Postfix Evaluation

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

left ⟶ right

Result = 16

③ 23 − 7
= 16

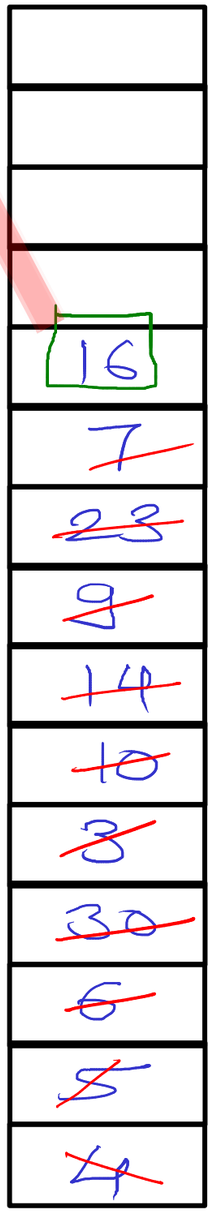
④ 14 + 9
= 23

③ 4 + 10
= 14

② 30 / 3
= 10

① 5 * 6
= 30

Stack



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

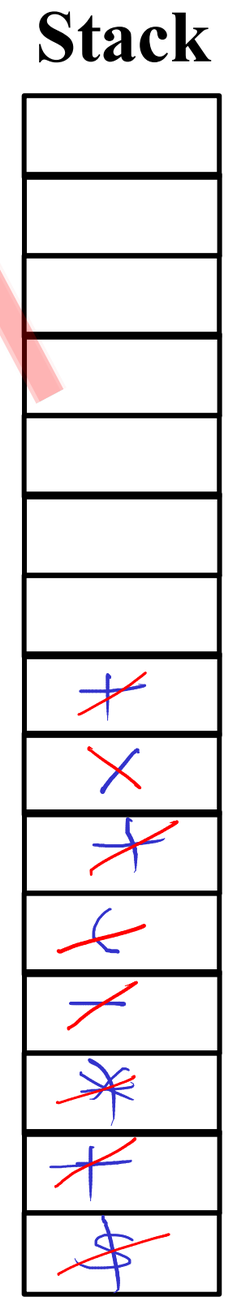
Stack

Infix to Postfix conversion

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

left → right

Postfix : 19$34*+682/+-7+



Infix to Prefix conversion

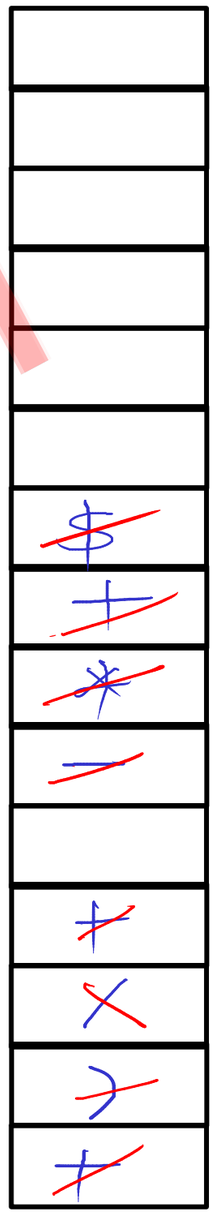
Infix : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7

left ← right

Expression: 728/6+43*91$+-+

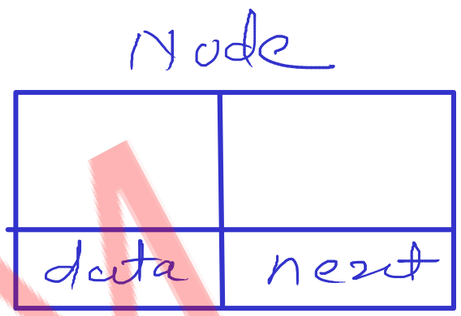
Prefix : +-+$19*34+6/827

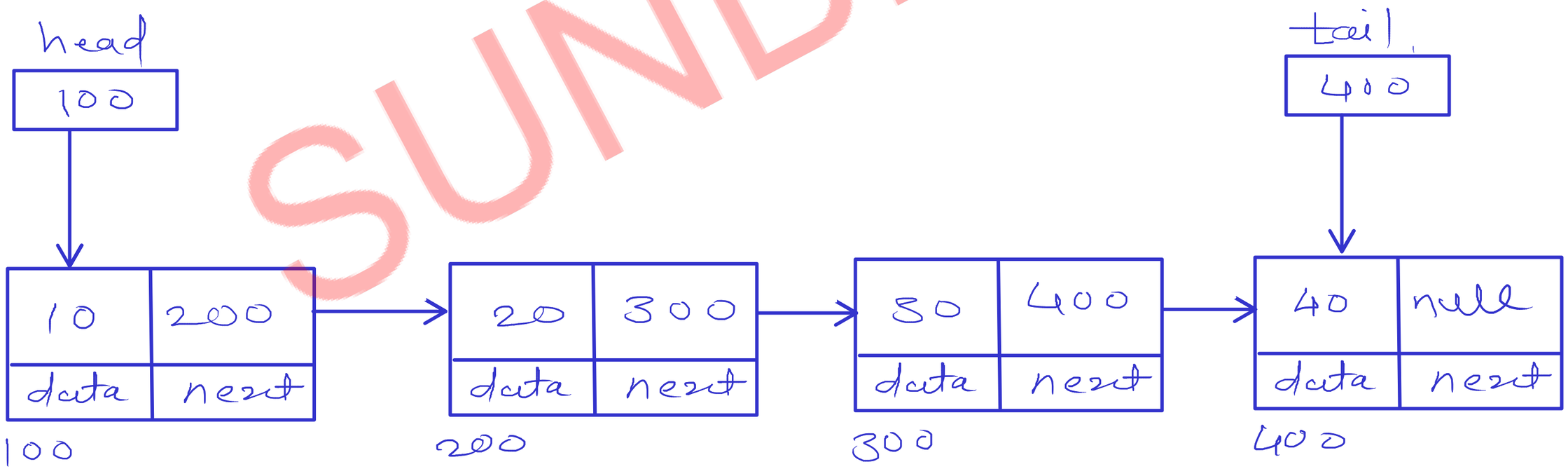
Stack



Linked List

- a linear data structure in which address of next data is kept with current data
- data element in linked list is known as "Node"
- Node will consist two parts:
  - data - actual data
  - link(next) - address of next node
- address of first node is kept into head pointer/referance
- address of last node is kept into tail pointer/referance (optional)





Linked List - Operations

1. Add First
2. Add Last
3. Add at pos (insert)

4. Delete First
5. Delete Last
6. Delete at pos (remove)

7. Travarse (Display)

8. search
9. sort
10. reverse
11. find Mid

Linked List - Types

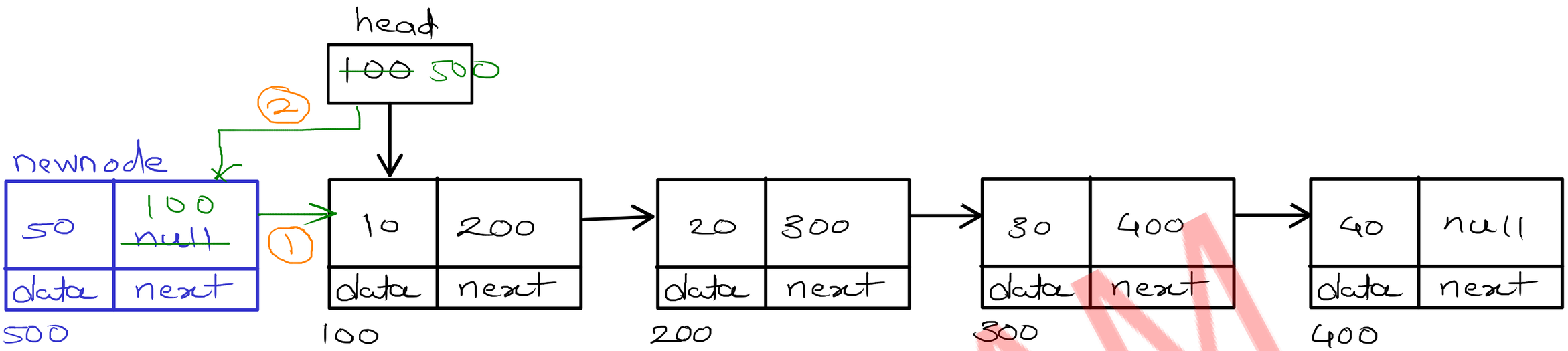
1. Singly Linear linked list
2. Singly Circular linked list
3. Doubly Linear linked list
4. Doubly Circular linked list

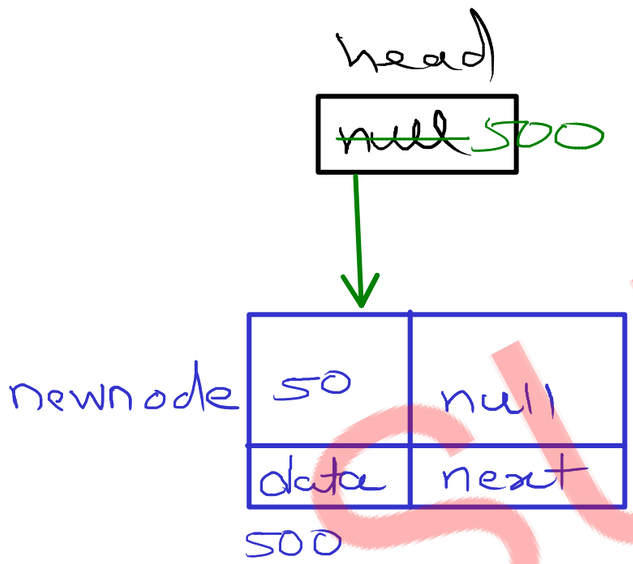
```
class List{
        class Node{
                private int data;
                private Node next;
                public Node(value){}
        }

        private Node head;
        public List(){}
        public isEmpty(){}
        public add(value){}
        public delete(){}
        public display(){}
}
```

Singly Linear Linked List - Add First

Make Before Break

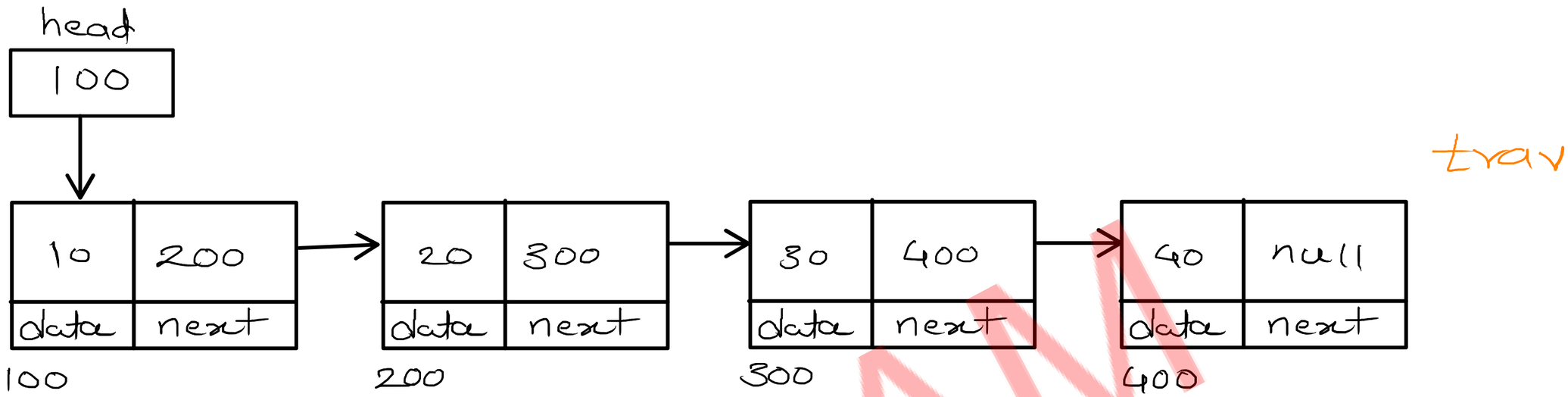




//1. create node with given value
//2. if list is empty
//a. add newnode into head itself
//3. if list is not empty
//a. add first node into next of newnode
//b. move head on newnode

Time Complexity : O(1)

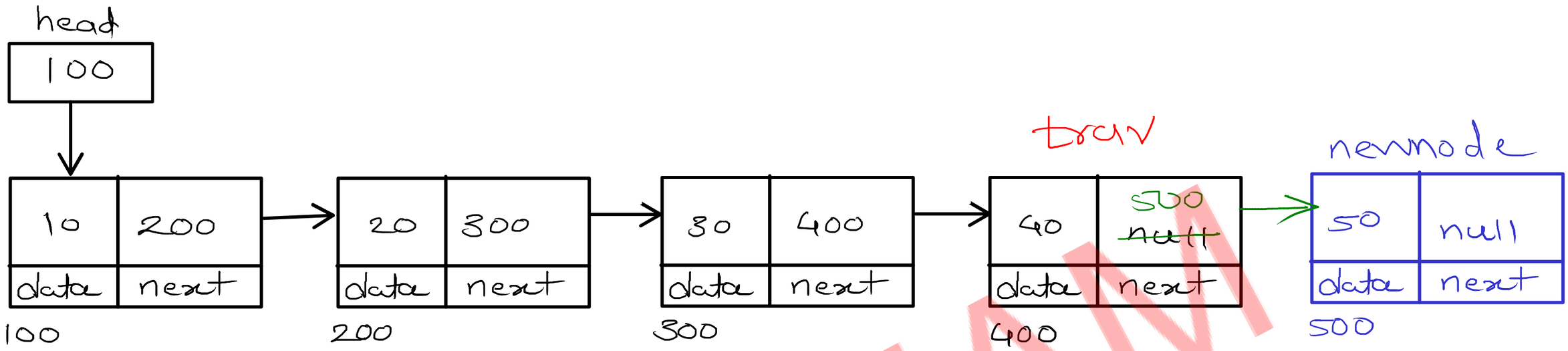
Singly Linear Linked List - Display



//1. create trav referance and start at head
//2. print data of current node (trav.data)
//3. go on next node (trav.next)
//4. repeat step 2 and 3 till last node

Time Complexity : O(n)

Singly Linear Linked List - Add Last

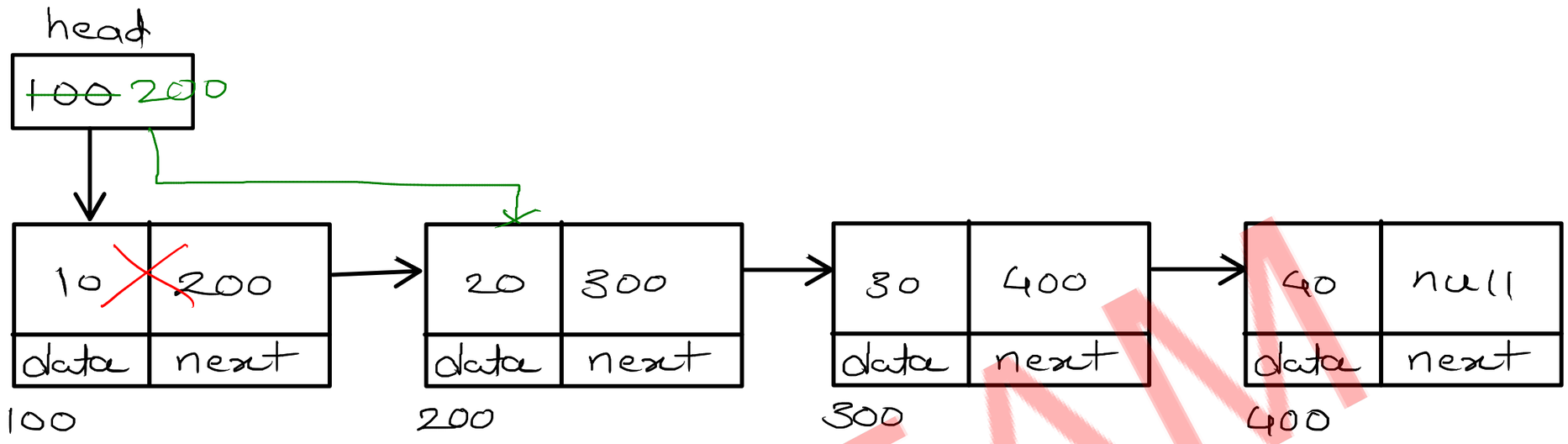



```
//1. create node with given value
//2. if list is empty
        //a. add newnode into head itself
//3. if list is not empty
        //a. traverse till last node
        //b. add newnode into next of last node
```

Time Complexity : O(n)

```
trav = head;
while(trav.next != null)
{
    trav = trav.next;
}
```

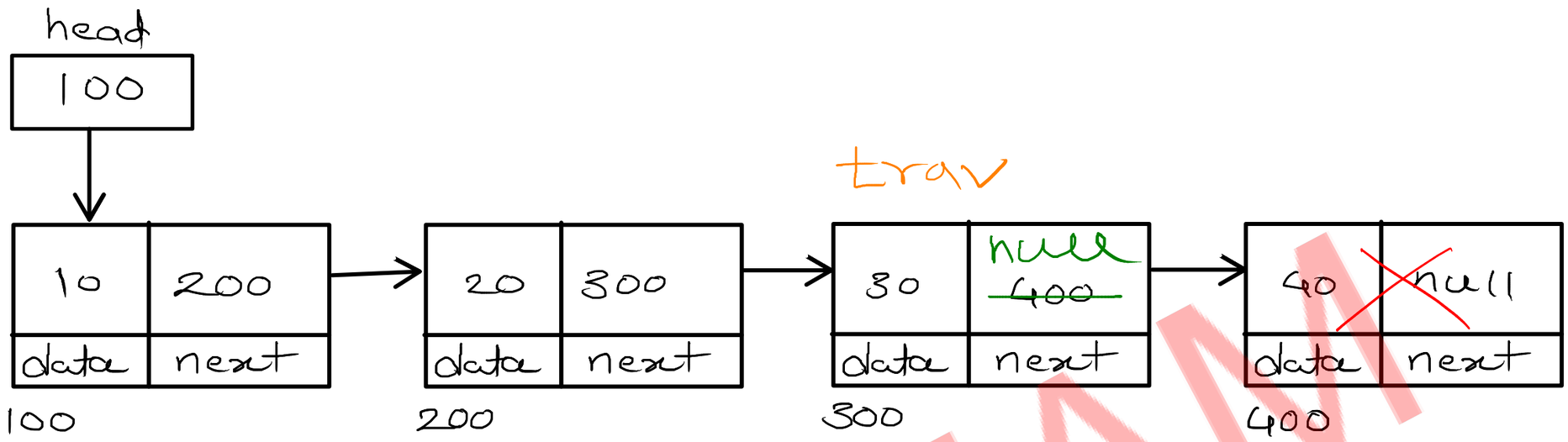
Singly Linear Linked List - Delete First




```
//1. if list is empty
    // do nothing
//2. if list is not empty
    //a. move head on second node
```

Time Complexity : O(1)

Singly Linear Linked List - Delete Last



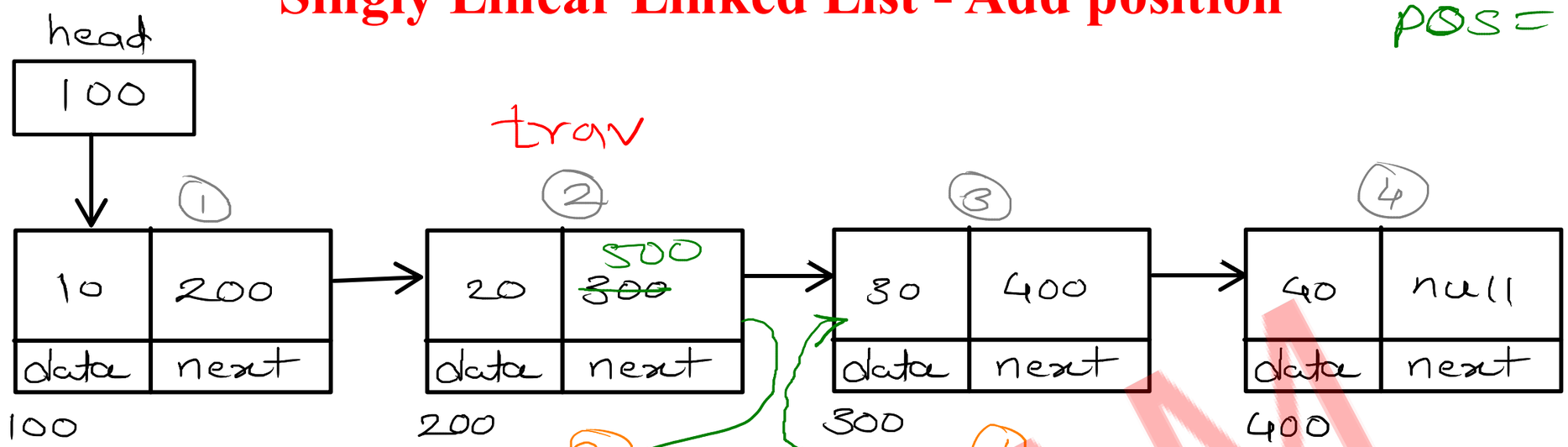

```
trav = head
while(trav.next.next != null)
    trav = trav.next;
```


```
//1. if list is empty
      // do nothing
//2. if list has single node
      // make head equal to null
//3. if list has multiple nodes
      //a. traverse till second last node
      //b. add null into next of second last node
```

Time Complexity : O(n)

Singly Linear Linked List - Add position

pos = 3




```
trav = head
for(i = 1; i < pos-1; i++)
    trav = trav.next;
```

pos = 3

trav	i	i < 2
100	1	T
200	2	F

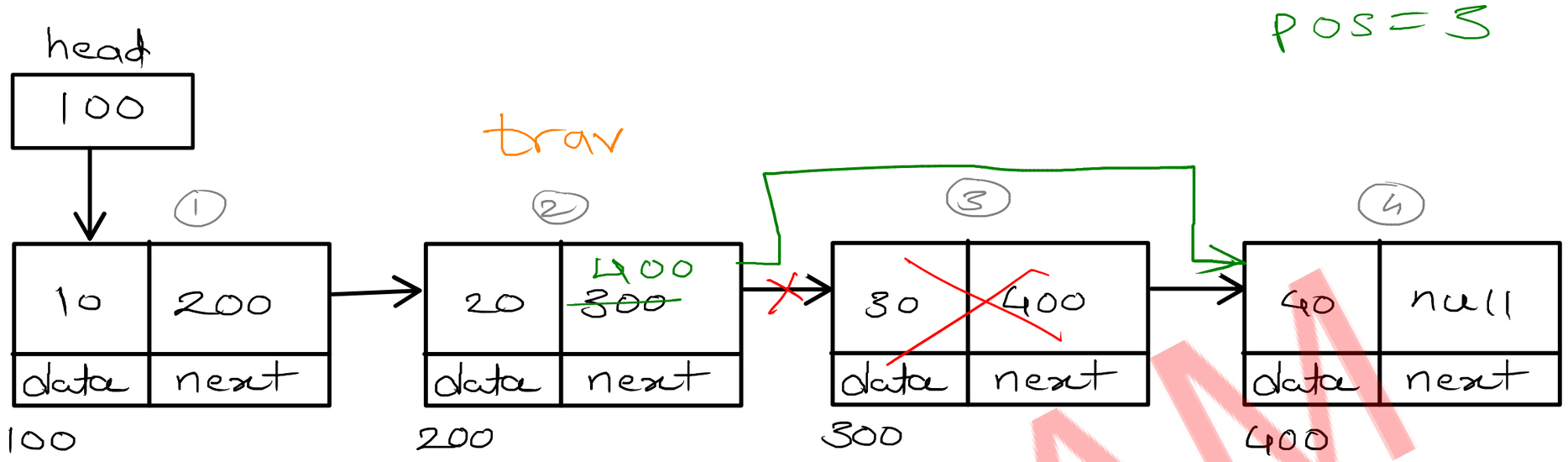
pos = 5

trav	i	i < 4
100	1	T
200	2	T
300	3	T
400	4	F

Time Complexity : O(n)

```
//1. create node with given value
//2. if list is empty
      //add newnode into head
//3. if list is not empty
      //a. traverse till pos -1 node
      //b. add pos node into next of newnode
      //c. add newnode into next of pos-1 node
```

Singly Linear Linked List - Delete position




```
//1. if list is empty
    // do nothing
//2. if list is not empty
    //a. traverse till pos -1 node
    //b. add pos+1 node into next of pos-1node
```

Time Complexity : O(n)