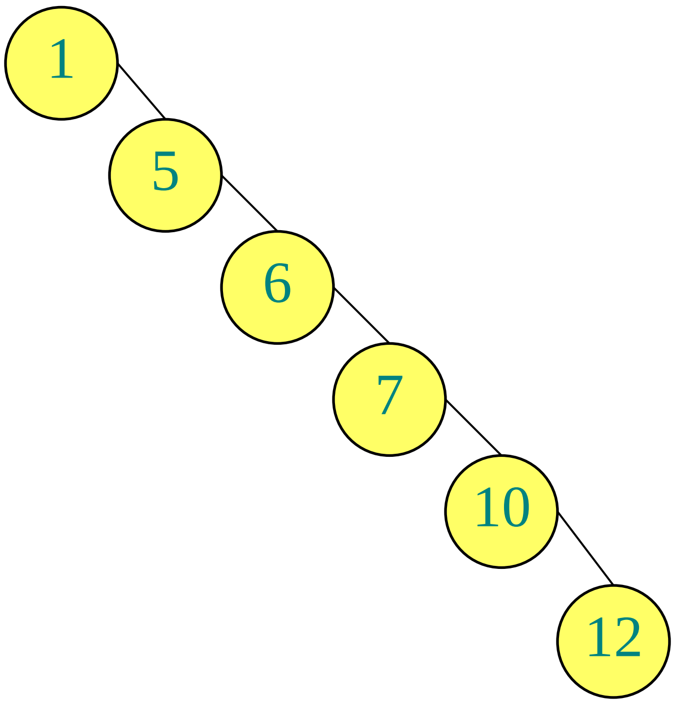
  - For all nodes y in the right subtree of x:

    $$key[y] \geq key[x]$$

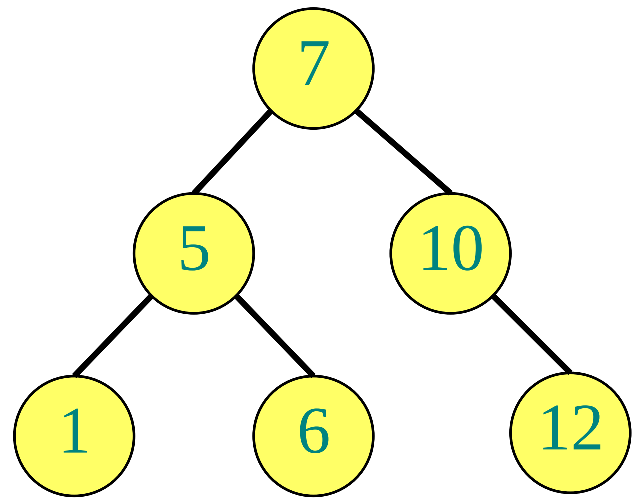


height = 3

The importance of being balanced



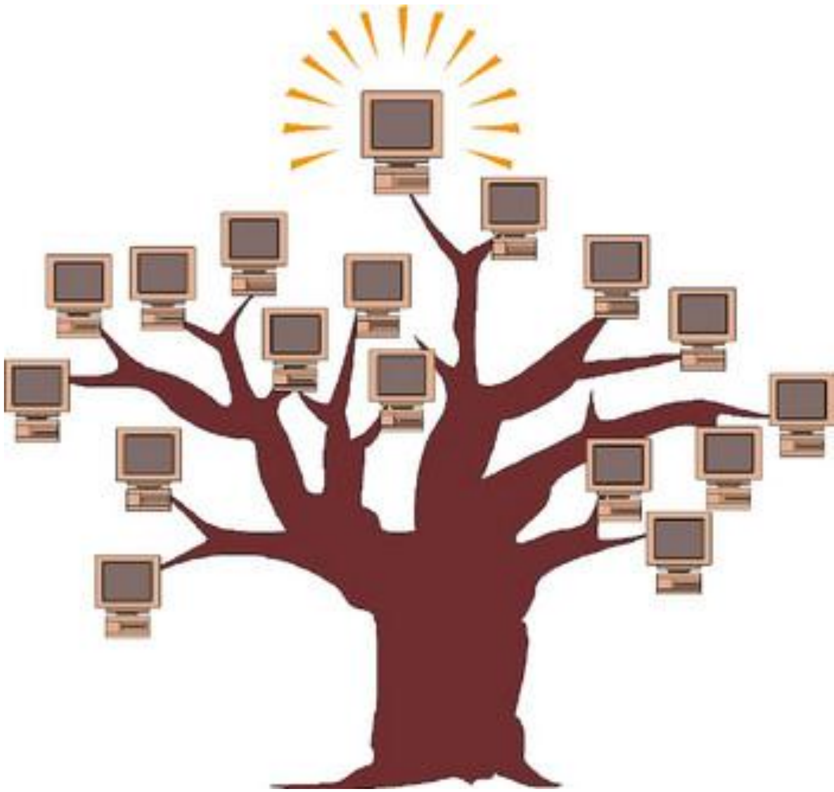
$h = \Theta(n)$



$h = \Theta(\log n)$

Balanced BST Strategy

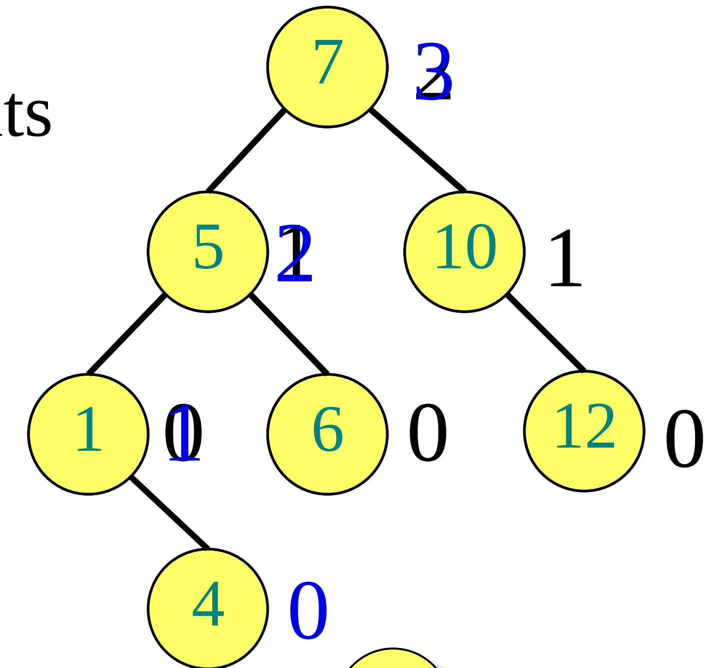
- Augment every node with some data
- Define a local invariant on data
- Show (prove) that invariant guarantees $\Theta(\log n)$ height
- Design tree update procedure to maintain data and the invariant



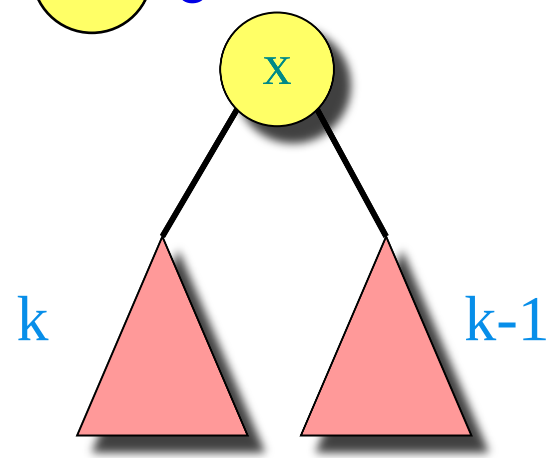
AVL Trees

[Adelson-Velskii and Landis'62]

- **Data**: for every node, maintain its height ("augmentation")
  - Leaves have height 0
  - NIL has "height" -1



- **Invariant**: for every node x, the heights of its left child and right child differ by at most 1



AVL trees have height $\Theta(\log n)$

Invariant: for every node x, the heights of its left child and right child differ by at most 1

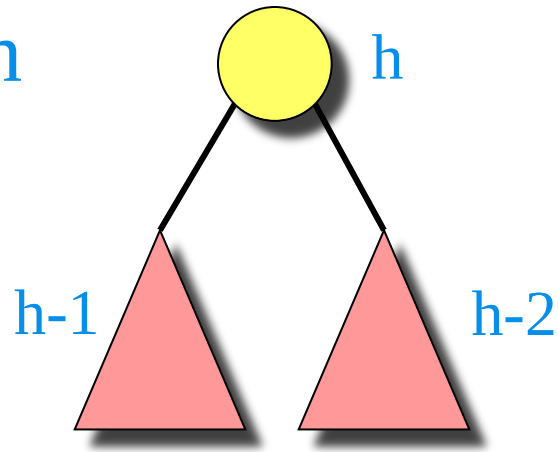
- Let n_h be the minimum number of nodes of an AVL tree of height h
- We have $n_h \geq 1 + n_{h-1} + n_{h-2}$

$$\Rightarrow n_h > 2n_{h-2}$$

$$\Rightarrow n_h > 2^{h/2}$$

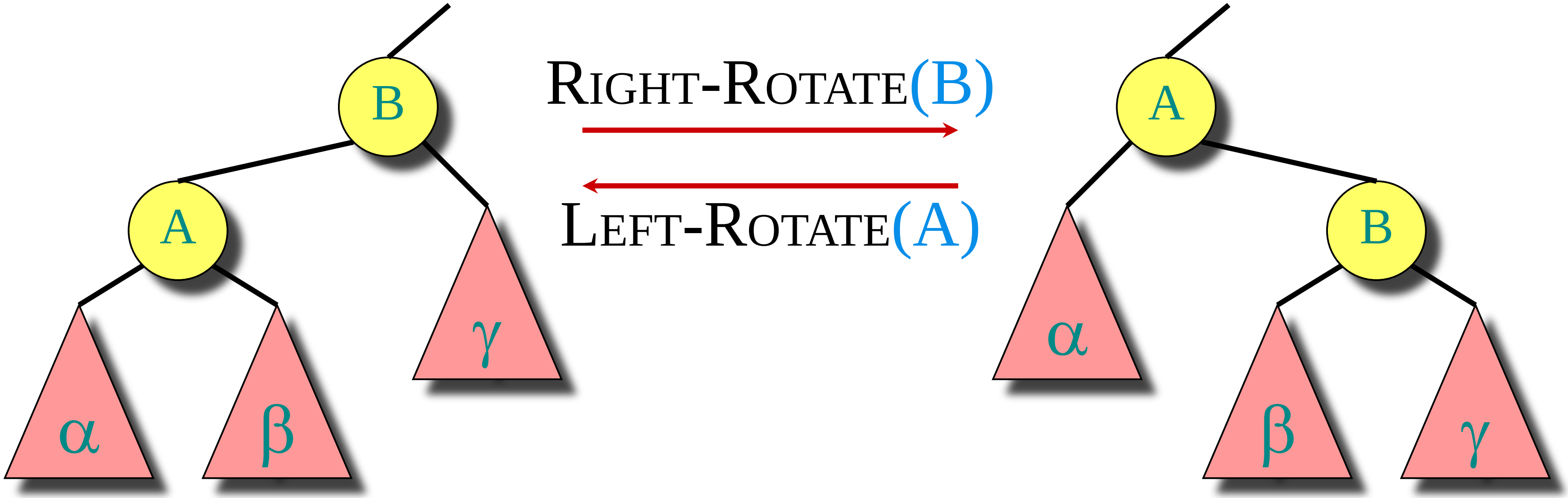
$$\Rightarrow h < 2 \lg n_h$$

- The constant "2" can be improved



How can we maintain the invariant ?

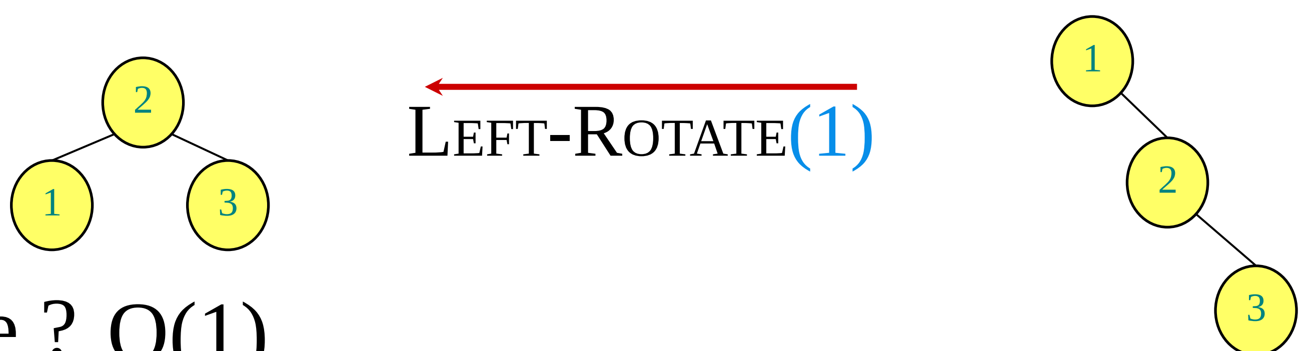
Tree acrobatics - rotation



Rotations maintain the inorder ordering of keys:

$$a \in \alpha,\ b \in \beta,\ c \in \gamma \ \Rightarrow\ a \le A \le b \le B \le c.$$

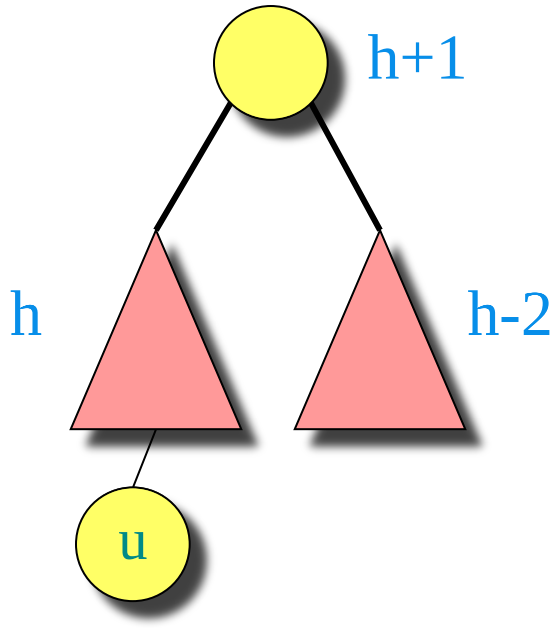
Moreover, they can reduce the height!



Time ? O(1)

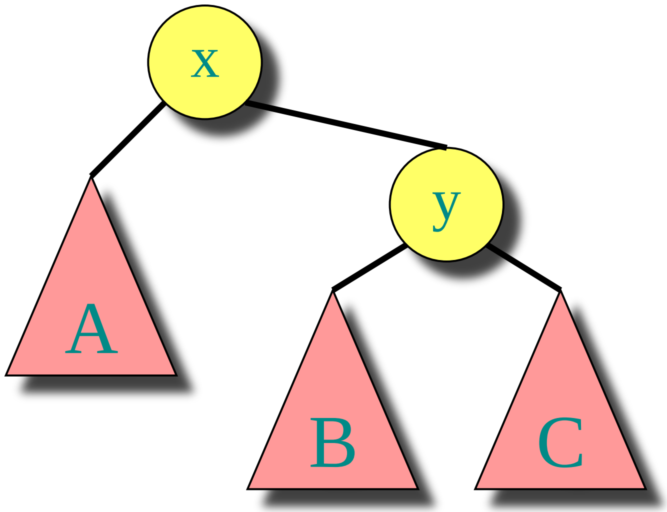
Insertions

- Insert new node u as in the simple BST
  - Can create imbalance
- Work your way up the tree, restoring the balance
- Similar issue/solution when deleting a node

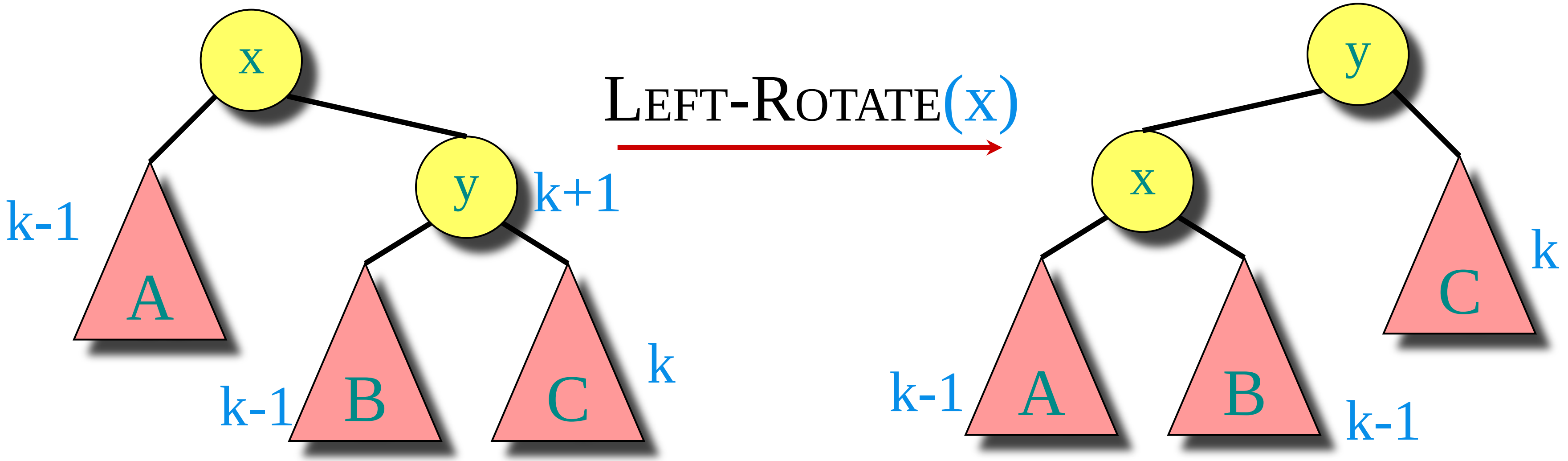


Balancing

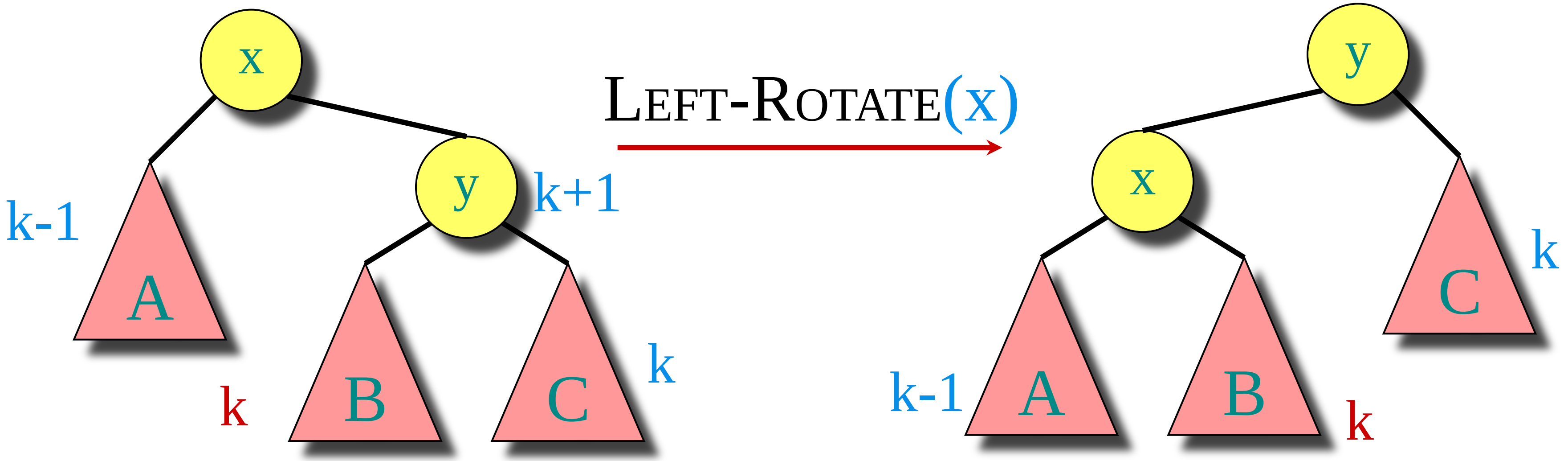
- Let x be the lowest "violating" node
  - We will fix the subtree of x and move up
- Assume the right child of x is deeper than the left child of x (x is "right-heavy")
- Scenarios:
  - Case 1: Right child y of x is right-heavy
  - Case 2: Right child y of x is balanced
  - Case 3: Right child y of x is left-heavy



Case 1: y is right-heavy

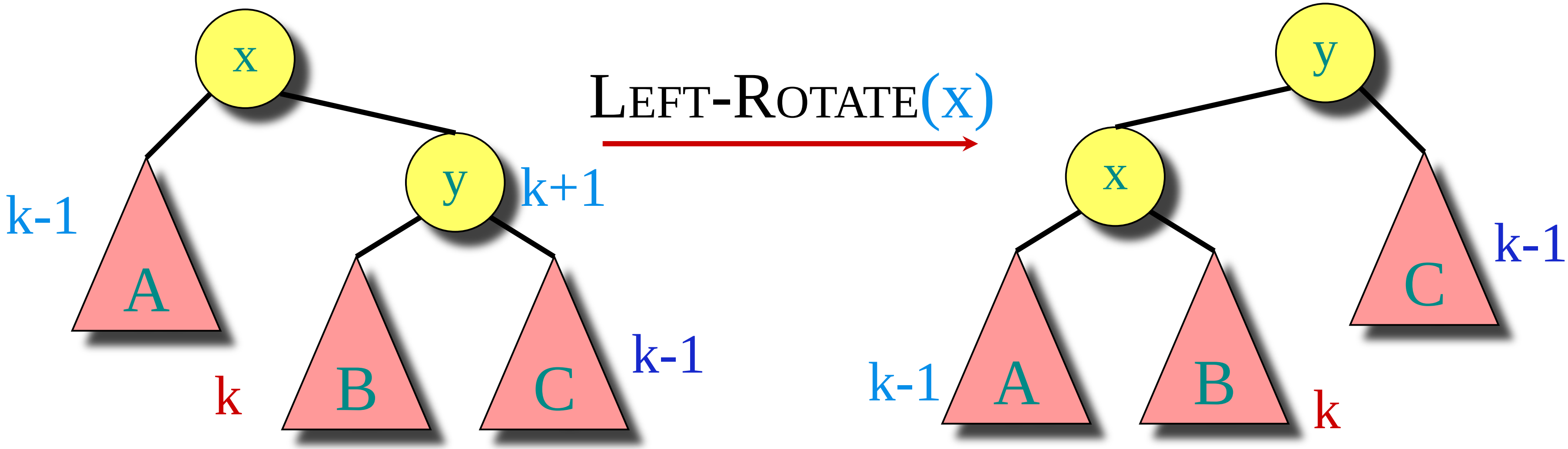


Case 2: y is balanced



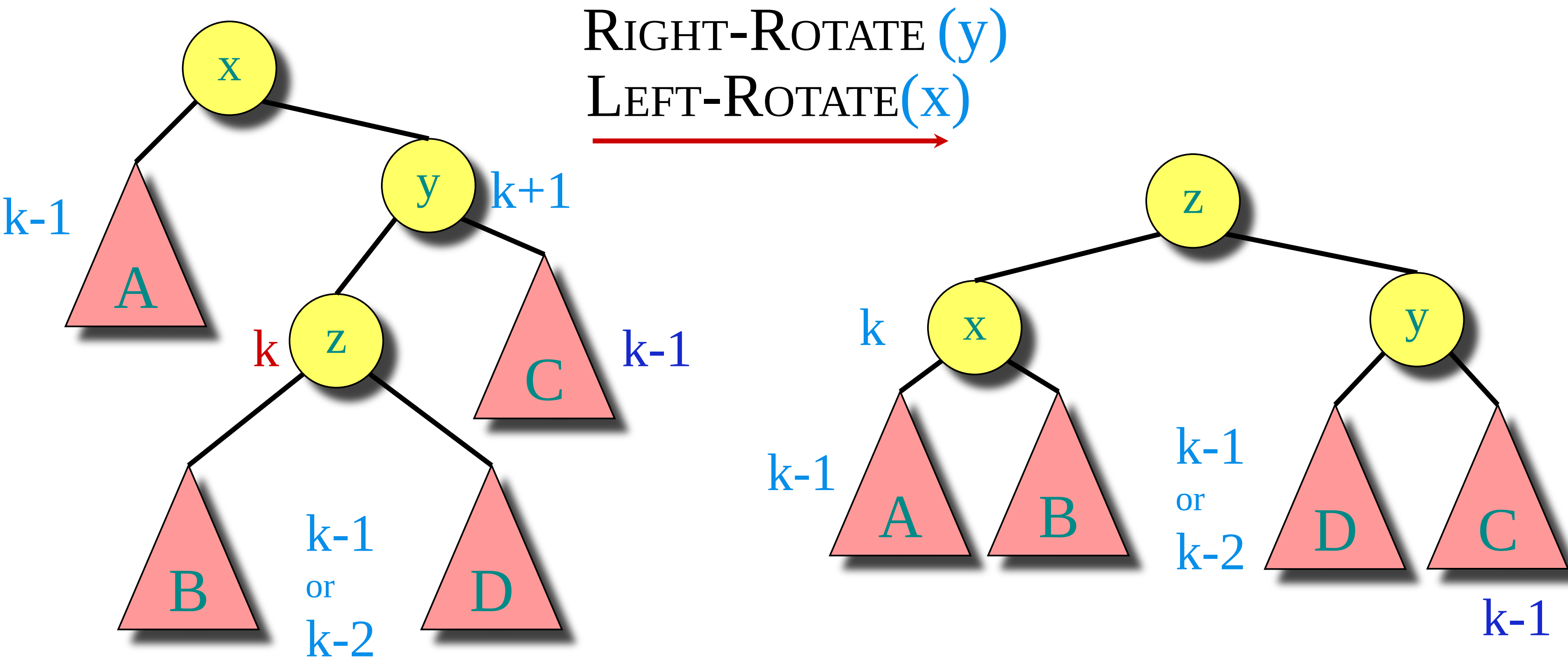
Same as Case 1

Case 3: y is left-heavy



Need to do more …

Case 3: y is left-heavy



And we are done!

Conclusions

- Can maintain balanced BSTs in O(log n) time per insertion
- Search etc take O(log n) time