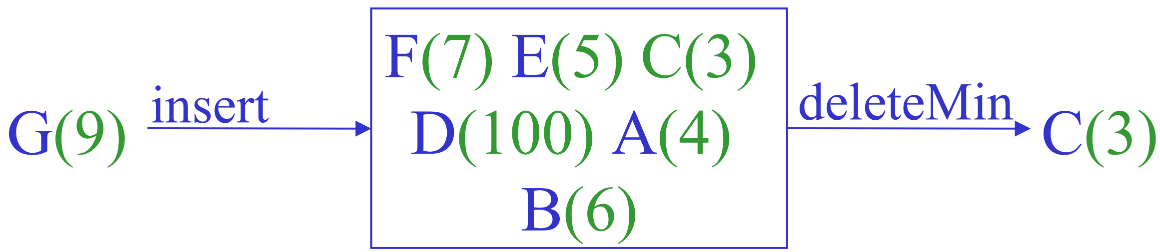
Priority Queues (Heaps) Data Structure

Queues : Limitations

- Consider applications
  - ordering CPU jobs
  - searching for the exit in a maze
  - emergency room admission processing
- Problems?
  - short jobs **should go first**
  - most promising nodes **should be searched first**
  - most urgent cases **should go first**

Priority Queue ADT

- Priority Queue operations
  - create
  - destroy
  - insert
  - deleteMin
  - is_empty



- Priority Queue property: for two elements in the queue, *x* and *y*, if *x* has a lower priority value than *y*, *x* will be deleted before *y*

Applications of the Priority Q

- Hold jobs for a printer in order of length
- Store packets on network routers in order of urgency
- Simulate events
- Anything *greedy*

Naïve Priority Queue Data Structures

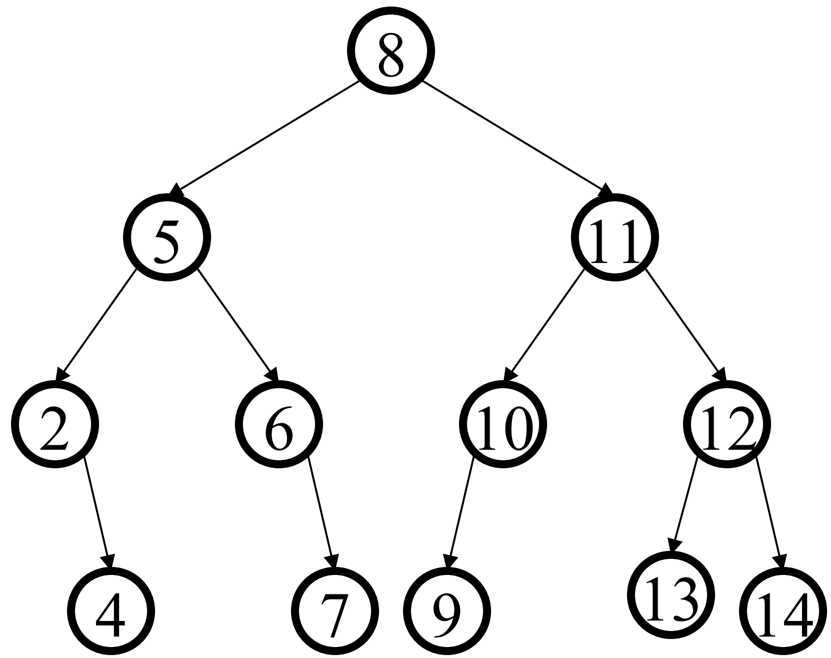
- Linked list:
  - *Insert: O(1)*
  - *DeleteMin: O(N)*
- Sorted Linked list:
  - *Insert: O(N)*
  - *DeleteMin: O(1)*

BST Tree Priority Queue Data Structure

- Regular BST:
  - *Insert: O(log n)*
  - *deleteMin:O(log n)*

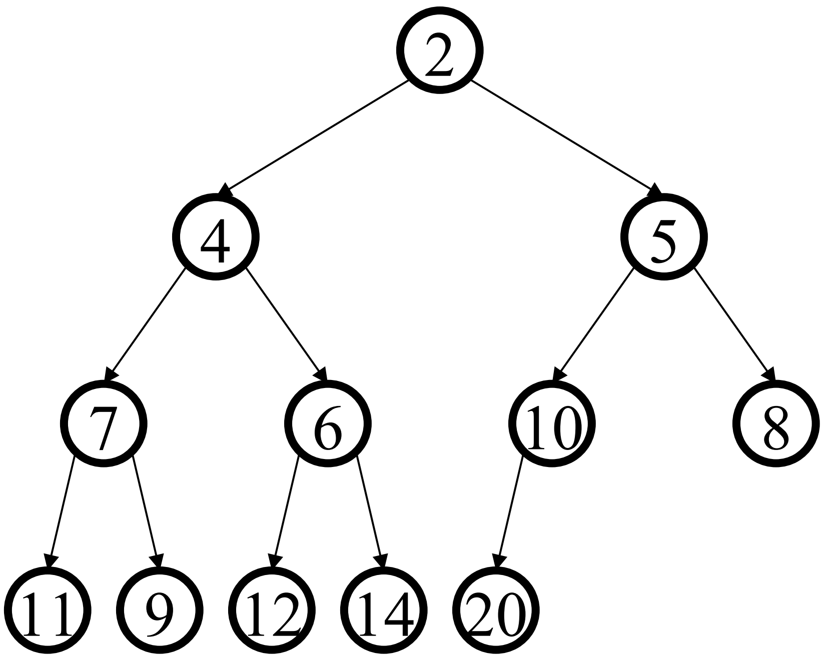
    (Average)

- AVL Tree:
  - *insert:O(log n)*
  - *deleteMin:O(log n)*

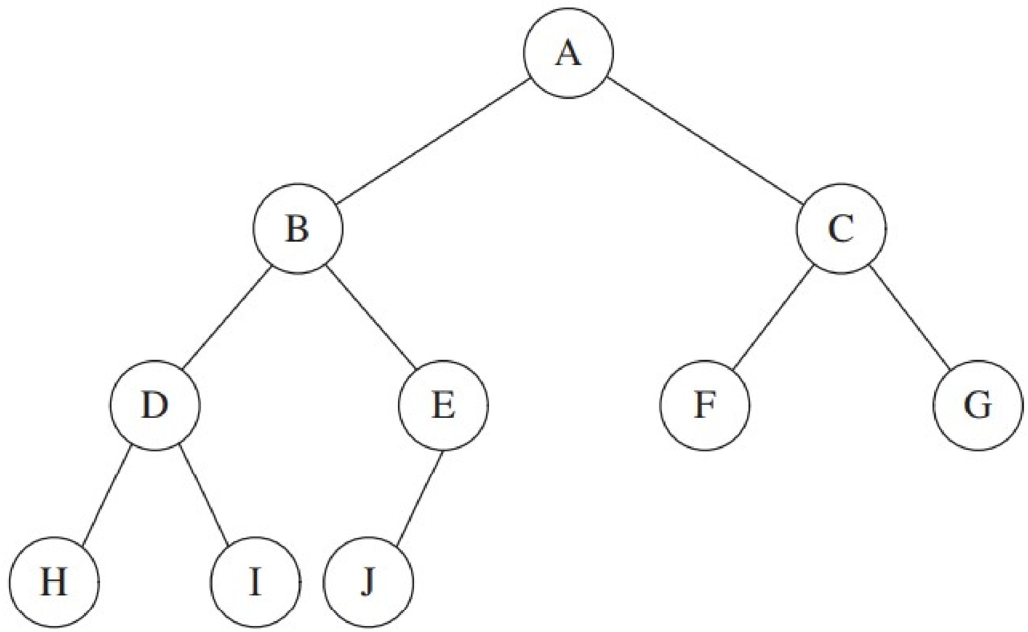


Binary Heap Priority Q Data Structure

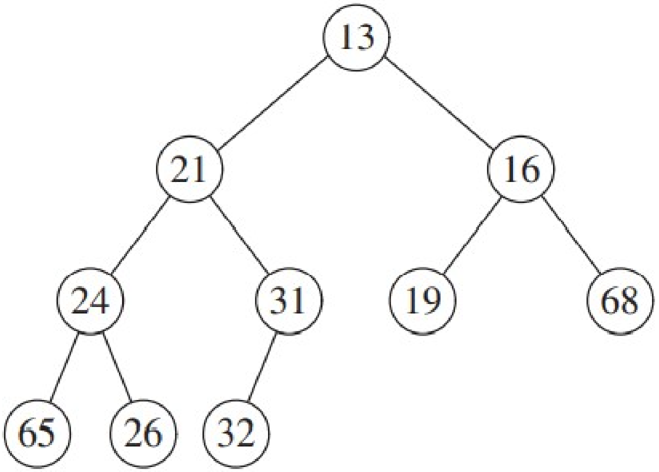
- Heap-order property
  - parent's key is less than children's keys
  - result: minimum is always at the top
- Structure property
  - Binary tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right. Such a tree is known as Complete Binary tree.
  - Complete binary tree of height 'h' has between 2^h and 2^(h+1) - 1 nodes. Height 'h' is O(log n).



Complete Binary Tree Example



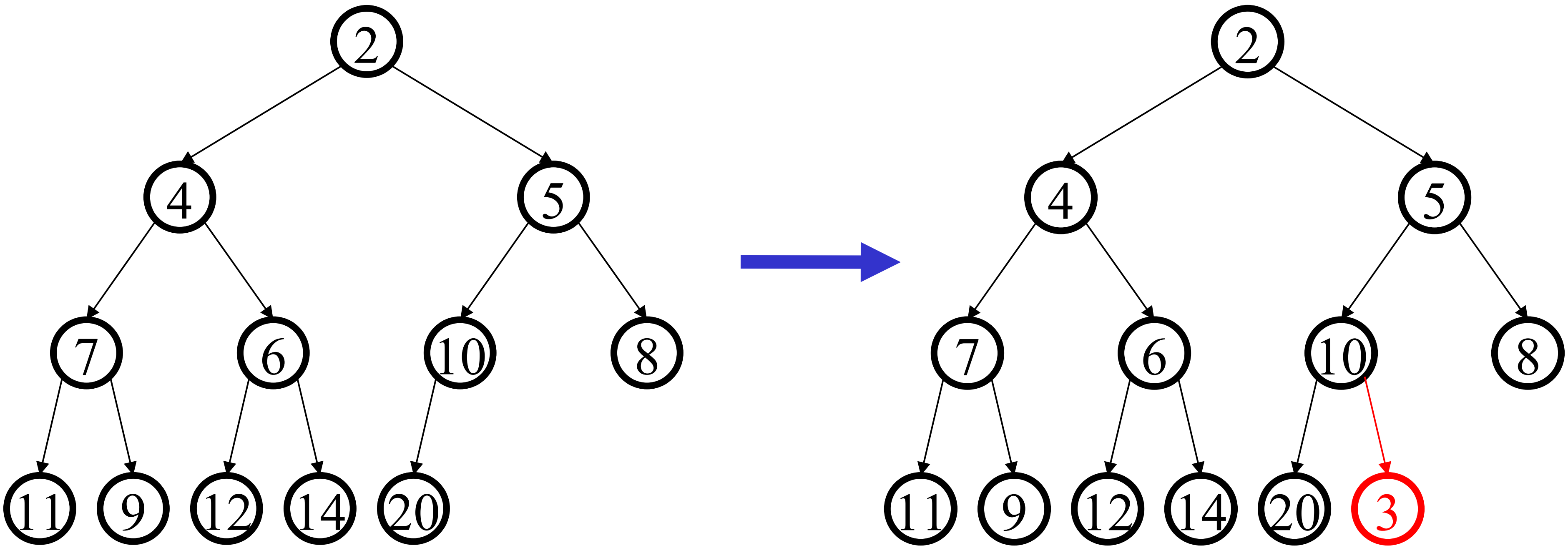
Binary heap Example



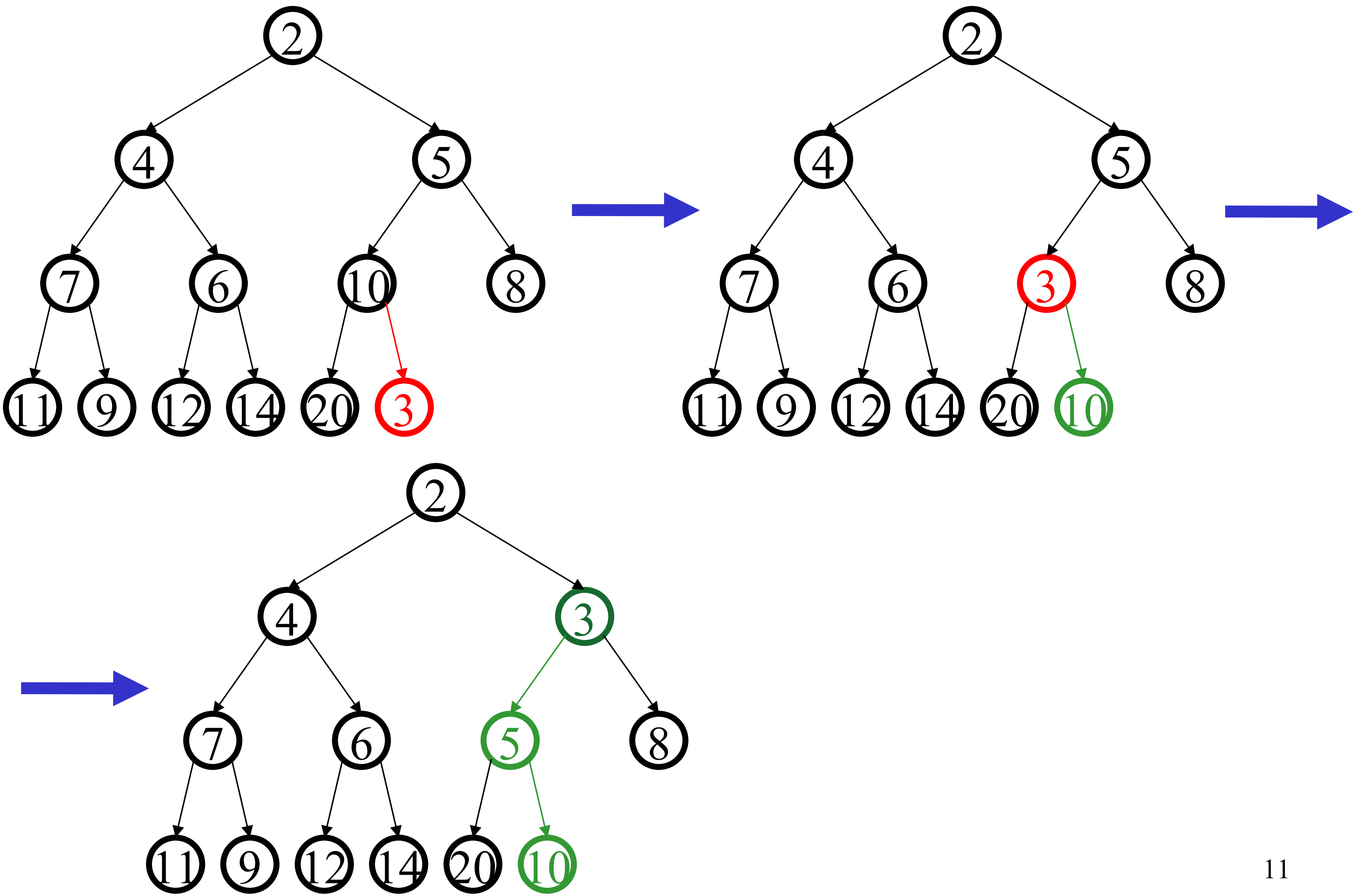
Satisfies both structure property and heap property

Basic Operations :Insert

```
pqueue.insert(3)
```

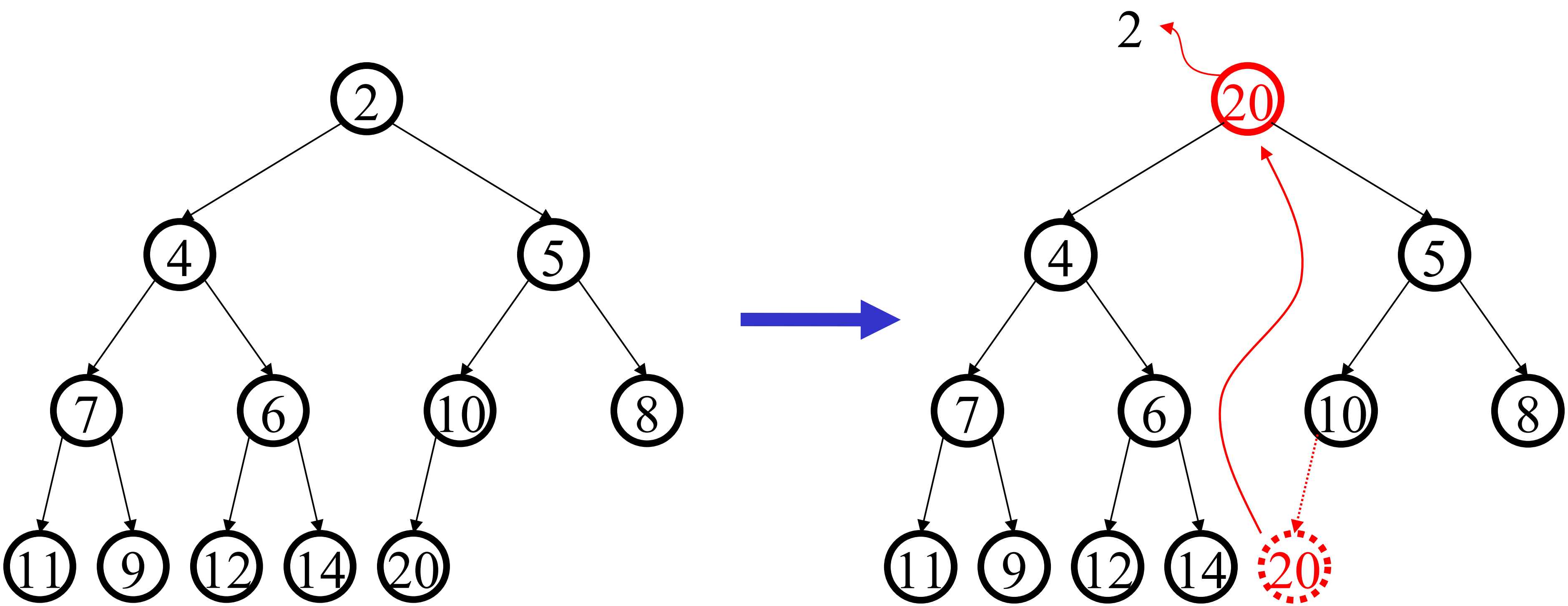


Percolate Up

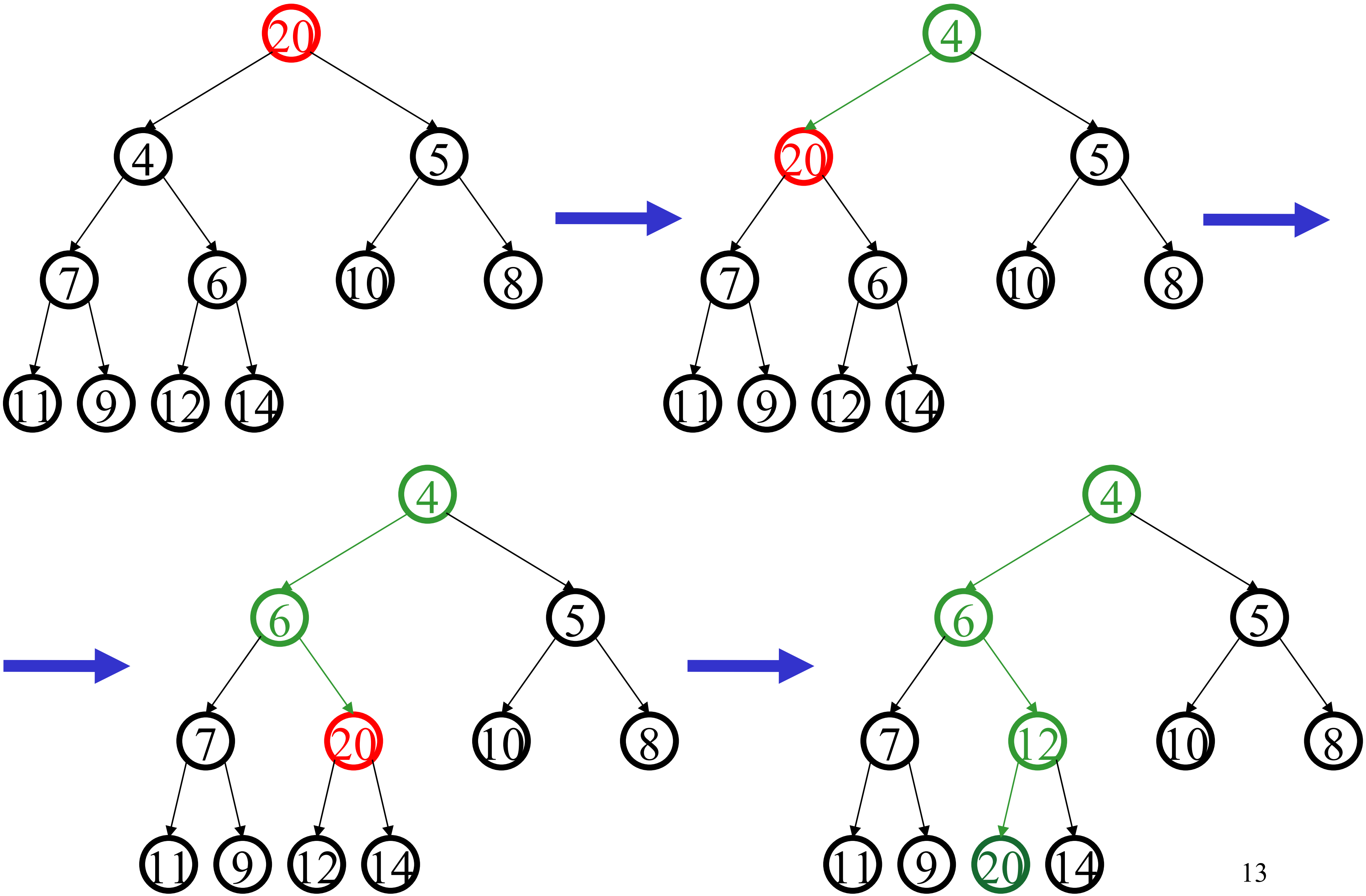


DeleteMin

`pqueue.deleteMin()`



Percolate Down



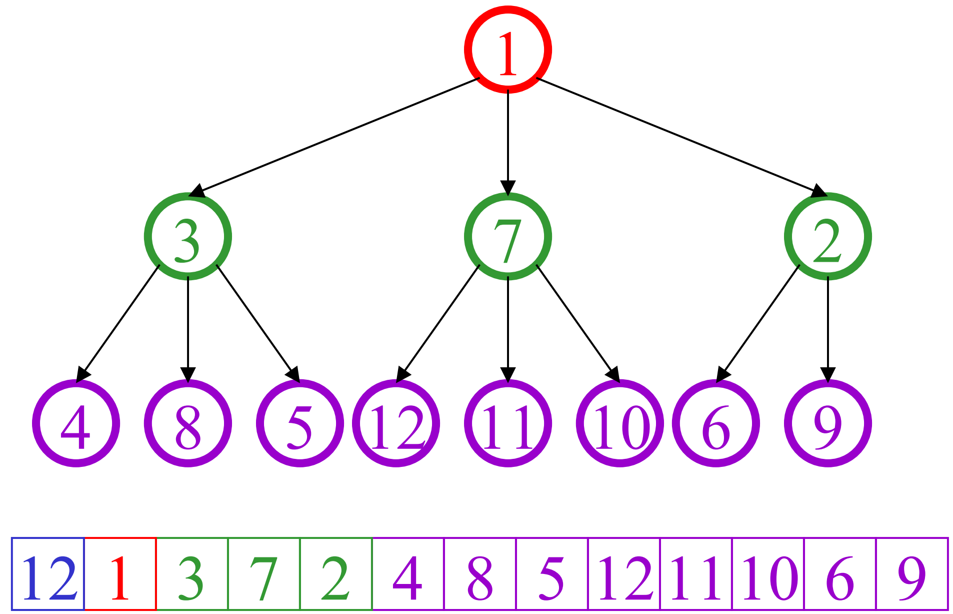
Performance of Binary Heap

	Binary heap worst case	Binary heap avg case	AVL tree worst case	BST tree avg case
Insert	O(log n)	O(1) percolates 1.6 levels	O(log n)	O(log n)
Delete Min	O(log n)	O(log n)	O(log n)	O(log n)

- In practice: binary heaps much simpler to code, lower constant factor overhead

d-Heaps

- Generalization of Binary heap
- Each node has *d* children
- Still representable by array
- Good choices for *d*:
  - optimize performance based on # of inserts/removes
  - choose a power of two for efficiency
  - fit one set of children in a cache line
  - fit one set of children on a memory page/disk block
  - Insert – $O(\log_d n)$ and Delete $O(n \log_d n)$



New Operation: Merge

Merge(H1,H2): Merge two heaps H1 and H2 of size O(N).

- *E.g*. Combine queues from two different sources to run on one CPU.

- Merging is harder in Binary Heap
- To support merge, we will discuss other data structures

Leftist Heaps

- An alternative heap structure that also enables fast merges
- Based on binary trees rather than *k*-ary trees

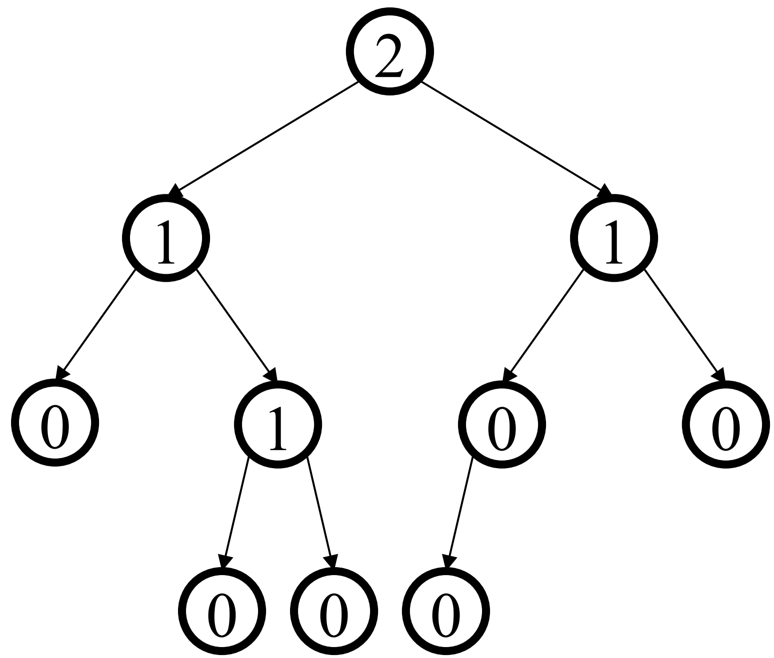
Leftist Heaps

- Leftist heap:
  - almost all nodes are on the left
  - all the merging work is on the right

Null Path Length

the *null path length (npl)* of a node is the number of nodes between it and a null in the tree

- npl(null) = -1
- npl(leaf) = 0
- npl(single-child node) = 0



Leftist Heap Properties

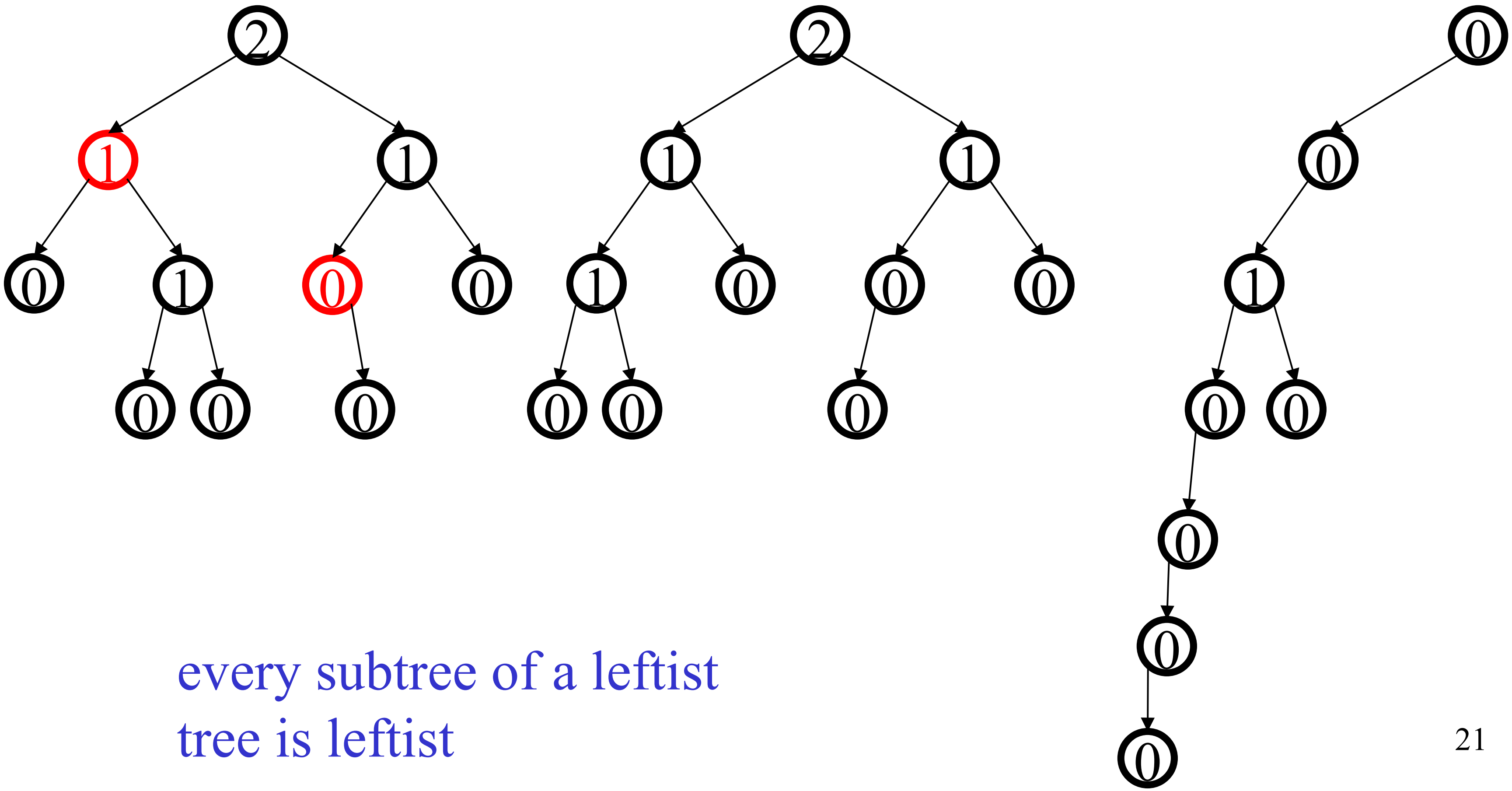
- Heap-order property
  - parent's priority value is $\leq$ to childrens' priority values
  - result: minimum element is at the root
- Leftist property
  - null path length of left subtree is $\geq$ npl of right subtree
  - result: tree is at least as "heavy" on the left as the right

Leftist tree examples

NOT leftist

leftist

leftist

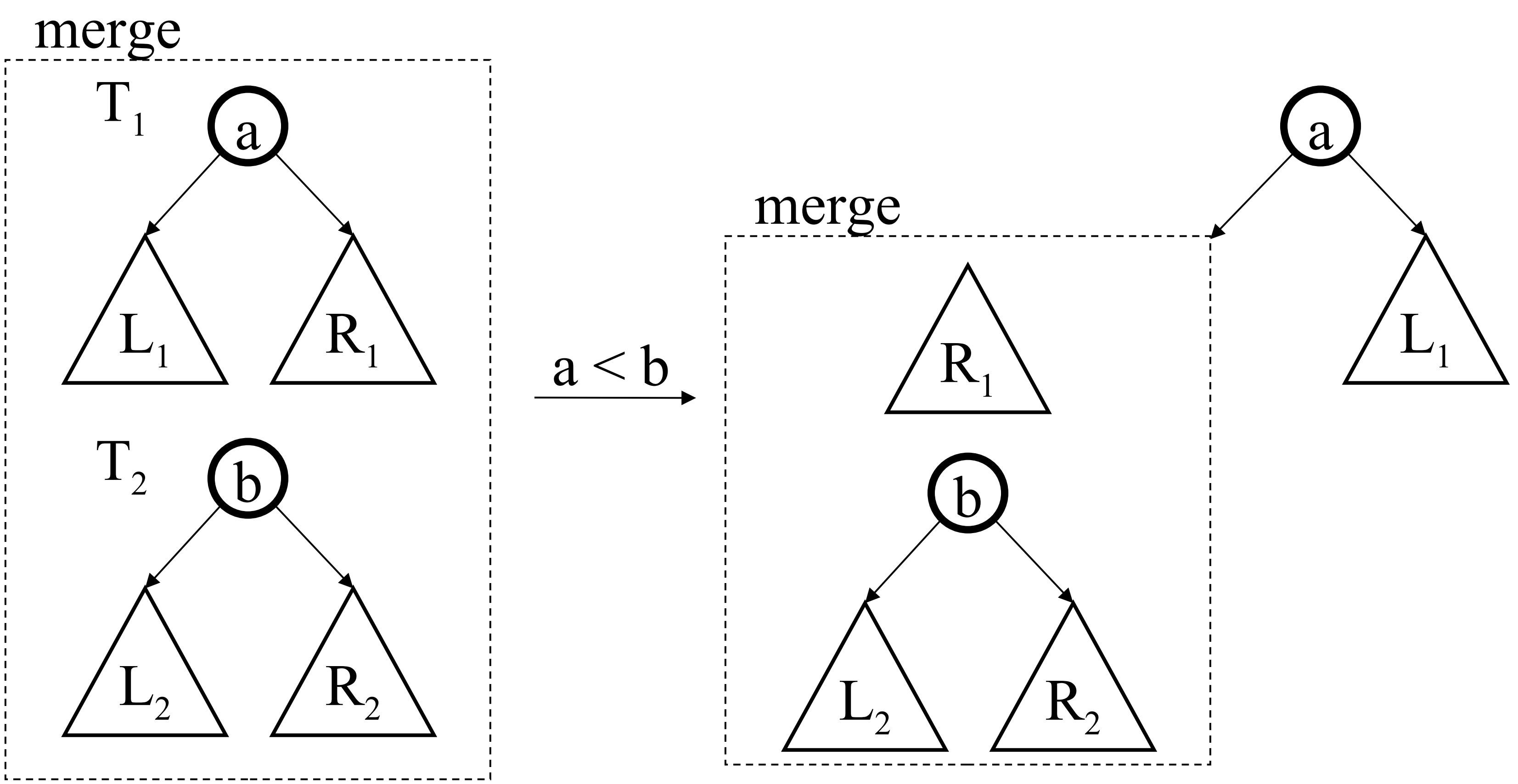


every subtree of a leftist tree is leftist

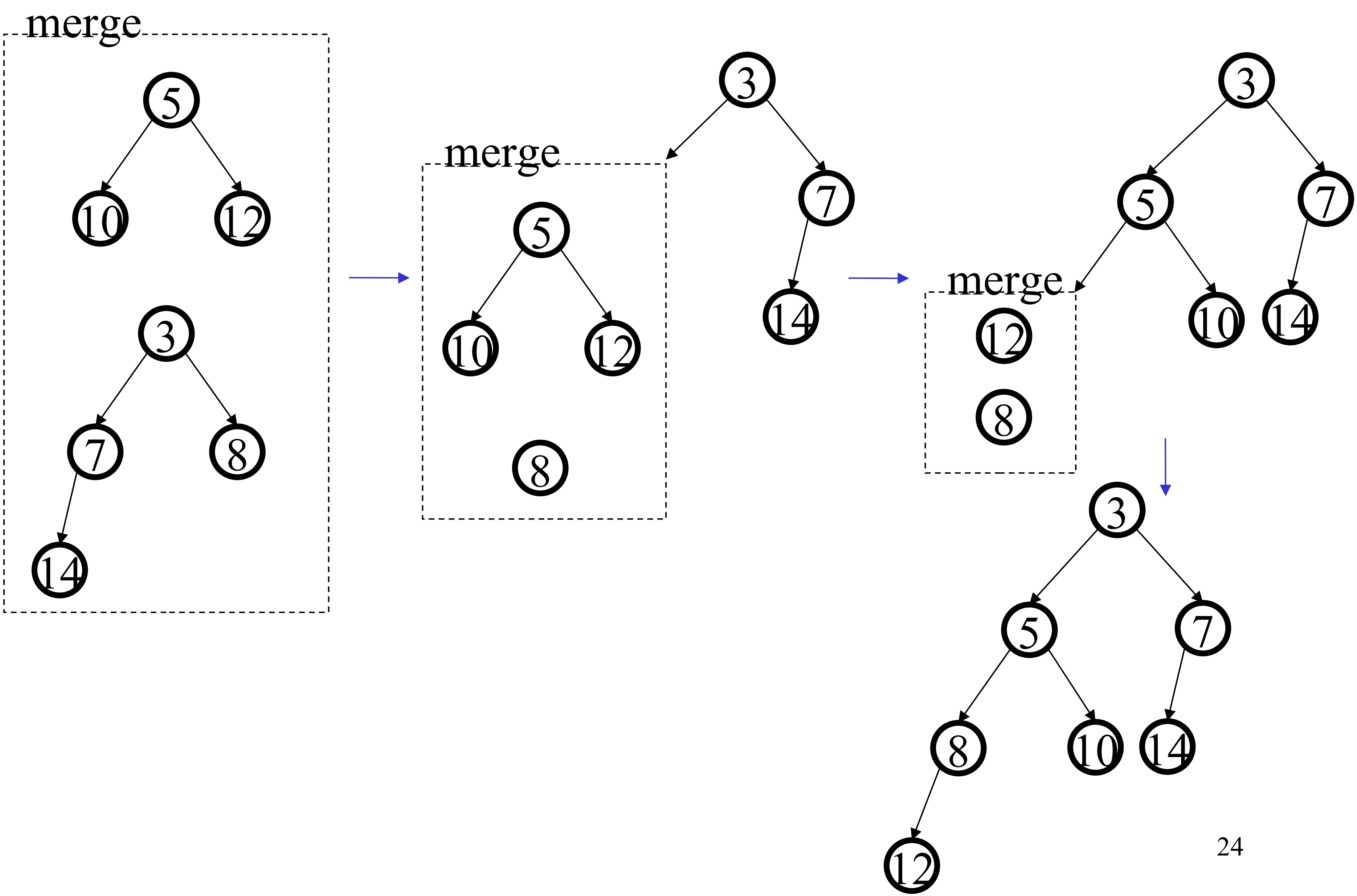
Skew Heaps

- Problems with leftist heaps
  - extra storage for npl
  - two pass merge (with stack!)
  - extra complexity/logic to maintain and check npl
- Solution: skew heaps
  - blind adjusting version of leftist heaps
  - amortized time for merge, insert, and deleteMin is O(log n)
  - worst case time for all three is O(n)
  - merge *always* switches children when fixing right path
  - iterative method has only one pass

Merging Two Skew Heaps



Example


merge
5
10
12
3
7
8
14
merge
5
10
12
8
3
7
14
merge
12
8
3
5
7
10
14
3
5
7
8
10
14
12