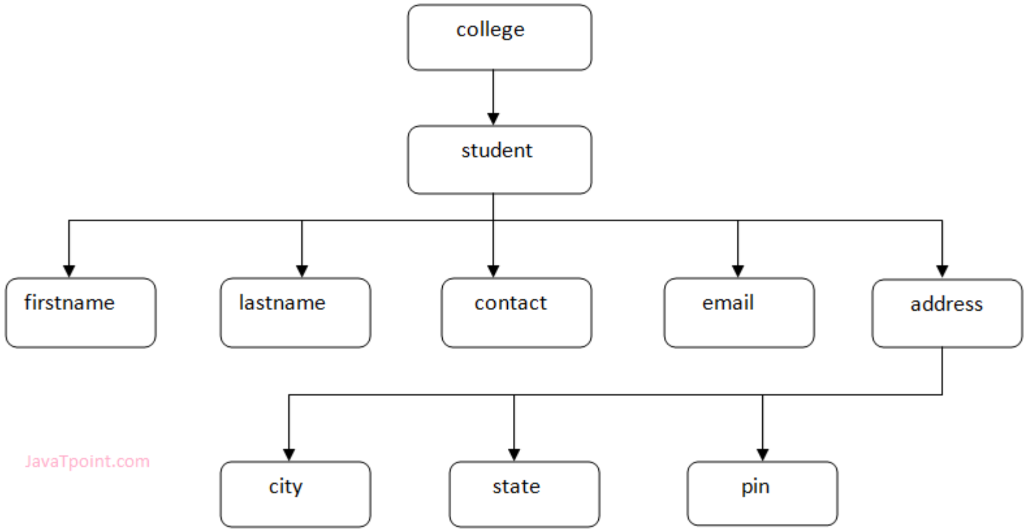
```
1.<?xml version="1.0"?>

2.<college>

3.  <student>

4.      <firstname>Tamanna</firstname>

5.      <lastname>Bhatia</lastname>

6.      <contact>09990449935</contact>

7.      <email>tammanabhatia@abc.com</email>

8.      <address>

9.          <city>Ghaziabad</city>

10.          <state>Uttar Pradesh</state>

11.          <pin>201007</pin>

12.     </address>

13.  </student>

14.</college>
```

XML Tree Structure



RED-BLACK trees

Objectives : search , $\underbrace{\text{insert, delete}}$ in $O(\log n)$ time

always maintain $\Theta(\log n)$ height & $\underbrace{\text{update}}$ in $O(\log n)$ time

RED-BLACK trees

Structure: 1) nodes are colored red or black.

RED-BLACK trees

Structure:

1) nodes are colored red or black.
2) root is always black.

RED-BLACK trees

Structure:

1) nodes are colored red or black.
2) root is always black.
3) add black "dummy" leaves so every "real" node has 2 children.

RED-BLACK trees

Structure:

1) nodes are colored red or black.
2) root is always black.
3) add black "dummy" leaves so every "real" node has 2 children.
4) every red node has a black parent.

RED-BLACK trees

Structure:

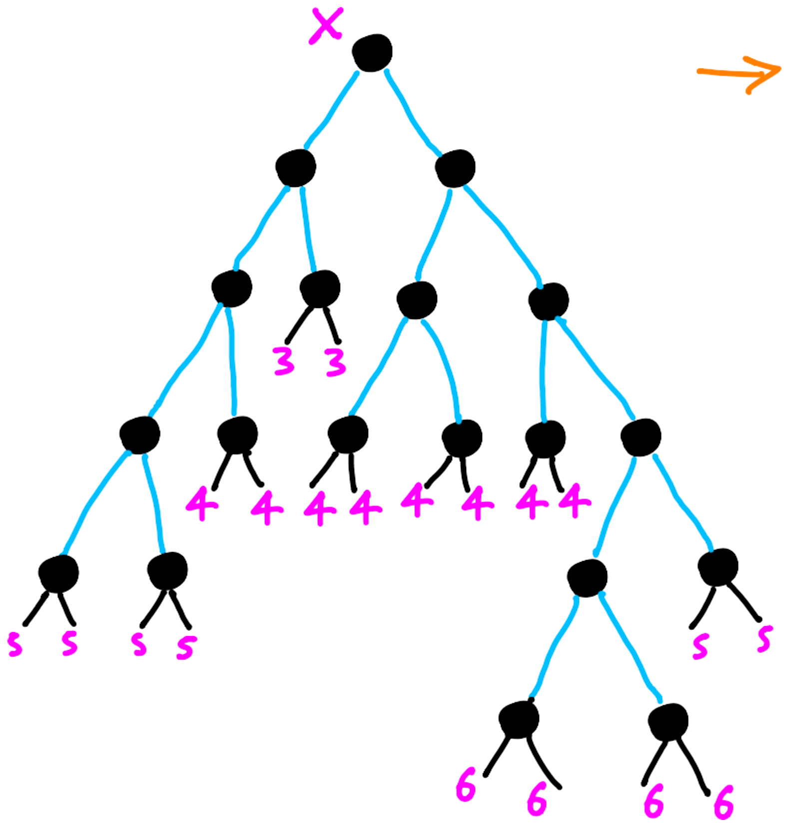
1) nodes are colored red or black.
2) root is always black.
3) add black "dummy" leaves so every "real" node has 2 children.

the important rules

4) every red node has a black parent.
5) for any node x: all paths down to leaves contain equal number of black nodes $= \underbrace{\text{black-height}[x]}_{\text{not including } x}$

4) every red node has a black parent.

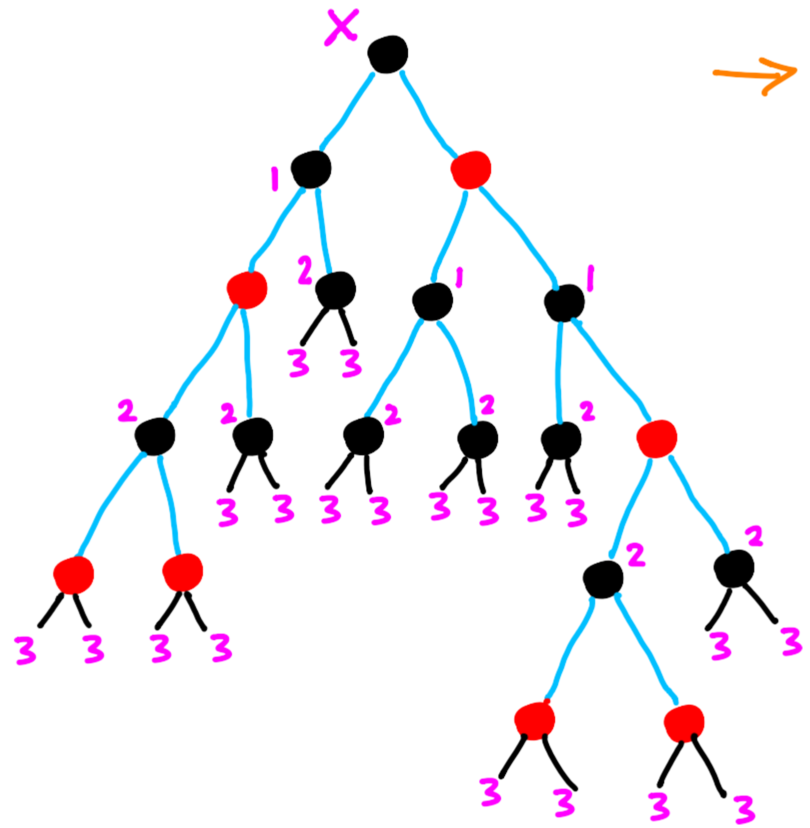
5) for any node x : all paths down to leaves contain equal number of black nodes = black-height[x]



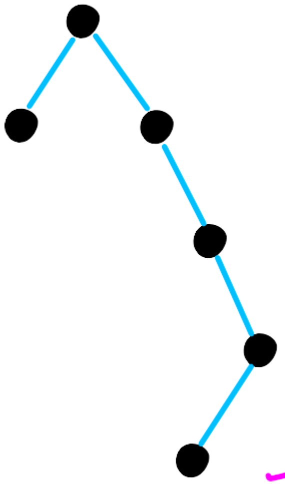
→ Fails rule 5

4) every red node has a black parent.

5) for any node x : all paths down to leaves contain equal number of black nodes = black-height[x]



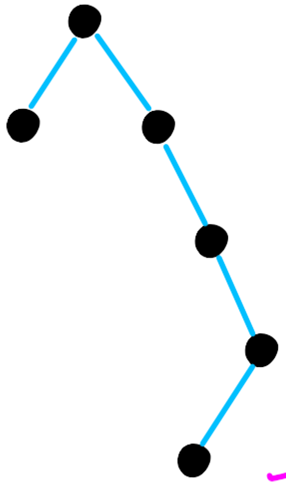
→ Fails rule 5 ⇒ fix by making some nodes red.



No hope to recolor

...

too unbalanced



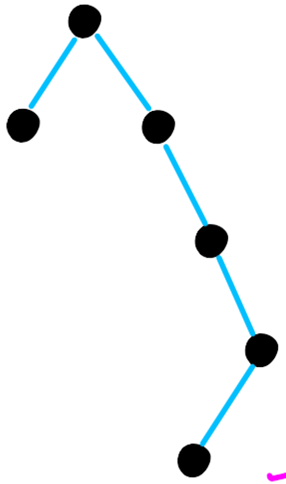
No hope to recolor

...

too unbalanced

Any root → leaf path of size k must have $\geq \frac{k}{2}$ black nodes.

Rule 4

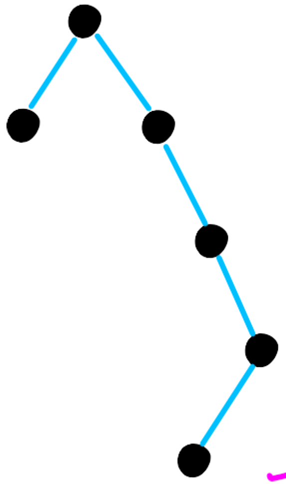


No hope to recolor
...
too unbalanced

Any root→leaf path of size k must have $\geq \frac{k}{2}$ black nodes.

Rule 4

So if any path is >2 times longer than another, we can't make it RB.



No hope to recolor

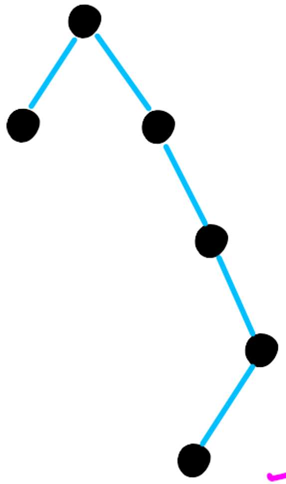
...

too unbalanced

Any root → leaf path of size k must have $\geq \frac{k}{2}$ black nodes. Rule 4

So if any path is >2 times longer than another, we can't make it RB.

Proof that RB trees have height < 2logn



No hope to recolor

...

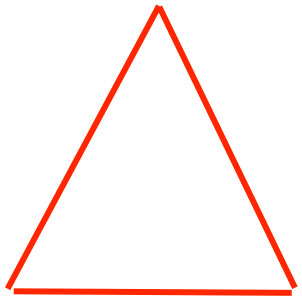
too unbalanced

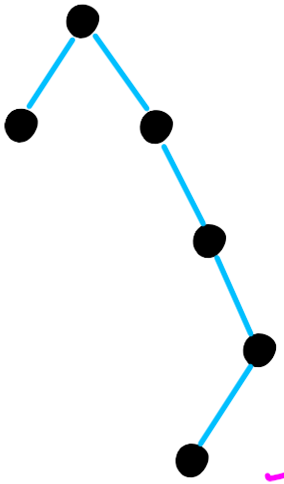
Any root→leaf path of size k must have $\geq \frac{k}{2}$ black nodes. Rule 4

So if any path is >2 times longer than another, we can't make it RB.

Proof that RB trees have height < 2logn

1) Fact: If a tree is perfectly balanced the height is logn





No hope to recolor

...

too unbalanced

Any root → leaf path of size k must have $\geq \frac{k}{2}$ black nodes.

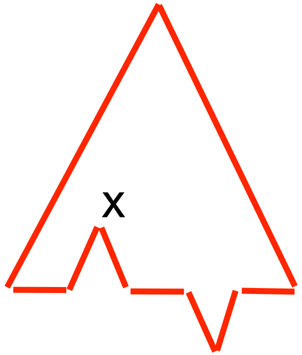
Rule 4

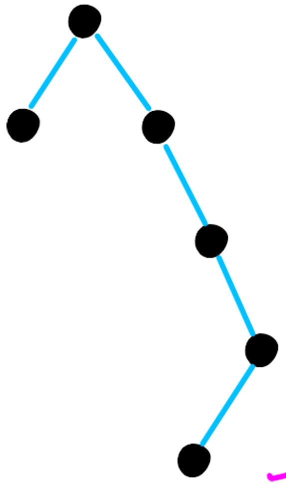
So if any path is >2 times longer than another, we can't make it RB.

Proof that RB trees have height < 2logn

1) Fact: If a tree is perfectly balanced the height is logn

2) If a tree is not perfectly balanced, there is a node, x, at depth d < logn





No hope to recolor

...

too unbalanced

Any root → leaf path of size k must have $\geq \frac{k}{2}$ black nodes. Rule 4

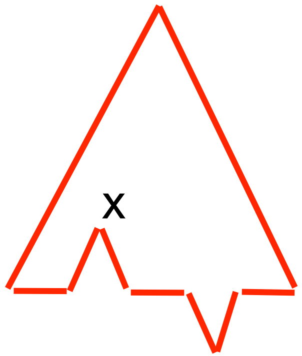
So if any path is >2 times longer than another, we can't make it RB.

Proof that RB trees have height < 2logn

1) Fact: If a tree is perfectly balanced the height is logn

2) If a tree is not perfectly balanced, there is a node, x, at depth d < logn

3) By our claim above there can't be any other node at depth > 2d



□

We have seen that RB trees are reasonably balanced : $\sim 2\log n$

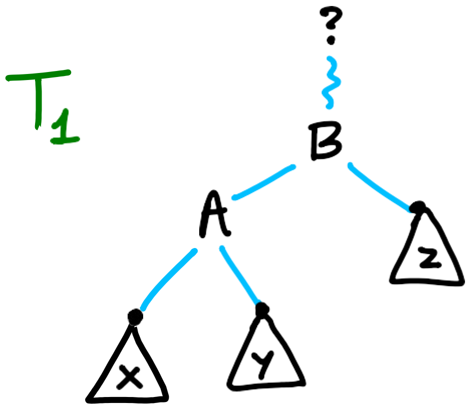
↳ search, min, max, next, prev : $O(\log n)$ time.

Next: how to update RB trees (insert, delete)

We have seen that RB trees are reasonably balanced: $\sim 2\log n$

↳ search, min, max, next, prev : $O(\log n)$ time.

Next: how to update RB trees (insert, delete)



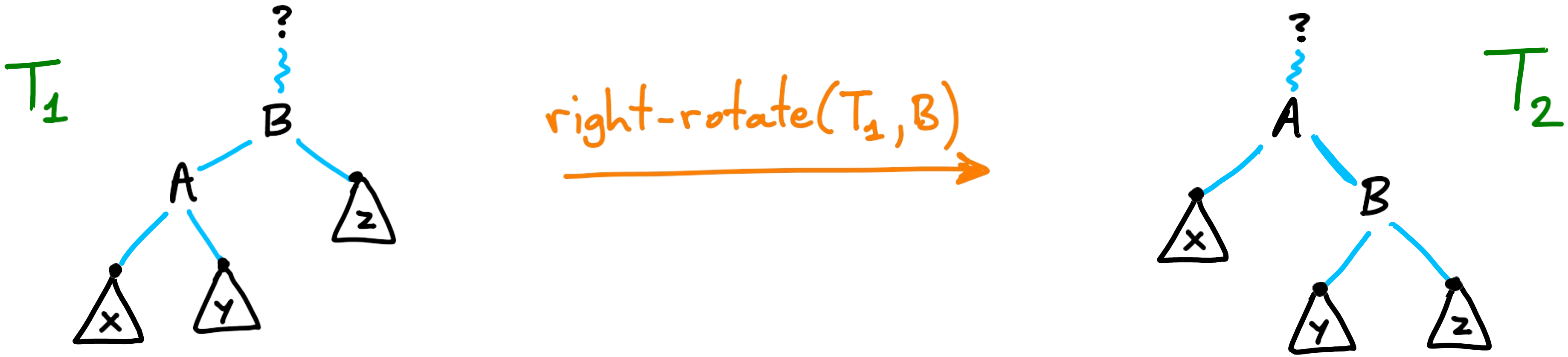
$x \leq A \leq y \leq B \leq z$

We have seen that RB trees are reasonably balanced : ~2logn

↳ search, min, max, next, prev : O(logn) time.

Next: how to update RB trees (insert, delete)

ROTATIONS in arbitrary BSTs



$X \le A \le Y \le B \le Z$ $\qquad\qquad$ $X \le A \le Y \le B \le Z$

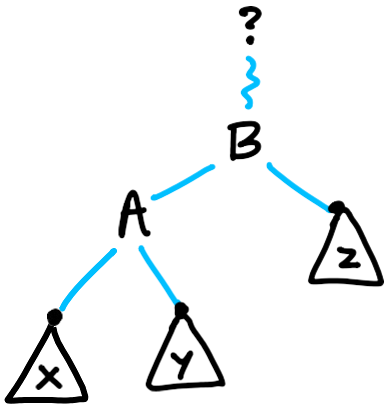
We have seen that RB trees are reasonably balanced : $\sim 2\log n$

↳ search, min, max, next, prev : $O(\log n)$ time.

Next: how to update RB trees (insert, delete)

ROTATIONS in arbitrary BSTs

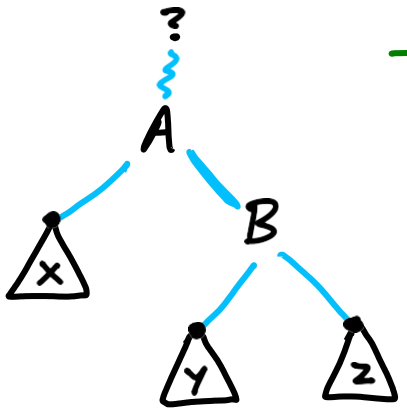
T_1



right-rotate(T_1, B) →

← left-rotate(T_2, A)

T_2



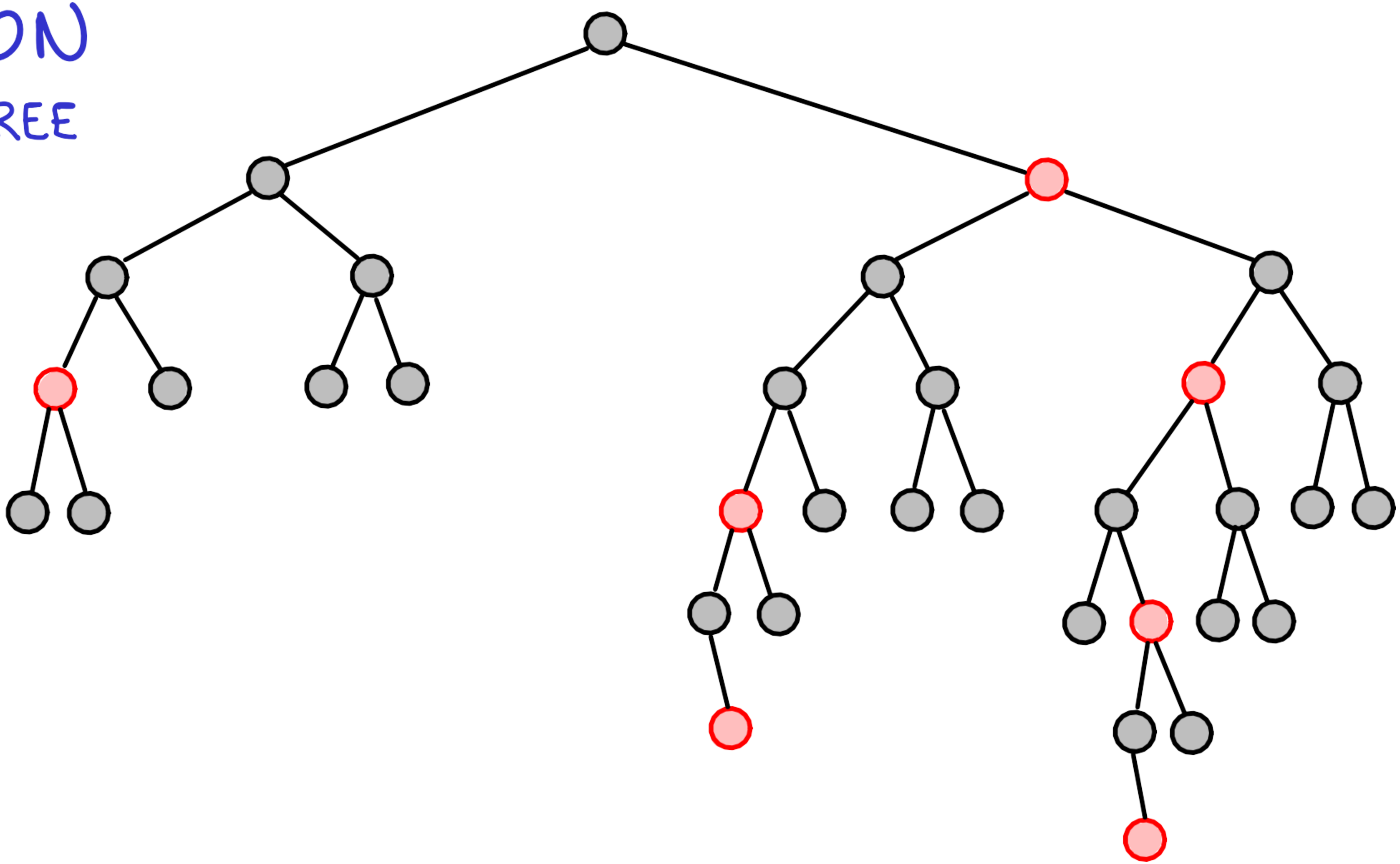
$X \le A \le Y \le B \le Z$

$X \le A \le Y \le B \le Z$

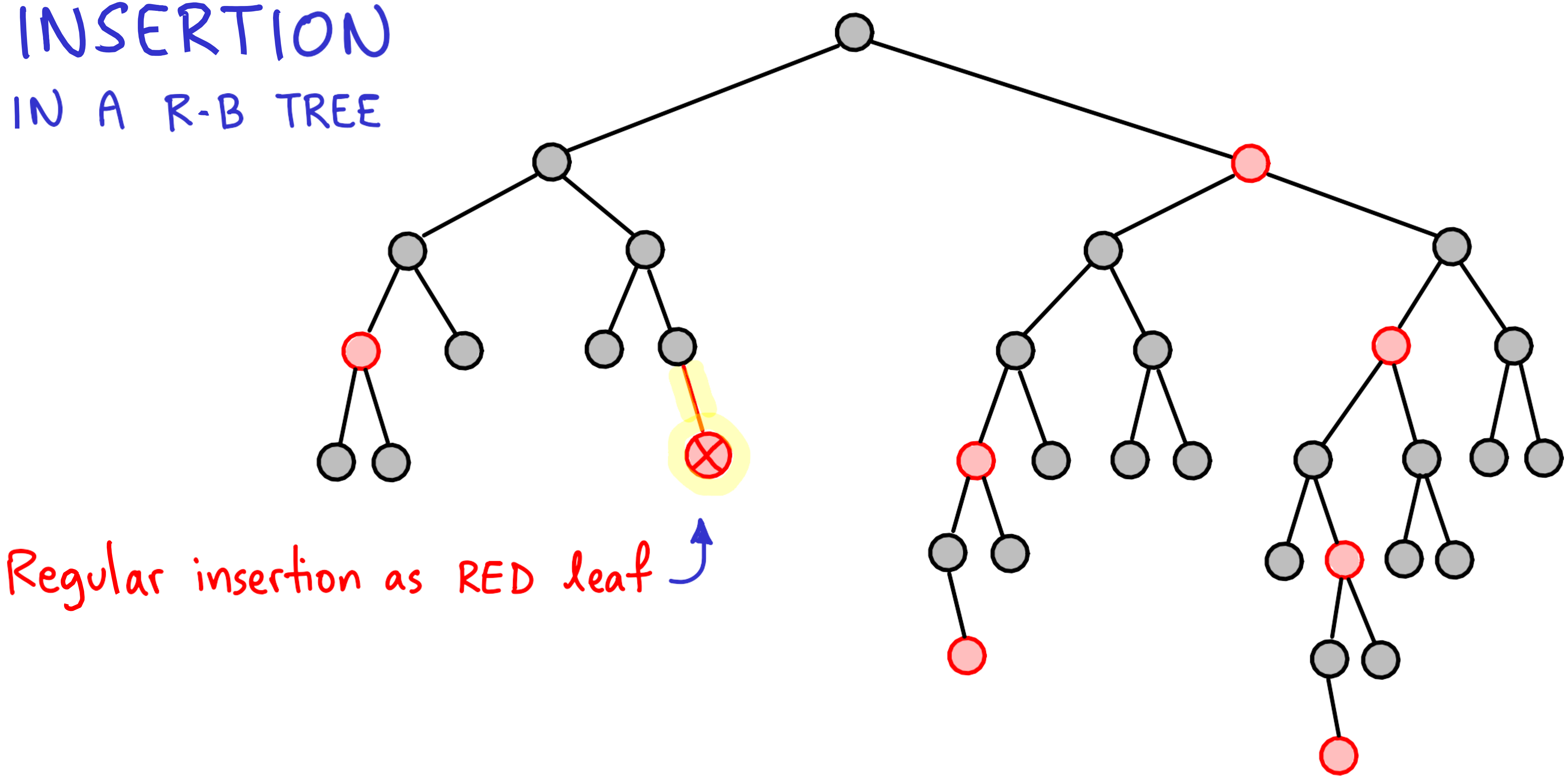
$O(1)$ time

INSERTION

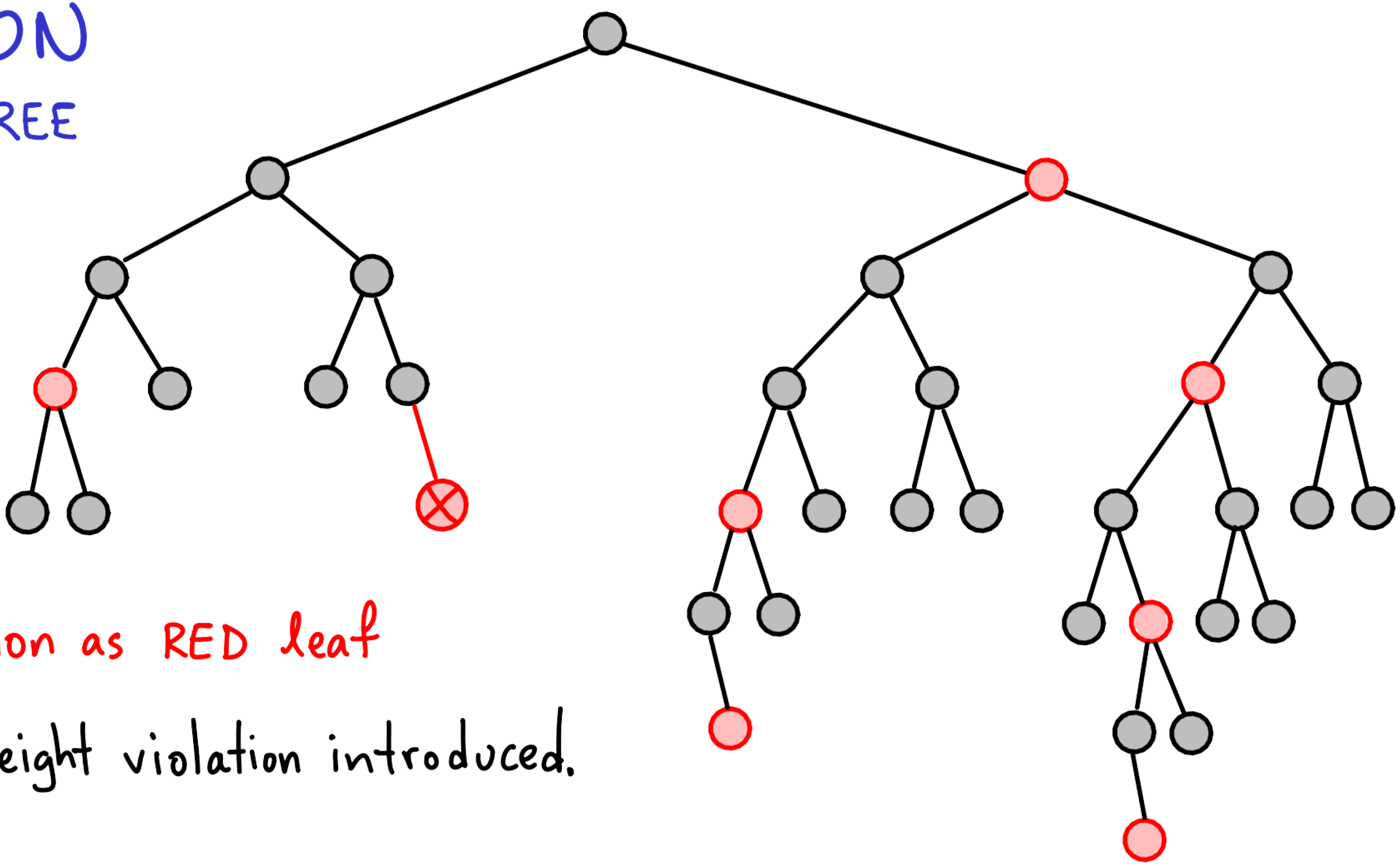
IN A R-B TREE



INSERTION
IN A R-B TREE



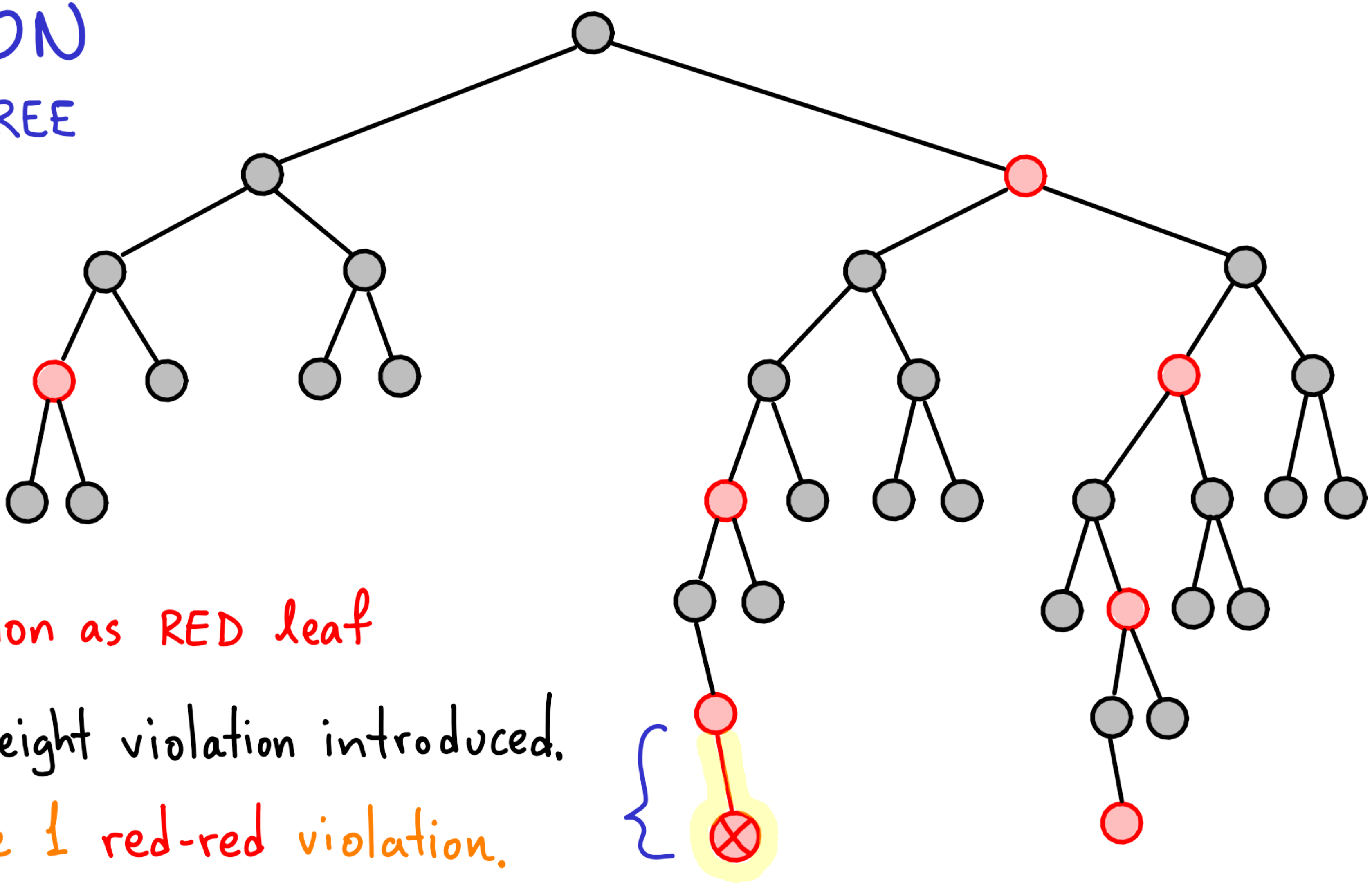
INSERTION
IN A R-B TREE



Regular insertion as RED leaf

↳ No black height violation introduced.

INSERTION
IN A R-B TREE



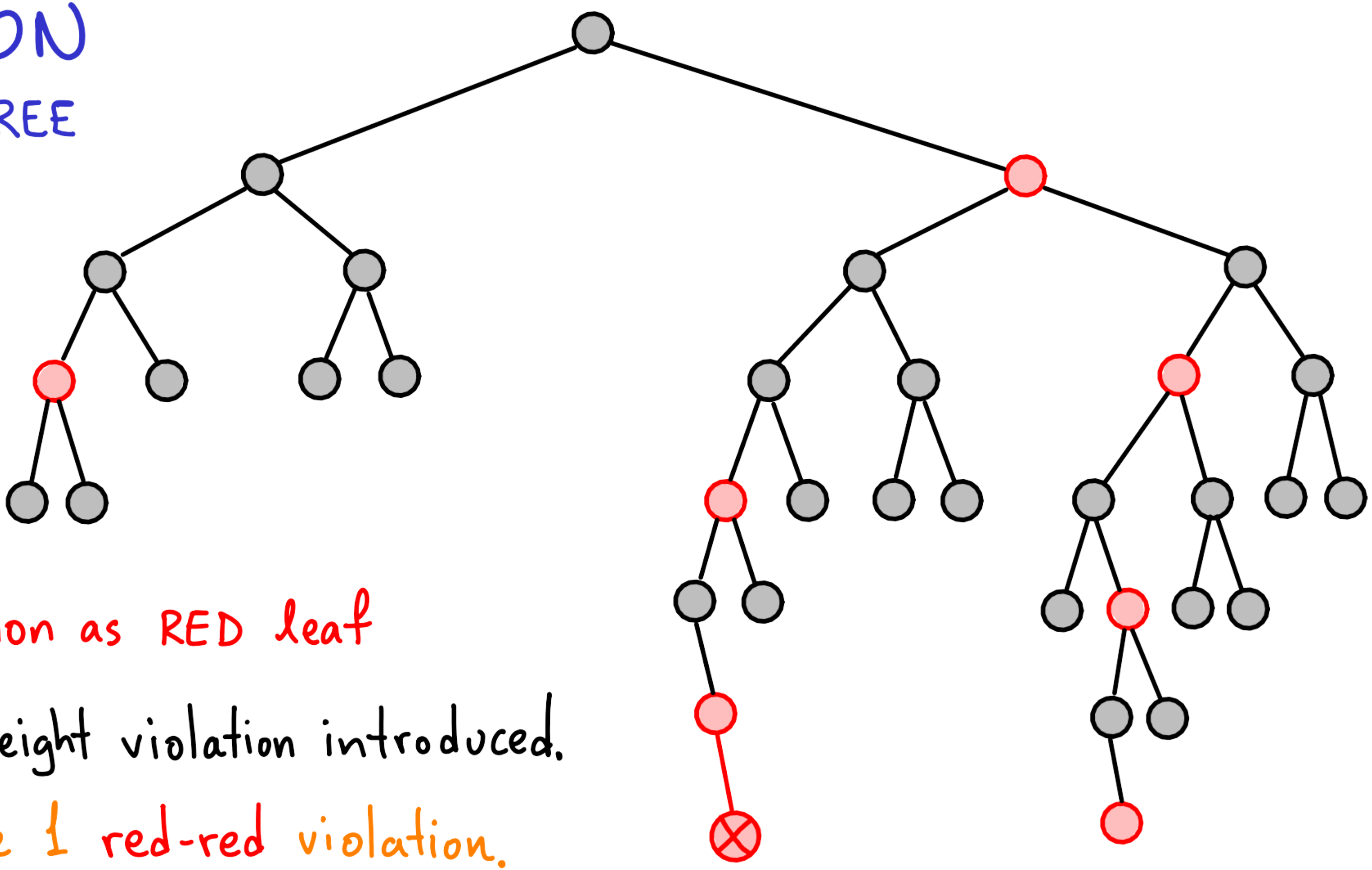
Regular insertion as RED leaf

↳ No black height violation introduced.

↳ Might have 1 red-red violation.

INSERTION

IN A R-B TREE



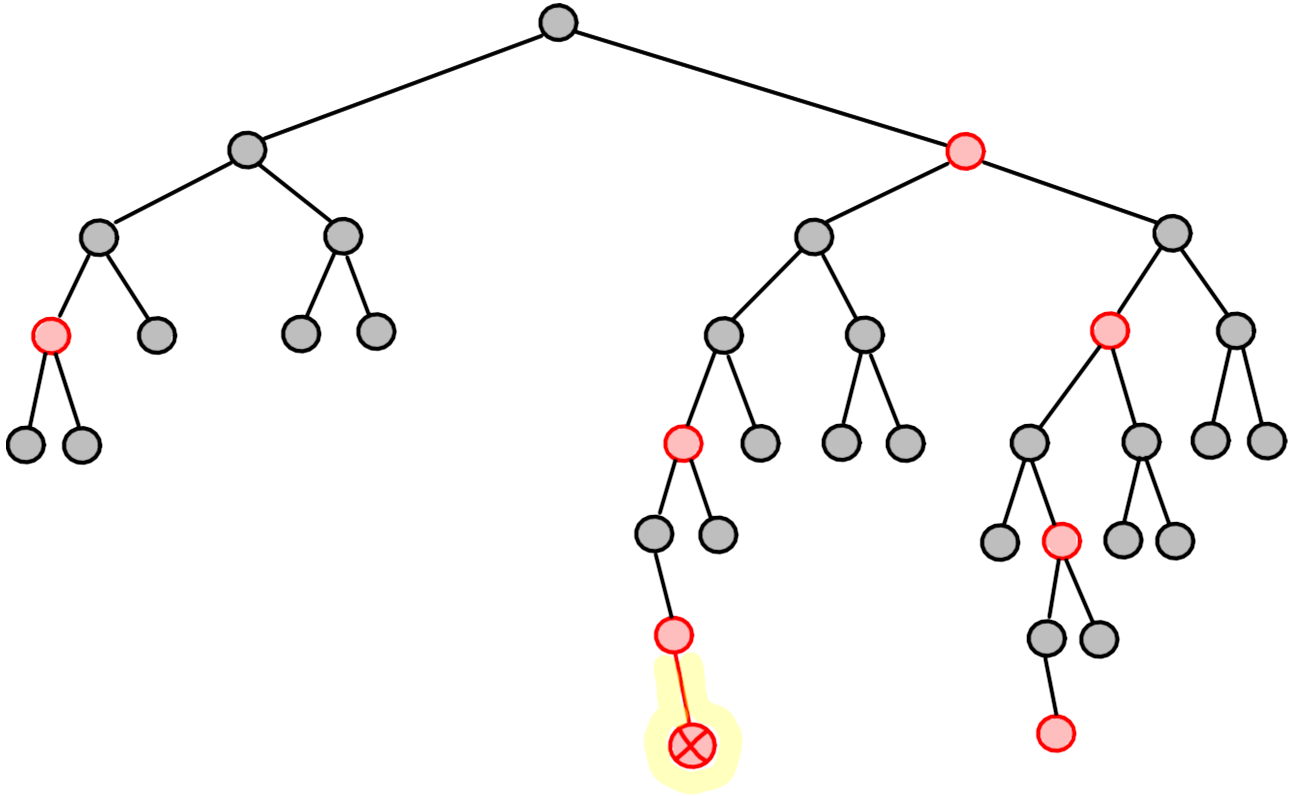
Regular insertion as RED leaf

↳ No black height violation introduced.

↳ Might have 1 red-red violation.

↳ If so, start fixing: locally & upward

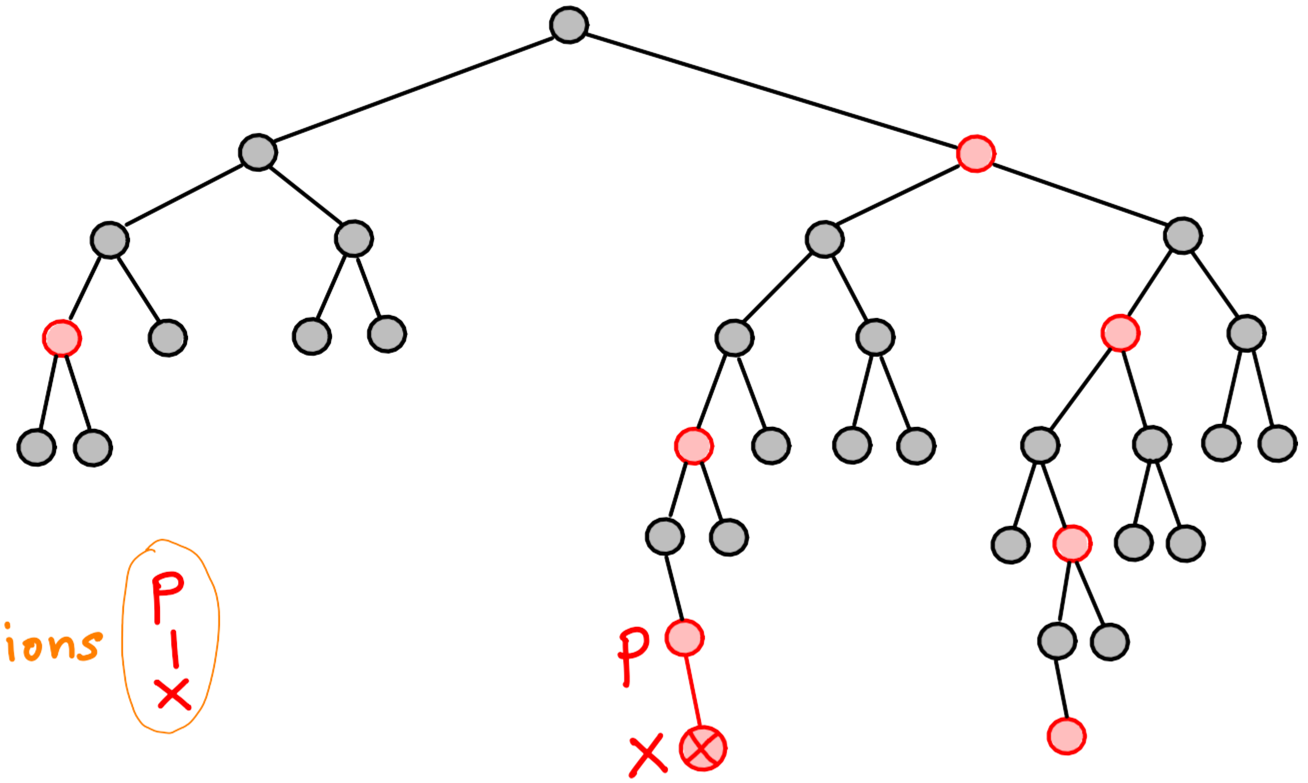
fixing: locally & upward



fixing: locally & upward

Loop invariants:

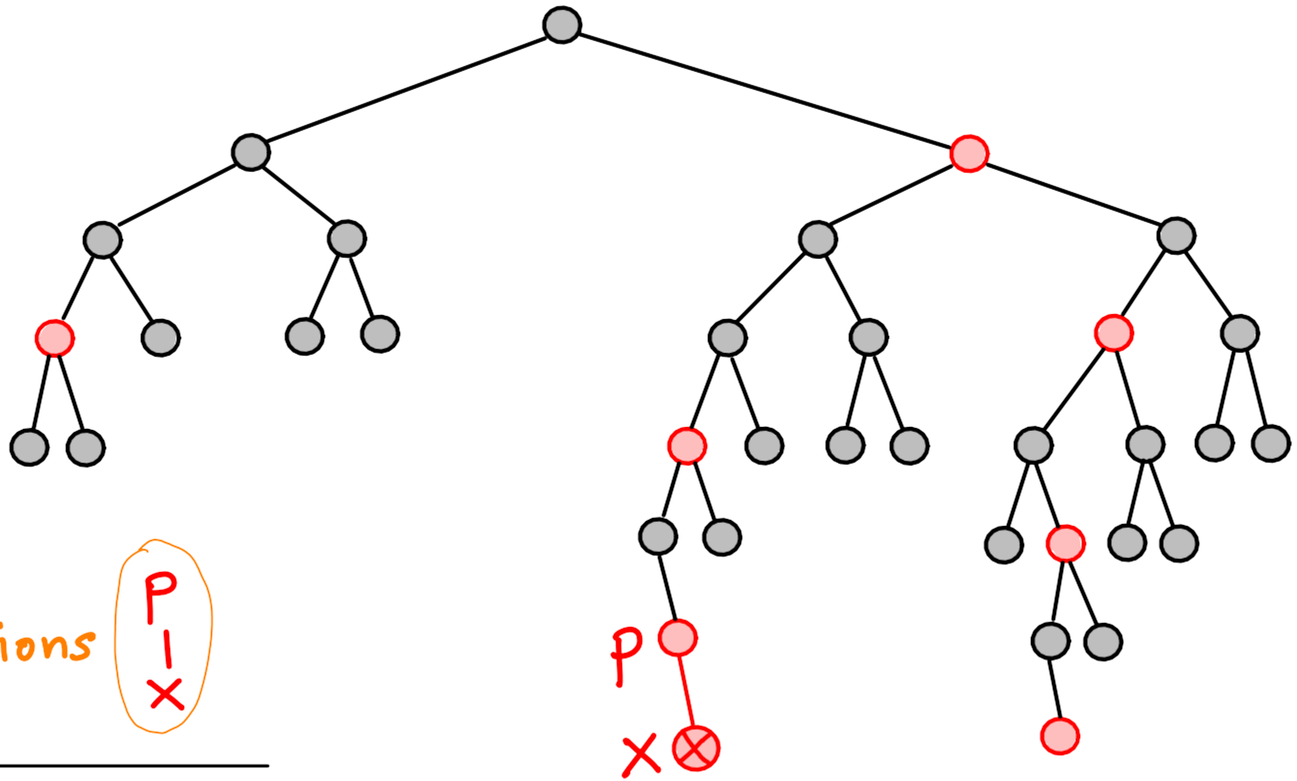
- No black height violation.
- 1 red-red violation, at positions p | x



fixing: locally & upward

Loop invariants:

- No black height violation.
- 1 red-red violation, at positions p – x



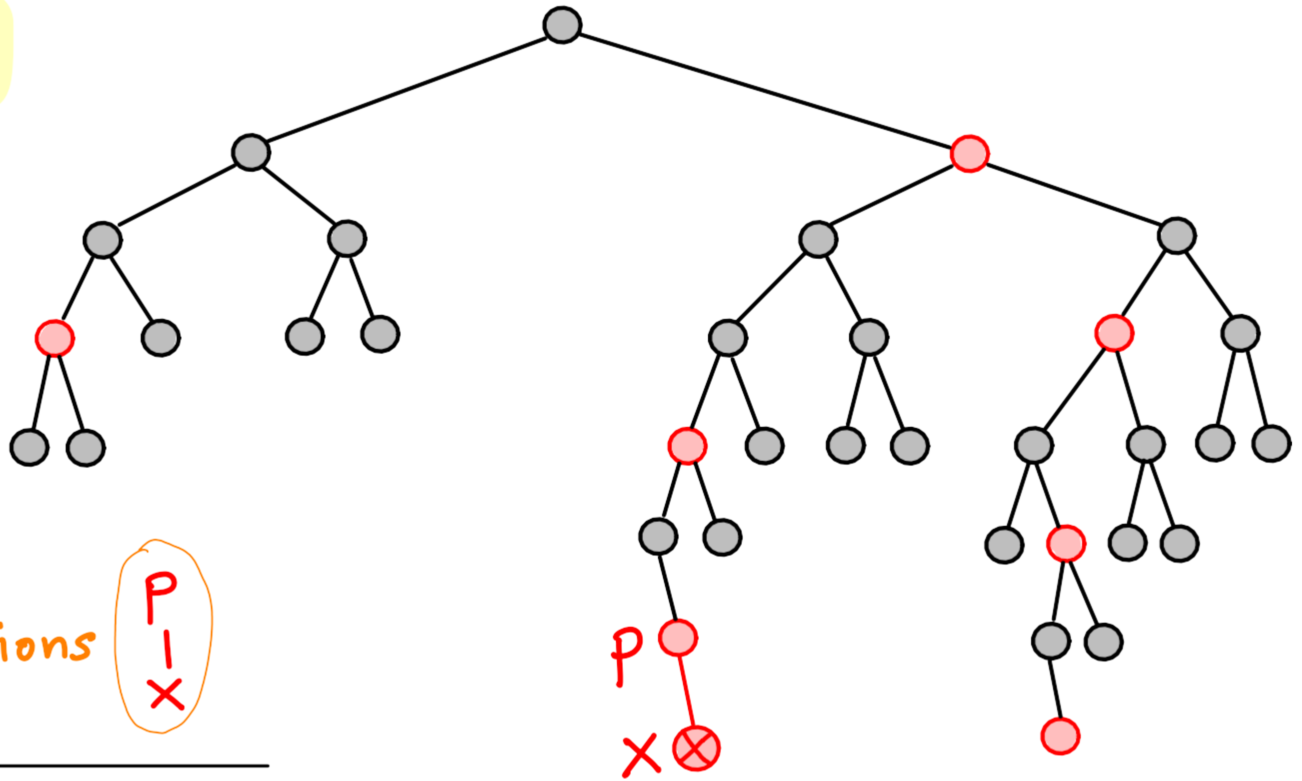
Loop outcomes:

Obtain R-B tree

OR

Redefine ancestor as x

fixing: locally & upward



Loop invariants:

- No black height violation.
- 1 red-red violation, at positions p | x

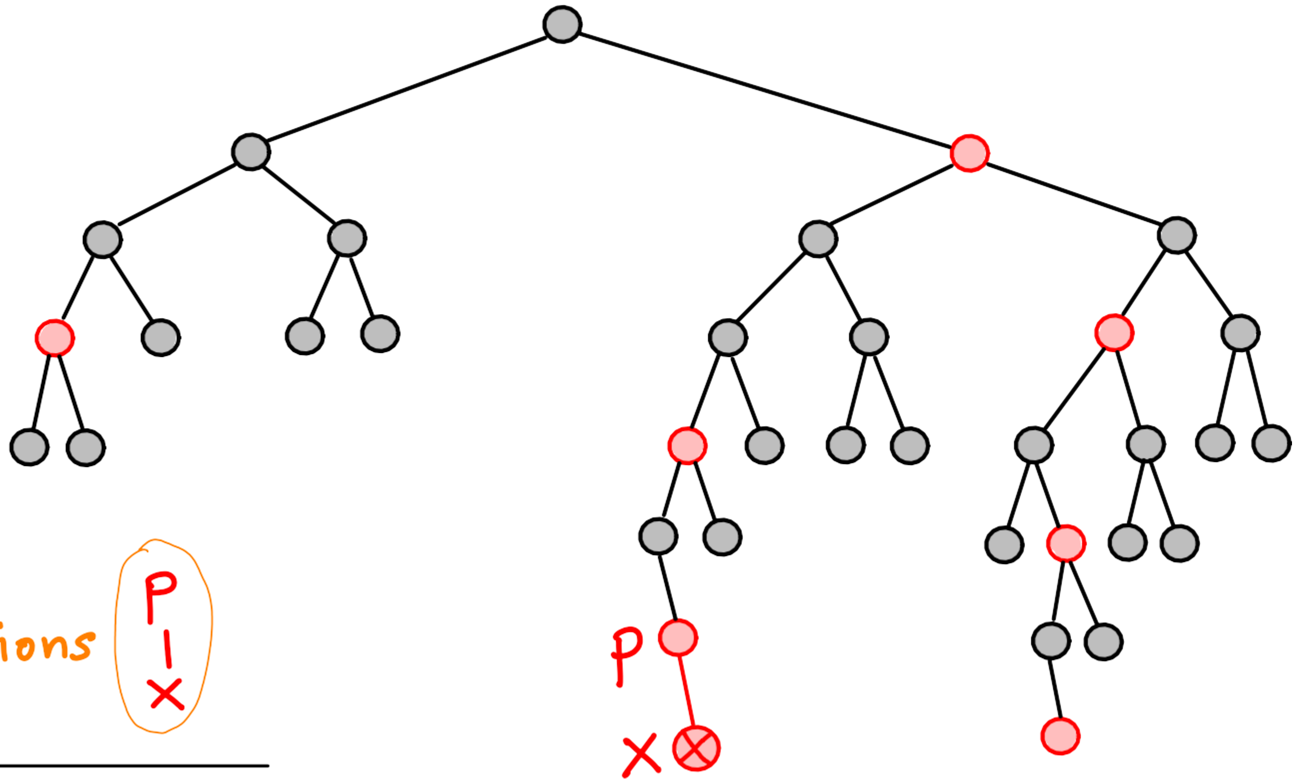
Loop outcomes:

Obtain R-B tree

OR

Redefine ancestor as x = fix upward

fixing: locally & upward



Loop invariants:

- No black height violation.
- 1 red-red violation, at positions p | x

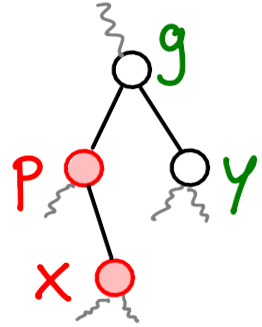
Loop outcomes:

Obtain R-B tree

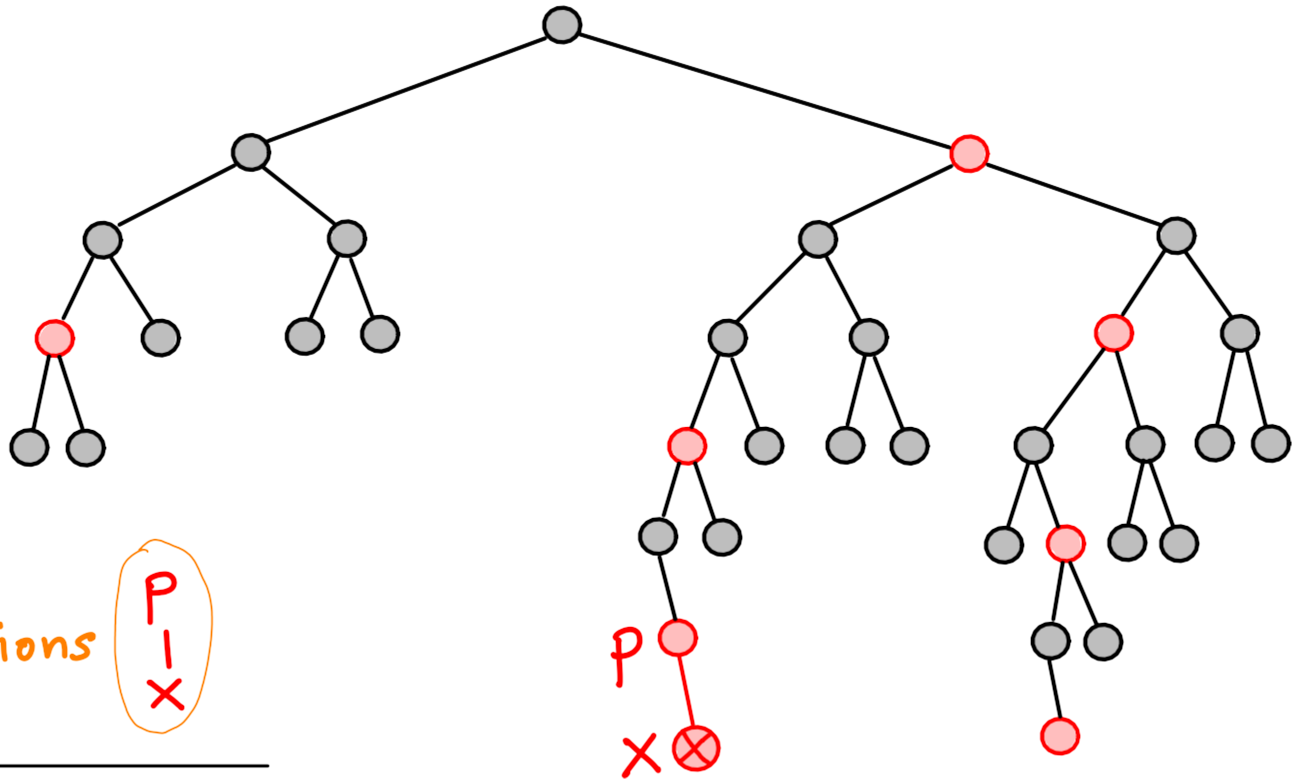
OR

Redefine ancestor as x = fix upward

Local positions considered in loop



fixing: locally & upward



Loop invariants:

- No black height violation.
- 1 red-red violation, at positions p | x

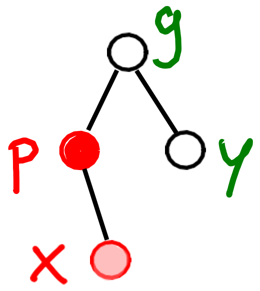
Loop outcomes:

Obtain R-B tree

OR

Redefine ancestor as x = fix upward

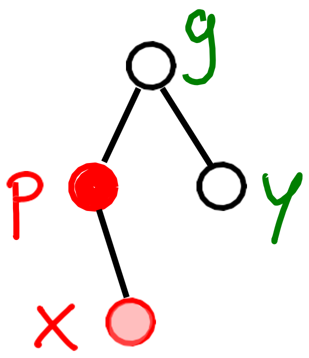
Local positions considered in loop



x & p : both red if we needed to continue loop

g: grandparent of x

y: "uncle" of x

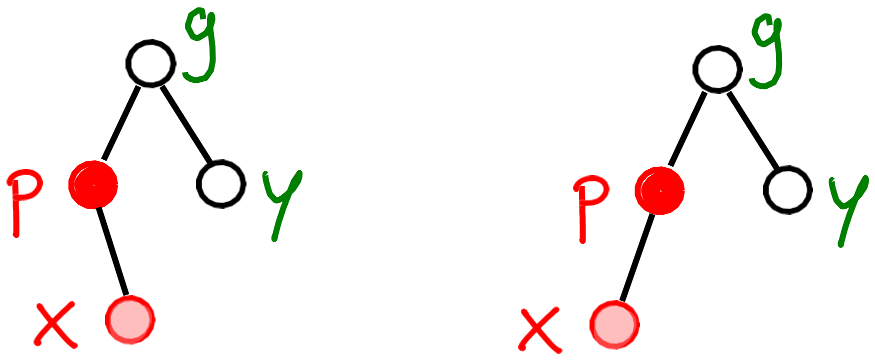


x & p : both red if we needed to continue loop

g: grandparent of x

y: "uncle" of x

2 possible shapes...

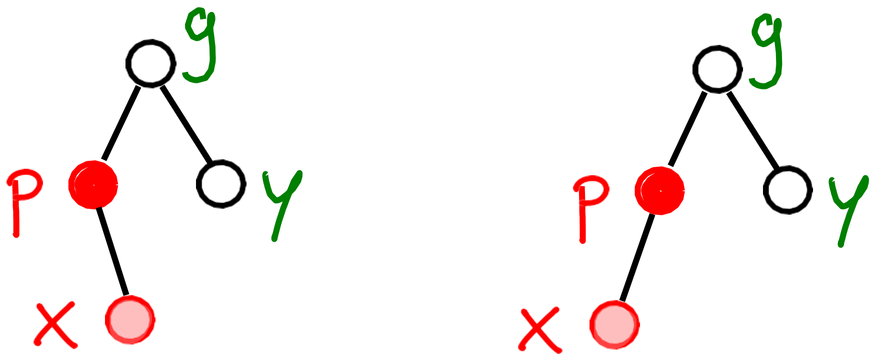


x & p : both red if we needed to continue loop

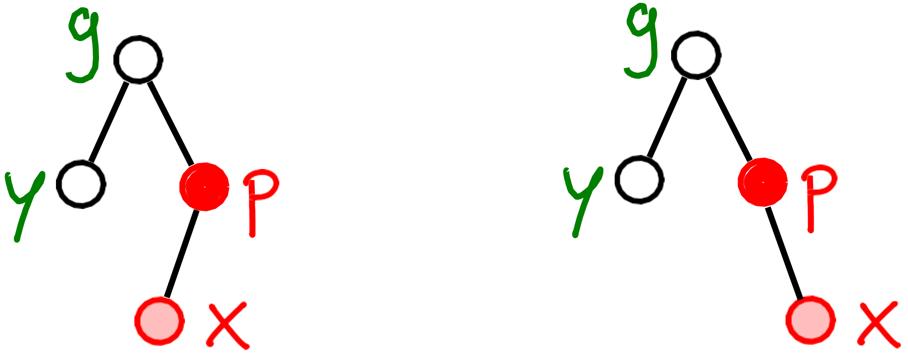
g: grandparent of x

y: "uncle" of x

2 possible shapes...



...and their mirror images

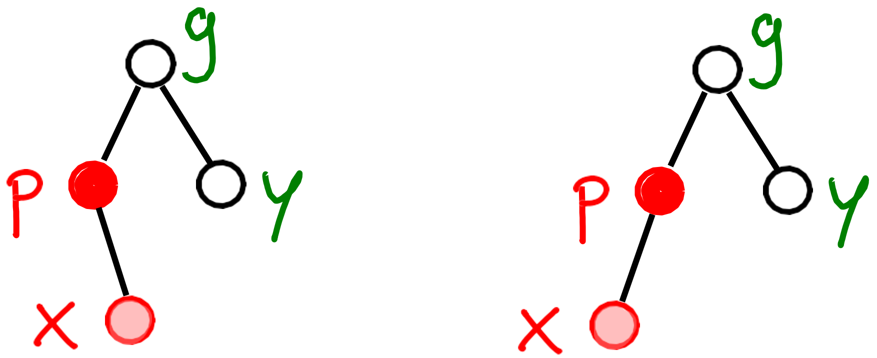


x & p : both red if we needed to continue loop

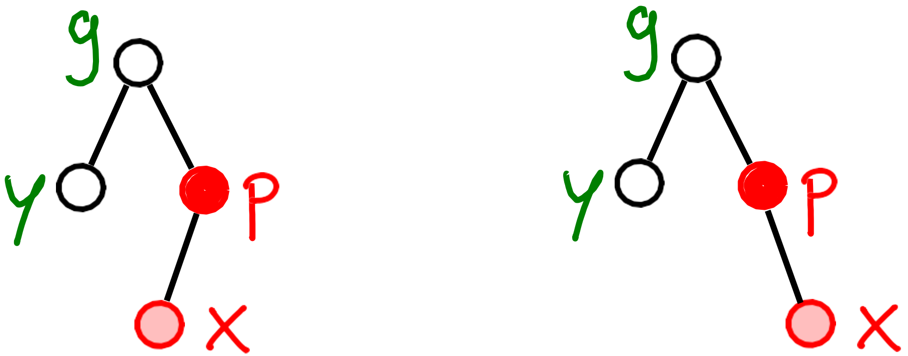
g: grandparent of x

y: "uncle" of x

2 possible shapes...



...and their mirror images



g must exist
because p can't be root

y must exist
(could be a fake leaf)

x & p : both red if we needed to continue loop

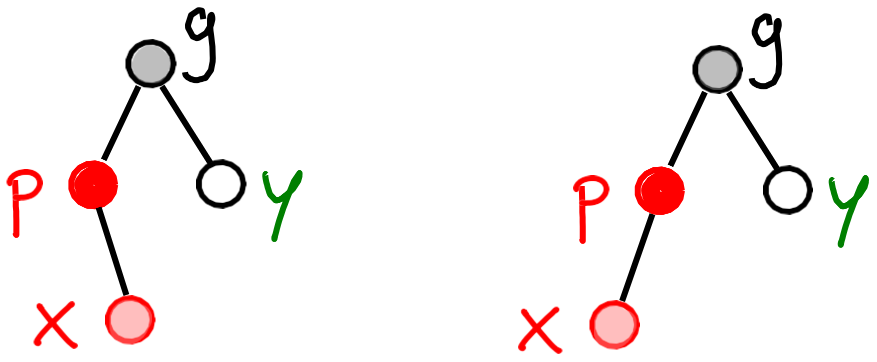
g: grandparent of x

y: "uncle" of x

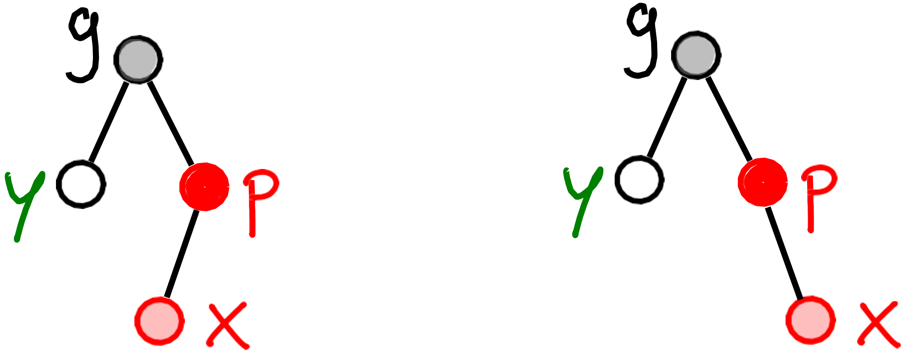
g must exist: must be black
because p can't be root

y must exist
(could be a fake leaf)

2 possible shapes...



...and their mirror images



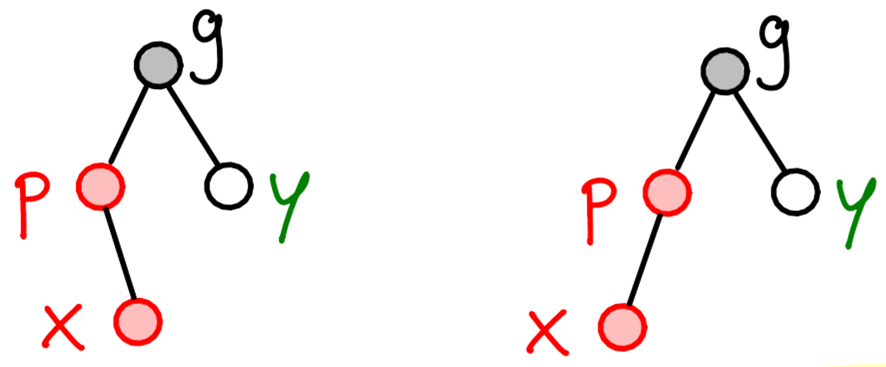
x & p : both red if we needed to continue loop

g: grandparent of x

y: "uncle" of x

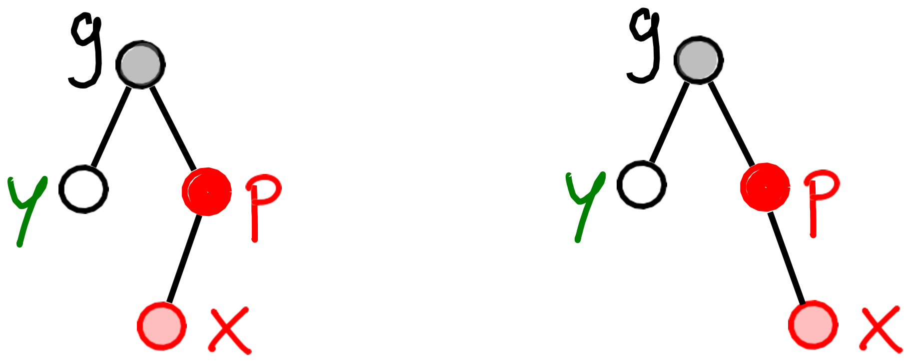
We will handle the 2 main shapes:

2 possible shapes...



...and their mirror images

The others can be done by symmetry:

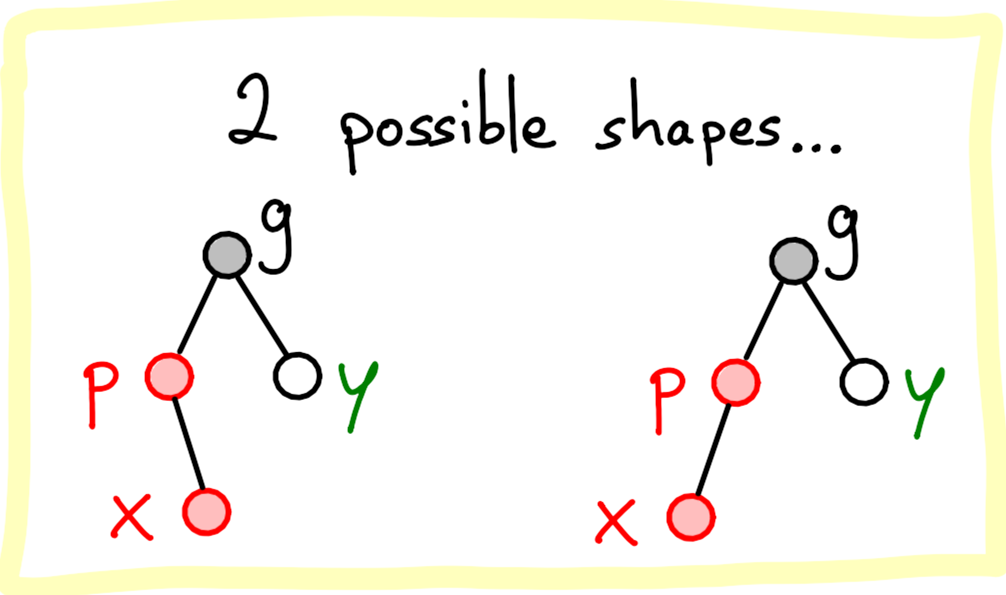


x & p : both red if we needed to continue loop

g: grandparent of x

y: "uncle" of x

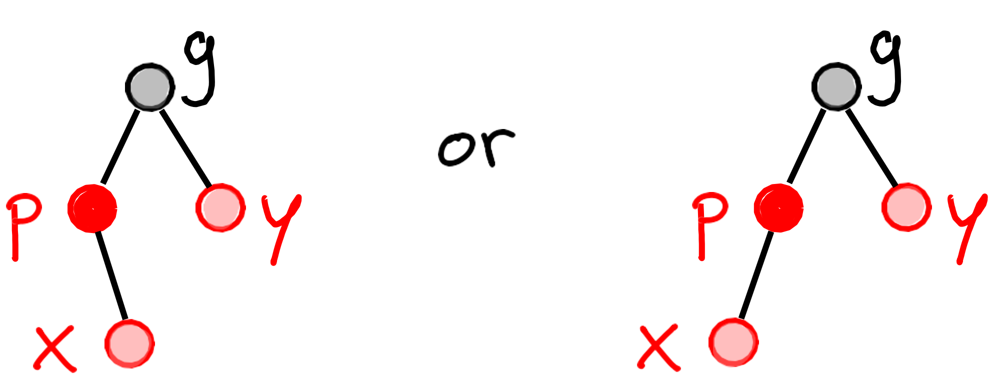
We will handle the 2 main shapes:



Case 1 : y is red

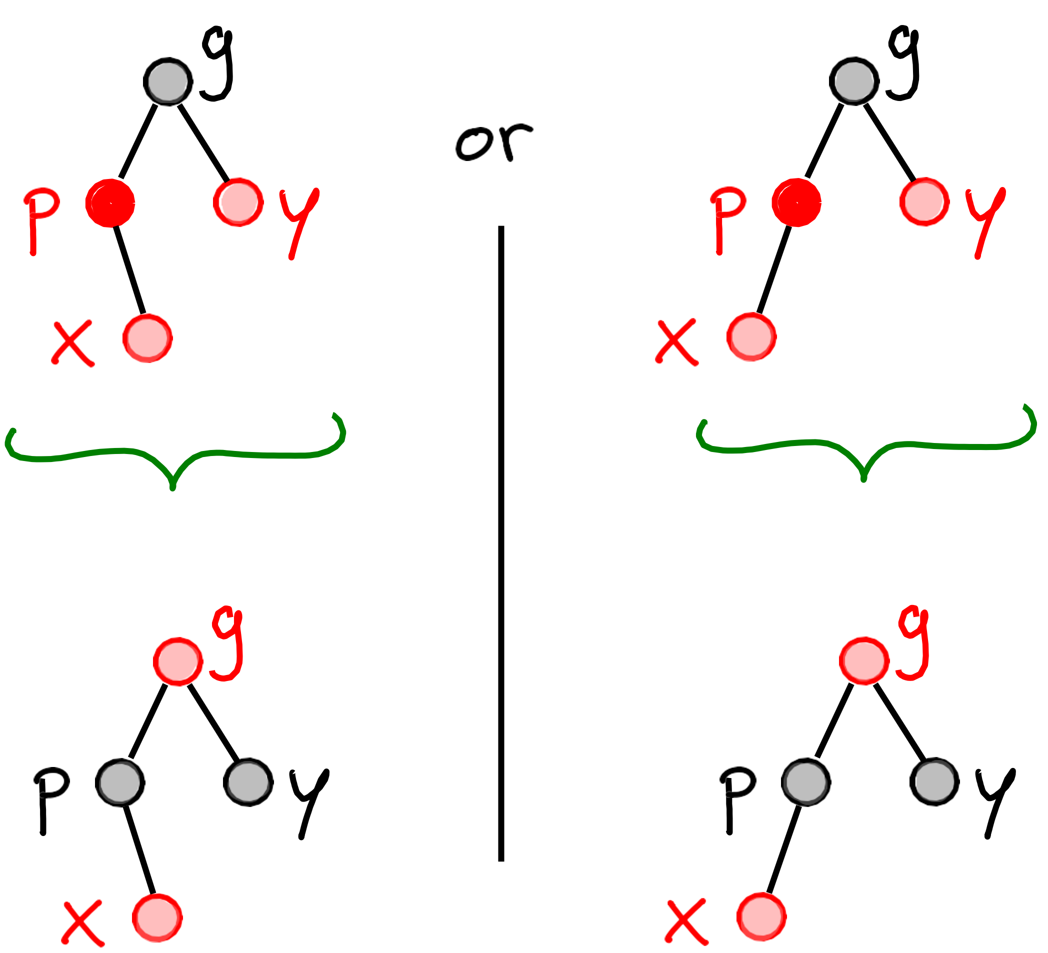
Cases 2 & 3 : y is black

Case 1 : y is red



Case 1 : y is red

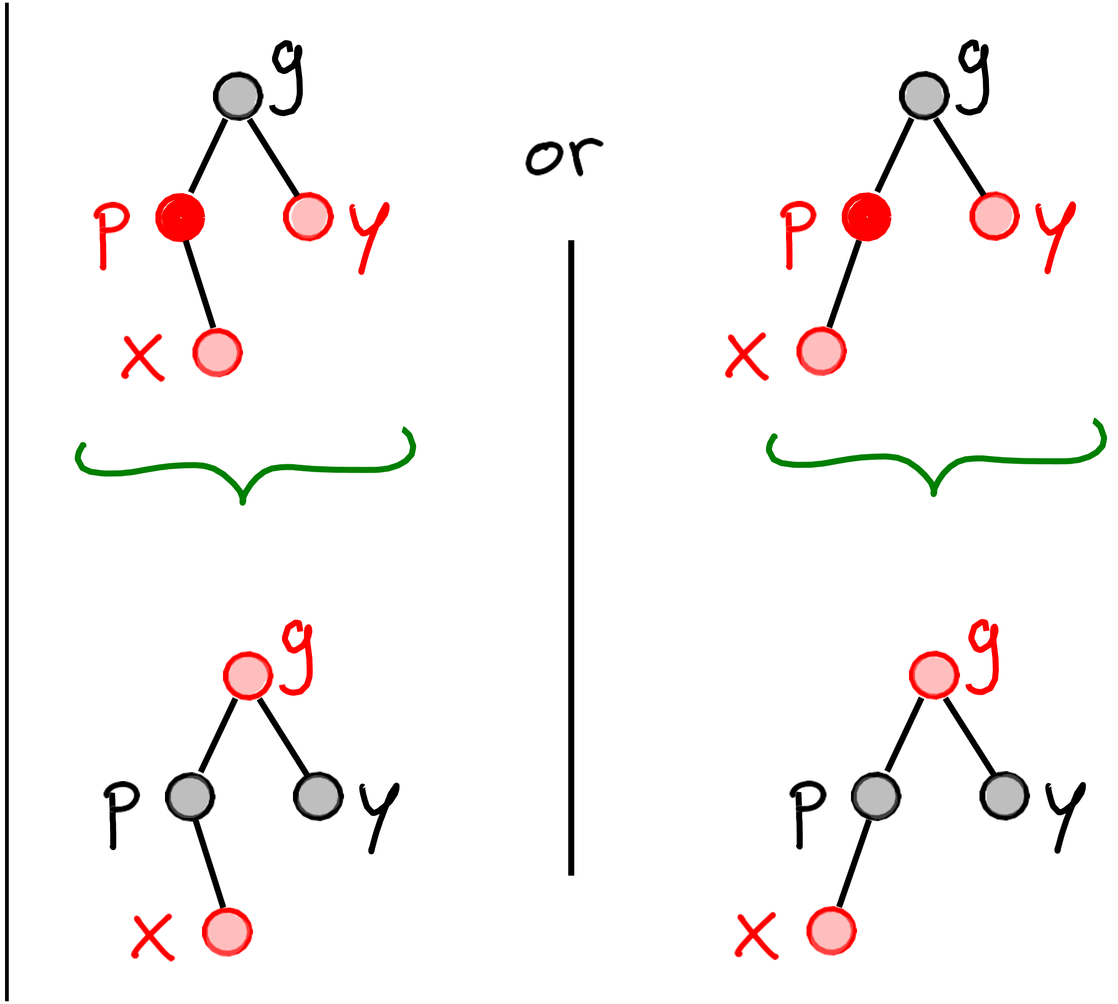
↳ Recolor p, g, y



Case 1 : y is red

↳ Recolor p, g, y

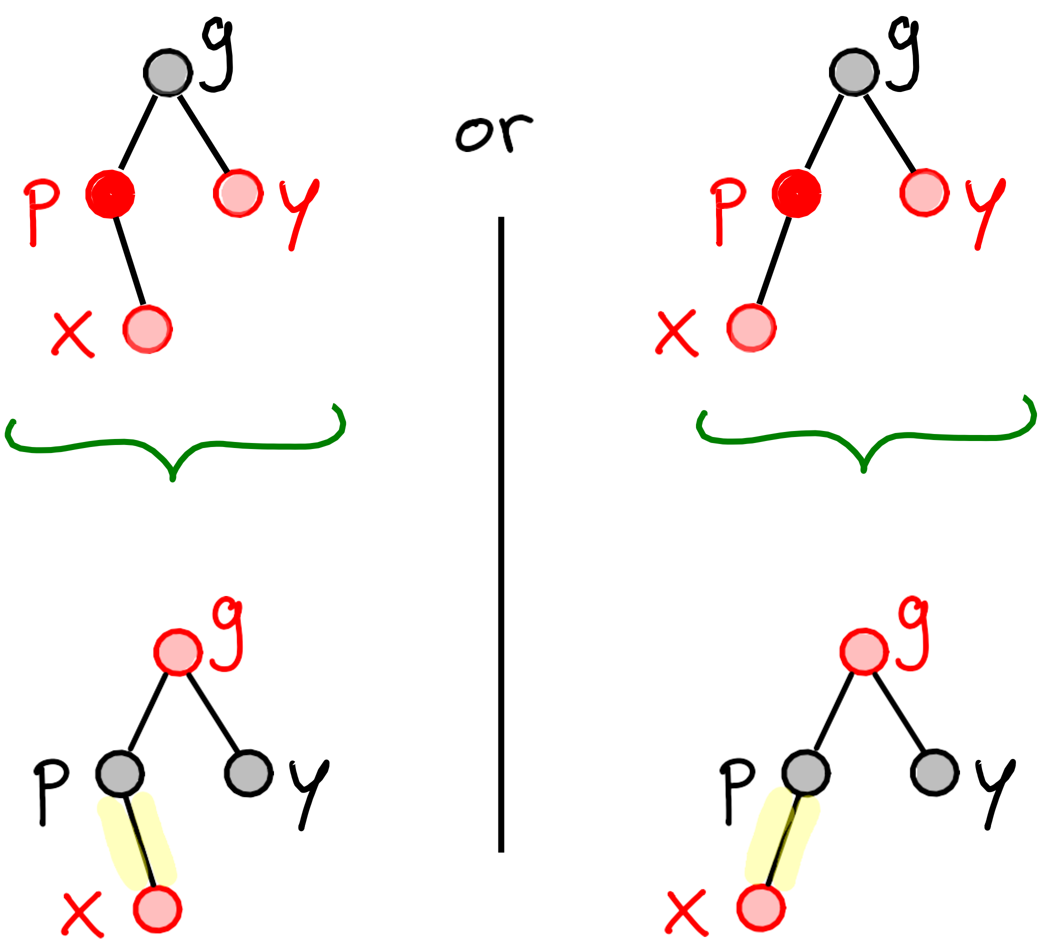
- Preserve black-height invariant.



Case 1 : y is red

↳ Recolor p, g, y

- Preserve black-height invariant
  (total B nodes on any path from global root down to leaves)
- Eliminate p–x violation.

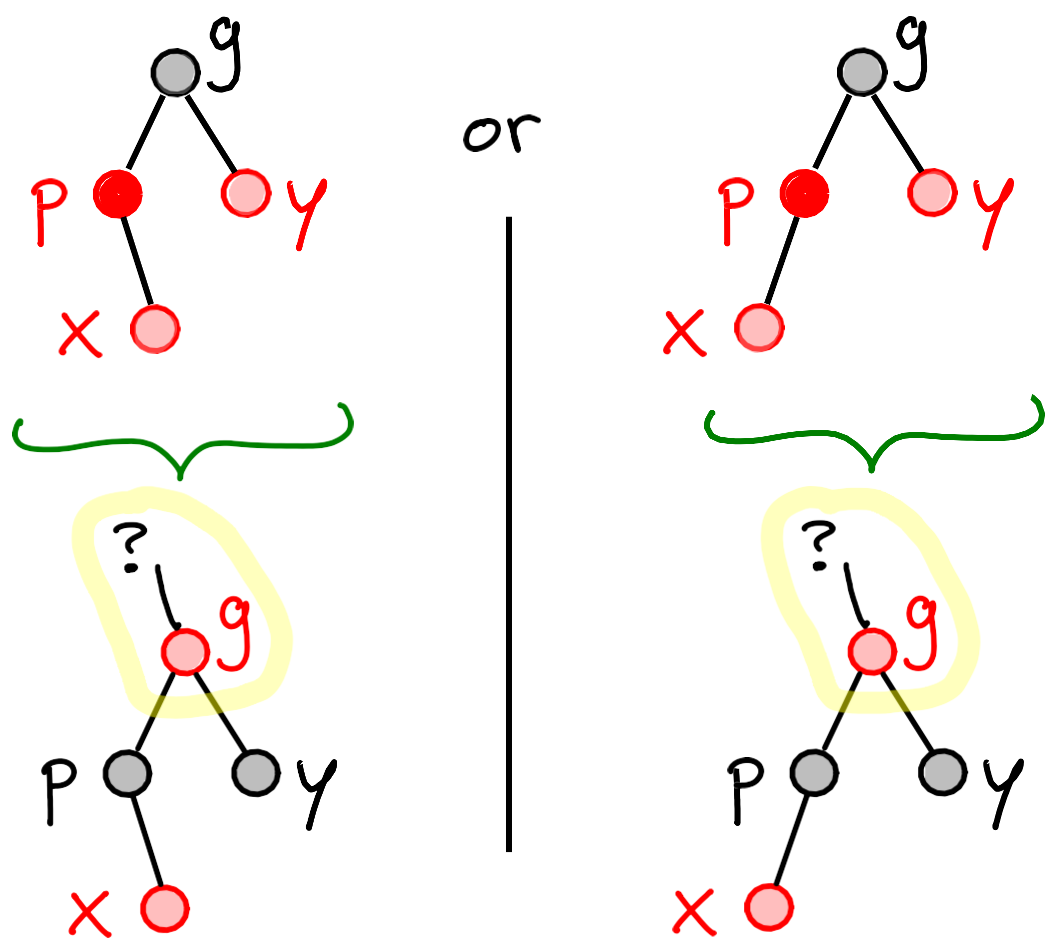


Case 1: y is red

↳ Recolor p, g, y

- Preserve black-height invariant.
  (total B nodes on any path from global root down to leaves)
- Eliminate p–x violation.
- If no new violation ? ↕ g then DONE

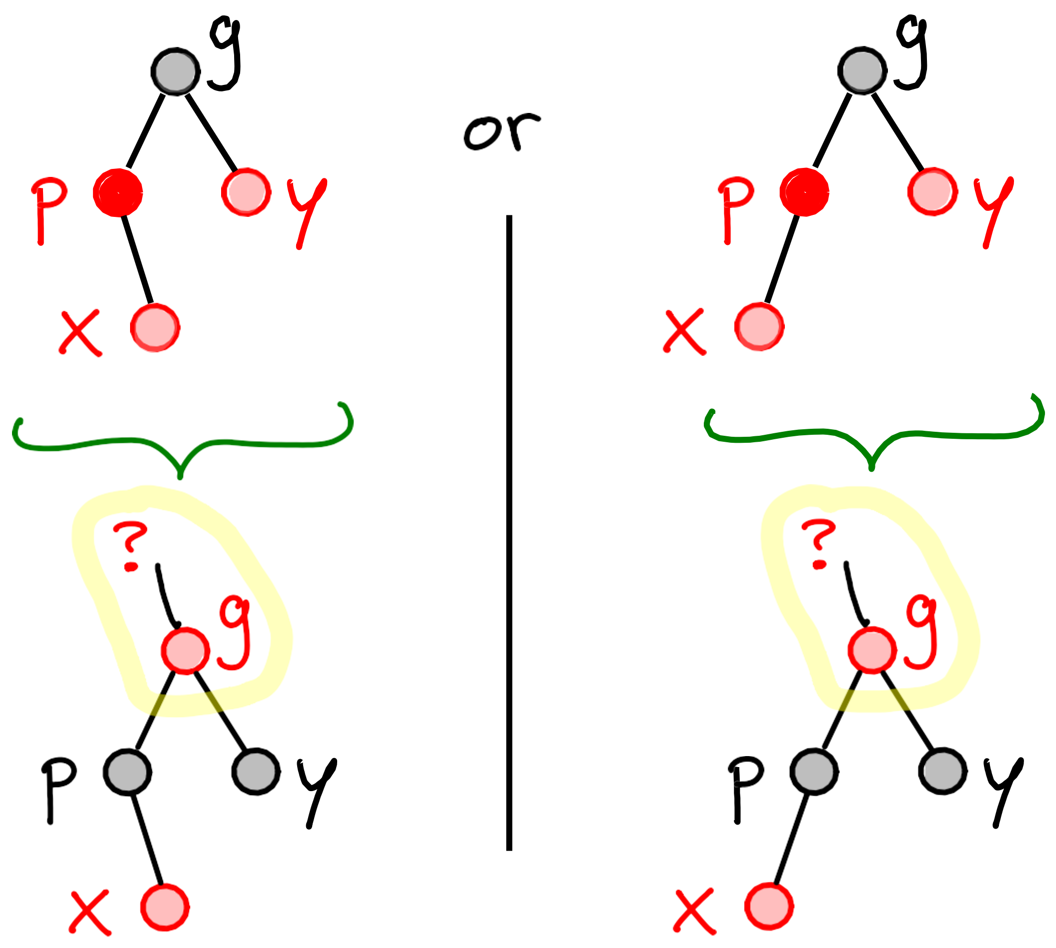
(note: if g = root
color g black)



Case 1 : y is red

↳ Recolor p, g, y

- Preserve black-height invariant.
  (total B nodes on any path from global root down to leaves)
- Eliminate p–x violation.
- Might cause violation ?–g
  otherwise DONE

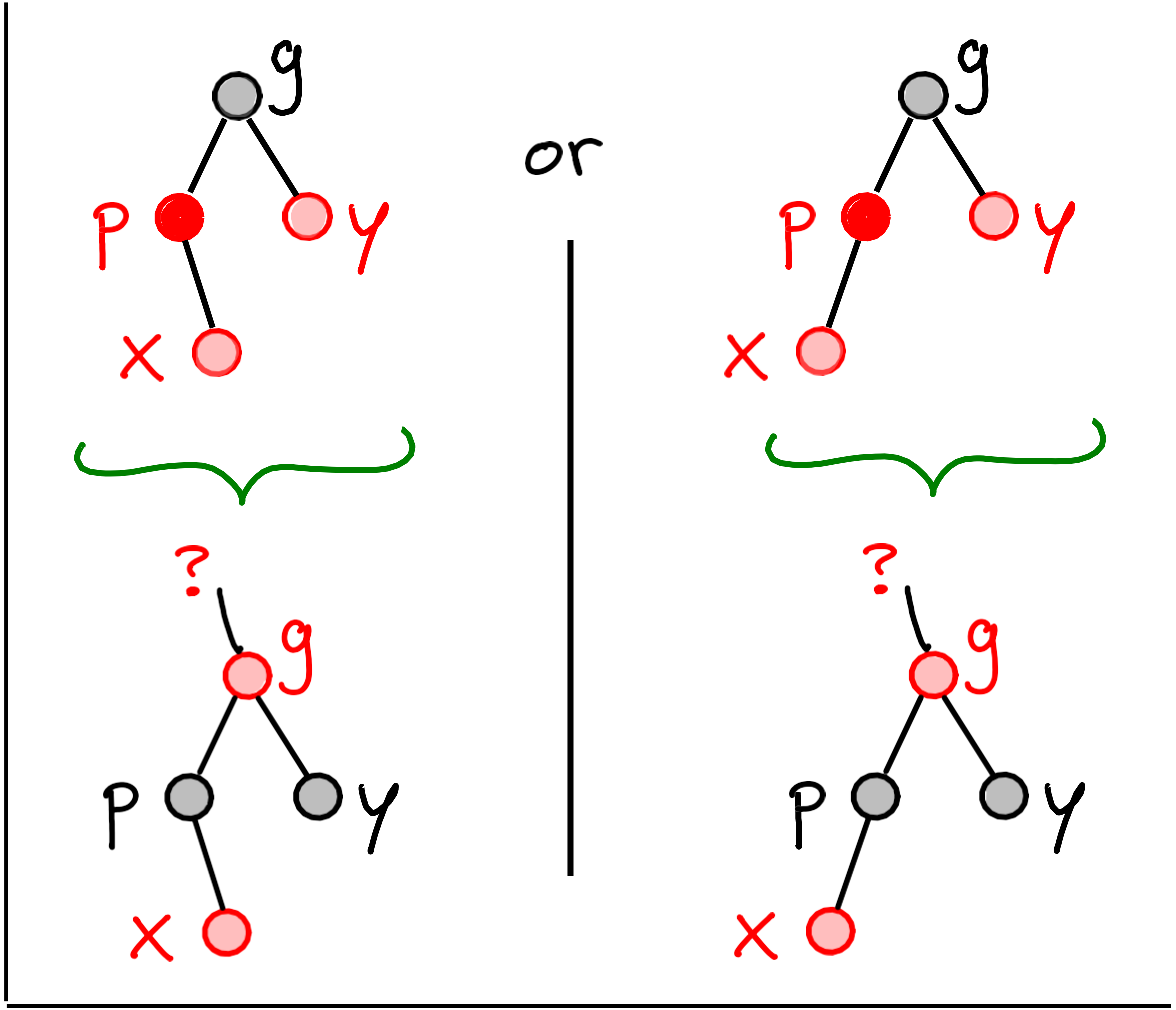


Case 1 : y is red

↳ Recolor p, g, y

- Preserve black-height invariant. (total B nodes on any path from global root down to leaves)
- Eliminate $\underset{x}{\overset{p}{|}}$ violation.
- Might cause violation $\underset{g}{\overset{?}{\updownarrow}}$

  otherwise DONE



→ fix upward → ? becomes p
g becomes x

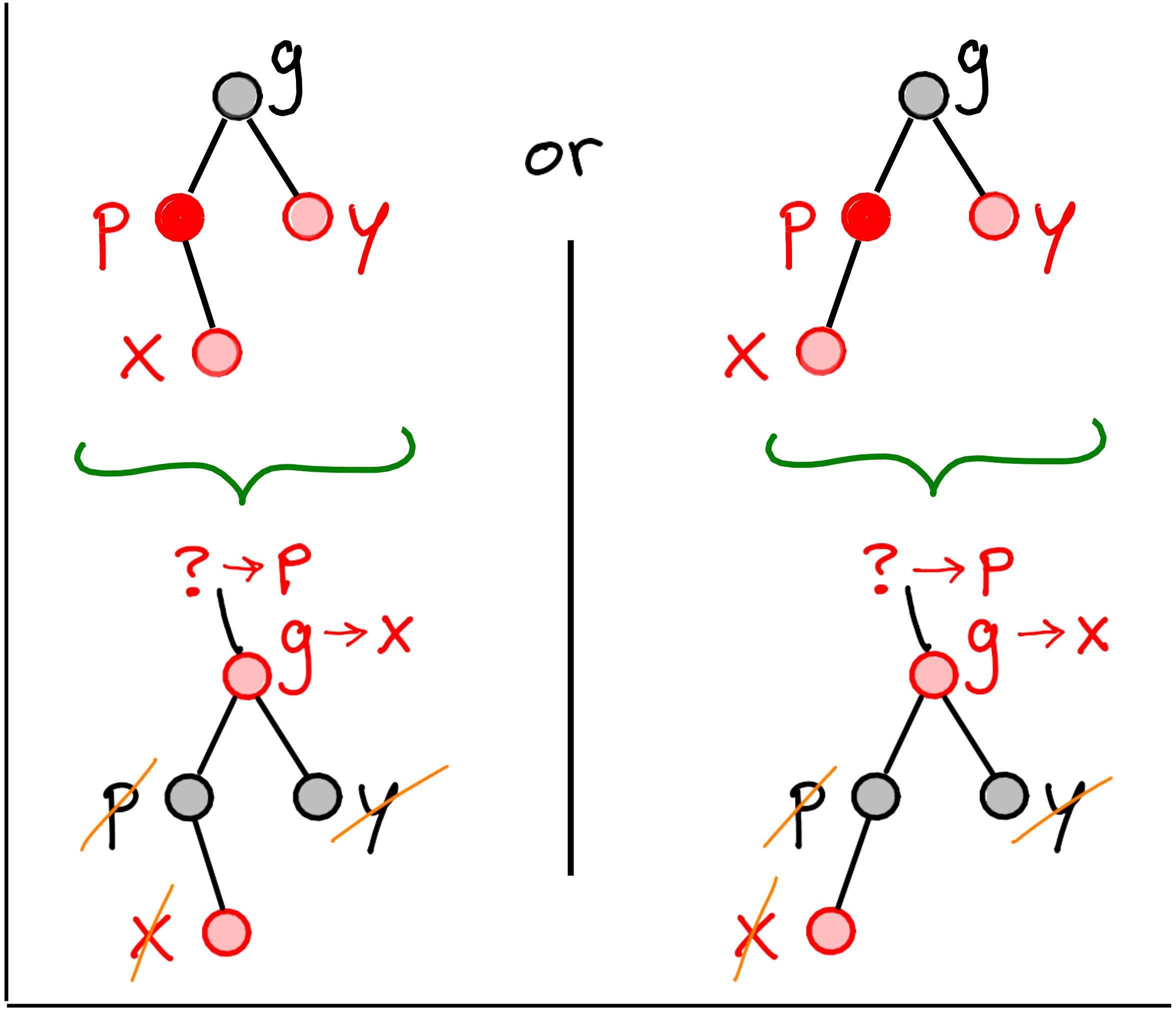
(Just changing labels)

Case 1 : y is red

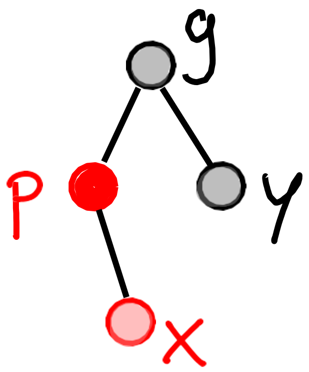
↳ Recolor p, g, y O(1)

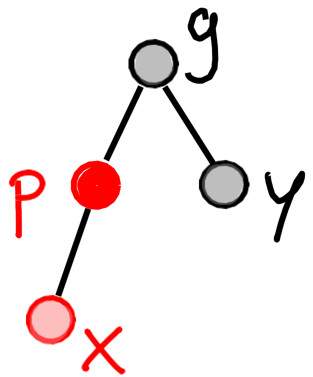
- Preserve black-height invariant.
  (total B nodes on any path from global root down to leaves)
- Eliminate $\frac{p}{x}$ violation.
- Might cause violation $\frac{?}{g}$
  otherwise DONE

↳ fix upward → ? becomes p, g becomes x → Invariants restored



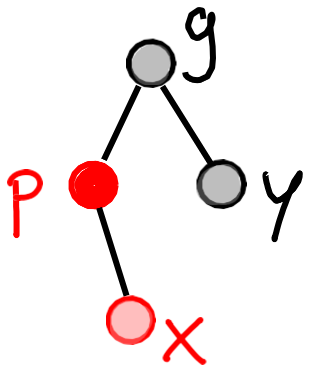
Cases 2 & 3: y is black



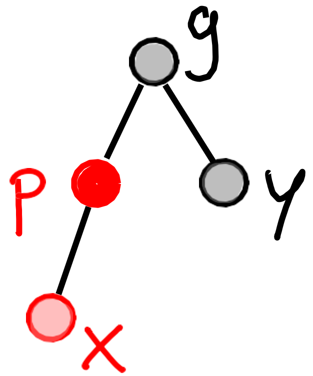


Cases 2 & 3 : y is black

Case 2

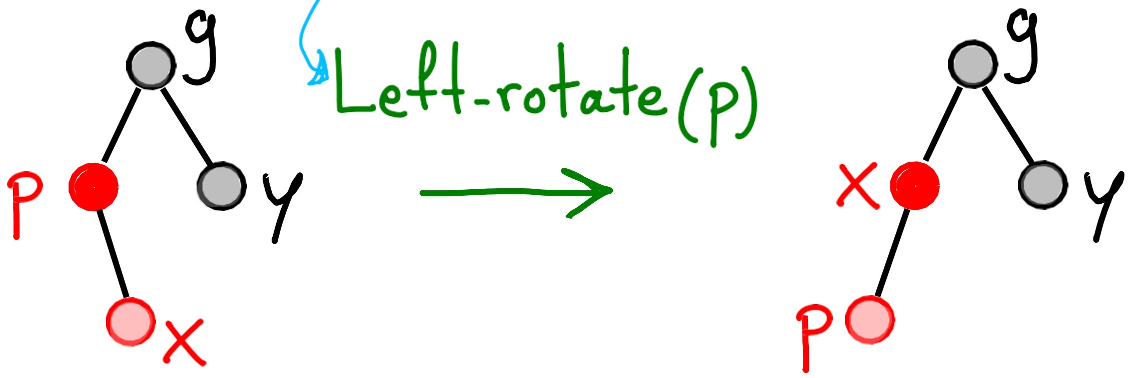


Case 3



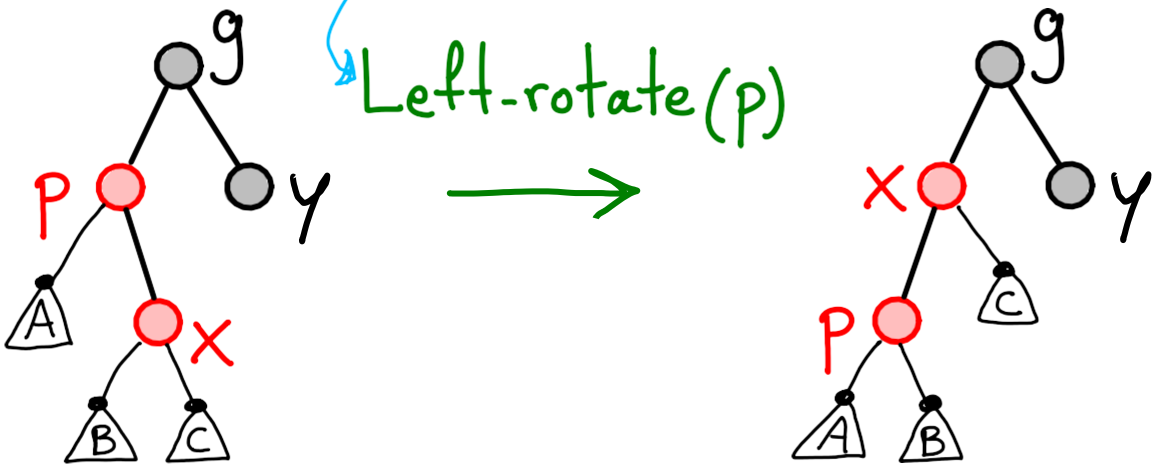
Cases 2 & 3: y is black

Case 2



Cases 2 & 3: y is black

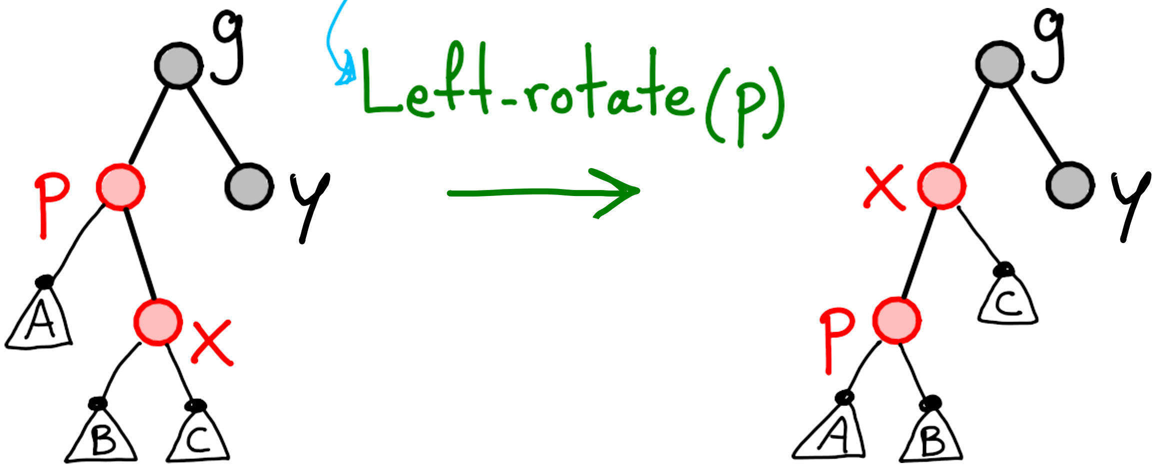
Case 2



Cases 2 & 3: y is black

- Preserve black-height invariant

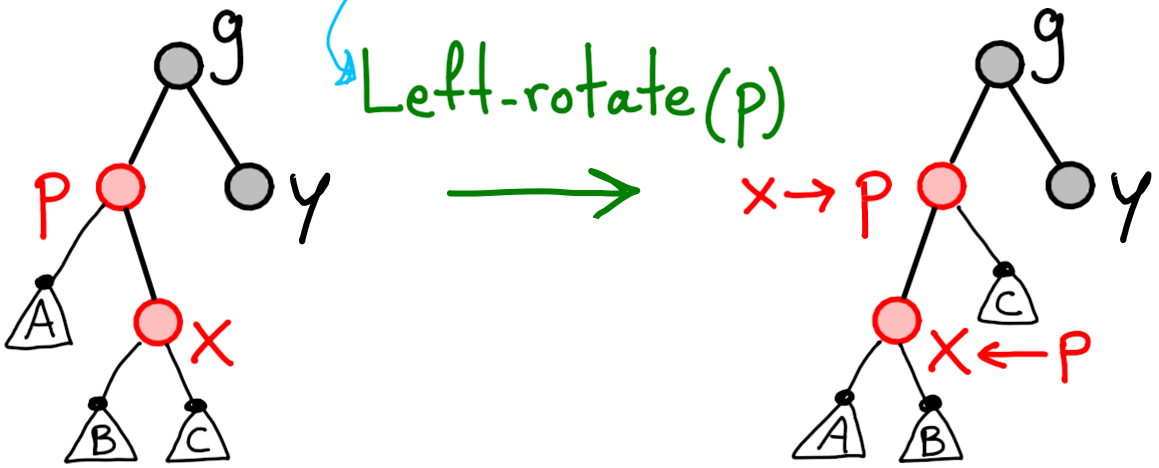
Case 2



Cases 2 & 3: y is black

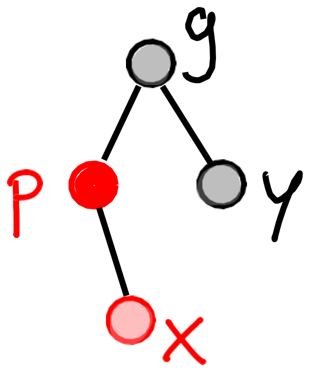
- Preserve black-height invariant
- Swap labels $x \leftrightarrow p$
- No new violation

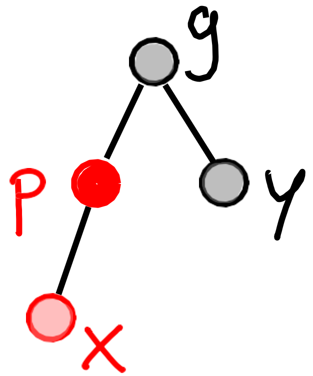
Case 2



Cases 2 & 3: y is black

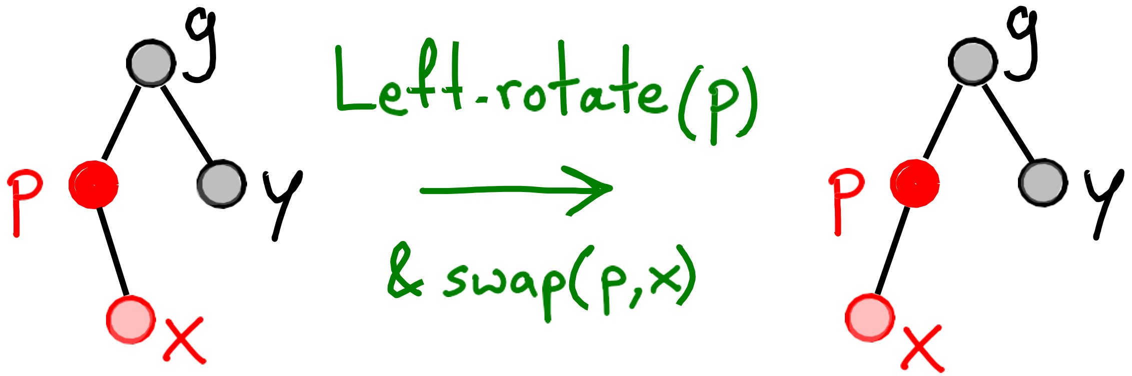
Case 2 → Case 3



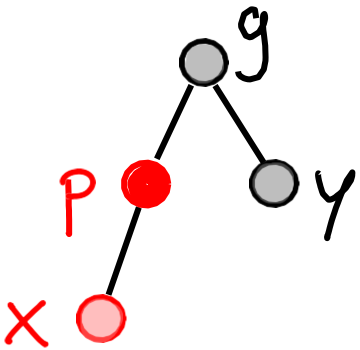


Cases 2 & 3: y is black

Case 2 Case 3

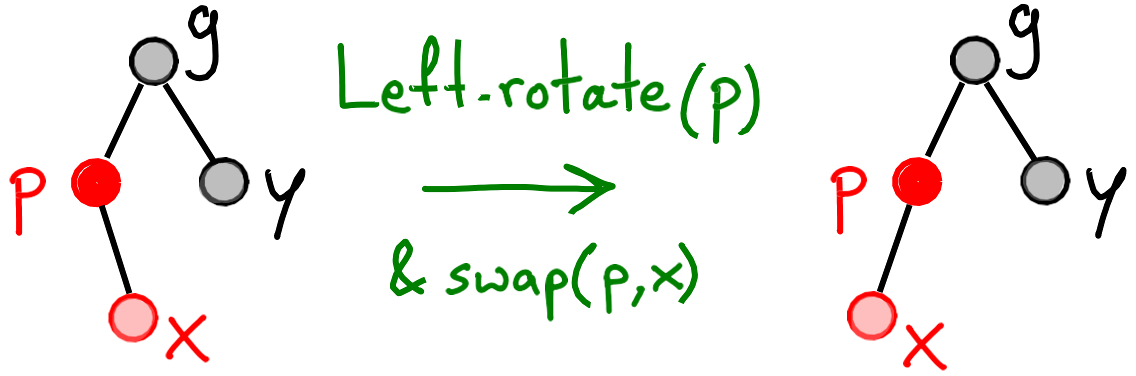


Handling Case 3:

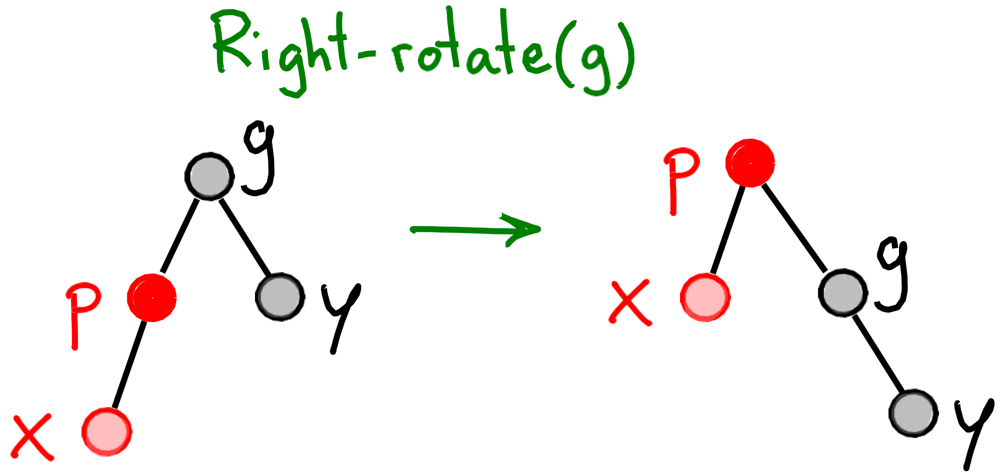


Cases 2 & 3: y is black

Case 2 / Case 3

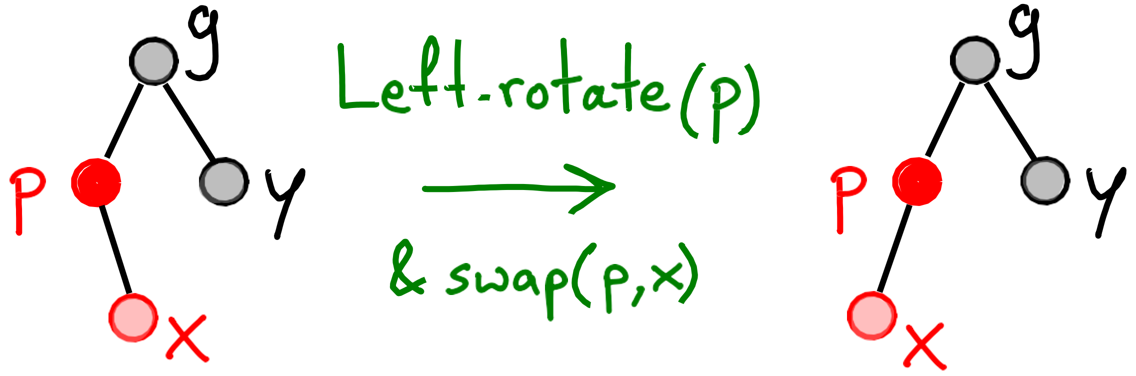


Handling Case 3:

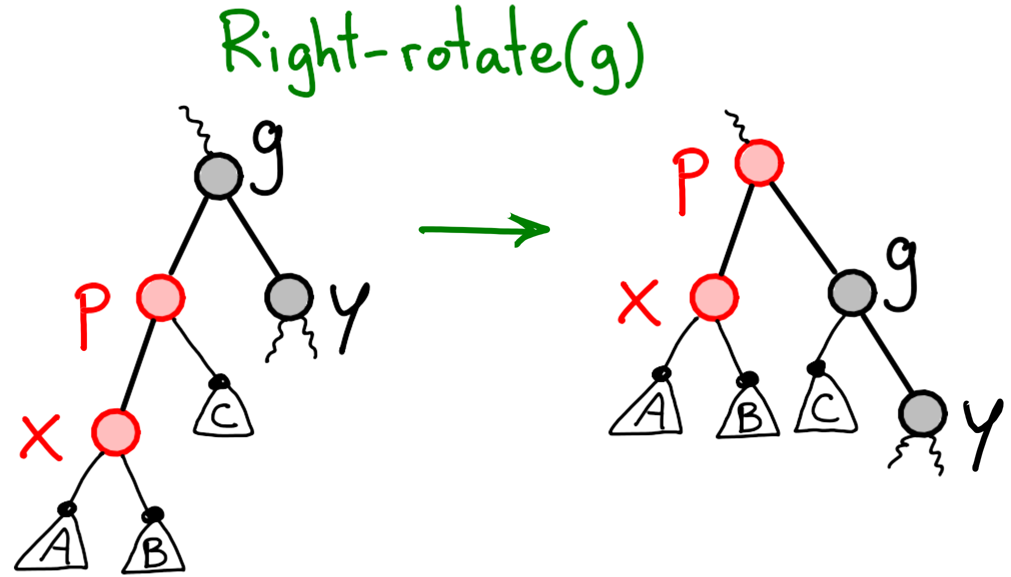


Cases 2 & 3: y is black

Case 2 Case 3



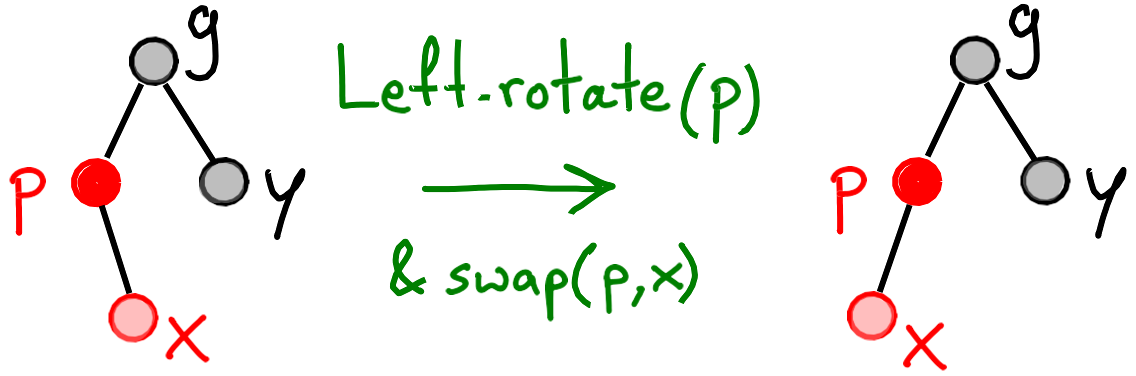
Handling Case 3:



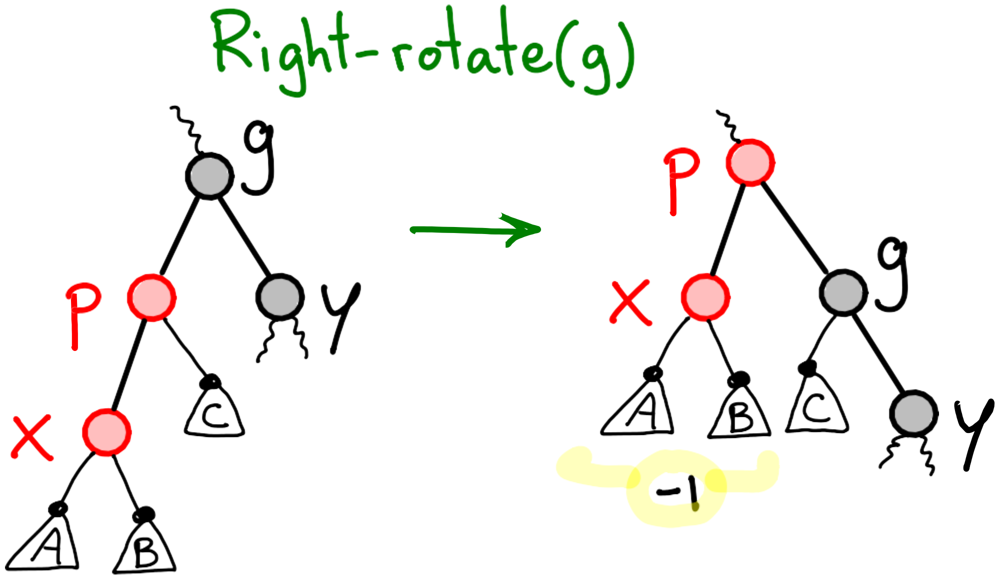
Cases 2 & 3: y is black

- ~~Preserve~~ black-height invariant.

Case 2 / Case 3



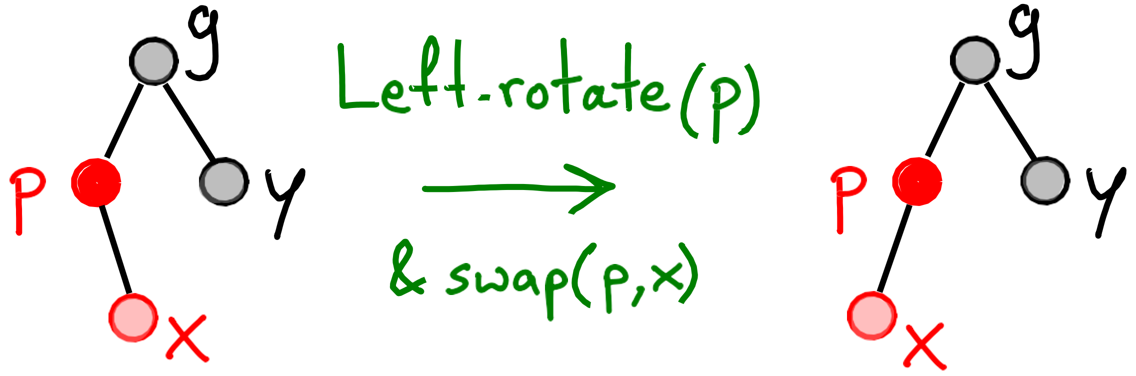
Handling Case 3:



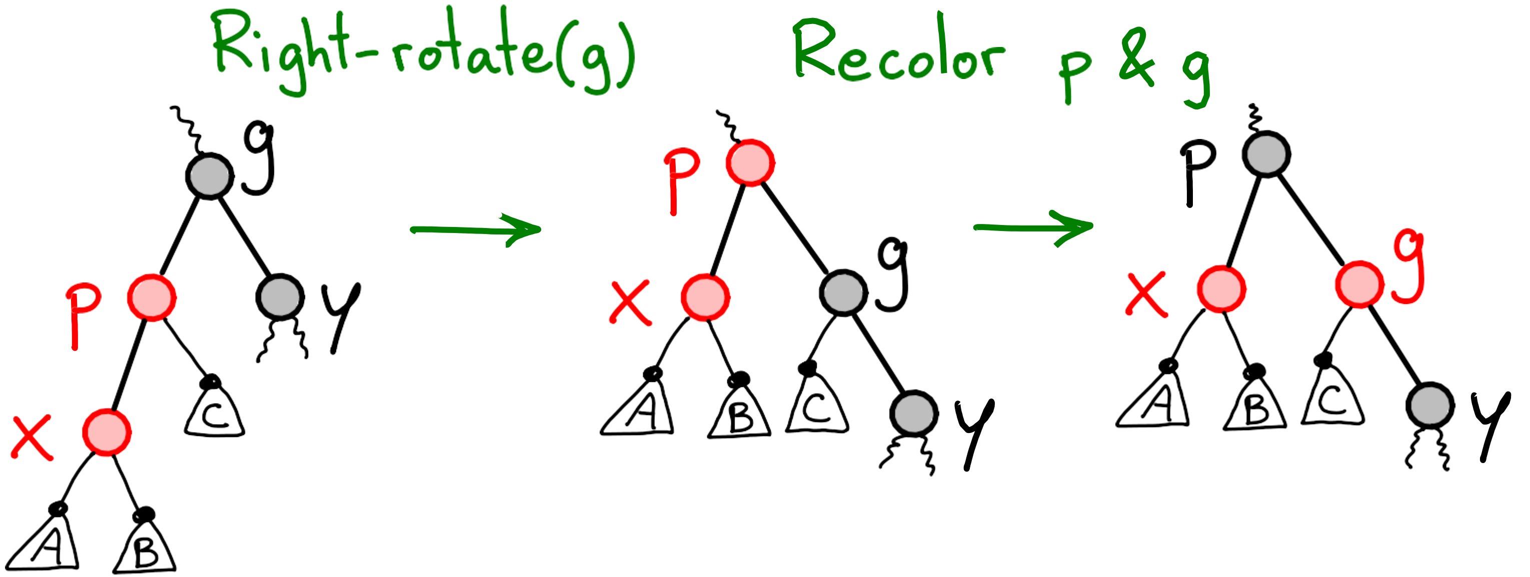
Cases 2 & 3: y is black

- Preserve black-height invariant ✓

Case 2 Case 3

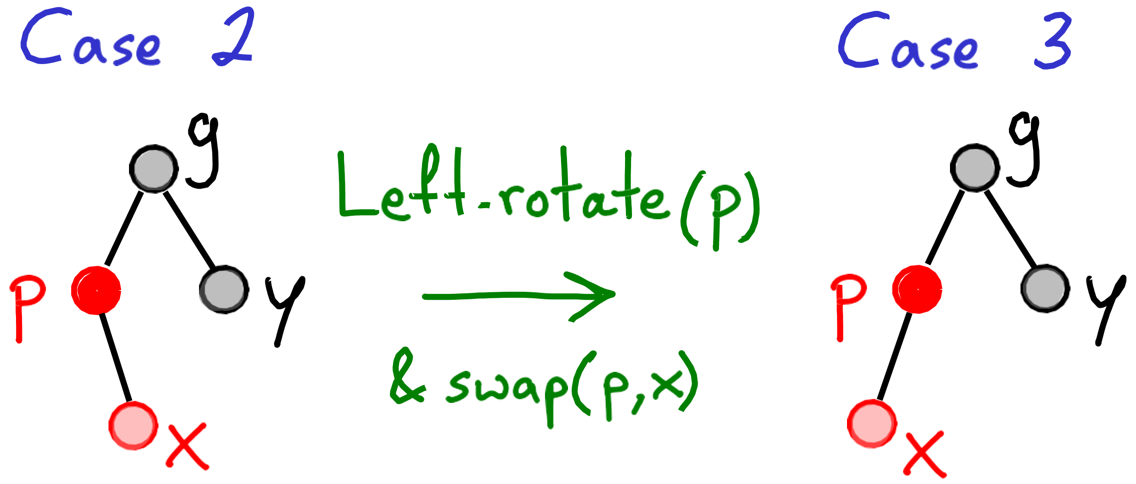


Handling Case 3:

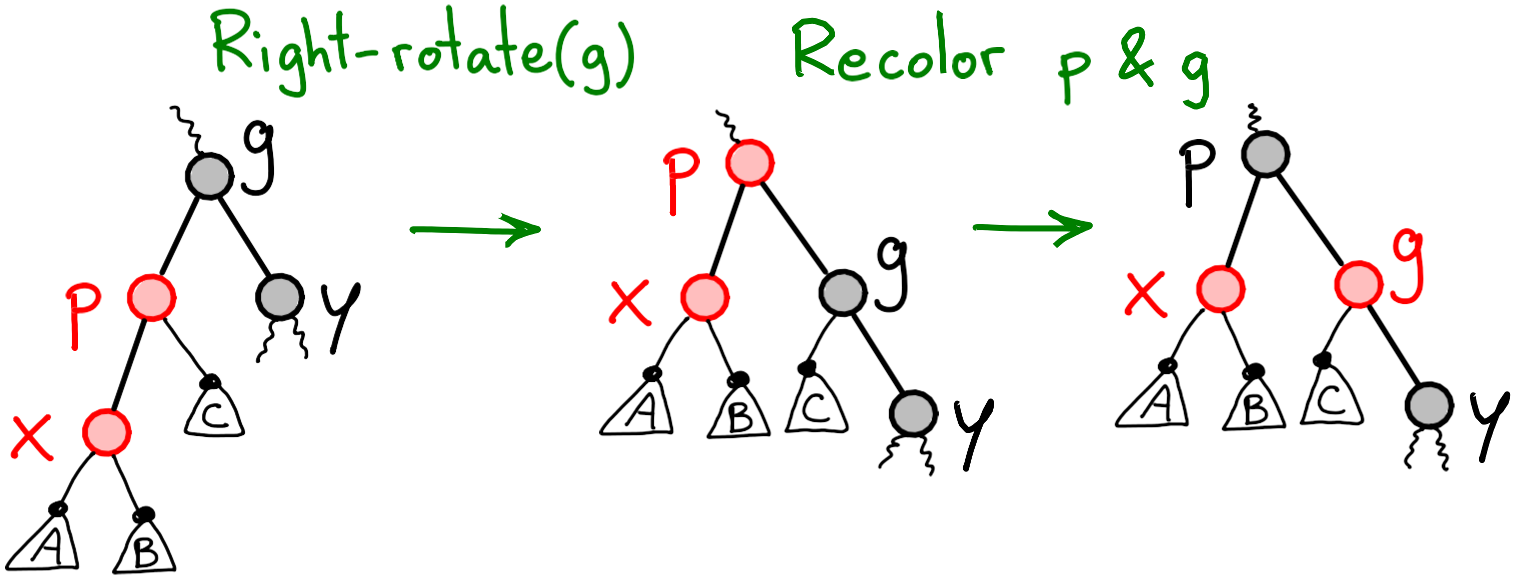


Cases 2 & 3: y is black

- Preserve black-height invariant.
- Eliminate p–x violation in case 3.

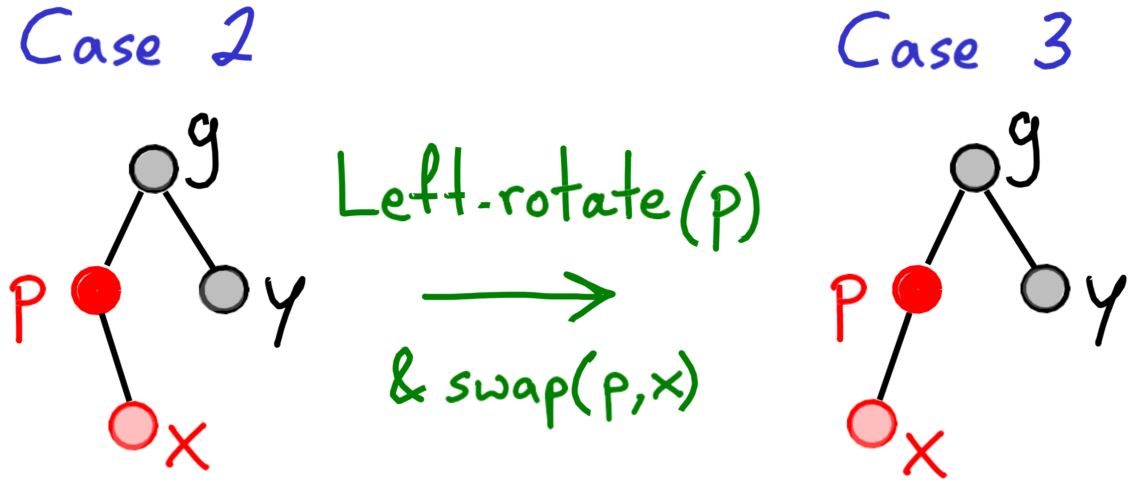


Handling Case 3:

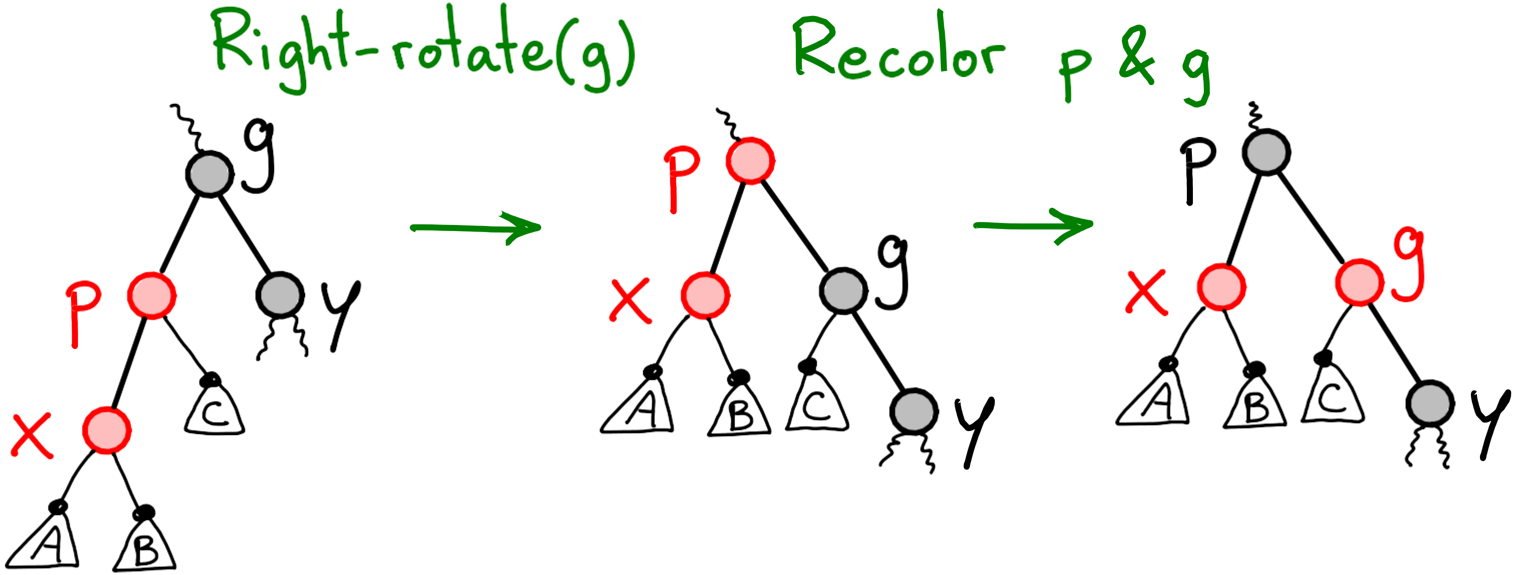


Cases 2 & 3: y is black

- Preserve black-height invariant.
- Eliminate p–x violation in case 3.
- No new violation

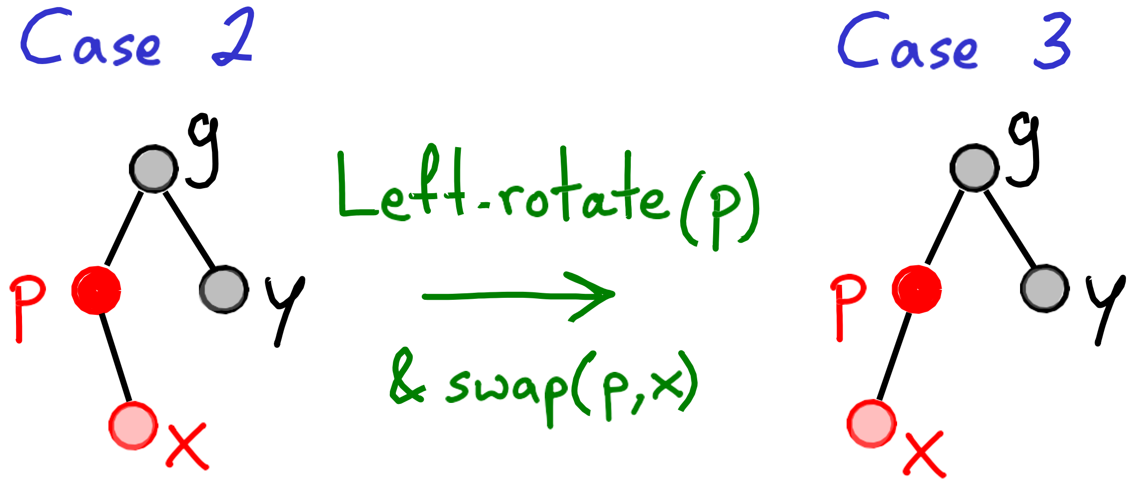


Handling Case 3:

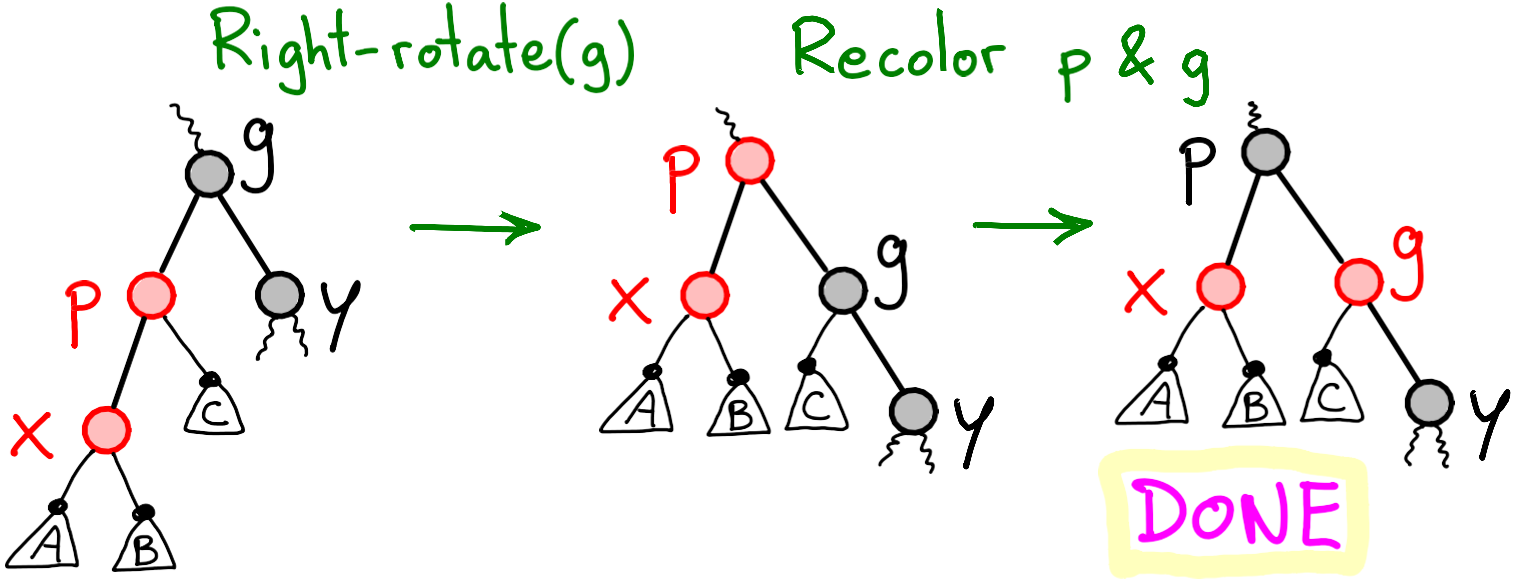


Cases 2 & 3: y is black

- Preserve black-height invariant.
- Eliminate p–x violation in case 3.
- No new violation



Handling Case 3:



Each case takes O(1) time

All together O(logn) time

Each case takes O(1) time

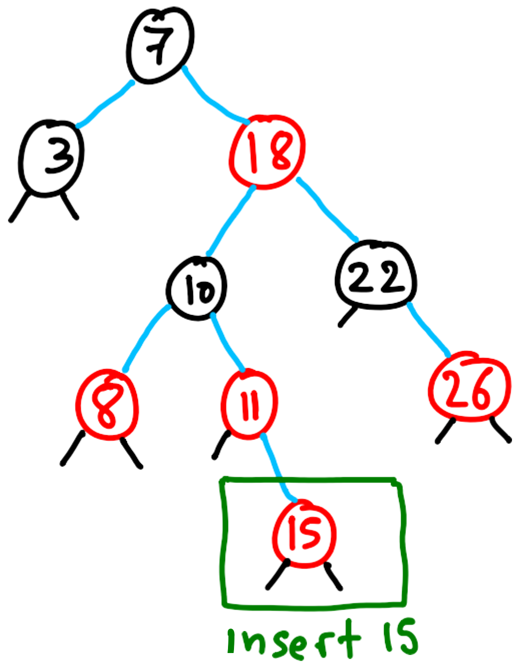
All together O(logn) time

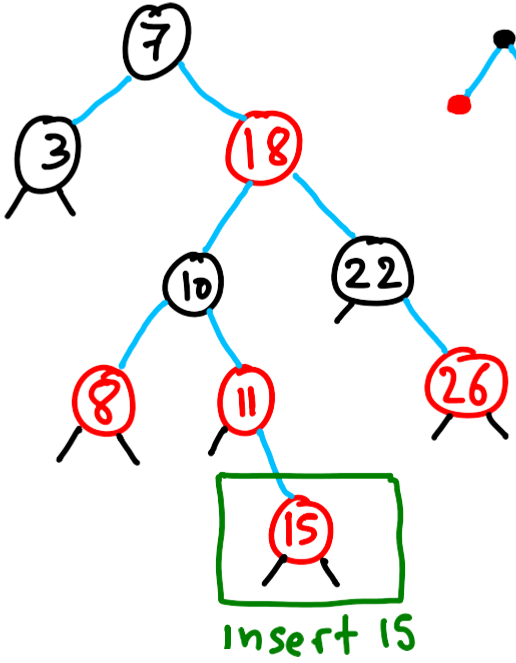
Possibly lots of recoloring (Case 1)

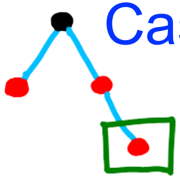
followed by Case 3   OR   Case 2 —> 3

(1 or 2 rotations, total)

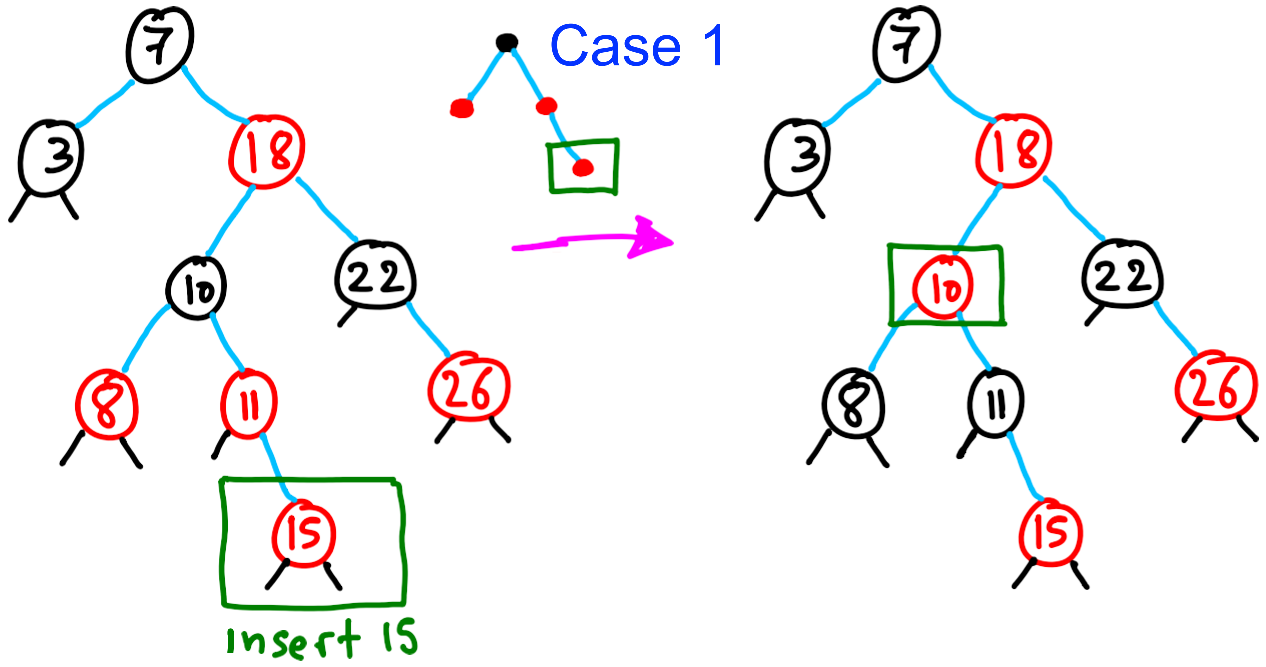
* useful property

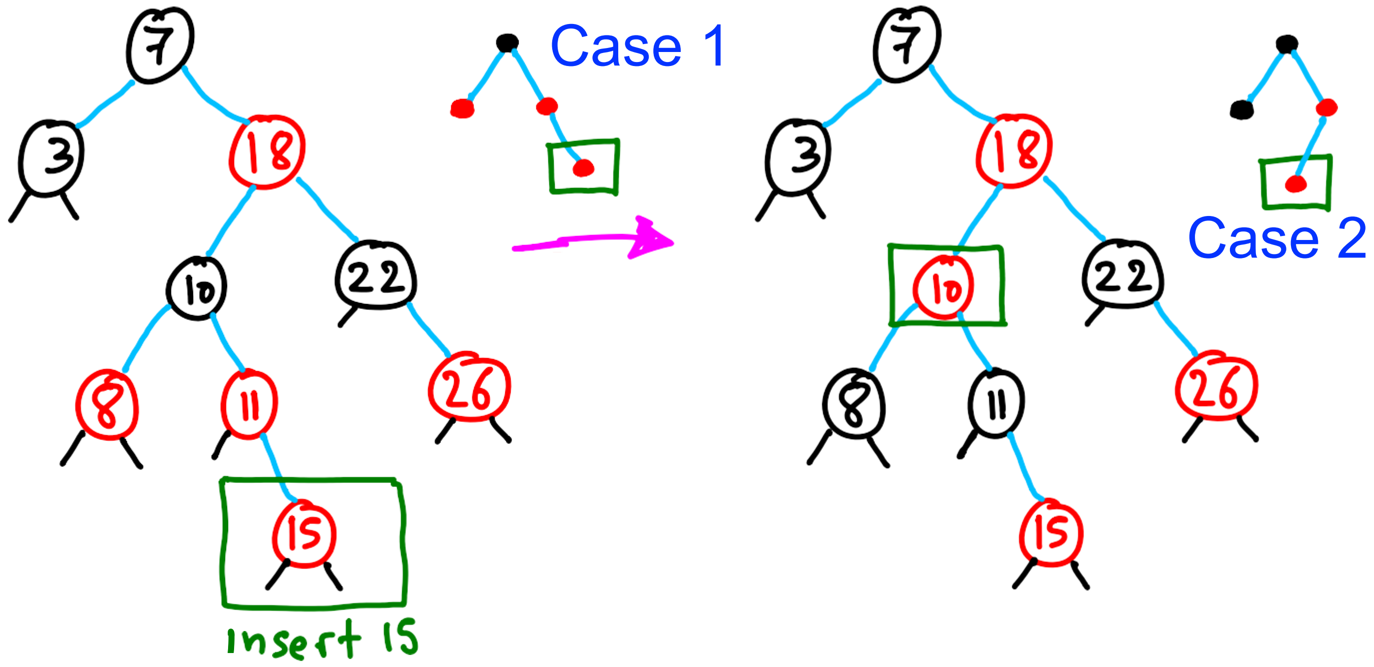

7
3
18
10
22
8
11
26
15
insert 15

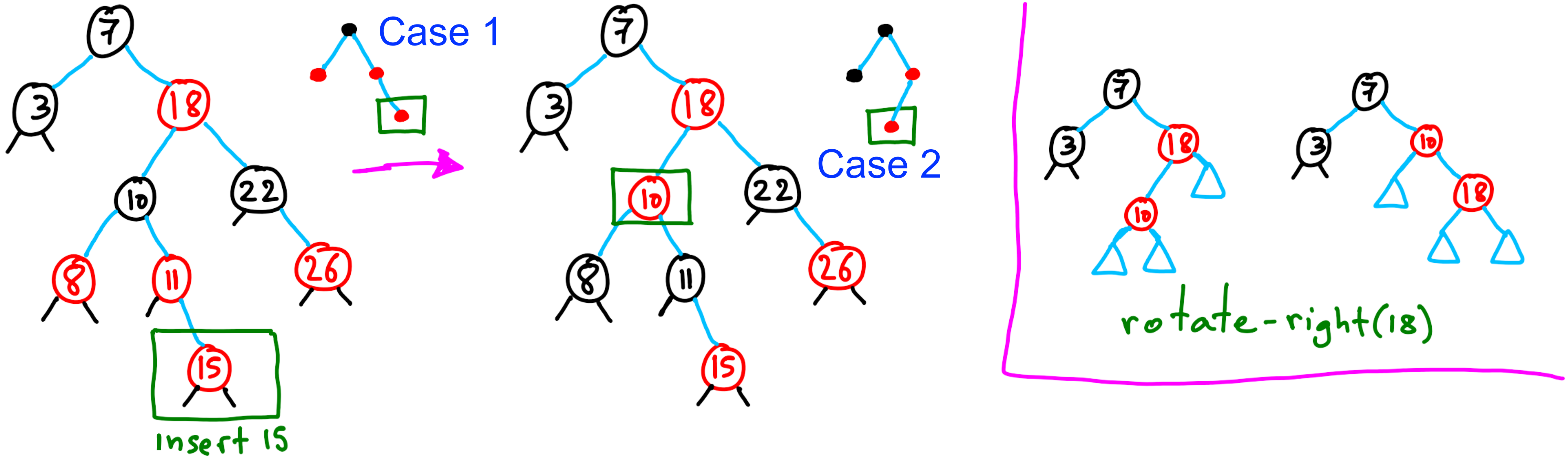


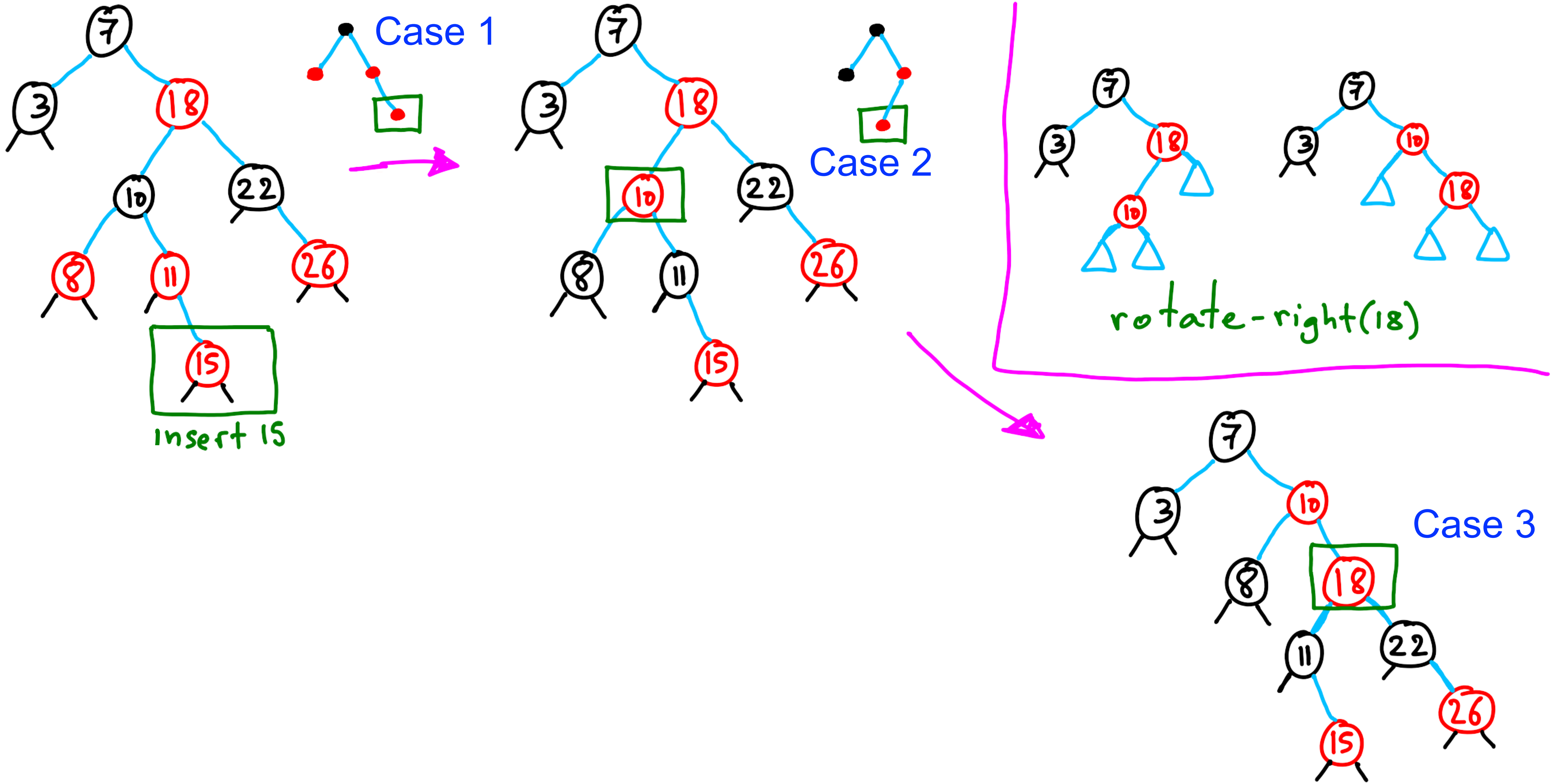


Case 1

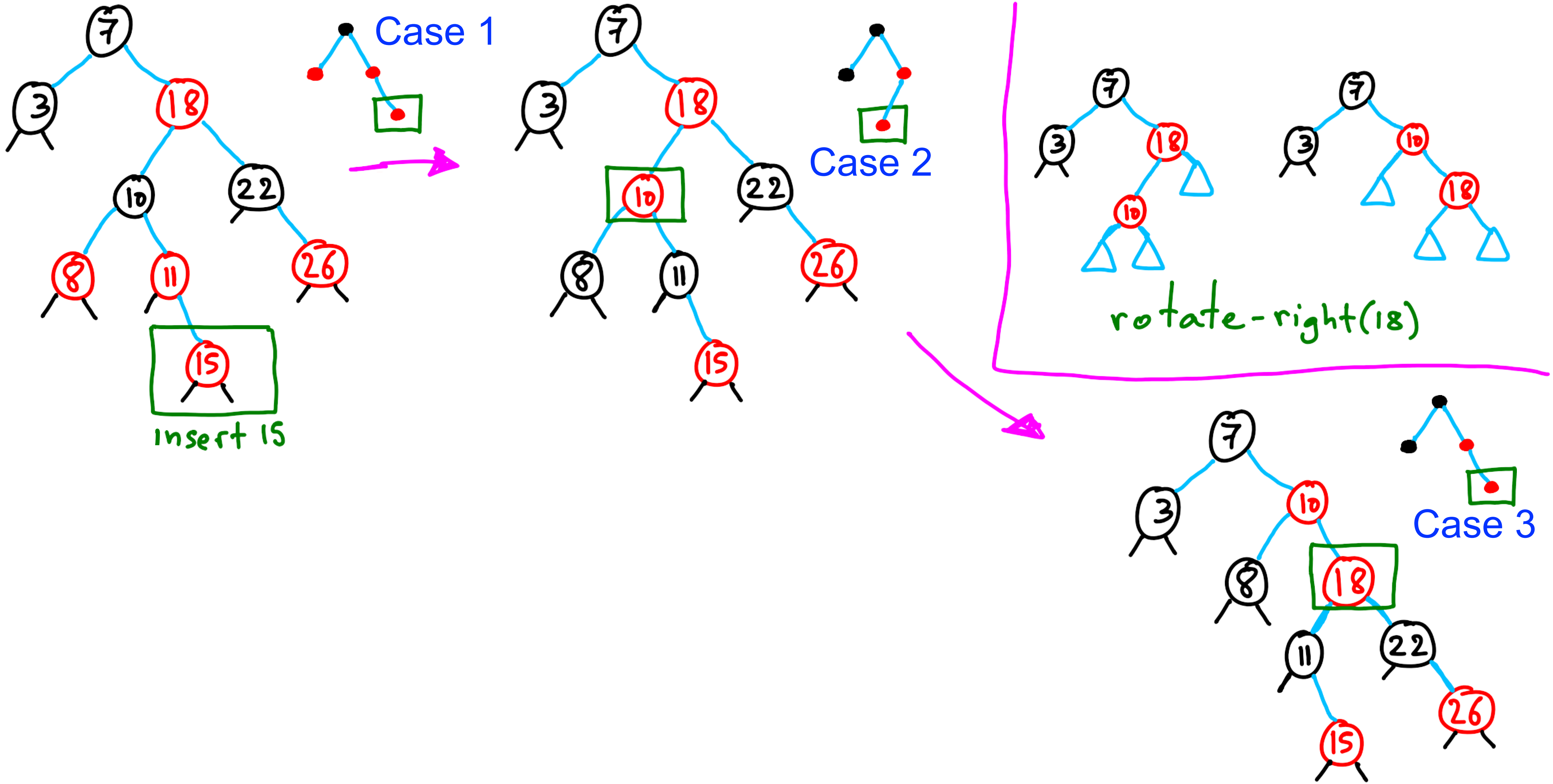

Case 1
7
3
18
10
22
8
11
26
15
insert 15
7
3
18
10
22
8
11
26
15

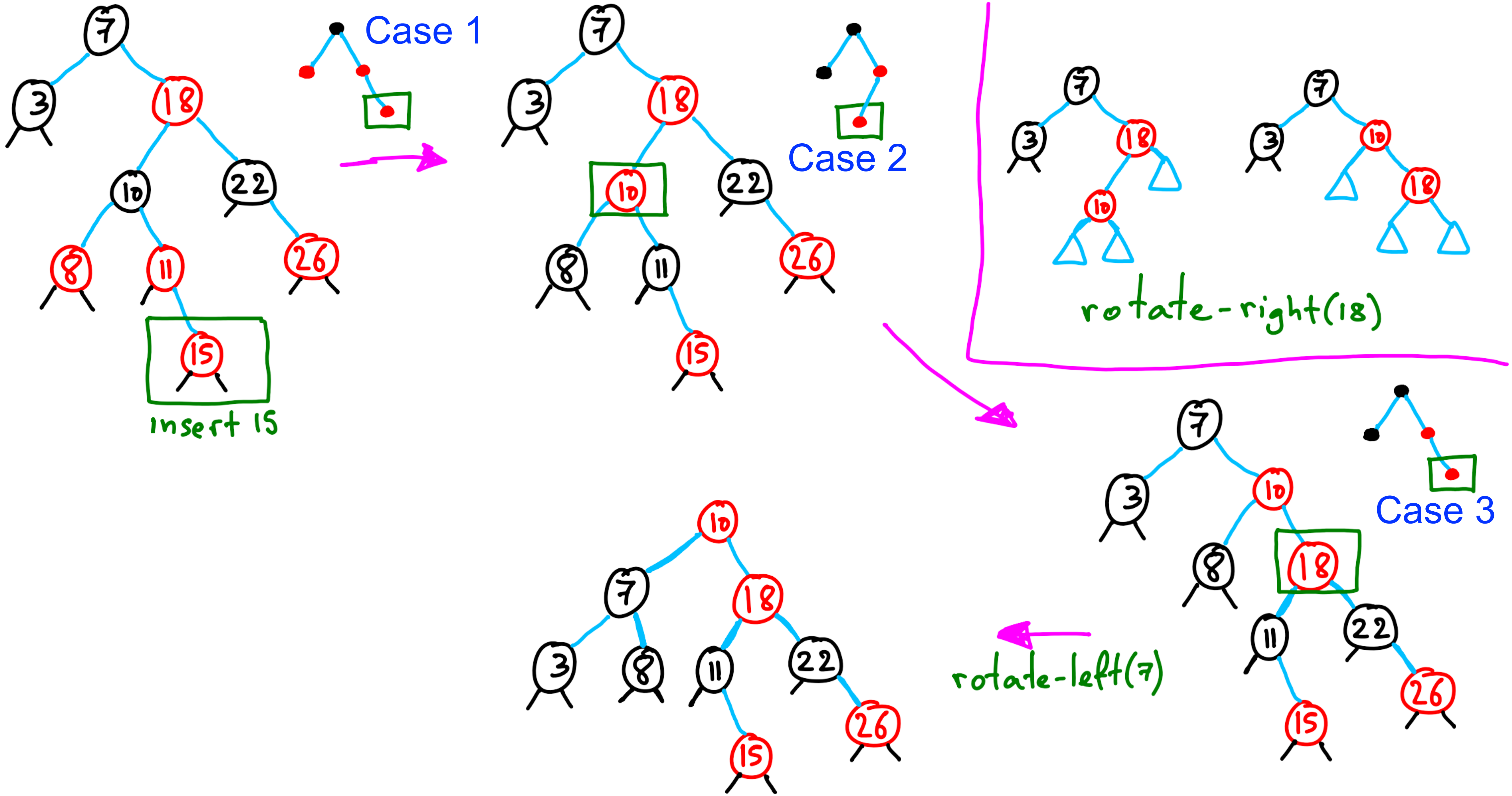

7
3
18
10
22
8
11
26
15
insert 15
Case 1
Case 2

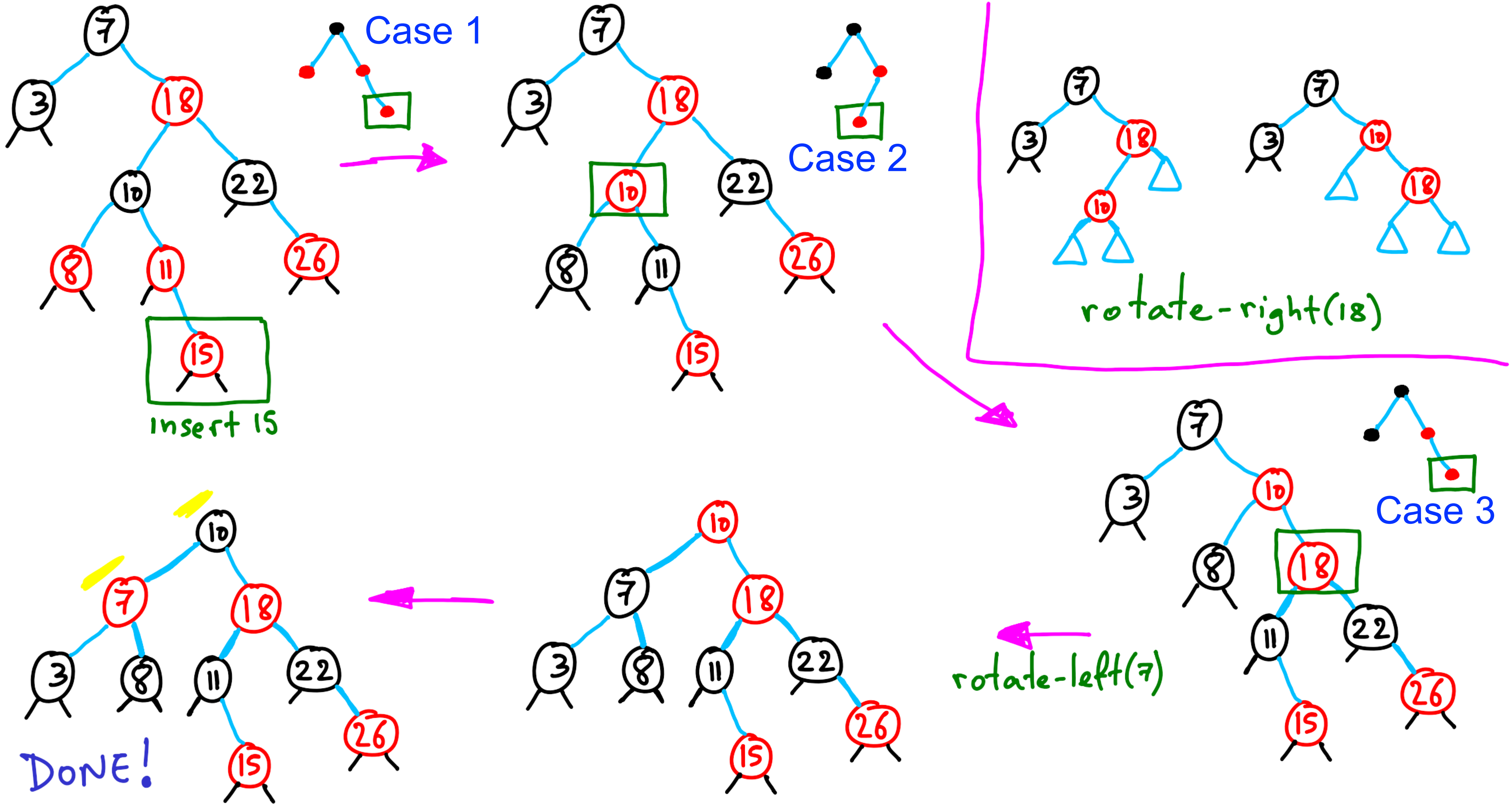

Case 1
Case 2
insert 15
rotate-right(18)



Case 1
Case 2
Case 3
insert 15
rotate-right(18)


Case 1
insert 15
Case 2
rotate-right(18)
Case 3


Case 1
insert 15
Case 2
rotate-right(18)
Case 3
rotate-left(7)


Case 1
insert 15
Case 2
rotate-right(18)
Case 3
rotate-left(7)
DONE!