CALIFORNIA STATE UNIVERSITY
FULLERTON

CPSC 131

Data Structures

AVL Trees

Tree Height and BST Issue

Depth

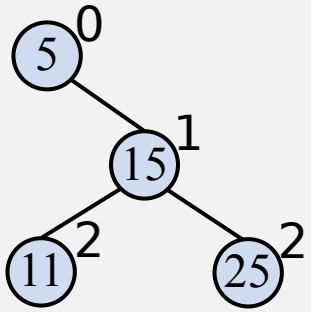
- **Depth of a node:** number of ancestors (between the node and the root)

  Root has depth 0

- Depth of p's node is recursively defined:

```
if (p.isRoot()) return 0;              // root has depth 0
else return 1 + depth(p.parent());     // 1 + (depth of parent)
```

```
map<int, string> ds;
ds[ 5] = "Monday";
ds[15] = "Tuesday";
ds[25] = "Wednesday";
ds[11] = "Thursday";
```

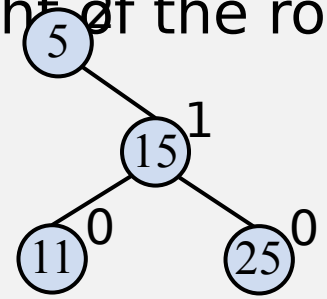


What is the depth at each node?

Height

- Height of a node p in a tree T is defined recursively:
  - If p is the external node, then the height of p is 0
  - Otherwise, the height of p is 1 + the maximum height of children of p
- Height of a tree == maximum depth of its external nodes
  - Height of a tree T is the height of the root of T

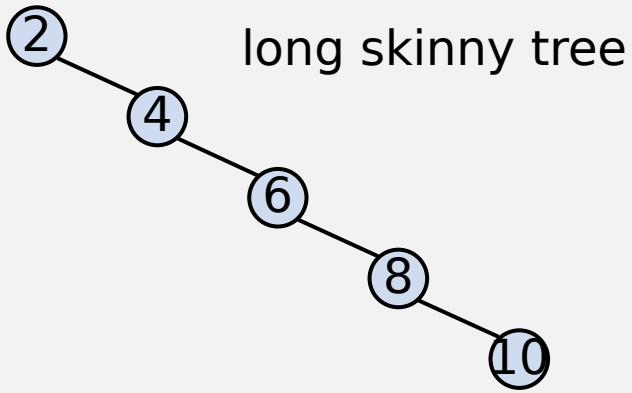
```
map<int, string> ds;
ds[5] = "Monday";
ds[15] = "Tuesday";
ds[25] = "Wednesday";
ds[11] = "Thursday";
```



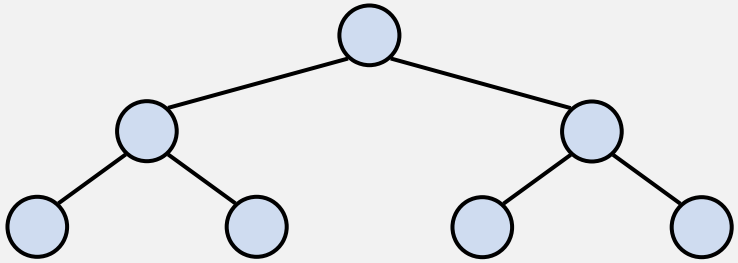
What is the height at each node?

Issue with Binary Search Tree Performance

- A binary search tree (BST) with ***n*** items
  - The space used is ***O*(*n*)**
  - Methods find, insert and erase take ***O*(*height*)** time
  - height = n, worst case ® long skinny tree
  - Example: insert 2, 4, 6, 8, 10 into a BST
  - What is the issue? **Lack of balance**!
  - height = log ***n***, best case ® balanced tree



long skinny tree



balanced tree

Self-Balancing Trees

- Always ensure that BST is balanced

  IF an insert (or delete) makes the BST not balanced, then *rearrange the nodes* so that the tree stays balanced

  Challenge: the node rearrangement should not take too long!

  - Otherwise, lose the search speed benefit

  Node rearrangement still keeps the same nodes in sorted order but in a different layout

- Many types of balanced trees

  AVL trees

  - The first self-balancing tree
  - Invented in 1962 by Russian mathematicians Georgy **A**delson-**V**elsky and Evgenii **L**andis

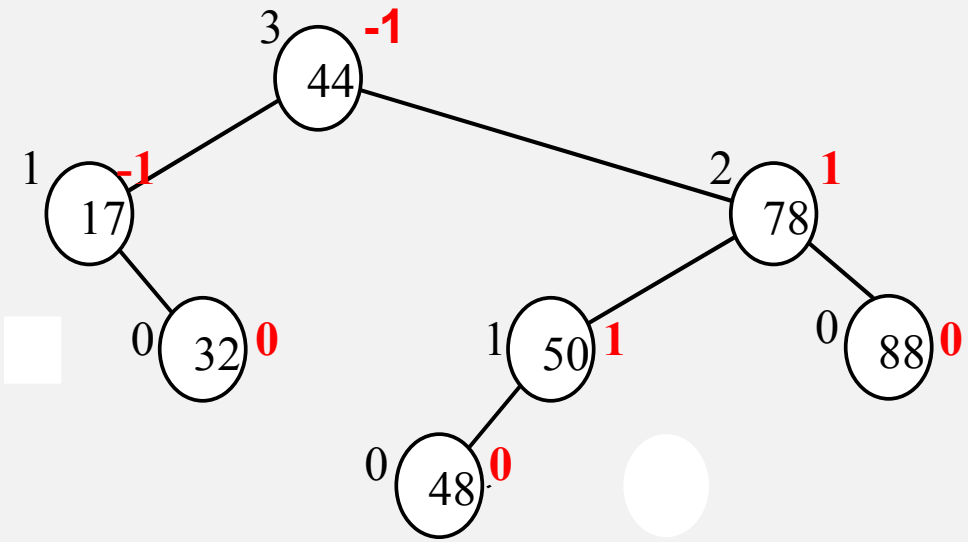
  Red-Black trees

  Splay trees

  (2,4) trees

AVL Trees

- AVL trees are height-balanced binary search trees such that for every internal node v of T, the heights of left and right subtrees of v can differ by at most 1
  - H = 1 + max of height(left subtree) and height(right subtree)



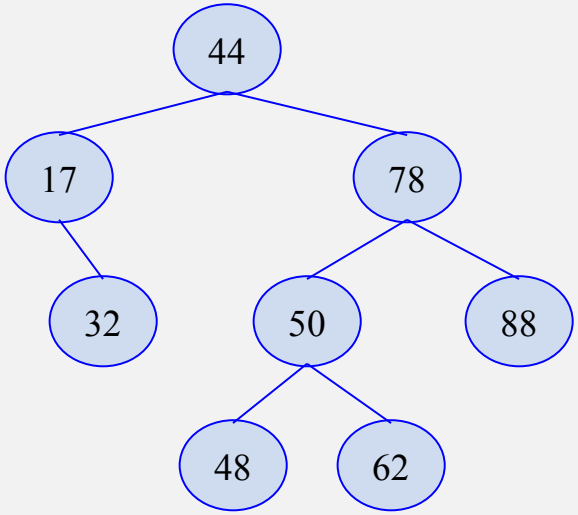
Treat an empty tree (nullptr) as height = -1

- **Balance factor** calculated at each node
  - BF = height(left subtree) − height(right subtree)

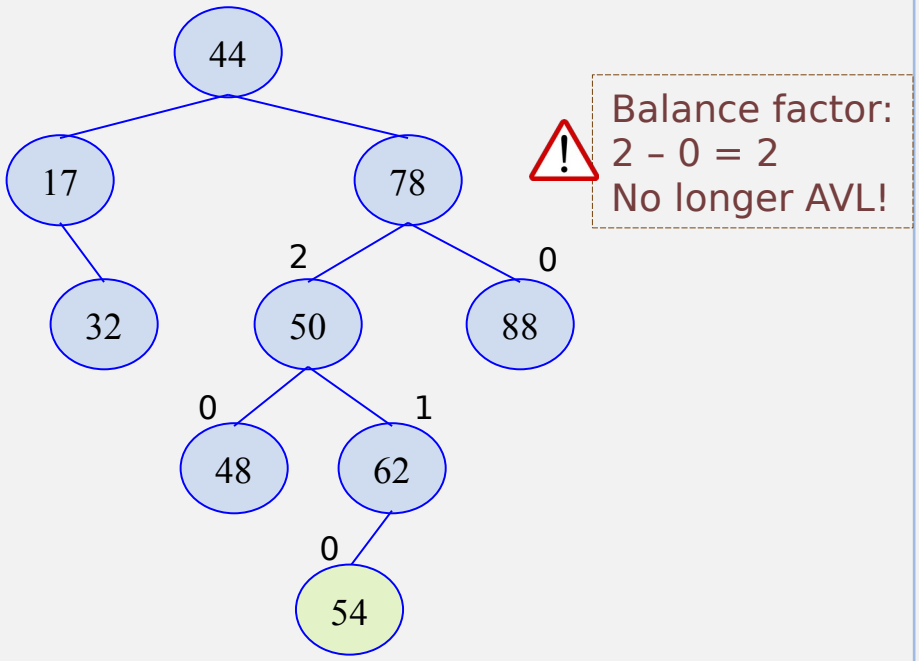
"good enough" balance but not perfectly balanced

Insertion

- Inserting a new key-value same as in a binary search tree
  - Always done by expanding an external node.
- Example: insert key 54



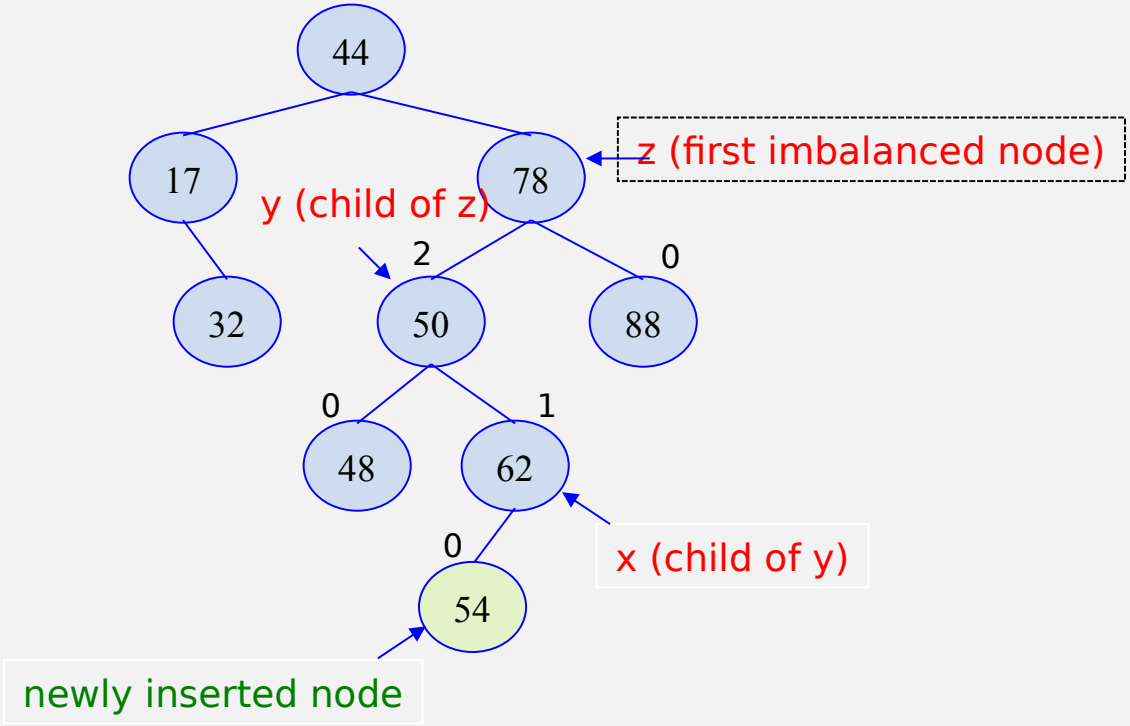
before insertion



after insertion

Trinode Restructuring

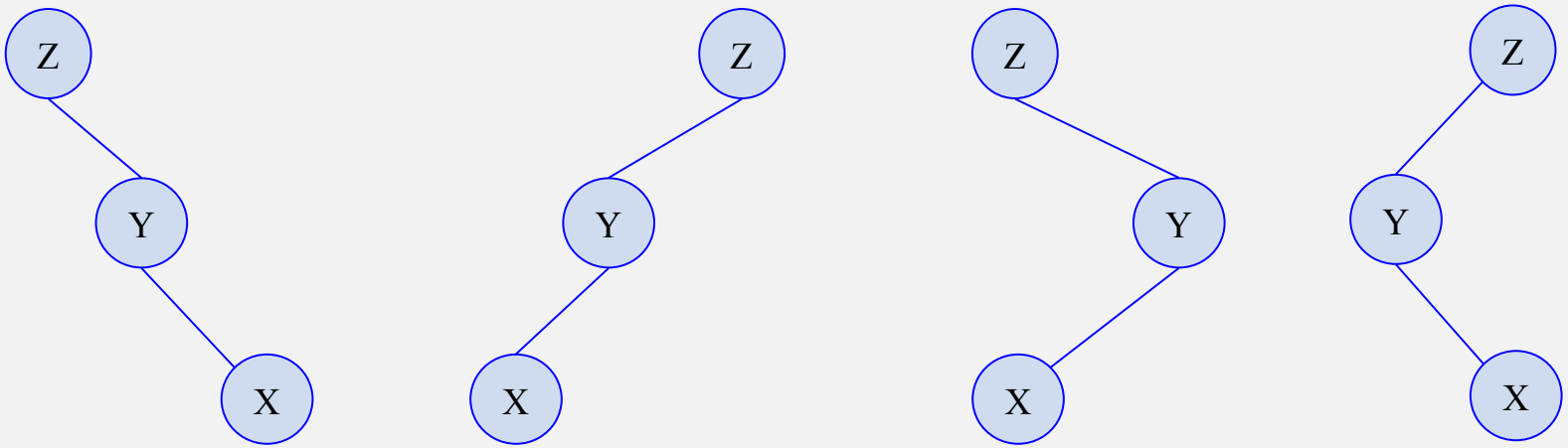
Focus on only three nodes



4 combinations: Y and X can be left/right child of Z

Trinode Restructuring

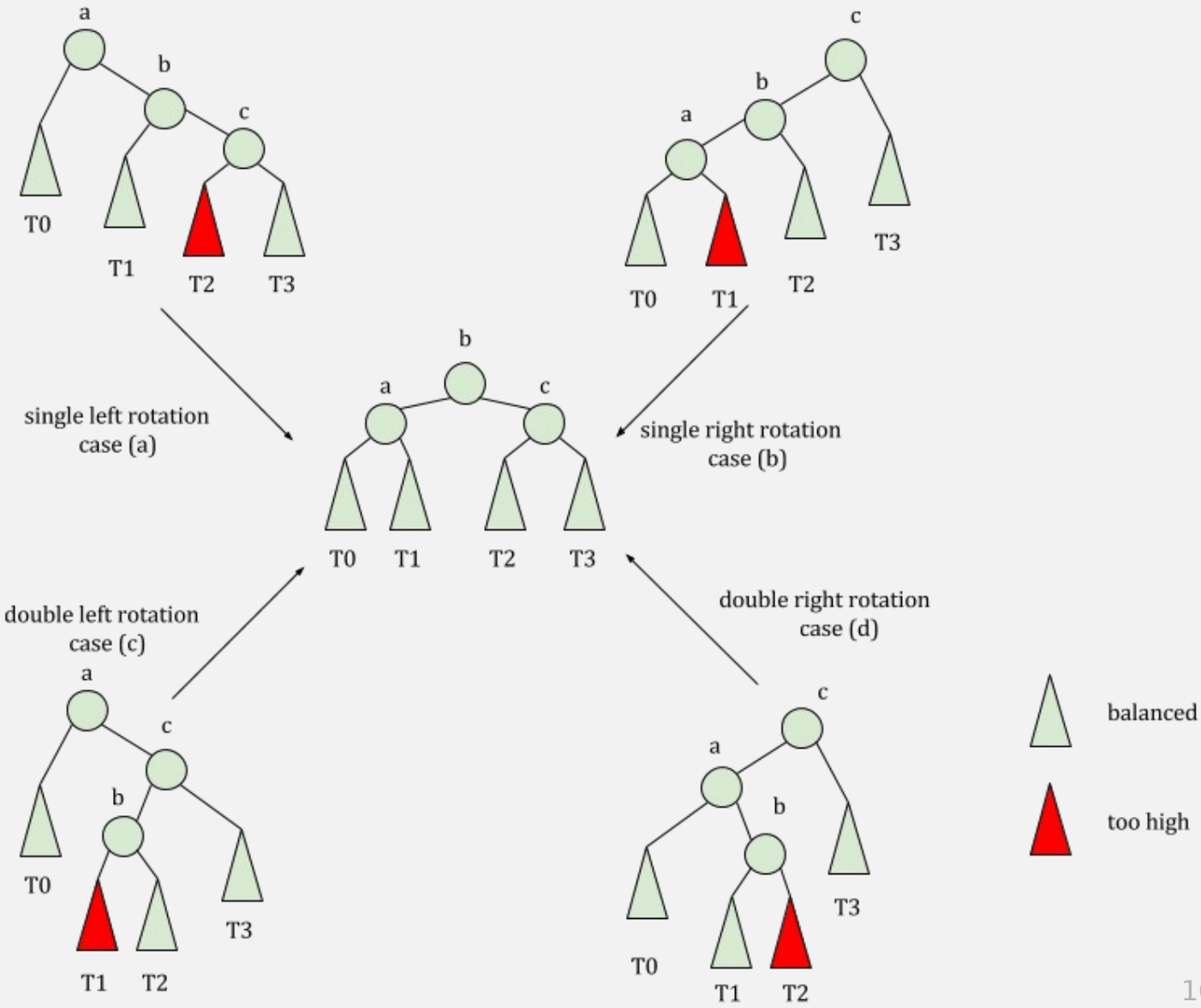
4 combinations: Y and X can be left/right child of Z



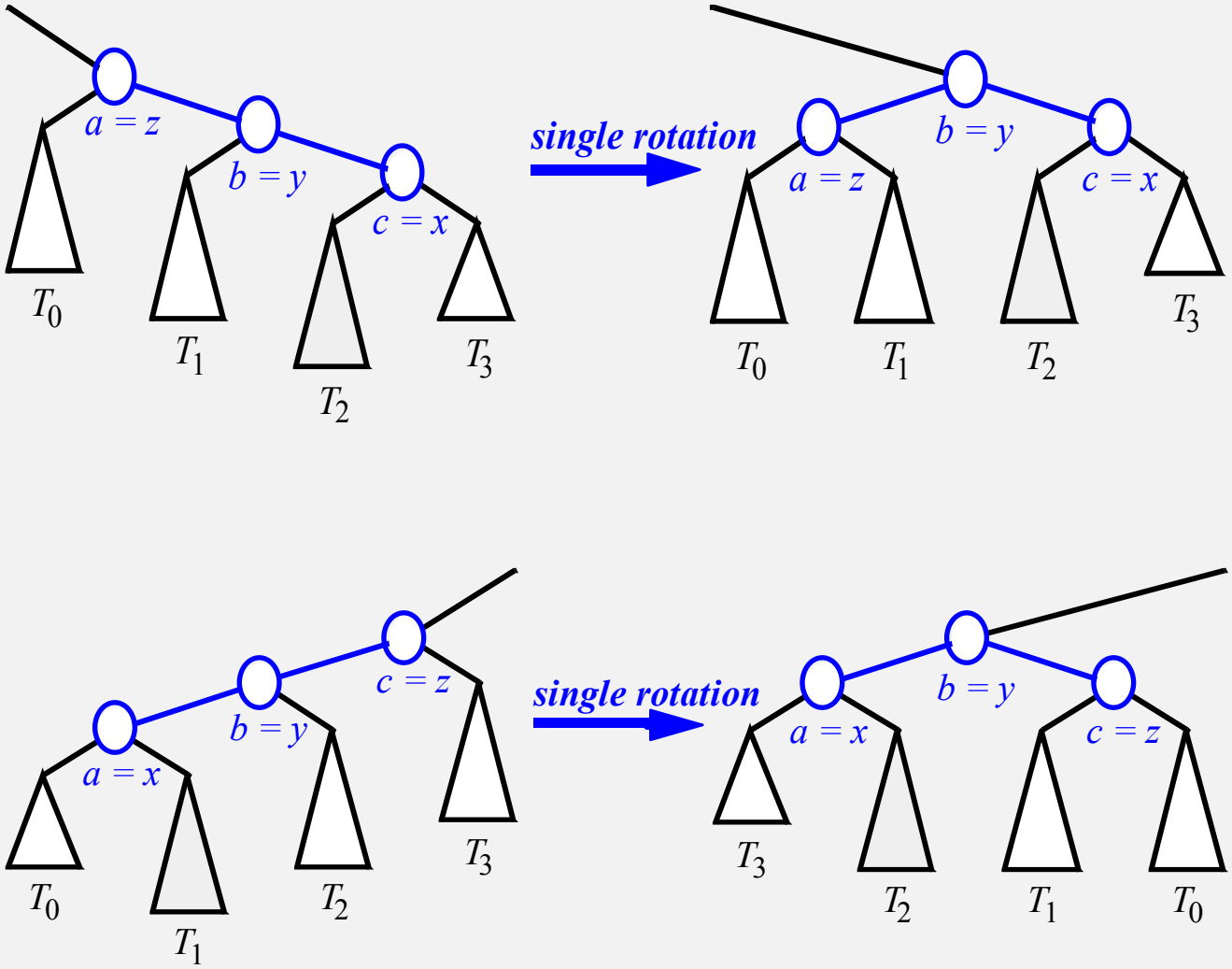
Insert with restructuring

1. Insert as usual for a BST
   At an external node
2. Check for imbalance starting from new external node, up to root
   If no imbalance, then done
3. Identify the three “problem nodes”
   First imbalanced node (call this z)
   “Tallest” child of x (call this y)
   “Tallest” child of y (call this x)
4. Refer to trinode restructuring handout and identify which rotation to do
   Single left rotation,
   Single right rotation,
   Double left rotation, or
   Double right rotation
5. Perform the rotation below z
   Keep the rest of the tree as it is

AVL Tree Trinode Restructuring

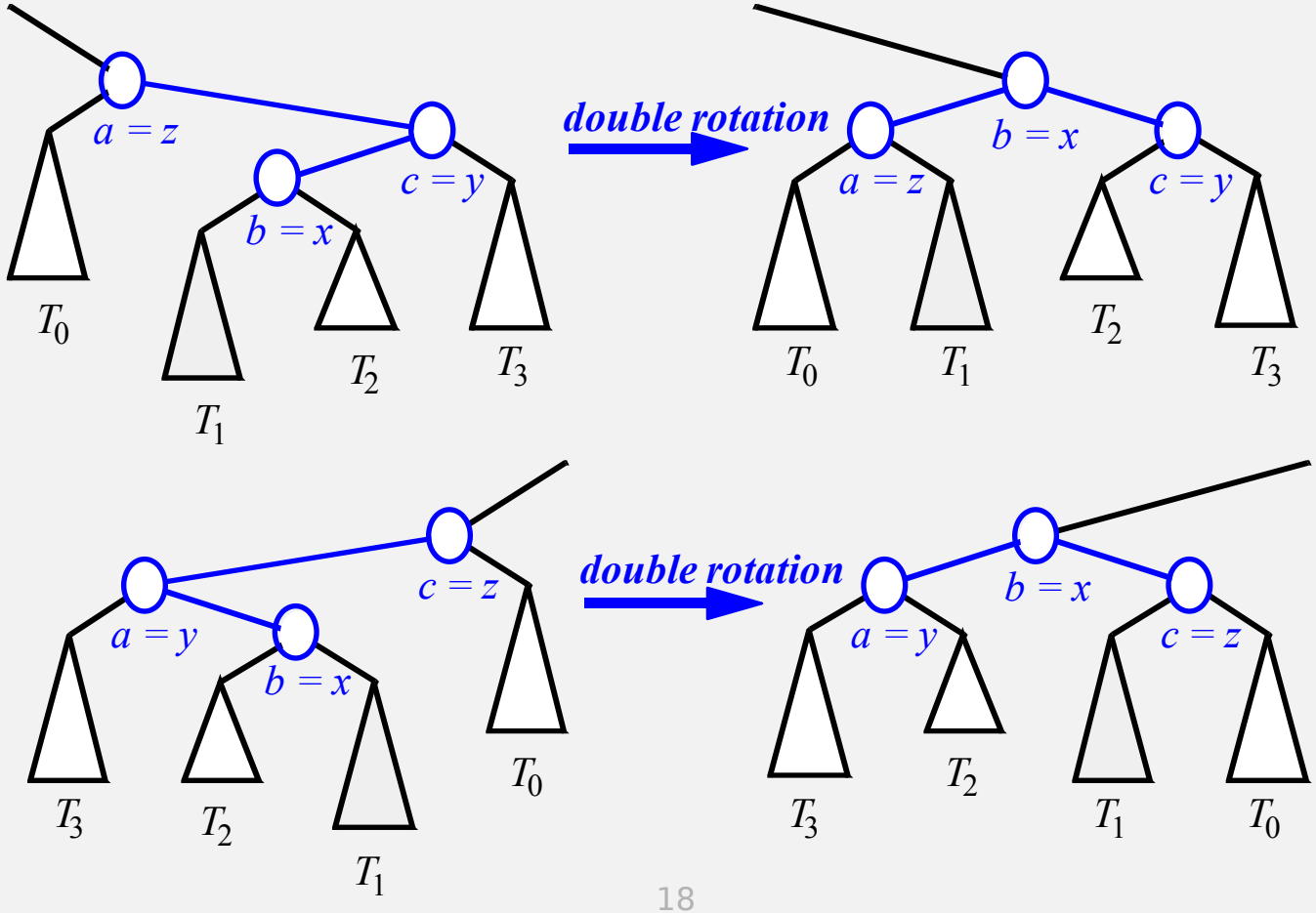


Restructuring – as Single Rotations



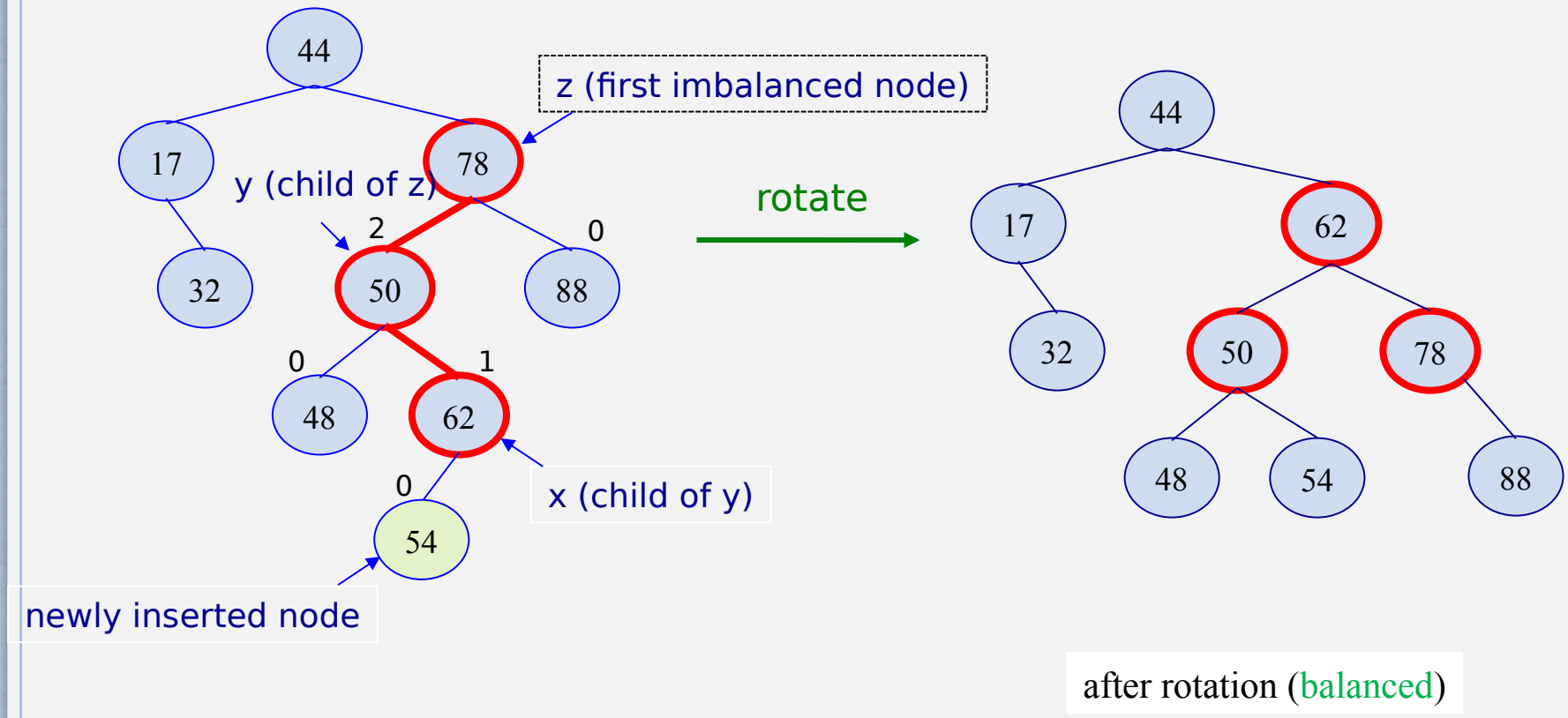
Restructuring – as Double Rotations

zyBook Annimations



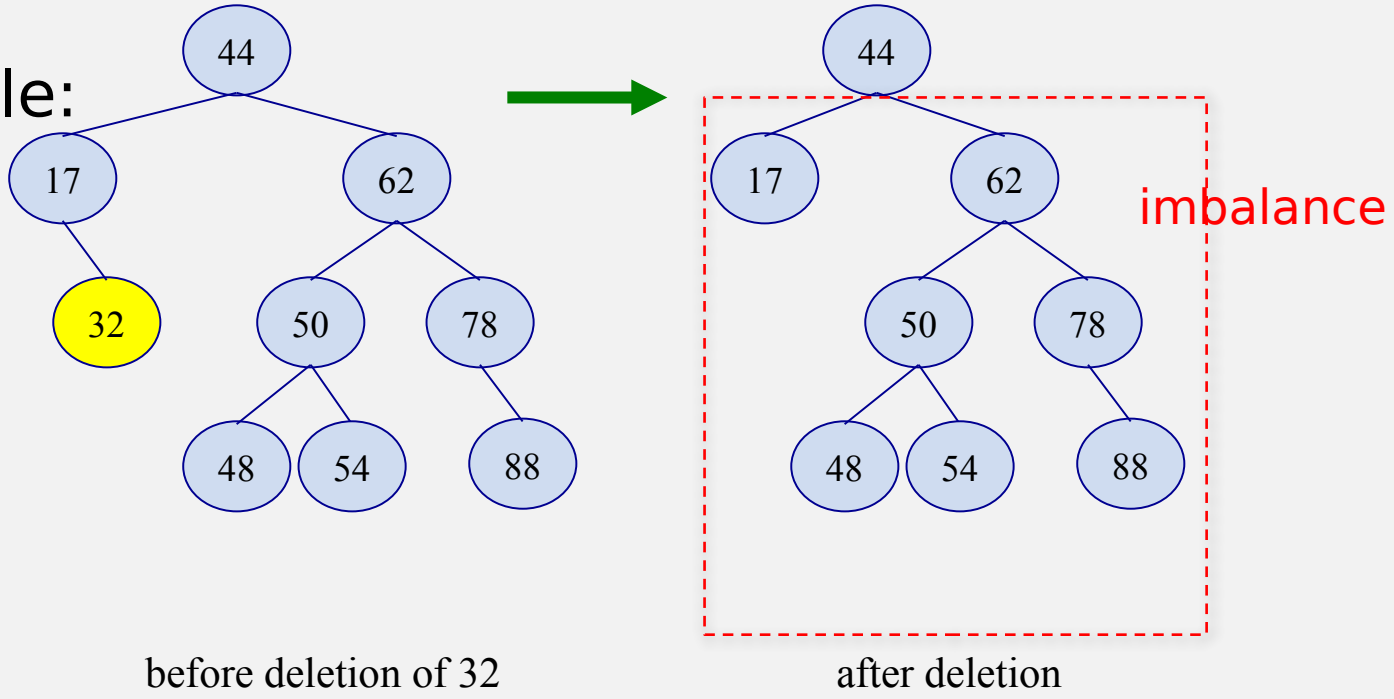
Trinode Restructuring (cont)

Focus on only three nodes



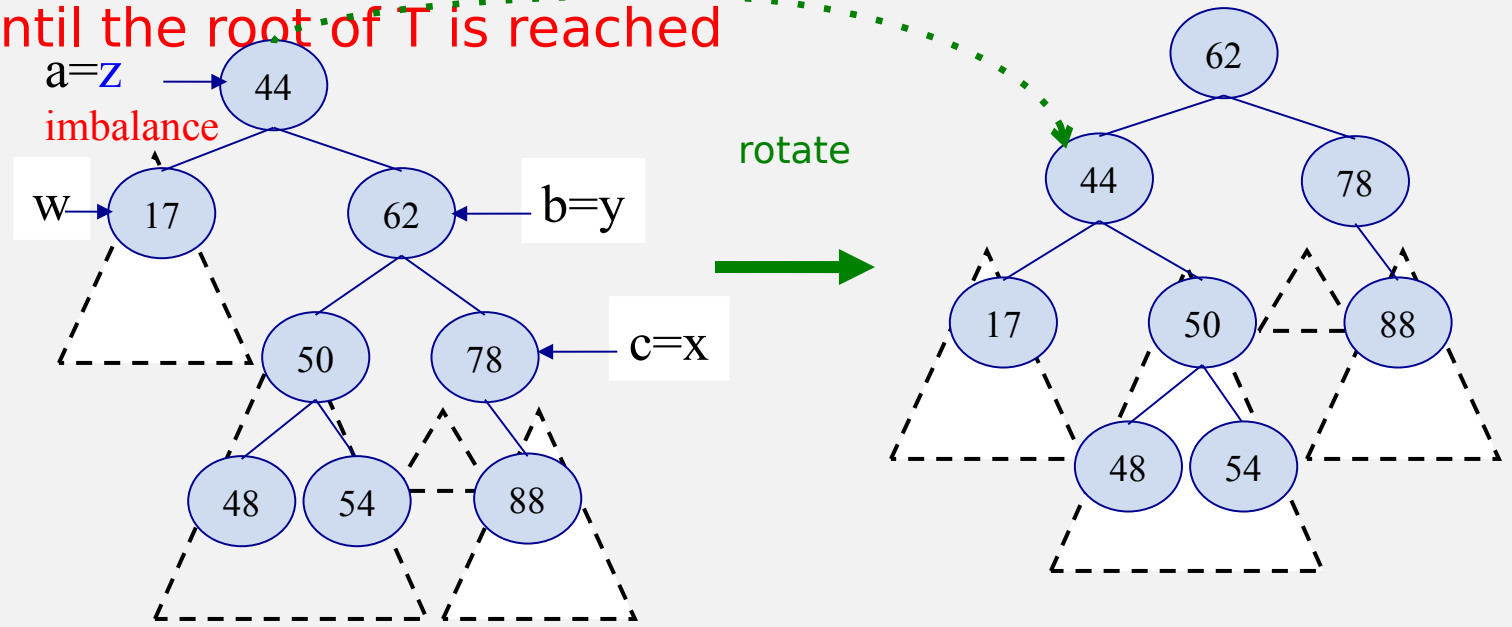
Removal

- Removal 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 Removal

- 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
- We perform restructure(x) to restore balance at z
- If this restructuring upsets the balance of another node higher in the tree, we would continue checking for balance until the root of T is reached



Time complexity of insert

Insert operations	BST	AVL
Insert as external node	O(height)	O(height)
Check for balance (at every node from new node up to root)	-	O(height)
Trinode restructuring	-	O(1)
Total in worst-case	O(height) = **O(n)**	O(height) = **O(log *n*)**

Height of an AVL Tree

- If an AVL tree contains n nodes (i.e., n key-value pairs), what is its height?

- Can show that:

  The maximum height of an AVL tree storing n keys is 2*log(n) = O(log n).

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)