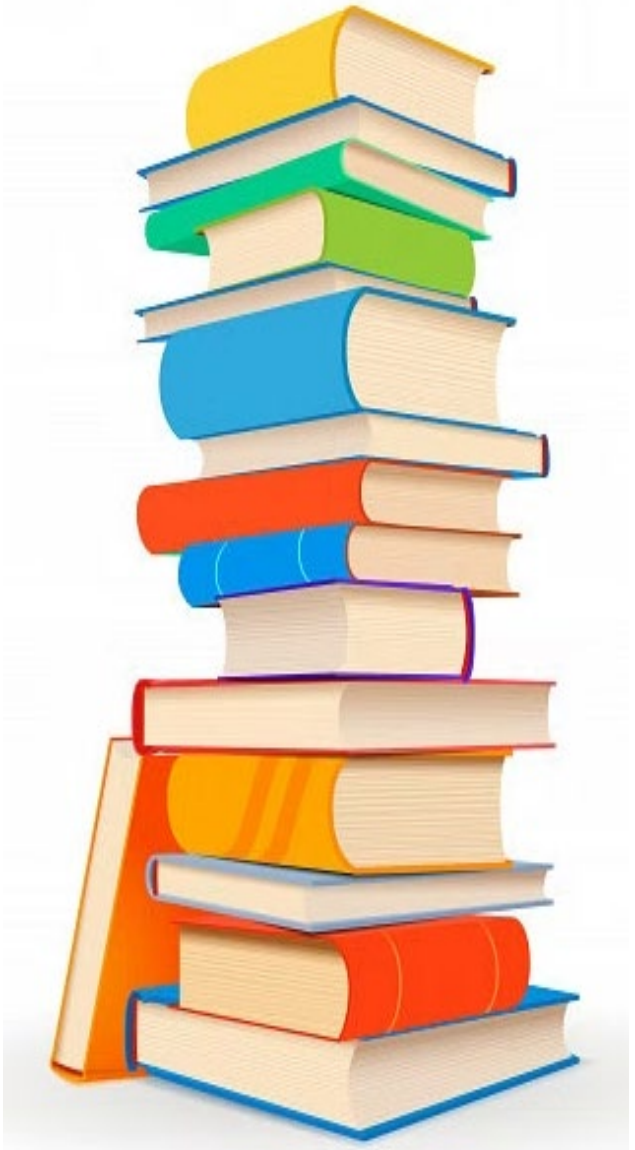
Unit 2

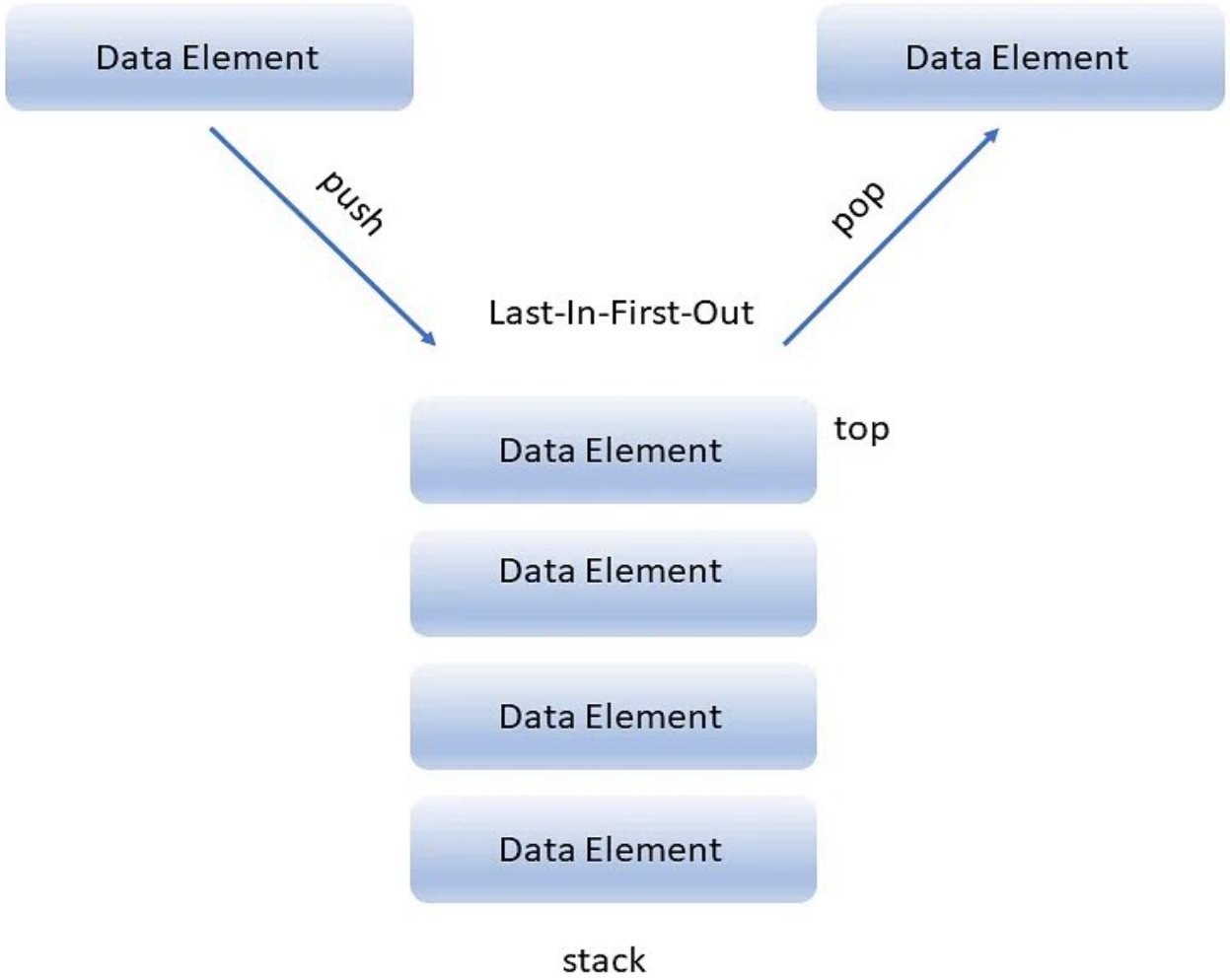
Data Structure and Algorithm
Stack and Queue

Stack

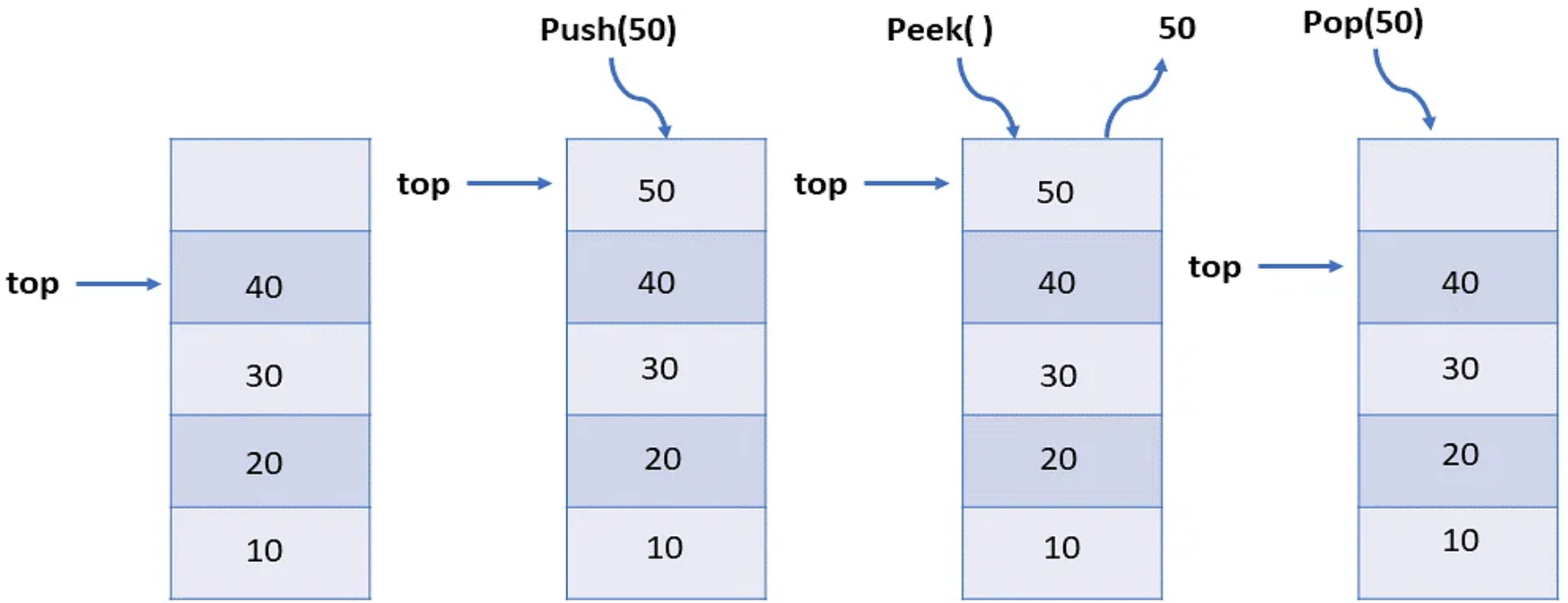
- The stack data structure is a linear data structure that accompanies a principle known as LIFO (Last In First Out) or FILO (First In Last Out).
- Real-life examples of a stack are a deck of cards, piles of books, piles of money, and many more.
- This example allows you to perform operations from one end only, like when you insert and remove new books from the top of the stack.
- It means insertion and deletion in the stack data structure can be done only from the top of the stack.
- You can access only the top of the stack at any given point in time.
- Inserting a new element in the stack is termed a push operation.
- Removing or deleting elements from the stack is termed pop operation.



Stack Representation in Data Structures



Working of Stack in Data Structures



Cont...

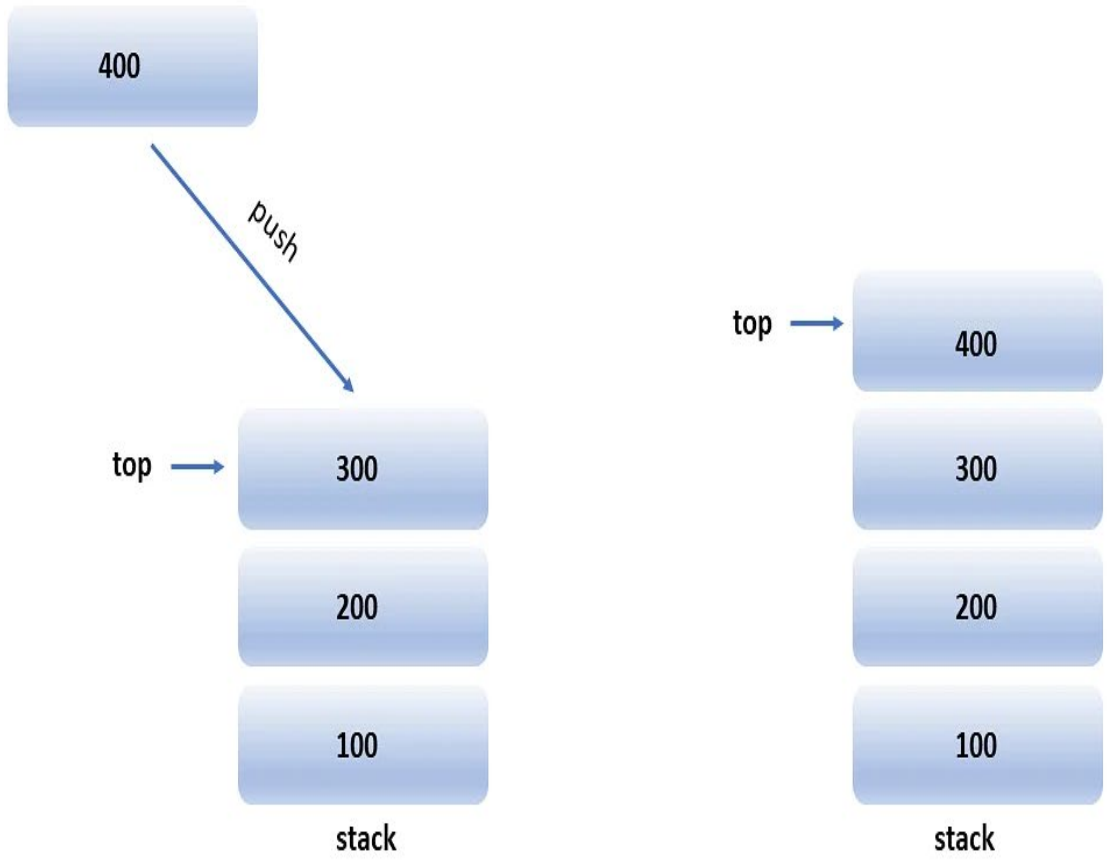
- Now, assume that you have a stack of books.
- You can only see the top, i.e., the top-most book, namely 40, which is kept top of the stack.
- If you want to insert a new book first, namely 50, you must update the top and then insert a new text.
- And if you want to access any other book other than the topmost book that is 40, you first remove the topmost book from the stack, and then the top will point to the next topmost book.
- After working on the representation of stacks in data structures, you will see some basic operations performed on the stacks in data structures.

Basic Operations on Stack in Data Structures

- There following are some operations that are implemented on the stack.

1. Push Operation

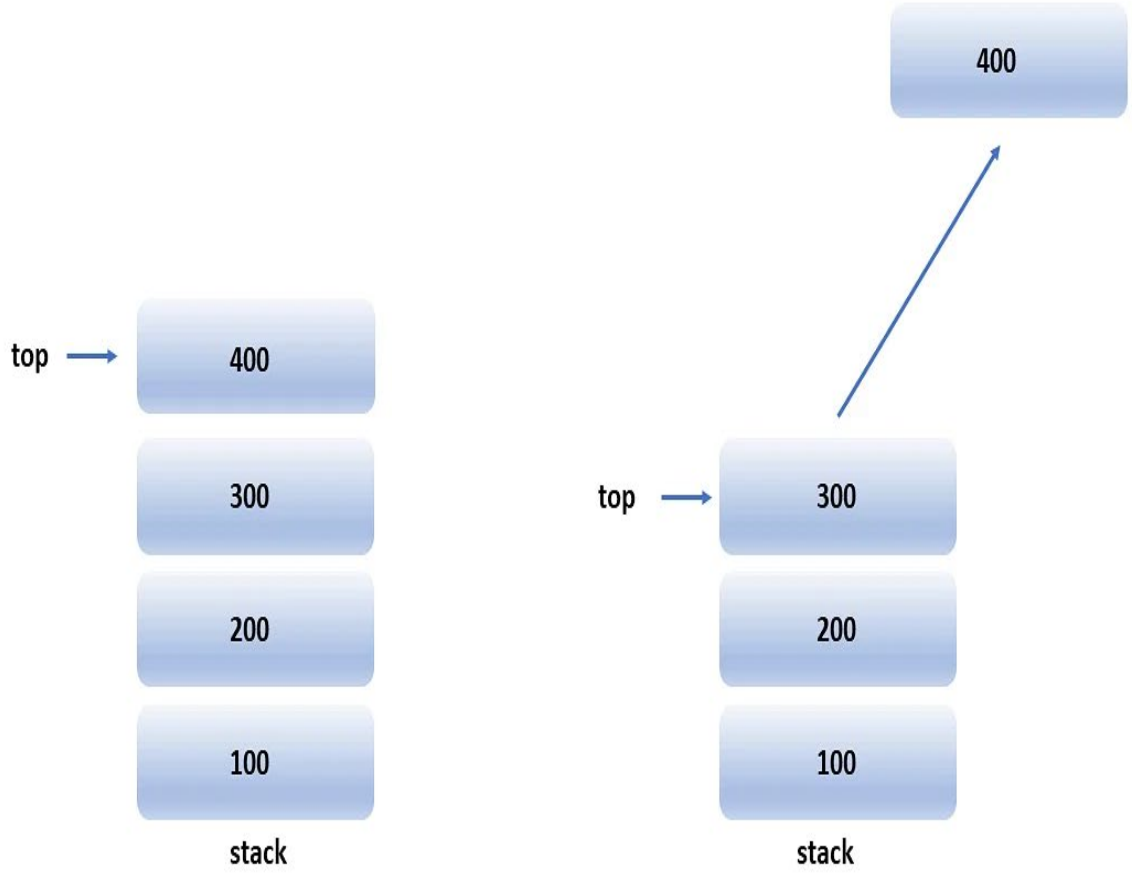
- Push operation involves inserting new elements in the stack. Since you have only one end to insert a unique element on top of the stack, it inserts the new element at the top of the stack.



Cont...

2. Pop Operation

- Pop operation refers to removing the element from the stack again since you have only one end to do all top of the stack. So removing an element from the top of the stack is termed pop operation.



Cont...

3. Peek Operation

- Peek operation refers to retrieving the topmost element in the stack without removing it from the collections of data elements.

4. isFull()

- isFull function is used to check whether or not a stack is empty.

5. isEmpty()

- isEmpty function is used to check whether or not a stack is empty.

6. isFull()

- The following is the algorithm of the isFull() function:

Algorithm for push:

- Adds an item to the stack. If the stack is full, then it is said to be an Overflow condition.

```
begin
if stack is full
 return
 endif
 else
 increment top
 stack[top] assign value
 end else
 end procedure
```

Algorithm for pop:

```
begin
 if stack is empty
   return
 endif
else
 store value of stack[top]
 decrement top
 return value
end else
end procedure
```

Algorithm for isFull():

```
begin
 If
 {
 if(top == maxsize)
 return true;
else
 return false;
}
```

Algorithm for isEmpty():

```
begin
stack isEmpty()
{
 if(top = = -1)
 return true;
else
 return false;

}
```

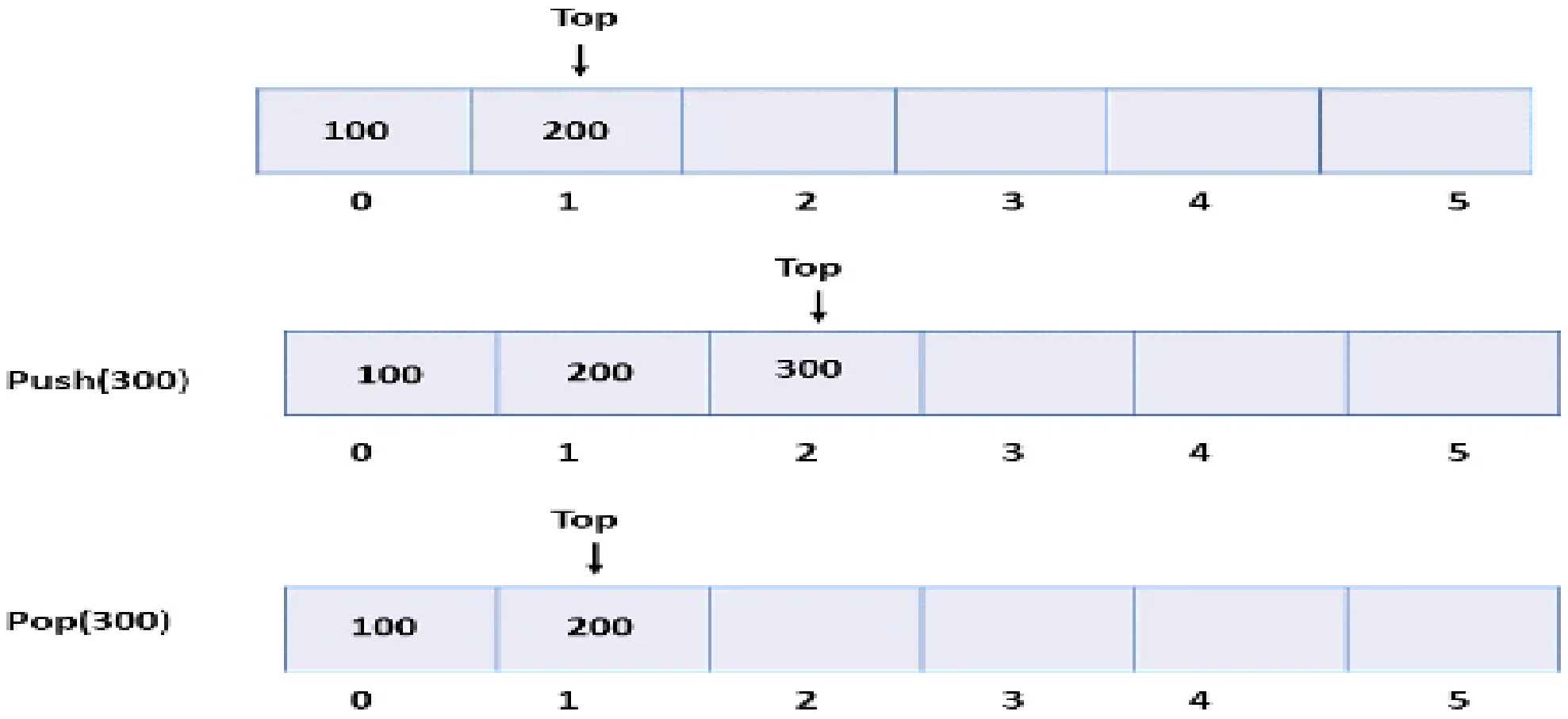
Implementation of Stack in Data Structures

- You can perform the implementation of stacks in data structures using two data structures that are an array and a linked list.

1. Array
2. Linked List

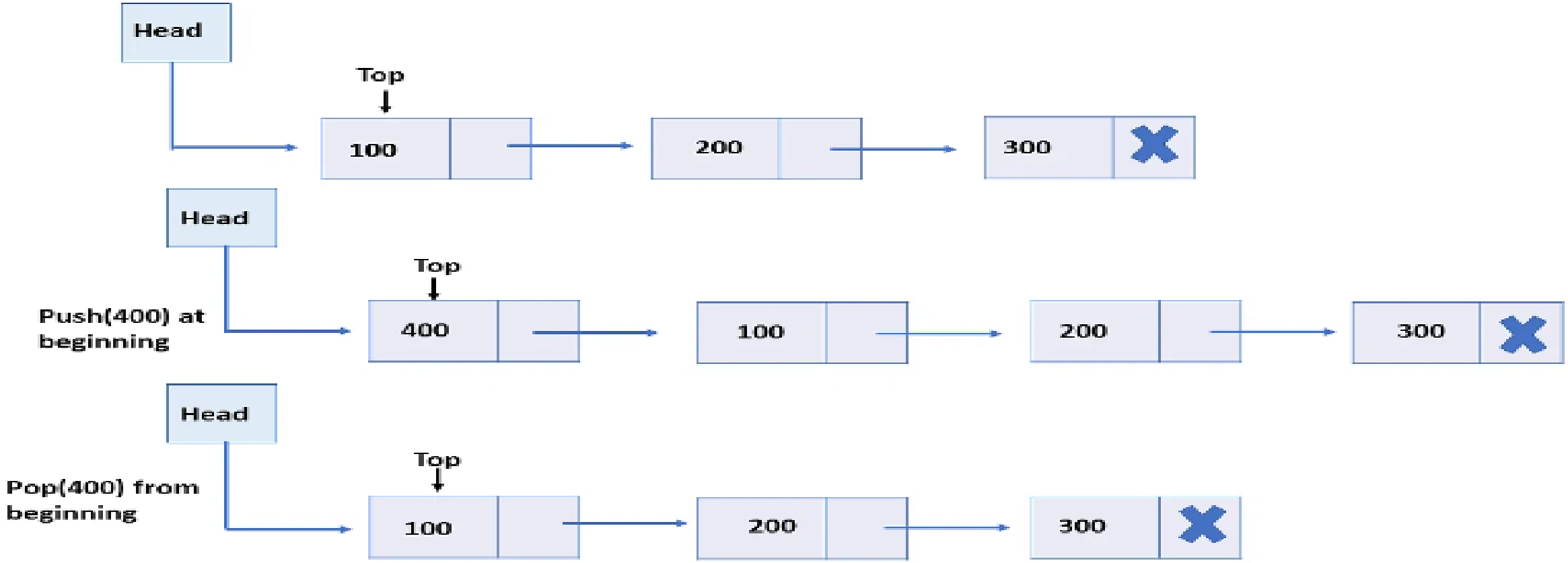
Implementation of Stack using Array

- Array: In array implementation, the stack is formed using an array. All the operations are performed using arrays.
- You will see how all operations can be implemented on the stack in data structures using an array data structure.



Implementation of Stack using Linked List

Linked-List: Every new element is inserted as a top element in the linked list implementation of stacks in data structures. That means every newly inserted element is pointed to the top. Whenever you want to remove an element from the stack, remove the node indicated by the top, by moving the top to its previous node in the list.



Program

```
switch(choice)
        {
case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
{
                display();
                break;
            }
case 4:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
            default:
            {
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
            }
        }
while(choice!=4);
    return 0;
}
```

Program for Push

```
        void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK is over flow");
   }
   else
   {
      printf(" Enter a value to be pushed:");
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
```

Program for POP

```
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack is under flow");
   }
   else
   {
      printf("\n\t The popped elements is %d",stack[top]);
      top--;
   }
}
```

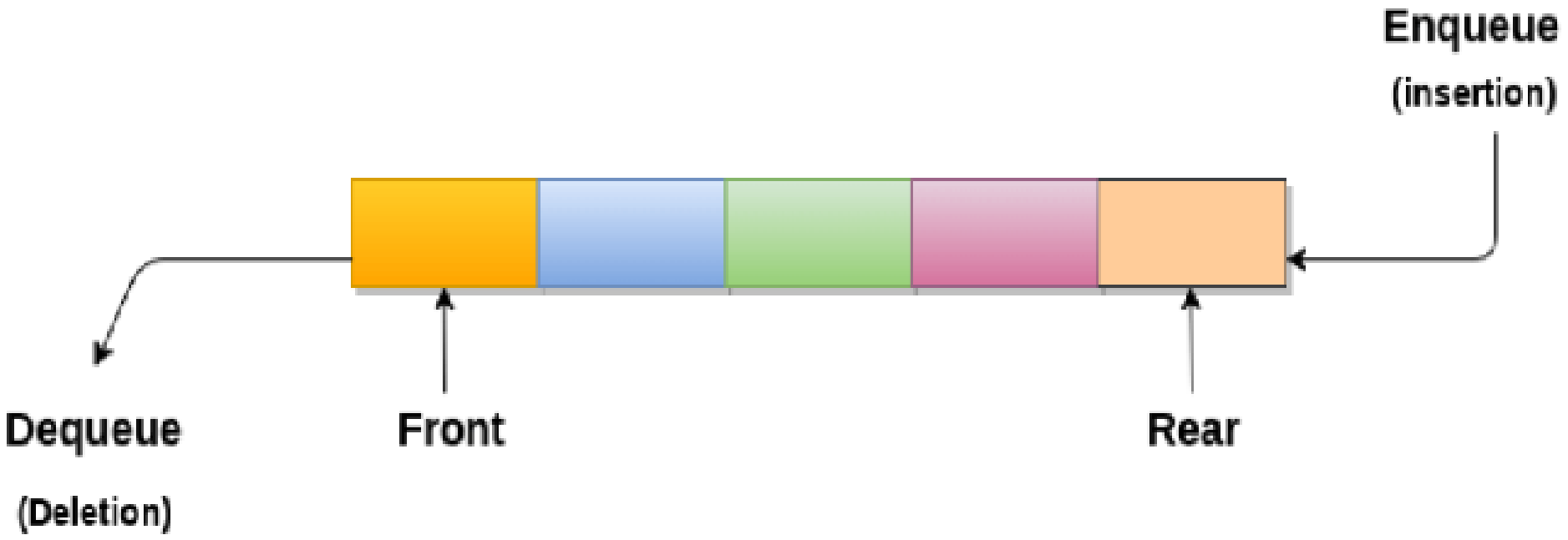
For Display

```
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK \n");
      for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
      printf("\n Press Next Choice");
   }
   else
   {
      printf("\n The STACK is empty");
   }
}
```

Queue

- Queue is the data structure that is similar to the queue in the real world.
- A Queue is a FIFO (First In First Out) data structure where the element that is added first will be deleted first.
- The basic queue operations are enqueue (insertion) and dequeue (deletion).
- Enqueue is done at the front of the queue and dequeue is done at the end of the queue.
- The elements in a queue are arranged sequentially and hence queues are said to be linear data structures.
- The real-world example of a queue is the ticket queue outside a cinema hall, where the person who enters first in the queue gets the ticket first, and the last person enters in the queue gets the ticket at last.
- Similar approach is followed in the queue in data structure.

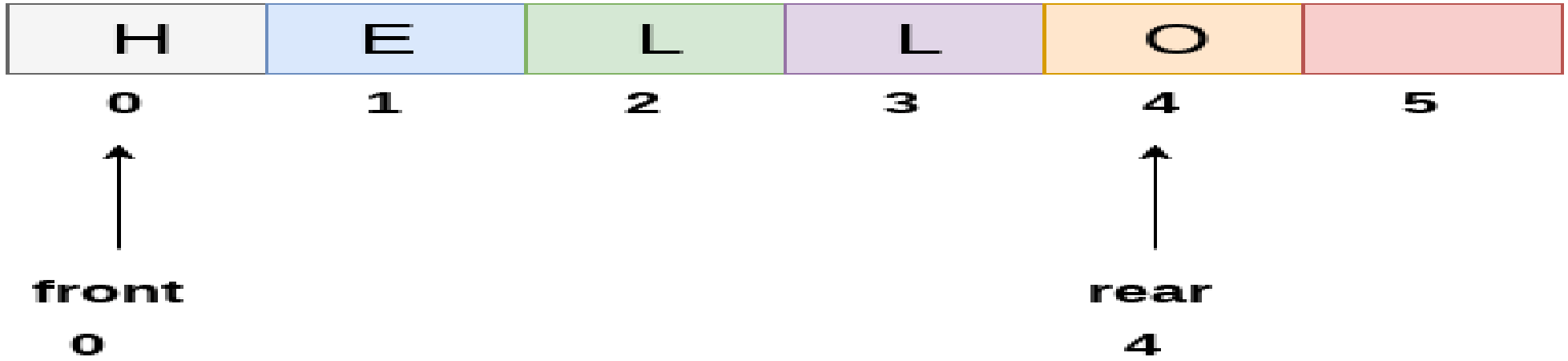
Cont...



Array representation of Queue

- We can easily represent queue by using linear arrays.
- There are two variables i.e. front and rear, that are implemented in the case of every queue.
- Front and rear variables point to the position from where insertions and deletions are performed in a queue.
- Initially, the value of front and queue is -1 which represents an empty queue.
- Array representation of a queue containing 5 elements along with the respective values of front and rear, is shown in the following figure.

Cont...



- The above figure shows the queue of characters forming the English word "HELLO".
- Since, No deletion is performed in the queue till now, therefore the value of front remains -1 .
- However, the value of rear increases by one every time an insertion is performed in the queue.

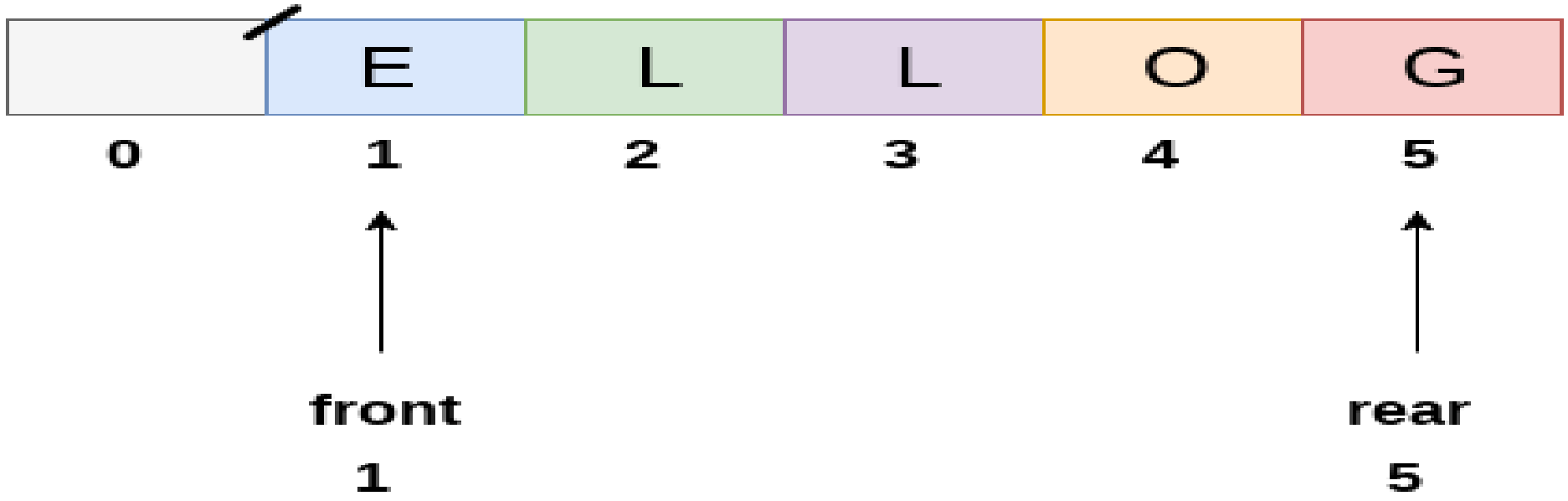
Cont...

- After inserting an element into the queue shown in the above figure, the queue will look something like following.
- The value of rear will become 5 while the value of front remains same.

Queue after inserting an element

Cont...

- After deleting an element, the value of front will increase from -1 to 0. however, the queue will look something like following.



Types of Queues

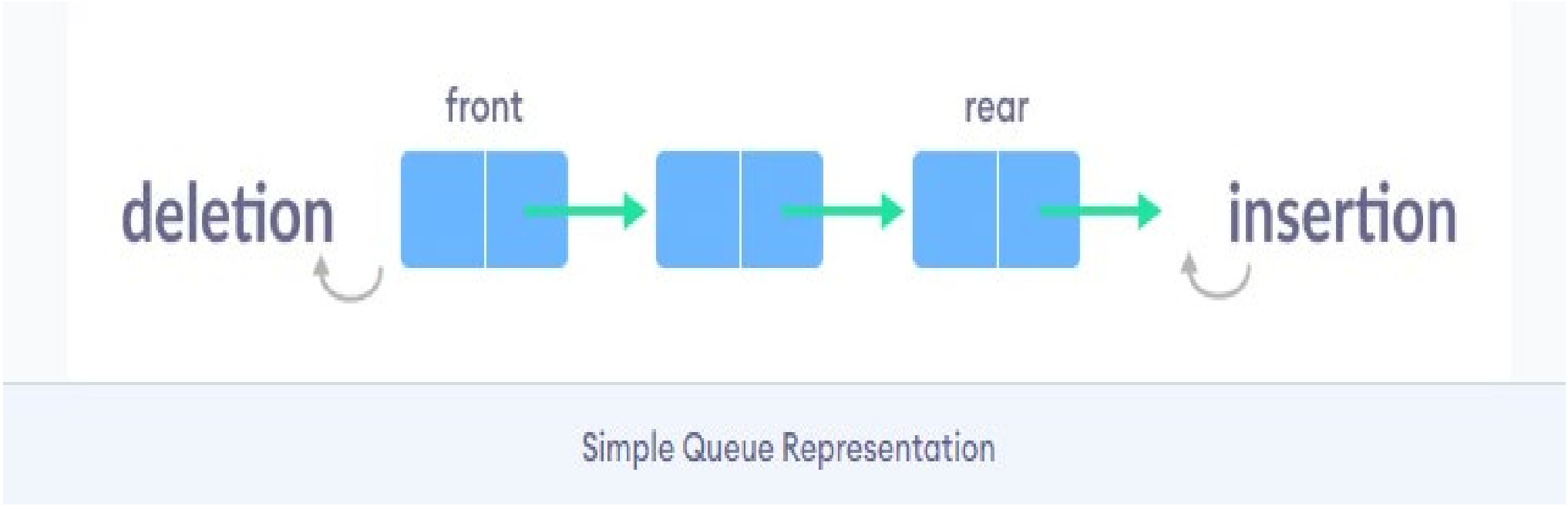
- A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

There are four different types of queues:

1. Simple Queue
2. Circular Queue
3. Priority Queue
4. Double Ended Queue

Simple Queue

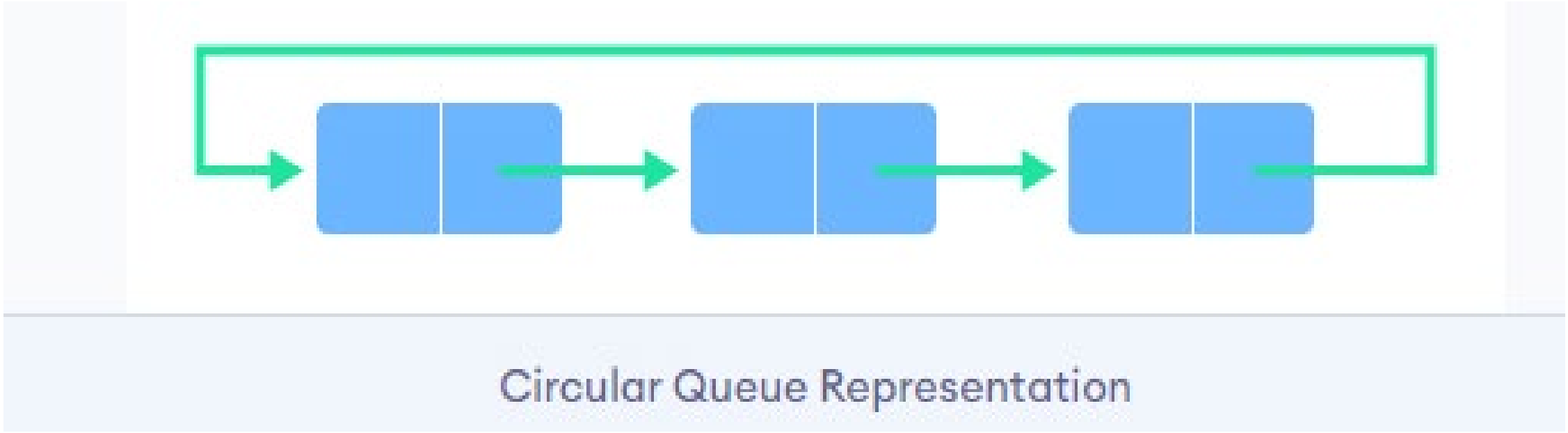
- In a simple queue, insertion takes place at the rear and removal occurs at the front.
- It strictly follows the FIFO (First in First out) rule.



Simple Queue Representation

Circular Queue

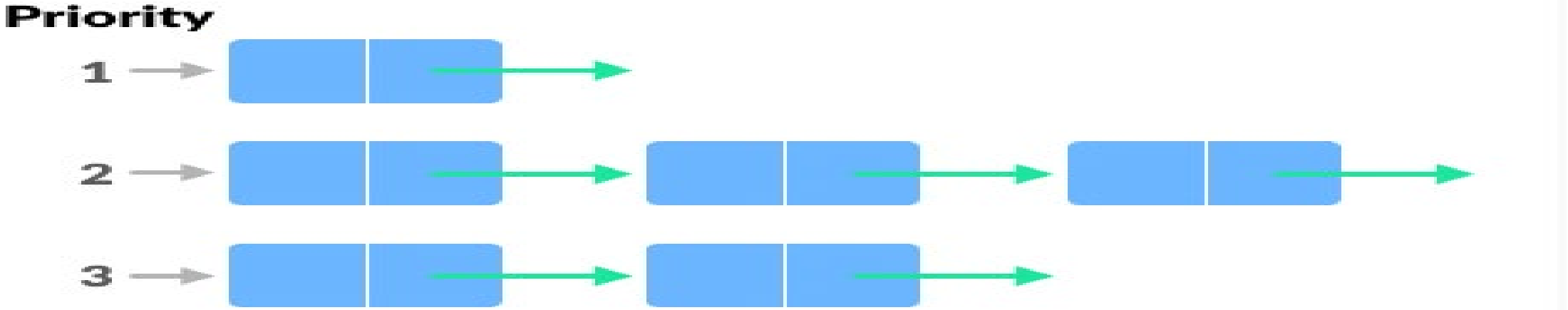
- In a circular queue, the last element points to the first element making a circular link.
- The main advantage of a circular queue over a simple queue is better memory utilization.
- If the last position is full and the first position is empty, we can insert an element in the first position.
- This action is not possible in a simple queue.
- Circular queue helps you to maintain round robin schedule.



Circular Queue Representation

Priority Queue

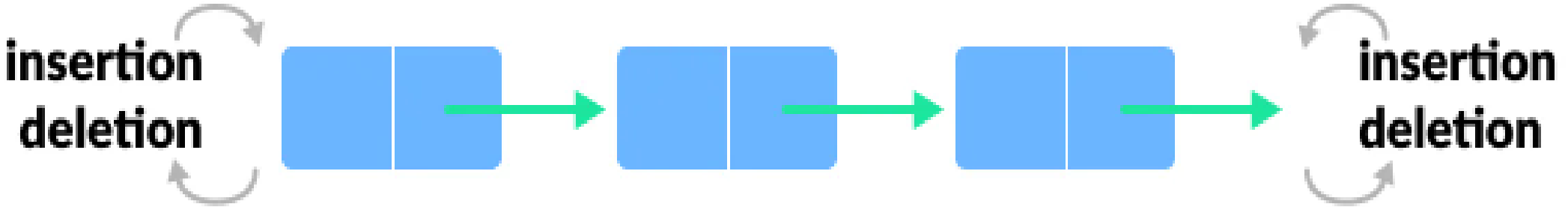
- A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority.
- If elements with the same priority occur, they are served according to their order in the queue.
- Insertion occurs based on the arrival of the values and removal occurs based on priority.



Priority Queue Representation

Deque (Double Ended Queue)

- In a double ended queue, insertion and removal of elements can be performed from either from the front or rear.
- Thus, it does not follow the FIFO (First In First Out) rule.



Deque Representation

Operation performed on queue:

- A queue is a linear data structure that implements the First-In-First-Out (FIFO) principle. Here are some common operations performed on queues:

1. Enqueue: Elements can be added to the back of the queue, adding a new element to the end of the queue.
2. Dequeue: The front element can be removed from the queue by performing a dequeue operation, effectively removing the first element that was added to the queue.
3. Peek: The front element can be inspected without removing it from the queue using a peek operation.
4. IsEmpty: A check can be made to determine if the queue is empty.
5. Size: The number of elements in the queue can be determined using a size operation.

Program for Queue

```
switch (choice)
      {
 case 1:
 insert();
  break;

  case 2:
  delete();
 break;

 case 3:
  display();
   break;

  case 4:
 exit(1);

  default:
 printf("Wrong choice \n");
 }
```

Code for Insertion

```
void insert()
{
 int add_item;
if (rear == MAX - 1)
 printf("Queue Overflow \n");
 else
{
 if (front == - 1)
  /*If queue is initially empty */
 front = 0;
printf("Inset the element in queue : ");
scanf("%d", &add_item);
rear = rear + 1;
queue_array[rear] = add_item;
 }
} /* End of insert() */
```

Code for Deletion

```
void delete()
{
 if (front == - 1 || front > rear)
 {
 printf("Queue Underflow \n");
 return ;
 }
  else
 {
printf("Element deleted from queue is : %d\n", queue_array[front]);
 front = front + 1;
 }
} /* End of delete() */
```

Code for Deletion

```
void display()
 {
  int i;
  if (front == - 1)
  printf("Queue is empty \n");
else
 {
printf("Queue is : \n");
  for (i = front; i <= rear; i++)
  printf("%d ", queue_array[i]);
  printf("\n");
  }
} /* End of display() */
```