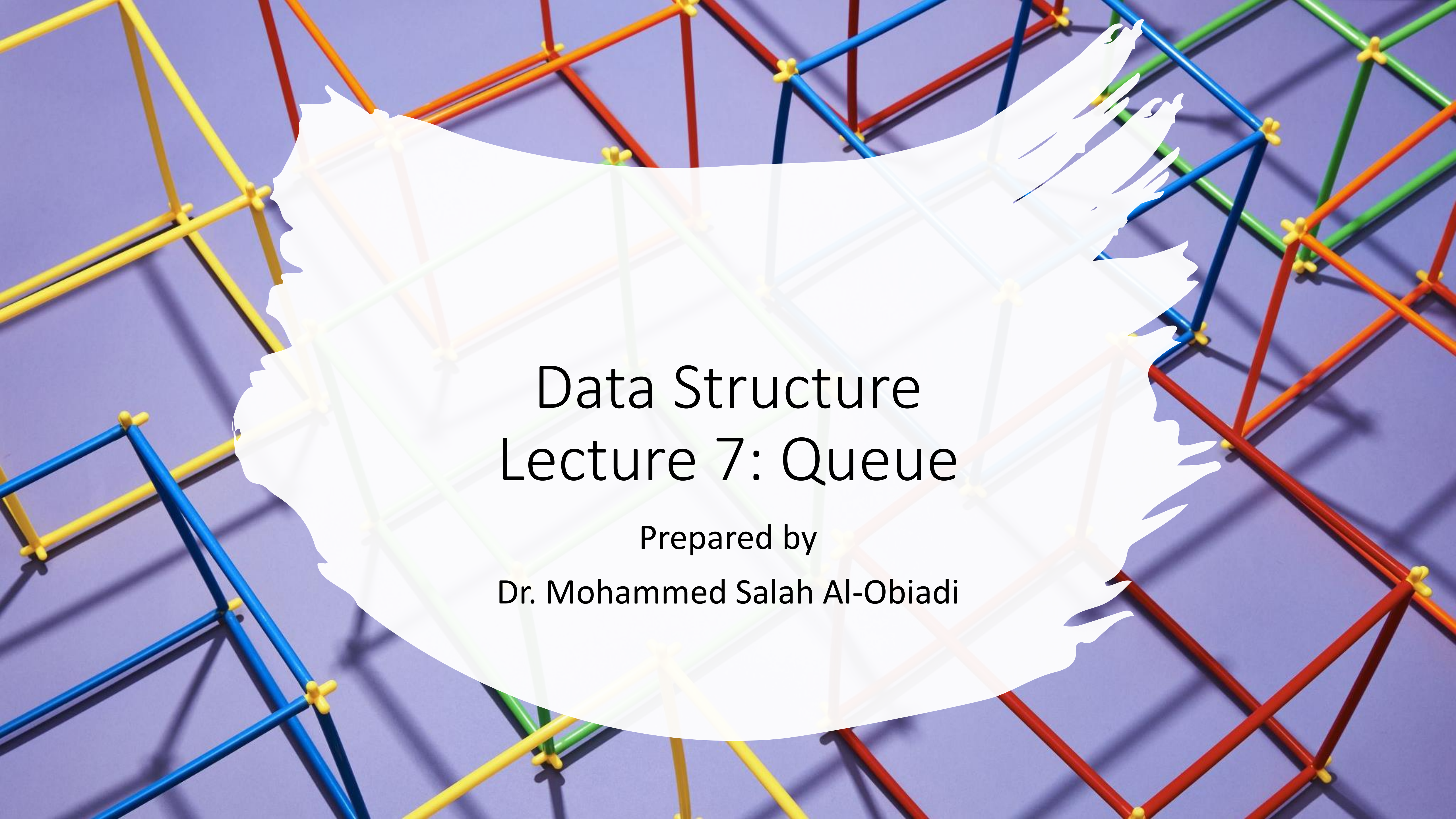


Data Structure
Lecture 7: Queue

Prepared by

Dr. Mohammed Salah Al-Obiadi

Queue

In a stack, insertion and removal of the item was permitted only from one end.

Item inserted at last removed first from the stack.

How to ensure that the items are removed in the order they have inserted?? Solution is the Queue.

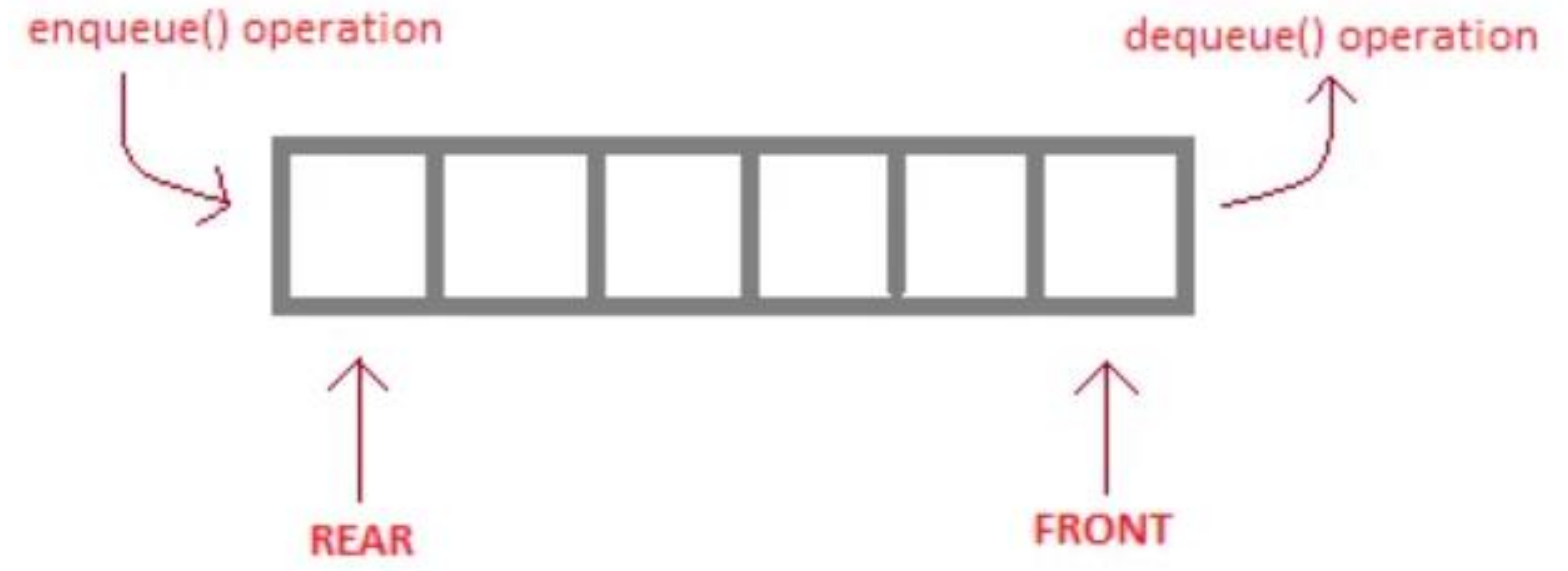
Queue is a data structure that can be considered as open from both the ends.

Queue follows the principle of **FIFO (First In First Out).**

Insertion is accomplished from one end known as **rear.**

Removal of the item is taking place on the other end known as **front**.

Queue



commonly implemented operations

1- Insert.

- During the INSERT operation we have to check the condition for **OVERFLOW**

2- Delete.

- During the DELETE operation we have to check the condition for **UNDERFLOW**.

Types of Queue

1- Linear Queue

2- Circular Queue

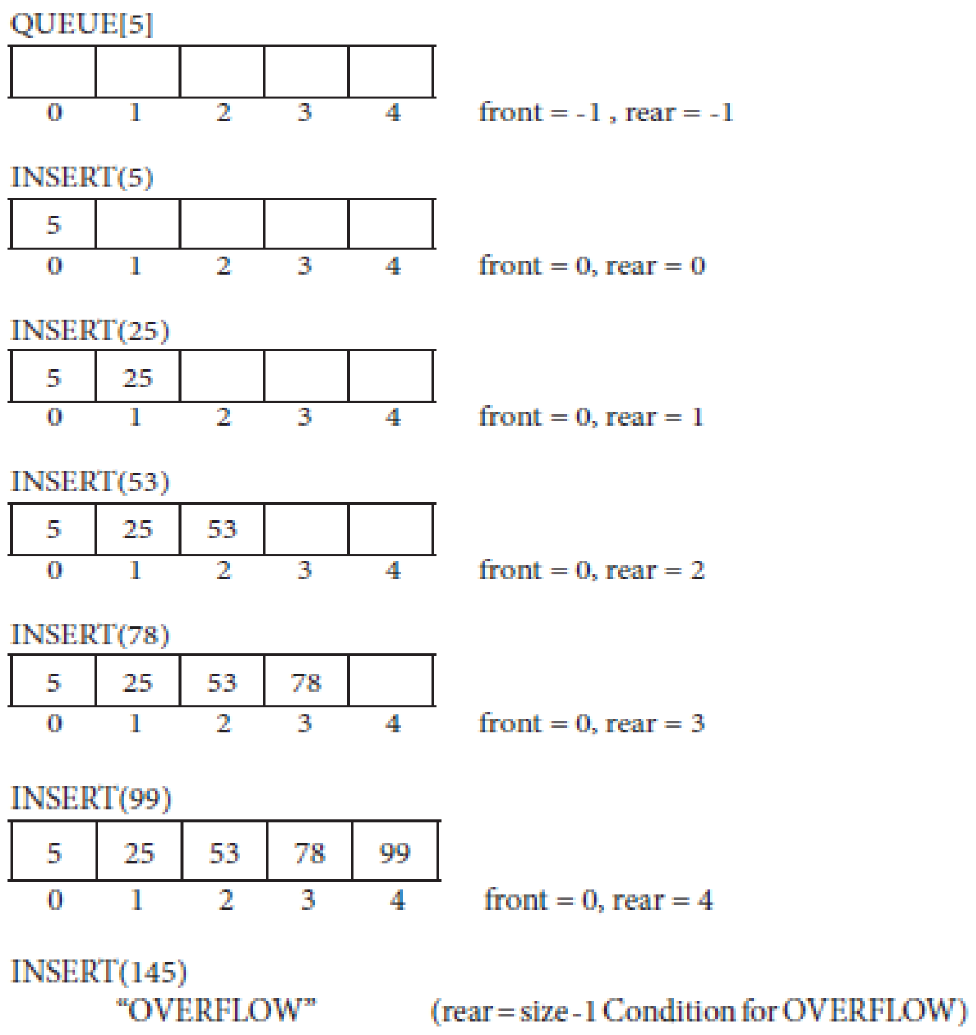
3- D - Queue (Double ended queue)

4- Priority Queue.

Linear Queue

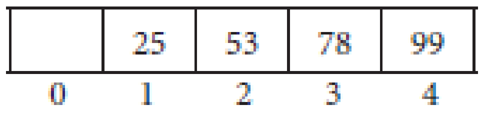
- A linear queue is a linear data structure that serves the request first, which has been arrived first.
- **Overflow**: insert an element with a filled QUEUE.
- Condition for OVERFLOW
  - Rear = size -1
- **UNDERFLOW:** delete an element from an empty QUEUE.
- Condition for UNDERFLOW
  - Front = -1 (for the QUEUE starts with 0)
- CONDITION FOR EMPTY QUEUE
  - Front = -1 and Rear = -1

Example: insert



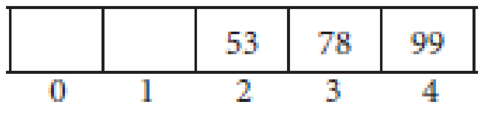
Example: Delete

DELETE



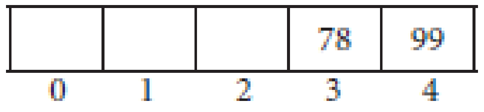
front = 1, rear = 4

DELETE



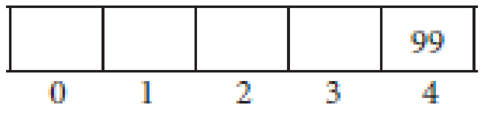
front = 2, rear = 4

DELETE



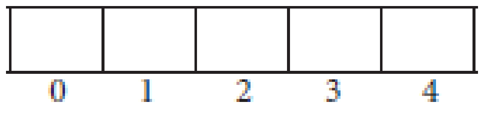
front = 3, rear = 4

DELETE



front = 4, rear = 4

DELETE



front = -1, rear = -1

DELETE

"UNDERFLOW" (front = -1 Condition for UNDERFLOW

Algorithm For Insert Operation

- **Insert**(queue[size], front, rear, no)
- **Step 1 :** if (rear = size – 1) then :
    - write : “overflow”
    - return
- **Step 2 :** if (rear = -1) then :
    - front := 0
    - rear :=0
  - else:
    - rear :=rear+1
- **Step 3:** queue[rear] :=no
- **Step 4:** return

Algorithm For Delete Operation

```
• Delet(queue[size], front, rear)
• Step 1 : if (front = -1) then :
        • write : "underflow"
        • return
• Step 2 : write: queue[front]
• Step 3 : if (front ==rear ) then :
            • front := -1
            • rear :=-1
        • else :
            • front := front +1
• Step 4: return
```

Algorithm For Traverse Operation

- **Traverse**(queue[size], front, rear)
- **Step 1 :** if (front = -1) then :
    - write : “ queue is empty ”
    - return
- **Step 2 :** set i:=0
- **Step 3 :** repeat for i = front to rear
    - write : queue[i]
- **Step 4:** return

Algorithm For Update Operation

- **Update**(queue[size], no, front, rear)
- **Step-1** : if (rear = - 1) then :
    - write : “stack is empty”
    - return
- **Step-2** : set i: =0
- **Step-3** : repeat for i = front to rear
    - if (no = queue[i]) then:
        - queue[i] = new no
        - return
    - if i=rear then:
        - write : “update not completed”
- **Step-4** : return

Applications of Queue

1- Operating systems schedule jobs (with equal priority) in the order of arrival (e.g., a print queue).

2- Simulation of real-world queues such as lines at a ticket counter or any other first come first-served scenarios.

3- Multiprogramming.

4- Asynchronous data transfer (file IO, pipes, sockets).

5- Waiting times of customers at call center.

6- Determining number of cashiers to have at a supermarket.