AVL-Trees: Motivation

- Recall our discussion on BSTs
  - The height of a BST depends on the order of insertion
    - E.g., Insert keys 1, 2, 3, 4, 5, 6, 7 into an empty BST
    - Problem: Lack of "balance" – Tree becomes highly asymmetric and degenerates to a linked-list!!
  - Since all operations take O(h) time, where $\log N \le h \le N-1$, worst case running time of BST operations are O(N)

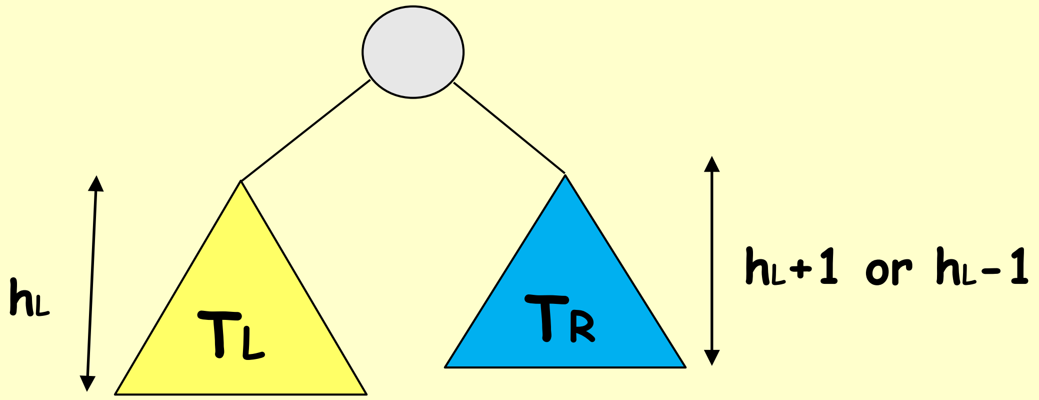
- Question: Can we make sure that regardless of the order of key insertion, the height of the BST is log(n)? In other words, can we keep the BST balanced?

Height-Balanced Trees

- Many efficient algorithms exist for balancing BSTs in order to achieve faster running times for the BST operations
  - Adelson-Velskii and Landis (AVL) trees (1962)
  - Splay trees and other self-adjusting trees (1978)
  - B-trees and other multiway search trees (1972)
  - Red-Black trees (1972)
    - Also known as Symmetric Binary B-Trees
    - Will not be covered in this course

AVL Trees: Formal Definition

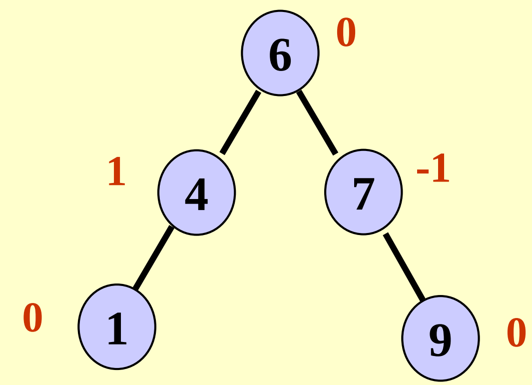
1. All empty trees are AVL-trees
2. If T is a non-empty binary search tree with T_L and T_R as its left and right sub-trees, then T is an AVL tree iff
   1. T_L and T_R are AVL trees
   2. $|h_L - h_R| <= 1$, where h_L and h_R are the heights of T_L and T_R respectively



AVL Trees

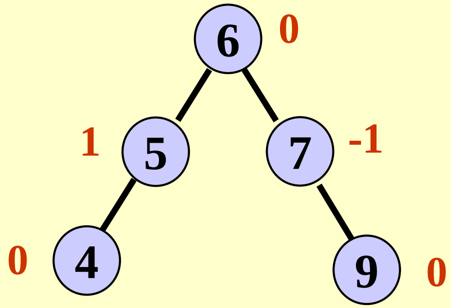
- AVL trees are height-balanced binary search trees
- Balance factor of a node = height(left subtree) - height(right subtree)
- An AVL tree can only have balance factors of –1, 0, or 1 at every node
- For every node, heights of left and right subtree differ by no more than 1

An AVL Tree

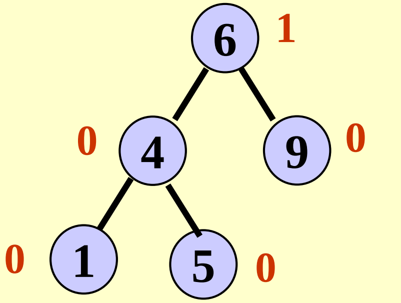


Red numbers
are Balance Factors

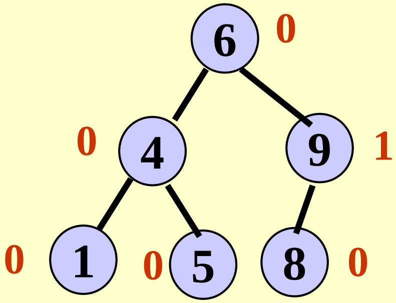
AVL Trees: Examples and Non-Examples



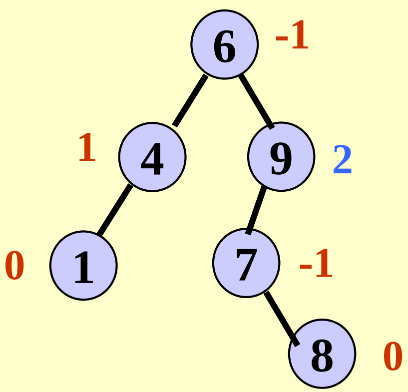
An AVL Tree



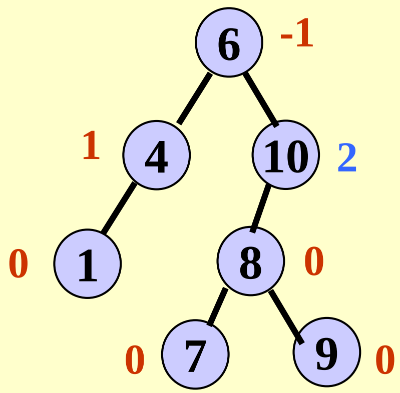
An AVL Tree



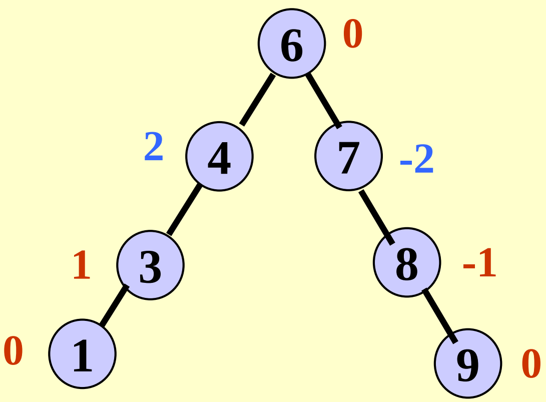
An AVL Tree



Non-AVL Tree



Non-AVL Tree

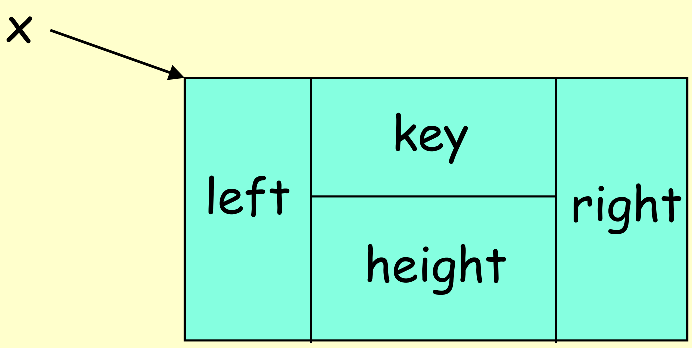


Non-AVL Tree

Red numbers are Balance Factors

AVL Trees: Implementation

- To implement AVL-trees, associate a height with each node "x"



Java Declaration

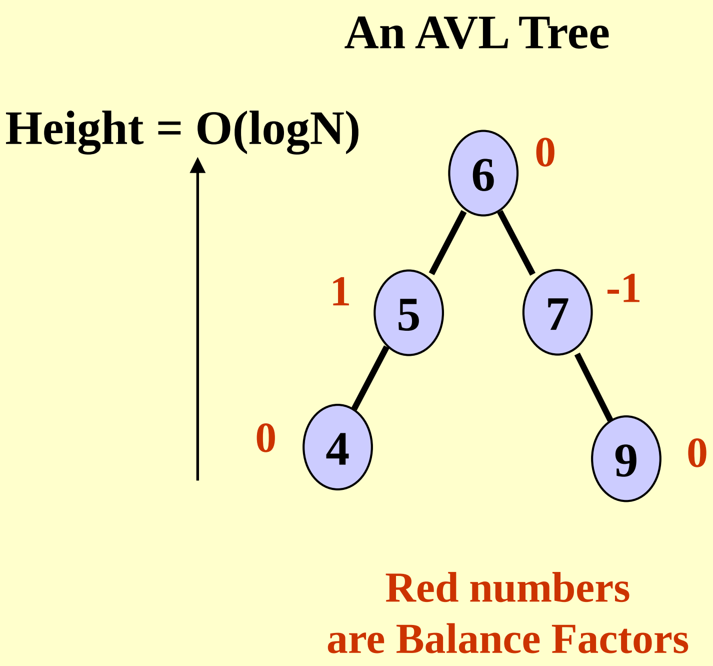
```
Class AVLTreeNode{
    int key;
    int height;
    AVLTreeNode left;
    AVLTreeNode right;
}
```

- Balance factor (bf) of x = height of left subtree of x – height of right subtree of x
- In an AVL-tree, "bf" can be one of {-1, 0, 1}

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

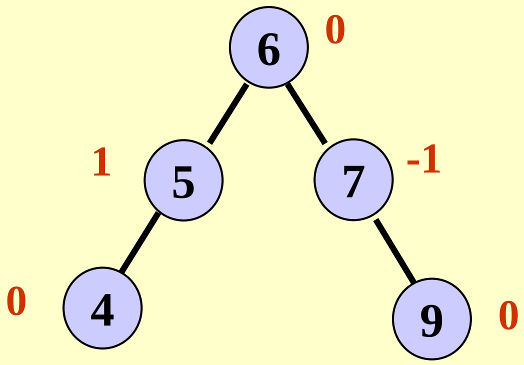
Good News about AVL Trees

- Can prove: Height of an AVL tree of N nodes is always O(log N)

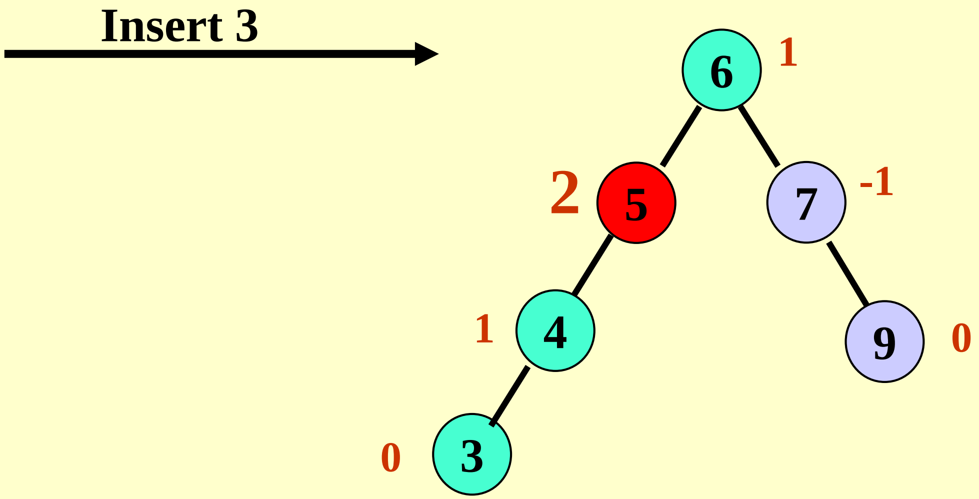


Good and Bad News about AVL Trees

- Good News:
  - Search takes O(h) = O(logN)
- Bad News
  - Insert and Delete may cause the tree to be unbalanced!



An AVL Tree

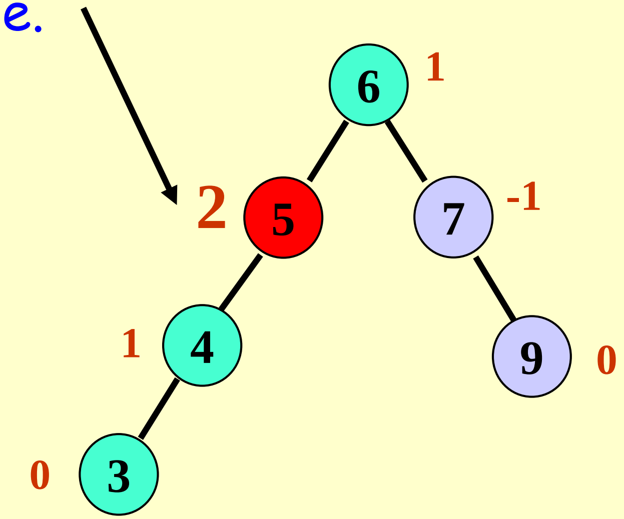


No longer an AVL Tree

Restoring Balance in an AVL Tree

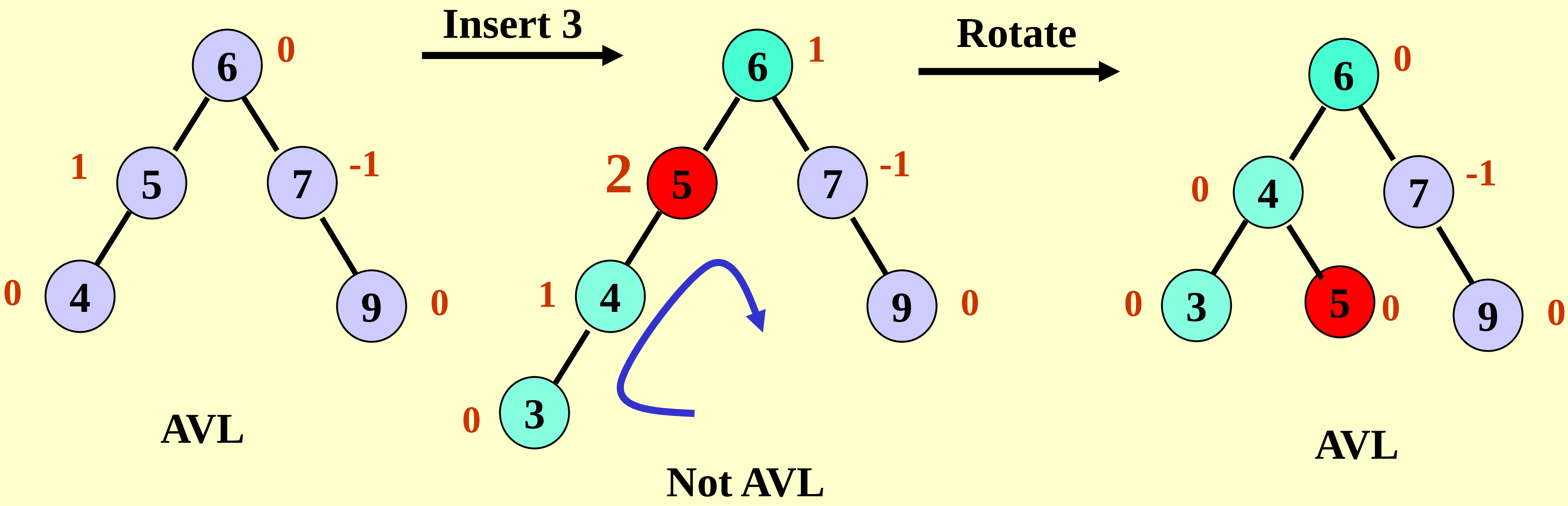
- **Problem**: Insert may cause balance factor to become 2 or –2 for some node on the path from root to insertion point

- **Idea**: After Inserting the new node
  1. Back up towards root updating balance factors along the access path
  2. If Balance Factor of a node = 2 or –2, adjust the tree by rotation around deepest such node.



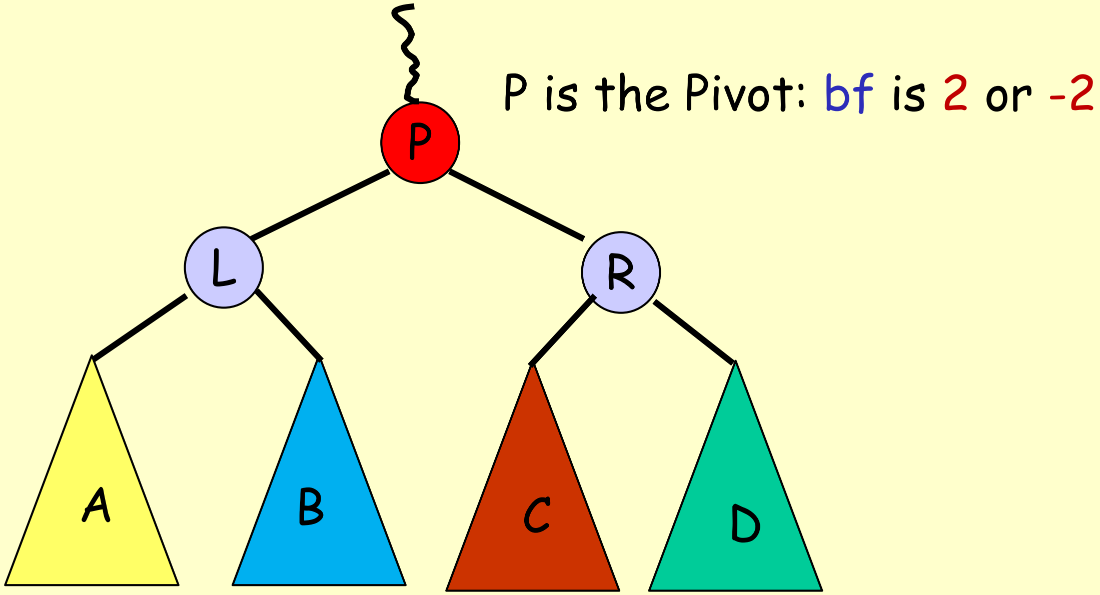
Non- AVL Tree

Restoring Balance: Example



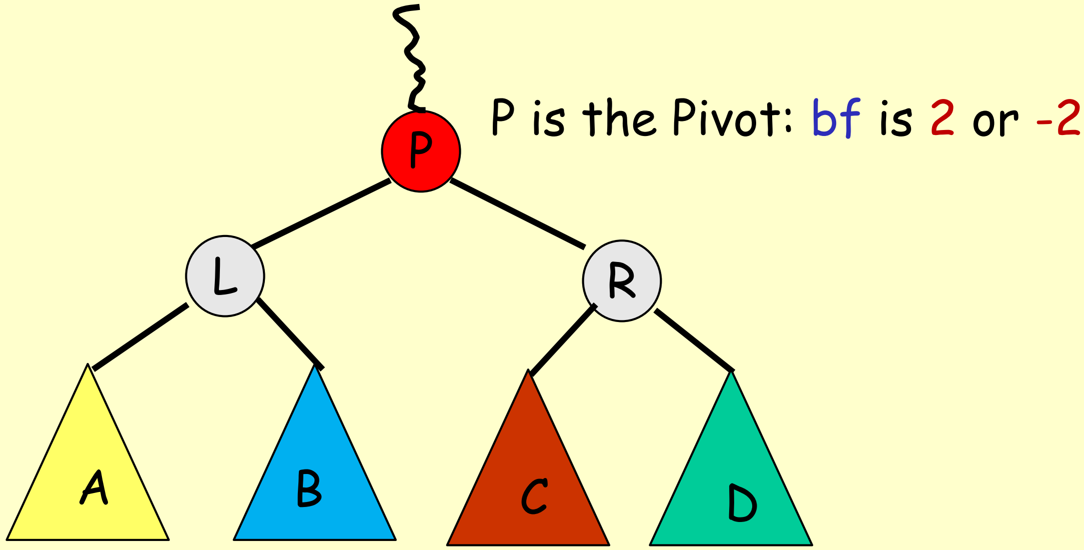
- After Inserting the new node
  1. Back up towards root updating heights along the access path
  2. If Balance Factor of a node = 2 or -2, adjust the tree by rotation around deepest such node.

AVL Tree Insertion (1)



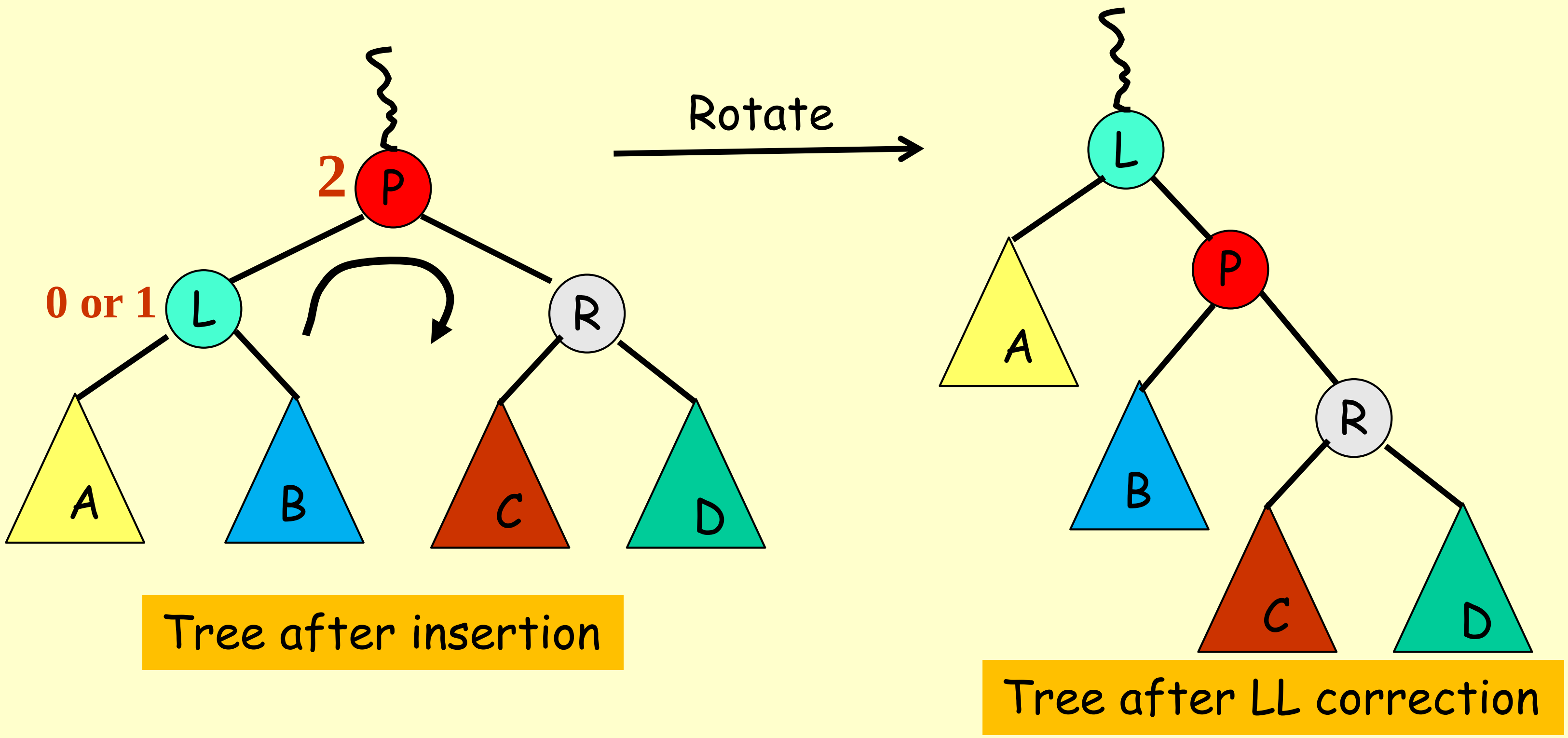
- Let the node that needs rebalancing be P.
  - P is called the pivot node
  - P is the first node that has a bf of 2 or -2 as we backup towards the root after an insertion

AVL Tree Insertion (2)



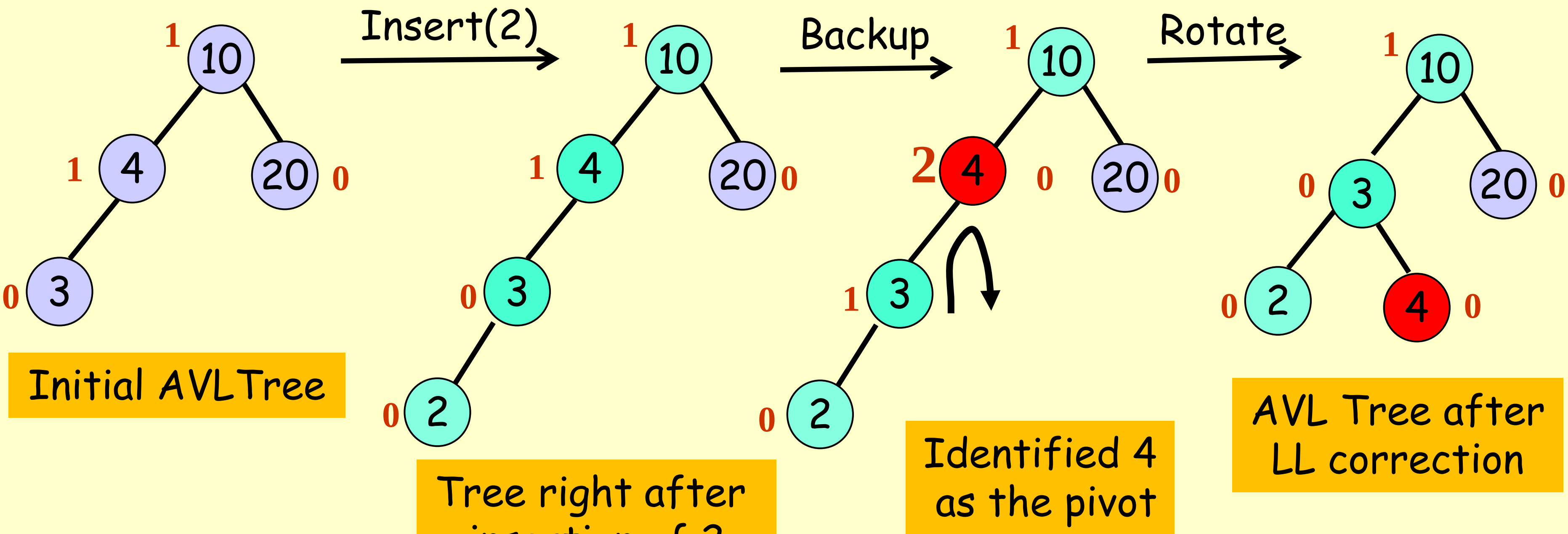
- There are 4 cases:
  - Outside Cases (require single rotation) :
    1. Insertion into left subtree of left child of P (LL Imbalance).
    2. Insertion into right subtree of right child of P (RR Imbalan.)
  - Inside Cases (require double rotation) :
    3. Insertion into left subtree of right child of P (RL Imbalance)
    4. Insertion into right subtree of left child of P (LR Imbalance)

LL Imbalance & Correction



- LL Imbalance: We have inserted into the left subtree of the left child of P (into subtree A)
  - bf of P is 2
  - bf of L is 0 or 1
- Correction: Single rotation to the right around P

LL Imbalance Correction Example (1)

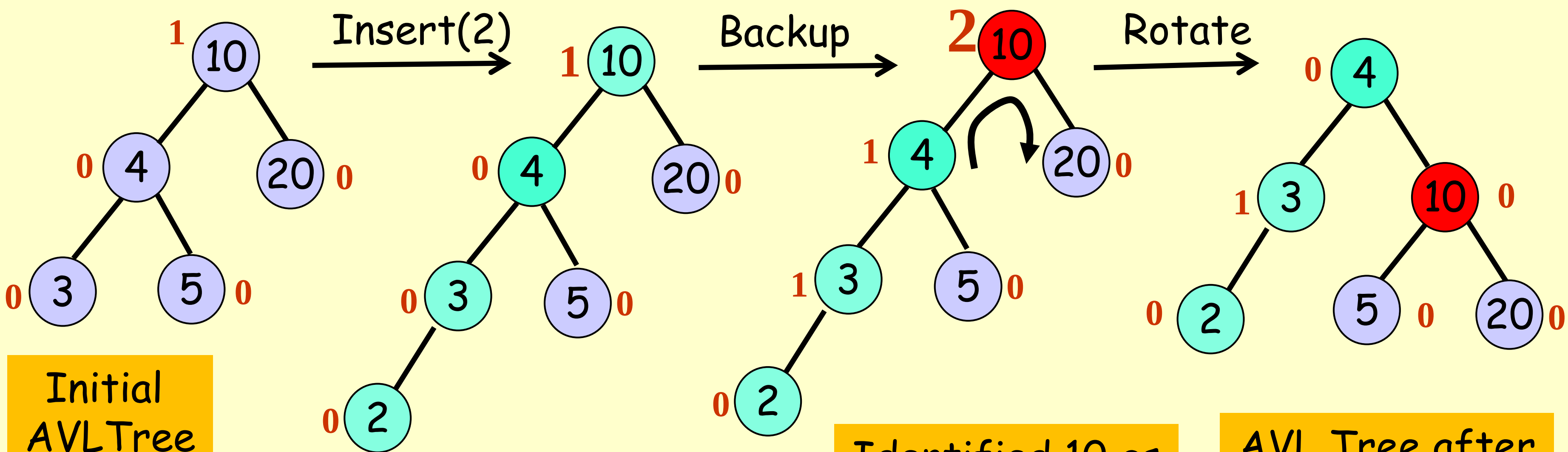


Now, backup the tree updating balance factors

Classify the type of imbalance

- LL Imbalance:
  - bf of P(4) is 2
  - bf of L(3) is 0 or 1

LL Imbalance Correction Example (2)



Initial AVLTree

Tree right after insertion of 2

Identified 10 as the pivot

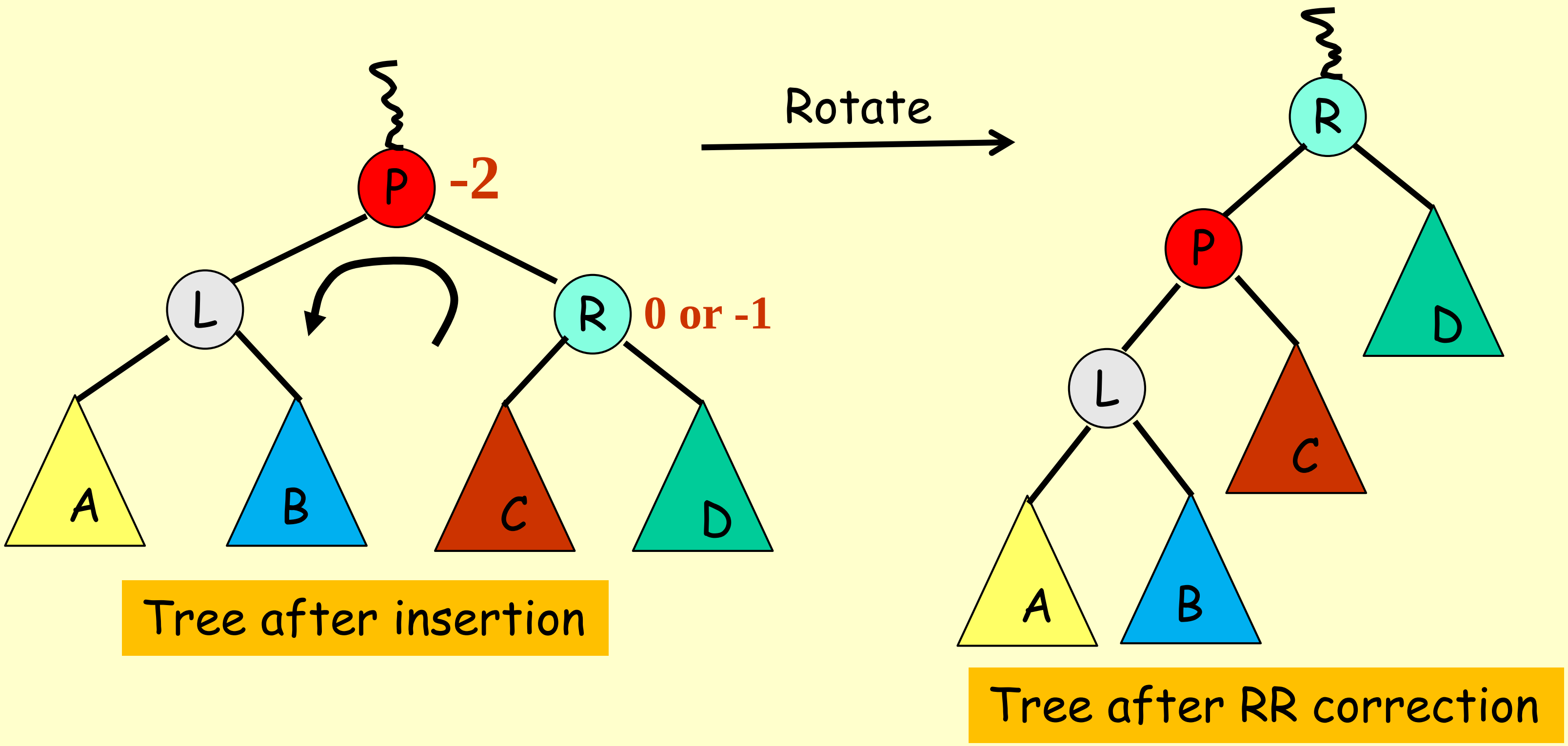
AVL Tree after LL Correction

Now, backup the tree updating balance factors

Classify the type of imbalance

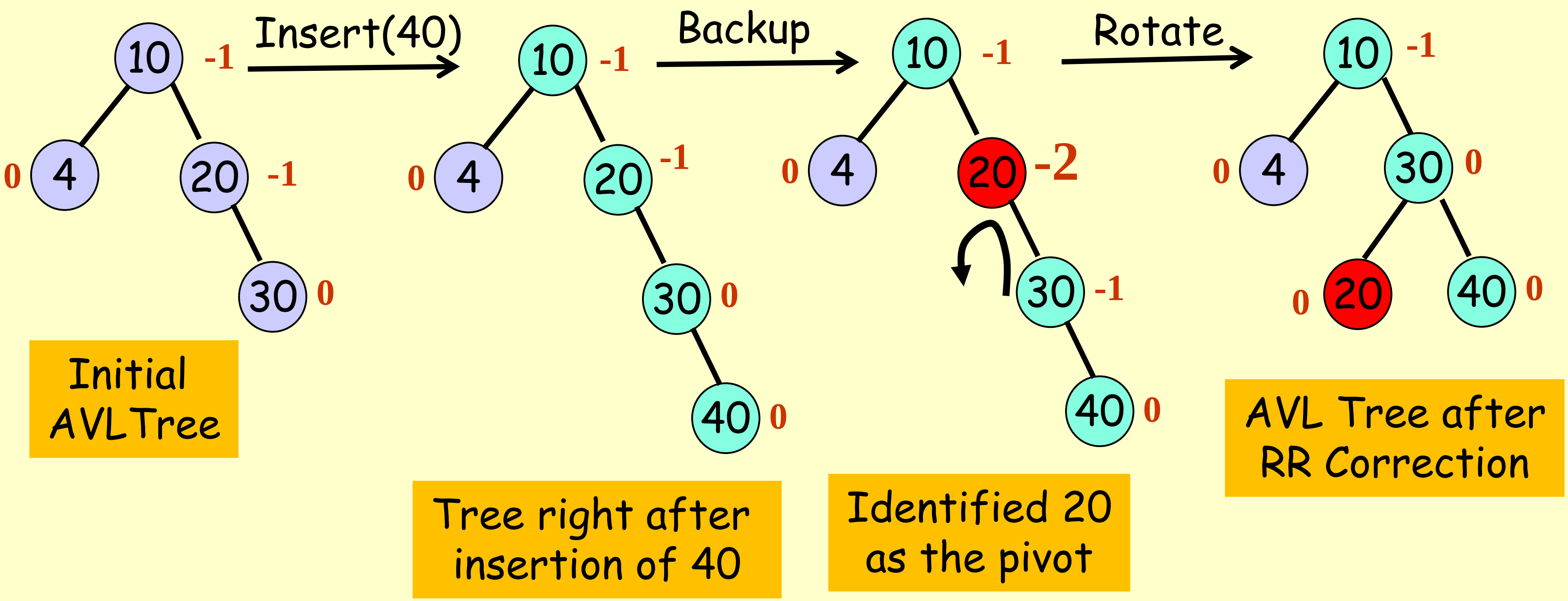
- LL Imbalance:
  - bf of P(10) is 2
  - bf of L(4) is 0 or 1

RR Imbalance & Correction



- RR Imbalance: We have inserted into the right subtree of the right child of P (into subtree D)
  - bf of P is -2
  - bf of R is 0 or -1
- Correction: Single rotation to the left around P

RR Imbalance Correction Example (1)

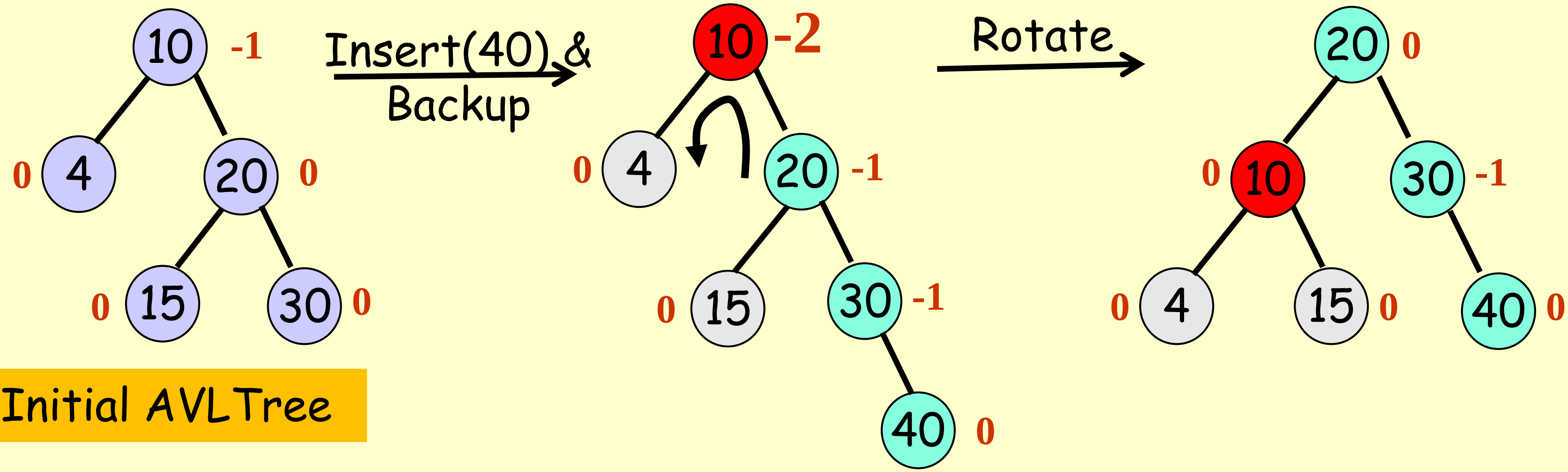


Now, backup the tree updating balance factors

Classify the type of imbalance

- RR Imbalance:
  - bf of P(20) is -2
  - bf of R(30) is 0 or -1

RR Imbalance Correction Example (2)



Initial AVLTree

Tree after insertion of 40

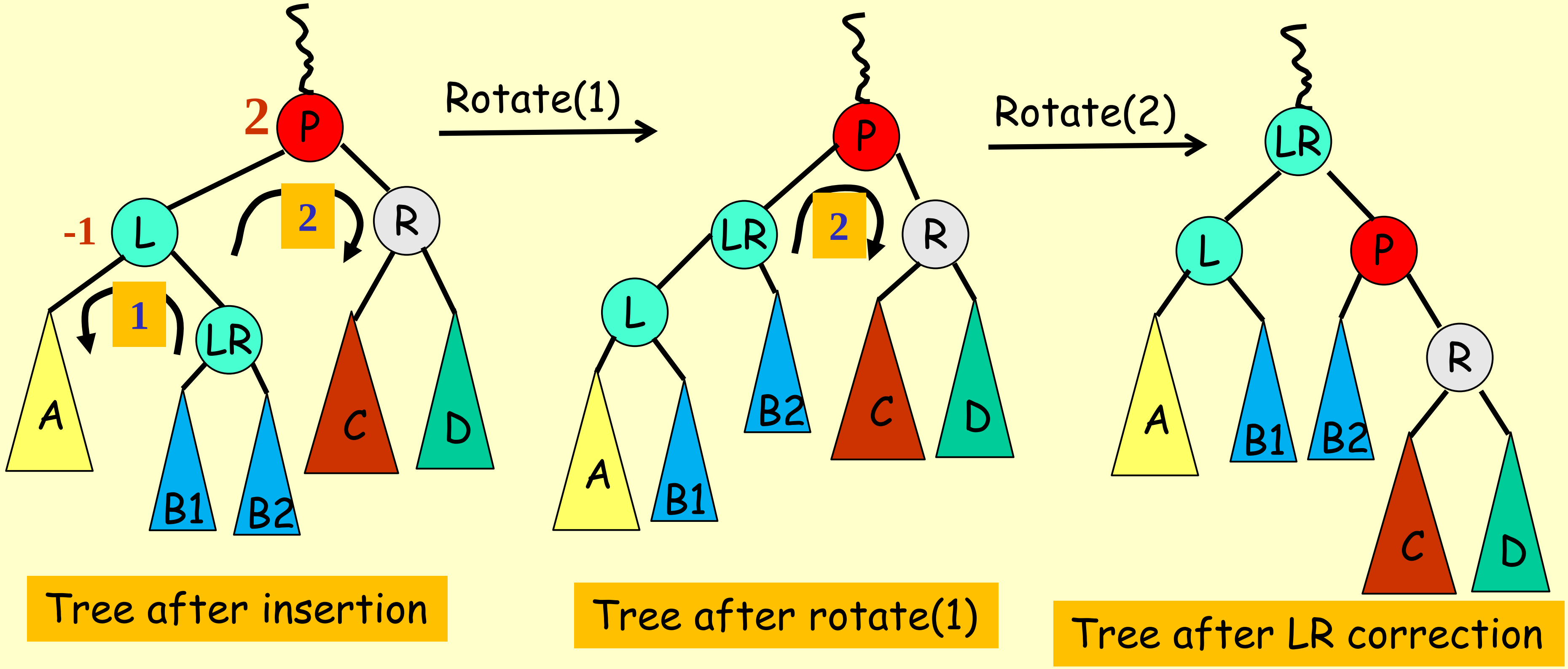
AVL Tree after RR Correction

As we backed up the tree, we updated balance factors and identified 10 as the pivot

Classify the type of imbalance

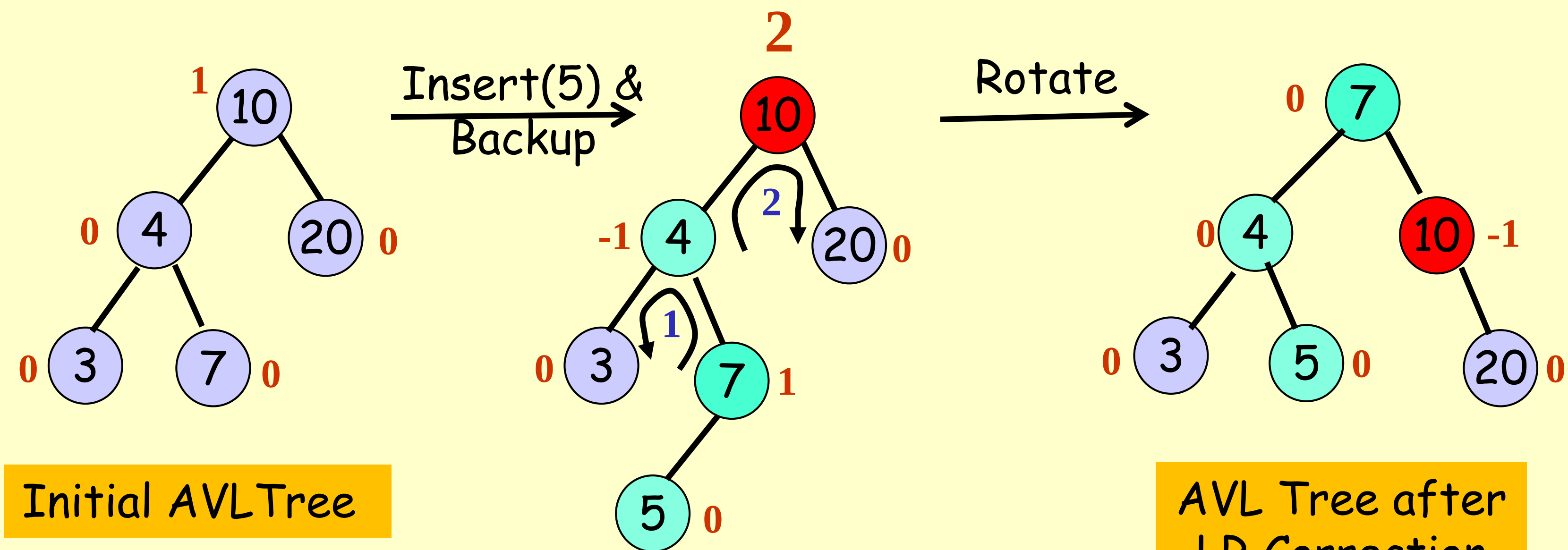
- RR Imbalance:
  - bf of P(10) is -2
  - bf of R(20) is 0 or -1

LR Imbalance & Correction



- LR Imbalance: We have inserted into the right subtree of the left child of P (into subtree LR)
  - bf of P is 2
  - bf of L is -1
- Correction: Double rotation around L & P

LR Imbalance Correction Example



Initial AVLTree

Tree after insertion of 5

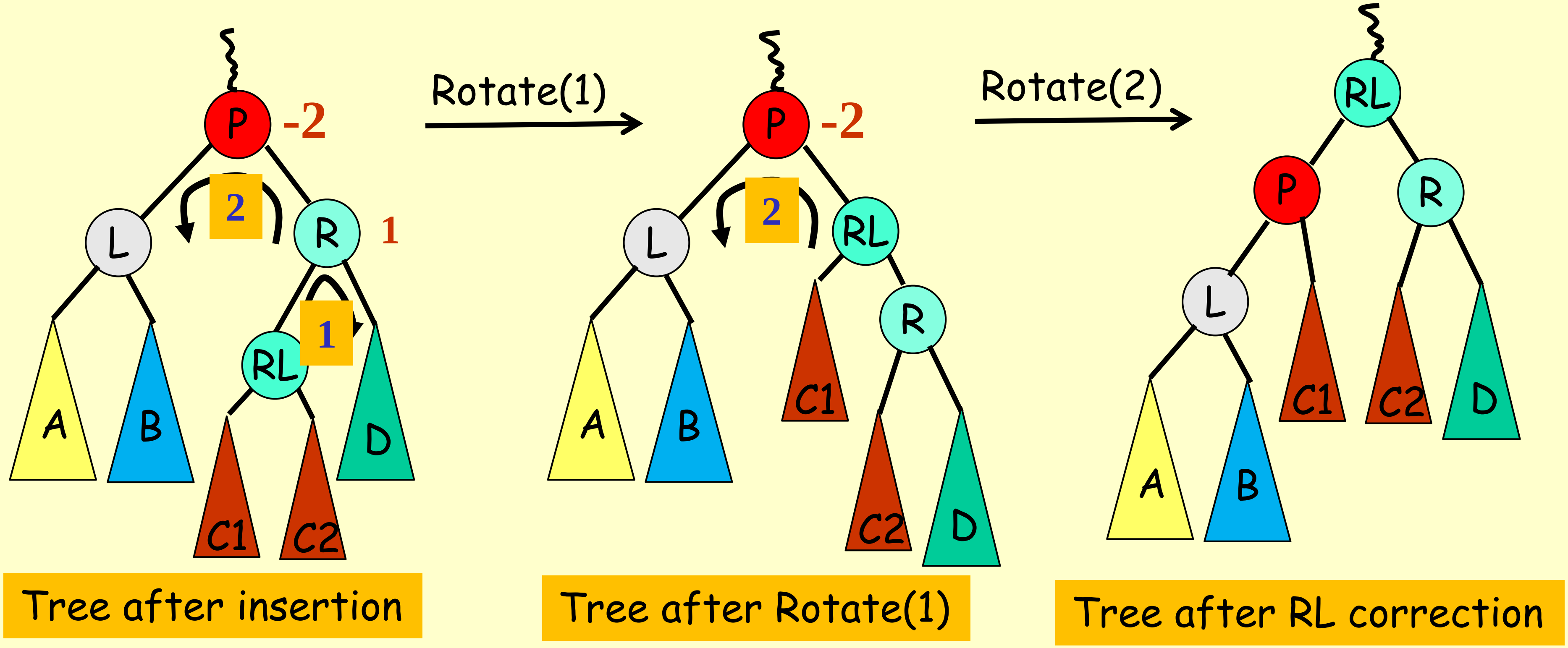
AVL Tree after LR Correction

As we backed up the tree, we updated balance factors and identified 10 as the pivot

Classify the type of imbalance

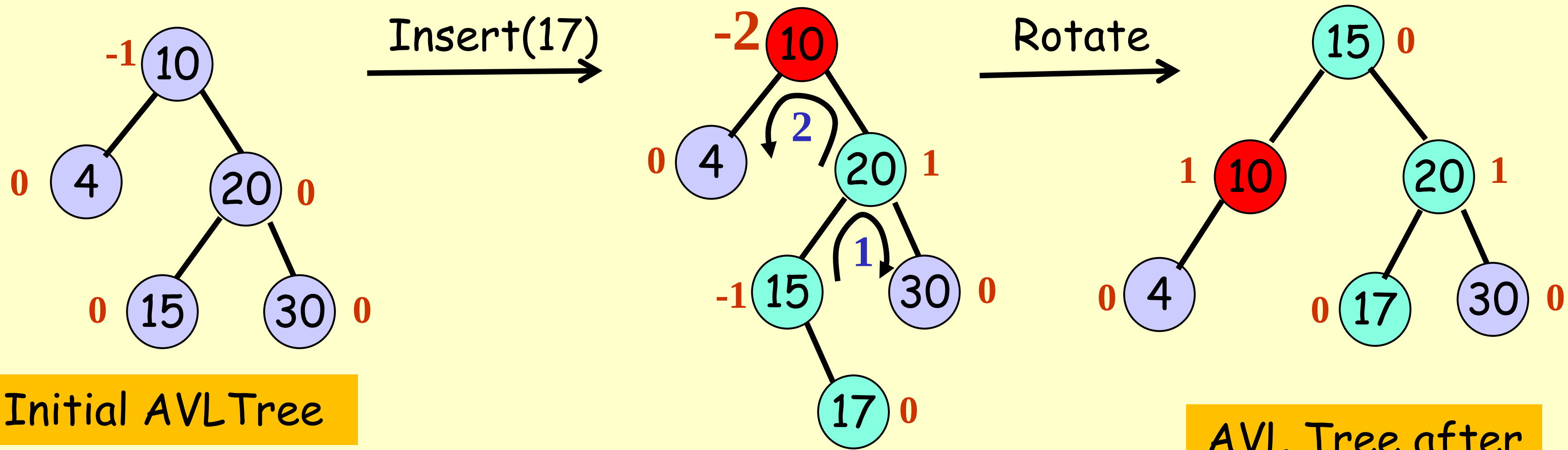
- LR Imbalance:
  - bf of P(10) is 2
  - bf of L(4) is -1

RL Imbalance & Correction



- RL Imbalance: We have inserted into the left subtree of the right child of P (into subtree RL)
  - bf of P is -2
  - bf of R is 1
- Correction: Double rotation around L & P

RL Imbalance Correction Example



Initial AVLTree

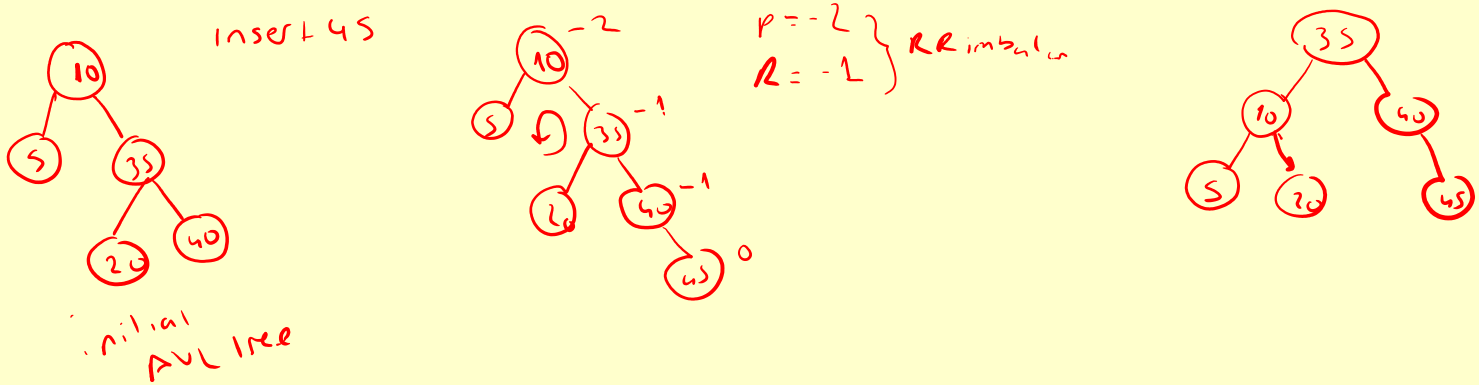
Tree after insertion of 17

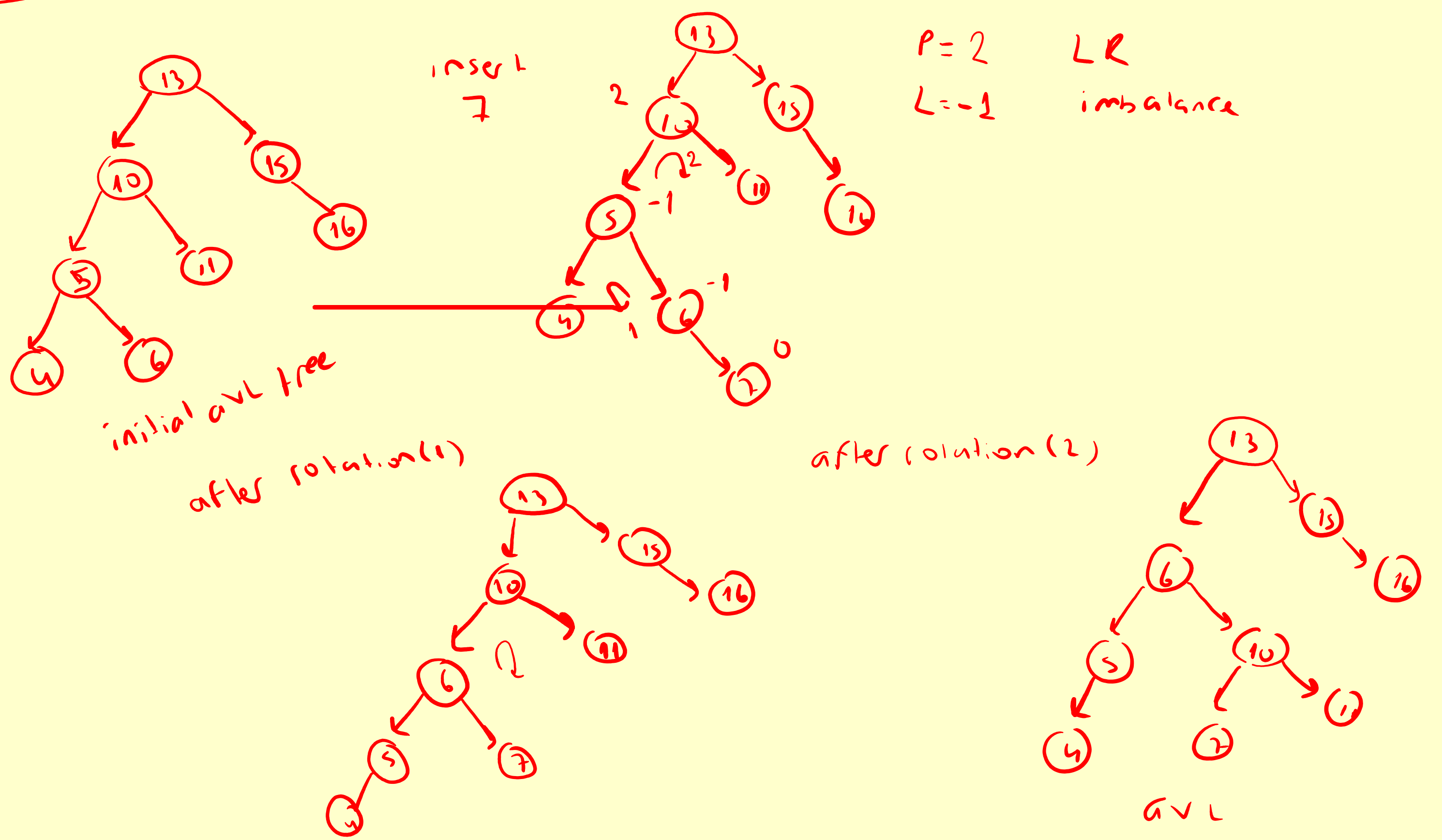
AVL Tree after RL Correction

As we backed up the tree, we updated balance factors and identified 10 as the pivot

Classify the type of imbalance

- RL Imbalance:
  - bf of P(10) is -2
  - bf of R(20) is 1


insert 45
initial AVL tree
P = -2
R = -1
RR imbalance


insert 7
initial avl tree
P = 2
L = -1
LR imbalance
after rotation(1)
after rotation(2)
AVL

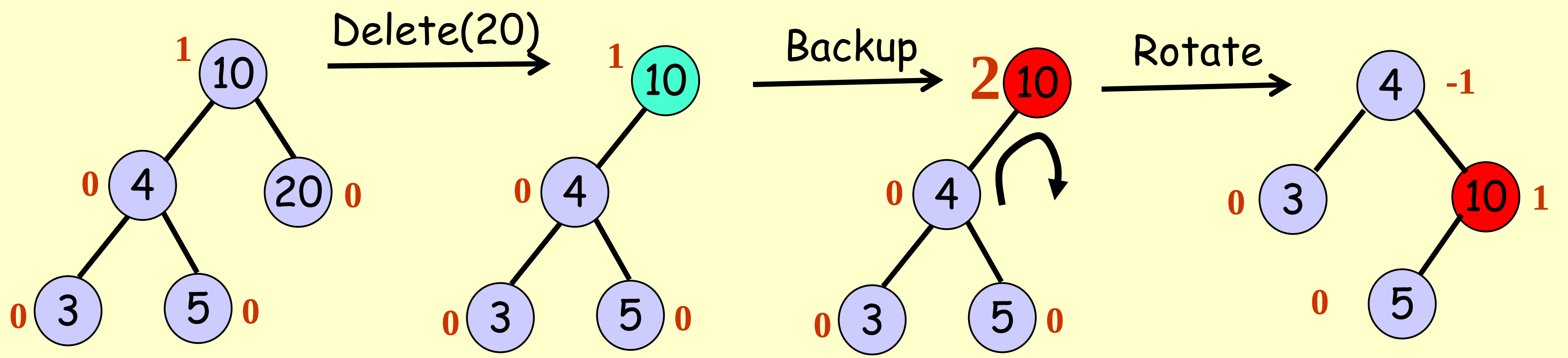
Deletion

- Deletion is similar to insertion

- First do regular BST deletion keeping track of the nodes on the path to the deleted node

- After the node is deleted, simply backup the tree and update balance factors
  - If an imbalance is detected, do the appropriate rotation to restore the AVL tree property
  - You may have to do more than one rotation as you backup the tree

Deletion Example (1)



Initial AVLTree

Tree after deletion of 20

Idenfitied 10 as the pivot

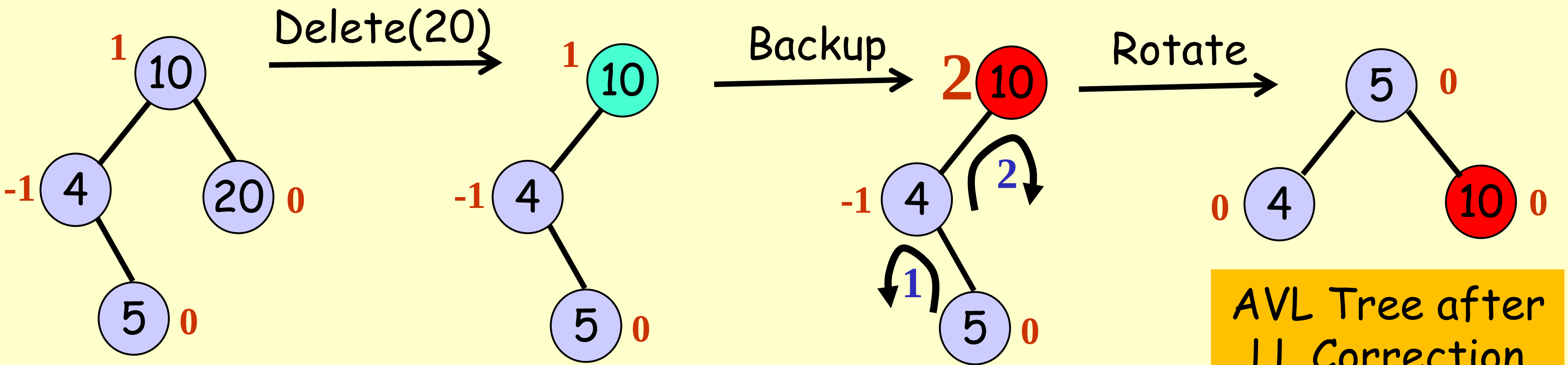
AVL Tree after LL Correction

Now, backup the tree updating balance factors

Classify the type of imbalance

- LL Imbalance:
  - bf of P(10) is 2
  - bf of L(4) is 0 or 1

Deletion Example (2)



Initial AVLTree

Tree after deletion of 20

Idenfitied 10 as the pivot

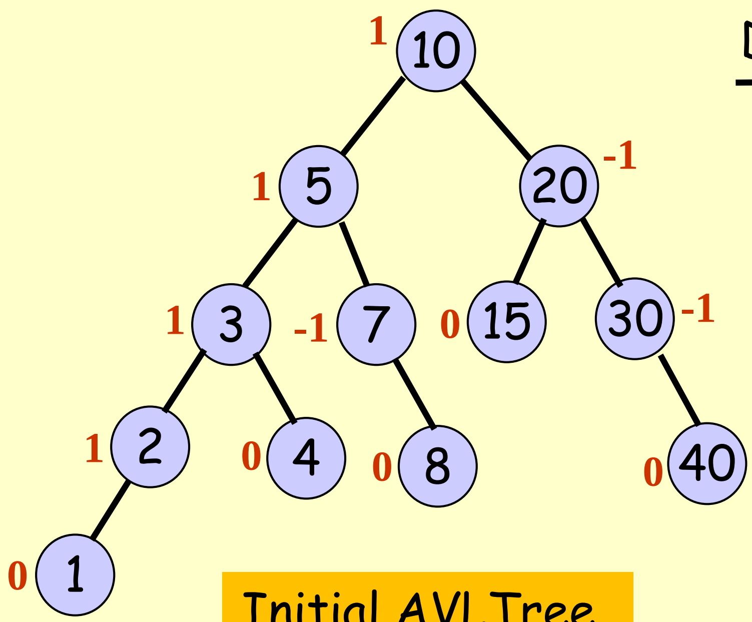
AVL Tree after LL Correction

Now, backup the tree updating balance factors

Classify the type of imbalance

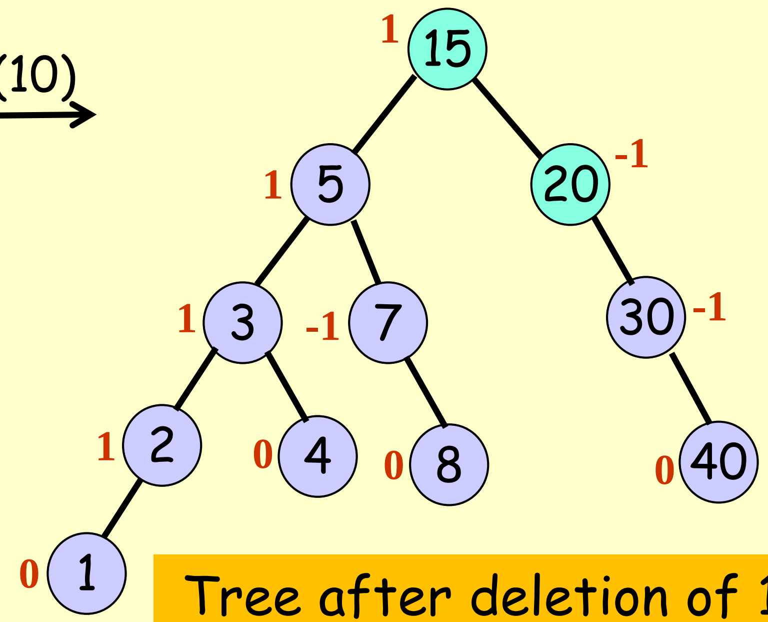
- LR Imbalance:
  - bf of P(10) is 2
  - bf of L(4) is -1

Deletion Example (3)



Initial AVLTree

Delete(10)

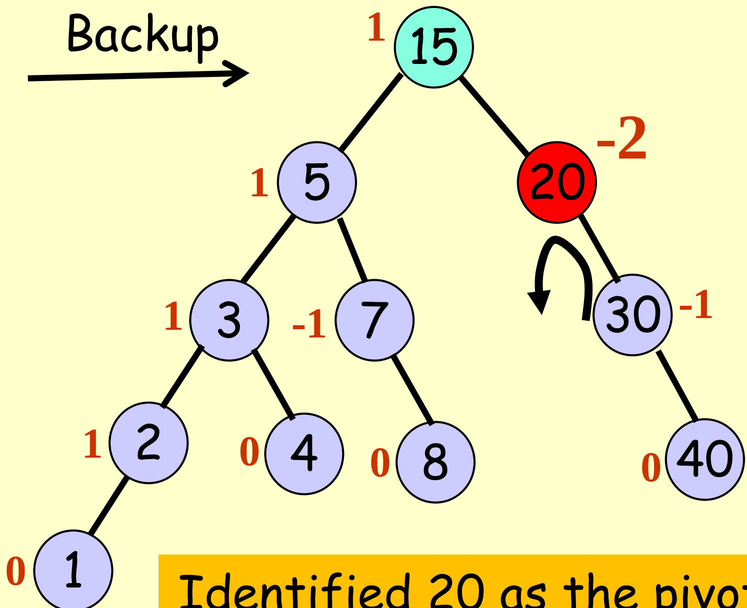


Tree after deletion of 10

We have copied the successor of 10 to root and deleted 10

Now, backup the tree updating balance factors

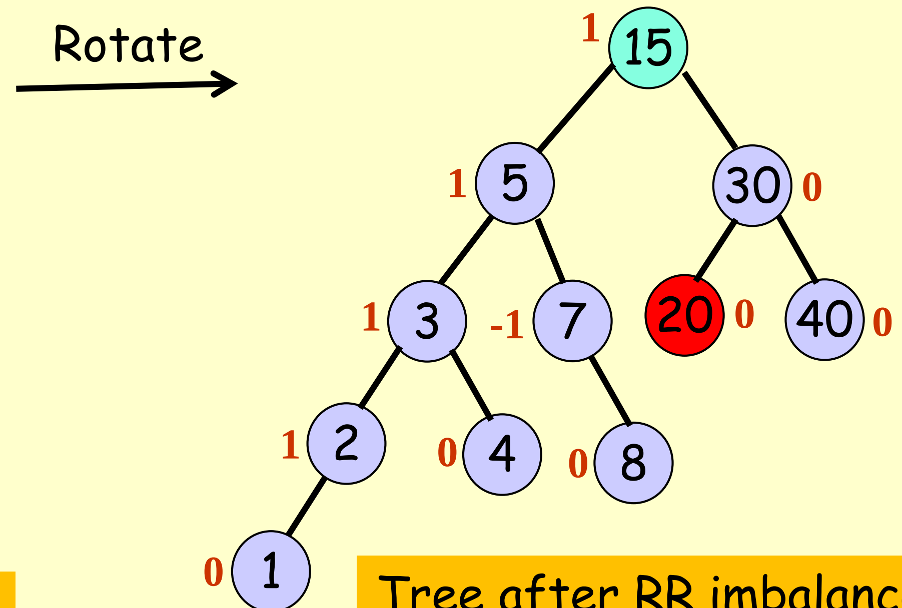
Deletion Example (3) - continued



Identified 20 as the pivot

Classify the type of imbalance

- RR Imbalance:
  - bf of P(20) is -2
  - bf of R(30) is 0 or -1

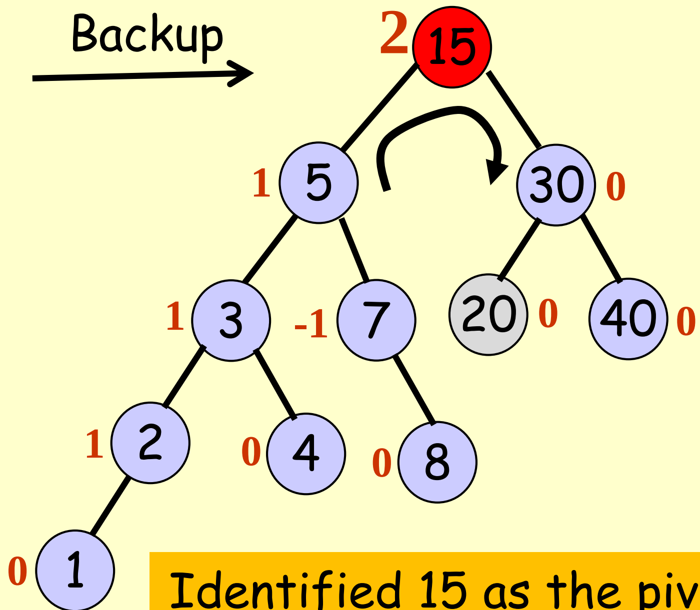


Tree after RR imbalance is corrected

Is this an AVL tree?

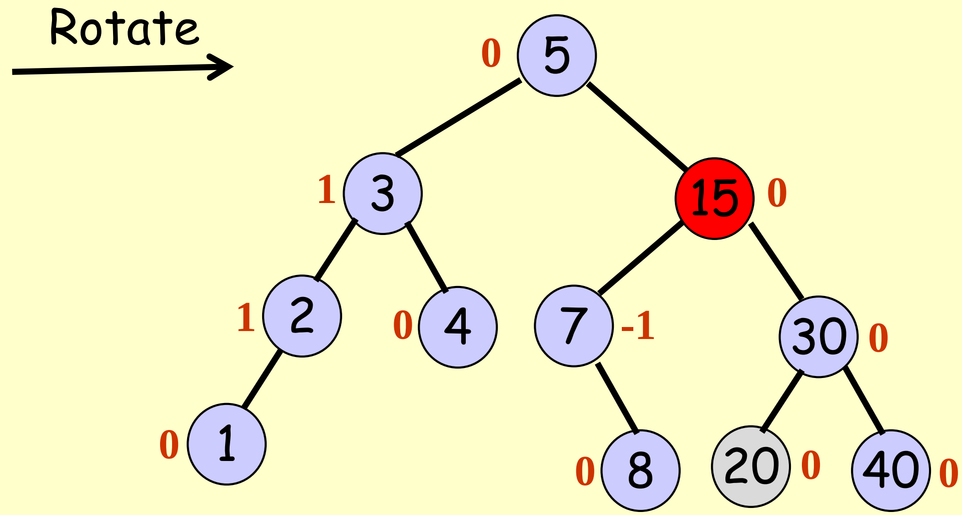
Continue backing up the tree updating balance factors

Deletion Example (3) - continued



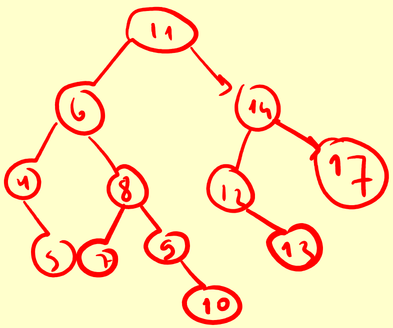
Identified 15 as the pivot

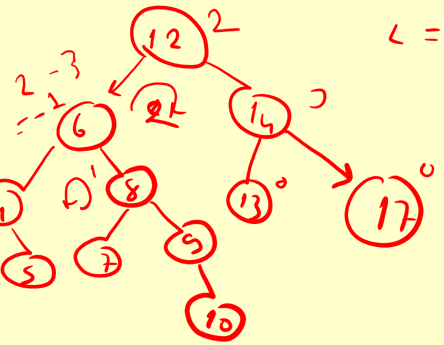
Classify the type of imbalance



Final Tree

- LL Imbalance:
  - bf of P(15) is 2
  - bf of L(5) is 0 or 1



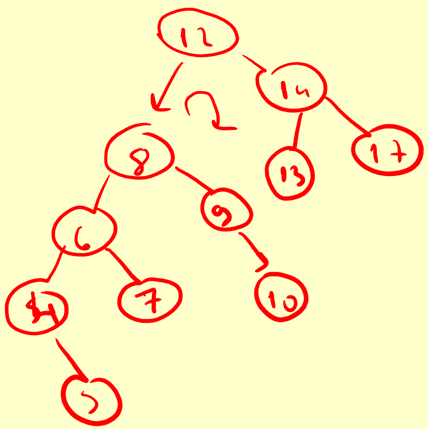


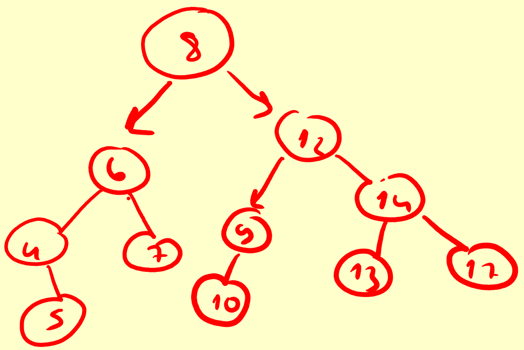
P = 2   LR

L = -1

initial avl tree

after rotation(1)





Search (Find)

- Since AVL Tree is a BST, search algorithm is the same as BST search and runs in guaranteed O(logn) time

Summary of AVL Trees

- Arguments for using AVL trees:
  1. Search/insertion/deletion is O(log N) since AVL trees are always balanced.
  2. The height balancing adds no more than a constant factor to the speed of insertion/deletion.

- Arguments against using AVL trees:
  1. Requires extra space for balancing factor (height)
  2. It may be OK to have a partially balanced tree that would give performance similar to AVL trees without requiring the balancing factor
     - Splay trees