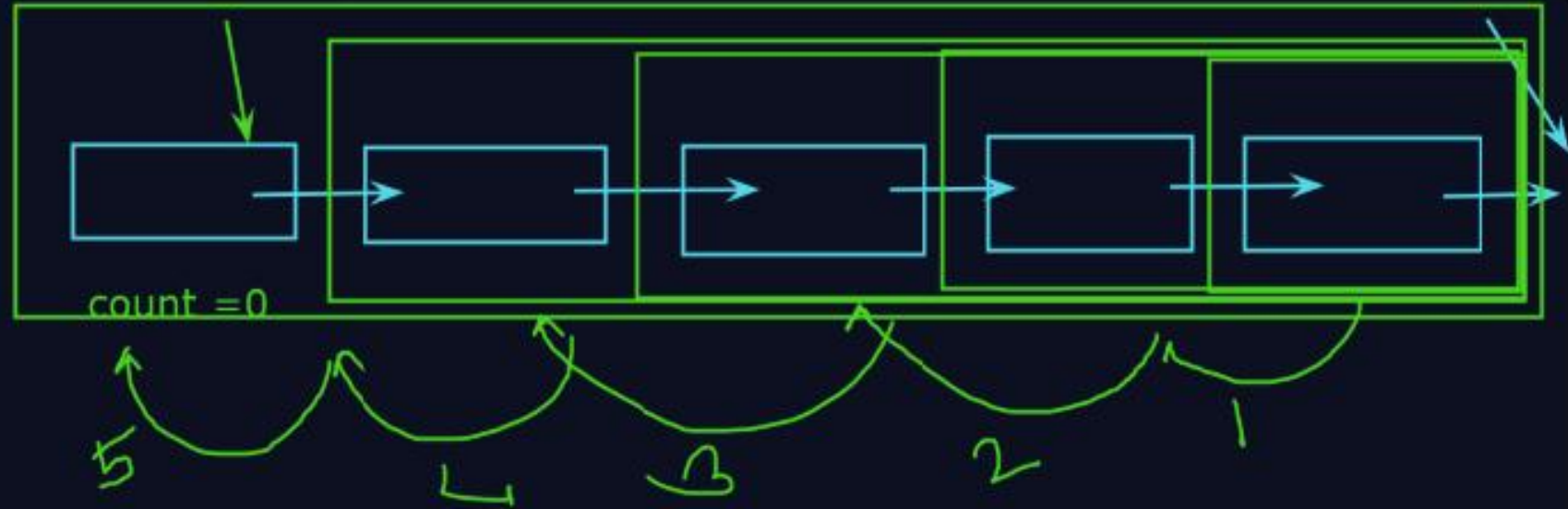


```

int counter(){
    Node temp = head;
    int count=0;
    while(temp != null){
        count++;
        temp = temp.next;
    }
    return count;
}

```



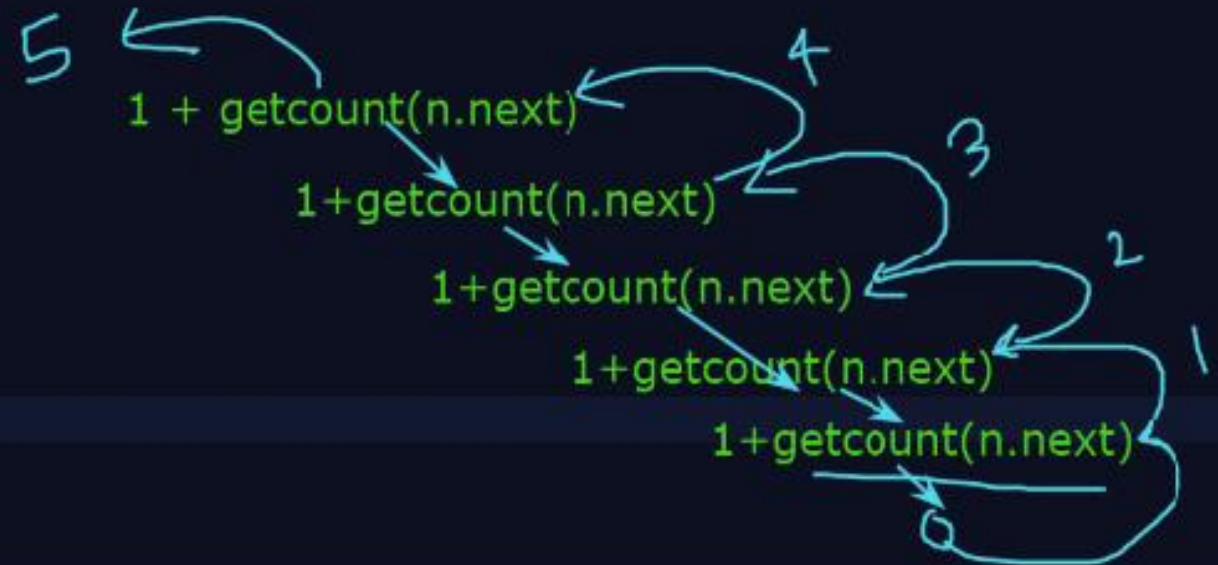
2. Write recursive function to get the vount of linkedlist.

```

int getcount(Node n){
    if(n == null)
        return 0;
    return 1 + getcount(n.next);
}

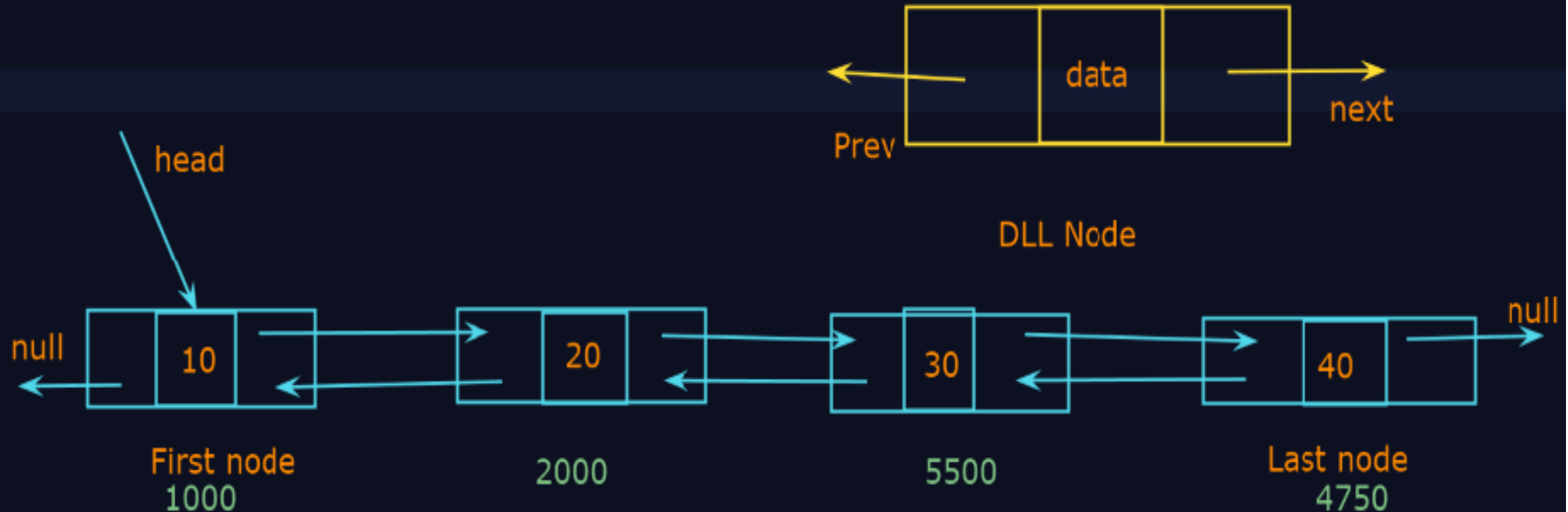
```

Diagram illustrating the recursive function call stack for the linked list. The function is called with the head pointer. The stack shows the sequence of calls: `getcount(head)`, `getcount(head->next)`, `getcount(head->next->next)`, `getcount(head->next->next->next)`, and `getcount(head->next->next->next->next)`. The final return value is 5.



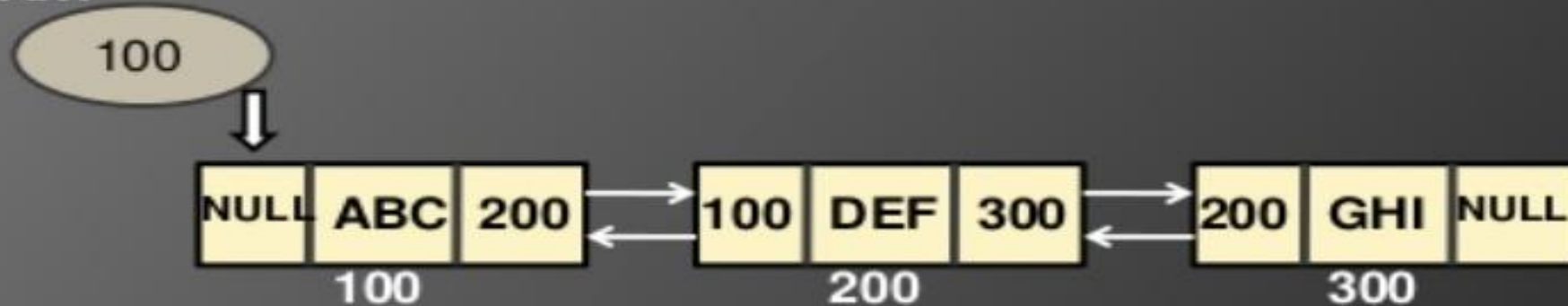
DLL:

- Not that much easy
- More memory
- Traverse in both direction (forward and backward directions)



DOUBLY LINKED LIST

START



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.



Topics:

- Doubly Linked List
- Tree

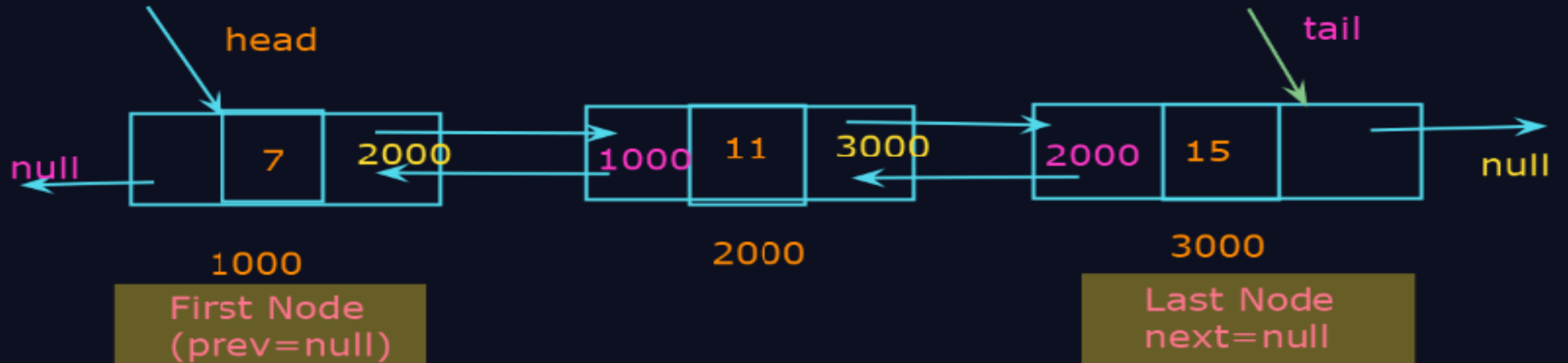


Topics:

- Doubly Linked List
- Tree

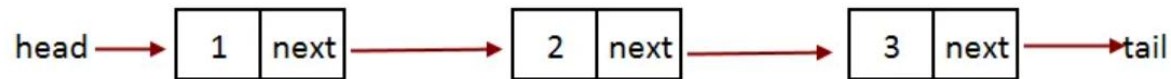


Node structure

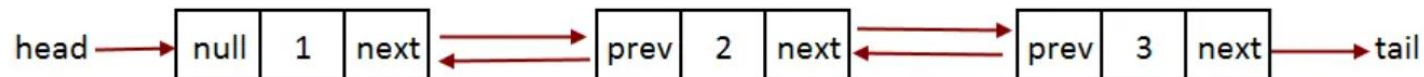


Singly Linked List vs Doubly Linked List

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



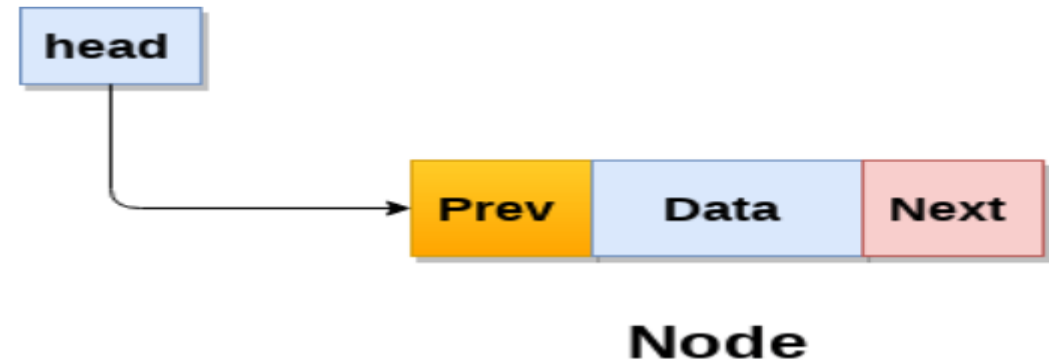
Singly Linked List



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

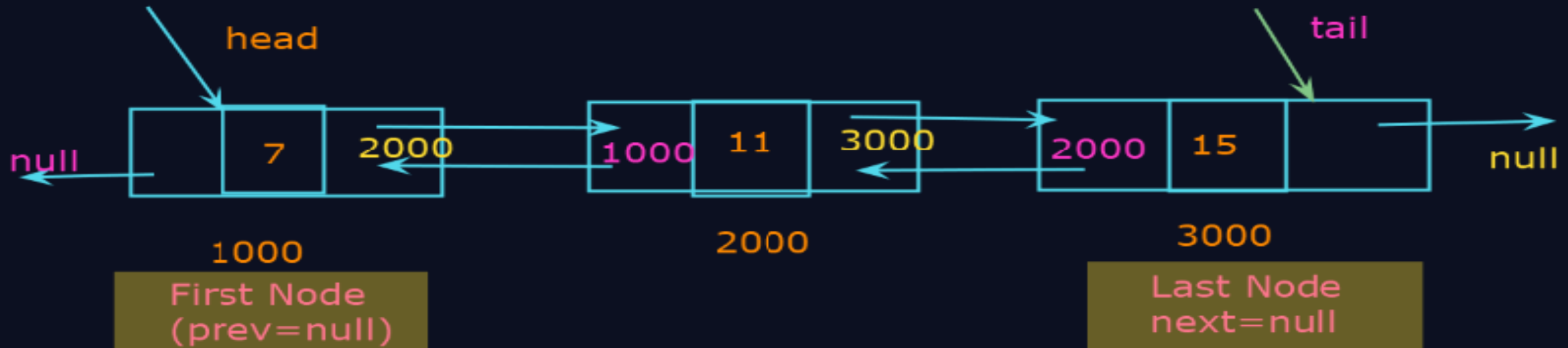


```

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.



OPERATIONS ON DOUBLY LINK LIST

```
graph TD; A[OPERATIONS ON DOUBLY LINK LIST] --> B[INSERTION]; A --> C[DELETION]; A --> D[TRAVERSING]; B --> B1[• AT FIRST]; B --> B2[• AT LAST]; B --> B3[• AT DESIRED]; C --> C1[• AT FIRST]; C --> C2[• AT LAST]; C --> C3[• AT DESIRED]; D --> D1[• LOOKUP];
```

INSERTION

- AT FIRST
- AT LAST
- AT DESIRED

DELETION

- AT FIRST
- AT LAST
- AT DESIRED

TRAVERSING

- LOOKUP

DLL Operations:

1. Insertion

- Insertion at the beginning
- Insertion in between
- Insertion at the end

2. Deletion

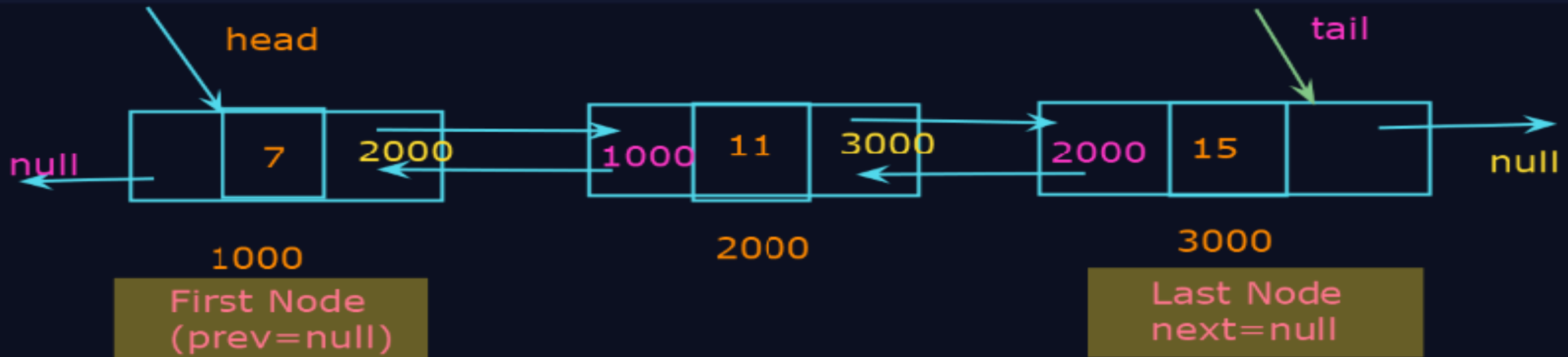
- Deletion at the beginning
- Deletion in between
- Deletion at the end

3. Traverse

- Forward traversal
- Backward traversal



Node structure



-Insertion at the beginning:

```
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;  
}
```

