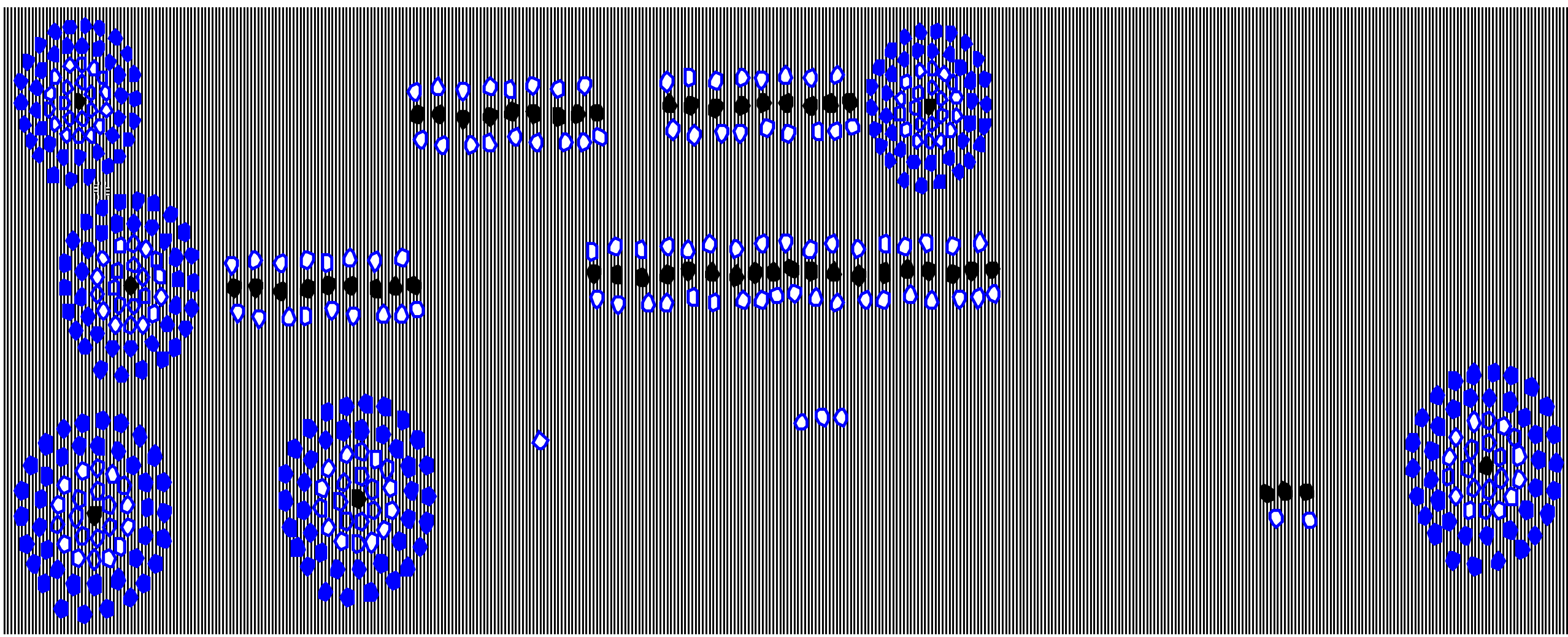
AVL Trees

- AVL Trees

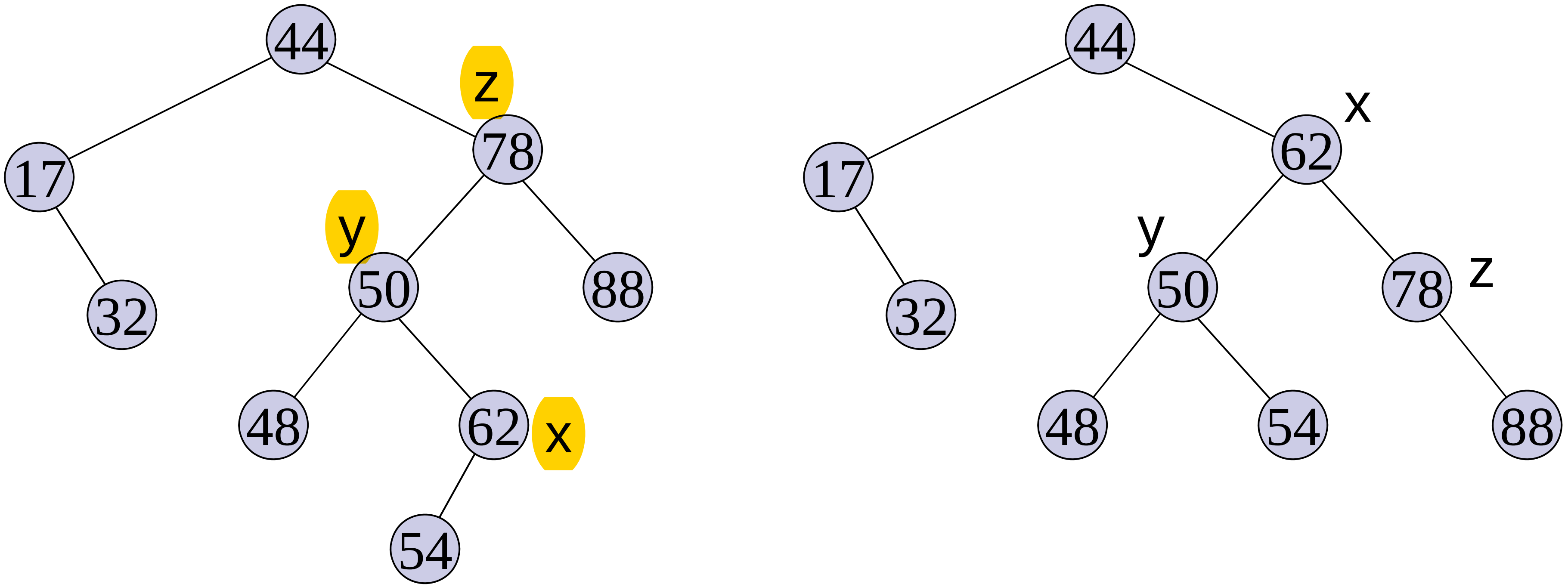


Insertion

- Inserting a node, v, into an AVL tree changes the heights of some of the nodes in T.
- The only nodes whose heights can increase are the ancestors of node v.
- If insertion causes T to become unbalanced, then some ancestor of v would have a height-imbalance.
- We travel up the tree from v until we find the first node x such that its grandparent z is unbalanced.
- Let y be the parent of node x.

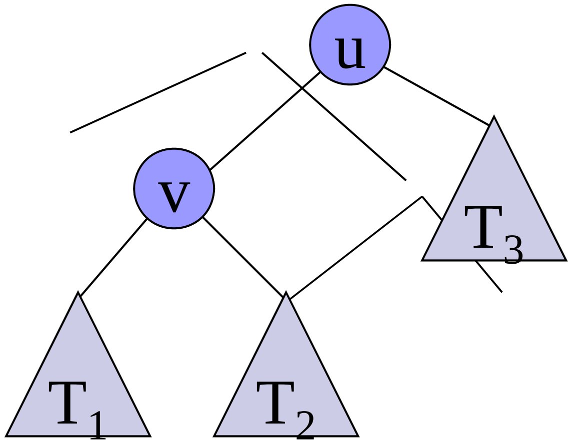
Insertion (2)

To rebalance the subtree rooted at z, we must perform a *rotation*.



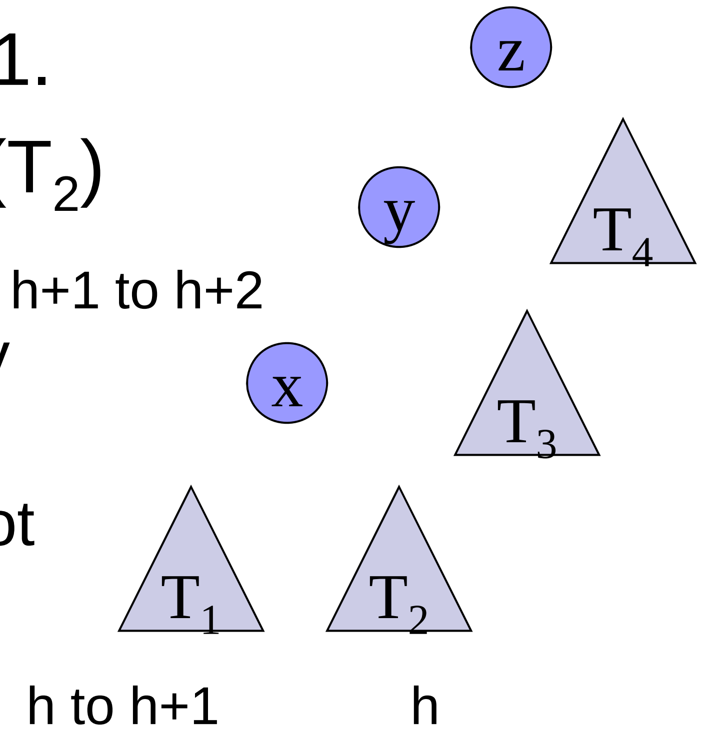
Rotations

- Rotation is a way of locally reorganizing a BST.
- Let u,v be two nodes such that u=parent(v)
- Keys(T_1) < key(v) < keys(T_2) < key (u) < keys(T_3)

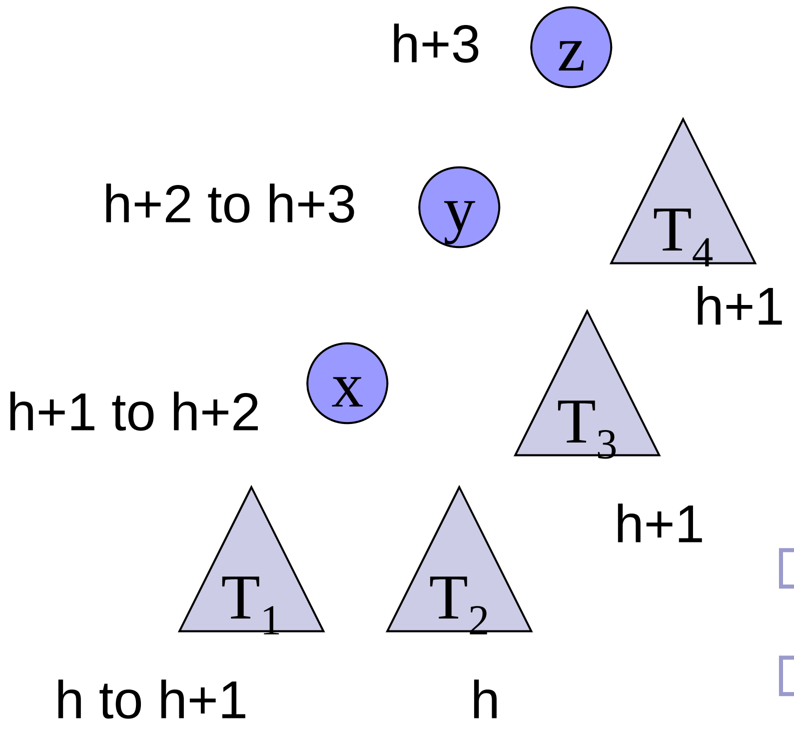


Insertion

- Insertion happens in subtree T_1.
- $ht(T_1)$ increases from h to h+1.
- Since x remains balanced $ht(T_2)$ is h or h+1 or h+2.
  - If $ht(T_2)$=h+2 then x is originally unbalanced
  - If $ht(T_2)$=h+1 then ht(x) does not increase.
  - Hence $ht(T_2)$=h.
- So ht(x) increases from h+1 to h+2.

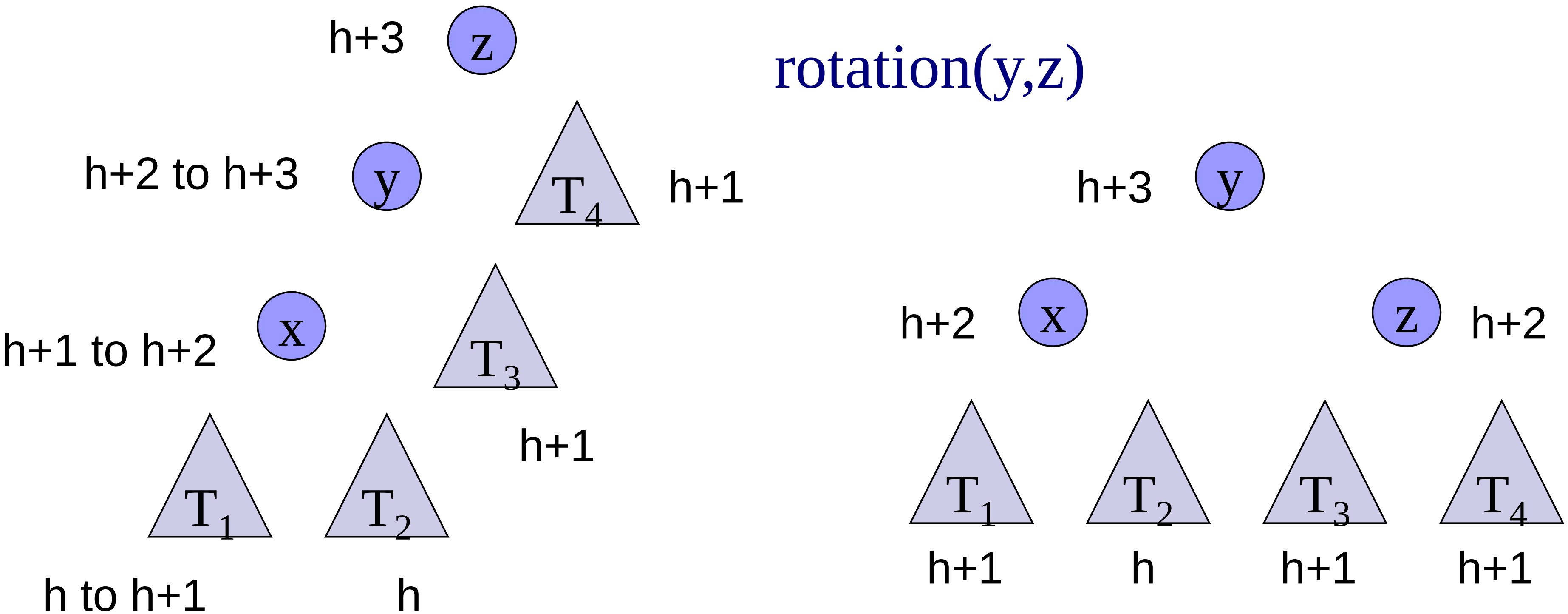


Insertion(2)



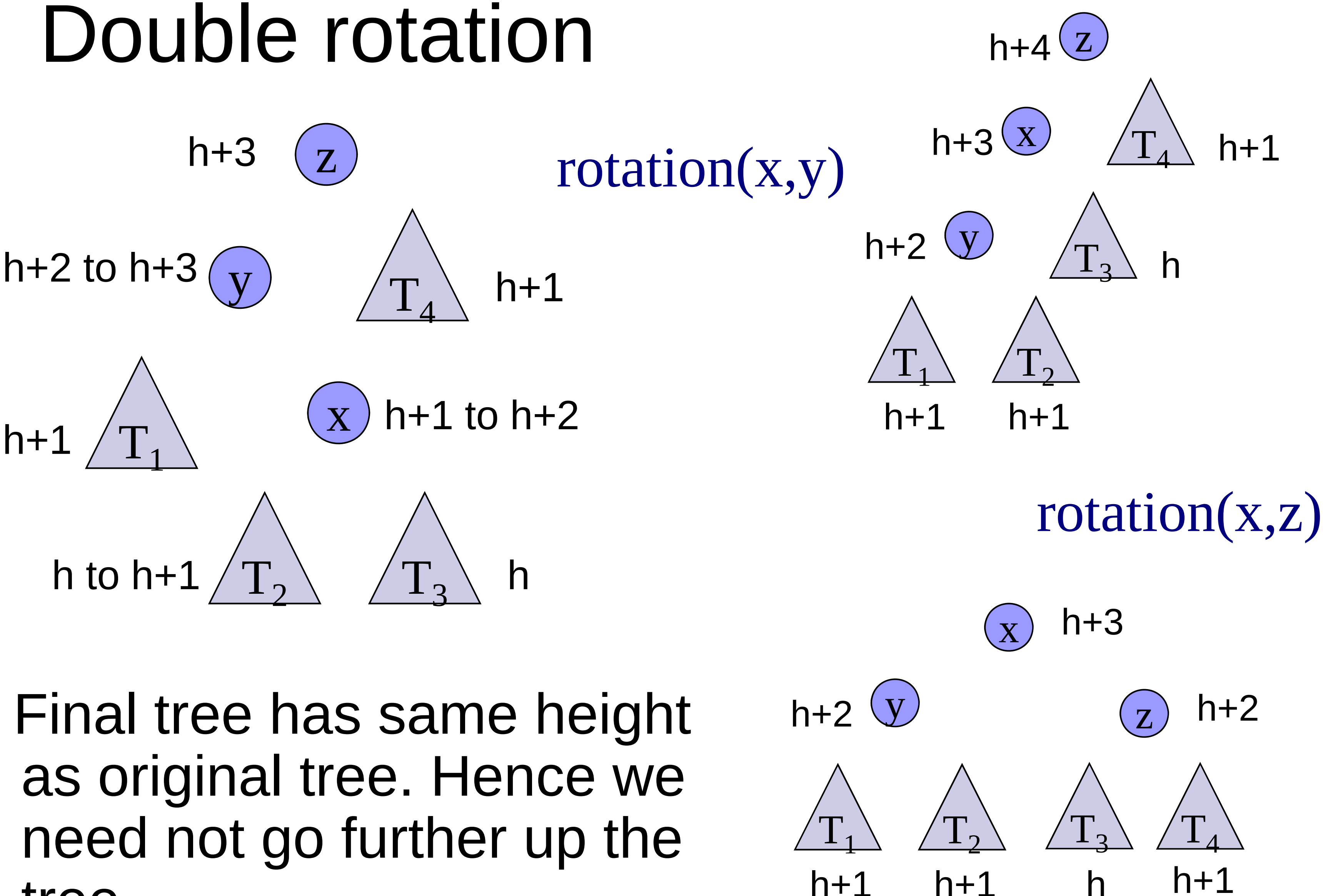
- Since y remains balanced, ht(T_3) is h+1 or h+2 or h+3.
  - If ht(T_3)=h+3 then y is originally unbalanced.
  - If ht(T_3)=h+2 then ht(y) does not increase.
  - So ht(T_3)=h+1.
- So ht(y) inc. from h+2 to h+3.
- Since z was balanced ht(T_4) is h+1 or h+2 or h+3.
- z is now unbalanced and so ht(T_4)=h+1.

Single rotation



The height of the subtree remains the same after rotation. Hence no further rotations required

Double rotation

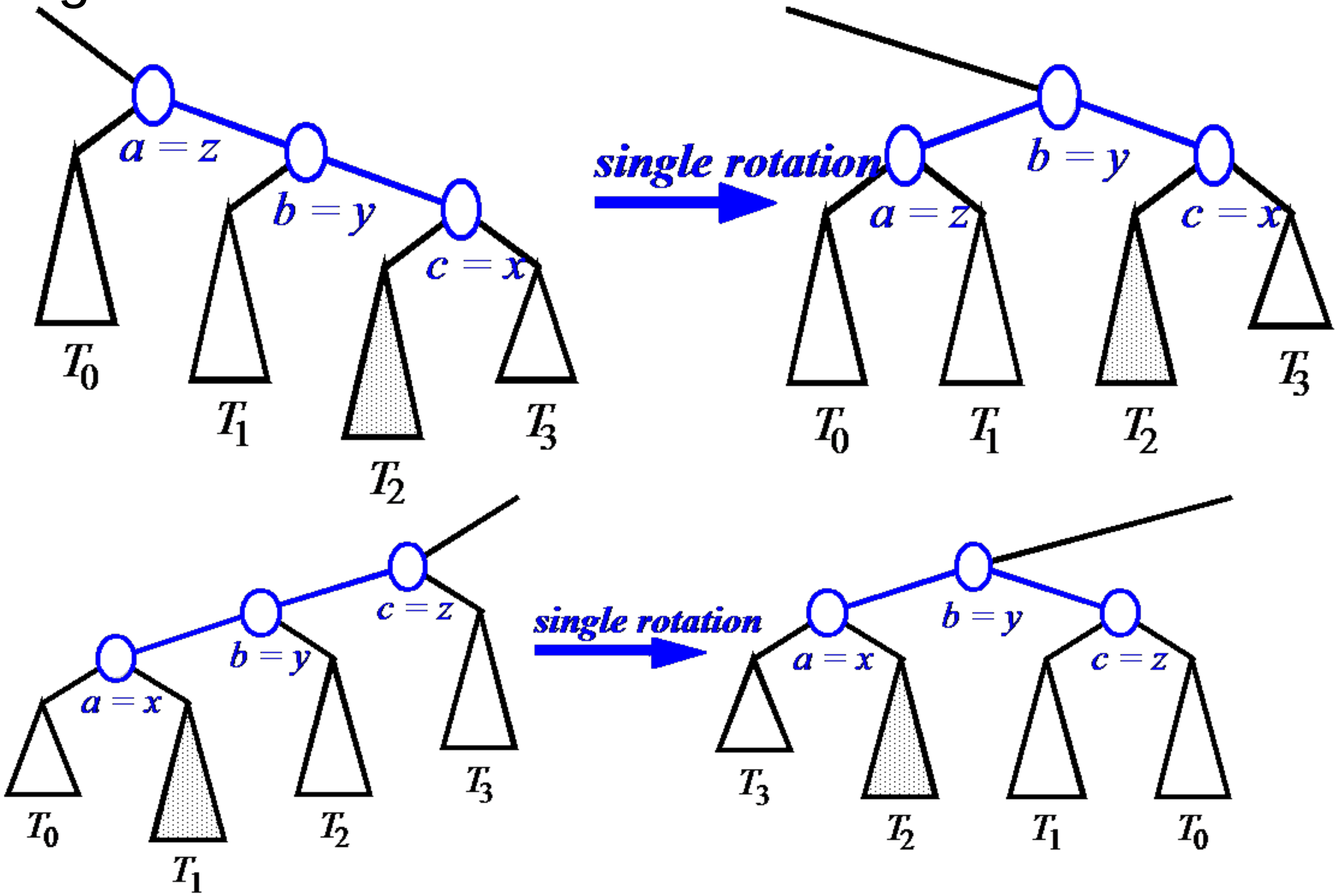


Final tree has same height as original tree. Hence we need not go further up the tree.

Restructuring

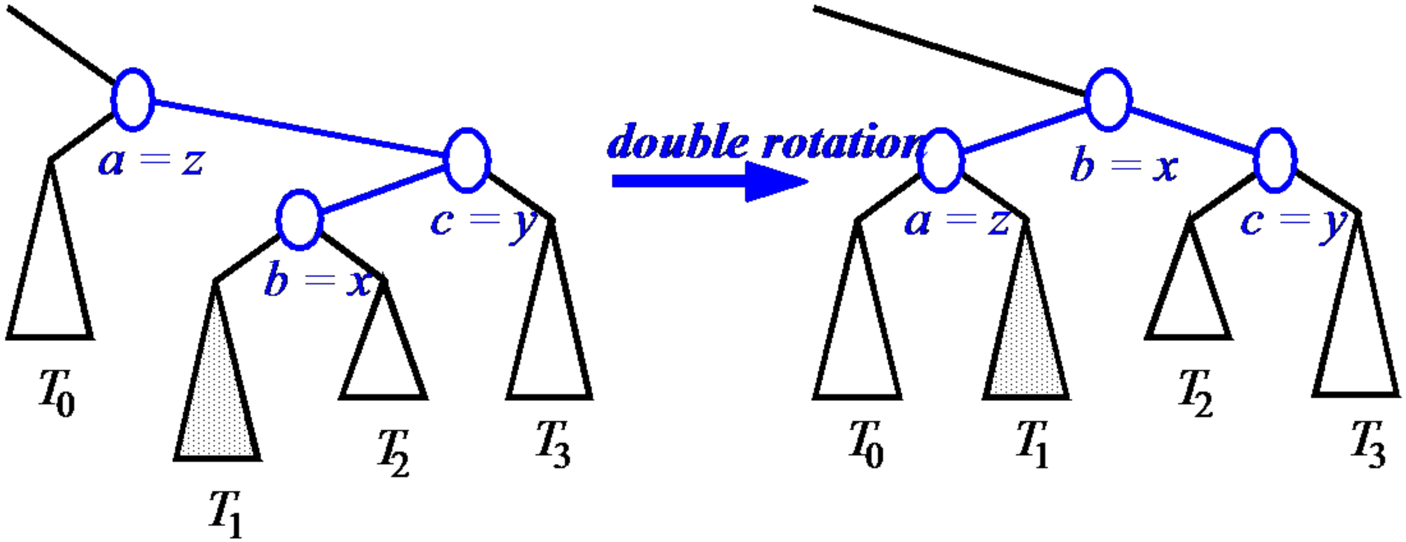
- The four ways to rotate nodes in an AVL tree, graphically represented

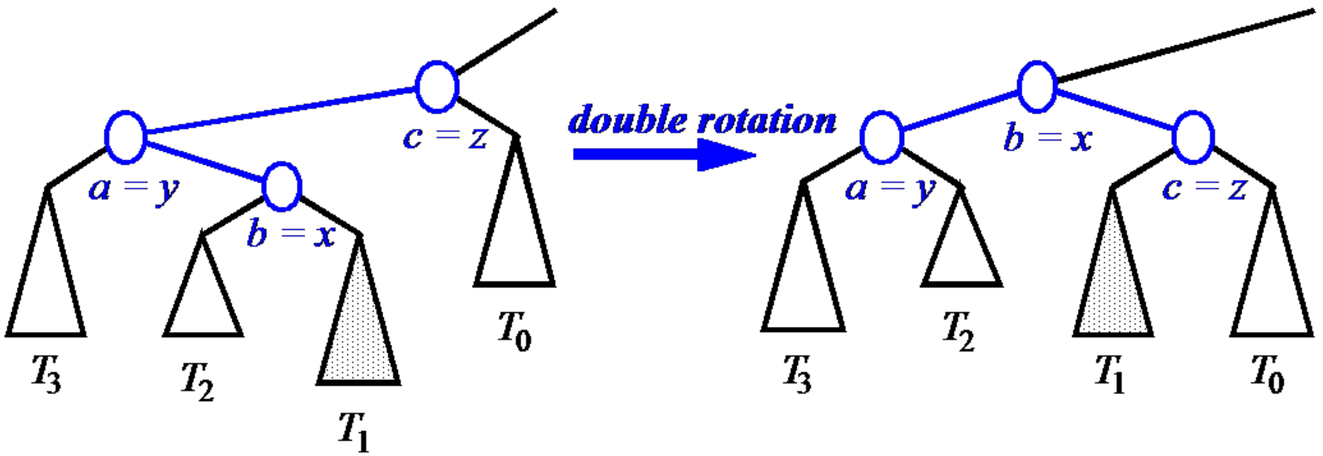
  -Single Rotations:



Restructuring (contd.)

- double rotations:





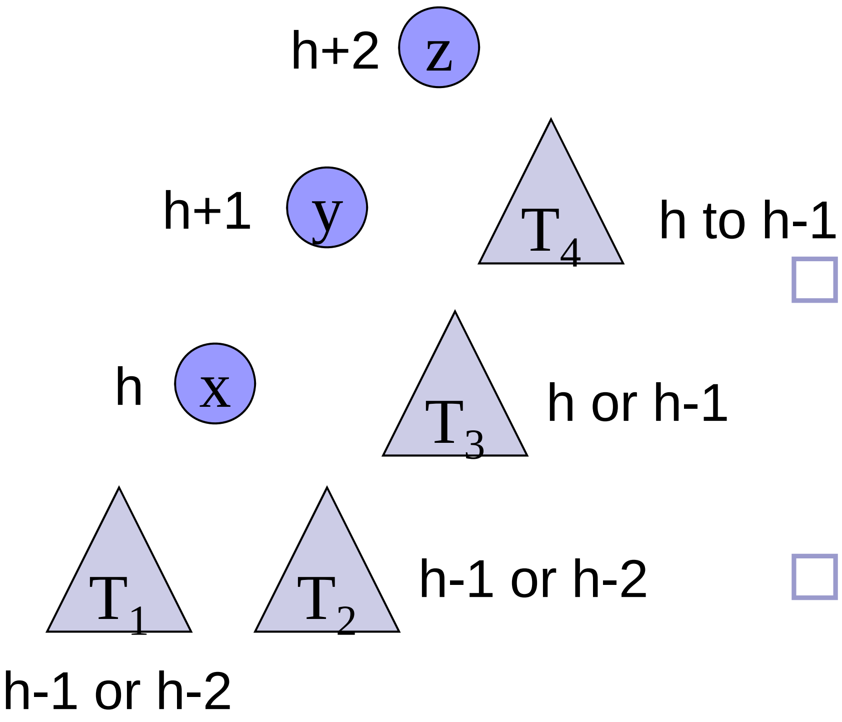
Deletion

- When deleting a node in a BST, we either delete a leaf or a node with only one child.
- In an AVL tree if a node has only one child then that child is a leaf.
- Hence in an AVL tree we either delete a leaf or the parent of a leaf.
- Hence deletion can be assumed to be at a leaf.

Deletion(2)

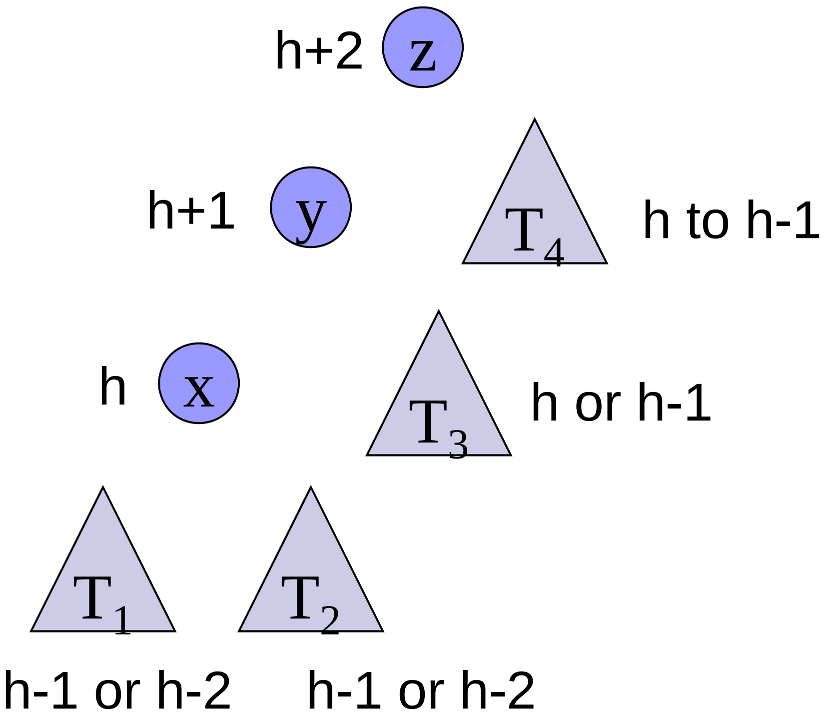
- Let w be the node deleted.
- Let *z* be the first unbalanced node encountered while travelling up the tree from w. Also, let y be the child of z with larger height, and let x be the child of y with larger height.
- We perform rotations to restore balance at the subtree rooted at z.
- As this restructuring may upset the balance of another node higher in the tree, we must continue checking for balance until the root of T is reached

Deletion(3)



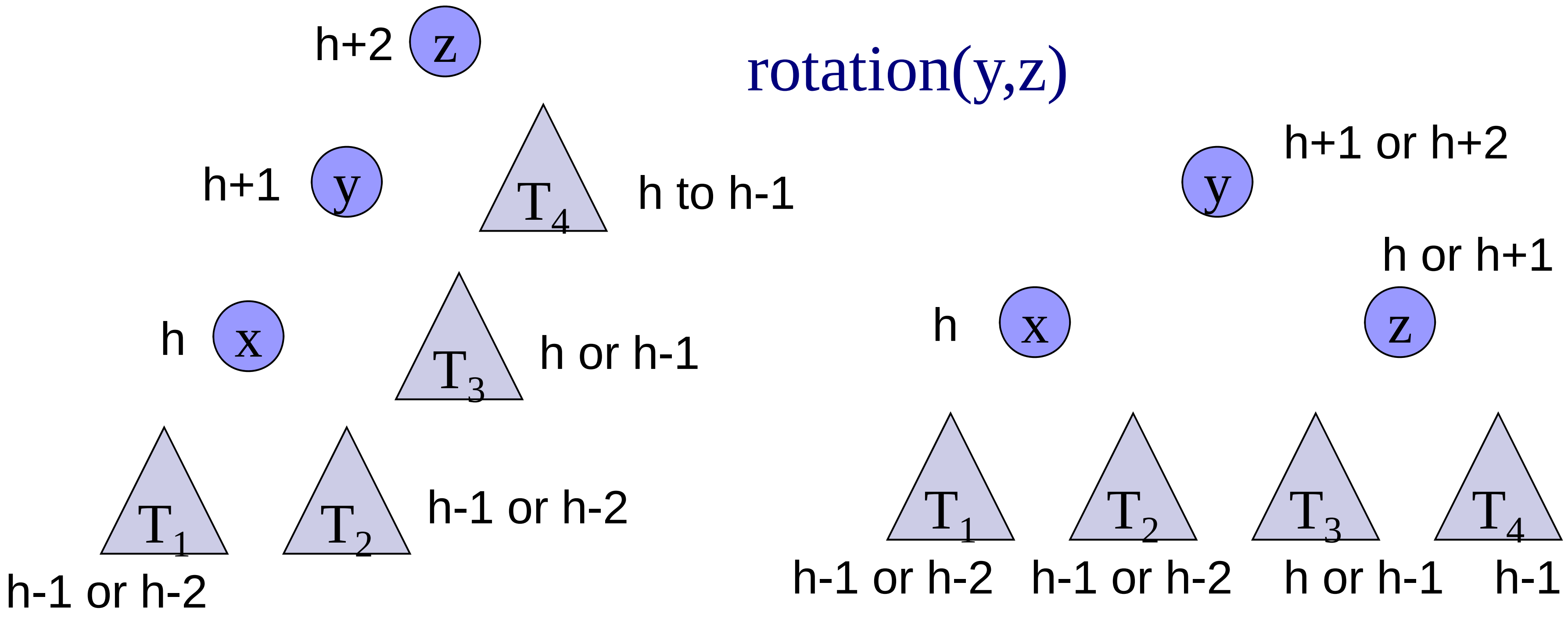
- Suppose deletion happens in subtree T_4 and its ht. reduces from h to h-1.
- Since z was balanced but is now unbalanced, ht(y) = h+1.
- x has larger ht. than T_3 and so ht(x)=h.
- Since y is balanced $ht(T_3)$= h or h-1

Deletion(4)



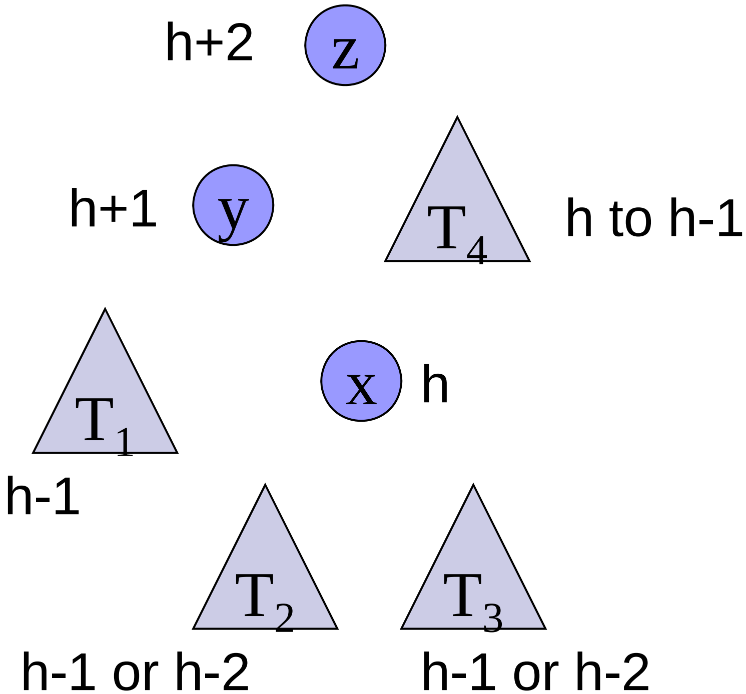
- Since ht(x)=h, and x is balanced ht(T_1), ht(T_2) is h-1 or h-2.
- However, both T_1 and T_2 cannot have ht. h-2

Single rotation (deletion)



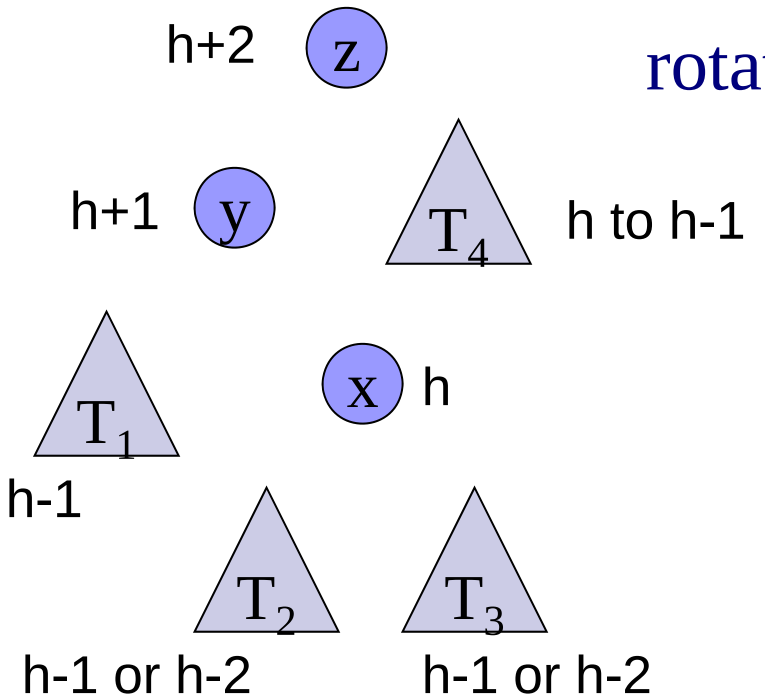
After rotation height of subtree might be 1 less than original height. In that case we continue up the tree

Deletion: another case

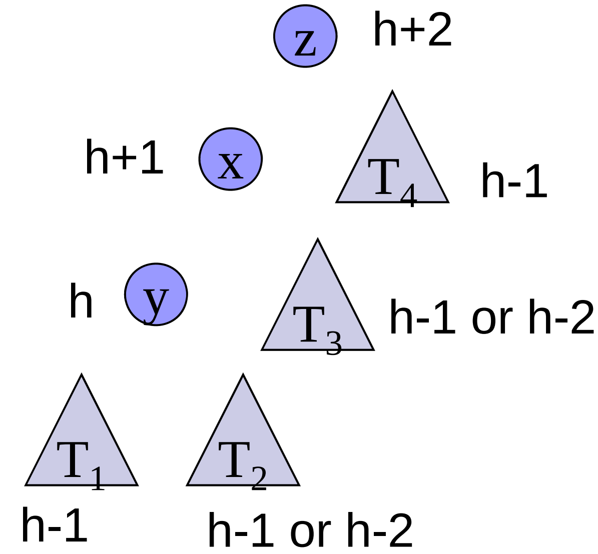


- As before we can claim that ht(y)=h+1 and ht(x)=h.
- Since y is balanced ht(T_1) is h or h-1.
- If ht(T_1) is h then we would have picked x as the root of T_1.
- So ht(T_1)=h-1

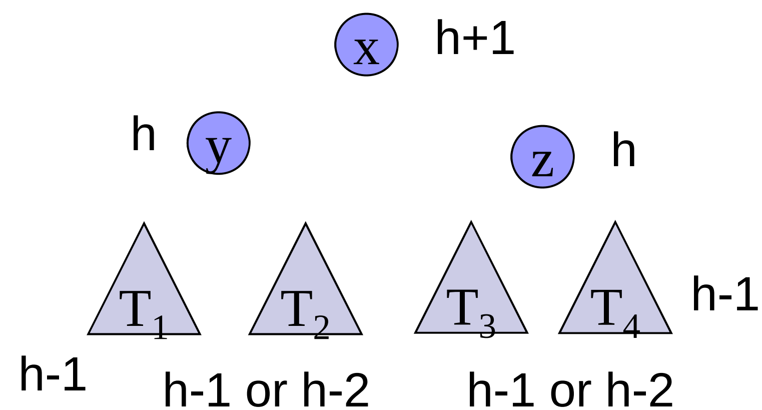
Double rotation



rotation(x,y)



rotation(x,z)



Final tree has height less than original tree. Hence we need to continue up the tree

Running time of insertion & deletion

- Insertion
  - We perform rotation only once but might have to go O(log n) levels to find the unbalanced node.
  - So time for insertion is O(log n)
- Deletion
  - We need O(log n) time to delete a node.
  - Rebalancing also requires O(log n) time.
  - More than one rotation may have to be performed.