AVL Trees

- Named after Adelson-Velskii and Landis
- Every node in the tree has a balance factor

$$h_L - h_R$$

h_L =Height of the left subtree

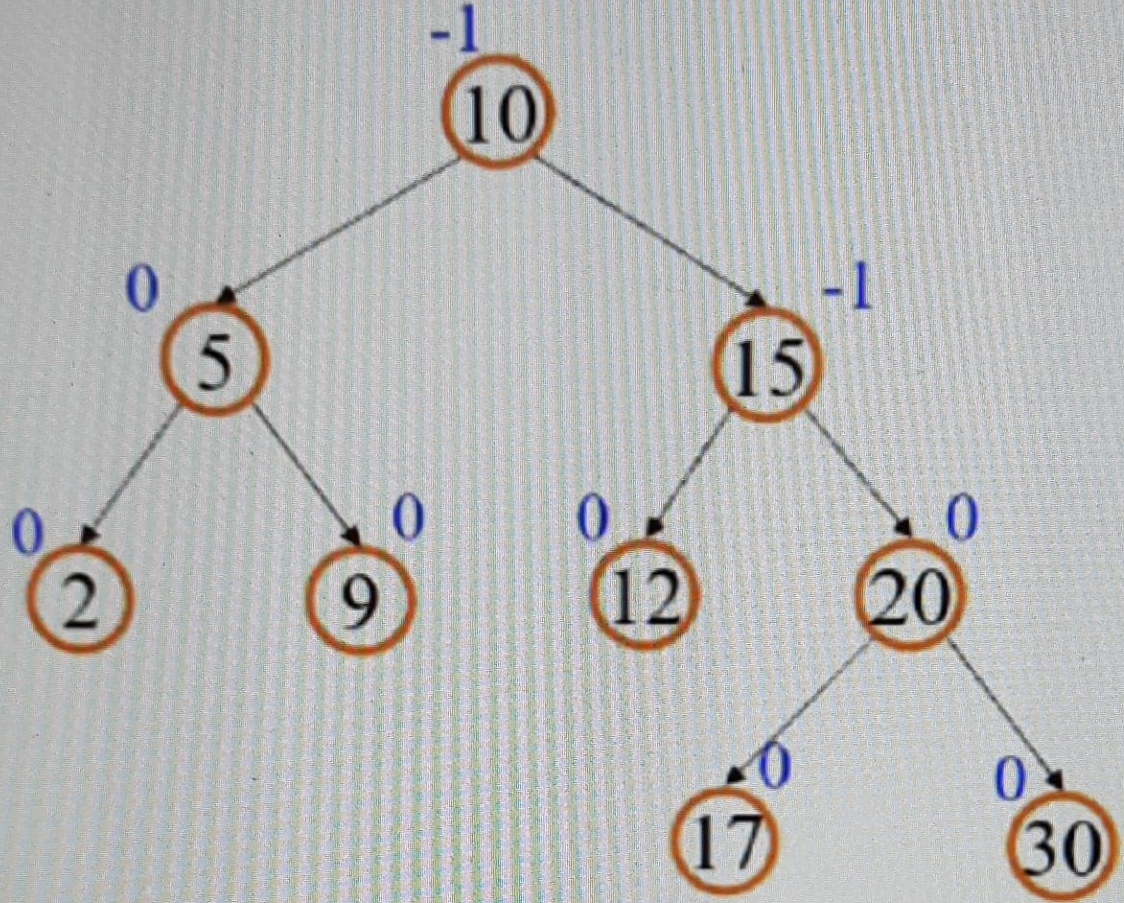
h_R =Height of the left subtree

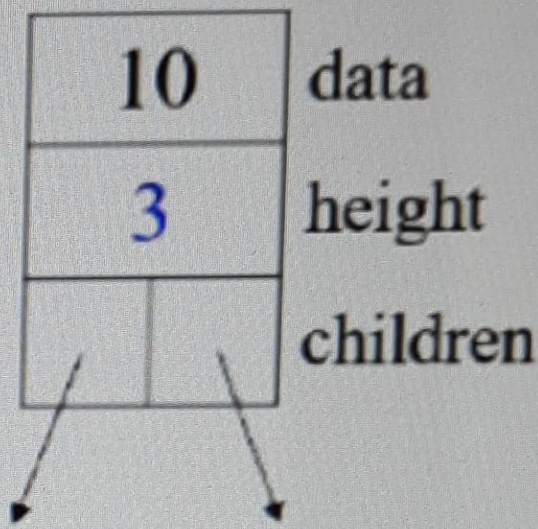
- The difference in the heights between the left and right sub-trees is at most 1

$$| h_L - h_R | <= 1$$

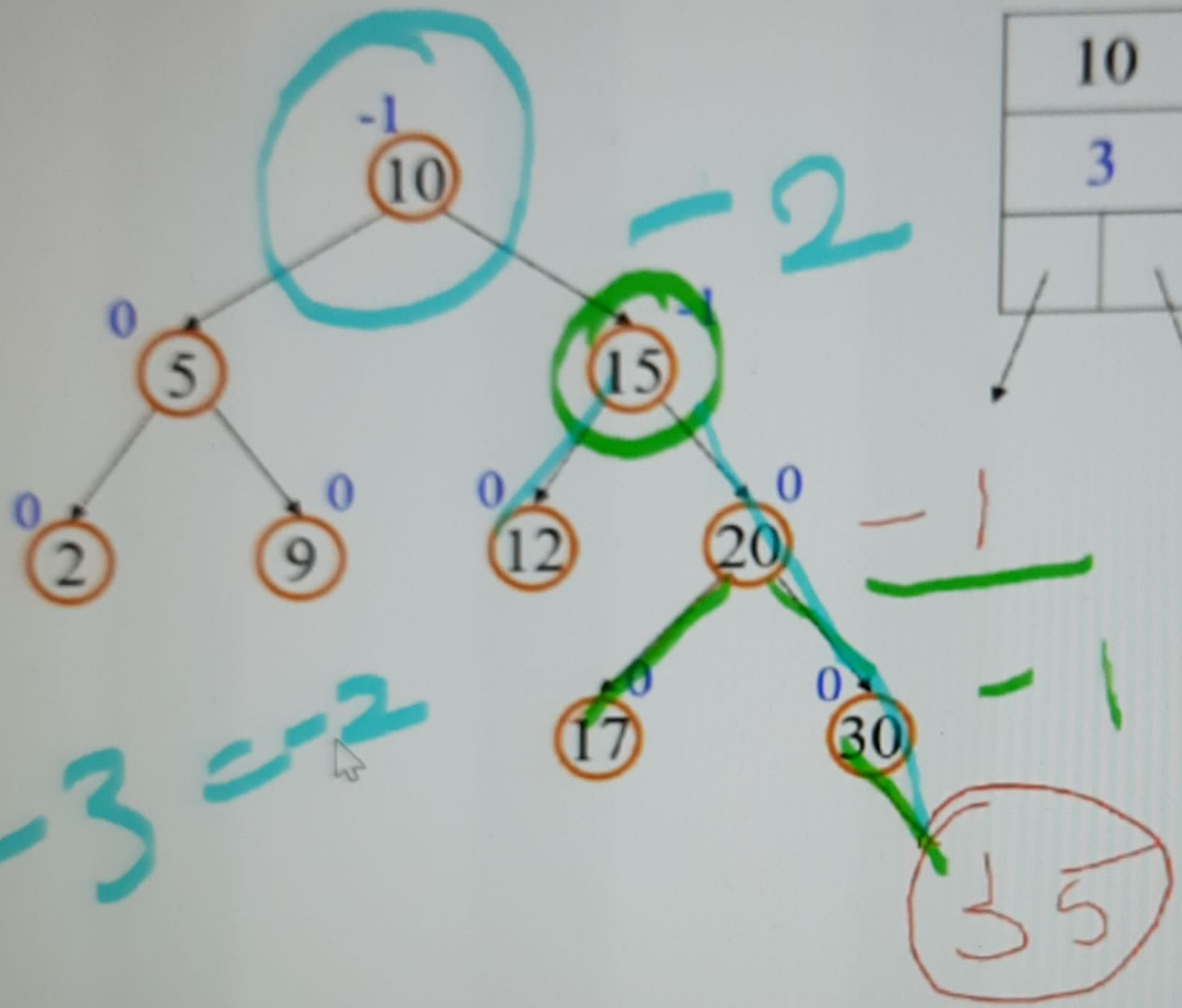
- An AVL tree that satisfies BST properties is a AVL search tree

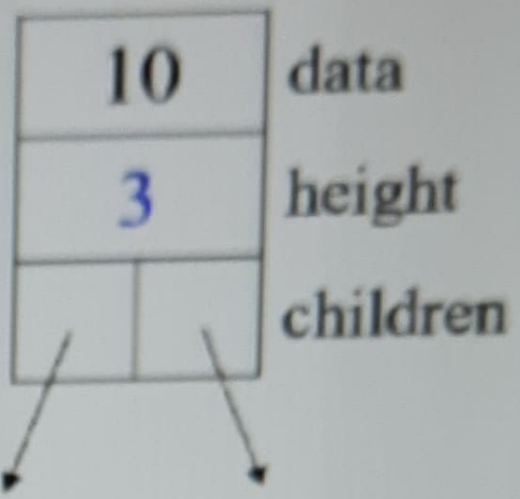
An AVL Tree



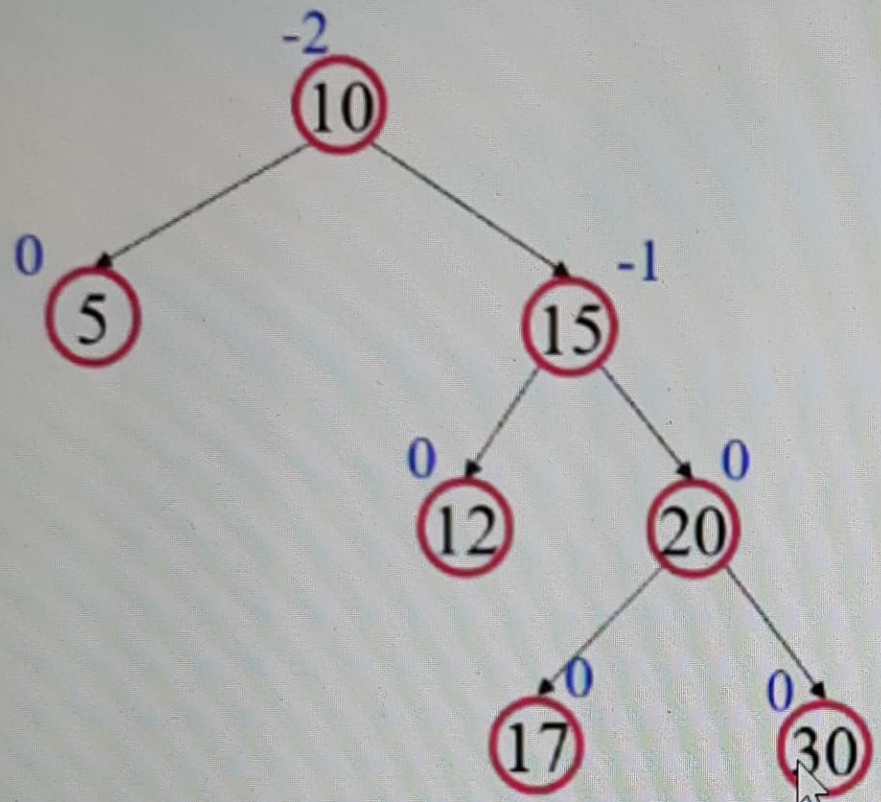


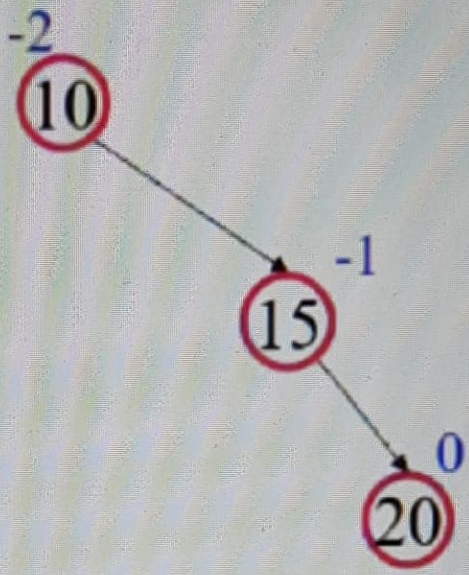
An AVL Tree


-1
10
0
5
15
0
2
0
9
0
12
0
20
0
17
0
30
35
-2
-1
-1
1-3=-2


10
data
3
height
children

Not AVL Trees





Operations on a AVL tree

- Search / retrieval
- Insertion
- Deletion

Maintaining Balance

To maintain AVL balance, observe that:

- Inserting a node can increase the height of a tree by at most 1
- Removing a node can decrease the height of a tree by at most 1

Rotation

Rotation

- Insertion may involve
  - LL Rotation
  - LR Rotation
  - RL Rotation
  - RR Rotation

Rotation

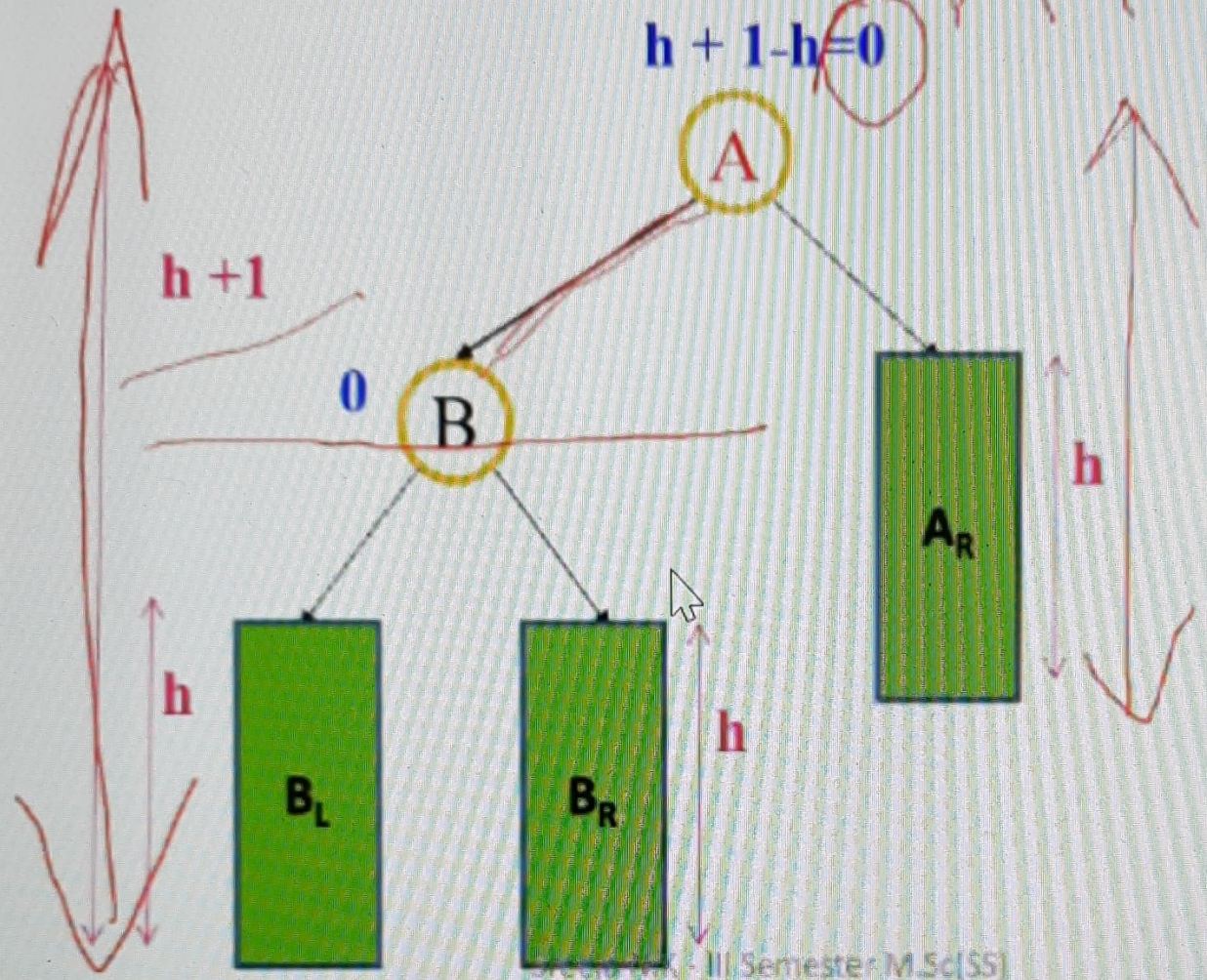
- Insertion may involve
  - LL Rotation
  - LR Rotation
  - RL Rotation
  - RR Rotation

Single Rotation

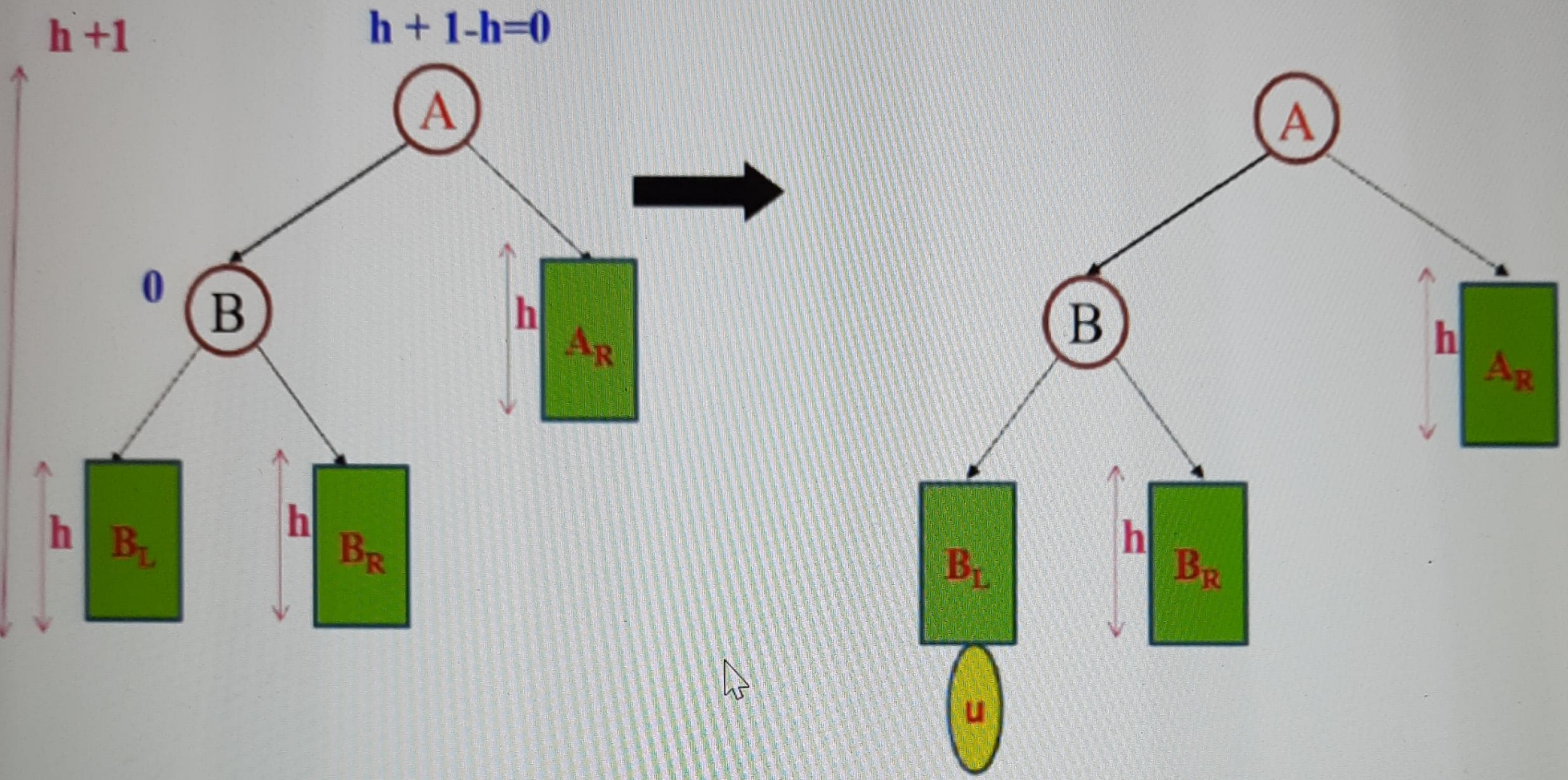
Double Rotation

LL Rotation

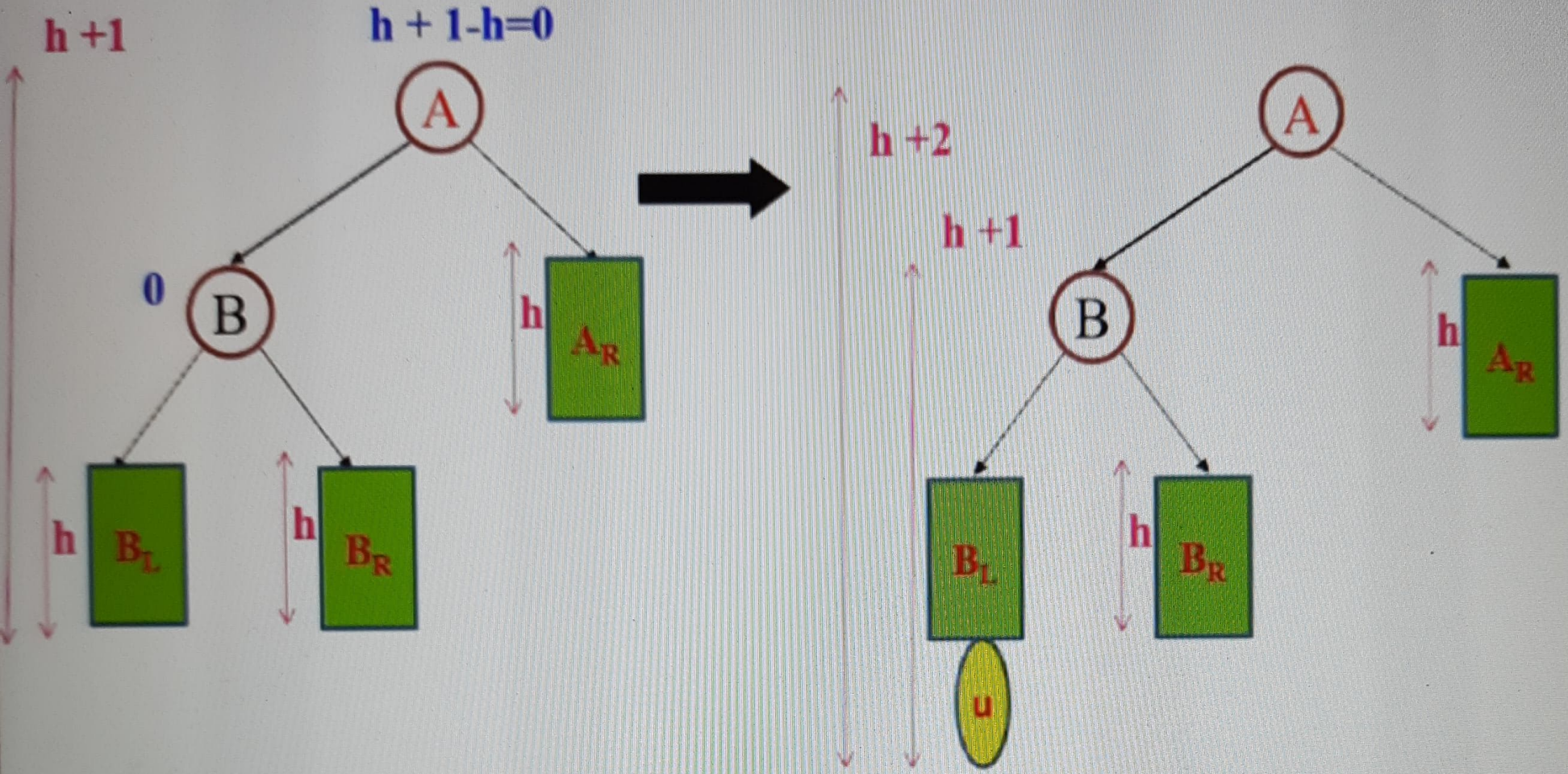
- Insert a node in the left subtree(L) of the left subtree(L) of A



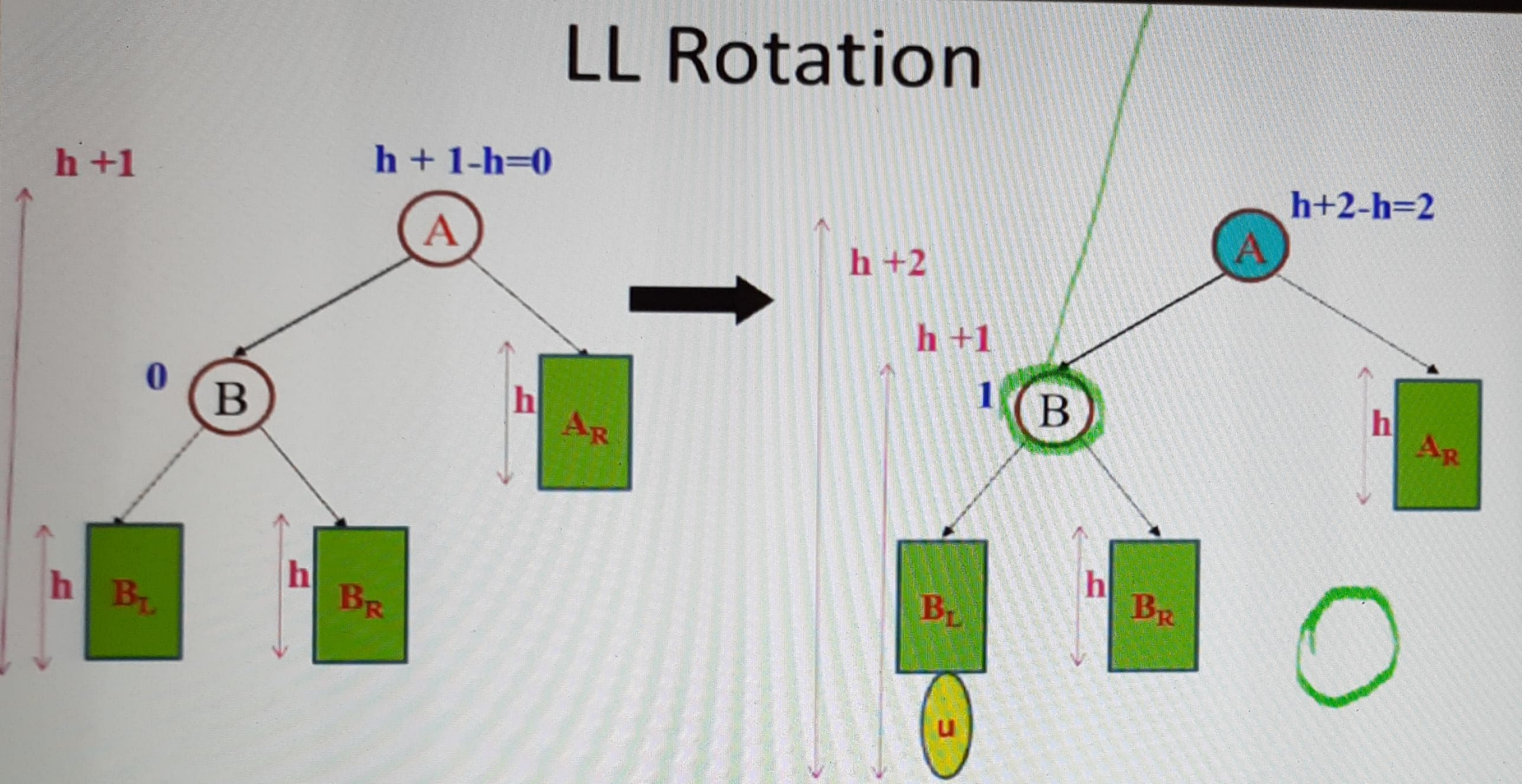
LL Rotation



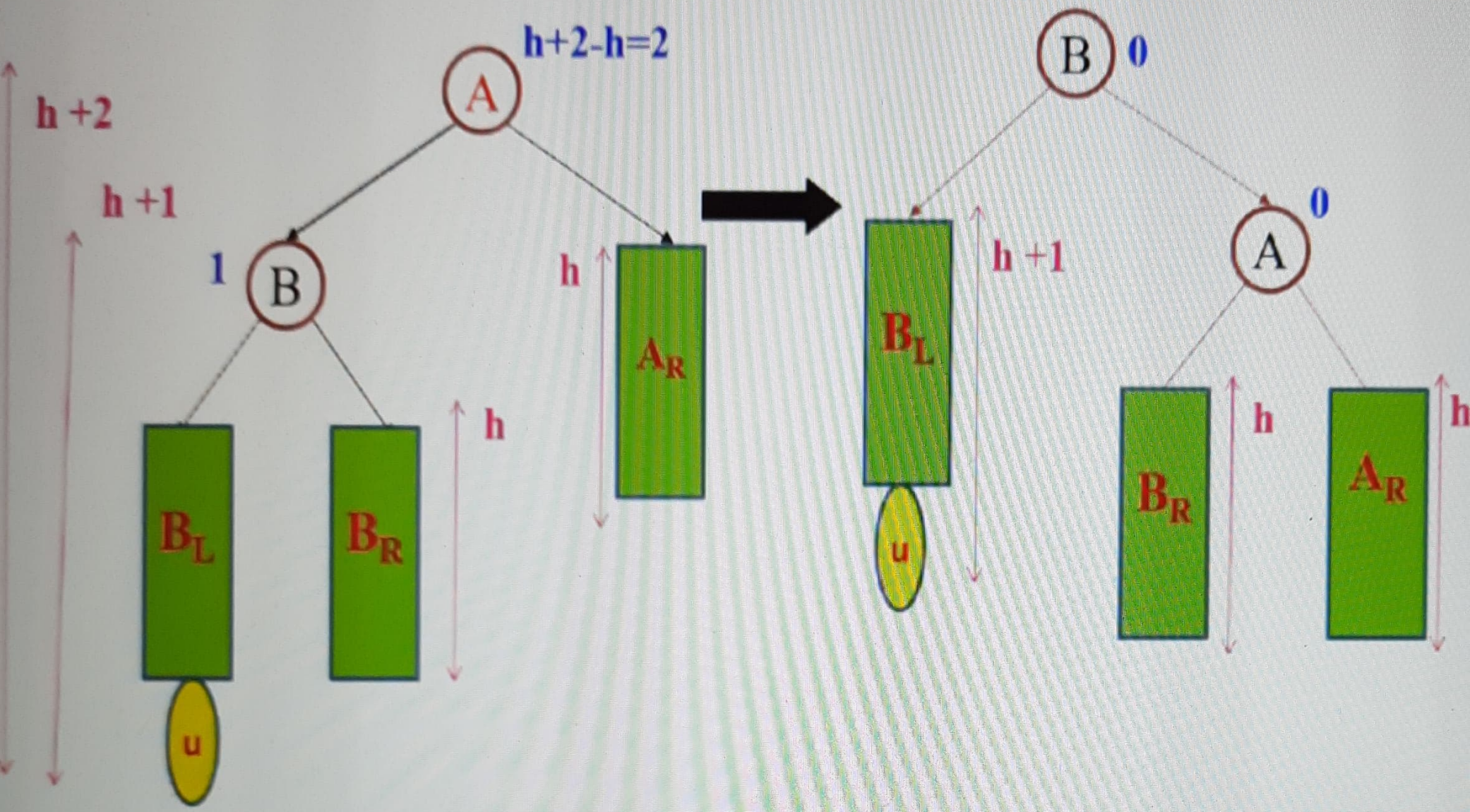
LL Rotation



LL Rotation

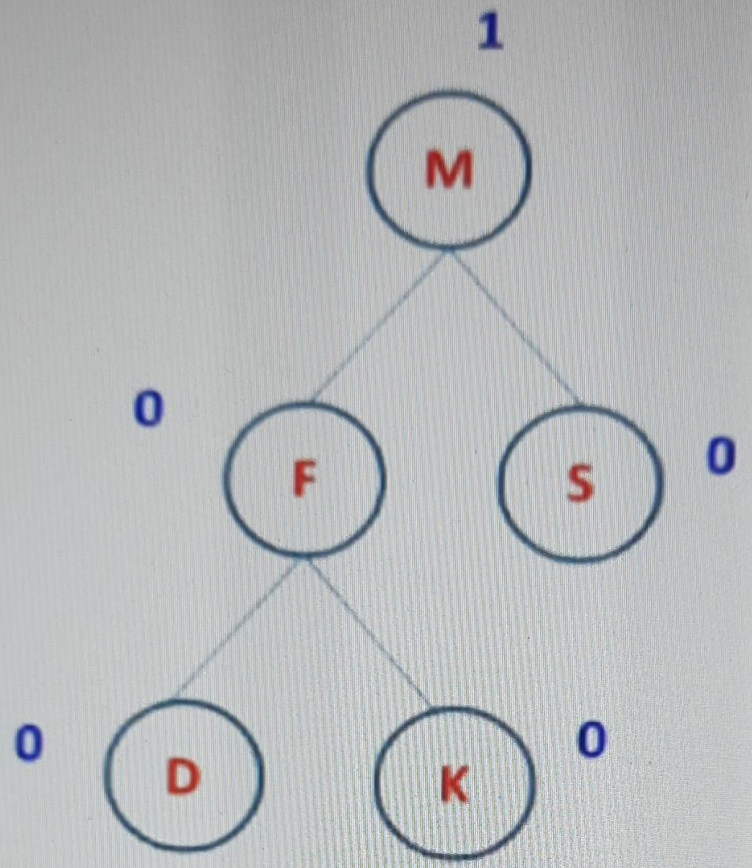


Single Rotation-LL



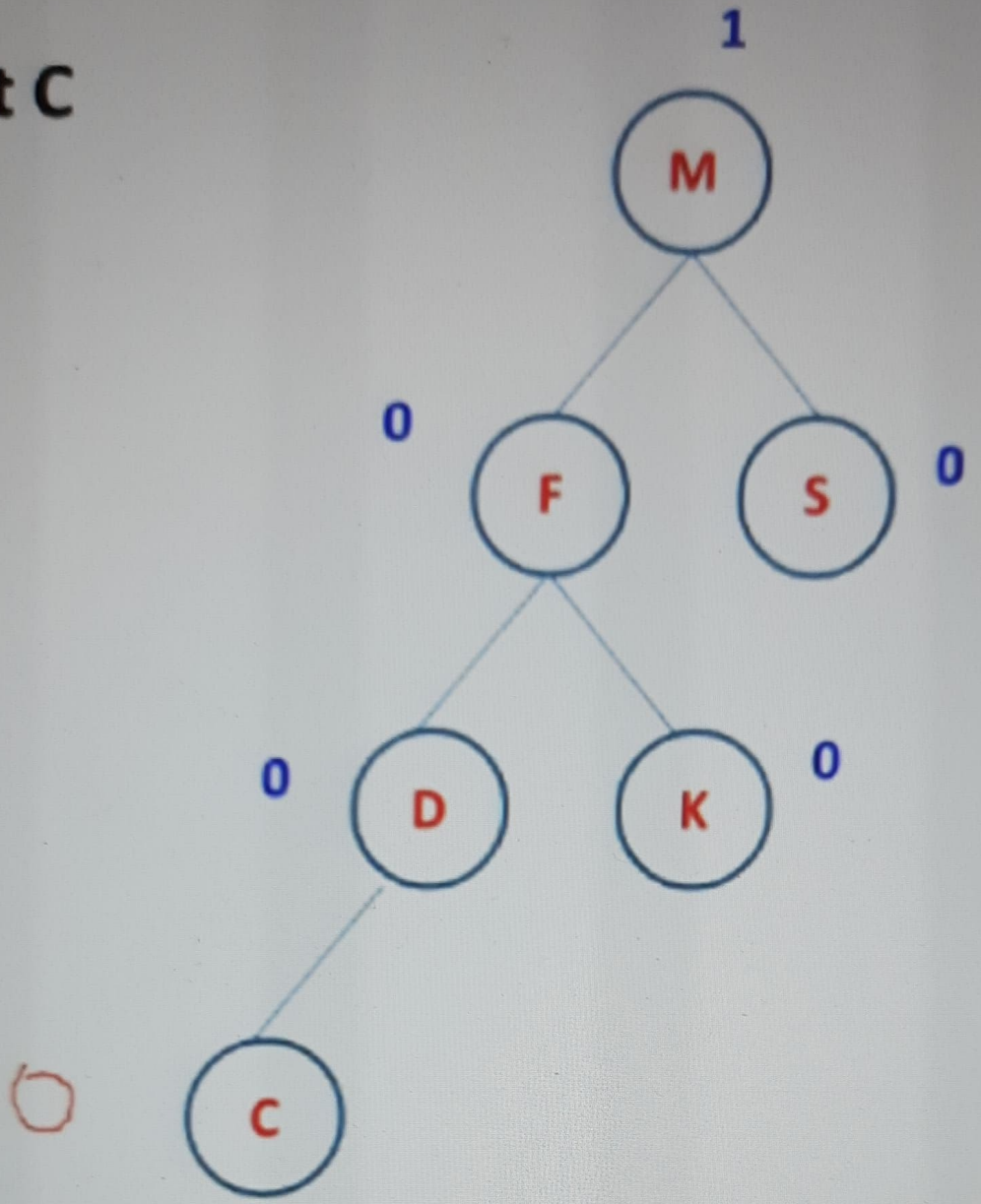
LL Rotation

Insert C

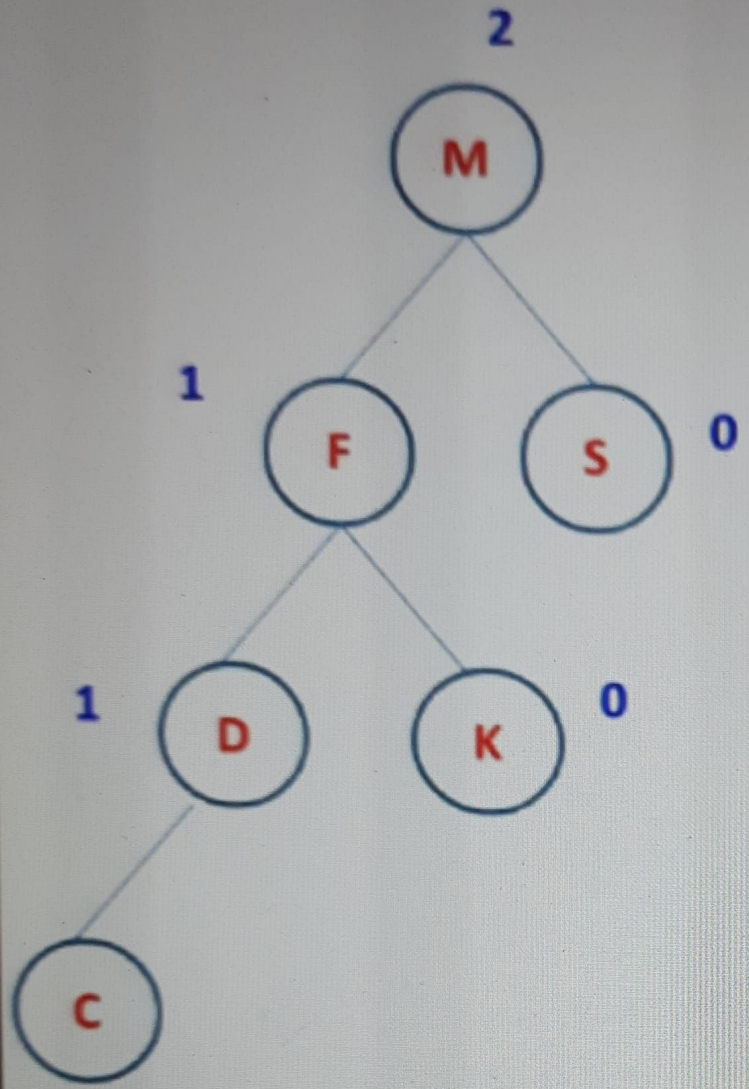


LL Rotation

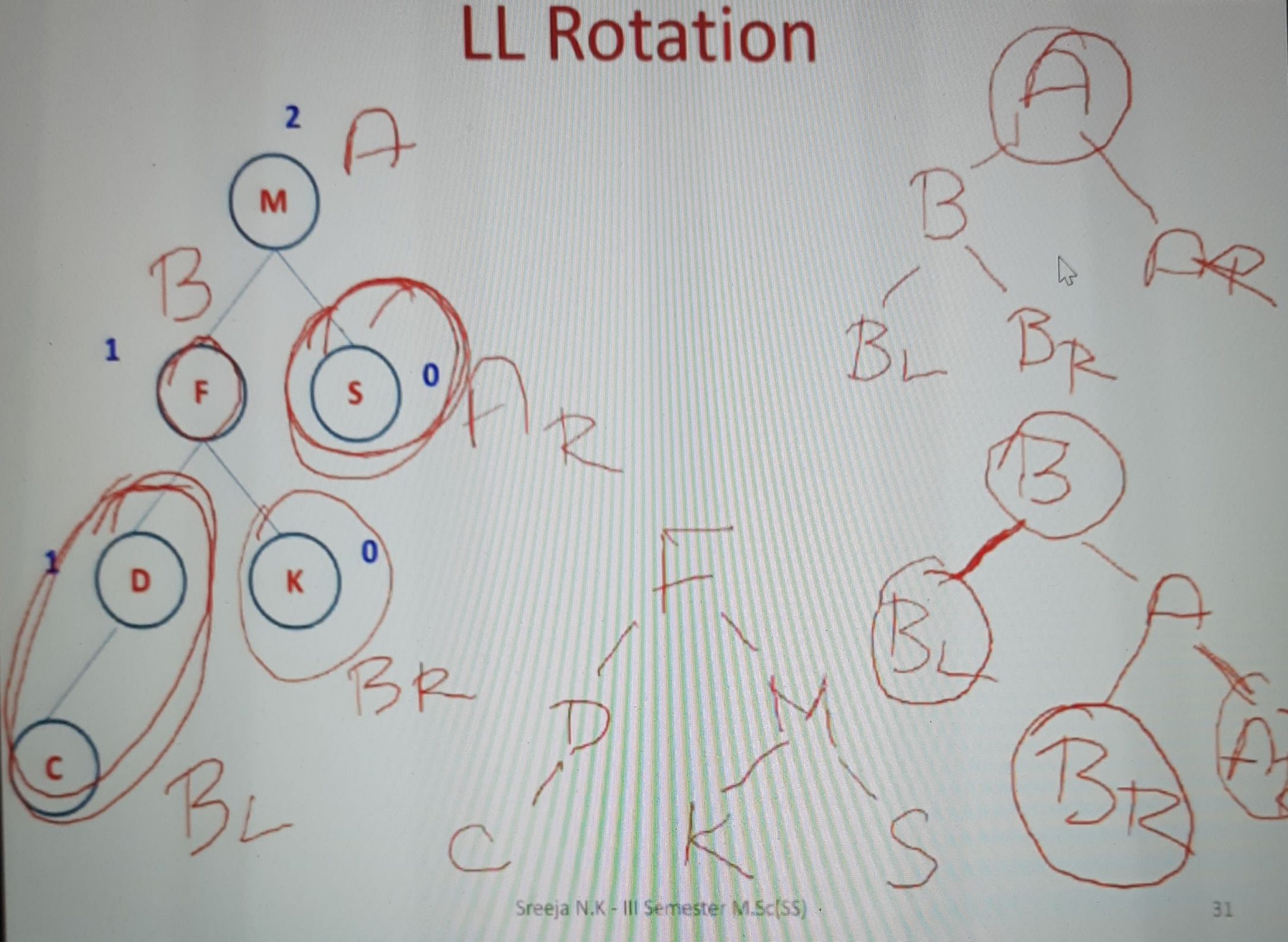
Insert C



LL Rotation

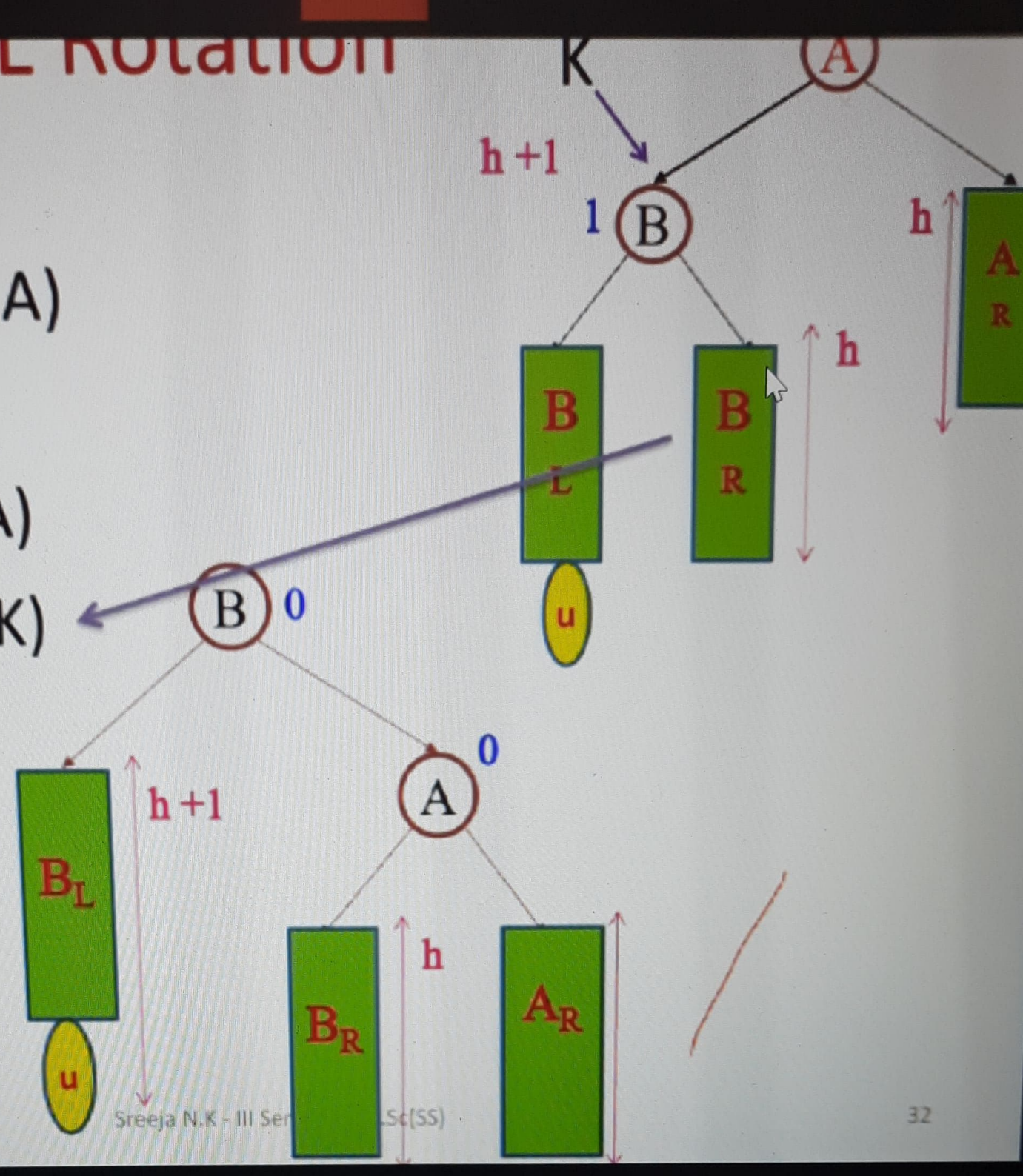


LL Rotation



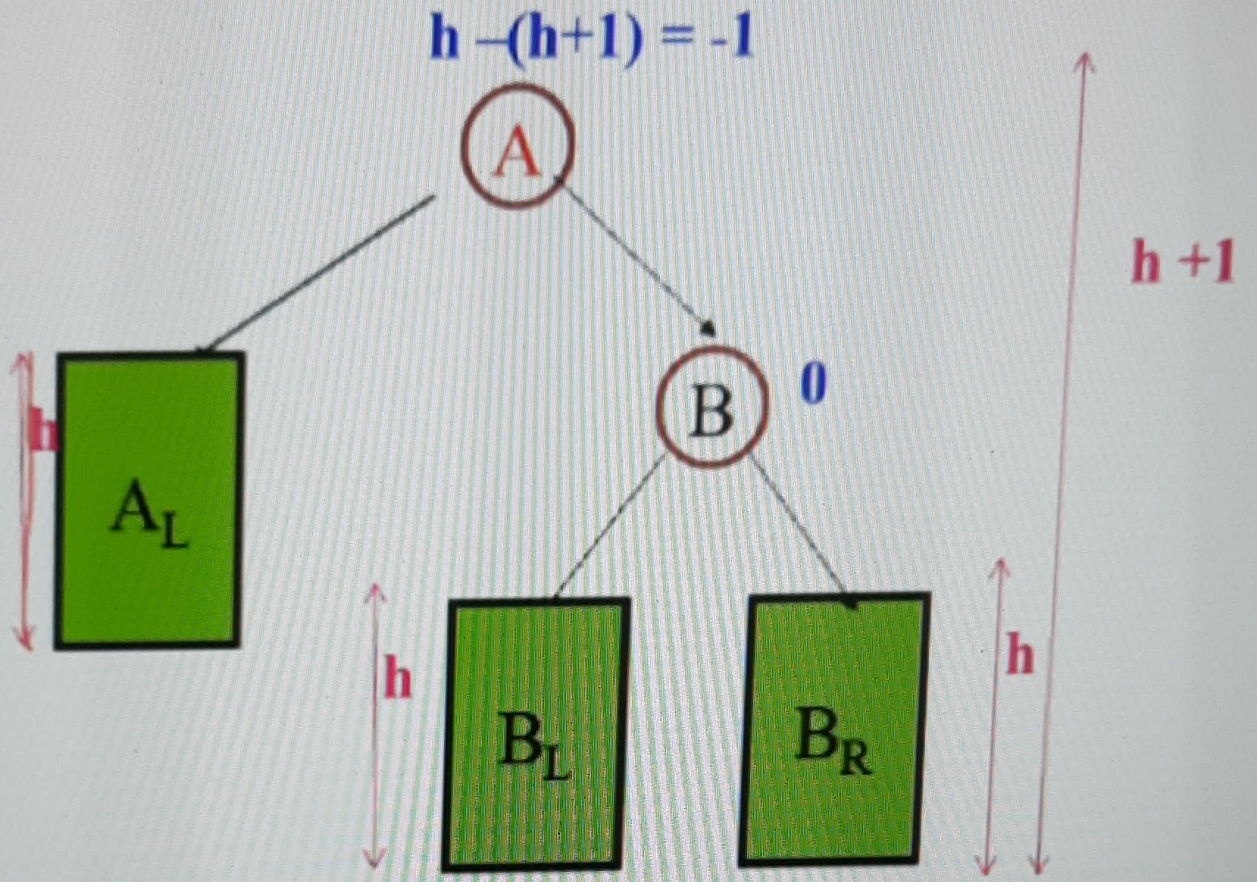
LL Rotation

```
Node LL (Node A)
{
 Node k = Left(A)
 Left (A)=Right(K)
 Right(K)=A
 return k
 }
```

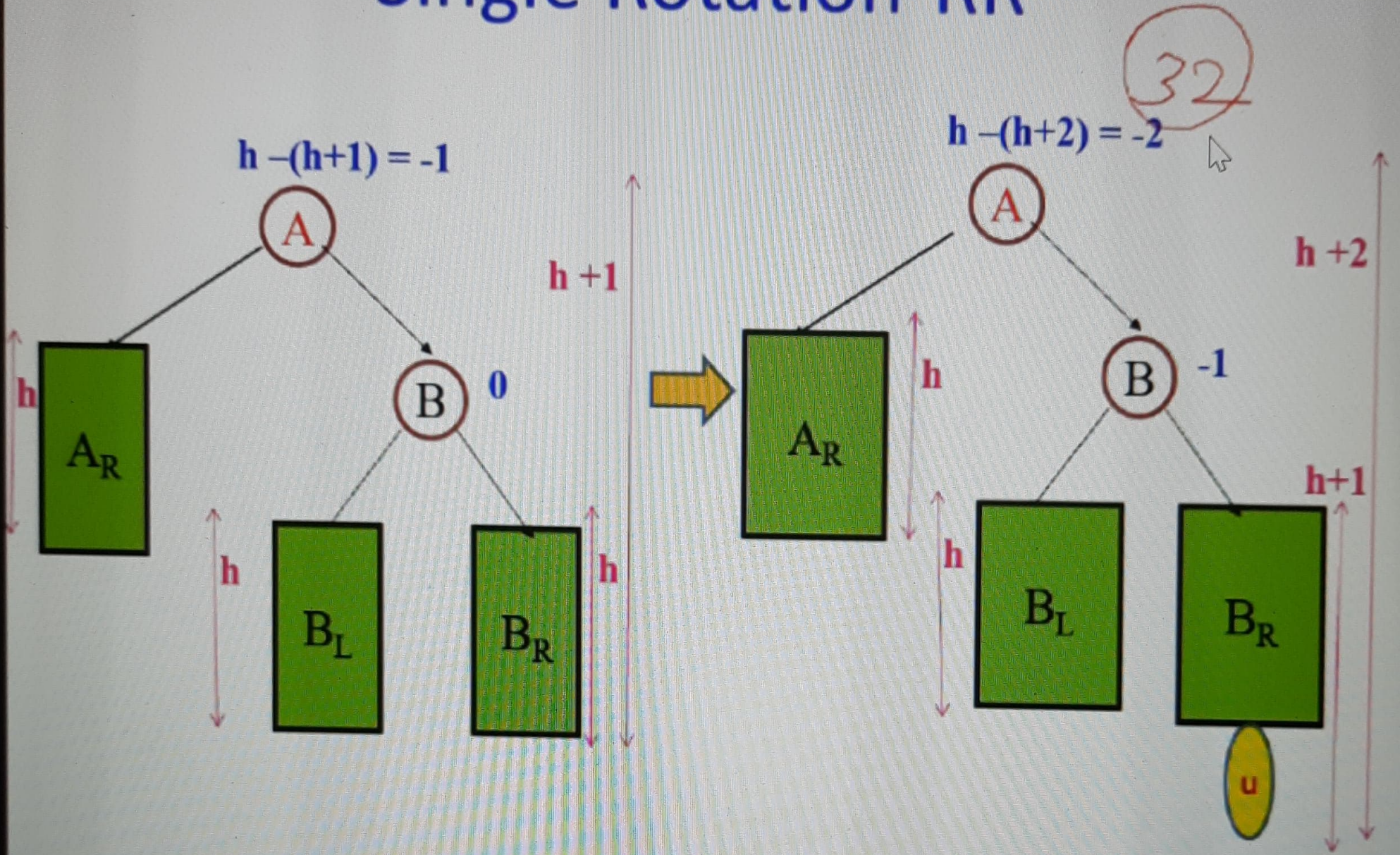


RR Rotation

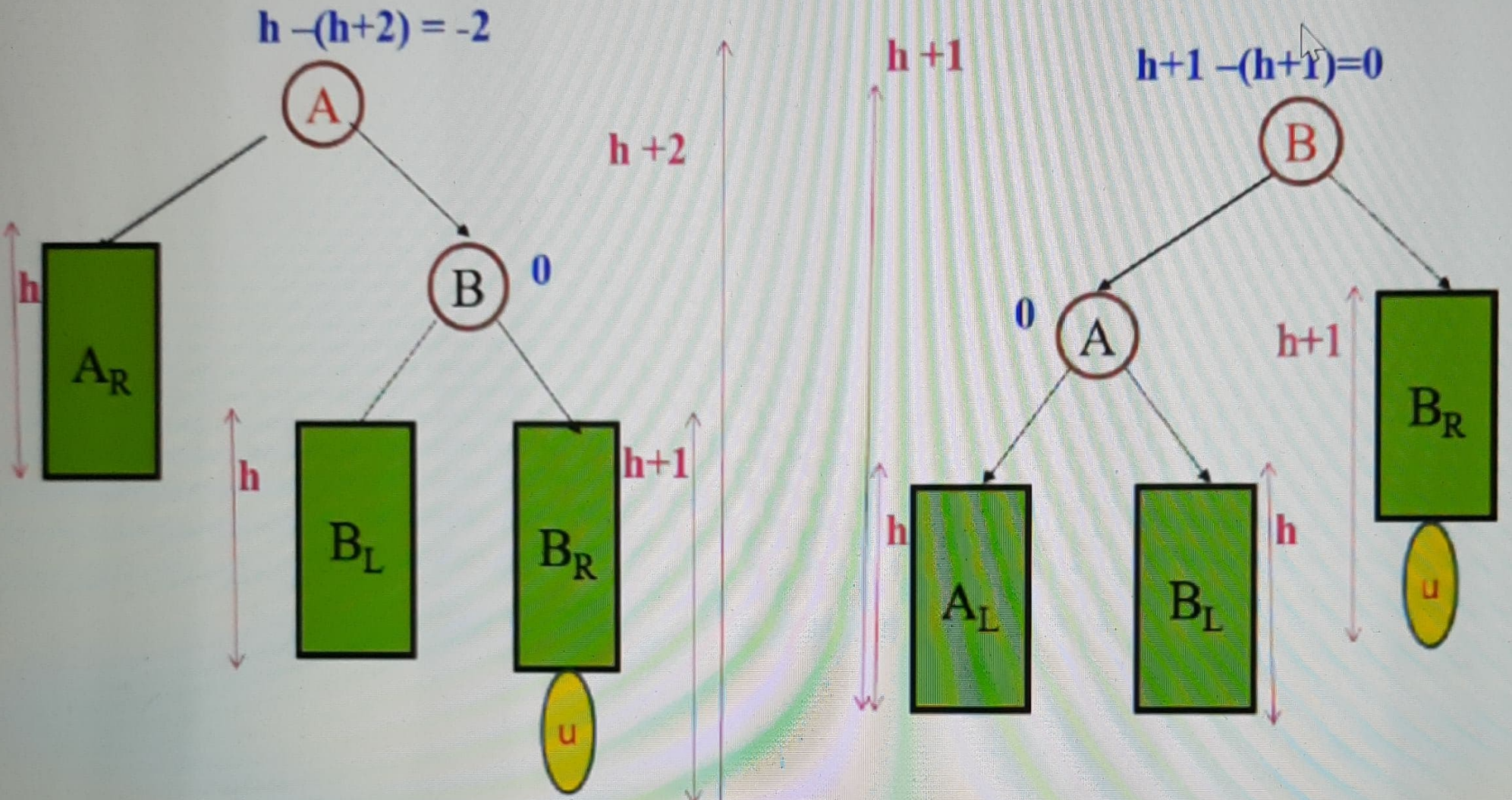
- Insert a node in the Right subtree(R) of the right subtree(R) of A



Single Rotation-RR

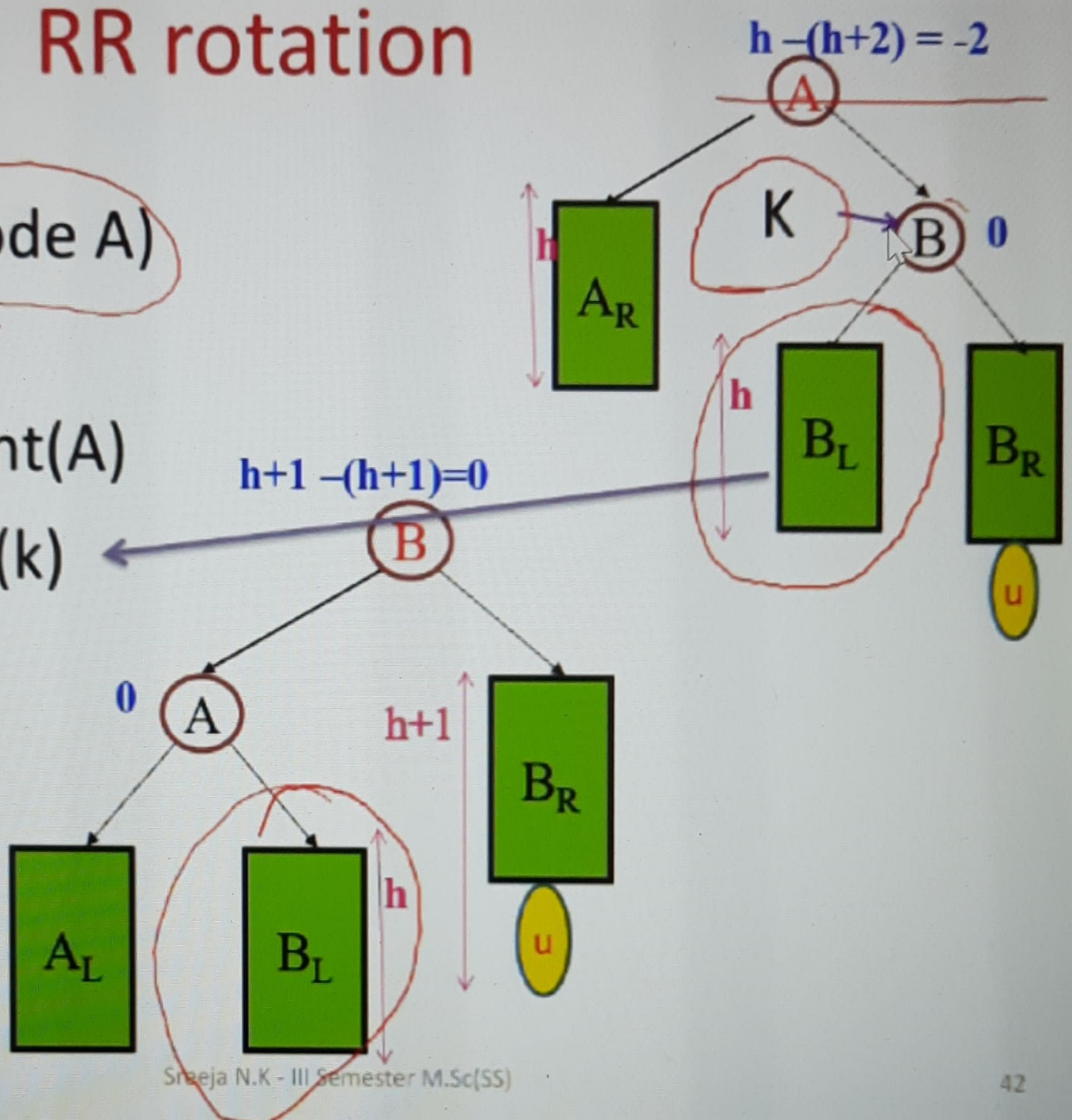


Single Rotation-RR

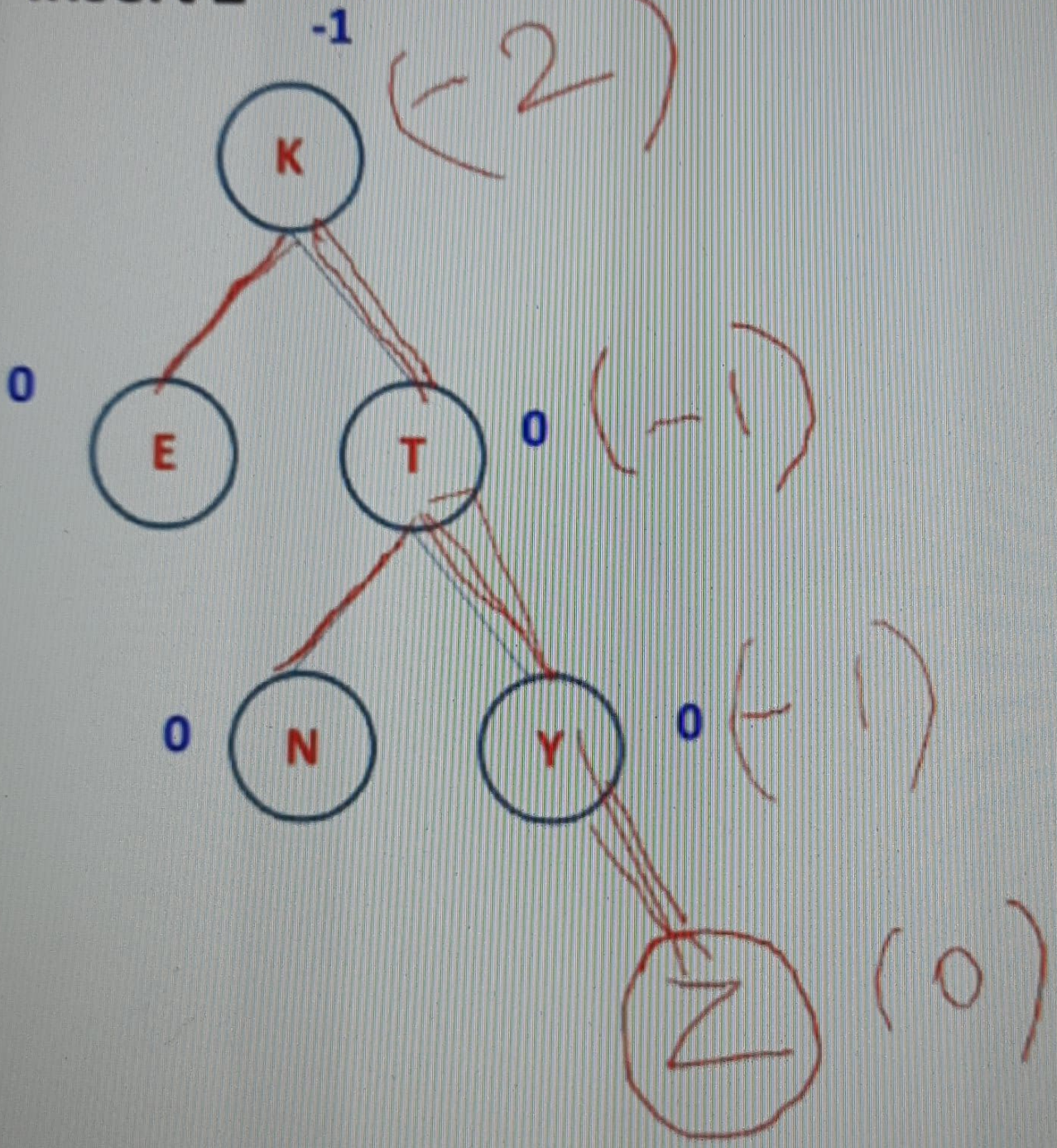


RR rotation

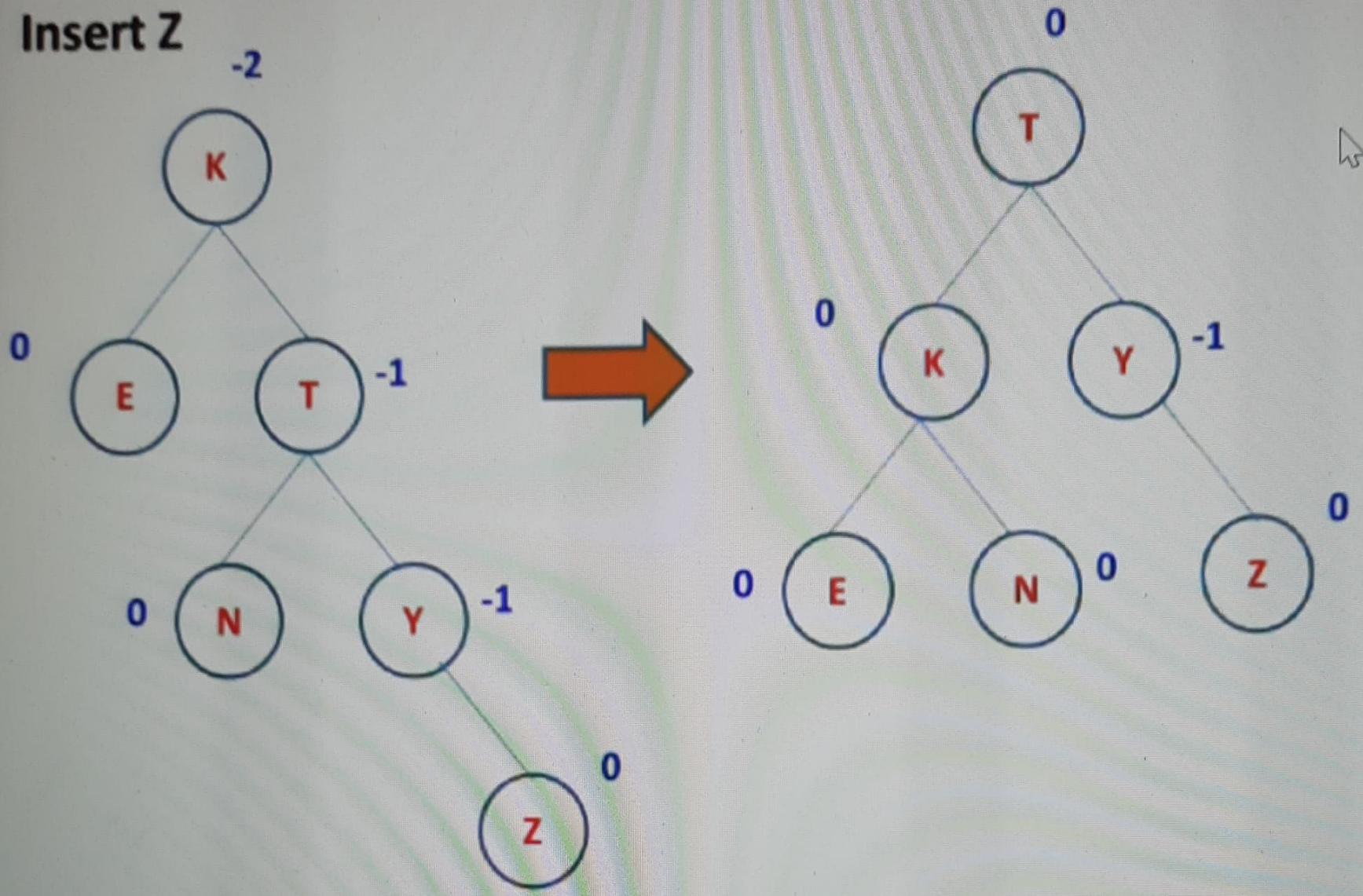
```
Node RR (Node A)
{
Node k = right(A)
right(A)=left(k)
left(K)=A
return k
}
```



Insert Z



Insert Z



LR Rotation

- Insert a node in the Right subtree(R) of the left subtree(L) of A