Doubt Class by Lakshay Bhaiya

Special class

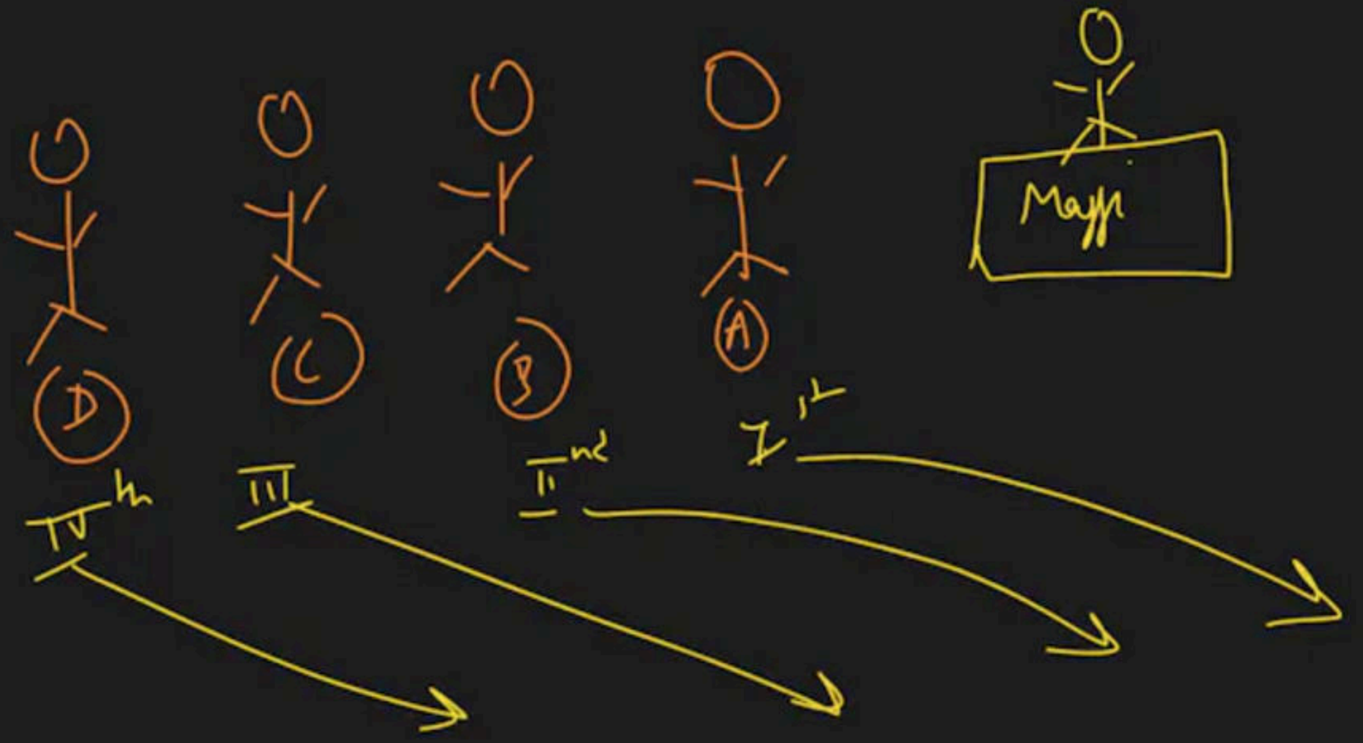
Love Babbar • Nov 26, 2023

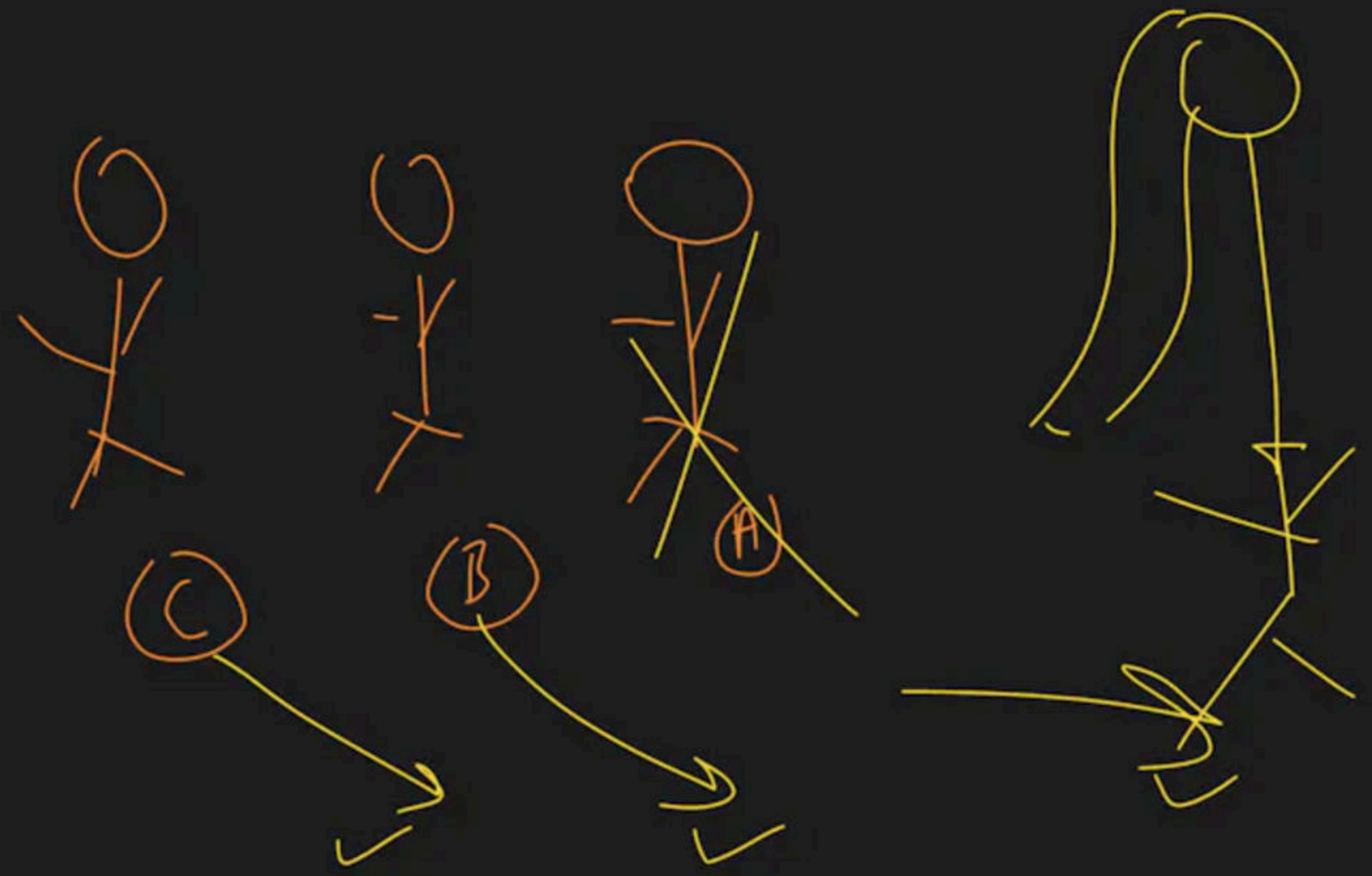
Queue Class - 1 [Join Here]

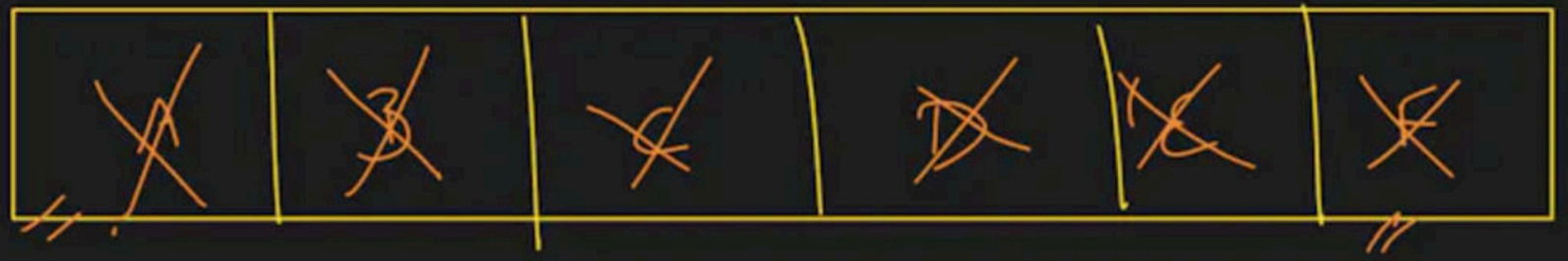
Special class

Love Babbar • Nov 21, 2023

→ Queue: D.S ⟶ FIFO (first in first out)



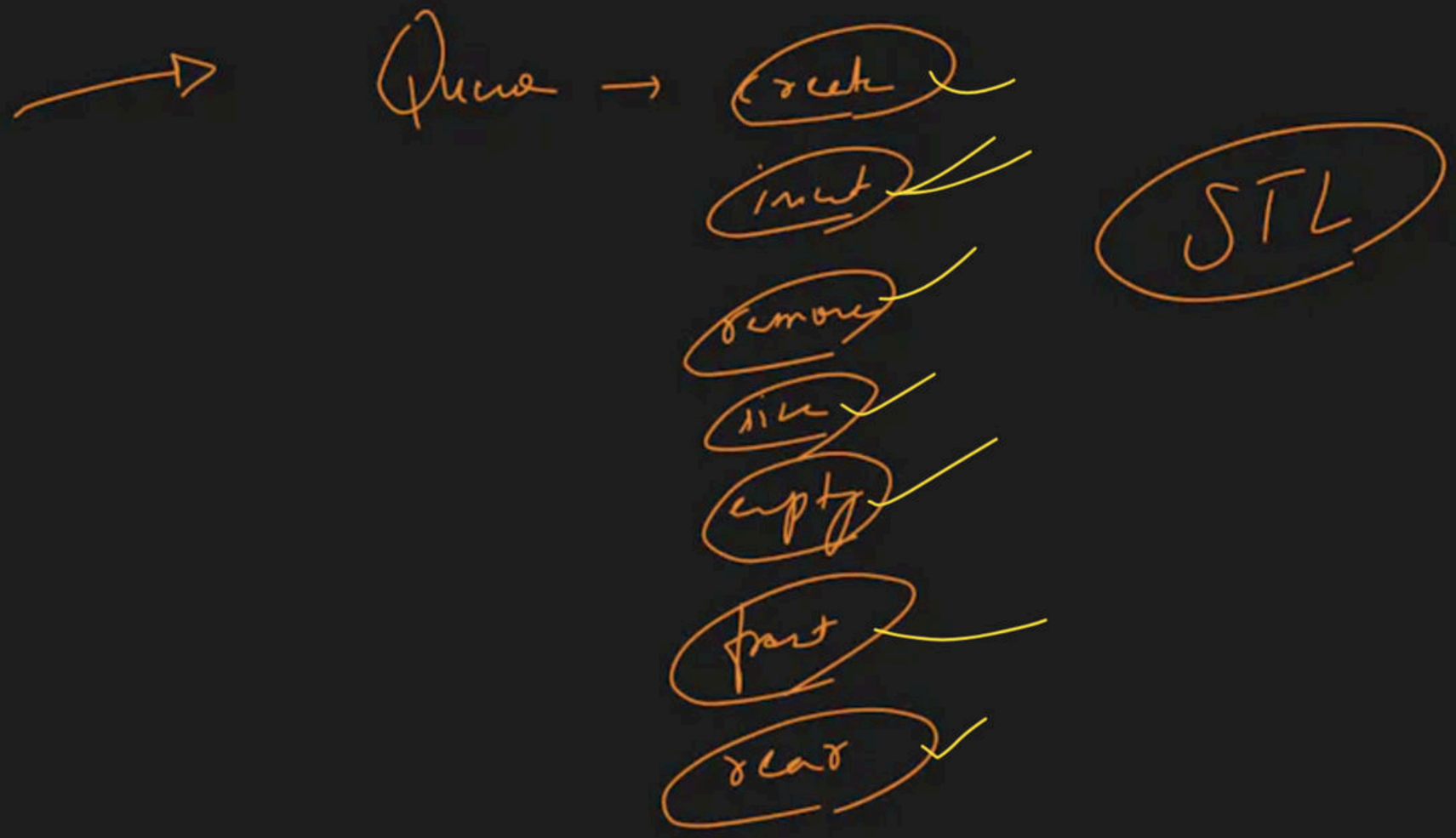

C
B
A

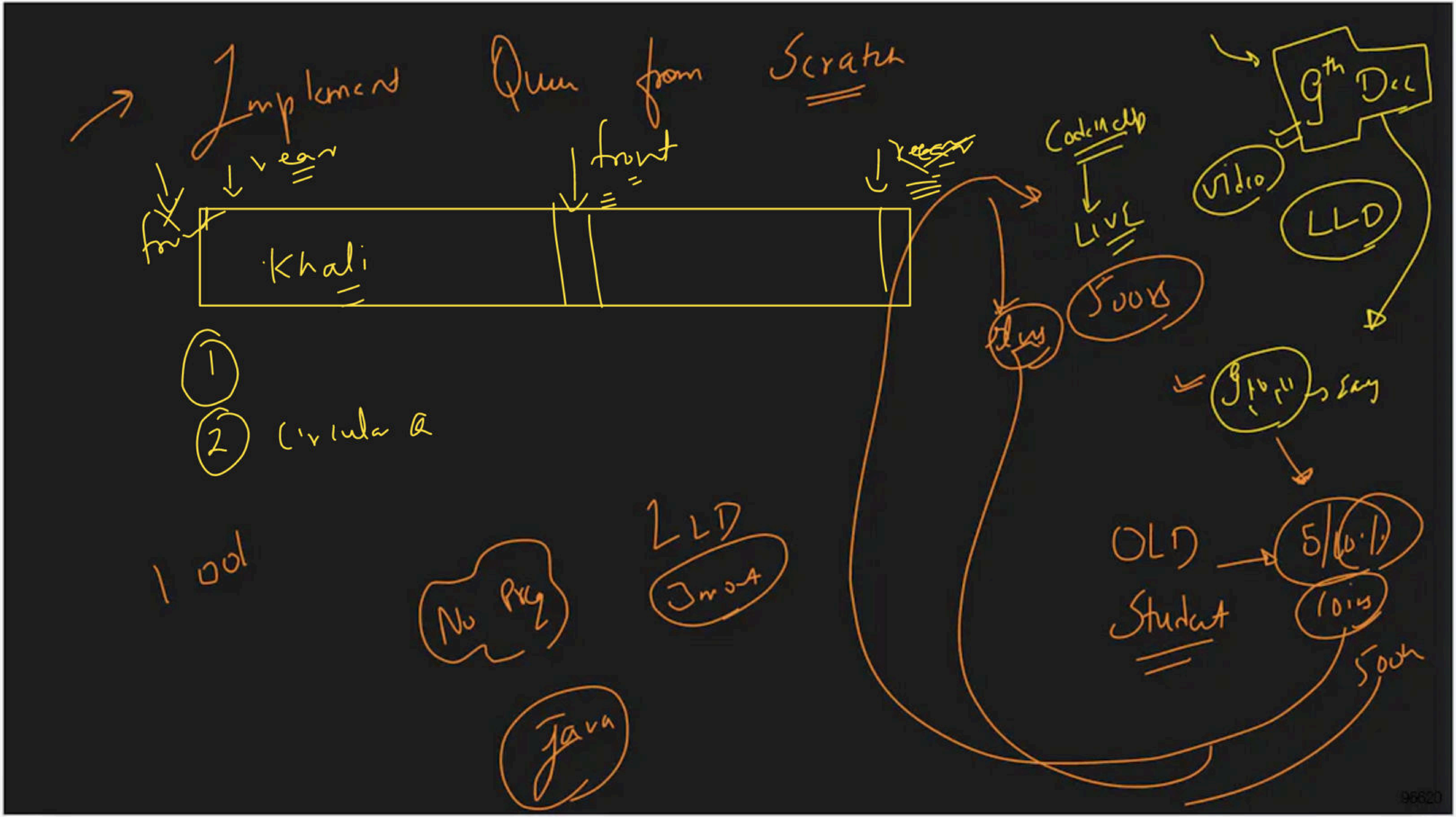


Queue

→ Stack → LIFO

→ Queue → FIFO


Queue
STL
Queue
Interview
Coding Test
STL



Implement Queue from Scratch
rear
front
front
Khali
1
2 Circular Q
LLD
No Prereq
Java
LIVE
9th Dec
Video
LLD
Easy
OLD Student

→ Implement Queue from Scratch

arr / LL

Queue → front / back

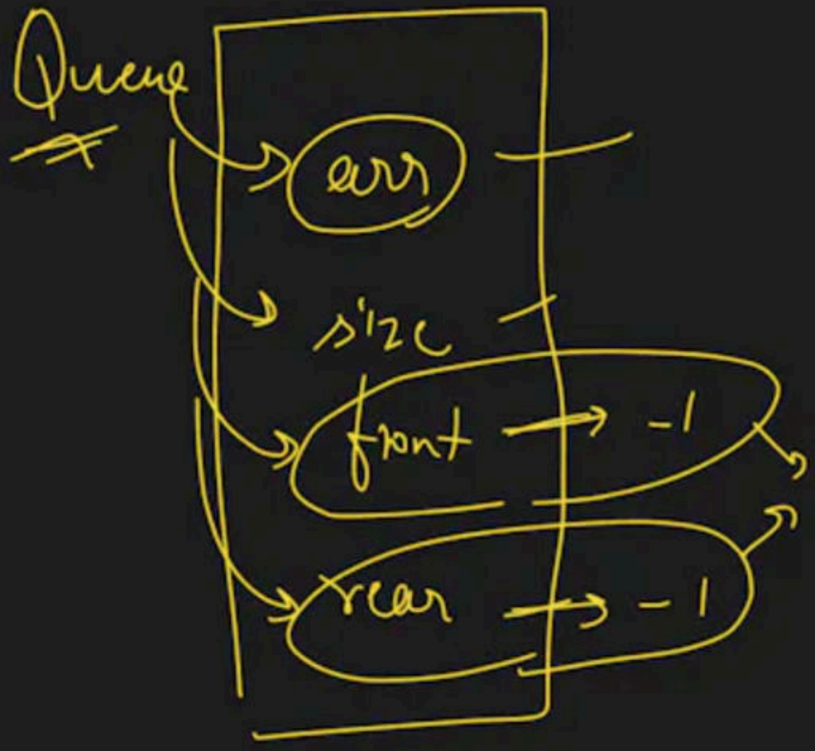
front rear

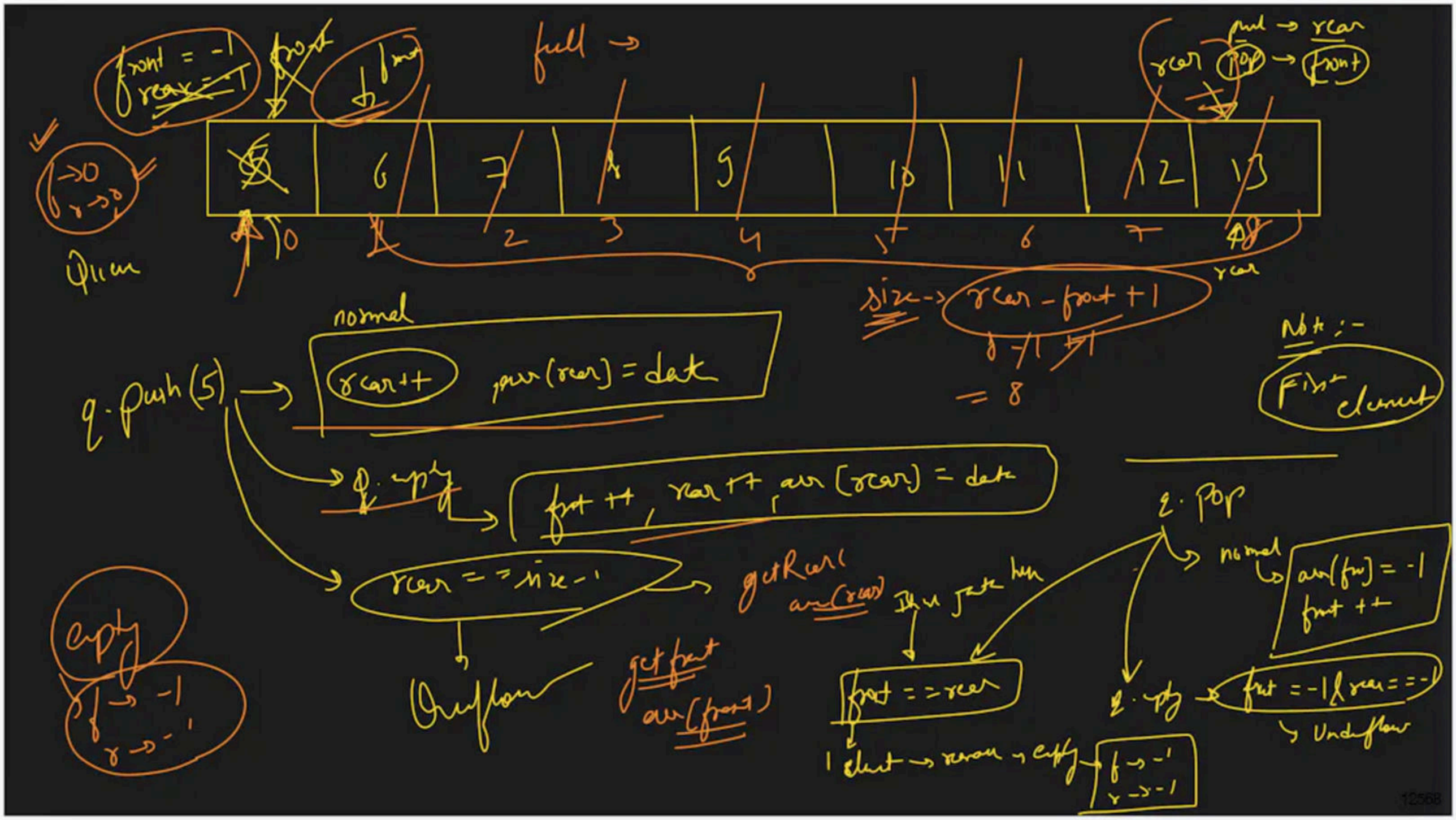
Stack

top

-1

Queue




front = -1
rear = -1
front
front
full →
rear
rear → rear
pop → front
0 1 2 3 4 5 6 7
Queue
normal
rear++
arr[rear] = data
q.push(5)
size → rear - front + 1
= 8
Note :-
First element
q.empty
front ++, rear++, arr[rear] = data
rear == size-1
getRear(arr[rear])
Overflow
getfront arr[front]
front == rear
1 elem → rear = front = -1
q.pop
normal
arr[fr] = -1
front ++
q.empty
front == -1 && rear == -1
Underflow
empty
f → -1
r → -1

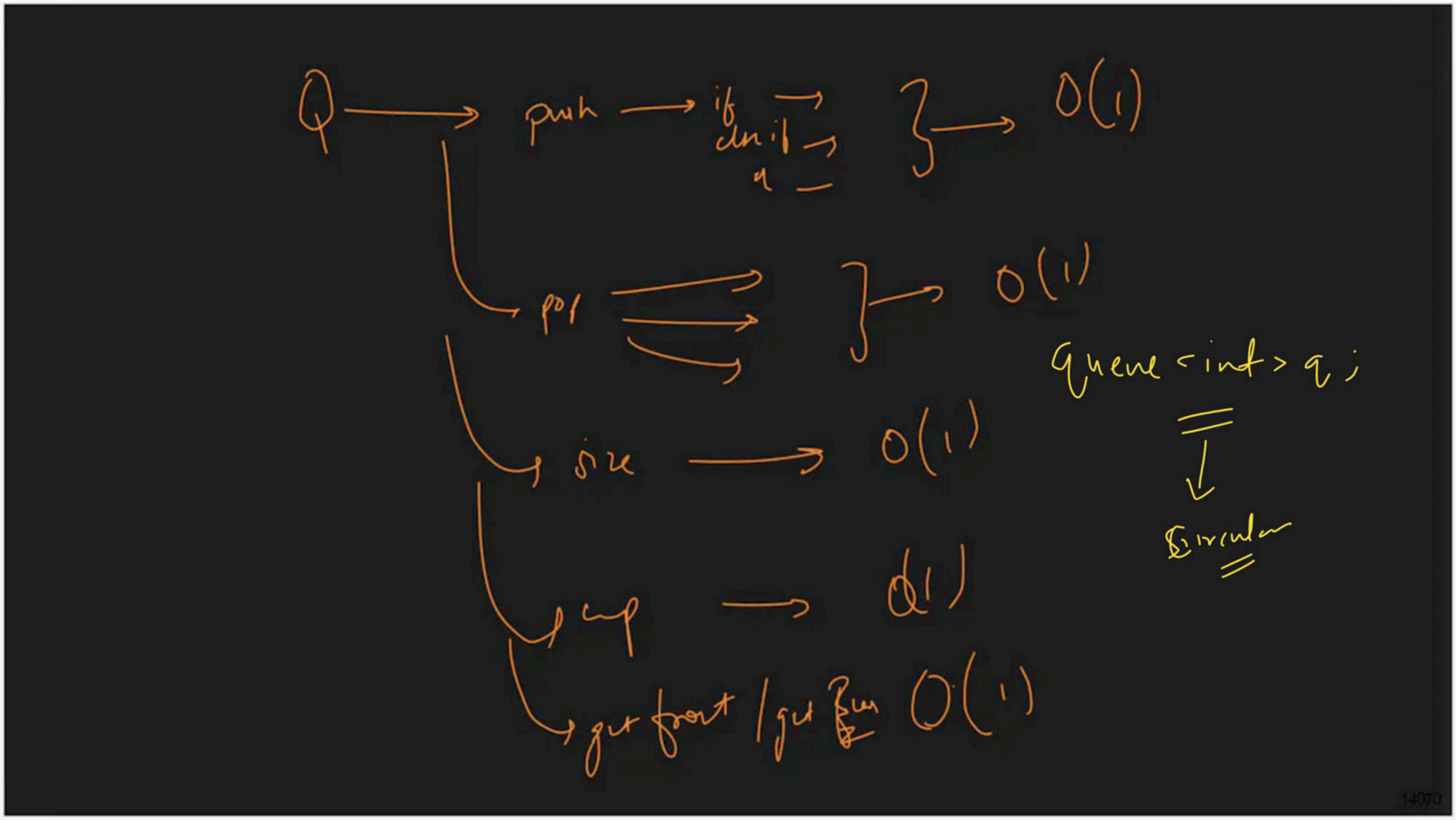
q.push(10)

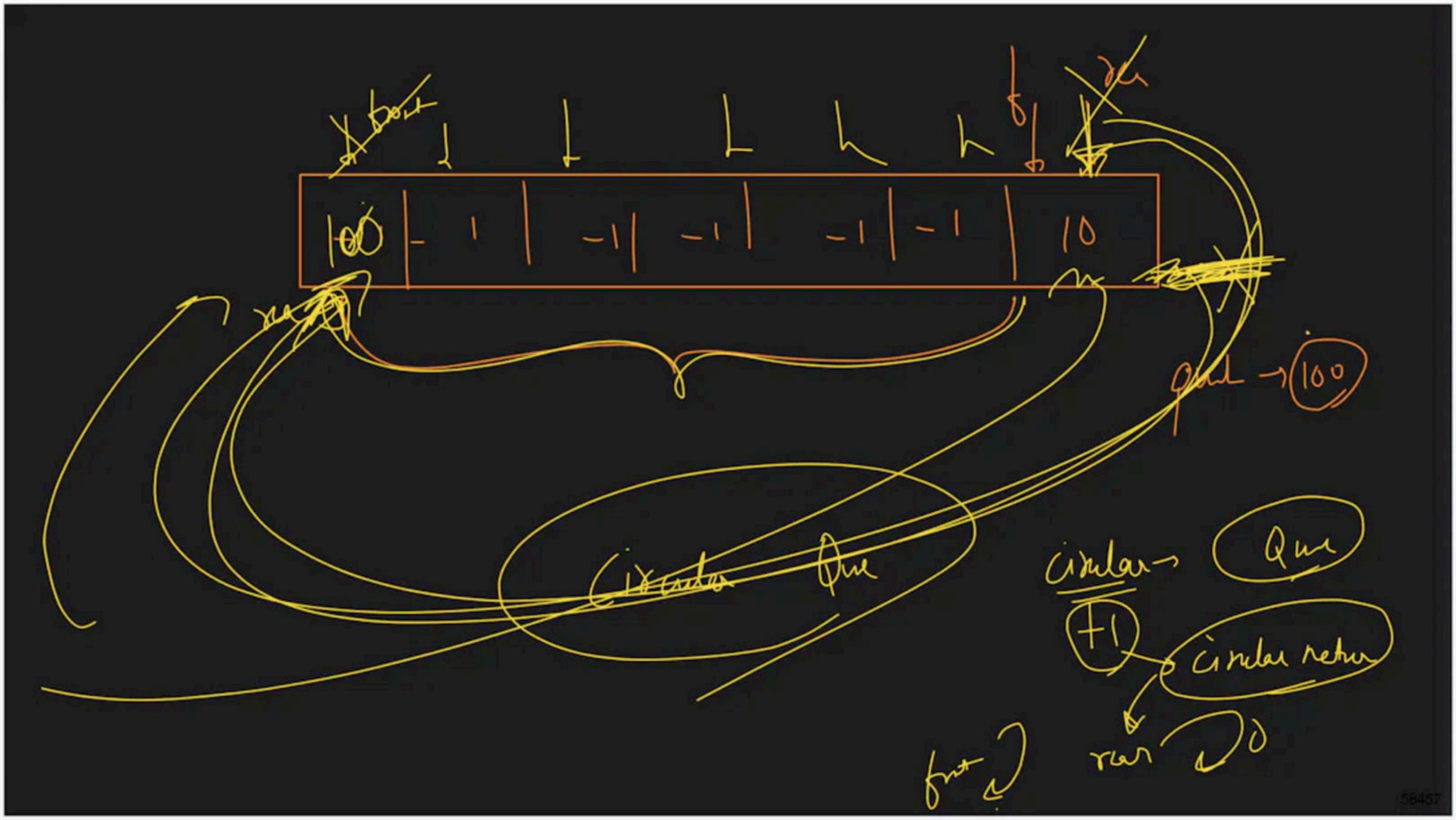
q.pop()

f = r = 0 → single elmt

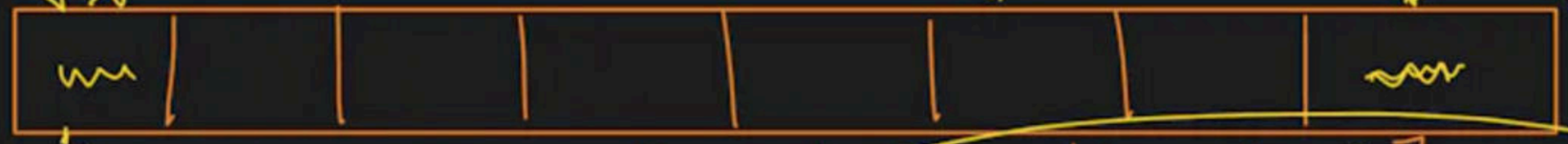
f = r → removed → q.empty()

f → -1
r → -1


Q
push
if limit
O(1)
pop
O(1)
size
O(1)
empty
O(1)
get front / get Rear O(1)
Queue<int> q;
Circular


100
-1
-1
-1
-1
10
100
Queue
Circular
+1
Queue
Circular return
front
rear
58457

Circular Queue



full → front = 0 && rear = size - 1

empty → front = rear = -1

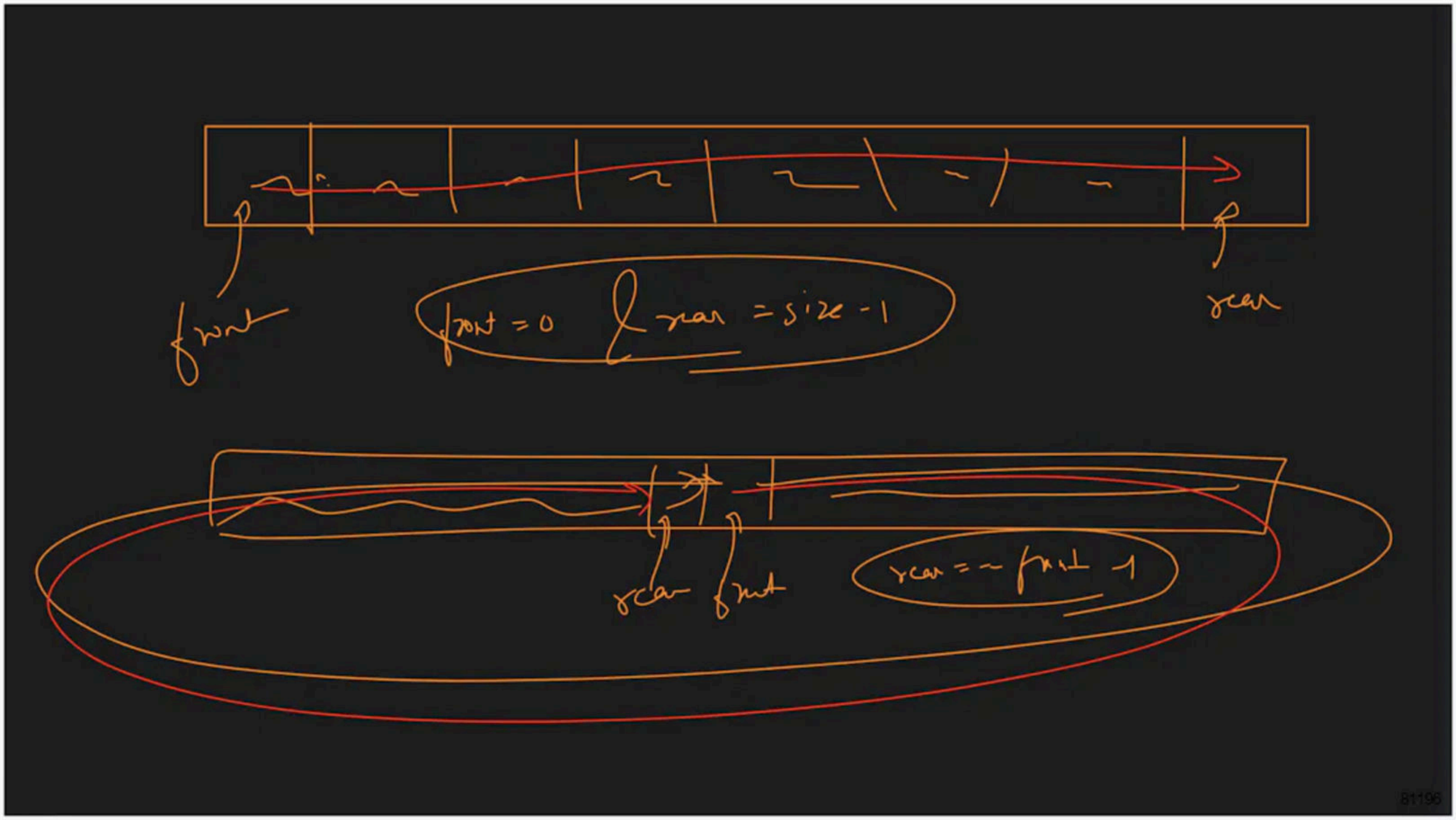
Return → front == rear

push

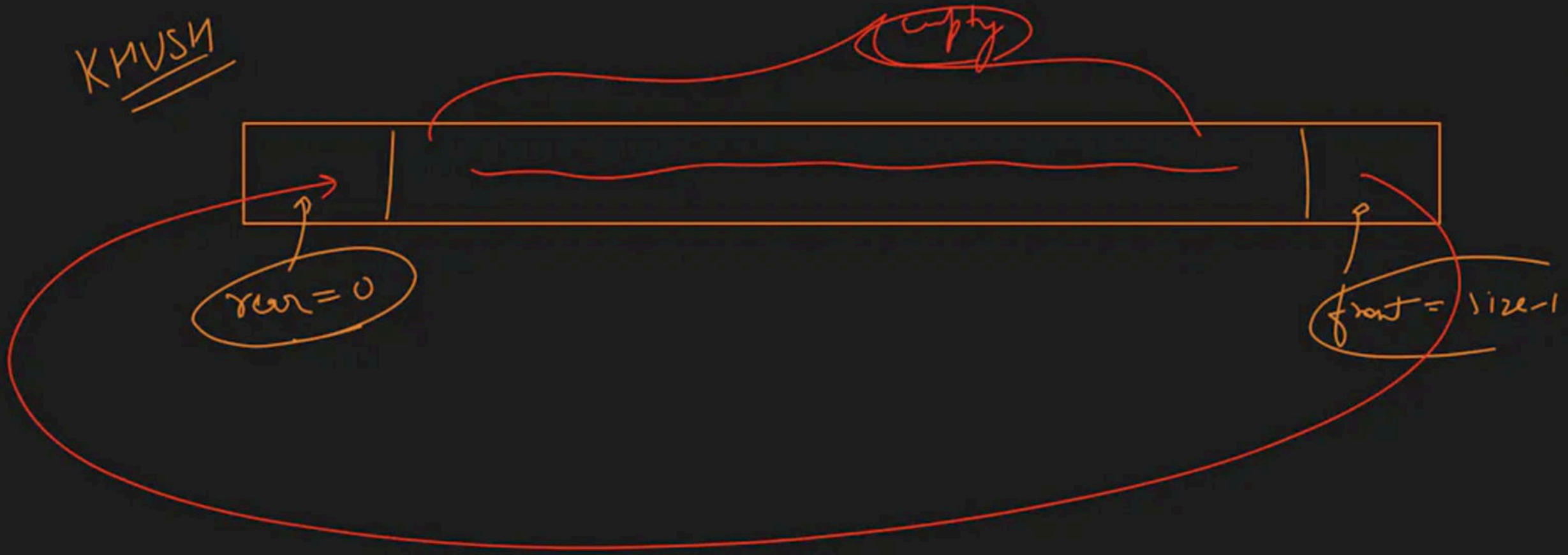
- Overflow → front = 0 && rear = size - 1
- Empty case → f = -1 && rear = -1 → front = rear = 0 → arr[rear] = val
- Circular return → rear == size - 1 && front != 0 → rear = 0 → arr[rear] = val
- normal flow → rear++ → arr[rear] = val

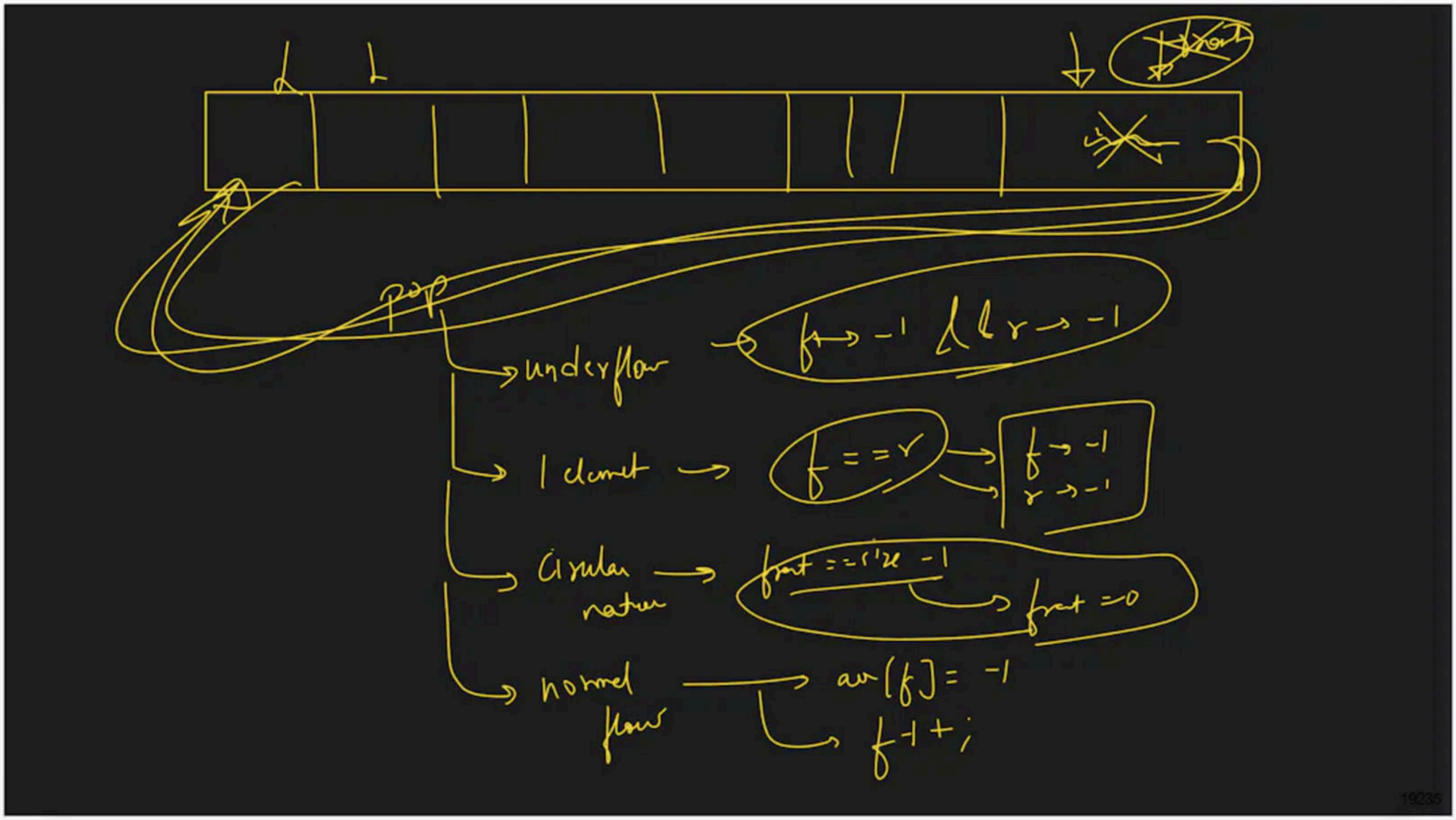
arr
size
front →
rear →

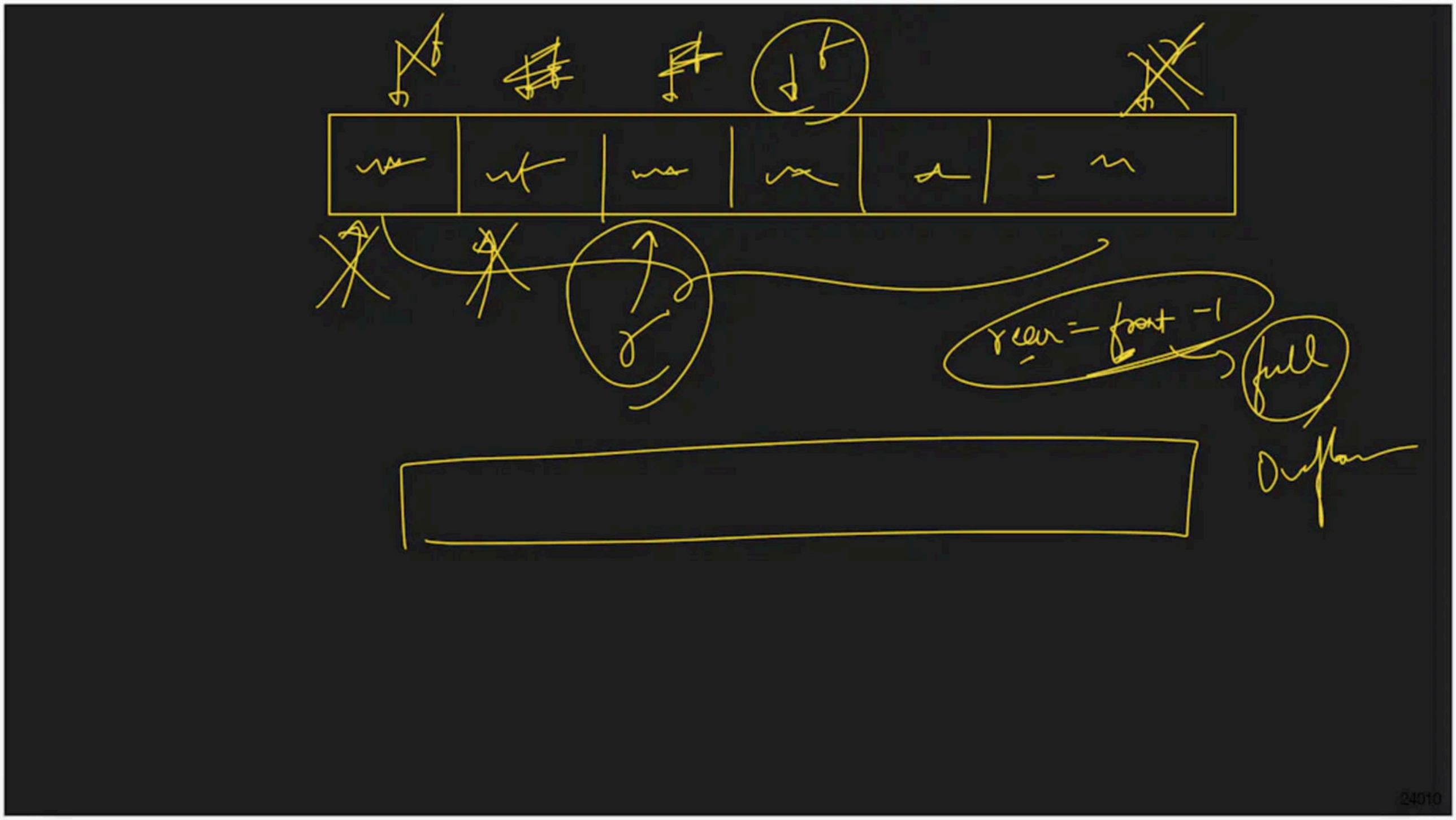
54321

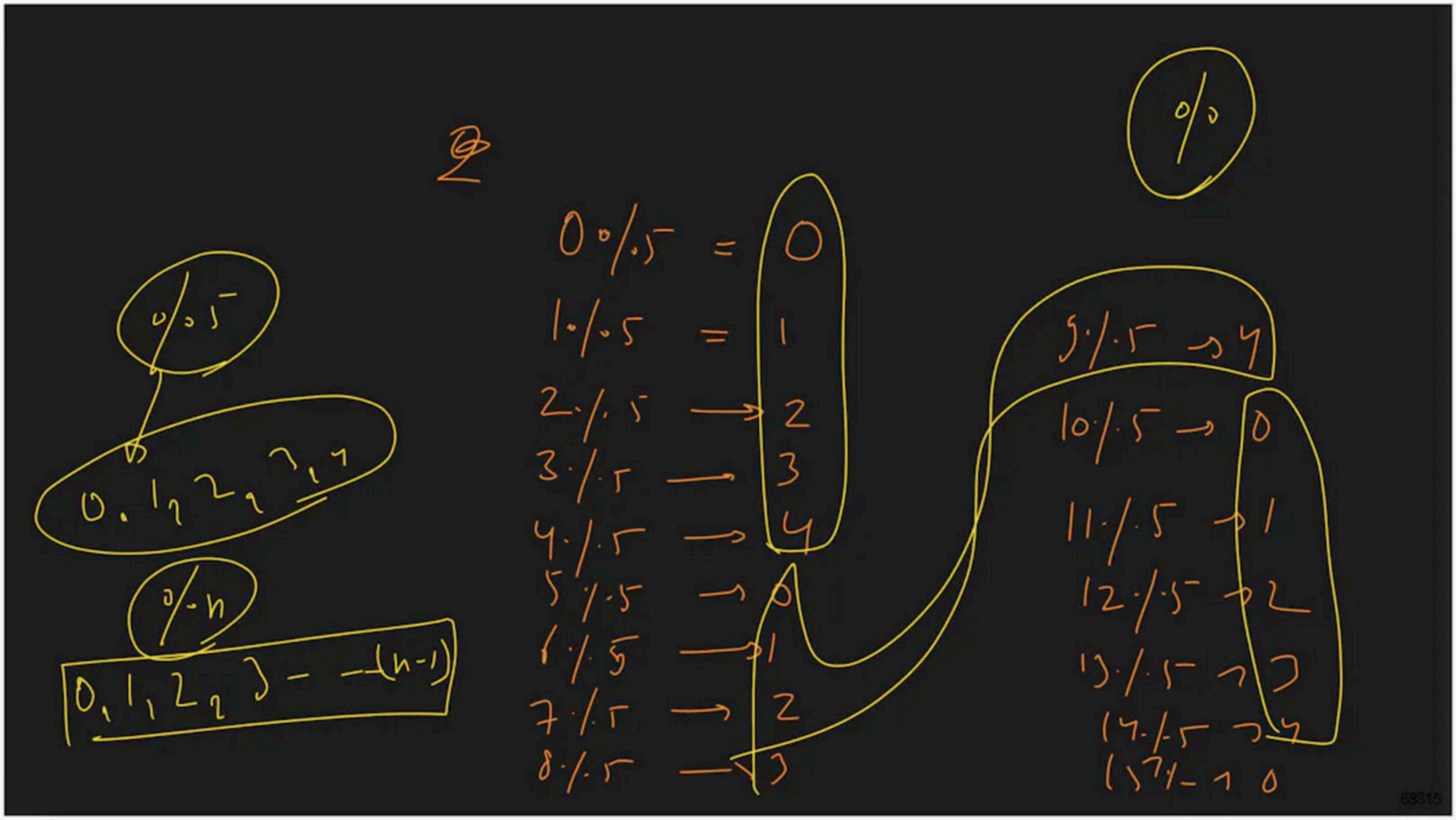

front
rear
front = 0
rear = size - 1
rear
front
rear = -- front - 1

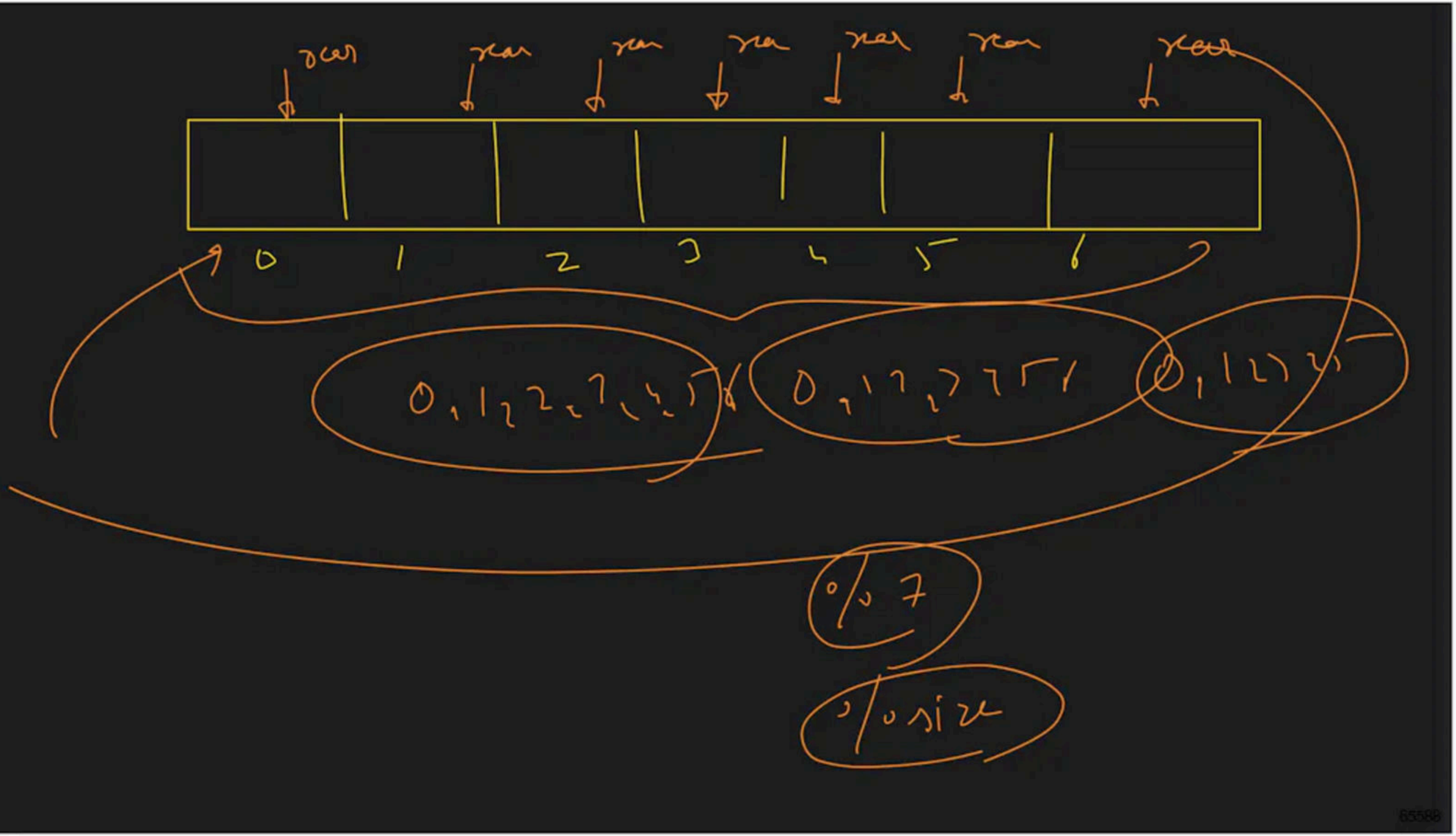
KHUSH




pop
underflow
f → -1 && r → -1
1 element
f == r
f → -1
r → -1
Circular nature
front == size -1
front = 0
normal flow
arr[f] = -1
f++;


rear = front - 1
full


2
0%5
0%5 = 0
1%5 = 1
2%5 → 2
3%5 → 3
4%5 → 4
5%5 → 0
6%5 → 1
7%5 → 2
8%5 → 3
%
5%5 → 4
10%5 → 0
11%5 → 1
12%5 → 2
13%5 → 3
14%5 → 4
15%5 → 0
0,1,2,3,4
0%n
0,1,2,3 ---(n-1)

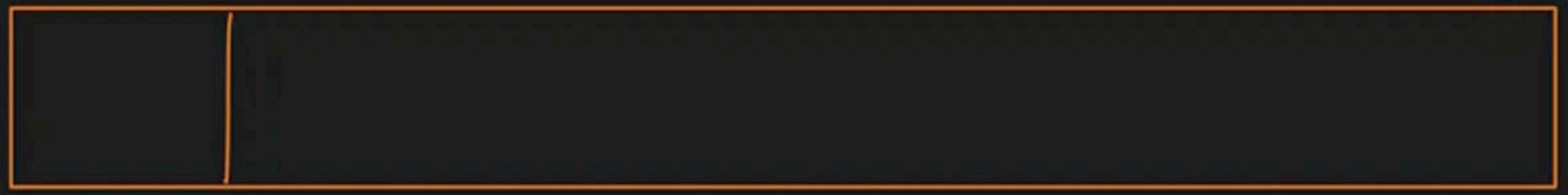

0 1 2 3 4 5 6
% 7
% size

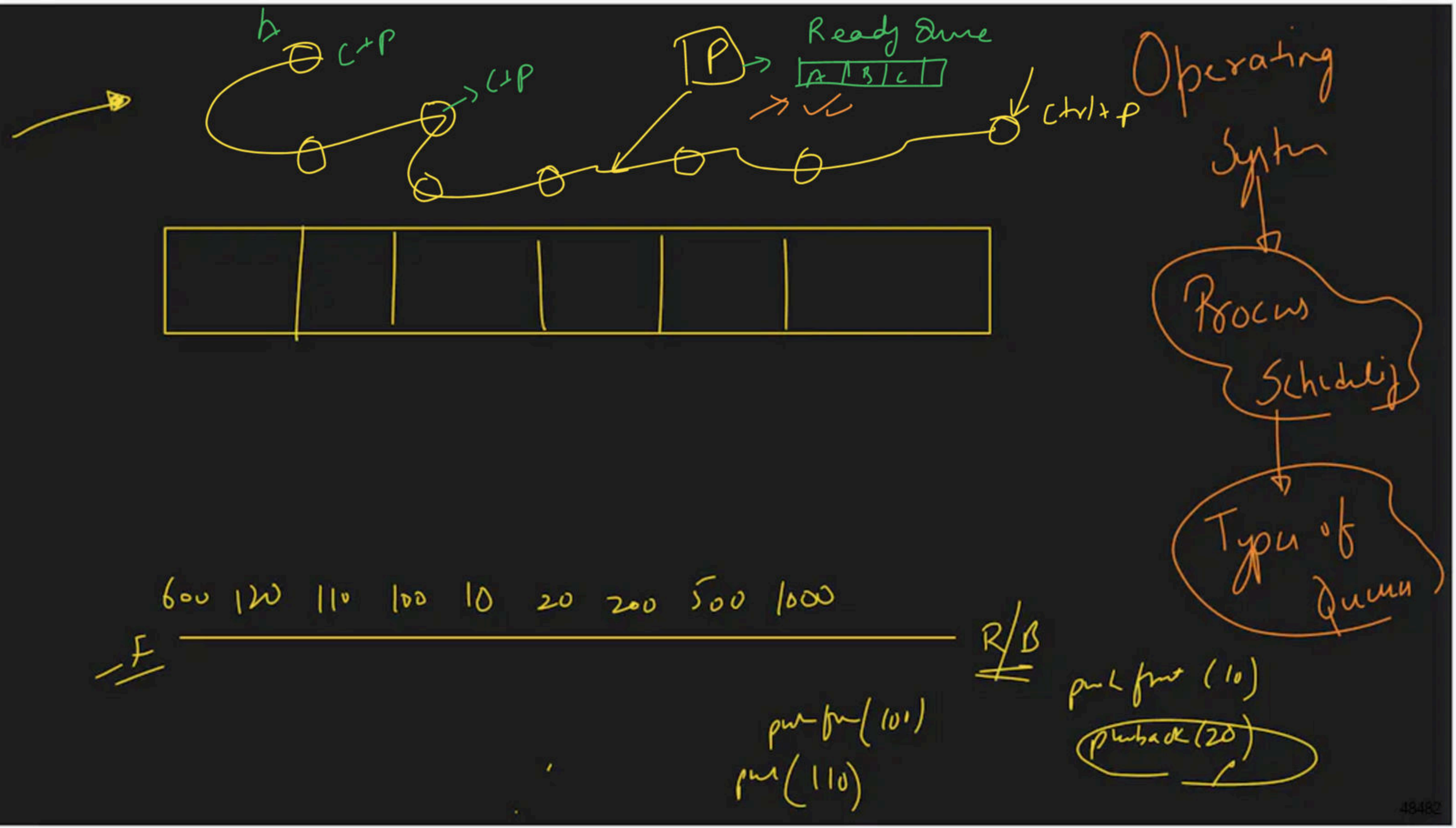
→ Doubly Ended Queue

deque

push_front
pop_front

dequeue
push_back()
pop_back()

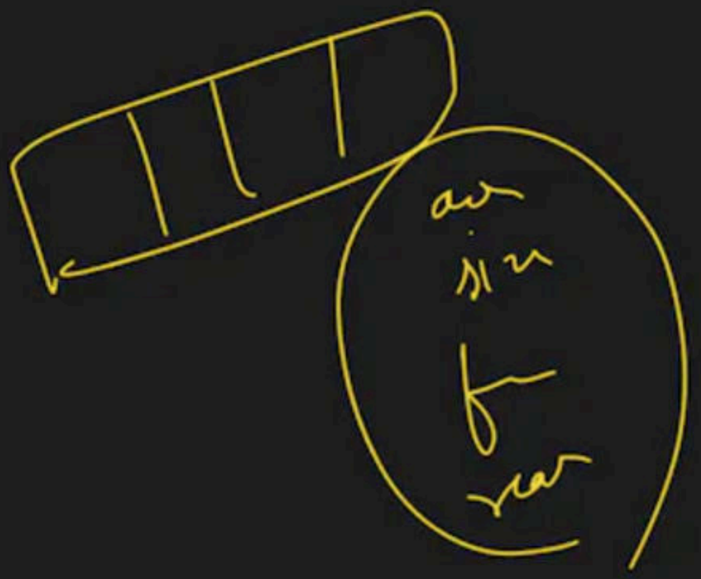


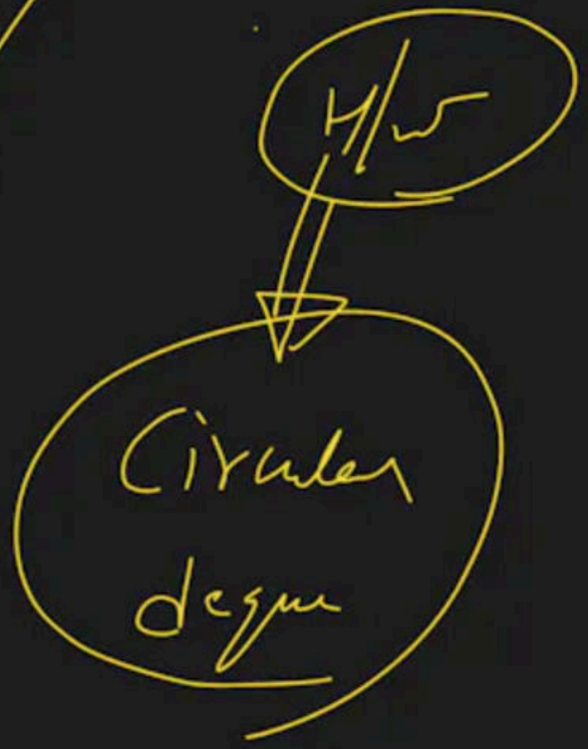

h
Ctrl+p
Ctrl+p
P
Ready Queue
A B C
Ctrl+p
Operating System
Process Scheduling
Types of Queue
600 120 110 100 10 20 200 500 1000
F
R/B
pwd func (101)
pwd (110)
pwd func (10)
Playback (20)

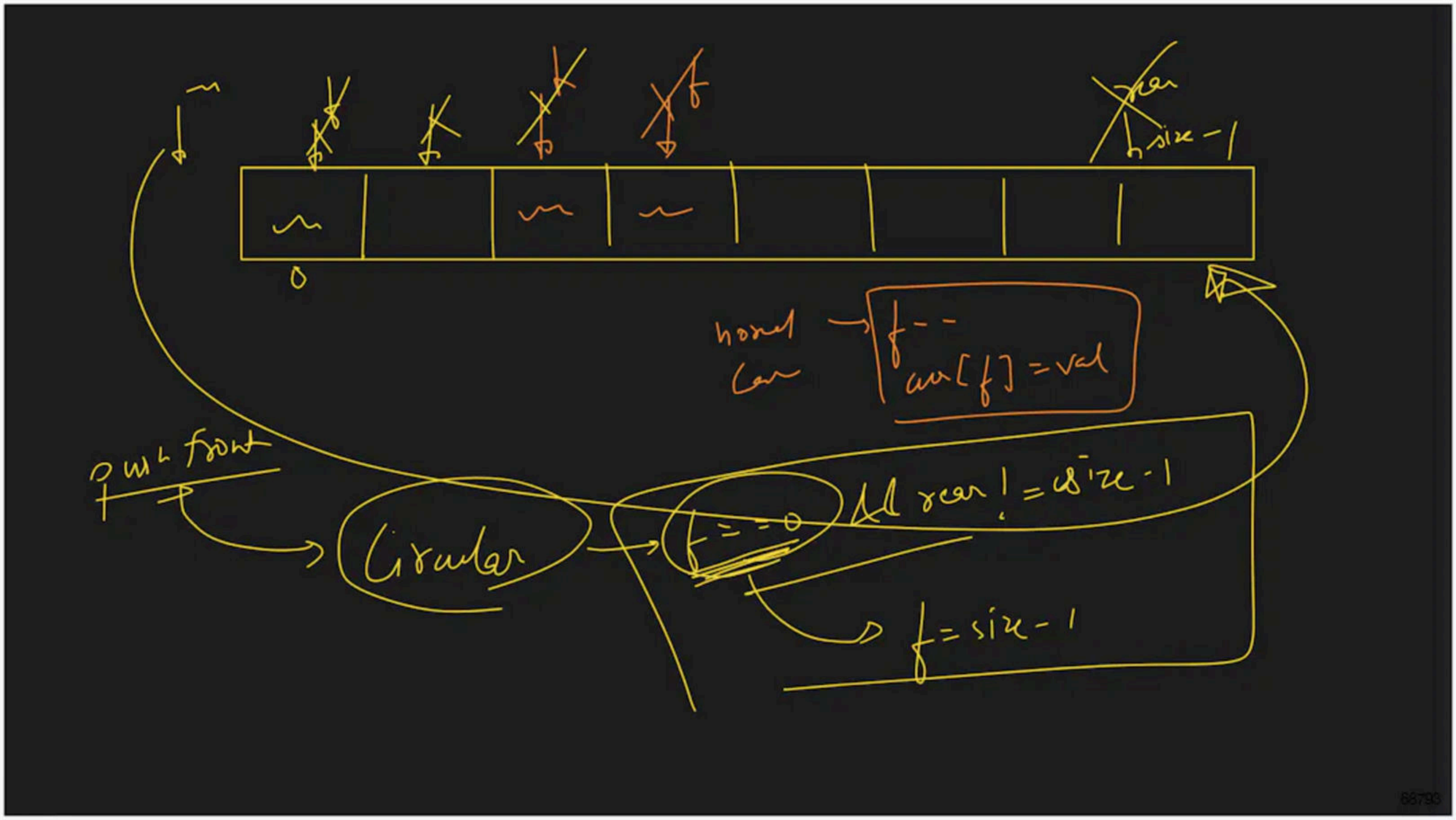
Implement Deque from Scratch

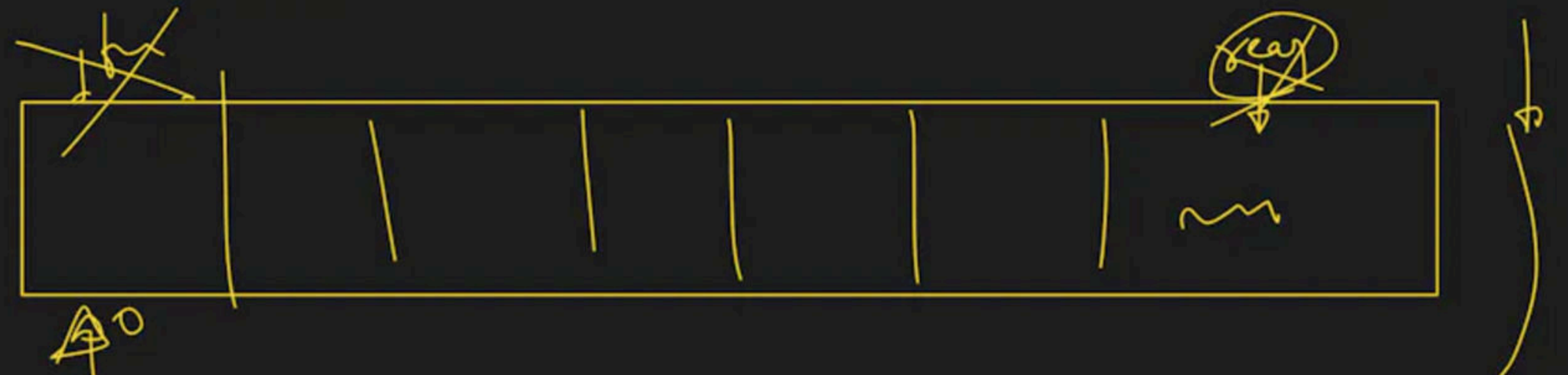
Normally

- push_front
- push_back
- pop_front
- pop_back



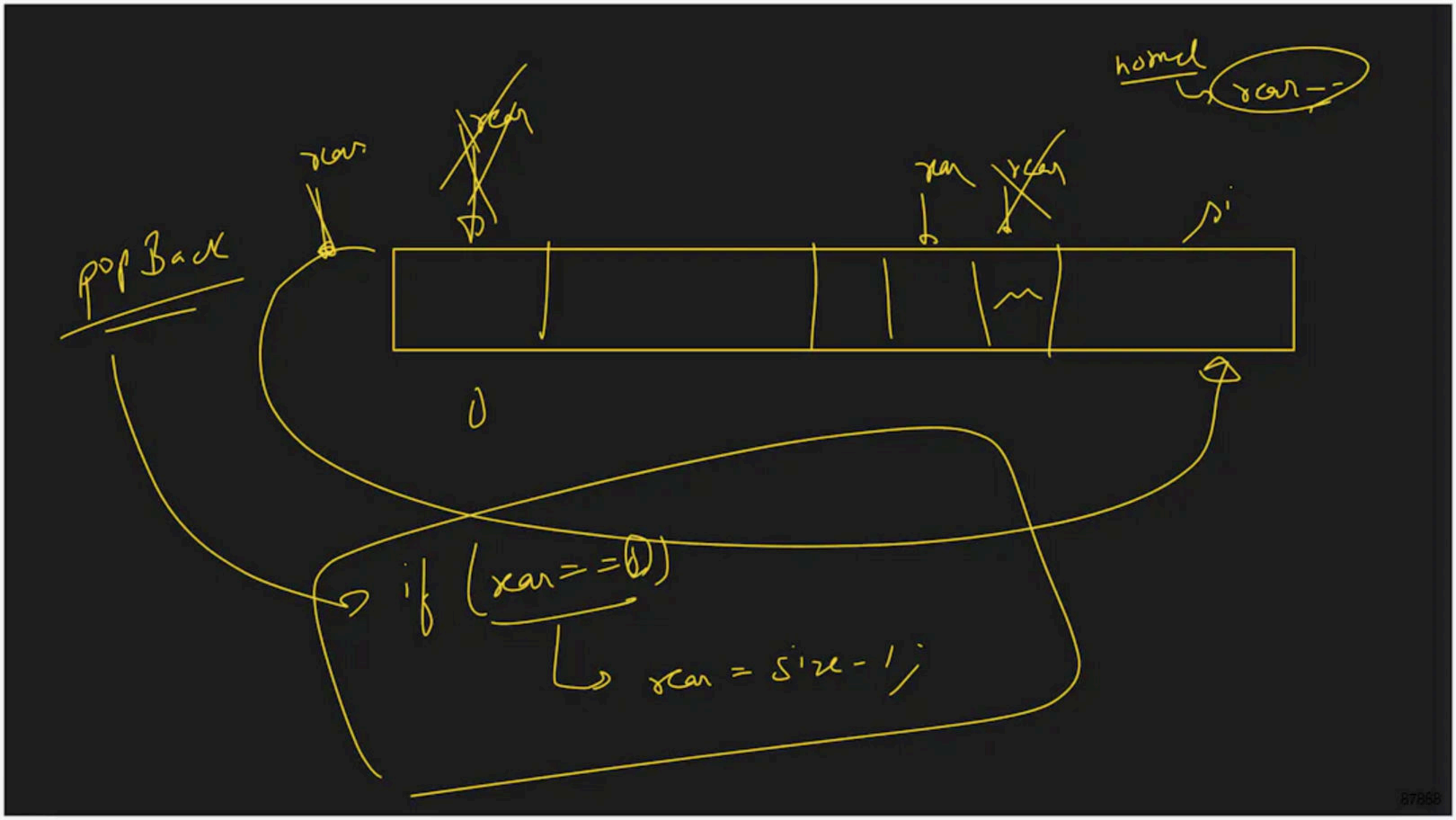


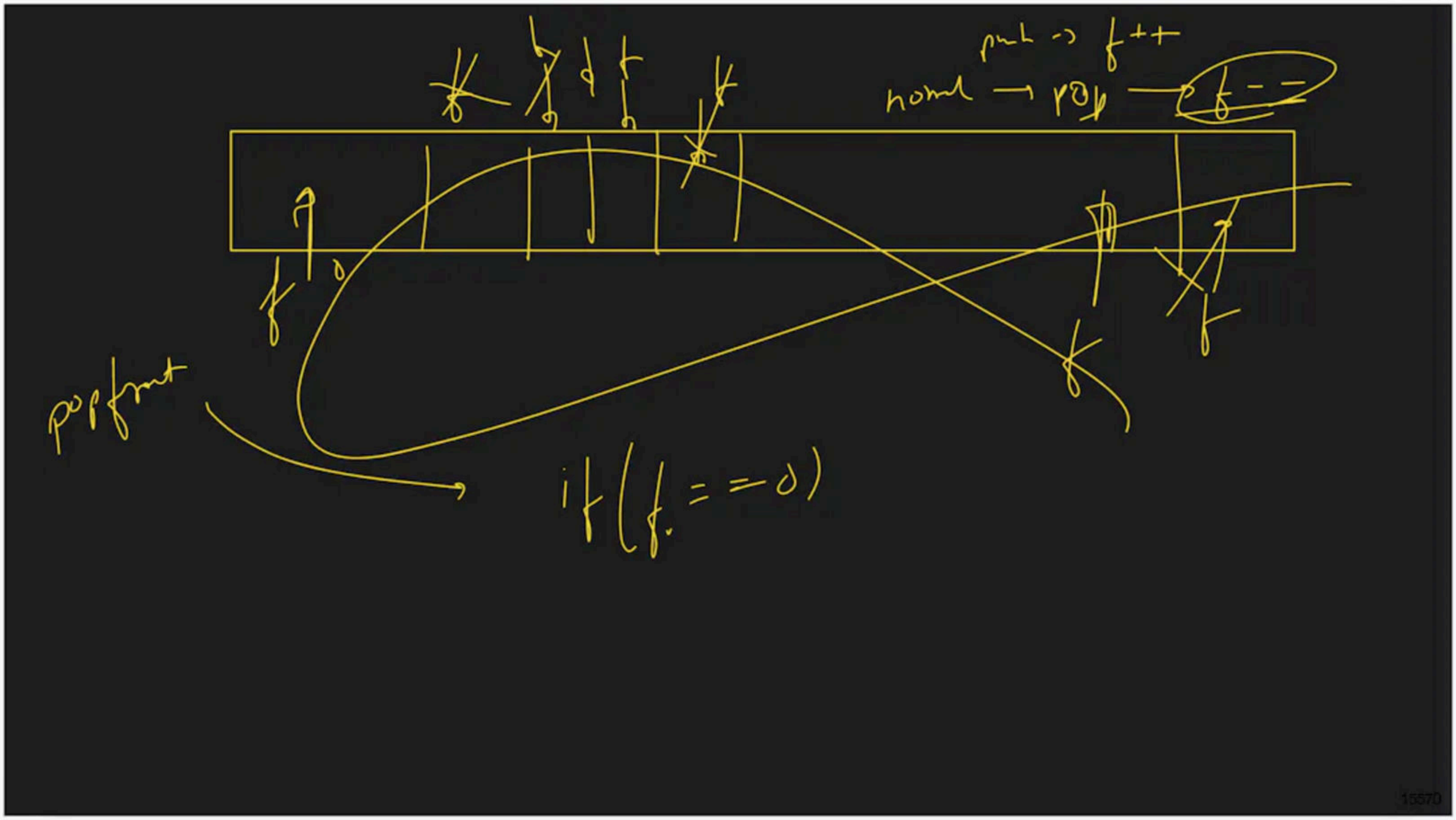

rear
size-1
0
normal case
f--
arr[f] = val
circular front
Circular
f == 0
All rear != size-1
f = size-1

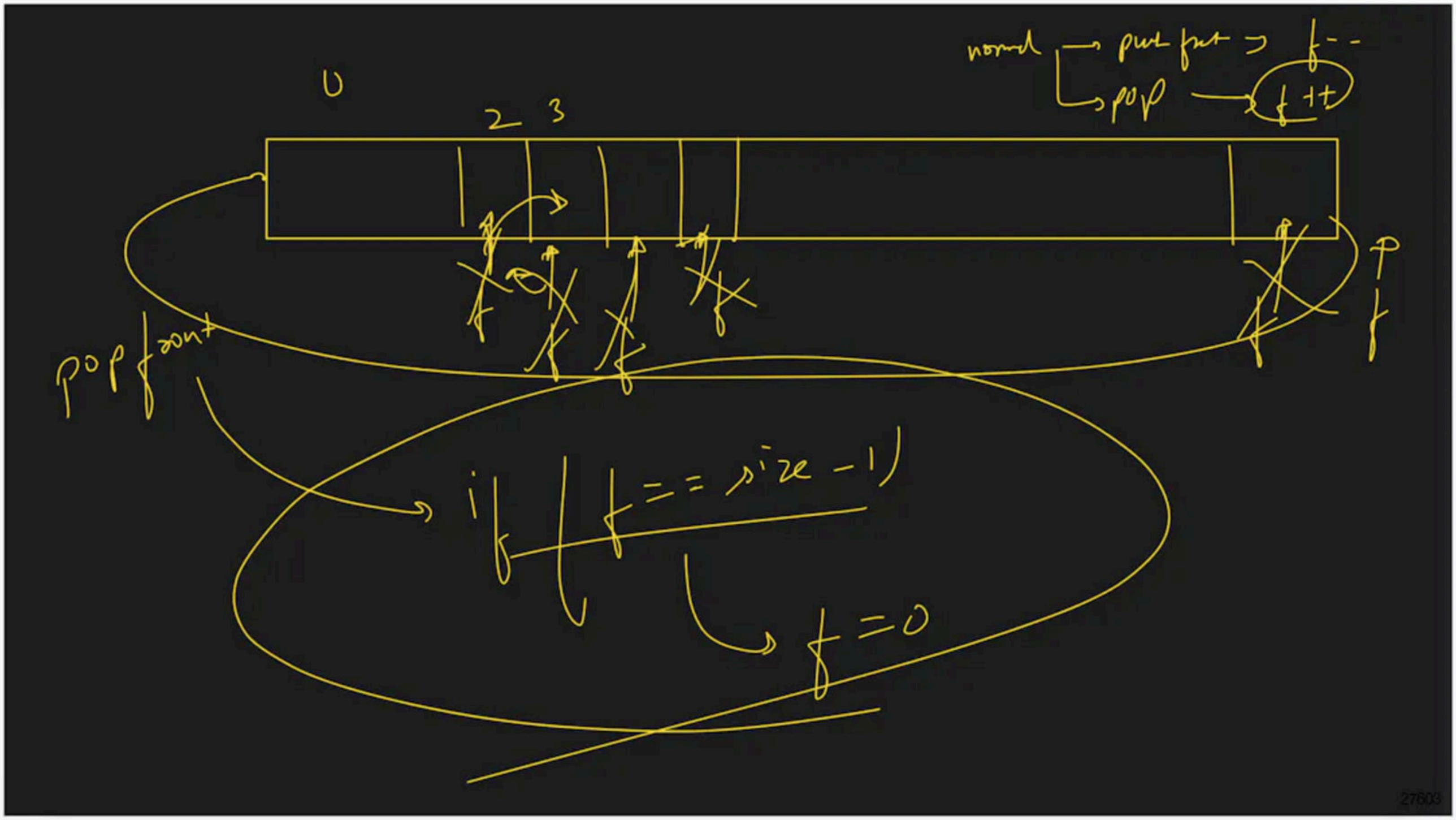


pushBack

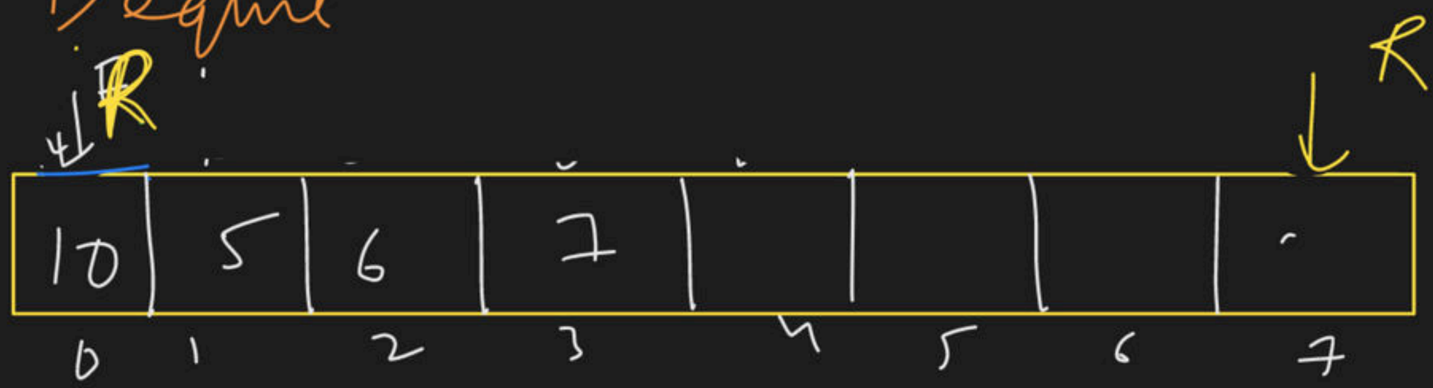
```
if(rear == size-1 && front != 0)
    rear = 0
    arr(rear) = val
```


homed
rear--
rear
rear
rear
rear
si
popBack
0
if (rear==0)
rear = size-1;


postfix
if (f == 0)


0
2 3
normal → push front → f--
pop → f++
pop front
if (f == size - 1)
f = 0
p
f

Implement Deque



=> size = size

arr = new int [size]

front = -1

rear = -1

is full => front == 0 & rear == size - 1

rear == front - 1;

is Empty => front == -1;

& rear == -1

insertfront()

=

① Overflow => isFull()

② empty

F == -1 & R == -1

F++, R++;

a[F] = val;

③ Normal case

F--;

a[F] = val;

④ Circular case

(front == 0 & R != S-1)

front = S-1, a[F] = val.

(2) Insert Last (1) IsFull

(2) Empty => F == -1 & R == -1

(3) Circular : R == Size-1 & F != 0

(4) R++ arr[R] = val;

(3) del Front

(1) isEmpty()

(2) F == R, single Element

a[f] = -1;, F = R = -1;

(3) Circular can

F == size - 1

=> F = 0

(4) Normal => a[F] = -1; F++;

④ del Rear+( ) ① is Empty

② front == R F = R = -1

③ R == 0, circular ca R = S - 1;

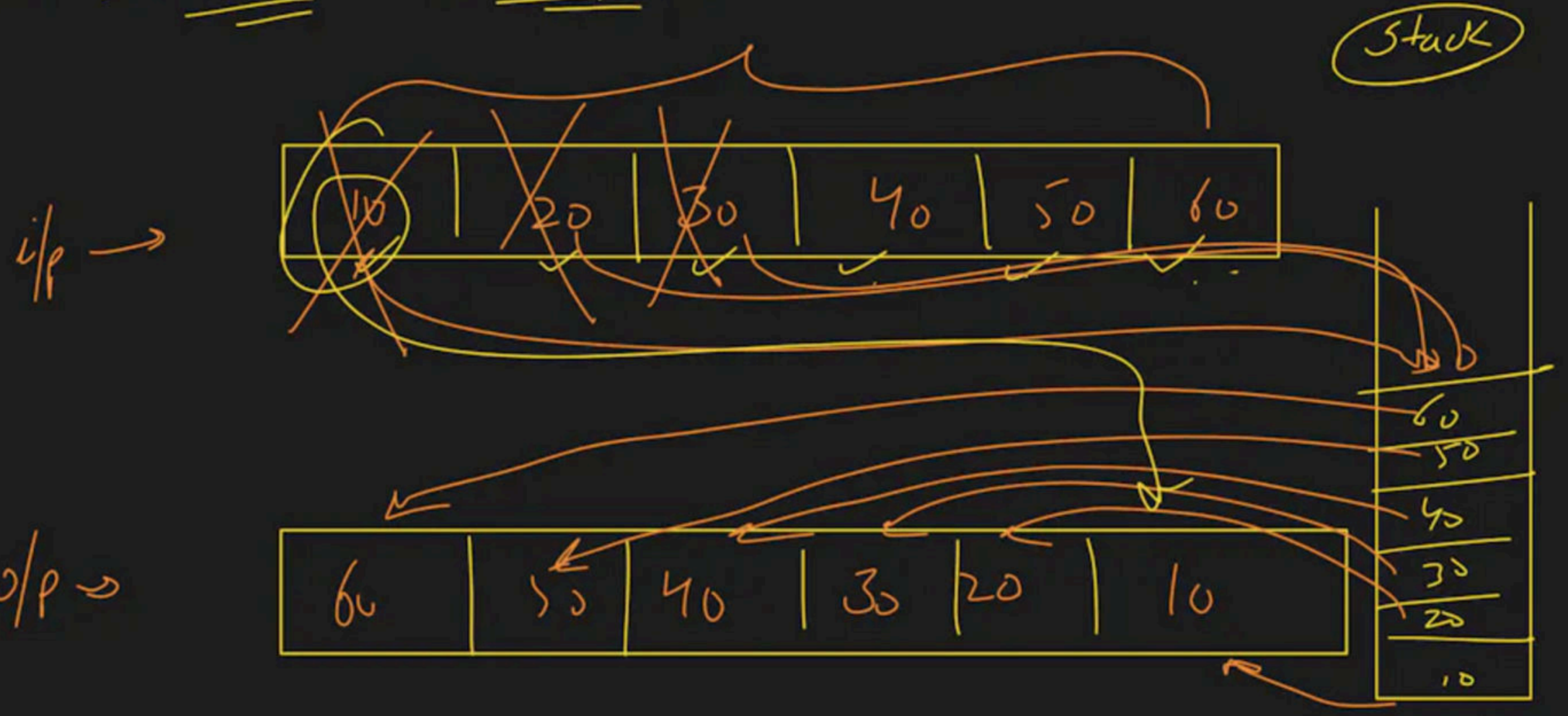
④ a[R] = -1, R = -;

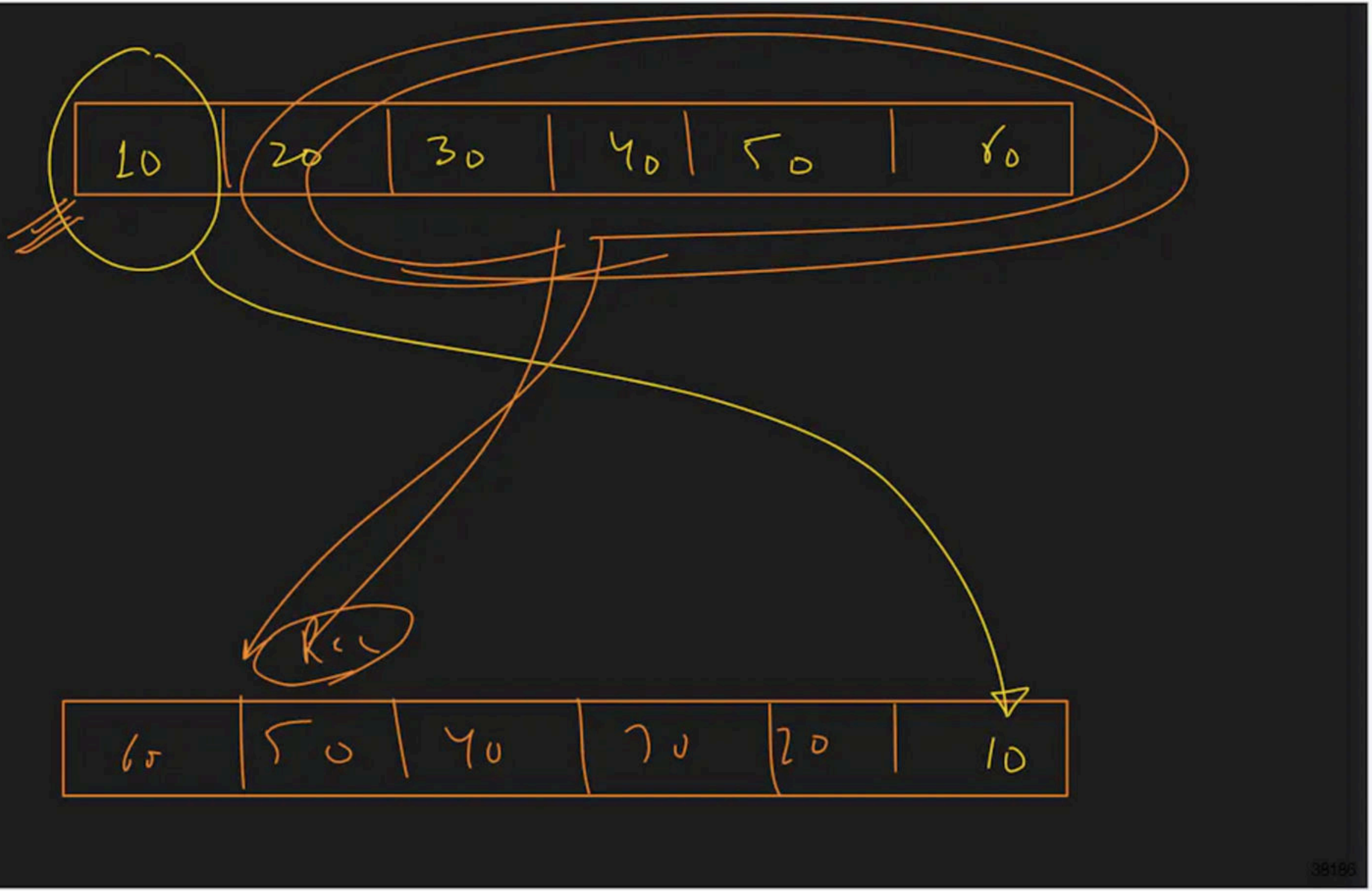
Queue Class - 2

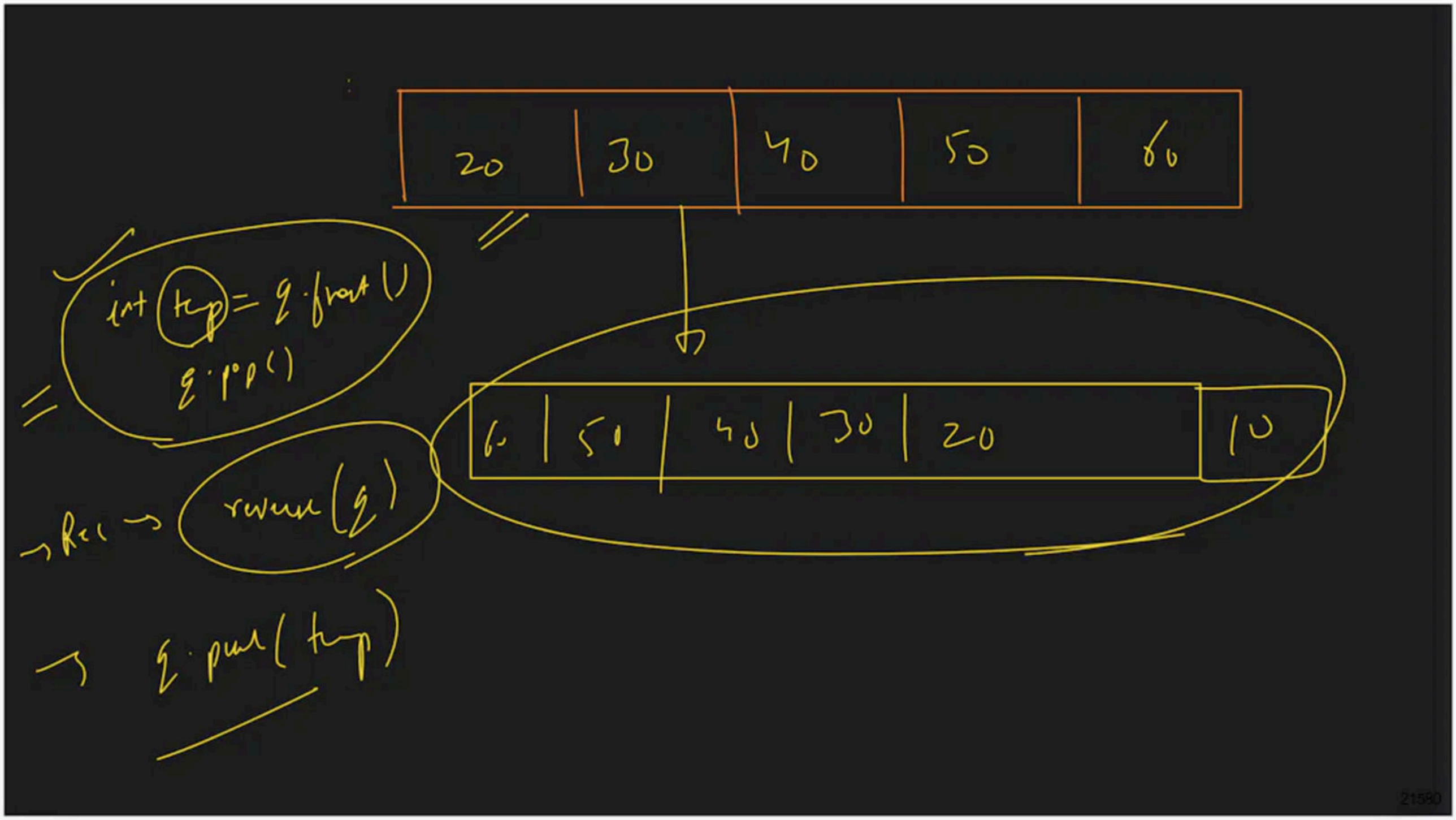
Special class

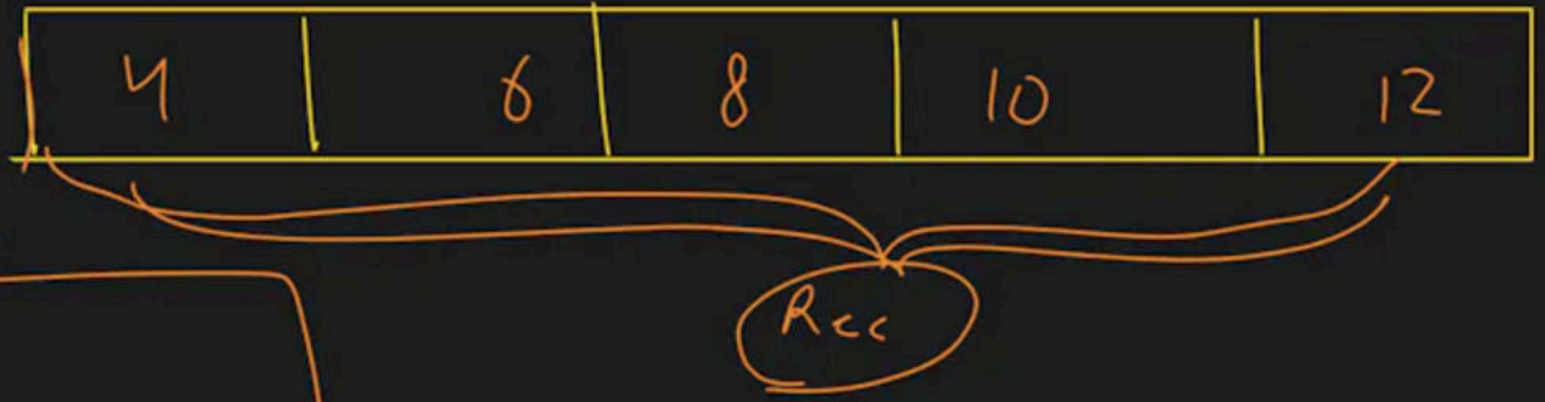
Love Babbar • Nov 22, 2023

→ Reverse a Queue

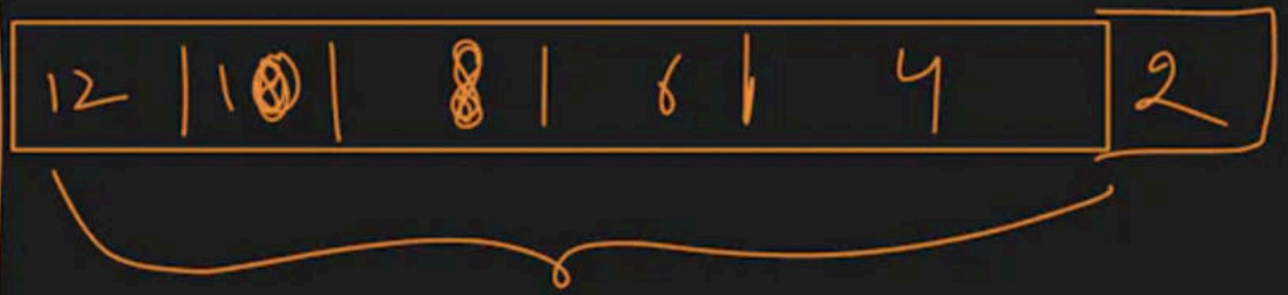



10
20
30
40
50
60
60
50
40
30
20
10

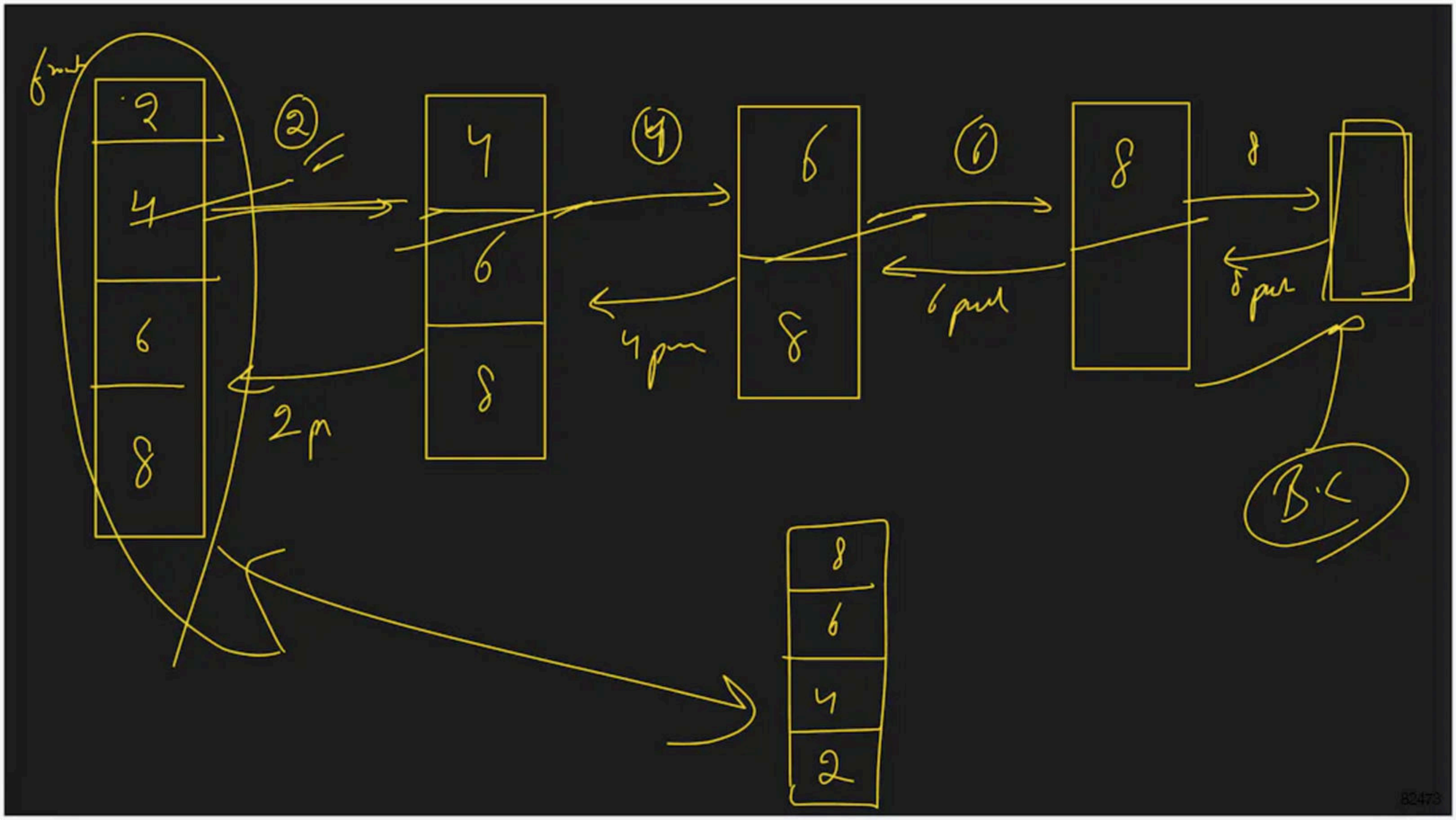

20
30
40
50
60
int temp = q.front()
q.pop()
=
60
50
40
30
20
10
→Rec→
return(q)
→ q.push(temp)




```
int elemnt = q.front()
q.pop()
reverse(q)
q.push(elemnt)
```

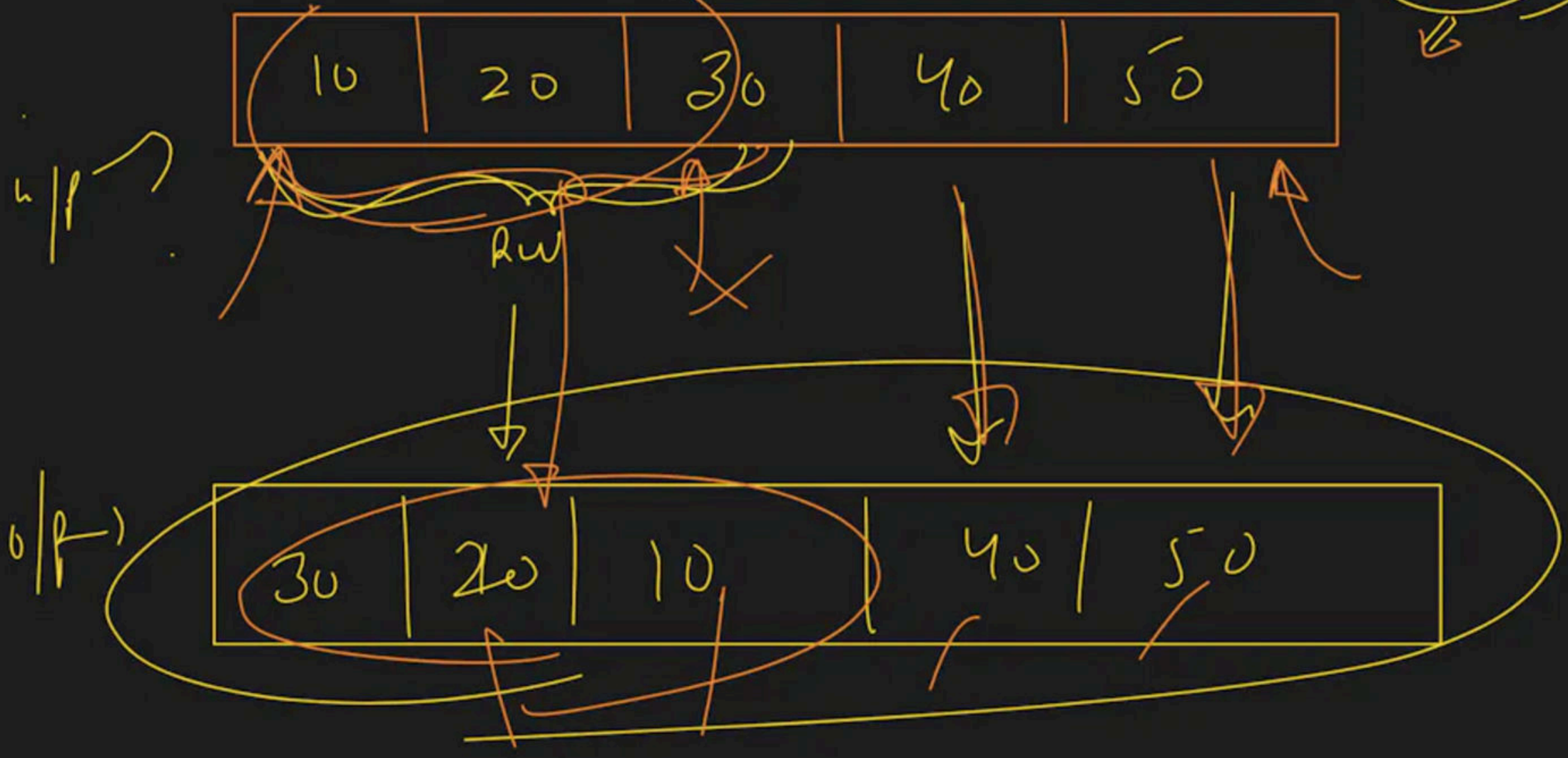


elemnt = 2


2
4
6
8
2
4
6
8
4
6
8
6
8
8
2 pm
4 pm
6 pm
8 pm
B.C
8
6
4
2

Queue → Reverse K elements

K = 3



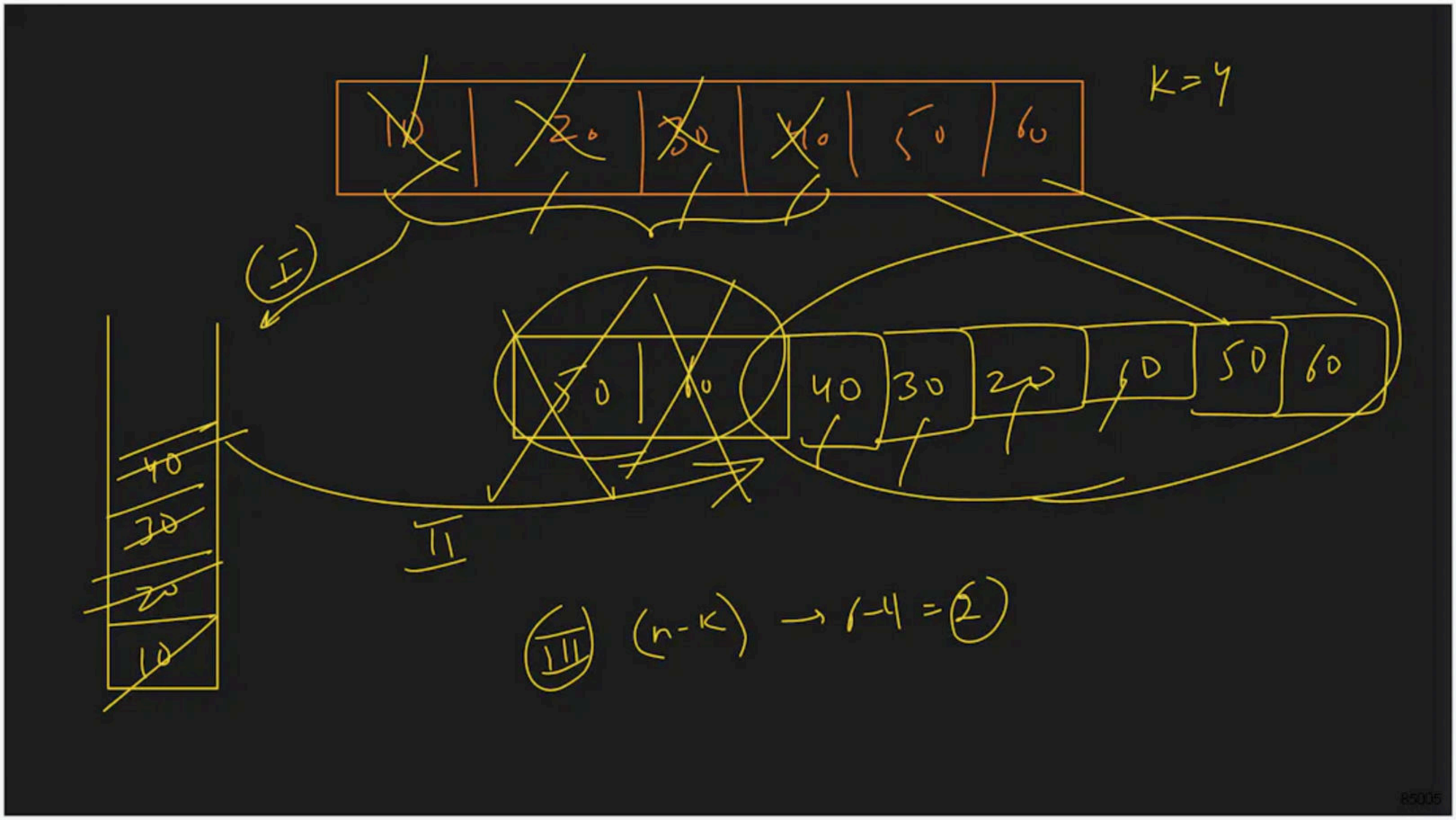
#

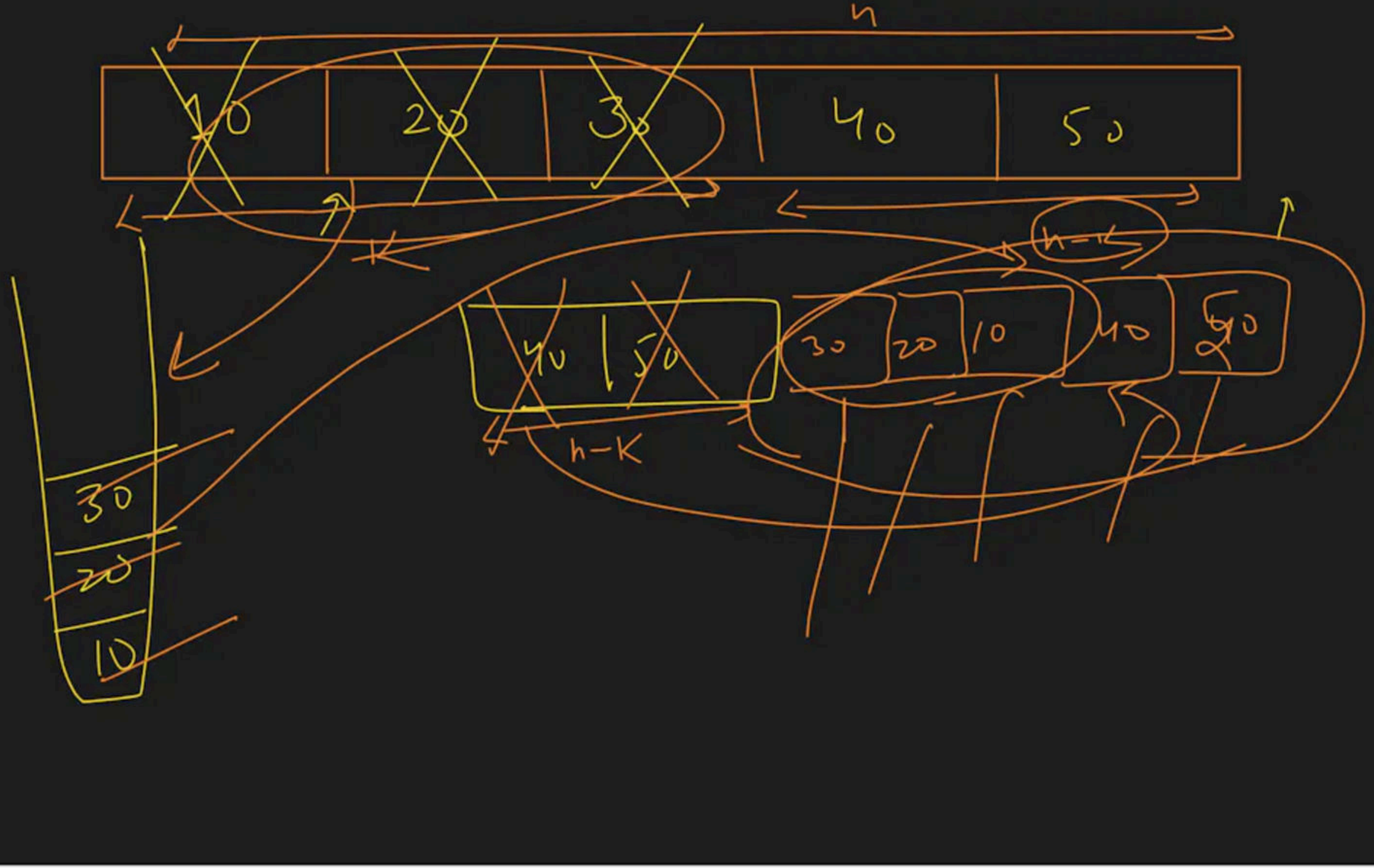
(1) push first K elements from Q → S

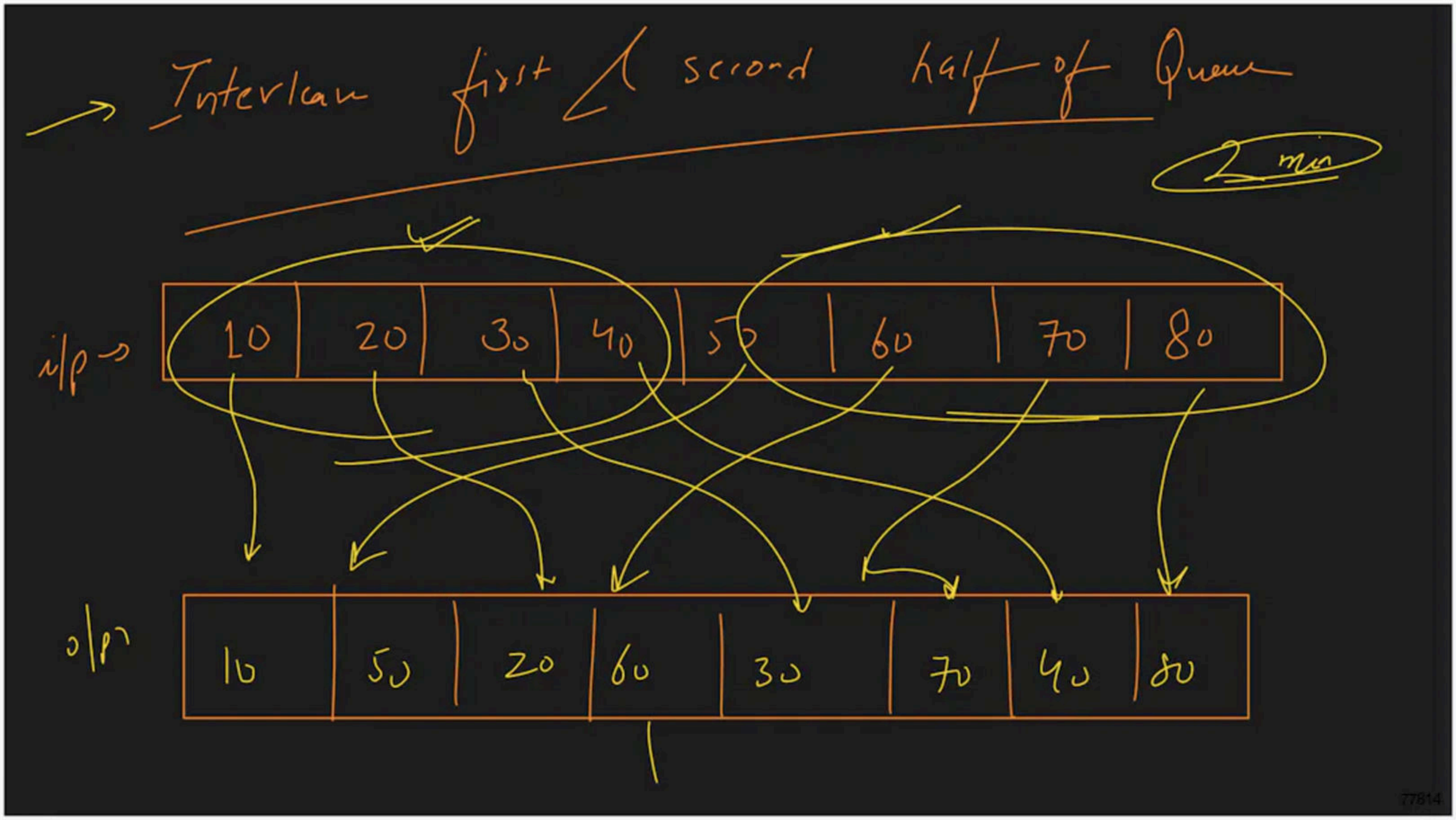
(2) push last K elements from S → Q

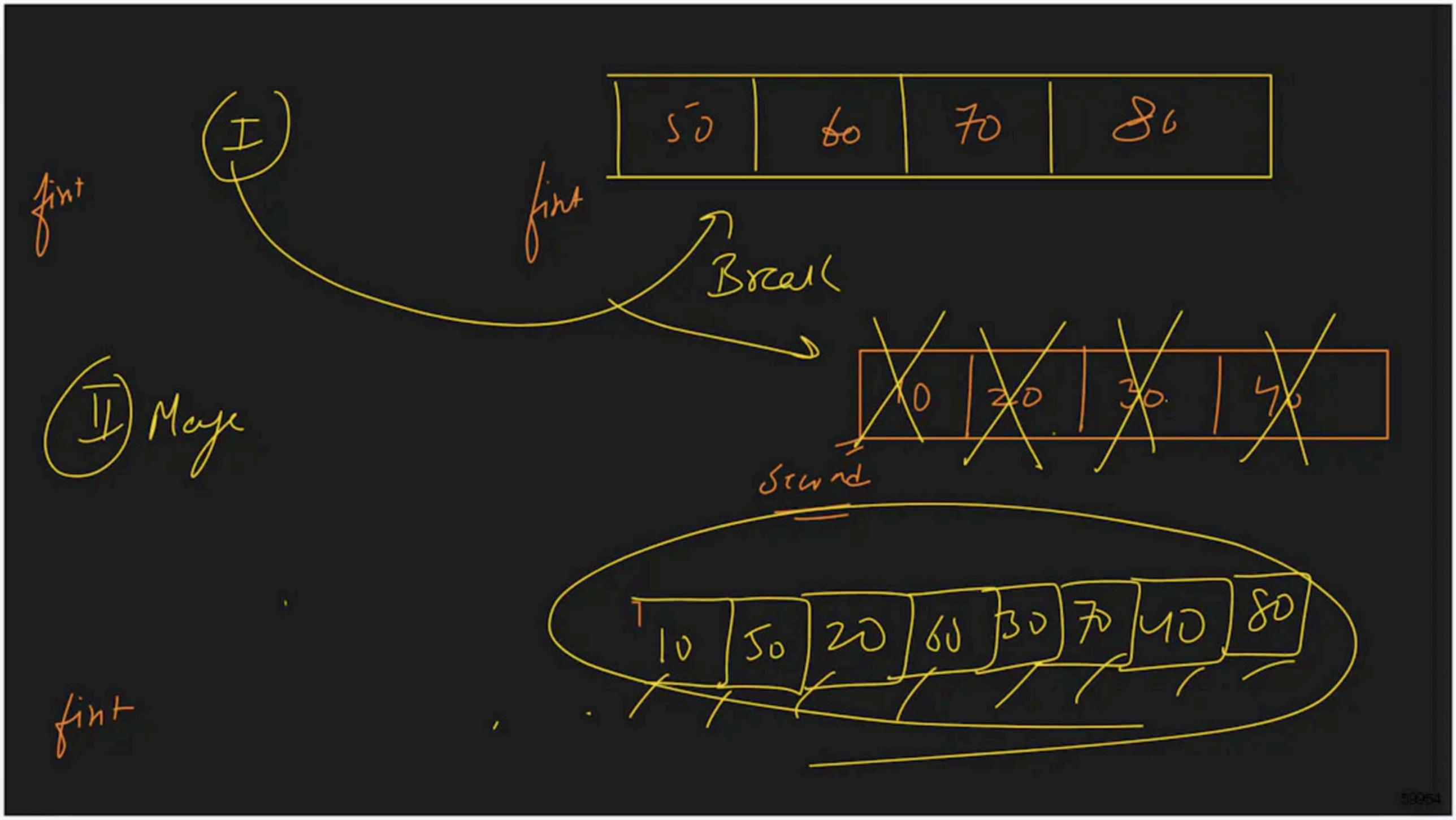
(3) Q → first (n − K)

pop → push

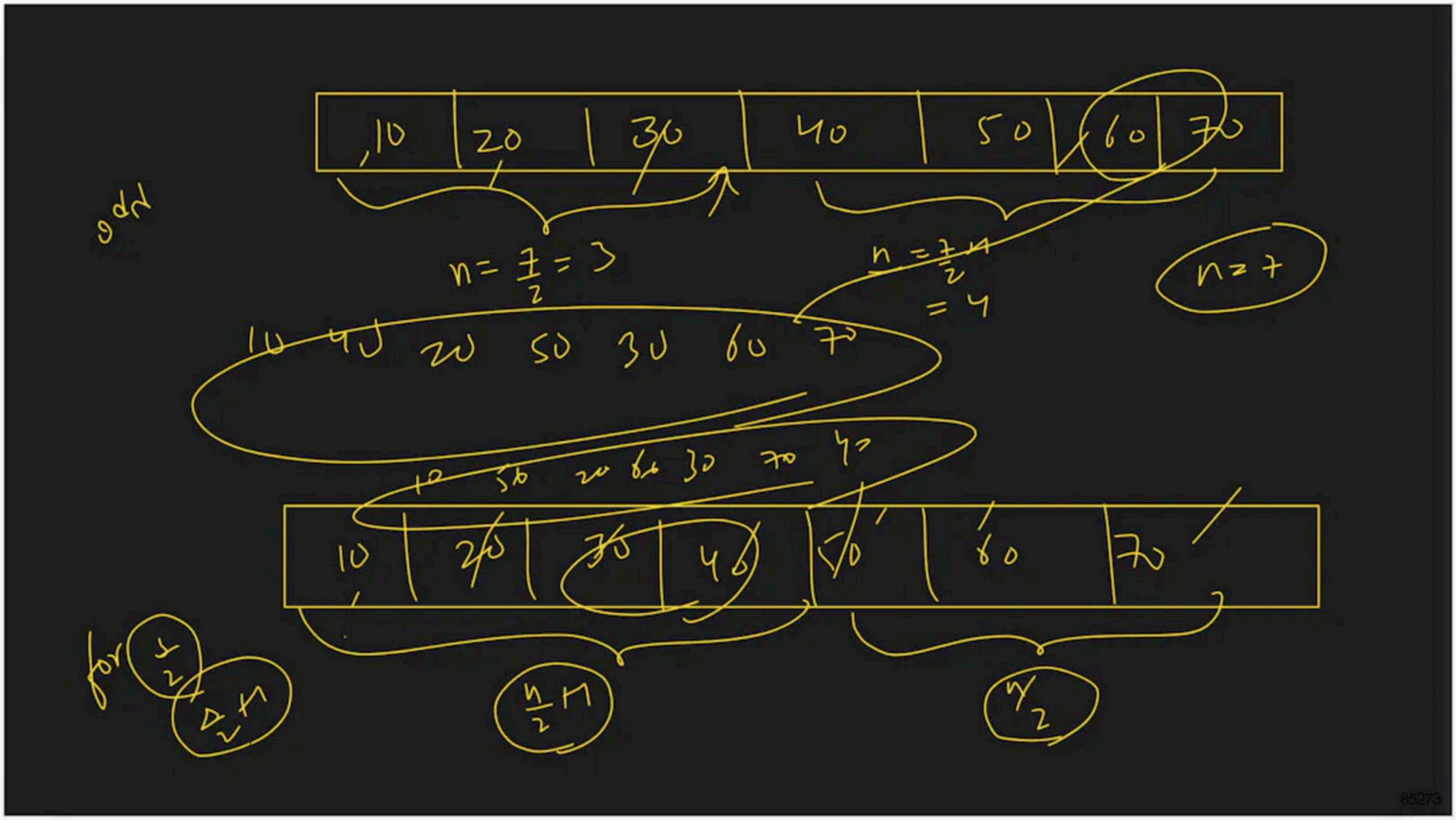

K=4
10
20
30
40
50
60
I
II
III
(n-k) → 6-4 = 2

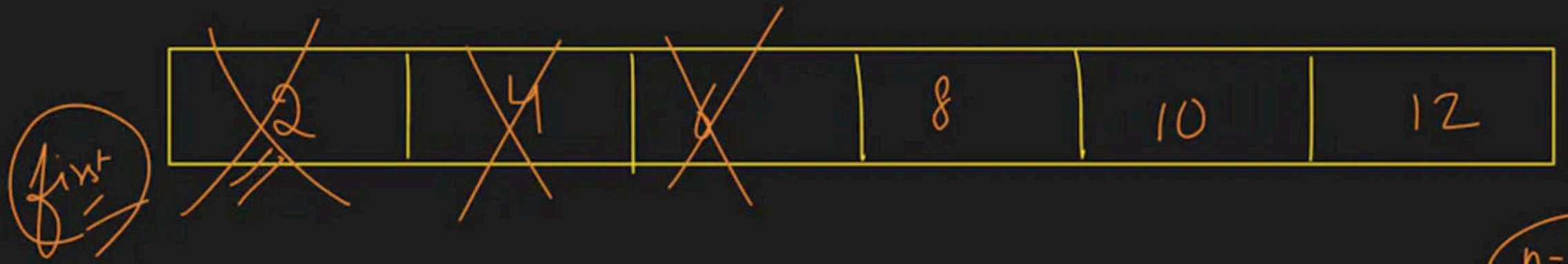

n
10
20
30
40
50
K
n-K
40
50
30
20
10
40
50
n-K
30
20
10


Interleave first & second half of Queue
2 min
i/p
10
20
30
40
50
60
70
80
o/p
10
50
20
60
30
70
40
80


(I)
first
first
50
60
70
80
Break
(II) Map
10
20
30
40
second
first
10
50
20
60
30
70
40
80


10 | 20 | 30 | 40
15 | 18 | 17 | 19


10 20 30 40 50 60 70
odd
n = 7/2 = 3
n = 7/2 = 4
n = 7
10 40 20 50 30 60 70
10 50 20 60 30 70 40
10 20 30 40 50 60 70
for
n/2
n/2 + 1
n/2 + 1
n/2

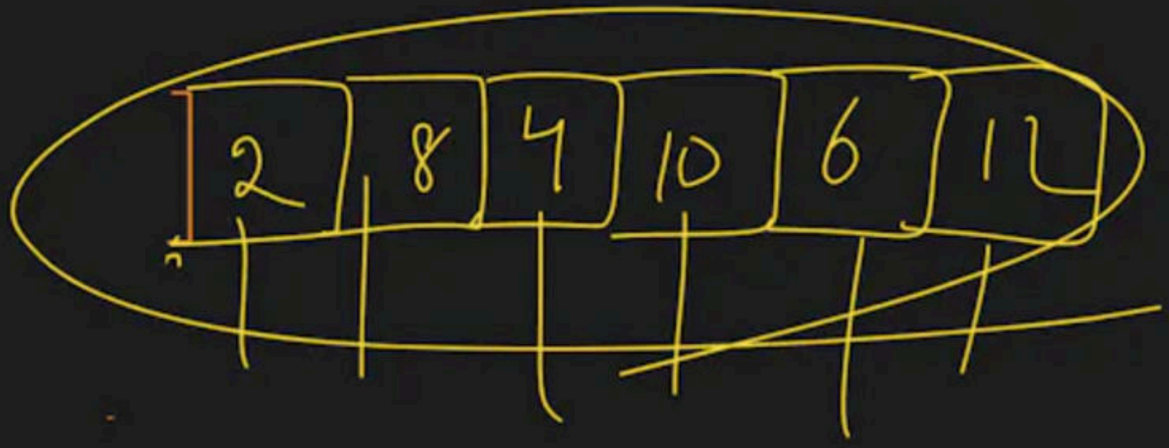


n = 6

i = 0

```
for (i=0; i<n/2; i++)
{
    int temp = first.front();
    first.pop();
    second.push(temp);
}
```

second → 2 4 6



second


```
for (i=0; i<n/2; i++)
{
  tmp = second.front()
        second.pop()
  first.push(tmp)

  tmp = first.front()
        first.pop()
  first.push(tmp)
}
```

→ first -ve integer in every window of "K"

K = 3

2	-5	4	-1	-2	0	5
0	1	2	3	4	5	6

arr

w1 → 2 -5 4 → -5

w2 → -5 4 -1 → -5

w3 → 4 -1 -2 → -1

w4 → -1 -2 0 → -1

w5 → -2 0 1 → -2

-ve ✗ → 0 print

Brute

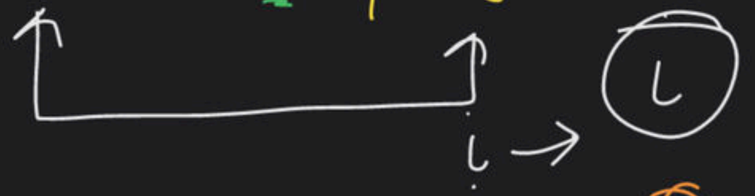
n-K+1

K

n*K

=> 2 | -5 | 4 | -1 | -2 | 0 | 5

=>



Queue

index

4

k = 3

① -5

② -5

③ -1

④ -1

⑤ -2

i - qF ≥ k

↳ Pop Front

qki wo Bahar hai

old window ka hai

⇒

./a.out

./main

Fatak

Run

1 int a;

2 int b;

Ruk jayega → 3 -- —

4 for (;

5 — —

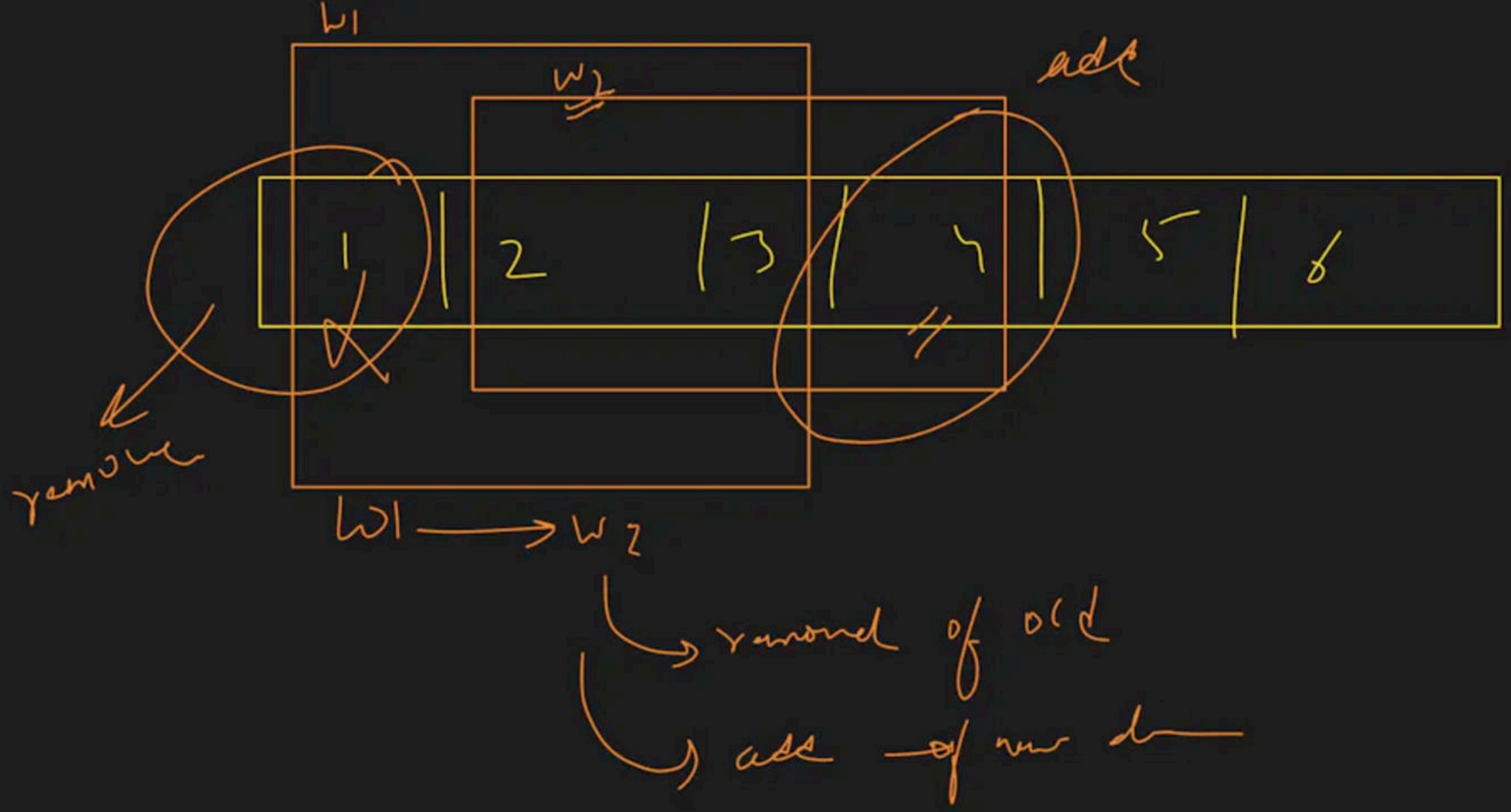
6 — —

7 - —

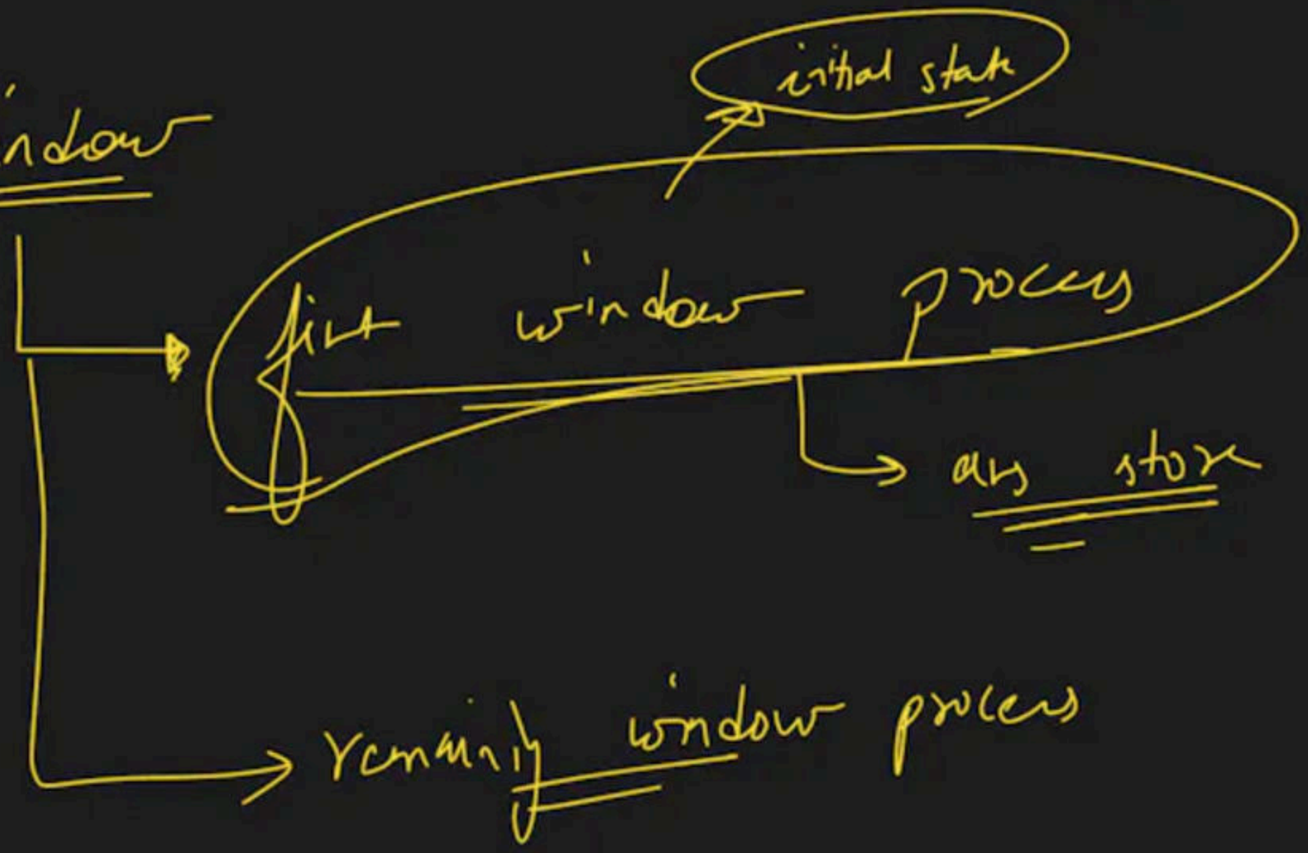
Magic saari states

↳ all variables

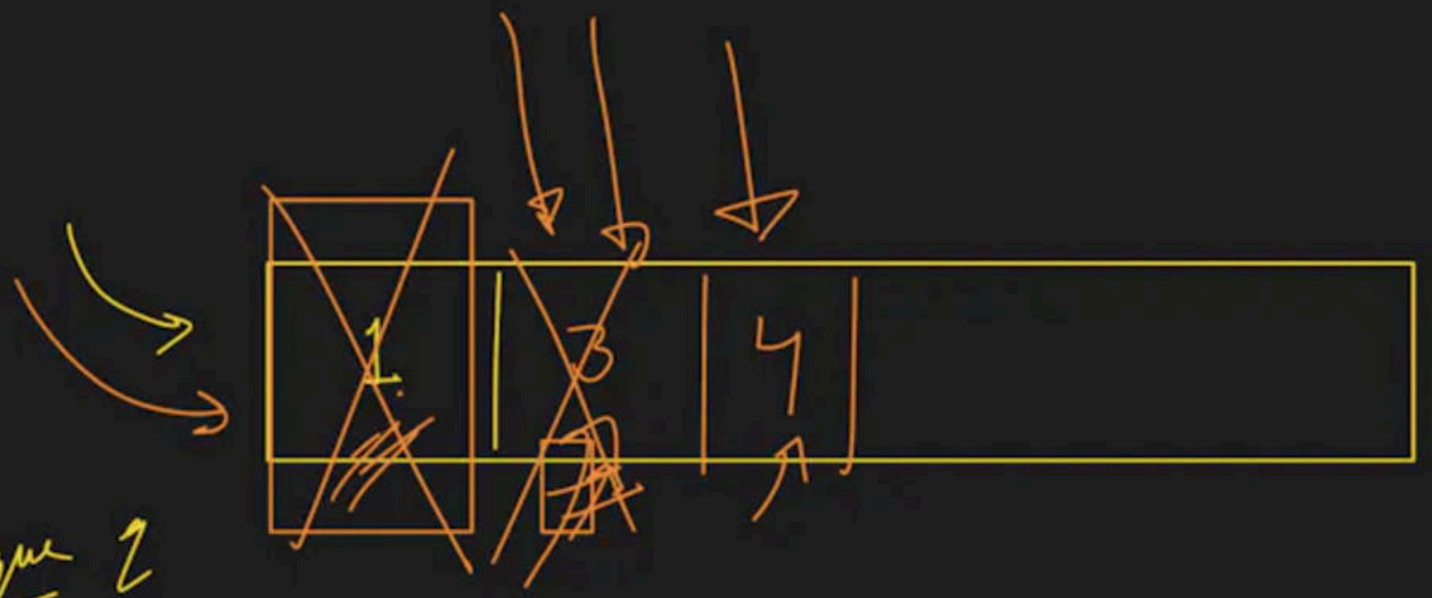
with values

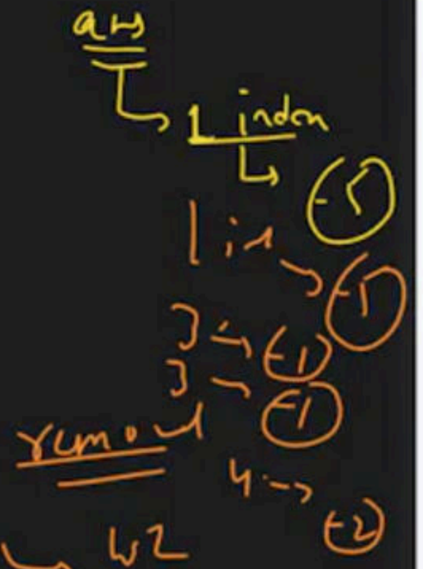

W1
W2
add
1
2
3
4
5
6
remove
W1 → W2
removed of old
add of new

→ Sliding Window



algo

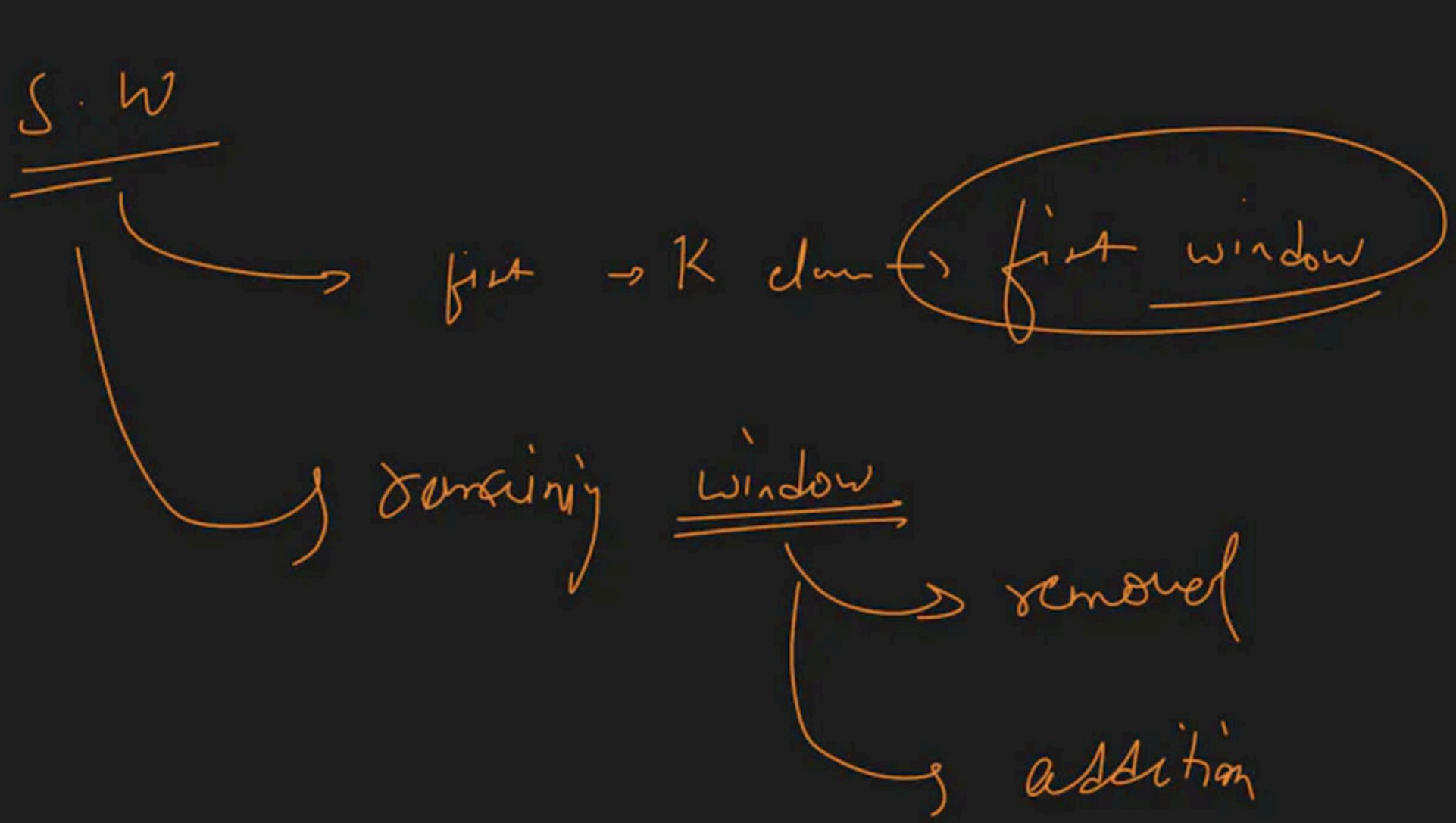


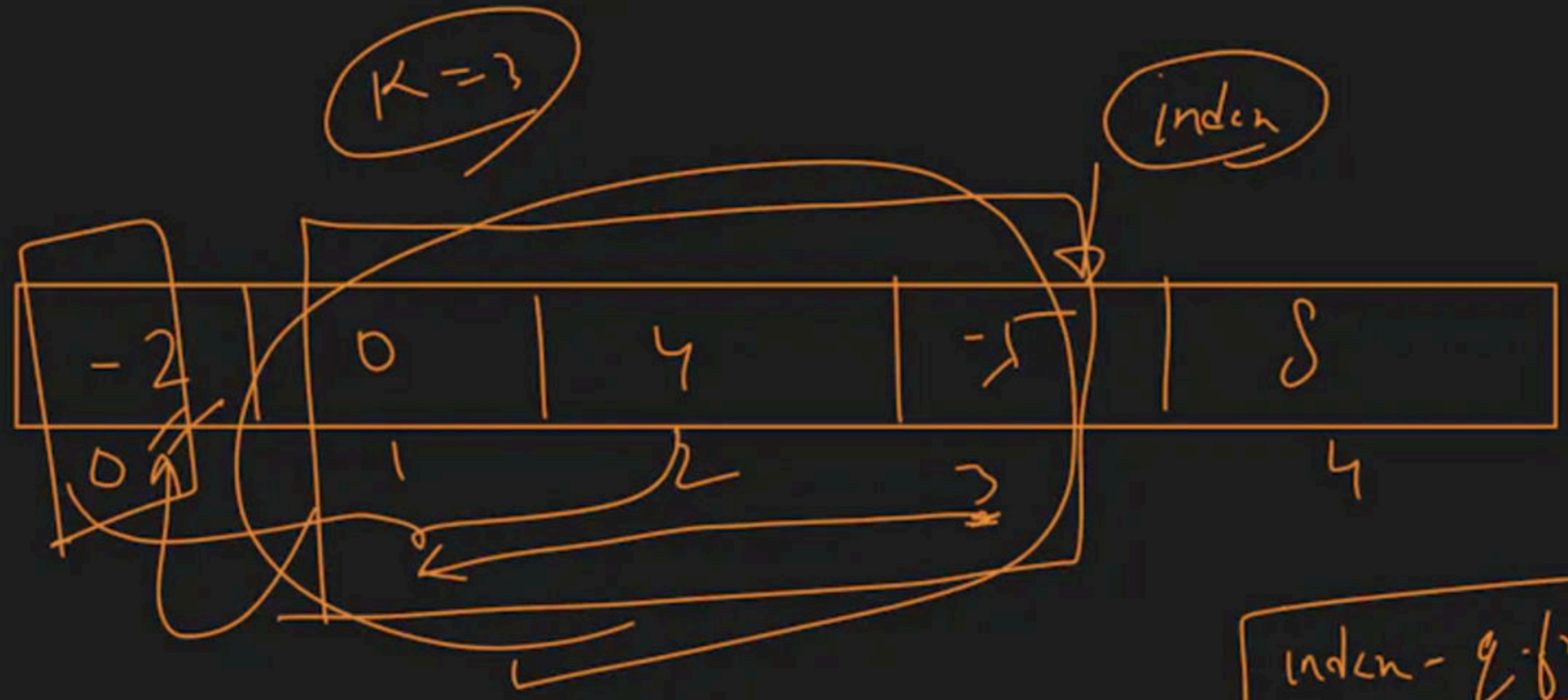


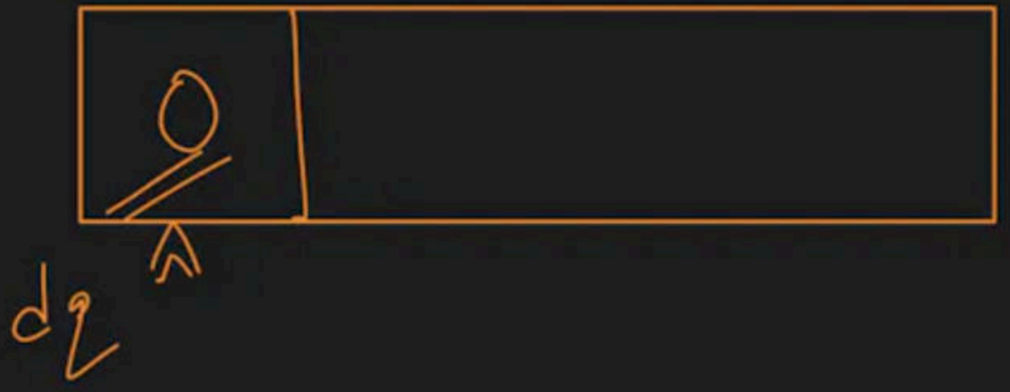
□. sum 2

index of -ve elmt

why


S.W
fixed → K elements → fixed window
sliding window
removed
addition



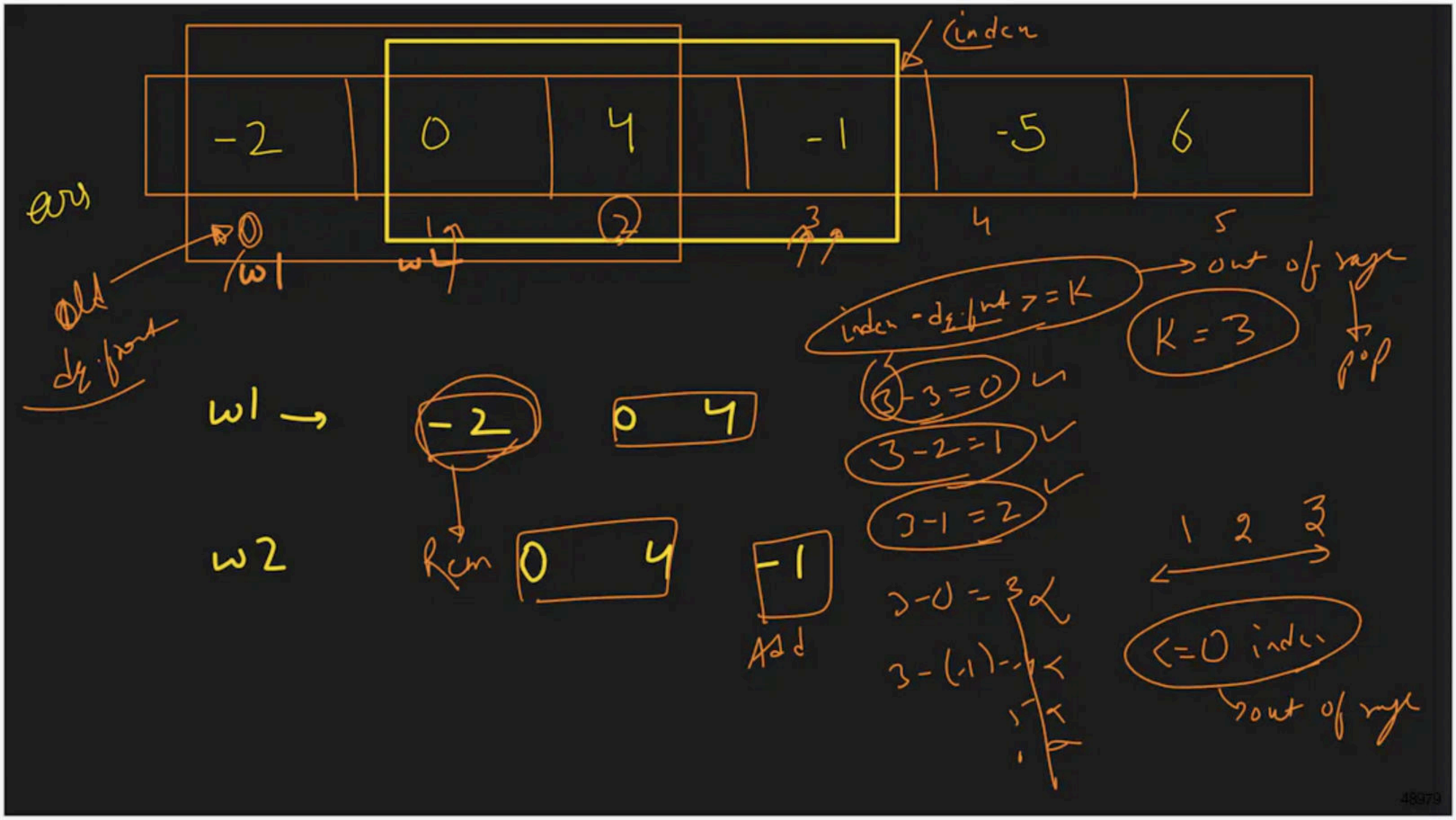


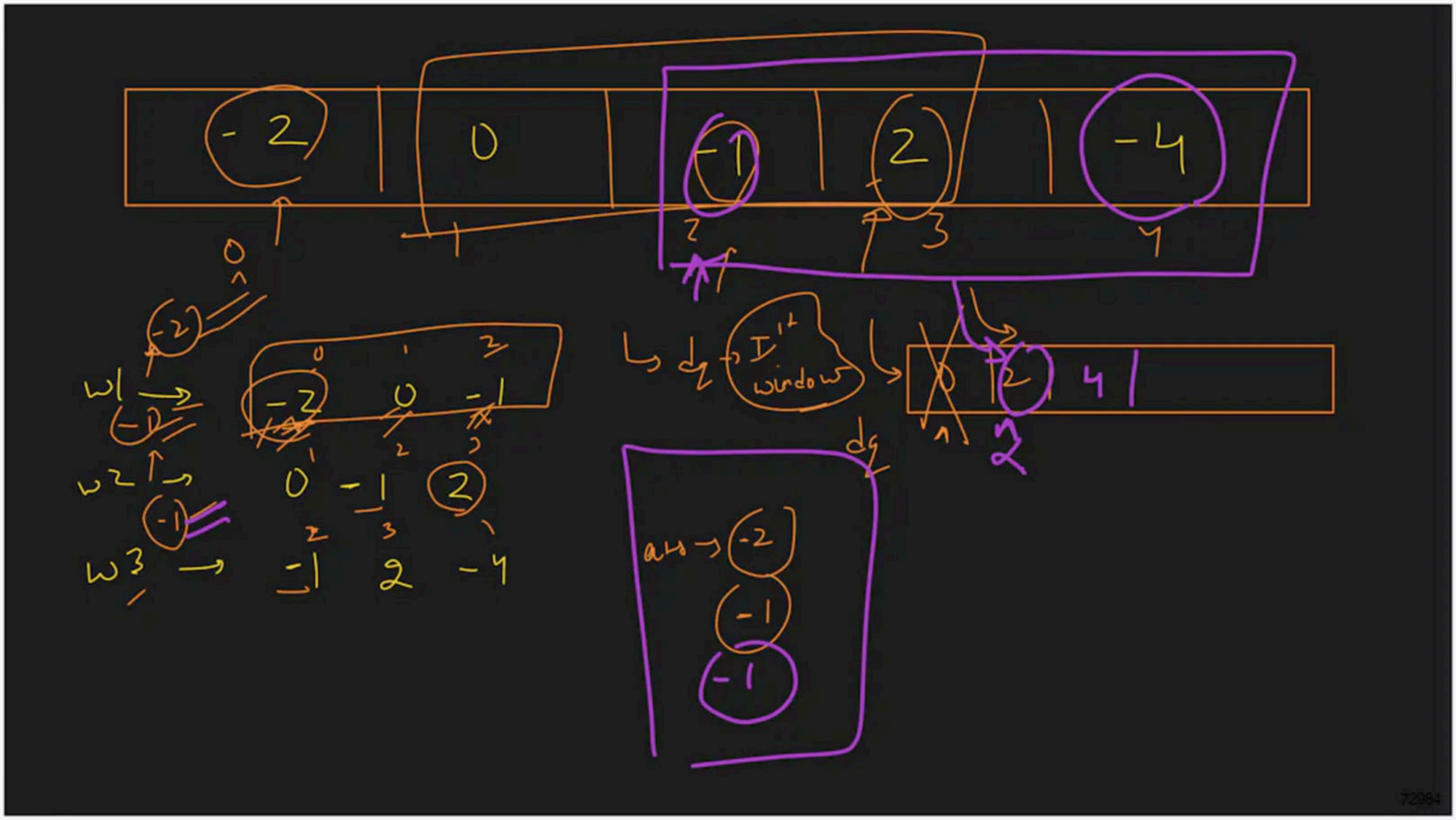
index - q.front >= K

3 - 0 >= 3

index - q.fr == K

index - q.fr < K


index
-2
0
4
-1
-5
6
arr
0
1
2
3
4
5
old
dq front
w1
w2
index - dq.front >= K
out of range
K = 3
pop
3-3=0
3-2=1
3-1=2
w1 →
-2
0 4
w2
Run
0 4
-1
Add
3-0=3
3-(-1)=4
1 2 3
<=0 index
out of range


-2
0
-1
2
-4
0
1
2
3
4
w1
w2
w3
-2
-1
-1
0
-1
2
-1
2
-4
I'st window
dq
ans
-2
-1
-1
72984