[SWE2015-41] Introduction to Data Structures (자료구조개론)

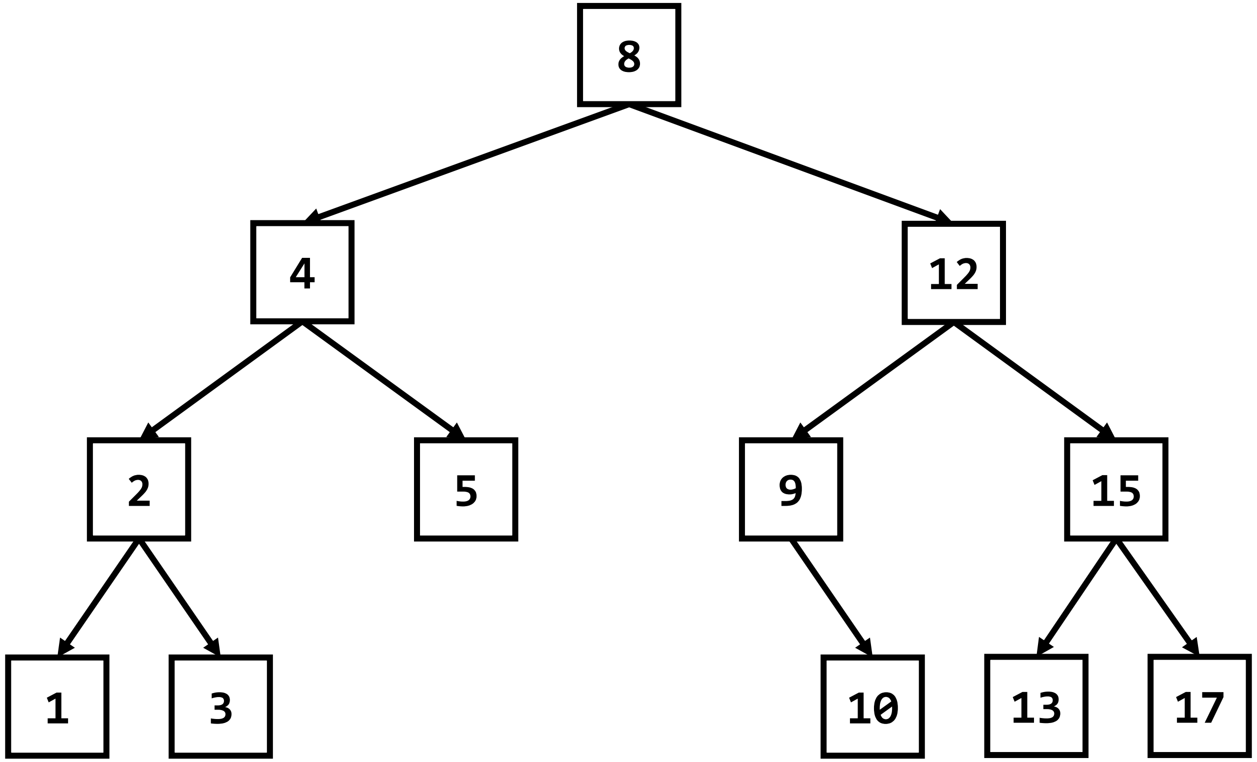
Red-Black Trees

Department of Computer Science and Engineering

Instructor: Hankook Lee (이한국)

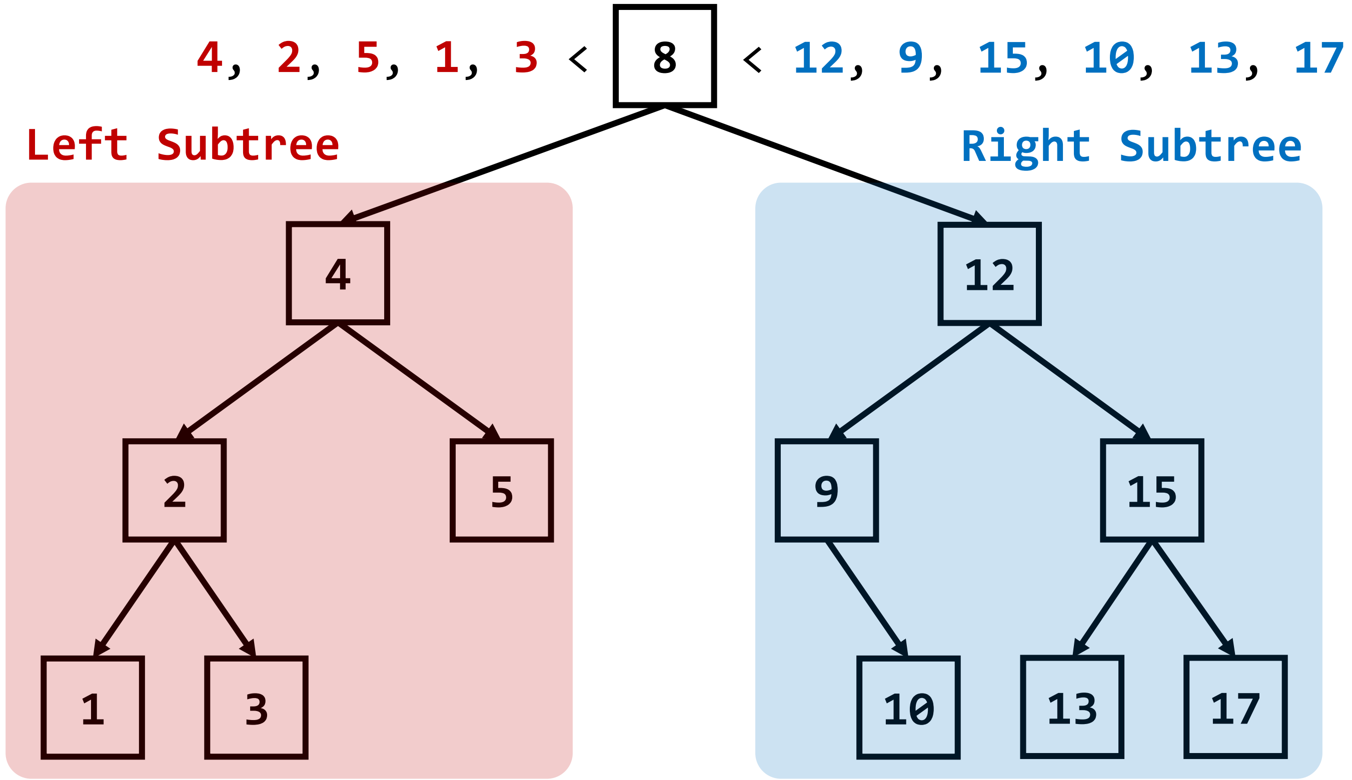
(Recap) Binary Search Trees (BSTs)

- **Binary Search Tree (BST)** satisfies the following conditions:
  1. Any two nodes **A** and **B** are comparable: **`A < B`**, **`A > B`**, or **`A == B`**
     - E.g., you can compare numbers numerically or strings in the alphabetical/dictionary order
     - Such a comparable value of a node is called **KEY** value



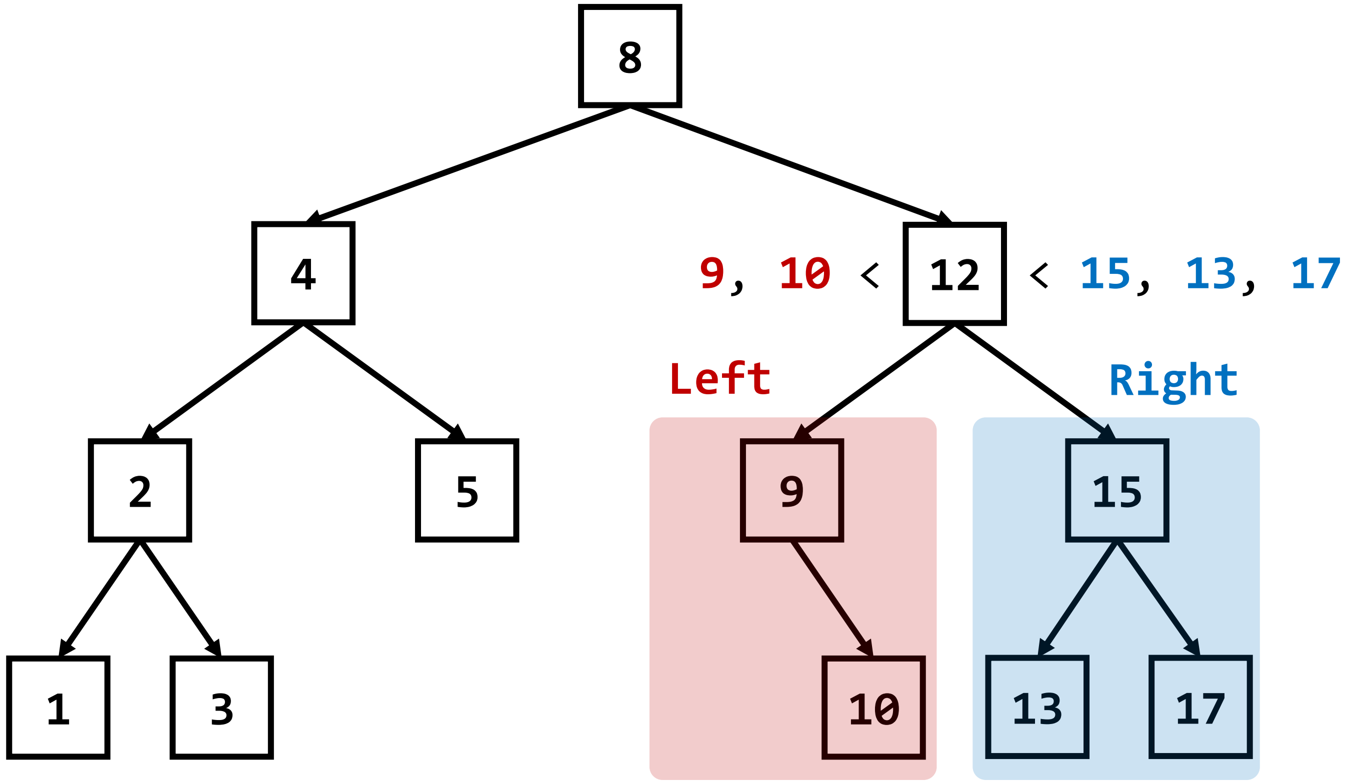
(Recap) Binary Search Trees (BSTs)

- **Binary Search Tree (BST)** satisfies the following conditions:
  2. For any node **X**, all nodes in its `left subtree` are less than **X**
  3. For any node **X**, all nodes in its `right subtree` are greater than **X**



(Recap) Binary Search Trees (BSTs)

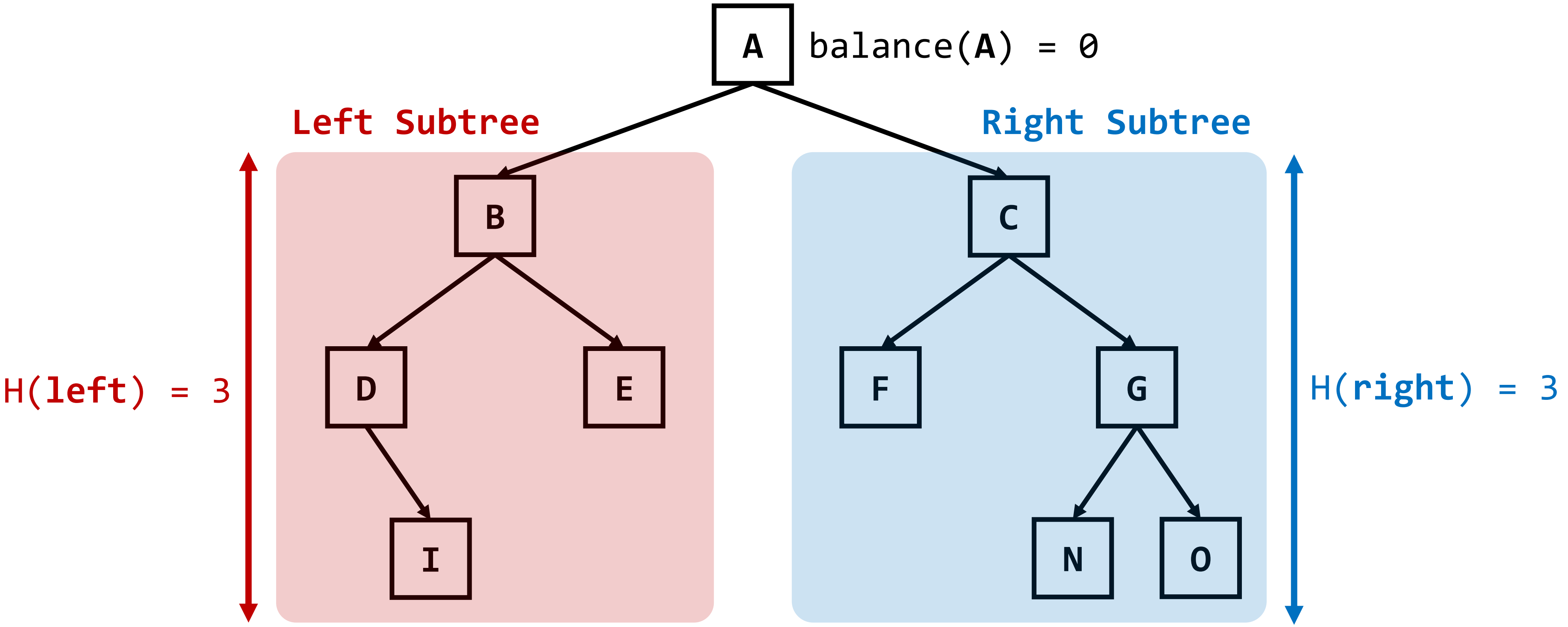
- **Binary Search Tree (BST)** satisfies the following conditions:
  2. For any node **X**, all nodes in its `left subtree` are less than **X**
  3. For any node **X**, all nodes in its `right subtree` are greater than **X**



(Recap) Balanced Binary Trees

- The **balance factor** of a node **X** in a binary tree is defined by

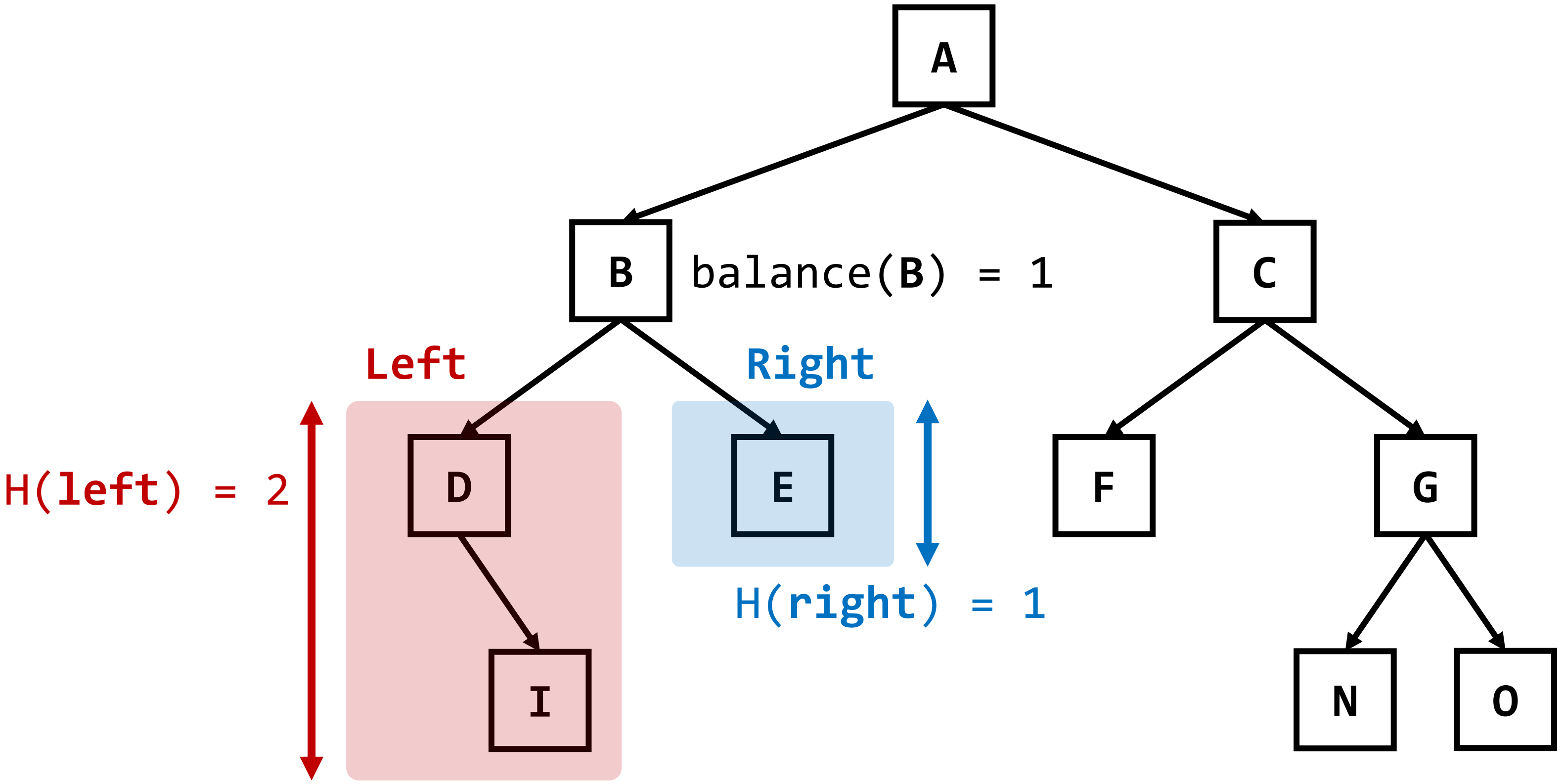
`balance(`**`X`**`) = height(`**`left subtree`**`) – height(`**`right subtree`**`)`



(Recap) Balanced Binary Trees

- The **balance factor** of a node **X** in a binary tree is defined by

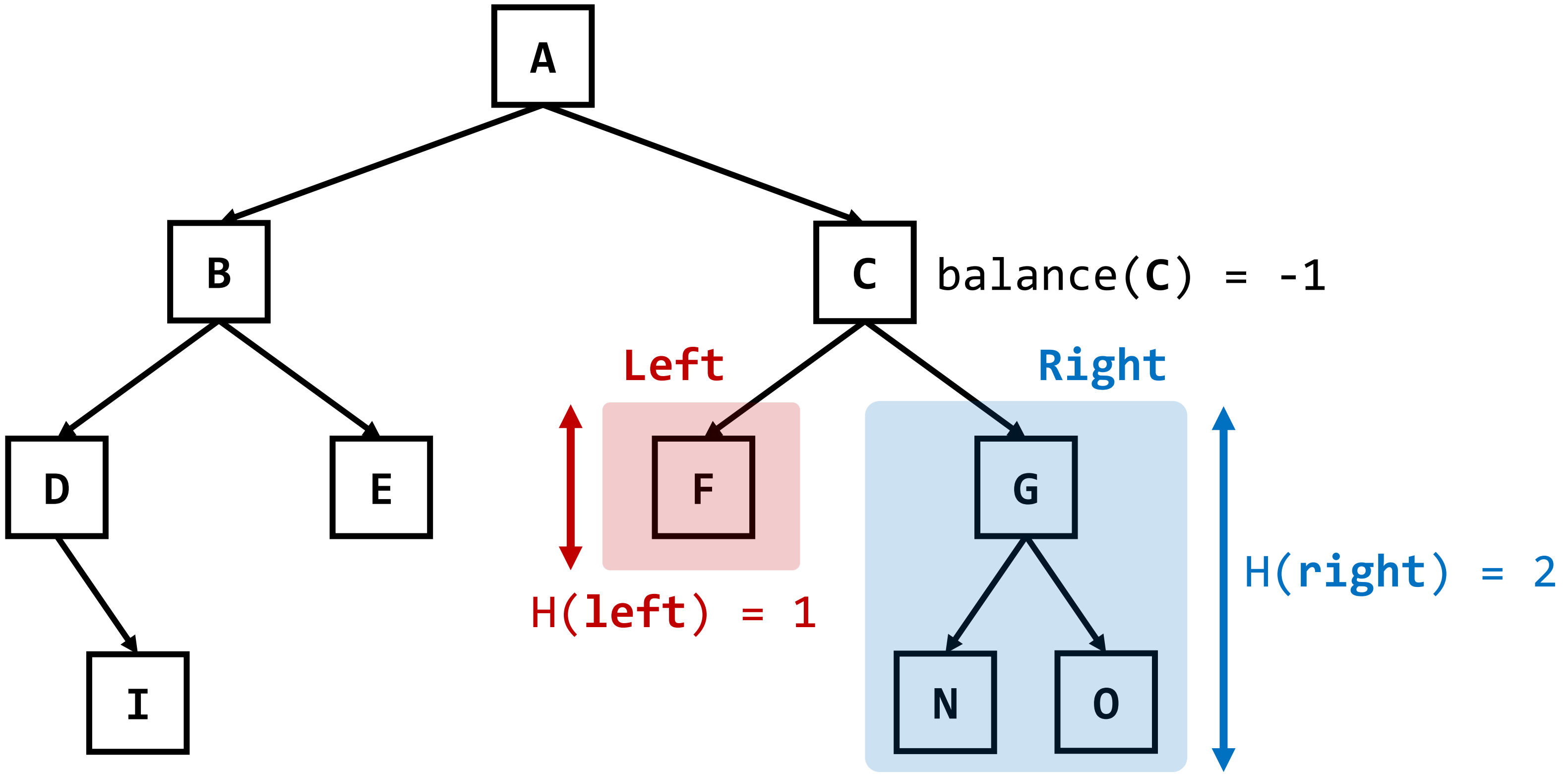
  balance(**X**) = height(**left subtree**) – height(**right subtree**)



(Recap) Balanced Binary Trees

- The **balance factor** of a node **X** in a binary tree is defined by

`balance(X) = height(left subtree) – height(right subtree)`

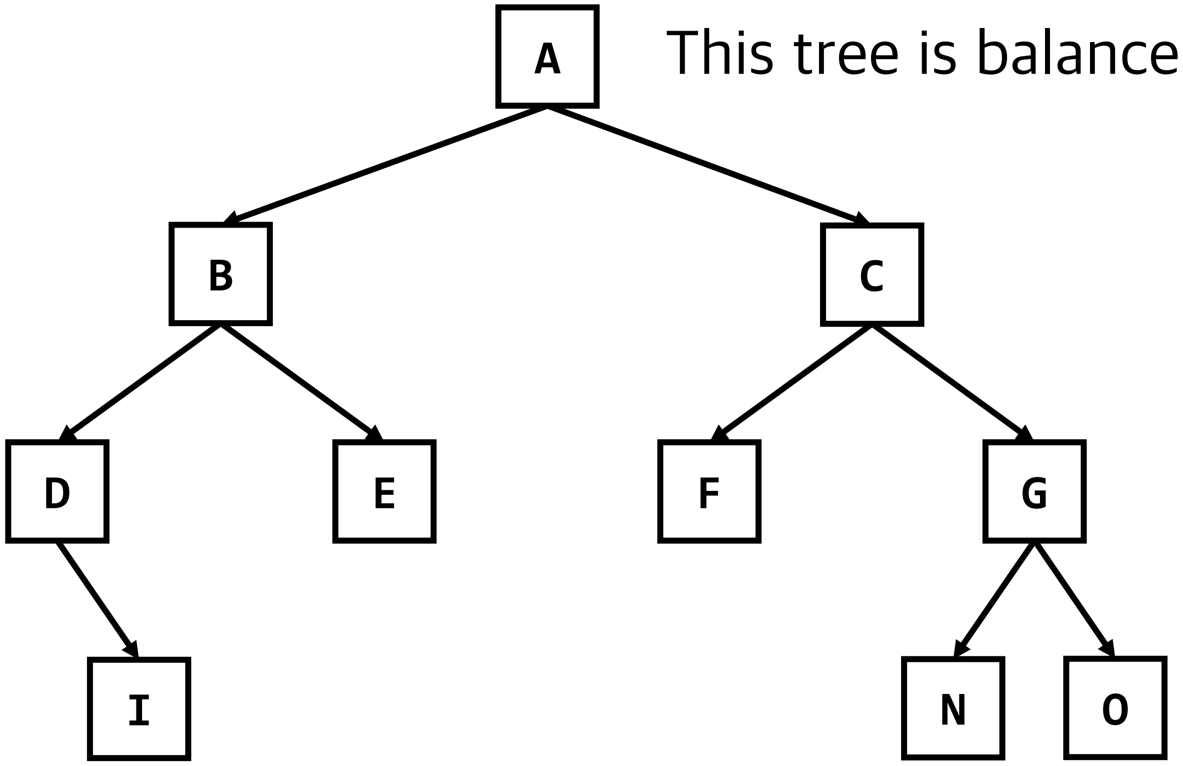


(Recap) Balanced Binary Trees

- The **balance factor** of a node **`X`** in a binary tree is defined by

  `balance(X) = height(left subtree) – height(right subtree)`

- A binary tree T is **balanced** if |`balance(X)`| ≤ 1 for any node **`X`**



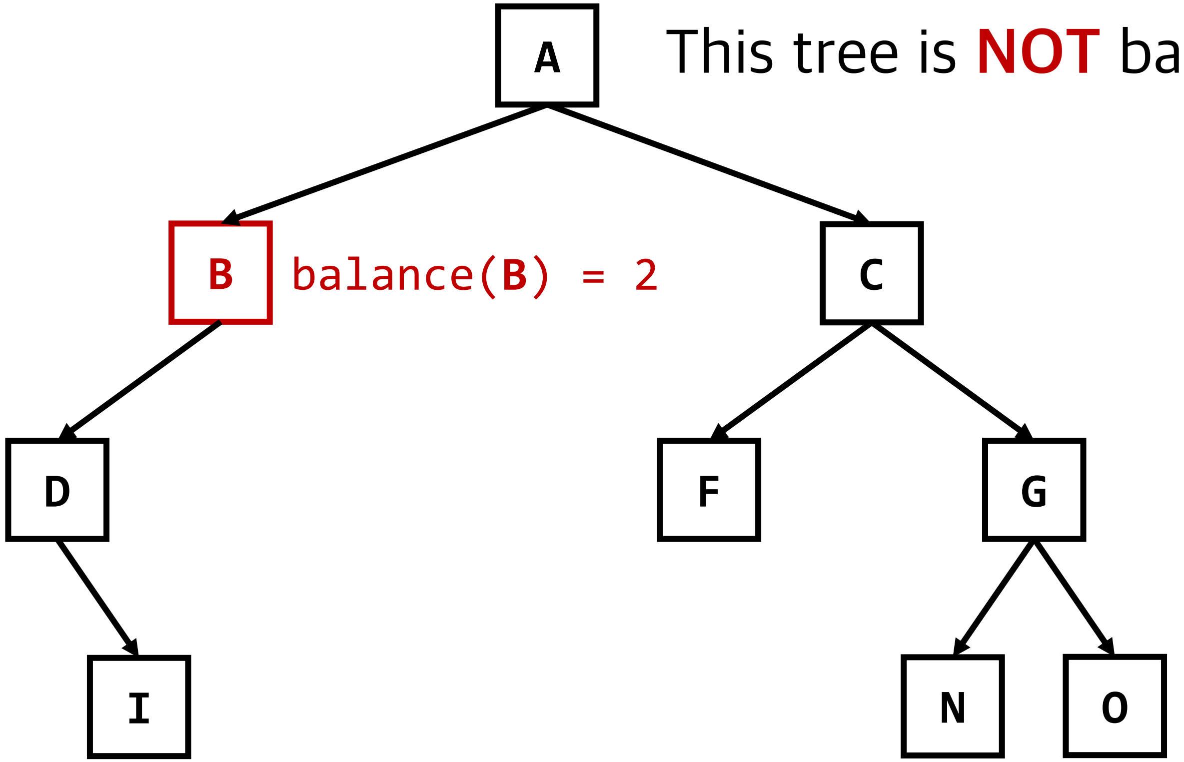
This tree is balanced

(Recap) Balanced Binary Trees

- The **balance factor** of a node **X** in a binary tree is defined by

  `balance(X) = height(left subtree) – height(right subtree)`

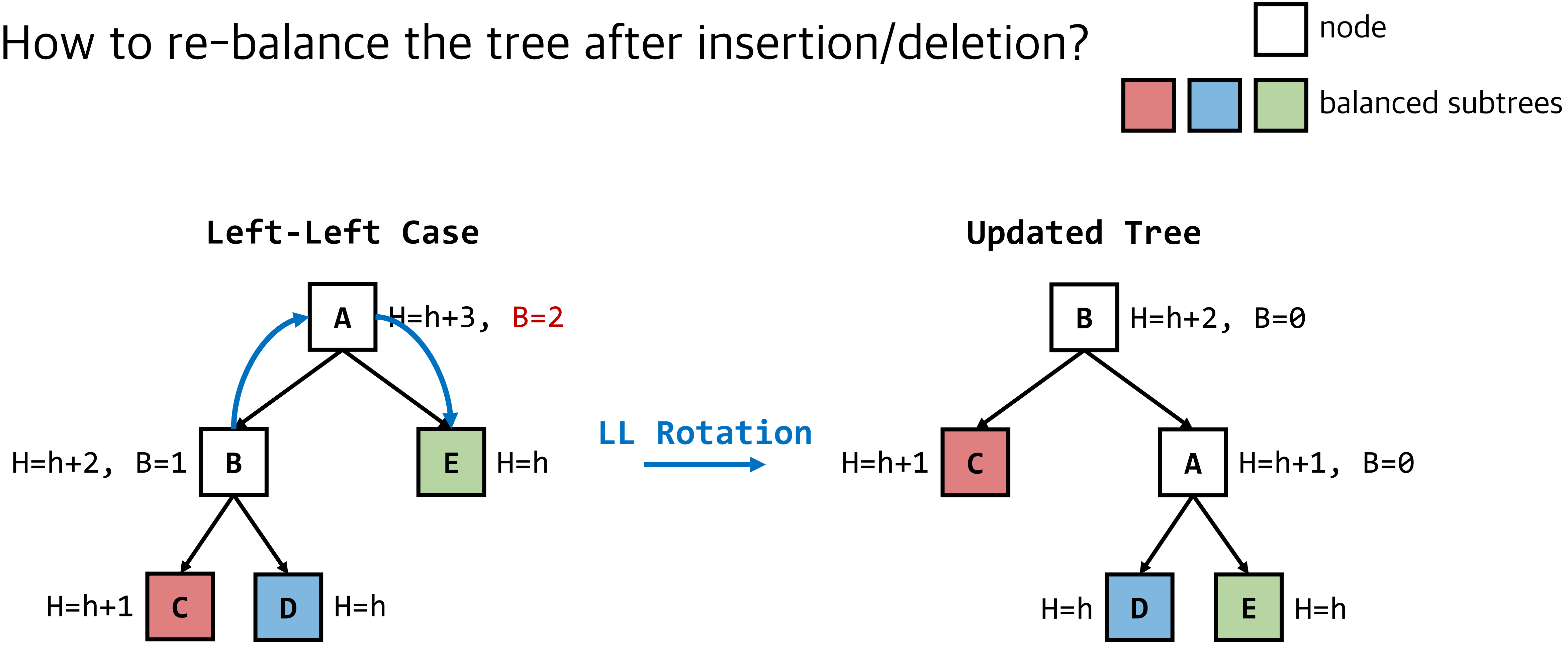
- A binary tree T is **balanced** if |`balance(X)`| ≤ 1 for any node **X**



This tree is **NOT** balanced

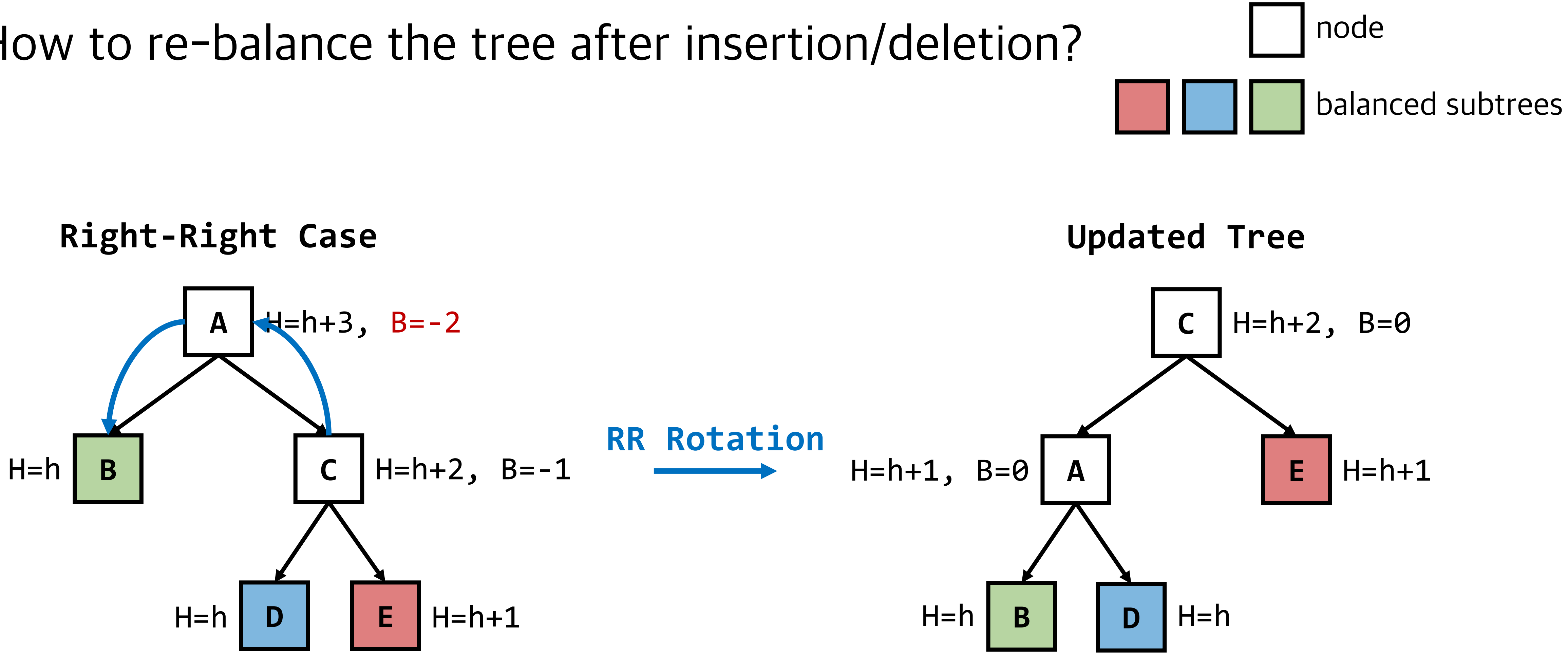
(Recap) AVL Trees - Rotations

(Q) How to re-balance the tree after insertion/deletion?



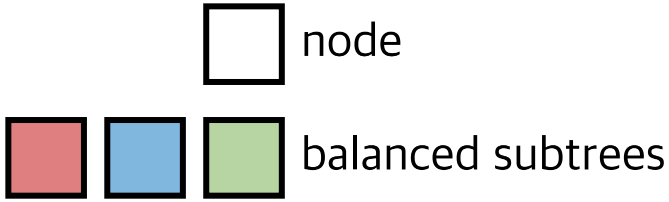
(Recap) AVL Trees - Rotations

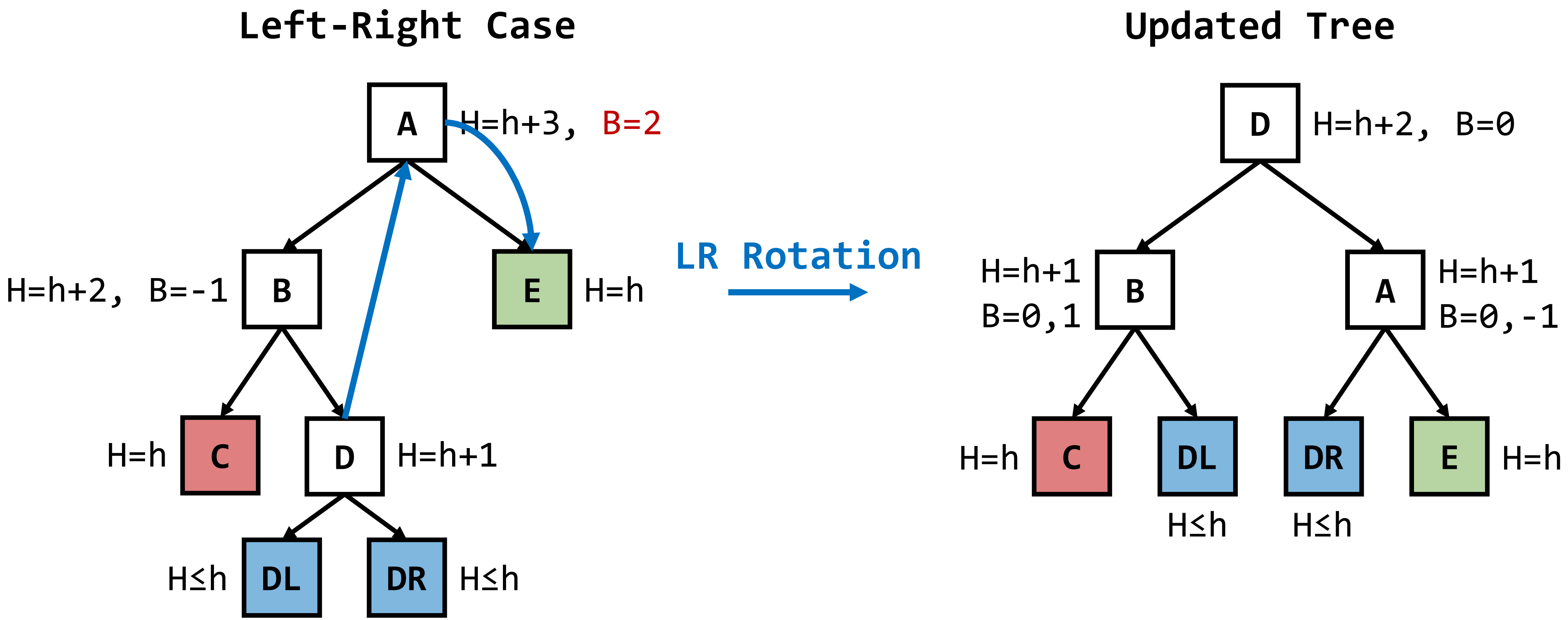
(Q) How to re-balance the tree after insertion/deletion?



(Recap) AVL Trees - Rotations

(Q) How to re-balance the tree after insertion/deletion?



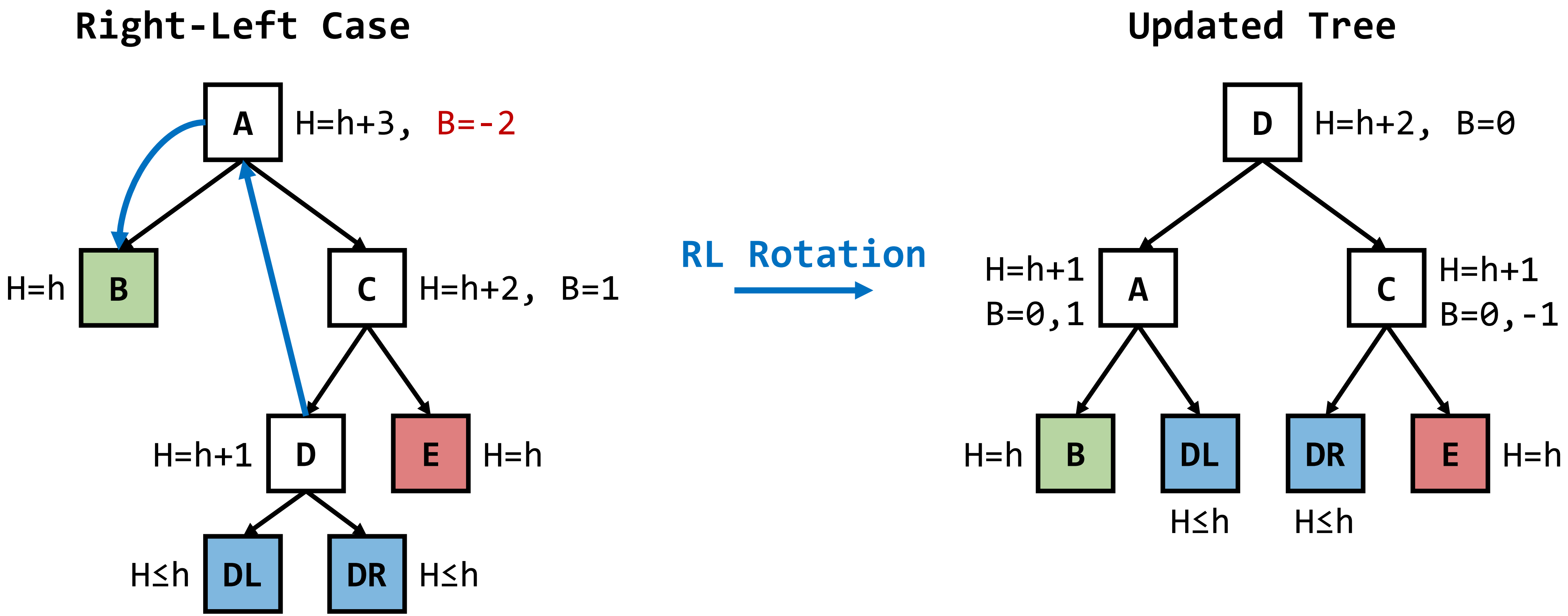


(Recap) AVL Trees - Rotations

(Q) How to re-balance the tree after insertion/deletion?

node

balanced subtrees

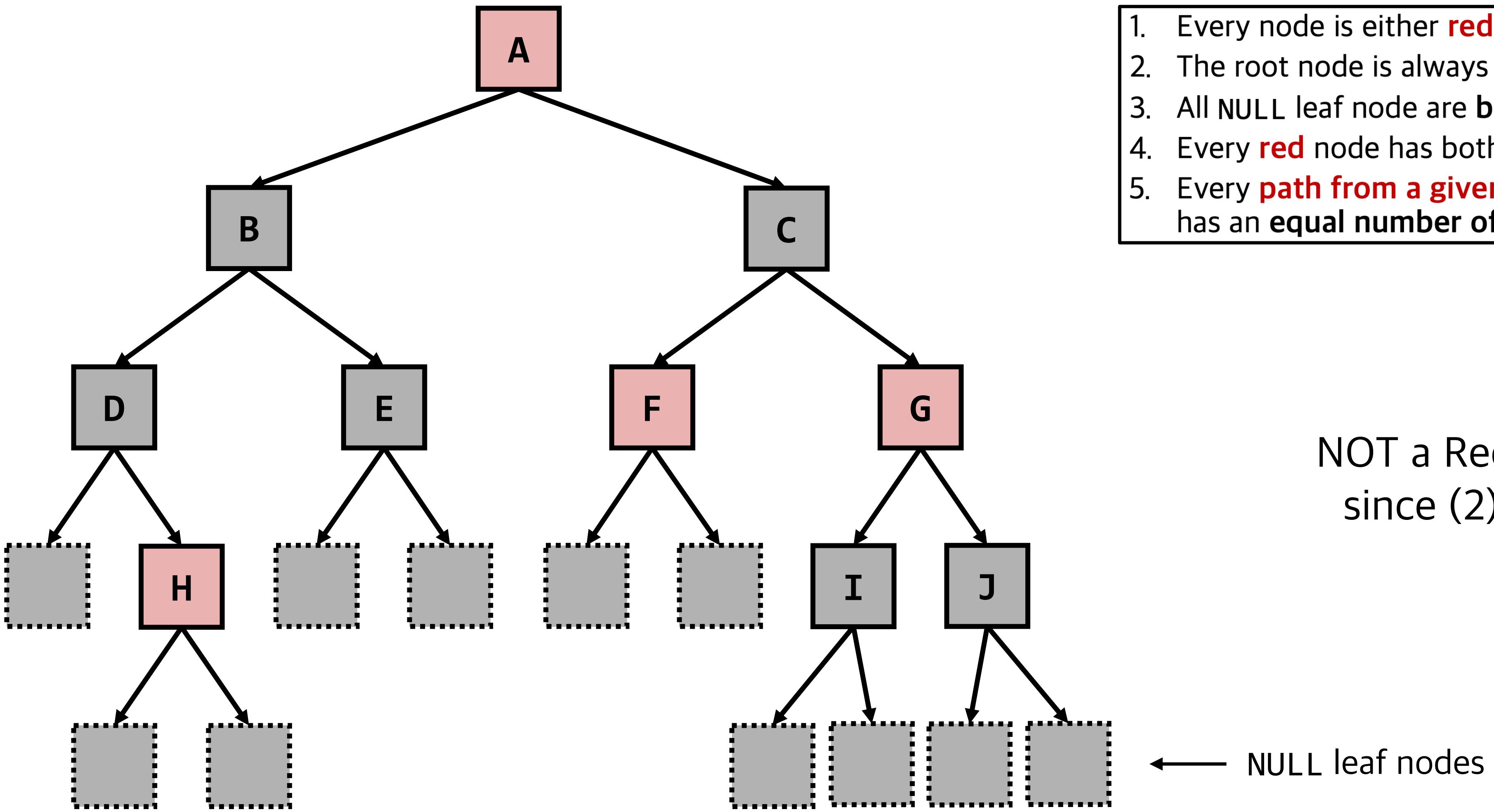


Red-Black Trees

- **Red-Black Tree** is another **self-balancing** BST
  - Its height is $O(\log N)$ like AVL Tree

- Red-Black Tree should satisfy the following properties:
  1. Every node is either **red** or **black**
  2. The root node is always **black**
  3. All `NULL` leaf node are **black**
  4. Every **red** node has both the children colored in **black**
  5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

Red-Black Trees - Examples

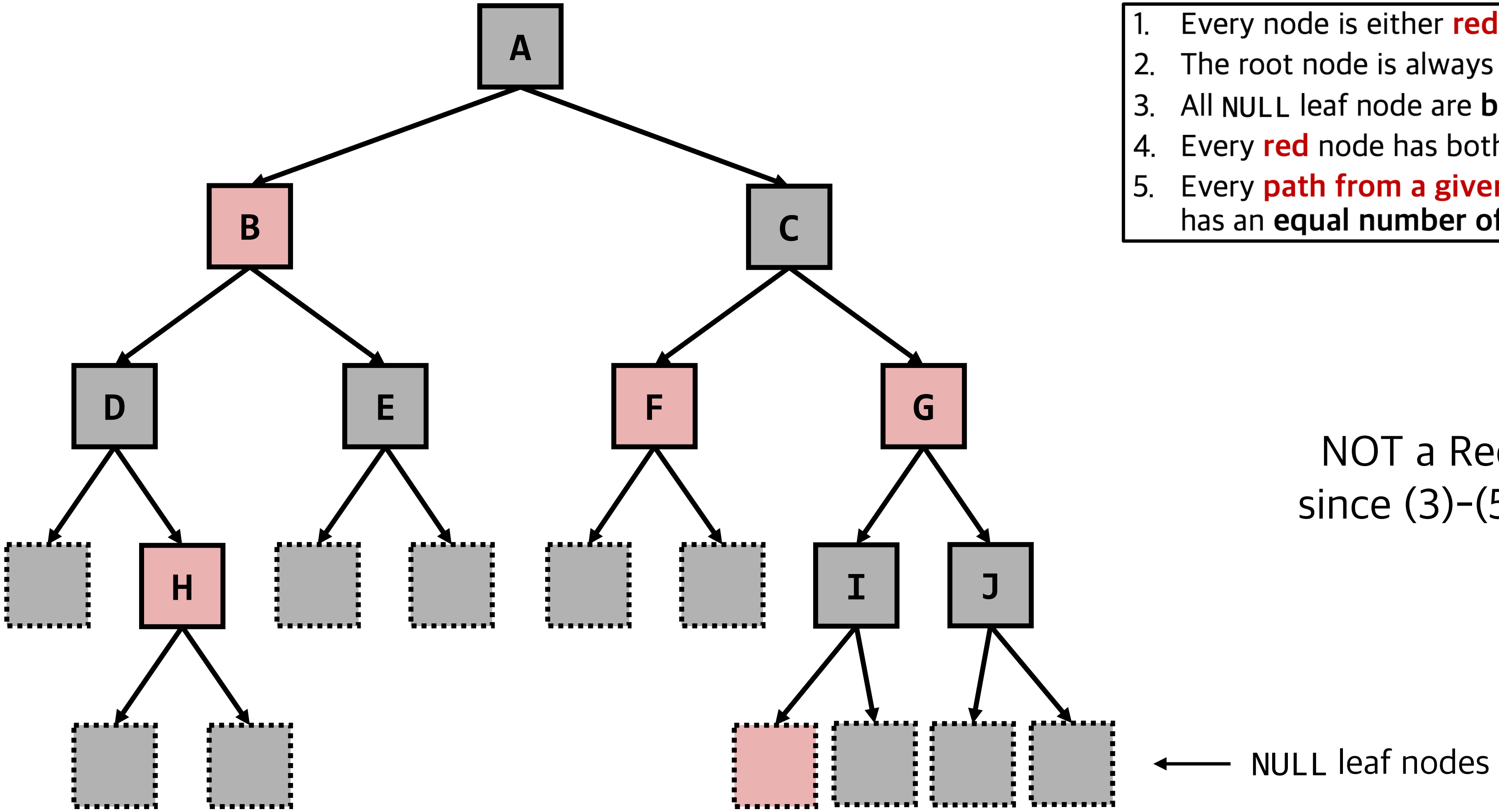


Red-Black Tree Properties

1. Every node is either **red** or **black**
2. The root node is always **black**
3. All NULL leaf node are **black**
4. Every **red** node has both the children colored in **black**
5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

NOT a Red-Black Tree
since (2) is violated

Red-Black Trees - Examples

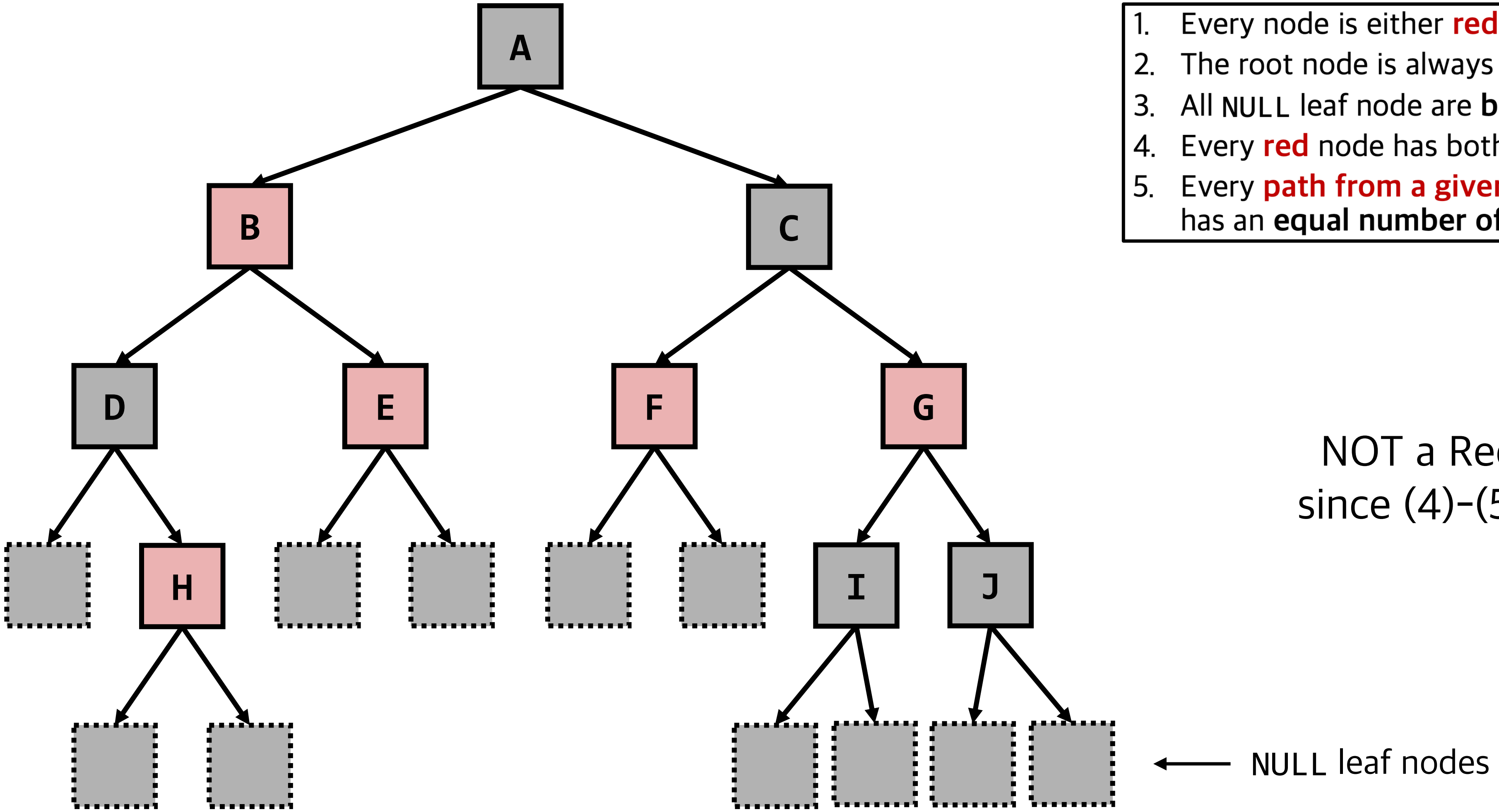


Red-Black Tree Properties

1. Every node is either **red** or **black**
2. The root node is always **black**
3. All NULL leaf node are **black**
4. Every **red** node has both the children colored in **black**
5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

NOT a Red-Black Tree
since (3)-(5) are violated

Red-Black Trees - Examples

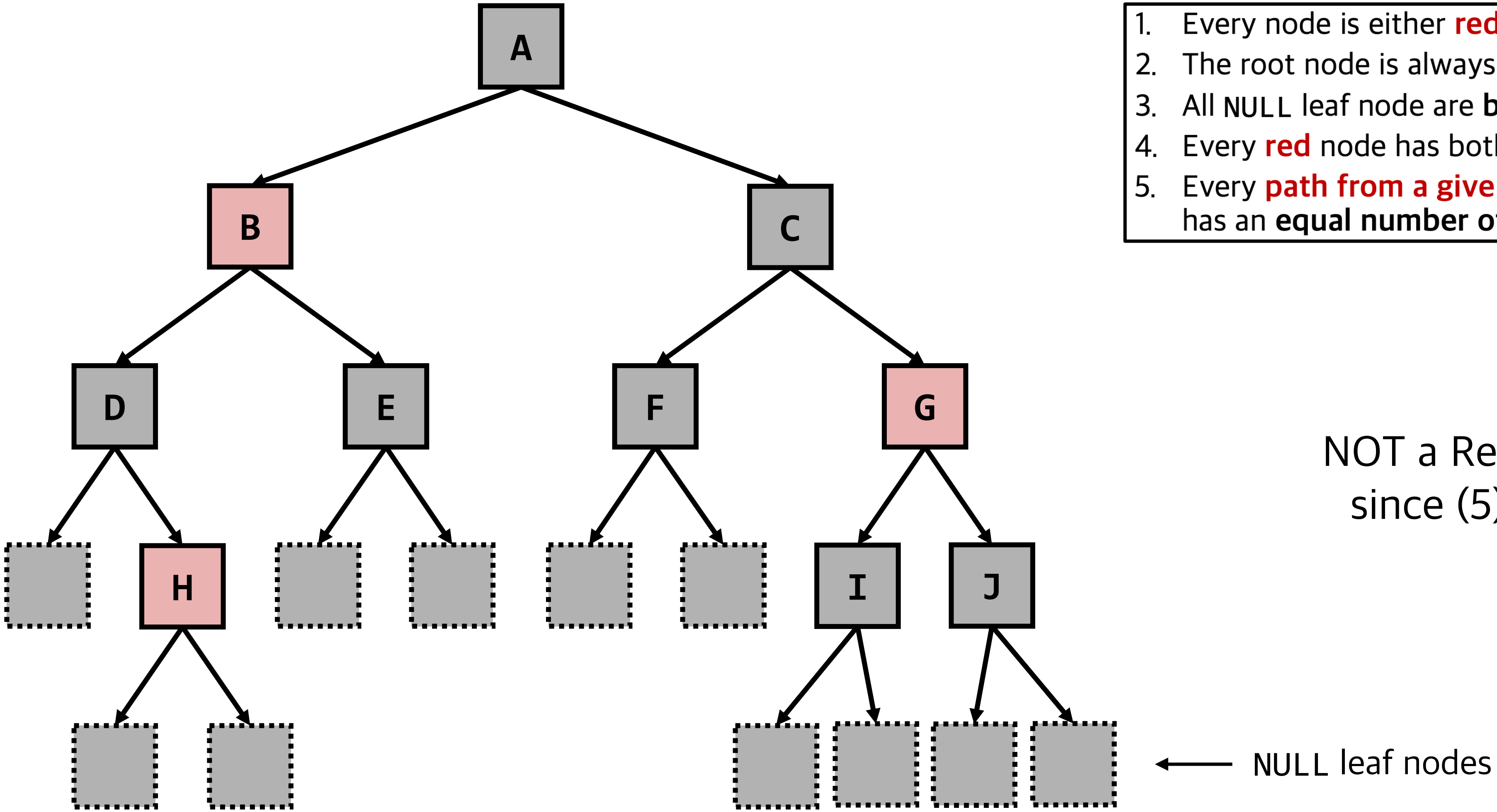


Red-Black Tree Properties

1. Every node is either **red** or **black**
2. The root node is always **black**
3. All NULL leaf node are **black**
4. Every **red** node has both the children colored in **black**
5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

NOT a Red-Black Tree
since (4)-(5) are violated

Red-Black Trees - Examples

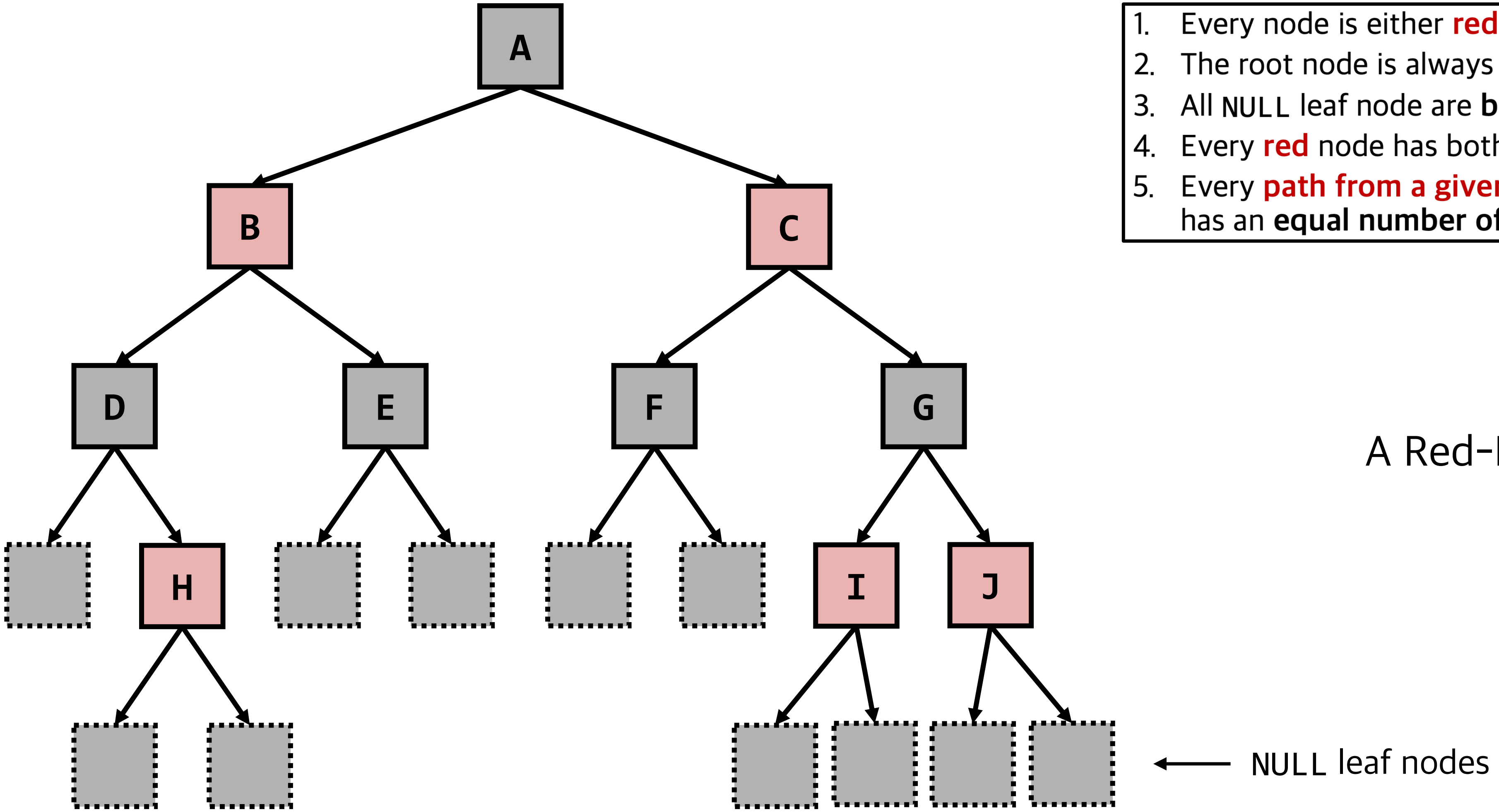


Red-Black Tree Properties

1. Every node is either **red** or **black**
2. The root node is always **black**
3. All `NULL` leaf node are **black**
4. Every **red** node has both the children colored in **black**
5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

NOT a Red-Black Tree
since (5) is violated

Red-Black Trees - Examples



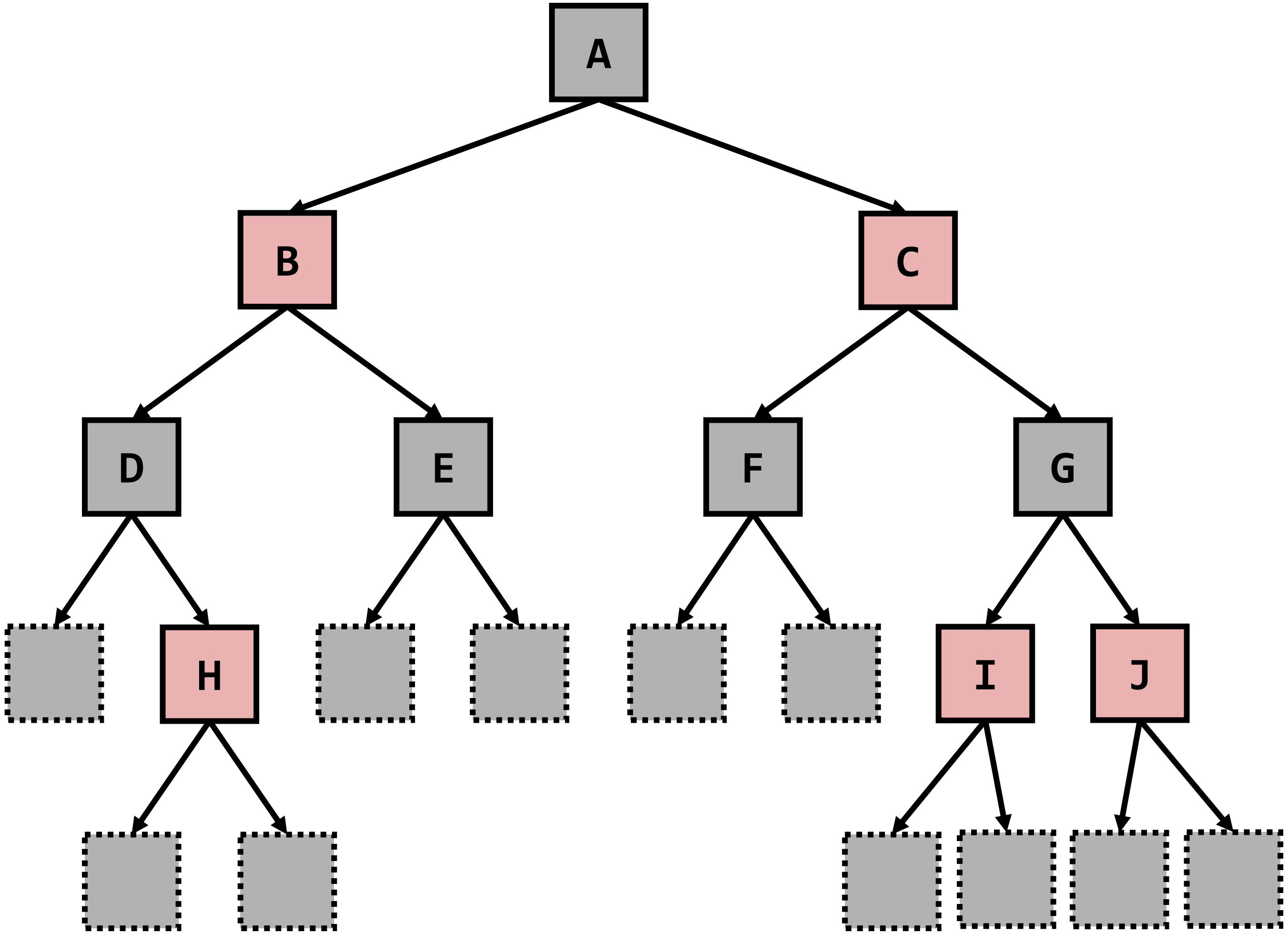
Red-Black Tree Properties

1. Every node is either **red** or **black**
2. The root node is always **black**
3. All `NULL` leaf node are **black**
4. Every **red** node has both the children colored in **black**
5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

A Red-Black Tree

Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node
  - BH(A) = 3
  - BH(B,C) = 2
  - BH(D,E,F,G) = 2
  - BH(H,I,J) = 1



Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 1) Any node `X` with height `H(X)` has `BH(X)≥H(X)/2`

- Consider a longest path from X to a leaf Y

$$X=Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow \ldots \rightarrow Z_{H(X)-2} \rightarrow Z_{H(X)-1} \rightarrow Y=Z_{H(X)}=\text{NULL}$$

Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 1) Any node X with height H(X) has BH(X)≥H(X)/2

- Consider a longest path from X to a leaf Y

$$X=Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow \ldots \rightarrow Z_{H(X)-2} \rightarrow Z_{H(X)-1} \rightarrow Y=Z_{H(X)}=NULL$$

- Since the properties (3) any NULL leaf nodes is black and (4) the children of any red node are black, the maximum number of red nodes in the path is H(X)/2

- In other words, the minimum number of black nodes is H(X)/2

Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 2) A subtree rooted at any node `X` has at least $2^{BH(X)}-1$ nodes

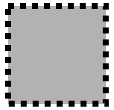
  - By induction on `H(X)`, the height of `X`

Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 2) A subtree rooted at any node `X` has at least $2^{BH(X)}-1$ nodes

- By induction on `H(X)`, the height of `X`
- Base case: `H(X)=1`  ⟵ NULL leaf node

Red-Black Trees - Heights

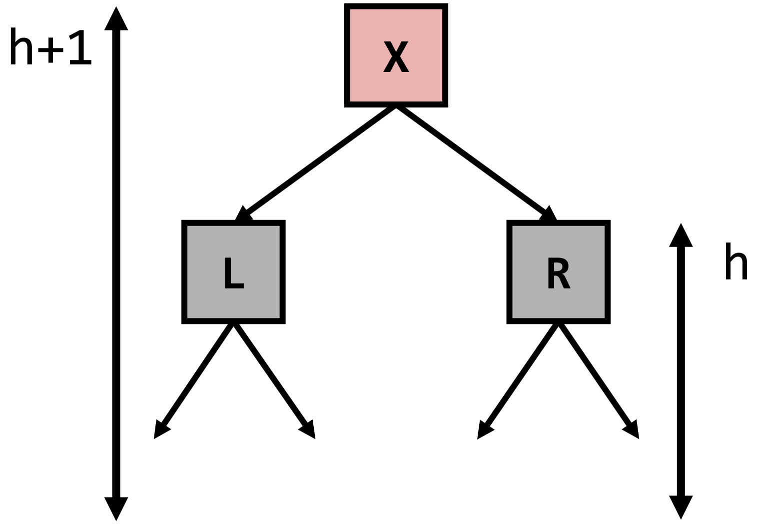
- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 2) A subtree rooted at any node X has at least $2^{BH(X)}-1$ nodes

- By induction on H(X), the height of X

- Inductive case:
  Assume the above statement is true when H(X)≤h.
  If H(X)=h+1 and X is **red**, then BH(X)=BH(L), ...



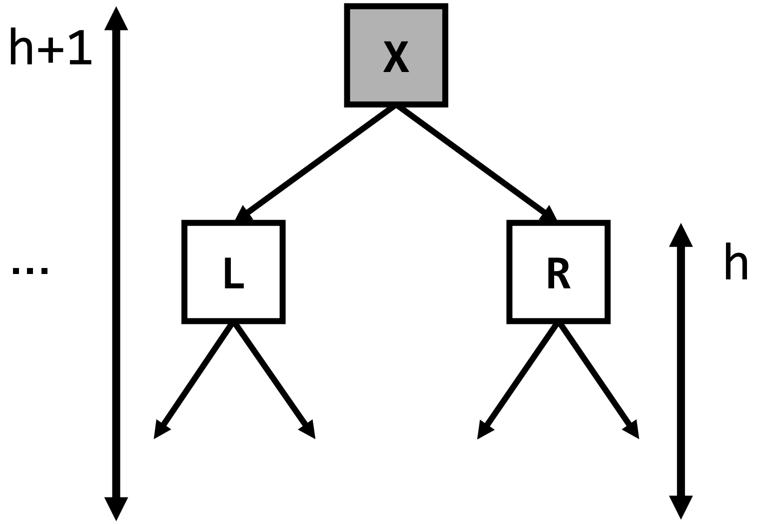
Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 2) A subtree rooted at any node X has at least $2^{BH(X)}-1$ nodes

- By induction on H(X), the height of X
- Inductive case:
  Assume the above statement is true when H(X)≤h.
  If H(X)=h+1 and X is **black**, then BH(L,R)=BH(X)-1, …



Red-Black Trees - Heights

- The black-height of a node is **the number of black nodes** in a path from the node to its leaf node

(Q) Why the height of a red-black tree of N nodes = $O(\log N)$?

(Step 1) Any node `X` with height `H(X)` has `BH(X)≥H(X)/2`

(Step 2) A subtree rooted at any node `X` has at least $2^{BH(X)}-1$ nodes

$$N \geq 2^{BH(X)}-1 \geq 2^{H(X)/2}-1$$

$$2\log(N+1) \geq H(X)$$

(Recap) Height of Balanced Binary Trees

- The **balance factor** of a node **`X`** in a binary tree is defined by

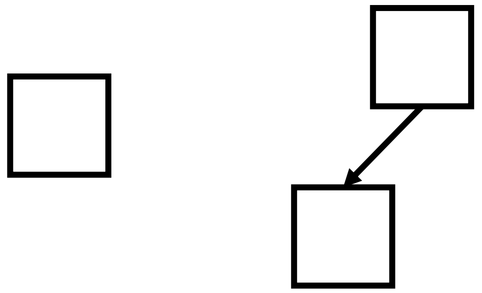
  `balance(X) = height(left subtree) – height(right subtree)`

- A binary tree T is **balanced** if |`balance(X)`| ≤ 1 for any node **`X`**

(Q) Why the height of a balanced binary tree of N nodes = $O(\log N)$?

(A) A subtree rooted at any node `X` has at least $2^{H(X)/2}-1$ nodes

- By induction on `H(X)`, the height of `X`
- Base case: `H(X)=1` and `H(X)=2`



(Recap) Height of Balanced Binary Trees

- The **balance factor** of a node **X** in a binary tree is defined by

  `balance(X) = height(left subtree) – height(right subtree)`

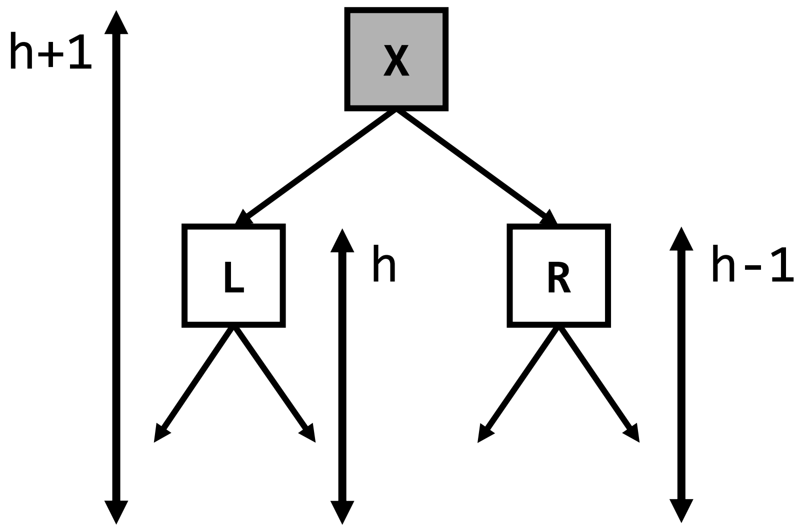
- A binary tree T is **balanced** if |`balance(X)`| ≤ 1 for any node **X**

(Q) Why the height of a balanced binary tree of N nodes = $O(\log N)$?

(A) A subtree rooted at any node X has at least $2^{H(X)/2}-1$ nodes

- By induction on H(X), the height of X
- Inductive case:
  Assume the above statement is true when H(X)≤h.

$$\begin{aligned} N(h+1) &\geq 1 + N(h) + N(h-1) \\ &\geq 2^{h/2} + 2^{(h-1)/2} - 1 \\ &\geq 2^{(h+1)/2} - 1 \end{aligned}$$



Red-Black Trees - Insertion

- How to insert a new node into a Red-Black tree?
  1. Insert an element as usual in the BST (i.e., replace `NULL` by the new node)
  2. Color the node **RED**
  3. Check if the properties of the red-black tree are violated
  4. If violated, modify the tree in the bottom-up direction

Red-Black Trees - Insertion

- How to insert a new node into a Red-Black tree?
  1. Insert an element as usual in the BST (i.e., replace `NULL` by the new node)
  2. Color the node **RED**
  3. Check if the properties of the red-black tree are violated
  4. If violated, modify the tree in the bottom-up direction

- **Which properties may be violated?**
  1. Every node is either **red** or **black**
  2. The root node is always **black**
  3. All `NULL` leaf node are **black**
  4. Every **red** node has both the children colored in **black**
  5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

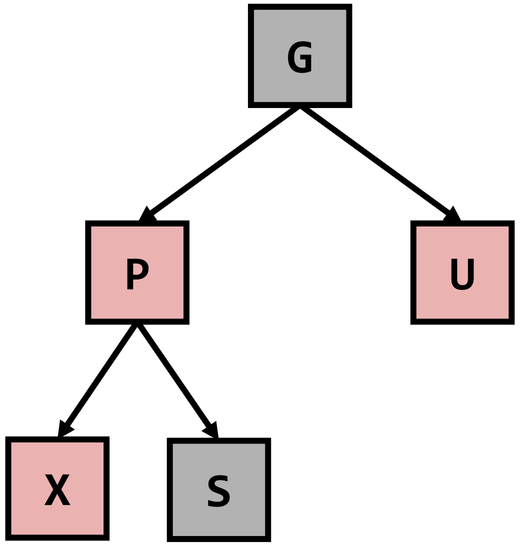
Red-Black Trees - Insertion

- How to insert a new node into a Red-Black tree?
  1. Insert an element as usual in the BST (i.e., replace `NULL` by the new node)
  2. Color the node **RED**
  3. Check if the properties of the red-black tree are violated
  4. If violated, modify the tree in the bottom-up direction

- **Which properties may be violated?**
  1. Every node is either **red** or **black**
  2. The root node is always **black**
  3. All `NULL` leaf node are **black**
  4. Every **red** node has both the children colored in **black**
  5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 1)` `X`** is not root, and its uncle is **red**

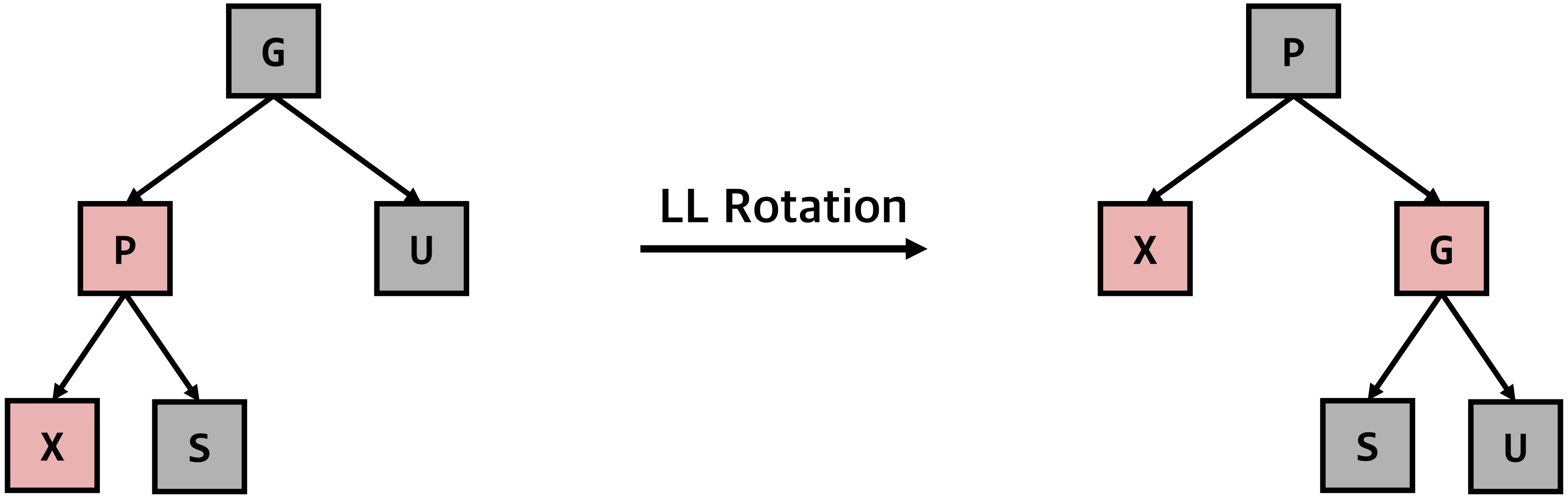


Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 1)` `X`** is not root, and its uncle is **red**
    1. Perform **Color Promotion**
    2. Check the grandparent **`G`** recursively

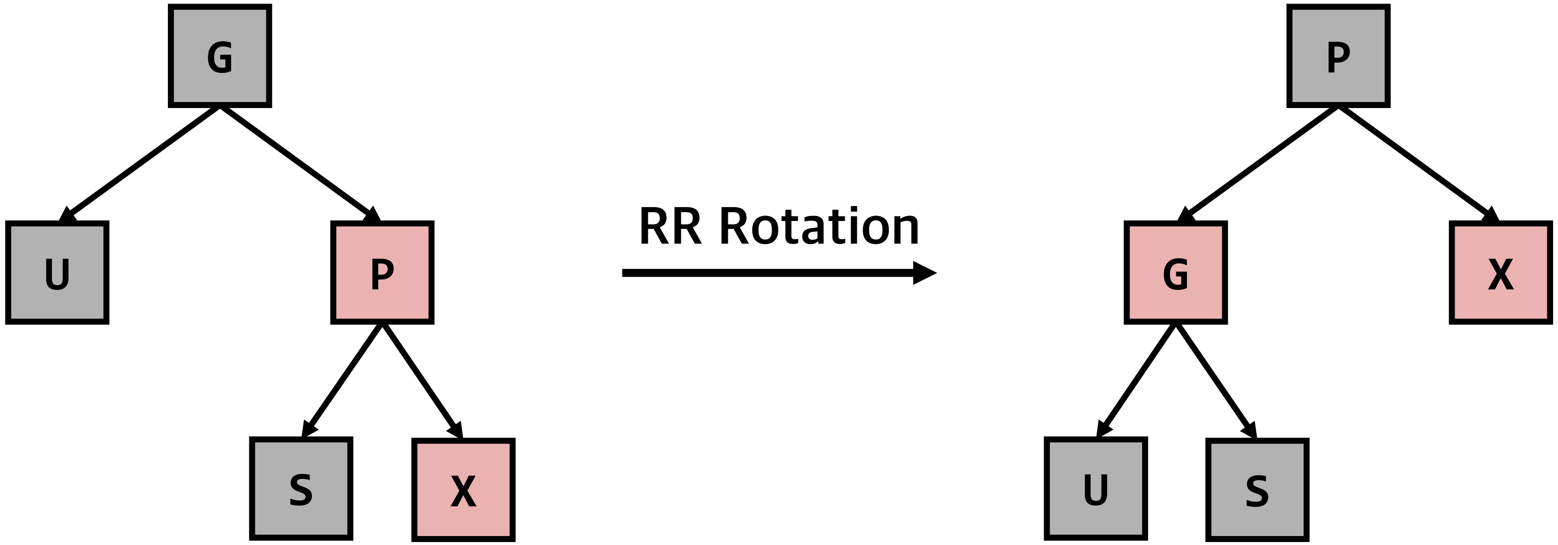
Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)`** **X** is not root, and its uncle is **black**, and **X** is on the **left-left** subtree
    1. Perform **LL Rotation** with color changes



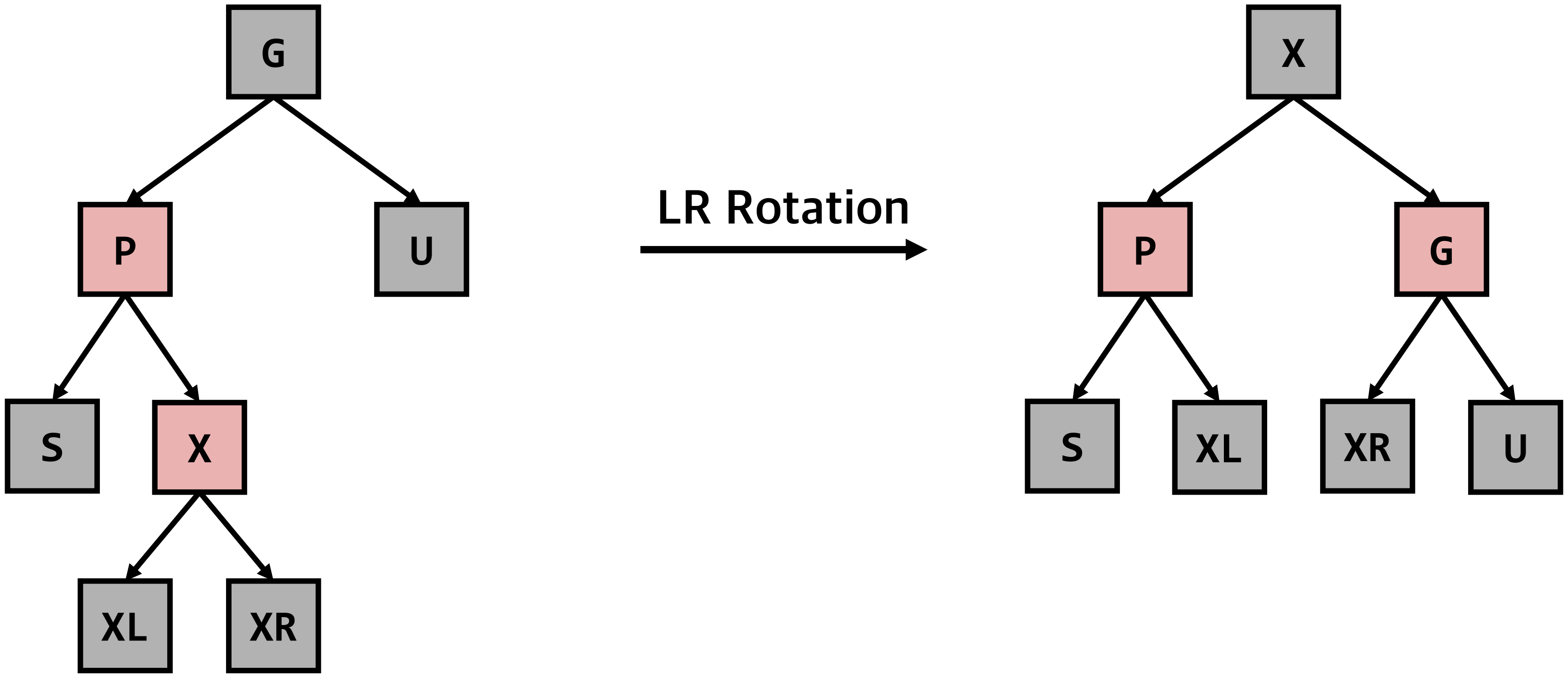
Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)` X** is not root, and its uncle is **black**, and **X** is on the **right-right** subtree
    1. Perform **RR Rotation** with color changes



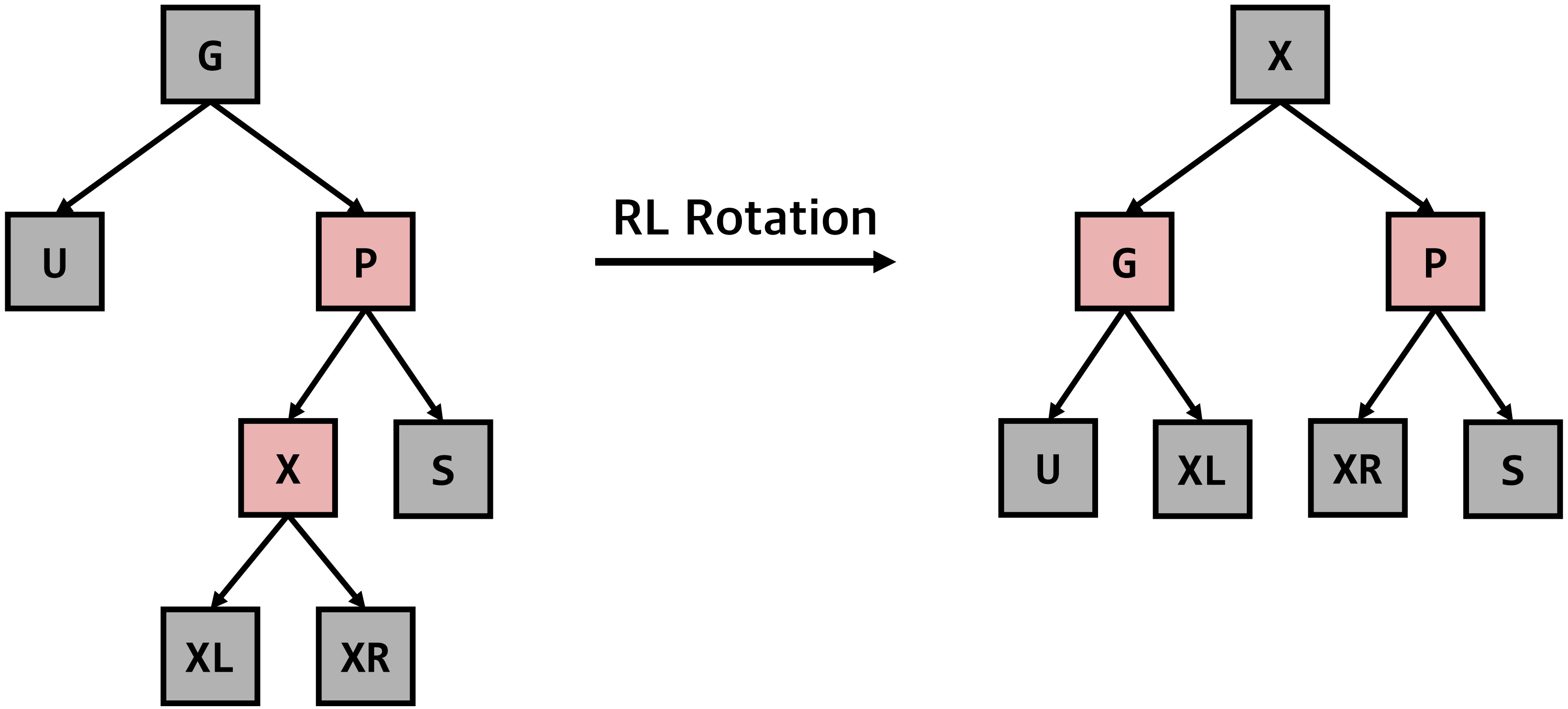
Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - `(Case 3)` **X** is not root, and its uncle is **black**, and **X** is on the **left-right** subtree
    1. Perform **LR Rotation** with color changes



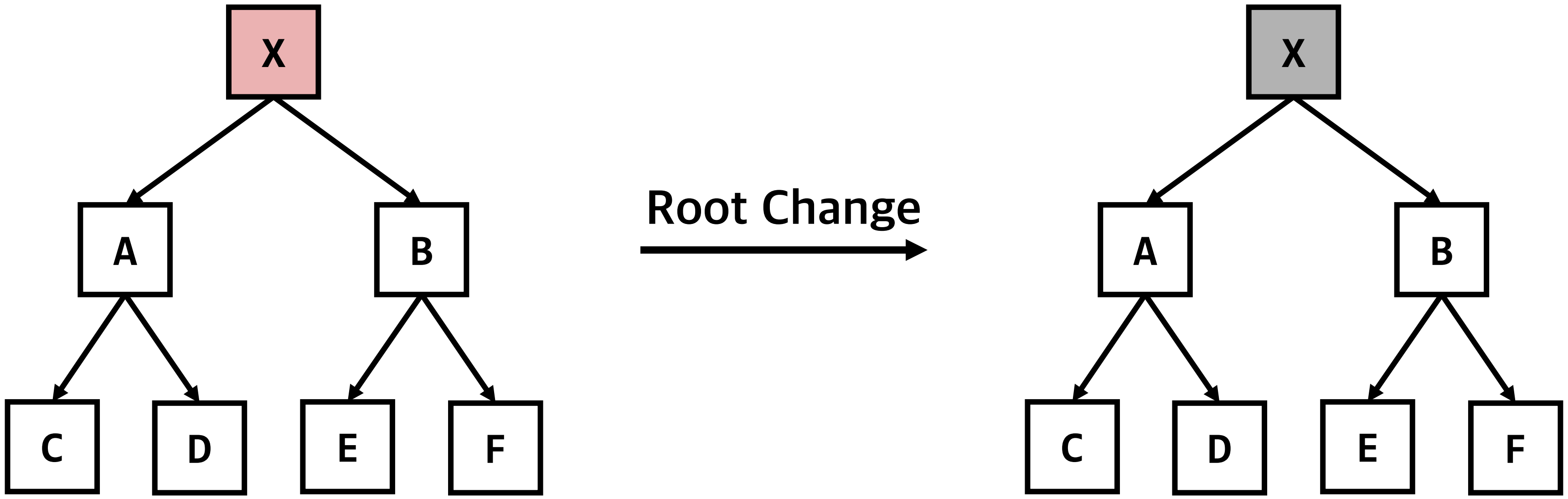
Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - `(Case 3)` **X** is not root, and its uncle is **black**, and **X** is on the **right-left** subtree
    1. Perform **RL Rotation** with color changes



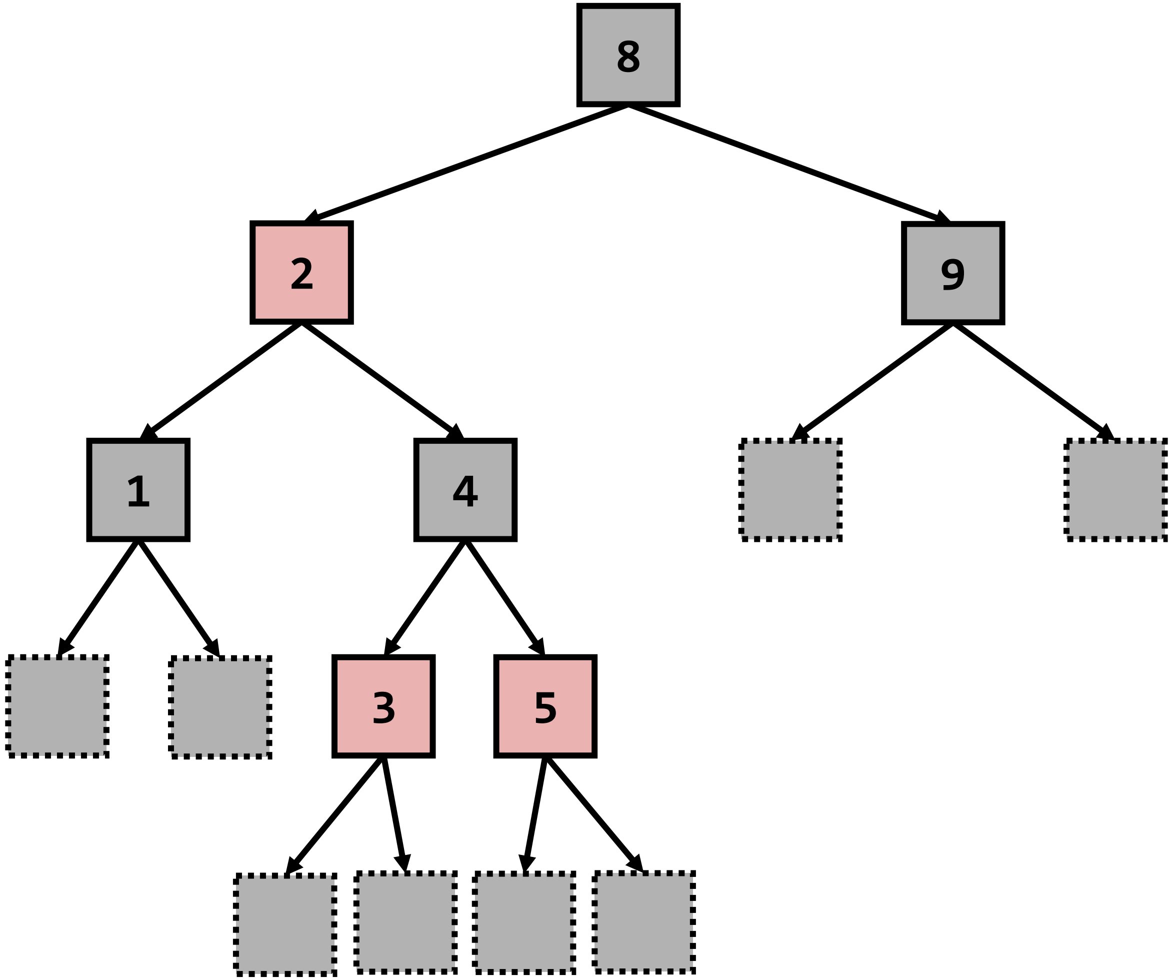
Red-Black Trees - Insertion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 0)` `X`** is root, and its color is **red**
    1. Color it **black**



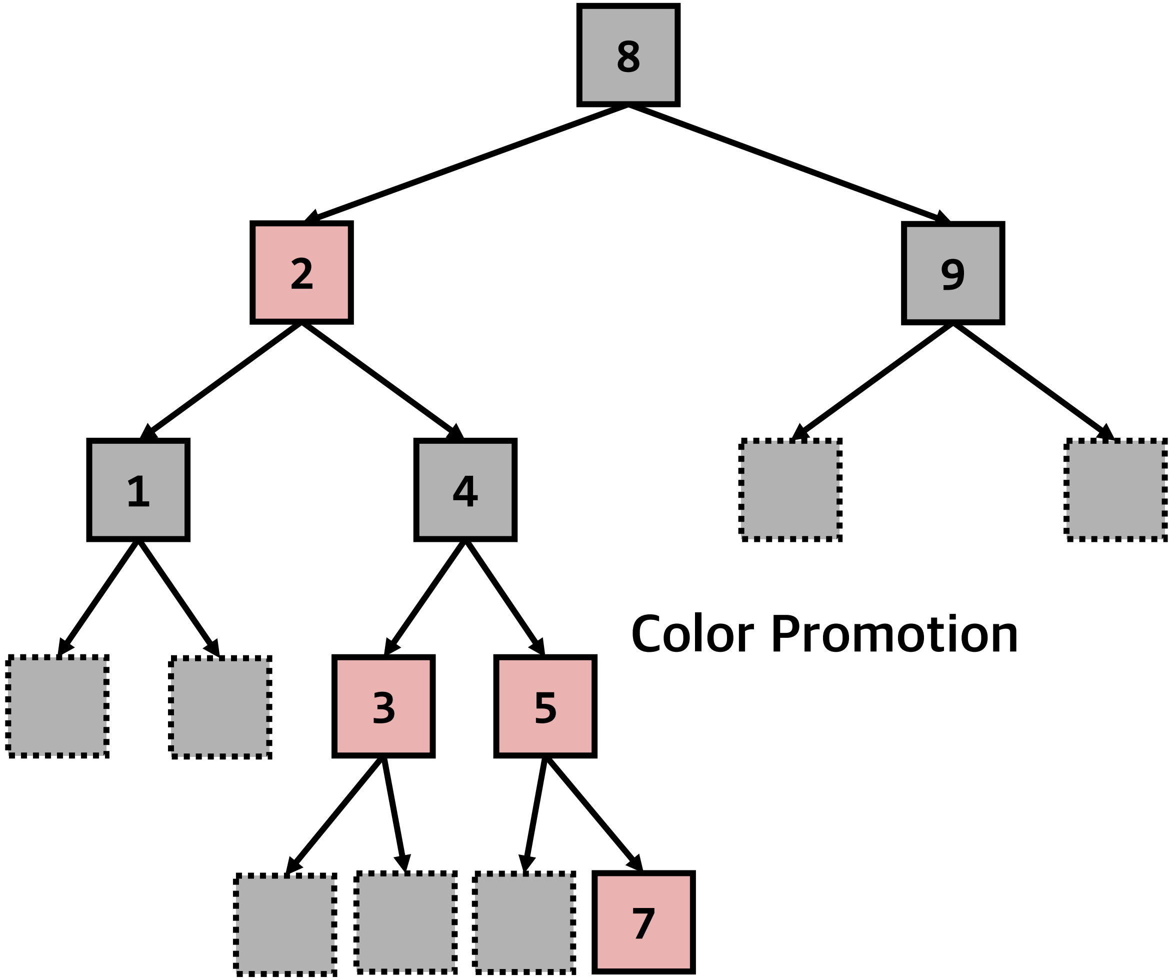
Red-Black Trees - Insertion

- Example - Insert 7



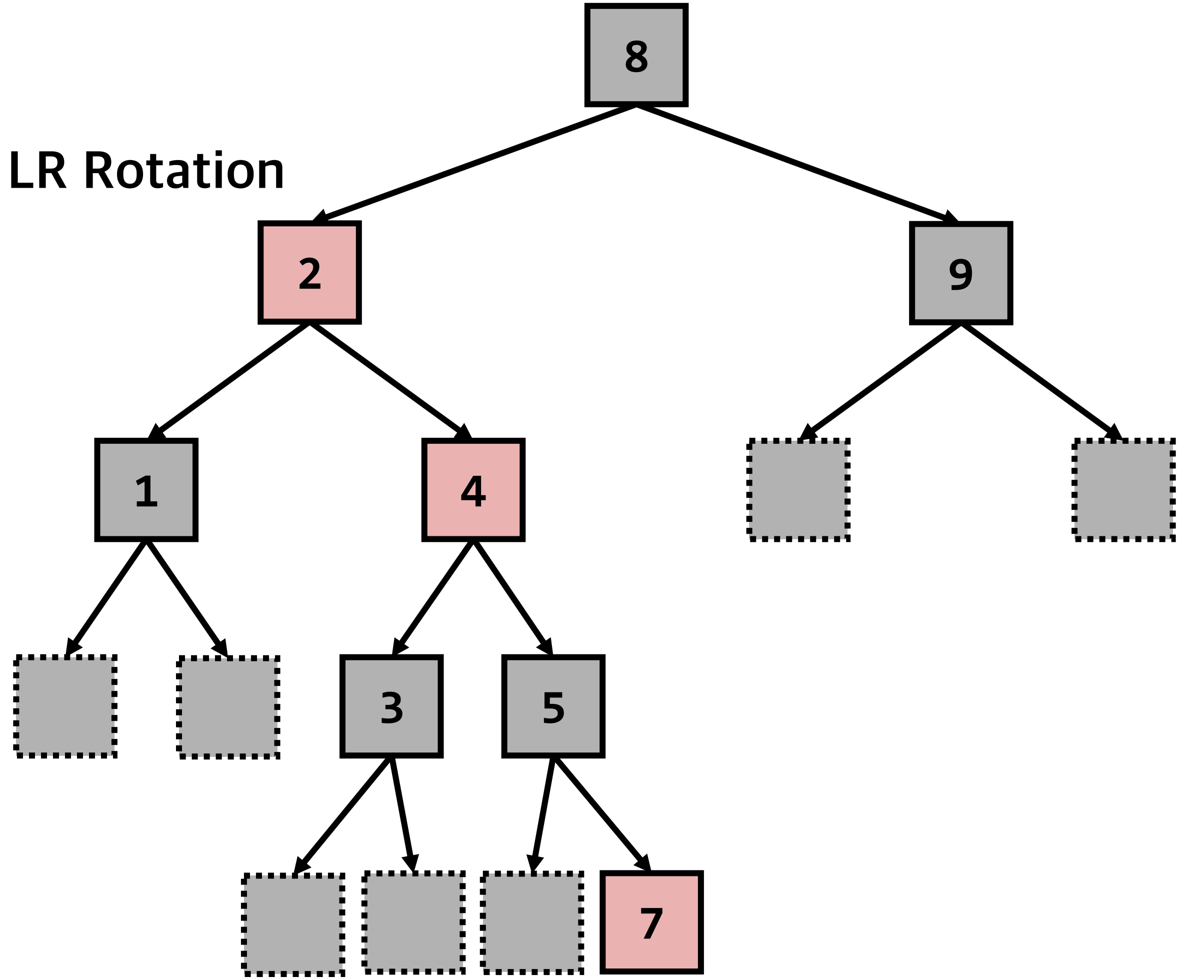
Red-Black Trees - Insertion

- Example - Insert 7



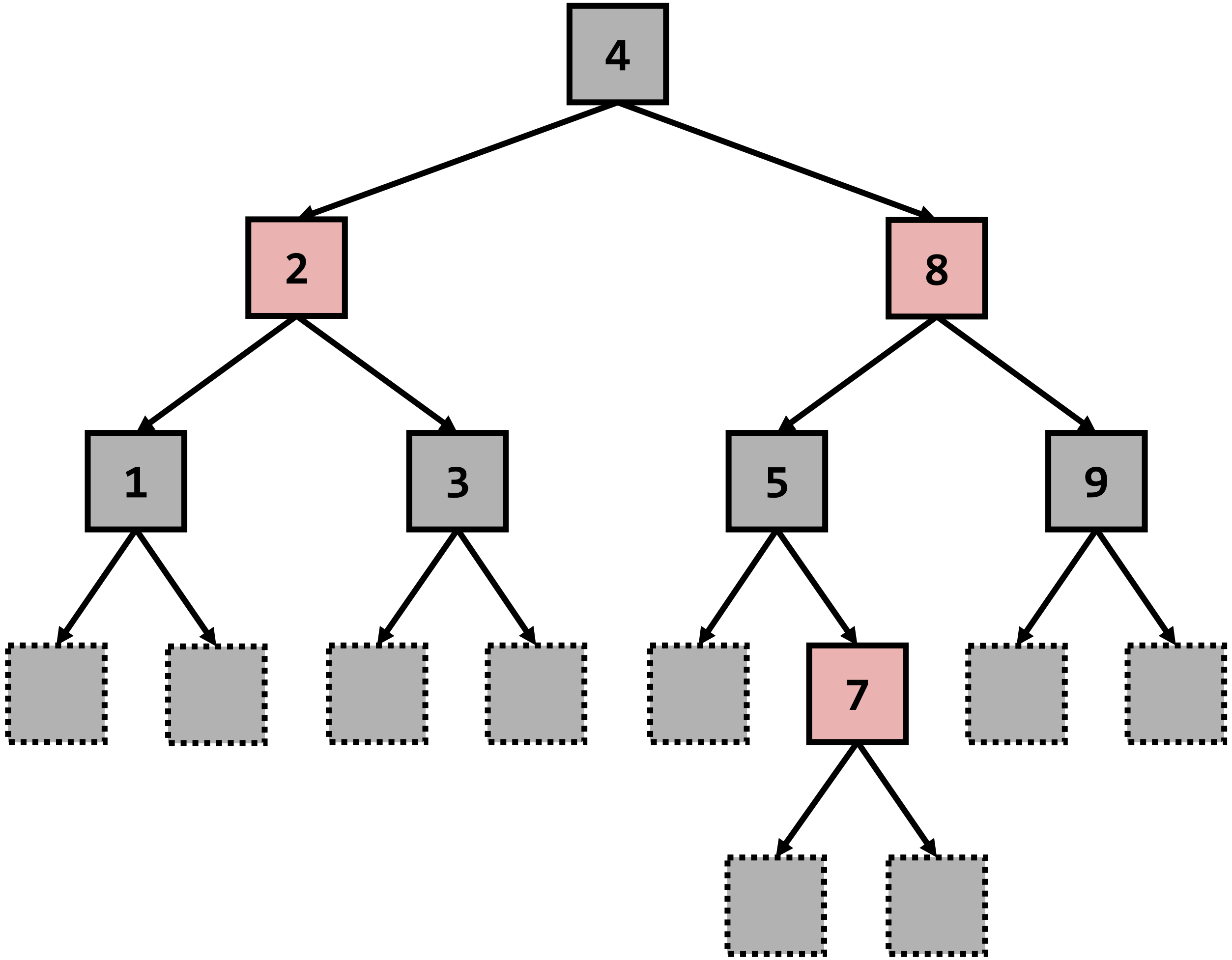
Red-Black Trees - Insertion

- Example - Insert 7



Red-Black Trees - Insertion

- Example - Insert 7



Red-Black Trees - Deletion

- How to delete a node from a Red-Black tree?
  1. Delete an element as usual in the BST
  2. Check if a property of the red-black tree is violated
  3. If violated, modify the tree in the **bottom-up** direction

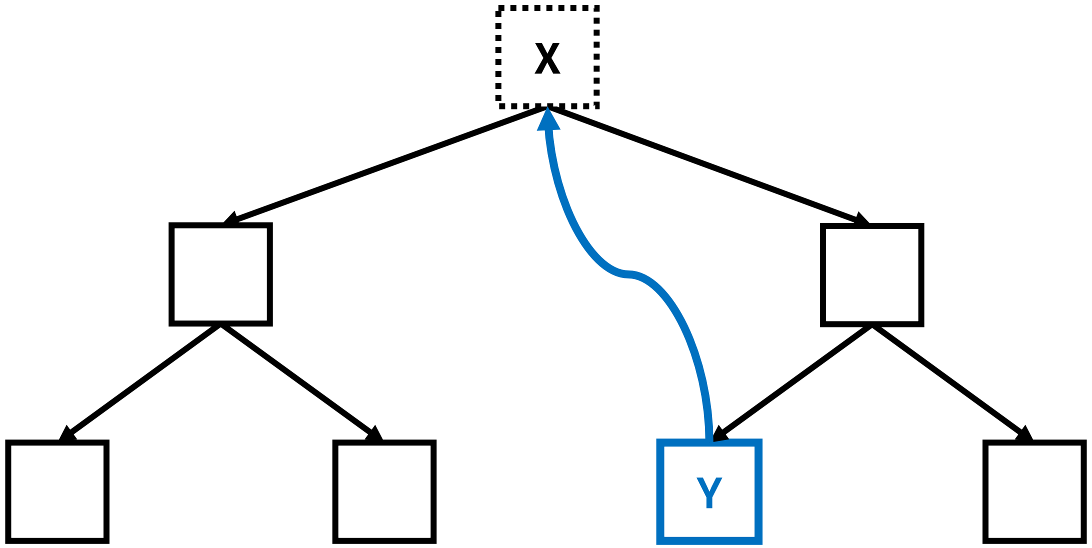
Red-Black Trees - Deletion

- How to delete a node from a Red-Black tree?
    1. Delete an element as usual in the BST
    2. Check if a property of the red-black tree is violated
    3. If violated, modify the tree in the **bottom-up** direction

- **Which properties may be violated?**
    1. Every node is either **red** or **black**
    2. The root node is always **black**
    3. All `NULL` leaf node are **black**
    4. Every **red** node has both the children colored in **black**
    5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

Red-Black Trees - Deletion

- (Recap) How to delete an element in the BST?
  - **`(Case 1)`** If the node has no child, it can be simply deleted
  - **`(Case 2)`** If the node has one child, it can be deleted like the linked list structure
  - **`(Case 3)`** If the node has two children, must find a replacement node
    - You don't need to care about this case
    - Check the replacement node in a recursive manner



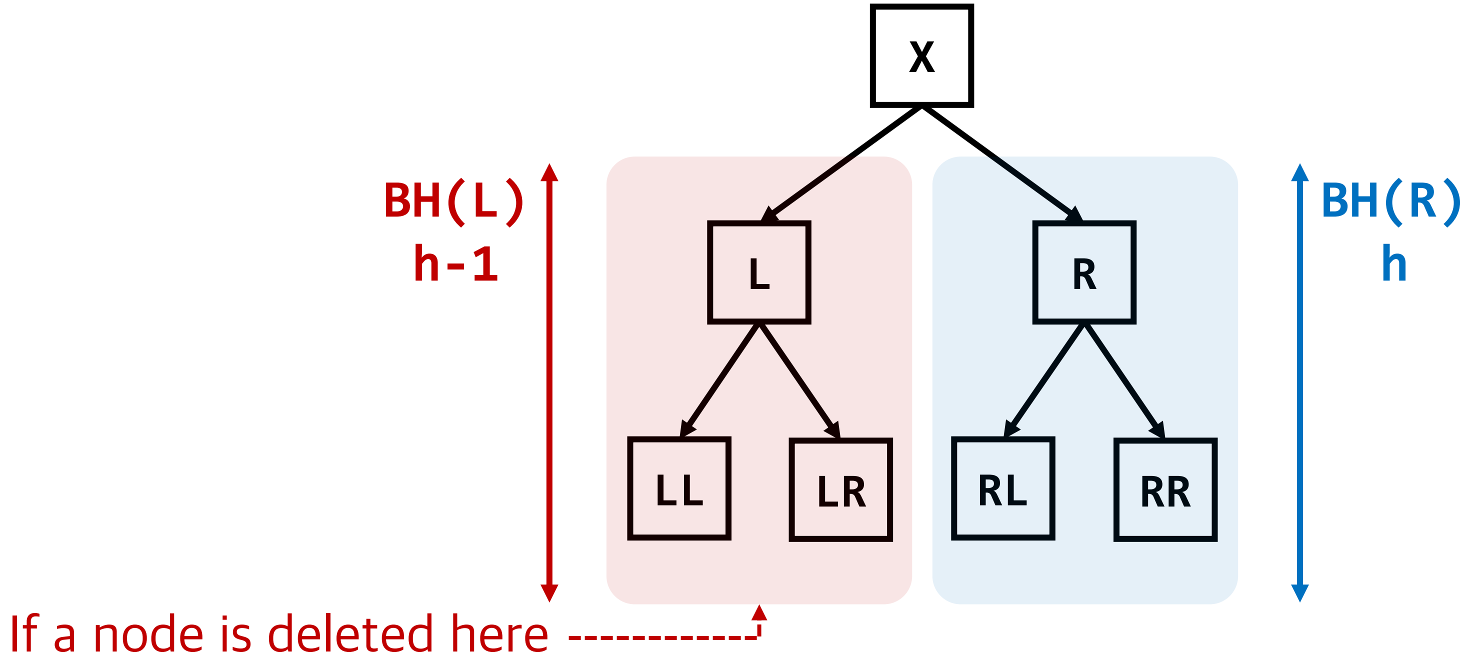
Red-Black Trees - Deletion

- How to delete a node from a Red-Black tree?
  1. Delete an element as usual in the BST
  2. Check if a property of the red-black tree is violated
  3. If violated, modify the tree in the **bottom-up** direction

- **Which properties may be violated?**
  1. Every node is either **red** or **black**
  2. The root node is always **black**
  3. All NULL leaf node are **black**
  4. Every **red** node has both the children colored in **black**
  5. Every **path from a given node to any of its leaf nodes** has an **equal number of black nodes**

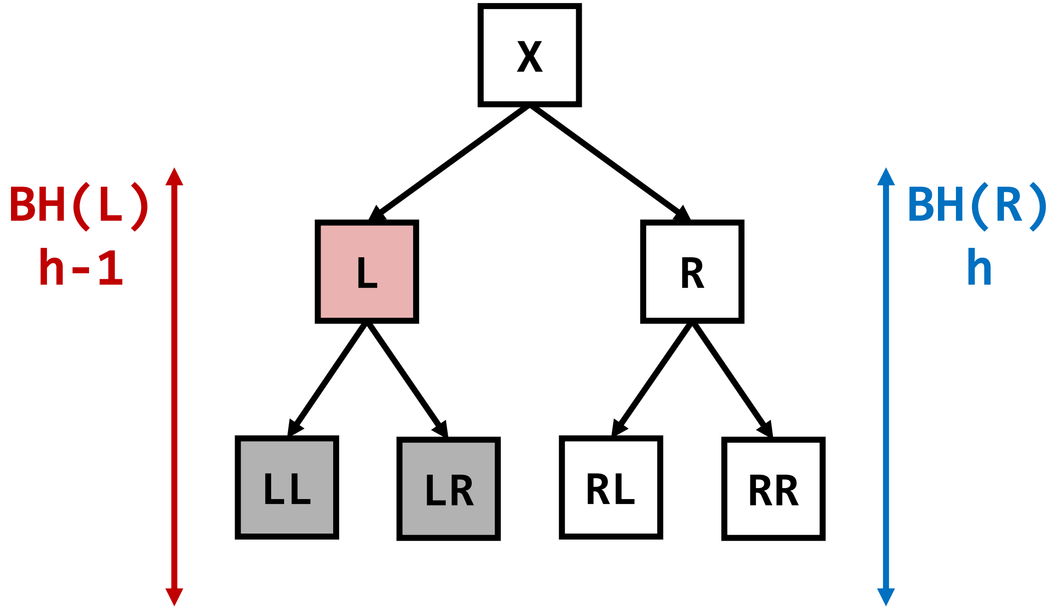
Red-Black Trees - Deletion

- How to delete a node from a Red-Black tree?
  1. Delete an element as usual in the BST
  2. Check if a property of the red-black tree is violated
  3. If violated, modify the tree in the **bottom-up** direction



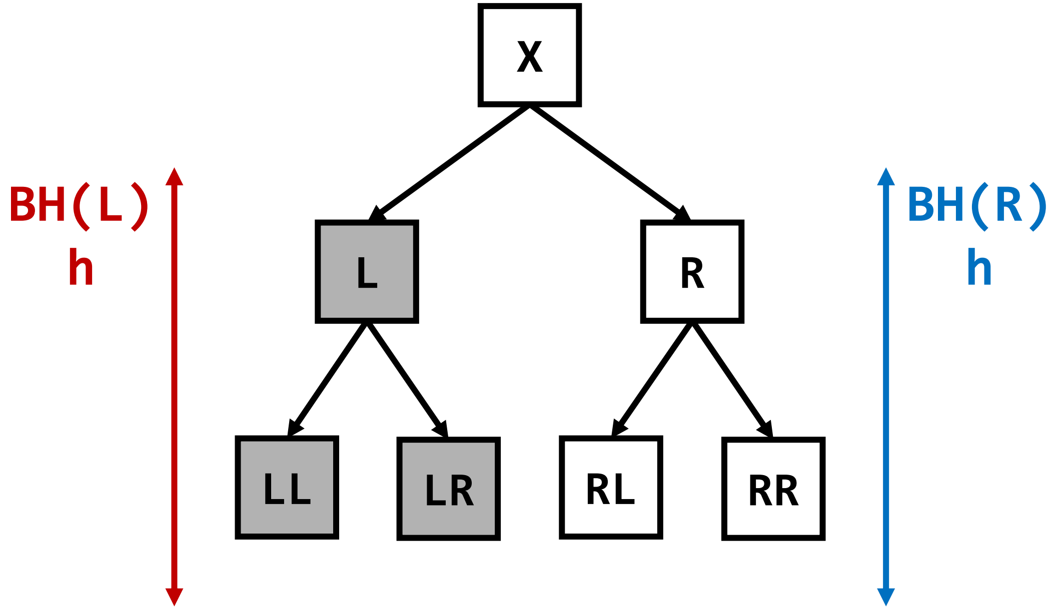
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 1) BH(L)+1=BH(R)`** and **`L`** is **red**
  - **`(Solution)`** Simply color **`L`** **black**



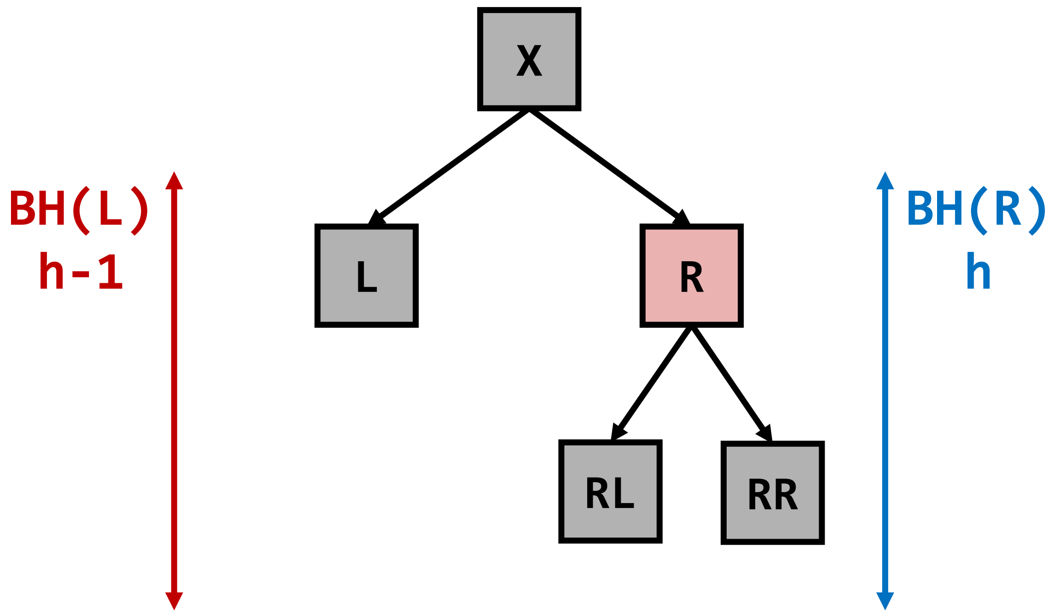
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 1)`** **`BH(L)+1=BH(R)`** and **`L`** is **red**
  - **`(Solution)`** Simply color **`L`** **black**



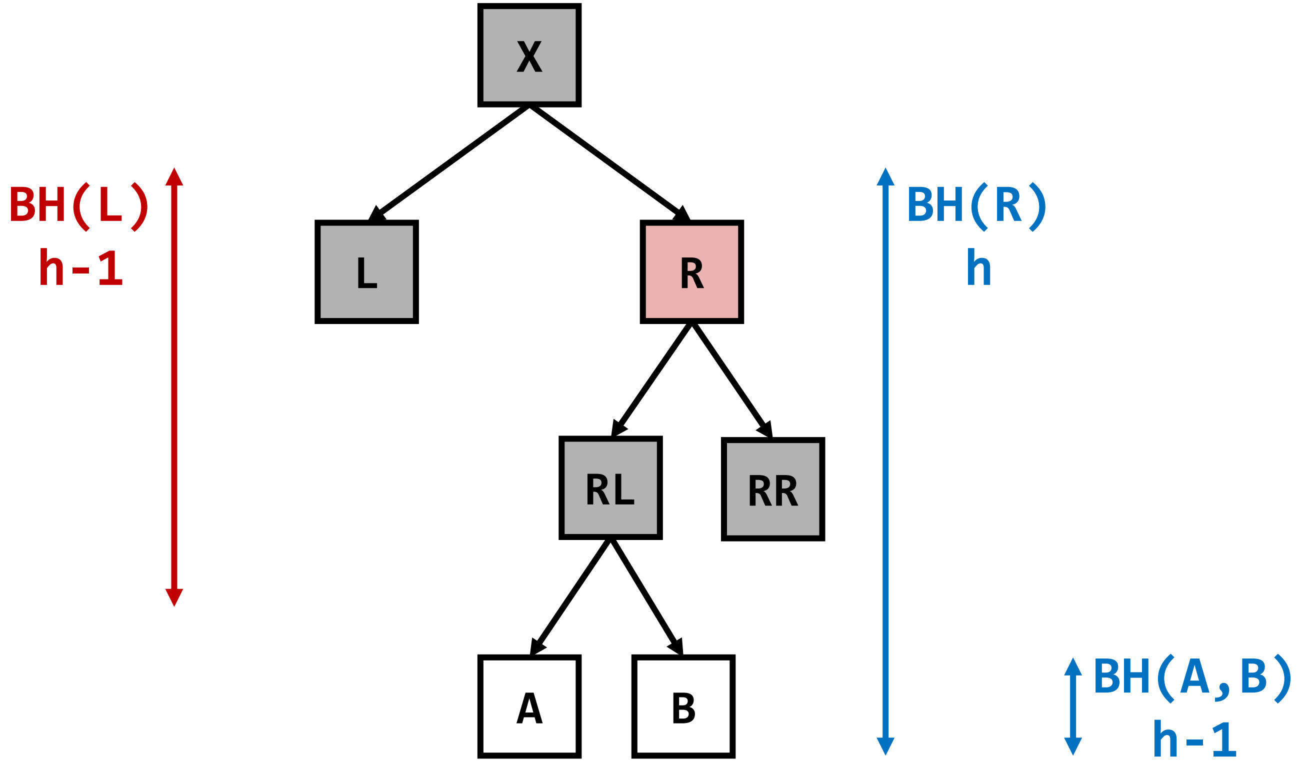
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)`** **BH(L)+1=BH(R)**, **L** is **black**, and **R** is **red**



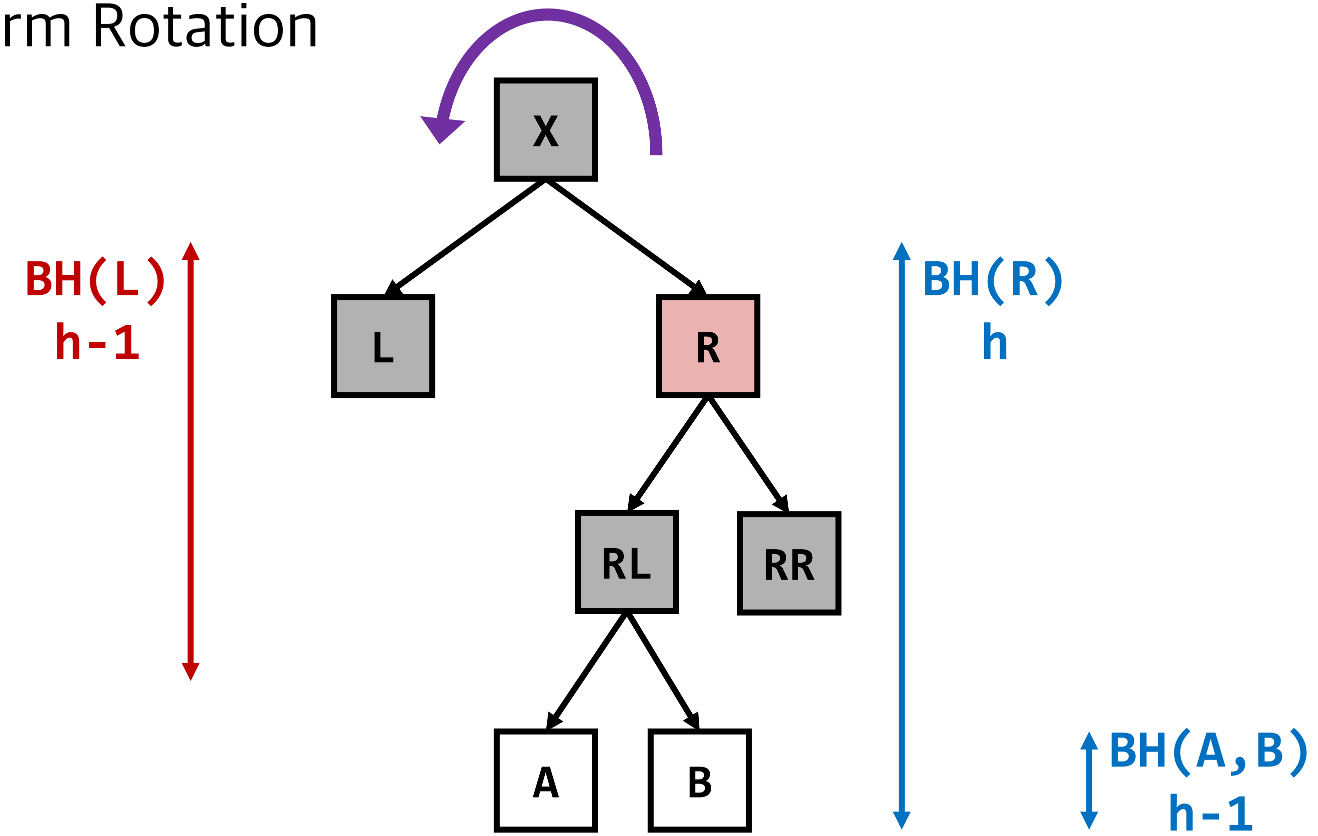
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)` BH(L)+1=BH(R)**, **L** is **black**, and **R** is **red**



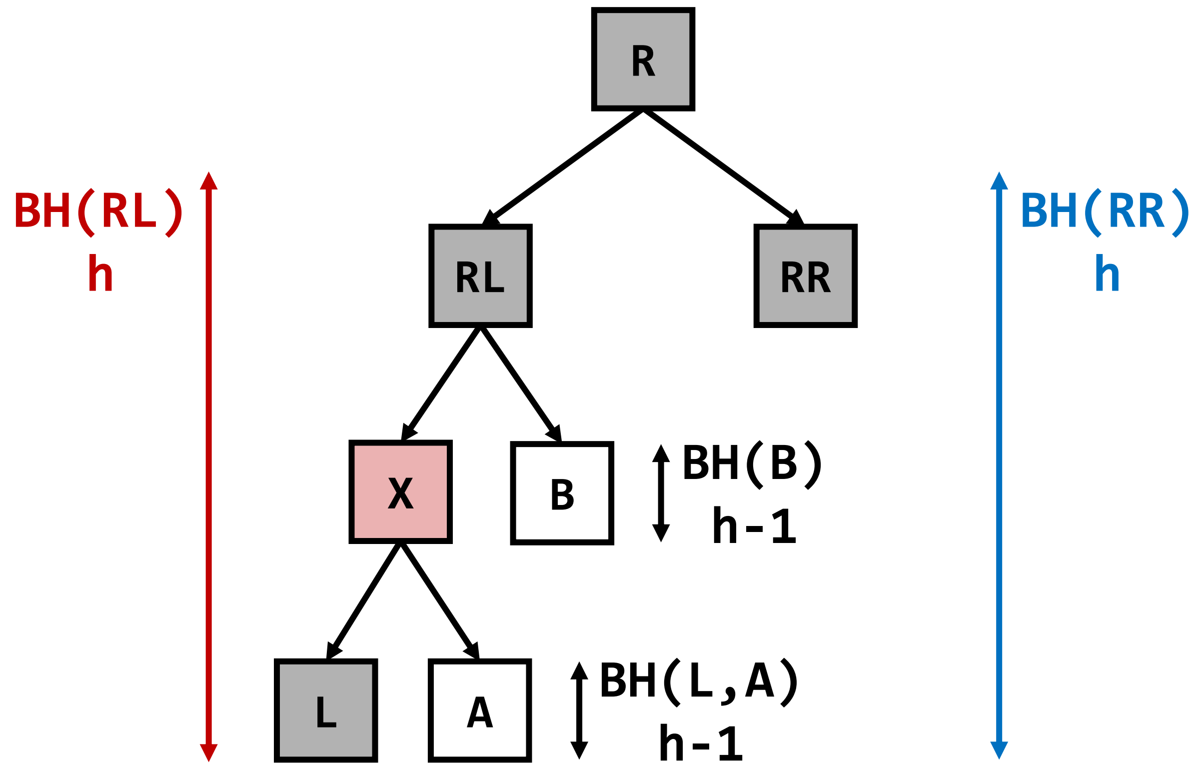
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **(Case 2) BH(L)+1=BH(R)**, **L** is **black**, and **R** is **red**
  - **(Solution)** Perform Rotation



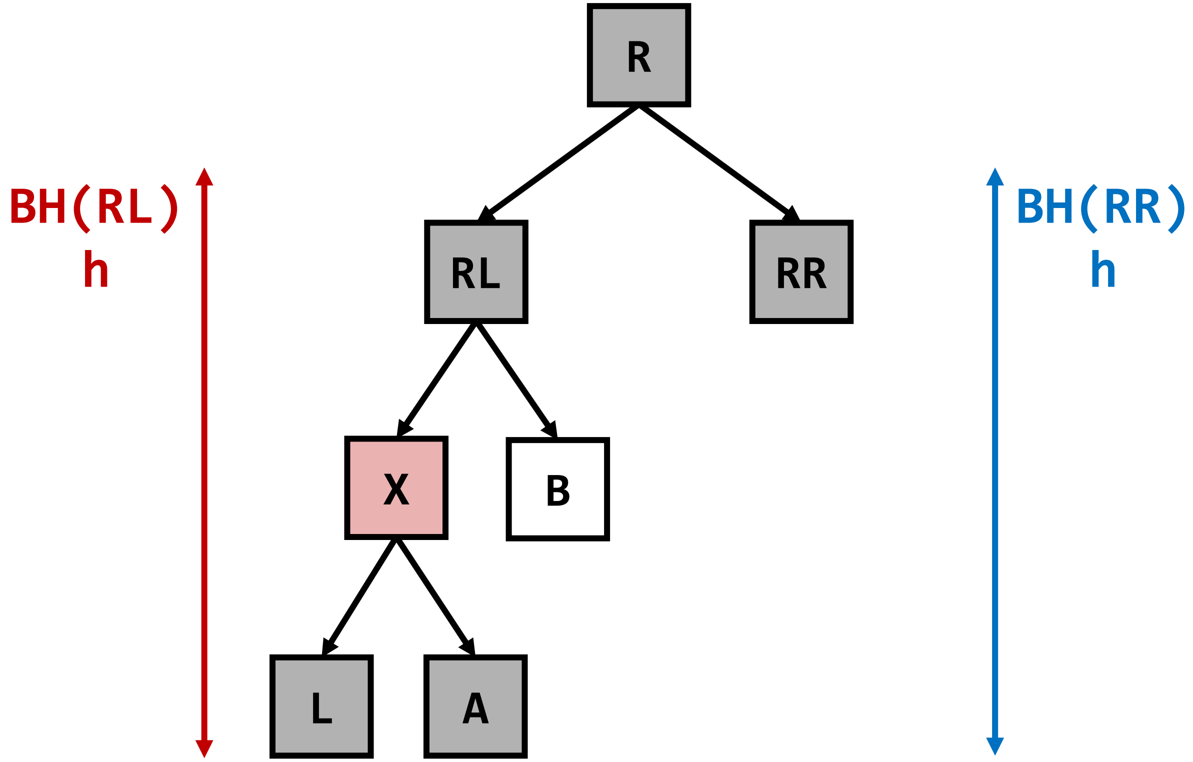
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)` `BH(L)+1=BH(R)`**, **`L`** is **black**, and **`R`** is **red**
  - **`(Solution)`** Perform Rotation



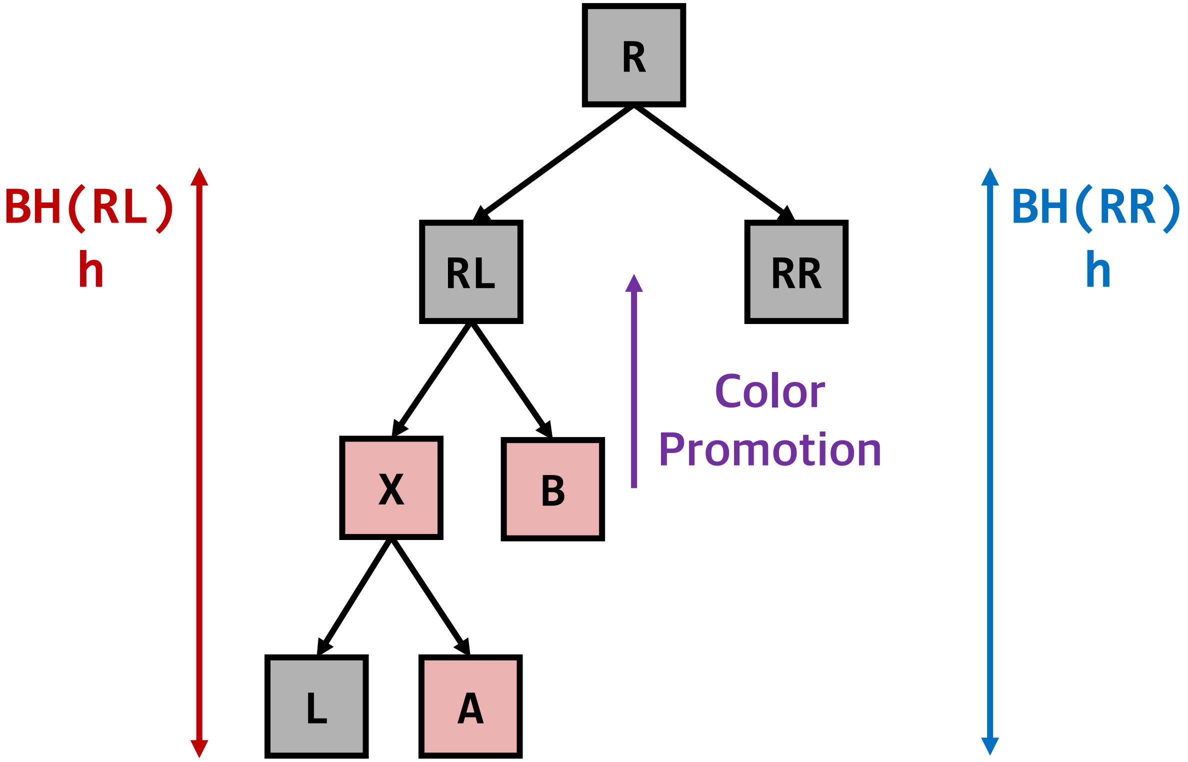
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **(Case 2) BH(L)+1=BH(R)**, **L** is **black**, and **R** is **red** + **(a) A** is **black**
  - **(Solution)** Perform Rotation + **(a)** Nothing to do



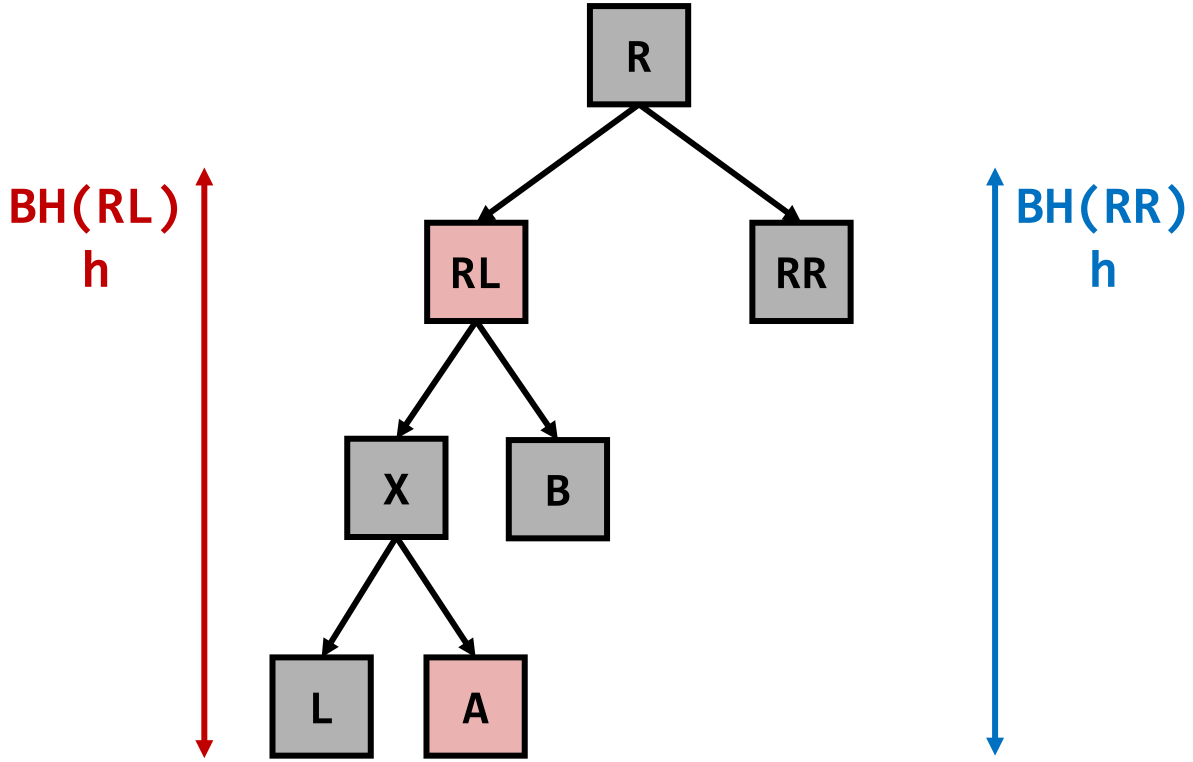
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)`** **`BH(L)+1=BH(R)`**, **`L`** is **black**, and **`R`** is **red** + **`(b)`** **`A`** & **`B`** are **red**
  - **`(Solution)`** Perform Rotation + **`(b)`** Color Promotion



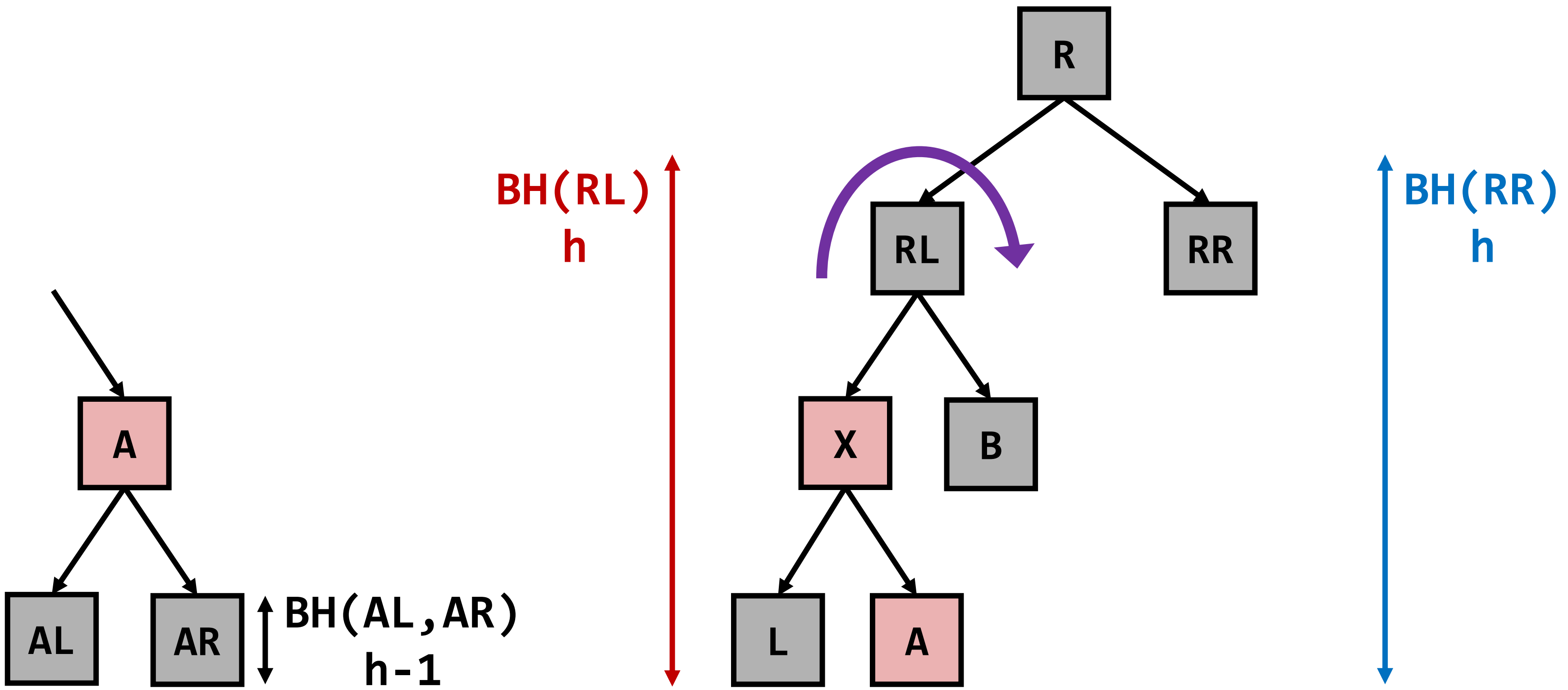
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 2)`** **`BH(L)+1=BH(R)`**, **`L`** is **black**, and **`R`** is **red** + **`(b)`** **`A`** & **`B`** are **red**
  - **`(Solution)`** Perform Rotation + **`(b)`** Color Promotion



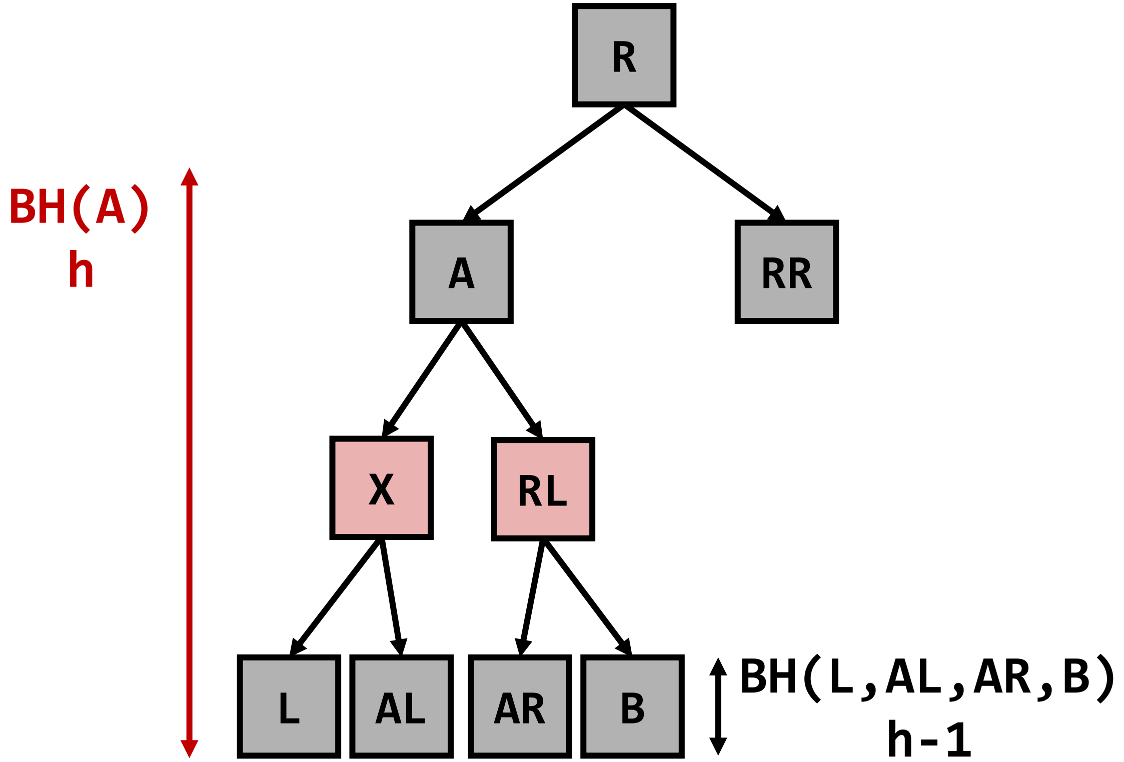
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **(Case 2) BH(L)+1=BH(R)**, **L** is **black**, and **R** is **red** + **(c) A** is **red** & **B** is **black**
  - **(Solution)** Perform Rotation + **(c)** Perform Rotation again



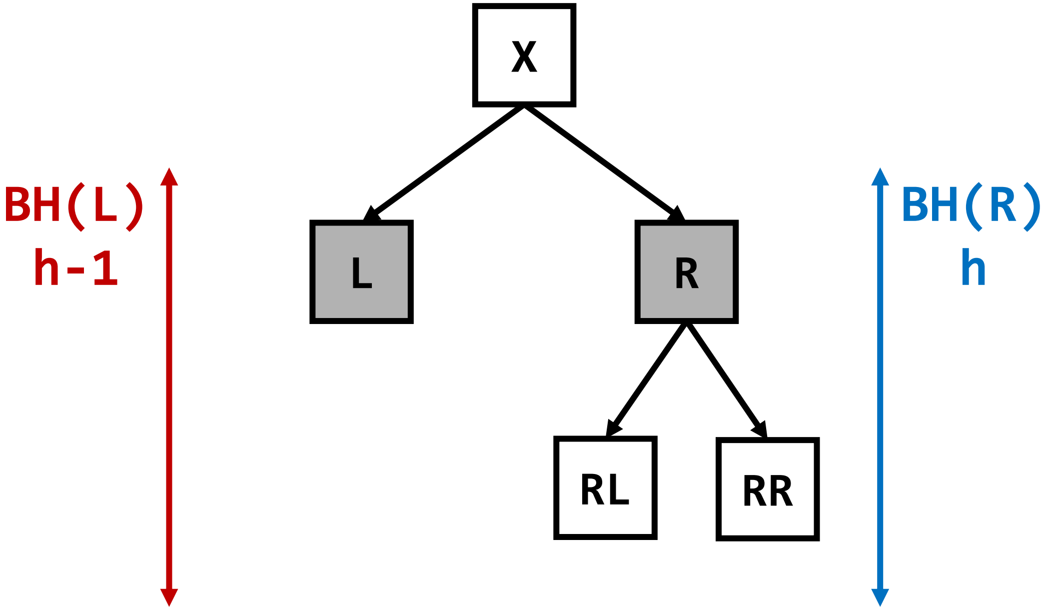
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - `(Case 2)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **red** + `(c)` `A` is **red** & `B` is **black**
  - `(Solution)` Perform Rotation + `(c)` Perform Rotation again



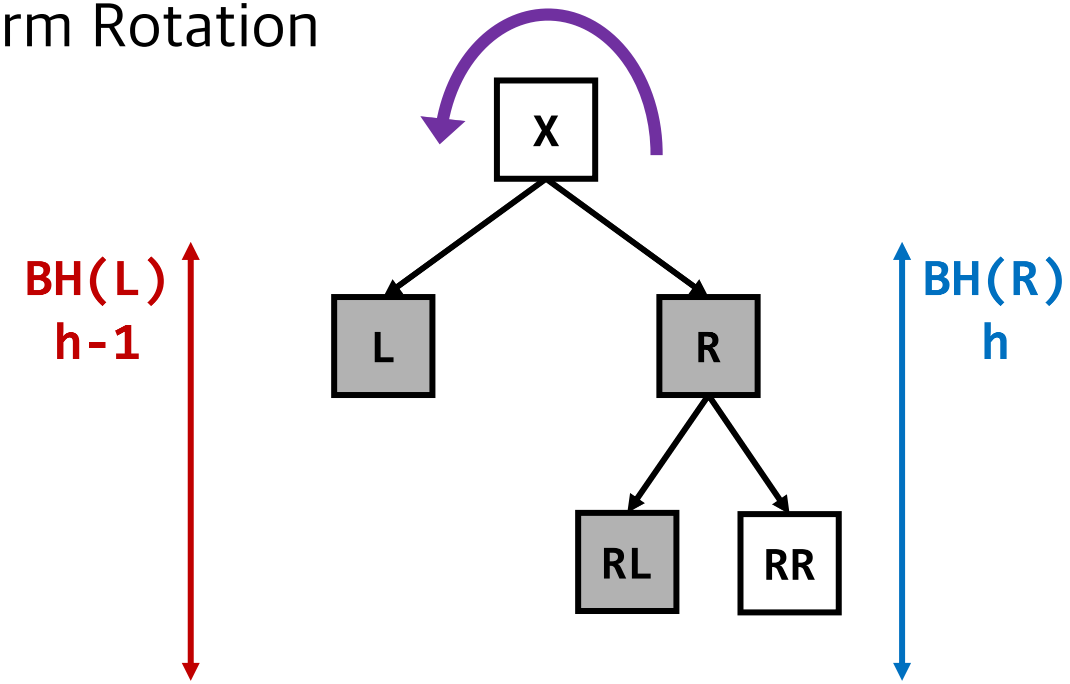
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 3)`** **`BH(L)+1=BH(R)`**, **`L`** is **black**, and **`R`** is **black**



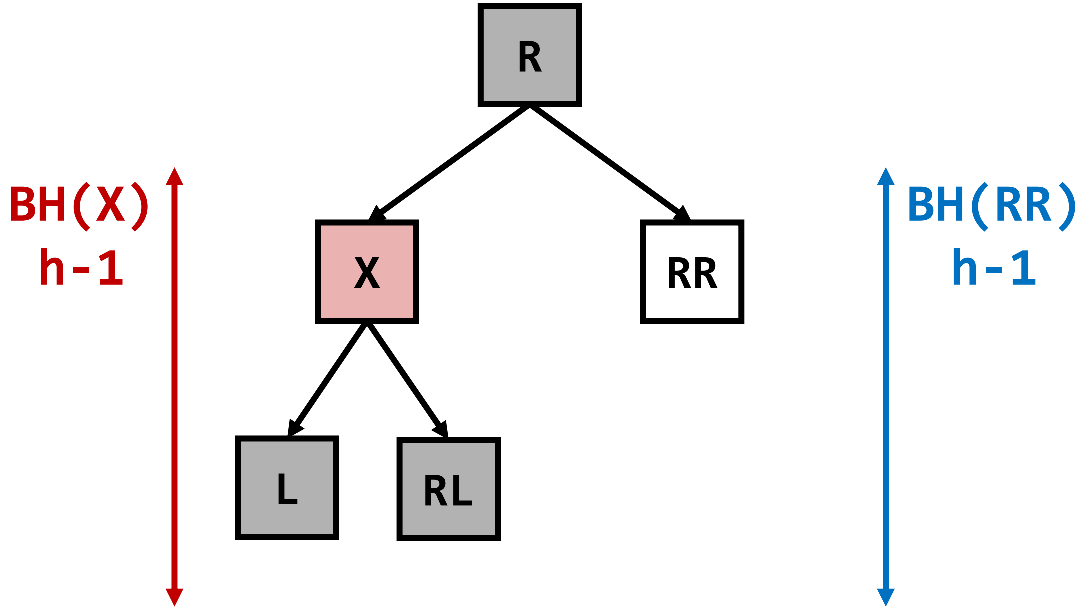
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **`(Case 3)` `BH(L)+1=BH(R)`**, **`L`** is **black**, and **`R`** is **black** + **`(a)`** **`RL`** is **black**
  - **`(Solution)`** Perform Rotation



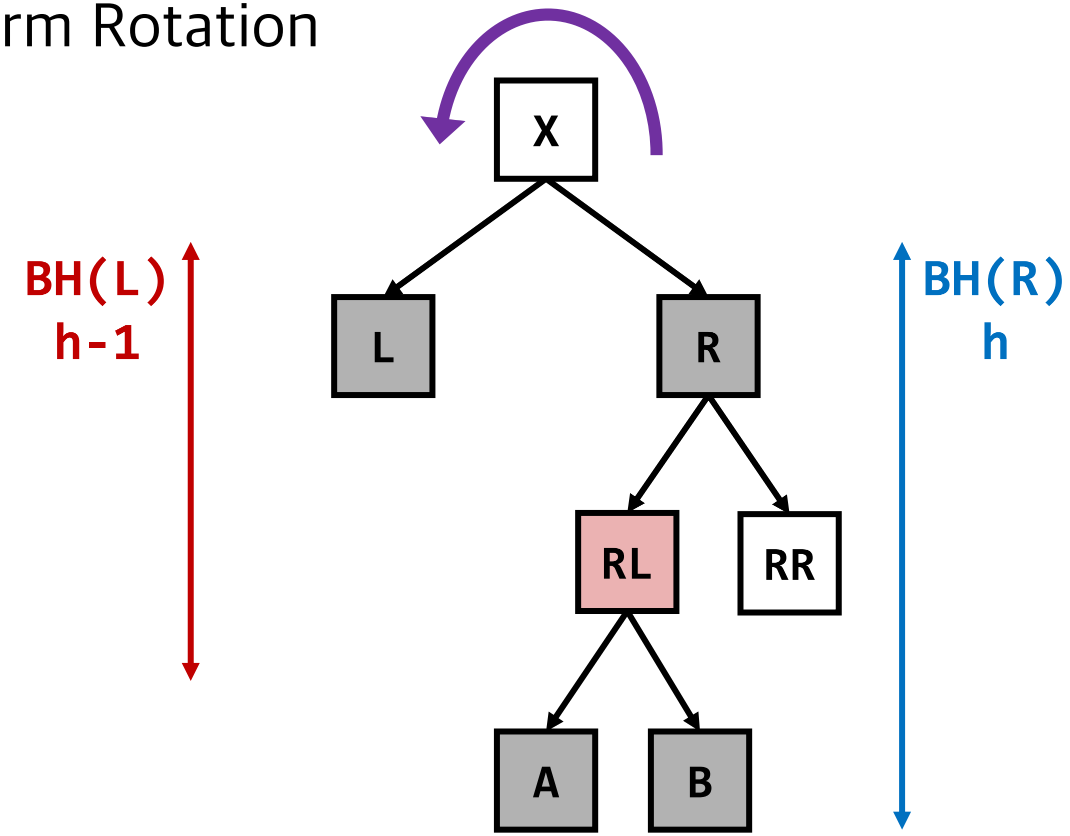
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - `(Case 3)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **black** + `(a)` `RL` is **black**
  - `(Solution)` Perform Rotation



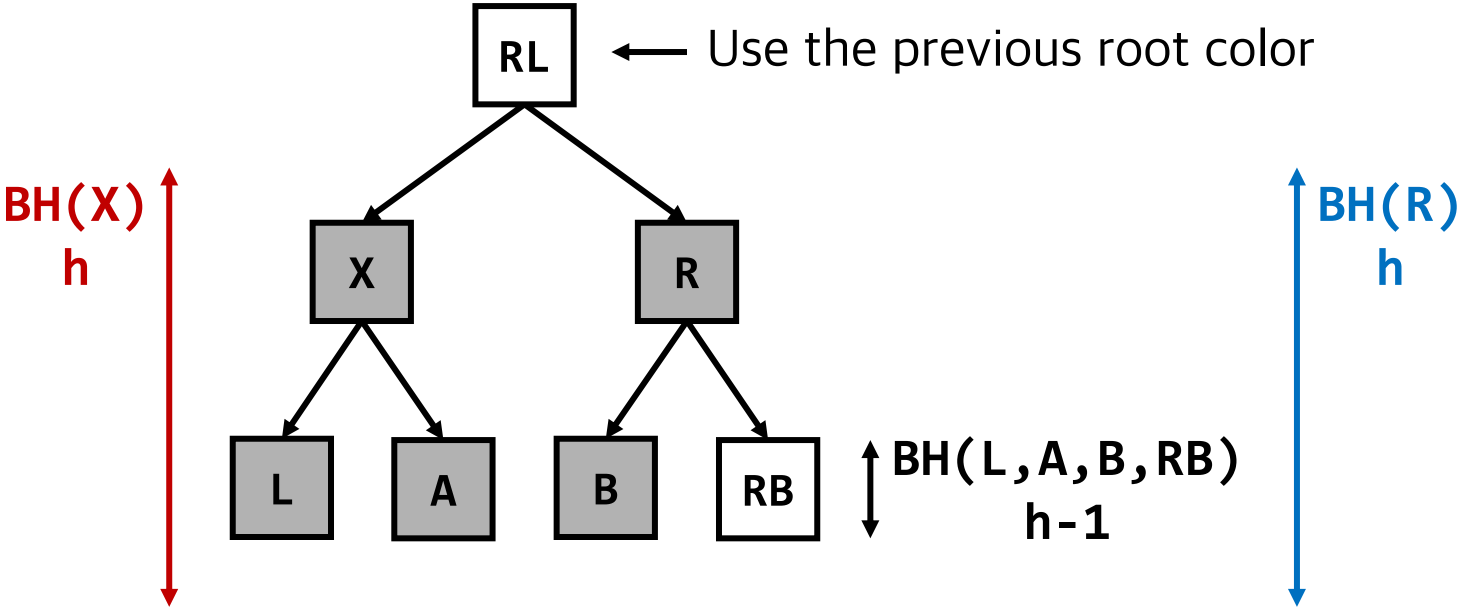
Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **(Case 3) BH(L)+1=BH(R)**, **L** is **black**, and **R** is **black** + **(b) RL** is **red**
  - **(Solution)** Perform Rotation



Red-Black Trees - Deletion

- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - **(Case 3) BH(L)+1=BH(R)**, **L** is **black**, and **R** is **black** + **(b) RL** is **red**
  - **(Solution)** Perform Rotation



Red-Black Trees - Deletion

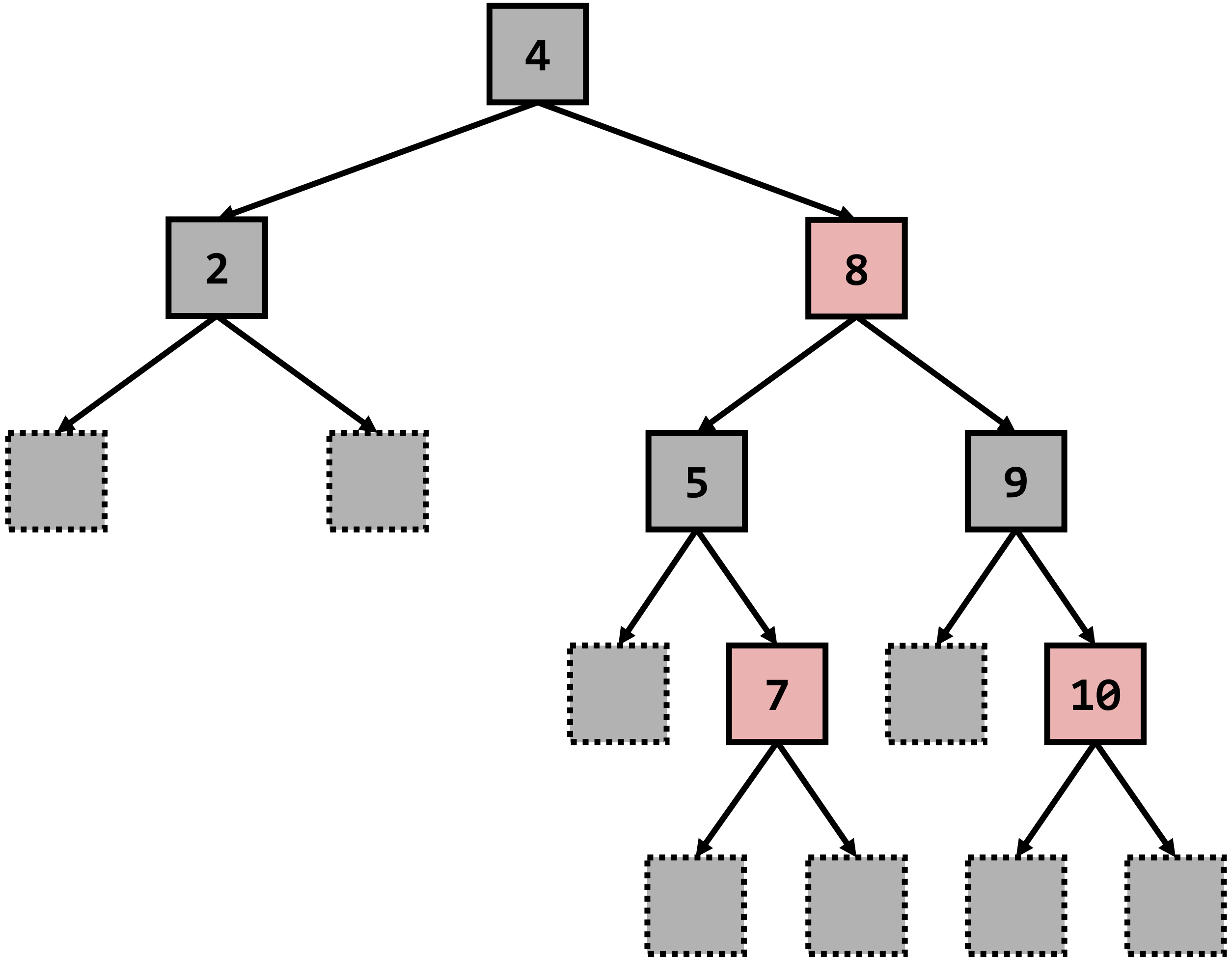
- How to modify the red-black tree?
  - Modify the tree in the **bottom-up** direction
  - `A` & `B` are left & right children of `RL`, respectively

  - `(Case 1)` `BH(L)+1=BH(R)` and `L` is **red**
  - `(Case 2)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **red** + `(a)` `A` is **black**
  - `(Case 2)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **red** + `(b)` `A` & `B` are **red**
  - `(Case 2)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **red** + `(c)` `A` is **red** & `B` is **black**
  - `(Case 3)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **black** + `(a)` `RL` is **black**
  - `(Case 3)` `BH(L)+1=BH(R)`, `L` is **black**, and `R` is **black** + `(b)` `RL` is **red**

  - `(Case 0)` The root color is **red**

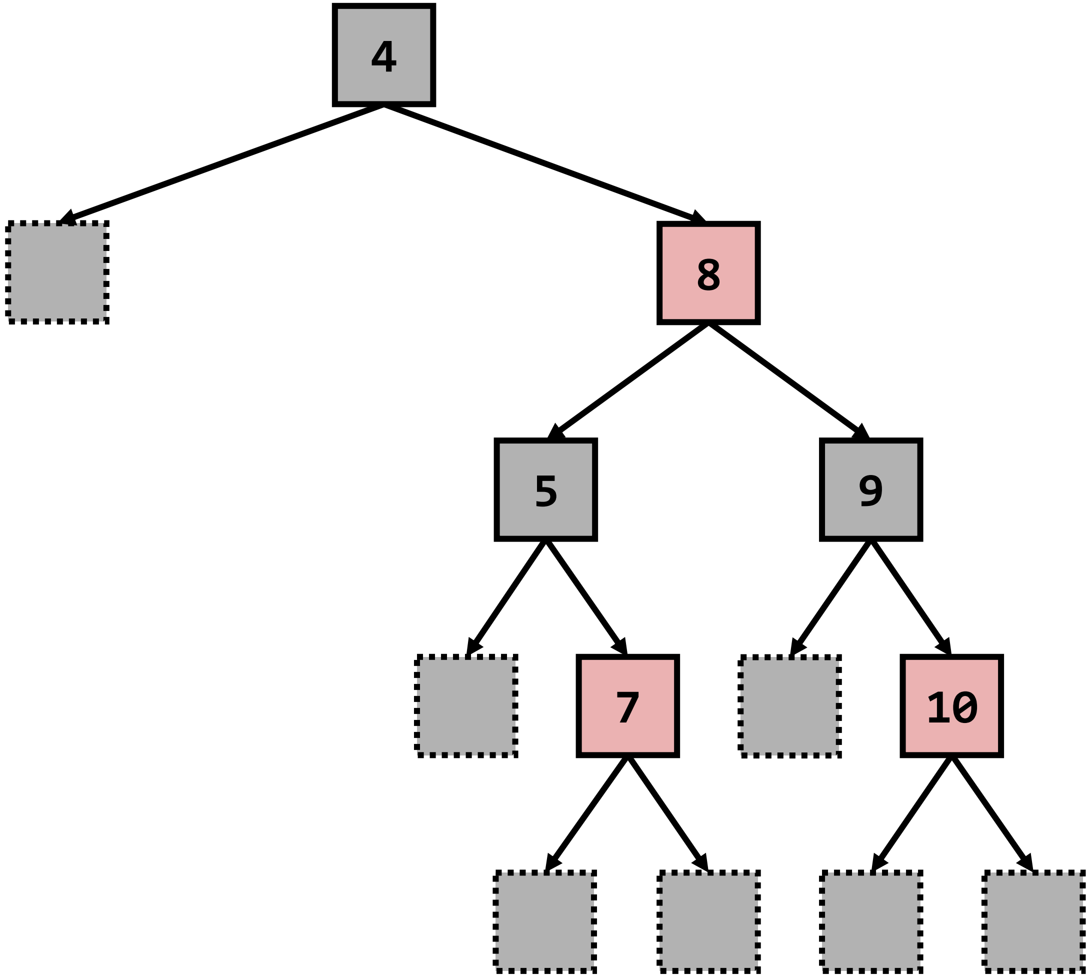
Red-Black Trees - Deletion

- Example - Delete 2



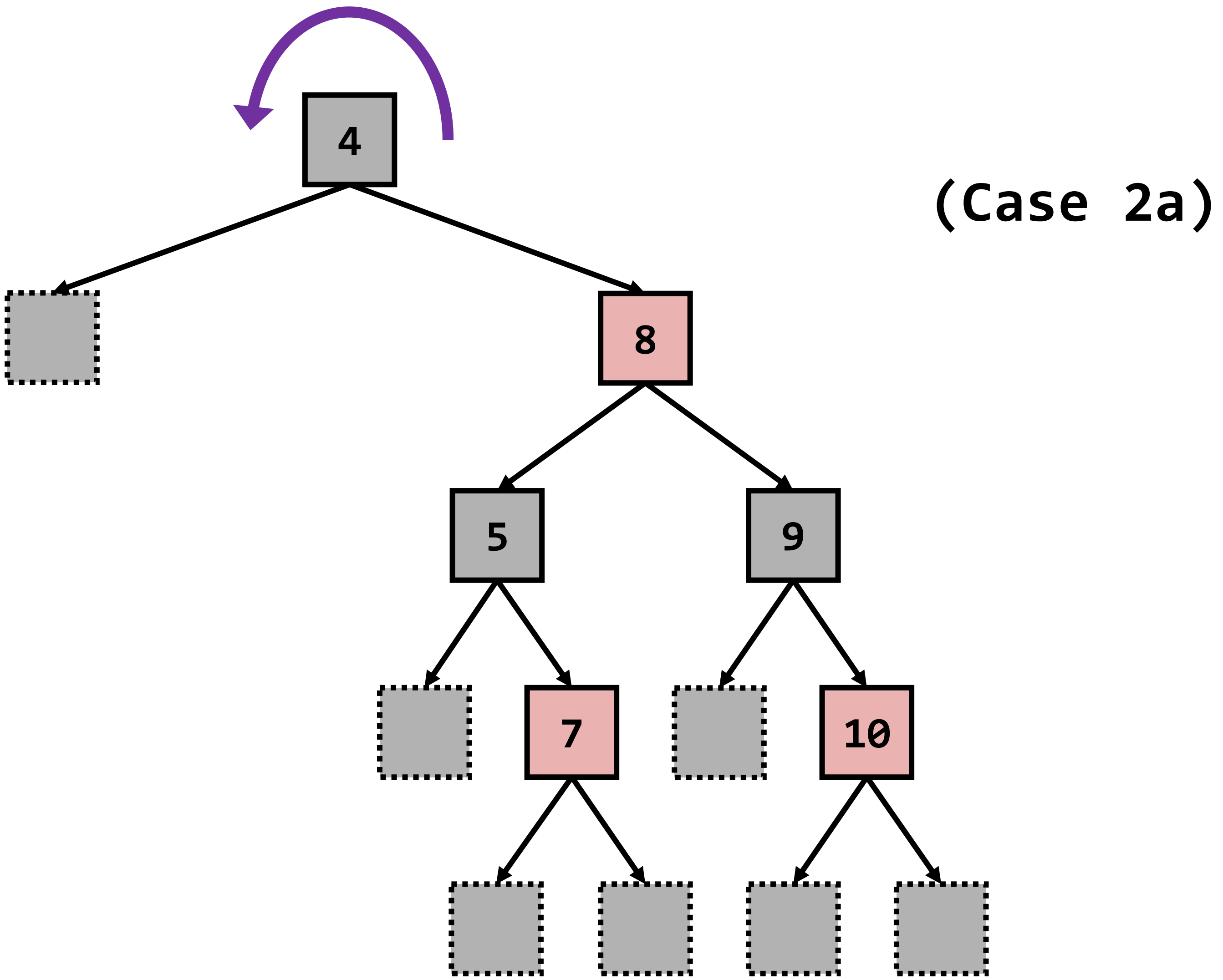
Red-Black Trees - Deletion

- Example - Delete 2



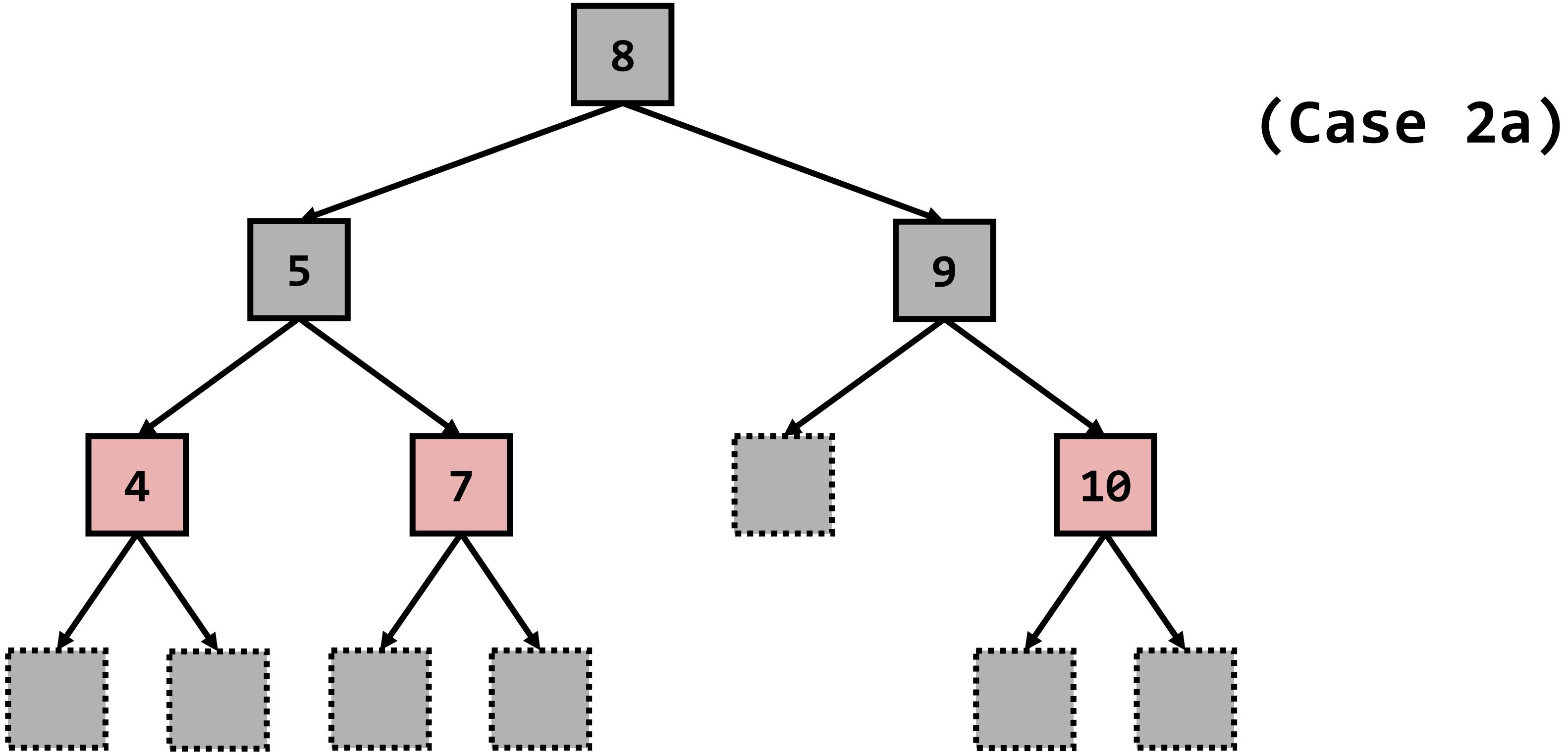
Red-Black Trees - Deletion

- Example - Delete 2



Red-Black Trees - Deletion

- Example - Delete 2



Any Questions?