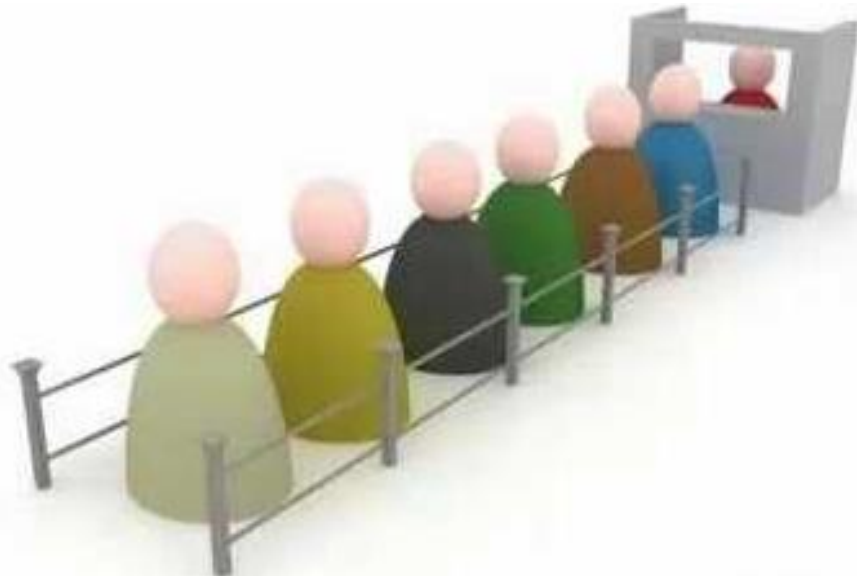


# Queue : Data Structure

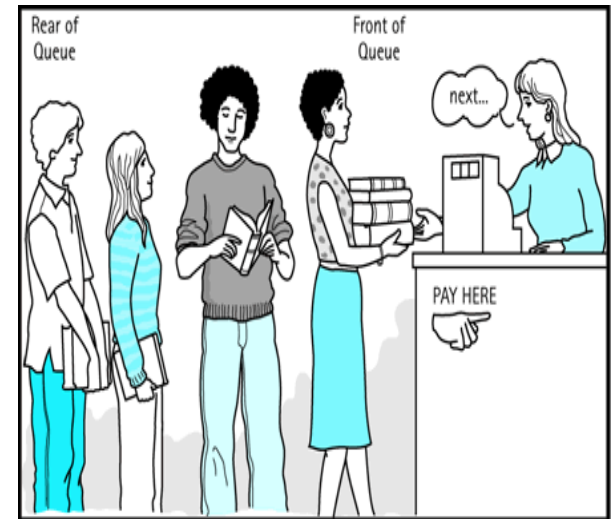


# Applications of Queue

single-lane one  
way road  
where the  
vehicle enters  
first, exits first

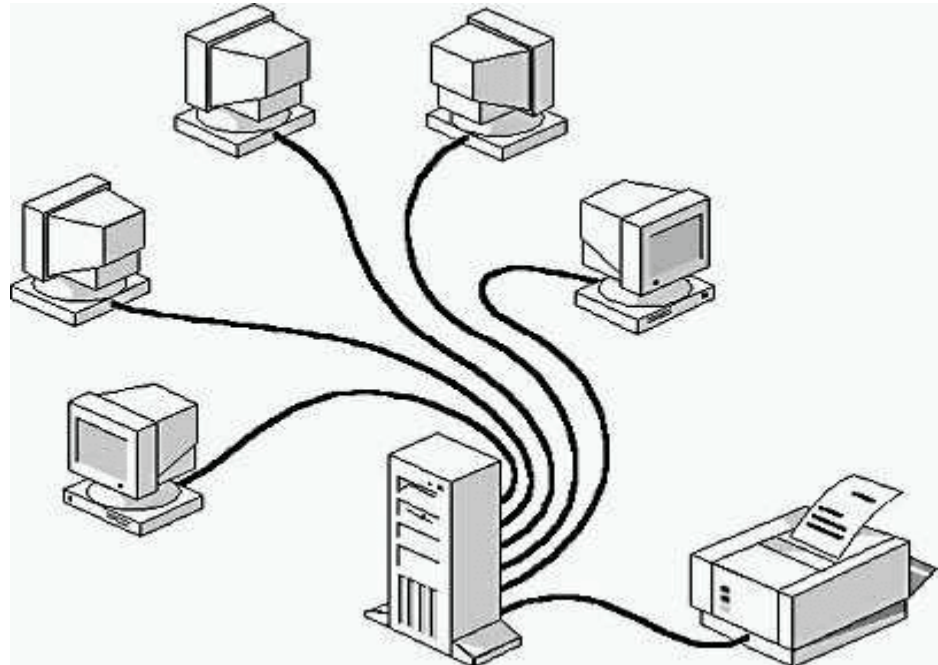


Passengers in queue  
at Railway station, Bus  
stop, Movie ticket  
counter, etc.



# Applications of Queue

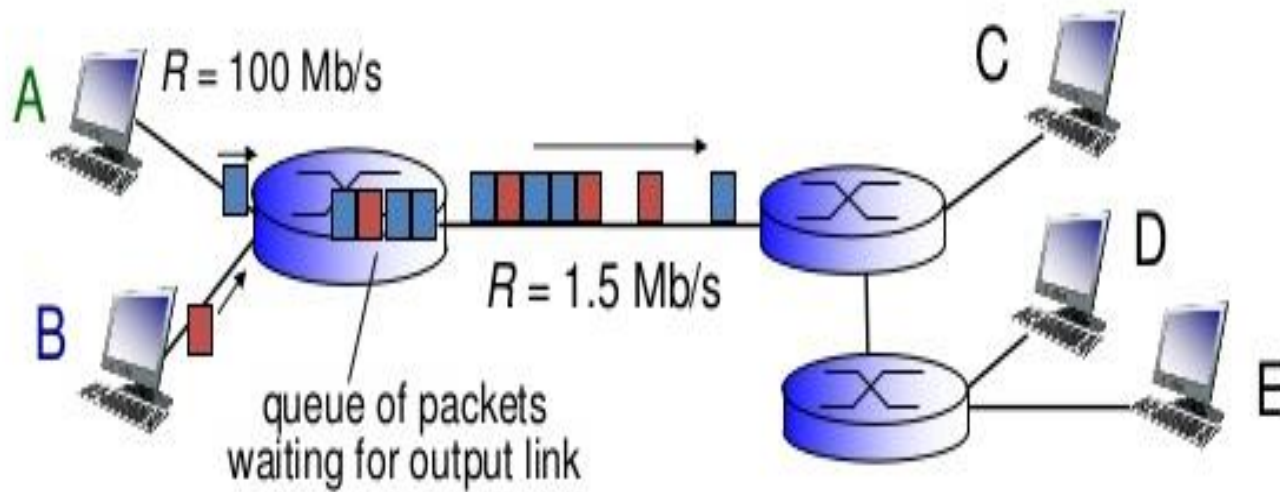
- Resource Sharing



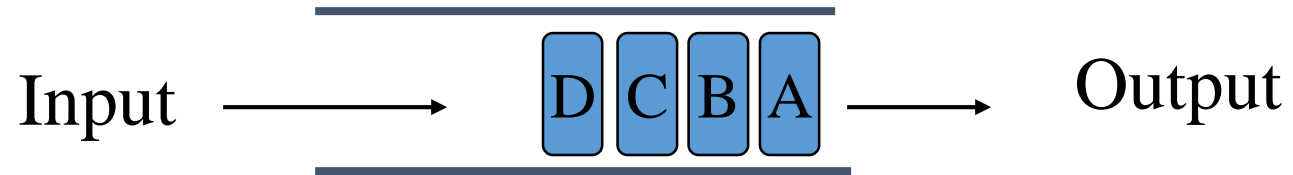
# Applications of Queue

## Networking

- Packet transmission

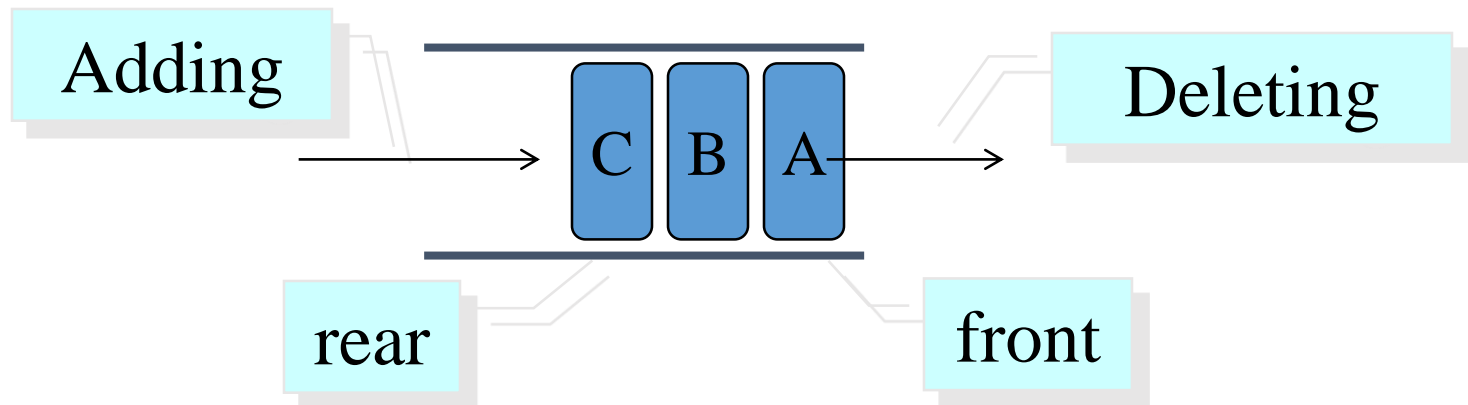


# Queue



# What Is Queue

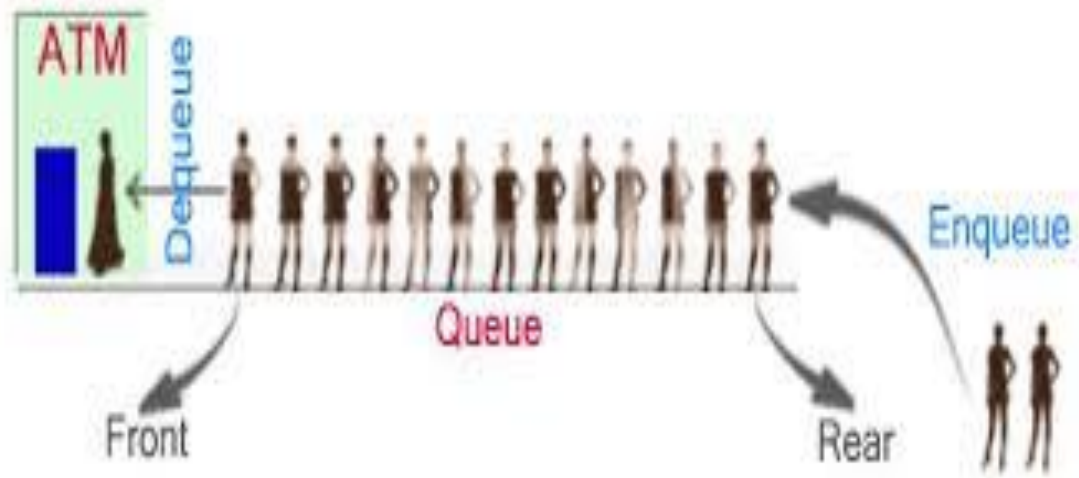
- Queue is an linear data structure
  - *First In First Out (FIFO)*
- Adding an entry at the rear
- Deleting an entry at the front



# Queue

- Queue is FIFO ( First-In First-Out)
- A queue is open at two ends.
  - add entry (**enqueue**) at the **rear**
  - delete entry (**dequeue**) at the **front**.

Note that you cannot add/extract entry in the middle of the queue.



# Linear Data structures

Array	Stack	Queue
Random Access	LIFO (Last In First Out)	FIFO (First In First Out)
Index is used to access array elements	TOP pointer is used to access top most element of the stack	FRONT and REAR pointer is used to access queue elements
	Only one end is used for insertion and deletion. Push() : Insertion Pop() : Deletion	Two end : FRONT end is used for deletion and REAR end is used for insertion. Enqueue() : Insertion Dequeue() : Deletion



# Operations

- Enqueue()
  - add a new item at the rear
- Dequeue()
  - remove a item from the front
- isEmpty()
  - check whether the queue is empty or not
- isFull()
  - check whether the queue is full or not
- Size()
  - return the number of items in the queue
- Peek()
  - return the front item without removing it.

# More Examples of Queue

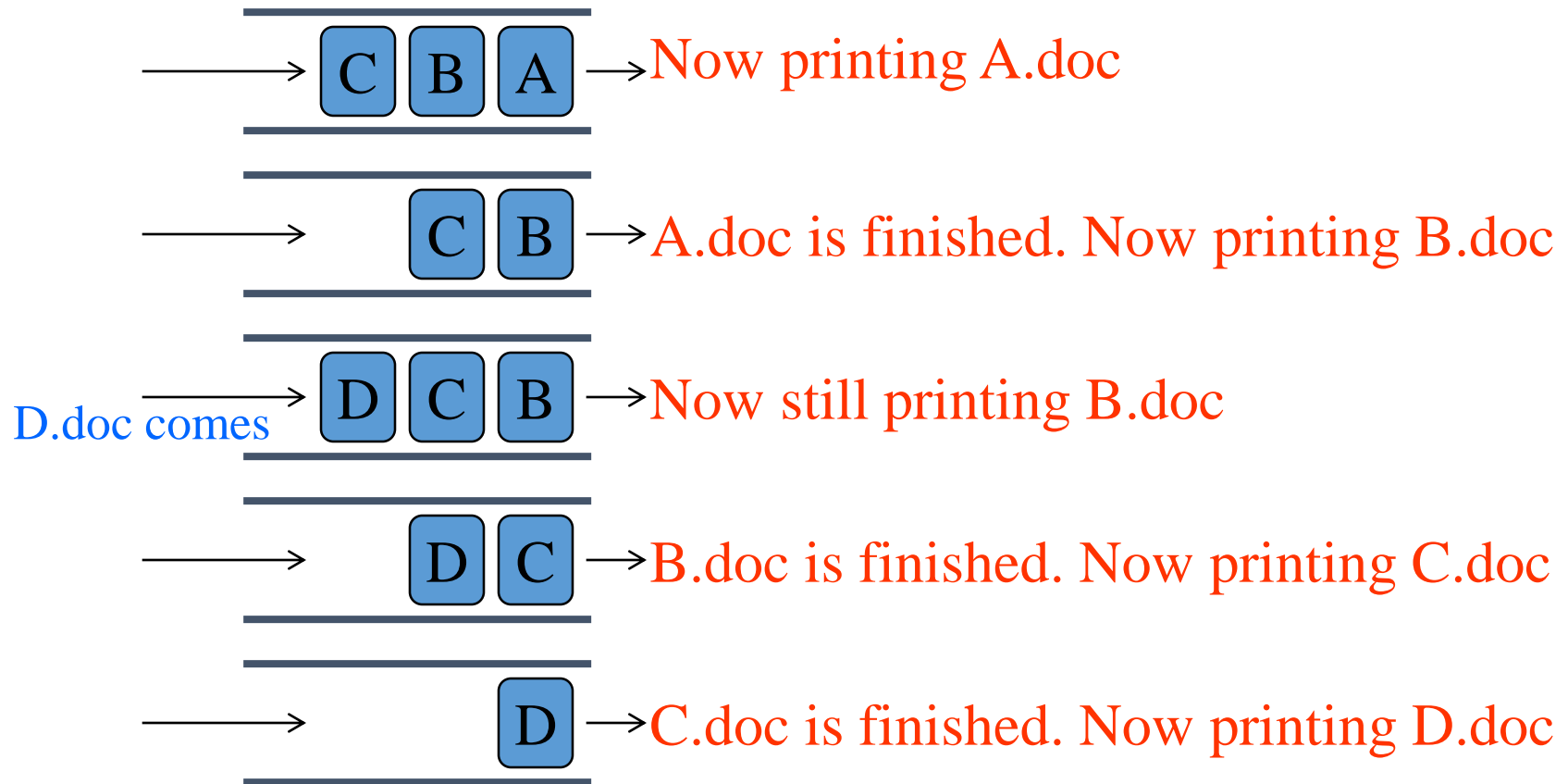
- In our daily life
  - Airport Security Check
  - Cinema Ticket Office
  - Bank, ATM
  - Packet Forwarding in Routers
  - Message Queue in Windows
  - Printing Job Management

q = new QueueImpl()	q → []
q.enqueue(a)	q → [a]
q.enqueue(b)	q → [a,b]
q.enqueue(c)	q → [a,b,c]
q.enqueue(d)	q → [a,b,c,d]
q.dequeue() → a	q → [b,c,d]
q.dequeue() → b	q → [c,d]
q.enqueue(e)	q → [c,d,e]
q.dequeue() → c	q → [d,e]
q.dequeue() → d	q → [e]

→ Elements of a queue are processed in the same order as the they are inserted into the queue, here “a” was the first element to join the queue and it was the first to leave the queue: first-come first-serve.

# Printing Queue

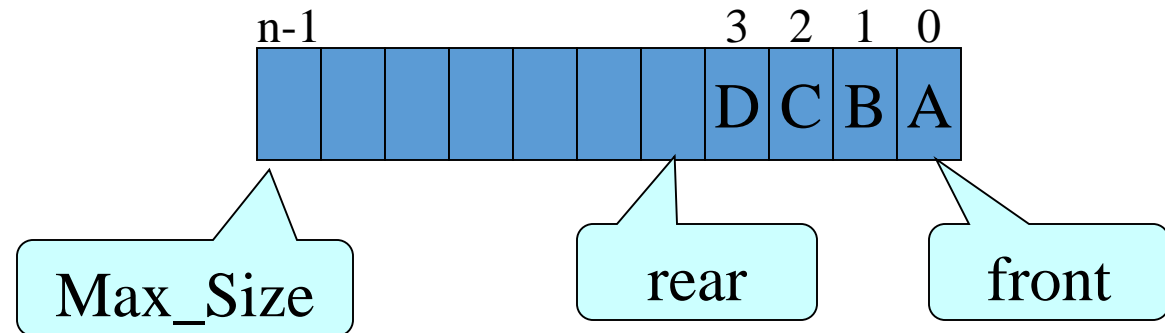
- A.doc B.doc C.doc arrive to printer.



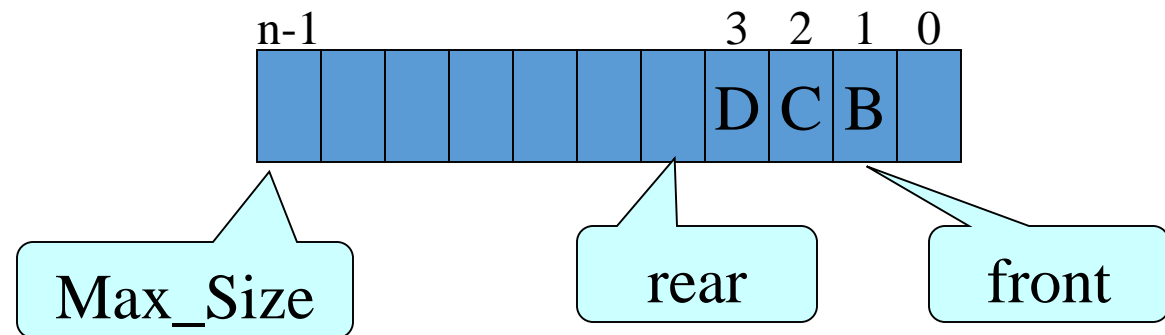
# Representation of Queue

- Two ways to represent a queue in memory:
  - 1) Using Array
  - 2) Using Linked List

# Array Representation of Queue



After A leaves,



# Qinsert

**Procedure Enqueue(Q, F, R, N, Y).** Given F and R, pointers to the front and rear elements of a queue, a queue Q consisting of N elements, and an element Y, this procedure inserts Y at the rear of the queue. **Prior to the first invocation of the procedure, F and R have been set to zero.**

1. **[Overflow?]**

If  $R \geq N$

then Write ("OVERFLOW")

Return

2. **[Increment rear pointer]**

$R \leftarrow R + 1$

3. **[Insert element]**

$Q[R] \leftarrow Y$

4. **[Is front pointer properly set?]**

If  $F = 0$

then  $F \leftarrow 1$

Return

---

**Function Dequeue (Q, F, R).** Given F and R, the pointers to the front and rear elements of a queue, respectively, and the queue Q to which they correspond, this function deletes and returns the last element of the queue. Y is a temporary variable.

### 1. [Underflow?]

If  $F = 0$

then Write (“UNDERFLOW”)

Return(0) (0 denotes an empty queue)

### 2. [Delete element]

$Y \leftarrow Q[F]$

### 3. [Queue empty? ]

If  $F = R$

then

$F \leftarrow 0$

$R \leftarrow 0$

else

$F \leftarrow F + 1$  (Increment Front pointer)

### 4. [Return element]

Return (Y)



# Qinsert : Enqueue

## 1. [Overflow?]

- If  $R \geq N$
- then Write ("OVERFLOW")
- Return

## 2. [Increment rear pointer]

- $R \leftarrow R + 1$

## 3. [Insert element]

- $Q[R] \leftarrow Y$

## 4. [Is front pointer properly set?]

- If  $F = 0$
- then  $F \leftarrow 1$
- Return

# Qdelete : Dequeue

## 1. [Underflow?]

If  $F = 0$

then Write ("UNDERFLOW")

Return(0) (0 denotes an empty queue)

## 2. [Delete element]

$Y \leftarrow Q[F]$

## 3. [Queue empty?]

If  $F = R$

then

$F \leftarrow R \leftarrow 0$

else

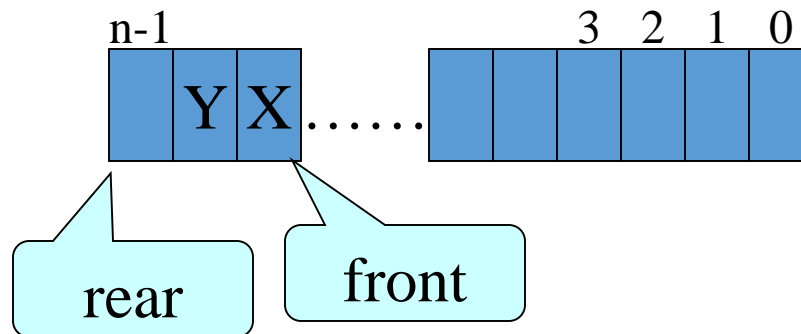
$F \leftarrow F + 1$  (Increment Front pointer)

## 4. [Return element]

Return (Y)

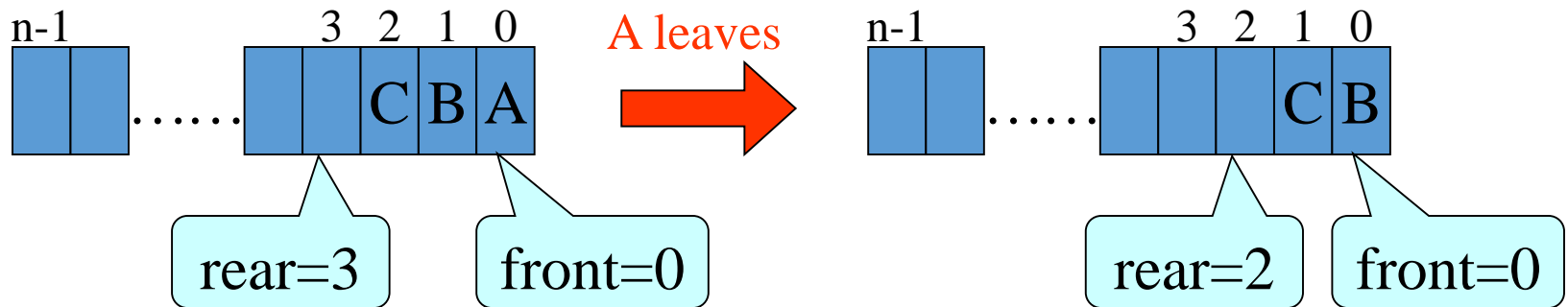
# Problem

- An array has limited size, once rear is at the end of this array, and there is new item coming in, what can we do?



# Two Solutions

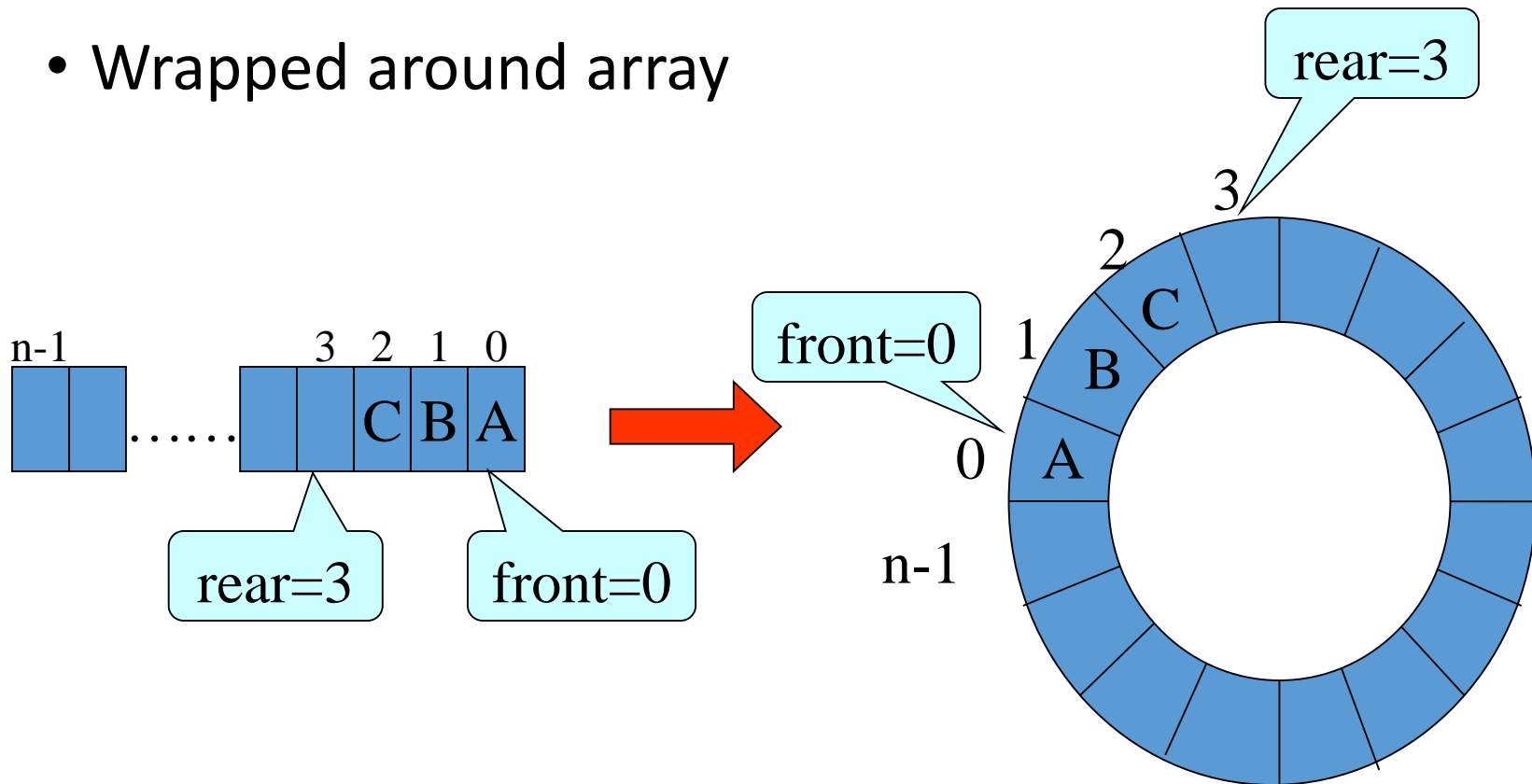
- Shifting all items to front in the array when dequeue operation. ( **Too Costly...** )



- Wrapped around array ---- Circular Array

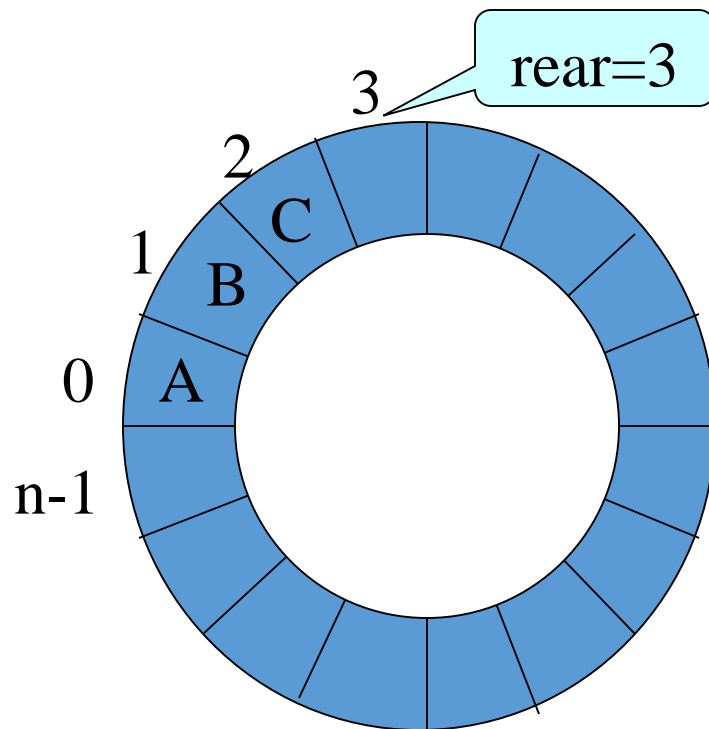
# Circular Array

- Wrapped around array

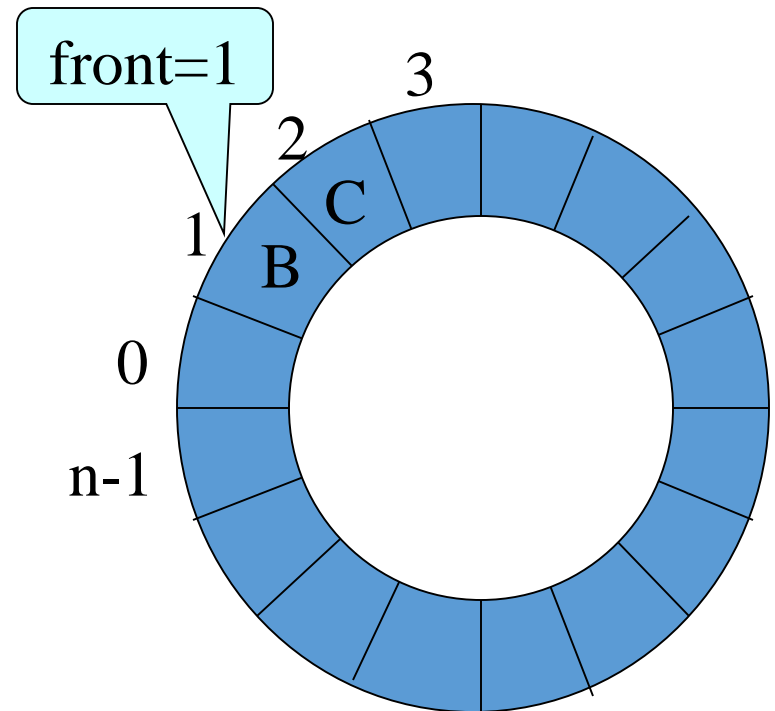


# EnQueue & DeQueue In Circular Array

- EnQueue



- DeQueue



## CIRCULAR QUEUE

**Procedure CQ\_Enqueue (F, R, Q, N, Y).** Given pointers to the front and rear elements of a circular queue, F and R, a vector Q consisting of N elements, and an element Y, this procedure inserts Y at the rear of the queue. Initially, F and R are set to **zero**.

### 1. [Reset rear pointer?]

If  $R == N$

then  $R \leftarrow 1$

else  $R \leftarrow R + 1$

### 2. [Overflow?]

If  $F == R$

then Write ("OVERFLOW")

Return

### 3. [Insert element]

$Q[R] \leftarrow Y$

### 4. [Is front pointer properly set?]

If  $F = 0$

then  $F \leftarrow 1$

Return

---

# CIRCULAR QUEUE

**Function CQ\_Dequeue( F, R, Q, N).** Given F and R, pointers to the front and rear elements of a circular queue, respectively, and a vector Q consisting of N elements, this function deletes and returns the last element of the queue. Y is a temporary variable.

## 1. [Underflow?]

If  $F == 0$   
then Write (“UNDERFLOW”)  
Return(0)

## 2. [Delete element]

$Y \leftarrow Q[F]$

## 3. [Queue empty? ]

If  $F == R$   
then  $F \leftarrow R \leftarrow 0$   
Return (Y)

## 4. [Increment front pointer]

If  $F == N$   
then  $F \leftarrow 1$   
else  $F \leftarrow F + 1$   
Return (Y)

# CQ\_Enqueue

## 1. [Reset rear pointer?]

If  $R == N$

then  $R \leftarrow 1$

else  $R \leftarrow R + 1$

## 2. [Overflow?]

If  $F == R$

then Write (“OVERFLOW”)

Return

## 3. [Insert element]

$Q[R] \leftarrow Y$

## 4. [Is front pointer properly set?]

If  $F = 0$

then  $F \leftarrow 1$

Return  $\square$

# CQ\_Dequeue

## 1. [Underflow?]

If  $F == 0$

then Write (“UNDERFLOW”)

Return(0)

## 2. [Delete element]

$Y \leftarrow Q[F]$

## 3. [Queue empty? ]

If  $F == R$

then  $F \leftarrow R \leftarrow 0$

Return (Y)

## 4. [Increment front pointer]

If  $F == N$

then  $F \leftarrow 1$

else  $F \leftarrow F + 1$

Return (Y)



# Example

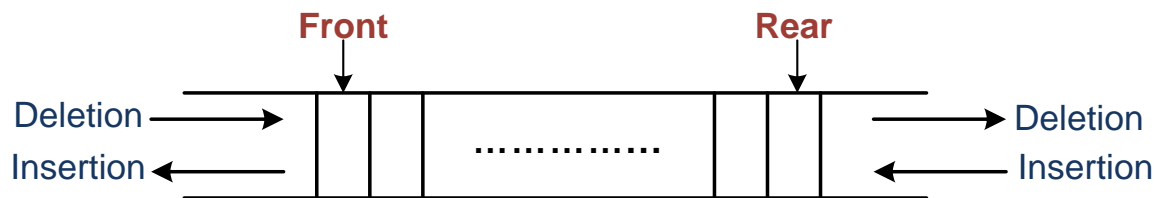
- Show circular queue contents with front and rear after each step with size=5.
  - (i) Insert 10, 20, 30.
  - (ii) Delete
  - (iii) Insert 40, 50, 60, 70.

# Customer Service In Royal Bank

- Suppose there is only one customer service available in Royal Bank on Saturday morning at 8:00 clock.
- In every 3 minutes, a new customer arrives at the end of waiting line
- Each customer will need 5 minutes for the service
- Print out the following information of queue at 8:30 am
  - The time of arriving and leaving for each customer
  - How many customers are in the line?
  - Who is the current serving customer?

# Deque

- Unlike a queue, in deque, both insertion and deletion operations can be made at either end of the structure. Actually, the term deque has originated from double ended queue.



- It is clear from the deque structure that it is a general representation of both stack and queue.
- In other words, a deque can be used as a stack as well as a deque.

# Operations

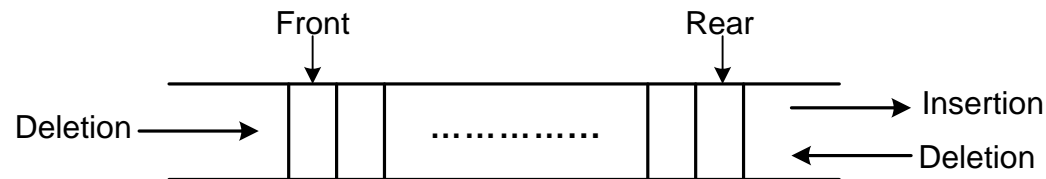
- The following four operations are possible on a deque which consists of a list of items:
- Push DQ(item): to insert item at the **Front** end of a deque.
- Pop DQ(): to remove the **Front** item from the deque.
- Inject(item): to insert item at the **Rear** end of a deque.
- Eject(): to remove the **Rear** item from a deque.

# Variations of Deque

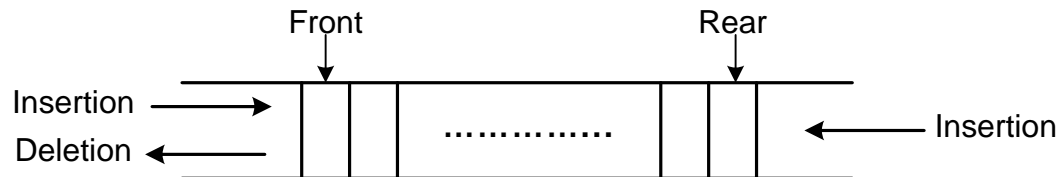
- There are two known variations of deque:

1. Input-restricted deque.

2. Output-restricted deque.



**Input-restricted deque**



**Output-restricted deque**

An input-restricted deque is a deque which allows insertions at one end(rear) only, but allows deletions at both ends.

An output-restricted deque is a deque where deletions take place at one end only(front), but allows insertions at both ends.

# Example

1. Perform following operations on Deque. Size = 8

IF A1, IF A2, IR B1, DF, IR B2, IF A3, DR, DR, IR B3, IF A4,  
DF, DF, DR, DR

2. Perform following operations on Input Restricted  
Deque. Size = 6

I1, I2, I3, I4, DR, DF, DF, I5, DR, I6, DF, DR

3. Perform following operations on Output Restricted  
Deque. Size = 7

IF O1, IR O2, IR O3, D, IF O4, IF O5, D, D, D, IR O6, IR O7,  
IF O8, D, D, D

# Priority Queue

- Two queues
  - one is high priority queue
  - the other is low priority queue
- Service rules:
  - First serve the people in high priority queue
  - If no passengers are in high priority queue, serve the passengers in low priority queue

# Priority Queue



- A priority queue is **not** a queue because it does not strictly follow the ***first-in-first-out*** (FIFO) principle which is the basic principle of queue.
1. An element of higher priority is processed before any element of lower priority.
  2. Two elements with the same priority are processed accordingly to the order in which they are added to the queue (FIFO).



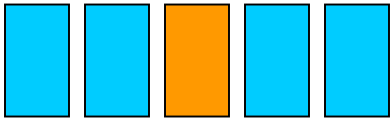
# Air Travel

- Only one check-in service in Air Canada at airport
- Two waiting lines for passengers
  - one is First class service
  - the other is Economy class service
- Passengers in the first-class waiting line have higher priority to check in than those in the economy-class waiting line.

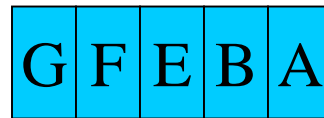
# Two Queues

- High Priority Queue,  will come in *hpQue*
- Low Priority Queue,  will come in *lpQue*

Customers coming in



High Priority Queue



Low Priority Queue

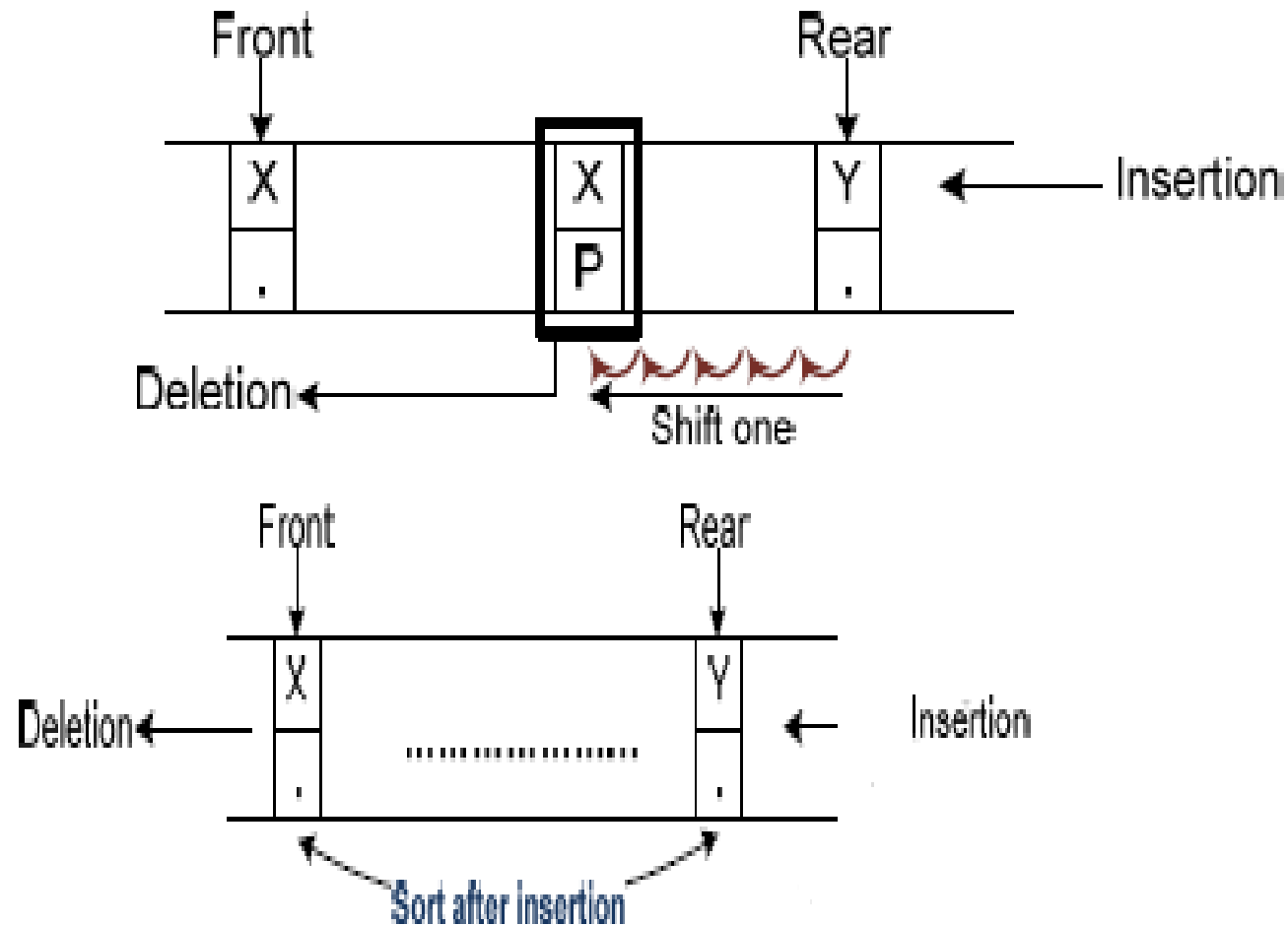
Check In



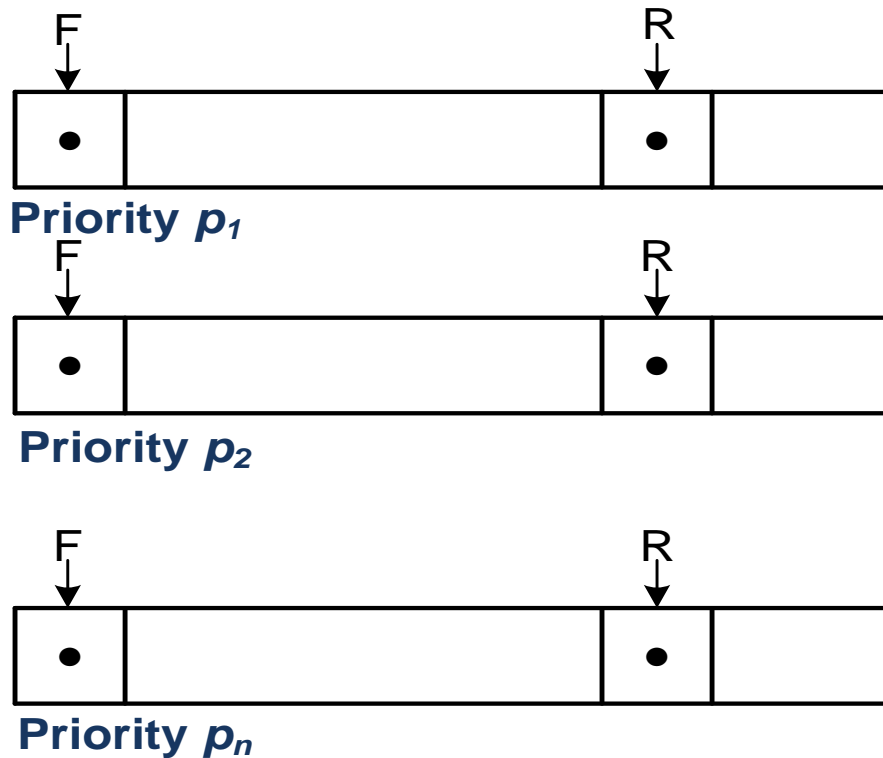
# Priority Queue

- There are various ways to implement Priority Queue :
  1. Using Simple array/ Circular array
  2. Multi-queue implementation
  3. Using double linked list
  4. Using heap tree

# Priority Queue Using Array



# Multi-queue implementation



# Example

Perform Following operations on Priority Queue using Simple Queue. Size = 6

Person1-4, Person2-3, Person3-5, Person4-2,

Delete, Delte, Person5-1, Person6-2, Person7-5, Delete, Delete, Delete, Delete

(Here 1 indicate higher priority)