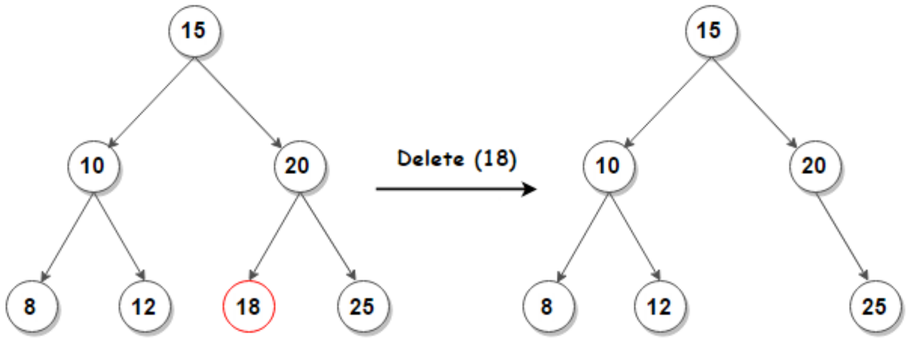
Trees 3 -- balancing

Remove

- `boolean remove(E element);`
- returns true if element existed and was removed and false otherwise
- Cases
  - element not in tree
  - element is a leaf
  - element has one child
  - element has two children

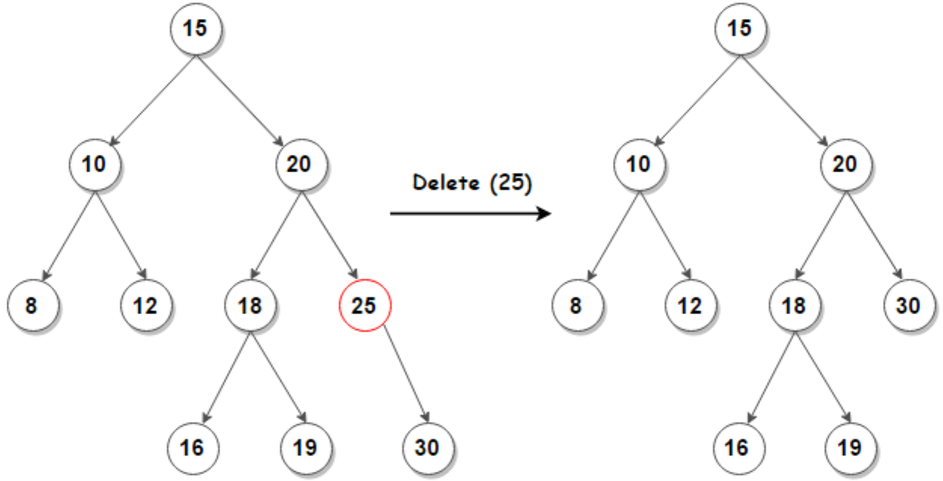
Leaf

- Just delete

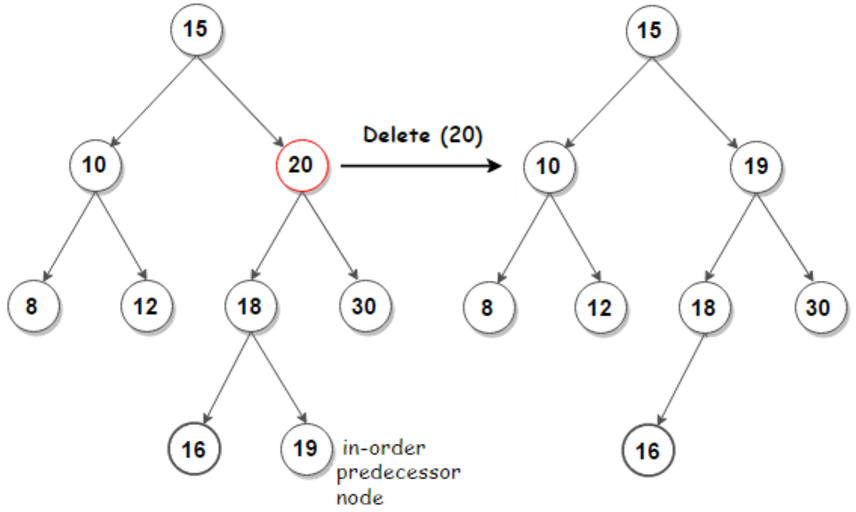


One child

- Replace with child – skip over like in linked list



2 Children
Replace with Predecessor



remove pseudocode

```
boolean remove(element)
  return removeUtil(element, root, null);

boolean removeUtil(element, node, parent)
  if (node==null) return false;
  if (node.payload>element)
    removeUtil(element, node.left, node);
  else if (node.payload<element)
    removeUtil(element, node.right, node);
  else
```

remove pseudocode 2

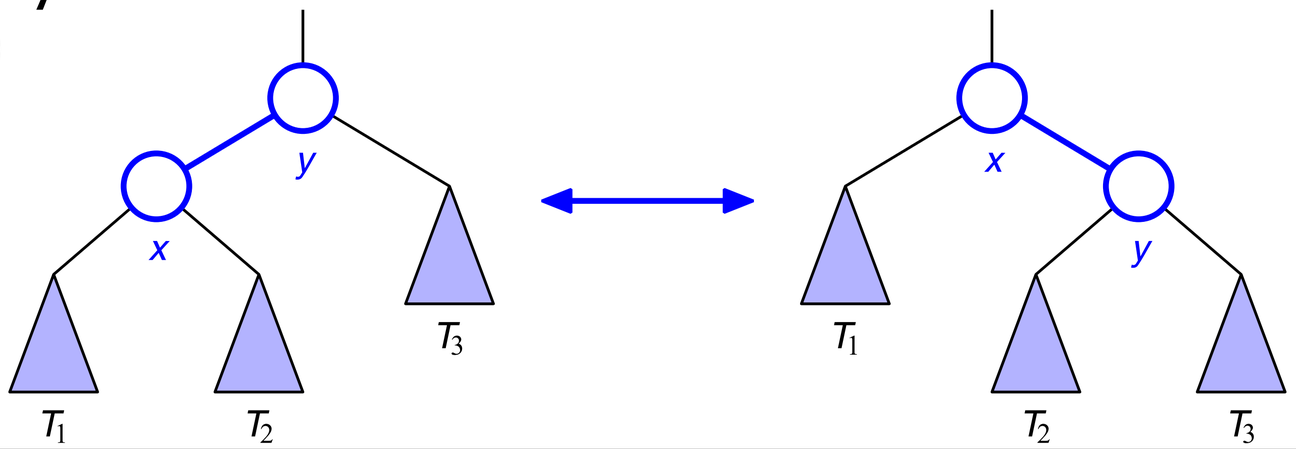
```
// found the node to delete
if (node.right==null && node.left==null)  // at a leaf
  if node==parent.right
     parent.right <- null
  else
     parent.left <- null
  return true

if (node.right==null) // one child on left
  if node==parent.right
     parent.right <- node.left
  else
     parent.left <- node.left
  return true
if (node.left==null)  // one child on right
  if node==parent.right
     parent.right <- node.right
  else
     parent.left <- node.right
 return true
```

remove pseudocode 3

```
// two children
successorNode = inorderSucessor(node.right)
node.payload <- successorNode.payload
removeUtil(successorNode.payload,
node.right, node);
return true;
```

Balanced Search Trees

- A variety of algorithms augment a standard BST with occasional operations to reshape, reduce height and maintain balance.
- General approach: Rotation — moves a child to be above its parent,
  - ideally O(1)
  - certainly no worse than O(lgn)

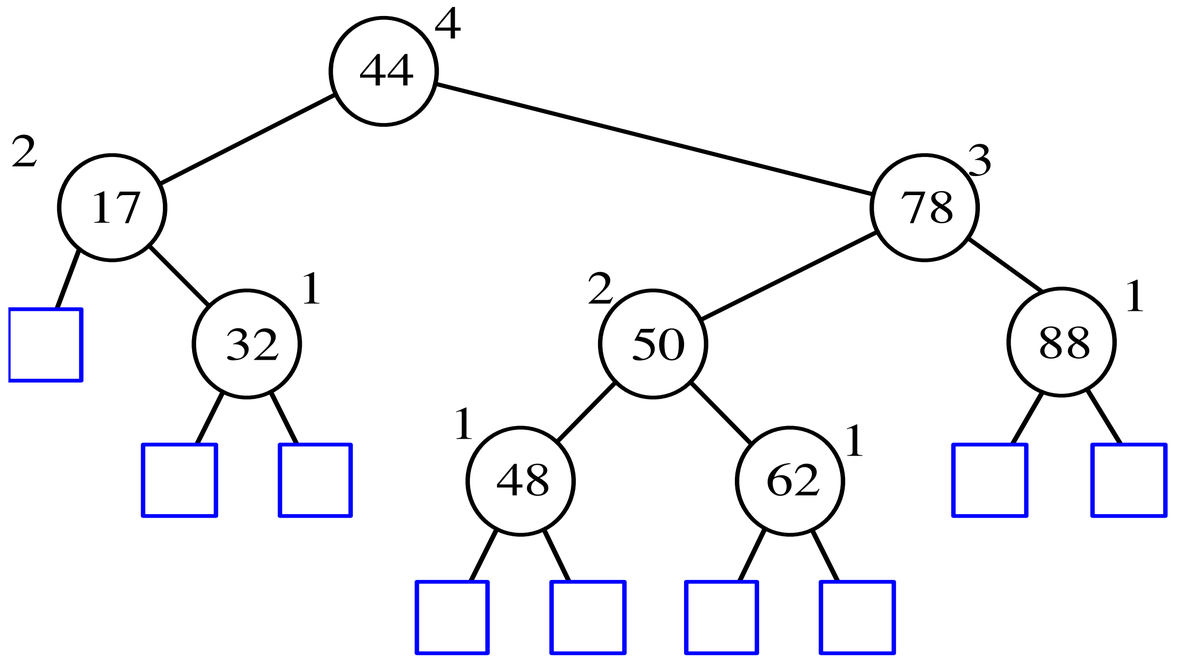


AVL Trees

Adleson-Velesky
Landis

- Height-balance property
  - as opposed to "weight" balanced
  - For every parent node, the avlHeight of the two children differ by at most 1
    - avlHeight = max distance from null endpoint
- Any binary tree satisfying the height-balance property is an AVL tree
- A height-balanced tree has height O(lg n)
  - max height is provably 1.44*lg(n)

AVL Tree Example



Insertion

- Maintain with each node the avlHeight.
- On insertion, first recur down through tree to insert.
- Then as you unwind recursion, update the avlHeight of each node.
- If height changes, check the height of other child
  - if not in balance then fix

Insertion code to maintain height

```
private class Node {
    Comparable<E> element;
    int avlHight;
    Node right;
    Node left;

    public Node(Comparable<E> e) {
        avlHight = 1;  // WHY??
        element=e;
        right=null;
        left=null;
    }
}
```

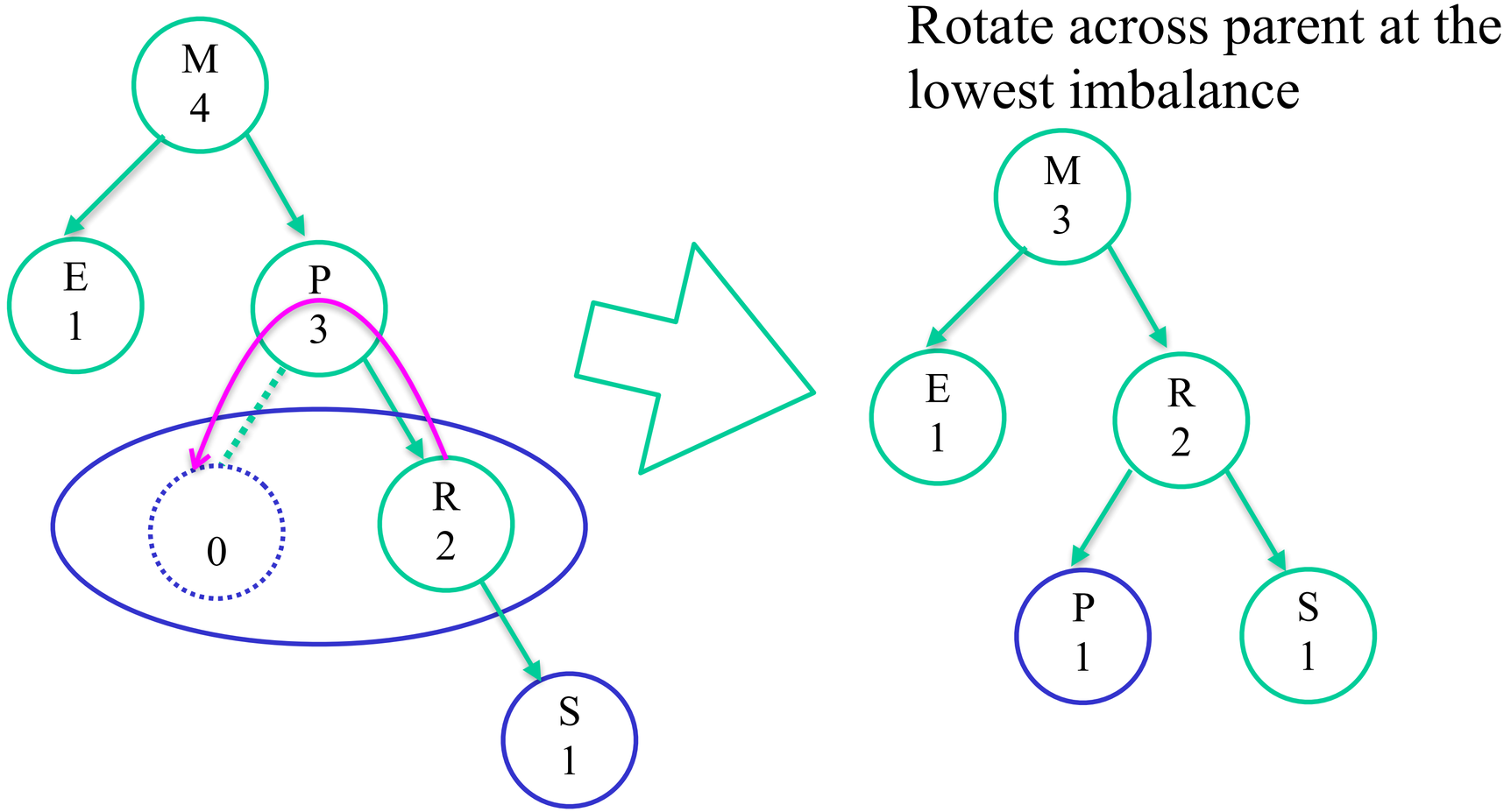
More insertion (pseudo)code

```
int insertUtil(node, element):

   if element==node.payload
       return node.avlLevel

   avlD=0

   if node.payload > element:
     if node.left==null
       node.left=new Node(payload)
       avlD = 2
     else
       avlD = 1+insertUtil(node.left,element);
   else
       // same but for right

   node.avlHieght = greater of avlD and
                    node.avlHeight

   return node.avlHeight
```

Fixing height imbalances
Rotation!!

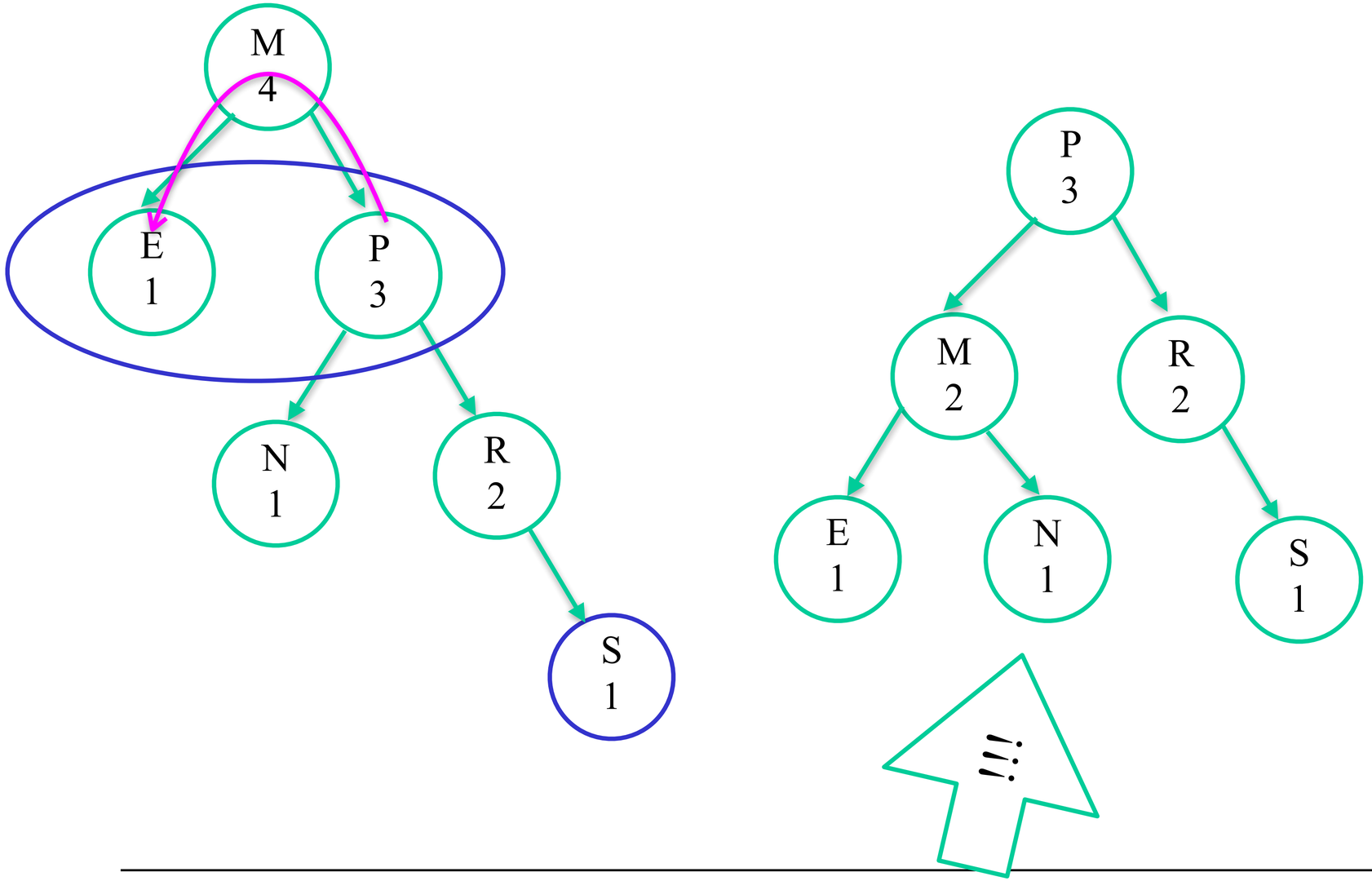
- Two types of rotation
- Single
  - left subtree of left node causes imbalance
  - right subtree of right node causes imbalance
- Double
  - right subtree of left node causes imbalance
  - left subtree of right node causes imbalance
    - The first rotation of a double puts the tree into position for a single rotation!

Single Rotation

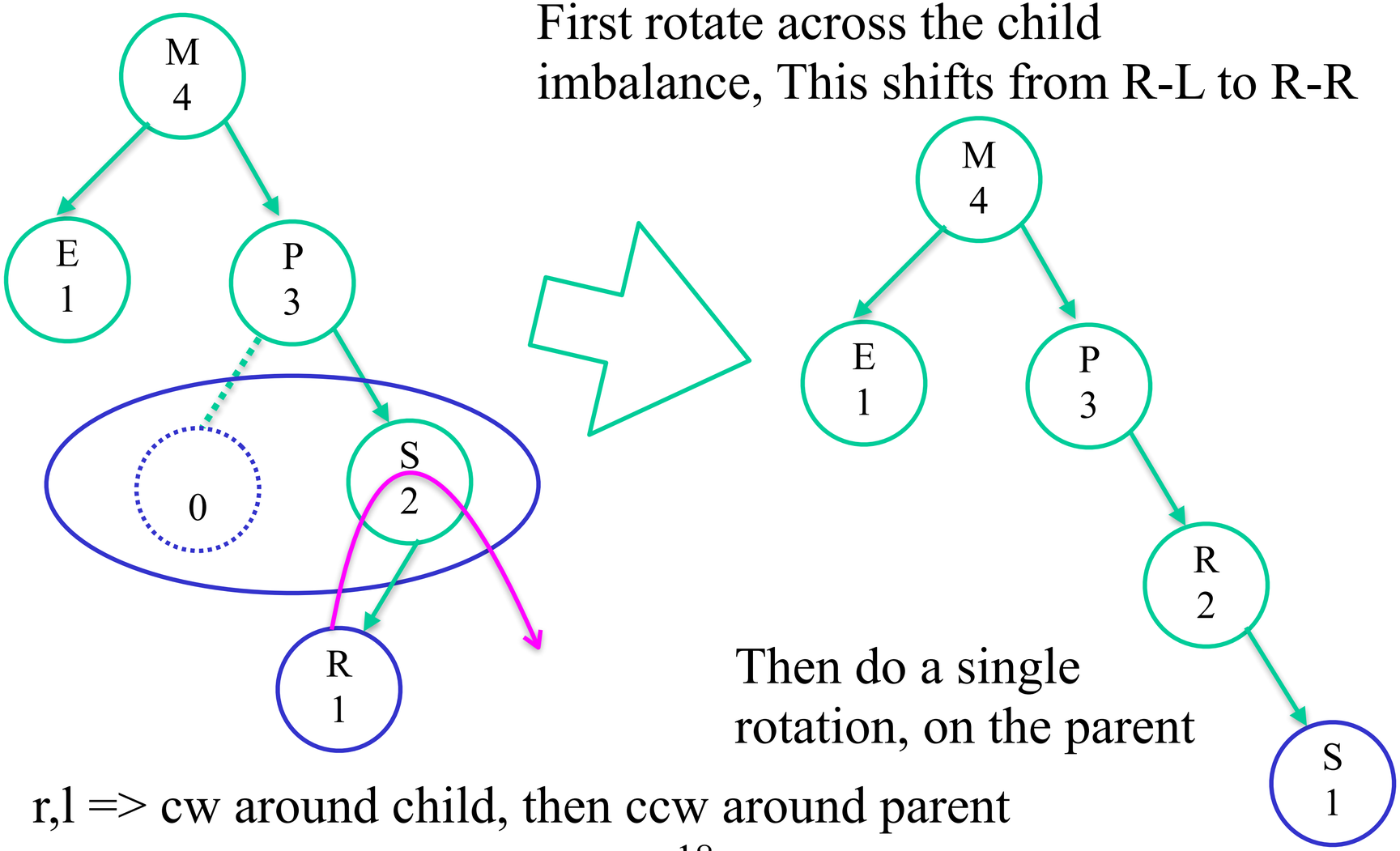


right-right => counter-clockwise rotation left-left => cw rotation

Single Rotation
"Detatchment Syndrome"



Double Rotation



r,l => cw around child, then ccw around parent

Groups

insert	100
insert	200
insert	300
insert	400
insert	500
insert	600
insert	700
insert	800
insert	900
insert	750
insert	1000
insert	850
delete	400
delete	300
delete	200
delete	700
delete	500