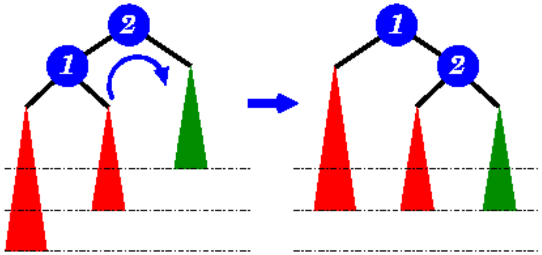
Data Structures and Algorithms

AVL TREES

Nguyễn Ngọc Thảo
nnthao@fit.hcmus.edu.vn

Ho Chi Minh City, 05/2022

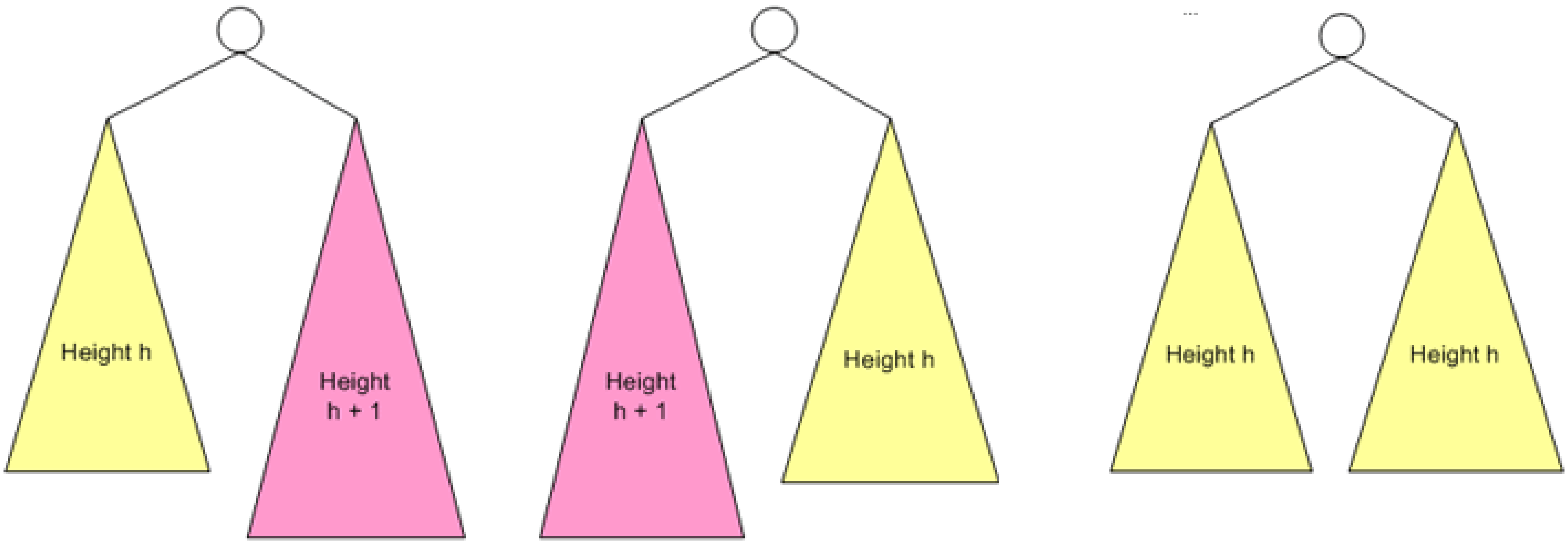
AVL Trees



- *A definition of AVL tree*
- *Inserting data into an AVL tree*
- *Removing data from an AVL tree*
- *Single rotations and Double rotations*

A definition of AVL tree

- Proposed in 1962 by G. M. Adelson-Velskii and E. M. Landis
- An AVL tree is a balanced binary search tree.
  - The heights of the left and right subtrees of any node in a balanced binary tree differ by no more than 1.

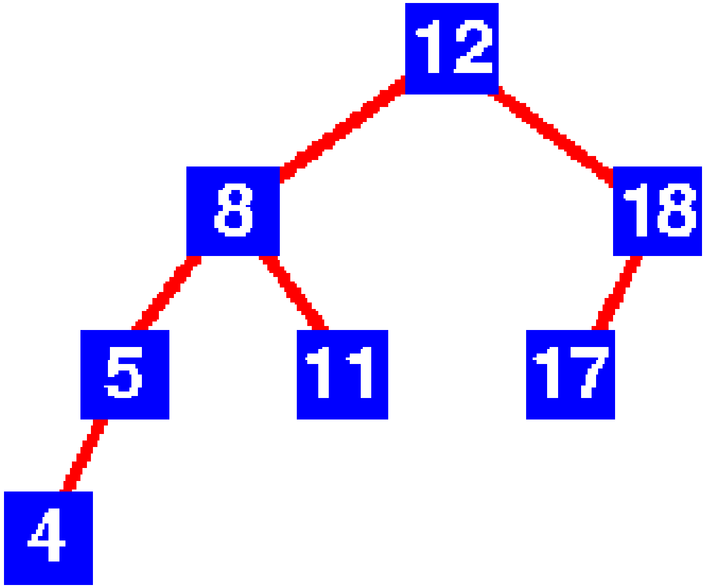


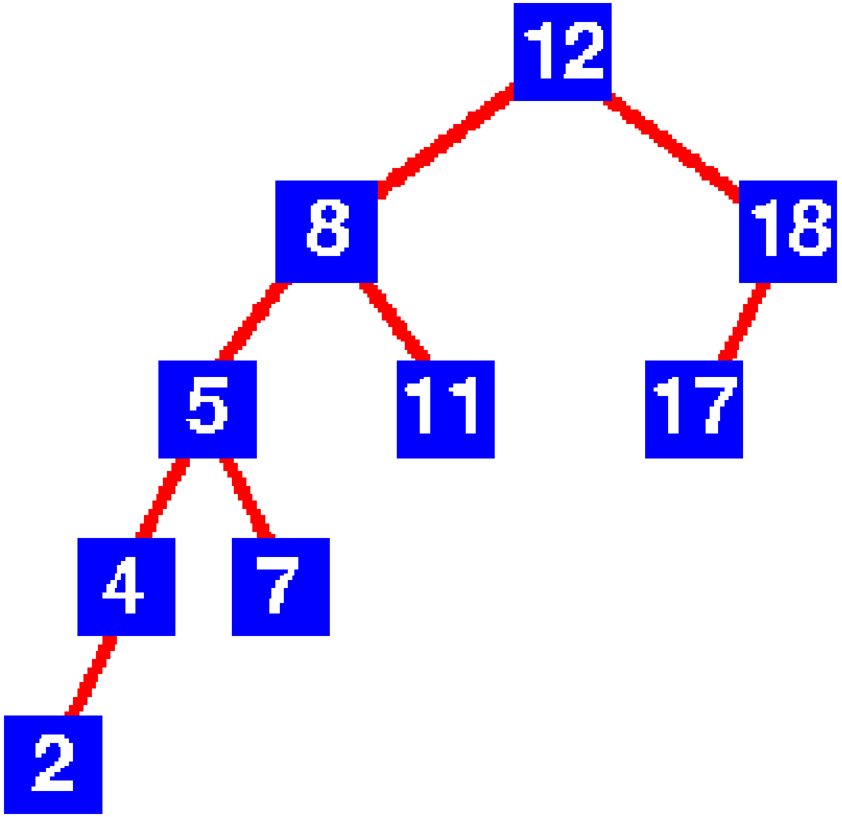
A definition of AVL tree

- The height of an AVL tree with n nodes will always be very close to the theoretical minimum of $\lceil \log_2(n+1) \rceil$
- Searching an AVL tree is almost as efficiently as searching a minimum-height BST.

Checkpoint 01: Represent a BST with an array

Which of the following trees is an AVL tree?





AVL tree implementation

- The implementation of an AVL node is quite similar to that of a BST node → Make it inherited from the class BinaryNode

```
class AVLNode{
  private:
    ItemType item;                  // Data portion
    AVLNode* leftChildPtr;          // Left-child pointer
    AVLNode* rightChildPtr;         // Right-child pointer
    int balanceFactor;              // Balance factor (-1,0,1)
  public:
    …
} // end AVLNode
```

AVL tree implementation

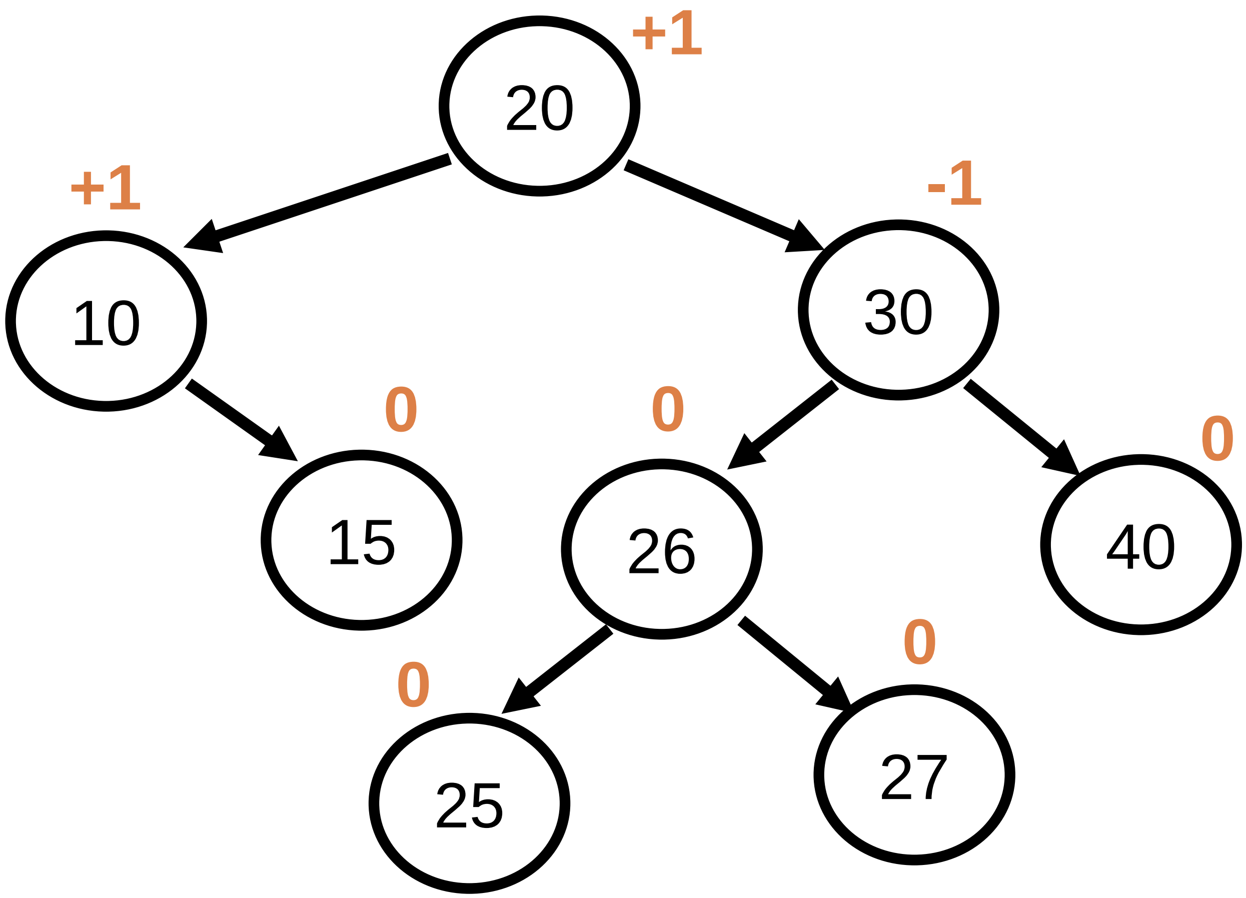
- The implementation of an AVL tree can also be inherited from the class BinaryTree

```
class AVLTree{
  private:
    unsigned int count;              // Number of nodes
    AVLNode* rootPtr;                // Pointer to the root
    // Internal operations: single rotations and double rotations
    …
  public:
    // Common operations: search traverse. Insert, delete, etc.
    …
} // end AVLNode
```

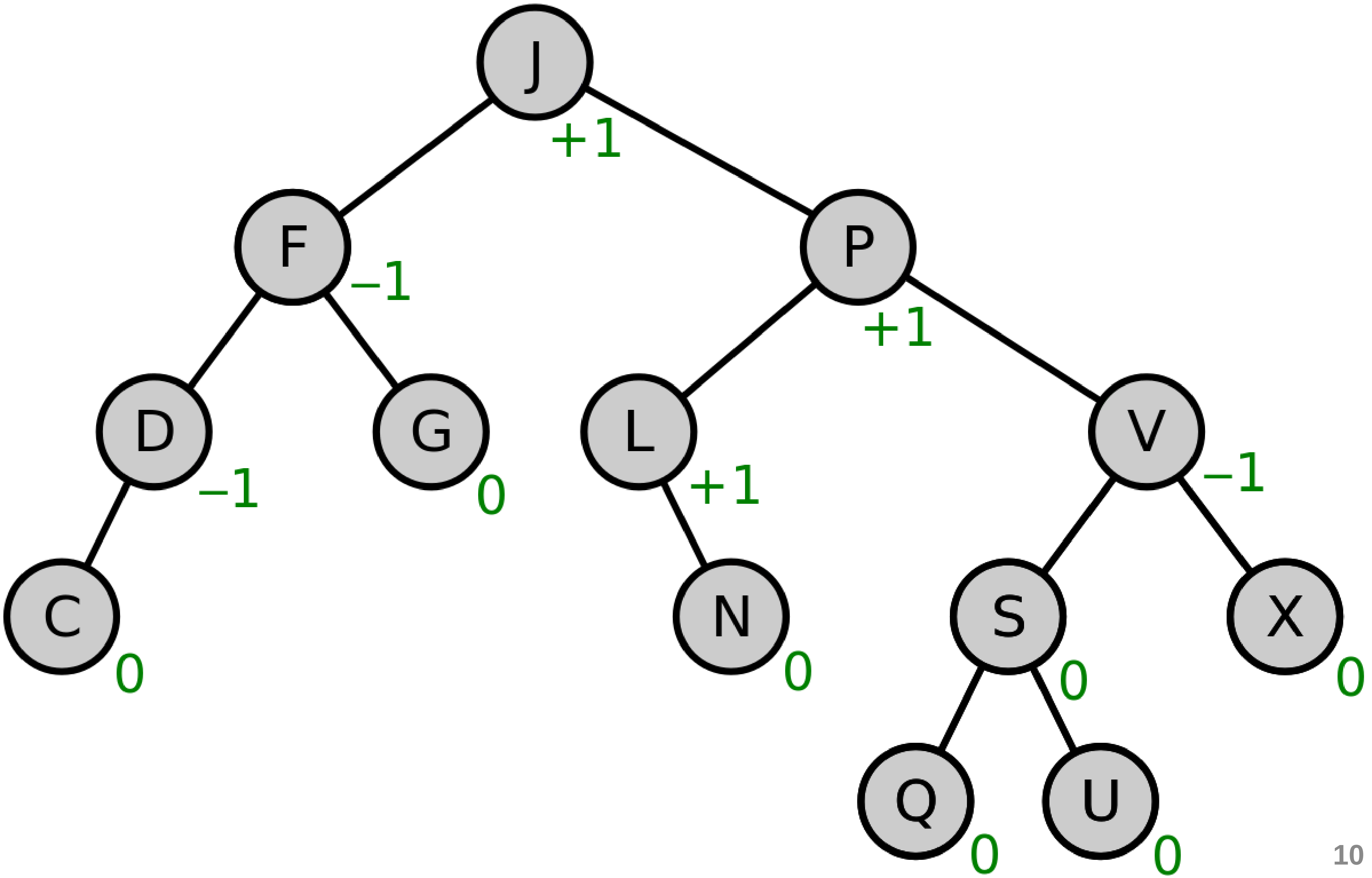
AVL tree implementation

- The **balanceFactor** of each node represents the relation between its two subtrees
- **balanceFactor** can have a value of $-1, 0$ or 1
  - -1 – left-heavy: the left subtree is higher than the right subtree
  - 0 – balanced: both subtrees has the same heights
  - 1 – right-heavy: the left subtree is lower than the right subtree

Balance factor: An example

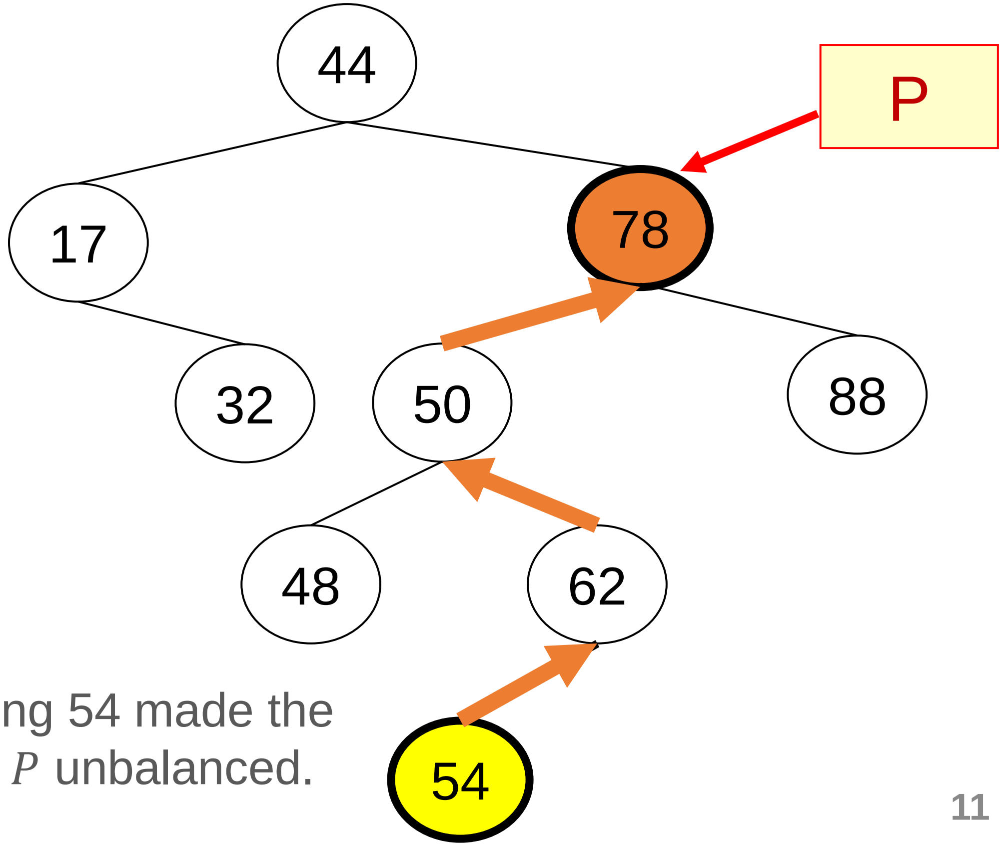


Balance factor: Another example



Inserting data into an AVL tree

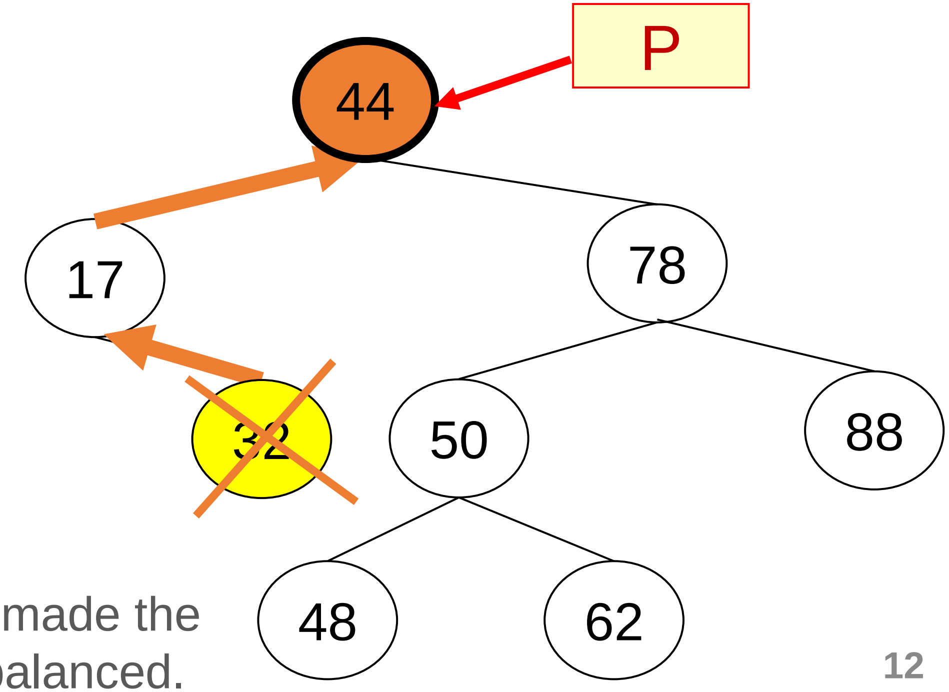
- Check the tree after each insertion if it is still an AVL tree.
- Trace back from the inserted node to the root
  - If a node P is found to be unbalance, perform a rotation at P
  - A single action is sufficient.



Inserting 54 made the node P unbalanced.

Deleting data from an AVL tree

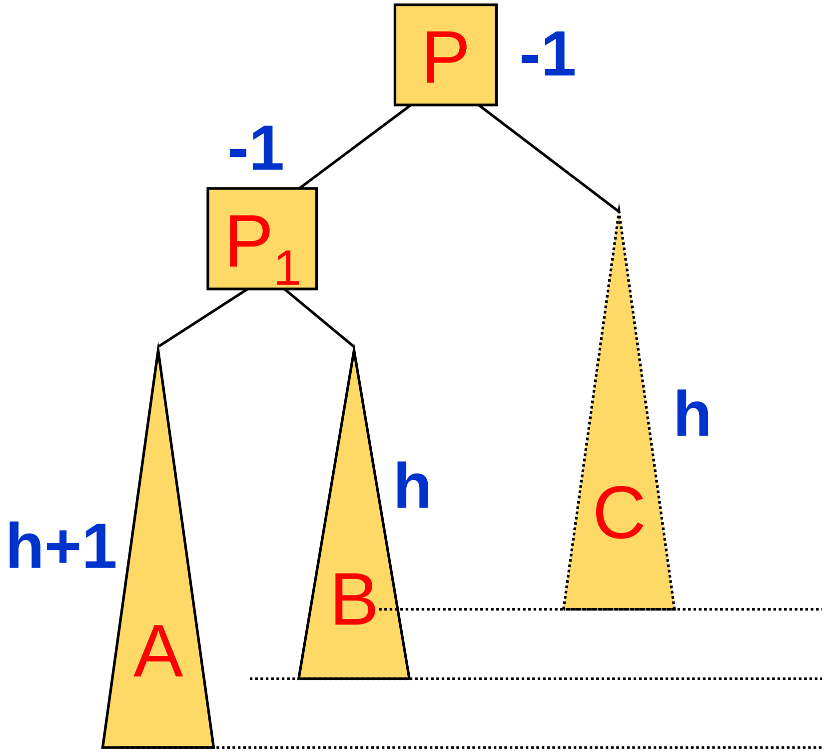
- Check the tree after each insertion if it is still an AVL tree.
- Trace back from the inserted node to the root
  - If a node P is found to be unbalance, perform a rotation at P $\rightarrow$ Ancestors of P may become unbalanced after the rotation.
  - Keep adjusting until no node is unbalanced.



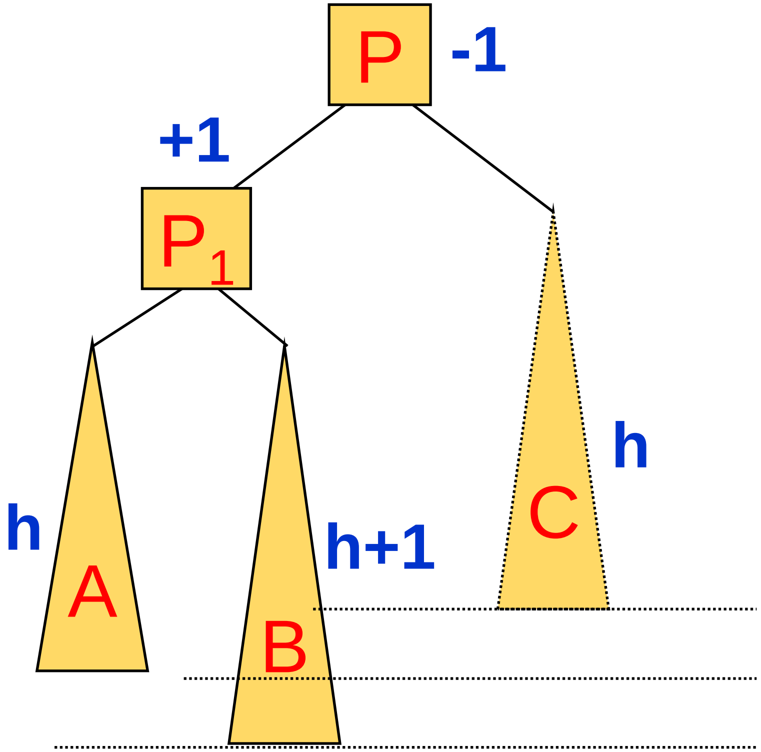
Deleting 32 made the node P unbalanced.

Single rotations and Double rotations

- The AVL tree is unbalanced with a higher left subtree.



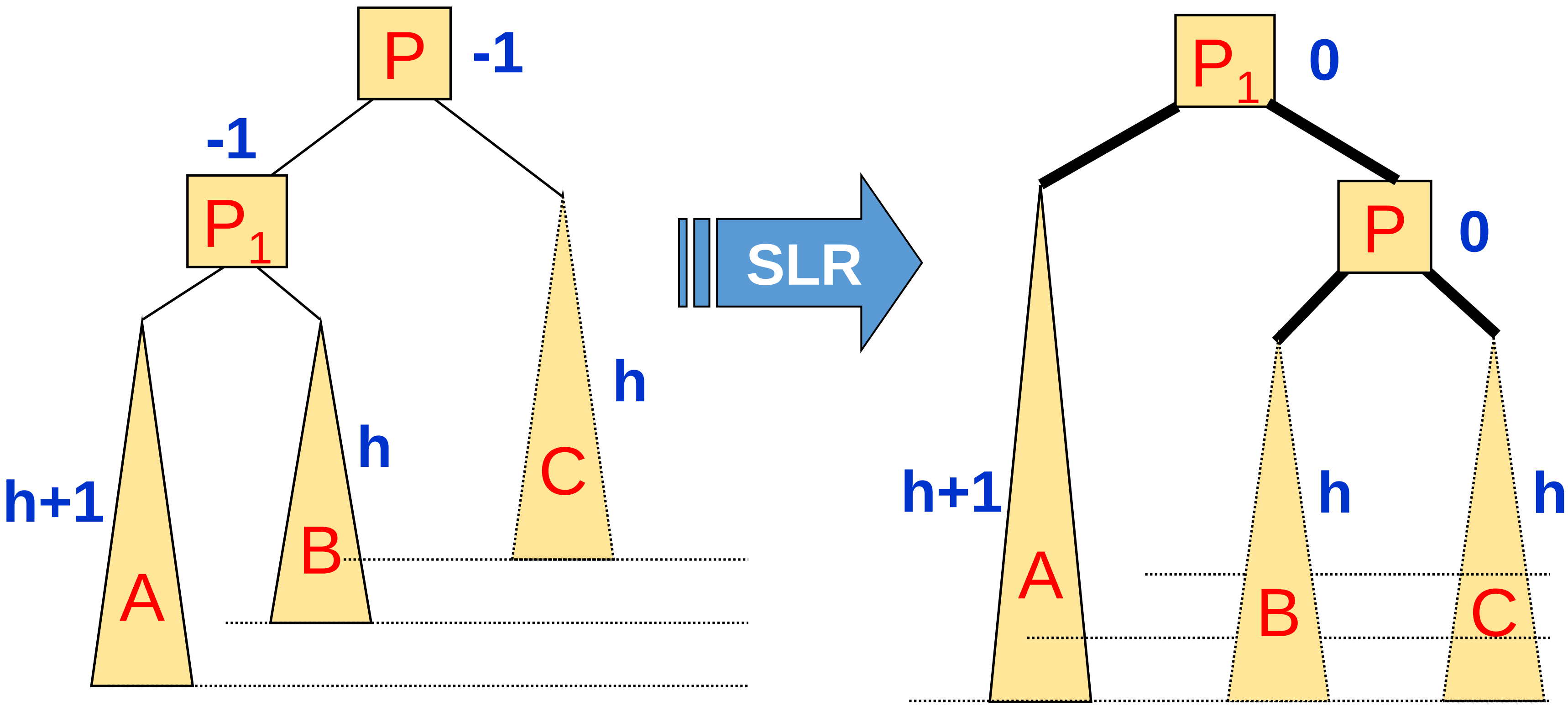
(a1)



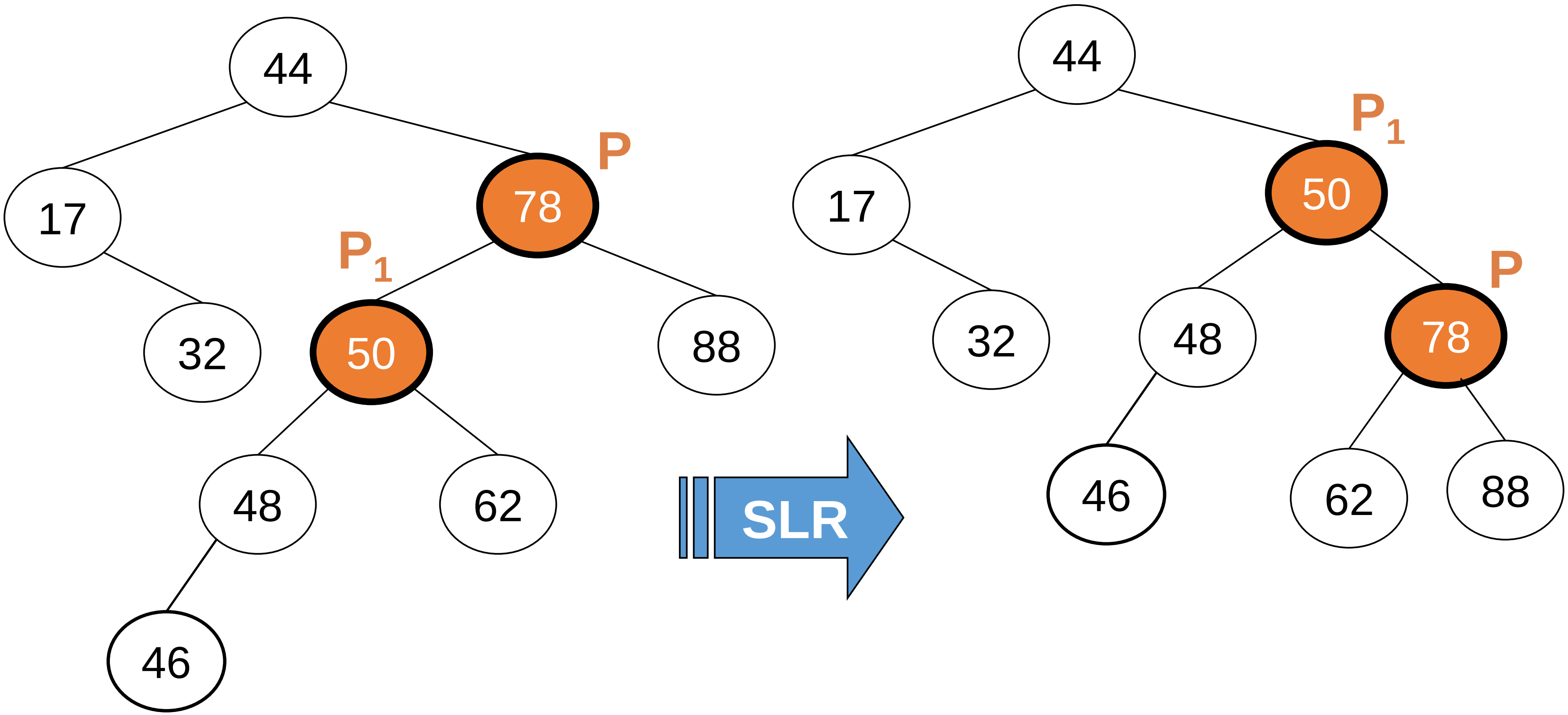
(b1)

Single rotations and Double rotations

- **Case (a1):** Apply a single rotation Single Left-to-Right (SLR)

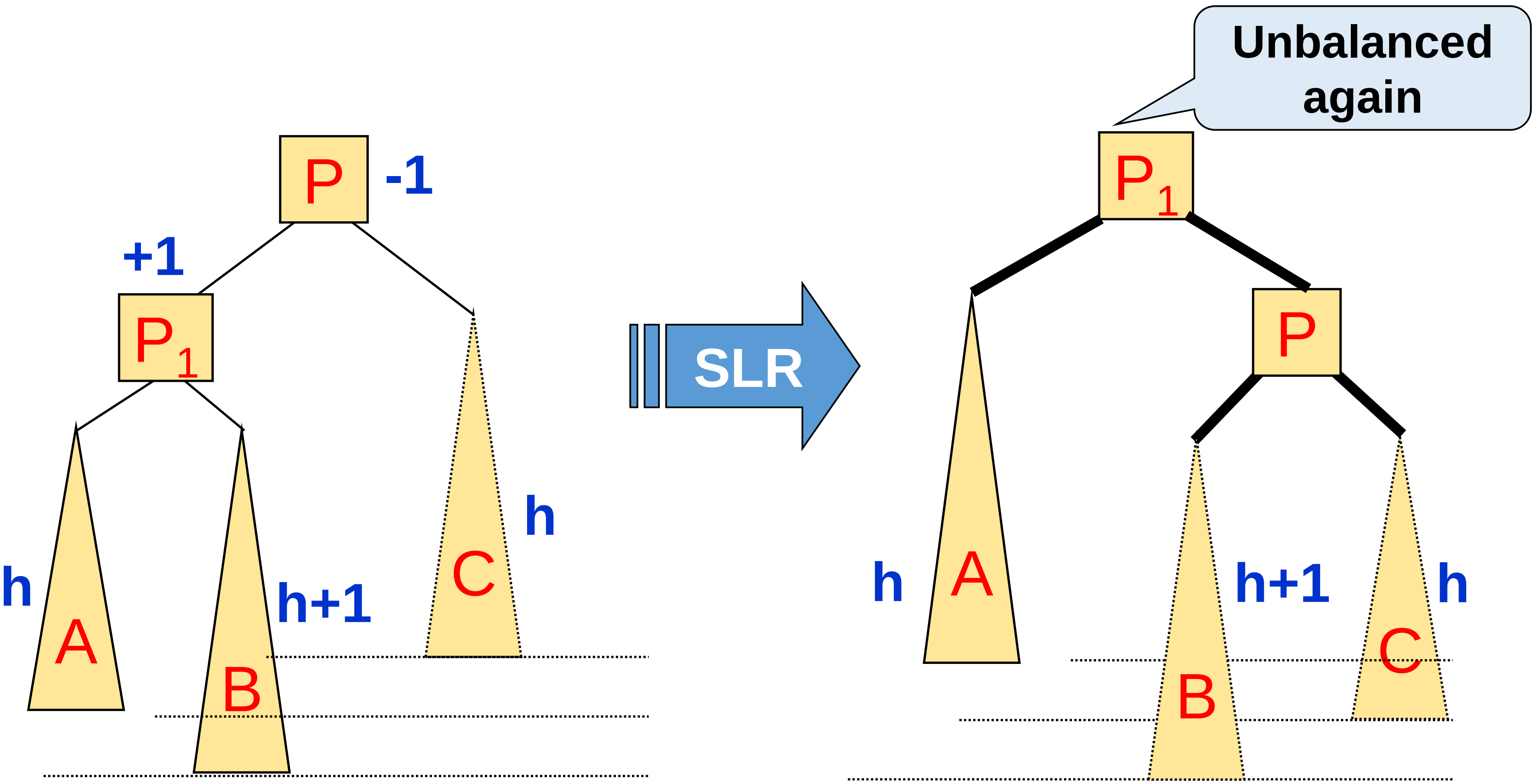


Single Left-to-Right: An example



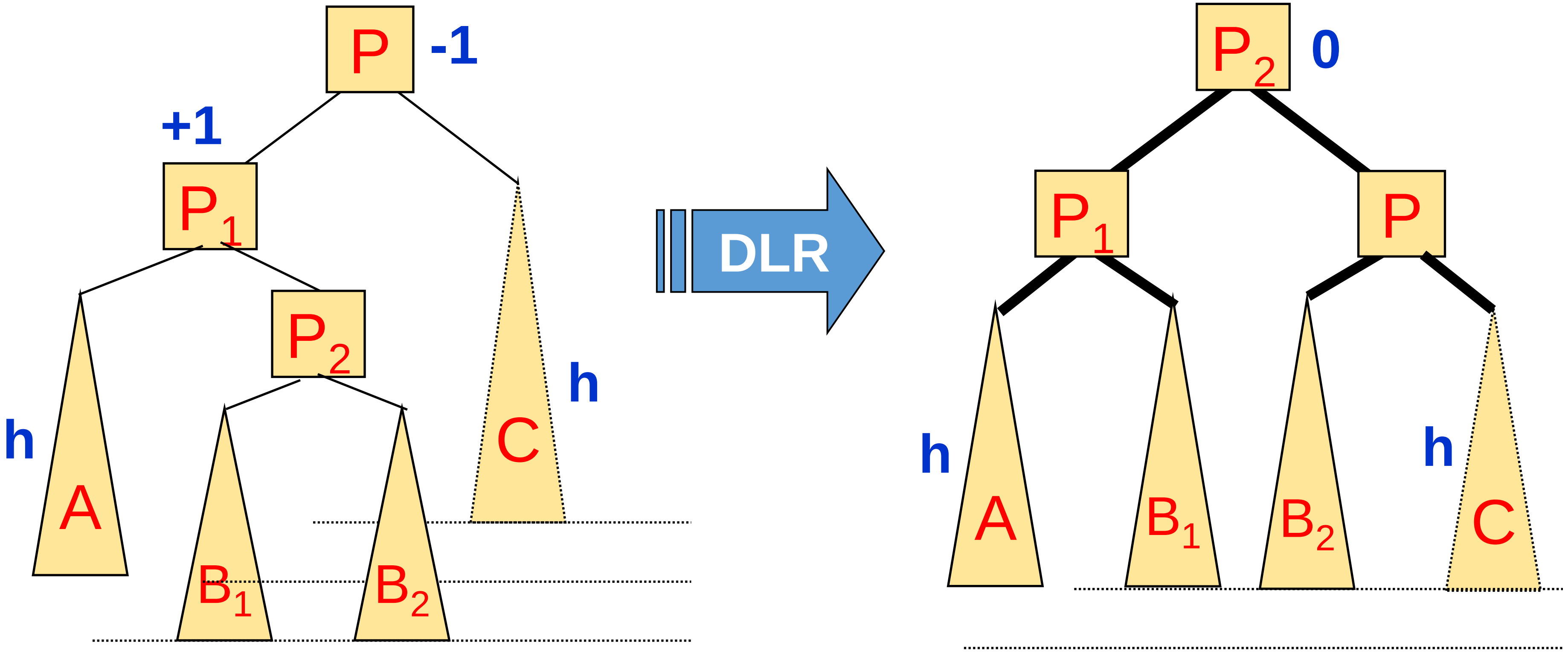
Single rotations and Double rotations

- **Case (b1):** **CANNOT** apply a single rotation SLR

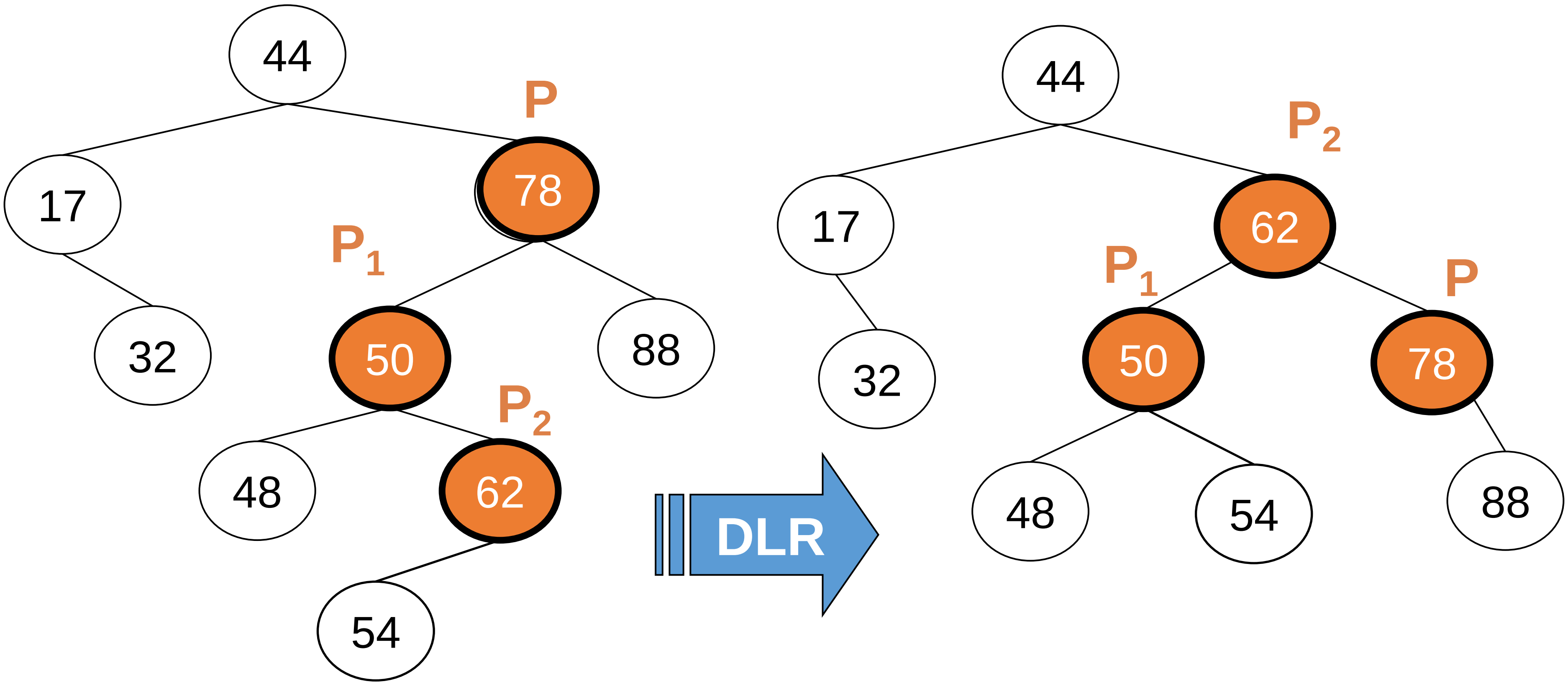


Single rotations and Double rotations

- **Case (b1):** Apply a double rotation Double Left-to-Right (DLR)

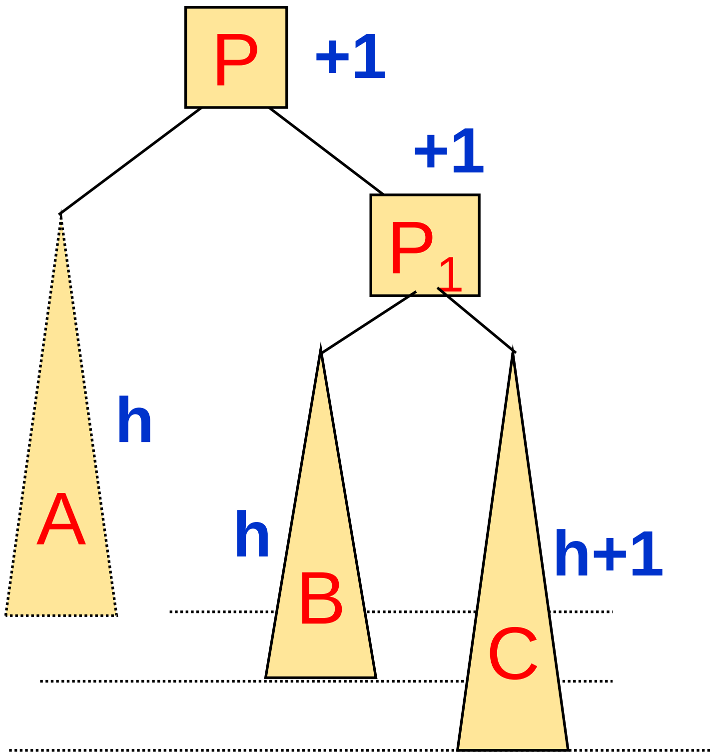


Double Left-to-Right: An example

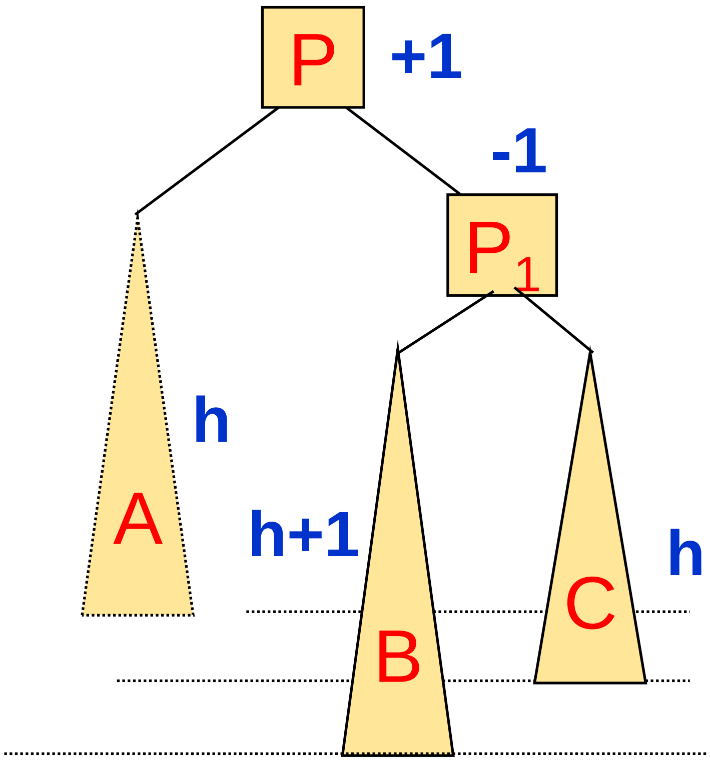


Single rotations and Double rotations

- The AVL tree is unbalanced with a higher right subtree.



(a2)



(b2)

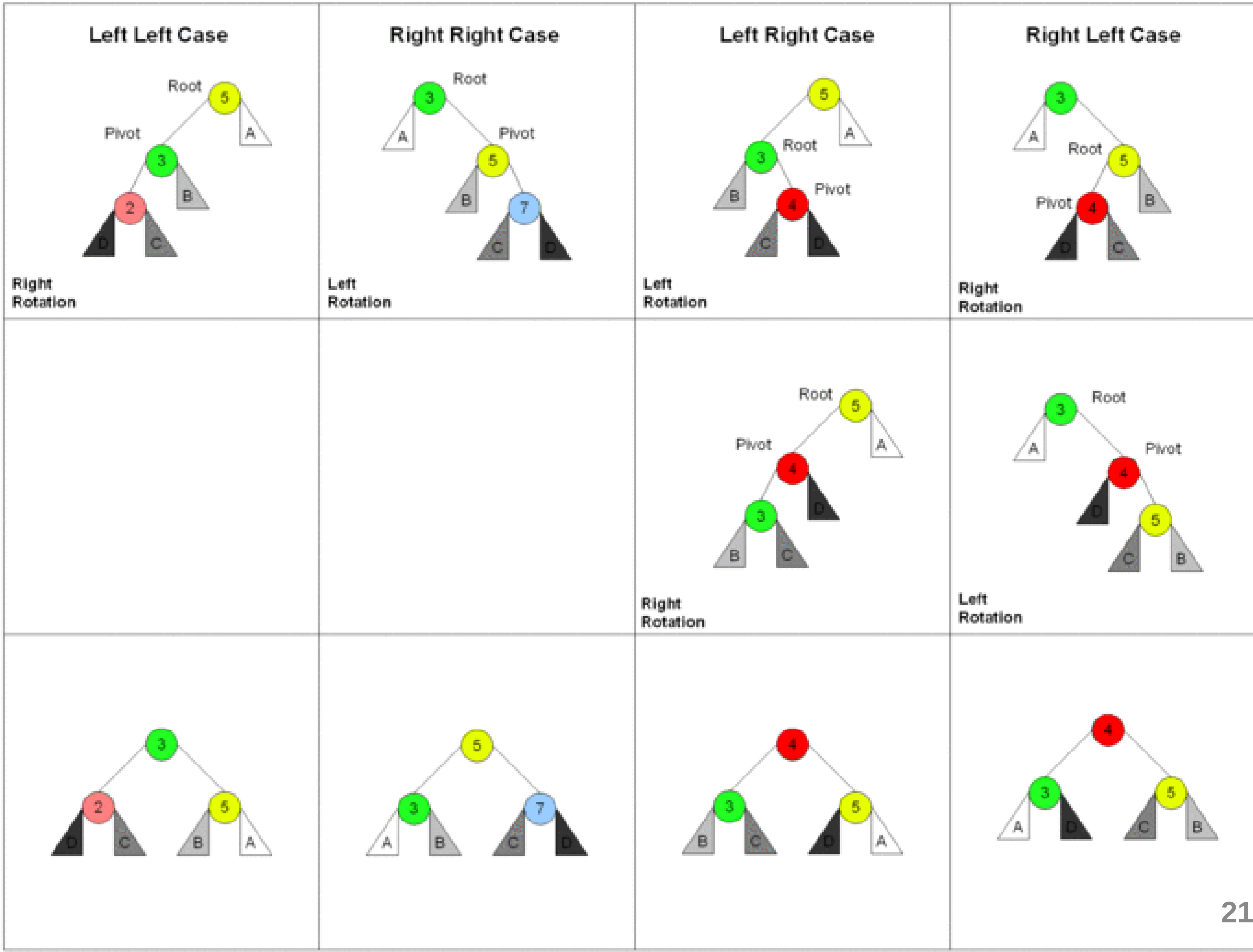
Single rotations and Double rotations

- **Cases (a2) and (b2):** Similar to the cases (a1) and (b1) but symmetric across the vertical axis
- Apply a single rotation Single Right-to-Left (SRL) for (a2)
- Apply a double rotation Double Right-to-Left (DRL) for (b2)

Single rotations and Double rotations

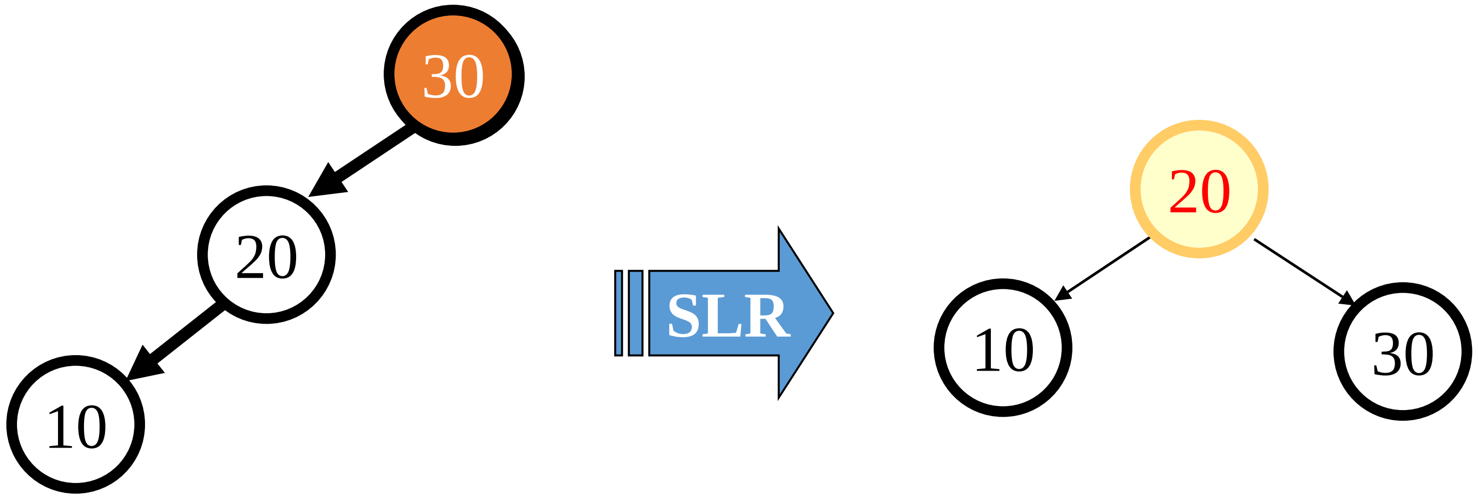
There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

Root is the initial parent before a rotation and **Pivot** is the child to take the root's place.



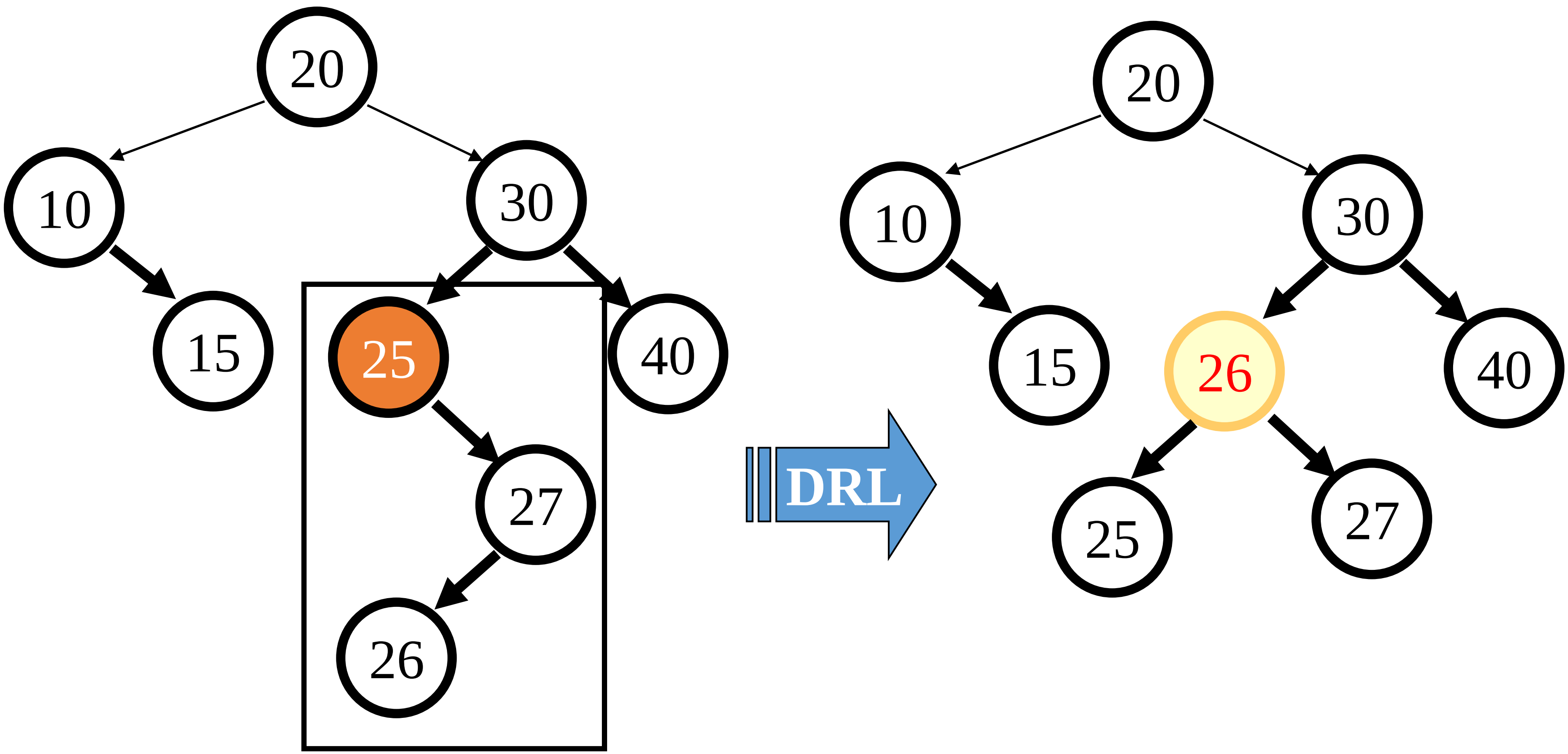
Inserting data to AVL: An example

- **Insert 30, 20, and 10** into an empty AVL tree



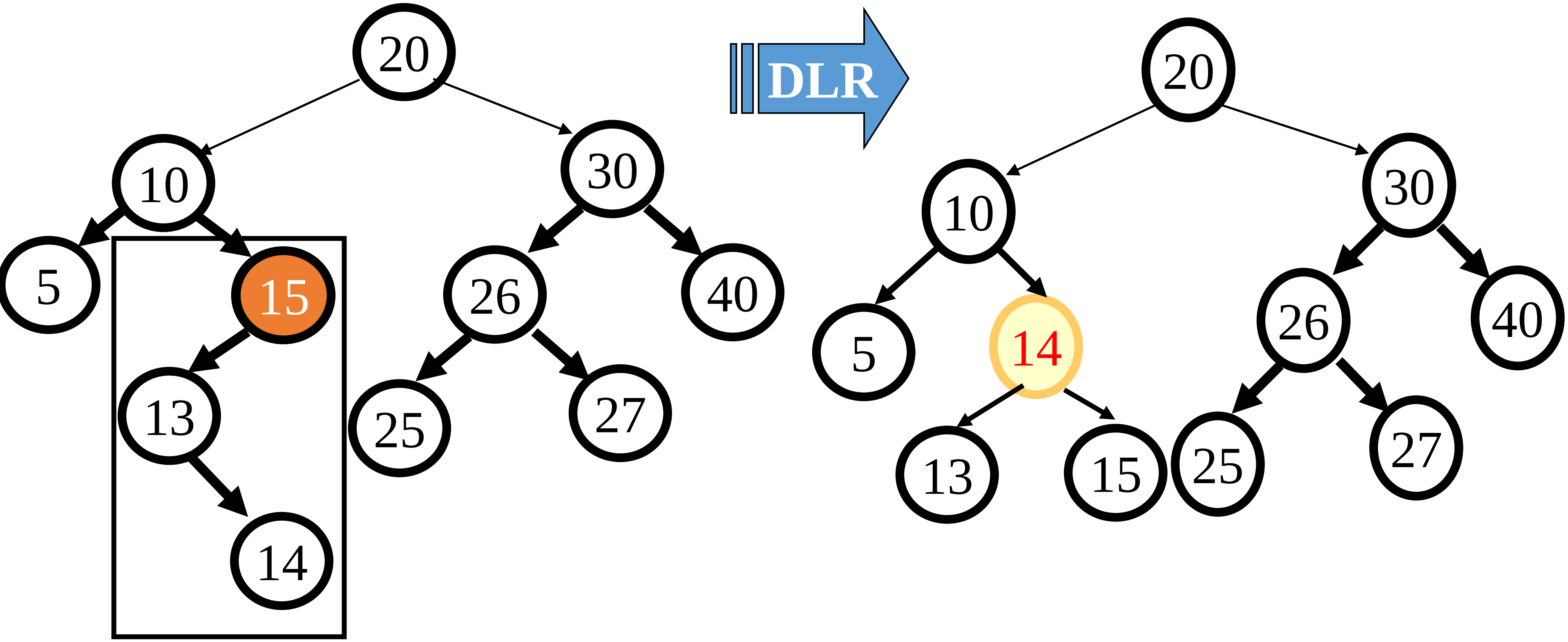
Inserting data to AVL: An example

- Continue to **insert 15, 40, 25, 27, and 26**



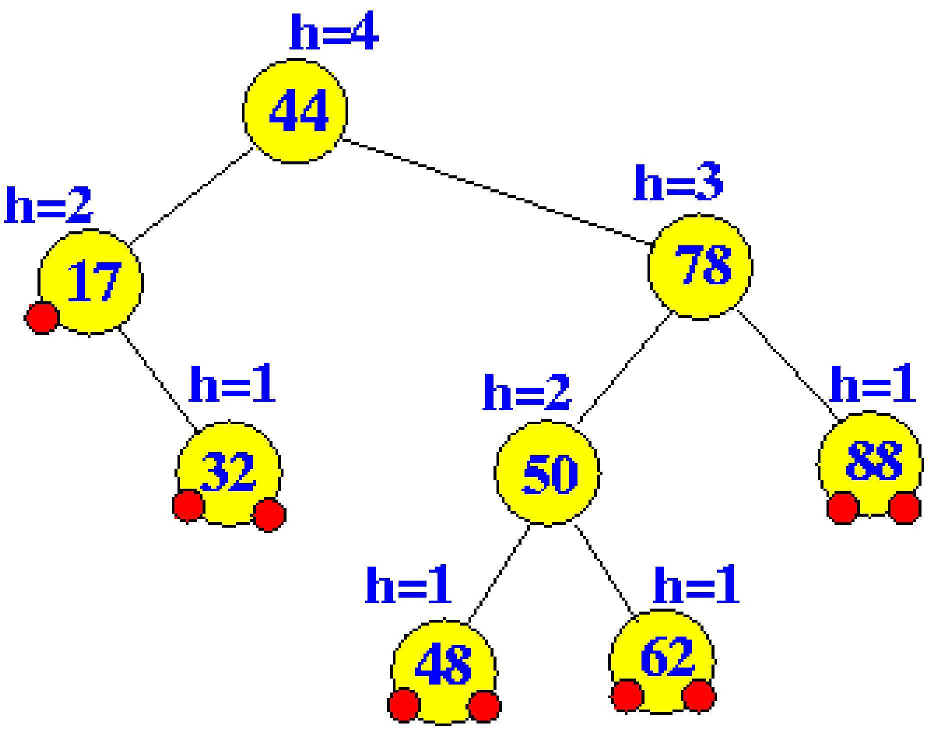
Inserting data to AVL: An example

- Continue to **insert 5, 13,** and **14**

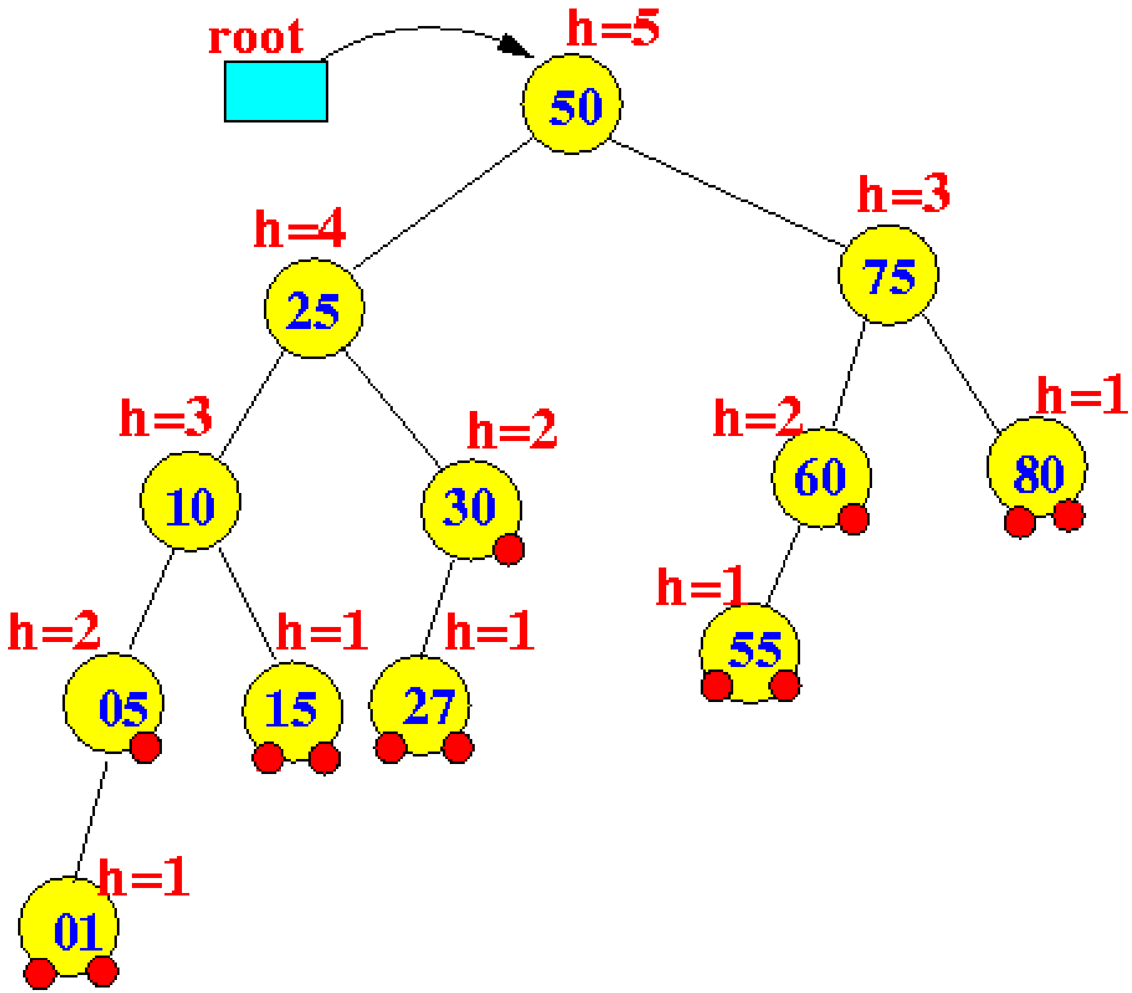


Checkpoint 02: Insert data into an AVL tree

What is the result of inserting 46 into the below AVL tree?



Checkpoint 03: Delete data from an AVL tree



What is the result of removing 80 from the given AVL tree?

Exercises

01. Operations on BST / AVL trees

- Consider the following sequence of operations on an initially empty search tree:

1. Insert 10
2. Insert 100
3. Insert 30
4. Insert 80
5. Insert 50
6. Remove 10
7. Insert 60
8. Insert 70
9. Insert 40
10. Remove 80
11. Insert 90
12. Insert 20
13. Remove 30
14. Remove 70

- What does the tree look like after these operations execute if the tree is a a) Binary search tree? b) AVL tree?

THE END