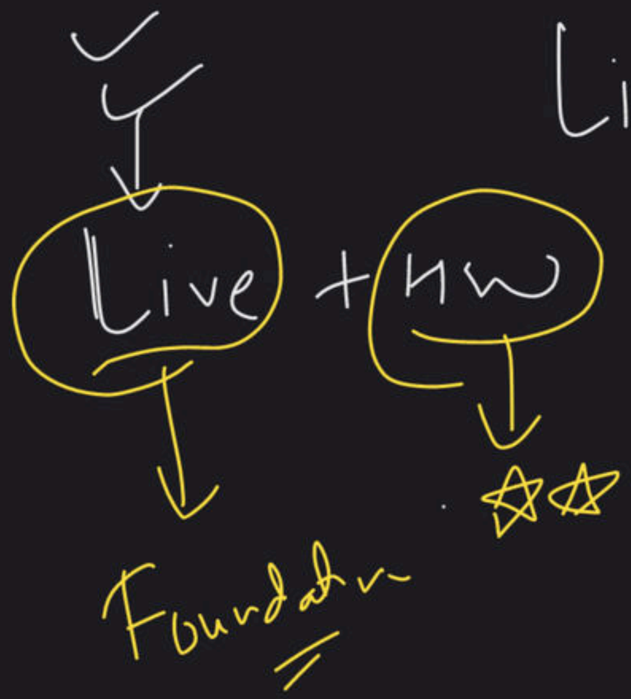
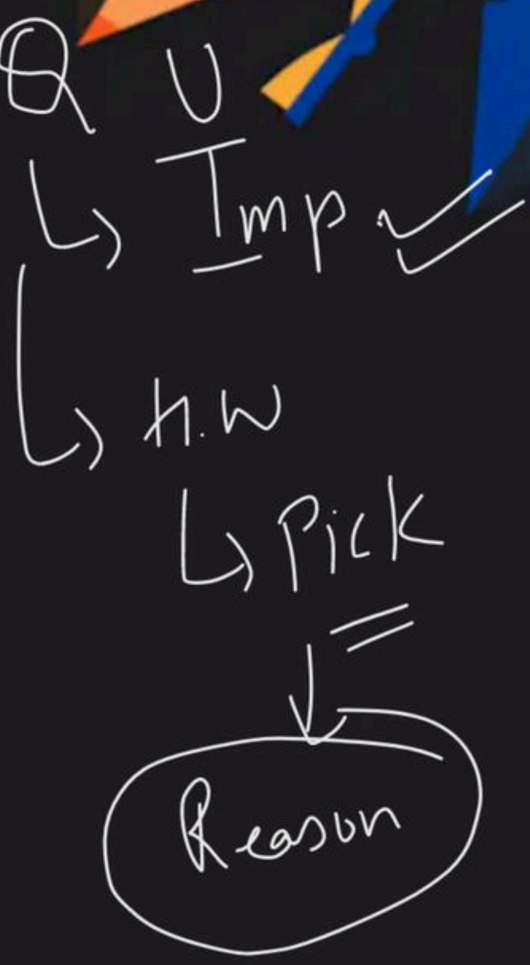
Stack Doubt Class by Lakshay Bhaiya

Special class

Love Babbar • Nov 18, 2023





Stack Class - 1

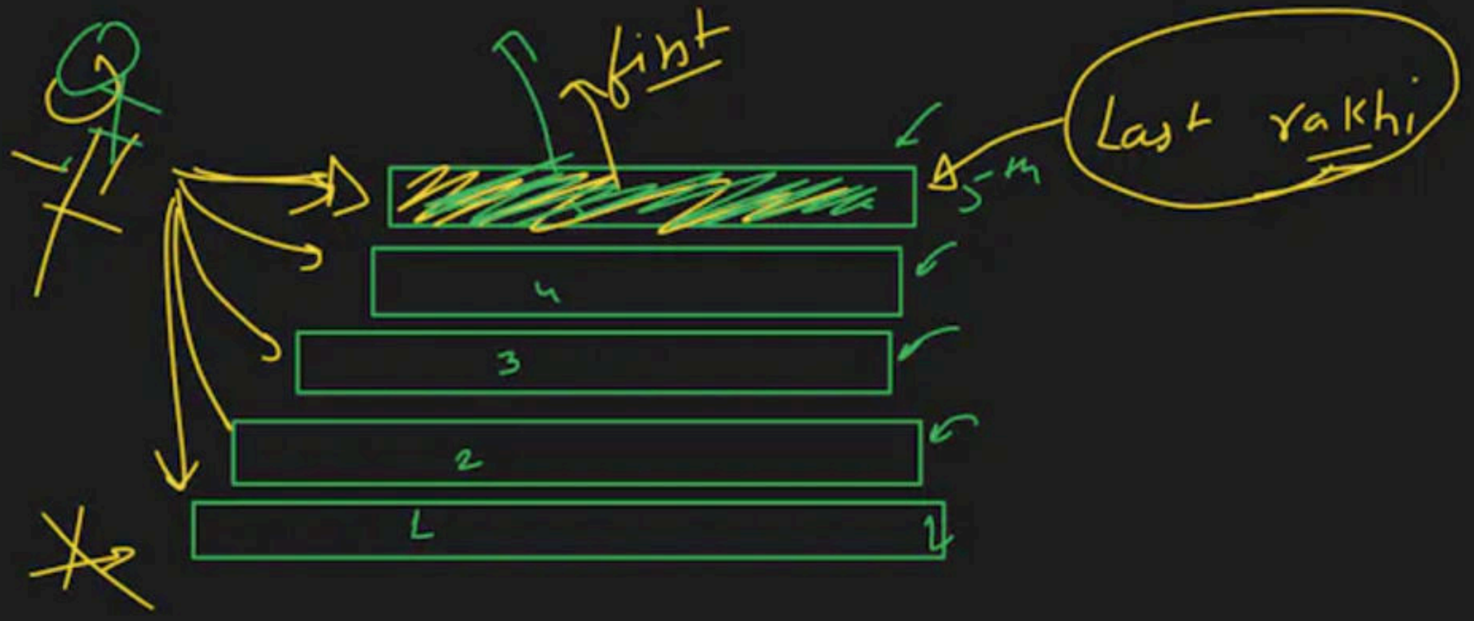
Special class

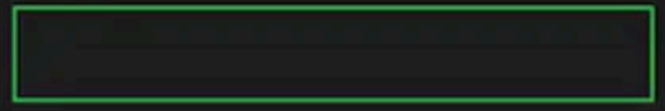
Love Babbar • Nov 8, 2023

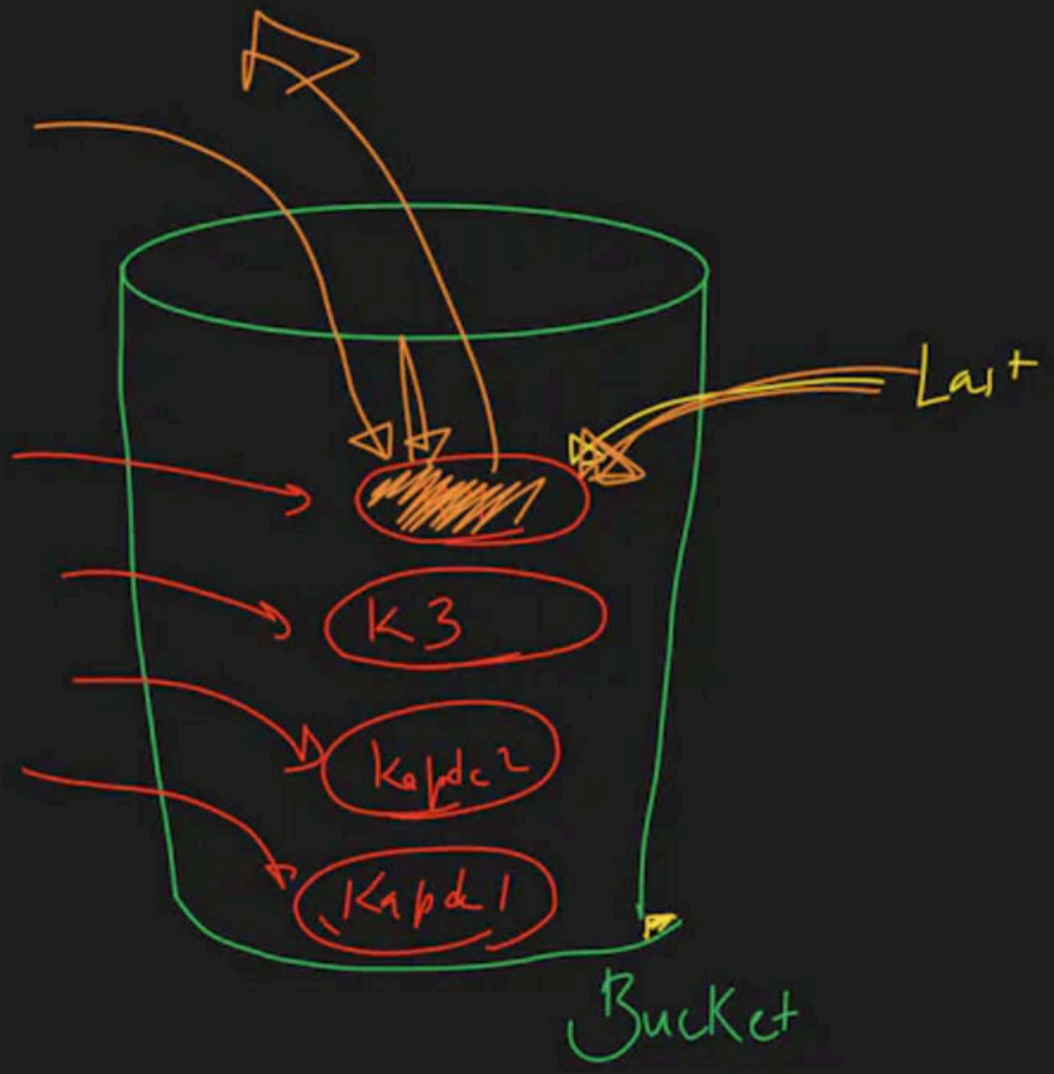
→ Stack: → Data Structure

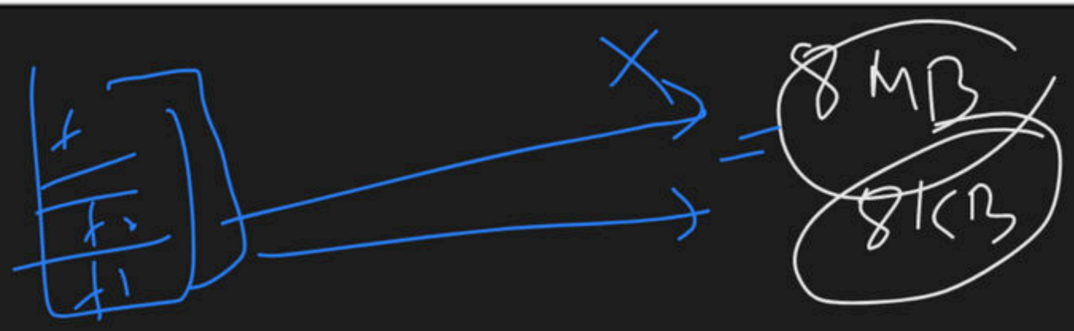
→ LIFO

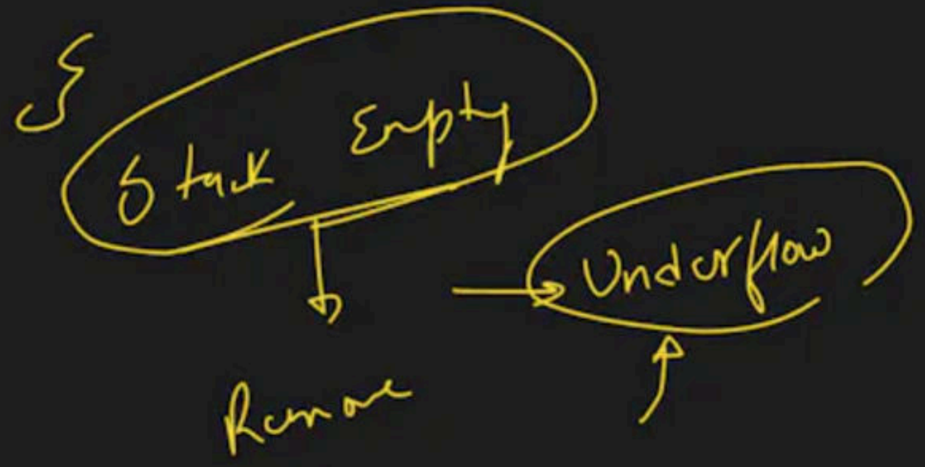
(Last in First Out)

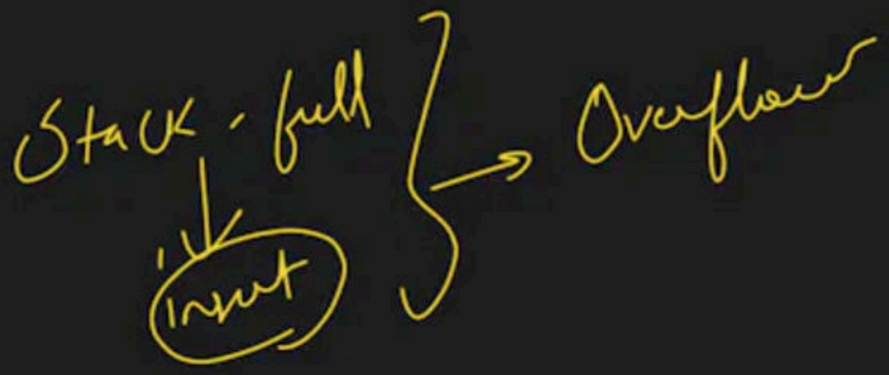


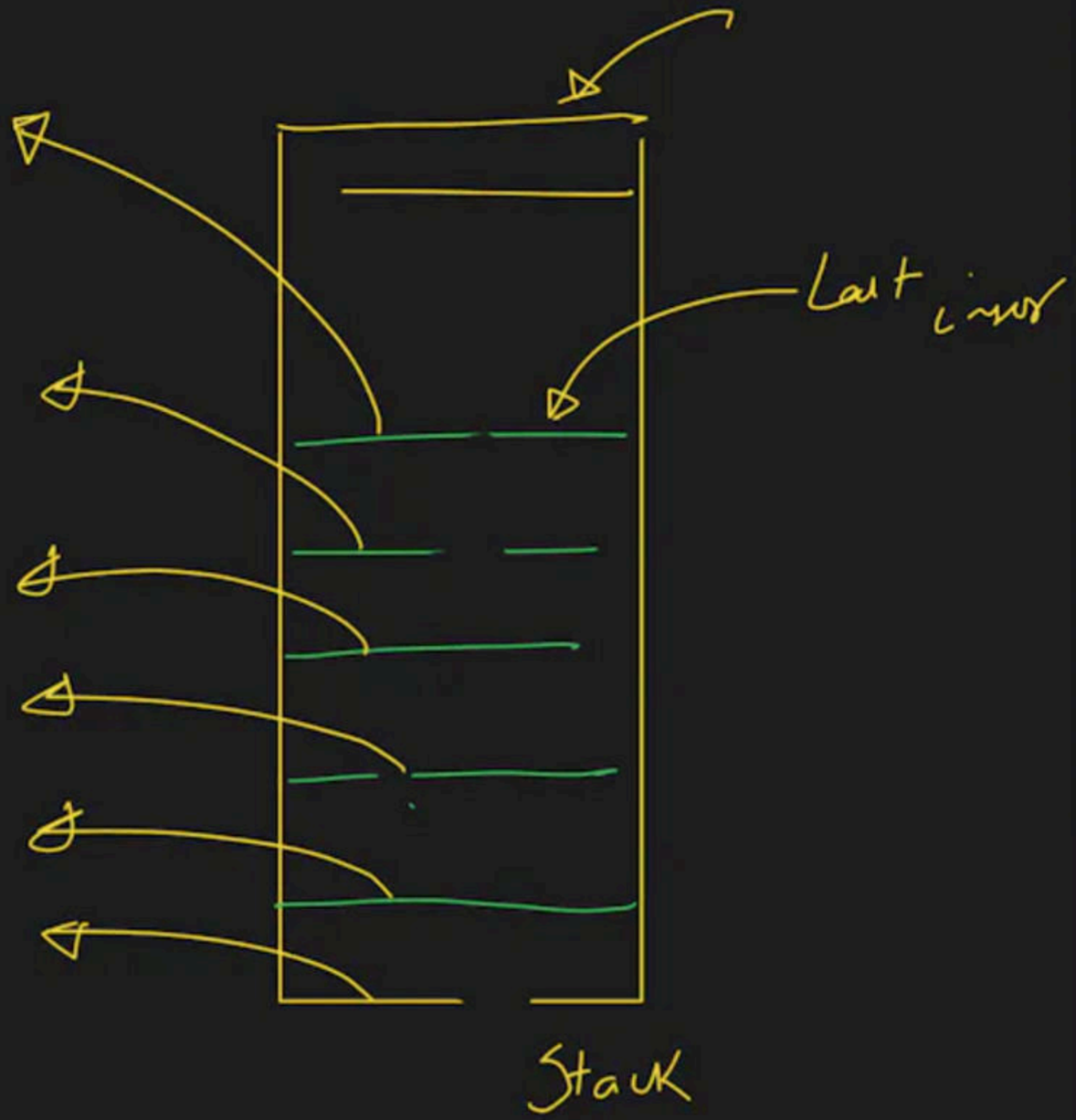



Last
K3
Kafka2
Kafka1
Bucket


8 MB
8 KB


Stack Empty
Underflow
Remove


Stack - full
Insert
Overflow

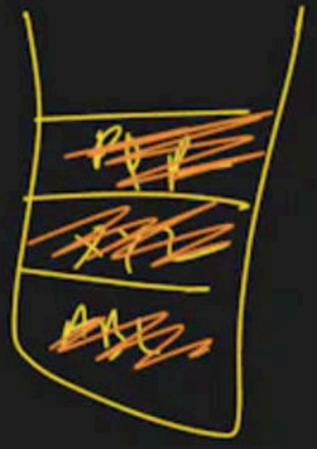

Last item
Stack

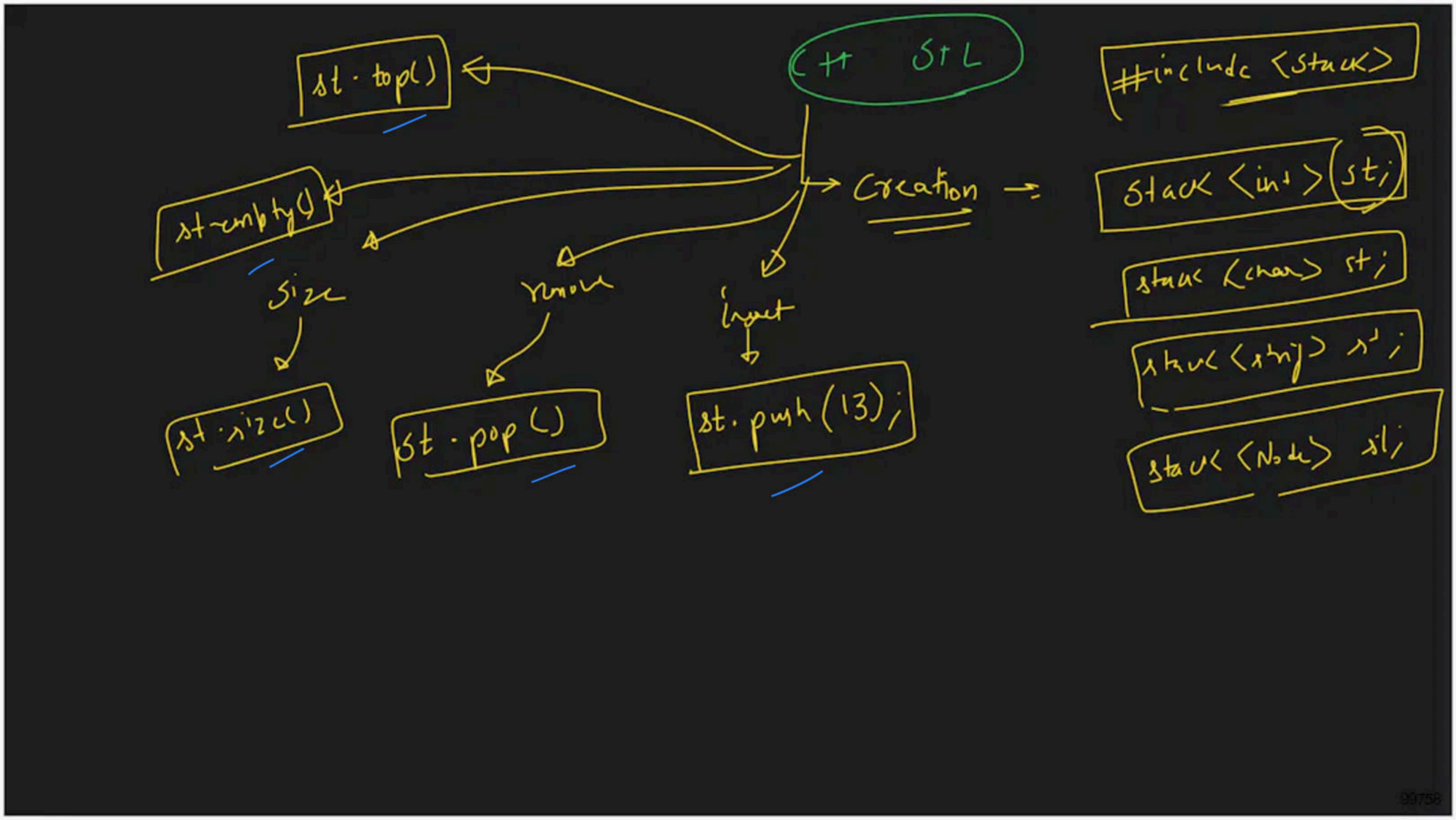
MS word

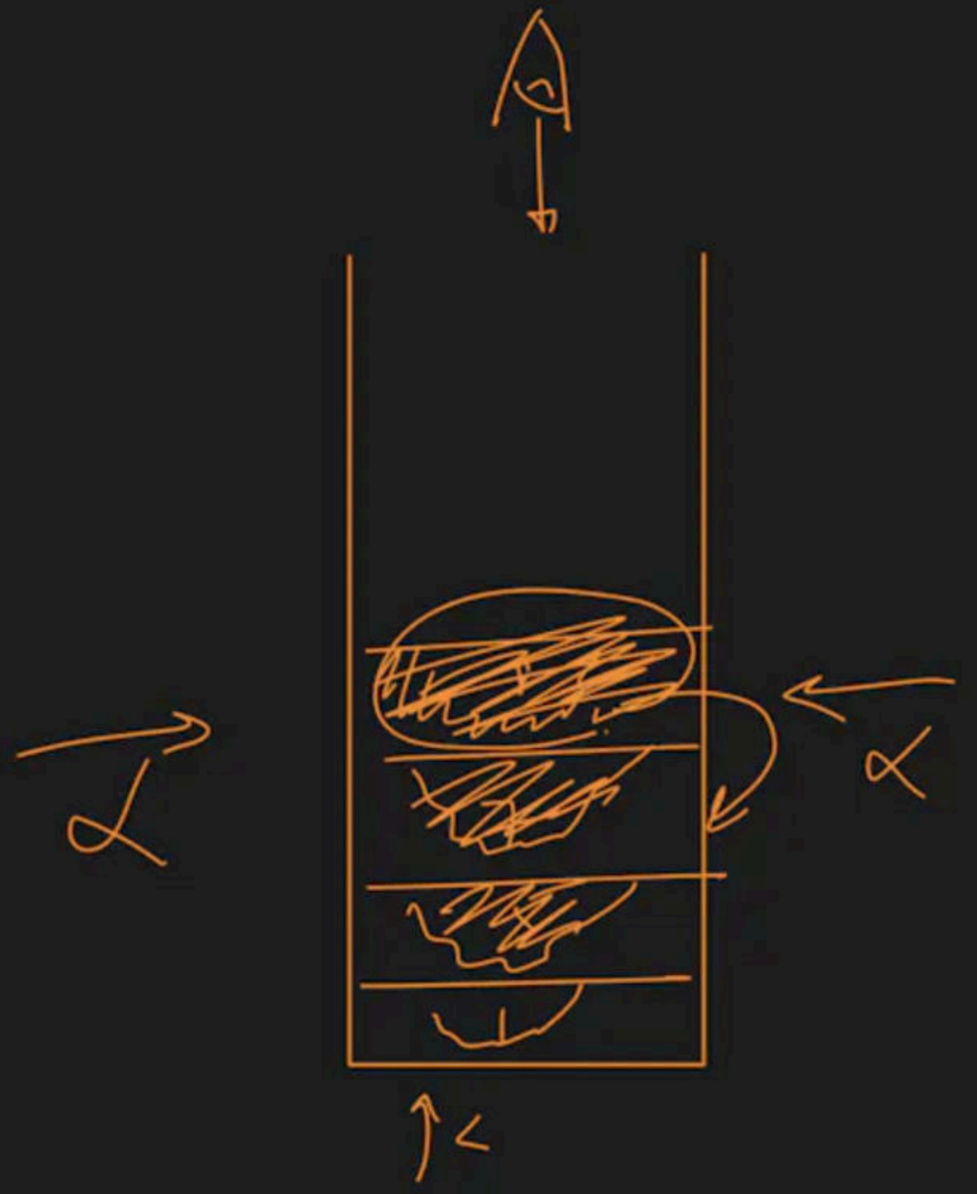
- Abcd
- XYZ
- POR

Ctrl + Z

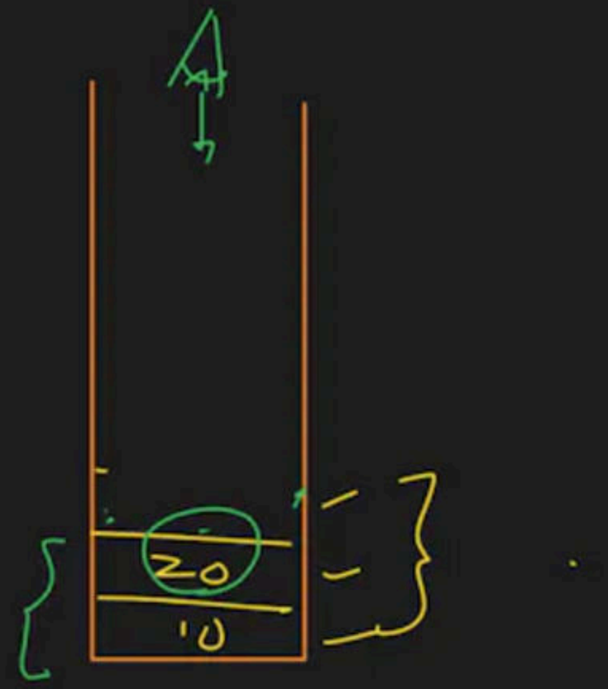
- Undo



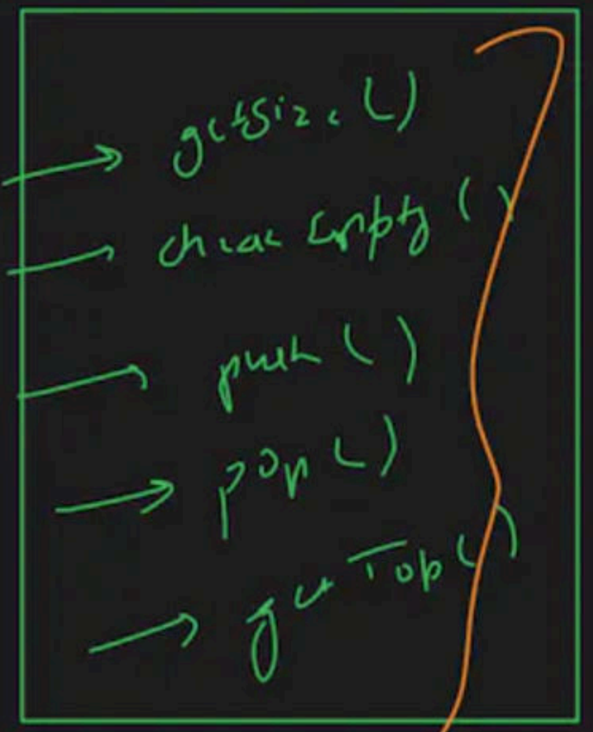

st.top()
C++ STL
#include <stack>
st.empty()
Creation
stack <int> st;
size
remove
input
stack <char> st;
st.size()
st.pop()
st.push(13);
stack <string> st;
stack <Node> st;



```
stack<int> st;

st.push(10);
st.push(20);
st.push(30)

cout << st.size();   → 3

st.pop()

if (st.empty())
    cout << Empty;
else
    cout << Not Empty;

注: cout << st.top();   → 20
```

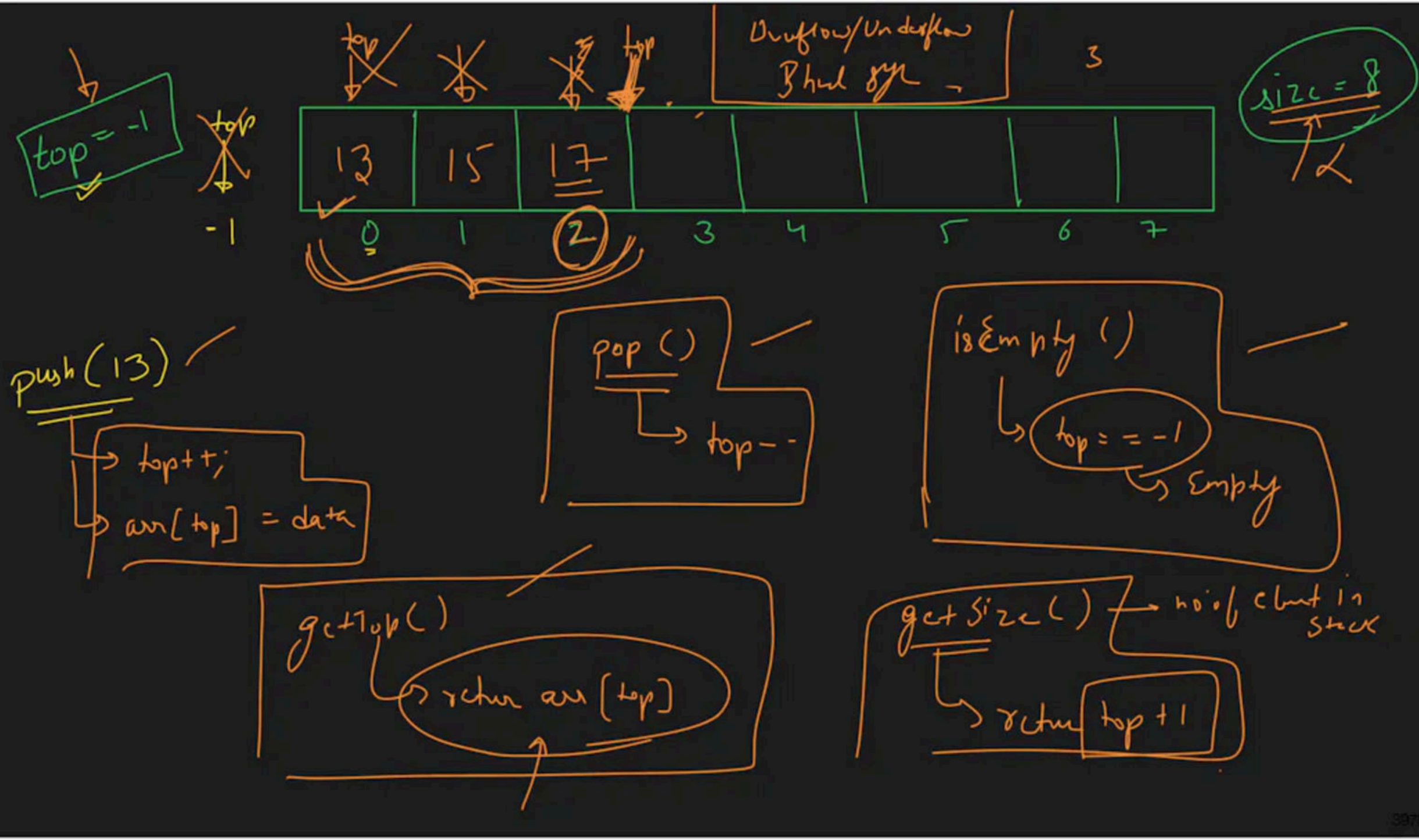


Stack → Implementation



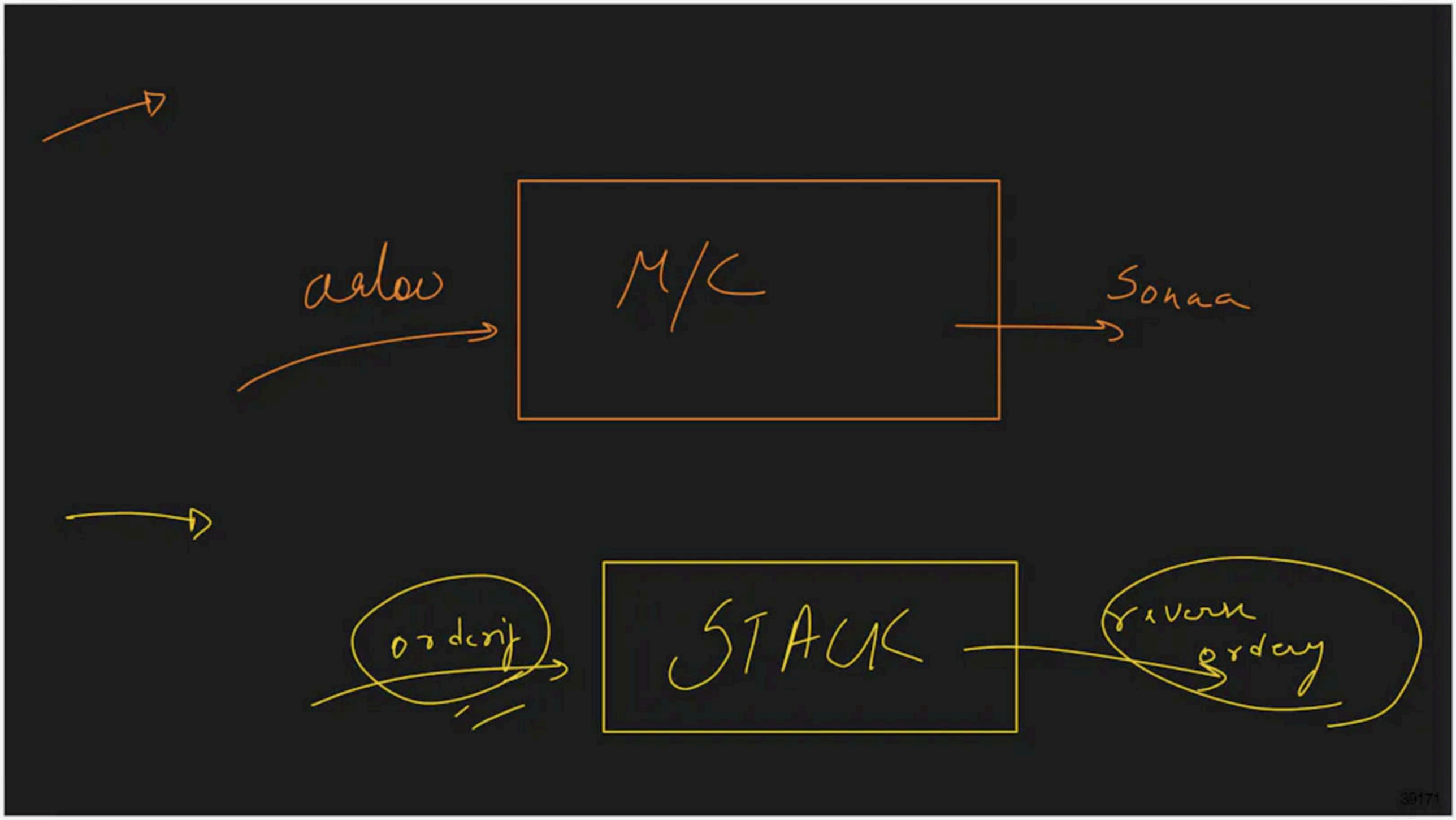
Stack → dynamic array

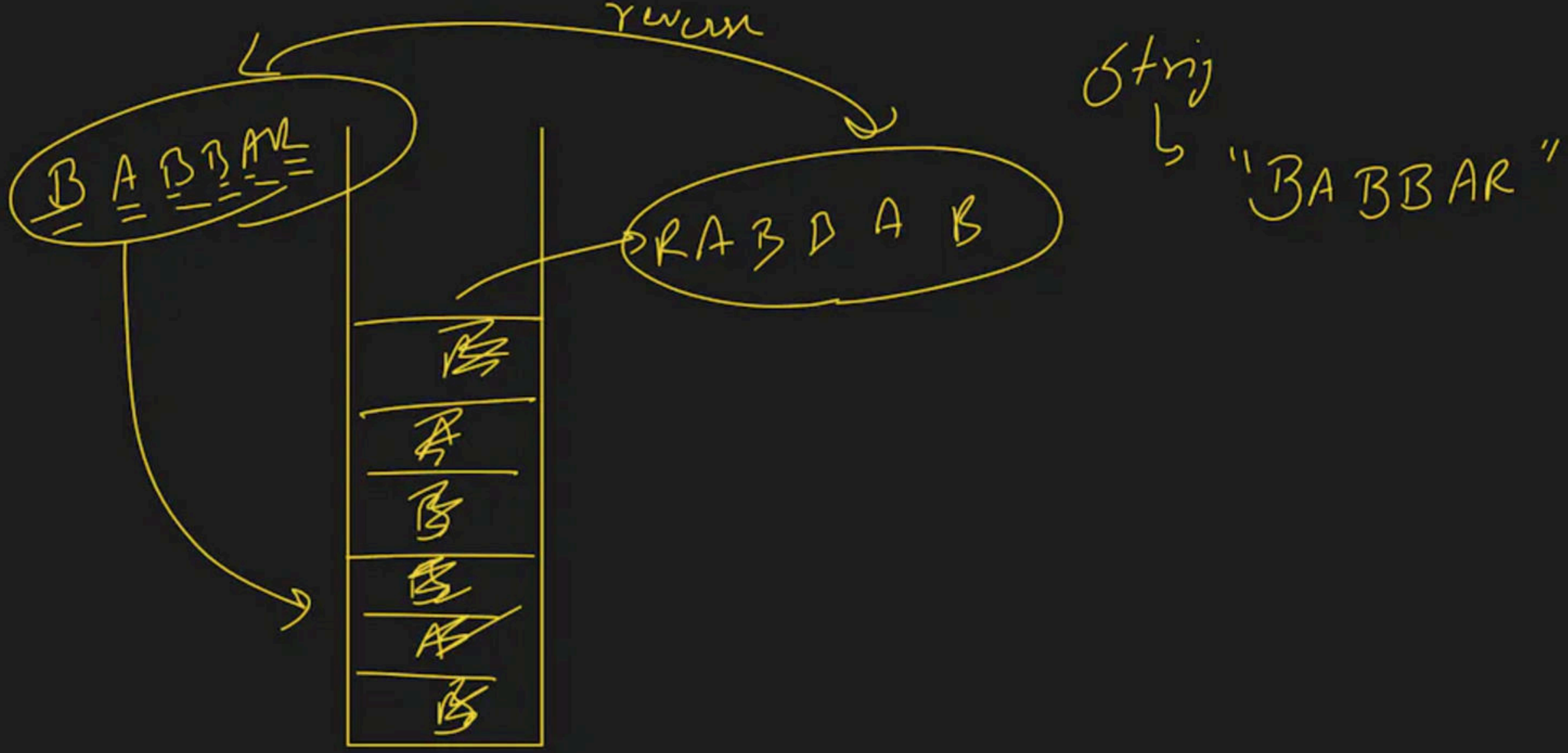
⇒ Kismat kharab = "N stacks in an Array"


top = -1
-1
top
top
top
Overflow/Underflow
Bhai sun
3
size = 8
13
15
17
0
1
2
3
4
5
6
7
push(13)
top++;
arr[top] = data
pop()
top--
isEmpty()
top == -1
Empty
getTop()
return arr[top]
getSize()
no of elements in stack
return top+1

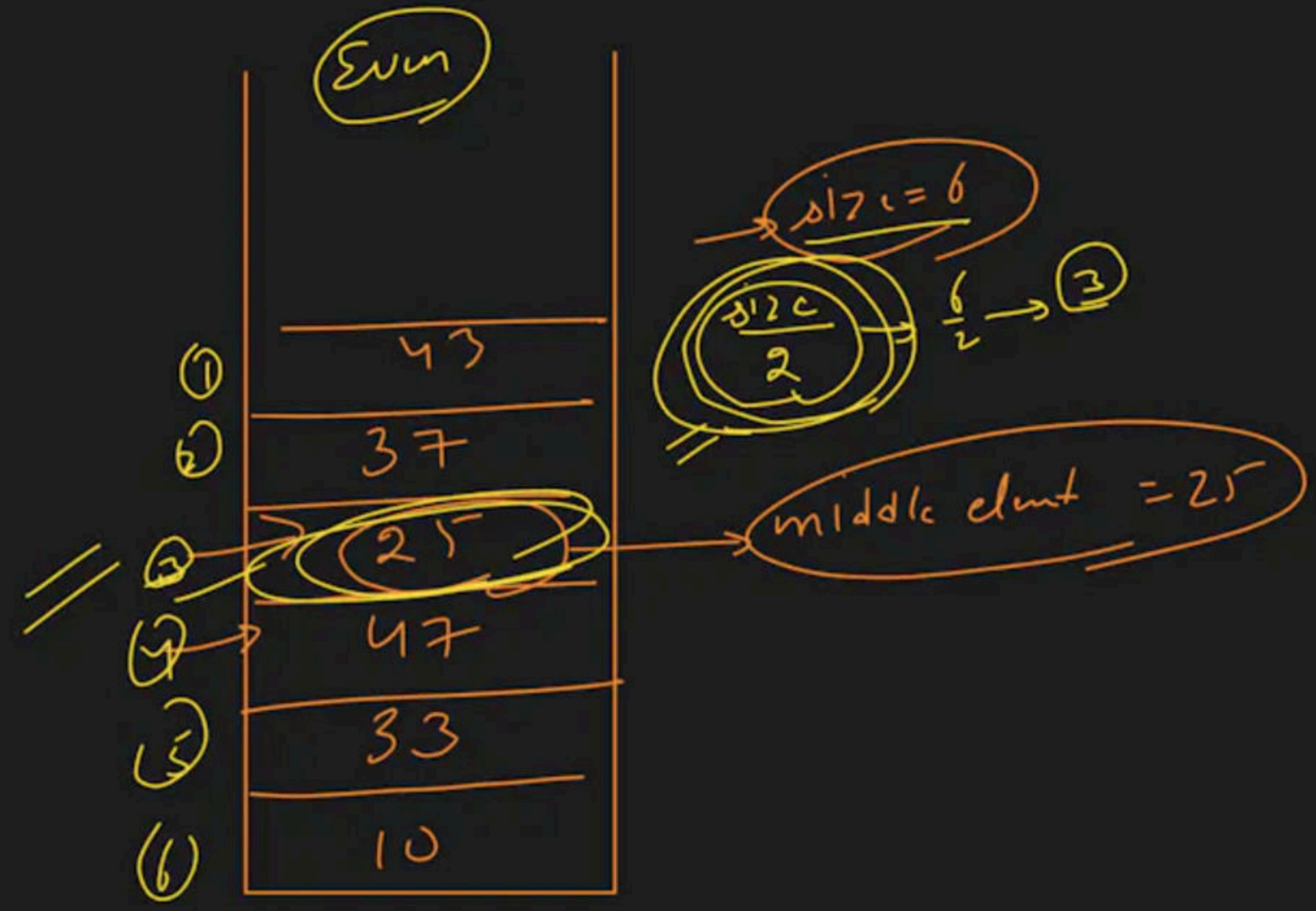
```
class Stack
{
    int *arr;
    int size;
    int top;

    Stack()

    getTop()
    pushL()
    pop()

    isEmpty()
    getSize()

};
```

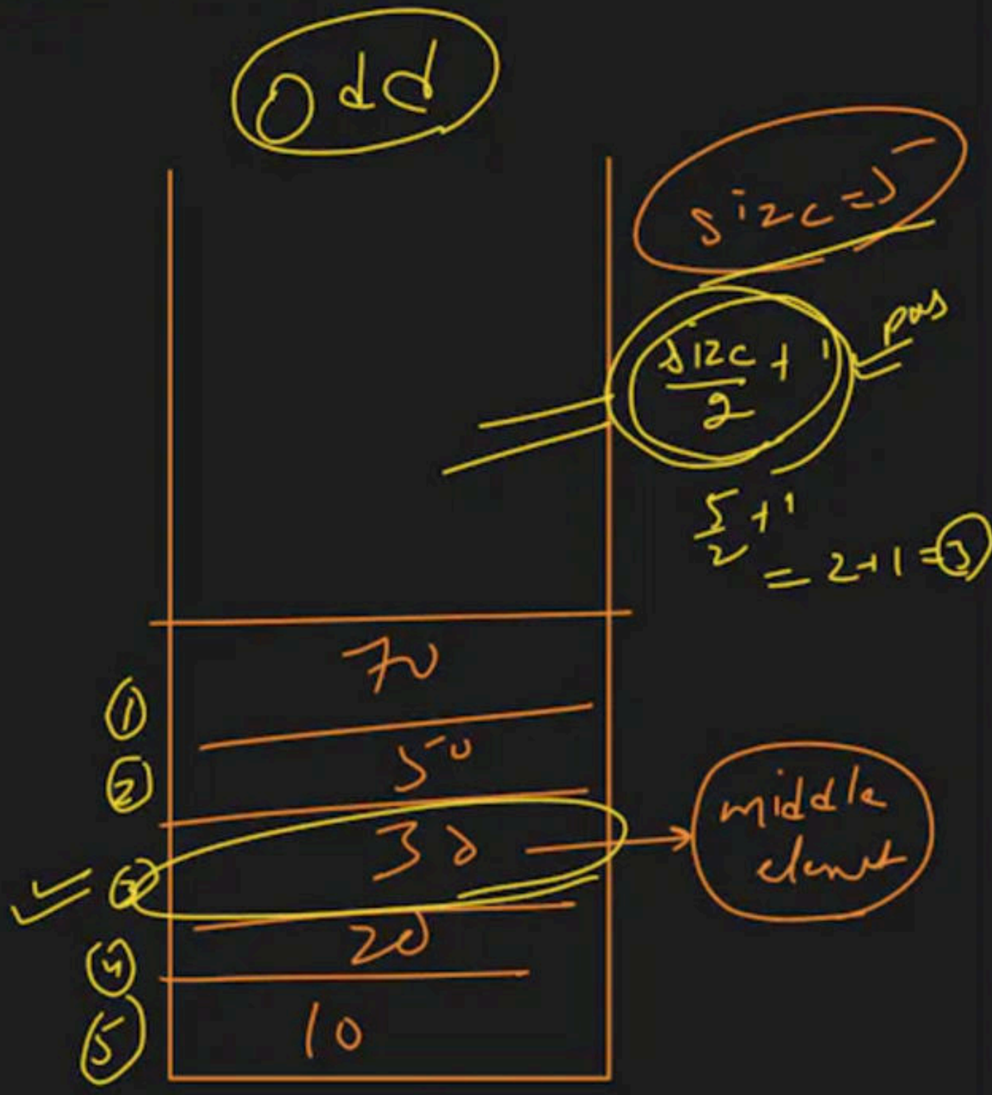
→ 1 Diwali

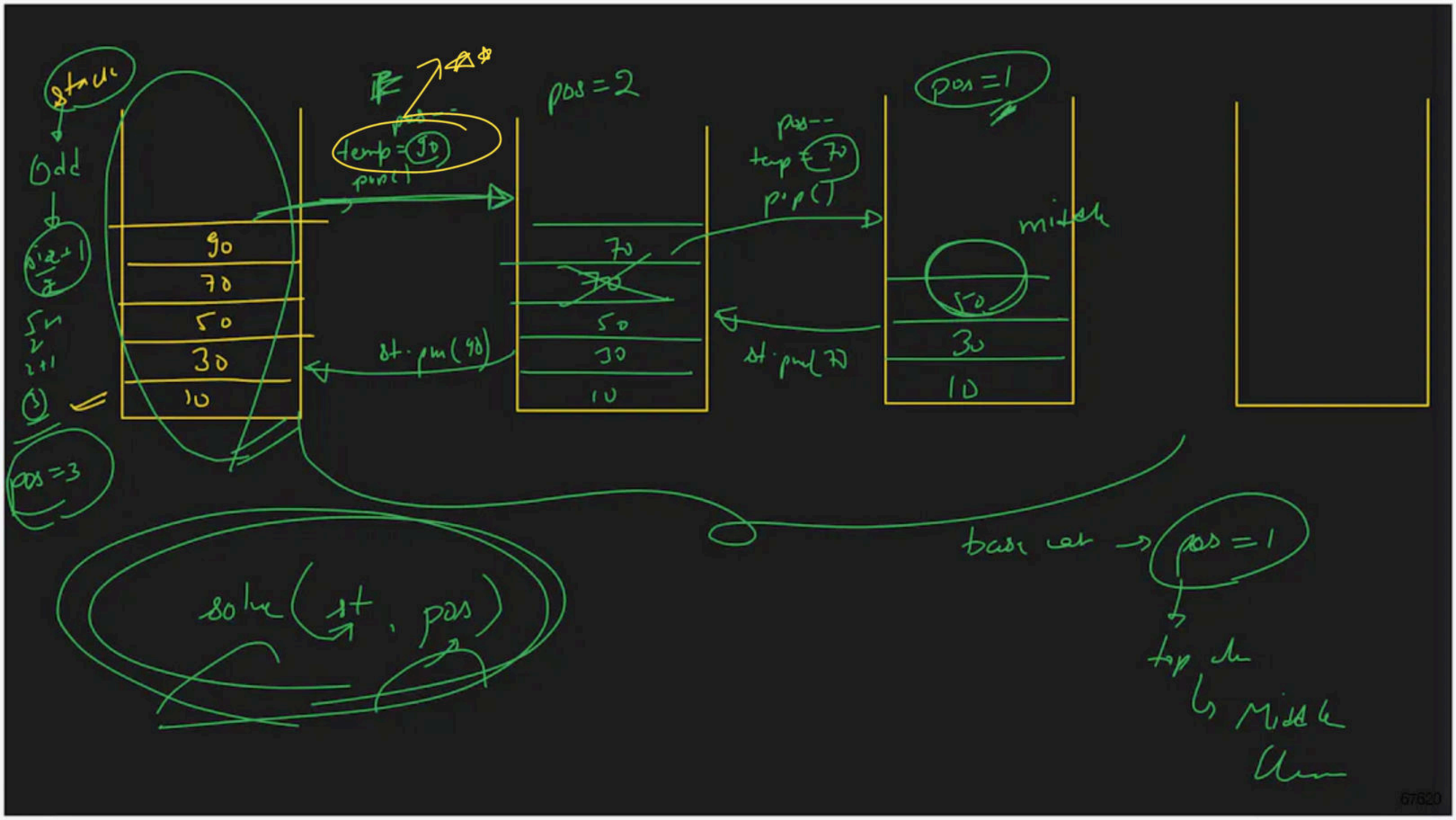

M/C
STACK

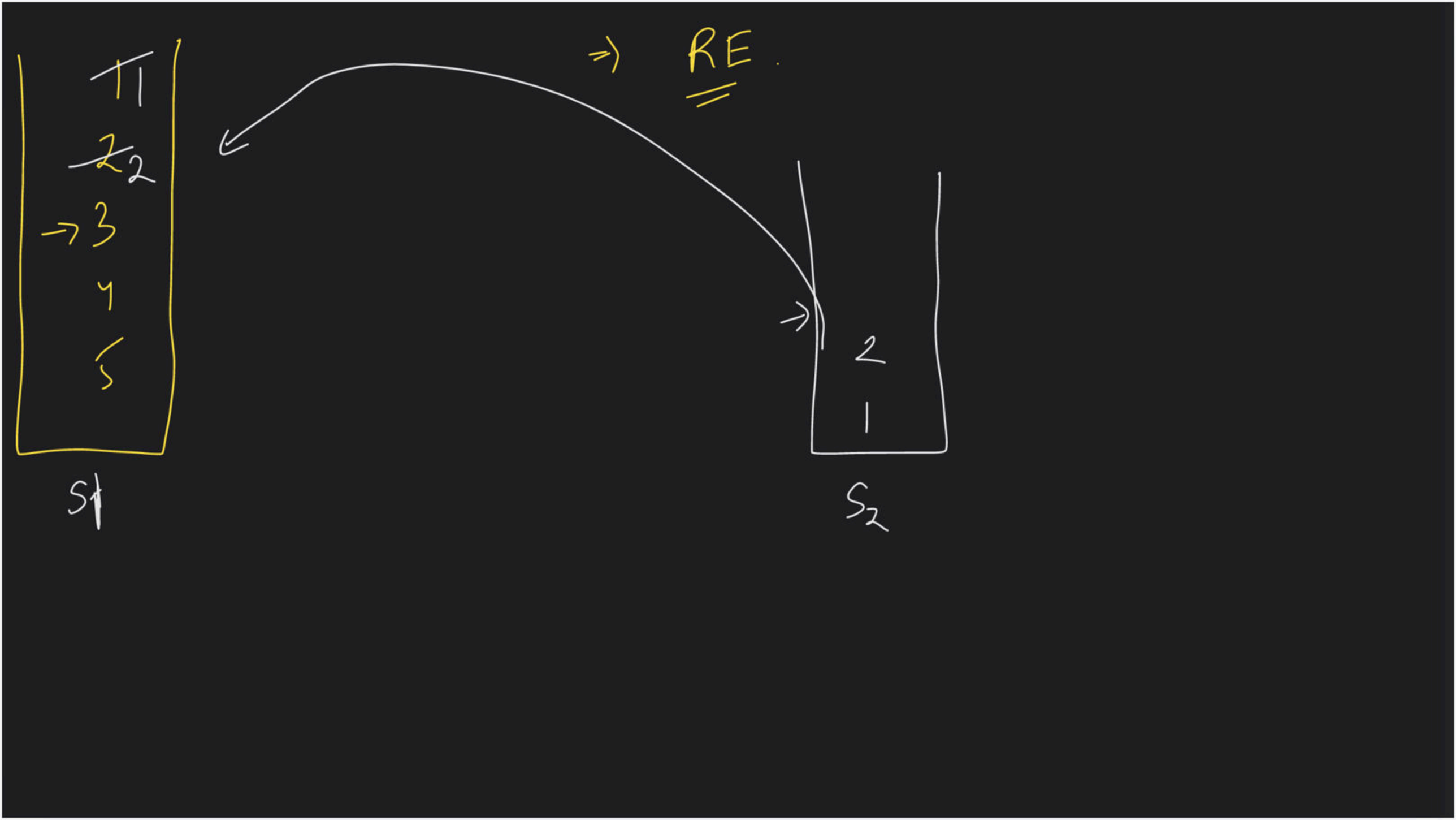

reverse
BABBAR
RABDAB
String
"BABBAR"

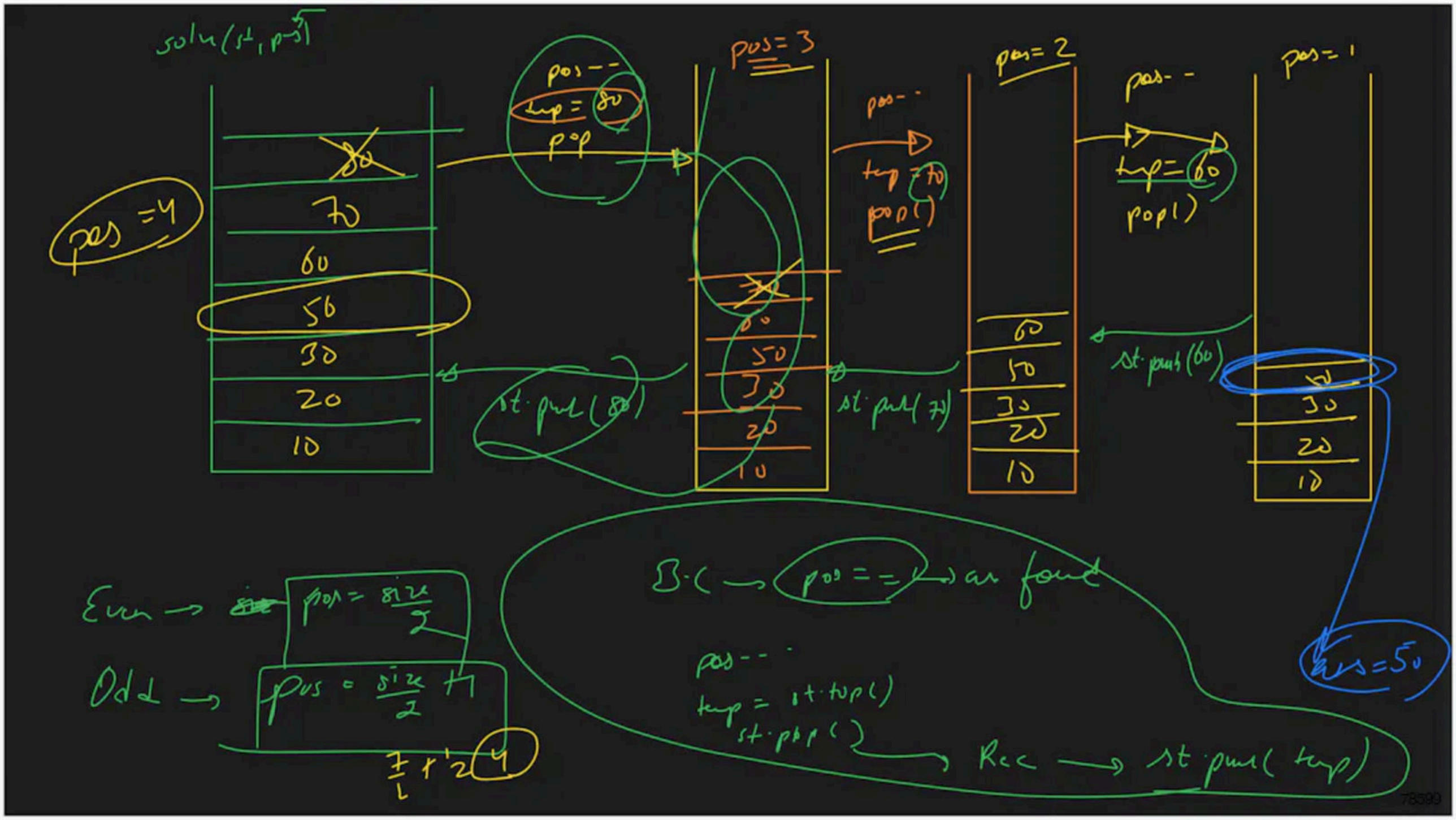
Middle Element of a Stack :-



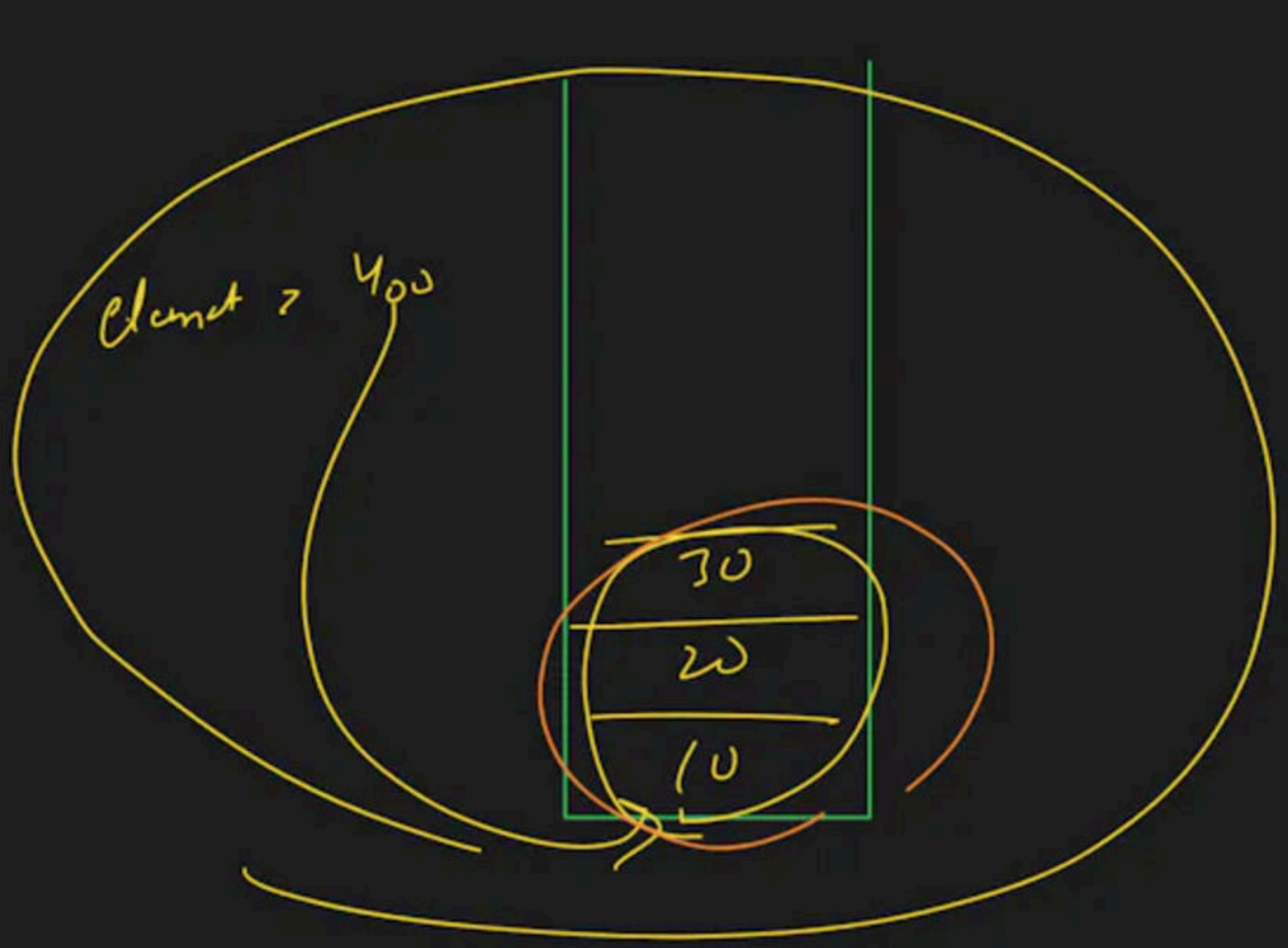


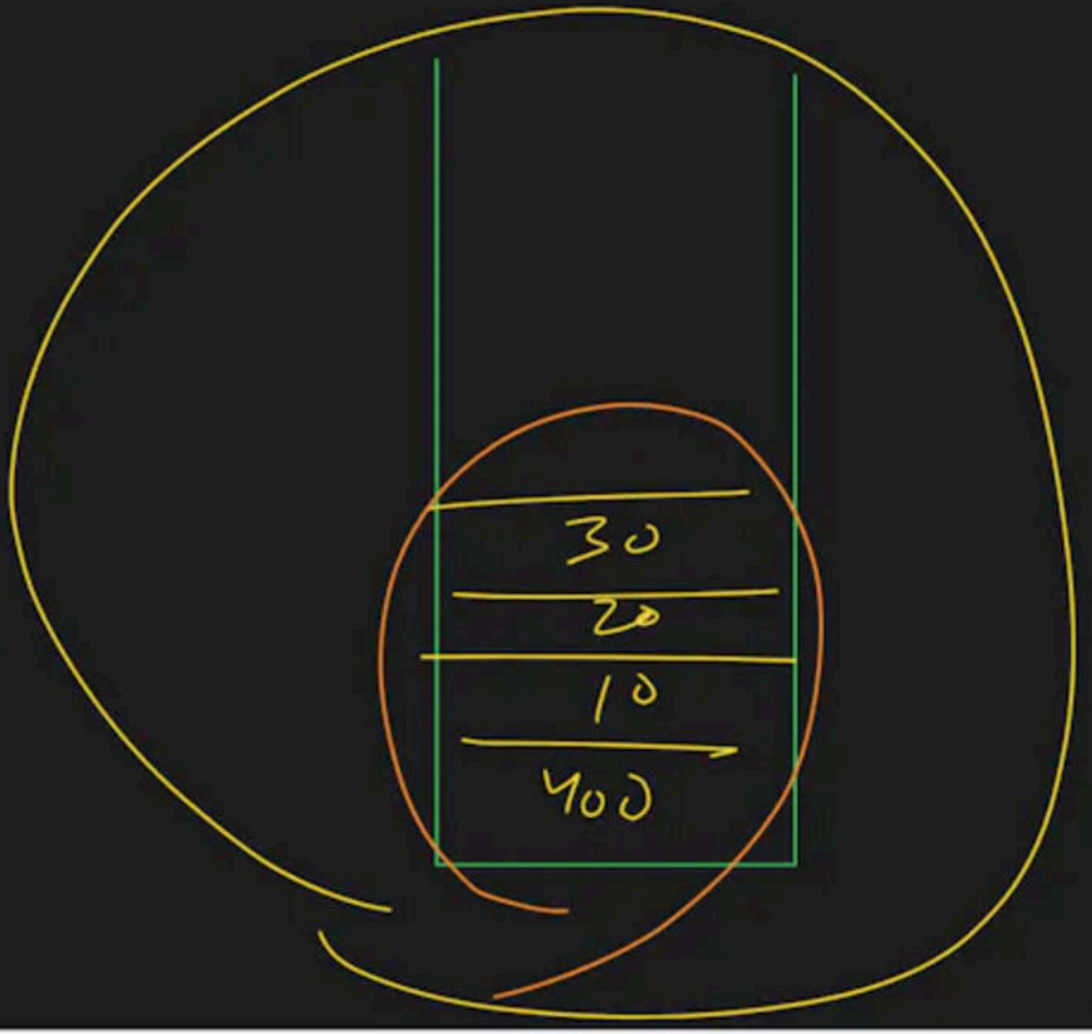

stack
Odd
size+1
2
pos=3
temp=90
pop()
pos=2
90
70
50
30
10
st.push(90)
70
50
30
10
pos--
temp=70
p.p()
pos=1
mitesh
50
30
10
st.push(70)
solve(st, pos)
base call
pos=1
top of
is Middle

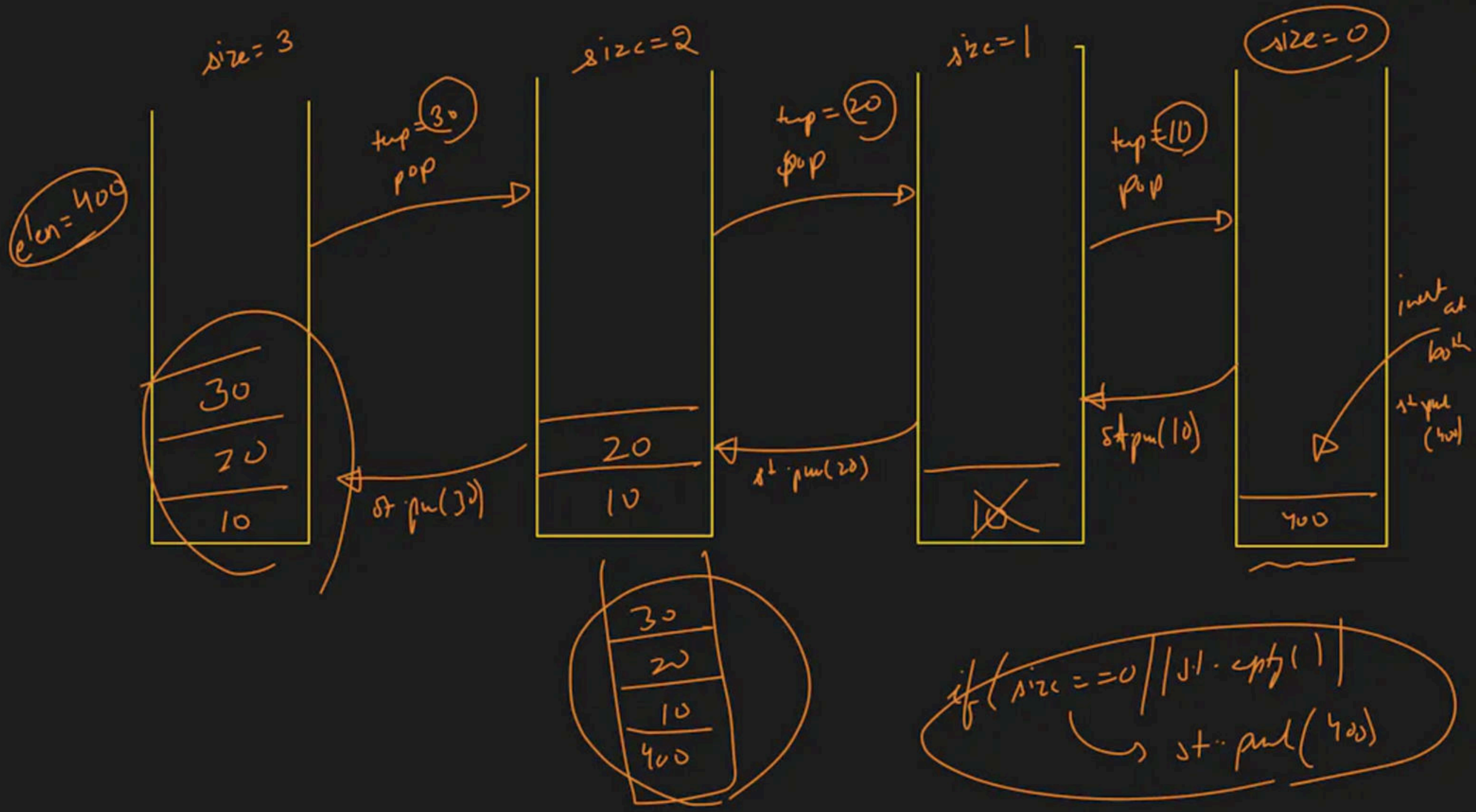

⇒ RE.
T1
~~2~~ 2
→3
4
5
S1
2
1
S2

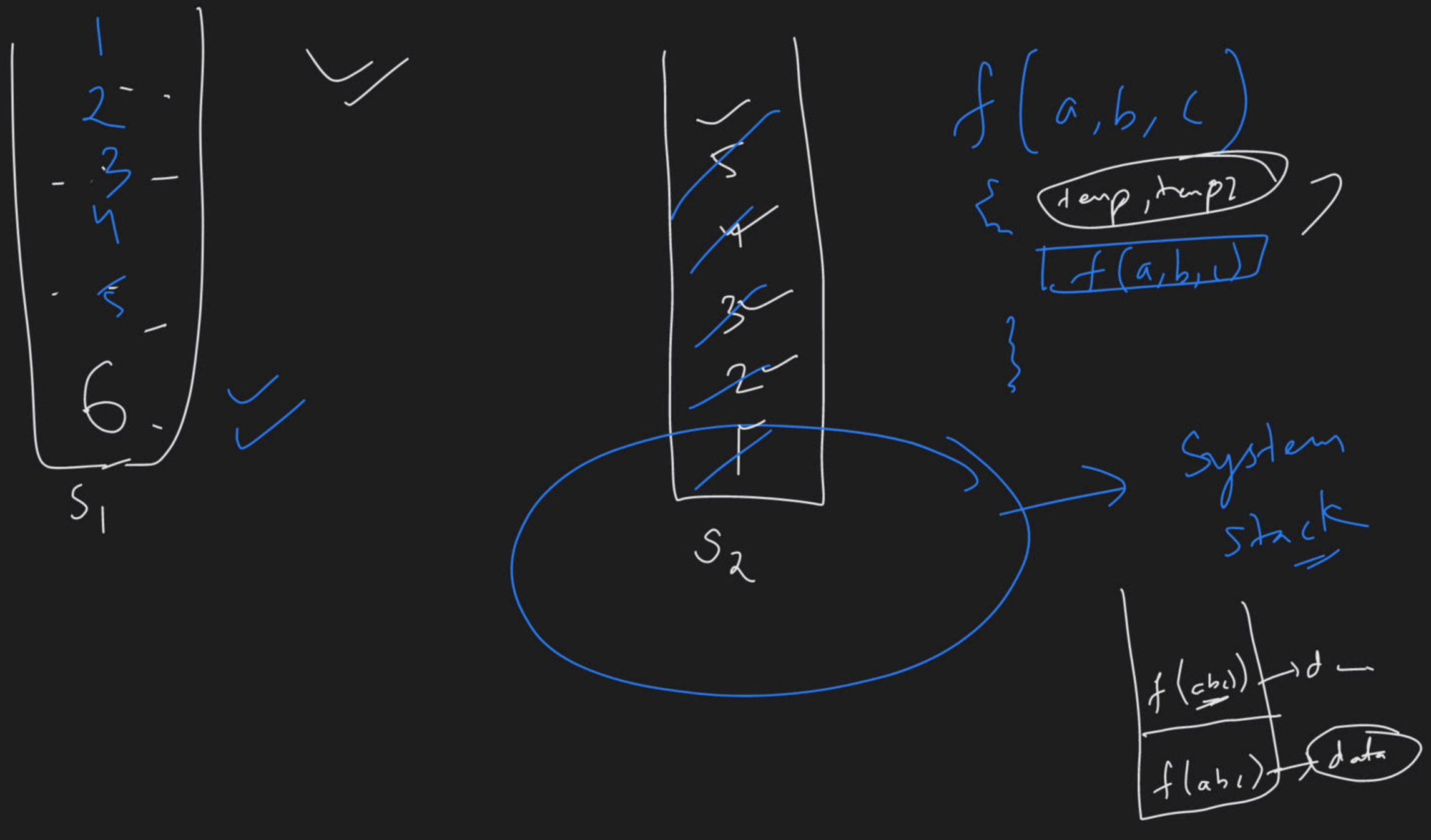

solve(st, pos)
pos = 4
80
70
60
50
30
20
10
pos--
temp = 80
pop
st.push(80)
pos = 3
50
30
20
10
pos--
temp = 70
pop()
st.push(70)
pos = 2
60
50
30
20
10
pos--
temp = 60
pop()
st.push(60)
pos = 1
30
20
10
Even → pos = size/2
Odd → pos = size/2 + 1
7/2 + 1 = 4
B.C → pos == 1 → ans found
pos--
temp = st.top()
st.pop()
Rec → st.push(temp)
ans = 50

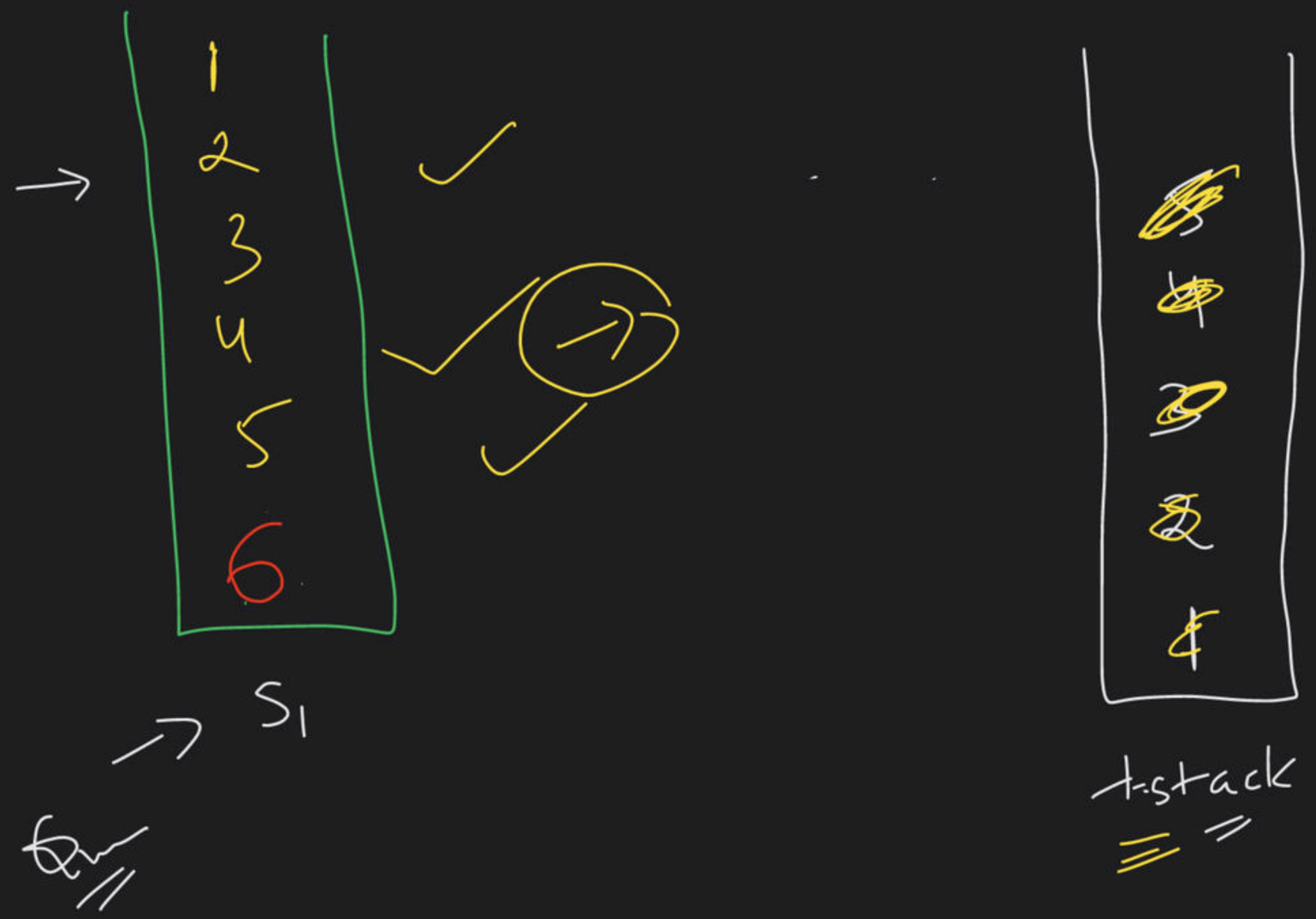
Insert at bottom of stack



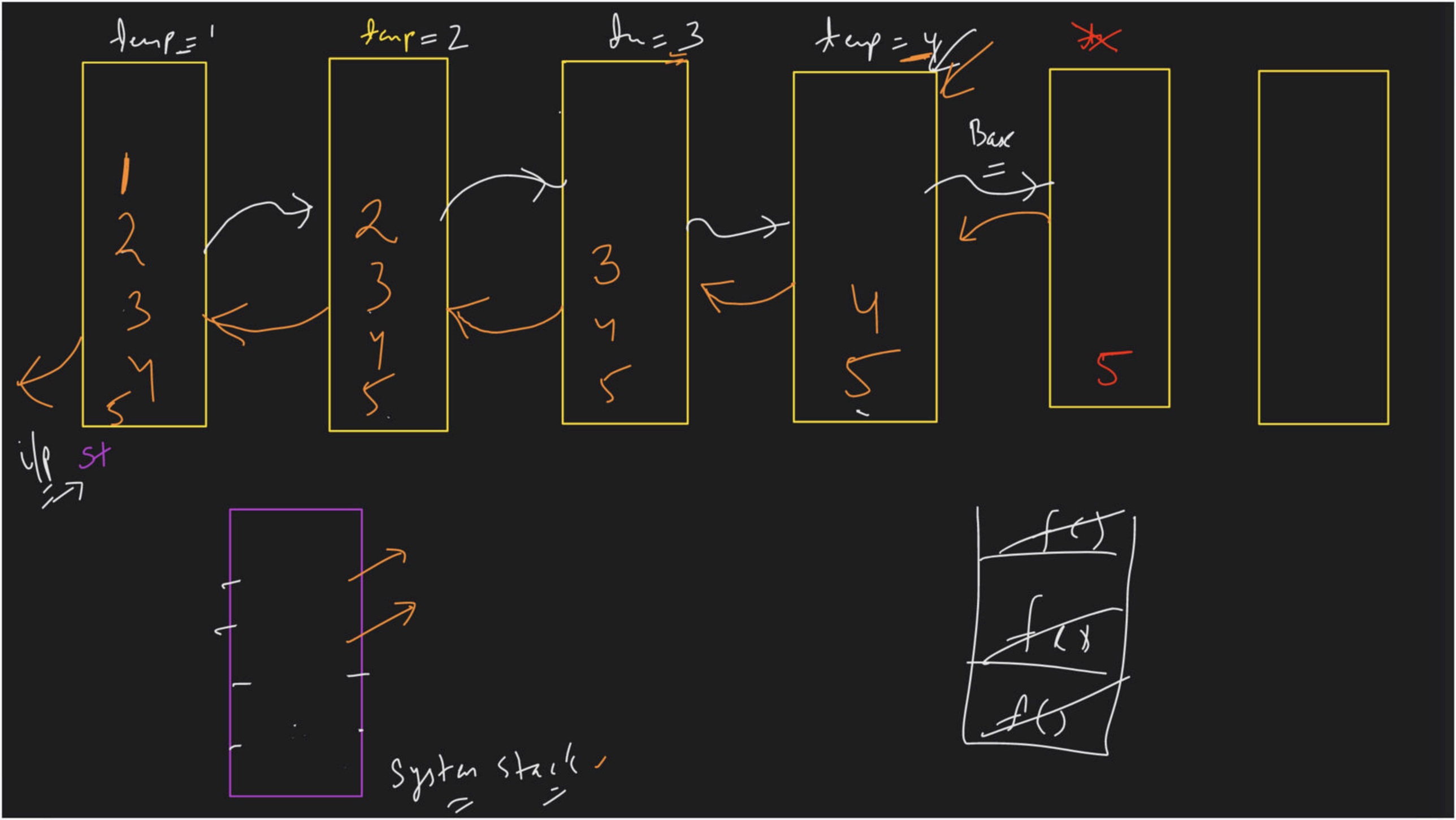



size = 3
size = 2
size = 1
size = 0
elem = 400
top = 30
pop
top = 20
pop
top = 10
pop
30
20
10
20
10
400
st.push(30)
st.push(20)
st.push(10)
insert at back
1st push (400)
if (size == 0 || st.empty())
st.push(400)

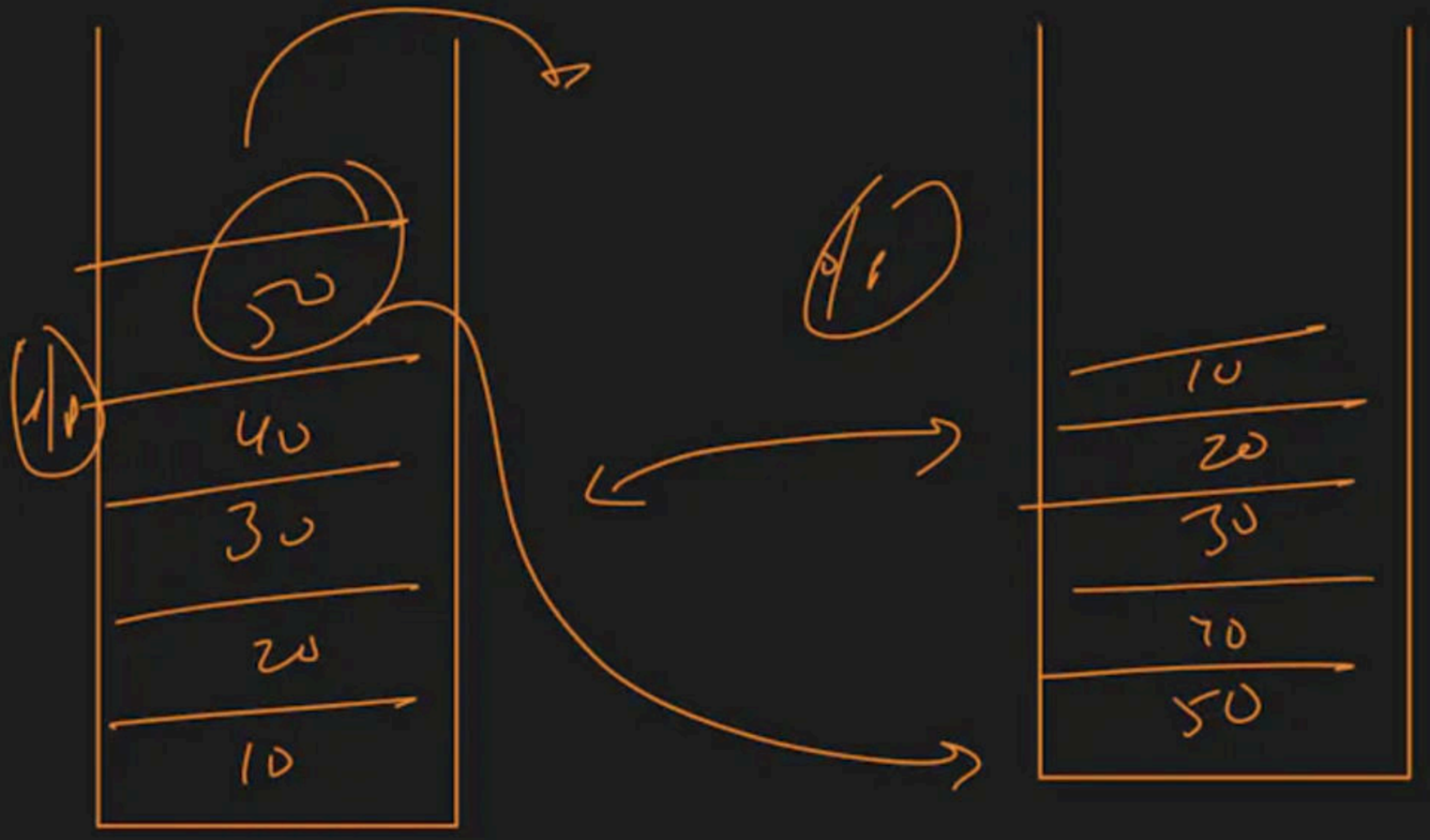

1
2
3
4
5
6
S_1
S_2
f(a,b,c)
{
temp, temp2
f(a,b,c)
System stack
f(cba)
f(abc)
data

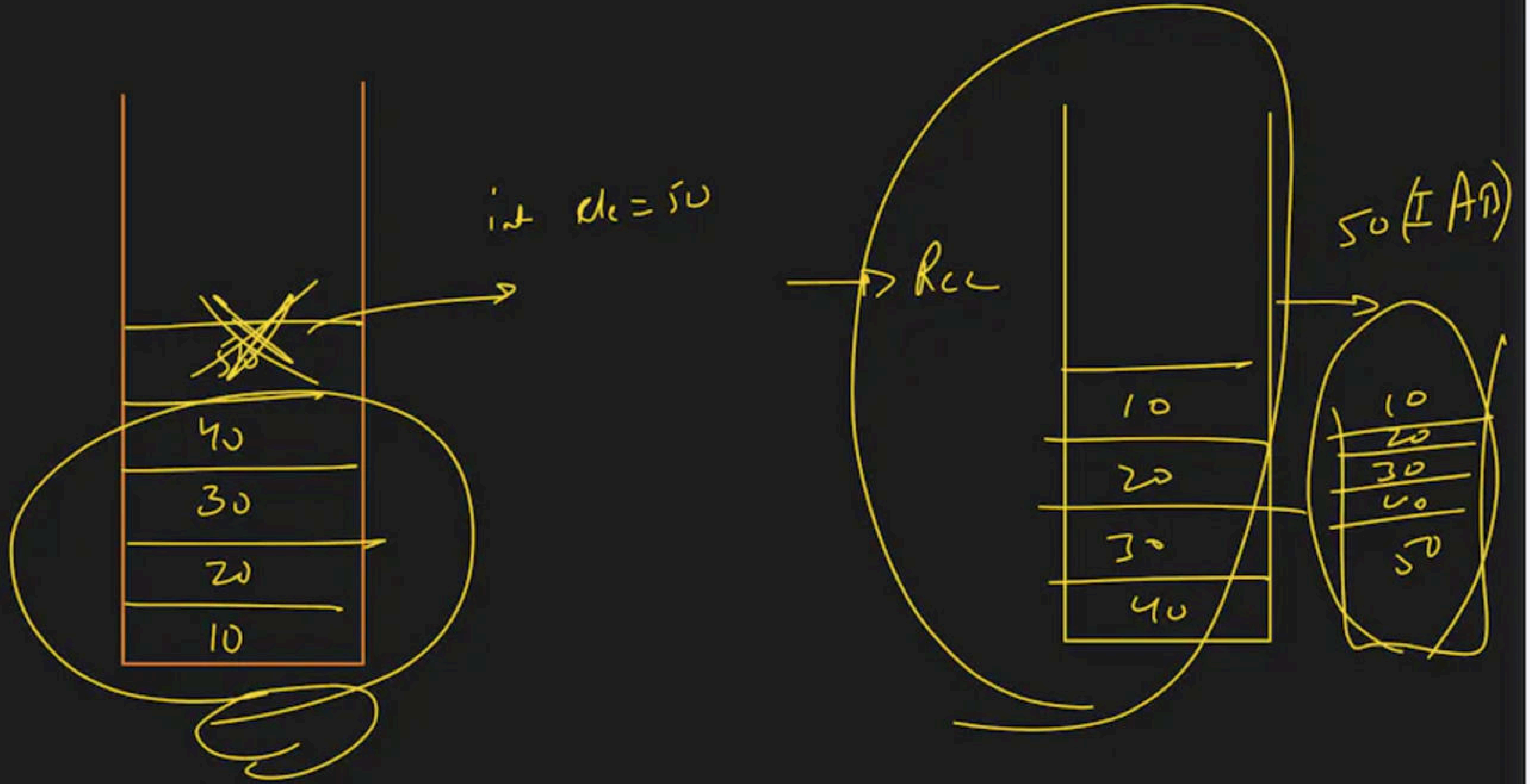


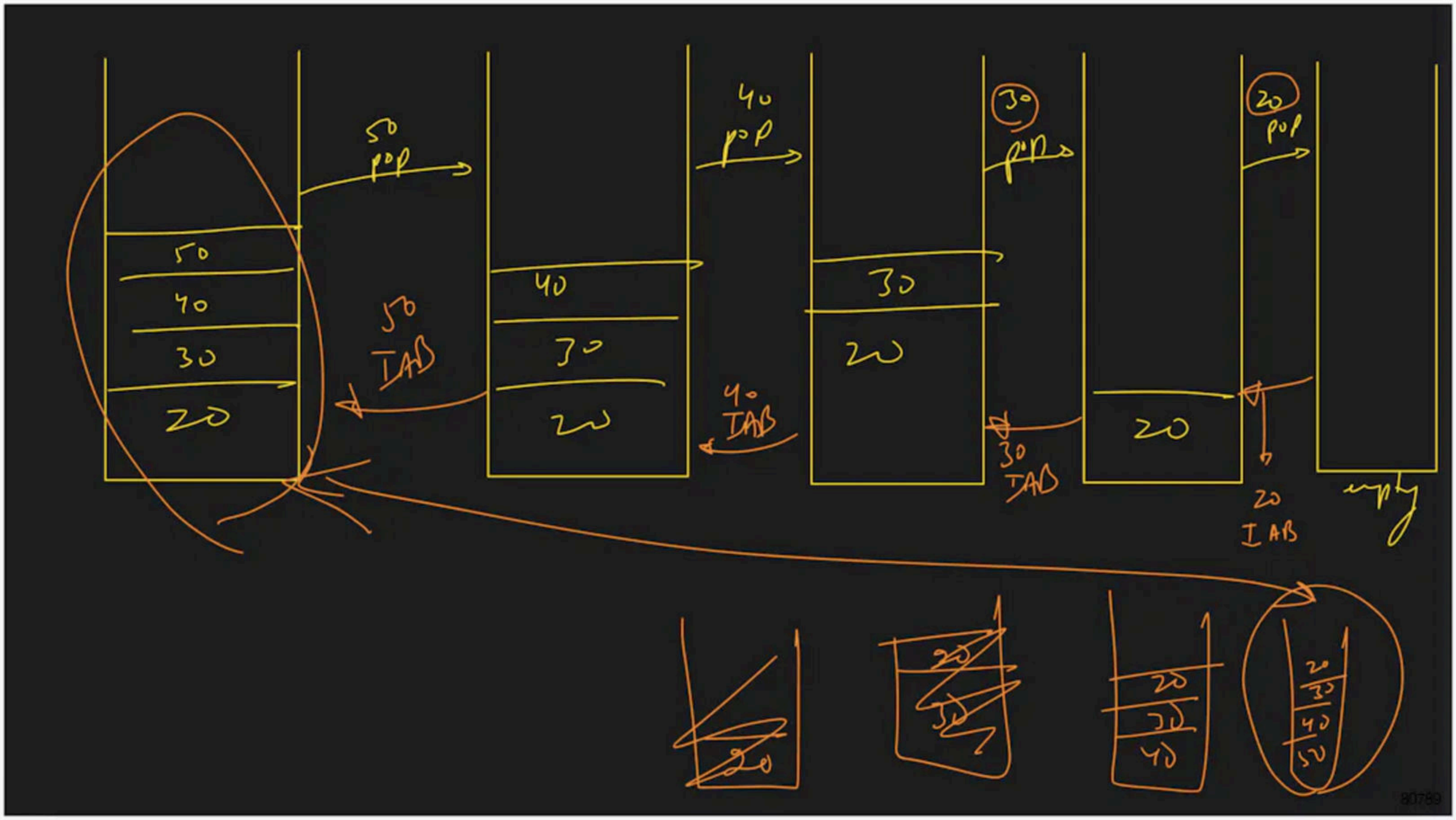
t.stack


temp = 1
temp = 2
fun = 3
temp = 4
Base
1
2
3
4
5
2
3
4
5
3
4
5
4
5
5
i/p
st
System stack
f()
f()
f()

Reverse a Stack #1

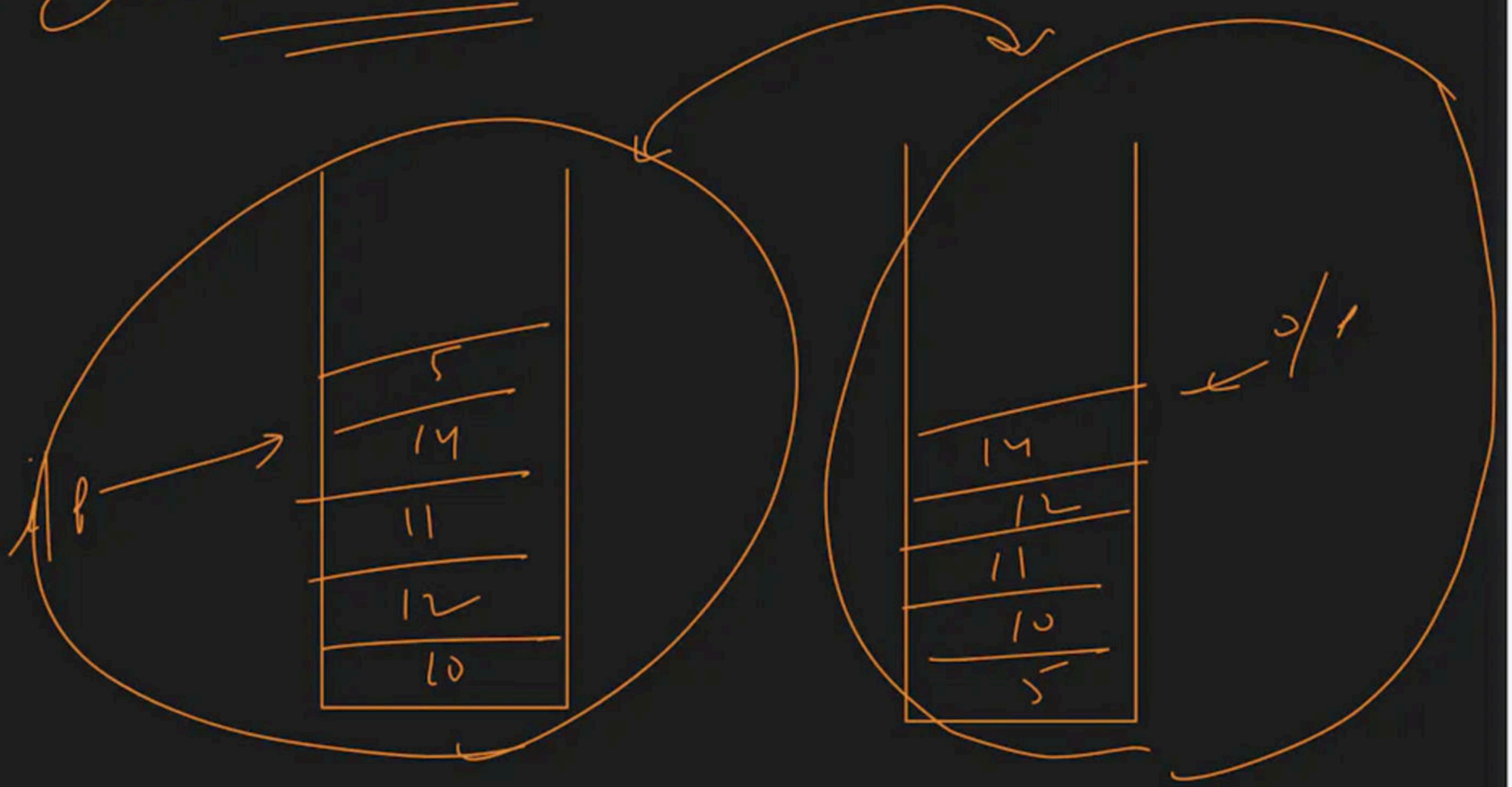


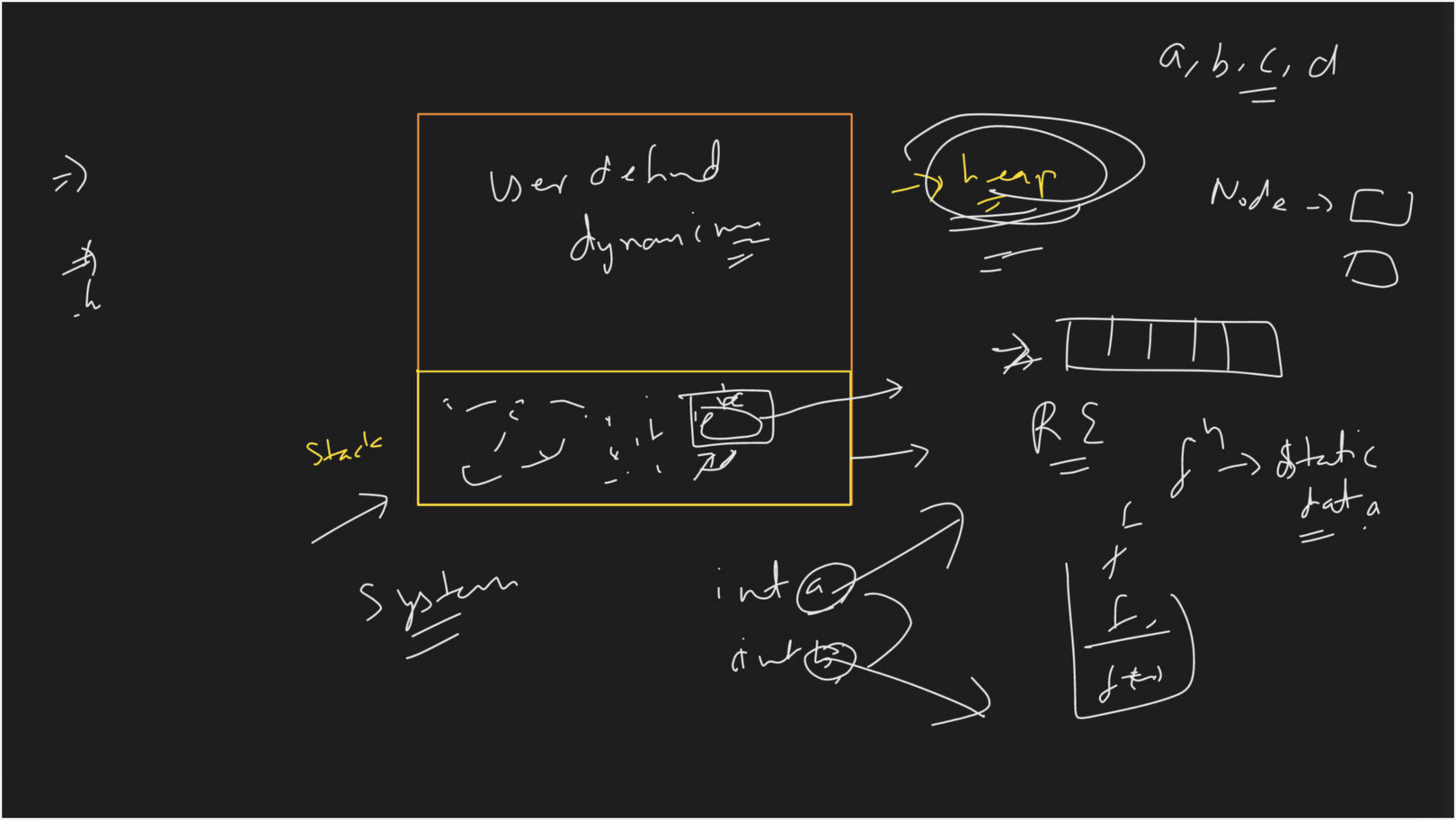

40
30
20
10
10
20
30
40
10
20
30
40
50


50
40
30
20
50 pop
50 IAB
40
30
20
40 pop
40 IAB
30
20
30 pop
30 IAB
20
20 pop
20 IAB
empty
20
20
30
20
30
40
20
30
40
50

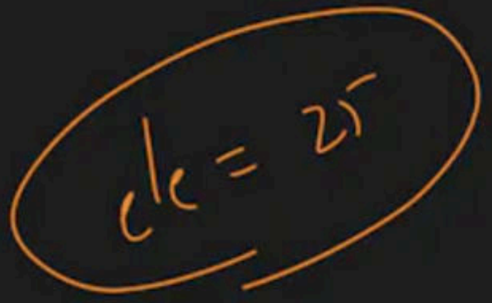
Sort a Stack

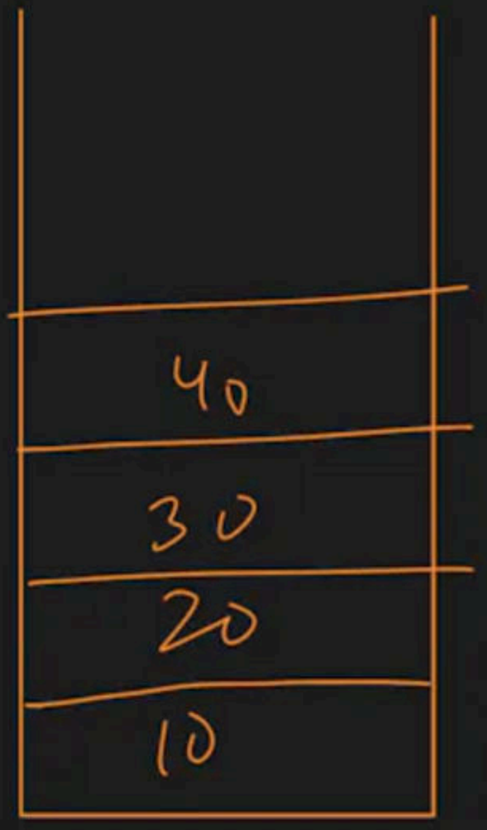
Now 2 mig

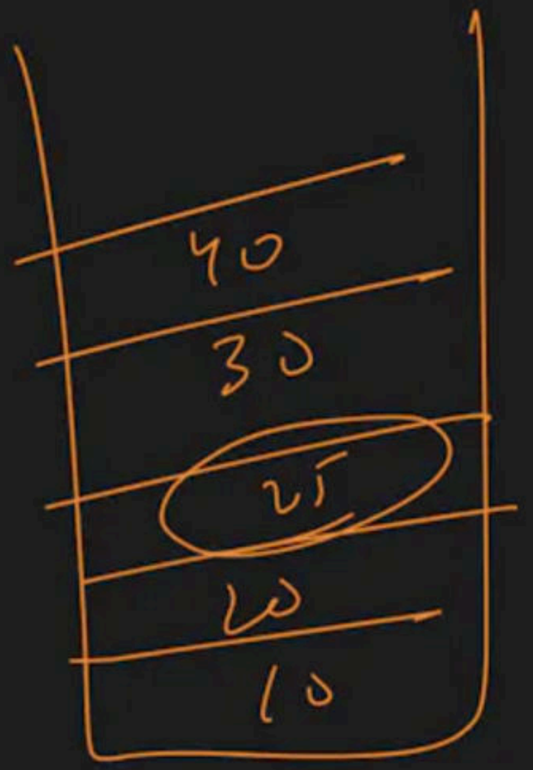


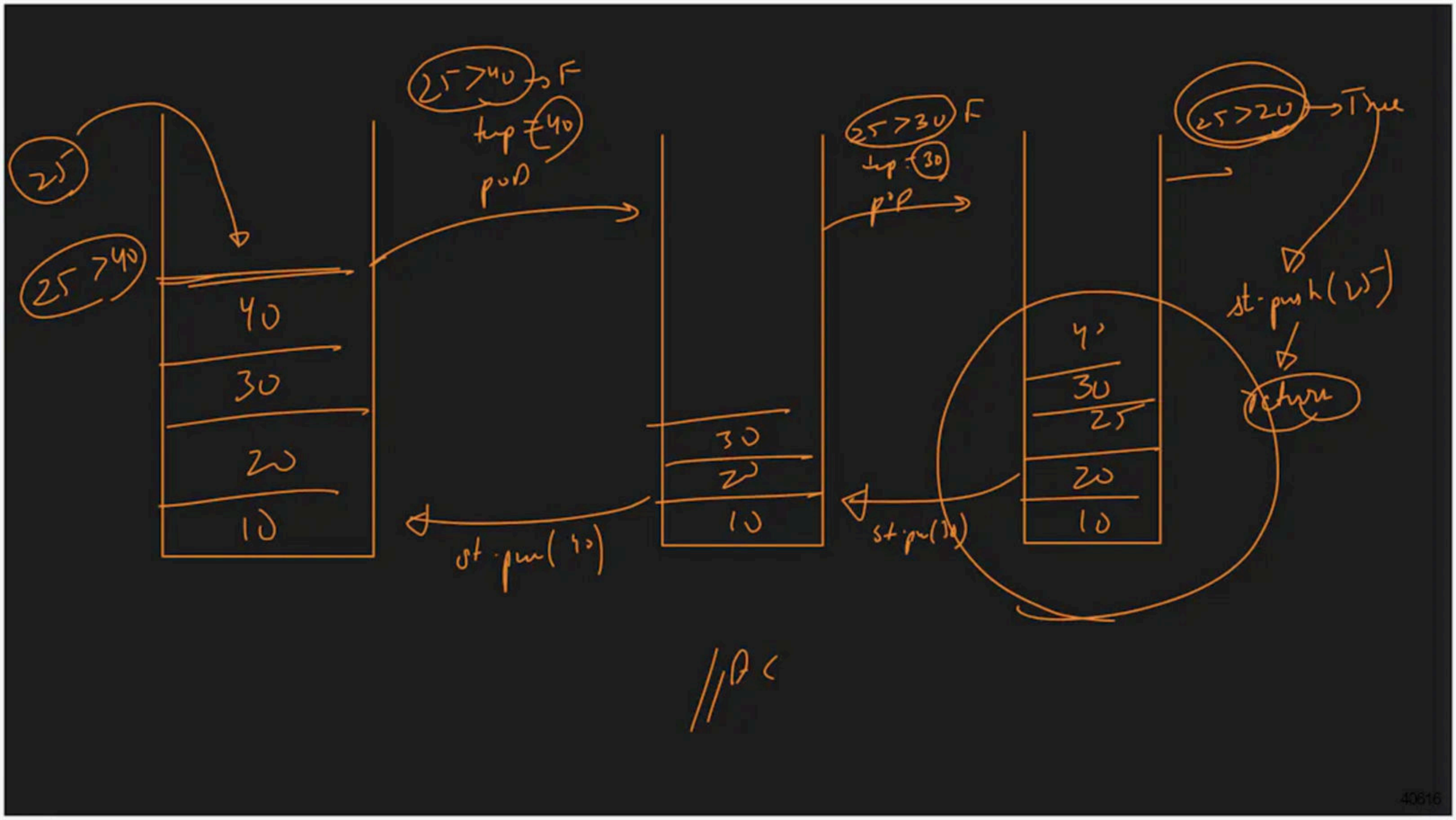

a, b, c, d
User defined
dynamic
heap
Node ->
Static
System
int a
int b
RE
f
f(x)

insert in a sorted ~~Array~~ Stack



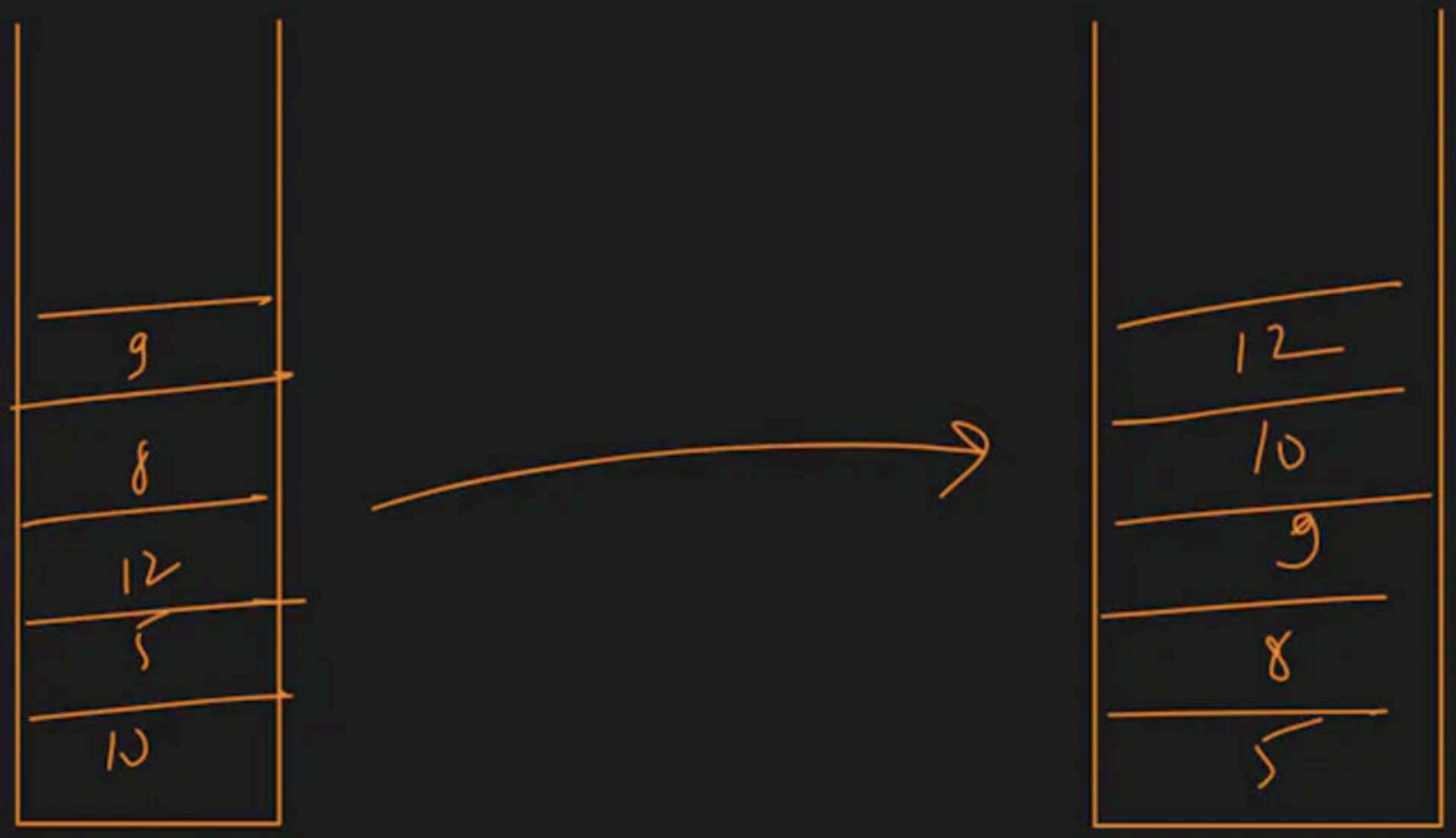


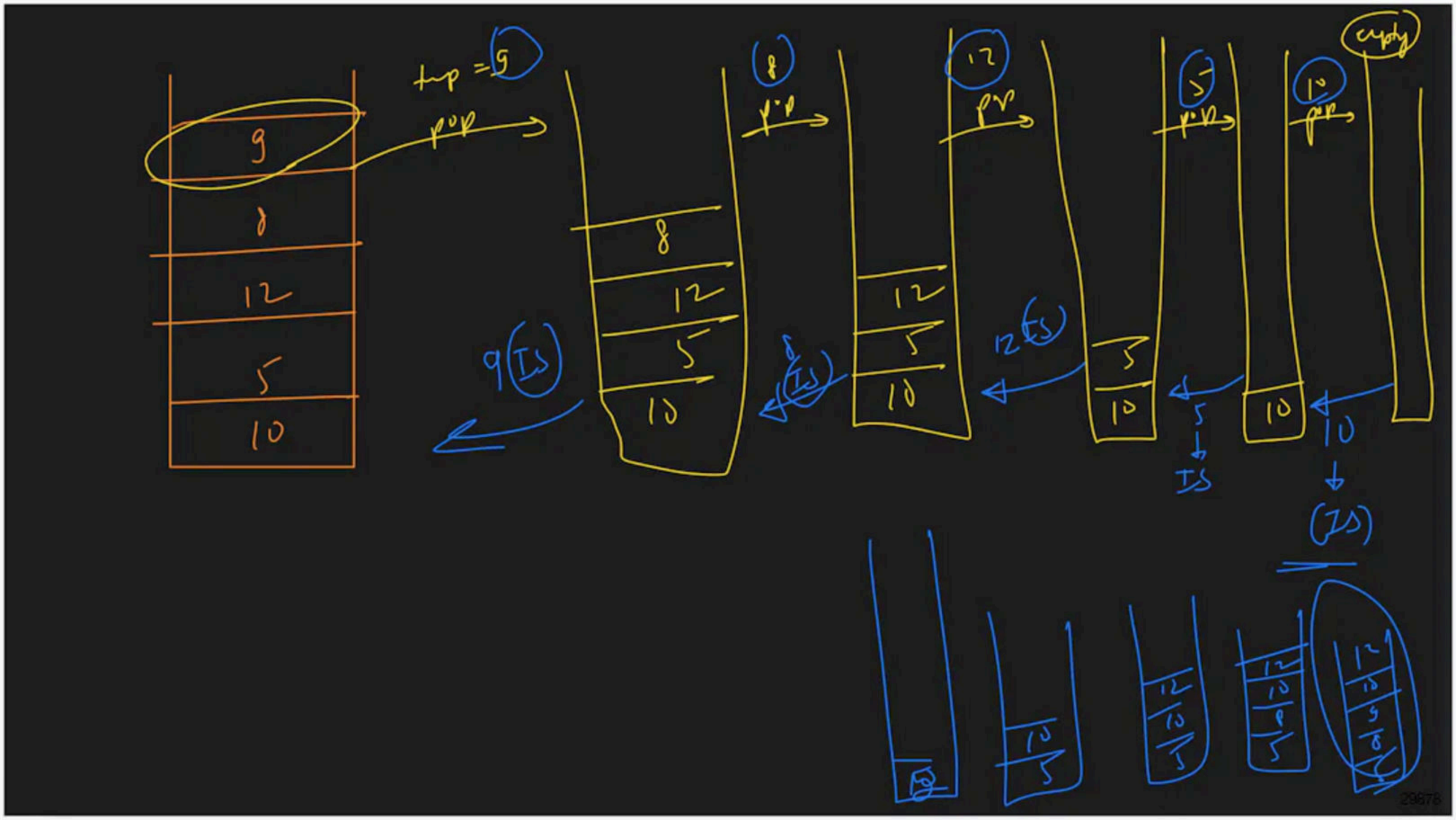



25
25 > 40
40
30
20
10
25 > 40 → F
top ≠ 40
pop
st.pop(40)
30
20
10
25 > 30 F
top : 30
pop
st.pop(30)
40
30
25
20
10
25 > 20 → True
st.push(25)
return

//Ac

Sort a Stack:-




top = 9
pop
9
8
12
5
10
9 (Is)
8
pop
8
12
5
10
(Is)
12
pop
12
5
10
12 (Is)
5
pop
5
10
5
Is
10
pop
10
empty
10
(Is)

⇒ (1) Implemet 2 stacks in an Array

⇒

2 Valid Parenthesis →

(3) Check Redundant Brackets

=> (oper(ont =>) ???

=>

=> (a+b) =>

((a)+(b+c))

flag = True;

while ({ st.top ≠ '(')

<

if (operator a ??)

flag = false;

>

st.pop();

if (flag == True) return True;

1 Min Stack

Stack Class-3

Special class

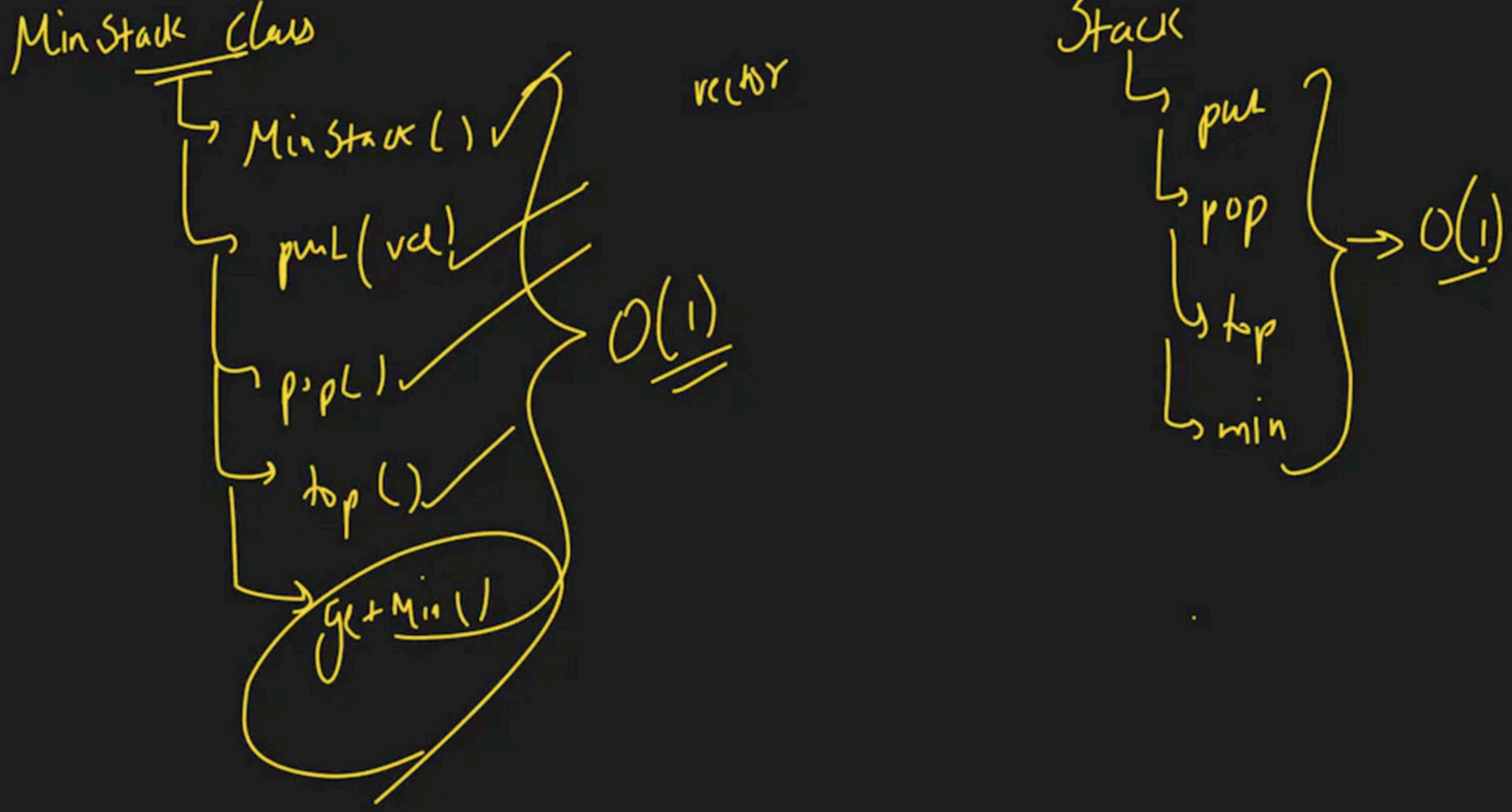
Love Babbar • Nov 15, 2023

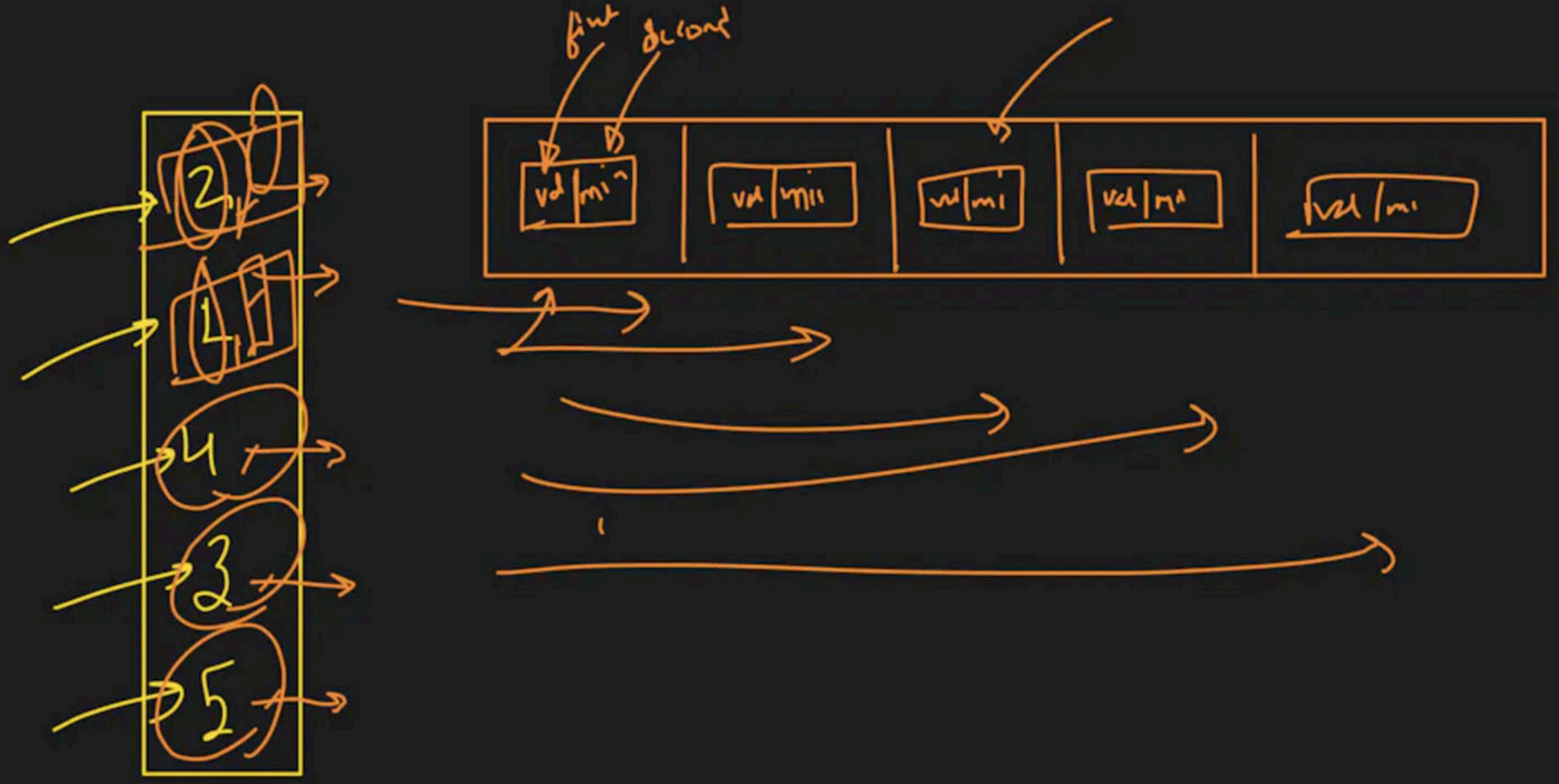
→ Implement a Min Stack

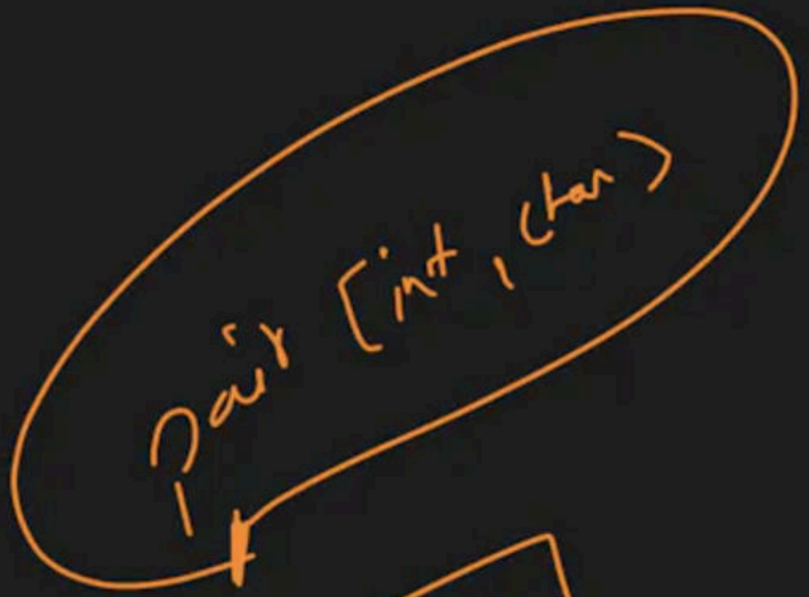
Stack
- push → O(1)
- pop → O(1)
- getMin() → O(n)

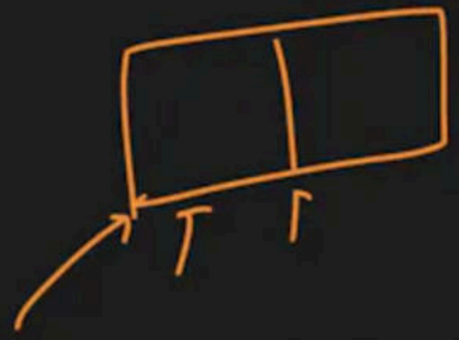
Min Stack
- push
- pop
- top
- getMin

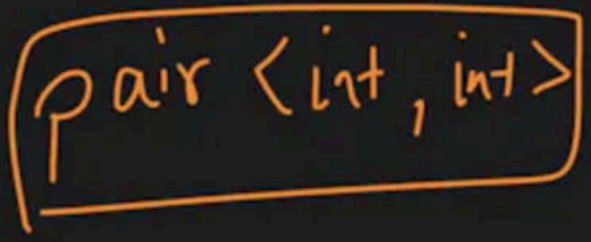
→ O(1)


MinStack Class
MinStack()
push(val)
pop()
top()
getMin()
O(1)
Stack
push
pop
top
min
O(1)


first
second
2
1
4
3
5


pair <int, char>



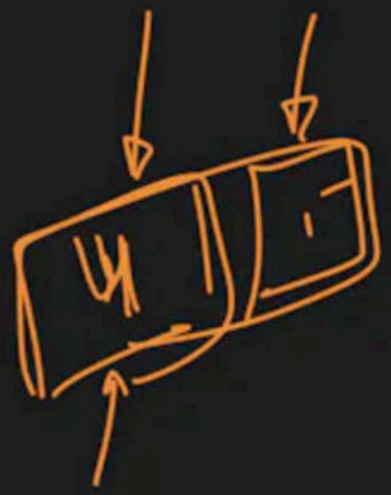

pair <int, int>

Creat

pair<int, int> p = make_pair(:);

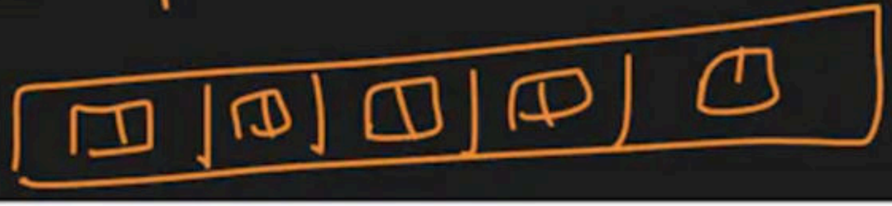
p.first = 4

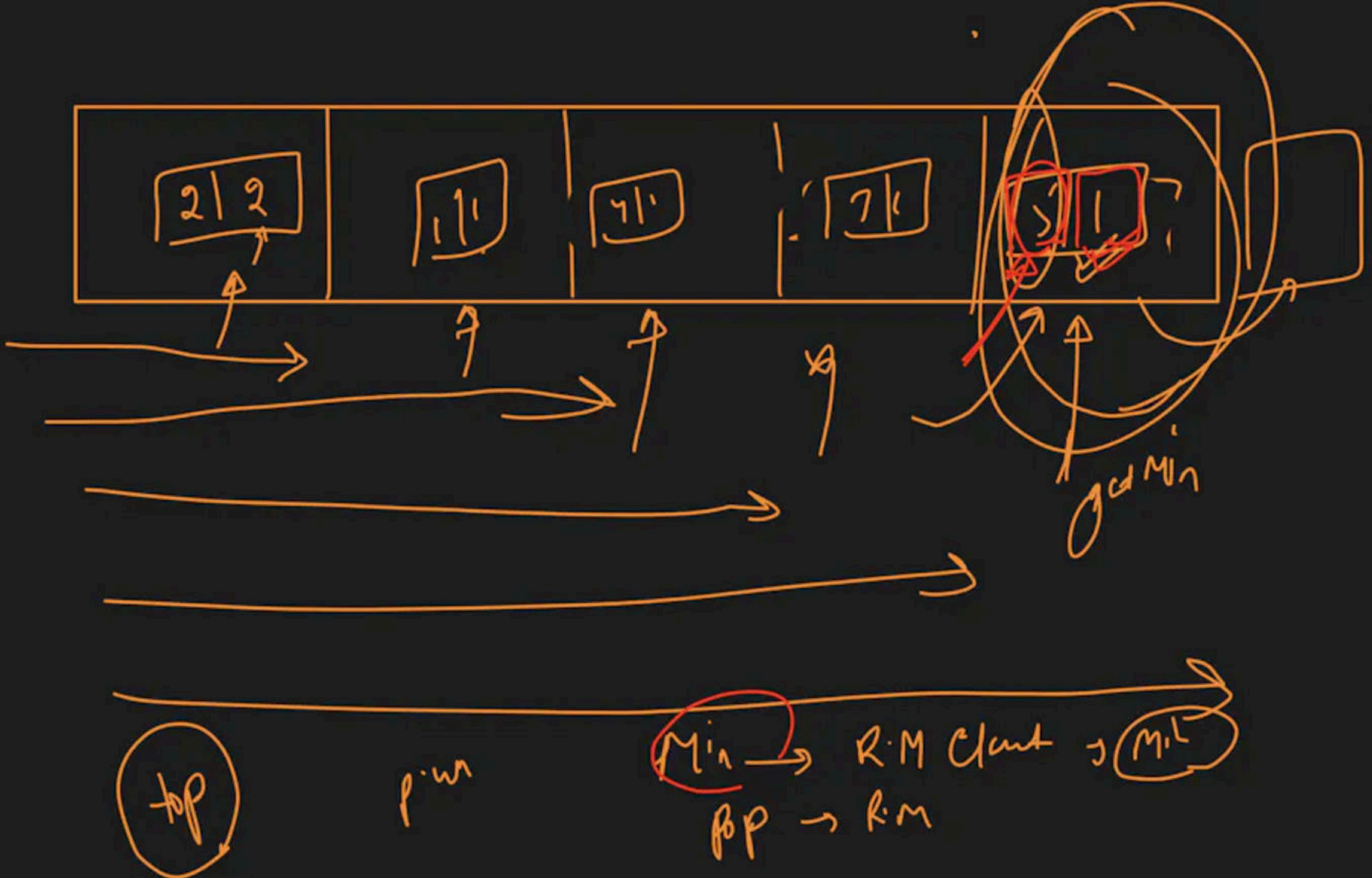
p.second = 1

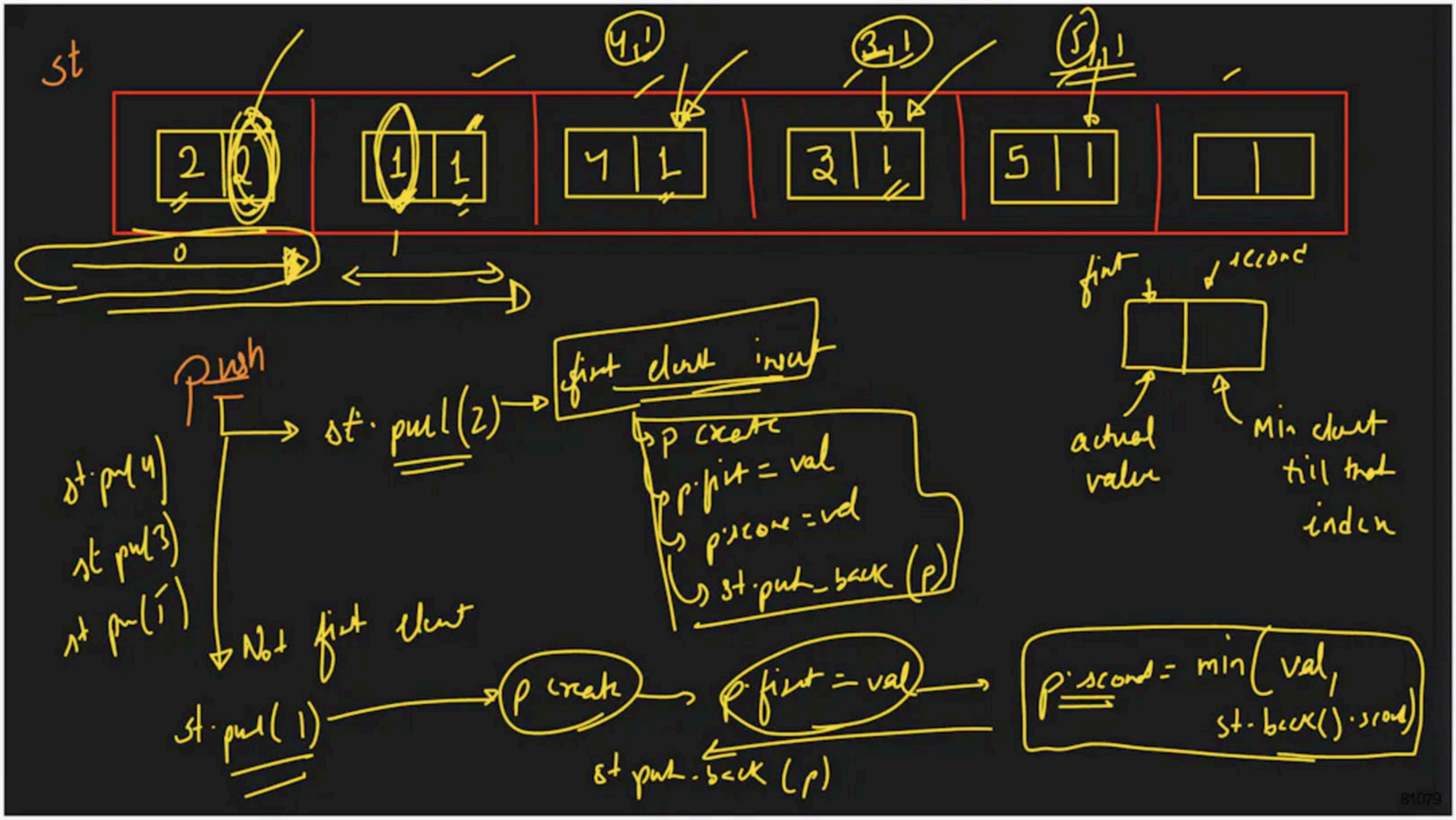


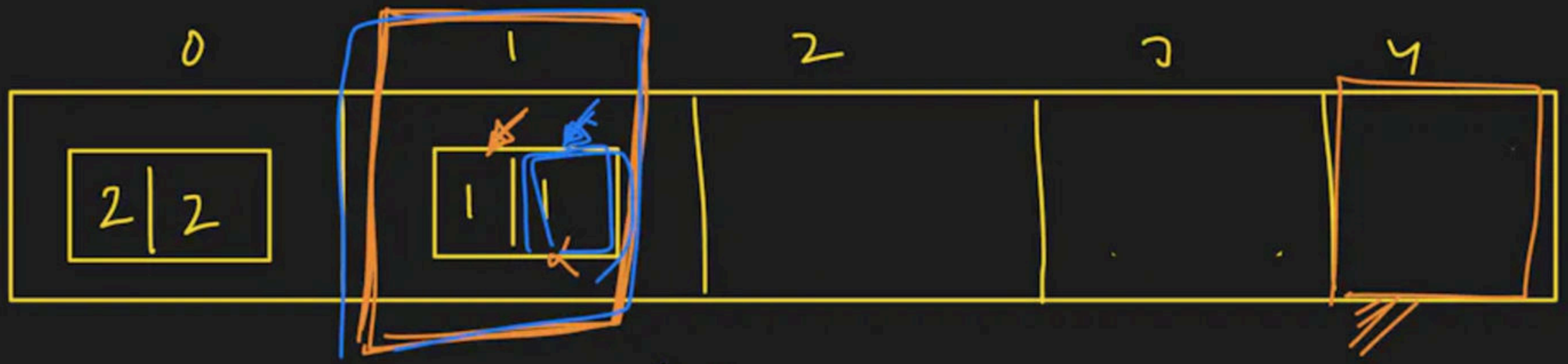
vector<int>

vector<pair<int, int> >




2 | 2
1 | 1
4 | 1
7 | 1
5 | 1
getMin
Min → R.M Count → Min
pop → R.M
top


st
2 | 2
1 | 1
4 | 1
3 | 1
5 | 1
0
first
second
actual value
Min count till that index
Push
st.push(2)
first elem inserted
p create
p.first = val
p.second = val
st.push_back(p)
st.push(4)
st.push(3)
st.push(1)
Not first elem
st.pop()
p create
p.first = val
st.push_back(p)
p.second = min(val, st.back().second)



pop ->

st.pop_back()

st.pop_back()

st.pop_back()

top

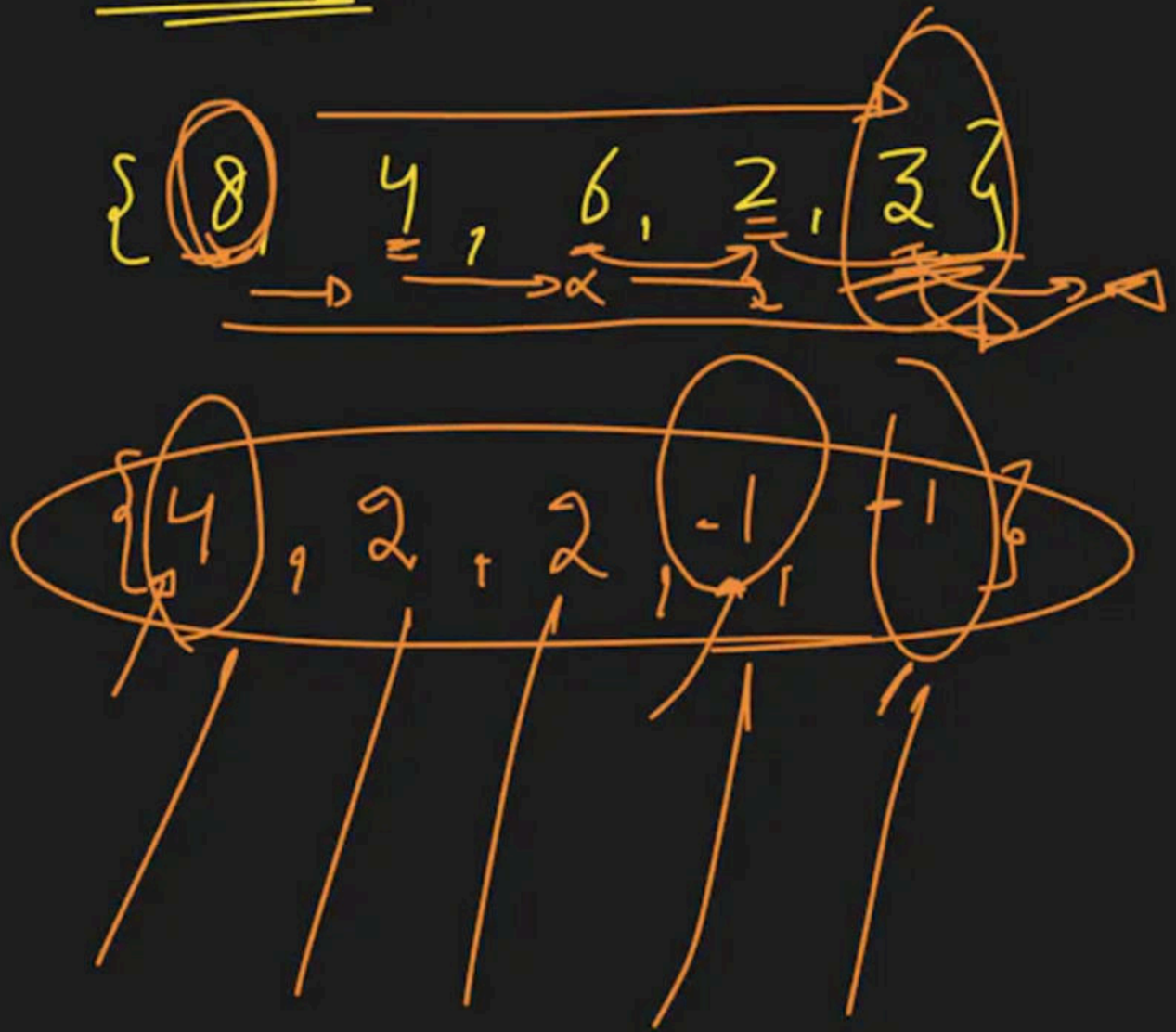
st.back().first

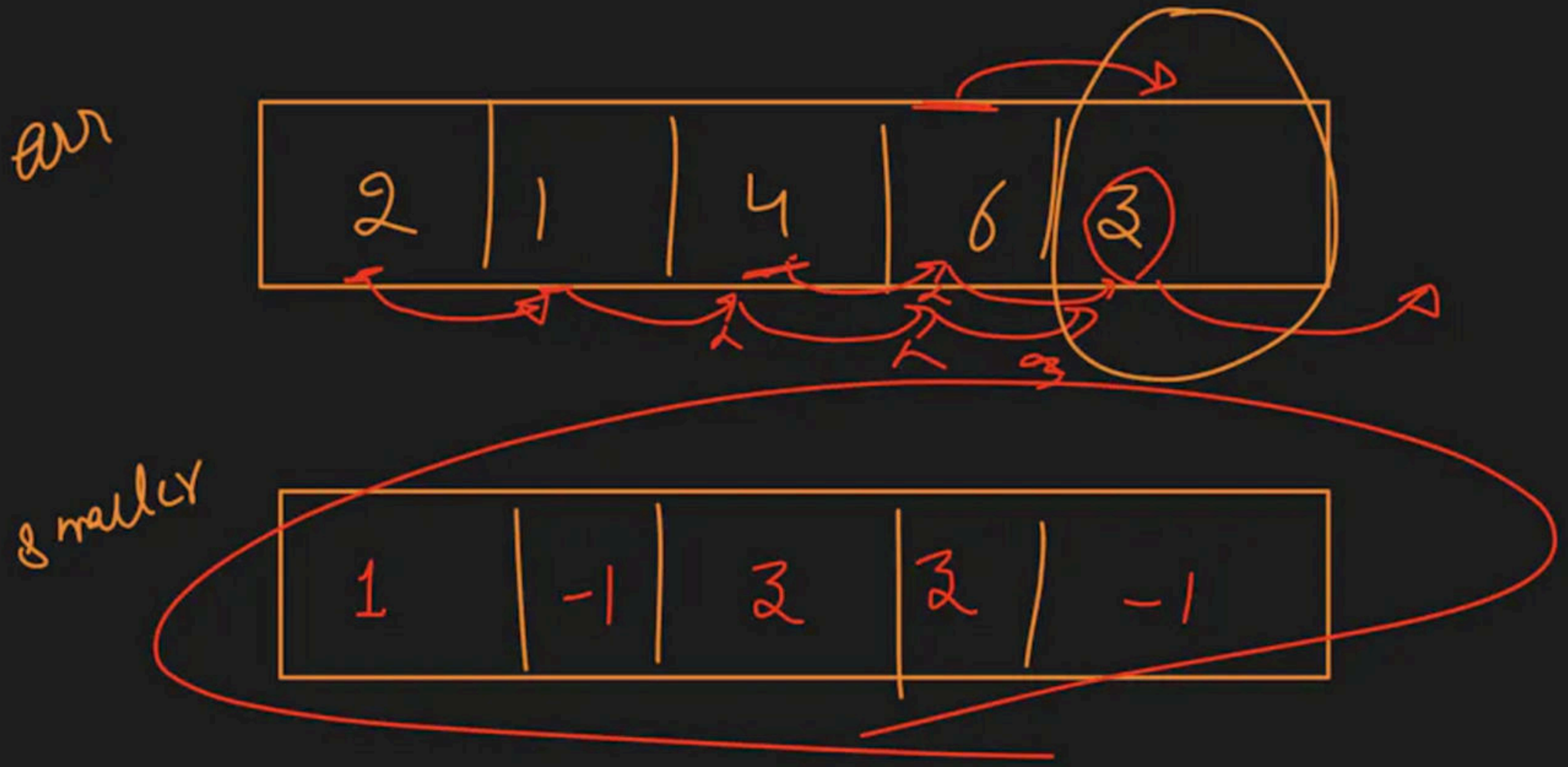
getMin()

st.back().second

Next Smaller Element

arr[] = {8, 4, 6, 2, 3}



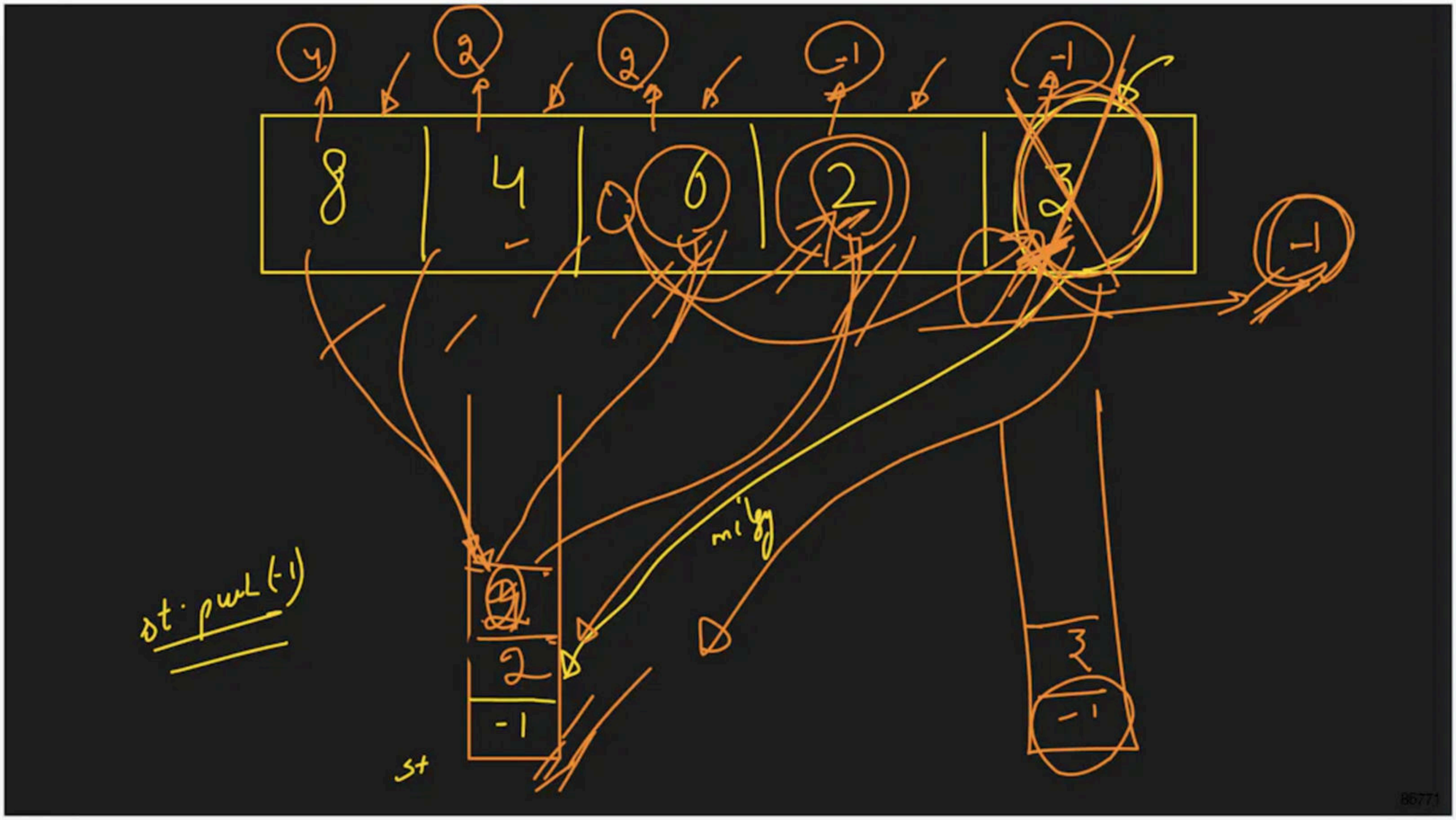

arr
2
1
4
6
3
smaller
1
-1
3
3
-1

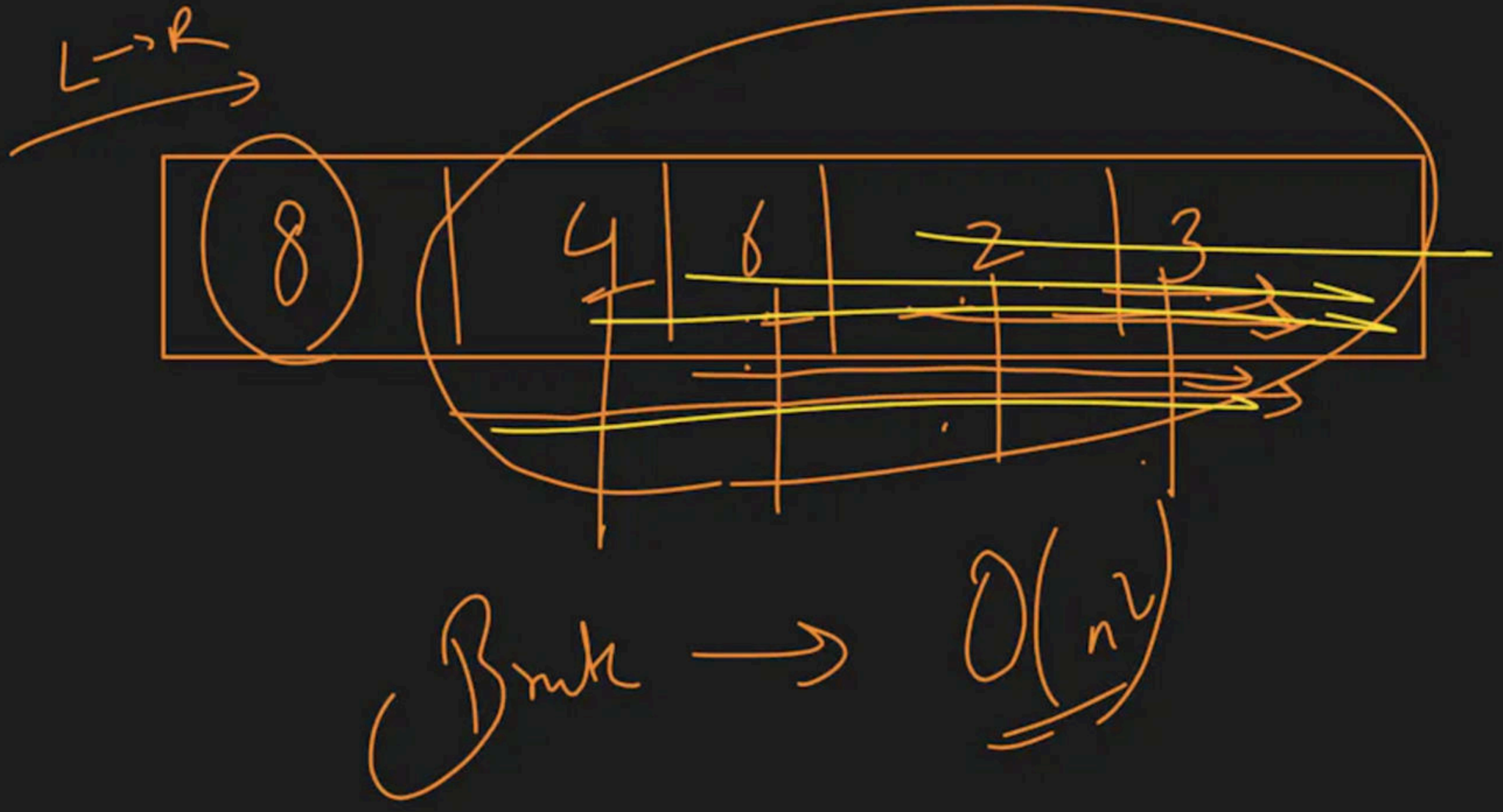
#Brute

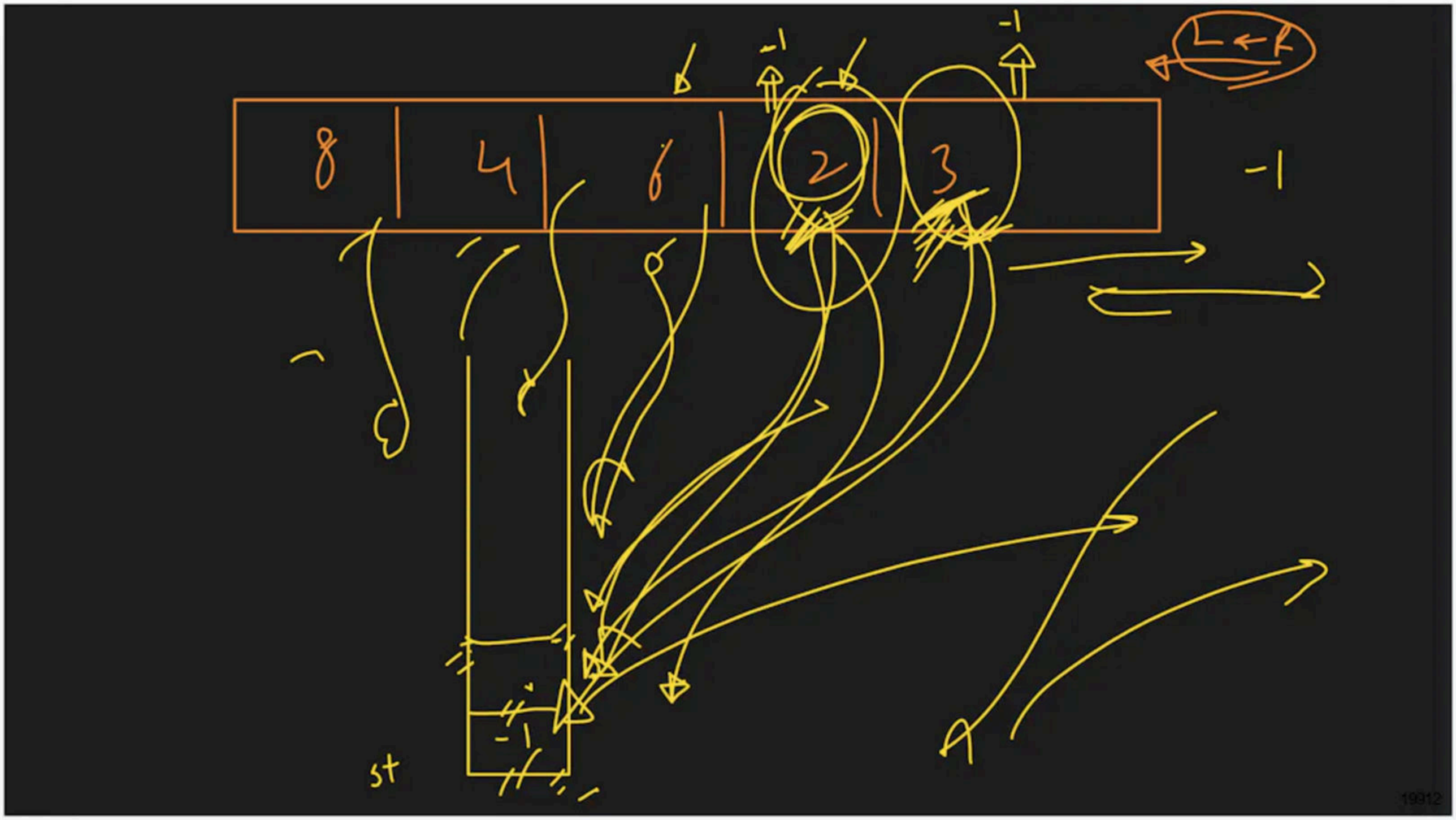
L → R

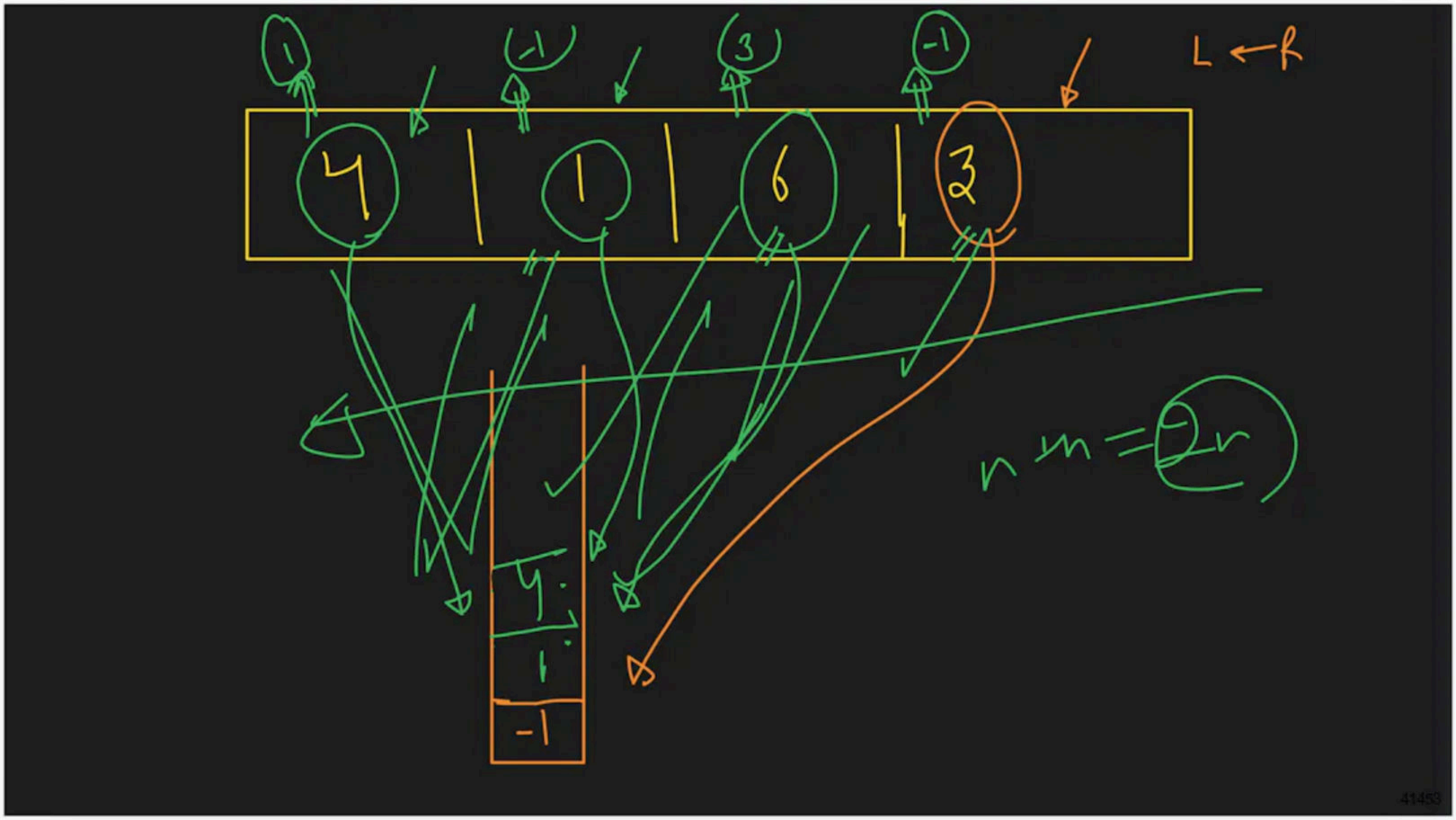
(n-1) + (n-2) + (n-3) - - ~ + 1 + 0

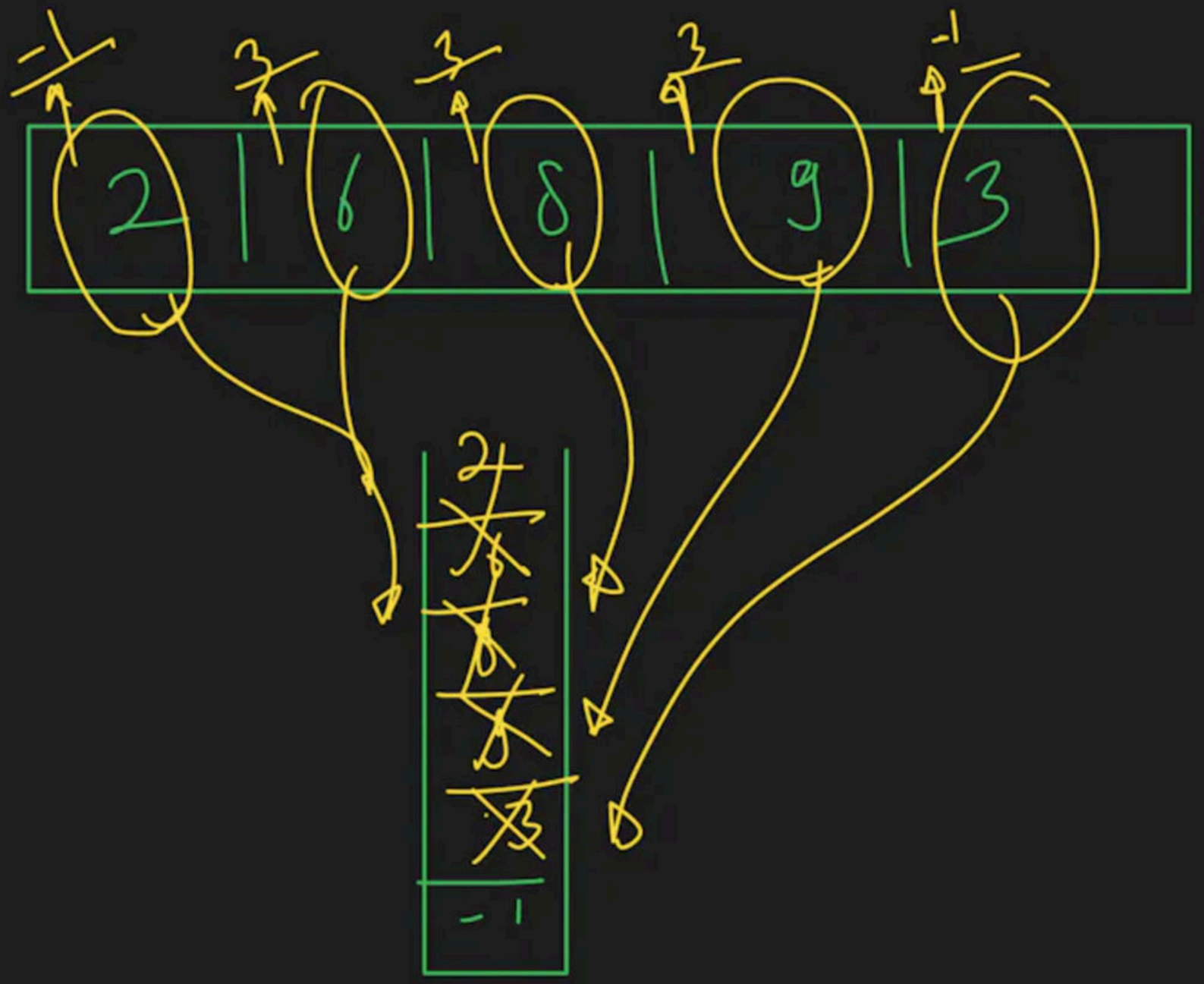
O(n!)


4
2
2
-1
-1
8
4
6
2
3
-1
st.pwL(-1)
st
3
-1

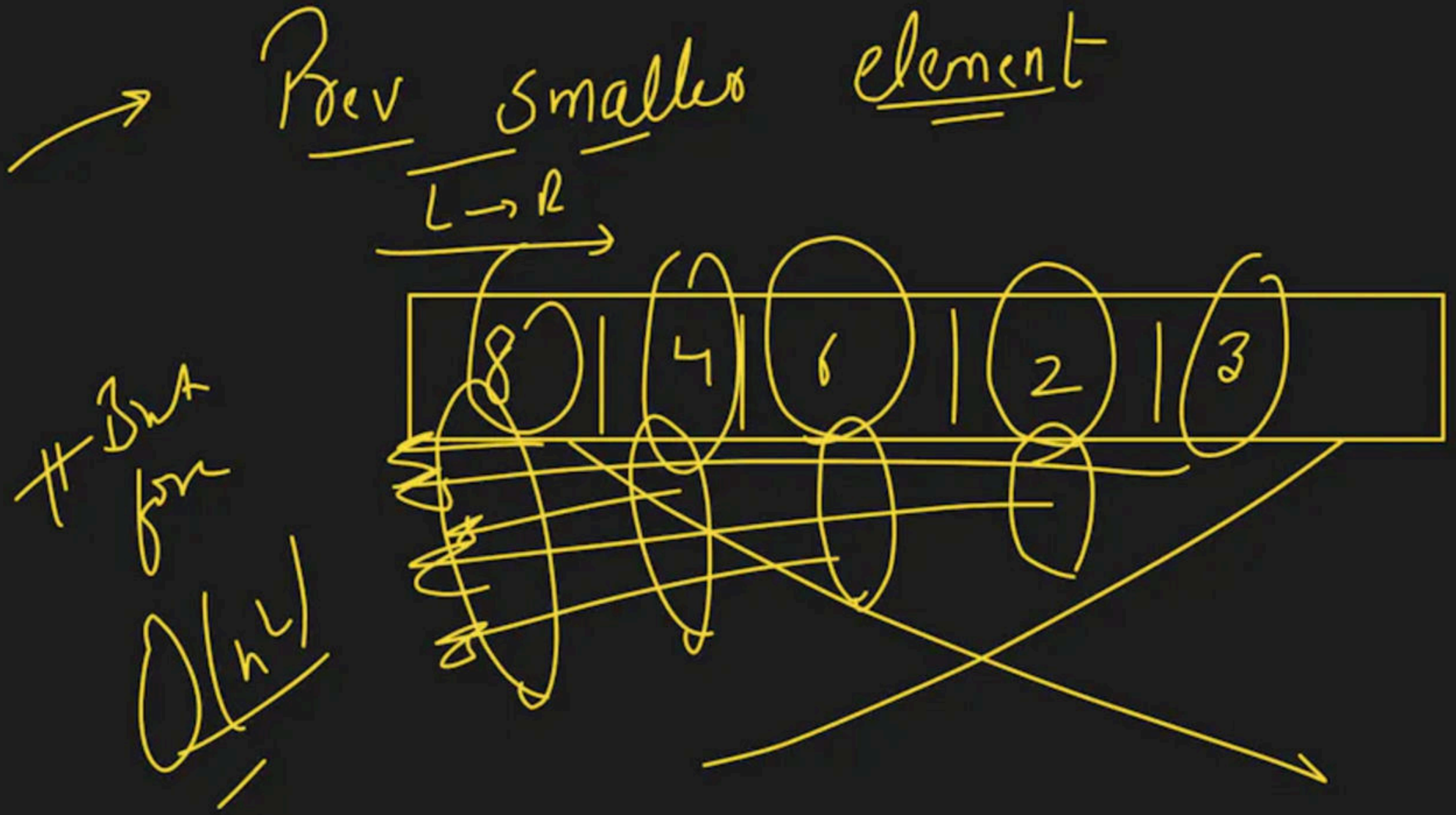

L -> R
8
4
6
2
3
Brute → O(n²)

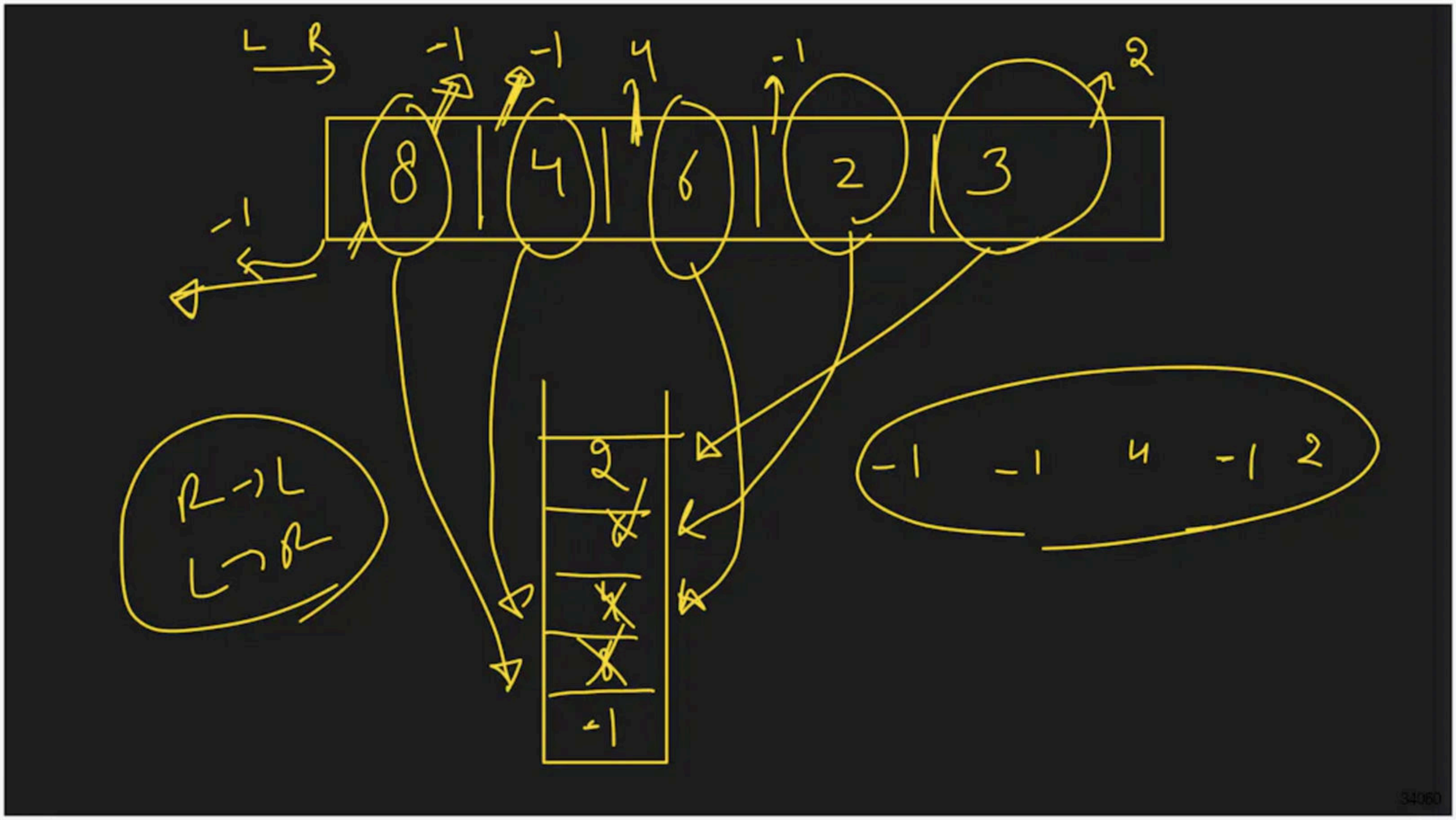

8
4
6
2
3
-1
-1
-1
-1
st


1
-1
3
-1
L ← R
4
1
6
3
4
1
-1
n m = 2n


-1
3
3
3
-1
2 | 6 | 8 | 9 | 3
2
-1

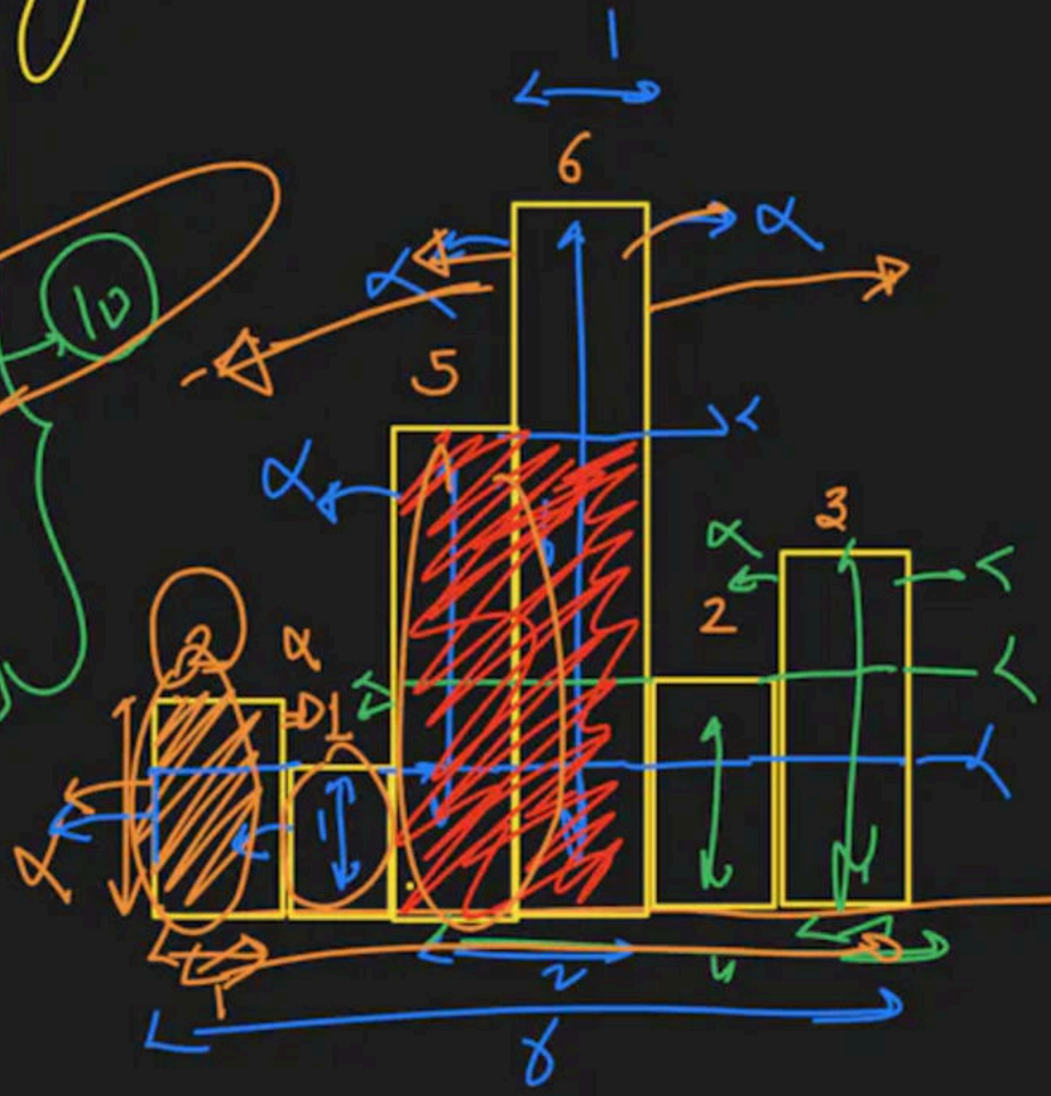
Prev Smaller element




L R
-1
-1
4
-1
2
8
4
6
2
3
-1
2
2
-1
R->L
L->R
-1 -1 4 -1 2

2 min → Break

Largest Area in a Histogram



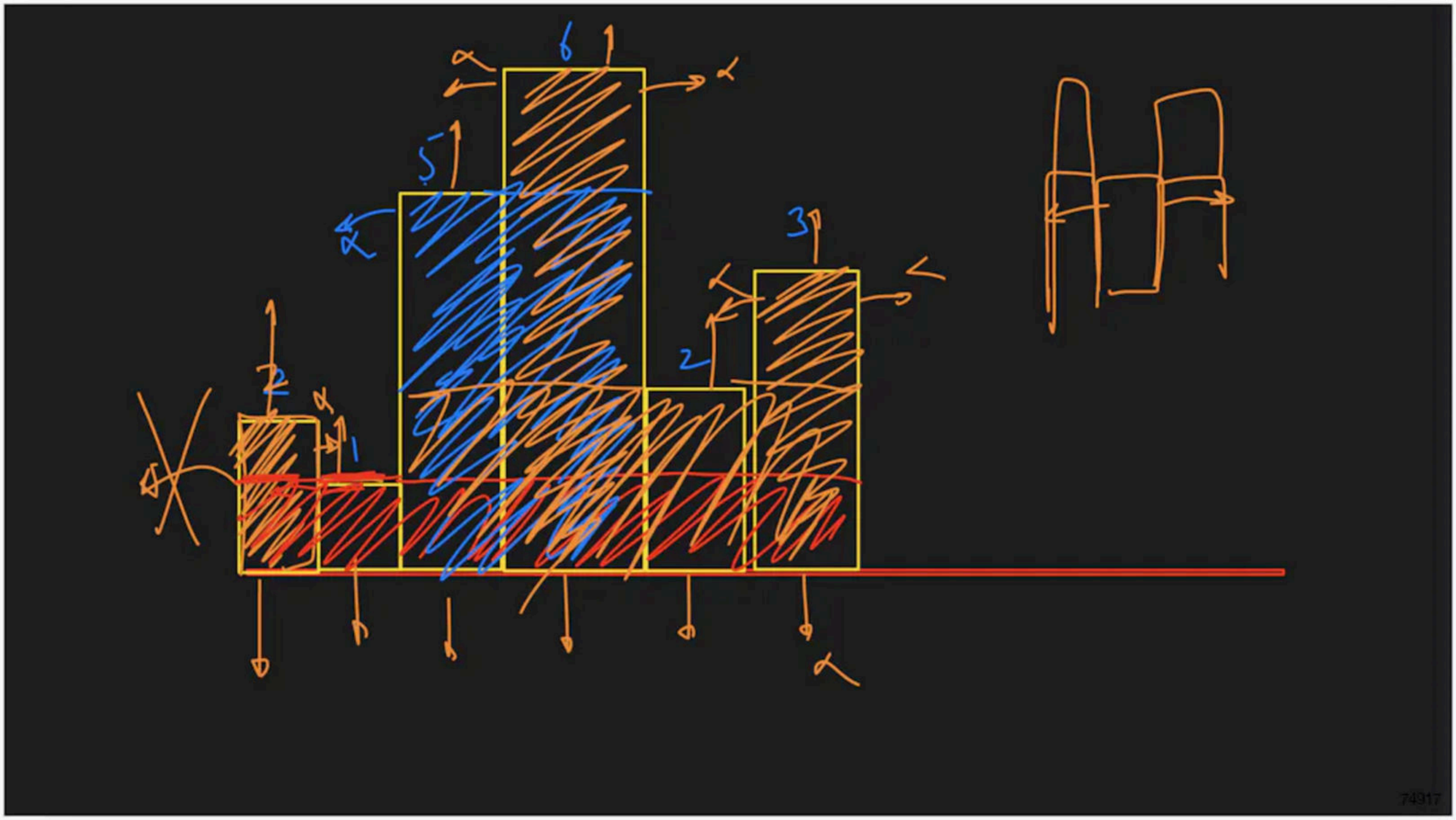
$A_1 = 2$
$A_2 = 6$
$A_3 = 10$
$A_4 = 6$
$A_5 = 8$
$A_6 = 3$

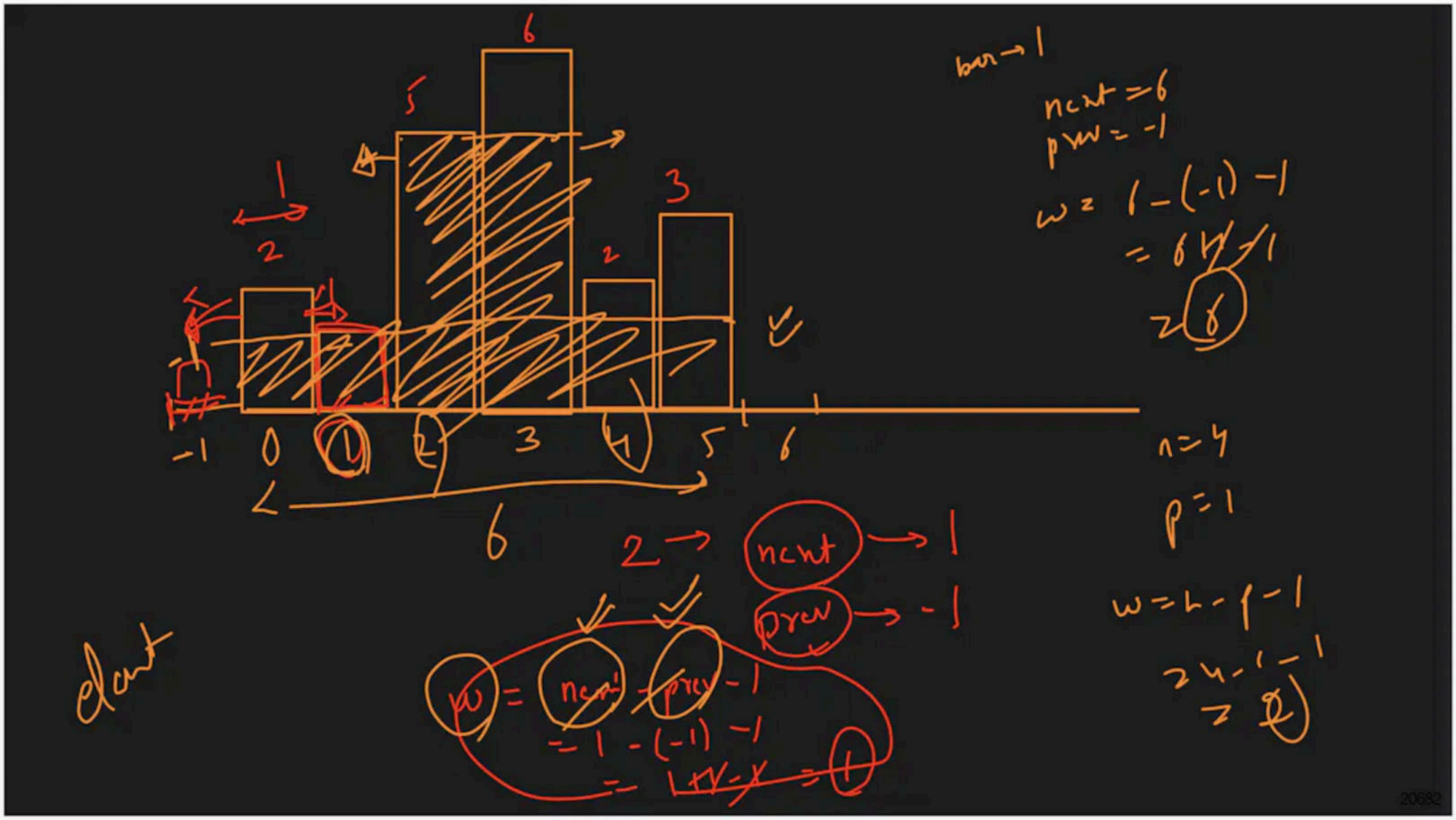
10

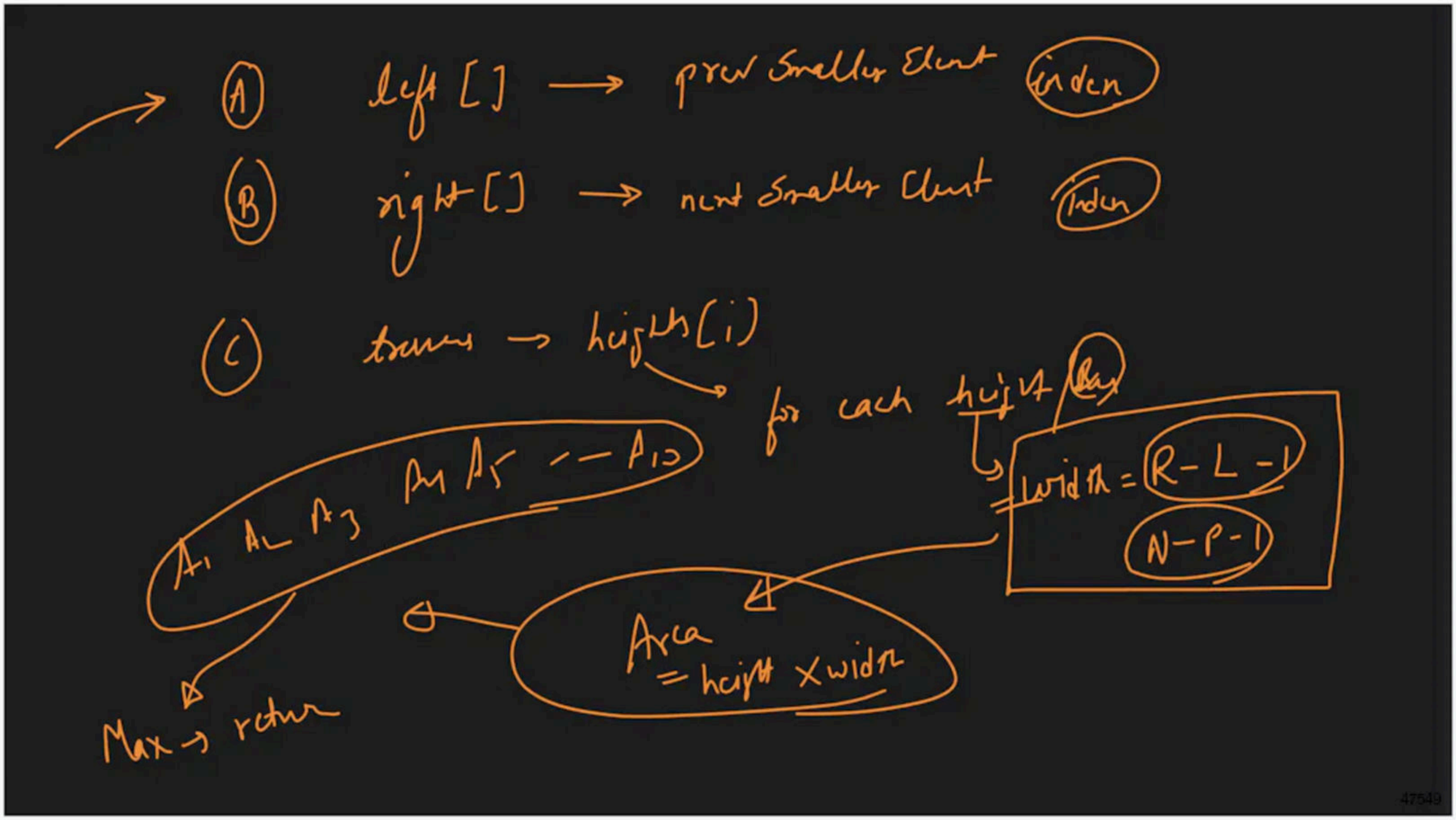
Largest rectangular area in a histogram

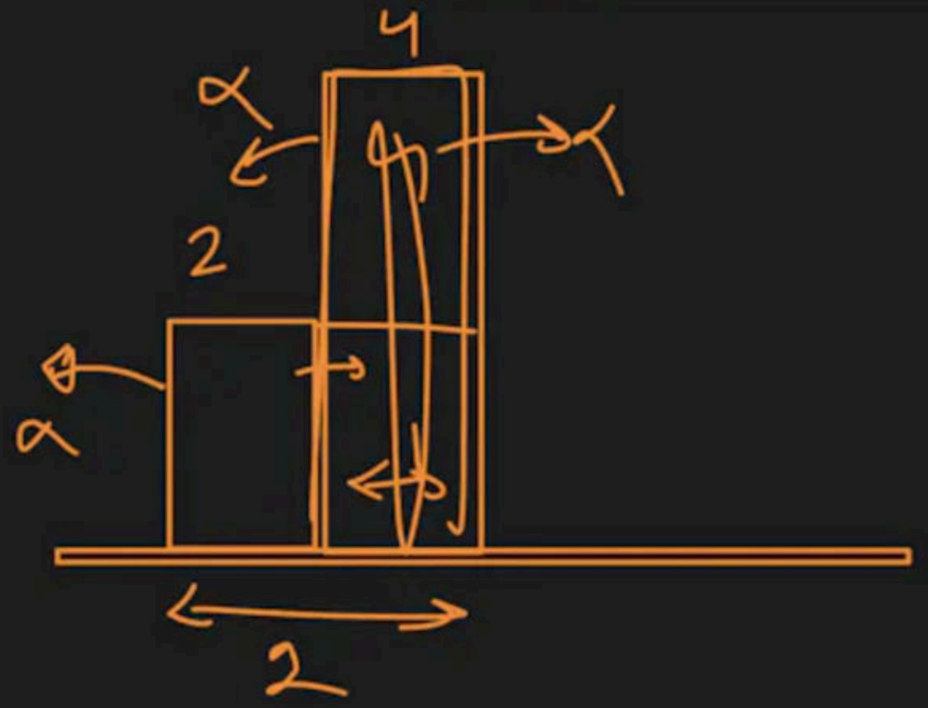
$W = 2$

$A = l \times w$


6
5
3
2
2
1


6
5
3
2
2
1
2
-1
0
1
2
3
4
5
6
6
bar → 1
next = 6
prev = -1
w = 1 - (-1) - 1
= 6 - 1
= 5
i = 4
p = 1
w = 4 - 1 - 1
= 2
2 →
next → 1
prev → -1
w = next - prev - 1
= 1 - (-1) - 1
= 1 + 1 - 1 = 1
elent


A) left [] → prev Smaller Elem
index
B) right [] → next Smaller Elem
index
C) traverse → heights[i]
for each height
width = R - L - 1
N - P - 1
Area = height × width
Max → return



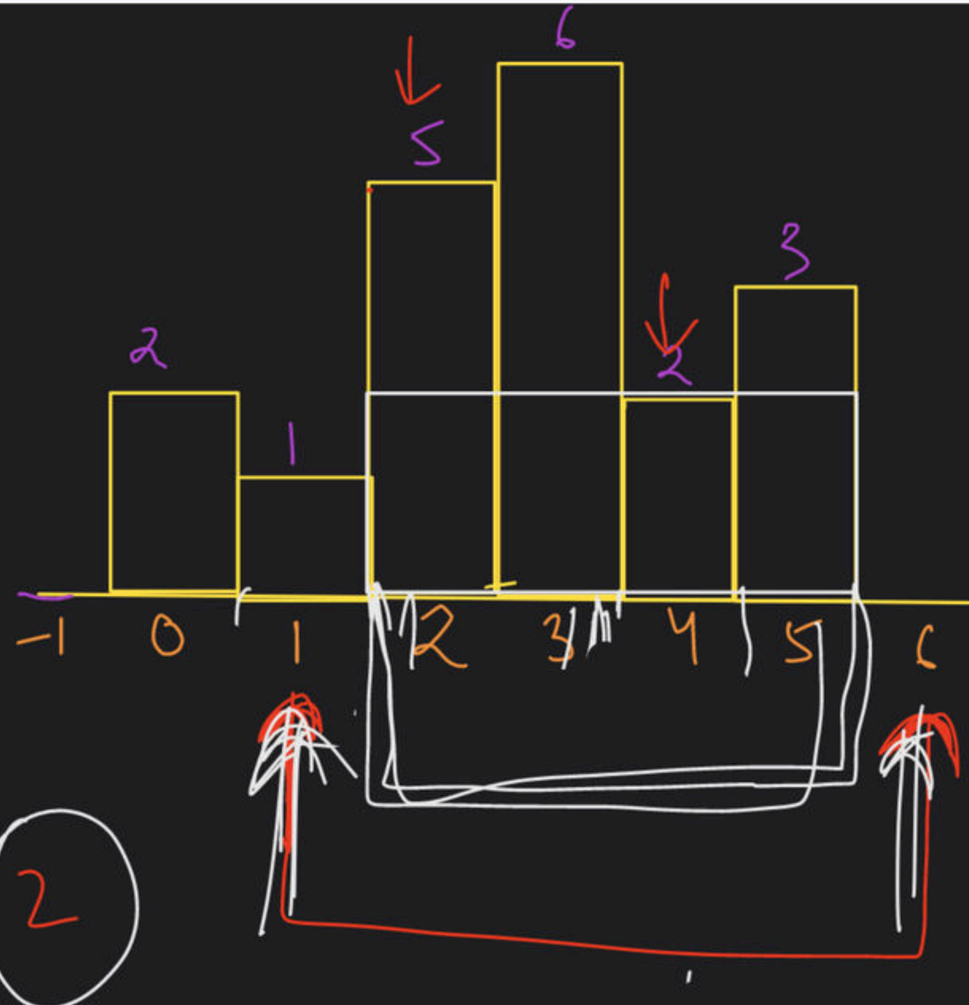
$A_1 = 2 \times 2$

$= 4$

$A_2 = 1 \times 1$

$= 1$

max $\rightarrow 4$



	2	1	5	6	2	3
Next	1	-1	4	4	-1	~~6~~ -1

Prev	-1	-1	1	2	1	4

$\Rightarrow$

$H = 2$

$W = 6 - 1 - 1$

$= 6 - 2 = 4$

$A = 8$

How to do DSA + Web Devl

← Push X

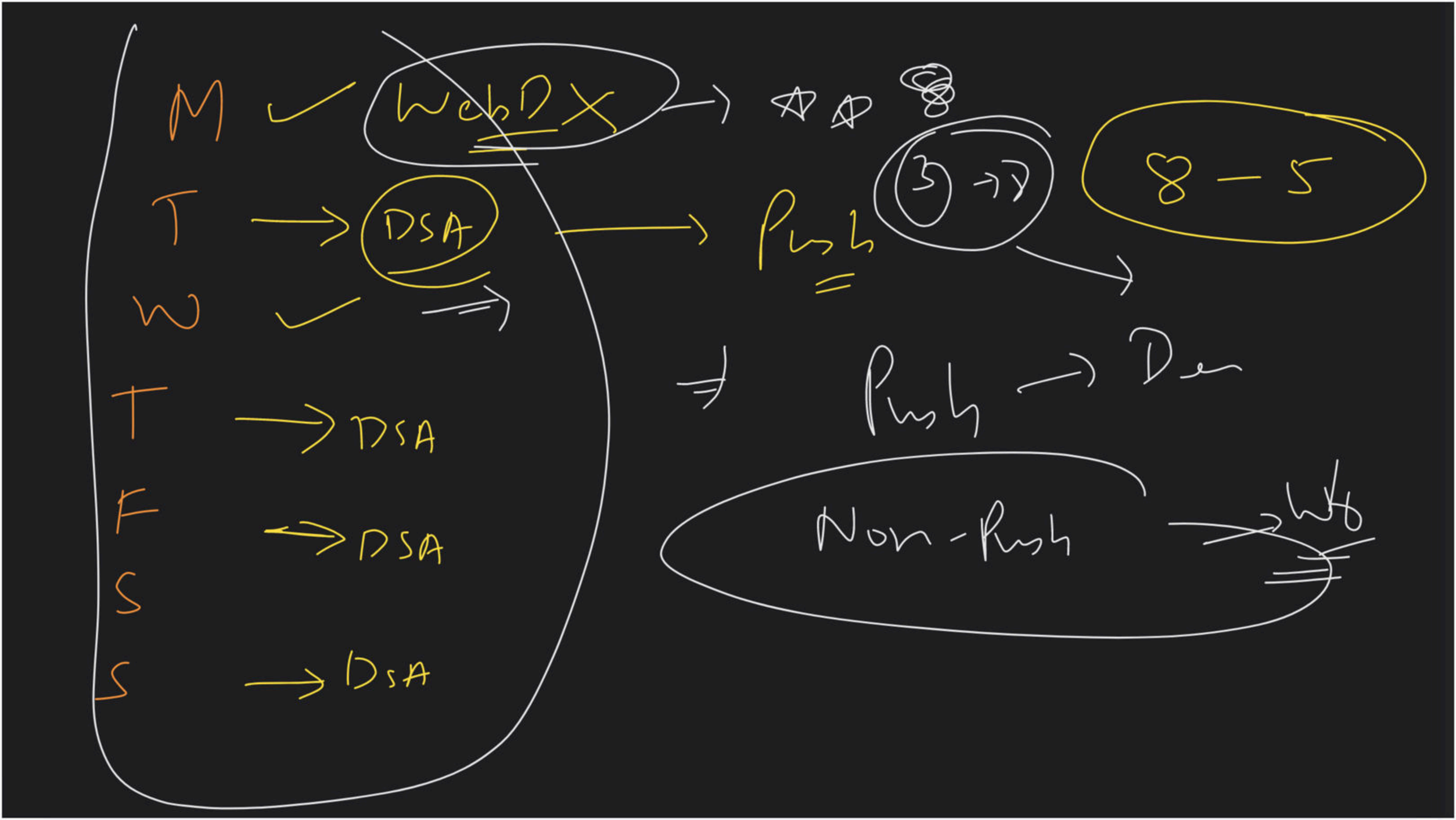
⇒

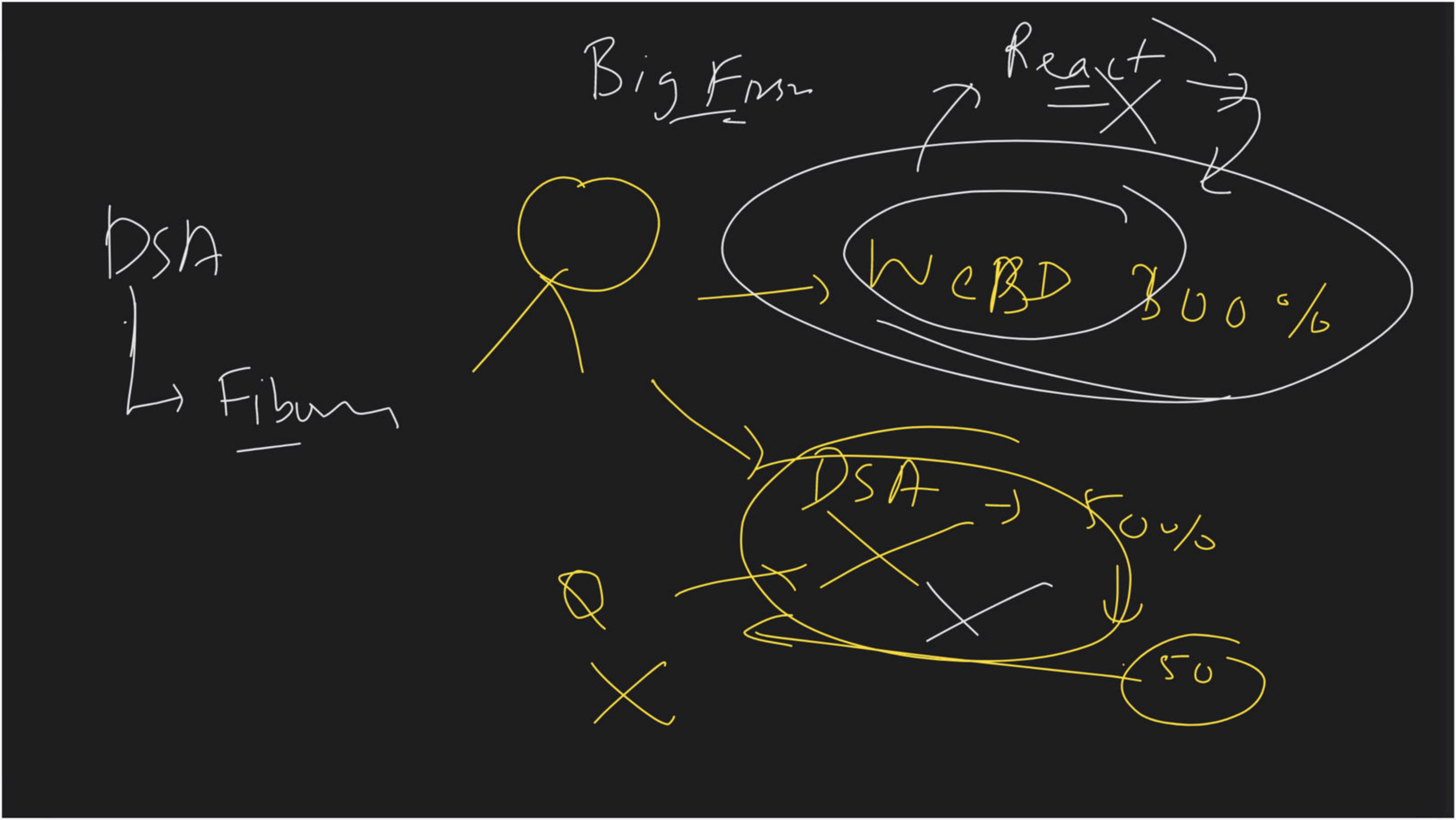
① Creative :

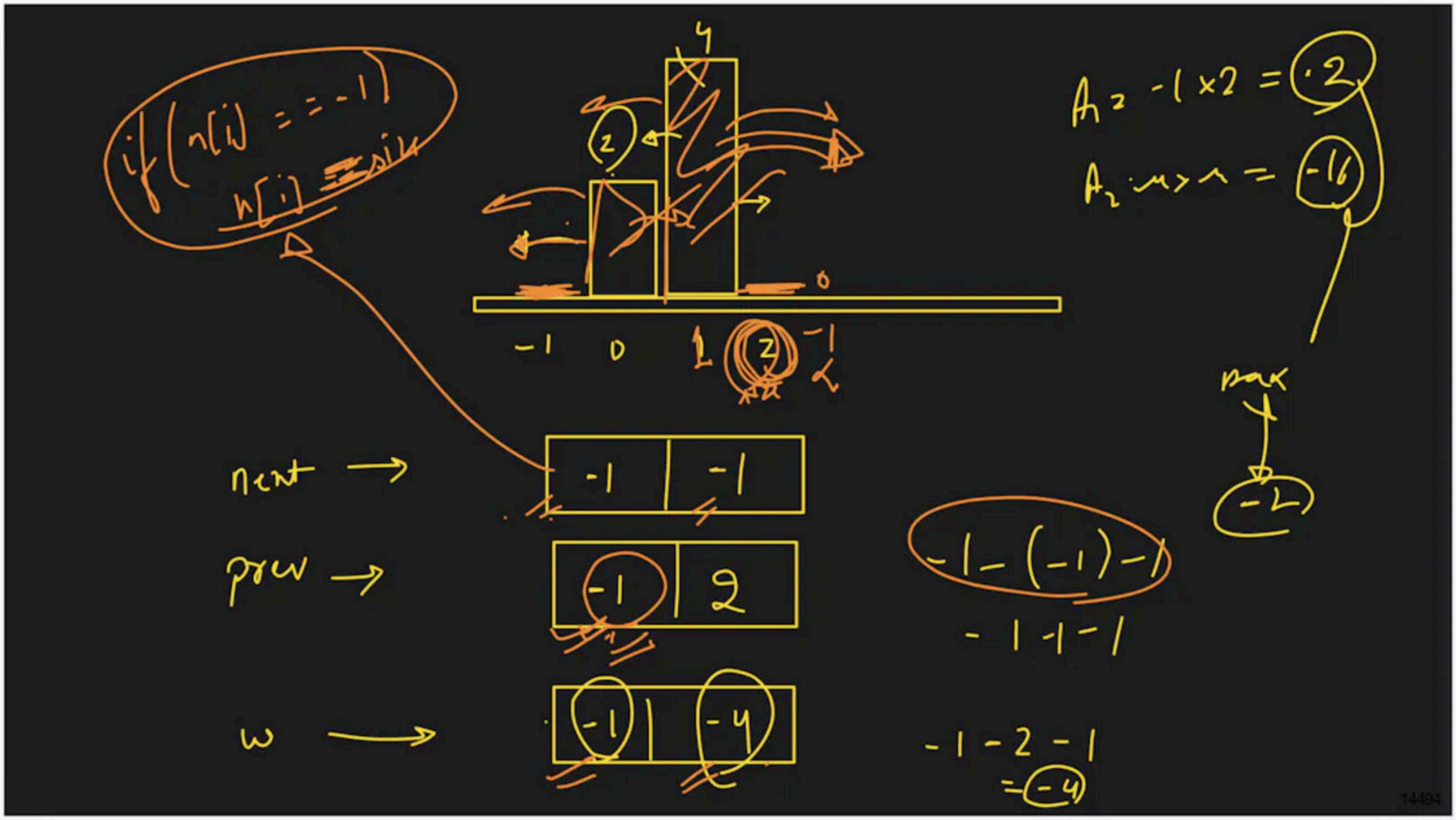
② Boring :

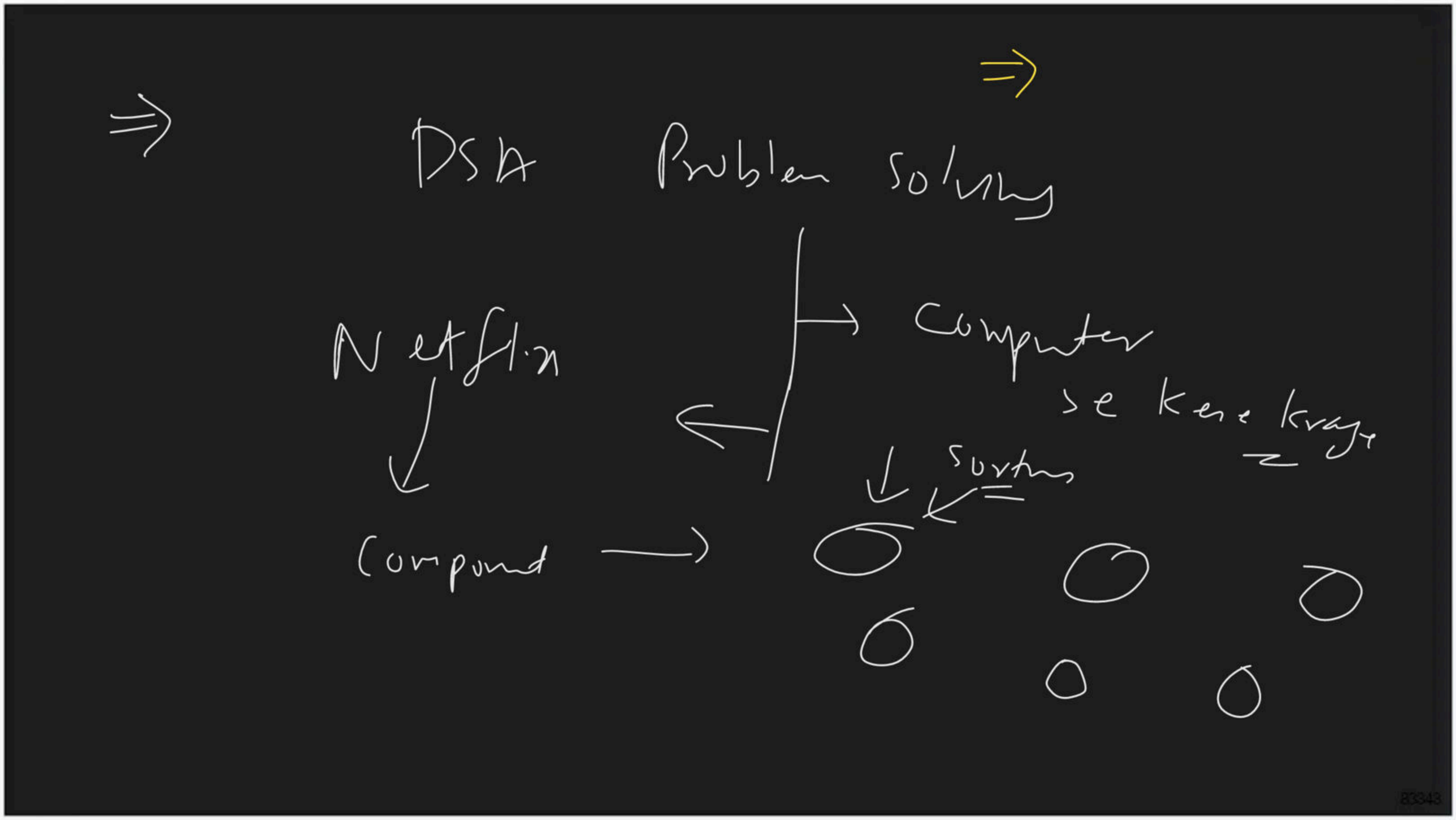
→ Push (Discipline)

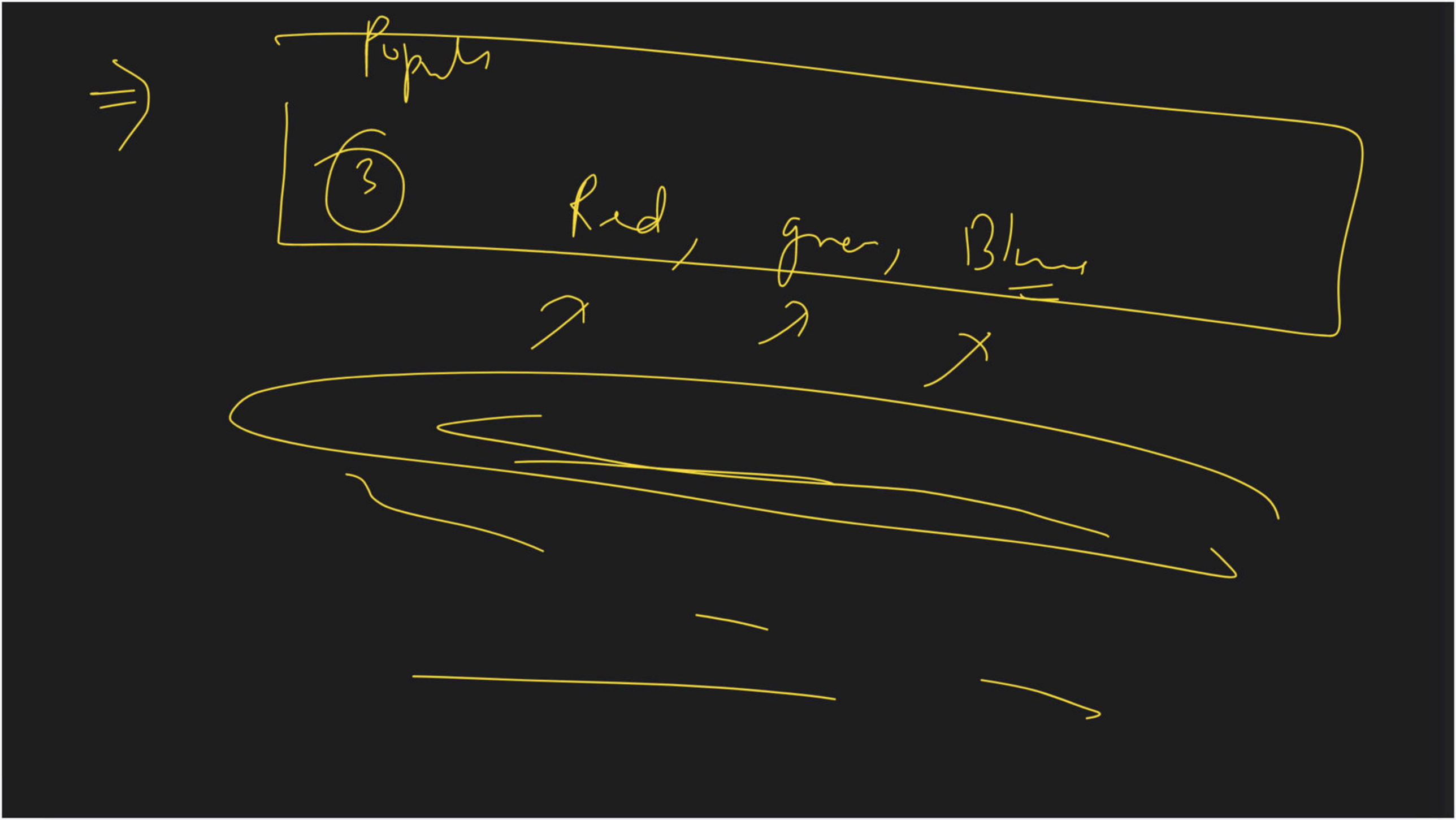
(Consistency)


M
WebDX
T
DSA
W
T
DSA
F
DSA
S
S
DsA
Push
3 →
8 – 5
Push → Do
Non-Push


Big Firm
React
DSA
Fibum
WebD
DSA
50%
50


if(n[i] == -1)
n[i] = size
4
2
0
-1 0 1 2 -1
A2 = -1 x 2 = -2
A2 max = -16
max
-2
next →
prev →
w →
-1 | -1
-1 | 2
-1 | -4
-1 - (-1) - 1
-1 -1 -1
-1 - 2 - 1
= -4


DSA
Problem solving
Netflix
Compound
computer
se kese karaya
sorting


Pupils
3
Red, green, Blue

Red

Bulb

Green

Sort()

nlogn

=>

Code