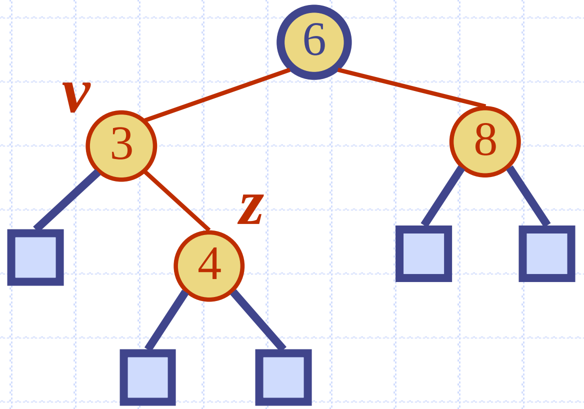
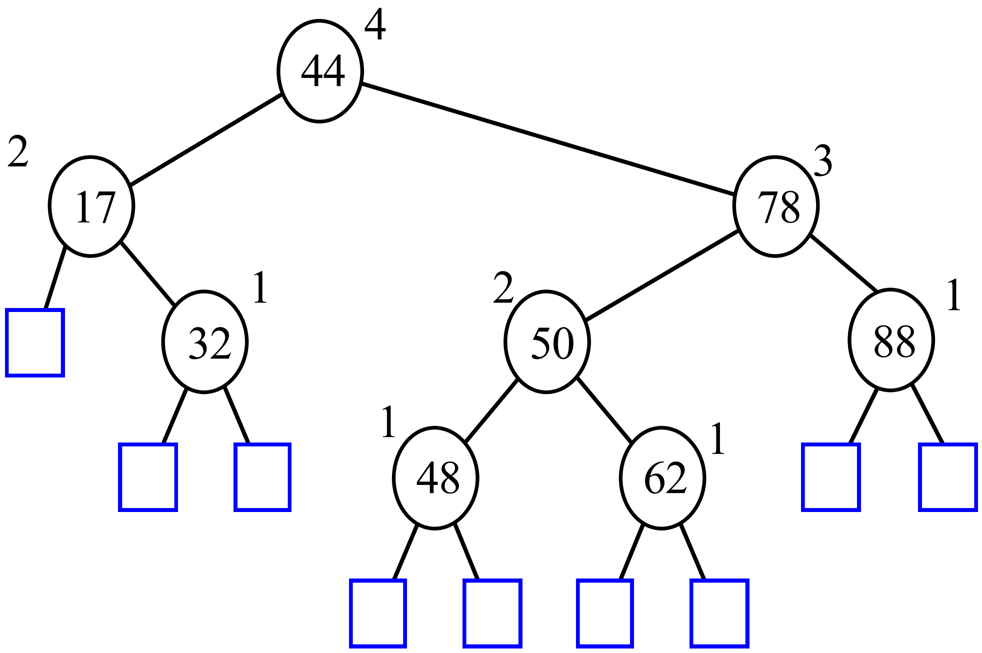
AVL Trees (高度平衡樹)



AVL Tree Definition

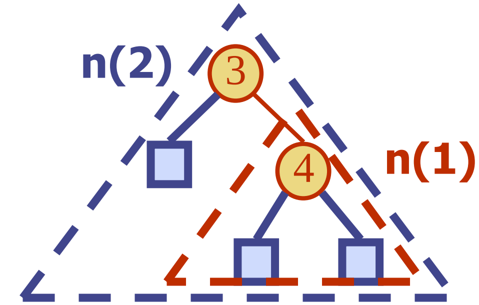
Quiz!

- An AVL Tree is a binary search tree such that for every internal node v of T, the heights of the children of v can differ by at most 1
- Proposed by G. M. Adelson-Velsky & E. M. Landisin 1962.



An example of an AVL tree where the heights are shown next to the nodes:

Height of an AVL Tree

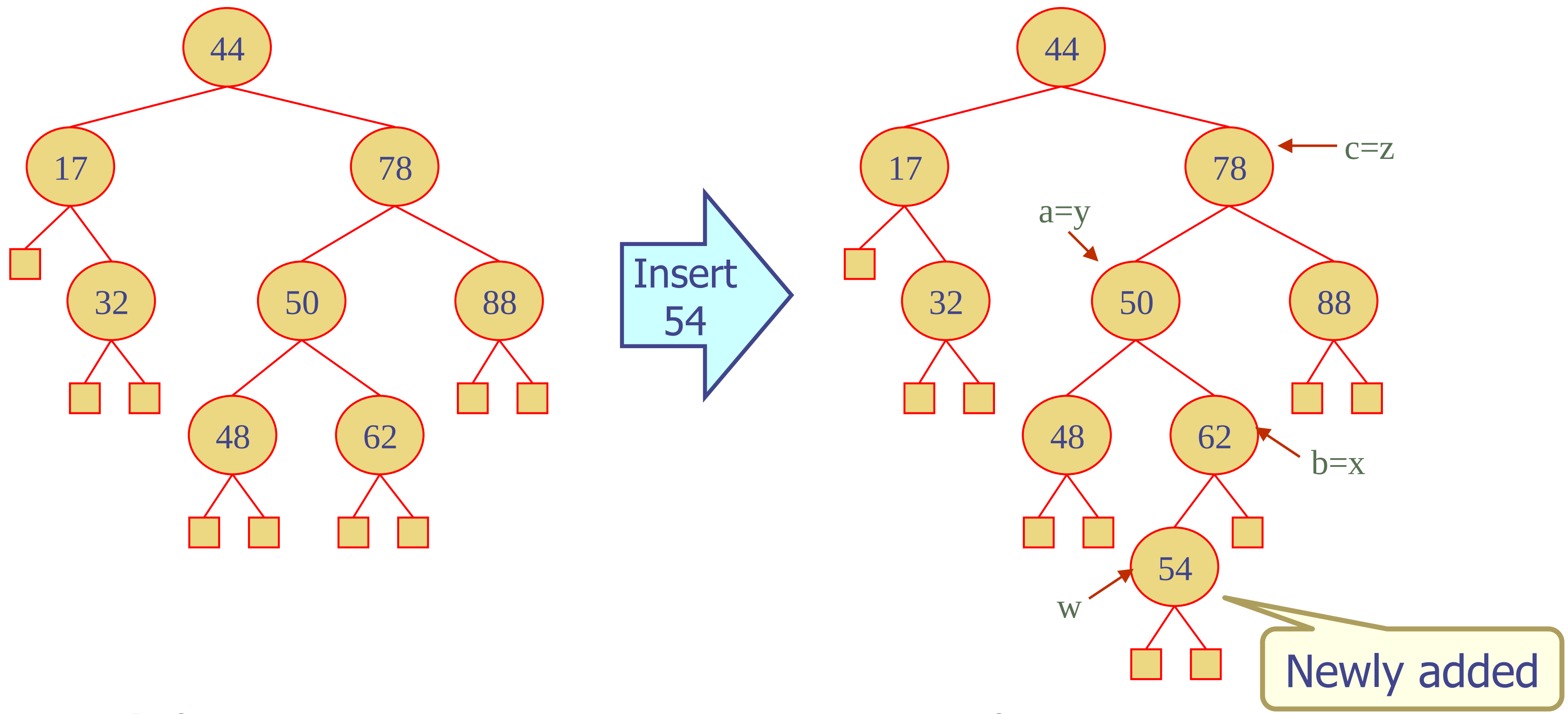


- Fact: The height of an AVL tree storing n keys is O(log n).
- Proof:
  - Let us bound n(h), the minimum number of internal nodes of an AVL tree of height h.
  - Base cases: $n(1) = 1$ and $n(2) = 2$
  - Recurrent formula: For $n > 2$, an AVL tree of height h contains the root node, one AVL subtree of height n-1 and another of height n-2
    $\Rightarrow n(h) = 1 + n(h-1) + n(h-2)$
  - Knowing $n(h-1) > n(h-2)$, we get $n(h) > 2n(h-2)$. So
    $n(h) > 2\, n(h-2) > 2^2\, n(h-4) > 2^3\, n(h-6) > \ldots > 2^i\, n(h-2i)$
    $\Rightarrow n(h) > 2^{h/2-1} \Rightarrow h < 2\log n(h) + 2$
  - Thus the height of an AVL tree is O(log n)

Moer details in textbook

Insertion

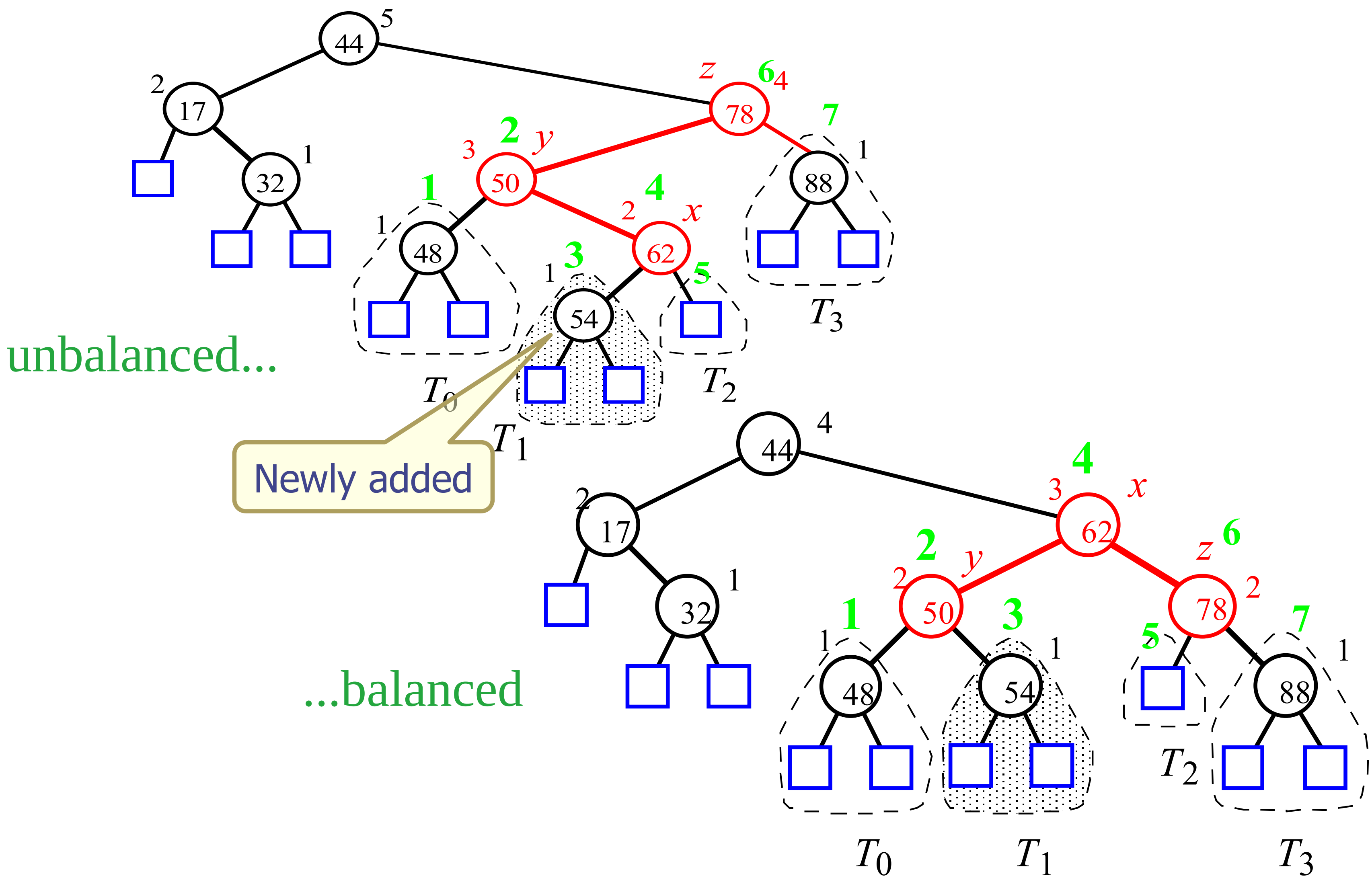
- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example:



before insertion

after insertion

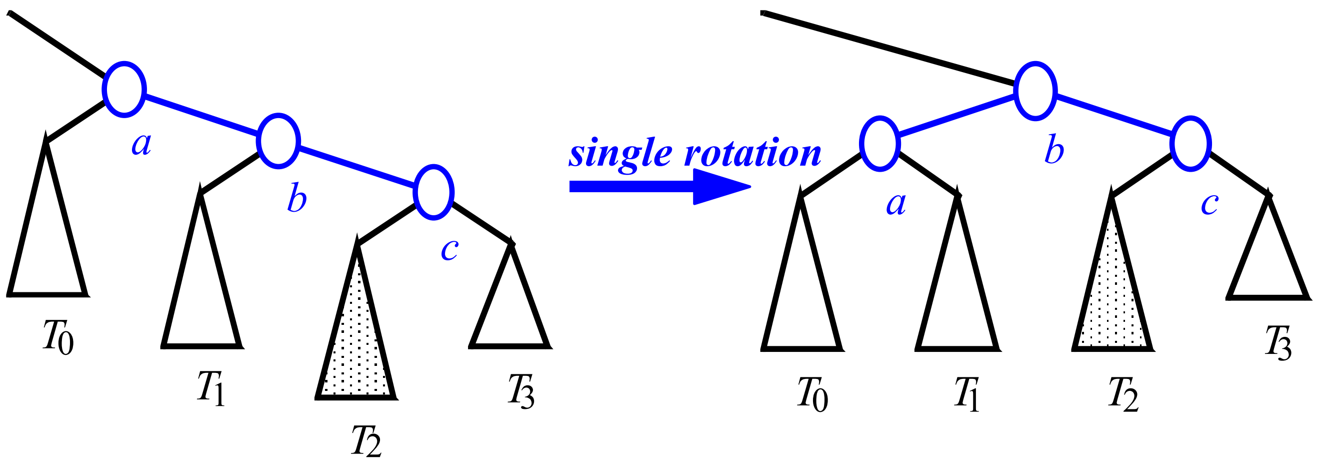
Insertion Example, continued



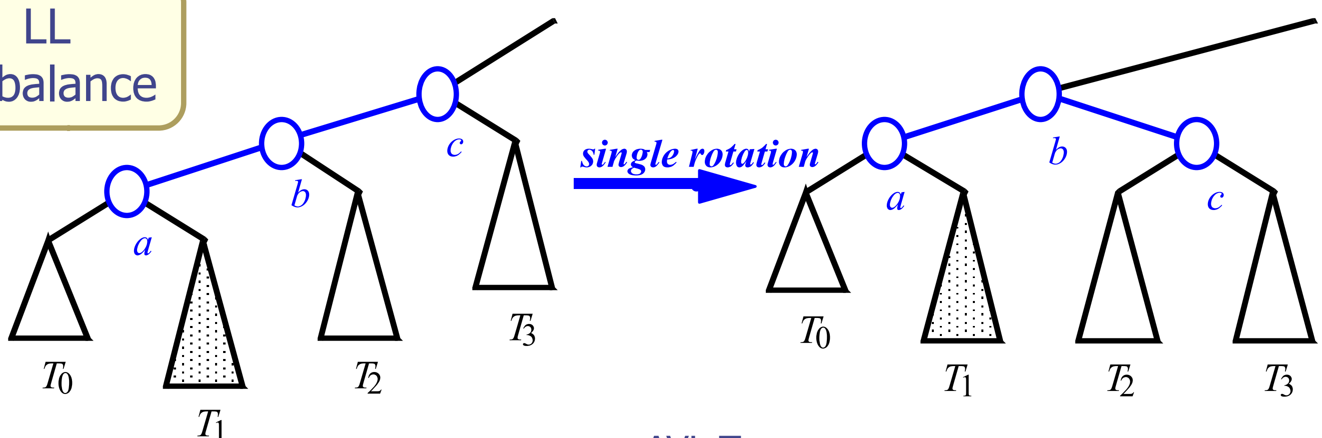
Single Rotations

- Single Rotations:

RR
Imbalance

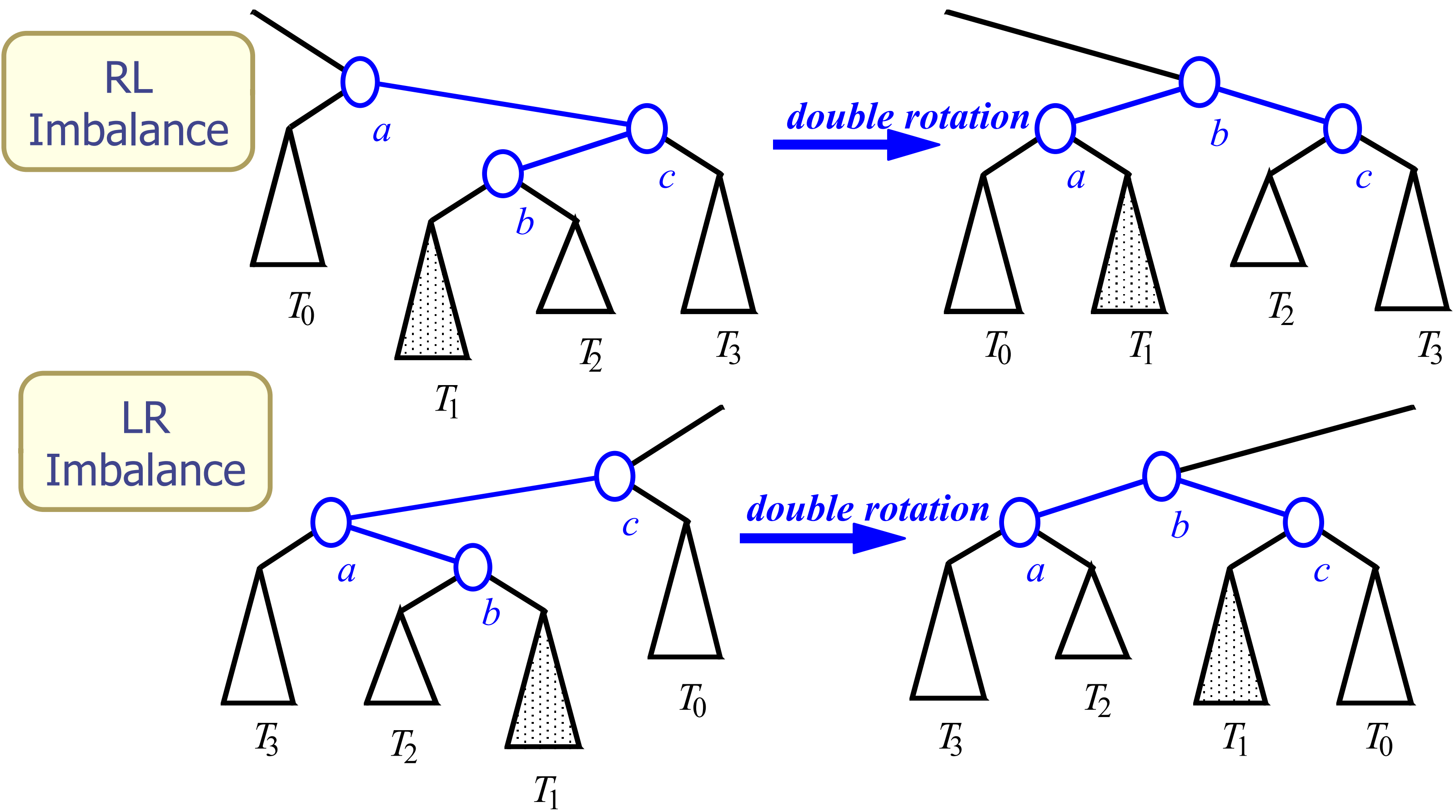


LL
Imbalance



Double Rotations

- double rotations:



Recap on Insert

- From the inserted node, you need to find the first node x leading to the root that has AVL violation.
- Perform restructure(x) only once to restore all AVL order leading to the root
- Restructure(x)
  - RR or LL imbalance → Single rotation
  - RL or LR imbalance → Double rotations

Example of Single Rotations

Quiz!

- Insert 1, 2, 3, 4, 5, 6, 7 into an AVL tree.

Example of Single and Double Rotations

Quiz!

- Insert 1, 3, 4, 15, 14, 12, 2 into an AVL tree.

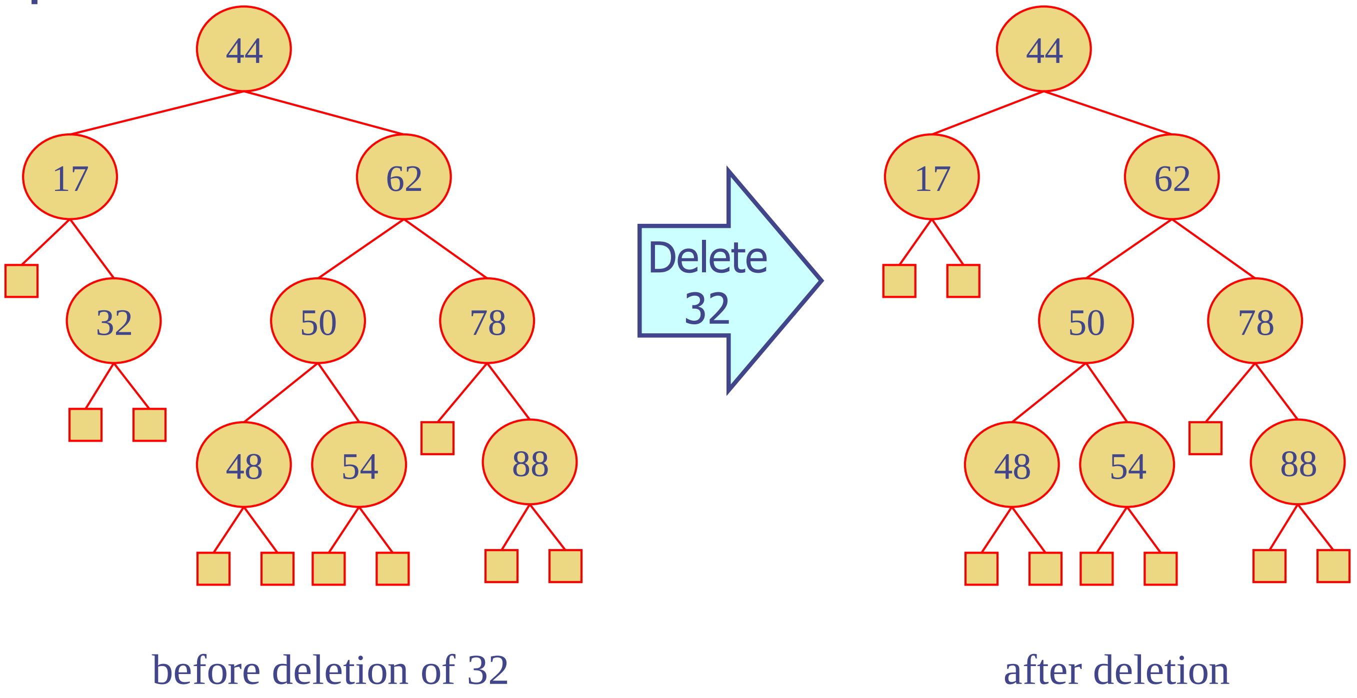
Exercise

Quiz!

- Insert 12, 8, 7, 14, 18, 10, 20, 16, 15 with AVL rotations.

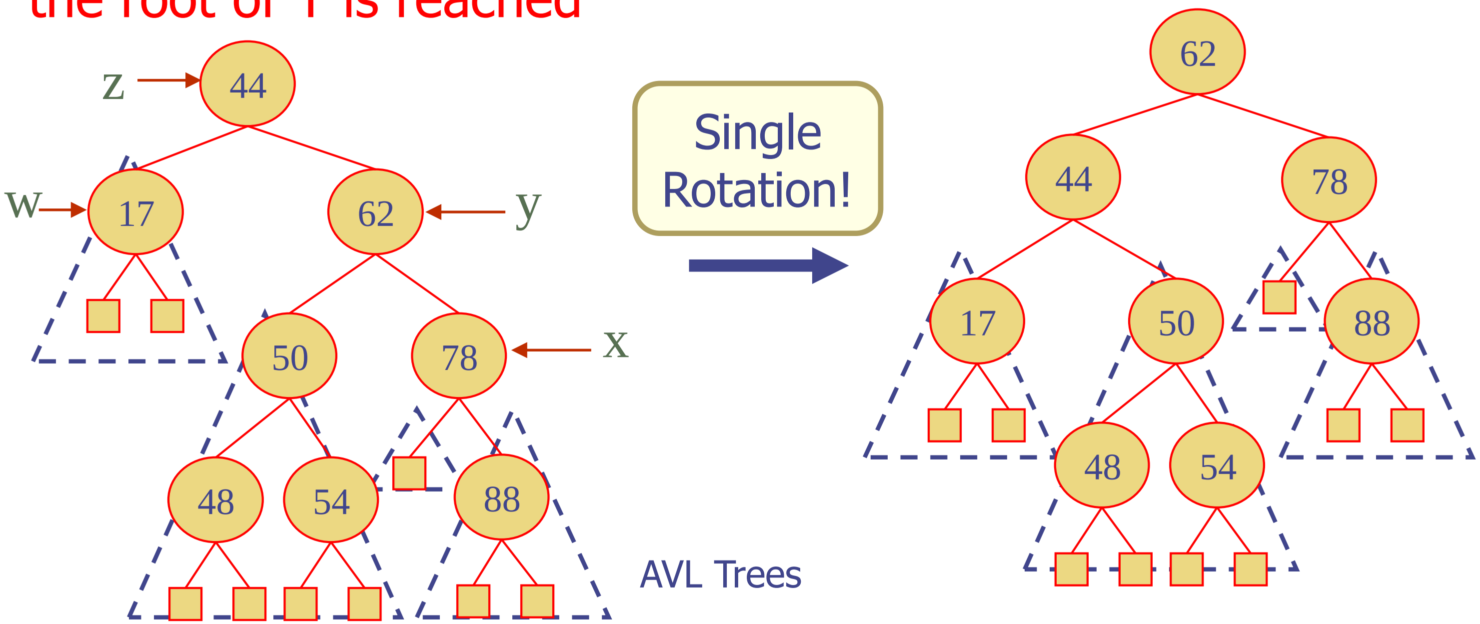
Delete

- Delete begins as in a binary search tree, which means the node removed will become an empty external node. Its parent, w, may cause an imbalance.
- Example:



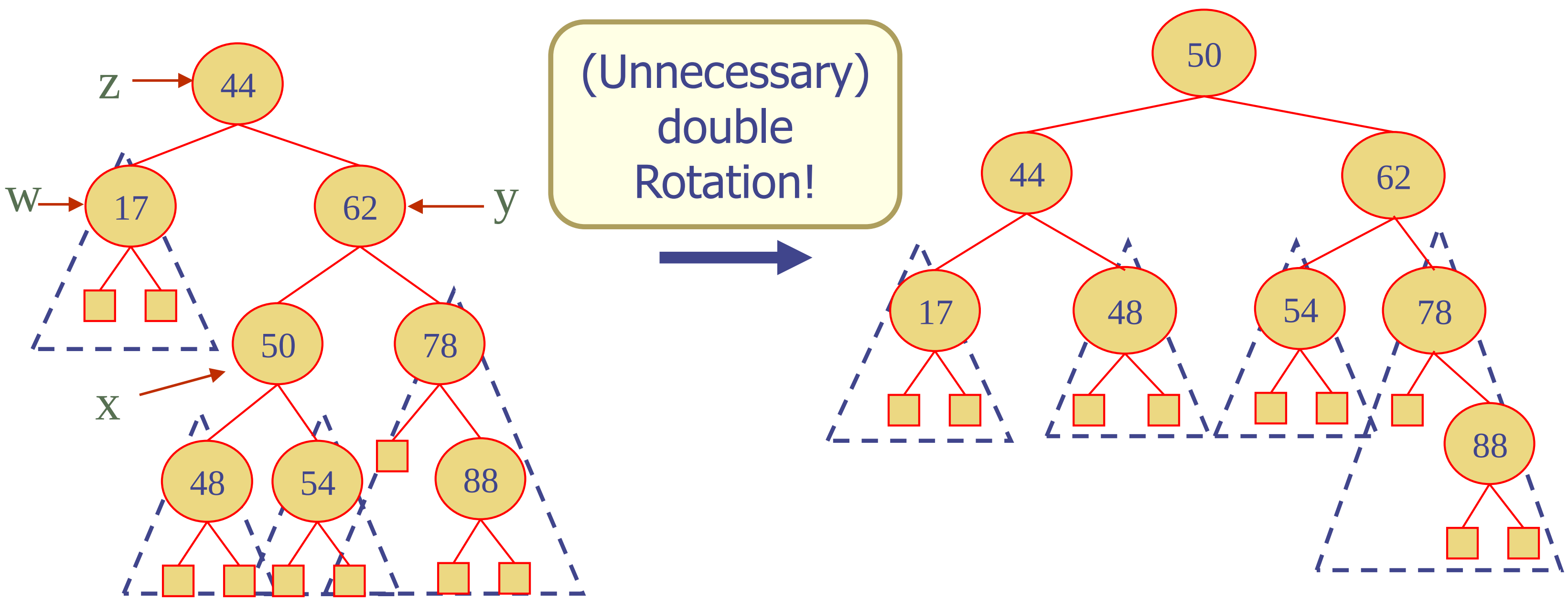
Rebalancing after a Delete (1/2)

- Let z be the first unbalanced node encountered while travelling up the tree from w. Also, let y be the child of z with the larger height, and let x be the child of y with the larger height
- If x has RR imbalance, we perform restructure(x) or single rotation to restore balance 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



Rebalancing after a Delete (2/2)

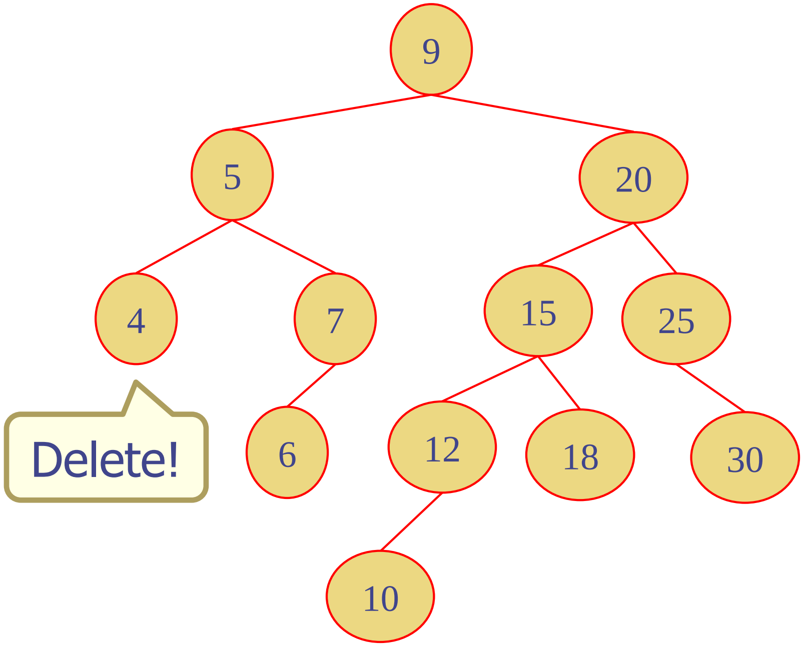
- If we have a tie and x is chosen to have RL imbalance
- We perform restructure(x) or double rotations (which is unnecessary) to restore balance at z



Example of Complex Deletes (1/2)

- Delete that involves two restructures

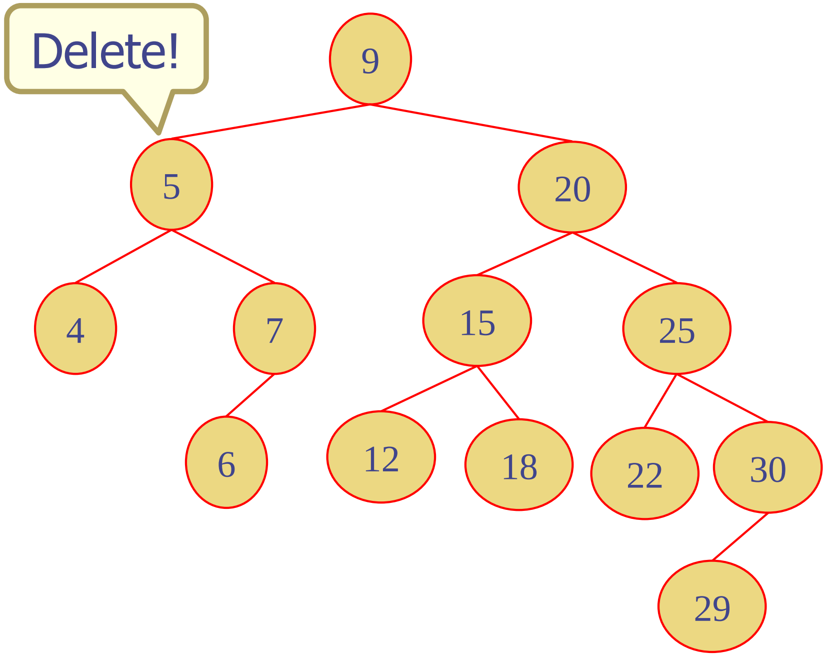
Quiz!



Example of Complex Deletes (2/2)

- Delete that involves two restructures

Quiz!



Recap on Deletes

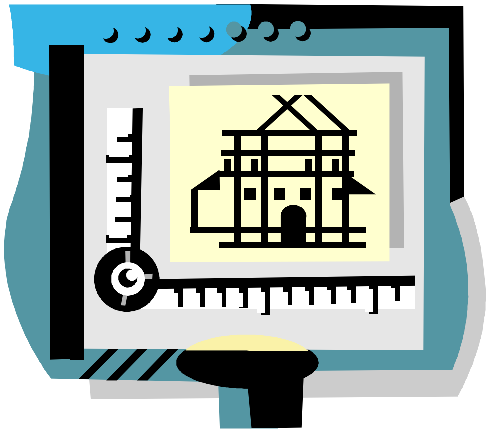
Quiz!

- Comparison
  - For deletes, you need to check imbalance all the way to the root
  - For inserts, you need only perform restructuring once.
- To make the delete sequence generate the same tree each time:
  - Use the in-order successor (if the node has both subtrees) to replace the deleted node.
  - Use single rotation whenever possible.

Youtube Links for AVL Trees

- Intro to AVL tree
  - MIT open course ware (For inserts only. The lecturer mistakenly said that you need to check all the way to the root…)
- Tree growing
  - A detailed example from SDSU: 43, 18, 22, 9, 21, 6, 8, 20, 63, 50, 62, 51.
- Tree shrinking
  - A simple example
  - Another example
  - Yet another example

AVL Tree Performance



- a single restructure takes O(1) time
  - using a linked-structure binary tree
- find takes O(log n) time
  - height of tree is O(log n), no restructures needed
- put takes O(log n) time
  - initial find is O(log n)
  - Restructuring up the tree, maintaining heights is O(log n)
- erase takes O(log n) time
  - initial find is O(log n)
  - Restructuring up the tree, maintaining heights is O(log n)