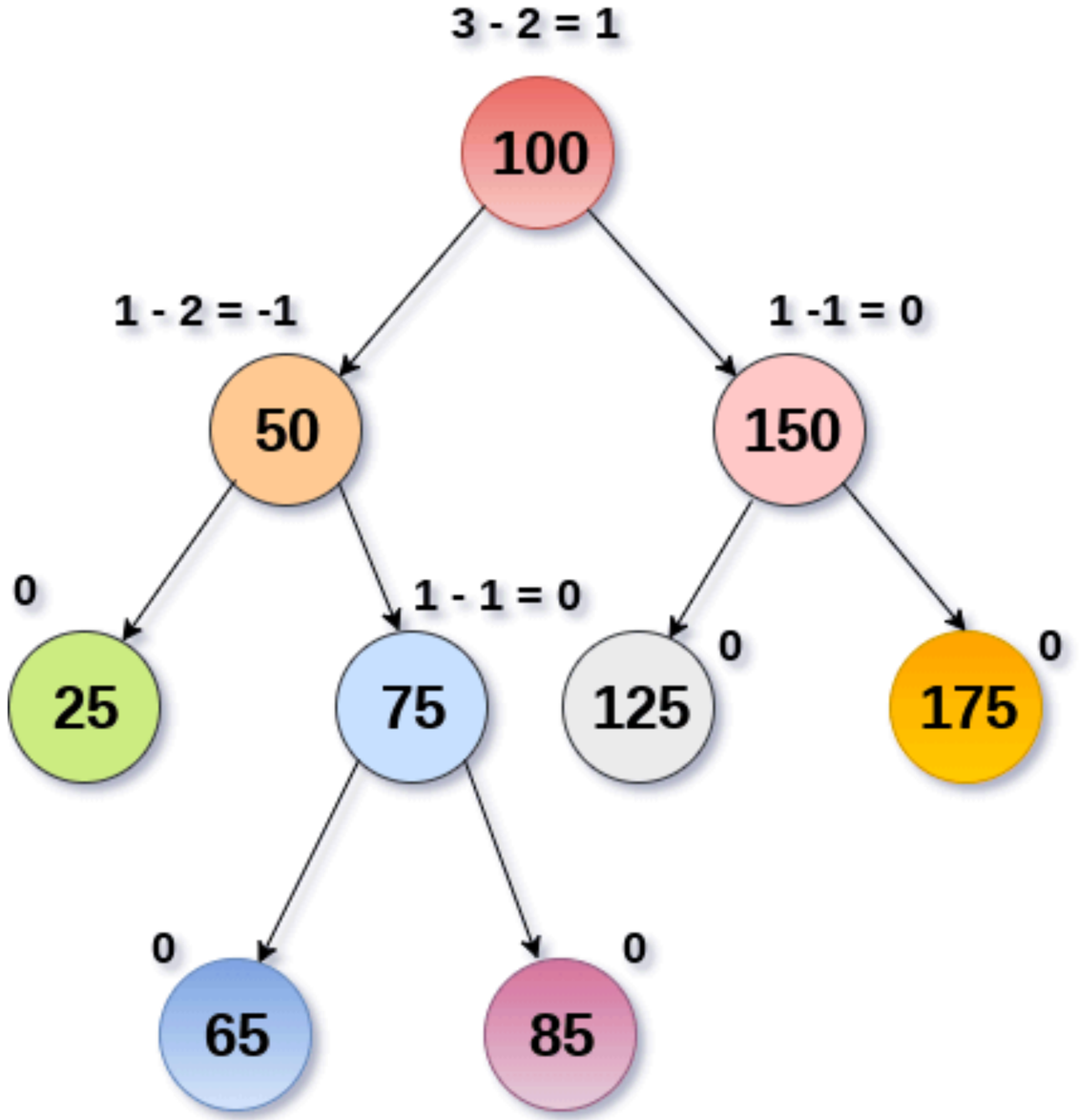
AVL Tree and B-Tree

Aniket Pingley, Ph.D.

AVL Tree

Height balanced BST

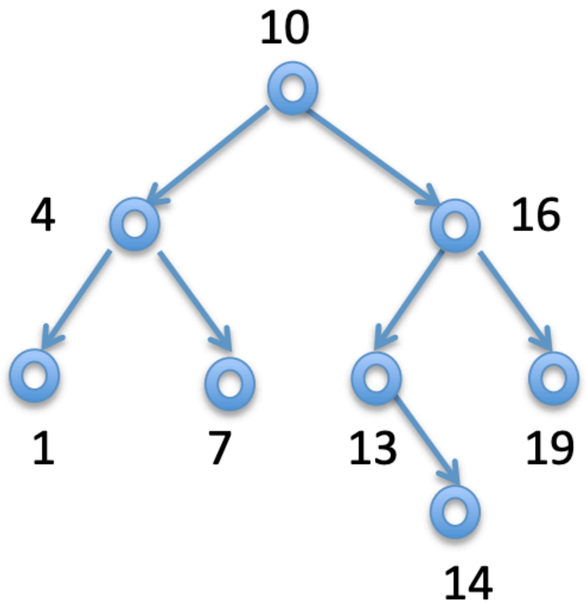
- Each node is associated with a balance factor: (height of left sub-tree) - (height of its right sub-tree)
- Balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.
- GM Adelson - Velsky and EM Landis in 1962



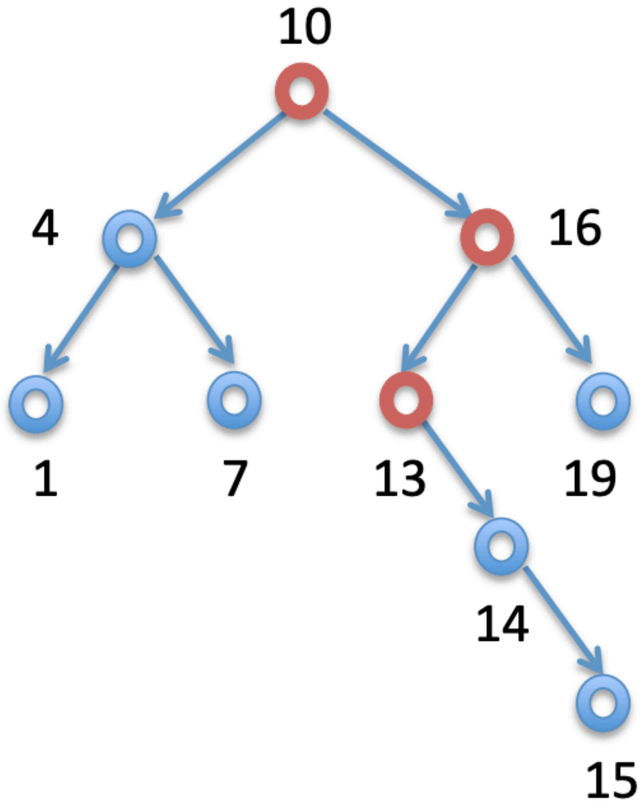
AVL Tree

Insertion in AVL Trees

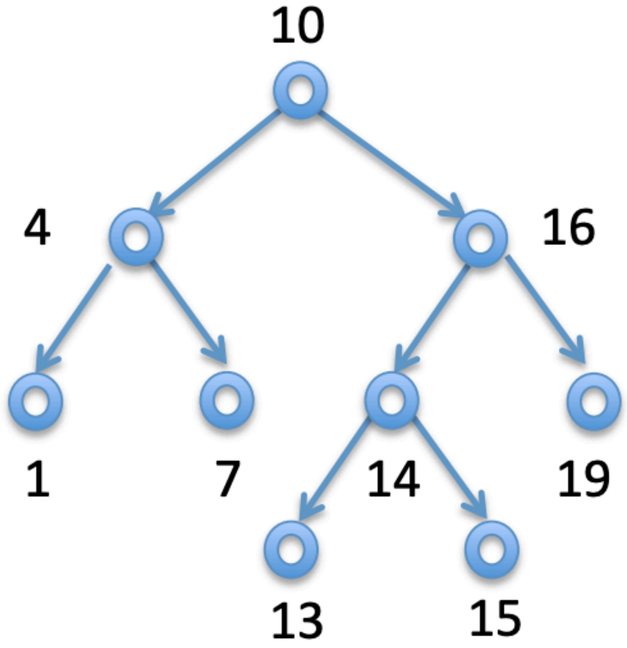
14 is inserted in BST
Balance factor is fine



15 is inserted in BST
Imbalance created at node 10, 16 and 13

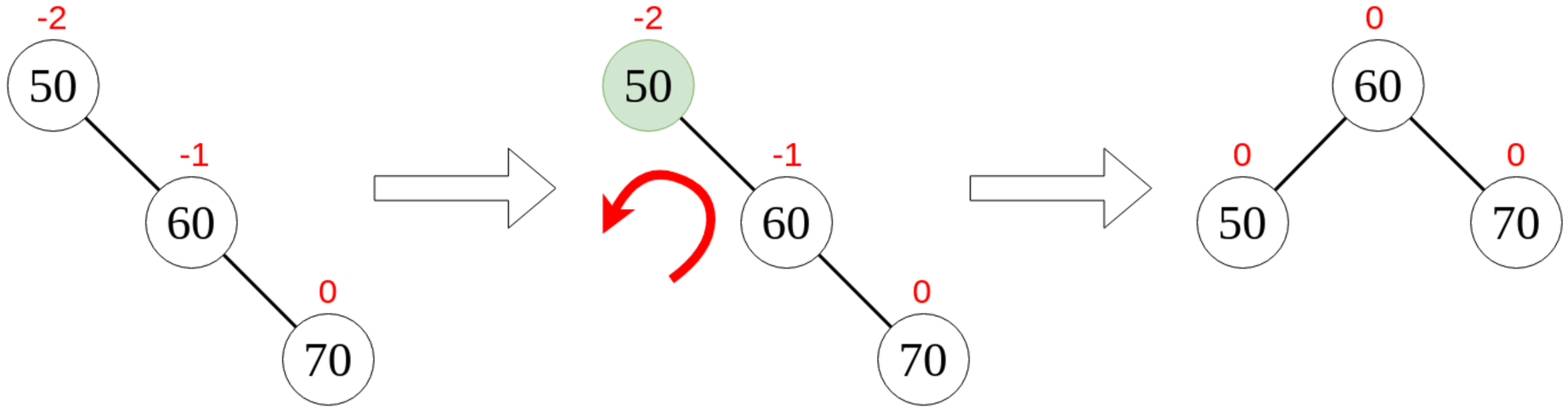


Balance is restored by
rearranging the subtree 13, 14, 15



Rotations in AVL Trees

Single Left Rotation



The tree is not balanced

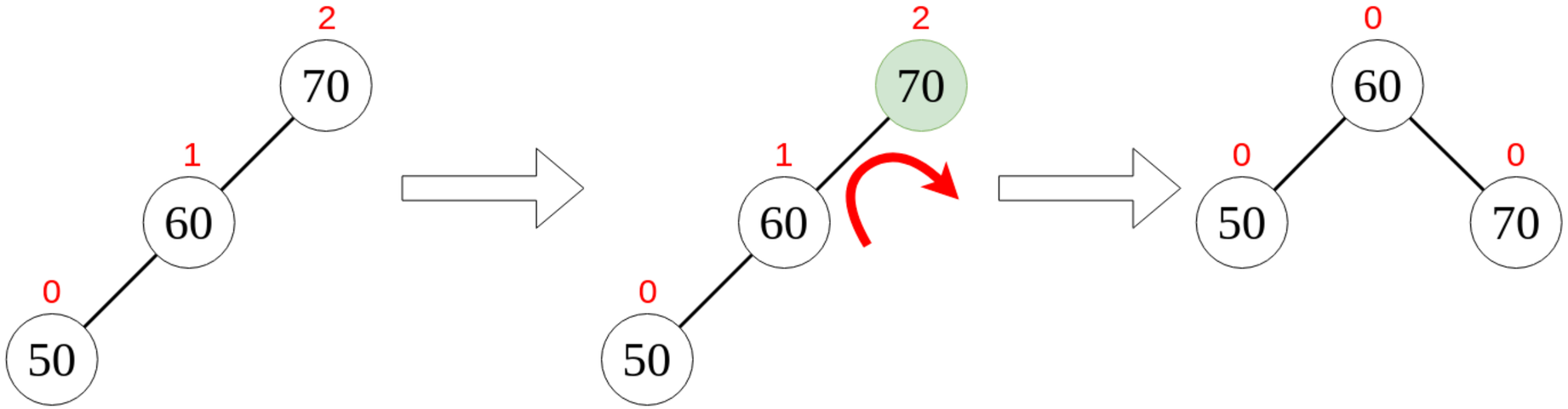
Do Left rotation about node 50

The tree is balanced

When the (sub)tree is skewed towards right

Rotations in AVL Trees

Single Right Rotation



The tree is not balanced

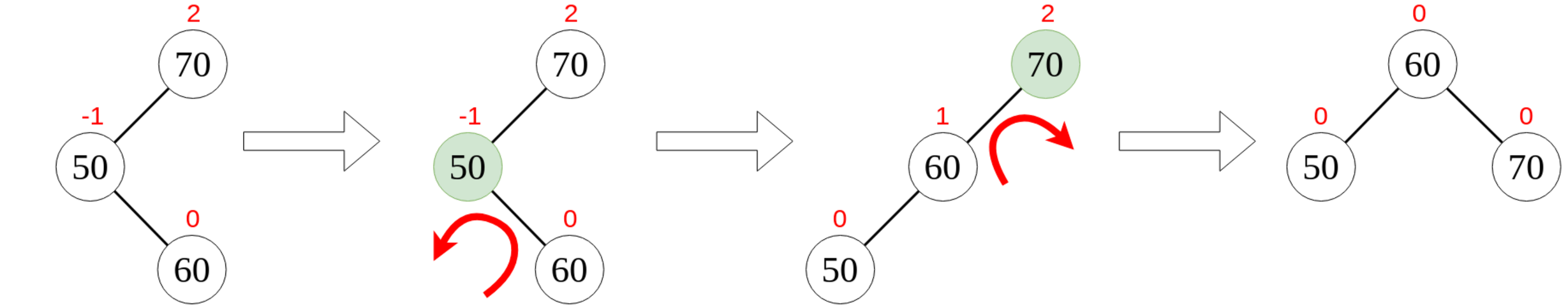
Do Right rotation about node 70

The tree is balanced

When the (sub)tree is skewed towards left

Rotations in AVL Trees

Double Left Right Rotation



The tree is not balanced

Do Left rotation about node 50

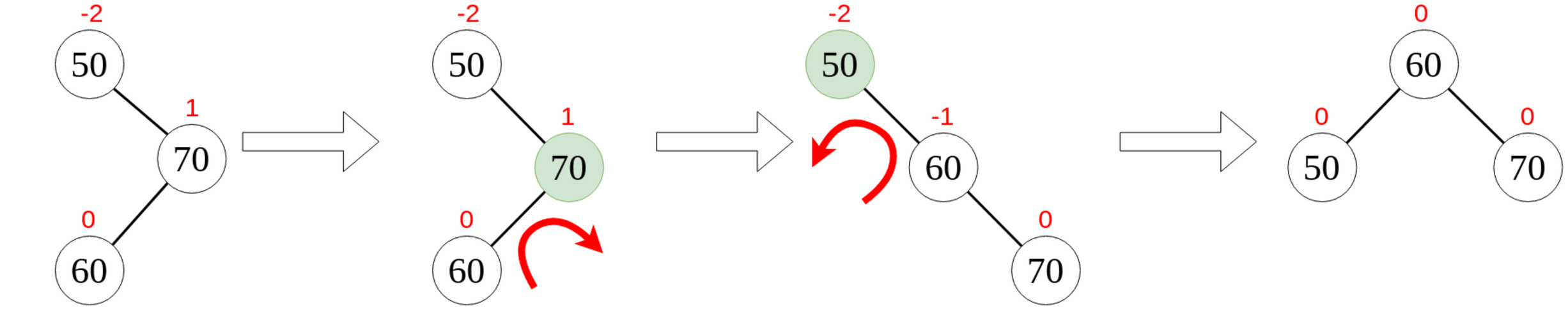
Do Right rotation about node 70

The tree is balanced

Modify the (sub)tree to become skewed towards left and the rotate right

Rotations in AVL Trees

Double Right Left Rotation



The tree is not balanced

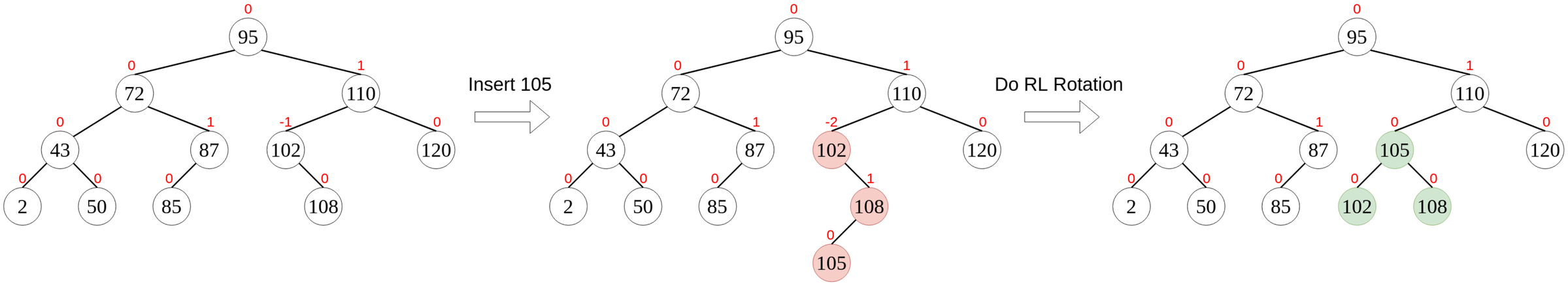
Do Right rotation about node 70

Do Left rotation about node 50

The tree is balanced

Modify the (sub)tree to become skewed towards right and the rotate left

Rebalancing after Insertion using Rotation



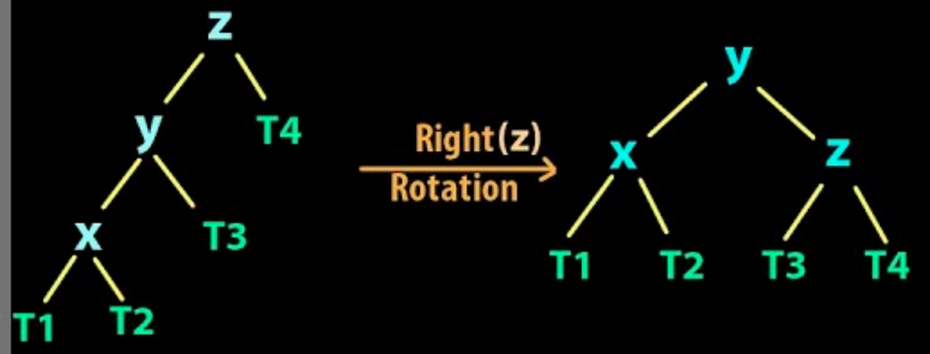
The tree is balanced

The tree is not balanced

The tree is balanced

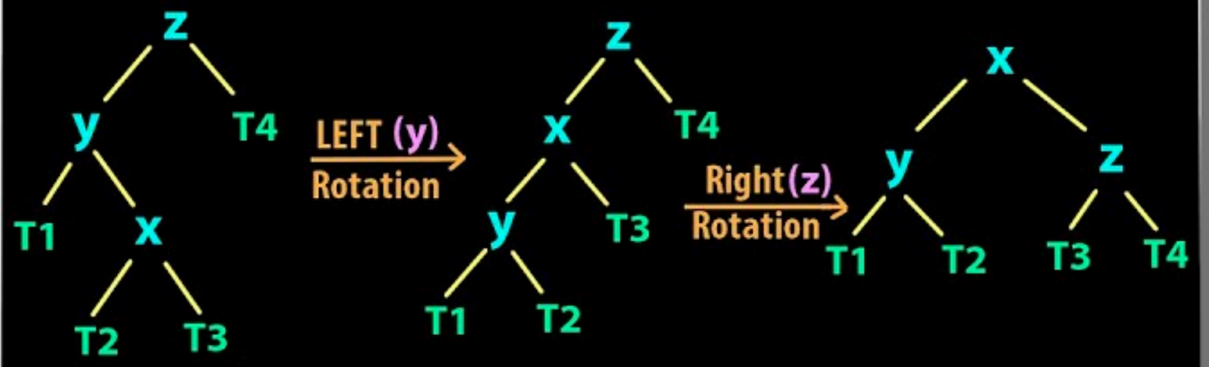
AVL TREE ROTATIONS (For more than 3 nodes)

LEFT LEFT Case/Imbalance



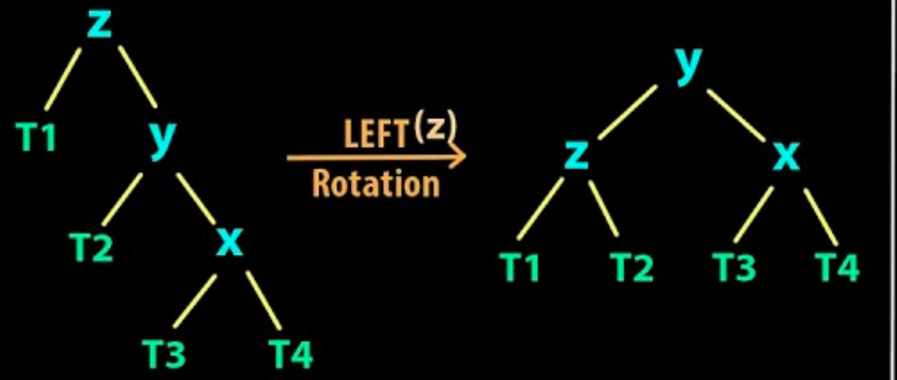
T1, T2, T3 and T4 are subtrees.

LEFT RIGHT case/imbalance



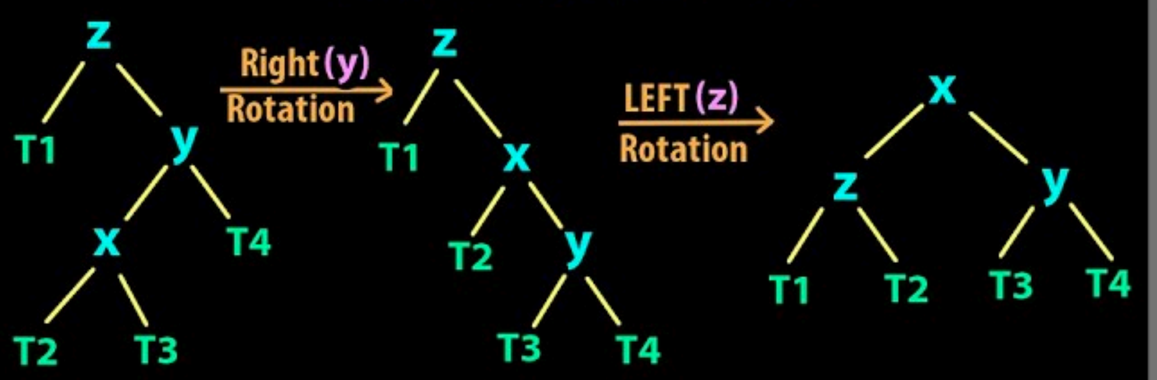
T1, T2, T3 and T4 are subtrees.

RIGHT RIGHT case/imbalance



T1, T2, T3 and T4 are subtrees.

RIGHT LEFT case/imbalance



T1, T2, T3 and T4 are subtrees.

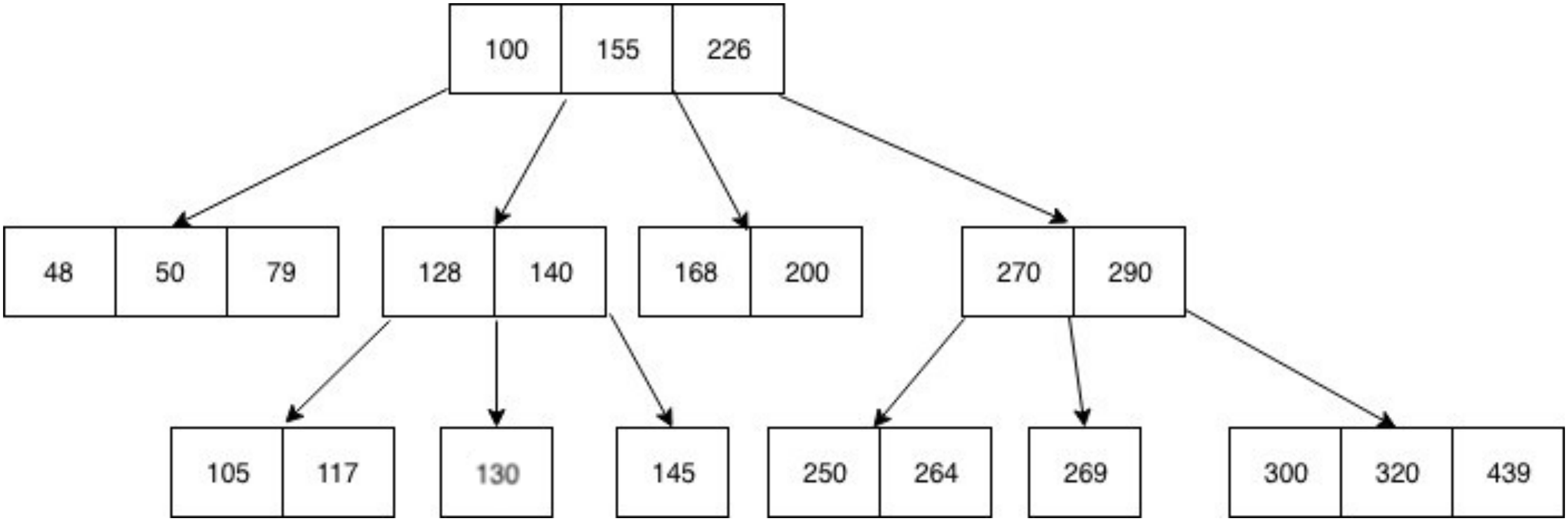
SIMPLE SNIPPETS

Binary Trees

Points to ponder over

- Is it limiting to have only two child nodes?
- For a very large tree, will the height of binary tree result is performance issues?
- What if the data is very large - something that cannot fit in main memory?
- Is binary tree the best mechanism to maintain access to data on disk?

B-Tree


100
155
226
48
50
79
128
140
168
200
270
290
105
117
130
145
250
264
269
300
320
439

B-Tree

- Invented by R. Bayer and E. McCreight in 1972
- Used is most modern file systems
- Generalized implementation of binary search tree - i.e. order is maintained but the branching factor can be greater than 2 - Multi-way search trees
- Self-balancing, shallow with high level of branching
- Height-balance is strict - all nodes at the same level
- Optimized for situations when part or all of the tree must be maintained in secondary storage
- Every node in the tree will be full at least to a certain minimum percentage. This improves space efficiency while reducing the number of disk fetches during a search or update operation.

B-Tree

- A B-Tree of order m is defined to have the following shape properties:
  - The root is either a leaf or has at least two children.
  - Each internal node, except for the root, has between m/2 and m children.
  - All leaves are at the same level in the tree, so the tree is always height balanced.
  - If a node contains N keys, then the node has N + 1 children
  - Keys in the nodes are ordered in ascending order (inorder traversal)

B-Tree

Run-time analysis

- For B-Tree of order M
  - Each internal node has up to M-1 keys and children between M/2 & M
  - Depth of B-Tree for storing N items is $\log_{M/2} N$
- Find/search/lookup
  - $O(\log_2 M)$ to determine the branch at a given node
  - Total time = $O(\log_2 M * \log_{M/2} N) = O(\log_{M/2} N)$, given N >>> M
- Time complexity for insertion and deletion ???

Insertion in B-Tree

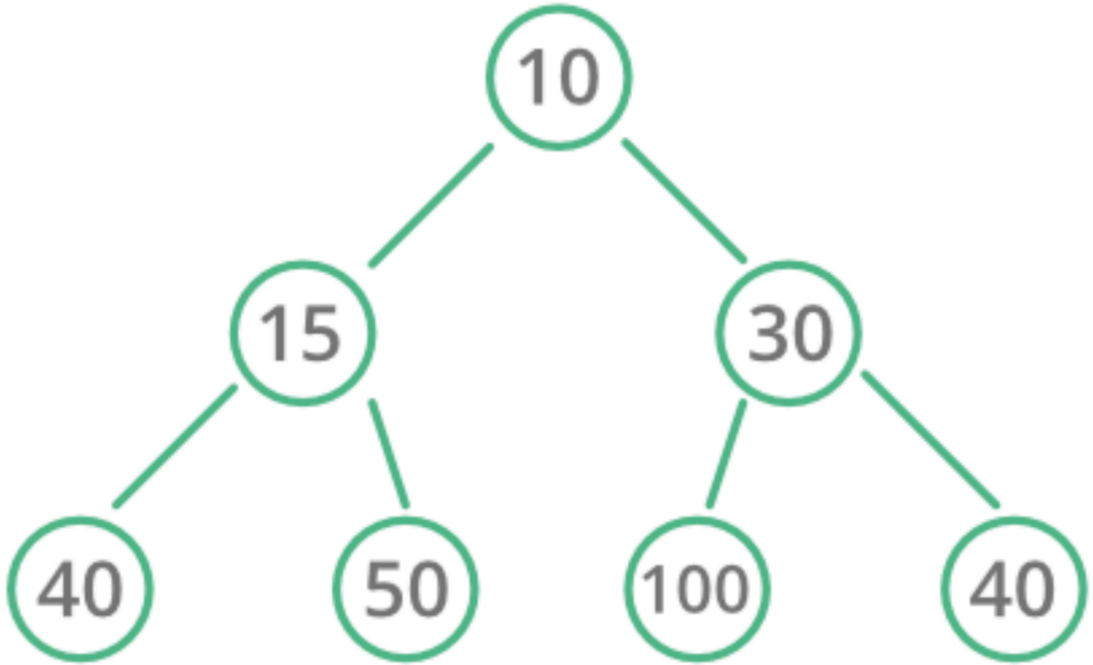
- Insert input value only into a leaf node
- Two conditions of a node in B-Tree
- Full node
  - Split the node into two at the median and move the median value into parent (in the right order)
  - If value to be inserted in less than the median, continue the process for the left part of the split node, otherwise the right part
- Non-full
  - If the node is leaf, insert the node in the correct order
  - Else, move to the correct child as per the order of lookup

Binary Heap

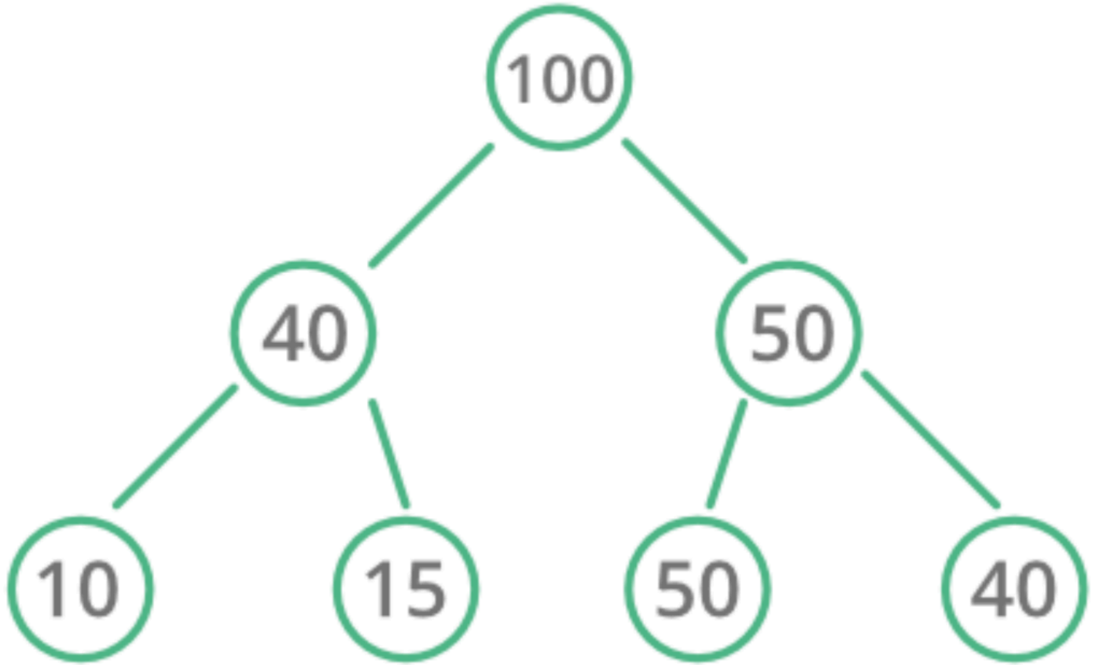
- A binary heap is a complete binary tree that satisfies the heap property - min and max
- A min-heap is a binary tree such that - the data contained in each node is less than (or equal to) the data in that node’s children
- A max-heap is a binary tree such that - the data contained in each node is greater than (or equal to) the data in that node’s children.

Binary Heap

Complete Binary Tree that satisfies the heap property



Min Heap



Max Heap

Array Representation for Binary Heap

- Use an array to hold the data
- Store the root at index 1
- For any node at index i
  - its left child (if any) is in position 2i
  - its right child (if any) is in position 2i + 1
  - its parent (if any) is in position i/2 (integer division)