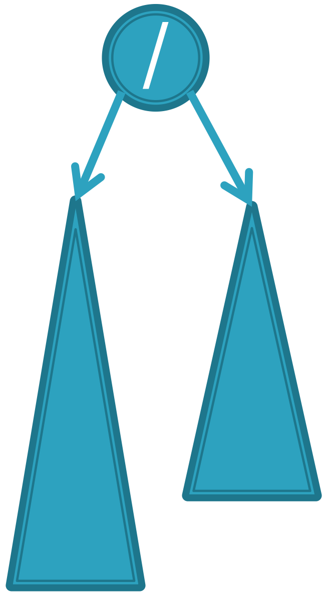


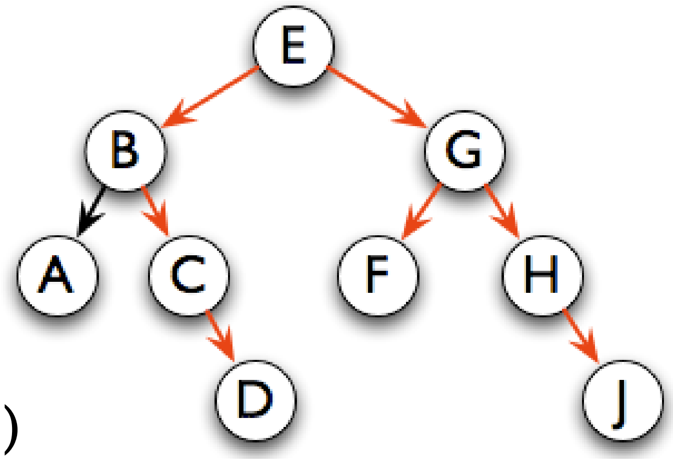
CSSE 230 Day 14

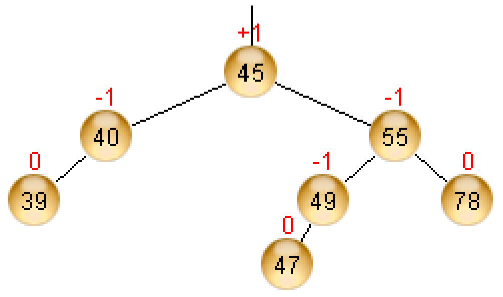
AVL trees and rotations

Summary: for fast tree operations, we must keep the tree somewhat balanced in O(log n) time

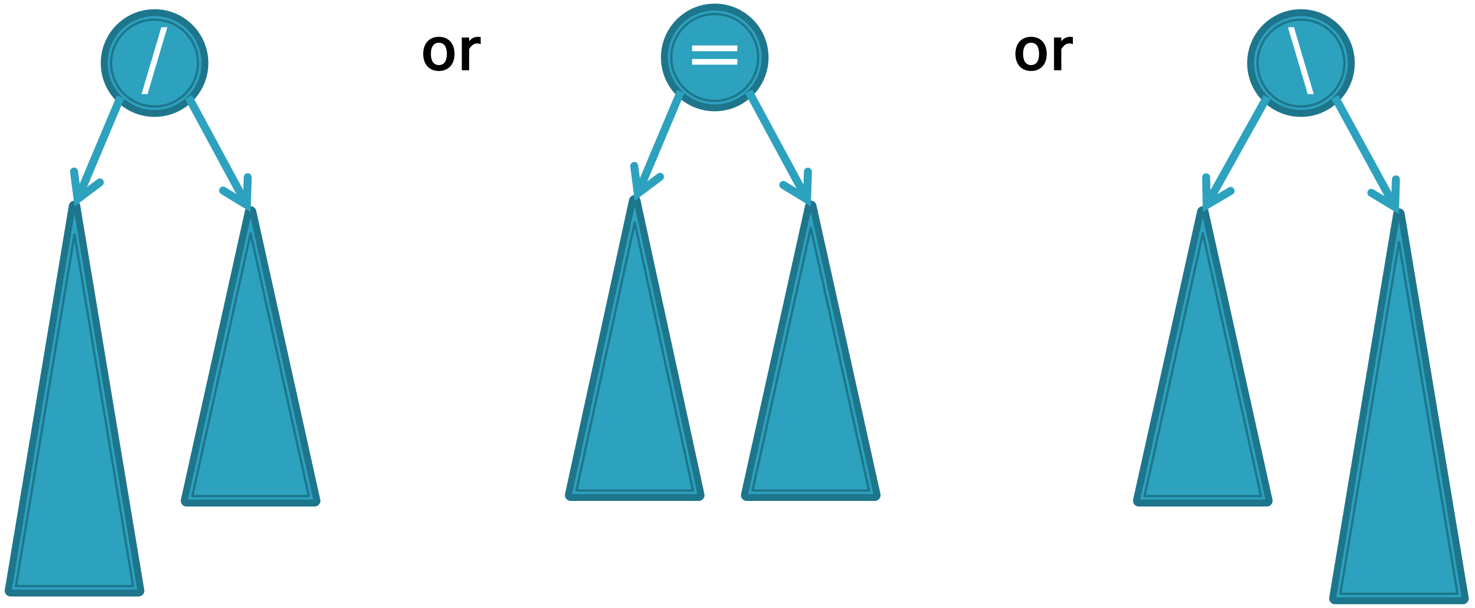
Q1

- Operations (insert, delete, search) are O(height)
- Tree height is O(log n) if perfectly balanced
  - But maintaining perfect balance is O(n)
- Height-balanced trees are still O(log n)
  - For T with height h, N(T) ≤ Fib(h+3) – 1
  - So H < 1.44 log (N+2) – 1.328 *
- AVL (Adelson-Velskii and Landis) trees maintain height-balance using rotations
- Are rotations O(log n)? We'll see…





AVL nodes are just like BinaryNodes, but also have an extra “balance code”

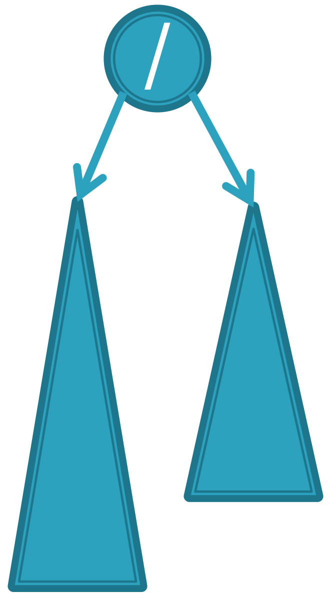


Different representations for / = \ :

- Just two bits in a low-level language
- Enum in a higher-level language

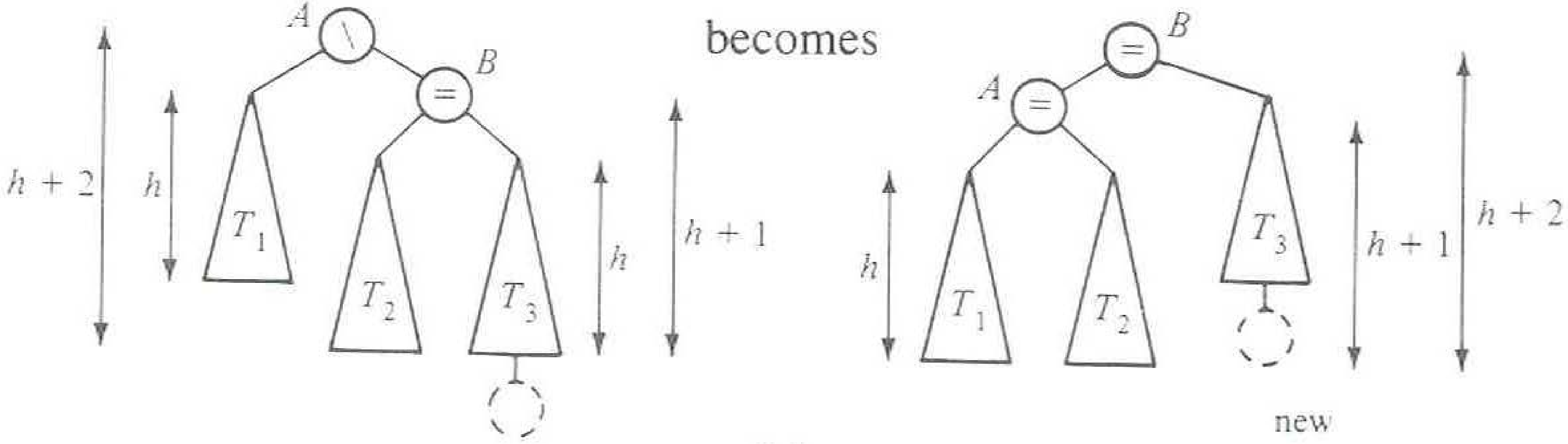
AVL Tree (Re)balancing Act

- Assume tree is height-balanced before insertion
- Insert as usual for a BST
- Move up from the newly inserted node to the lowest "unbalanced" node (if any)
    - Use the **balance code** to detect unbalance – how?
- Do an appropriate rotation to balance the sub-tree rooted at this unbalanced node



Four types of rotations are required to remove different cases of tree imbalances

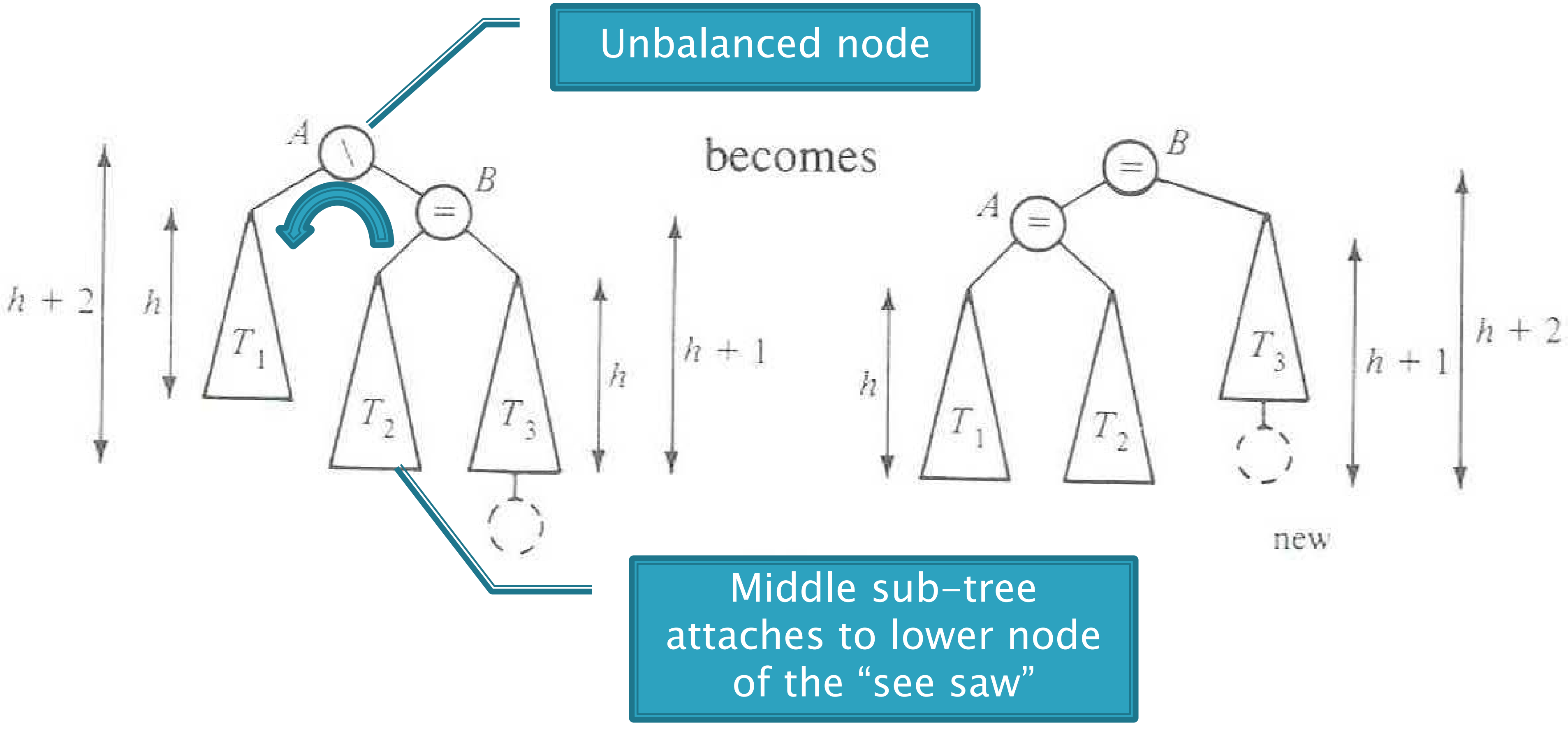
- For example, a *single left rotation*:



We rotate by pulling the “too tall” sub–tree up and pushing the “too short” sub–tree down

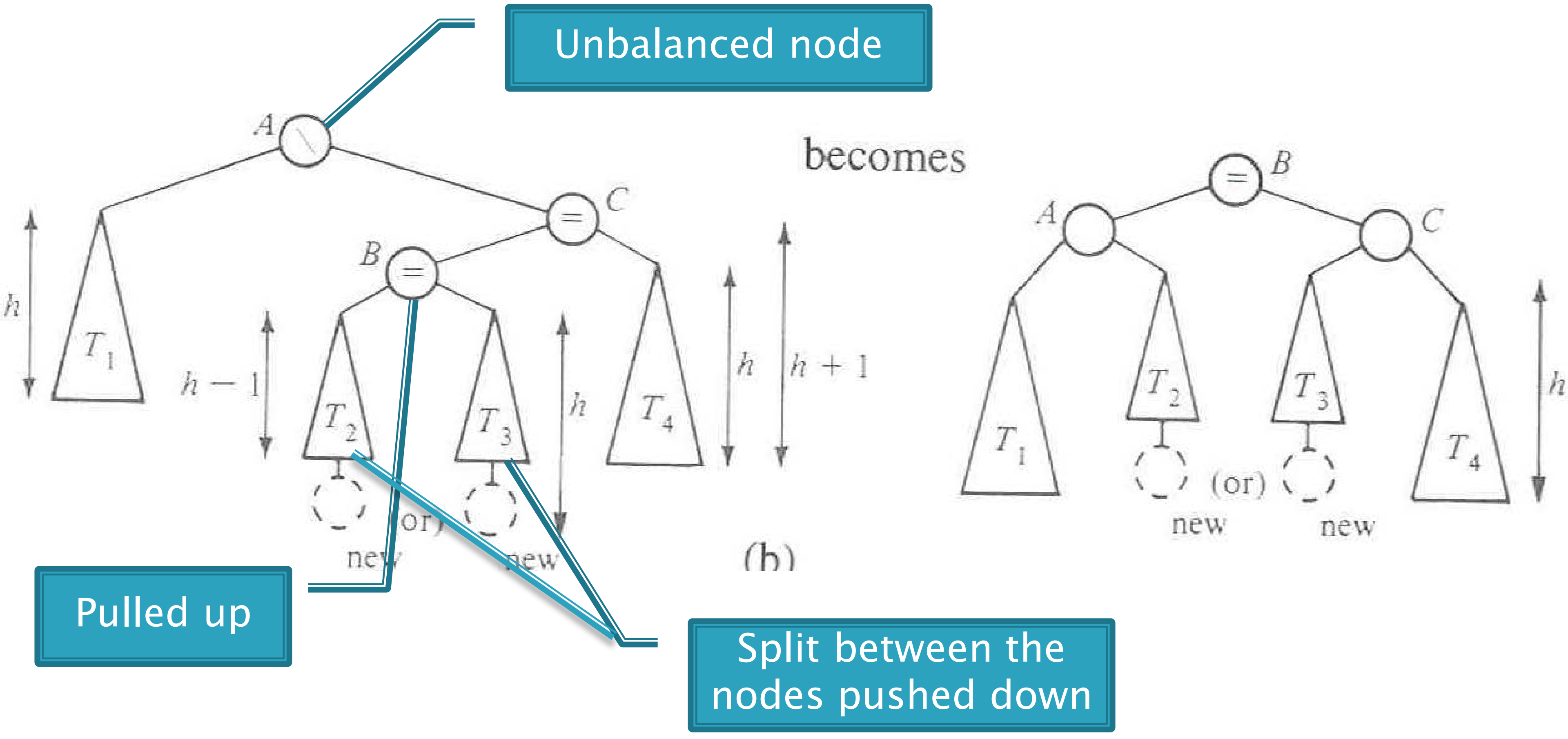
- Two basic cases
  - “See saw” case:
    - Too–tall sub–tree is on the outside
    - So tip the see saw so it’s level
  - “Suck in your gut” case:
    - Too–tall sub–tree is in the middle
    - Pull its root up a level

Single Left Rotation



Diagrams are from *Data Structures* by E.M. Reingold and W.J. Hansen

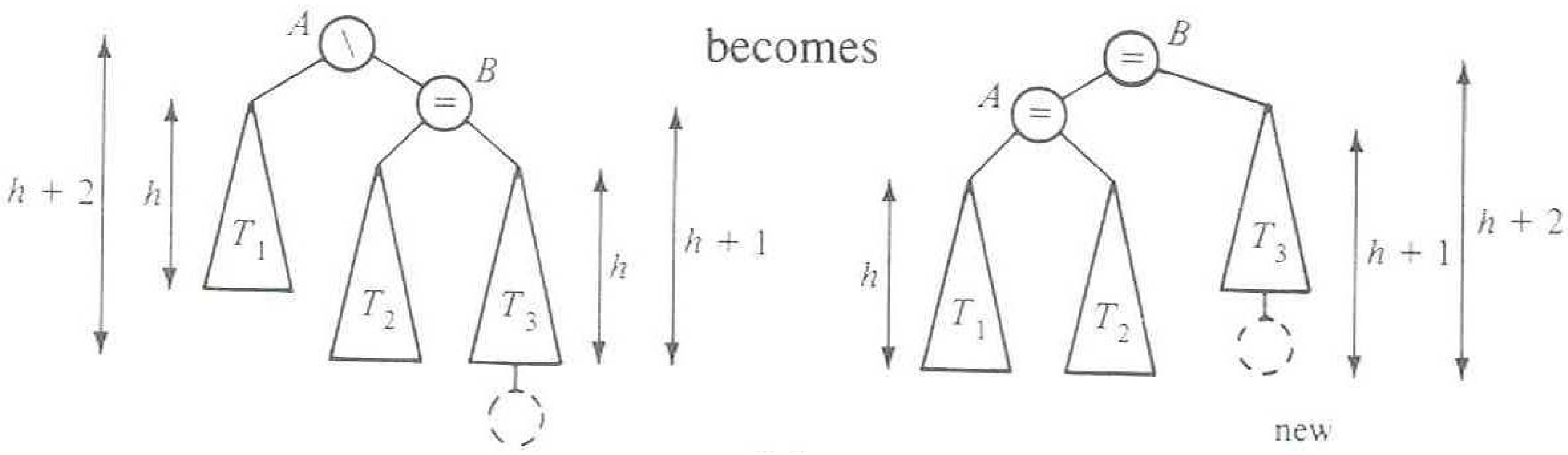
Double Left Rotation



Weiss calls this "right-left double rotation"

Your turn — work with a partner

Q6



- Write the method:

-
  ```
  static BalancedBinaryNode singleRotateLeft (
      BalancedBinaryNode parent,    /* A */
      BalancedBinaryNode child      /* B */  ) {

  }
  ```
- Returns a reference to the new root of this subtree.
- Don't forget to set the balanceCode fields of the nodes.

More practice—(sometime after class)

- Write the method:
-
```
BalancedBinaryNode doubleRotateRight (
    BalancedBinaryNode parent,     /* A */
    BalancedBinaryNode child,      /* C */
    BalancedBinaryNode grandChild  /* B */ ) {


}
```
- Returns a reference to the new root of this subtree.
- Rotation is mirror image of double rotation from an earlier slide

O(log N)?

- Both kinds of rotation leave height the same as before the insertion!
- Is insertion plus rotation cost really O(log N)?

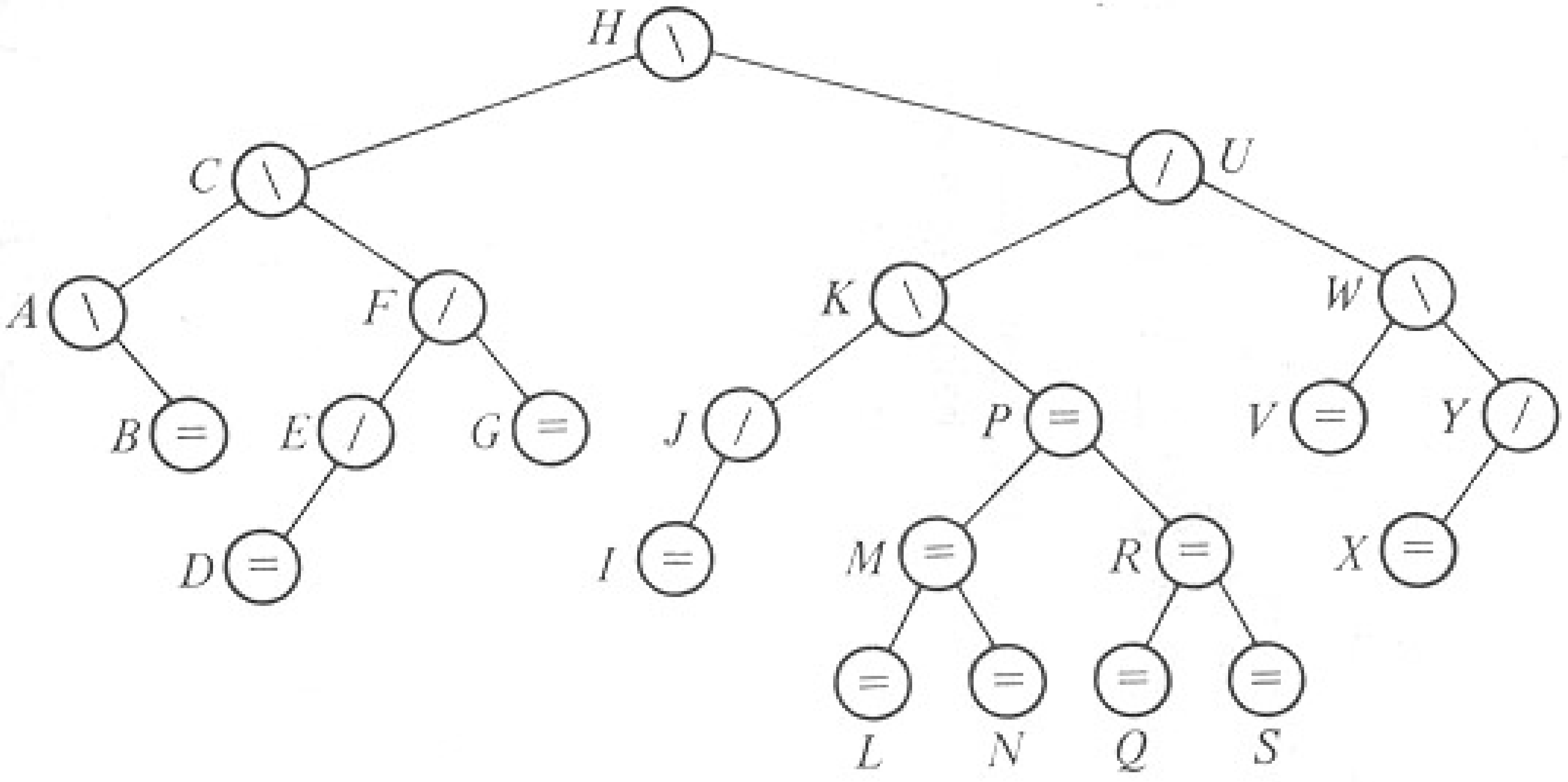
Insertion/deletion in AVL Tree:	O(log n)
Find the imbalance point (if any):	O(log n)
Single or double rotation:	O(1)
in deletion case, may have to do O(log N) rotations	
Total work:	O(log n)

Which kind of rotation to do after an insertion?

Depends on the first two links in the path from the lowest node that has the imbalance (A) down to the newly-inserted node.

First link (down from A)	**Second link** (down from A's child)	**Rotation type** (rotate "around A's position")
Left	Left	Single right
Left	Right	Double right
Right	Right	Single left
Right	Left	Double left

A sample AVL tree

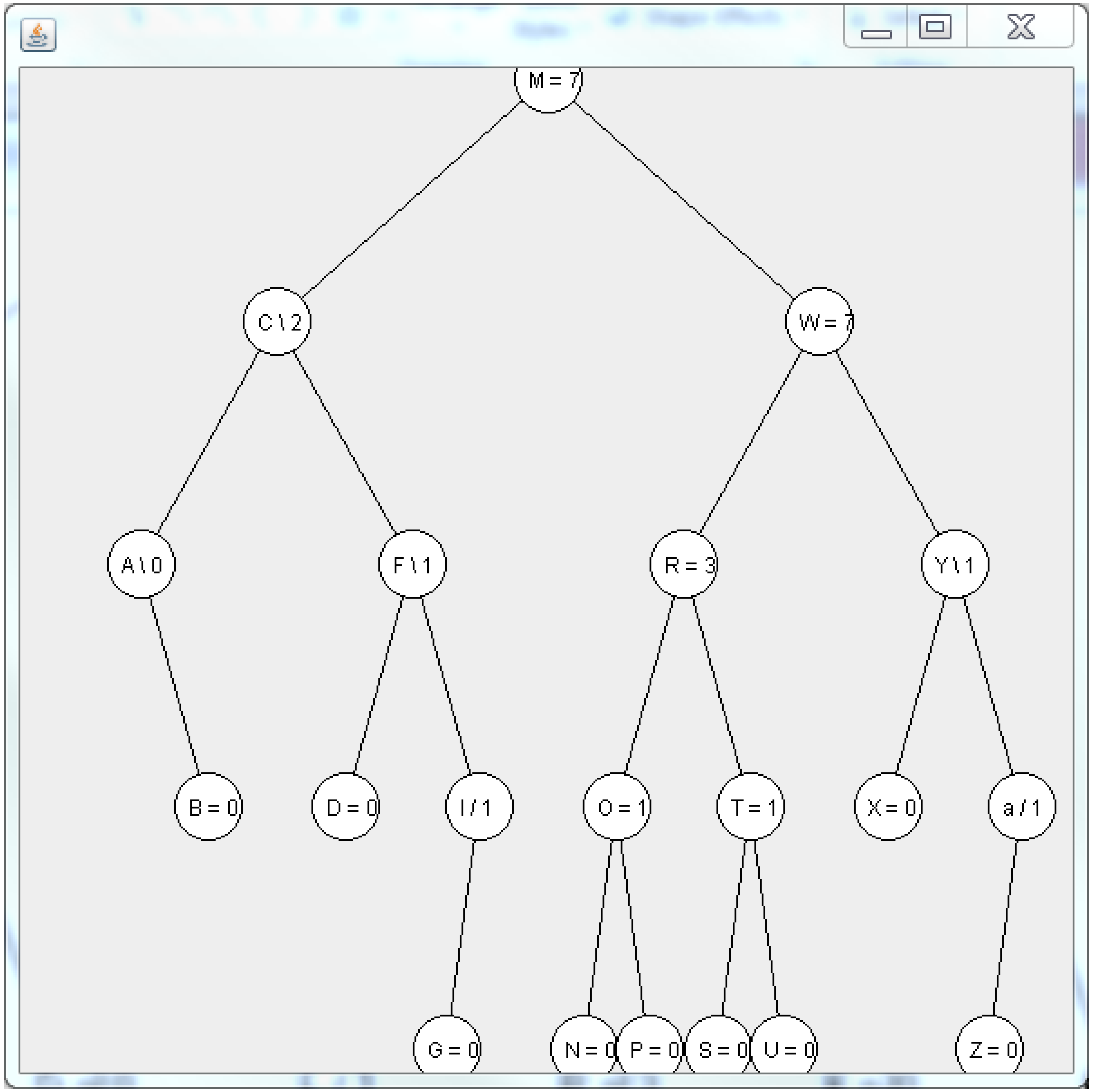


Insert **HA** into the tree, then **DA**, then **O**.
Delete **G** from the original tree, then **I**, **J**, **V**.

Your turn again

- Start with an empty AVL tree.
- Add elements in the following order; do the appropriate rotations when needed.
  - 1 2 3 4 5 6 11 13 12 10 9 8 7
- How should we rebalance if each of the following sequences is deleted from the above tree?
  - (10 9 7 8) (13) (1 5)
  - For each of the three sequences, start with the original 13-element tree. E.g. when deleting 13, assume 10 9 8 7 are still in the tree.

Work with your Doublets partner.
When you finish, work on Doublets or WA5.
Or write the rotateDoubleRight code from a previous slide


M = 7
C \ 2
W = 7
A \ 0
F \ 1
R = 3
Y \ 1
B = 0
D = 0
I / 1
O = 1
T = 1
X = 0
a / 1
G = 0
N = 0
P = 0
S = 0
U = 0
Z = 0