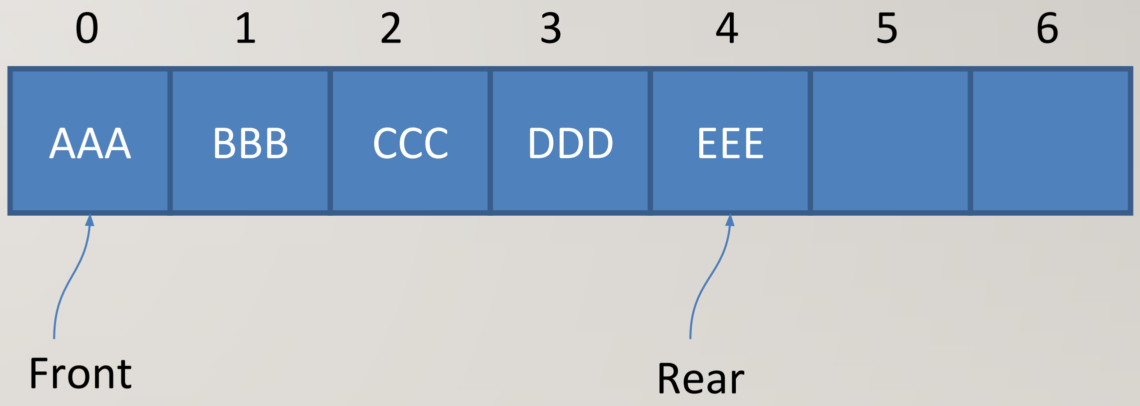
QUEUE DATA STRUCTURE

- LINEAR QUEUE
- CIRCULAR QUEUE

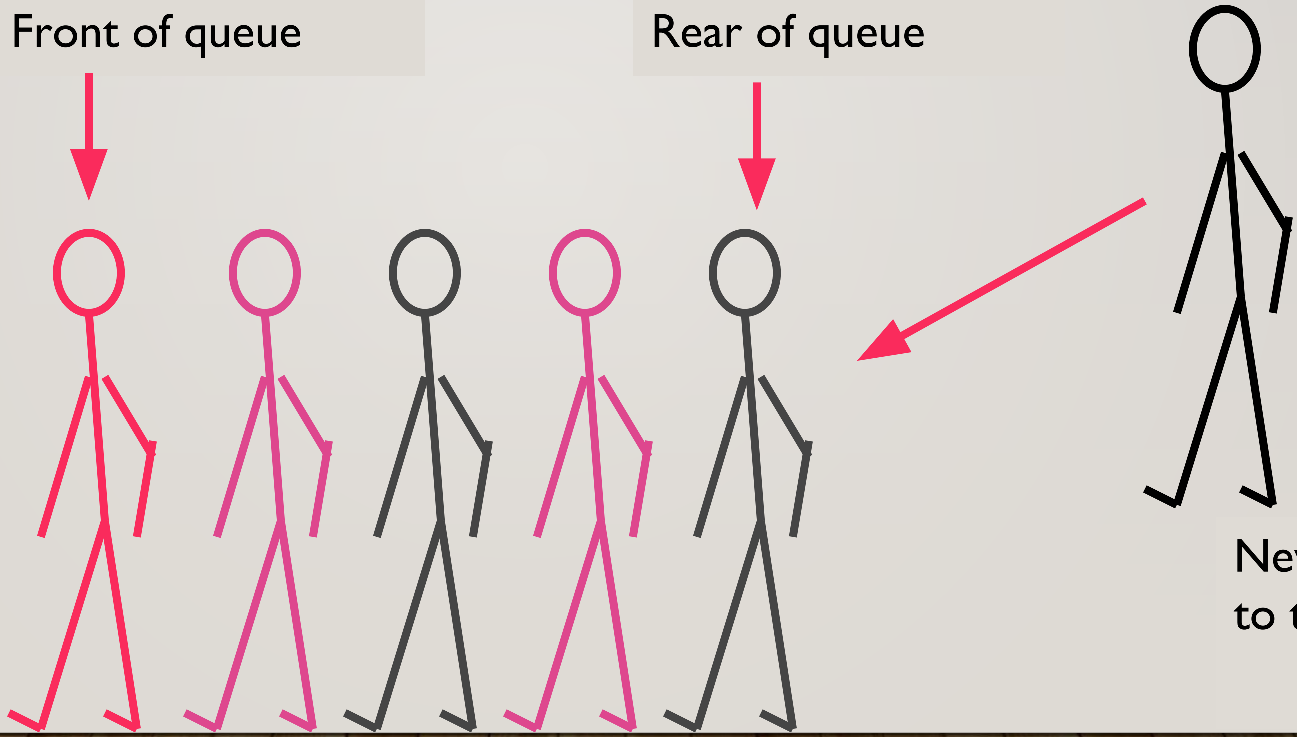
QUEUE DATA STRUCTURE

- **Queue**:
  - a First In, First Out (***FIFO)*** data structure
  - a collection
    - whose elements are added at one end (the ***rear*** or ***tail*** of the queue)
    - and removed from the other end (the ***front*** or ***head*** of the queue)
  - Any waiting line is a queue:
    - The check-out line at a grocery store
    - The cars at a stop light
    - An assembly line



QUEUE - INSERTION VIEW

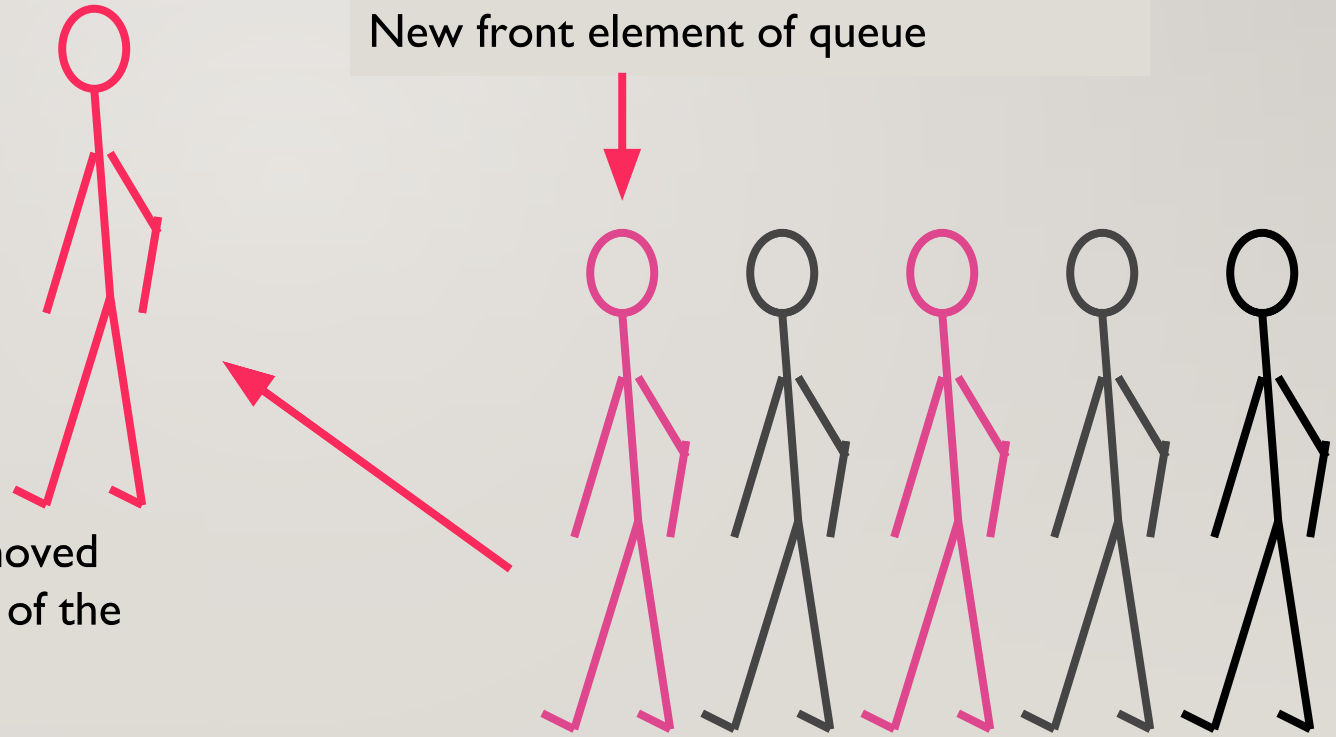
Adding an element



New element is added to the rear of the queue

QUEUE – DELETION VIEW

Removing an element



New front element of queue

Element is removed from the front of the queue

APPLICATIONS OF QUEUE

- For any kind of problem involving FIFO data
- Printer queue
- Keyboard input buffer
- GUI event queue (click on buttons, menu items)

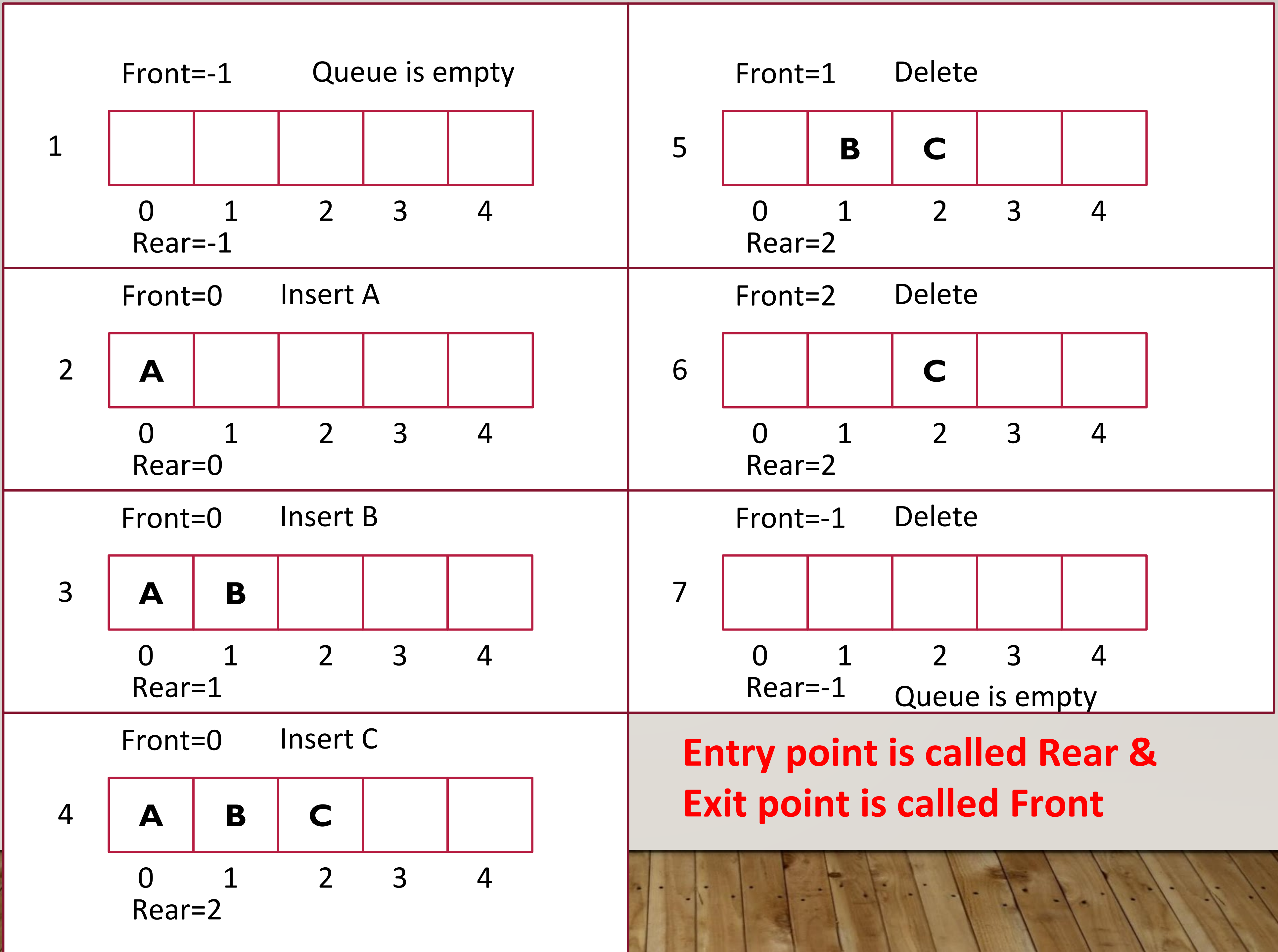
QUEUE AS ADT(ABSTRACT DATA TYPE)

Elements of a Queue

- Array
- Front
- Rear

Operations on a Queue

Operation	Description
isFull	Determines whether the queue is full
enqueue	Adds an element to the rear of the queue
isEmpty	Determines whether the queue is empty
dequeue	Removes an element from the front of the queue
display	Prints the values of queue



Reset

Entry point is called Rear &
Exit point is called Front

LINEAR QUEUE IMPLEMENTATION USING ARRAY/LIST

Initialization:

front =-1, rear = -1

ISFULL FUNCTION

//returns true, if queue is full and false, otherwise

//capacity: Capacity of the queue (List size)

```
Algorithm isFull()
    return (rear == capacity-1)
```

ENQUEUE FUNCTION

```
Algorithm enQueue(value )
  if (isFull() )
          print(”Error: Overflow”)
   else
    if(front==-1)
        front=0
    rear += 1
    queue[rear] = value
```

LINEAR QUEUE IMPLEMENTATION USING ARRAY/LIST

ISEMPTY FUNCTION

//returns true, if queue is empty, false otherwise

```
Algorithm isEmpty()
    return (front == -1)
```

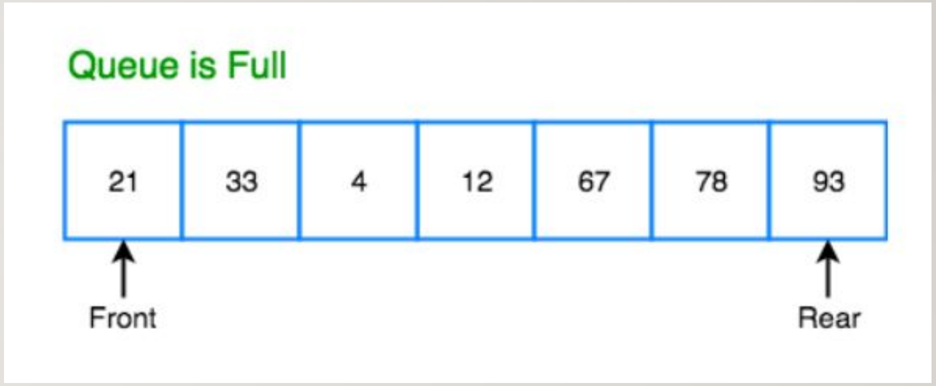
DEQUEUE FUNCTION

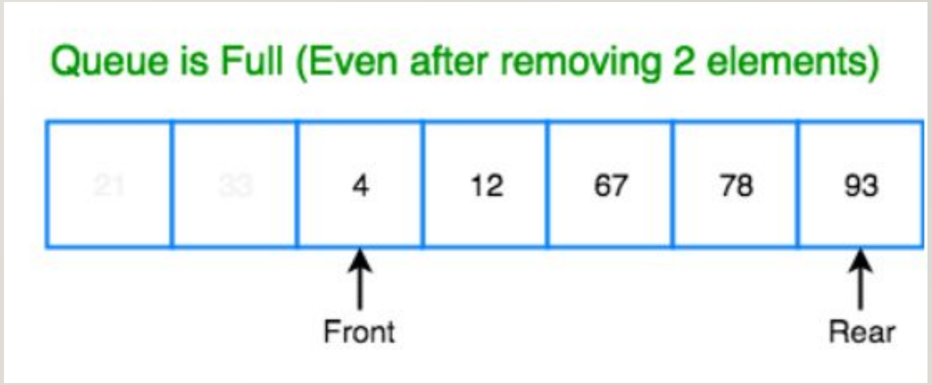
```
Algorithm deQueue( )
  if(isEmpty())
    print("Queue  underflow")
    return -1
  else
    x=queue[front]
    if (front==rear)
        front=rear=-1
    else
        front++
    return x
```

Issue in Linear Queue DS

In a Linear queue, once the queue is completely full, it's not possible to insert more elements.

Even if we dequeue the queue to remove some of the elements, **until the queue is reset, no new elements can be inserted.**





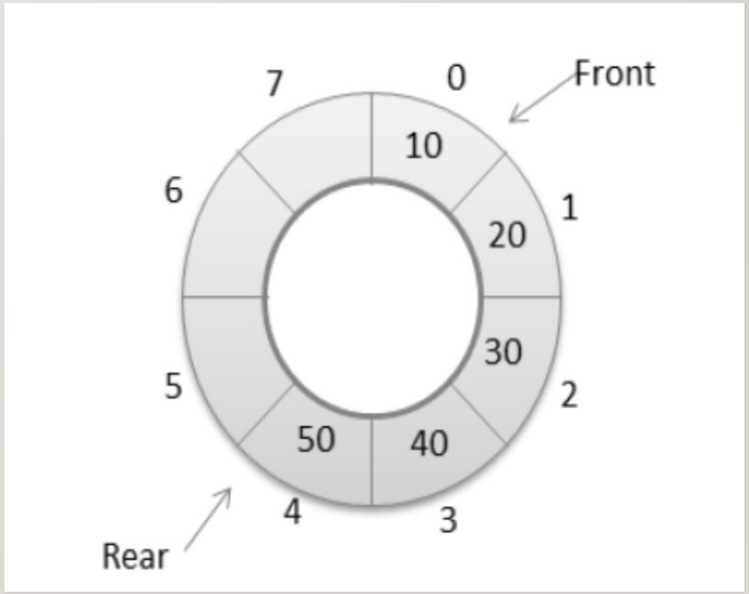
INTRODUCTION TO CIRCULAR QUEUE

DRAWBACK OF LINEAR QUEUE

- Once the the rear has reached the Queue's rear most position, even though few elements from the front are deleted, it is not possible to add anymore new elements

SOLUTION TO OVERCOME THE DRAWBACK

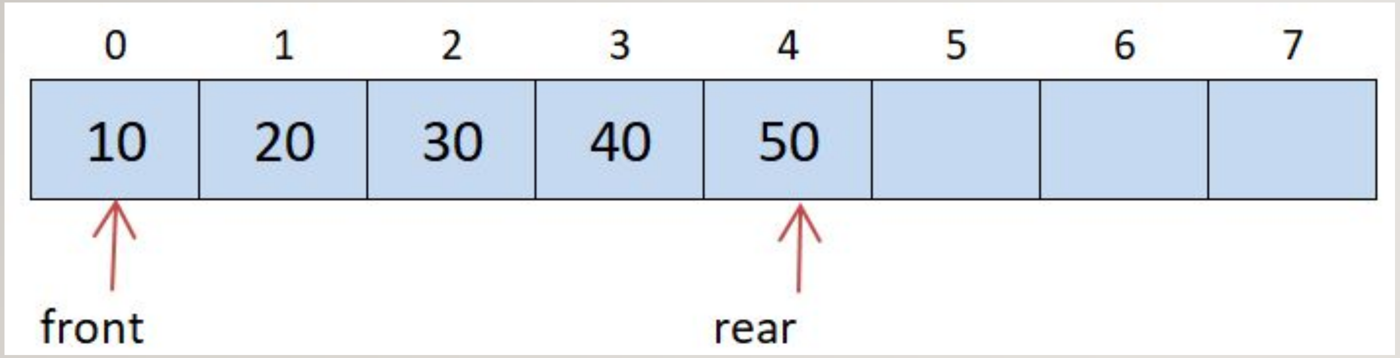
- Circular Queue
  - also called as "Ring buffer".

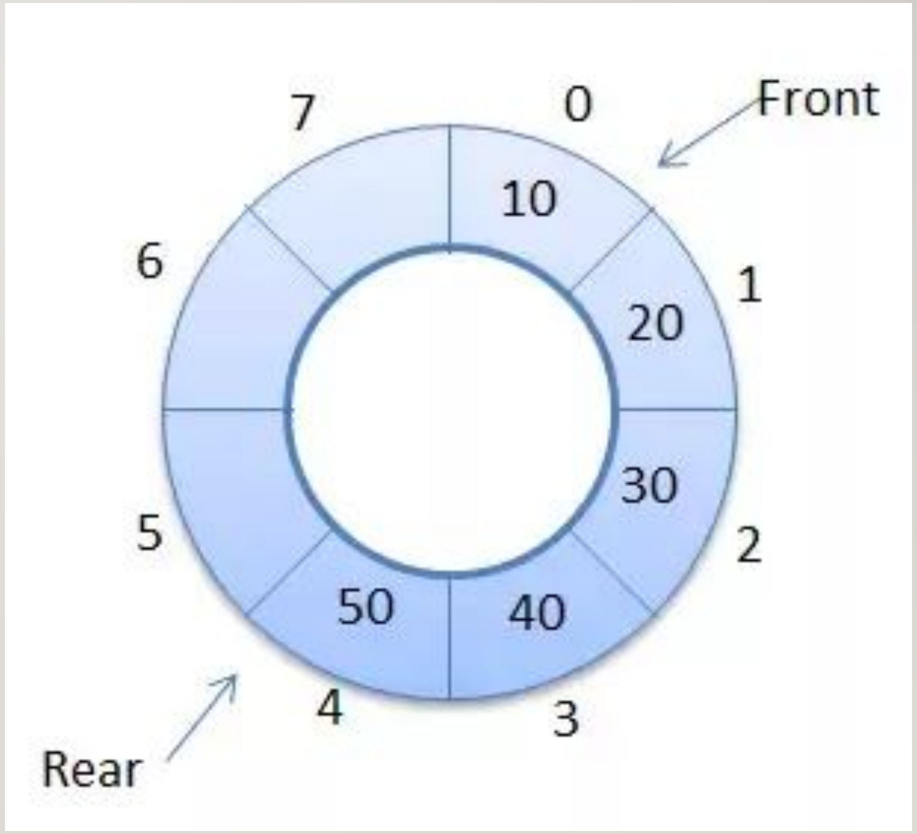


CIRCULAR QUEUE

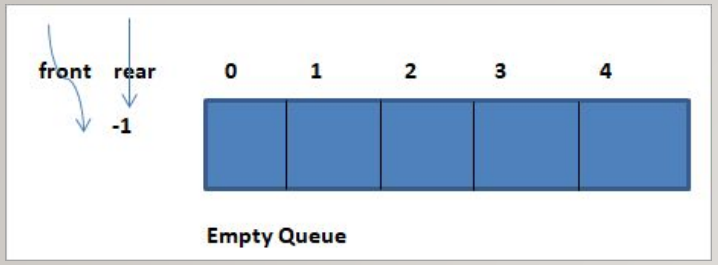
- After rear reaches the last position, i.e. capacity-1, in order to reuse the vacant positions,
    - bring rear back to the 0th position, if it is empty, and continue incrementing rear in same manner as earlier. Thus rear will have to be incremented circularly.
    - Similarly for deletion, front will also have to be incremented circularly…

CIRCULAR QUEUE

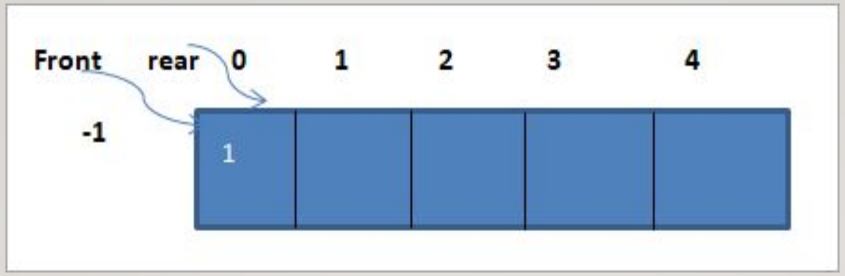




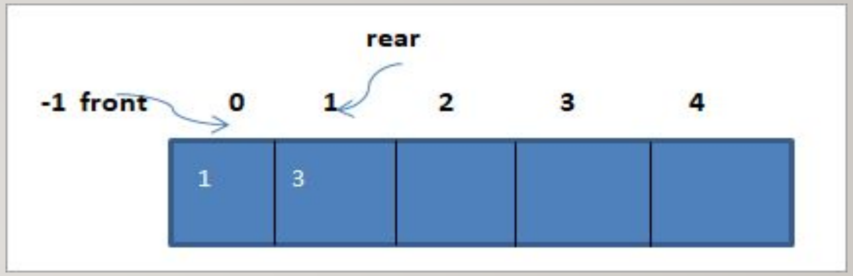
Empty Queue (Capacity = 5)



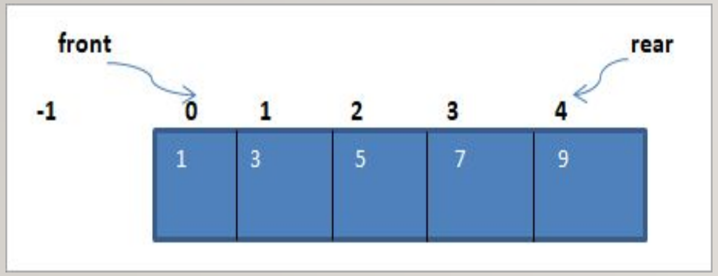
Enqueue 1



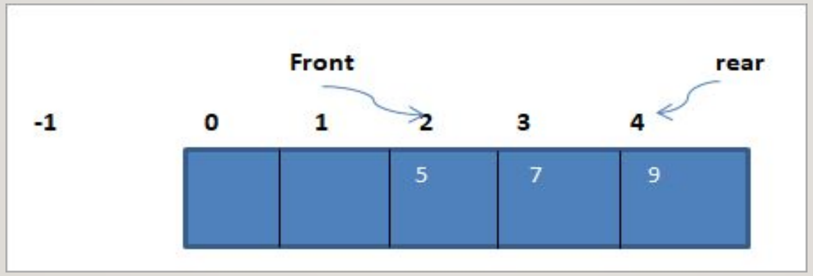
Enqueue 3



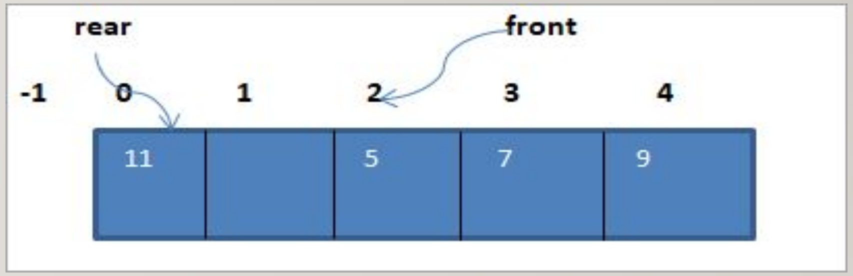
Enqueue 5, 7, 8



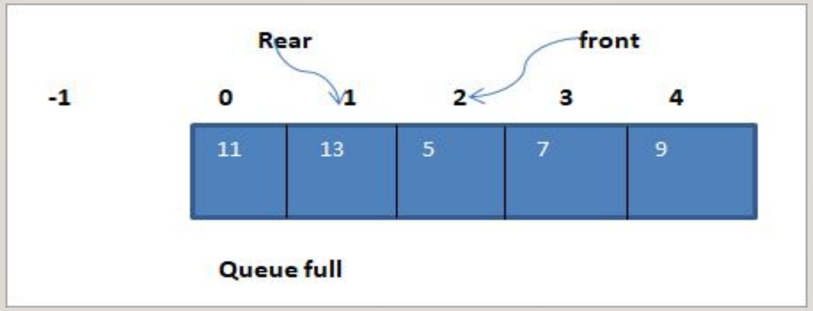
Dequeue two elements

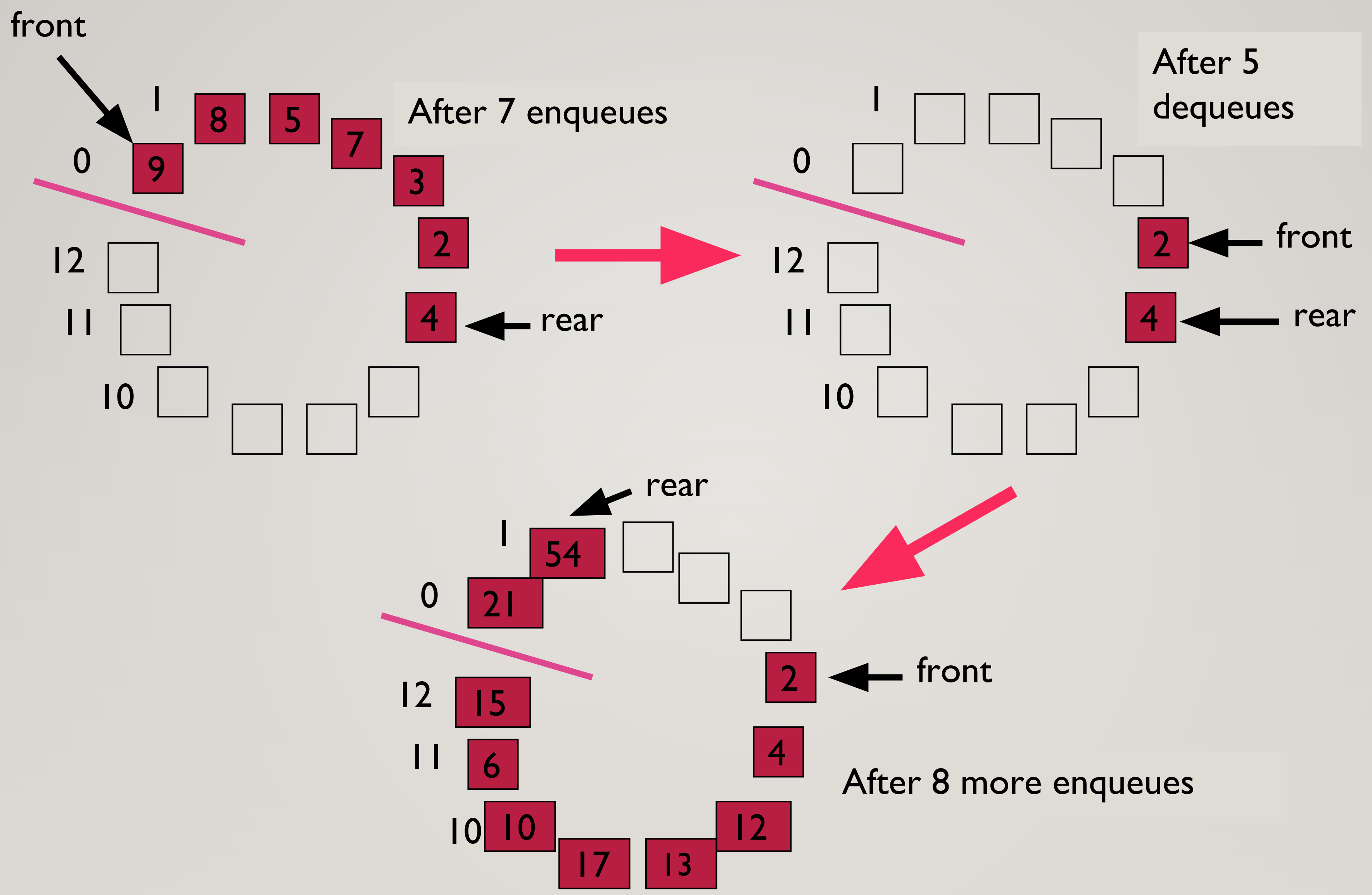


Enqueue 11



Enqueue 13





CONCEPTUAL EXAMPLE OF A CIRCULAR QUEUE

CIRCULAR QUEUE IMPLEMENTATION USING ARRAY/LIST

Inititalization:

front =-1, size = 0, rear = -1

ISFULL FUNCTION

//returns true, if queue is full and false, otherwise
//capacity: Capacity of the queue (List size)
//size: number of elements in queue

```
Algorithm isFull()
    return (size == capacity)
```

ENQUEUE FUNCTION

```
Algorithm enQueue(value )
    if (isFull())
        print("Full")
        return
    rear = (rear + 1) % capacity
    Q[self.rear] = value
    size = size + 1
```

CIRCULAR QUEUE IMPLEMENTATION USING ARRAY/LIST

ISEMPTY FUNCTION

//returns true, if queue is empty and false, otherwise

```
Algorithm isEmpty()
    return (size == 0)
```

DEQUEUE FUNCTION

```
Algorithm deQueue( )
  if (isEmpty())
    print(“Empty Queue")
    return
  x = Q[front]
  front = (front + 1) % capacity
  size = size -1
```