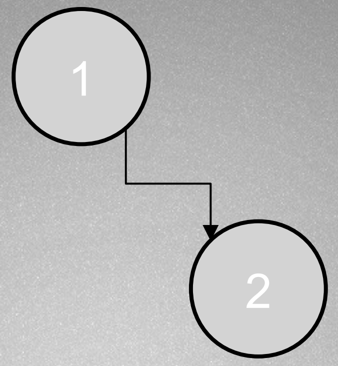
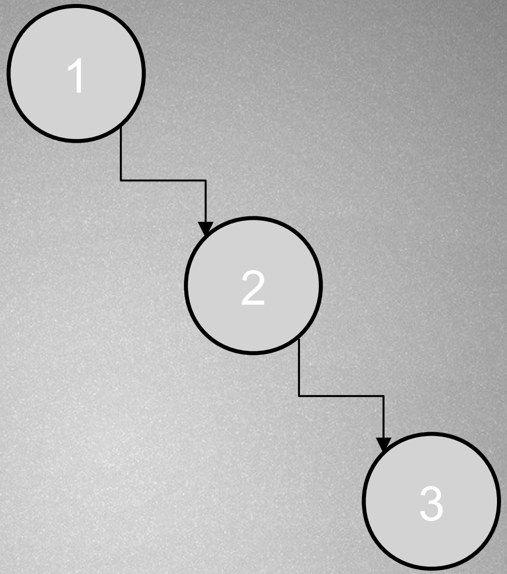
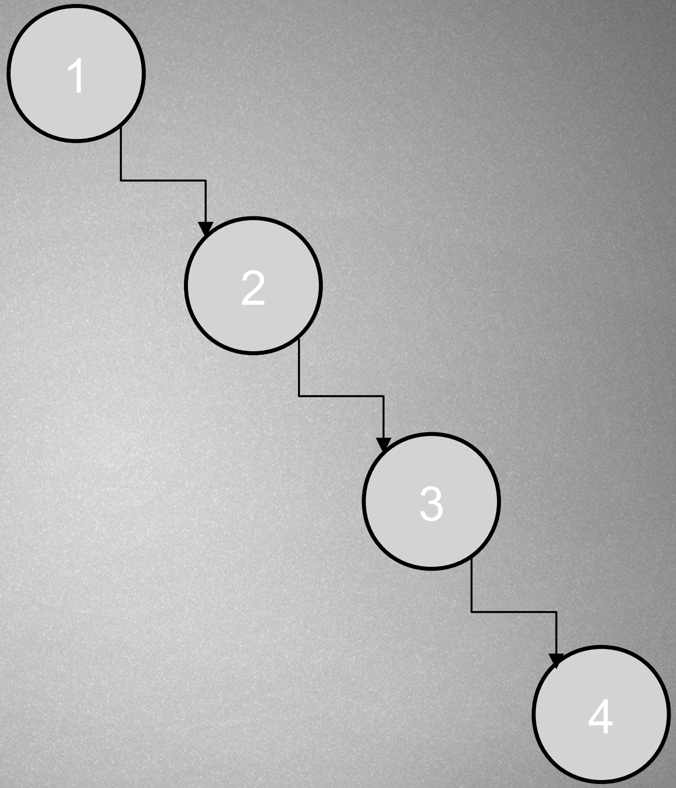
RED-BLACK TREES

Construct a BST from a sorted array
[1,2,3,4]

1


1
2


1
2
3



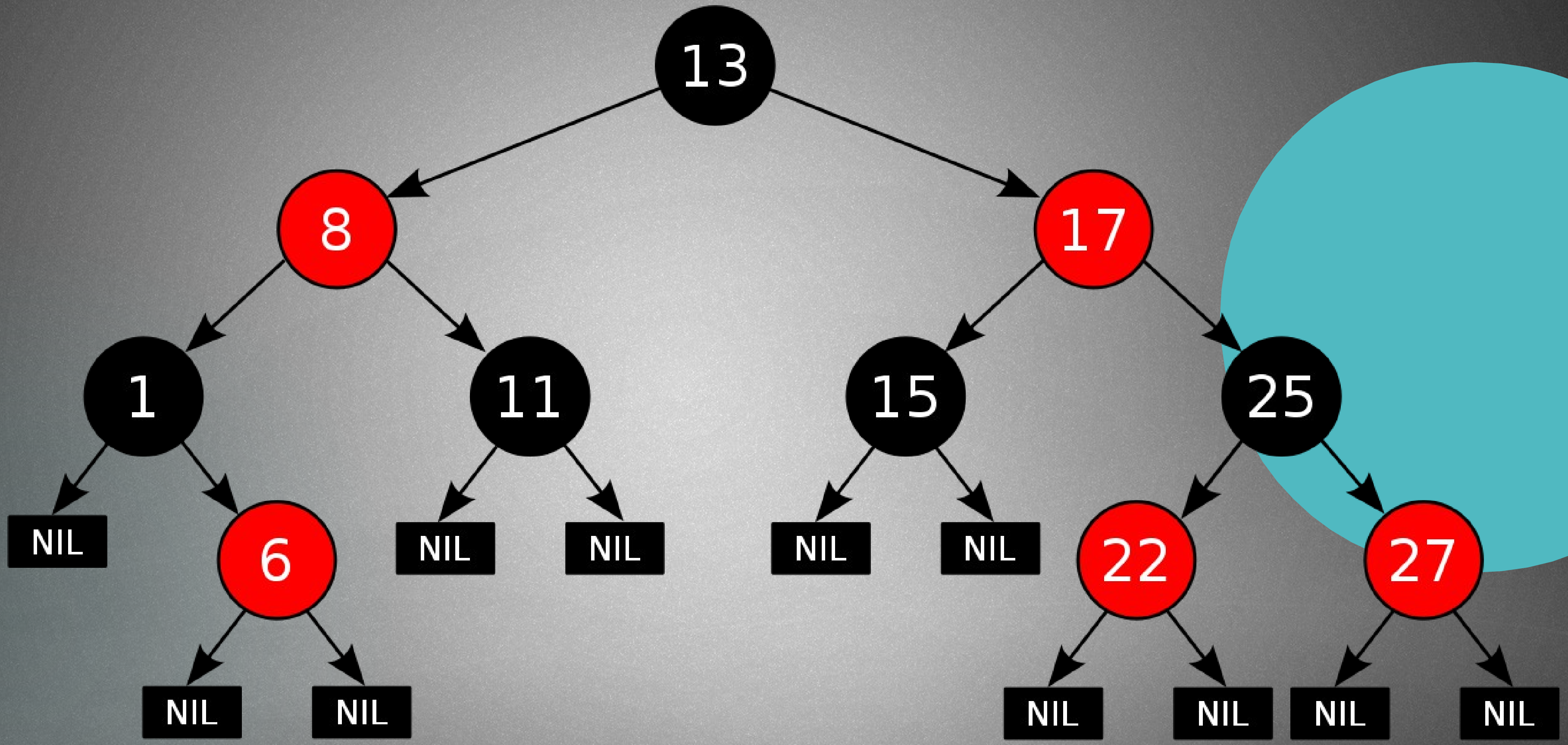
It is a linked list: **O(N)** !!!

- The running time of BST operations depends on the height of the binary search tree: we should keep the tree balanced in order to get the best performance
- In an AVL tree, the heights of the two child subtrees of any node differ by at most one
- Another solution to the problem is a red-black trees
- AVL trees are faster than red-black trees because they are more rigidly balanced BUT needs more work
- Operating Systems relies heavily on these data structures !!!

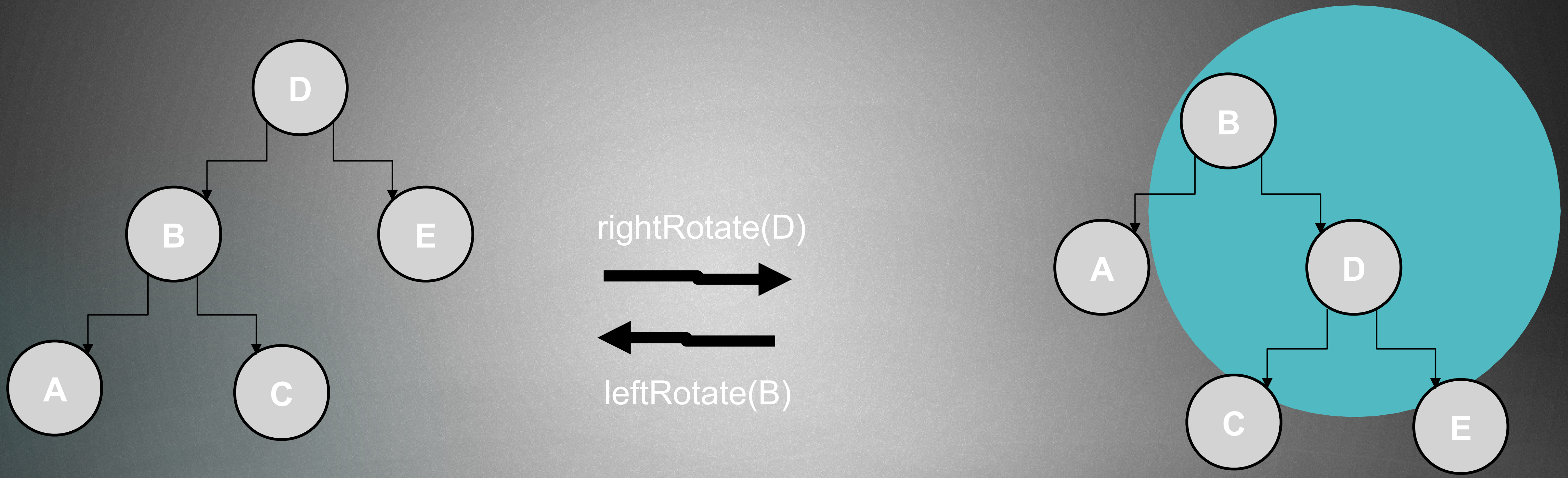
	Average case	Worst case
Space	O(n)	O(n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)
Search	O(log n)	O(log n)

Red-black properties:

- Each node is either red or black
- The root node is always **black**
- All leaves (NIL or NULL) are black
- Every red node must have two black child nodes and no other children → it must have a black parent
- Every path from a given node to any of its descendant NIL/NULL nodes contains the same number of black nodes


13
8
17
1
11
15
25
NIL
6
NIL
NIL
NIL
NIL
22
27
NIL
NIL
NIL
NIL
NIL
NIL

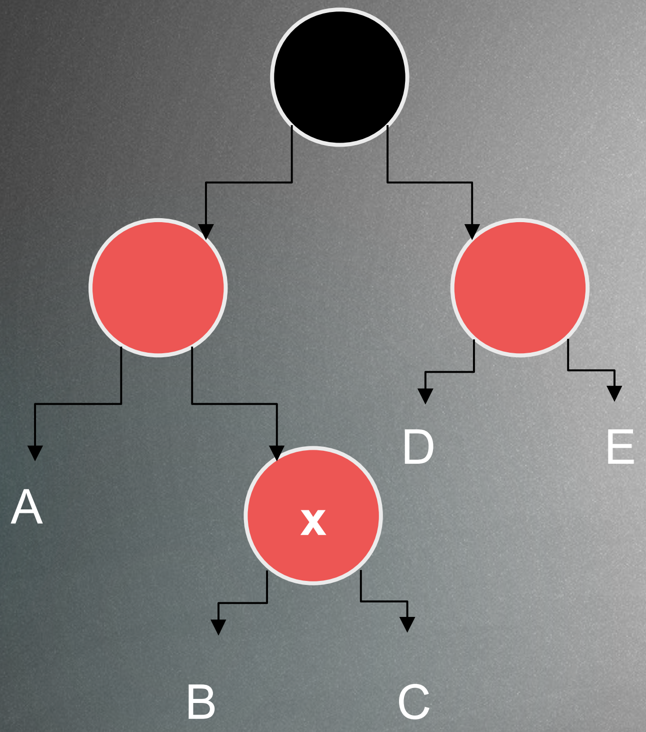
Rotations



We just have to update the references which can be done in **O(1)** time complexity !!! (the in-order traversal is the same)

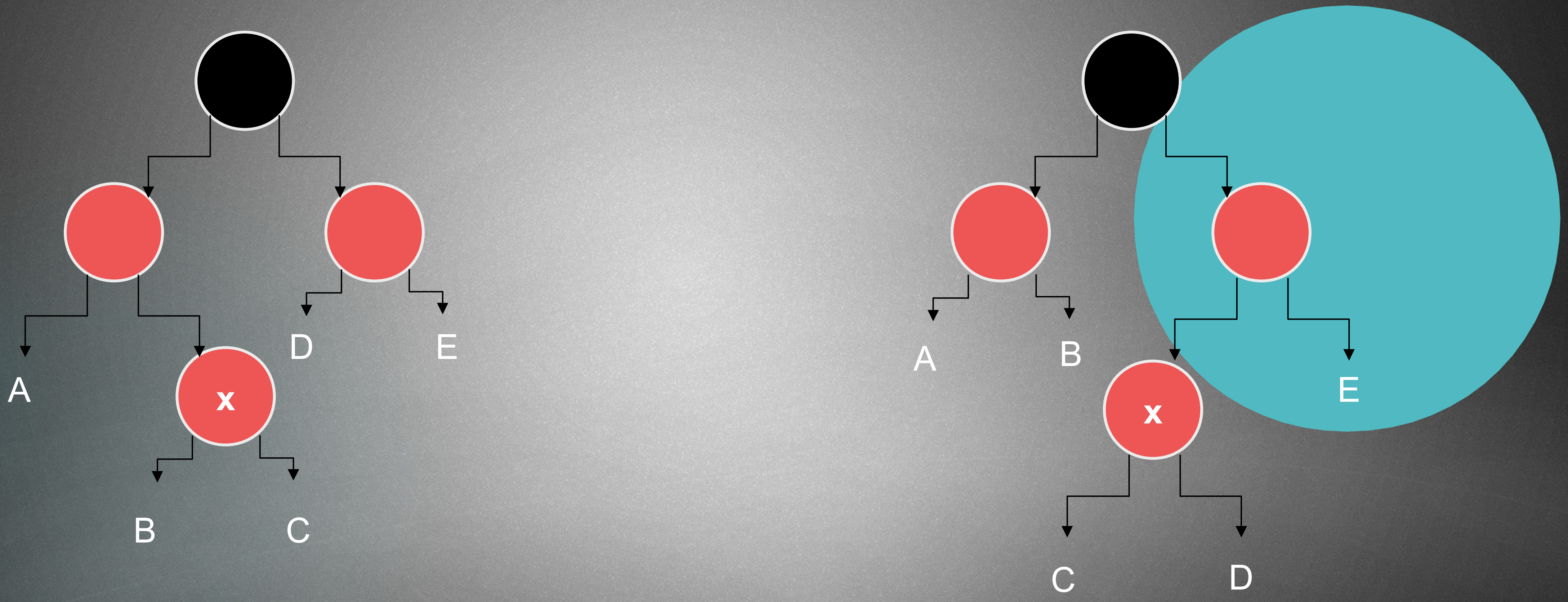
- Every new node is red by default
- We keep inserting new node in the same way as we have seen for binary search trees (or AVL trees)
- On every insertion → we have to check whether we have violated the red-black properties or not
- If we have violated the RB properties: we have to rebalance the tree
  - 1.) make rotations
  - 2.) OR just recolor the nodes

Case 1:



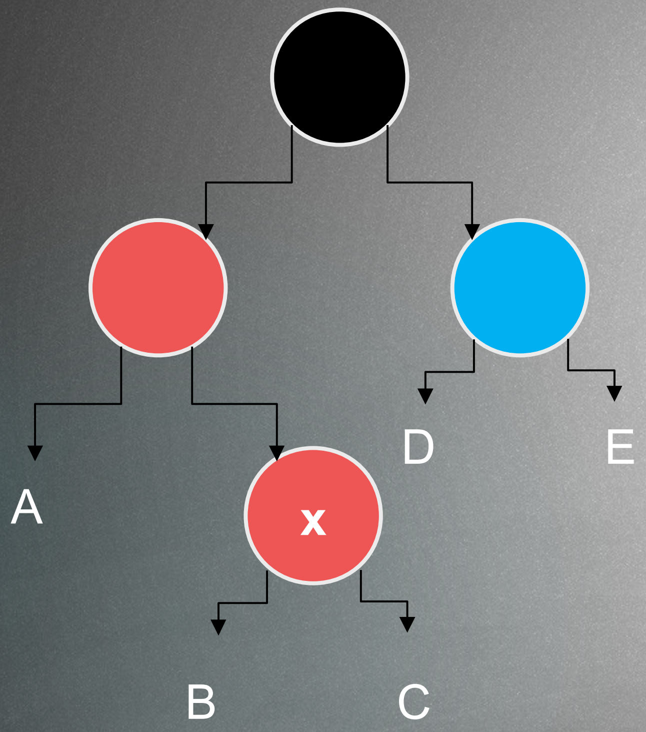
Case 1:

This problem is symmetric !!!



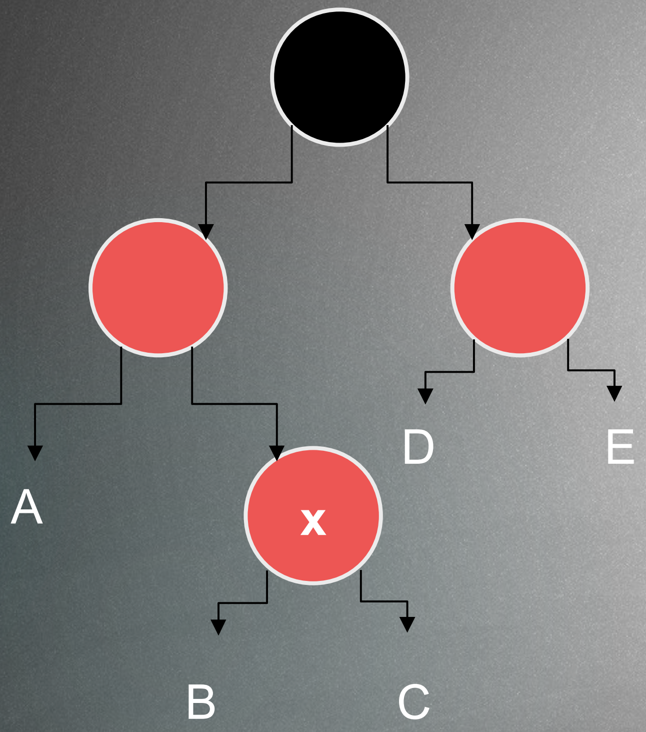
Case 1:

The „uncle” of x is red too !!!



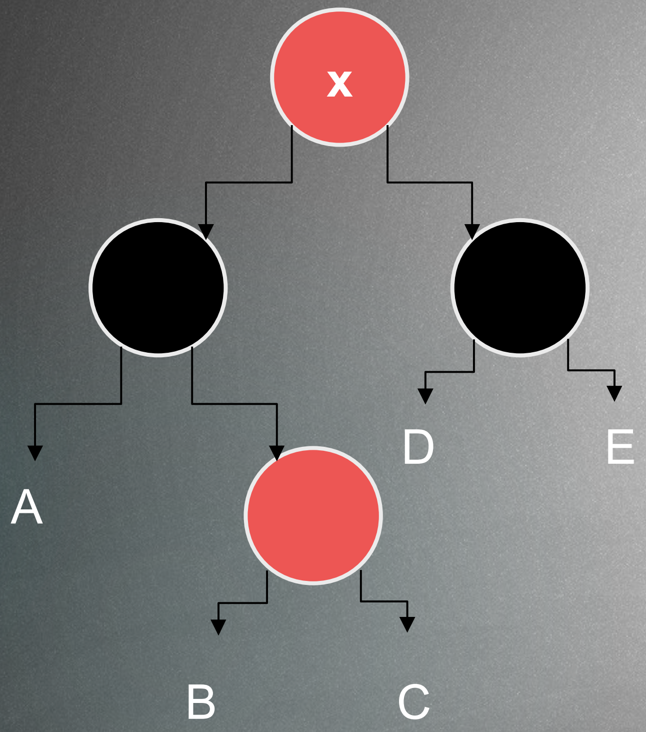
Case 1:

We just have to recolor some nodes, quite easy case



Case 1:

We just have to recolor some nodes, quite easy case + the **x** will be the root node in this case

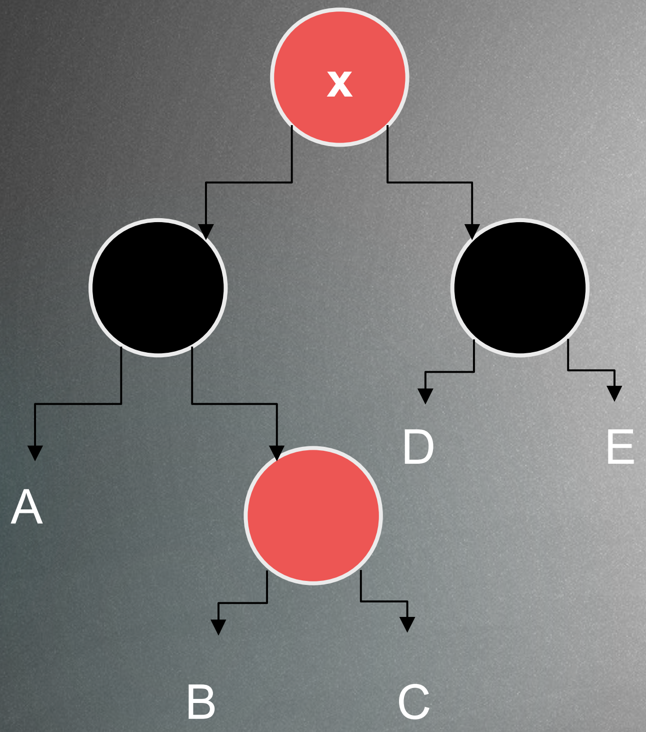


IMPORTANT: with this recoloring, maybe we violate the red-black properties in other parts of the tree !!!

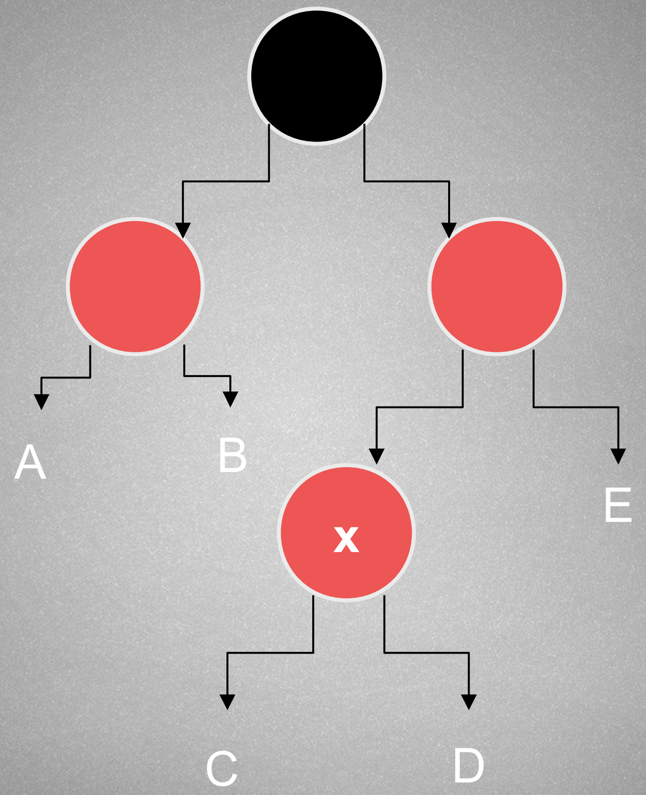
~ have to check recursively on the whole tree

Case 1:

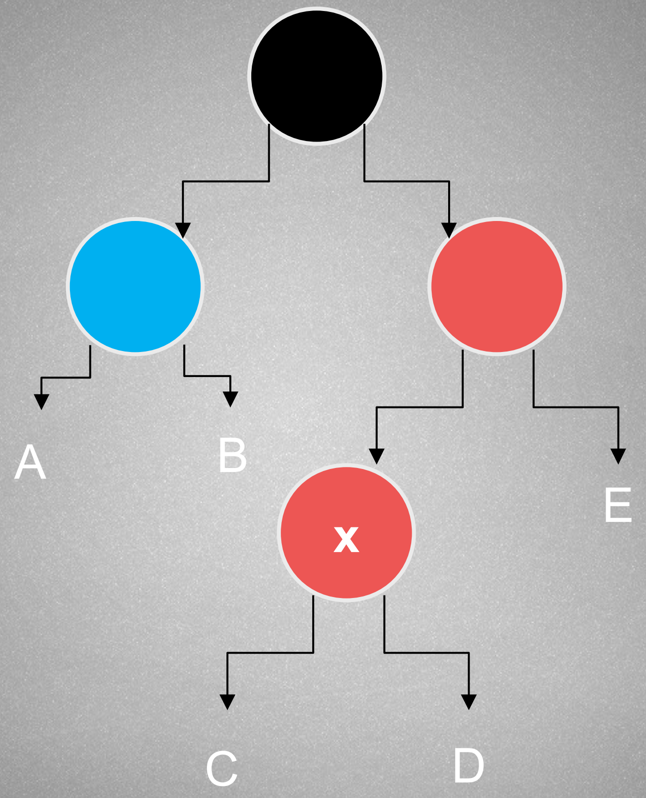
We have to check recursively (fom bottom to top) whether the red-black properties are violated or not



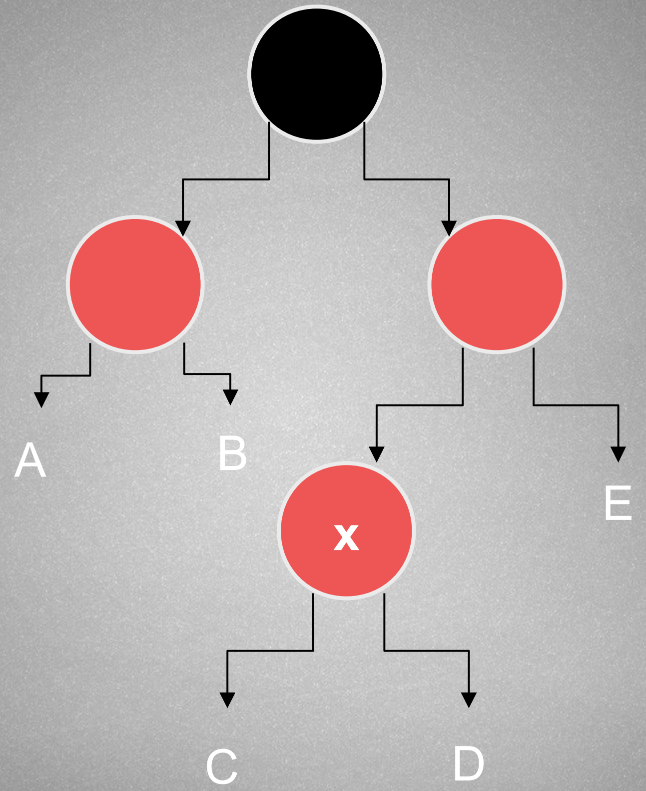
Case 1:



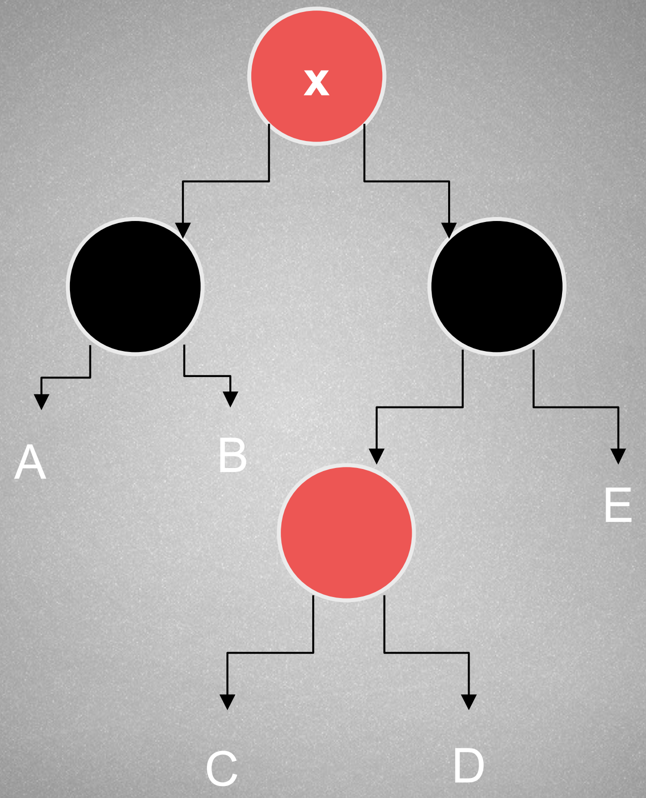
Case 1:



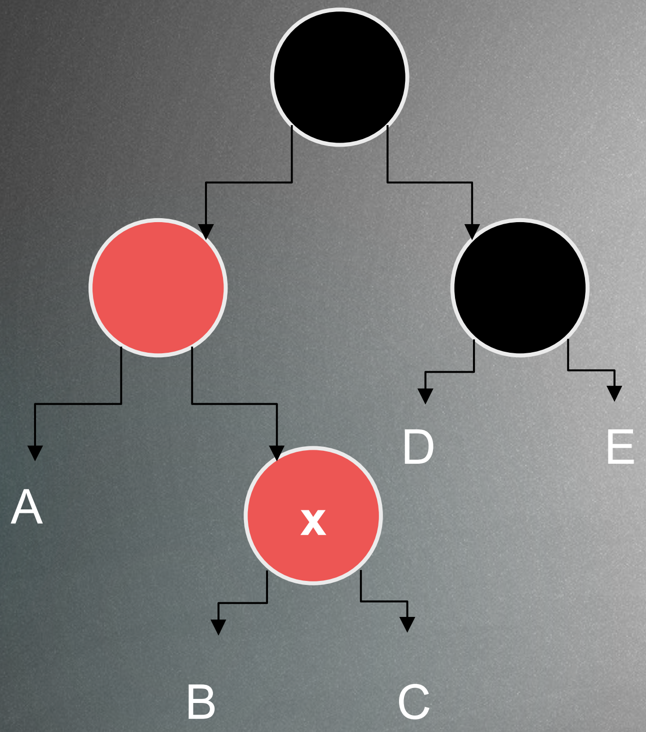
Case 1:



Case 1:

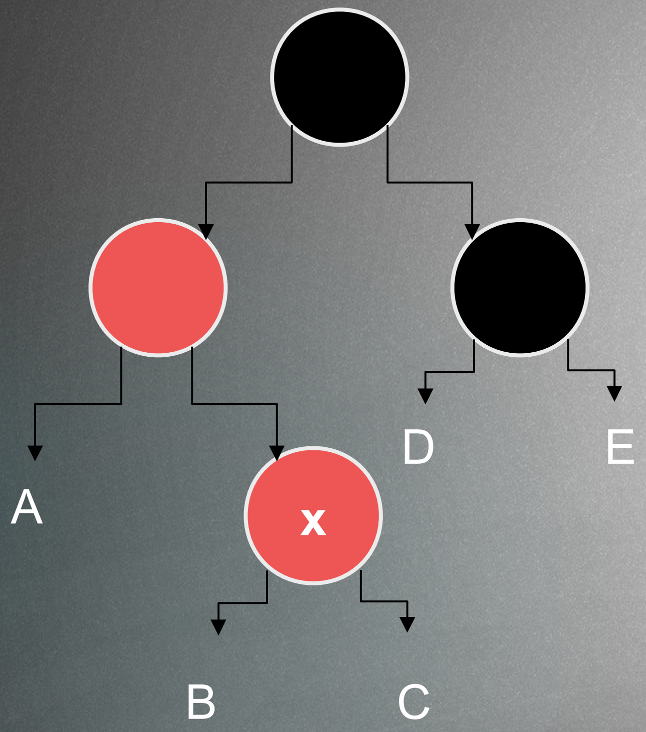


Case 2:



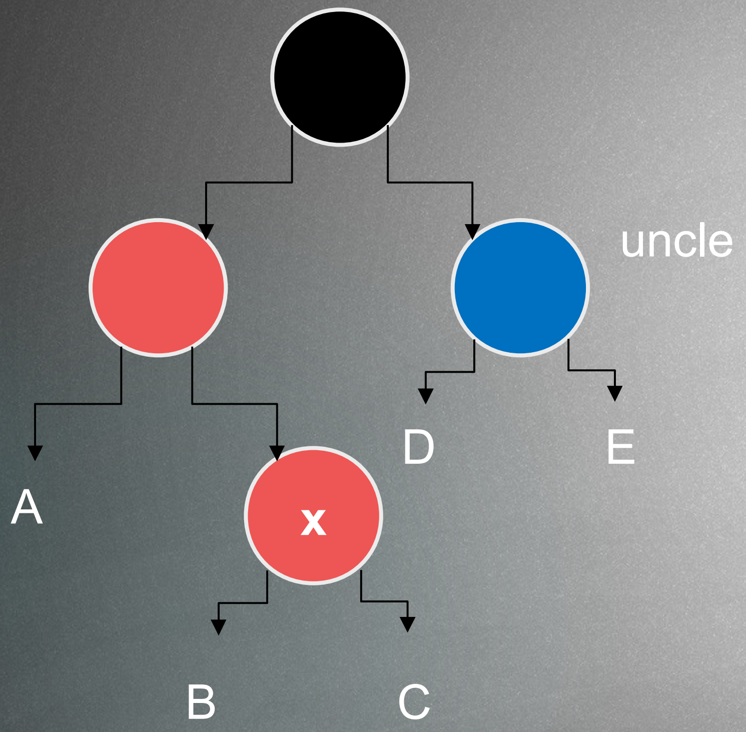
Case 2:

The uncle of node **x** is a black node + node **x** is a right child

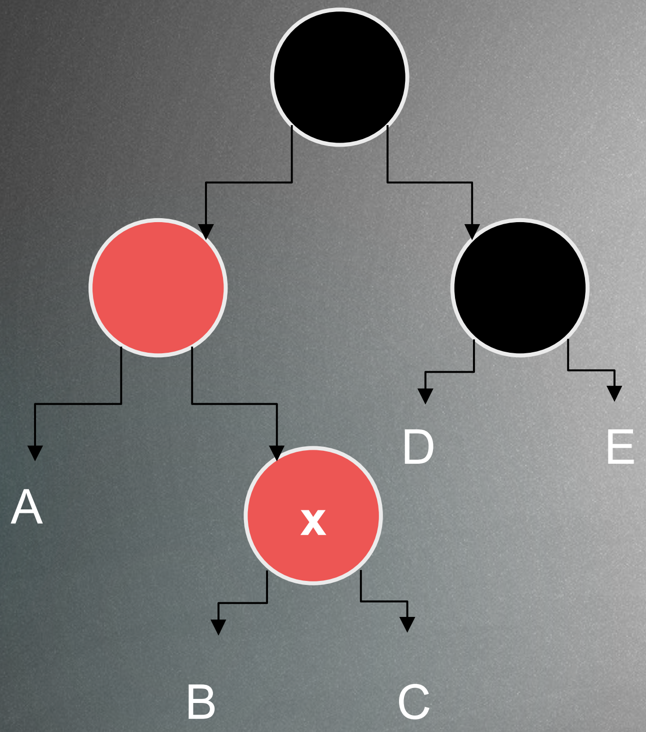


Case 2:

The uncle of node x is a black node + node x is a right child

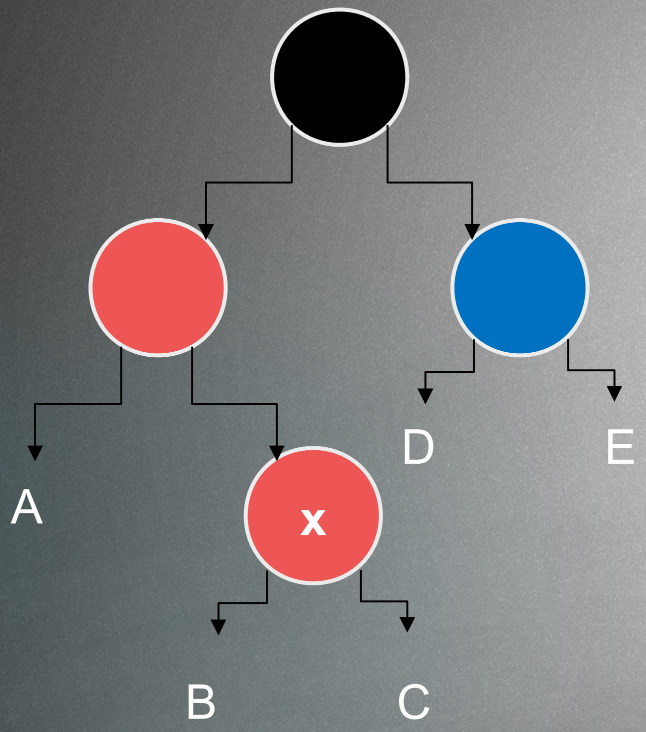


Case 2:

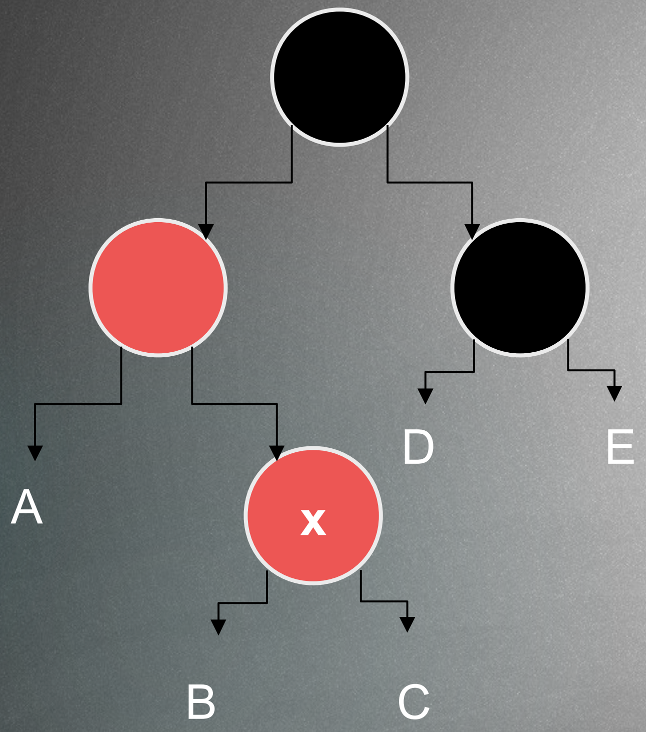


<u>Case 2:</u>

The uncle of node **x** is a black node + node **x** is a right child
We have to make a left rotation on the node **x**’s parent

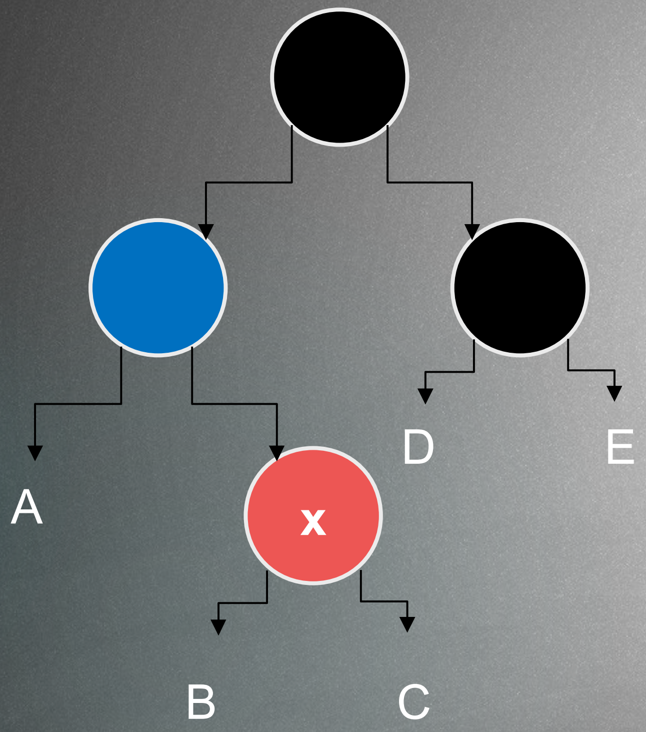


Case 2:



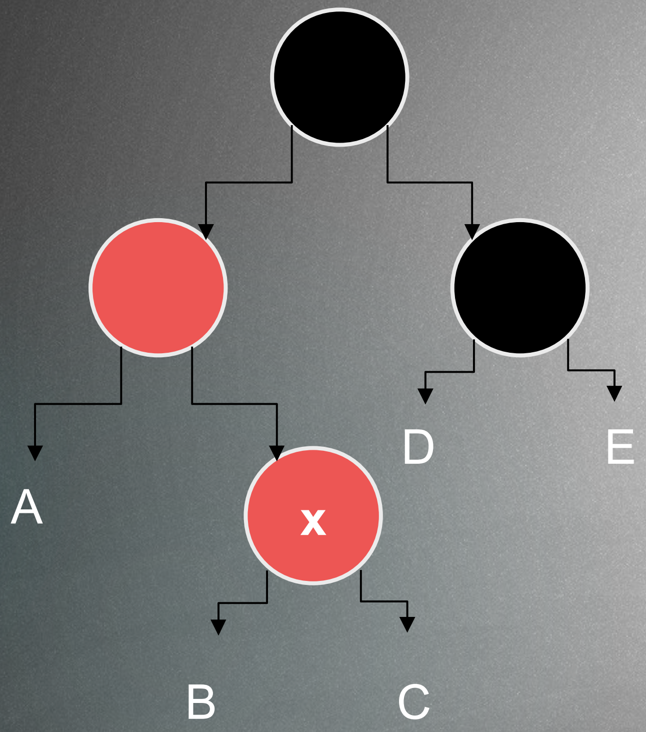
Case 2:

The uncle of node **x** is a black node + node x is a right child
We have to make a left rotation on the node **x**'s parent



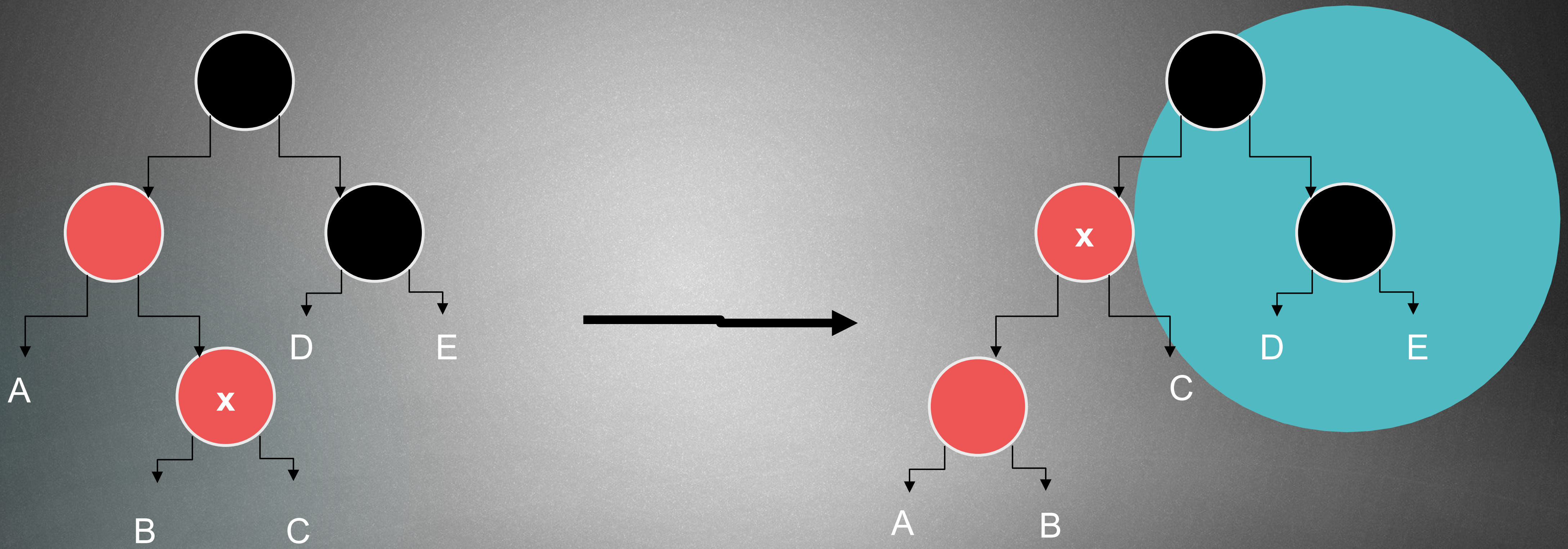
We have to rotate the blue node !!!

Case 2:



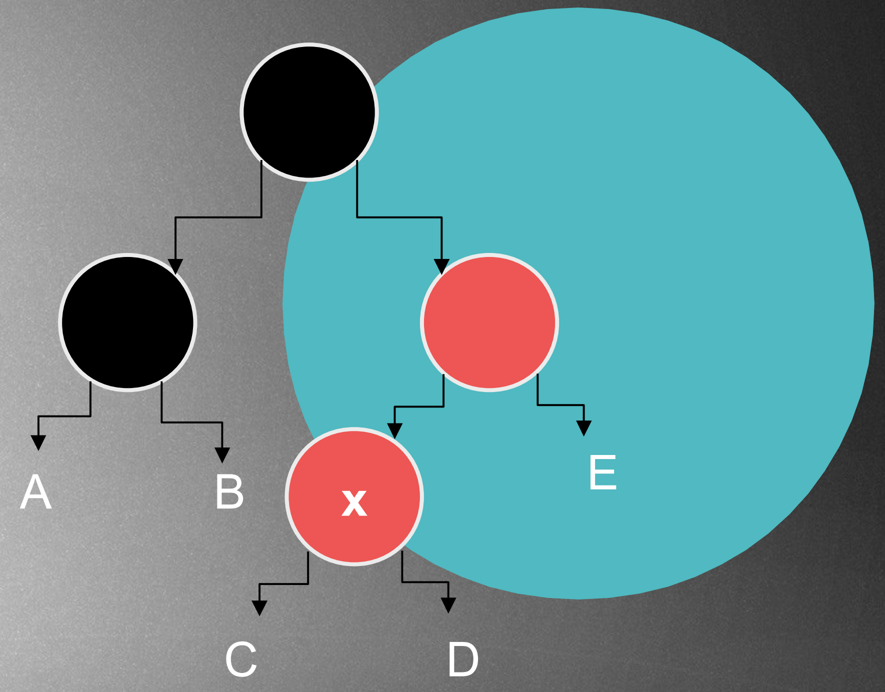
Case 2:

The uncle of node **x** is a black node + node **x** is a right child
We have to make a left rotation on the node **x**'s parent



Case 2:

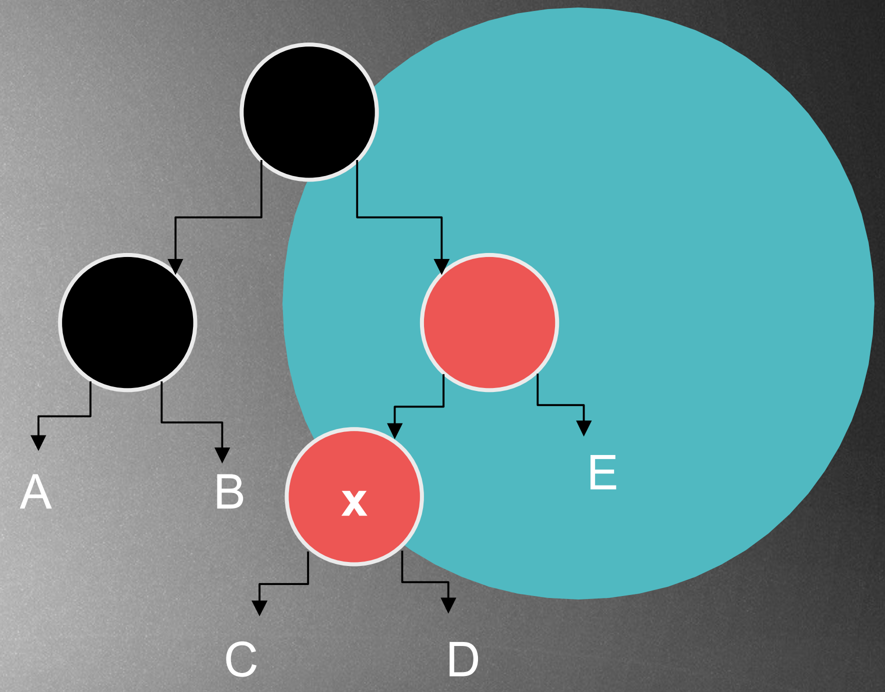
same problem basically but the symmetric version



Case 2:

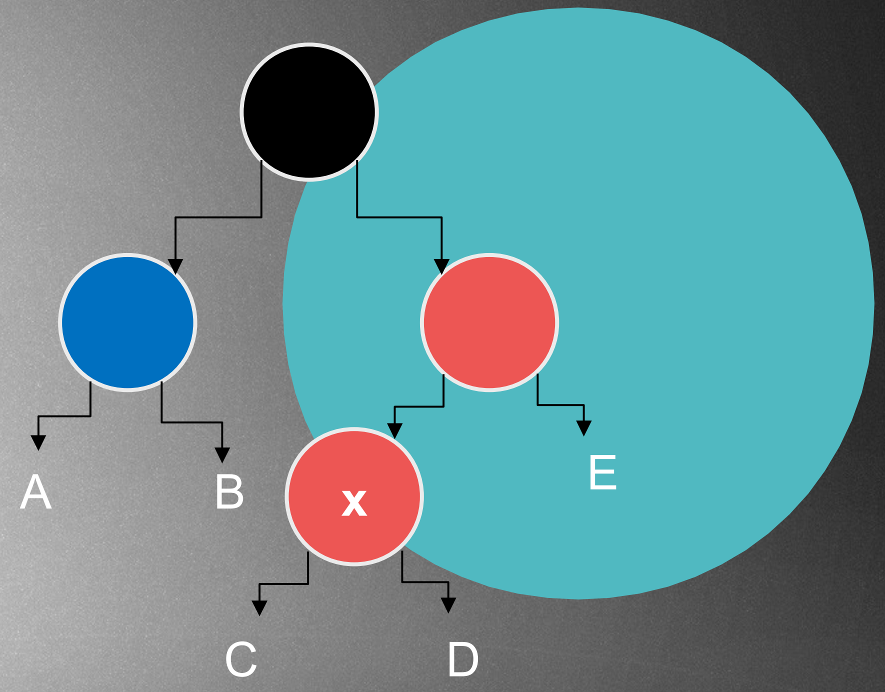
The given **x** node is a left child

+ the parent is red
+ the uncle is black



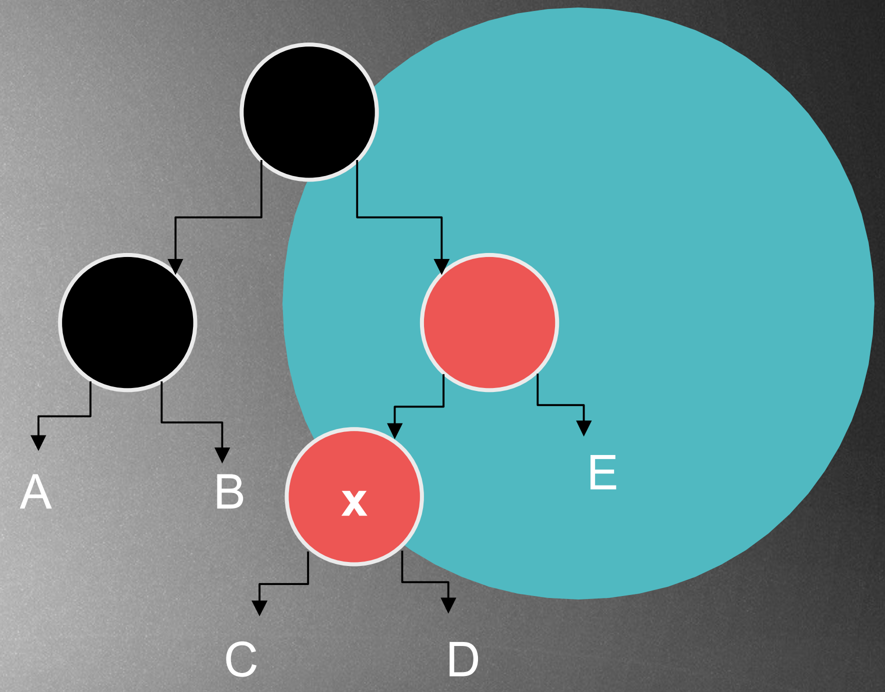
Case 2:

The given **x** node is a left child
+ the parent is red
+ the uncle is black



Case 2:

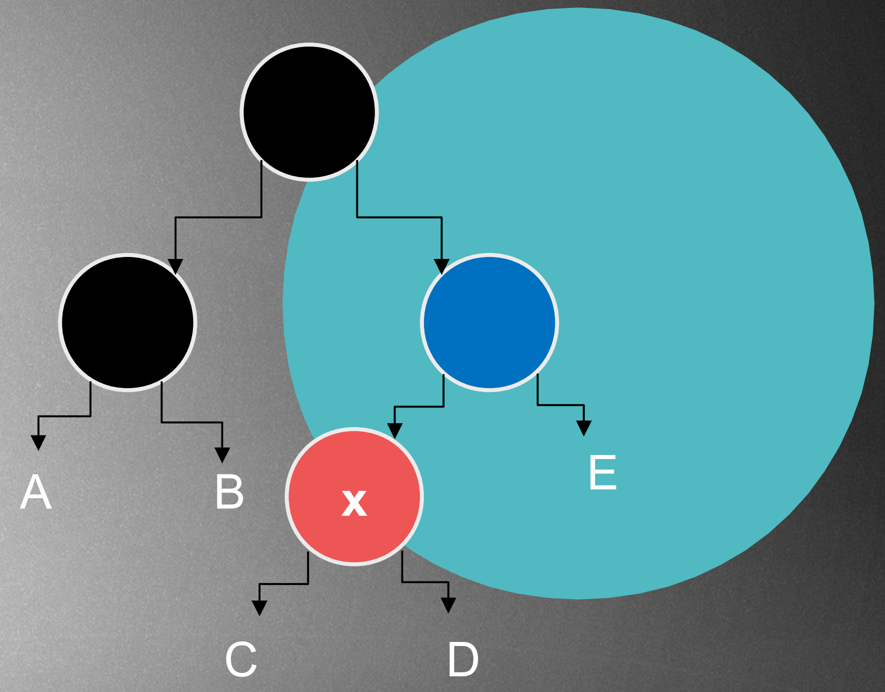
The given **x** node is a left child
+ the parent is red
+ the uncle is black



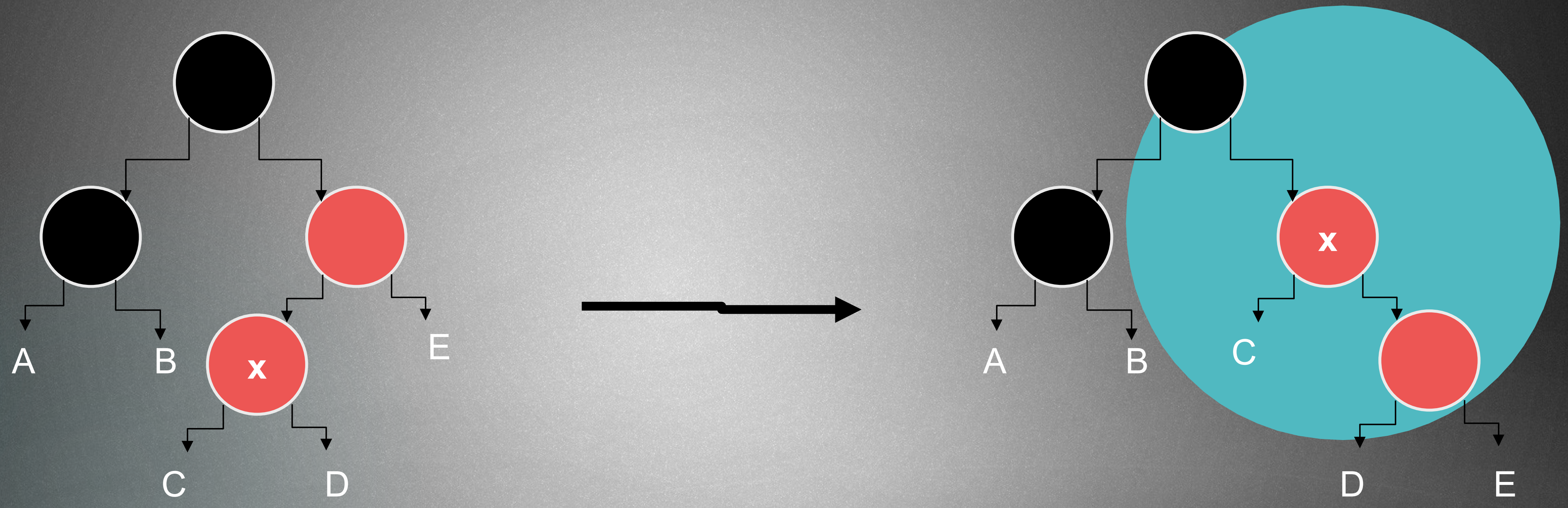
Case 2:

The given x node is a left child
+ the parent is red
+ the uncle is black

We just have to make a right rotation
on the blue node !!!

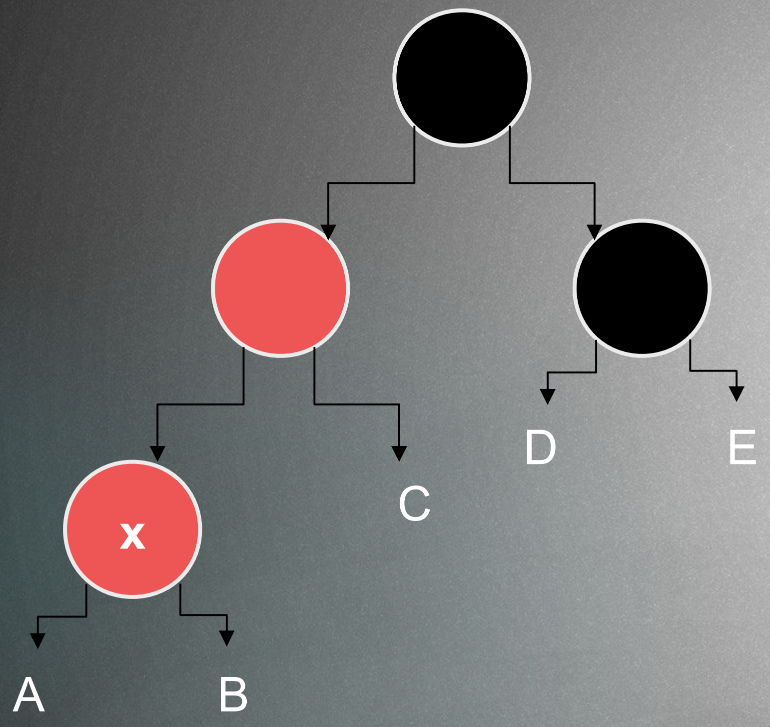


Case 2:



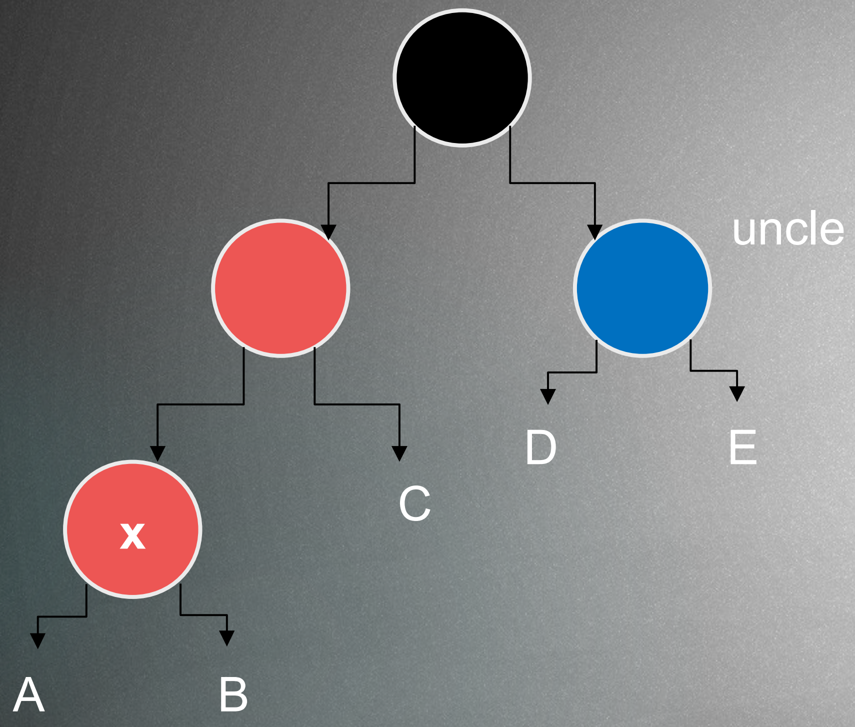
Case 3:

Uncle of node **x** is black and node **x** is a left child



Case 3:

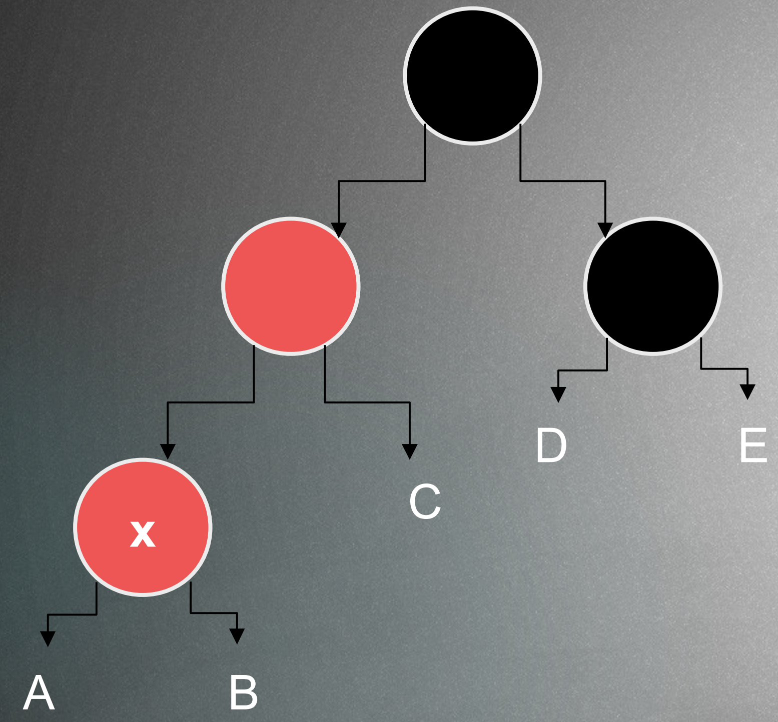
Uncle of node **x** is black and node **x** is a left child



Case 3:

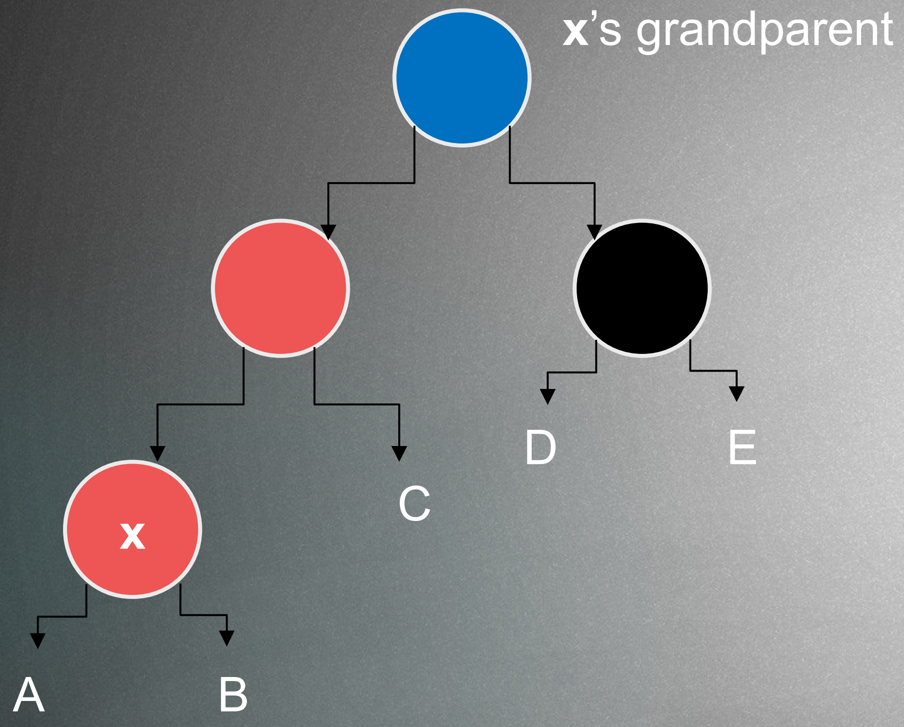
Uncle of node x is black and node **x** is a left child

We have to make a rotation on **x** node's grandparent



Case 3:

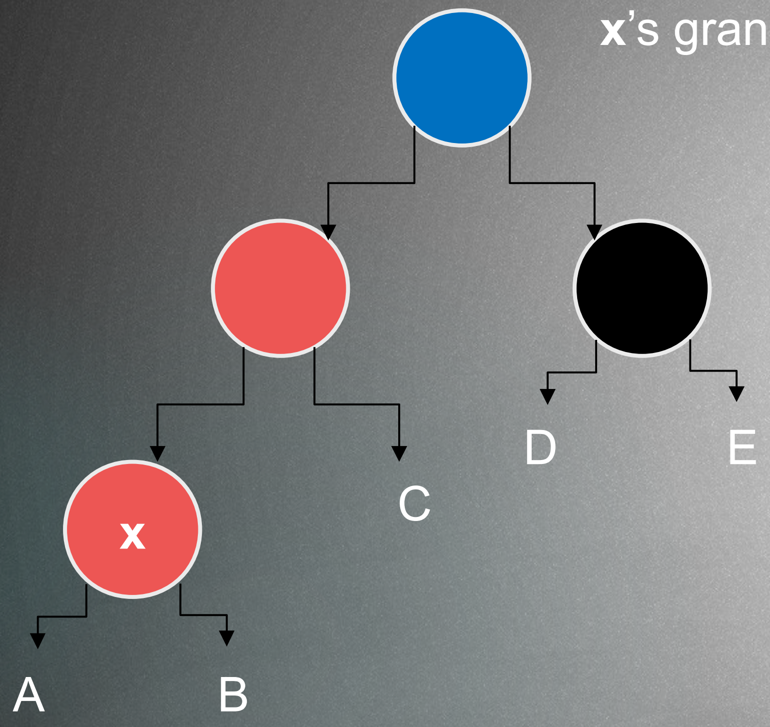
Uncle of node **x** is black and node **x** is a left child
We have to make a rotation on **x** node's grandparent



Case 3:

Uncle of node **x** is black and node **x** is a left child
We have to make a rotation on **x** node's grandparent

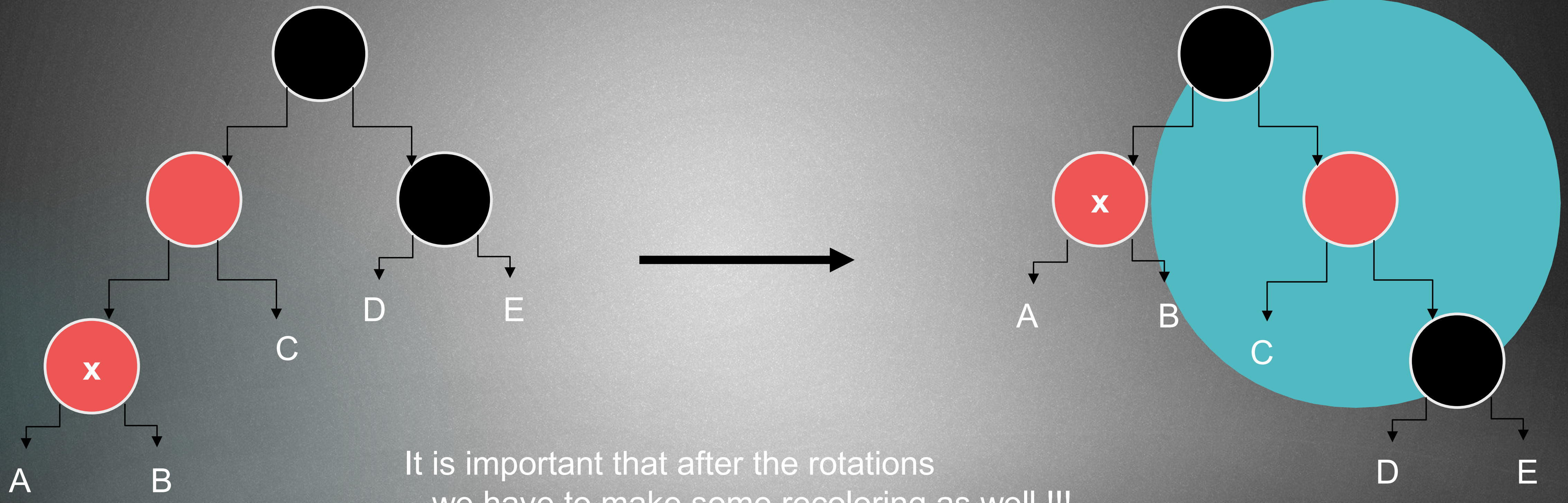
x's grandparent: we have to rotate this node



Case 3:

Uncle of node **x** is black and node **x** is a left child

We have to make a rotation on **x** node's grandparent

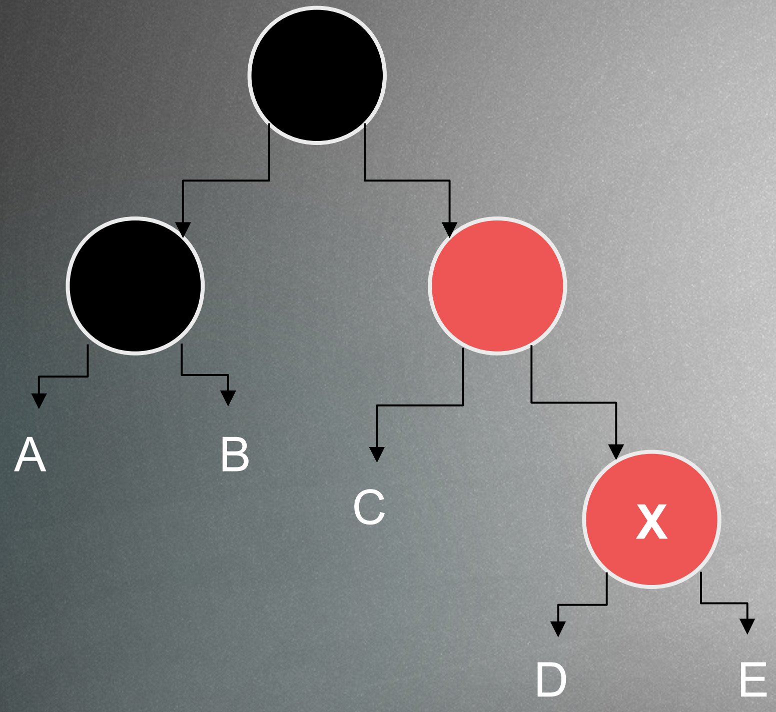


It is important that after the rotations

we have to make some recoloring as well !!!

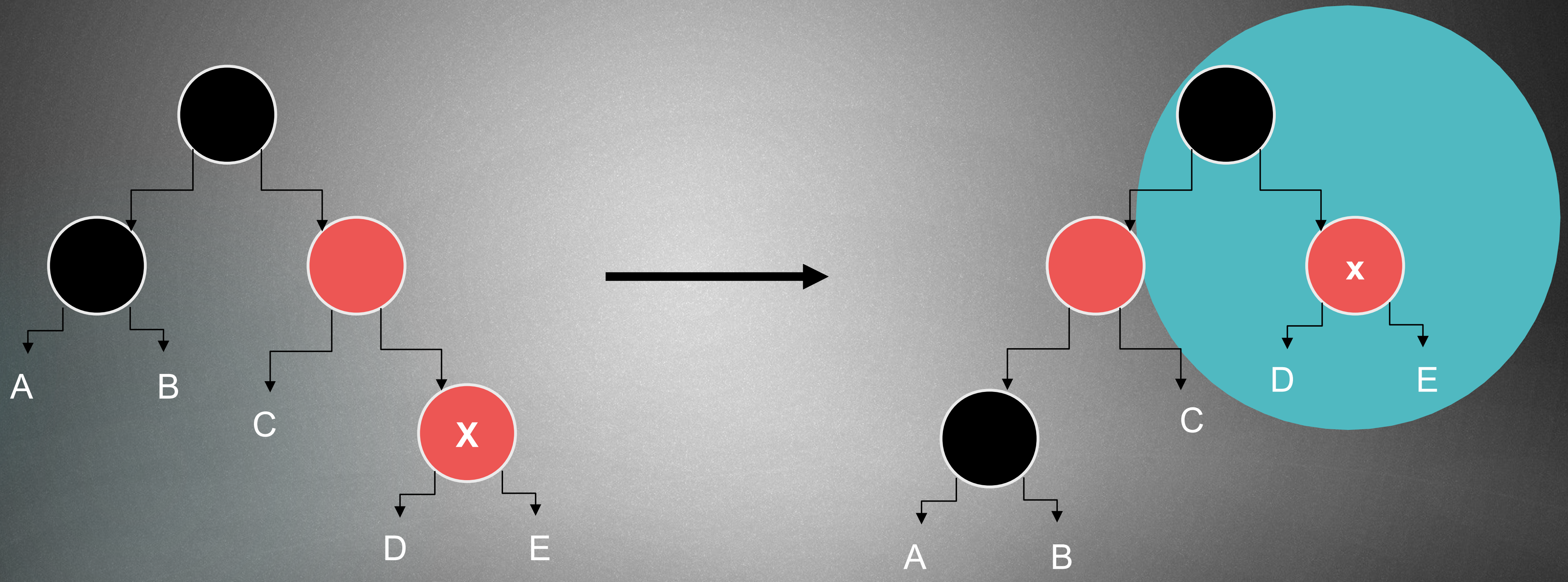
Case 3:

The symmetric version is basically the same just rotate in the opposite direction



Case 3:

The symmetric version is basically the same just rotate in the opposite direction

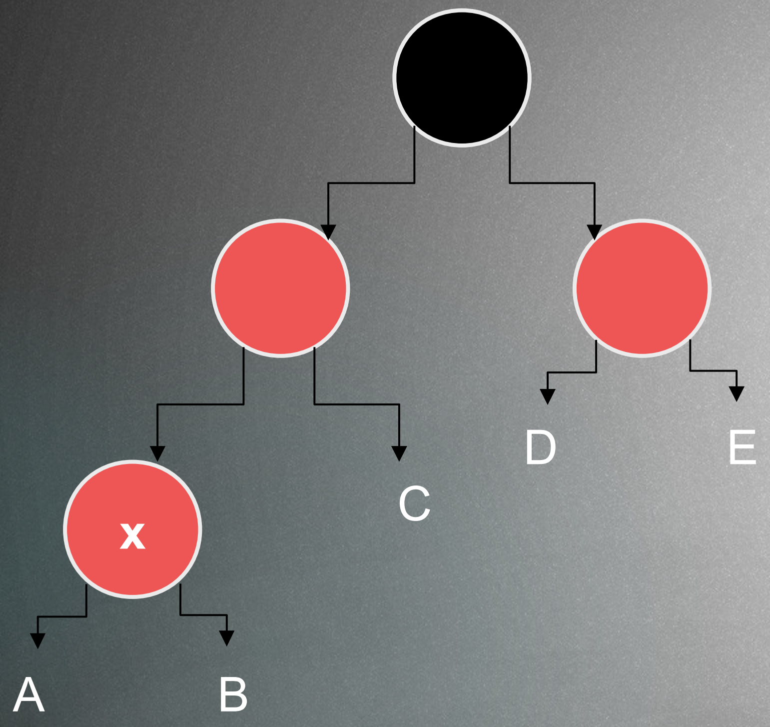


Case 4:

Uncle of node **x** is black and node **x** is a left child

+ uncle is red here in this case

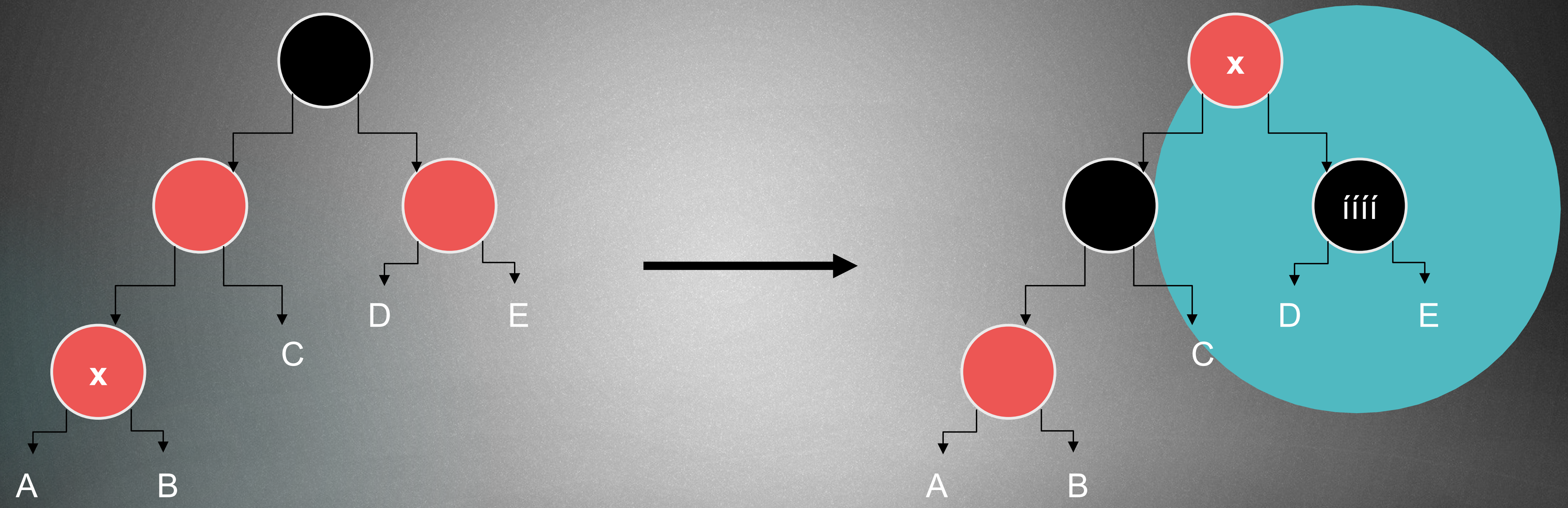
We have to recolor some nodes !!!



Case 4:

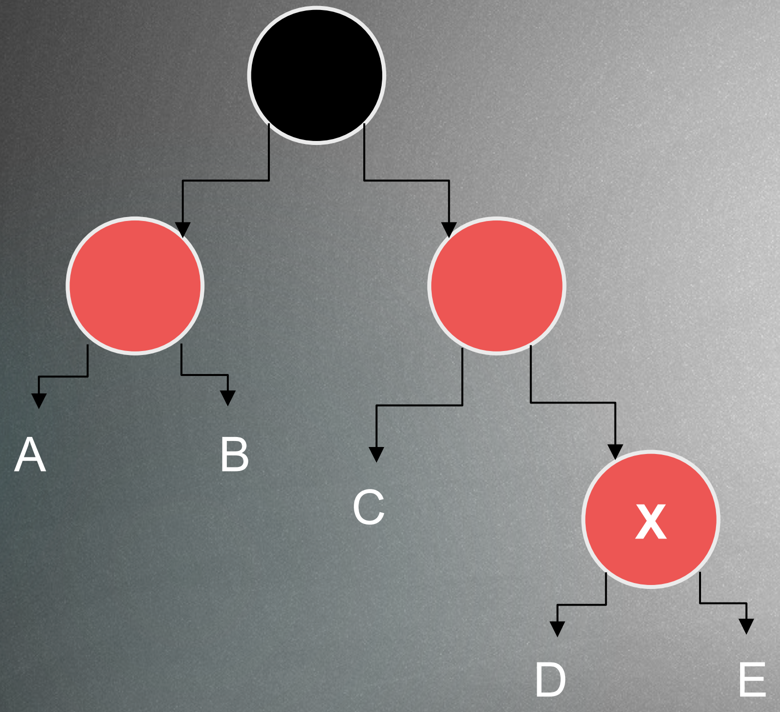
Uncle of node **x** is black and node **x** is a left child

We have to recolor some nodes



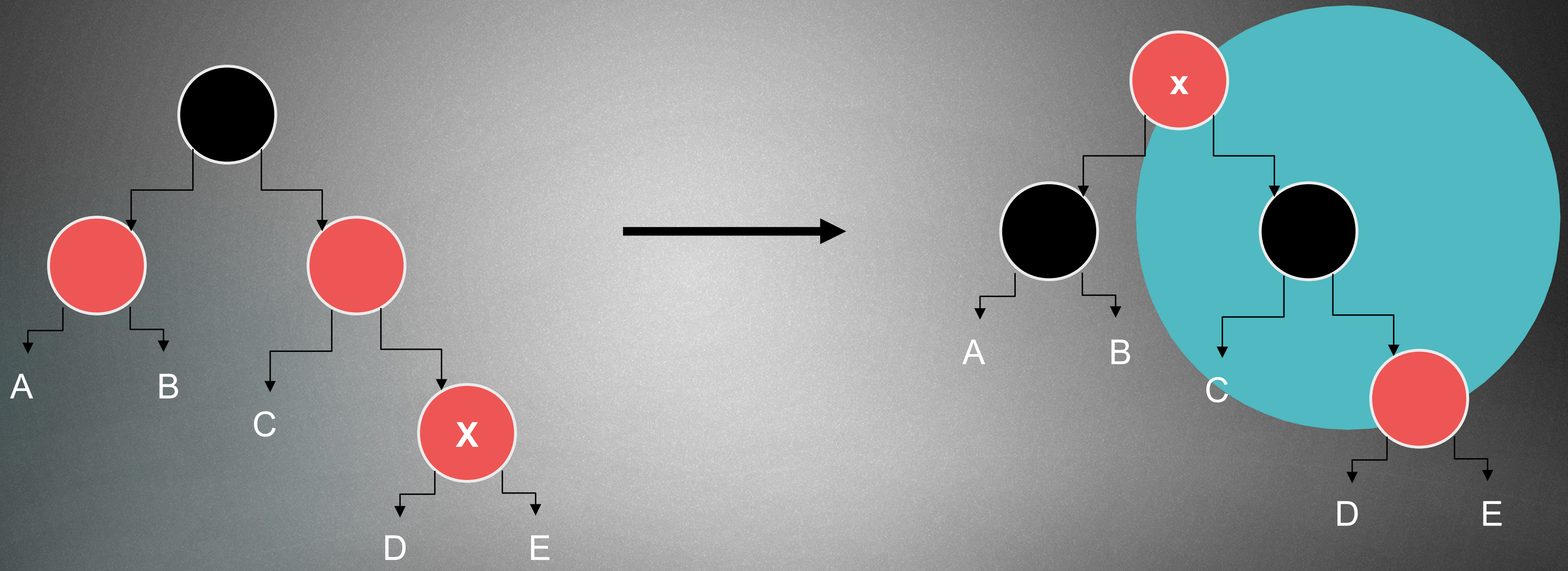
Case 4:

This is the same problem but the symmetric version !!!



Case 4:

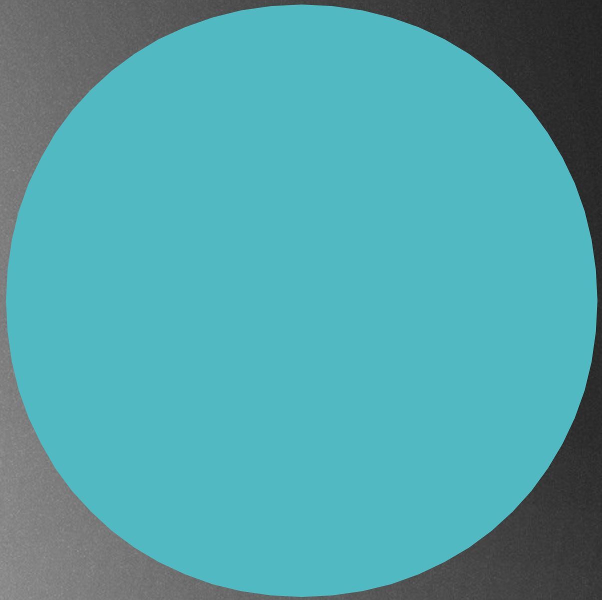
This is the same problem but the symmetric version !!!



Example:

Example:

insert(1)

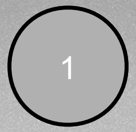


Example:

insert(1)

Example:

insert(1)

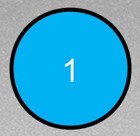


Example:

insert(2)

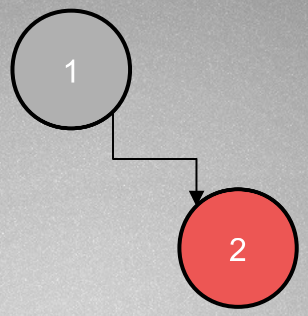
Example:

insert(2)



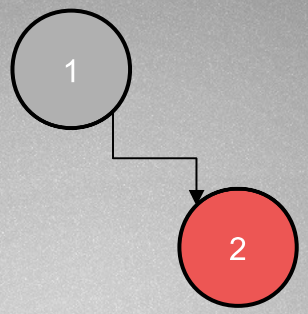
Example:

insert(2)



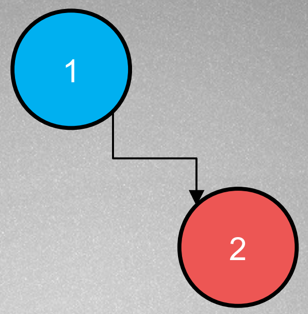
Example:

insert(3)



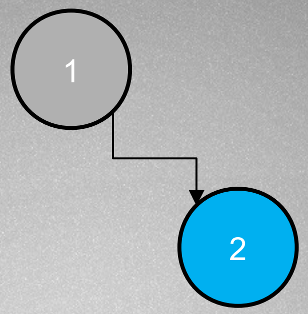
Example:

insert(3)

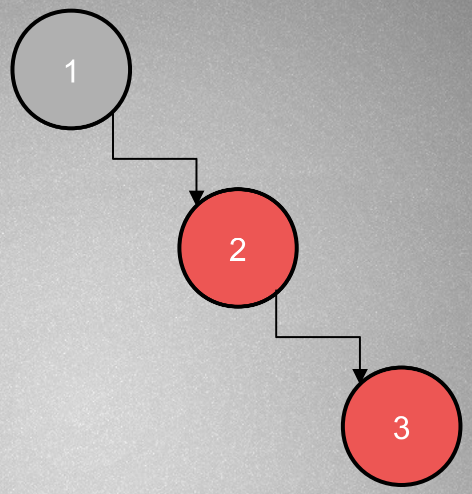


Example:

insert(3)



Example:

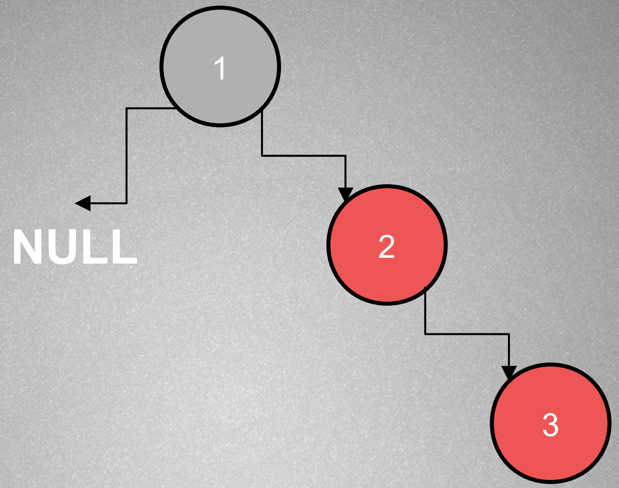


It is the Case 3: because the NULL is considered to be a black node

→ the uncle of Node 3 is black

→ Have to make a rotation + recolor the nodes if necessary

Example:



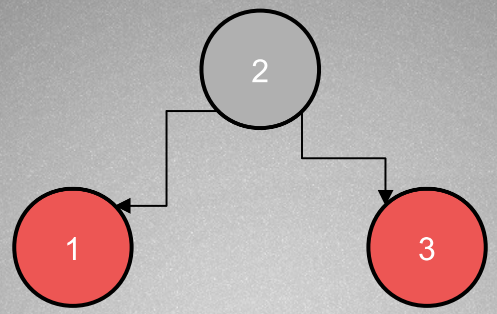
It is the Case 3: because the NULL is considered to be a black node

→ the uncle of Node 3 is black

→ Have to make a rotation + recolor the nodes if necessary

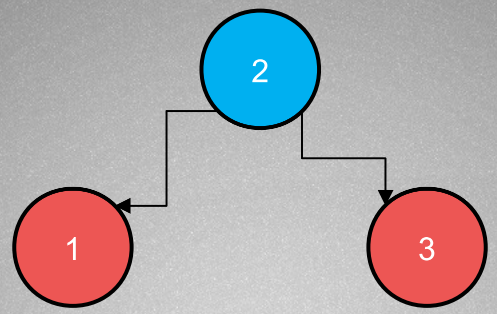
Example:

insert(4)



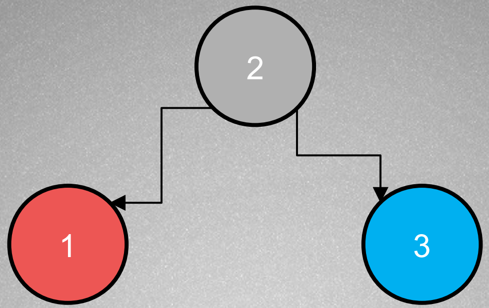
Example:

insert(4)

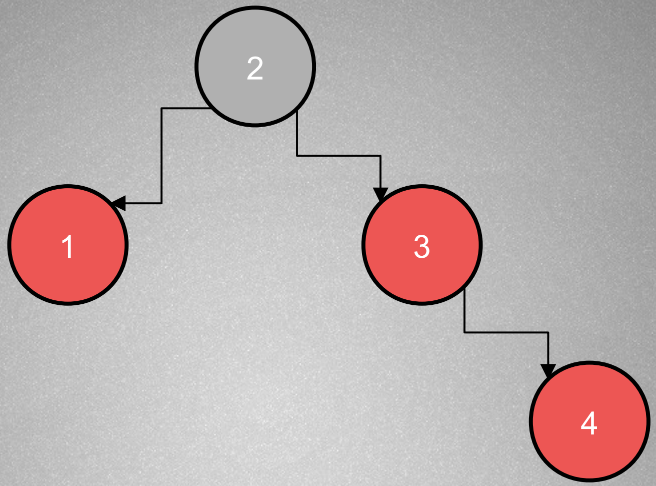


Example:

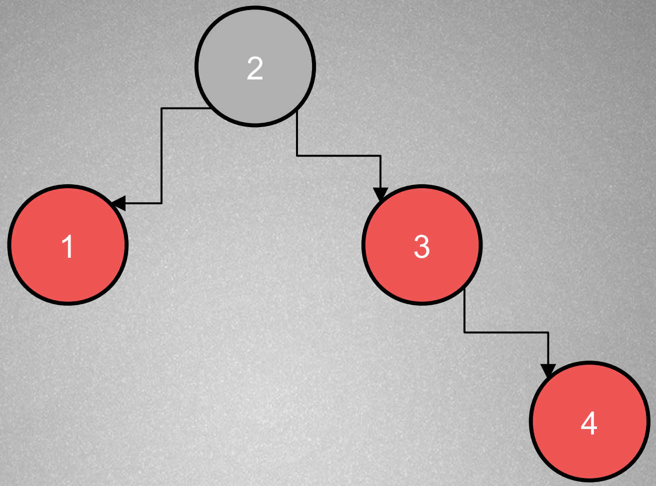
insert(4)



Example:



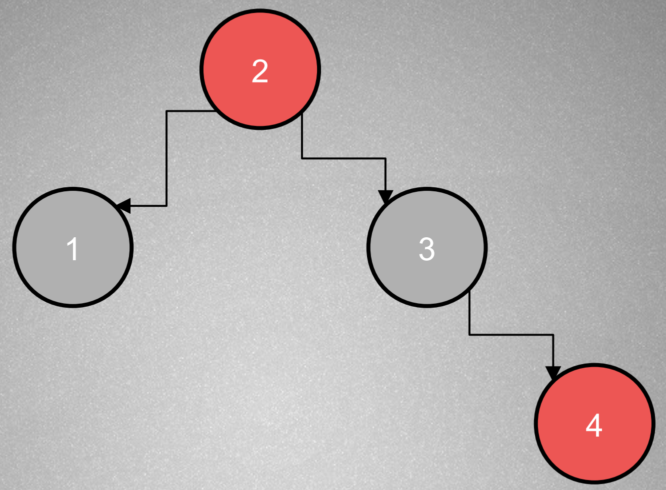
Example:



It is the Case 1: the given node 4 and the parent are both red + uncle is red

Color uncle + parent to be black

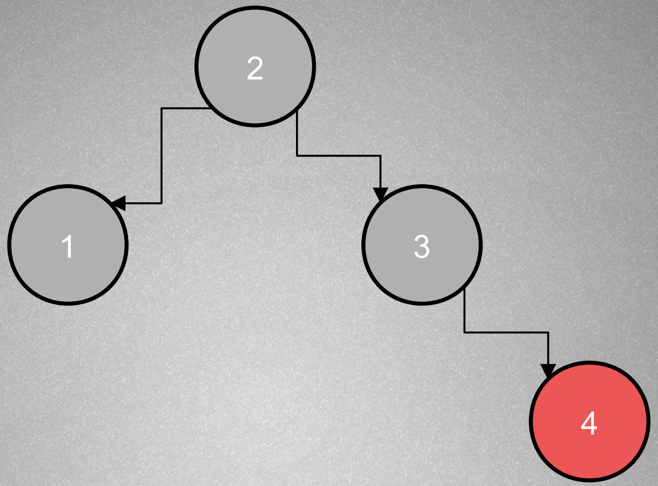
Example:



It is the Case 1: the given node 4 and the parent are both red + uncle is red

Color uncle + parent to be black

Example:



It is the Case 1: the given node 4 and the parent are both red + uncle is red

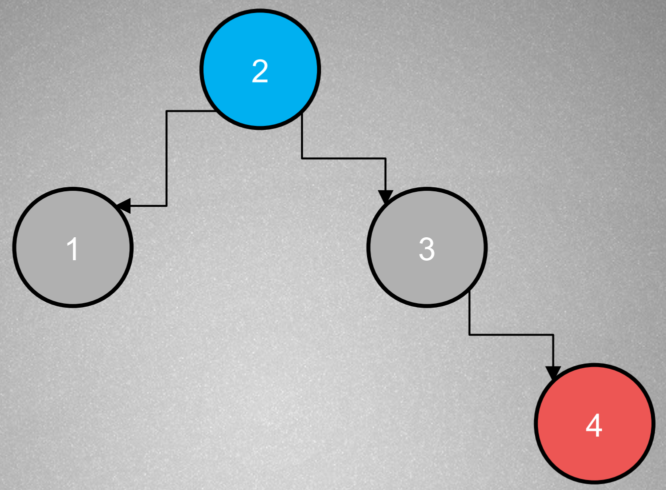
Color uncle + parent to be black

VIOLATES THE PROPERTIES AGAIN

Root has to be black as well

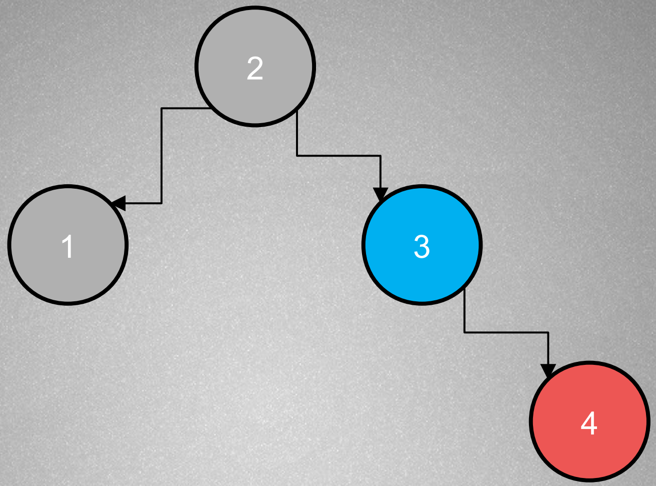
Example:

insert(5)



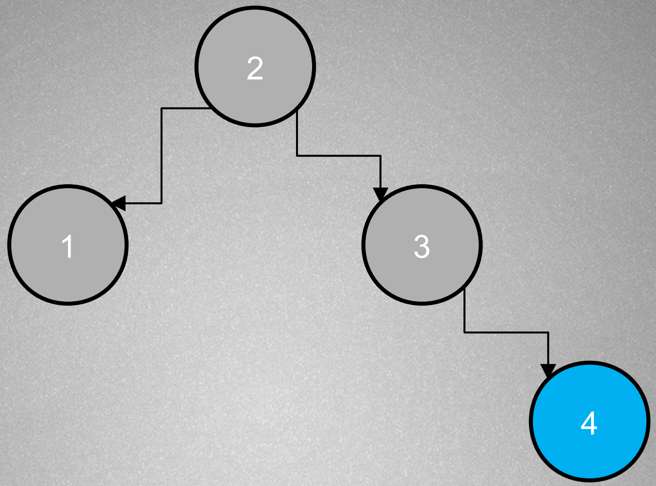
Example:

insert(5)



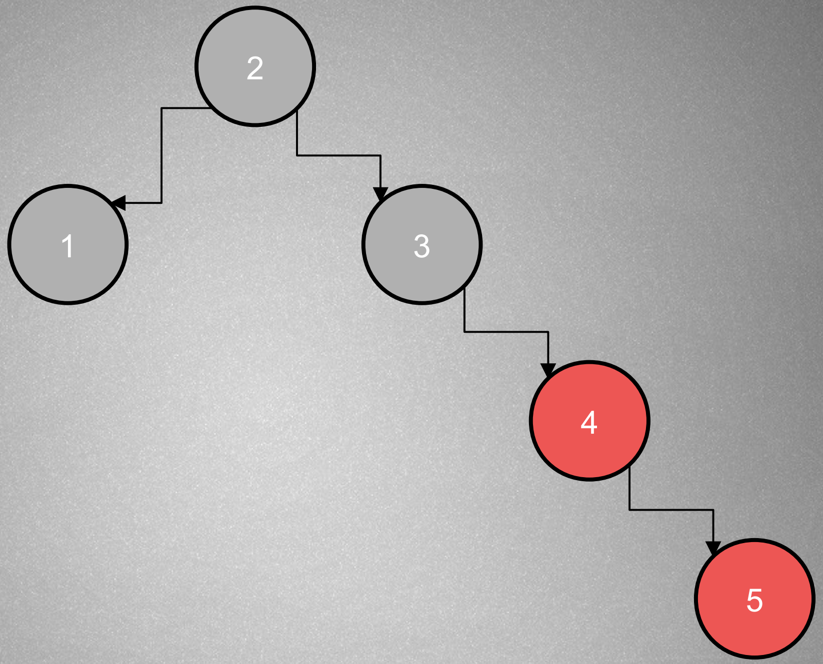
Example:

insert(5)



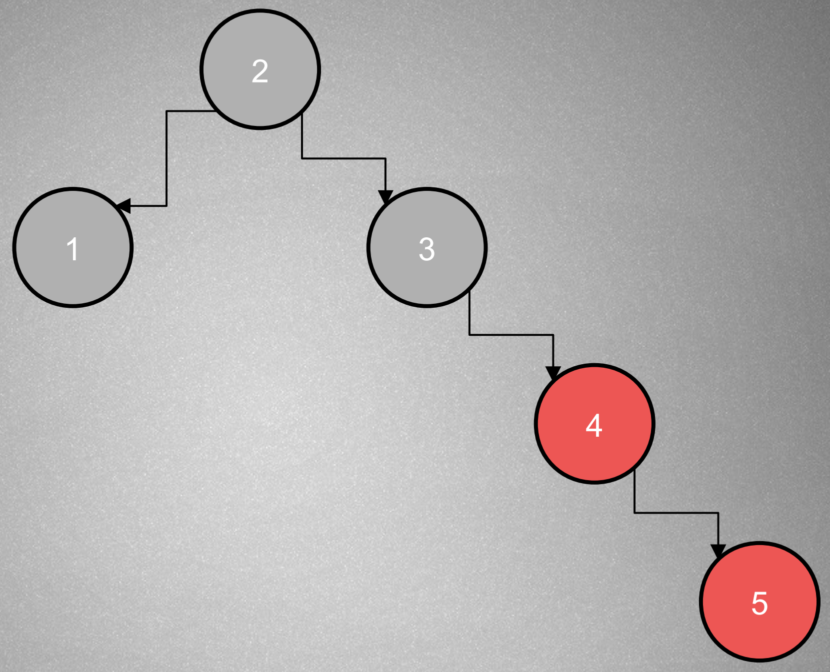
Example:

insert(5)

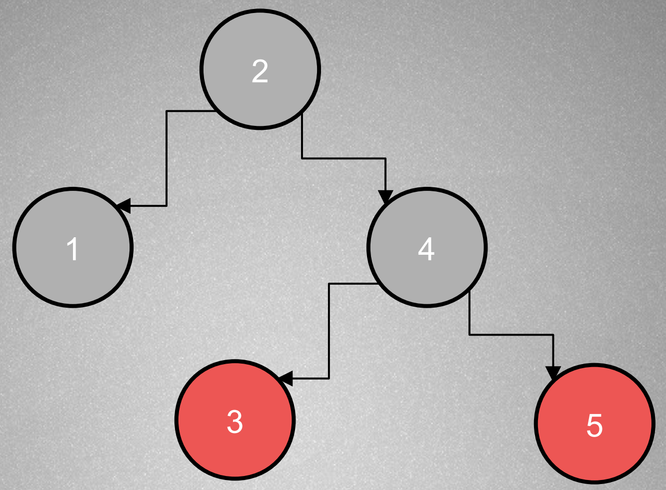


Example:

This is the Case 3 situation, we have to make some rotations !!!

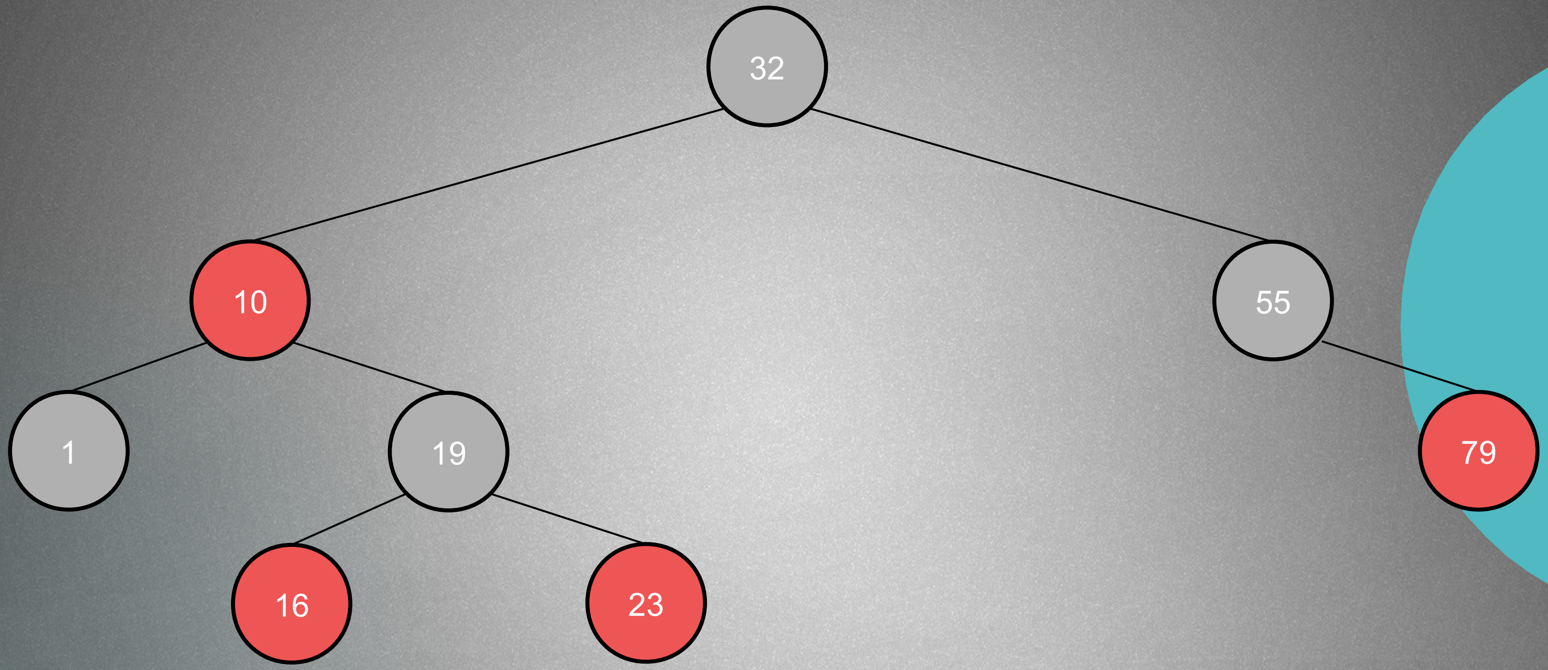


Example:



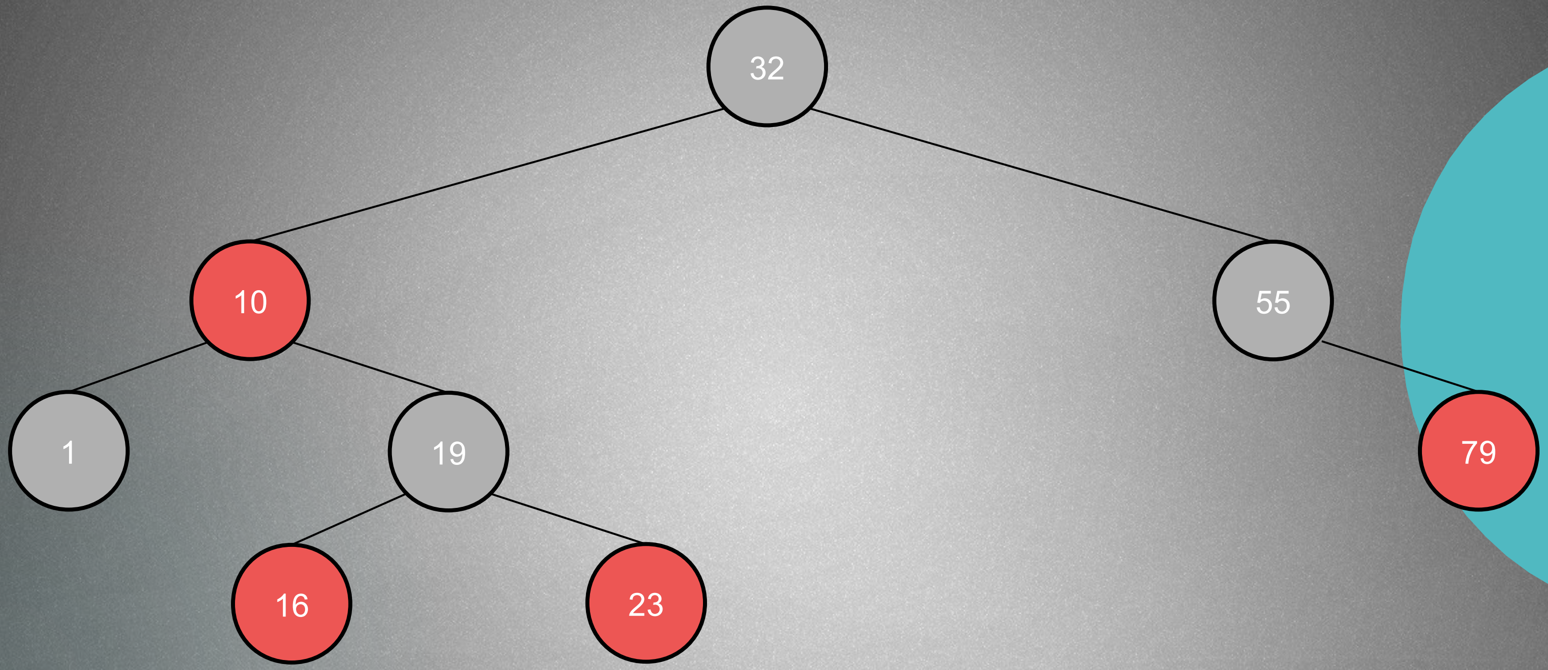
Example:

Example:



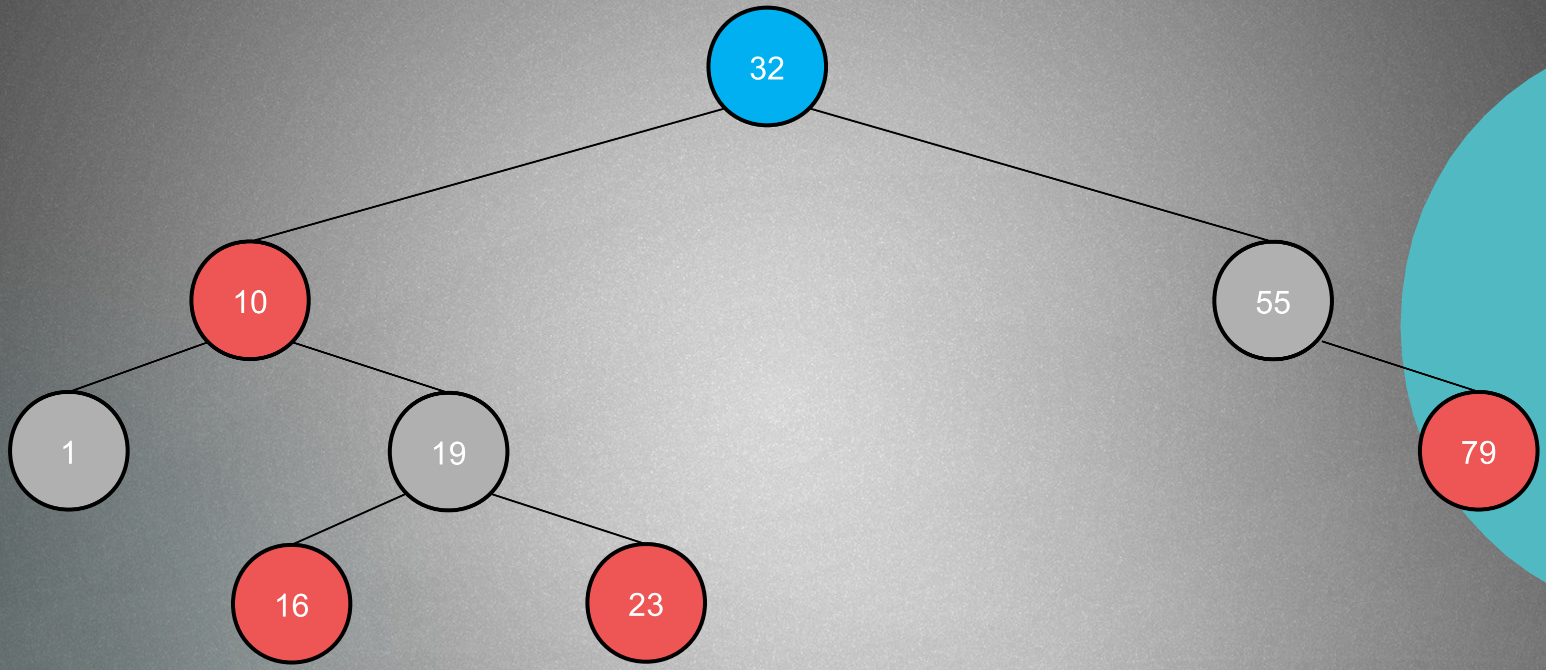
Example:

We want to insert 12



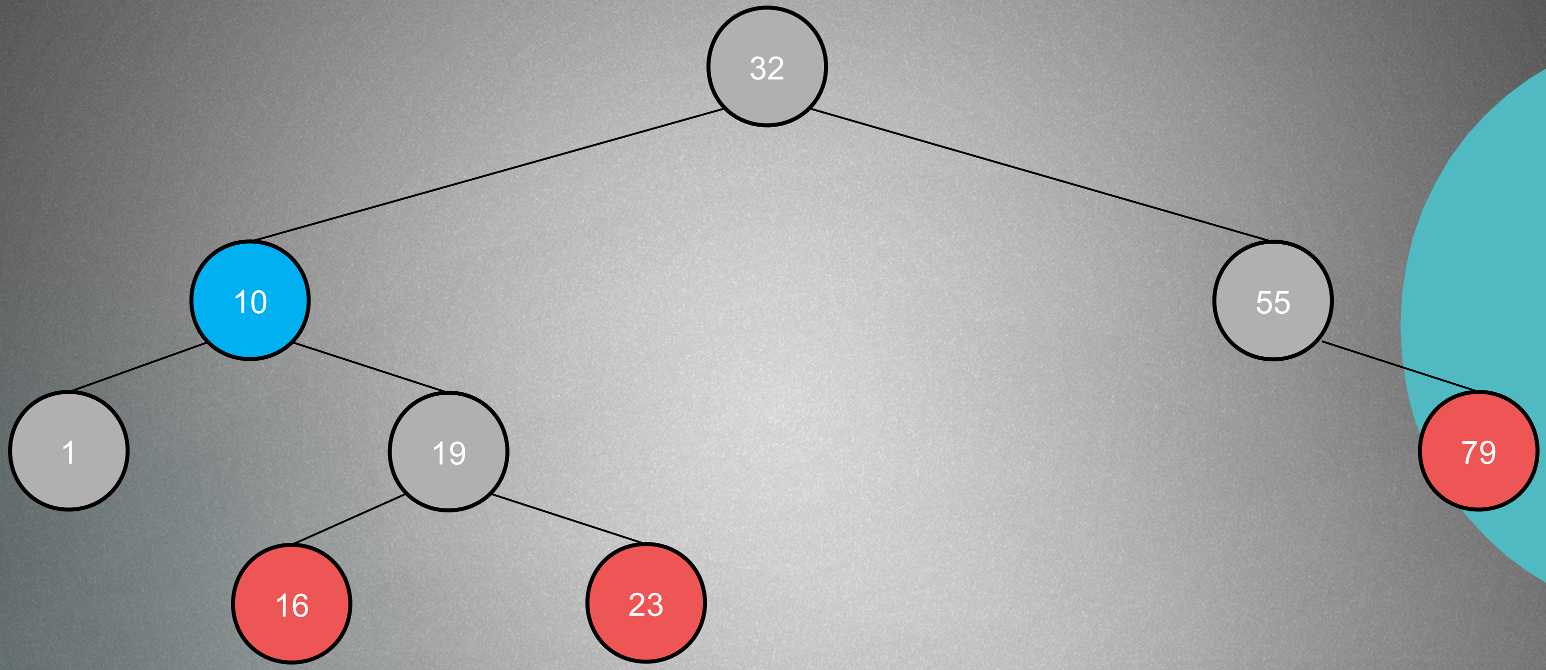
Example:

We want to insert 12



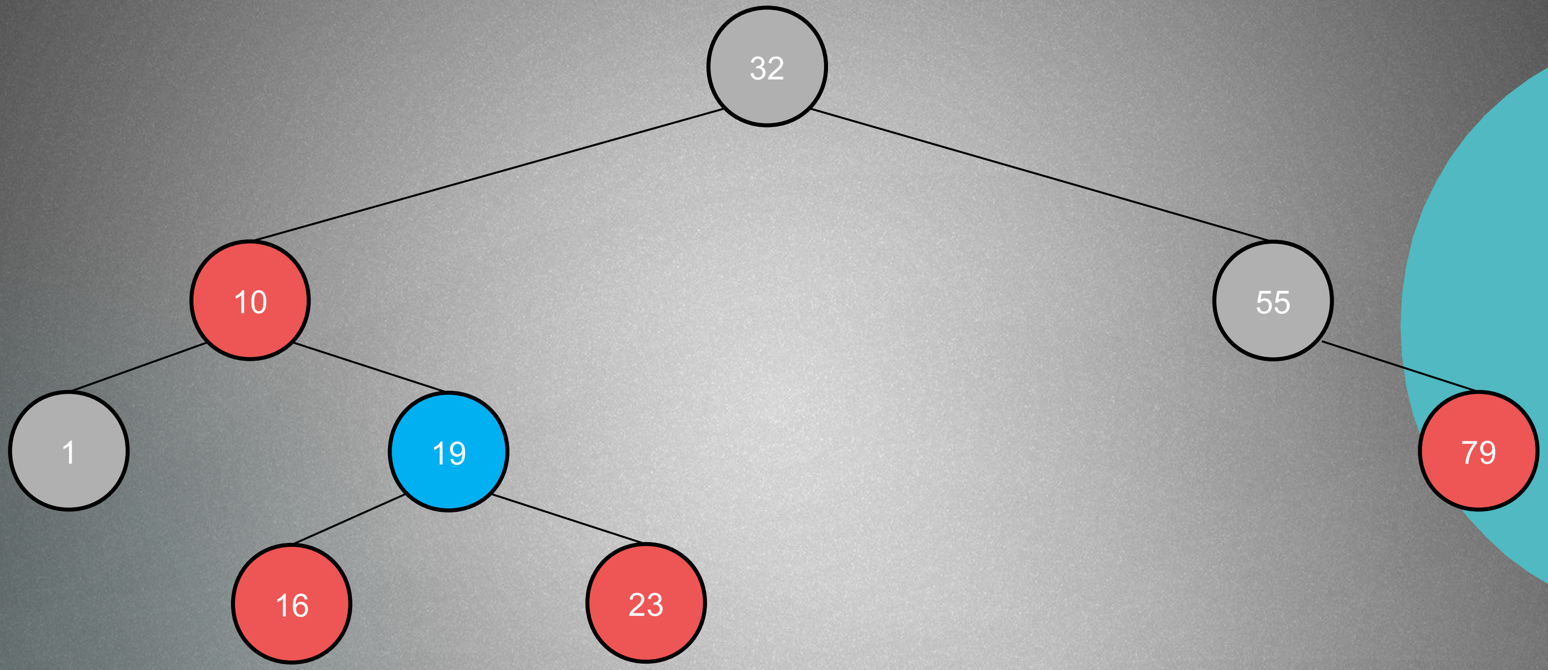
Example:

We want to insert 12



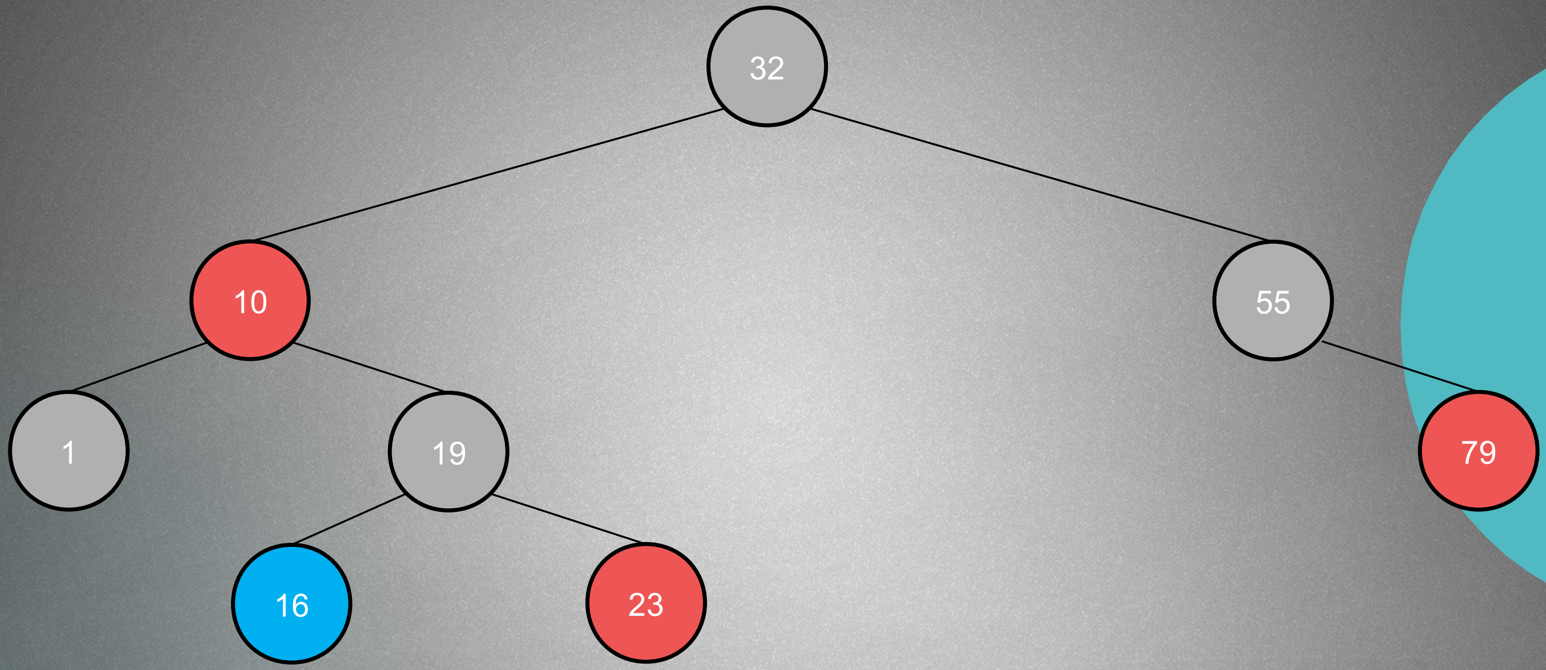
Example:

We want to insert 12

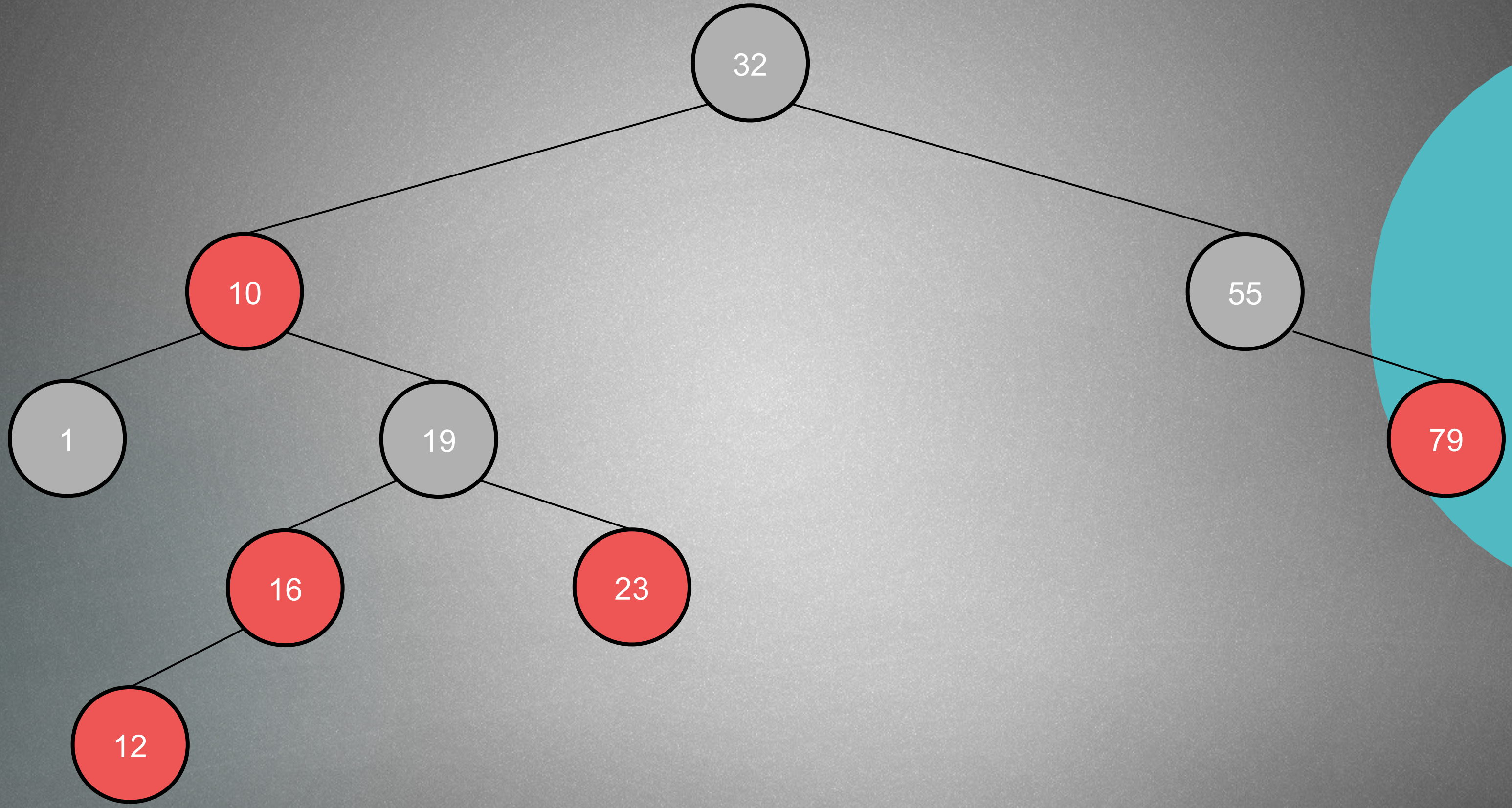


Example:

We want to insert 12

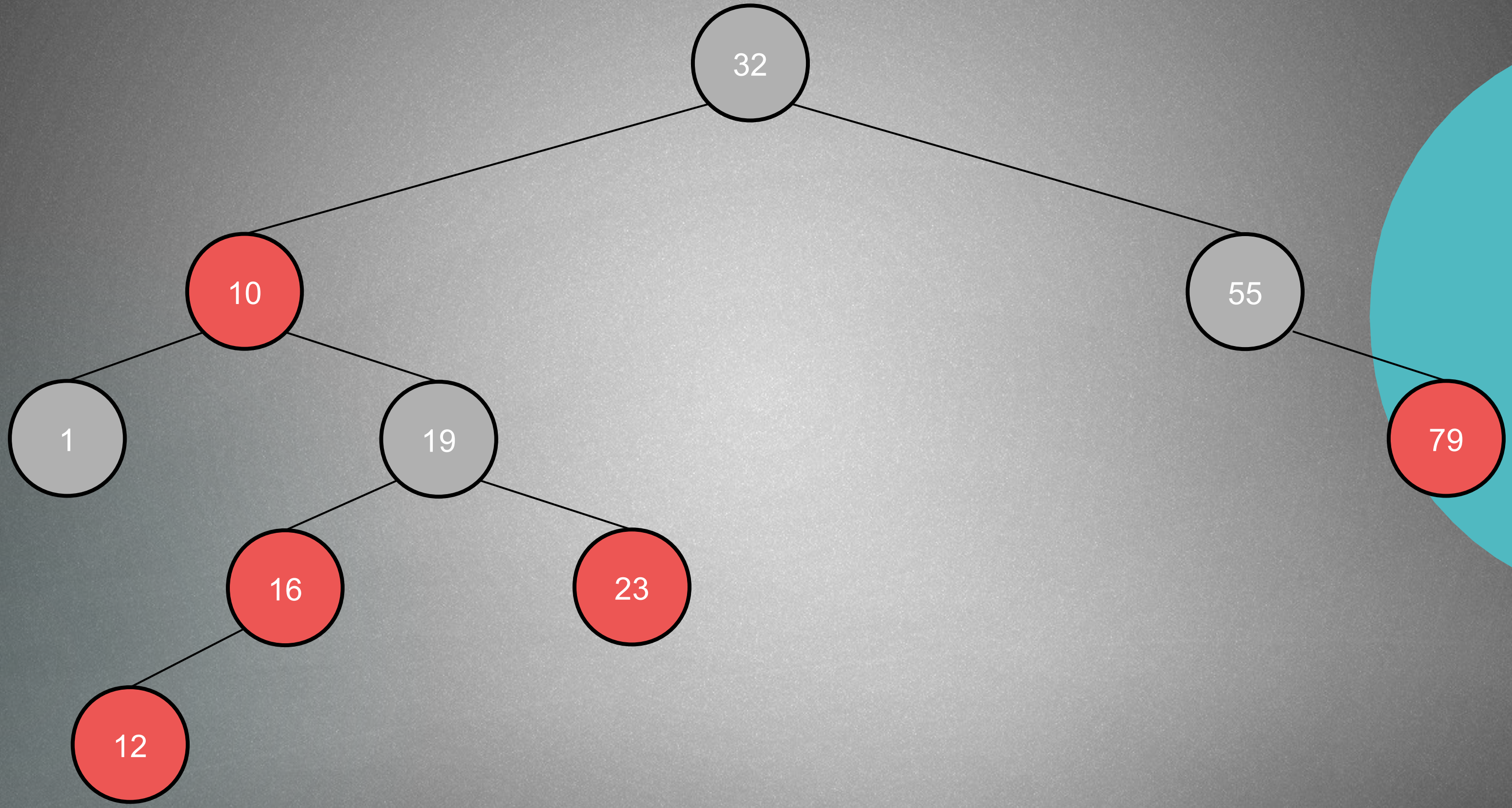


Example:



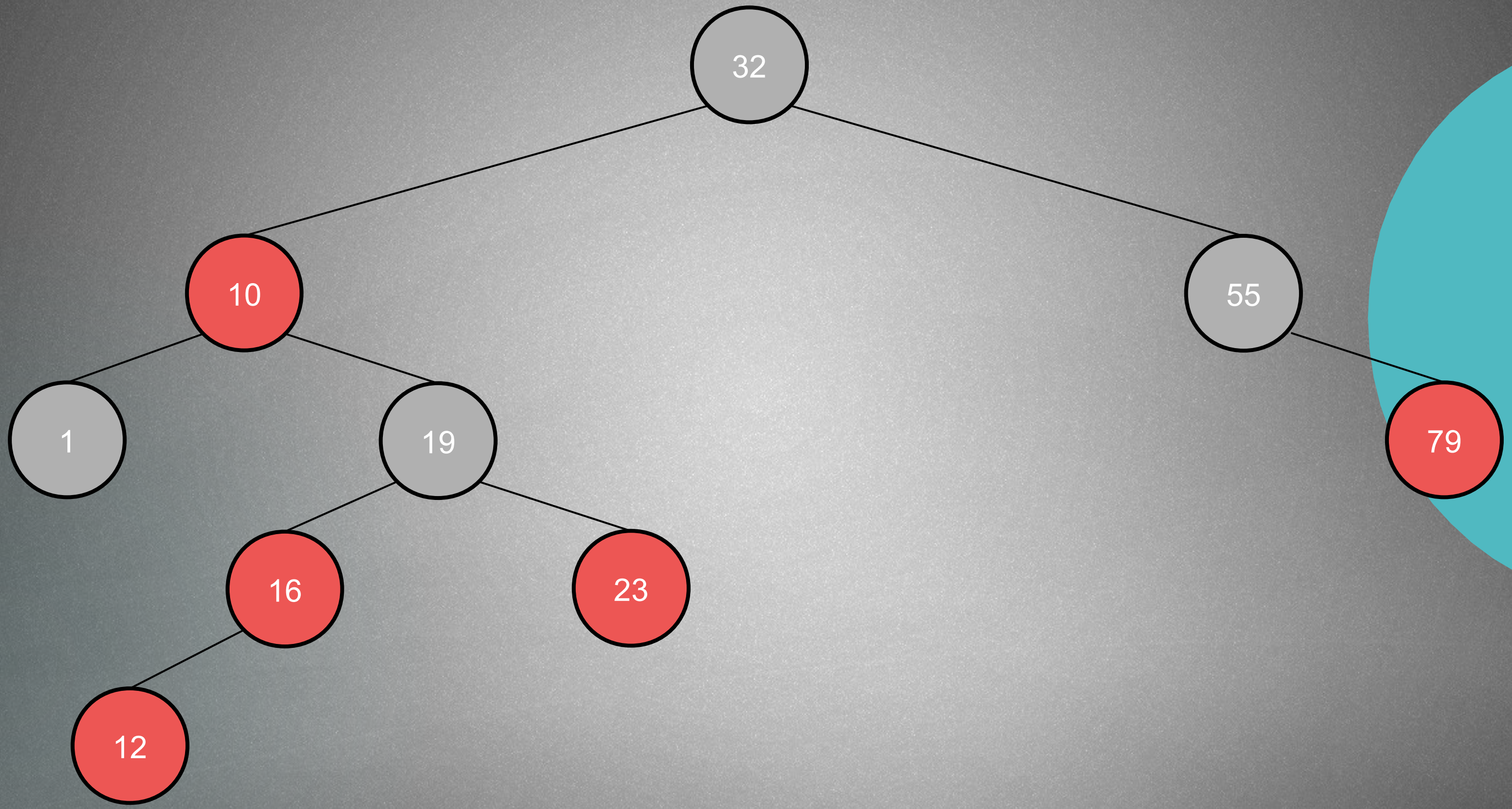
Example:

On every insertion we have to check whether the red black tree properties are violated or not !!!



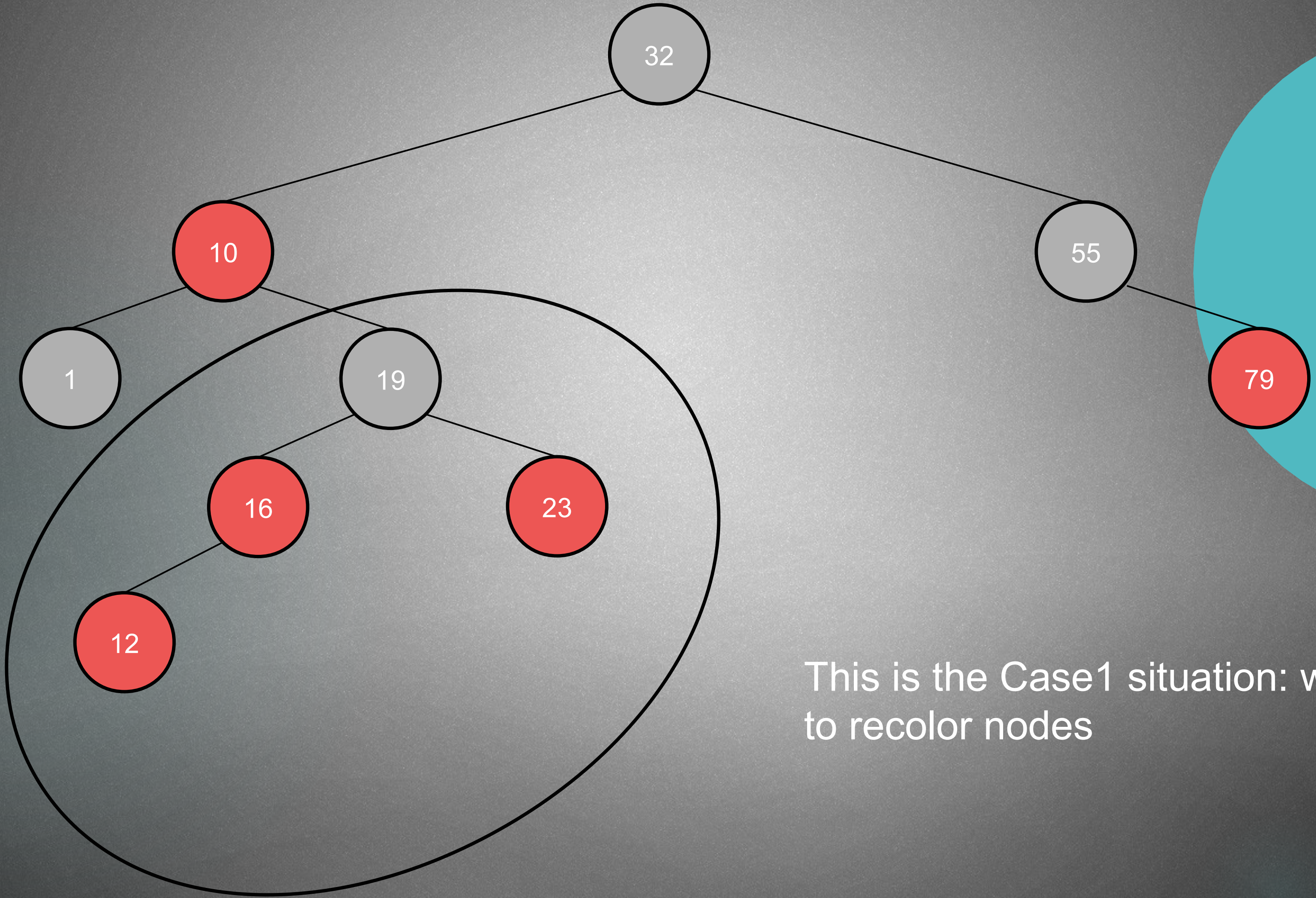
Example:

On every insertion we have to check whether the red black tree properties are violated or not !!!



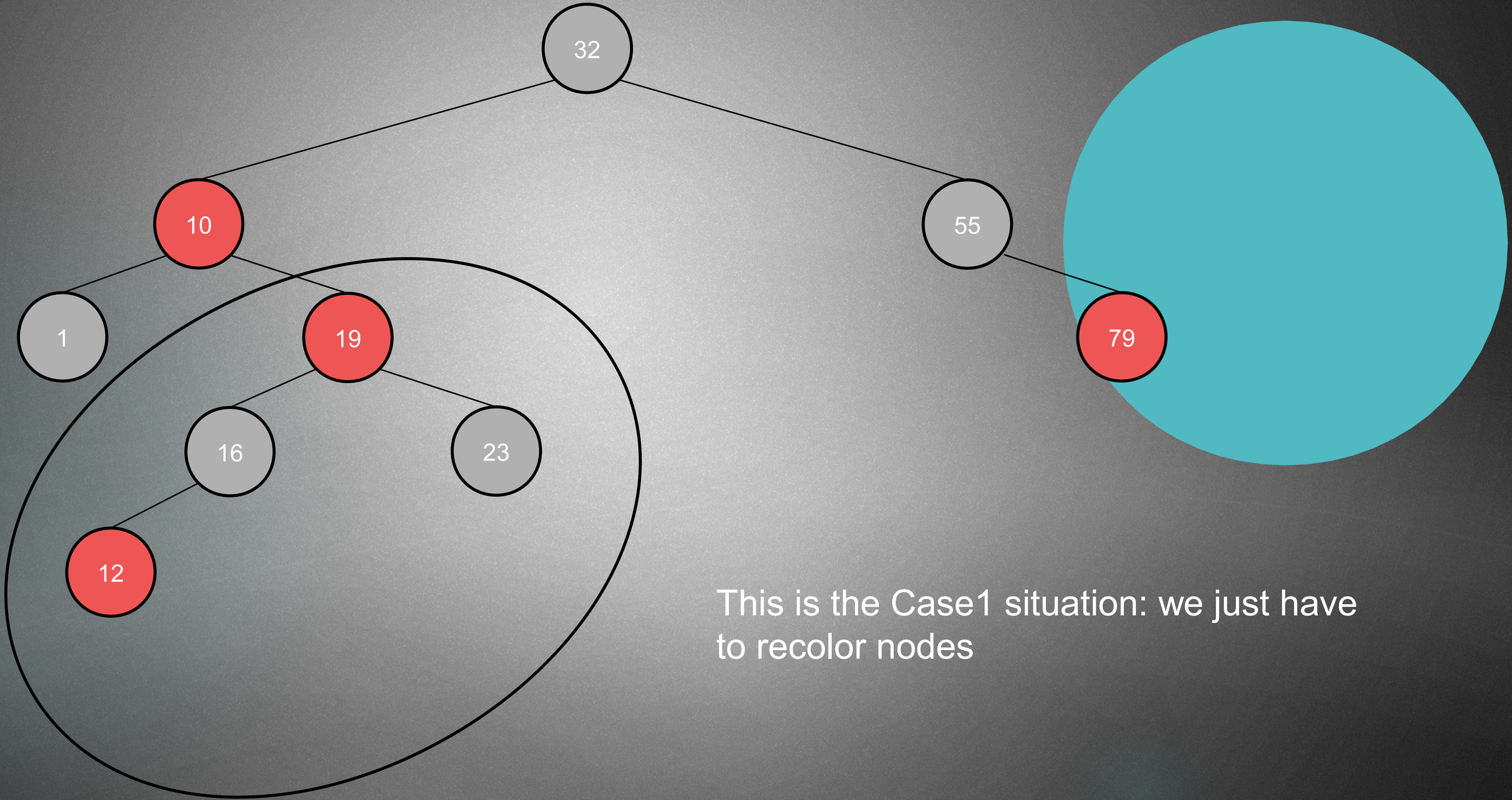
It is violated because red node has a single red child

Example:



This is the Case1 situation: we just have to recolor nodes

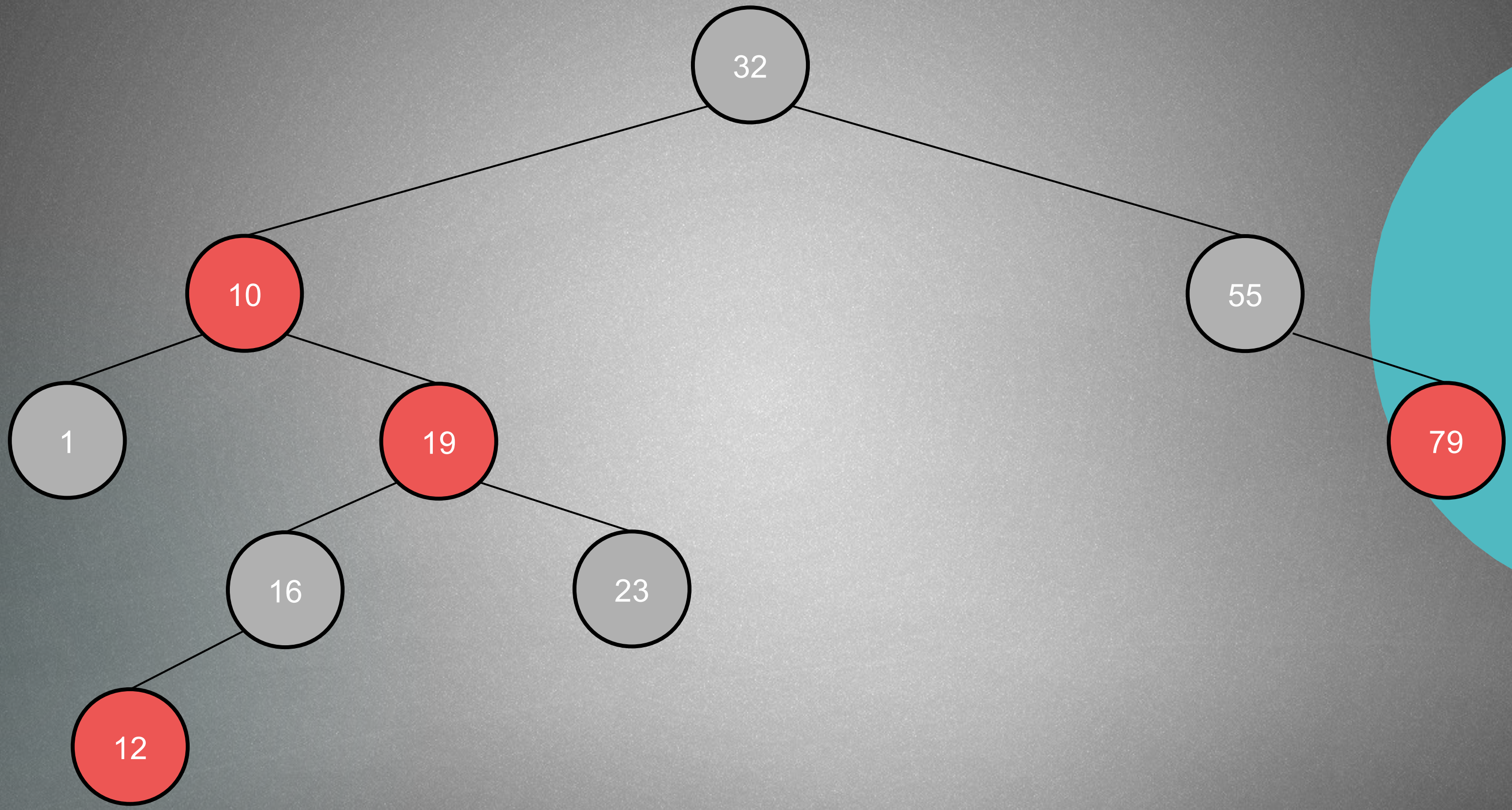
Example:



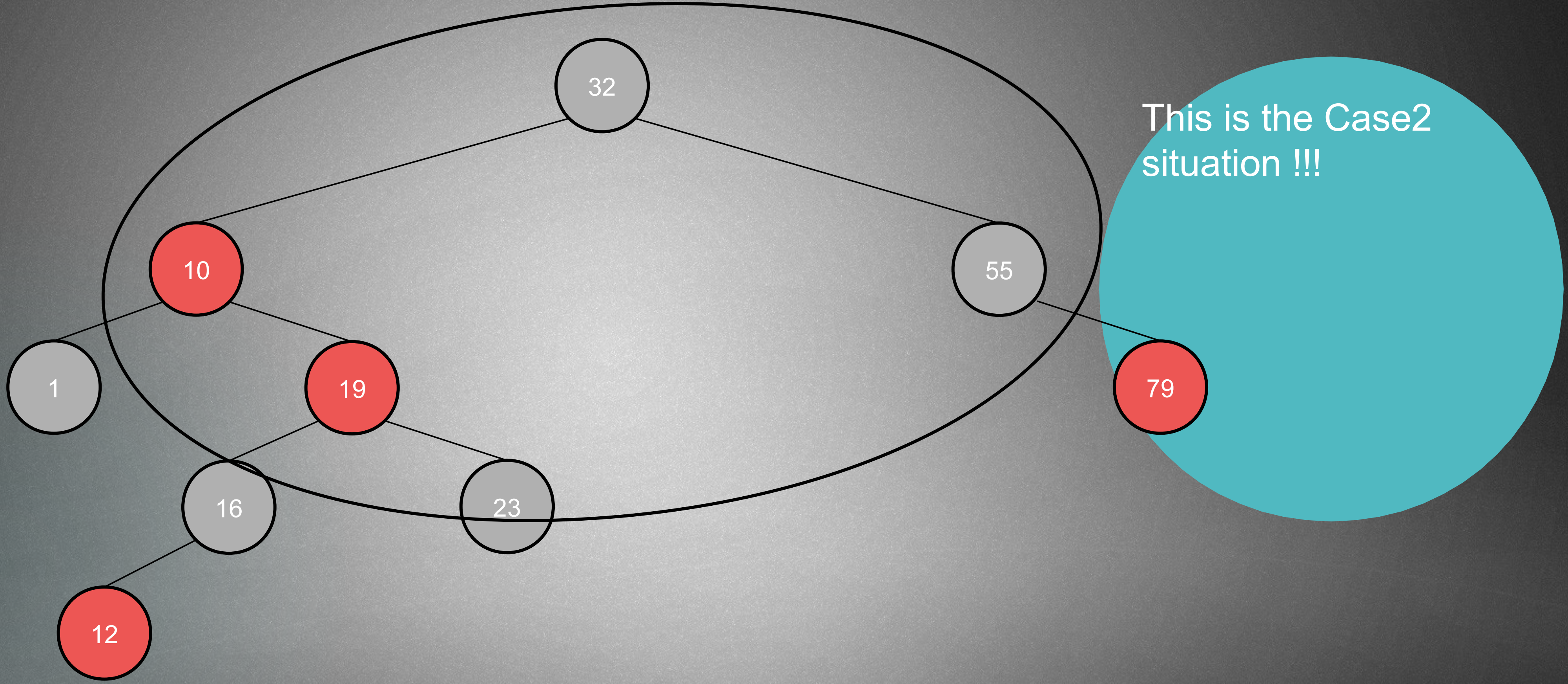
This is the Case1 situation: we just have to recolor nodes

Example:

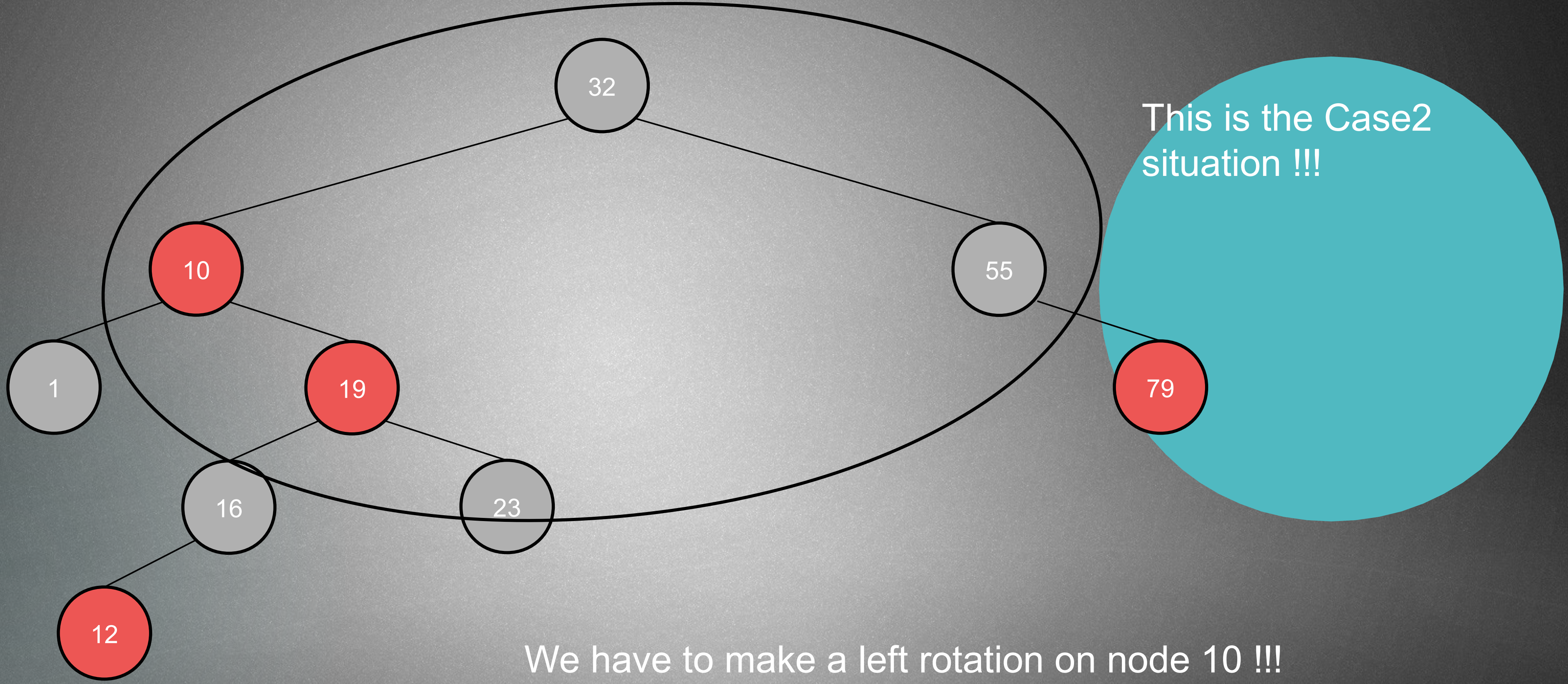
We have to check whether
the red black tree properties are violated or not !!!



Example:

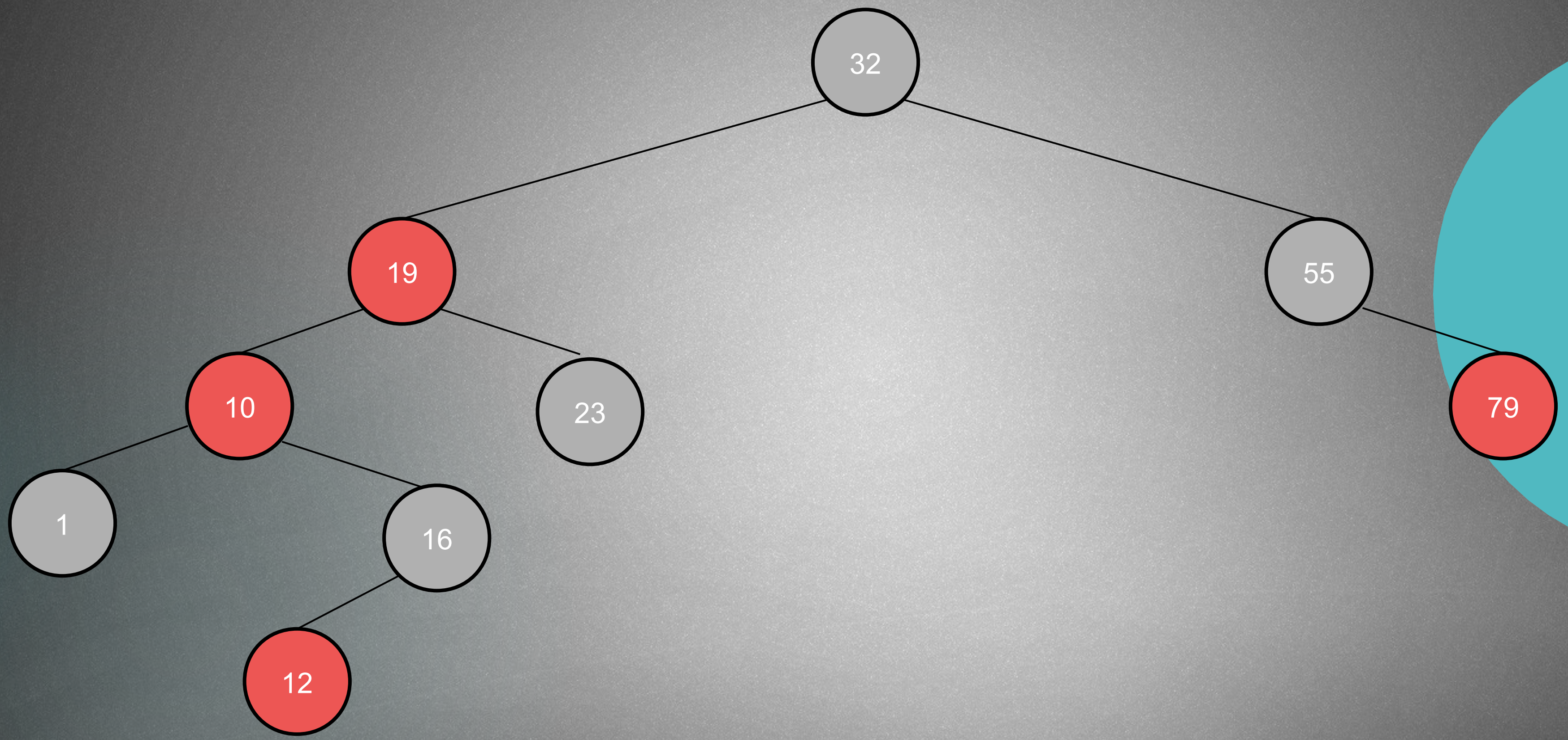


Example:



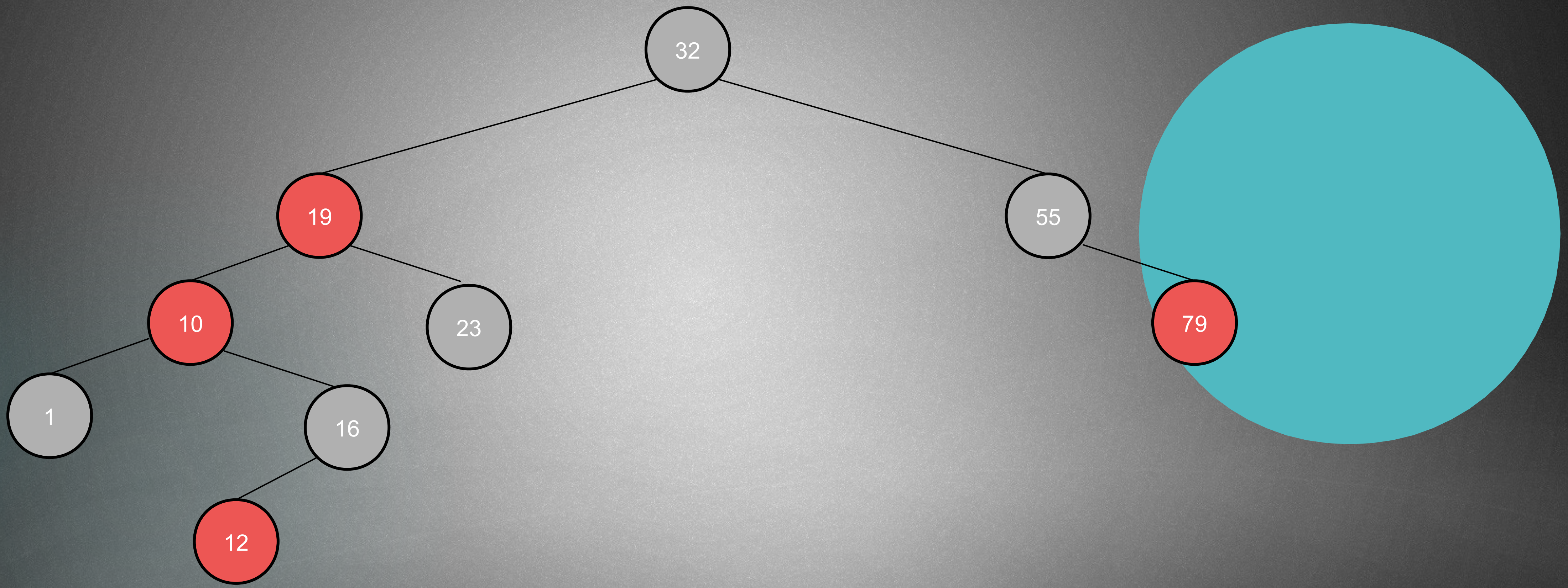
We have to make a left rotation on node 10 !!!

Example:



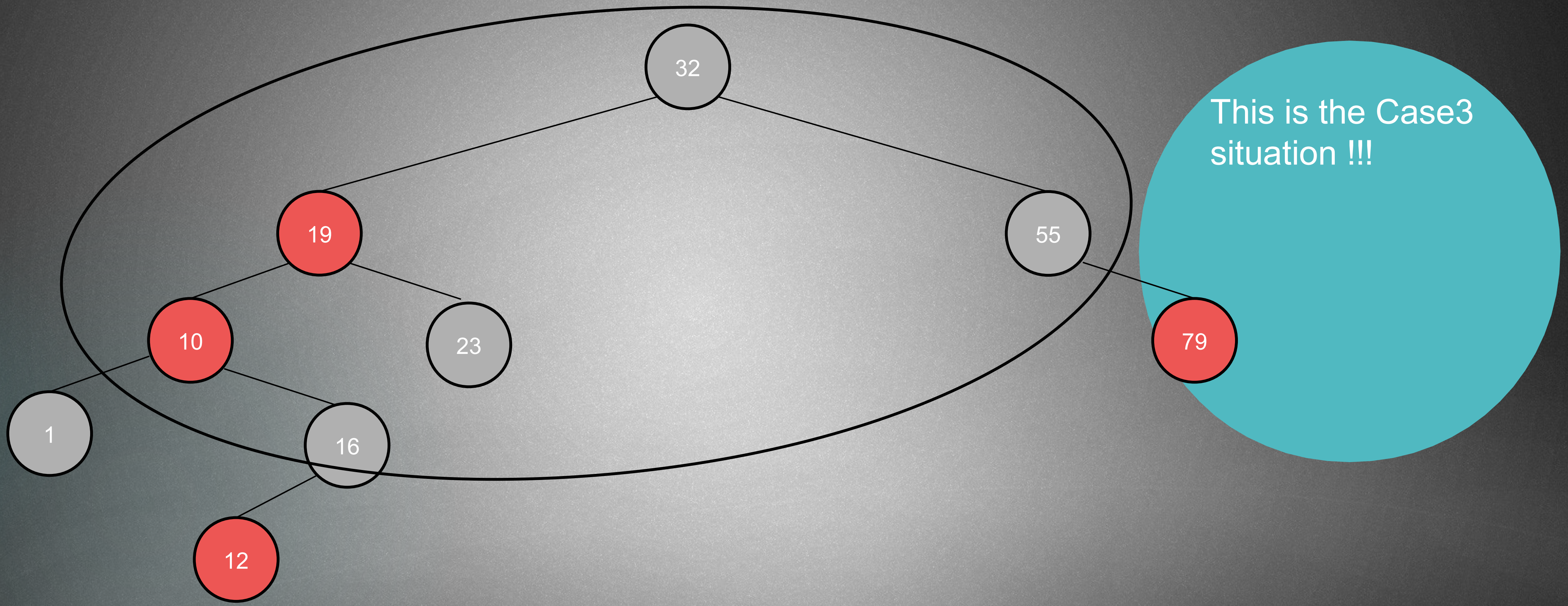
Example:

We have to check whether
the red black tree properties are violated or not !!!



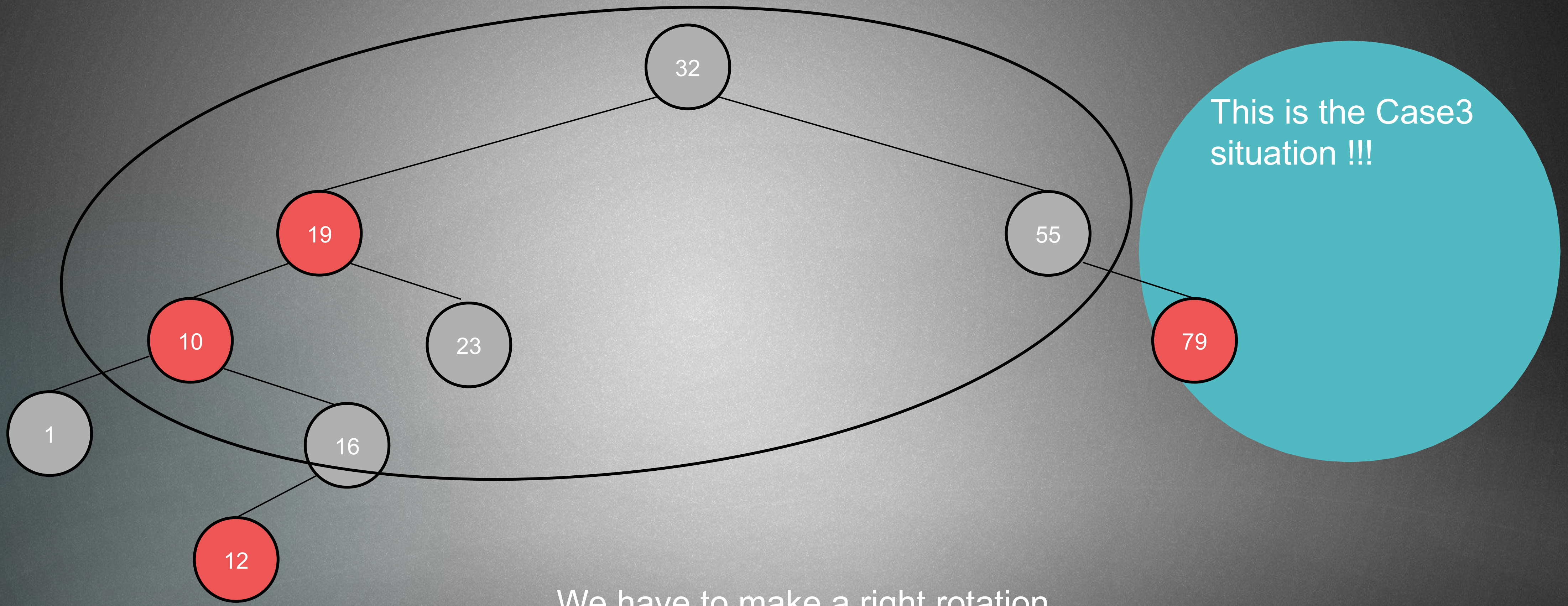
Example:

We have to check whether
the red black tree properties are violated or not !!!



Example:

We have to check whether
the red black tree properties are violated or not !!!



We have to make a right rotation
on the root node !!!

Example: