SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23ITT101-PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

UNIT IV

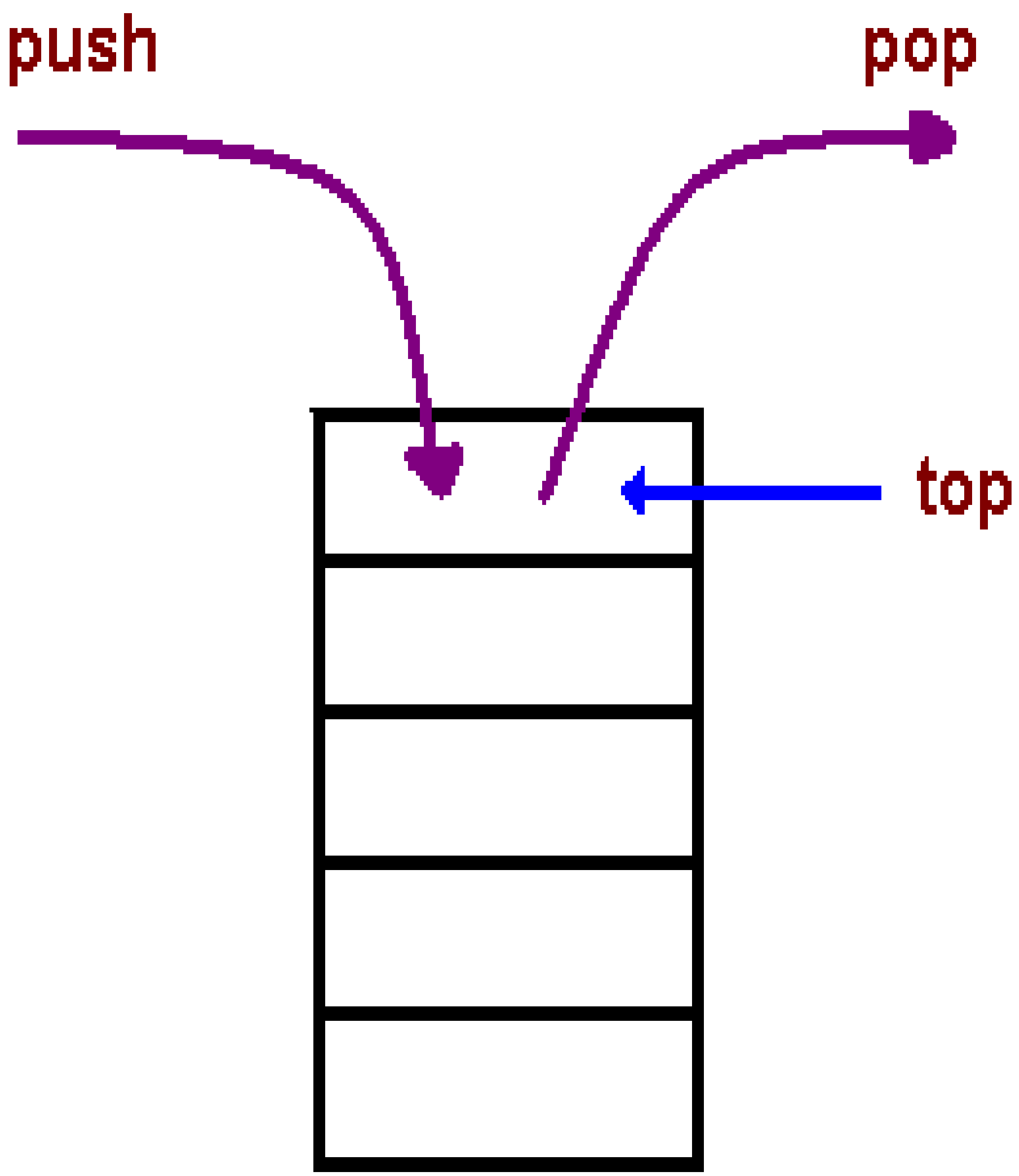
STACK AND QUEUE

STACK

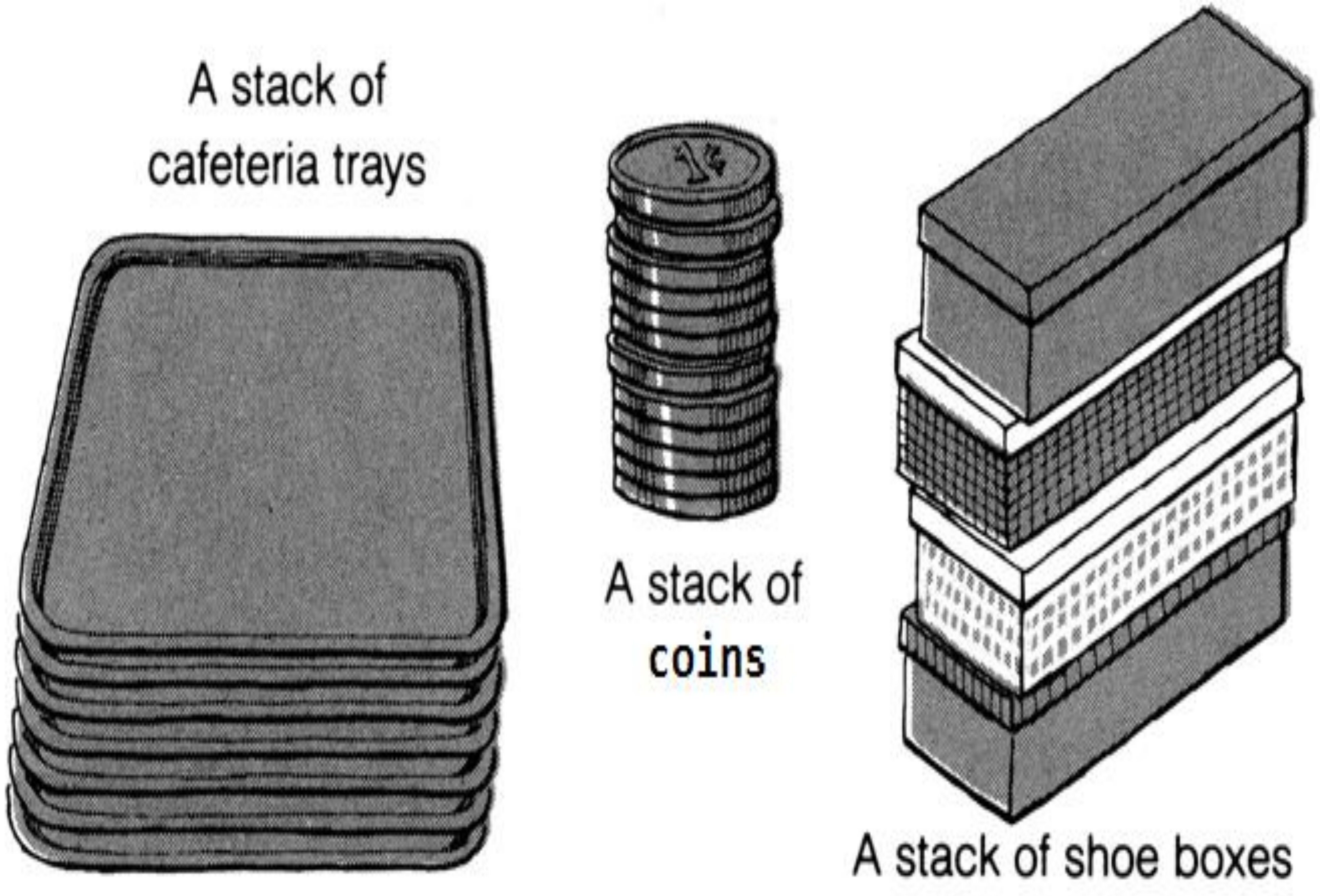
What is a stack?

- Stack is a **collection of similar data** items in which both push (insertion) and pop (deletion) operations are performed at one end called **Top**
- Both push and pop are allowed at only one end of Stack called Top
- **LIFO Principle**: Last In, First Out

Operation of the stack



Real time example of the stack



Basic Operations of Stack

Primary Operations

push() – Pushing (storing) an element on the stack

pop() – Removing (accessing) an element from the stack

Secondary Operations

peek() – get the top data element of the stack, without removing it

isFull() – check if stack is full

isEmpty() – check if stack is empty

Push Operation

- The process of **adding a new data element** onto stack is known as a Push Operation
- Push operation involves a series of steps

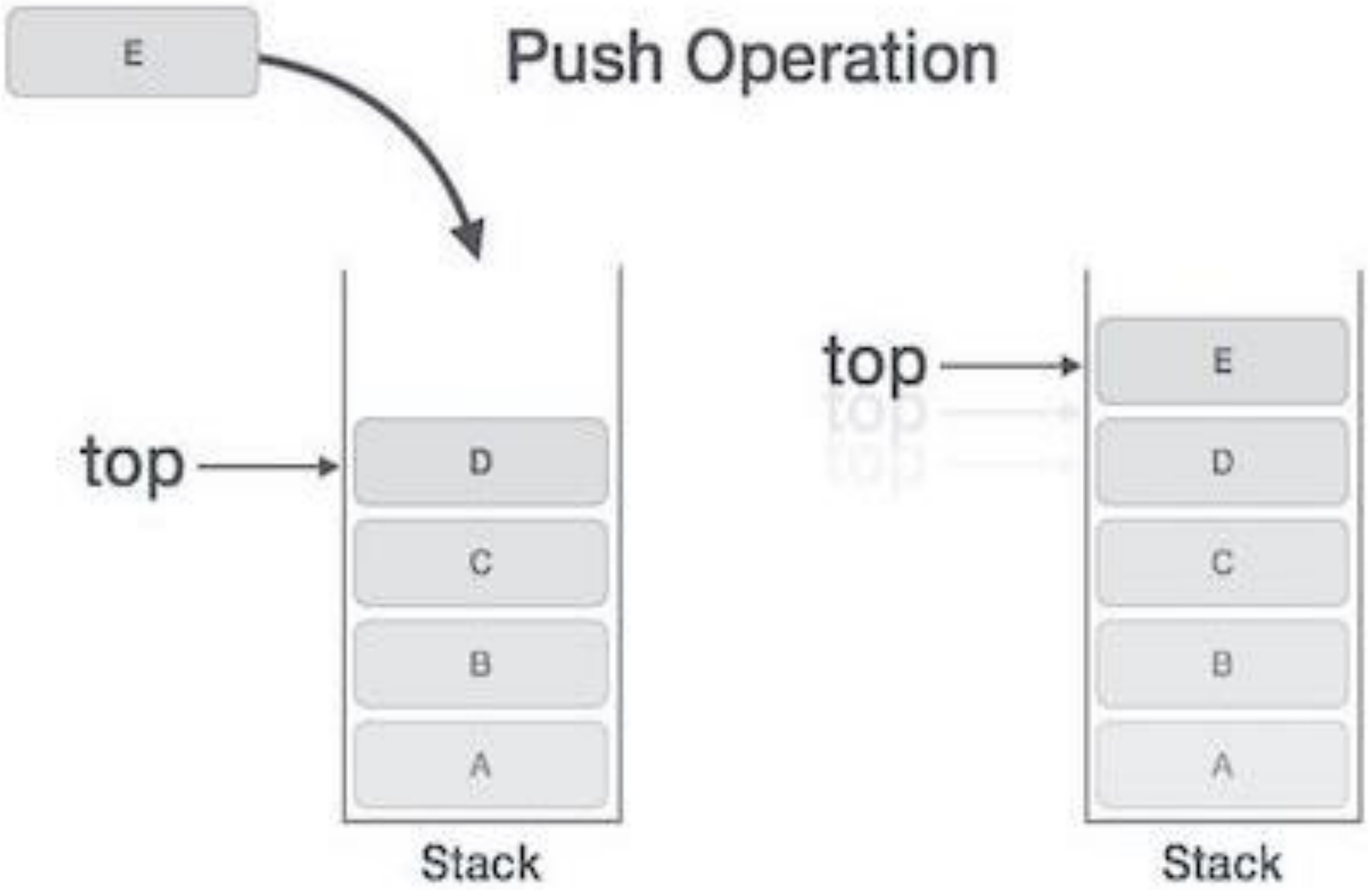
Step 1 – Checks if the stack is full

Step 2 – If the stack **is full, produces an error** and exit

Step 3 – If the stack is **not full, increments top** to point next empty space

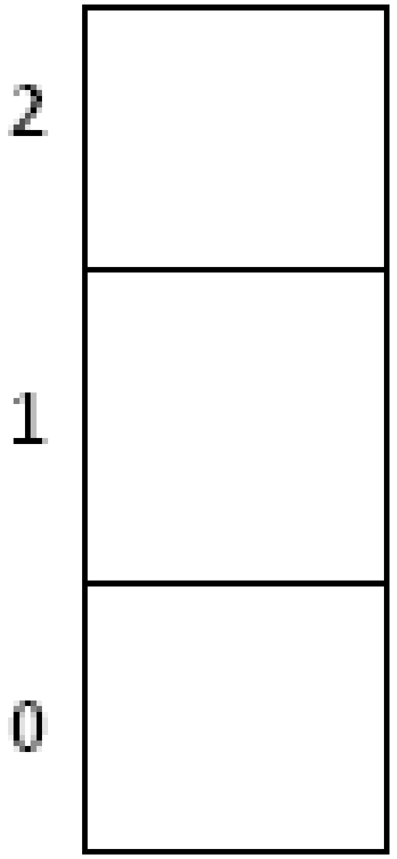
Step 4 – **Adds new data** element to the stack , where top is pointing

Step 5 – Returns success



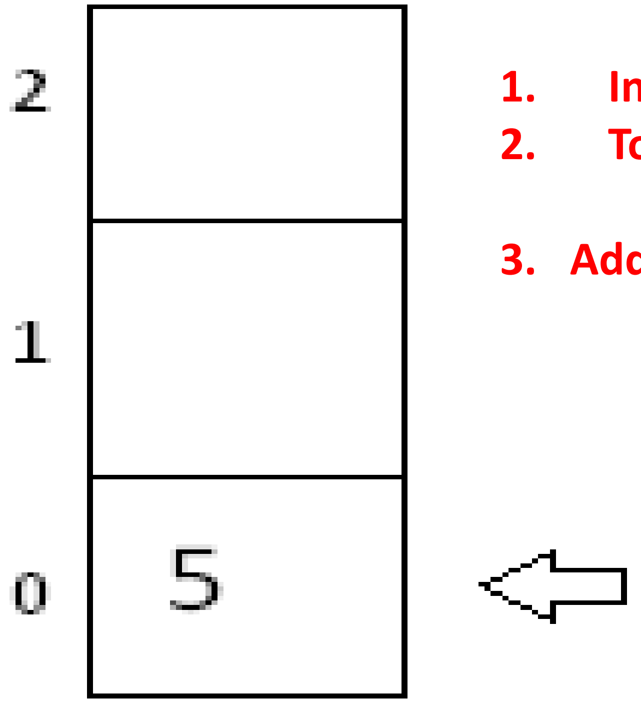
Push Operation

1)



Initially stack is empty.
top = -1.

2) push(stack, 5, 3)

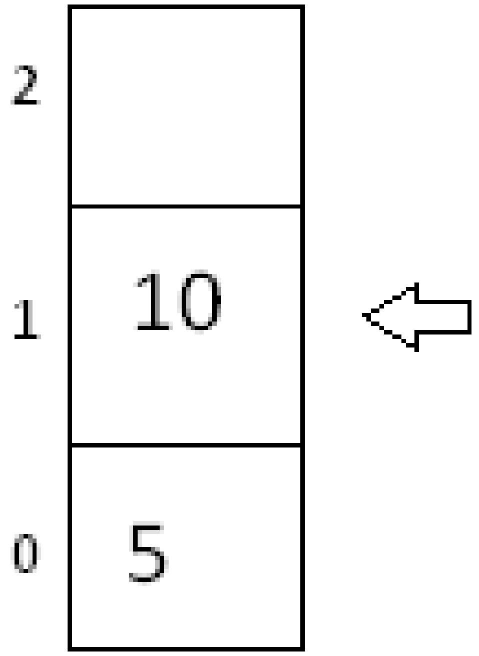


1. Increment top value by 1;
2. Top=Top+1
   = -1 + 1 =0
3. Add new element 5 on top of stack -0

top element
top = 0

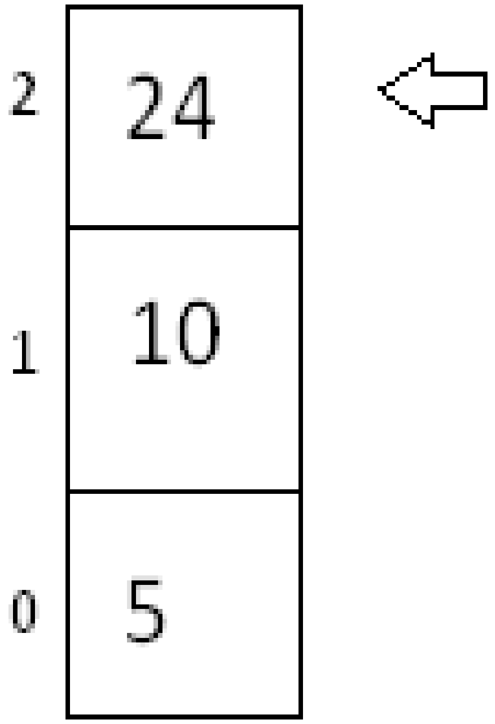
3) push(stack, 10, 3)

1. Increment top value by 1;
2. Top=Top+1
   = 0 + 1 =1
3. Add new element 10 on top 1



top element
top = 1

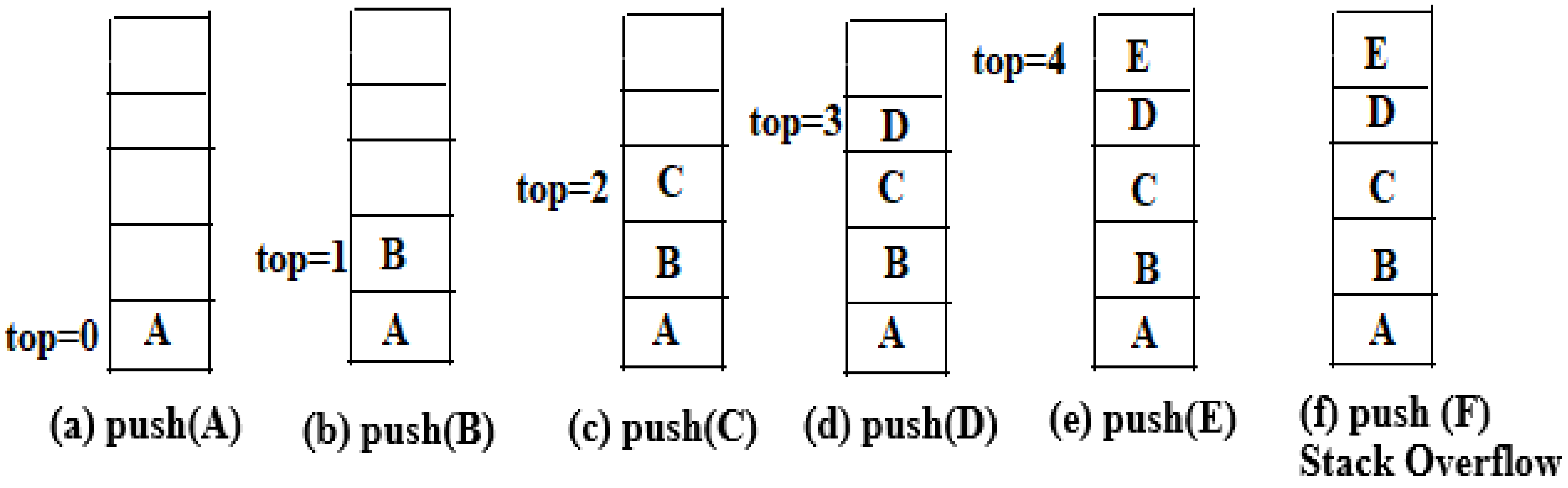
4) push(stack, 24, 3)



top element
top = 2

1. Increment top value by 1;
2. Top=Top+1
   = 1 + 1 =2
3. Add new element 10 on top 2

Example 2
push operation



An error condition that occurs when **there is no room** in the stack for adding a new item called **stack overflow** , it occurs if the stack pointer exceeds the stack bound

Pseudocode for push operation

```
void push(int data)
      {
                if( ! isFull() )                    //if stack is not full
        {           top = top + 1;              // Increment top by 1
                    stack[top] = data; }      // add new data at  the position of top
                else
        {      printf("Could not insert data, Stack is full.\n")    };
    }
```

Pop Operation

- Removing an element from the stack is known as a Pop Operation
- Pop operation involves a series of steps

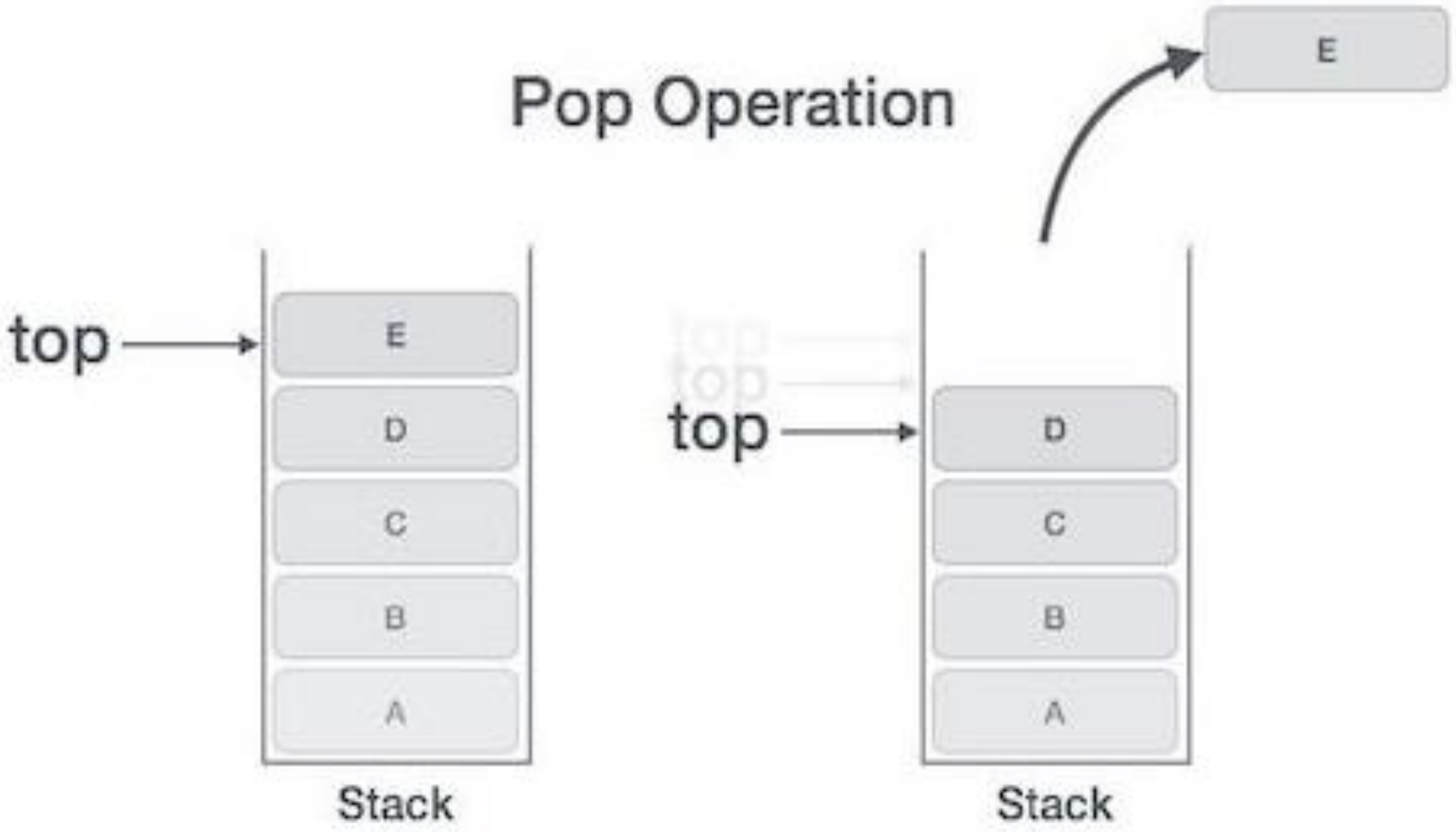
Step 1 – Checks if the stack is empty

Step 2 – If the **stack is empty, produces an error** and exit

Step 3 – else, accesses the data element at which top is pointing

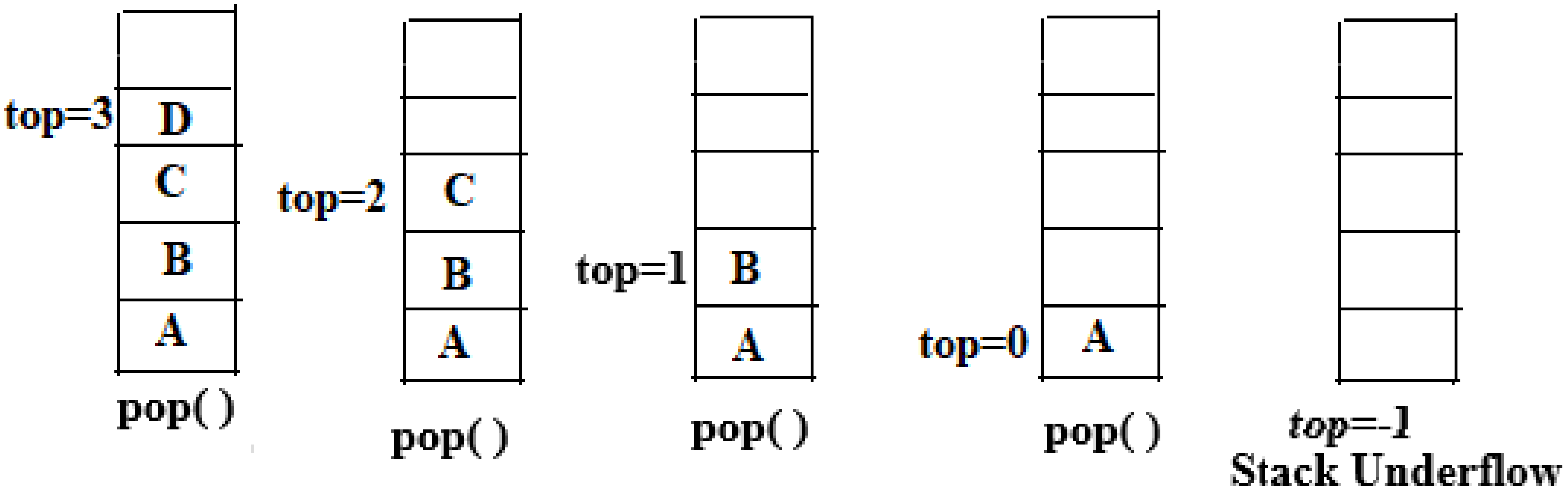
Step 4 – Decreases the value of top by 1

Step 5 – Returns success


Pop Operation
top
E
D
C
B
A
Stack
top
top
top
E
D
C
B
A
Stack

Pseudocode for pop operation

```
int pop(int data)
 {       if(!isempty())                              //if stack is not empty
         {        data = stack[top];     //save the value on top of the stack to data
top = top - 1;           // decrement top by 1
                  return data;
          }
        else
         {    printf(“Stack is empty.\n"); }
}
```



An error condition that occurs when stack is empty for deleting an element called **stack Underflow** , it occurs if the stack pointer top=-1

Stack is said to be in Overflow state when it is completely full

and

Underflow state if it is completely empty

Applications of Stacks

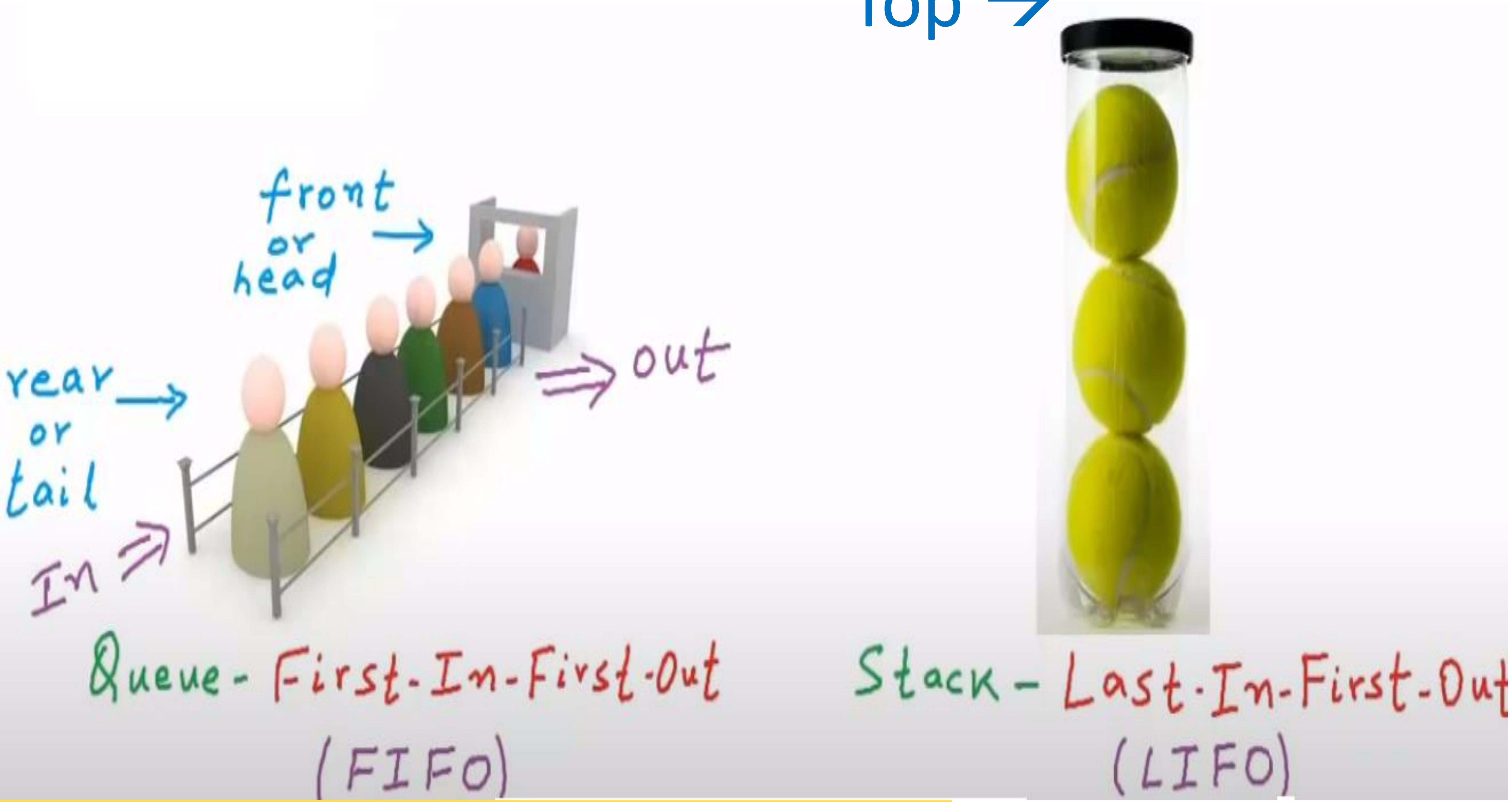
1. Reverse a string
2. Check well-formed (nested) parenthesis(Balancing the symbols)
3. Convert infix expression to postfix expressions
4. Evaluate the postfix expression

QUEUES

What is a queue?

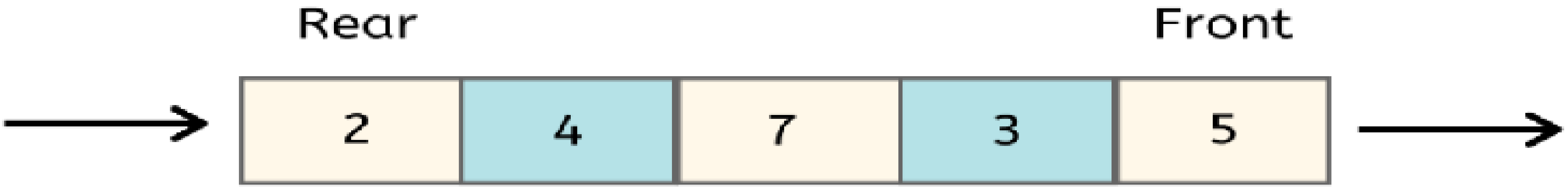
- Queue is a **Linear Data Structure**
- **Collection of similar data** items
- In which enqueue (insertion)operation performed at **rear end** and
- dequeue (deletion) operations performed at **front end**
- **FIFO**: First In, First Out / **LILO**: Last In Last Out Data Structure

Top →


front or head →
rear or tail →
In ⇒
⇒ out
Queue - First-In-First-Out (FIFO)
Stack - Last-In-First-Out (LIFO)

Queue

insertion and deletion happens at different ends



Enqueue

Insertion

Dequeue

Deletion

First in First out (FIFO)

Basic Operations of Queue (Queue ADT)

Primary Operations

enqueue() – Adds an element to the rear of the queue (end of the queue)

dequeue() – Removes an element from the front of the queue

Secondary Operations

peek() – get the front data element of the queue, without removing it

isFull() – check if queue is full

isEmpty() – check if queue is empty

Size() **-** Determines the number of elements in the queue

toString - Returns a string representation of the queue

Enqueue Operation

- The process of **adding a new data element** in end of the queue is known as a Enqueue Operation (Rear)
- Enqueue operation involves a series of steps

Step 1 – Checks if the queue is full

Step 2 – If queue **is full, produces an error** and exit

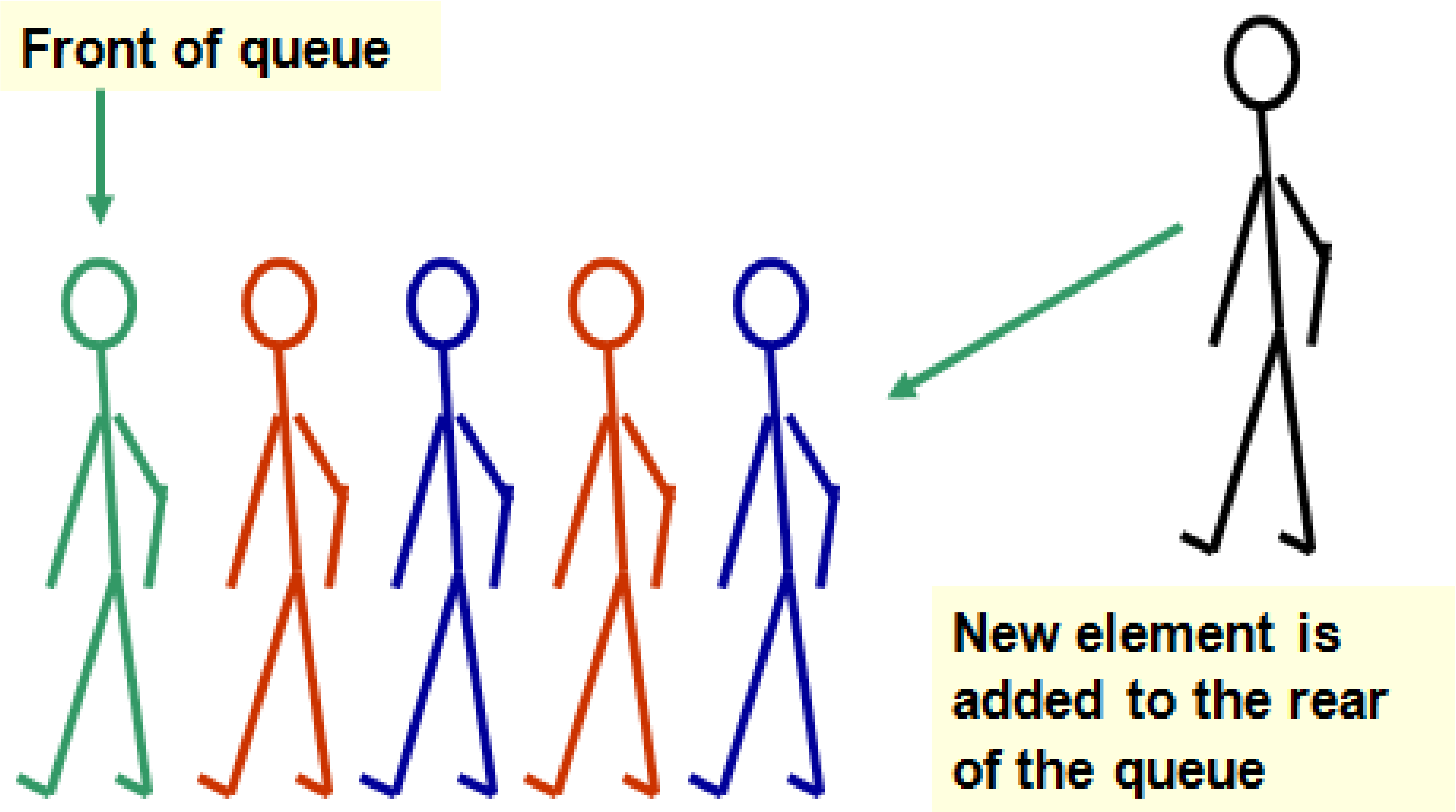
Step 3 – If queue is **not full,**

- If queue **is empty**, **increment both rear and front** pointer by one
- **else increment rear** by one which points to next empty space

Step 4 – **Adds new data** element to the queue , where rear is pointing

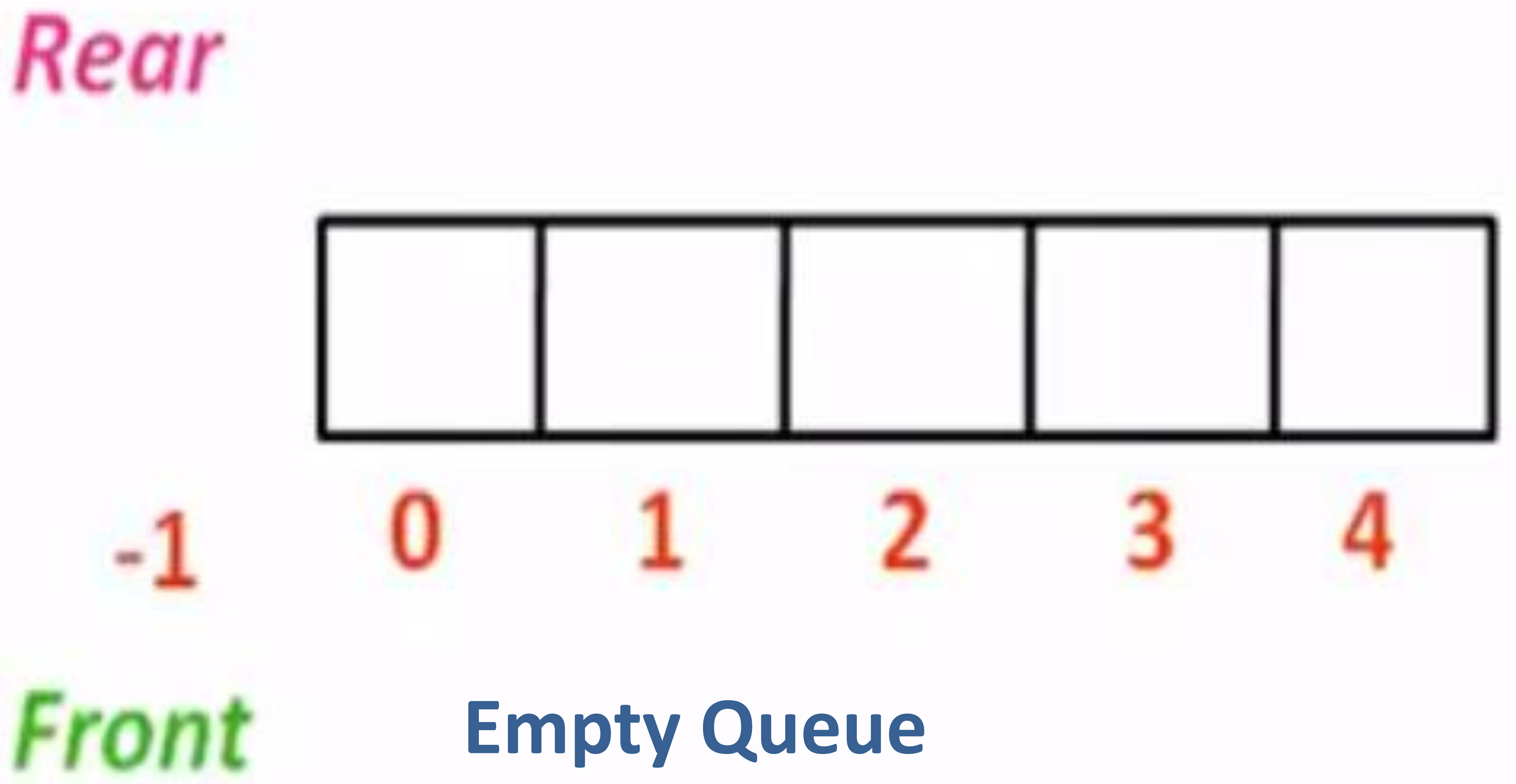
Step 5 – Returns success

enqueue() – inserting a new element at the end of queue (Rear end)

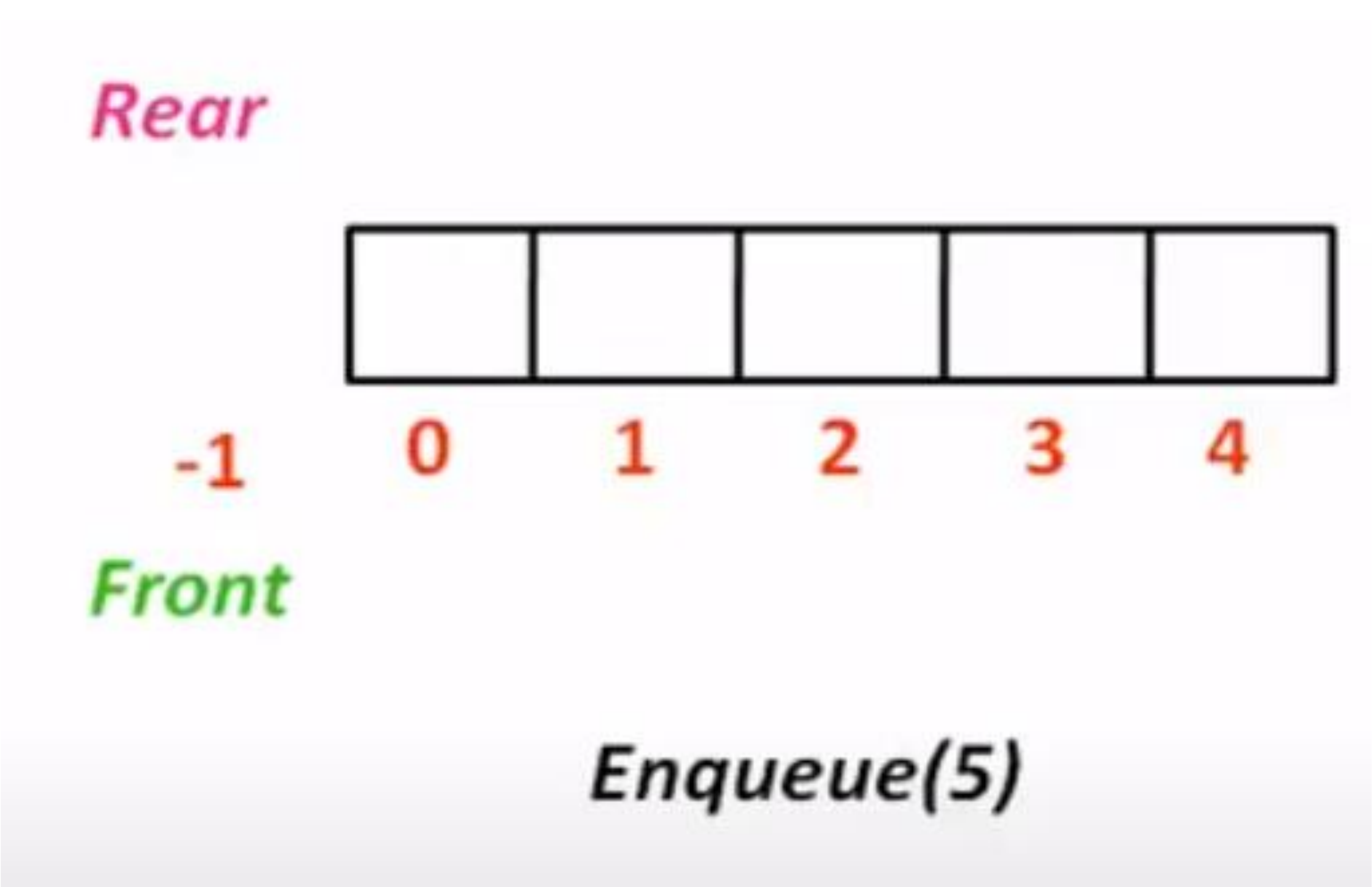


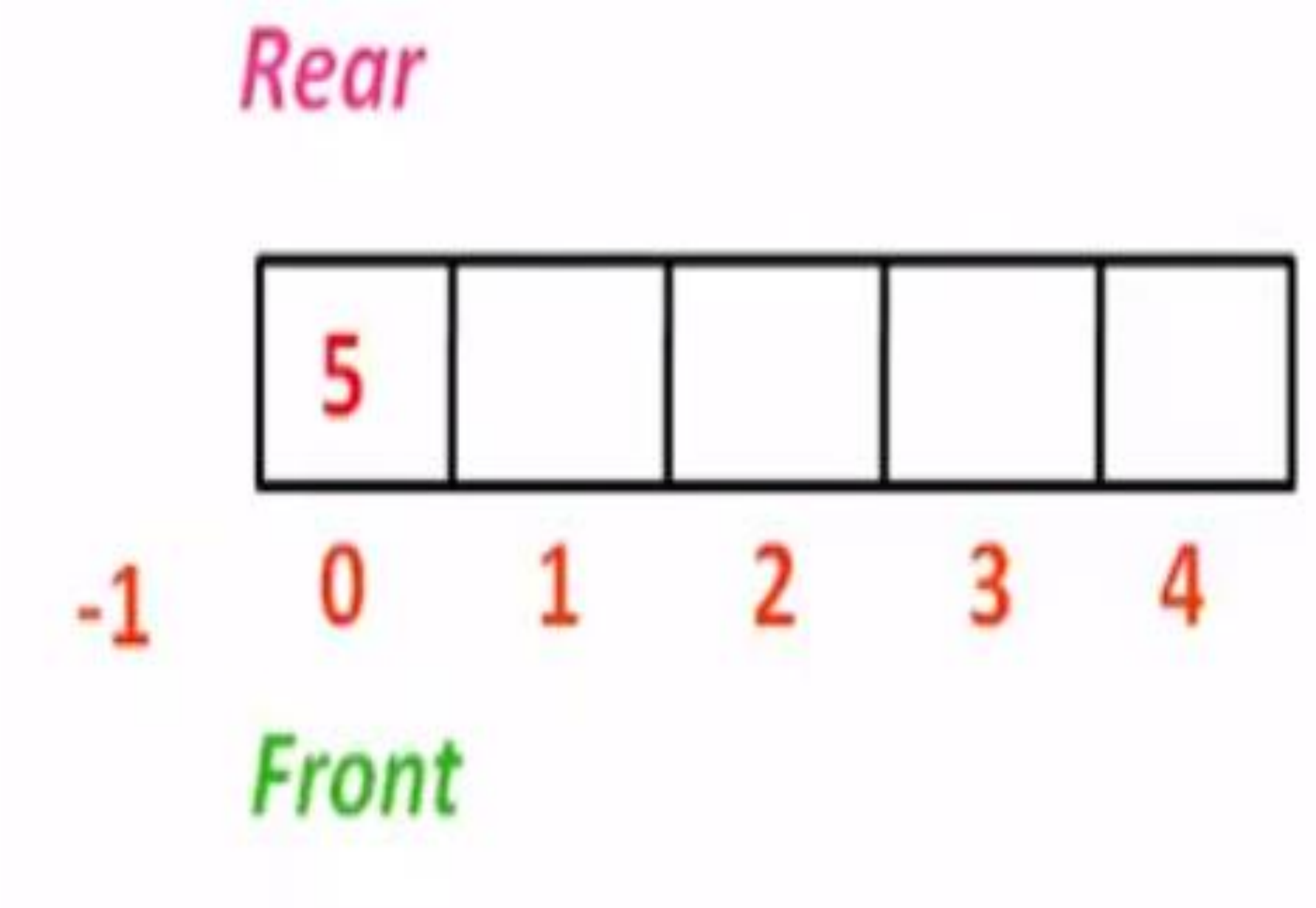
Front of queue

New element is added to the rear of the queue



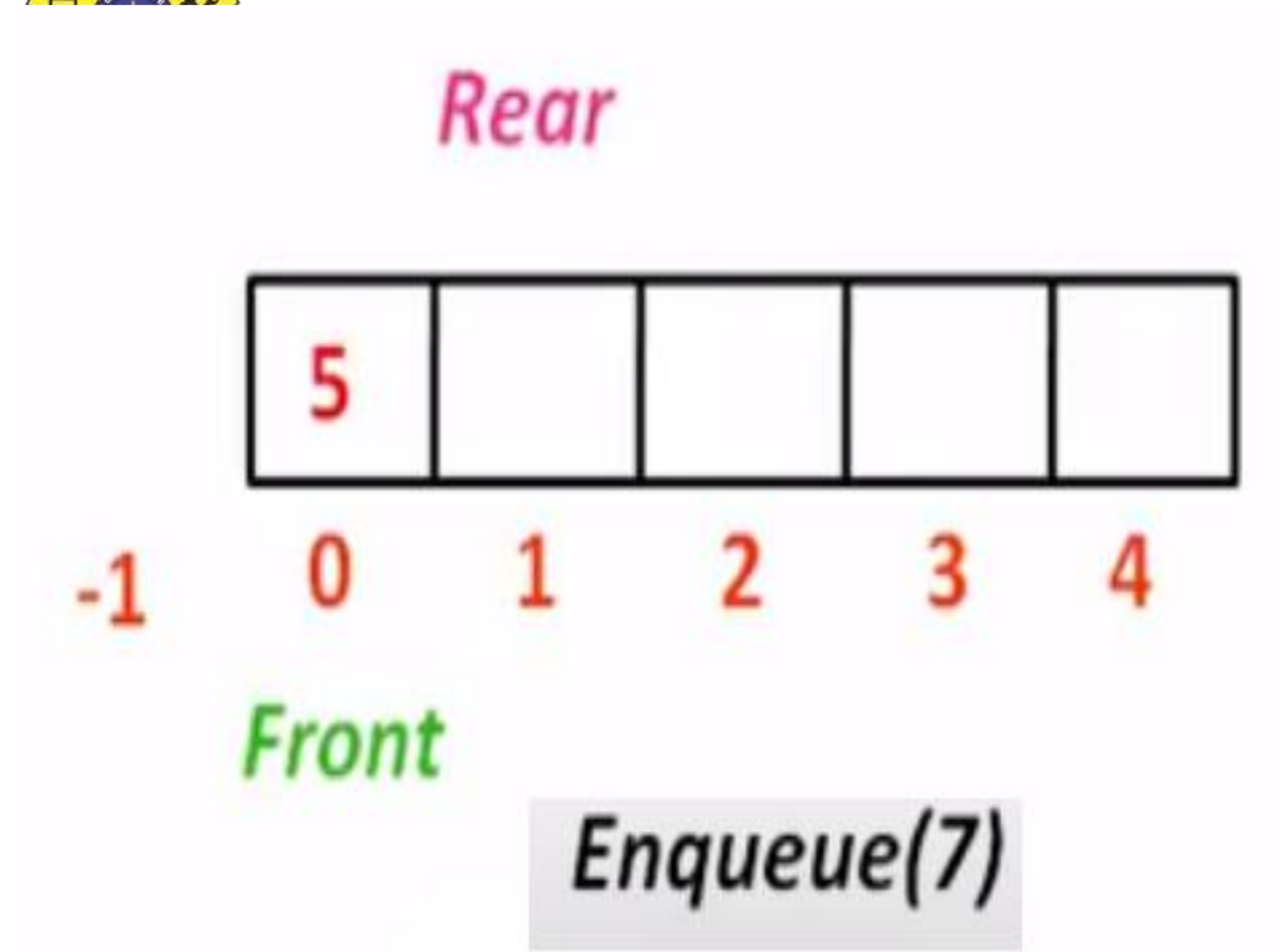
Empty Queue

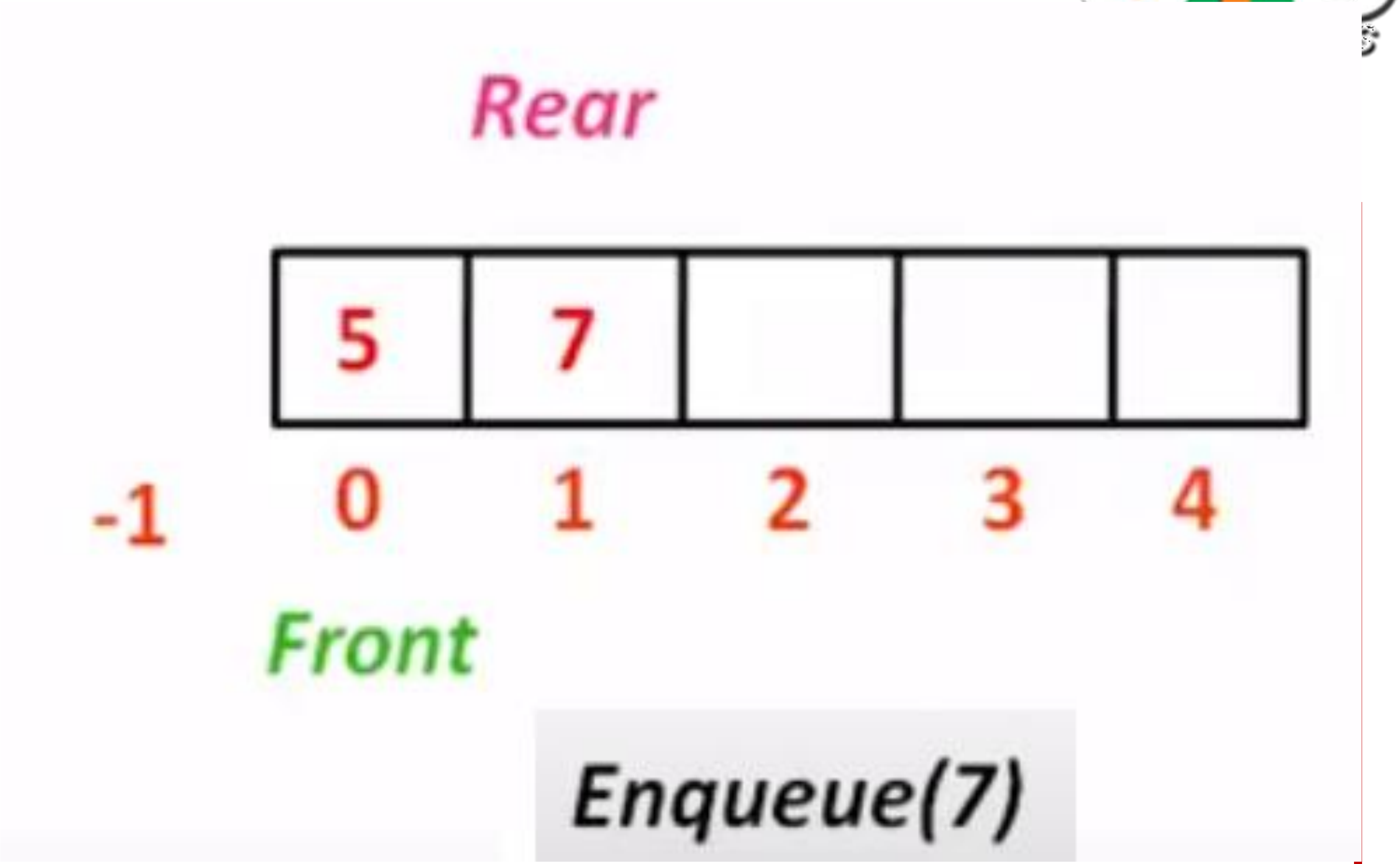




Increment front and rear

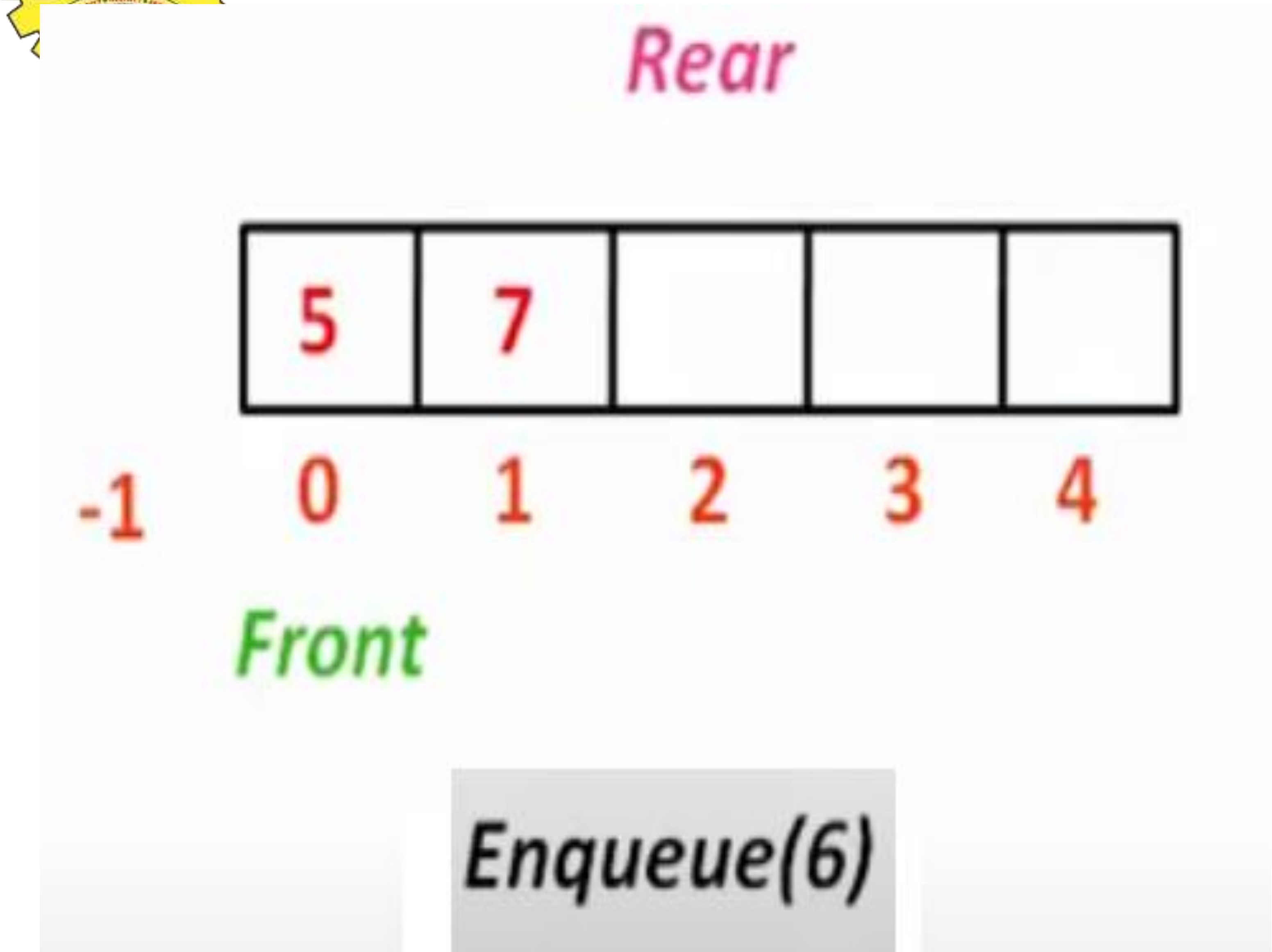
Insert 5 at that position

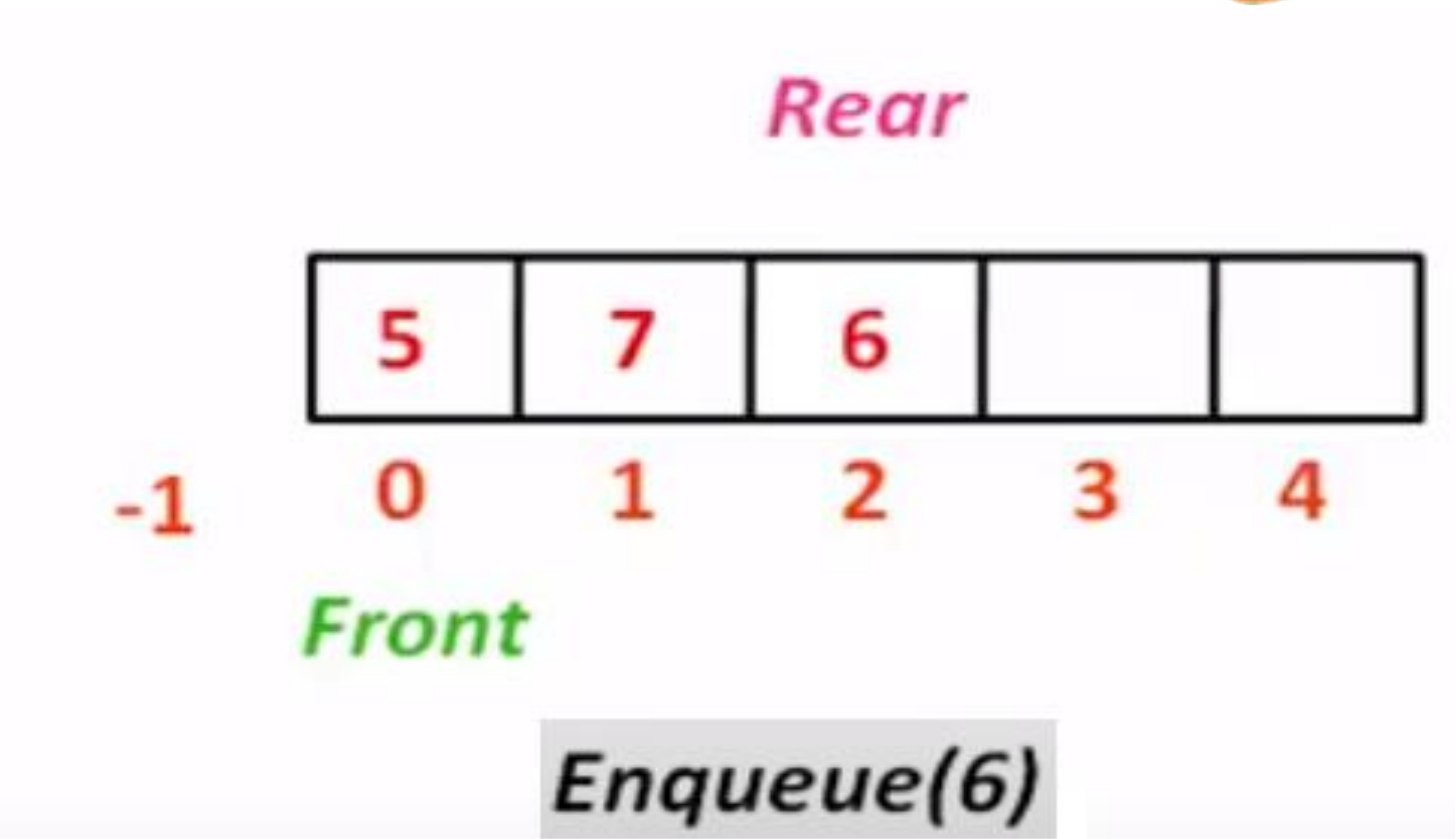




Increment rear

Insert 7 at that position

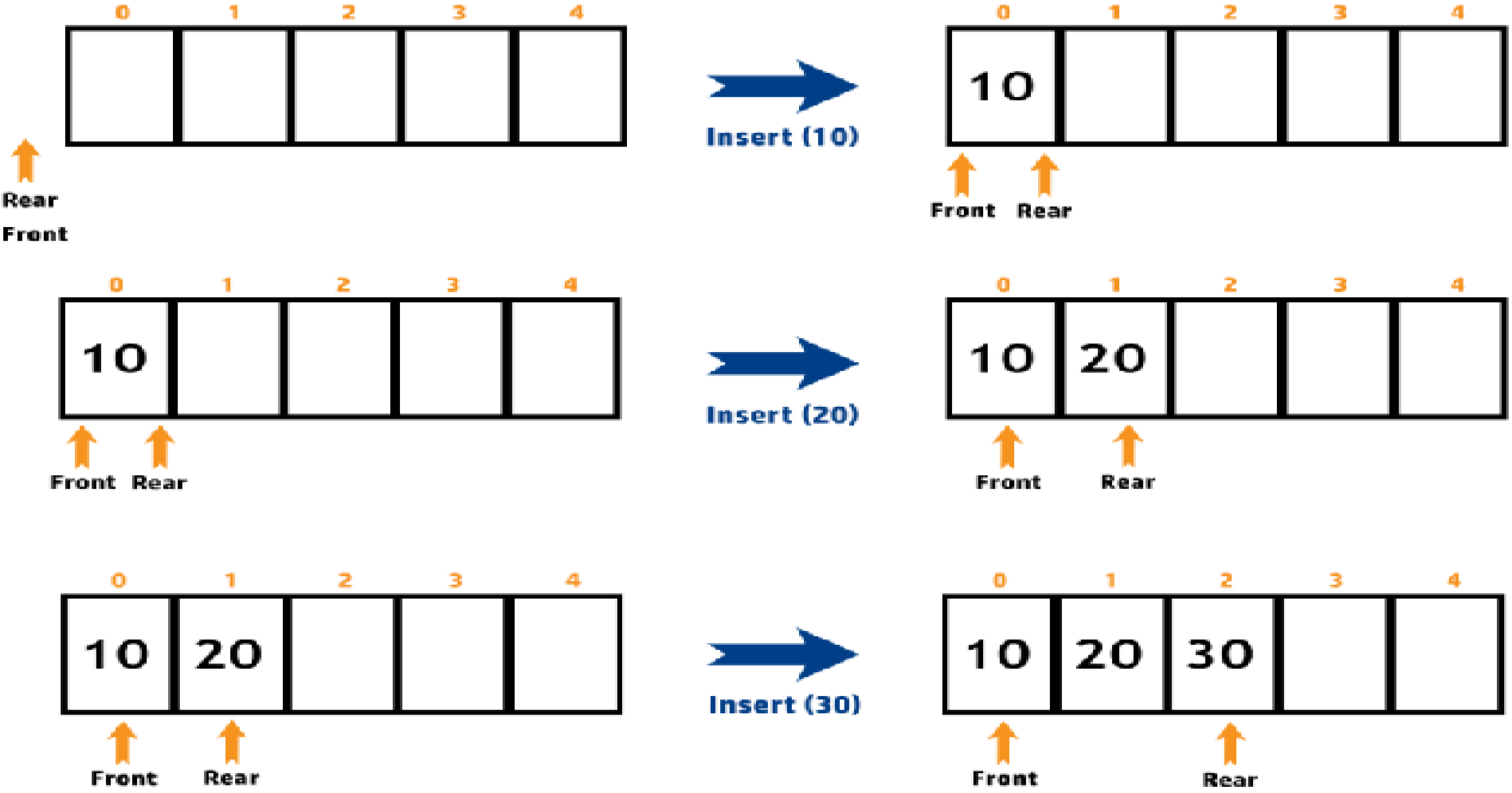


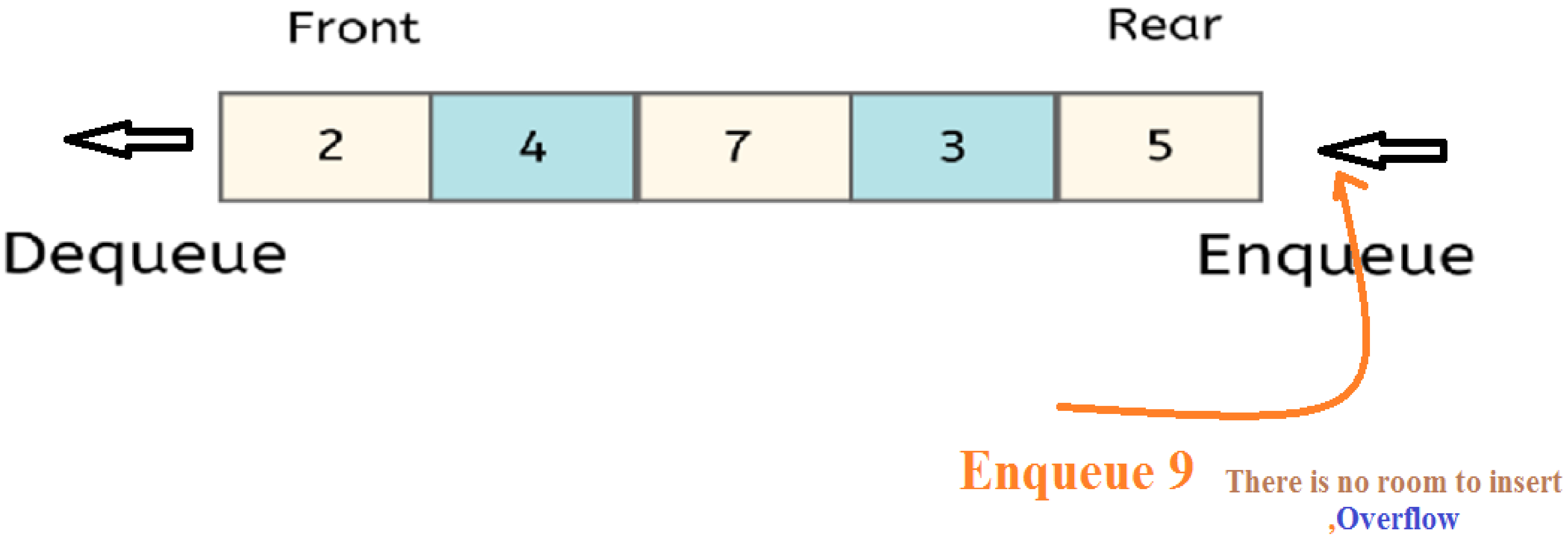


Increment rear

Insert 6 at that position

Example 2
Enqueue


0 1 2 3 4
Rear
Front
Insert (10)
0 1 2 3 4
10
Front
Rear
0 1 2 3 4
10
Front
Rear
Insert (20)
0 1 2 3 4
10 20
Front
Rear
0 1 2 3 4
10 20
Front
Rear
Insert (30)
0 1 2 3 4
10 20 30
Front
Rear



An error condition that occurs when **there is no room** in the queue for adding a new item called **queue overflow** , it occurs if the queue pointer exceeds the queue bound

Pseudocode for enqueue operation

```
void enqueue(int data)
    {     printf("Enter data to insert in a queue\t");
          scanf("%d",&data);
            if( ! isFull() )                        //if Queue is not full
             {        if(front=-1)                   //if Queue is empty
                 {    front=front +1; rear=rear+1; // Increment front & rear by 1
                      queue[rear] = data; } // add new data at  the position of rear
                       else                    //if Queue is not empty
                 {   rear=rear+1;    queue[rear] = data; }    // Increment rear by 1
        }    printf("Could not insert data, Queue is full.\n")   ;
  }
```

Dequeue Operation

- Removing an element from the queue at front end is known as a Dequeue Operation
- dequeue operation involves a series of steps

Step 1 – Checks if the queue is empty (**front==-1**)

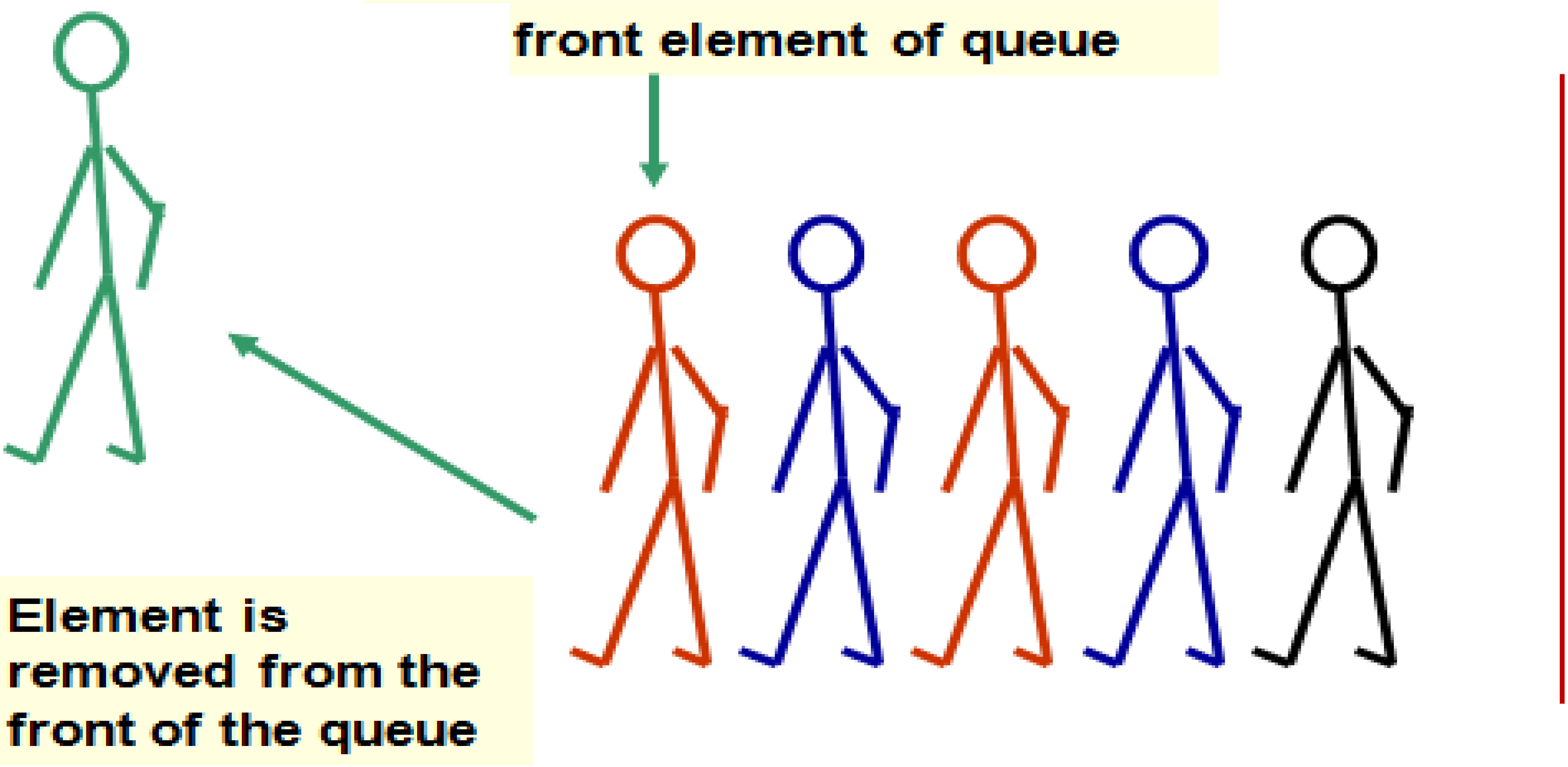
Step 2 – If the **queue is empty,** produces an **error** and exit

Step 3 – **else, remove** the data element at which **front** is pointing

Step 4 – **Increment the value of front** by 1

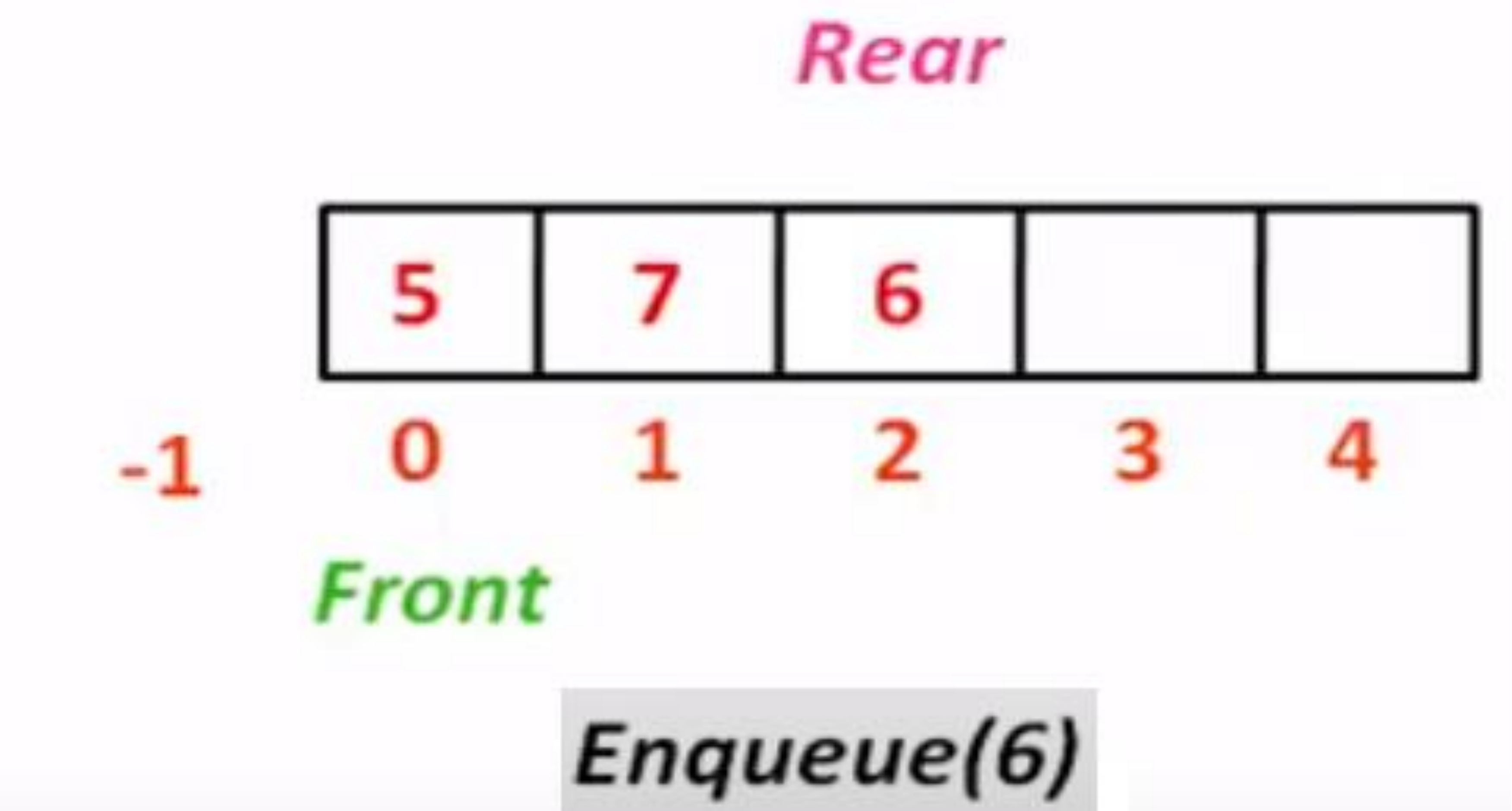
Step 5 – Returns success

dequeue() – Removing an front element from the queue

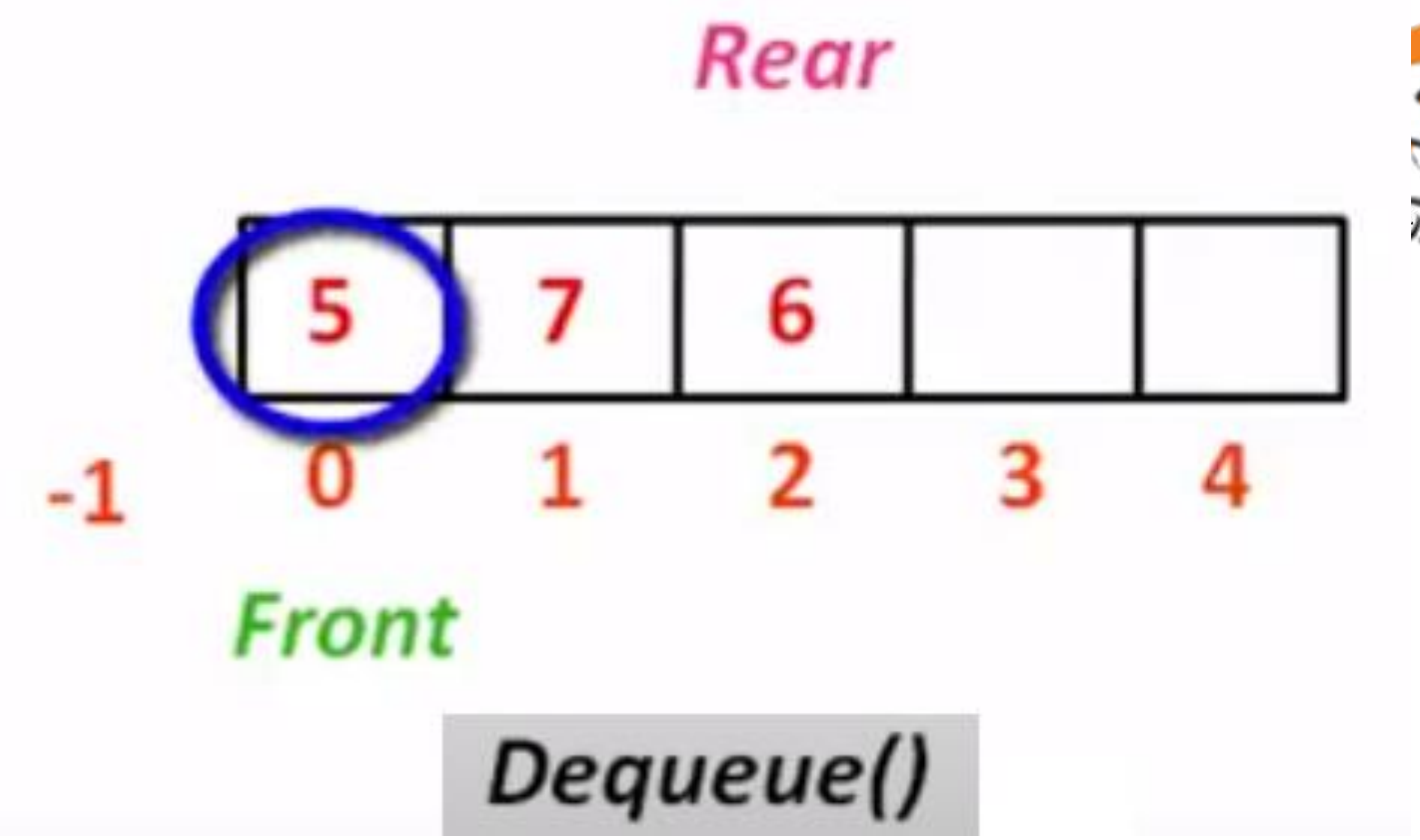


front element of queue

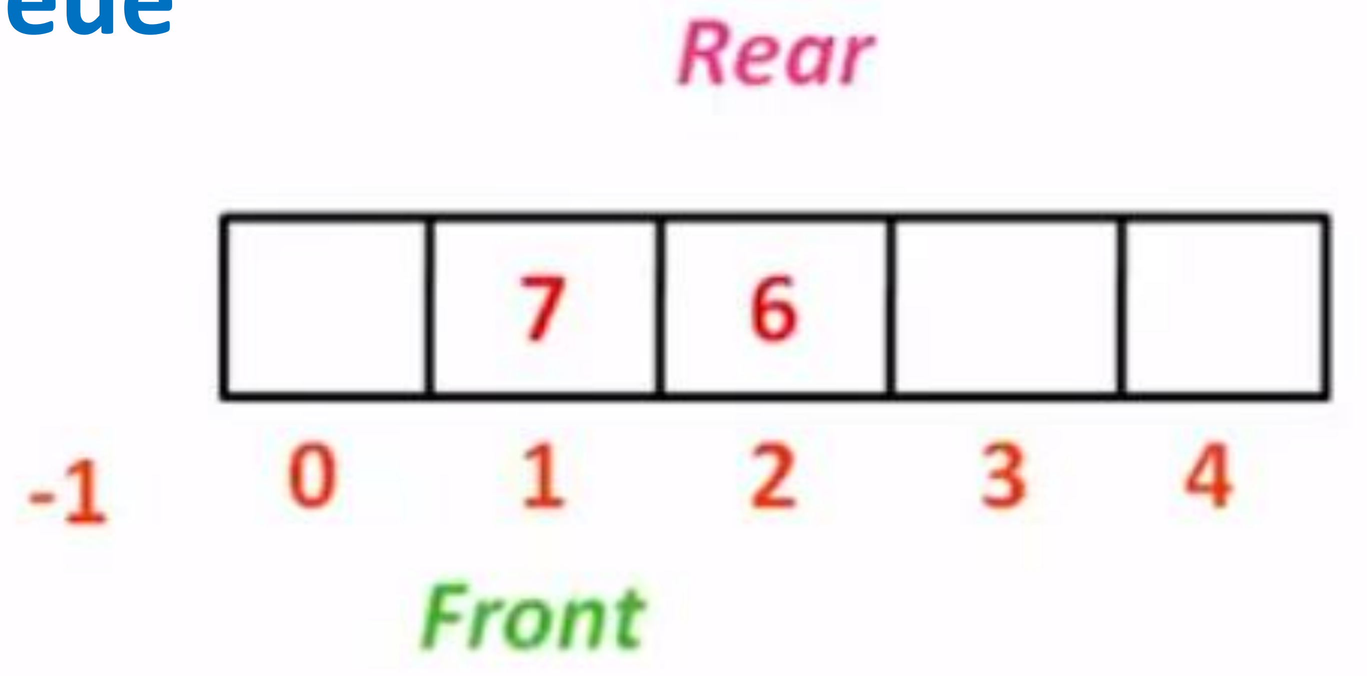
Element is removed from the front of the queue



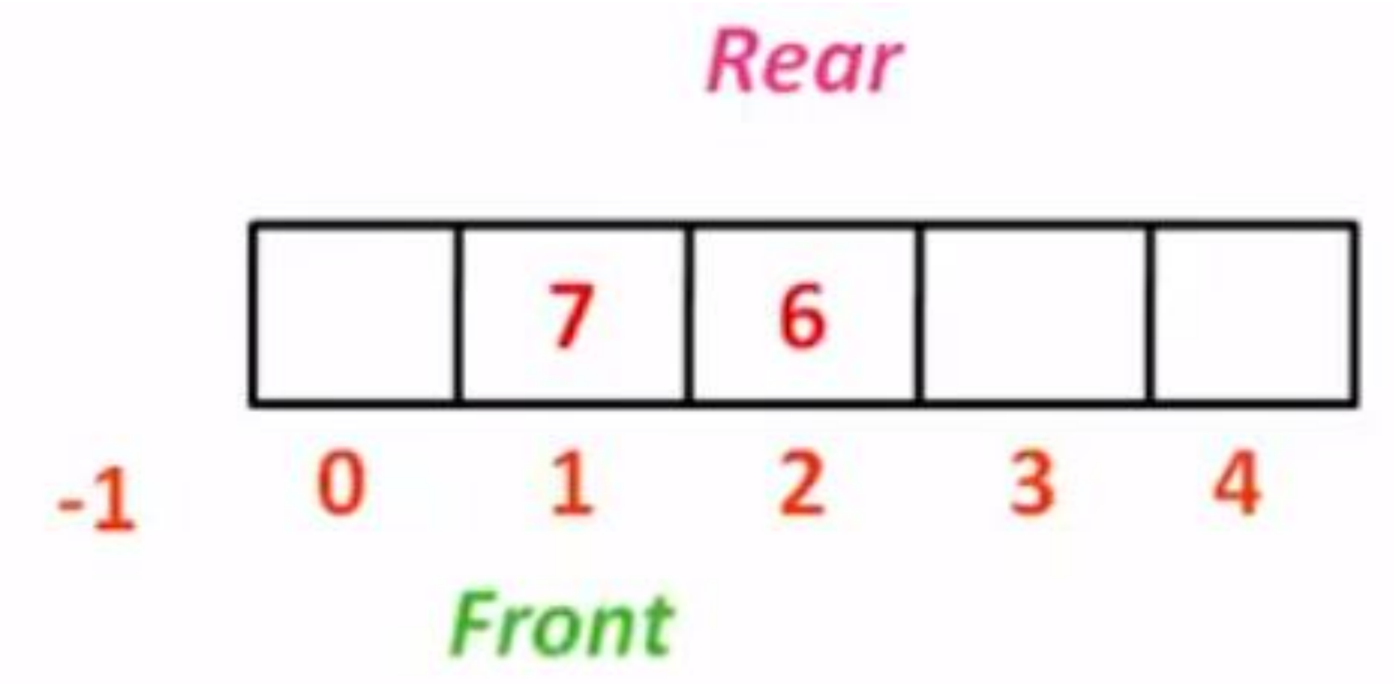
Initial Queue

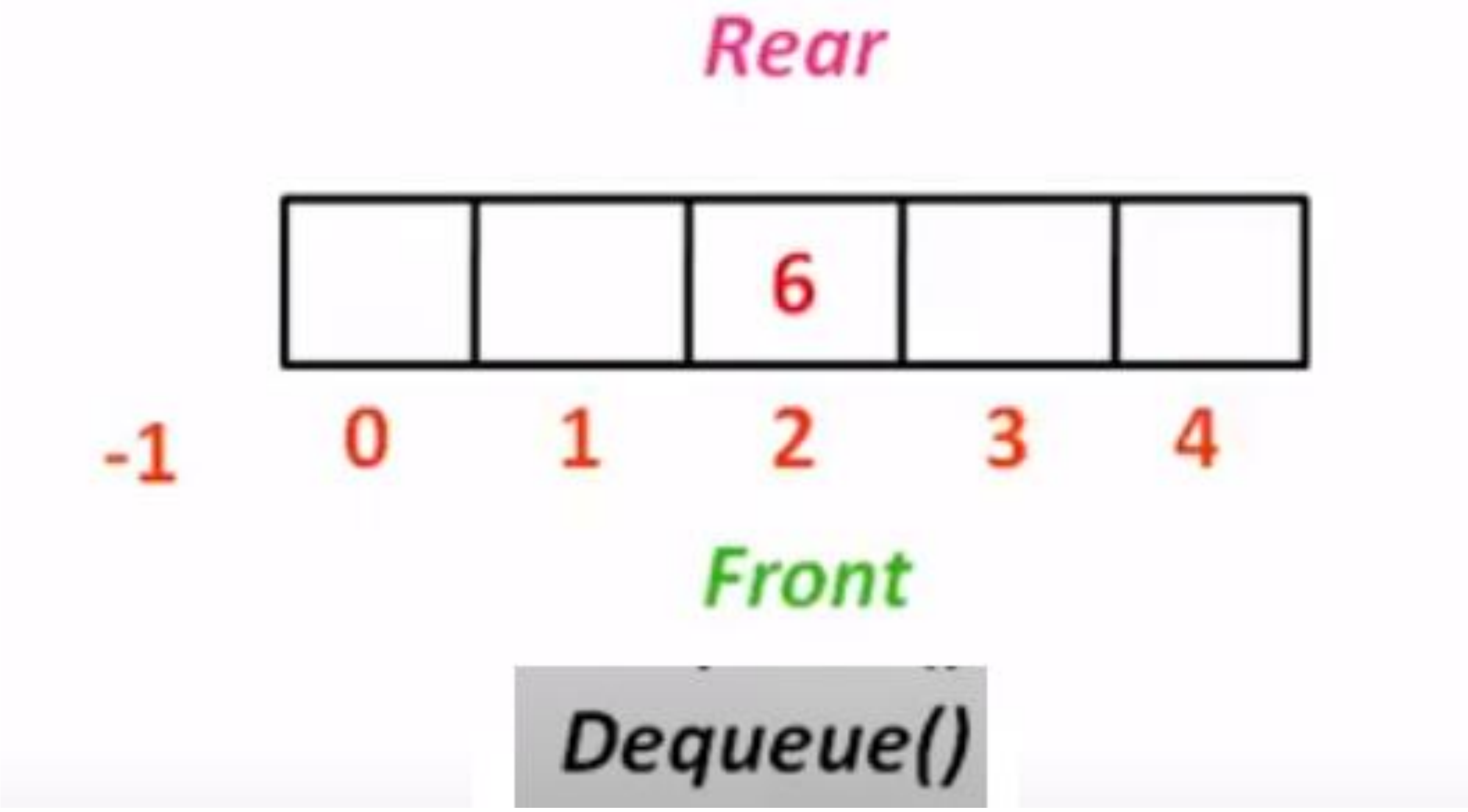


Dequeue ()-remove 5 from queue



Increments front value





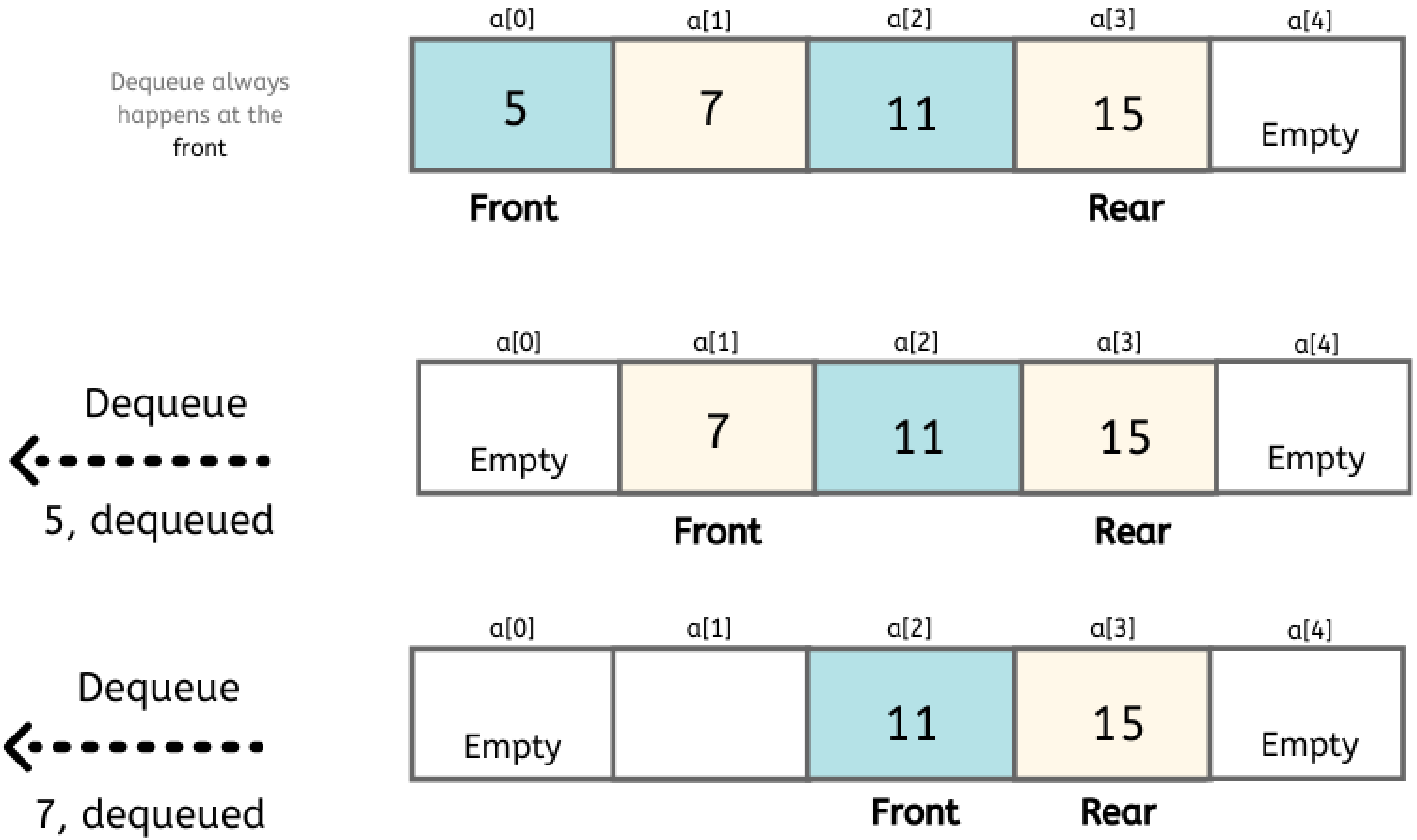
Dequeue remove 7 from queue, increment front to 2

Example 2

Dequeue

Enqueue always happens at the rear

Dequeue always happens at the front



a[0]	a[1]	a[2]	a[3]	a[4]
5	7	11	15	Empty
Front			Rear	

Dequeue

5, dequeued

a[0]	a[1]	a[2]	a[3]	a[4]
Empty	7	11	15	Empty
	Front		Rear	

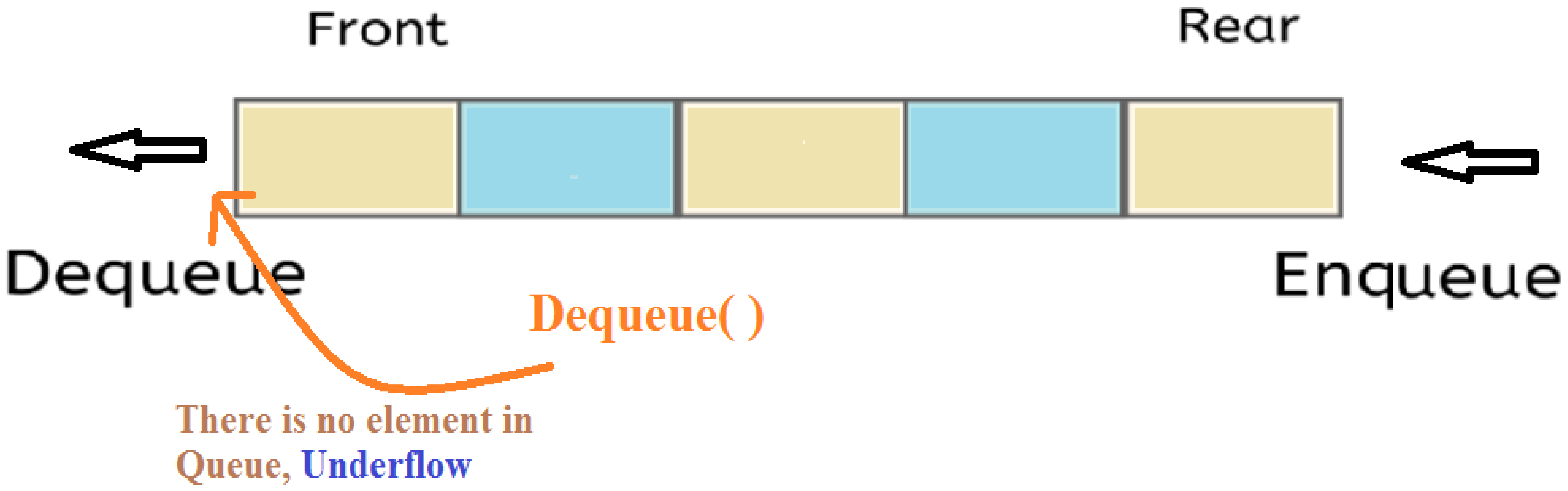
Dequeue

7, dequeued

a[0]	a[1]	a[2]	a[3]	a[4]
Empty		11	15	Empty
		Front	Rear	

Pseudocode for dequeue operation

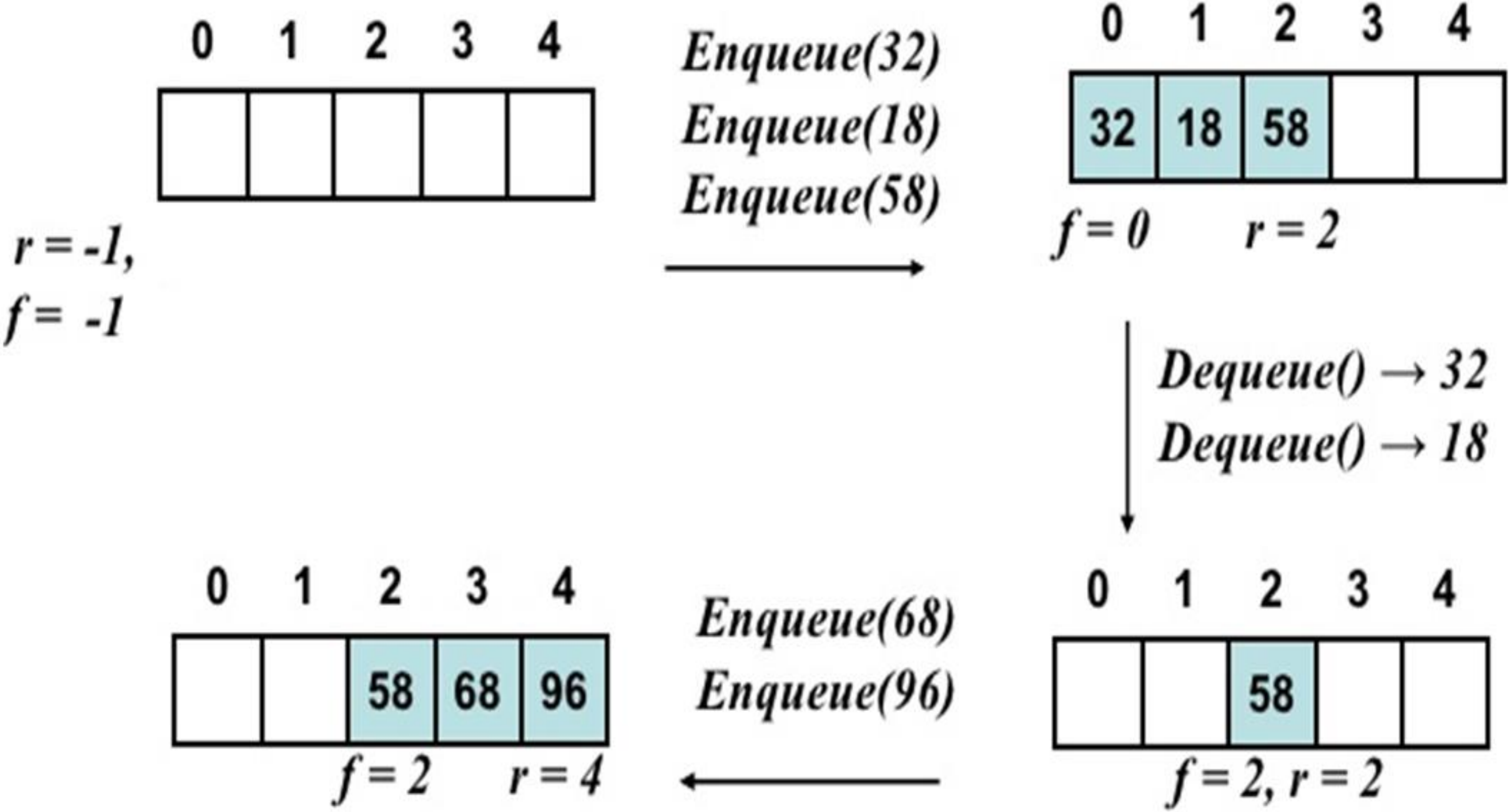
```
int dequeue(int data)
 {       if(! isempty())                 //if queue is not empty
         {       data = queue[front]; //save the value on front of the queue to data
                 front = front+ 1;     // increment front by 1
                 return data;
         }
        else
         {    printf("queue is empty\n"); }
}
```



An error condition that occurs when queue is empty for deleting an element called **Queue Underflow** , it occurs if the Queue ,pointer front=-1

Example

Enqueue operation & Dequeue operation



0	1	2	3	4

r = -1,
f = -1

Enqueue(32)
Enqueue(18)
Enqueue(58)

0	1	2	3	4
32	18	58		

f = 0 *r = 2*

Dequeue() → 32
Dequeue() → 18

0	1	2	3	4
		58		

f = 2, r = 2

Enqueue(68)
Enqueue(96)

0	1	2	3	4
		58	68	96

f = 2 *r = 4*

Queue is said to be in Overflow state when it is full (rear=max_size_queue)

and

Underflow state if it is completely empty(front=-1)

Distinguish between stack and queue

Si.No	STACK	QUEUE
1	It is LIFO(Last In First Out) data structure	It is FIFO (First In First Out) data structure.
2	Insertion and deletion take place at only one end called top	Insertion takes place at rear and deletion takes place at front.
3	It has only one pointer variable (top)	It has two pointer variables(rear & front)
4	No memory wastage	Memory wastage in linear queue
5	Operations: 1.push() 2.pop()	Operations: 1.enqueue() 2.dequeue()
6	In computer system it is used in procedure calls	In computer system it is used time/resource sharing
7.	Plate counter at marriage reception is an example of stack	Student standing in a line at fee counter is an example of queue.

Thank you