AVL Trees 2

AVL Tree Operations

Objectives

- Describe types balanced BSTs
- Describe AVL trees
- Show that AVL trees are O(log *n*) height
- Describe and implement rotations
- Implement AVL tree insertion
- Implement AVL tree removal

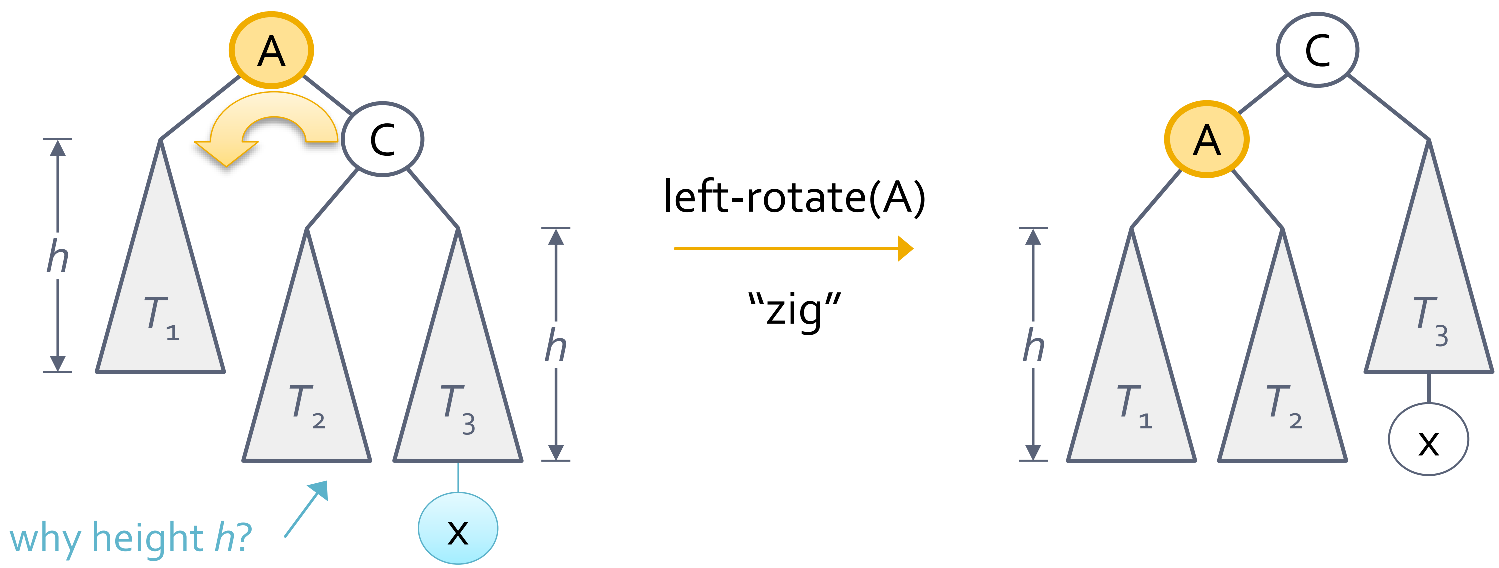
AVL material with thanks to Brad Bart

Insertion into an AVL Tree

- When an item is inserted into an AVL tree its height property may be violated
  - i.e. a node on the path from the insertion point to the root may have subtree heights that differ by greater than 1
- To correct this, perform rotations
  - Either a **single** rotation or
  - A **double** rotation
- There are four general cases
  - Two pairs of symmetric cases

Single Rotations

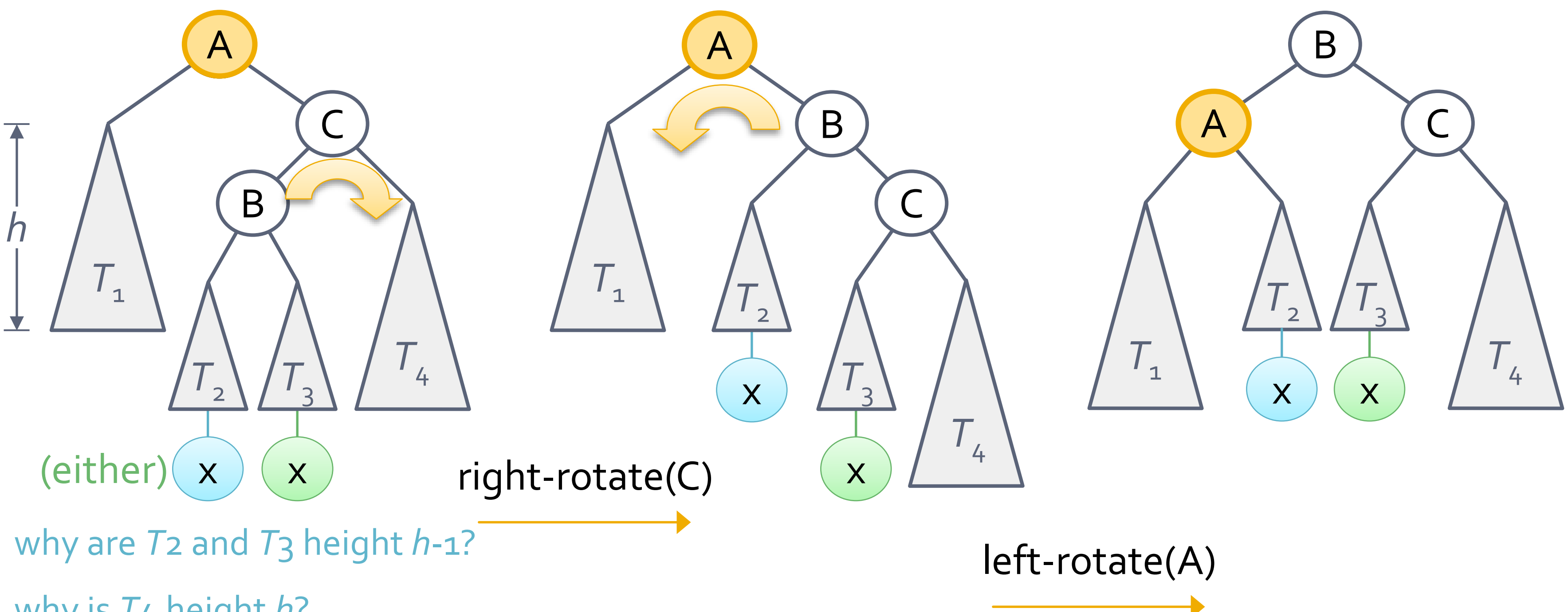
- Assume that the left child of a node has height *h* and *x* is inserted onto its R subtree
- Case 1: *x* is on the right / right grandchild



Double Rotations

- Case 2: *x* is on the right / left grandchild

why not left rotate A?



why are T_2 and T_3 height h-1?

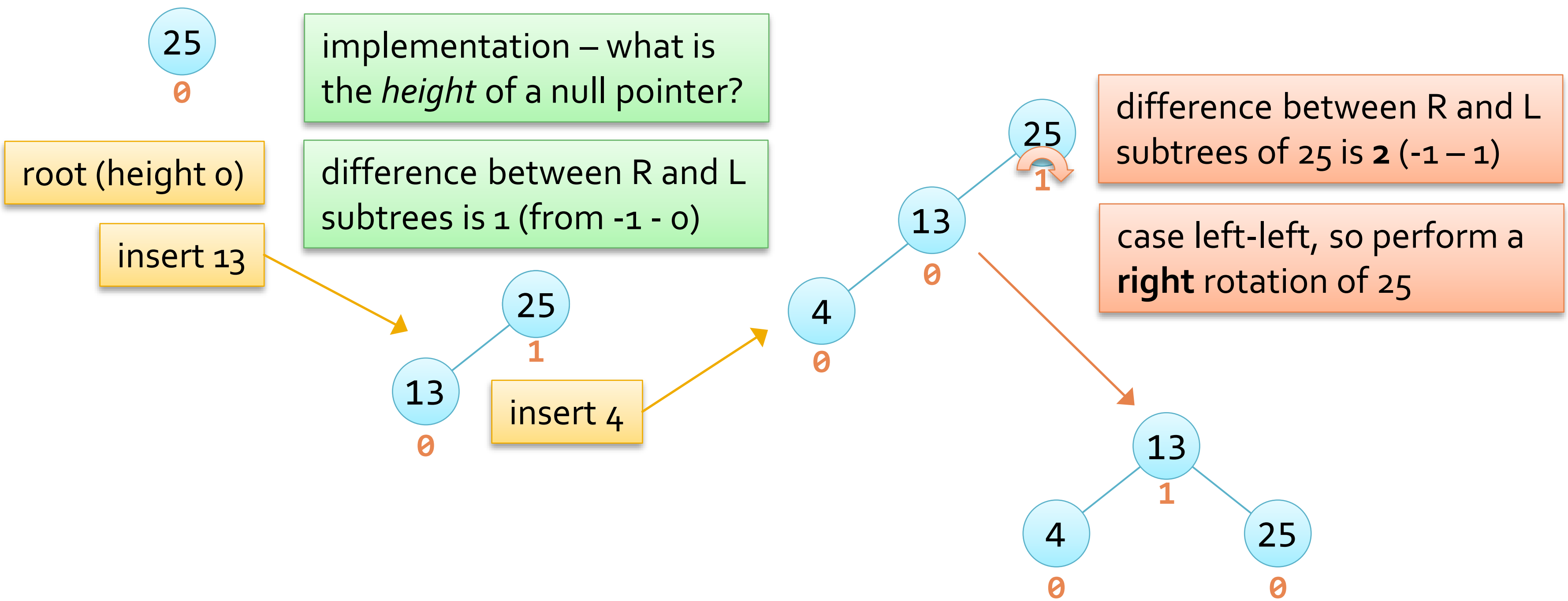
why is T_4 height h?

Insertion into an AVL Tree

- Insertion steps, for inserted node *x* with height of 0
  - Perform the standard BST insertion of *x*
  - From *x*, move up the tree (through *x*'s ancestors)
  - If a node is balanced adjust height if necessary and move up
  - If a node is unbalanced, let *z* be the unbalanced node, and let *y* and *w* be its child and grandchild on the path from *x*
  - Perform a rotation
    - *y* is **L** child of *z* and *w* is **L** child of *y* (*left left*) – right single rotation
    - *y* is **L** child of *z* and *w* is **R** child of *y* (*left right*) – left, right double rotation
    - *y* is **R** child of *z* and *w* is **R** child of *y* (*right right*) – left single rotation
    - *y* is **R** child of *z* and *w* is **L** child of *y* (*right left*) – right, left double rotation

check height of both subtrees, if the subtree with insertion is higher, +1 to node's height

Insertion Example 1



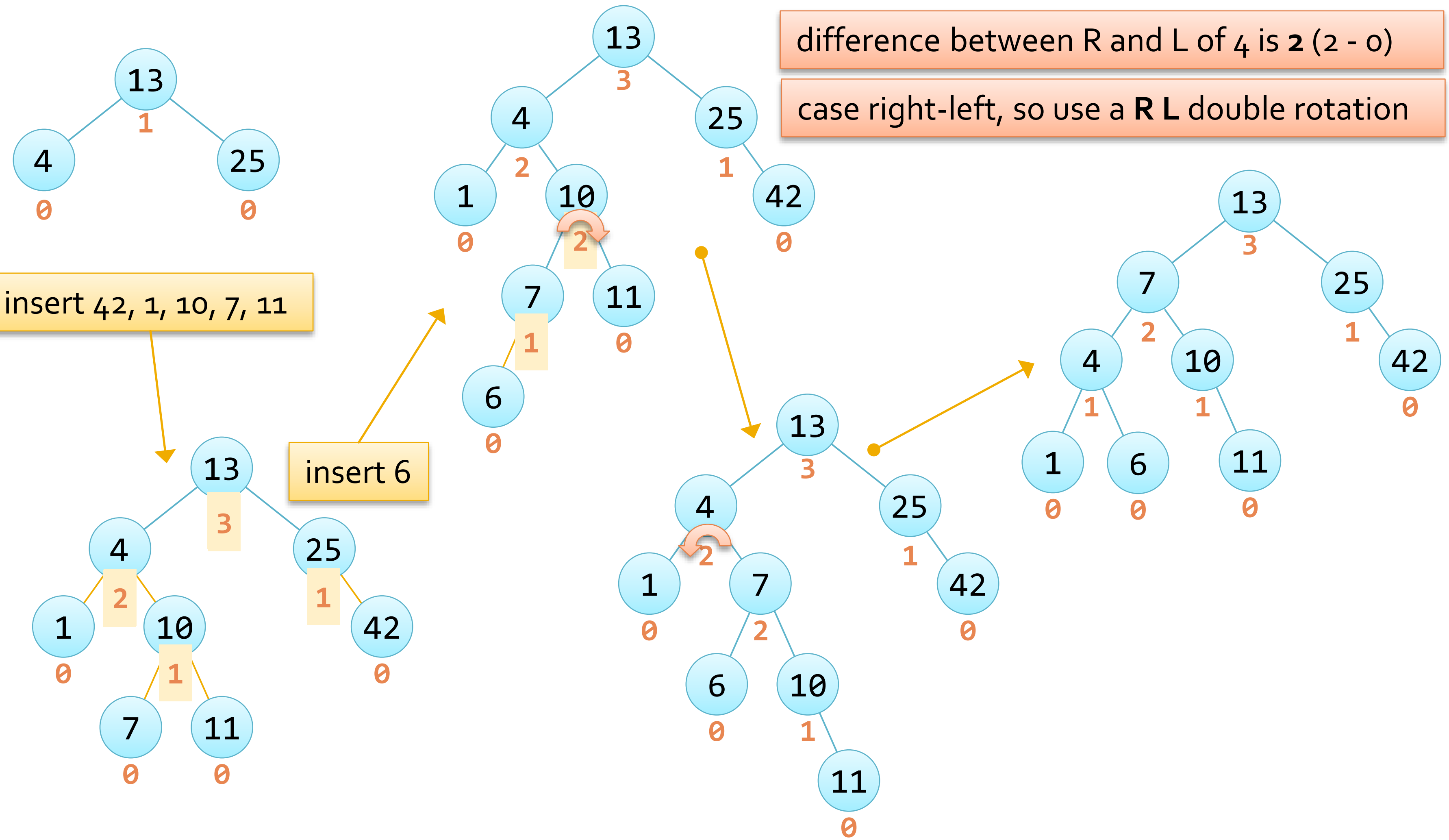
Insertion Example 2a

difference between R and L of 4 is **2** (2 - 0)

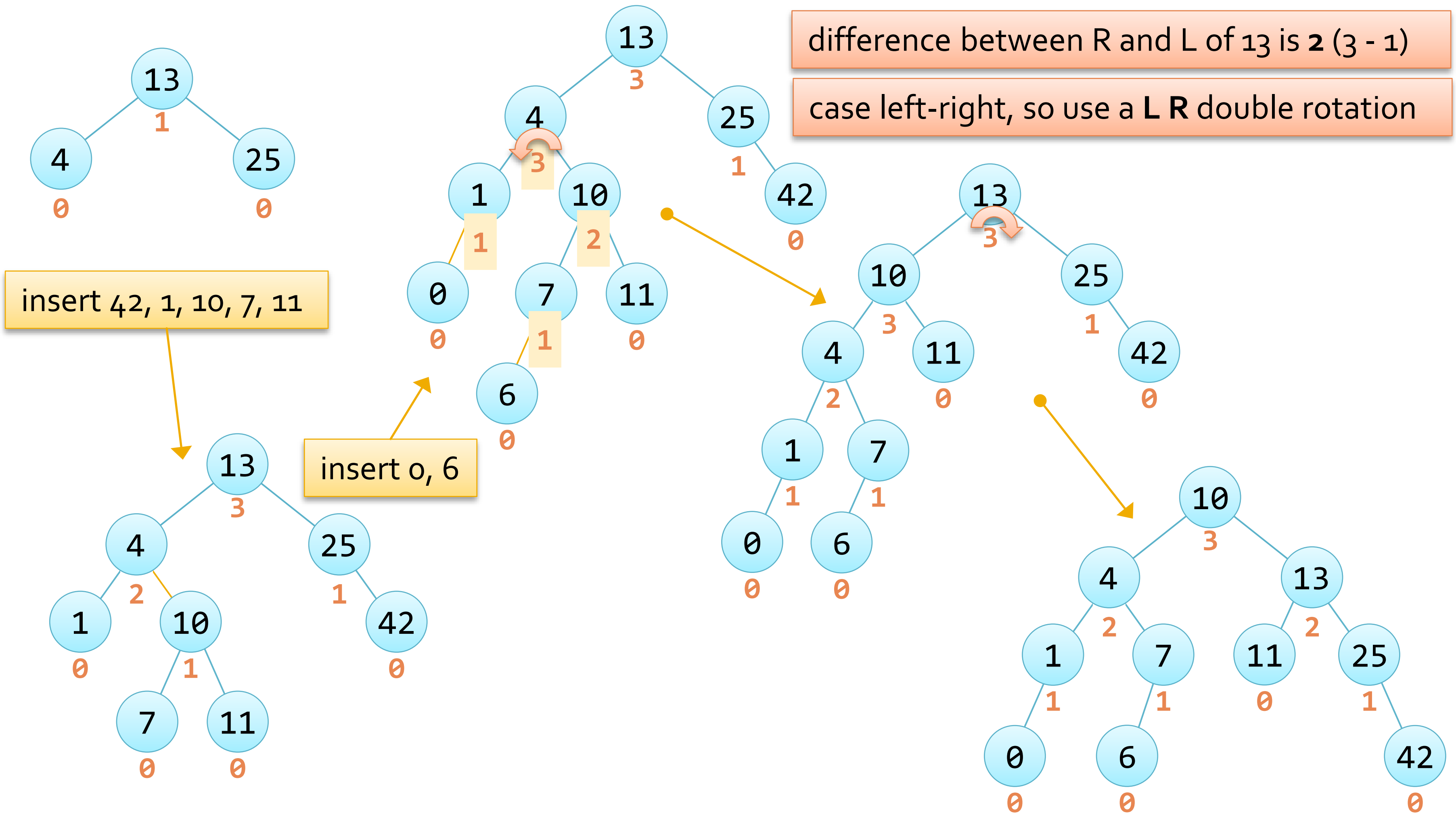
case right-left, so use a **R L** double rotation

insert 42, 1, 10, 7, 11

insert 6



Insertion Example 2b



Removal from an AVL Tree

- Follow the standard BST removal process
  - If the node, *x*, to be removed has two children, replace it with its predecessor

If *x* is replaced by its *predecessor* set *predecessor*'s height to *x*'s height

  - Let *v* be the parent of *x*, or if *x* is replaced by its predecessor let *v* be the parent of the predecessor
- If *v* has a child set *v* to its child

v can have only one child, and that child cannot have children – why?

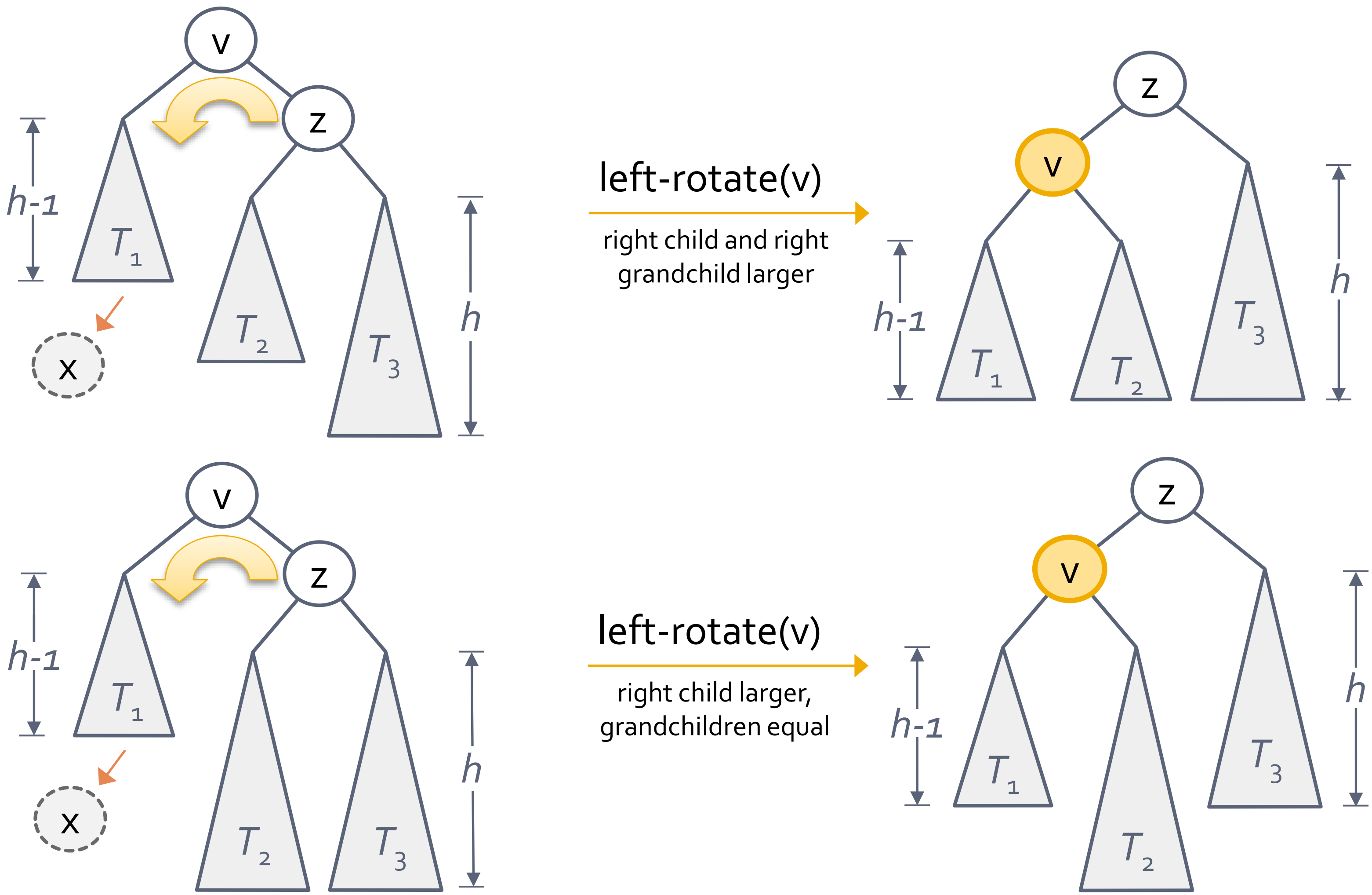
  - Start the balancing process from *v*
- Multiple rotations may be required

Rebalancing After Removal

- The process for fixing an AVL tree after removal of a node is similar to insertion
- From *v* (see previous slide), move up the tree through *v*'s ancestors
  - If a node is balanced adjust height and move up
  - If a node is unbalanced perform a rotation of that node
- Identifying the rotation is different from insertion
  - And depends on the relative heights of the subtrees of the larger child, let it be *z*

    where the larger child is the one *not* on the path of the removal
    - If larger child of *z* is innermost a double rotation is required

Removal Single Rotation



Removal Double Rotation