Stack and Queue Time Complexity Analysis (Array Implementation)

	Stack	Linear Queue	Circular Queue
Push	O(1)	O(1)	O(1)
Pop	O(1)	O(1)	O(1)
Peek	O(1)	O(1)	O(1)

Stack Application

Expression Evaluation and Conversion

1. Postfix Evaluation
2. Prefix Evaluation
3. Infix to Postfix Conversion
4. Infix to Prefix Conversion

Expression:

- Combination of operators and operands
  - Operators - mathematical symbols (+, -, *, /)
  - Operands - Values/Numbers/Variables
- eg. a+b-c, a * b + c

Types:

1. Infix	:	a + b	:	human
2. Prefix	:	+ a b	:	computer
3. Postfix	:	a b +	:	computer

Priority:

()
$
* / %
+ -

Postfix Evaluation

Postfix : 4 5 6 * 3 / + 9 + 7 -

left ⟶ **right**

Result = 16

23 - 7
= 16

14 + 9
= 23

4 + 10
= 14

30 / 3
= 10

5 * 6
= 30

Prefix Evaluation

Prefix : - + + 4 / * 5 6 3 9 7

left ← **right**

Result = 16

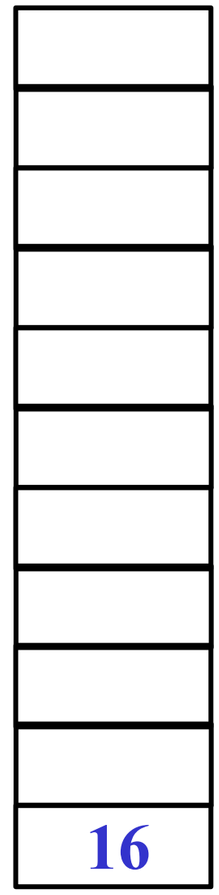
23 - 7
=16

14 + 9
= 23

4 + 10
= 14

30 / 3
= 10

5 * 6
= 30

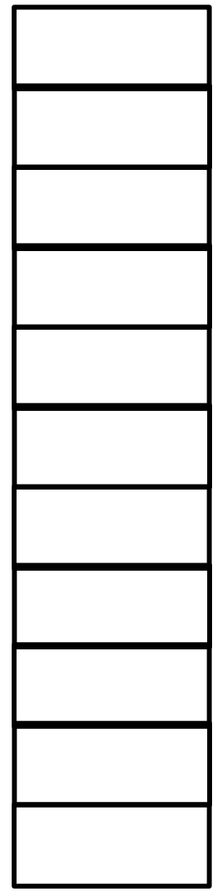


Infix to Postfix conversion

Infix : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7

left → **right**

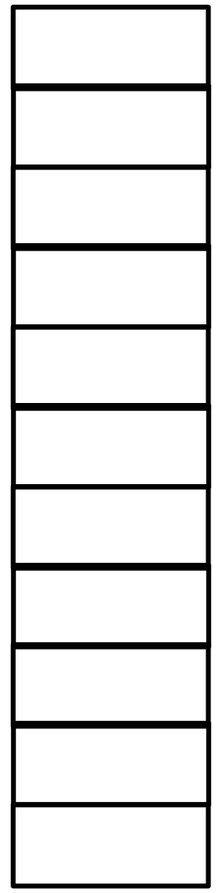
Postfix : 1 9 $ 3 4 * + 6 8 2 / + - 7 +



Infix to Prefix conversion

Infix : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7

left ← **right**

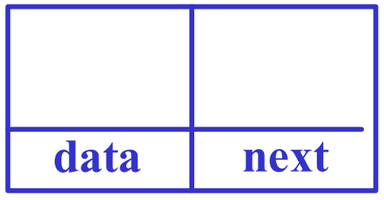


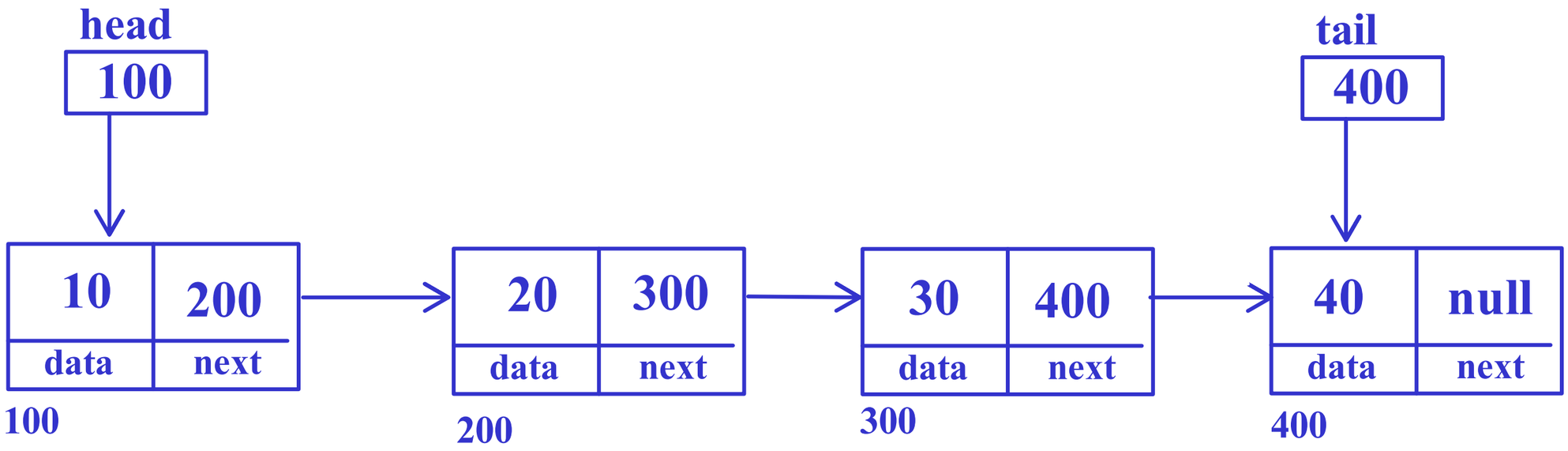
Expression: 7 2 8 / 6 + 4 3 * 9 1 $ + - +

Prefix : + - + $ 1 9 * 3 4 + 6 / 8 2 7

Linked List

- linear data structure in which data is stored sequentially
- address of next data is kept with current data
- Every element of linked list is called as node
- Node consist of two parts
    - data - actual data
    - link/next - address (referance) of next node
- Address of first node is kept into one of the pointer (head)
- Address of last node is kept into one of the pointer (tail) - optional





Linked List

Types:

1. Singly linear linked list
2. Singly circular linled list
3. Doubly linear linked list
4. Doubly circular linked list

Operations:

1. Add first
2. Add last
3. Add at position

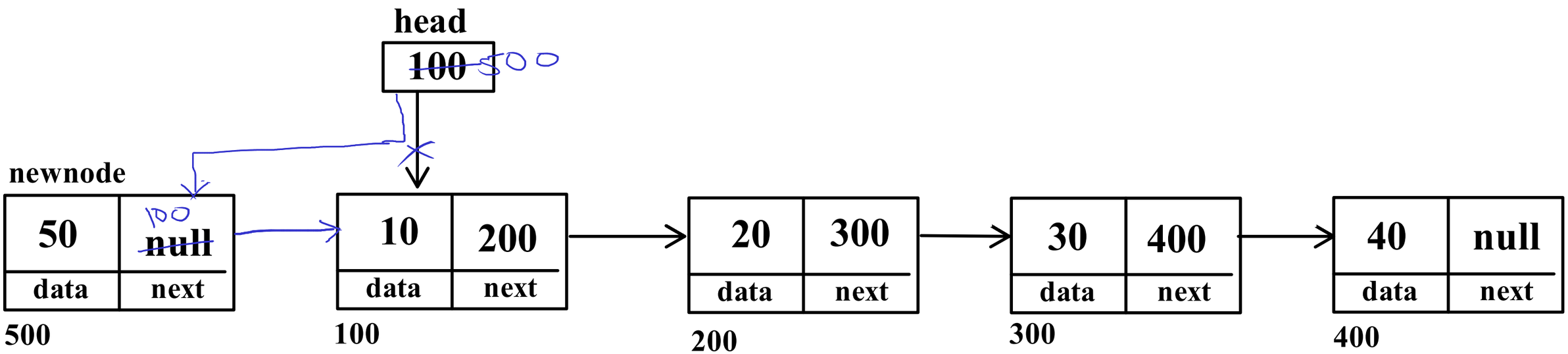
4. Delete first
5. Delete last
6. Delete at position

7. Display (Traverse)
8. Free list
9. Search
10. Sort
11. Reverse list
12. Find Mid

Node consist of two part:
1. data - char/int/double/class
2. next - referance

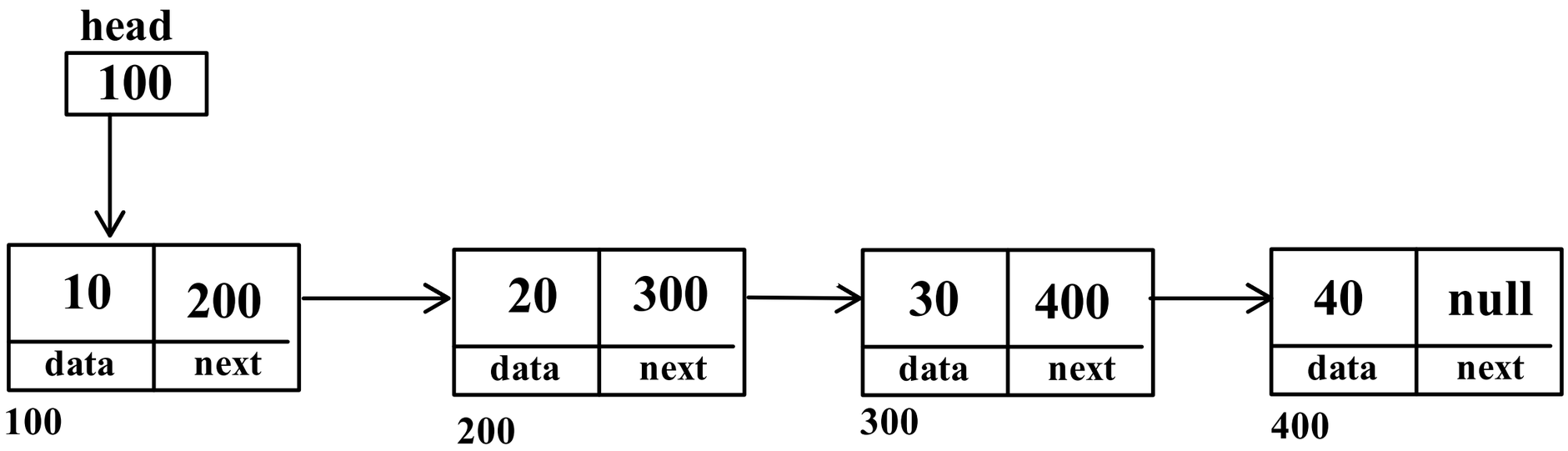
```
class List{
        static class Node{
            int data;
            Node next;
        }

        Node head;
        List()
        isEmpty()
        Add()
        Delete()
        Display()
}
```

Singly Linear Linked List - Add First



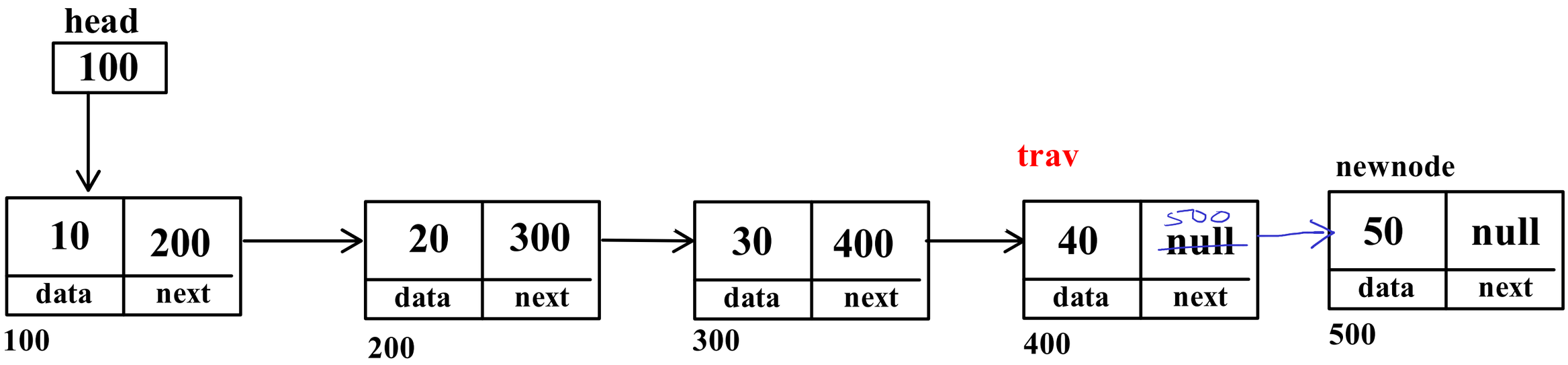

```
//0. create node with given data
//1. if list is empty
        //a. add newnode into head itself
//2. if list is not empty
        //a. add first node into next of newnode
        //b. add newnode into head
```

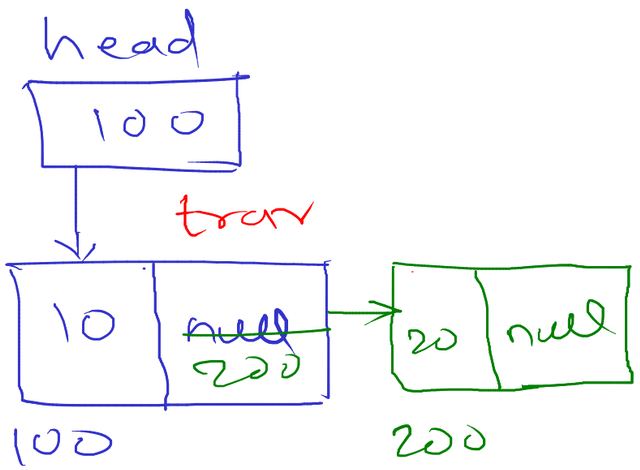
Singly Linear Linked List - Display (Traverse)




```
//1. create one referance and start from first node
//2. print/visit current node
//3. go on next node
//4. repeat step 2 and 3 untill trav != null
```

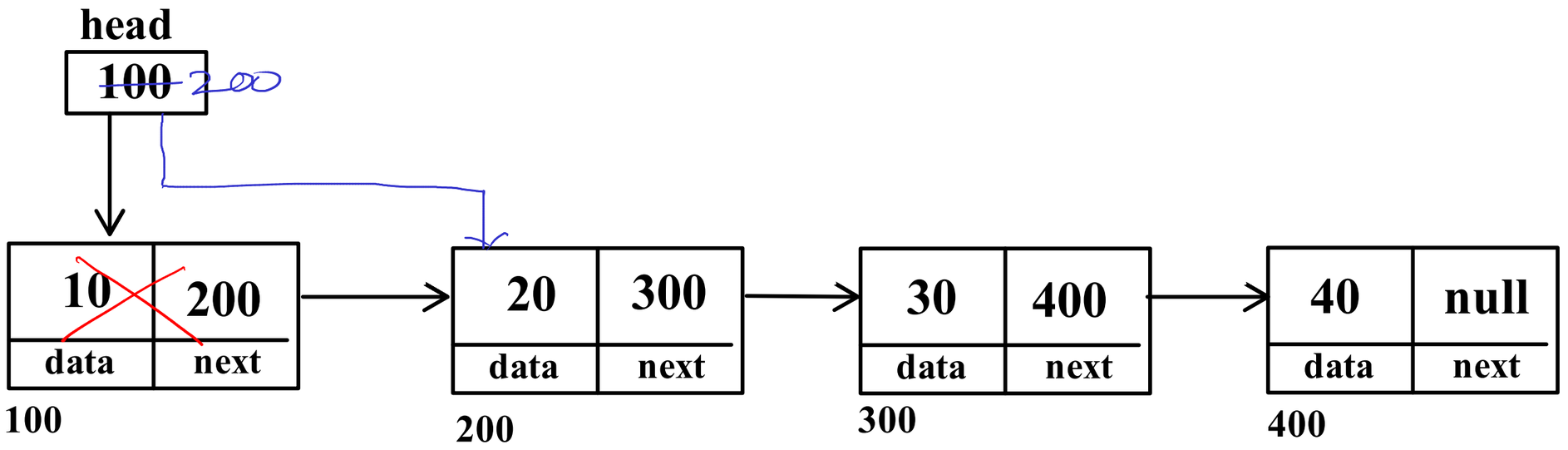
Singly Linear Linked List - Add Last

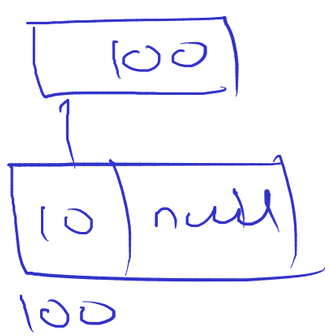





```
//1. create node with given data
//2. if list is empty
        //a. add newnode into head
//3. if list is not empty
        //a. traverse till last node (trav.next != null)
        //b. add newnode into next of last node
```

Singly Linear Linked List - Delete First

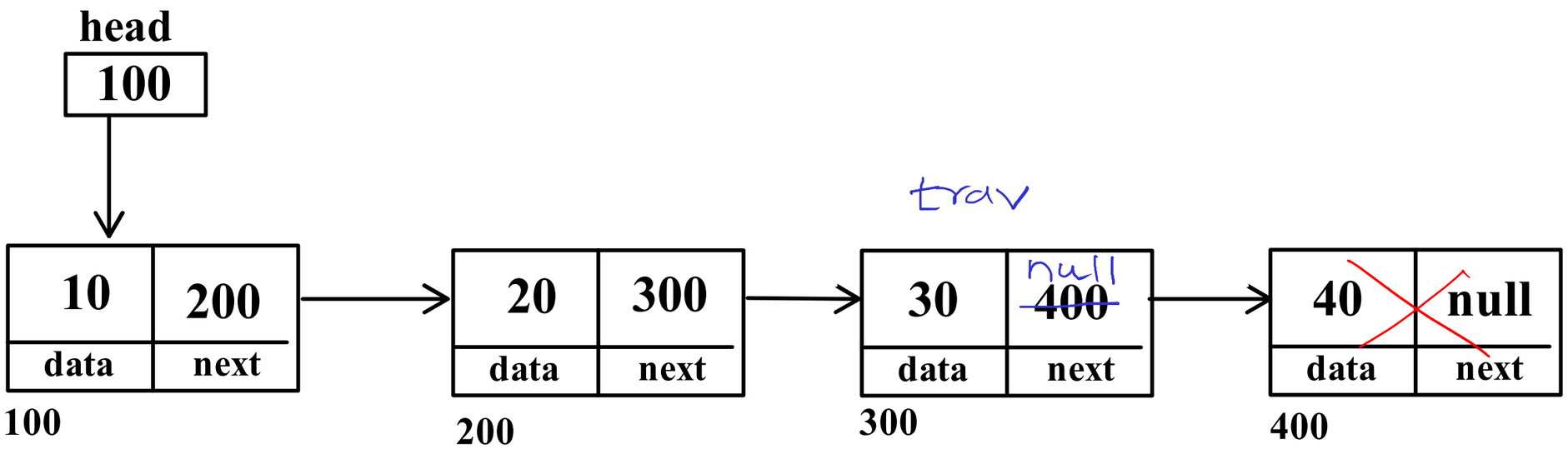




//1. if list is not empty

//a. move head on second node

Singly Linear Linked List - Delete Last



//1. if list has single node

//a. make head equal to null

//2. if list has multiple nodes

//a. traverse till second last node

//b. make next of second last node equal to null

trav != null → trav = null

trav.next != null → trav = addr of last node

trav.next.next != null → trav = addr of second last node