6.006 Recitation

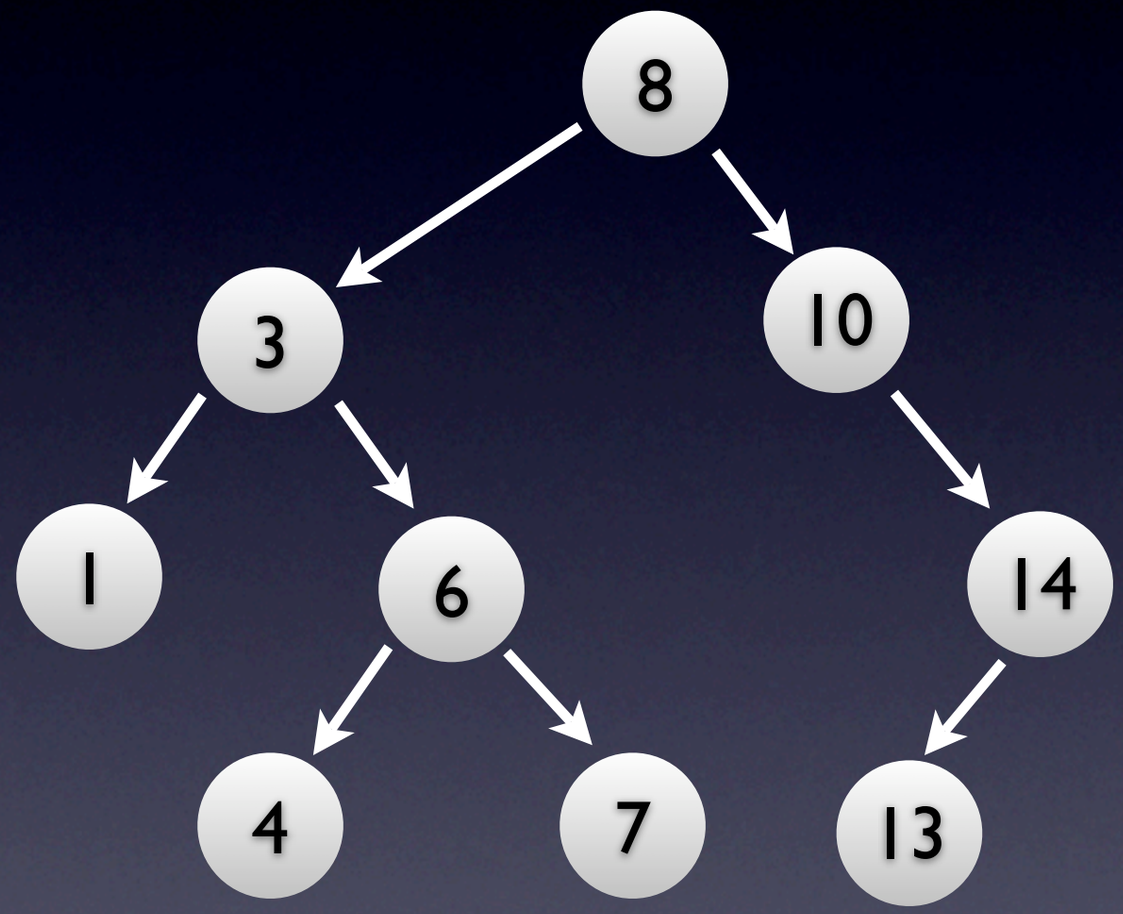
Build 2008.7

Outline

- Basic concepts review
- AVL algorithms
- Python implementation for AVLs

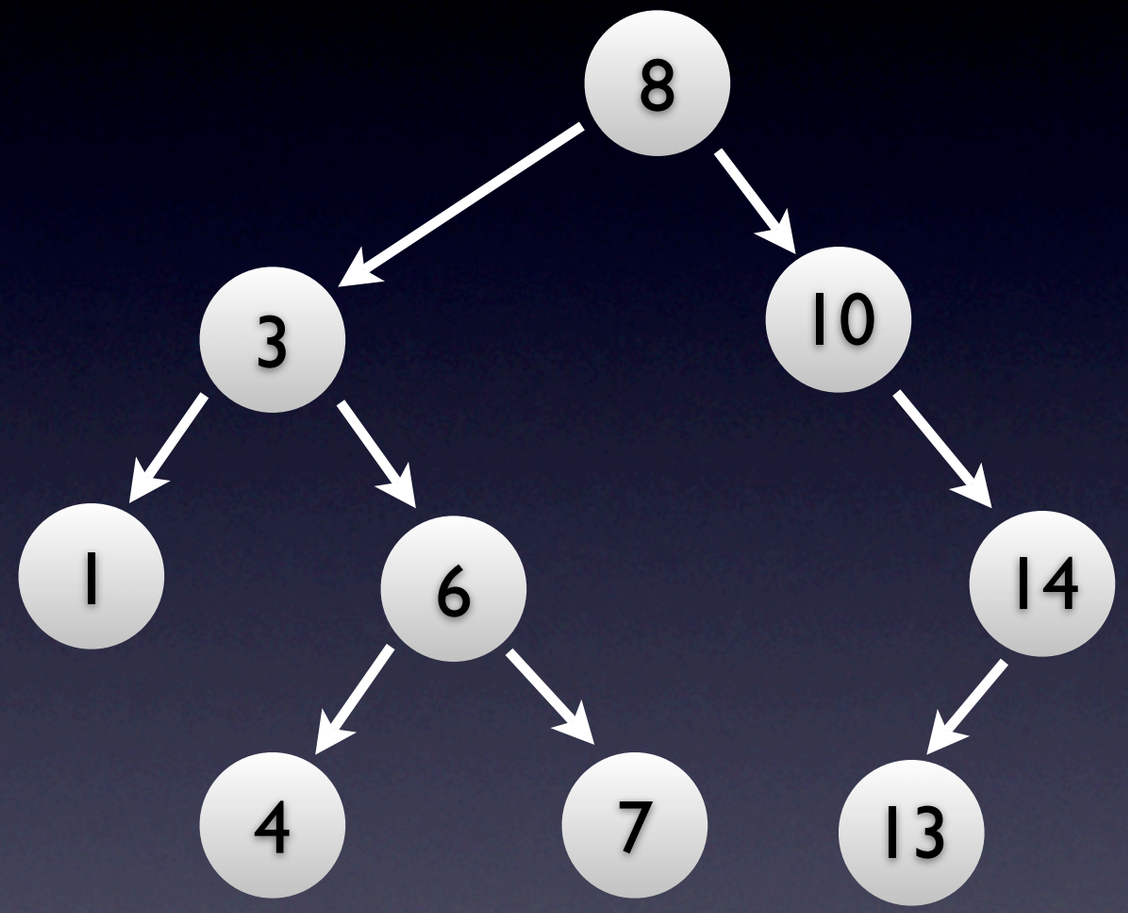
BST Invariants

- Binary rooted tree
- All left descendants have keys < node's key
- All right descendants have keys > node's key



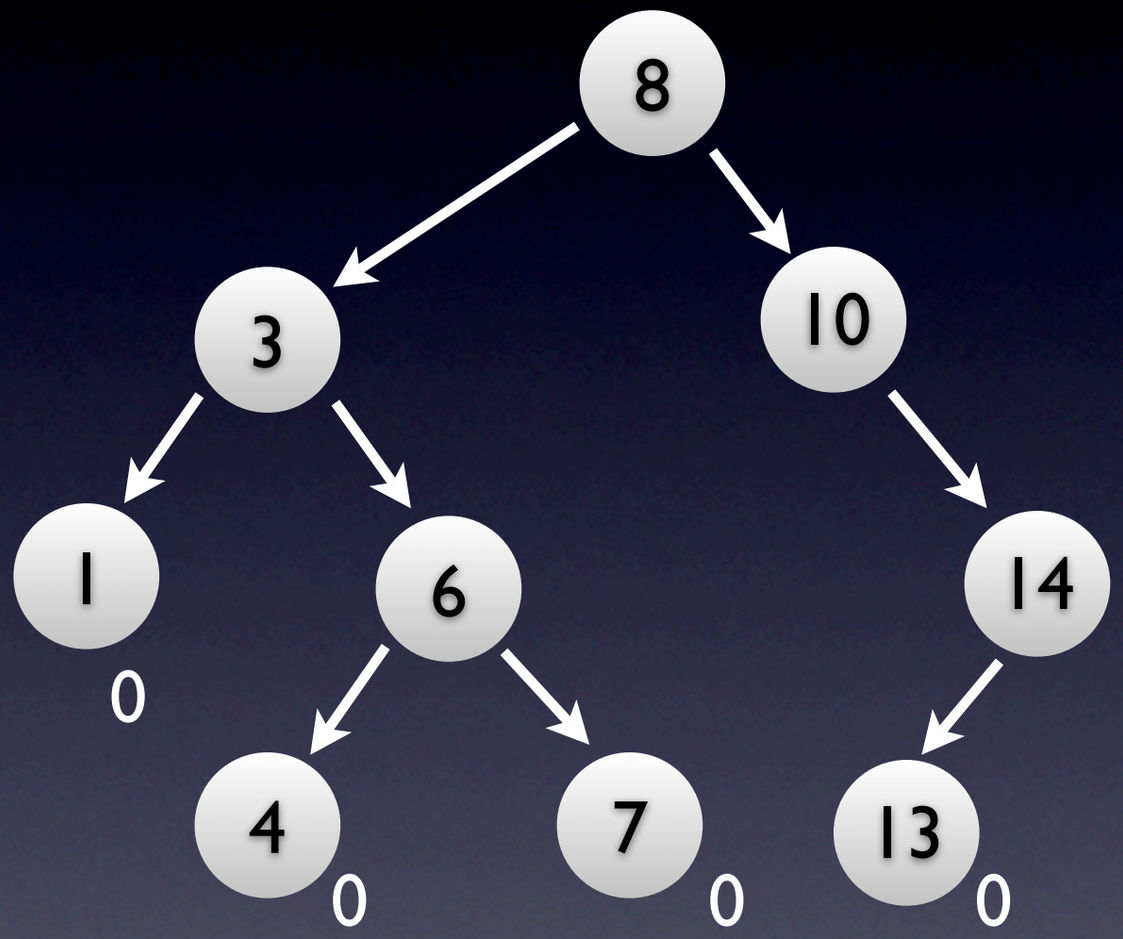
Node Height

- Leaves: height = 0
- Inner nodes: height = max(children height) + 1
- Null tree: height = -1
- Rationale:
  - a subtree operation takes O(h) time



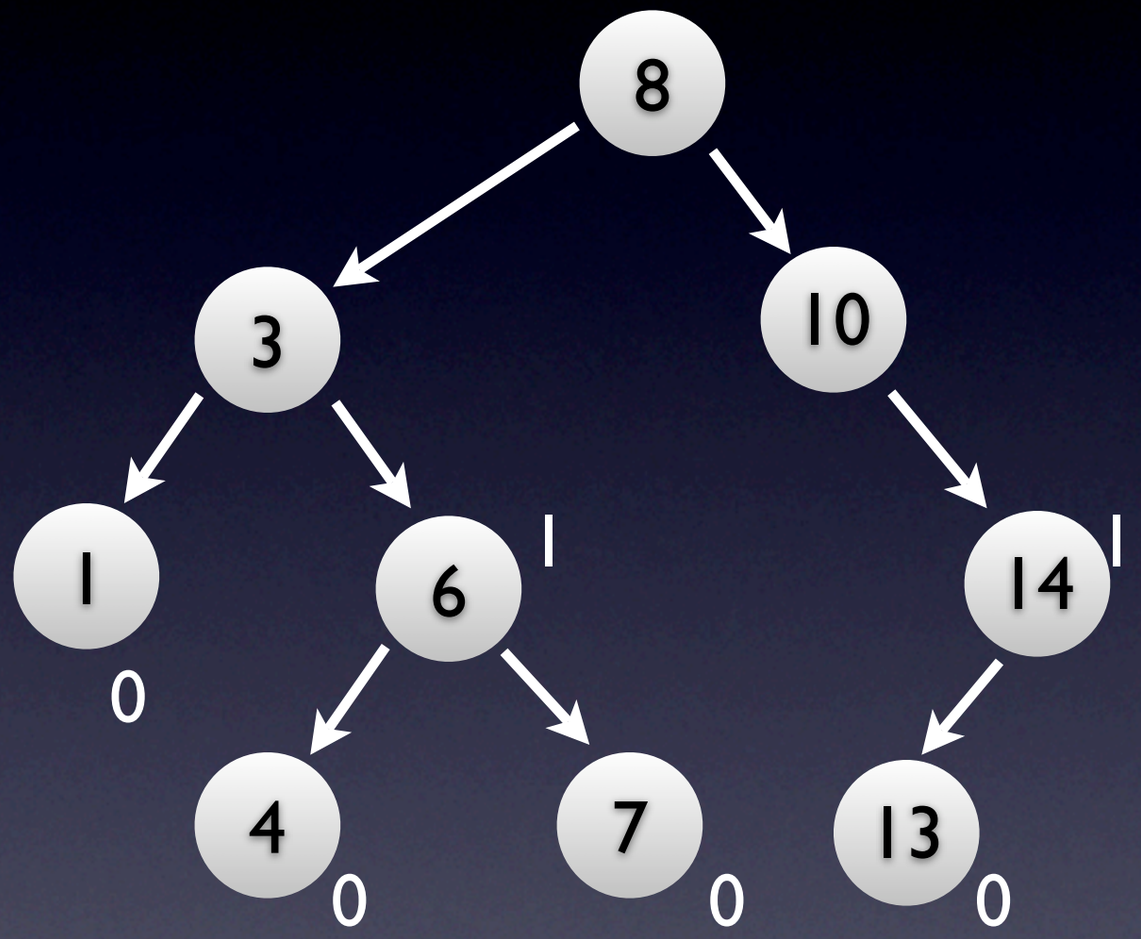
Node Height

- Leaves: height = 0
- Inner nodes: height = max(children height) + 1
- Null tree: height = -1
- Rationale:
  - a subtree operation takes O(h) time



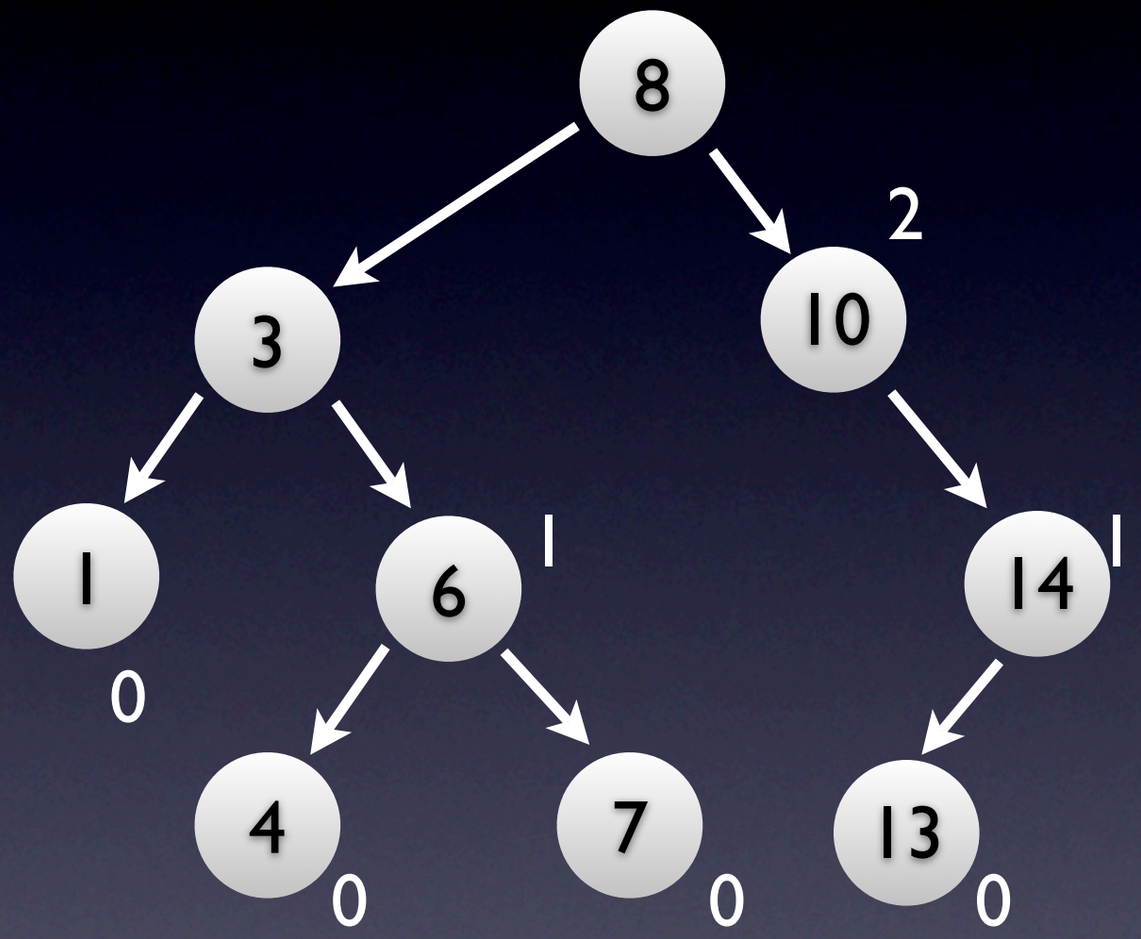
Node Height

- Leaves: height = 0
- Inner nodes: height = max(children height) + 1
- Null tree: height = -1
- Rationale:
  - a subtree operation takes O(h) time



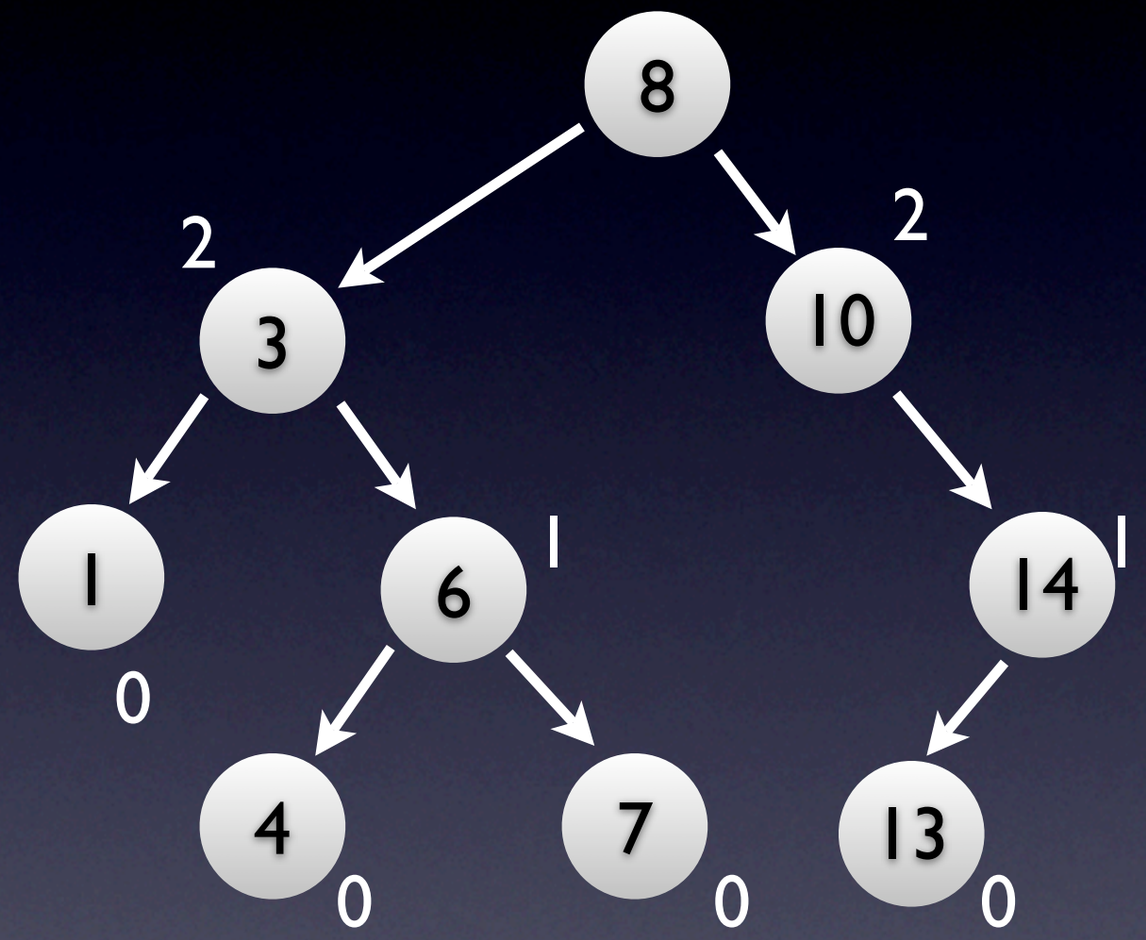
Node Height

- Leaves: height = 0
- Inner nodes: height = max(children height) + 1
- Null tree: height = -1
- Rationale:
  - a subtree operation takes O(h) time



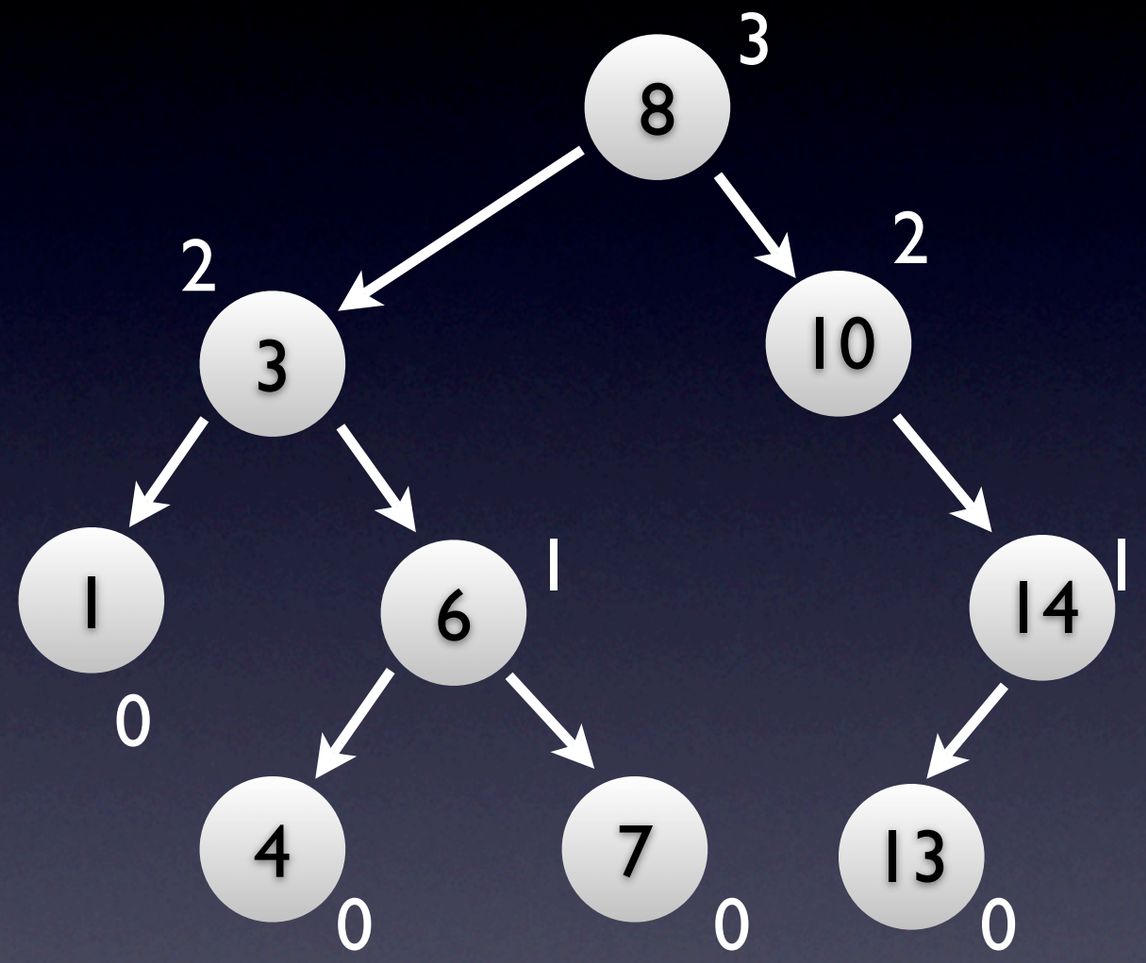
Node Height

- Leaves: height = 0
- Inner nodes: height = max(children height) + 1
- Null tree: height = -1
- Rationale:
  - a subtree operation takes O(h) time



Node Height

- Leaves: height = 0
- Inner nodes: height = max(children height) + 1
- Null tree: height = -1
- Rationale:
  - a subtree operation takes O(h) time

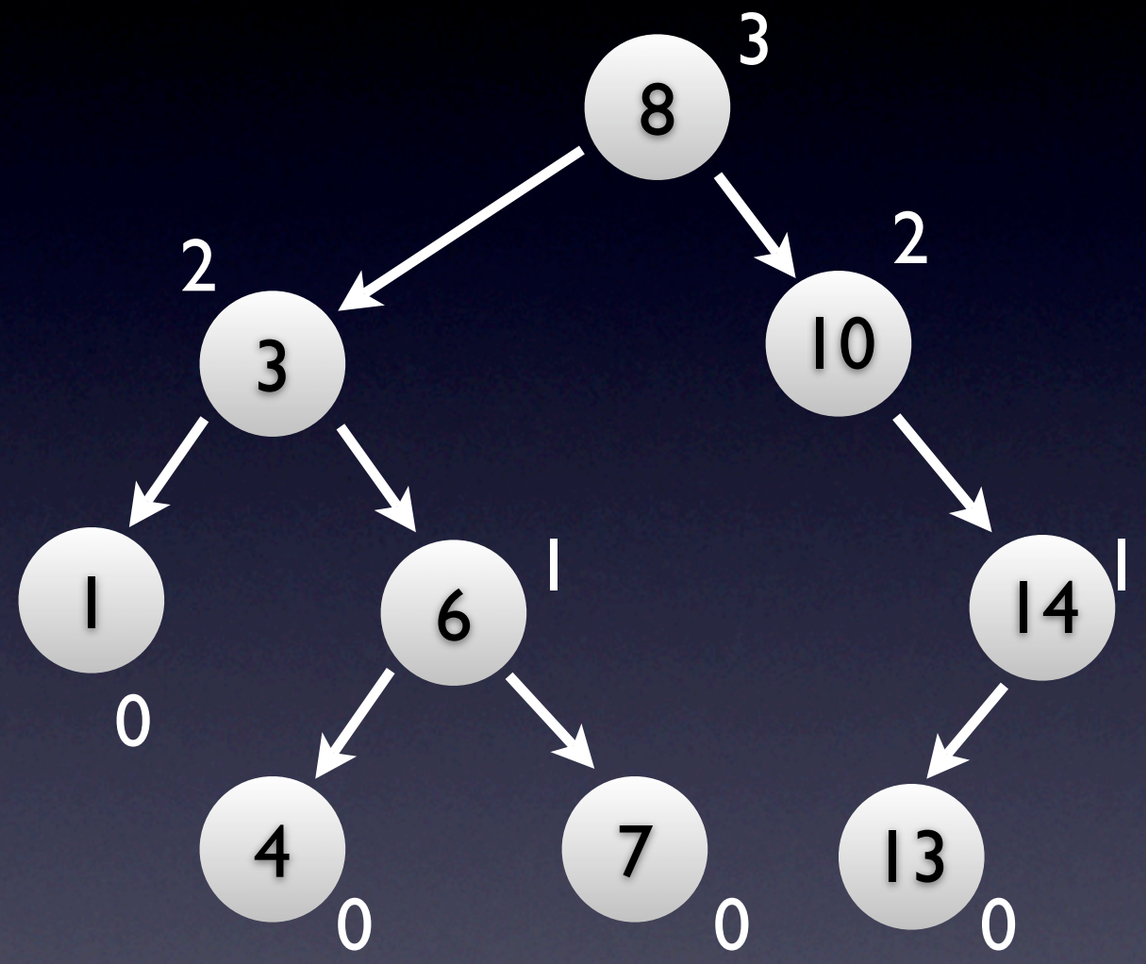


Balanced Trees

- Small tree height means fast operations
- Pack many nodes in trees with low heights
- Perfectly balanced tree: $2^{h+1} - 1$ nodes
- We only care about asymptotic notation
  - Nodes = f(height) must be exponential

AVL Trees

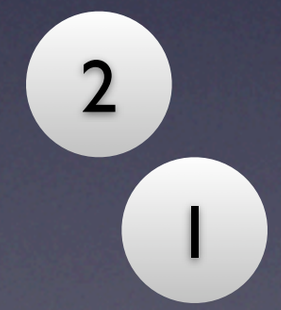
- Regular BST with extra invariants:
  - absolute value(left child height - right child height) <= 1
  - Each subtree is AVL



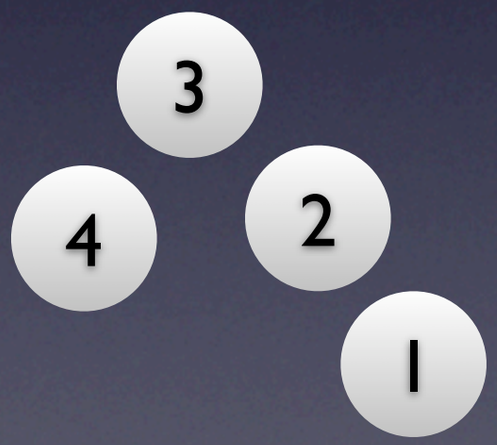
Least dense AVL

Least dense AVL

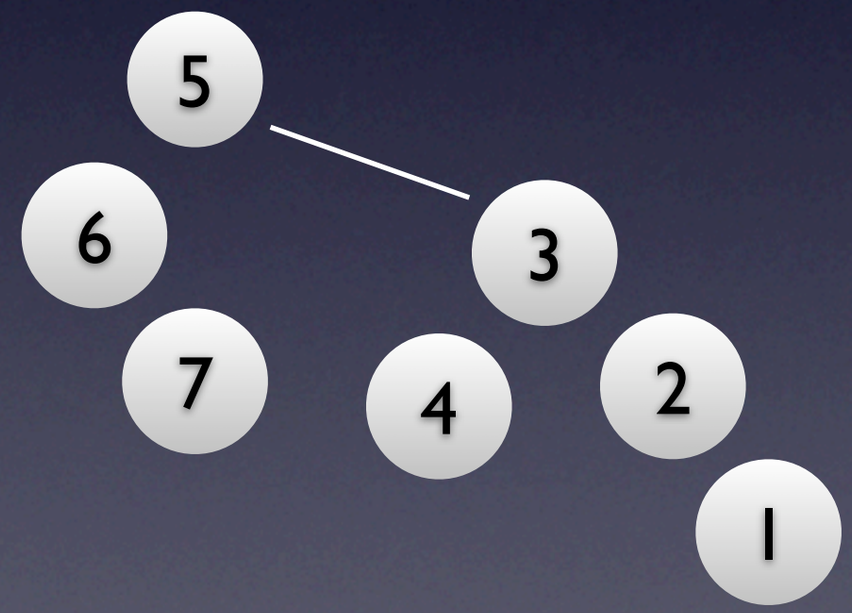
Least dense AVL



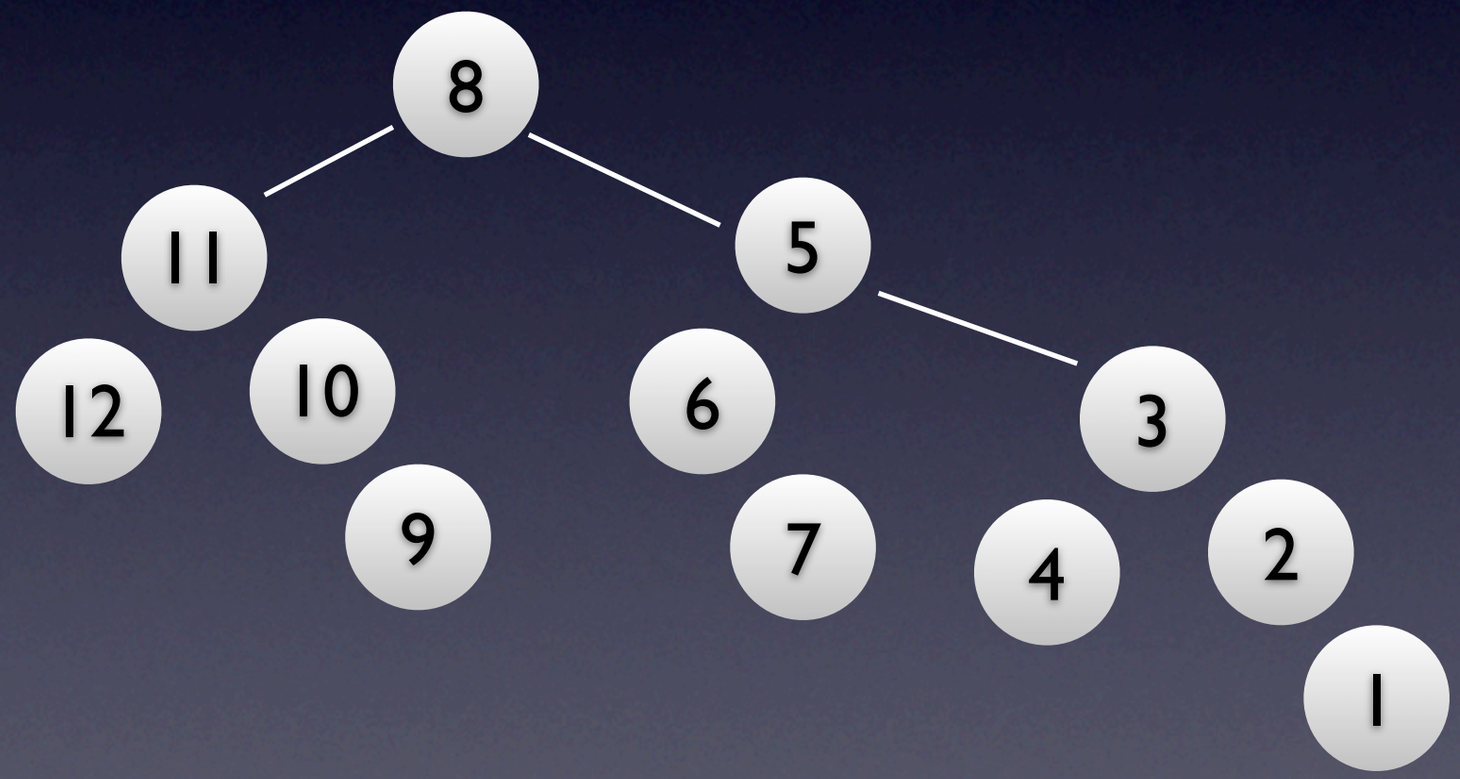
Least dense AVL



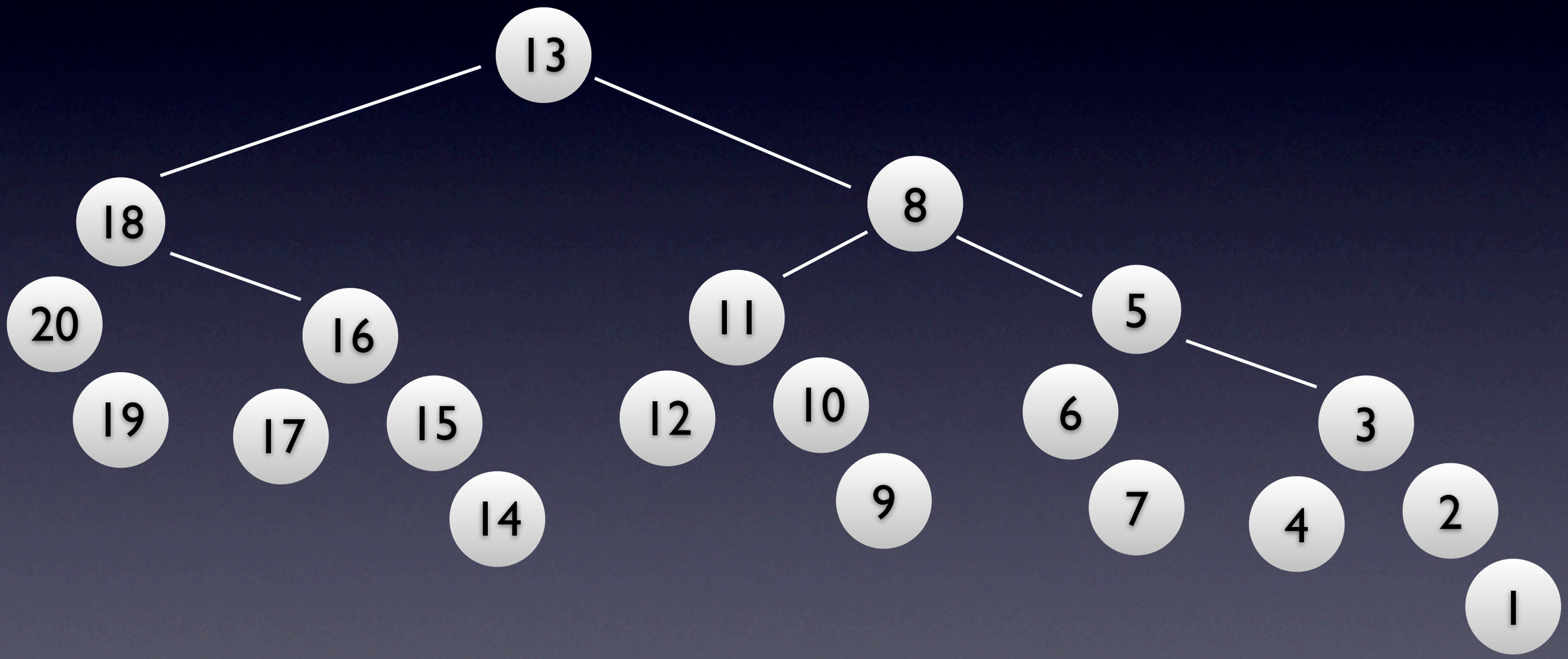
Least dense AVL



Least dense AVL

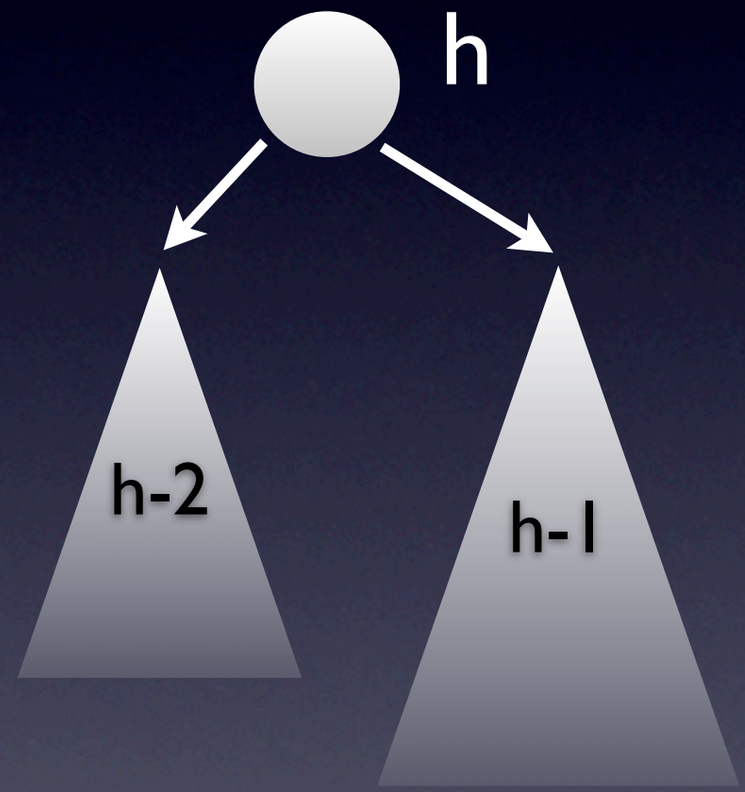


Least dense AVL



Least dense structure

- Nodes(-1) =0
- Nodes(0) = 1
- Nodes(h) = 1 + Nodes(h-1) + Nodes(h-2)
- Looks like Fibonacci, must be exponential



Pwnage with AVLs 101

- Goals
  - Reuse the code we wrote before
  - Start with an AVL, end up with an AVL
- Managerial Input (the ‘doh’ words)
  - Insert and delete like it’s a BST
  - Patch to make it an AVL again

Key Observation

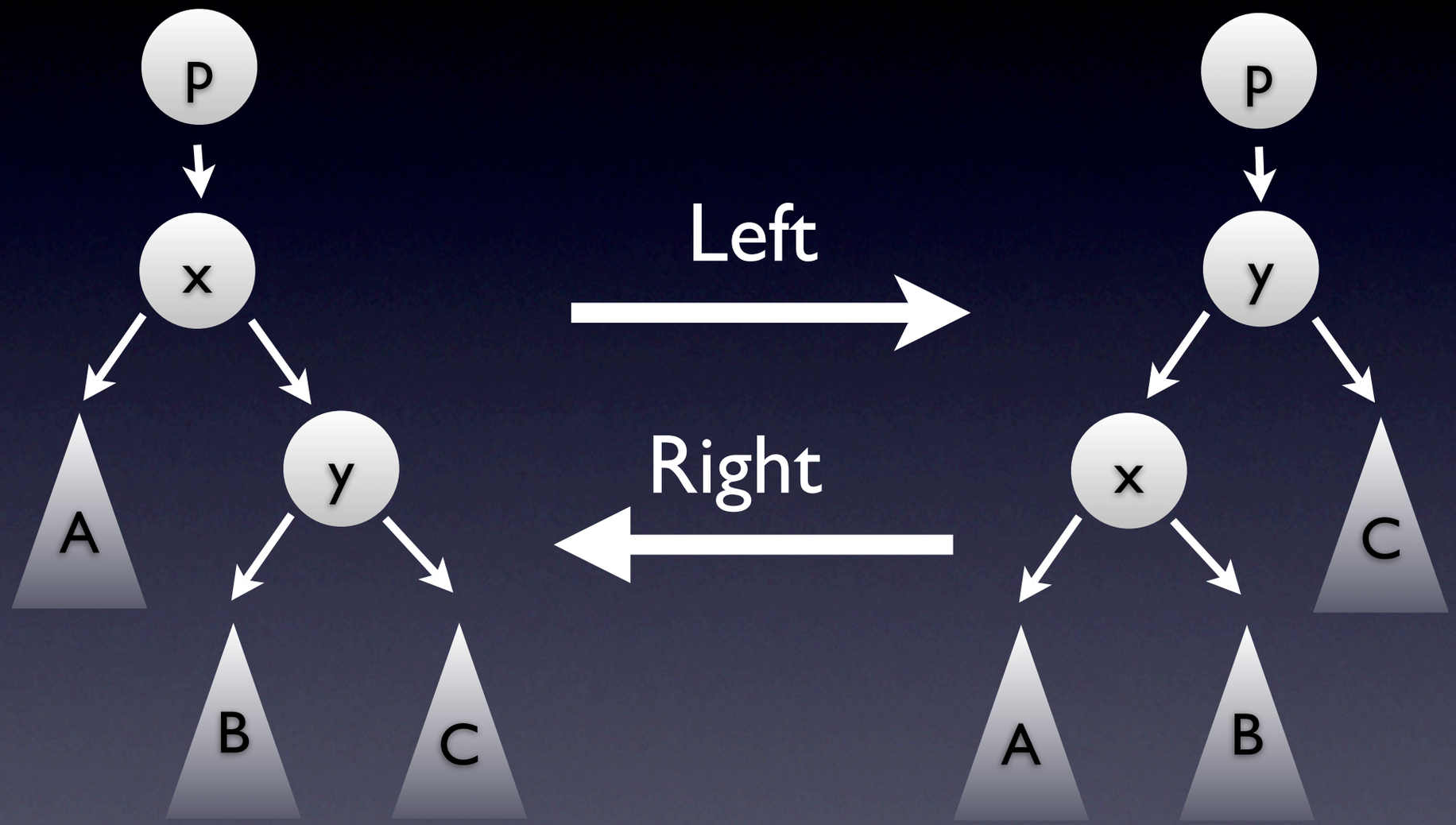
Key Observation

Adding or removing a node only upsets the heights on a single path to the root.

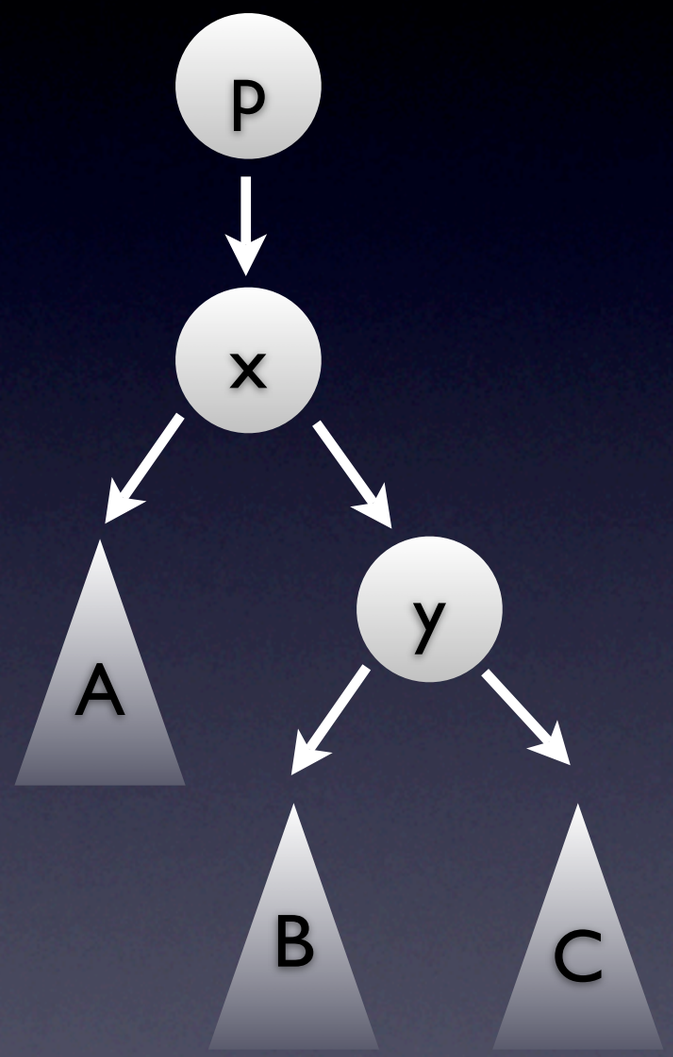
Pwnage with AVLs 201

- Will obviously have to move nodes around
- But must keep track of
  - Height
  - Augmented data
  - Invariants for AVL, BST
- Need a tool that preserves most structure

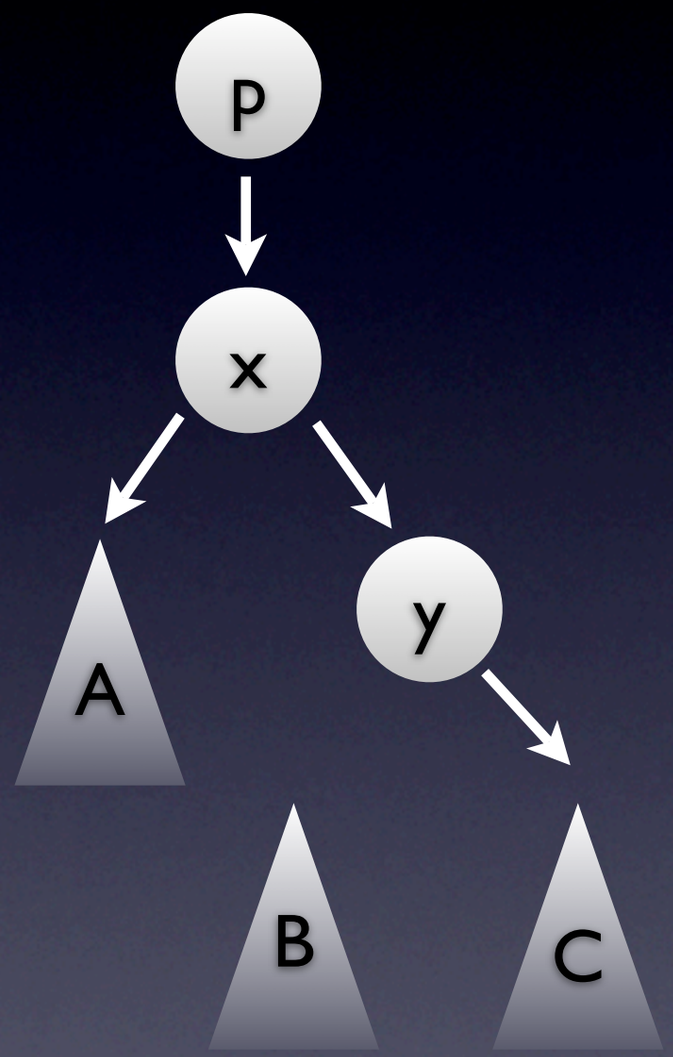
Uberpoke (rotations)



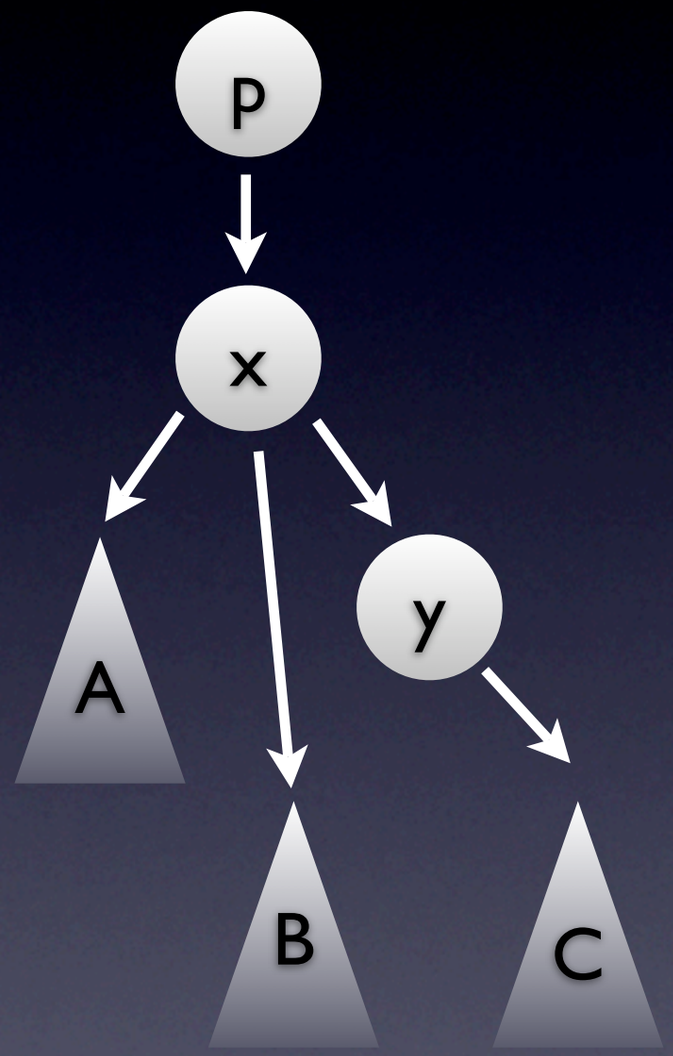
Huh? Do that again ?



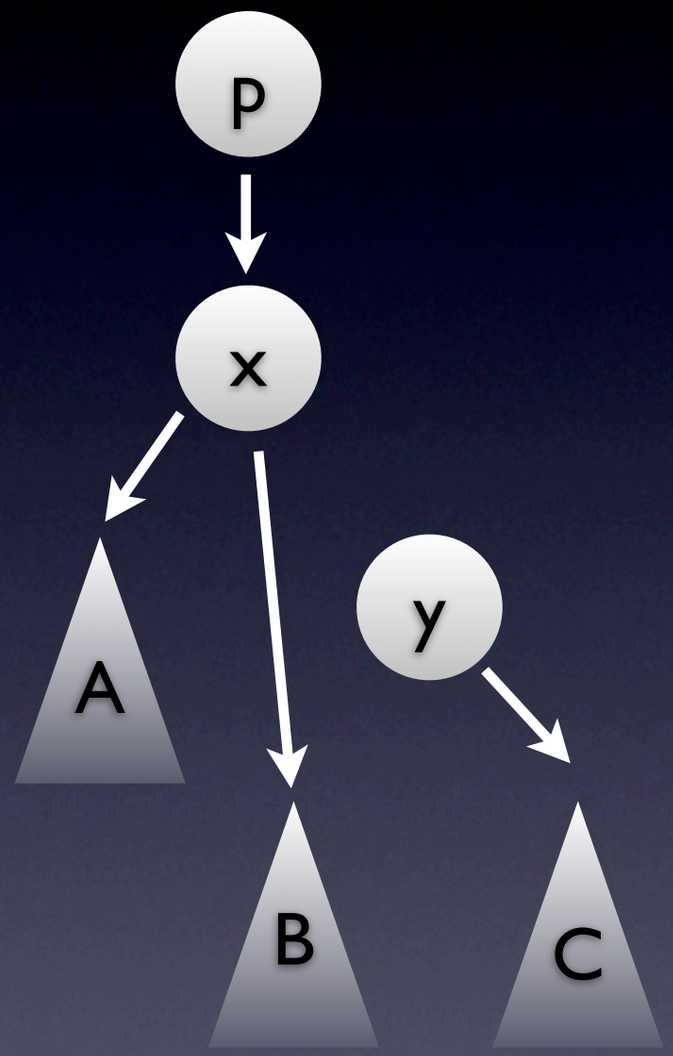
Huh? Do that again ?



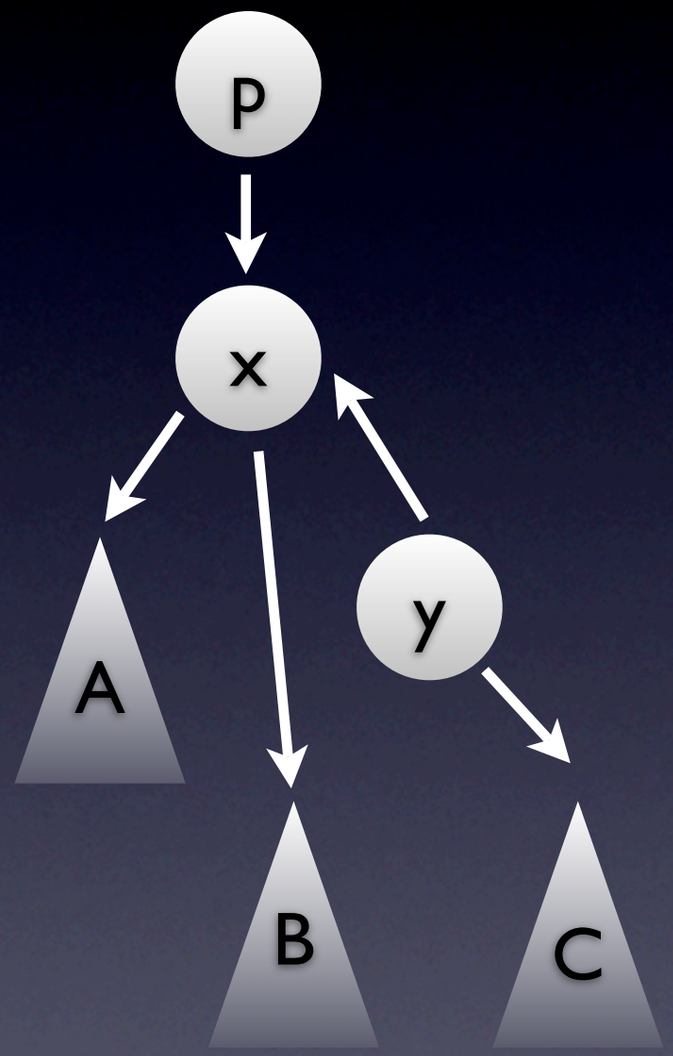
Huh? Do that again ?



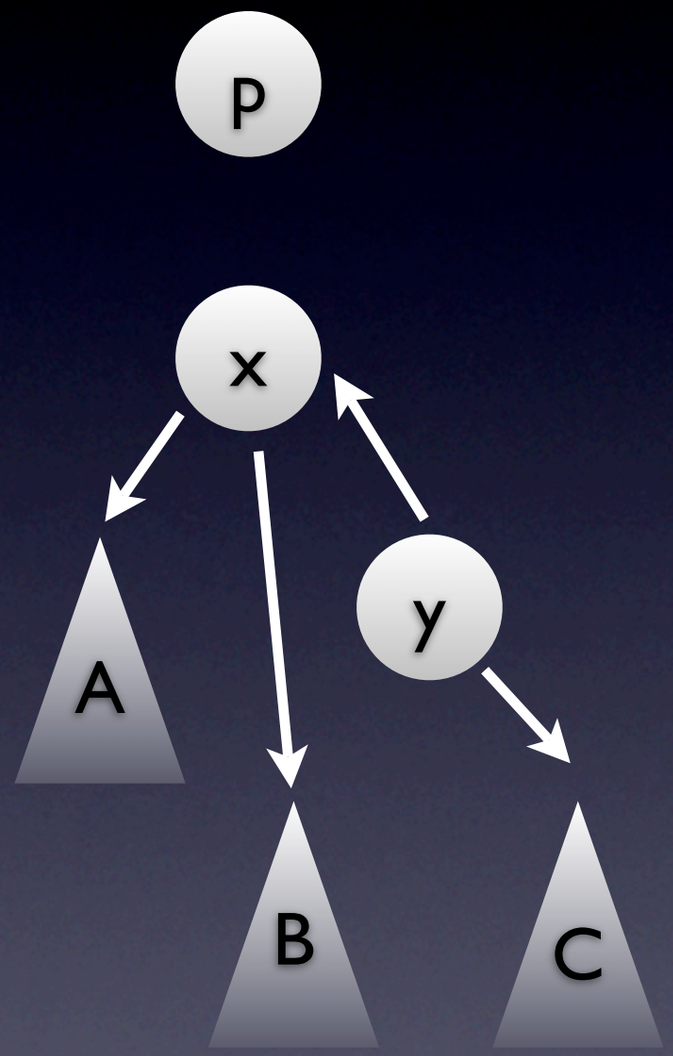
Huh? Do that again ?



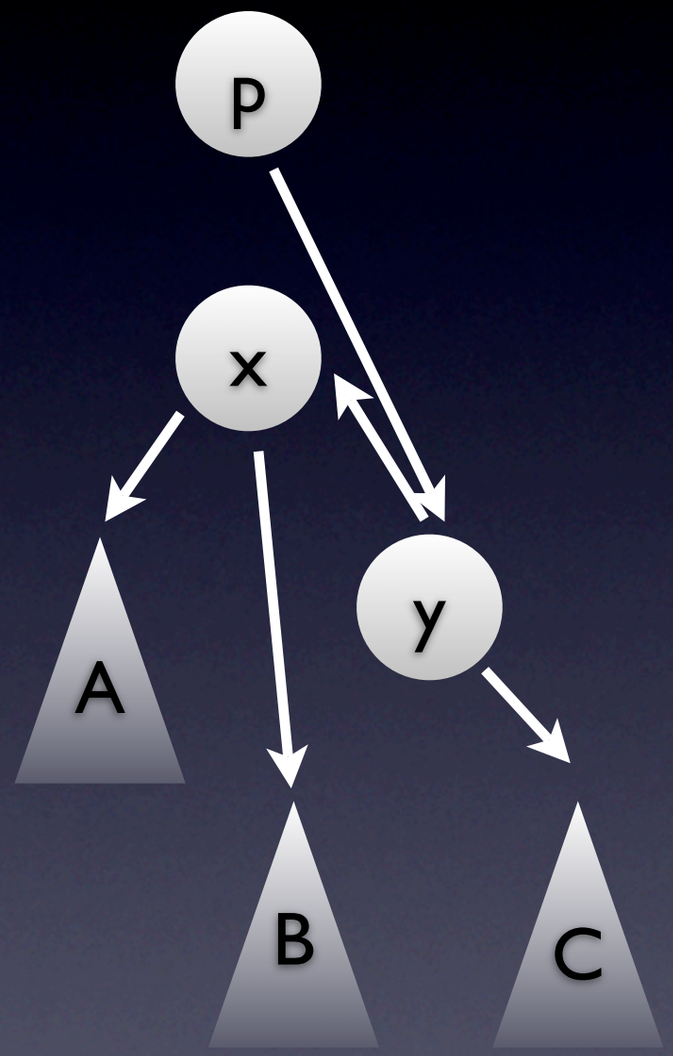
Huh? Do that again ?



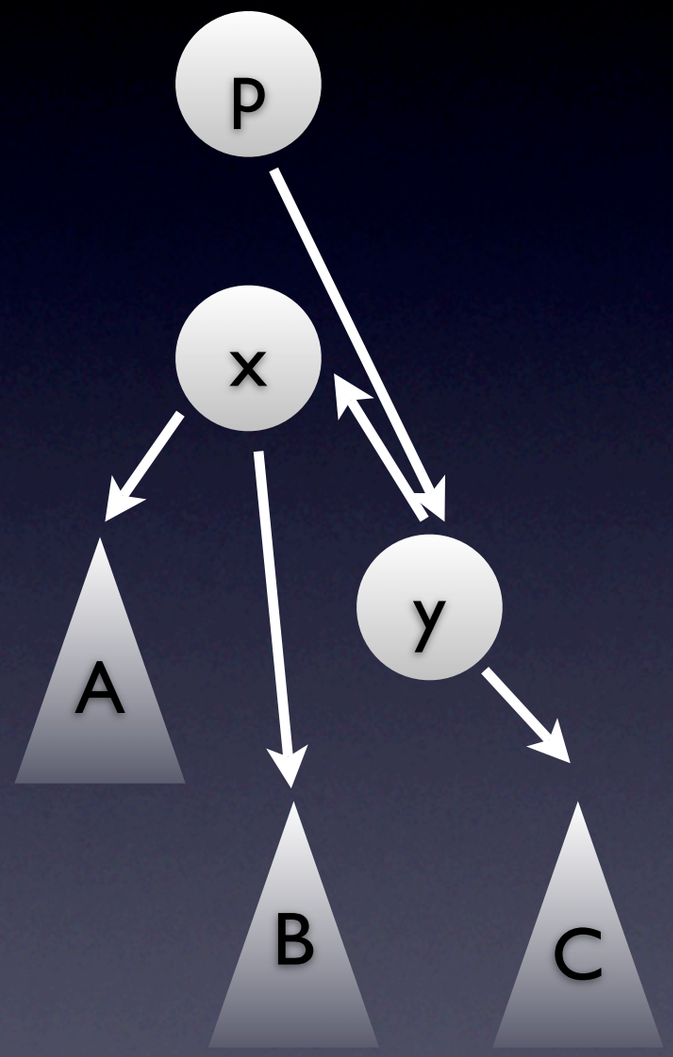
Huh? Do that again ?



Huh? Do that again ?

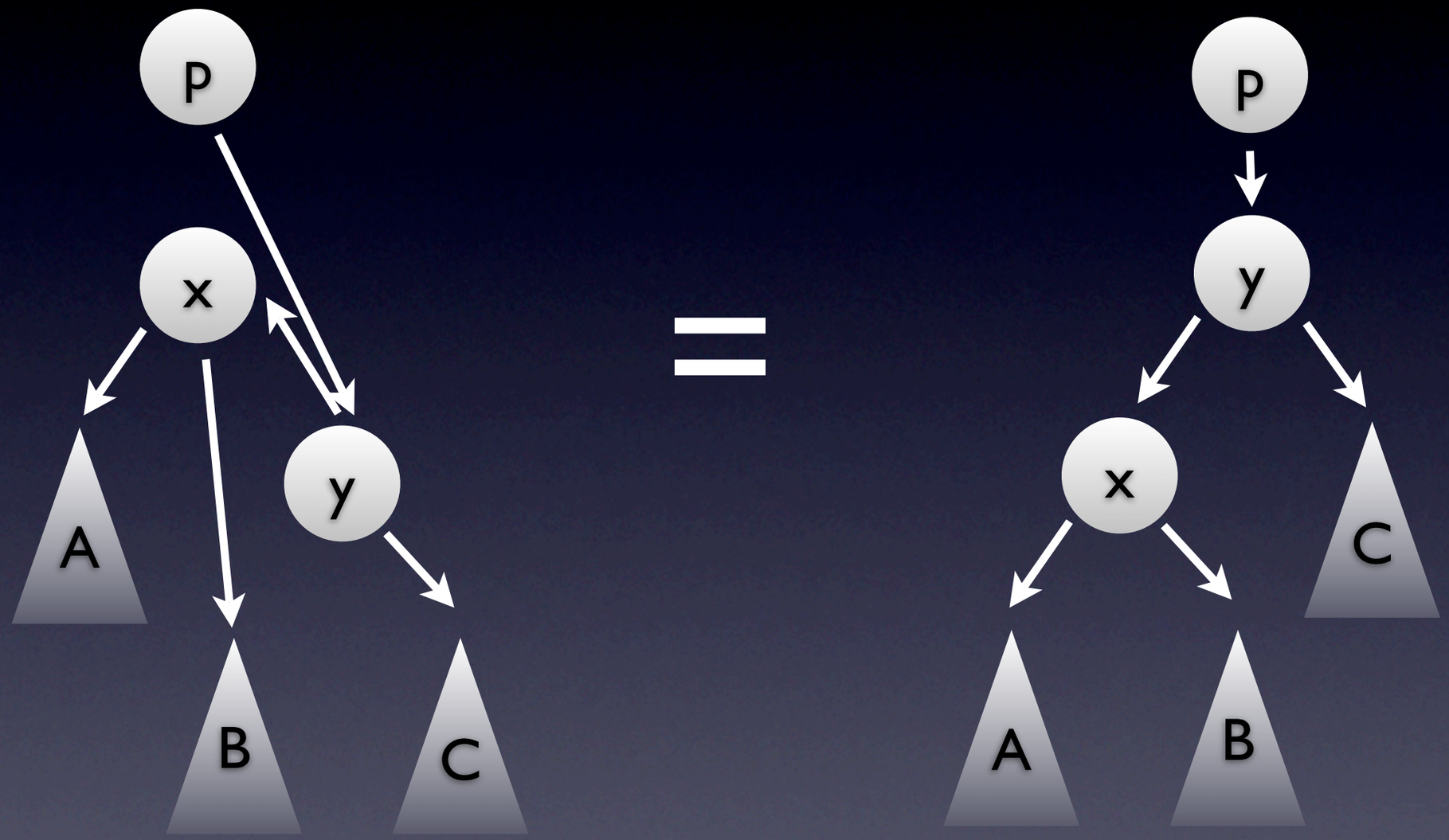


Huh? Do that again ?



=

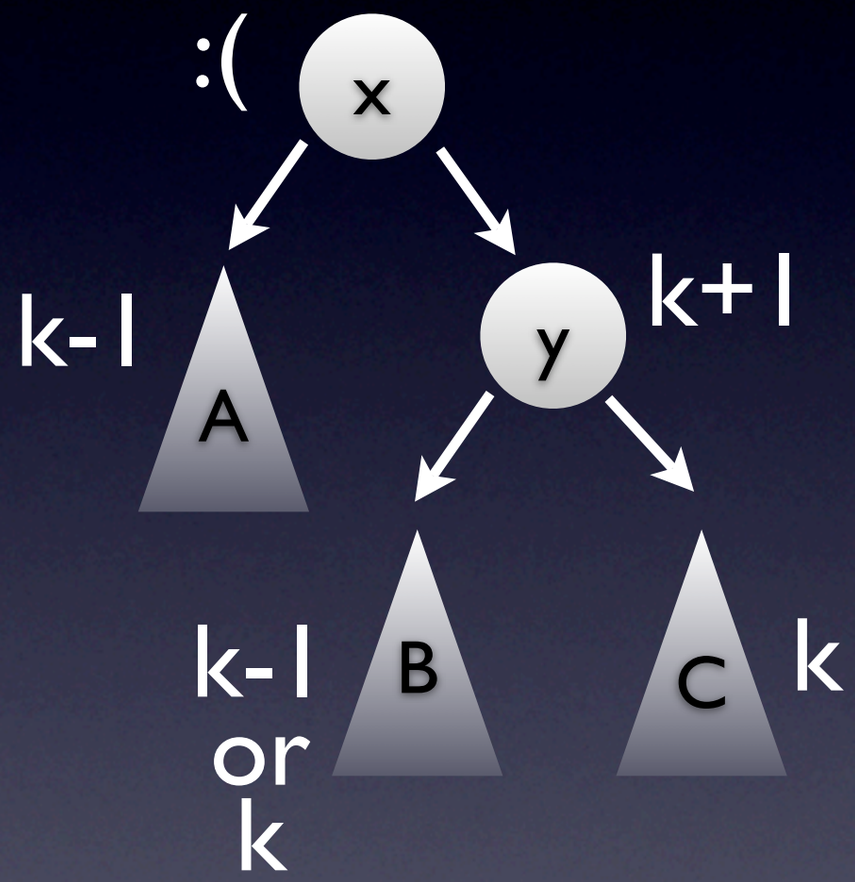
Huh? Do that again ?



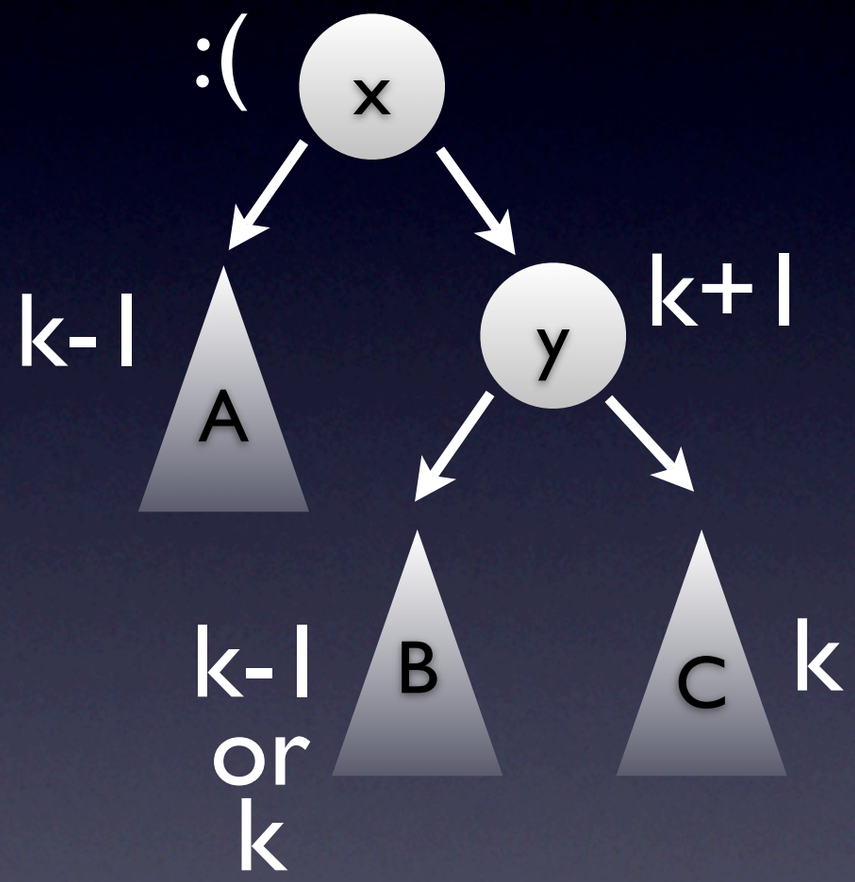
Rebalancing

- Rotations are quite teh uberpoke
- Need a master plan for using them
  - Managerial Input: call it ‘rebalancing’
  - Divide and conquer: start from the bottom, fix up the tree level by level

Rebalancing: easy

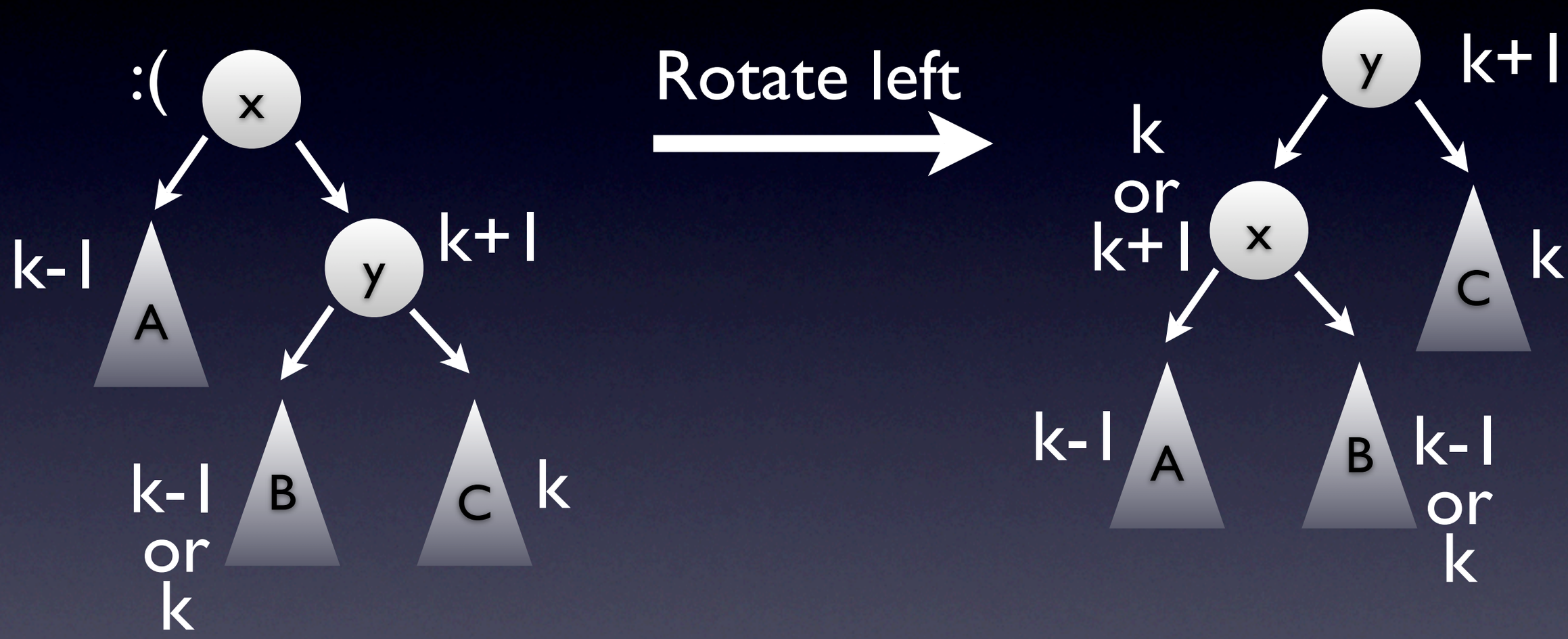


Rebalancing: easy

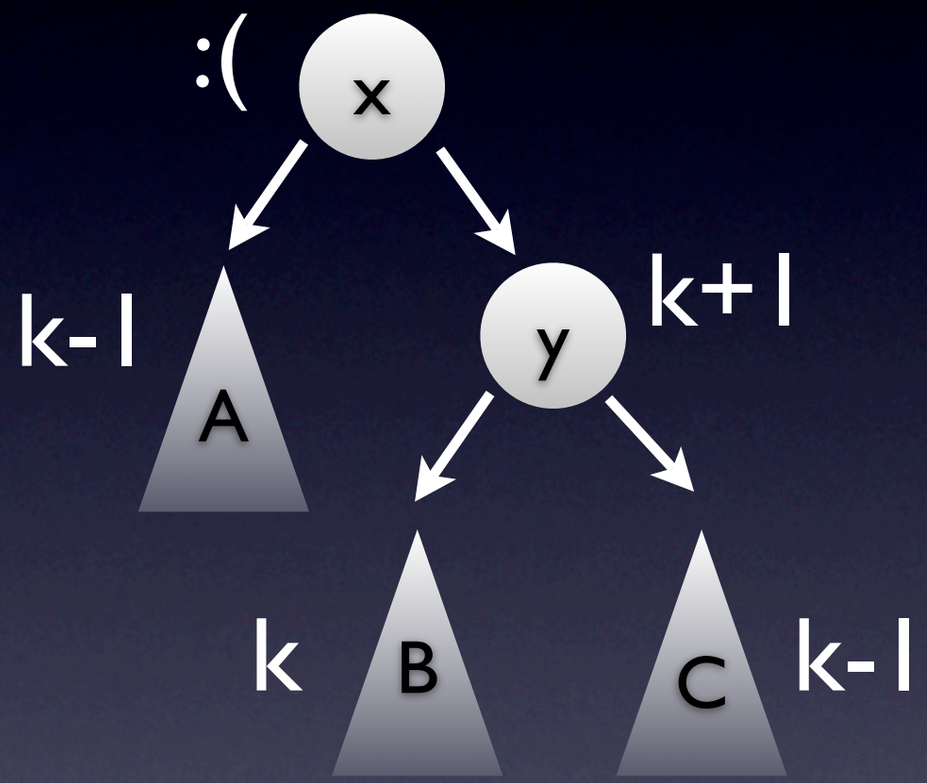


Rotate left

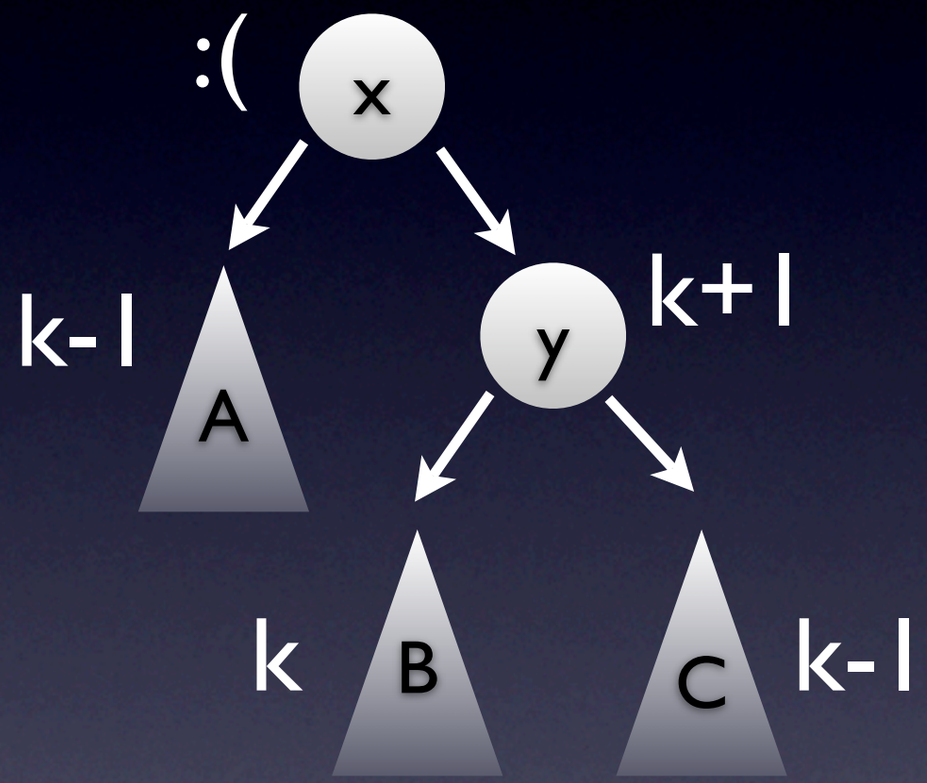
Rebalancing: easy



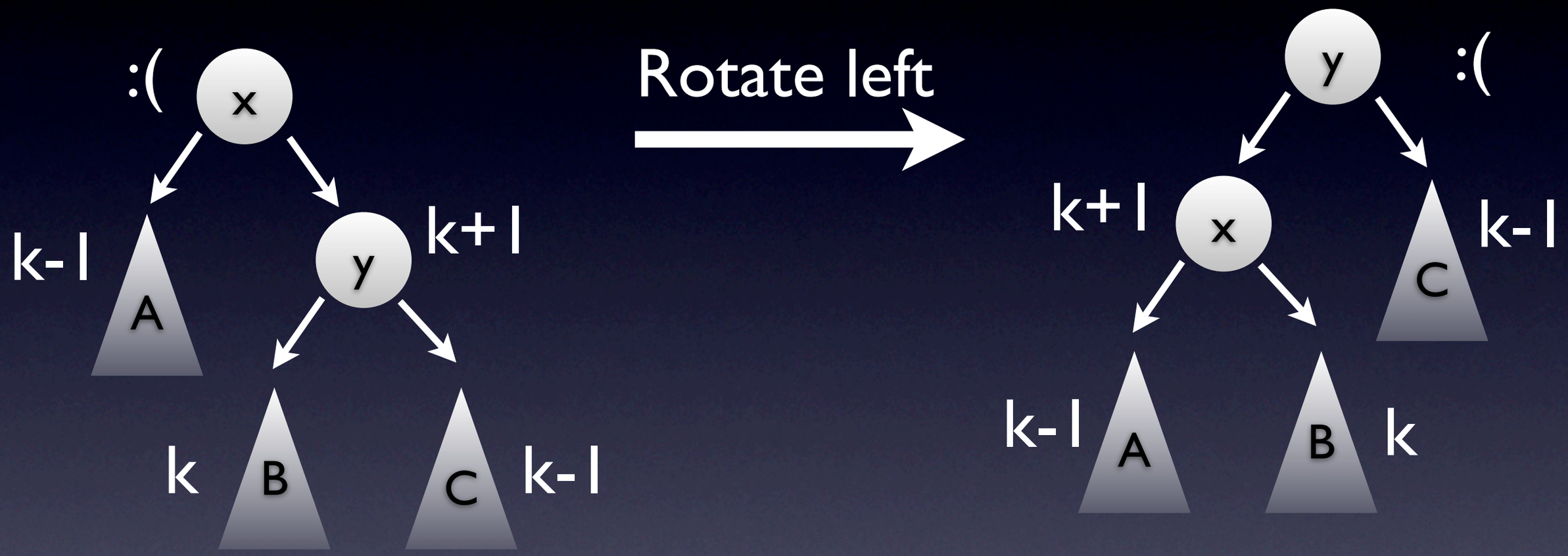
Rebalancing: easy?



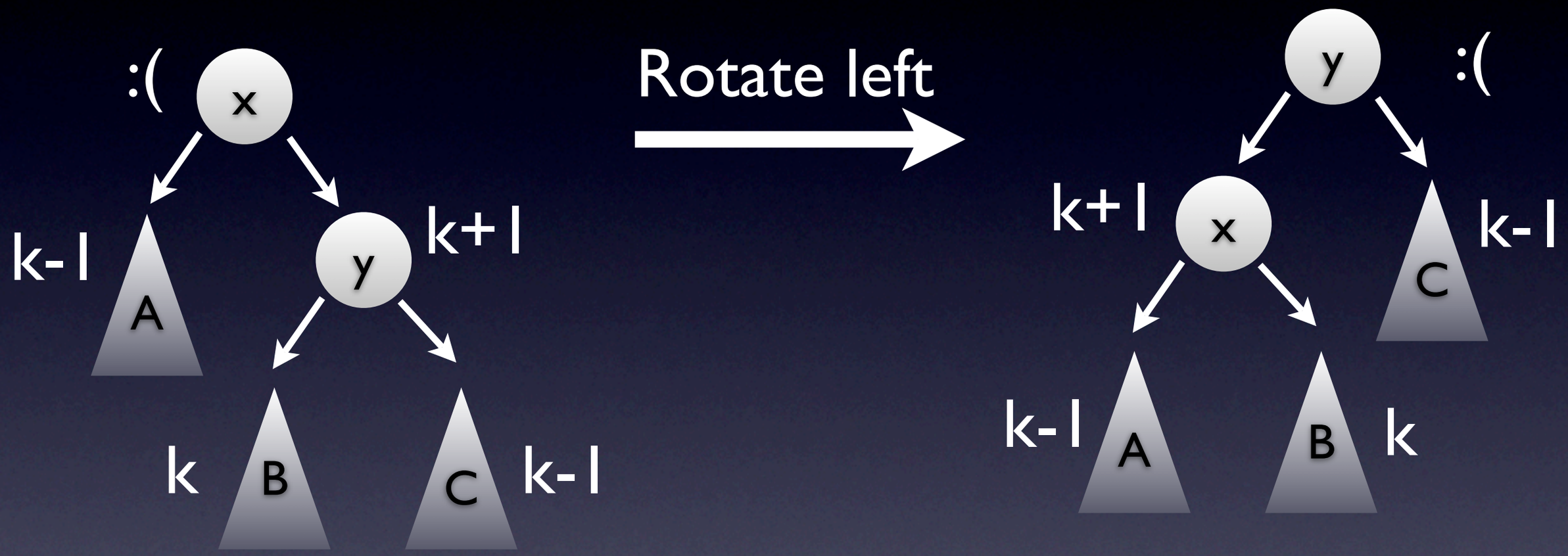
Rebalancing: easy?



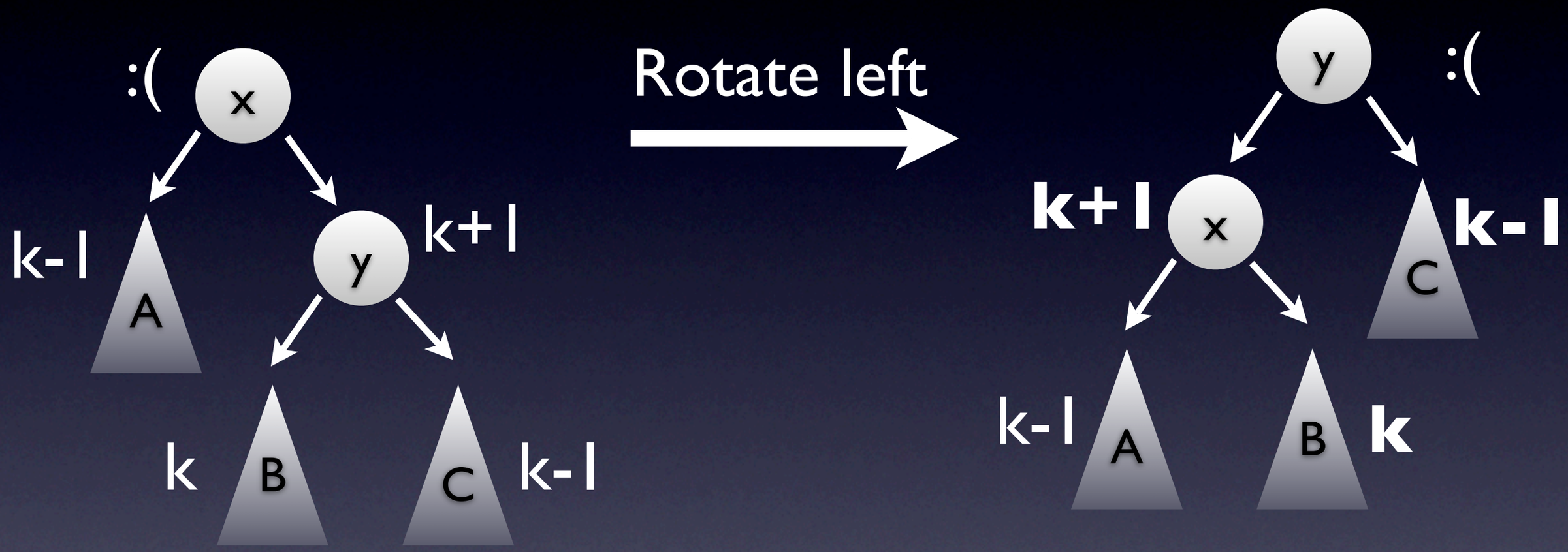
Rebalancing: easy?



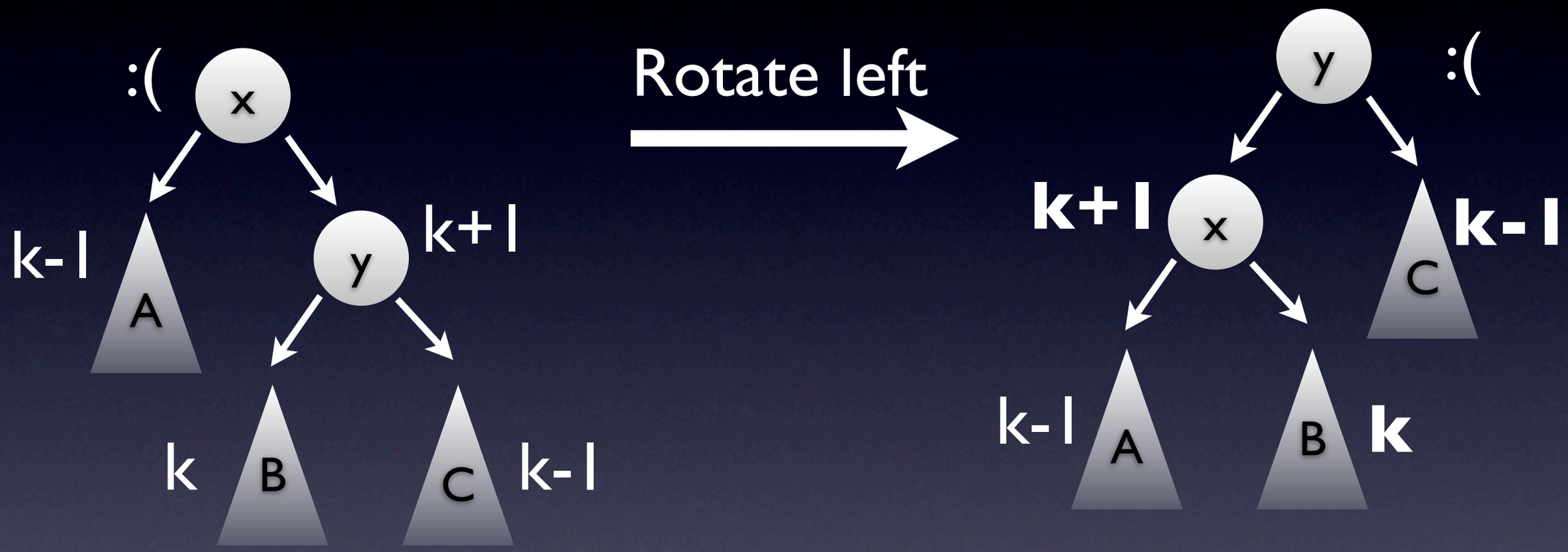
WTF?



WTF?

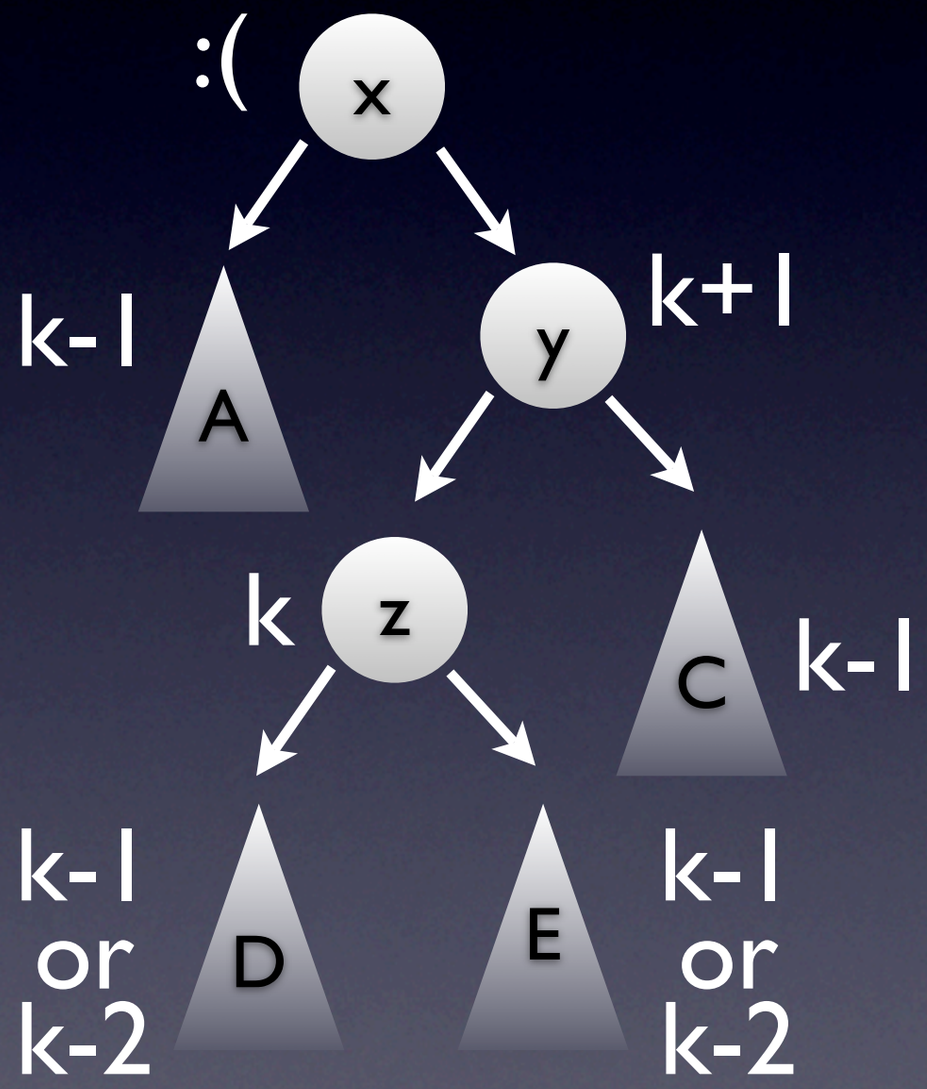


WTF?

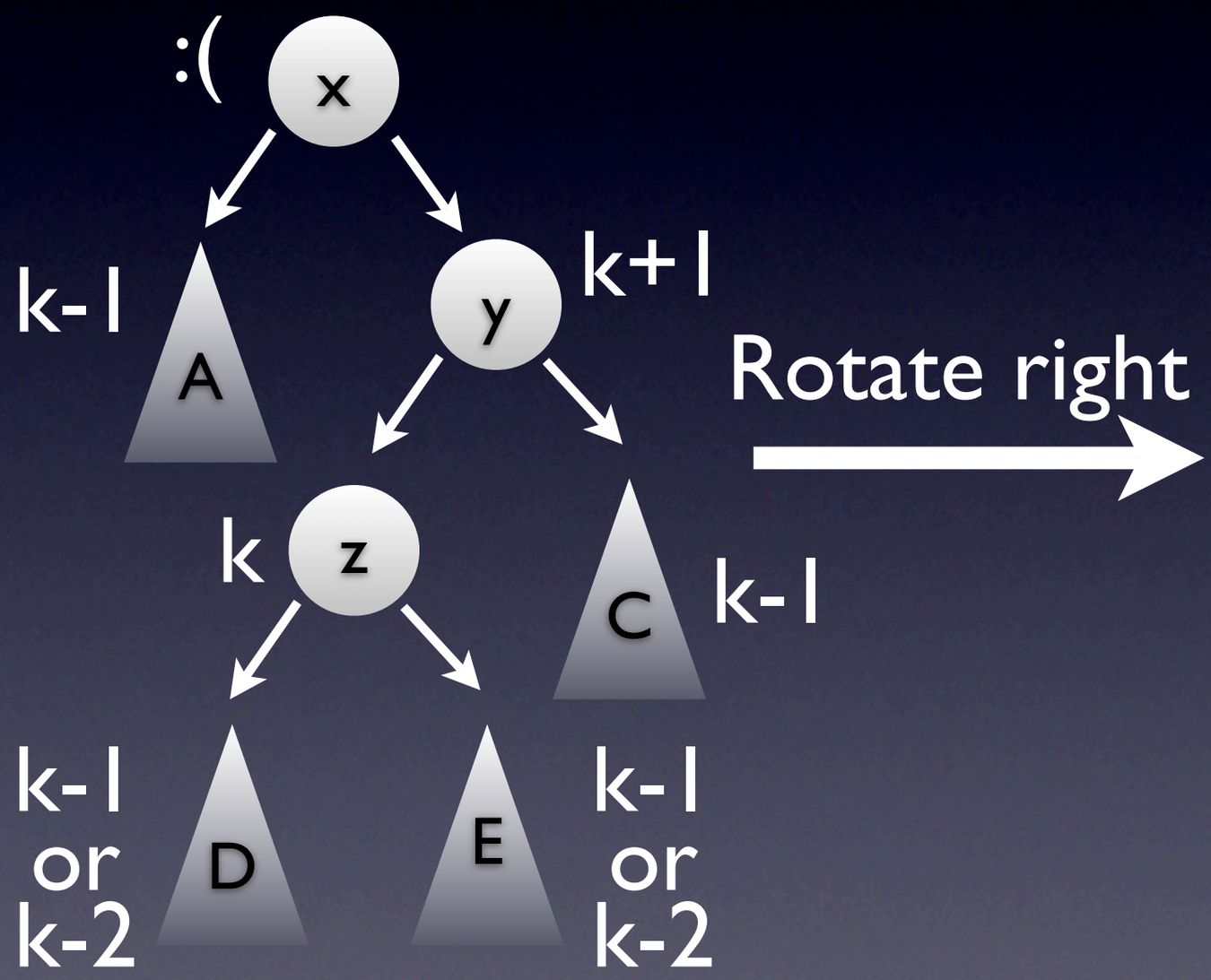


B cannot be taller than C

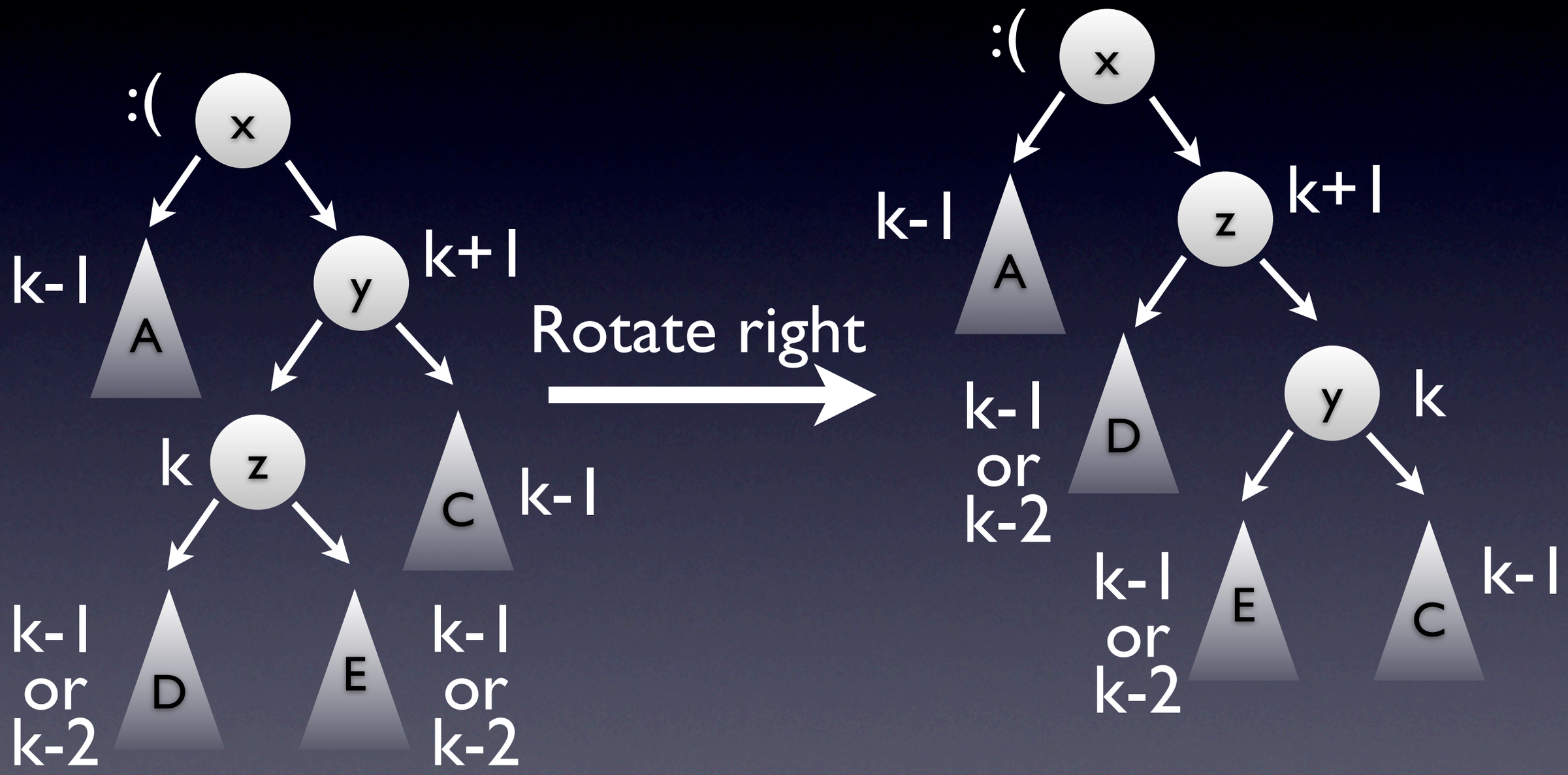
Rebalancing: hack it up



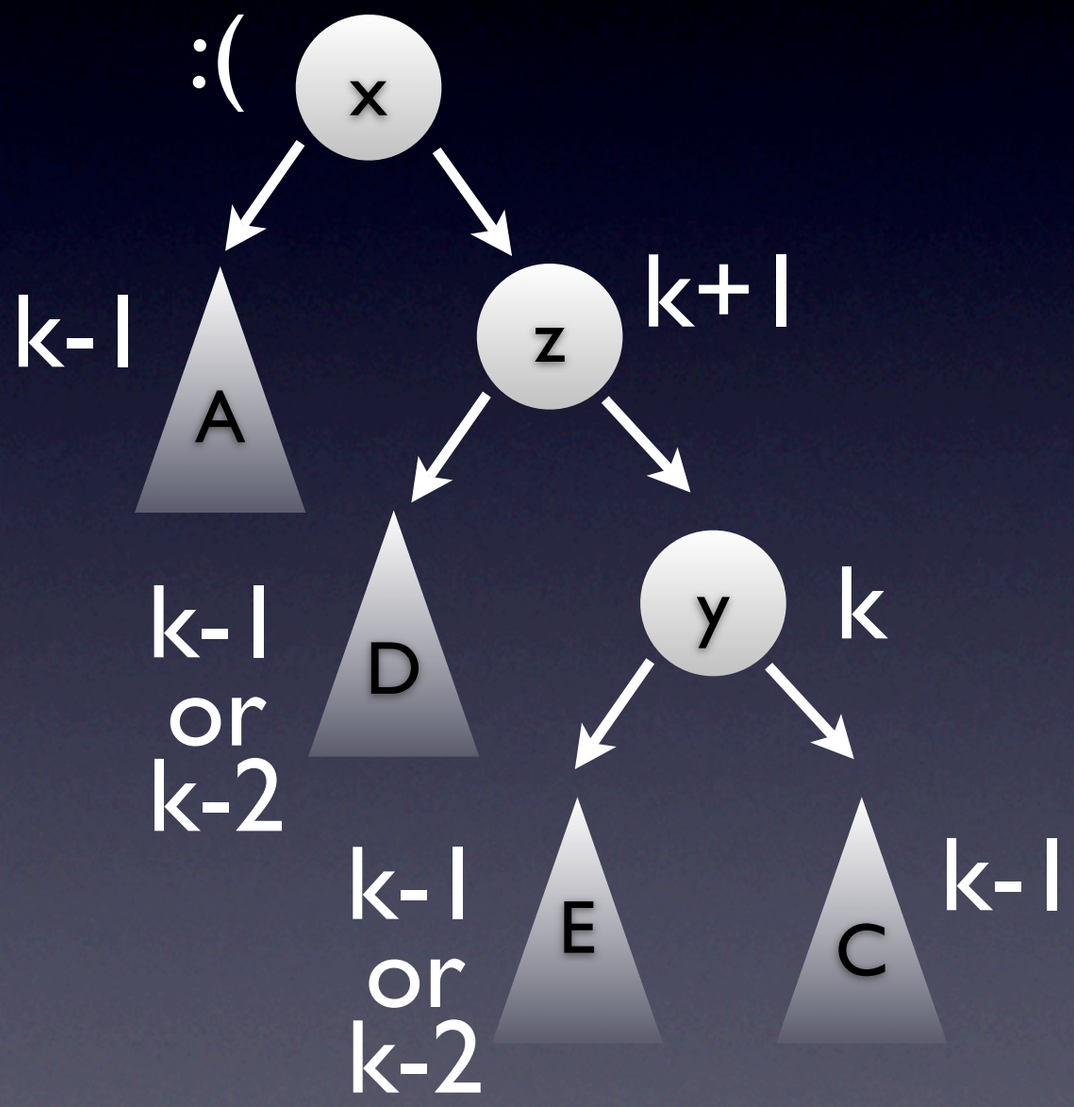
Rebalancing: hack it up



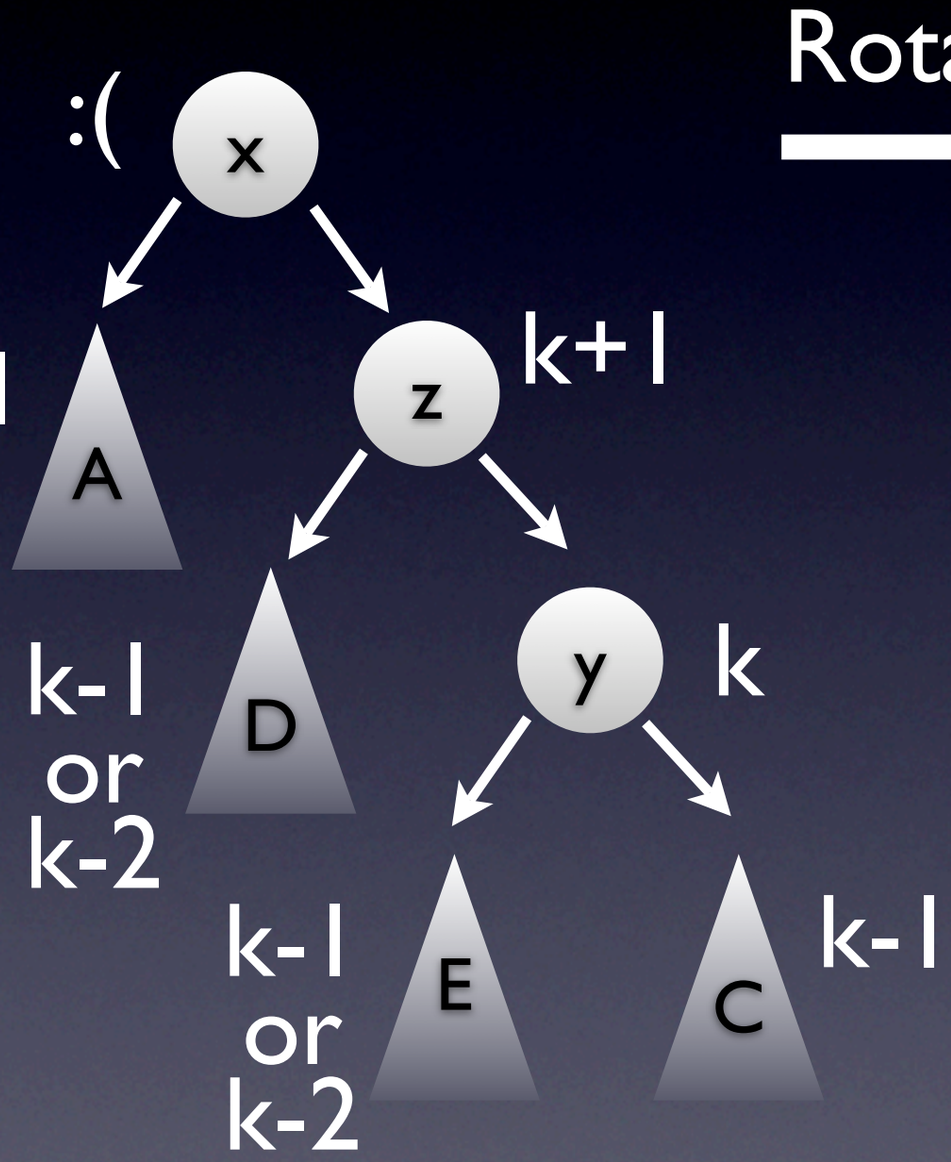
Rebalancing: hack it up



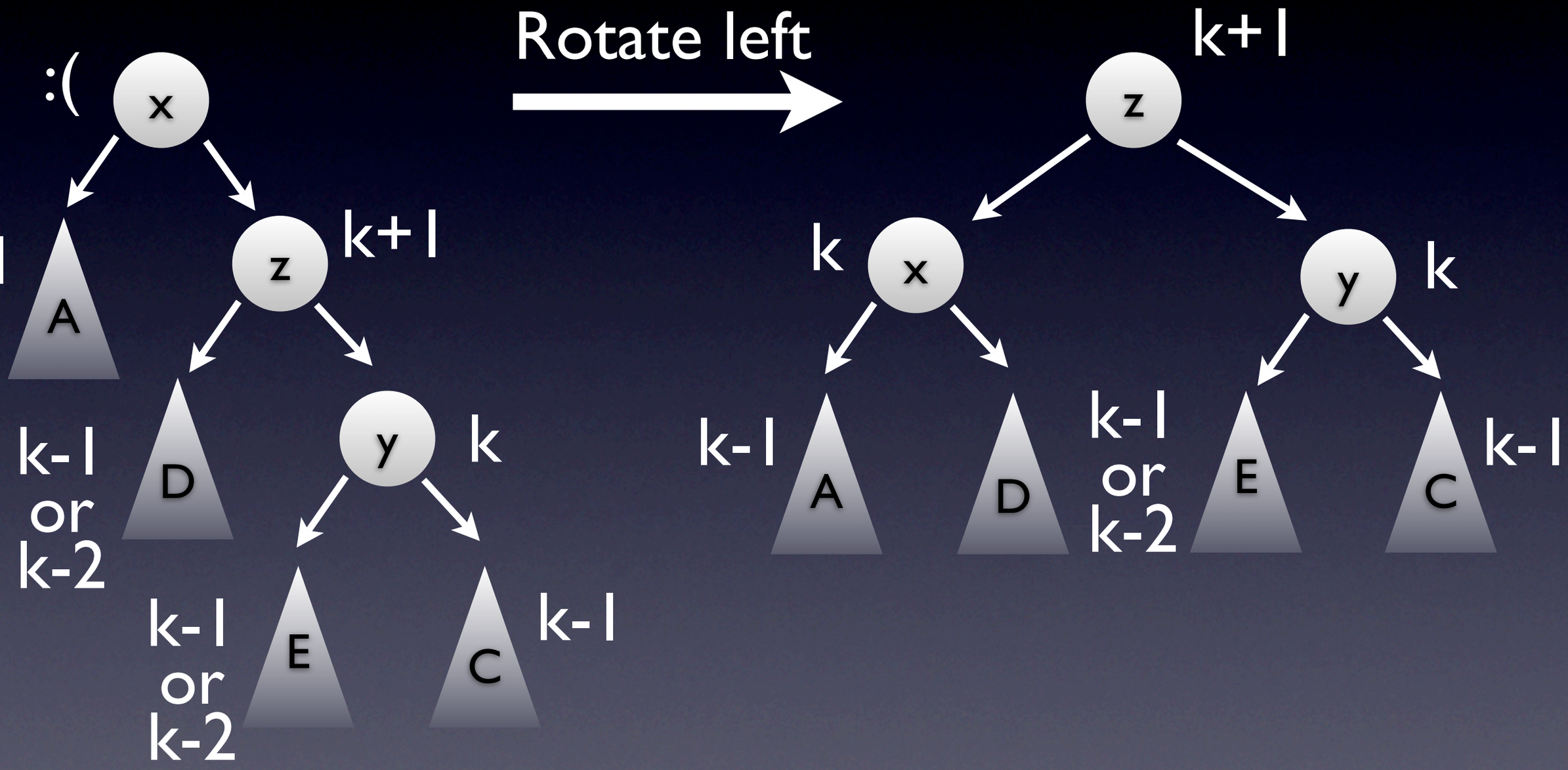
and in the end it's right



and in the end it's right



and in the end it's right



Rebalancing one level

- AVL violation at current node
- Right is than left?
  - Right.left taller than right.right?
    - Rotate right to the right
  - Either way, rotate current to the left
- Left is heavier than right: symmetry

Rebalancing wrap-up

- Know how to fix one level, use that to fix everything along the path to the root
- Must recompute height on-the-go
  - If recomputing for all nodes along the path on each rotation, $O(\log^2(h))$
- Why is rebalancing $O(\log(h))$?

Python Code

‘cause you can’t live on bubbles and lines

AVL Design

- BST
  - incorporate the deletion hack
- AVL
  - inherited from BST, uses AVLnode
- AVLnode
  - does all the heavy lifting

Return values matter!

- insert: returns the newly inserted node
- delete: returns the deleted node (its parent link still indicates where it was hanging)

BST, take 2

```
1 class BST(object):
2     def __init__(self, NodeType=BSTnode):
3         self.root = None
4         self.NodeType = NodeType
5         self.psroot = self.NodeType(None, None)
6
7     def reroot(self):
8         self.root = self.psroot.left
9
10    def insert(self, t):
11        if self.root is None:
12            self.psroot.left = self.NodeType(self.psroot, t)
13            self.reroot()
14            return self.root
15        else:
16            return self.root.insert(t, self.NodeType)
```

AVL

```
1 class AVL(BST):
2     def __init__(self):
3         BST.__init__(self, AVLnode)
4
5     def insert(self, t):
6         node = BST.insert(self, t)
7         node.rebalance()
8         self.reroot()
9
10    def delete(self):
11        node = BST.delete(self)
12        node.parent.rebalance()
13        self.reroot()
```

AVLnode: helpers

```
1 def height(node):
2     if node is None:
3         return -1
4     else:
5         return node.height
6
7 class AVLnode(BSTnode):
8     def update_stats(self):
9         self.height = max(height(self.left), height(self.right)) + 1
10        BSTnode.update_stats(self)
```

AVLnode: rotation

```
1 class AVLnode(BSTnode):
2     def left_rotate(self):
3         x = self; y = x.right
4         y.parent = x.parent
5         if y.parent.left is x:
6             y.parent.left = y
7         elif y.parent.right is x:
8             y.parent.right = y
9         x.right = y.left
10        if x.right is not None:
11            x.right.parent = x
12        y.left = x
13        x.parent = y
14        x.update_stats()
15        y.update_stats()
16        return y
```

AVLnode: rebalancing

```
1 class AVLnode(BSTnode):
2     def rebalance(self):
3         if self.key is None: return
4 
5         self.update_height()
6         if height(self.left) >= 2 + height(self.right):
7             if height(self.left.left) < height(self.left.right):
8                 self.left.left_rotate()
9             self.right_rotate()
10        elif height(self.right) >= 2 + height(self.left):
11            if height(self.right.right) < height(self.right.left):
12                self.right.right_rotate()
13            self.left_rotate()
14        self.parent.rebalance()
```

And we're done!

- costan@mit.edu
- (617) 230-9694, no voicemail
- AIM: victorcostan
- Google Talk: costan@gmail.com
- 32G-8th Floor

v. Next

- Don’t need to reroot(), can just use root() as a function that returns psroot.left
- Put the [1, 2... 8] exercise on the slides so it’s remembered
- Quote relevant CLRS chapters on the wiki