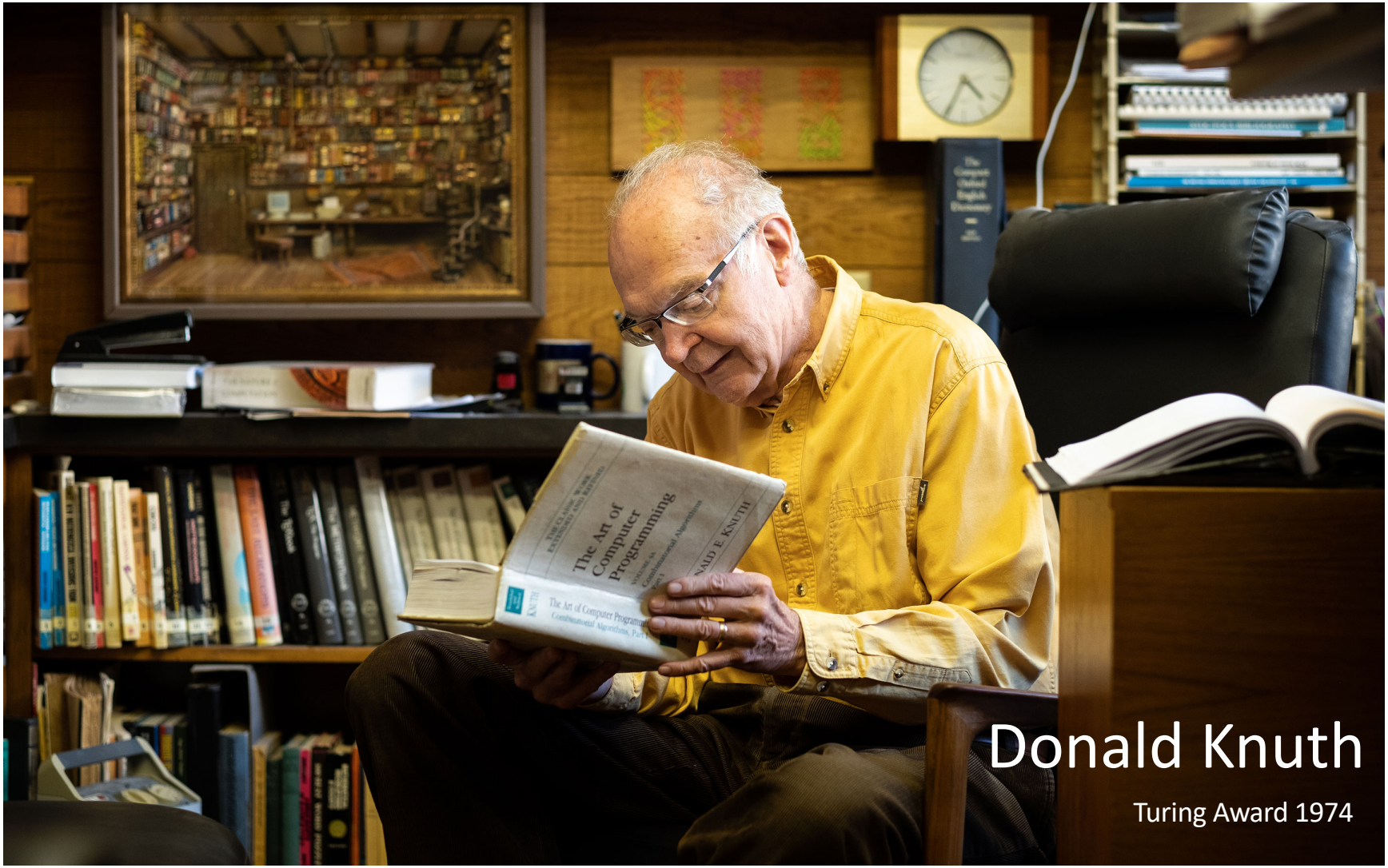
CSCI 1102 Computer Science 2

Meeting 20: Thursday 4/8/2021

More on BSTs

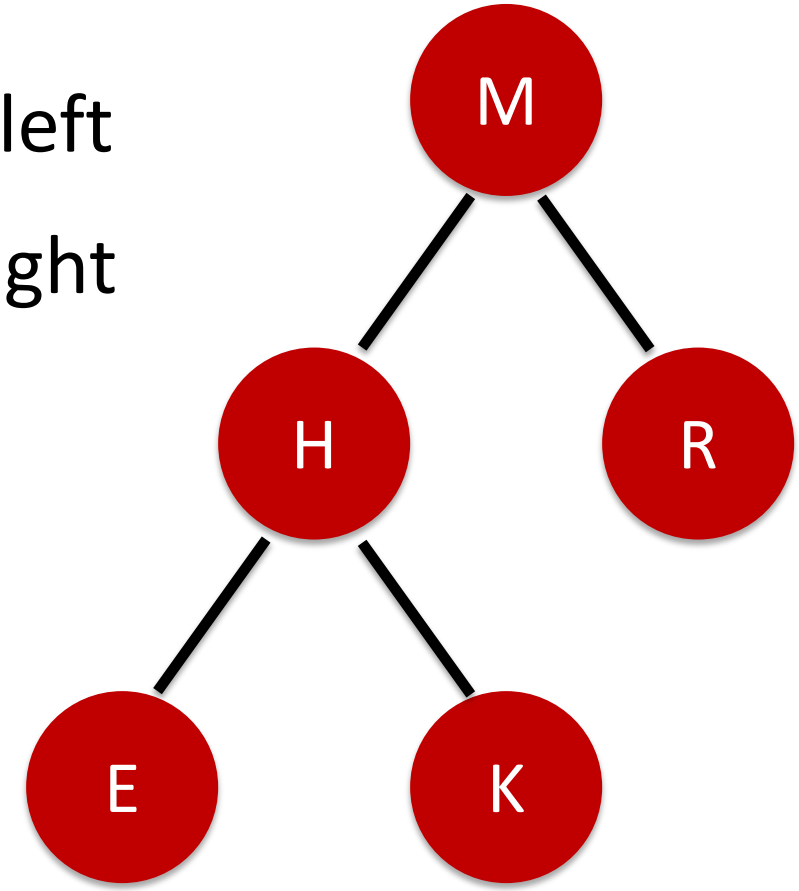


Donald Knuth

Turing Award 1974

Binary Search Trees

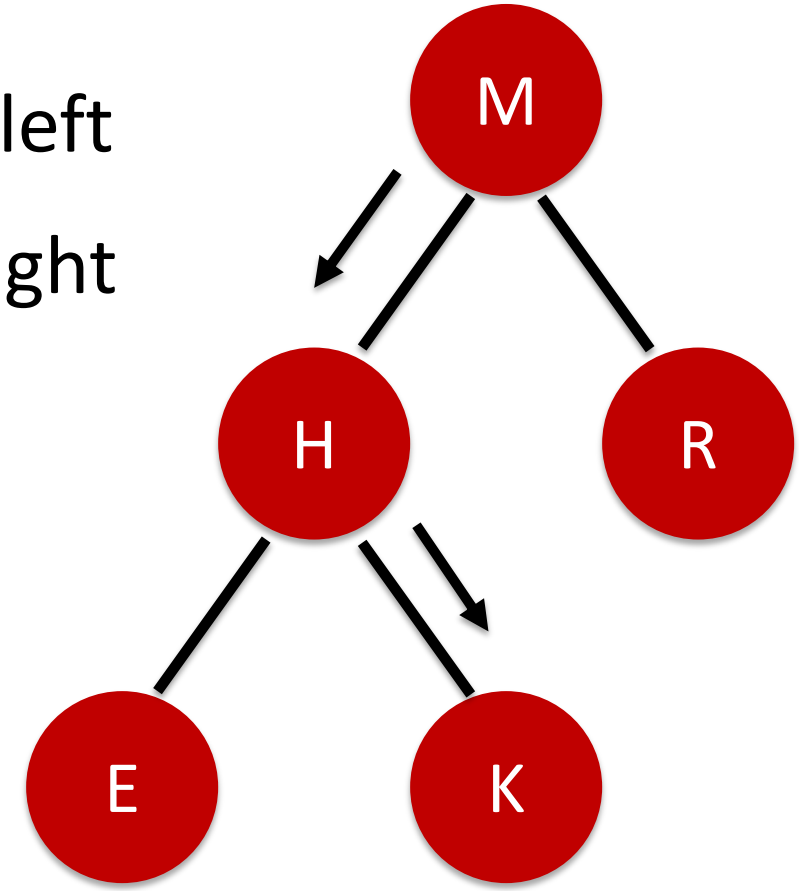
- Smaller keys to the left
- Larger keys to the right



Binary Search Trees

- Smaller keys to the left
- Larger keys to the right

bst.get(K)

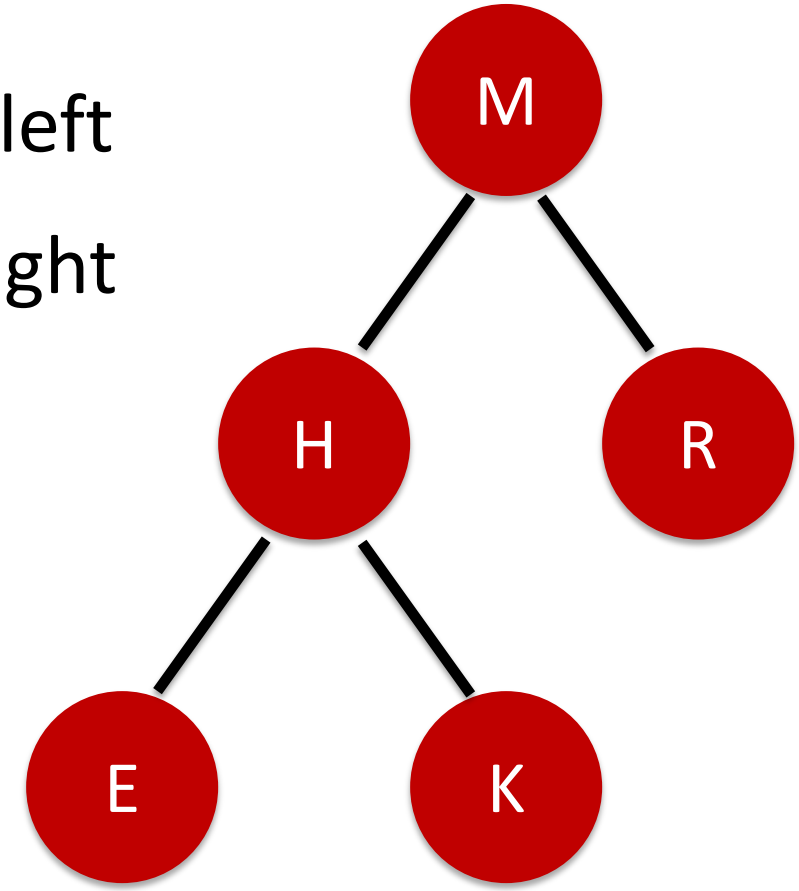


Operations on BSTs must Preserve the Invariants

Binary Search Trees

- Smaller keys to the left
- Larger keys to the right

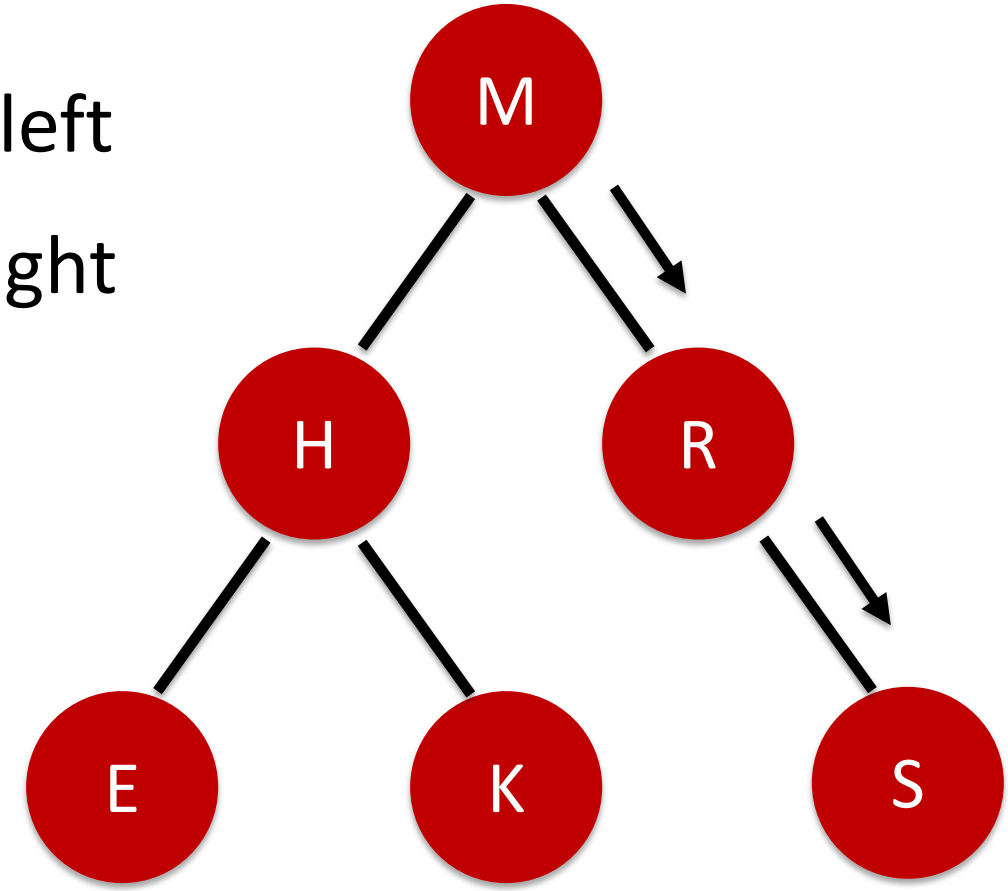
bst.put(S, 3)



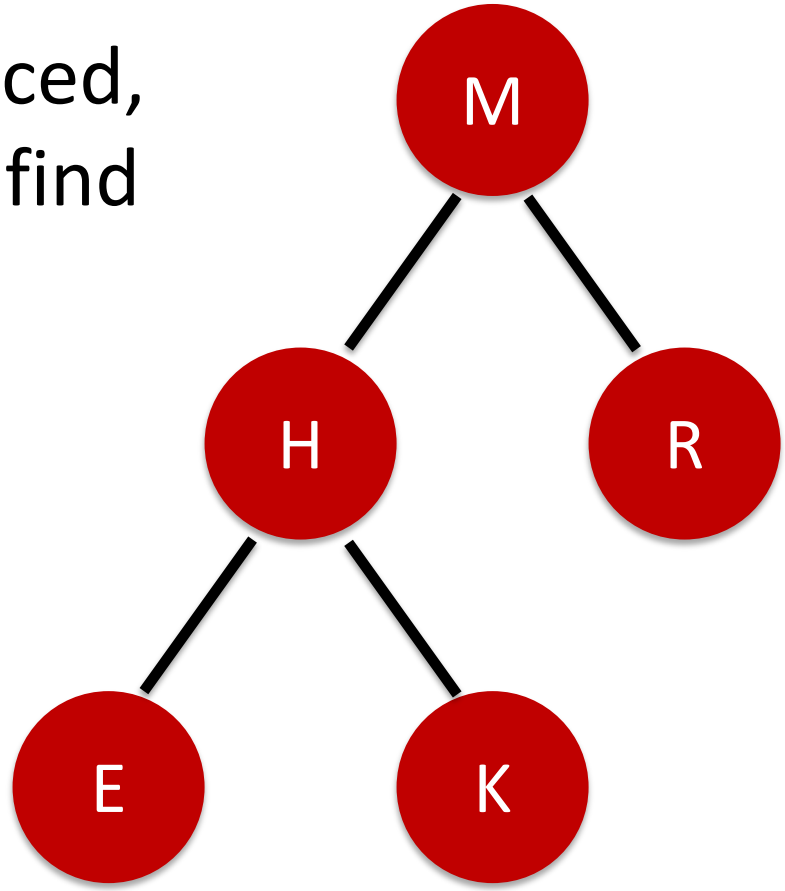
Binary Search Trees

- Smaller keys to the left
- Larger keys to the right

bst.put(S, 3)

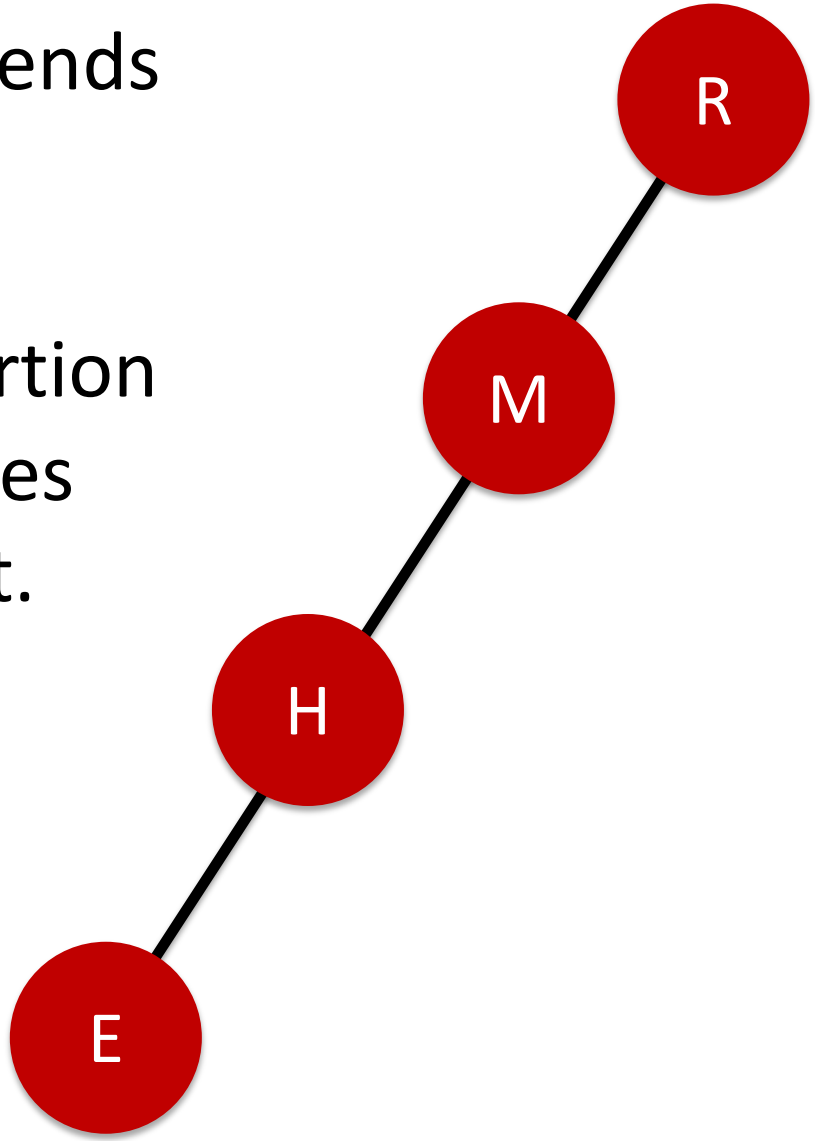


If the tree is balanced, $\log_2$ insertion and find time.

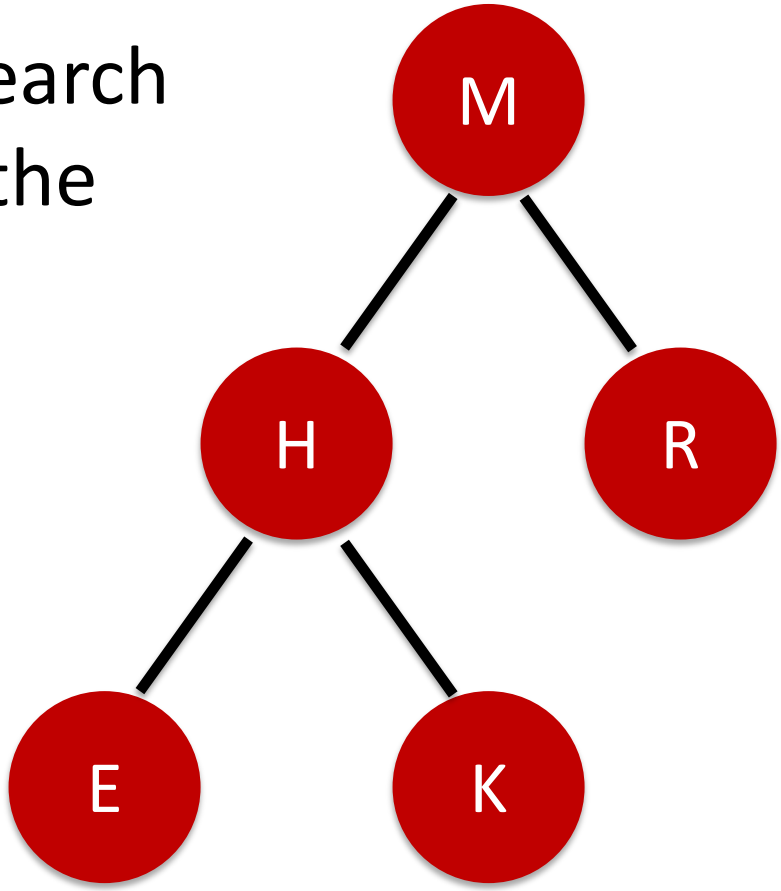


Structure of tree depends on insertion order.

With an unlucky insertion order, the tree behaves like a singly-linked list.

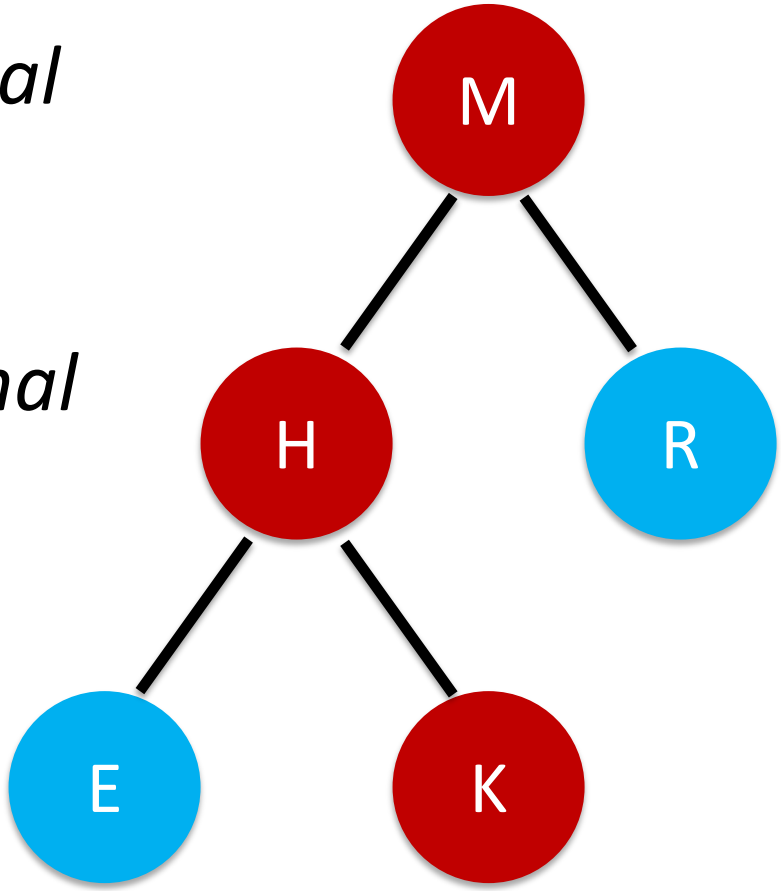


Balanced binary search trees ensure that the tree remains well-balanced.



Finding the *minimal* key E – go left.

Finding the *maximal* key R – go right.

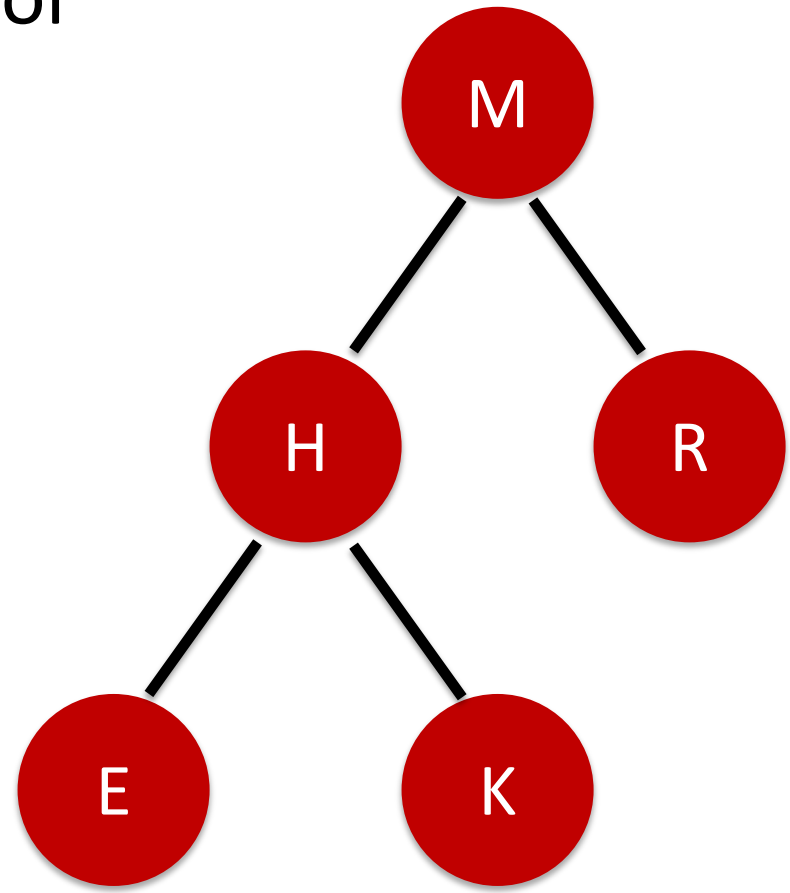


The *floor* of key is the greatest key less than or equal to key.

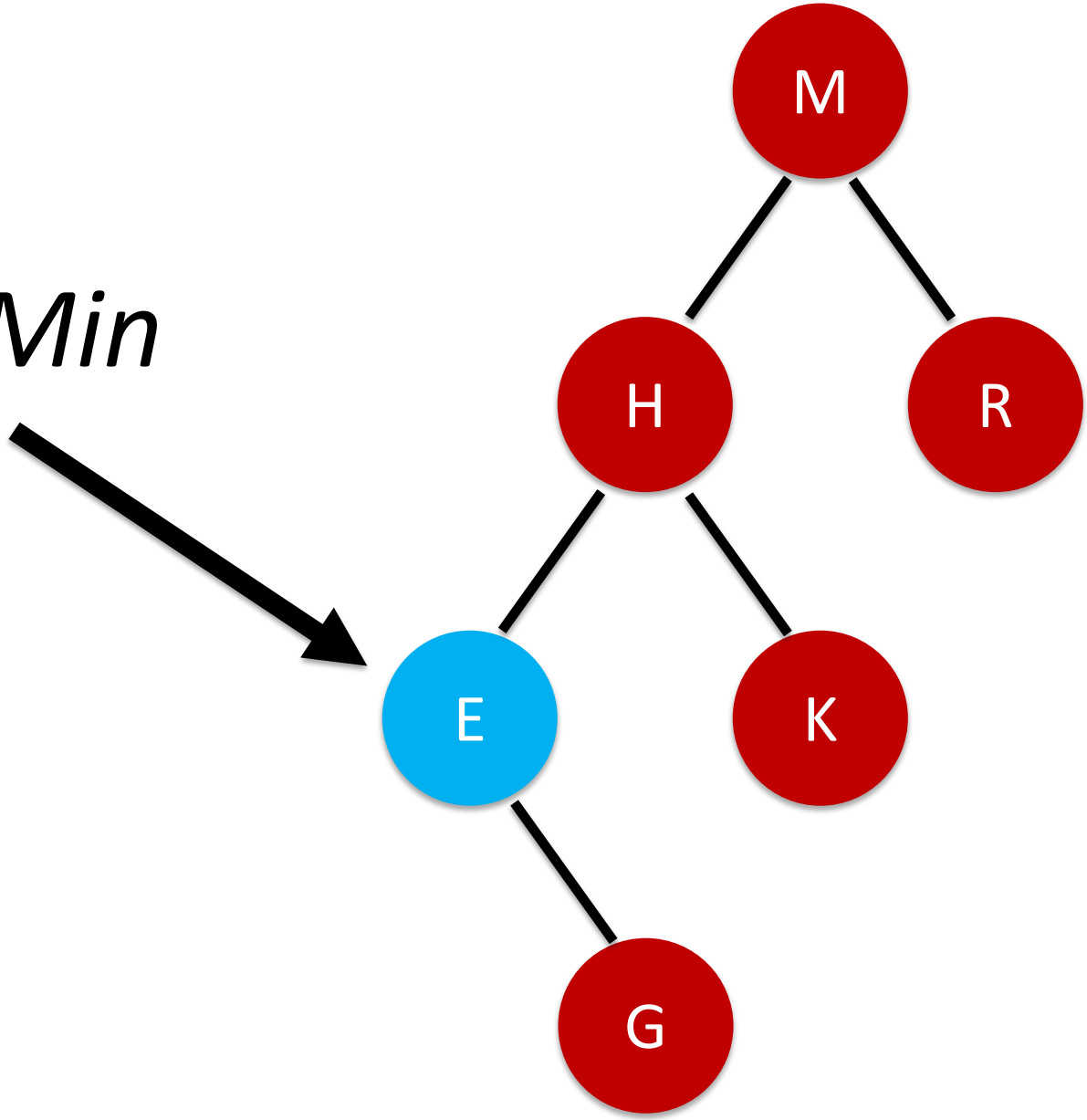
The floor of Z is R.

The floor of H is H.

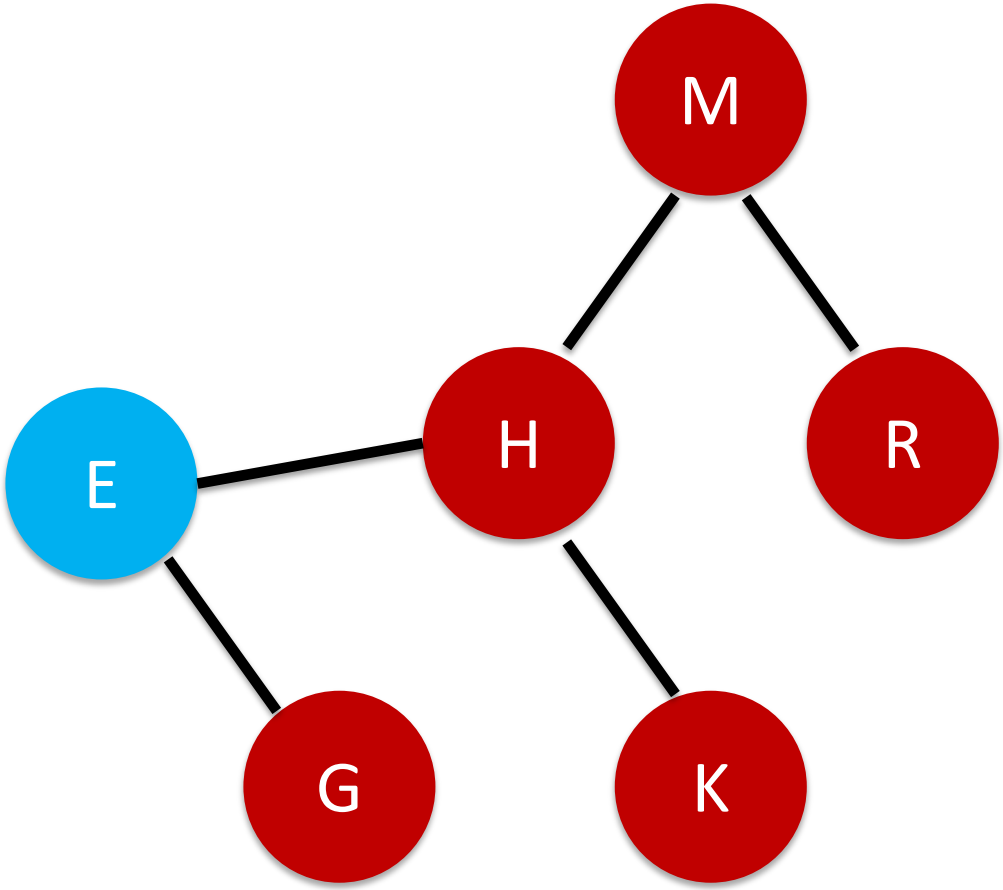
The floor of B doesn't exist.



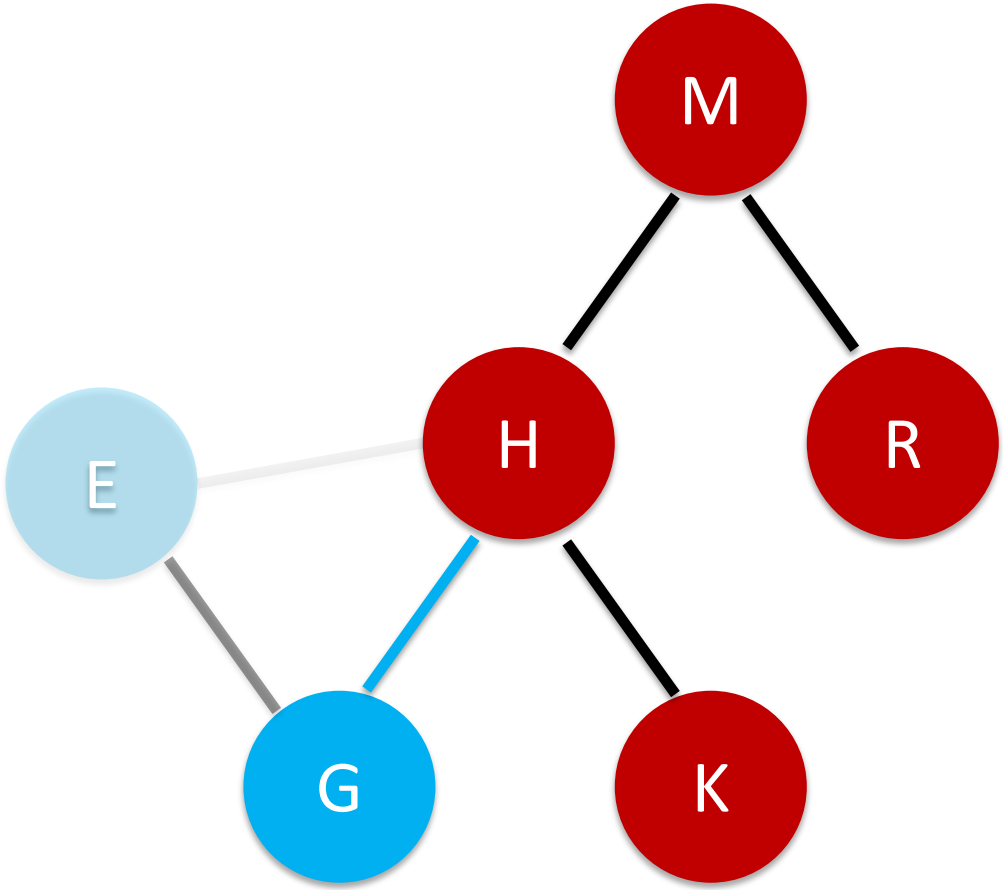
deleteMin



deleteMin

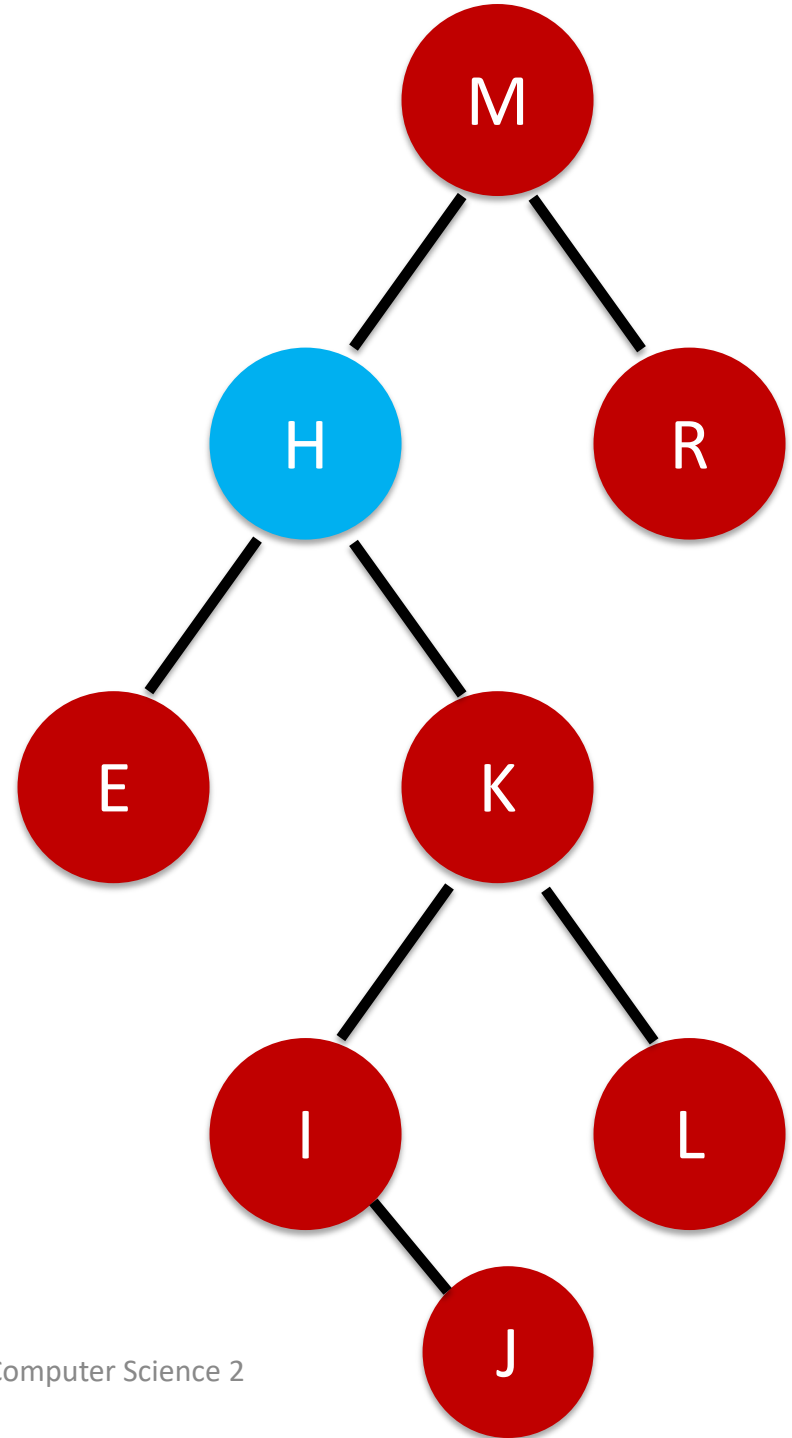


deleteMin



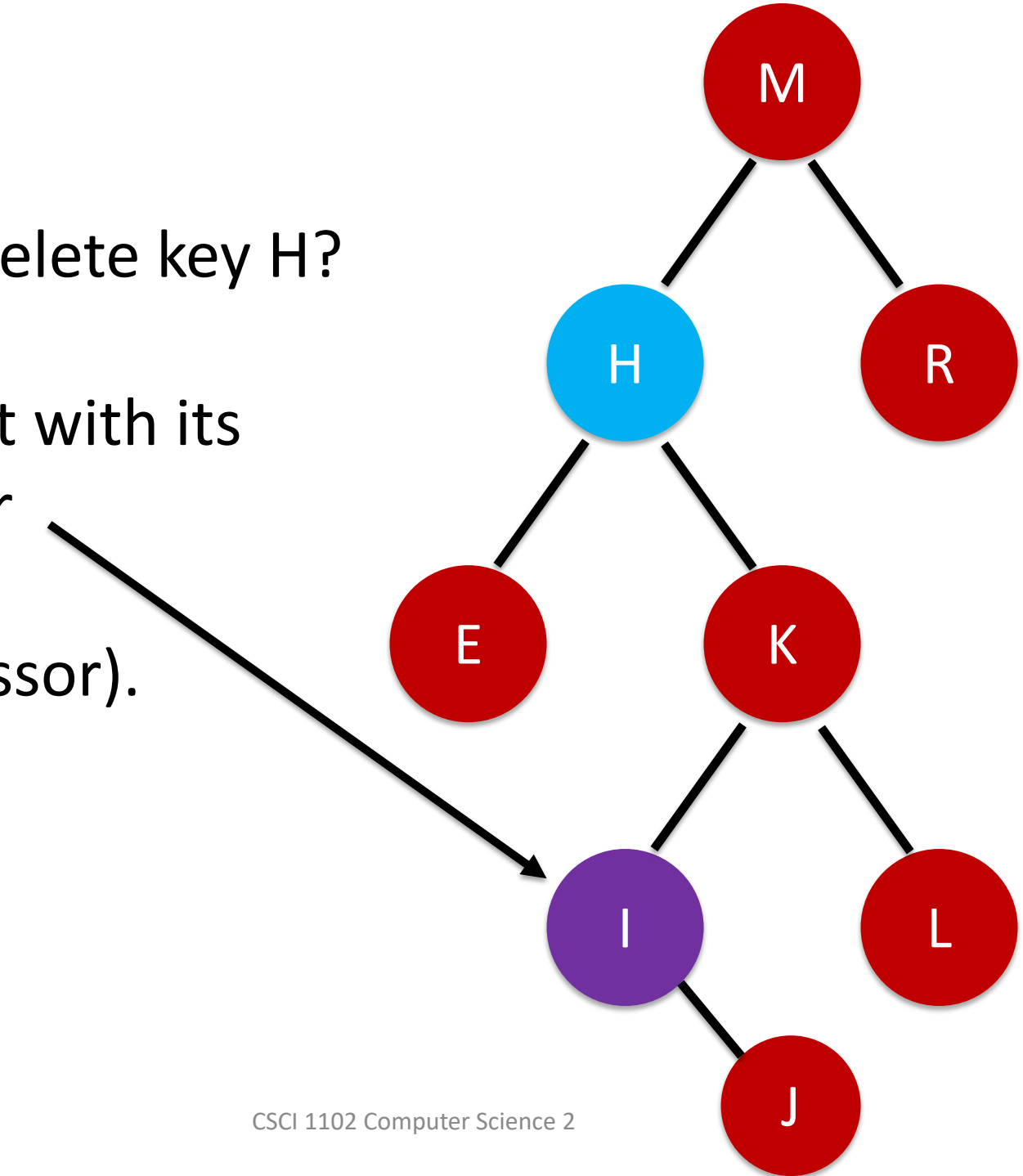
```java
public void deleteMin() {
  if (isEmpty())
    throw new NoSuchElementException("deleteMin: underflow");
  root = deleteMin(root);
}

private Node deleteMin(Node x) {
  if (x.left == null) return x.right;
  x.left = deleteMin(x.left);
  x.size = size(x.left) + size(x.right) + 1;
  return x;
}
```

How to delete key H?

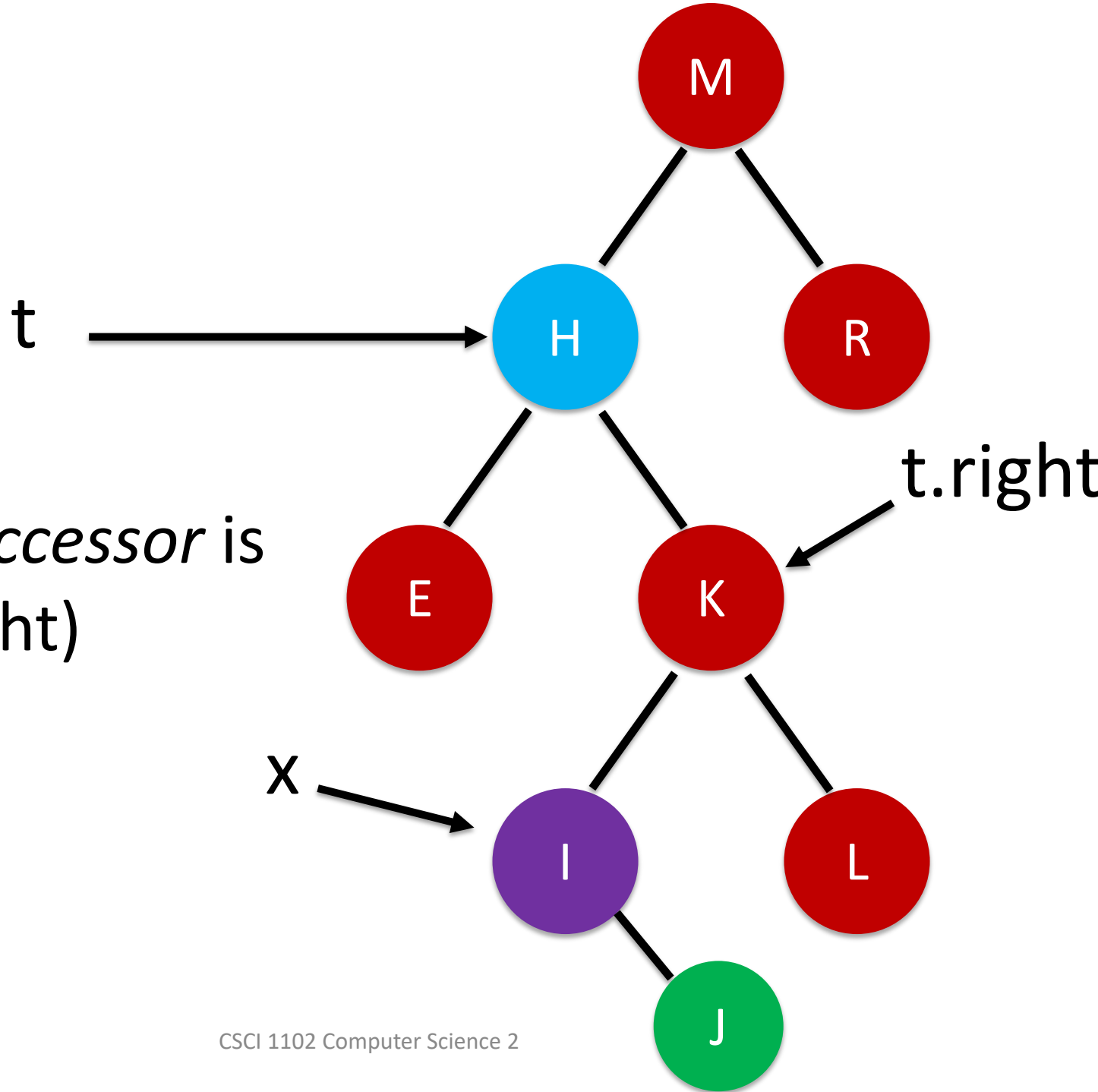


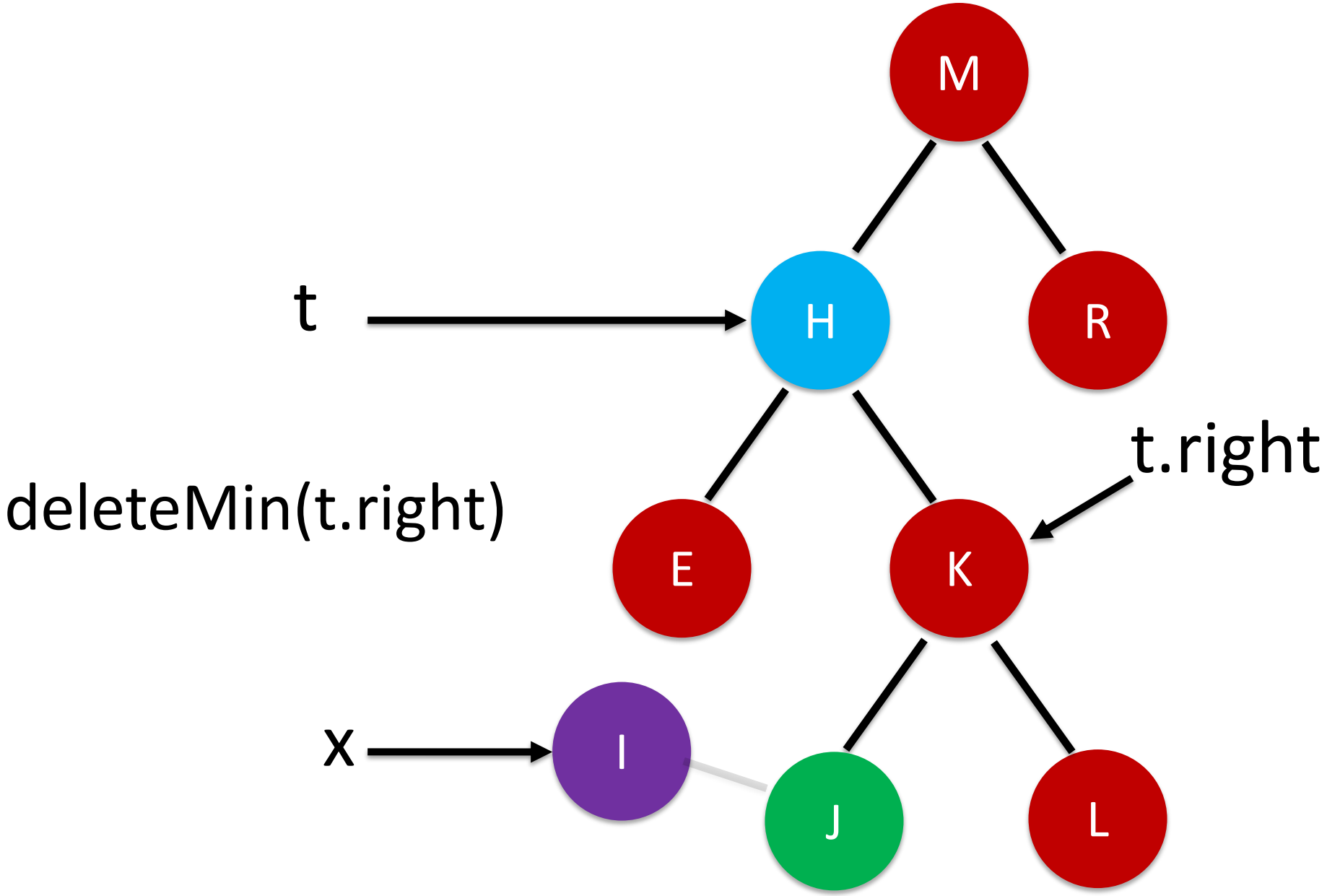
How to delete key H?

Replace it with its successor (or predecessor).

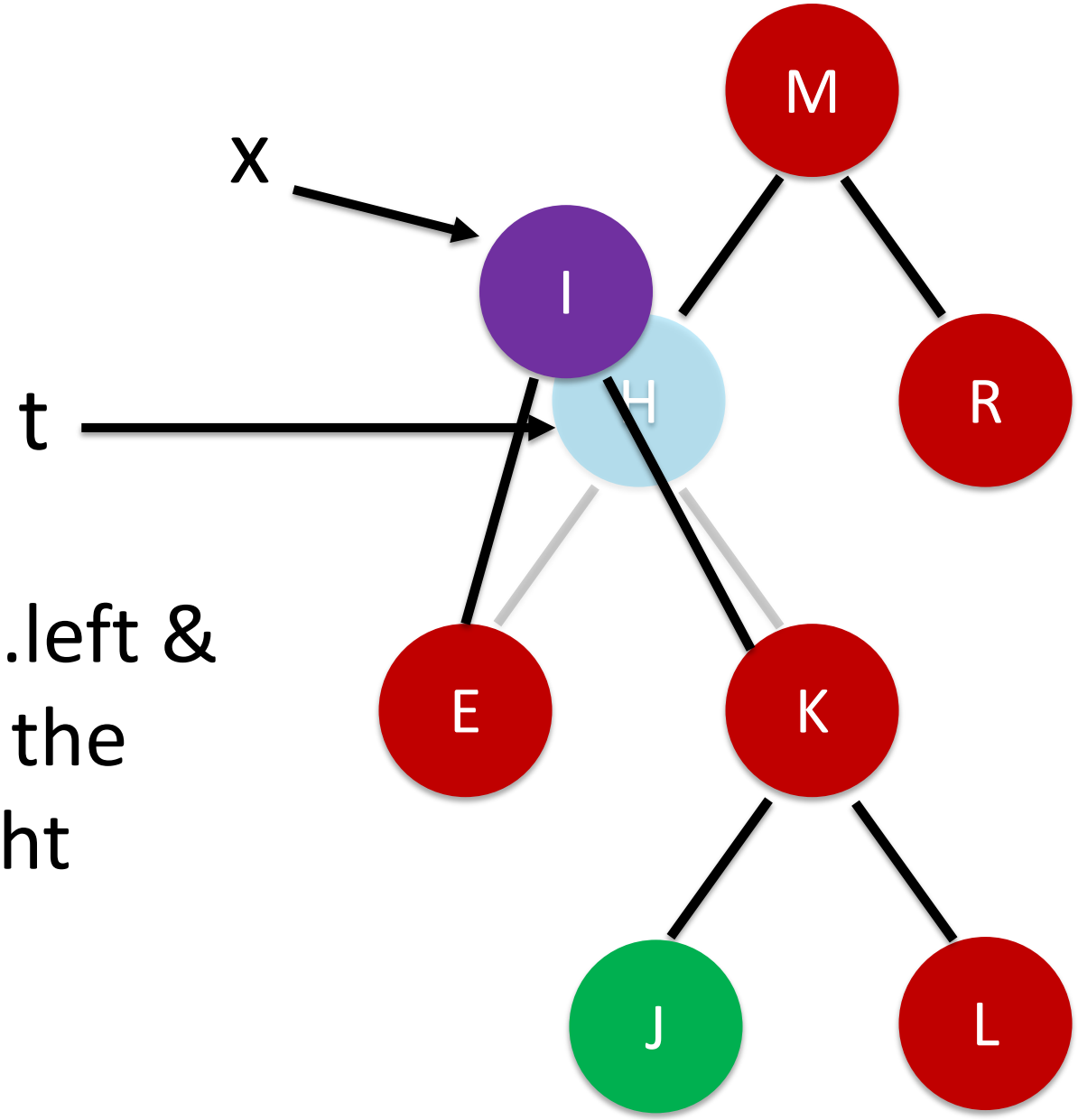


x, the *successor* is min(t.right)

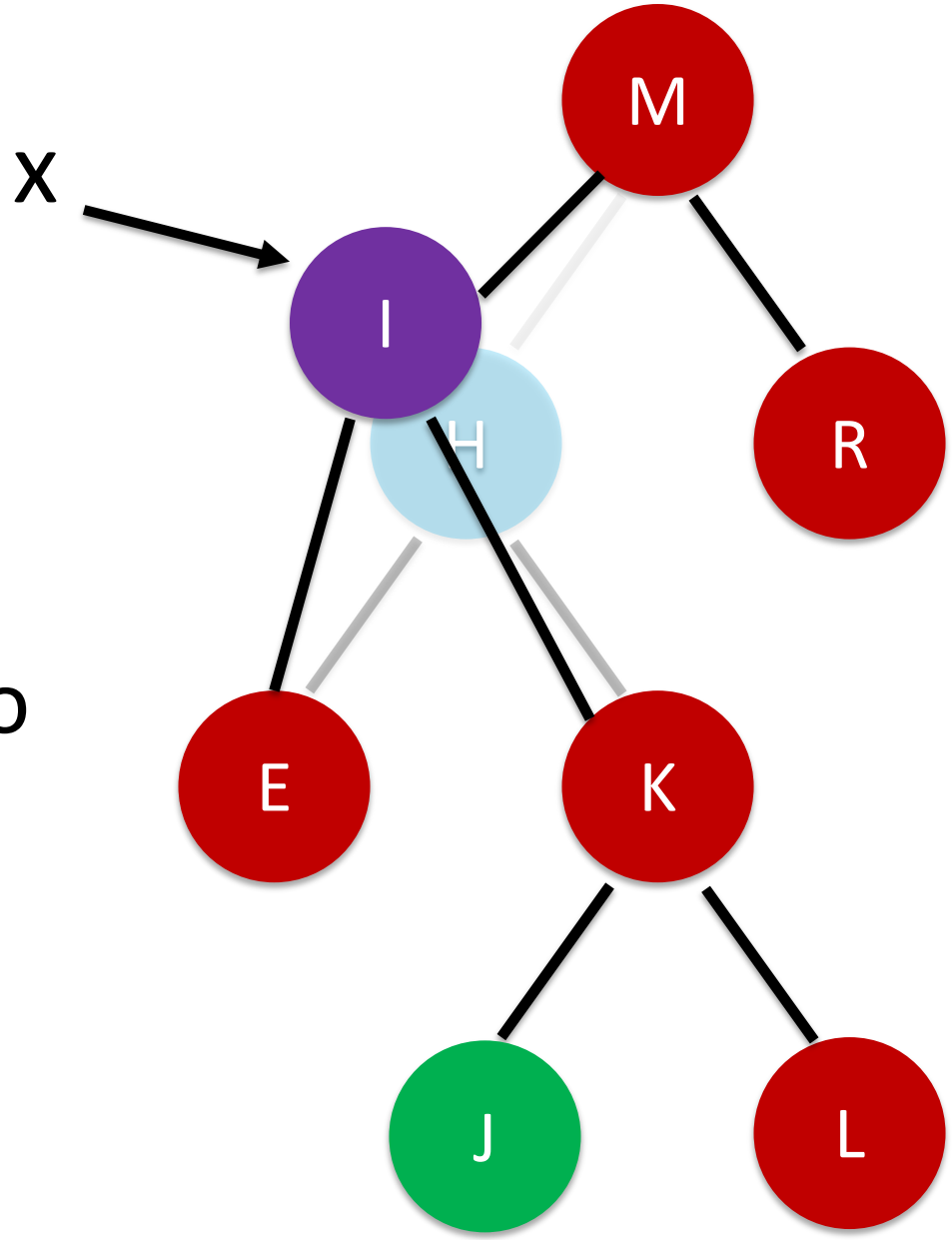



M
t
H
R
t.right
deleteMin(t.right)
E
K
x
I
J
L

Now plug t.left & t.right into the left and right fields of x



Finally, plug x into the left field of the parent node.



```java
public void delete(K key) {
  if (key == null)
    throw new IllegalArgumentException("delete: null key");
  root = delete(root, key);
}
```

```java
private Node delete(Node x, K key) {
  if (x == null) return null;

  int cmp = key.compareTo(x.key);
  if (cmp < 0) x.left = delete(x.left, key);
  else if (cmp > 0) x.right = delete(x.right, key);
  else {
    if (x.right == null) return x.left;
    if (x.left == null) return x.right;
    Node t = x;
    x = min(t.right);
    x.right = deleteMin(t.right);
    x.left = t.left;
  }
  x.size = size(x.left) + size(x.right) + 1;
  return x;
}
```

An Alternative API

```java
public interface OrderedMap<Key extends Comparable<Key>, Value> {
  int size();

  boolean isEmpty();

  boolean contains(Key key);

  Key floor(Key key);

  Key ceiling(Key key);

  Key max();

  Key min();

  void put(Key key, Value value);

  void deleteMin();

  void deleteMax();

  void delete(Key key);
}
```

```java
public interface OrderedMap<Key extends Comparable<Key>, Value> {
  boolean isEmpty();

  int size();

  Value get(Key key);

  boolean contains(Key key);

  Key ceiling(Key key);

  Key floor(Key key);

  Key max();

  Key min();

  OrderedMap<Key, Value> put(Key key, Value value);

  OrderedMap<Key, Value> deleteMin();

  OrderedMap<Key, Value> deleteMax();

  OrderedMap<Key, Value> delete(Key key);
```

Code

```java
public class BST<K extends Comparable<K>, V> implements OrderedMap<K, V> {

  private Node root;

  private class Node {
    private K key;
    private V value;
    private Node left;
    private Node right;
    private int size;
```

Work

How Much?

- How much time does an operation take?
- How much memory does it use?
- In the worst case? On average?
- How does the work grow as the input grows?

Measure the Input Size – N