Doubly Linked List

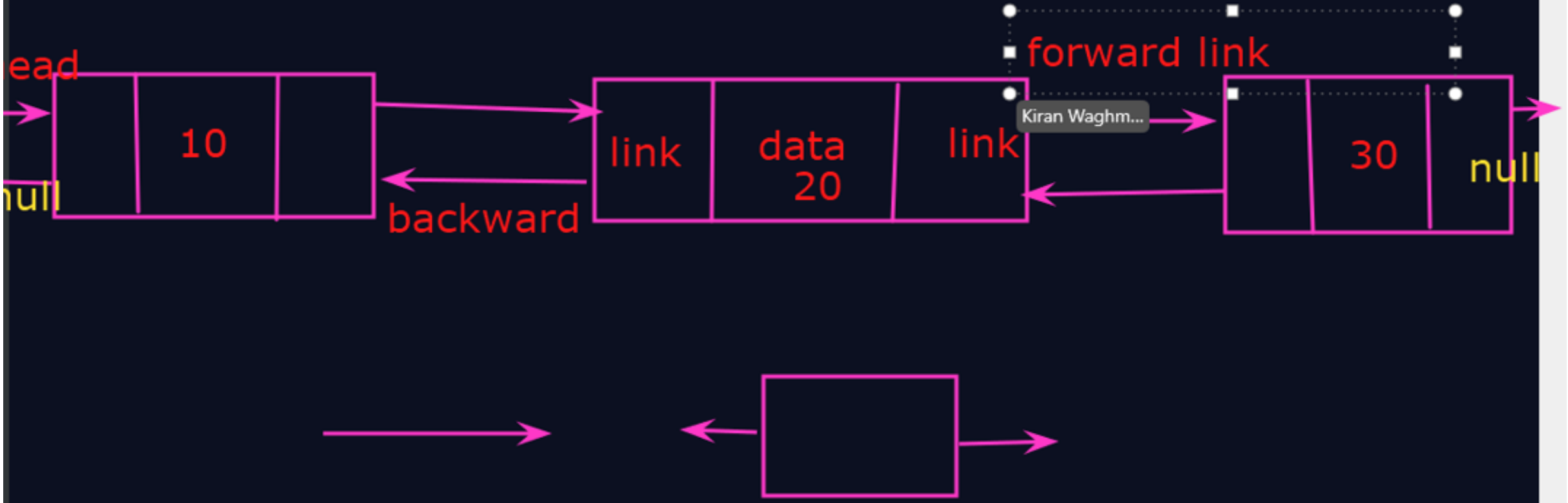
Kiran Waghmare

Doubly Linked List:

-DLL is a variation in Linked List.

-Navigation in both direction.

-forward and backward direction.

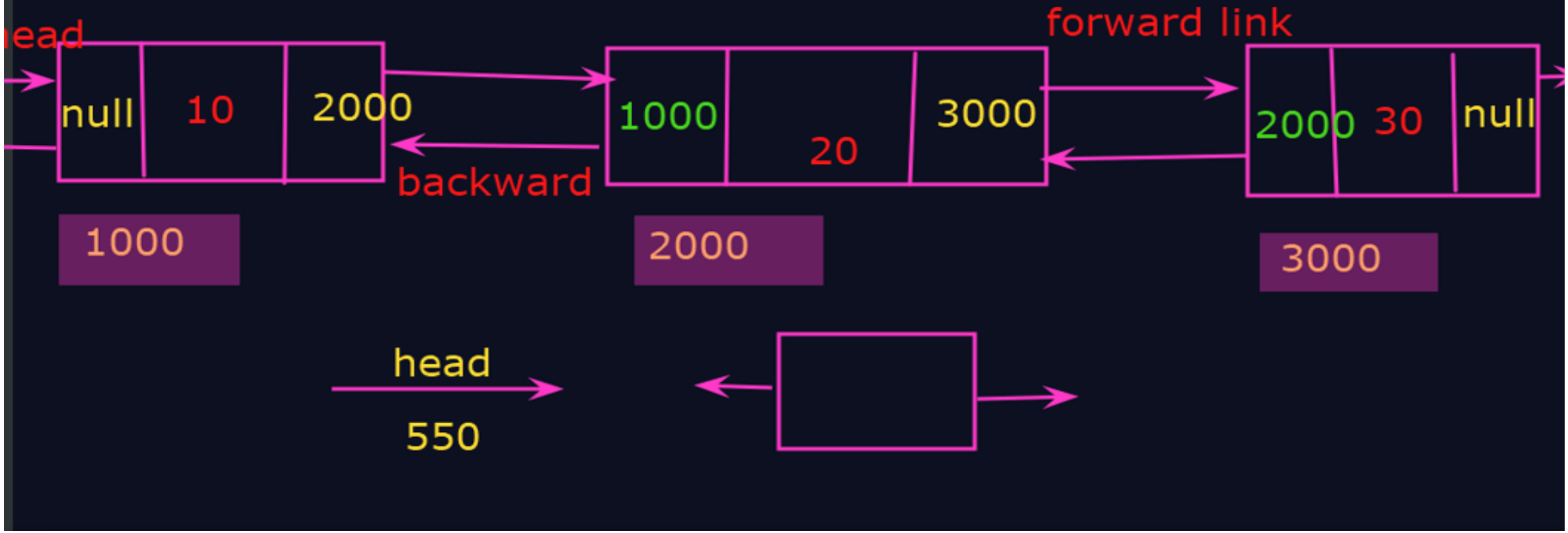


Doubly Linked List:

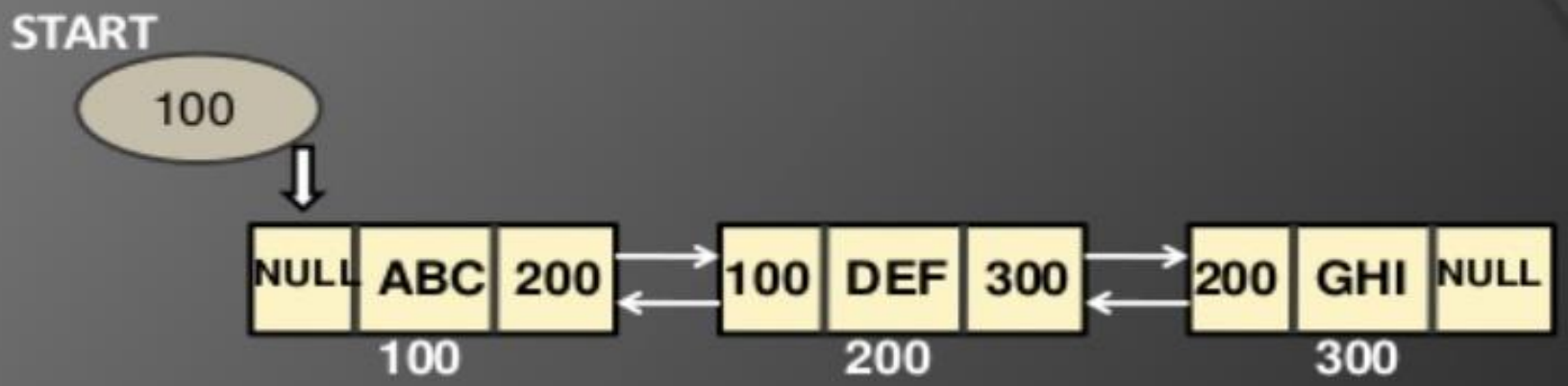
-DLL is a variation in Linked List.

-Navigation in both direction.

-forward and backward direction.



DOUBLY LINKED LIST

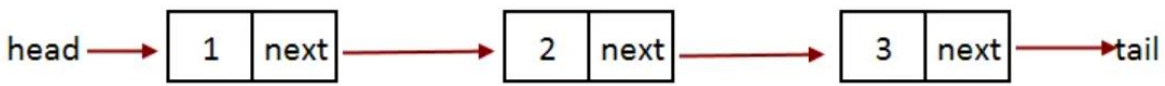


Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List.

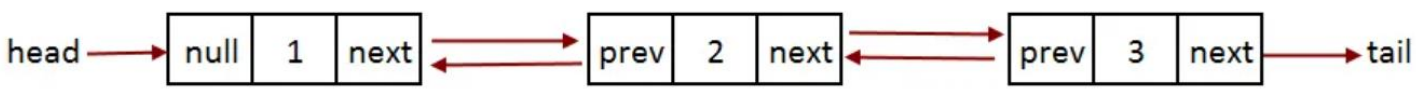
SS

Singly Linked List vs Doubly Linked List

Singly Linked List	Doubly Linked List
Easy Implement	Not easy
Less memory	More Memory
Can traverse only in forward direction	Traverse in both direction, back and froth



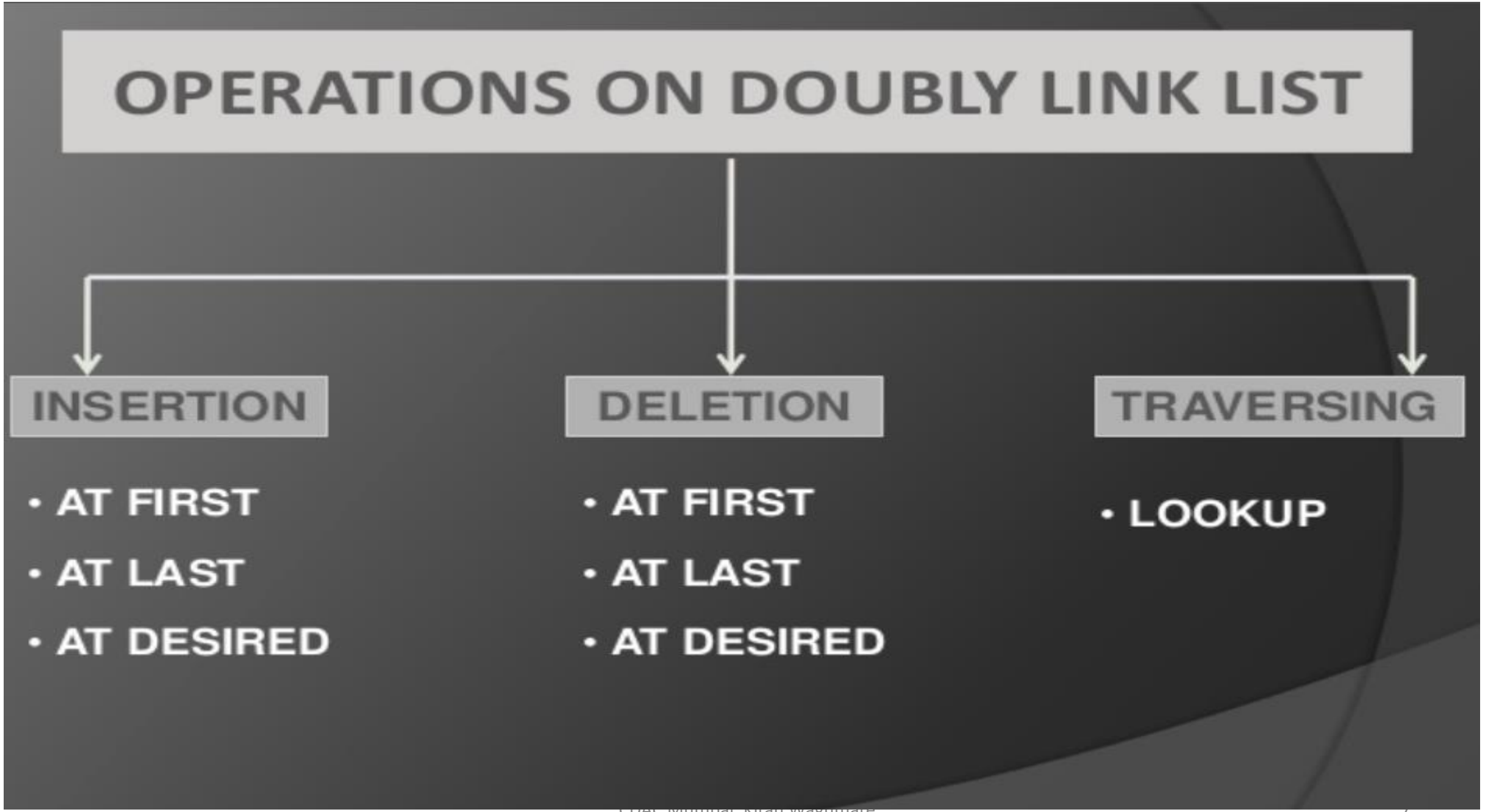
Singly Linked List

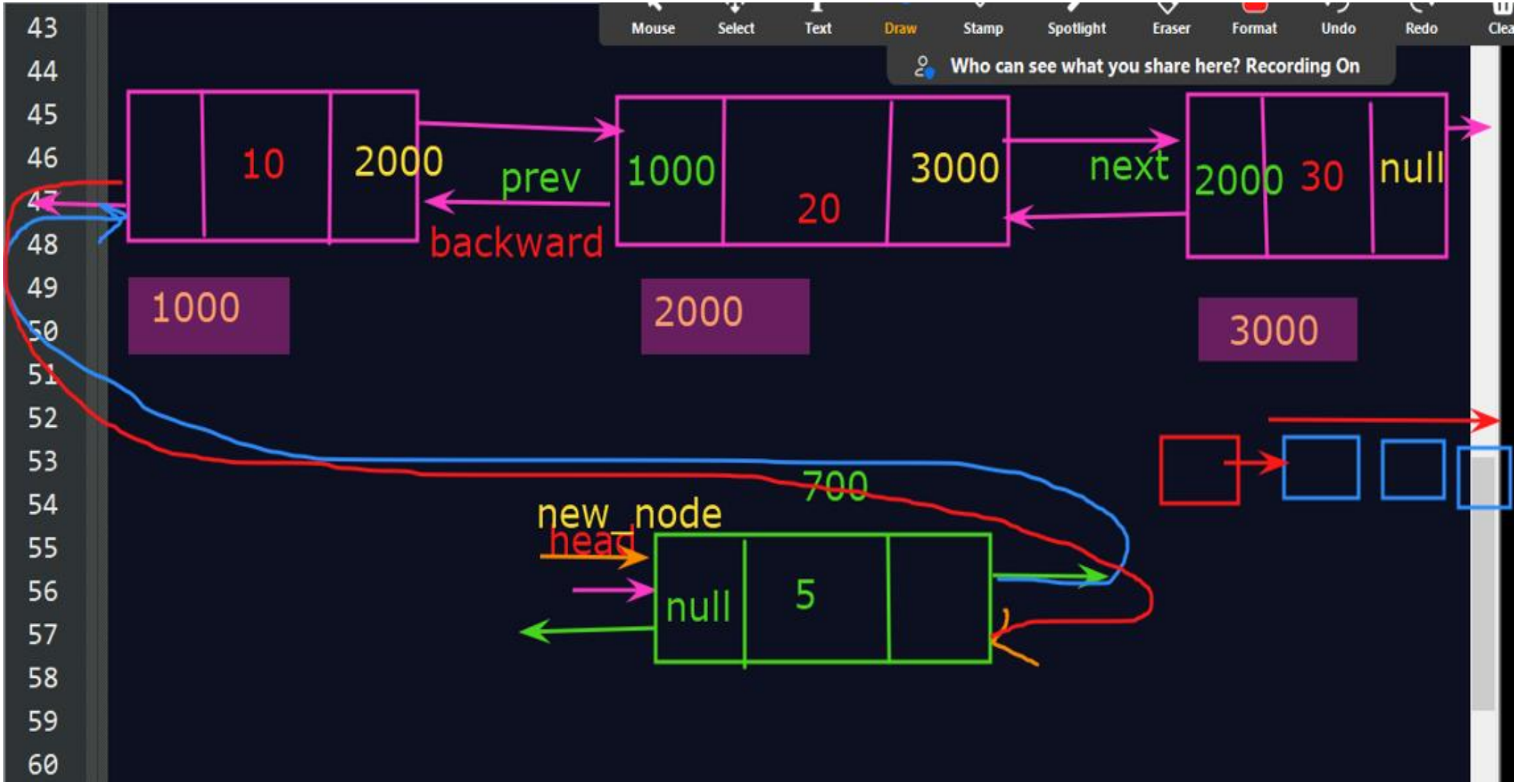


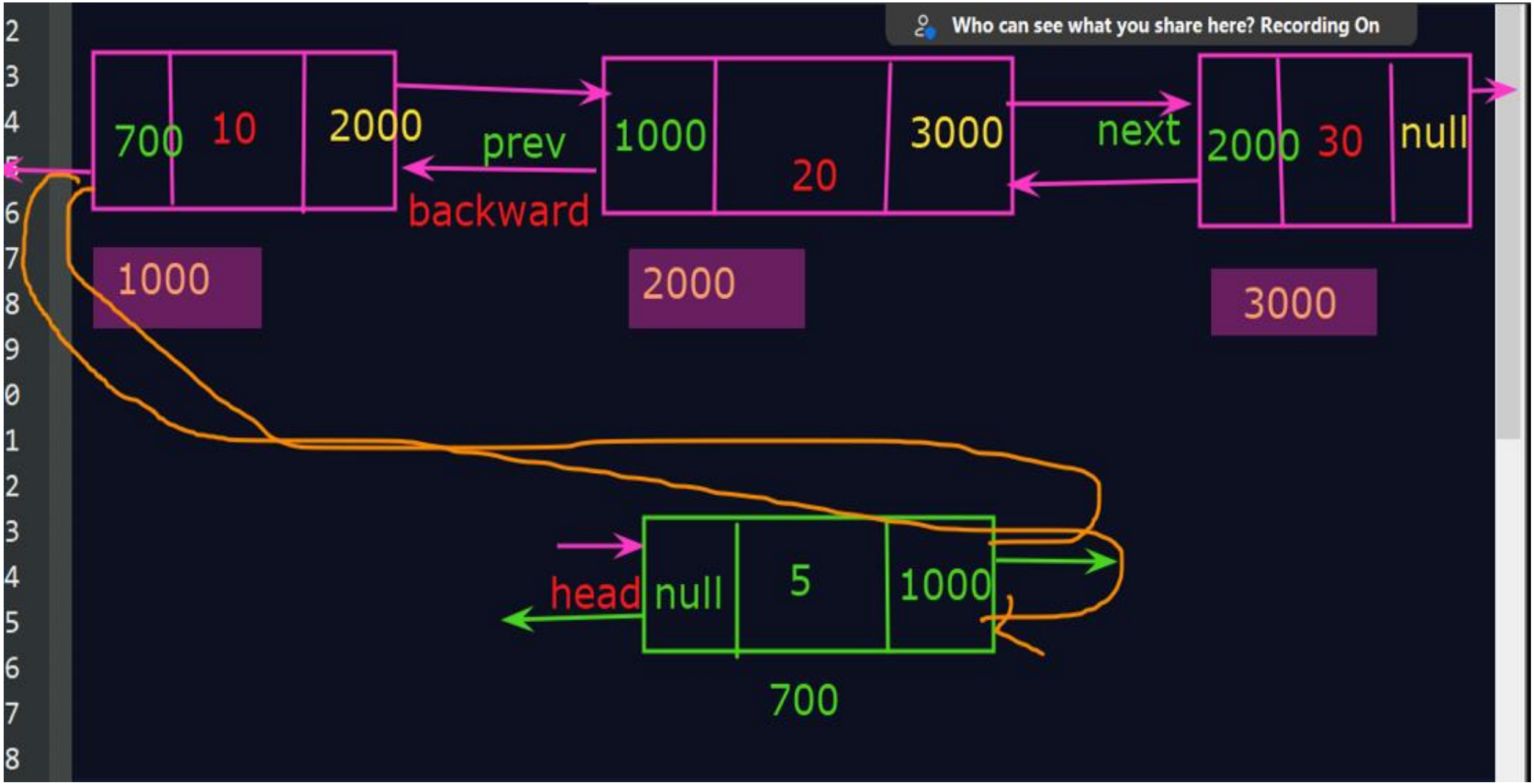
Doubly Linked List

Why Doubly linked list ?

- In singly linked list we cannot traverse back to the previous node without an extra pointer. For ex to delete previous node.
- In doubly there is a link through which we can go back to previous node.

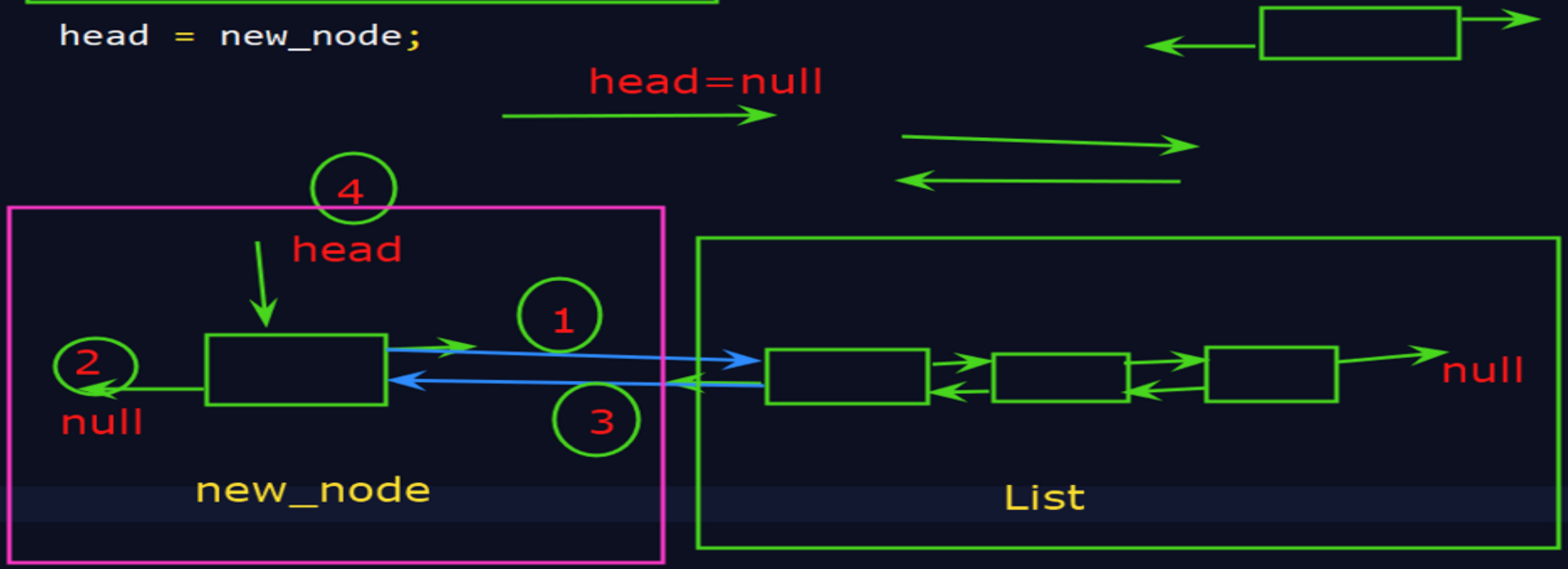

OPERATIONS ON DOUBLY LINK LIST
INSERTION
DELETION
TRAVERSING
• AT FIRST
• AT LAST
• AT DESIRED
• AT FIRST
• AT LAST
• AT DESIRED
• LOOKUP

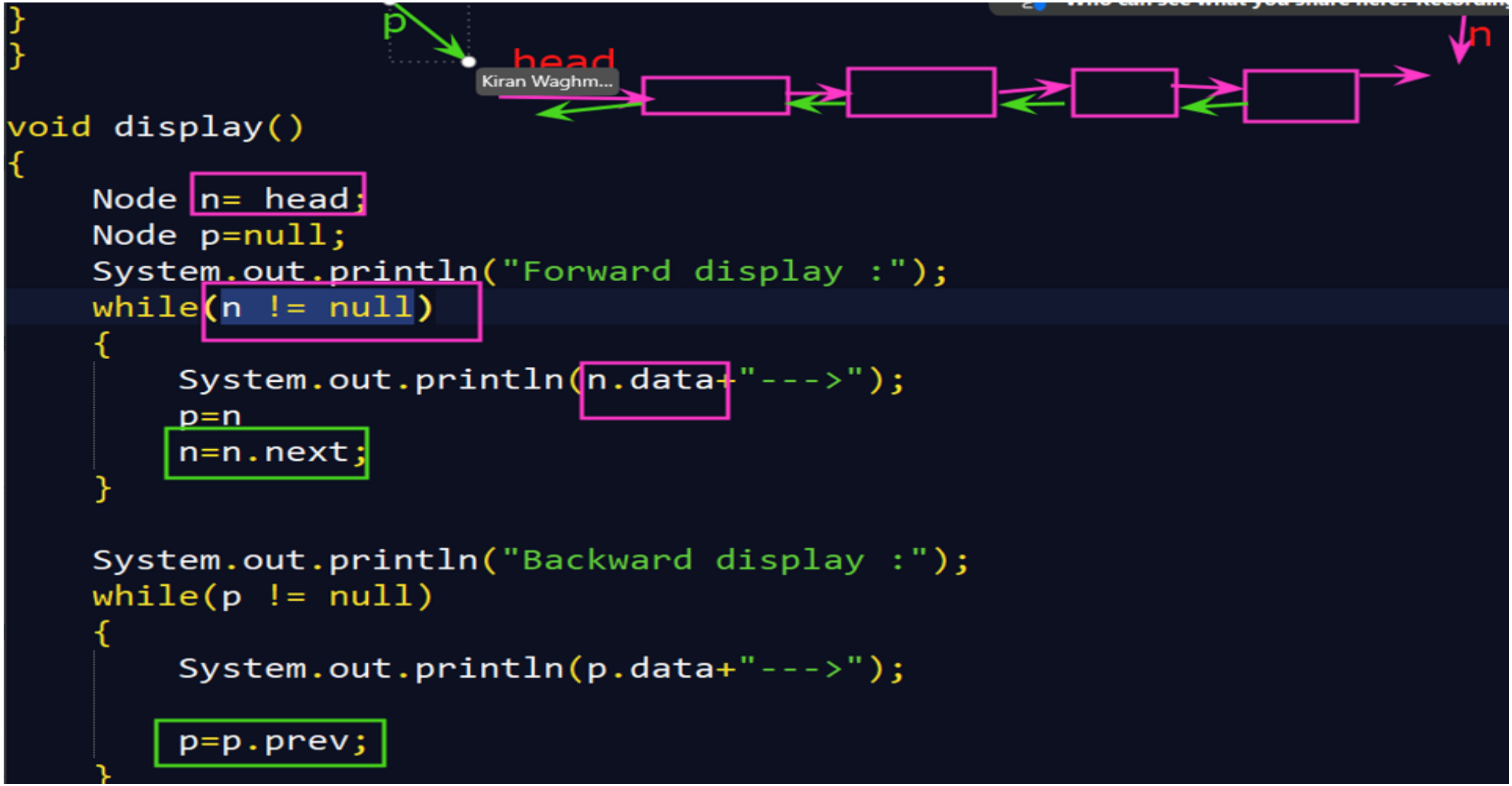

Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Who can see what you share here? Recording On
10
2000
prev
backward
1000
20
3000
next
2000
30
null
1000
2000
3000
700
new node
head
null
5


Who can see what you share here? Recording On
700
10
2000
prev
backward
1000
20
3000
next
2000
30
null
1000
2000
3000
head
null
5
1000
700

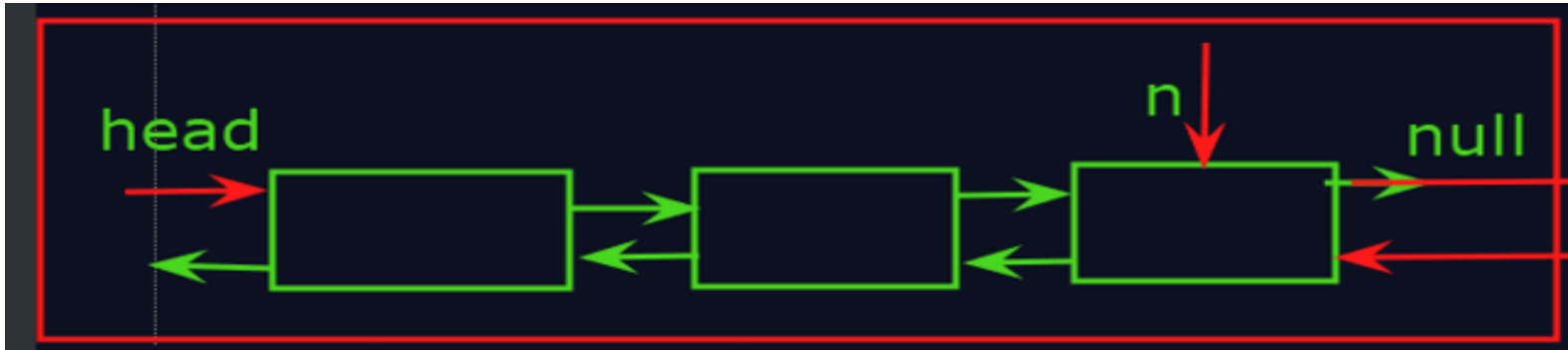
```
class DLL1
{
    Node head;
static class Node
{
    int data;
    Node prev;
    Node next;

    Node(int d)
    {
        data=d;
        prev = next = null;
    }

}


public void insert(int new_data)
{
    Node new_node = new Node(new_data);
    new_node.next = head;
    new_node.prev = null;

    if(head != null)
```

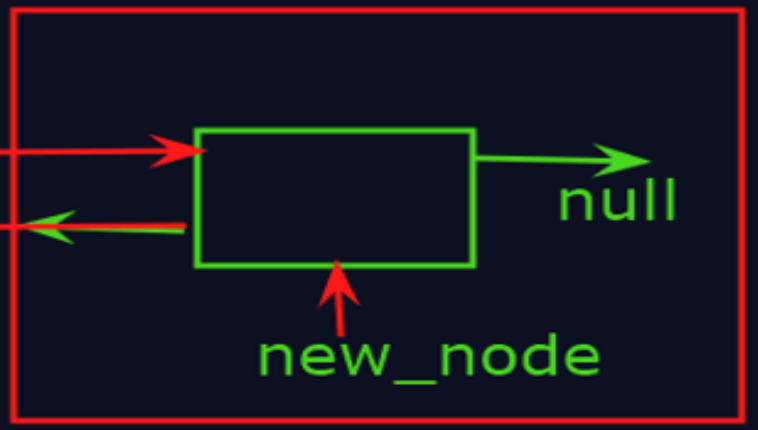
```
public void insert(int new_data)
{
    Node new_node = new Node(new_data);
    new_node.next = head;
    new_node.prev = null;

    if(head != null)
        head.prev = new_node;

    head = new_node;

}
}
```



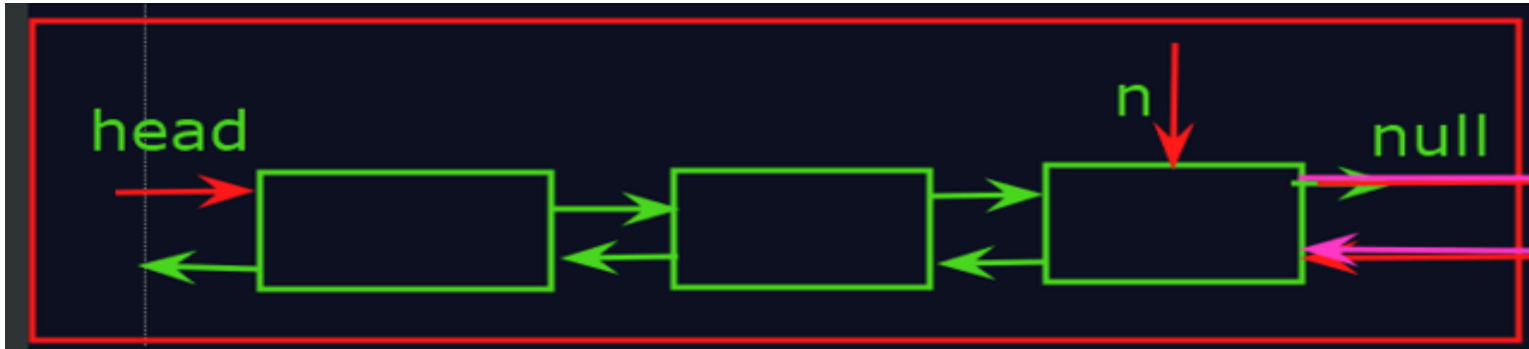


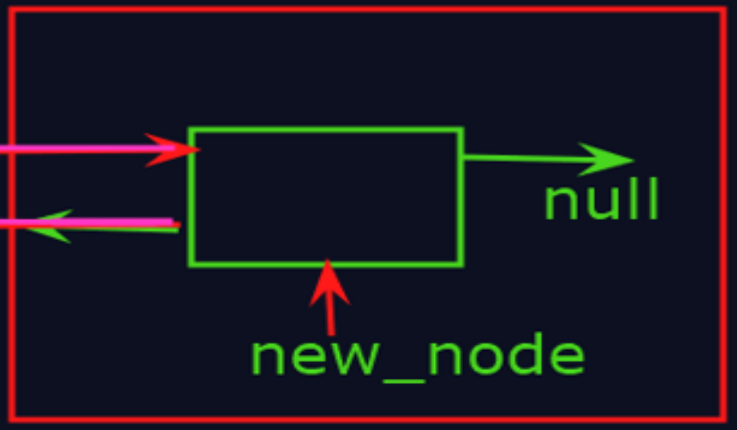

```
}
}

void display()
{
    Node n= head;
    Node p=null;
    System.out.println("Forward display :");
    while(n != null)
    {
        System.out.println(n.data+"--->");
        p=n
        n=n.next;
    }

    System.out.println("Backward display :");
    while(p != null)
    {
        System.out.println(p.data+"--->");

        p=p.prev;
    }
```



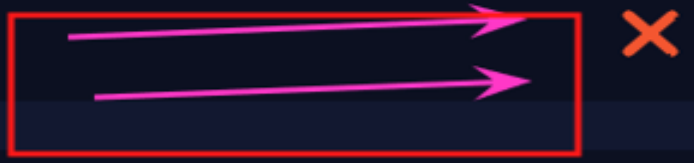


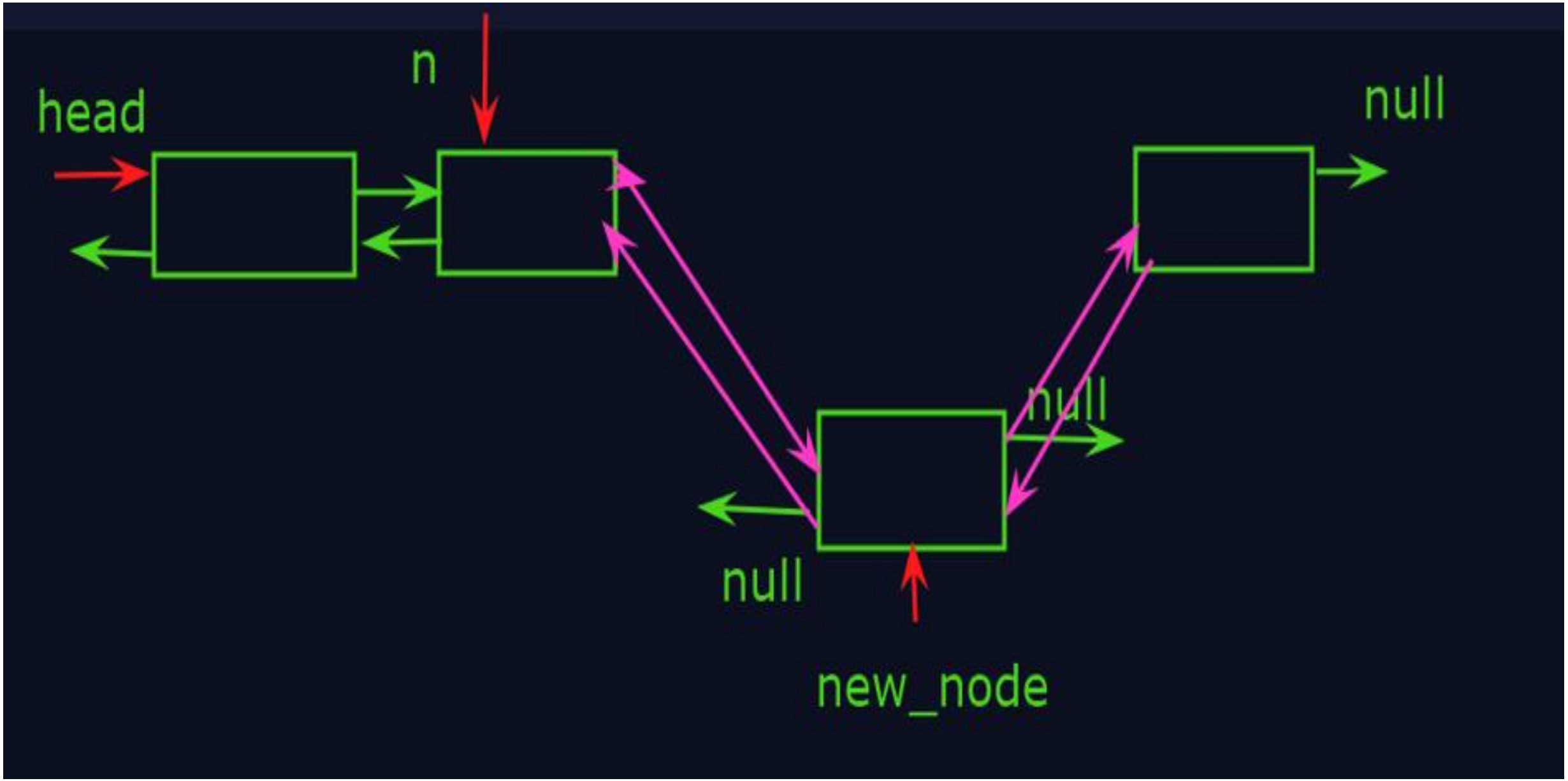

```
void append(int new_data)
{
    Node new_node = new Node();
    Node n = head;
    new_node.next = null;

    if(head == null)
    {
        new_node.prev = null;
        head = new_node;
        return;
    }

    while(n.next != null)
        n=n.next;

    n.next = new_node;
    new_node.prev =n;
}
```

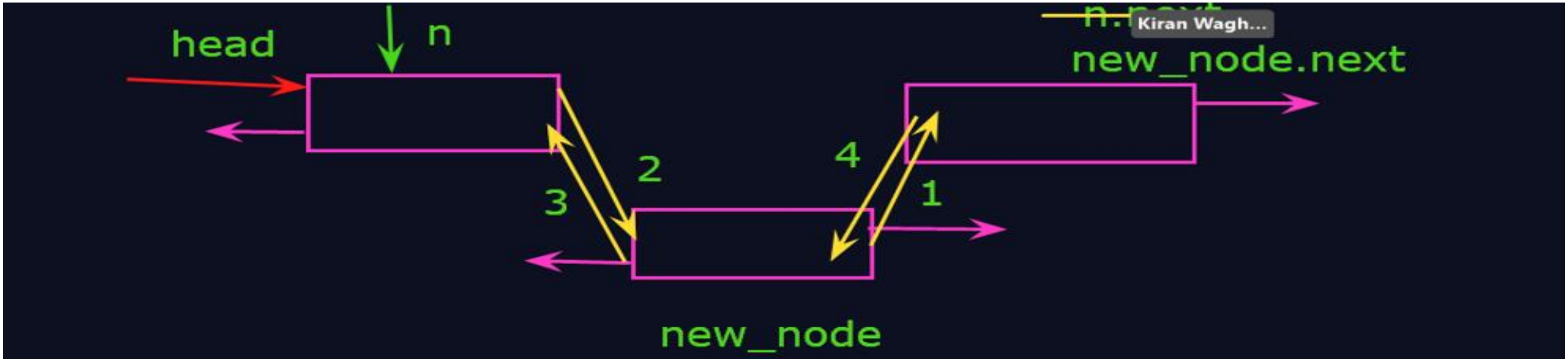





```
void append(int new_data)
{
    Node new_node = new Node();
    Node n = head;
    new_node.next = null;

    if(head == null)
    {
        new_node.prev = null;
        head = new_node;
        return;
    }

    while(n.next != null)
        n=n.next;

    n.next = new_node;
    new_node.prev =n;
}
```




n
head
null
null
null
new_node




```
void insertAfter(Node n, int new_data)
{
    n = head;
    Node new_node = new Node(new_data);
    new_node.next = n.next;
    n.next = new_node;
    new_node.prev = n;
    if(new_node.next != null)
        new_node.next.prev = new_node;
```

```
    d1.insert(11);
    d1.insert(12);
    d1.insert(13);
    d1.insert(14);
    d1.append(15);
    d1.insertAfter(d1.head,
    d1.display(d1.head);


}

}
```

```
C:\Test>java DLL1
Forward display :
14--->13--->12--->11--->15--->
Backward display :
15--->11--->12--->13--->14--->
C:\Test>javac DLL1.java

C:\Test>java DLL1
Forward display :
14--->16--->13--->12--->11--->15--->
Backward display :
15--->11--->12--->13--->16--->14--->
C:\Test>
```

Thanks