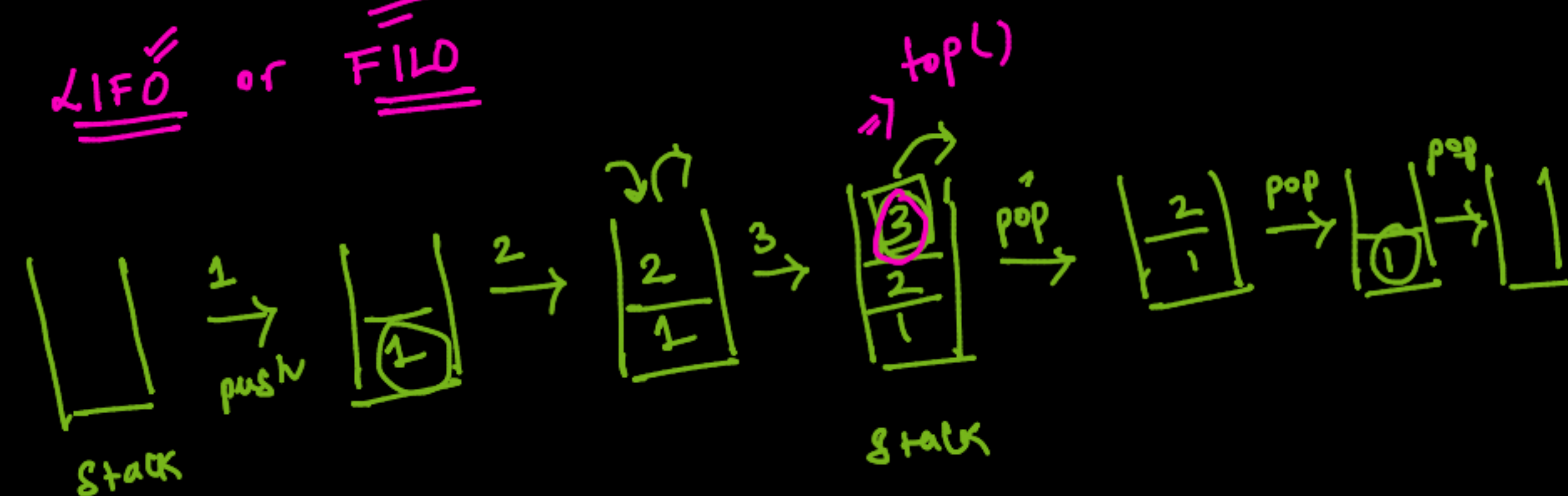


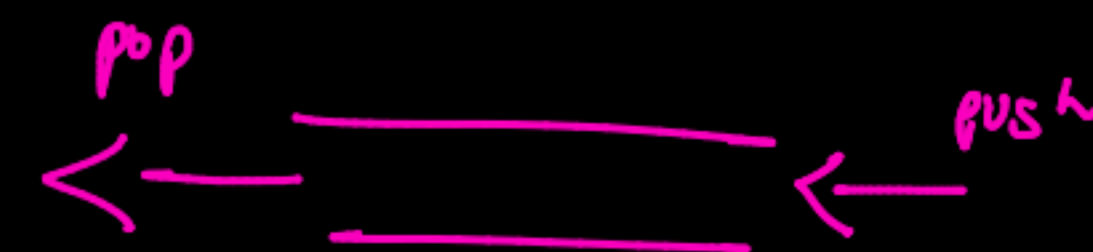
Stacks

↳ DS where the elements are inserted and extracted from the same end.

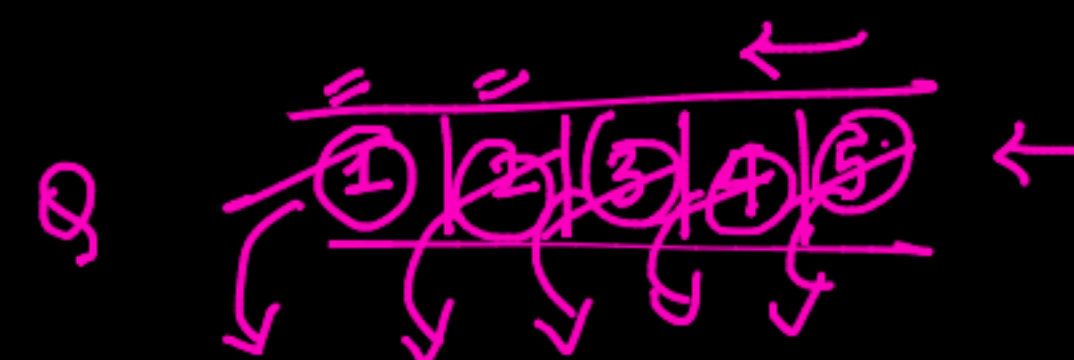
↳ LIFO or FIFO



Queue



FIFO or LIFO



Operations a stack supports

$\left\{ \begin{array}{l} \text{push}() \\ \text{pop}() \\ \text{top}() \end{array} \right.$

$\text{size}() \rightarrow \text{int}$
 $\text{empty}() \rightarrow \text{true/false}$

C++ Container Adapter

sequence containers

- vector ✓
- array ✓
- forward-list ✓
- list ✓

↳ (using vectors)

↳ using linked list

Stacks \rightarrow vectors =
 \rightarrow linked list =
 \rightarrow array \rightarrow fixed

Stack Overflow

arr

10	20	30	40	50
----	----	----	----	----

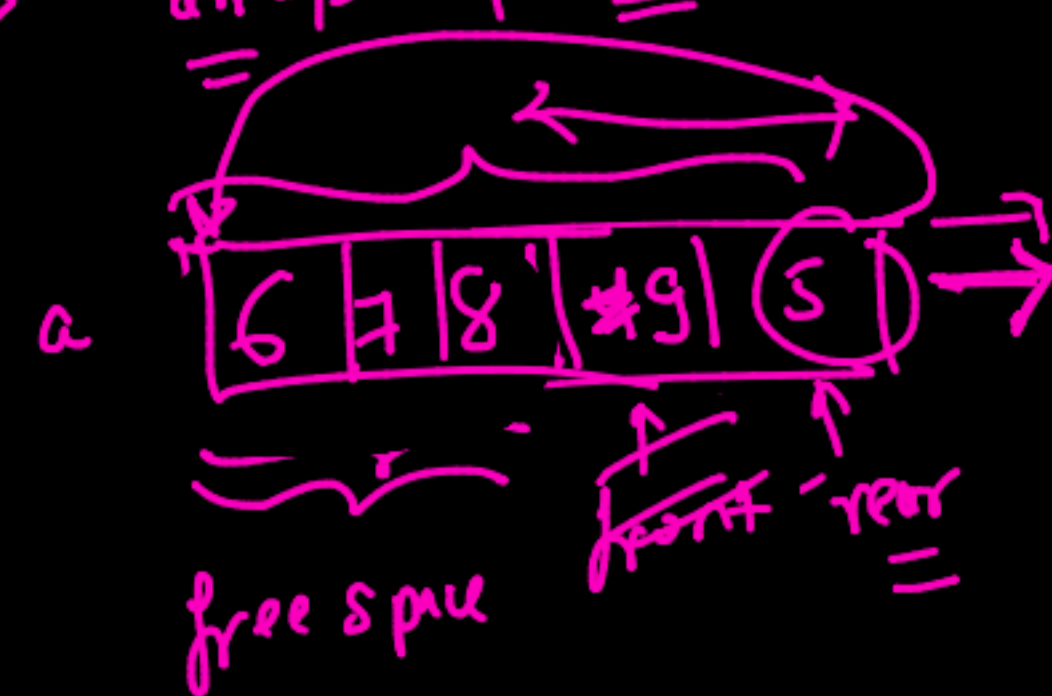
top \rightarrow fixed size

top = -1

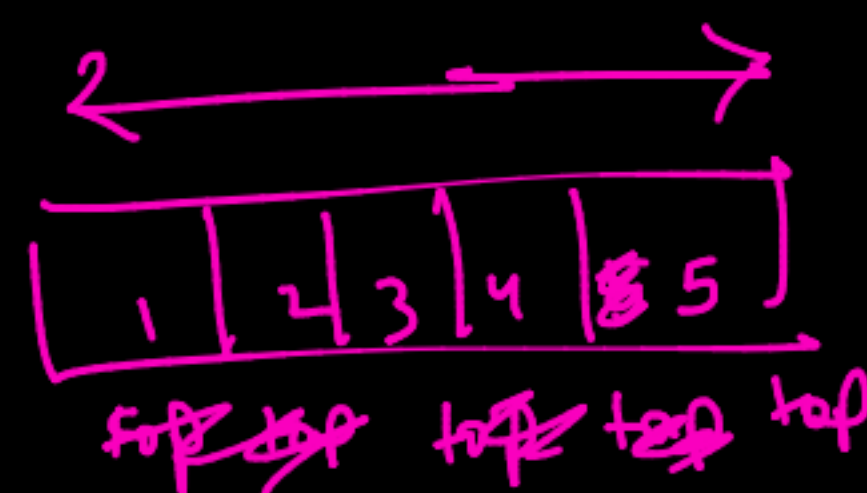
arr[top++] = data top()

pop

= Queues → Linked list (dynamic)
→ arrays (fixed size)



$$a[\text{rear} + 1] = \text{data}_i$$



$$\text{top} = -1$$

top++

$$CQ \Rightarrow \textcircled{8}$$

len = 0

capacity = 9

20

$$(8+1)^{-19} = 0$$
$$= 0$$

Diagram illustrating a queue implemented as an array. The array has 9 slots, indexed 0 to 8. The values are 10, 20, 30, 40, 50, 60, 20, 50, and an empty slot. A 'front' pointer points to index 0, and a 'rear' pointer points to index 8. The value 20 is written in both index 6 and index 8.

↑
front = 0
rear = 0

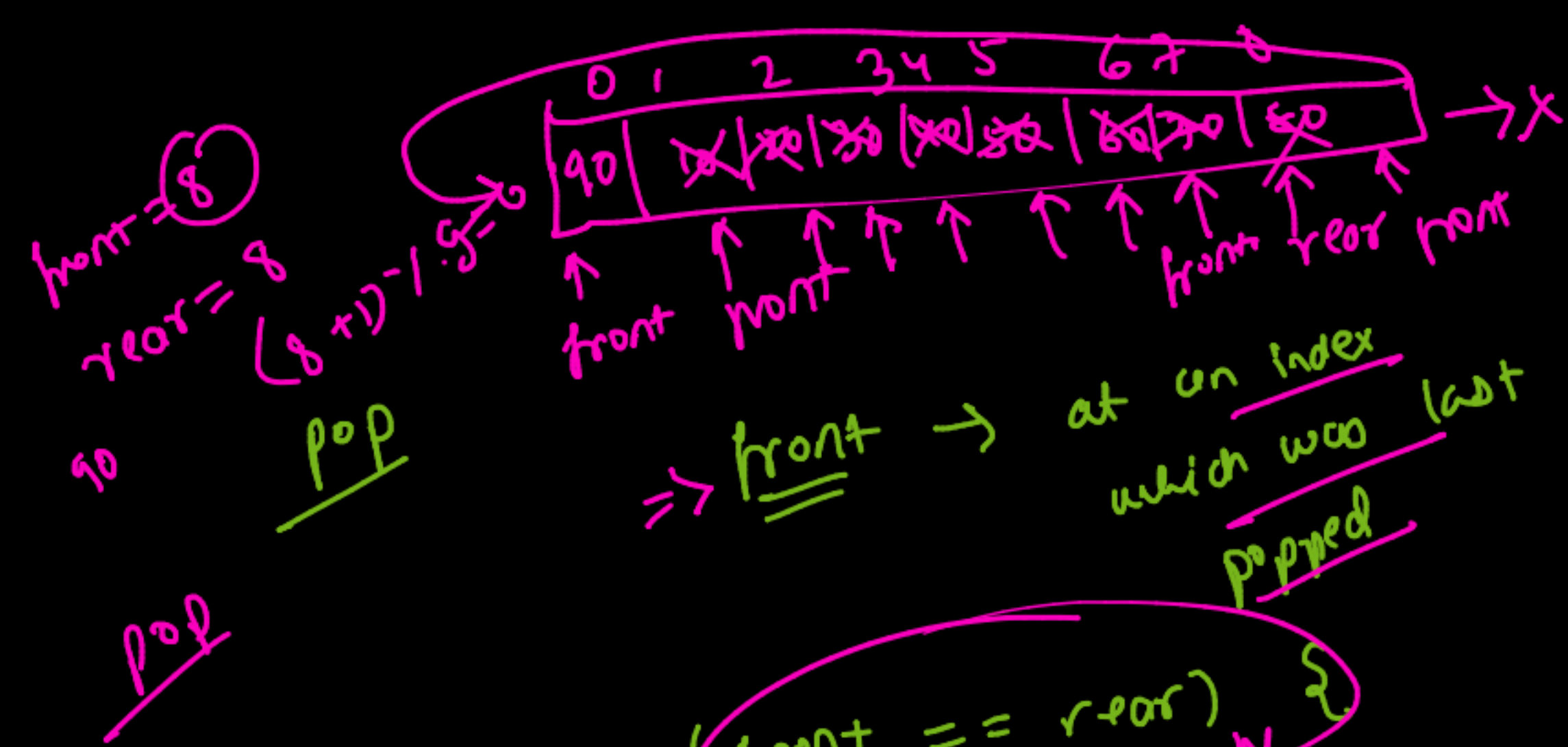
push → insertion into the queue

```

✓ ✓ x if (front == (rear + 1) % c) {
    // que is full
    else {
        rear = (rear + 1) % c;
        arr[rear] = data;
    }
}

```

- ⇒ insert at rear
- ⇒ delete from front

$$\begin{aligned} &= \text{front} = 0 \\ &= \text{rear} = 1 \cdot 9 = 1 \end{aligned}$$


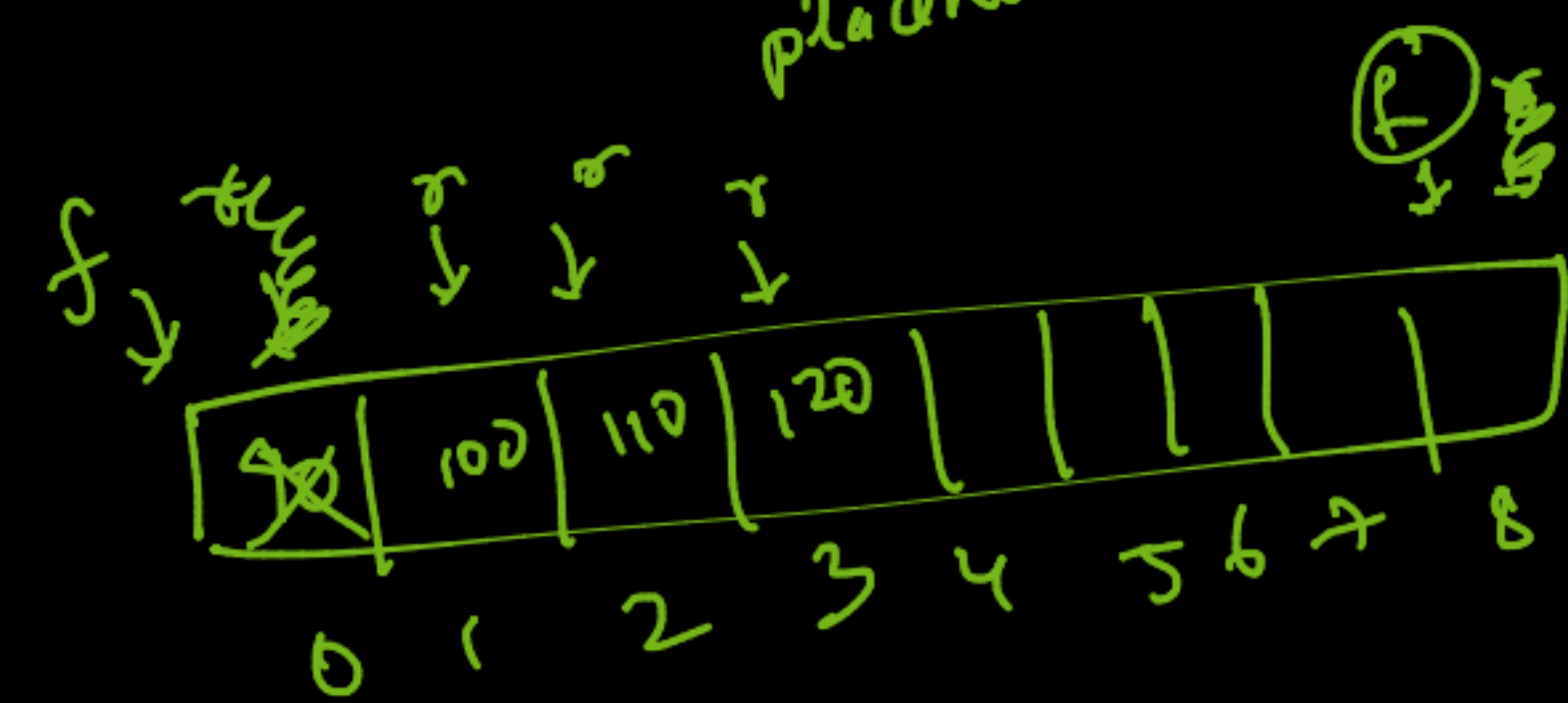
```

if (front == rear) {
    // queue is empty
} else {
    front = (front + 1) % C
}

```

$\frac{C+1}{\uparrow}$

 $\frac{r}{(b+1)-1.9}$



$$\frac{f}{r} = \frac{(r+1) \cdot C}{r} \cdot X$$