

# CS350: Data Structures

## Splay Trees

---

James Moscola

Department of Engineering & Computer Science  
York College of Pennsylvania



# Splay Trees

---

- Another variation of the binary tree
- Does **NOT** guarantee  $O(\log N)$  worst case operations
- Splay tree may exhibit  $O(N)$  worst case for any single operation
  - Time for operations is amortized
  - On average, splay tree operations will exhibit  $O(\log N)$  worst case time
- Does not require any accounting information (i.e. color, level, height, etc.)
- Key Feature: Frequently accessed keys are moved up so that they are near the root
  - 90/10 rule - 90% of data access in data structure are to only 10% of the data

# Splay Operation

---

- **Operation used to rearrange tree to make a specified node the root node of the tree**
  - Has a nice side-effect that any imbalances in the tree are corrected
- **Splay operation is called whenever a node is accessed**
  - After a node is inserted, a splay operation is called on that node
    - A node inserted into the tree will always end up as the root when the insertion completes
  - After searching for a node, a splay operation is called on it
    - Subsequent searches for that node should execute more quickly
  - To delete a node, first splay it to the top of the tree, then delete it

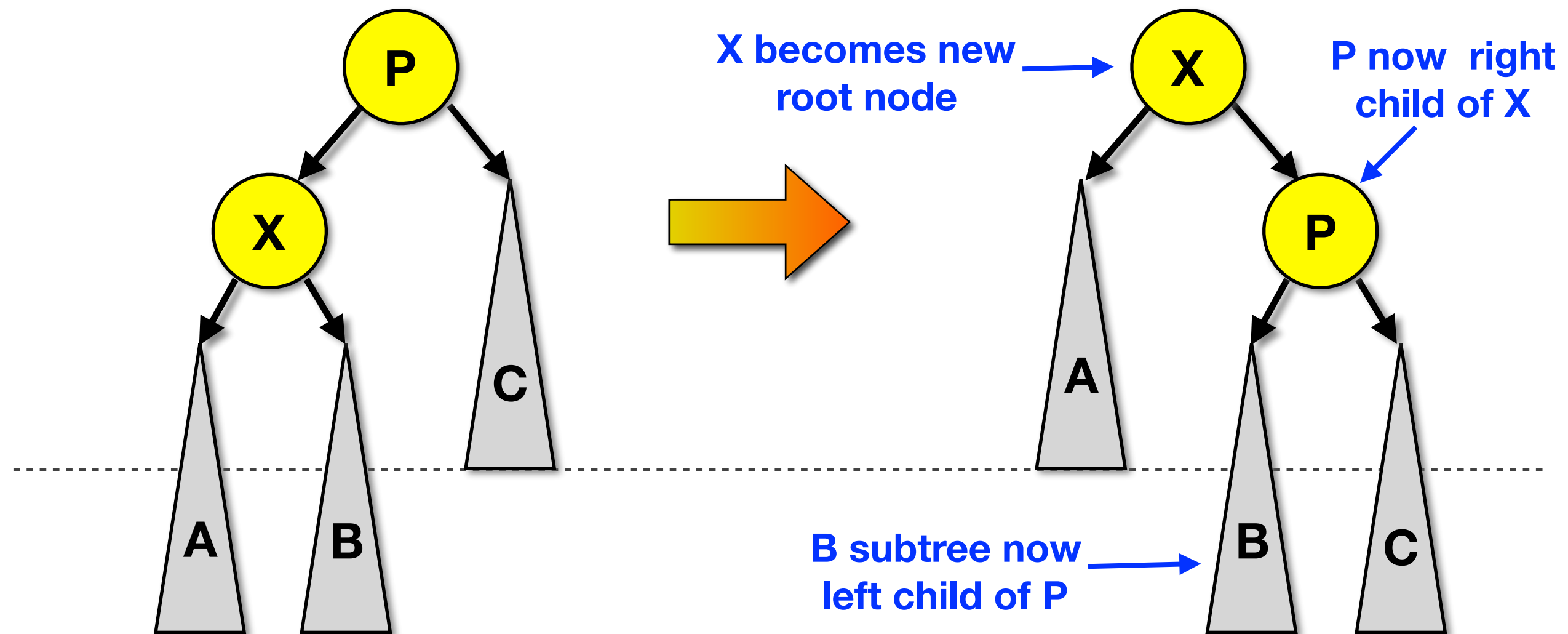
# Splay Operation

---

- **When splay is called on a node, continue to splay that node until it becomes the root of the entire tree**
- **There are three cases to consider when splaying a node**
  - Case #1 - a zig - a node X is the left child of a node P
  - Case #2 - a zig-zag - a node X is the inside grandchild of a node G
  - Case #3 - a zig-zig - a node X is the outside grandchild of a node G

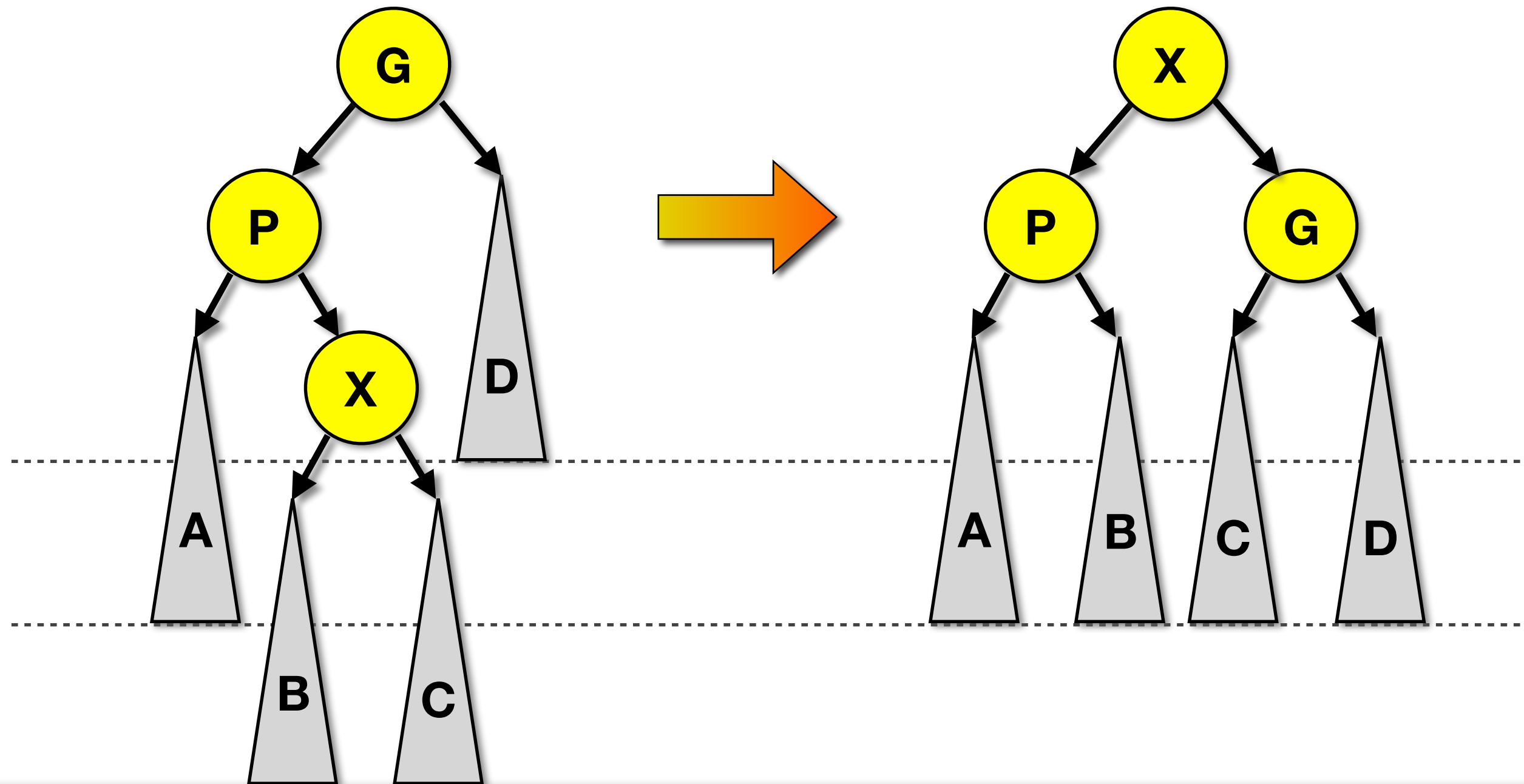
# Splay Operation -- Case #1 (a Zig)

Splaying a node X whose parent P is the root of the subtree.  
Same as a single rotation of the node X with the root P.



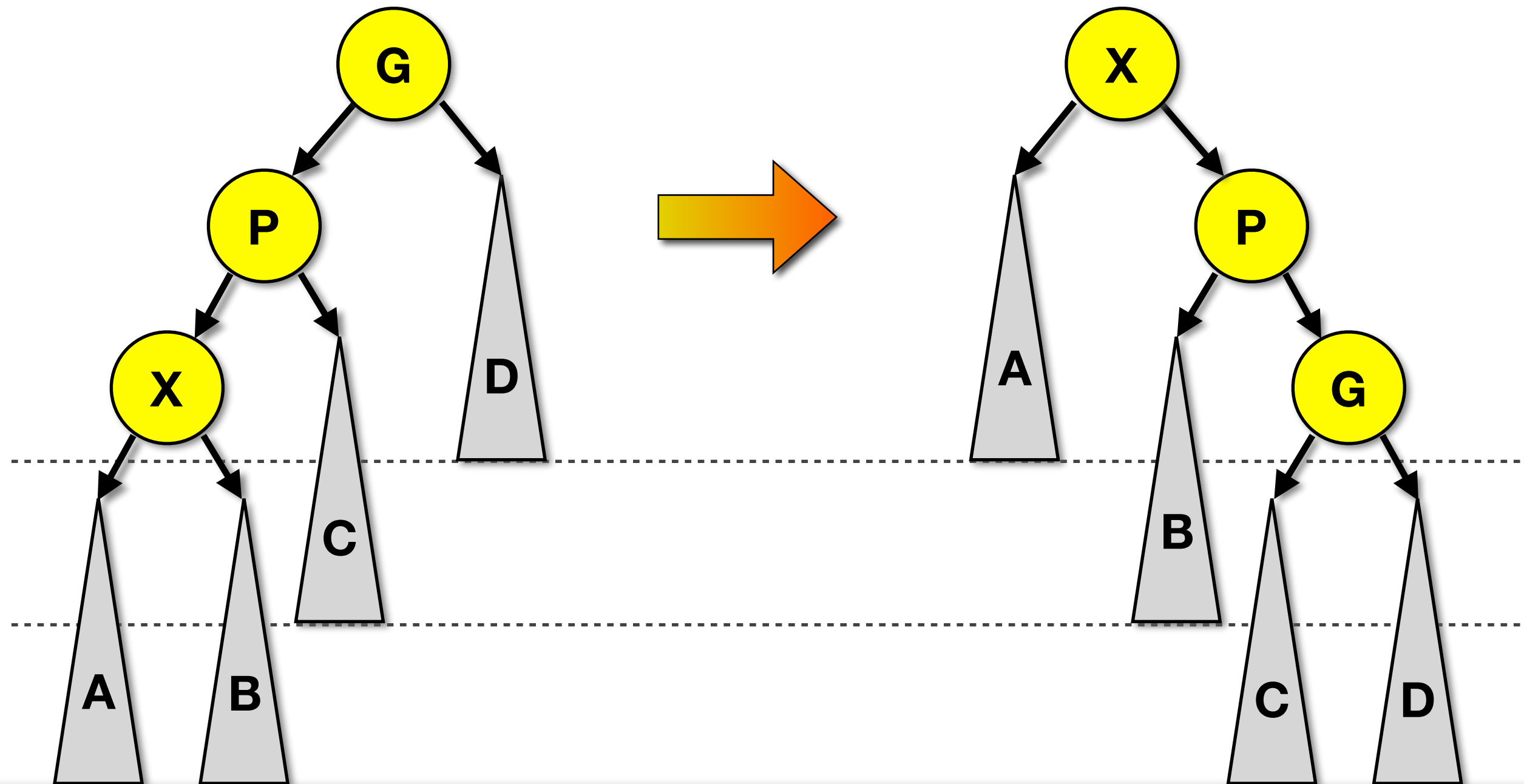
# Splay Operation -- Case #2 (a Zig-Zag)

Splaying a node X which is the inside grandchild of G.  
Same as a double rotation of the node X.



# Splay Operation -- Case #3 (a Zig-Zig)

Splaying a node X which is the outside grandchild of G.  
A single rotation between P and G followed by single rotation between X and P.



# Splay Tree Operations

---

- **The Splay tree supports same operations as other binary search trees**
  - **Find** - starts with the standard BST Find algorithm; then performs the splay operation on the node after it is found to bring it up to the root of the tree
    - If node is not found, then the splay operation is performed on the last node in the tree that was visited while performing the search
  - **Insert** - executes a standard BST Insert algorithm; then performs the splay operation on the newly inserted node
    - A node inserted into the tree will always end up as the root when the insertion completes



# Splay Tree Operations (Cont.)

---

- **The Splay tree supports same operations as other binary search trees**

- **Delete**

- (1) Performs a Find to locate the node to be deleted, this also brings the node to be deleted to the root of the tree;
- (2) Delete the root of the tree leaving two unattached subtrees
- (3) Perform a FindMax on the left subtree to find the maximum node and splay that maximum node to the root of the left subtree (it will not have a right child, even after the splay)
- (4) The maximum node from step 3 becomes the new root node for the entire tree; attach the unattached right subtree from step 2 as a right child of this new root node

# Insertion Example

---

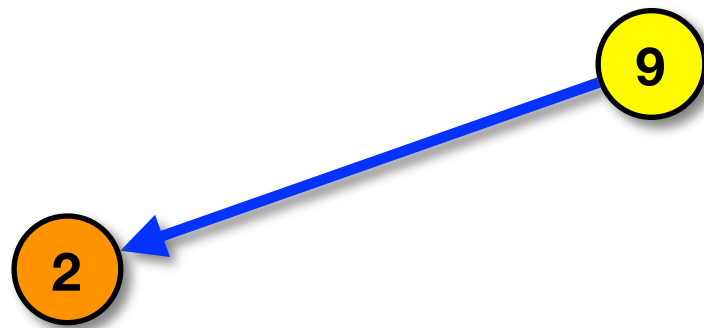
9

**Insert node 9**

**Calling splay on node 9 doesn't do anything**

# Insertion Example

---

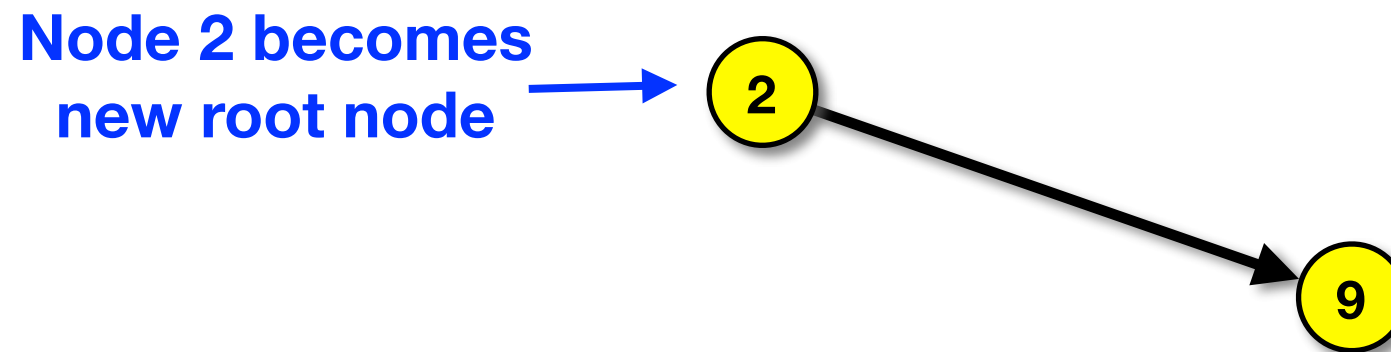


**Insert node 2**

**Case #1 - perform a zig operation**

# Insertion Example

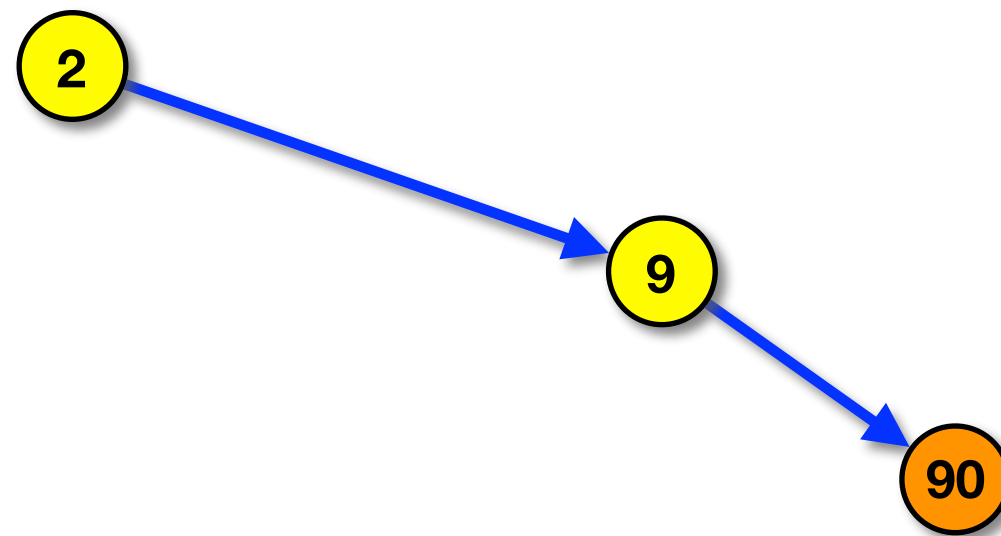
---



**Insert node 2**  
**Result of zig operation**

# Insertion Example

---

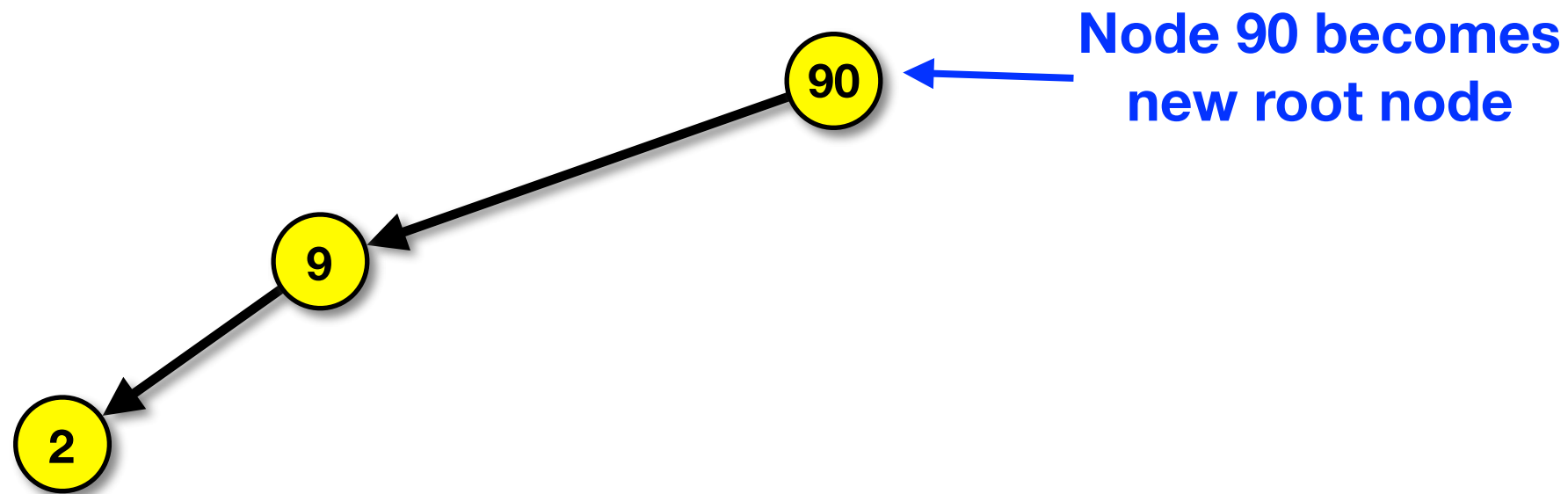


**Insert node 90**

**Case #3 - perform a zig-zig operation**

# Insertion Example

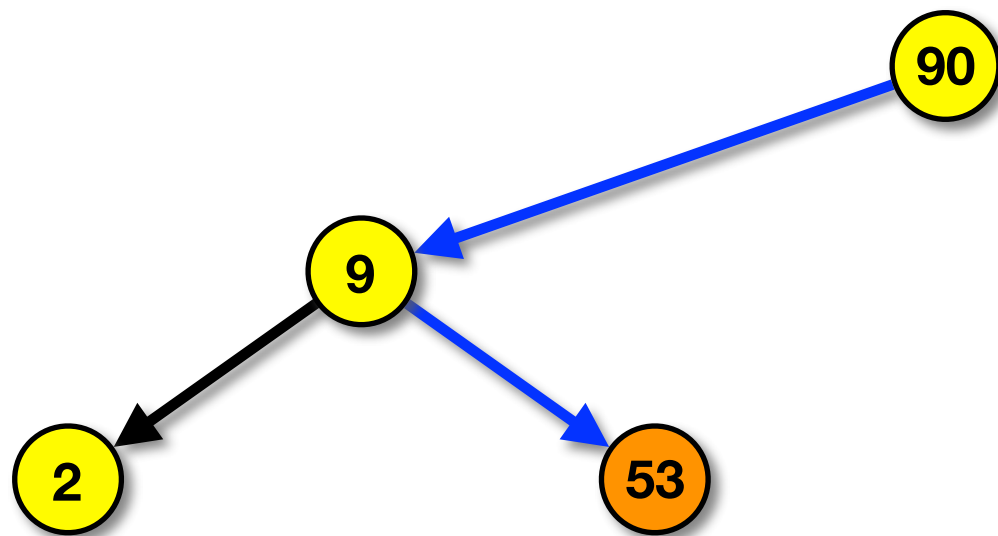
---



**Insert node 90**  
**Result of zig-zig operation**

# Insertion Example

---

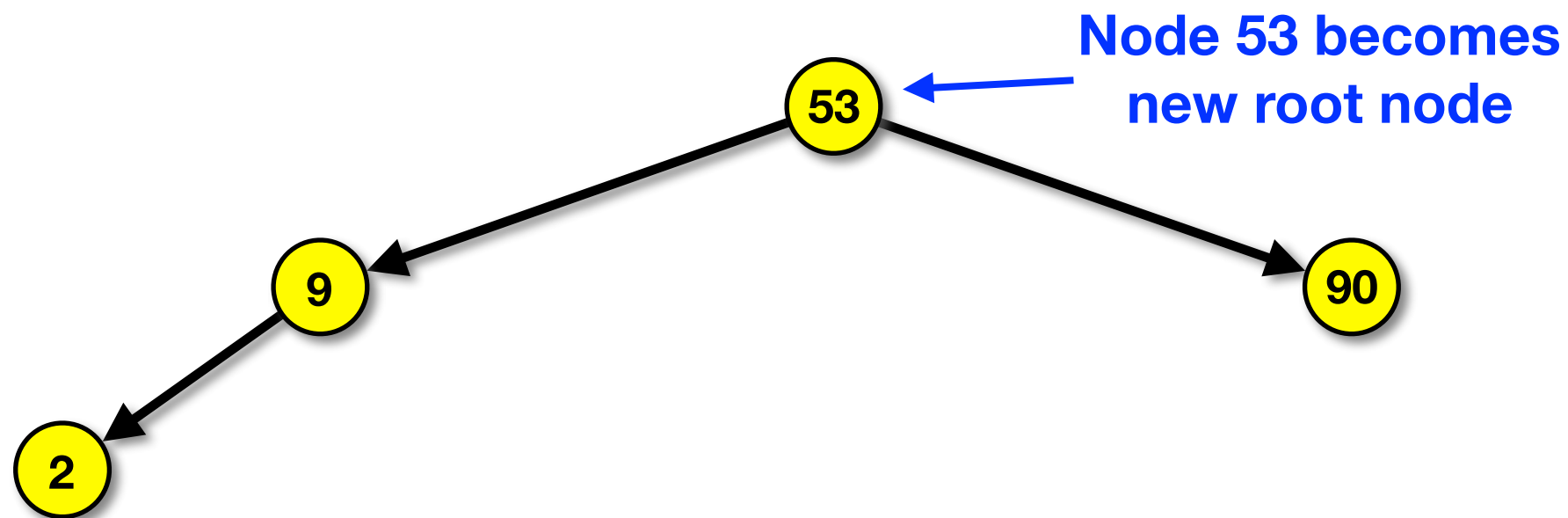


**Insert node 53**

**Case #2 - perform a zig-zag operation**

# Insertion Example

---

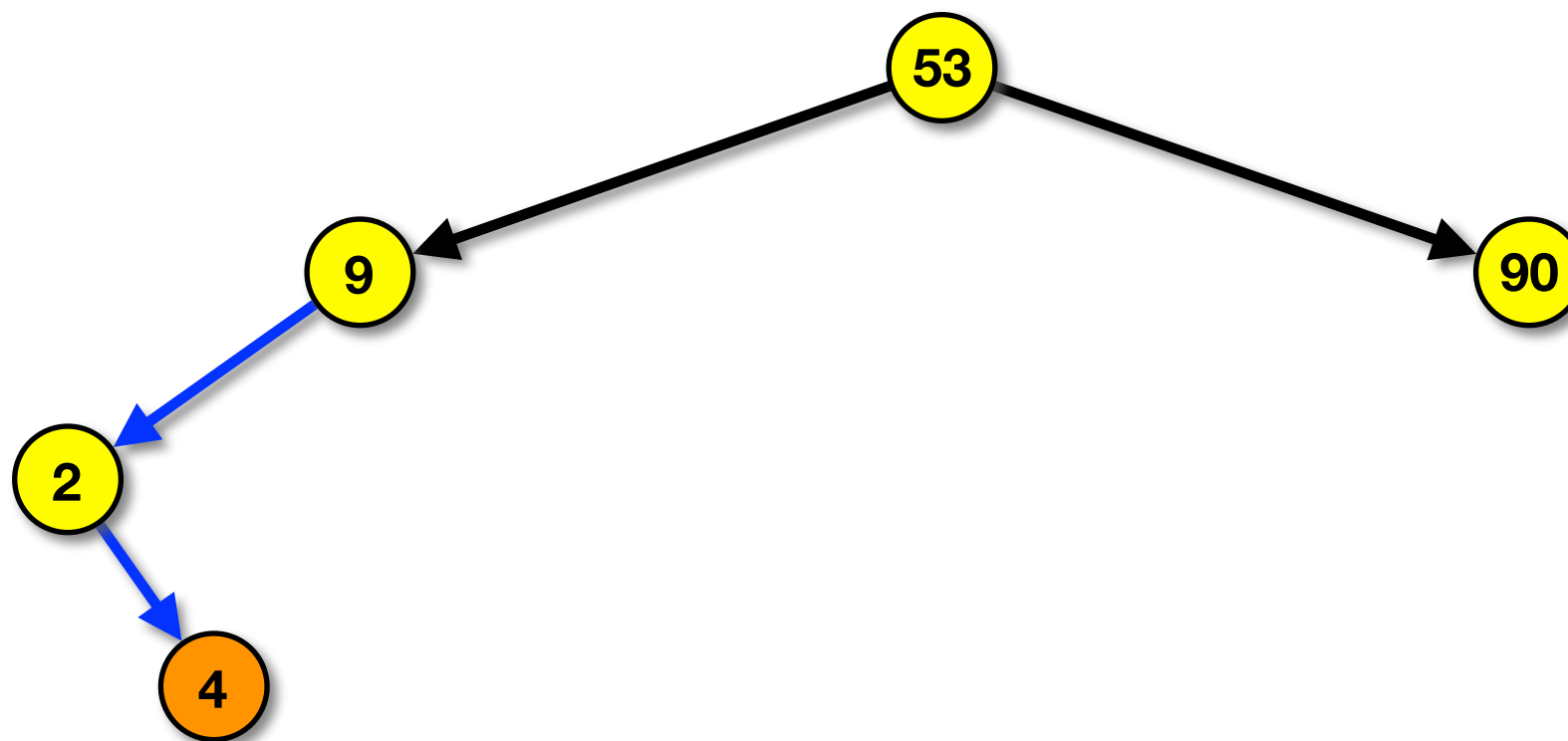


**Insert node 53**  
**Result of zig-zag operation**



# Insertion Example

---



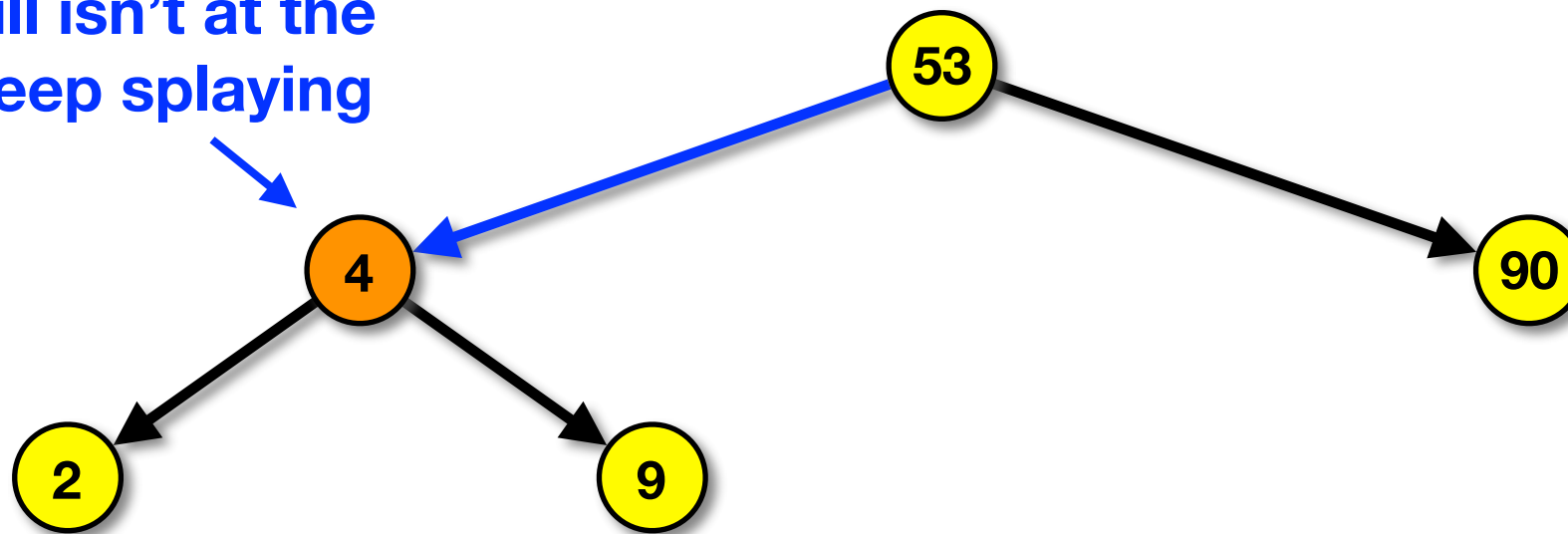
**Insert node 4**

**Case #2 - perform a zig-zag operation**

# Insertion Example

---

Node 4 still isn't at the root, so keep splaying

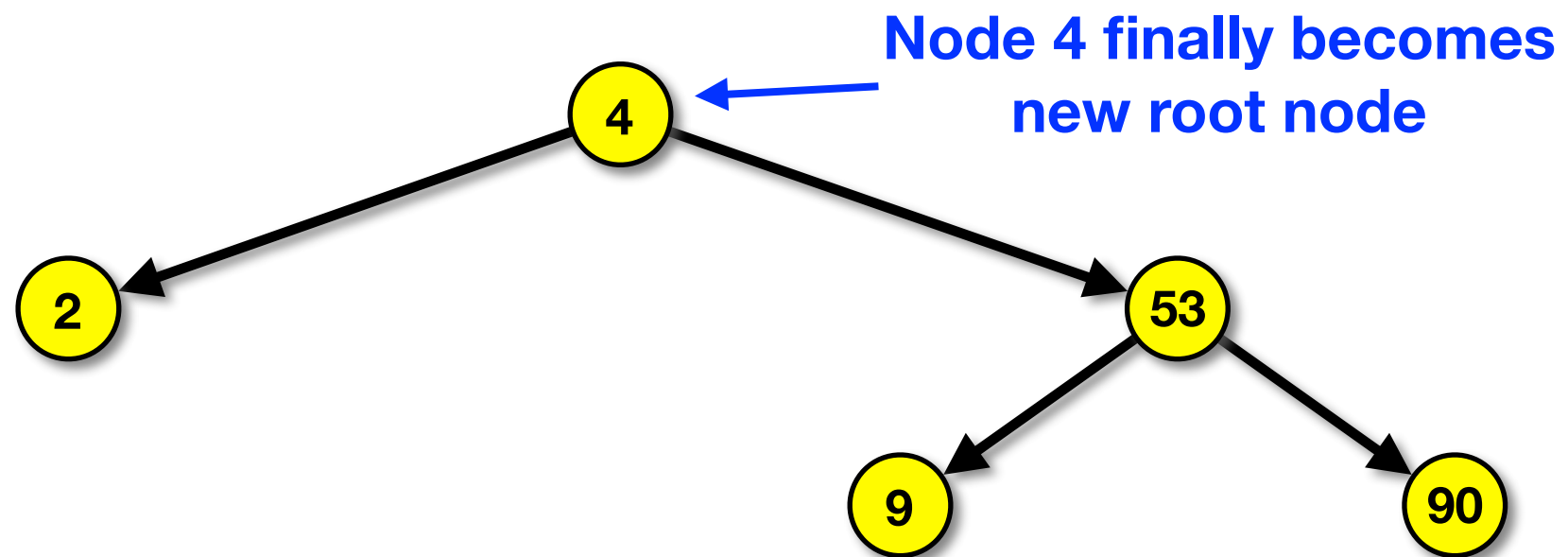


**Insert node 4 (Cont.)**

**Case #1 - perform a zig operation**

# Insertion Example

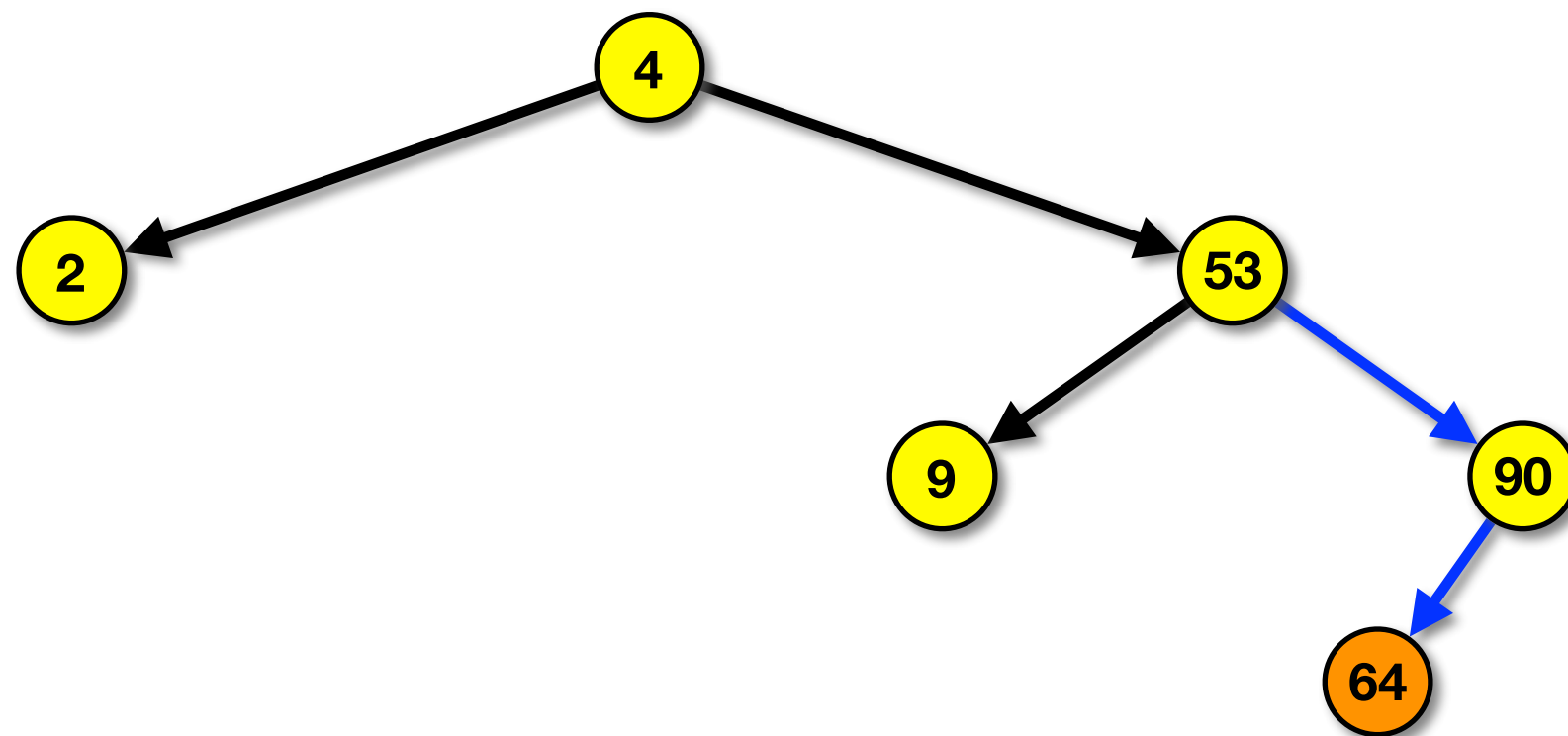
---



**Insert node 4**  
**Result of zig-zag operation**  
**followed by a zig operation**

# Insertion Example

---

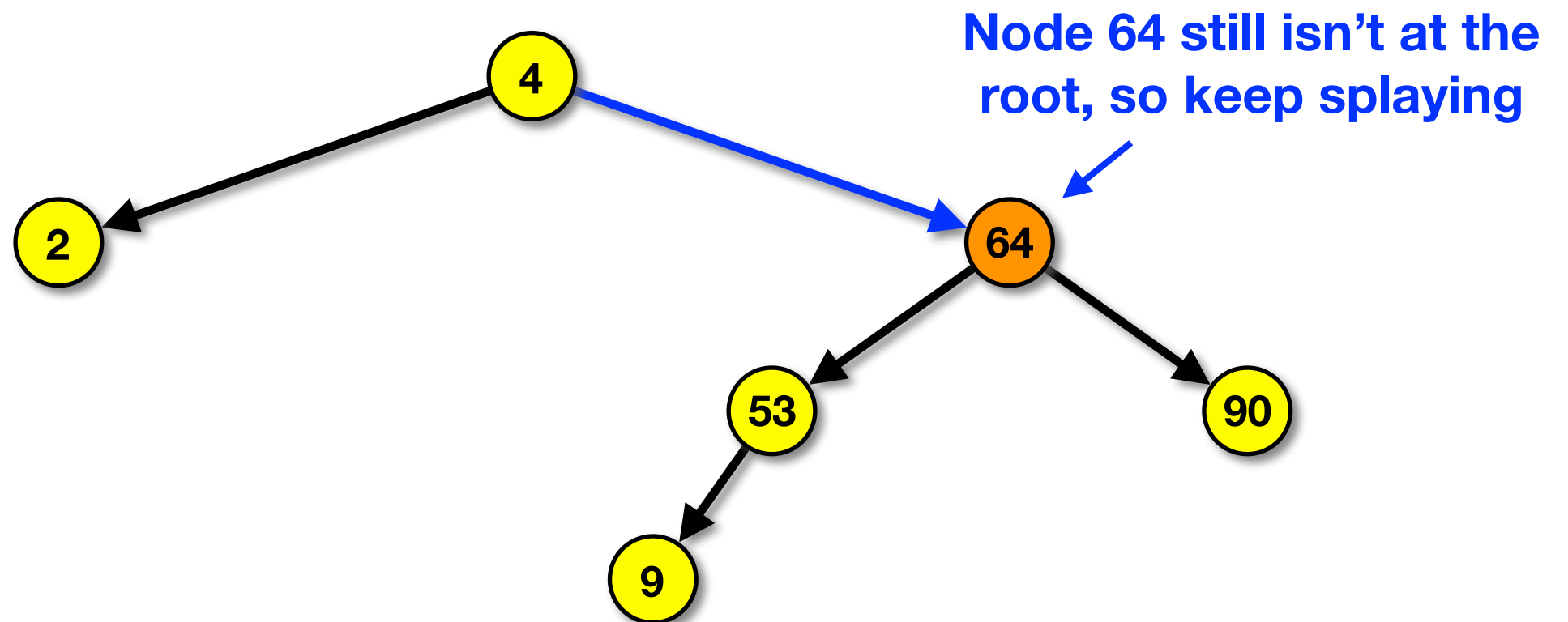


**Insert node 64**

**Case #2 - perform a zig-zag operation**

# Insertion Example

---

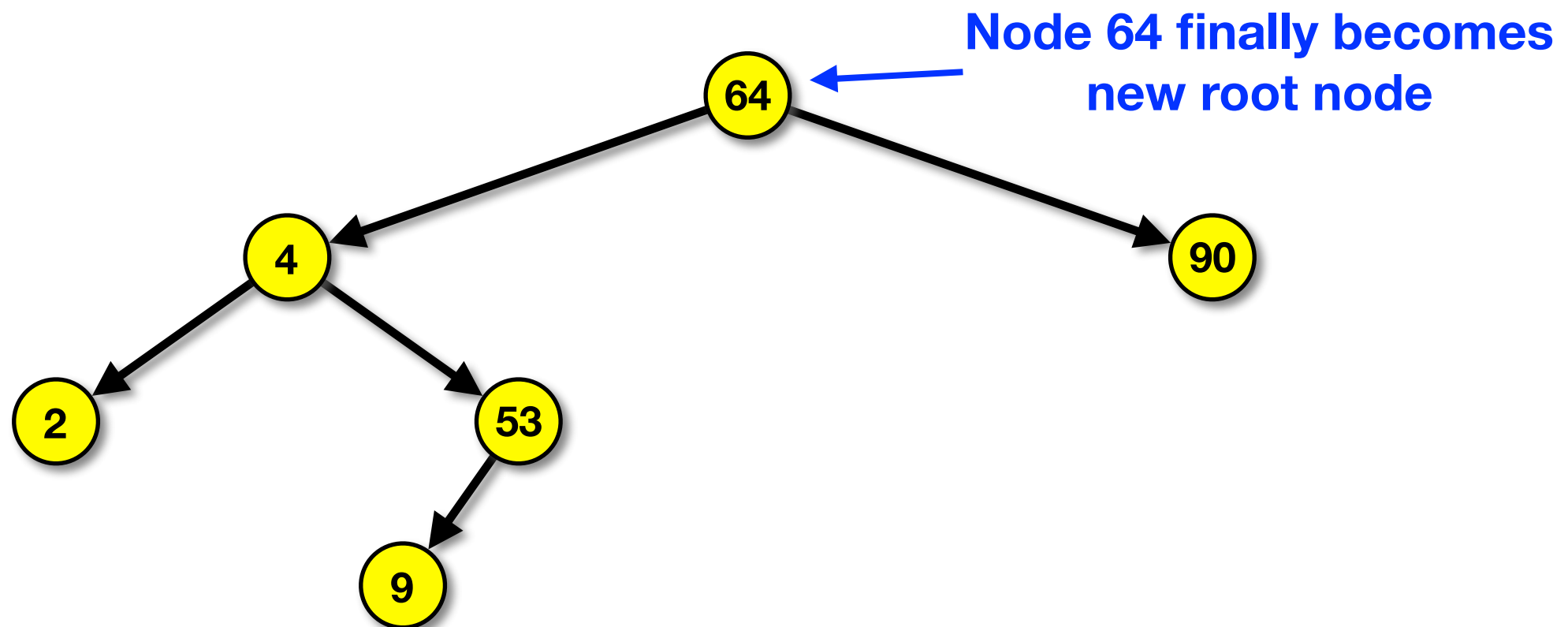


**Insert node 4 (Cont.)**

**Case #1 - perform a zig operation**

# Insertion Example

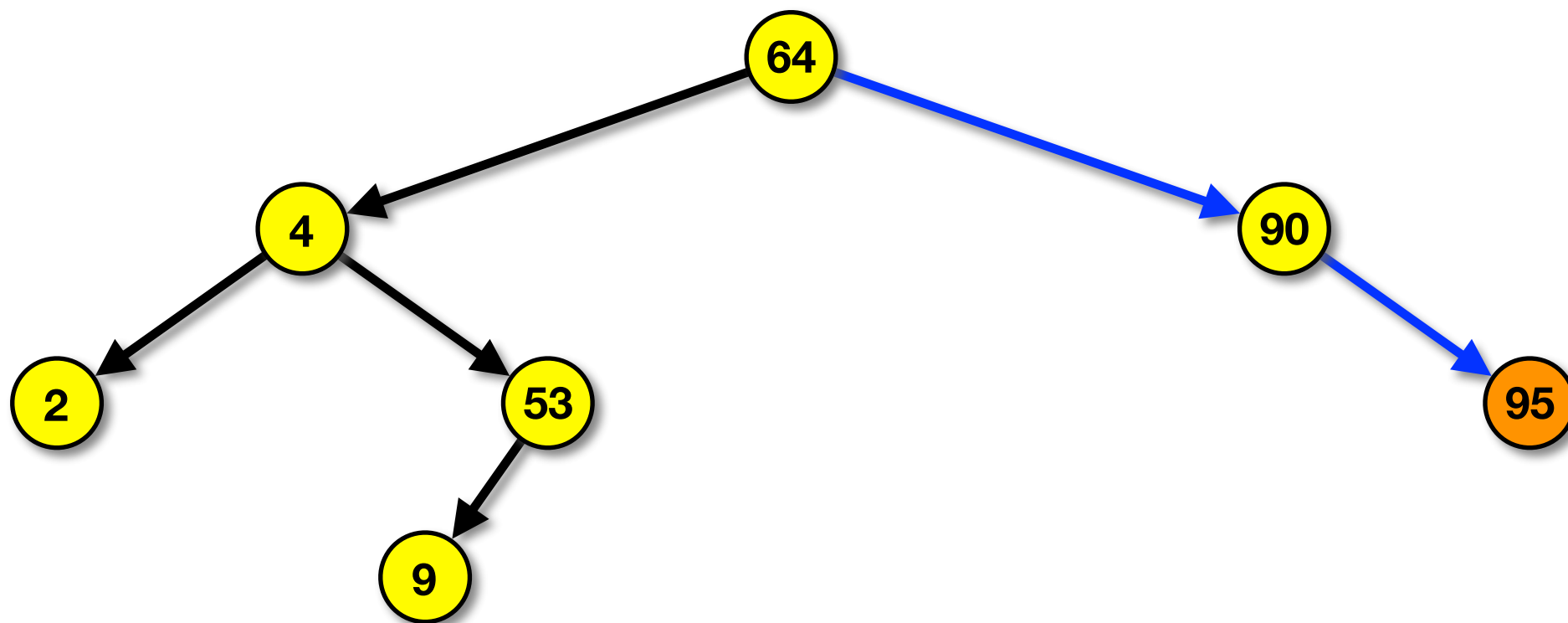
---



**Insert node 64**  
**Result of zig-zag operation**  
**followed by a zig operation**

# Insertion Example

---

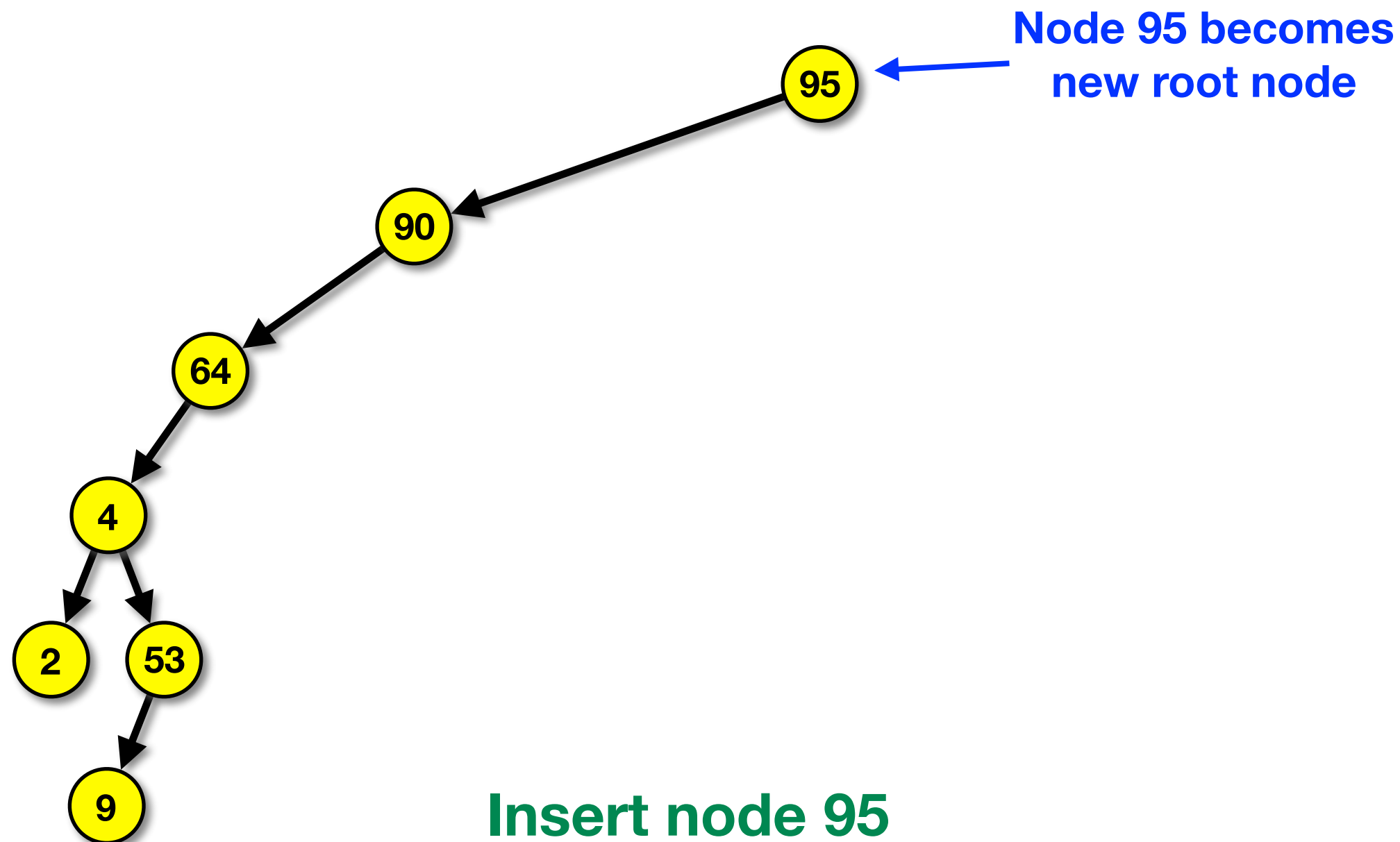


**Insert node 95**

**Case #3 - perform a zig-zig operation**

# Insertion Example

---

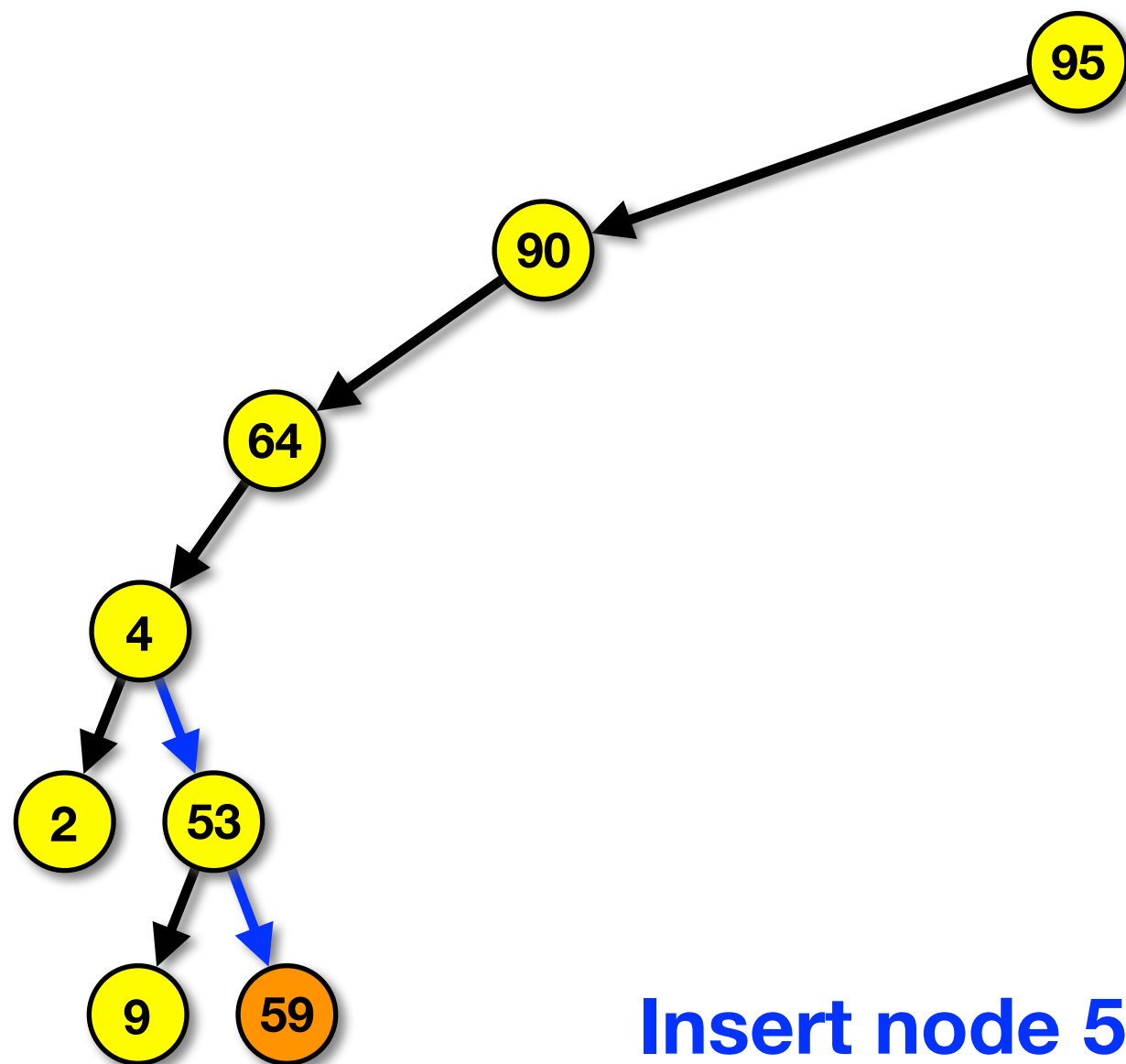


**Insert node 95**  
**Result of zig-zig operation**



# Insertion Example

---

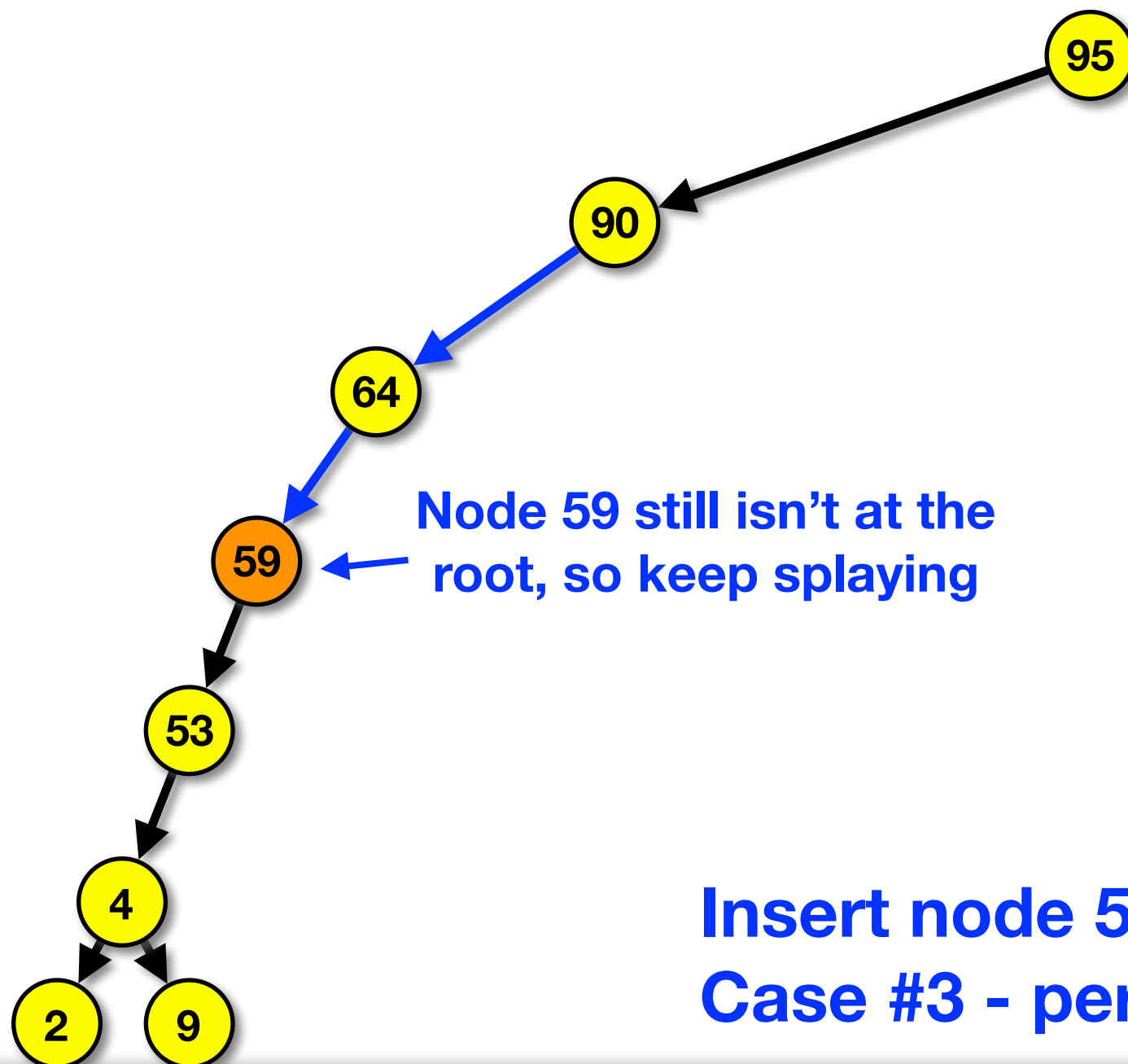


**Insert node 59**

**Case #3 - perform a zig-zig operation**

# Insertion Example

---

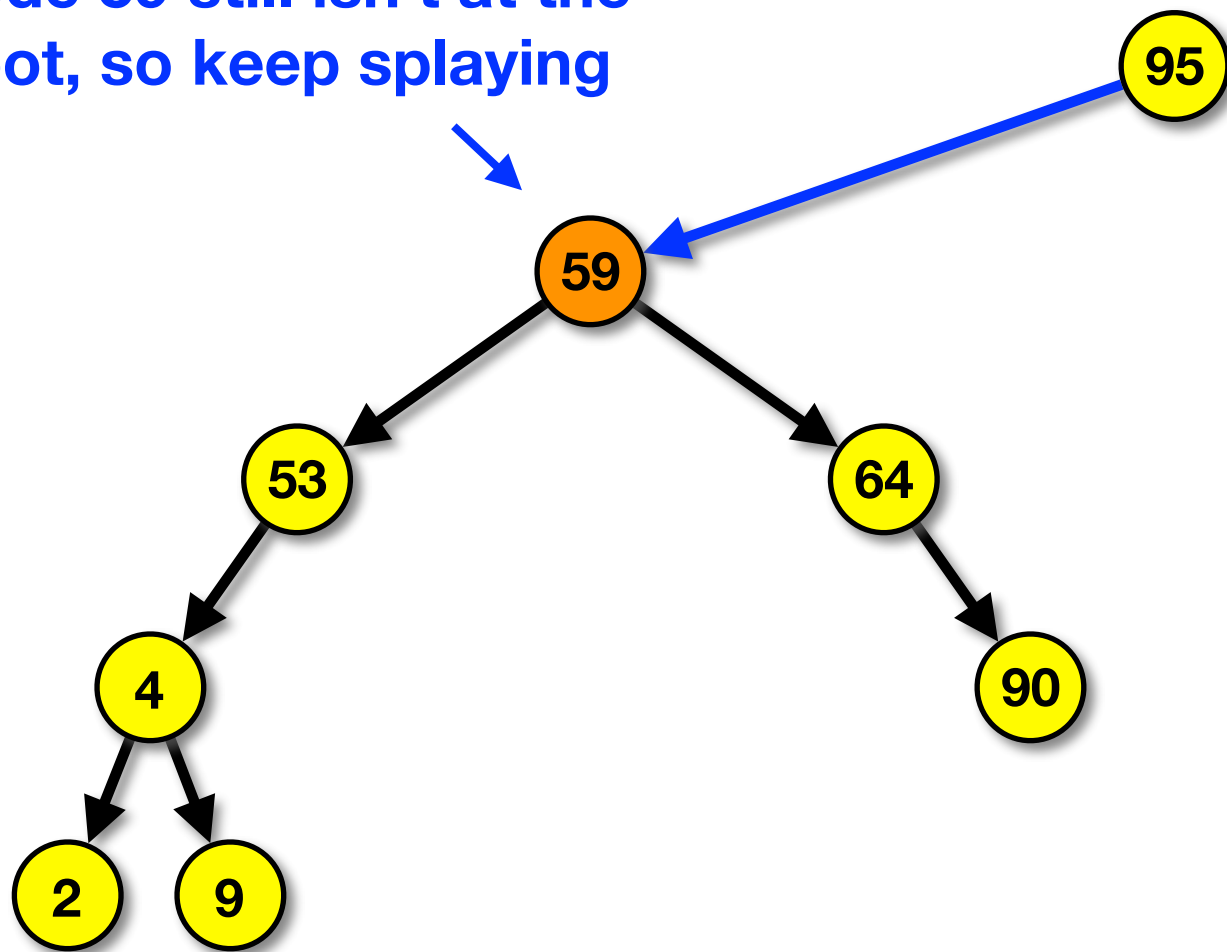


**Insert node 59 (Cont.)**  
**Case #3 - perform a zig-zig operation**

# Insertion Example

---

Node 59 still isn't at the root, so keep splaying

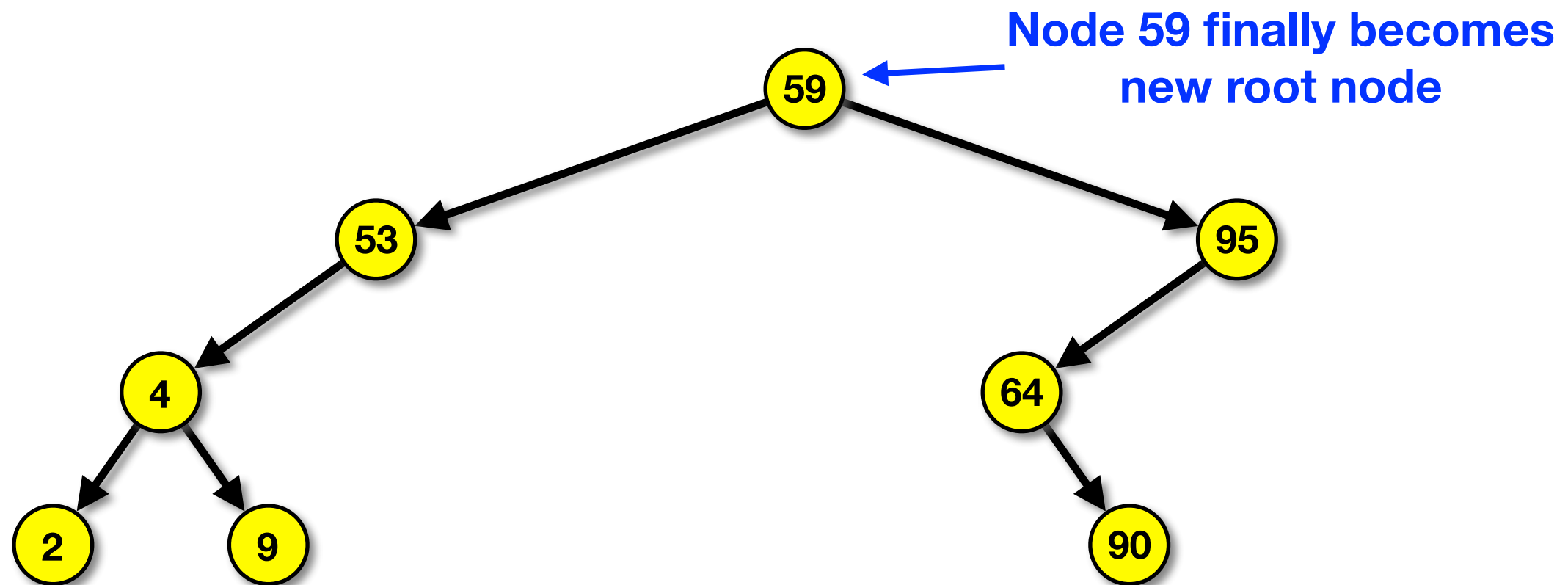


**Insert node 59 (Cont.)**

**Case #1 - perform a zig operation**

# Insertion Example

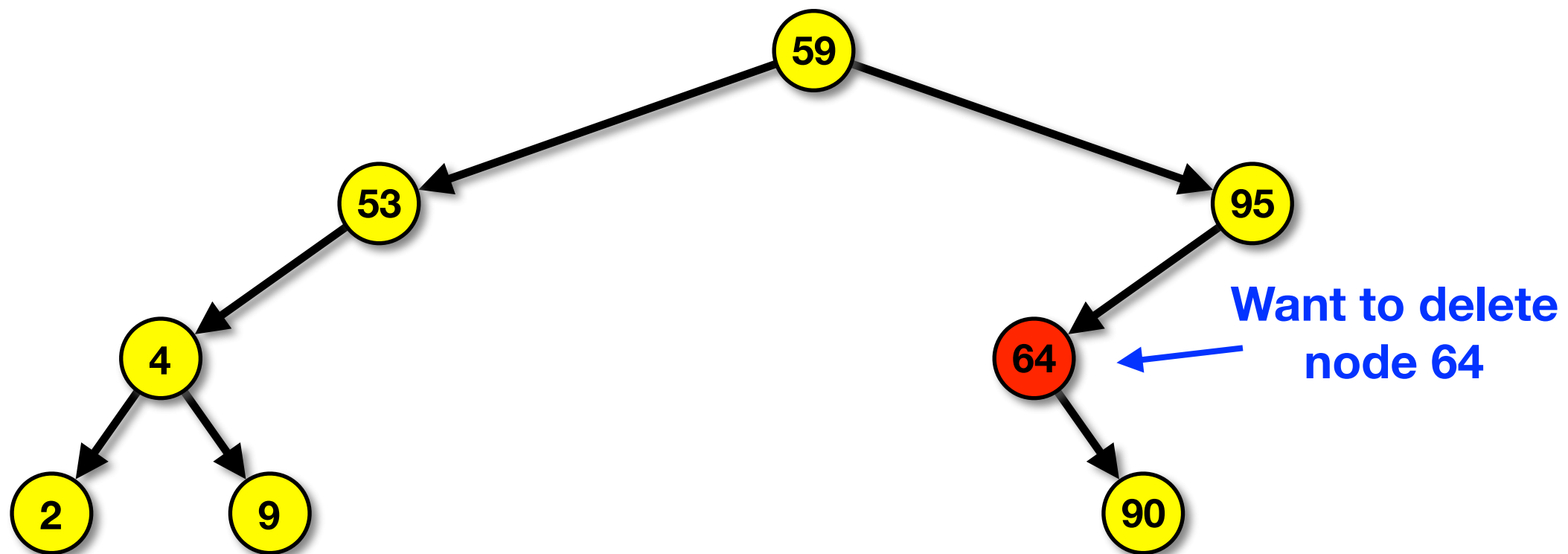
---



**Insert node 59**  
**Result of zig-zig operation**  
**followed by a zig-zig operation**  
**followed by a zig operation**

# Deletion Example

---

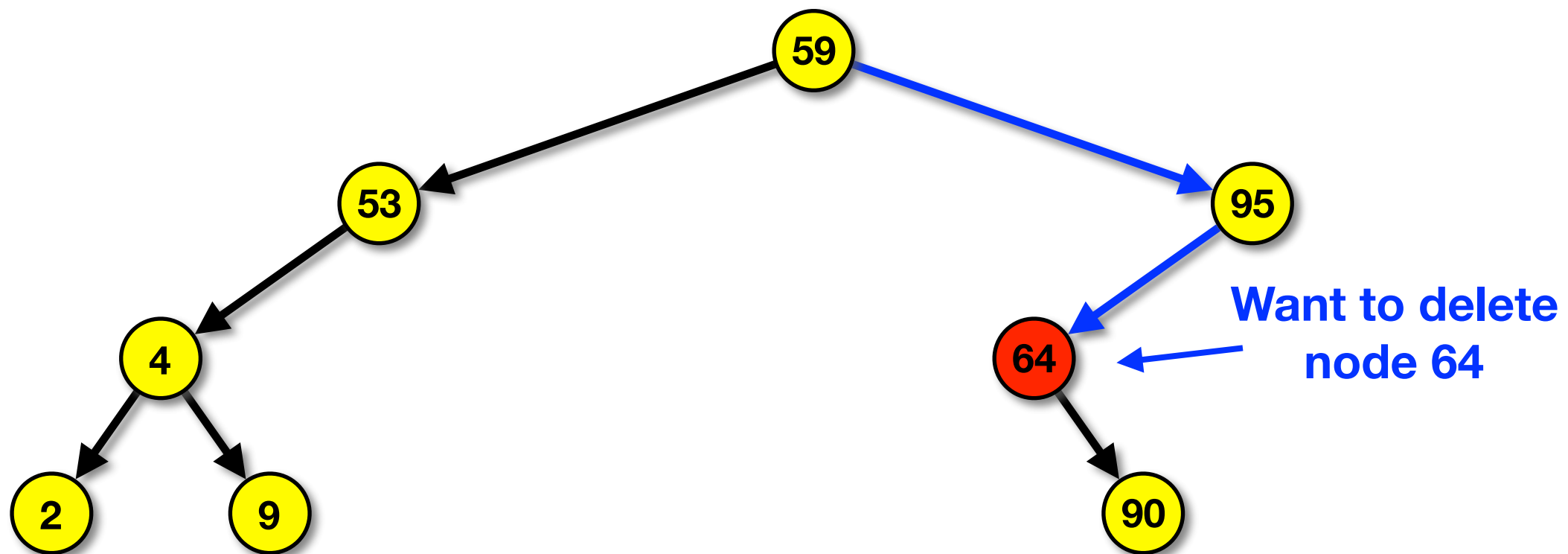


**Delete node 64**

**Step #1 - perform a Find operation to move node 64 to the root**

# Deletion Example

---



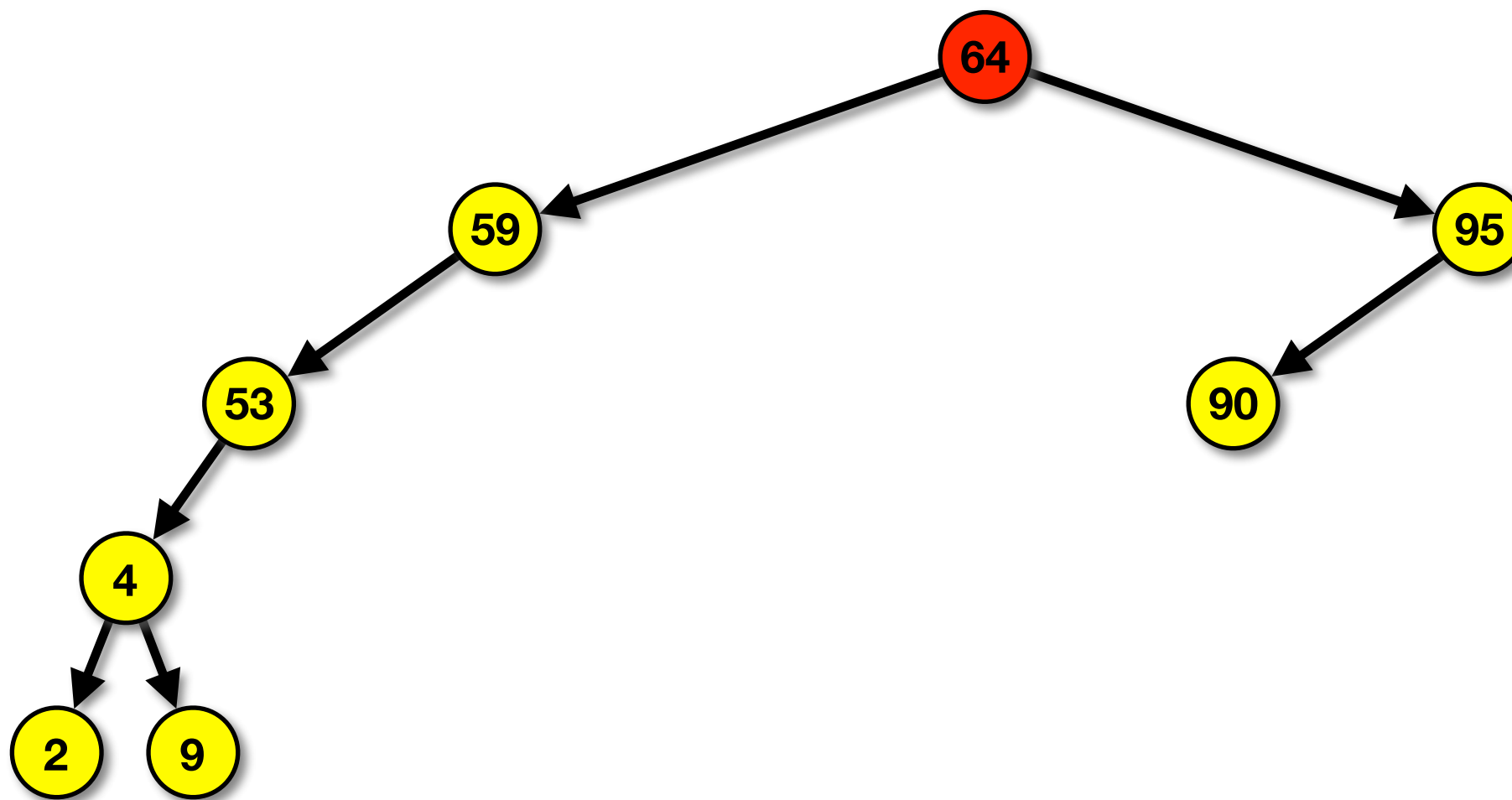
**Delete node 64**

**Step #1 - perform a Find operation to move node 64 to the root**

**- a zig-zag operation is needed**

# Deletion Example

---



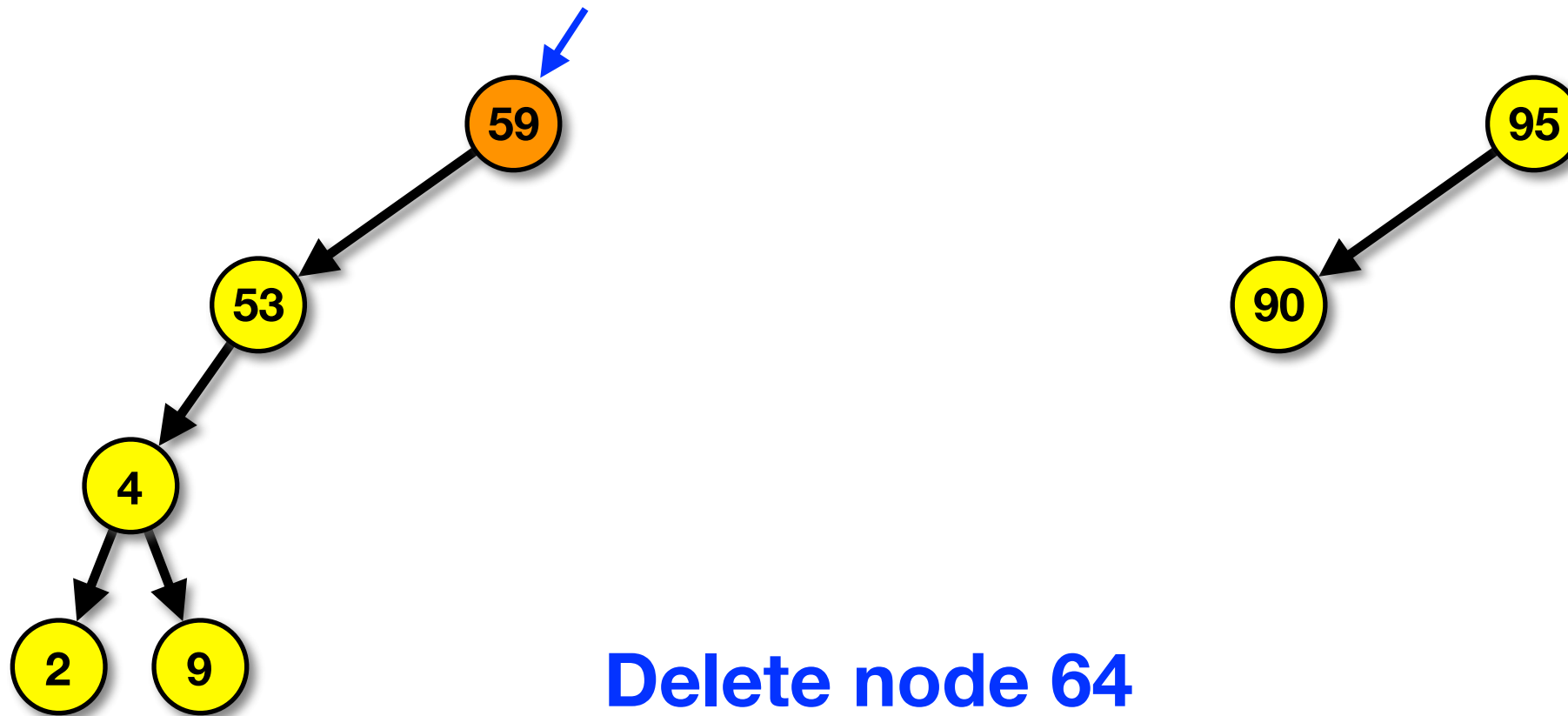
**Delete node 64**

**Step #2 - Now that node 64 is at the root, delete it leaving two unattached subtrees**

# Deletion Example

---

FindMax returns 59, since 59 is already root of left subtree, splay doesn't do anything



**Delete node 64**

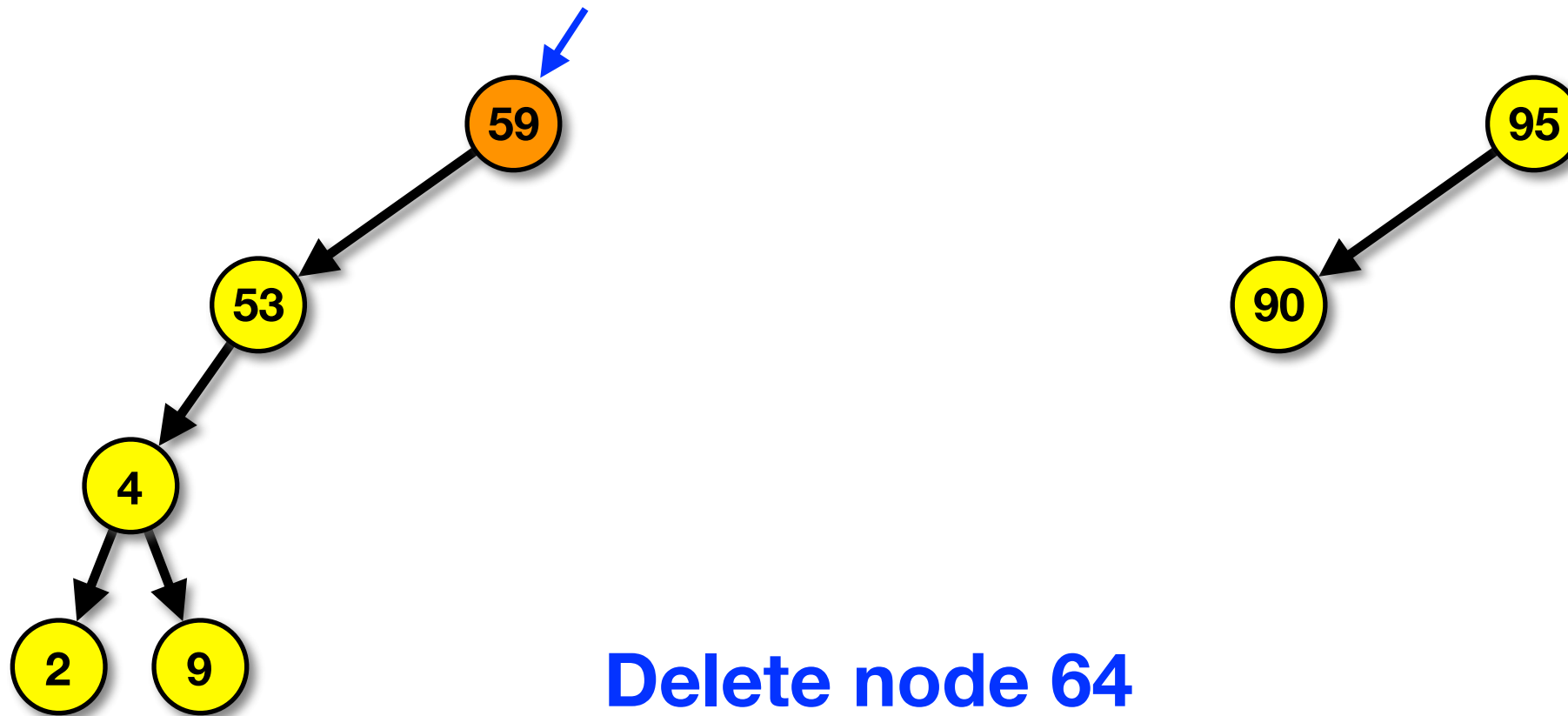
**Step #3 - perform a FindMax on the left subtree and splay that max node to the root of the left subtree**



# Deletion Example

---

FindMax returns 59, since 59 is already root of left subtree, splay doesn't do anything

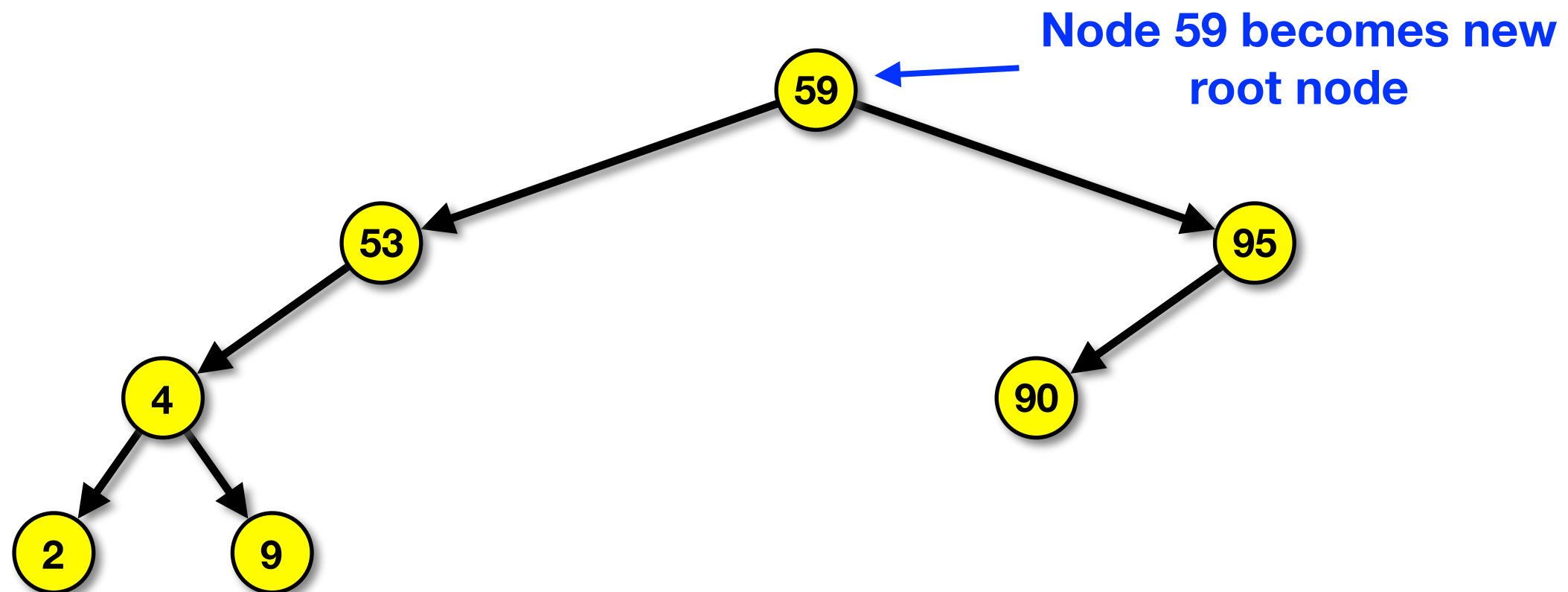


**Delete node 64**

**Step #4 - node 59 will become the new root of the tree and the right subtree will become the right child of node 59**

# Deletion Example

---



**Result of deleting node 64**