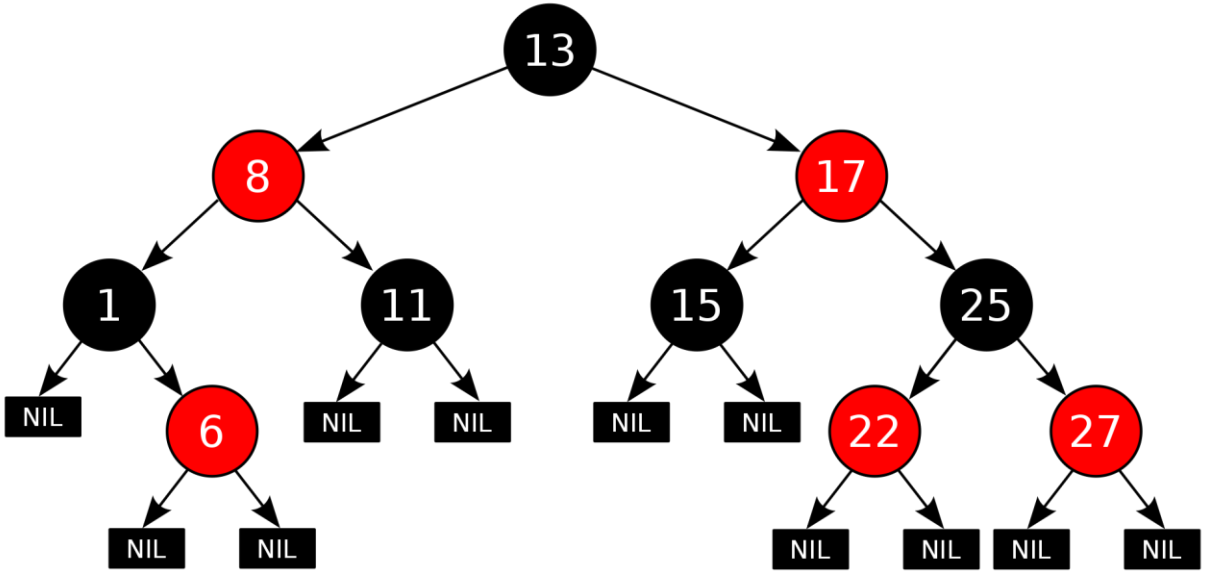
Definition of a **Red-Black** Tree

- A red-black tree is a binary search tree
- Each node is either red or black.
- The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa, this rule has little effect on analysis.
- All leaves (NIL) are black.
- If a node is red, then both its children are black.
- Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes.

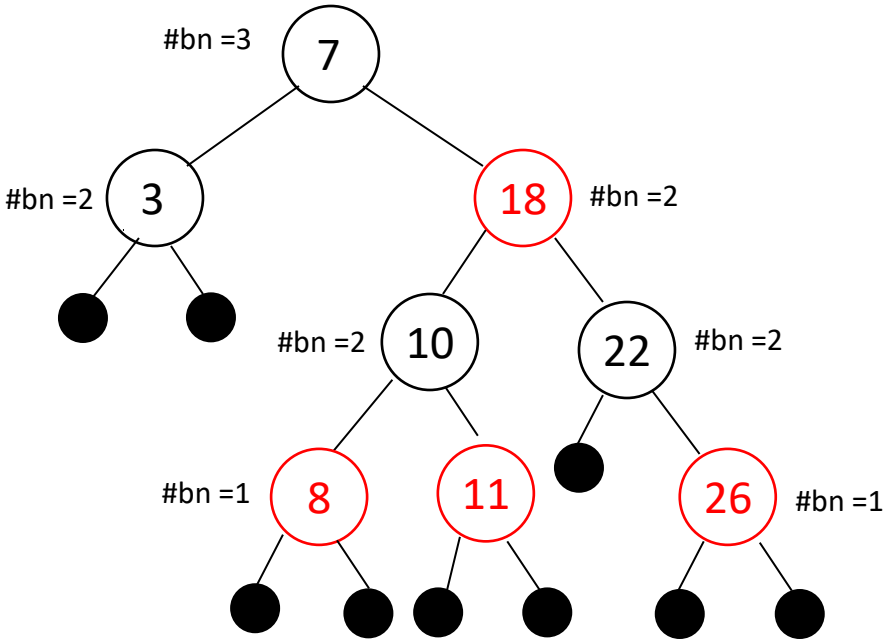


A **red-black** tree is balanced because it depth $O(\log n)$

Depth-balanced

We have to maintain the following balanced property:

Each path from the root to a leaf contains the same number of black nodes



Insertion and Deletion

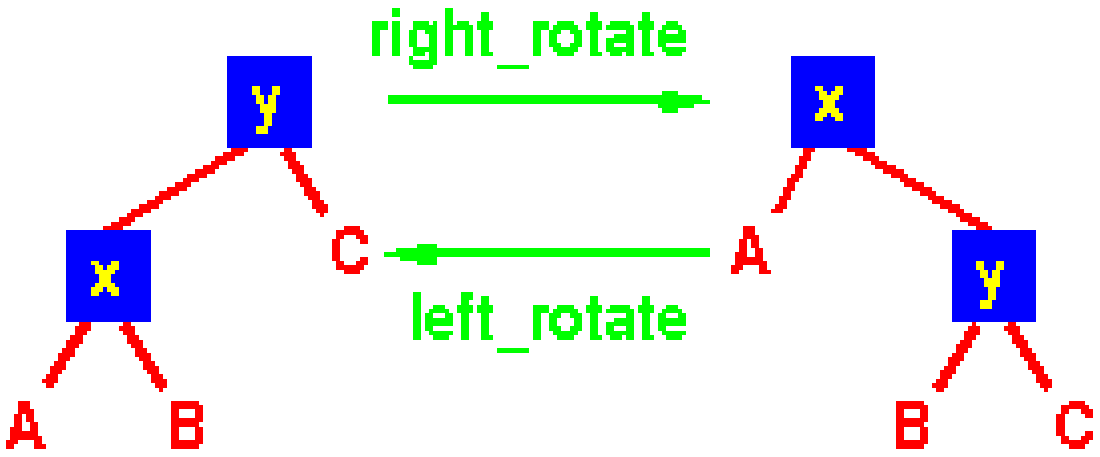
A red-black tree is a binary search tree.

The operations of insertion and deletion are done in two steps.

1. Do the usual binary search tree insertion (or deletion)
2. Perform a sequence of rotation operations starting from the node where insertion (or deletion) occurred working upwards towards the root until the red-black tree property is restored.

Rotation

A rotation is a local operation in a search tree that preserves *in-order* traversal key ordering, so that it preserve the binary search tree property.



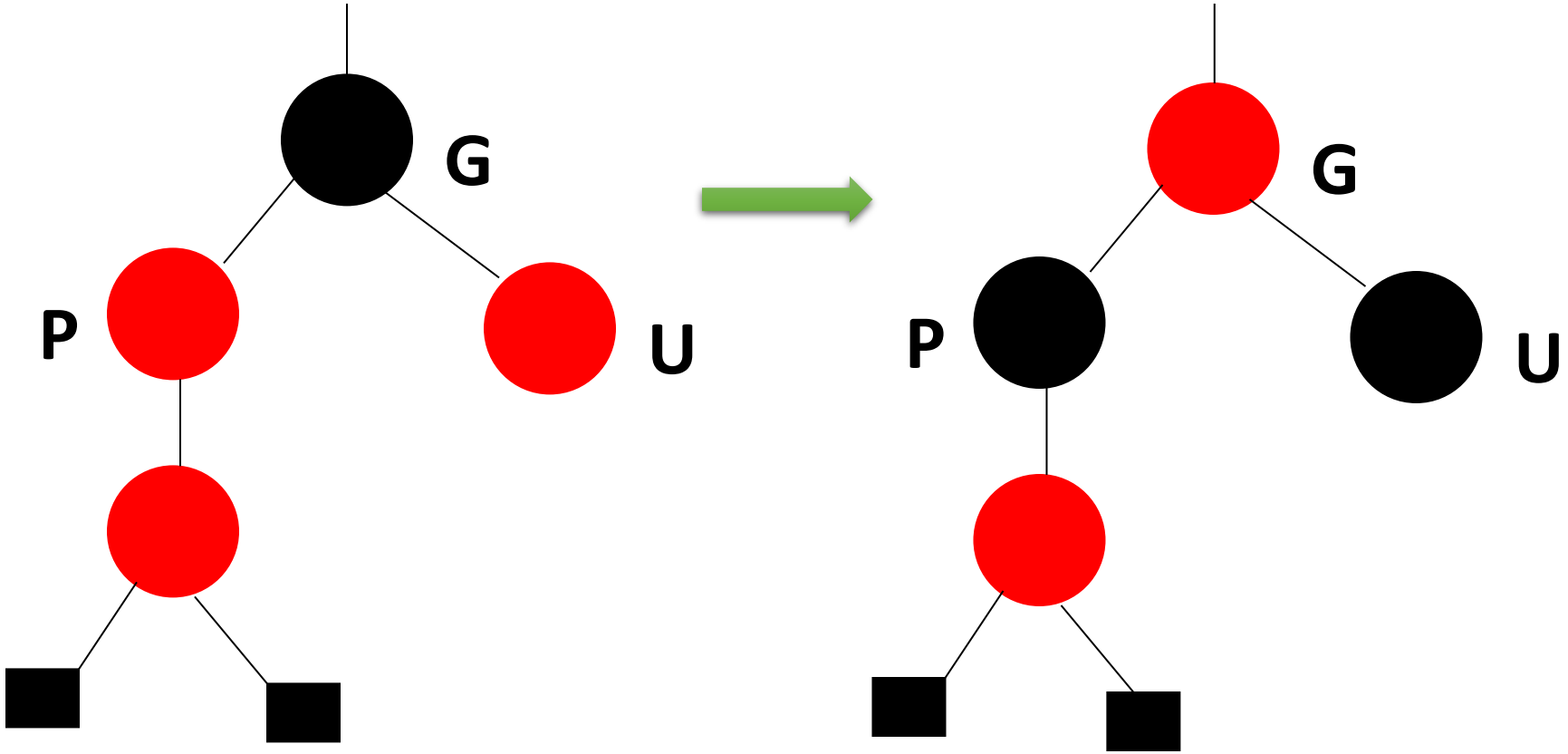
Insertion into Red-Black Trees

The idea for insertion in a red-black tree is to

- top-down insert like in a binary search tree,
- color the inserted node red,
- then reestablish the color properties through a recoloring and rotations.

After finishing the top-down insert we have 3 cases for when performing bottom up recoloring and rotations.

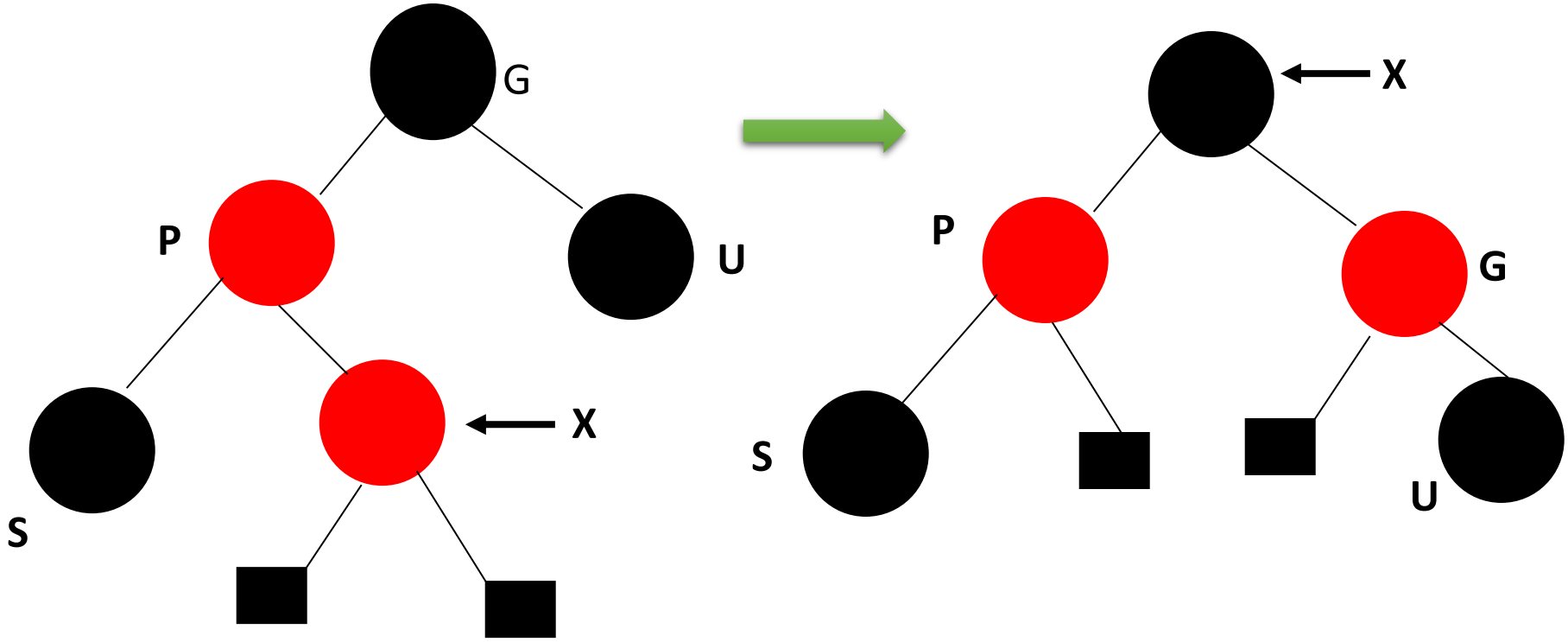
Case 1: Recolor (Uncle is Red)



Case 2:

Double Rotate: X around P then X around G.

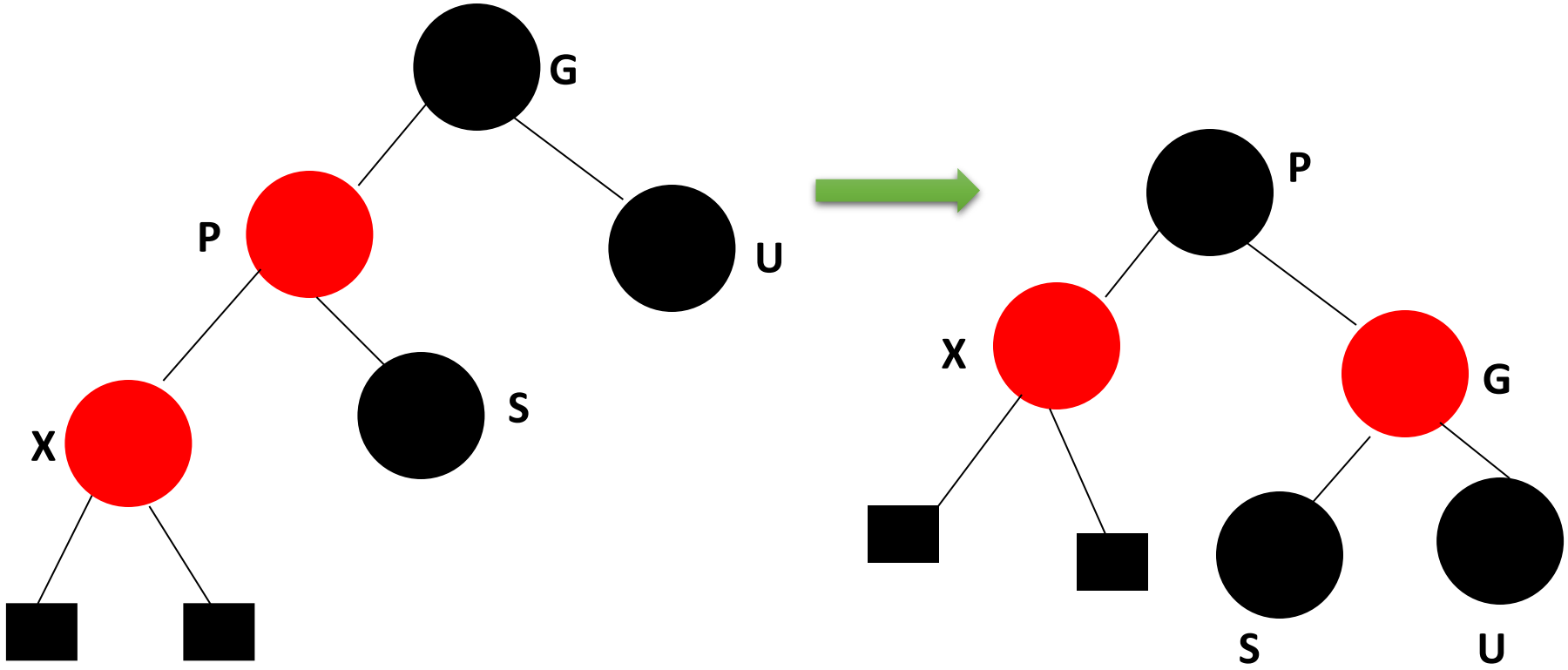
Recolor G and X



Case 3:

Single Rotate P around G

Recolor P and G



Analysis of Insertion

- A red-black tree has depth $O(\log n)$. Thus, search for insertion location takes $O(\log n)$
- Rotations and recolorings takes taking $O(1)$ time.
- Thus, an insertion in a red-black tree takes $O(\log n)$ time.

Deletion

First apply the rules for BST (Binary Search Tree) deletion

1. If a node to be deleted is a leaf, just delete it.
2. If a node to be deleted has just one child, replace it with that child
3. If a node to be deleted has two children, replace the value of the node by inorder predecessor's value, then delete the inorder predecessor by applying either 1. or 2. (could also use inorder successor).

Note that the node to be deleted is either a **leaf node or a node with one child.**

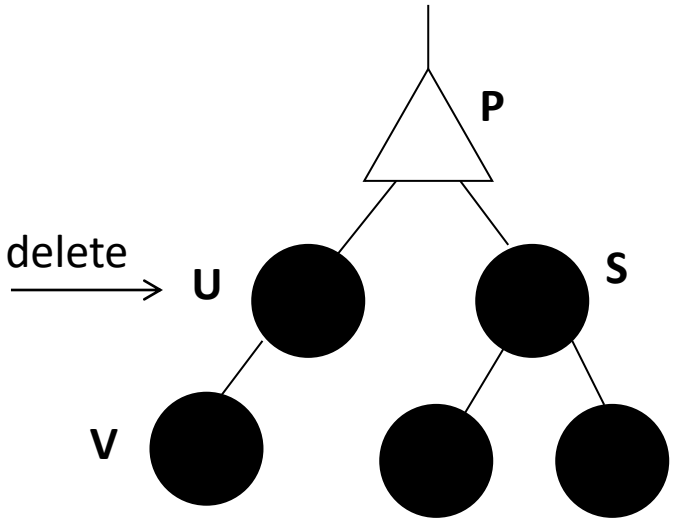
1. If the delete node is red?

   Not a problem – no RB properties violated

2. If the deleted node is black?

   If the node is not the root, deleting it will change the black-depth along some path

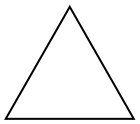
We need to consider several cases. We illustrate of few of these cases.



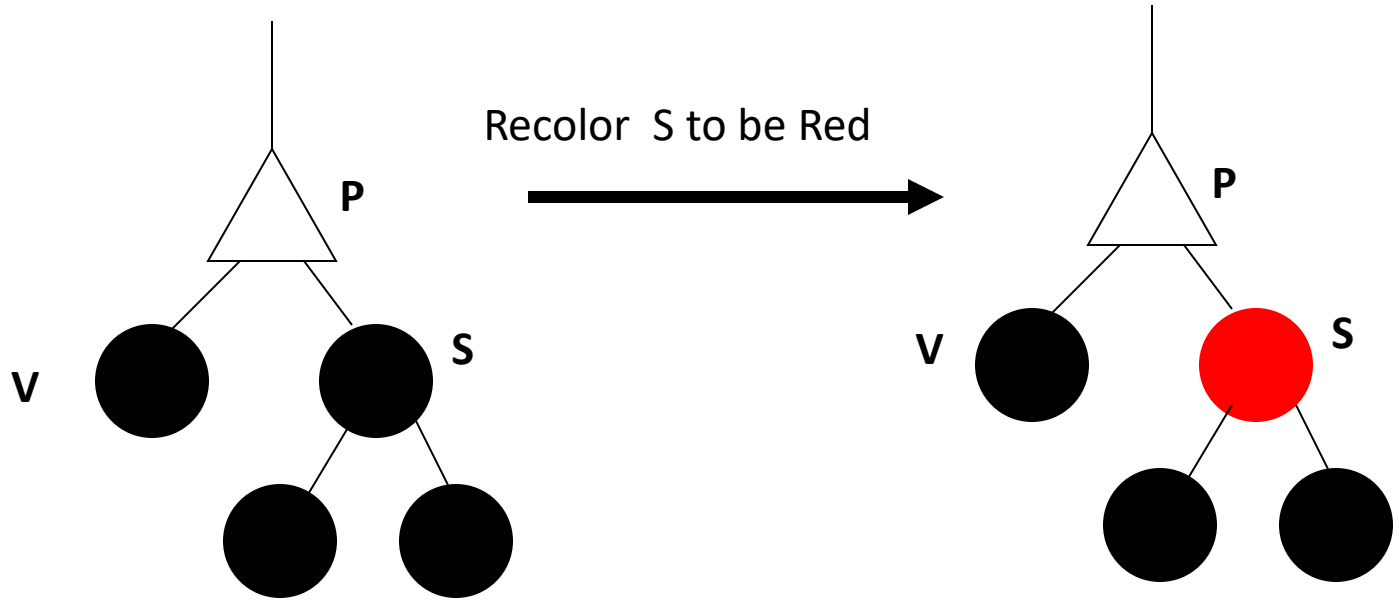
Case A:

- U's sibling S is black and has two black children.

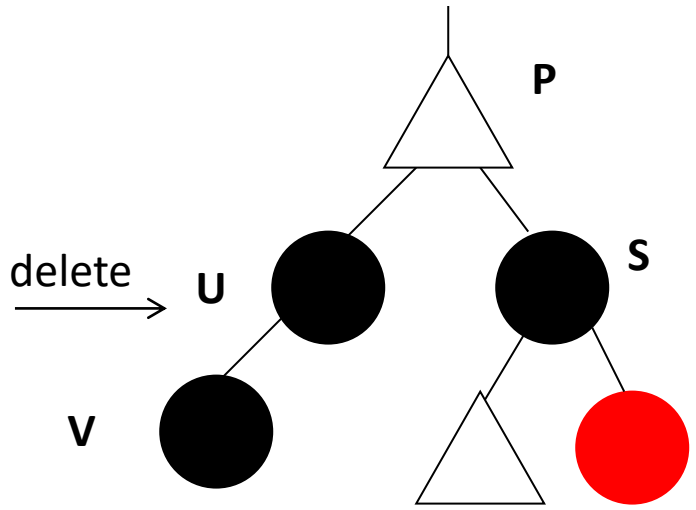
  Recolor S to be Red



Red or Black and don't care



The red-black tree property is restored locally for the tree rooted at the *P*. But, We may need to recurse where *P* plays the role of *V*, possibly until we get to the root.



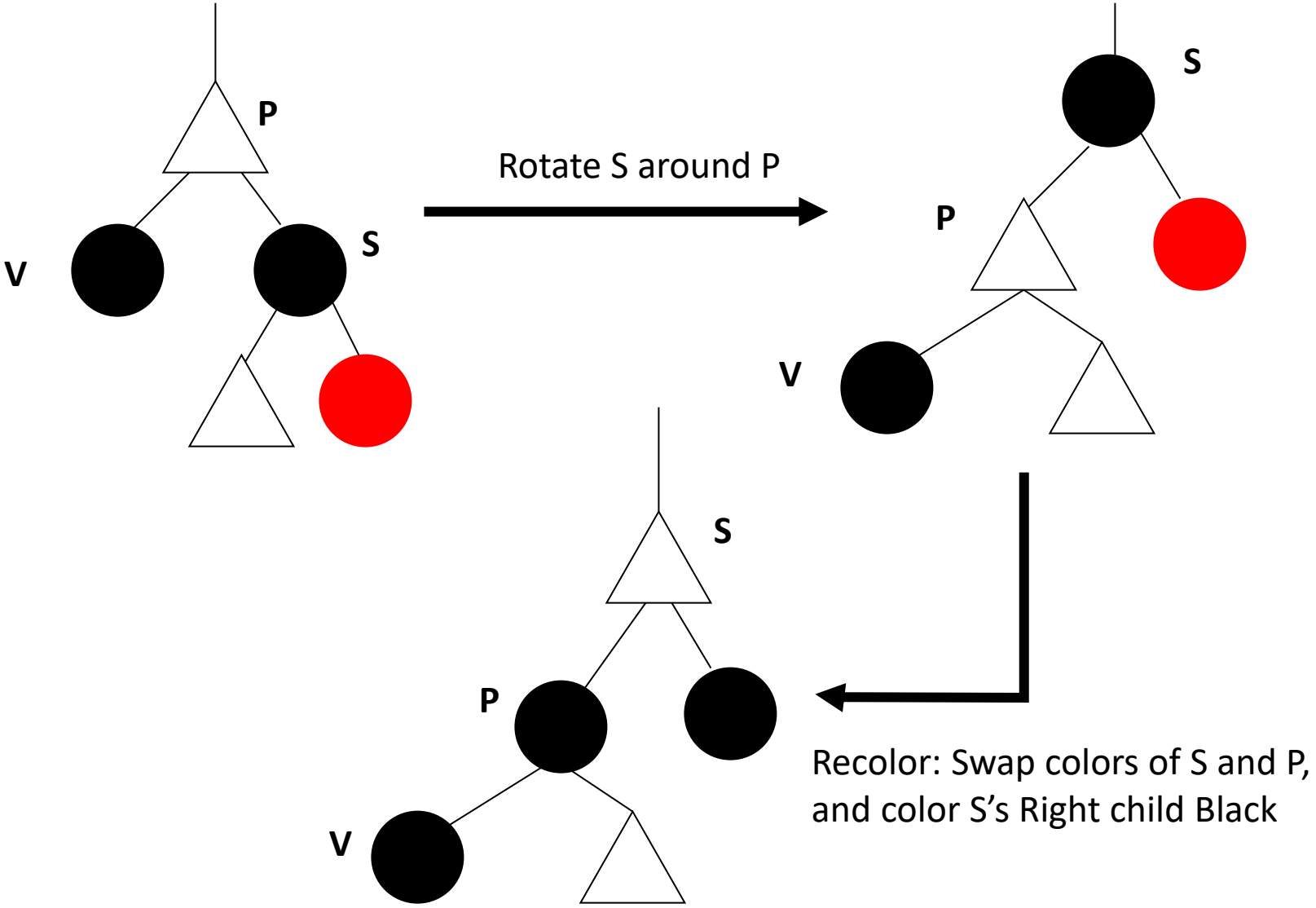
Case B:

- S is black

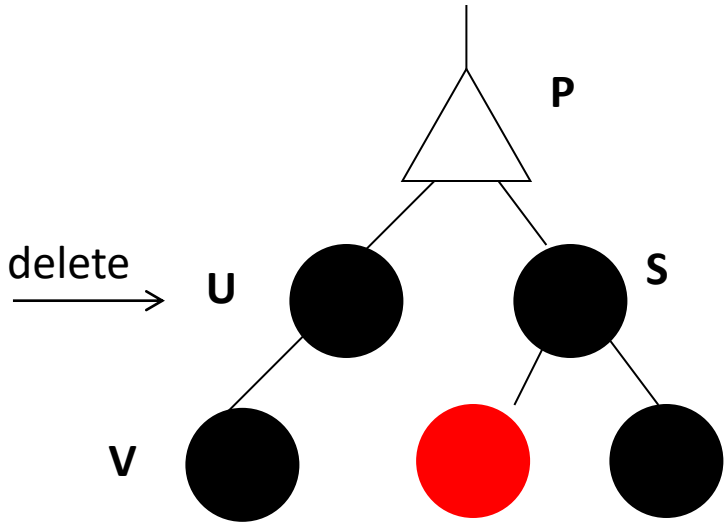
  S's RIGHT child is RED (Left child either color)

  Rotate S around P

  Swap colors of S and P, and color S's Right child Black

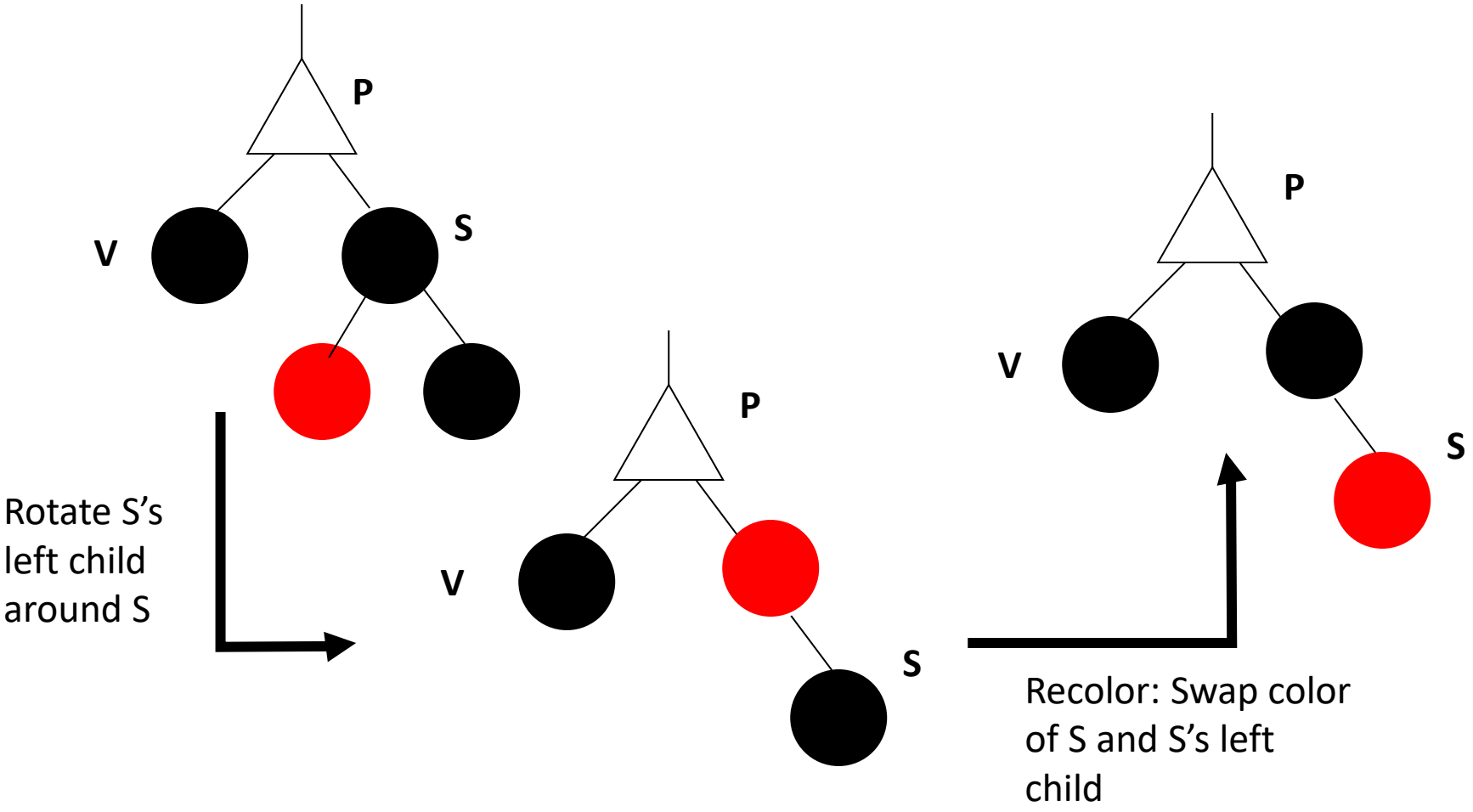


The red-black tree property is restored.



Case C:

- S is Black, S's right child is Black and S's left child is Red
  - i) Rotate S's left child around S
  - ii) Swap color of S and S's left child



The red-black tree property is restored locally for the tree rooted at the *P*. But, We may need to recurse where *P* is playing the role of *V*, possibly until we get to the root.

Analysis of Deletion

- A red-black tree has depth $O(\log n)$
- Search for deletion location takes $O(\log n)$ time
- The swapping and deletion is $O(1)$.
- Each repair iterations of rotation and/or recoloring is $O(1)$.
- Need to do at most $O(\log n)$ repair iterations.
- Thus, the deletion in a red-black tree takes $O(\log n)$ time.