Concurrent Access of Priority Queues

R.V. Nageshwara and V. Kumar

Priority Queue

A **Priority Queue** is a data structure for maintaining a set S of elements, each with an associated key [CLRS]

Operations:

- *INSERT*: insert an element into the set
- *MIN*: peek at the smallest element of the set
- *EXTRACT-MIN*: remove and return the smallest element in the set
- *INCREASE-KEY / DECREASE-KEY*: adjust the key value of an element

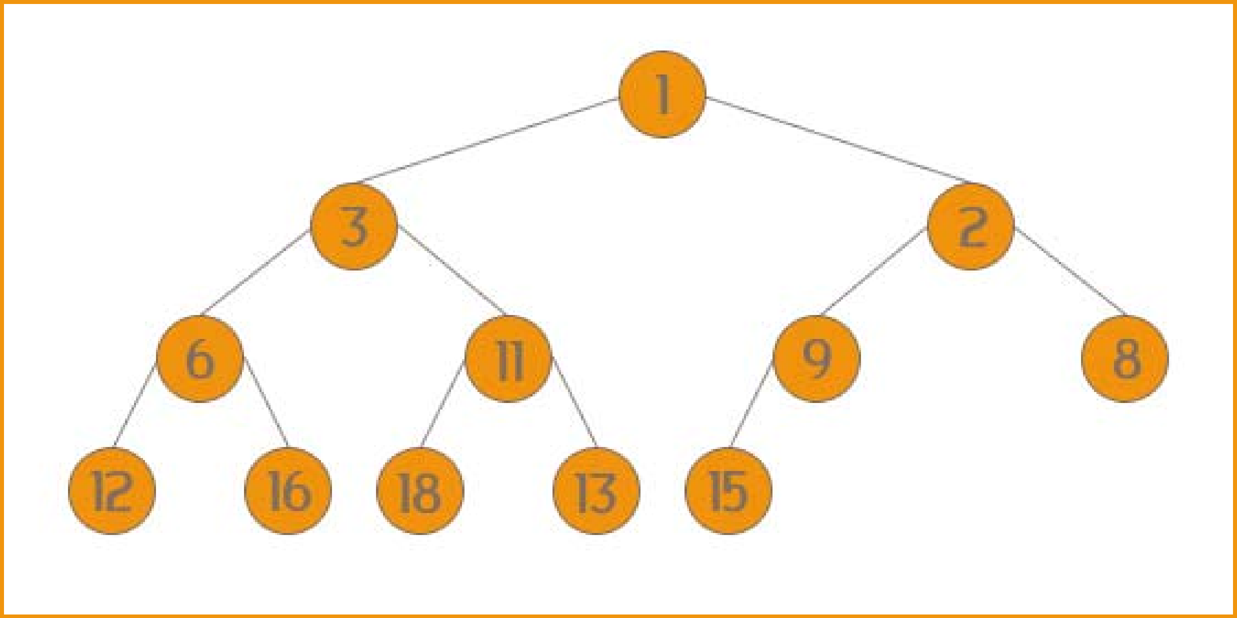
Applications:

- Job scheduling (max-ordered priority queue)
- Event Simulation (min-ordered priority queue)

Binary Heap

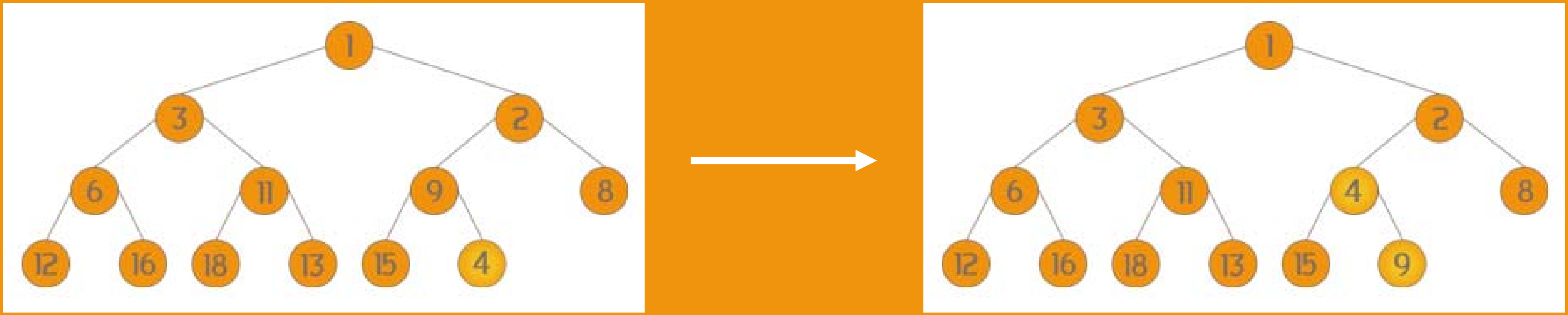
A **Binary Heap** is a data structure that is an array object that can be viewed as a nearly complete binary tree. The tree is completely filled on all levels except the lowest, which may only be partially filled. [CLRS]

Heap Order Property: Key stored at the parent is smaller or equal to the key stored at either child.



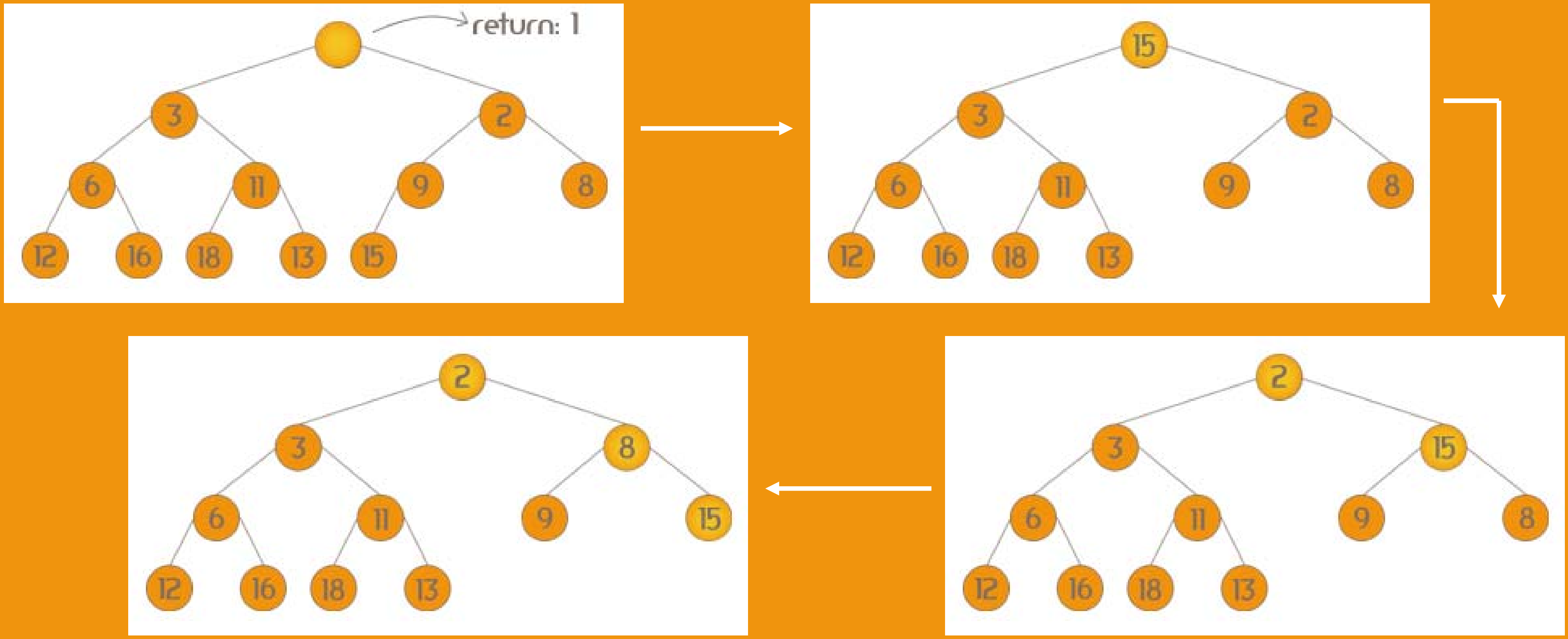
Binary Heap

Inserts into a Binary Heap are done in a bottom up manner. The new element is inserted into the next available spot and bubbled up the tree to satisfy the heap order property.



Binary Heap

Deletes from a Binary Heap are done in a top down manner. The min element is removed and replaced with the most recently inserted element. Then the element is bubbled down the tree to maintain the heap order property.



Binary Heap

The problem that arises from preparing a Binary Heap for concurrent access is the fact the insert and delete operations work in opposite directions, this could face potential deadlocks if the tree was locked on a node by node basis.

The solution to this problem, as presented by Nageshwara and Kumar, is to modify the insert operation so that it too proceeds in a top down manner. Thus having both procedures would progress down the heap in the same direction and deadlocks would be eliminated.

Modified Binary Heap

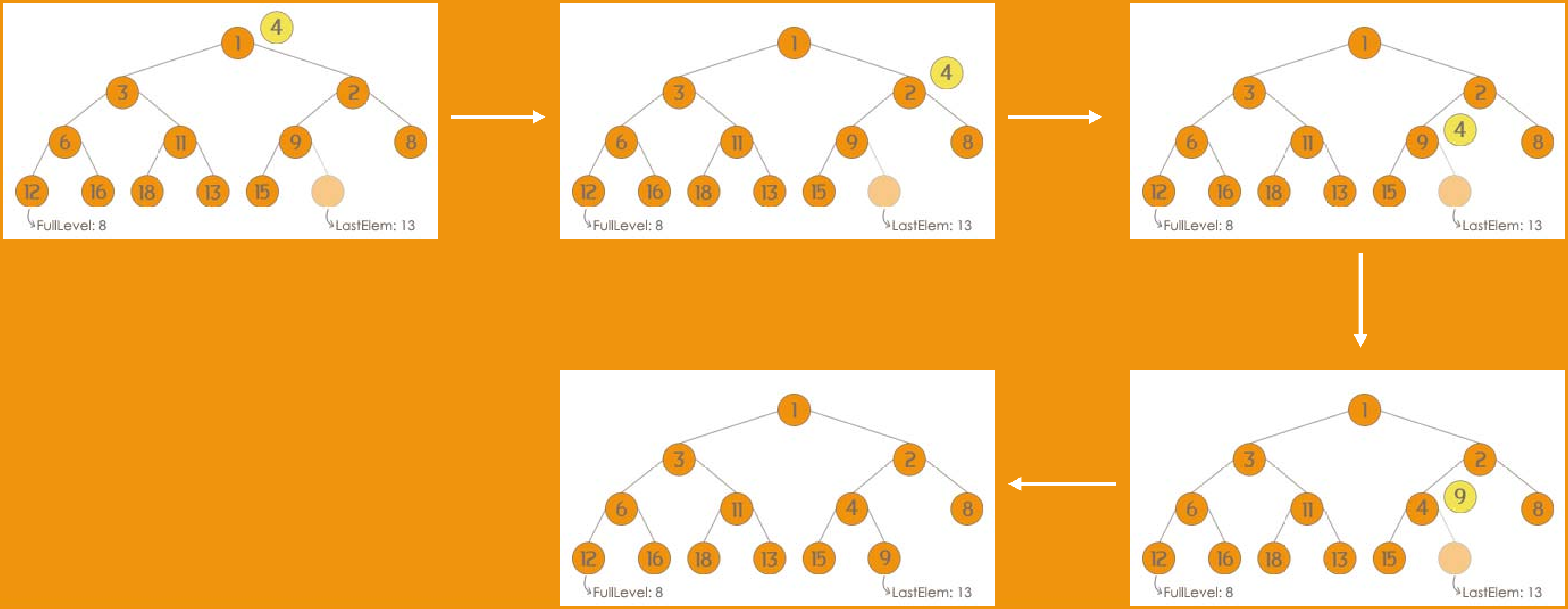
Top down insert can be done by predicting the path that a node will follow to it’s destination and then performing swaps as one proceeds down the path. Thus when the destination node is reach all the necessary swaps will have already been done in order to maintain the heap order property.

Predicting the path can be done by taking the difference between the index of the destination node and the index of the first node in the deepest level and then representing that as a binary number with the number of bits being equal to the height of the tree.

Reading this binary number from most significant bit to least significant bit and interpreting 1’s as a right turn and 0’s as a left turn one can identify the unique path to follow down to the destination node.

Modified Binary Heap

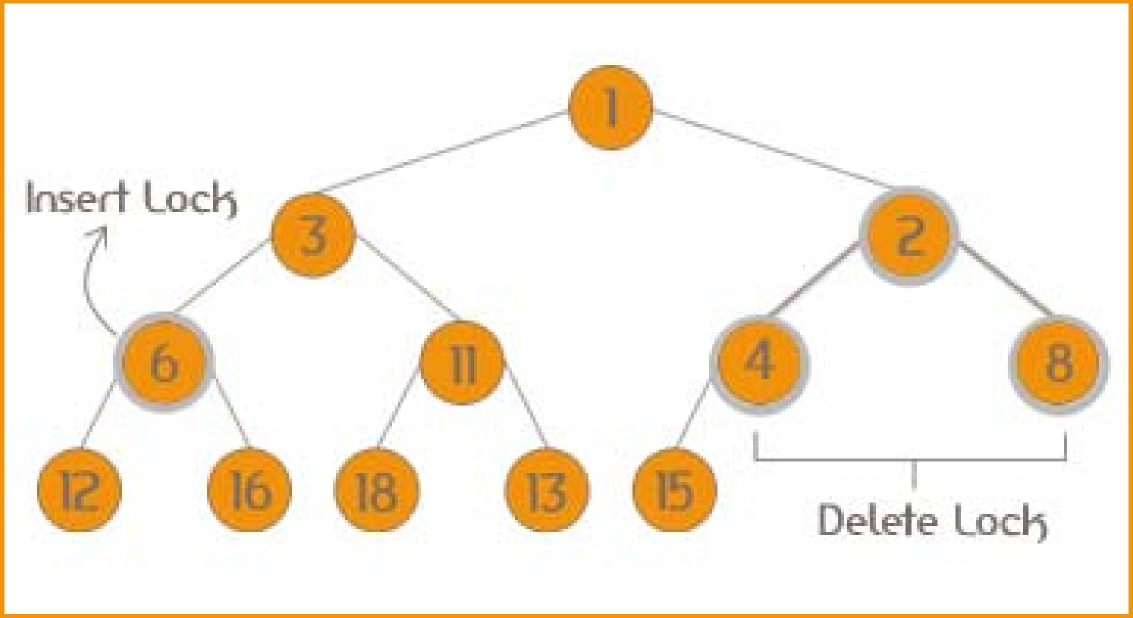
Difference = LastElem - FullLevel = 13 – 8 = 5
Height of Heap = 3
Binary Representation with 3 digits = 101
Path from Root: right $\rightarrow$ left $\rightarrow$ right



Concurrent
Binary Heap

With the top down insert procedure it now becomes possible to open the heap to multiple processes and lock it on a node by node basis.

- **Insert Processes** would lock only one node
- **Delete Processes** would lock 3 nodes: The parent and the 2 children



Concurrent

Binary Heap

The only problem left to address now is how to handle a situation where a delete process needs a node that another insert process is in the process of inserting.

The solution is to set a status field associated with each node that will be checked after each iteration on an insert to allow a “fast-forward” and minimize the wait time for a delete process.

Status	**Meaning**
PRESENT	A key exists at the node
PENDING	An insert is in progress which will ultimately insert a value to the node
WANTED	A delete process is waiting for the key
ABSENT	No key is present

Concurrent

Binary Heap

Other Considerations:

- Both the LastElement and FullLevel pointers also need to be locked for adjusting, this will be done by associating them with the root node since they need to be adjusted only once.
- This implementation assumes the Binary Heap is implemented via an array where each node can have an index mapped to it.

Sources:

[NK]

R.V. Nageshwara, V. Kumar. *Concurrent Access of Priority Queues*. IEEE Transactions on Computers, 37(12): 1657-1665, December 1988.

[CLRS]

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001.

[SGG]

Abraham Silberschatz, Greg Gagne, and Peter B. Galvin. *Operating System Concepts*, Seventh Edition. Wiley, 2005