Self Balancing Trees

Data Structures and
Algorithms

Warm Up

What will the binary search tree look like if you insert nodes in the following order:

5, 8, 7, 10, 9, 4, 2, 3, 1

What is the pre-order traversal order for the resulting tree?

Socrative:

www.socrative.com
Room Name: CSE373
Please enter your name as: Last, First

Implement Dictionary

Binary Search Trees allow us to:

- quickly find what we're looking for
- add and remove values easily

Dictionary Operations:

Runtime in terms of height, "h"

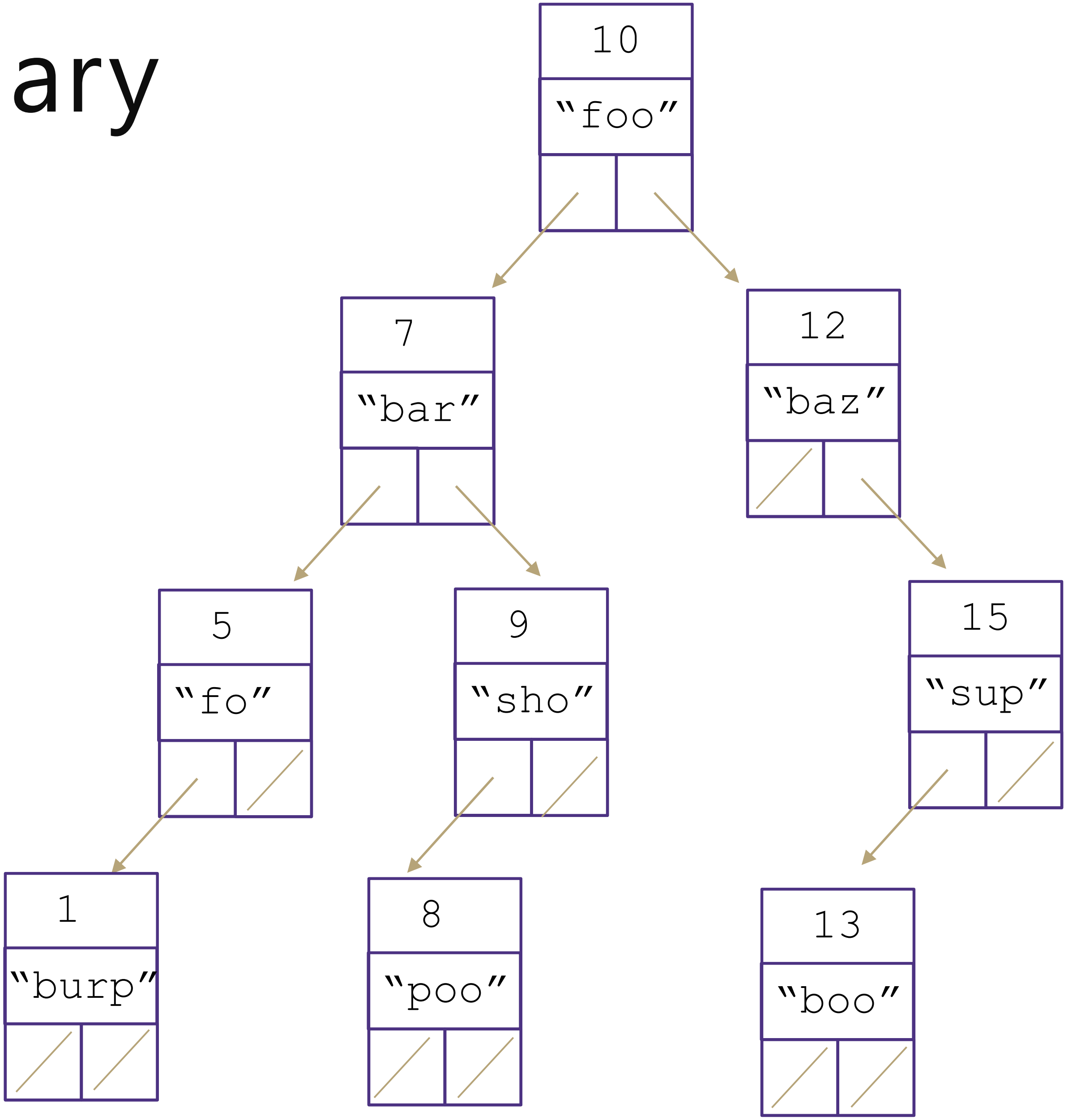
get() – O(h)

put() – O(h)

remove() – O(h)

What do you replace the node with?

Largest in left sub tree or smallest in right sub tree



Implementing Put and Remove

Height in terms of Nodes

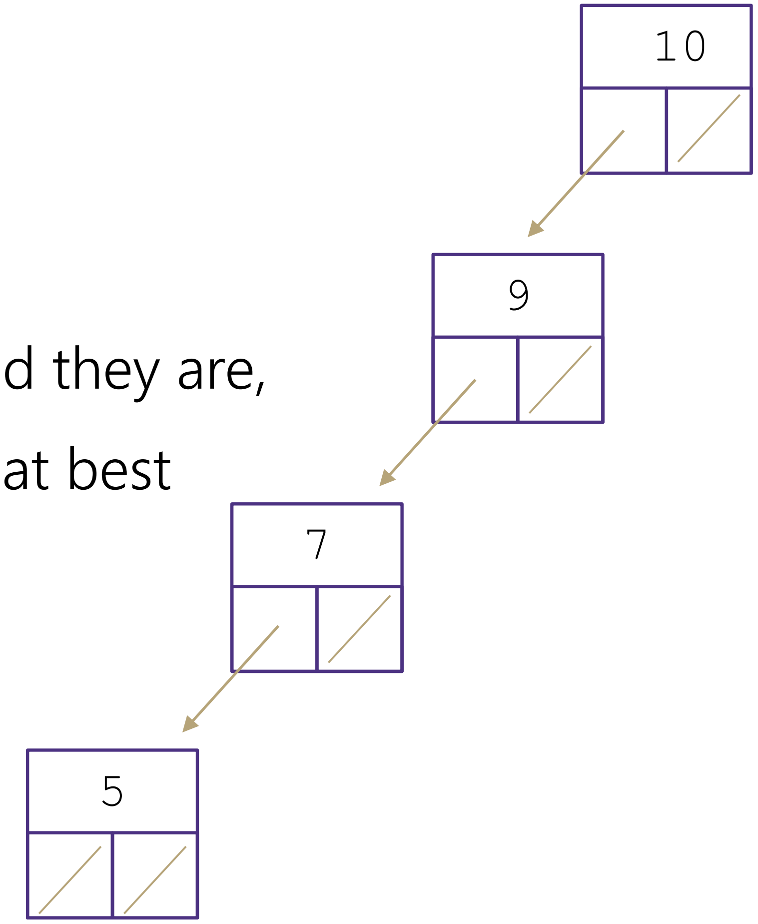
For “balanced” trees $h \approx \log_c(n)$ where c is the maximum number of children

Balanced binary trees $h \approx \log_2(n)$

Balanced trinary tree $h \approx \log_3(n)$

Thus for balanced trees operations take $\Theta(\log_c(n))$

Unbalanced Trees



Is this a valid Binary Search Tree?

Yes, but…

We call this a **degenerate tree**

For trees, depending on how balanced they are,

Operations at worst can be O(n) and at best

can be O(logn)

How are degenerate trees formed?

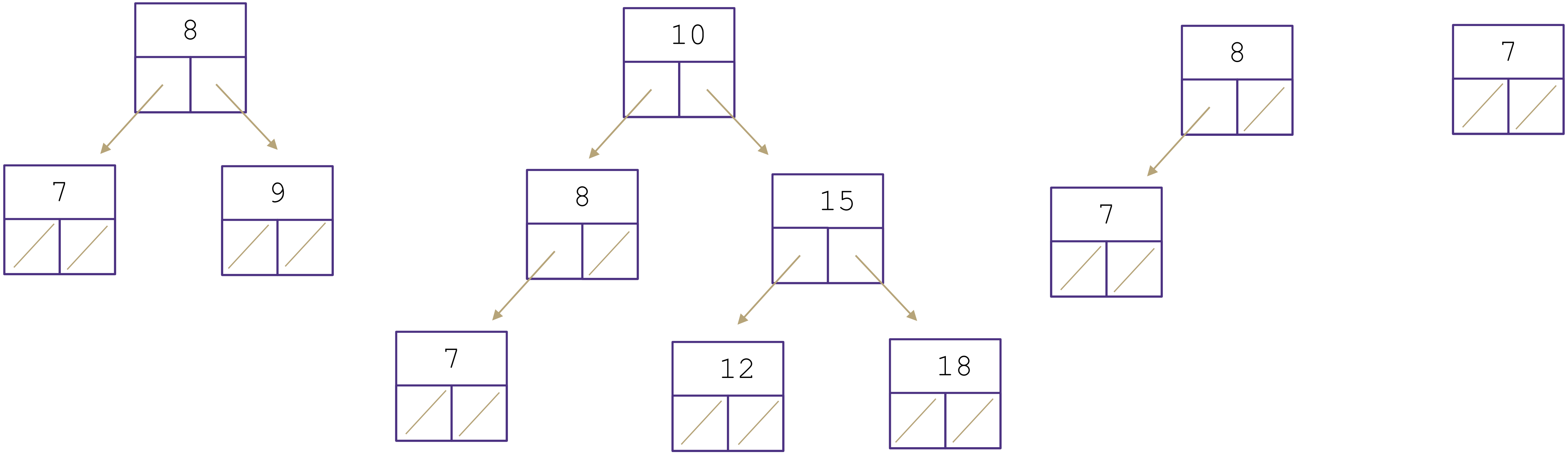
- insert(10)
- insert(9)
- insert(7)
- insert(5)

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



Meet AVL Trees

AVL Trees must satisfy the following properties:

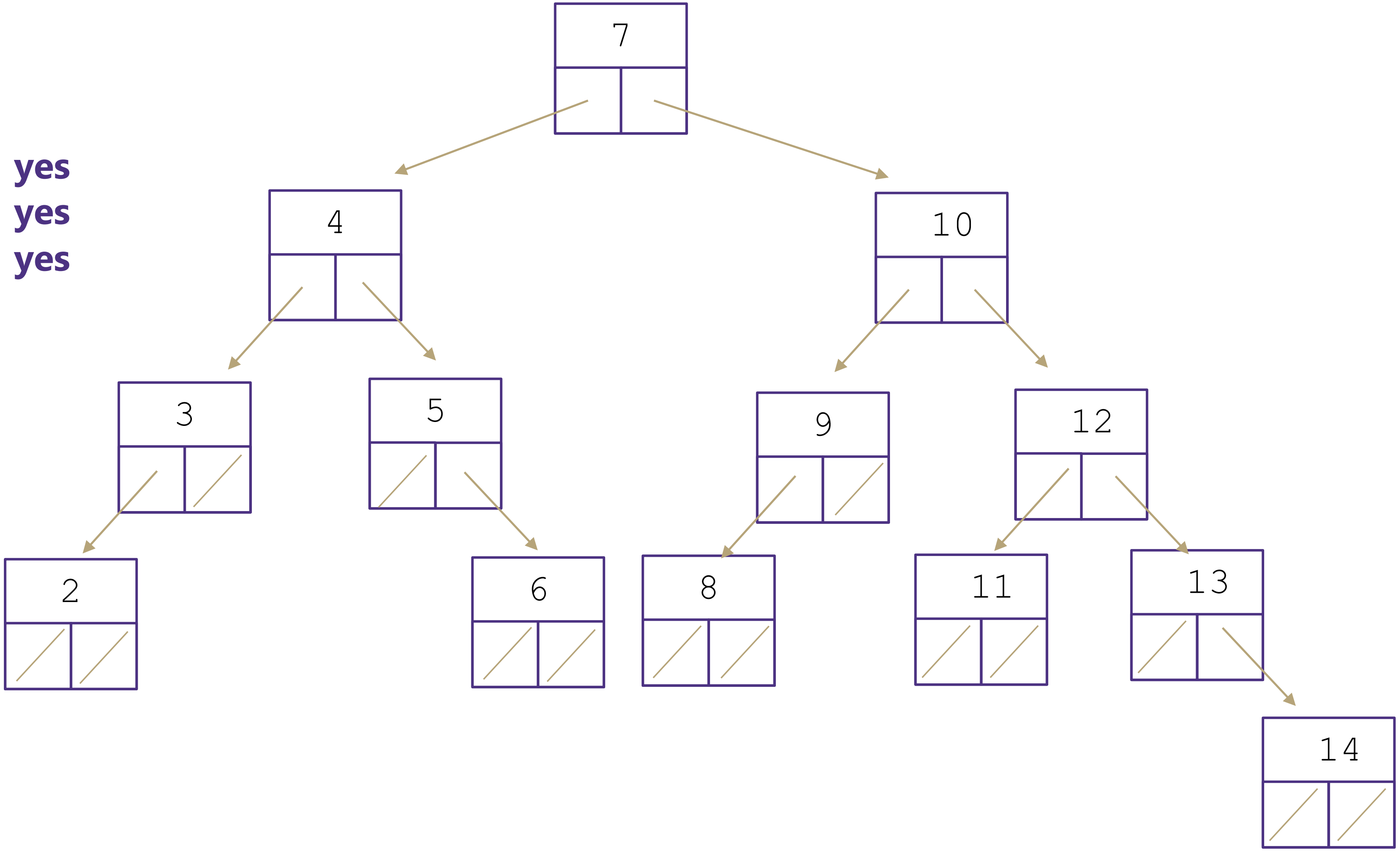
- binary trees: all nodes must have between 0 and 2 children
- binary search tree: for all nodes, all keys in the left subtree must be smaller and all keys in the right subtree must be larger than the root node
- balanced: for all nodes, there can be no more than a difference of 1 in the height of the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

Is this a valid AVL tree?

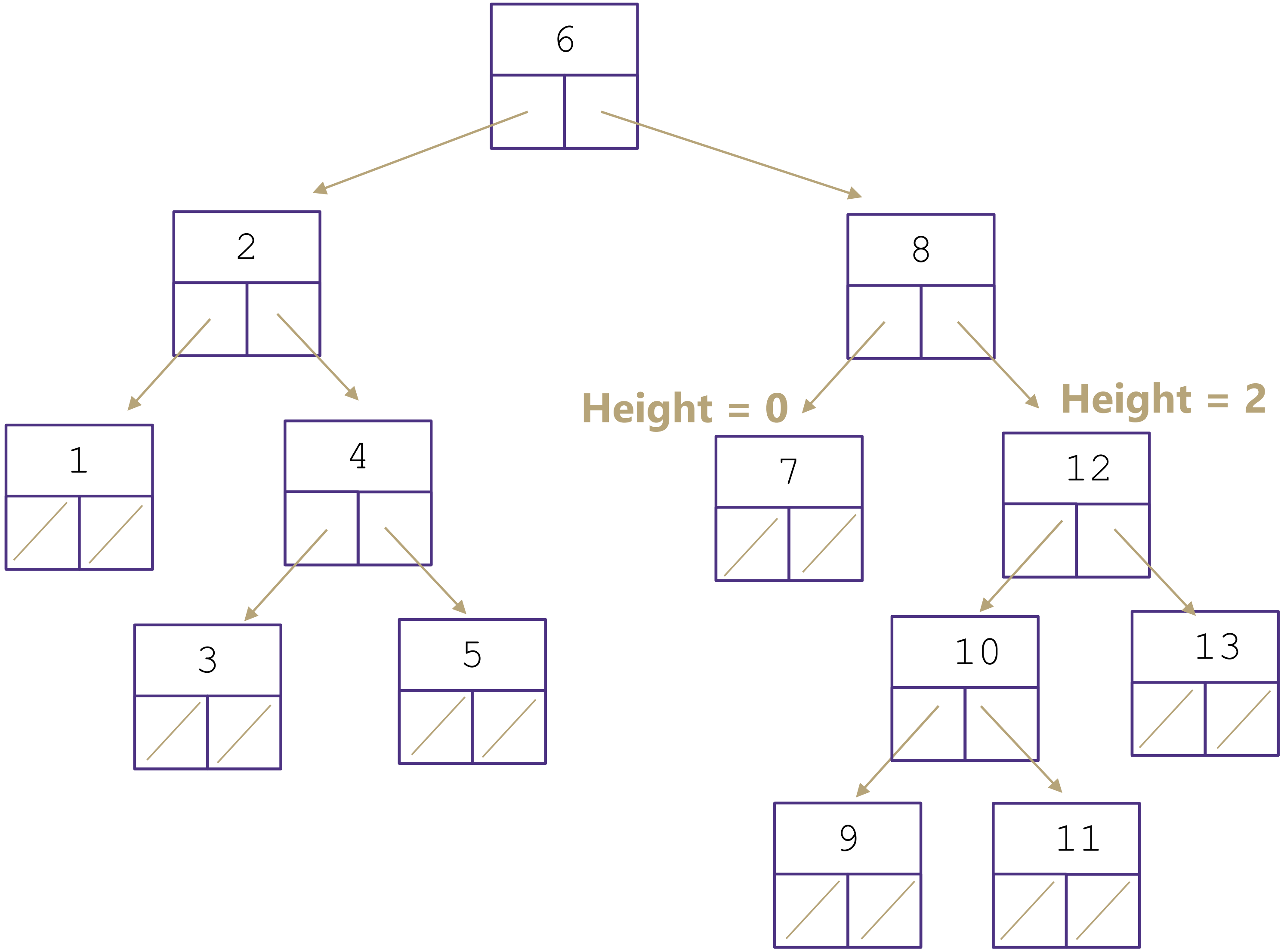
Is it…

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



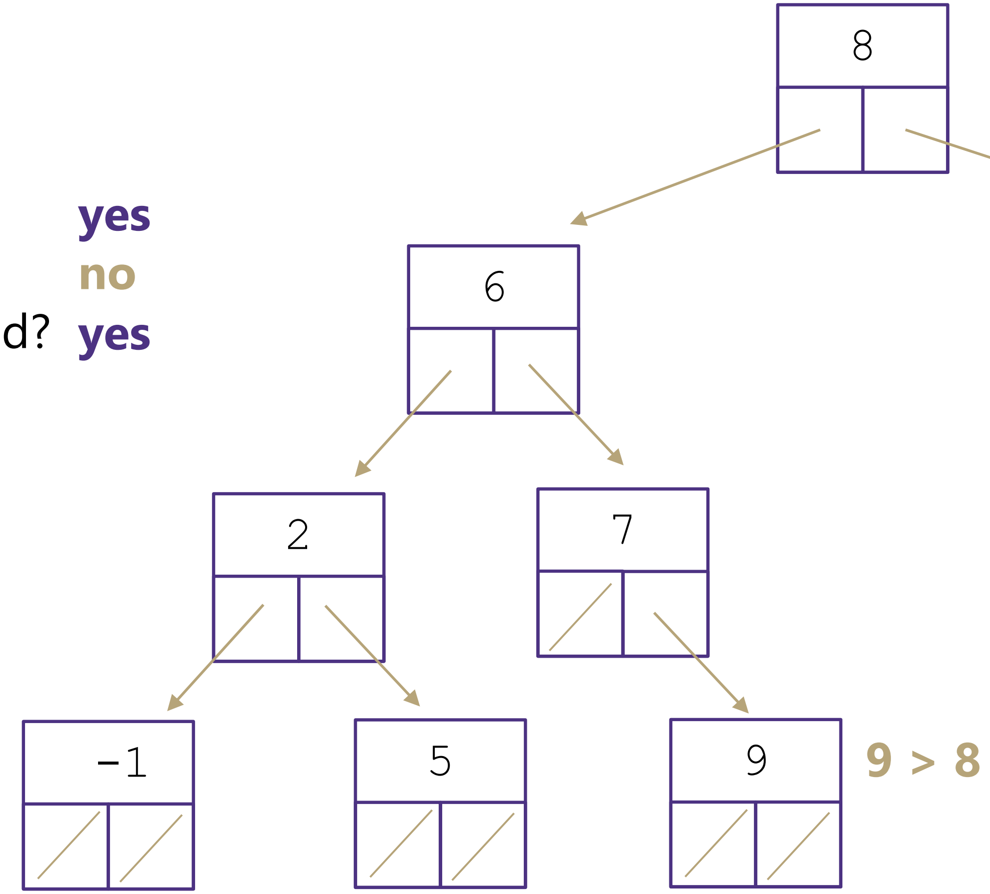
Is this a valid AVL tree?

Is it…
- Binary **yes**
- BST **yes**
- Balanced? **no**



Is this a valid AVL tree?

Is it…
- Binary **yes**
- BST **no**
- Balanced? **yes**



Implementing an AVL tree dictionary

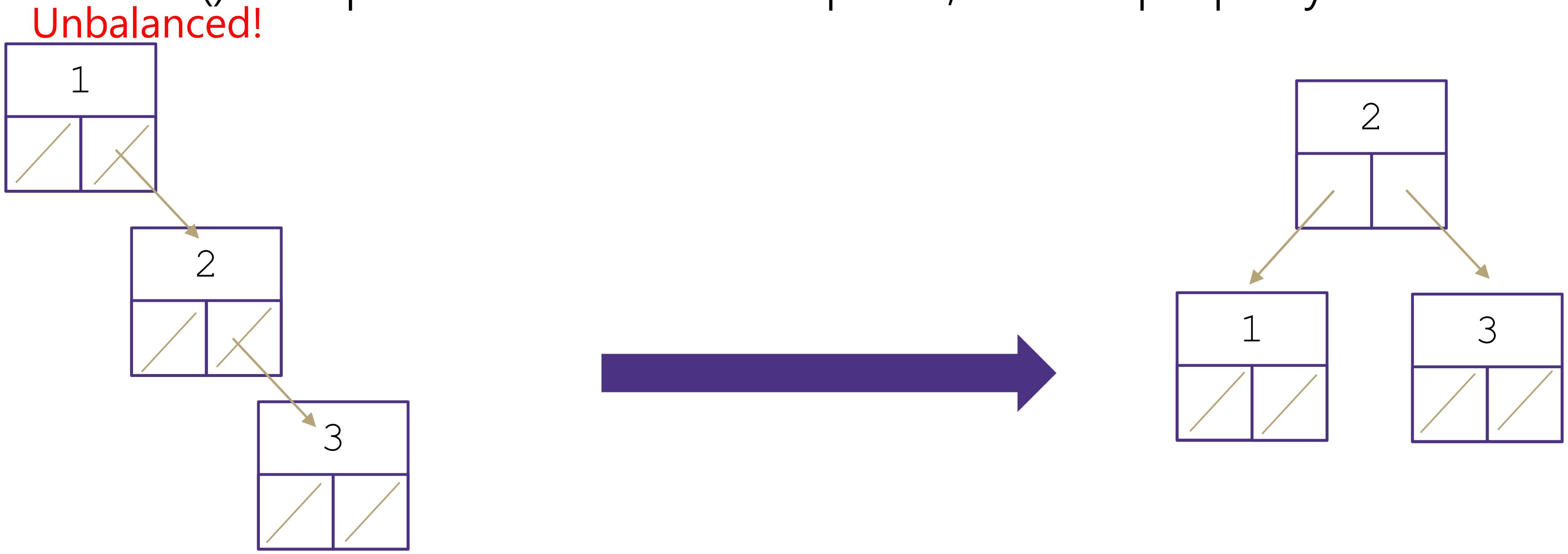
Dictionary Operations:

get() – same as BST

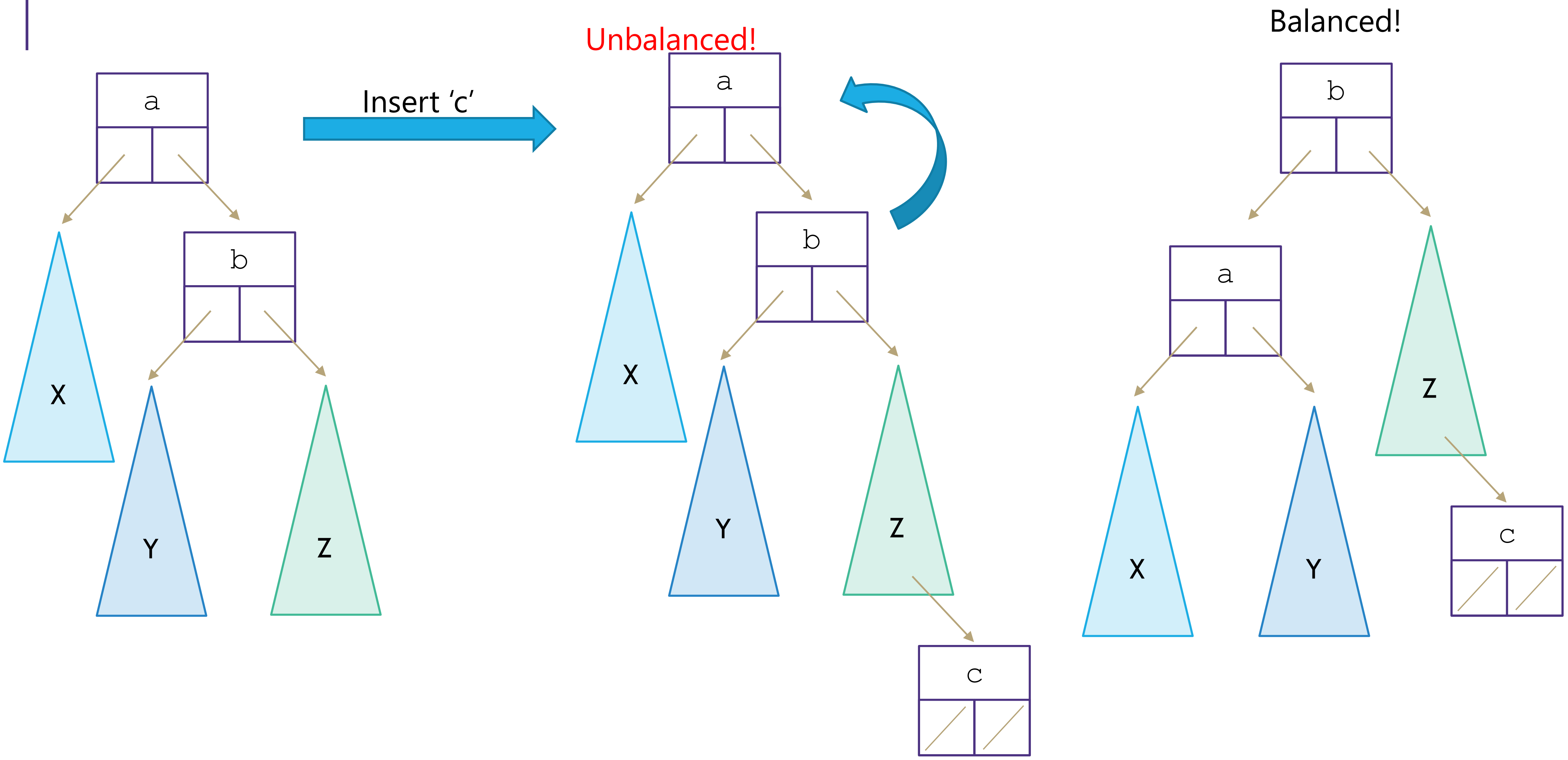
containsKey() – same as BST

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

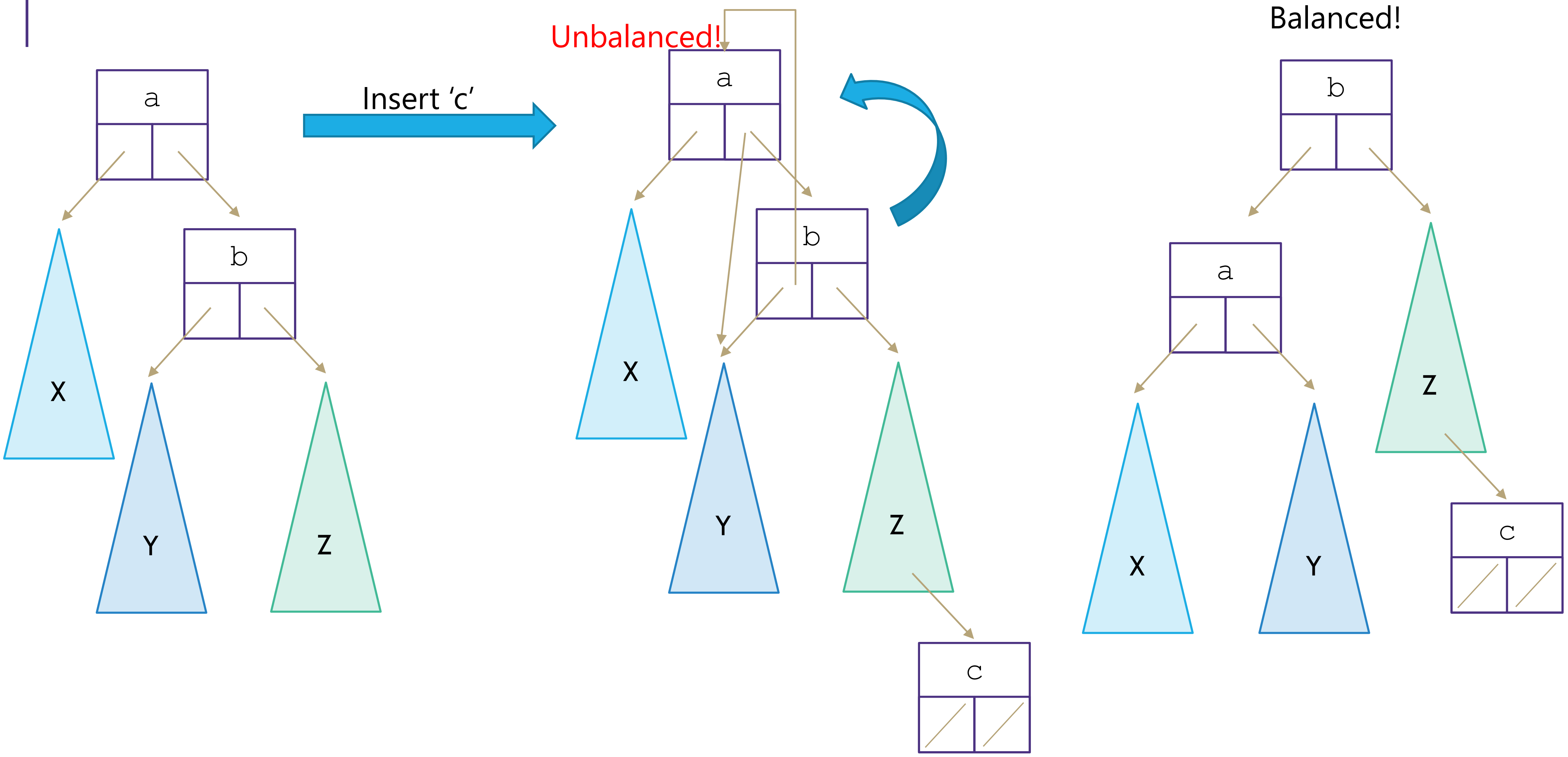
remove() - Replace the node to keep BST, fix AVL property if necessary



Rotations!

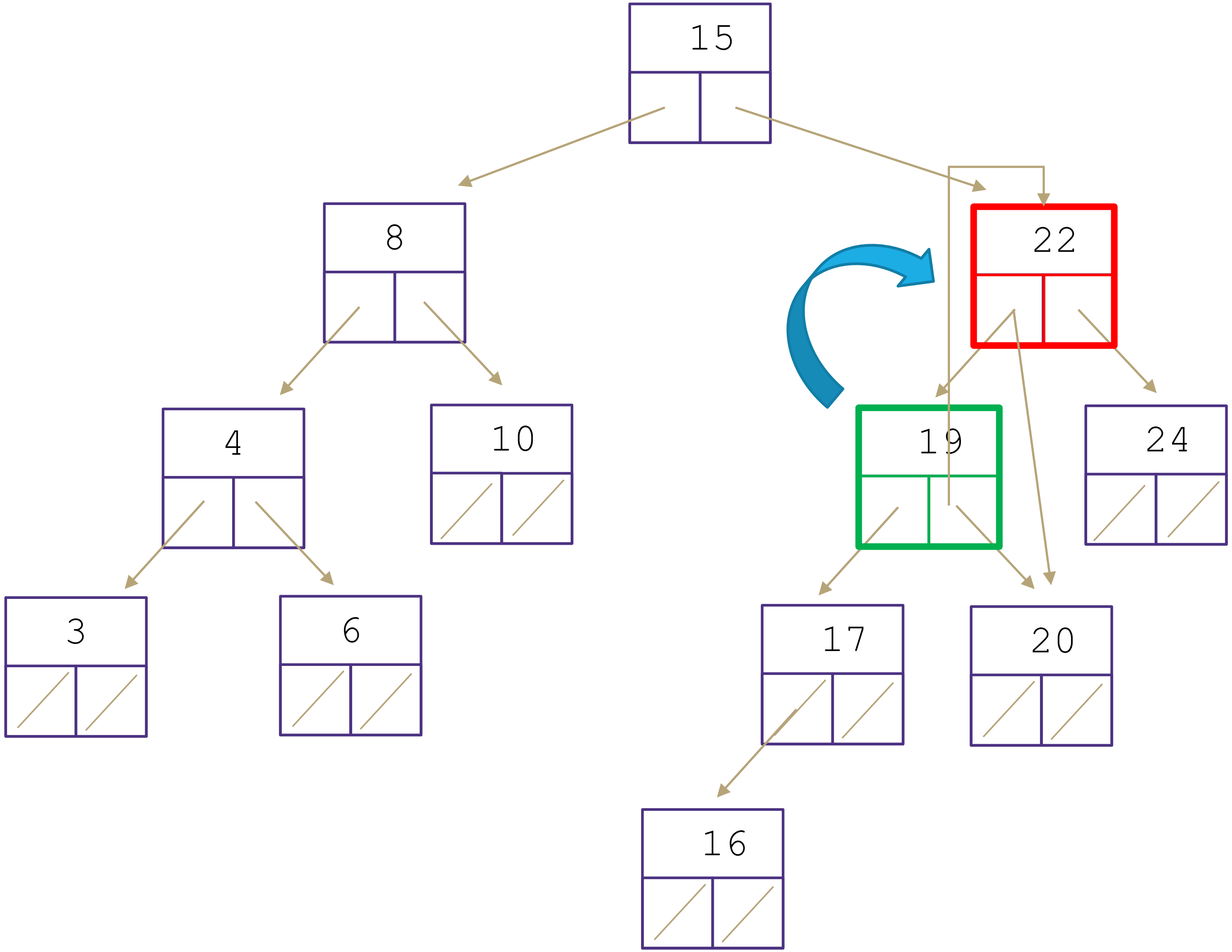


Rotations!



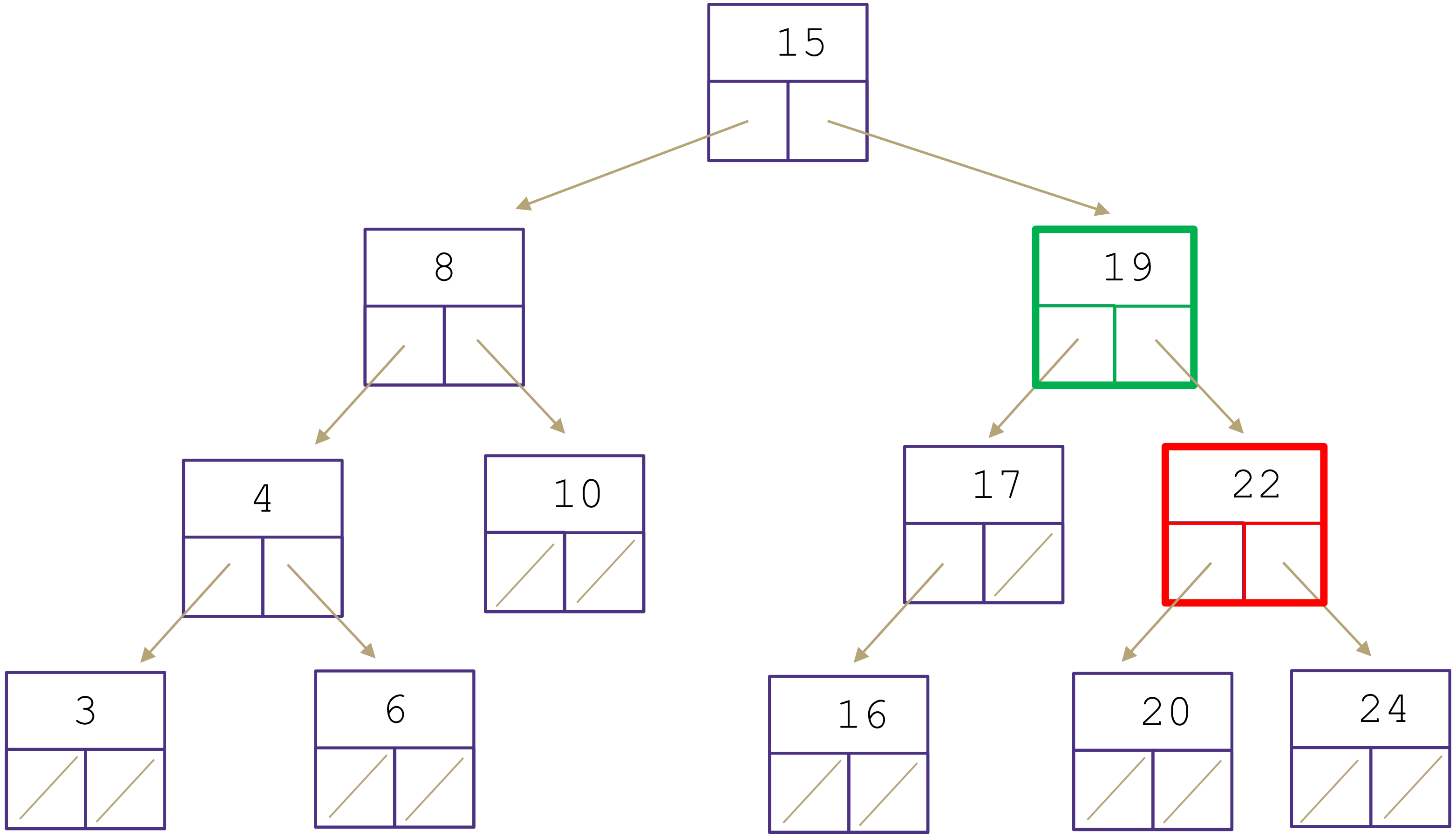
Practice

put(16);

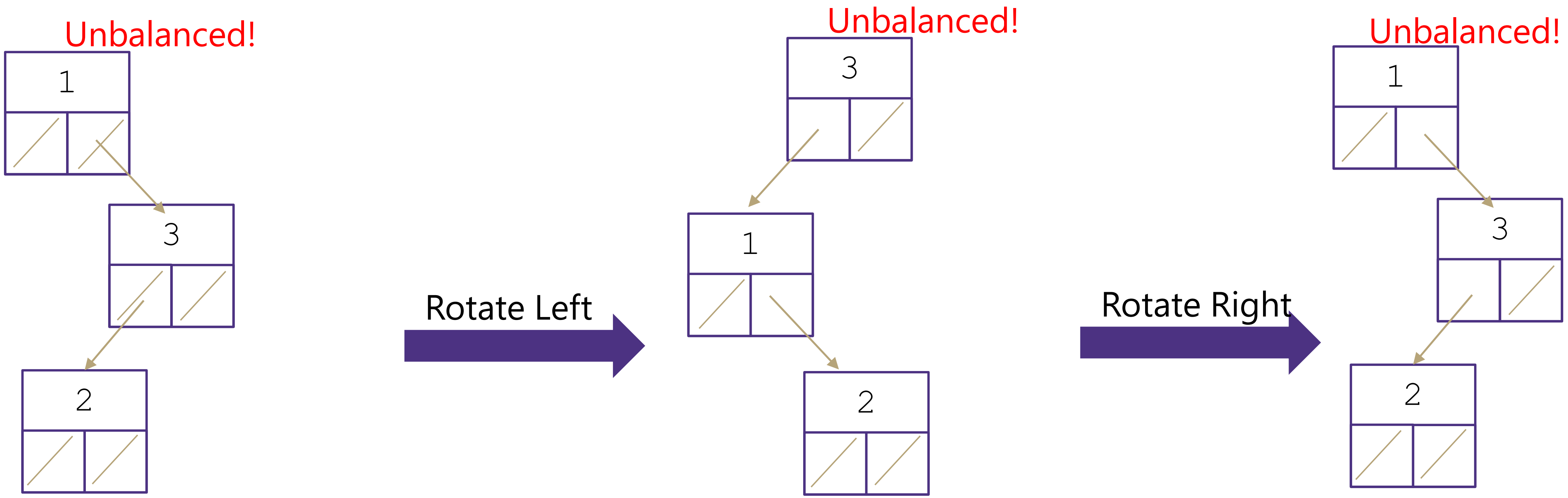


Practice

put(16);



So much can go wrong



Unbalanced!

Rotate Left

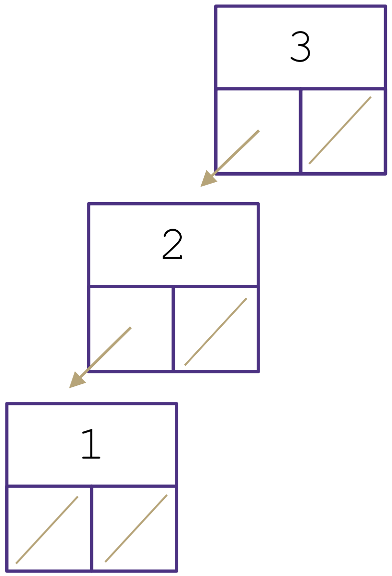
Unbalanced!

Rotate Right

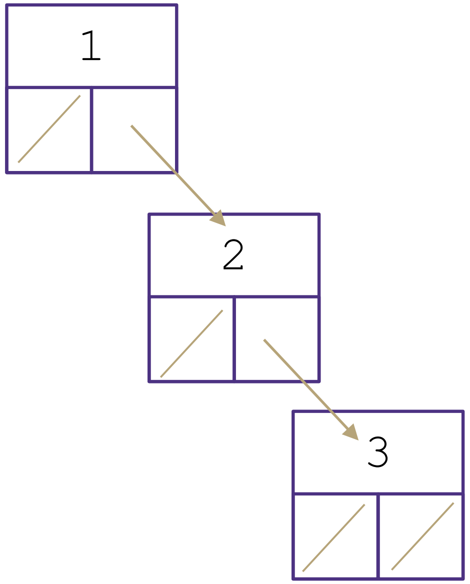
Unbalanced!

Two AVL Cases

Line Case
Solve with 1 rotation

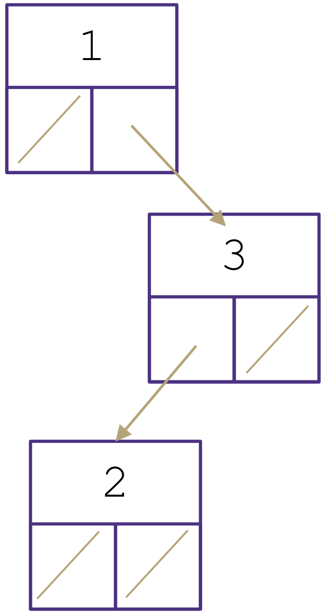


Rotate Right
Parent's left becomes child's right
Child's right becomes its parent

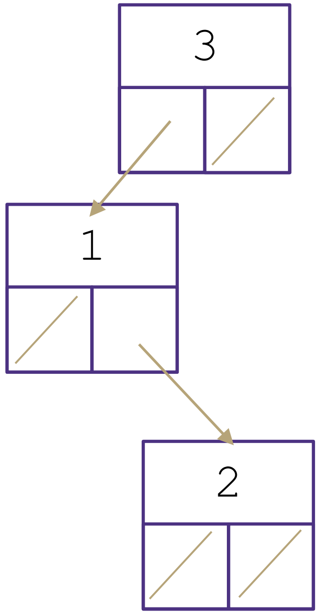


Rotate Left
Parent's right becomes child's left
Child's left becomes its parent

Kink Case
Solve with 2 rotations

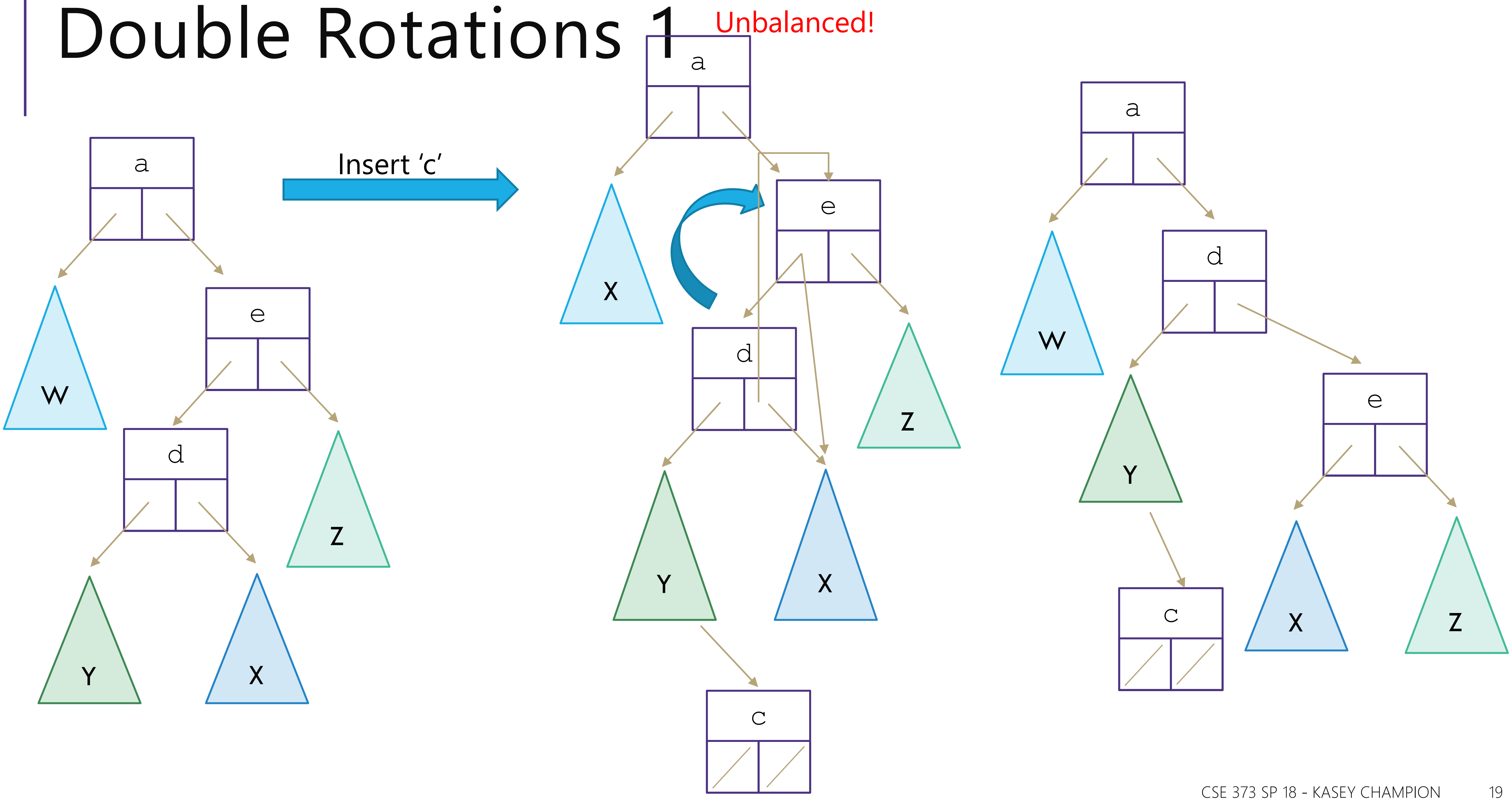


Rotate subtree left
Rotate root tree right



Rotate subtree right
Rotate root tree left

Double Rotations 1



Double Rotations 2