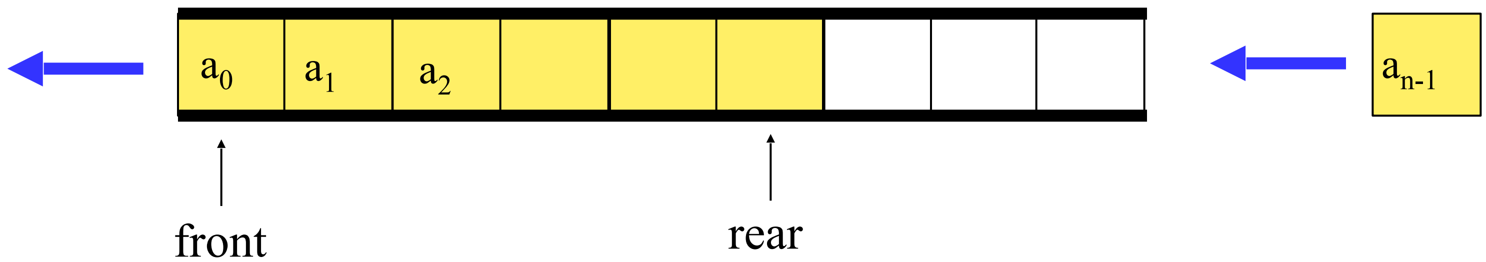
Queue

- *A* queue is an ordered list in which all insertions take place at one end and all deletions take place at the opposite end. It is also known as First-In-First-Out (FIFO) lists.



Queue: a First-In-First-Out (FIFO) list

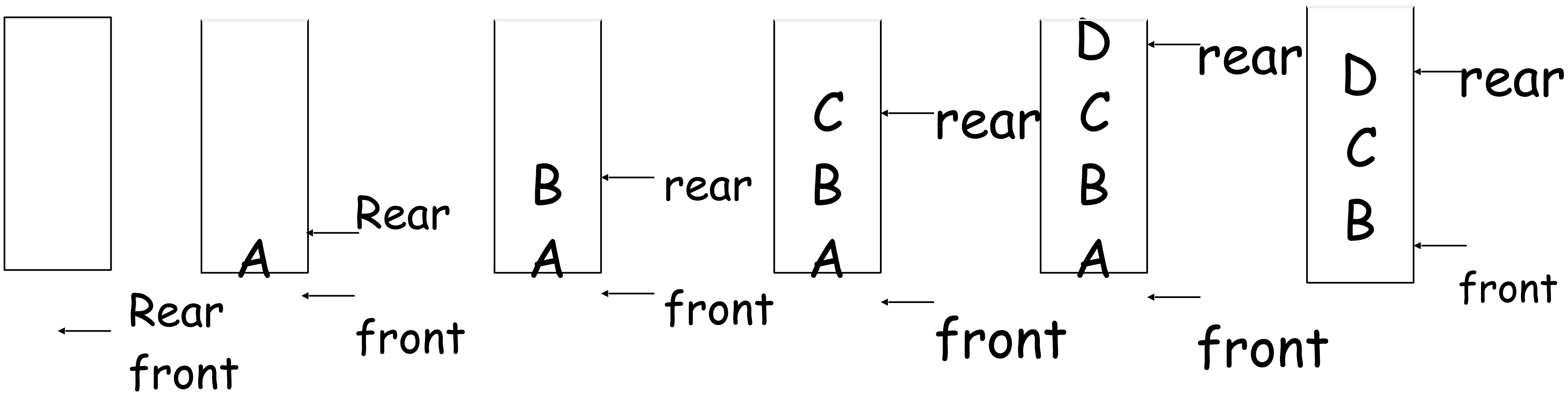


***Figure : Inserting and deleting elements in a queue**

Application: Job scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

***Figure : Insertion and deletion from a sequential queue**

Implementation 1: using array

```
# define MAX_QUEUE_SIZE 100/* Maximum queue size */

element queue[MAX_QUEUE_SIZE];

int rear = -1;
int front = -1;

int IsEmpty(){return (front == rear) ; }
int IsFull(){return (rear == MAX_QUEUE_SIZE - 1);}
        ...
```

Add to a queue

```
void Insert( int item)
{
/* add an item to the queue */
    if    (! IsFull())
        queue [++rear] = item;
    else
       printf("Queue Overflow");

}
```

***Function:** Add to a queue

Delete from a queue

```
Int  Delete()
{
/* remove element at the front of the queue */
    if  (!IsEmpty())
            return queue [++ front];
     else
     {
            printf("Queue Underflow");
            return -1
      }
}
```

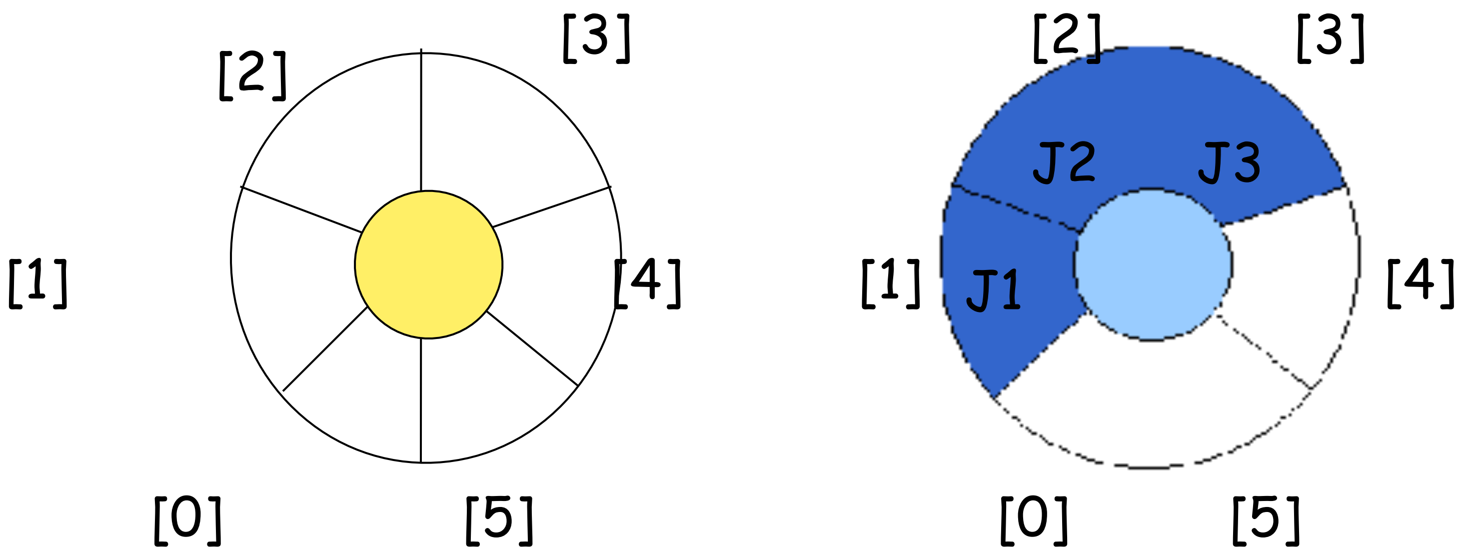
***Function:** Delete from a queue

Implementation 2: regard an array as a circular queue

front: one position counterclockwise from the first element

rear: current end

EMPTY QUEUE

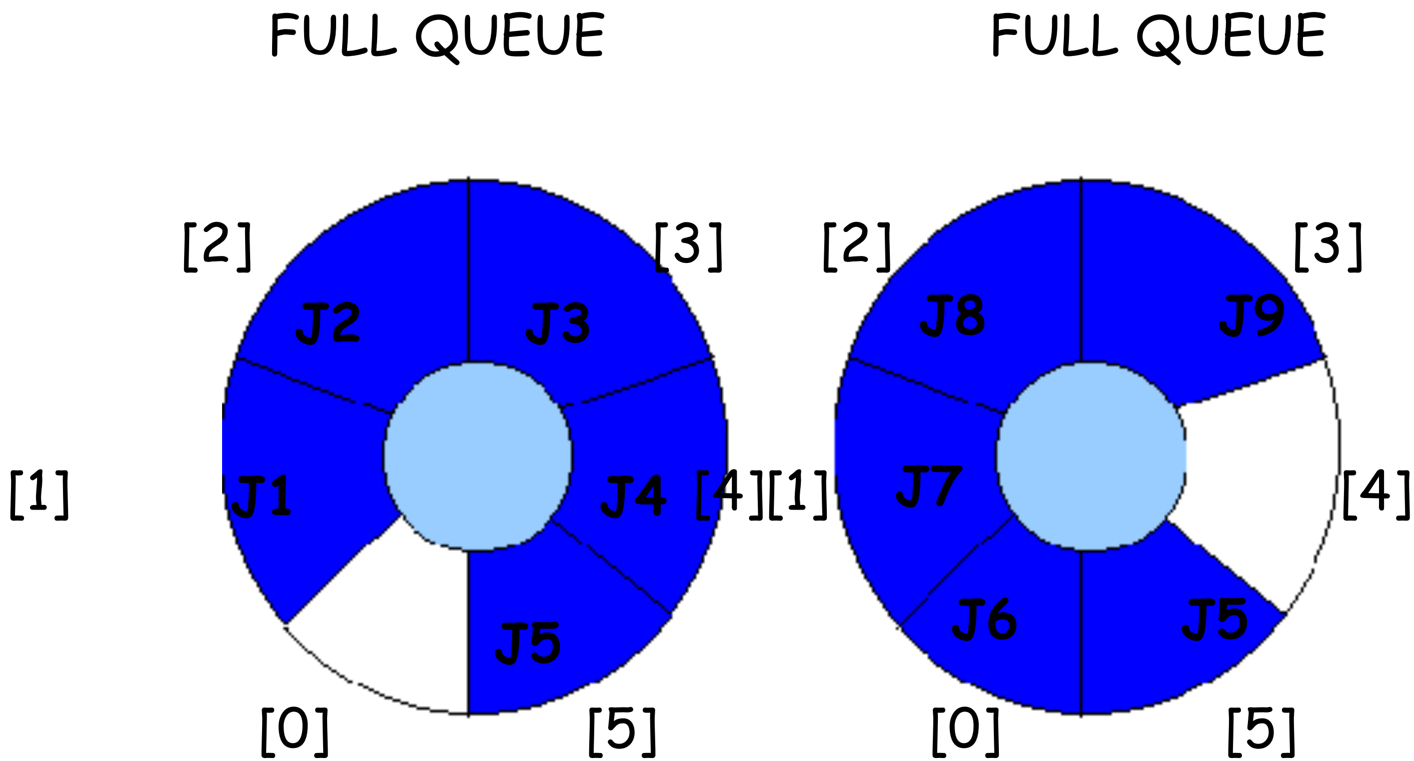


front = 0
rear = 0

front = 0
rear = 3

***Figure:** Empty and nonempty circular queues

Problem: one space is left when queue is full



front =0
rear = 5

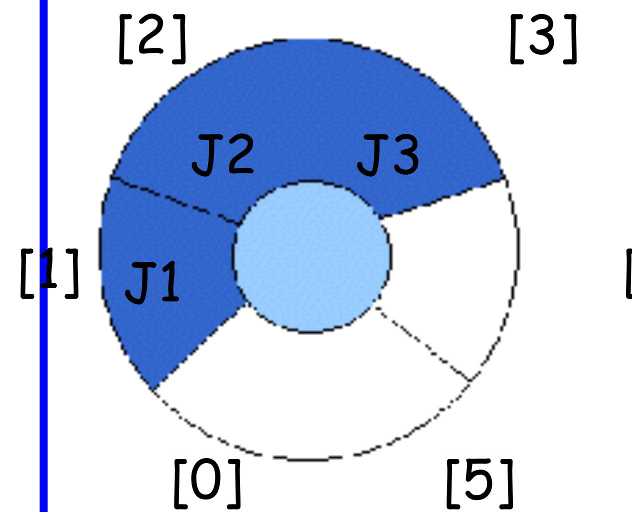
front =4
rear =3

***Figure:** Full circular queues and then we remove the item

Add to a circular queue

```
void addq(element item)
{
/* add an item to the queue */
    int k = (rear +1) % MAX_QUEUE_SIZE;
     if (front == k) /* reset rear and print error */
     {
       printf(" Q Full");
       return;
     }
      rear = k;
      queue[rear] = item;
}
```

***Function:** Add to a circular queue*



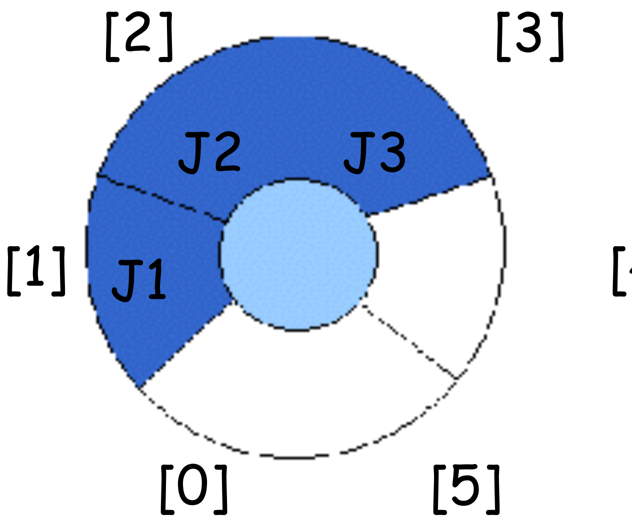
front = 0

rear = 3

Delete from a circular queue

```
element deleteq()
{
   element item;
   /* remove front element from the queue and put it
in item */
      if (front == rear)
      {
         printf(" Q Empty");
         return ERROR;
      }
              /* queue_empty returns an error key */
     front = (front+1) % MAX_QUEUE_SIZE;
     return queue[front];
}
```

***Function:** Delete from a circular queue



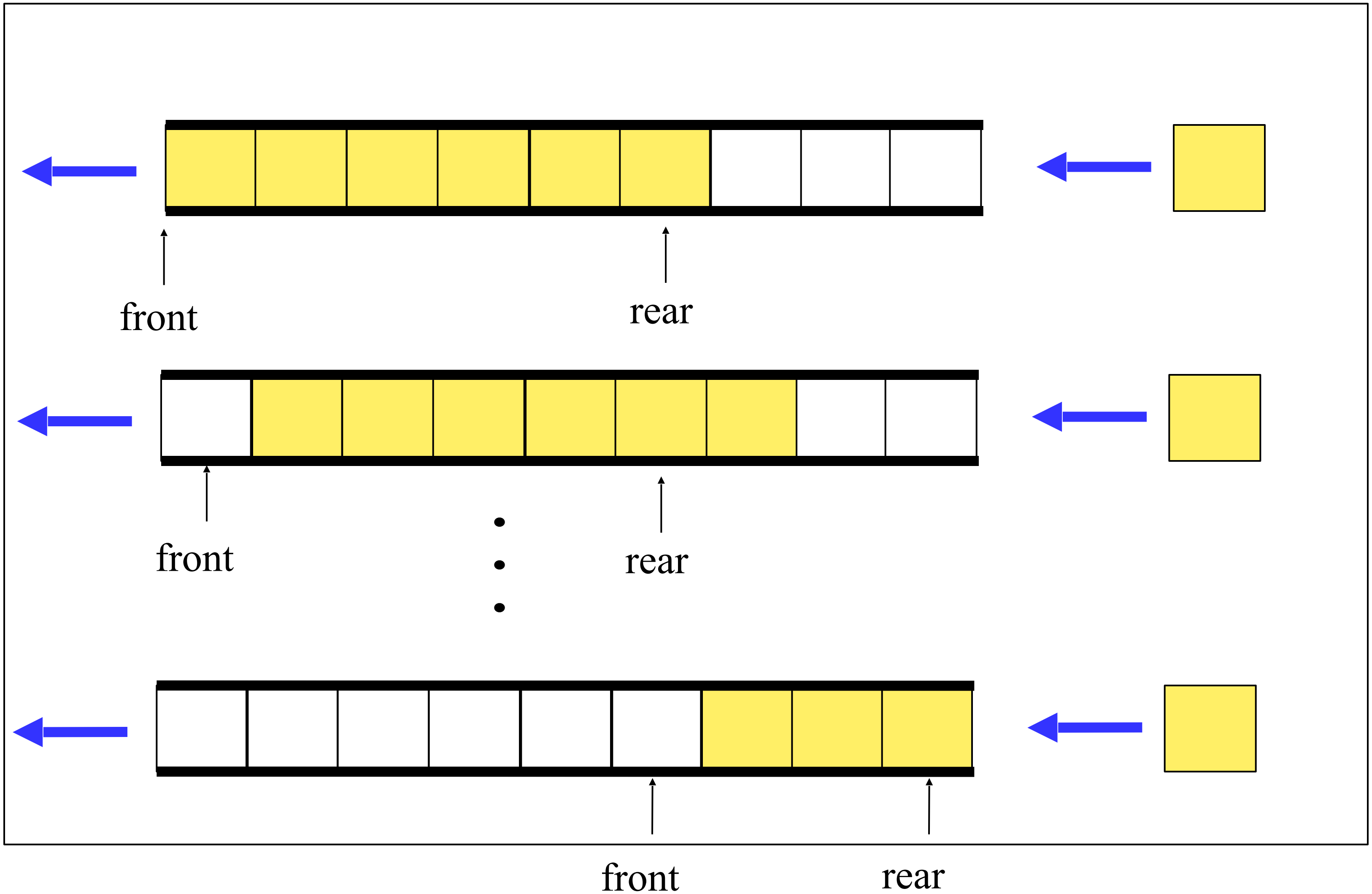
front = 0

rear = 3

Queue Manipulation Issue

- It's intuitive to use array for implementing a queue. However, queue manipulations (add and/or delete) will require elements in the array to move. In the worse case, the complexity is of O(MaxSize).

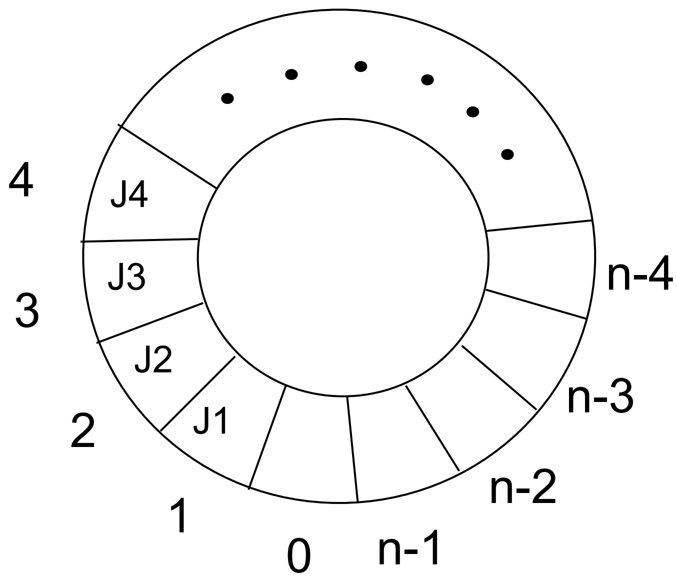
Shifting Elements in Queue



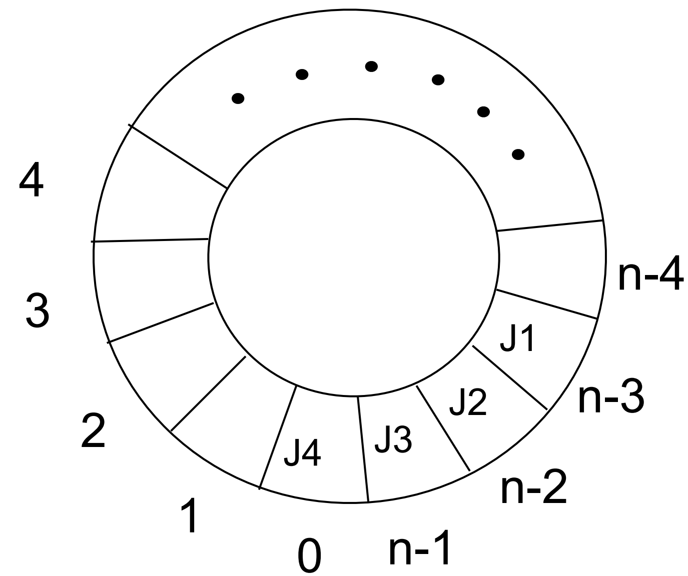
Circular Queue

- To resolve the issue of moving elements in the queue, circular queue assigns next element to q[0] when rear == MaxSize – 1.
- Pointer front will always point one position counterclockwise from the first element in the queue.
- Queue is empty when front == rear. But it is also true when queue is full. This will be a problem.

Circular Queue (Cont.)



front = 0; rear = 4



front = n-4; rear = 0

Circular Queue (Cont.)

- To resolve the issue when front == rear on whether the queue is full or empty, one way is to use only MaxSize – 1 elements in the queue at any time.
- Each time when adding an item to the queue, newrear is calculated before adding the item. If newrear == front, then the queue is full.
- Another way to resolve the issue is using a flag to keep track of last operation. The drawback of the method is it tends to slow down Add and Delete function.