Data Structure
Algorithms & Applications

CT-159

Prepared by
Muhammad Kamran

Queue (in real world)

Queue

- Queue is a data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue)
- Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue Representation

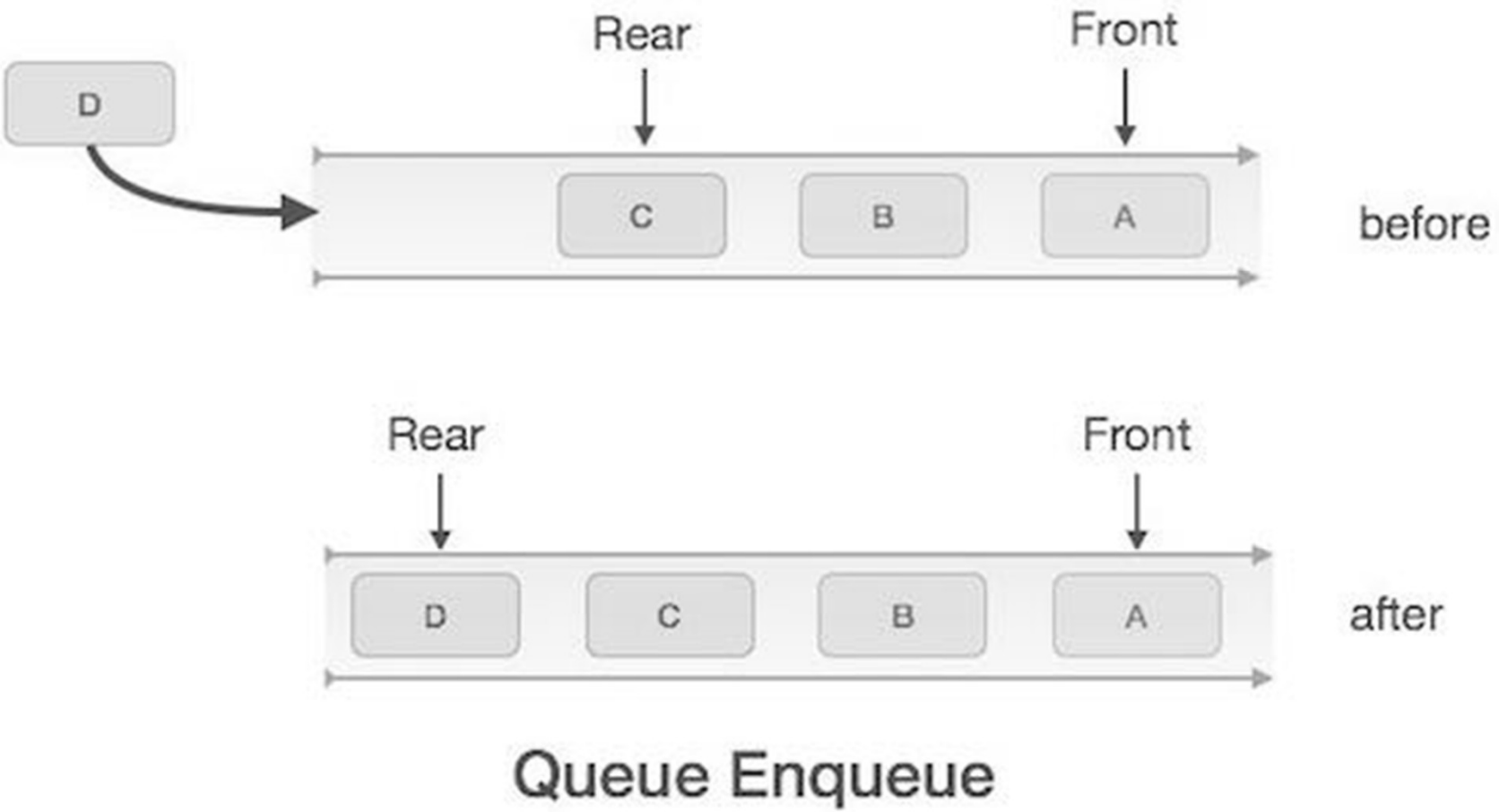
Queue (basic operations)

- **enqueue()** – add (store) an item to the queue.
- **dequeue()** – remove (access) an item from the queue.
- **peek()** – Gets the element at the front of the queue without removing it.
- **isfull()** – Checks if the queue is full.
- **isempty()** – Checks if the queue is empty.

Enqueue Operation

- **Step 1** – Check if the queue is full.
- **Step 2** – If the queue is full, produce overflow error and exit.
- **Step 3** – If the queue is not full, increment **rear** pointer to point the next empty space.
- **Step 4** – Add data element to the queue location, where the rear is pointing.
- **Step 5** – return success.

Enqueue Operation



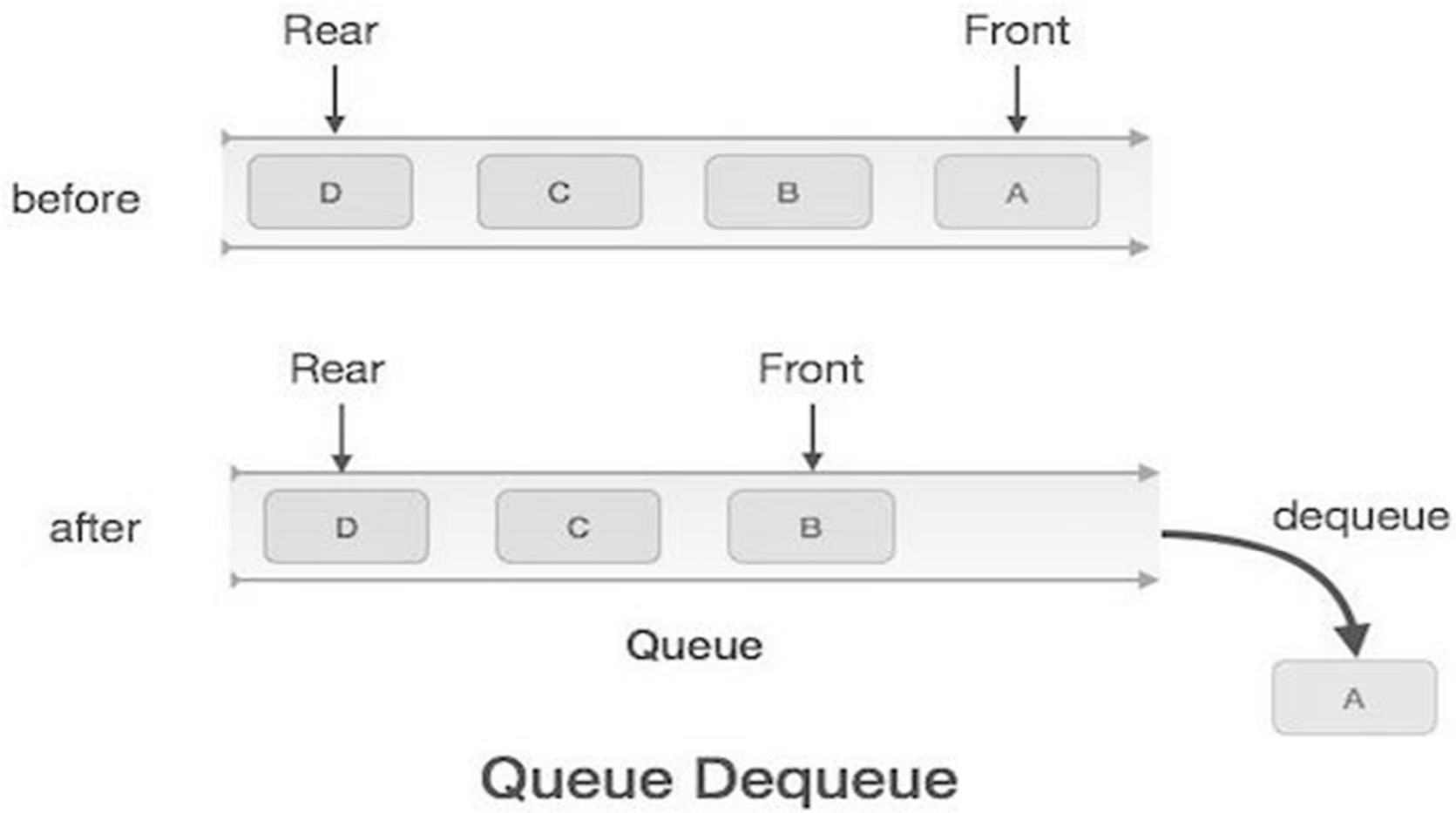
Algorithm for ENQUEUE Operation

```
procedure enqueue(data)
        if queue is full
                return overflow
        endif
        rear ← rear + 1
        queue[rear] ← data
        return true
end procedure
```

Dequeue Operation

- **Step 1** – Check if the queue is empty.
- **Step 2** – If the queue is empty, produce underflow error and exit.
- **Step 3** – If the queue is not empty, access the data where **front** is pointing.
- **Step 4** – Increment **front** pointer to point to the next available data element.
- **Step 5** – Return success.

Dequeue Operation



Queue Dequeue

Algorithm for dequeue operation

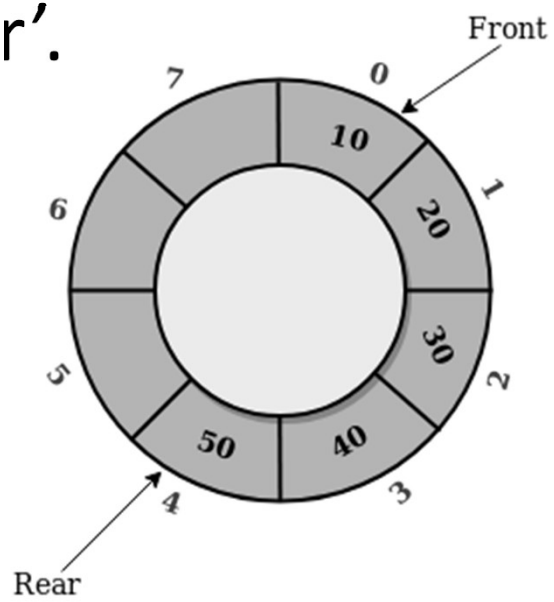
```
procedure dequeue
        if queue is empty
                return underflow
        end if
        data = queue[front]
        front ← front + 1
        return true
end procedure
```

Application of Queue ??

- When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
- 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 can be used as an essential component in various other data structures.

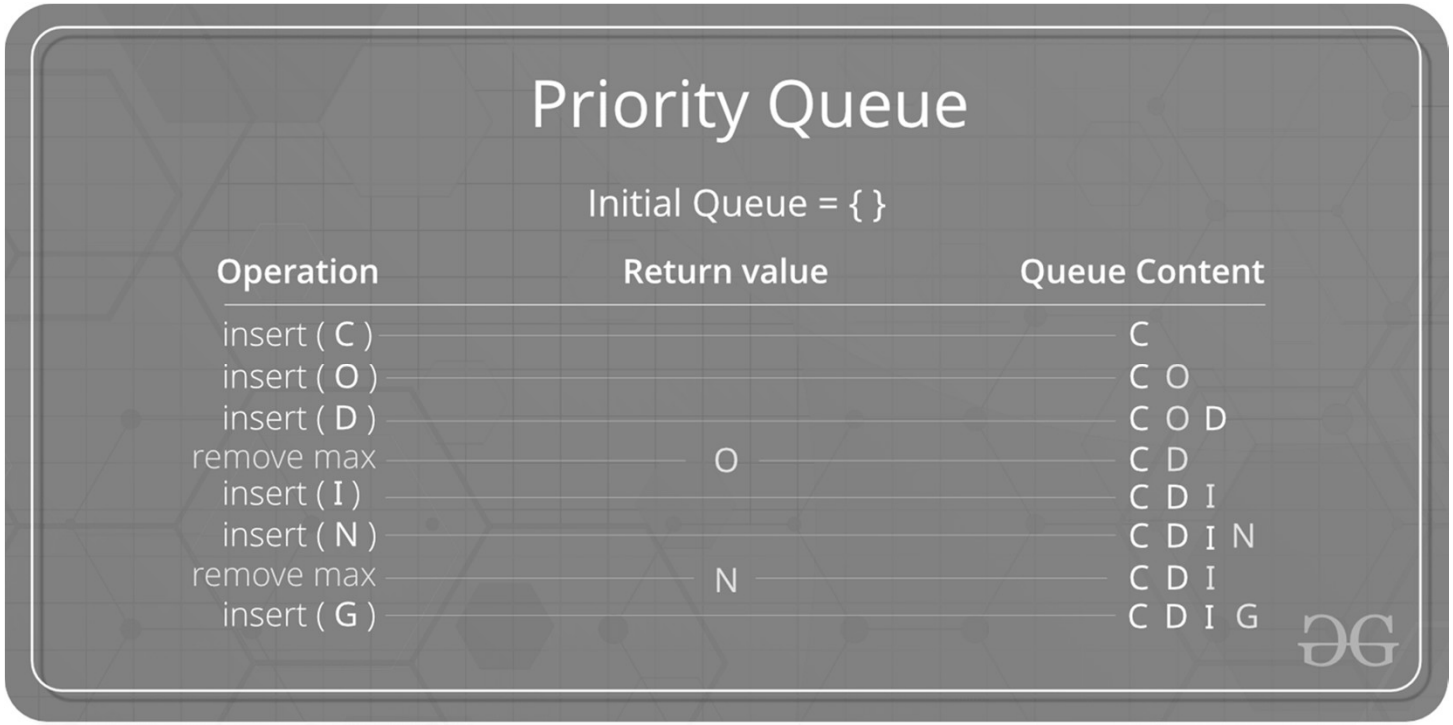
Circular Queue

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called ‘Ring Buffer’.



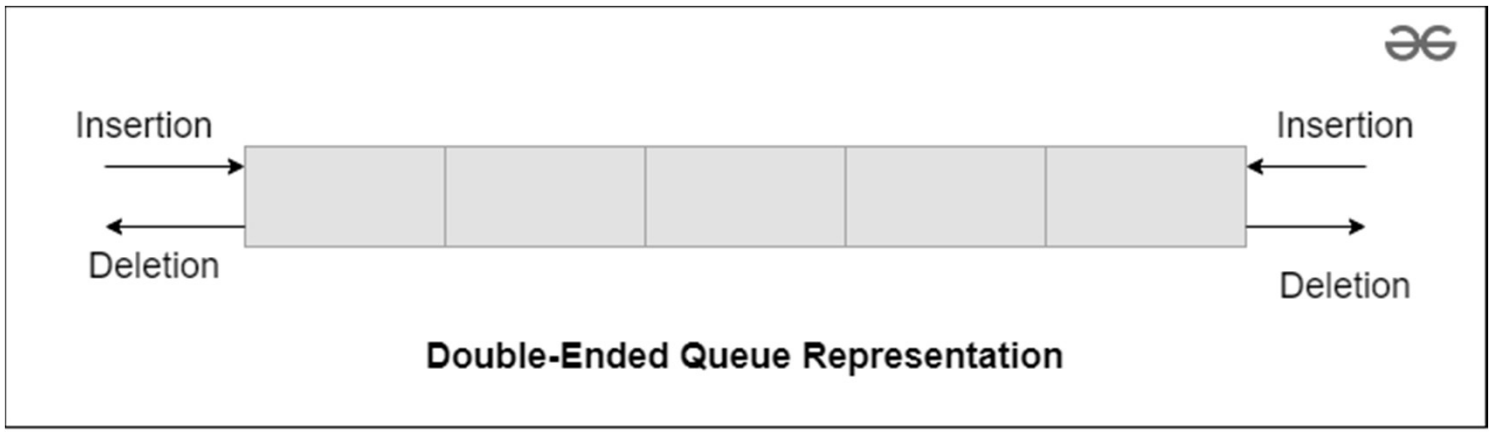
Priority Queue

A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority.



Double Ended Queue

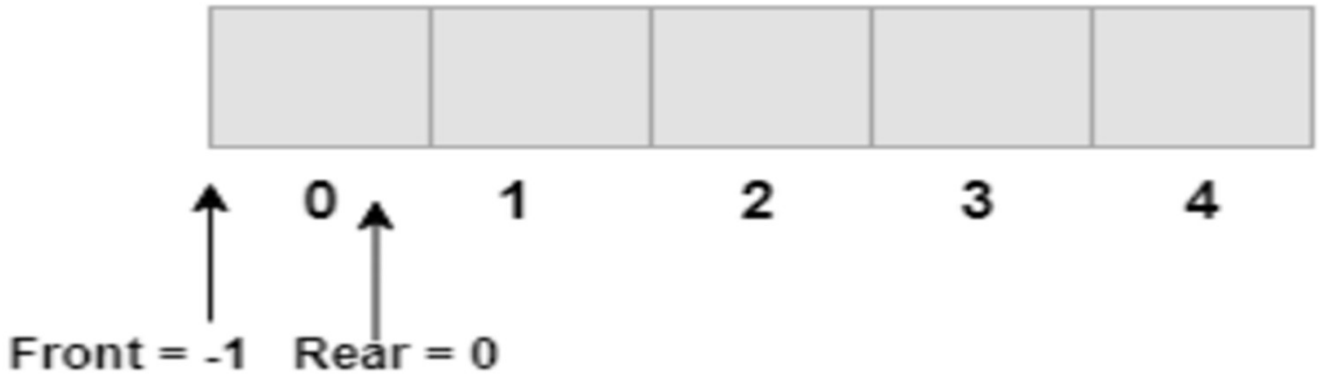
A **Deque** is a linear collection of elements that supports insertion and deletion at both ends. The name “deque” is short for “double-ended queue”. It can be thought of as a combination of a stack and a queue that allows insertion and deletion of elements from both ends.



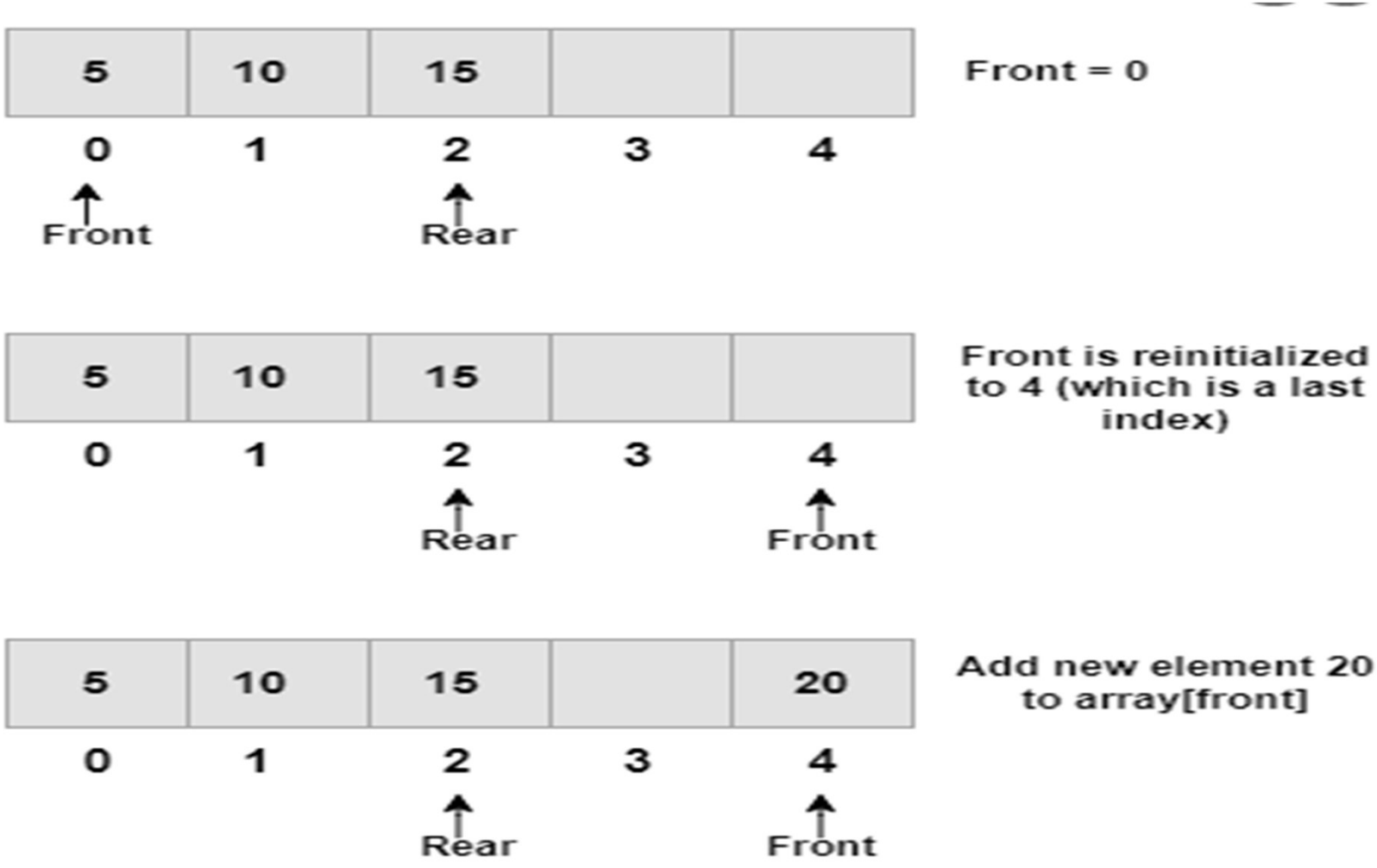
Properties of Deques

- Elements can be added to both the front and rear ends of the queue.
- Elements can be removed from both the front and rear ends of the queue.
- Deques support both LIFO (Last In First Out) and FIFO (First In First Out) operations.
- Random access is generally not allowed.
- Deques can be implemented using either arrays or linked lists.

Representation of Double Ended Queue

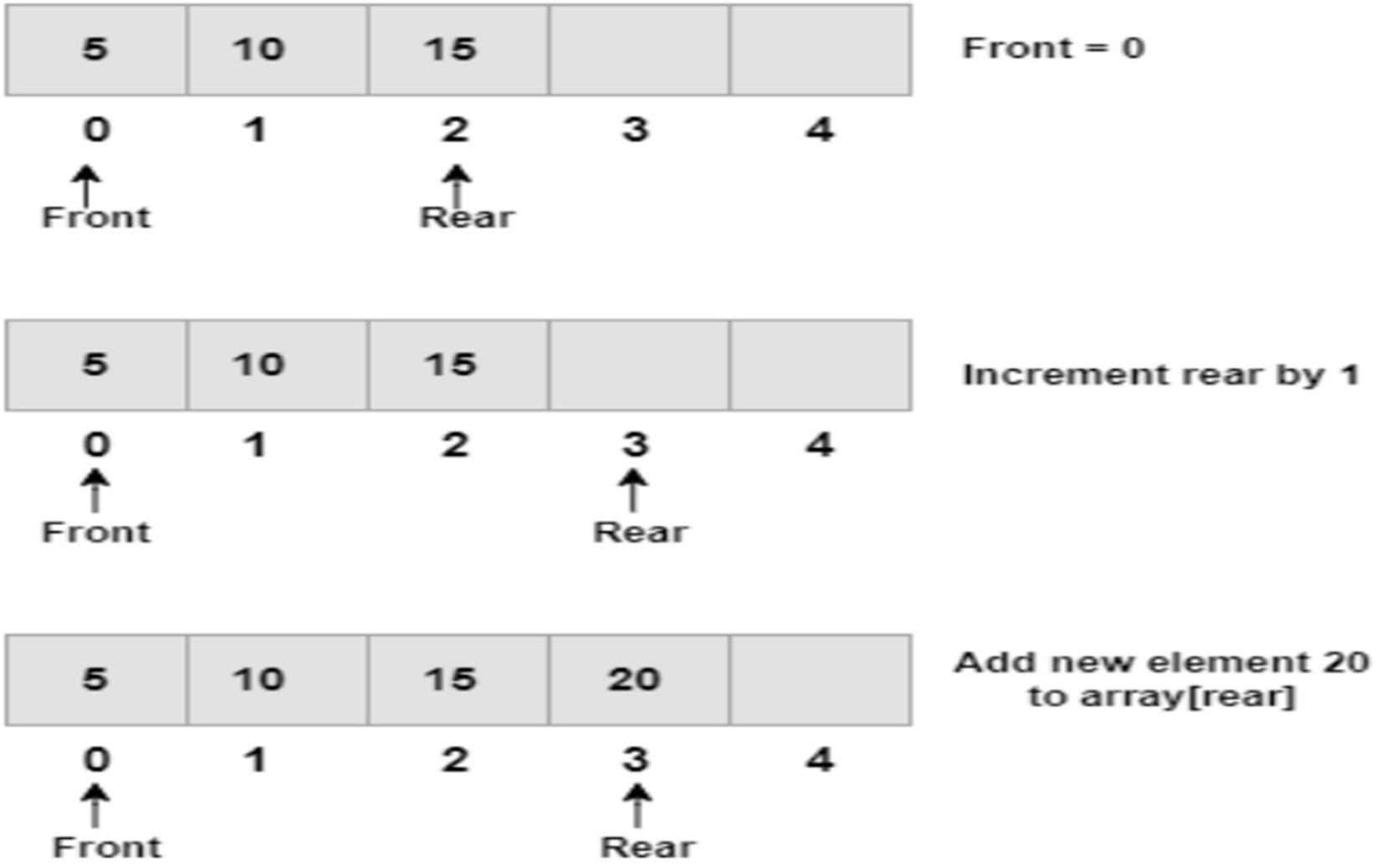


Insertion at Front in Deque



Insertion at Front in Deque

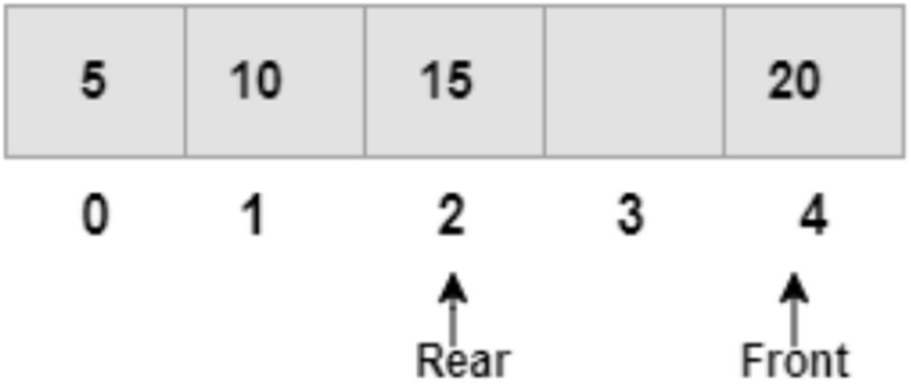
Insertion at Rear in Deque



Insertion at Rear in Deque

Deletion at Front in Deque

Case 1:



Front is at last index

Delete the element 20 and set front = 0 (starting index)

Deletion at Front in Deque

Deletion at Front in Deque

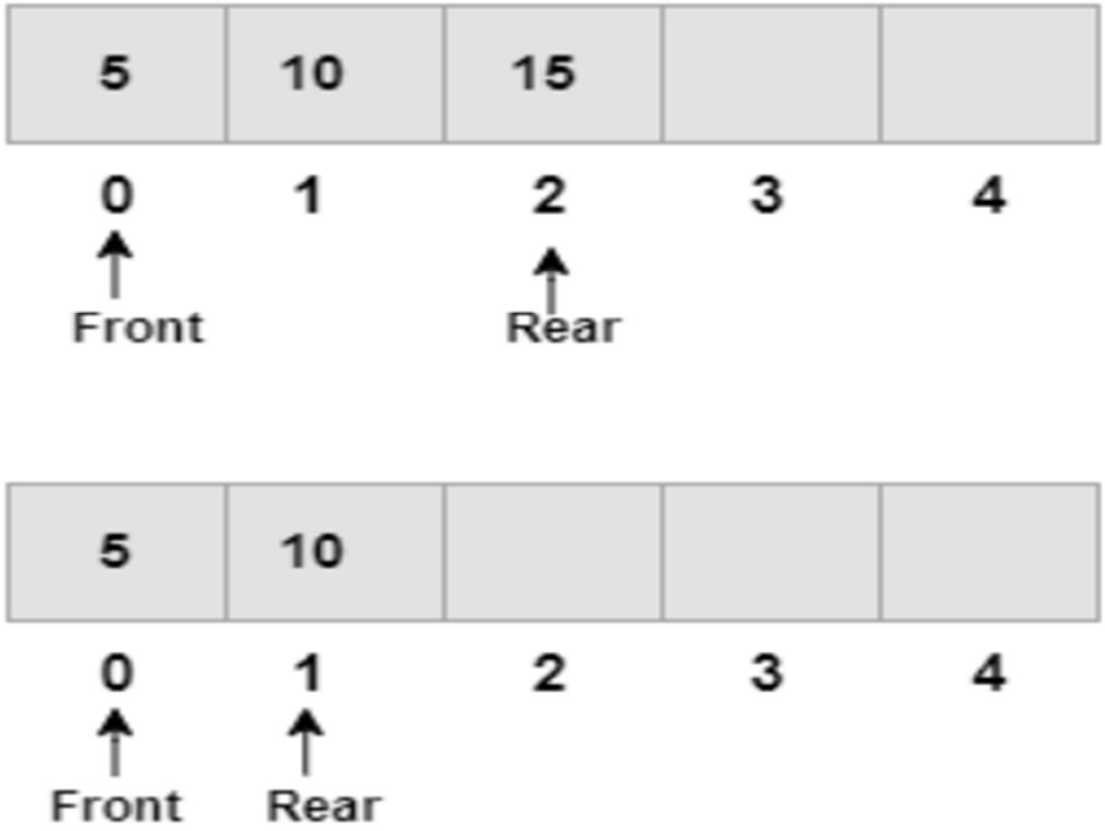
Case 2:

Front at any Index

Delete the element 5 at front and set front = front + 1

Deletion at Front in Deque

Deletion at Rear in Deque



Deletion at Rear in Deque

Thank You