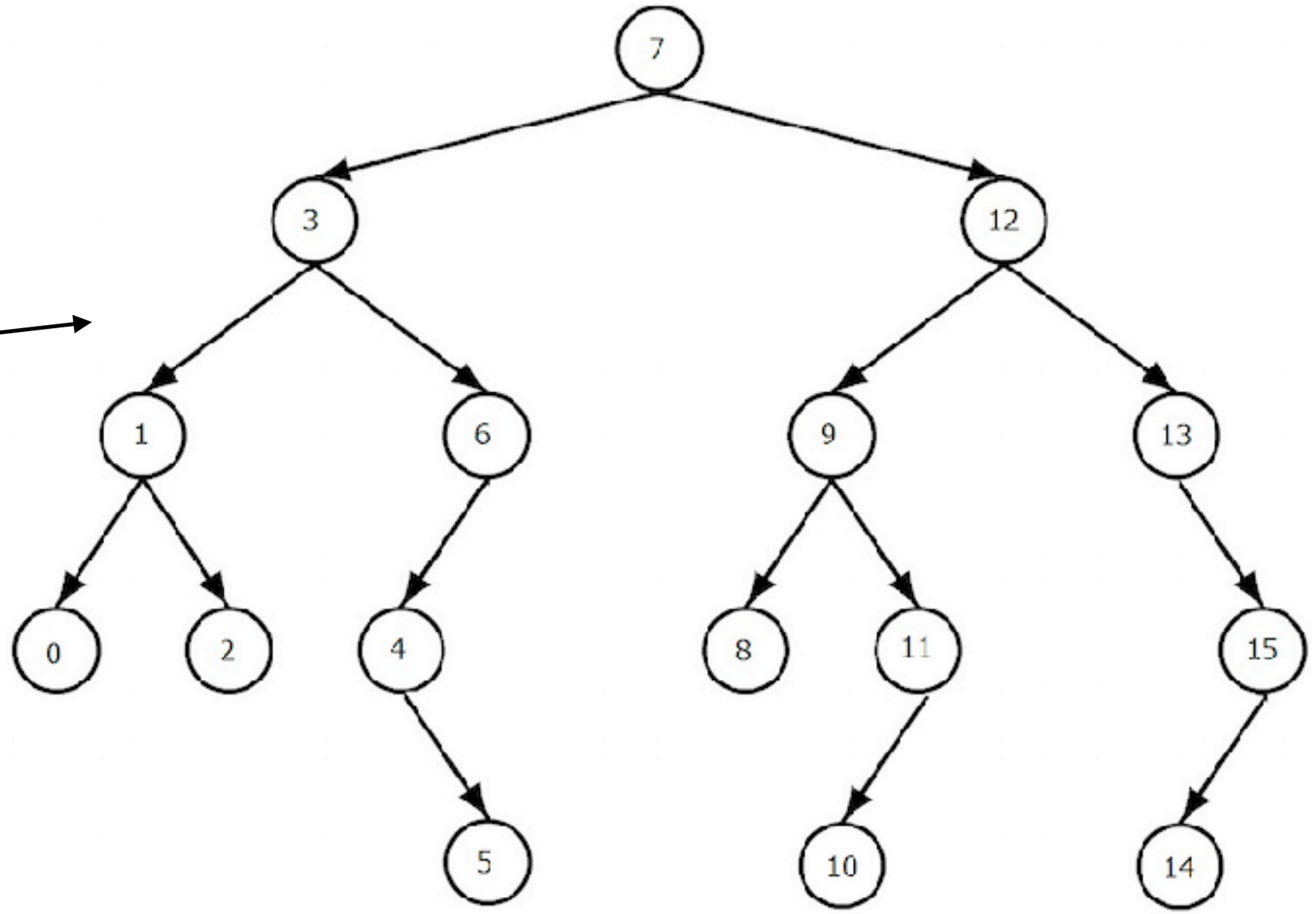
Binary search trees and skiplists

CS 146 - Spring 2017

Review question (from 46B)

binary search tree

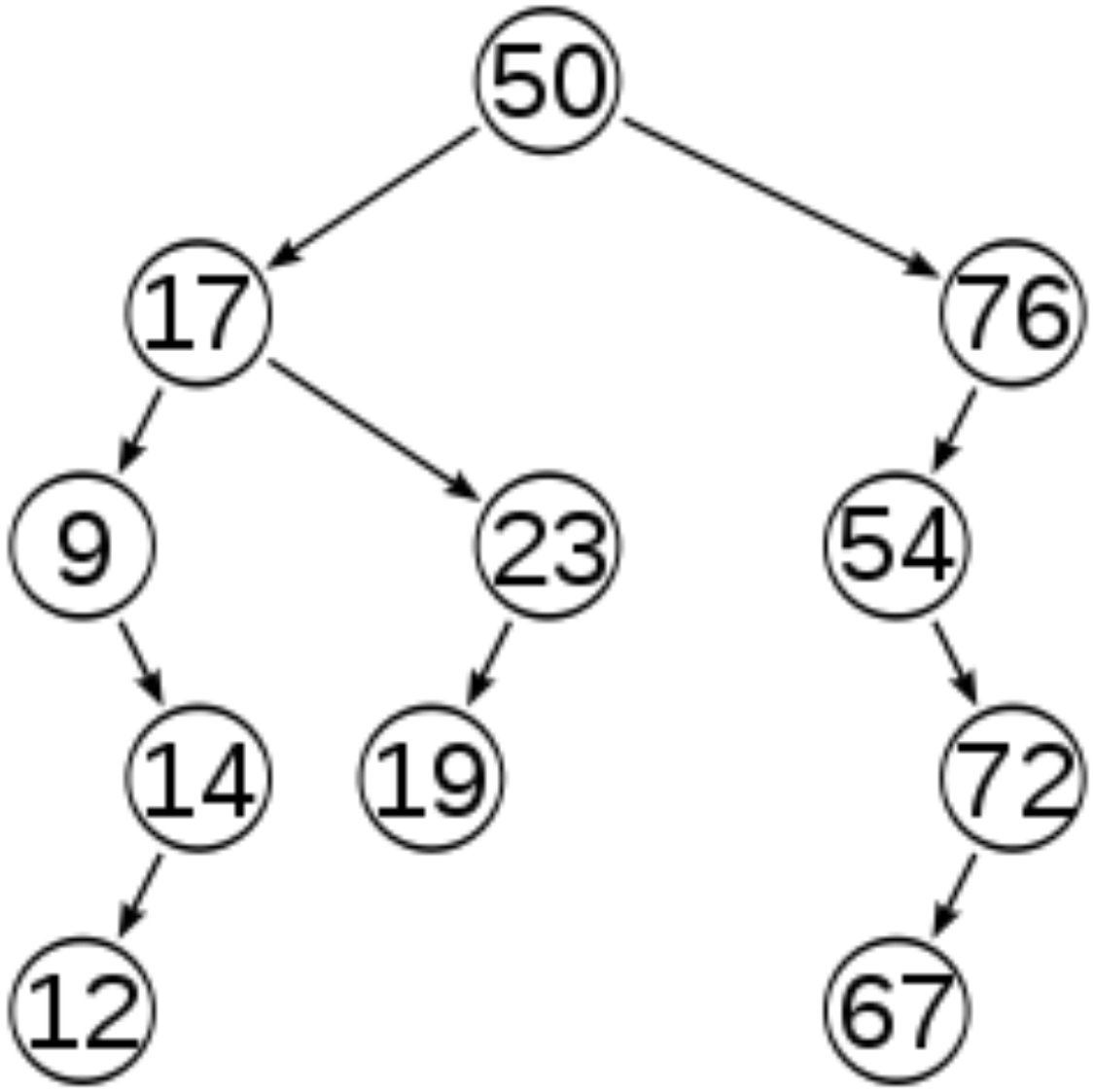


- Who is the successor of 12? 3? 8?
- What is the pattern for finding the successor?

Today

- 2 OrderedSet implementations
  - Binary search tree
  - Skiplist

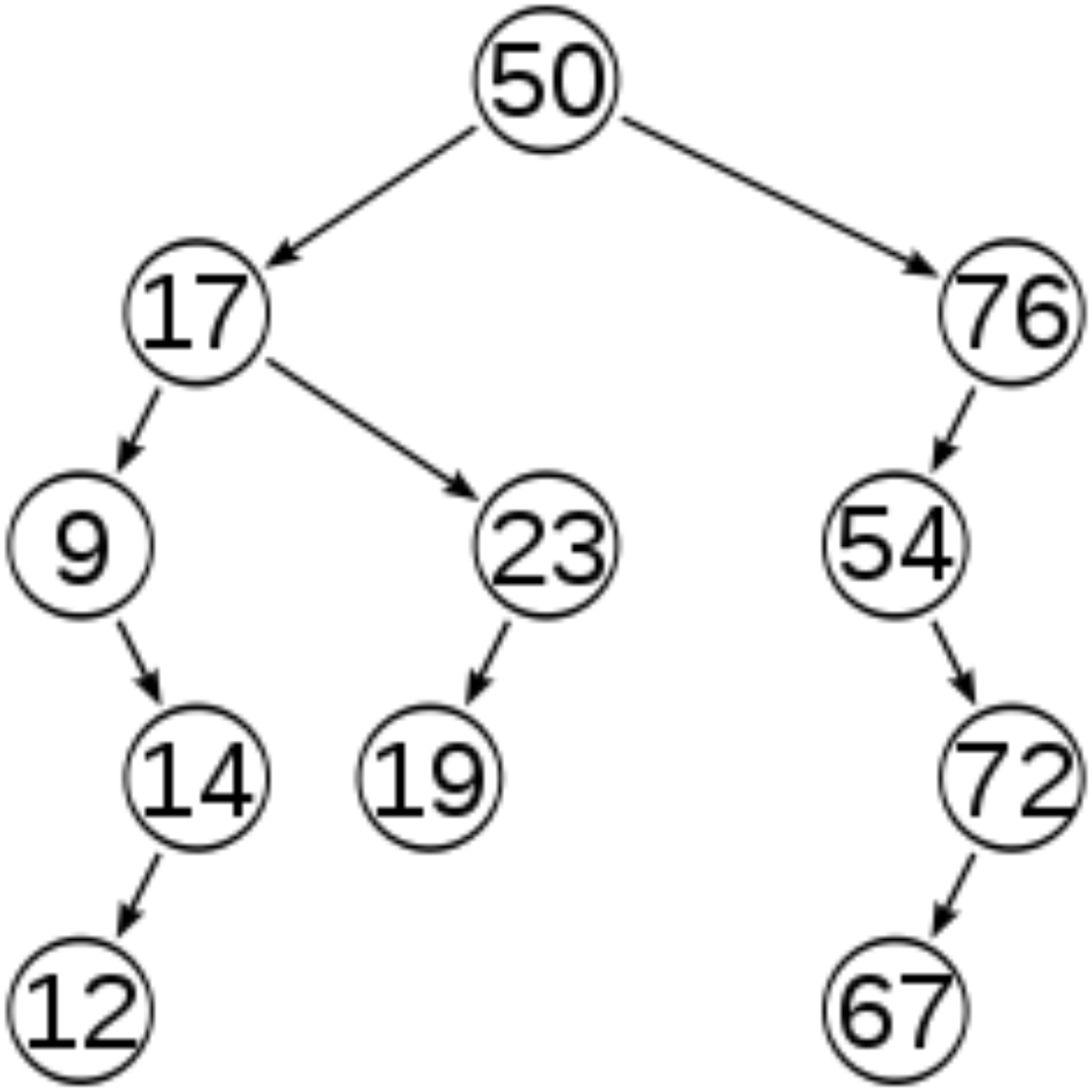
Binary search tree is



- a binary tree with…
- the **binary search tree property**

for every node
if x is in the node's left subtree,
x.key ≤ node.key
if x is in the node's right subtree,
node.key ≤ x.key

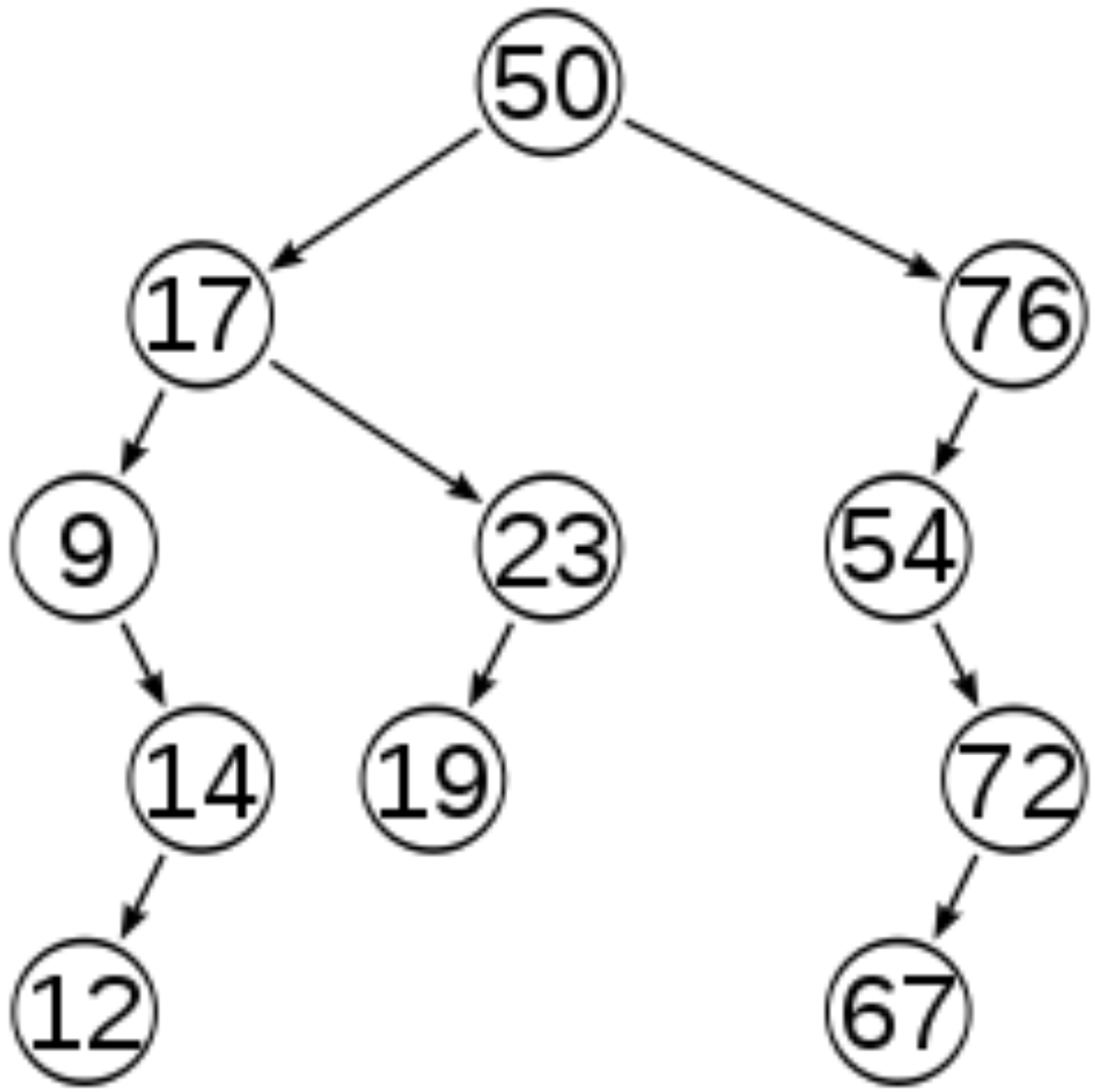
To search for a key in a BST



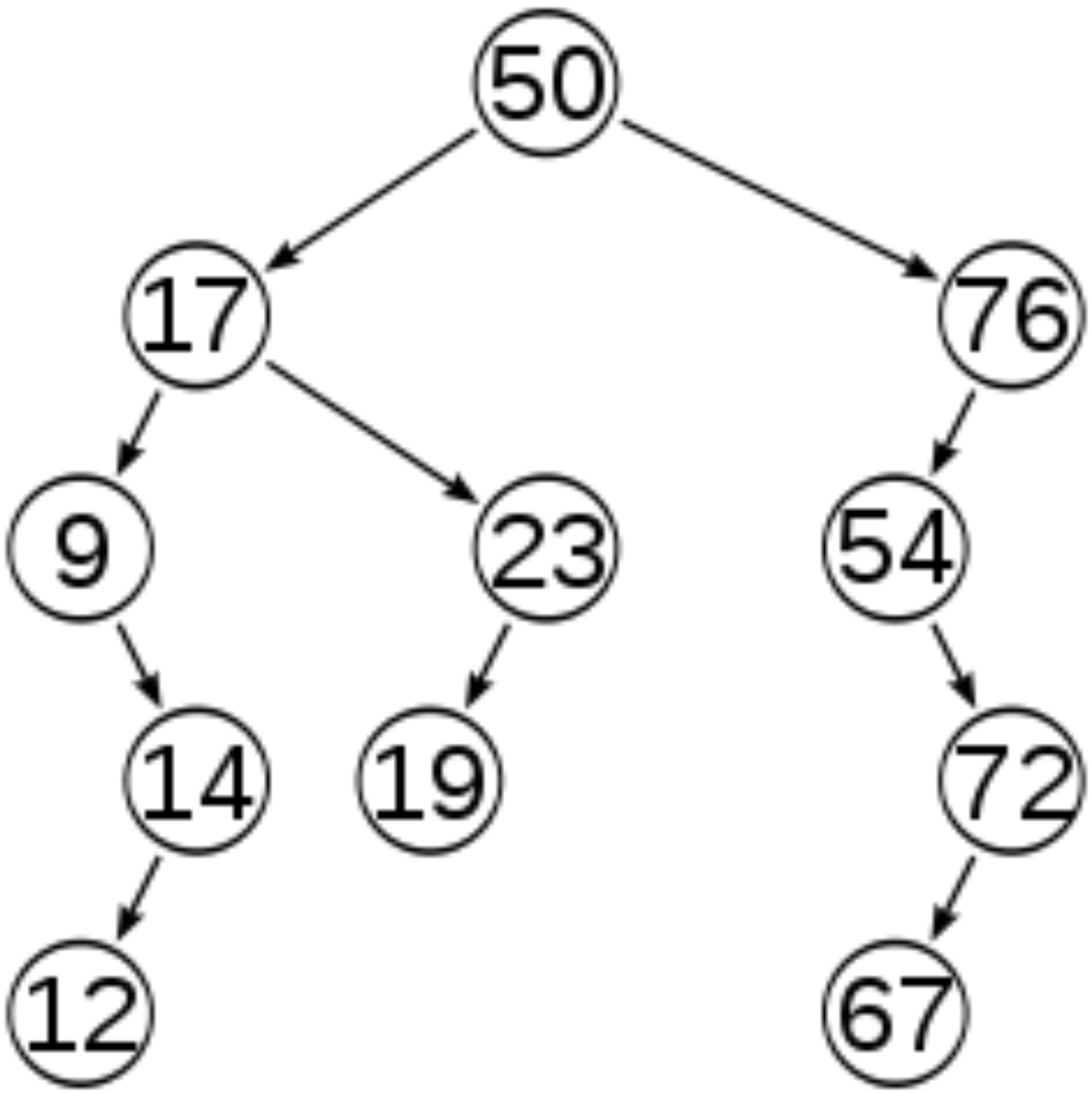
- start at root
- if key == node.key
  - (Key found!) return
- else if node.key < key
  - search left or
  - return (NOT FOUND) if no left
- else (key < node.key)
  - search right or
  - return (NOT FOUND) if no right

To insert

- perform a search until you can't keep going down
- insert there



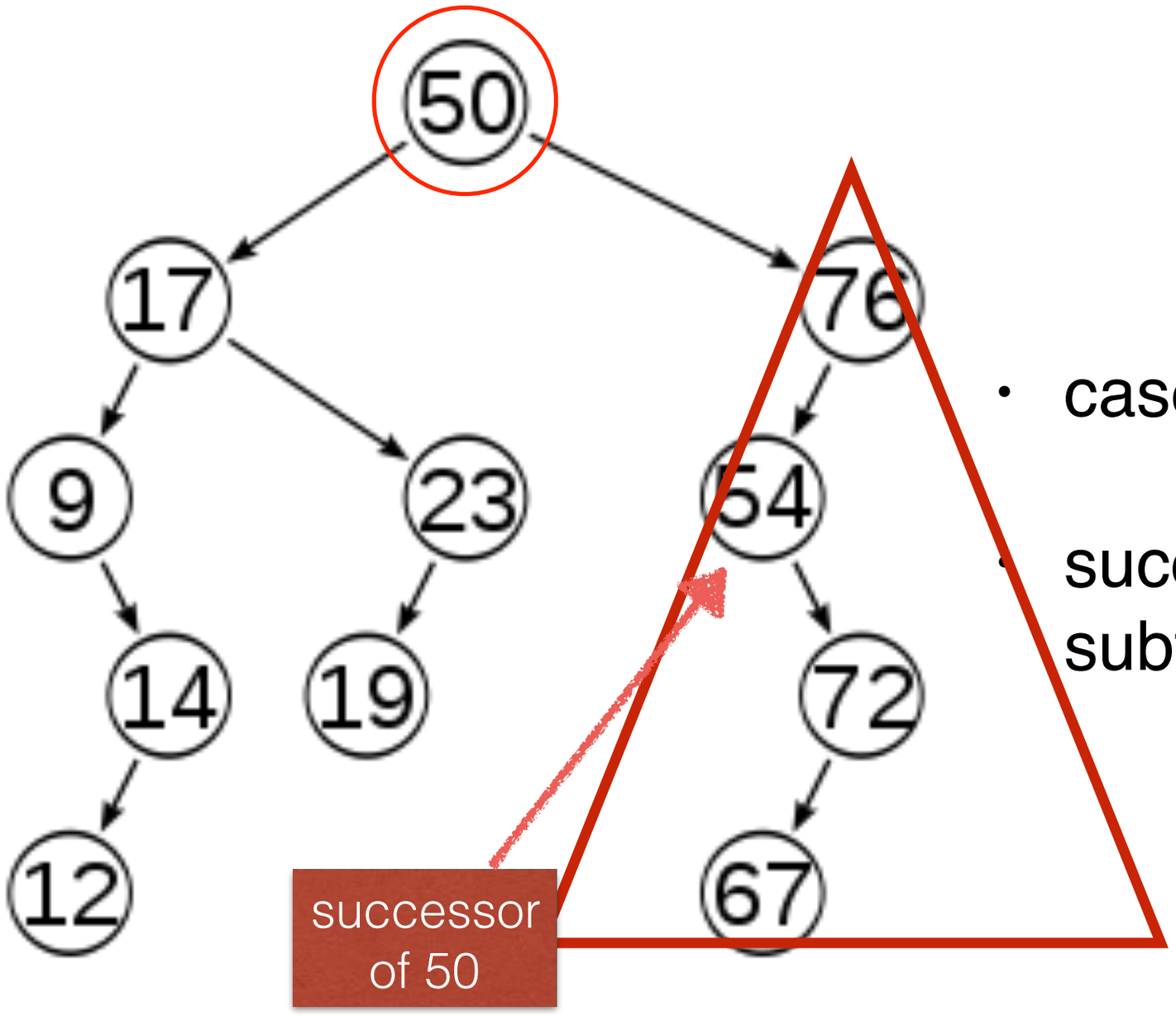
To find min



- keep going to the left child until you can't

similar idea with finding max

To find successor (1/2)

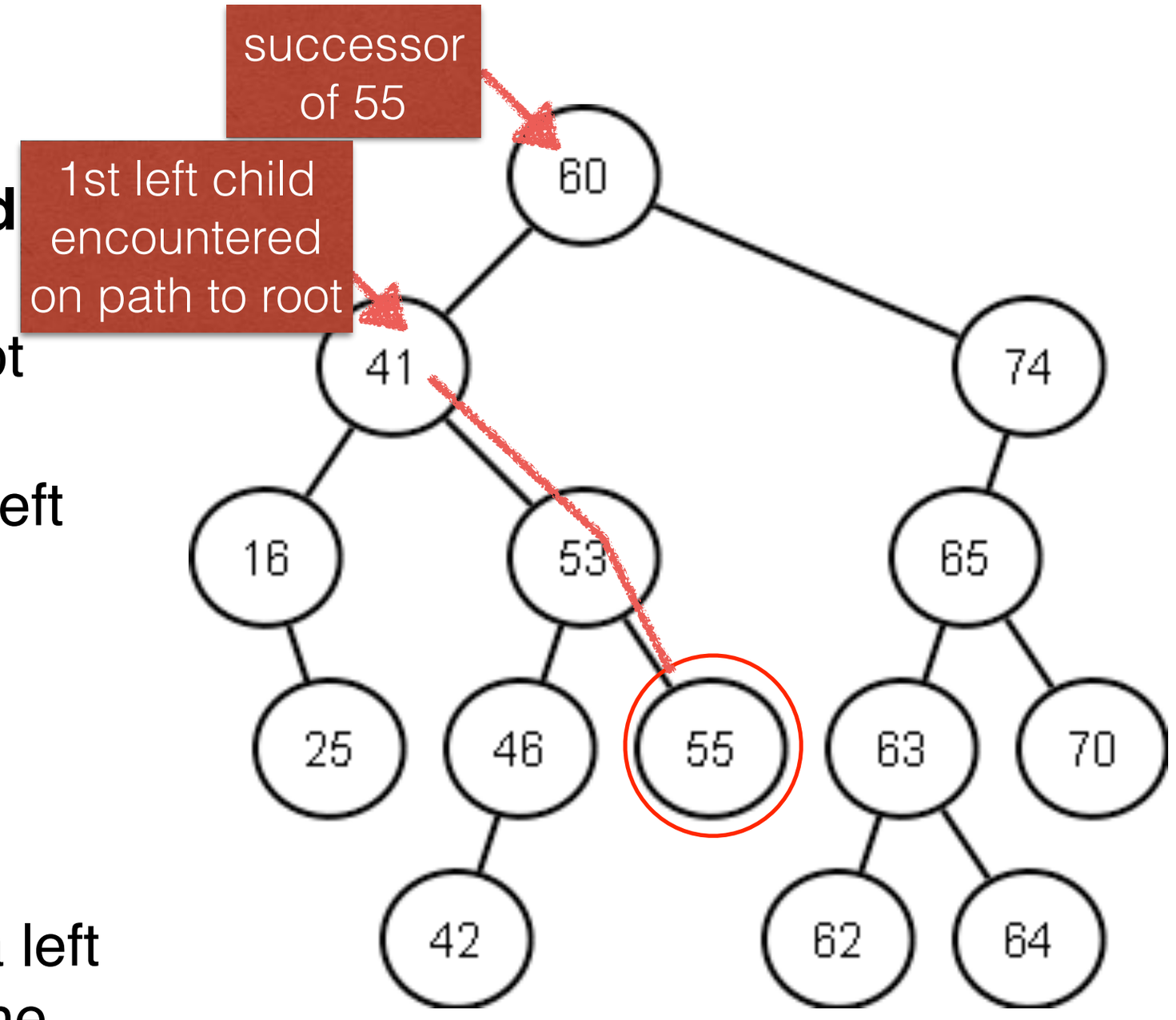


- case 1: **has right child**
- successor = min of right subtree

similar idea with finding predecessor

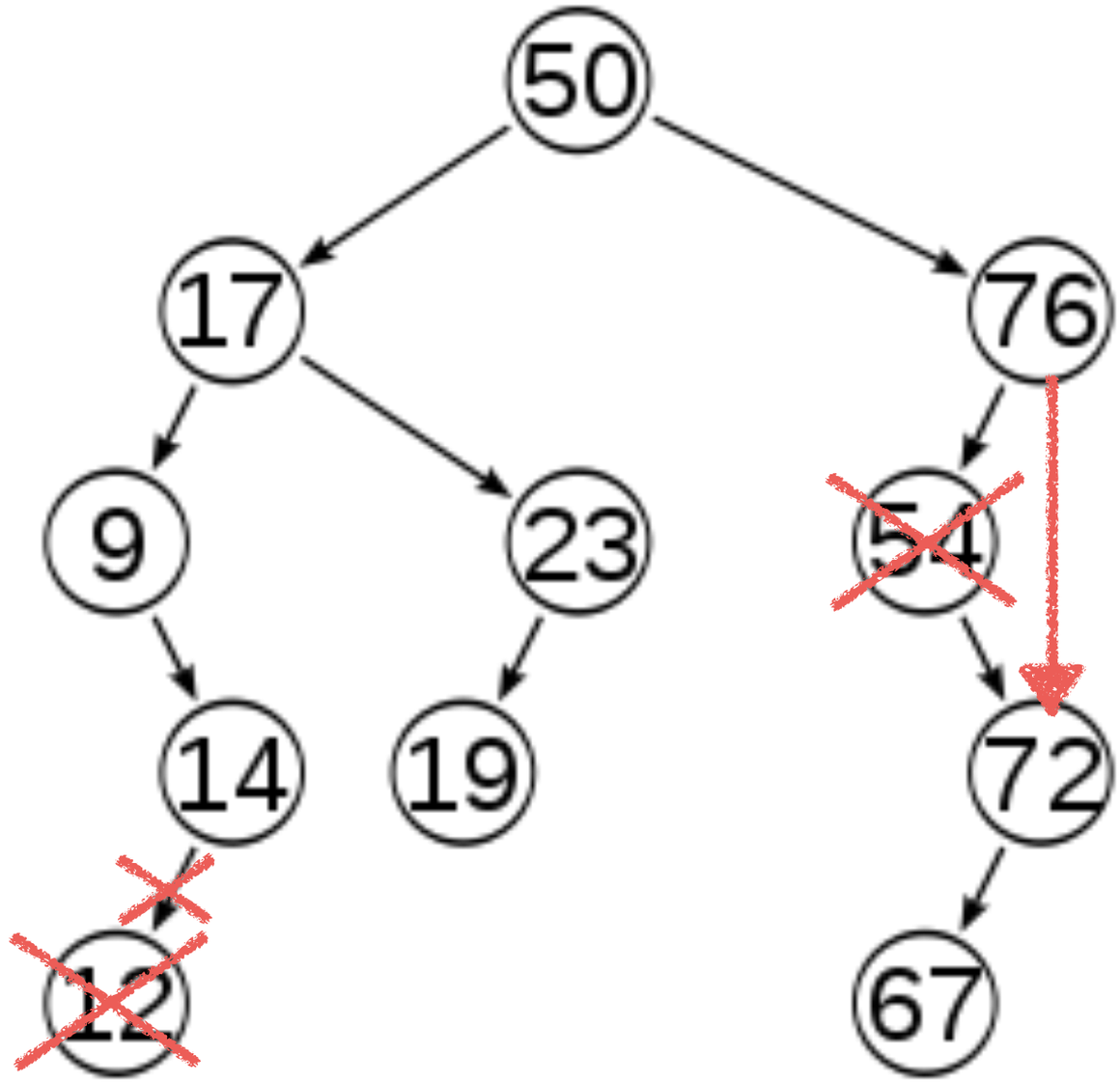
To find successor (2/2)

- case 2: **no right child**
- go up towards the root
- stop when you are a left child
- your parent is the successor
- note: if you're never a left child and you reach the root, you had no successor



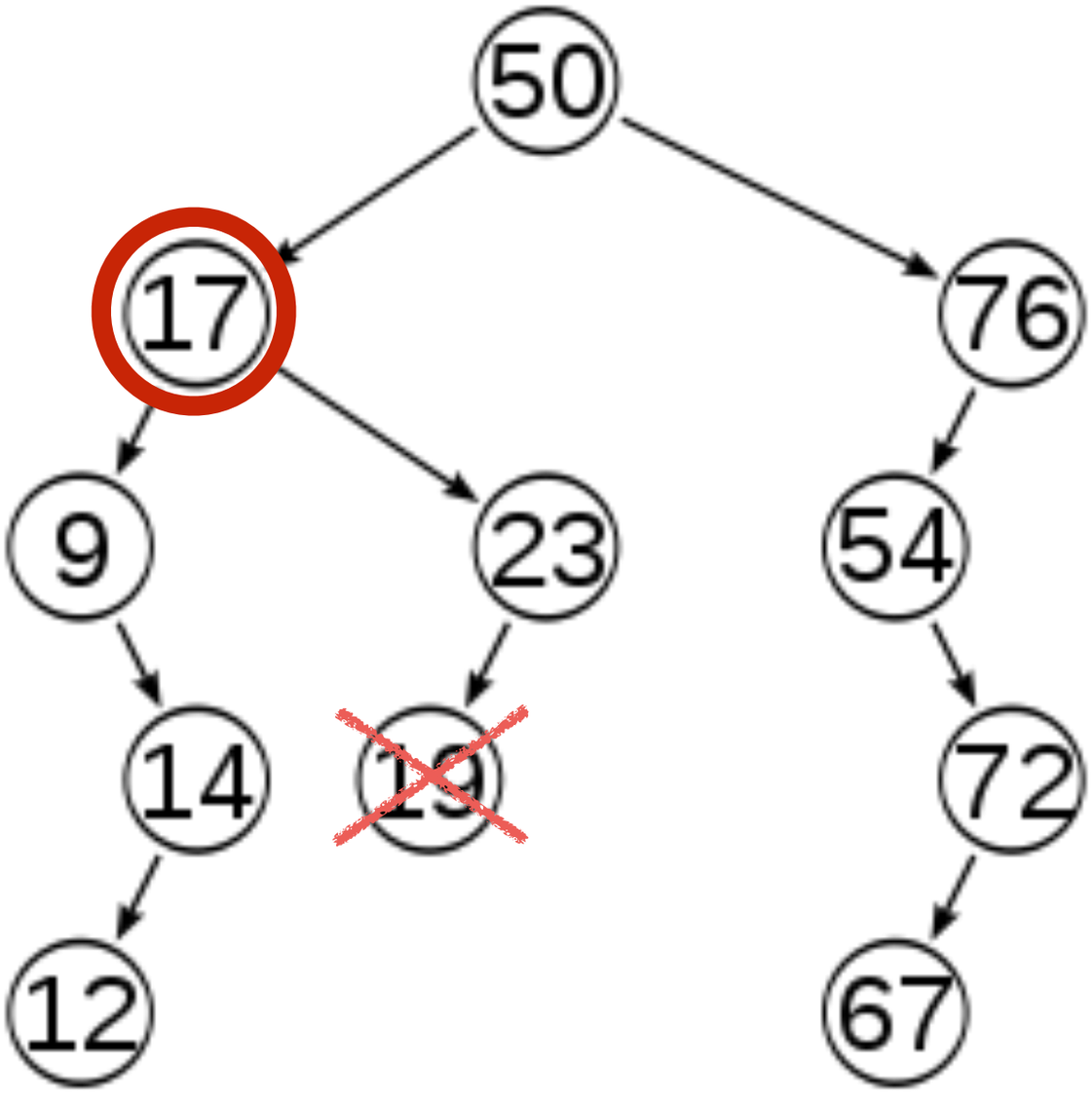
similar idea with finding predecessor

To delete a node (1/2)



- case 1: **no children**
  - delete link parent->itself
- case 2: **one child**
  - relink parent and child

To delete a node (2/2)



- case 3: **two children**
  - find the node's successor
  - **swap key** with successor
  - delete successor

to delete 17, swap values 17 and 19 (or 14), then delete node that used to contain 19 (but now contains 17)

Skiplists

For slides, go to
https://www.cs.umd.edu/class/spring2008/cmsc420/L12.SkipLists.pdf

Recap: BST vs skiplists

today

today

later: red-black, AVL, treaps, …

today

sorted set operations	**BST (h means height)**	**unbalanced BST (h = O(n))**	**balanced BST (h = O(log n))**	**skiplists**
add/delete	O(h)	O(n)	O(log n) + time to rebalance	2 log_2 n
search	O(h)	O(n)	O(log n)	2 log_2 n
predecessor/ successor	O(h)	O(n)	O(log n)	2 log_2 n
	worst-case	worst-case	worst-case	on expectation, and with high probability