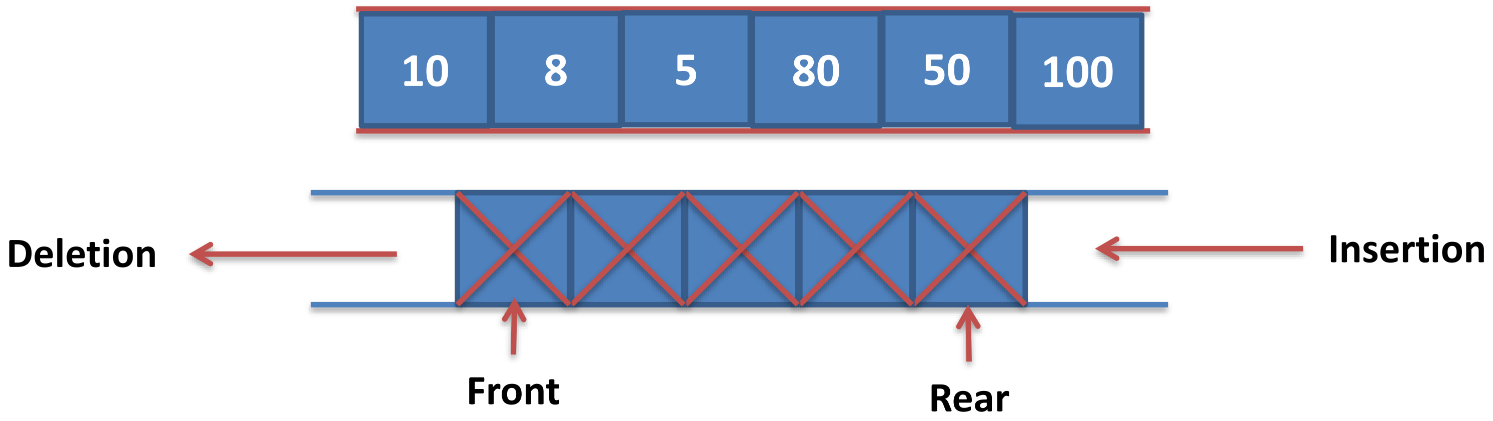
Queue

Queue

- A linear list which permits **deletion** to be performed **at one** end of the list and **insertion at the other end** is called **queue**.
- The information in such a list is processed **FIFO (first in first out) or FCFS (first come first served)** manner.
- **Front** is the end of queue from that deletion is to be performed.
- **Rear** is the end of queue at which new element is to be inserted.
- Insertion operation is called **Enqueue** & deletion operation is called **Dequeue**.



Applications of Queue

- Queue of people at any service point such as ticketing etc.
- Queue of air planes waiting for landing instructions.
- **Queue of processes** in OS.
- Queue is also used by Operating systems for **Job Scheduling**.
- When a **resource is shared** among multiple consumers. E.g., in case of printers the first one to be entered is the first to be processed.
- When **data is transferred asynchronously** (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
- Queue is used in **BFS (Breadth First Search)** algorithm. It helps in traversing a tree or graph.
- Queue is used in networking to **handle congestion**.

Queue Operations

- *Initialize* the queue
- *Insert* to the rear of the queue
- *Remove* (Delete) from the front of the queue
- Is the Queue Empty
- Is the Queue Full
- What is the size of the Queue

Procedure: Enqueue (Q, F, R, N,Y)

- This procedure inserts **Y** at rear end of Queue.
- **Queue** is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the front element of a queue.
- **R** is pointer to the rear element of a queue.

```
1. [Check for Queue Overflow]
       If      R >= N
       Then    write ('Queue Overflow')
               Return
2. [Increment REAR pointer]
       R ← R + 1
3. [Insert element]
       Q[R] ← Y
4. [Is front pointer properly set?]
      IF      F=0
      Then    F ← 1
      Return
```

Procedure: Enqueue (Q, F, R, N,Y)

```
1. [Check for Queue Overflow]
      If      R >= N
      Then    write ('Queue Overflow')
              Return
2. [Increment REAR pointer]
      R ← R + 1
3. [Insert element]
      Q[R] ← Y
4. [Is front pointer properly set?]
      IF      F=0
      Then    F ← 1
      Return
```

N=3, R=0, F=0

F = 1

R = 3

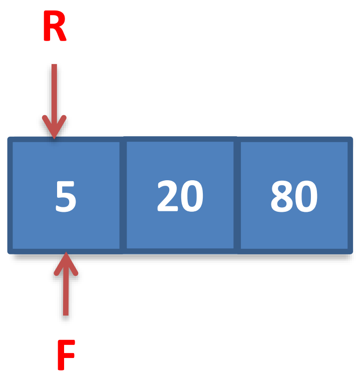
Enqueue (Q, F, R, N=3,Y=5)

Enqueue (Q, F, R, N=3,Y=20)

Enqueue (Q, F, R, N=3,Y=80)

Enqueue (Q, F, R, N=3,Y=3)

Queue Overflow



Function: Dequeue (Q, F, R)

- This function **deletes & returns** an element **from front end** of the Queue.
- **Queue** is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the **front** element of a queue.
- **R** is pointer to the **rear** element of a queue.

```
1. [Check for Queue Underflow]
    If    F = 0
    Then write ('Queue Underflow')
          Return(0)
2. [Delete element]
    Y ← Q[F]
```

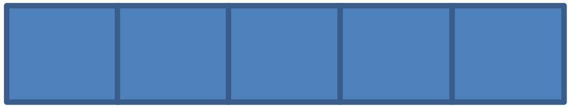
```
3. [Is Queue Empty?]
    If    F = R
    Then F ← R ← 0
    Else F ← F + 1
4. [Return Element]
    Return (Y)
```

Function: Dequeue (Q, F, R)

```
1. [Check for Queue Underflow]
     If    F = 0
     Then write (‘Queue Underflow’)
          Return(0)
2. [Delete element]
     Y ← Q[F]
3. [Is Queue Empty?]
     If    F = R
     Then F ← R ← 0
     Else F ← F + 1
4. [Return Element]
     Return (Y)
```

Case No 1:

F=0, R=0

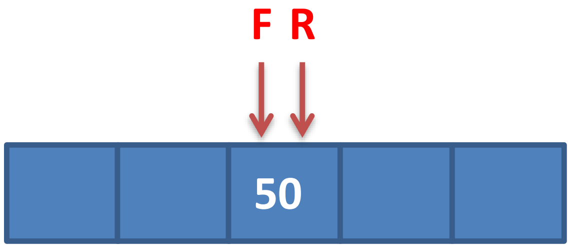


Queue Underflow

Case No 2:

F=3, R=3

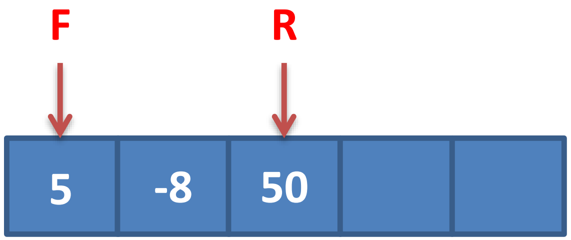
F=0, R=0



Case No 3:

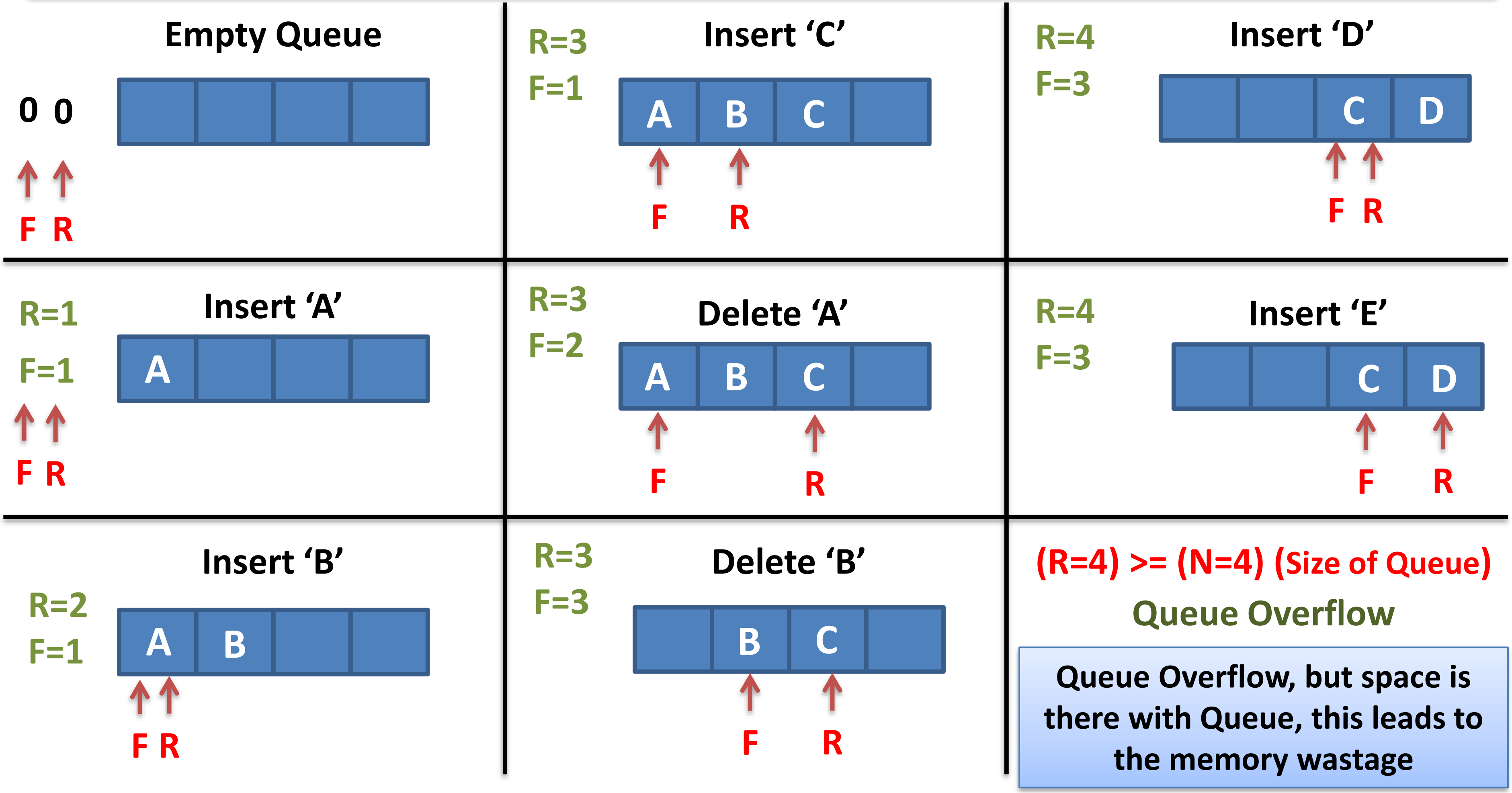
F=1, R=3

F=2, R=3



Example of Queue Insert / Delete

Perform following operations on queue with size 4 & draw queue after each operation
Insert ‘A’ | Insert ‘B’ | Insert ‘C’ | Delete ‘A’ | Delete ‘B’ | Insert ‘D’ | Insert ‘E’



Empty Queue

Insert ‘A’

Insert ‘B’

Insert ‘C’

Delete ‘A’

Delete ‘B’

Insert ‘D’

Insert ‘E’

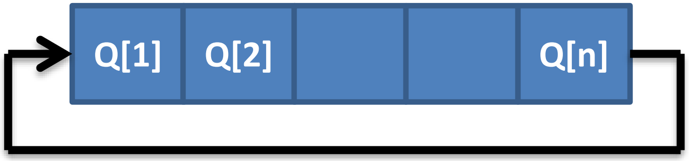
(R=4) >= (N=4) (Size of Queue)

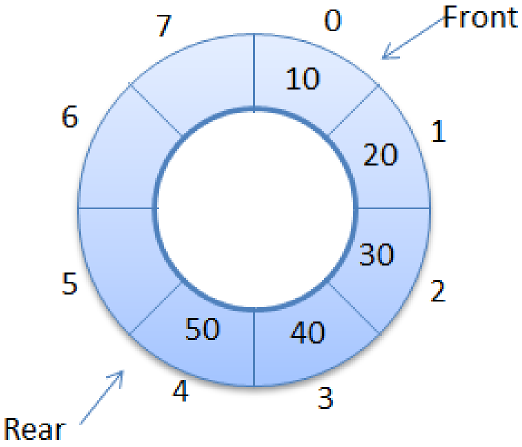
Queue Overflow

Queue Overflow, but space is there with Queue, this leads to the memory wastage

Circular Queue

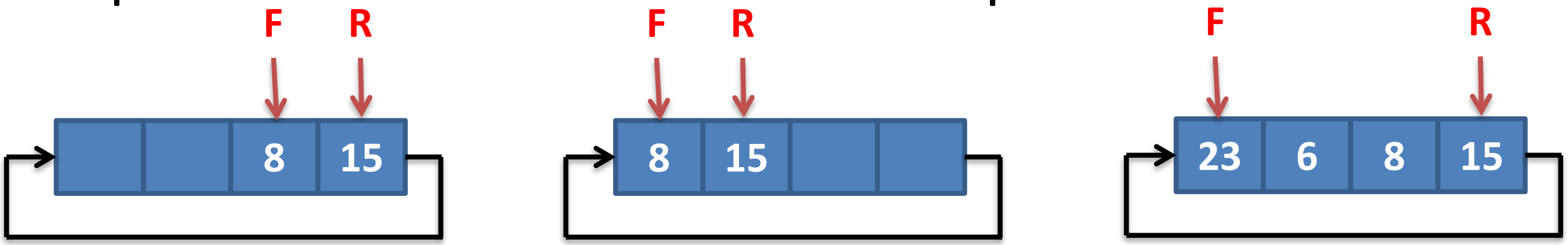
- A more suitable method of representing simple queue which prevents an excessive use of memory is to **arrange the elements** Q[1], Q[2]….,Q[n] **in a circular fashion** with Q[1] following Q[n], this is called **circular queue.**
- In circular queue the last node is connected back to the first node to make a circle.
- Circular queue is a linear data structure. It follows **FIFO** principle.
- It is also called as **"Ring buffer"**.





Procedure: CQINSERT (F, R, Q, N, Y)

- This procedure inserts **Y** at rear end of the Circular Queue.
- **Queue** is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the front element of a queue.
- **R** is pointer to the rear element of a queue.



```
1. [Reset Rear Pointer]
    If     R = N
    Then   R ← 1
    Else   R ← R + 1
2. [Overflow]
    If     F=R
    Then   Write(‘Overflow’)
           Return
```

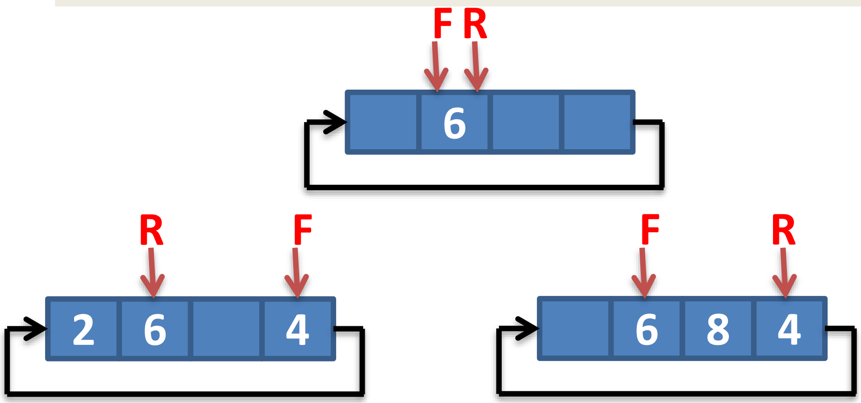
```
3. [Insert element]
           Q[R] ← Y
4. [Is front pointer
   properly set?]
           IF      F=0
           Then    F ← 1

           Return
```

Function: CQDELETE (F, R, Q, N)

- This function **deletes & returns** an element **from front end** of the Circular Queue.
- **Queue** is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the **front** element of a queue.
- **R** is pointer to the **rear** element of a queue.

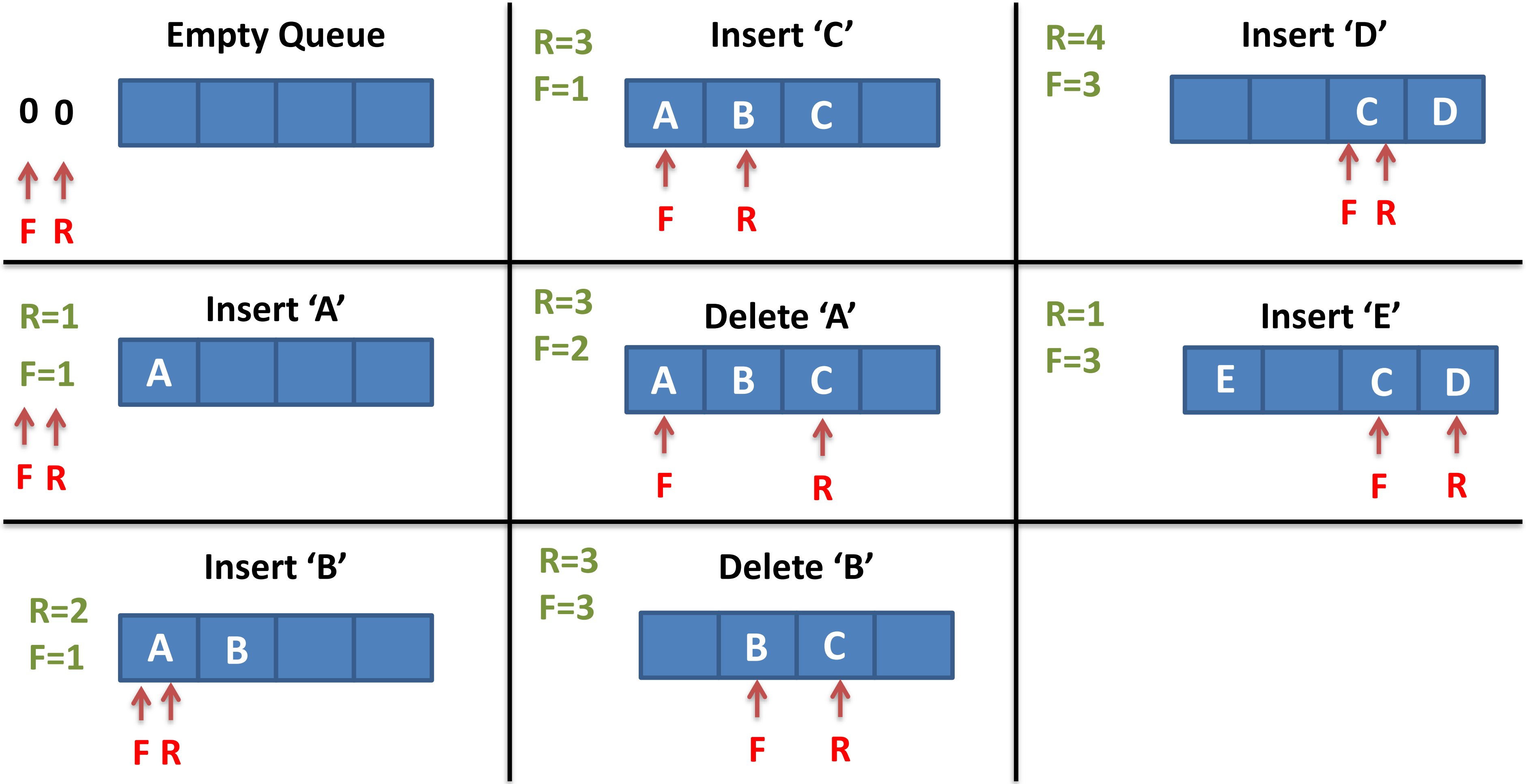
```
1. [Underflow?]
    If     F = 0
    Then   Write('Underflow')
           Return(0)
2. [Delete Element]
     Y ← Q[F]
3. [Queue Empty?]
    If     F = R
    Then   F ← R ←  0
           Return(Y)
```

```
4. Increment Front Pointer]
      IF F = N
      Then F ← 1
      Else F ← F + 1
      Return(Y)
```



Example of CQueue Insert / Delete

Perform following operations on Circular queue with size 4 & draw queue after each operation Insert 'A' | Insert 'B' | Insert 'C' | Delete 'A' | Delete 'B' | Insert 'D' | Insert 'E'



Empty Queue

0 0

F R

Insert 'A'

R=1

F=1

A

F R

Insert 'B'

R=2

F=1

A B

F R

Insert 'C'

R=3

F=1

A B C

F R

Delete 'A'

R=3

F=2

A B C

F R

Delete 'B'

R=3

F=3

B C

F R

Insert 'D'

R=4

F=3

C D

F R

Insert 'E'

R=1

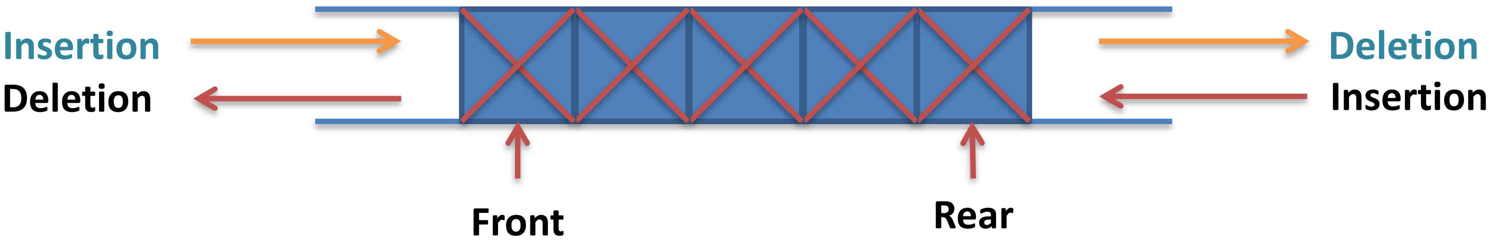
F=3

E C D

F R

DQueue

- A **DQueue (double ended queue)** is a linear list in which insertion and deletion are performed **from the either end of the structure**.
- There are two variations of Dqueue
  - ***Input restricted dqueue***- allows insertion at only one end
  - ***Output restricted dqueue***- allows deletion from only one end



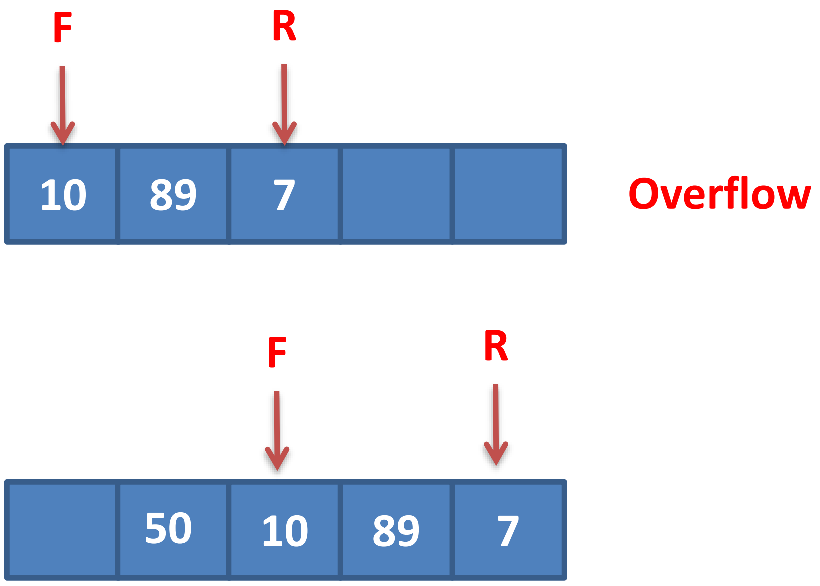
DQueue Algorithms

- DQINSERT_REAR is same as QINSERT (Enqueue)
- DQDELETE_FRONT is same as QDELETE (Dequeue)
- DQINSERT_FRONT
- DQDELETE_REAR

Procedure: DQINSERT_FRONT (Q,F,R,N,Y)

- This procedure **inserts Y** at **front** end of the Circular Queue.
- Queue is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the **front** element of a queue.
- **R** is pointer to the **rear** element of a queue.

```
1. [Overflow?]
   If     F = 0
   Then   Write(‘Empty’)
          Return
   If     F = 1
   Then   Write(‘Overflow’)
          Return
2. [Decrement front Pointer]
    F ← F - 1
3. [Insert Element?]
   Q[F] ← Y
   Return
```



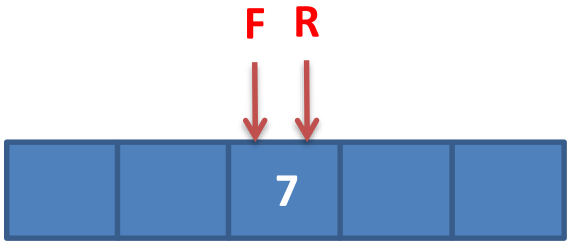
10	89	7		

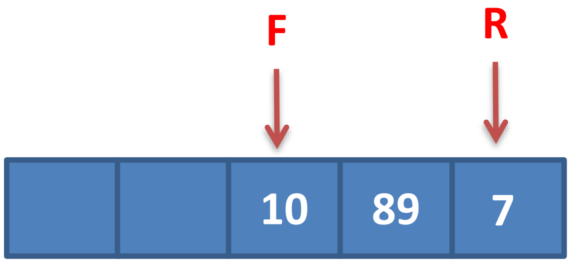
	50	10	89	7

Function: DQDELETE_REAR(Q,F,R)

- This function **deletes & returns** an element from **rear end** of the Queue.
- **Queue** is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the **front** element of a queue.
- **R** is pointer to the **rear** element of a queue.

```
1. [Underflow?]
    If     R = 0
    Then   Write('Underflow')
           Return(0)
2. [Delete Element]
     Y ← Q[R]
3. [Queue Empty?]
    IF    R = F
    Then R ← F ← 0
    Else R ← R – 1
4. [Return Element]
    Return(Y)
```

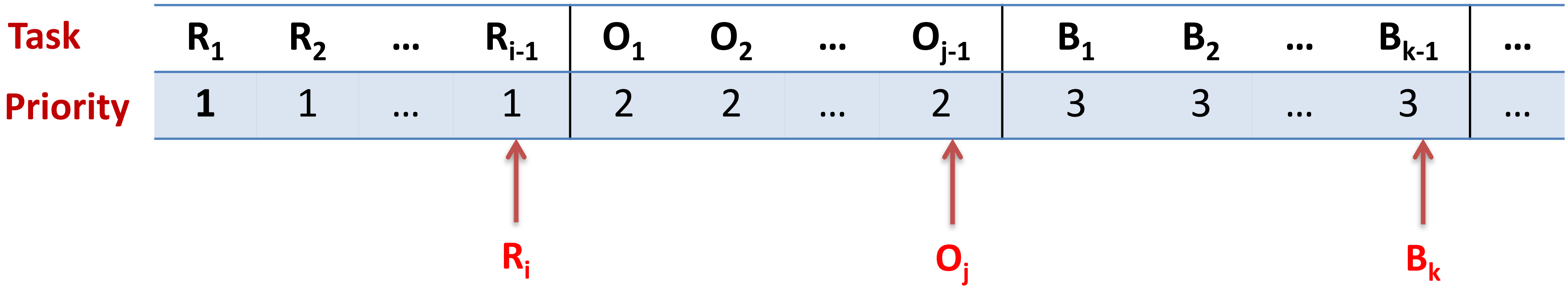




Priority Queue

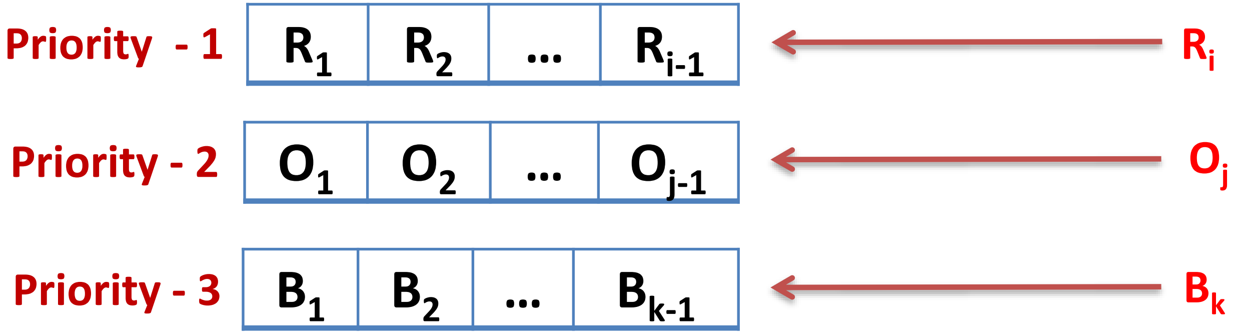
- A queue in which we are able to **insert & remove items** from **any position based on** some property (such as **priority** of the task to be processed) is often referred as **priority queue**.
- Below fig. represent a priority queue of jobs waiting to use a computer.
- Priorities are attached with each Job
  - **Priority 1** indicates **Real Time Job**
  - **Priority 2** indicates **Online Job**
  - **Priority 3** indicates **Batch Processing Job**
- Therefore if a job is initiated with priority i, it is inserted immediately at the end of list of other jobs with priorities i.
- Here jobs are always removed from the front of queue

Priority Queue Cont…



Task	R_1	R_2	…	R_{i-1}	O_1	O_2	…	O_{j-1}	B_1	B_2	…	B_{k-1}	…
Priority	1	1	…	1	2	2	…	2	3	3	…	3	…

Priority Queue viewed as a single queue with insertion allowed at any position



Priority - 1	R_1	R_2	…	R_{i-1}
Priority - 2	O_1	O_2	…	O_{j-1}
Priority - 3	B_1	B_2	…	B_{k-1}

Priority Queue viewed as a Viewed as a set of queue

Priority Queue

- It is like the “normal” queue except that the dequeuing elements follow a priority order.
- The priority order dequeues those items first that have the highest priority
- The element order in a priority queue depends on the element’s priority in that queue
- Each item has some priority associated with it.
- An item with the highest priority is moved at the front and deleted first.
- If two elements share the same priority value, then the priority queue follows the first-in-first-out principle for de queue operation.