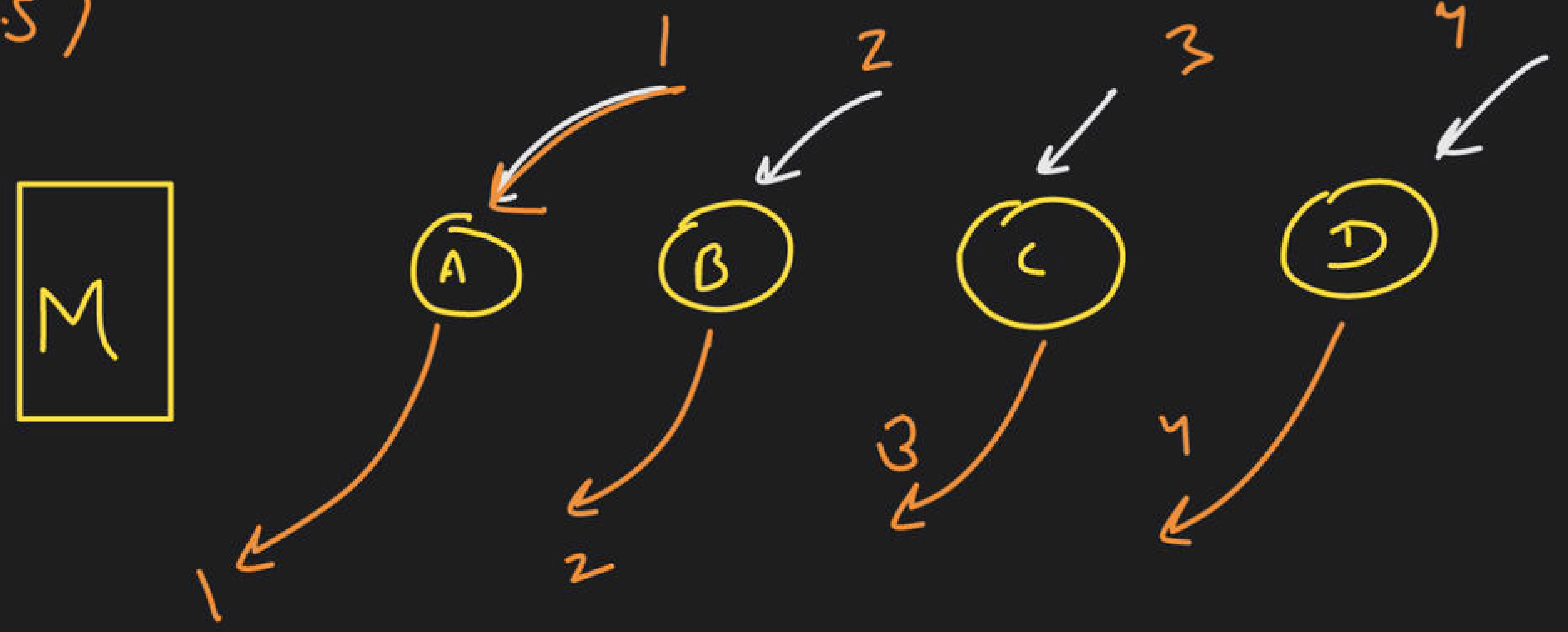
Queue - I & Doubt Clearing Session

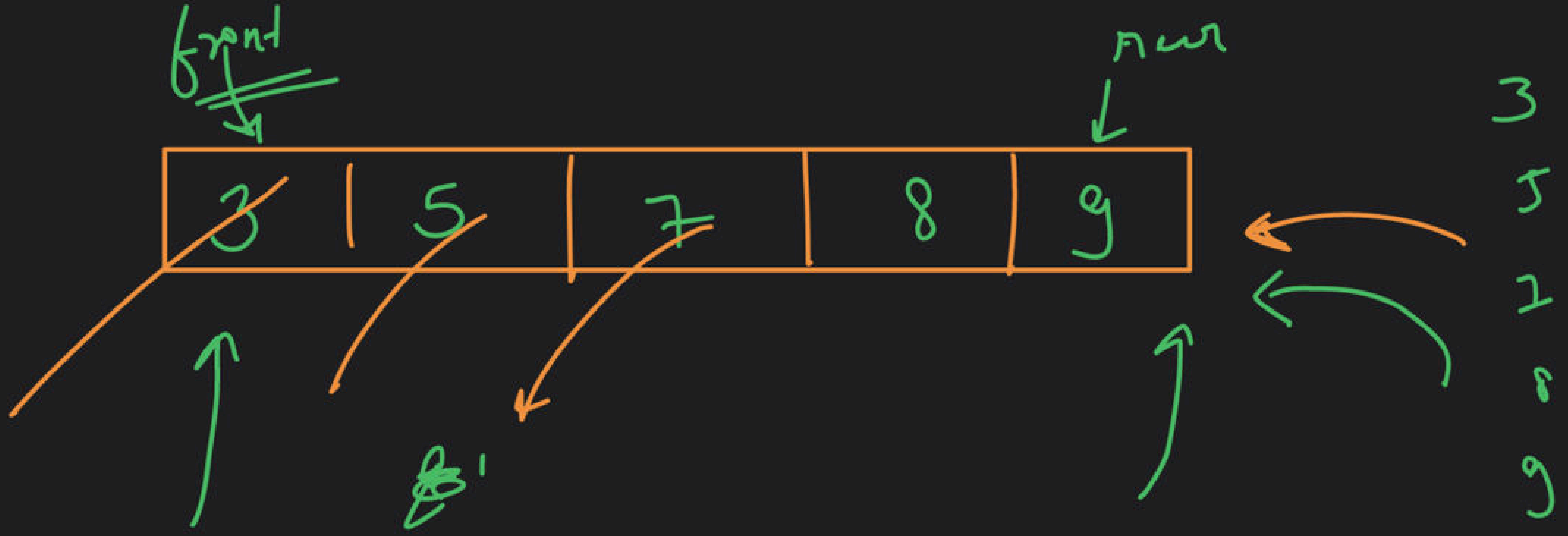
Foundation Course on Data Structures & Algorithms - Part II

Love Babbar • Lesson 4 • July 2, 2022

→ Queue (D.S)



FIFO → ordering

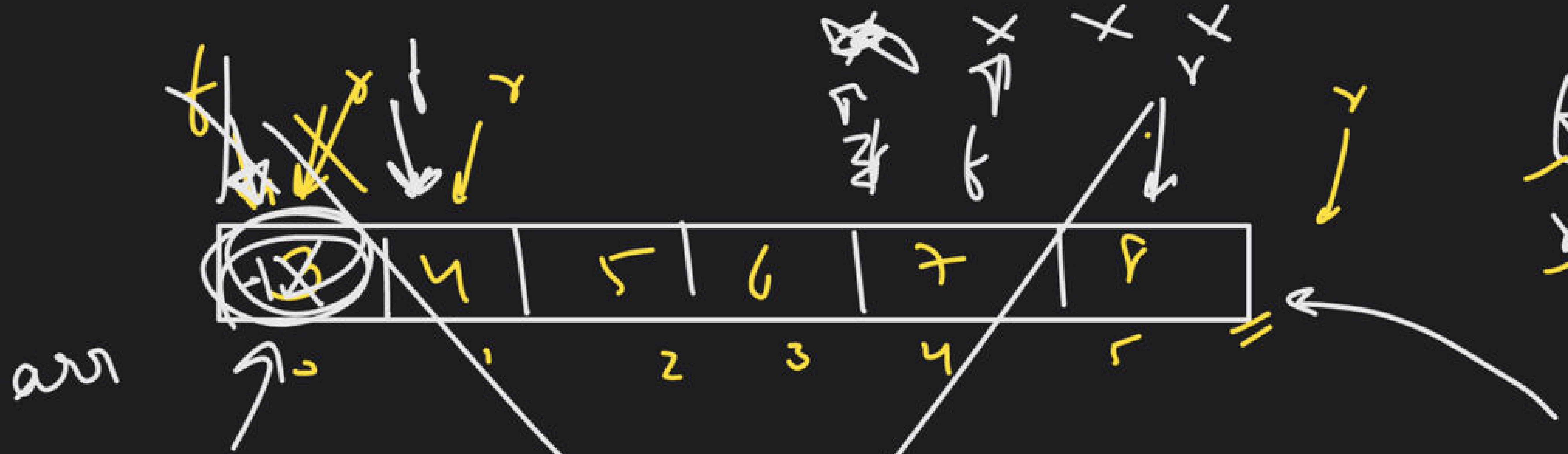

front
rear
3
5
7
8
9
3
5
7
8
9

$$\begin{tikzcd}
{} \arrow[r] & \text{Queue} \arrow[r] & \text{STL} \arrow[d] & \\
& & \text{creation} \arrow[r] \arrow[d] & \texttt{queue <char> a;} \\
& & \text{push} \arrow[r] \arrow[d] & \texttt{a.push('z');} \\
& & \text{pop} \arrow[r] \arrow[d] & \texttt{a.pop()} \\
\text{empty} \arrow[r] & \texttt{a.empty()} & \text{front()} \arrow[r] \arrow[d] & \texttt{a.front()} \\
& & \text{size} \arrow[r] \arrow[ul, bend left] & \texttt{a.size()}
\end{tikzcd}$$

Queue -> class implementation → array ->
→ linked list

```
Class Queue
{
        arr[];
        front;
        rear;
        size;

        Queue()  | push() | pop() | isEmpty | getSize()
        { }
                                   | overflow
}
```



front = 0

rear = 0

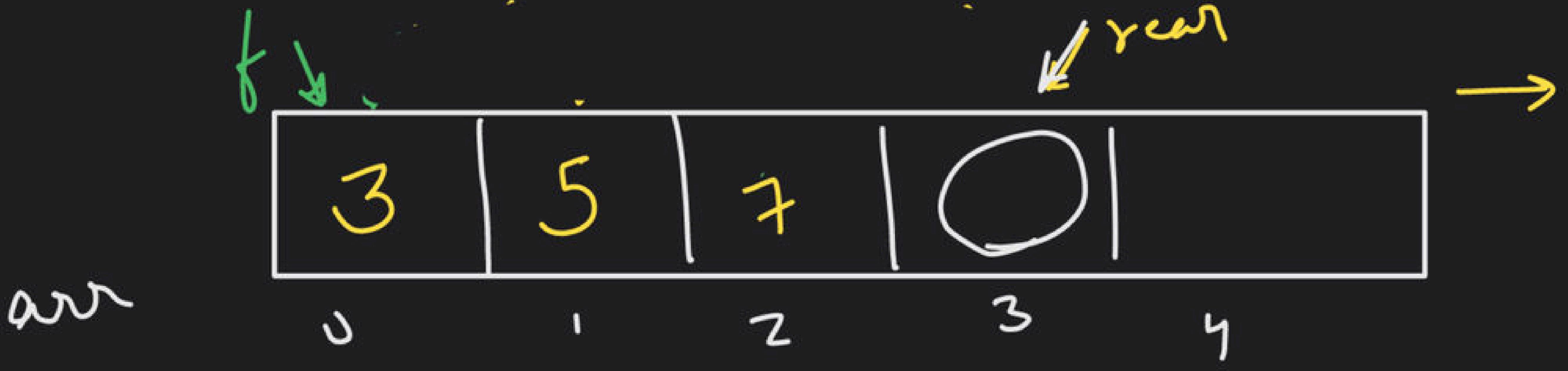
size = n

push → arr[rear] = element; } → if (rear < size)

rear++;

pop → arr(front) = -1;

front++



front = 0

rear = 0

q.push(3)

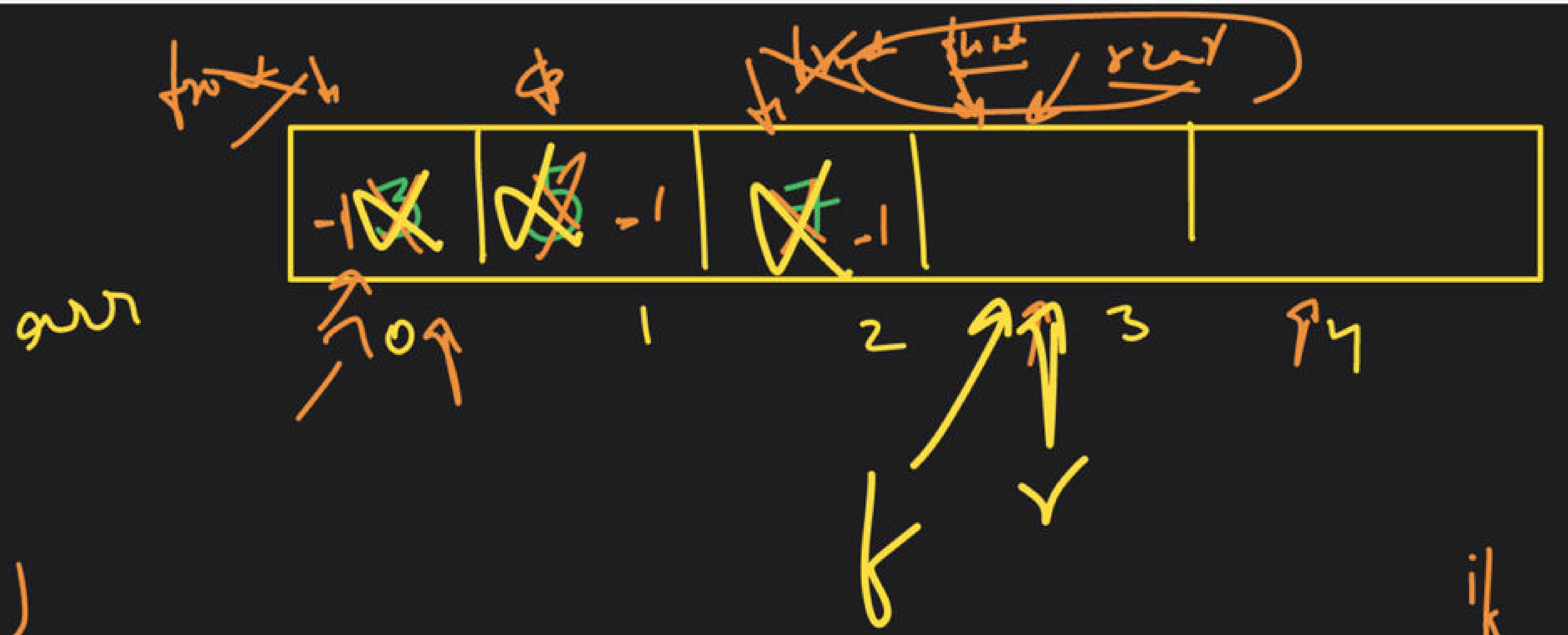
push → rear

```
arr [rear] = element;
rear++
```

if (rear >= size)
    cout << overflow;
else

q.push(5)

q.push(7)



q.pop()

pop → ~~f~~ front

arr(front) = -1

front++;

if (front == rear)

q → empty

cout << UF;

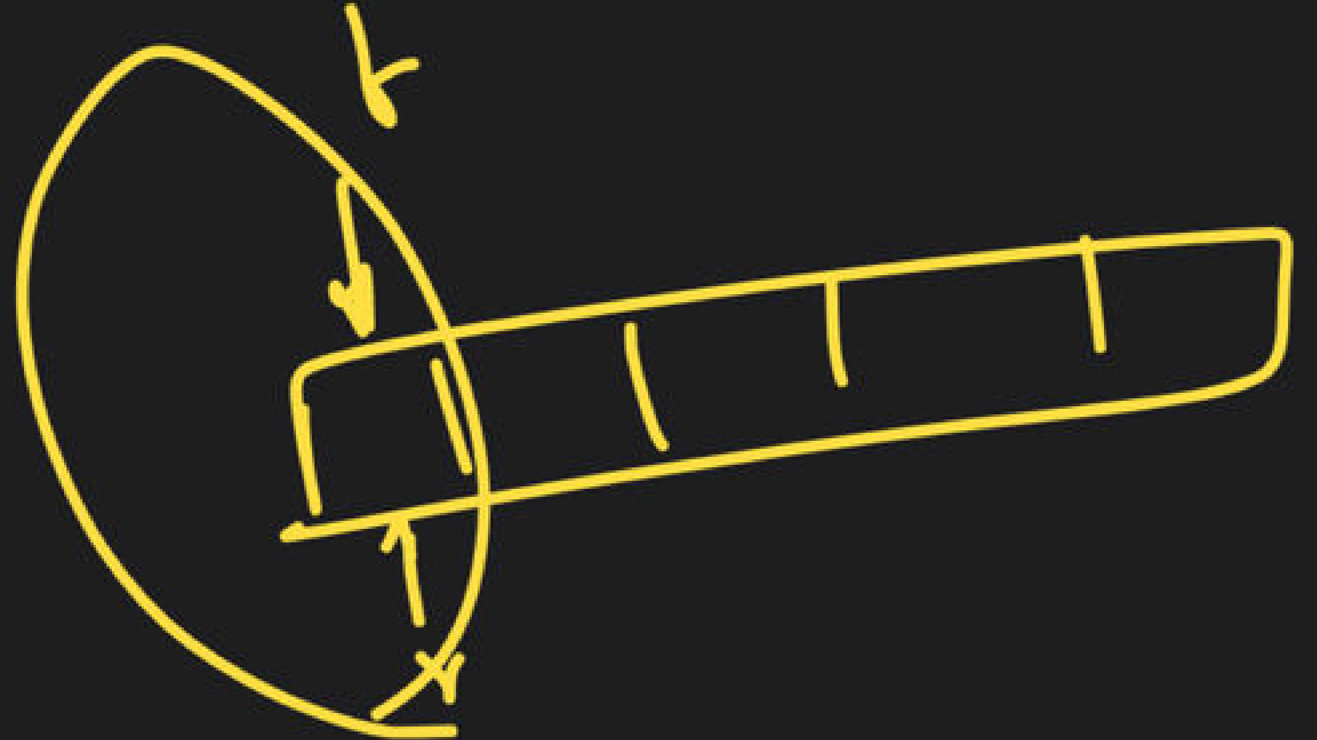
else
{ arr(front) = -1

front++

if (front == rear)
front = 0
rear = 0
}

q.pop()

q.pop



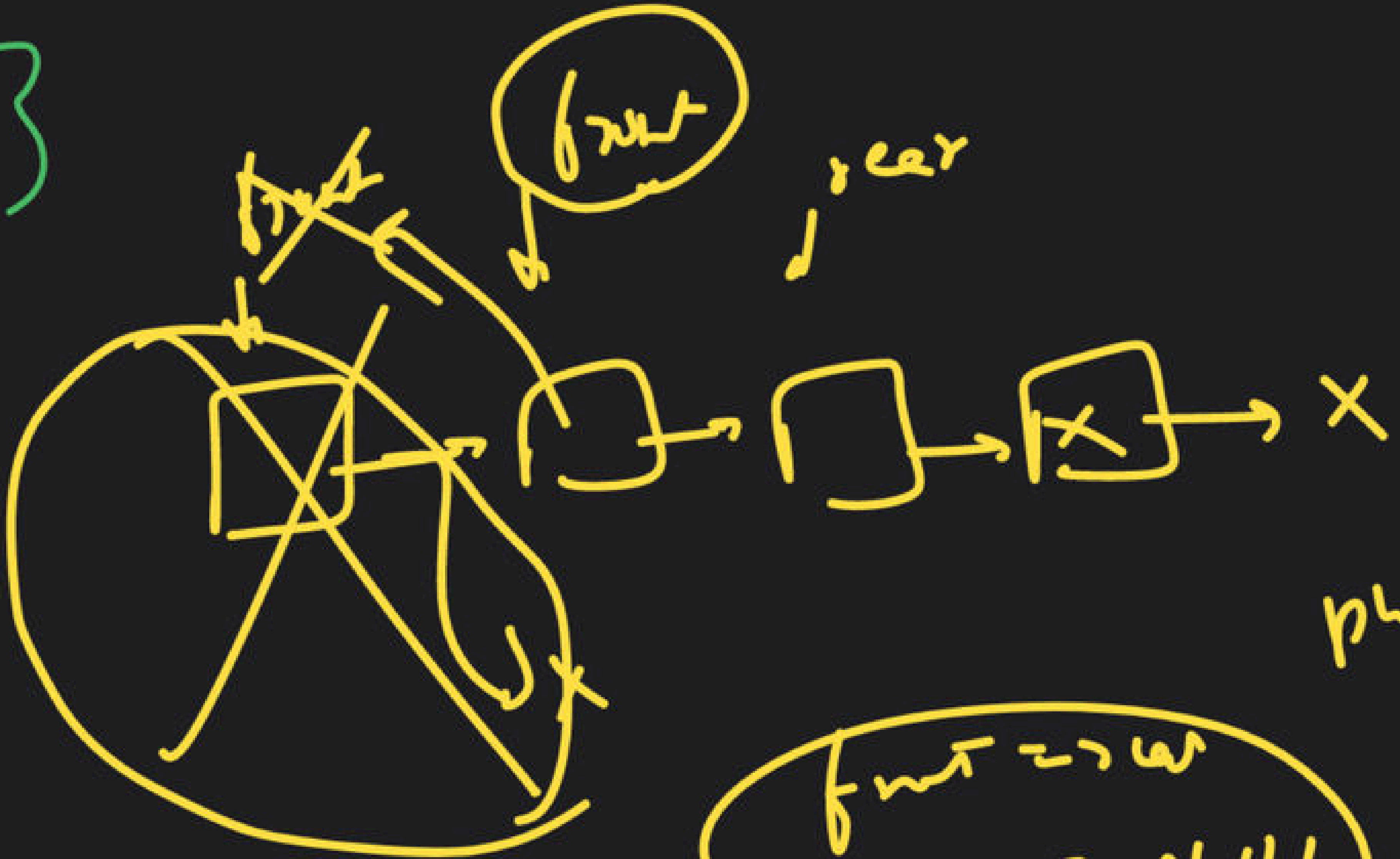
$\rightarrow$ isEmpty $\rightarrow$

```
if (front == rear)
        return 1;
else
        return 0;
```

→ getfront()
{

}

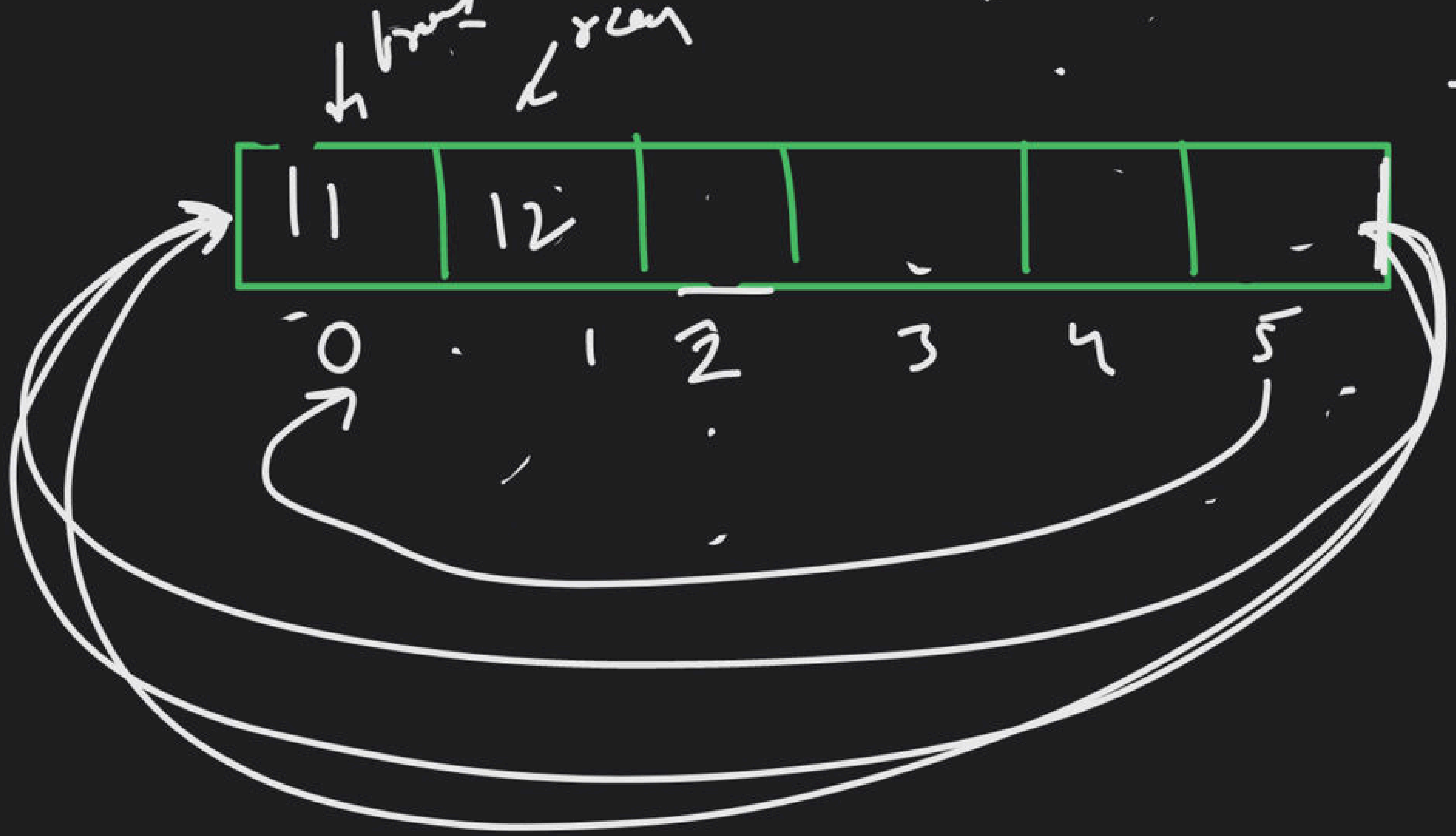
```
if (front == rear)
    return -1;
else
    return arr[front];
```



front == rear
= NULL

push → size++
pop → s--;

Circular Queue:-

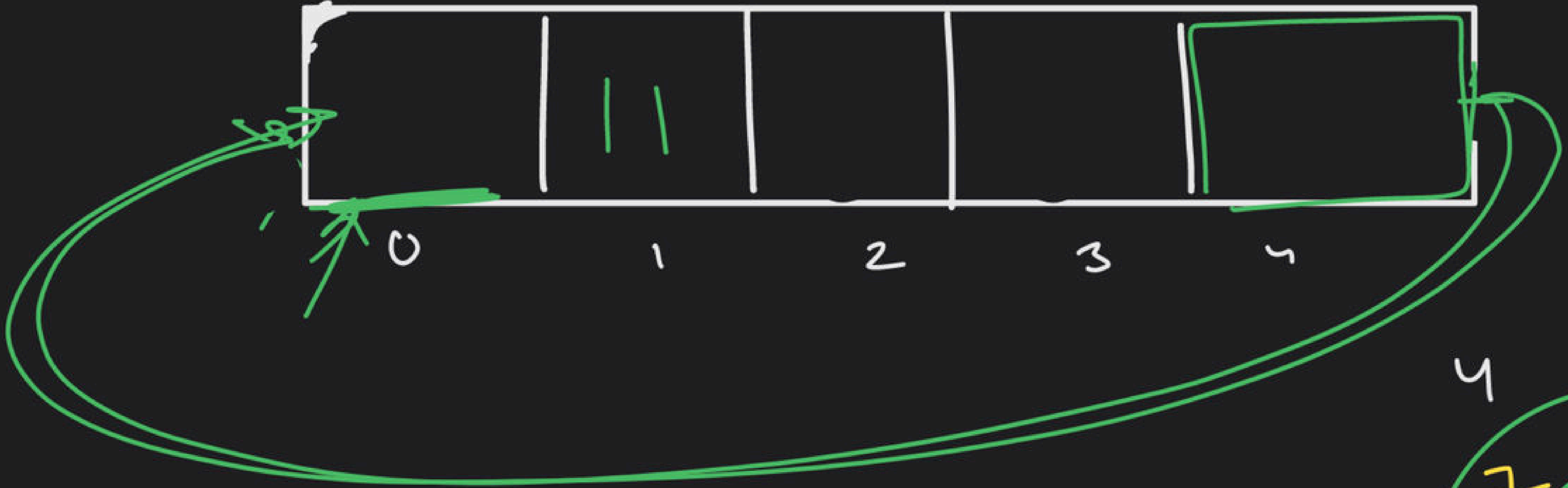


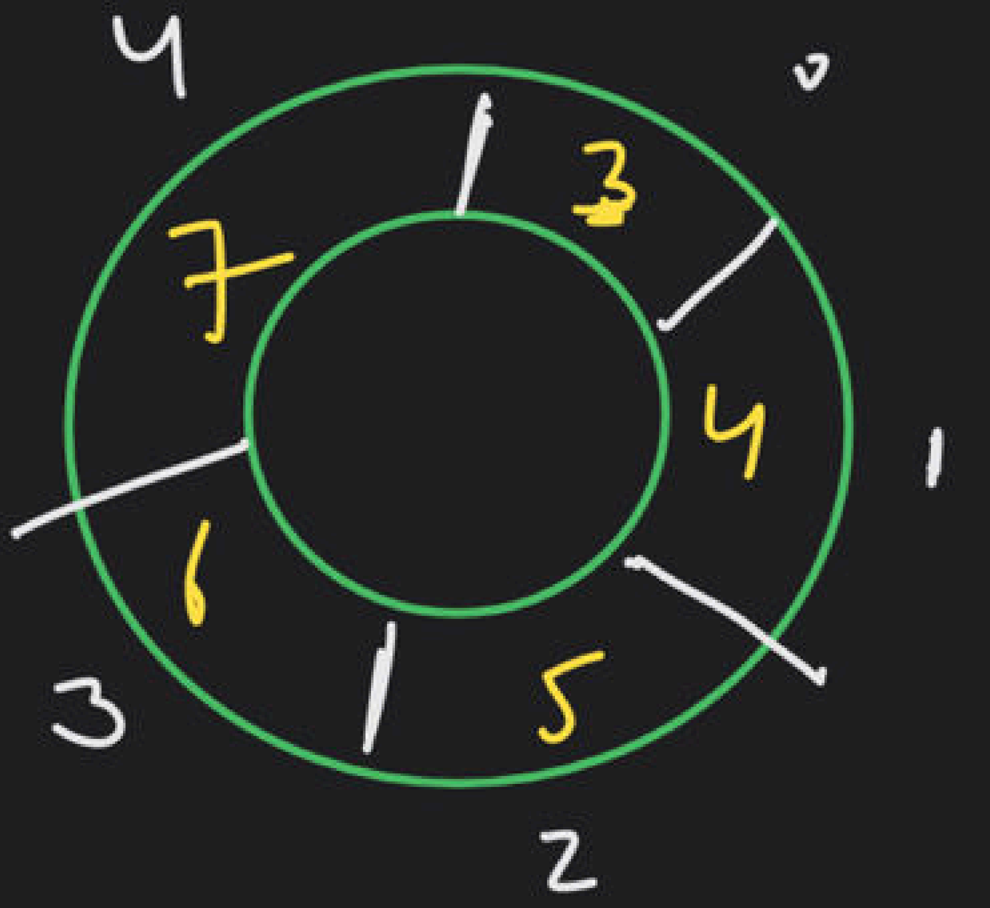
→ 10 ₹

→ Kaju roll

11

Circular queue





class CircularQueue

getRear

{

public:

getfront

arr[]

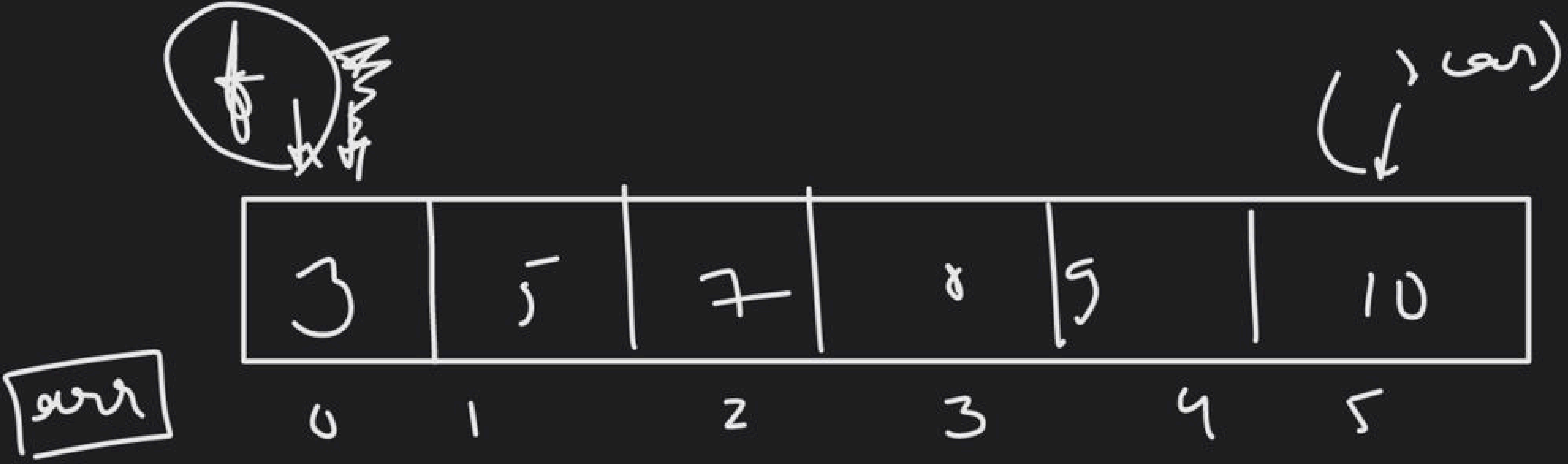
getSize()

front

rear

size

Queue() | enqueue() | dequeue() | isEmpty()



front = -1

rear = -1

q.push()

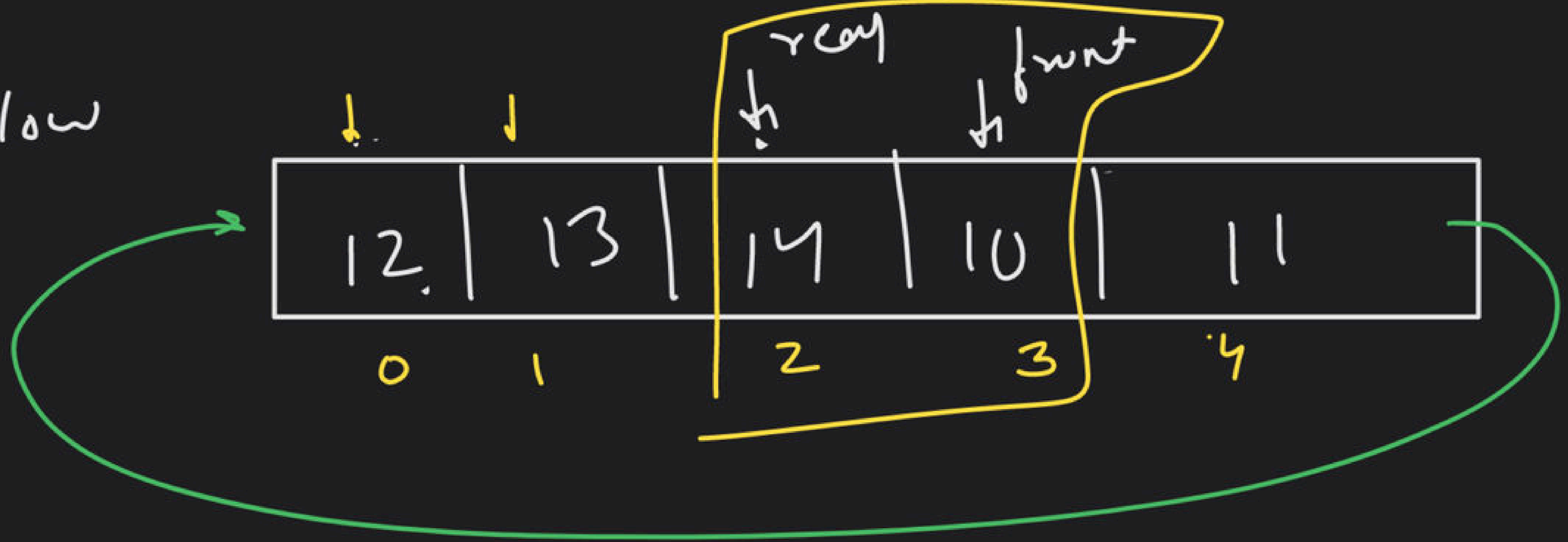
push → rear

```
if (front == -1)    // first element to push
    front = rear = 0
    arr[rear] = element;
```

overflow

```
if (front == 0 & rear == size - 1)
    cout << OF;
```

overflow



R F

~~0~~ 1 → 0 % 5 = 0

1 2 → 1 % 5 = 1

2 3 → 2 % 5 → 2

3 4 → 3 % 5 → 3

~~0 1 4~~

size = 5

rear = (front + 1) % size

front = (rear + 1) % size

1 – 0

Ist → Overflow

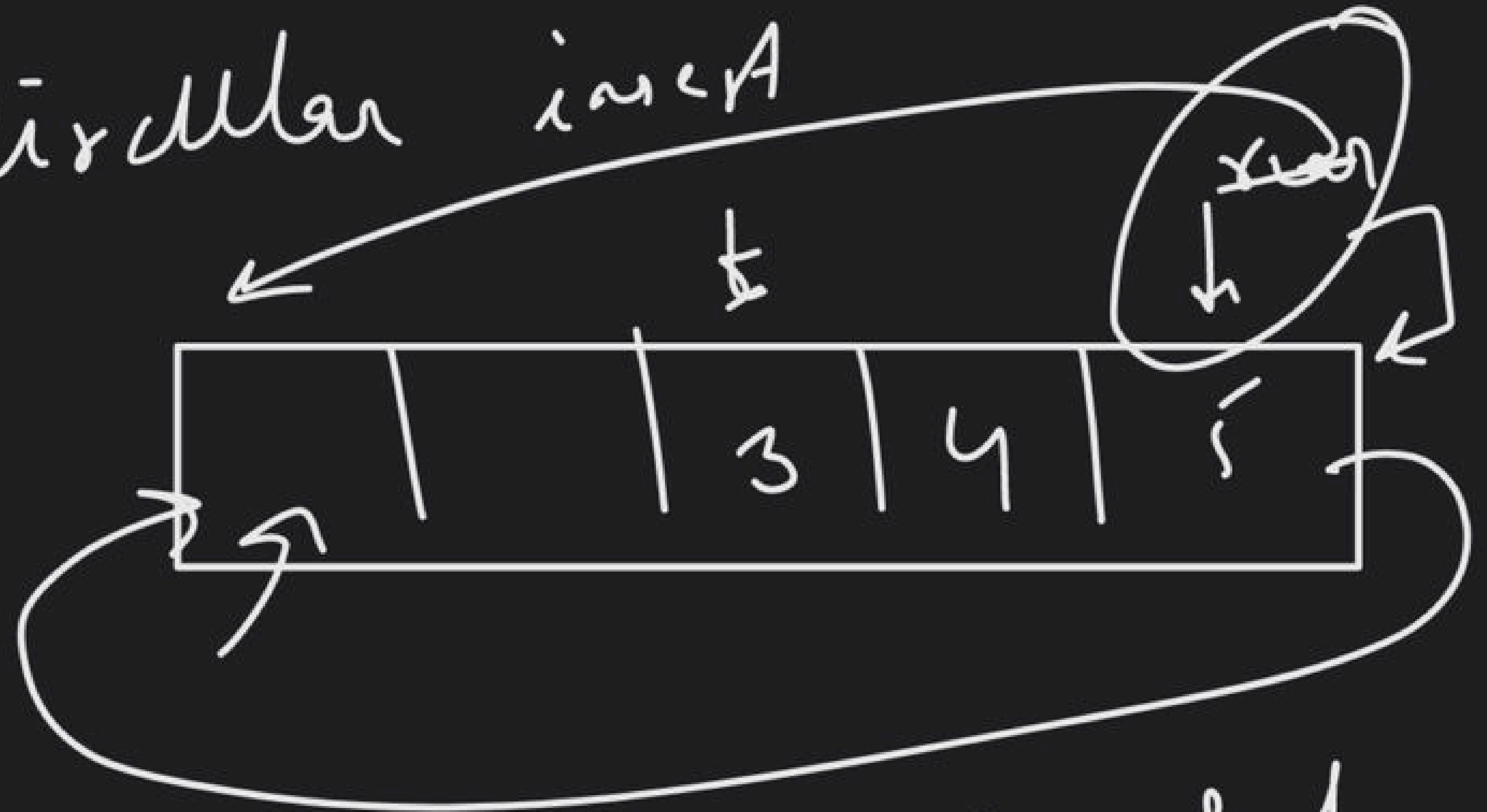
IInd → first element insert .

IIIrd → circular insert

IV → normal insert

↓

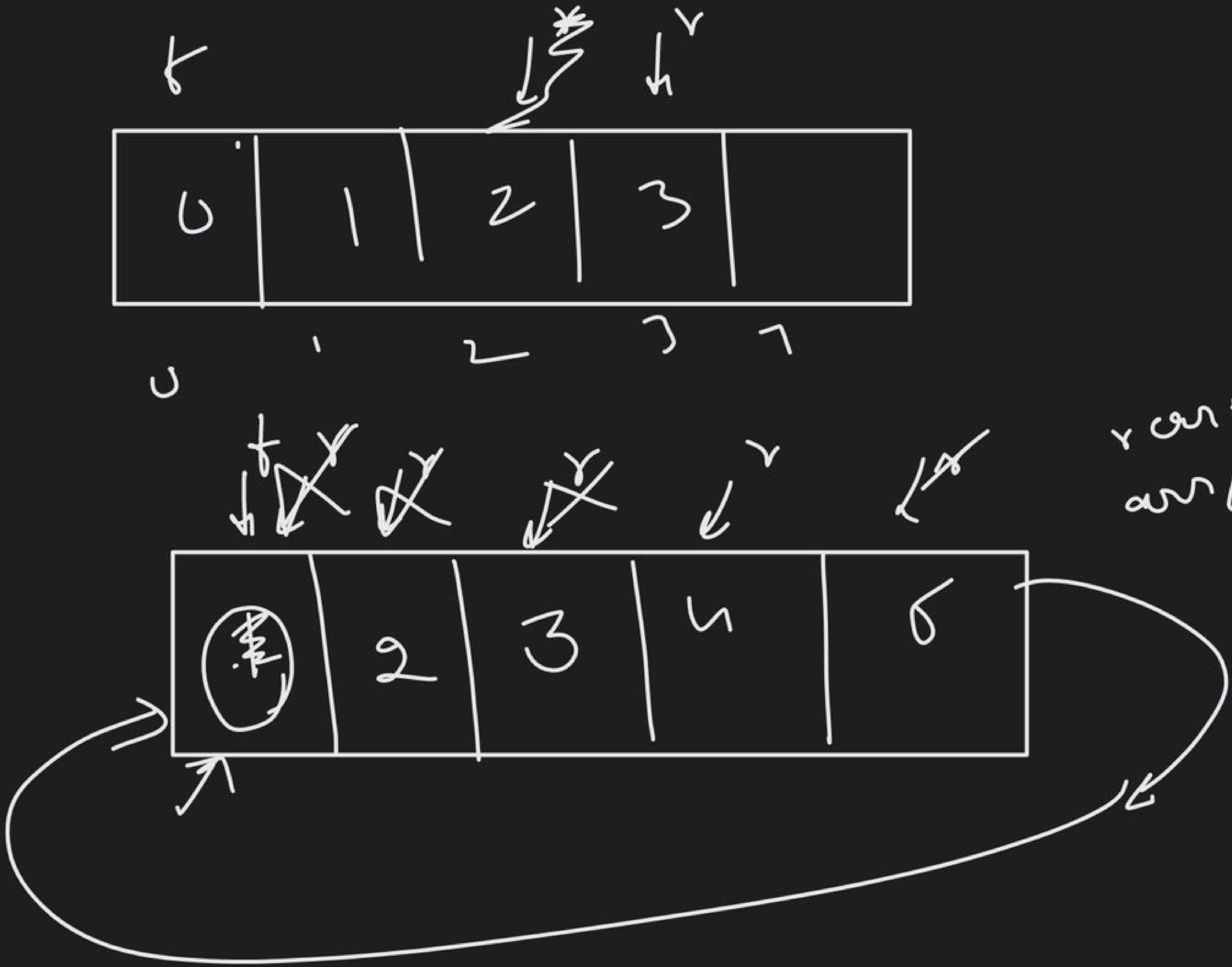
rear++



(rear = = size - 1 && front != 0)

rear = 0

arr [rear] = element

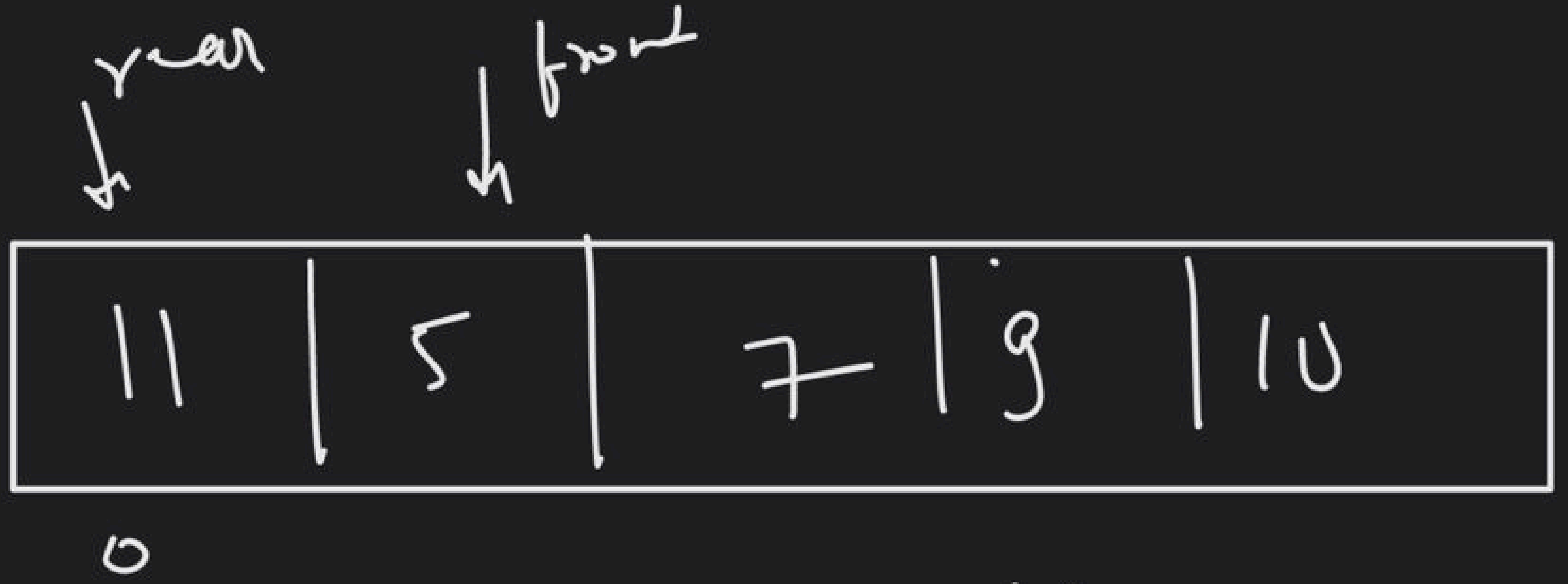


rear++
arr[rear] = element

Push :-

front / rear = $\boxed{-1}$

- → Overflow
- → single element → first element
- → circular wala game
- → normal game



$\boxed{12} \longrightarrow$ overflow

rear $\Rightarrow$ 0

(fron-1)

1-1 = 0 % 5 ... 0

Dequeue

(1) Underflow :-

```
if (front == -1)        extra
    cout << underflow
```

(2) single element

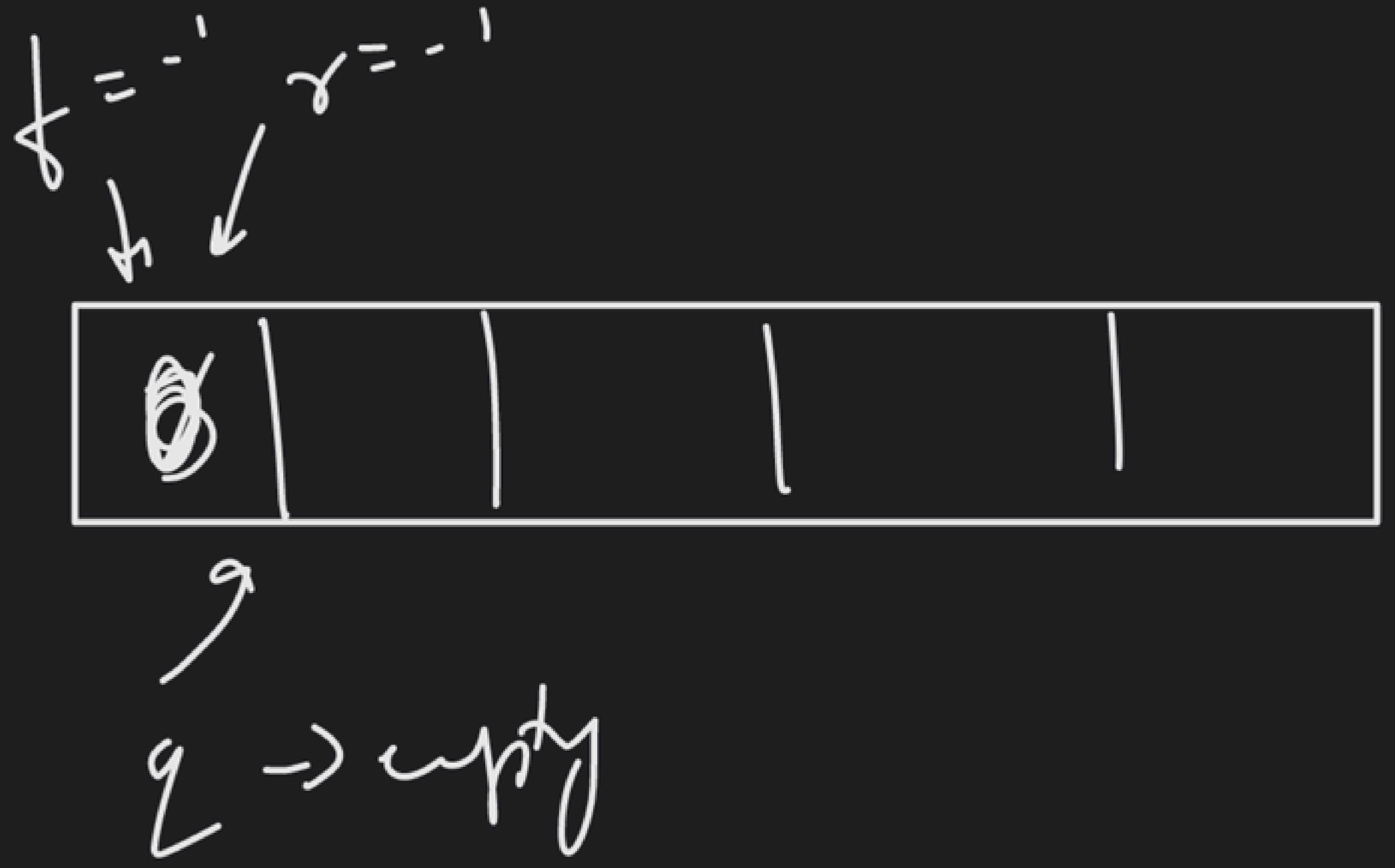
```
if (front == rear)
    front = -1
    rear = -1
```

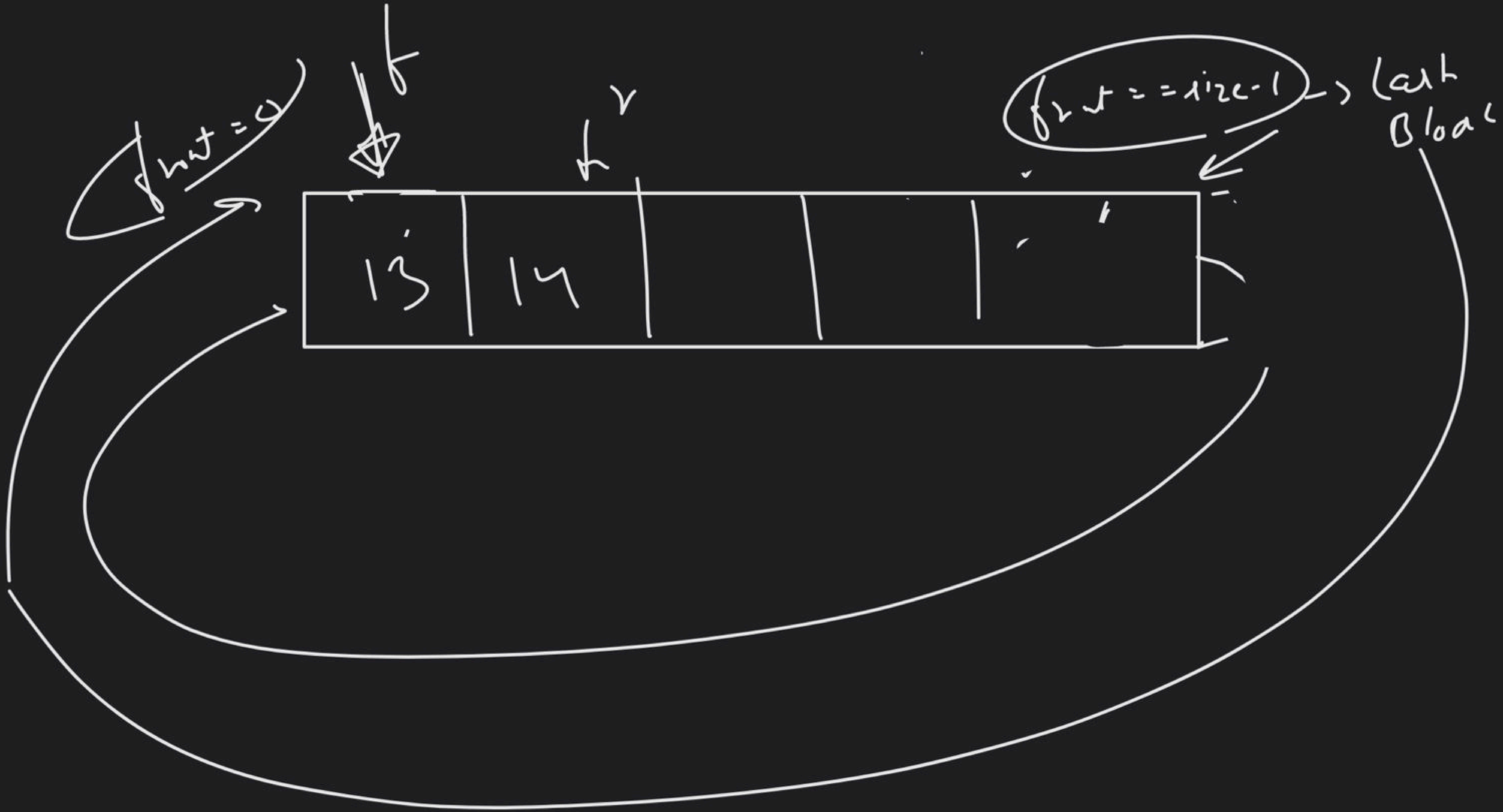
(3) cyclic nature

```
if (front == size - 1)
    front = 0;
```

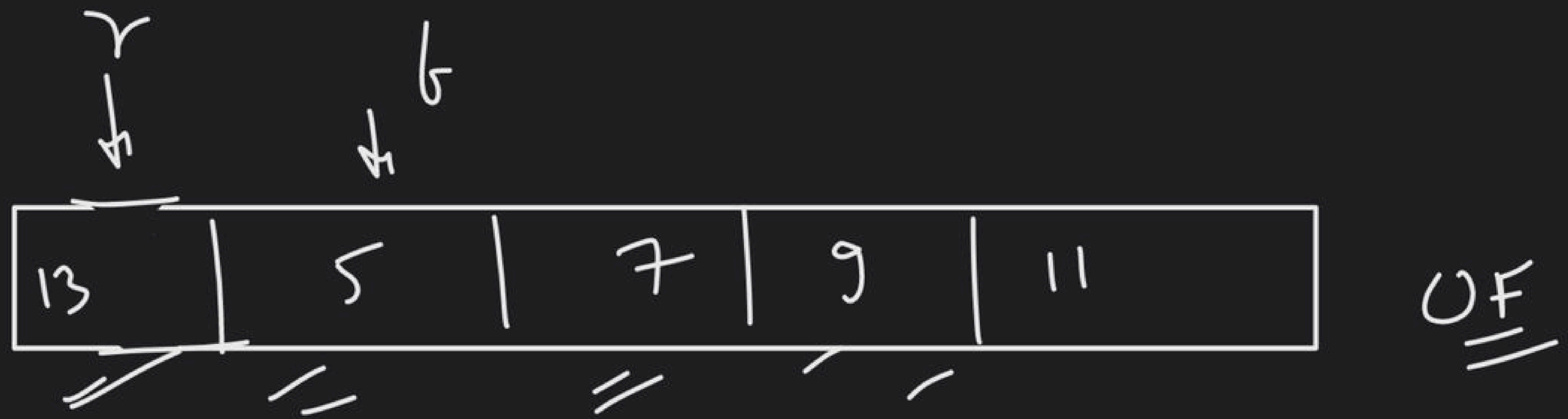
(4) normal

```
front ++
```


f = -1
r = -1
~~0~~
q
q → empty



front = 0
f
f
r
13
14
front = = size-1 → last Block

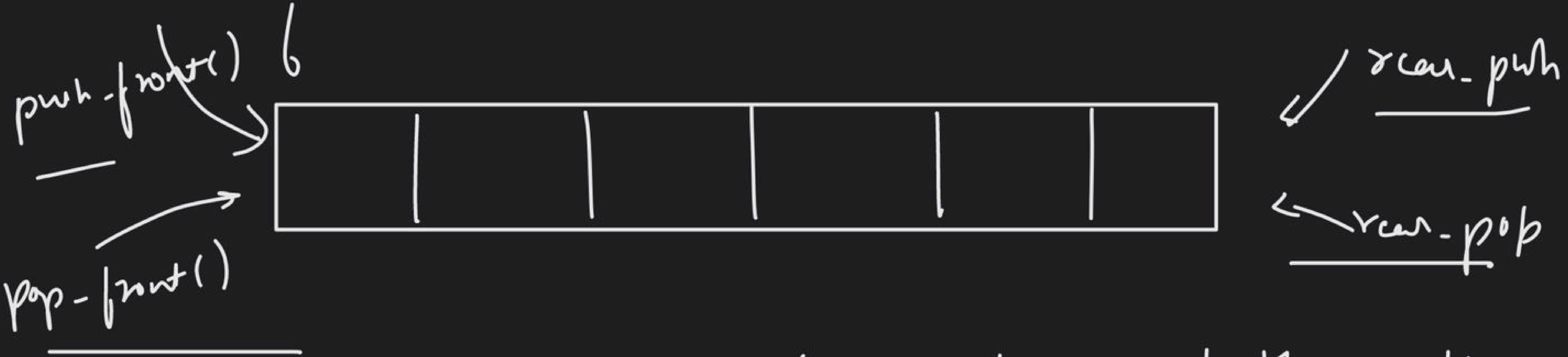


dequeue ()

enqueue (13)

enque (14) $\longrightarrow$ OF

→ Deque → [Doubly ended Queue]



→ insertion possible from both ends

→ deletion

① i/p restricted Queue → deletion → Both
                                    → insertion → 1 end

② o/p restricted Queue → deletion → 1 end
                                    → insertion → both

H/W → Scratch Implementation

↳ Deque

↳ array

1.5 hr → 2 hr