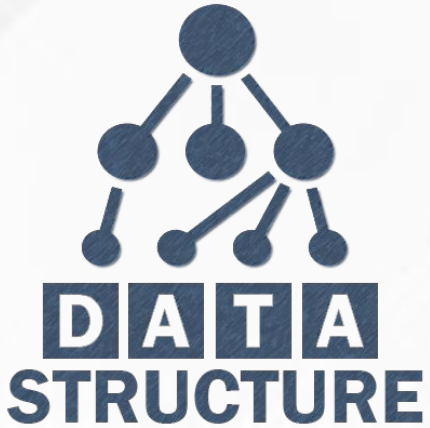
Data Structures (DS)
GTU # 3130702

Unit-2

Linear Data Structure (Queue)



COMPUTER ENGINEERING DEPARTMENT

Queue

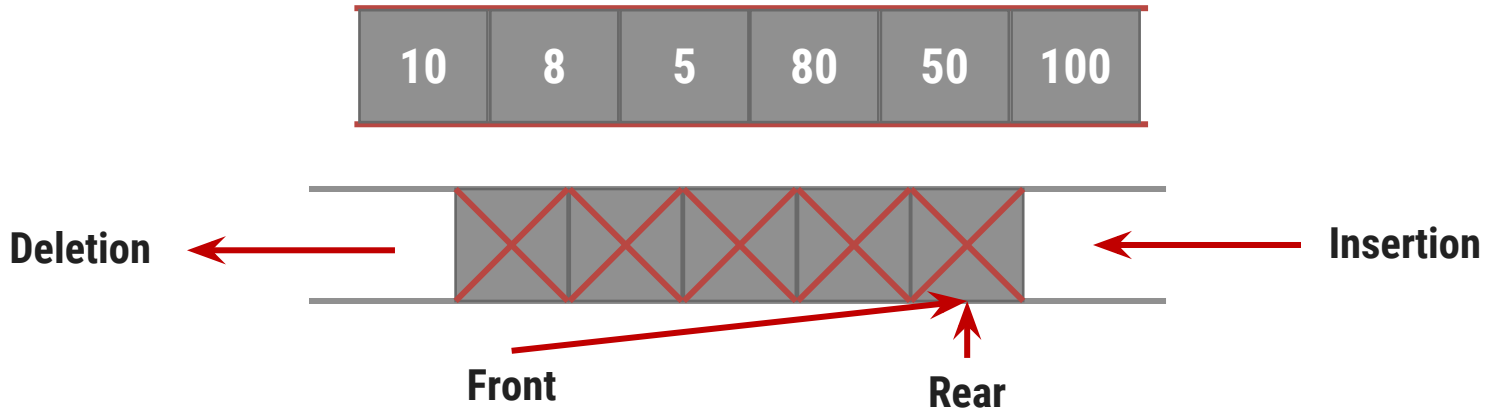
- Imagine a line of people waiting at a ticket counter. People are **served in the order they come**, that is, people who **come first are served first** whereas people who come later are served after that.
- Let the action of someone joining a queue be called **enqueue** and someone being serve and getting out of the queue be called **dequeue**.
- A type of structure, similar to the example of the line of people, can be represented as a data structure. Such a data structure is known as a queue.

Types of Queue

- **SIMPLE QUEUE** :
  - A simple queue is a type of queue where **insertion is at the end** of the queue and **removal is at the front**.
- **CIRCULAR QUEUE** :
  - A circular queue is a type of queue where the **last element is connected to the first**. An element is added to the end of the queue and removed from the front.
- **PRIORITY QUEUE** :
  - A priority queue is a type of queue where the **elements are arranged in some priority order**. An element with the highest priority is removed first and insertion occurs according to the priority.
- **DOUBLE ENDED QUEUE** :
  - A double ended queue is a type of queue where **insertion and deletion can happen at both ends of the 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**.

Simple Queue

- **Linear** Data Structure
- Perform Operation on **two ends**.
- Can be **implemented** using **Array** or **Linked List**.
- The simple queue is a **normal queue**
- **insertion** takes place at the **REAR END** of the queue - **enqueue**
- **deletion** takes place at the **FRONT END** of the queue – **dequeue**
- follows **FIFO – First In First Out** **or** **FCFS – First Come First In** principle.
- It is also called as **“Buffer”**.

Simple Queue

enqueue() - QINSERT

1. Check if the queue is full.
2. If the queue is full, then print **"Queue overflow"**.
3. Else increment **REAR** by 1.
4. Assign QUEUE [REAR] = ELEMENT.

dequeue() - QDELETE

1. Check if the queue is empty.
2. If the queue is empty, the print **"Queue underflow"**.
3. Else Copy the element at the front of the queue to some temporary variable, TEMP = QUEUE[FRONT].
4. Increment **FRONT** by 1.
5. Print temp and delete it.

Example of Simple 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

0 0

F R

Insert ‘A’

R=1
F=1

F R

Insert ‘B’

R=2
F=1

F R

Insert ‘C’

R=3
F=1

F R

Delete ‘A’

R=3
F=2

F R

Delete ‘B’

R=3
F=3

F R

Insert ‘D’

R=4
F=3

F R

Insert ‘E’

R=4
F=3

F R

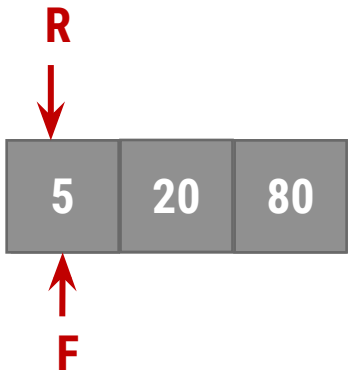
(R=4) >= (N=4) (Size of Queue)

Queue Overflow

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

Procedure: QINSERT(Q, F, R, N,Y) – Simple Queue [Enqueue]

- 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
```

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)

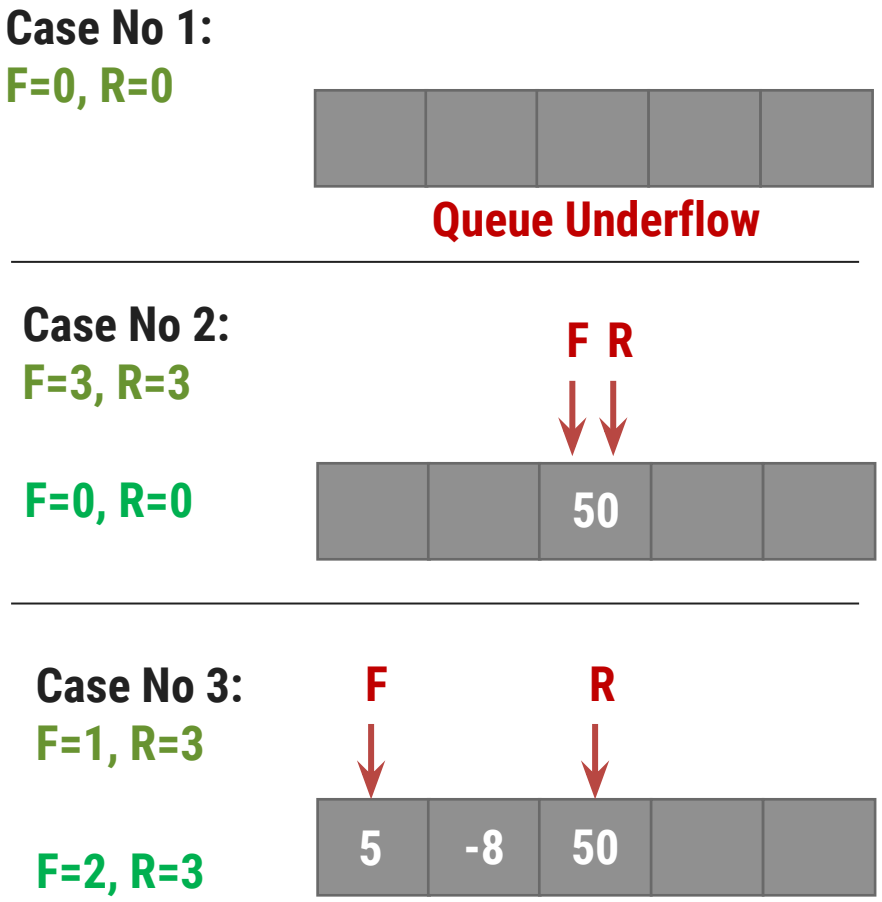
Enqueue (Q, F, R, N=3,Y=3)

Queue Overflow

Function: QDELETE(Q, F, R) – Simple Queue [Dequeue]

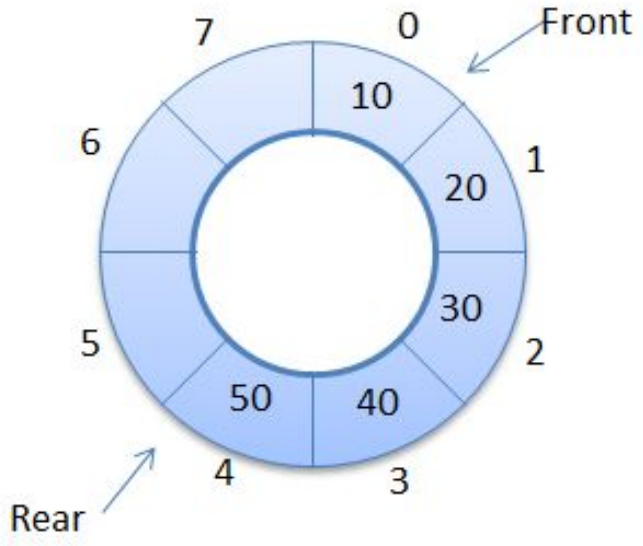
- This function **deletes & returns** an element **from front end** of the Queue.
- **Queue** is represented by a vector **Q** containing **N** elements.
- **F** is pointer to the **front** element of a queue.
- **R** is pointer to the **rear** element of a queue.

```
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)
```



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”**.



Example of CQueue Insert / Delete

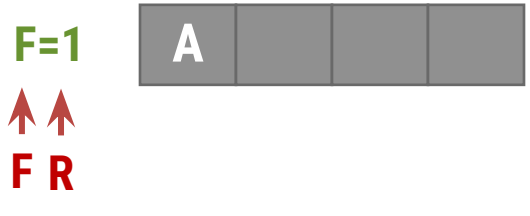
Perform following operations on Circular queue with size 4 & draw queue after each operation
Insert ‘A’ | Insert ‘B’ | Insert ‘C’ | Delete ‘A’ | Delete ‘B’ | Insert ‘D’ | Insert ‘E’

Empty Queue

0 0

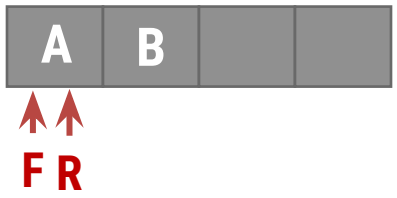
Insert ‘A’

R=1
F=1



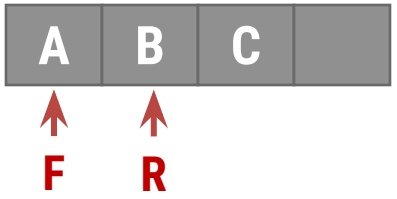
Insert ‘B’

R=2
F=1



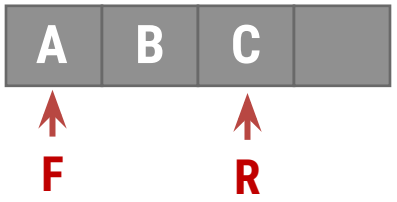
Insert ‘C’

R=3
F=1



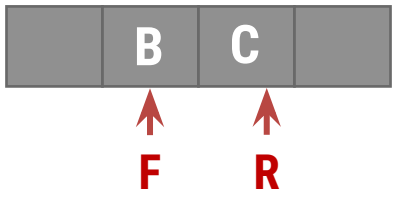
Delete ‘A’

R=3
F=2



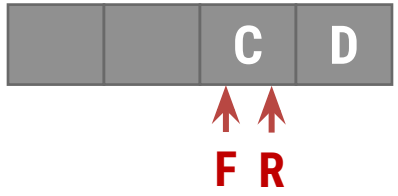
Delete ‘B’

R=3
F=3



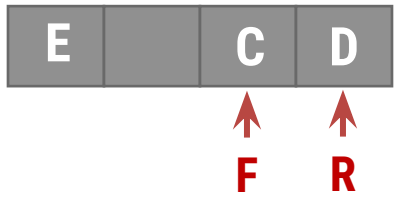
Insert ‘D’

R=4
F=3



Insert ‘E’

R=1
F=3

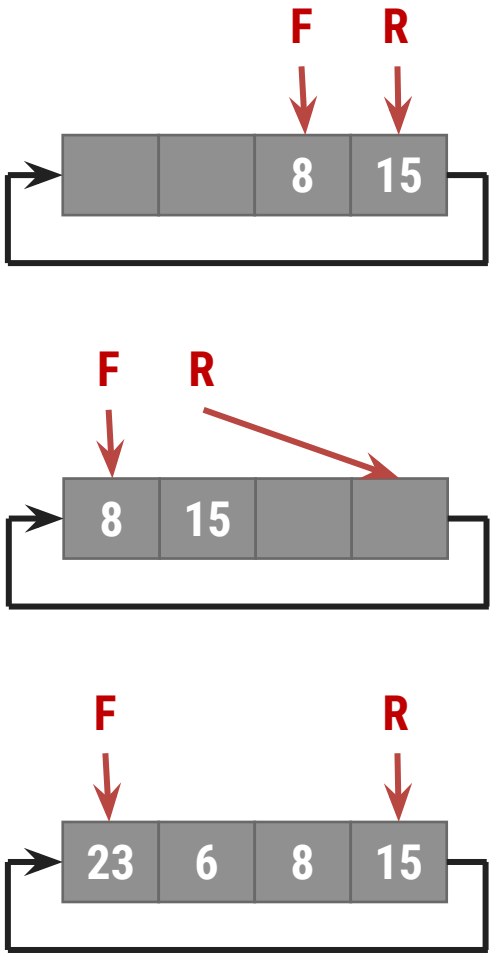


Procedure: CQINSERT (F, R, Q, N, Y)

- 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
```

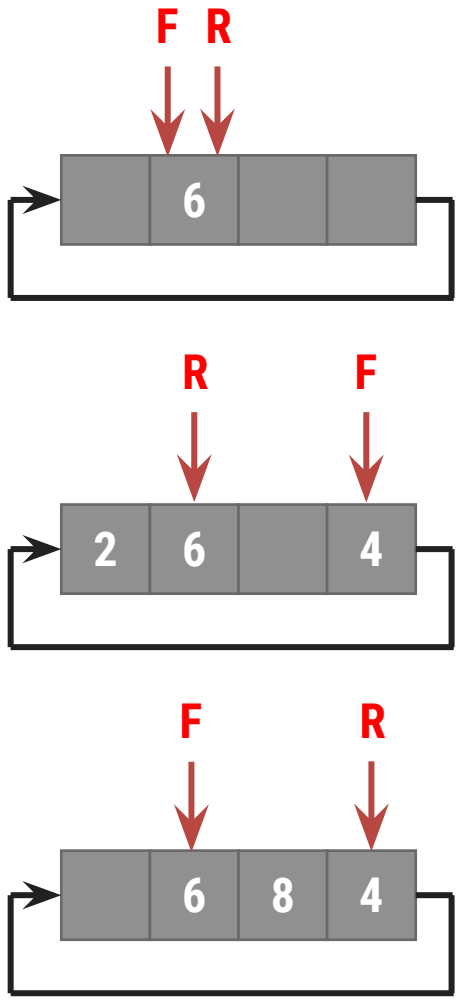


Function: CQDELETE (F, R, Q, N)

- 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.

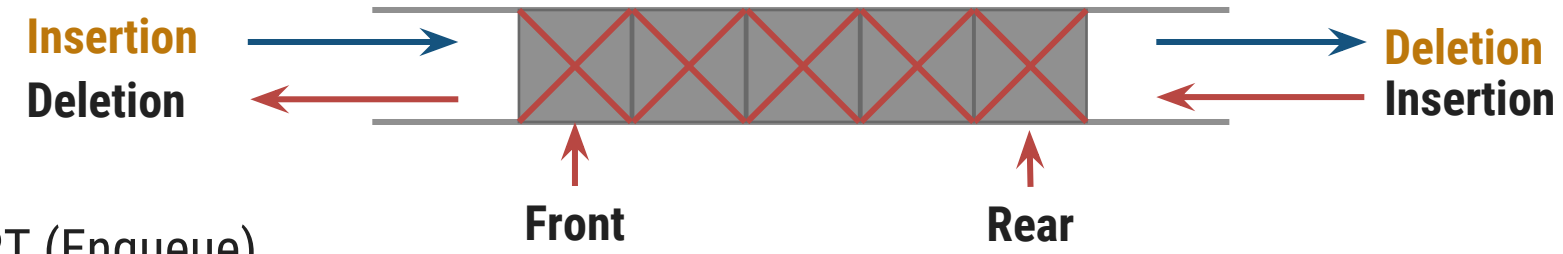
```
1. [Underflow?]
   If     F = 0
   Then   Write('Underflow')
          Return(0)
2. [Delete Element]
    Y ← Q[F]
3. [Queue Empty?]
   If     F = R
   Then   F ← R ← 0
          Return(Y)
```

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



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

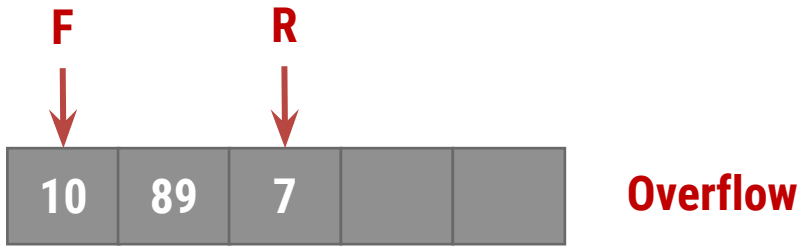


- Dqueue Algorithms
  - DQINSERT_REAR is same as QINSERT (Enqueue)
  - DQDELETE_FRONT is same as QDELETE (Dequeue)
  - DQINSERT_FRONT
  - DQDELETE_REAR

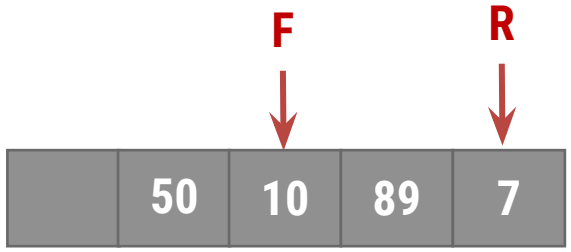
Procedure: DQINSERT_FRONT (Q,F,R,N,Y)

- 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
```



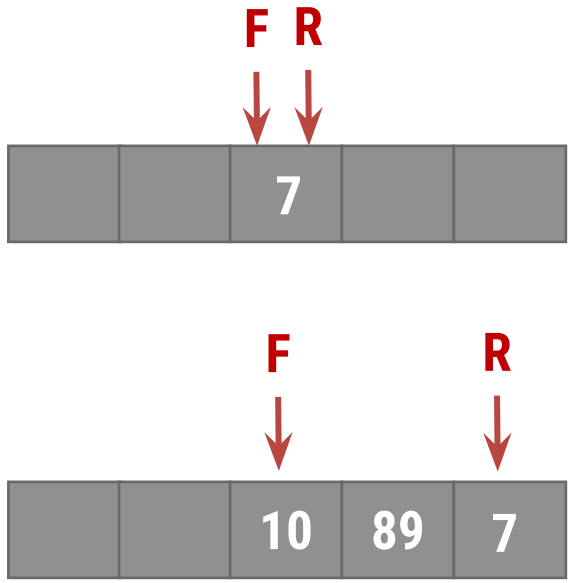
Overflow



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.

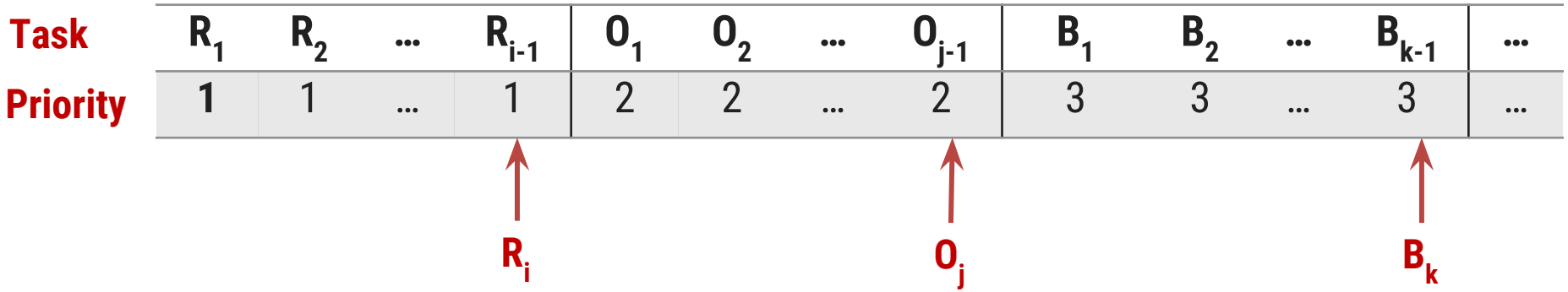
```
1. [Underflow?]
   If     R = 0
   Then   Write('Underflow')
          Return(0)
2. [Delete Element]
    Y ← Q[R]
3. [Queue Empty?]
   IF    R = F
   Then R ← F ← 0
   Else R ← R - 1
4. [Return Element]
   Return(Y)
```



Priority Queue

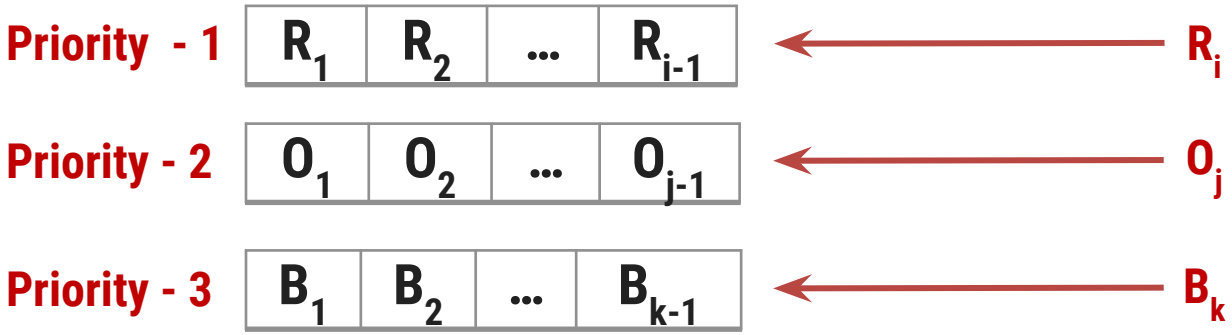
- A queue in which we are able to **insert & remove items** from **any position based on** some property (such as **priority** of the task to be processed) is often referred as **priority queue**.
- Below fig. represent a priority queue of jobs waiting to use a computer.
- Priorities are attached with 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…



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

---



Priority Queue viewed as a Viewed as a set of queue

Thank You