



<https://algs4.cs.princeton.edu>

## 3.3 BALANCED SEARCH TREES

---

- ▶ *2–3 search trees*
- ▶ *red–black BSTs (representation)*
- ▶ *red–black BSTs (operations)*
- ▶ *context*

# Symbol table review

implementation	guarantee			ordered ops?	key interface
	search	insert	delete		
sequential search (unordered list)	$n$	$n$	$n$		<code>equals()</code>
binary search (sorted array)	$\log n$	$n$	$n$	✓	<code>compareTo()</code>
BST	$n$	$n$	$n$	✓	<code>compareTo()</code>
goal	$\log n$	$\log n$	$\log n$	✓	<code>compareTo()</code>

Challenge.  $\Theta(\log n)$  time in worst case.

optimized for teaching and coding

(introduced in COS 226)

This lecture. 2–3 trees and left-leaning red–black BSTs.

co-invented by Bob Sedgewick in 1970s



## 3.3 BALANCED SEARCH TREES

---

- ▶ *2–3 search trees*
- ▶ *red–black BSTs (representation)*
- ▶ *red–black BSTs (operations)*
- ▶ *context*

ROBERT SEDGEWICK | KEVIN WAYNE

<https://algs4.cs.princeton.edu>

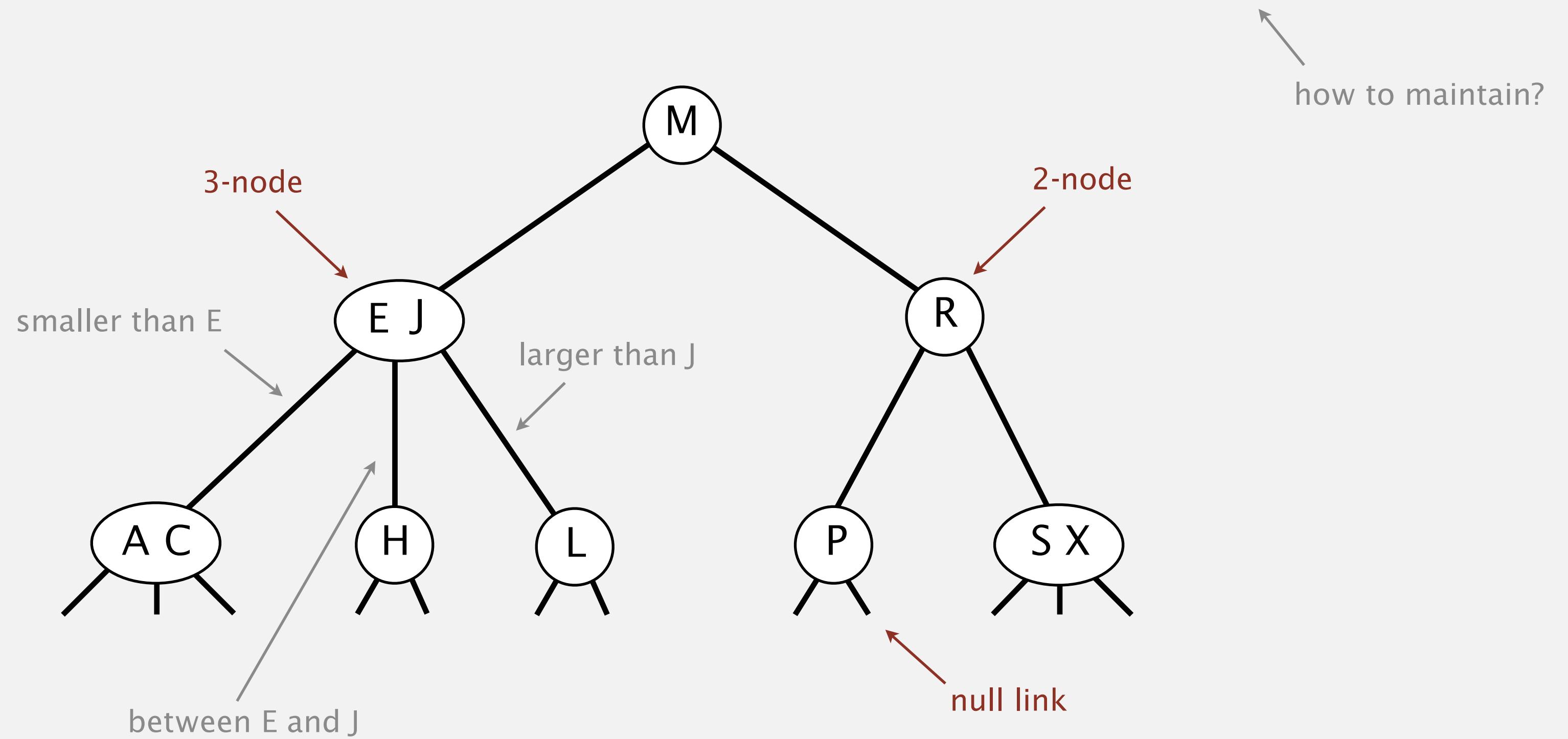
## 2-3 tree

Allow 1 or 2 keys per node.

- 2-node: one key, two children.
- 3-node: two keys, three children.

Symmetric order. Inorder traversal yields keys in ascending order.

Perfect balance. Every path from the root to a null link has the same length.



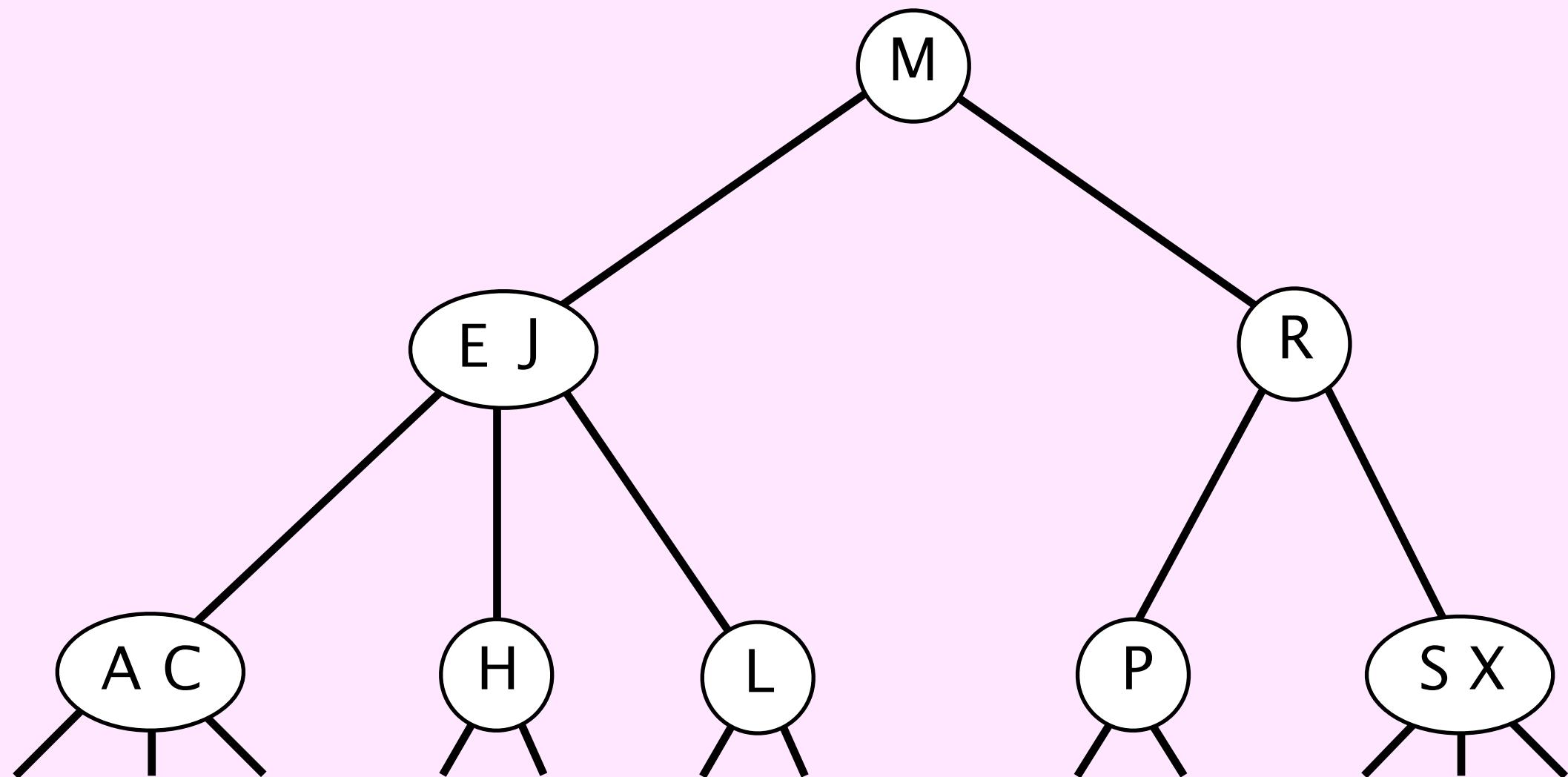


## 2-3 tree demo

Search.

- Compare search key against key(s) in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for H



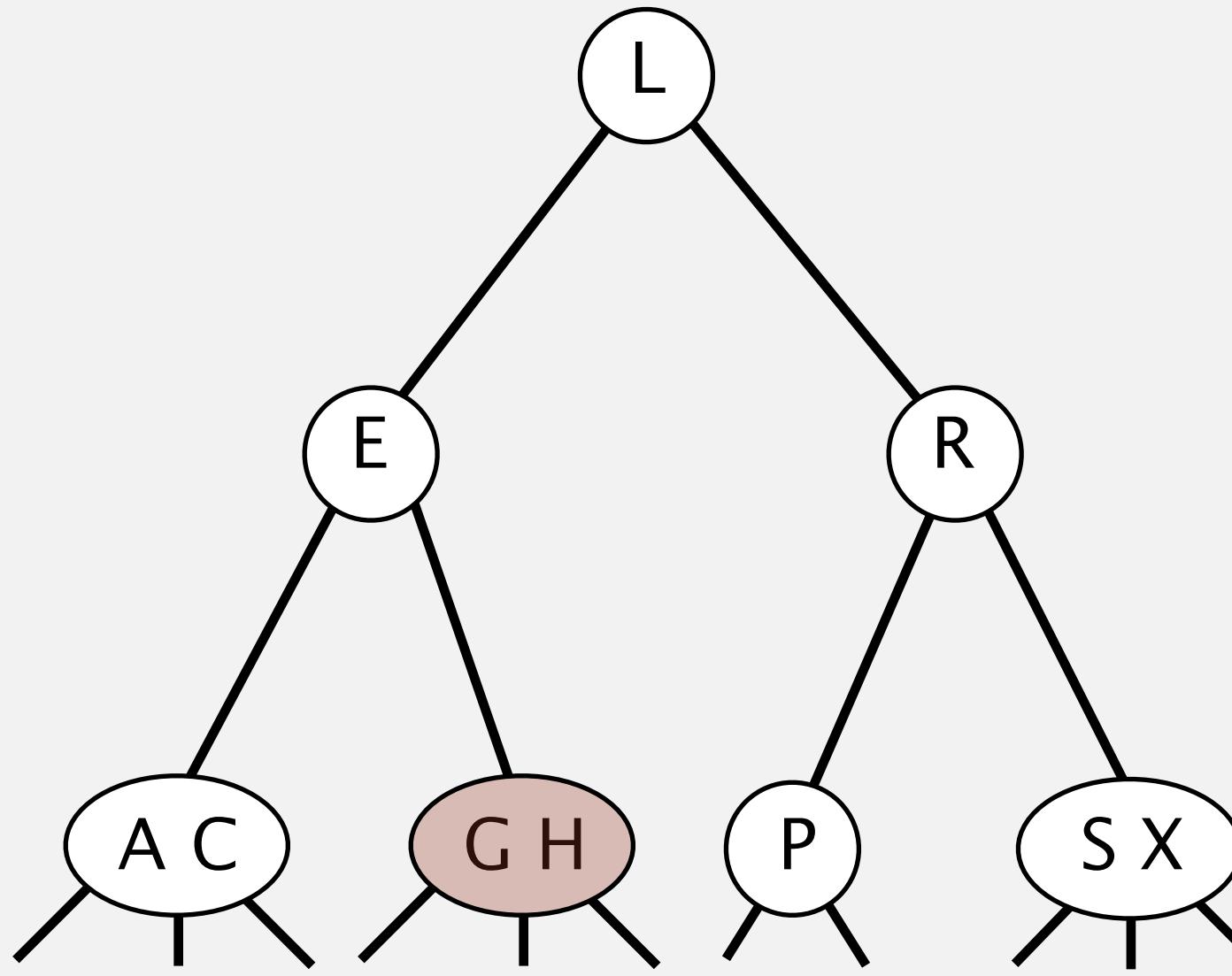
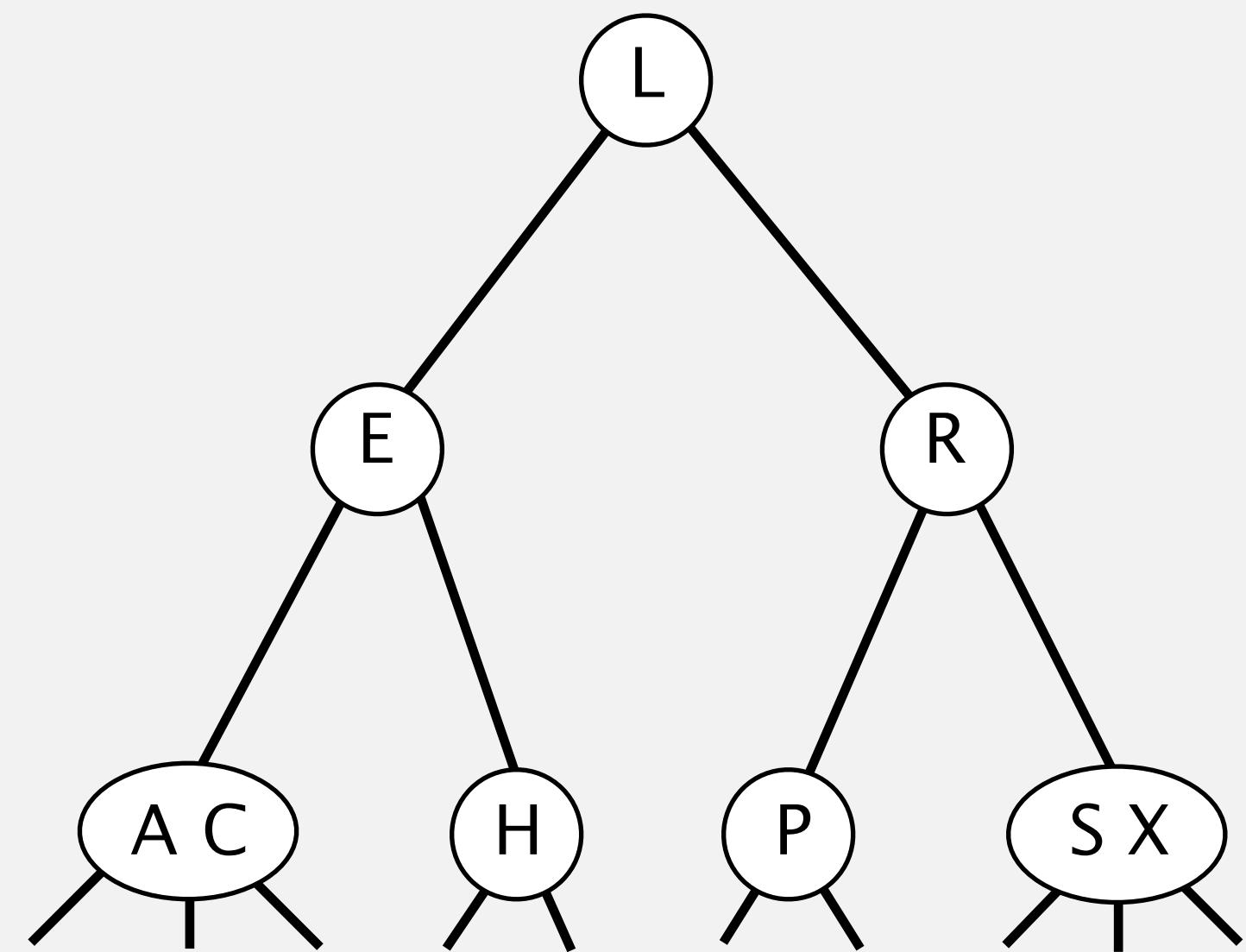
## 2-3 tree: insertion

---

Insertion into a 2-node at bottom.

- Add new key to 2-node to create a 3-node.

**insert G**



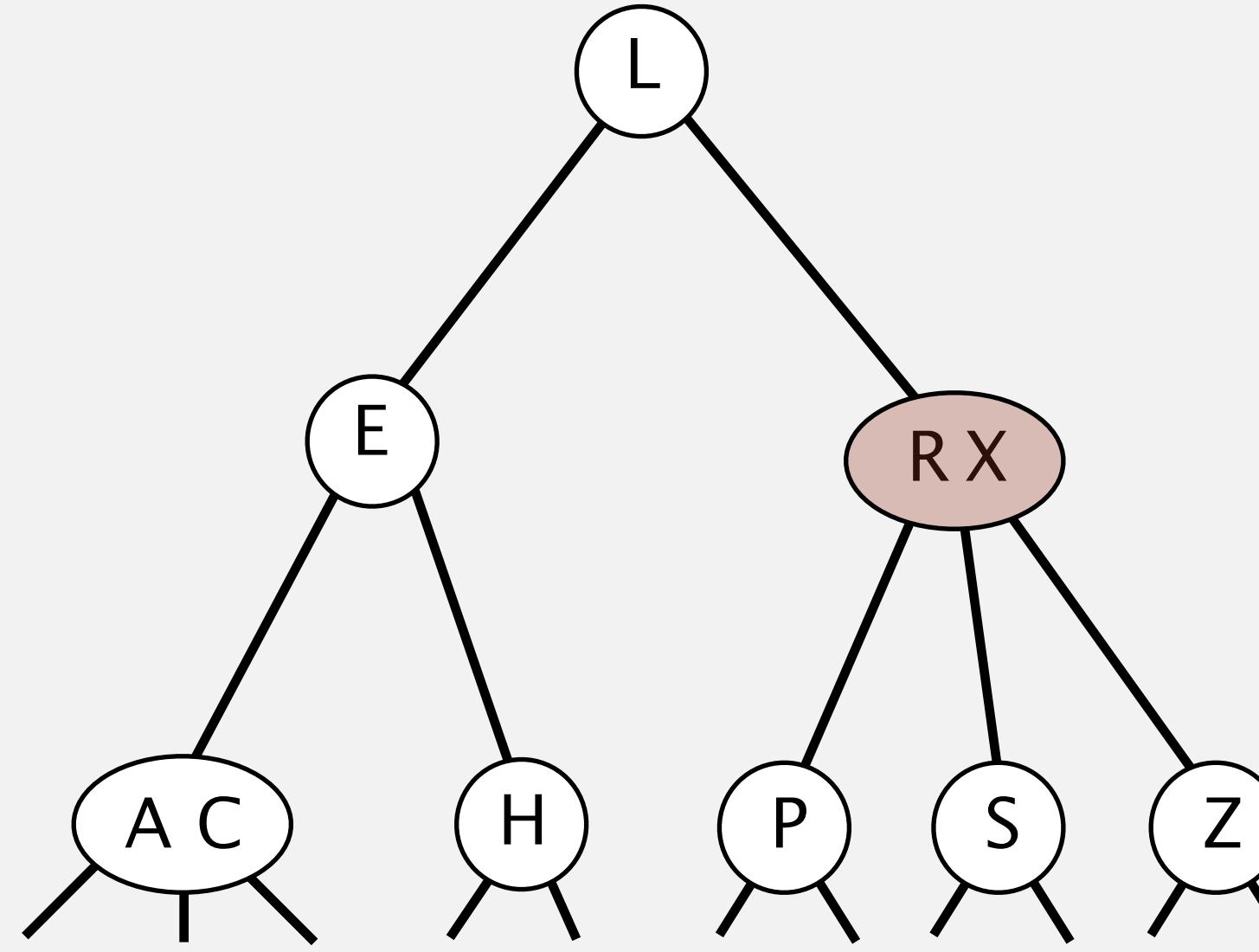
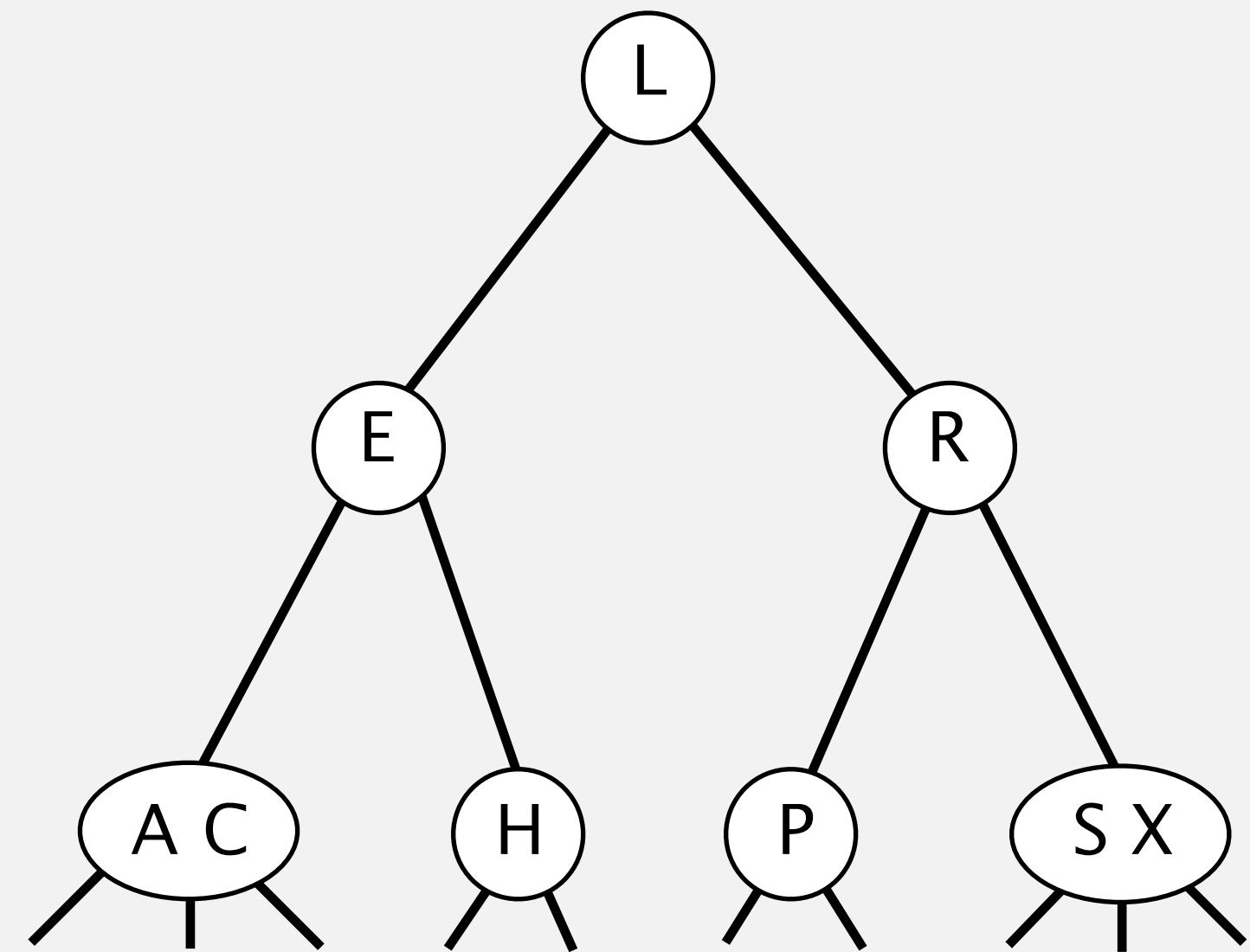
## 2-3 tree: insertion

---

Insertion into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert Z

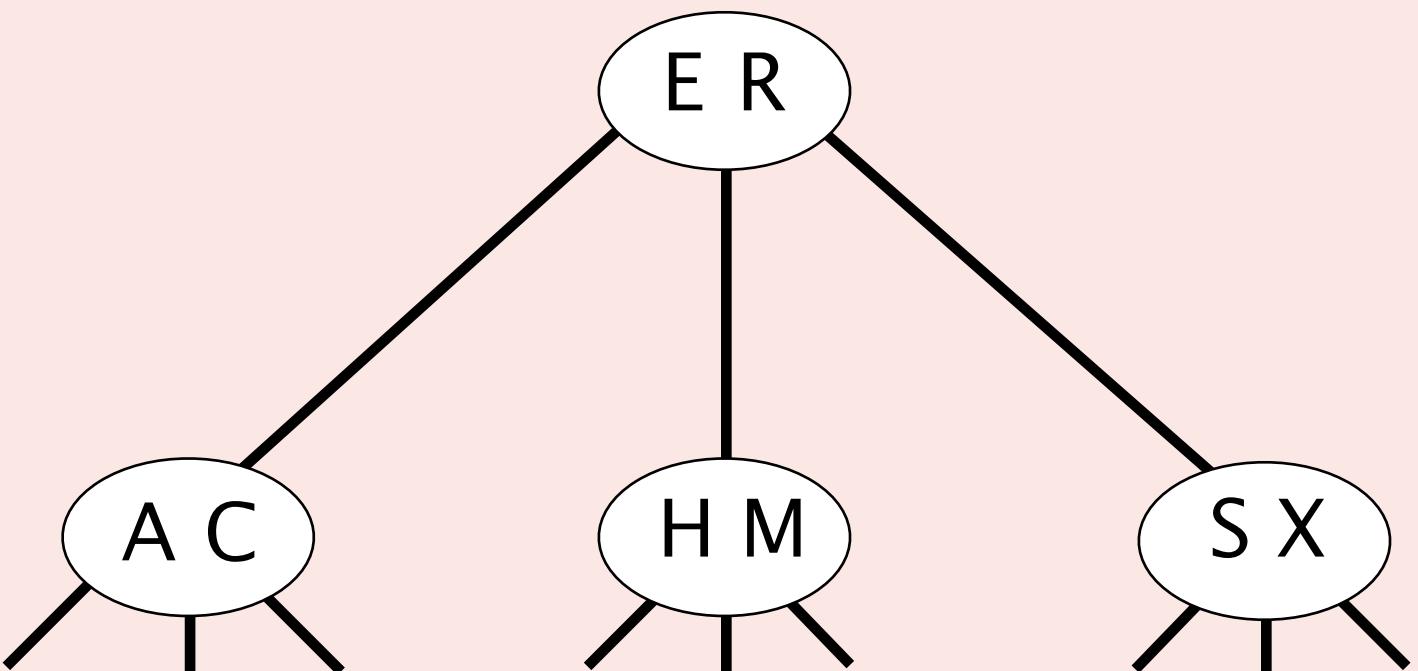


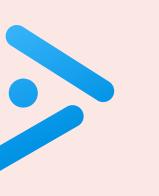


Suppose that you insert P into the following 2–3 tree.

What will be the root of the resulting 2–3 tree?

- A. E
- B. E R
- C. M
- D. P
- E. R





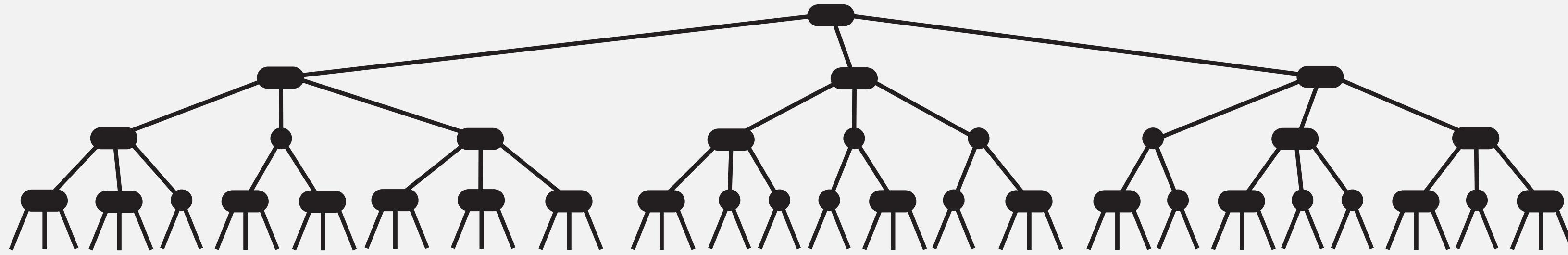
What is the **maximum** height of a 2-3 tree containing  $n$  keys?

- A.  $\sim \log_3 n$
- B.  $\sim \log_2 n$
- C.  $\sim 2 \log_2 n$
- D.  $\sim n$

## 2–3 tree: performance

---

Perfect balance. Every path from the root to a null link has the same length.



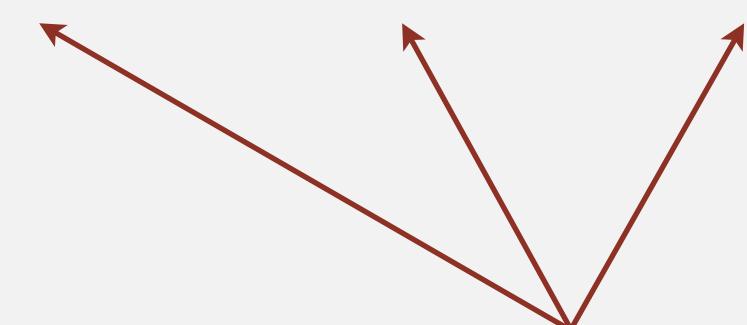
Key property. The height of a 2–3 tree containing  $n$  keys is  $\Theta(\log n)$ .

- Min:  $\log_3 n \approx 0.631 \log_2 n$ . [all 3-nodes]
- Max:  $\log_2 n$ . [all 2-nodes]
- Between 12 and 20 for a million keys.
- Between 18 and 30 for a billion keys.

Bottom line. Search and insert take  $\Theta(\log n)$  time in the worst case.

# ST implementations: summary

implementation	guarantee			ordered ops?	key interface
	search	insert	delete		
sequential search (unordered list)	$n$	$n$	$n$		<code>equals()</code>
binary search (sorted array)	$\log n$	$n$	$n$	✓	<code>compareTo()</code>
BST	$n$	$n$	$n$	✓	<code>compareTo()</code>
2–3 trees	$\log n$	$\log n$	$\log n$	✓	<code>compareTo()</code>



but hidden constant  $c$  is large  
(depends upon implementation)

## 2-3 tree: implementation?

---

Direct implementation is complicated, because:

- Maintaining multiple node types is cumbersome.
- Need multiple compares to move down tree.
- Need to move back up the tree to split 4-nodes.
- Large number of cases for splitting.

**fantasy code**

```
public void put(Key key, Value val)
{
    Node x = root;
    while (x.getTheCorrectChild(key) != null)
    {
        x = x.getTheCorrectChildKey();
        if (x.is4Node()) x.split();
    }
    if (x.is2Node()) x.make3Node(key, val);
    else if (x.is3Node()) x.make4Node(key, val);
}
```

Bottom line. Could do it, but there's a better way.



ROBERT SEDGEWICK | KEVIN WAYNE

<https://algs4.cs.princeton.edu>

## 3.3 BALANCED SEARCH TREES

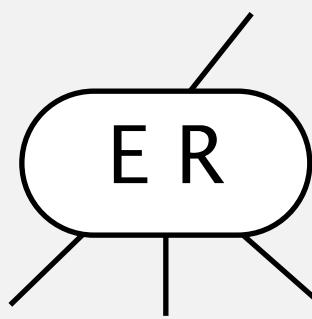
---

- ▶ 2–3 search trees
- ▶ red–black BSTs (representation)
- ▶ red–black BSTs (operations)
- ▶ context

# How to implement 2–3 trees as binary search trees?

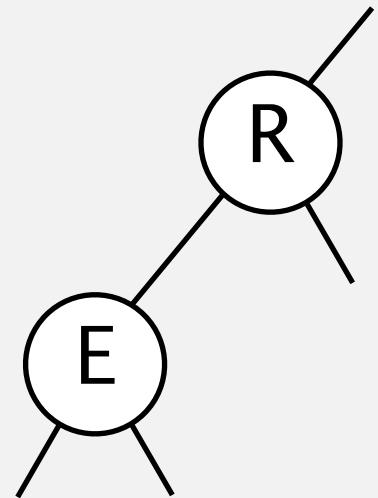
---

Challenge. How to represent a 3 node?



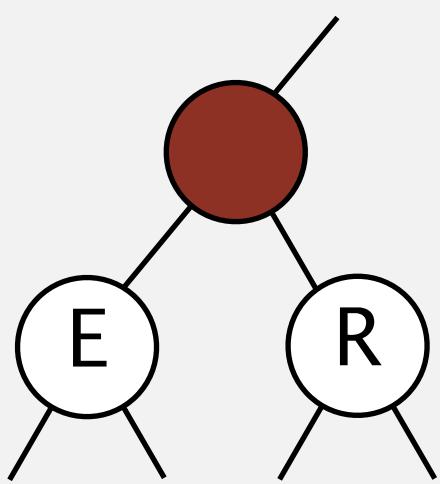
Approach 1. Two BST nodes.

- No way to tell a 3-node from two 2-nodes.
- Can't (uniquely) map from BST back to 2–3 tree.



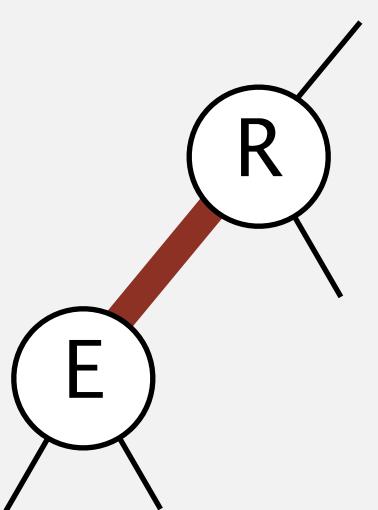
Approach 2. Two BST nodes, plus red “glue” node.

- Wastes space for extra node.
- Messy code.



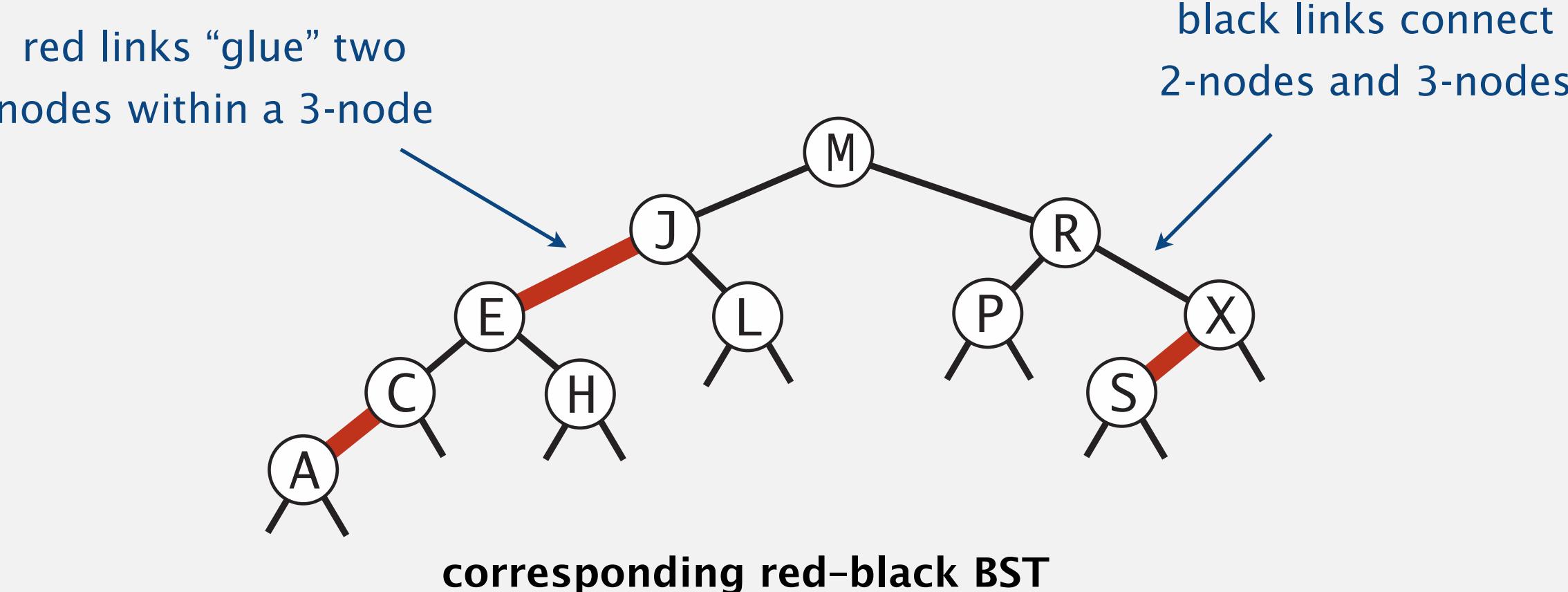
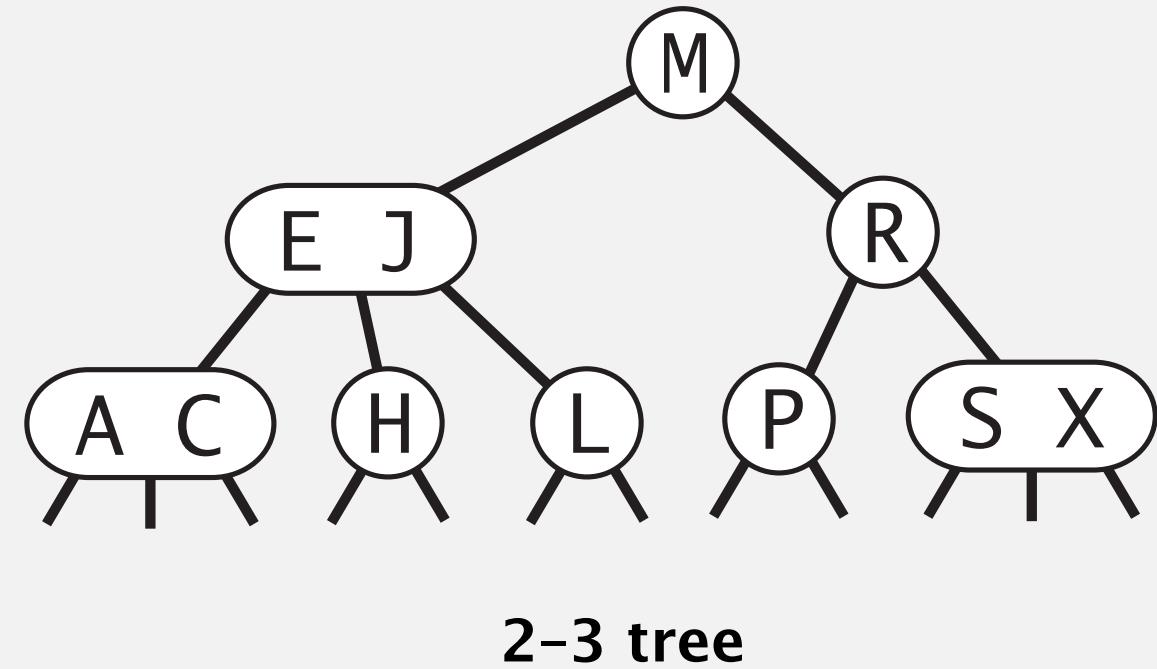
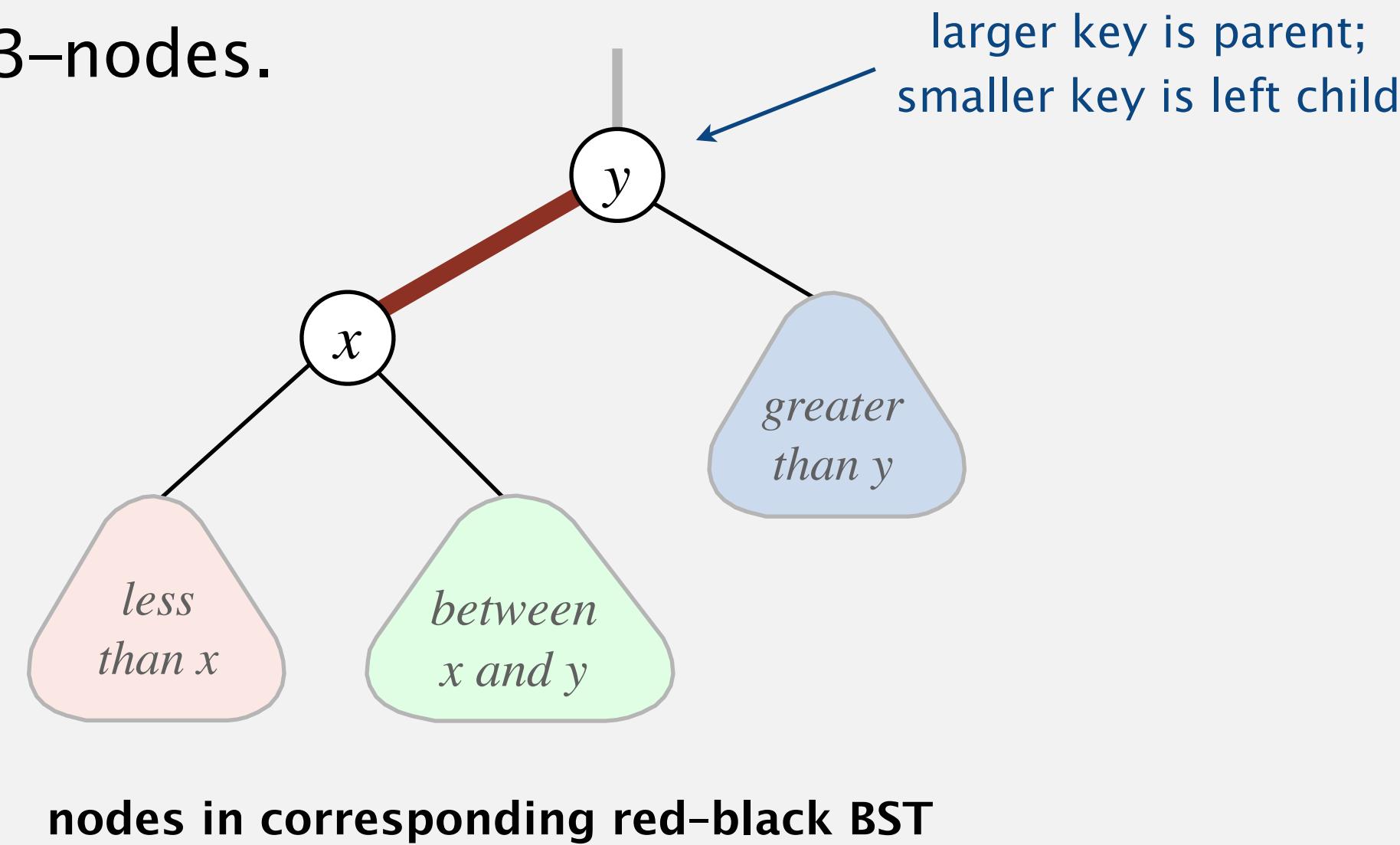
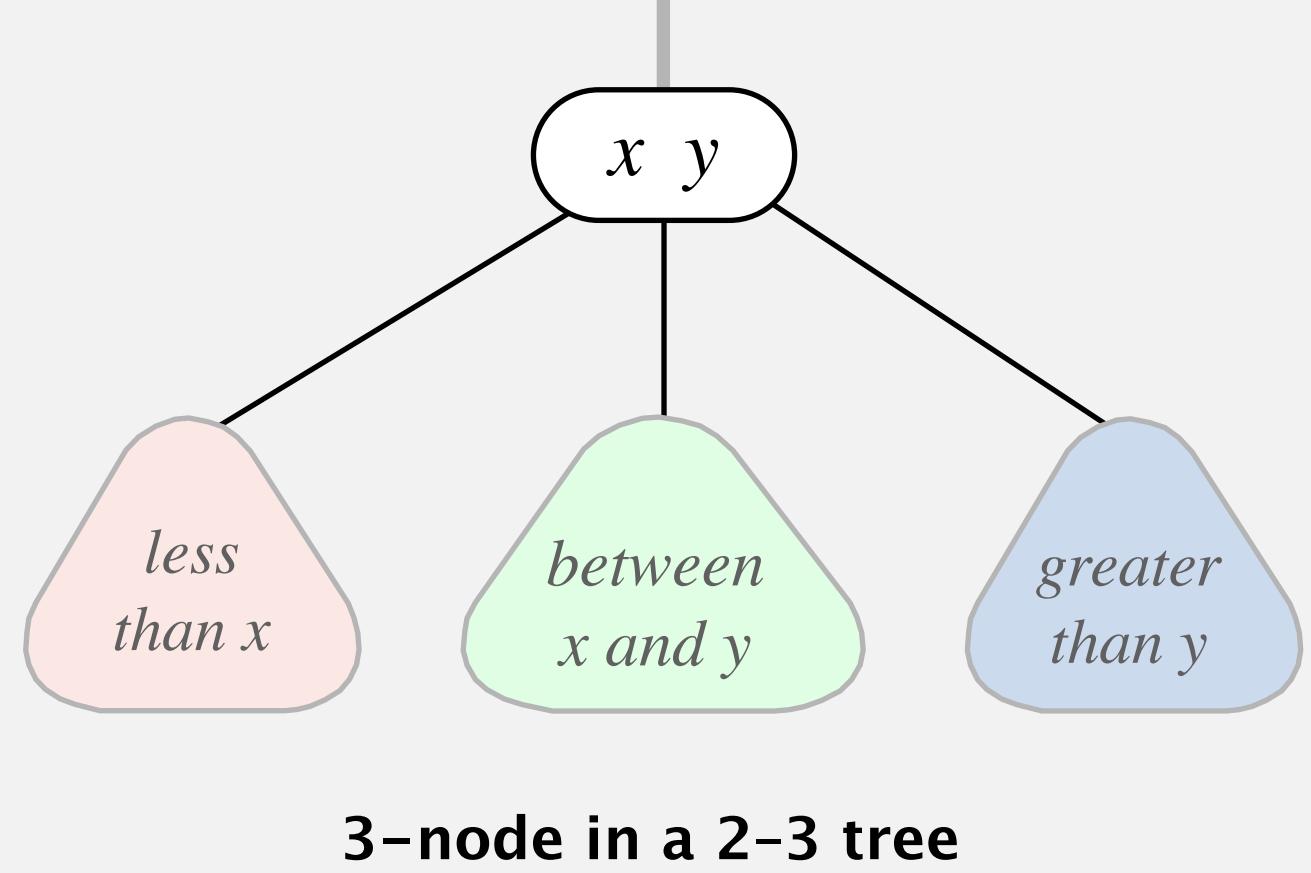
Approach 3. Two BST nodes, with red “glue” link.

- Widely used in practice.
- Arbitrary restriction: red links lean left.



# Left-leaning red-black BSTs

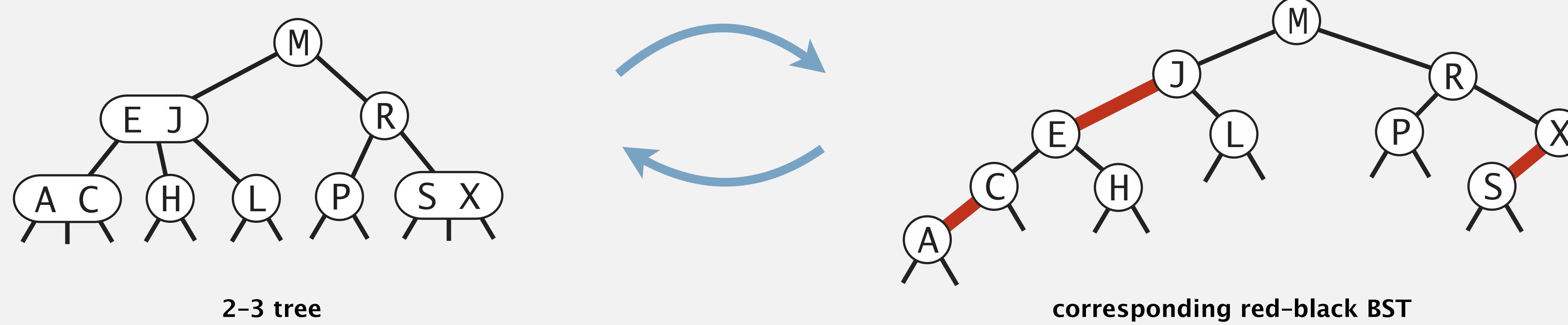
1. Represent 2–3 tree as a BST.
2. Use “internal” left-leaning red links as “glue” for 3-nodes.



## Left-leaning red-black BSTs: 1-1 correspondence with 2-3 trees

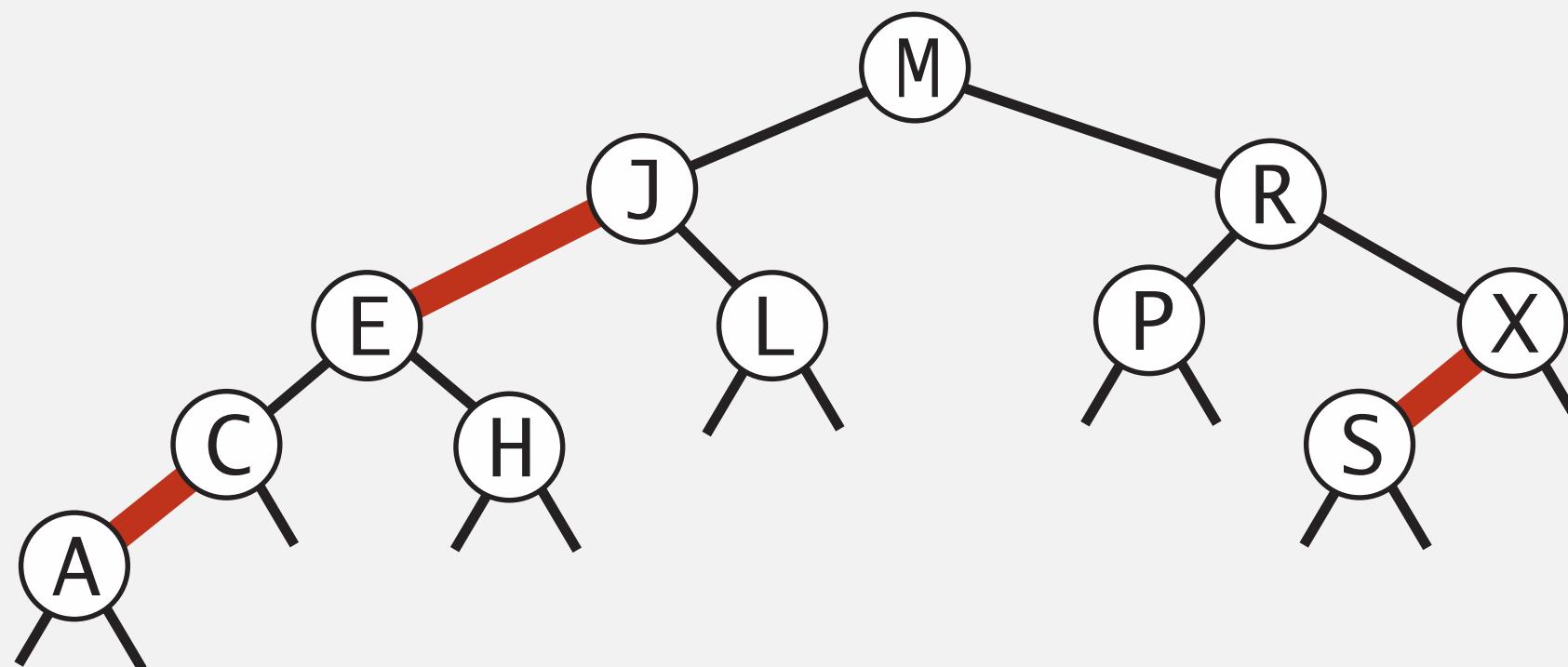
---

Key property. 1-1 correspondence between 2-3 trees and LLRB trees.



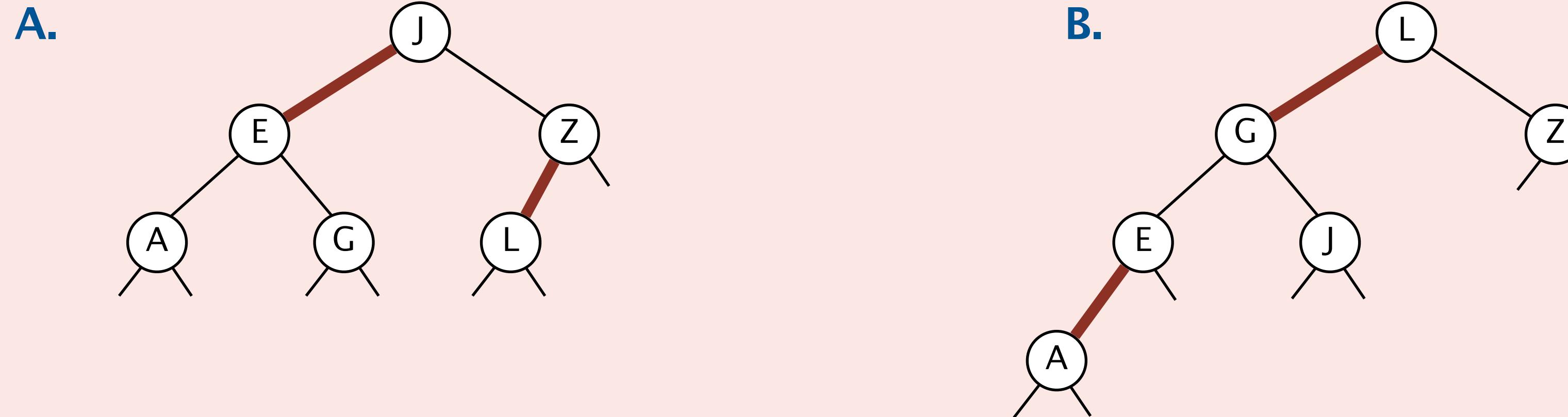
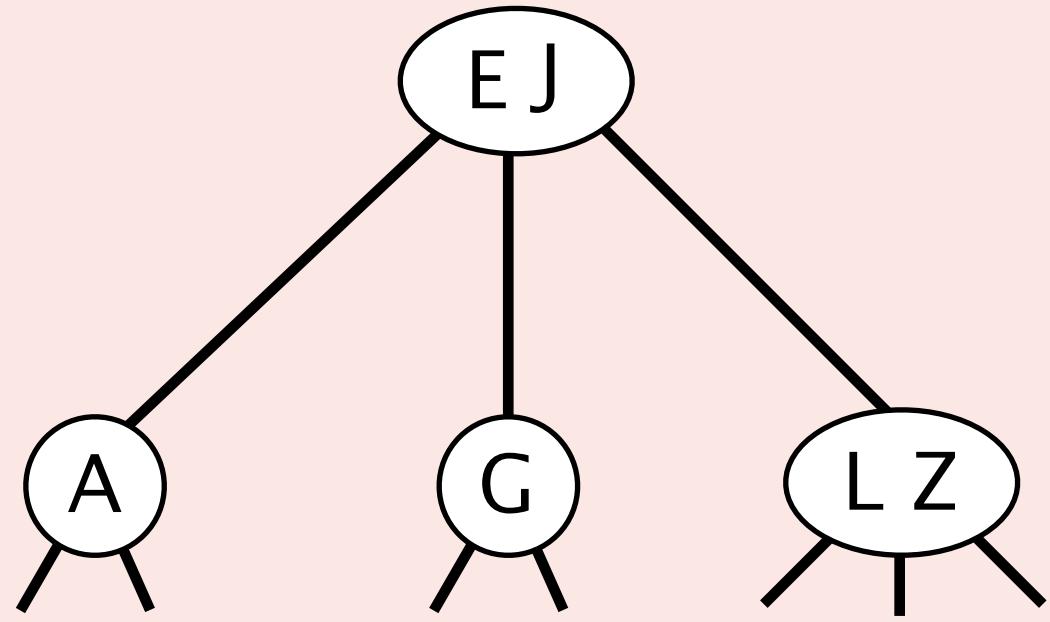
## An equivalent definition of LLRB trees (without reference to 2-3 trees)

- Def. A **red-black BST** is a BST such that:
- No node has two red links connected to it.
  - Red links lean left.
  - Every path from root to null link has the same number of black links.
- | ← color invariants  
↑  
“perfect black balance”





Which LLRB tree corresponds to the following 2-3 tree?



- C. Both A and B.
- D. Neither A nor B.

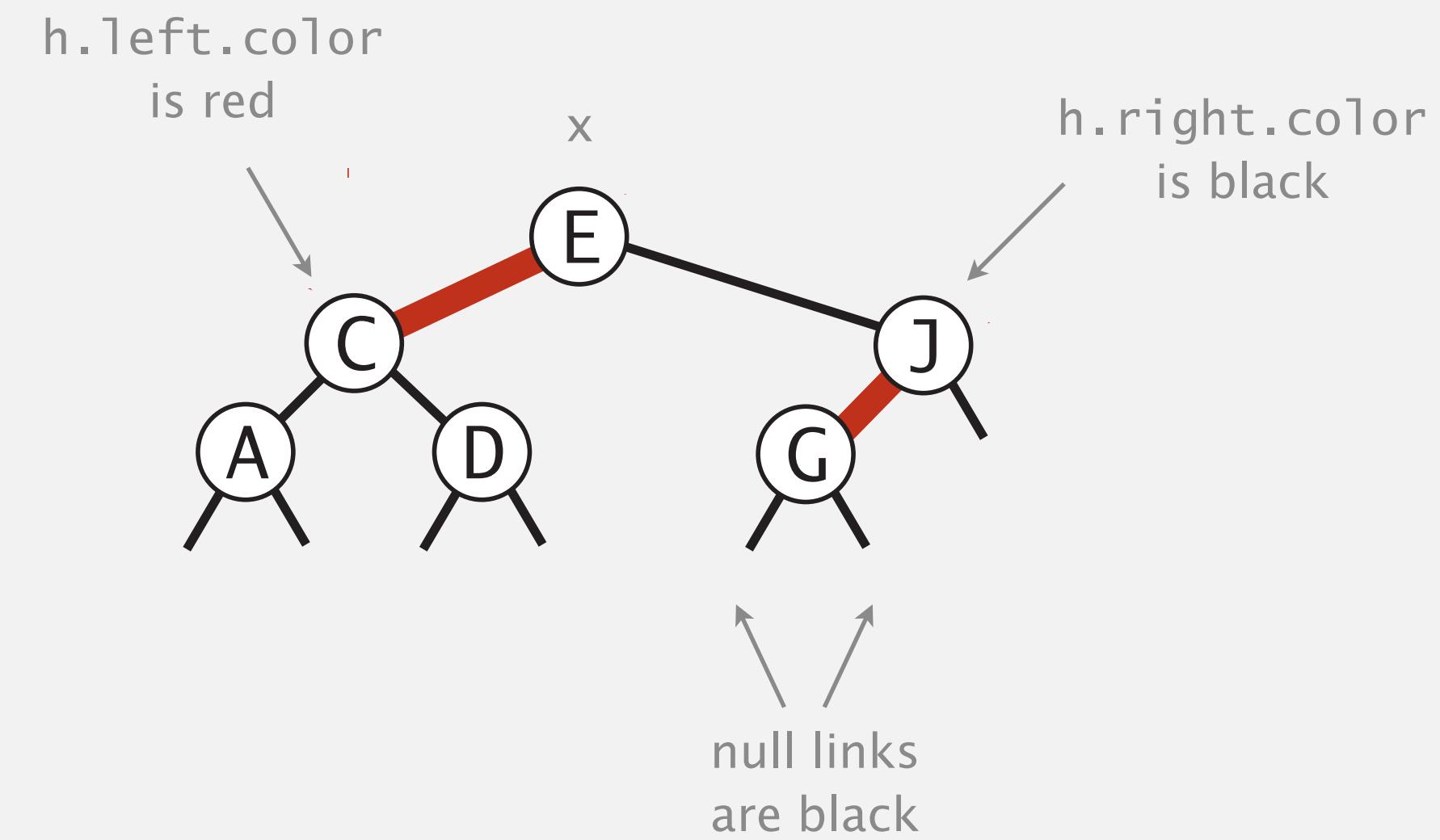
# Red-black BST representation

Each node is pointed to by precisely one link (from its parent)  $\Rightarrow$   
can encode color of links in nodes.

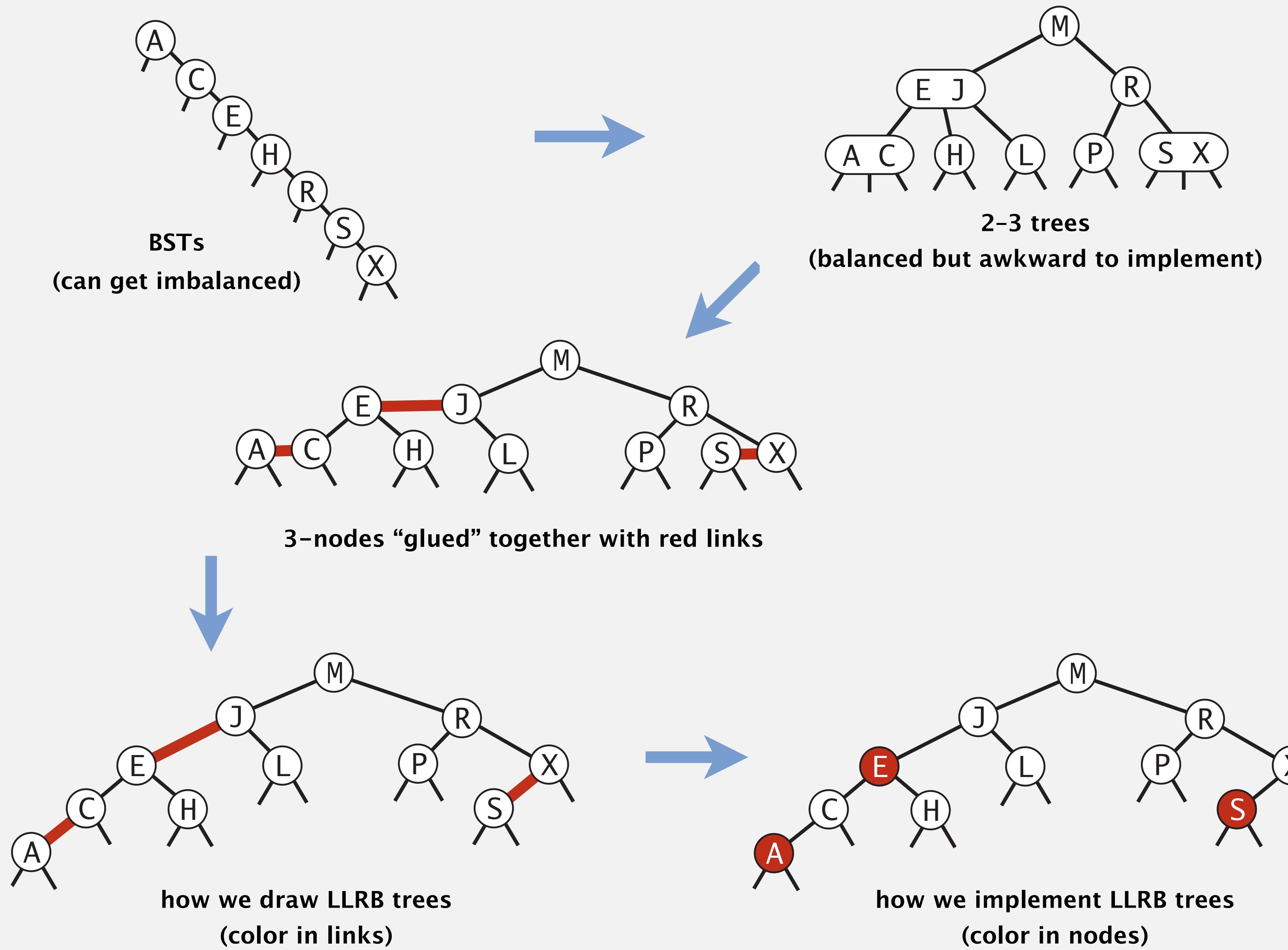
```
private static final boolean RED  = true;
private static final boolean BLACK = false;

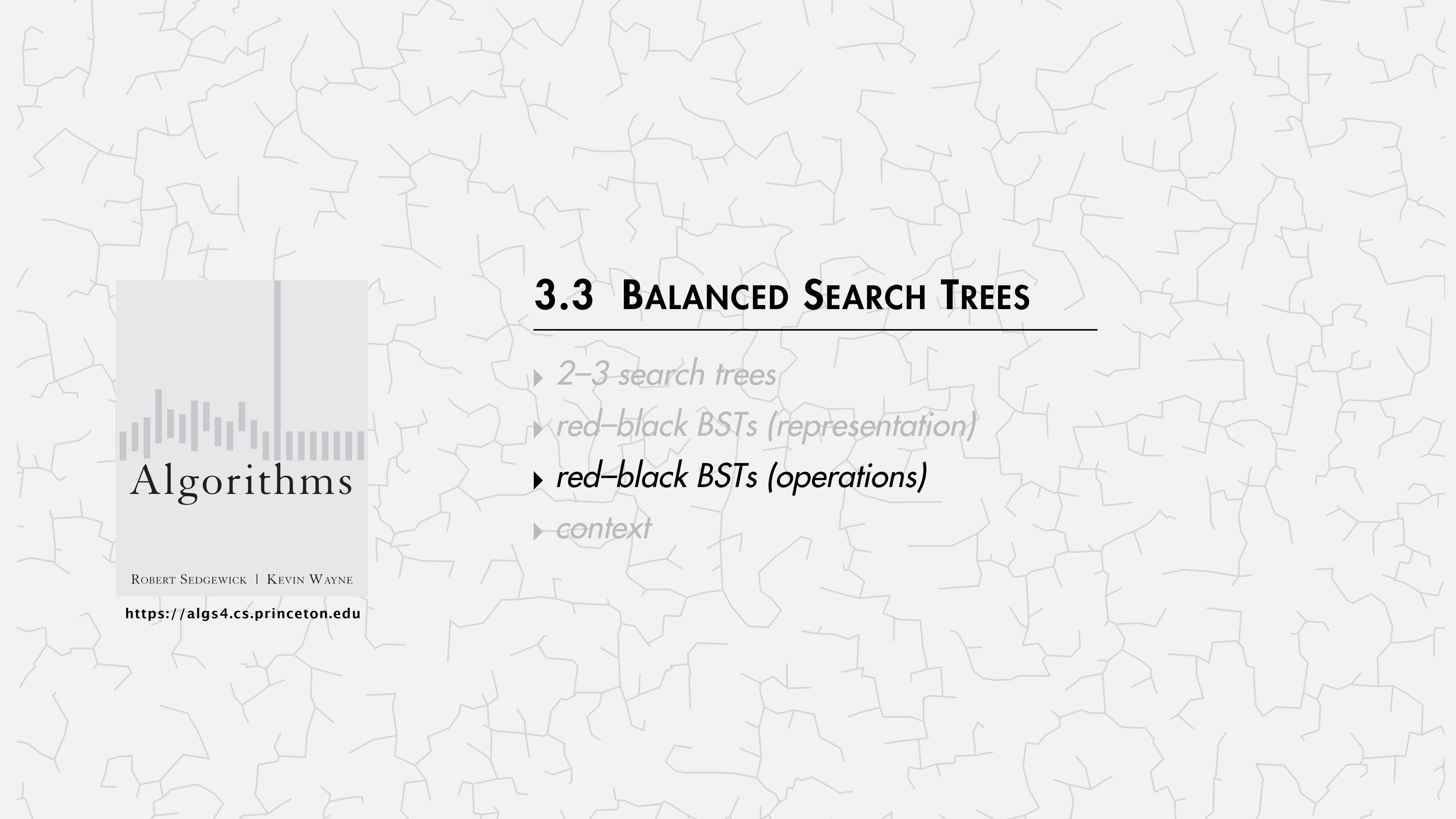
private class Node
{
    private Key key;
    private Value val;
    private Node left, right;
    private boolean color;   ← color of parent link
}

private boolean isRed(Node h)
{
    if (h == null) return false;
    return h.color == RED;   ← null links are black
}
```



## Review: the road to LLRB trees





ROBERT SEDGEWICK | KEVIN WAYNE

<https://algs4.cs.princeton.edu>

# Algorithms

## 3.3 BALANCED SEARCH TREES

---

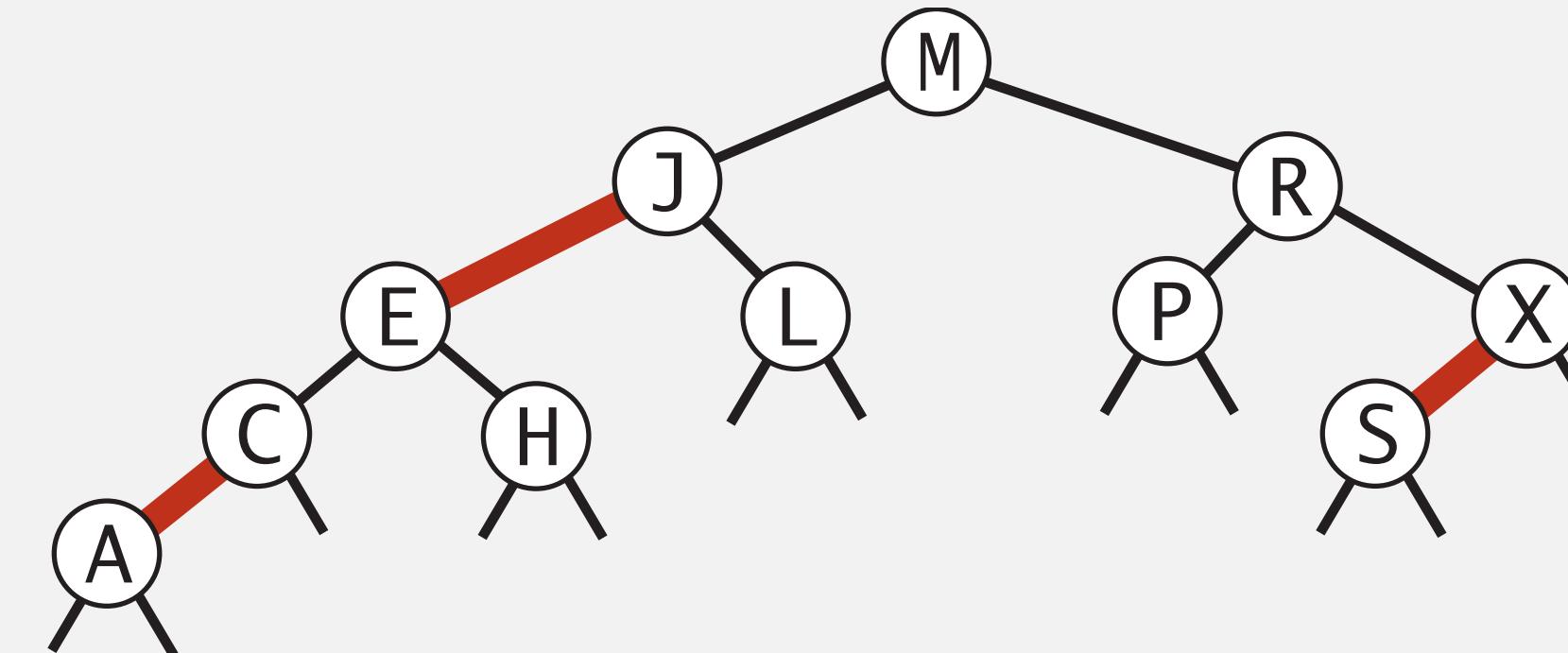
- ▶ 2–3 search trees
- ▶ red–black BSTs (representation)
- ▶ red–black BSTs (operations)
- ▶ context

## Search in a red-black BST

Observation. Red-black BSTs are BSTs  $\Rightarrow$  search is the same as for BSTs (ignore color).

but runs faster  
(because of better balance)

```
public Value get(Key key)
{
    Node x = root;
    while (x != null)
    {
        int cmp = key.compareTo(x.key);
        if      (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
        else return x.val;
    }
    return null;
}
```



Remark. Many other operations (iteration, floor, rank, selection) are also identical.

## Insertion into a LLRB tree: overview

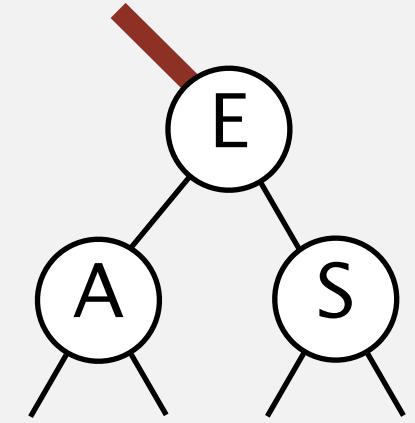
---

Basic strategy. Maintain 1–1 correspondence with 2–3 trees.

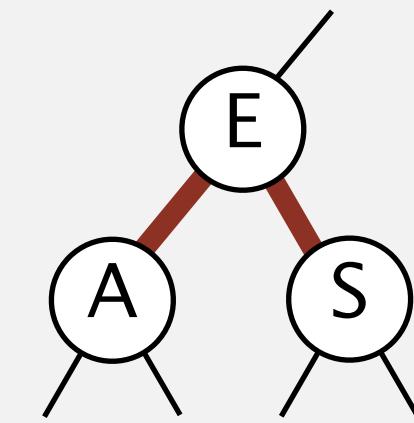
During internal operations, maintain:

- Symmetric order.
- Perfect black balance.
- [ but not necessarily color invariants ]

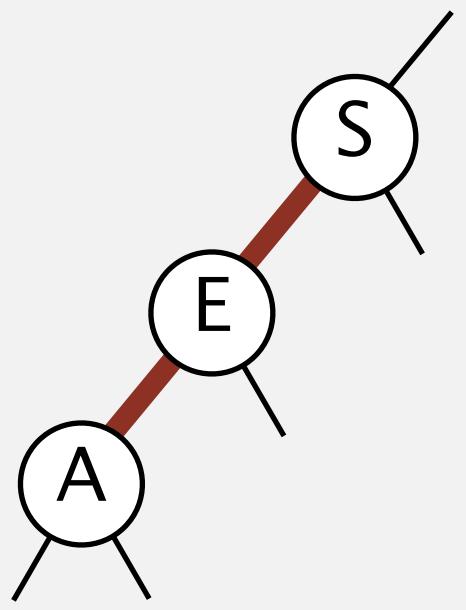
Example violations of color invariants:



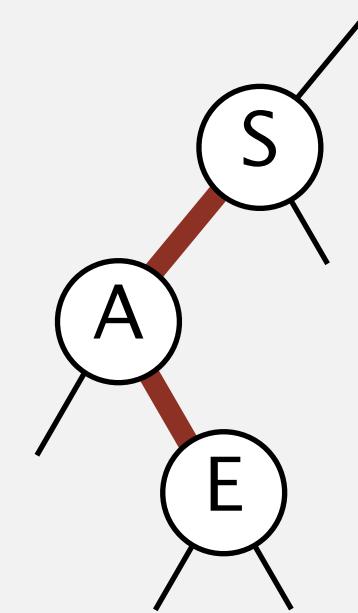
right-leaning  
red link



two red children  
(a temporary 4-node)



left-left red  
(a temporary 4-node)



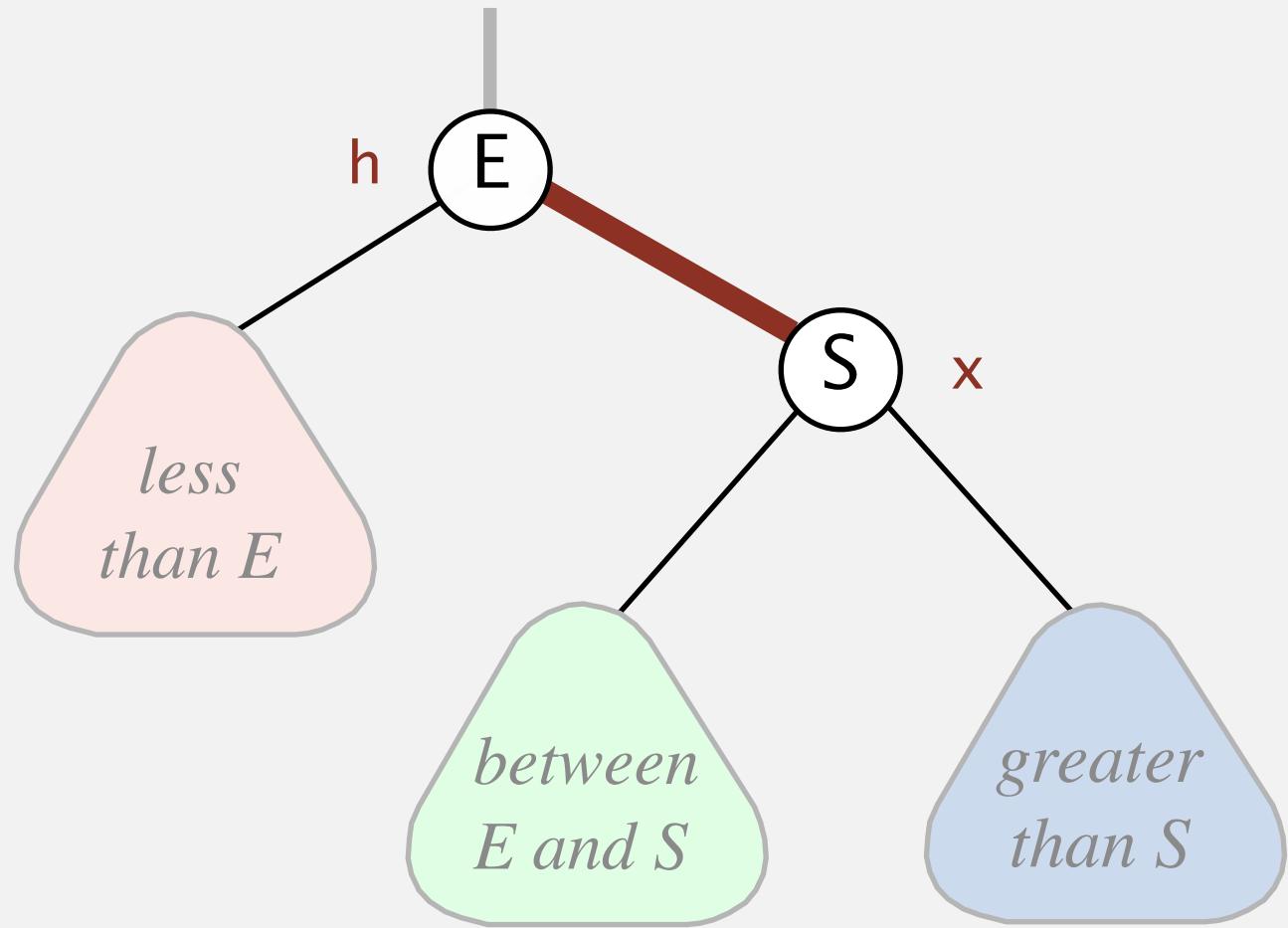
left-right red  
(a temporary 4-node)

To restore color invariants: perform rotations and color flips.

# Elementary red-black BST operations

Left rotation. Orient a (temporarily) right-leaning red link to lean left.

rotate E left  
(before)

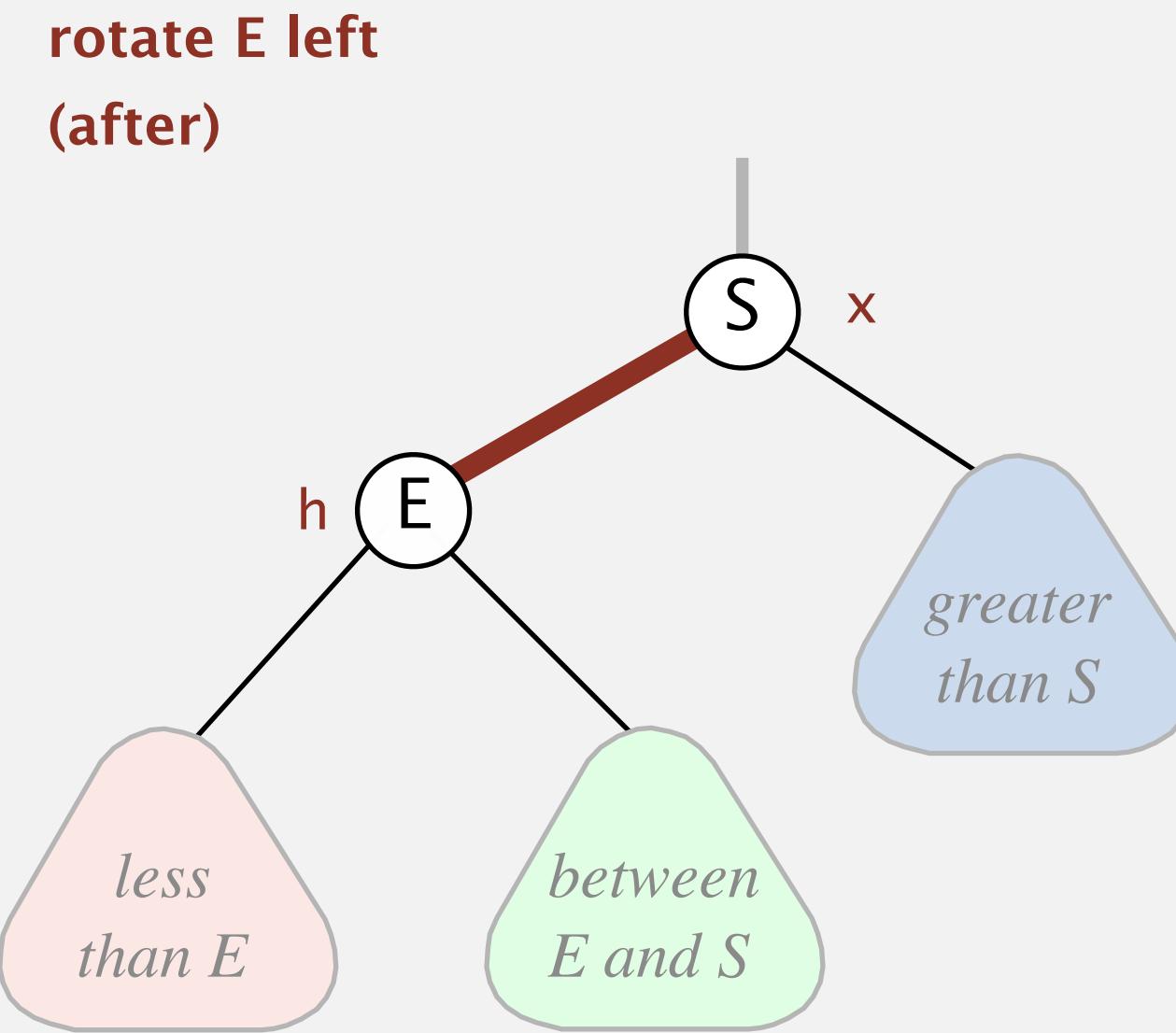


```
private Node rotateLeft(Node h)
{
    assert !isRed(h.left);
    assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

Invariants. Maintains symmetric order and perfect black balance.

# Elementary red-black BST operations

Left rotation. Orient a (temporarily) right-leaning red link to lean left.

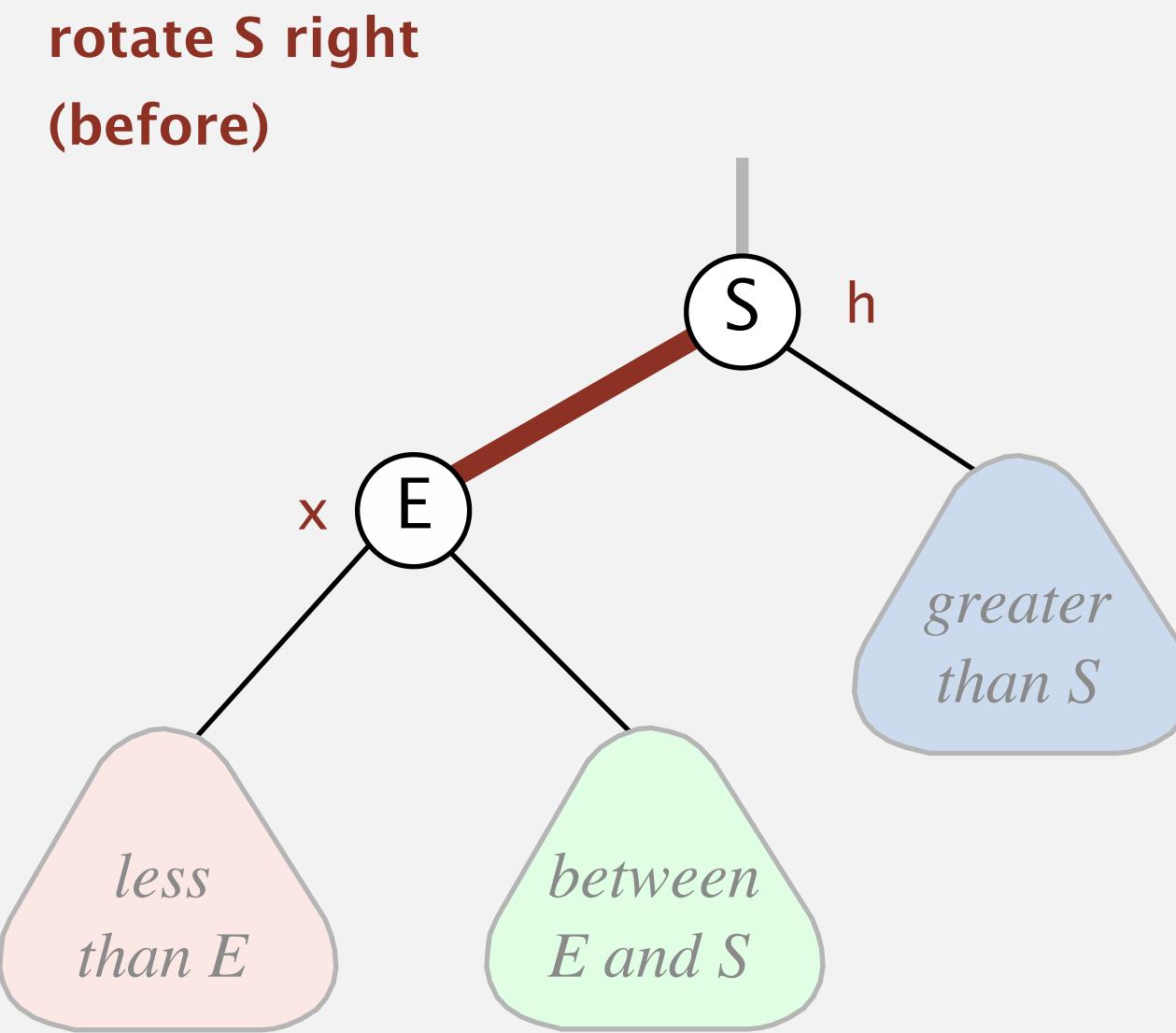


```
private Node rotateLeft(Node h)
{
    assert !isRed(h.left);
    assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

Invariants. Maintains symmetric order and perfect black balance.

# Elementary red-black BST operations

Right rotation. Orient a left-leaning red link to (temporarily) lean right.



```
private Node rotateRight(Node h)
{
    assert isRed(h.left);
    assert !isRed(h.right);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

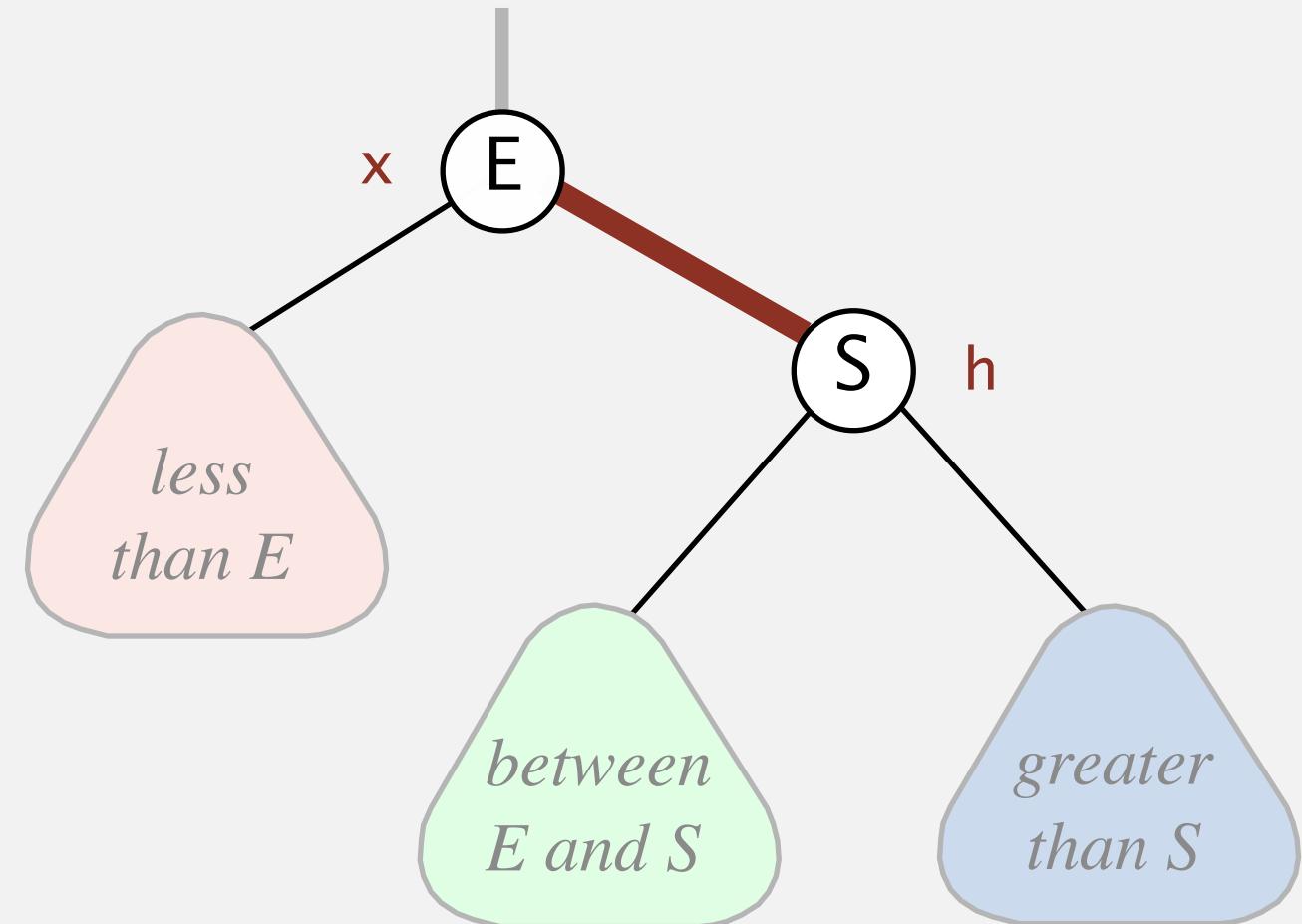
Invariants. Maintains symmetric order and perfect black balance.

# Elementary red-black BST operations

Right rotation. Orient a left-leaning red link to (temporarily) lean right.

rotate S right

(after)

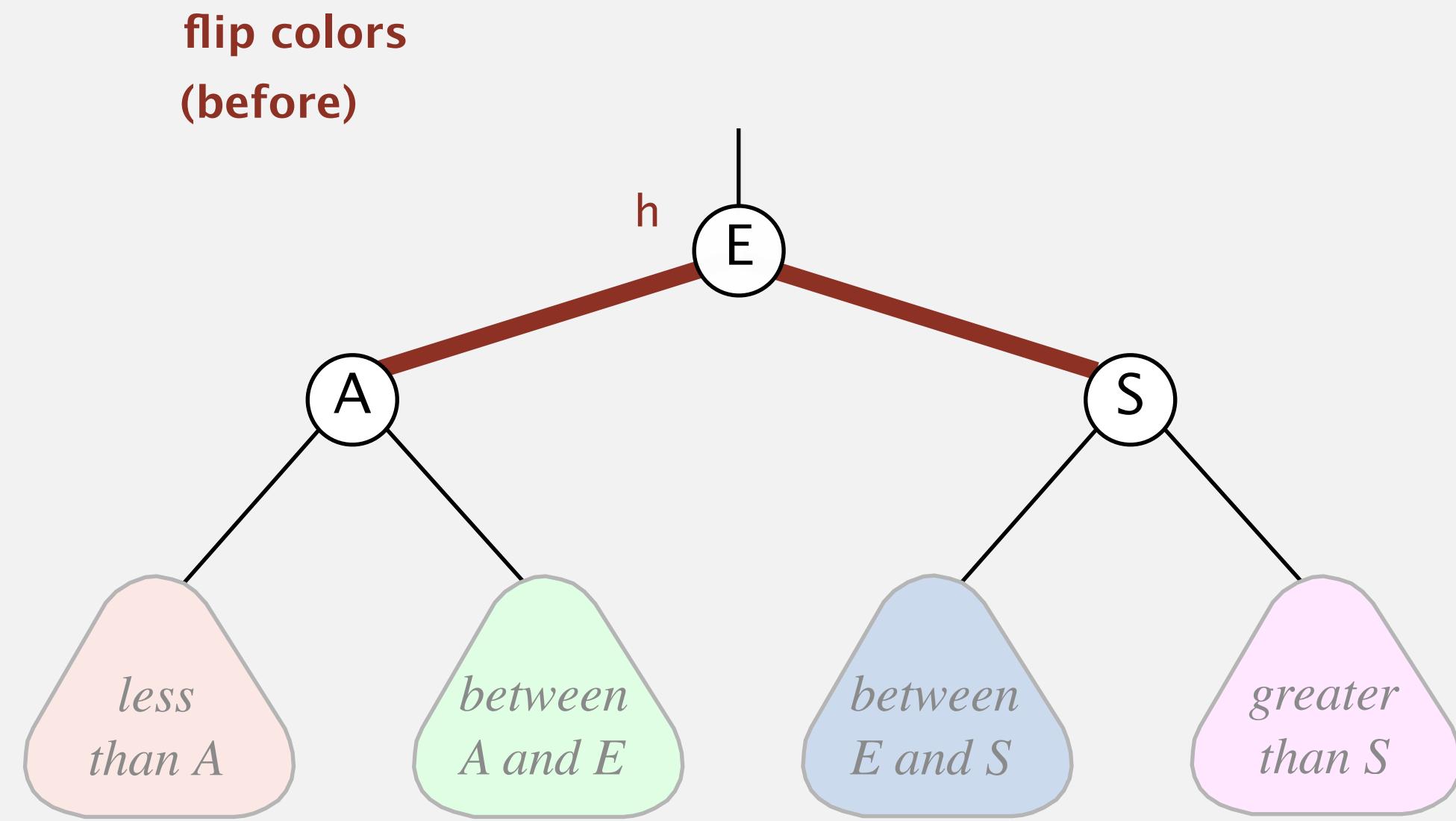


```
private Node rotateRight(Node h)
{
    assert isRed(h.left);
    assert !isRed(h.right);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

Invariants. Maintains symmetric order and perfect black balance.

# Elementary red-black BST operations

Color flip. Recolor to split a (temporary) 4-node.

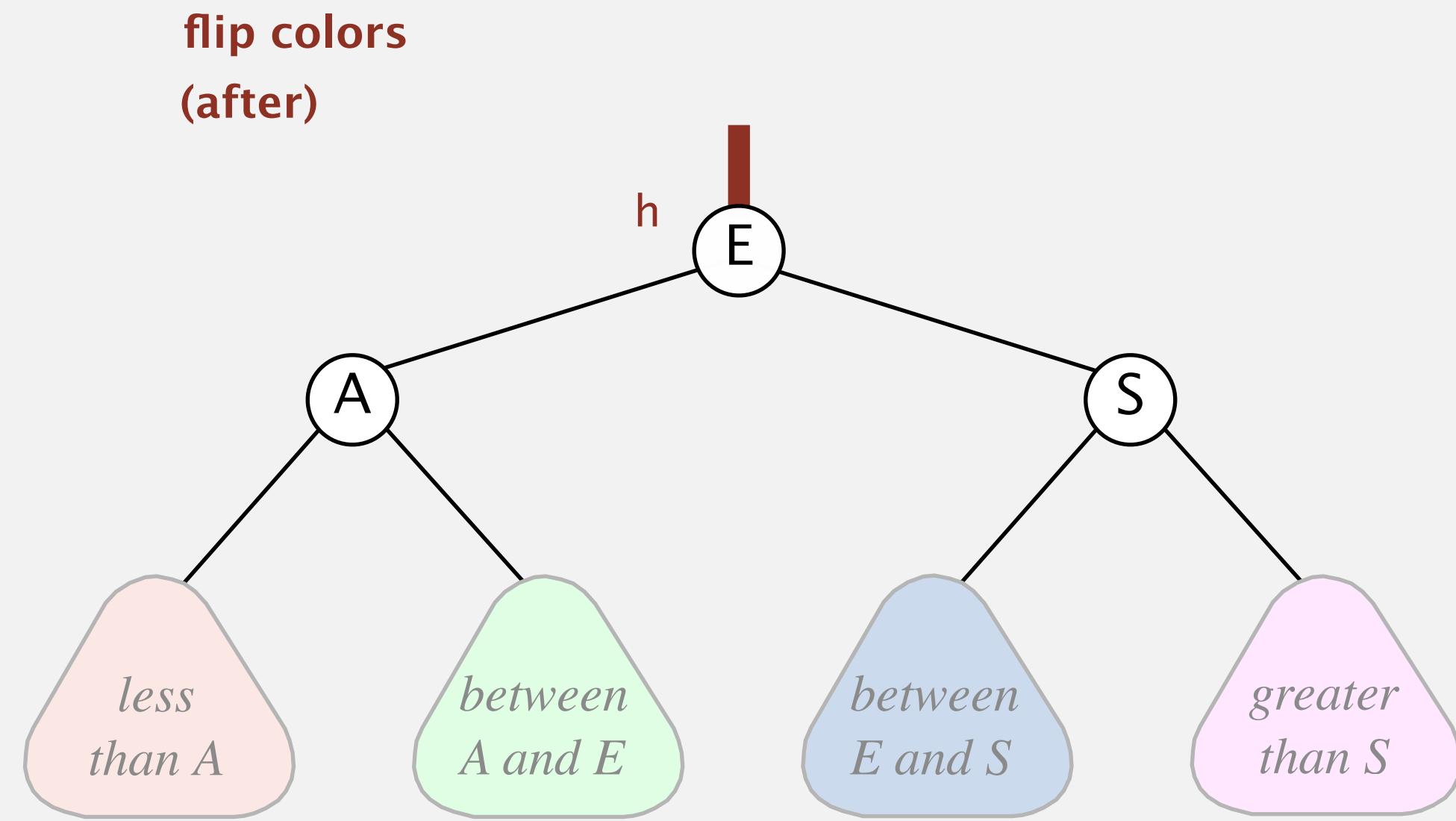


```
private void flipColors(Node h)
{
    assert !isRed(h);
    assert isRed(h.left);
    assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```

Invariants. Maintains symmetric order and perfect black balance.

# Elementary red-black BST operations

Color flip. Recolor to split a (temporary) 4-node.

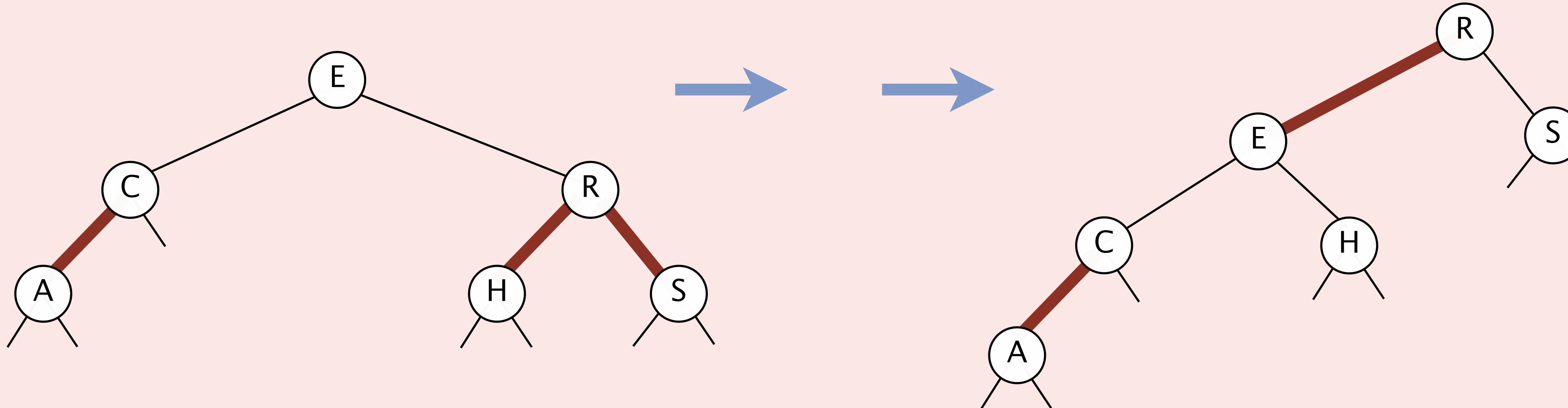


```
private void flipColors(Node h)
{
    assert !isRed(h);
    assert isRed(h.left);
    assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```

Invariants. Maintains symmetric order and perfect black balance.



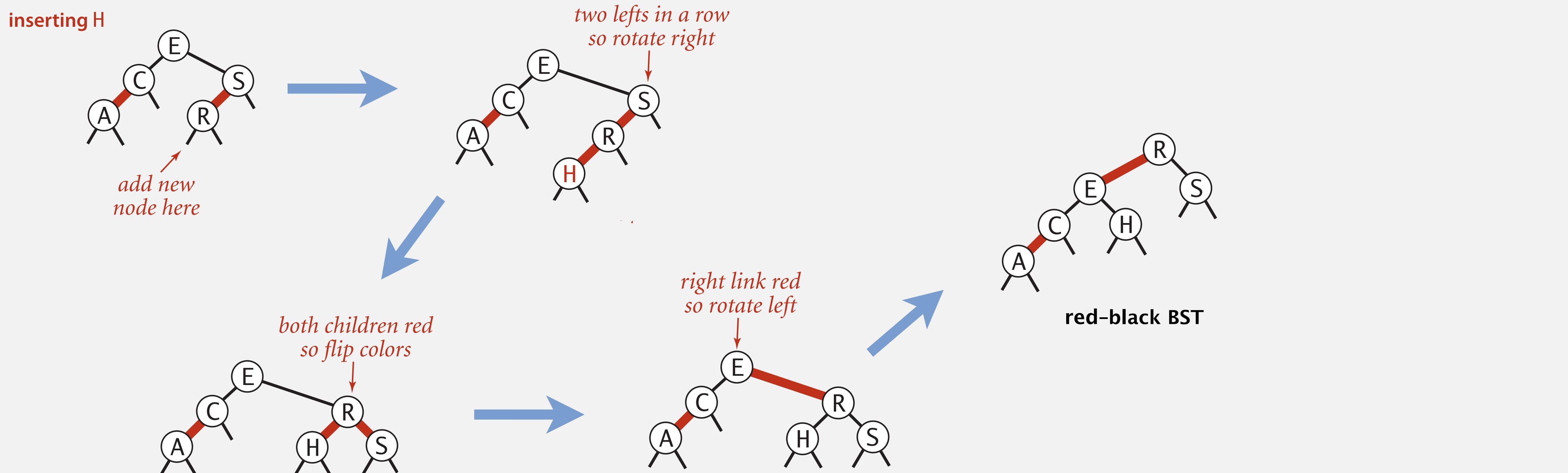
Which sequence of elementary operations transforms the red-black BST at left to the one at right?



- A. Color flip E; left rotate R.
- B. Color flip R; left rotate E.
- C. Color flip R; left rotate R.
- D. Color flip R; right rotate E.

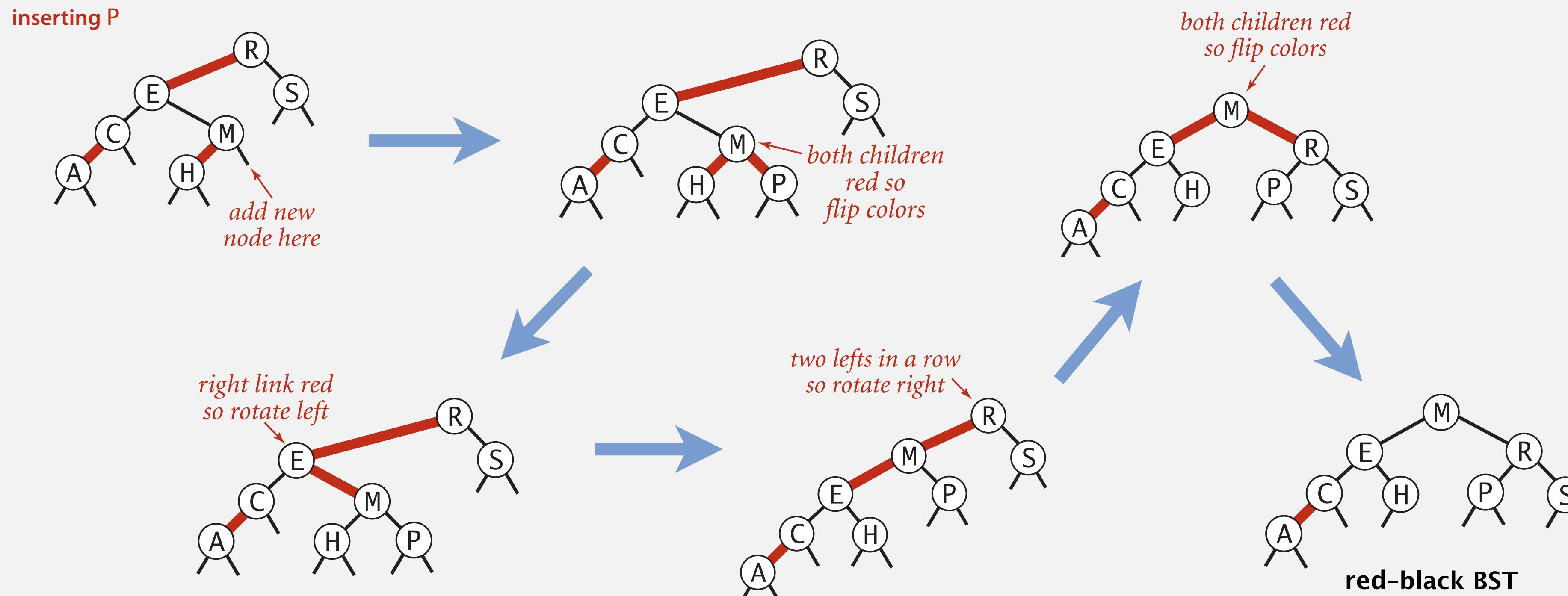
## Insertion into a LLRB tree

- Do standard BST insert and color new link red. ← to preserve symmetric order and perfect black balance
- Repeat up the tree until color invariants restored:
  - two left red links in a row? ⇒ rotate right
  - left and right links both red? ⇒ color flip
  - only right link red? ⇒ rotate left



## Insertion into a LLRB tree

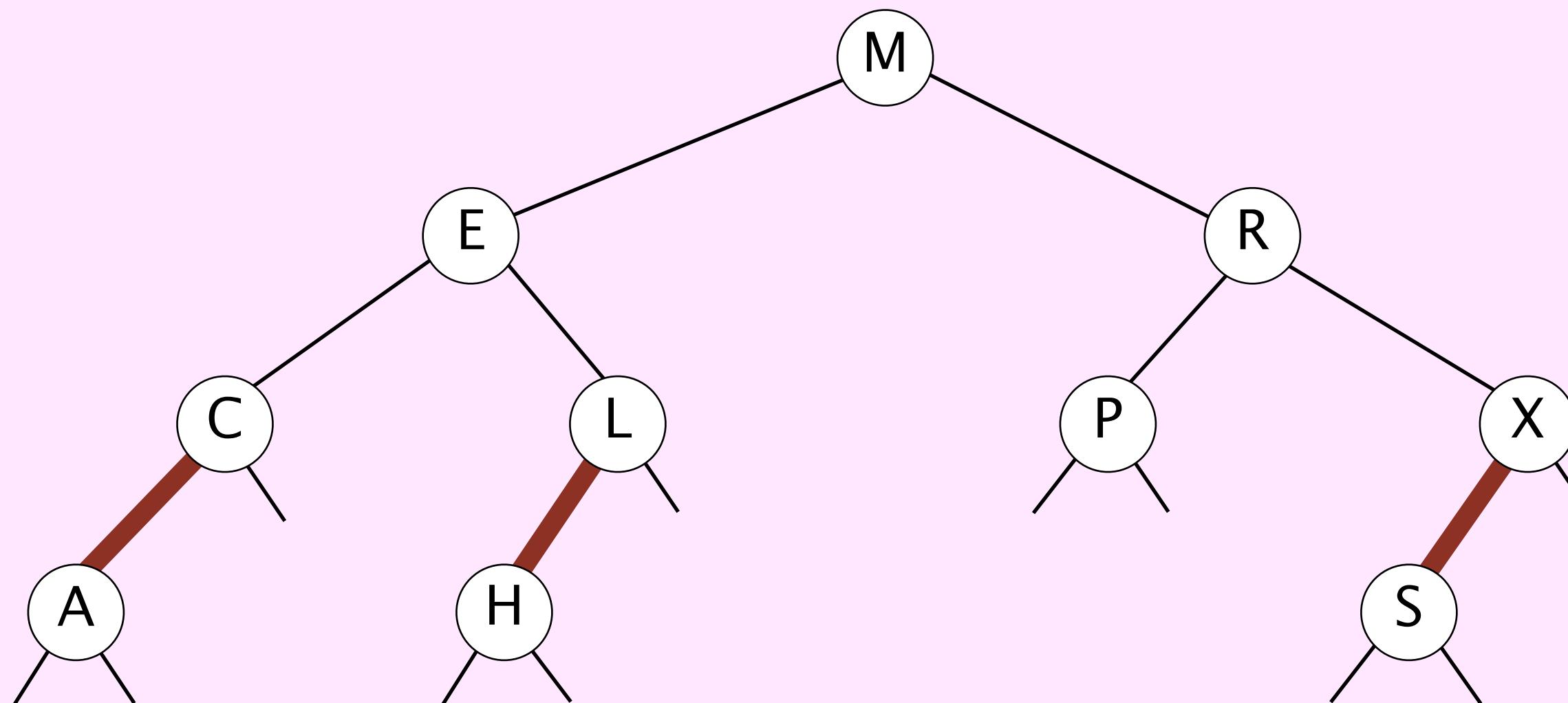
- Do standard BST insert and color new link red.
- Repeat up the tree until color invariants restored:
  - two left red links in a row?  $\Rightarrow$  rotate right
  - left and right links both red?  $\Rightarrow$  color flip
  - only right link red?  $\Rightarrow$  rotate left



# Red-black BST construction demo



insert S E A R C H X M P L



## Insertion into a LLRB tree: Java implementation

- Do standard BST insert and color new link red.
- Repeat up the tree until color invariants restored:
  - only right link red?  $\Rightarrow$  rotate left
  - two left red links in a row?  $\Rightarrow$  rotate right
  - left and right links both red?  $\Rightarrow$  color flip

```
private Node put(Node h, Key key, Value val)
{
    if (h == null) return new Node(key, val, RED); ← insert at bottom
    if (h == null) return new Node(key, val, RED); ← (and color it red)

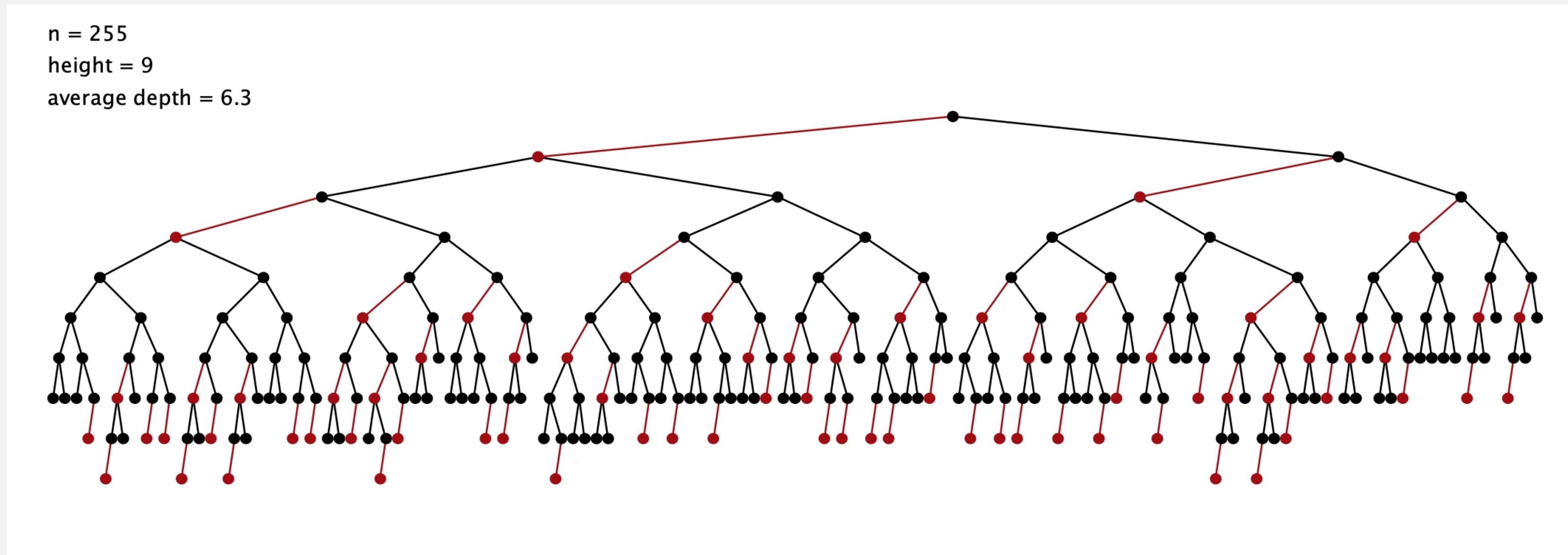
    int cmp = key.compareTo(h.key);
    if (cmp < 0) h.left = put(h.left, key, val);
    else if (cmp > 0) h.right = put(h.right, key, val);
    else h.val = val;

    if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
    if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
    if (isRed(h.left) && isRed(h.right)) flipColors(h); ← restore color
    invariants

    return h; ← only a few extra lines of code provides near-perfect balance
}
```

## Insertion into a LLRB tree: visualization

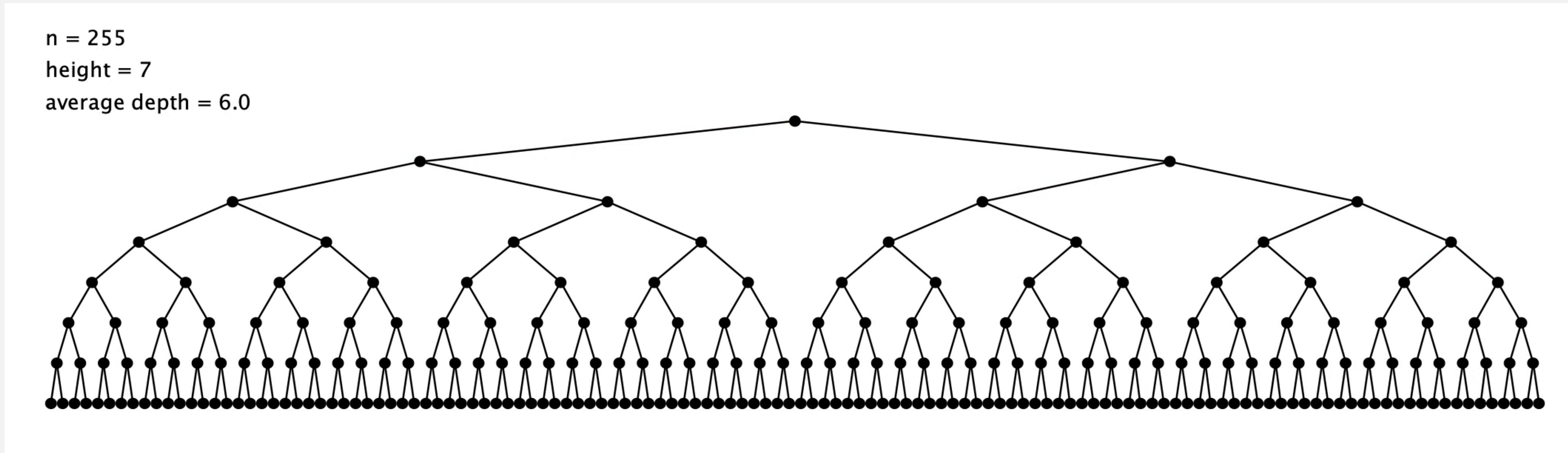
---



255 insertions in random order

## Insertion into a LLRB tree: visualization

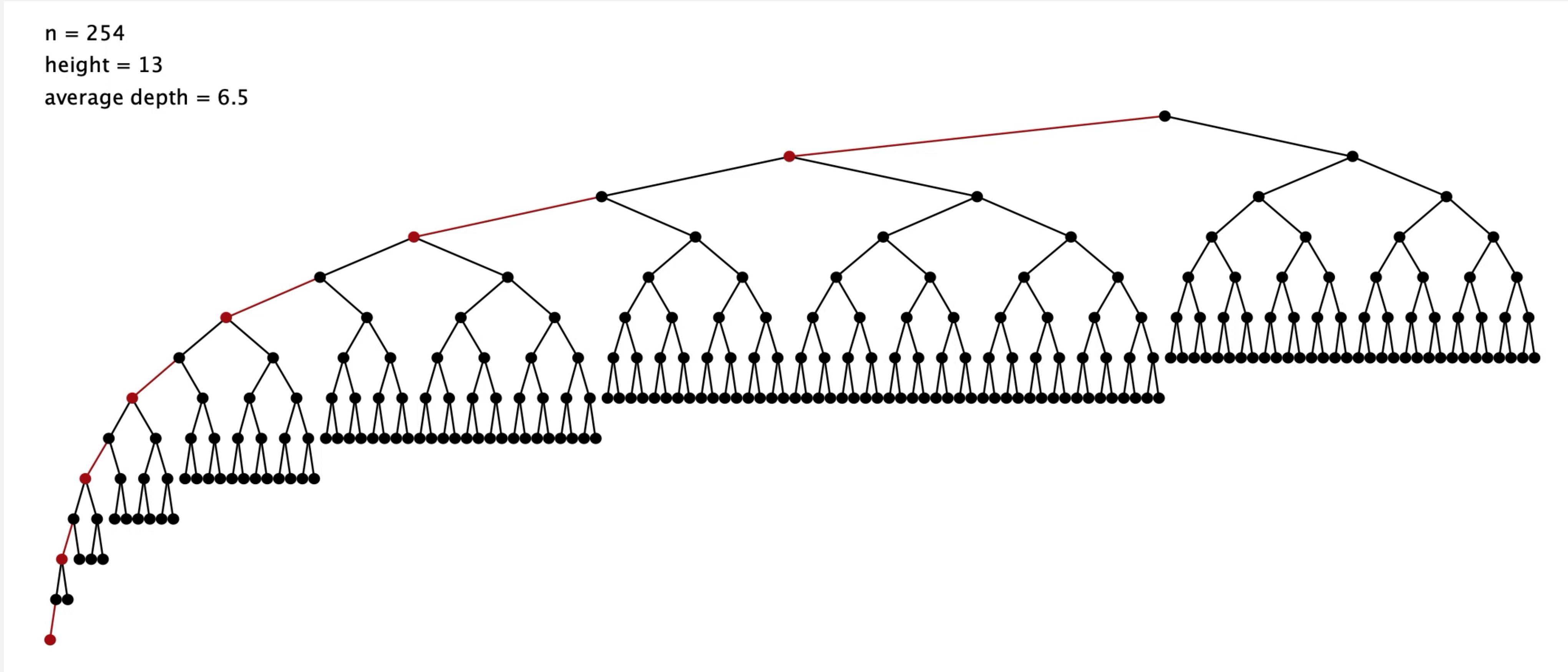
---



255 insertions in ascending order

## Insertion into a LLRB tree: visualization

---



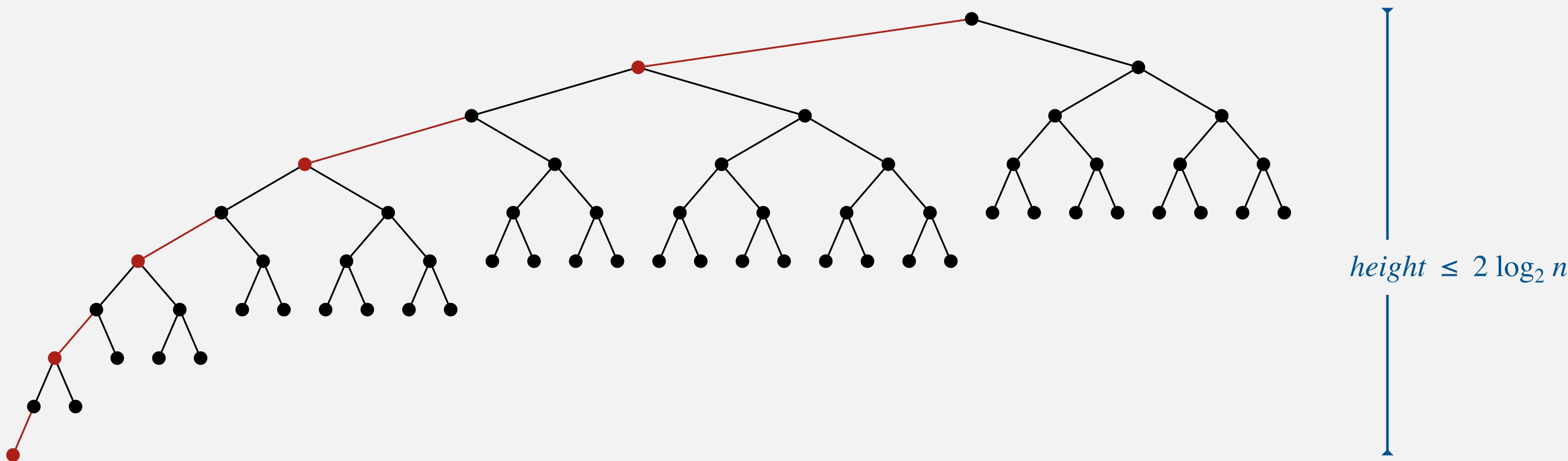
254 insertions in descending order

## Balance in LLRB trees

Proposition. Height of LLRB tree is  $\leq 2 \log_2 n$ .

Pf.

- Black height = height of corresponding 2–3 tree  $\leq \log_2 n$ .
- Never two red links in a row.  
 $\Rightarrow$  height of LLRB tree  $\leq (2 \times \text{black height}) + 1$   
 $\leq 2 \log_2 n + 1$ .
- [ A slightly more careful argument shows height  $\leq 2 \log_2 n$ . ]



# ST implementations: summary

implementation	guarantee			ordered ops?	key interface
	search	insert	delete		
sequential search (unordered list)	$n$	$n$	$n$		<code>equals()</code>
binary search (sorted array)	$\log n$	$n$	$n$	✓	<code>compareTo()</code>
BST	$n$	$n$	$n$	✓	<code>compareTo()</code>
2-3 trees	$\log n$	$\log n$	$\log n$	✓	<code>compareTo()</code>
red-black BSTs	$\log n$	$\log n$	$\log n$	✓	<code>compareTo()</code>

hidden constant  $c$  is small  
( $\leq 2 \log_2 n$  compares)



## 3.3 BALANCED SEARCH TREES

---

- ▶ *2–3 search trees*
- ▶ *red–black BSTs (representation)*
- ▶ *red–black BSTs (operations)*
- ▶ ***context***

## Balanced search trees in the wild

---

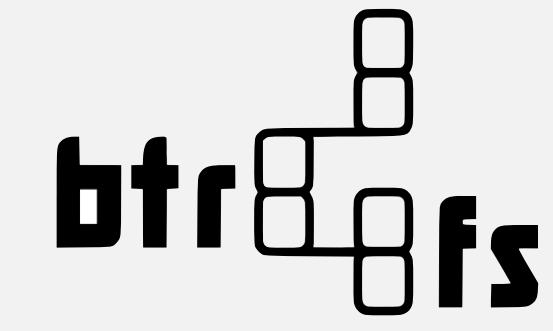
Red-black BSTs are widely used as system symbol tables.

- Java: `java.util.TreeMap`, `java.util.TreeSet`.
- C++ STL: `map`, `multimap`, `multiset`.
- Linux kernel: CFQ I/O scheduler, VMAs, `linux/rbtree.h`.

Other balanced BSTs. AVL trees, splay trees, randomized BSTs, rank-balanced BSTs, ....

B-trees (and cousins) are widely used for file systems and databases.

- Windows: NTFS.
- Mac OS X: HFS, HFS+, APFS.
- Linux: ReiserFS, XFS, ext4, JFS, Btrfs.
- Databases: Oracle, DB2, Ingres, SQL, PostgreSQL.



## War story 1: red-black BSTs

---

Telephone company contracted with database provider to build real-time database to store customer information.

### Database implementation.

- Red-black BST.
- Exceeding height limit of 80 triggered error-recovery process.

should support up to  $2^{40}$  keys

### Extended telephone service outage.

- Main cause = height bound exceeded!
- Telephone company sues database provider.
- Legal testimony:

*“If implemented properly, the height of a red-black BST with  $n$  keys is at most  $2 \log_2 n$ . ” — expert witness*



## War story 2: red-black BSTs

 **Celestine Omin**   
@cyberomin

[Follow](#) ▾

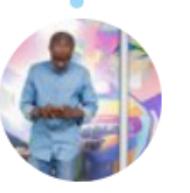
I was just asked to balance a Binary Search Tree by JFK's airport immigration. Welcome to America.

8:26 AM - 26 Feb 2017 from [Manhattan, NY](#)

---

8,025 Retweets 7,087 Likes



 **Celestine Omin**  @cyberomin · 26 Feb 2017  
I was too tired to even think of a BST solution. I have been travelling for 23hrs. But I was also asked about 10 CS questions.  
8 164 244

 **Celestine Omin**  @cyberomin · 26 Feb 2017  
sad thing is, if I didn't give the Wikipedia definition for these questions, it was considered a wrong answer.  
19 324 703

---

 **Simon Sharwood** @ssharwood · 26 Feb 2017  
Replies to @cyberomin  
seriously? am reporter for [@theresister](#) and would love to know more about your experience  
2 22 171



<https://twitter.com/cyberomin/status/835888786462625792>

# The red-black tree song (by Sean Sandys)

© Copyright 2021 Robert Sedgewick and Kevin Wayne