Indian Institute of Information Technology Kottayam

Queues

BY

Arun Cyril Jose

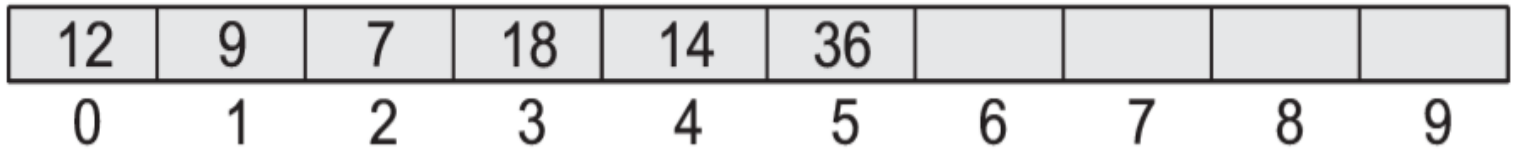
Queues

- Element inserted first is taken out first.
- Added at one end called the REAR.
- Removed from the other end called the FRONT.

- FIFO (First-In, First-Out).

- Implementation: Arrays or Linked lists.

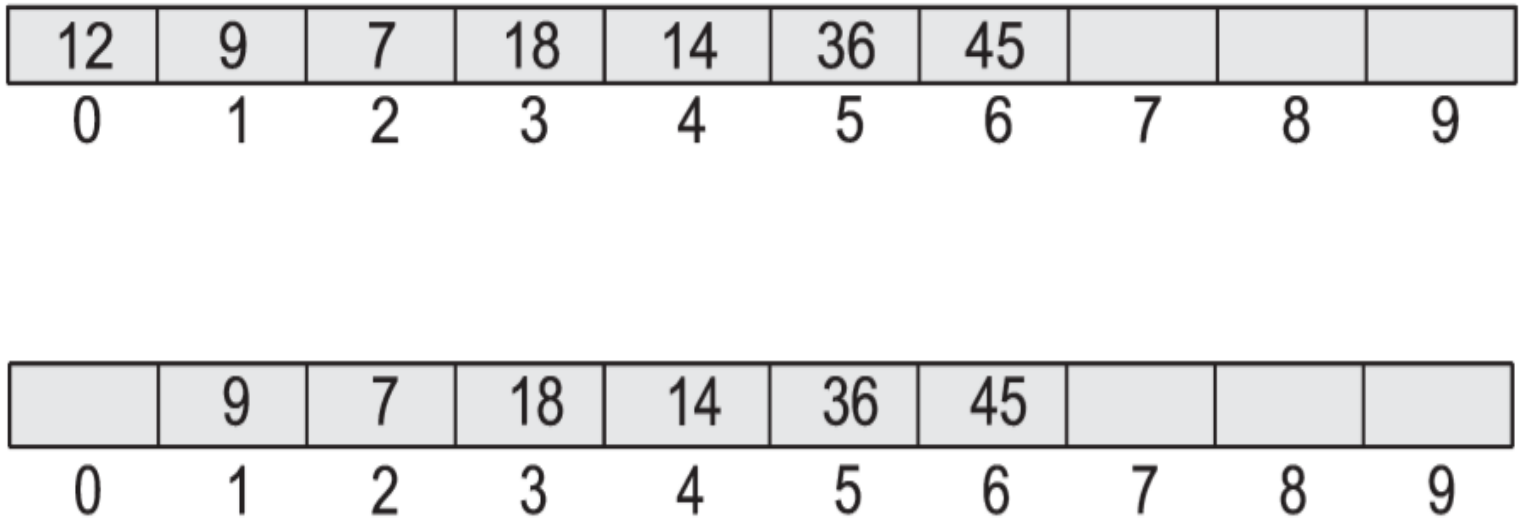
Queues: Insertion



Queues: Insertion

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

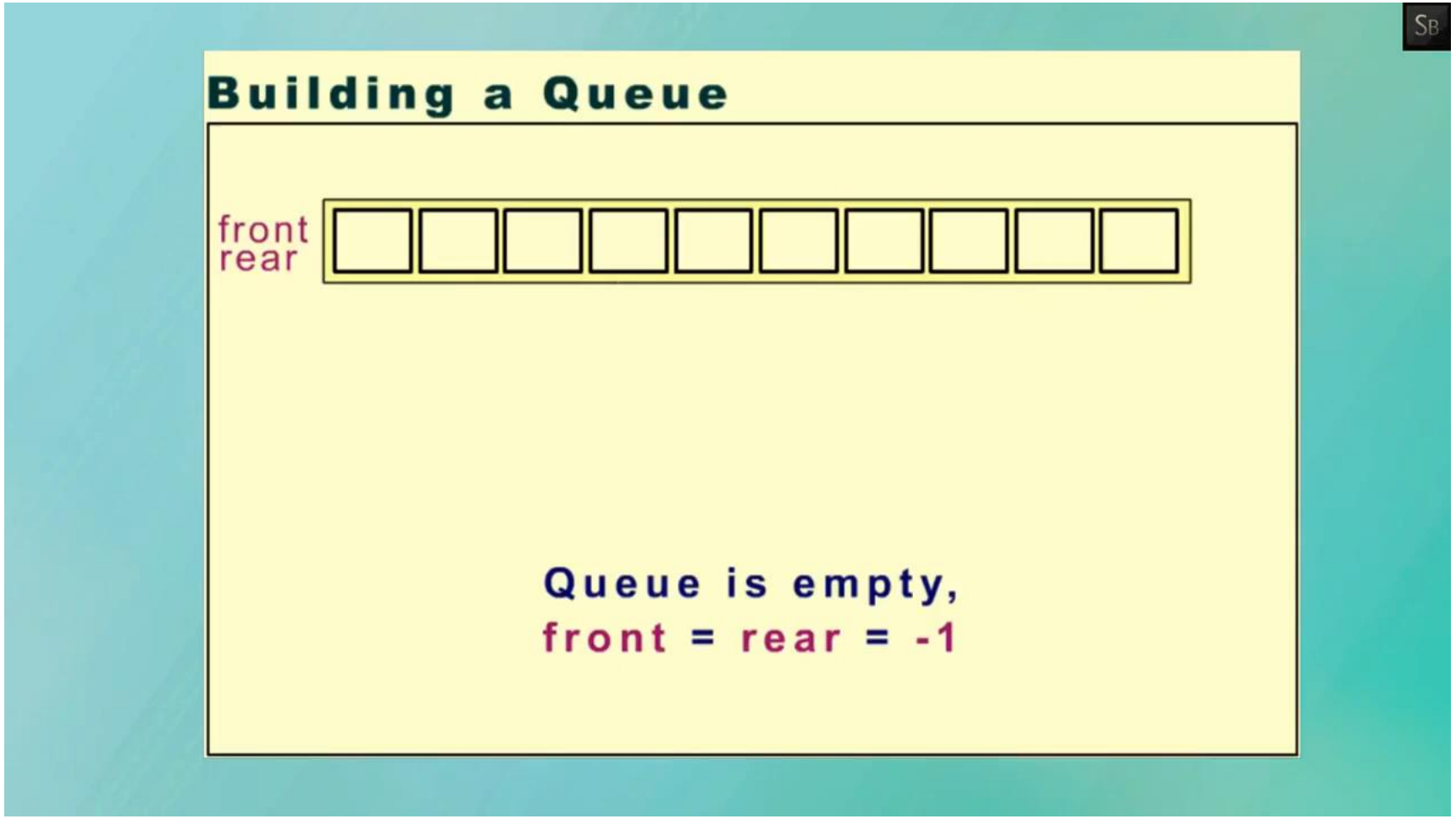
Queues: Deletion



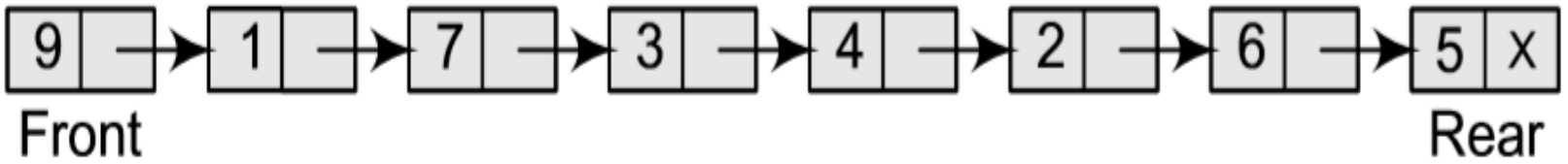
Queues: Deletion

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Array Implementation of Queues



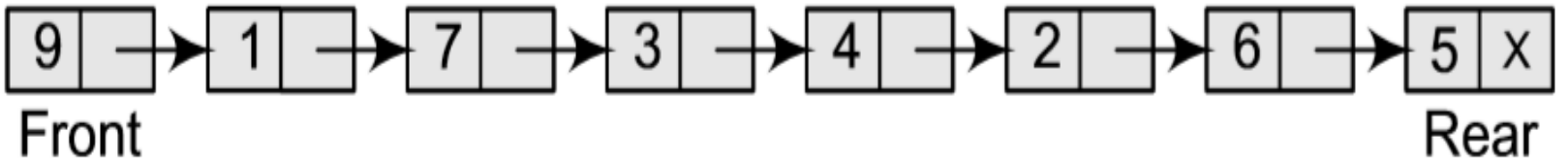
Queues: Insertion

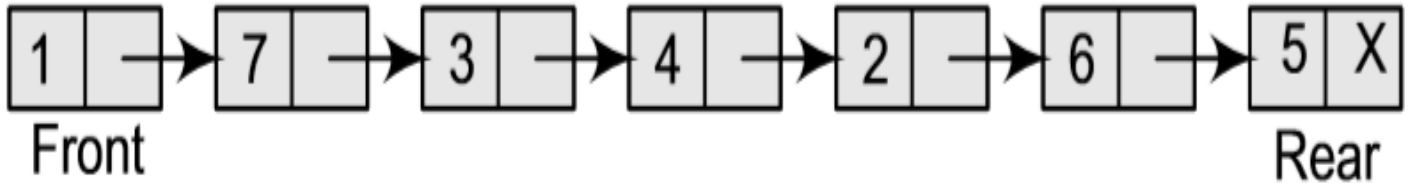


Queues: Insertion

```
Step 1: Allocate memory for the new node and name
        it as PTR
Step 2: SET PTR -> DATA = VAL
Step 3: IF FRONT = NULL
            SET FRONT = REAR = PTR
            SET FRONT -> NEXT = REAR -> NEXT = NULL
        ELSE
            SET REAR -> NEXT = PTR
            SET REAR = PTR
            SET REAR -> NEXT = NULL
        [END OF IF]
Step 4: END
```

Queues: Deletion





Queues: Deletion

```
Step 1: IF FRONT = NULL
            Write "Underflow"
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: SET FRONT = FRONT -> NEXT
Step 4: FREE PTR
Step 5: END
```

Types of Queues

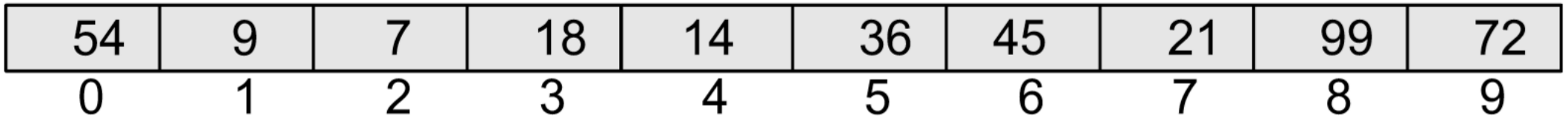
Multiple Queue

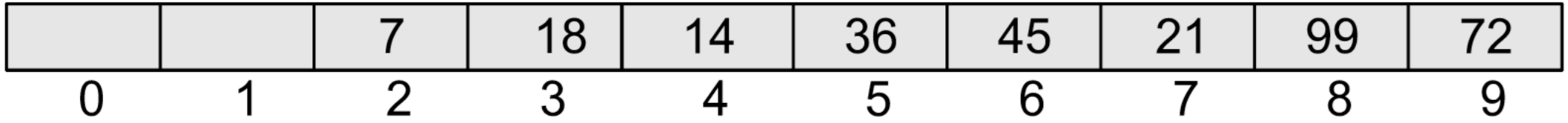
Circular Queue

Deque

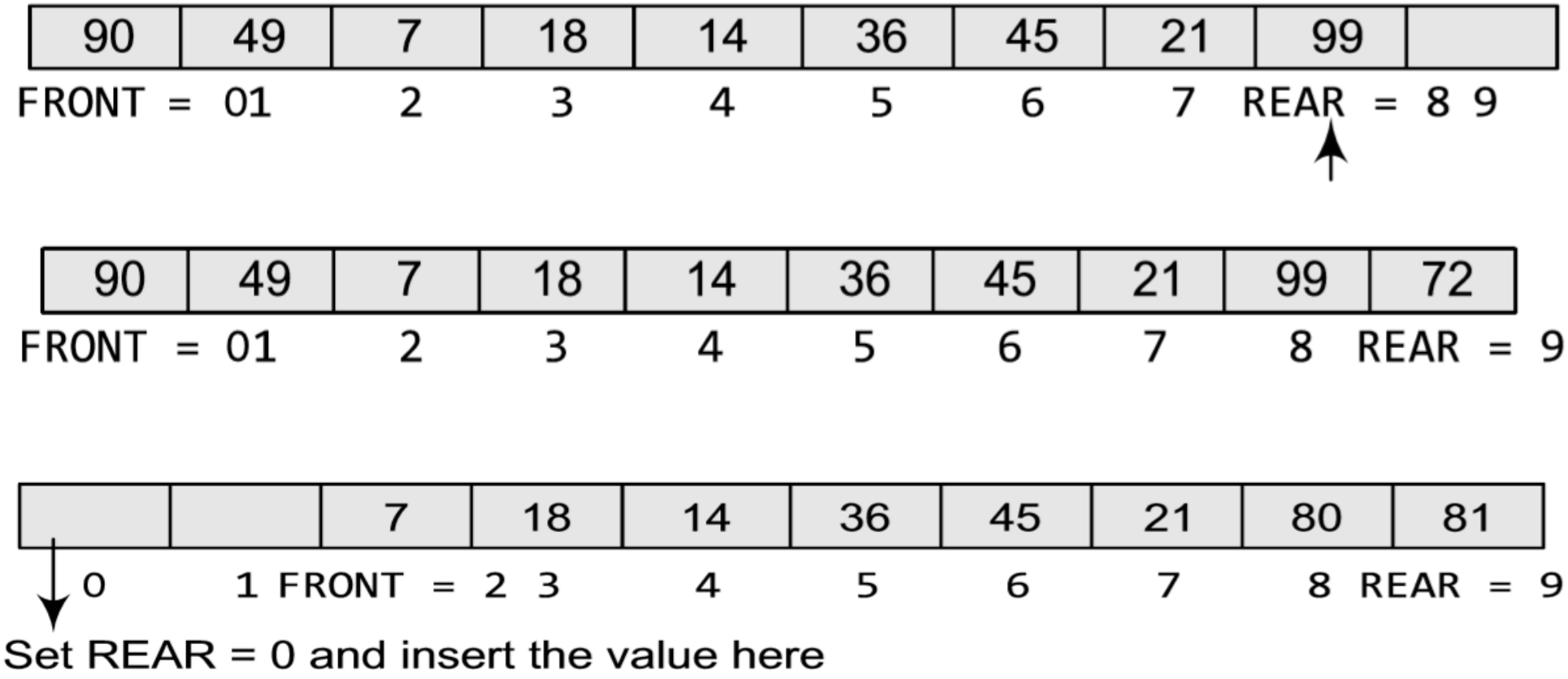
Priority Queue

Circular Queues





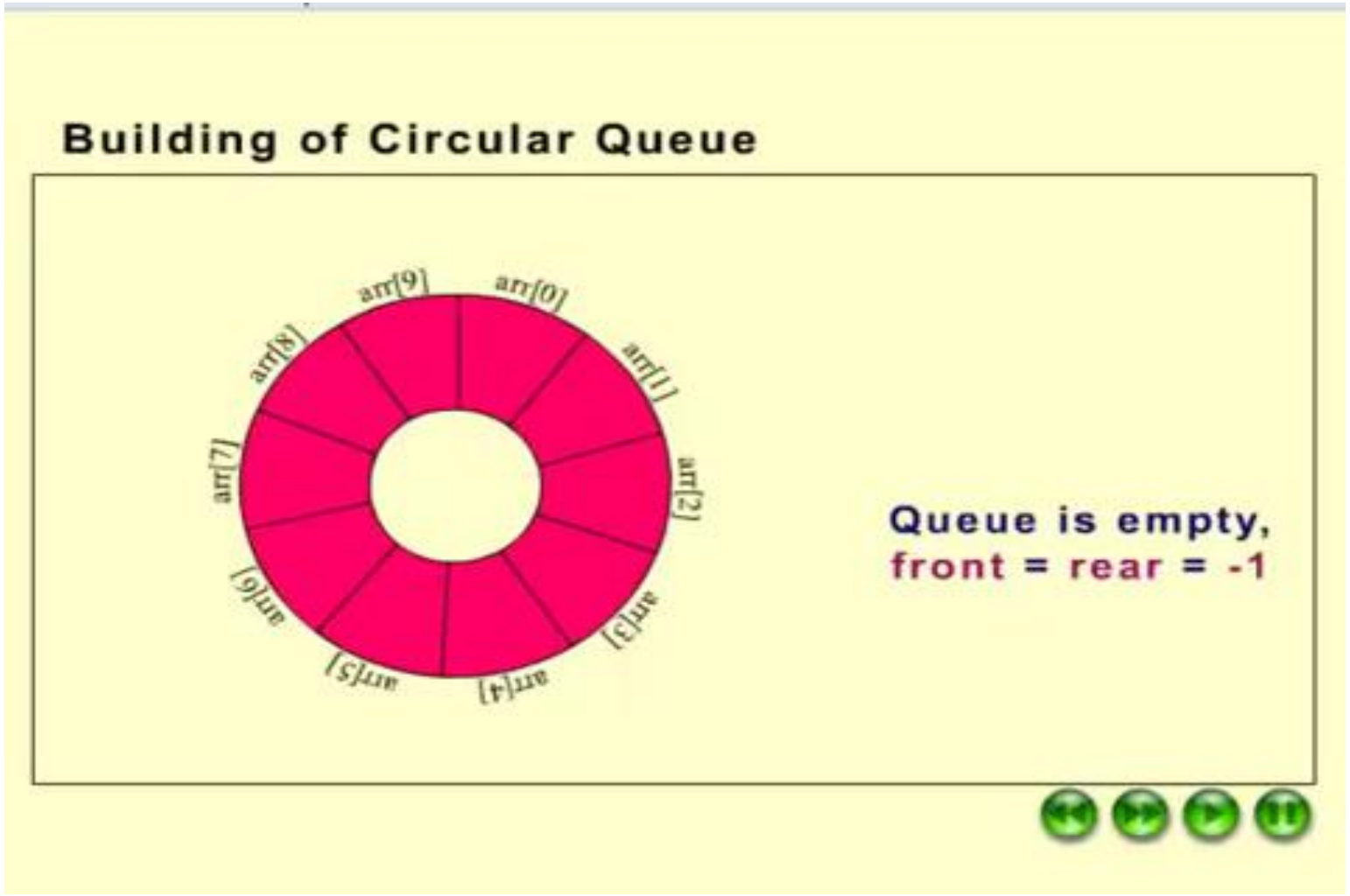
Circular Queues: Insertion



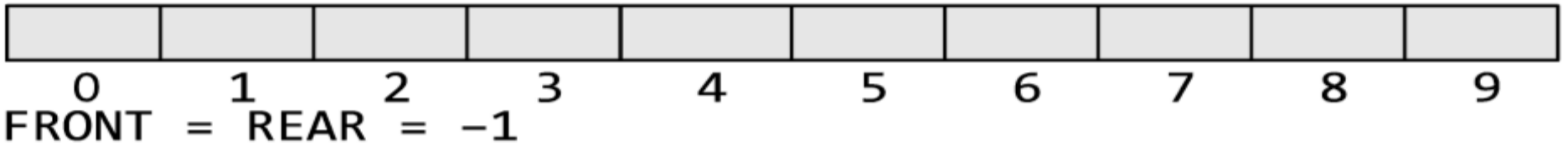
Circular Queues: Insertion

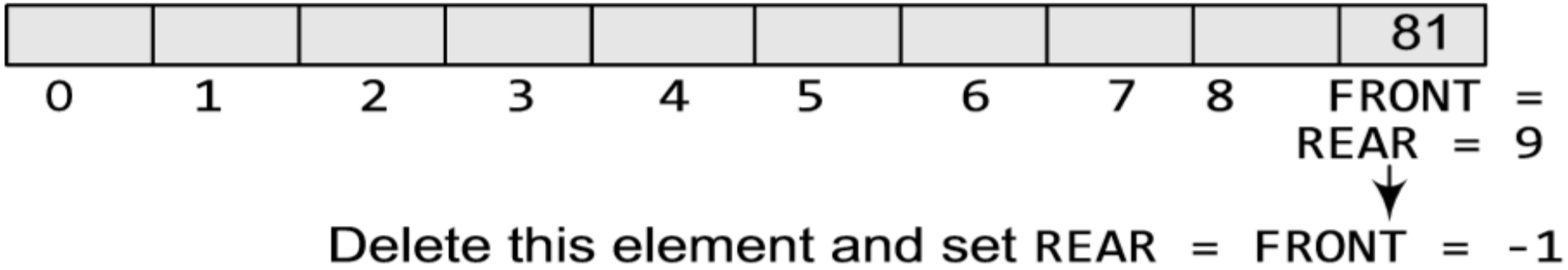
```
Step 1: IF (FRONT = 0 and Rear = MAX - 1) OR (Rear == FRONT-1)
            Write "OVERFLOW”
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Circular Queues: Operations

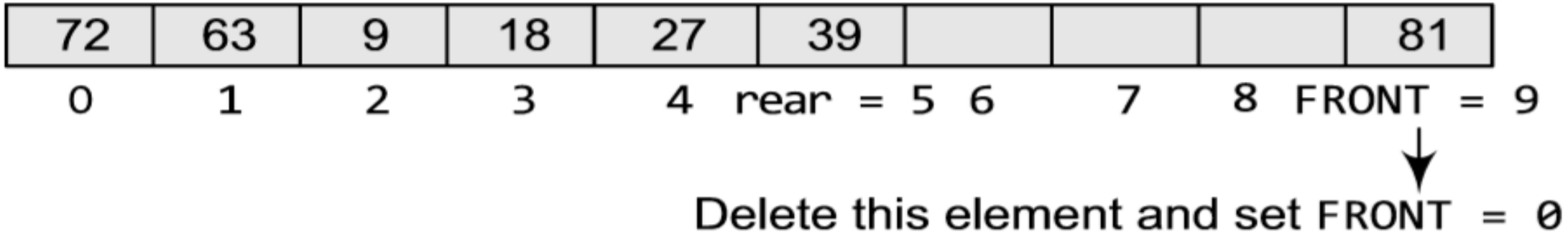


Circular Queues: Deletion





Circular Queues: Deletion

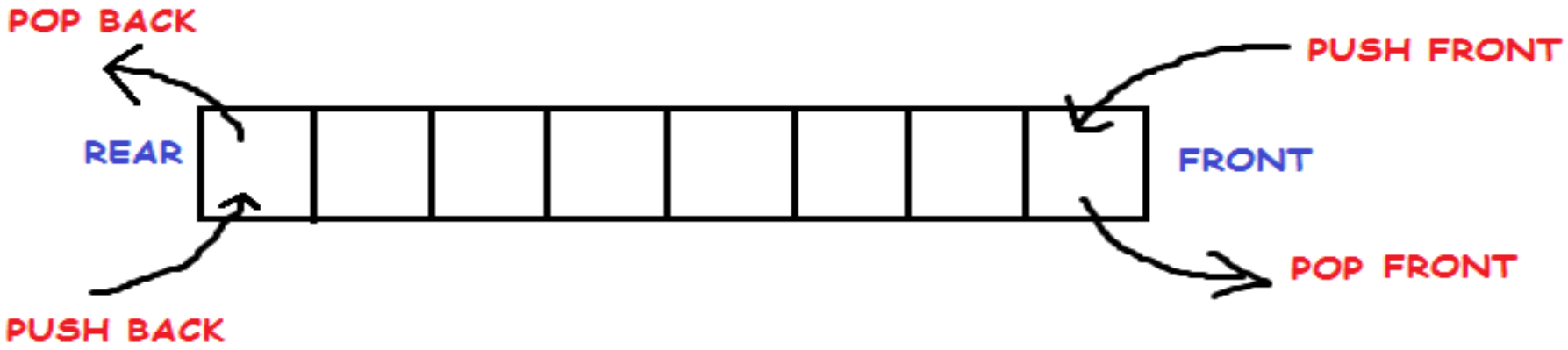


Circular Queues: Deletion

```
Step 1: IF FRONT = -1
            Write "UNDERFLOW"
            Goto Step 4
        [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                SET FRONT = 0
            ELSE
                SET FRONT = FRONT + 1
            [END of IF]
        [END OF IF]
Step 4: EXIT
```

Dequeues

- Elements can be inserted or deleted at either end (**double-ended queue**).
- Head-tail linked list.
- No element can be added and deleted from the middle.



Dequeues

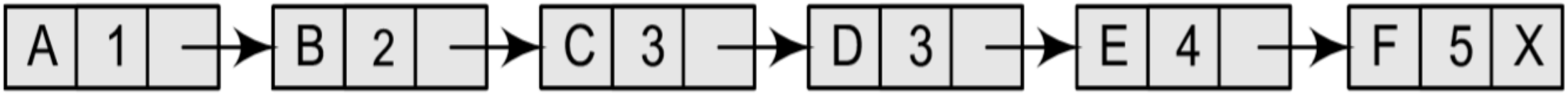
- Implementation:
  - Circular array
  - Circular Doubly Linked List.
- Maintains two pointers FRONT and REAR.
- **Input restricted deque**: insertions only at one of the ends, deletions can be from both ends.
- **Output restricted deque**: deletions only at one of the ends, insertions can be from both ends.

Priority Queues

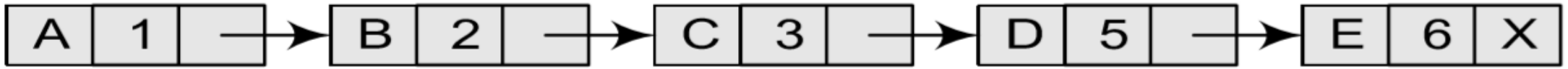
- Each element is assigned a priority.
- Priority of the element is used to determine the order in which the elements will be processed:
  - An element with higher priority is processed before an element with a lower priority.
  - Two elements with same priority are processed on **First-Come-First-Served basis**.

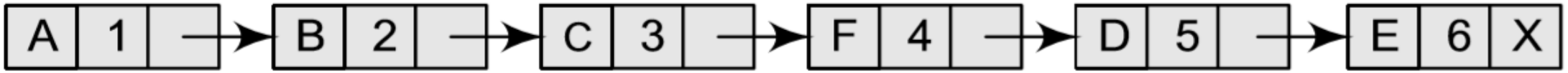
Priority Queues: Representation

- Arrays and Linked Lists.
- Linked List implementation
- A node has 3 parts:
  - Data, Priority number and Address of the next node.
- For **sorted linked list**, elements with the higher priority will precede the element with the lower priority.

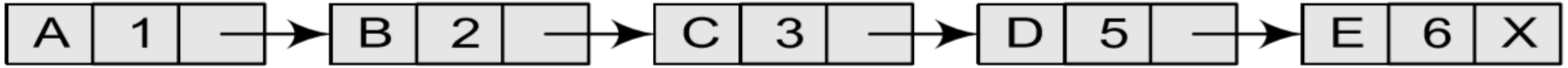


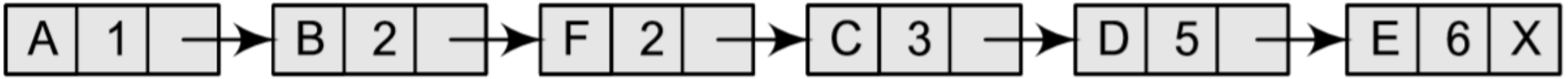
Priority Queues: Representation





Priority Queues: Representation





Priority Queues: Representation

- **Insertion** into sorted List.
- Traverse the entire list find a node with priority lower than the new element.

- New node is inserted before the node with lower priority.

- If there exists an element that has the same priority as the new element, the new element is inserted after that element.

Priority Queues: Representation

- Deletion from sorted List:

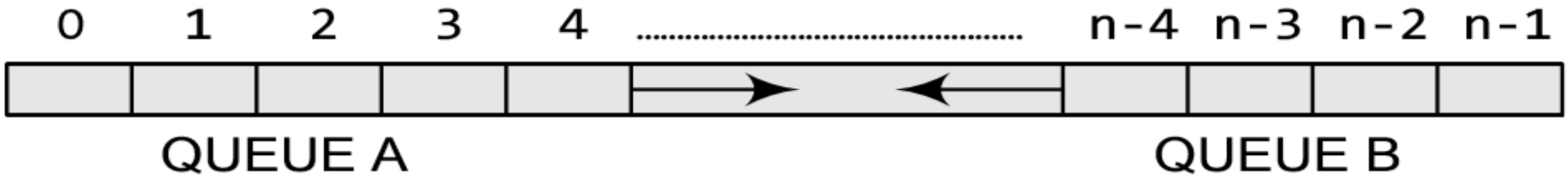
- First node of the list will be deleted.

Priority Queues: Implementation

- Sorted list
- Queue is sorted based on priority.
- Insertion Time Complexity: O(n)
- Deletion Time Complexity :
- Unsorted list
- New element is inserted at the end of the queue.
- Insertion Time Complexity:
- Deletion Time Complexity :

Multiple Queues

- In array implementation, size of the array is always a challenge.
- If queue size is too small, OVERFLOW will happen.
- If queue is too big, it is inefficient use of memory.



- Array QUEUE[n] is used to represent two queues, QUEUE A and QUEUE B.