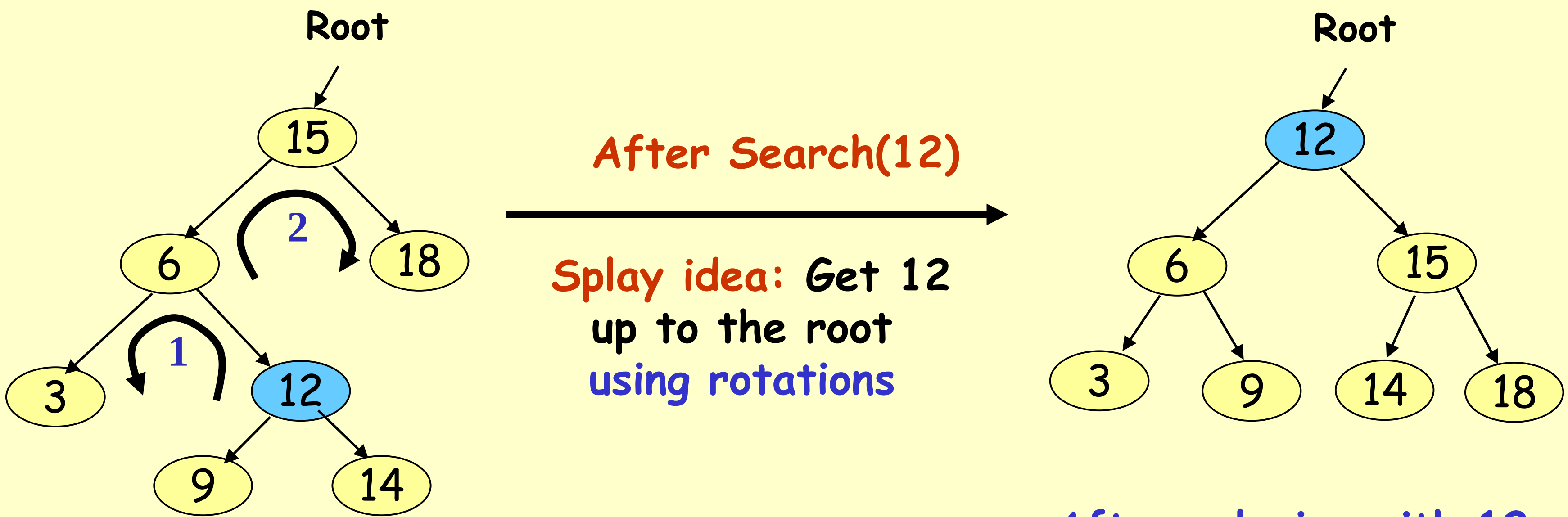
Splay Trees

- Splay trees are binary search trees (BSTs) that:
  - Are not perfectly balanced all the time
  - Allow search and insertion operations to try to balance the tree so that future operations may run faster

- Based on the heuristic:
  - If X is accessed once, it is likely to be accessed again.
  - After node X is accessed, perform "splaying" operations to bring X up to the root of the tree.
  - Do this in a way that leaves the tree more or less balanced as a whole.

Motivating Example



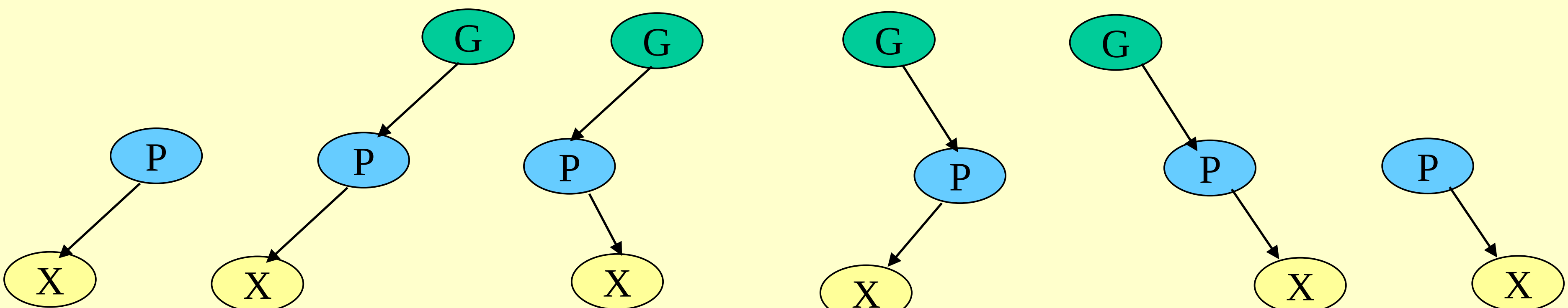
After splaying with 12

Initial tree

- Not only splaying with 12 makes the tree balanced, subsequent accesses for 12 will take O(1) time.
- Active (recently accessed) nodes will move towards the root and inactive nodes will slowly move further from the root

Splay Tree Terminology

- Let X be a non-root node, i.e., has at least 1 ancestor.
- Let P be its parent node.
- Let G be its grandparent node (if it exists)
- Consider a path from G to X:
  - Each time we go left, we say that we "zig"
  - Each time we go right, we say that we "zag"
- There are 6 possible cases:



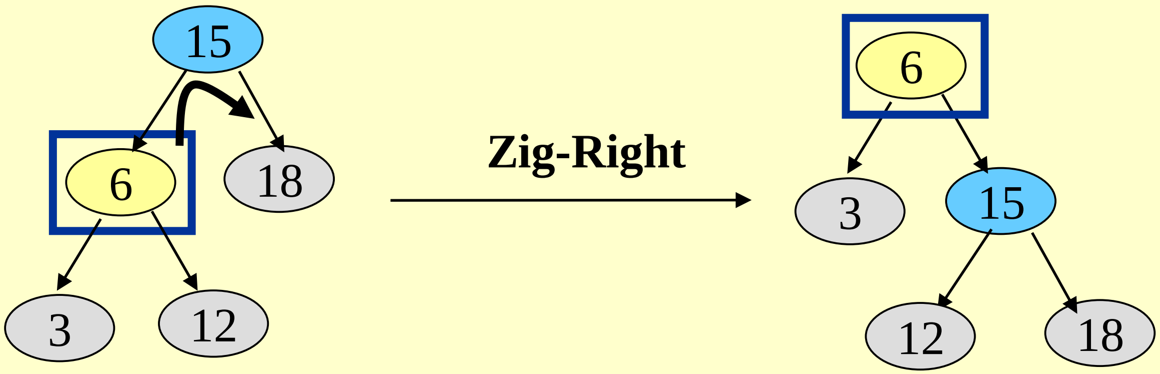
1. zig 2. zig-zig 3. zig-zag 4. zag-zig 5. zag-zag 6. zag

Splay Tree Operations

- When node X is accessed, apply one of six rotation operations:
  - Single Rotations (X has a P but no G)
    - zig, zag
  - Double Rotations (X has both a P and a G)
    - zig-zig, zig-zag
    - zag-zig, zag-zag

Splay Trees: Zig Operation

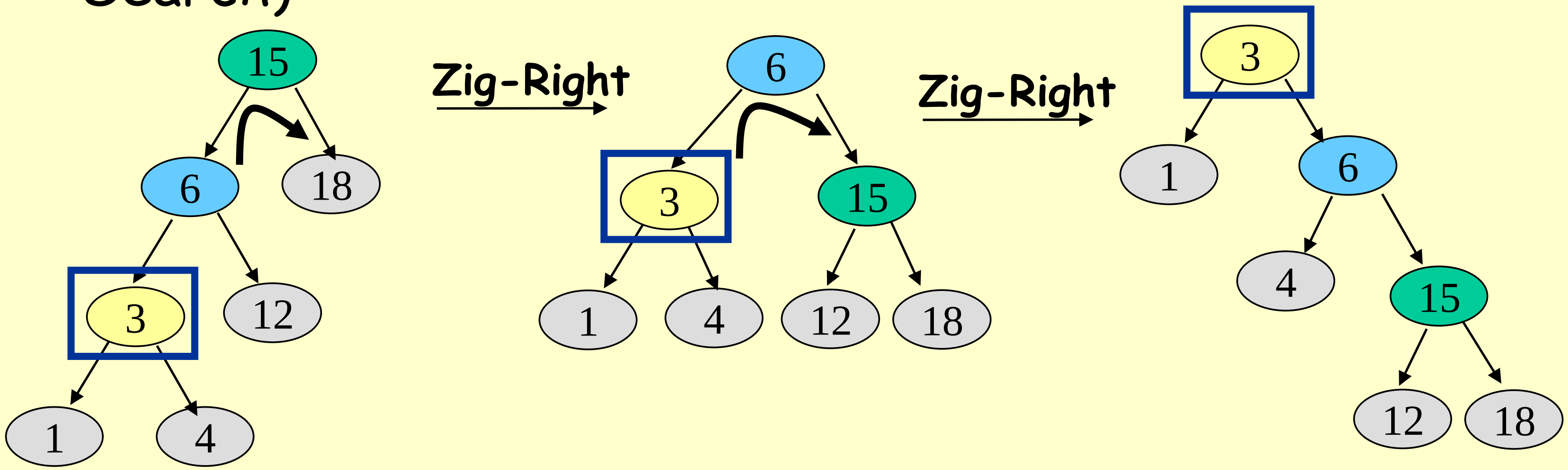
- "Zig" is just a single rotation, as in an AVL tree
- Suppose 6 was the node that was accessed (e.g. using Search)



- "Zig-Right" moves 6 to the root.
- Can access 6 faster next time: O(1)
- Notice that this is simply a right rotation in AVL tree terminology.

Splay Trees: Zig-Zig Operation

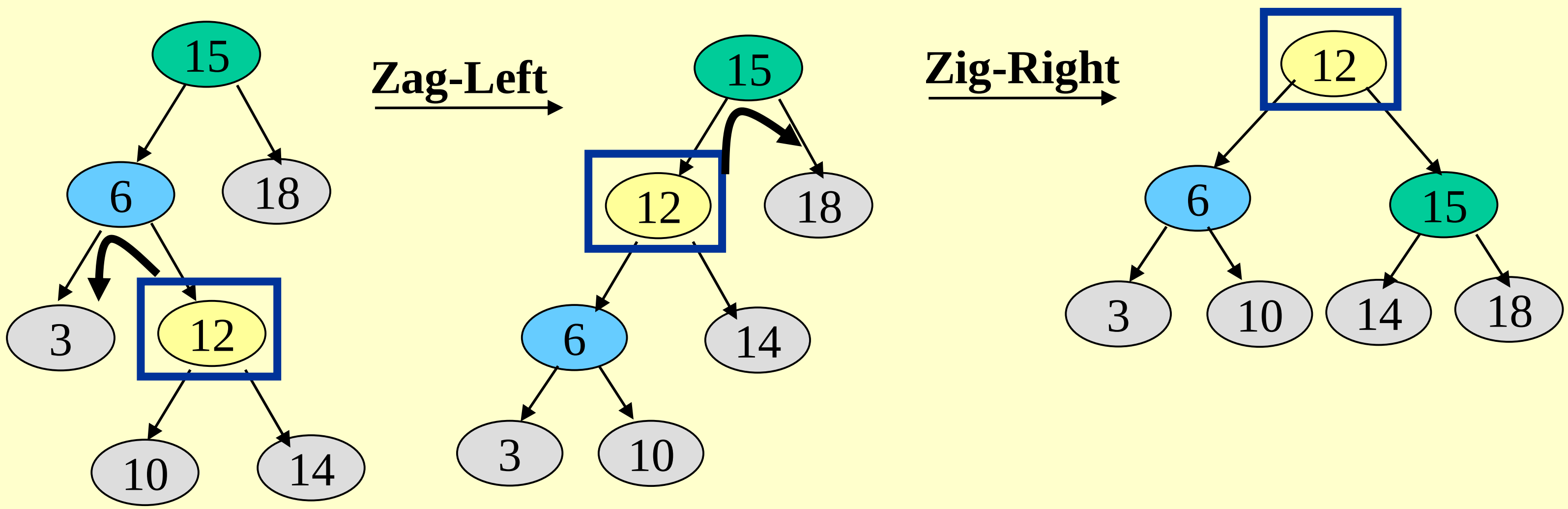
- "Zig-Zig" consists of two single rotations of the same type
- Suppose 3 was the node that was accessed (e.g., using Search)



- Due to "zig-zig" splaying, 3 has bubbled to the top!
- Note: Parent-Grandparent is rotated first.

Splay Trees: Zig-Zag Operation

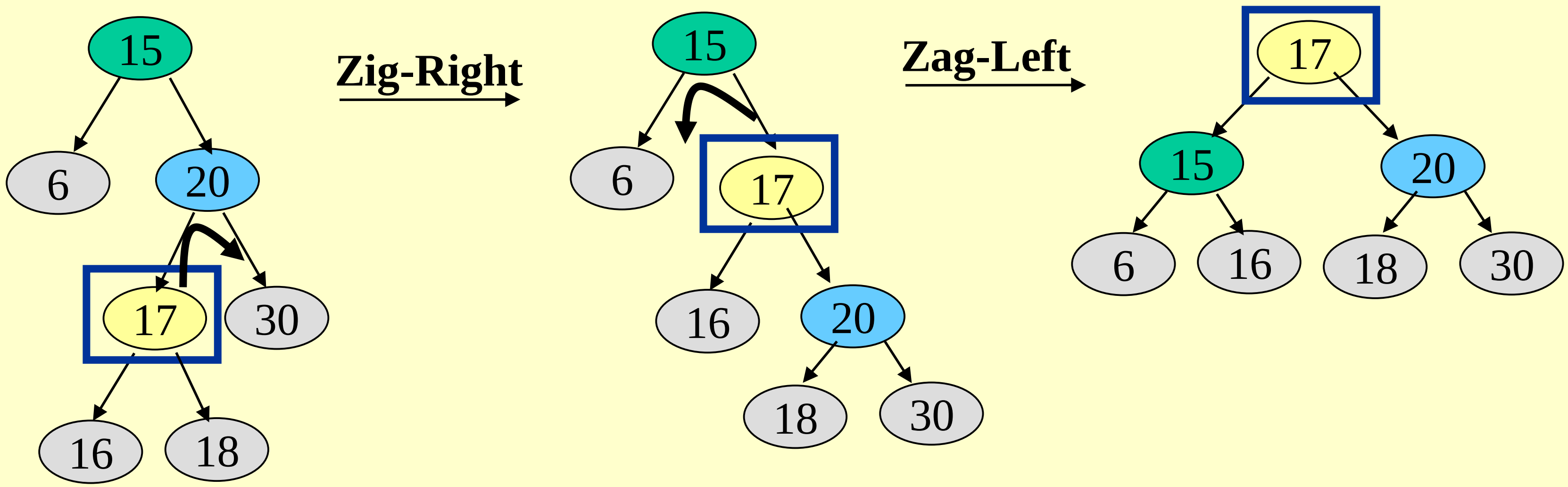
- "Zig-Zag" consists of two rotations of the opposite type
- Suppose 12 was the node that was accessed (e.g., using Search)



- Due to "zig-zag" splaying, 12 has bubbled to the top!
- Notice that this is simply an LR imbalance correction in AVL tree terminology (first a left rotation, then a right rotation)

Splay Trees: Zag-Zig Operation

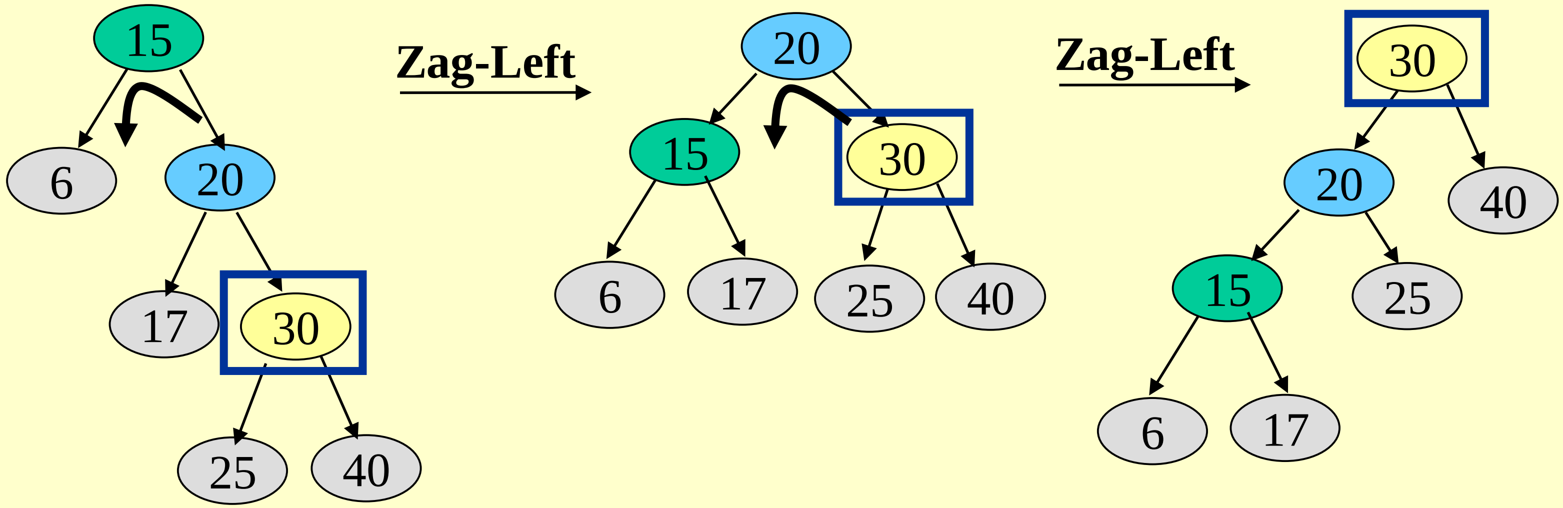
- "Zag-Zig" consists of two rotations of the opposite type
- Suppose 17 was the node that was accessed (e.g., using Search)



- Due to "zag-zig" splaying, 17 has bubbled to the top!
- Notice that this is simply an RL imbalance correction in AVL tree terminology (first a right rotation, then a left rotation)

Splay Trees: Zag-Zag Operation

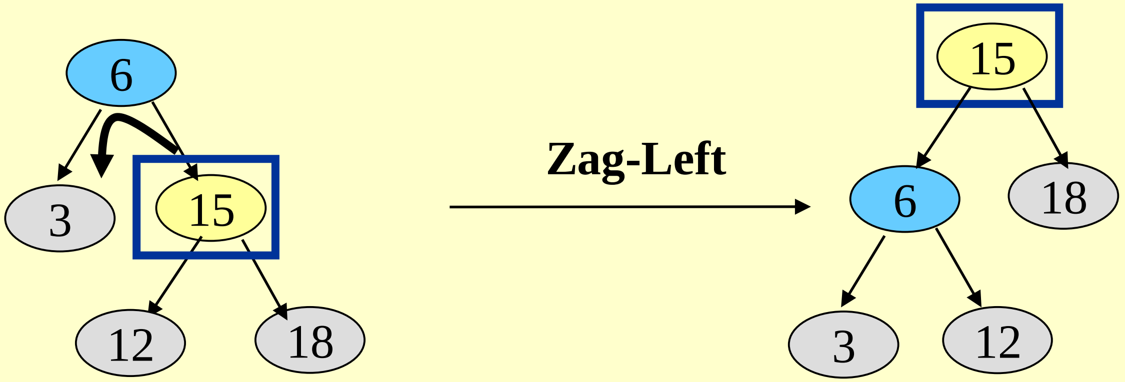
- "Zag-Zag" consists of two single rotations of the same type
- Suppose 30 was the node that was accessed (e.g., using Search)



- Due to "zag-zag" splaying, 30 has bubbled to the top!
- Note: Parent-Grandparent is rotated first.

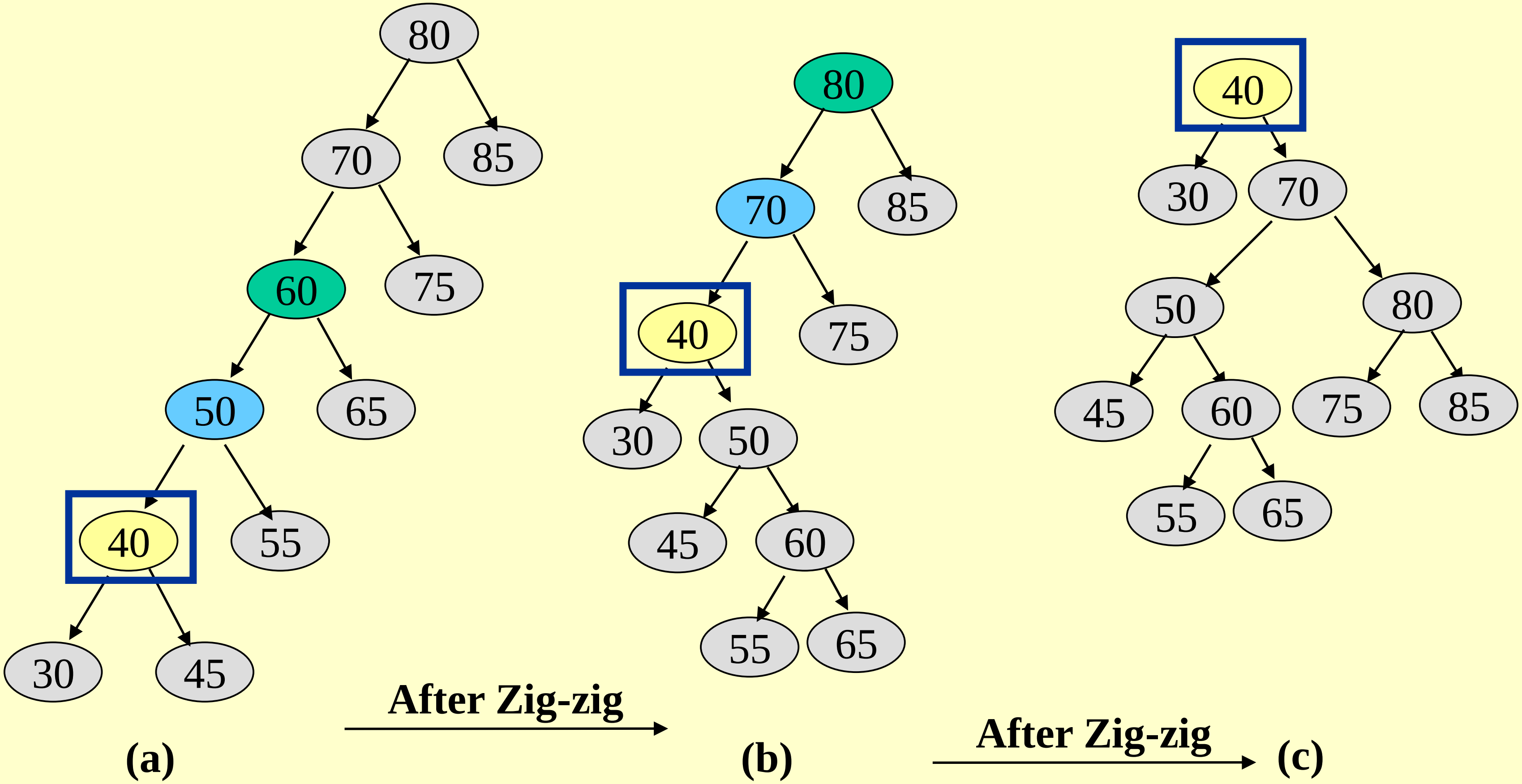
Splay Trees: Zag Operation

- "Zag" is just a single rotation, as in an AVL tree
- Suppose 15 was the node that was accessed (e.g., using Search)



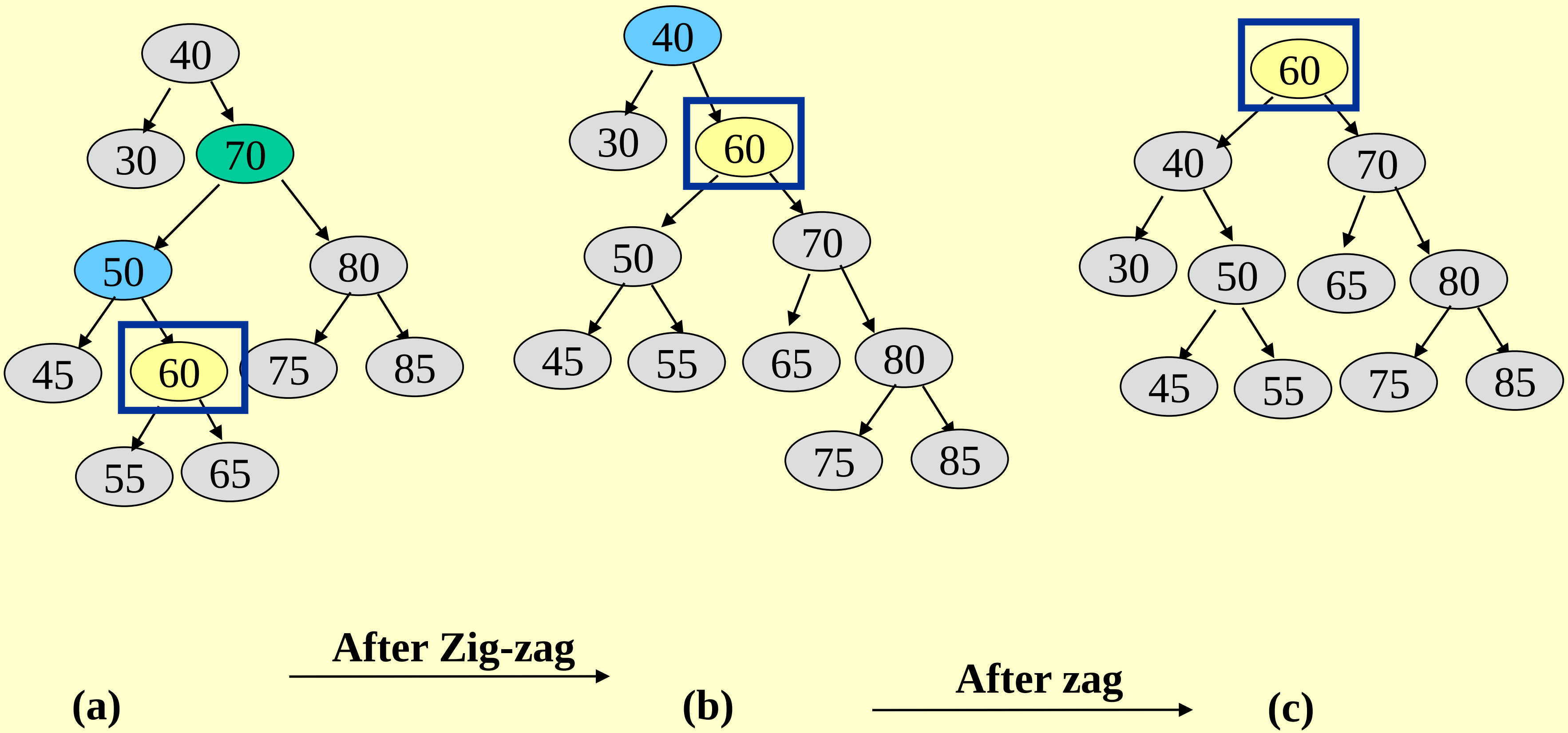
- "Zag-Left"moves 15 to the root.
- Can access 15 faster next time: O(1)
- Notice that this is simply a left rotation in AVL tree terminology

Splay Trees: Example – 40 is accessed



Ex: https://www.cs.usfca.edu/~galles/visualization/SplayTree.html

Splay Trees: Example – 60 is accessed



Splaying during other operations

- Splaying can be done not just after Search, but also after other operations such as Insert/Delete.

- Insert X: After inserting X at a leaf node (as in a regular BST), splay X up to the root

- Delete X: Do a Search on X and get X up to the root. Delete X at the root and move the largest item in its left sub-tree, i.e, its predecessor, to the root using splaying.

- Note on Search X: If X was not found, splay the leaf node that the Search ended up with to the root.

Example

- Insert 30, 4, 12,25, 9, 45 into an empty splay tree in the given order.

Summary of Splay Trees

- Examples suggest that splaying causes tree to get balanced.
- The actual analysis is rather advanced and is in Chapter 11. Such analysis is called "amortized analysis"
- Result of Analysis: Any sequence of M operations on a splay tree of size N takes O(M log N) time. So, the amortized running time for one operation is O(log N).
- This guarantees that even if the depths of some nodes get very large, you cannot get a long sequence of O(N) searches because each search operation causes a rebalance.
- Without splaying, total time could be O(MN).