Lecture 9: Self Balancing Trees

CSE 373: Data Structures and Algorithms

Warm Up

Meet AVL Trees

AVL Trees must satisfy the following properties:

- binary trees: all nodes must have between 0 and 2 children
- binary search tree: for all nodes, all keys in the left subtree must be smaller and all keys in the right subtree must be larger than the root node
- balanced: for all nodes, there can be no more than a difference of 1 in the height of the left subtree from the right. Math.abs(height(left subtree) – height(right subtree)) ≤ 1

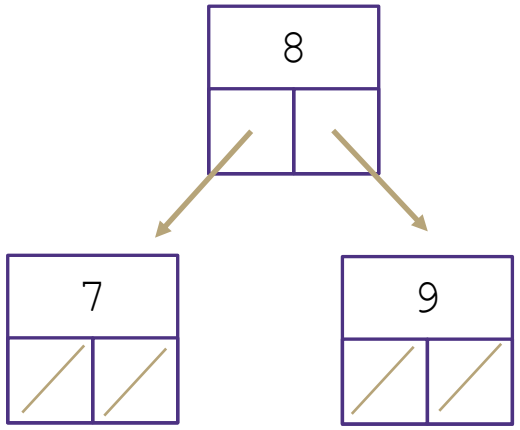
AVL stands for **A**delson-**V**elsky and **L**andis (the inventors of the data structure)

Measuring Balance

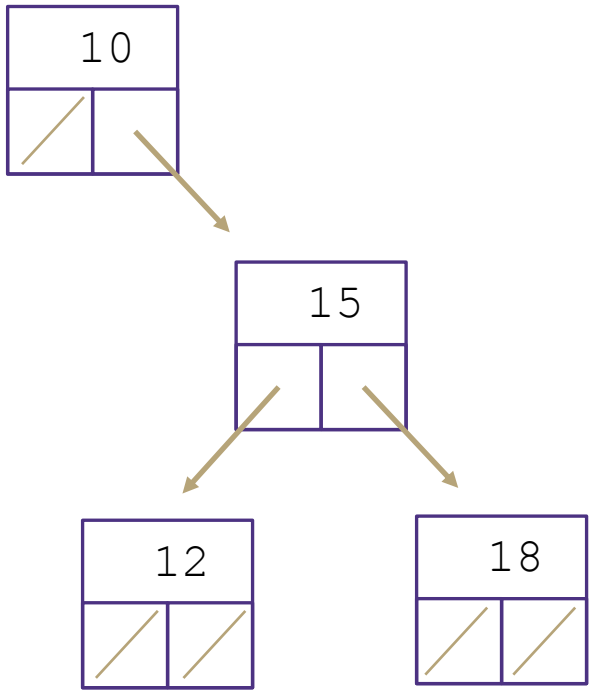
Measuring balance:

For each node, compare the heights of its two sub trees

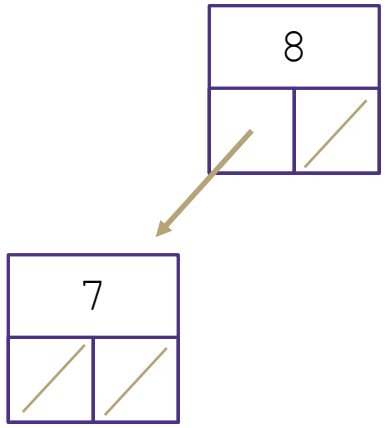
Balanced when the difference in height between sub trees is no greater than 1



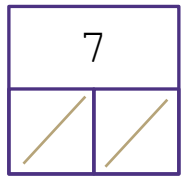
Balanced



Unbalanced



Balanced

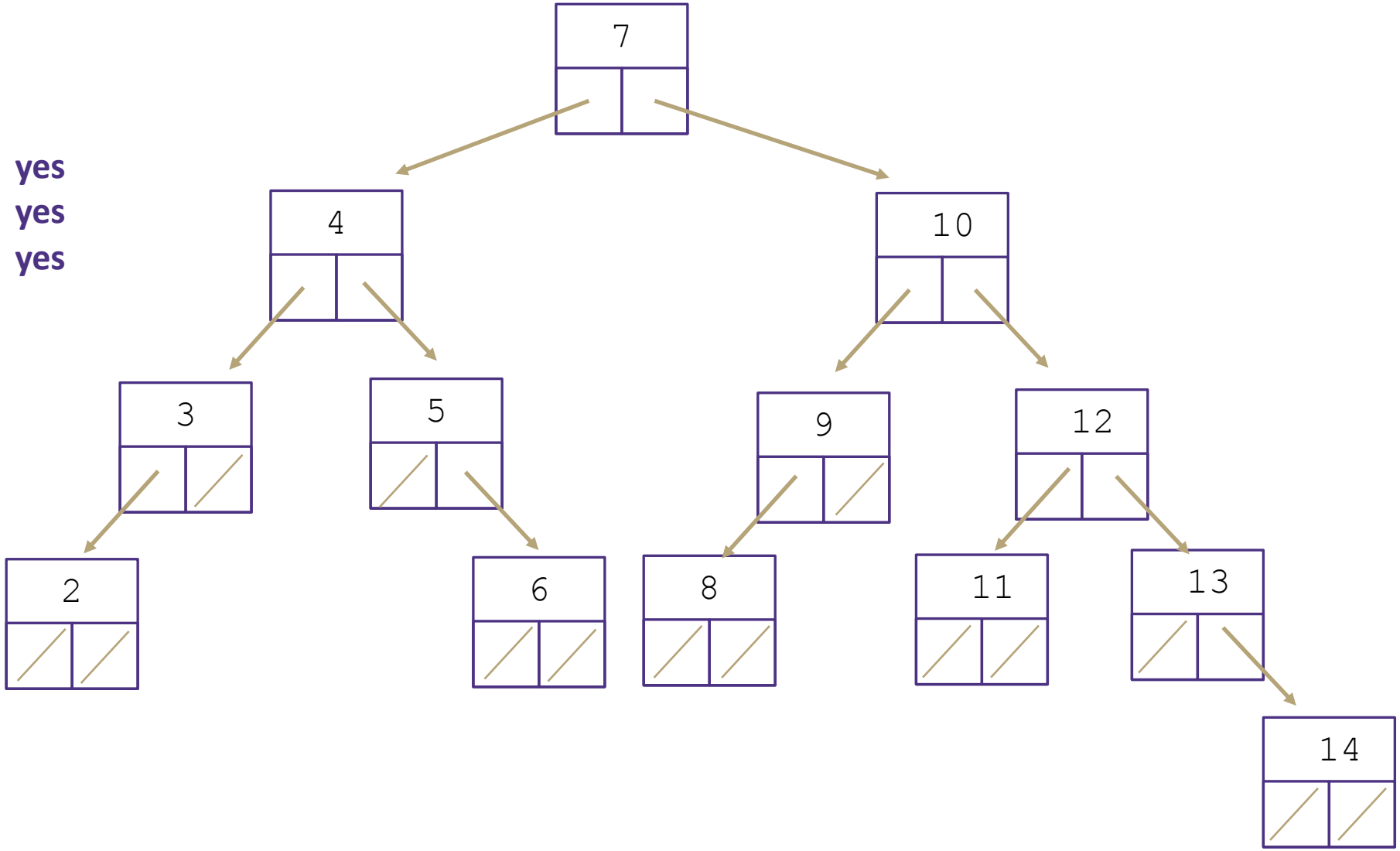


Balanced

Is this a valid AVL tree?

Is it…

- Binary **yes**
- BST **yes**
- Balanced? **yes**

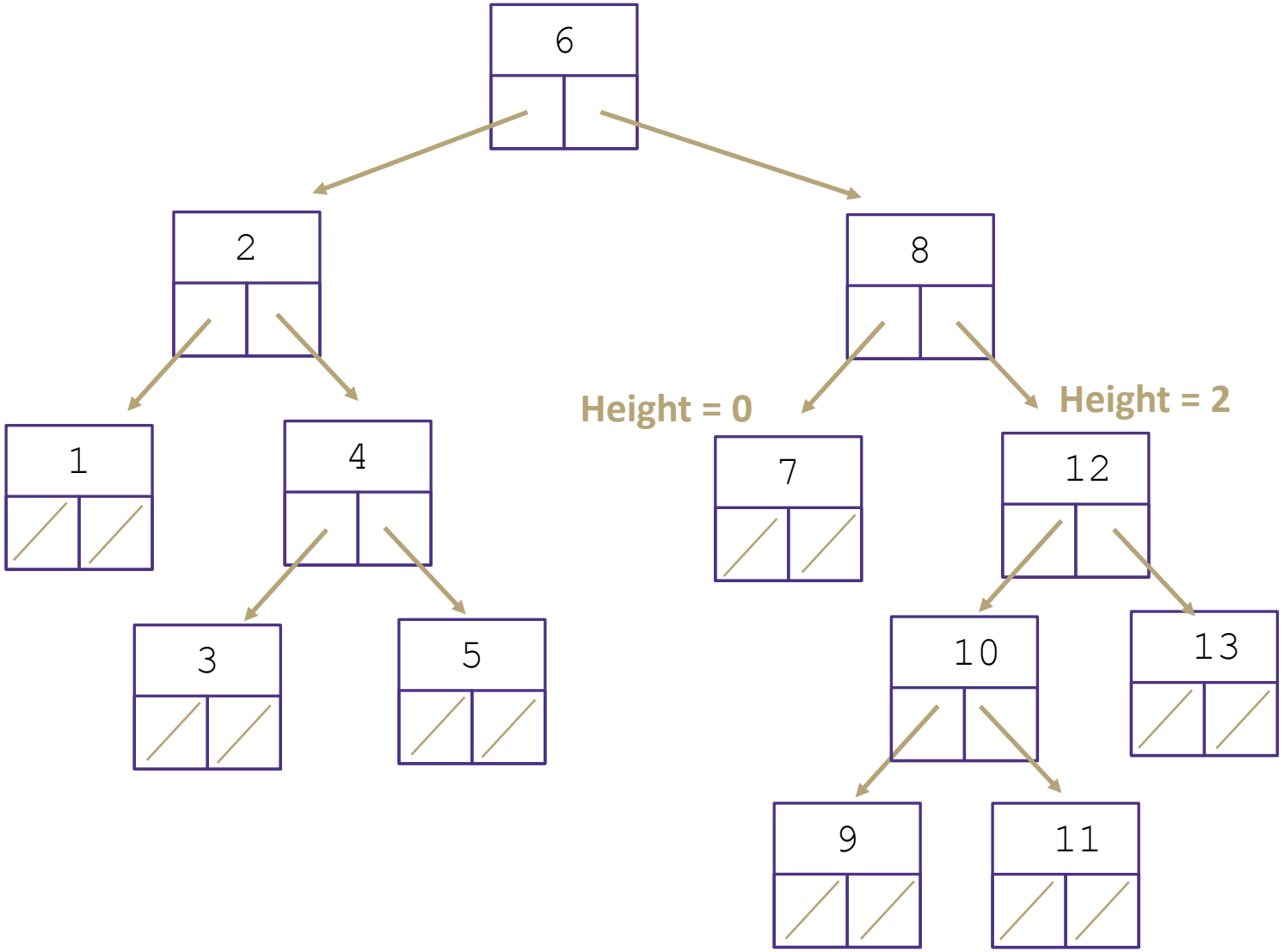


2 Minutes

Is this a valid AVL tree?

Is it…

- Binary **yes**
- BST **yes**
- Balanced? **no**

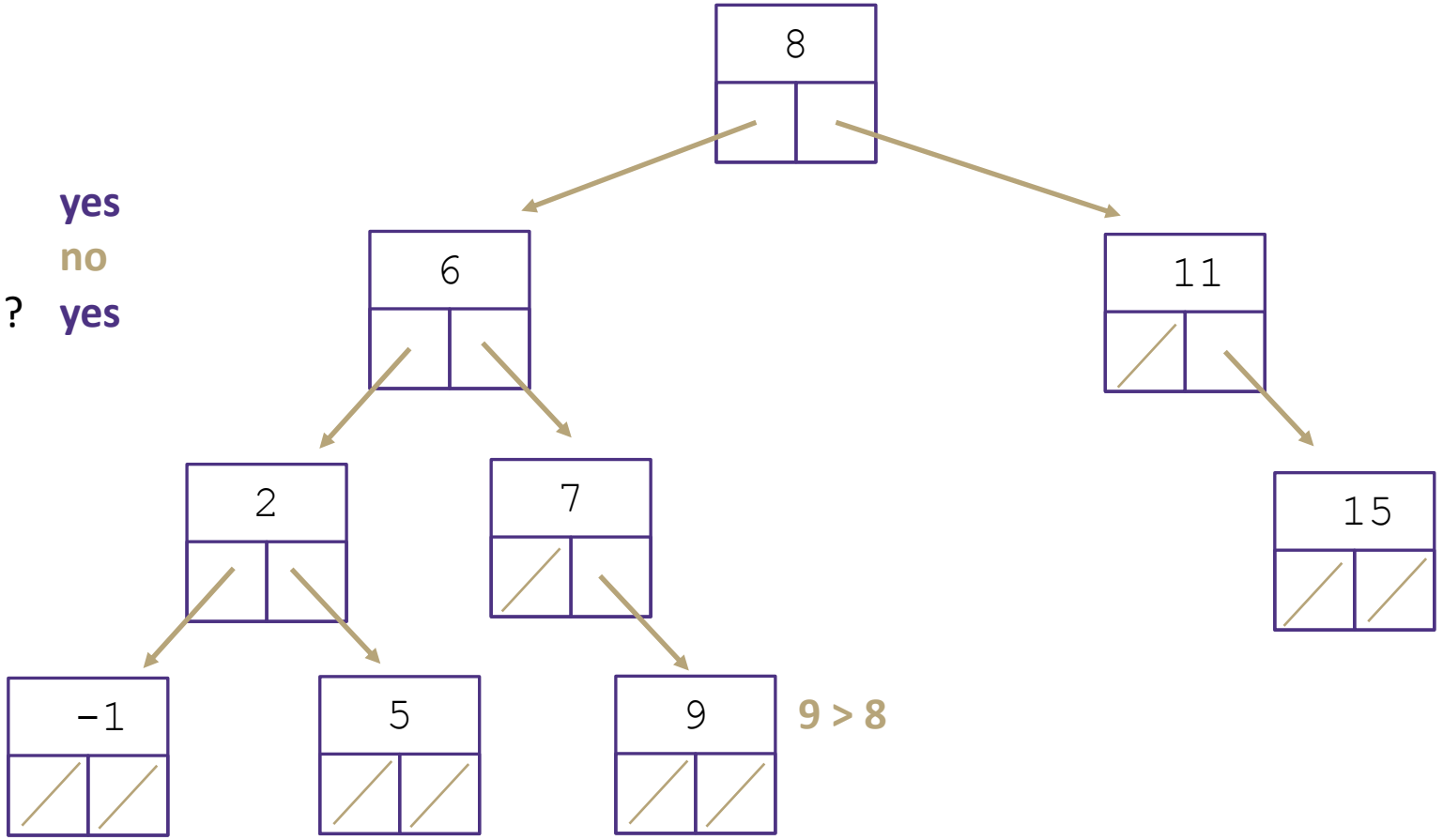


2 Minutes

Is this a valid AVL tree?

Is it…

- Binary **yes**
- BST **no**
- Balanced? **yes**



Implementing an AVL tree dictionary

Dictionary Operations:

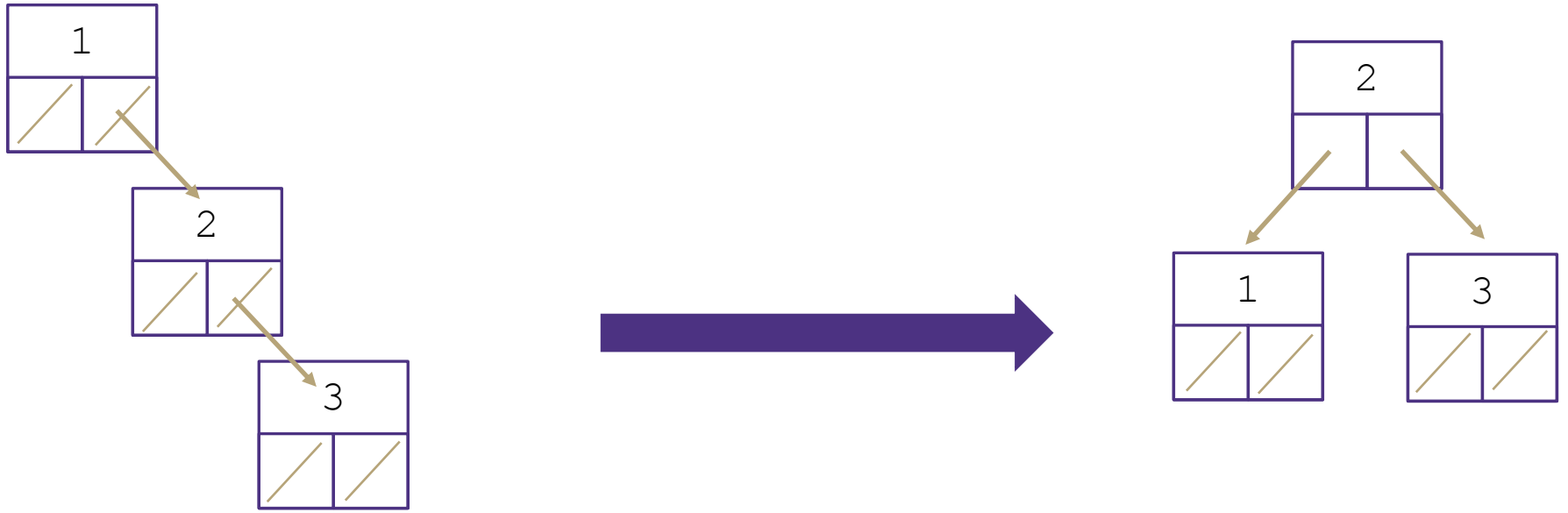
get() – same as BST

containsKey() – same as BST

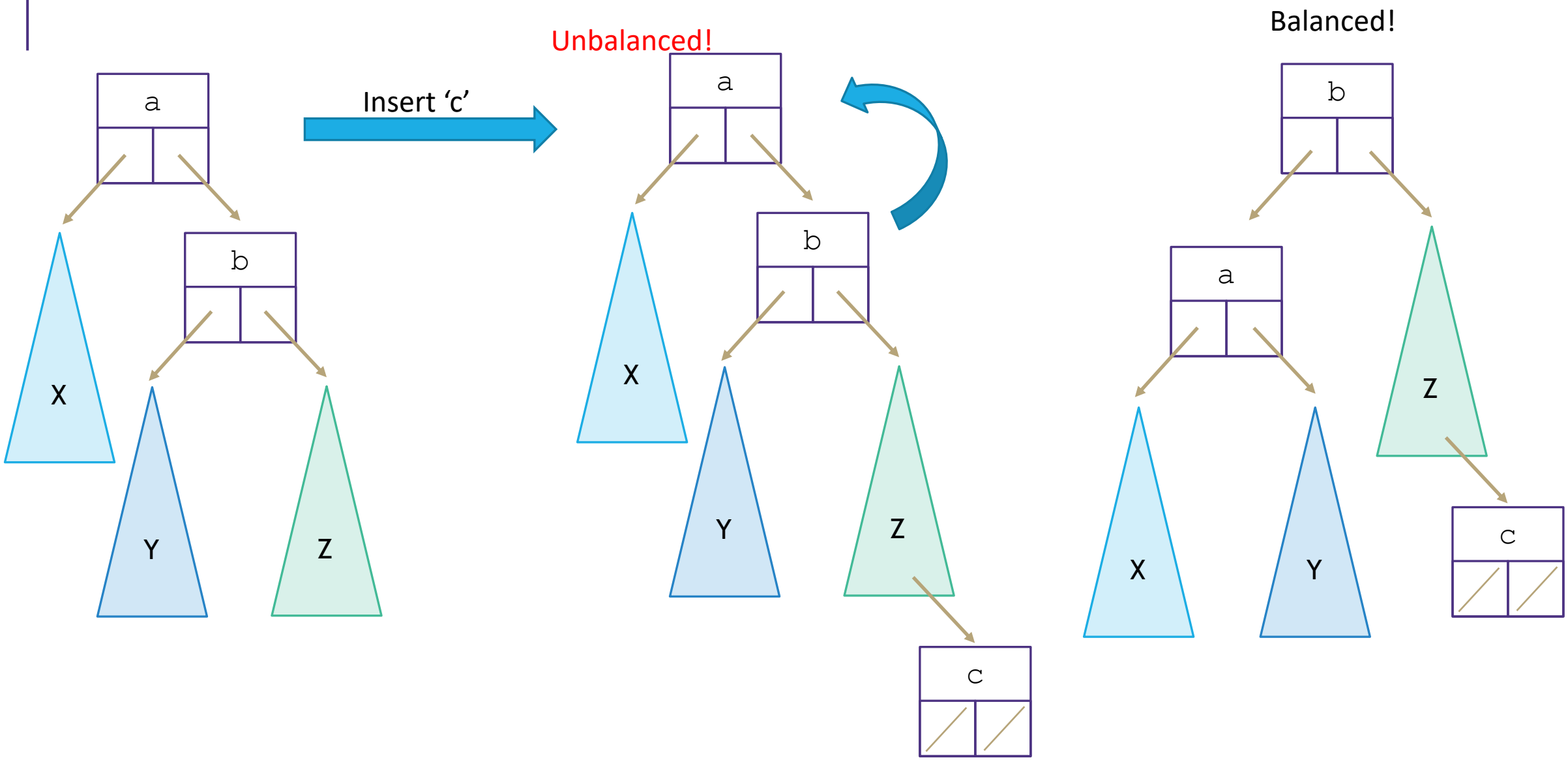
put() - Add the node to keep BST, fix AVL property if necessary

remove() - Replace the node to keep BST, fix AVL property if necessary

Unbalanced!

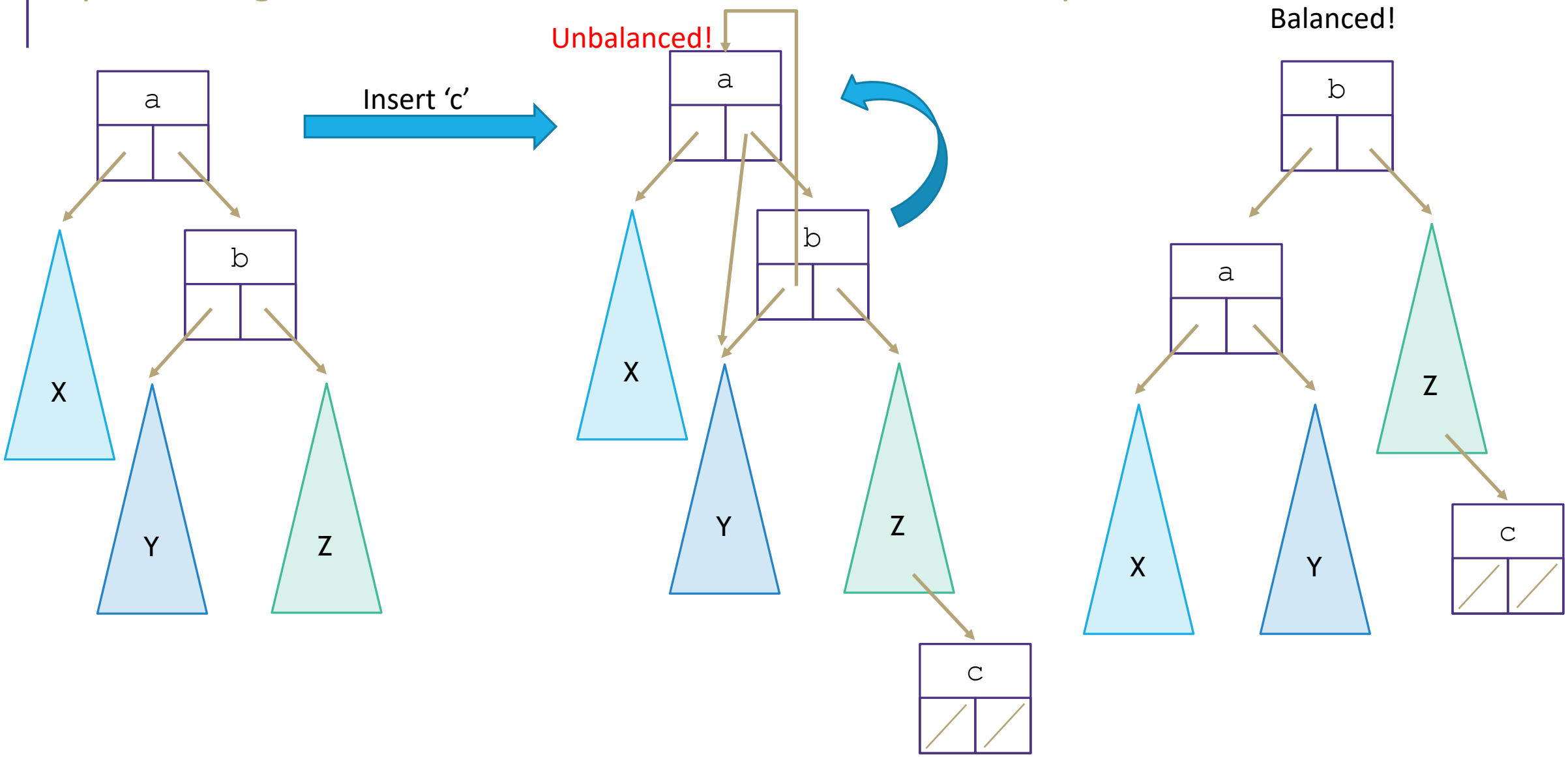


Rotations!



Rotate Left

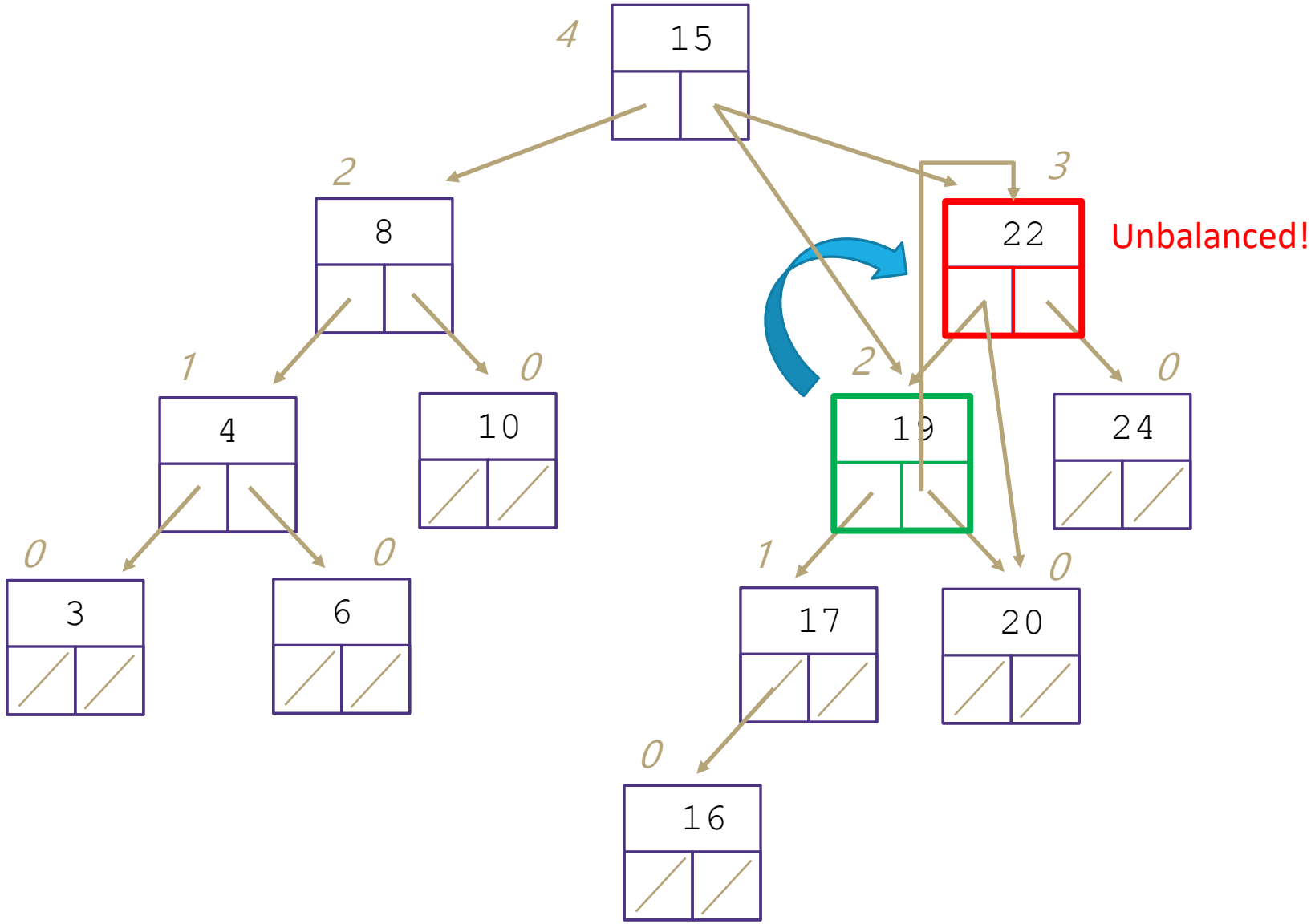
parent's right becomes child's left, child's left becomes its parent



Rotate Right

parent's left becomes child's right, child's right becomes its parent

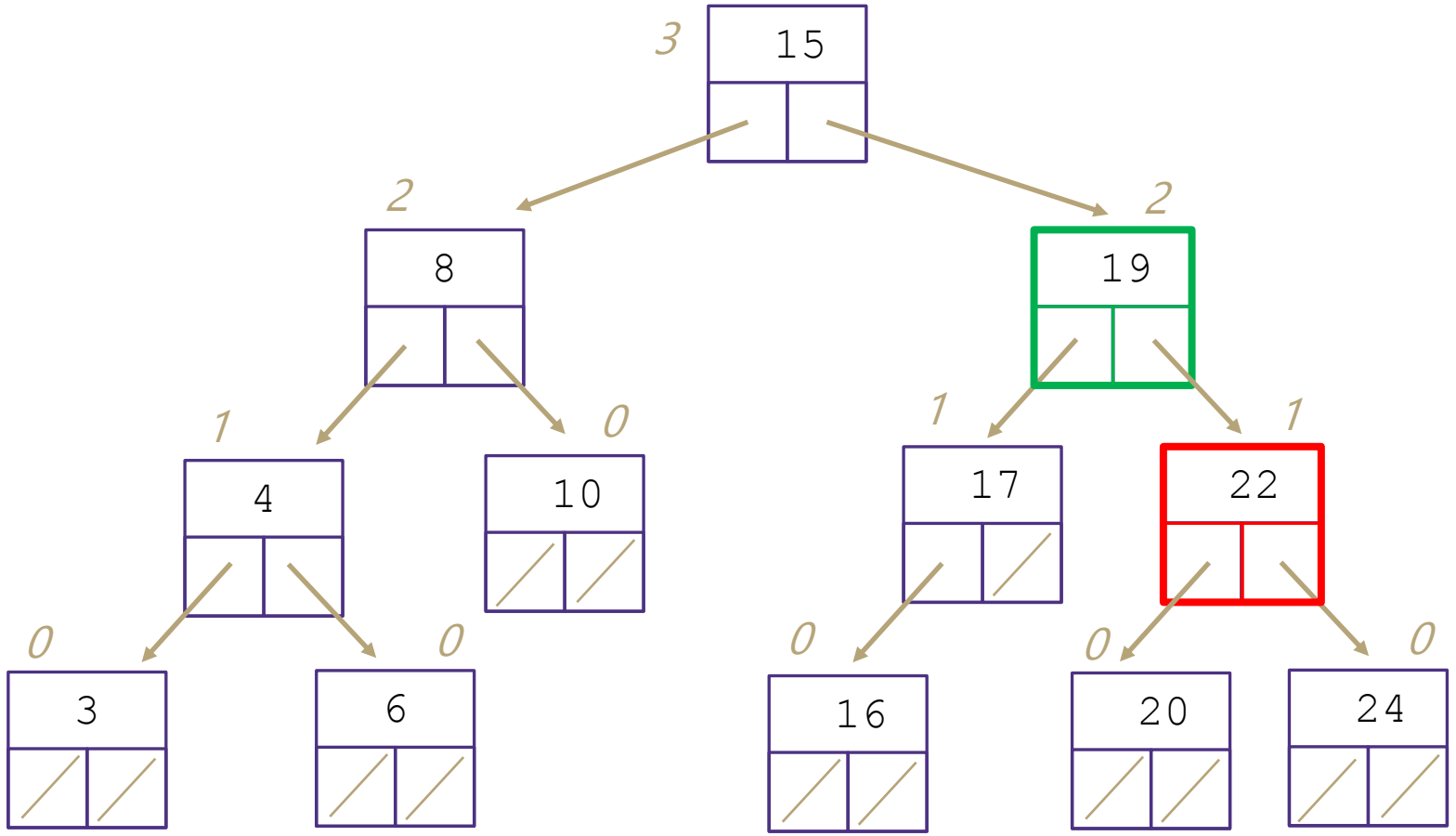
put(16);



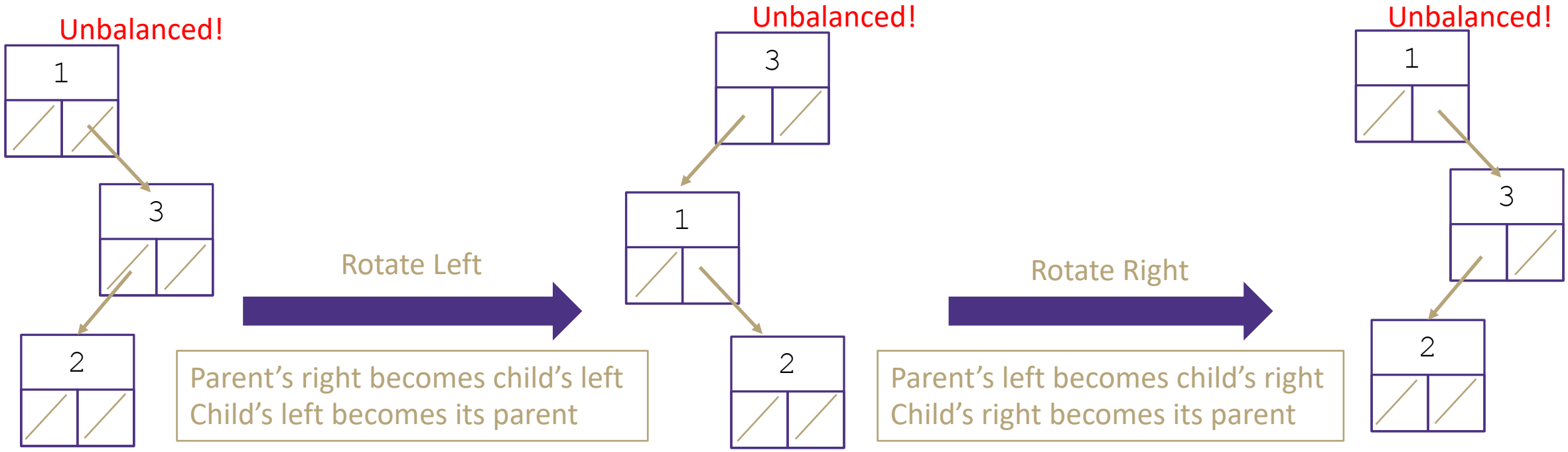
Rotate Right

parent's left becomes child's right, child's right becomes its parent

put(16);



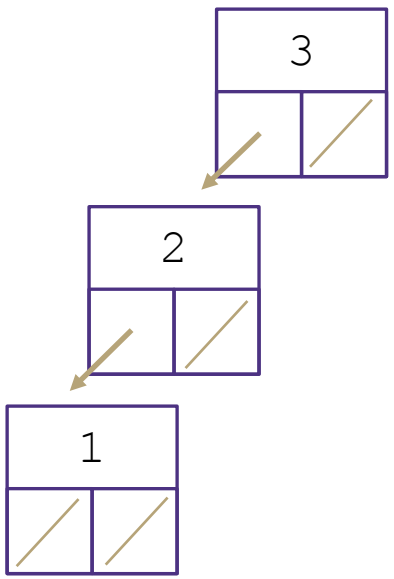
So much can go wrong

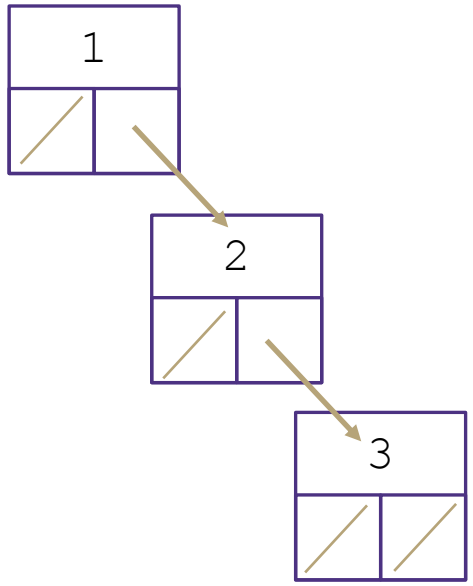


Two AVL Cases

Line Case

Solve with **1** rotation





Rotate Right

Parent's left becomes child's right

Child's right becomes its parent

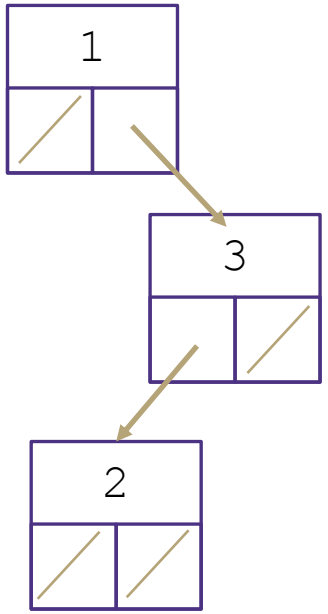
Rotate Left

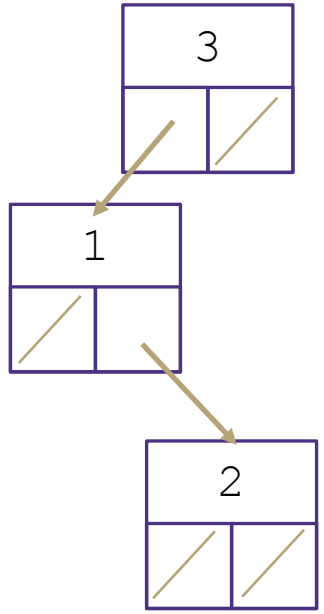
Parent's right becomes child's left

Child's left becomes its parent

Kink Case

Solve with **2** rotations





Right Kink Resolution

Rotate subtree left

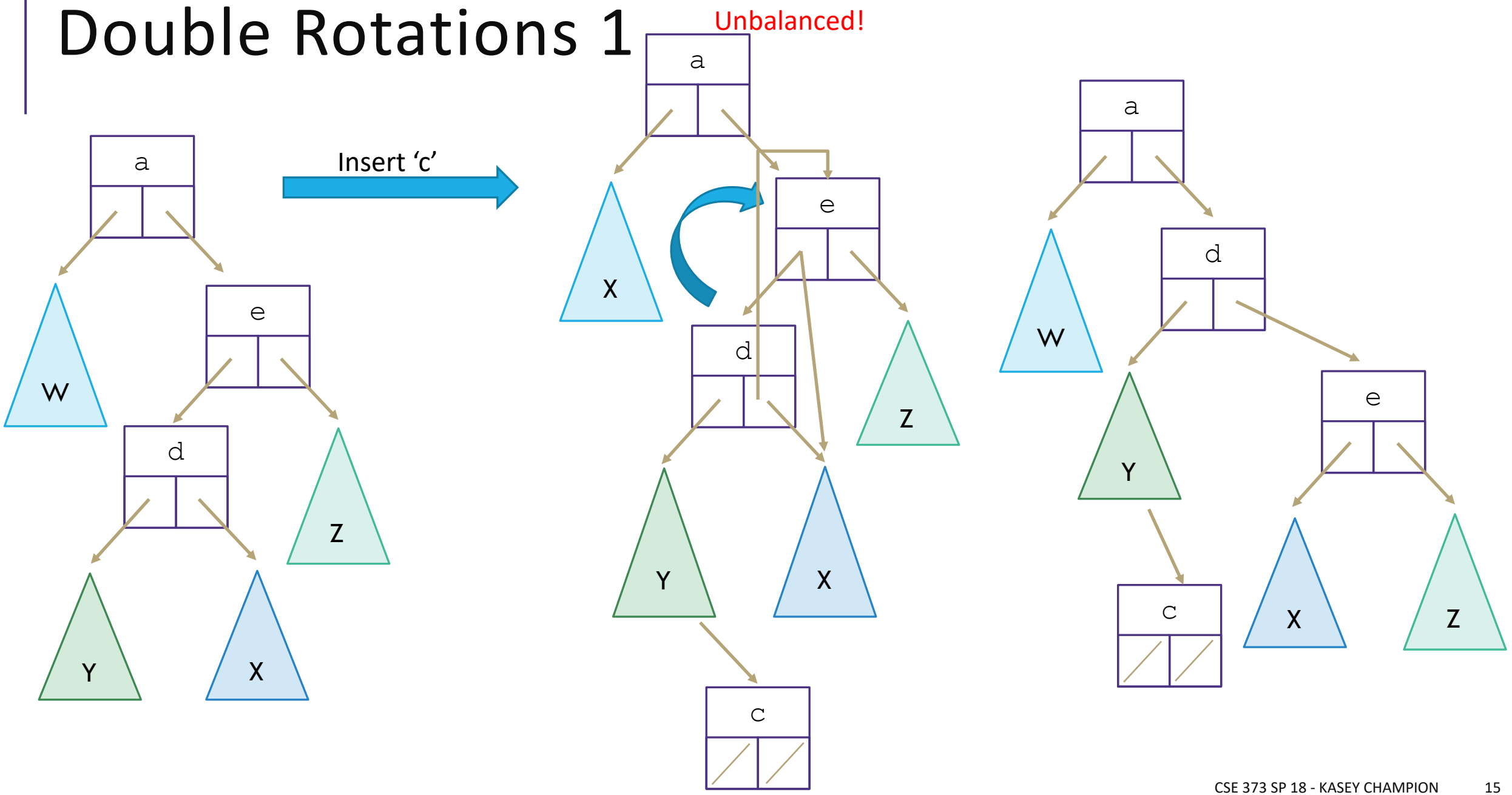
Rotate root tree right

Left Kink Resolution

Rotate subtree right

Rotate root tree left

Double Rotations 1



Double Rotations 2