CPSC 221
Basic Algorithms and Data Structures

Balanced BST (AVL Trees)

Textbook References:
Koffman:11.1, 11.2

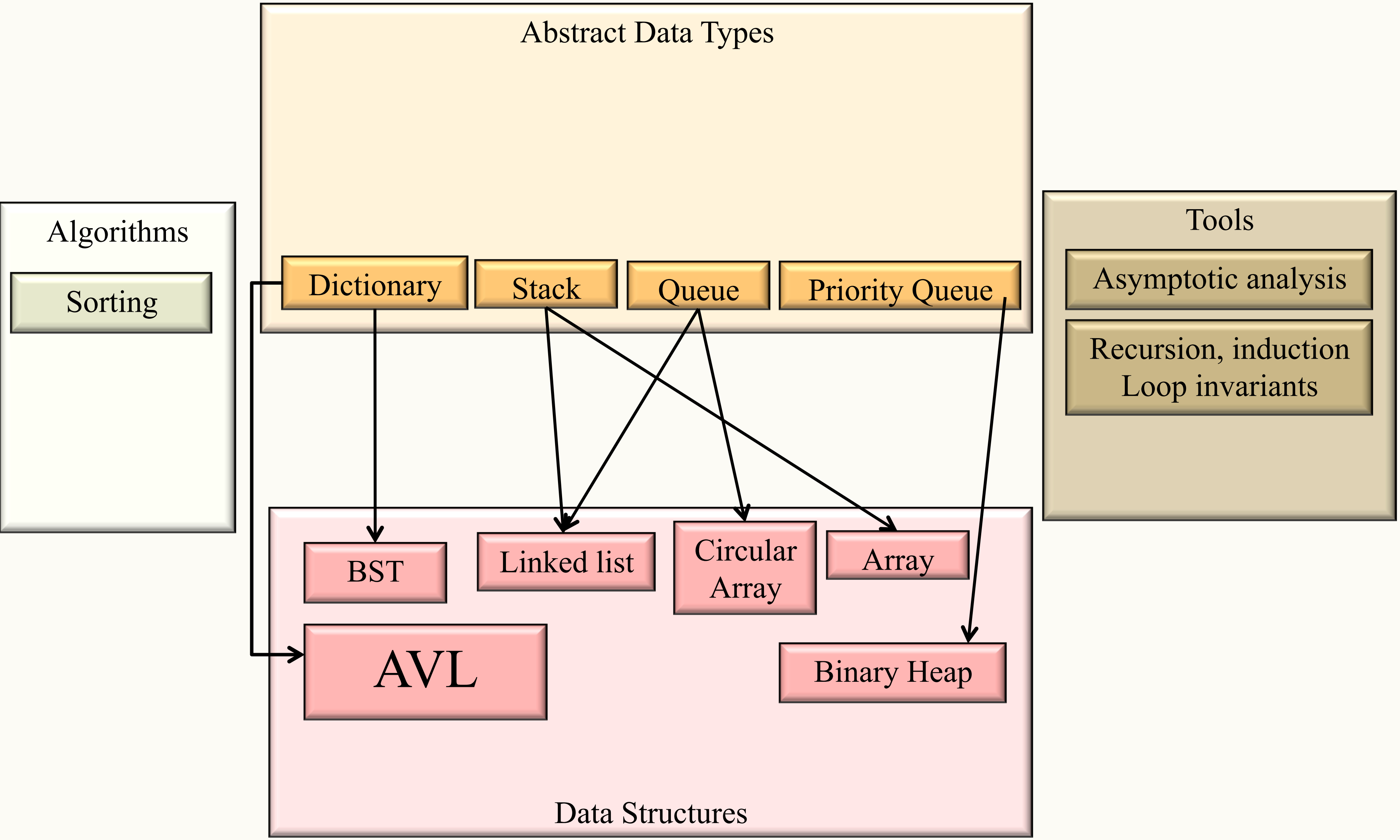
Hassan Khosravi
January - April 2015

(Borrowing many slides from Alan Hu and Steve Wolfman)

Learning goals

- Compare and contrast balanced/unbalanced trees.
- Describe and apply rotation to a BST to achieve a balanced tree.
- Recognize balanced binary search trees (among other tree types you recognize, e.g., heaps, general binary trees, general BSTs).

CPSC 221 Journey



CPSC Administrative Notes

- Written Assignment 2 is due March 20 (5pm)

- Labs
  - Currently doing lab 8, which is on AVL trees
  - Marking lab 7, which is on QuickSort
  - Starting lab 9, which is on Hashing (Friday Mar 20)

- PeerWise
  - Call #3 grades are available on Connect
  - Call #4 will be out soon

The bigger picture

- http://visualgo.net/bst.html

- Insert the following values into a BST
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

- Insert the following values into an AVL
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Beauty is Only Θ(log n) Deep

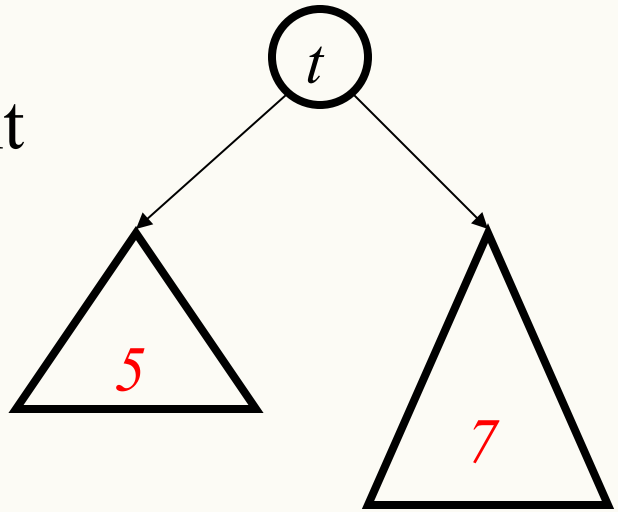
- Binary Search Trees are fast if they're shallow:
  - perfectly complete
  - perfectly complete except the one level fringe (like a heap)
  - anything else?

What matters here?

Problems occur when one subtree is much taller than the other!

Balance

- Balance
  - height(left subtree) - height(right subtree)
  - zero everywhere $\Rightarrow$ perfectly balanced
  - small everywhere $\Rightarrow$ balanced enough

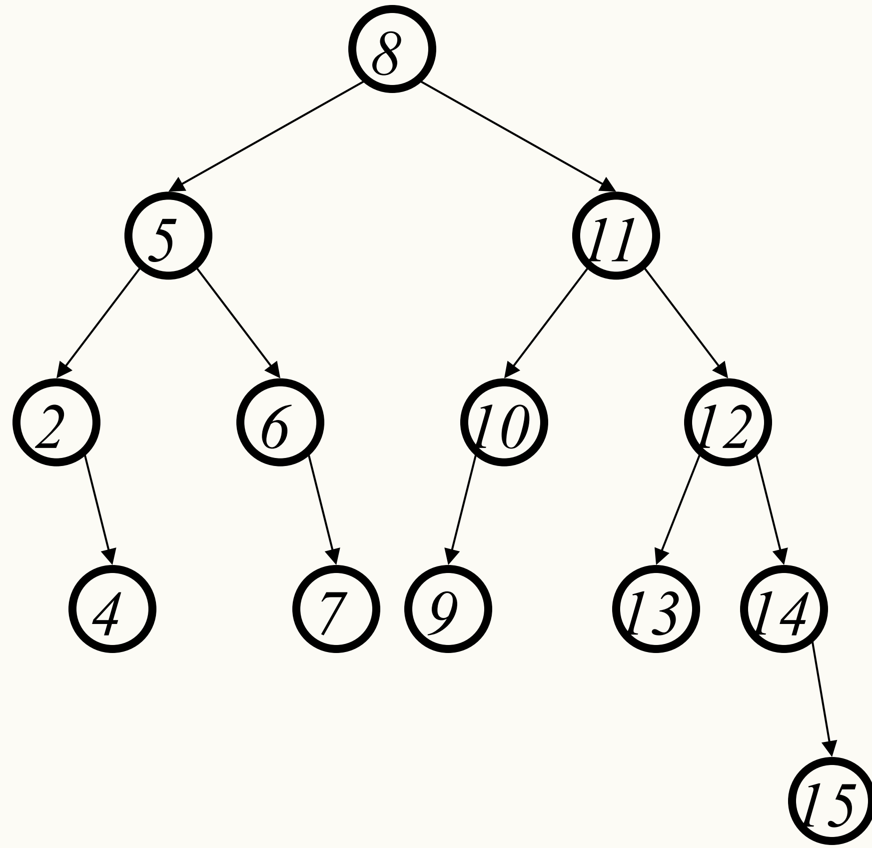


Balance between -1 and 1 everywhere
maximum height of ~1.44 lg n

AVL Tree
Dictionary Data Structure

- Binary search tree properties
  - binary tree invariant
  - search tree invariant
- Balance invariant
  - balance of every node is:

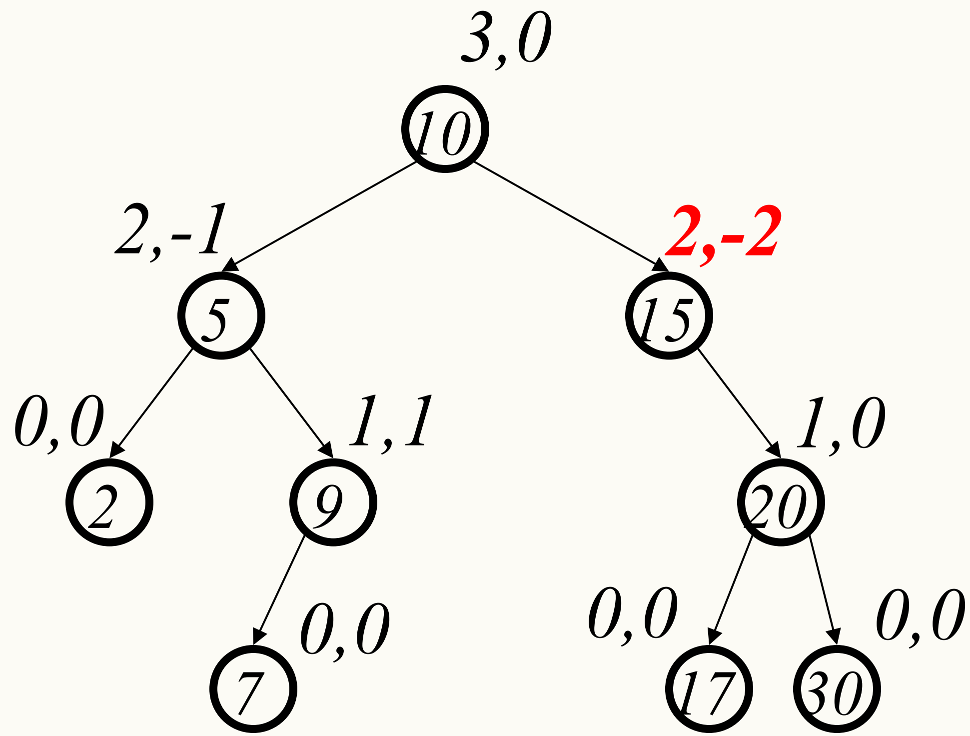
    $-1 \le b \le 1$
  - result:
    - depth is Θ(log n)



Note that (statically… ignoring how it updates) an AVL tree ***is*** *a BST.*

Testing the Balance Property

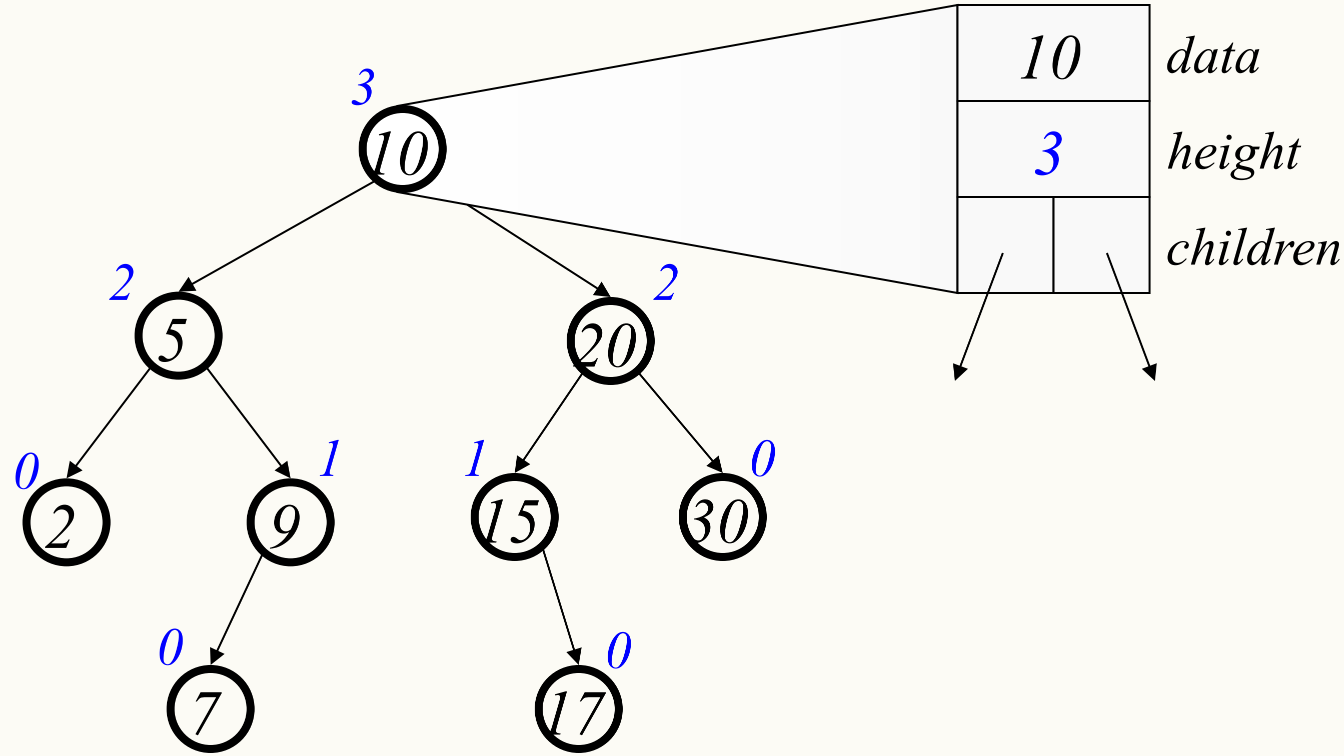
How do we **track the balance**?



NULL**s have height **-1

***FIRST** calculate heights*
***THEN** calculate balances*

An AVL Tree



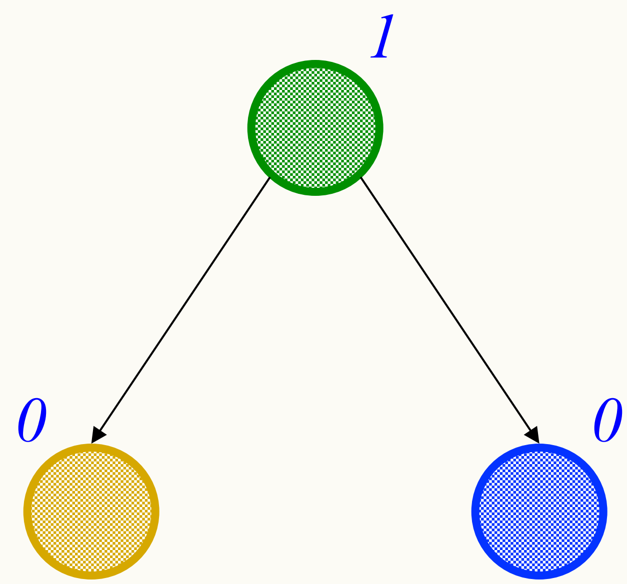
Adding a node can potential change the height of all of the nodes in that path

Beautiful Balance (SIMPLEST version)

Insert(middle)

Insert(small)

Insert(tall)



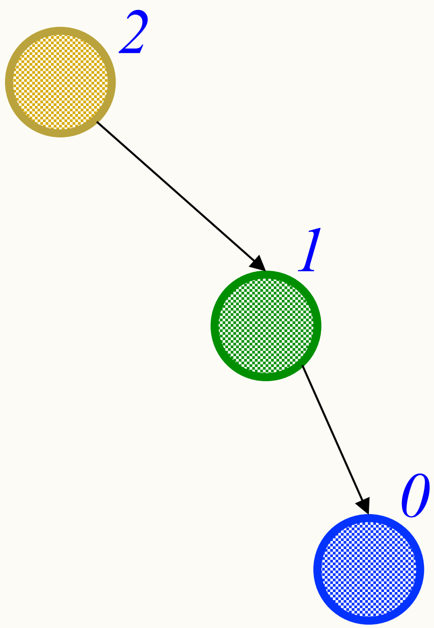
But… BSTs are under-constrained in unfortunate ways; ours may not look like this.

Bad Case #1 (SIMPLEST version)

Insert(small)

Insert(middle)

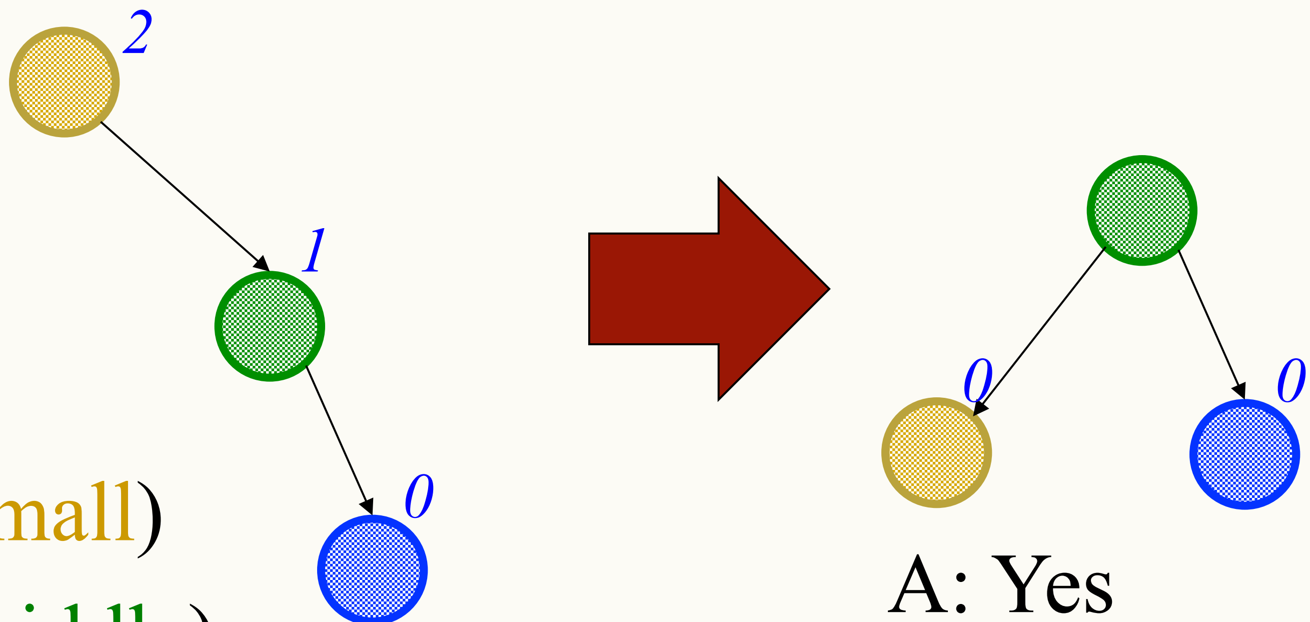
Insert(tall)



How do we fix the bad case?
How do we transition among different possible trees?

Clicker question

- Would the following rotation be valid?



Insert(small)

Insert(middle)

Insert(tall)

A: Yes

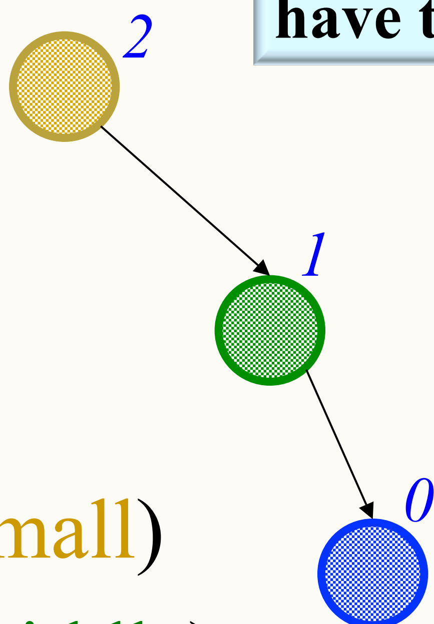
B: No

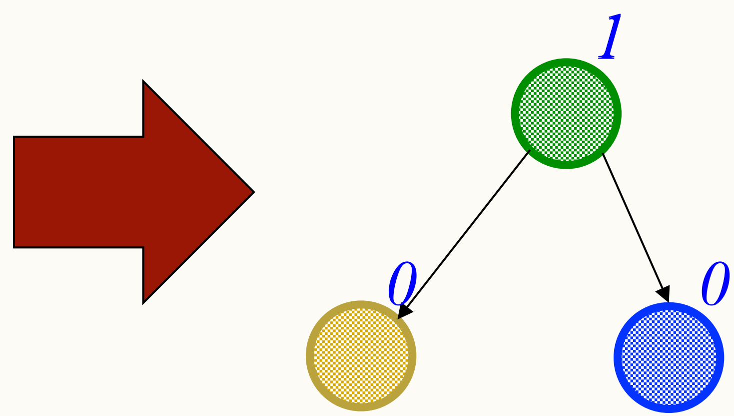
C: I don’t know

Single Rotation (SIMPLEST version)

- Would the following rotation be a valid?

Since this is a right child, it could legally have the parent as its left child.





Insert(small)

Insert(middle)

Insert(tall)

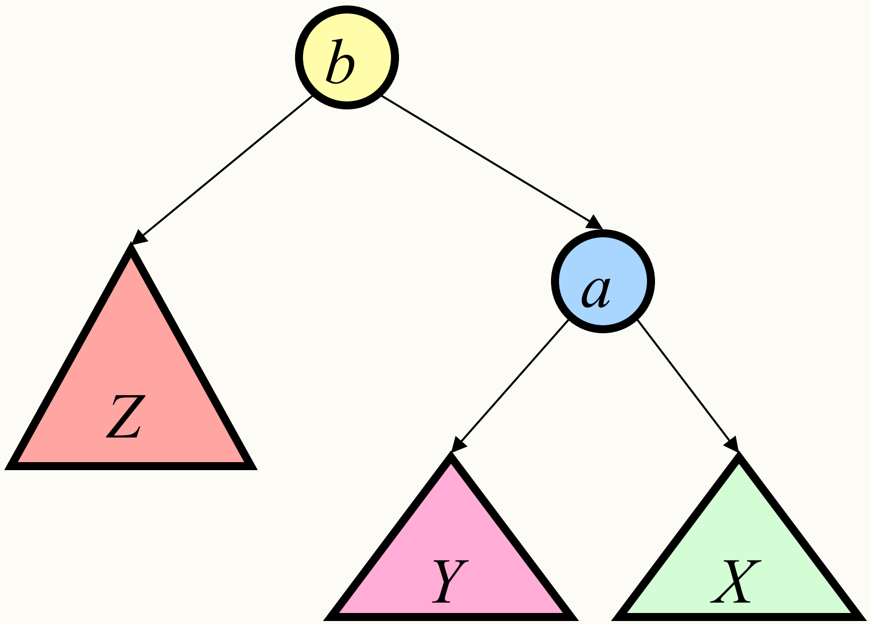
A: Yes

B: No

C: I don’t know

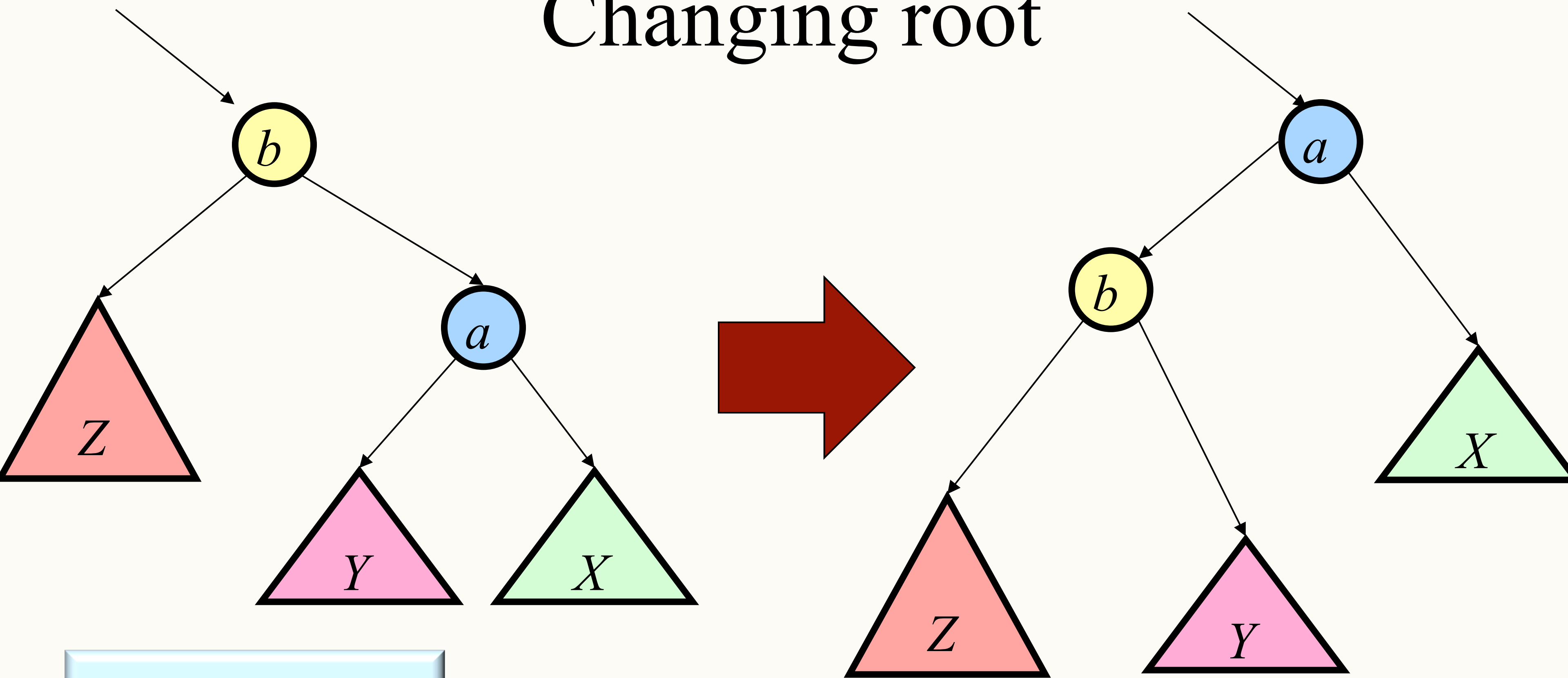
Changing root

- What's everything we know about nodes **a** and **b** and subtrees X,Y, and Z?
  - b < a
  - Z < b
  - Y < a
  - X > a



- How can we make **a** the root?

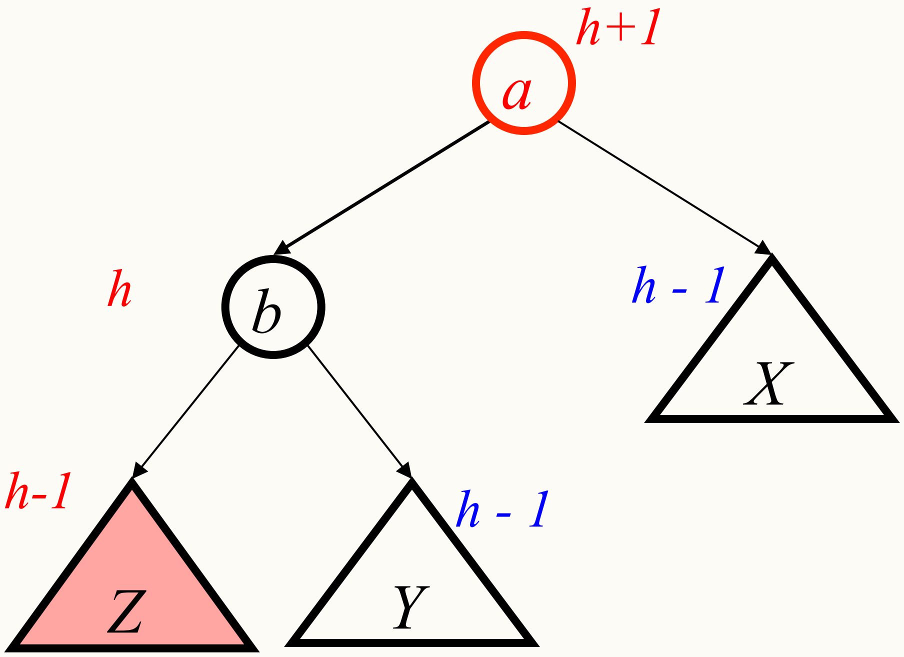
Changing root



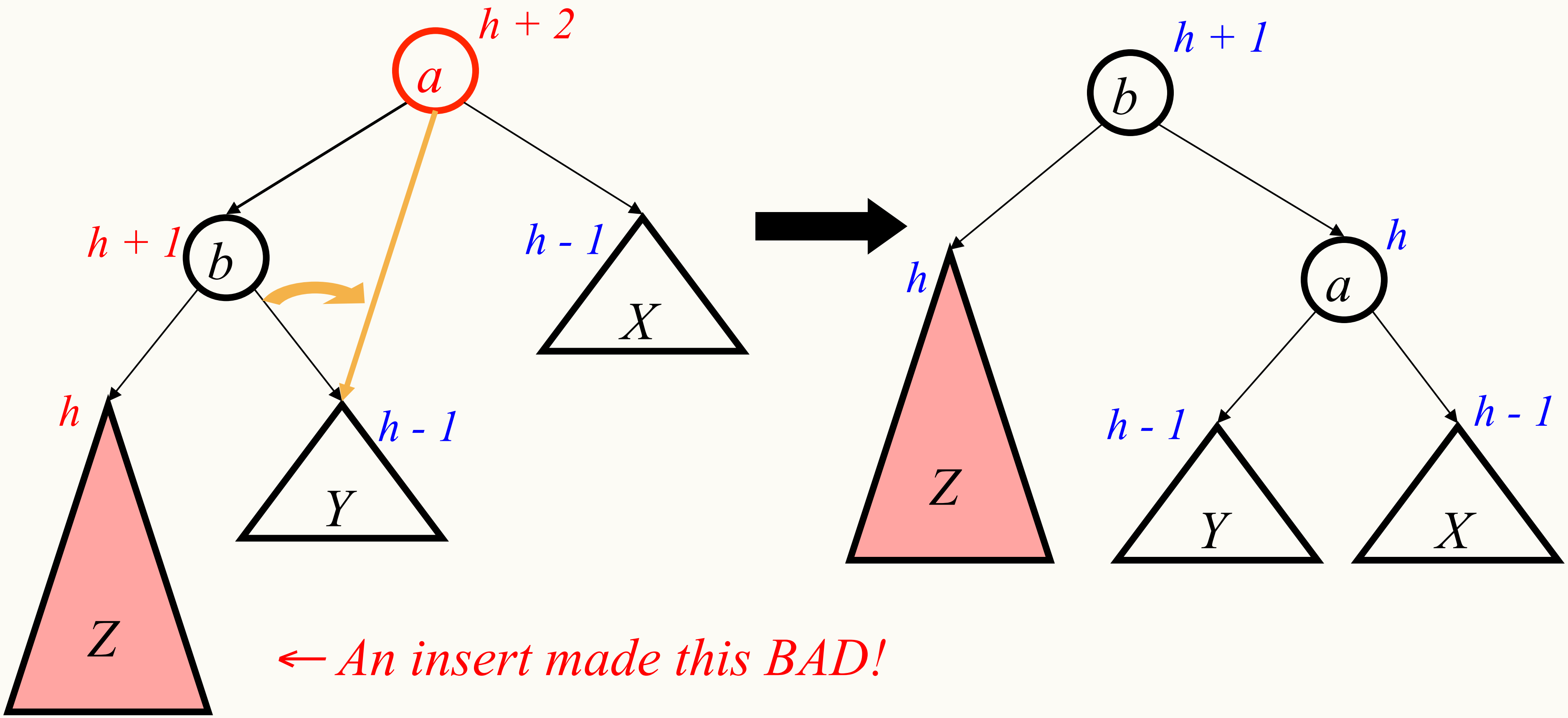
- b < a
- Z < b
- Y < a
- X > a

(1) Change left child of a to right child of b
(2) Change arrow b→ a to a → b
(3) Change the root pointer

Before Insertion (Single Rotation)



General Single Rotation



Why couldn't the bad insert be in X?

Time Complexity of Rotation?

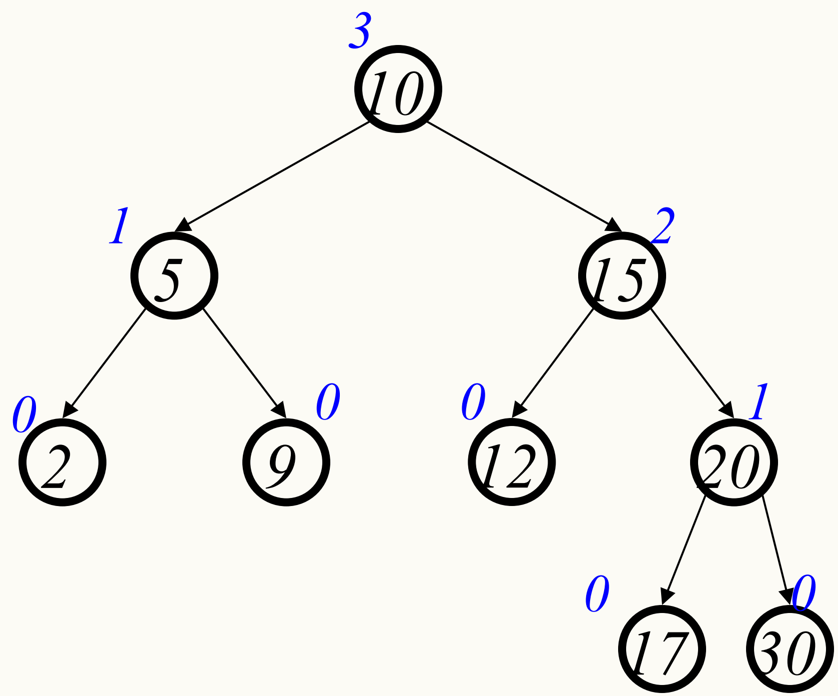
- $\Theta(1)$?
- $\Theta(\lg n)$?
- $\Theta(n)$?
- $\Theta(n \lg n)$?
- $\Theta(n^2)$?
- All of the above?

Time Complexity of Rotation?

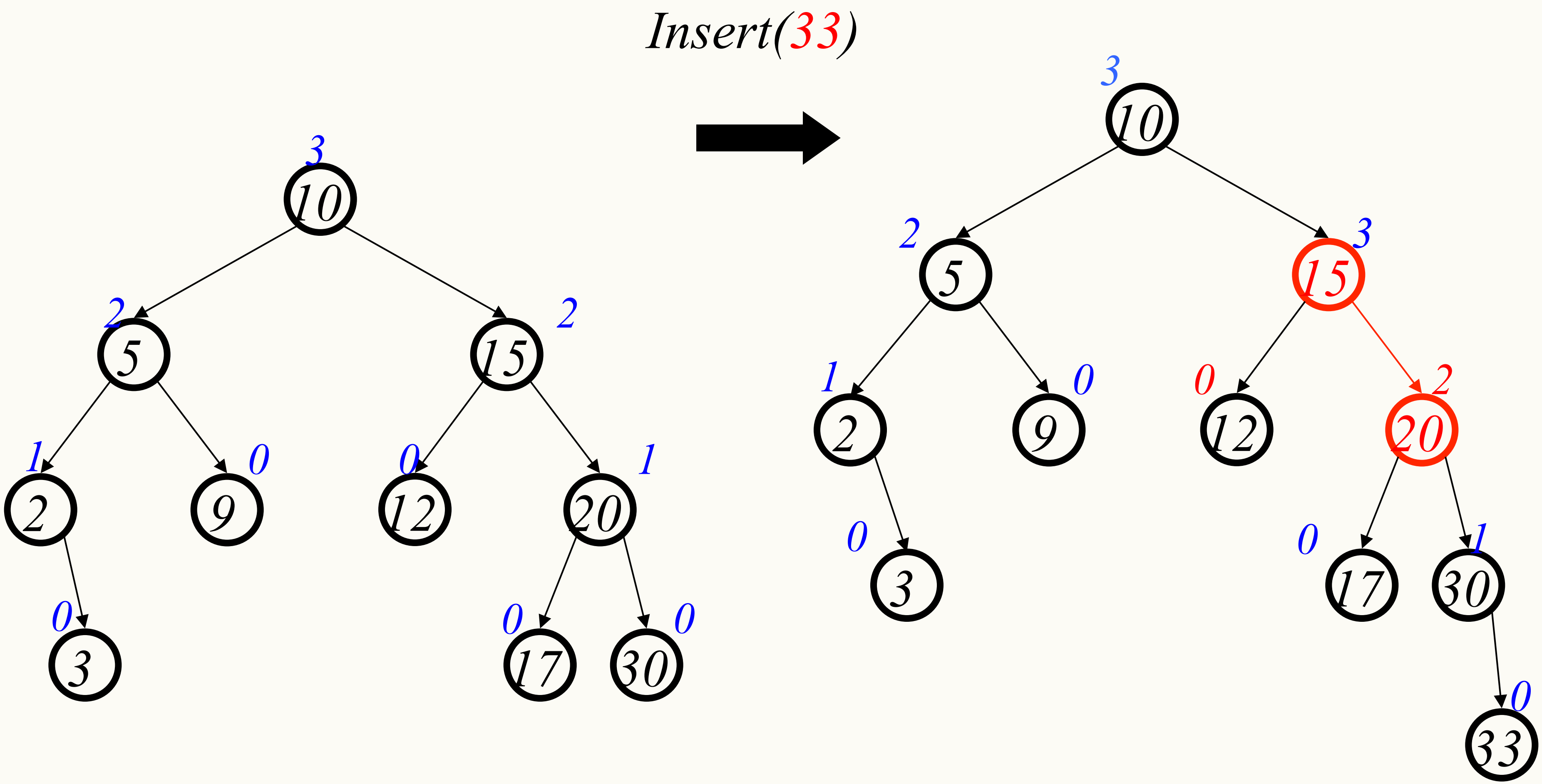
- $\Theta(1)$?
- $\Theta(\lg n)$?
- $\Theta(n)$?
- $\Theta(n \lg n)$?
- $\Theta(n^2)$?
- All of the above?

Example: Easy Insert

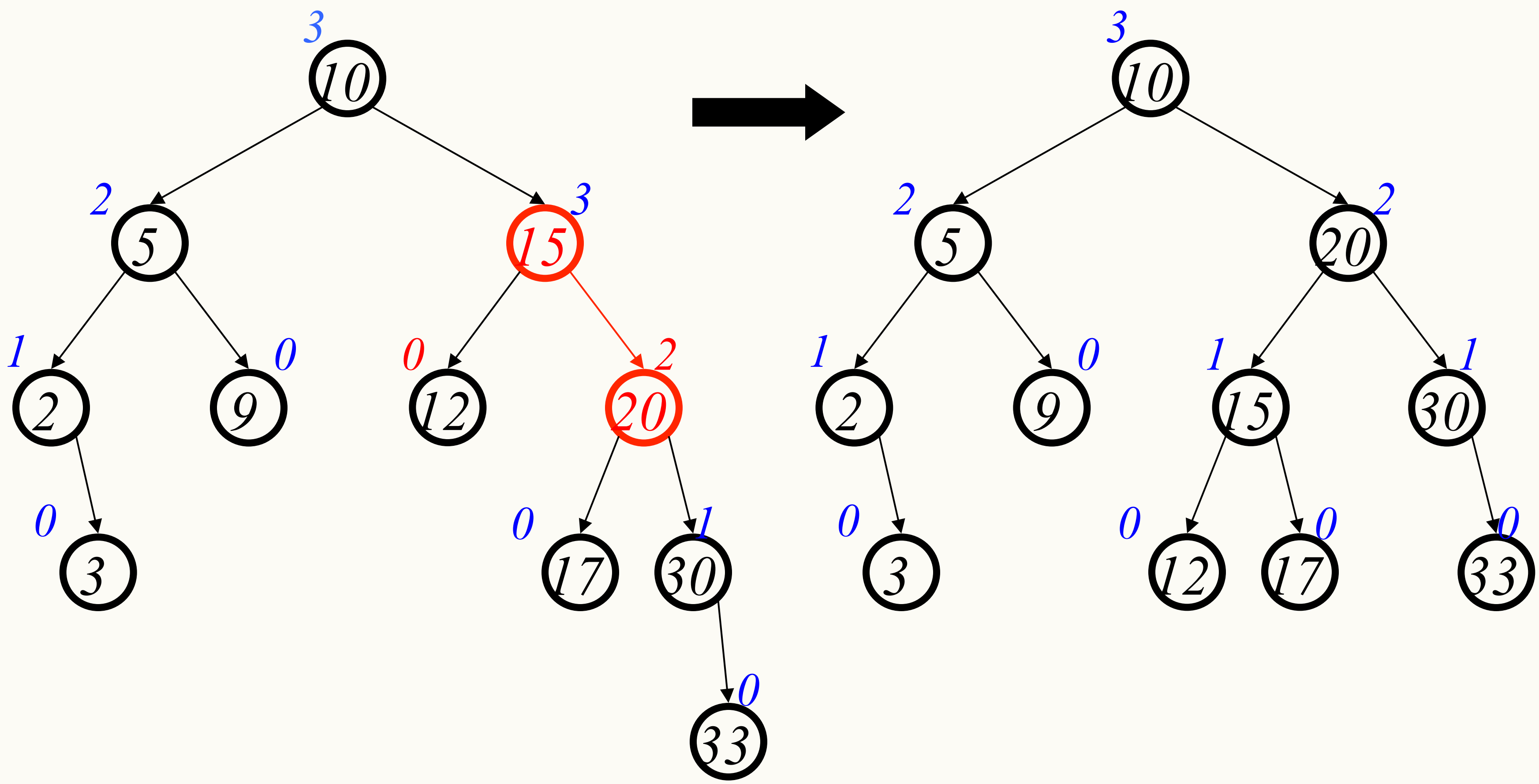
Insert(3)



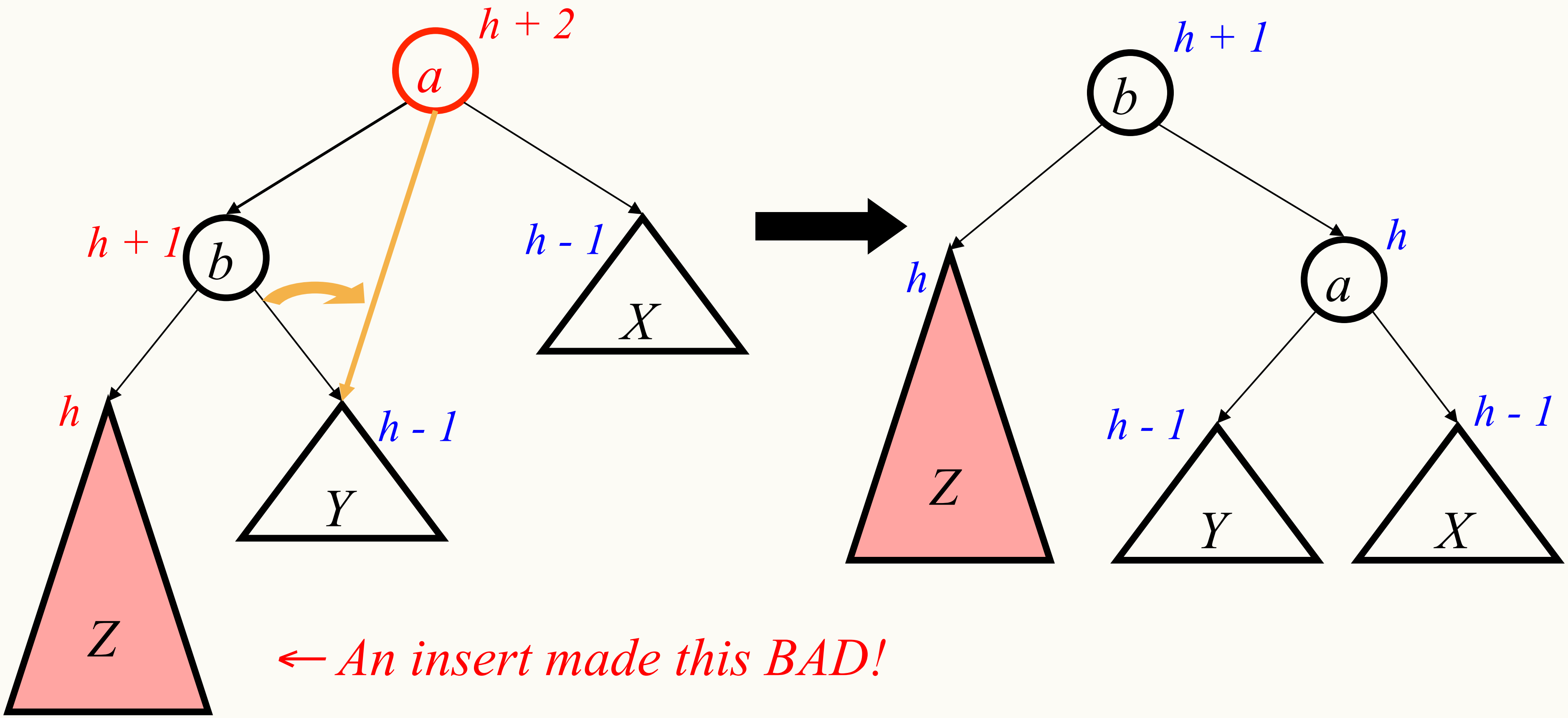
Hard Insert (Bad Case #1)



Single Rotation



General Single Rotation



← An insert made this BAD!

- After rotation, subtree's height same as before insert!
- Height of all ancestors unchanged.

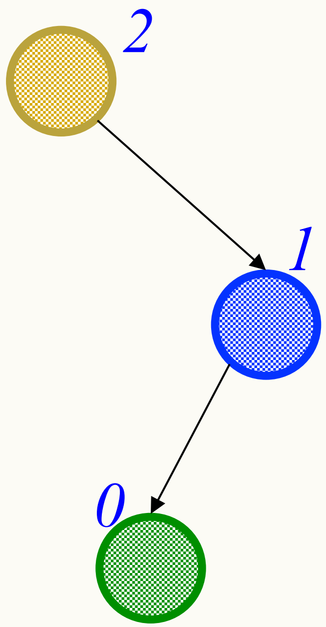
Why does it matter?

Bad Case #2 (SIMPLEST version)

Insert(small)

Insert(tall)

Insert(middle)



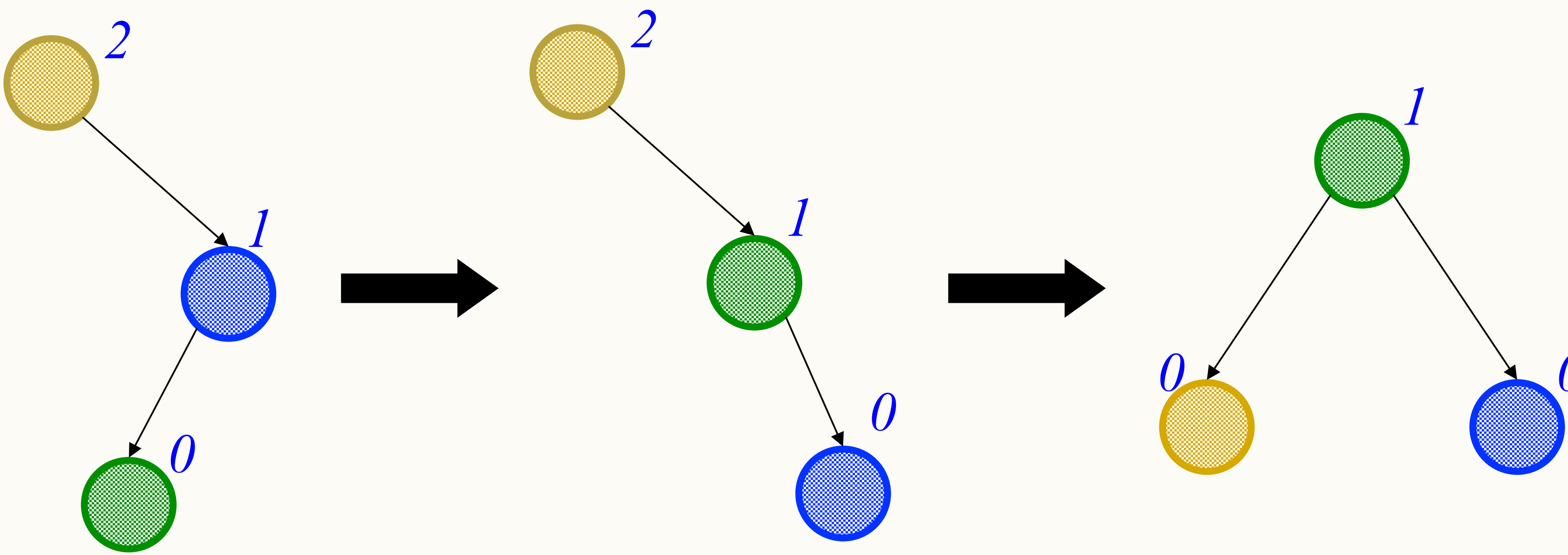
Try to balance this tree

Double Rotation (SIMPLEST version)

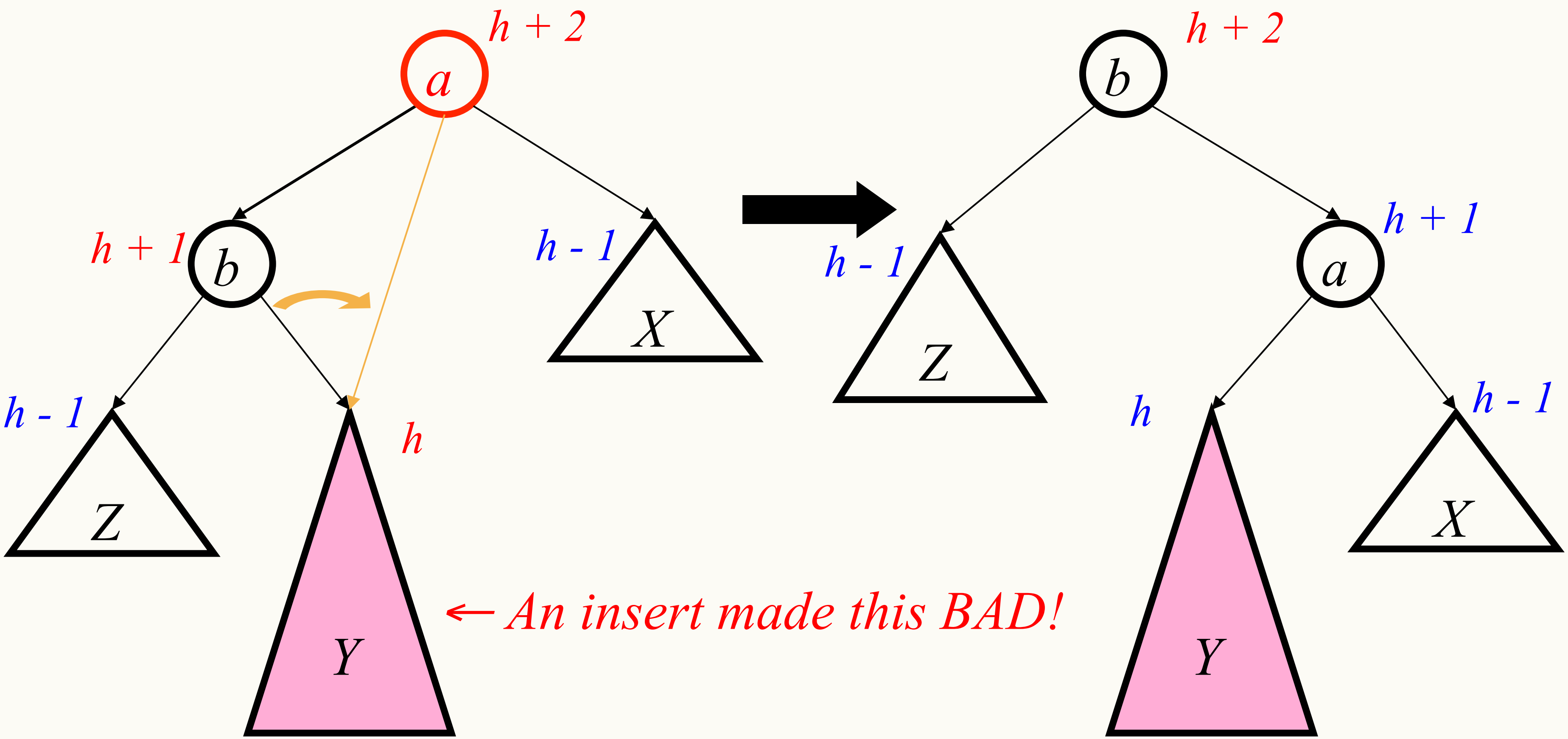
Insert(small)

Insert(tall)

Insert(middle)

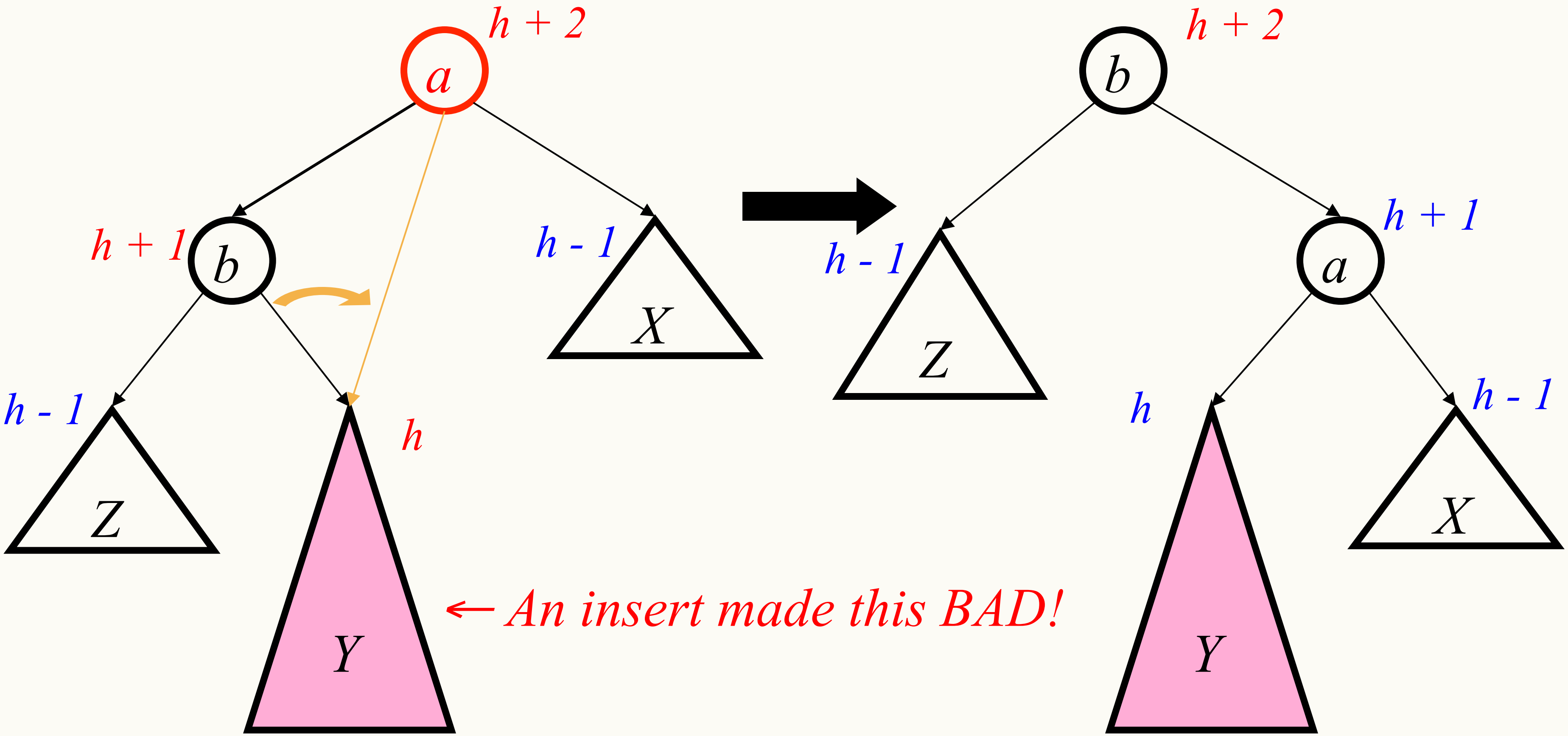


When Single Rotation Doesn't Help



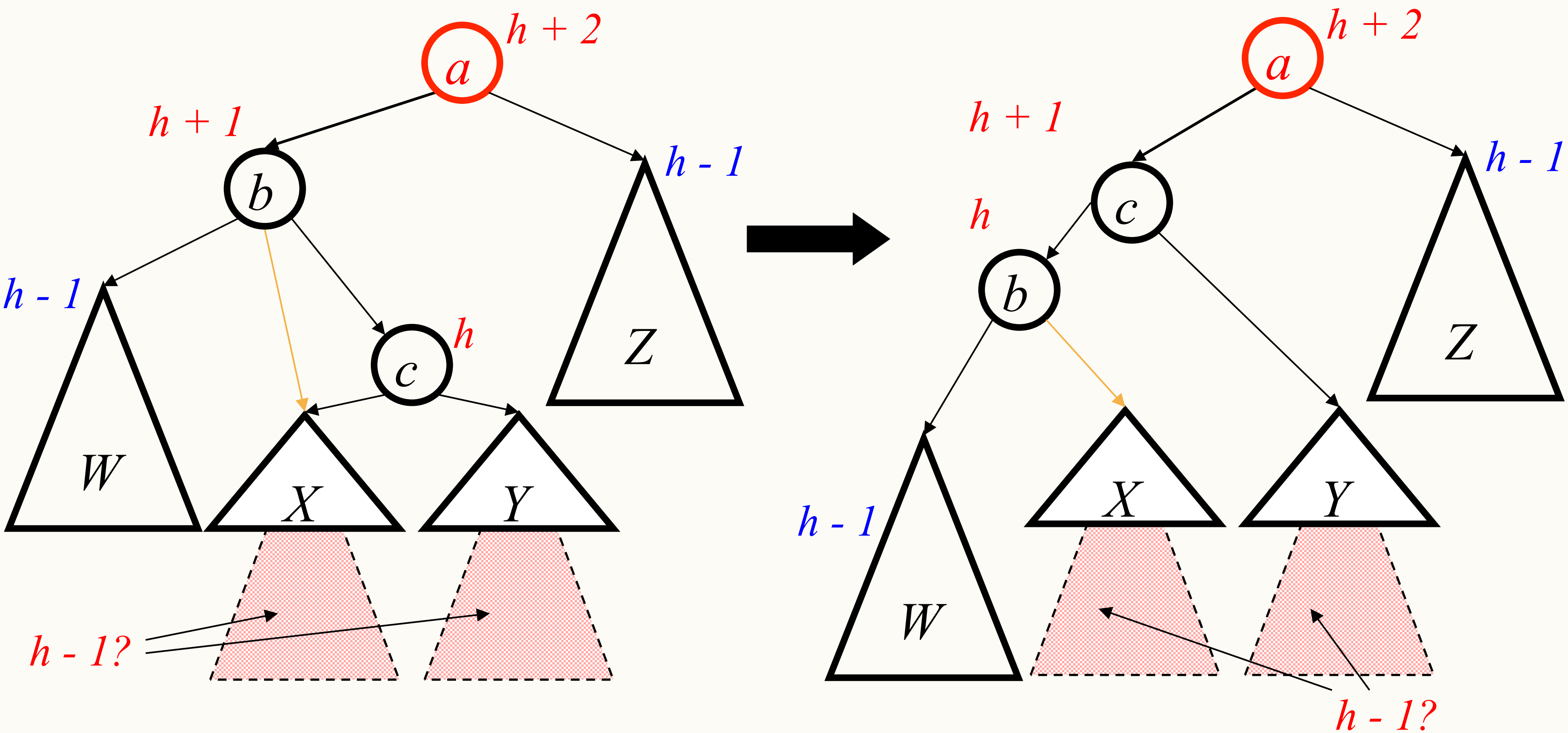
- After rotation, still unbalanced!
- What can you do?

When Single Rotation Doesn't Help



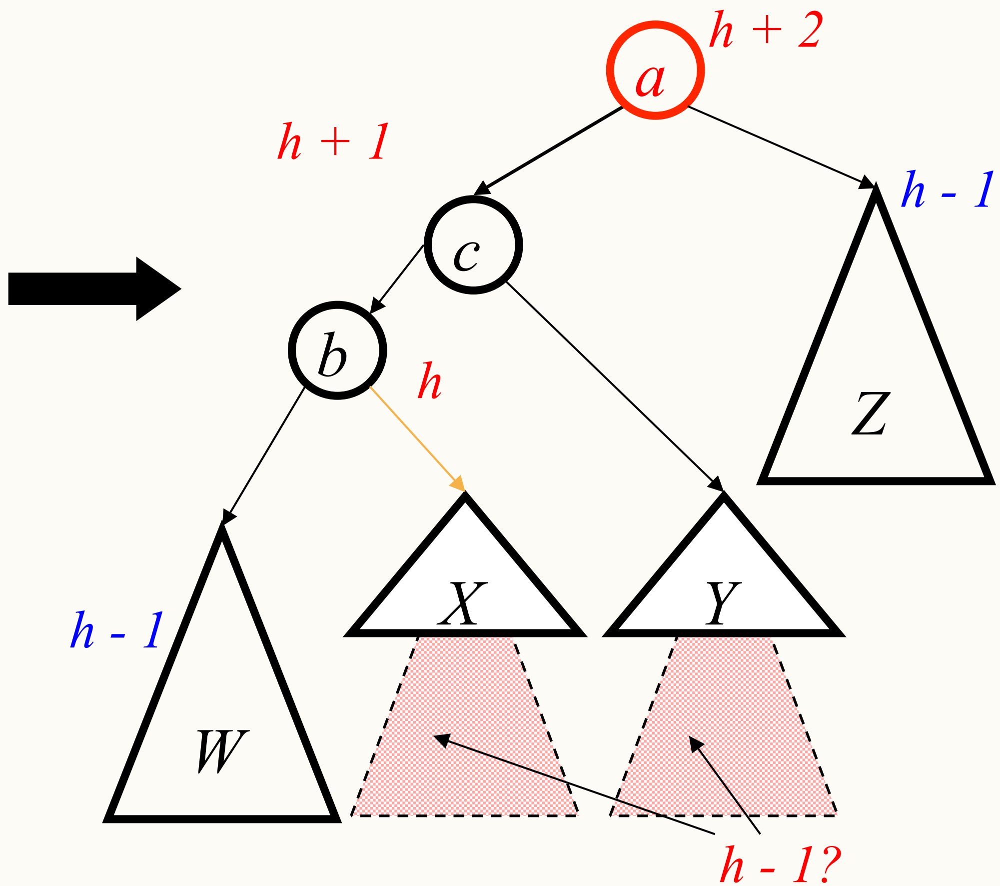
- After rotation, still unbalanced!
- The problem is Y is too heavy, so rotate stuff out of Y!

Double Rotation Part 1



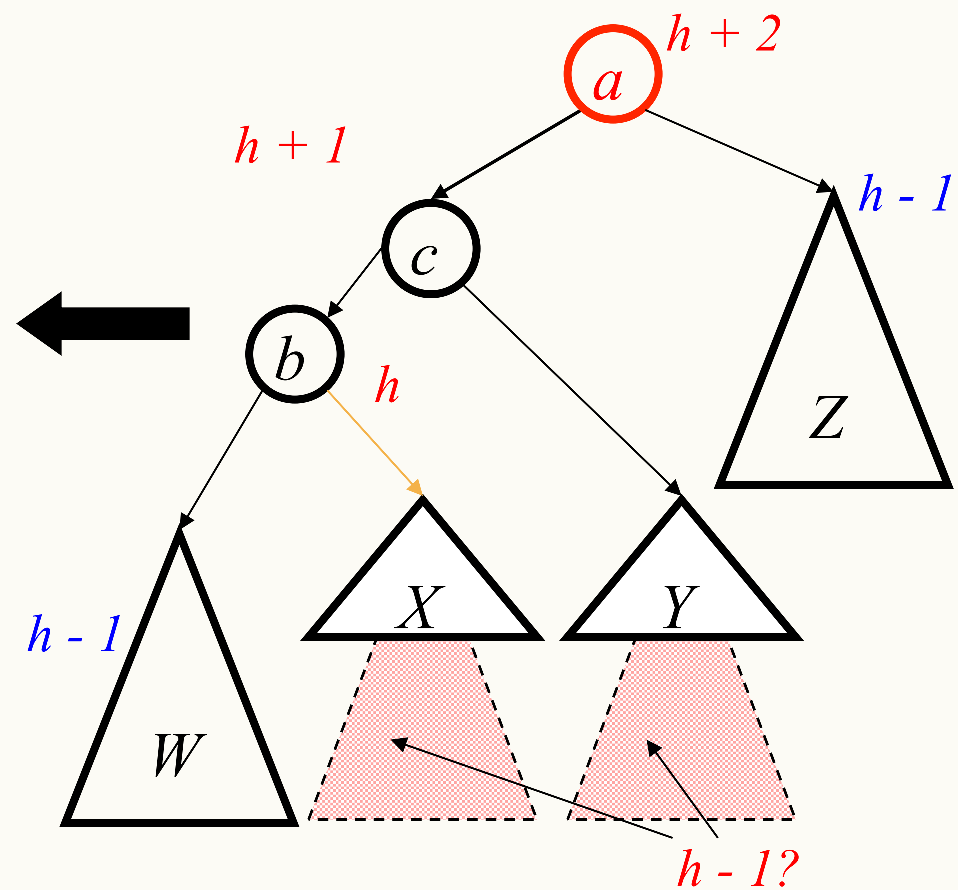
- First, do a single rotation farther down, to split up the big subtree.

Double Rotation Part 1



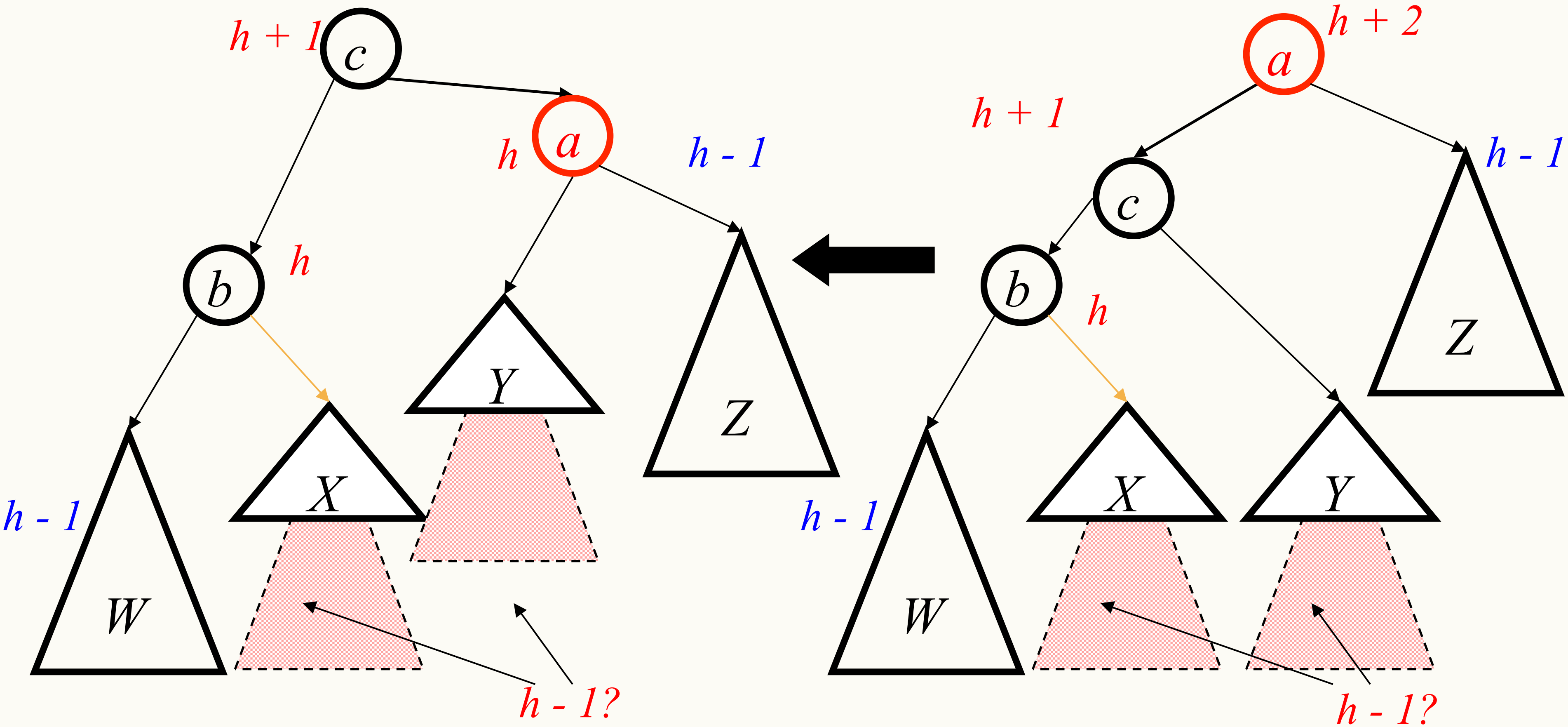
- First, do a single rotation farther down, to split up the big subtree.

Double Rotation Part 2



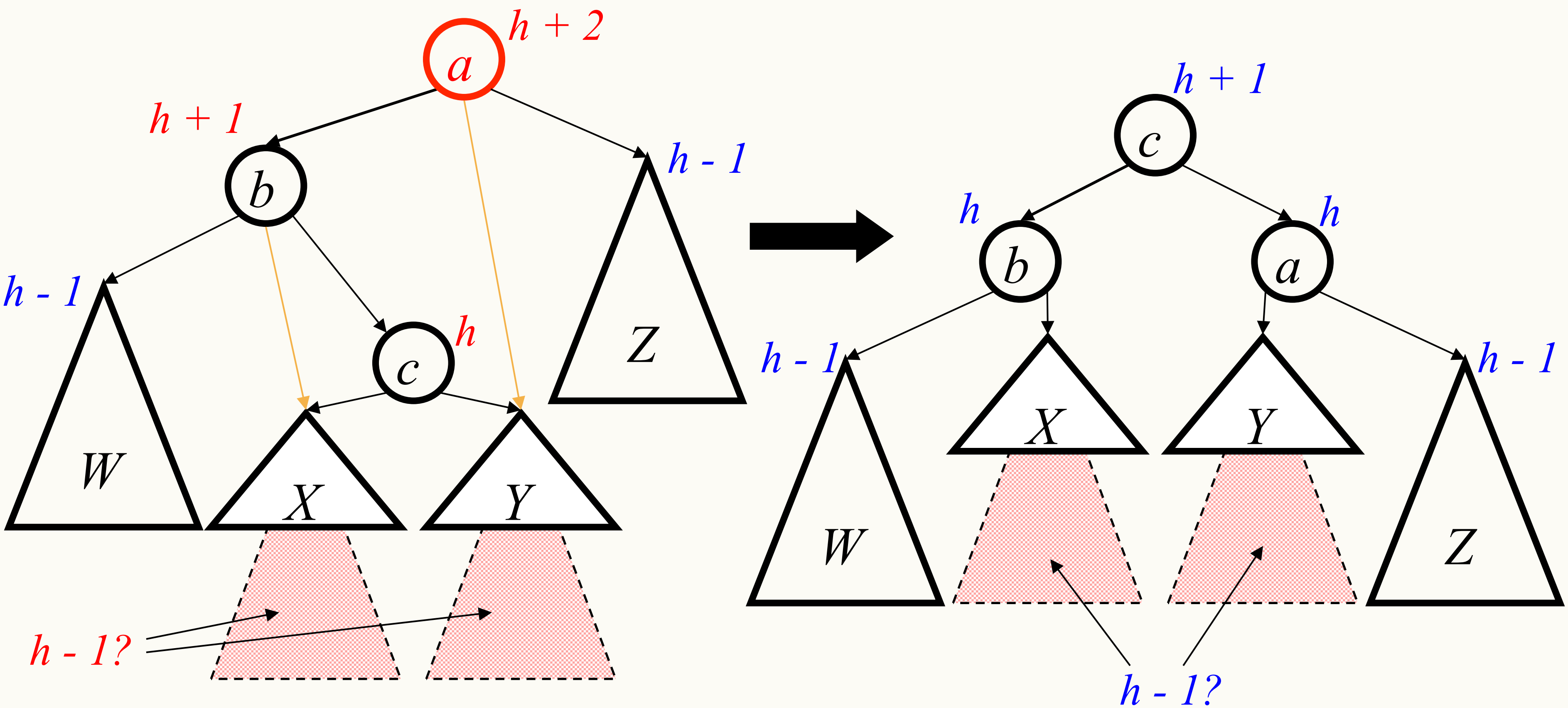
- Now, we can do the originally planned rotation, and not have too much height shift over…

Double Rotation Part 2



- Now, we can do the originally planned rotation, and not have too much height shift over…

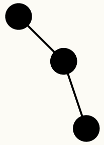
General Double Rotation



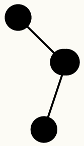
- *Height of subtree* ***still*** *the same as it was before insert!*
- *Height of all ancestors unchanged.*

Insert Algorithm

- Find spot for the new value
- Hang new node
- Search back up for imbalance
- If there is an imbalance:
  - case #1: Perform single rotation and exit



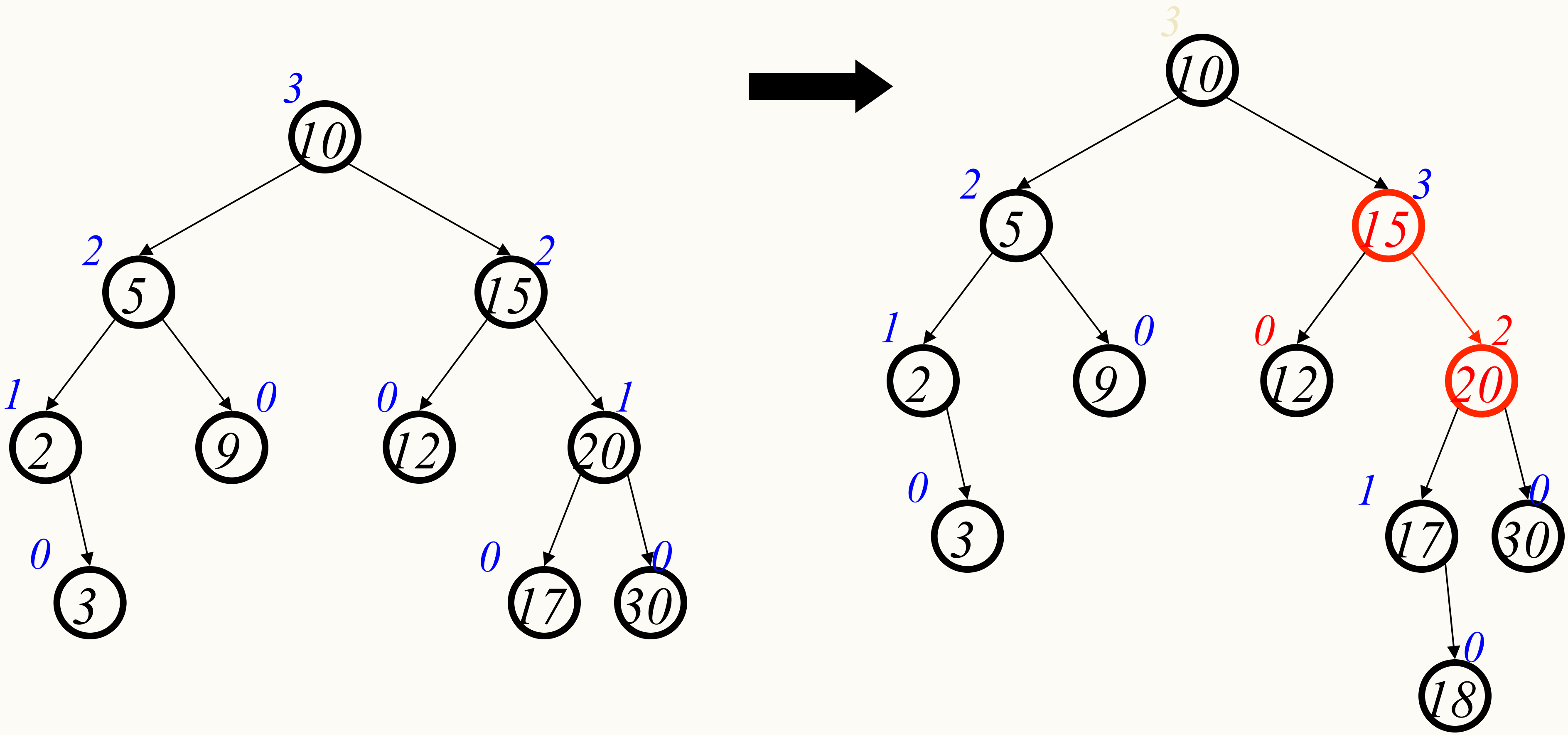
  - case #2: Perform double rotation and exit



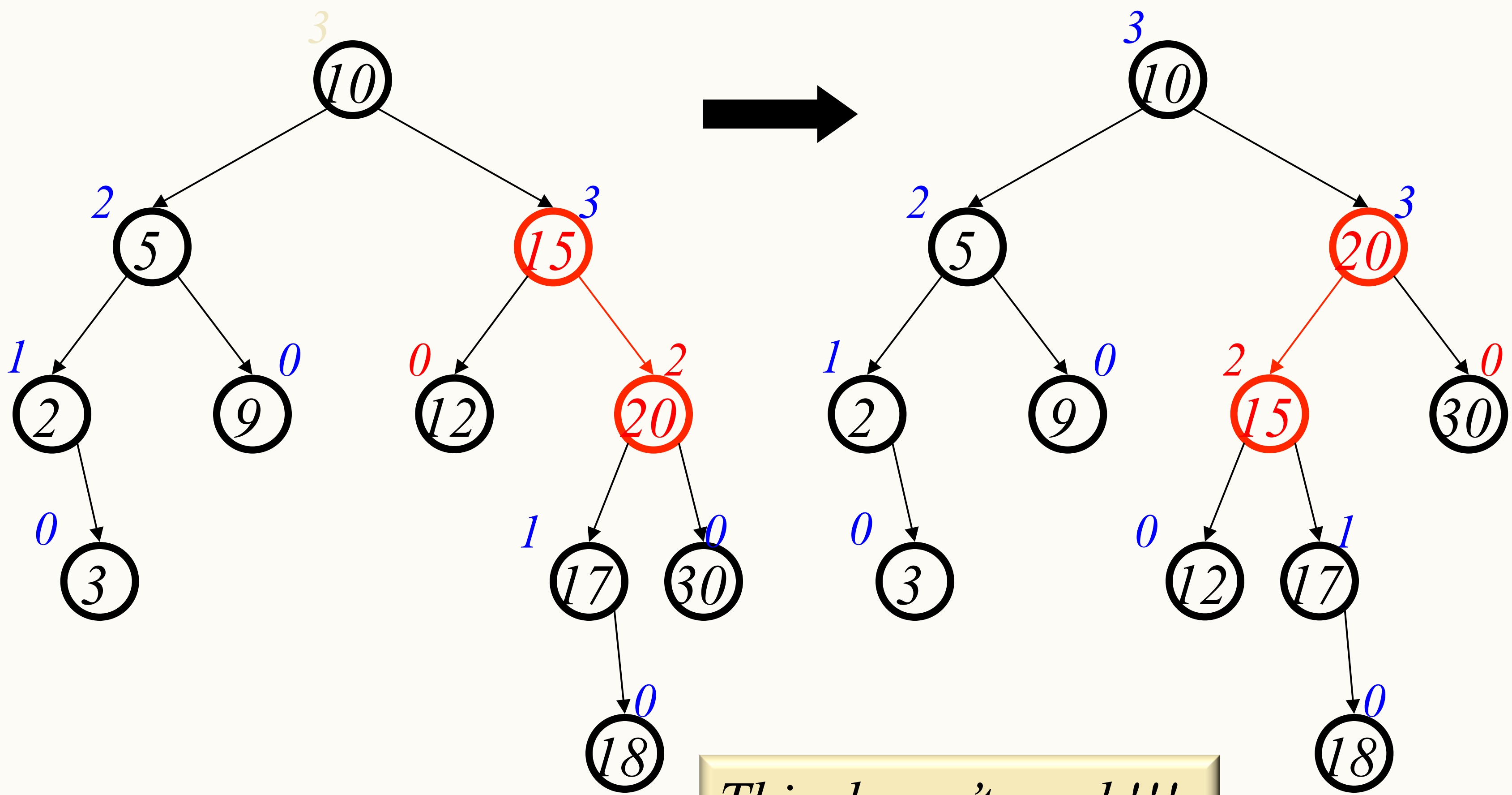
  - Mirrored cases also possible

Hard Insert (Bad Case #2)

Insert(18)

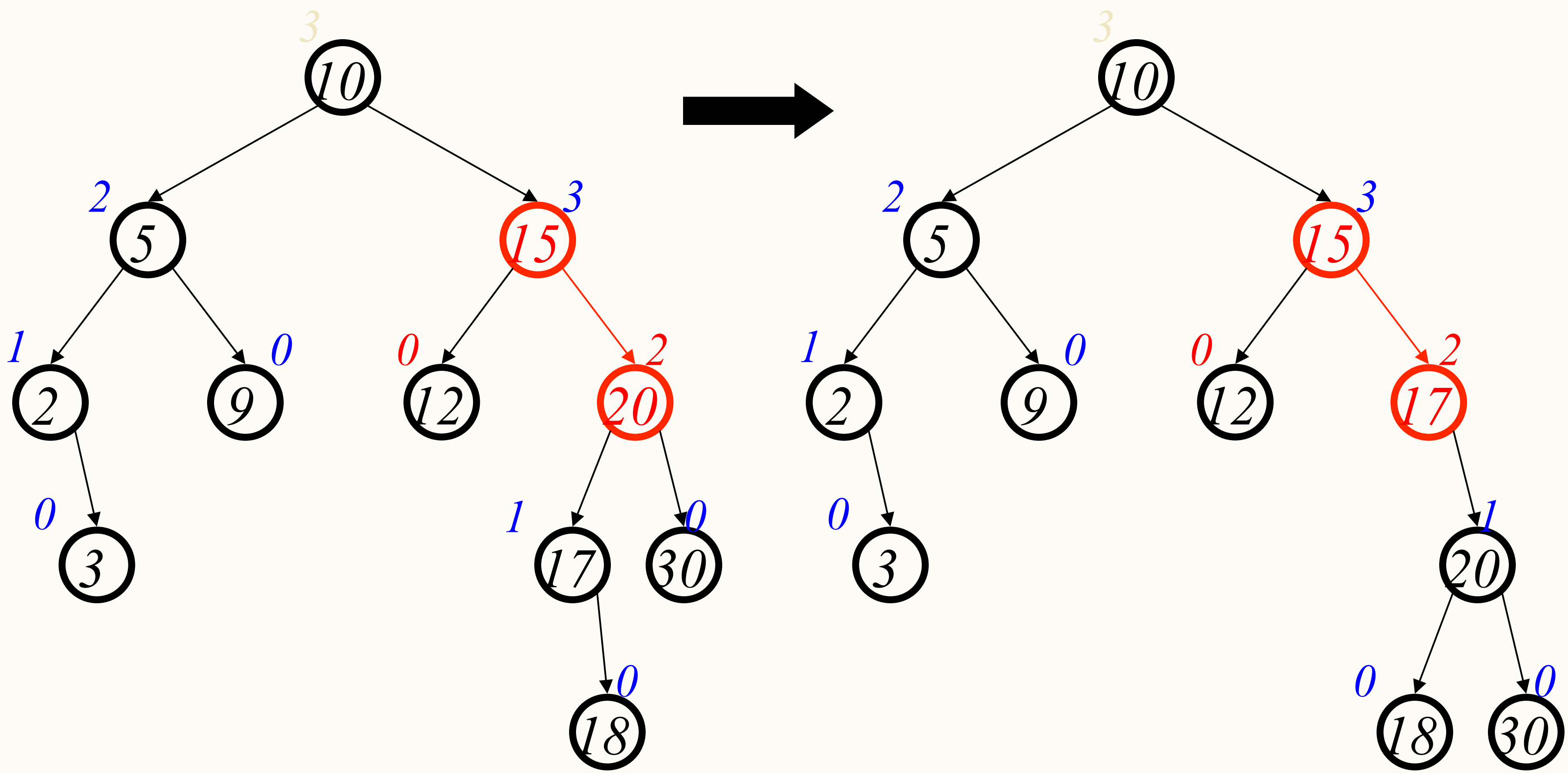


Single Rotation (oops!)



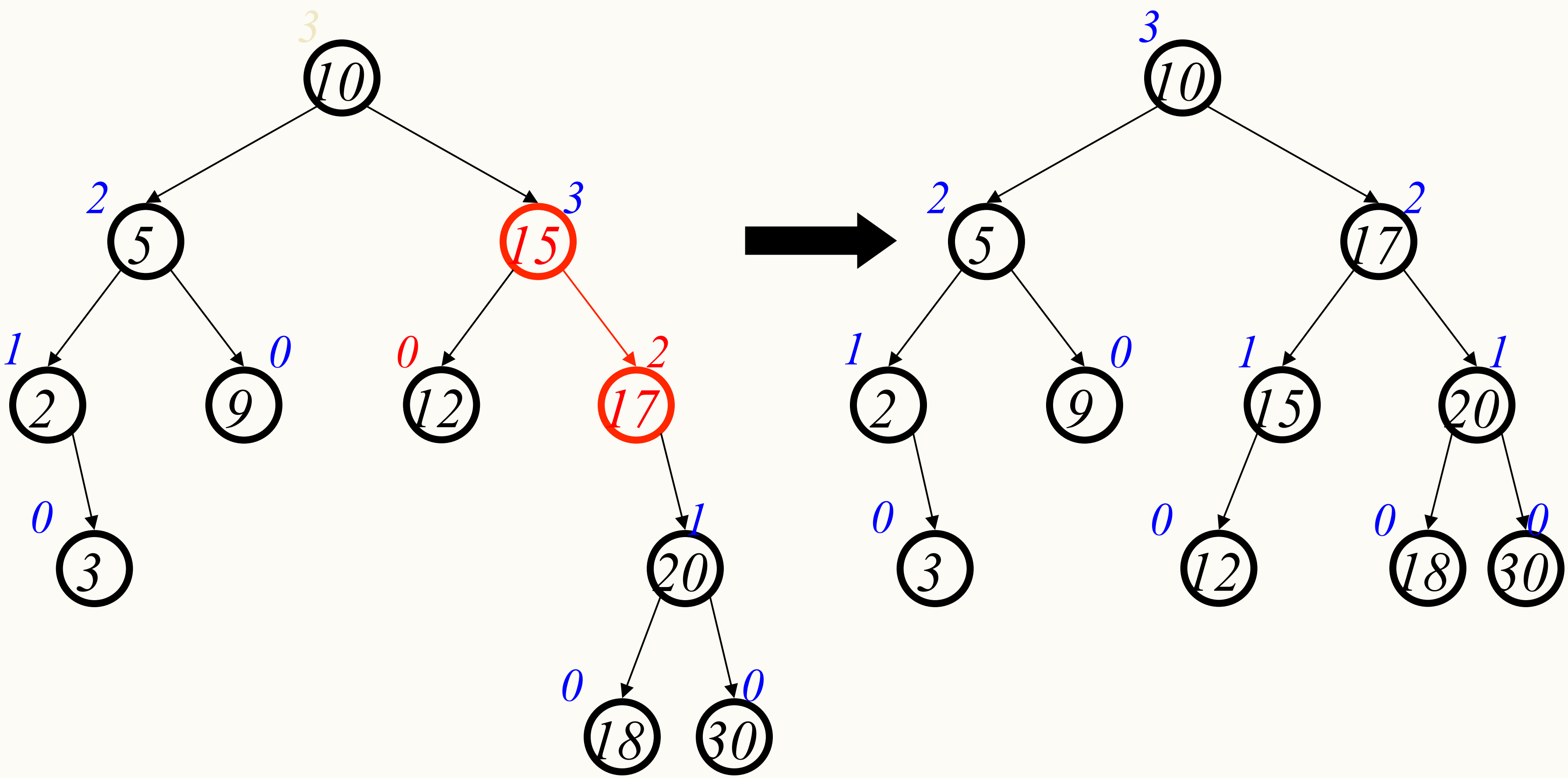
This doesn't work!!!

Double Rotation (Step #1)



Look familiar?

Double Rotation (Step #2)



AVL Algorithm Revisited

- Recursive

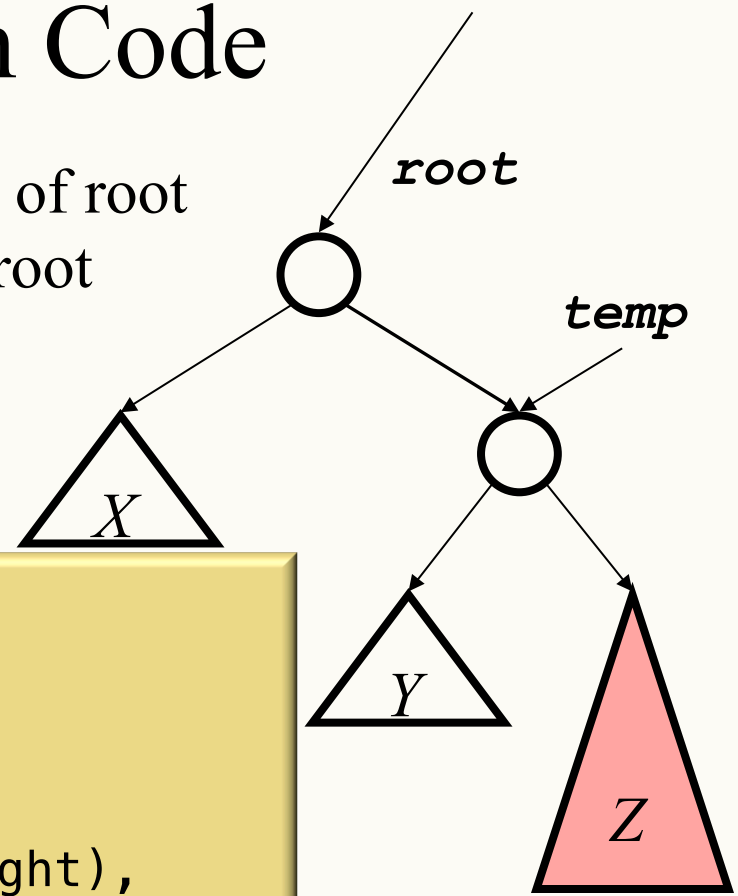
1. **Search downward for spot**
2. **Insert node**
3. **Unwind stack, correcting heights**
   a. **If imbalance #1, single rotate**
   b. **If imbalance #2, double rotate**

- Iterative

1. **Search downward for spot, stacking parent nodes**
2. **Insert node**
3. **Unwind stack, correcting heights**
   a. **If imbalance #1, single rotate and exit**
   b. **If imbalance #2, double rotate and exit**

Single Rotation Code

(1) Change left child of temp to right child of root
(2) Change arrow root→ temp to temp → root
(3) Change the root pointer



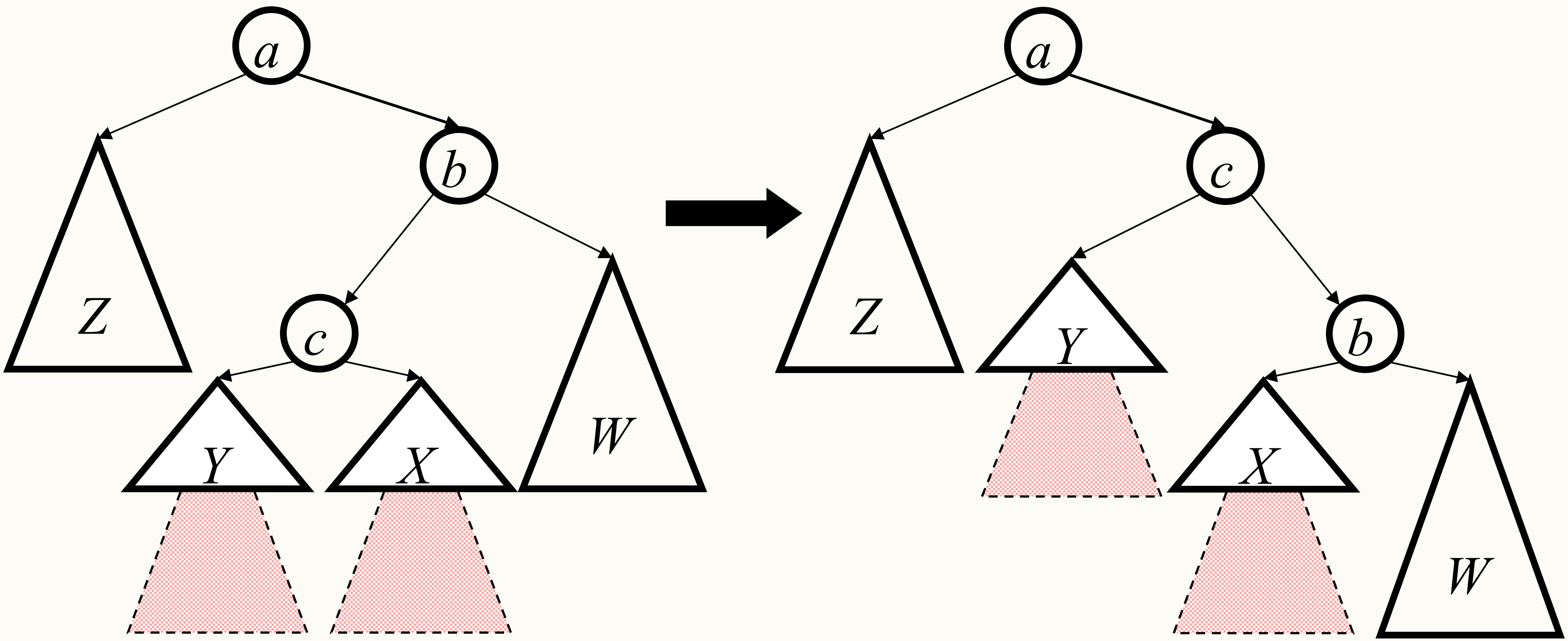

```
void RotateLeft(Node *& root) {
  Node * temp = root->right;
  root->right = temp->left; /* 1 */
  temp->left = root; /* 2 */
  root->height = max(height(root->right),
                     height(root->left)) + 1;
  temp->height = max(height(temp->right),
                     height(temp->left)) + 1;
  root = temp; /* 3 */
}
```

Height of Null tree is -1

Double Rotation Code

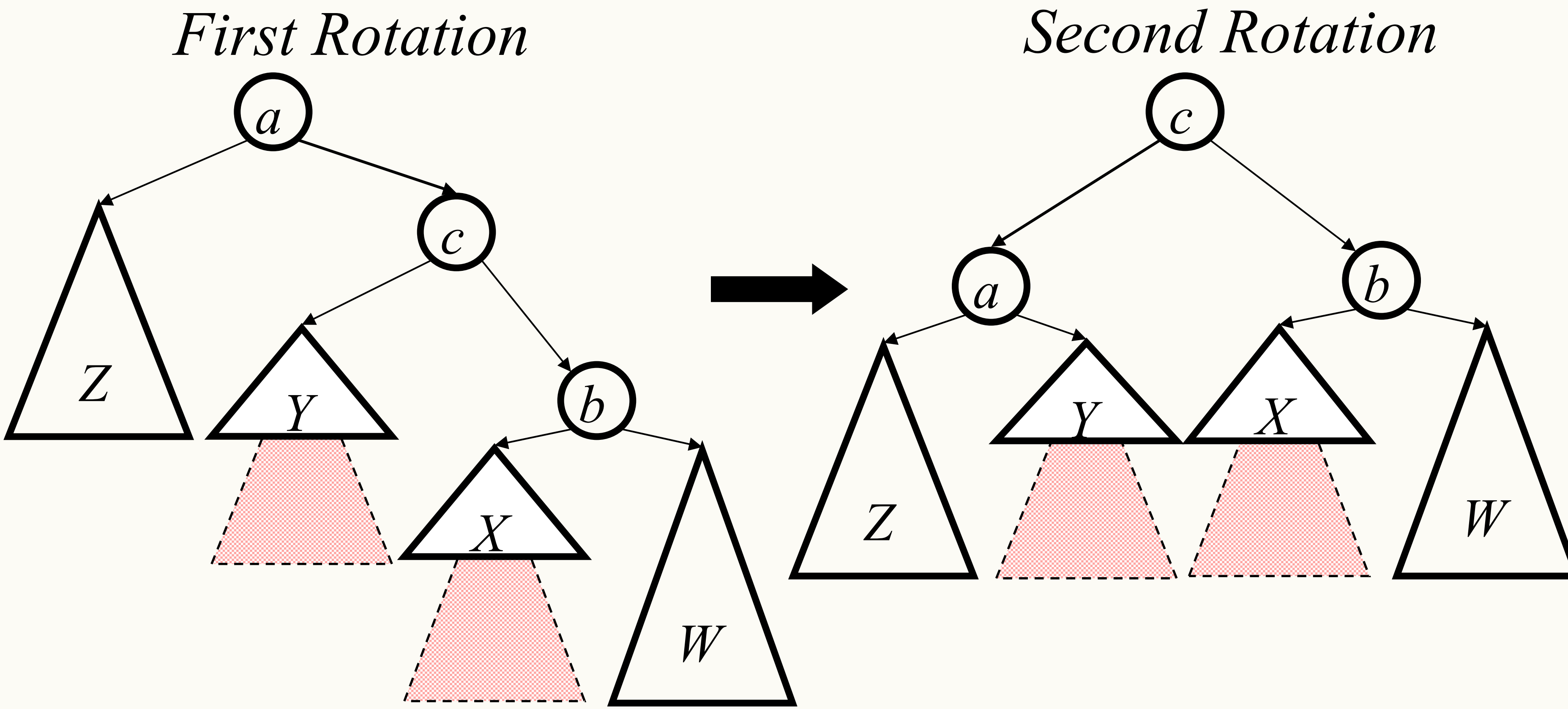
```
void DoubleRotateLeft(Node *& root) {
  RotateRight(root->right);
  RotateLeft(root);
}
```

First Rotation



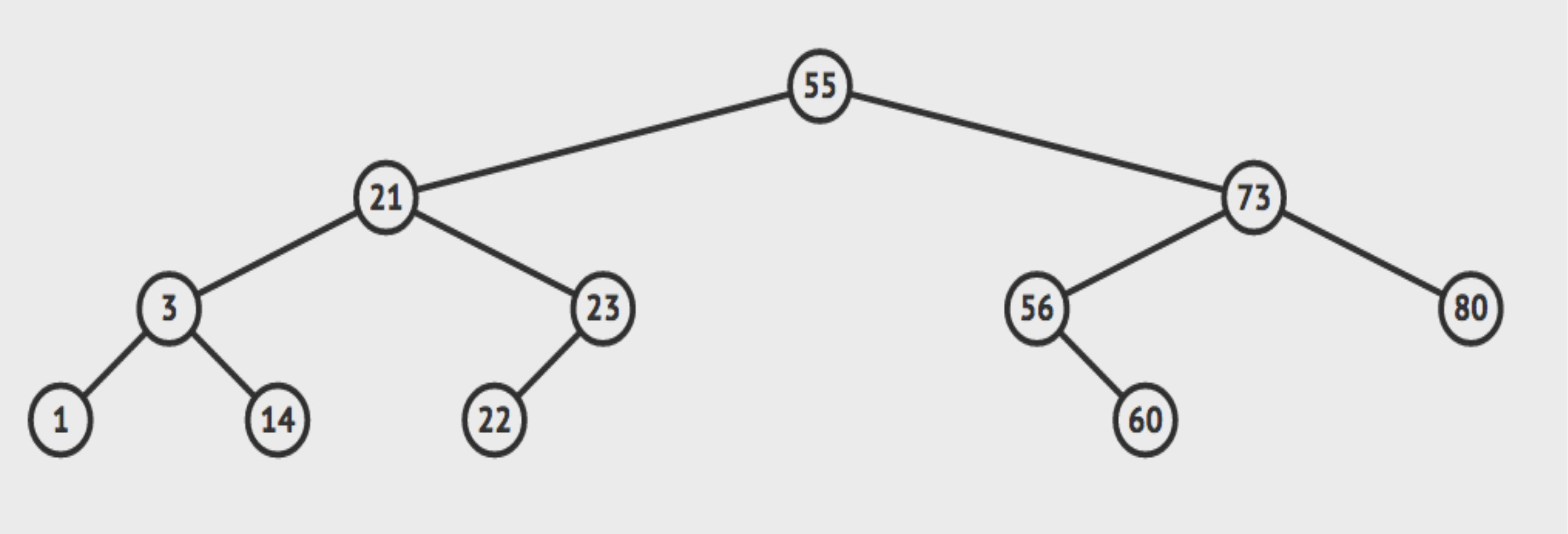
Double Rotation Completed

```
void DoubleRotateLeft(Node *& root) {
  RotateRight(root->right);
  RotateLeft(root);
}
```



Exercise

- Insert the following values into an AVL
  - 73, 80, 21, 22, 3, 14, 1, 55, 23, 56, 60



- Check all the steps http://visualgo.net/bst.html

What Does AVL Stand for?

- Automatically Virtually Leveled
- Architecture for inVisible Leveling (the "in" is inVisible)
- All Very Low
- Articulating Various Lines
- Amortizing? Very Lousy!
- Absolut Vodka Logarithms
- Amazingly Vexing Letters

Adelson-Velskii Landis

Learning goals revisited

- Compare and contrast balanced/unbalanced trees.
- Describe and apply rotation to a BST to achieve a balanced tree.
- Recognize balanced binary search trees (among other tree types you recognize, e.g., heaps, general binary trees, general BSTs).