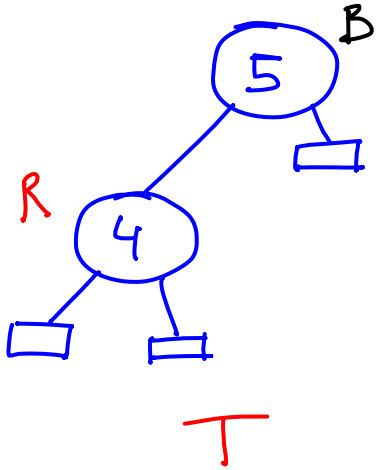
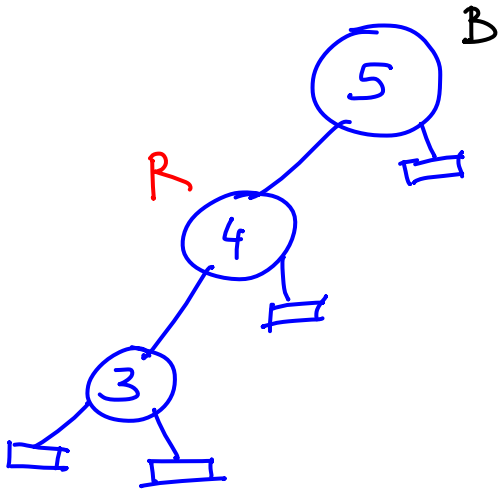
Insertion in Red-Black Trees



T

Insert(T, 3)



What color to give to 3?

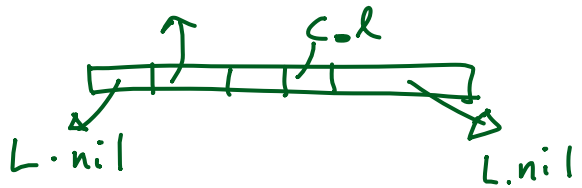
We need to change color of some other nodes and pointer structure (rotations) to restore R-B properties.

(i) Red color
Red node has a red child ✗

(ii) Black color
5-4-3-leaf 3 black nodes
5-leaf 2 black nodes
✗

Pseudo-Code

```
Insert(T, z)
 if (T.root == T.nil)
      T.root = z
      z.col = black
      z.p   = T.nil
 else Insert1(T.root, z)
```



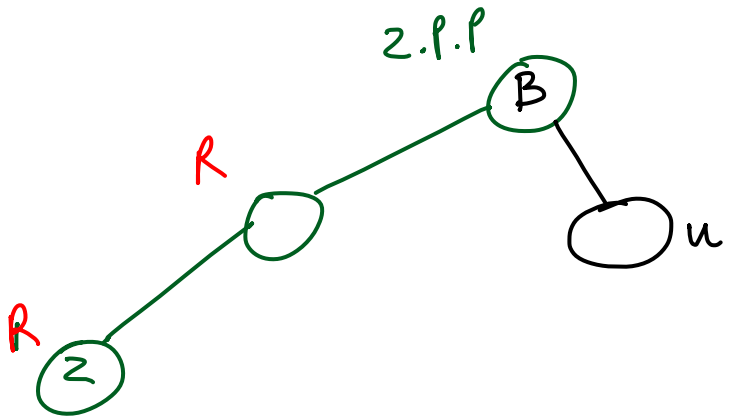

```
Insert1(x, z)
 if z.key ≤ x.key and x.left == T.nil
      x.left  = z
      z.p     = x
      z.col   = red
      if (x.col == red)
           Ifixup(T, z)
      return

 if z.key ≤ x.key and x.left ≠ T.nil
      Insert1(x.left, z)
      return

// Fill in the other two symmetric cases
   z.key > x.key
```

Ifixup (T, z) // gets called only when z.col and z.p.col both are red

This is the only violation of R-B properties in T

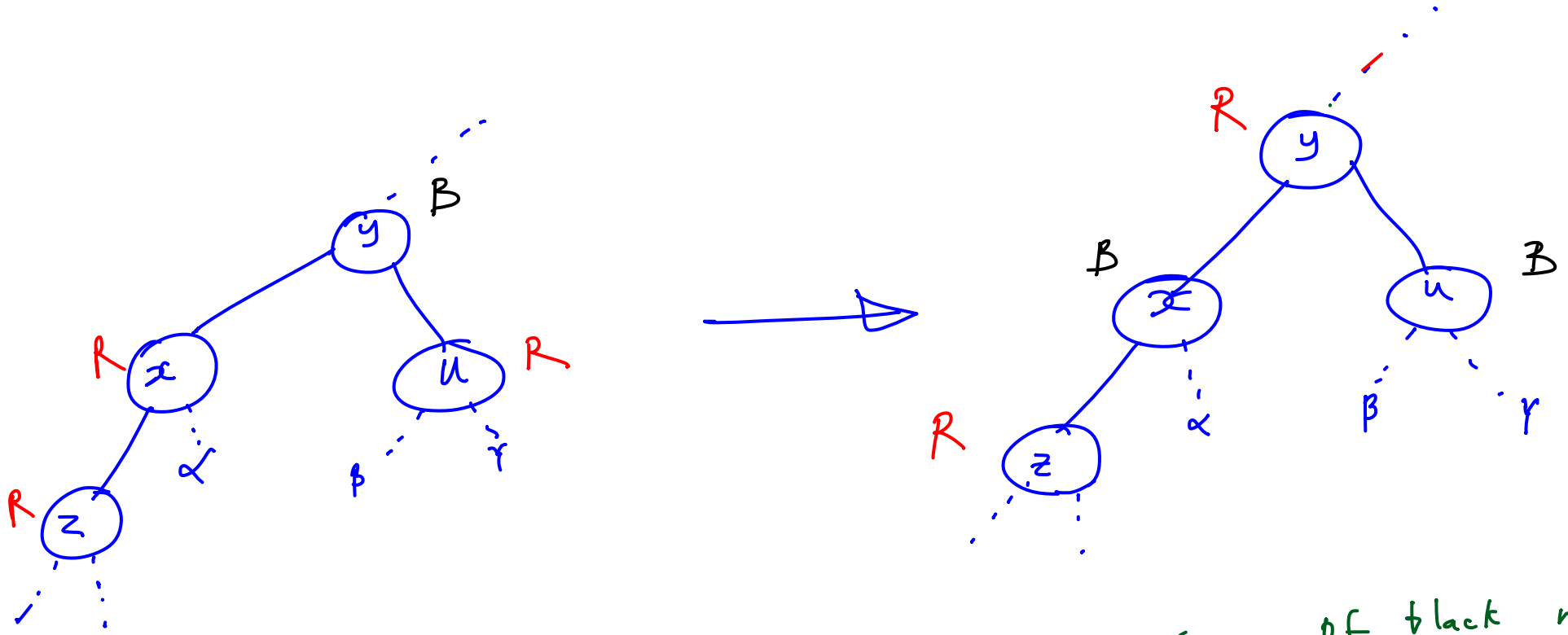


u is uncle of z
(sibling of z's father)

Exercise Write pseudo-code for finding uncle of z in T.

Uncle (T, z).

Case I — z's uncle has red color



Exercise: verify property 5 (no. of black nodes on any path from a given node to any descendent leaf).

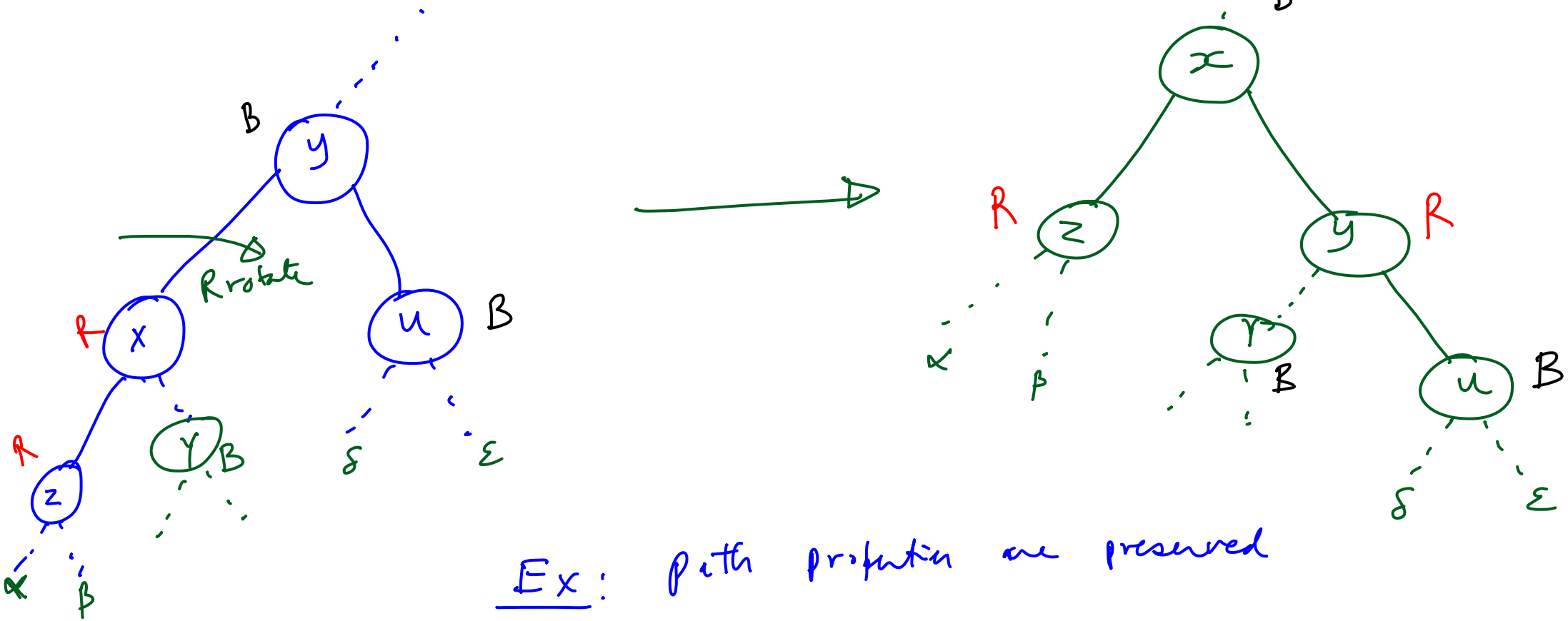
Only possible violations in the RB property are
(i) y may be root
(ii) y.p may be red

In (i) color of the root is changed to black

(Ex: if a tree satisfies all R-B properties, except for color of its root, then coloring the root to black yields a R-B tree)

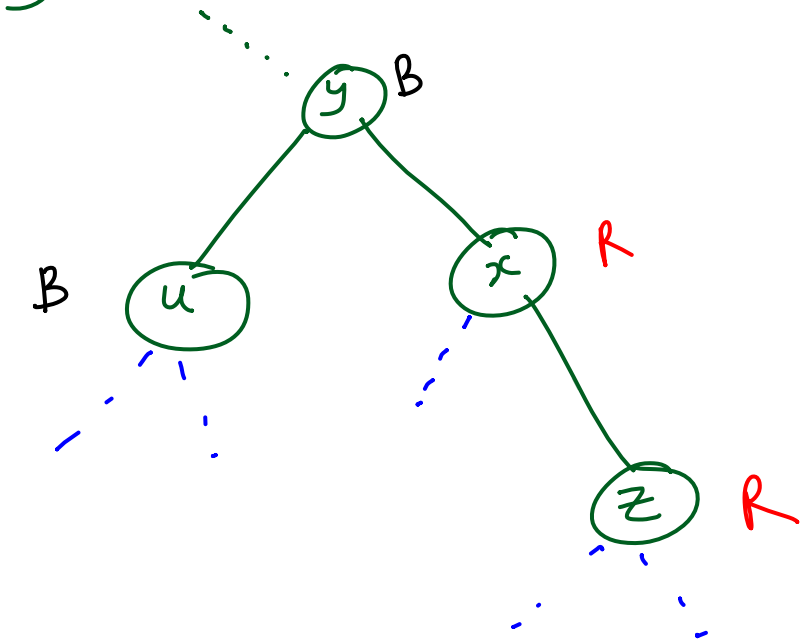
In (ii): call RBfixup(T, y) // y.p in red

Case II Color of z's uncle is black.



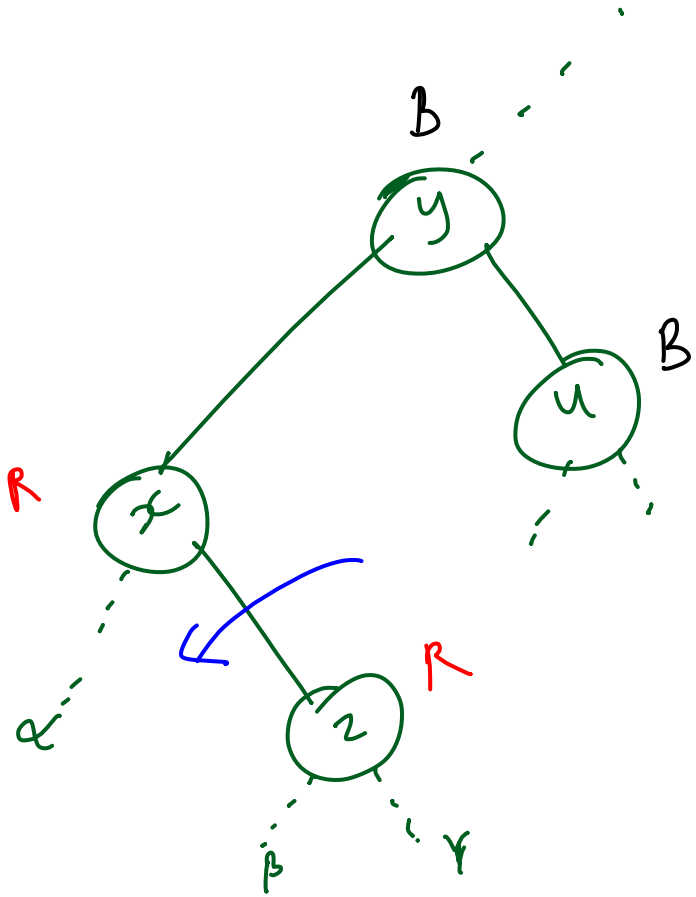
Ex: path properties are preserved

Symmetric Case

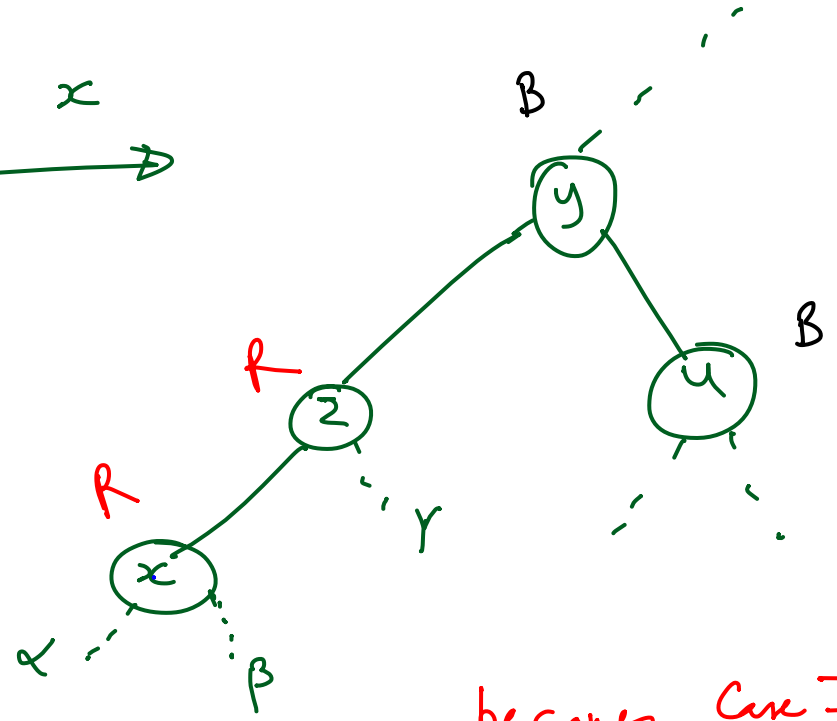


Ex: verify this case.

Case II (b) [x is left child of y and z is ~ right child of x
OR
x is right child of y and z is left child of x]



Left rotate on x



becomes Case II (a)
[previous case]

```
IfixUp(T, z)        // z.col = red, z.p.col = red
                    // This is the only violation in R-B properties of T.
x = z.p
y = x.p
u = uncle(z)

① if(u.col == red)
        x.col = black
        u.col = black
        y.col = red
        if(y == T.root)
            y.col = black
        elif(y.p.col == red)
            IfixUp(T, y)
        else return
    return

② if (x == y.left and z == x.left)
        Right-rotate(T, y)
        x.col = black
        y.col = red
        return

③ if (x == y.right and z == x.right)
        Left-rotate(T, y)
        x.col = black
        y.col = red
        return
```

④ if (x == y.left) and (z == x.right)
    Left-rotate(T, x)
    Ifixup(T, x)
    return

⑤ if (x == y.right) and (z == x.left)
    Right-rotate(T, x)
    Ifixup(T, x)
    return

Termination

In cases 2–3, Ifixup terminates (after constant amount of instructions)

In cases 4–5, Ifixup terminates in the next recursive call.
(next call to Ifixup lands in case 2–3)

In case 1, either Ifixup terminates or calls itself from a node of higher height. Since the height is bounded by height of the tree, This also terminates.
(worst case O(h) iterations)

Time Complexity of Ifixup O(h)
" of Insert(T, z)
O(h) + time complexity of Ifixup

Overall Insert works in $O(h)$ time

$O(\log n)$ time [R-B properties]