AVL Tree

- Searching
- Finding min/max
- Insertion
- Deletion

الدكتور
اثير العاني

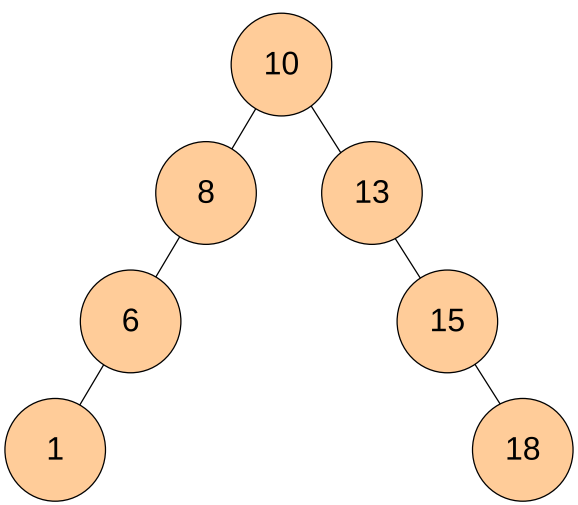
AVL Tree

- AVL Tree is a binary search tree that satisfies
  - For each node, the height of the left and right subtrees can differ by at most 1
  - Recall that the height of a node is defined as the length of the longest path from that node to a leaf node
    - Define the height of an empty tree to be -1 for convenience
    - Usually, the height of every node are stored in the implementation
- A tree satisfying this property can be proven height = O(log n), since it is an almost balanced tree

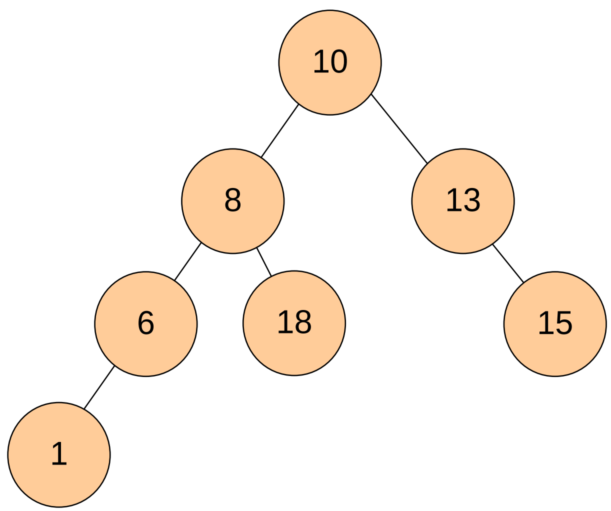
  ➔ Fast operations (search, insert, delete) can be supported

AVL Tree Example

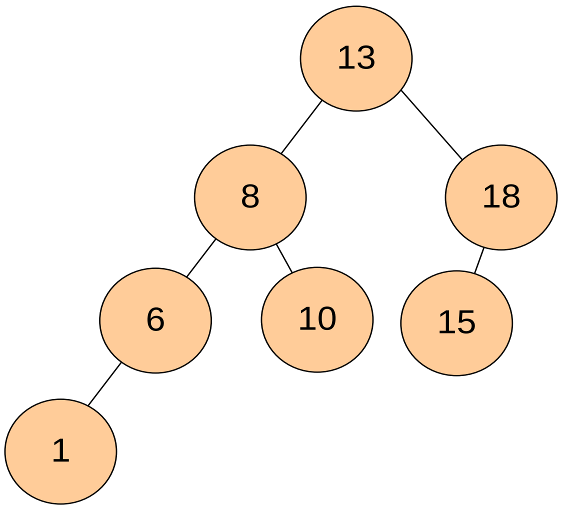
- Which of the following is an AVL tree?



no



no
(this is not a BST)



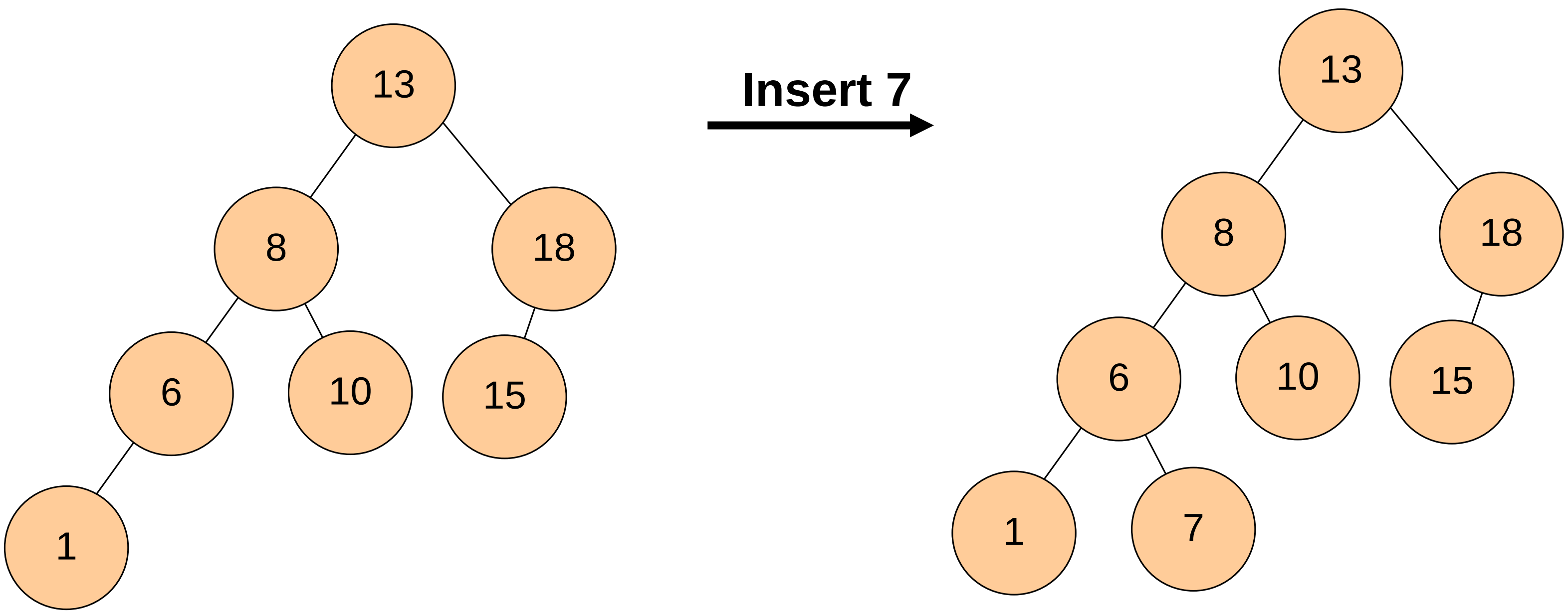
yes

AVL Tree - Operations

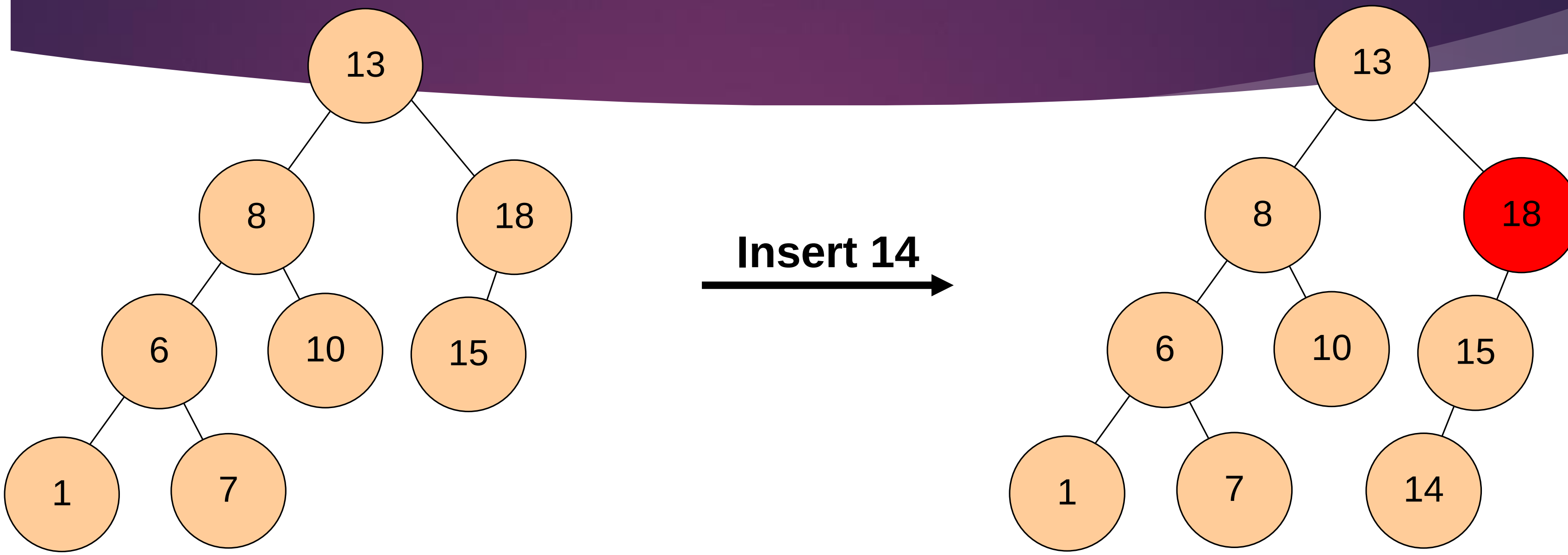
- Operations we will consider in this tutorial
  - Searching
  - Finding minimum/maximum
  - Insertion
  - Deletion
- Searching and finding min/max are the same as BST
  - But now it is guaranteed to finish in O(log n) time

Insertion in AVL Tree

- Basically, insertion can be done as in BST
- Eg. Insert 7 and 14 into the following tree



Insertion in AVL Tree



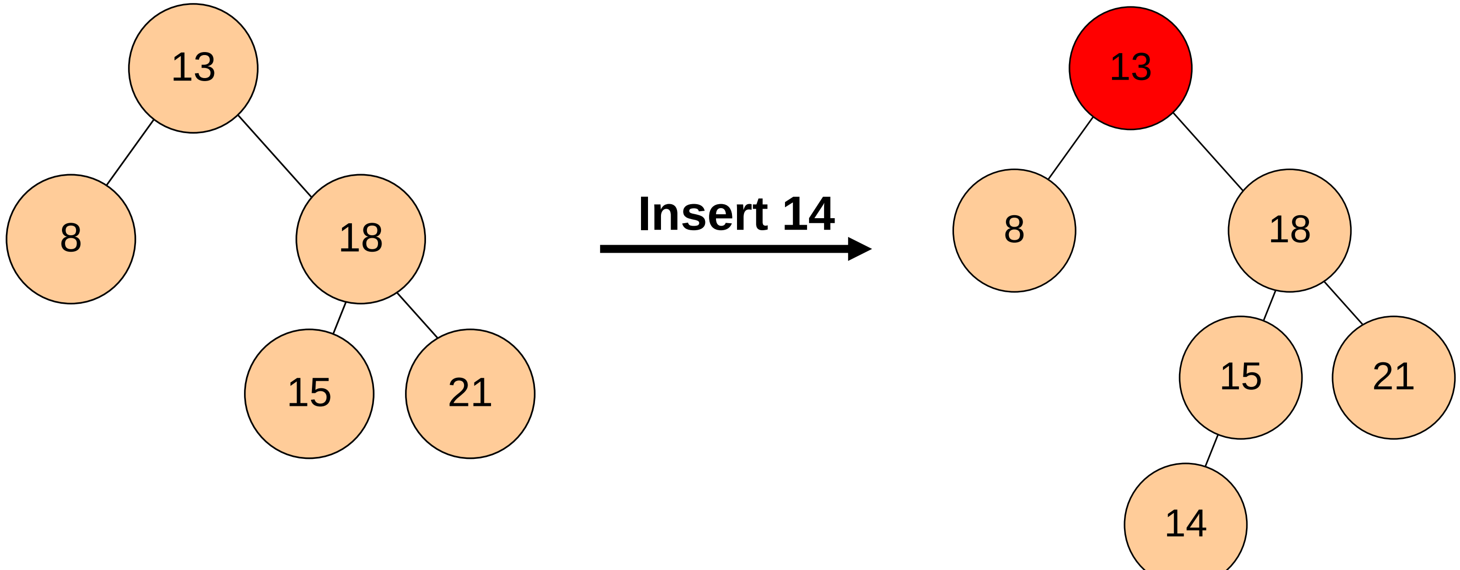
- AVL Property may be violated after insertion
  - We need to restore the AVL property

Which nodes may violate AVL property?

- Before talking about how to maintain the AVL property, how can we tell whether the AVL property is violated after insertion?
  - Checking the height of subtrees in every node
    - Too slow (O(n) time to check all nodes)
    - Note that all non-ancestors nodes remain OK
      - → AVL property on these nodes must still be satisfied
      - → No need to check them
  - Checking the direct parent of the inserted node?
    - No, the direct parent of the inserted node should not violate AVL property.

Which nodes may violate AVL property?

- Checking the grandparent of the inserted node?
  - Yes
- In general, checking all ancestors of the inserted node
  - After insertion, the height of subtrees of ancestors are updated

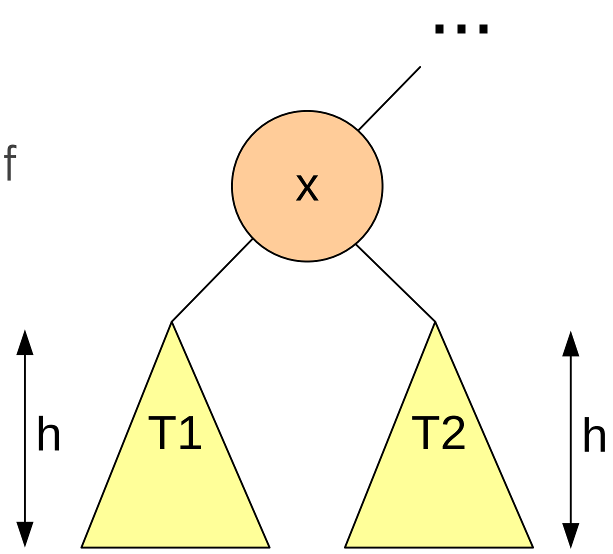


Which nodes may violate AVL property?

- Therefore, we need to check whether the AVL property holds for all ancestors of the inserted node (up to the root)
  - Indeed, you may skip the checking of its parent and the inserted node itself
- If there is no ancestor violate AVL property
  - We are done
- Otherwise, we need to restore the AVL property at that node

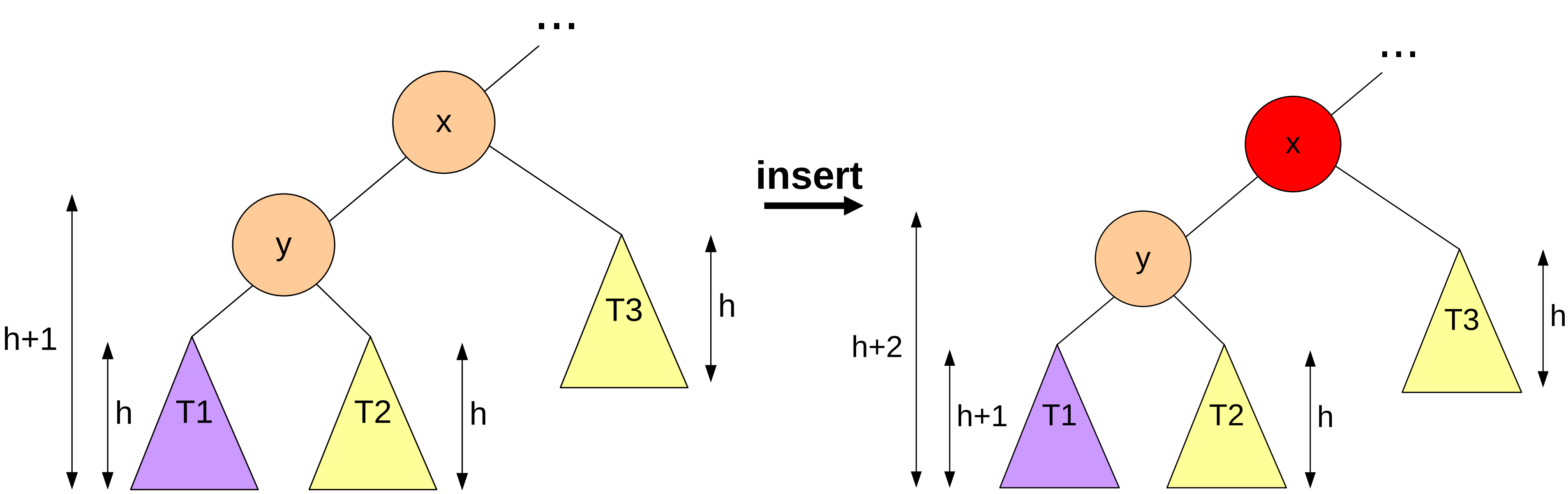
Restore AVL property (Case 0)

- A node is inserted in T1 or T2
  - AVL property cannot be violated at node x
    - The height of T1 and T2 can be increased by at most 1 after insertion
- AVL property can only be violated if the height of T1 and T2 differ by 1 originally, and an insertion is occurred at the taller subtree to make it even taller



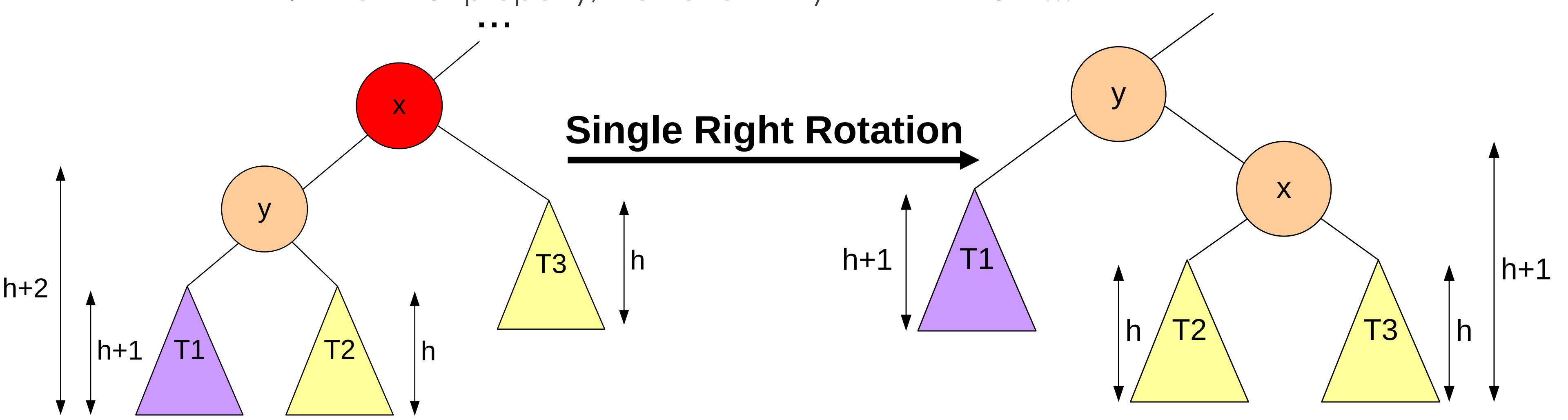
Restore AVL property (Case 1)

- A node is inserted in T1 and AVL property is violated in node x
  - And we assume node x is the lowest node that violate the AVL property



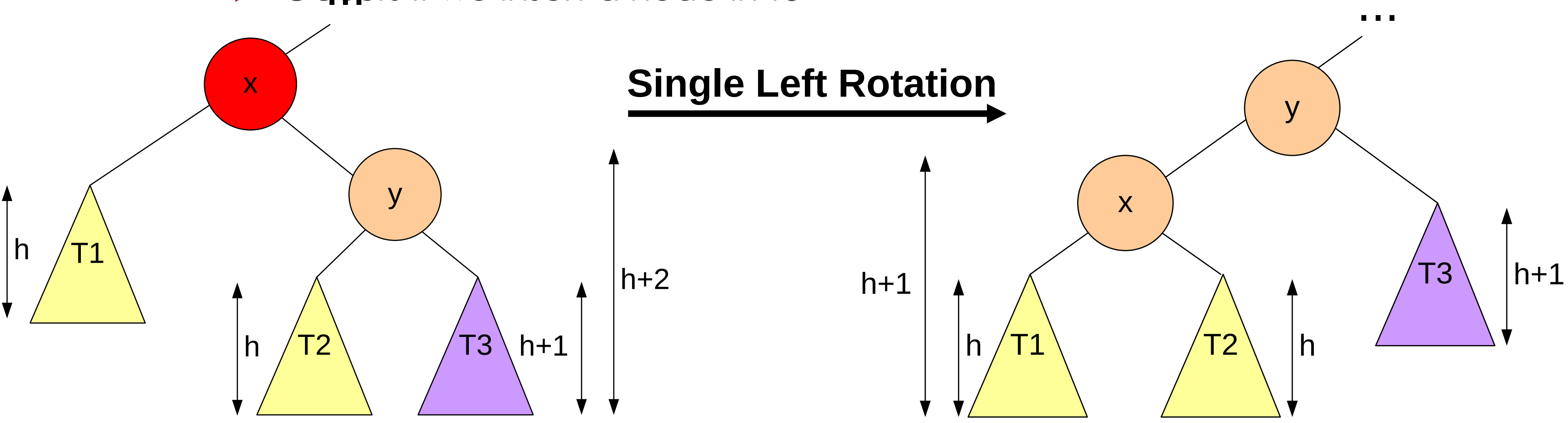
Restore AVL property (Case 1)

- Swap node x and node y
  - After swapping, the subtrees T1, T2 and T3 must be in their final position
    - From BST property, we have T1 < y < T2 < x < T3 < …



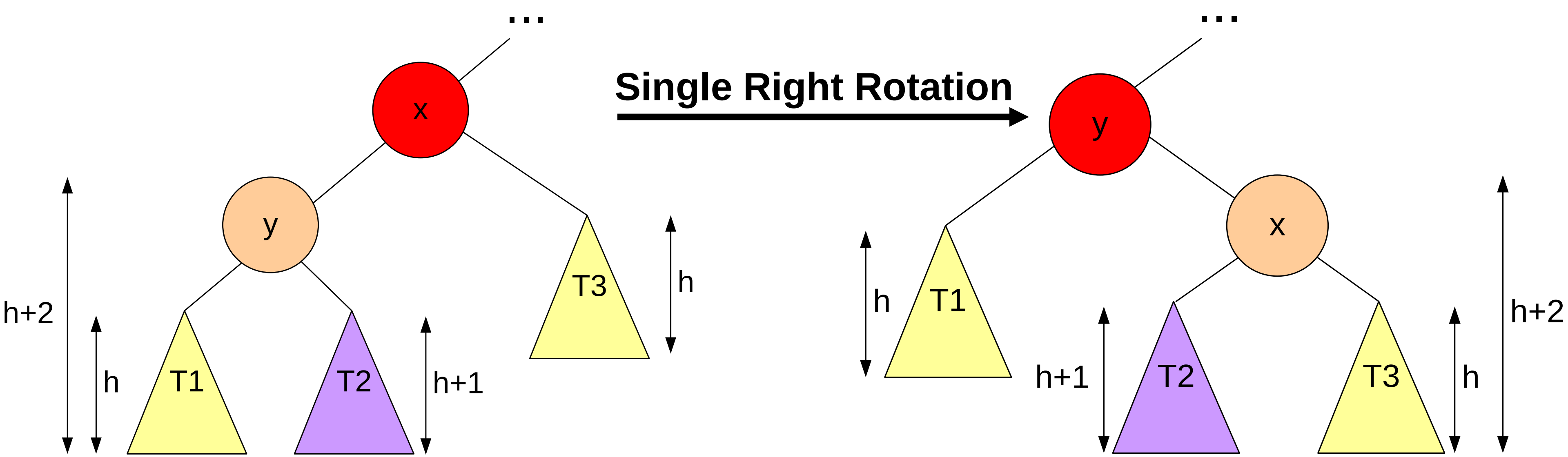
Restore AVL property (Case 4)

- Symmetric case as case 1
  - Occurs if we insert a node in T3



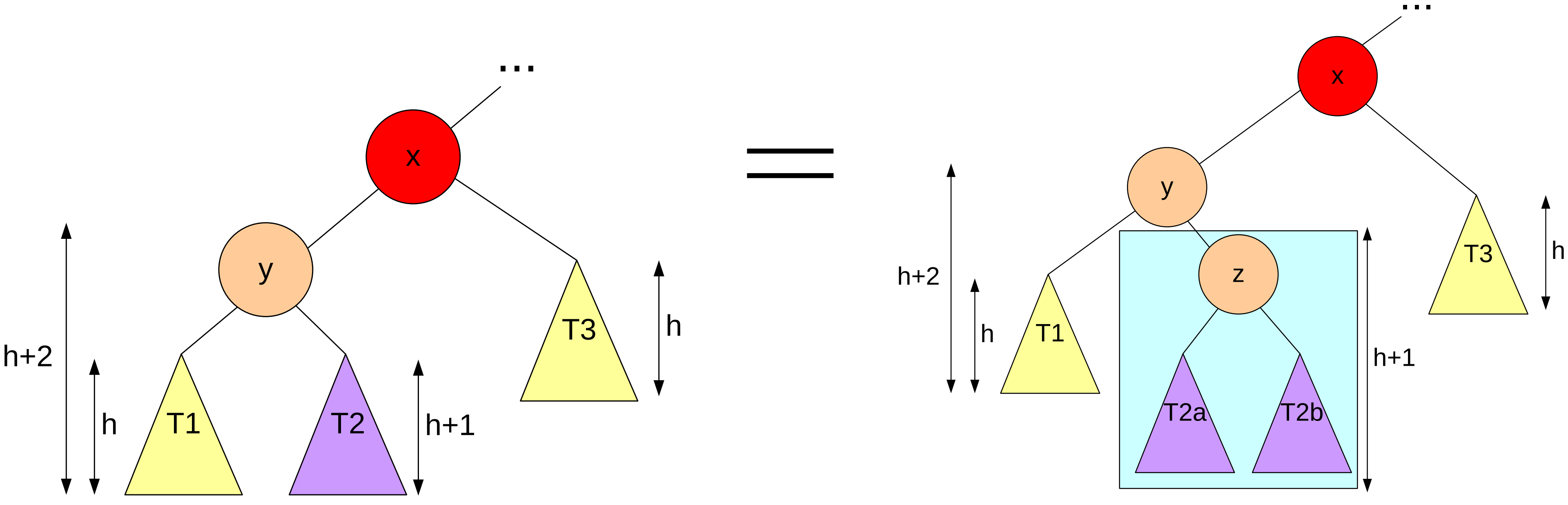
Restore AVL property (Case 2)

- Single rotation may fail if a node is inserted in T2
  - In this case, we need double rotation



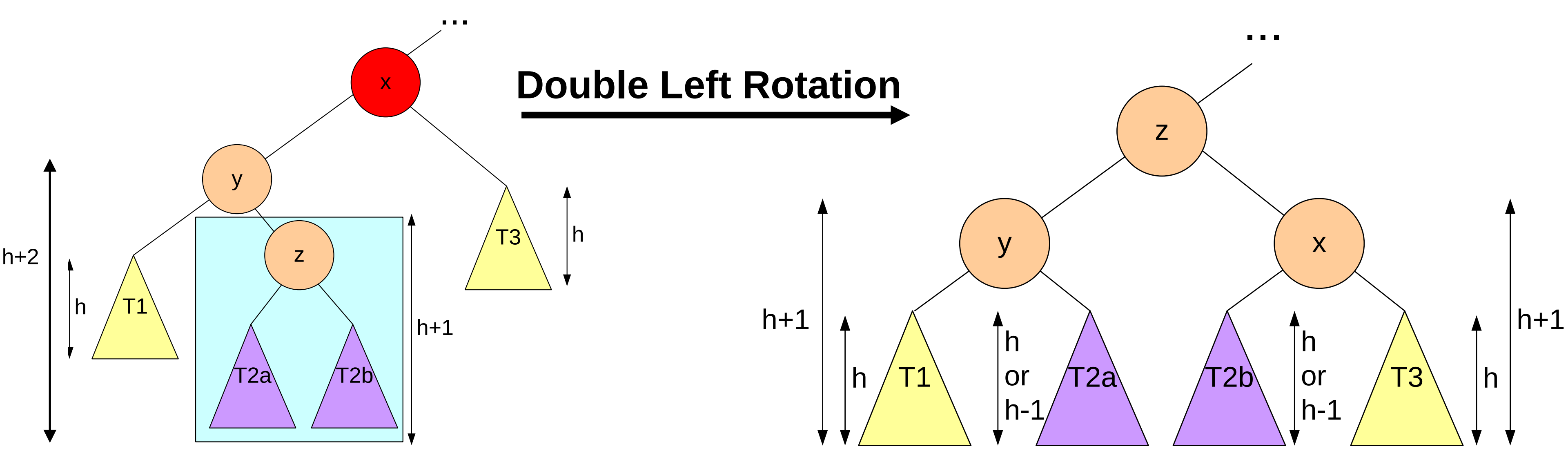
Restore AVL property (Case 2)

- Further examine the subtree T2
  - A node is inserted either in T2a or T2b



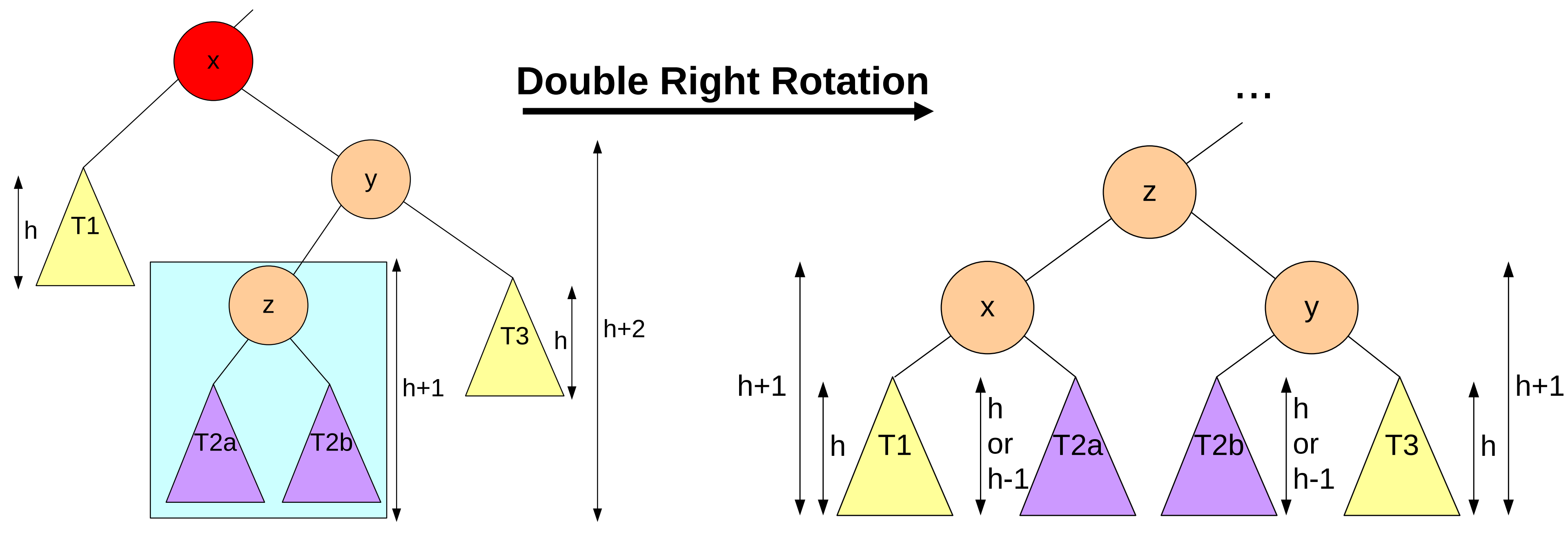
Restore AVL property (Case 2)

- Relocate node x, y and z
  - From BST property, we have
    T1 < y < T2a < z < T2b < x < T3



Restore AVL property (Case 3)

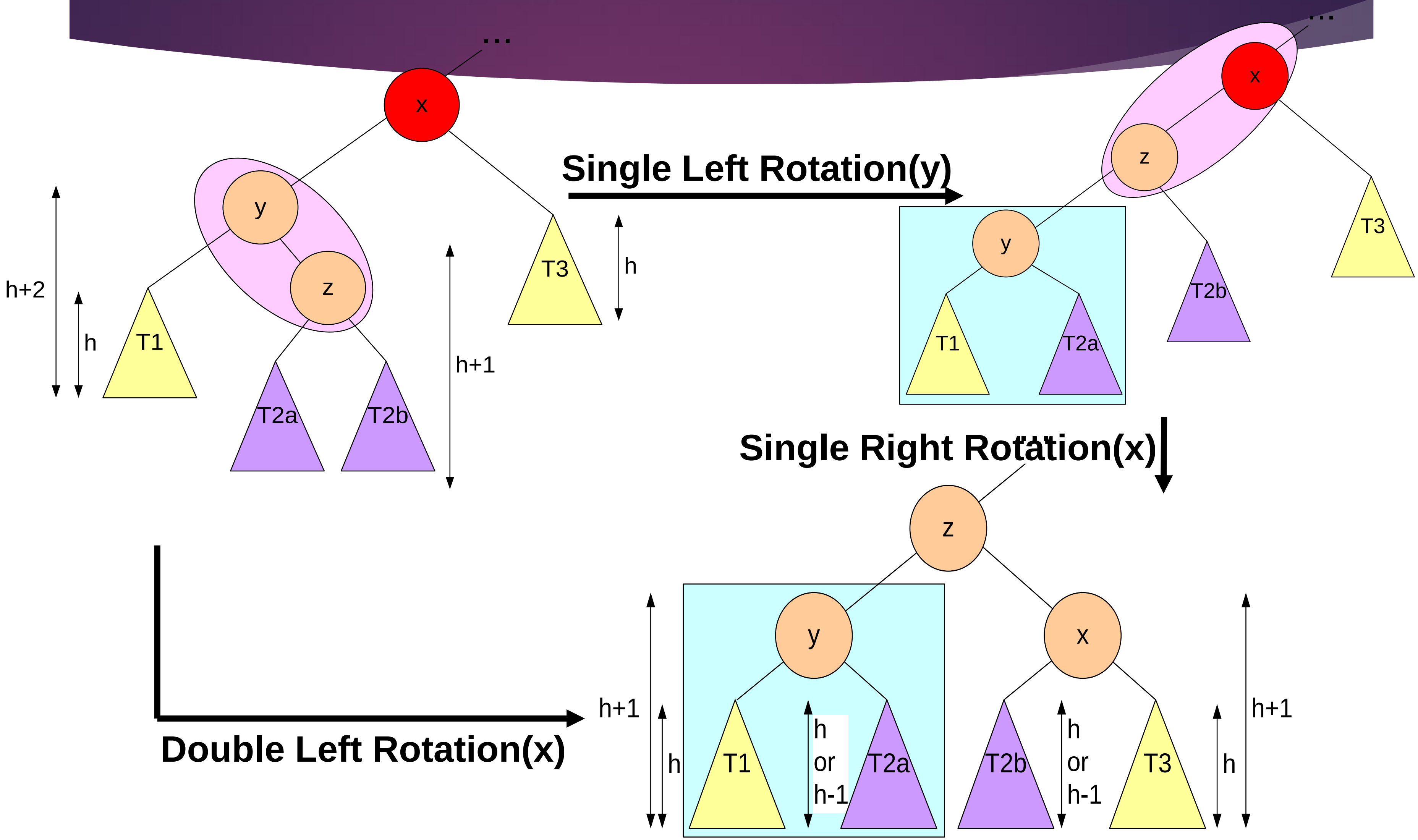
- Symmetric case as case 2
  - …Occurs if we insert a node in T2a or T2b



Why it is called "Double" rotation?

- Double Left Rotation is equivalent to two single rotations
  - Single Left Rotation at y
  - Then, Single Right Rotation at x
- Similarly, Double Right Rotation is equivalent to
  - Single Right Rotation at y
  - Then, Single Left Rotation at x

Why it is called “Double” rotation?



Insertion in AVL Tree – Time Complexity

- Time complexity
  - Find the location to insert
    - O(log n)
  - Checking and rotation (if needed) for a node
    - O(1)
  - Checking and rotation for all ancestors of the inserted node
    - O(log n)
- Overall, time complexity of insertion is O(log n)

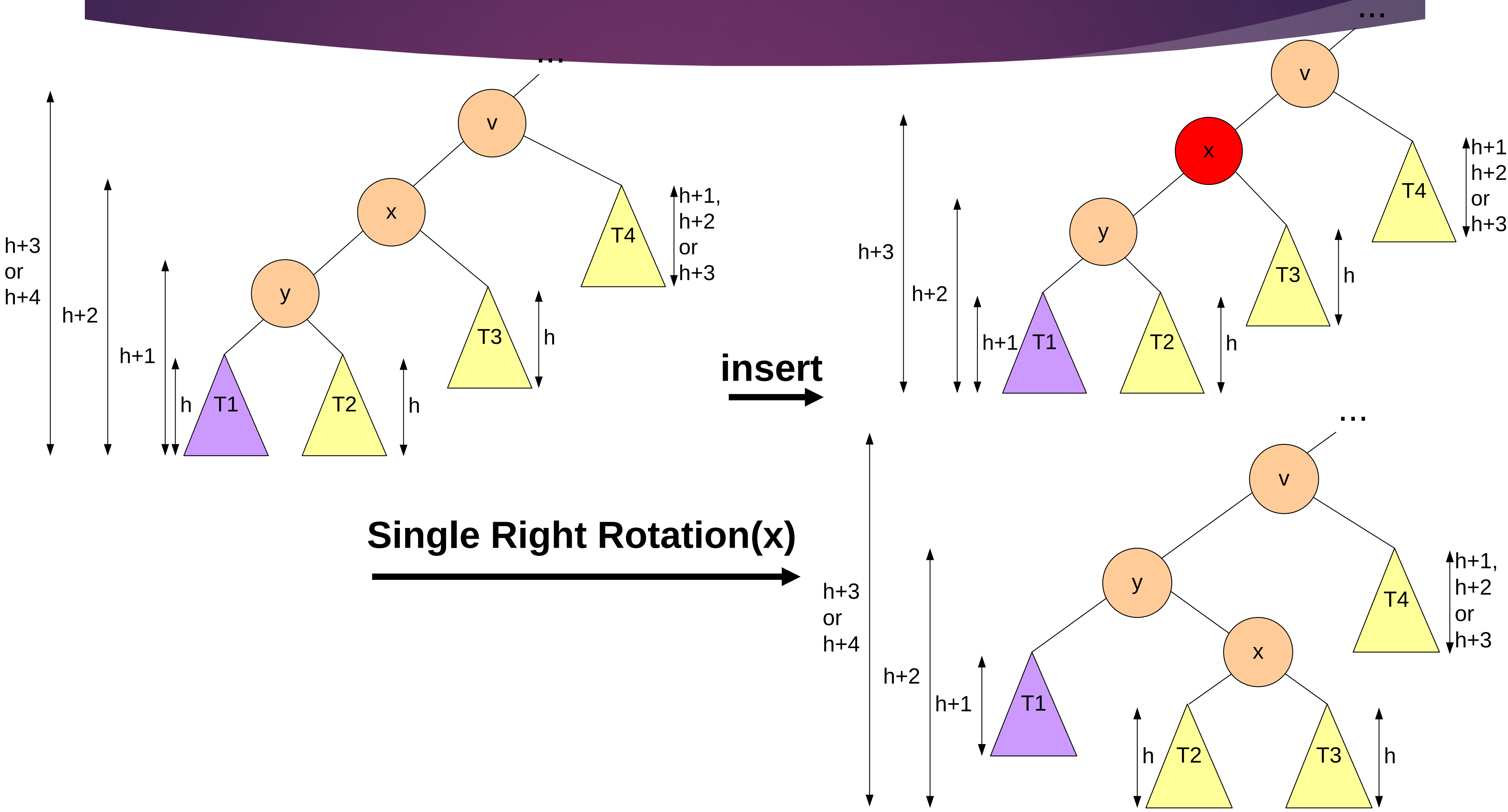
Continue to examine its ancestors?

- After a rotation, it is easy to see that the AVL property is fixed at the violated node x
- Recall that the AVL property can only be violated at ancestors of the inserted node
  - Then, shall we continue to examine the ancestors of x?
  - No

Continue to examine its ancestors?

- We can study all 4 cases, but let's consider case 1 here
  - If node x is the root, clearly no further examination is needed
  - Otherwise, we can assume node x has a parent (say, node v)

Continue to examine its ancestors?



Continue to examine its ancestors?

- In node v
  - No violation of AVL property in node v after rotation AND
  - Height of node v before and after insertion (with rotation) remains constant (either h+3 or h+4)

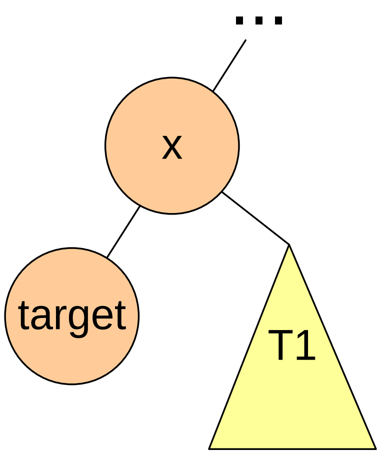
  ➔ Therefore the AVL property will not be violated in the parent (or any ancestor) of node v
- Therefore, no further examination is needed

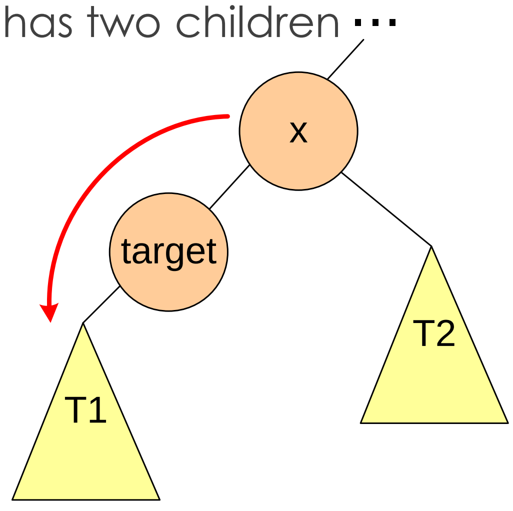
Summary: Insertion in AVL tree

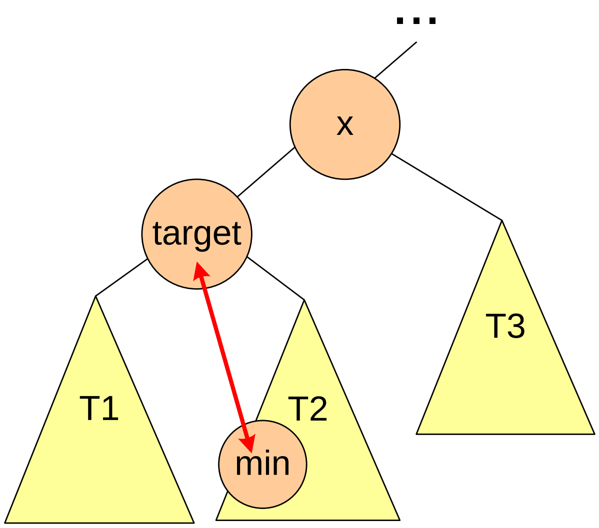
- Perform BST insertion $\mathrm{h}(x) = \max\{\mathrm{h}(x.\mathrm{left}), \mathrm{h}(x.\mathrm{right})\} + 1$
- For each ancestor x of the inserted node,
  - Update its height by
  - If AVL property is violated at node x
    - If h(x.left) = h(x)-1 (i.e. h(x.right) = h(x)-3)
      - If h(x.left.left) = h(x)-2 ➔ Single Right Rotation(x) [case 1]
      - Else If h(x.left.right) = h(x)-2 ➔ Double Left Rotation(x) [case 2]
    - If h(right(x)) = h(x)-1
      - If h(x.right.left) = h(x)-2 ➔ Double Right Rotation(x) [case 3]
      - Else If h(x.right.right) = h(x)-2 ➔ Single Left Rotation(x) [case 4]
    - Finish // early termination
- Caution: don't forget
  - Rotation may change x, so remember to connect the resulting tree to x.parent
  - Update the height of nodes involved in rotations

Review: Deletion in BST

- There are 3 cases
  - Deleted node is a leaf
  - Deleted node has one child
  - Deleted node has two children







Deletion in AVL Tree

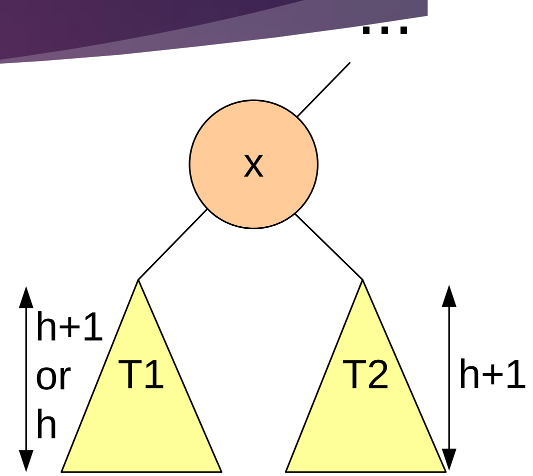
- Perform BST deletion
- Similar to insertion, we have to restore the AVL property after deletion

Which nodes may violate AVL property?

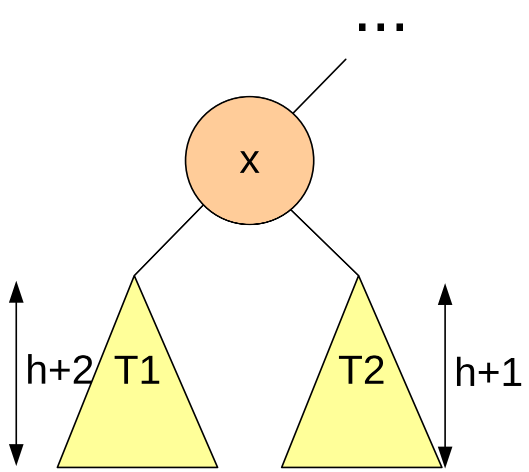
- Recall the three cases
  - Deleted node is a leaf
    - Height of x may be modified
  - Deleted node has one child
    - Height of x may be modified
  - Deleted node has two children
    - Height of the parent of the minimum node of right subtree of x may be modified
- And the height of all ancestors of the altered node may need to be update
  - As before, consider node x is the lowest node that violate the AVL property after deletion

Restore AVL property (Case 0)

- A node is deleted in T2
  - AVL property cannot be violated at node x
    - The height of T2 can be decreased by at most 1 after deletion

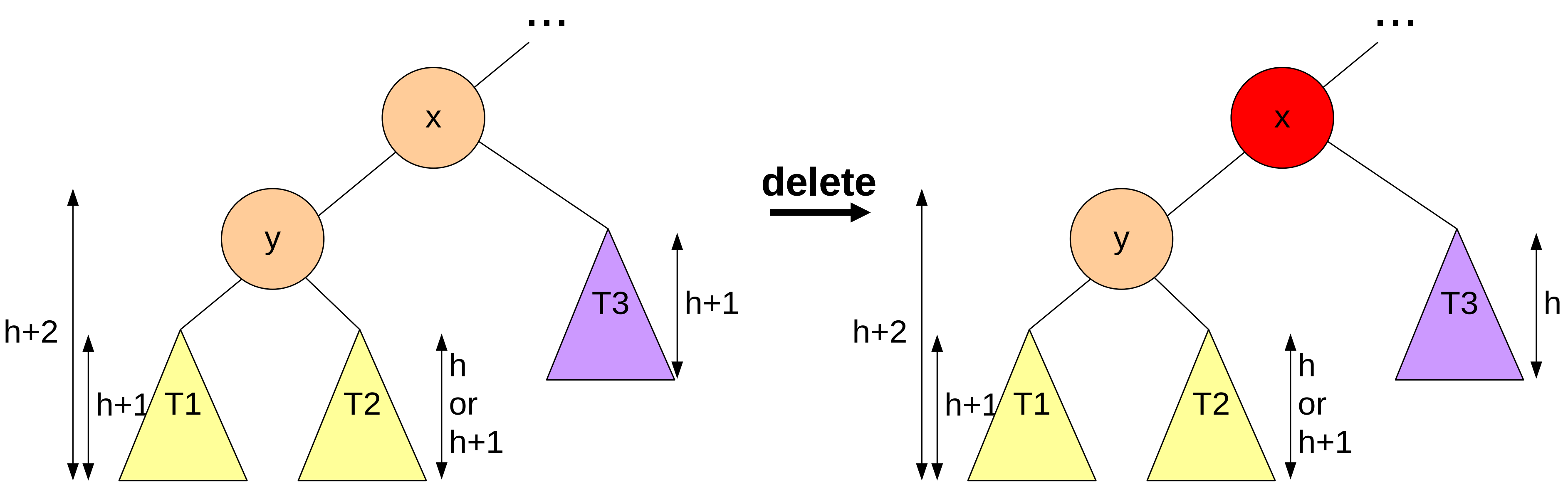


- AVL property can only be violated if the height of T2 is less than that of T1 by 1 originally, and a node is deleted from T2 to make T2 even shorter by 1
  - Symmetrically, similar case for T1



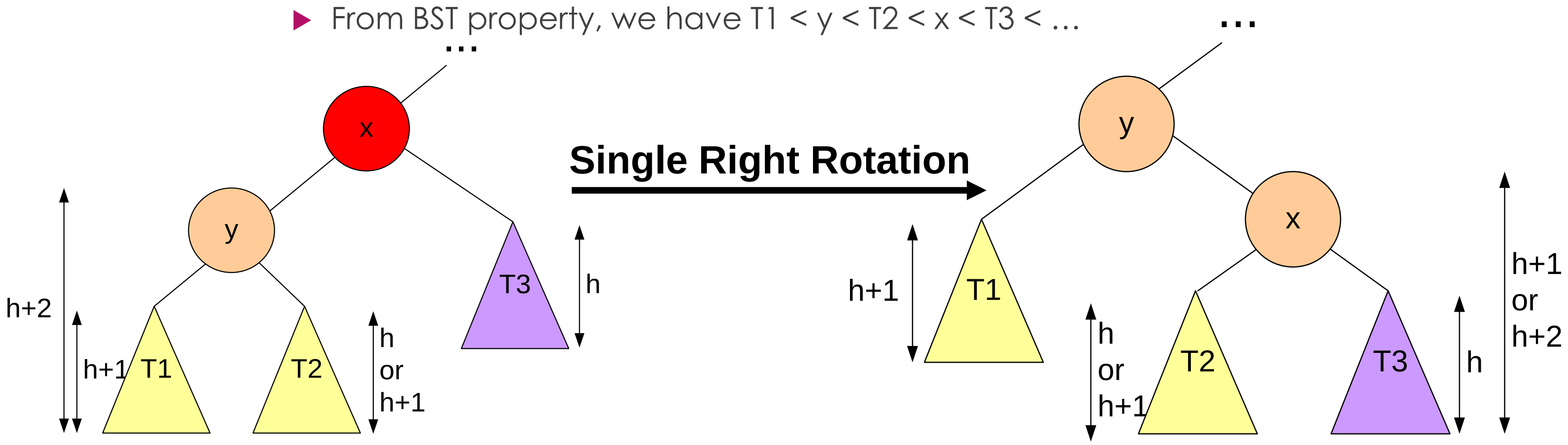
Restore AVL property (Case 1)

- A node is deleted from T3 and AVL property is violated in node x



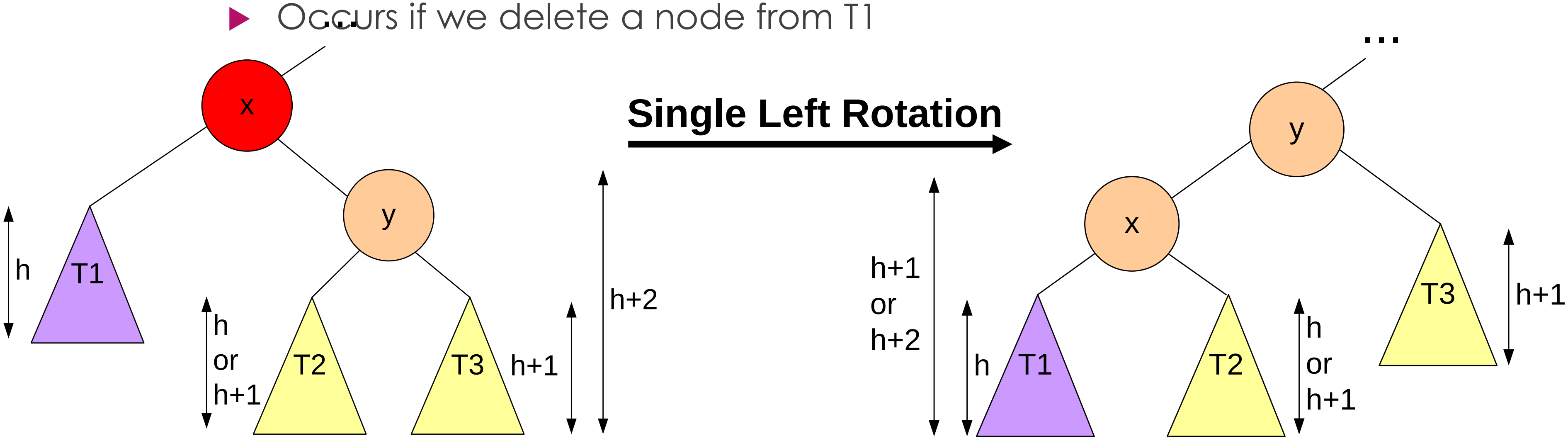
Restore AVL property (Case 1)

- Swap node x and node y
  - After swapping, the subtrees T1, T2 and T3 must be in their final position
    - From BST property, we have T1 < y < T2 < x < T3 < …



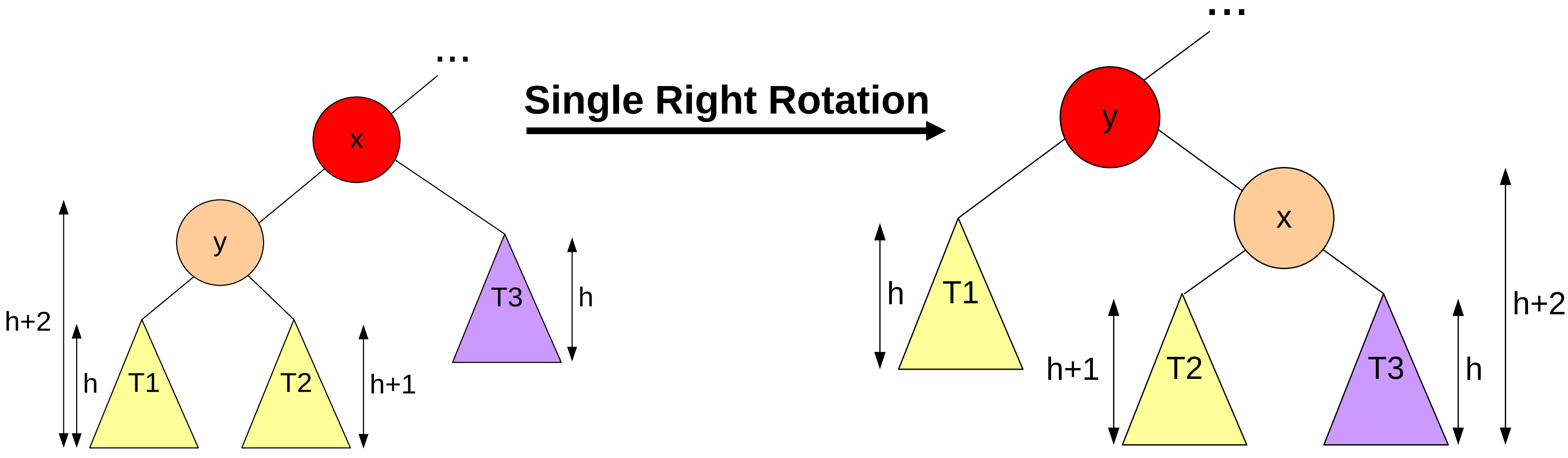
Restore AVL property (Case 4)

- Symmetric case as case 1
  - Occurs if we delete a node from T1



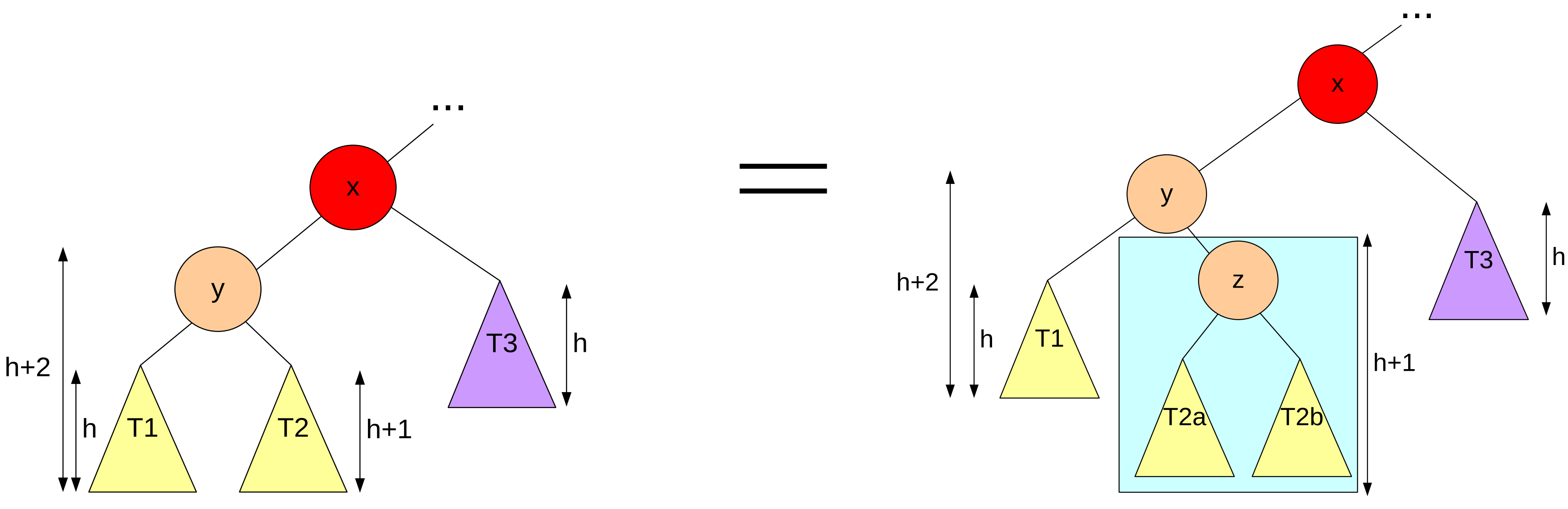
Restore AVL property (Case 2)

- Single rotation may also fail as in deletion
  - In this case, we need double rotation



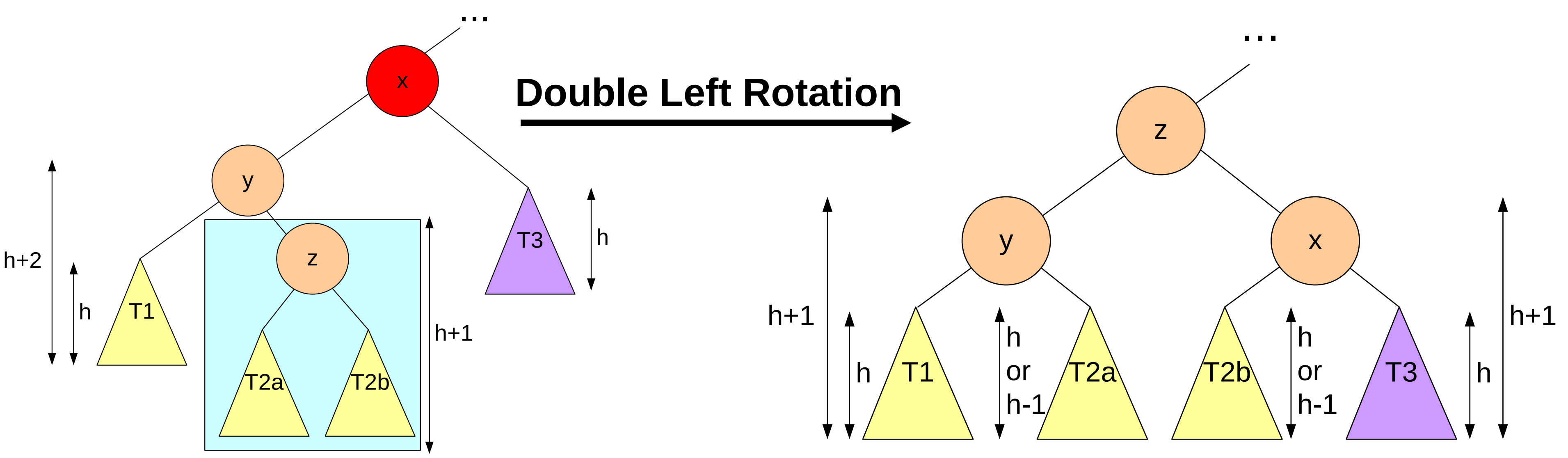
Restore AVL property (Case 2)

- Further examine the subtree T2
  - The height of T2a and T2b can either be h or h-1



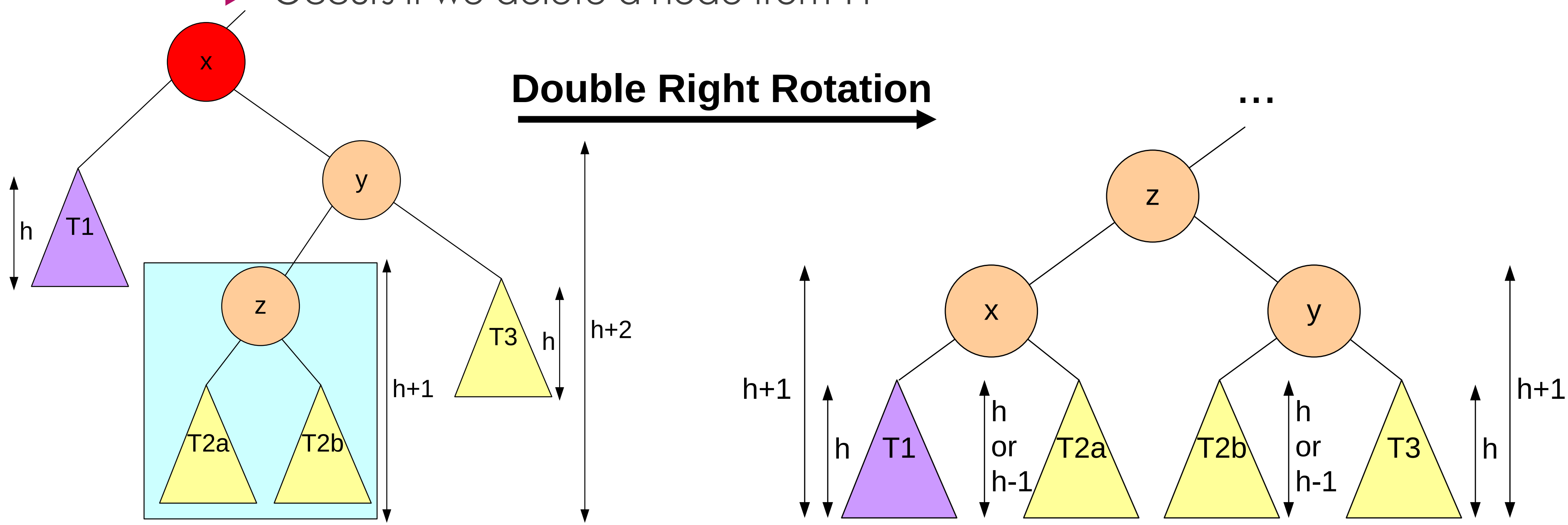
Restore AVL property (Case 2)

- Swap the order of x, y, z, so that two of them are children of the other one



Restore AVL property (Case 3)

- Symmetric case as case 2
  - Occurs if we delete a node from T1



Summary: Restore AVL property

- After deletion, you may need to check the AVL property for all ancestors of the last deleted node
- If AVL property violated, you may need to perform either single rotation or double rotation to fix it

Deletion in AVL Tree – Time Complexity

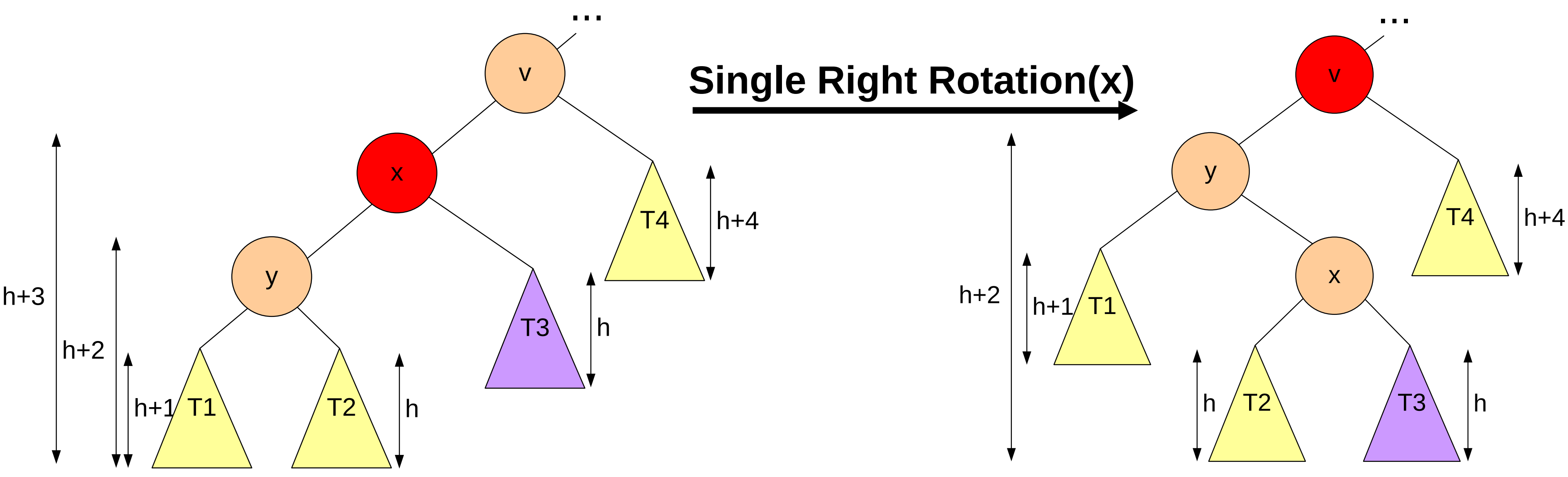
- Time complexity
  - Find the location to delete
    - O(log n)
  - Checking and rotation (if needed) for a node
    - O(1)
  - Checking and rotation for all ancestors of the last deleted node
    - O(log n)
- Overall, time complexity of deletion is O(log n)

Continue to examine its ancestors?

- After a rotation, it is easy to see that the AVL property is fixed at the violated node x
- Could we stop checking after one rotation, as in insertion?
  - No

Continue to examine its ancestors?

- Let's consider the following counter-example
  - A node is deleted from T3



Continue examine its ancestors?

- Indeed, all kinds of rotations also need further examination until we reach the root
- But, anyway, the time complexity of deletion is still O(log n)

Summary: Deletion in AVL tree

- Perform BST deletion $h(x) = \max\{h(x.\text{left}), h(x.\text{right})\} + 1$
- For each ancestor x of the last deleted node,
  - Update its height by
  - If AVL property is violated at node x
    - If h(x.left) = h(x)-1 (i.e. h(x.right) = h(x)-3)
      - If h(x.left.left) = h(x)-2 ➔ Single Right Rotation(x) [case 1]
      - Else If h(x.left.right) = h(x)-2 ➔ Double Left Rotation(x) [case 2]
    - If h(right(x)) = h(x)-1
      - If h(x.right.left) = h(x)-2 ➔ Double Right Rotation(x) [case 3]
      - Else If h(x.right.right) = h(x)-2 ➔ Single Left Rotation(x) [case 4]
    - // cannot early termination, until we reach the root
- Caution: don't forget
  - Rotation may change x, so remember to connect the resulting tree to x.parent
  - Update the height of nodes involved in rotations