CSE 326: Data Structures
AVL Trees

Richard Anderson, Steve Seitz

Winter 2014

Announcements

- HW 2 due now
- HW 3 out today

Balanced BST

Complexity of operations depend on **tree height**

For a BST with n nodes

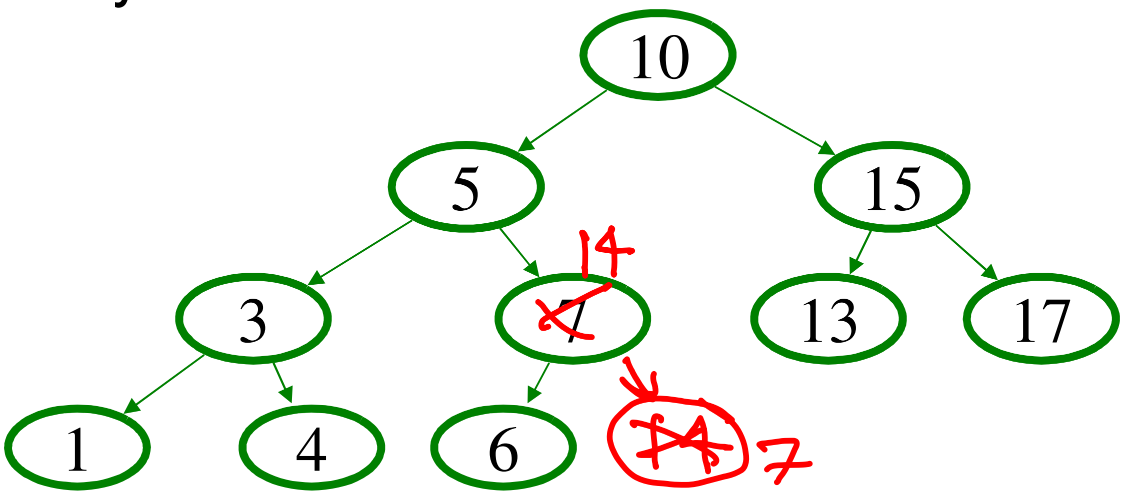
- Want height to be ~ log n
- “Balanced”

But balancing cost must be low

How about complete trees?

This worked for heaps

- balance maintained via percolate up/down
- Let's try with BST



(add 14 in rightmost leaf, percolate up)

Balancing Trees

- Many algorithms exist for keeping trees balanced
  - Adelson-Velskii and Landis (AVL) trees
  - Splay trees and other *self-adjusting* trees
  - B-trees and other multiway search trees (for *very* large trees)

- Today we will talk about **AVL trees**…

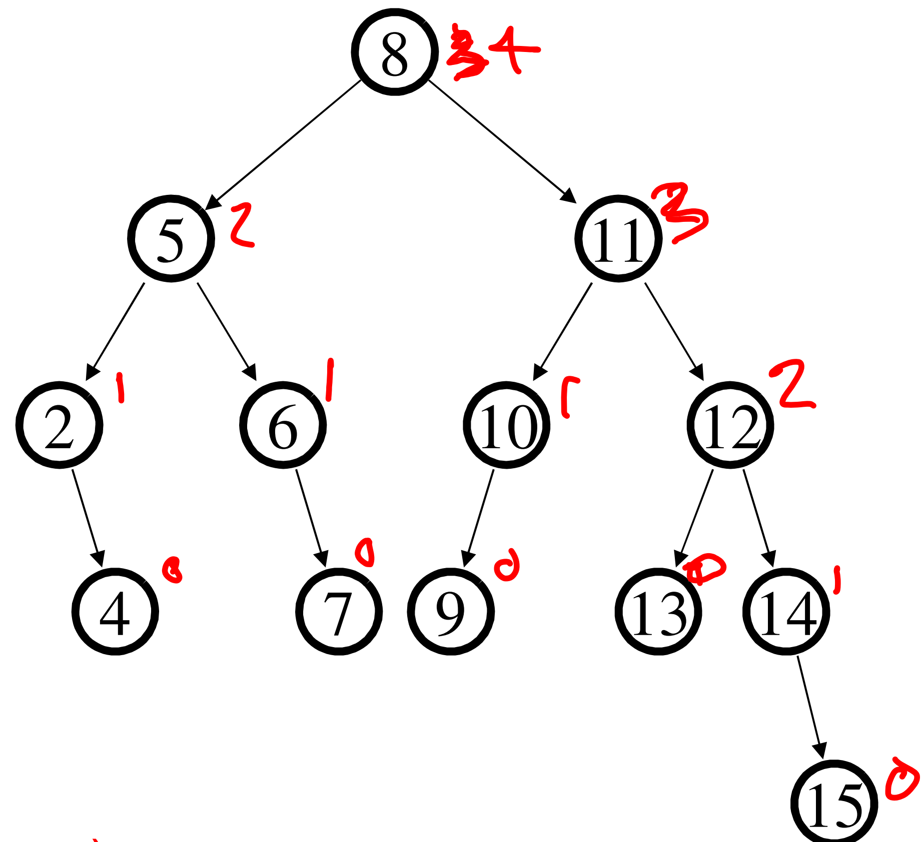
The AVL Tree Data Structure

<u>Ordering property</u>

– Same as for BST

<u>Structural properties</u>

1. Binary tree property (0,1, or 2 children)
2. Heights of left and right subtrees of *every node* **differ by at most 1**



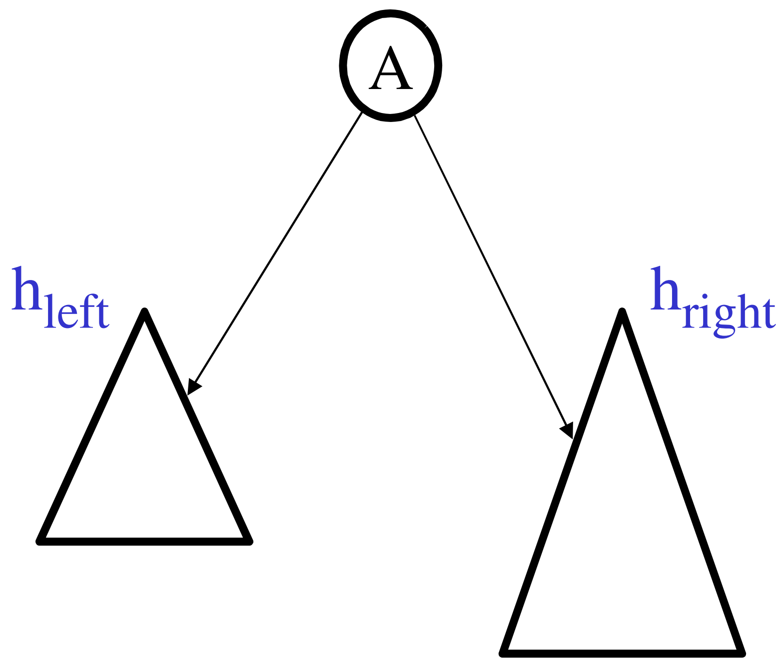
Result: worst case height: O(log n)

Recursive Height Calculation

Recall: height is max number of edges from root to a leaf

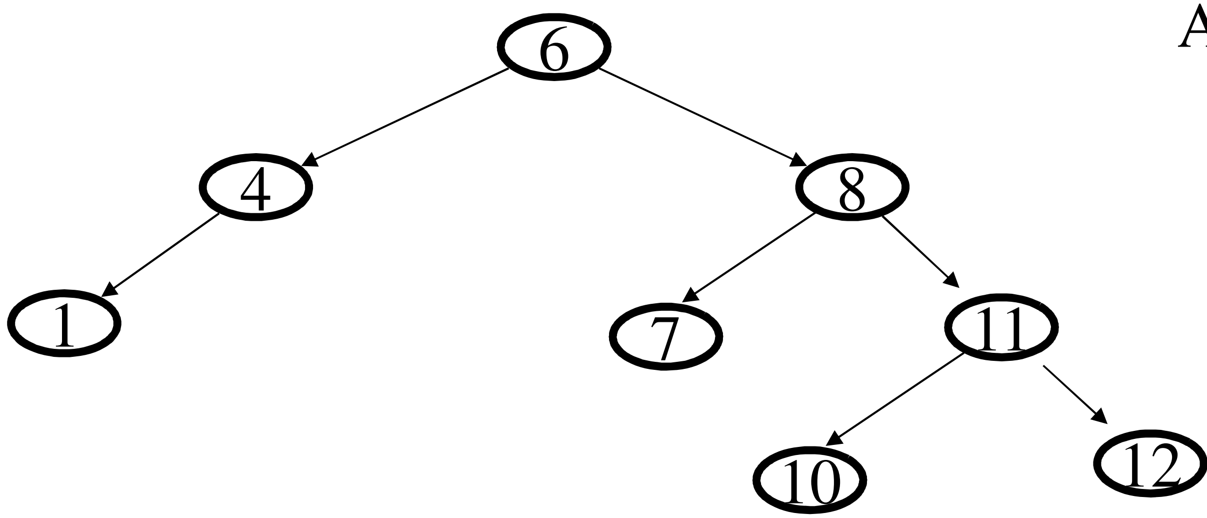
What is the height at A?

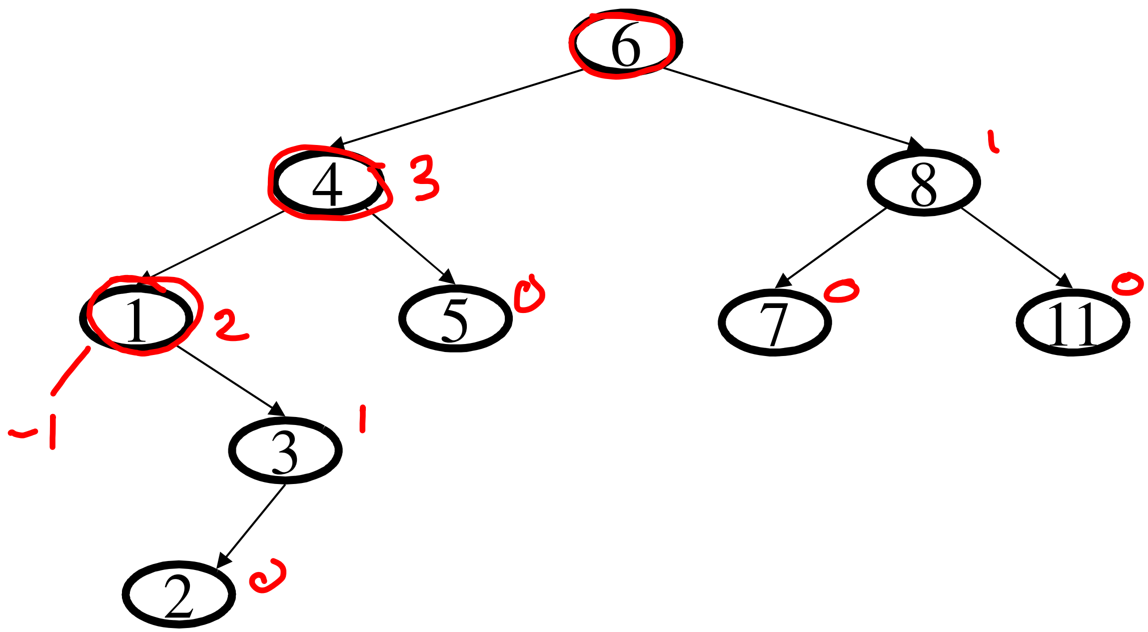
$\max(h_{left}, h_{right}) + 1$



Define: height(null) = -1

AVL trees or not?



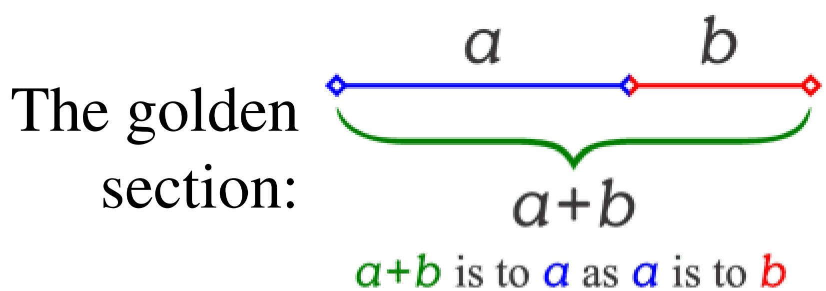


Goal

h $\in$ O(log n)

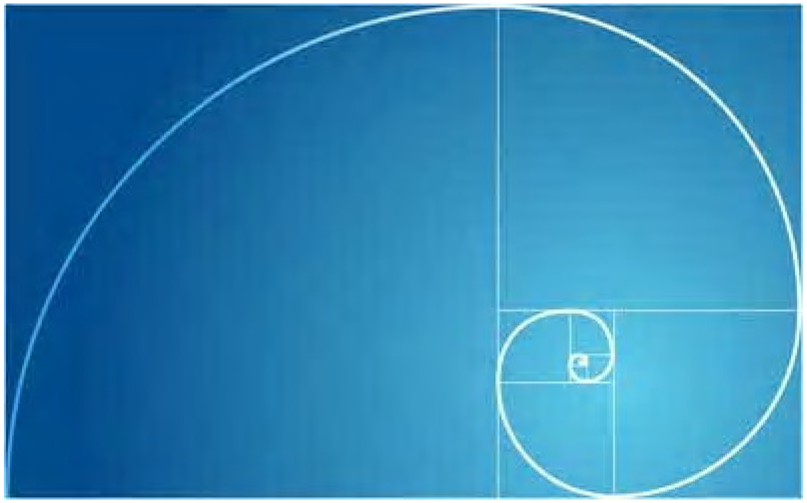
- we will do this by showing: $n + 1 > \phi^h$ $\Rightarrow \log_\phi(n+1) > \log_\phi \phi^h = h$
- What's ϕ?

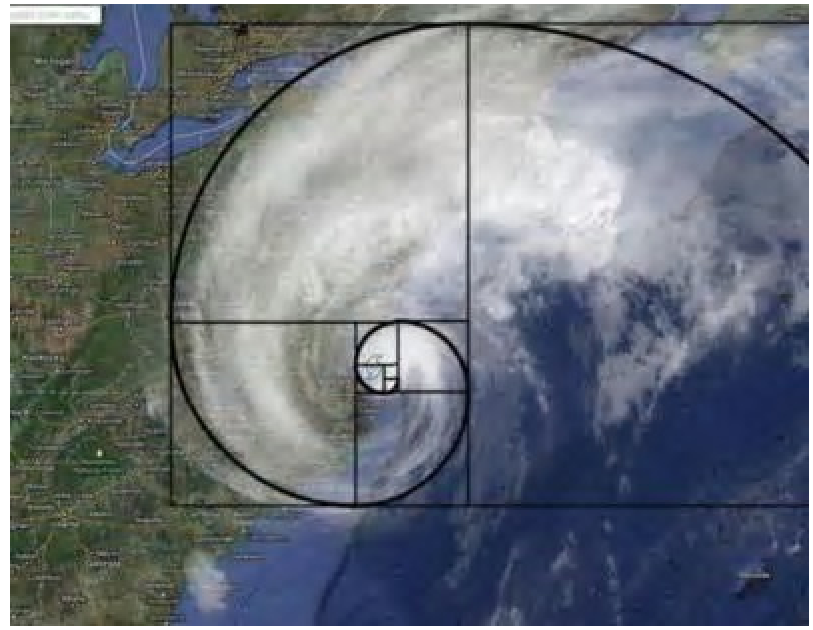
  ϕ is the golden ratio, $(1+\sqrt{5})/2$

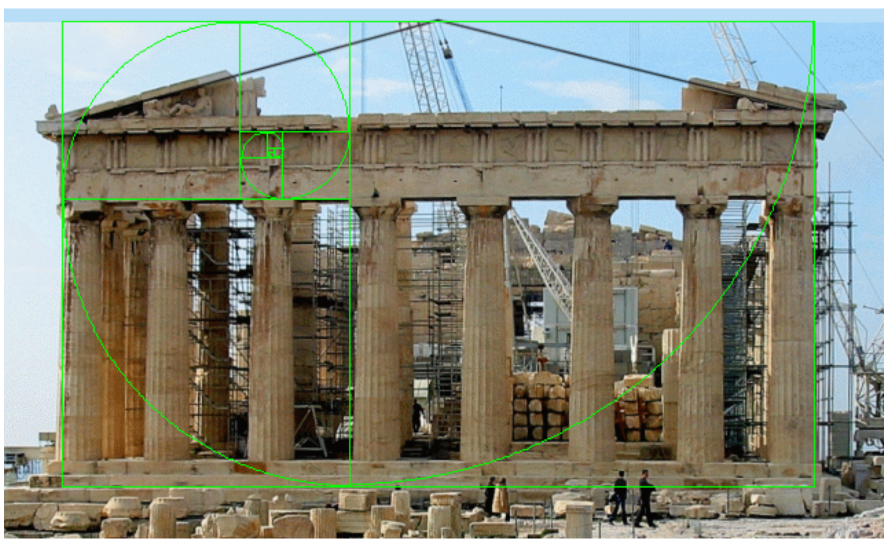


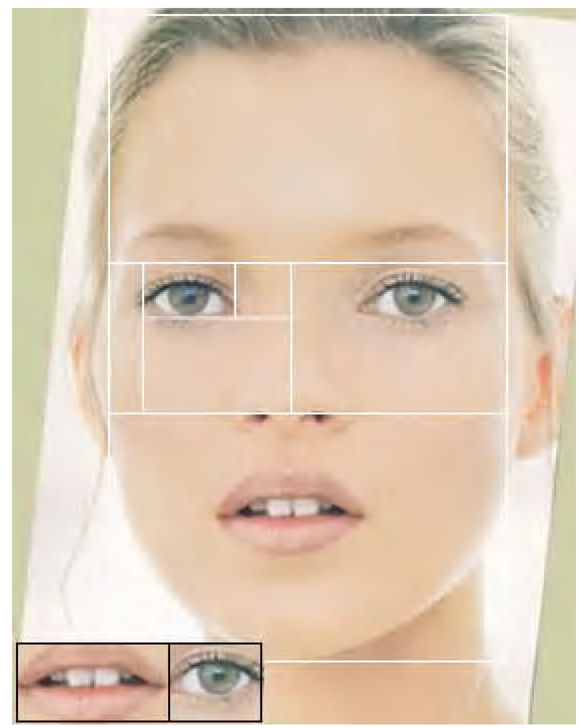
$$\frac{a+b}{a} = \frac{a}{b} = \phi$$

– Since the Renaissance, many artists and architects have proportioned their work (e.g., length:height) to approximate the golden ratio ϕ



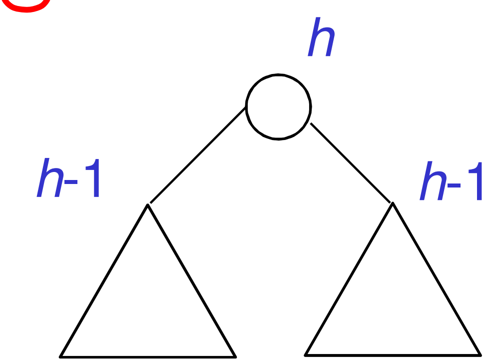




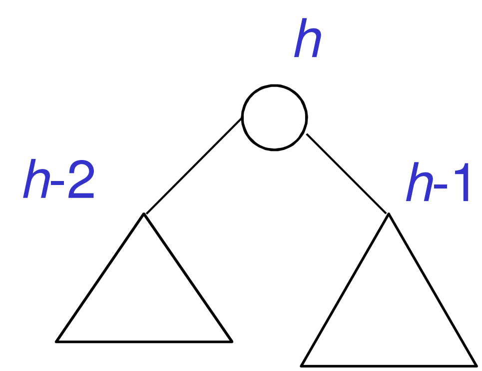


Minimum Size of an AVL Tree

- n $\geq$ m(h) = minimum # of nodes in an AVL tree of height h.
- **Base cases:**
  - $m(0) = 1$ $m(1) = 2$ $m(2) = 4$
- **Inductive case:**
  - $m(h) = m(h-2) + m(h-1) + 1$



- Can prove:
  - $m(h) > \phi^h - 1$



Proof that $m(h) > \phi^h - 1$

- Base cases h=0,1:

$$m(0) = 1 > \phi^0 - 1 = 0 \qquad\qquad m(1) = 2 > \phi^1 - 1 \approx 0.62$$

- Assume true for h-2 and h-1:

$$m(h\text{-}2) > \phi^{h-2} - 1 \qquad\qquad m(h\text{-}1) > \phi^{h-1} - 1$$

- Induction step:

$$m(h) = m(h\text{-}1) + m(h\text{-}2) + 1 \;>\; (\phi^{h-1} - 1) + (\phi^{h-2} - 1) + 1$$

$$\begin{aligned}(\phi^{h-1} - 1) + (\phi^{h-2} - 1) + 1 &= \phi^{h-2}(\phi + 1) - 1 \\ &= \phi^{h-2}(\phi^2) - 1 \\ &= \phi^h - 1\end{aligned}$$

$$\rightarrow \quad m(h) \cancel{>} \phi^h - 1$$

$>$

Maximum Height of an AVL Tree

Suppose we have n nodes in an AVL tree of height h.

We can now say:

$$n \geq m(h) > \phi^h - 1$$

What does this say about n?

$$n > \phi^{h} - 1$$

What does this say about the complexity of h?

$h \in \Theta(\log n)$

$\phi^h < n+1 \Rightarrow \log_\phi \phi^h < \log_\phi (n+1) \Rightarrow h \in O(\log n)$

$\log_\phi \phi^h = h$

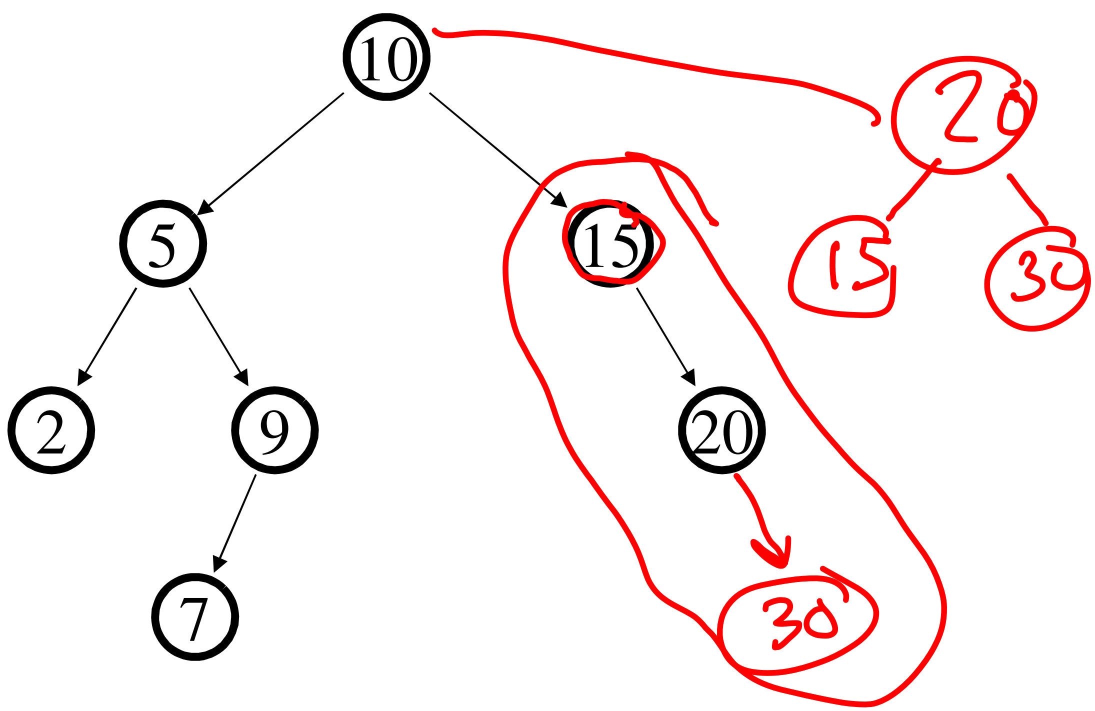
$n \leq 2^{h+1} - 1$

$\Rightarrow 2^{h+1} \geq n+1 \Rightarrow \log_2 2^{h+1} \geq \log_2 (n+1)$

$\log_2 2^{h+1} = h+1$

$h \in \Omega(\log n)$

Testing the Balance Property

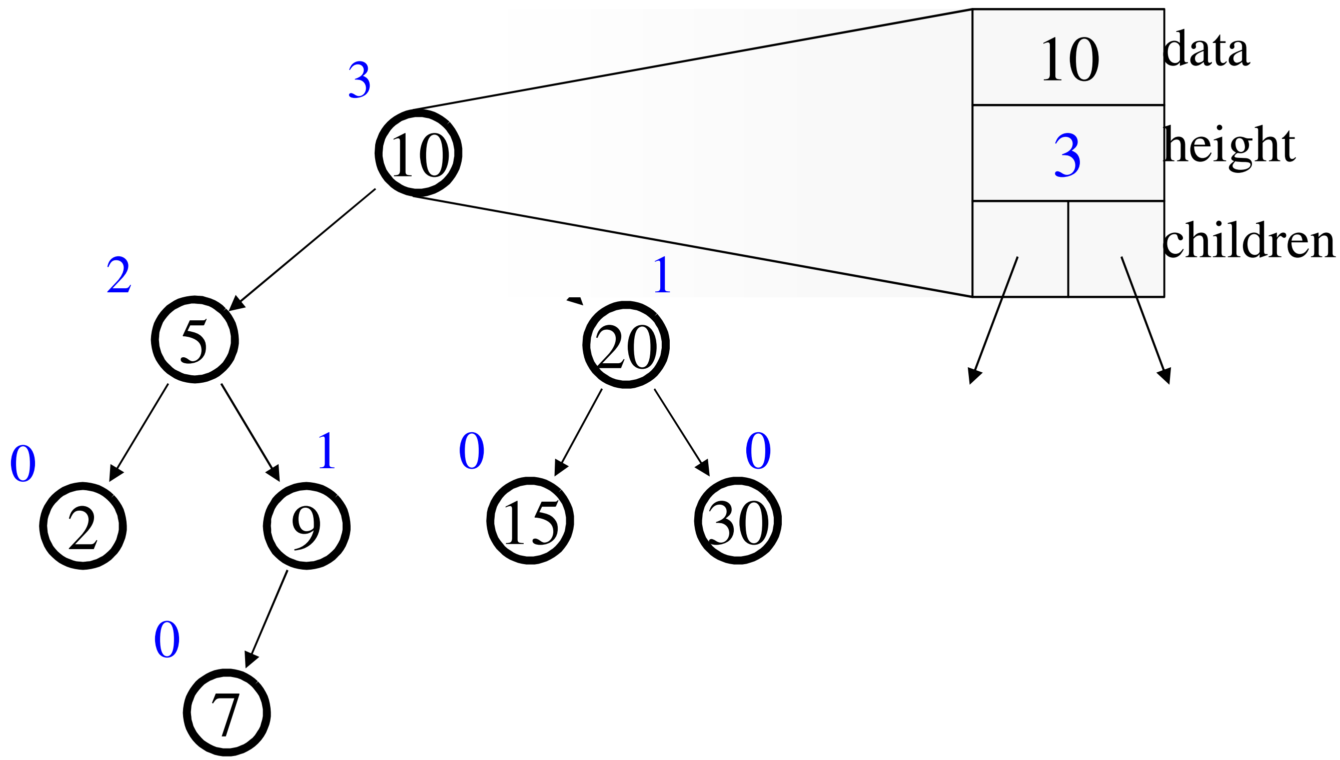


Is this AVL tree balanced?
How about after insert(30)?

We need to be able to:

1. Track Balance
2. Detect Imbalance
3. Restore Balance

An AVL Tree

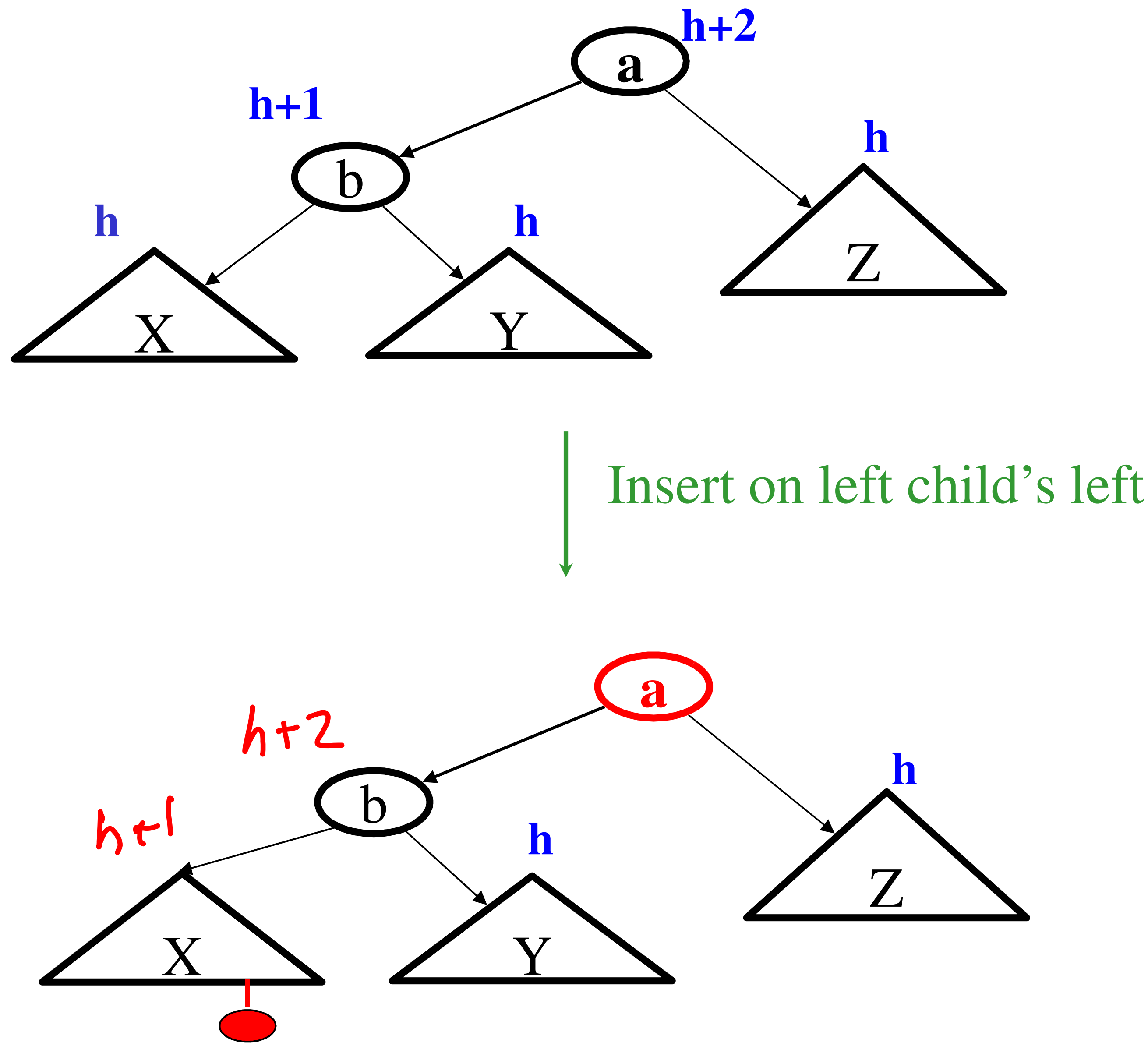


AVL trees: find, insert

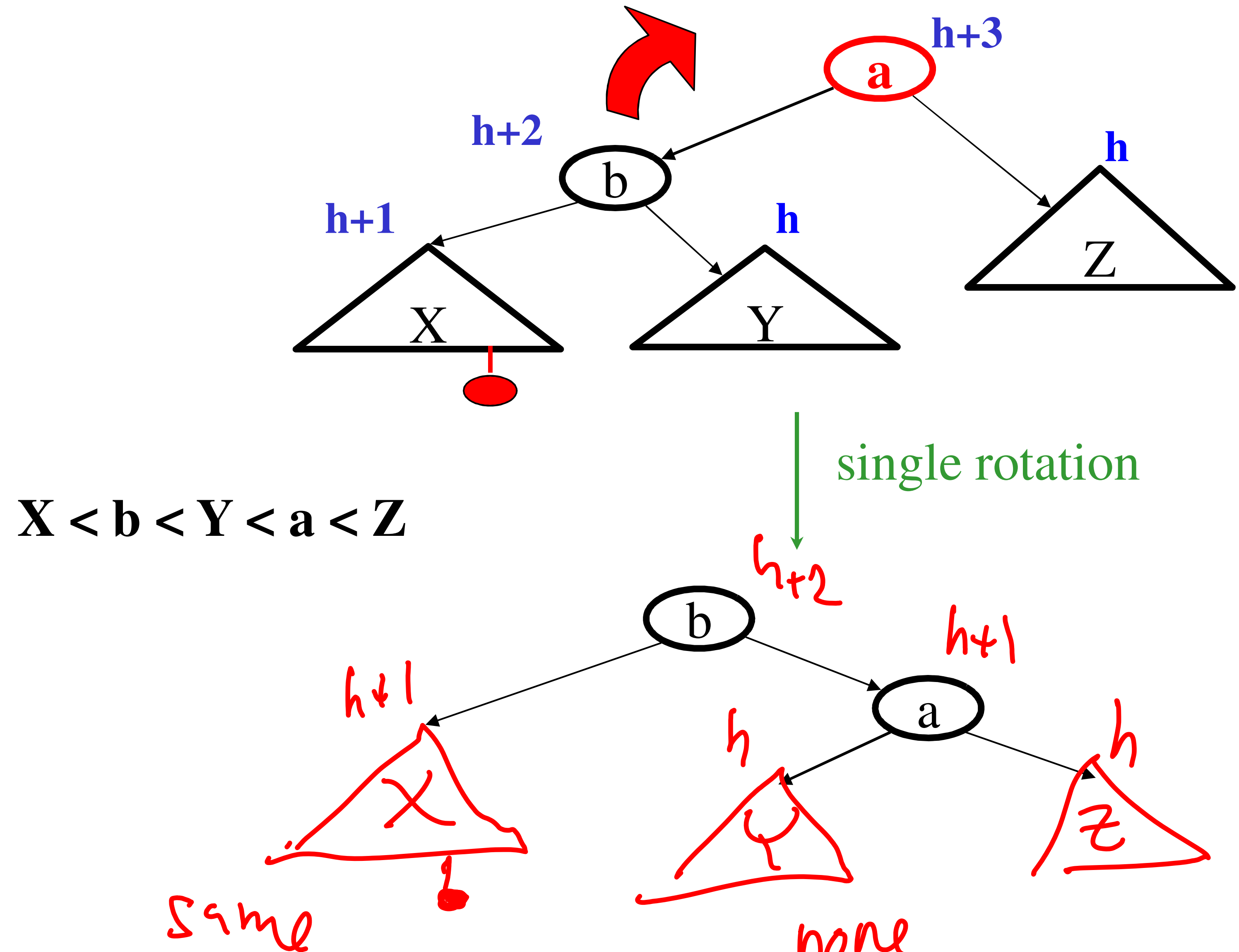
- **AVL find**:
  – same as BST find.
- **AVL insert**:
  – same as BST insert, *except* may need to “fix” the AVL tree after inserting new value.

We will consider the 4 fundamental insertion cases…

Case #1: left-left insertion (zig)

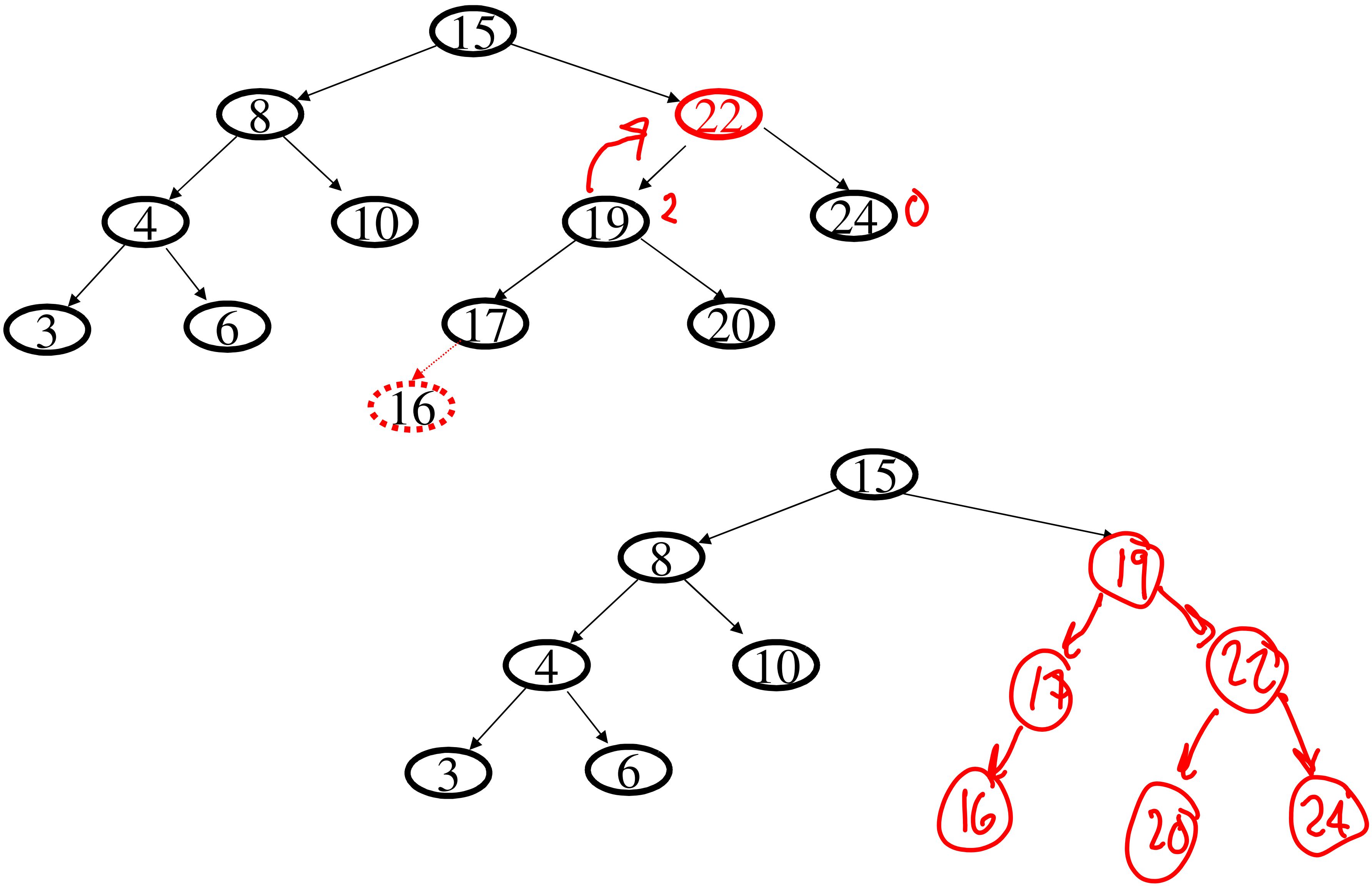


Case #1: repair with single rotation

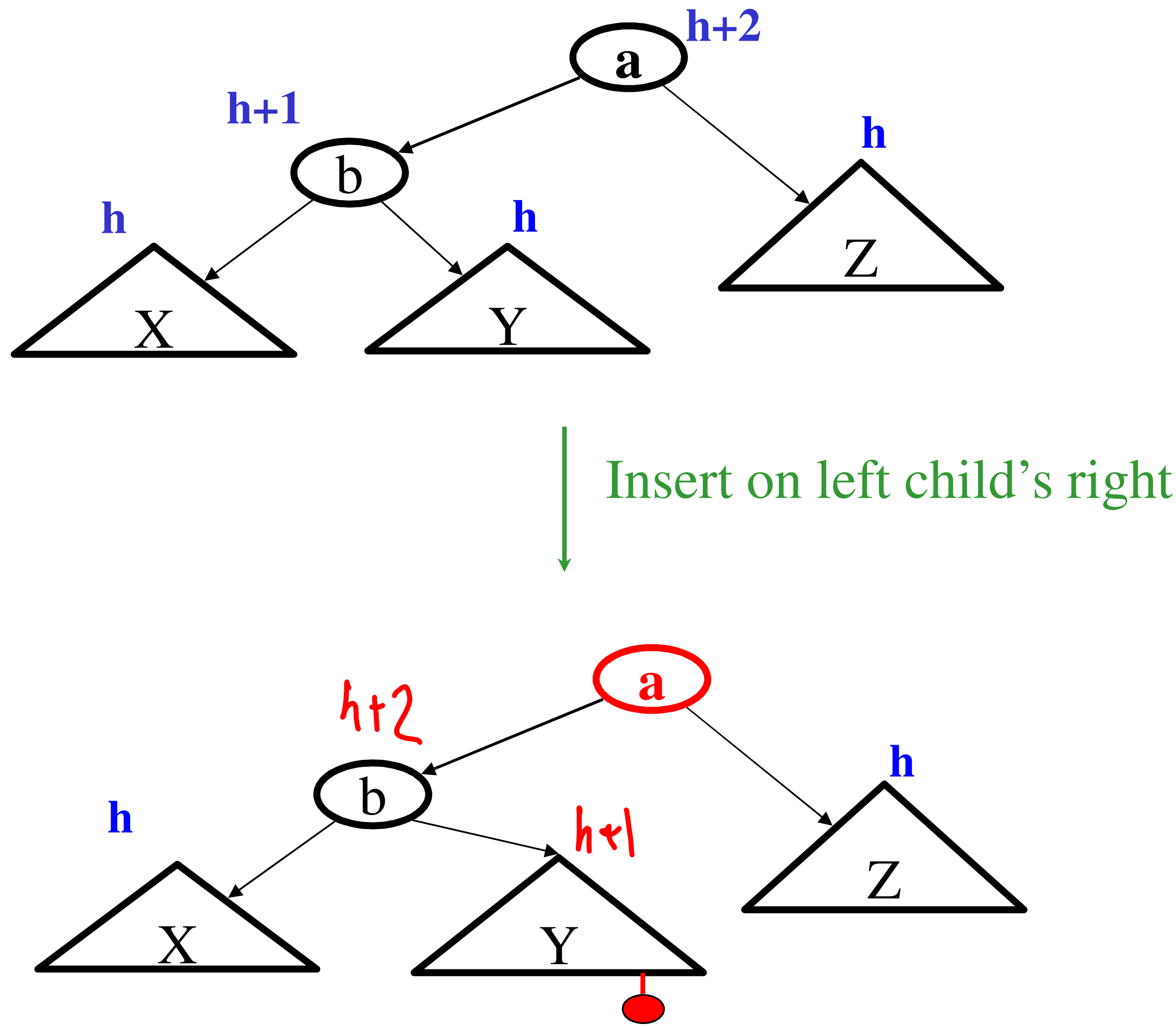


Height of tree before/after? Effect on Ancestors? Cost? O(1)

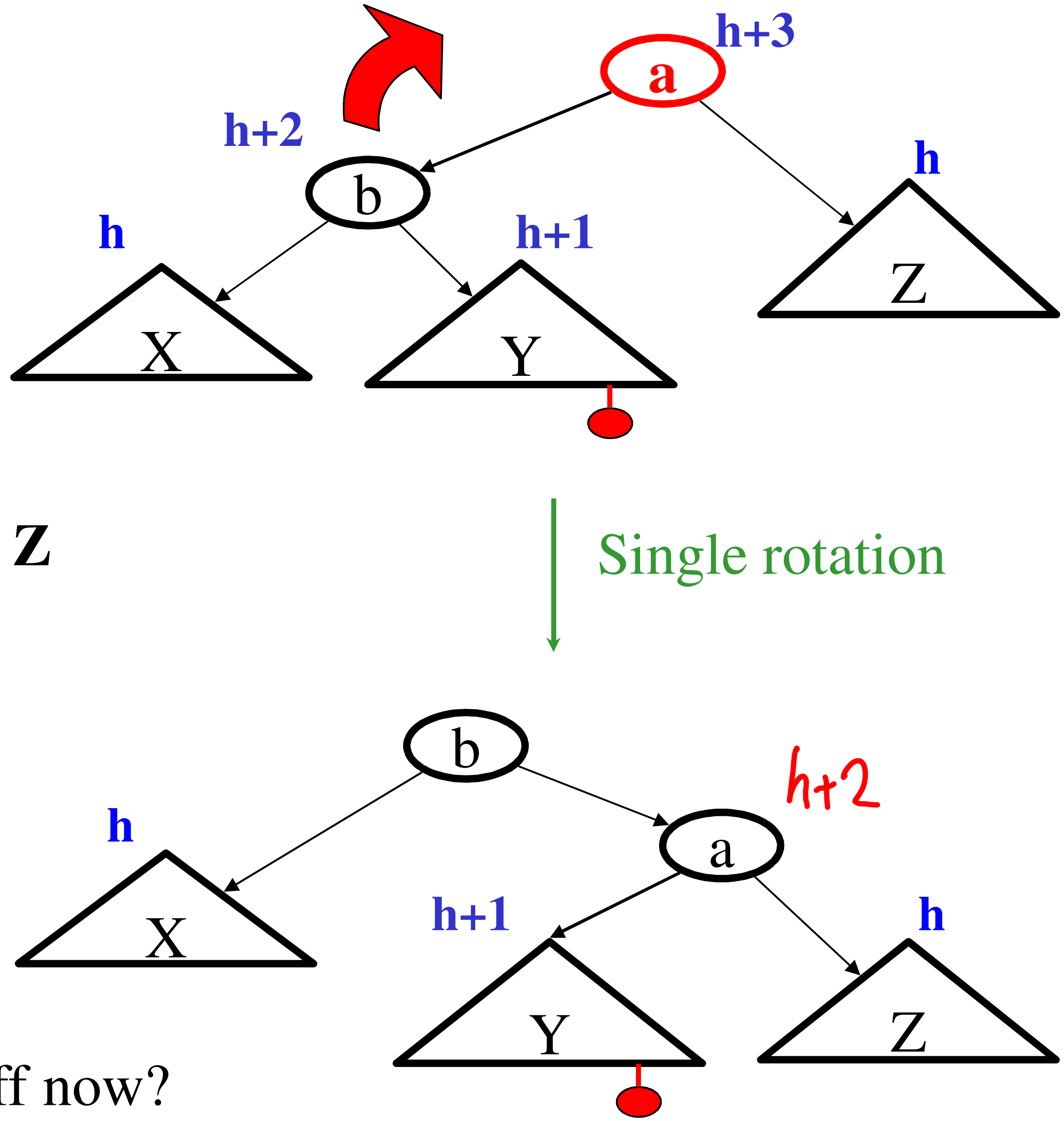
Single rotation example



Case #2: left-right insertion



Case #2: repair with single rotation?

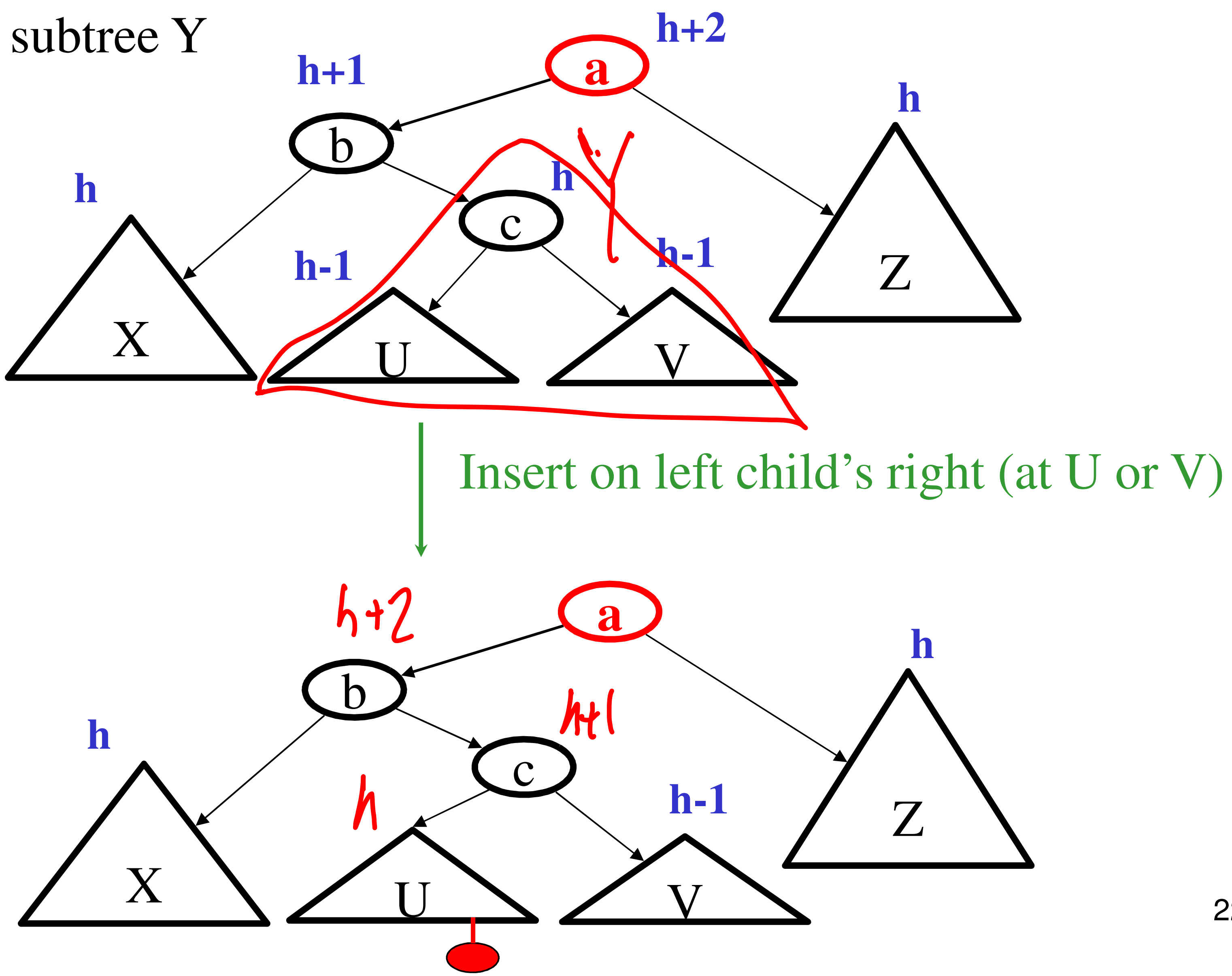


X < b < Y < a < Z

Are we better off now?

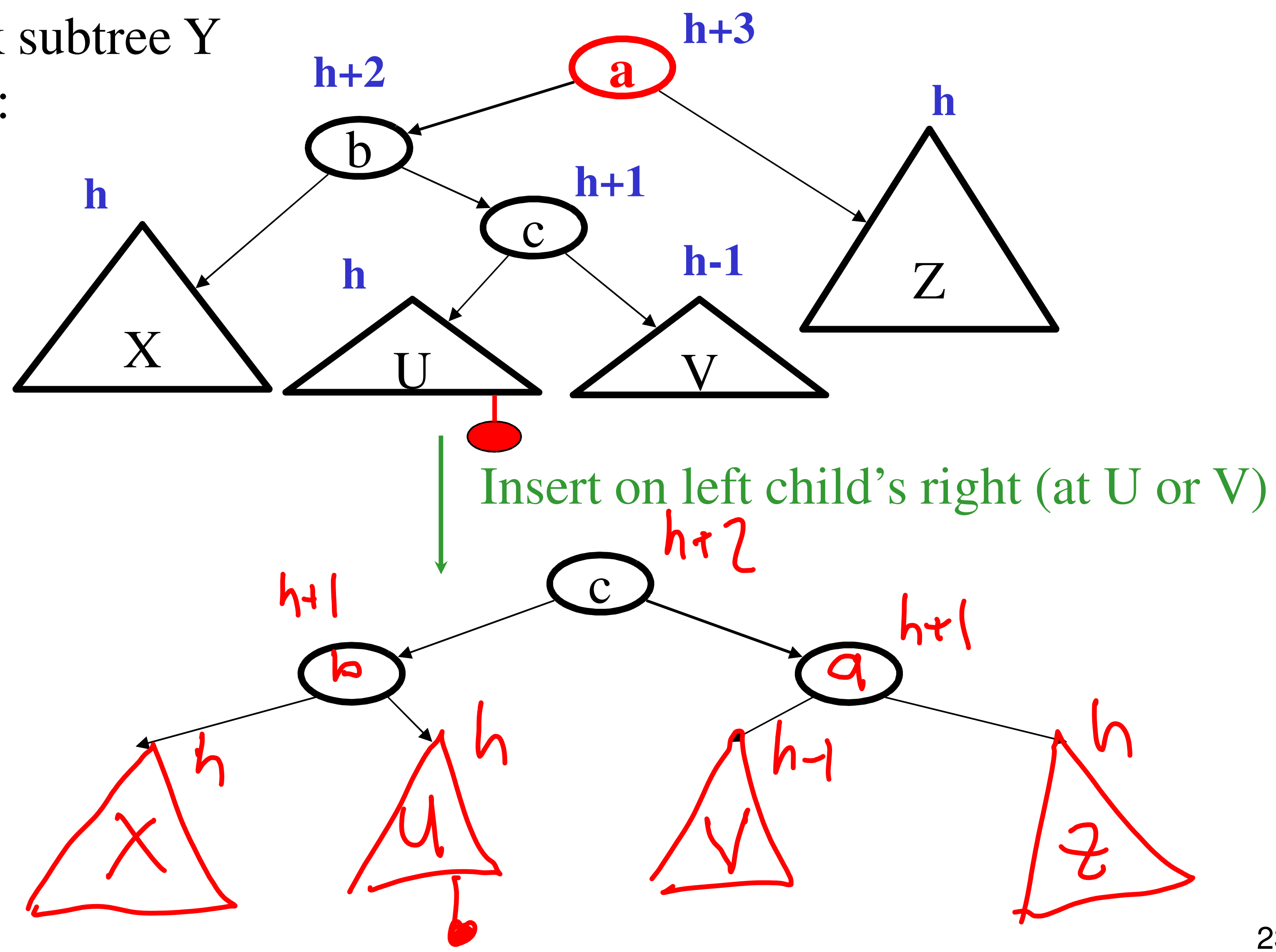
Case #2: trying again

Let's break subtree Y into pieces:

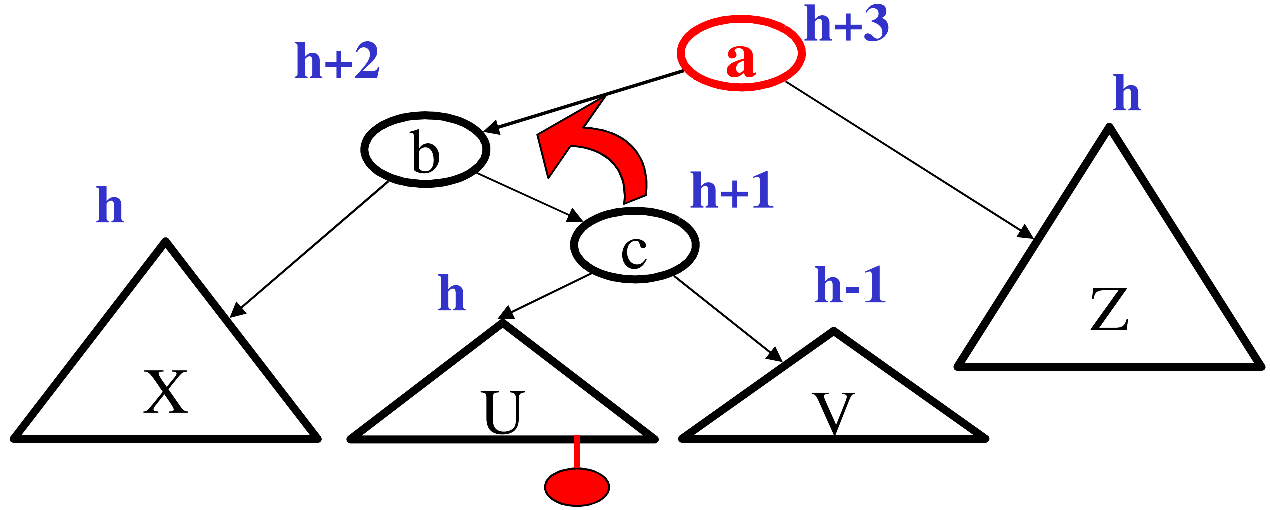


Case #2: trying again

Let's break subtree Y into pieces:

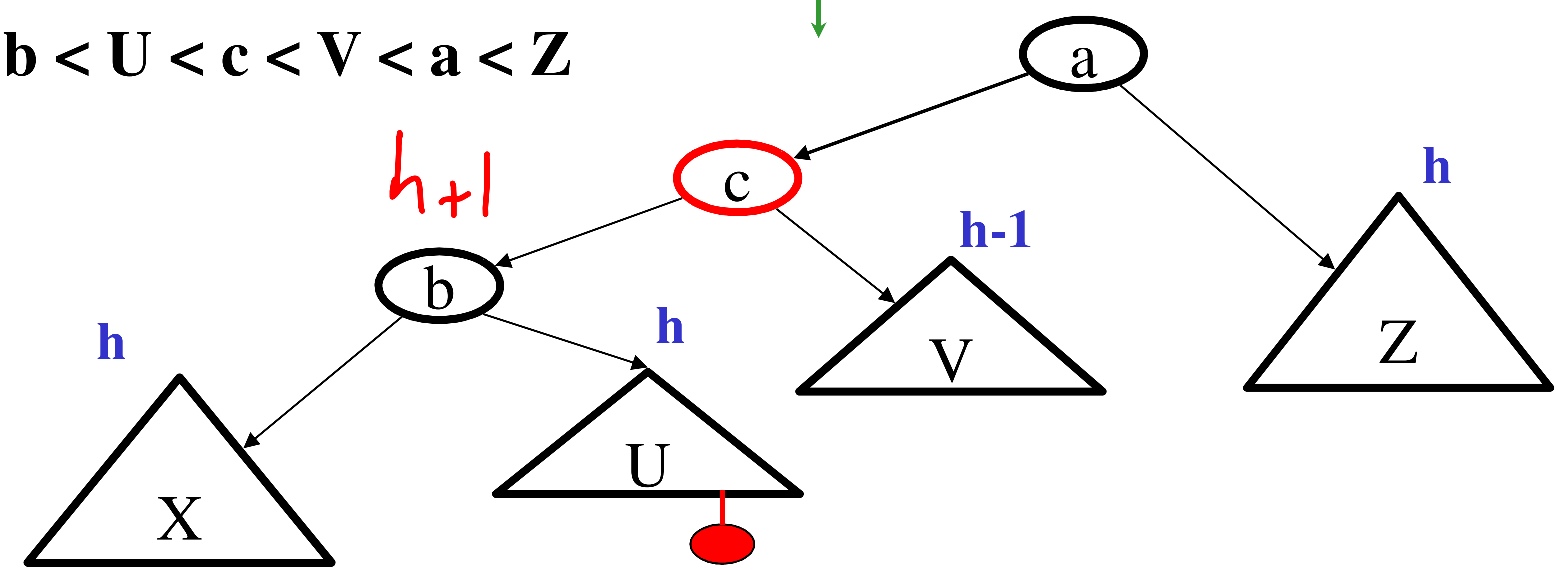


Can also do this in two rotations

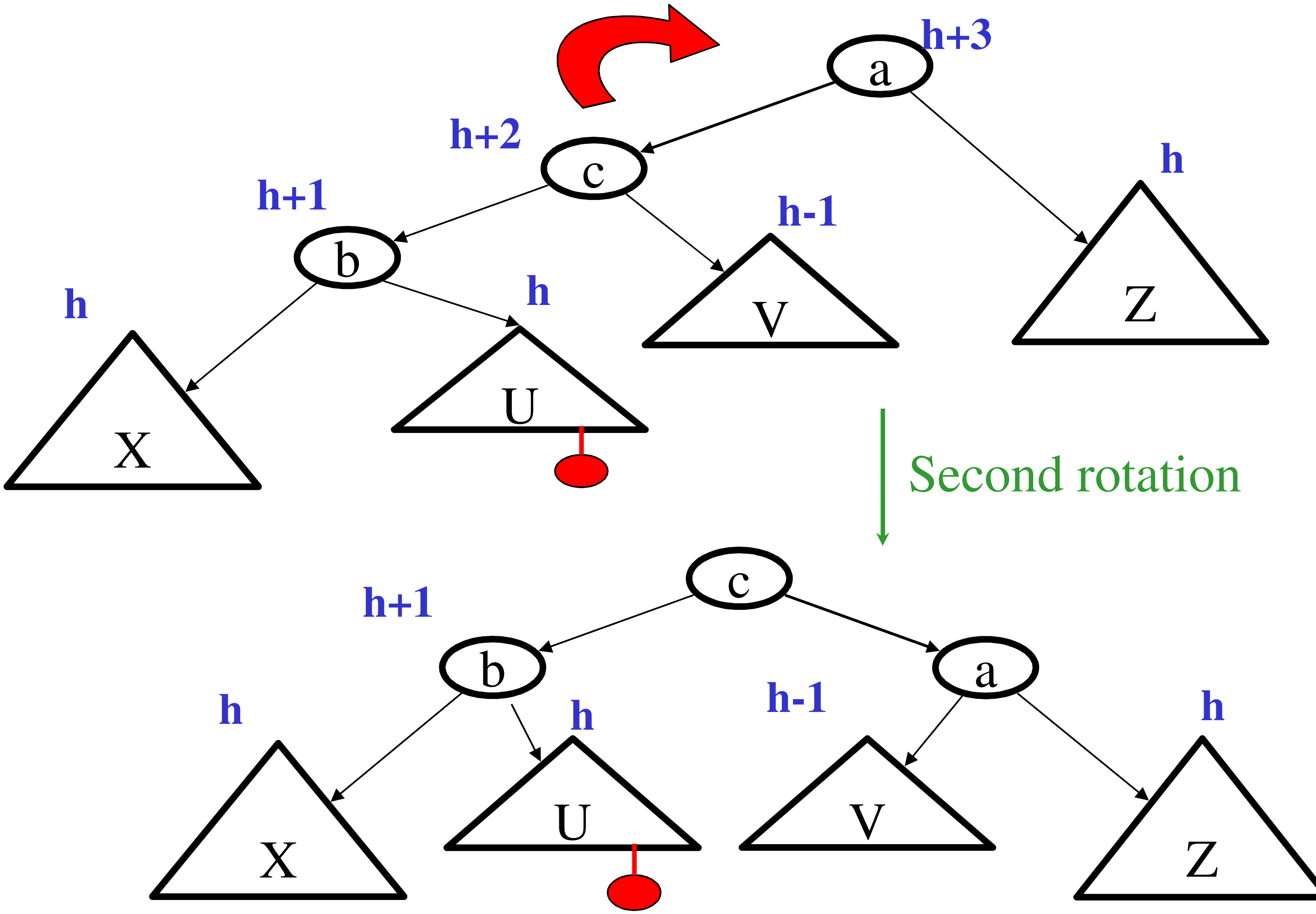


First rotation

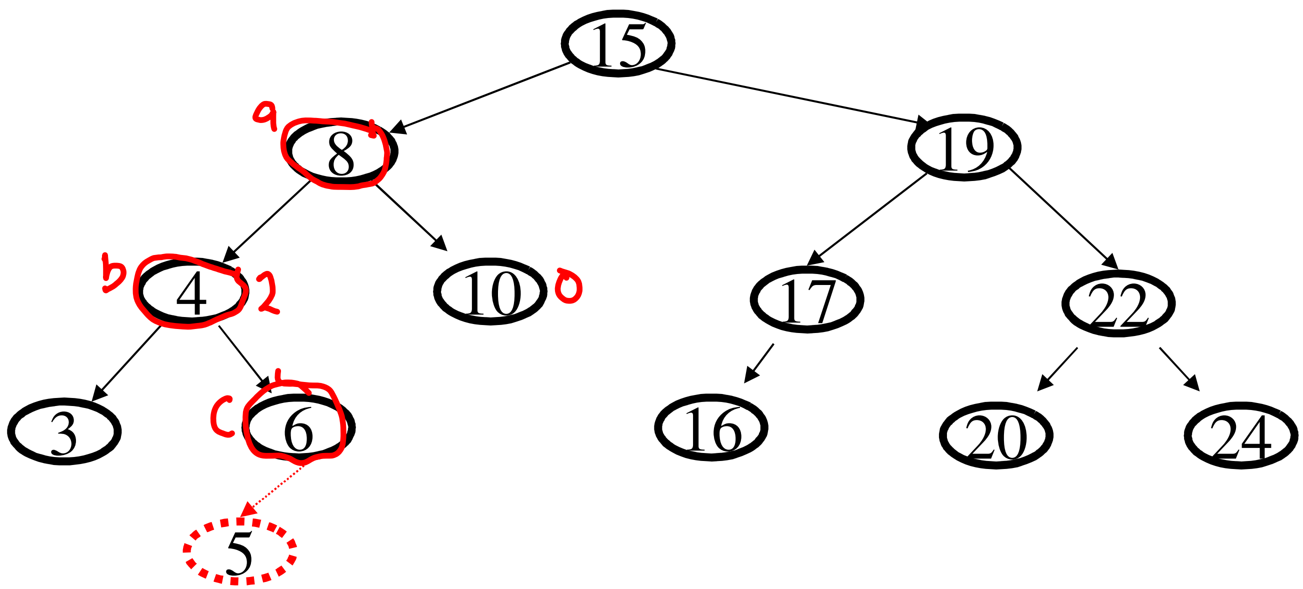
$X < b < U < c < V < a < Z$

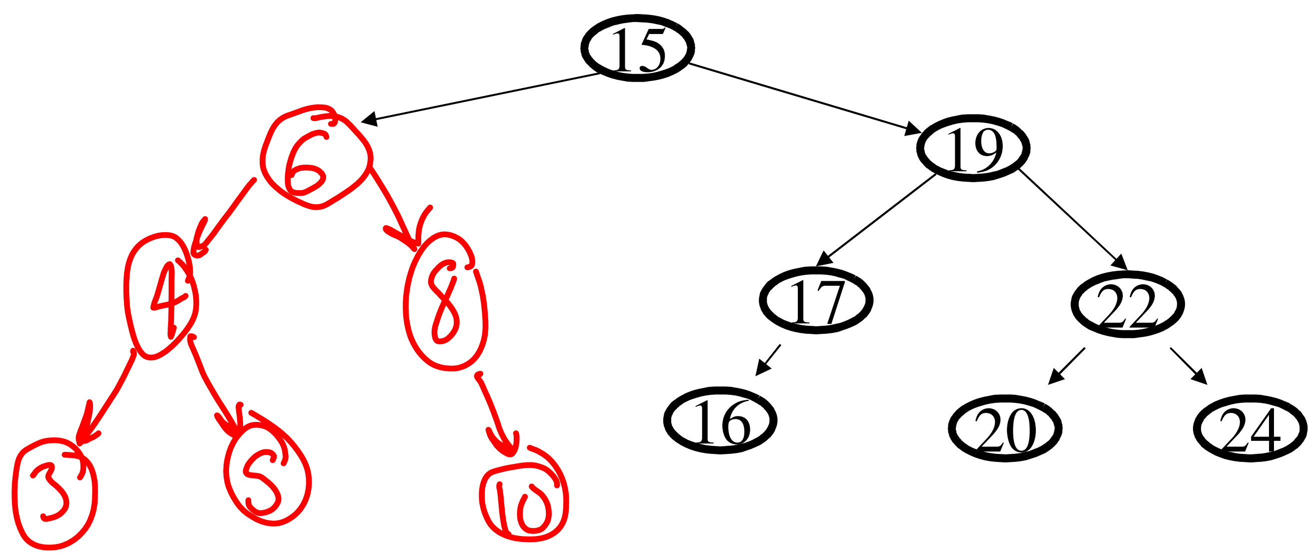


second rotation

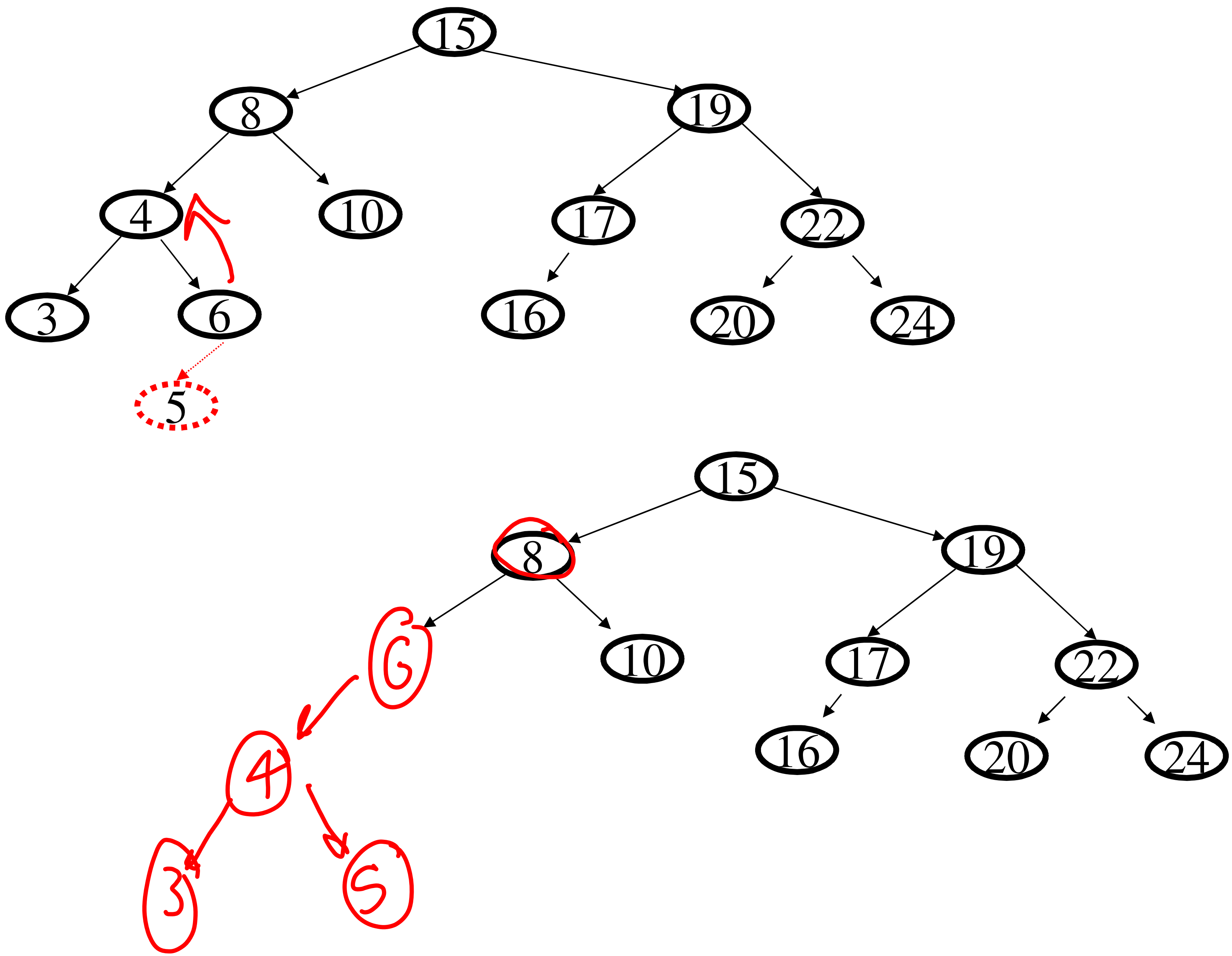


Double rotation example

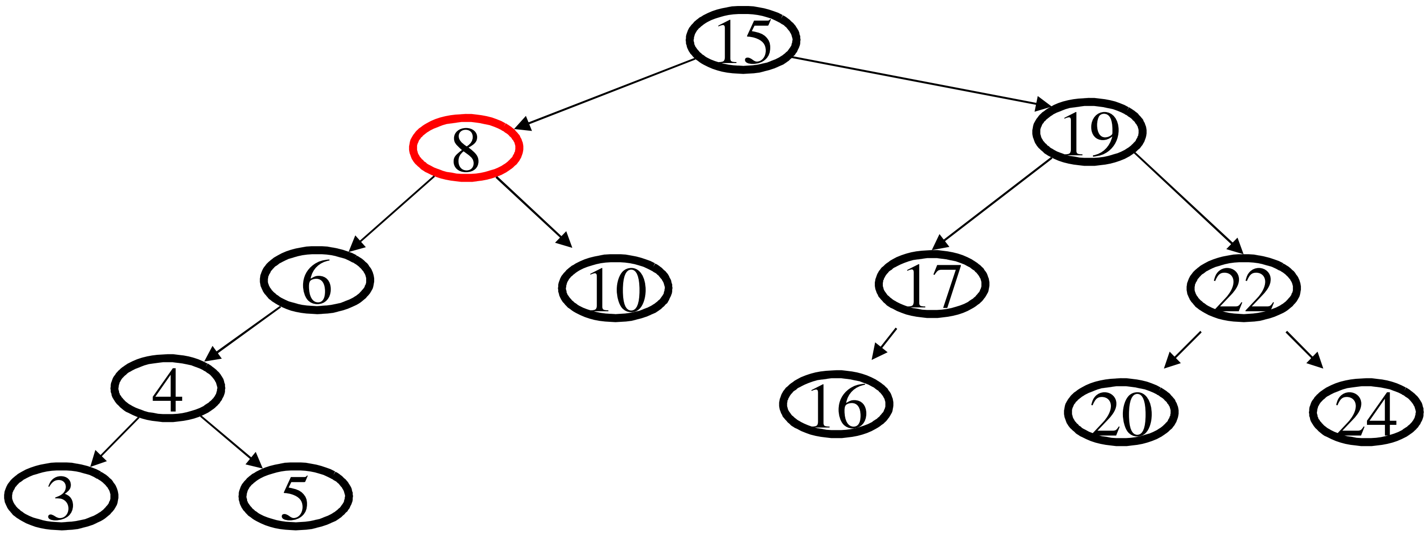


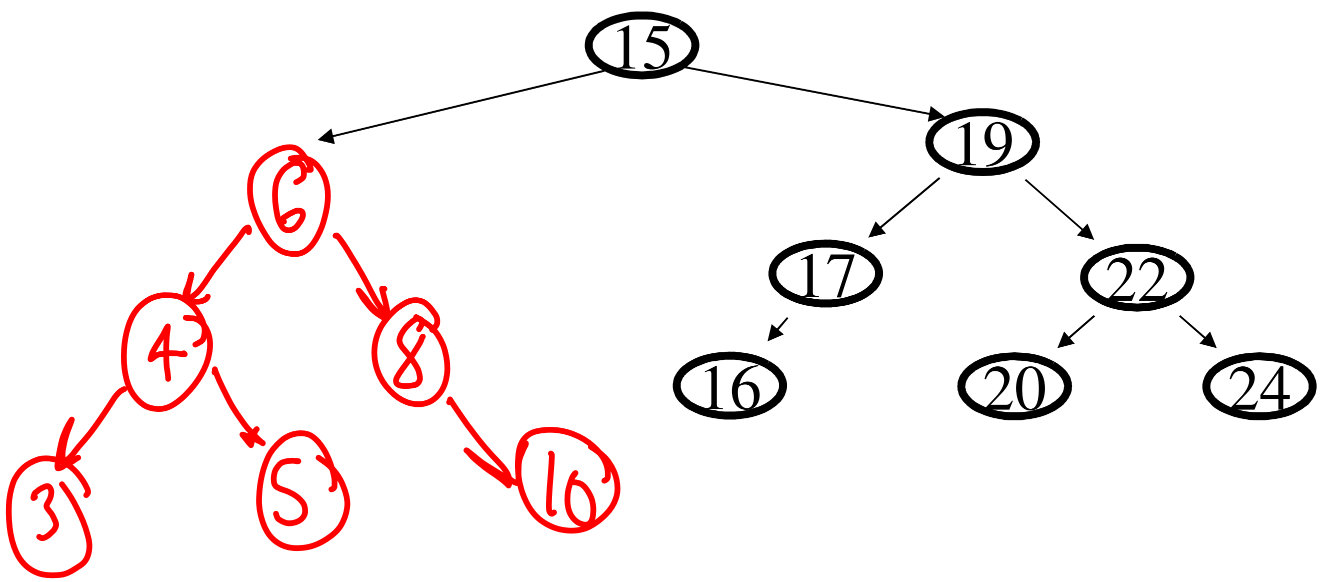


Double rotation, step 1

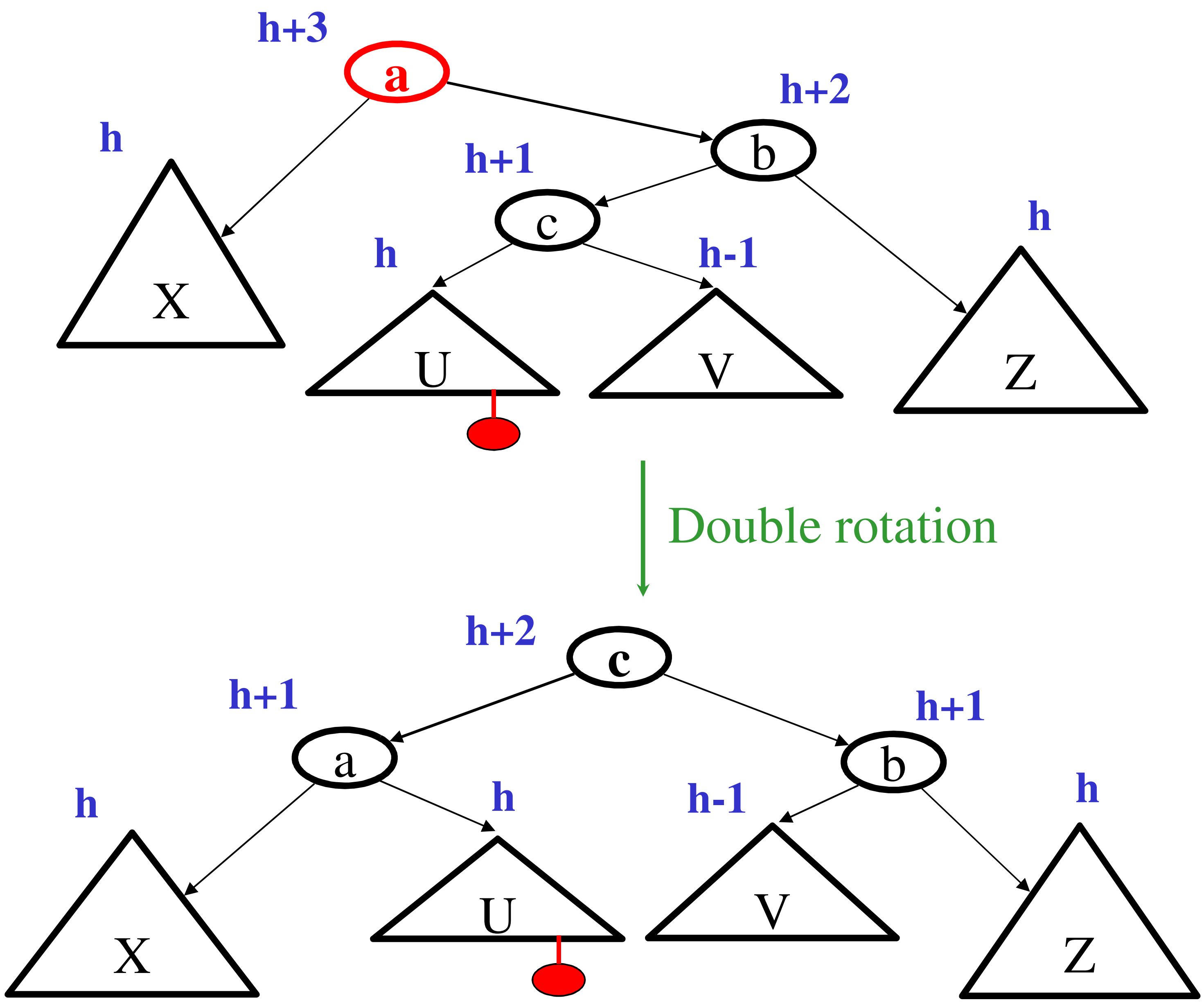


Double rotation, step 2

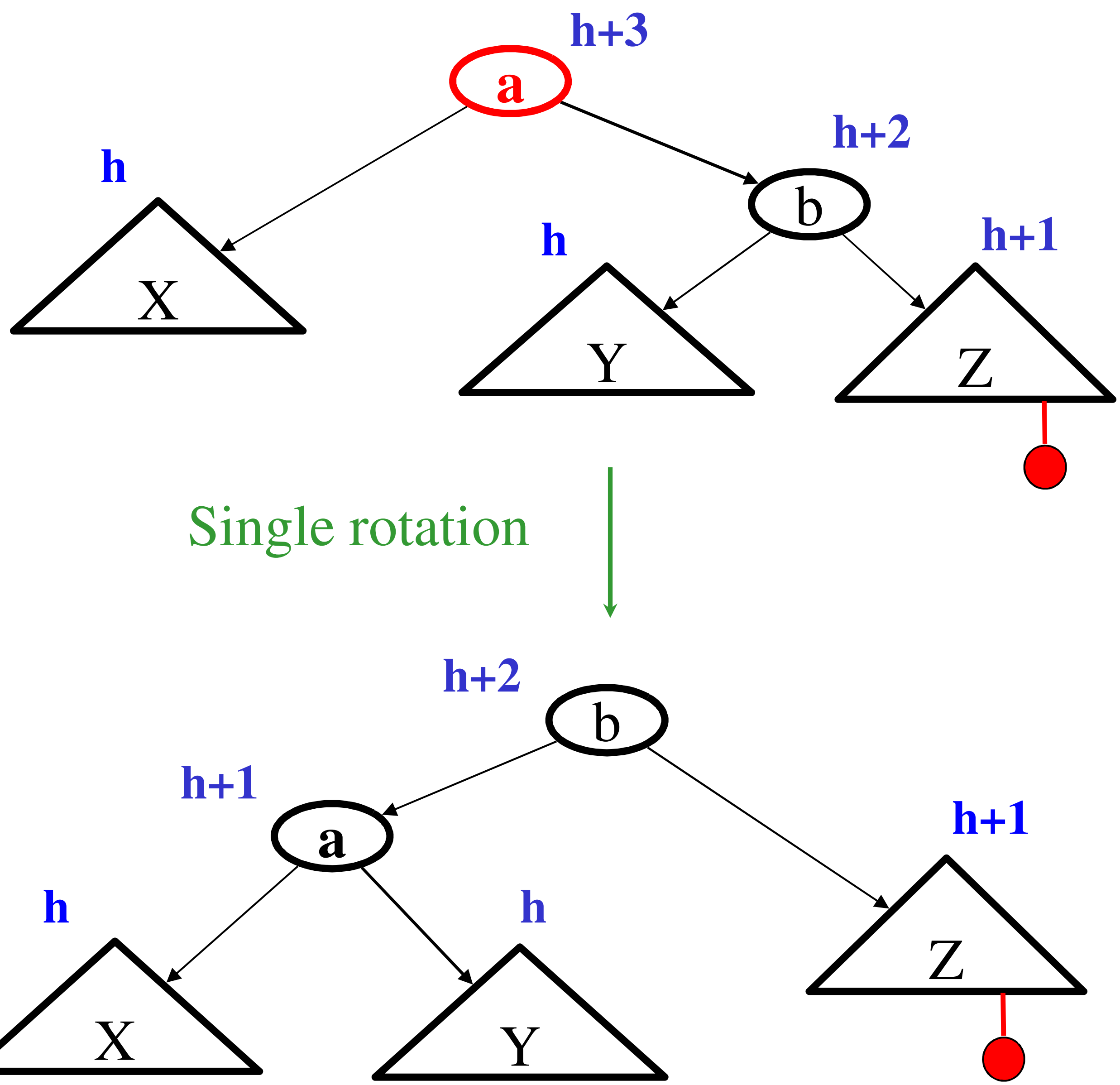




Case #3: right-left insertion



Case #4: right-right insertion



AVL tree case summary

Let a be the node where an imbalance occurs.

Four cases to consider. The insertion below a is in the

1. **left** child's **left** subtree. (zig)
2. **left** child's **right** subtree. (zig-zag)
3. **right** child's **left** subtree. (zig-zag)
4. **right** child's **right** subtree. (zig)

Cases 1 & 4 are solved by a single rotation:

1. Rotate between a and child

Cases 2 & 3 are solved by a double rotation:

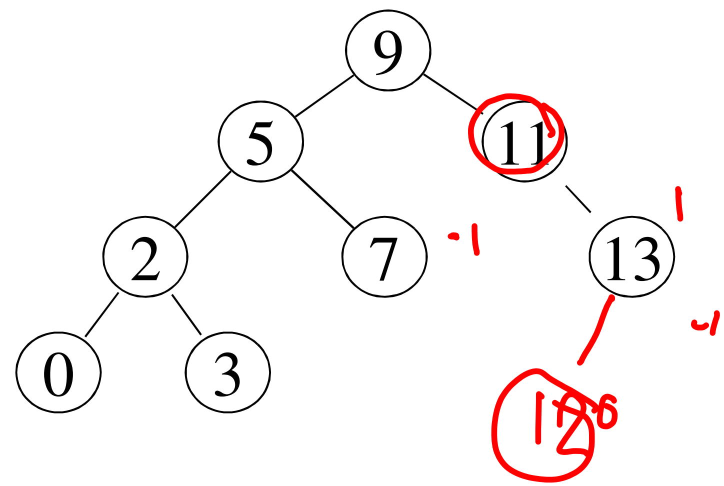
1. Rotate between a's child and grandchild
2. Rotate between a and a's new child

Single and Double Rotations:

Consider inserting one of {1, 4, 6, 8, 10, 12, 14}
Which values require:

1. single rotation?

1, 14



2. double rotation?

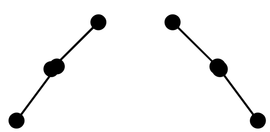
4, 12

3. no rotation?

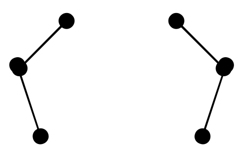
6, 8, 10

Insertion procedure

1. Find spot for new key
2. Hang new node there with this key
3. Search back up the path for imbalance
4. If there is an imbalance:



cases #1,#4: Perform single rotation and exit

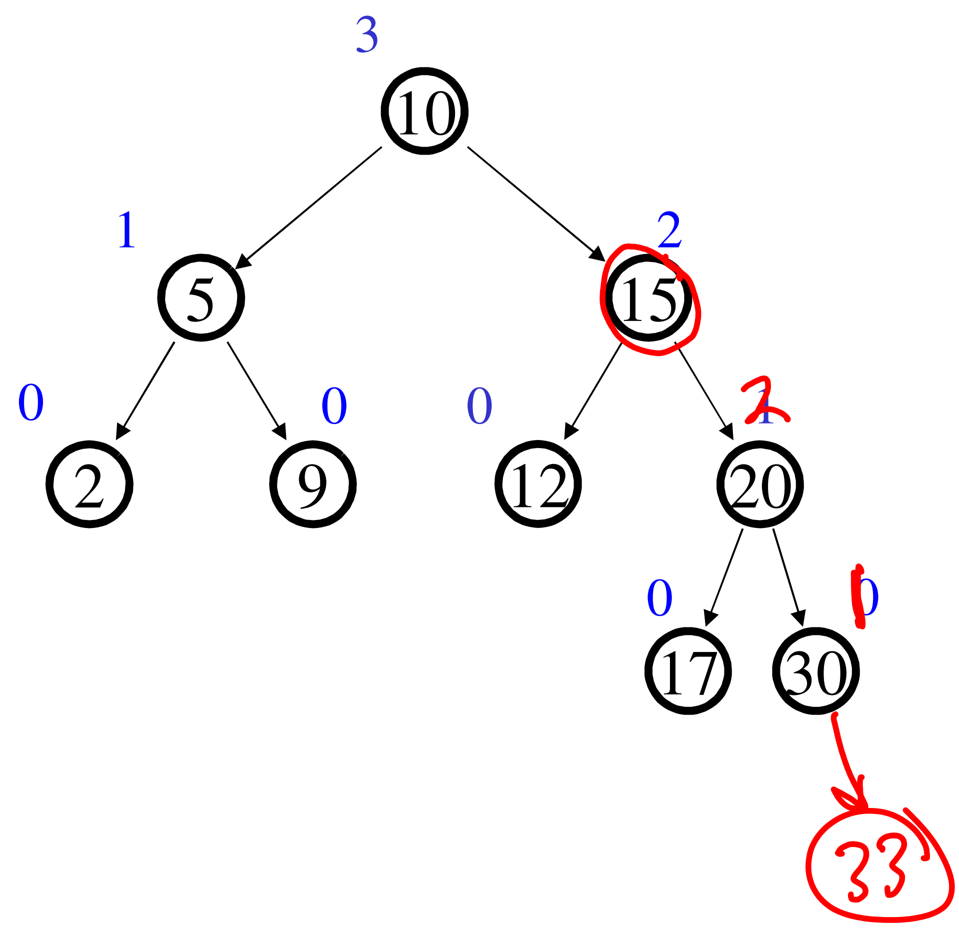


cases #2,#3: Perform double rotation and exit

Both rotations restore subtree height to value before insert.
Hence only type of rotation is sufficient per insert!

More insert examples

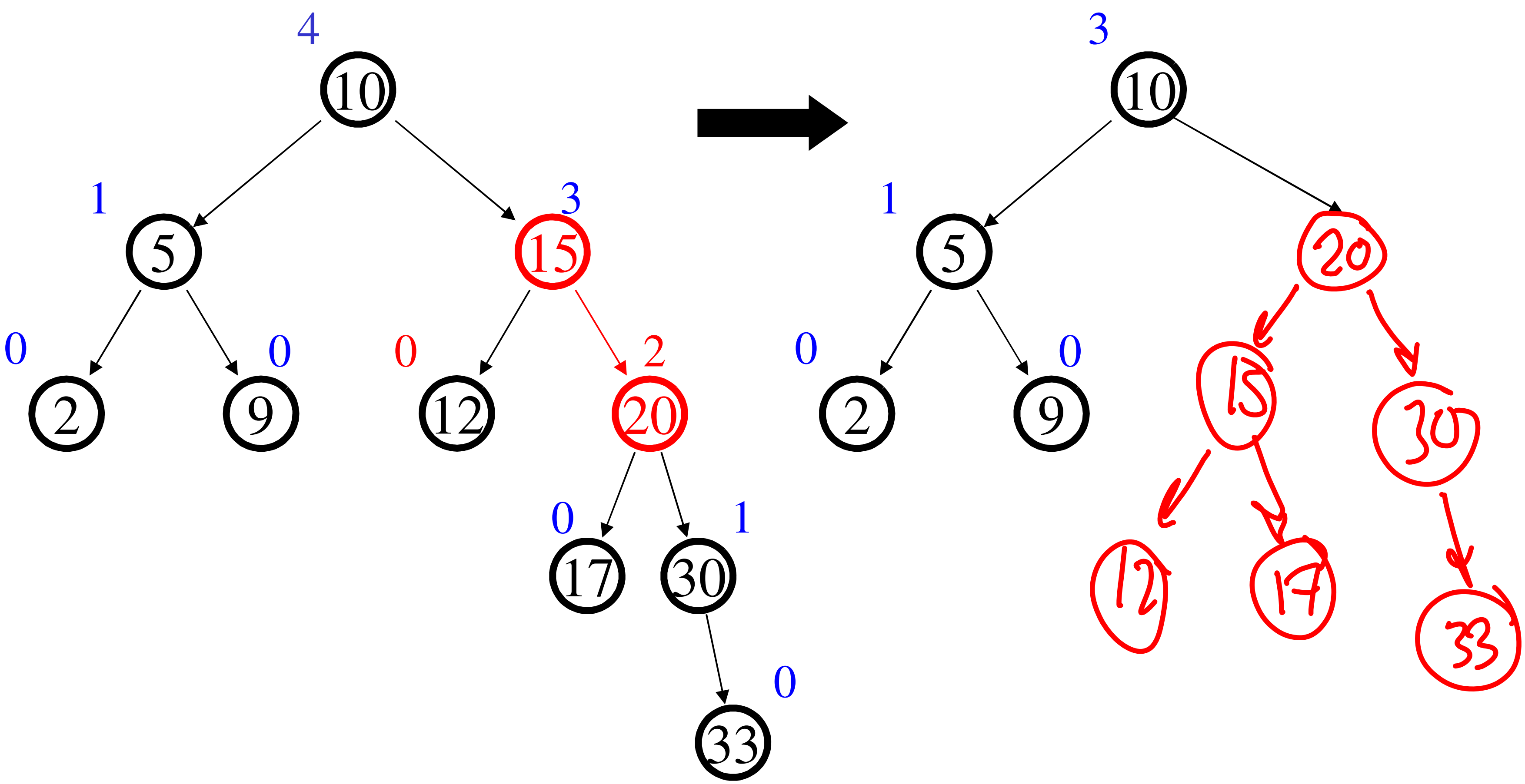
Insert(33)



Unbalanced?

How to fix?

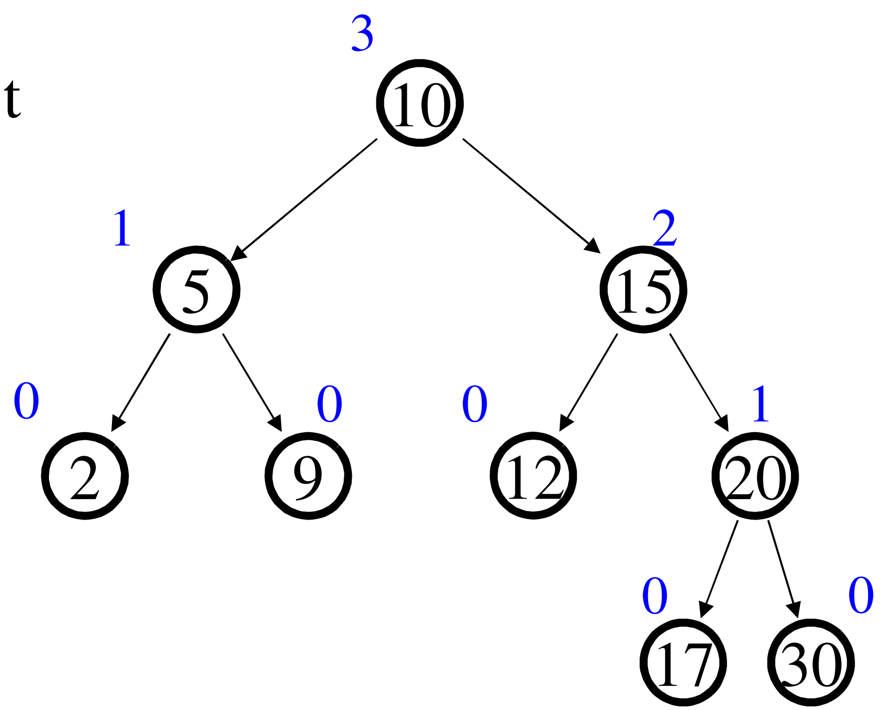
Single Rotation



More insert examples

Suppose we didn't do that last insert.

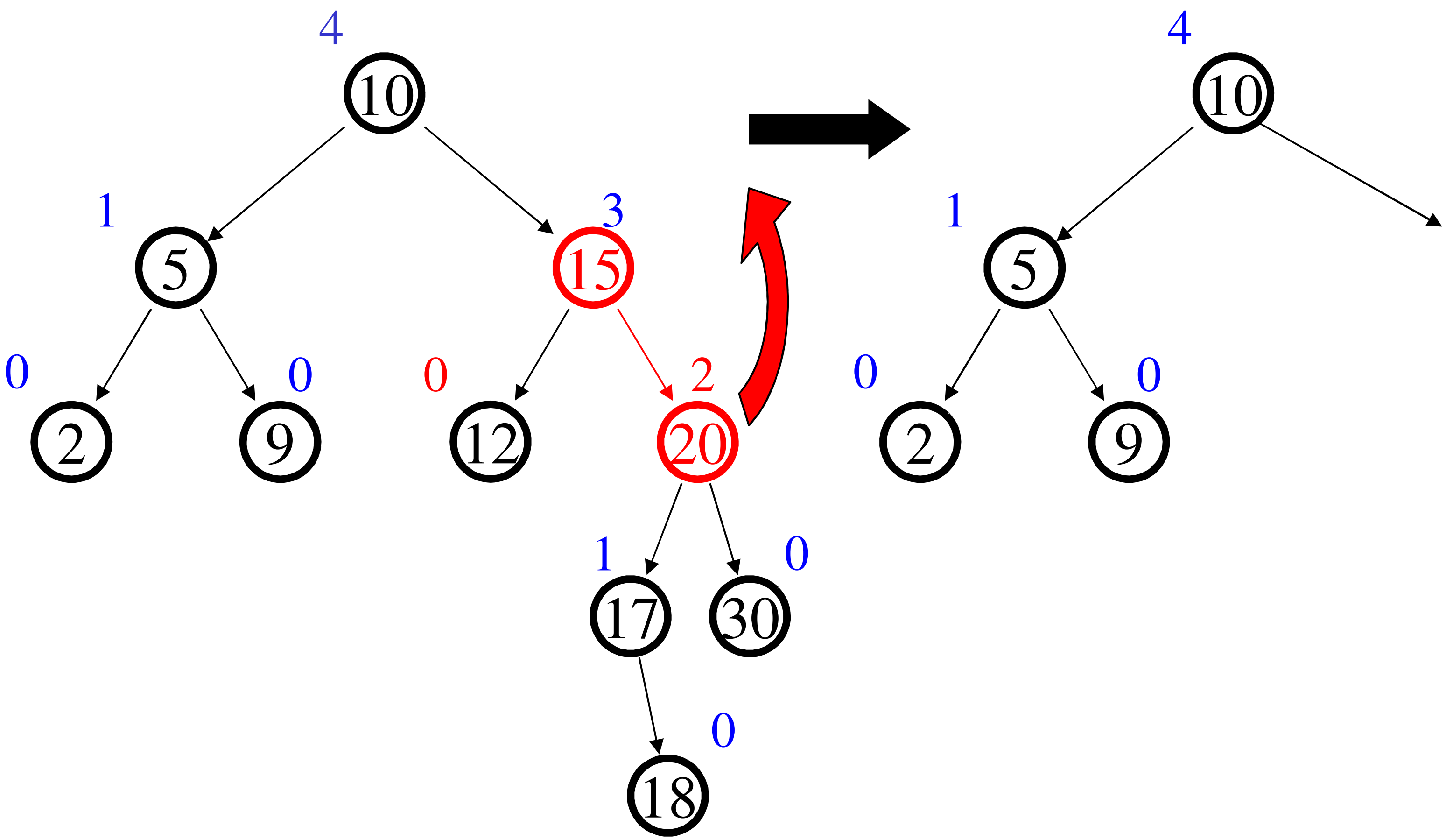
Now do:
  Insert(18)



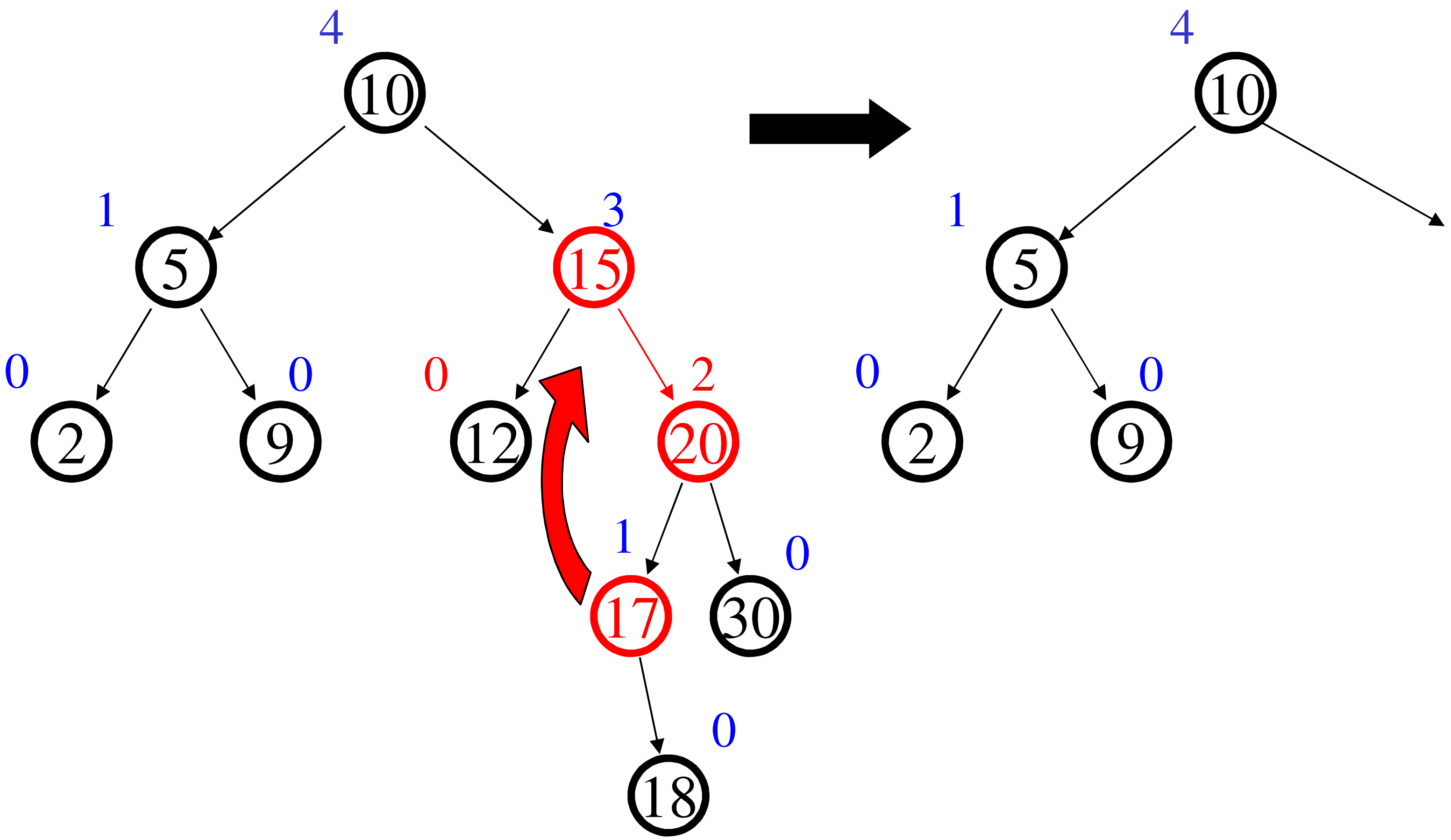
Unbalanced?

How to fix?

Single Rotation (oops!)

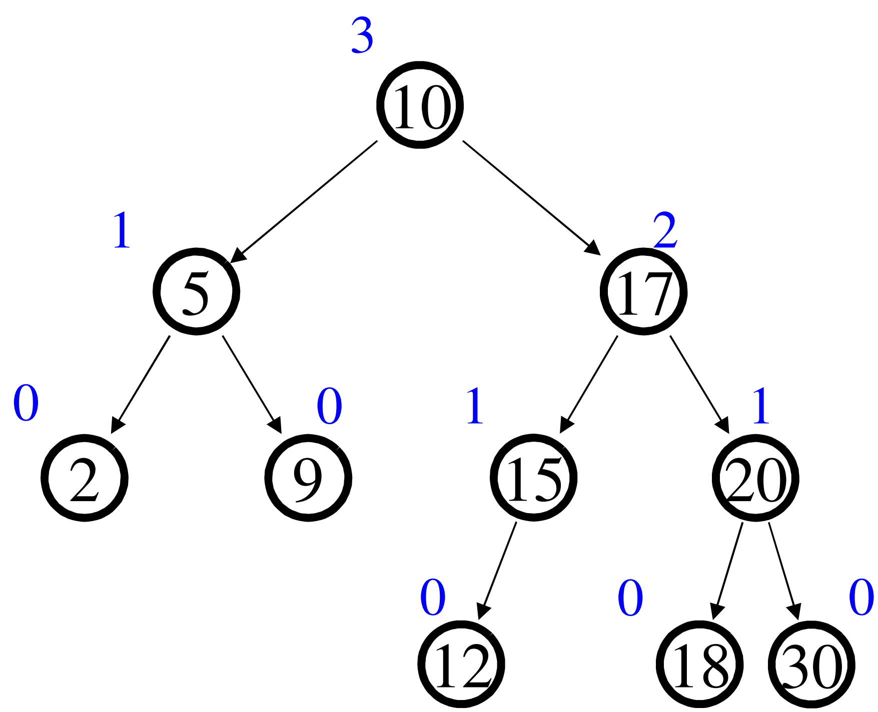


Double Rotation



More insert examples

Insert(3)



Unbalanced?

How to fix?

Insert into an AVL tree: 5, 8, 9, 4, 2, 7, 3, 1

AVL complexity

What is the worst case complexity of a find?

$O(\log n)$

What is the worst case complexity of an insert?

find: $O(\log n)$ + detect imbalance $O(\log n)$ + $O(1)$

$\Rightarrow O(\log n)$

What is the worst case complexity of buildTree?

$O(n \log n)$