Discussion Session 9: Red-Black Trees

CS 225: Data Structures
& Software Principles

Agenda

- **Red-Black Trees**
- Red-Black Tree Properties
- Operations on Red-Black Trees
  - Insert
    - Case-wise Analysis
  - Delete
    - Case-wise Analysis

By the end of this class, you

- Need to
  - Understand the properties of Red-Black Trees
  - Manually simulate Insertion and Deletion in these trees.
- Might want to be able to implement Red-Black Trees

Red-Black Trees

- Red-Black trees are BSTs in which every node has an additional **color** attribute (can be either **red** or **black**), and which satisfies 3 additional properties (more about them later).
- Imposing the additional properties ensures that the tree is approximately **balanced**.
- So, now every child pointer can point to a red child or a black child (**important:** we treat the case in which the pointer points to a NULL as a pointer to a black child).

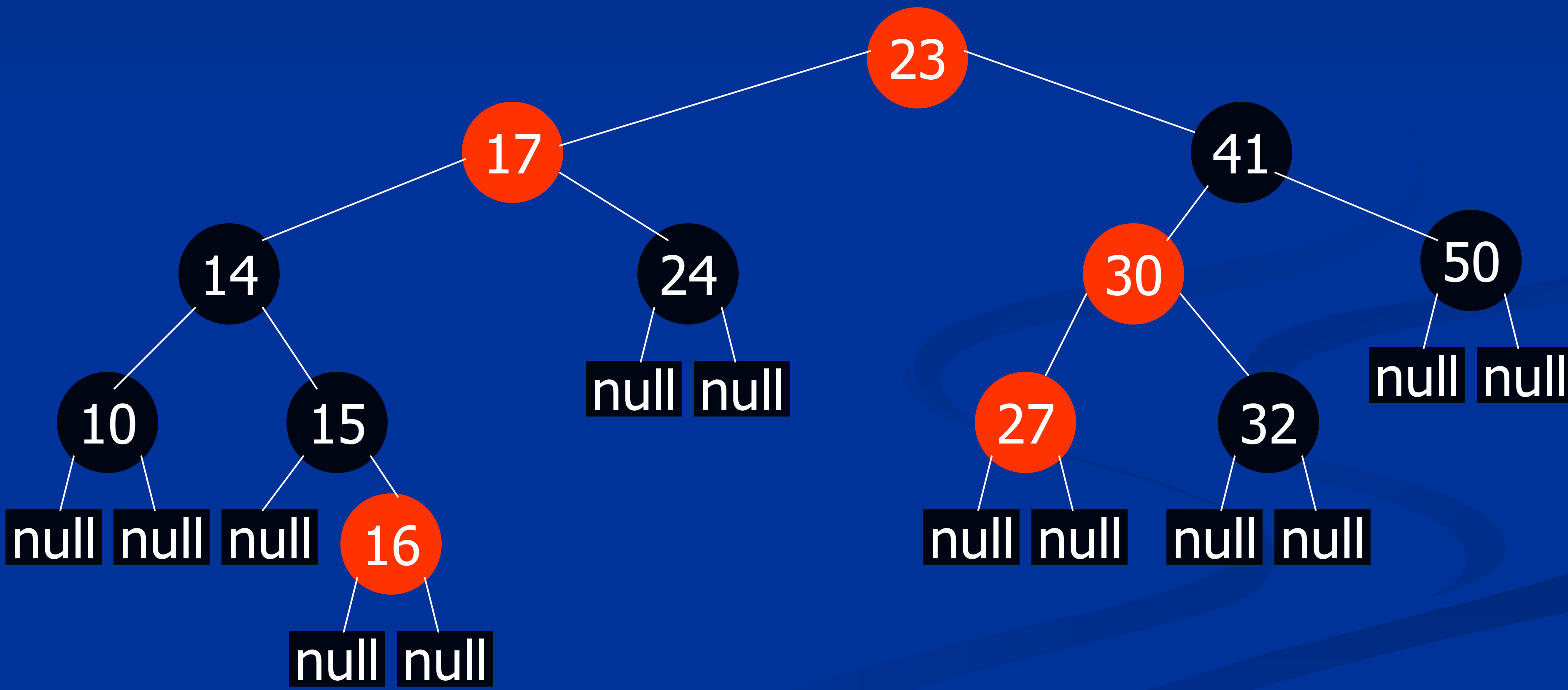
Agenda

- Red-Black Trees
- **Red-Black Tree Properties**
- Operations on Red-Black Trees
  - Insert
    - Case-wise Analysis
  - Delete
    - Case-wise Analysis

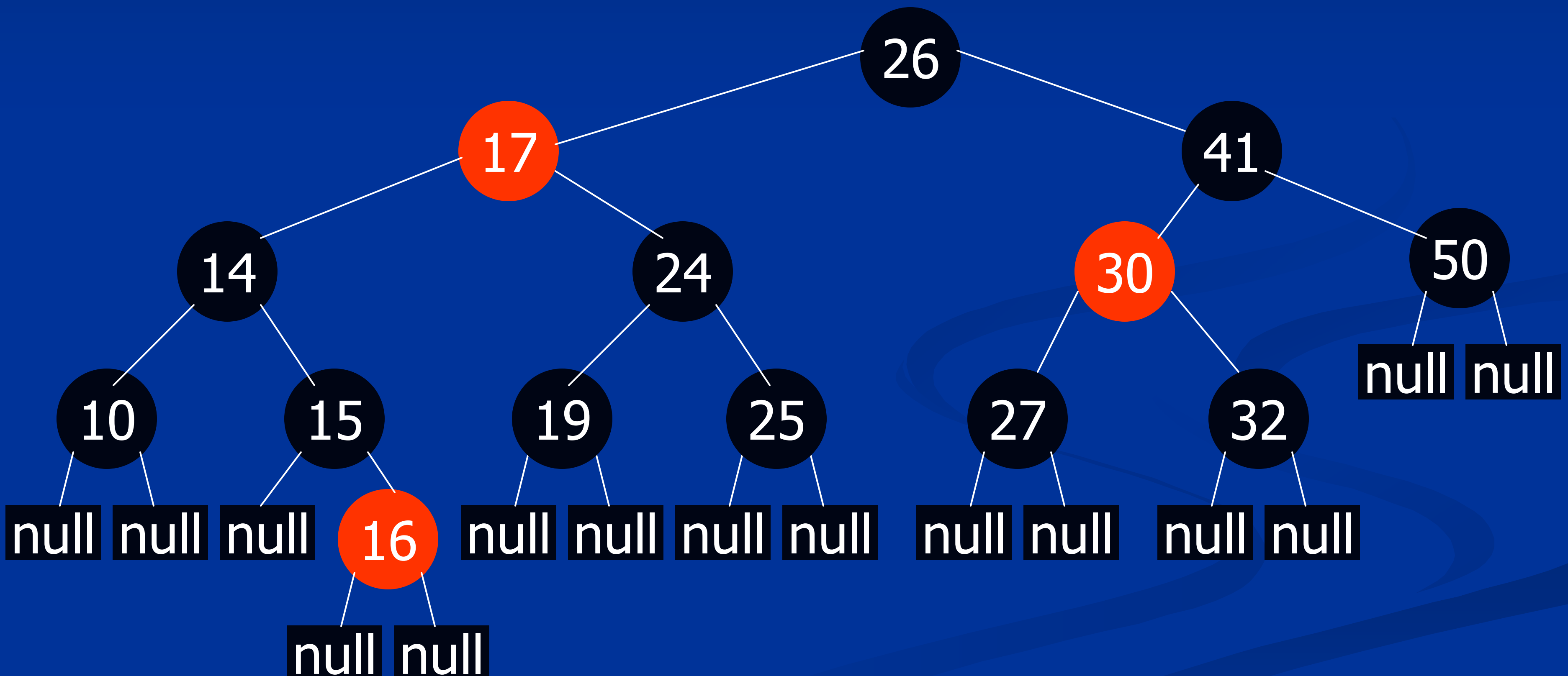
Red-Black Trees

- Red-Black Tree Properties
  - Any red node has only black children (remember that NULL pointers are considered as pointers to black children).
  - Every path from a node to a descendant leaf has the same number of black nodes.
  - The root node is always a black node.

Example: Red-Black Tree?



Example: Red-Black Tree!



Agenda

- Red-Black Trees
- Red-Black Tree Properties
- **Operations on Red-Black Trees**
  - Insert
    - Case-wise Analysis
  - Delete
    - Case-wise Analysis

Delete / Insert

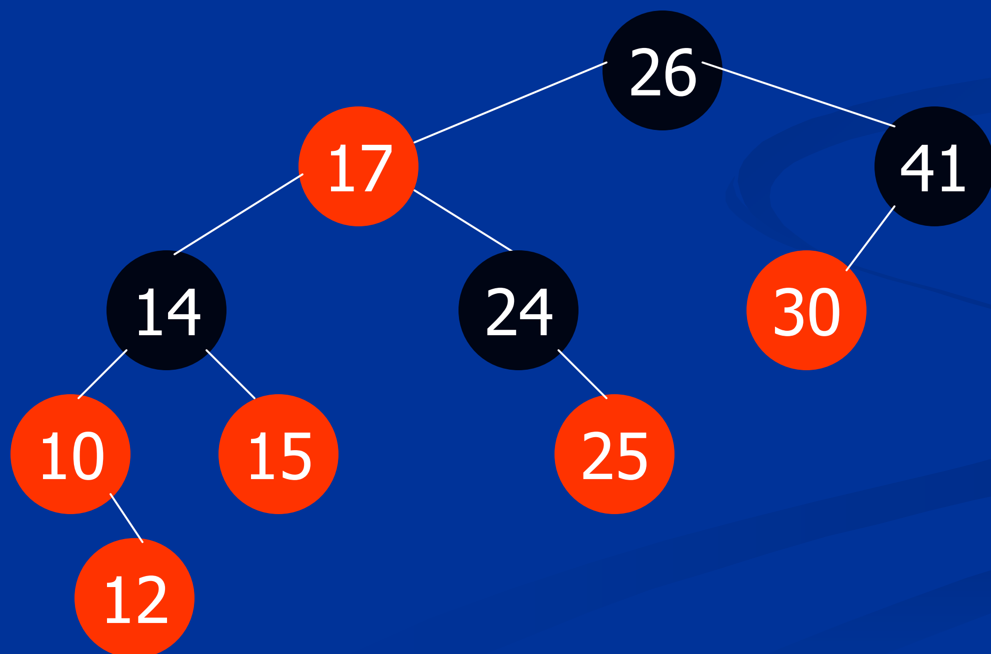
- **Insert** and **Delete** operations modify the tree and can violate the Red-Black tree properties.
  - We will start with the BST **Insert/Delete**.
  - We then may need to fix the tree to restore the properties.
  - Change colors.
  - Change structure (rotate, recall AVL trees).

Agenda

- Red-Black Trees
- Red-Black Tree Properties
- Operations on Red-Black Trees
  - **Insert**
    - Case-wise Analysis
  - Delete
    - Case-wise Analysis

Inserts

- Insert the node (12) as if we had a simple BST.
- Color the new node red. What properties are broken?



Inserts

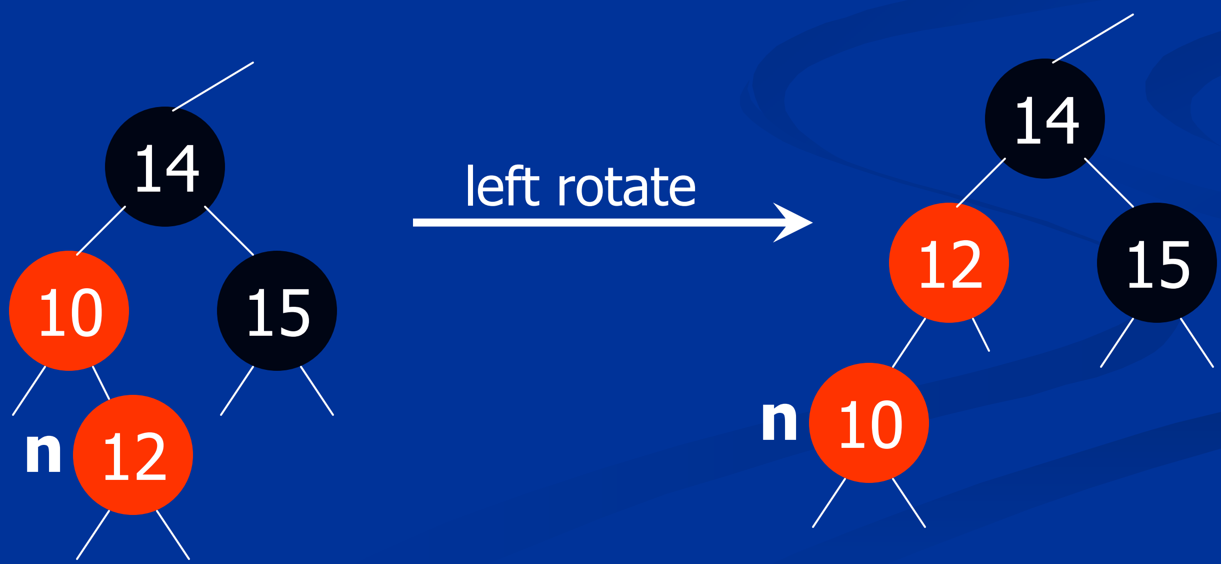
- Insert the node (12) as if we had a simple BST.
- Color the new node red. What properties are broken?
  - Red node 10 does not have two black children
- We then go up the tree and either:
  - Move the violation up while making sure property 2 holds, or perform some rotations and stop.
  - As we go up we can encounter six possibilities (but three are symmetric to the other three).

Inserts: Case 1.

- If both the node being inserted (**n**) and its uncle are red.
  - Grandparent of n must be black.
  - Color both the parent and the uncle black, and grandparent red.
  - Keep going up.

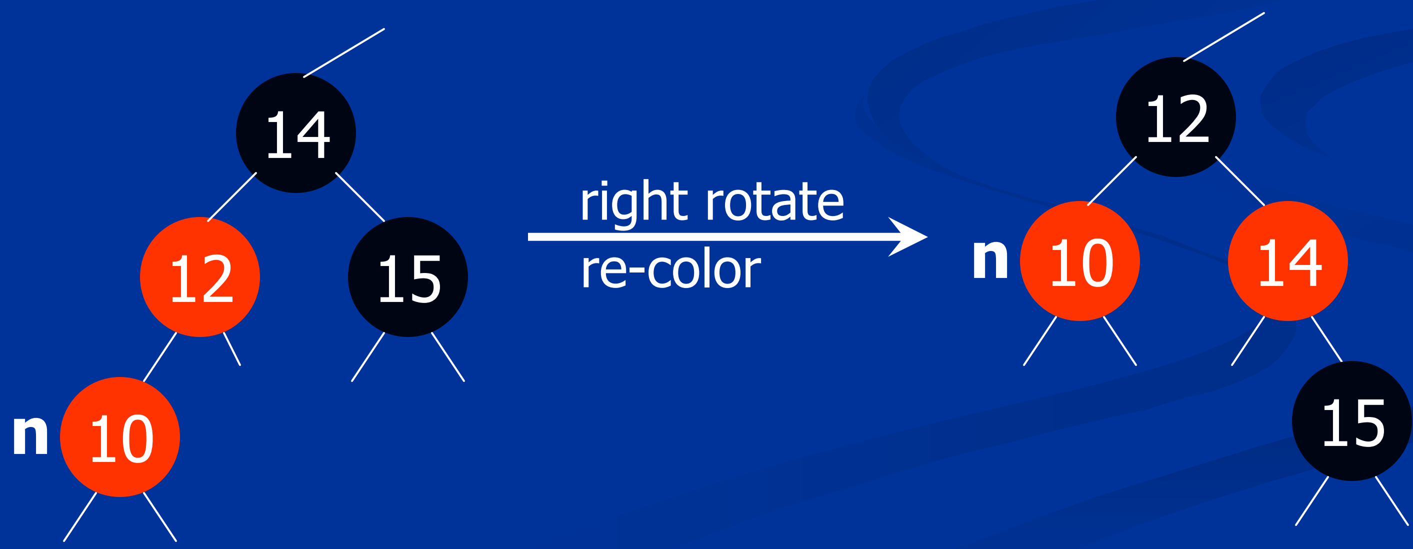
Inserts: Case 2.

- If **n**'s uncle is black and **n** is a *right* child.
  - Left rotation.
  - None of the properties are affected.
  - Reduces to case 3.

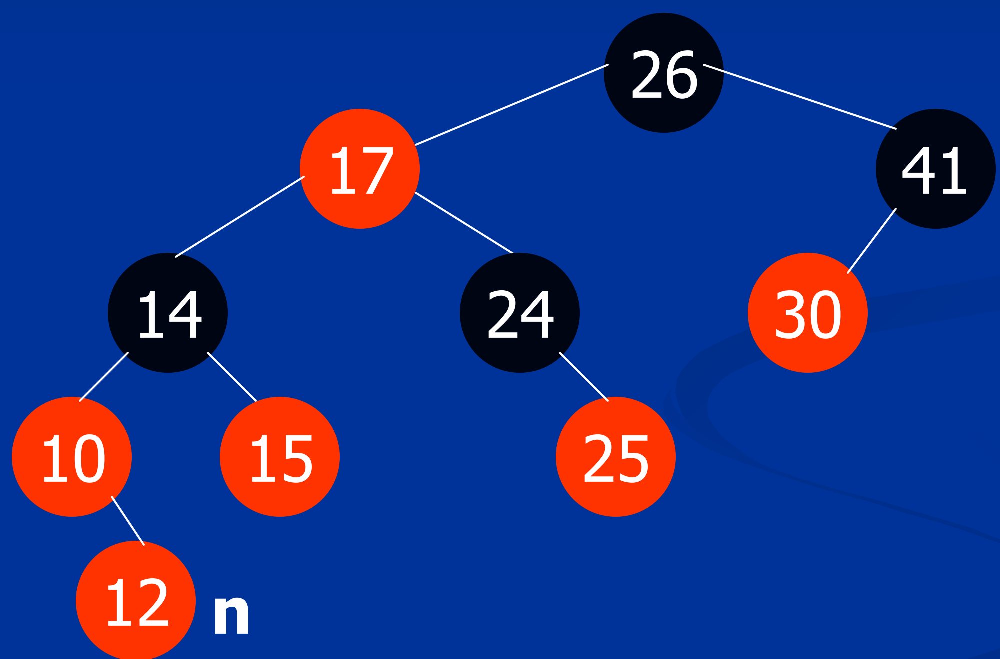


Inserts: Case 3.

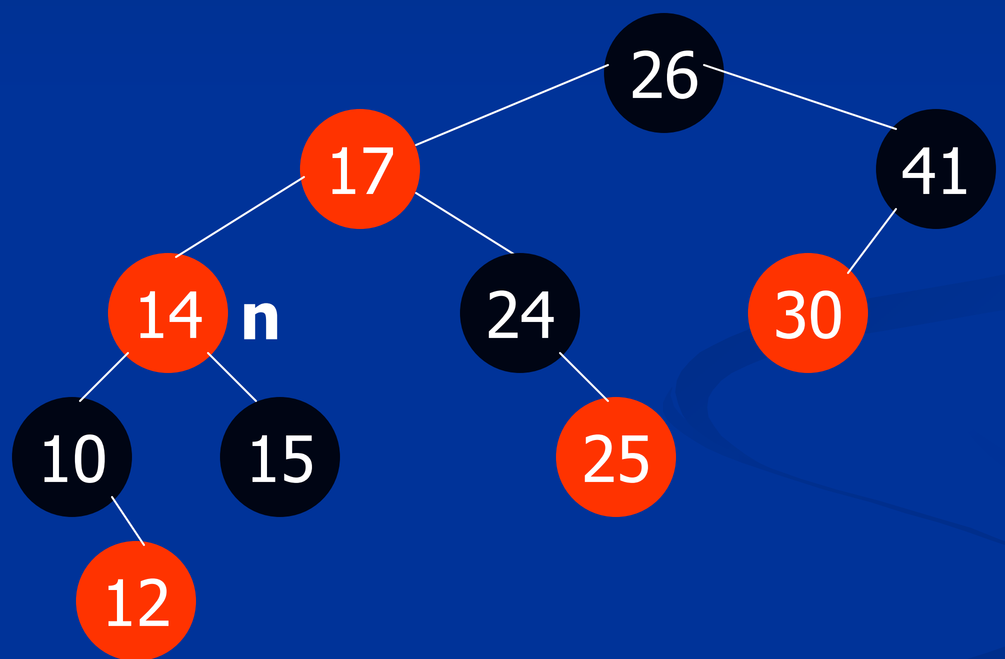
- If **n**'s uncle is black and **n** is a *left* child.
  - Right rotate and re-color.
  - The parent of **n** is black and we are done.



Inserts: Example [Case 1]

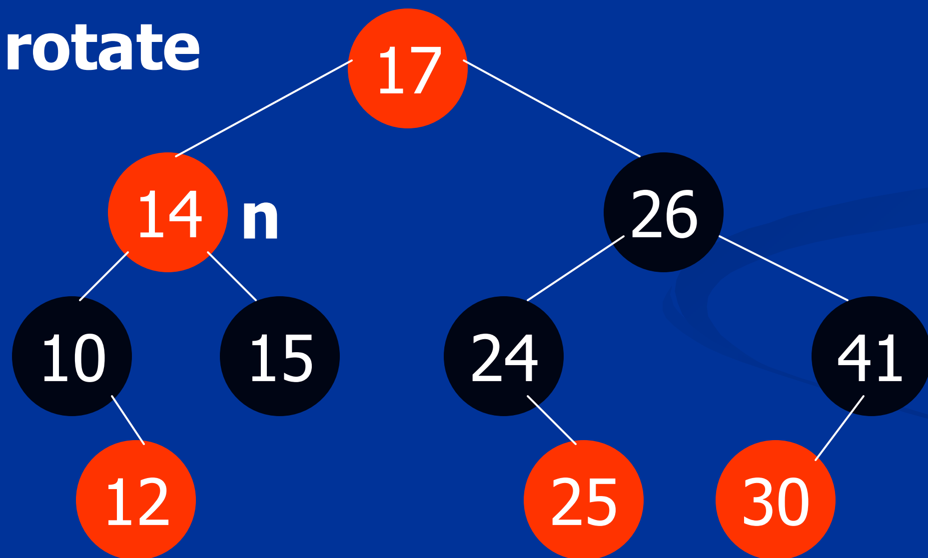


Inserts: Example [Case 3]



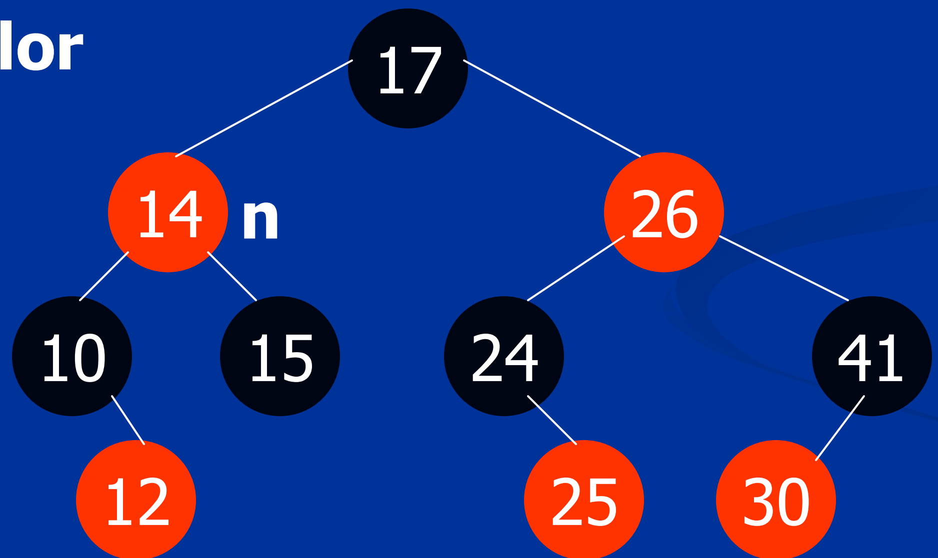
Inserts: Example [Case 3]

Right rotate



Inserts: Example [Fixed]

Re-color



Agenda

- Red-Black Trees
- Red-Black Tree Properties
- Operations on Red-Black Trees
  - Insert
    - Case-wise Analysis
  - **Delete**
    - Case-wise Analysis

Delete

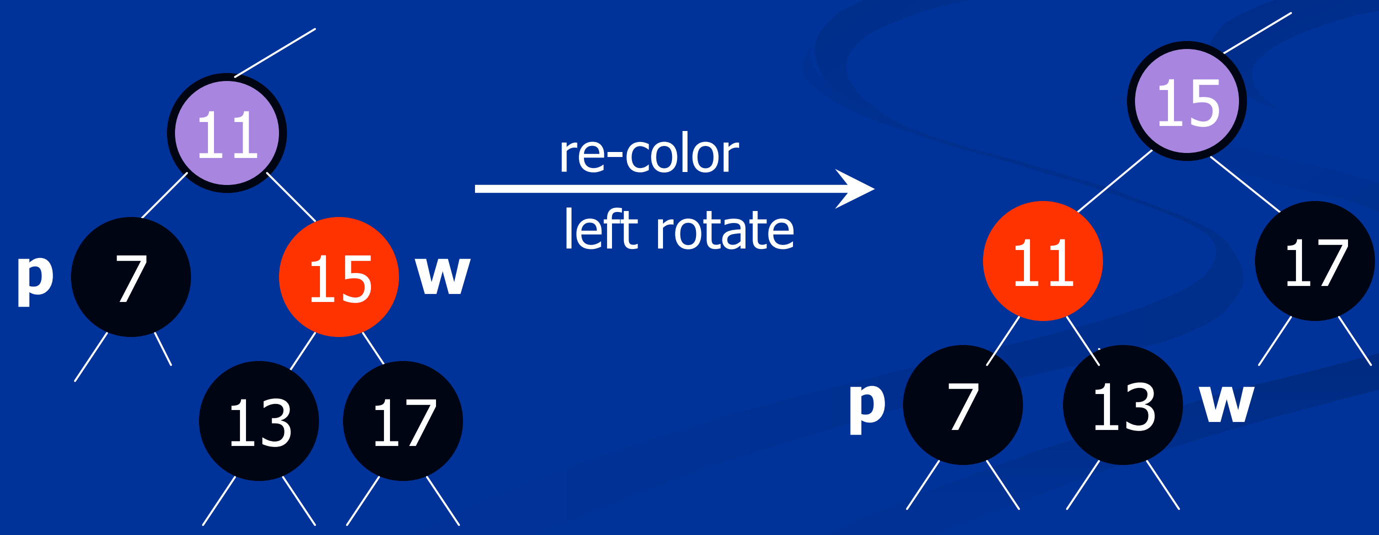
- Delete the node as if we had a simple BST.
  - Replace the deleted node **n** with its in-order successor or predecessor **k**.
- If the node **k** that was spliced out was red than no properties are violated (e.g. **n** = 12, **k** = 16).
- If **k** is black, any path that previously contained **k** has one fewer black nodes (e.g. **n** = 24).

Delete

- Delete the node as if we had a simple BST.
  - Replace the deleted node **n** with its in-order successor or predecessor **k**.
- If the node **k** that was spliced out was red than no properties are violated (e.g. **n** = 12, **k** = 16).
- If **k** is black, any path that previously contained **k** has one fewer black nodes (e.g. **n** = 24).
  - **k** has at most one child (**p**), we push its blackness to its child or null node.
- If the **k**'s child **p** was already black, we fix the tree by moving the "extra black" up the tree. Four cases are possible.

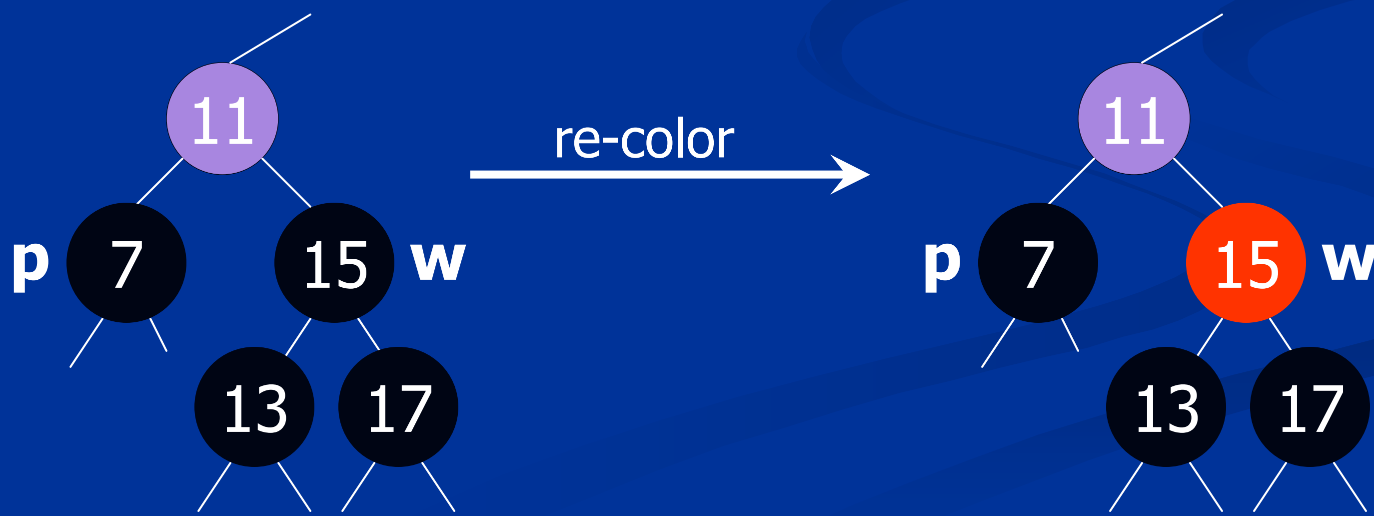
Delete : Case 1

- If the sibling **w** of **p** is red
  - Left rotate and re-color.
  - New sibling of p is black, Case 1 reduces to case 2,3, or 4.



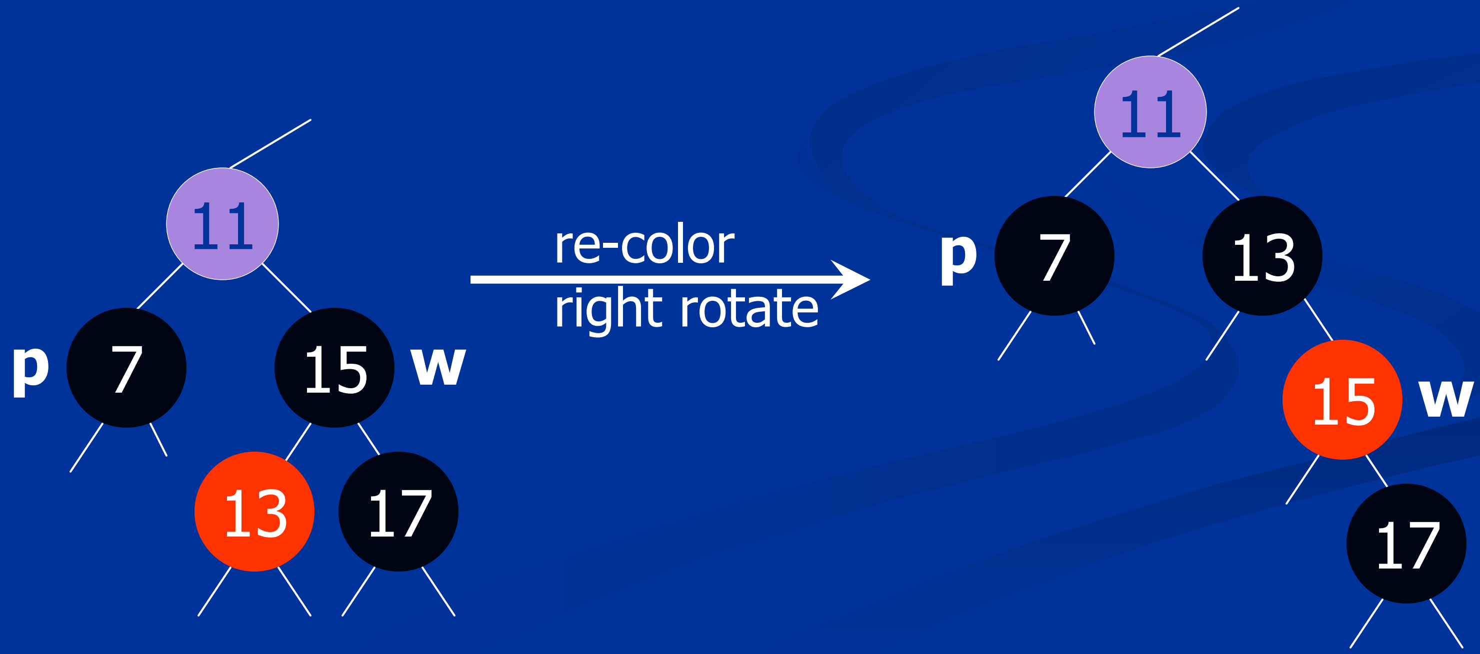
Delete: Case 2

- If the sibling **w** is black and both children of **w** are black
  - Remove 'one black' from **p** and color **w** red.
  - Push the problem up the tree



Delete: Case 3

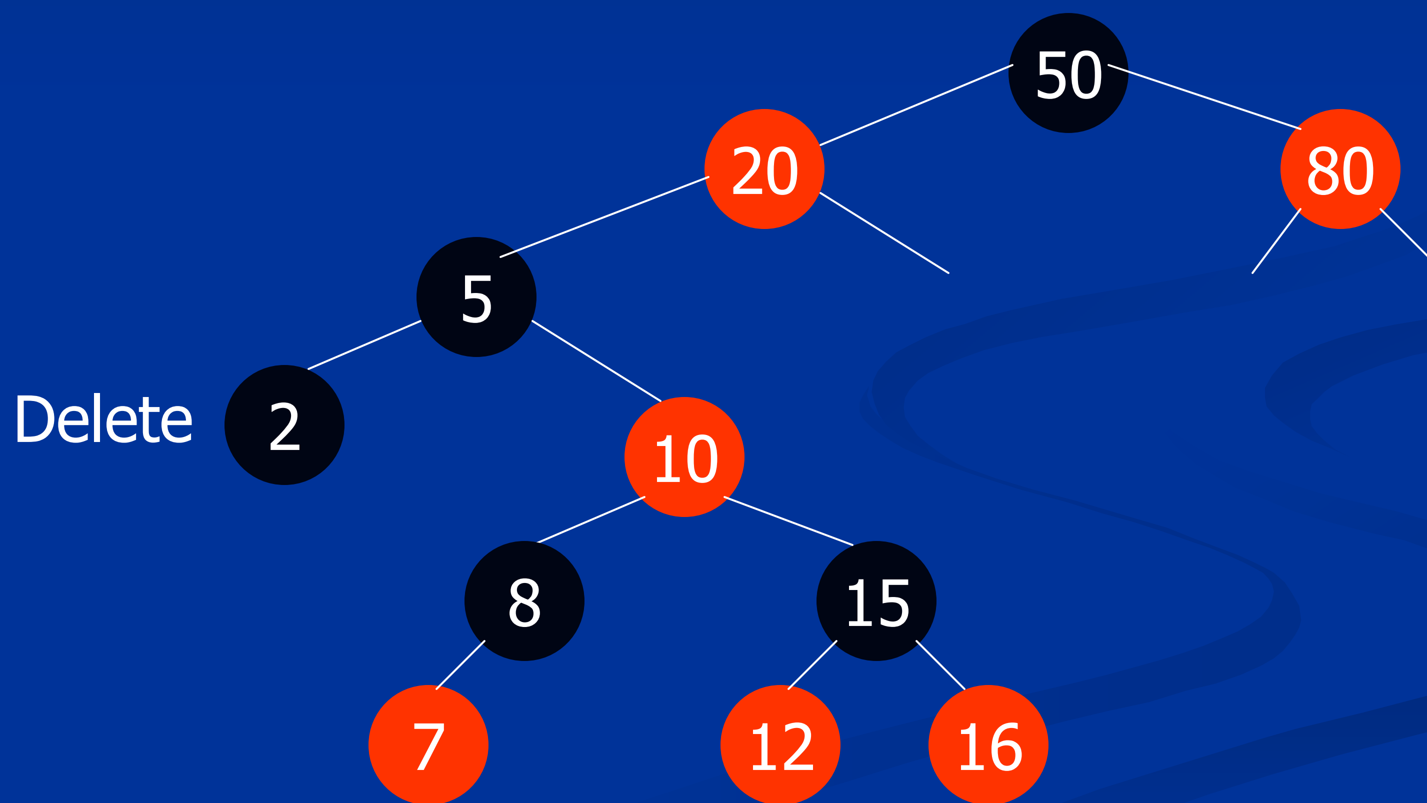
- If the sibling **w** is black and **w**'s left child is red, and its right child is black.
  - Switch color of w and its left child and right rotate.
  - Reduces to Case 4.



Delete: Case 4

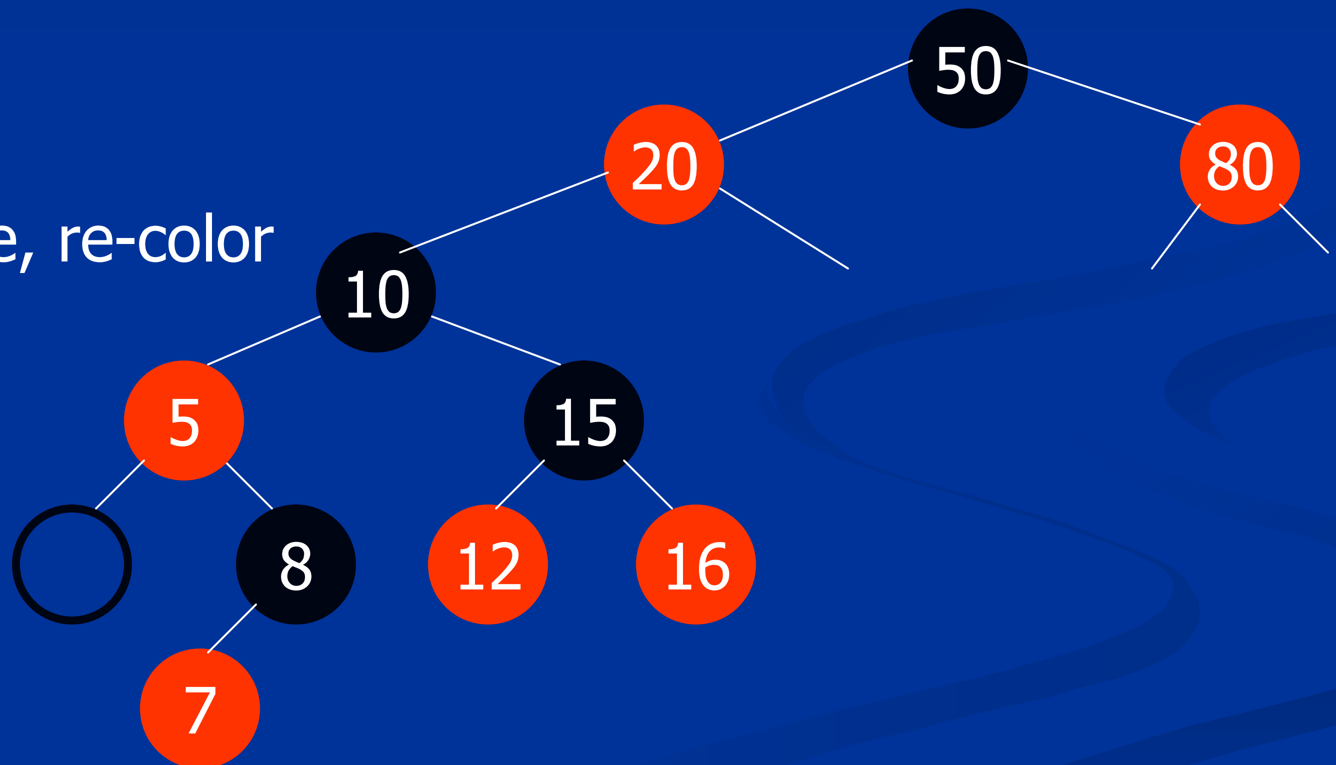
- If the sibling **w** is black and **w**'s right child is red:
  - Left rotate and re-color.
  - Done.

Deletes: Example [Case 1]

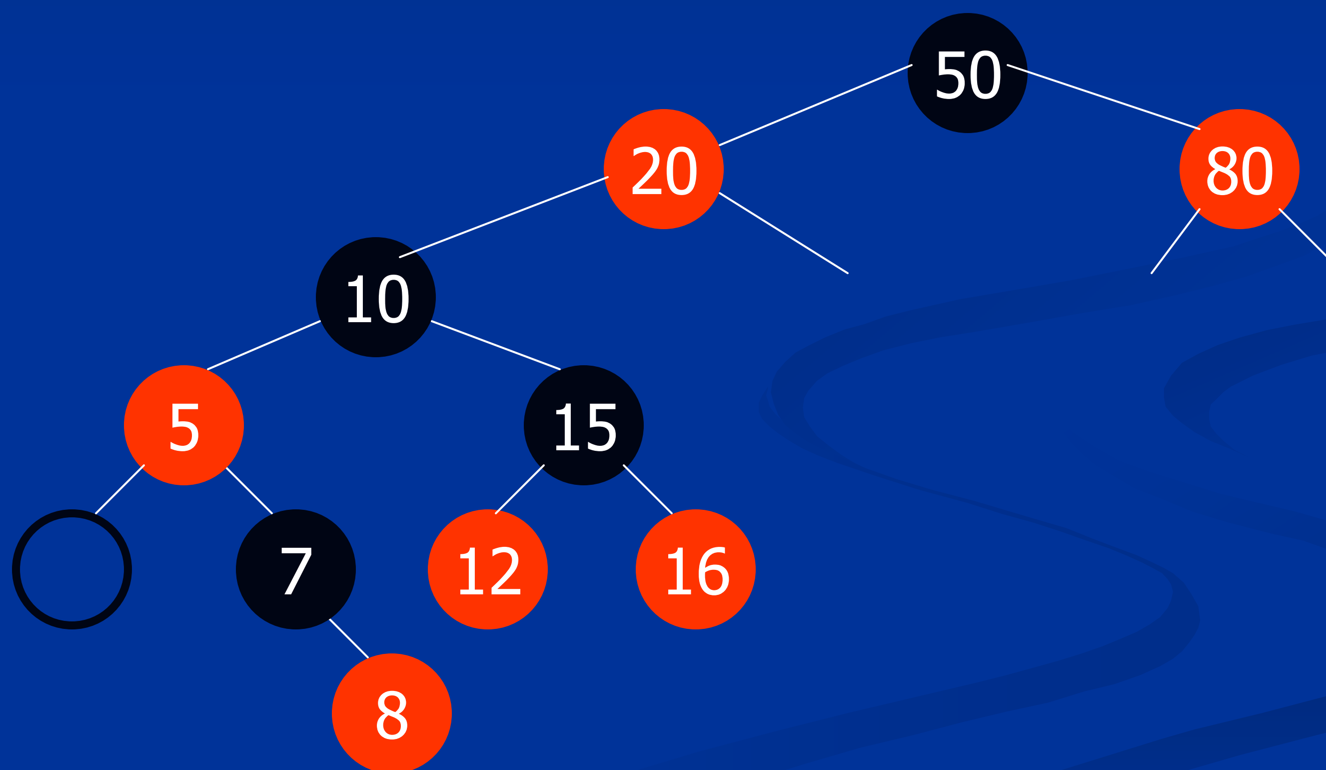


Deletes: Example [Case 3]

Right rotate, re-color



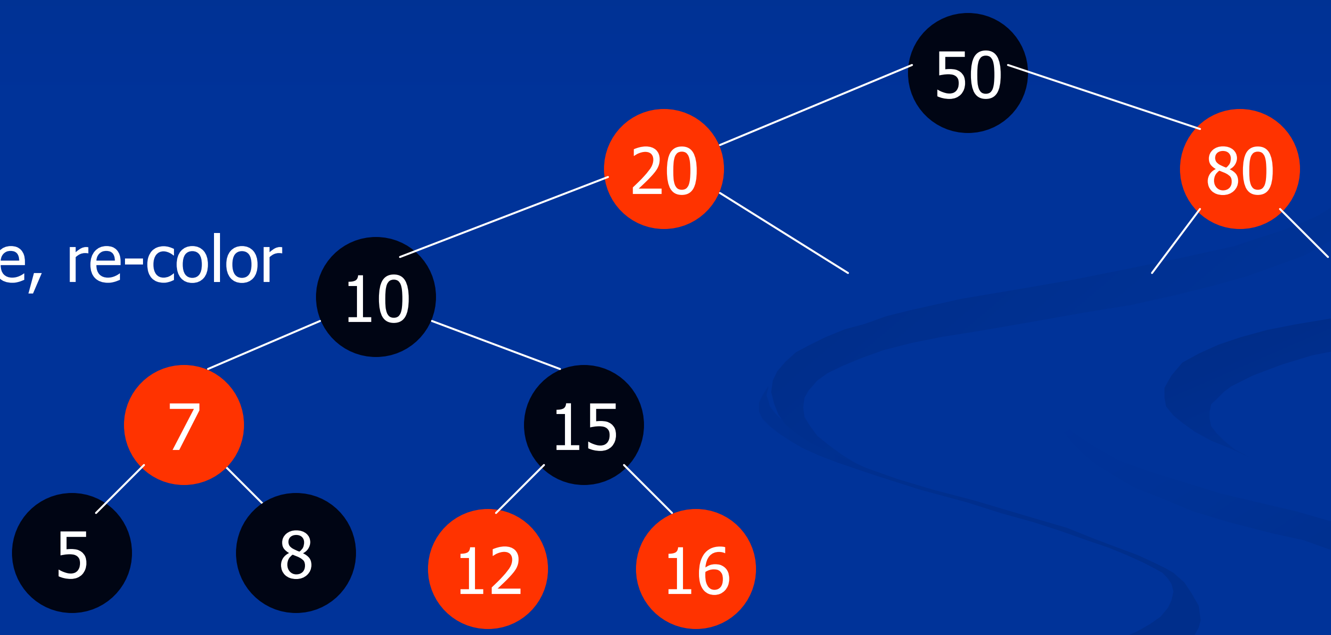
Deletes: Example [Case 4]



Right rotate, re-color

Deletes: Example [Fixed]

Left rotate, re-color



Red-Black Tree Applets

- http://www.ece.uc.edu/~franco/C321/html/RedBlack/redblack.html

- http://reptar.uta.edu/NOTES5311/REDBLACK/RedBlack.html

Summary

- Red-Black Trees
- Red-Black Tree Properties
- Operations on Red-Black Trees
  - Insert
    - Case-wise Analysis
  - Delete
    - Case-wise Analysis