COMP3600/6466 Algorithms

Lecture 20

S2 2016

Dr. Hassan Hijazi

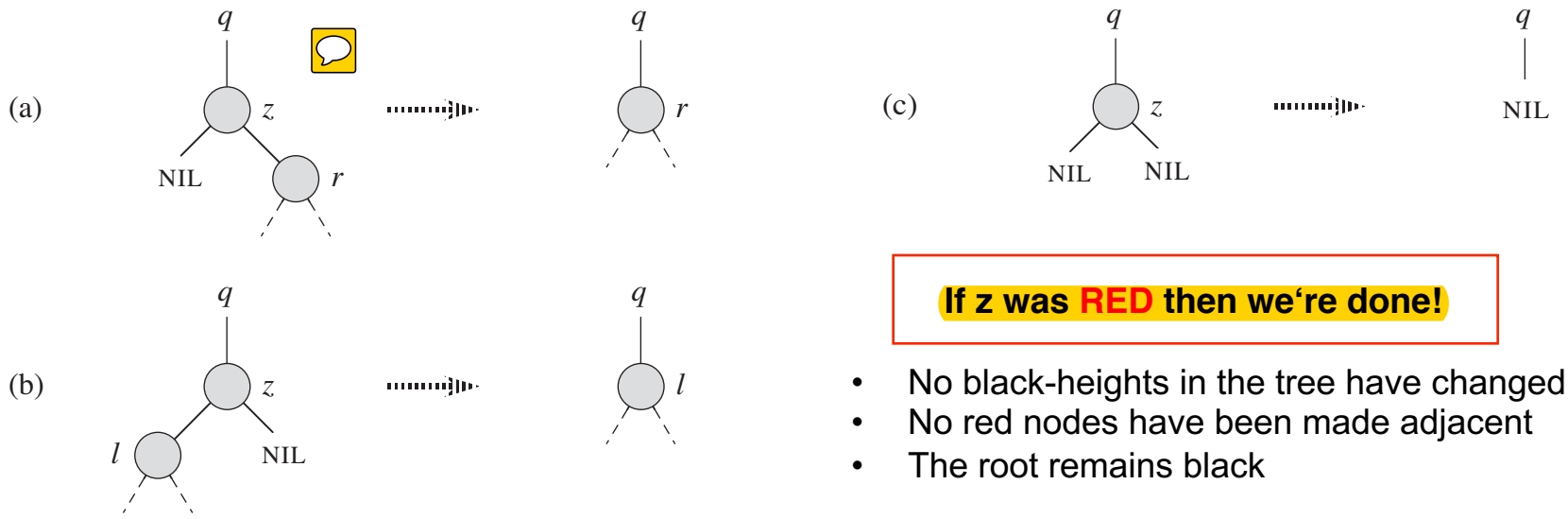
Prof. Weifa Liang

Red Black Trees

AND YOU THOUGHT INSERT WAS TRICKY?

Let's look at DELETE..

CASE A: z has at most one child

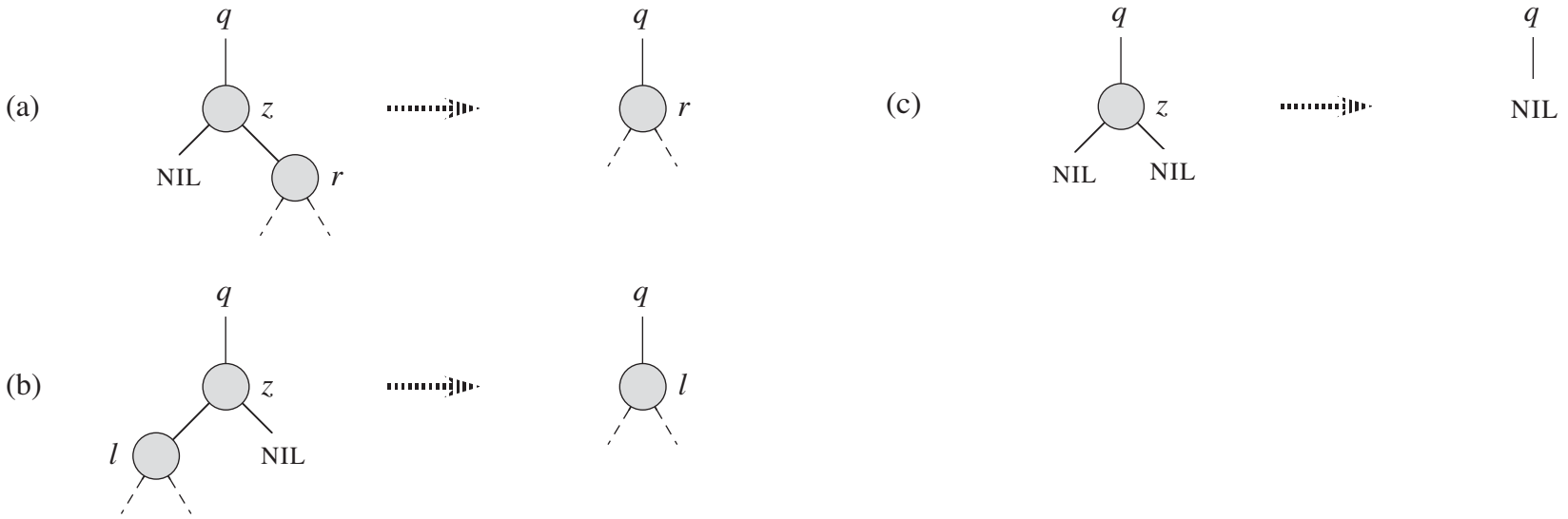


If z was RED then we're done!

- No black-heights in the tree have changed.
- No red nodes have been made adjacent
- The root remains black

Red Black Trees

z is BLACK



LET x be the node that replaces z

Red Black Trees

z is BLACK

~~(a) **z** was the root and **x** (the new root) was a red child: Violates P2.~~

~~(b) both **x** and **x.*p*** are red: Violates P4.~~

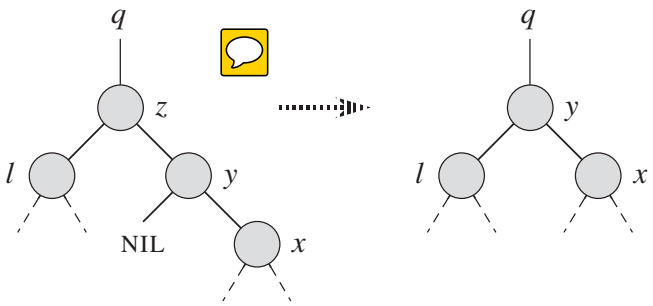
(c) A path that previously contained **z** now has one fewer black node

If **x** was **RED,** by changing its colour to **BLACK,**
all violations are fixed!

Problematic case: x is **BLACK**, then only **(c)** is triggered

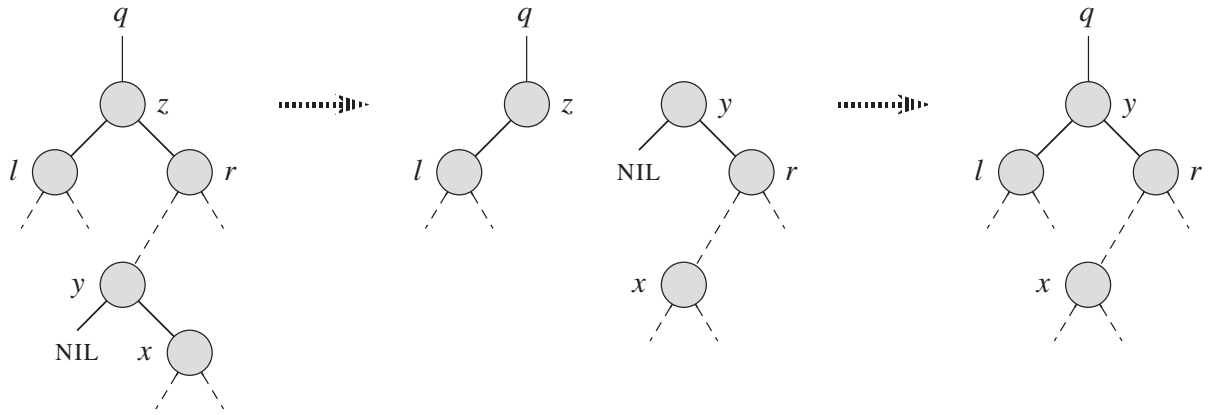
Red Black Trees

CASES B and C: z has 2 children and is replaced by his successor (y)



If y was RED then we're done!

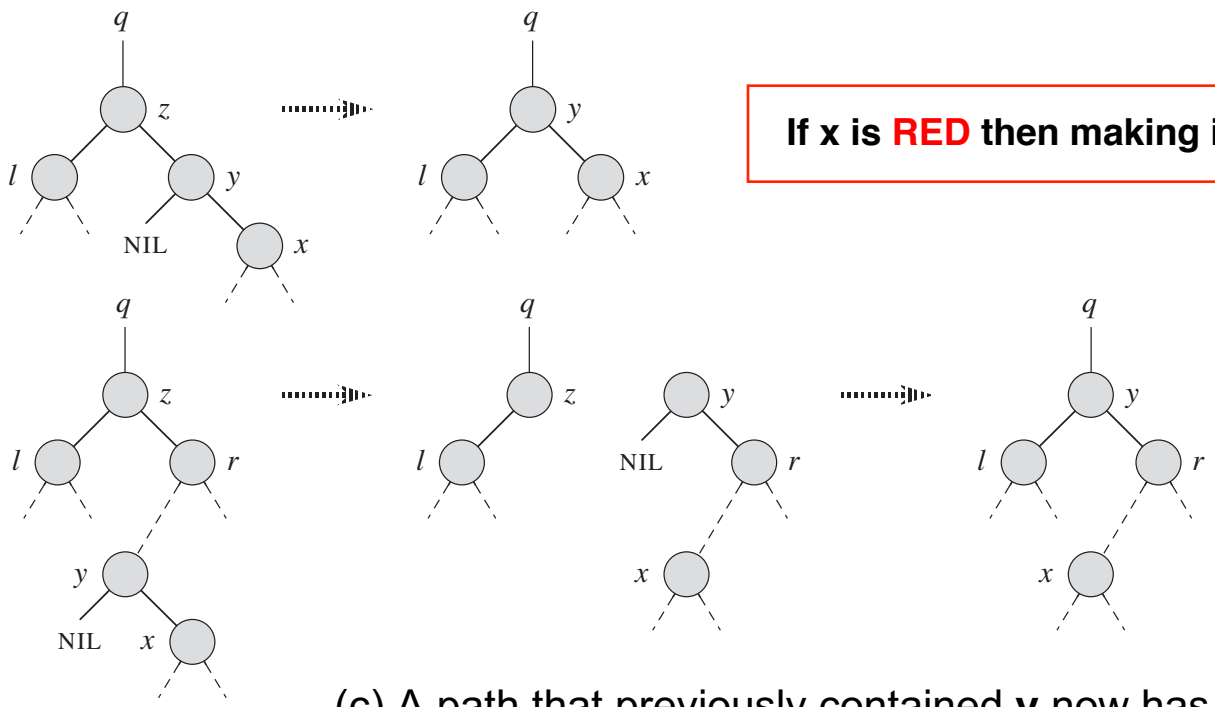
- No black-heights in the tree have changed.
- No red nodes have been made adjacent
- The root remains black



y takes the colour of **z**

Red Black Trees

Problematic case: y was BLACK, then only (c) is triggered



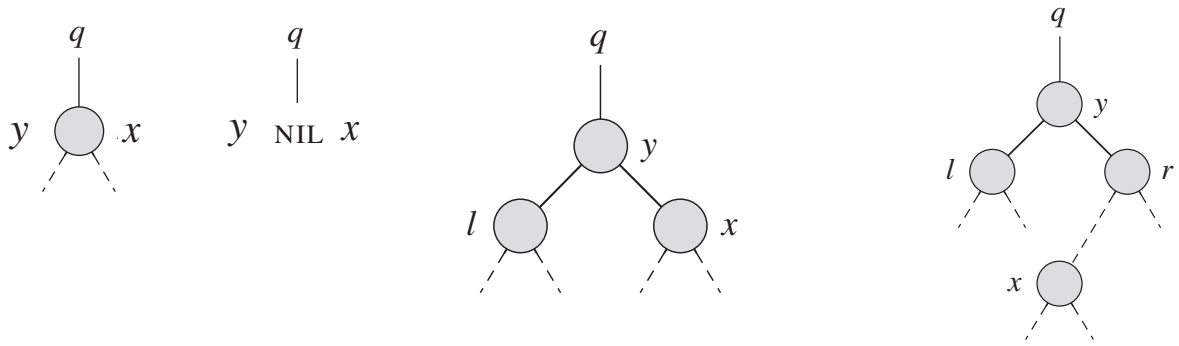
If x is RED then making it BLACK fixes the problem!

(c) A path that previously contained **y** now has one fewer black node

Note that **x** took the place of **y** in the tree

Red Black Trees

In all problematic cases, **x** is **BLACK** and
we have a missing **BLACK** node between **y** and **x**



(c) A path that goes through (q,y,x) now has one fewer black node

Let's assume that the we can add an **IMAGINARY EXTRA BLACKNESS** to a node:
"RED" can become **"RED-BLACK"** and **"BLACK"** can become **"DOUBLE-BLACK"**

With bh(RED) = 0, bh(BLACK) = 1, bh(RED-BLACK) = 1 and bh(DOUBLE-BLACK) = 2

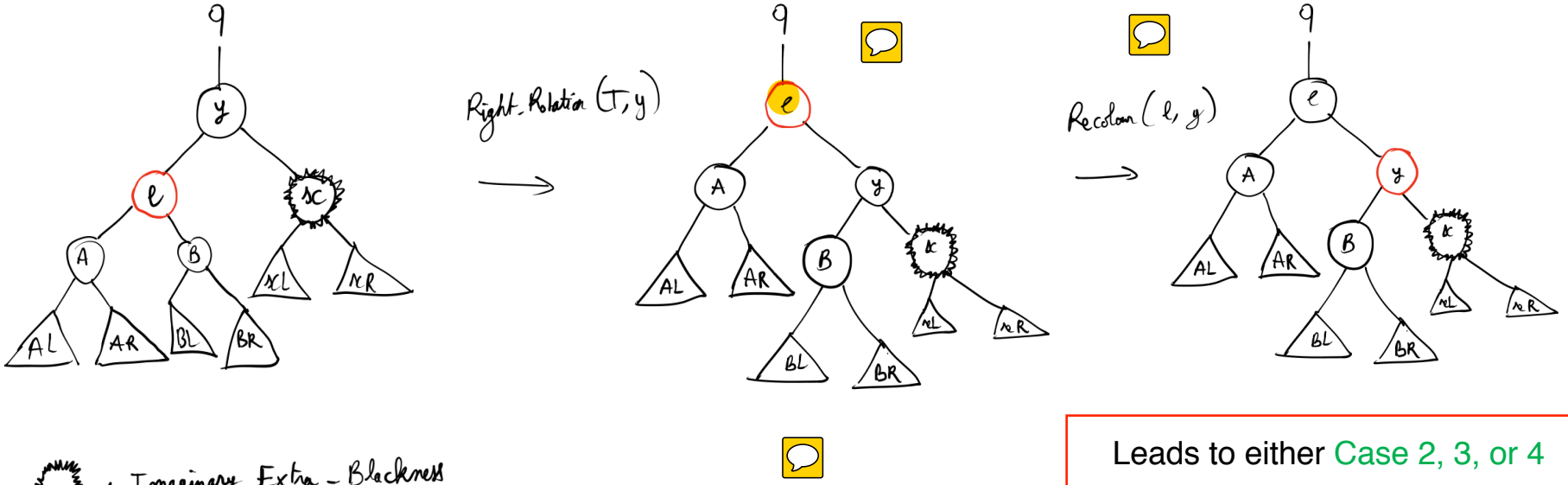
If we make **x DOUBLE-BLACK,** P5 is now satisfied!

Red Black Trees

MOVING THE DOUBLE-BLACKNESS UP THE TREE

CASE 1: The sibling of **x** is RED

GOAL: Have a **BLACK** sibling

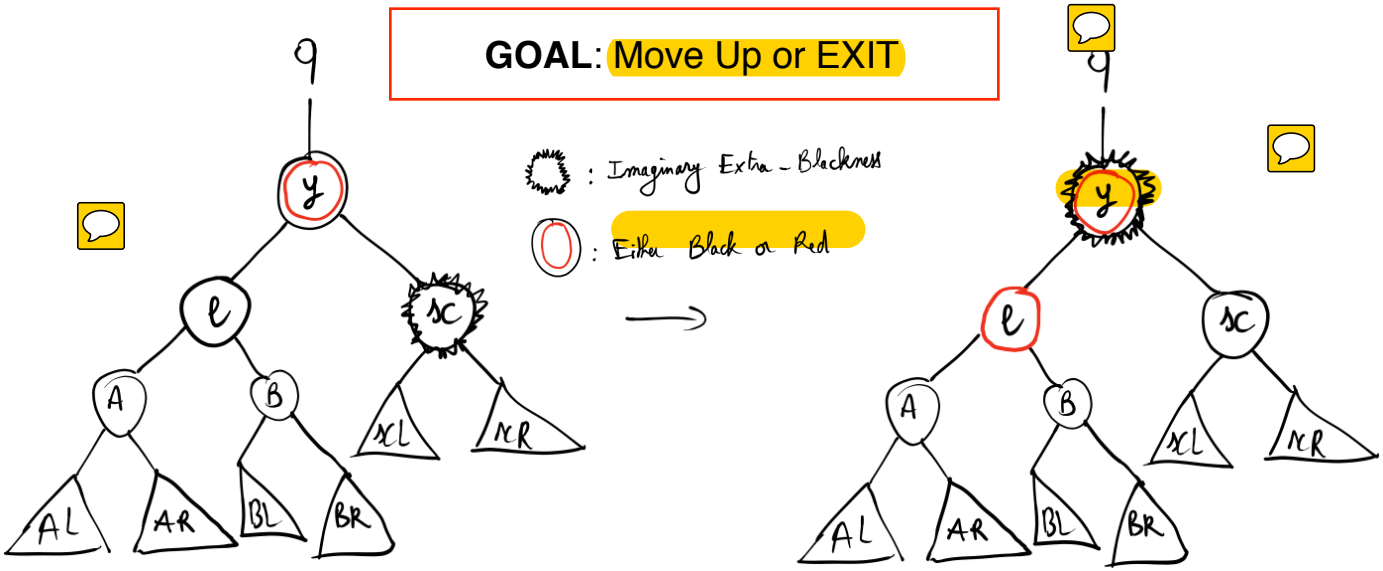


Leads to either Case 2, 3, or 4

Red Black Trees

MOVING THE DOUBLE-BLACKNESS UP THE TREE

CASE 2: The sibling of **x** and its children are **BLACK**



If **y** was RED, we colour it **BLACK** and exit, if not, we are going up the Tree!

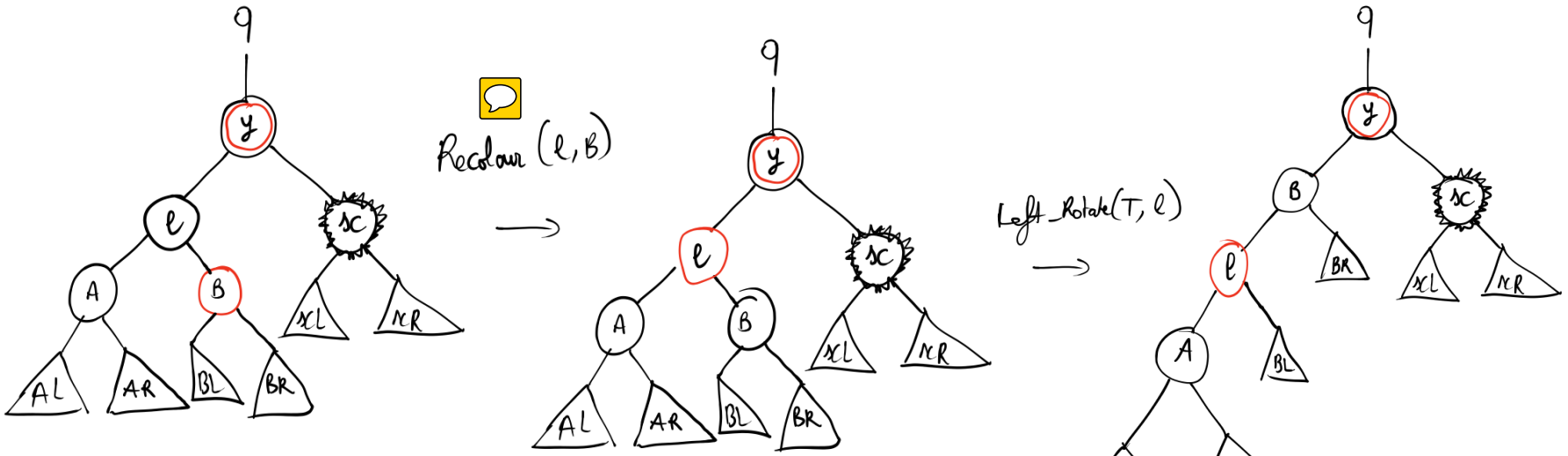
Leads to either EXIT, Case 1, 2, 3, or 4

Red Black Trees

MOVING THE DOUBLE-BLACKNESS UP THE TREE

CASE 3: The sibling of **x is BLACK** and has a **BLACK** left child and a RED right child

GOAL: Have a sibling that has a RED left child



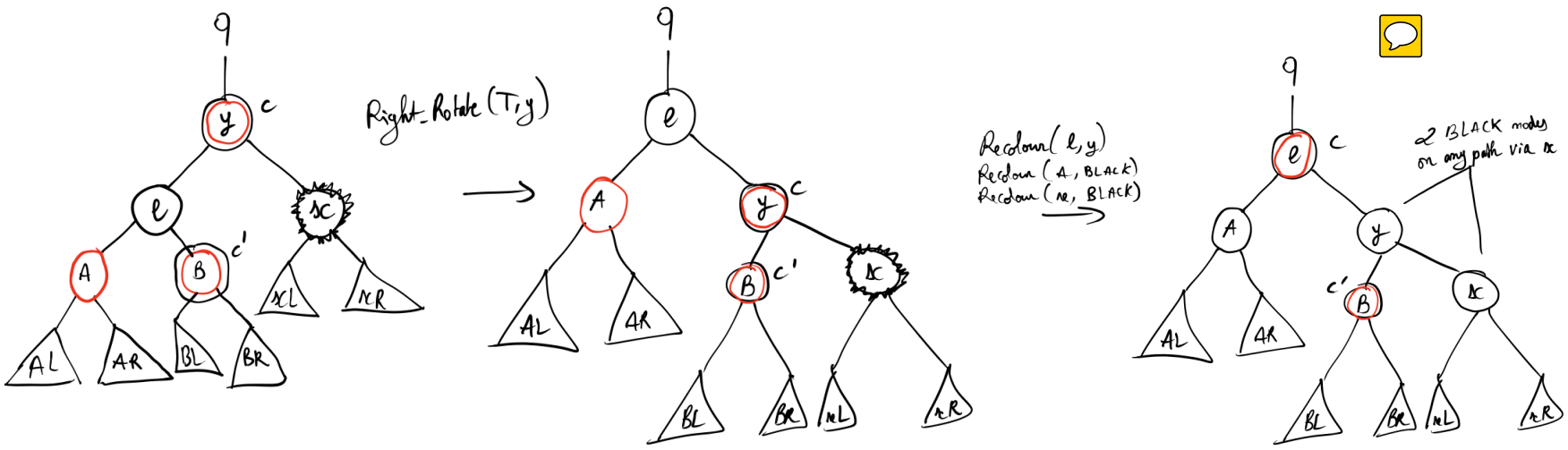
Leads to Case 4

Red Black Trees

MOVING THE DOUBLE-BLACKNESS UP THE TREE

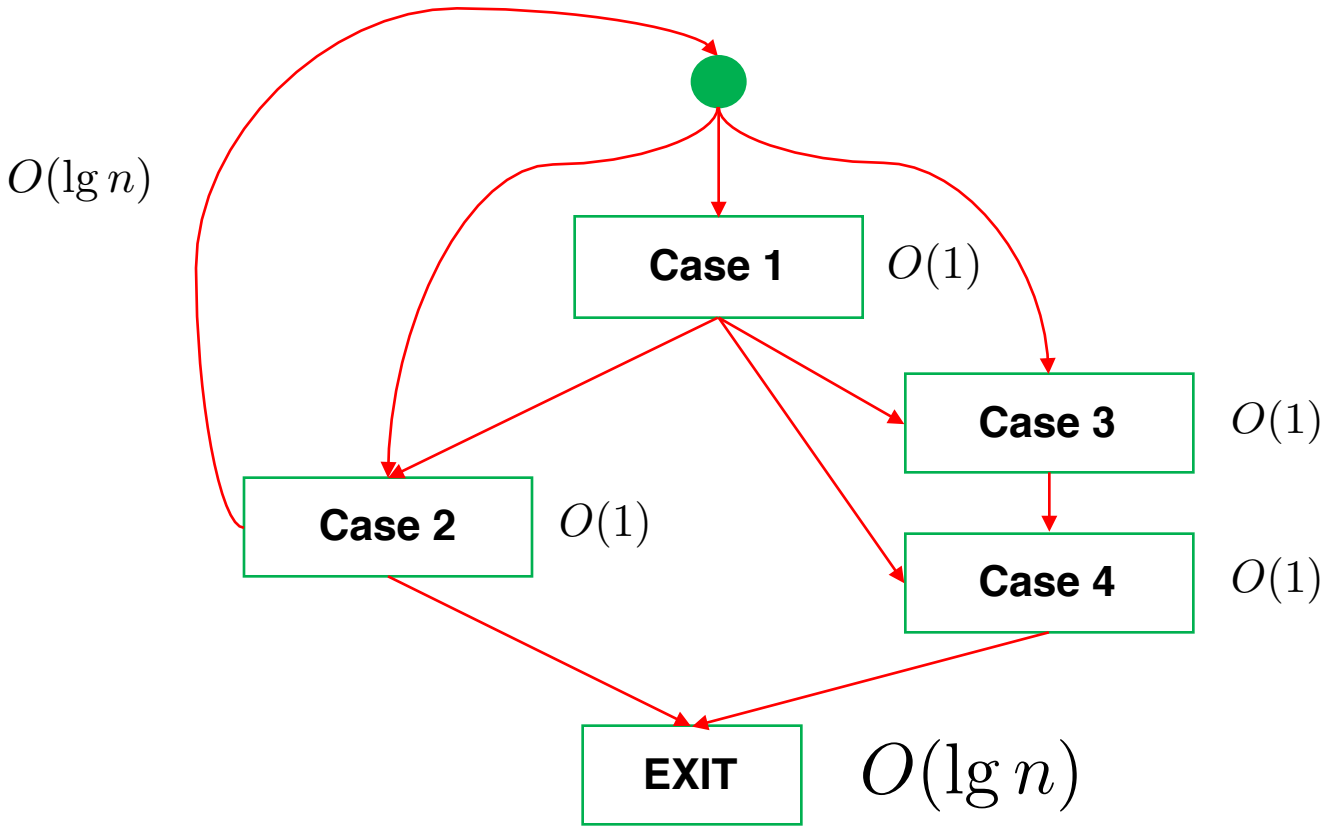
CASE 4: The sibling of **x is BLACK** and has a RED left child

GOAL: EXIT!



Red Black Trees

DELETION FIXUP ACTIONS



Exercise 20.1

Using RB-Insert and RB-Delete build a Red-Black tree that has only black nodes and a height of 1

Exercise 20.2

Implement Right-Rotate(T,y)