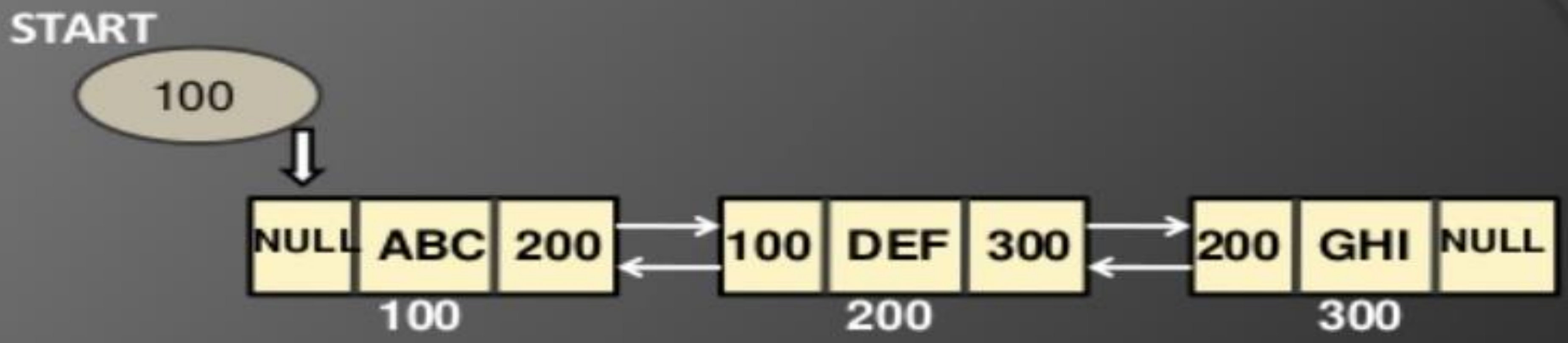
Mar24 : Day 5

Kiran Waghmare

CDAC Mumbai

Clip slide

DOUBLY LINKED LIST

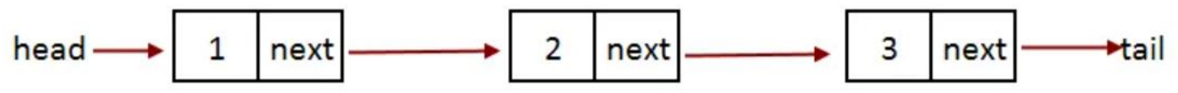


Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List.

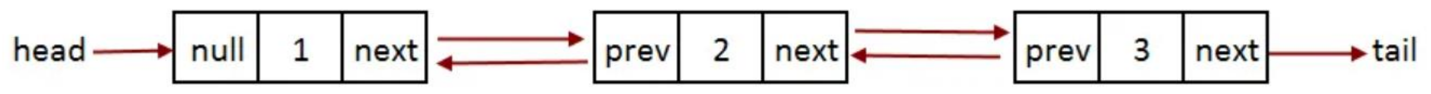
SS

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



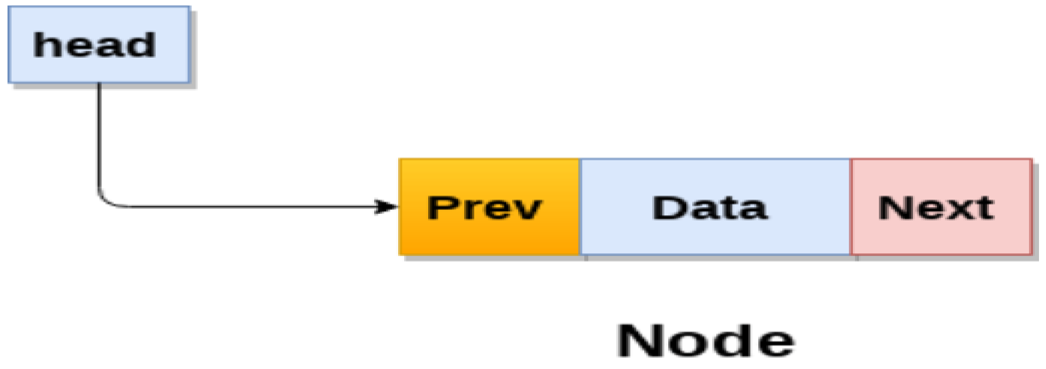
Doubly Linked List

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.

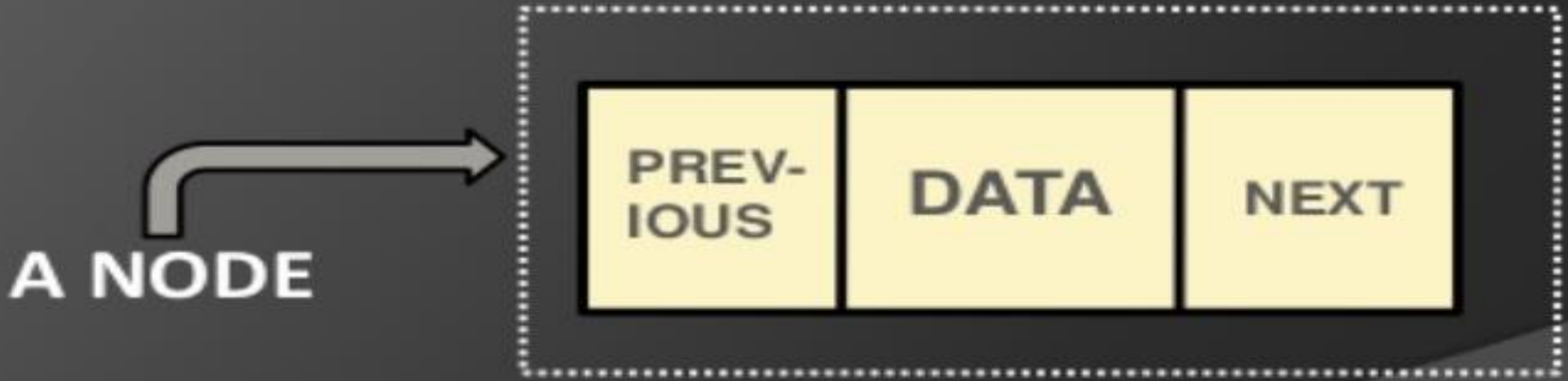
- In a doubly linked list, a node consists of three parts:

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

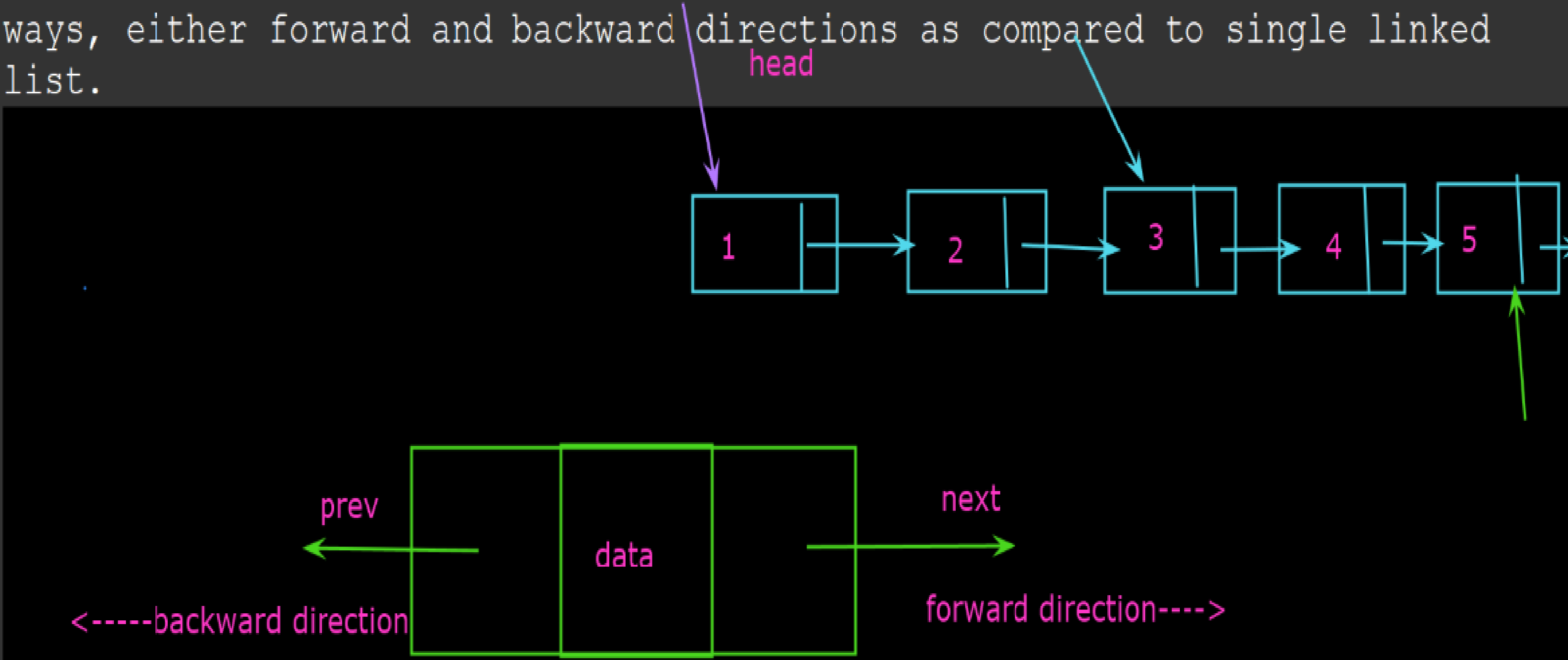


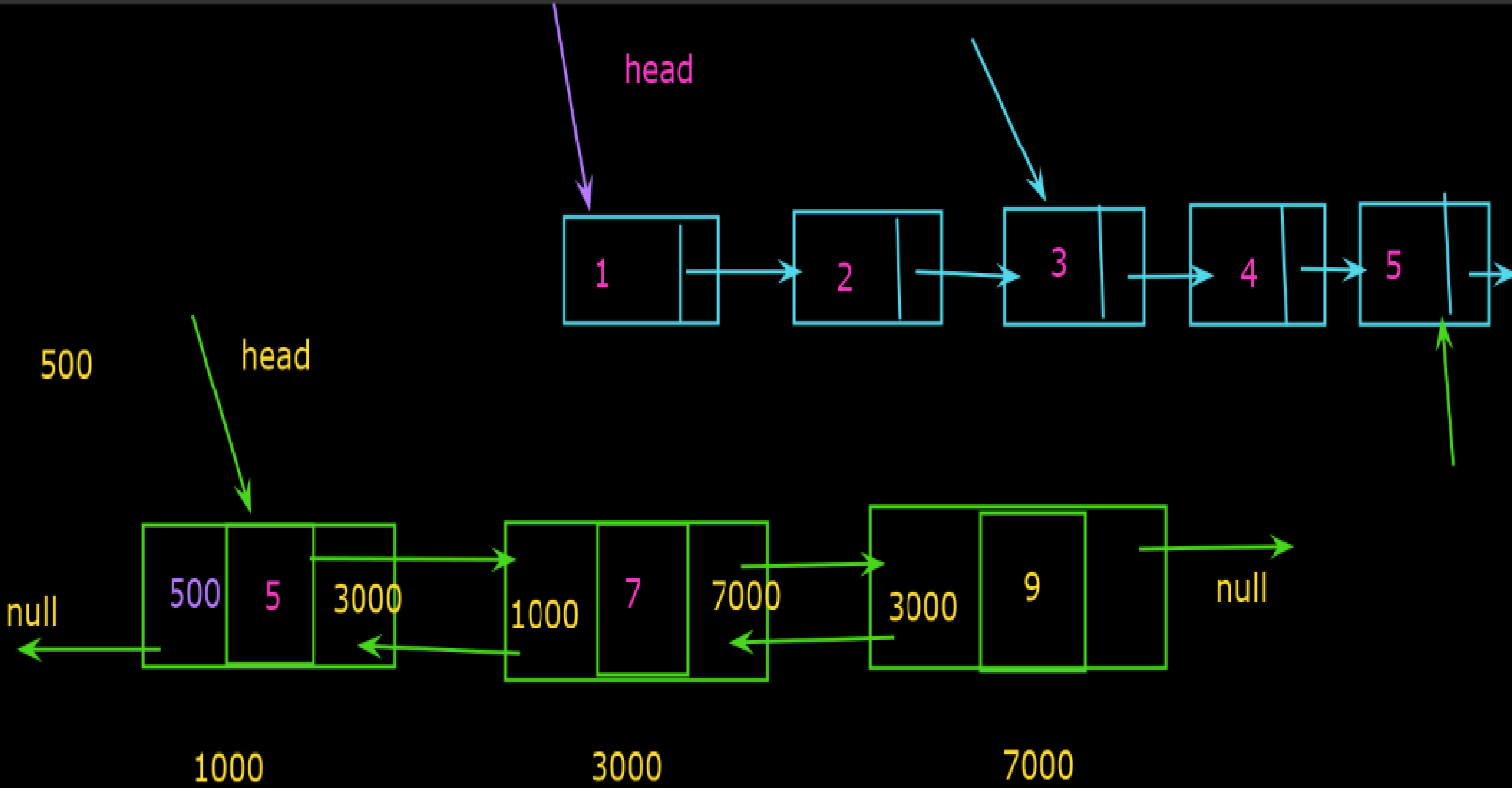
Why Doubly linked list ?

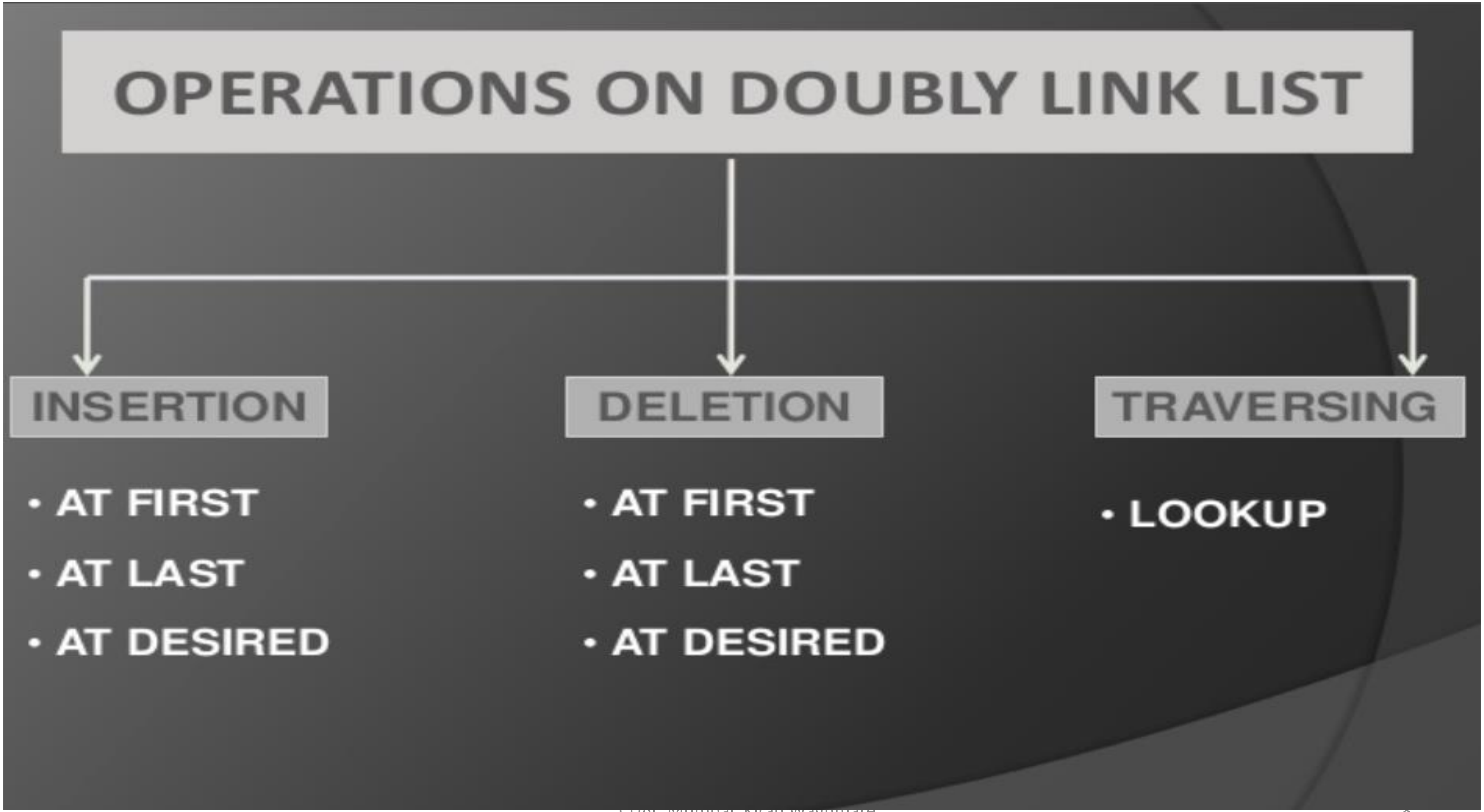
- In singly linked list we cannot traverse back to the previous node without an extra pointer. For ex to delete previous node.
- In doubly there is a link through which we can go back to previous node.



DLL is a variation of Linked list in which navigation is possible in both ways, either forward and backward directions as compared to single linked list.




head
1
2
3
4
5
500
head
null
500
5
3000
1000
7
7000
3000
9
null
1000
3000
7000



OPERATIONS ON DOUBLY LINK LIST

INSERTION

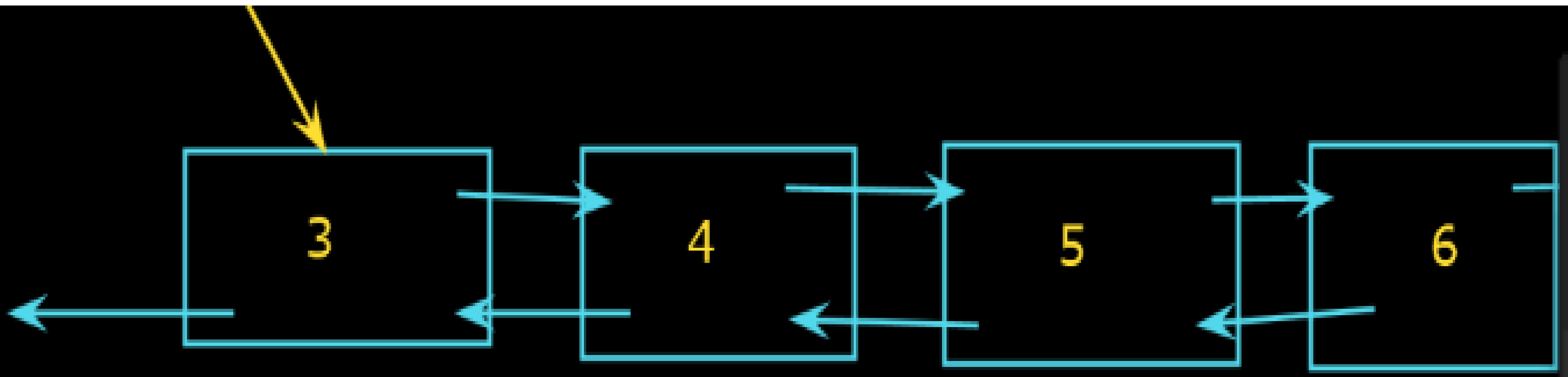
- AT FIRST
- AT LAST
- AT DESIRED

DELETION

- AT FIRST
- AT LAST
- AT DESIRED

TRAVERSING

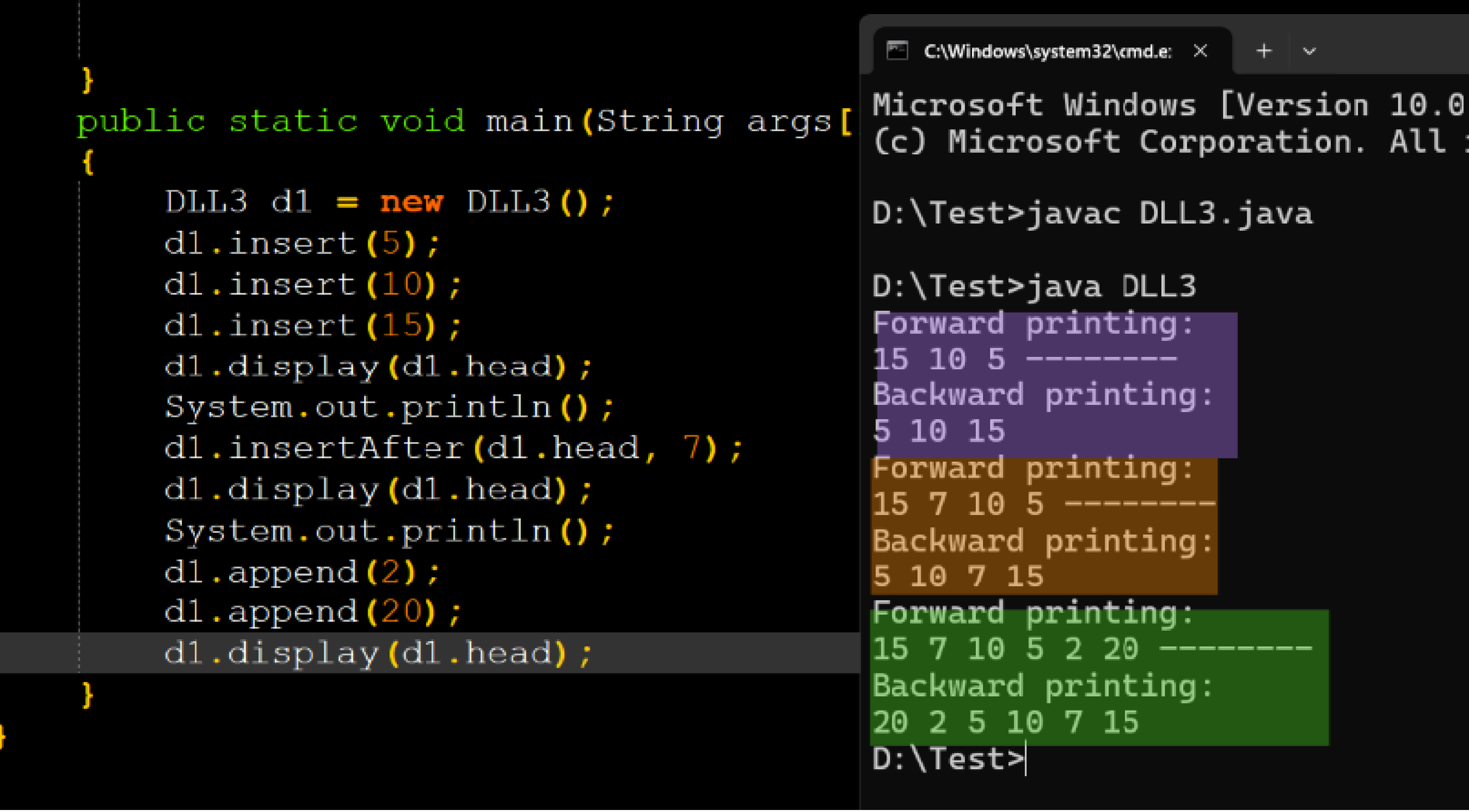
- LOOKUP



```
DLL Node structure
------------------

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

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



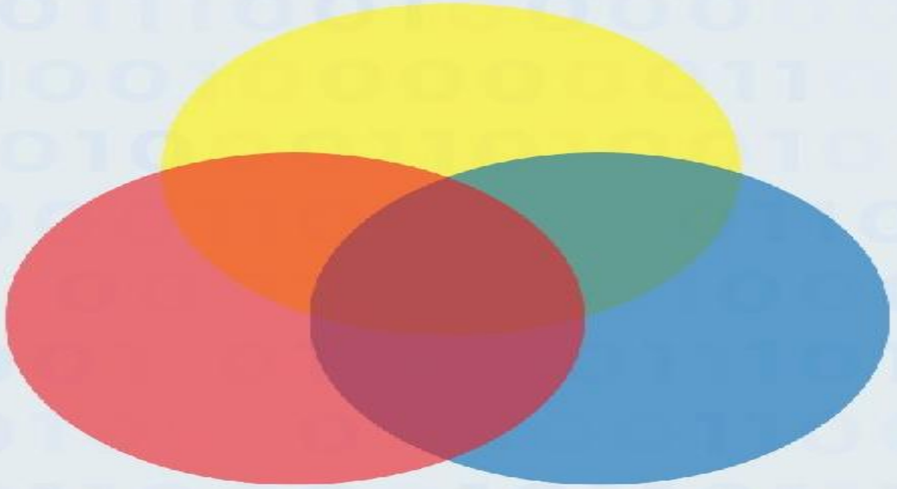
```
    }
    public static void main(String args[
    {
        DLL3 d1 = new DLL3();
        d1.insert(5);
        d1.insert(10);
        d1.insert(15);
        d1.display(d1.head);
        System.out.println();
        d1.insertAfter(d1.head, 7);
        d1.display(d1.head);
        System.out.println();
        d1.append(2);
        d1.append(20);
        d1.display(d1.head);
    }
}
```

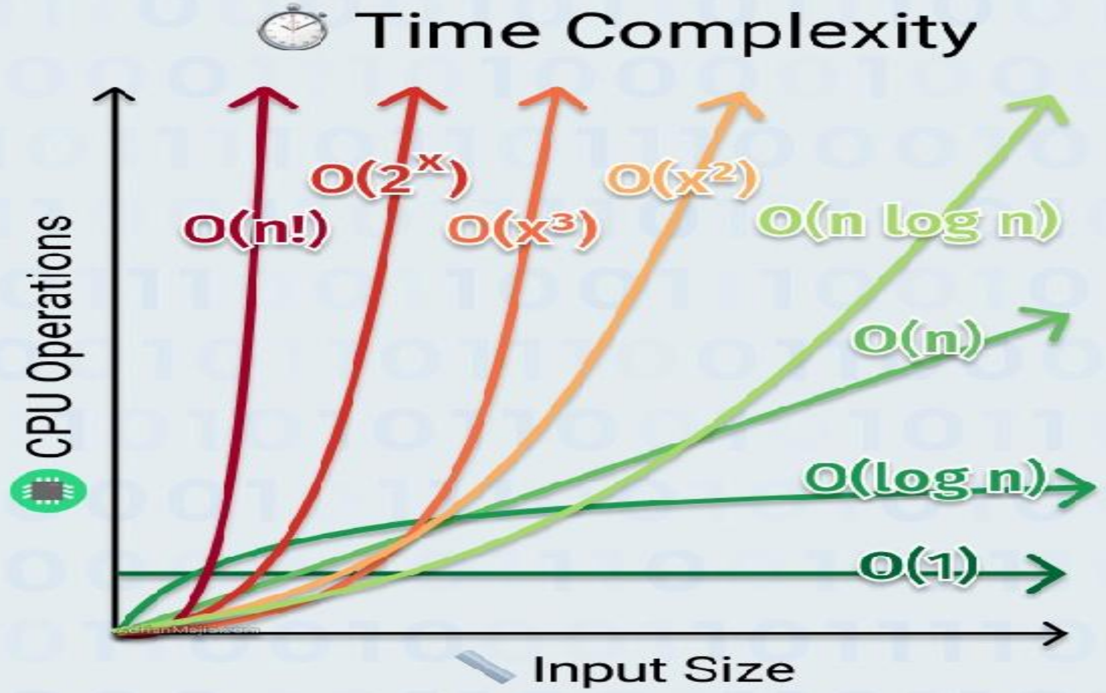
```
Microsoft Windows [Version 10.0
(c) Microsoft Corporation. All

D:\Test>javac DLL3.java

D:\Test>java DLL3
Forward printing:
15 10 5 --------
Backward printing:
5 10 15
Forward printing:
15 7 10 5 --------
Backward printing:
5 10 7 15
Forward printing:
15 7 10 5 2 20 --------
Backward printing:
20 2 5 10 7 15
D:\Test>
```

Time and Space Complexity in Data Structure





```
Analysis of Algorithms:
----------------------------
Algorithm:
------------
    -Design
    -Domain Knowledge
    -Language
    -Hardware, Opeating System
    -Analysis

Programs:
------------
    -Implementation
    -Programmers
    -Programming Language
    -Hardware, Opeating System
    -Testing
```

Priori Analysis

-Algorithms
-Independent of PL
-Independent of Hardware
-Time & Space

Posterior Analysis

-Programs
-Dependent of PL
-Dependent on H/W
-Time

Asymptotic Notations:

-Asymptotic analysis of an algorithm refers to defining the mathematical boundings of ite runtime performance.

-It cn defined in following terms:

- -Best case :
  - -Minimum time required for program execution. Ω
- -Average case :
  - -Average time required for program execution. Θ
- -Worst case :
  - -Maximum time required for program execution. O ✓

Ex:

Algorithm for swapping of 2 numbers:

```
swap(a,b)
{
    temp = a;
    a = b;
    b = temp;
}
```

	Time
temp = a;	1
a = b;	1
b = temp;	1
	$f(n) = \boxed{3}$
	O(1)

Space

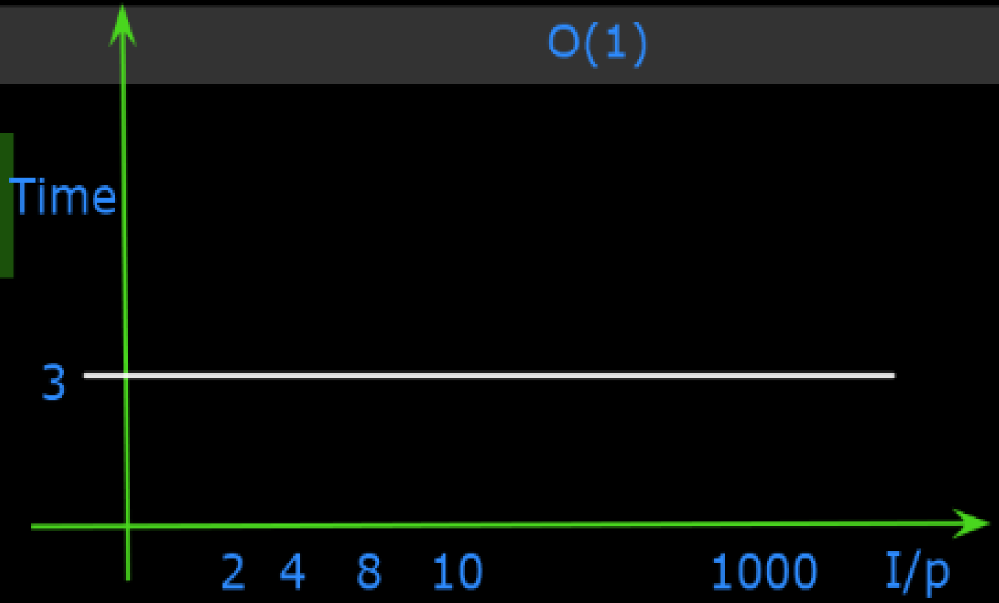
```
a ------------> 1
b ------------> 1
temp --------> 1
```

$S(n) = \boxed{3}$ words

O(1)

```
x = 5*a + 6*b --------> 1s
x = 5*a + 6*b
x = 5*a + 6*b
x = 5*a + 6*b
x = 5*a + 6*b
-----------------------------------
f(n) = 5 -------> O(1)
```

Constant Complexity



Ex: Algorithm : sum of array elements

n=5 i =0,1,2,3,4

	0	1	2	3	4
A	8	3	9	7	2

```
sum(A, n)
{
    s=0;                    --->  1
    for(i=0;i<n;i++)        --->  n+1
    {
        s=s+A[i];           --->  n
    }
    return s;               --->  1
}
```

Time

1
n+1
n
1

$\boxed{f(n) = 2n+3}$

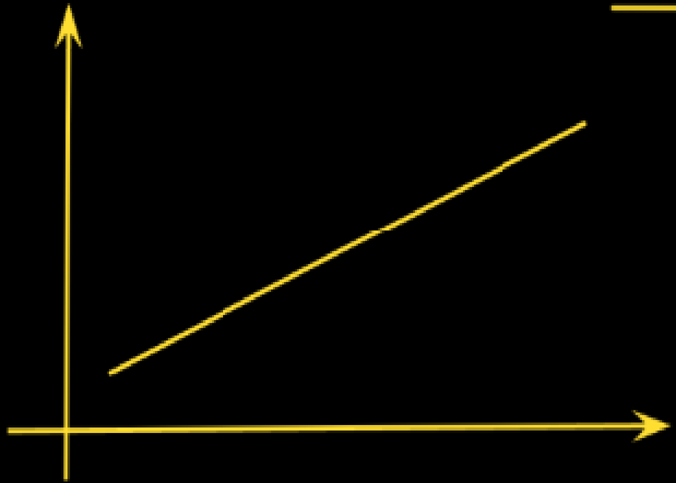
$\boxed{O(n)}$

Space

A----->n
n----->1
s----->1
i------>1

$\boxed{s(n) = n+3}$

$\boxed{O(n)}$



Linear Complexity