COMP3143Data Structures and Algorithms

AVL-Trees (Part 2: Double Rotations)

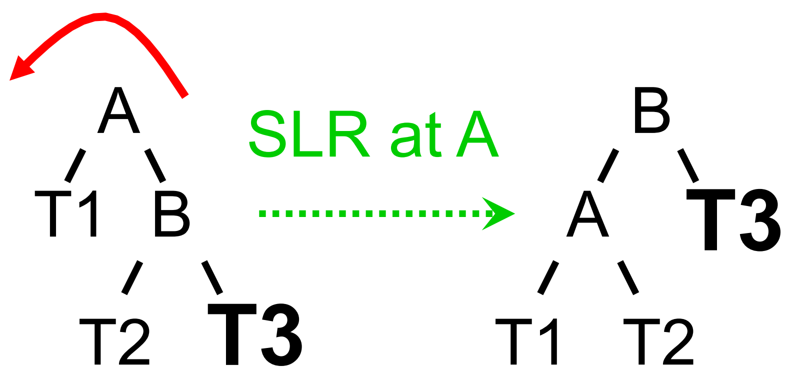
Review of Rotations

When the AVL property is lost we can rebalance the tree via rotations

- Single Right Rotation (SRR)
  - Performed when A is unbalanced to the left (the left subtree is 2 higher than the right subtree) and B is left-heavy (the left subtree of B is 1 higher than the right subtree of B).

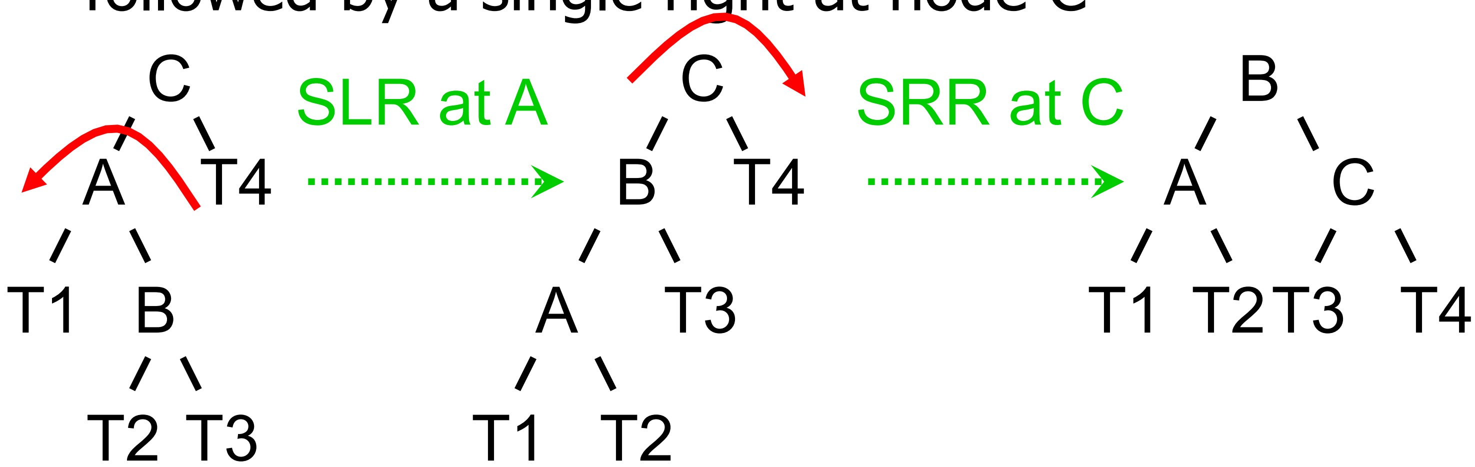
Rotations

- Single Left Rotation (SLR)
  - performed when A is unbalanced to the right (the right subtree is 2 higher than the left subtree) and B is right-heavy (the right subtree of B is 1 higher than the left subtree of B).



Rotations

- Double Left Rotation (DLR)
  - Performed when C is unbalanced to the left (the left subtree is 2 higher than the right subtree), A is right-heavy (the right subtree of A is 1 higher than the left subtree of A)
  - Consists of a single left rotation at node A, followed by a single right at node C

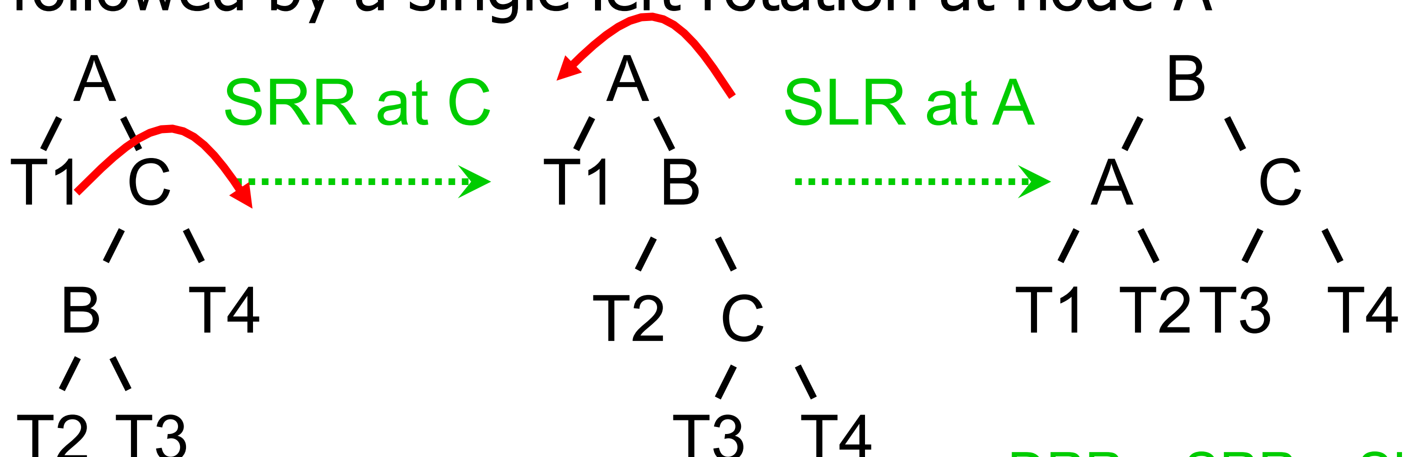


Note: this is case 1

DLR = SLR + SRR

Rotations

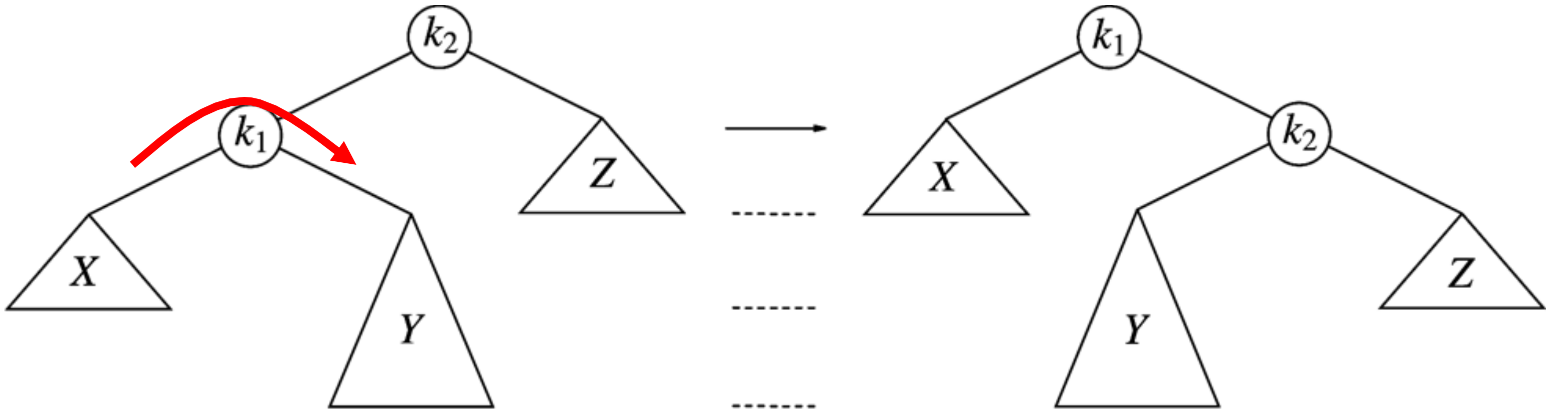
- Double Right Rotation (DRR)
  - Performed when A is unbalanced to the right (the right subtree is 2 higher than the left subtree), C is left heavy (the left subtree of C is 1 higher than the right subtree of C)
  - Consists of a single right rotation at node C, followed by a single left rotation at node A



DRR = SRR + SLR

Note: this is case 4!

Recall Cases 2&3

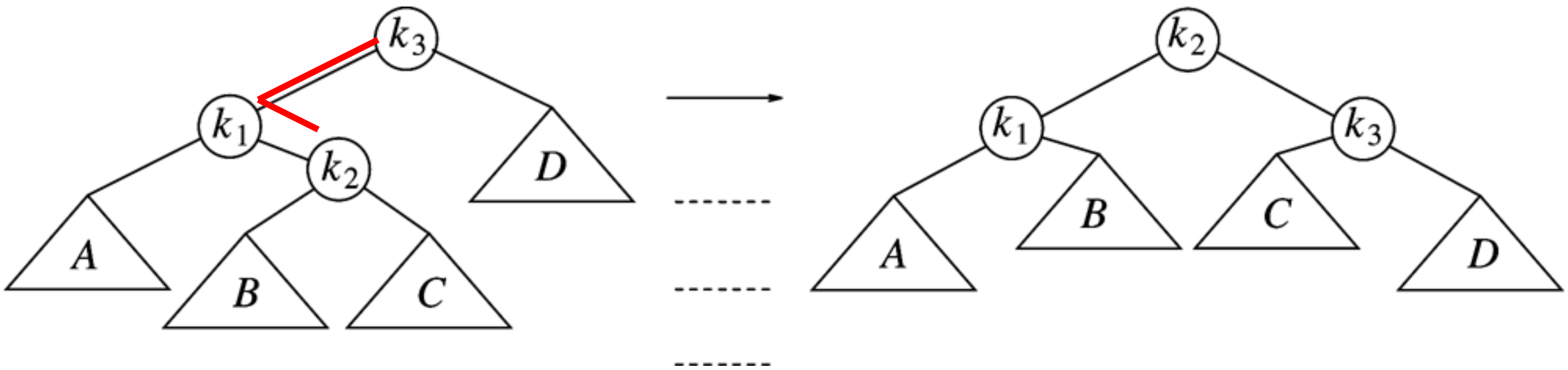


Case 2: violation in k2 because of insertion in subtree Y

Single rotation fails

- Single rotation fails to fix case 2&3
- Take case 2 as an example (case 3 is a symmetric to it)
  - The problem is that the subtree Y is too deep
  - Single rotation doesn't make Y any less deep…

Double Rotation

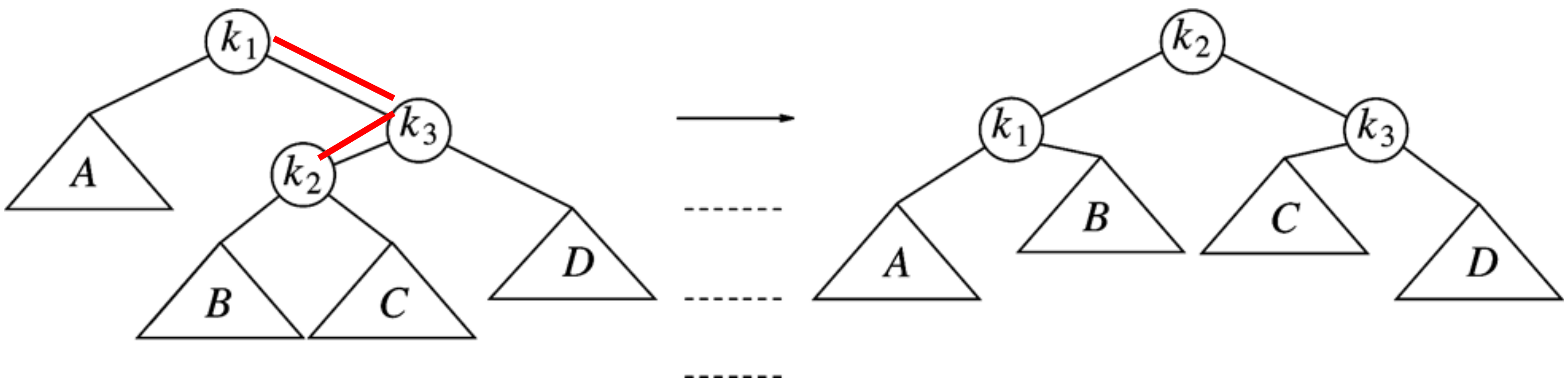


Double rotation to fix case 2

- Facts
  - The new key is inserted in the subtree B or C
  - The AVL-property is violated at k_3
  - k_3-k_1-k_2 forms a zig-zag shape: LR case
- Solution
  - place k_2 as the new root

k2 is the median of k1, k2, and k3

Double Rotation to fix Case 3(right-left)



Double rotation to fix case 3

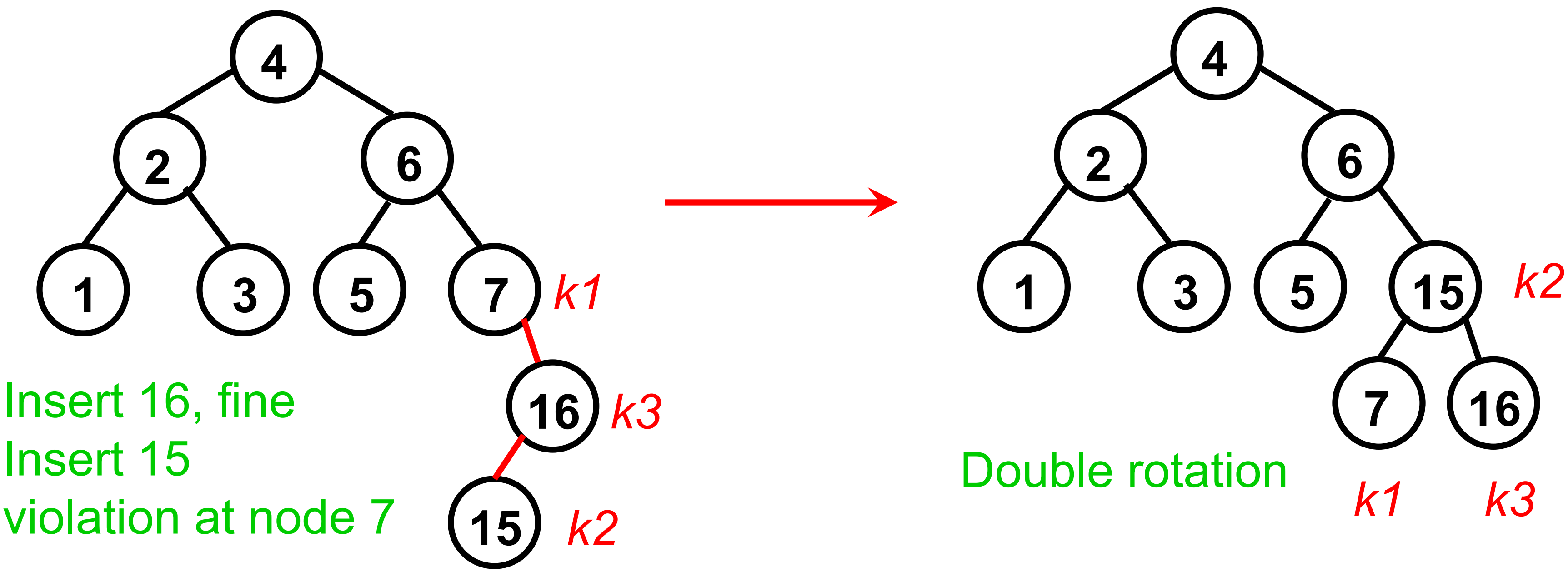
- Facts
  - The new key is inserted in the subtree B or C
  - The AVL-property is violated at k_1
  - k_1-k_3-k_2 forms a zig-zag shape
- Case 3 is a symmetric case to case 2

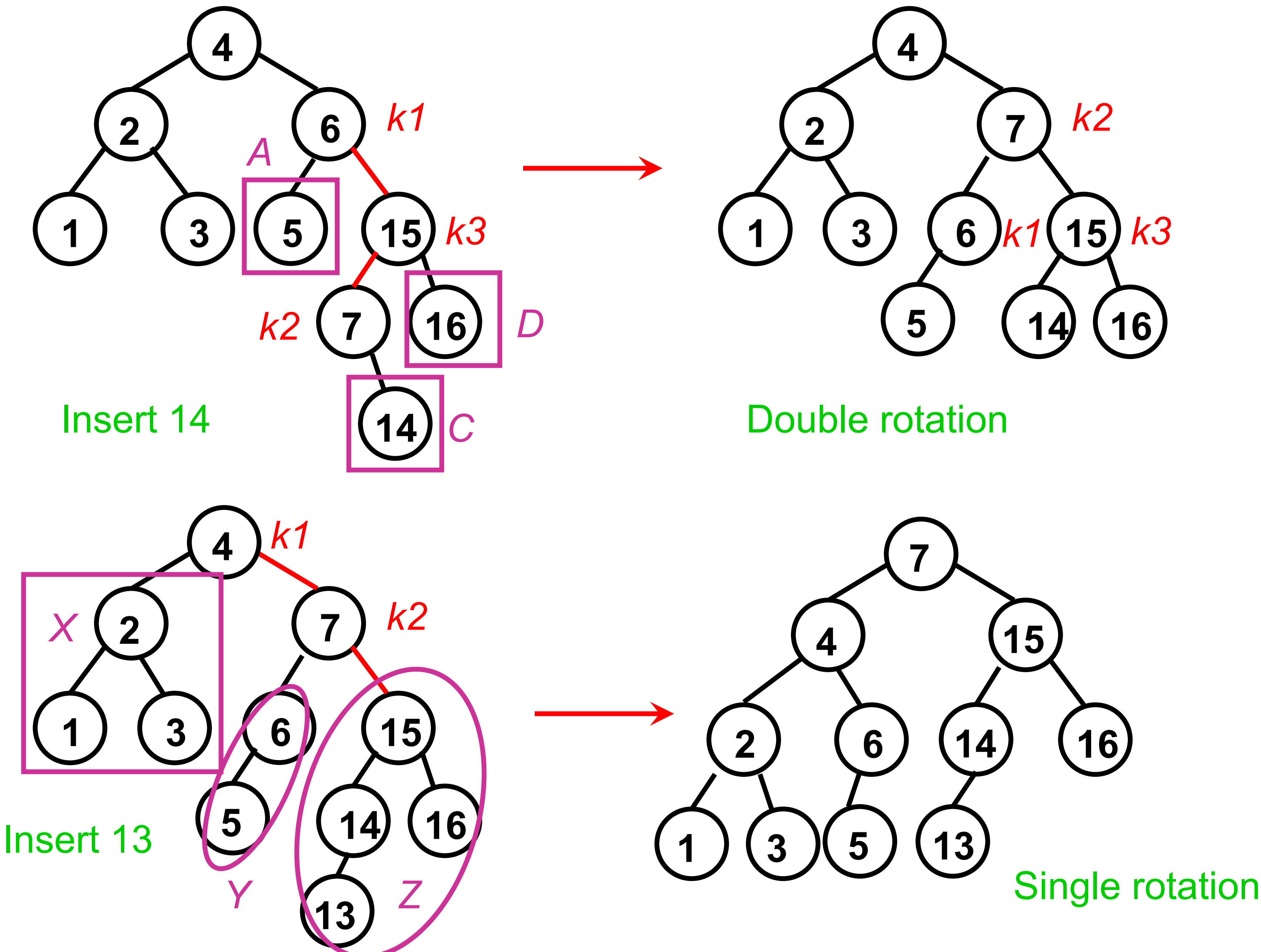
Example

- Restart our example

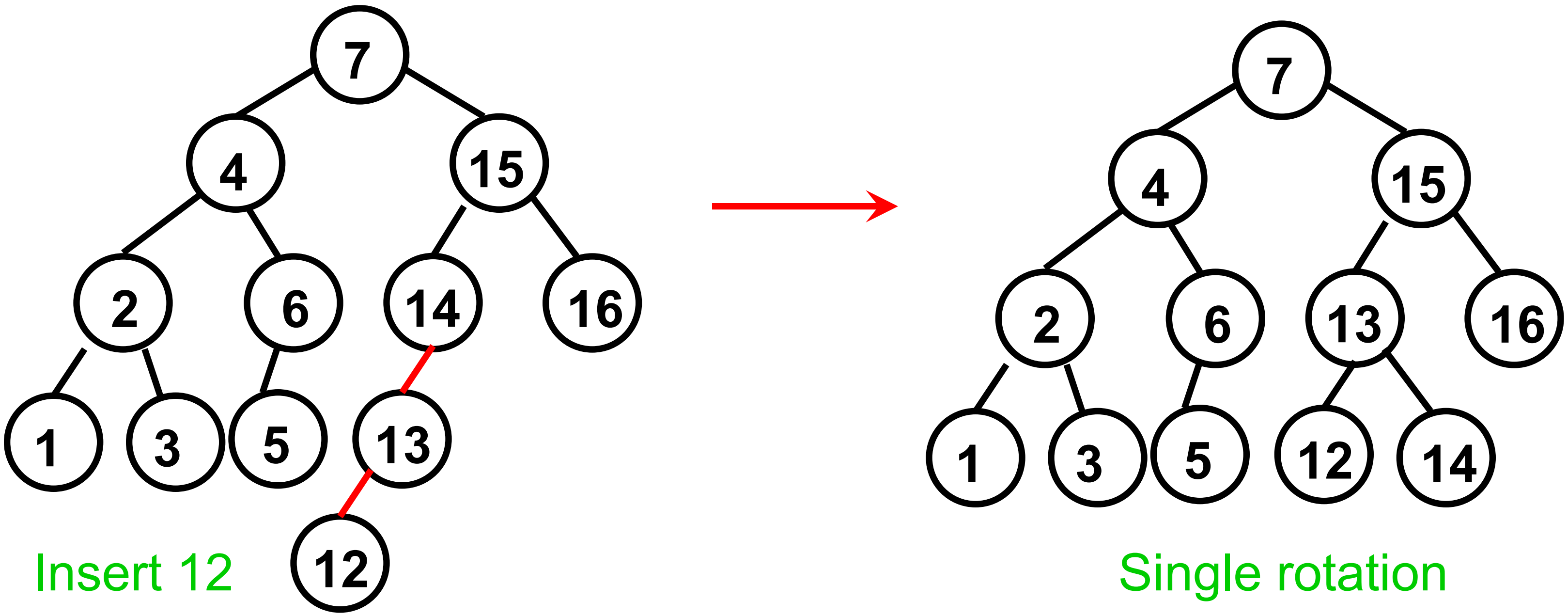
  We've inserted 3, 2, 1, 4, 5, 6, 7, 16

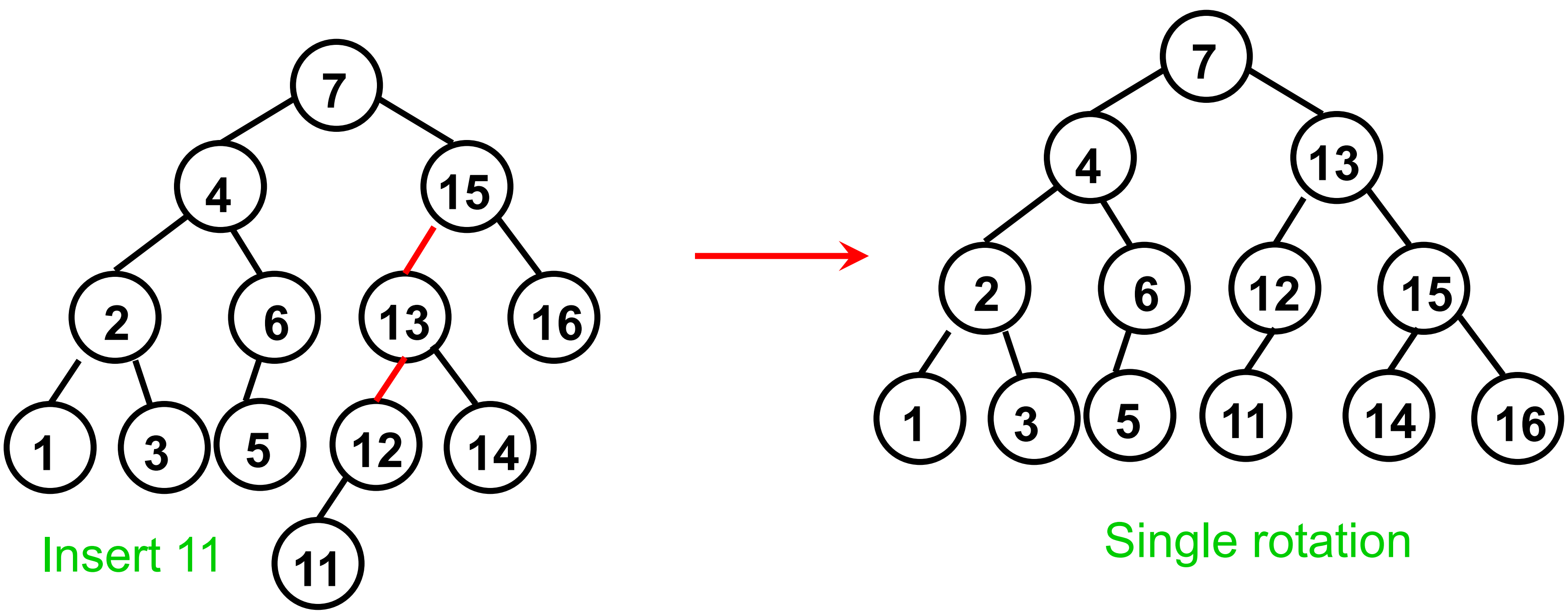
  We'll insert 15, 14, 13, 12, 11, 10, 8, 9

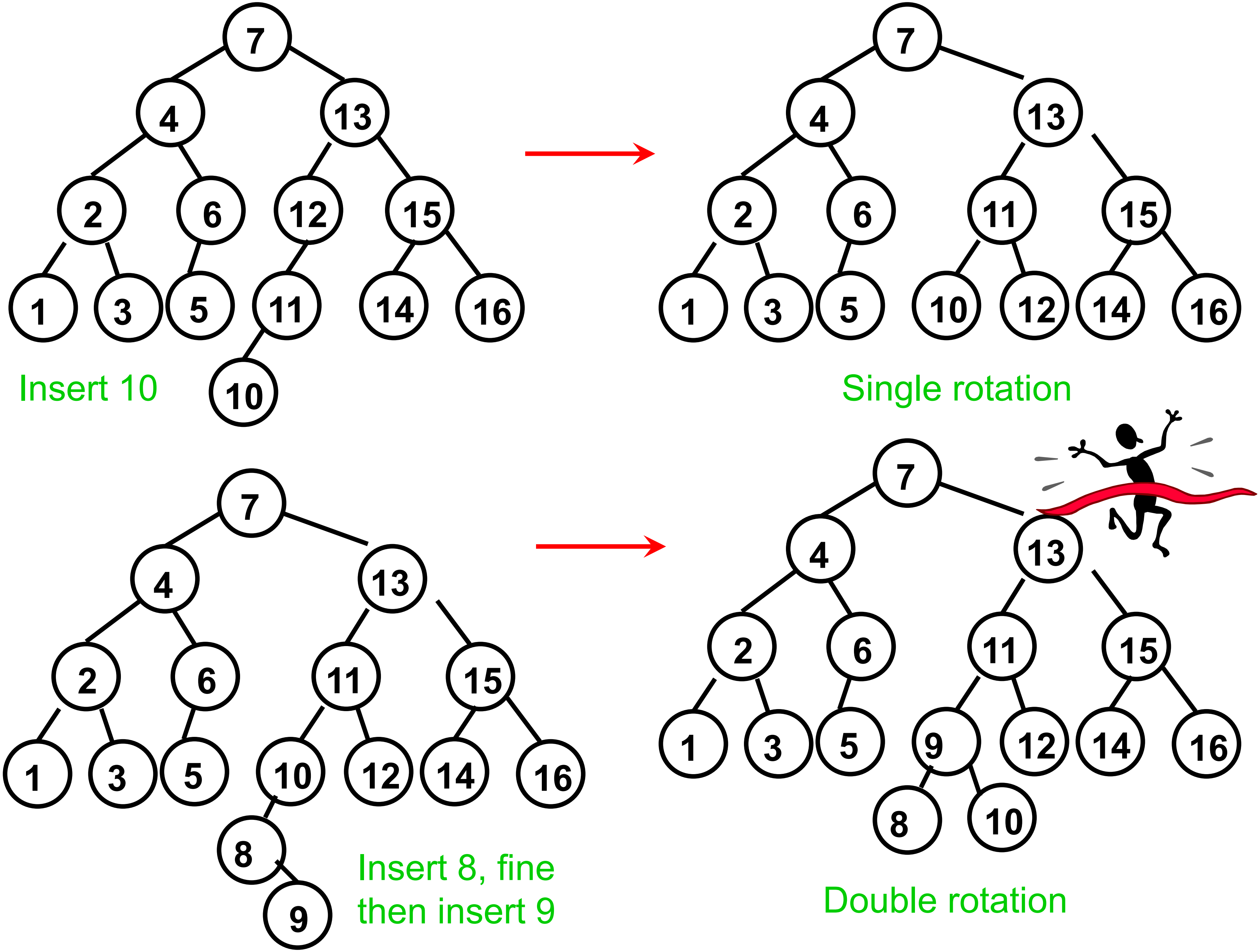




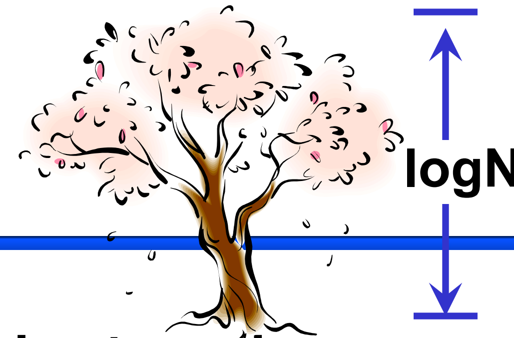
4
2
6
k1
A
1
3
5
15
k3
k2
7
16
D
14
C
Insert 14
4
2
7
k2
1
3
6
k1
15
k3
5
14
16
Double rotation
4
k1
X
2
7
k2
1
3
6
15
5
14
16
Insert 13
Y
13
Z
7
4
15
2
6
14
16
1
3
5
13
Single rotation


Insert 12
Single rotation


Insert 11
Single rotation


Insert 10
Single rotation
Insert 8, fine
then insert 9
Double rotation

Insertion Analysis



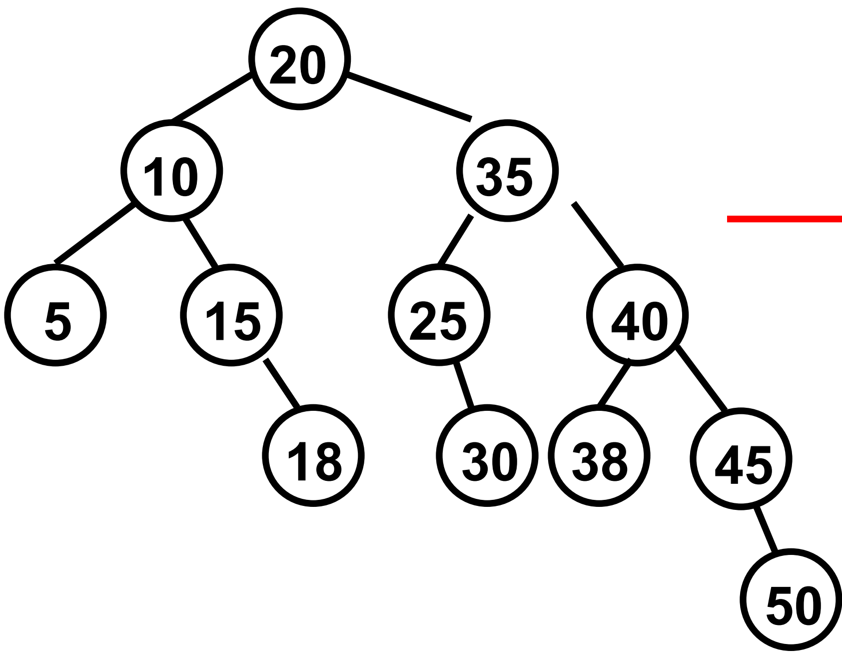
- Insert the new key as a new leaf just as in ordinary binary search tree: O(logN)
- Then trace the path from the new leaf towards the root, for each node x encountered: O(logN)
  - Check height difference: O(1)
  - If satisfies AVL property, proceed to next node: O(1)
  - If not, perform a rotation: O(1)
- The insertion stops when
  - A rotation is performed
  - Or, we've checked all nodes in the path
- Time complexity for insertion O(logN)

Deletion from AVL Tree

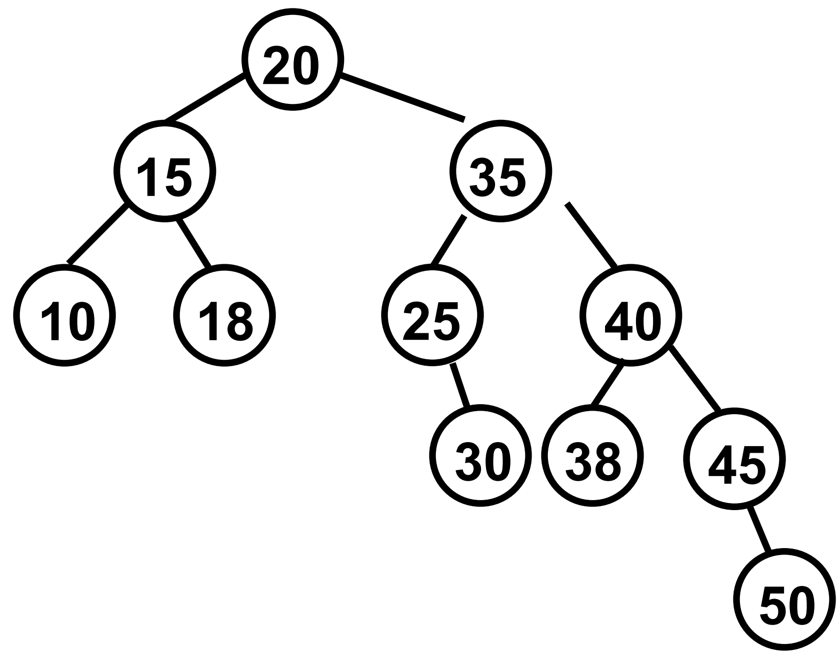
- Delete a node x as in ordinary binary search tree
  - Note that the last (deepest) node in a tree deleted is a leaf or a node with one child
- Then trace the path from the new leaf towards the root
- For each node x encountered, check if heights of left(x) and right(x) differ by at most 1.
  - If no, perform an appropriate rotation at x
  - If yes, proceed to parent(x)

  Continue to trace the path until we reach the root

Deletion Example 1

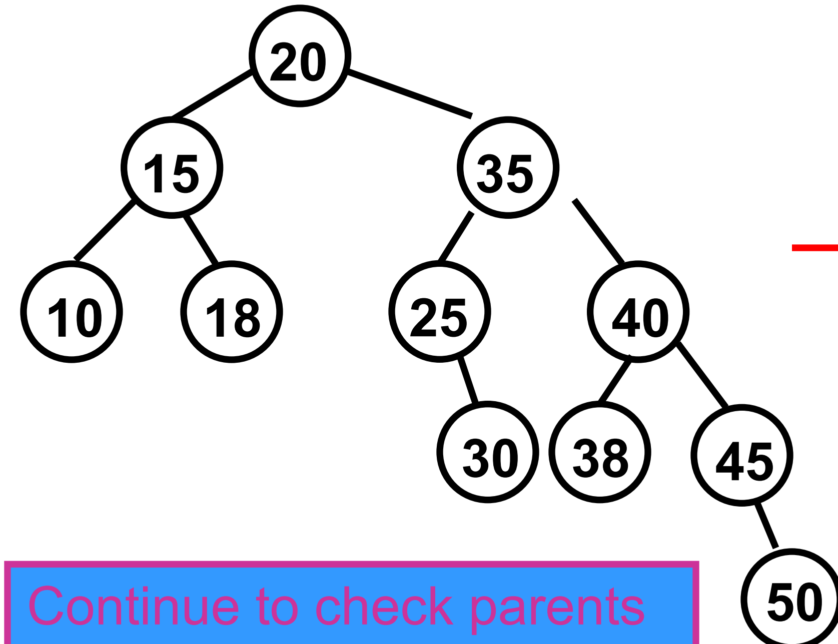


Delete 5, Node 10 is unbalanced



Single Rotation

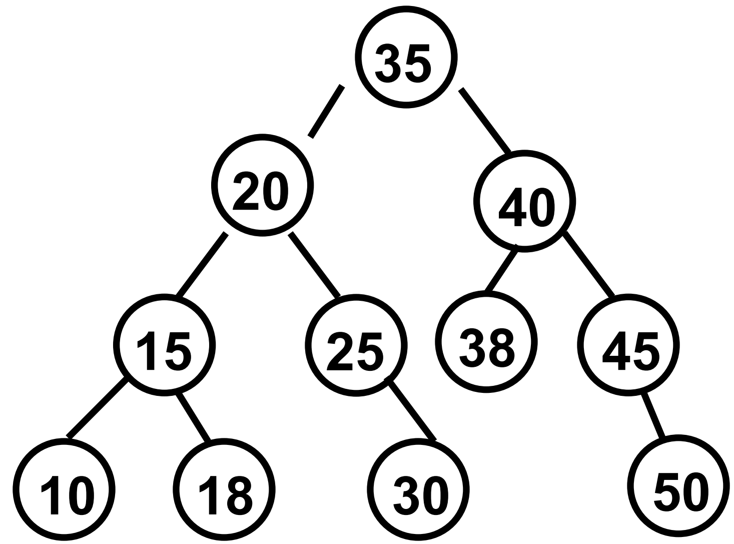
Cont'd



Continue to check parents

Oops!!

Node 20 is unbalanced!!

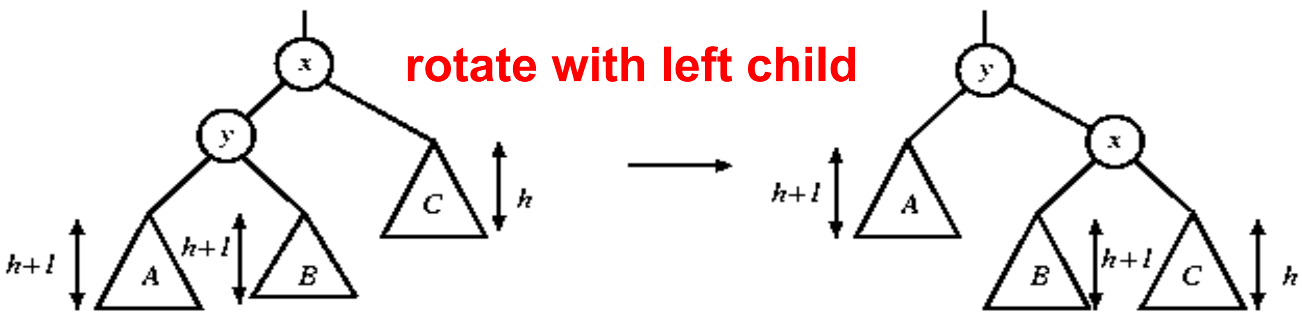


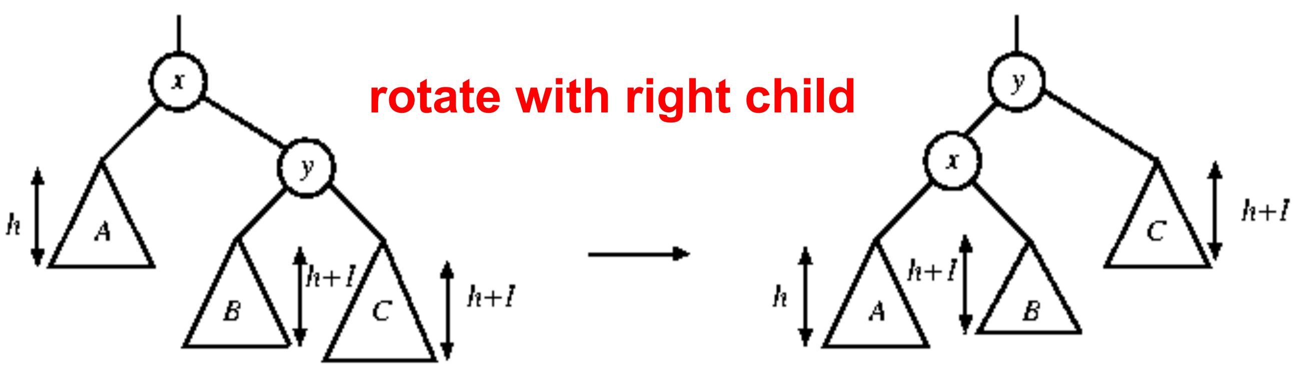
Single Rotation

For deletion, after rotation, we need to continue tracing upward to see if AVL-tree property is violated at other node.

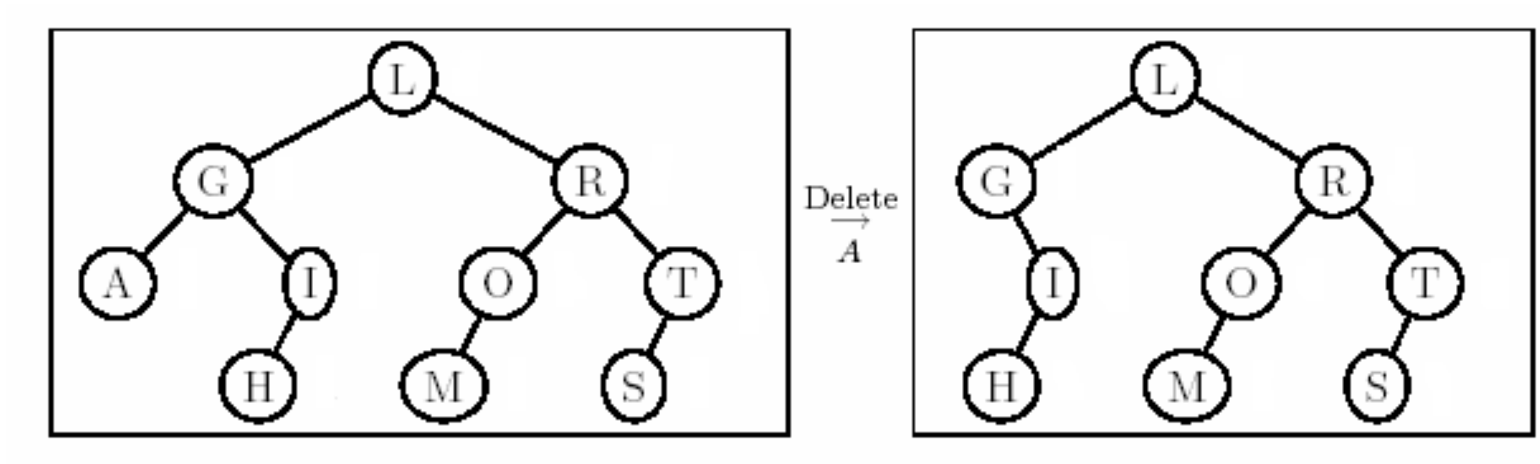
Rotation in Deletion

- The rotation strategies (single or double) we learned can be reused here
- Except for one new case: two subtrees of y are of the same height ➔ in that case, a single rotation is ok

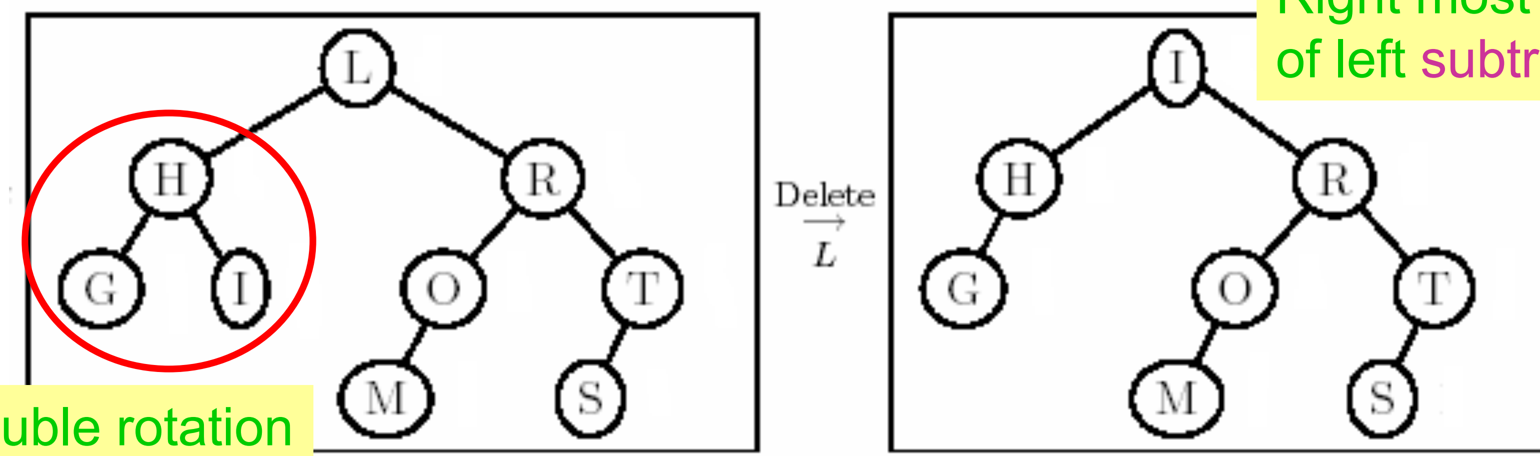




Deletion Example 2



Right most child of left subtree = I



Double rotation

Ok here!

Example 2 Cont'd