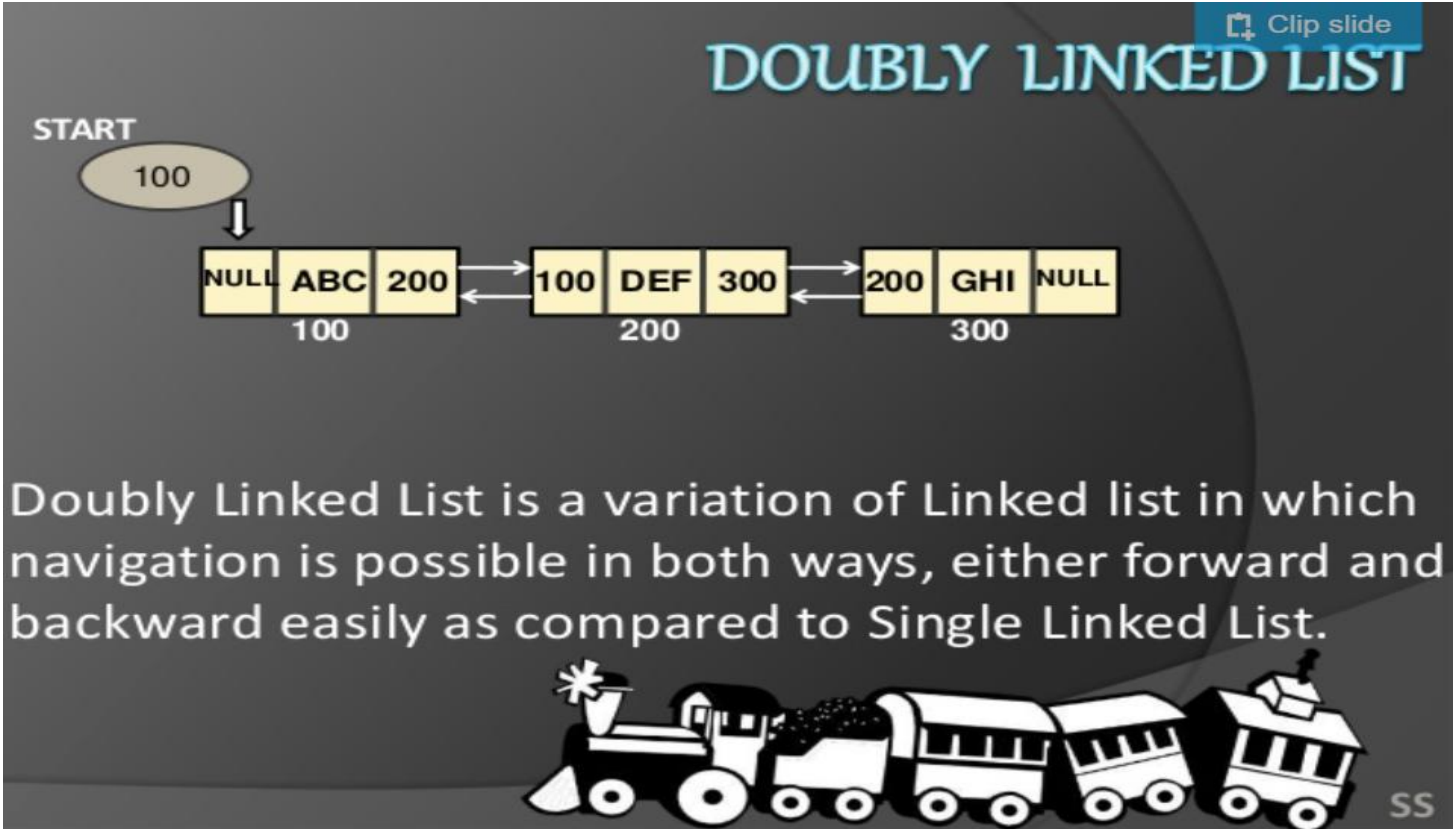
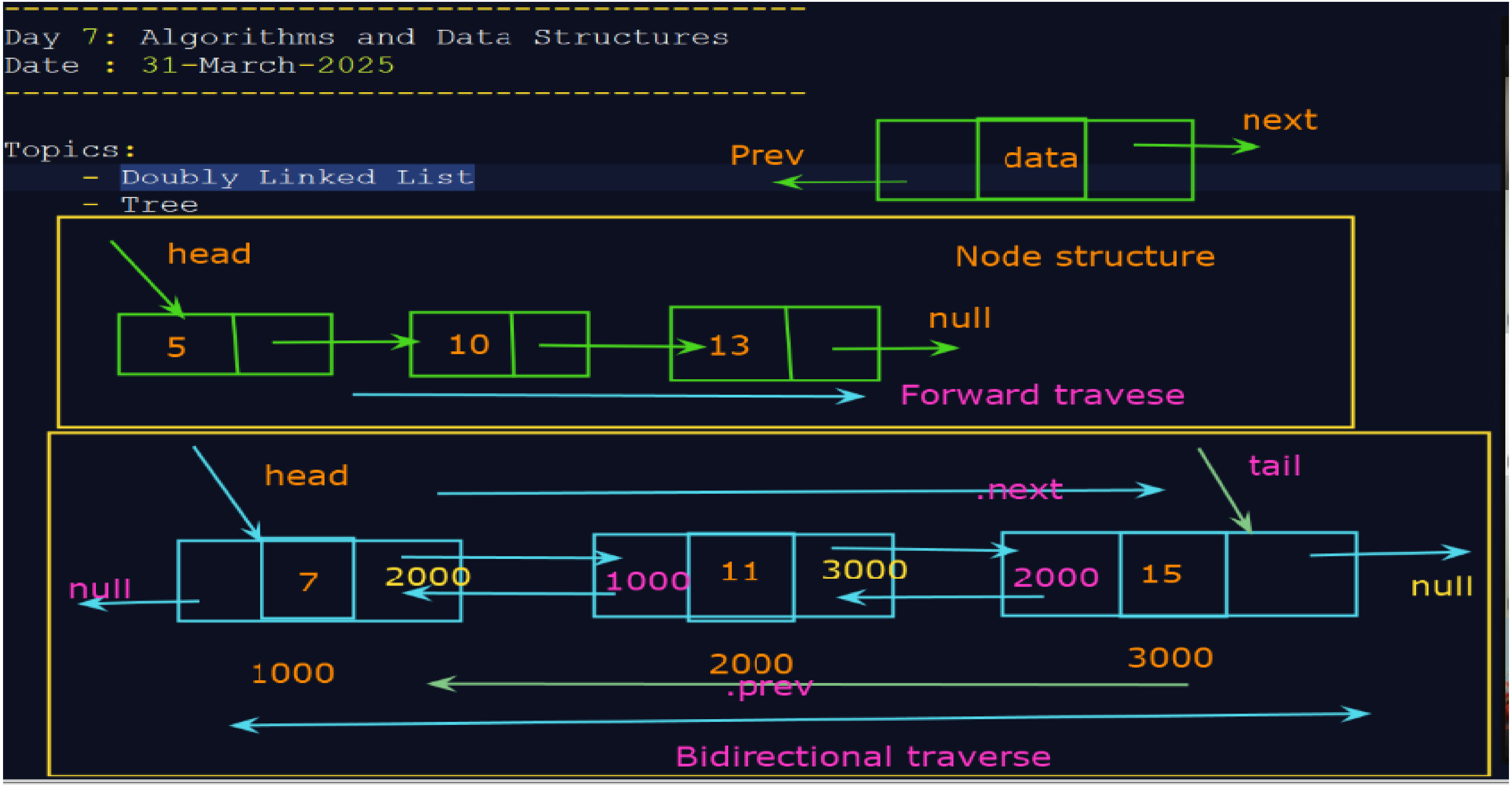
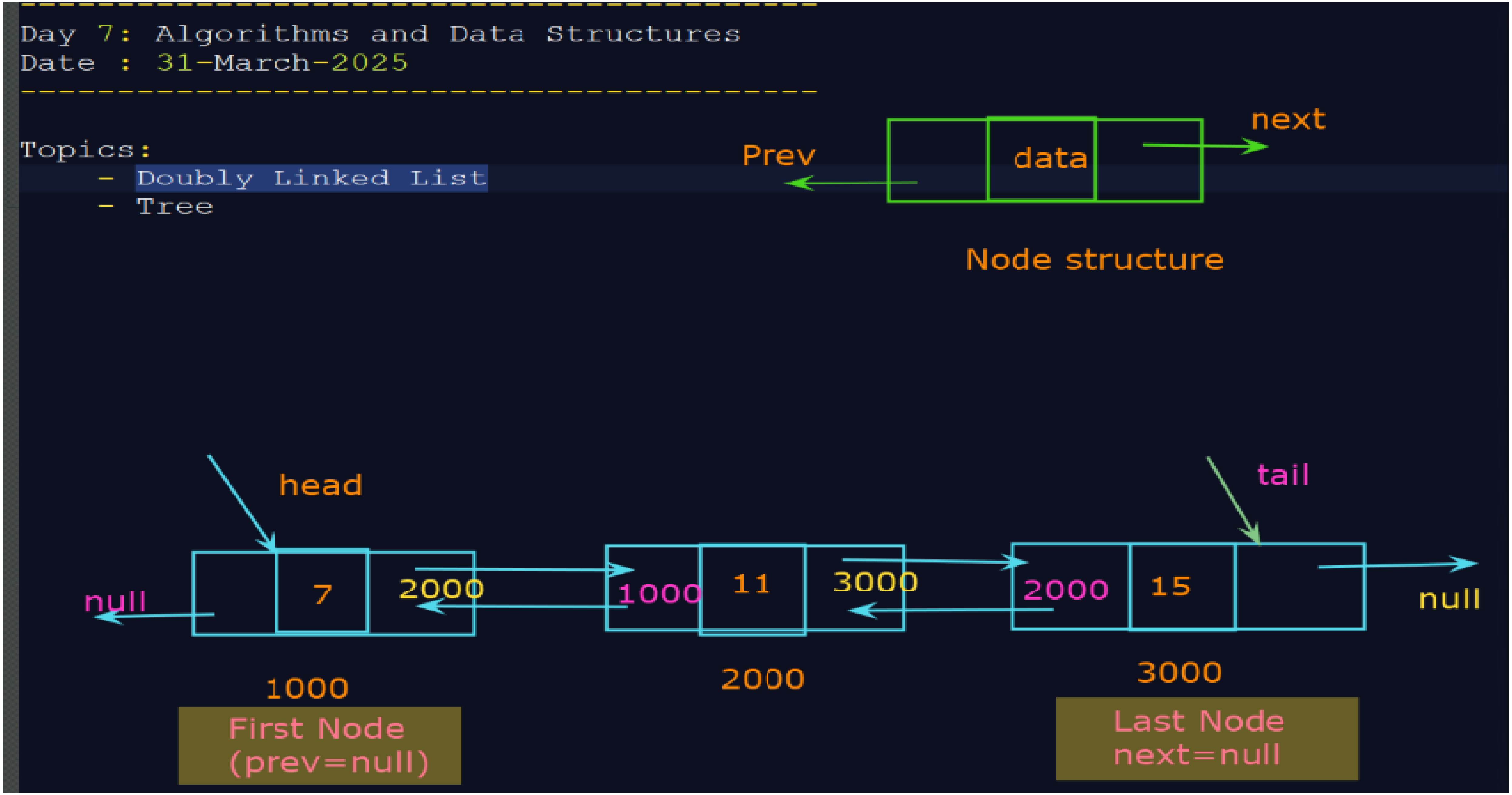


# DOUBLY LINKED LIST

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.



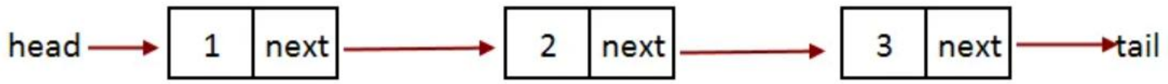
```
----------------------------------------------------
Day 7: Algorithms and Data Structures
Date : 31-March-2025
----------------------------------------------------

Topics:
     - Doubly Linked List
     - Tree
```



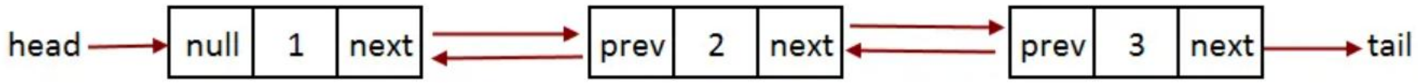
```
------------------------------------------
Day 7: Algorithms and Data Structures
Date : 31-March-2025
------------------------------------------

Topics:
    - Doubly Linked List
    - Tree
```

## 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



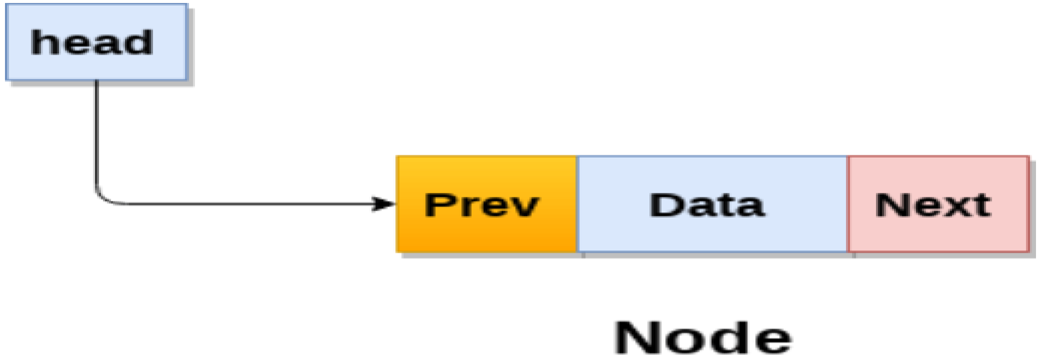
Doubly Linked List

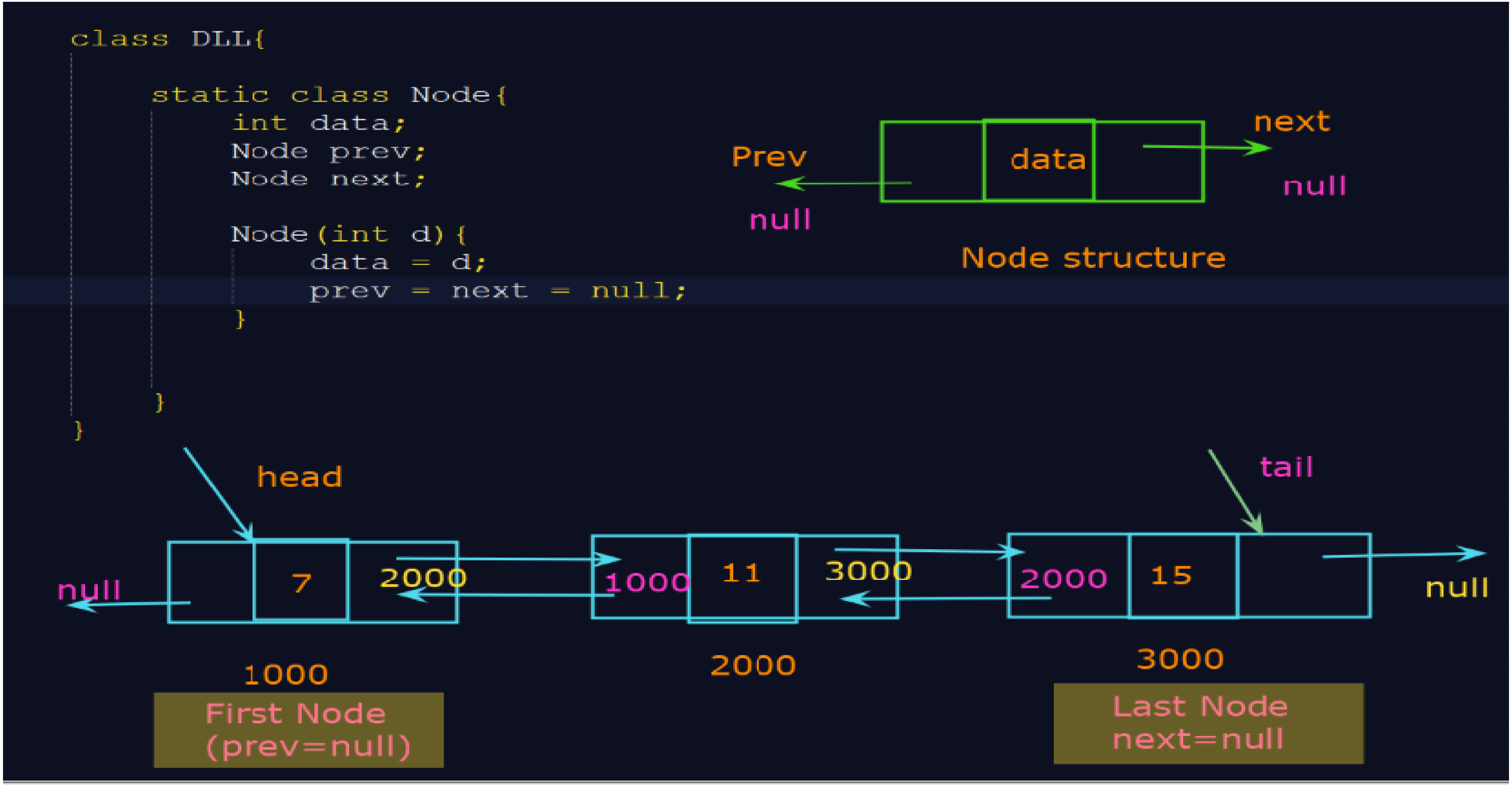
# Doubly linked list

- Doubly linked list is a complex type of linked list
  - in which a node contains a pointer to the previous as well as the next node in the sequence.

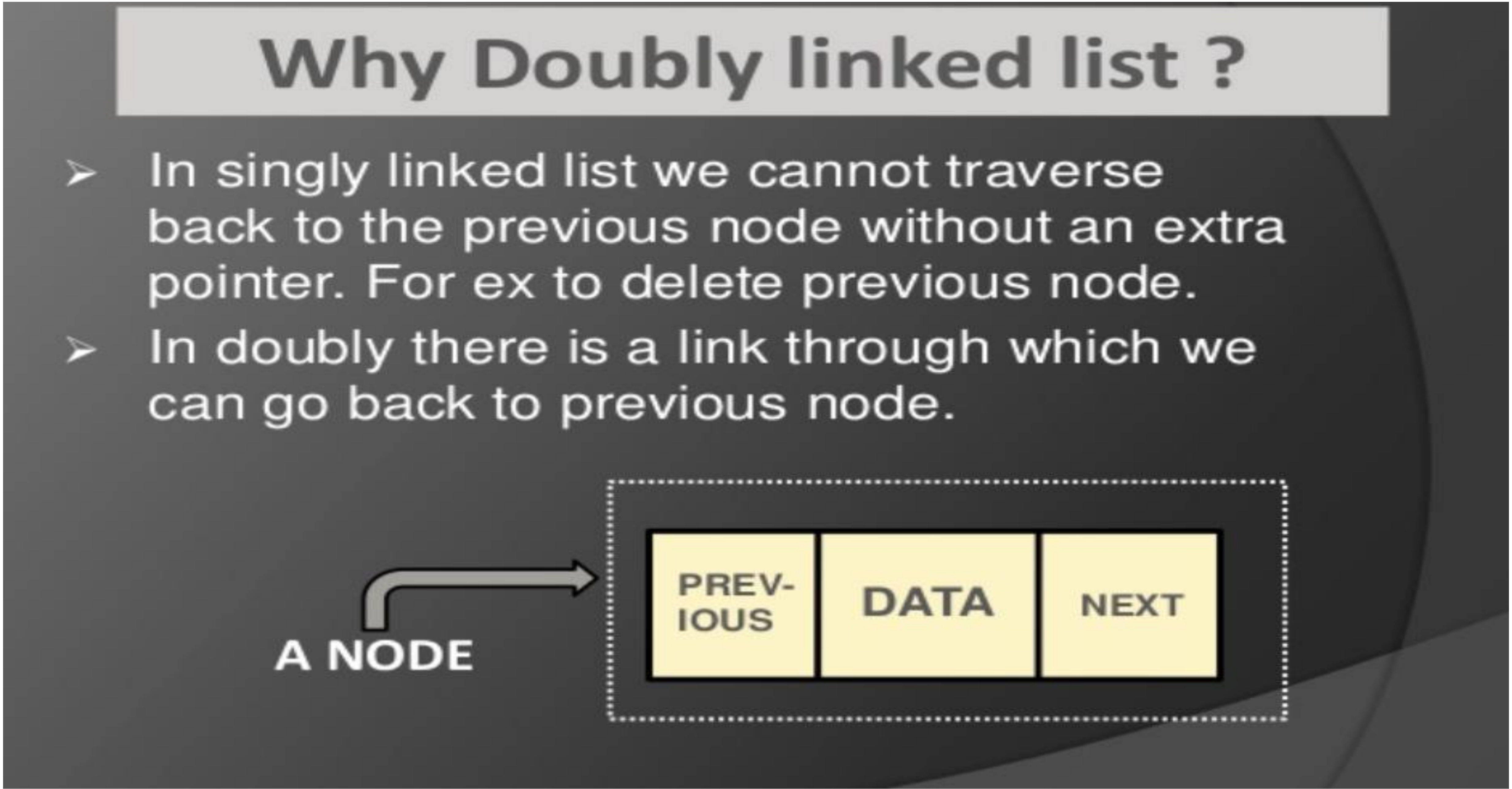
- In a doubly linked list, a node consists of three parts:

1. Data
2. Pointer to the previous node
3. pointer to the next node





```
class DLL{

    static class Node{
        int data;
        Node prev;
        Node next;

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

    }
}
```



# Why Doubly linked list ?

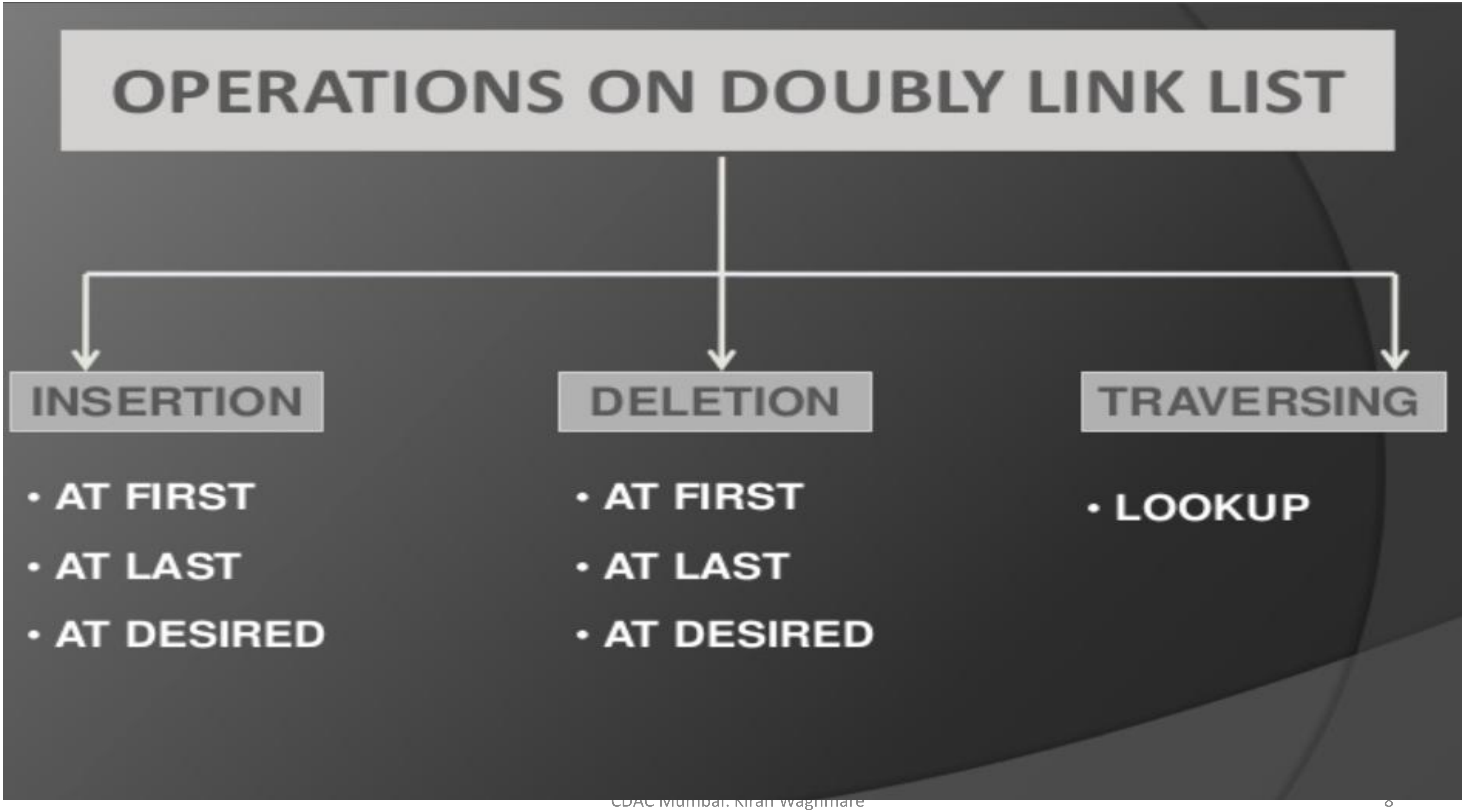
- In singly linked list we cannot traverse back to the previous node without an extra pointer. For ex to delete previous node.
- In doubly there is a link through which we can go back to previous node.

A NODE

PREVIOUS

DATA

NEXT



# OPERATIONS ON DOUBLY LINK LIST

INSERTION

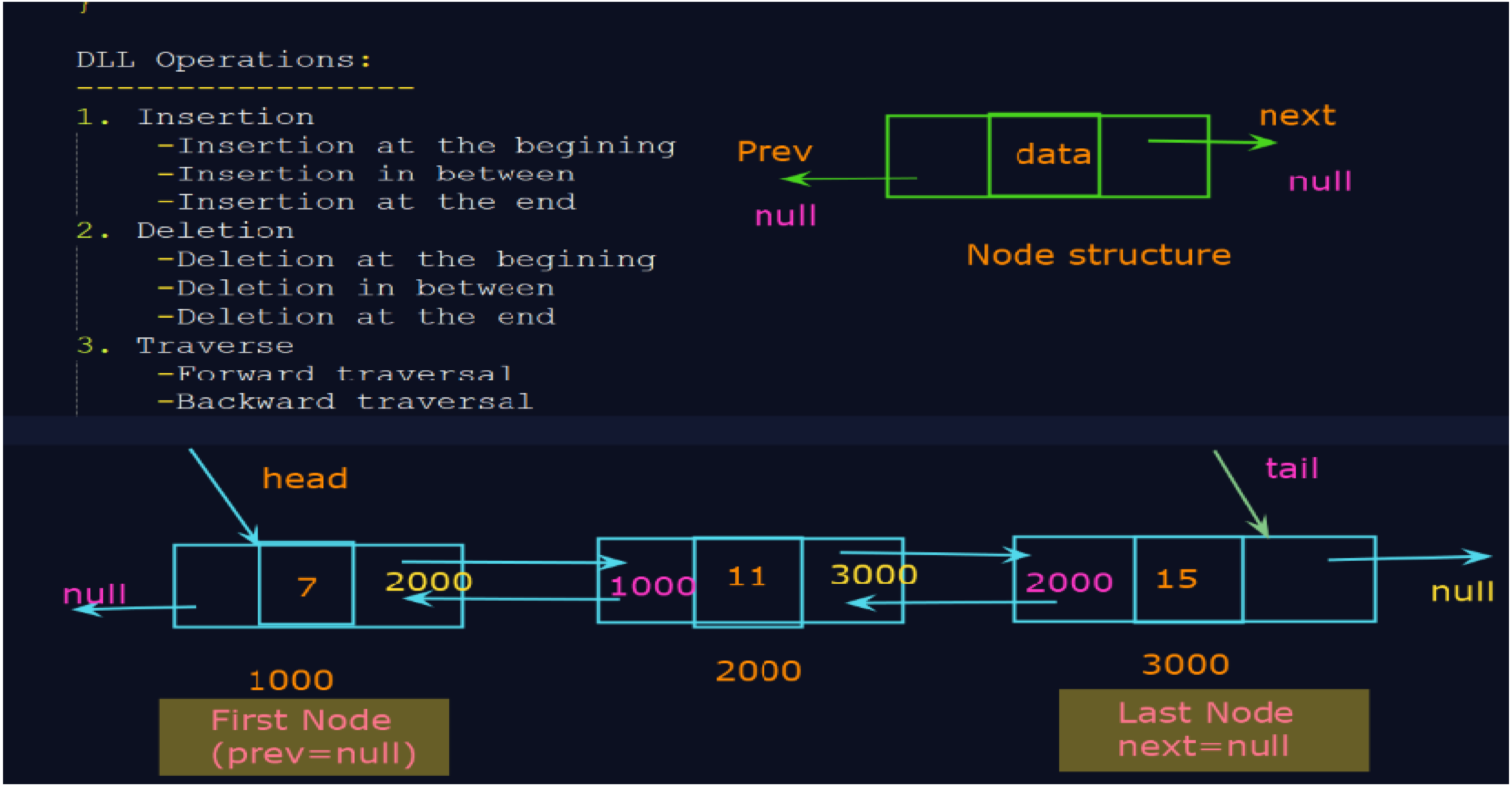
- AT FIRST
- AT LAST
- AT DESIRED

DELETION

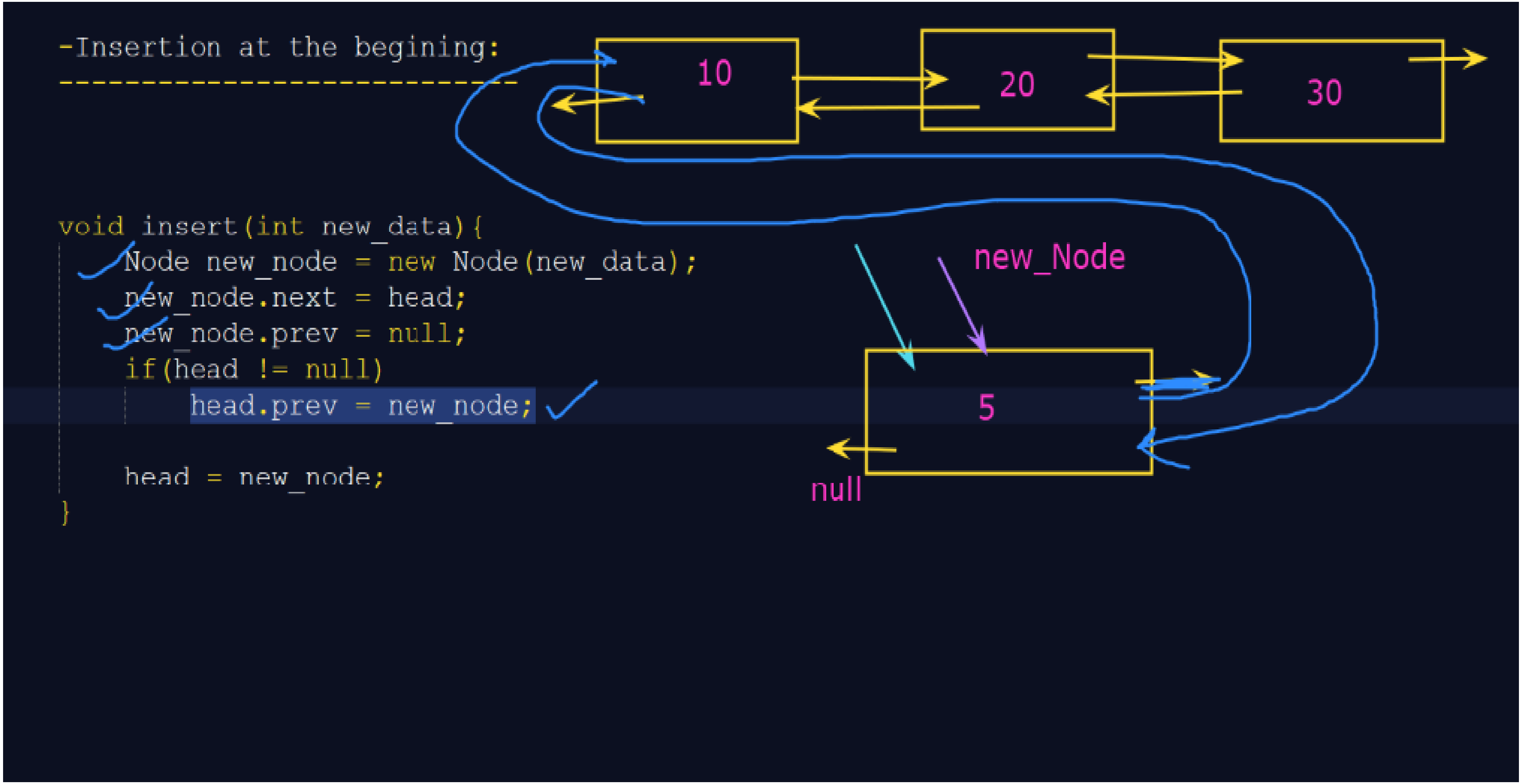
- AT FIRST
- AT LAST
- AT DESIRED

TRAVERSING

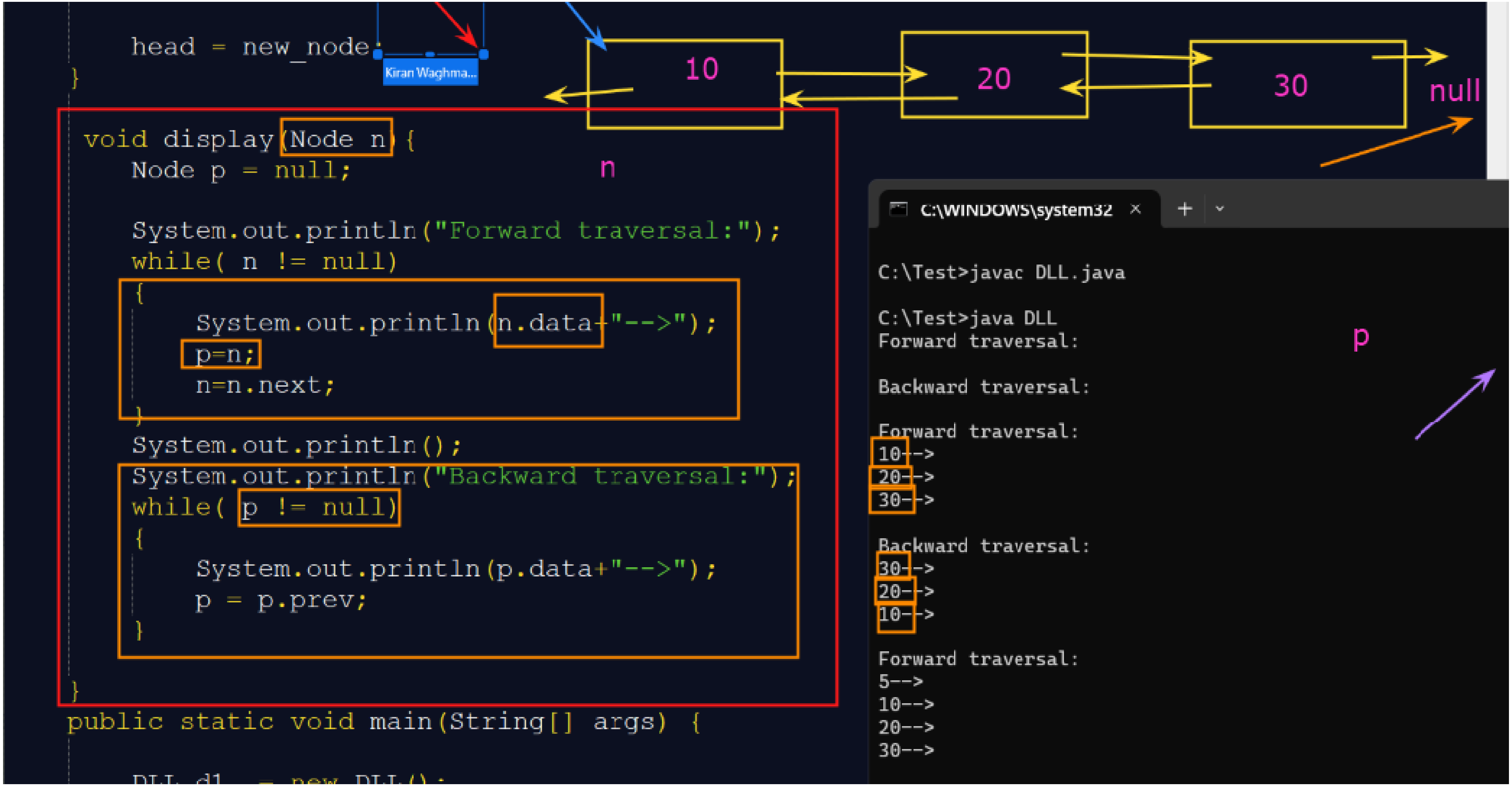
- LOOKUP



```
}

DLL Operations:
-----------------
1.  Insertion
      -Insertion at the begining
      -Insertion in between
      -Insertion at the end
2.  Deletion
      -Deletion at the begining
      -Deletion in between
      -Deletion at the end
3.  Traverse
      -Forward traversal
      -Backward traversal
```

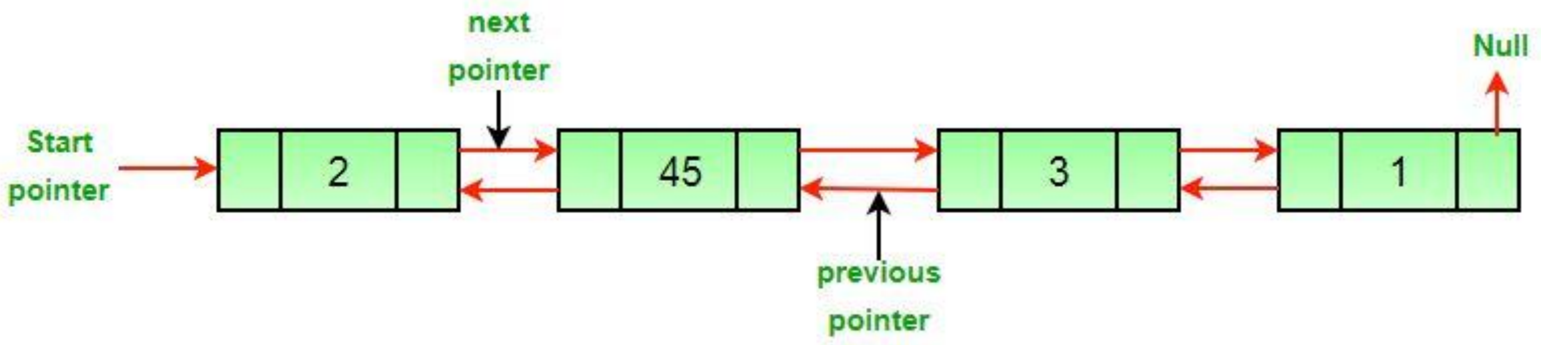


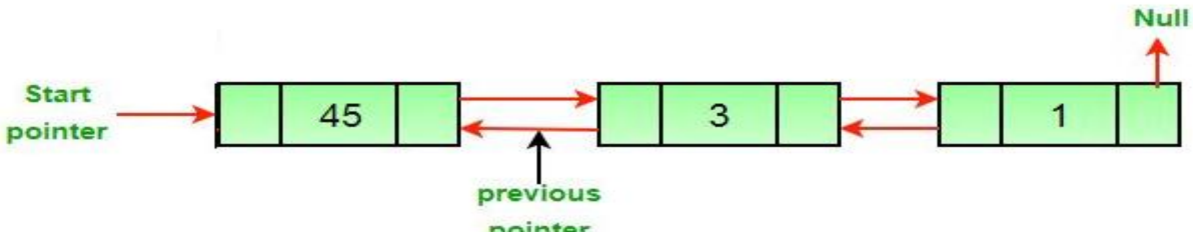
```
-Insertion at the begining:
-----------------------------

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;
}
```

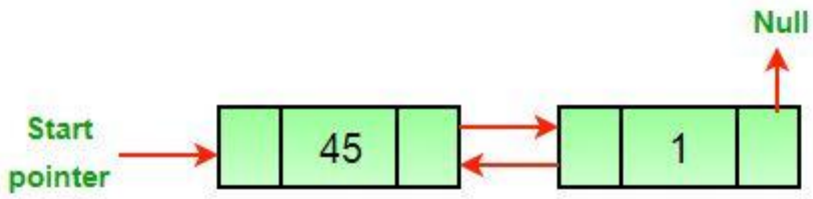


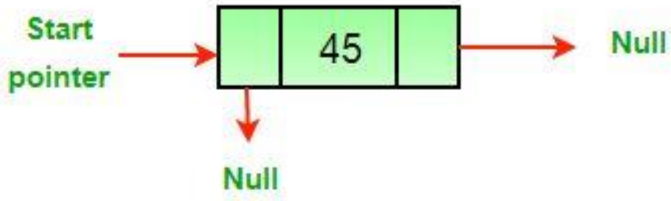
```
    head = new_node;
}

void display(Node n){
    Node p = null;

    System.out.println("Forward traversal:");
    while( n != null)
    {
        System.out.println(n.data+"-->");
        p=n;
        n=n.next;
    }
    System.out.println();
    System.out.println("Backward traversal:");
    while( p != null)
    {
        System.out.println(p.data+"-->");
        p = p.prev;
    }
}
public static void main(String[] args) {

    DLL d1  = new DLL();
```

```
C:\WINDOWS\system32

C:\Test>javac DLL.java

C:\Test>java DLL
Forward traversal:

Backward traversal:

Forward traversal:
10-->
20-->
30-->

Backward traversal:
30-->
20-->
10-->

Forward traversal:
5-->
10-->
20-->
30-->
```


next
pointer
Null
Start
pointer
2
45
3
1
previous
pointer


Null
Start
pointer
45
3
1
previous
pointer


Null
Start
pointer
45
1


Start
pointer
45
Null
Null

# Thanks