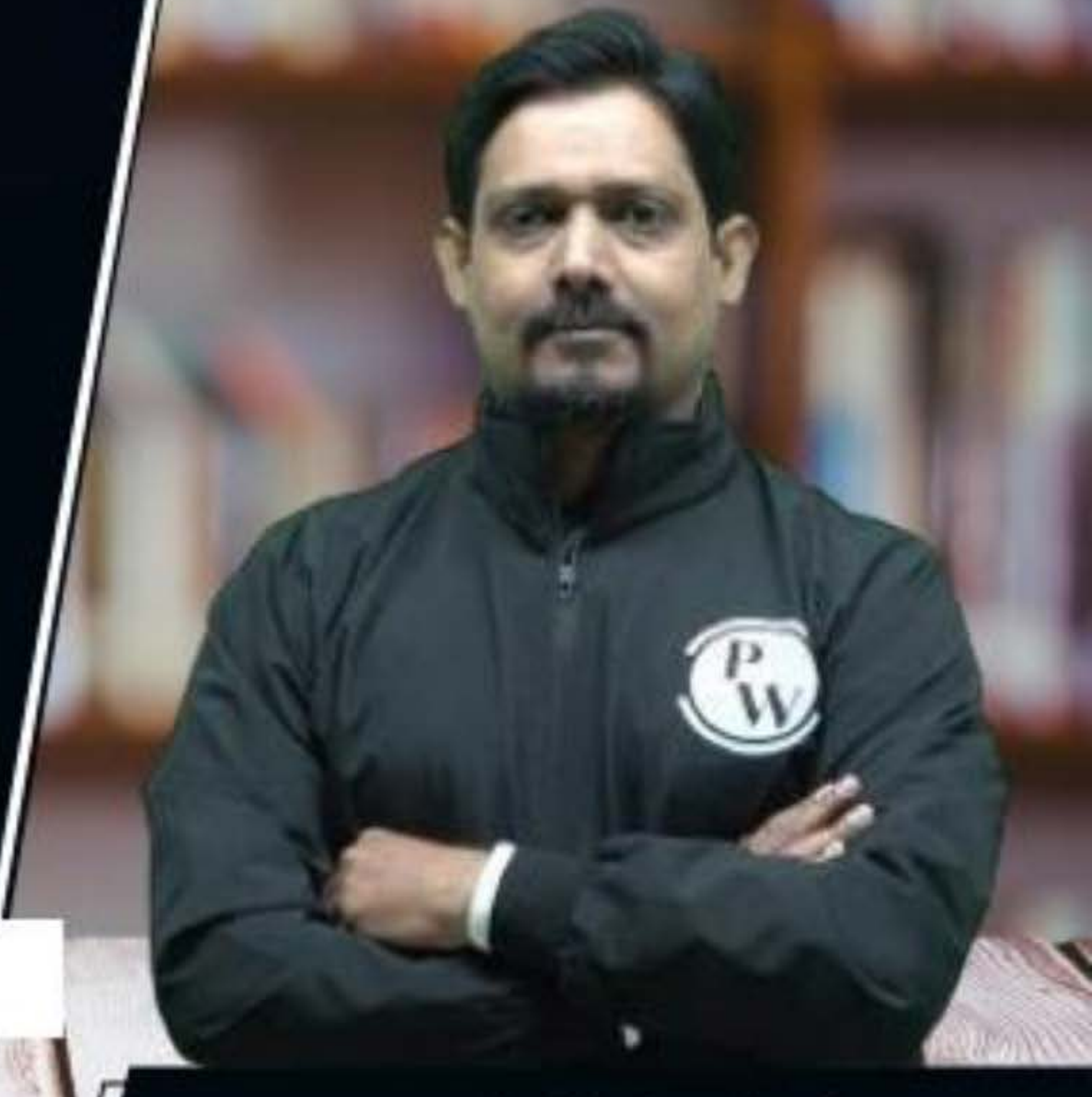
Computer Science & IT

Data Structure & programming

Linked List

Lecture No. 05



By- Abhishek Sir

Recap of Previous Lecture

Topic

Double linked List

Topic

Addbegin. Addend. getnode. build 123

Topic

Stack using linked List

Topic

Topic

Slide

Topics to be Covered

Topic

Topic

Topic

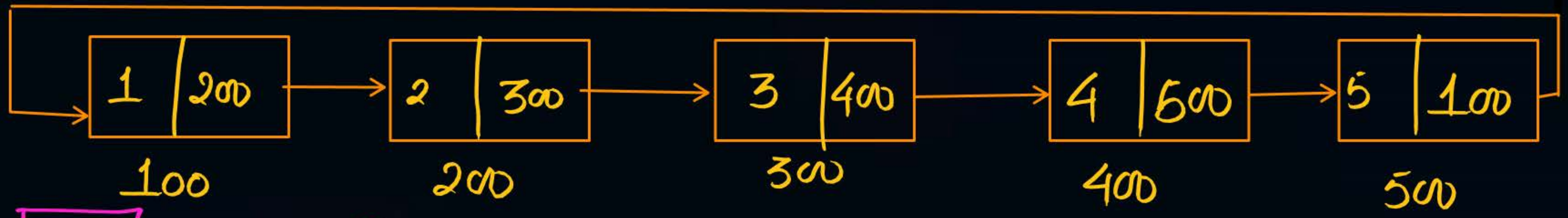
Topic

Topic

Circular Linked List

Slide

Topic : Circular inked List



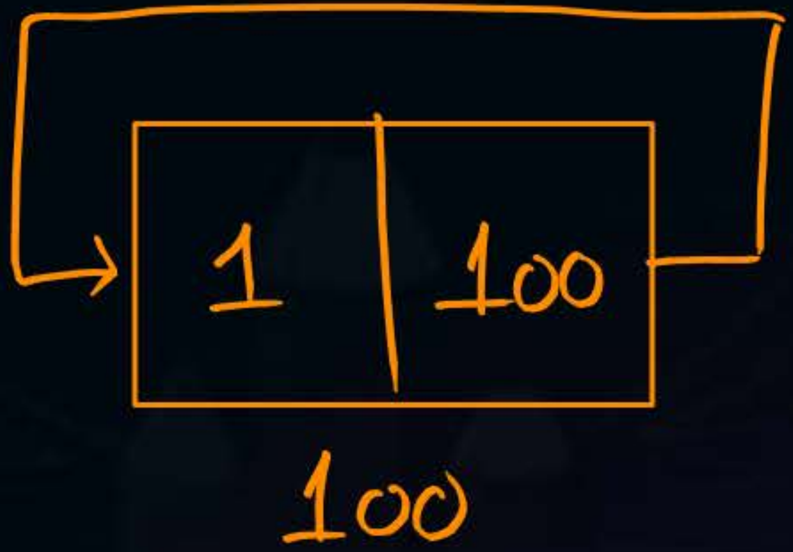
HP = $\boxed{100}$

Single Linked List.

Slide

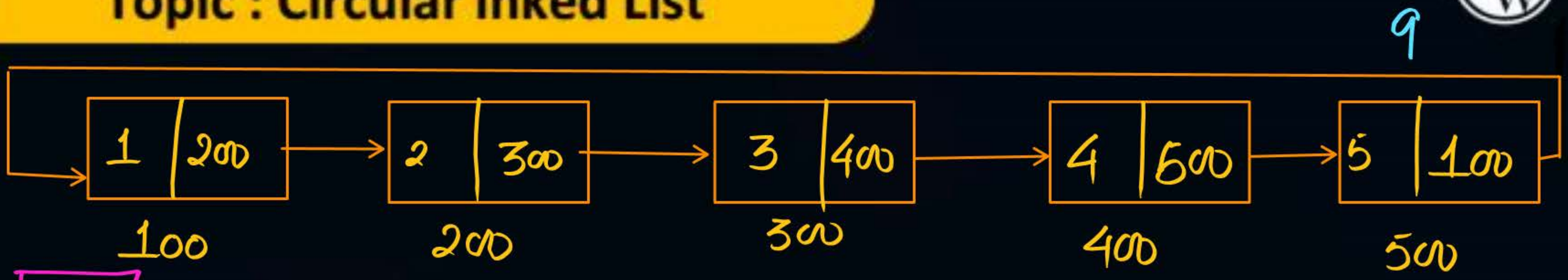
Topic : Circular inked List

```
void display() {
    Node* q = HP;
    if(q) {
        while(q→next != HP) {
            printf("%d", q→data);
            q = q→next;
        }
        printf("%d", q→data); // ← to print Last value
    }
    else {
        printf(" Empty List ");
    }
}
```



Slide

Topic : Circular inked List



HP = $\boxed{100}$

Add begin

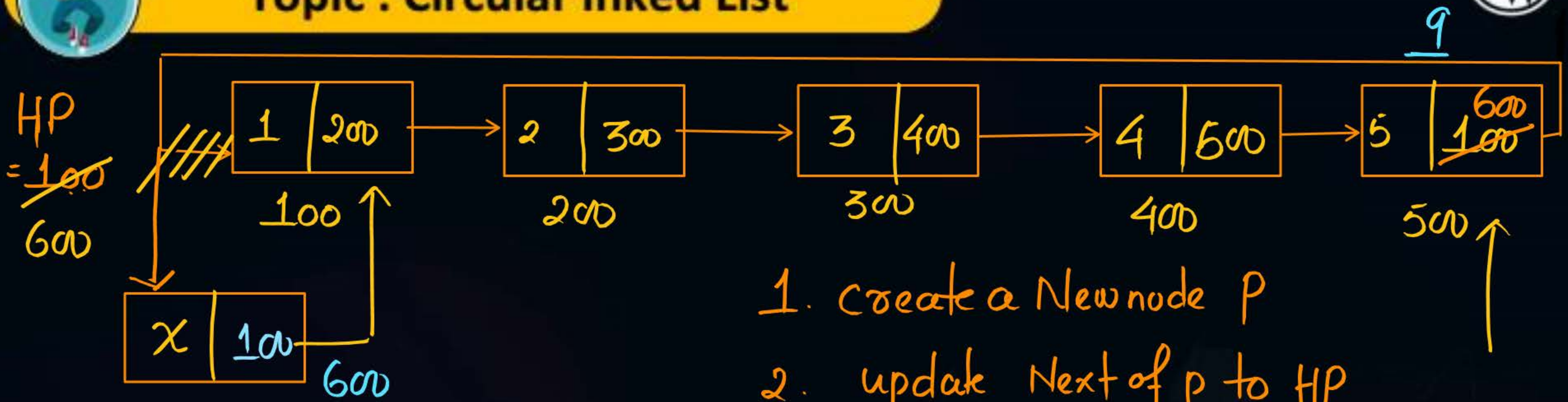
Add end

which code will depends upon length of list?

both

Slide

Topic : Circular inked List



P

HP will be updated to 600 (Address of p)

1. create a New node p
2. update Next of p to HP
3. Traverse till Last Node
4 update Next pointer last Node to p
5 update HP

Slide

Topic : Circular inked List

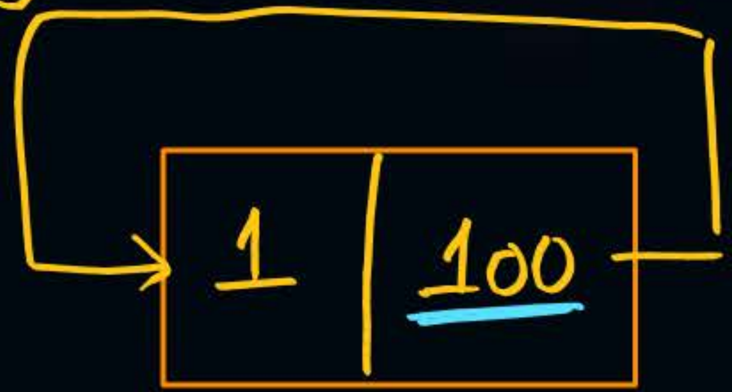
```
void  Addbegin ( int x) {
  Node* p = getnode(x); //Single Linked List
  Node*q = HP;
  if(HP == NULL){
        HP = p;
        p→next = HP;
  }     return;
```

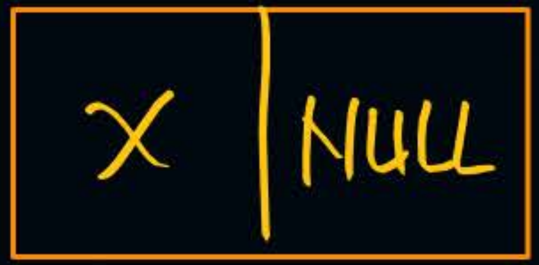
```
while(q→next != HP)
        q = q→next

   p→next = HP;
   q→next = p;
   HP = p;
}
```

Slide

Topic : Circular inked List

Single Node CLL





100

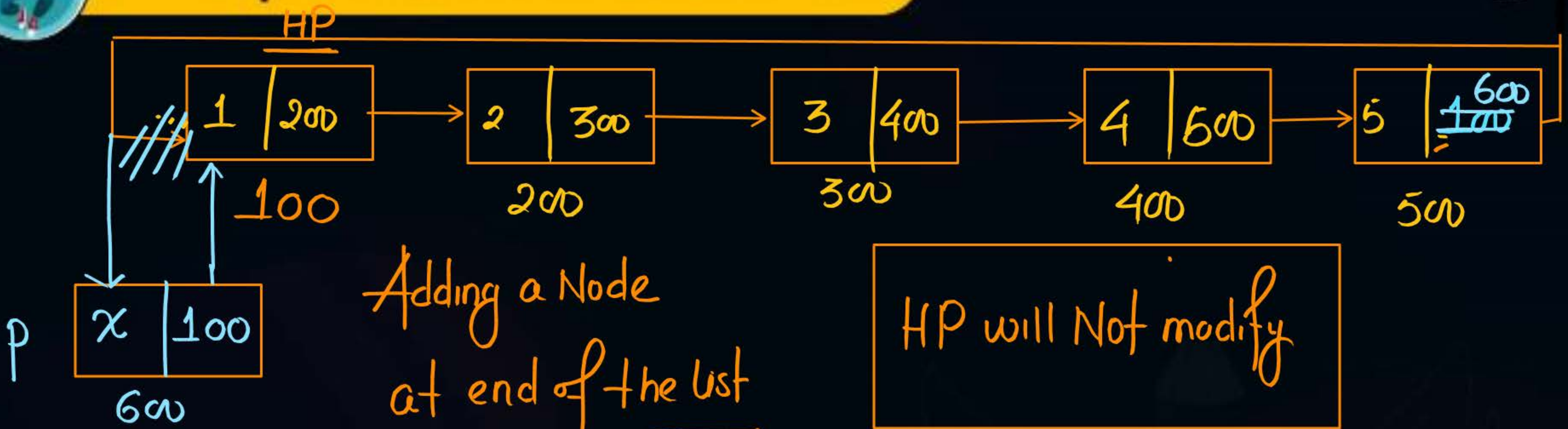
getnode(x)

HP = 100

p→next =

Slide

Topic : Circular inked List



Adding a Node at end of the list

HP will Not modify

Same code will be used as Add begin

Slide

Topic : Circular inked List

```
void Addend (int x) {
  Node* p = getnode(x); //Single Linked List
  Node *q = HP;
  if(HP == NULL){
        HP = p;
        p→next = HP;
  }     return;

  while(q→next != HP)
        q = q→next

    p→next = HP;
    q→next = p;

}
```

Slide

Question

A circular queue has been implemented using a single linked list where each node consists of a value and a single pointer pointing to the next node. We maintain exactly two external pointers **FRONT** and **REAR** pointing to the front node and the rear node of the queue, respectively. Which of the following statements is/are **CORRECT** for such a circular queue, so that insertion and deletion operation can be performed in O (1) time ? Independent of length

I. Next pointer of front node points to the rear node.

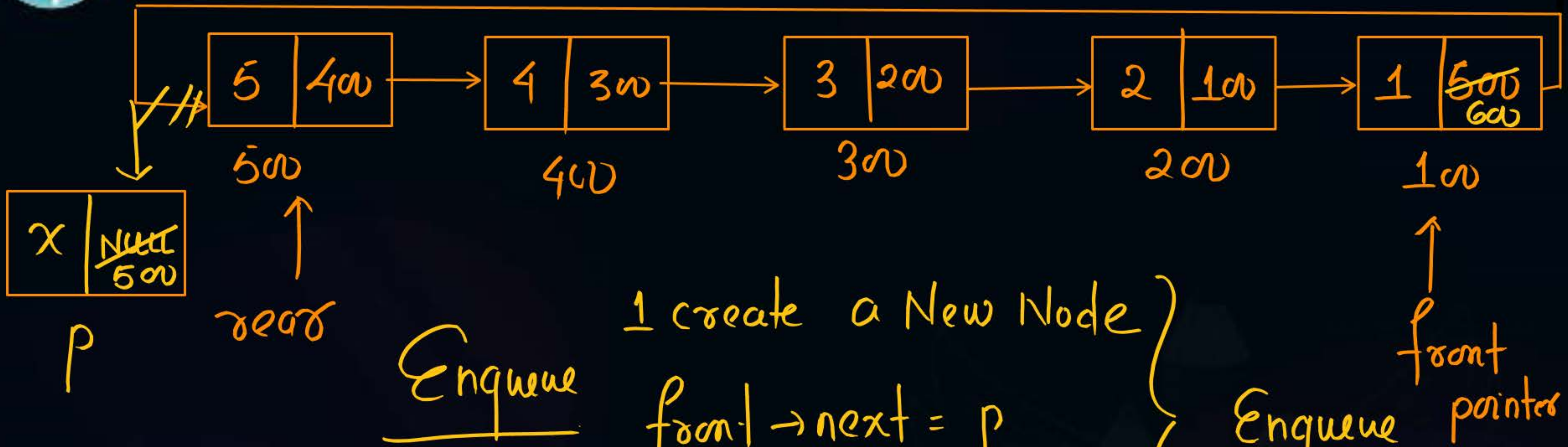
II. Next pointer of rear node points to the front node. ✓ II

(A) I only [✓ (B) II only]

(C) Both I and II (D) Neither I nor II

Slide

Topic : Circular inked List



Enqueue

1 create a New Node

front → next = p

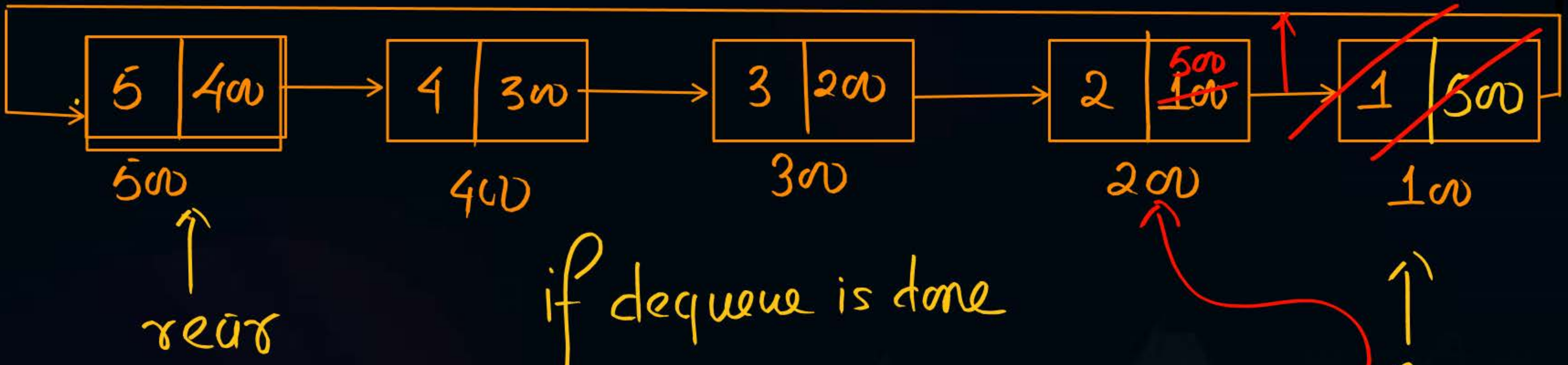
p → next = rear

rear = p

Enqueue

Constant O(1)

Topic : Circular inked List



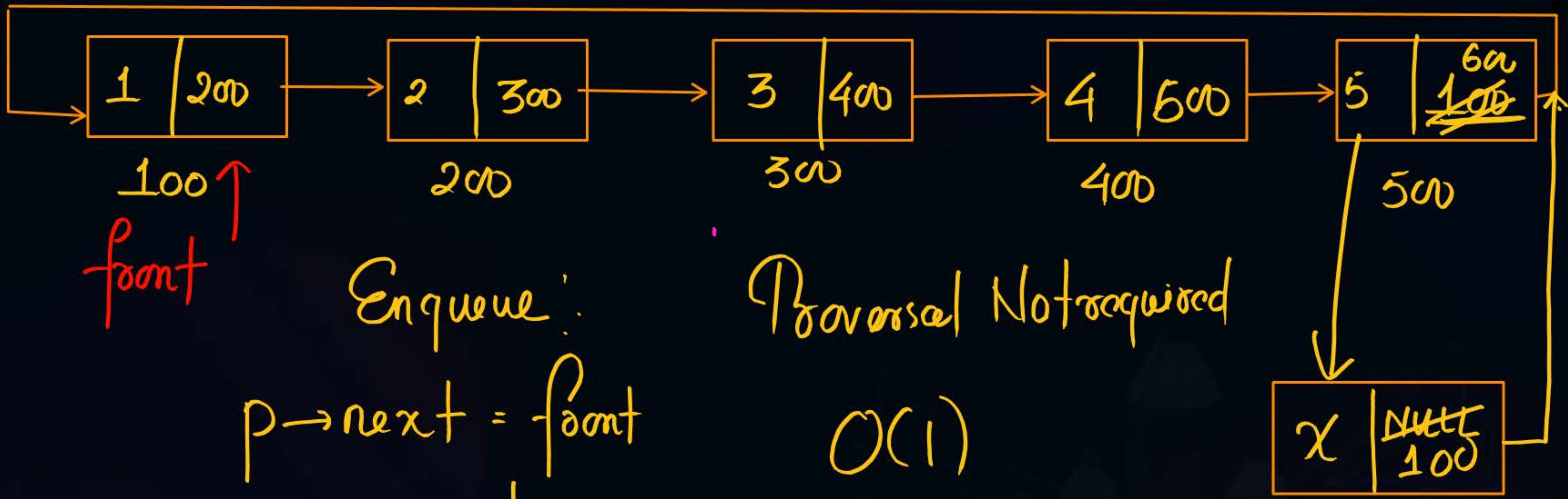
if dequeue is done

front pointer updated to 2

traversal required   Not O(1)

Slide

Topic : Circular inked List



Enqueue:

p→next = front

rear→next = p

rear = p

Traversal Notrequired

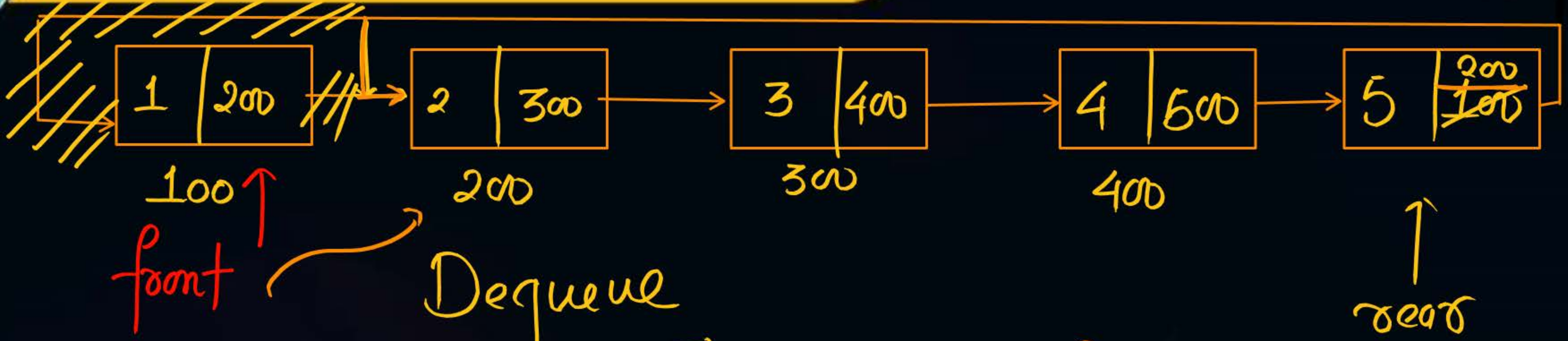
O(1)

Constant time

rear ↑ P 600

Slide

Topic : Circular inked List



Dequeue

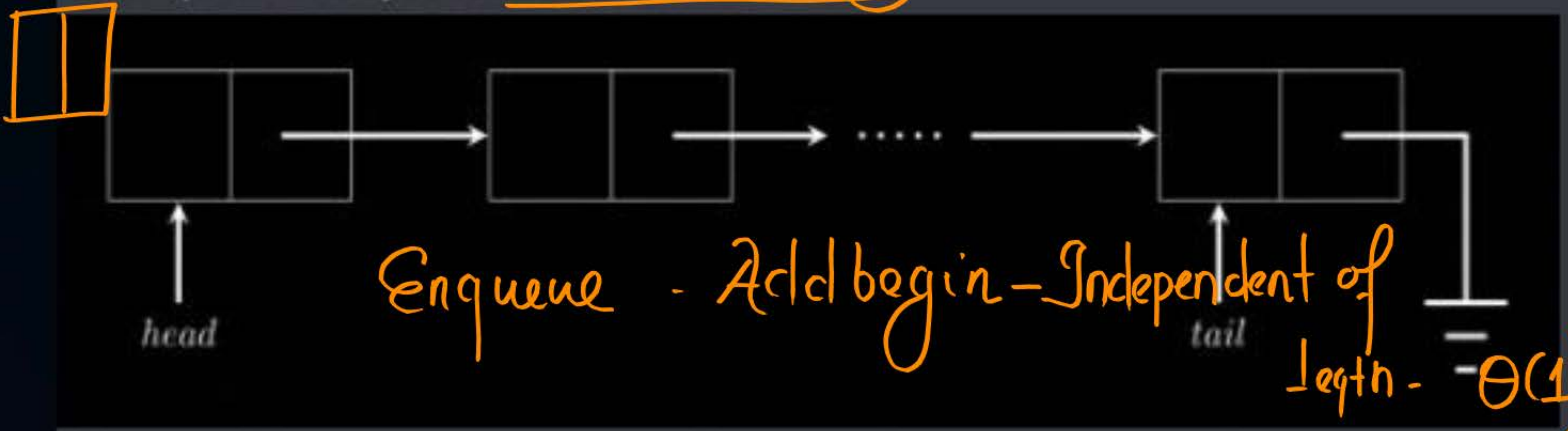
Traversal Not required O(1)

```
rear → next = front → next
free(front)
front = rear. next
```

Slide

Question

A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head, and 'dequeue' be implemented by deletion of a node from the tail.



Enqueue - Add begin - Independent of length - $\Theta(1)$

Which one of the following is the time complexity of the most time-efficient implementation of 'enqueue' and 'dequeue, respectively, for this data structure?

A. $\Theta(1), \Theta(1)$

B. $\Theta(1), \Theta(n)$

C. $\Theta(n), \Theta(1)$

D. $\Theta(n), \Theta(n)$

Slide

Question

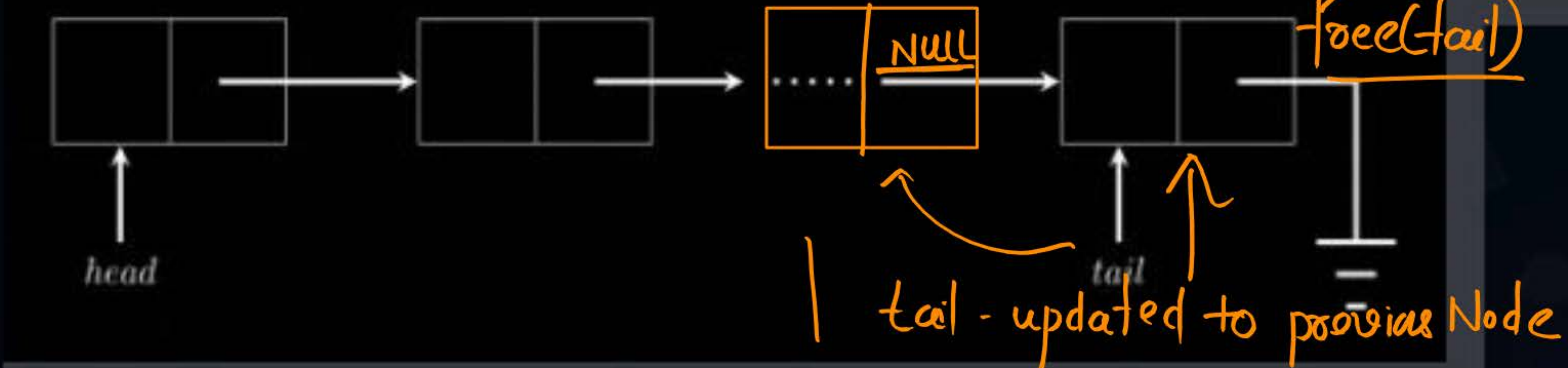
A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head, and 'dequeue' be implemented by deletion of a node from the tail.

A. $\Theta(1), \Theta(1)$

[✓ B. $\Theta(1), \Theta(n)$]

C. $\Theta(n), \Theta(1)$

D. $\Theta(n), \Theta(n)$



Which one of the following is the time complexity of the most time-efficient implementation of 'enqueue' and 'dequeue, respectively, for this data structure?

Slide

Single Linked List Enqueue — $\Theta(1)$

dequeue — $\Theta(n)$

Double Linked List Enqueue - $\Theta(1)$

dequeue - $\Theta(1)$

Question

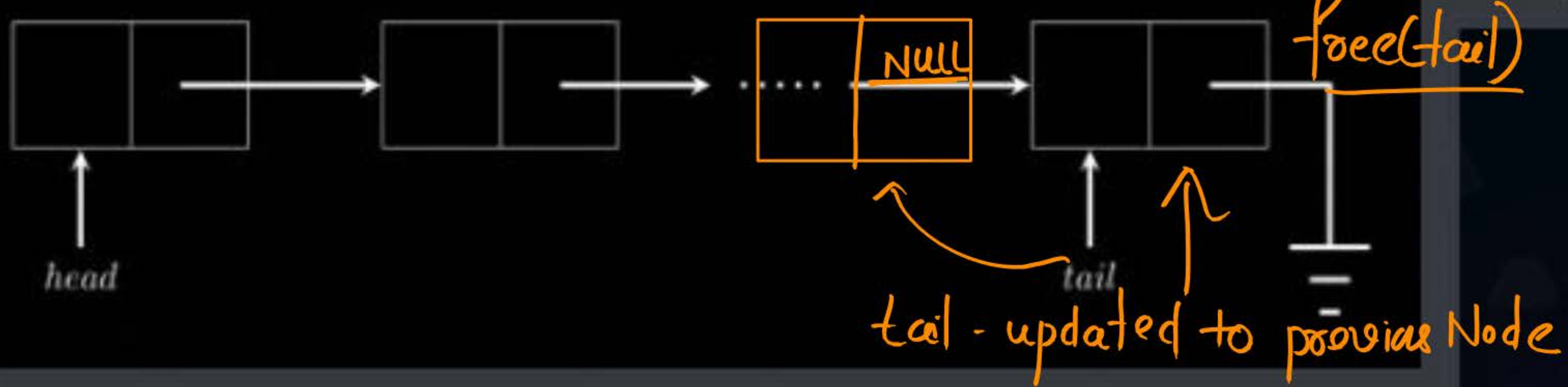
A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head, and 'dequeue' be implemented by deletion of a node from the tail.

A. $\Theta(1), \Theta(1)$

[✓ B. $\Theta(1), \Theta(n)$]

C. $\Theta(n), \Theta(1)$

D. $\Theta(n), \Theta(n)$



Which one of the following is the time complexity of the most time-efficient implementation of 'enqueue' and 'dequeue, respectively, for this data structure?

Slide

Question

Consider the C program below

```
#include <stdio.h>
int *A, stkTop;
int stkFunc (int opcode, int val){
    static int size=0, stkTop=0;
    switch (opcode) {
        case -1: size = val; break;
        case 0: if (stkTop < size ) A[stkTop++]=val;
                break;
        default: if (stkTop) return A[--stkTop];
    }
    return -1;
}
```

-1, 10

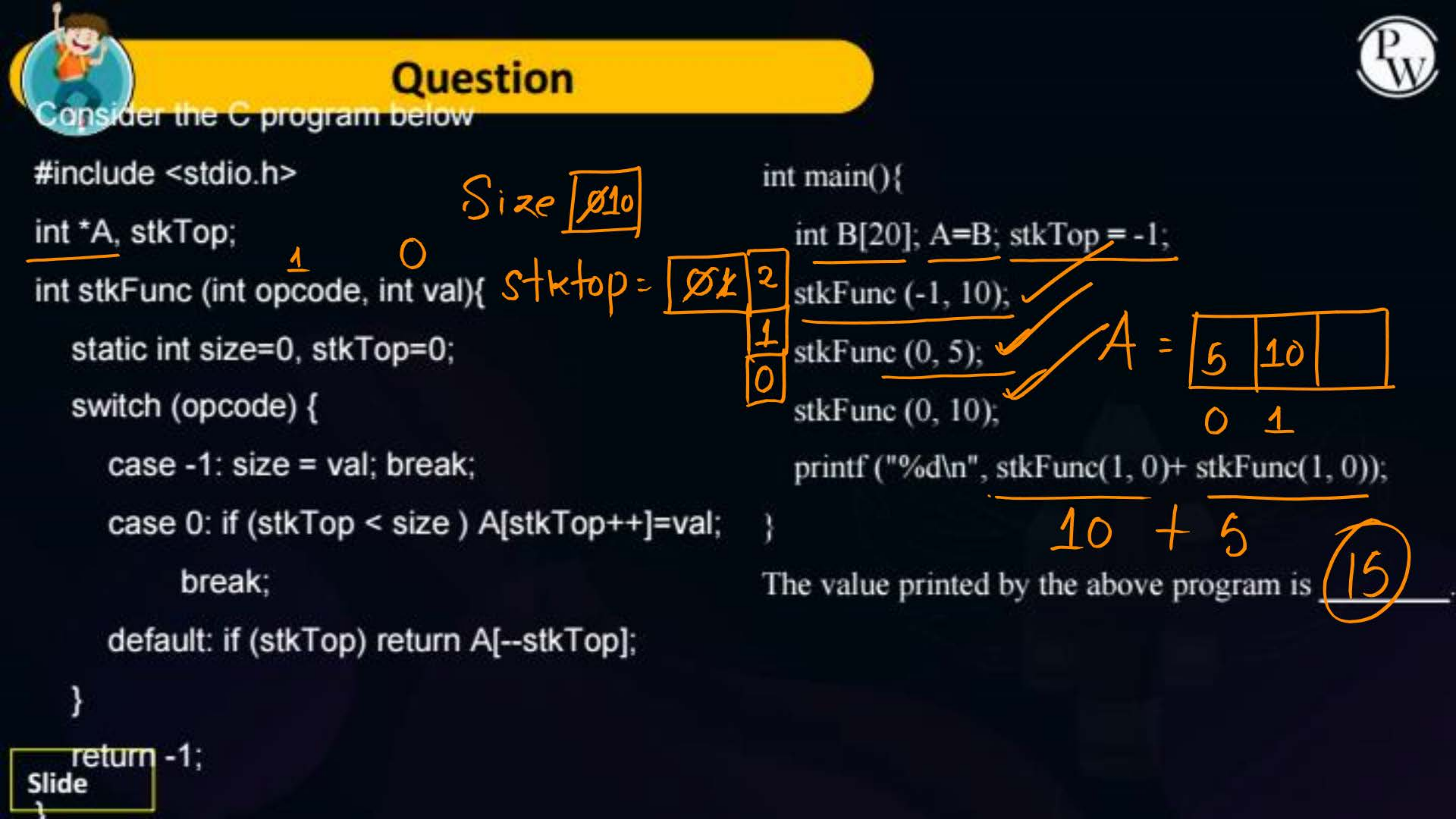
Size $\boxed{\cancel{0}10}$

stktop = $\boxed{0}$

```
int main(){
    int B[20]; A=B; stkTop = -1;
    stkFunc (-1, 10);
    stkFunc (0, 5);
    stkFunc (0, 10);
    printf ("%d\n", stkFunc(1, 0)+ stkFunc(1, 0));
}
```

The value printed by the above program is ________.

Slide



Question

Consider the C program below

```
#include <stdio.h>
int *A, stkTop;
int stkFunc (int opcode, int val){
    static int size=0, stkTop=0;
    switch (opcode) {
        case -1: size = val; break;
        case 0: if (stkTop < size ) A[stkTop++]=val;
                break;
        default: if (stkTop) return A[--stkTop];
    }
    return -1;
}
```

```
int main(){
    int B[20]; A=B; stkTop = -1;
    stkFunc (-1, 10);
    stkFunc (0, 5);
    stkFunc (0, 10);
    printf ("%d\n", stkFunc(1, 0)+ stkFunc(1, 0));
}
```

The value printed by the above program is $\underline{15}$.

1
0
Size $\boxed{\cancel{0}10}$
stktop = $\boxed{\cancel{0}\cancel{1}}$ 2 1 0
A = 5 10
0 1
10 + 5
15
Slide

Question

Consider the C program below

```
#include <stdio.h>
int *A, stkTop;
int stkFunc (int opcode, int val){
    static int size=0, stkTop=0;
    switch (opcode) {
        case -1: size = val; break;
        case 0: if (stkTop < size ) A[stkTop++]=val;
                break;
        default: if (stkTop) return A[--stkTop];
    }
    return -1;
```

Slide

2 mins Summary

Topic

Topic

Topic

Topic

Topic

Circular Linked List

queue Implementation of CLL

Slide

Tree

THANK - YOU

Slide