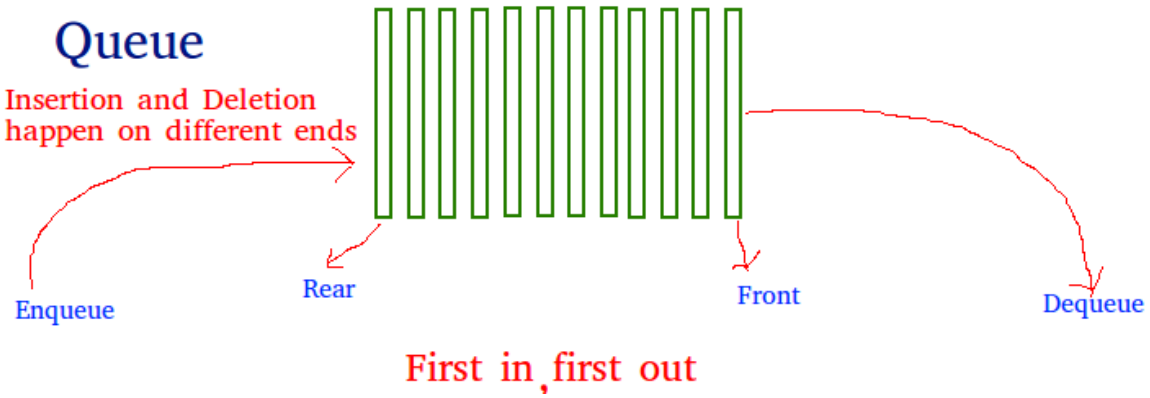
Queue :

- Queue is an abstract data structure
- It is linear and sequential
- It is open at both its ends.
- One end is always used to insert data (enqueue)…Rear end of queue
- Another end is used to remove data (dequeue)…. Front End of the queue
- Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

- A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Queue using Array :



Queue Operations

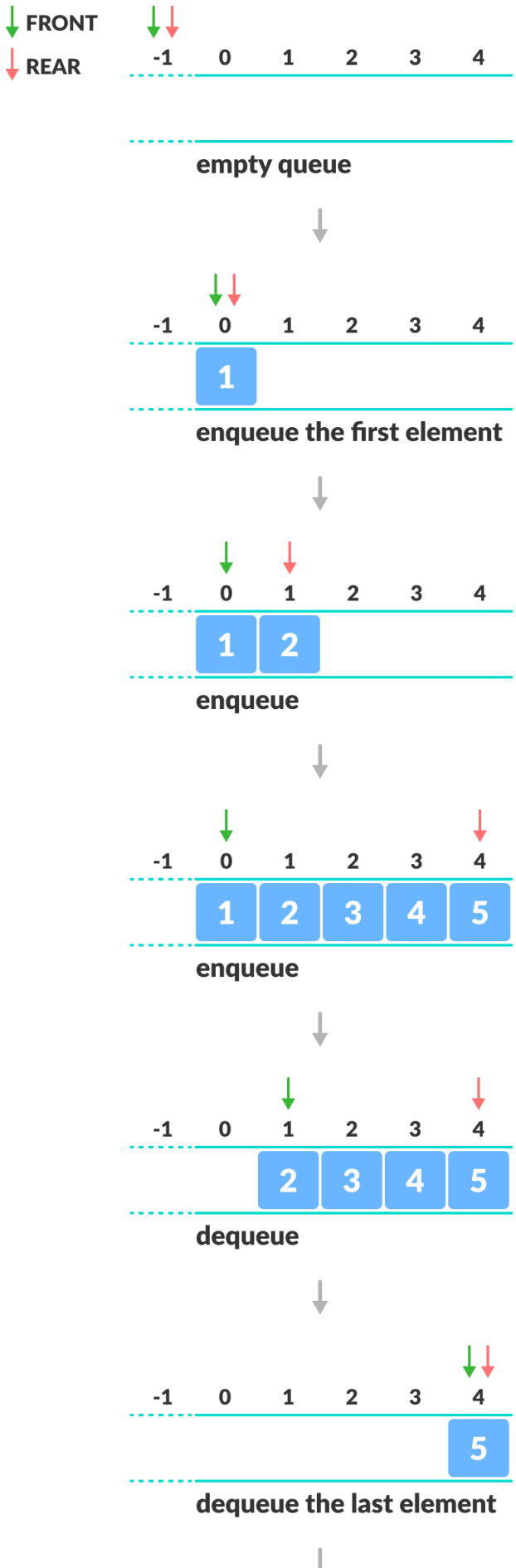
Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory.

Basic operations

- **En(ter)queue()** − add (store) an item to the queue.
- **De(lete)queue()** − remove (access) an item from the queue.

Supportive Operations

- **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.


FRONT
REAR
-1 0 1 2 3 4
empty queue
-1 0 1 2 3 4
1
enqueue the first element
-1 0 1 2 3 4
1 2
enqueue
-1 0 1 2 3 4
1 2 3 4 5
enqueue
-1 0 1 2 3 4
2 3 4 5
dequeue
-1 0 1 2 3 4
5
dequeue the last element

peek()

- This Function helps to see the data at the **front** of the queue.

Algorithm

```
Peek()
Begin
1.  Return queue[front]
end
```

isfull()

- To check for the full queue, check if rear pointer reached at MAXSIZE

Algorithm

```
Isfull()
Begin
1. if rear = MAXSIZE
     return true
   else
     return false
End
```

Isempty()

```
begin
1.    if front < MIN  OR front > rear
//      Let Front = rear = -1
      return true
   else
     return false
   endif
end
```

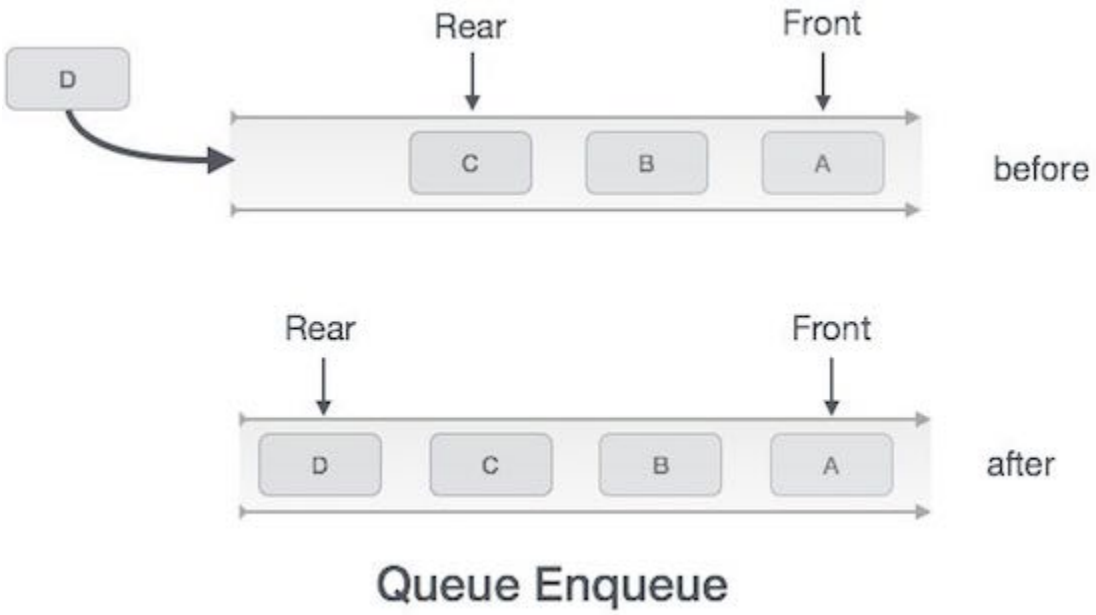
```
Begin
1. If (front = rear) = -1
      return true
   else
      return false
   endif
end
```

enqueue()

- check if the queue is full
- If Queue is full proceed for overflow and retuen 0
- for the first element, set value of FRONT to 0
- increase the REAR index by 1
- add the new element in the position pointed to by REAR

```
enqueue( data)
begin
 1.  if(isfull())
            then  return 0;
 2   rear = rear + 1;
 3   queue[rear] = data;
//3.a if front = -1 then front = 0;
 4   return 1;
 end
```

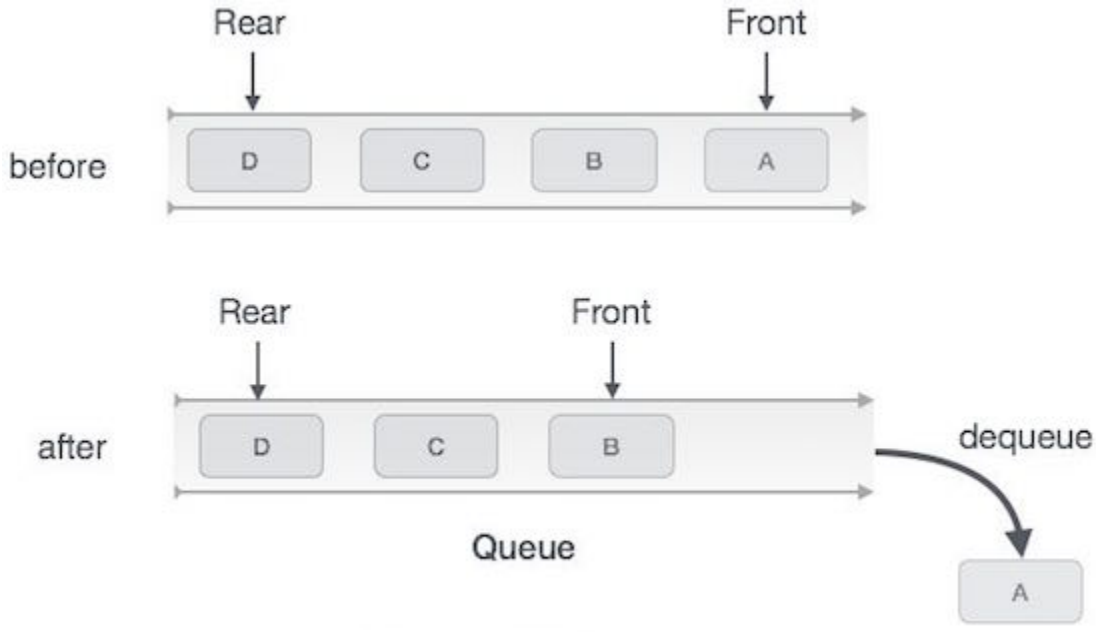
You may check Empty (front = rear = -1) then special case entry
Front = front +1
Rear = real+1
Queue[rear] = data



Queue Enqueue

Dequeue()

- check if the queue is empty
- If empty proceed for under flow and return false
- return the value pointed by FRONT
- increase the FRONT index by 1
- for the last element, reset the values of FRONT and REAR to -1

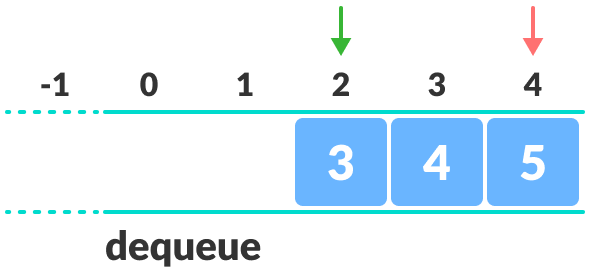


Queue Dequeue

```
procedure dequeue
Begin
1.   if (isempty()) //queue is empty
         return -1
     end if
2.   data = queue[front]
3.   front ← front + 1
4    return data
 end
```

Limitation of Queue

As you can see in the image below, after a bit of enqueuing and dequeuing, the size of the queue has been reduced.



Limitation of a queue

- The indexes 0 and 1 can only be used after the queue is reset when all the elements have been dequeued.
- After REAR reaches the last index, if we can store extra elements in the empty spaces (0 and 1), we can make use of the empty spaces. This is implemented by a modified queue called the circular queue.

Complexity Analysis

The complexity of enqueue and dequeue operations in a queue using an array is O(1).

Applications of Queue Data Structure

- CPU scheduling, Disk Scheduling
- When data is transferred asynchronously between two processes.The queue is used for synchronization. eg: IO Buffers, pipes, file IO, etc
- Handling of interrupts in real-time systems.
- Call Center phone systems use Queues to hold people calling them in an order

CPU Scheduling in Operating Systems

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

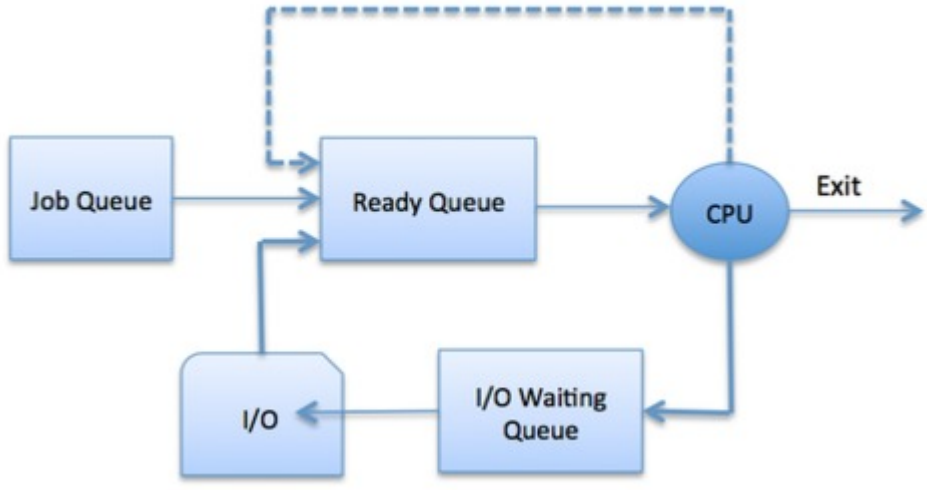
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCB (Process Control Block)s in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which

can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

- **Planning in advance, so need the information about process in the beginning only**
- It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.
- The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

- On some systems, the long-term scheduler may not be available or minimal**. Time-sharing operating systems have no long term scheduler.** When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler (Using Queue)

- It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.
- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler (Not to be focused)

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.

5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

CPU Scheduling (Process Scheduling)

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

- **First-Come, First-Served (FCFS) Scheduling**
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**.

Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time,

Preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state priority based or time sharing].

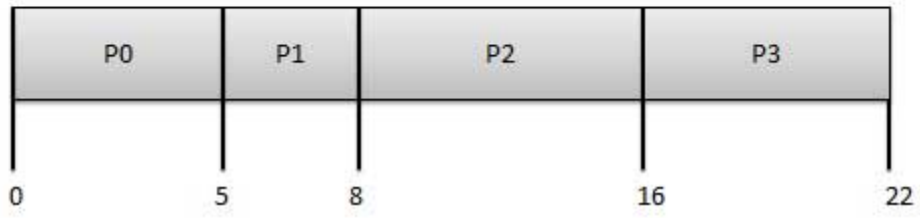
Below are different time with respect to a process.

- ***Arrival Time:*** *Time at which the process arrives in the ready queue.*
- ***Completion Time:*** *Time at which process completes its execution.*
- ***Burst Time:*** *Time required by a process for CPU execution.*
- ***Turn Around Time:*** *Time Difference between completion time and arrival time.*
    - *Turn Around Time = Completion Time – Arrival Time*
- ***Waiting Time(W.T):*** *Time Difference between turn around time and burst time.*
    - *Waiting Time = Turn Around Time – Burst Time*

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75