CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

Accredited with Grade A by NAAC
Accredited with Grade A by KCG

cspit
Chandubhai S Patel Institute of Technology

CE245 : Data Structure and Algorithms

Topic: Queue Data Structure

Prepared By,
Dr. Nikita Bhatt

Queue Data Structure

- It is an homogeneous collection of elements in which new elements are added at one end called the Rear end (R) and the existing elements are deleted from other end called the Front end (F).

- A queue follows First In First Out (FIFO)

Simple Queue: Array Implementation

Insert

```
void Qenqueue(int item)
{
    if(rear == n − 1)
          printf("Queue Overflow");
    else
     {
               rear = rear + 1;
                q[rear] = item;
      }
if (front == −1)
front = 0;
return;

}
```

Simple Queue: Array Implementation(continue)

Delete

```
int Qdequeue
{
    if (front == −1)
    {
            printf("Underflow");
             return 0;
      }
item = q[front];
if(front == rear)
        front = rear = −1;
else
front = front + 1;
return item;
}
```

Simple Queue: Linked List Implementation

Insert

```
struct node {
int data;
struct node *next; };

struct node *front = NULL, *rear = NULL;
void enqueue(int val) {
struct node *newNode = malloc(sizeof(struct node));
newNode->data = val;
newNode->next = NULL;
if(front == NULL && rear == NULL)
front = rear = newNode;
else
{
rear->next = newNode;
rear = newNode;
} }
```

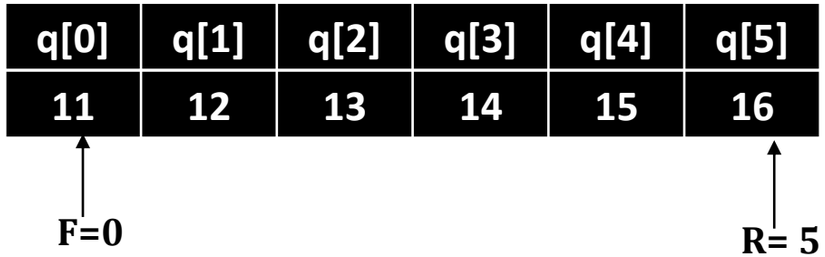
Simple Queue: Linked List Implementation(continue)

Delete

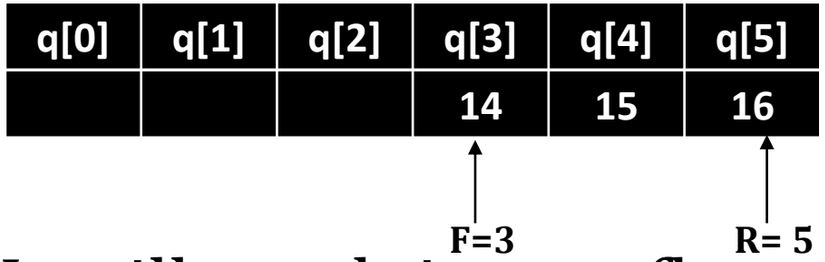
```
void dequeue()
{
    struct node *temp;
    if(front == NULL)
       printf("Queue is Empty);
   else
   {
        temp = front;
        front = front->next;
           if(front == NULL)
                    rear = NULL;
        free(temp);
   }
}
```

Disadvantage of Simple Queue

Consider the full queue:



After 3 delete operations:



Call Insert function for 17. It will result in overflow as R= n-1 (where n is the size of the queue)

This is the disadvantage of simple queue as there are blank spaces in an array but we cannot store the elements.

To solve this, we can use *Circular queue*.

Circular Queue: Array Implementation

```
1. [Reset rear pointer?]
If  R ==  N
then
R ← 1
else R ← R + 1
2. [Overflow?]
If F == R
then Write ("OVERFLOW")     //Update R to its previous position
        If  R ==  1
                then R ← N
        else
                R ← R - 1
Return
3. [Insert element]
Q [R] ← Y
4. [Is front pointer properly set?]
If F = 0
then
        F ←1
Return
```

Circular Queue: Array Implementation(continue)

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

- Show circular queue contents with front and rear after each step with size=5.

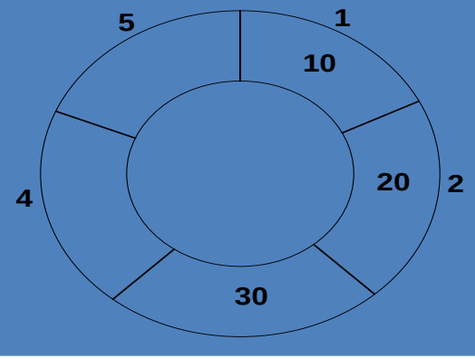
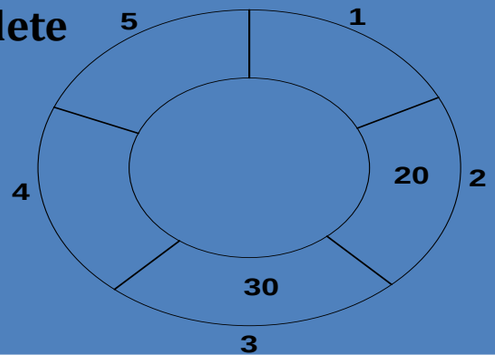
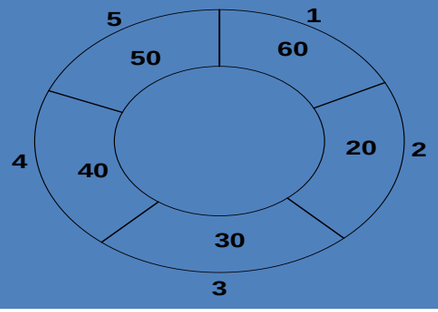
(i) Insert 10, 20, 30.

(ii) Delete

(iii) Insert 40, 50, 60, 70.

10

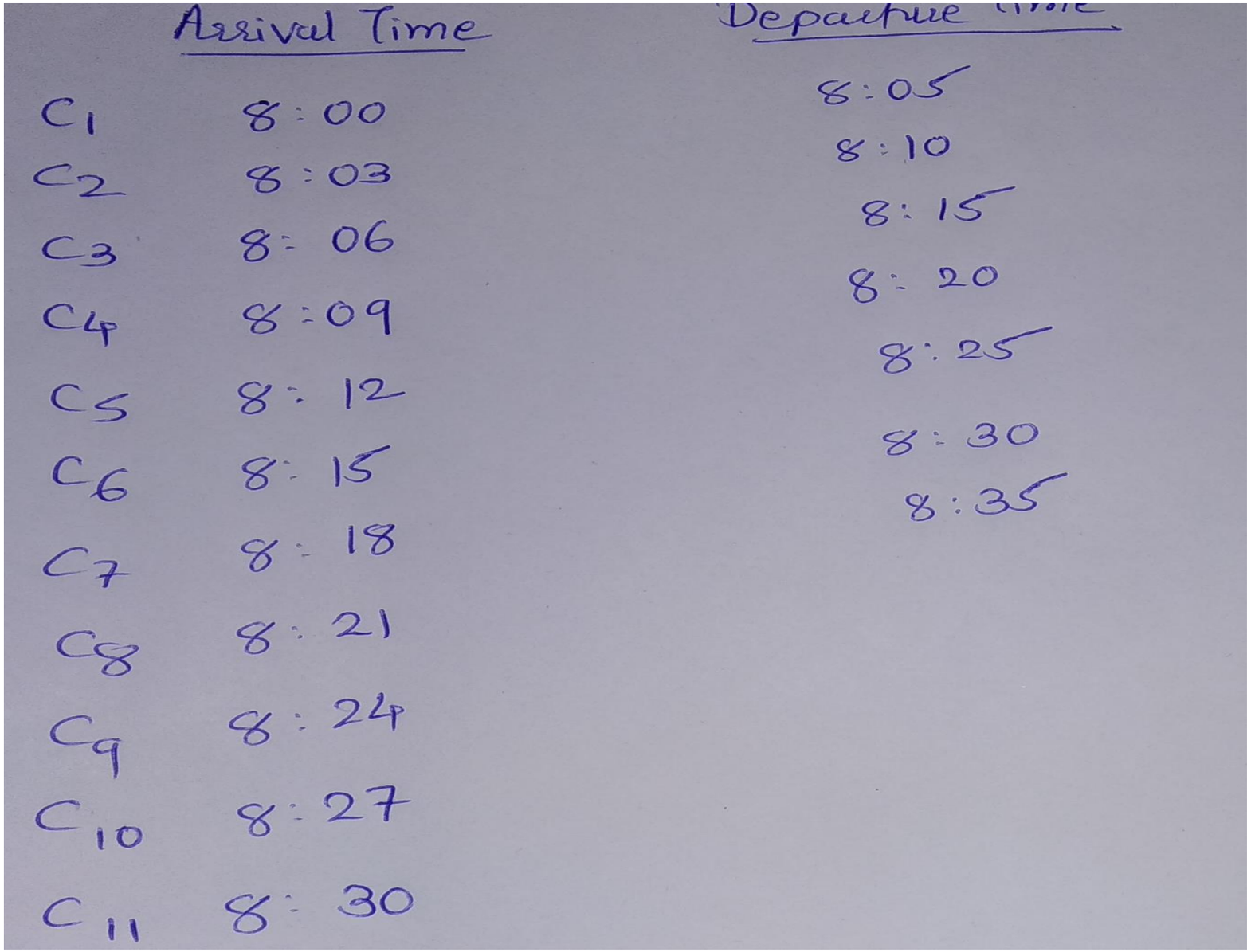
Example

(i) Insert 10, 20, 30  	**After Insert 10 R=1, F=1** **After Insert 20 R=2, F=1** **After Insert 30 R=3, F=1**
Delete  	**After delete R=3, F=2**
Insert 40, 50, 60, 70  	**After Insert 40 R=4, F=2** **After Insert 50 R=5, F=2** **After Insert 60 R=1, F=2** **After insertion of 60 circular queue is full. So, when 70 is tried to be inserted it will be OVERFLOW. So, 70 can't be inserted.**

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?

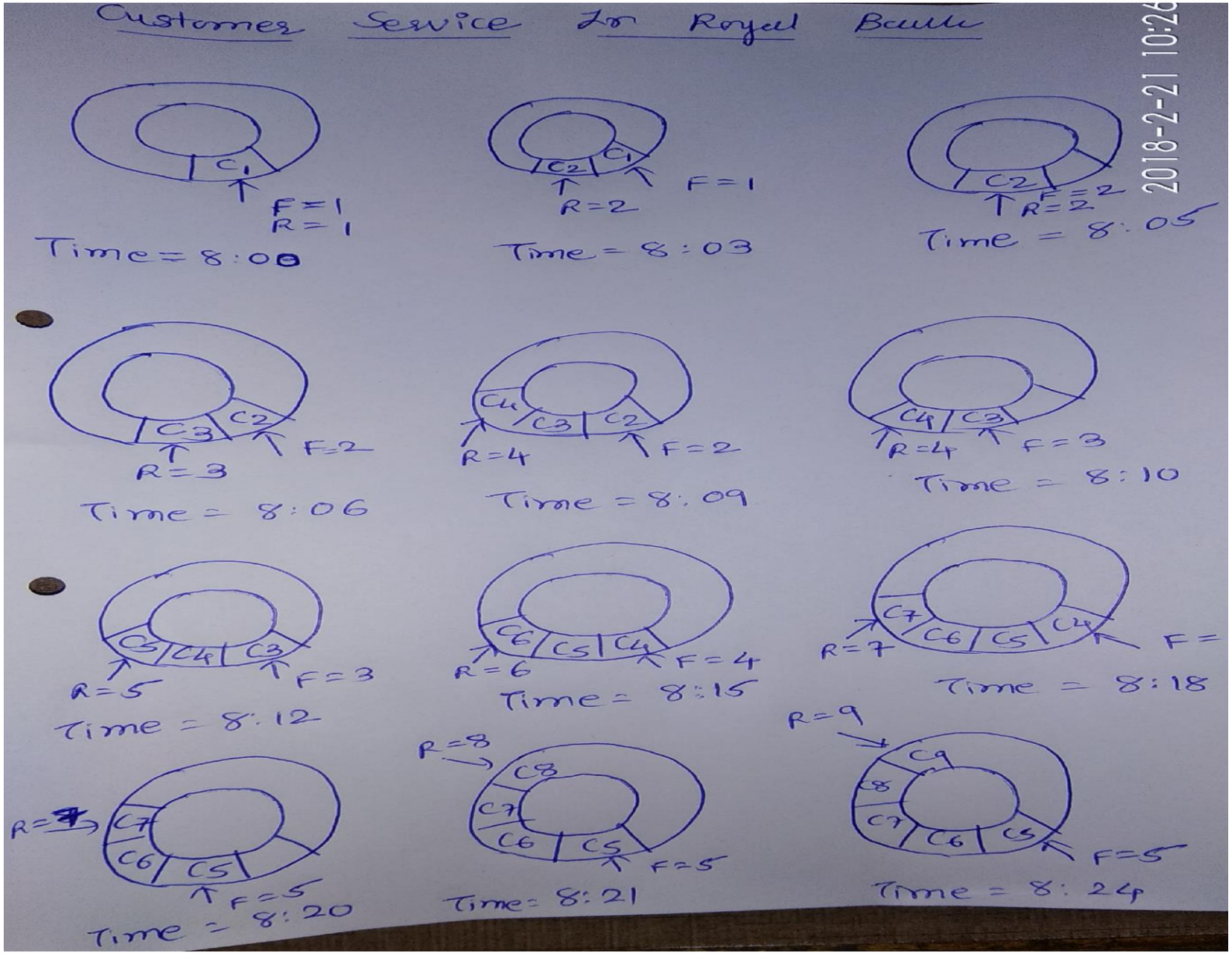
Customer Service In Royal Bank(Continue)



	Arrival Time	Departure Time
C_1	8:00	8:05
C_2	8:03	8:10
C_3	8:06	8:15
C_4	8:09	8:20
C_5	8:12	8:25
C_6	8:15	8:30
C_7	8:18	8:35
C_8	8:21	
C_9	8:24	
C_{10}	8:27	
C_{11}	8:30	

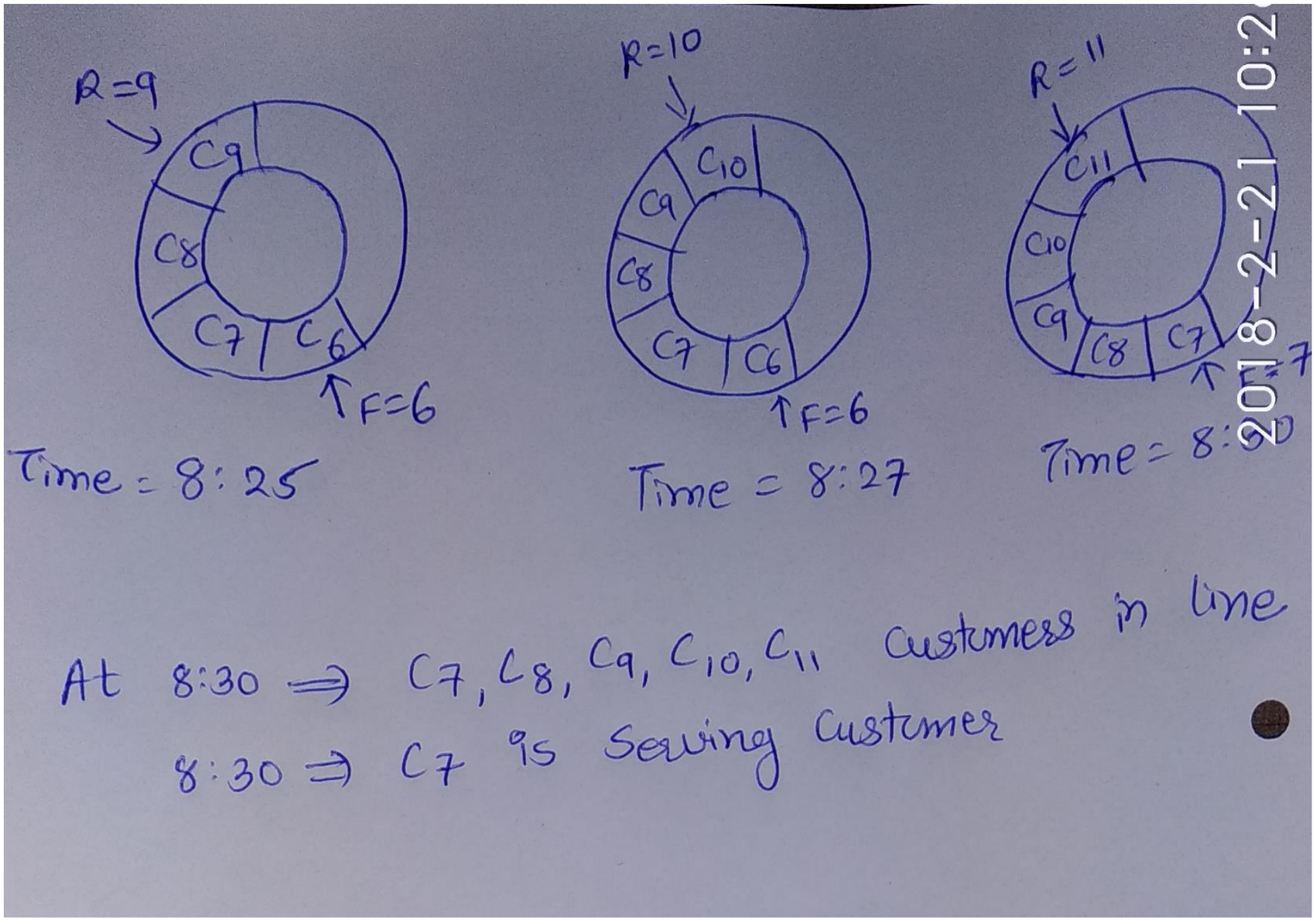
13

Customer Service In Royal Bank(Continue)



14

Customer Service In Royal Bank(Continue)



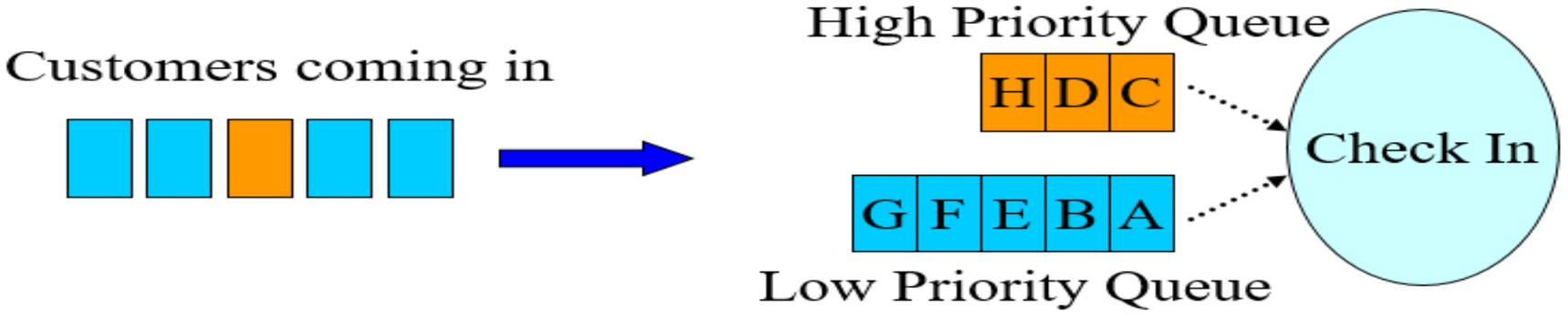
15

Priority Queue: Application: Air Travel

- Only one check-in service at airport
- Two waiting lines for passengers
  - One: First class service
  - Second: Economy class service

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).

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



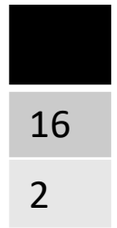
Priority Queue

- Two methods to access the elements of priority Queue:
- **Method 1**:

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
Values	11	12	13	14	15
Priority	5	1	6	4	3

F R

Consider Insert operation for value 16 with priority 2.



Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	Q[5]
Values	11	12	13	14	15	16
Priority	5	1	6	4	3	2

F R

Priority Queue(Continue)

Preference will be given to highest priority element.

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	Q[5]
Values	11	12	13	14	15	16
Priority	5	1	6	4	3	2

Find max priority element, which is Q[1].
Delete that.
Then shift the elements.

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
Values	11	13	14	15	16
Priority	5	6	4	3	2

F R

Priority Queue(Continue)

Method 2:

After Insertion, sort the elements based on priority.

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
Values	12	16	15	14	11
Priority	1	2	3	4	5

Then we can delete from front.

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	
Values		16	15	14	11	
Priority		2	3	4	5	

Example- Priority Queue(Continue)

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

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

Delete, Delete,

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

Delete, Delete, Delete, Delete

(Here 1 indicate higher priority)

20

Multi Queue implementation of Priority Queue

Priority 1 :

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
Values	11	12	13	14	15

Priority 2 :

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
Values	16	17	18	19	20

Priority 3 :

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
Values	21	22	23	24	25

F R

- Insertion will be done in queue with matching priority queue of element. Suppose we want to insert 26 with priority 2.

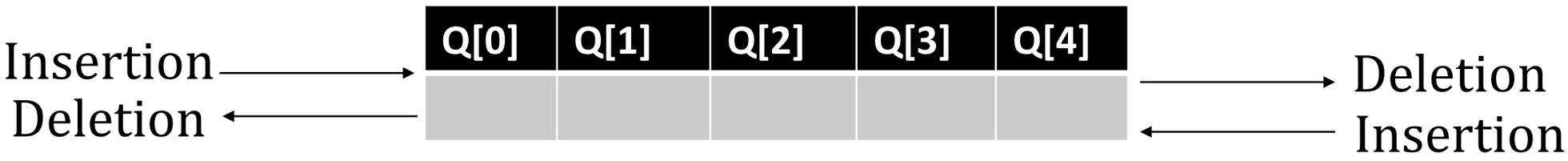
Priority 2 :

Element Names	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	Q[5]
Values	16	17	18	19	20	26

F R

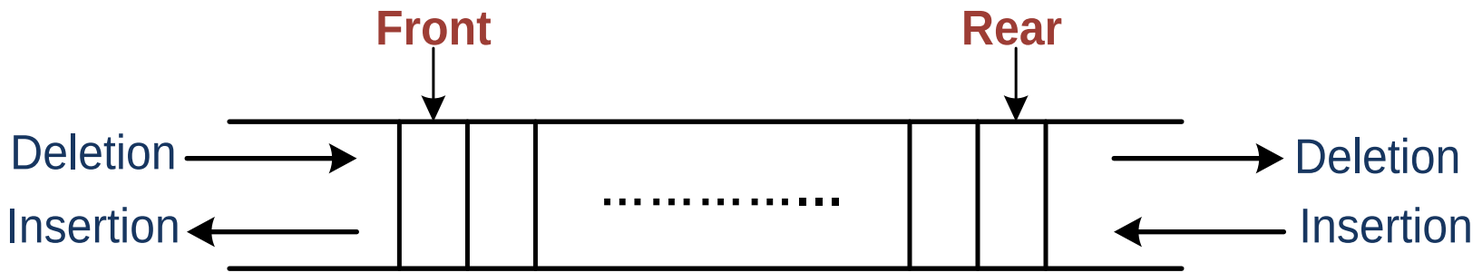
Double Ended Queue (Dequeue)

- It is a list of elements in which insertion and deletion operations are performed from both the ends.
- We can insert elements from the rear end or from the front ends. Hence it is called double-ended queue. It is commonly referred as a Deque.
- There are two types of Deque. These two types are due to the restrictions put to perform either the insertions or deletions only at one end.
  - Input Restricted Dequeue
  - Output Restricted Dequeue



Double Ended Queue (Dequeue)

- 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 **d**ouble **e**nded **que**ue.



- 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:
- Insert(item): to insert item at the **Front** end of a deque.
- Deque(): to remove the **Front** item from the deque.
- Enqueue(item): to inert item at the **Rear** end of a deque.
- Eject(): to remove the **Rear** item from a deque.

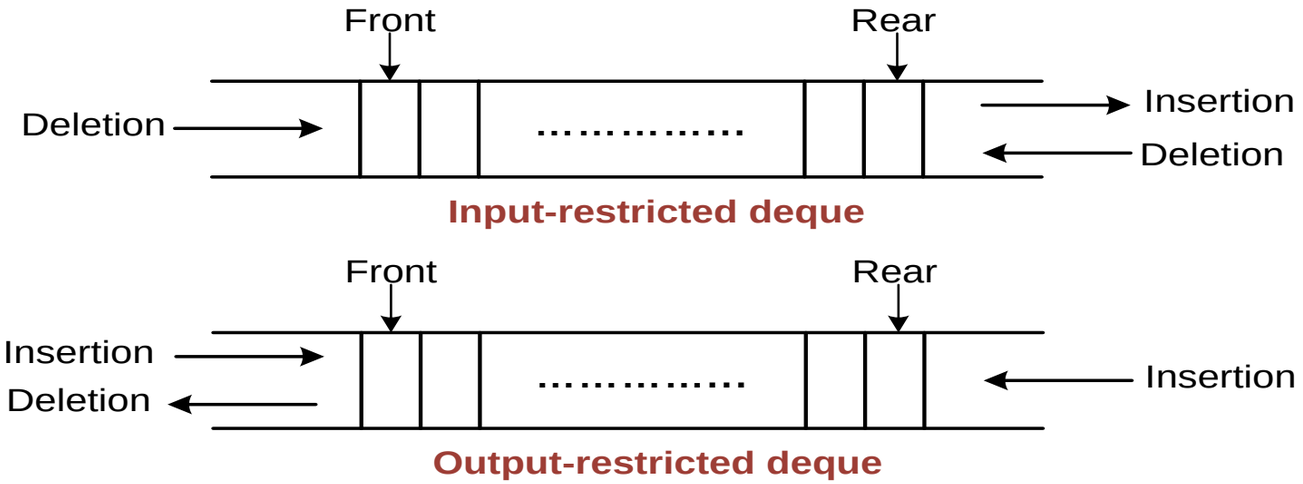
24

Variations of Deque

- There are two known variations of deque:

1.Input-restricted deque.

2.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.

25

Self Assessment

- Following is C like pseudo code of a function that takes a Queue as an argument, and uses a stack S to do processing.

Self Assessment(Continue)

```
void fun(Queue *Q)
{
   Stack S;  // Say it creates an empty stack S
    while (!isEmpty(Q)) {
            push(&S, deQueue(Q));
   }
       while (!isEmpty(&S))
   {
          enQueue(Q, pop(&S));
   }
}
```

Self Assessment

What does the above function do in general?

(A) Removes the last from Q

(B) Keeps the Q same as it was before the call

(C) Makes Q empty

(D) Reverses the Q

Answer: (D)

Self Assessment

Consider the following pseudo code. Assume that IntQueue is an integer queue. What does the function fun do?

Self Assessment(Continue)

```
void fun(int n)
{
    IntQueue q = new IntQueue();
    q.enqueue(0);
    q.enqueue(1);
    for (int i = 0; i < n; i++) {
        int a = q.dequeue();
        int b = q.dequeue();
        q.enqueue(b);
        q.enqueue(a + b);
        print(a);
    }
}
```

Self Assessment

(A) Prints numbers from 0 to n-1

(B) Prints numbers from n-1 to 0

(C) Prints first n Fibonacci numbers

(D) Prints first n Fibonacci numbers in reverse order.

Answer: (C)

Self Assessment

An array of size MAX_SIZE is used to implement a circular queue. Front, Rear, and count are tracked. Suppose front is 0 and rear is MAX_SIZE -1. How many elements are present in the queue?

(a)Zero (b) One (c) MAX_SIZE-1 (d) MAX_SIZE

Answer : (d)

Self Assessment –IV

A normal queue, if implemented using an array of size MAX_SIZE, gets full when

(a) Rear = MAX_SIZE – 1

(b) Front = (rear + 1)mod MAX_SIZE

(c) Front = rear + 1

(d) Rear = front

Answer : (a)

Self Assessment –V

A circular queue, if implemented using an array of size MAX_SIZE, gets full when

(a) Rear = MAX_SIZE – 1

(b)Front = (rear + 1)mod MAX_SIZE

(c) Front = rear + 1

(d) Rear = front

Answer : (b)

Self Assessment –VI

In linked list implementation of a queue, where does a new element be inserted?

(a) At the head of link list

(b) At the tail of the link list

(c) At the center position in the link list

(d) None

Answer : (b)

Self Assessment –VI

Which one of the following is an application of Queue Data Structure?

(A) When a resource is shared among multiple consumers.
(B) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes

(C) Load Balancing

(D) All of the above

Answer: (D)

Self Assessment –VI

Which of the following is true about linked list implementation of queue?

(A) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end.

(B) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning.

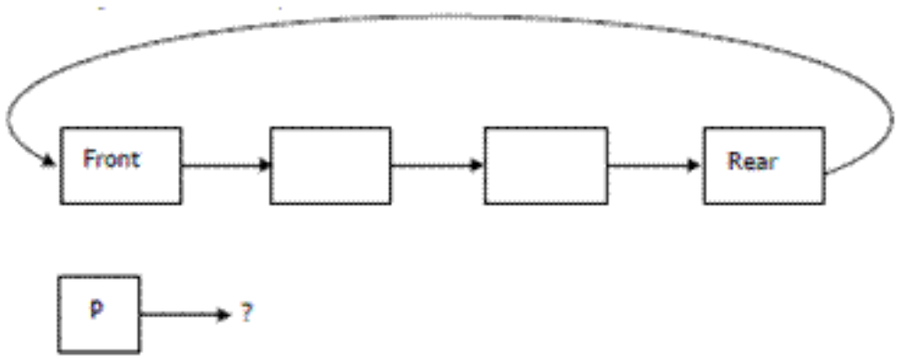
(C) Both of the above

(D) None of the above

Answer: (C)

Gate 2004

A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time?



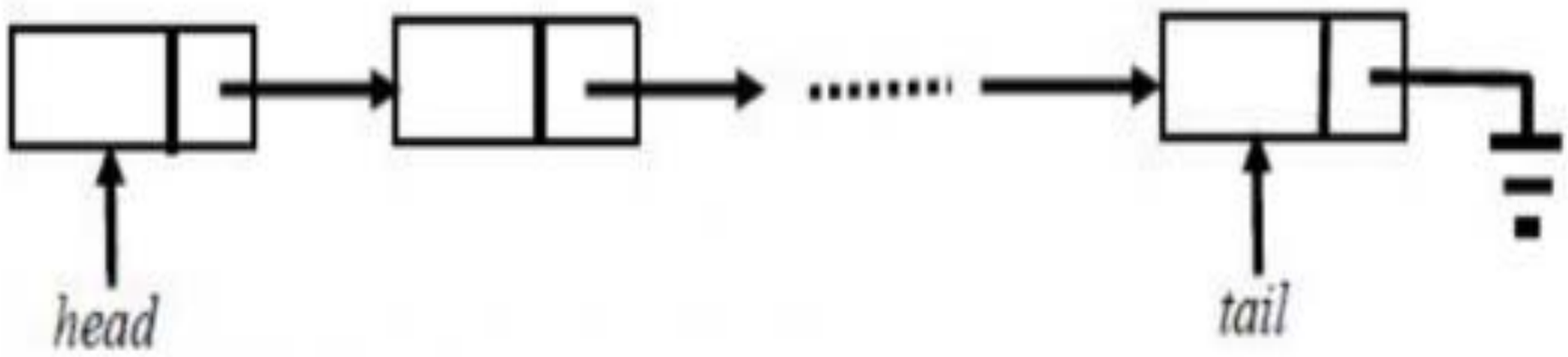
Gate 2004(Continue)

(A) rear node
(B) front node
(C) not possible with a single pointer
(D) node next to front

Answer : (A)

Gate 2018

A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let ‘enqueue’ be implemented by inserting a new node at the head, and ‘dequeue’ be implemented by deletion of a node from the tail.



Gate 2018

Which one of the following is the time complexity of the most time-efficient implementation of ‘enqueue’ and ‘dequeue, respectively, for this data structure?

(A) $\Theta(1), \Theta(1)$ **(B)** $\Theta(1), \Theta(n)$

(C) $\Theta(n), \Theta(1)$ **(D)** $\Theta(n), \Theta(n)$

Answer : (B)

Gate 2017

A Circular queue has been implemented using singly linked list where each node consists of a value and a pointer to next node. We maintain exactly two pointers **FRONT** and **REAR** pointing to the front node and rear node of queue. Which of the following statements is/are correct for circular queue so that insertion and deletion operations can be performed in O(1) i.e. constant time.

I. Next pointer of front node points to the rear node.

II. Next pointer of rear node points to the front node.

(A) I only **(B)** II only

(C) Both I and II **(D)** Neither I nor II

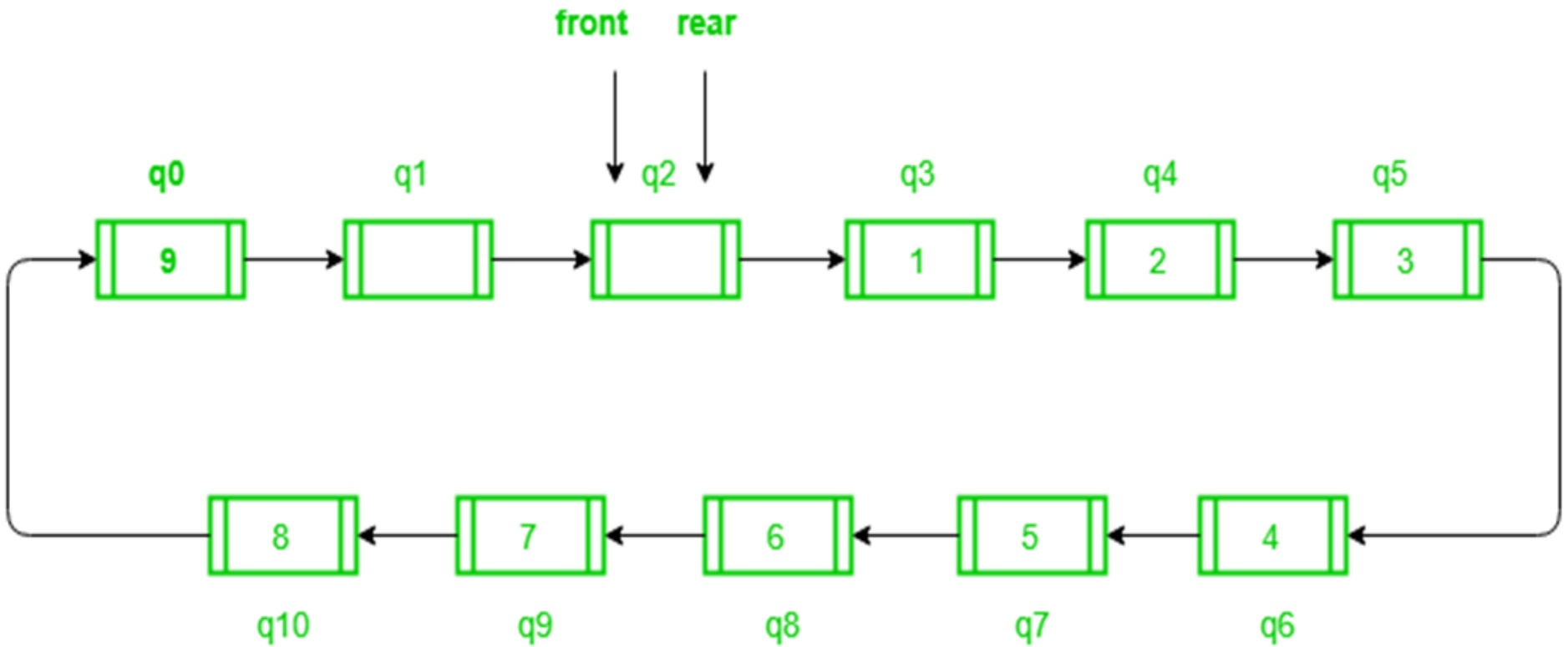
Answer: (B)

ISRO CS 2014

Consider a standard Circular Queue ‘q’ implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue are q[0], q[1], q[2].....,q[10].
The front and rear pointers are initialized to point at q[2] . In which position will the ninth element be added?

(A) q[0] **(B)** q[1] **(C)** q[9] **(D)** q[10]

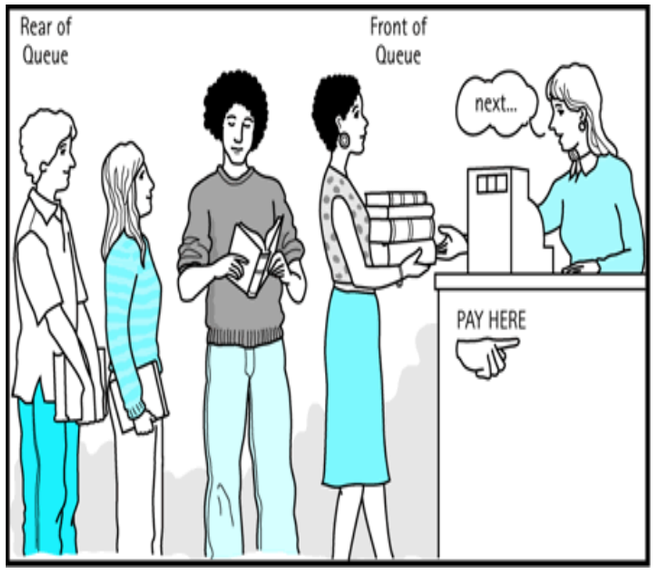
ISRO CS 2014(Continue)



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.



45

Applications of Queue (Continue)

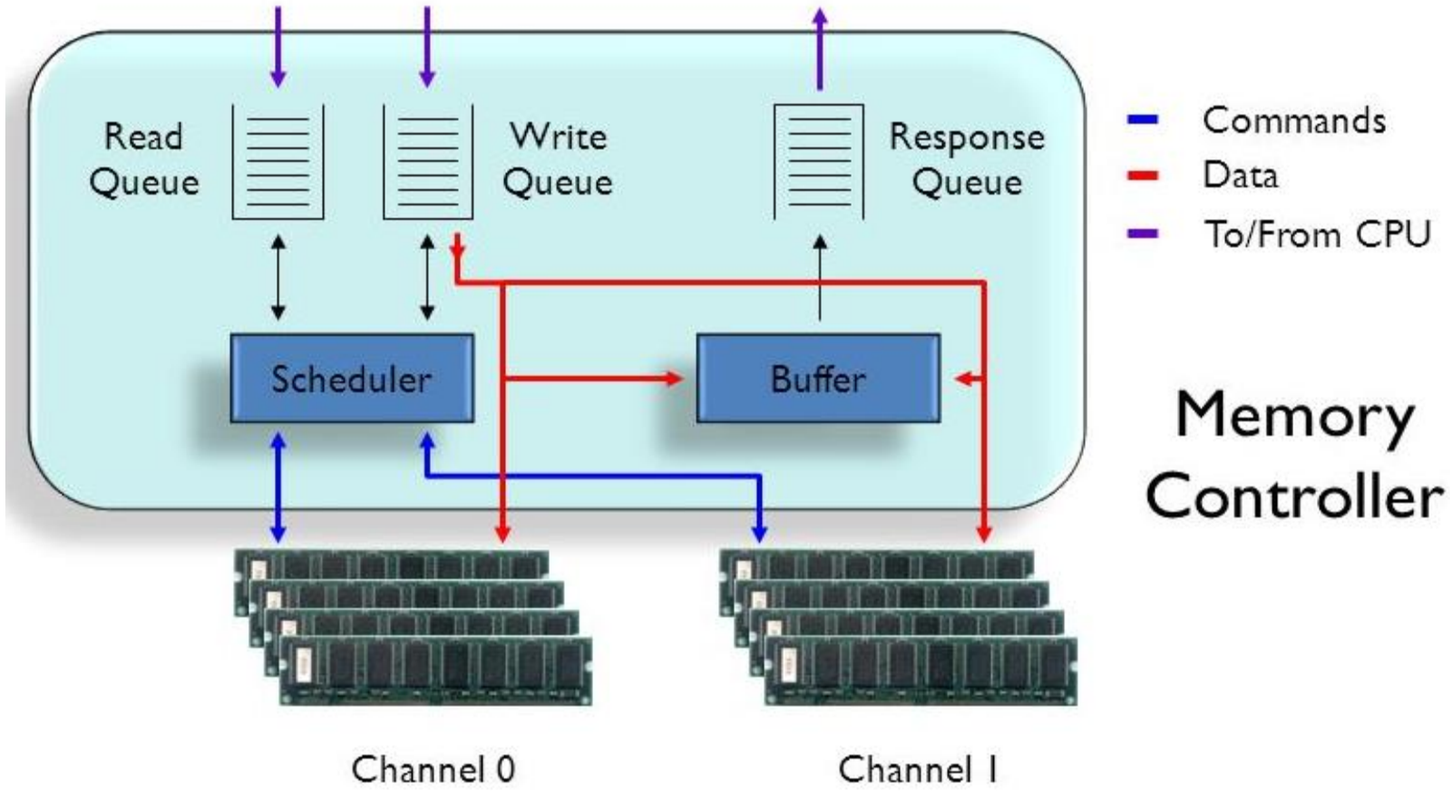
Operating System

- **Handling of interrupts** in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.
- **Data transmission** between two processes. Examples include IO Buffers, pipes, file IO, etc.

46

Applications of Queue(Continue)

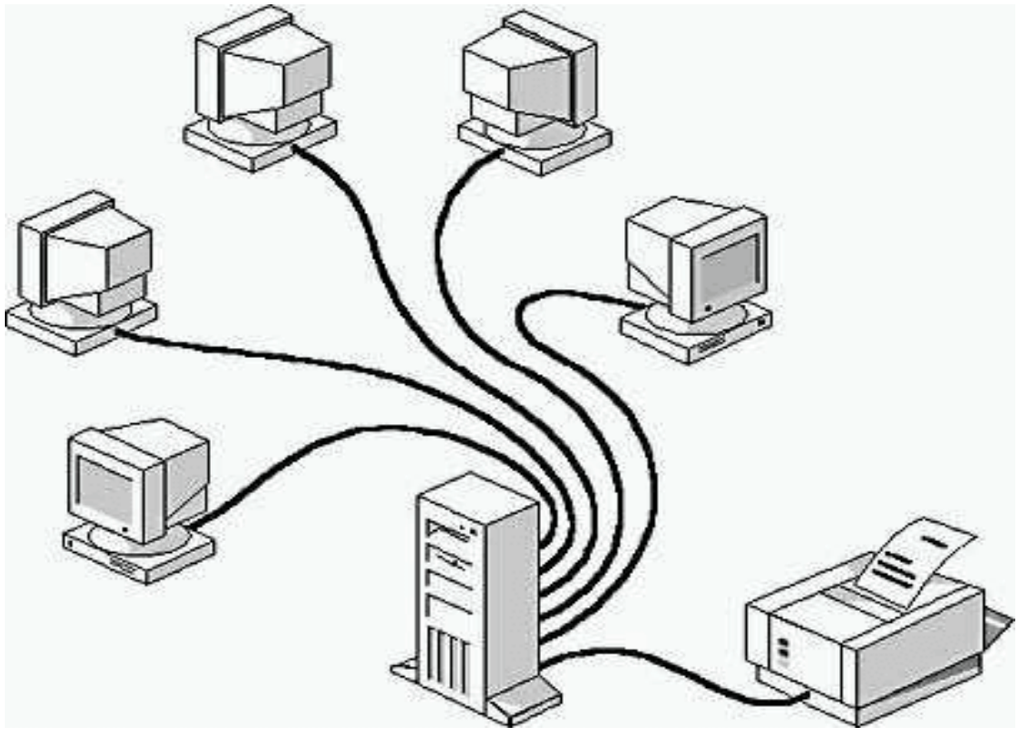
- **CPU Scheduling:** Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.



47

Applications of Queue(Continue)

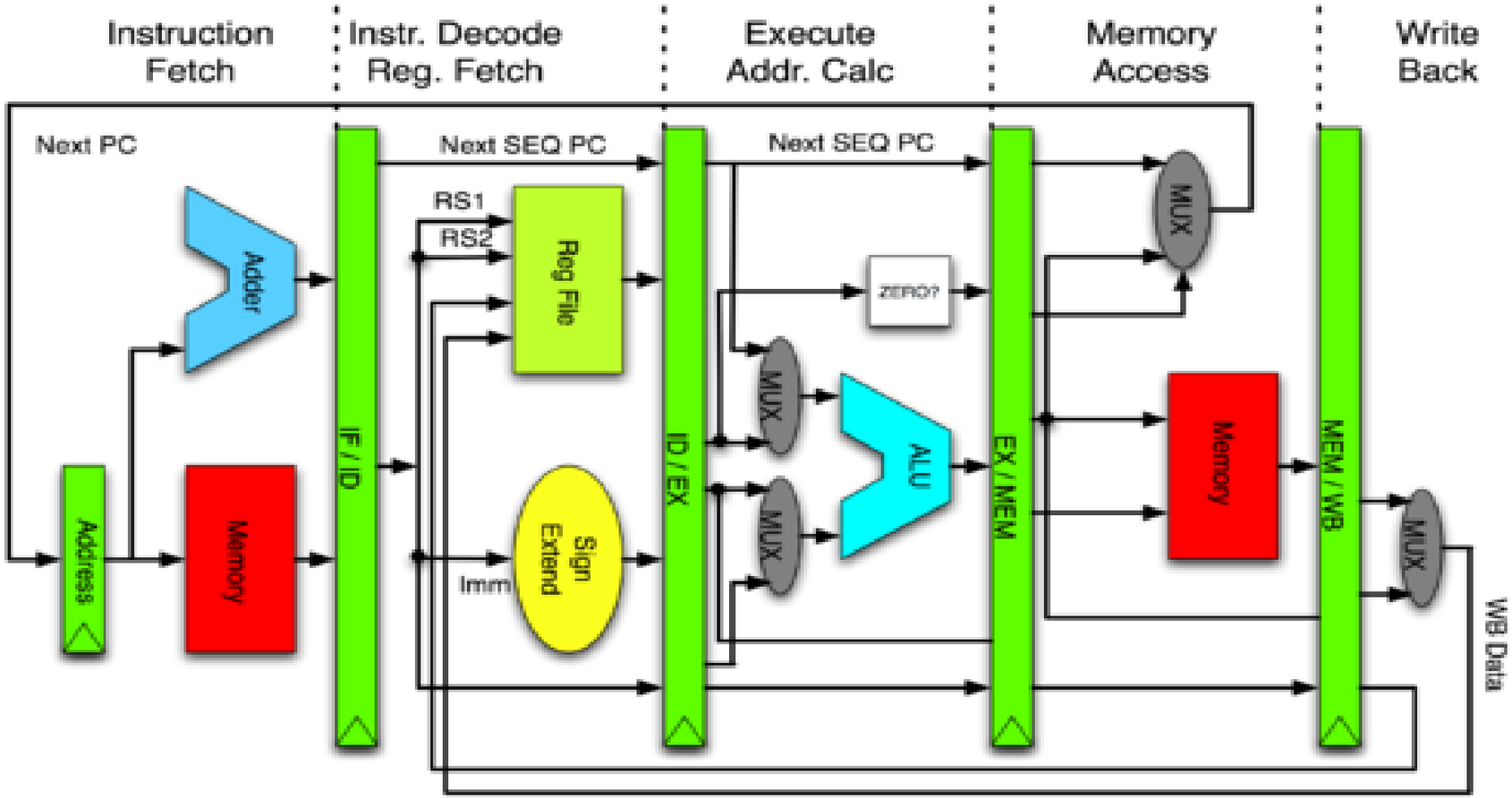
- Resource Sharing



48

Applications of Queue(Continue)

- Instruction Pipelining



49

Applications of Queue(Continue)

- **Traffic system:** In computer controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.

50