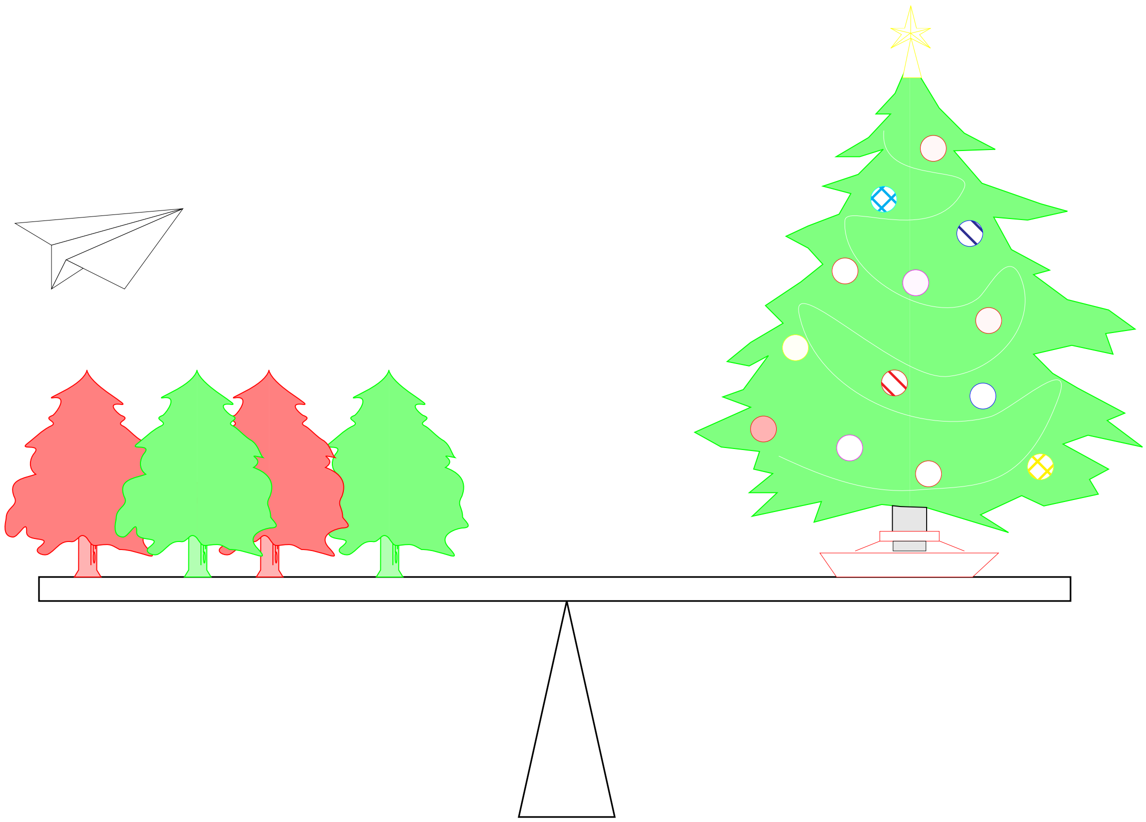
Red-Black Trees

- Insertion
- Deletion



Beyond (2,4) Trees

What do we know about (2,4)Trees?

- Balanced

- O(log *n*) search time

- Different node structures

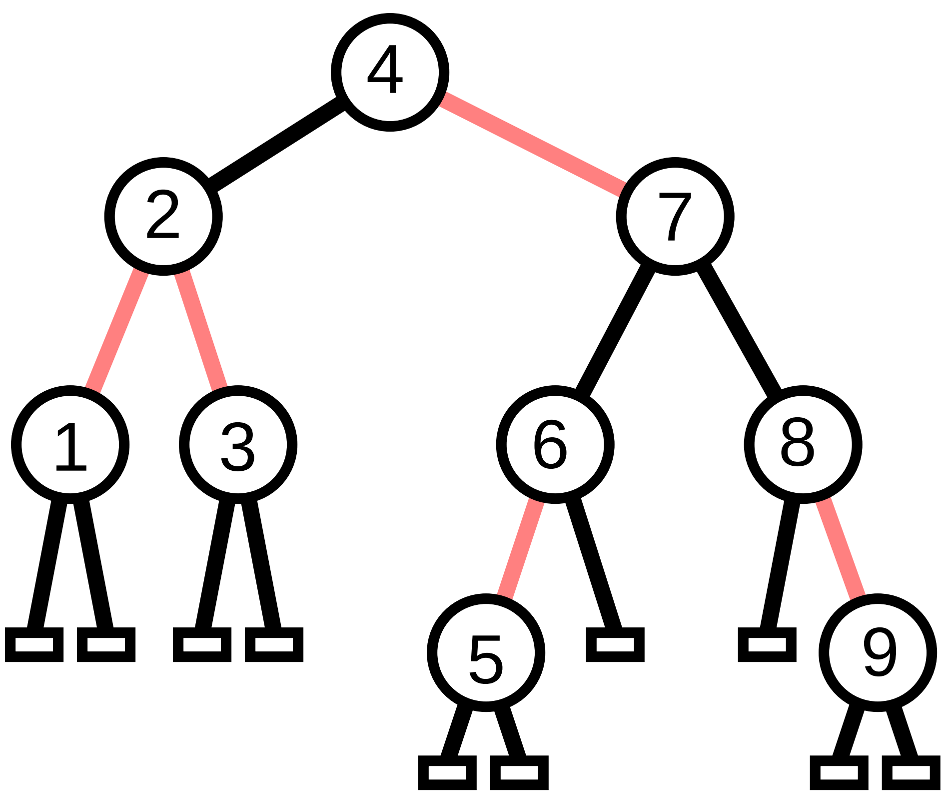
Can we get the (2,4) tree advantages in a binary tree format???

Welcome to the world of Red-Black Trees!!!

Red-Black Tree

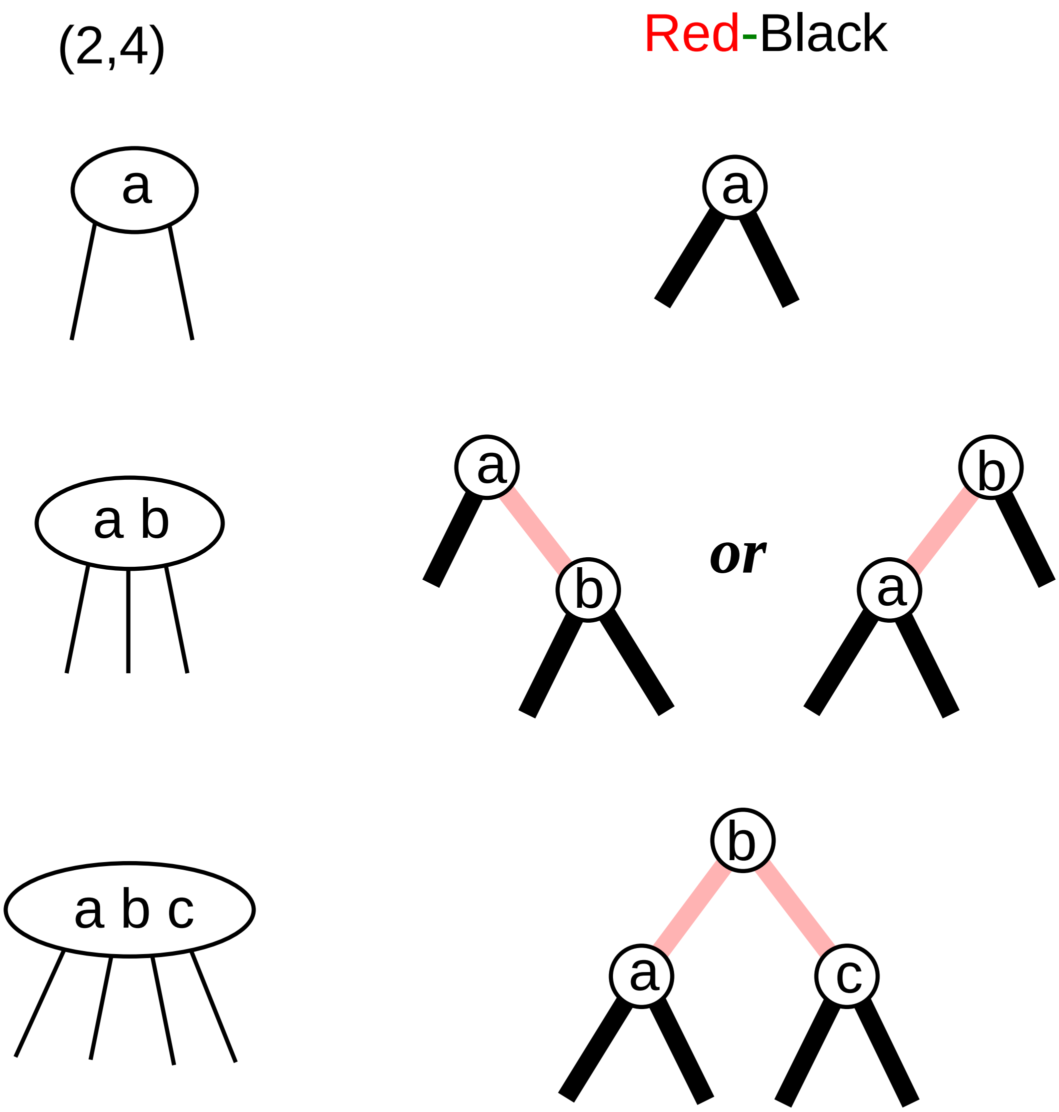
A **red-black tree** is a binary search tree with the following properties:

- edges are colored ***red*** or ***black***
- ***no two consecutive red edges on any root-leaf path***
- ***same number of black edges on any root-leaf path*** (***black height***)
- ***edges connecting leaves are black***



(2,4) Tree Evolution

Note how **(2,4) trees** relate to red**-black trees**

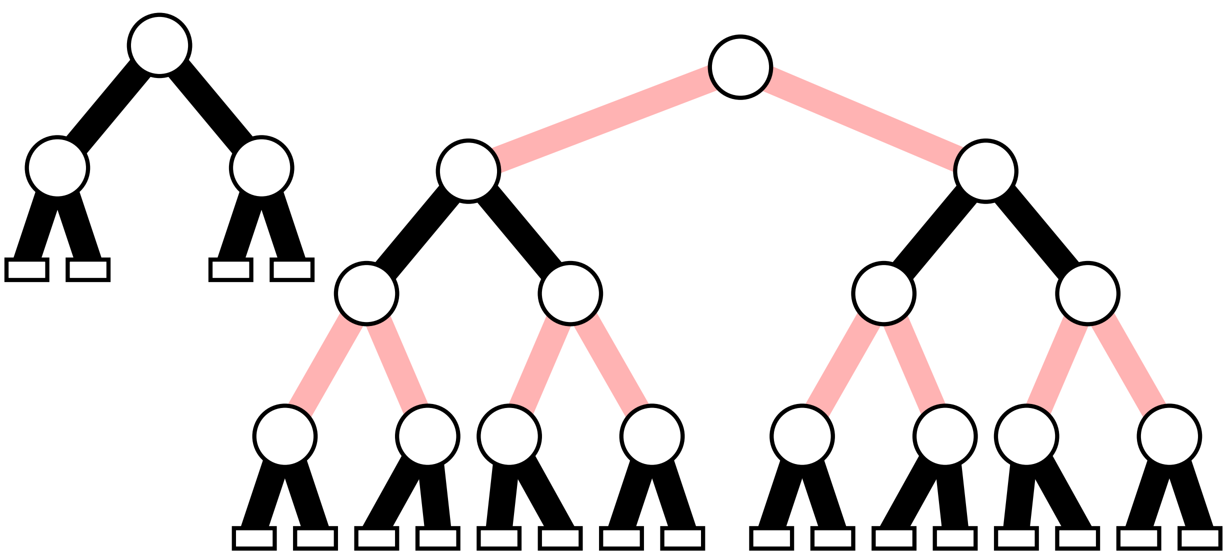


Now we see red-black trees are just a way of representing 2-3-4 trees!

Red-Black Tree Properties

N # of internal nodes
L # leaves (= N + 1)
H height
B black height

Property 1: $2^B \leq N + 1 \leq 4^B$



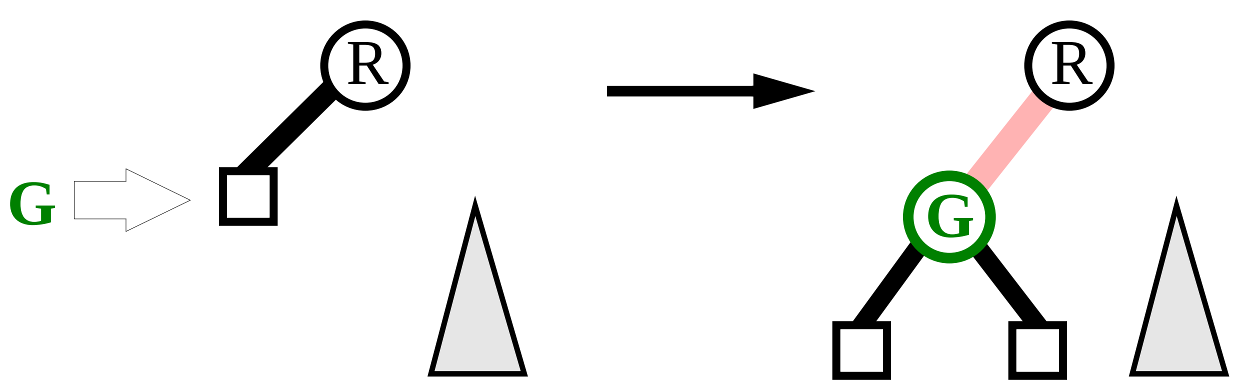
Property 2: $\frac{1}{2}\log(N + 1) \leq B \leq \log(N + 1)$

Property 3: $\log(N + 1) \leq H \leq 2\log(N + 1)$

This implies that searches take time O(log N)!

Insertion into Red-Black

1. Perform a standard search to find the leaf where the key should be added
2. Replace the leaf with an internal node with the new key
3. Color the incoming edge of the new node red
4. Add two new leaves, and color their incoming edges black
5. If the parent had an incoming red edge, we now have two consecutive red edges! We must reorganize tree to remove that violation. What must be done depends on the sibling of the parent.

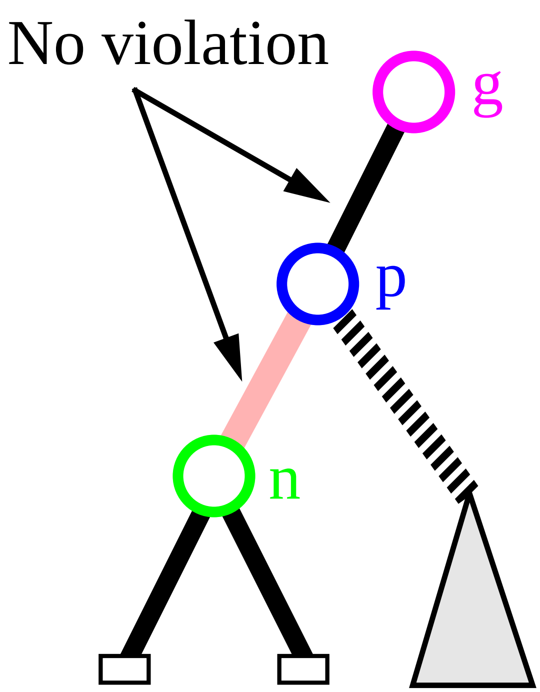


Insertion - Plain and Simple

Let:

- n be the new node
- p be its parent
- g be its grandparent

Case 1: Incoming edge of p is black



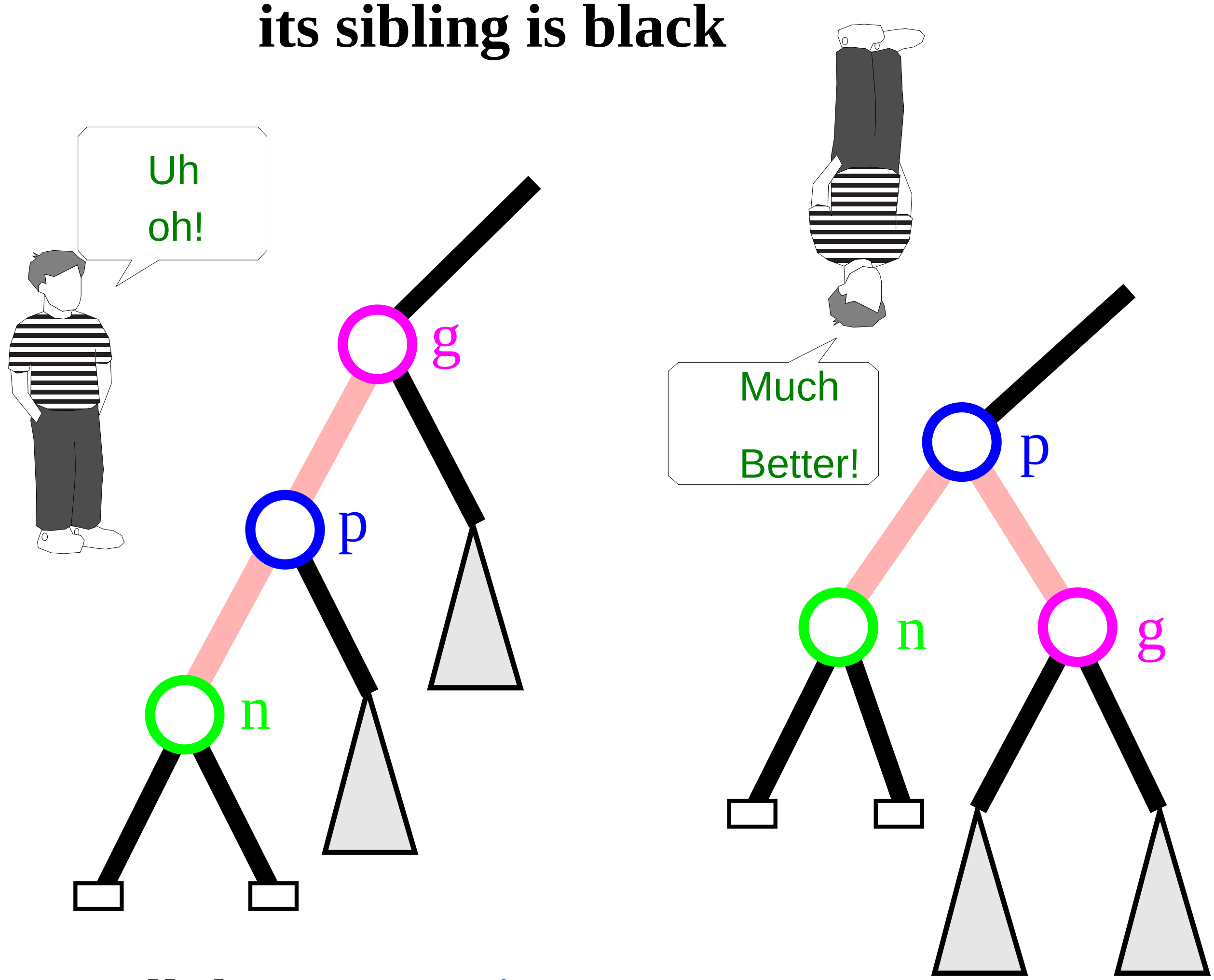
STOP!

Pretty easy, huh?

Well... it gets messier...

Restructuring

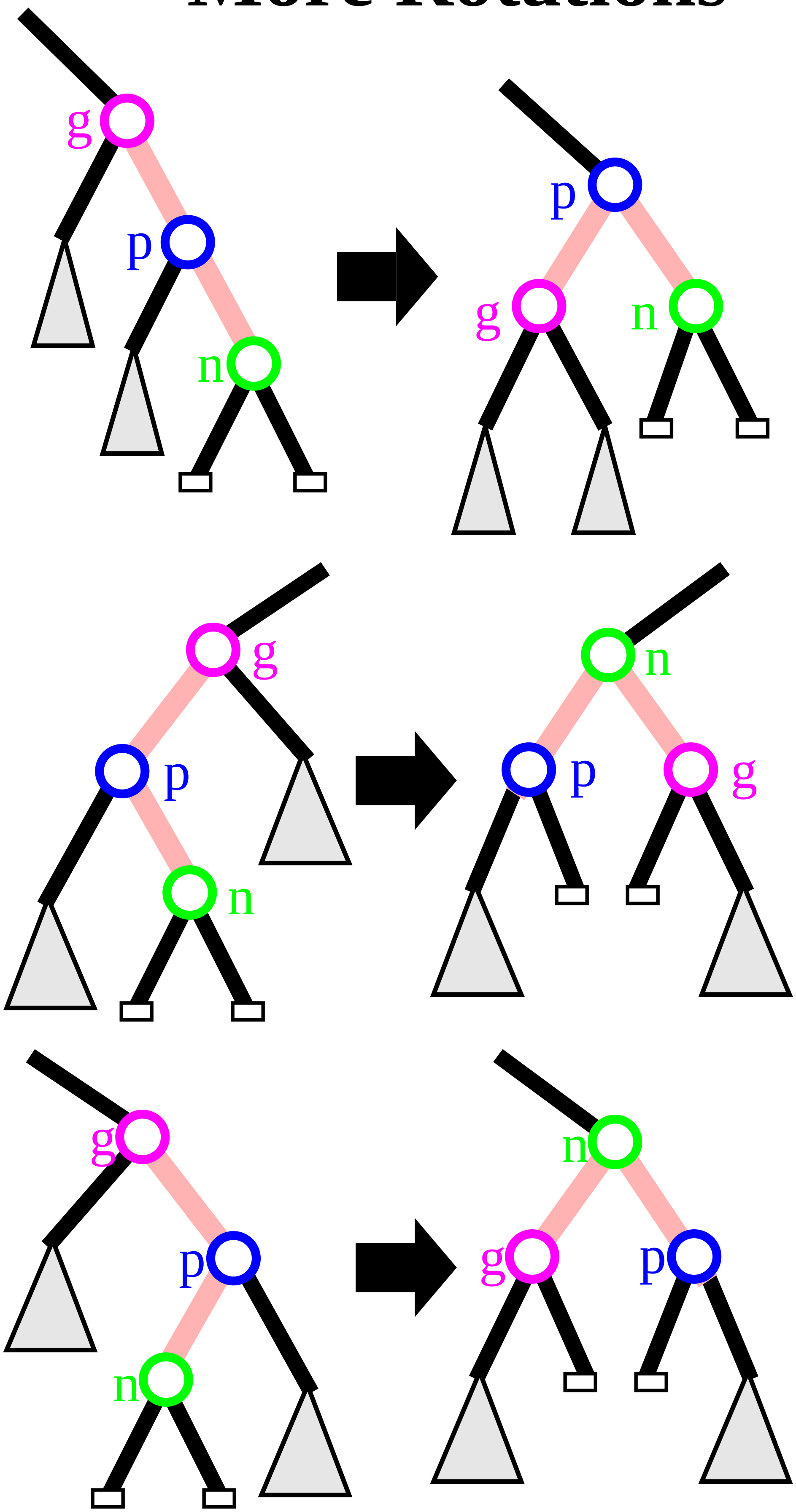
Case 2: Incoming edge of p is red, and its sibling is black



We call this a "***rotation***"

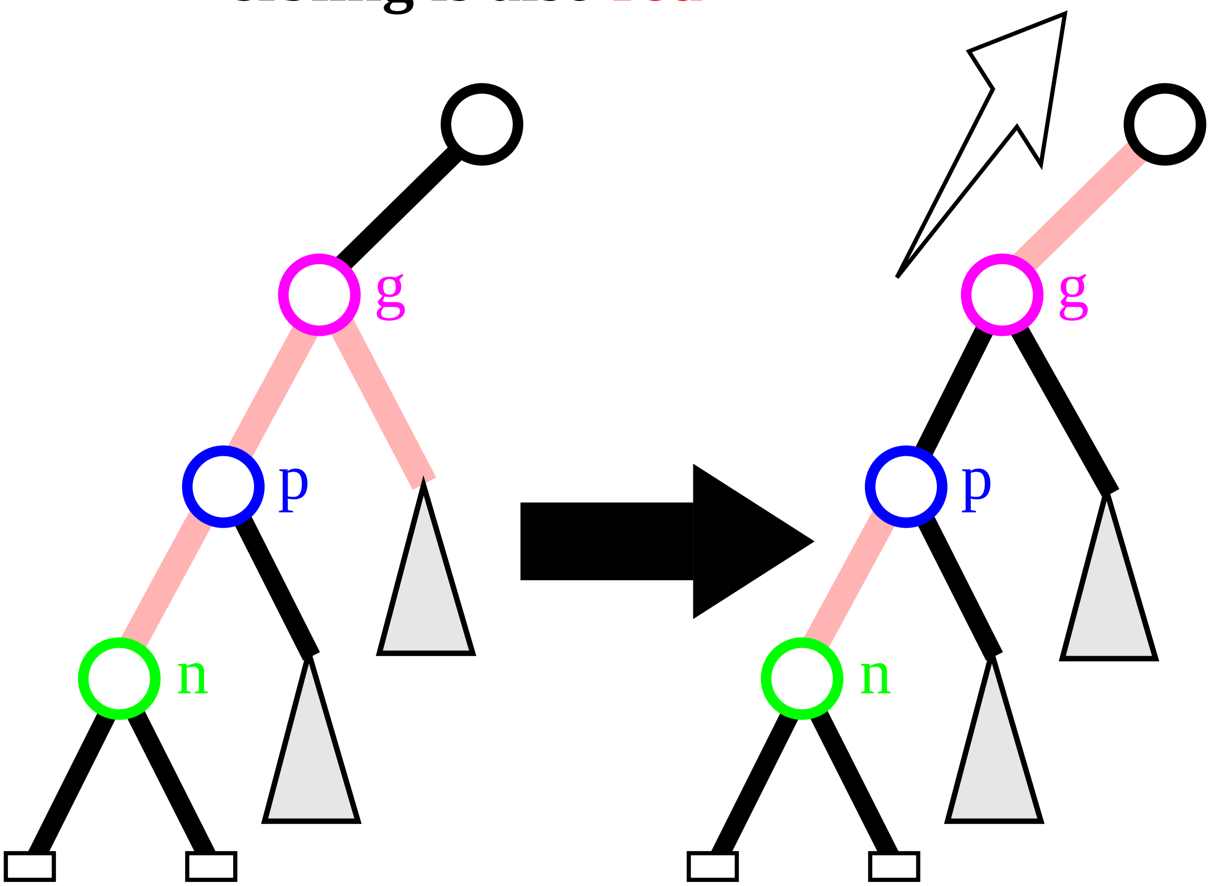
- No further work necessary
- Inorder remains unchanged
- Black depth is preserved for all leaves
- No more consecutive red edges!
- Corrects "malformed" 4-node in the associated (2,4) tree

More Rotations



Promotion

Case 3: Incoming edge of p is red and its sibling is also red



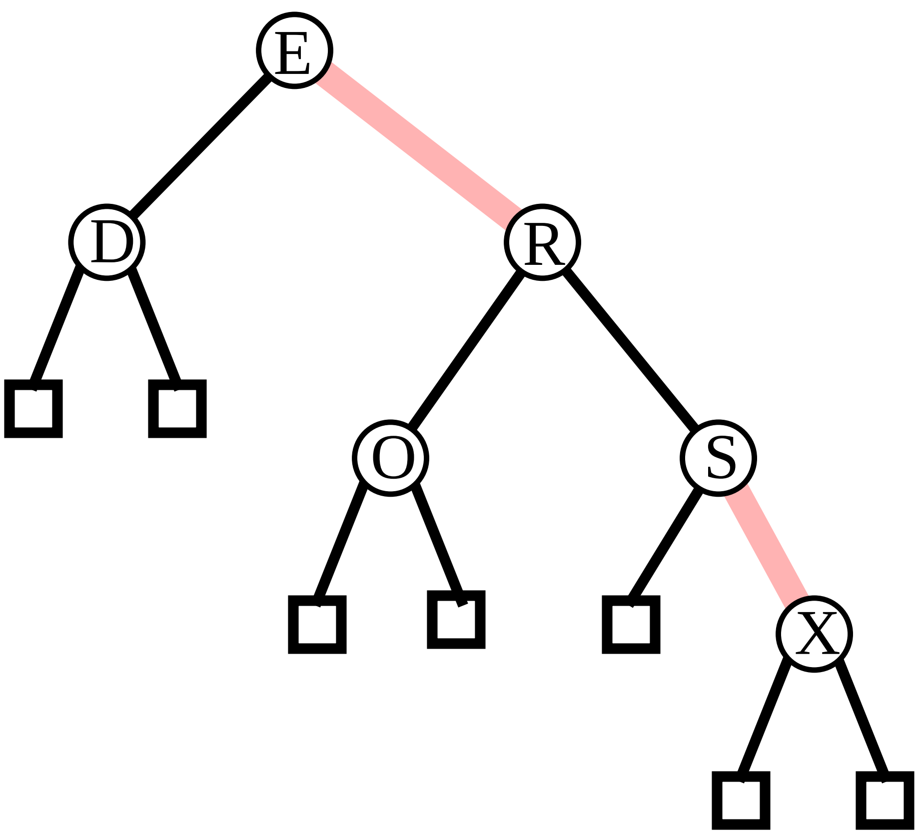
- We call this a "***recoloring***"
- The black depth remains unchanged for all the descendants of **g**
- This process will continue upward beyond **g** if necessary: rename **g** as **n** and repeat.
- Splits 5-node of the associated (2,4) tree

Summary of Insertion

- If two red edges are present, we do either
  - a *restructuring* (with a simple or double rotation) and **stop**, or
  - a recoloring and continue
- A restructuring takes constant time and is performed at most once. It reorganizes an off-balanced section of the tree.
- Recolorings may continue up the tree and are executed O(log N) times.
- The **time complexity** of an insertion is **O(log N).**

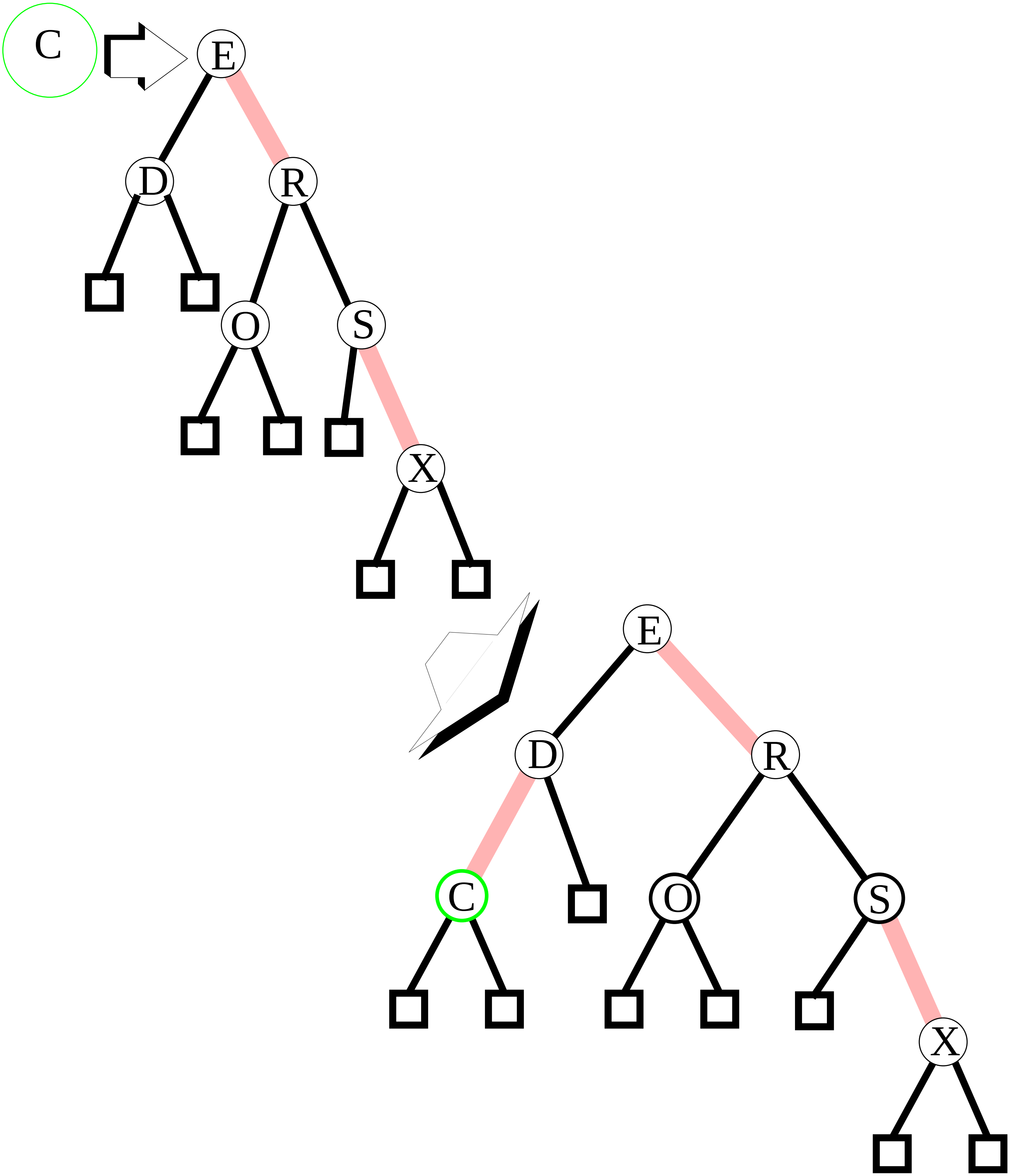
An Example

Start by inserting "REDSOX" into an empty tree

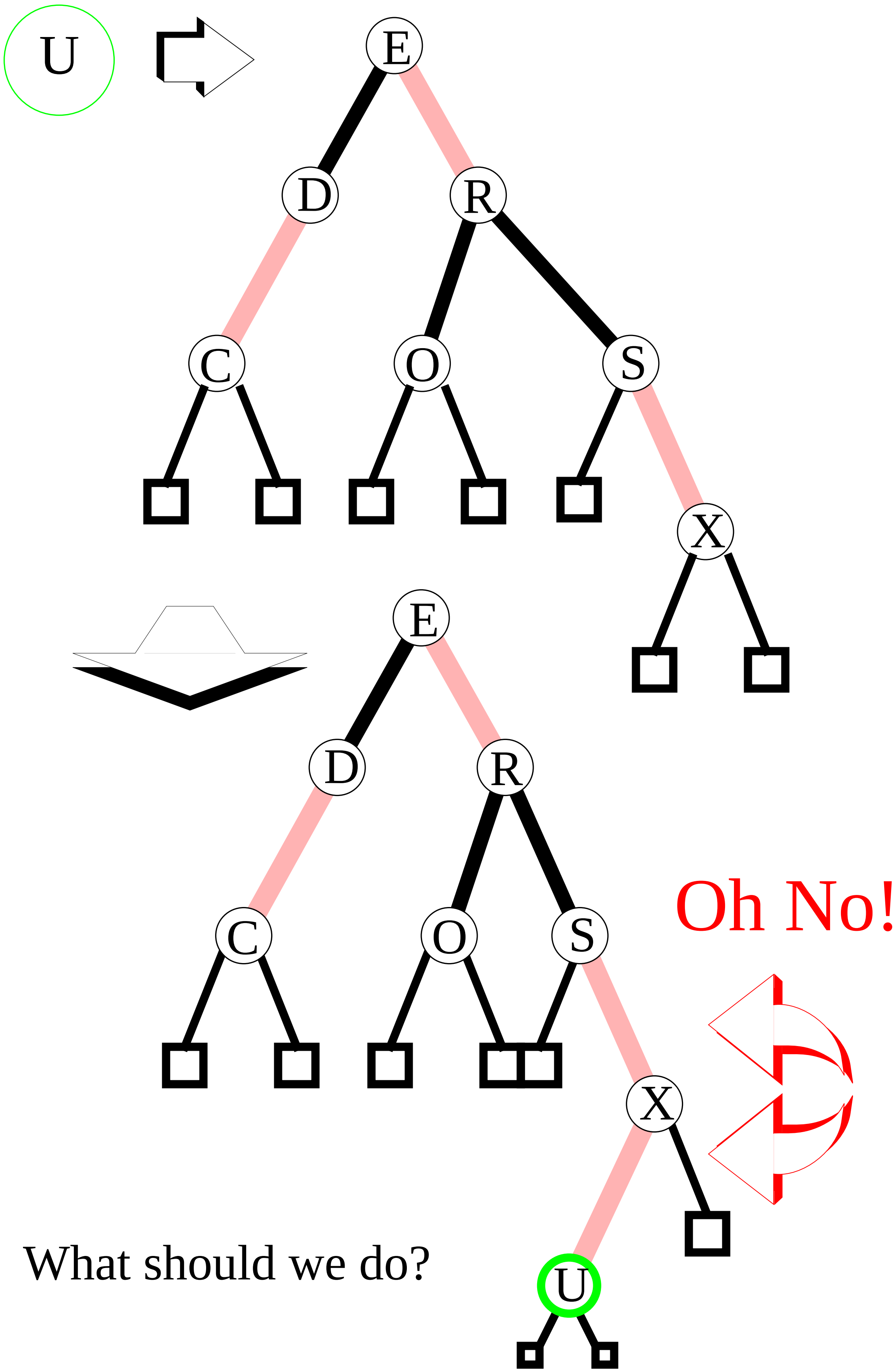


Now, let's insert "C U B S"...

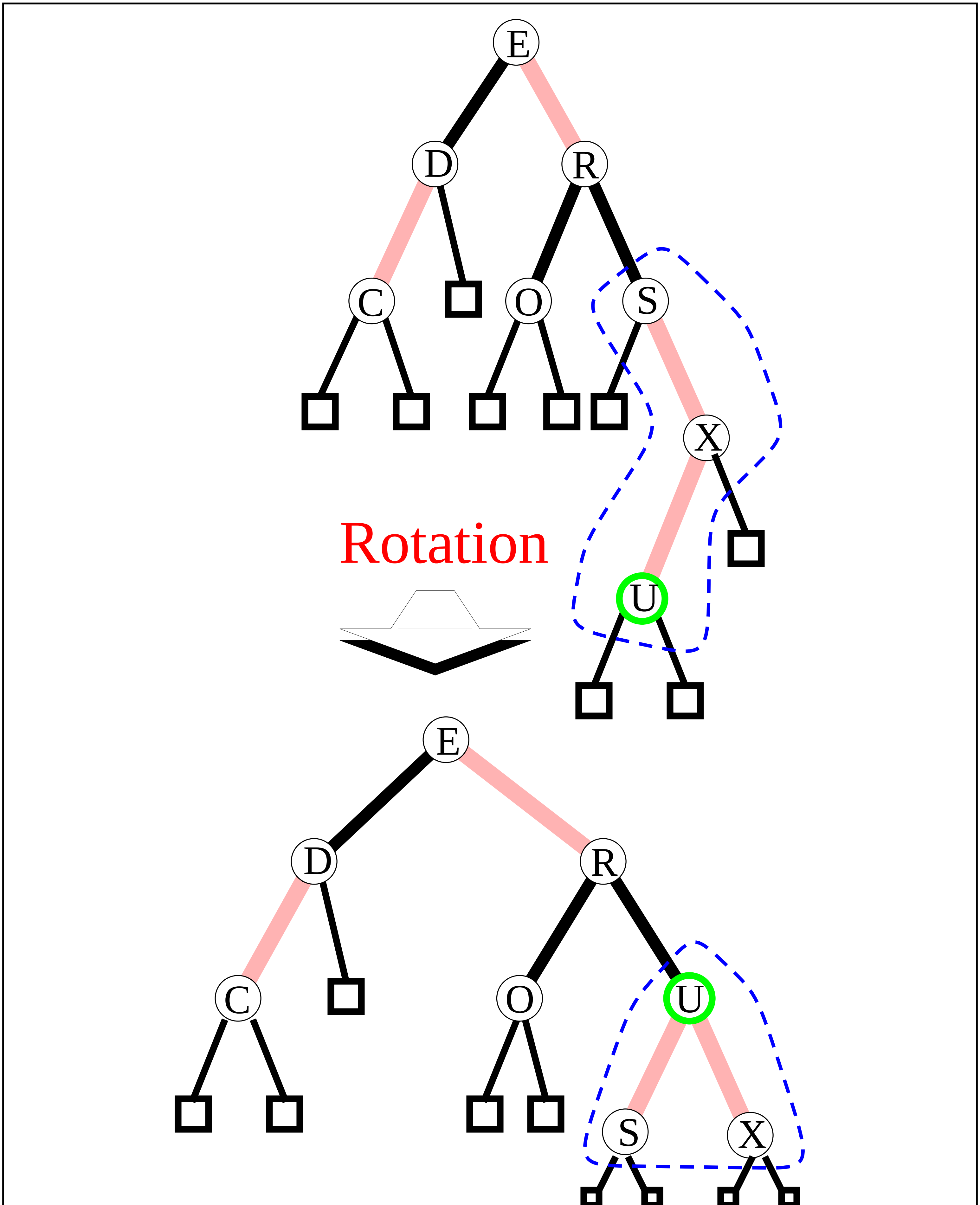
A Cool Example



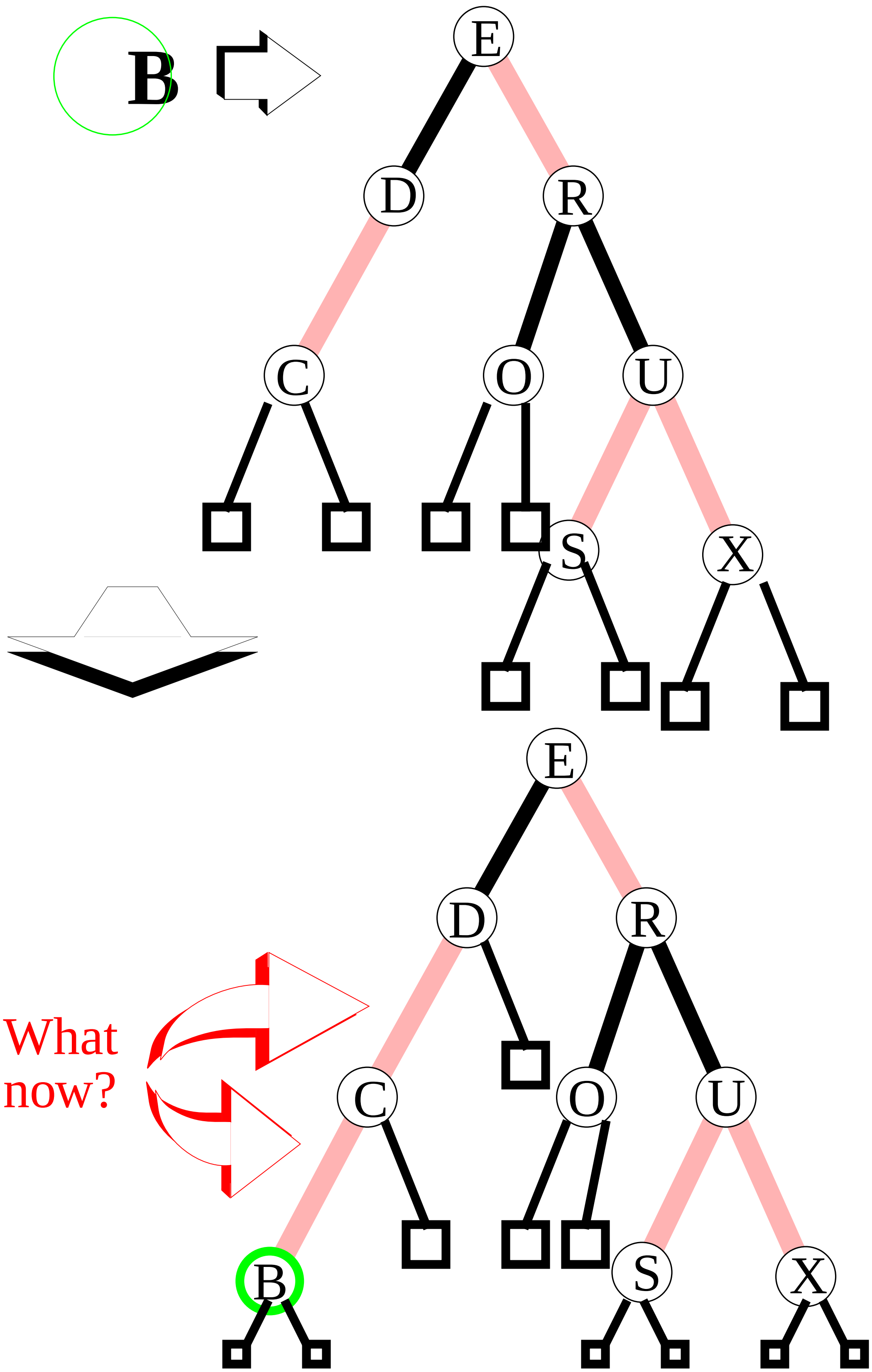
An Unbelievable Example

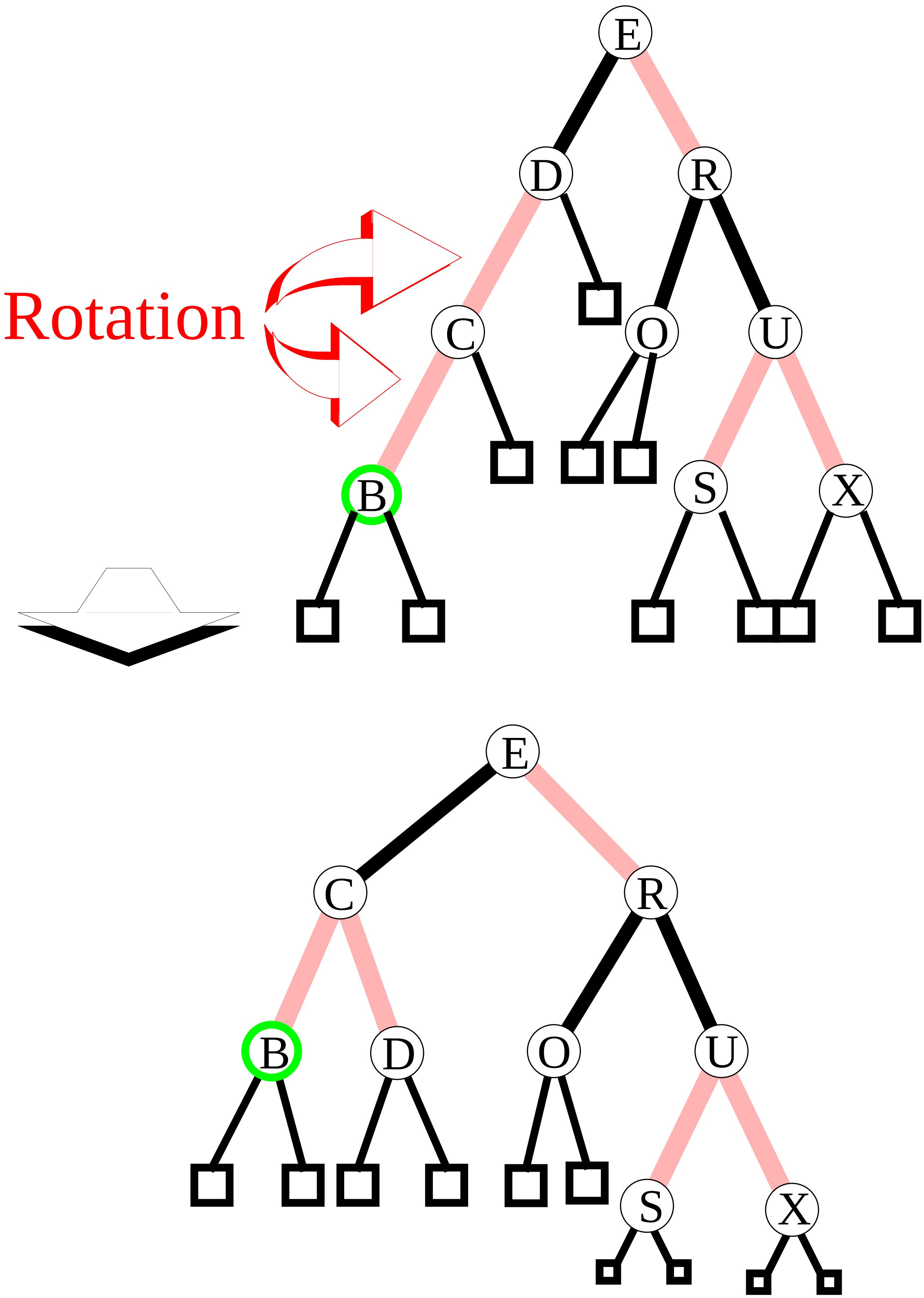


What should we do?

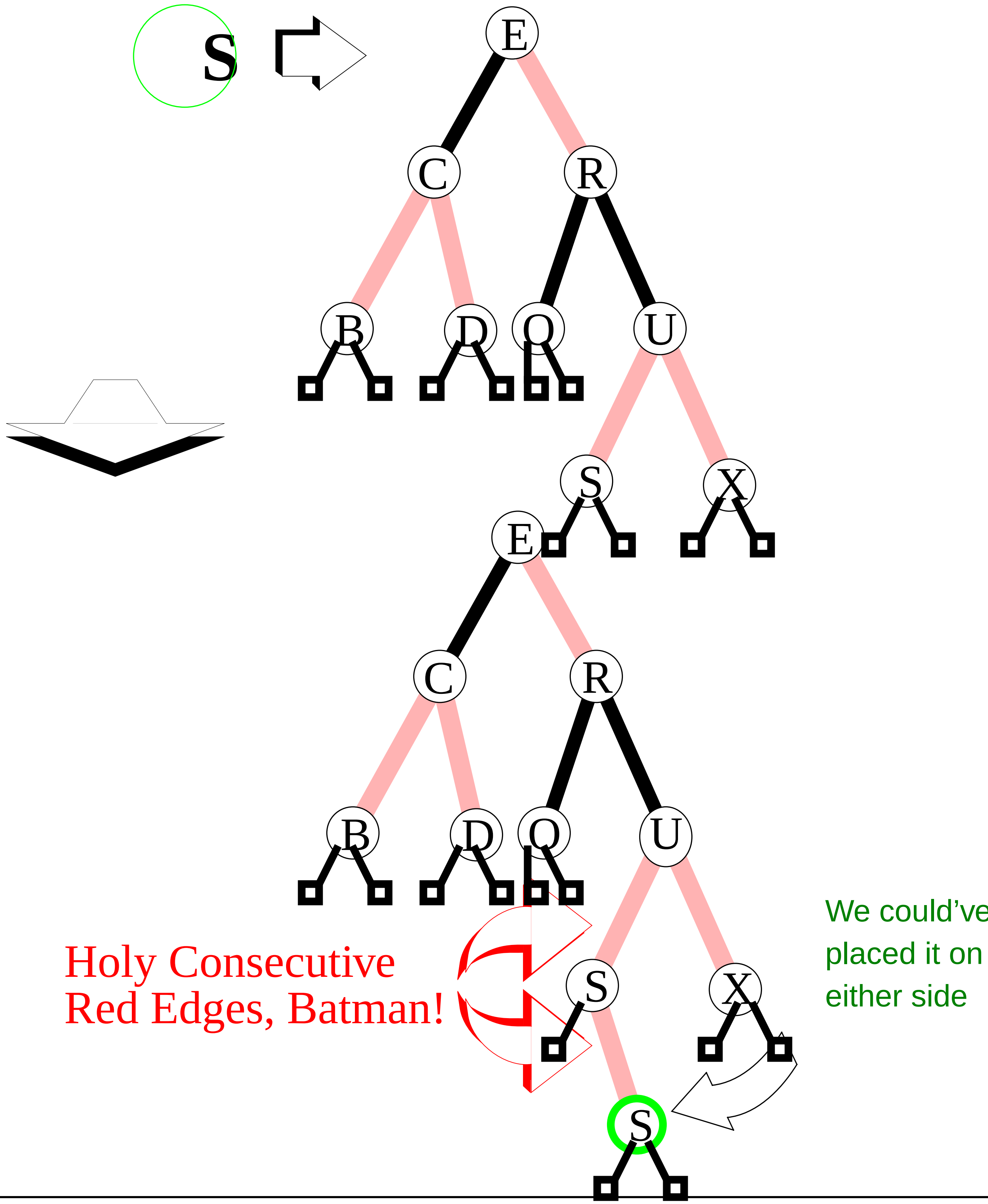

E
D
R
C
O
S
X
Rotation
U
E
D
R
C
O
U
S
X

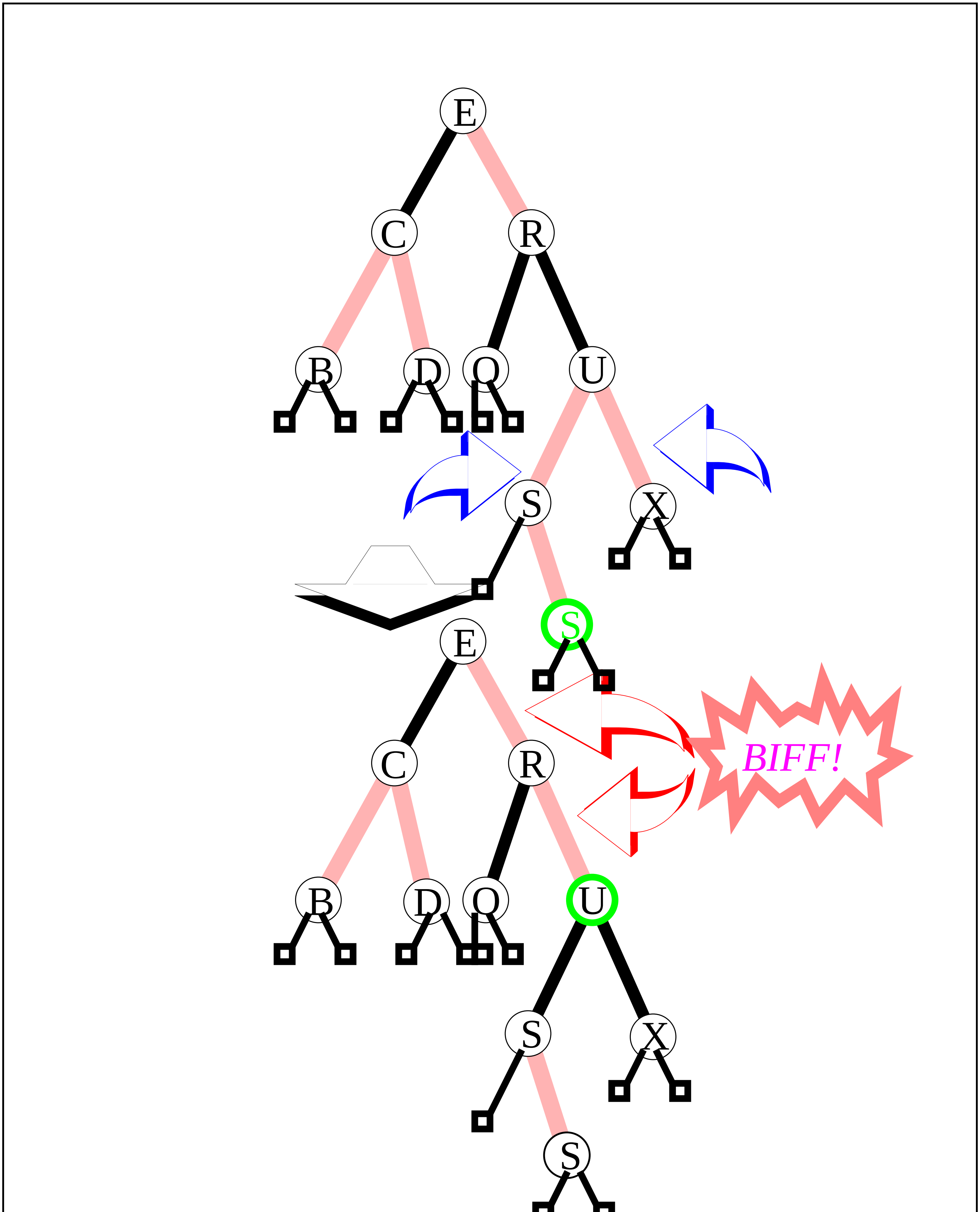
A Beautiful Example

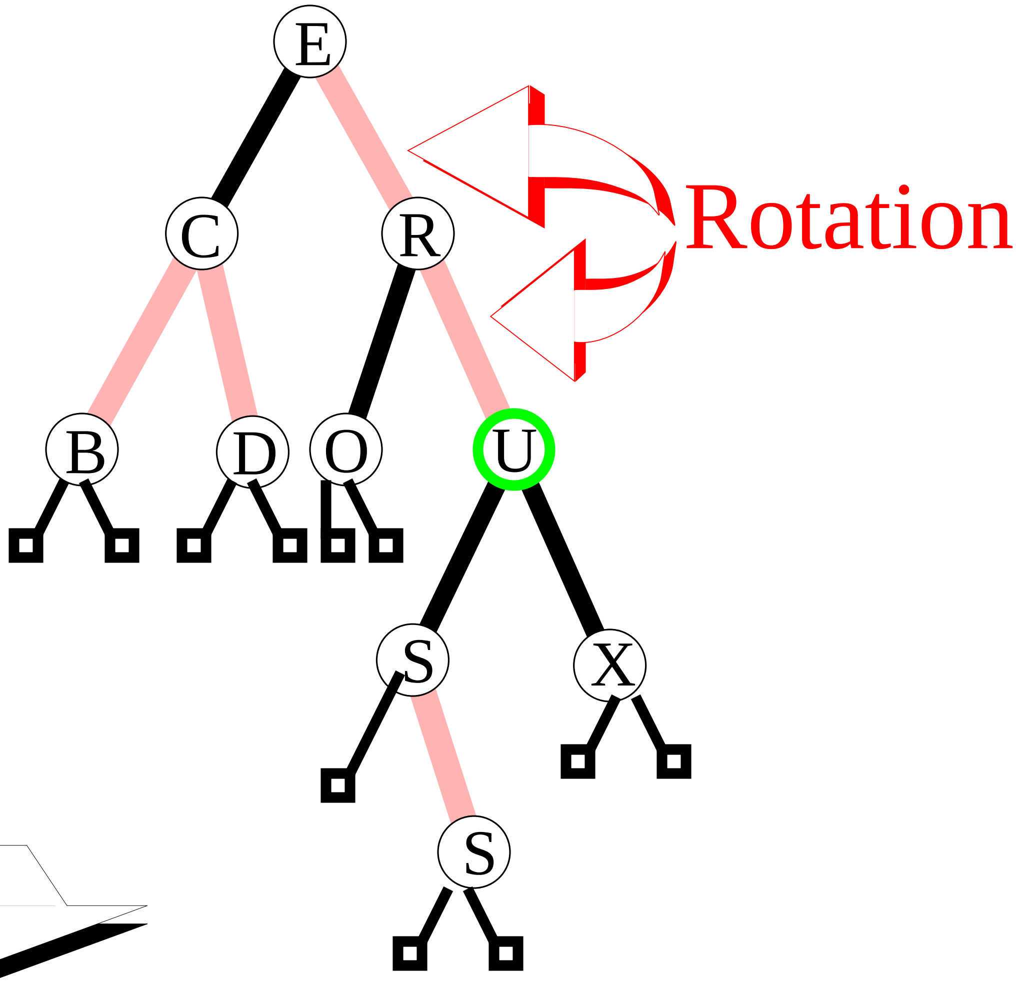



Rotation
E
D
R
C
O
U
B
S
X
E
C
R
B
D
O
U
S
X

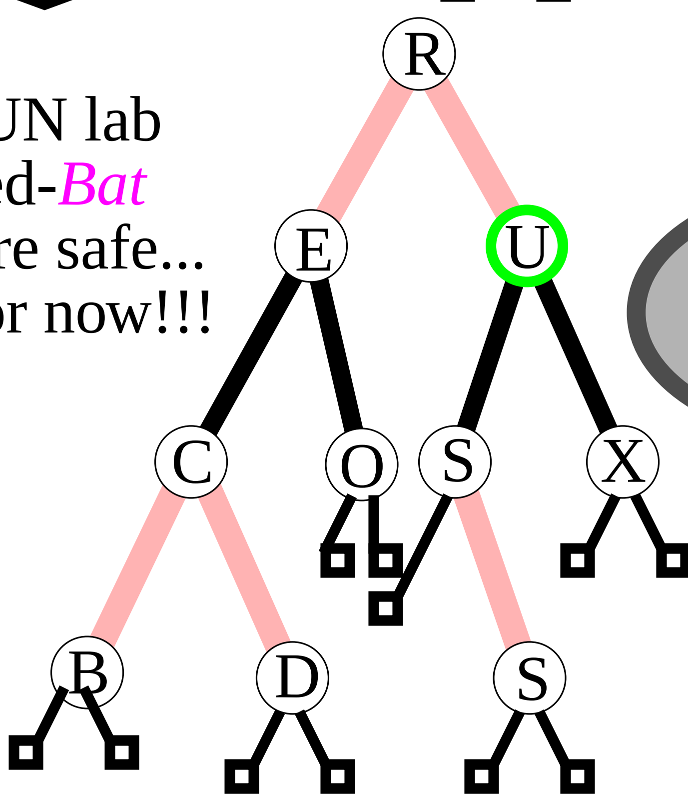
A Super Example

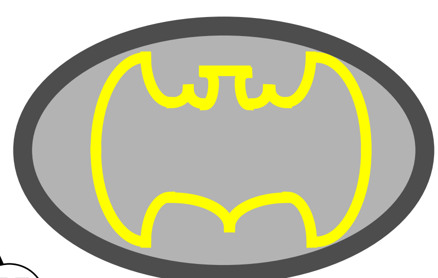



E
C
R
B
D
Q
U
S
X
S
E
C
R
BIFF!
B
D
Q
U
S
X
S



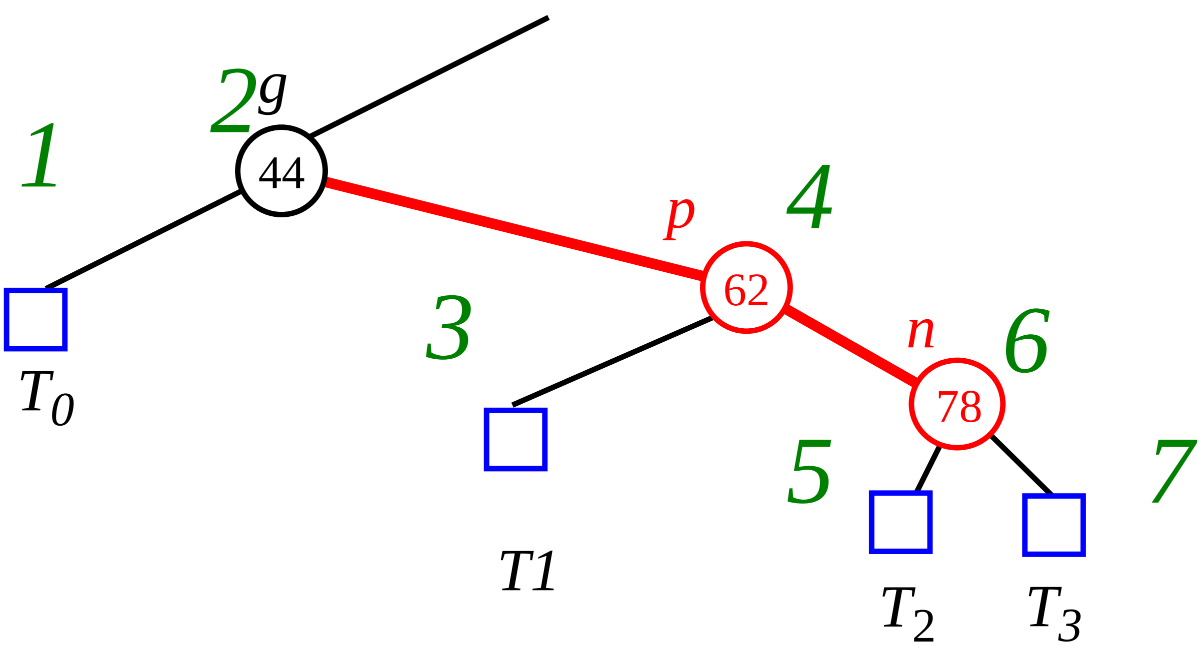
The SUN lab
and Red-*Bat*
trees are safe...
...for now!!!





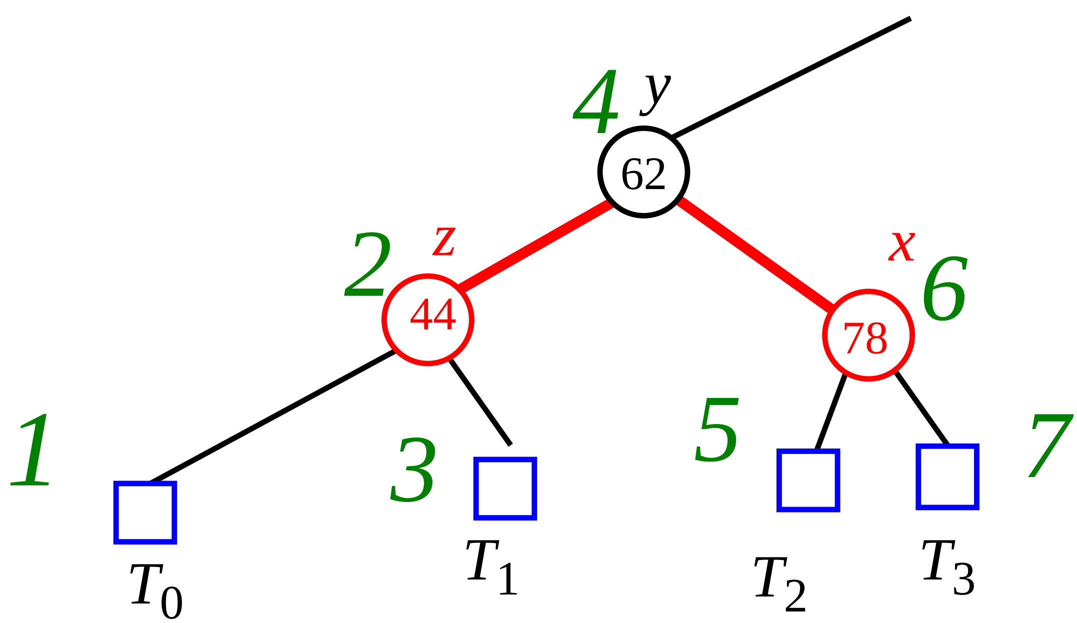
Cut/Link Restructure Algorithm

- Remember the cut/link restructure algorithm from AVL tree lecture? We can use it to implement rotation.
- We use an inorder traversal to restructure the tree as before
- For example, below we have a subtree with two consecutive red edges.

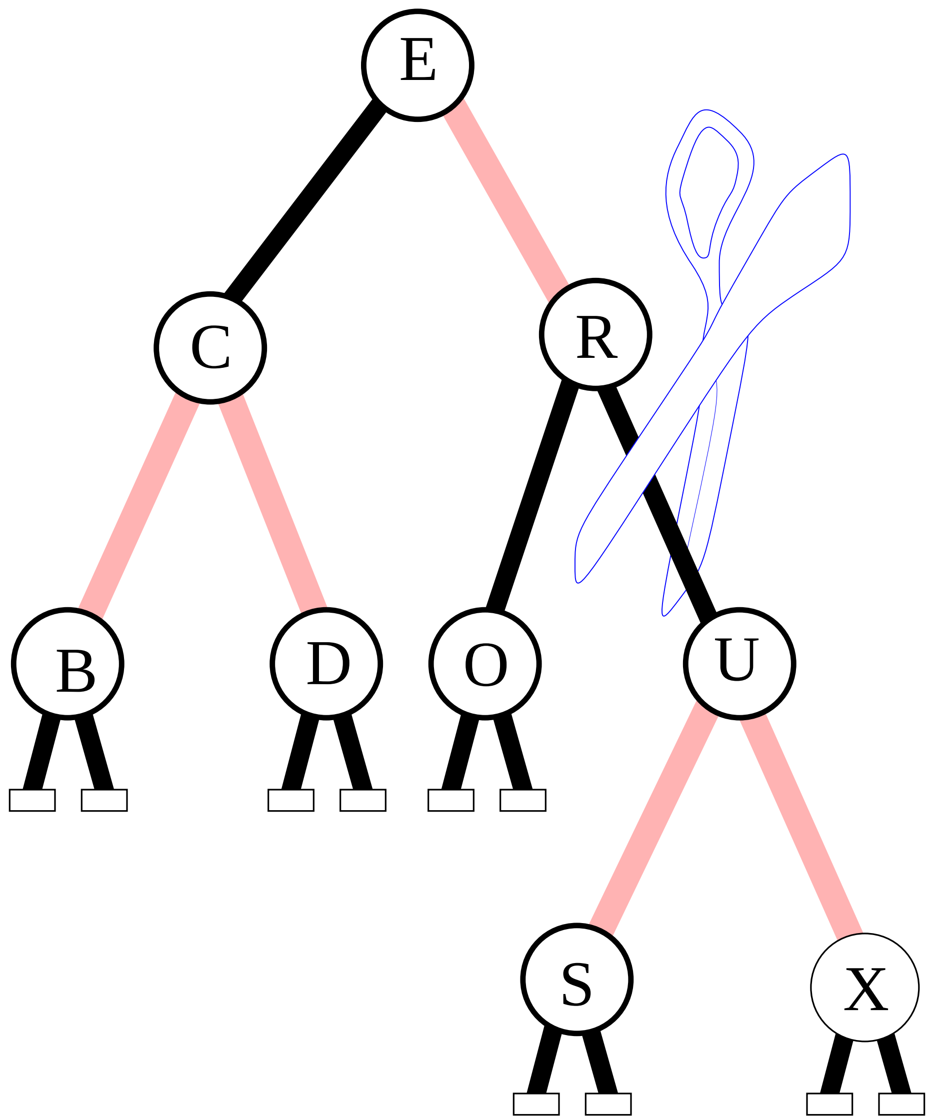


Cut/Link Restructure Algorithm(cont.)

- But there is one more consideration in the case of a red-black tree: recoloring.
- In this case, the root of the subtree should be the same color as the former root was, and both of its children should be colored red. This is the only recoloring case for Insertion.
- For deletion, you will need to perform “color compensation” (you’ll hear about it in a minute) on the grandchildren.



Deletion from Red-Black Trees

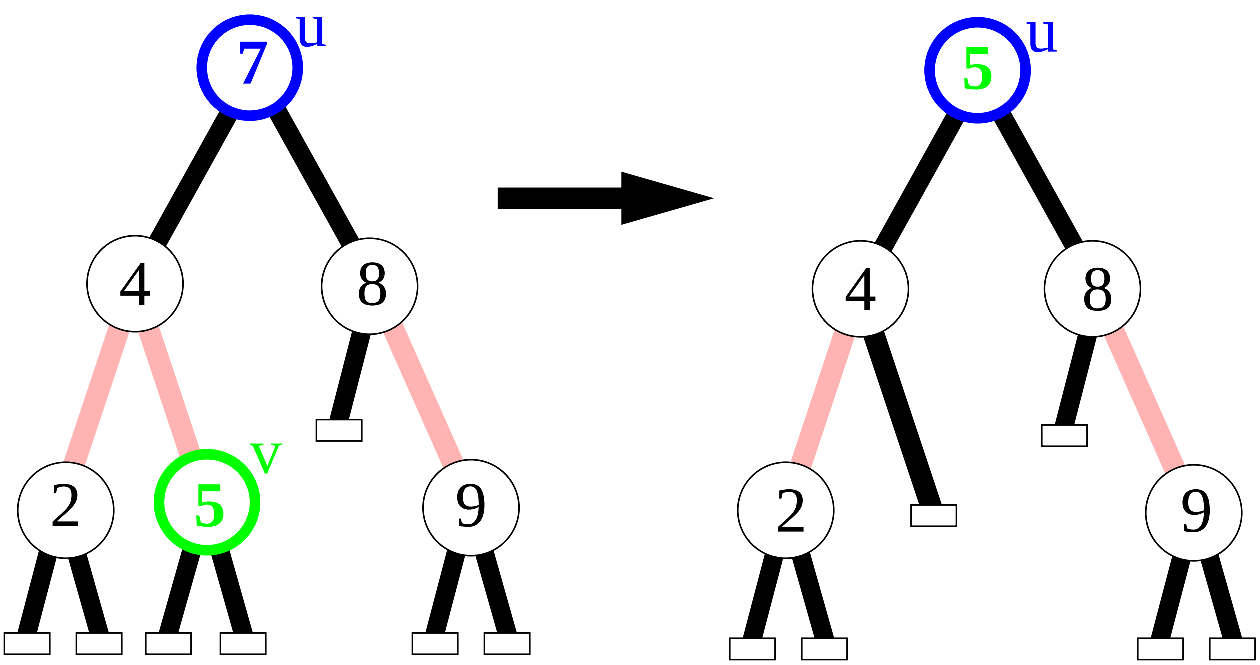


Setting Up Deletion

As with binary search trees, we can always delete a node that has at least one external child

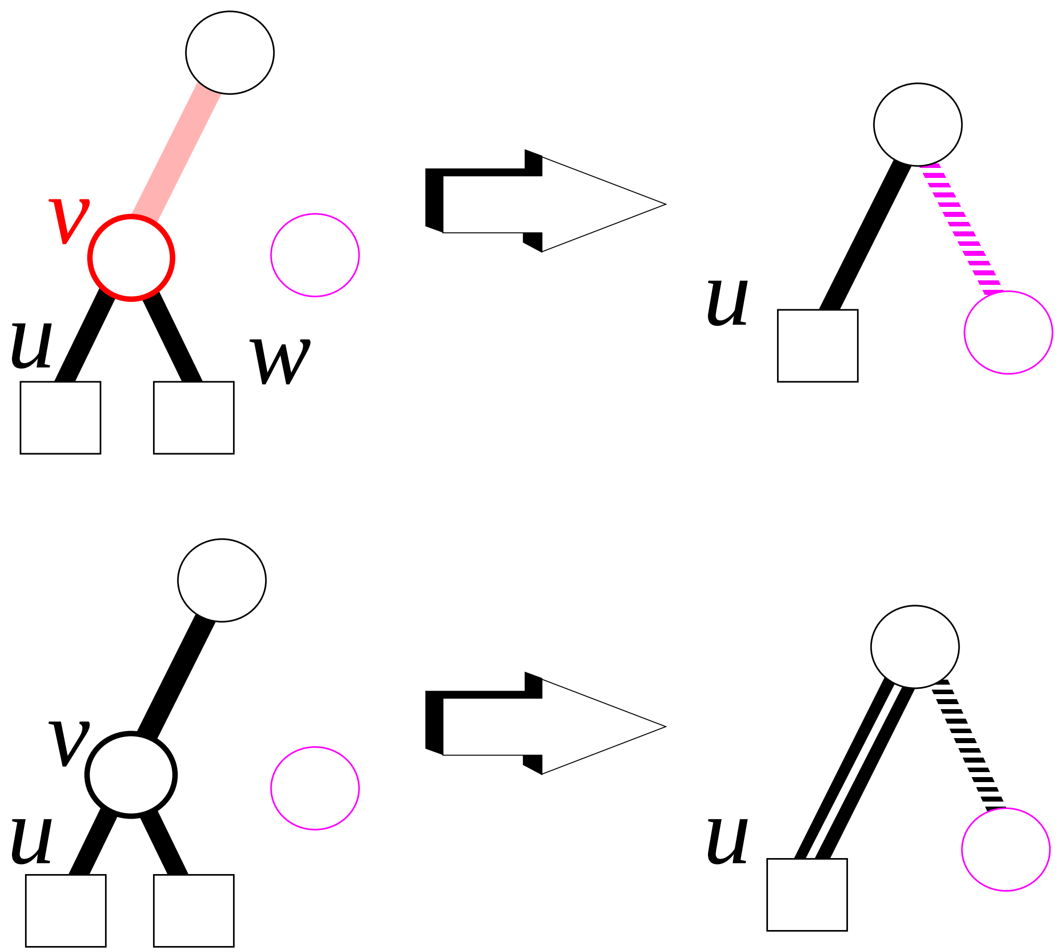
If the key to be deleted is stored at a node that has no external children, we move there the key of its inorder predecessor (or successor), and delete that node instead

Example: to delete key 7, we move key 5 to node u, and delete node v



Deletion Algorithm

1. Remove *v* with a removeAboveExternal operation on a leaf child *w* of *v*
2. If *v* was red or *u* is red, color *u* black. Else, color *u* ***double black***.



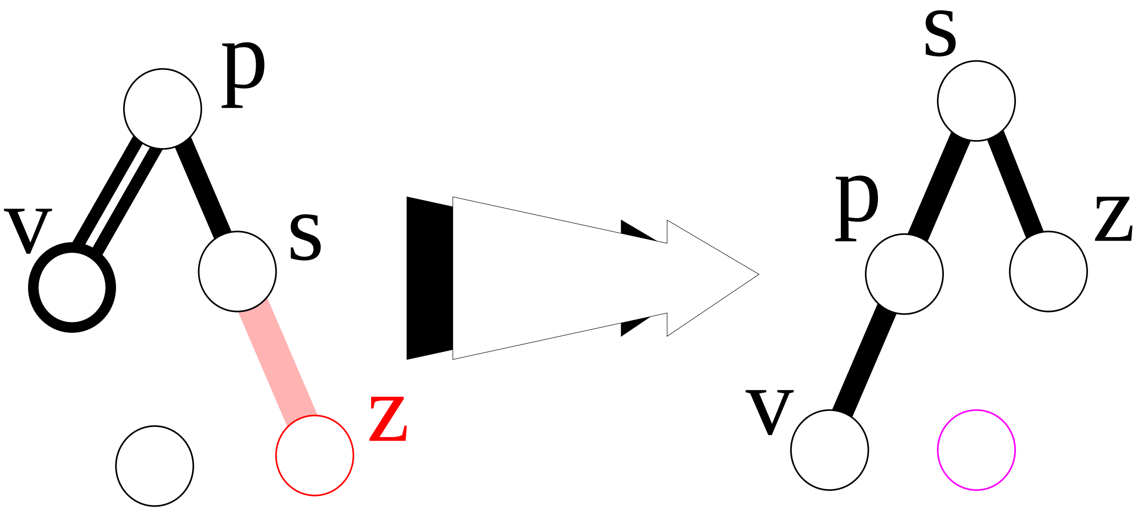
3. While a ***double black*** edge exists, perform one of the following actions ...

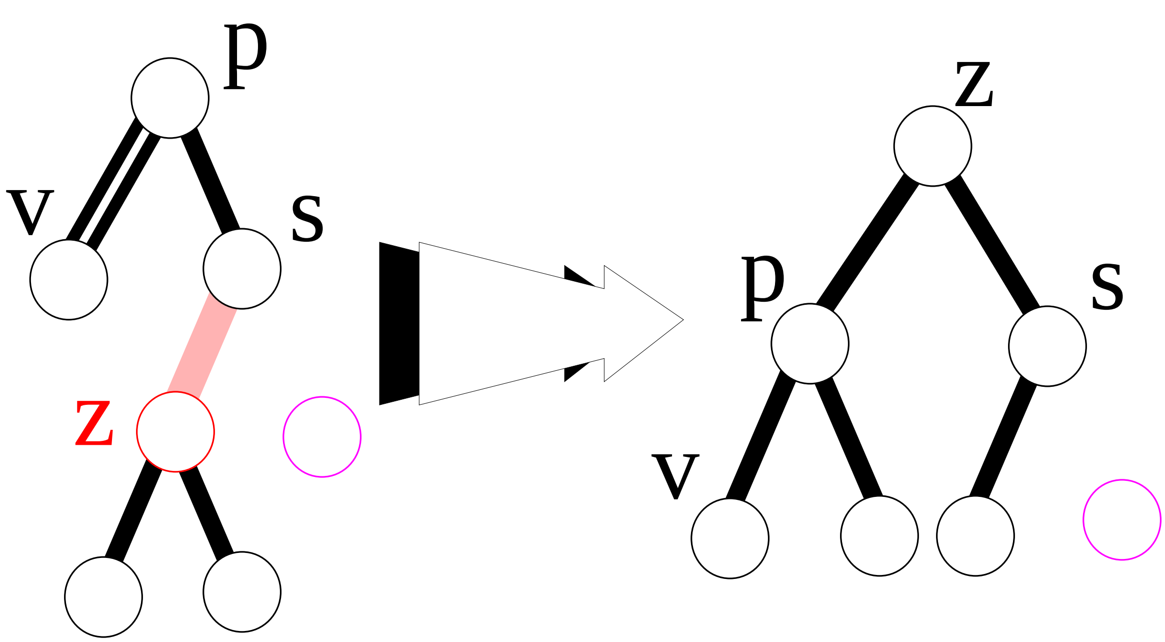
How to Eliminate the Double Black Edge

- The intuitive idea is to perform a "*color compensation*"
- Find a red edge nearby, and change the pair (*red* , *double black*) into
  (*black* , *black*)
- As for insertion, we have two cases:
  - *restructuring*, and
  - *recoloring* (*demotion*, inverse of promotion)
- Restructuring resolves the problem locally, while *recoloring* may propagate it two levels up
- Slightly more complicated than insertion, since two restructurings may occur (instead of just one)

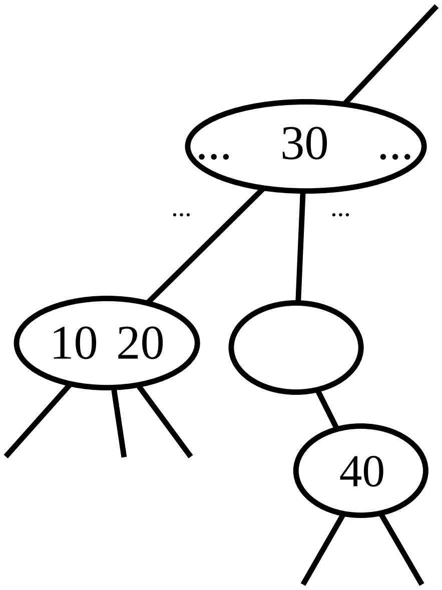
Case 1: black sibling with a red child

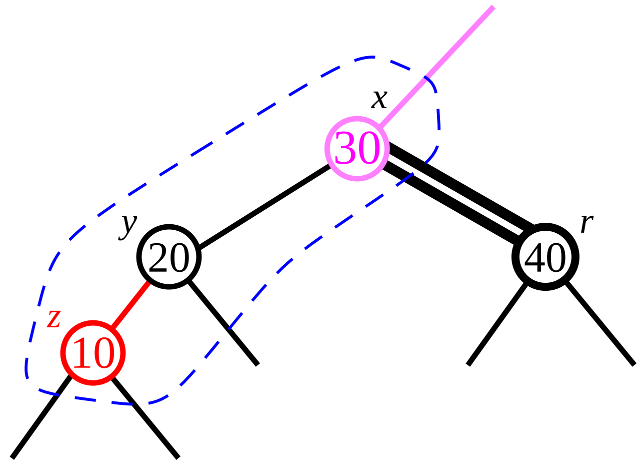
- If sibling is **black** and one of its children is **red**, perform a ***restructuring***

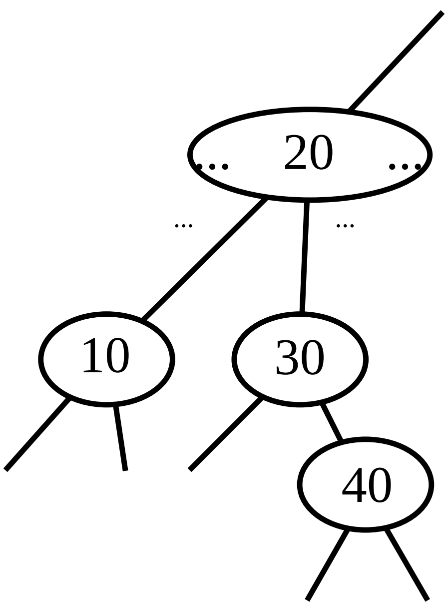


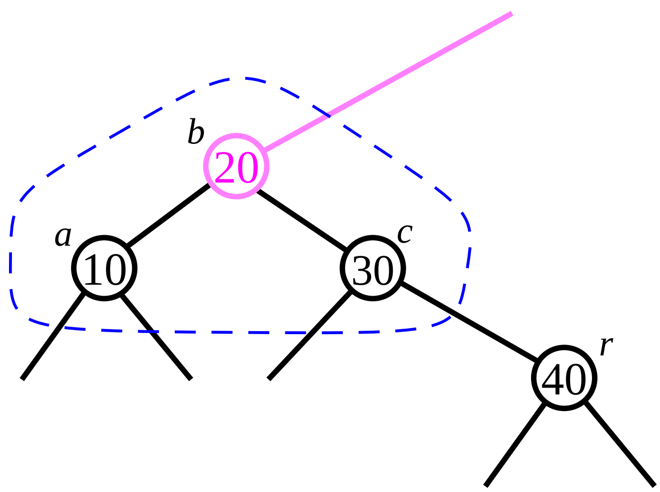


(2,4) Tree Interpretation



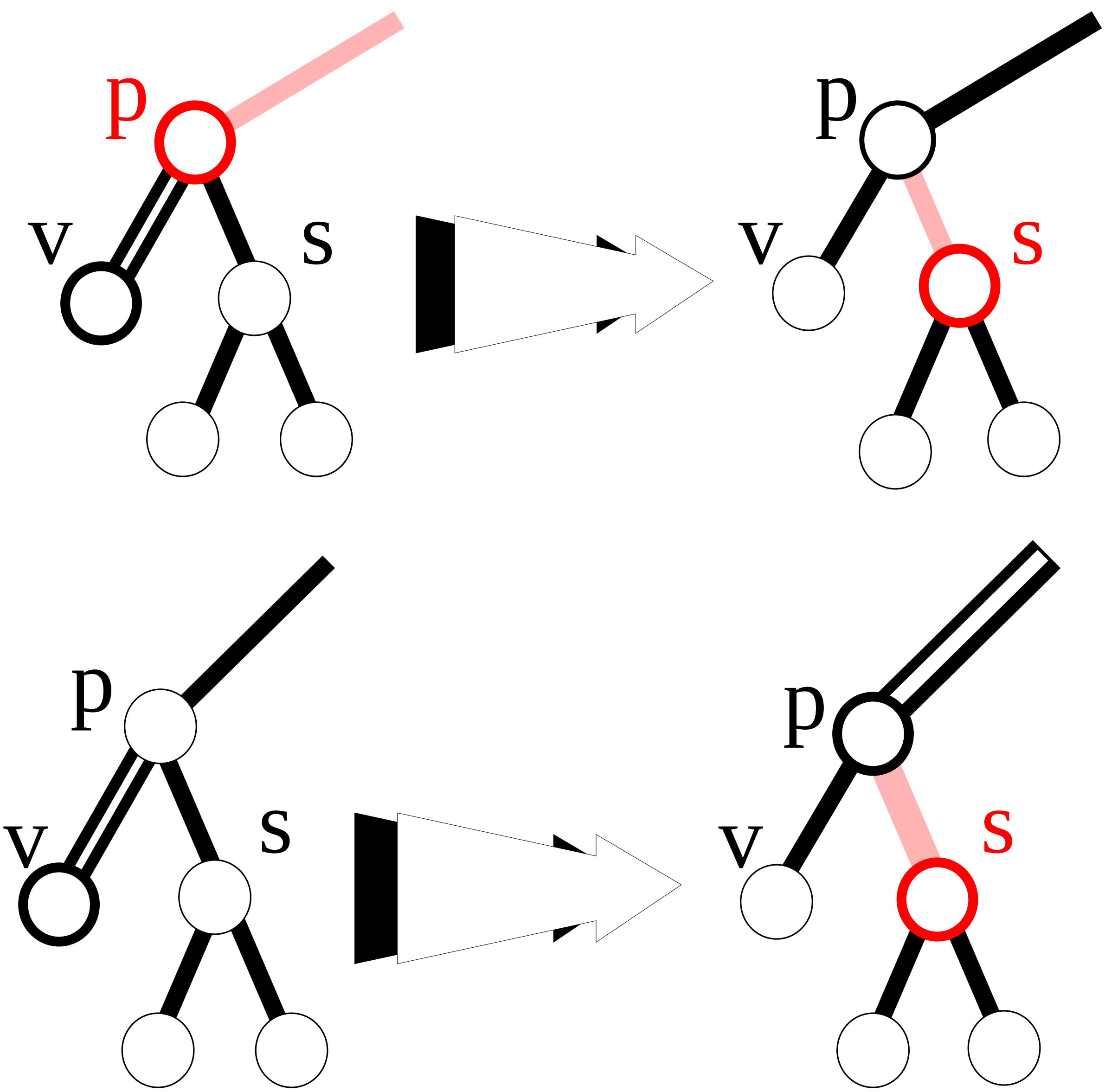




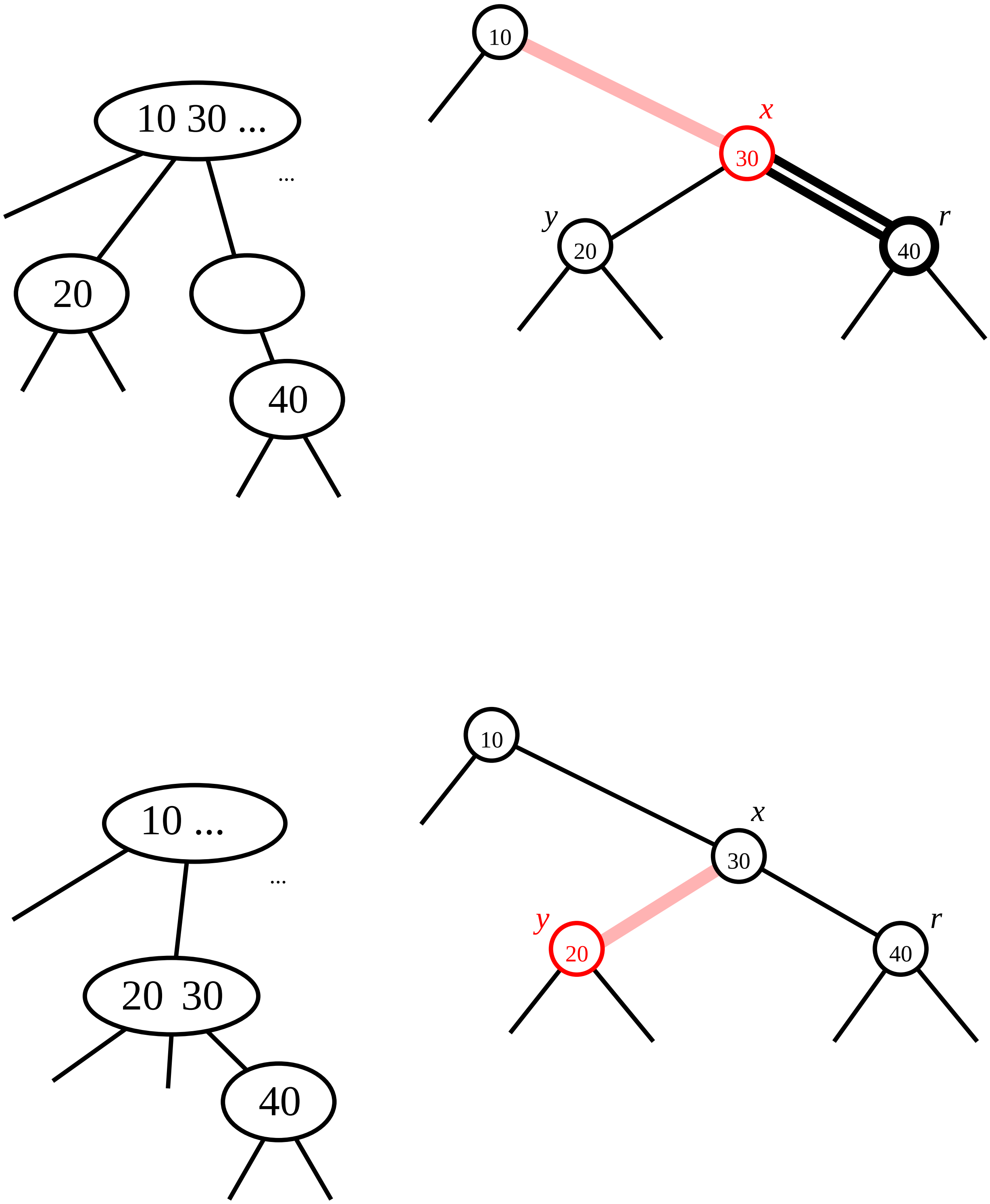


Case 2: black sibling with black childern

- If sibling and its children are **black**, perform a ***recoloring***
- If parent becomes **double black**, *continue* upward

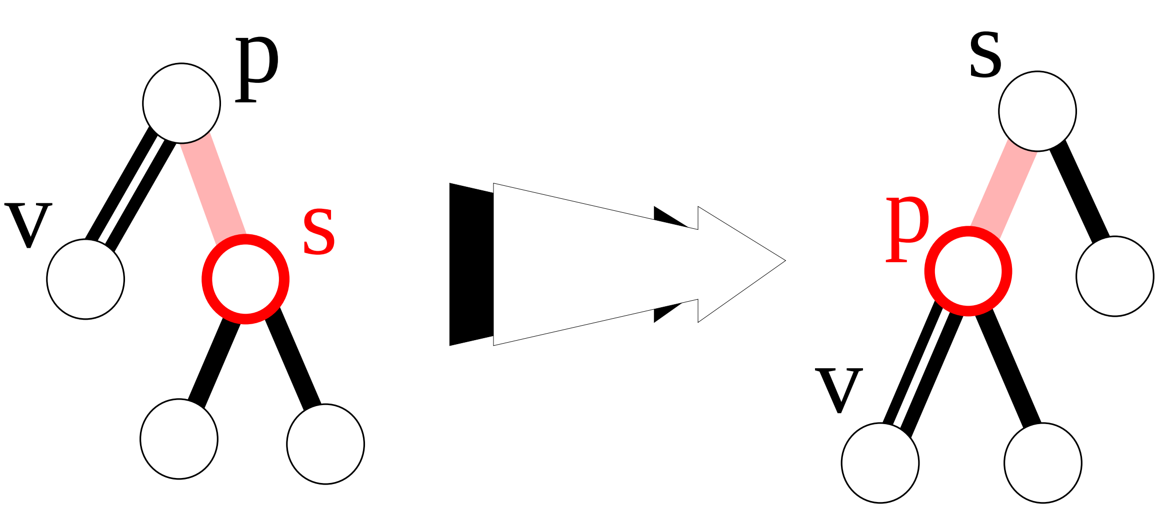


(2,4) Tree Interpretation



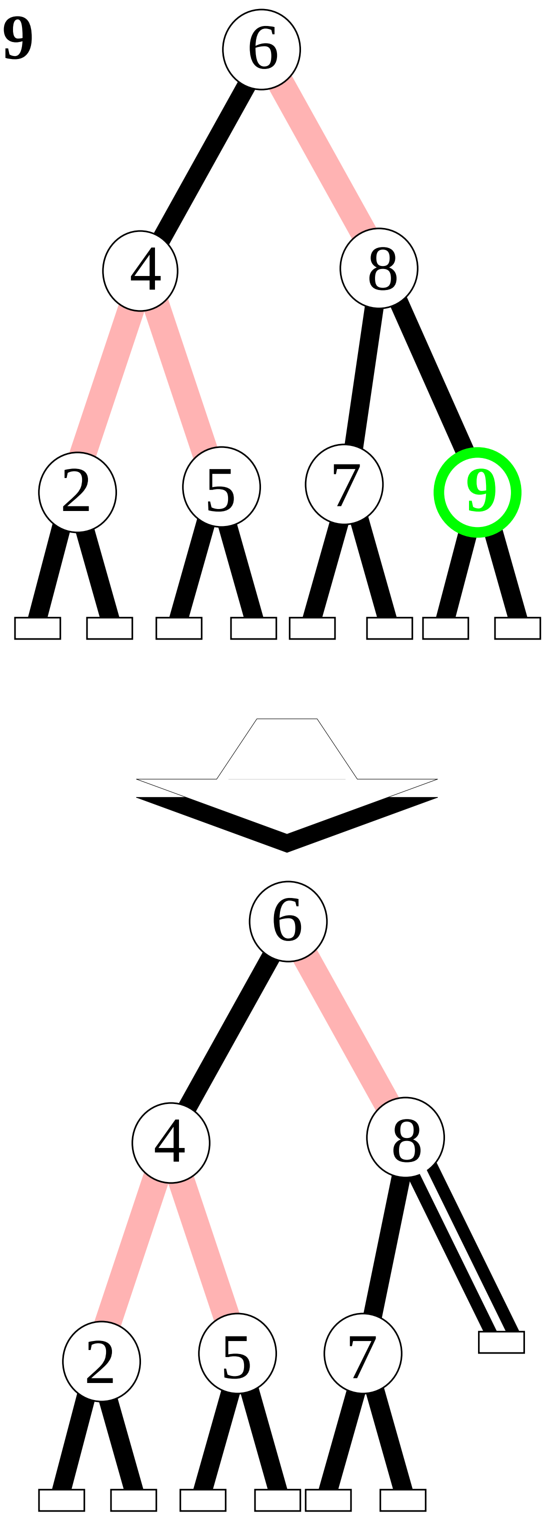
Case 3: red sibling

- If sibling is red, perform an ***adjustment***
- Now the sibling is **black** and one the of previous cases applies
- If the next case is recoloring, there is no propagation upward (parent is now red)



How About an Example?

Remove 9



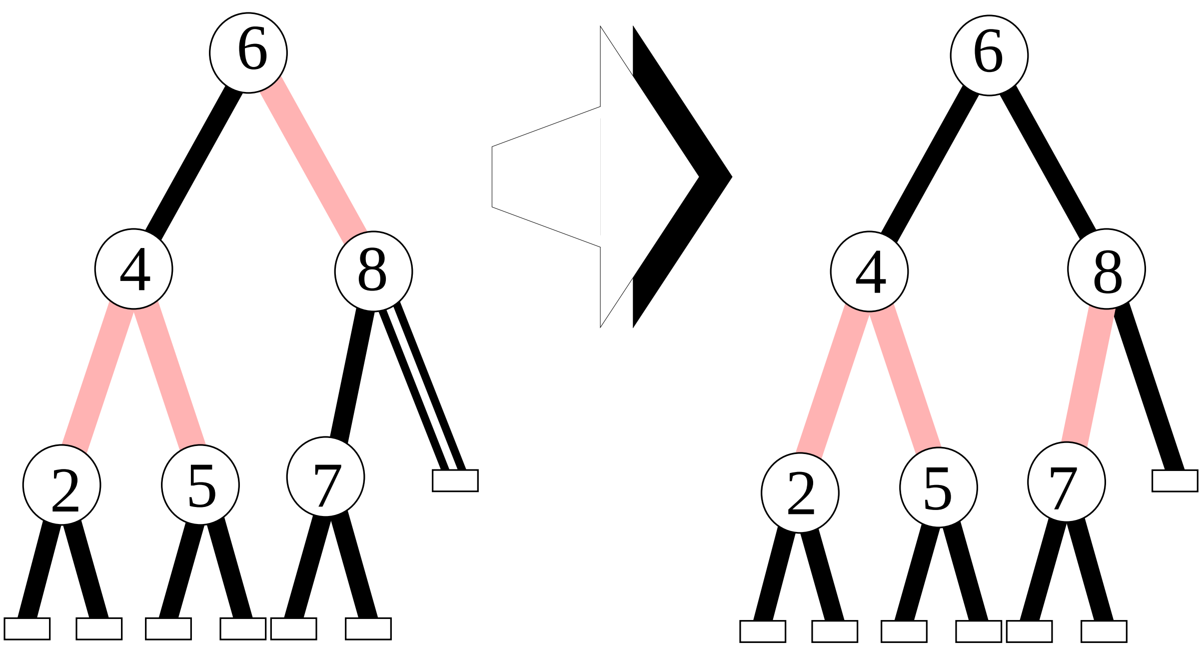
Example

What do we know?

- Sibling is black with black children

What do we do?

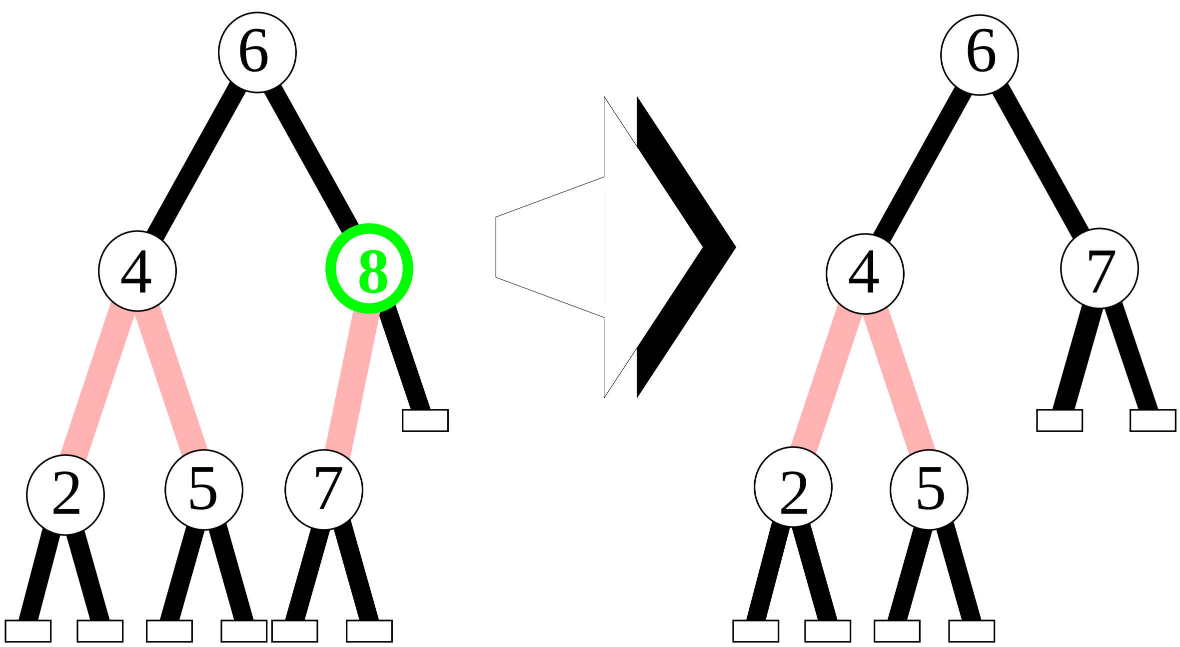
- Recoloring



Example

Delete 8

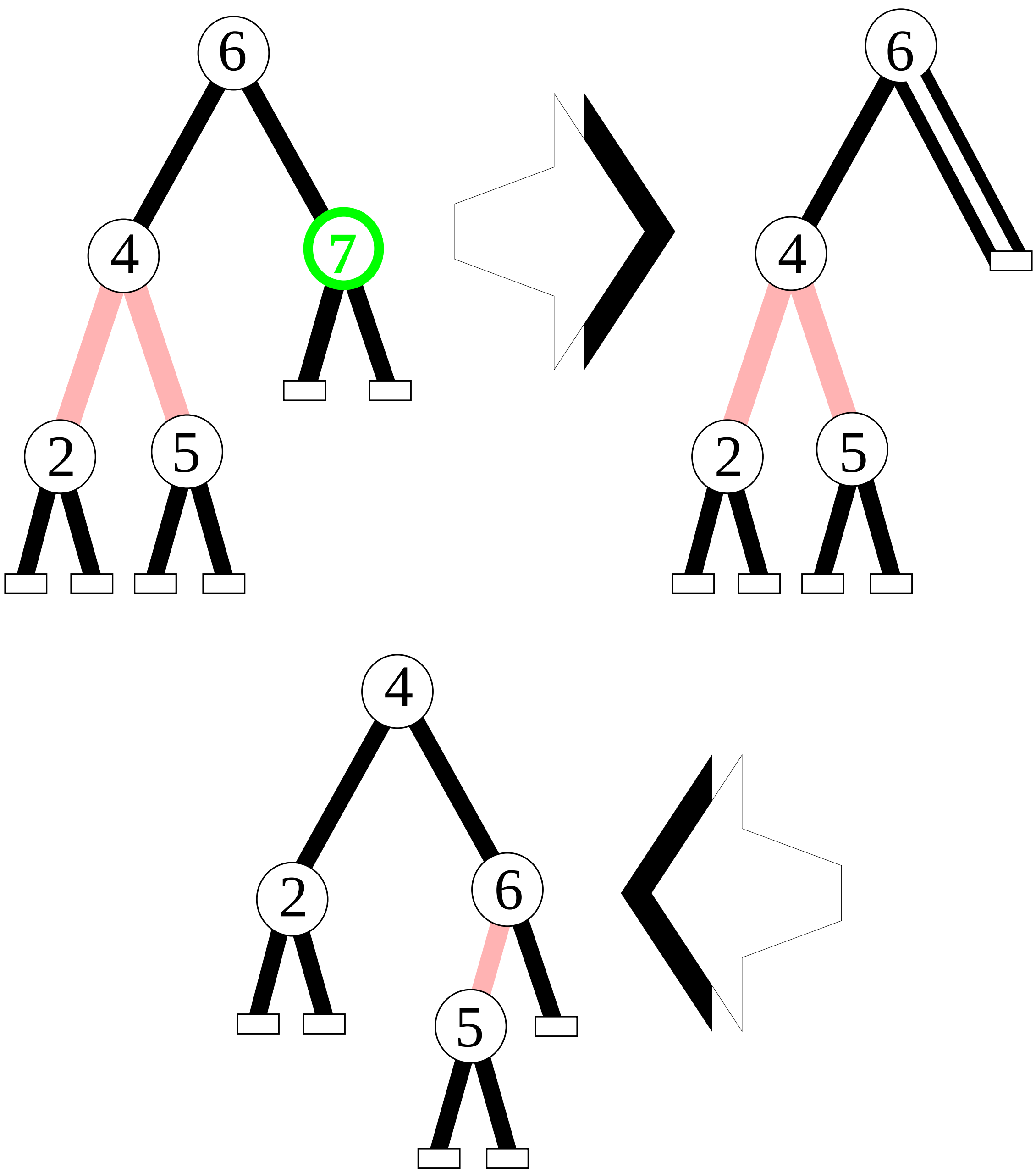
- no double black



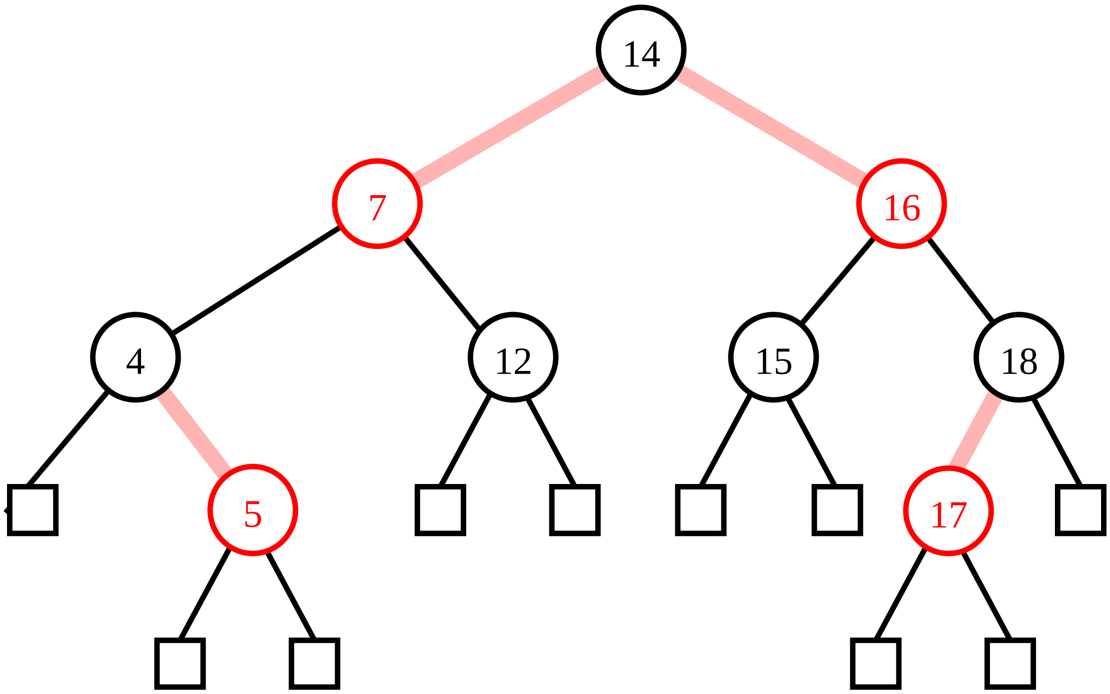
Example

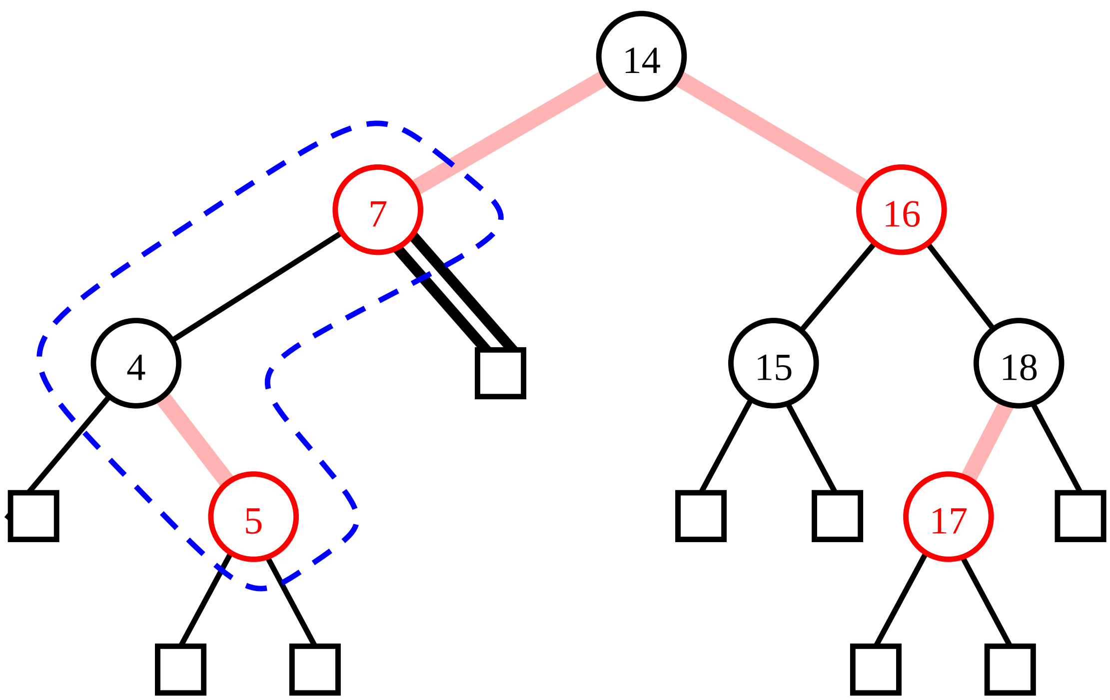
Delete 7

- Restructuring

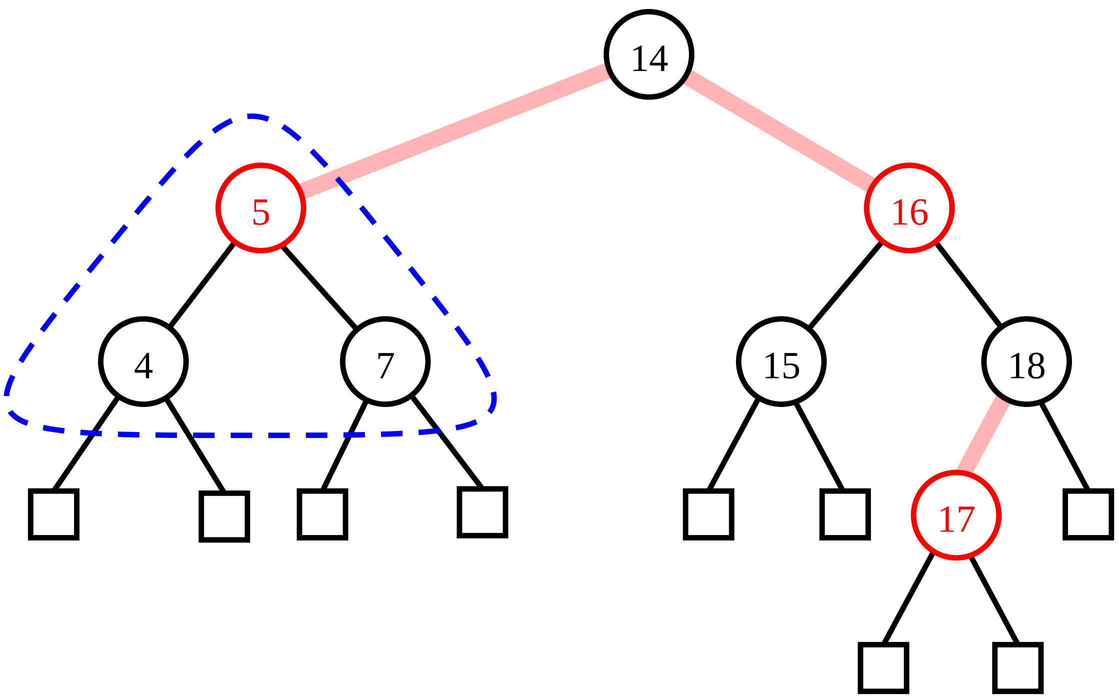


Example





Example



Summary of Red-Black Trees

- An insertion or deletion may cause a local *perturbation* (two consecutive red edges, or a **double-black** edge)
- The perturbation is either
  - *resolved locally* (restructuring), or
  - *propagated* to a higher level in the tree by recoloring (promotion or demotion)
- O(1) time for a restructuring or recoloring
- At most one restructuring per insertion, and at most two restructurings per deletion
- O(log N) recolorings
- Total time: O(log N)