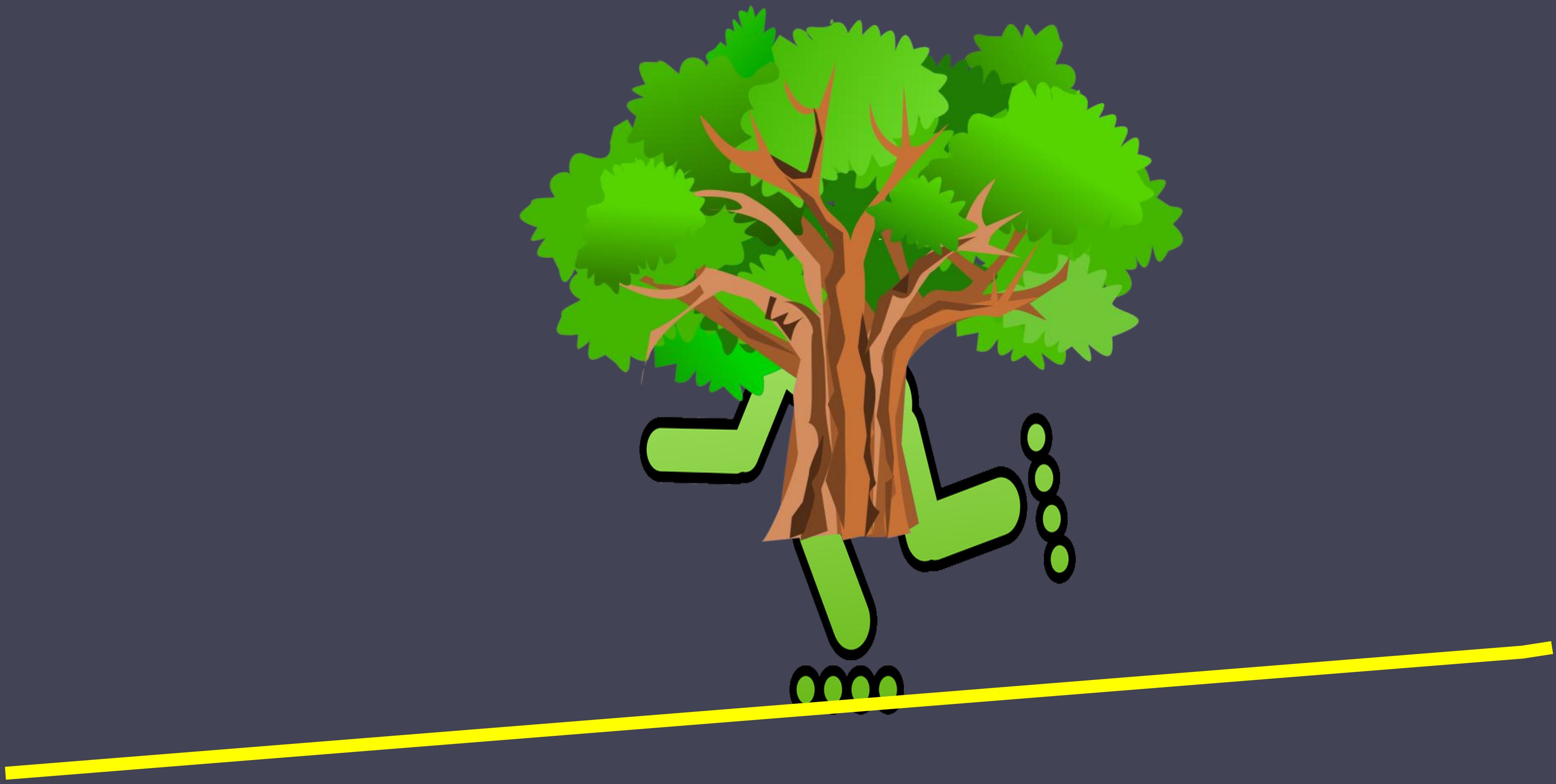


AVL TREES

Lecture 36

Assignment

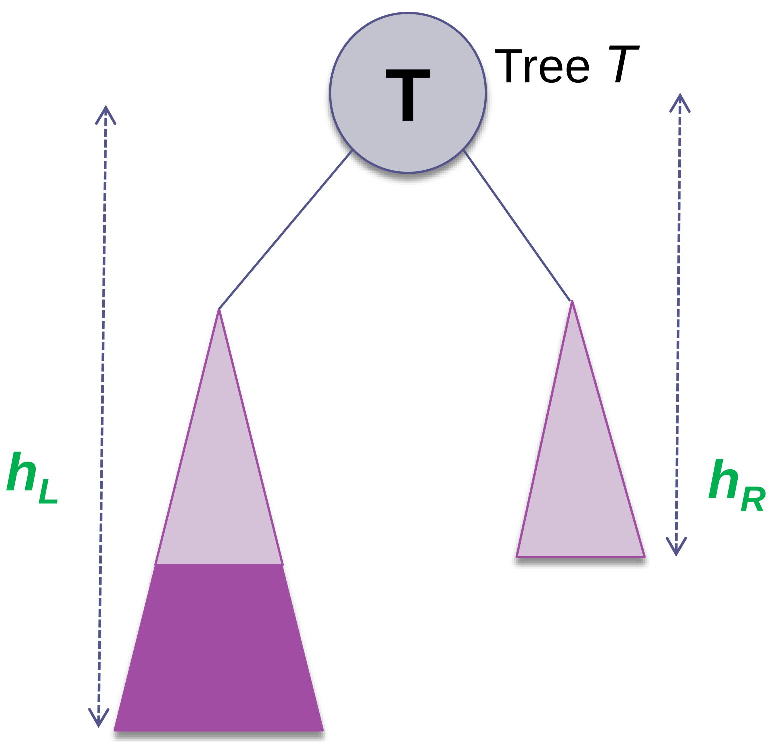
- Do self-check exercises for section 9.1

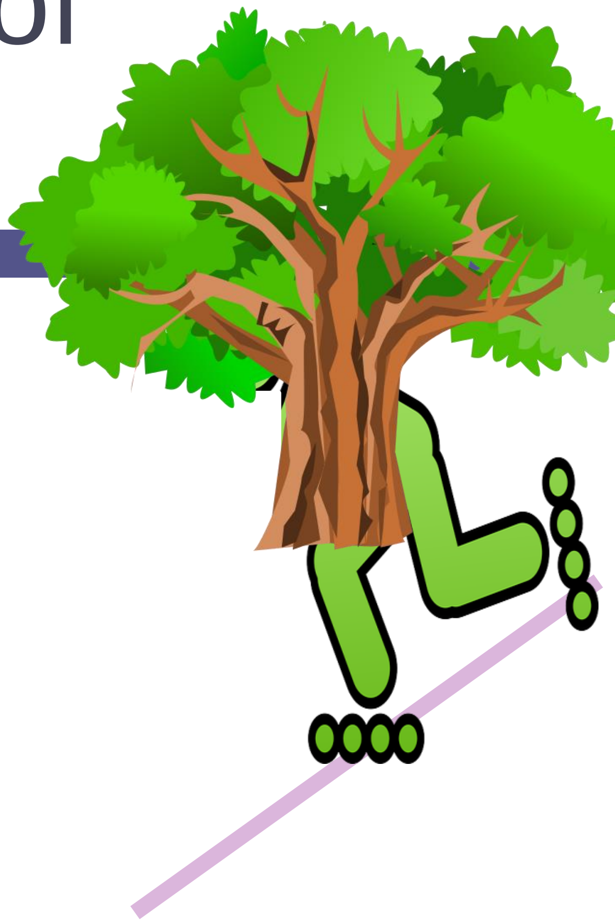
(Most important, ex. 2: write an algorithm for rotation to the left!)

- Again, ensure you can repeat what we have discussed *without* looking at the notes or in the book

The *balance factor* of a tree

$$b(T) = h_R(T) - h_L(T)$$





The balance *b(N)* is simliarly defined for each node *N* based on its subtree

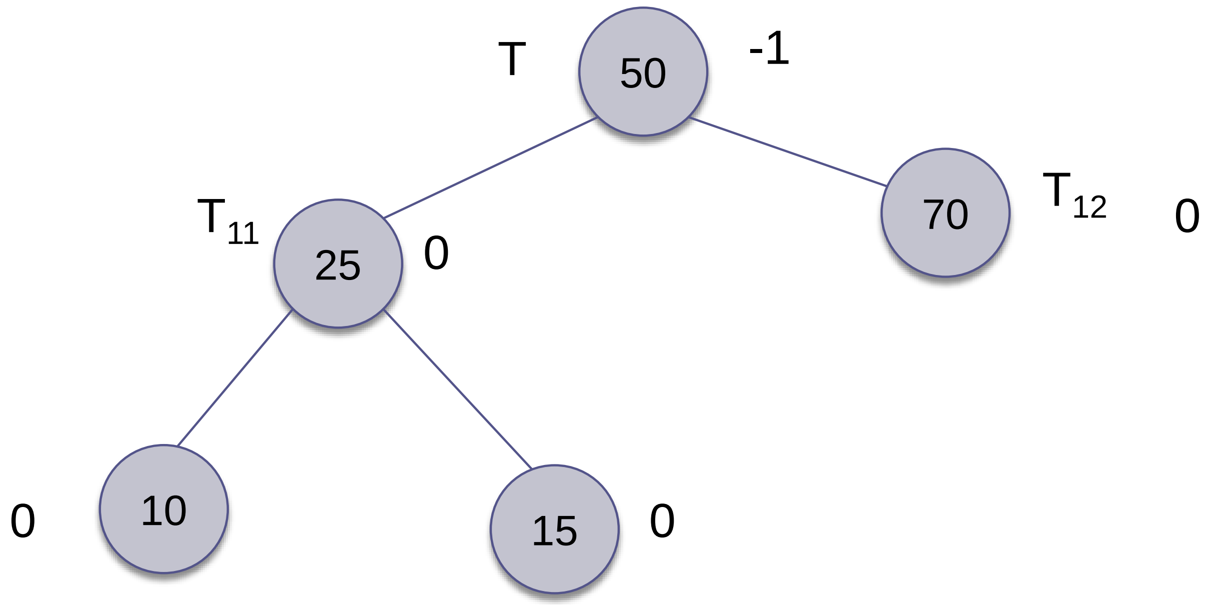
Definition of a balanced tree

A binary tree *T* is balanced if the absolute value of the *balance factor* of each of its subtrees is less than 2 (i.e, the balance factor is either 0, -1 or +1:

$|b(T)| = |h_R(T) - h_L(T)| < 2,$

for each node T

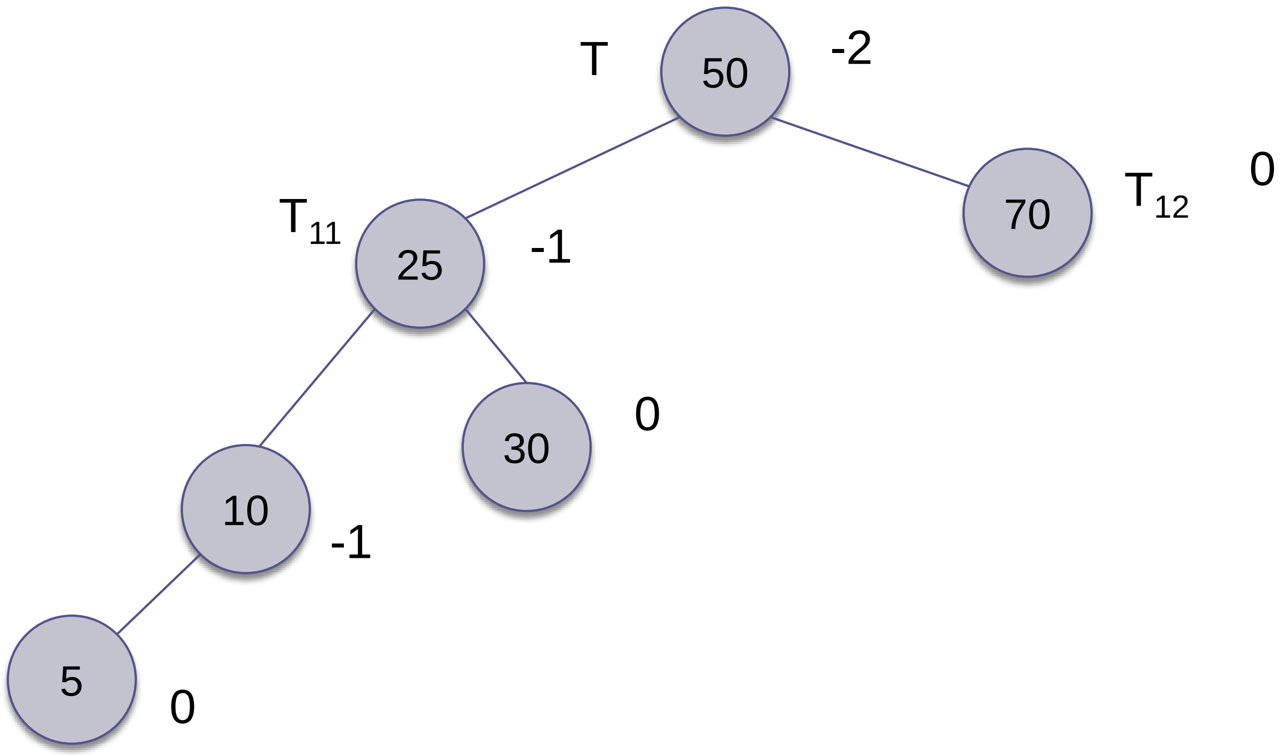
An example



b(T) = 1 – 2 = -1;

b(T_{11}) = 1 – 1 = 0;

Another example



$b(T) = 1 – 3 = -2;$

$b(T_{11}) = 1 – 2 = -1;$

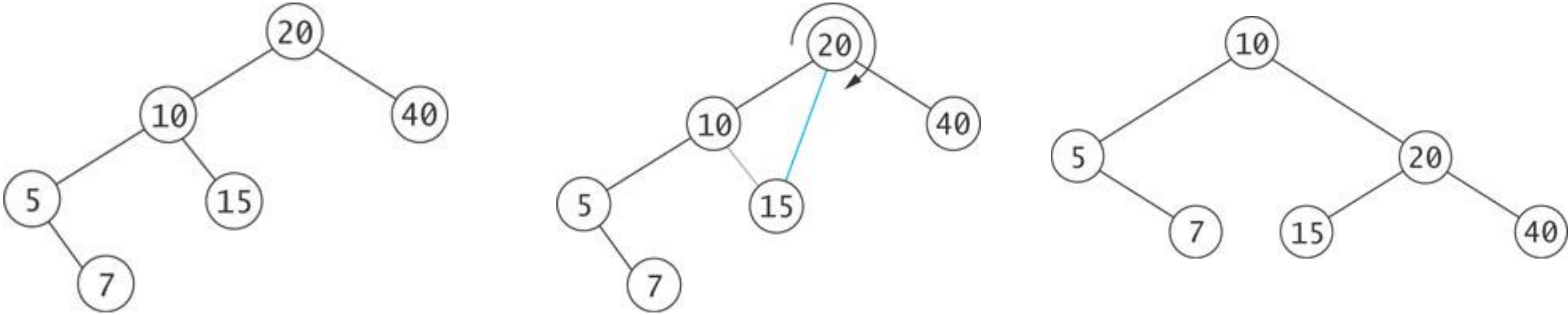
When does the balance factor of a tree change?

1. On insertion
2. On deletion
3. On rotation

Hence the strategy: **To keep a tree balanced, *do something* the moment it becomes unbalanced** (i.e., the moment there is a subtree with the balance factor of 2)

Right rotation (from the previous lecture)

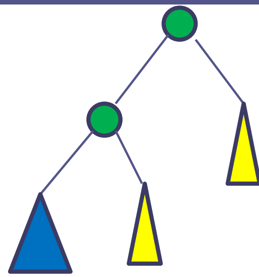
We need an operation on a binary tree that changes the relative heights of left and right subtrees, but preserves the binary search tree property



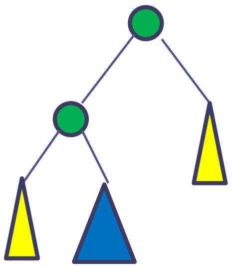
Again, an exercise: how do we rotate to the left?

The taxonomy of unbalanced trees

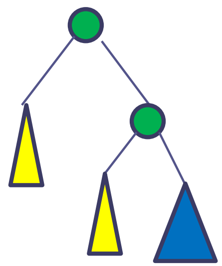
- Left-Left Tree
  - Root's balance factor is *-2*
  - Left child's balance factor is *-1*



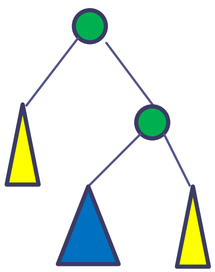
- Left-Right Tree
  - Root's balance factor is *-2*
  - Left child's balance factor is +*1*



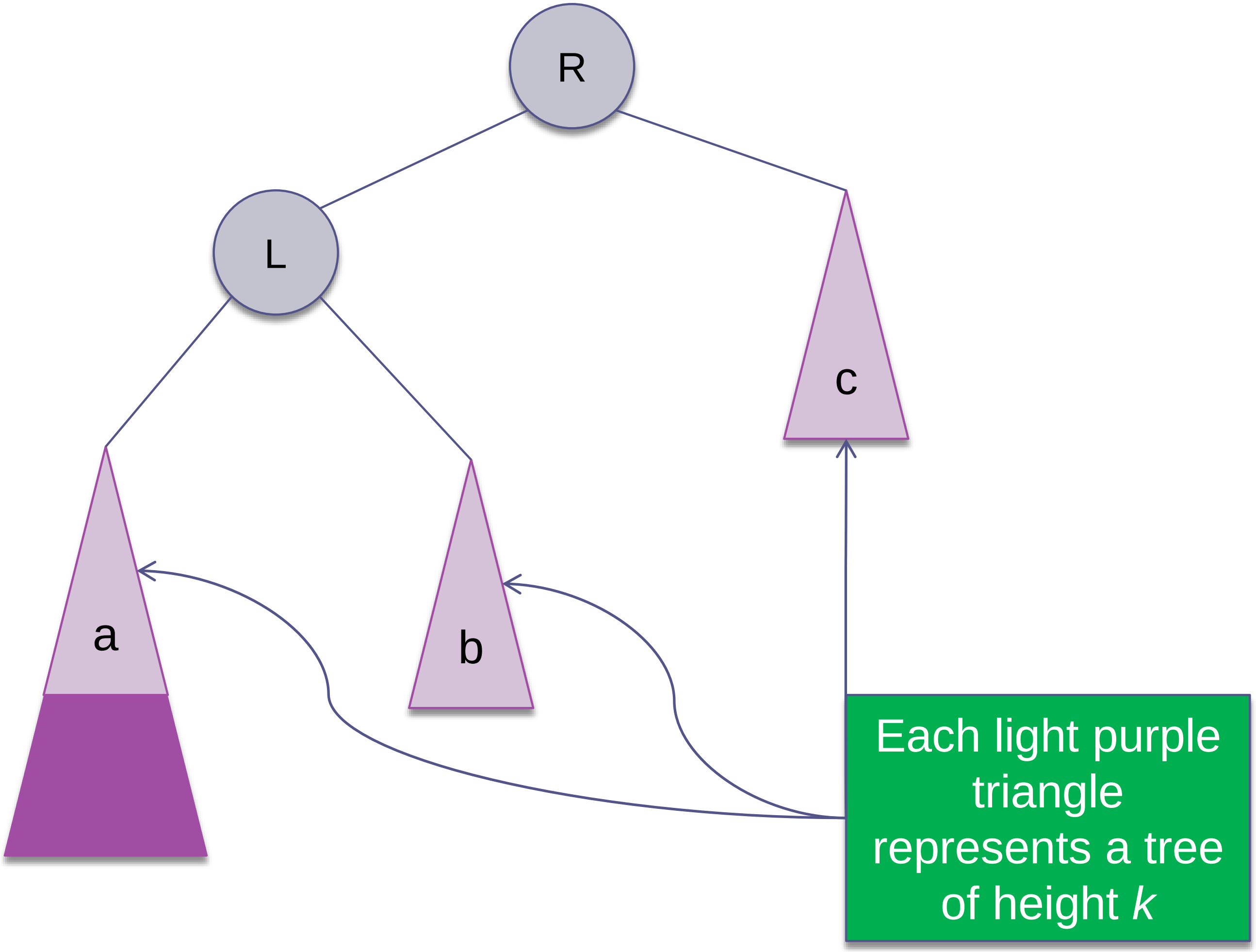
- Right-Right Tree
  - Root's balance factor is +*2*
  - Right child's balance factor is +*1*



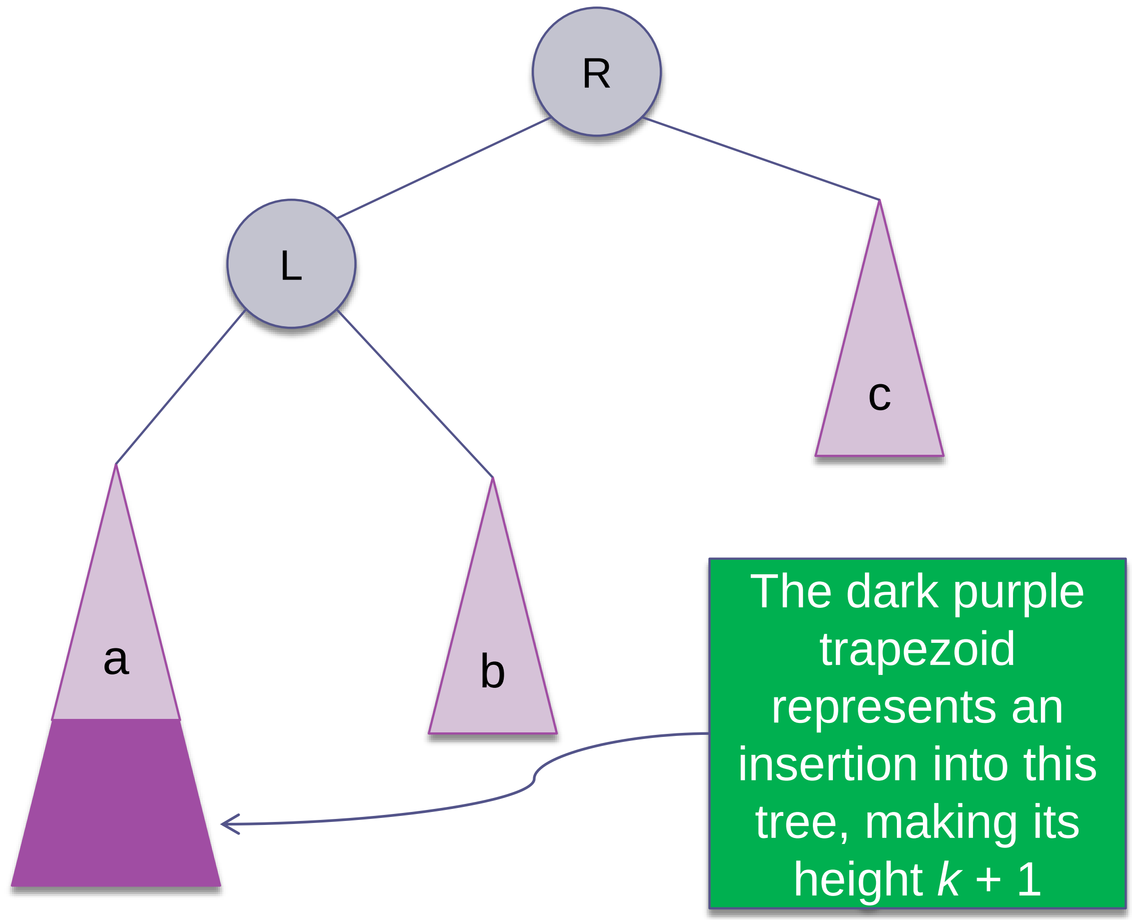
- Right-Left Tree
  - Root's balance factor is +*2*
  - Right child's balance factor is *-1*



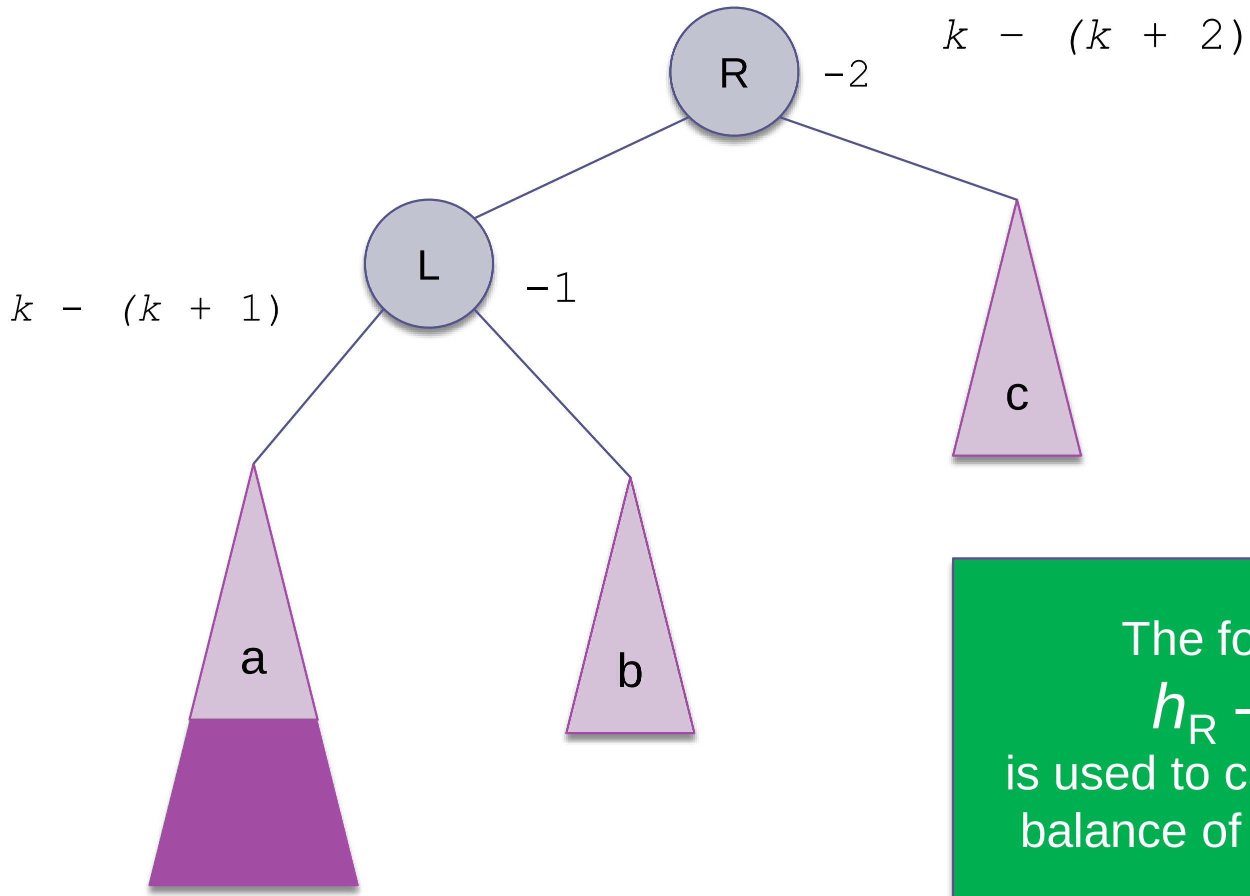
Balancing a Left-Left Tree



Balancing a Left-Left Tree (cont.)



Balancing a Left-Left Tree (cont.)

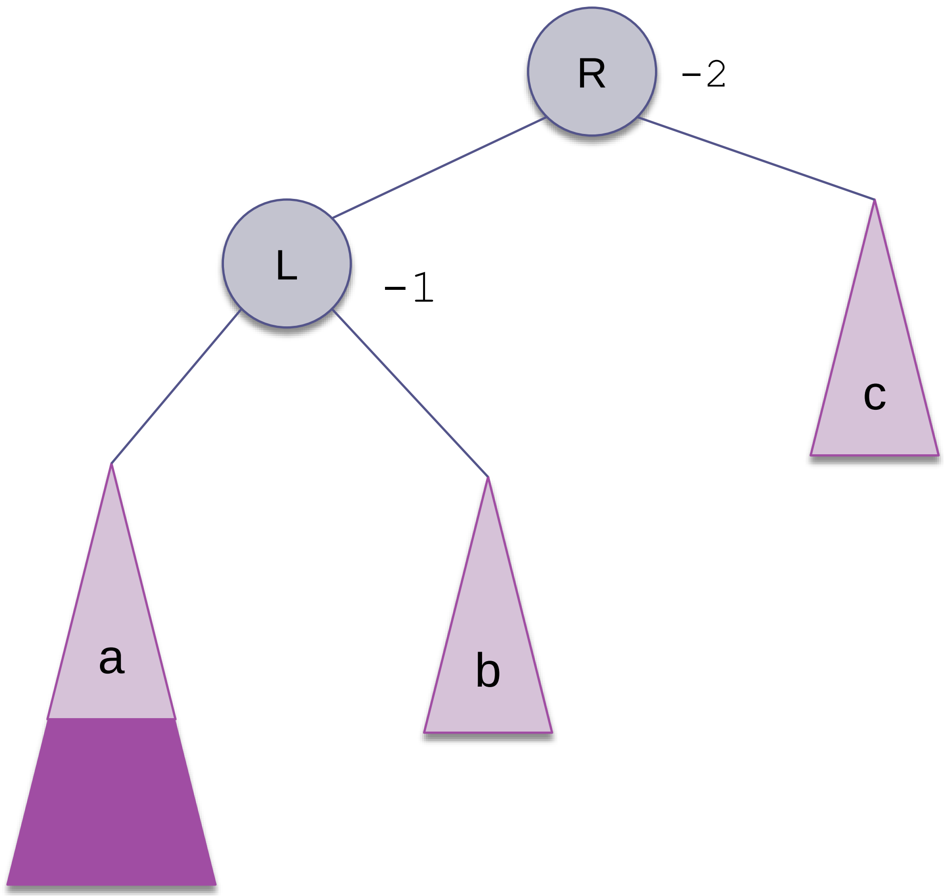


The formula

$h_R - h_L$

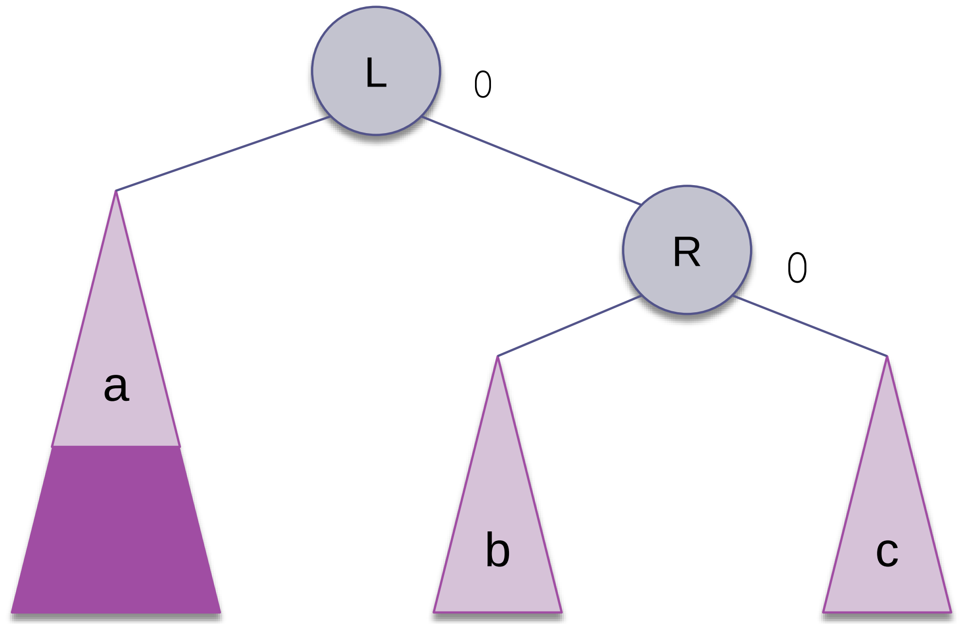
is used to calculate the balance of each node

Balancing a Left-Left Tree (cont.)



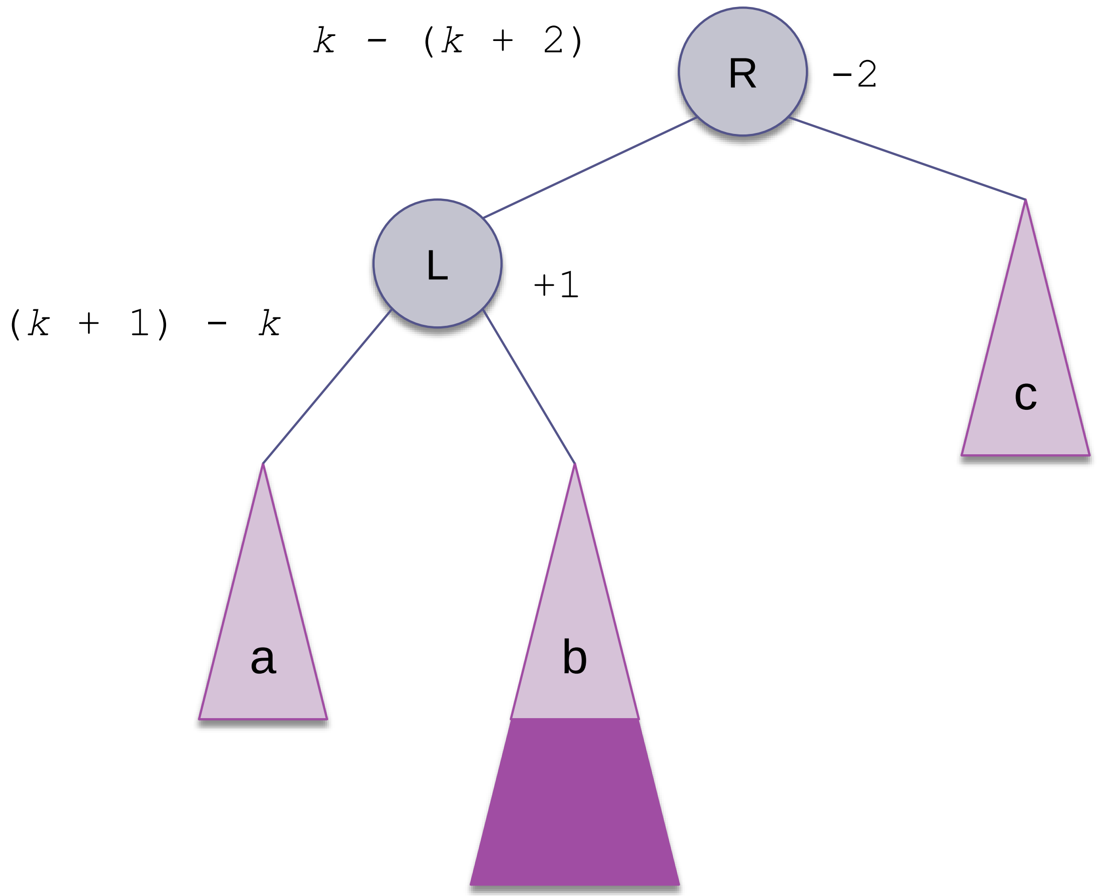
A Left-Left tree can be balanced by a rotation right

Balancing a Left-Left Tree (cont.)

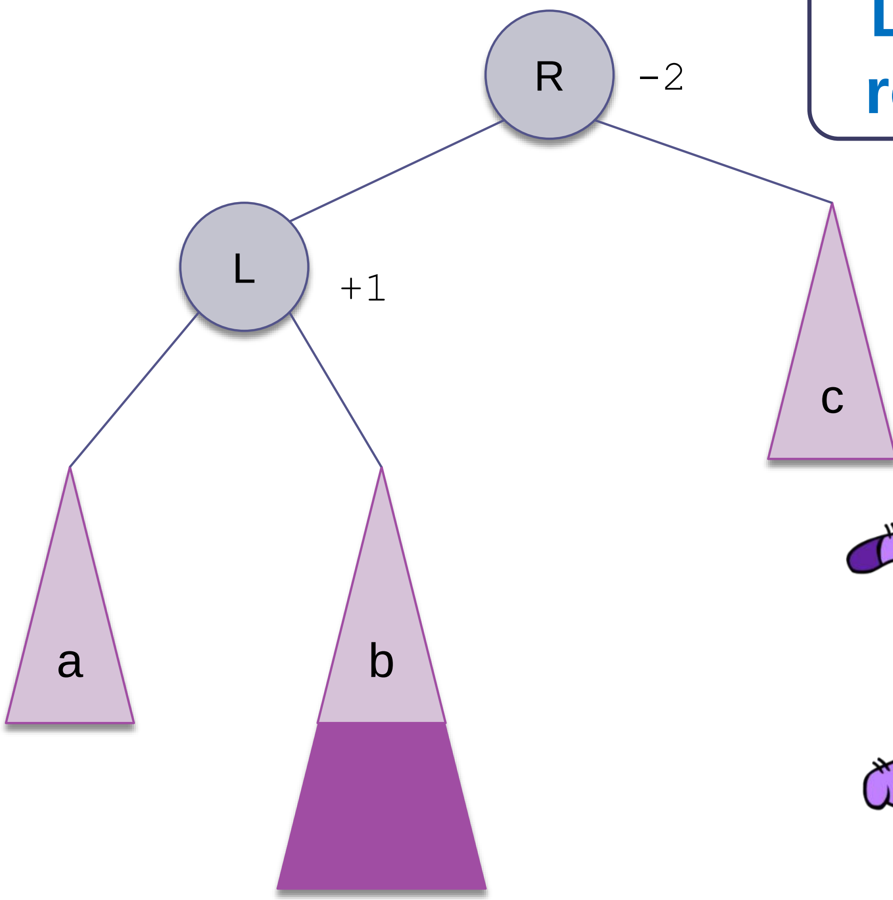


The overall height is *k + 2*

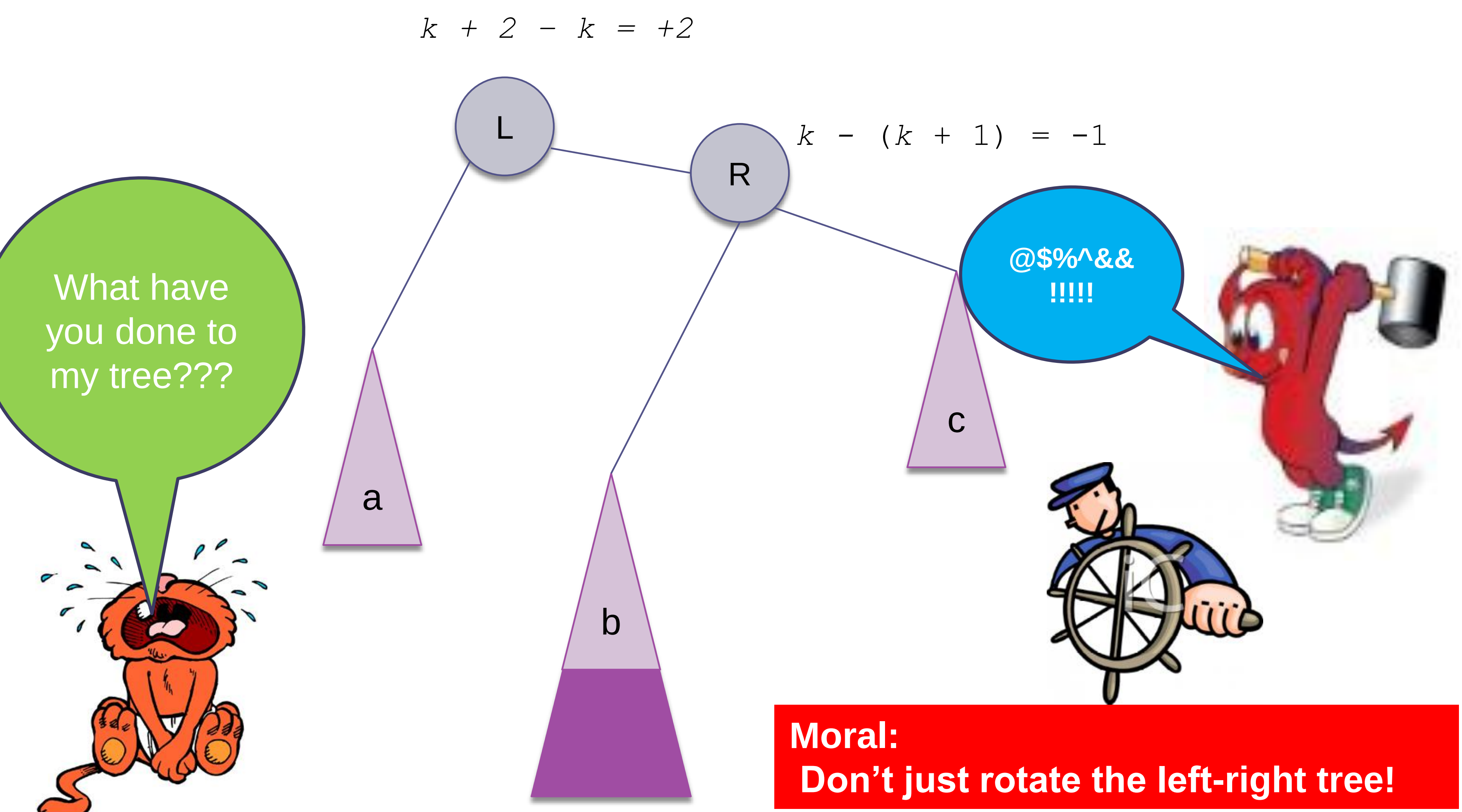
Balancing a Left-Right Tree



What should we do?

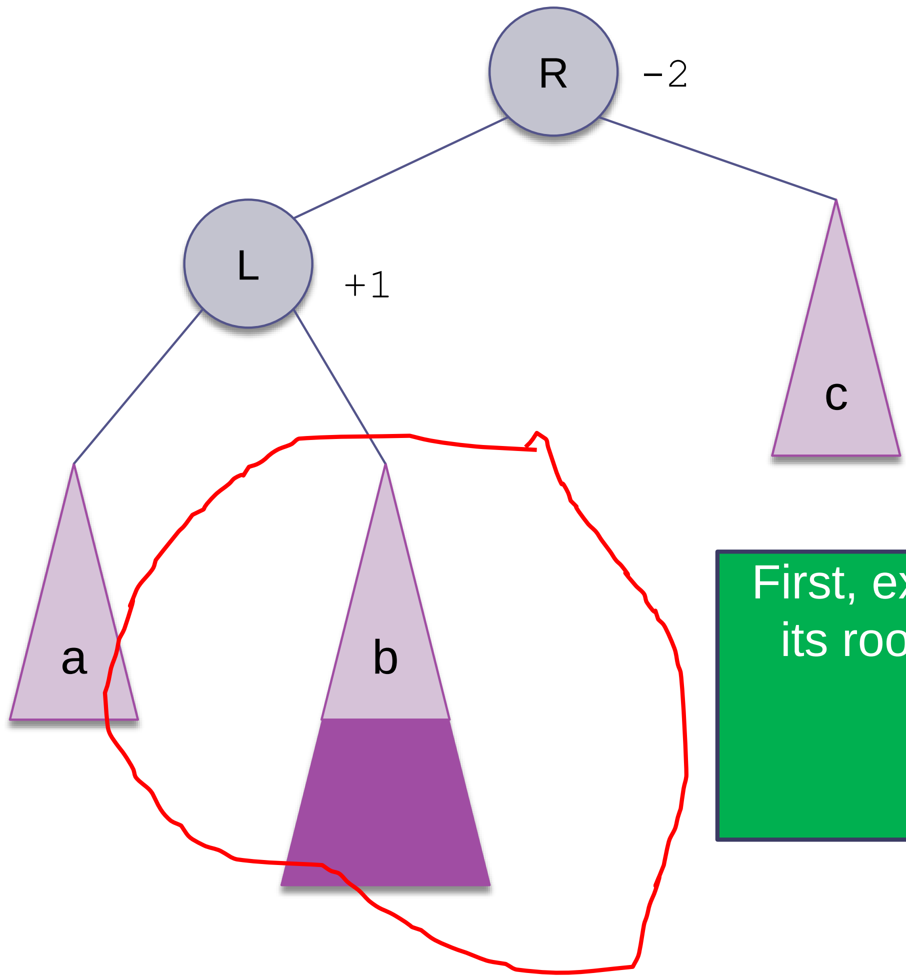


The aftermath…



Moral:
Don’t just rotate the left-right tree!

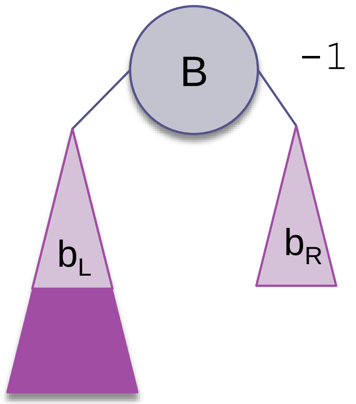
Let us slow down…



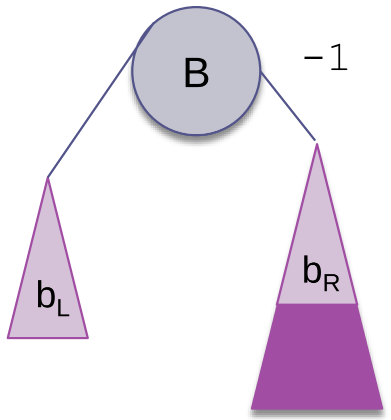
First, expand subtree b into its root B and subtrees b_L and b_R

Two possible cases for the expansion

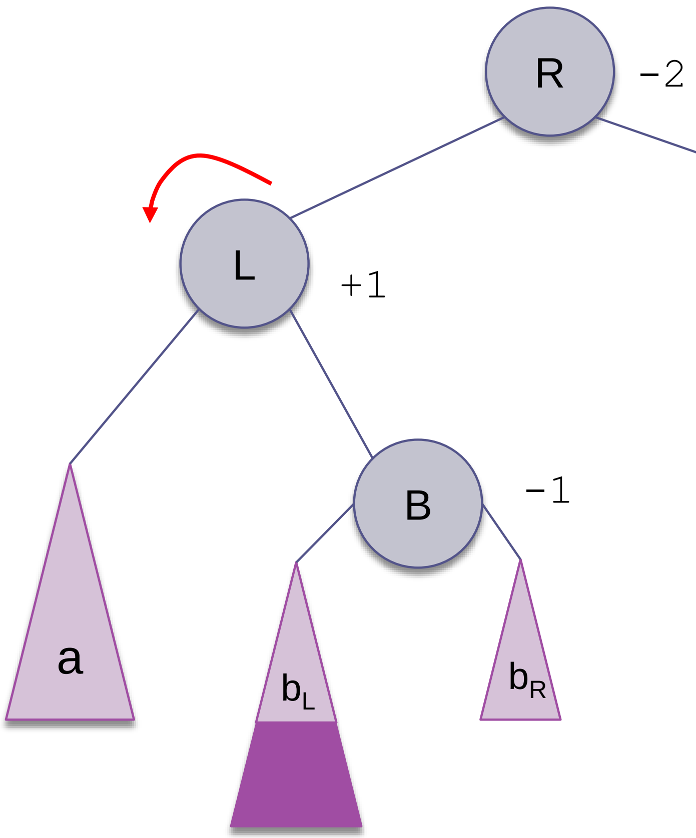
- Case 1



- Case 2

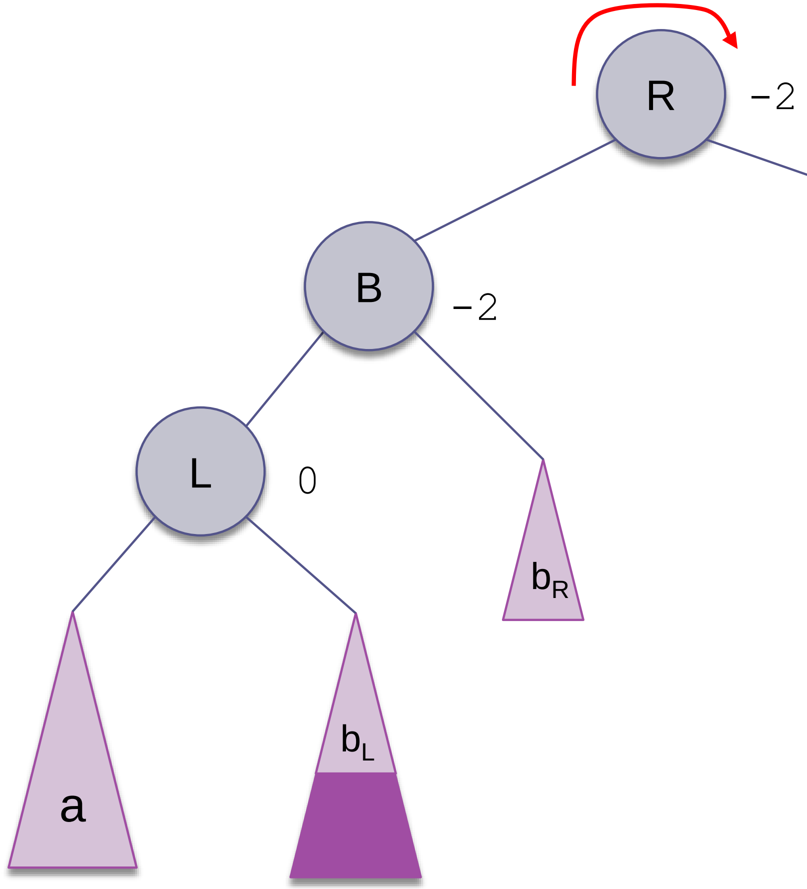


The first step in balancing a Left-Right Tree with Case 1



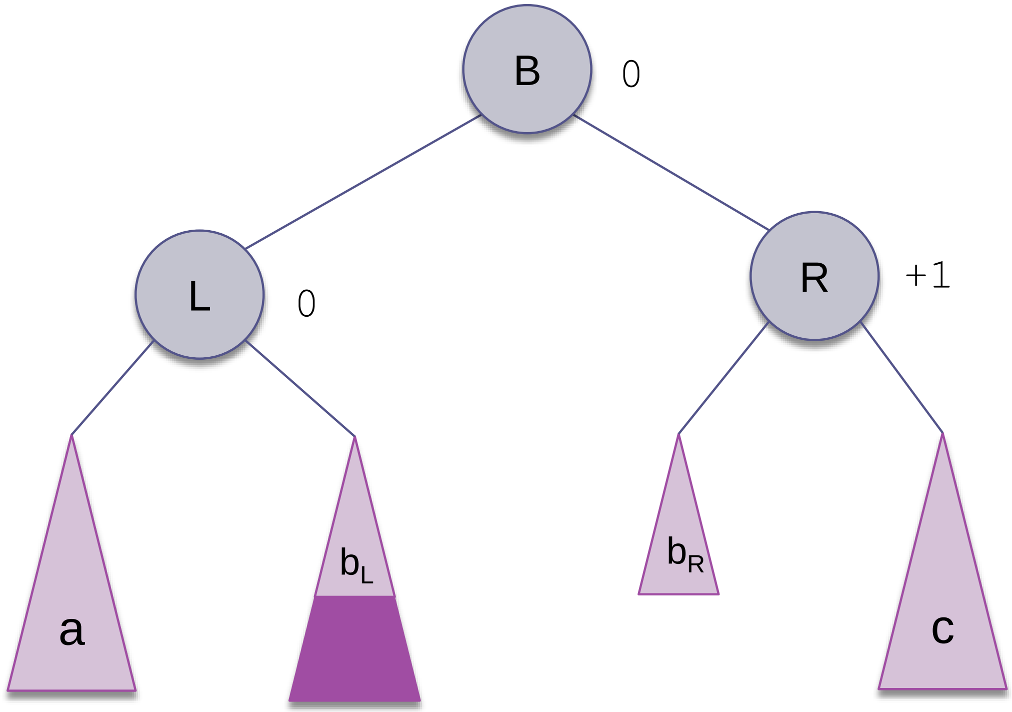
The left subtree L can now be rotated left

The first step accomplished!

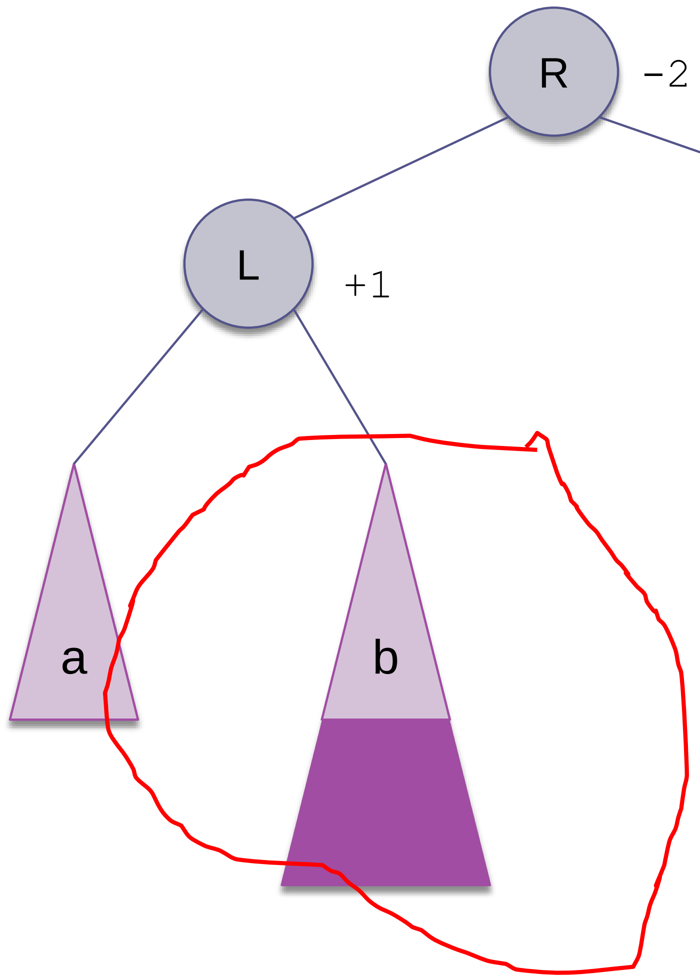


Now, a rotation right at the root will work!

Case 1 finished!

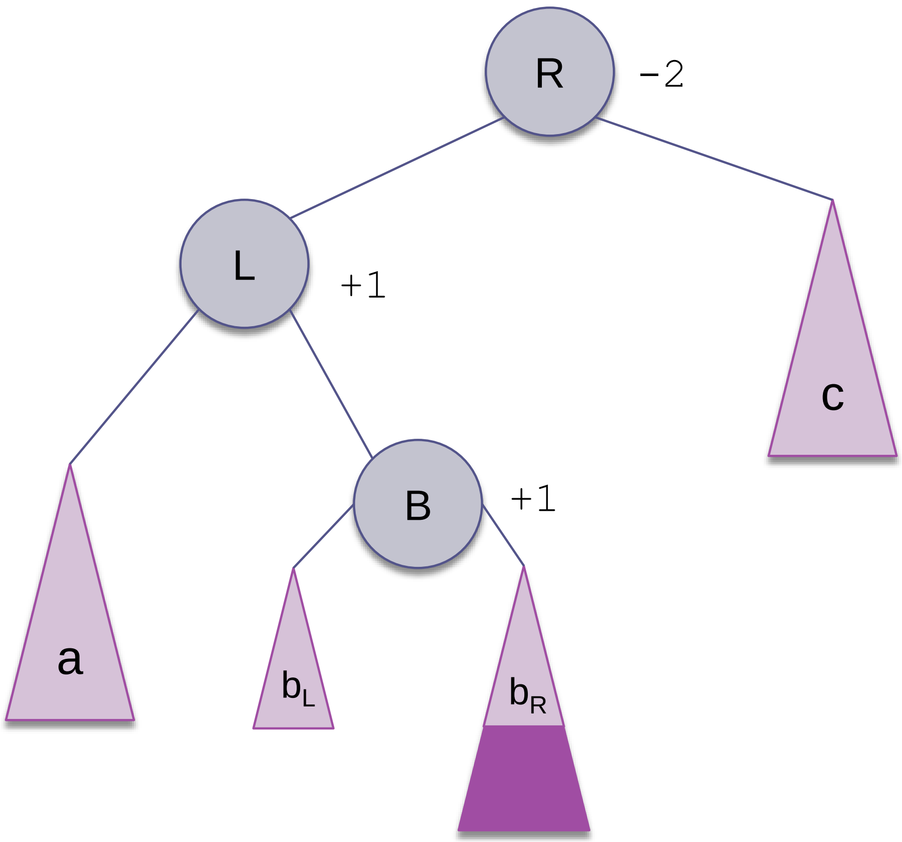


We were here

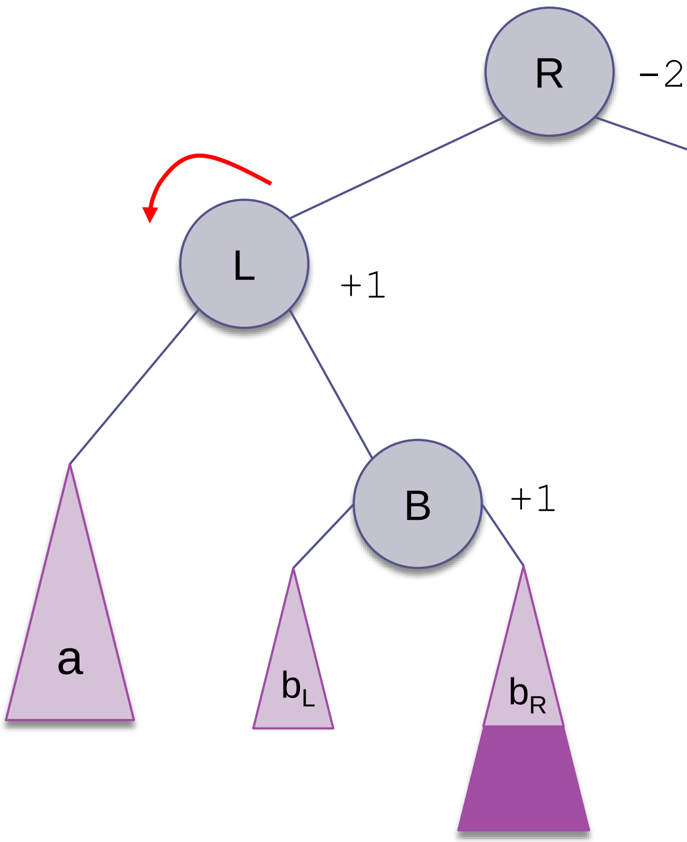


First, expand subtree b into its subtrees b_L and b_R

Now, to Case 2

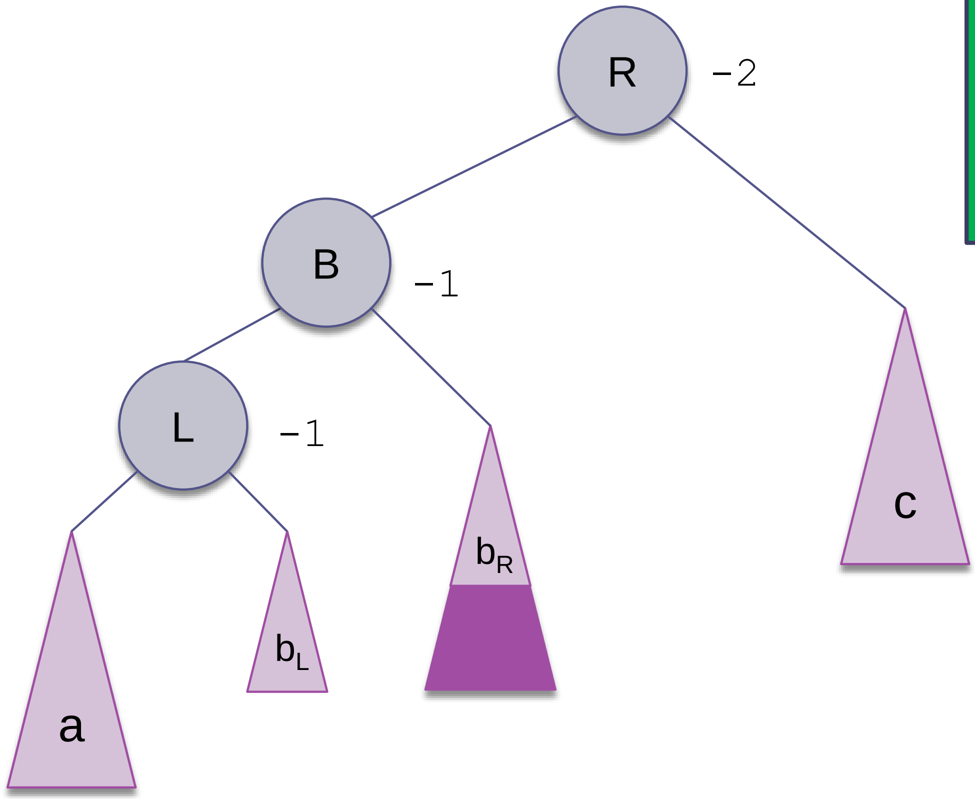


Balancing a Left-Right Tree, Case 2



Again, we rotate the left subtree to the left

Balancing a Left-Right Tree (Case 2)



Again, rotate the whole tree to the right

Balancing a Left-Right Tree