Lecture 8

Red-Black Trees, Height Bound, Insertion

RB-Trees: Height Bound

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x,

RB-Trees: Height Bound

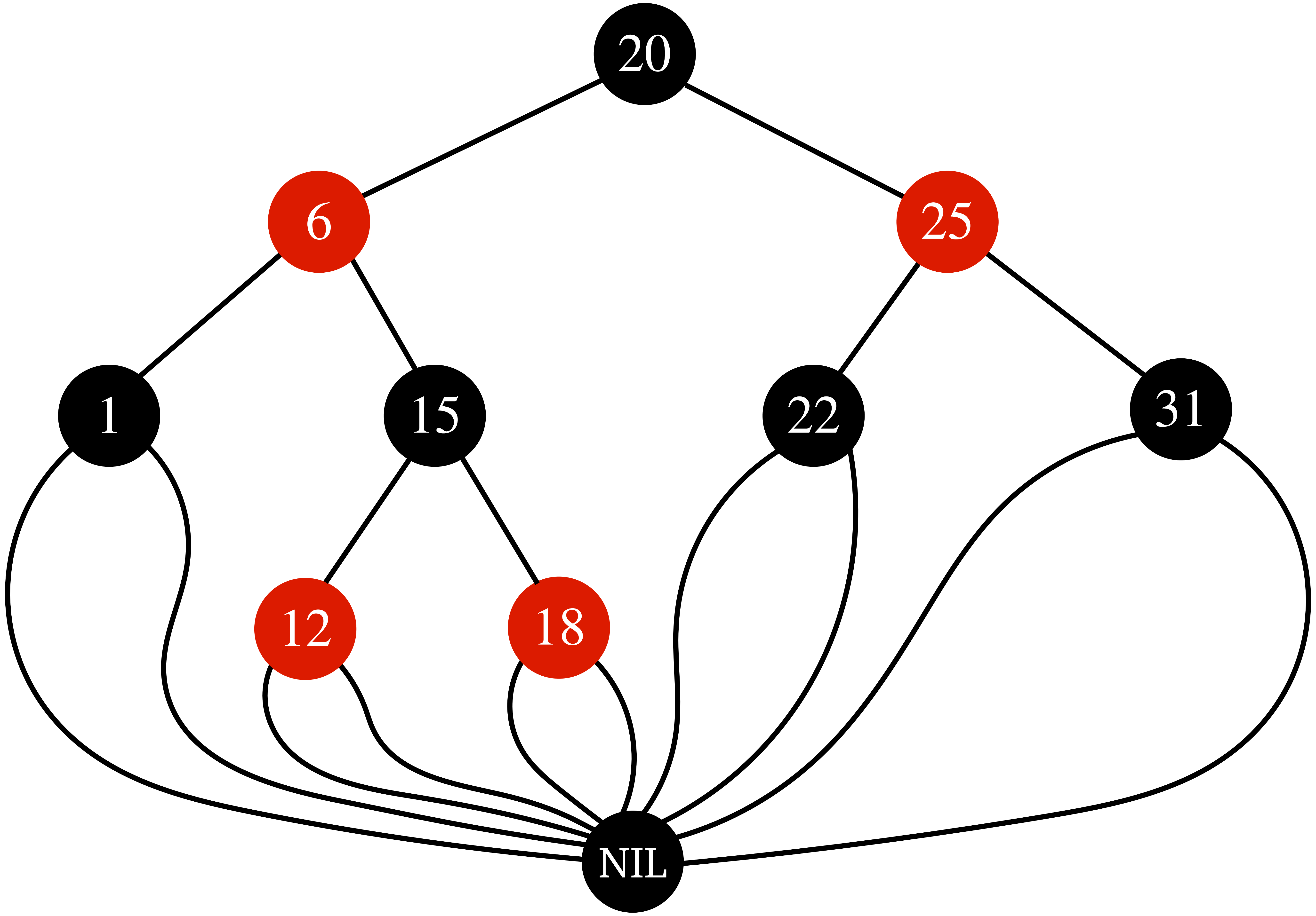
Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

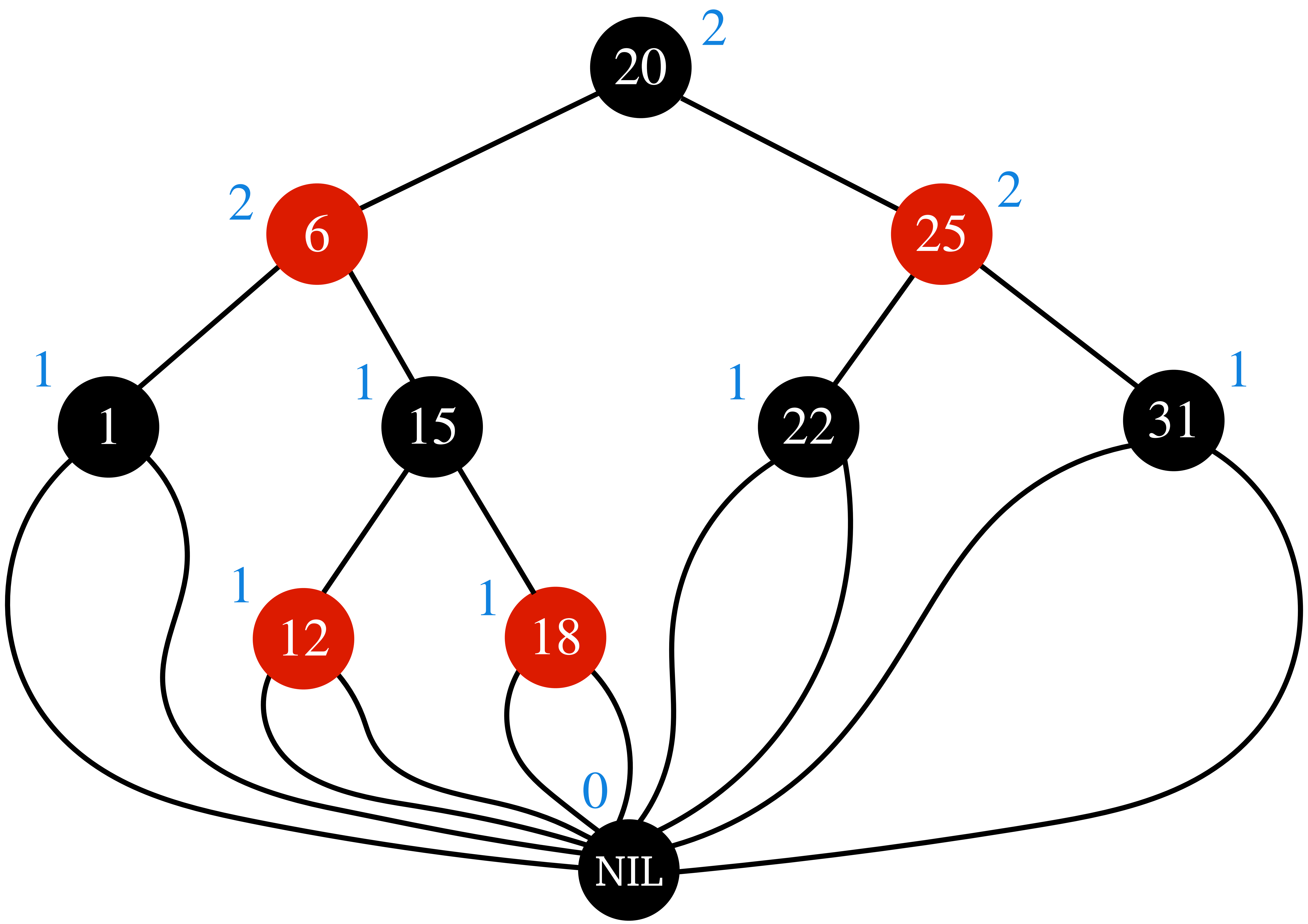
RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof:

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: For nodes of height 0, i.e., NIL node, the claim is trivially true as:

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: For nodes of height 0, i.e., NIL node, the claim is trivially true as:

- $bh(x) = 0$

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: For nodes of height 0, i.e., NIL node, the claim is trivially true as:

- $bh(x) = 0$
- Subtree at NIL node contains $0 = 2^0 - 1$ internal nodes.

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof:

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

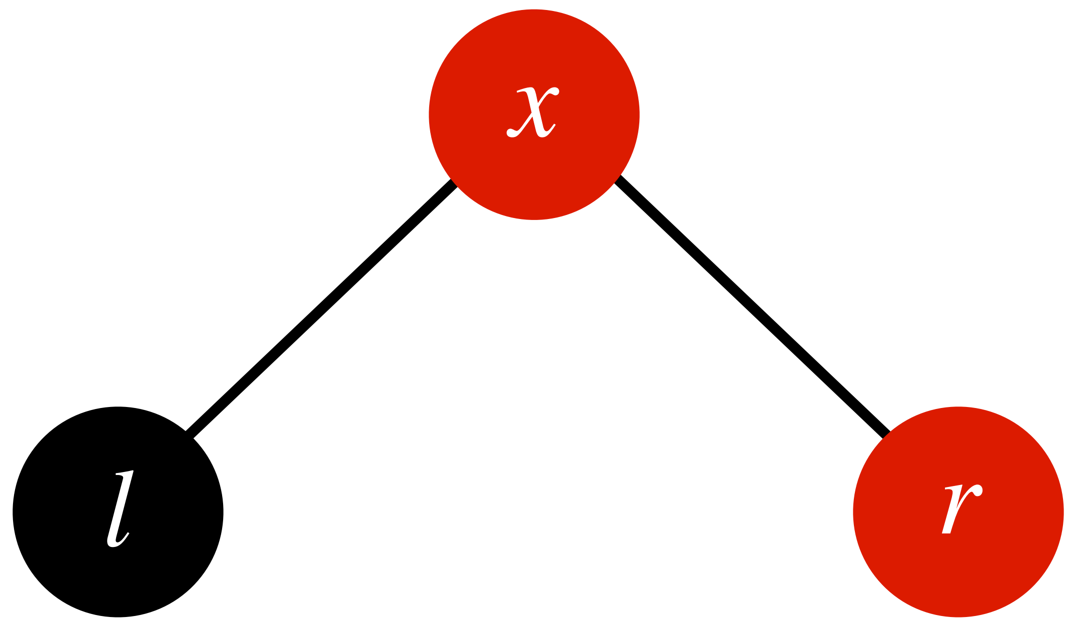
Let x be a node of height $i + 1$, and l and r be its children.

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.



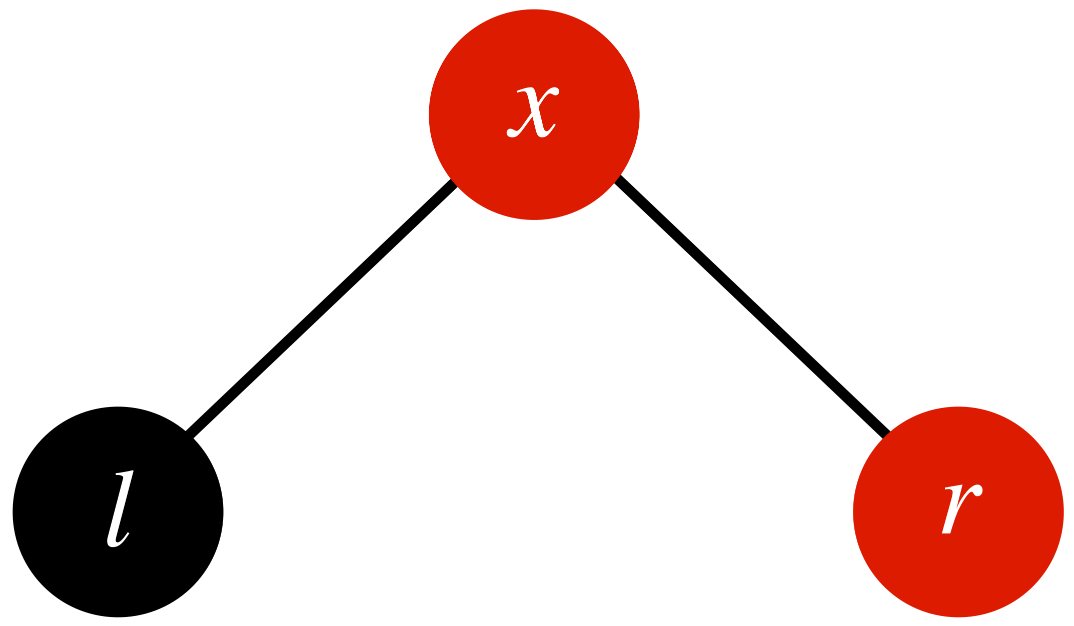
RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)



RB-Trees: Height Bound

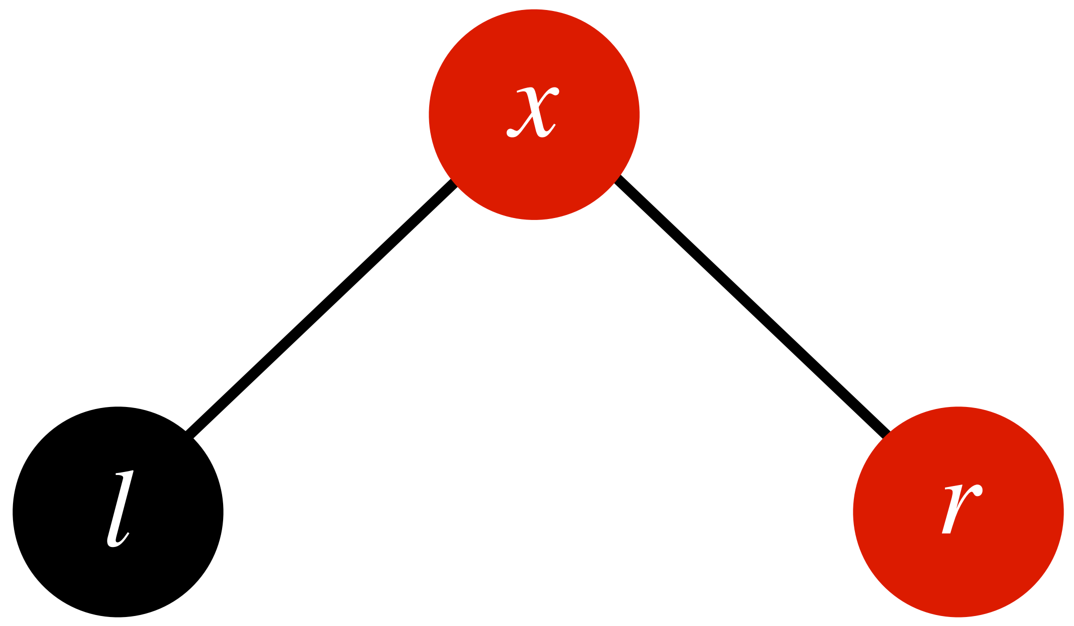
Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$= 1 +$ # internal nodes in subtree(l), subtree(r)



RB-Trees: Height Bound

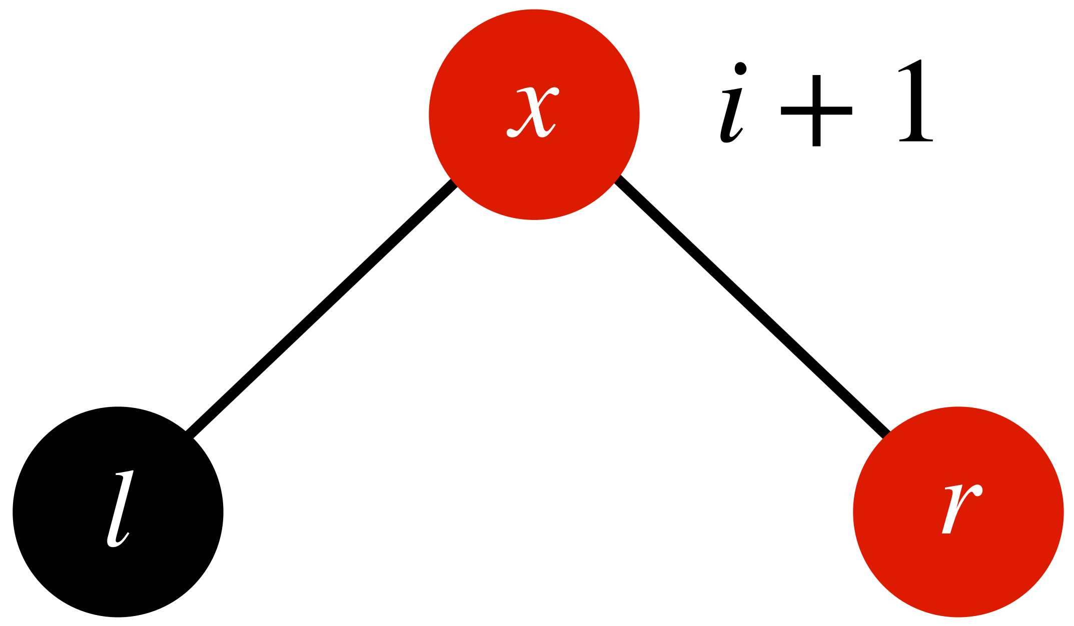
Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$= 1 +$ # internal nodes in subtree(l), subtree(r)



RB-Trees: Height Bound

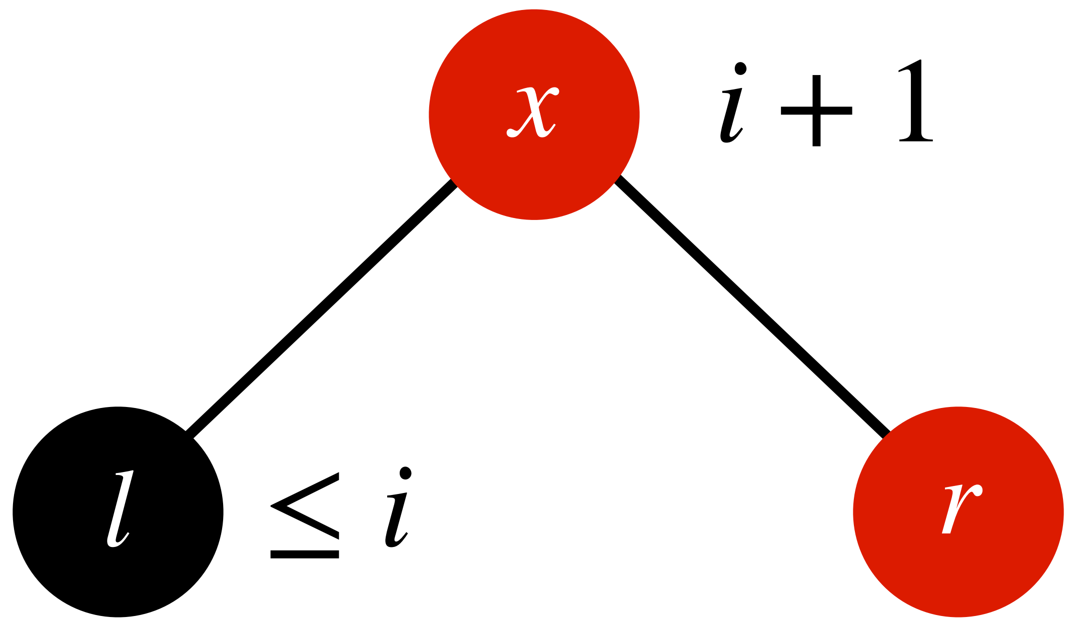
Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$= 1 +$ # internal nodes in subtree(l), subtree(r)



RB-Trees: Height Bound

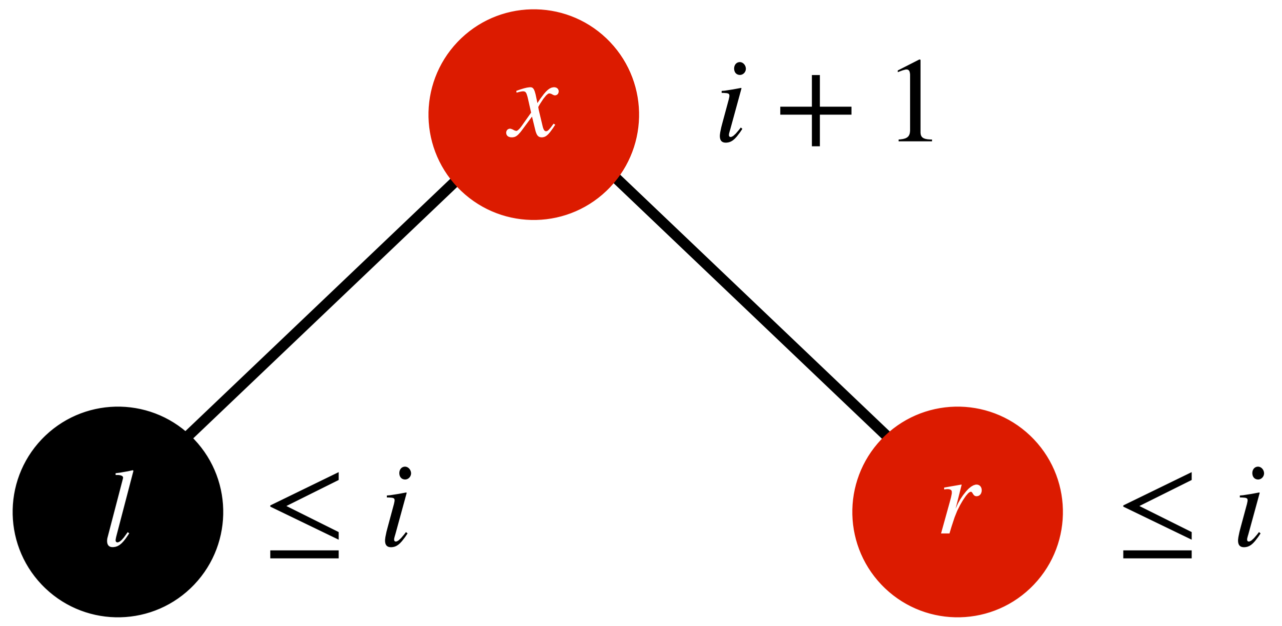
Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$= \ 1 +$ # internal nodes in subtree(l), subtree(r)



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

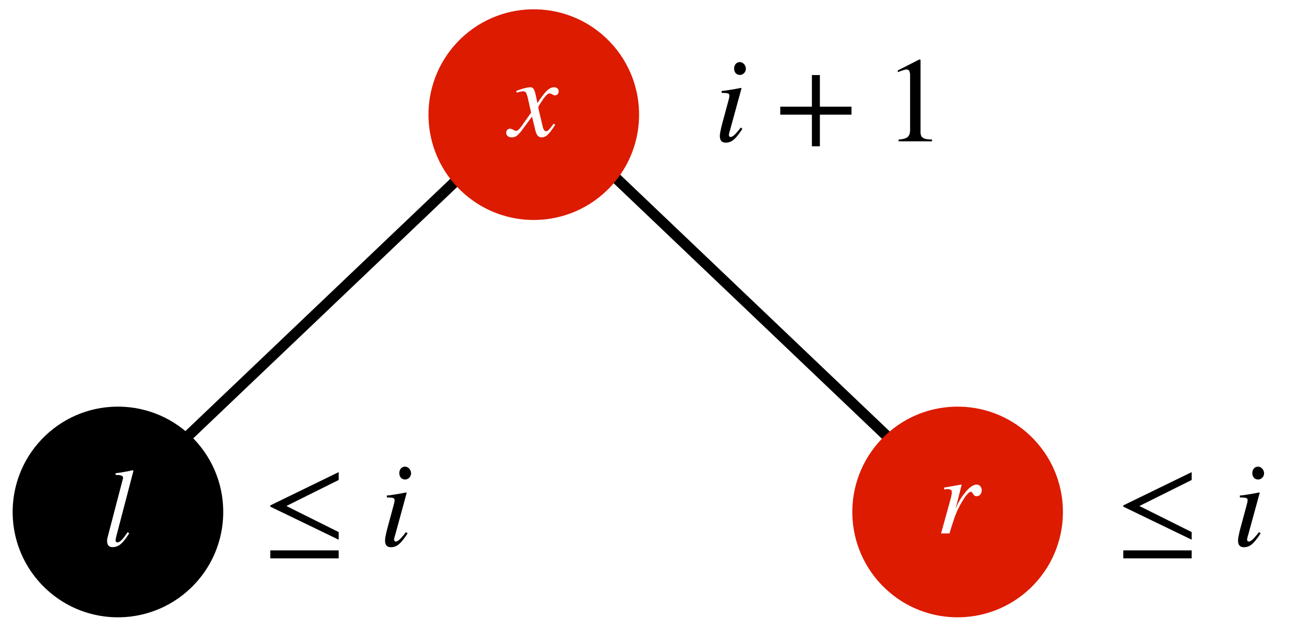
Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$= 1 +$ # internal nodes in subtree(l), subtree(r)

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

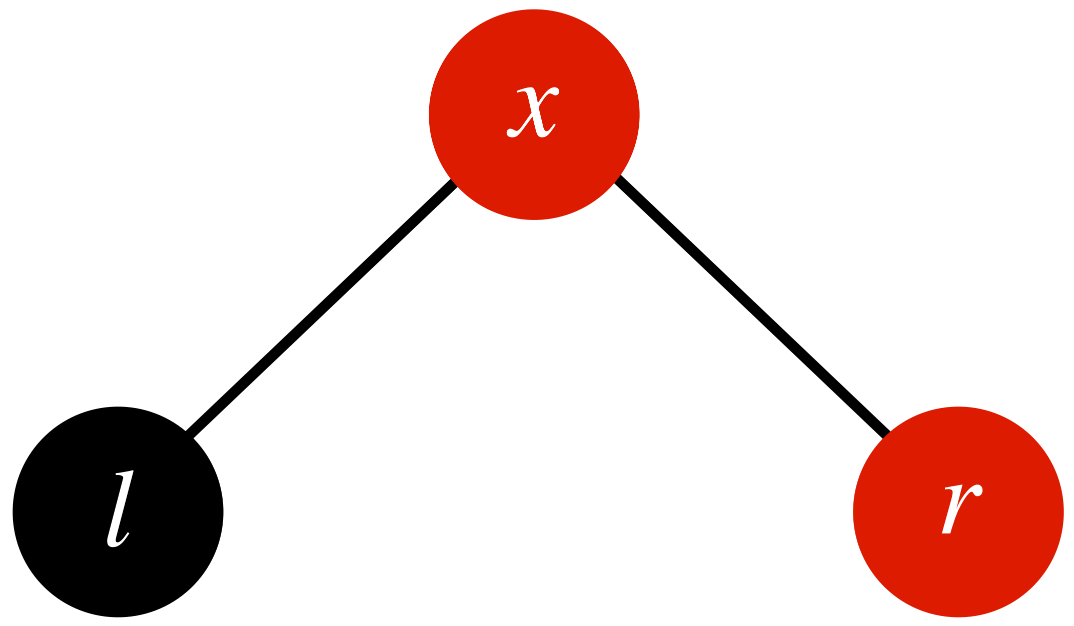
Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$= 1 +$ # internal nodes in subtree(l), subtree(r)

$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

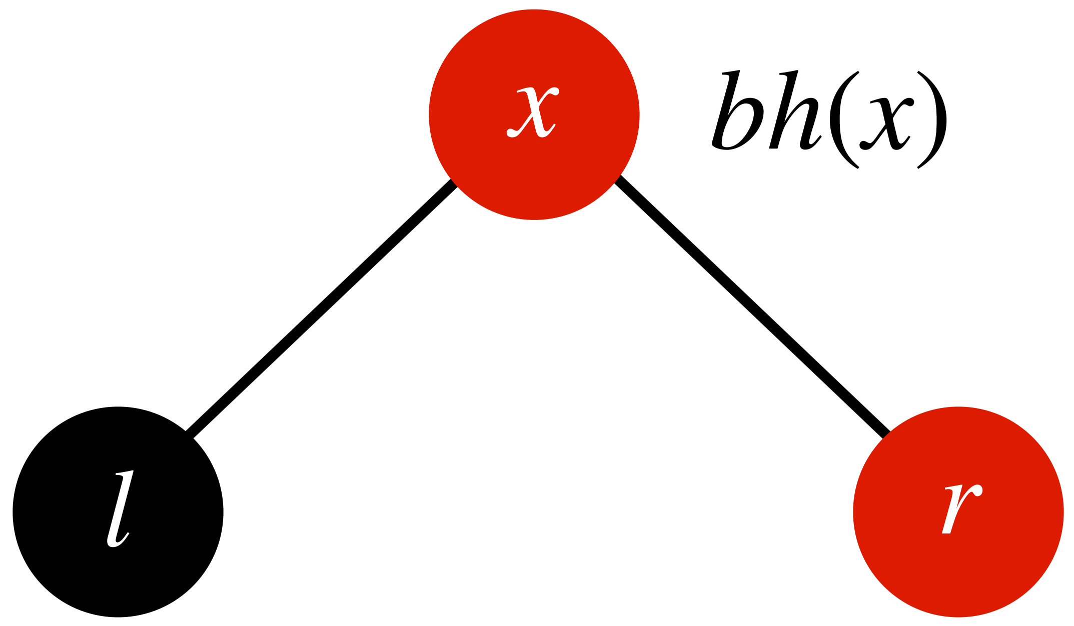
Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$$= 1 + \text{\# internal nodes in subtree}(l), \text{subtree}(r)$$

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

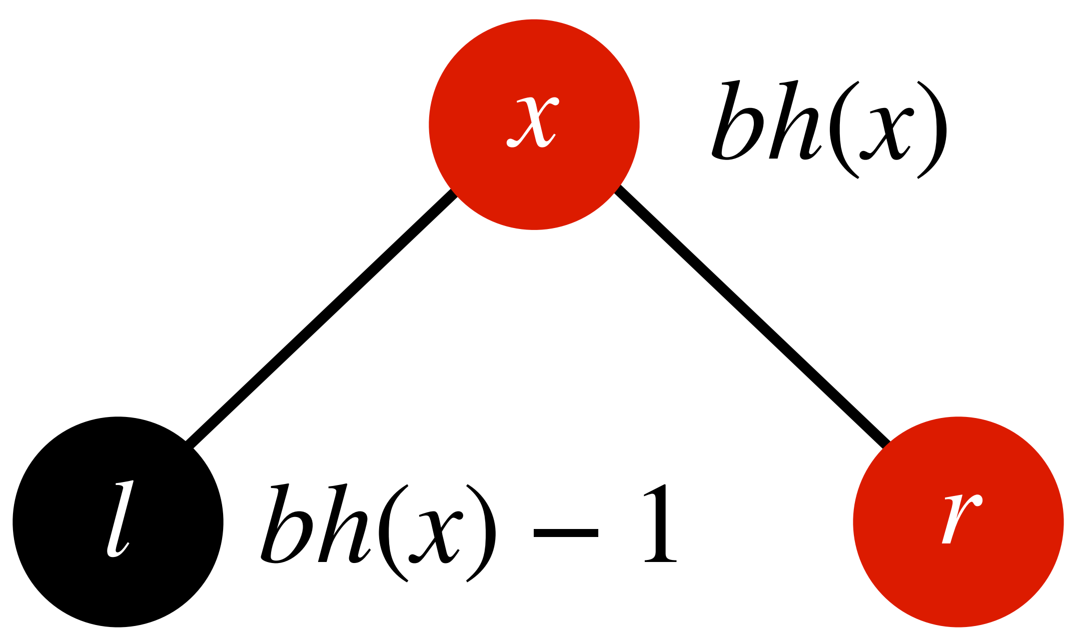
Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$$= 1 + \text{\# internal nodes in subtree}(l), \text{subtree}(r)$$

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

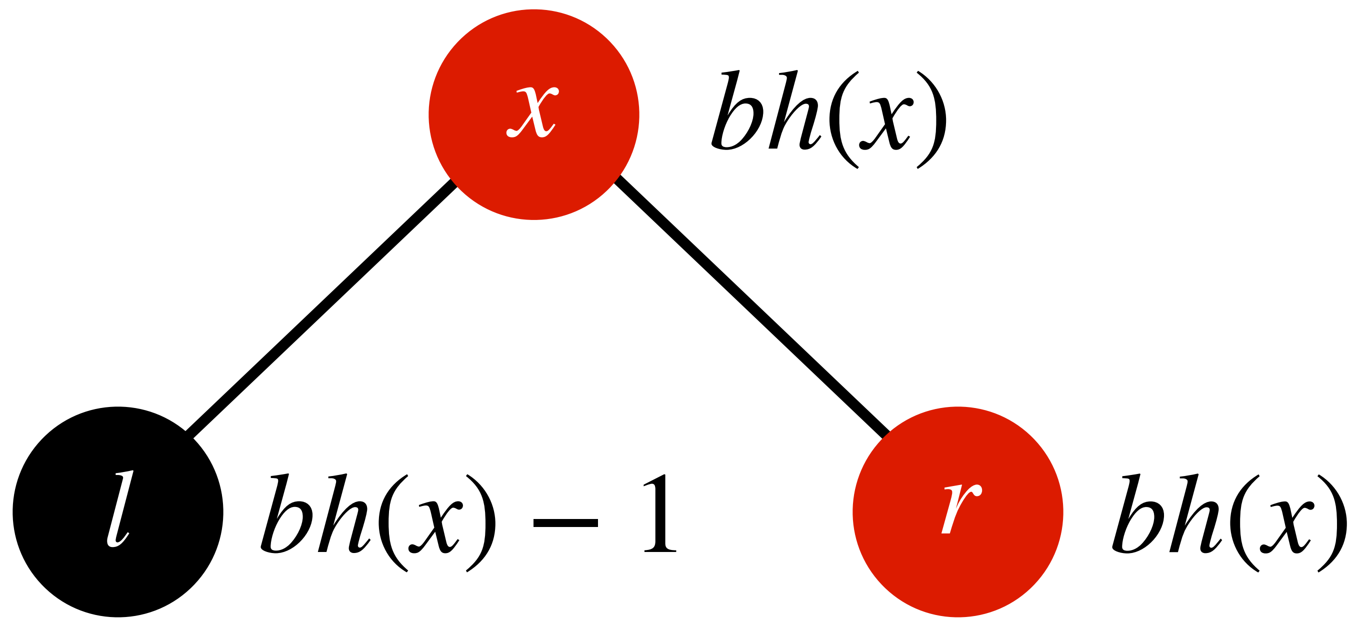
Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$$= 1 + \text{\# internal nodes in subtree}(l), \text{subtree}(r)$$

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

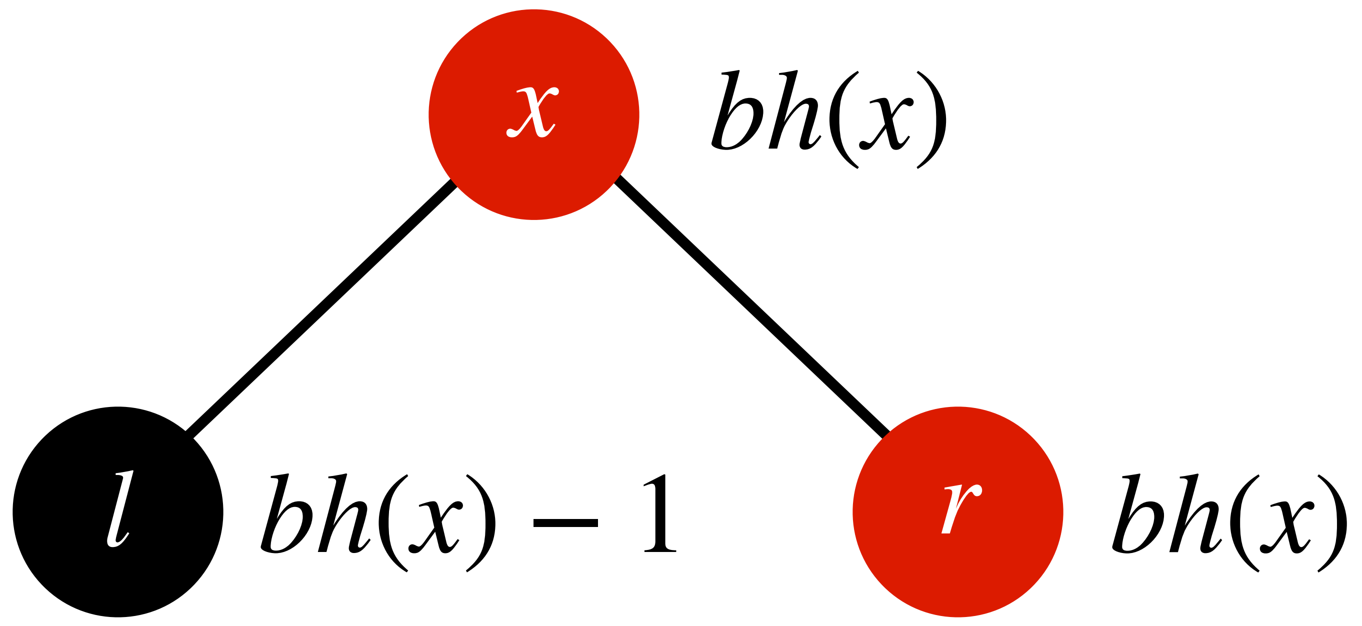
Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$$= 1 + \text{\# internal nodes in subtree}(l), \text{subtree}(r)$$

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$

$$\geq 1 + 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1$$



RB-Trees: Height Bound

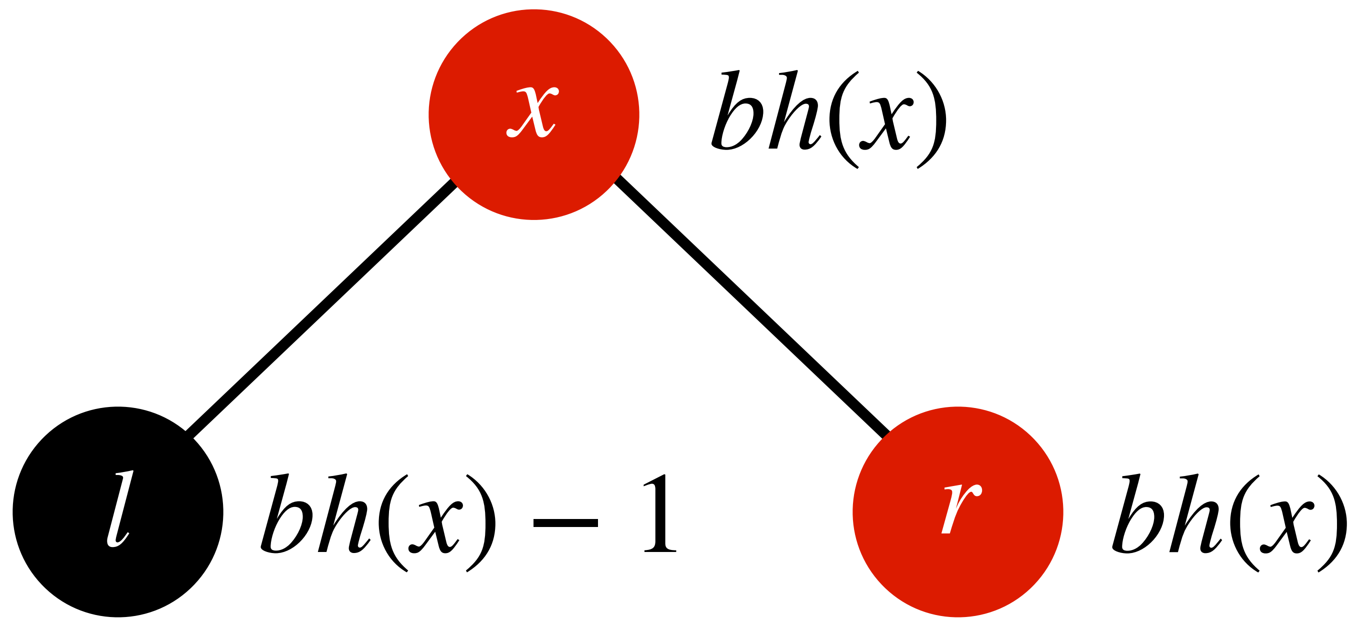
Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

internal nodes in subtree(x)

$$
\begin{aligned}
&= 1 + \text{\# internal nodes in subtree}(l), \text{subtree}(r) \\
&\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1 \\
&\geq 1 + 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1 \\
&= 2^{bh(x)} - 1
\end{aligned}
$$



RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

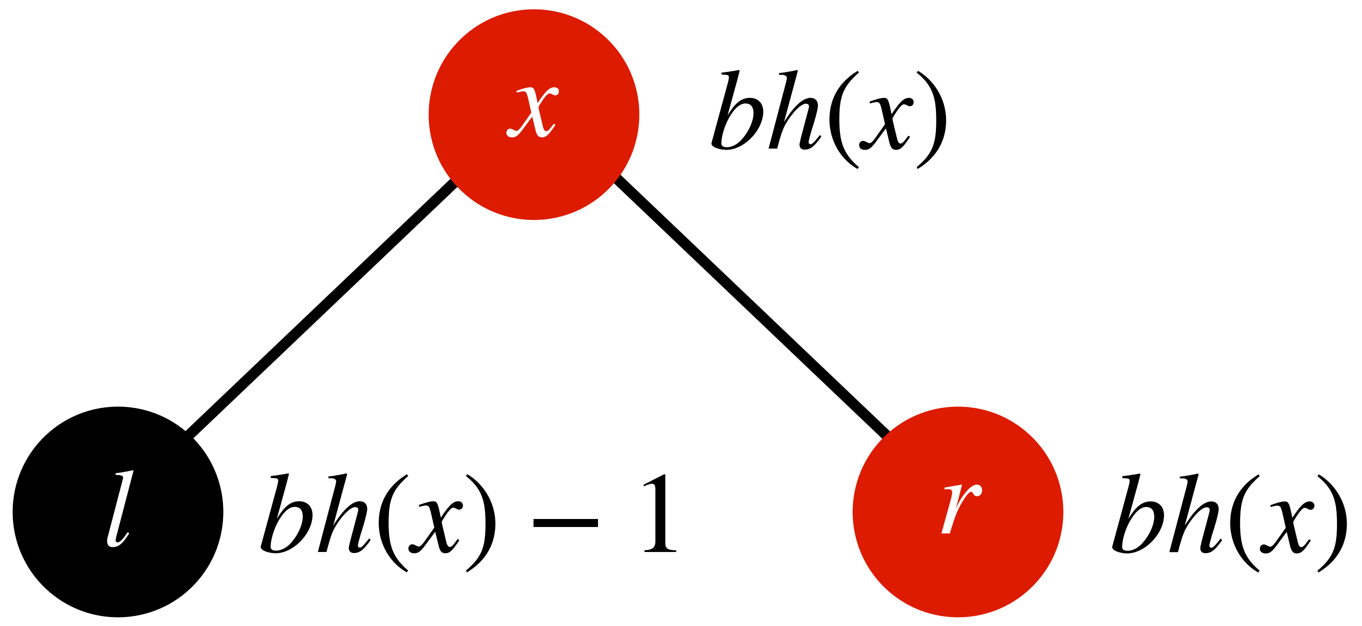
internal nodes in subtree(x)

$$= 1 + \text{\# internal nodes in subtree}(l), \text{subtree}(r)$$

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$

$$\geq 1 + 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1$$

$$= 2^{bh(x)} - 1$$



Since claim for height 0 nodes is already proven, this proves the claim for all nodes.

RB-Trees: Height Bound

Claim: In any RB-tree, every subtree rooted at any node, say x, contains at least $2^{bh(x)} - 1$ internal nodes.

Proof: Assume the claim is true for any node of height $\leq i$.

Let x be a node of height $i + 1$, and l and r be its children.

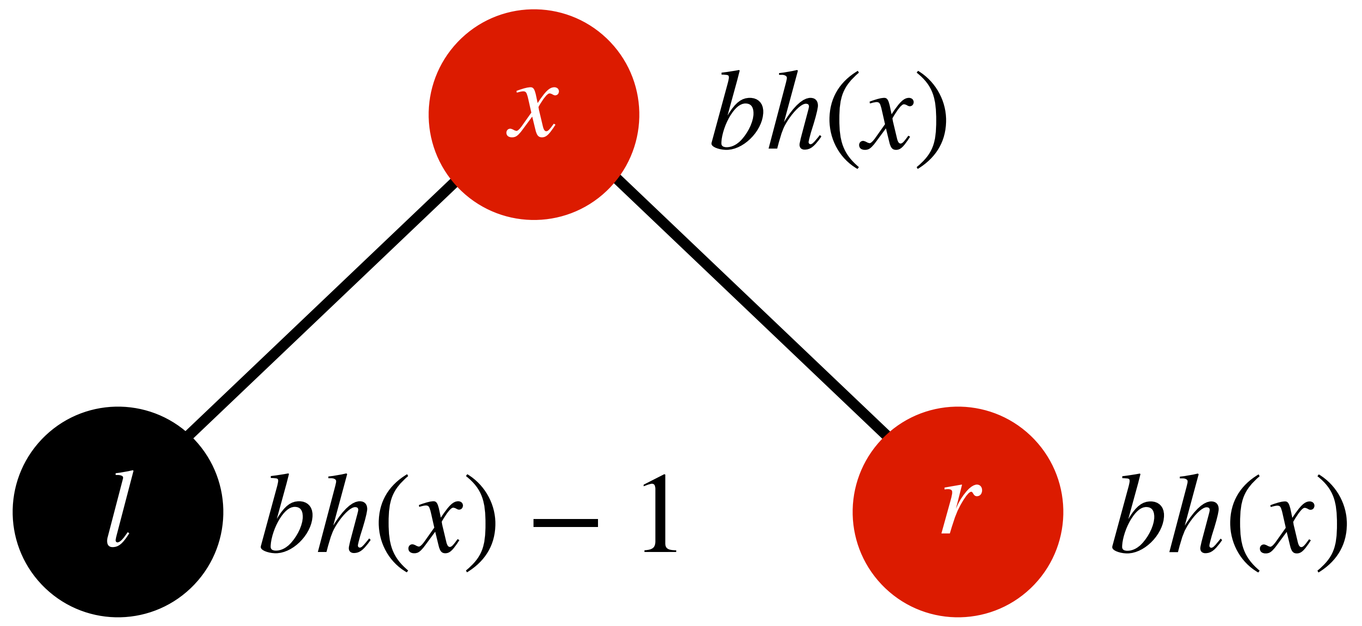
internal nodes in subtree(x)

$$= 1 + \#\text{ internal nodes in subtree}(l), \text{subtree}(r)$$

$$\geq 1 + 2^{bh(l)} - 1 + 2^{bh(r)} - 1$$

$$\geq 1 + 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1$$

$$= 2^{bh(x)} - 1$$



Since claim for height 0 nodes is already proven, this proves the claim for all nodes. ■

RB-Trees: Height Bound

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof:

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

RB-Trees: Height Bound

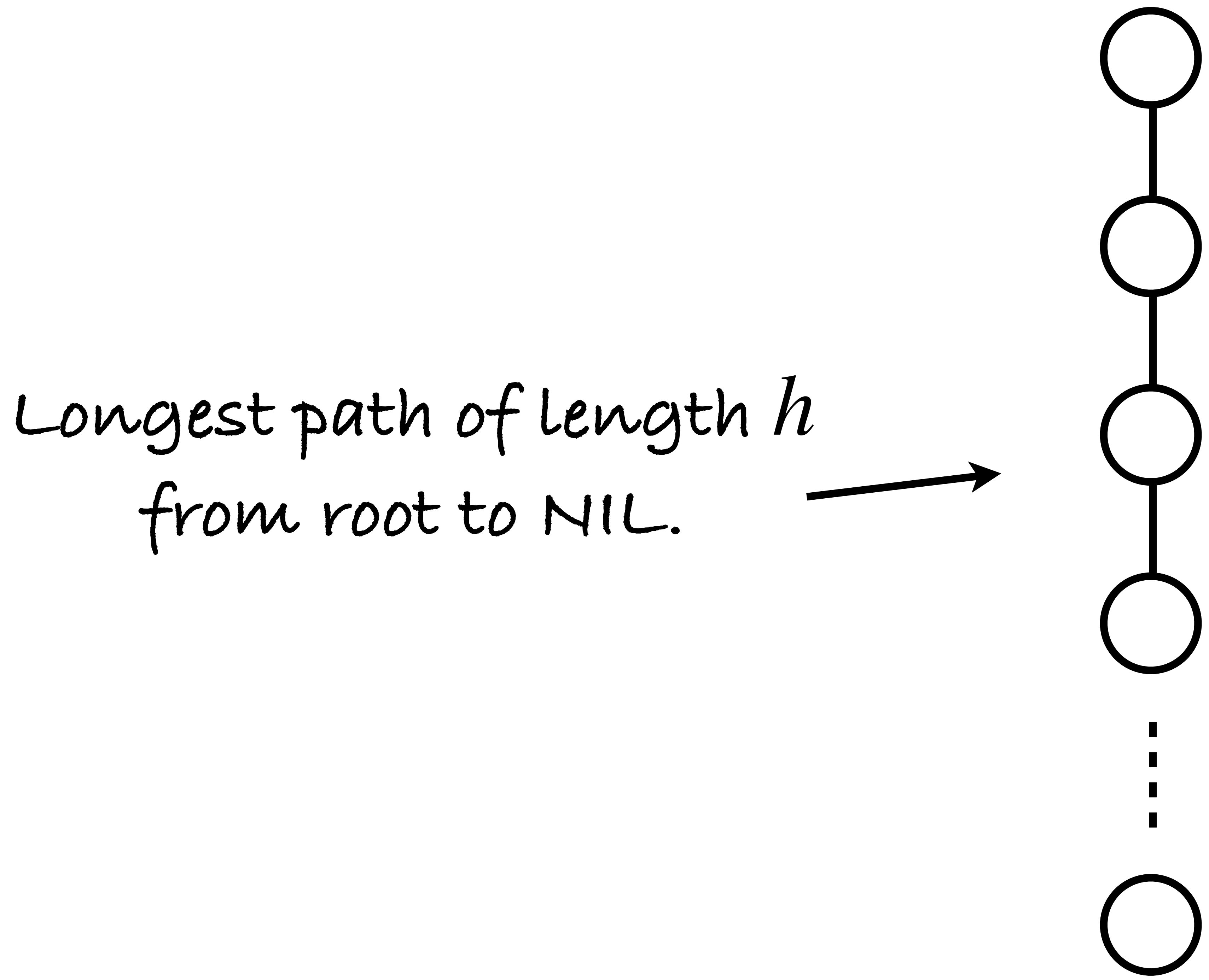
Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$



RB-Trees: Height Bound

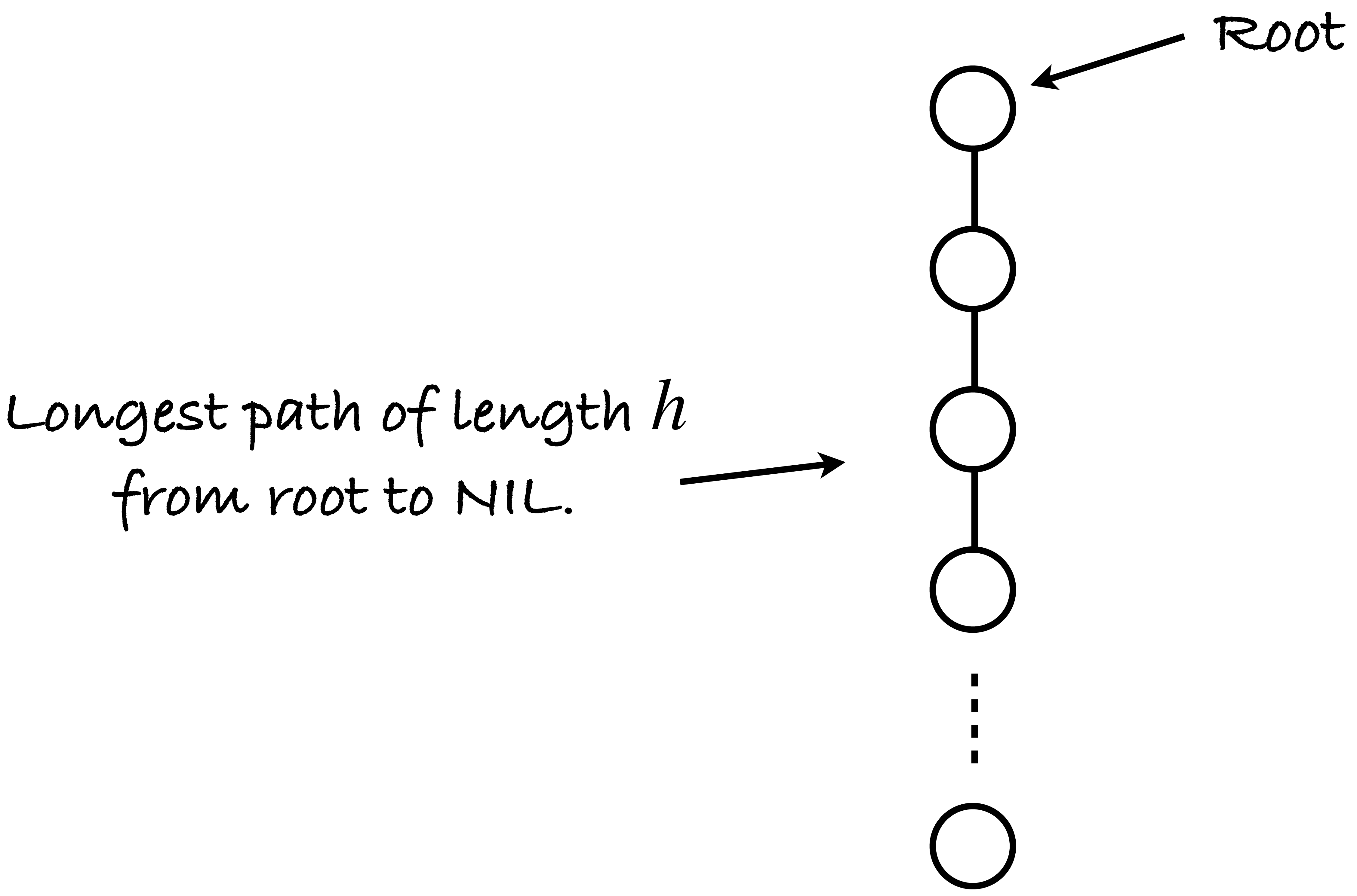
Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$



RB-Trees: Height Bound

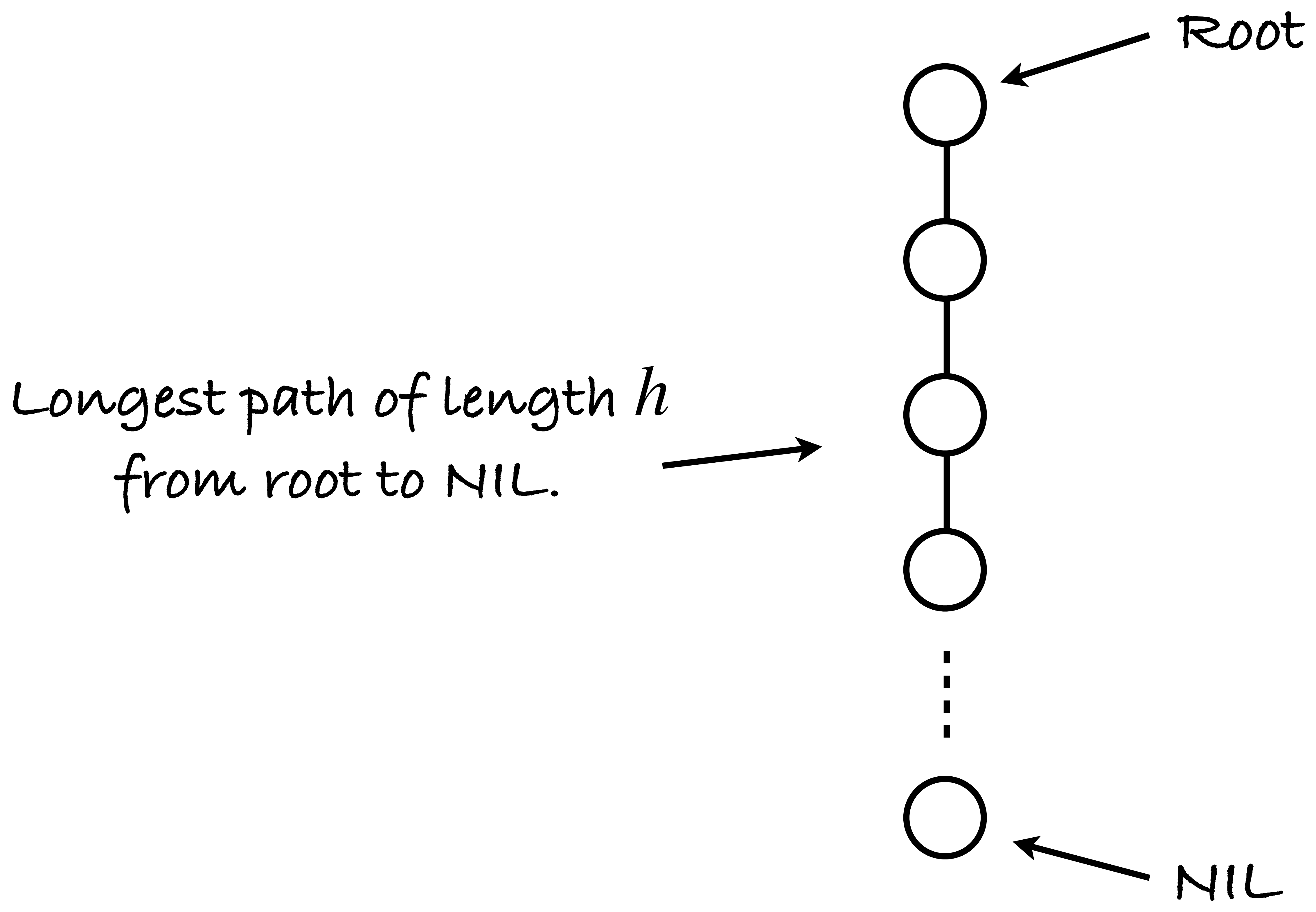
Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$



RB-Trees: Height Bound

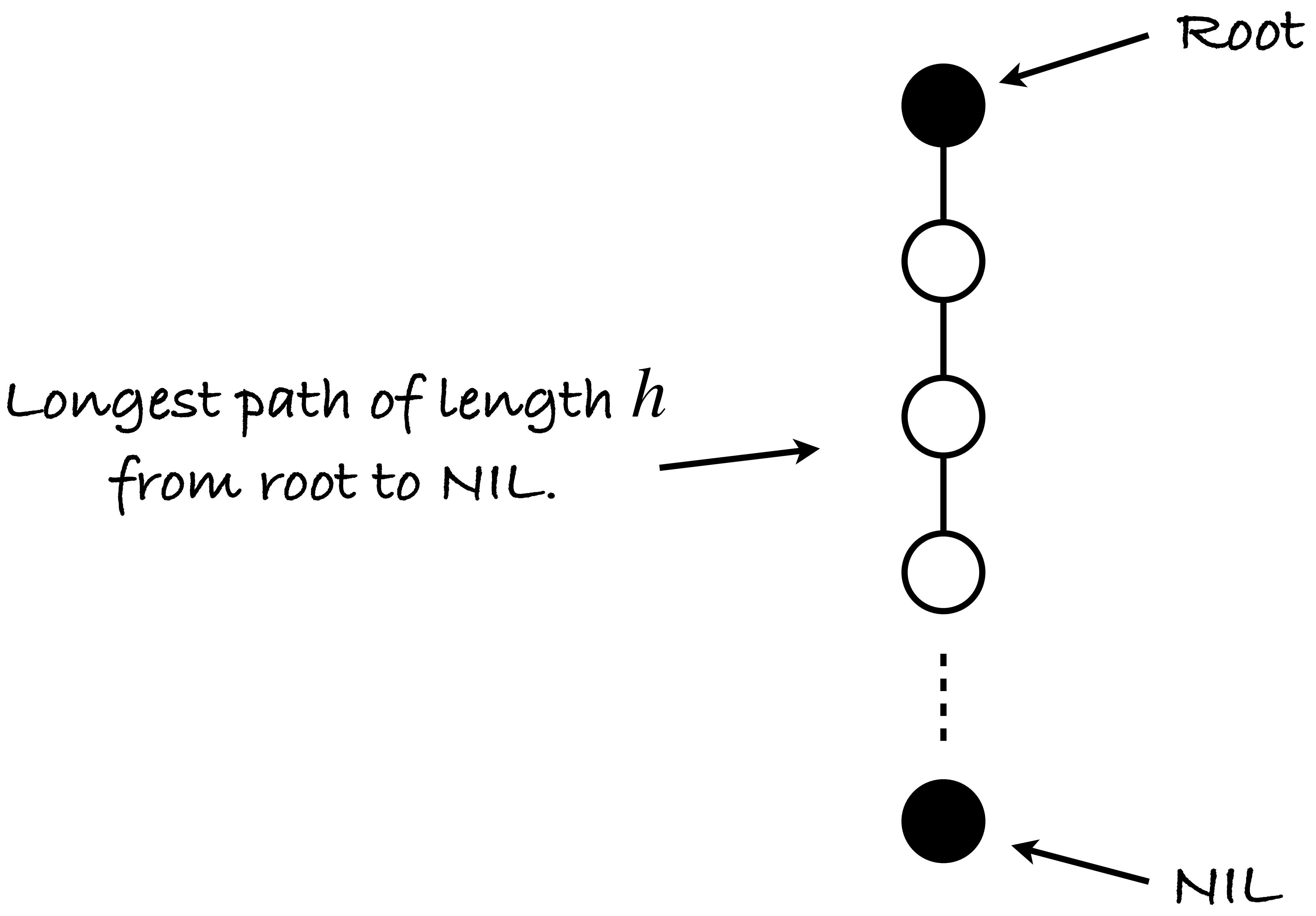
Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$



RB-Trees: Height Bound

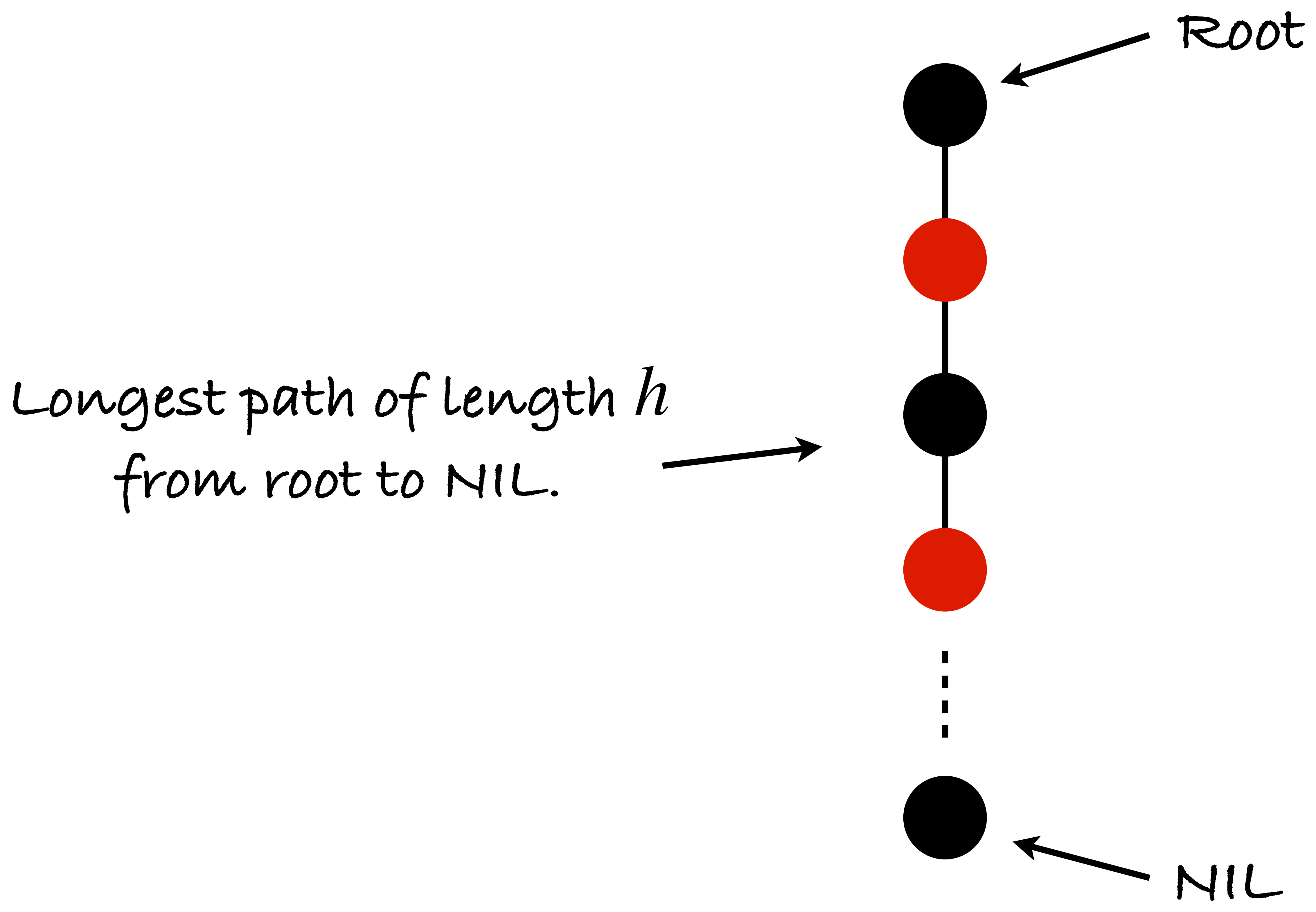
Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$



RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

$$\Longrightarrow \quad n \geq 2^{h/2} - 1$$

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

$$\implies \quad n \geq 2^{h/2} - 1 \quad (\because \text{ red nodes can at most alternate on a path from root to NIL})$$

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

$$\implies n \geq 2^{h/2} - 1 \quad (\because \text{red nodes can at most alternate on a path from root to NIL})$$

$$\implies n + 1 \geq 2^{h/2}$$

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

$$\implies \quad n \geq 2^{h/2} - 1 \quad (\because \text{ red nodes can at most alternate on a path from root to NIL})$$

$$\implies \quad n + 1 \geq 2^{h/2}$$

$$\implies \quad \lg(n+1) \geq h/2$$

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

$$\implies \quad n \geq 2^{h/2} - 1 \quad (\because \text{ red nodes can at most alternate on a path from root to NIL})$$

$$\implies \quad n + 1 \geq 2^{h/2}$$

$$\implies \quad \lg(n+1) \geq h/2$$

$$\implies \quad h \leq 2\lg(n+1)$$

RB-Trees: Height Bound

Lemma: A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

Proof: Let h be the height of root and the tree.

We know that subtree at root contains least $2^{bh(root)} - 1$ internal nodes.

Therefore,

$$n \geq 2^{bh(root)} - 1$$

$$\implies n \geq 2^{h/2} - 1 \quad (\because \text{ red nodes can at most alternate on a path from root to NIL})$$

$$\implies n + 1 \geq 2^{h/2}$$

$$\implies \lg(n+1) \geq h/2$$

$$\implies h \leq 2\lg(n+1)$$

$\blacksquare$

RB-Trees: Operations

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

- Insert an element.

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

- Insert an element.
- Delete an element.

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

- Insert an element.
- Delete an element.
- Search for an element with the key k.

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

- Insert an element.
- Delete an element.
- Search for an element with the key k.
- Minimum or Maximum of the set.

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

- Insert an element.
- Delete an element.
- Search for an element with the key k.
- Minimum or Maximum of the set.
- Successor or Predecessor of an element of the set.

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on **BSTs**:

- Insert an element.
- Delete an element.
- Search for an element with the key k.
- Minimum or Maximum of the set.
- Successor or Predecessor of an element of the set.

Same as BSTs

RB-Trees: Operations

We can perform the same operations on RB-Trees as we did on BSTs:

- Insert an element.
- Delete an element.
- Search for an element with the key k.
- Minimum or Maximum of the set.
- Successor or Predecessor of an element of the set.

Require care

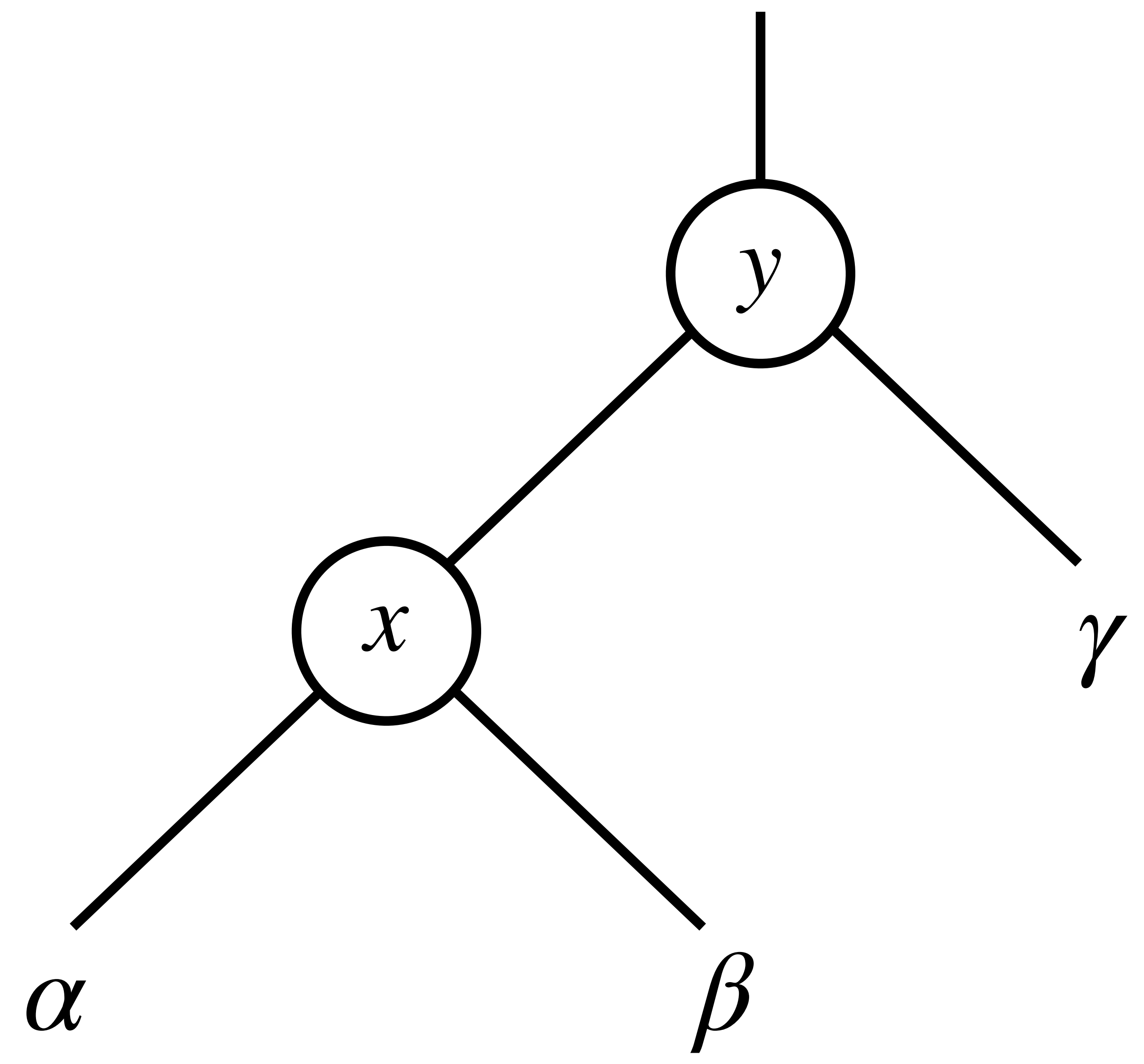
RB-Trees: Rotations

RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:

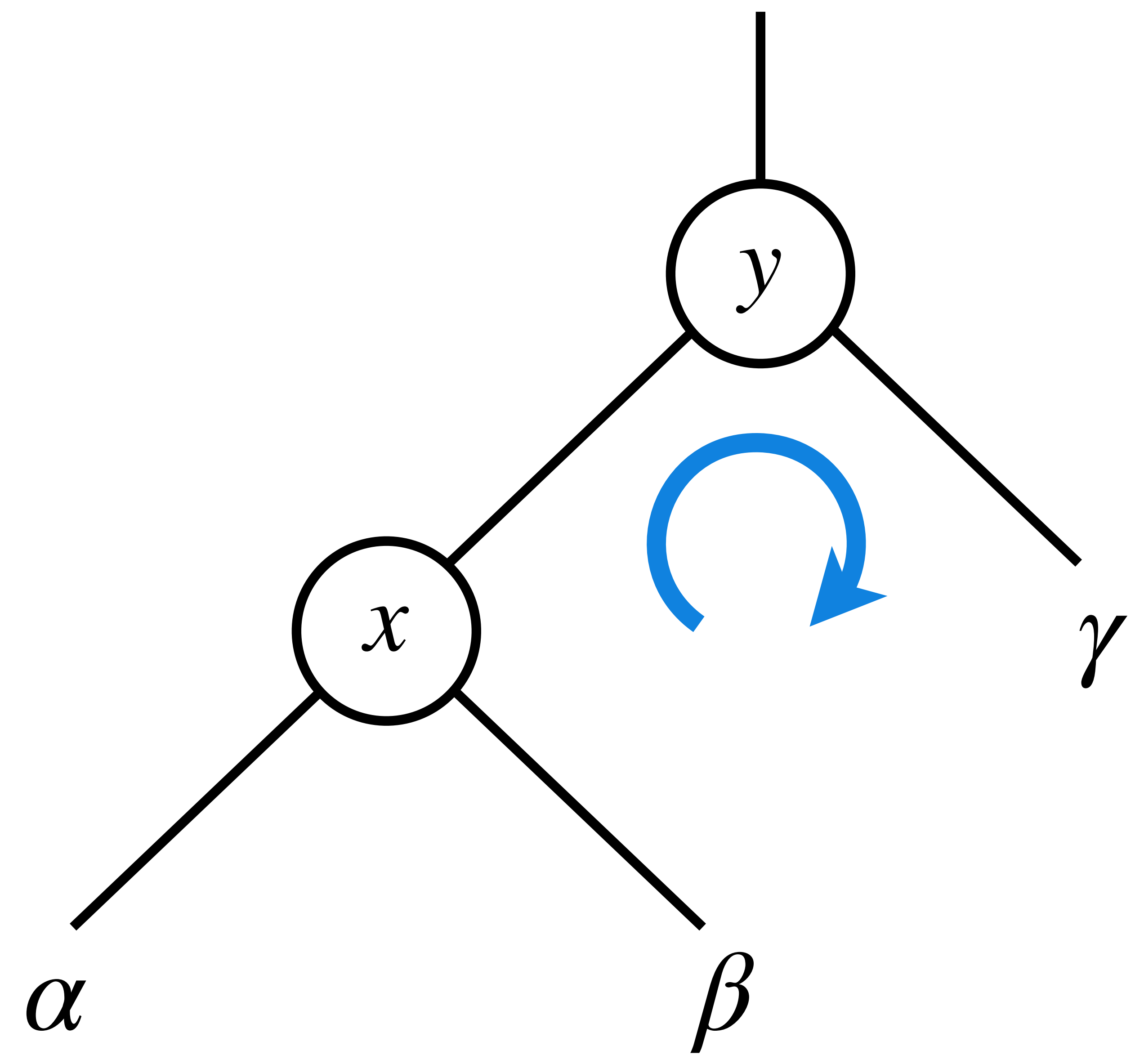
RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



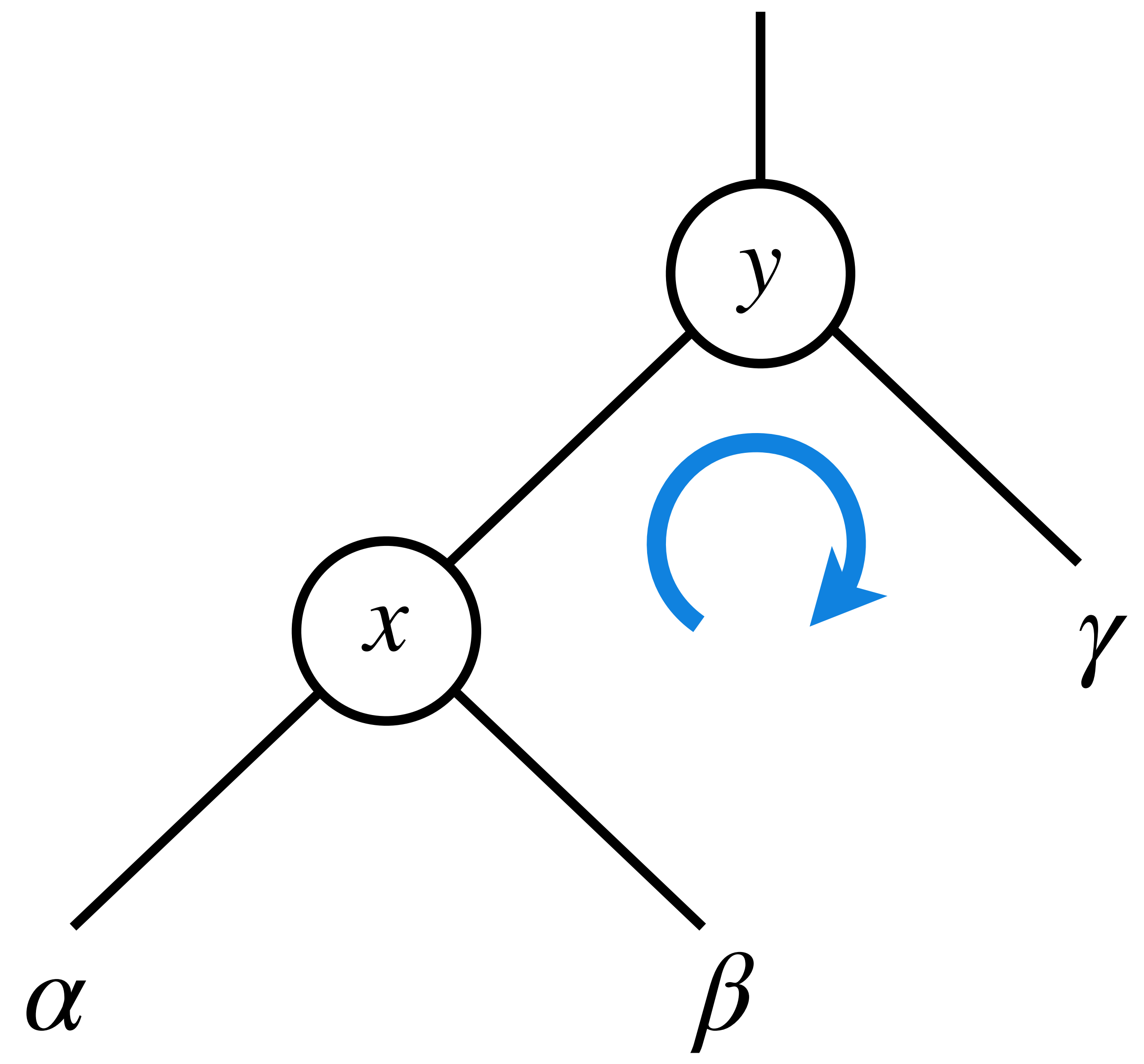
RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



RB-Trees: Rotations

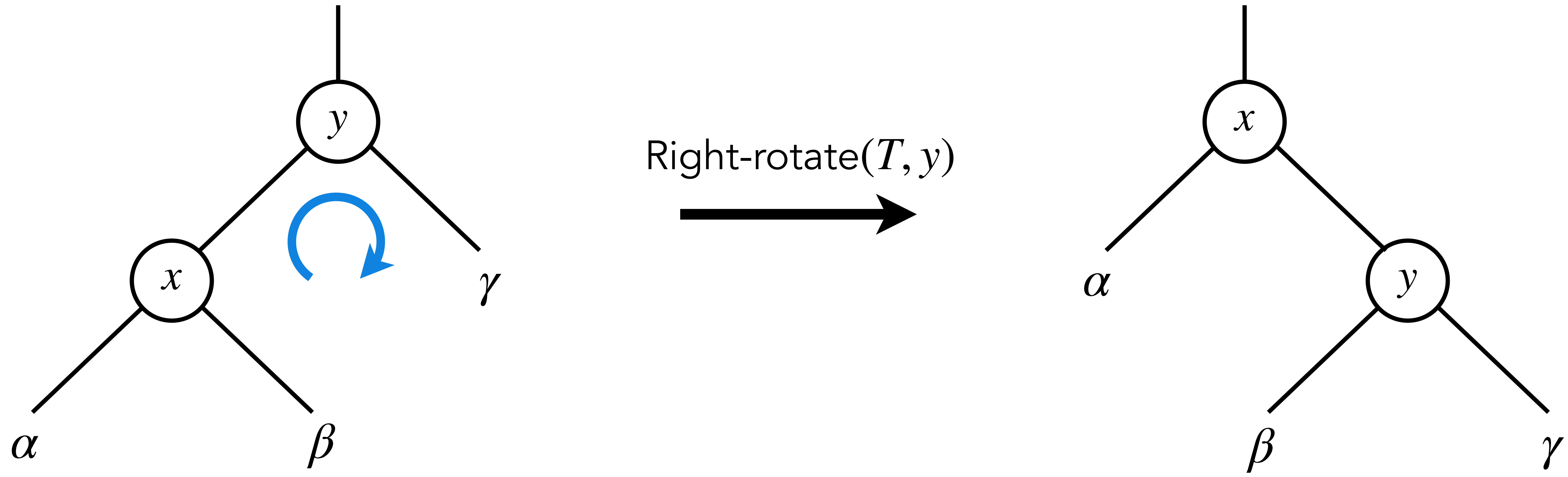
Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



Right-rotate(T, y)

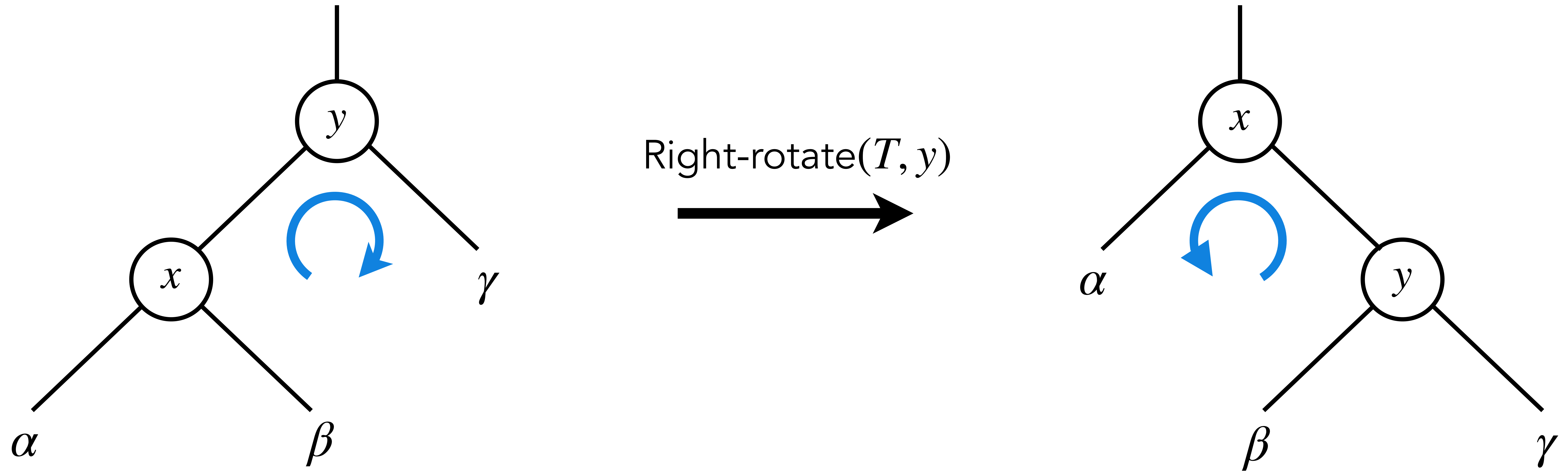
RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



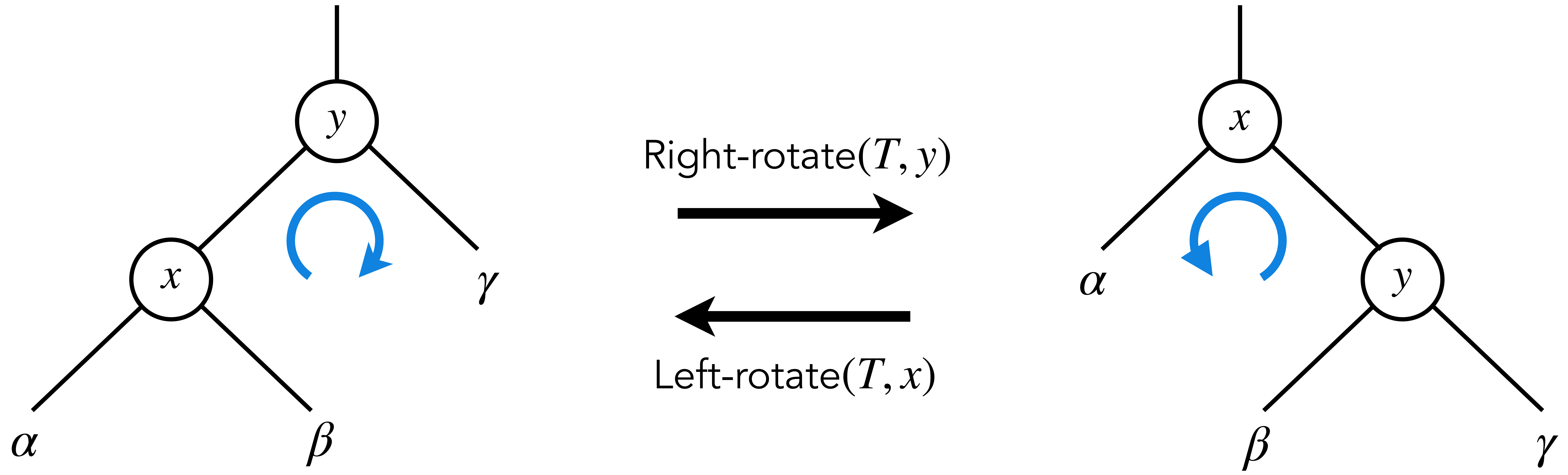
RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



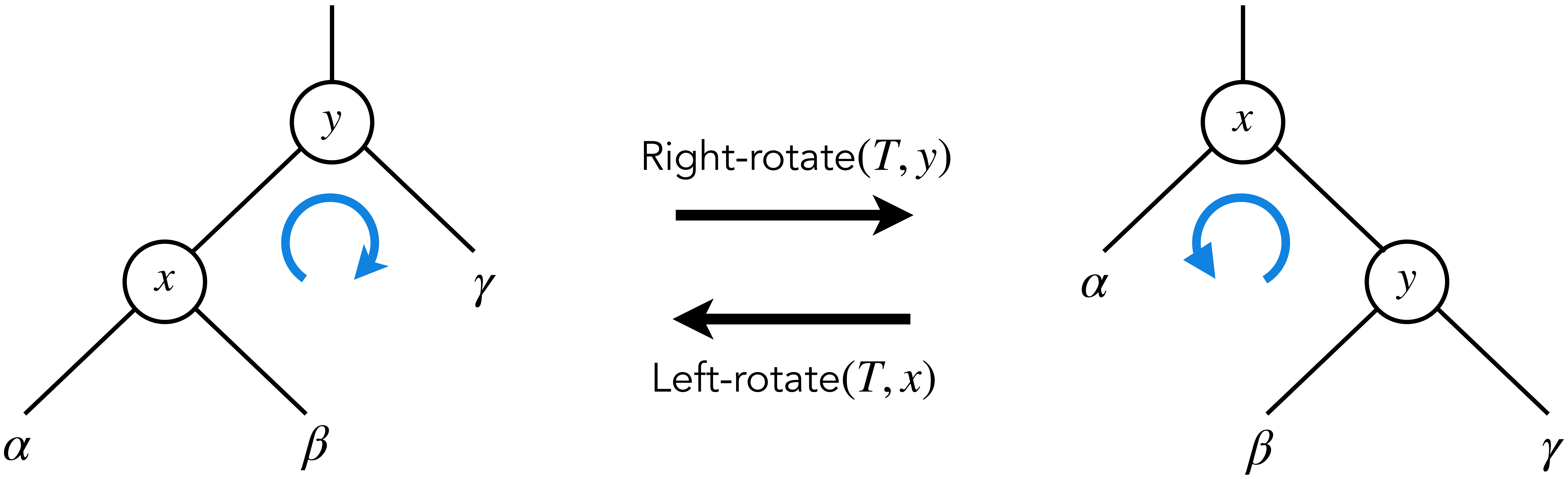
RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



RB-Trees: Rotations

Rotations are basic operations useful in Insertion and Deletion on an RB-tree:



Note: Rotations do not disturb BST property and can be performed in constant time.

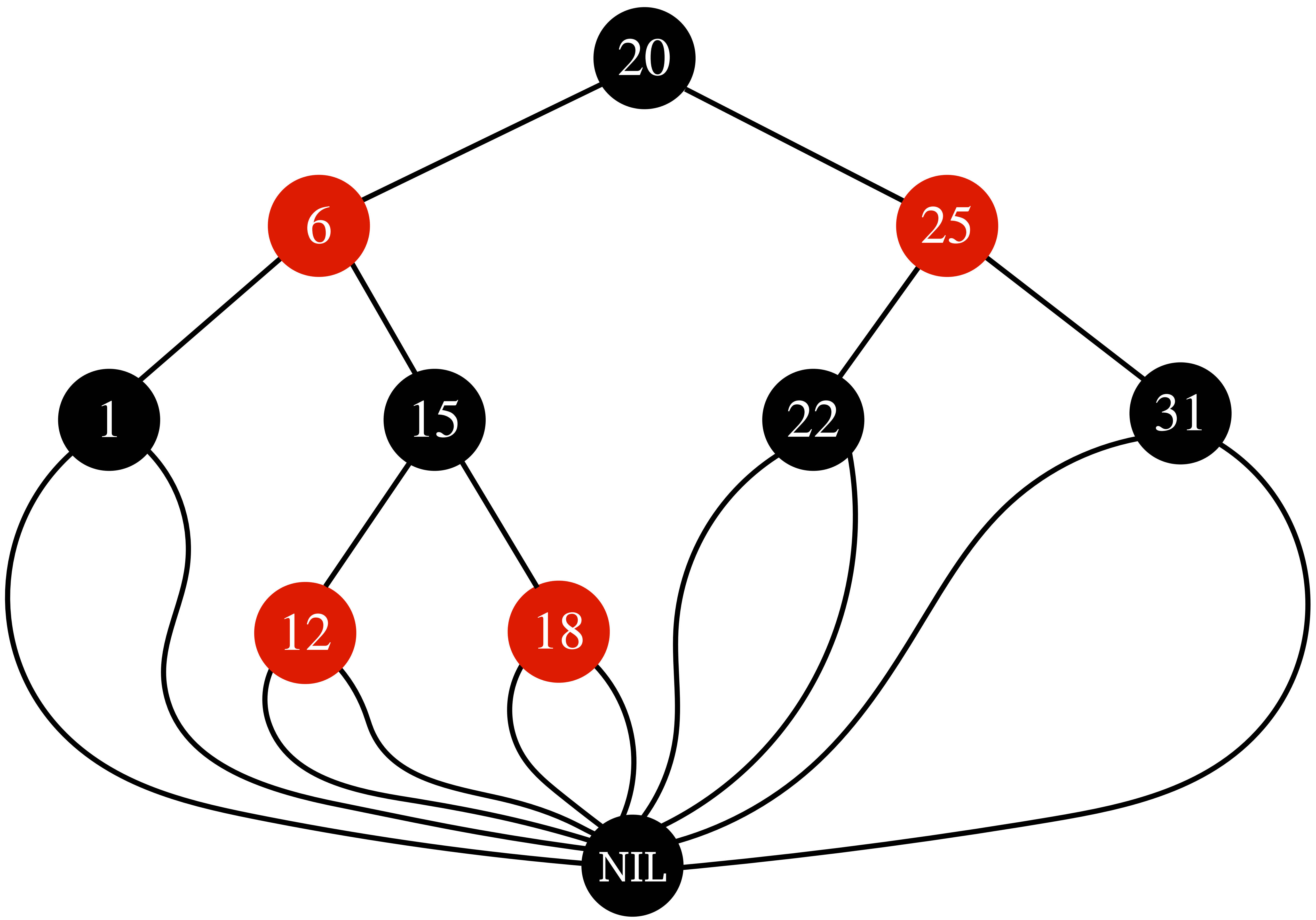
RB-Trees: Insertion

RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:

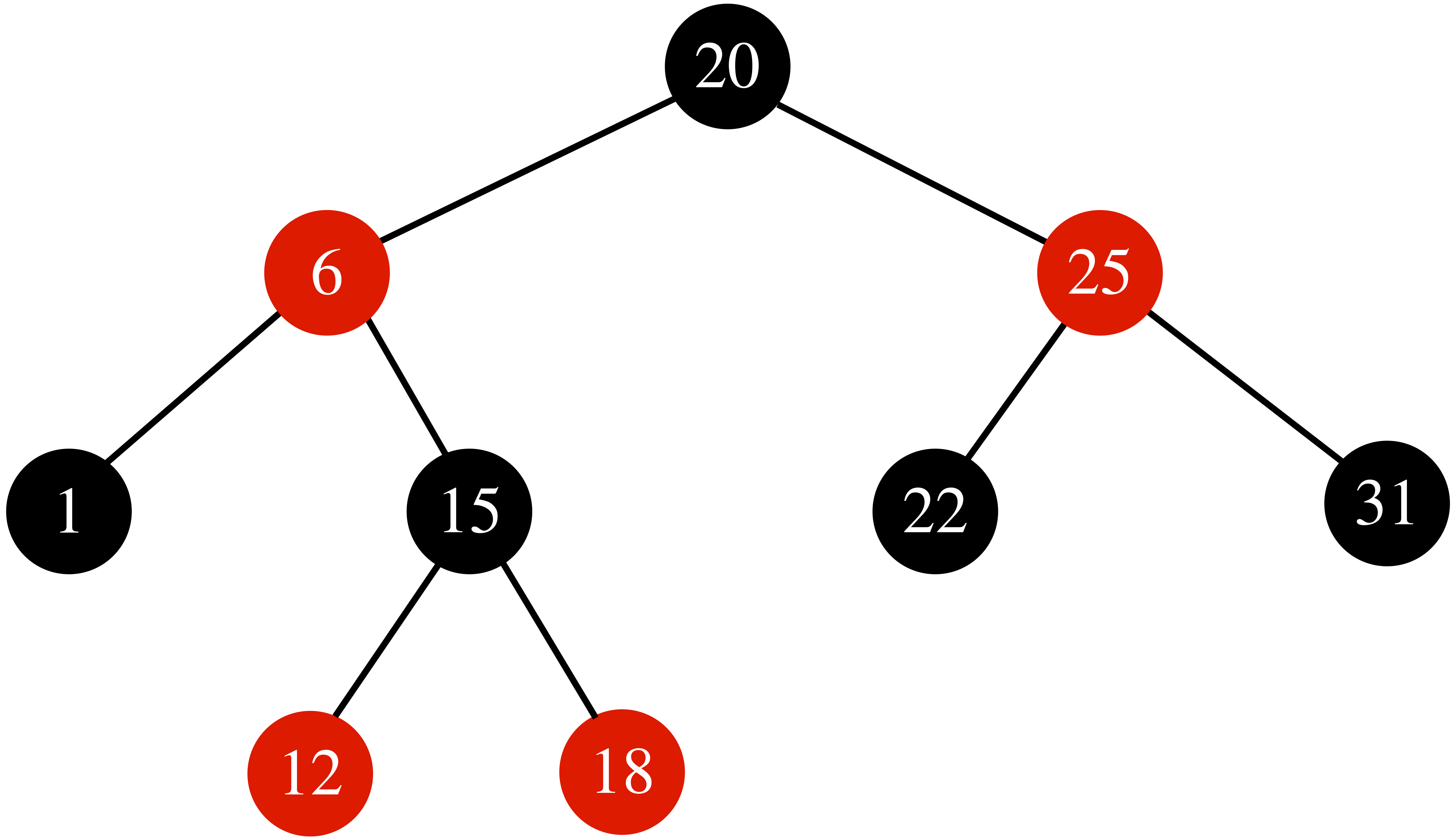
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



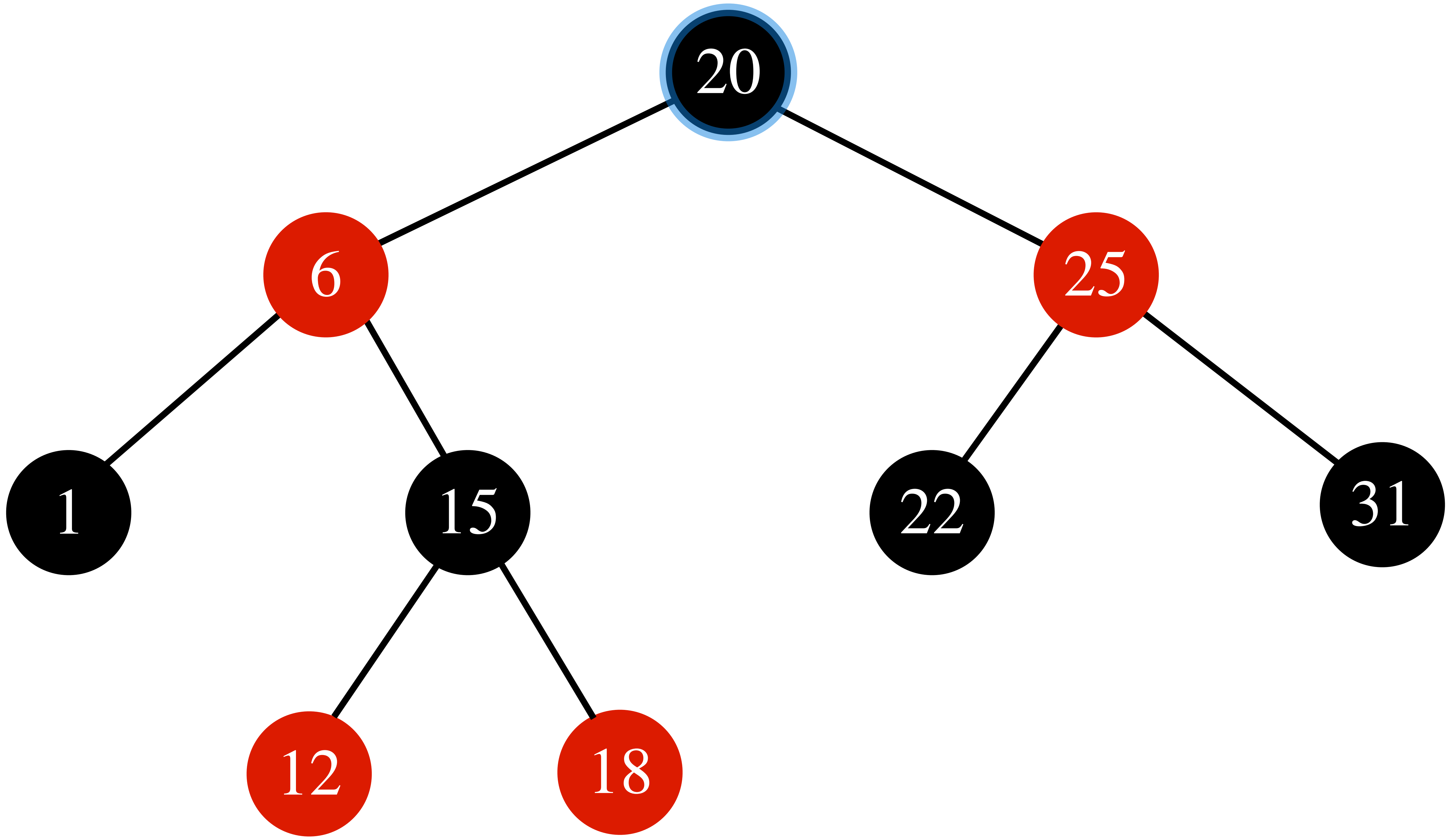
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



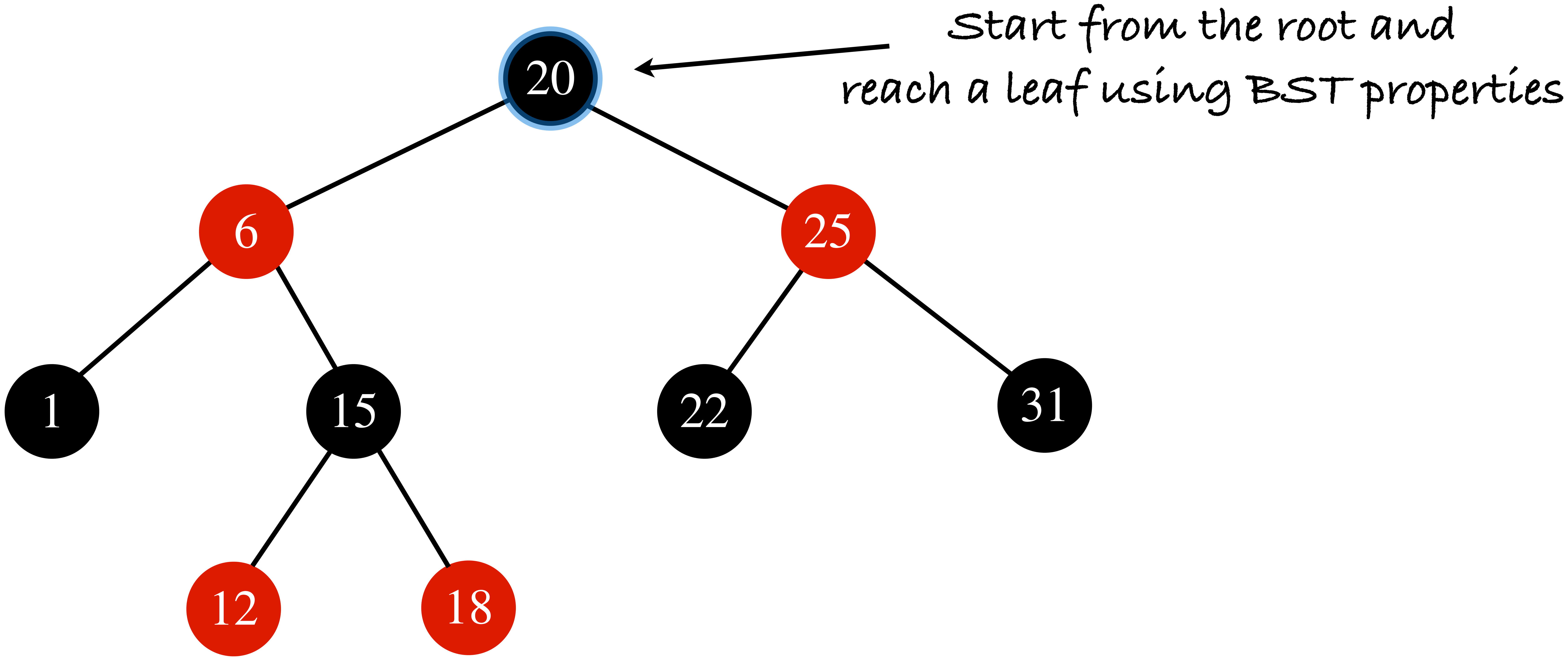
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



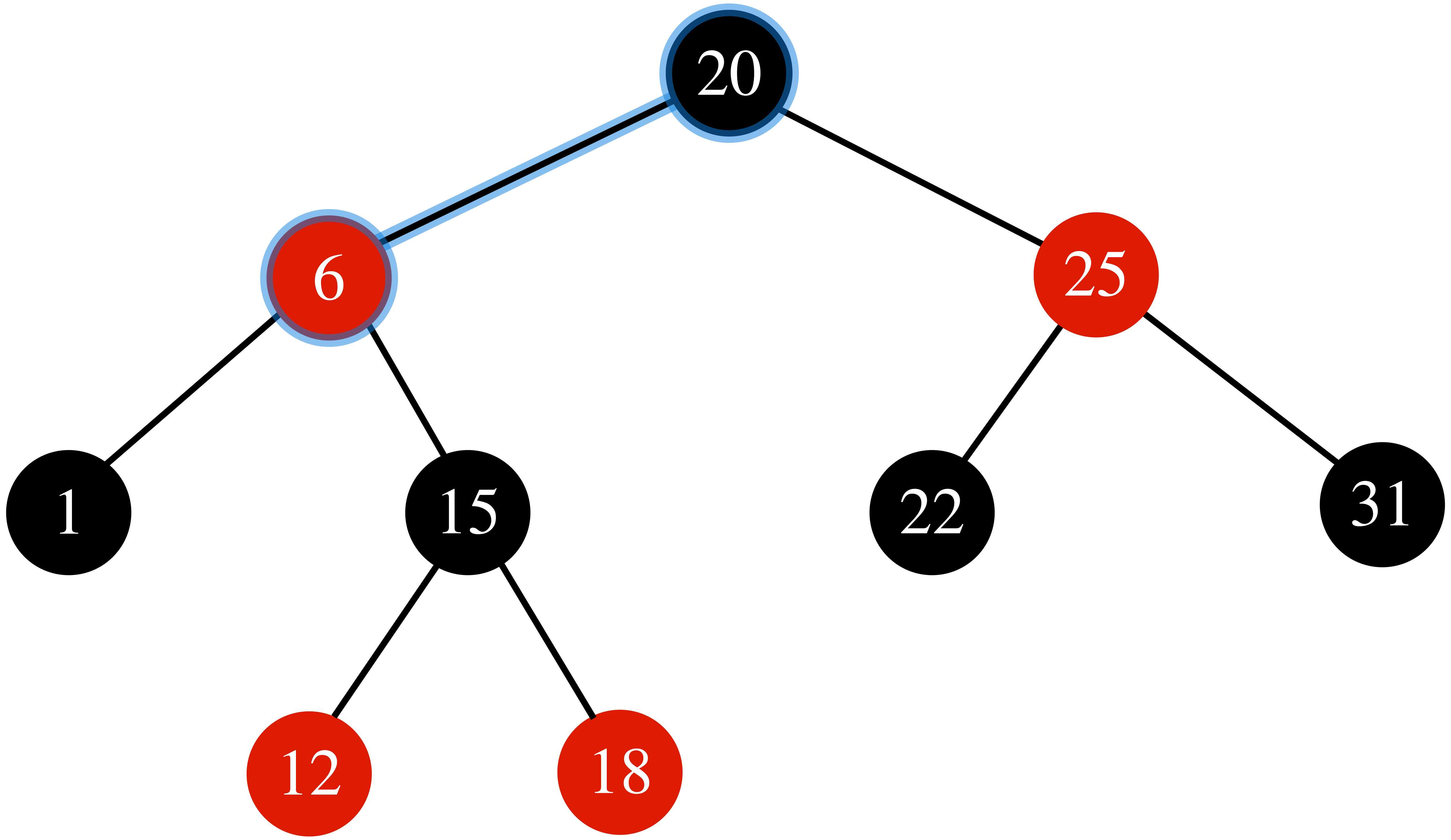
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



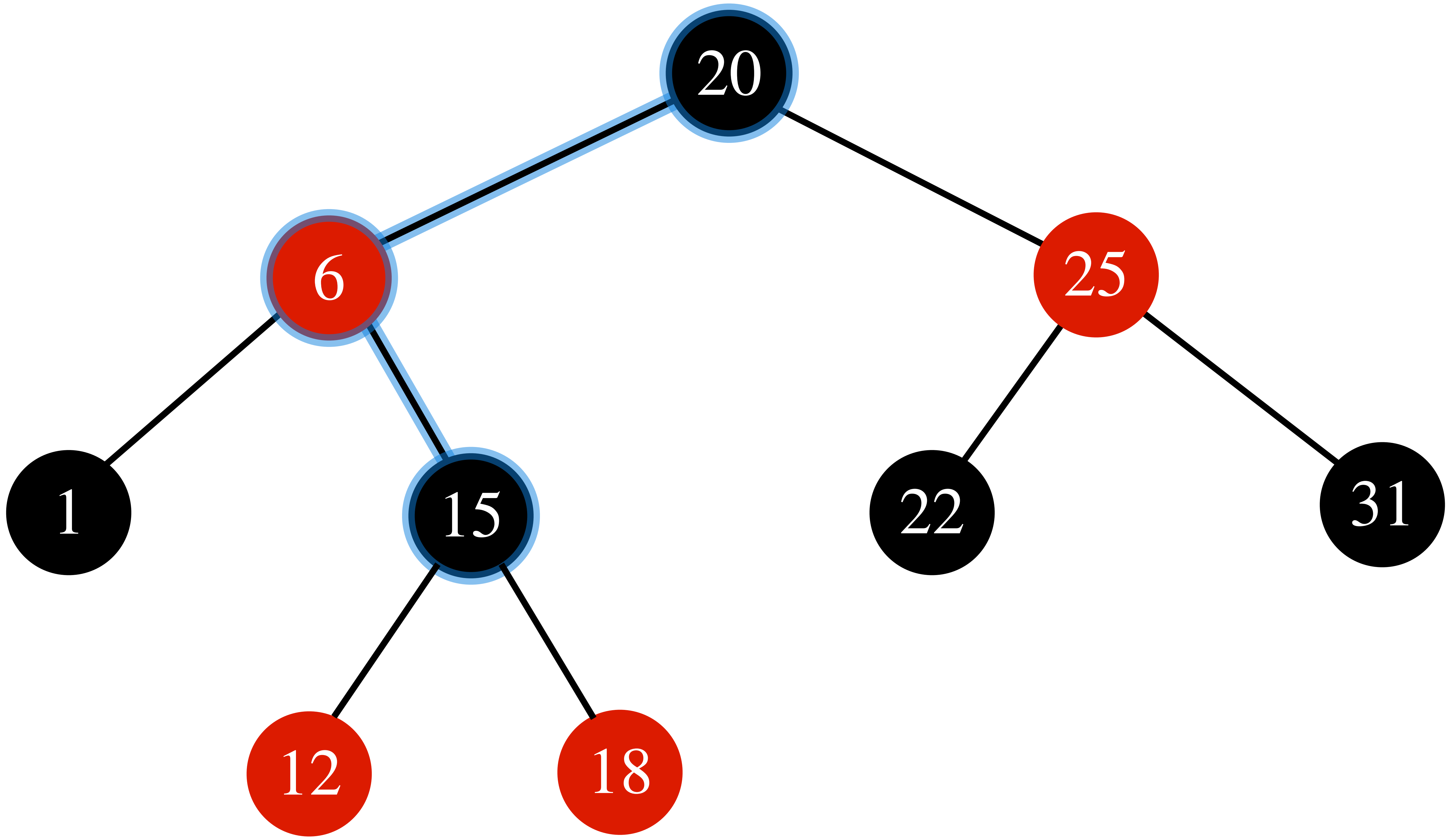
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



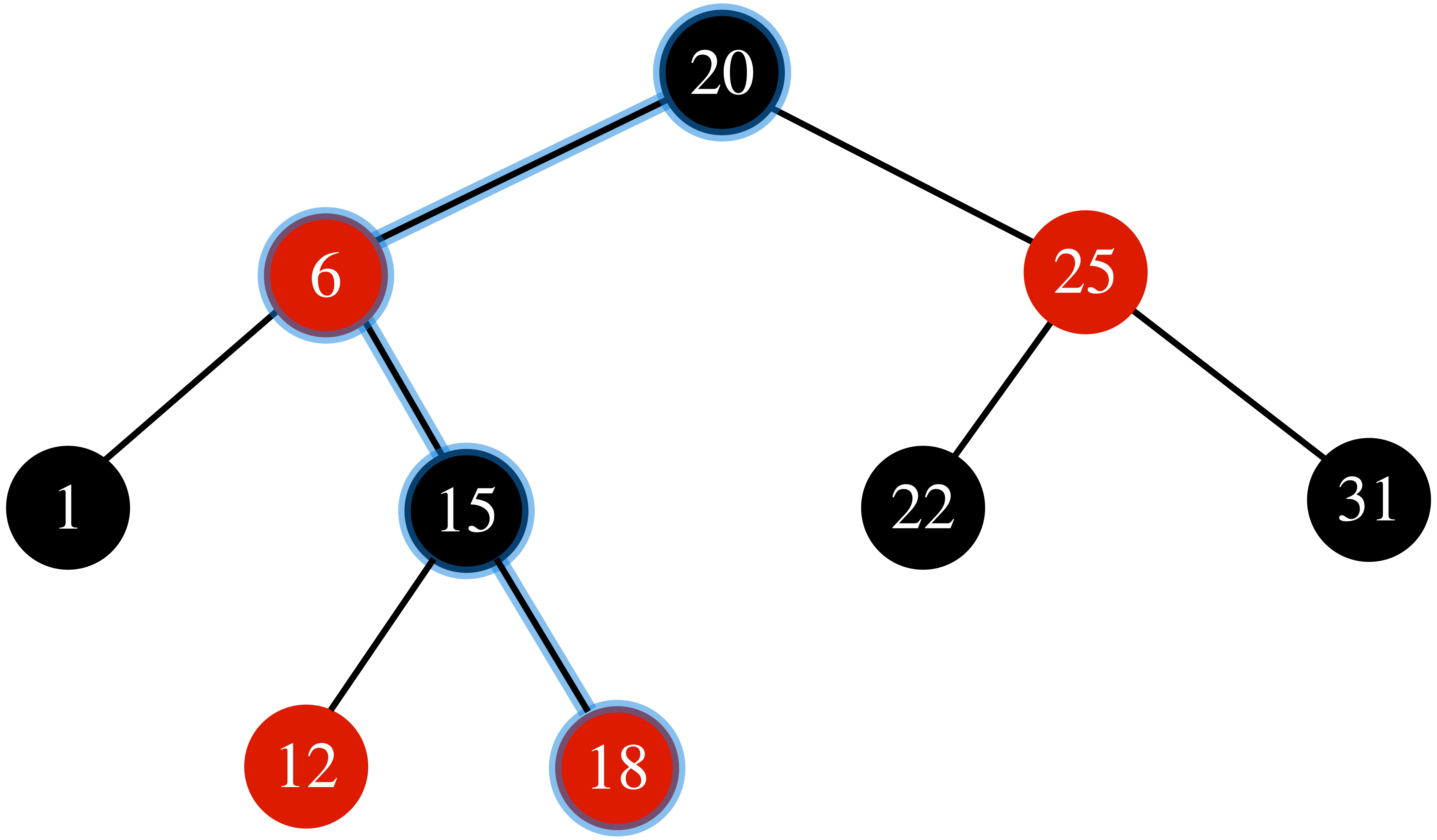
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



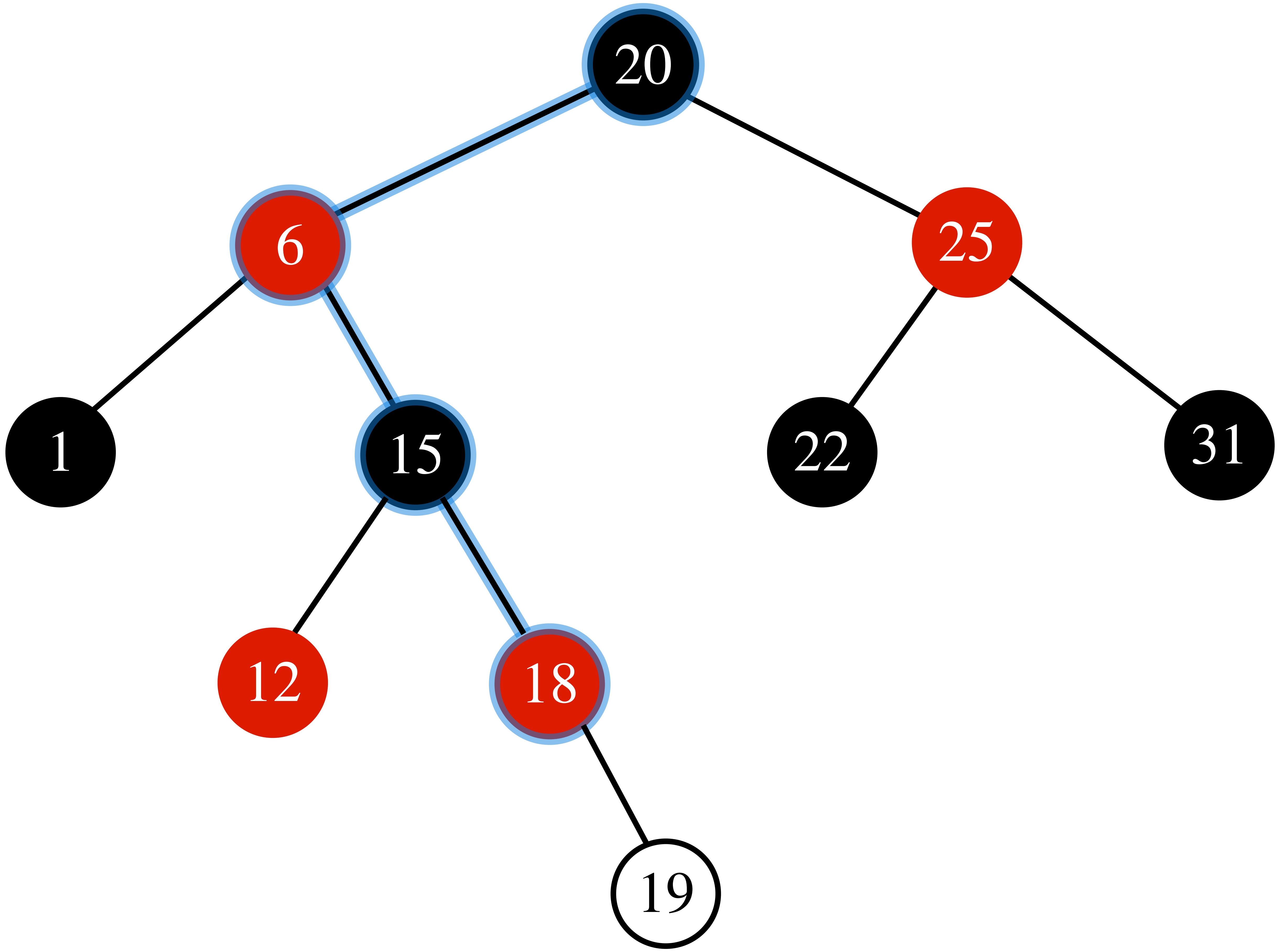
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



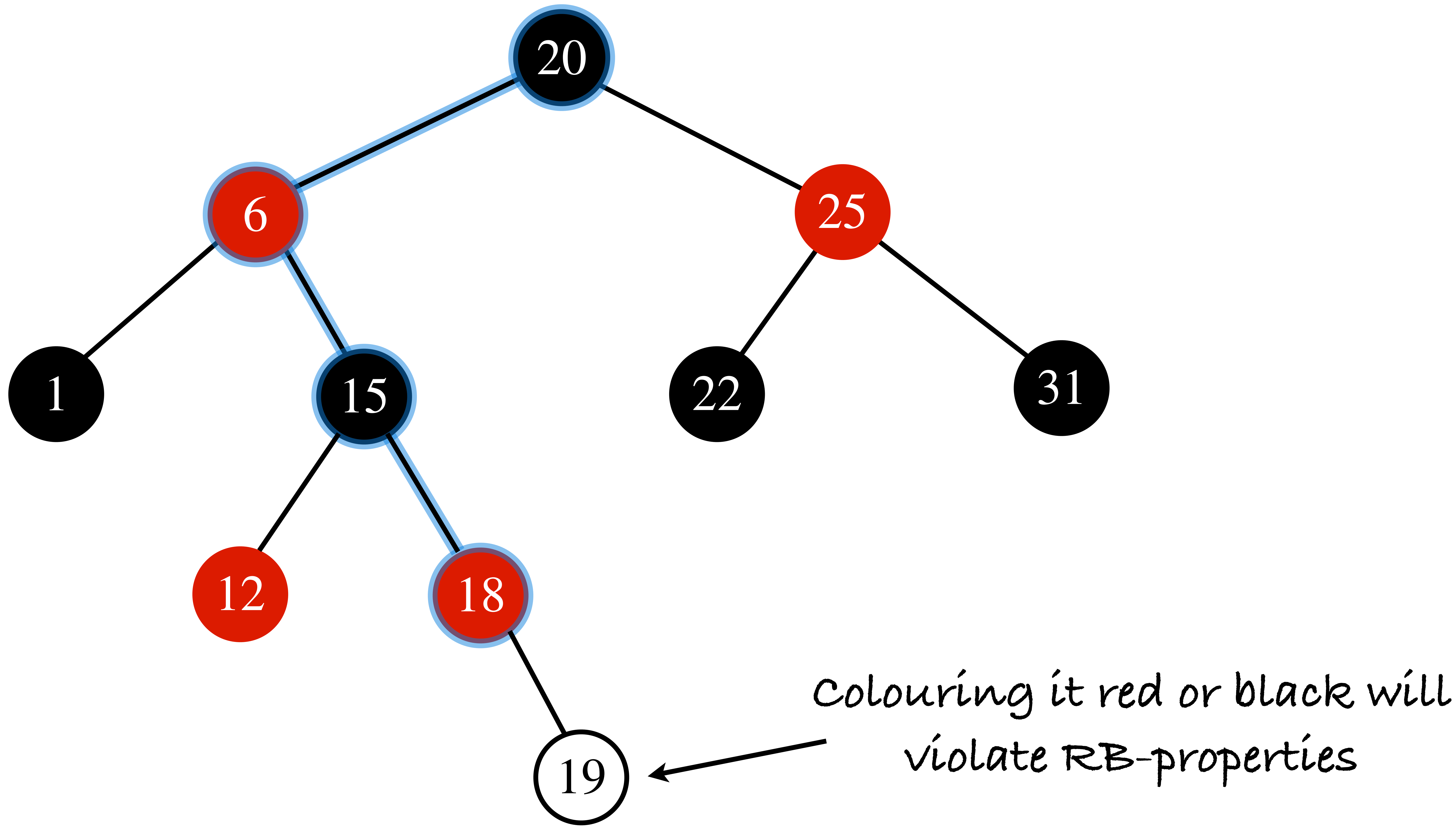
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



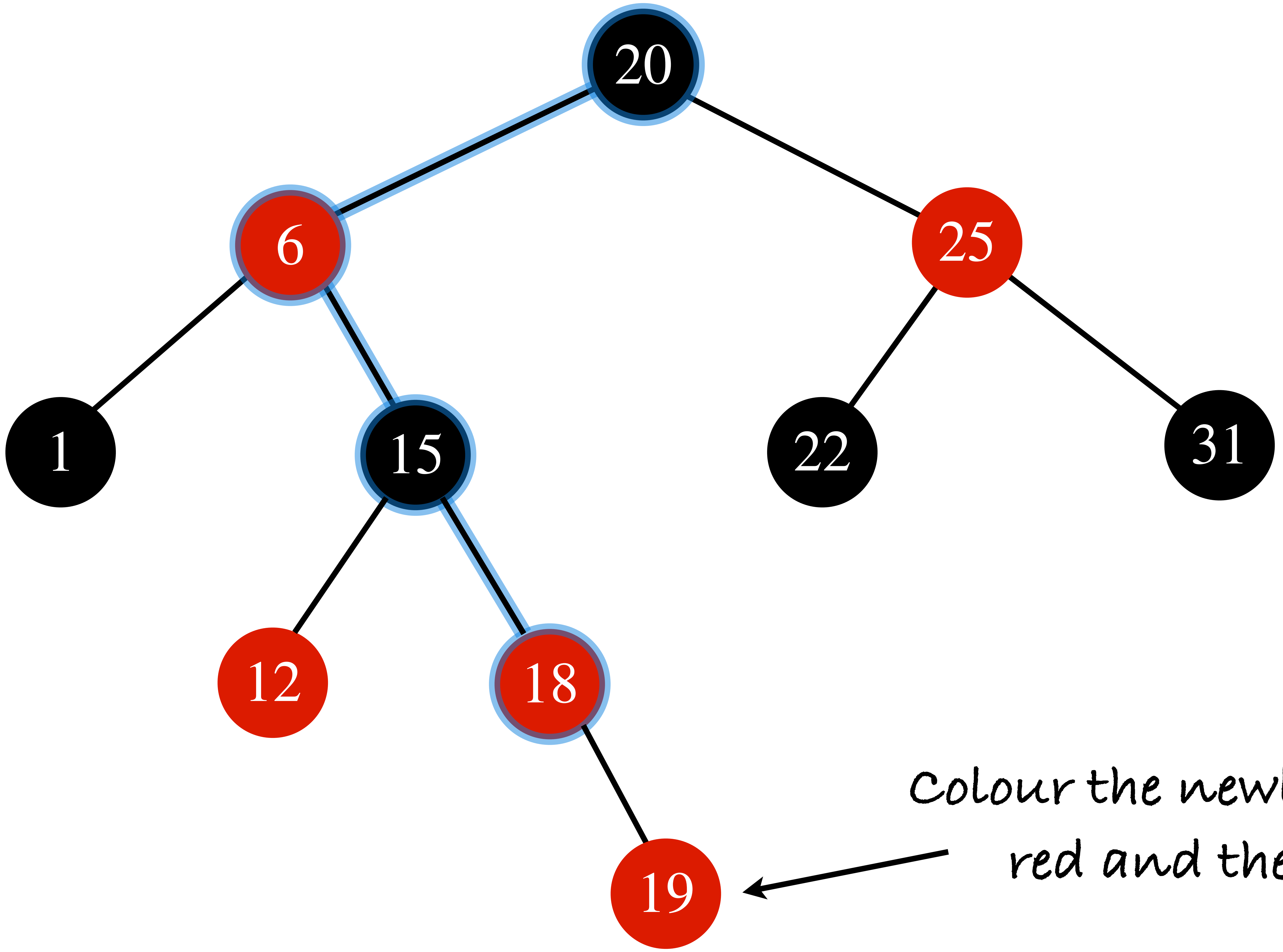
RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



RB-Trees: Insertion

Suppose we want to insert 19 in the following RB-tree:



Colour the newly inserted node red and then do fix-up.

RB-Trees: Insertion

RB-Trees: Insertion

Two stages of insertion:

RB-Trees: Insertion

Two stages of insertion:

- Insert the new node as it is done in a BST and colour the new node red.

RB-Trees: Insertion

Two stages of insertion:

- Insert the new node as it is done in a BST and colour the new node red.
- Do fix-ups as parent of the new node may also be a red node.

RB-Trees: Insertion Cases

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is red, we do fix-ups for the following cases:

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is red, we do fix-ups for the following cases:
  - **Case 1:** z's **uncle** (sibling of z's parent) is red.

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is red, we do fix-ups for the following cases:
  - **Case 1:** z's **uncle** (sibling of z's parent) is red.
  - **Case 2:** z's **uncle** is **black** and z is a **right child**.

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is **red**, we do fix-ups for the following cases:
  - **Case 1:** z's **uncle** (sibling of z's parent) is **red**.
  - **Case 2:** z's **uncle** is **black** and z is a **right child**.
  - **Case 3:** z's **uncle** is **black** and z is a **left child**.

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is red, we do fix-ups for the following cases:
  - **Case 1:** z's **uncle** (sibling of z's parent) is red.
  - **Case 2:** z's **uncle** is **black** and z is a **right child**.
  - **Case 3:** z's **uncle** is **black** and z is a **left child**.

After doing local fix-up, z will set to its parent's parent.

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is red, we do fix-ups for the following cases:
  - **Case** 1**:** z's **uncle** (sibling of z's parent) is red.
  - **Case** 2**:** z's **uncle** is **black** and z is a **right child**.
  - **Case 3:** z's **uncle** is **black** and z is a **left child**.

After doing local fix-up, z will set to its parent's parent.

Gets converted to Case 3

RB-Trees: Insertion Cases

Let z be the newly inserted node with colour red.

- If parent of z is **black**, nothing needs to be done.
- If parent of z is red, we do fix-ups for the following cases:
  - **Case 1:** z's **uncle** (sibling of z's parent) is red.
  - **Case 2:** z's **uncle** is **black** and z is a **right child**.
  - **Case 3:** z's **uncle** is **black** and z is a **left child**.

After doing local fix-up, z will set to its parent's parent.

Gets converted to Case 3

Fix-up will be enough to terminate the process

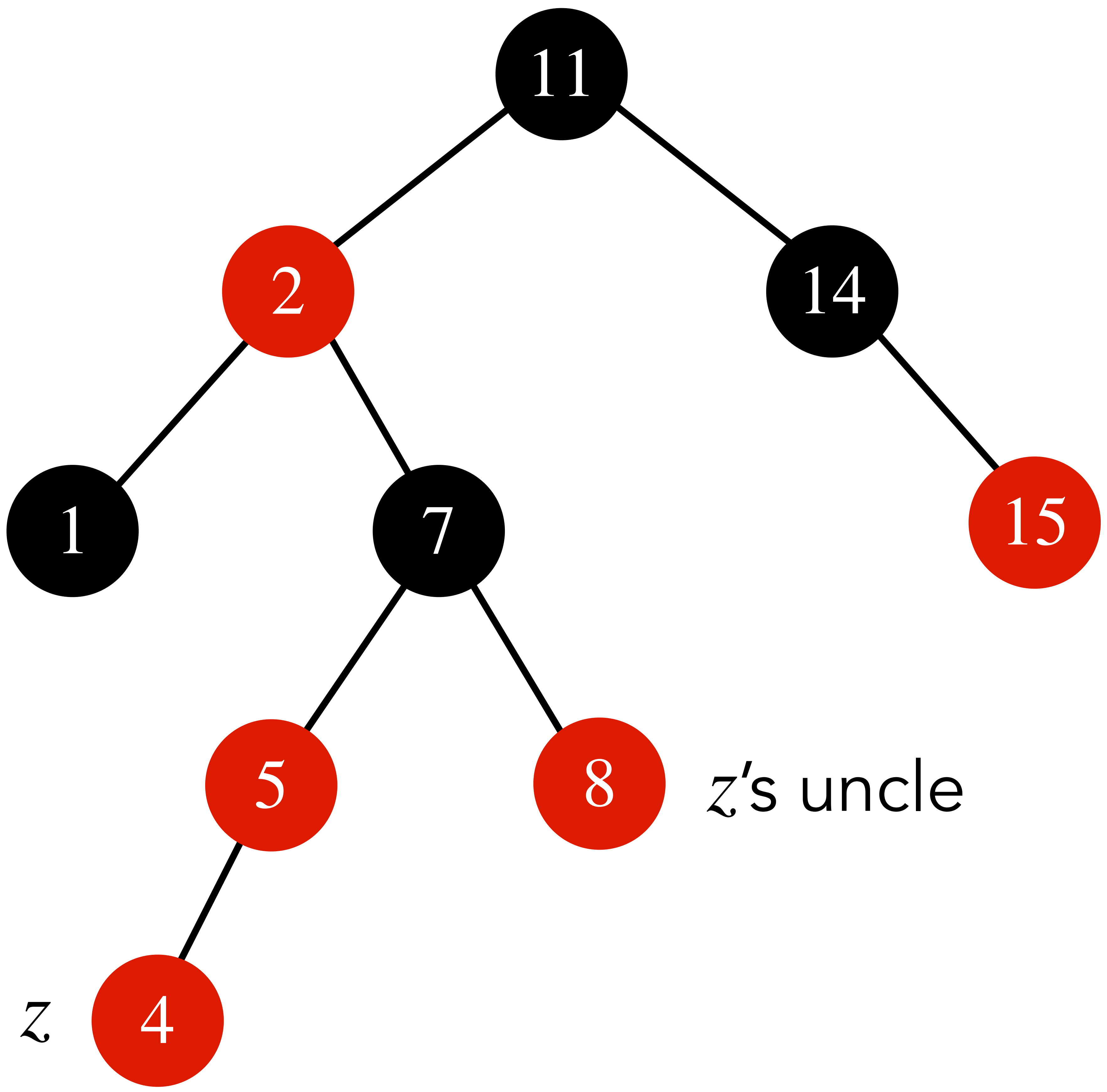
RB-Trees: Insertion Case 1

RB-Trees: Insertion Case 1

Case 1: z's **uncle** (sibling of z's parent) is red.

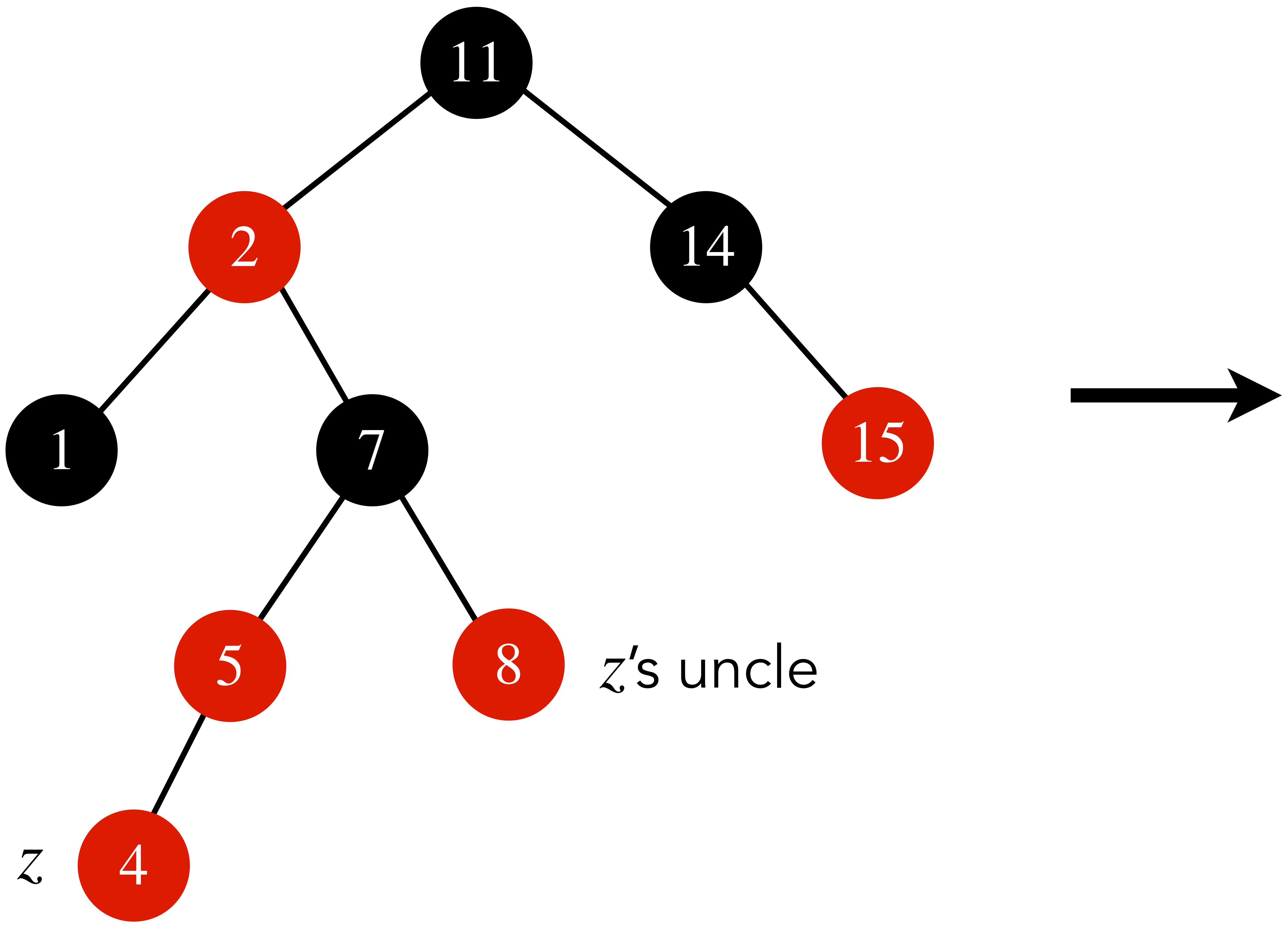
RB-Trees: Insertion Case 1

Case 1**:** z's **uncle** (sibling of z's parent) is red.



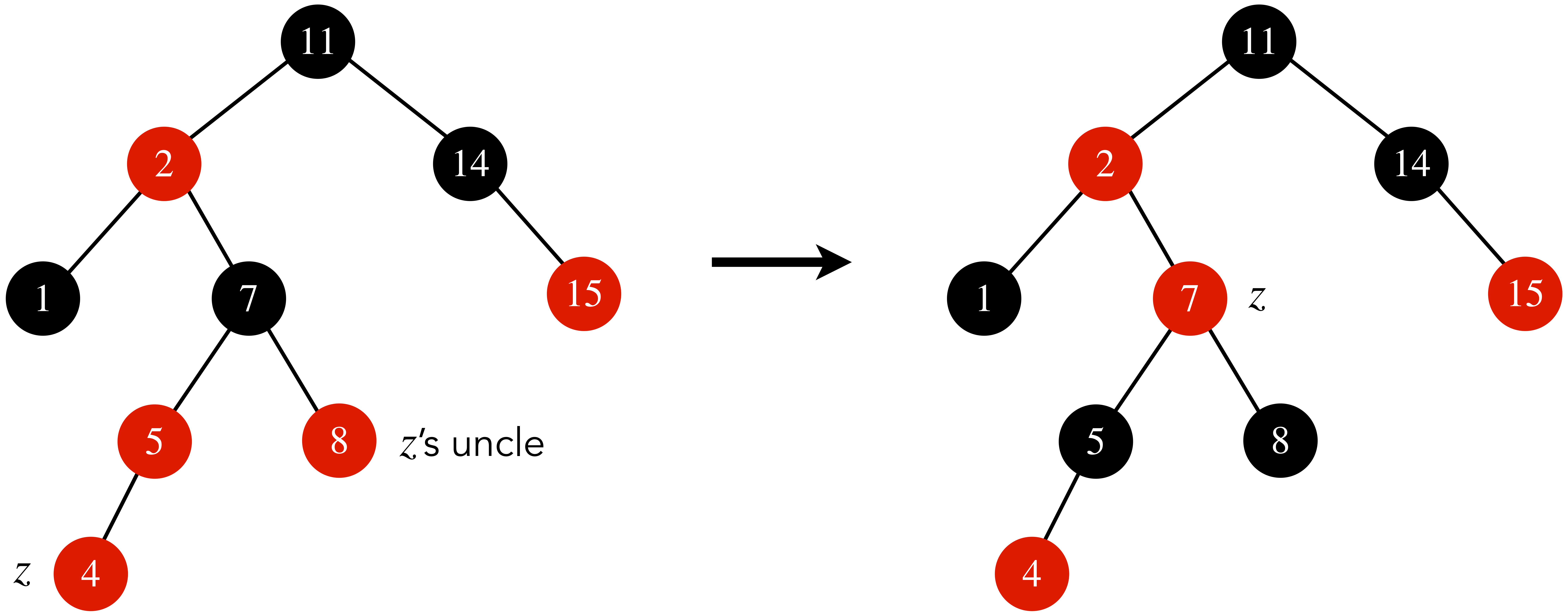
RB-Trees: Insertion Case 1

Case 1: z's **uncle** (sibling of z's parent) is red.



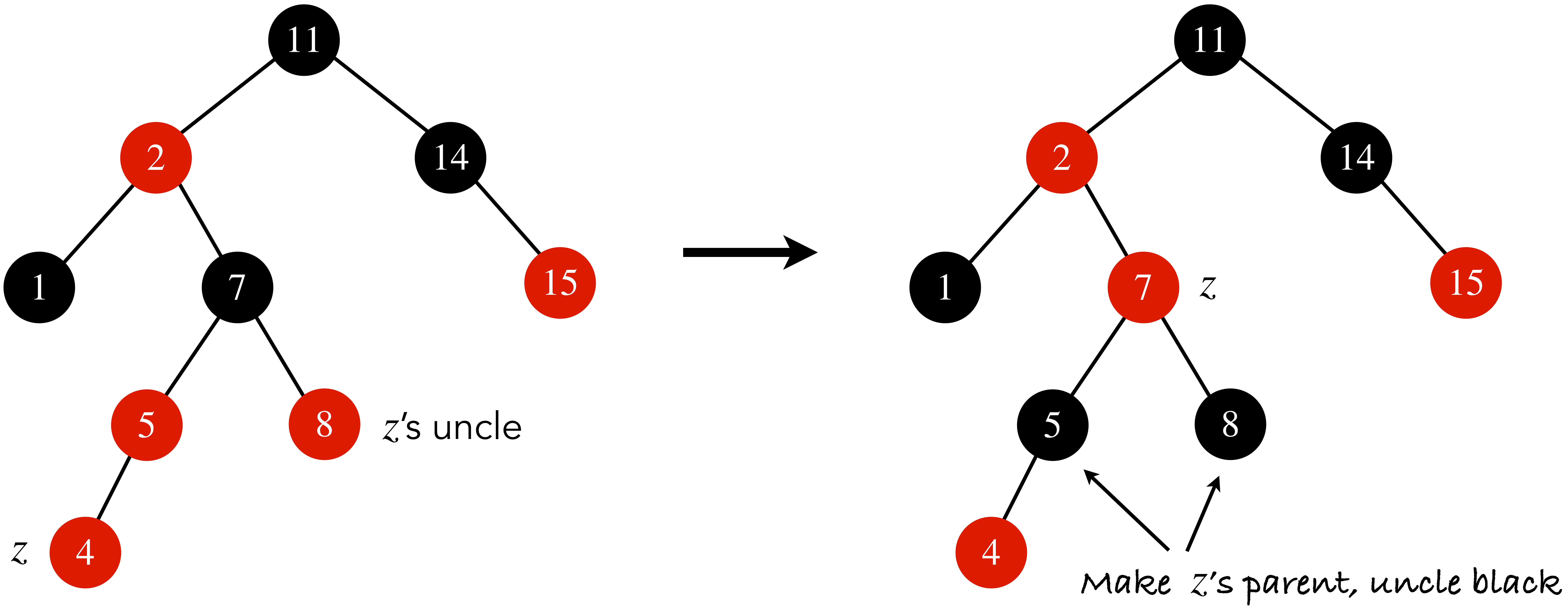
RB-Trees: Insertion Case 1

Case 1**:** z's **uncle** (sibling of z's parent) is red.



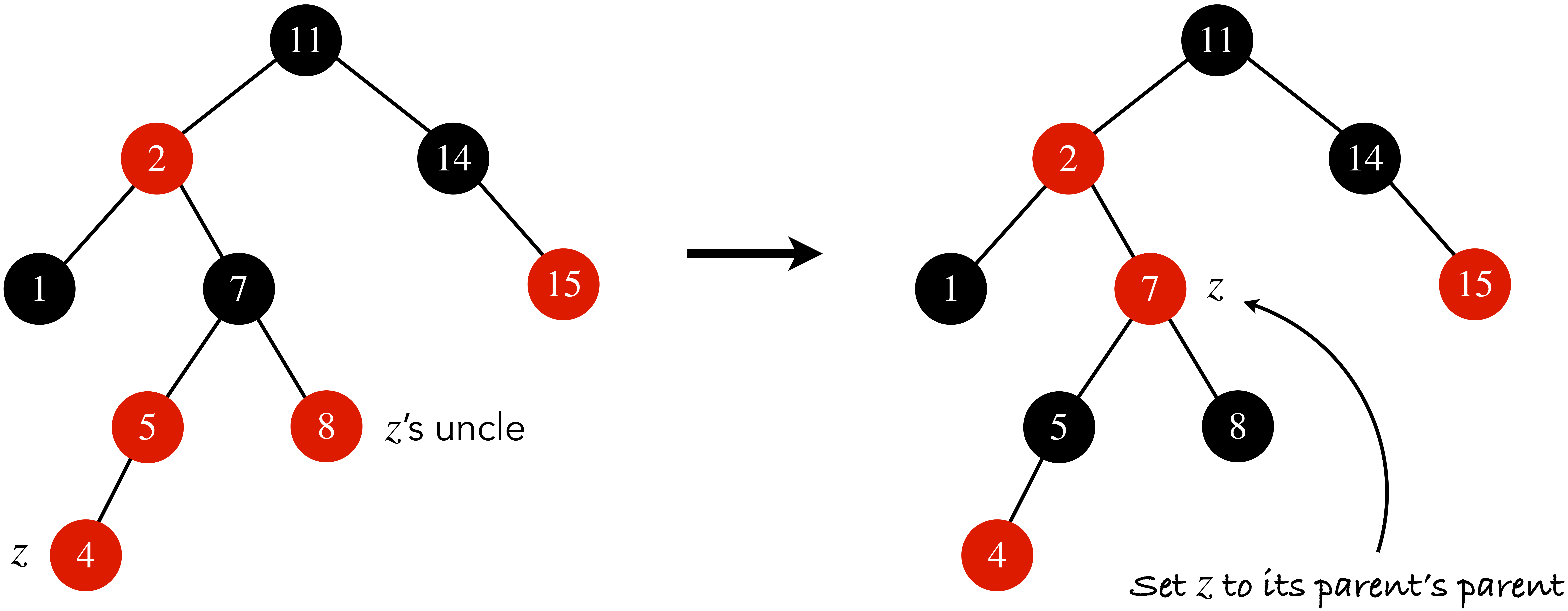
RB-Trees: Insertion Case 1

Case 1**:** z's **uncle** (sibling of z's parent) is red.



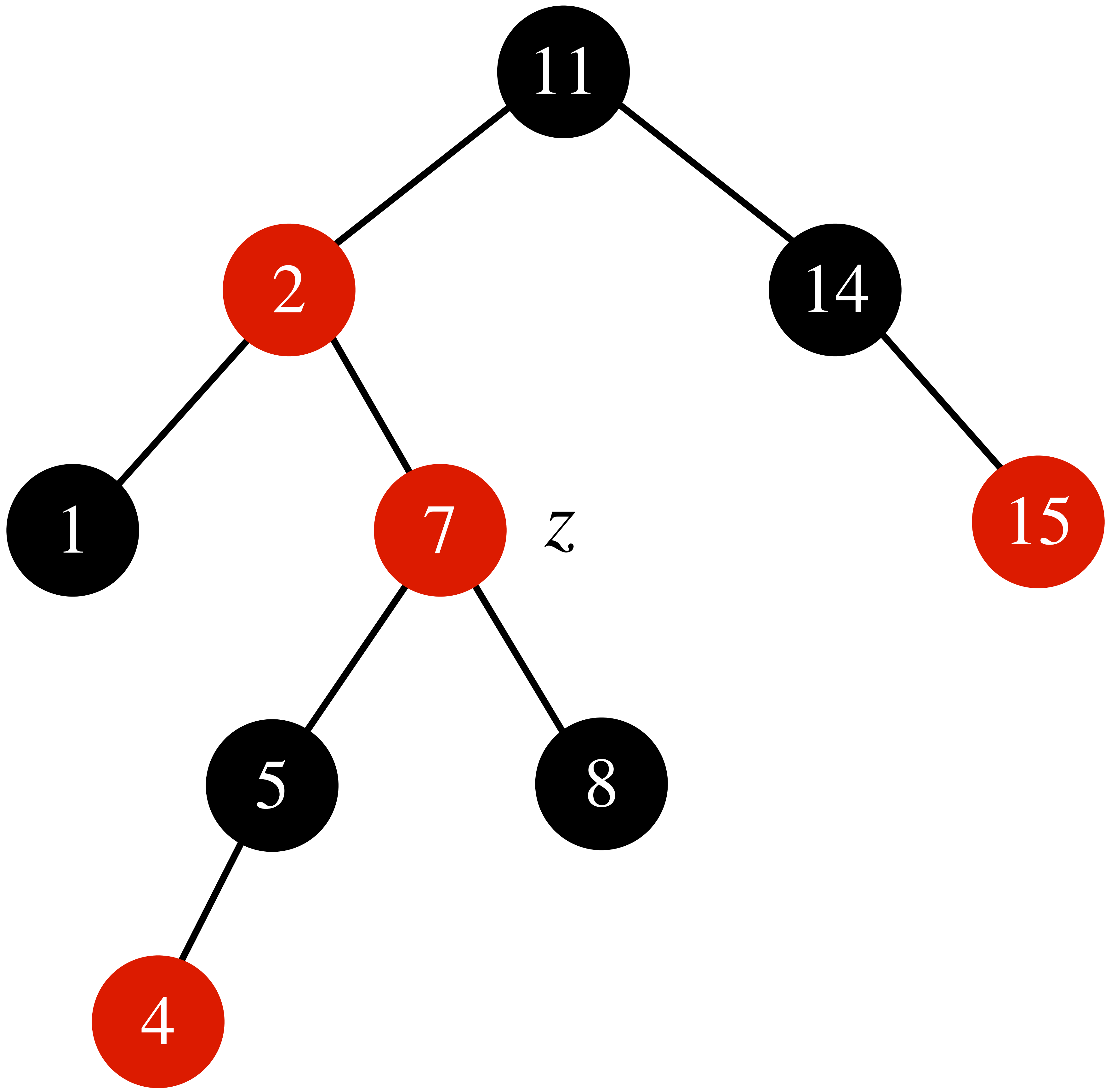
RB-Trees: Insertion Case 1

Case 1: z's **uncle** (sibling of z's parent) is red.



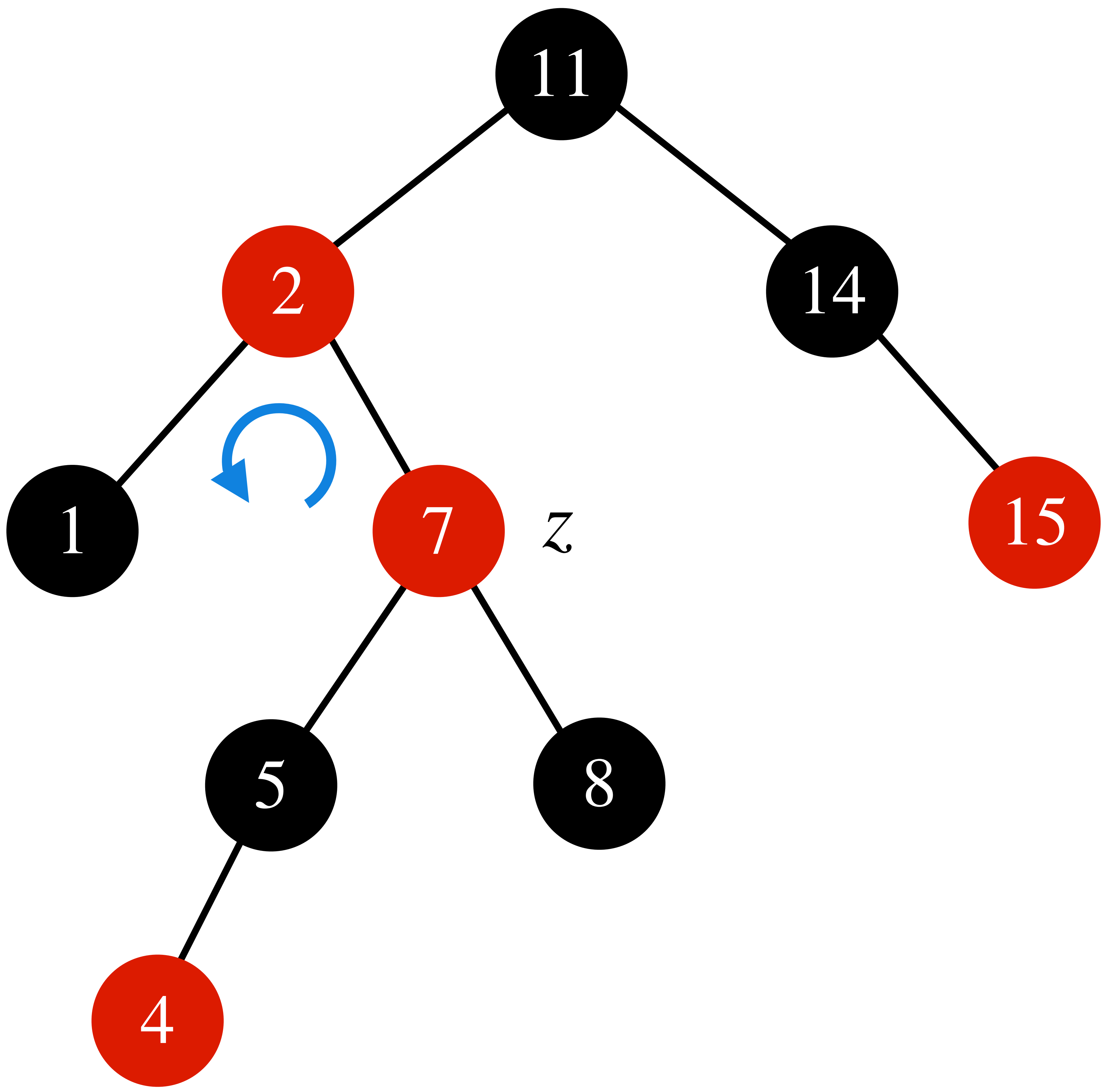
RB-Trees: Insertion Case 2

Case 2: z's uncle is black and z is a right child.



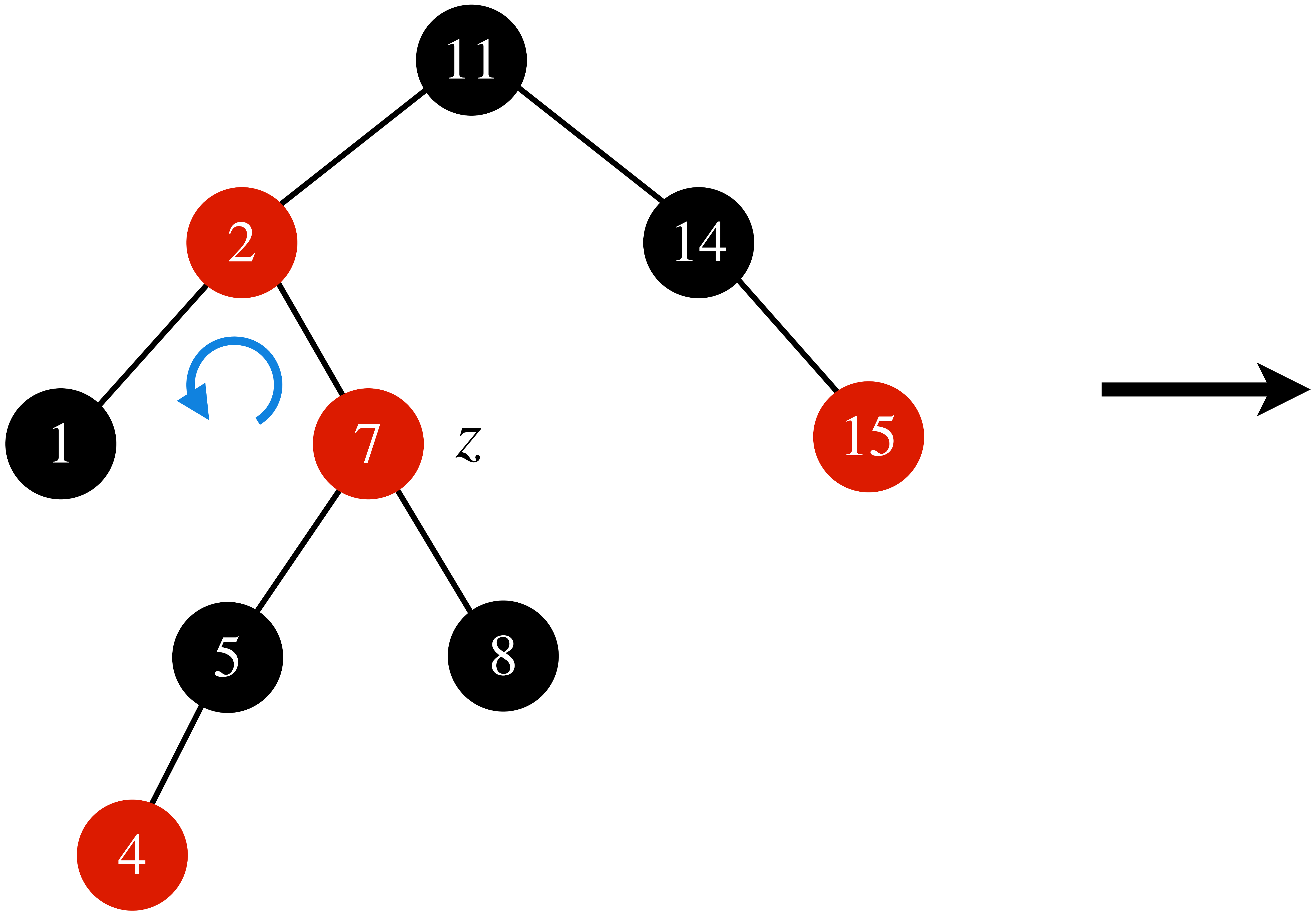
RB-Trees: Insertion Case 2

Case 2: z's uncle is black and z is a right child.



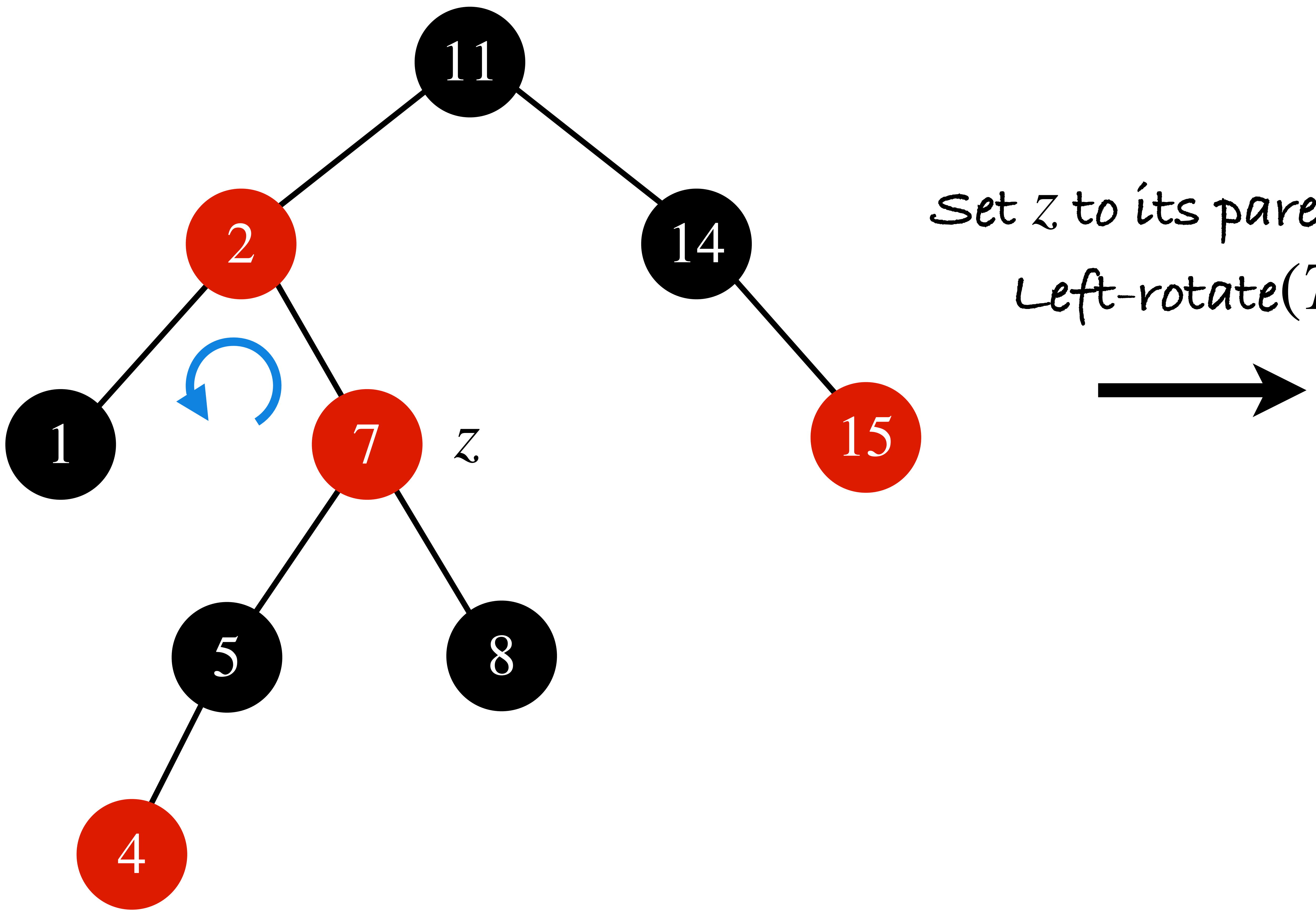
RB-Trees: Insertion Case 2

Case 2**:** z's uncle is black and z is a right child.



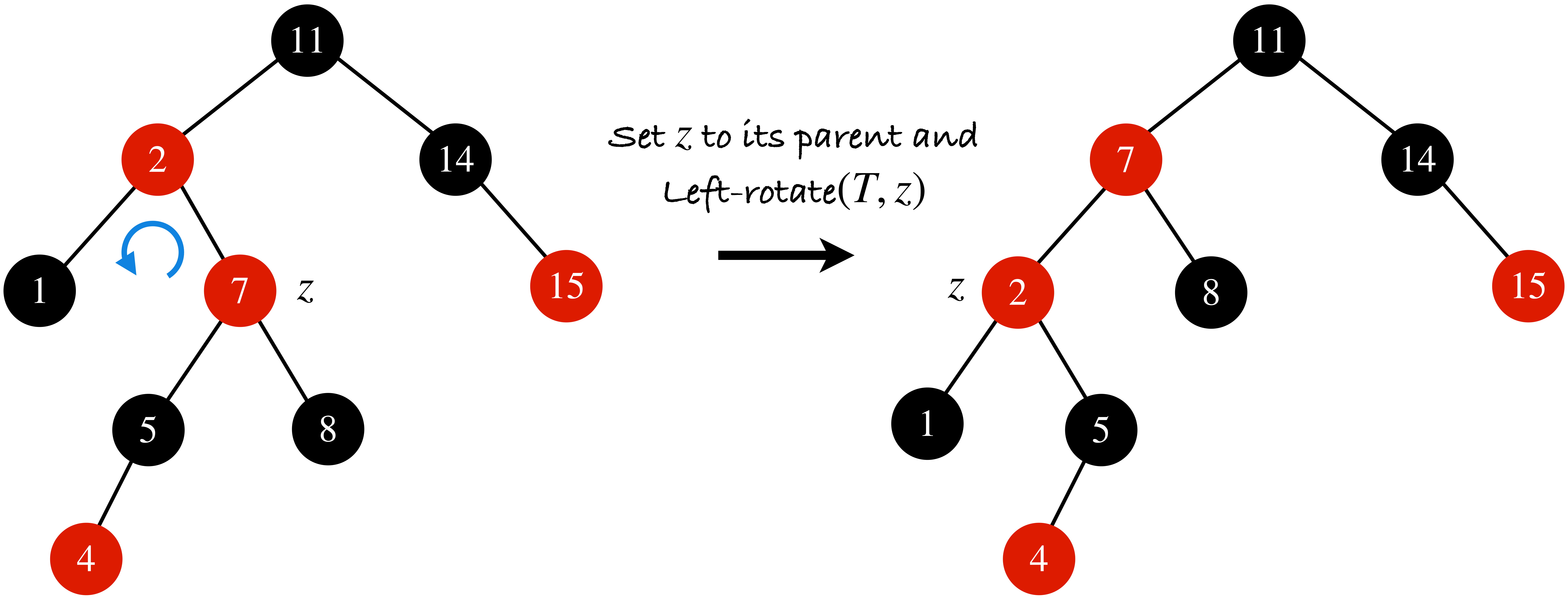
RB-Trees: Insertion Case 2

Case 2**:** z's uncle is black and z is a right child.



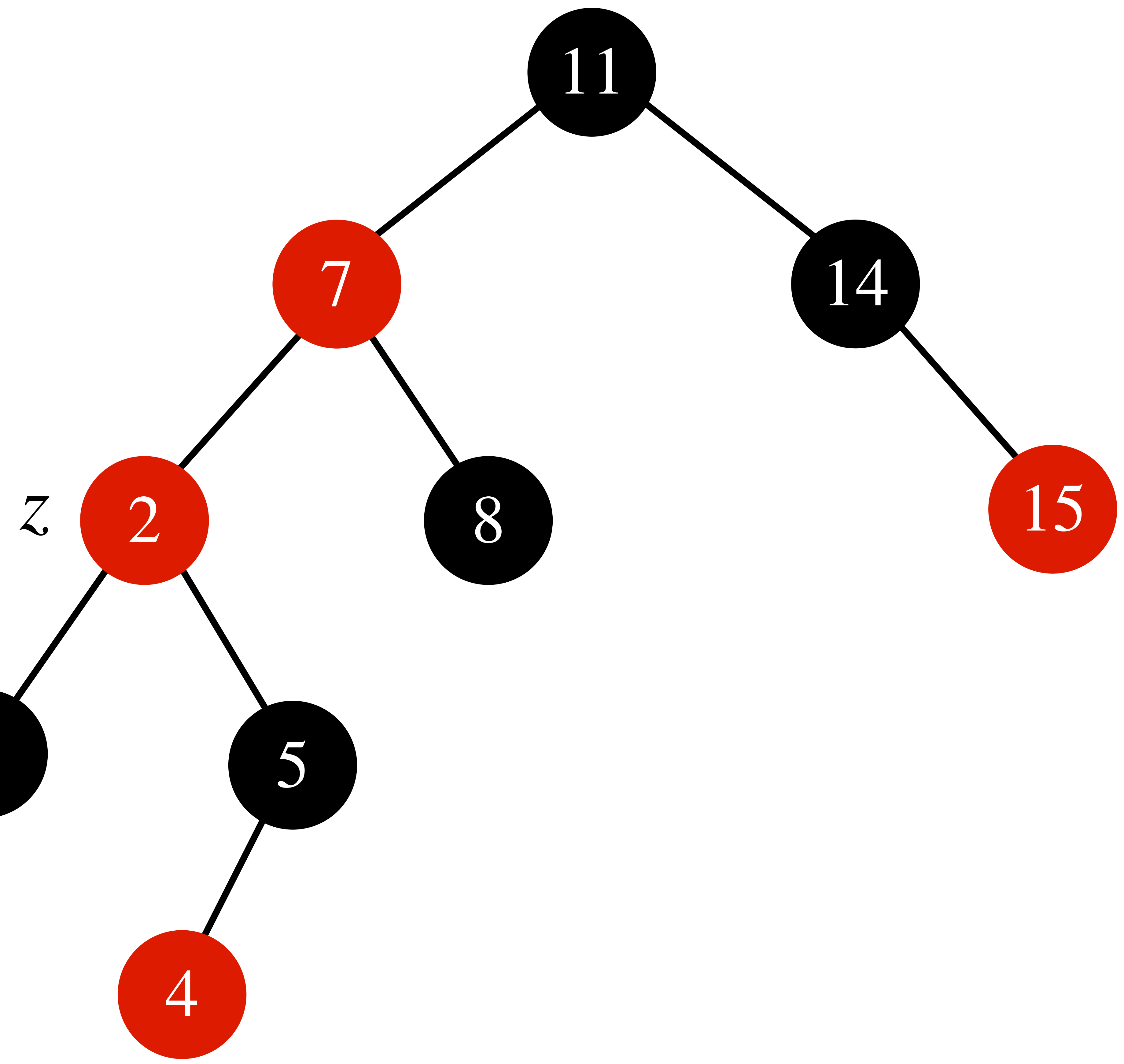
RB-Trees: Insertion Case 2

Case 2**:** z's uncle is black and z is a right child.



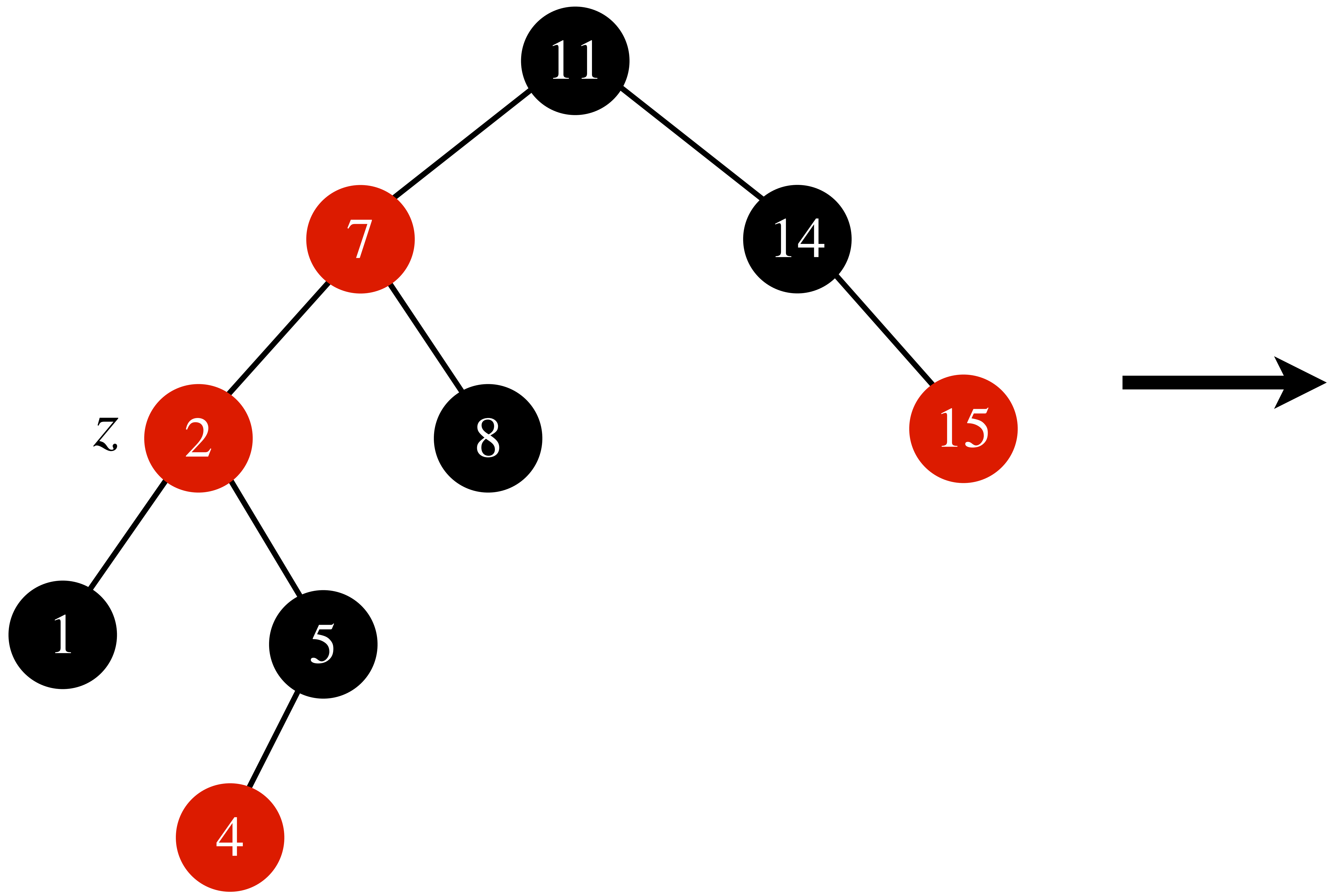
RB-Trees: Insertion Case 3

Case 3**:** z's uncle is black and z is a left child.



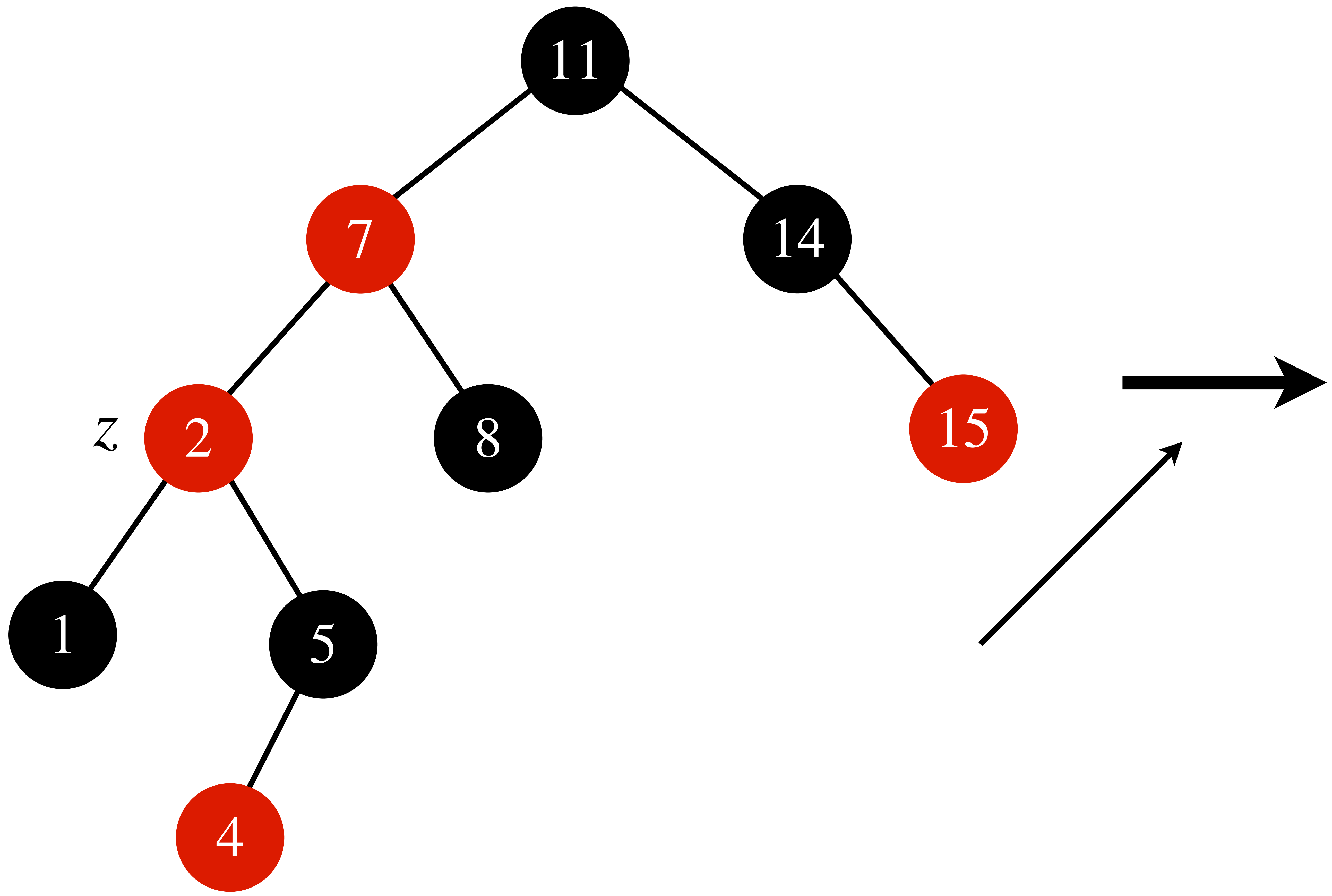
RB-Trees: Insertion Case 3

Case 3**:** z's uncle is black and z is a **left child**.



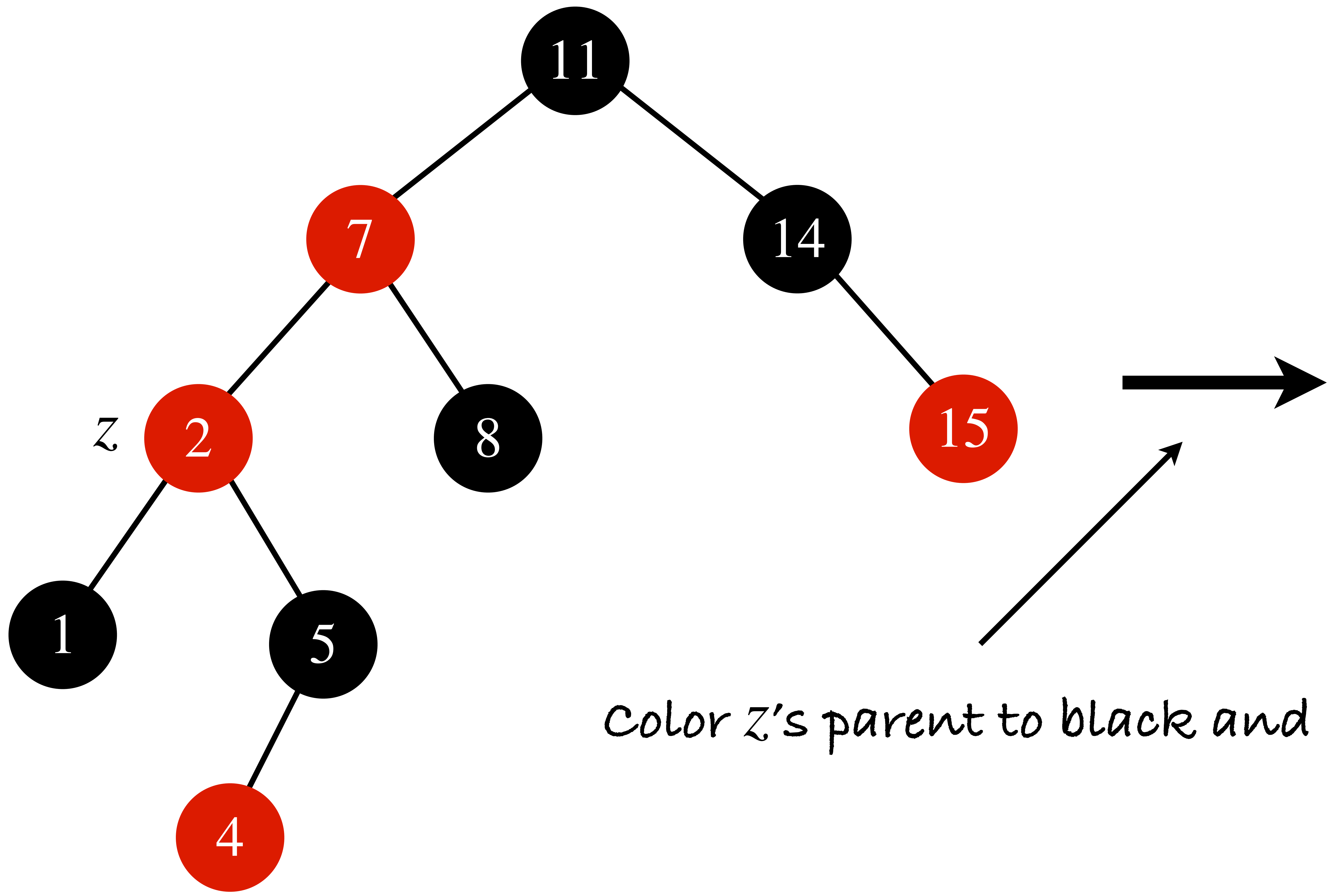
RB-Trees: Insertion Case 3

Case 3**:** z's uncle is black and z is a left child.



RB-Trees: Insertion Case 3

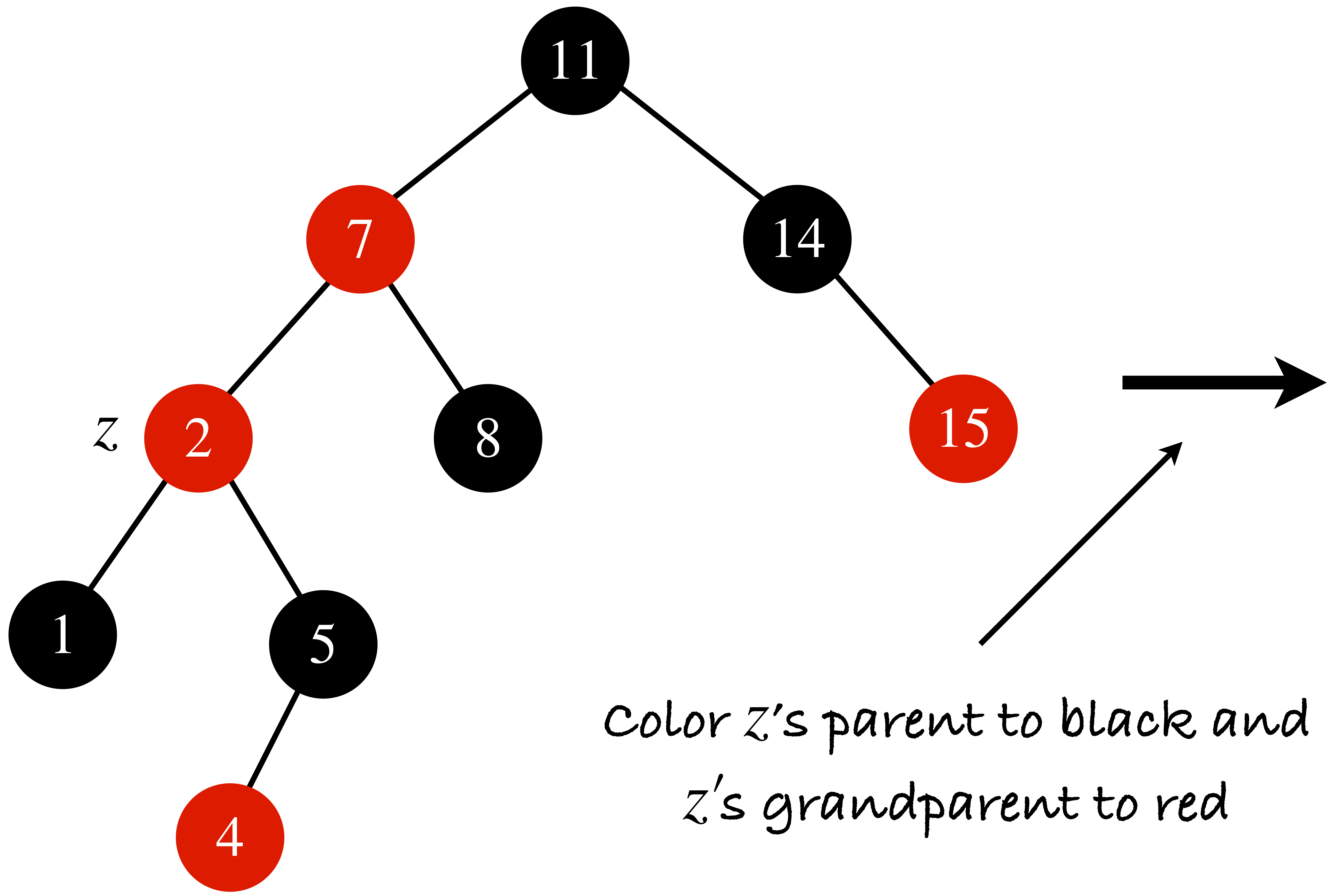
Case 3**:** z's uncle is black and z is a **left child**.



Color z's parent to black and

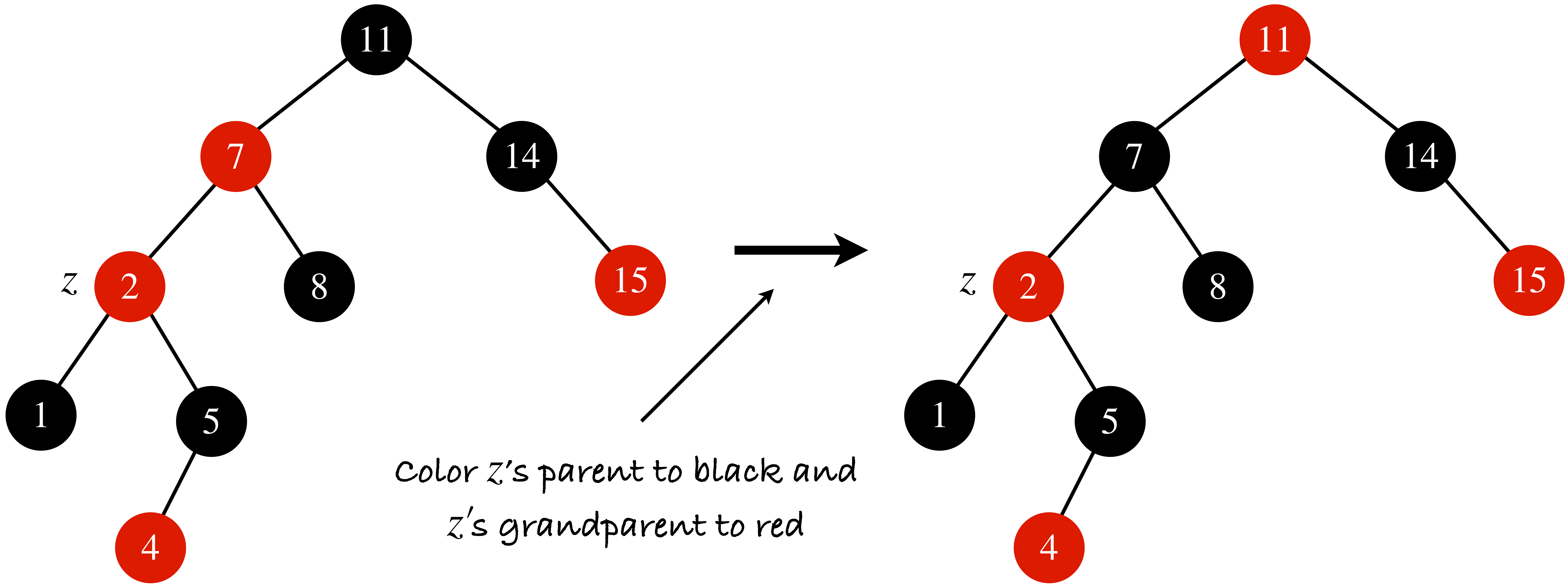
RB-Trees: Insertion Case 3

Case 3**:** z's **uncle is black** and z is a **left child**.



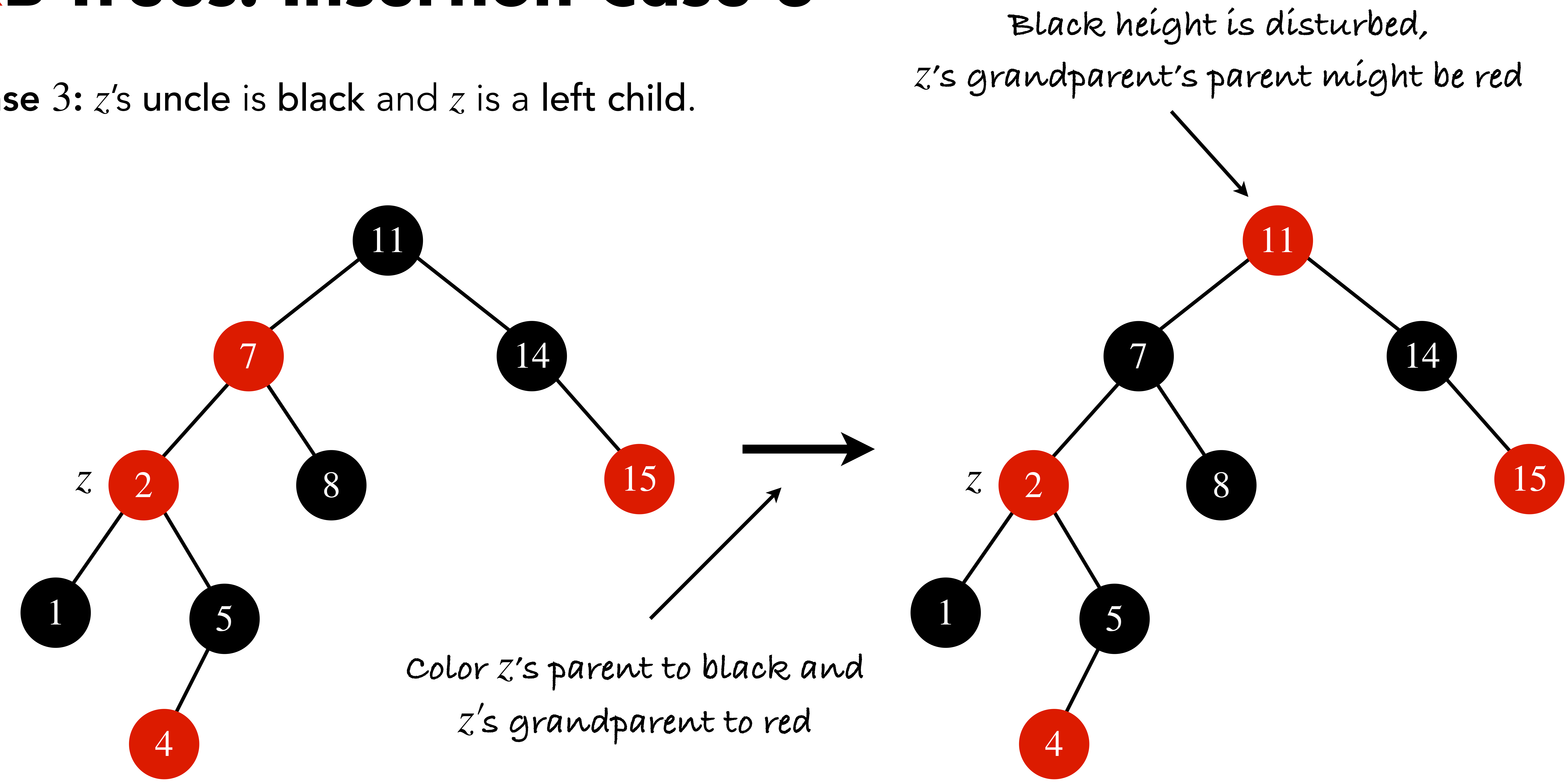
RB-Trees: Insertion Case 3

Case 3**:** z's **uncle is black** and z is a **left child**.



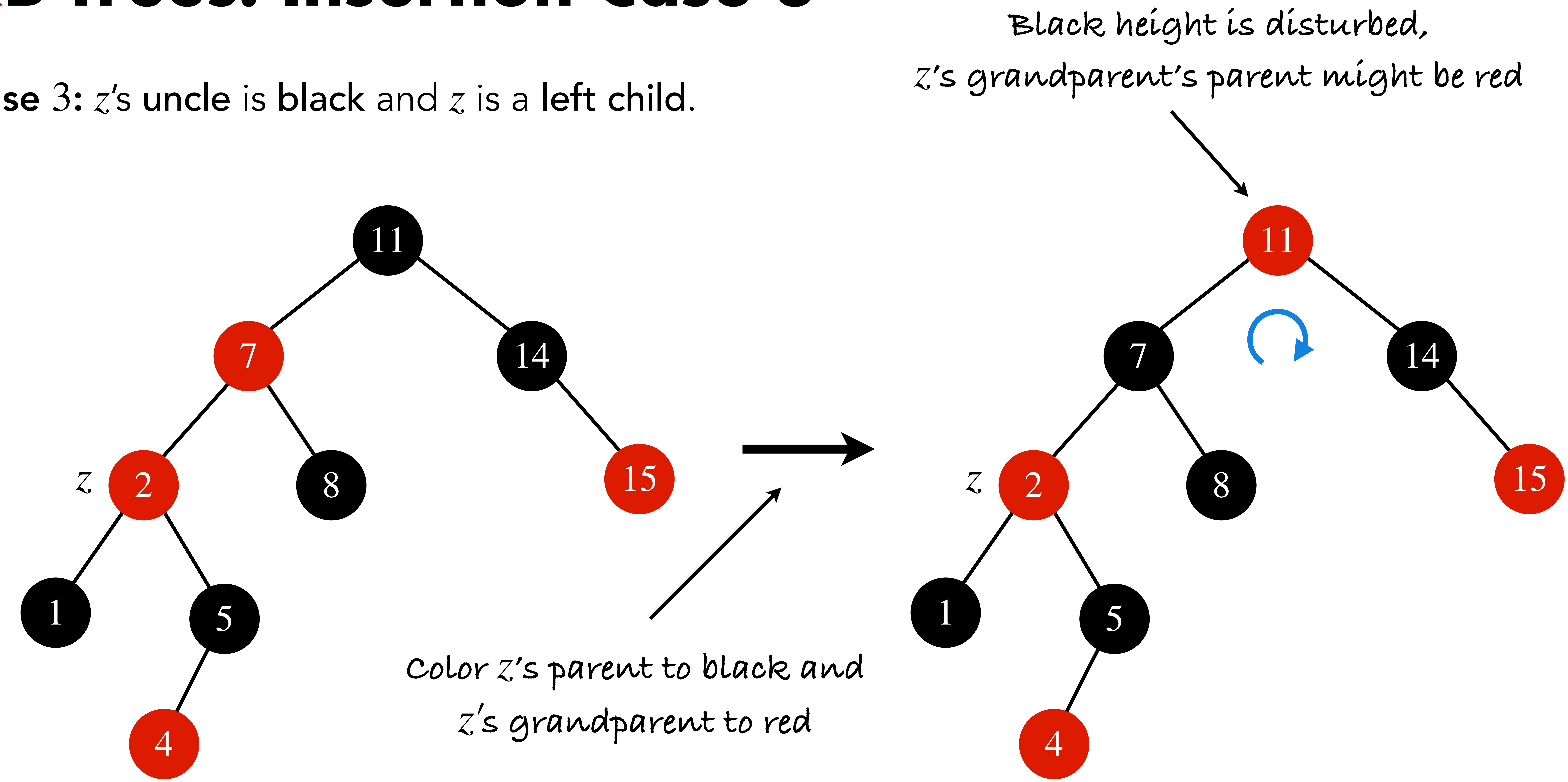
RB-Trees: Insertion Case 3

Case 3**:** z's uncle is black and z is a **left child**.



RB-Trees: Insertion Case 3

Case 3**:** z's uncle is **black** and z is a **left child**.



RB-Trees: Insertion Case 3

Case 3: z's uncle is black and z is a left child.