College of computer science & mathematics

Dep. Of Computer Science

DATA STRUCTURE

هياكل بيانات

Lecture 8 :

Queue

Prepared & Presented by

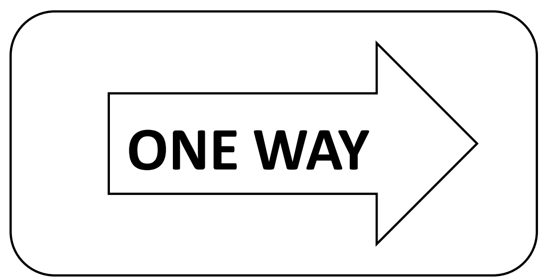
Mohammed B. Omar

2023 -2024

Definition of Queue

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

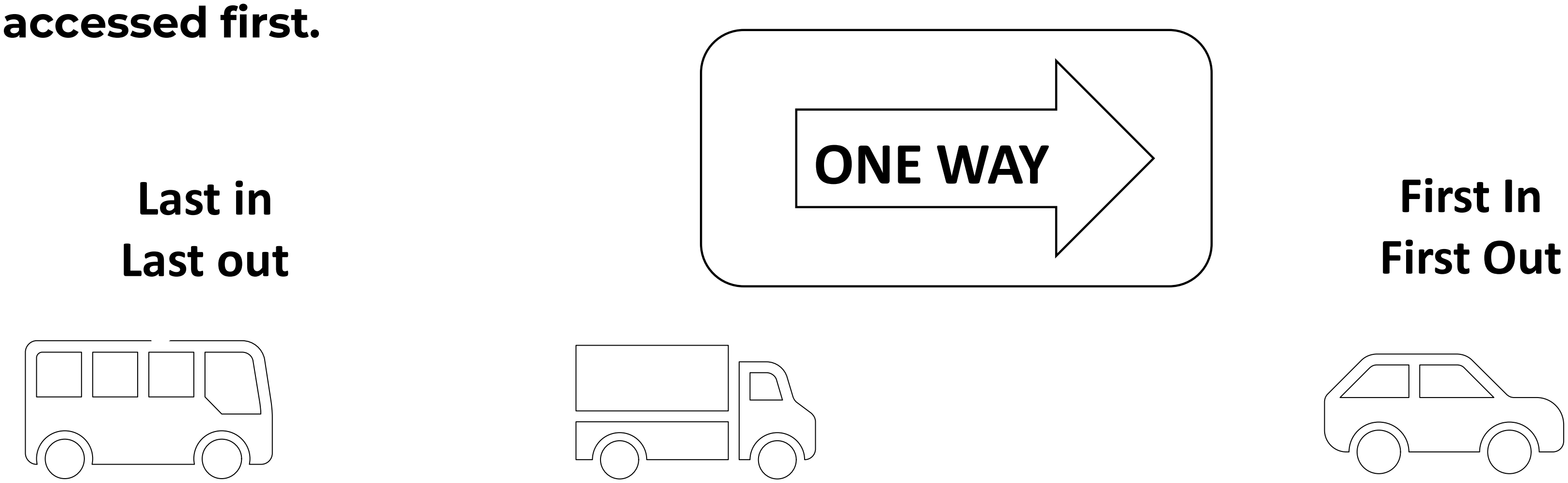
Last in
Last out



First In
First Out

Definition of Queue

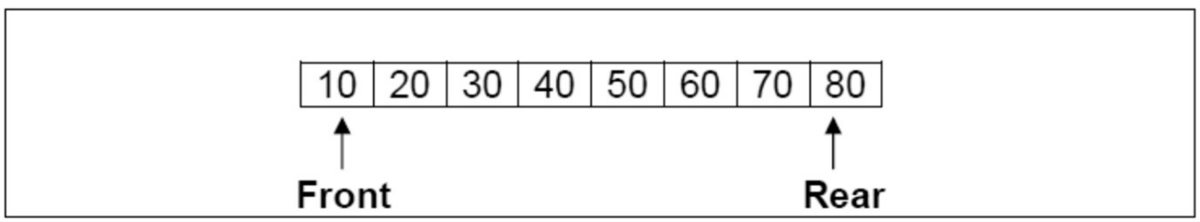
- **Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).**
- **Queue follows First-In-First-Out methodology, i.e., the data item stored first will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



Queue

Queue is a linear data structure.

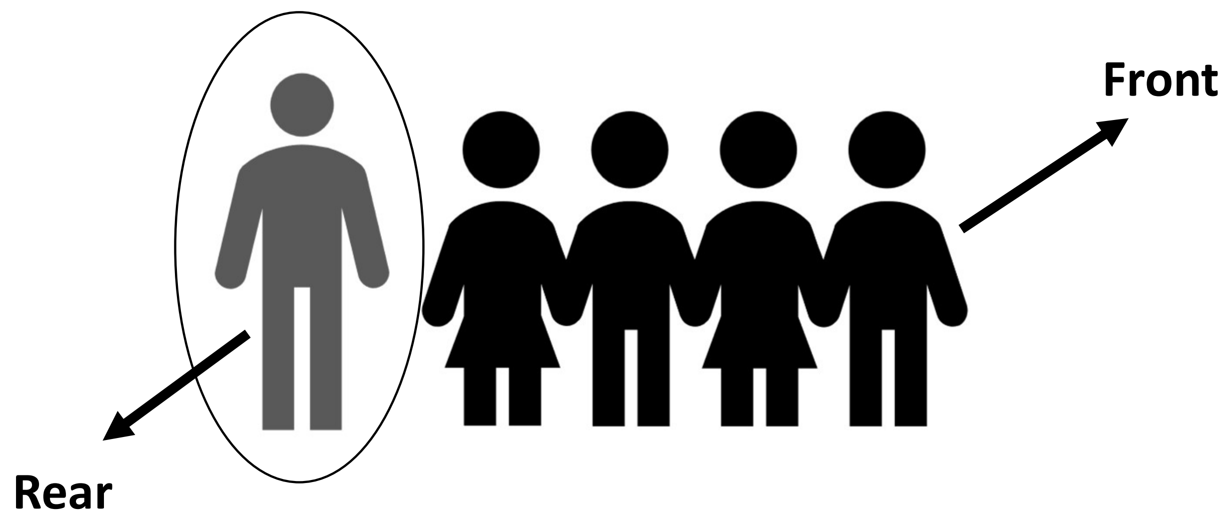
- **It is used for temporary storage of data values.**
- **A new element is added at one end called rear end.**
- **The existing elements deleted from the: front end.**



Queue. Similarly, 10 would be the first element to get removed and 80 would be the last element to get removed.

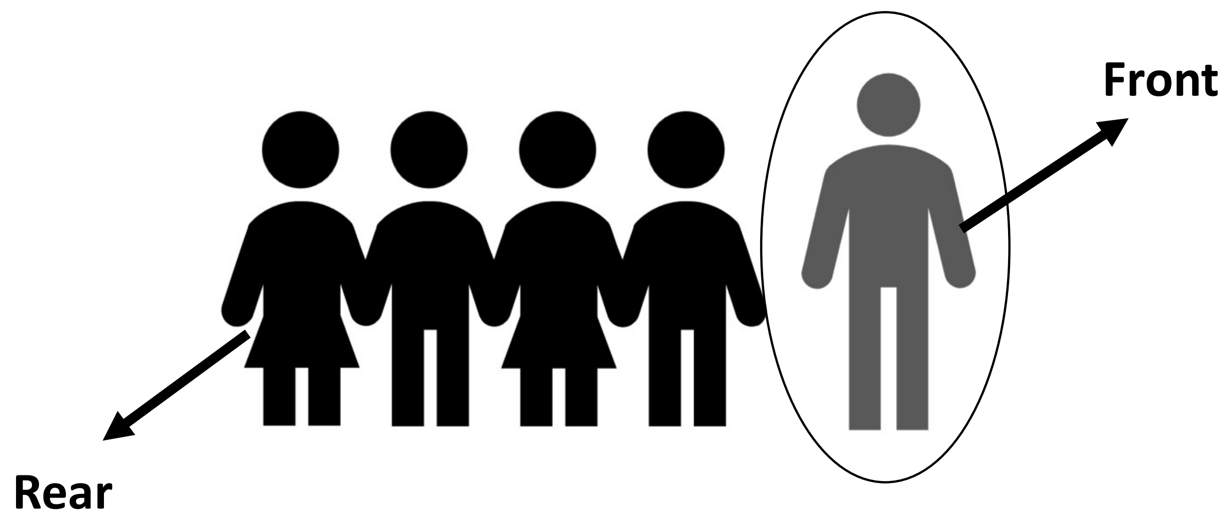
1.Insertion :

Placing an item in a queue is called "<u>insertion or enqueue</u>", which is done at the end of the queue called "<u>rear</u>".



2.Deletion :

Removing an item from a queue is called "<u>deletion or dequeue</u>" , which is done at the other end of the queue called "<u>front</u>".



Basic operations in Queue

Two basic operation:

- **enqueue() – add (store) an item to the queue.**
- **dequeue() – remove (access) an item from the queue.**
- **Functions are required to make the above-mentioned queue operation efficient.**

These are –

- **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.**

Basic operations in Queue

Enqueue

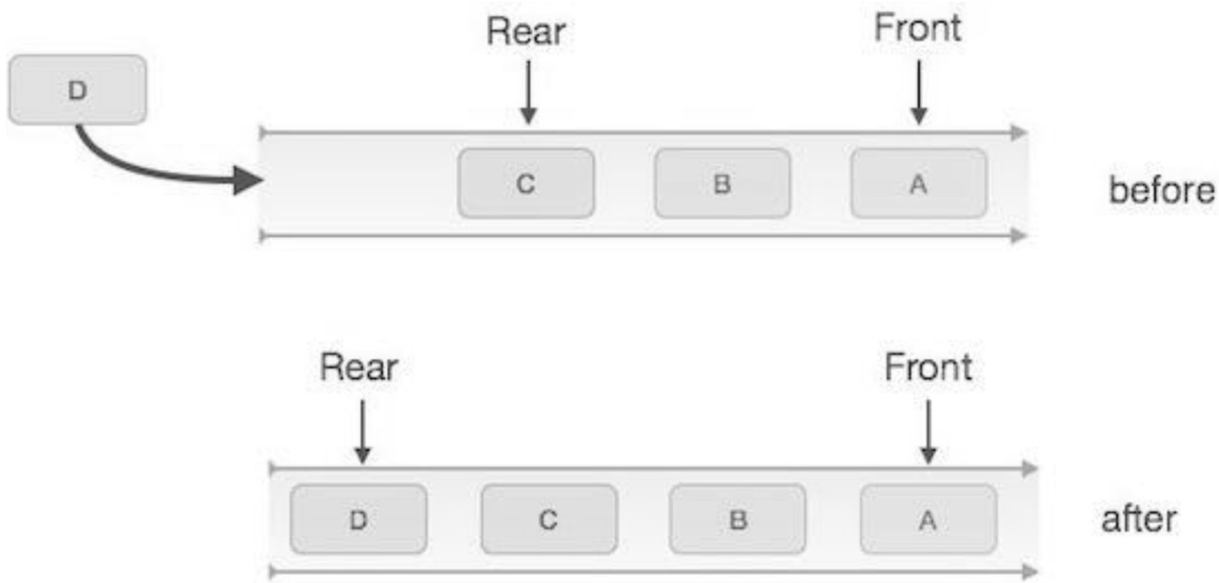
- Queues maintain two data pointers, front and rear.

Therefore, its operations are comparatively difficult to implement than that of stacks.

- **The following steps should be taken to enqueue (insert) data into a queue –**

- **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.**

Basic operations in Queue



Queue Enqueue

Basic operations in Queue

Enqueue

- Algorithm:

If Queue is Full

Then Overflow ← True

Else

Overflow ← False

Rear ← Rear + 1

Queue [Rear] ← New element

Basic operations in Queue

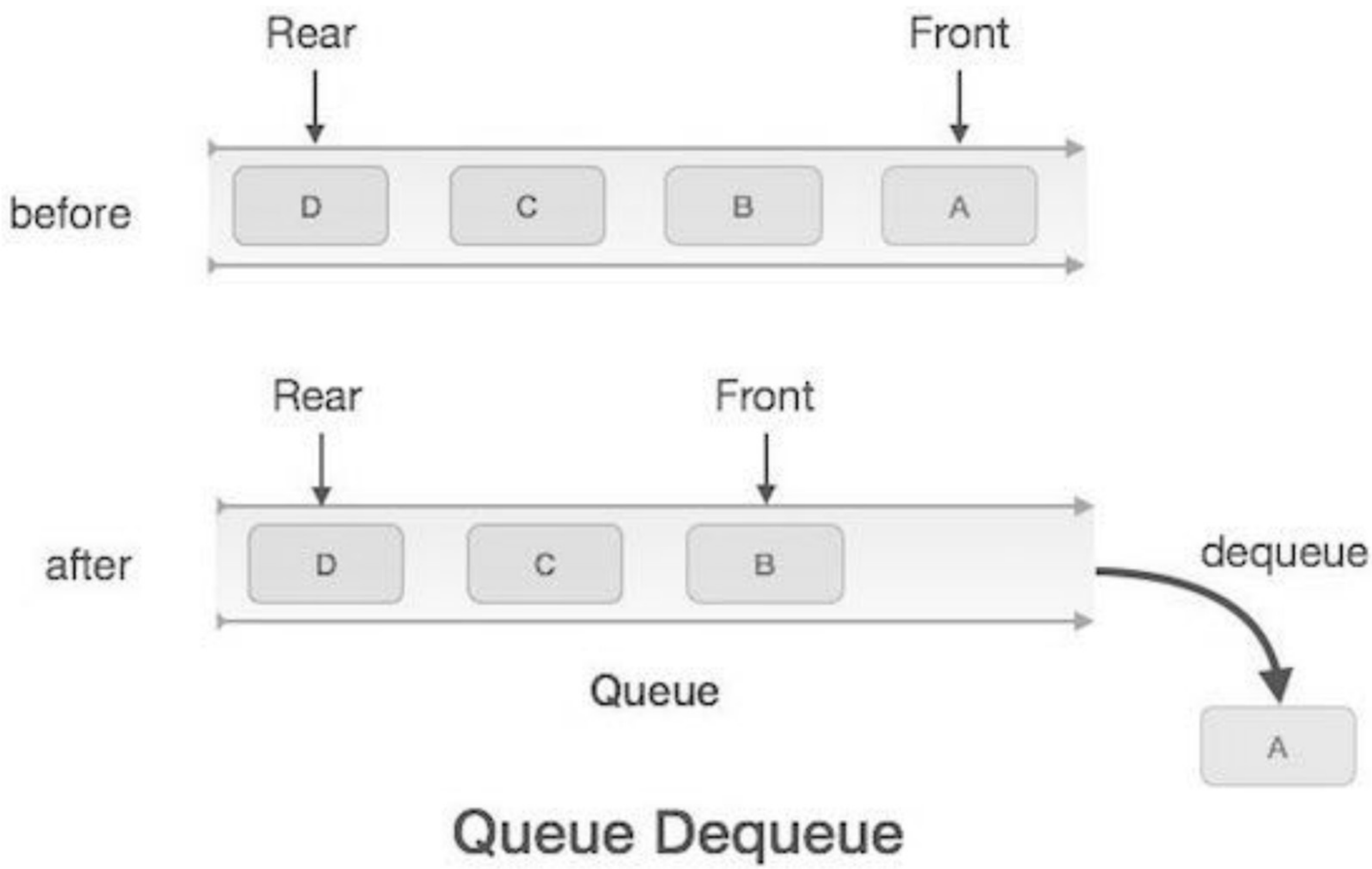
Dequeue

Accessing data from the queue is a process of two tasks access the data where front is pointing and remove the data after access.

- The following steps are taken to perform 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.**

Basic operations in Queue



Basic operations in Queue

Dequeue

- Algorithm:

If Queue is Empty

Then Underflow ← True

Else

Underflow ← False

Element ← Queue[front]

Front ← Front + 1

Basic operations in Queue

peek()

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

The algorithm of peek() function is as follows –

- **Algorithm**

begin procedure peek

return queue[front]

end procedure

- **Implementation of peek() function :**

Example

```
int peek()
{
return queue[front];
}
```

Basic operations in Queue

isfull()

- **In Queue have to check the rear pointer to reach at MAXSIZE to determine that the queue is full.**
- **Algorithm:**

```
If Rear = (size -1)
Then   FullQueue ← True
Else   FullQueue ← False
```

Basic operations in Queue

isempty()

If the value of front is less than 0, it tells that the queue is not yet initialized, hence empty.

- **Algorithm:**

If Front = -1

Then EmptyQueue ← True

Else EmptyQueue ← False

Basic operations in Queue

It is clear from the above figures that whenever we **insert** an element in the queue, the **value of Rear** is incremented by one i.e.
Rear = Rear + 1
Also, during the insertion of the **first element** in the queue we always incremented the Front by one i.e.
Front = Front + 1
Afterwards the Front will not be changed during the entire operation.

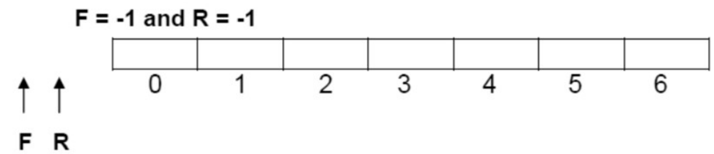


Fig. 2(a) Empty Queue

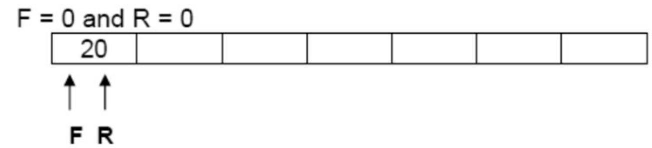


Fig. 2(b) One Element Queue

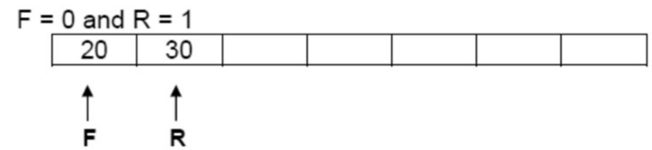


Fig. 2(c) Two Element Queue

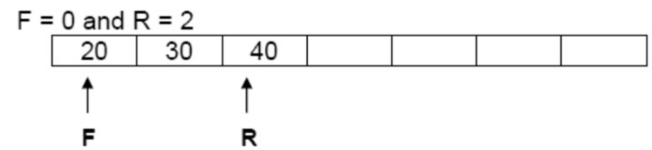


Fig. 2(d) Three Element Queue

Basic operations in Queue

The following figures show queue graphically during deletion operation:

Fig. 2(e) One Element (20) Deleted from Front

that whenever an element is **<u>removed from the queue</u>**, the value of Front is incremented by one i.e.,

<u>Front = Front + 1</u>

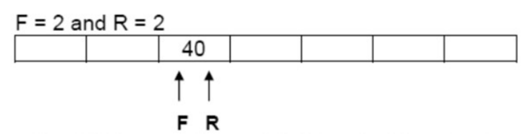


Fig. 2(f) Second Element (30) Deleted from Front

Now, if we insert any element in the queue, the queue will look like:

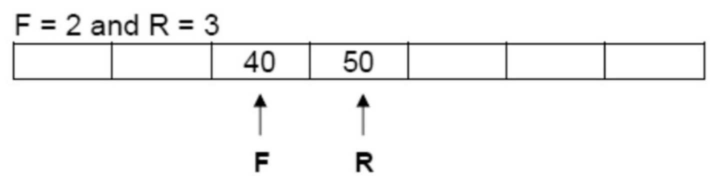


Fig. 2(g) Insertion after Deletion

Basic operations in Queue

(1) **Algorithm for Insertion in a Linear Queue**

Let QUEUE[MAXSIZE] is an array for implementing the Linear Queue & NUM is the element to be inserted in linear queue, FRONT represents the index number of the element at the beginning of the queue and REAR represents the index number of the element at the end of the Queue.

Step 1 :If REAR = (maxsize –1) : then
Write : "Queue Overflow" and return
[End of If structure]
Step 2 : Read NUM to be inserted in Linear Queue.
Step 3 : Set REAR = REAR + 1
Step 4 : Set QUEUE[REAR] = NUM
Step 5 : If FRONT = –1 : then Set FRONT=0.
[End of If structure]
Step 6 : Exit

Basic operations in Queue

Function for insert element in a linear queue (using arrays) in c++

```
void enqueue ( int NUM )
{
if (rear == maxsize -1)
cout<<"Queue is full \n";
else
{
if( front == -1)
front = 0;
rear++;
A[rear] = value;
}
}
```

Basic operations in Queue

```
void enqueue () {
int val;
if (rear == n - 1)
cout<<"Queue Overflow"<<endl;
else {
front = 0;
cout<<" insert value in the queue : "<<endl;
cin>>val;
rear++;
queue[rear] = val;
}
}
```

Basic operations in Queue

(2) Algorithm for Delete element from a Linear Queue

Let QUEUE[MAXSIZE] is an array for implementing the Linear Queue & NUM is the element to be deleted from linear queue, FRONT represents the index number of the element at the beginning of the queue and REAR represents the index number of the element at the end of the Queue.

Step 1 : If FRONT = -1 : then
Write : "Queue Underflow" and return
[End of If structure]
Step 2 : Set NUM := QUEUE[FRONT]
Step 3 : Write "Deleted item is : ", NUM
Step 4 : Set FRONT = FRONT + 1.
Step 5 : If FRONT>REAR : then
Set FRONT = REAR = -1.
[End of If structure]
Step 6 : Exit

Basic operations in Queue

Function for delete element from linear queue (using arrays) in c++

```
void Delete()
{
if (front == - 1)
{
cout<<"Queue Underflow ";
return ;
}
else
{
cout<<"Element deleted from queue is : "<<queue[front];
front++;;
}
}
```

Basic operations in Queue

Function of display Queue in C++

```
void Display_Queue ()
{
if (front == - 1 )
cout<<"Queue is empty";
else {
cout<<"Queue elements are : ";
for (int i = front; i <= rear;
i++)
cout<<queue[i]<<" ";
}
}
```

Basic operations in Queue

Function to check if queue is empty

```
bool isempty()
{
if(front == -1 && rear == -1)
return true;
else
return false;
}
```

Queue Data Structure

front = 50 **Rear = 70**

0 1 2 3 4 99

Array Queue

Max Size = 100

Elements of the queue **50,51,52,………….70**

Queue Data Structure

rear = 5

front = 99

0 1 2 3 4 99

Array Queue

Max Size = 100

Elements of the queue **99,0,1,2,3,4,5**

Queue Data Structure

rear = 2 front = 13

0 1 2 3 4 14

Array Queue

count= 5 **Max Size = 15**

Rear pointer point in the =2 13,14,0,1,2

because the account are have five elements, and the front pointer point in 13 that is means are being counted from 13,14,0,1,2 for just five elements.

Queue Data Structure

front = 99

rear = 3

0 1 2 3 4 99

Array Queue

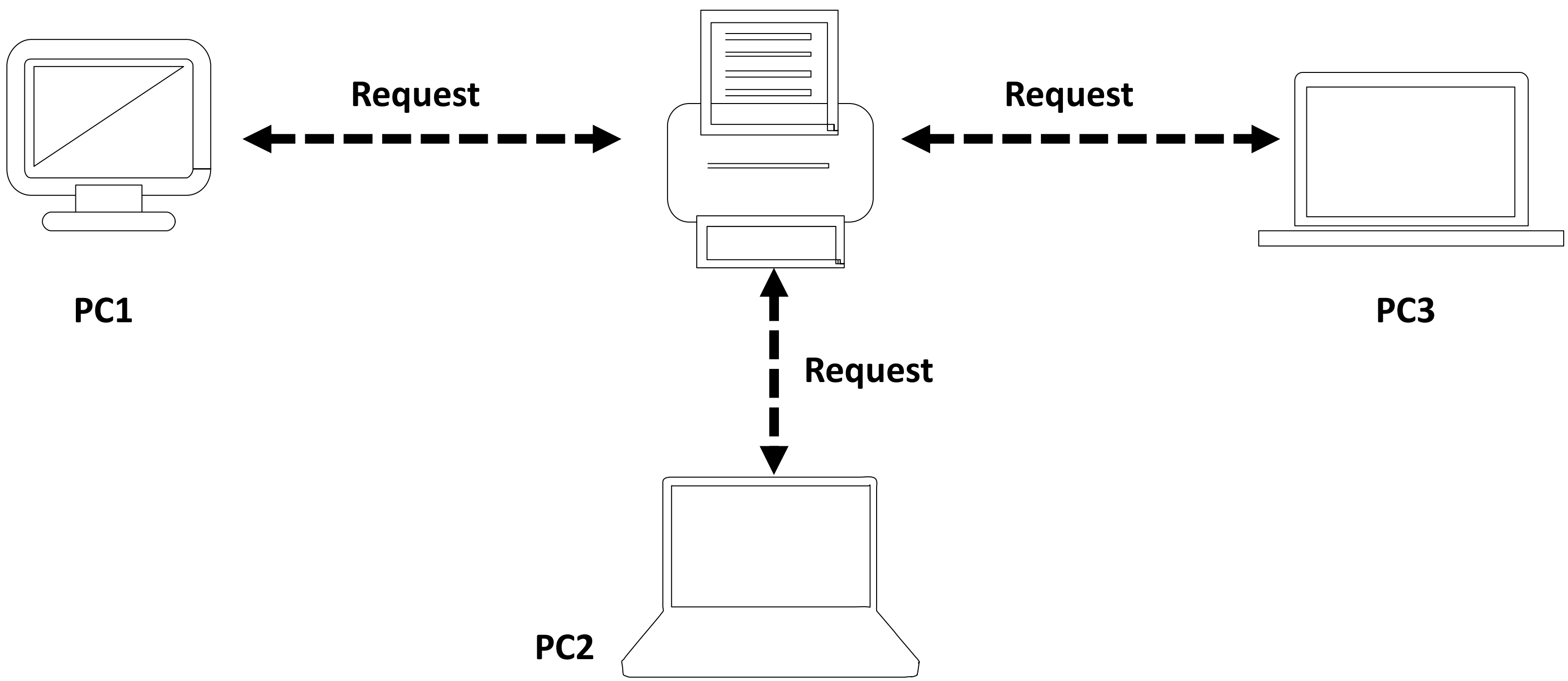
Count = ?

Max Size = 100

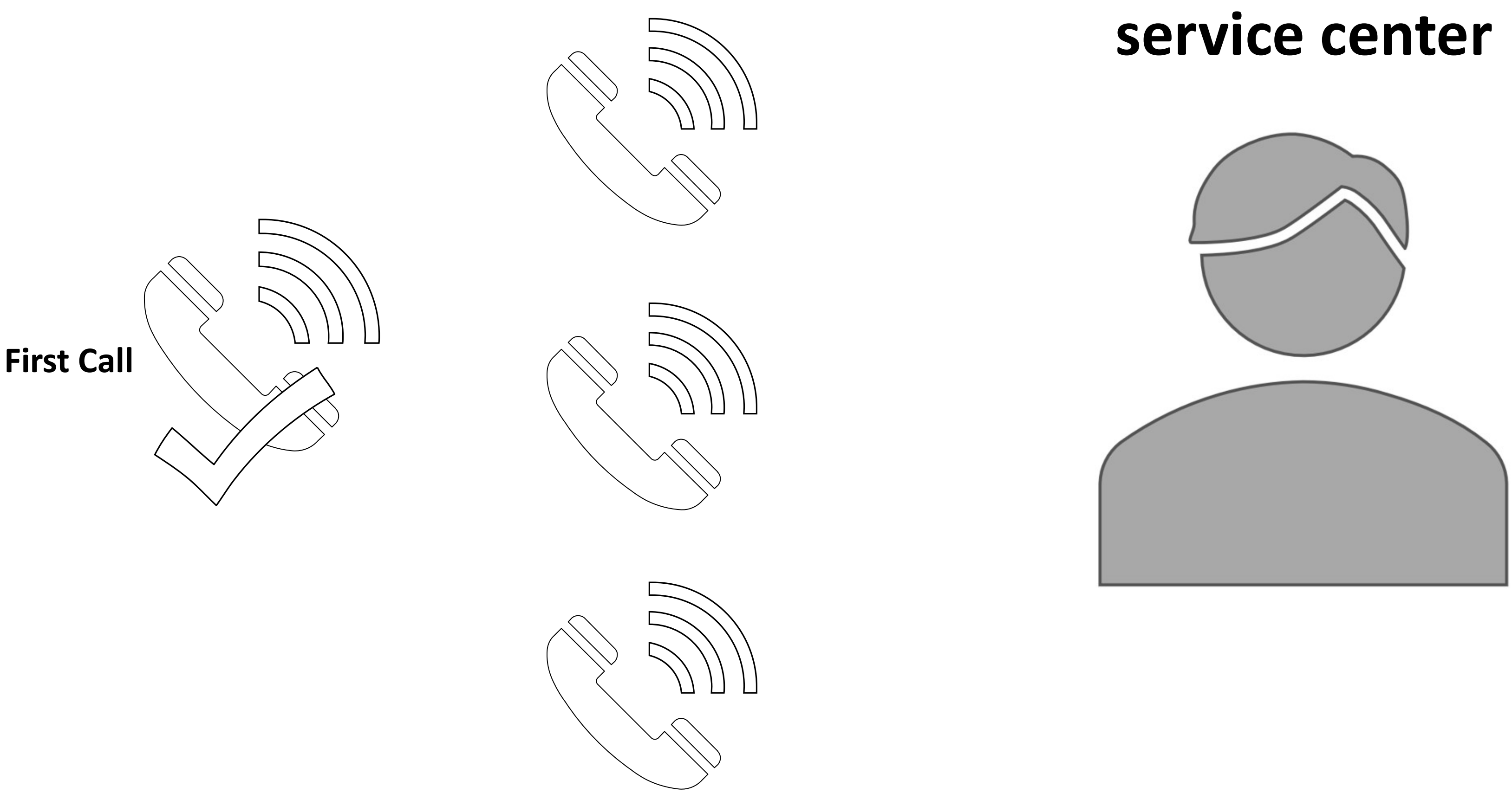
Elements of the queue **99,0,1,2,3**

Count = 5

Queue Data Structure



Queue Data Structure



Thank You

&

Good luck