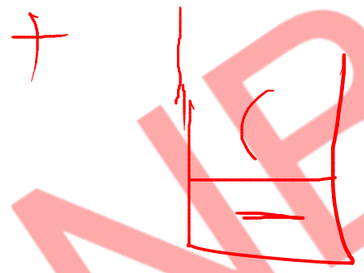
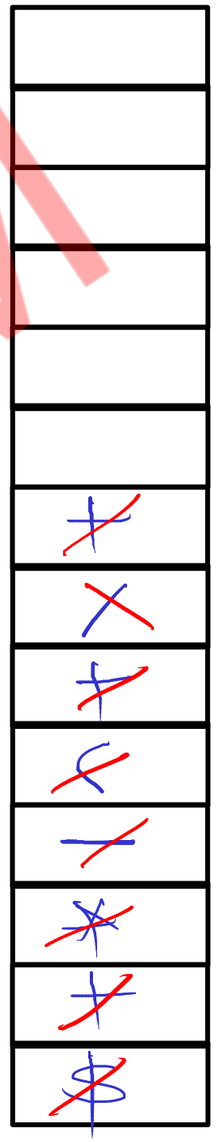
Infix to Postfix conversion

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

left → right

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





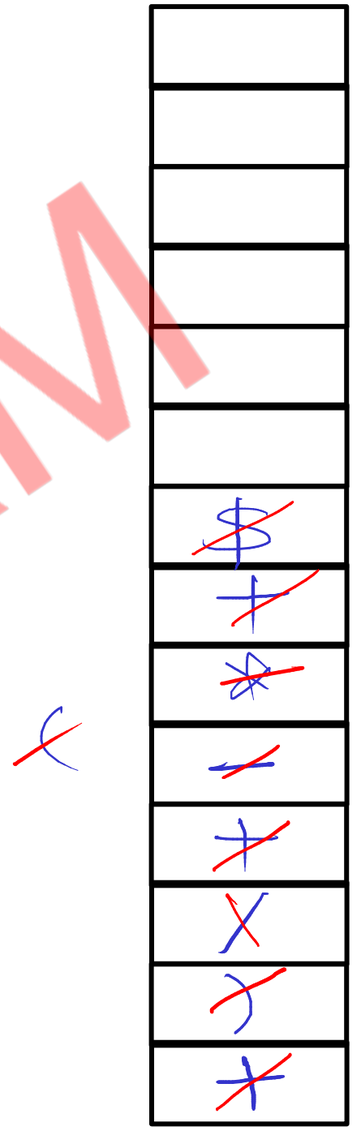
Infix to Prefix conversion

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

left ← right

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

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

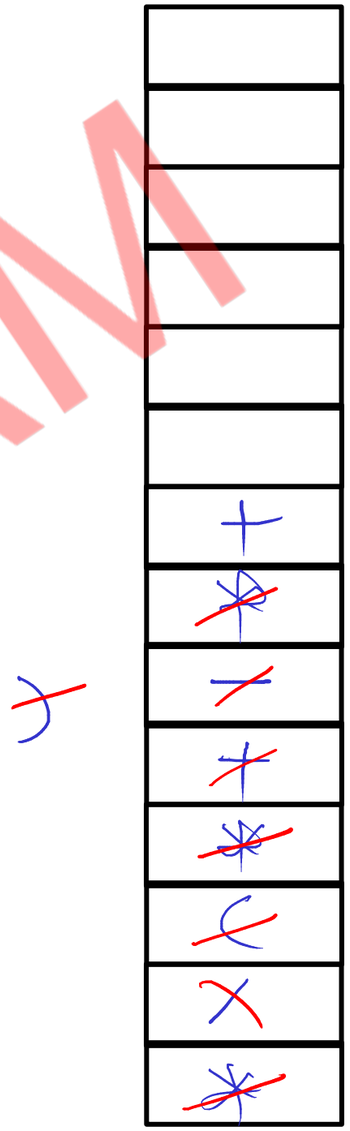


Infix to Postfix conversion

Infix : a * b / (c * d + e) - f * h + i

left → right

Postfix : a b * c d * e + / f h * - i +



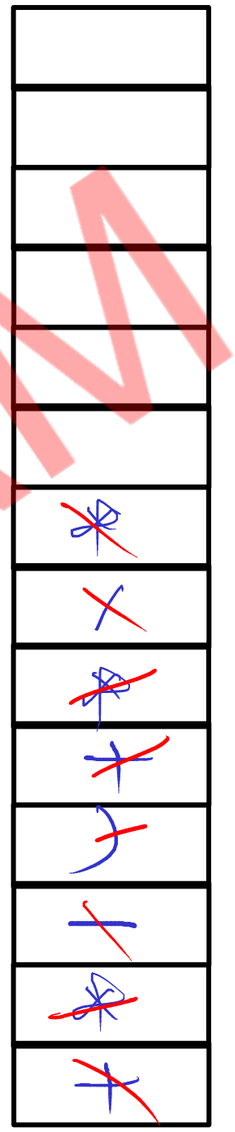
Infix to Prefix conversion

Infix : a * b / (c * d + e) - f * h + i

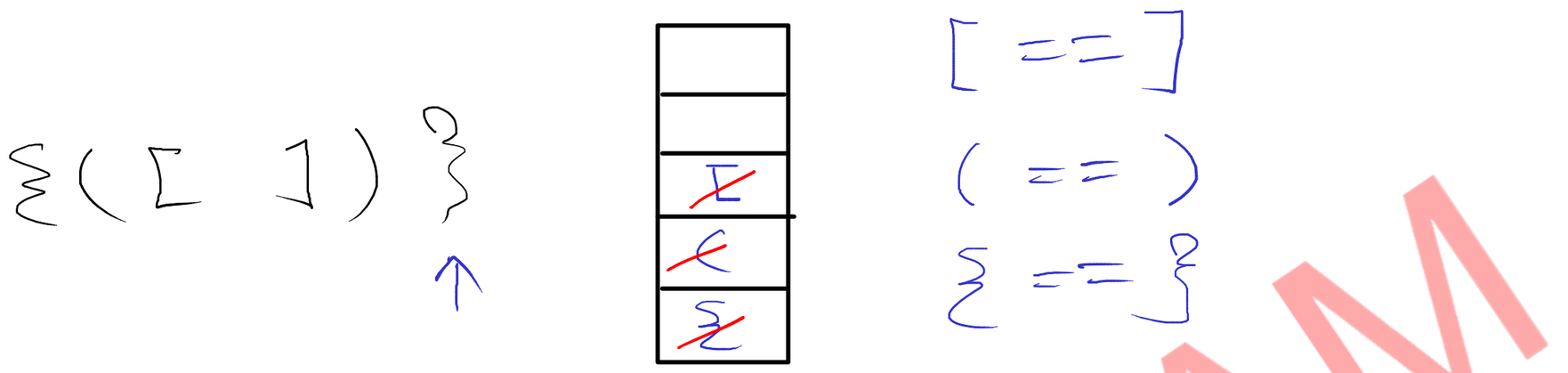
left ← right

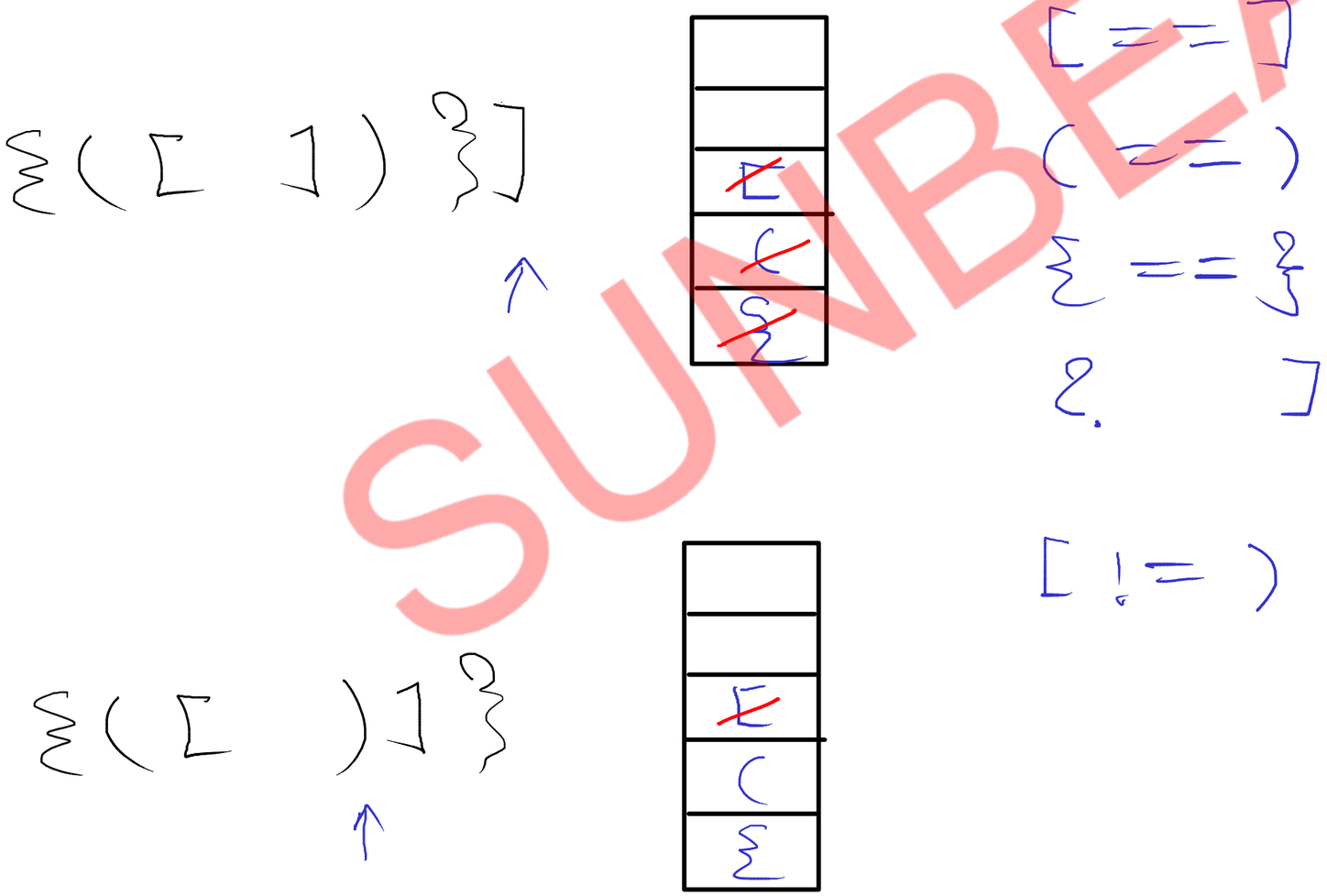
Expression : ihf*edc*+ba*/-+

Prefix : +-/*ab+*cde*fhi



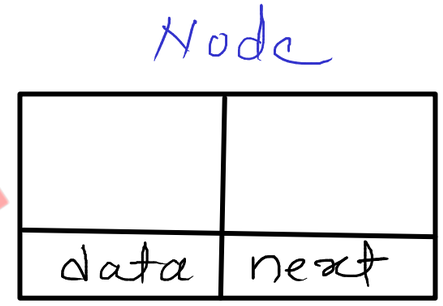
Paranthesis balancing using stack



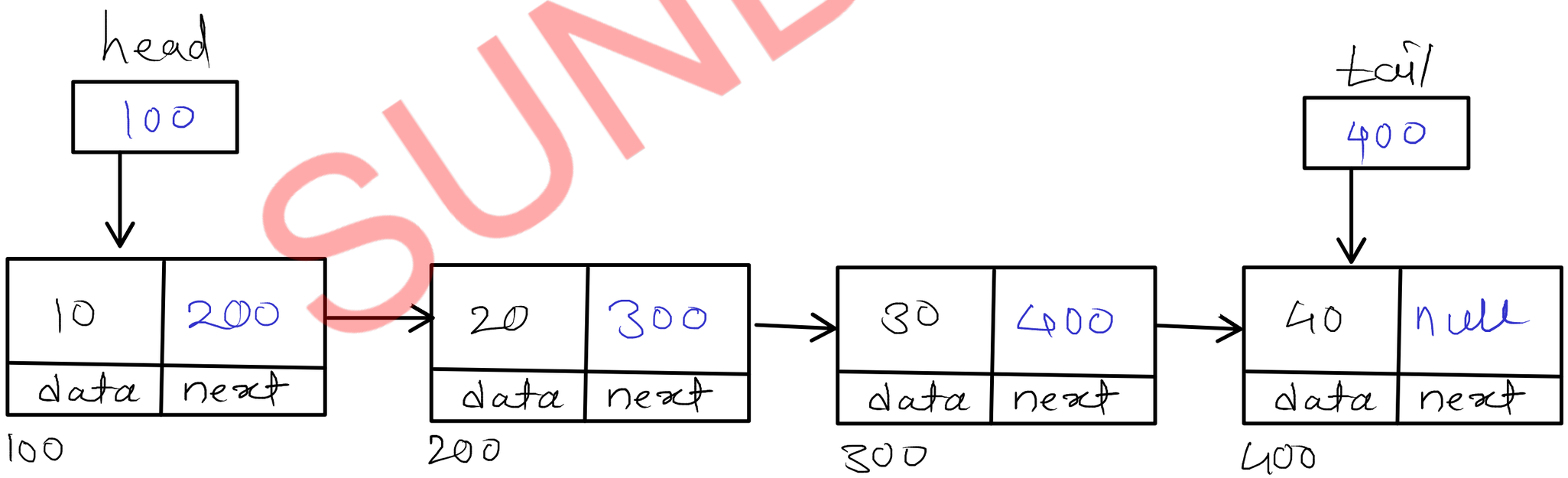


Linked List

- it is linear data structure which stores similar type of data
- link (address/reference) of next data is kept with previous data
- every element of linked list is known as Node
- node consist of two parts
    1. data - actual data of the node
    2. link/next - address/reference of next node



- address of first node is always kept into head (pointer/reference)
- address of last node is always kept into tail (pointer/reference) (optional)



Operations

1. Add first
2. Add last
3. Add pos (in between)

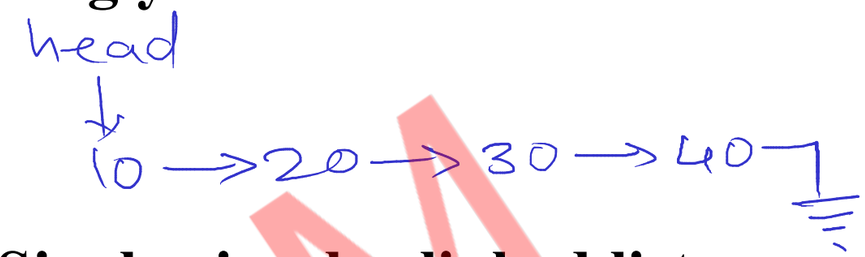
4. Delete first
5. Delete last
6. Delete pos (in between)

7. Display (traverse)

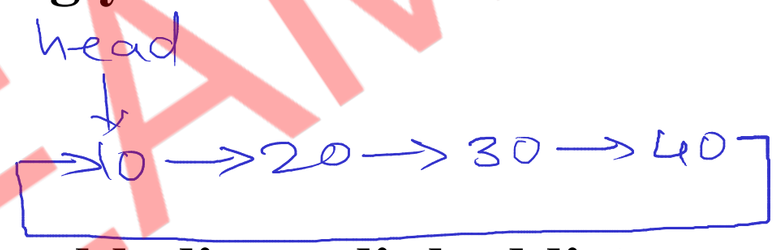
8. serach
9. sort
10. reverse
11. find mid

Types

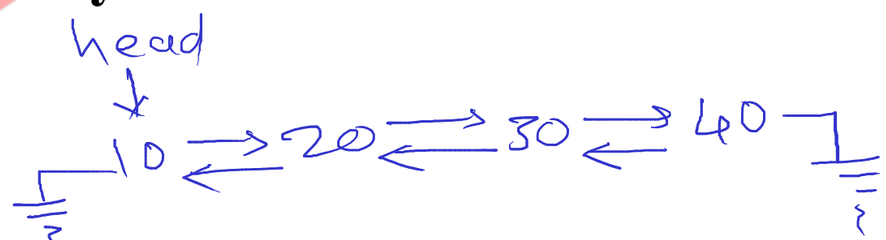
1. Singly linear linked list



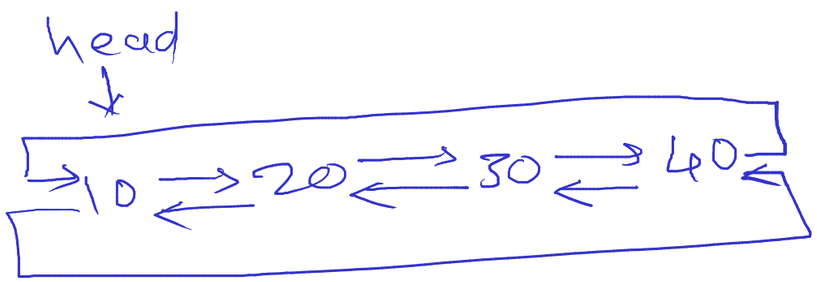
2. Singly circular linked list



3. Doubly linear linked list



4. Doubly circular linked list



Node

- **data : int, char, strings, long, user defined type**
- **next : reference**

```
class Node{
    int data;
    Node next;
}
class List{
    static class Node{
        int data;
        Node next;
    }
    Node head;
    Node tail;
    int count;
    public List(){...}
    public isEmpty(){...}
    public addNode(){...}
    public deleteNode(){...}
    public deleteAll(){...}
    public displayList(){...}
}
```

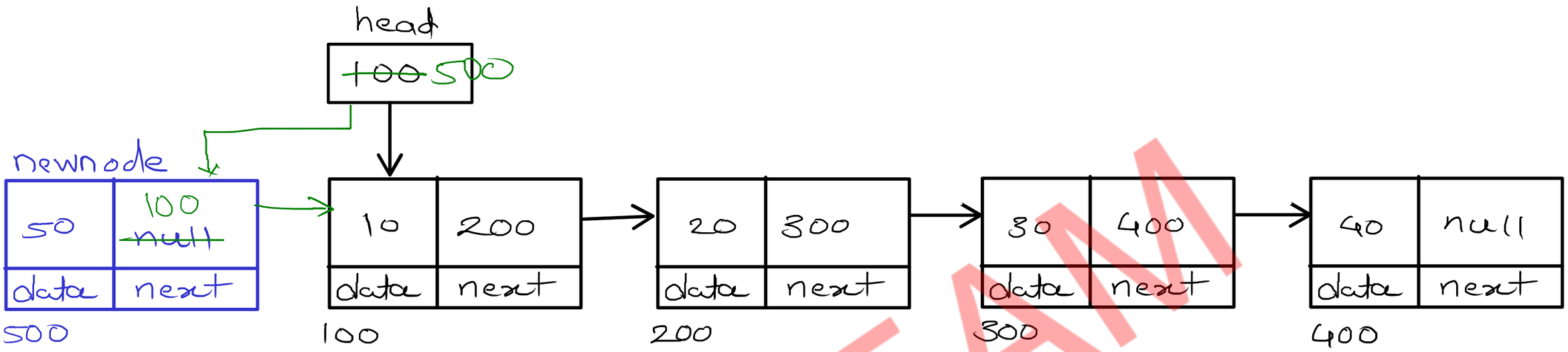
Self referential class

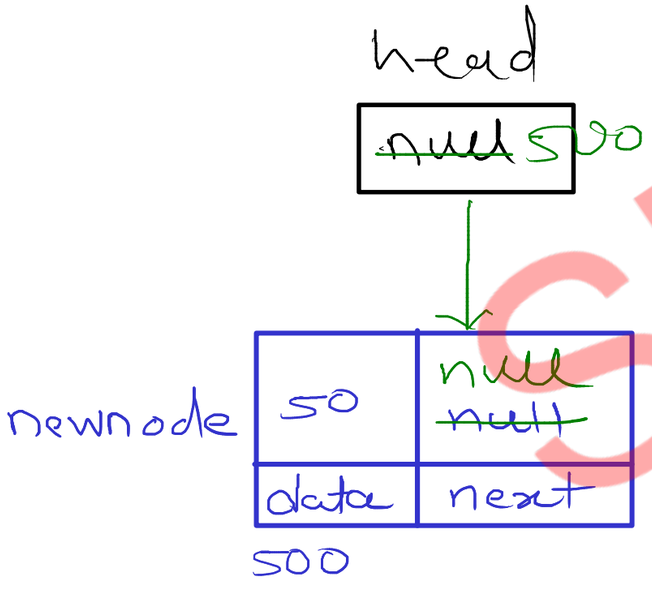
reference of same Node type is kept into Node class

① no dependency of List class to create object of Node class

② non static fields of outer class are not directly accessible into inner class

Singly Linear Linked List - Add First



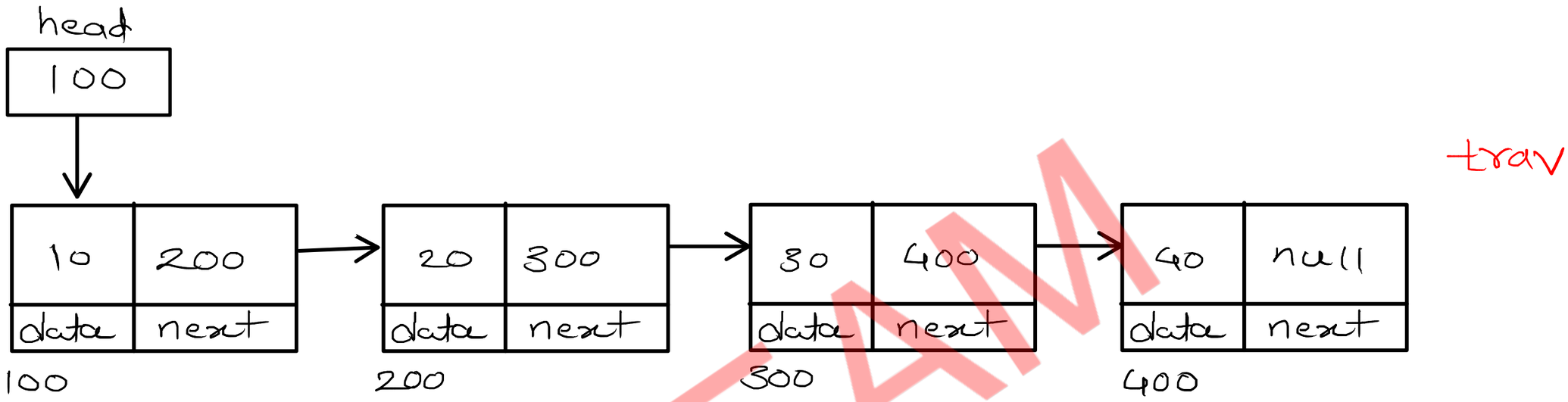


//1. create node with given data
//2. add first node into next of newnode
//3. move head on newnode

```
newnode.next = head;
head = newnode;
```

$T(n) = O(1)$

Singly Linear Linked List - Display

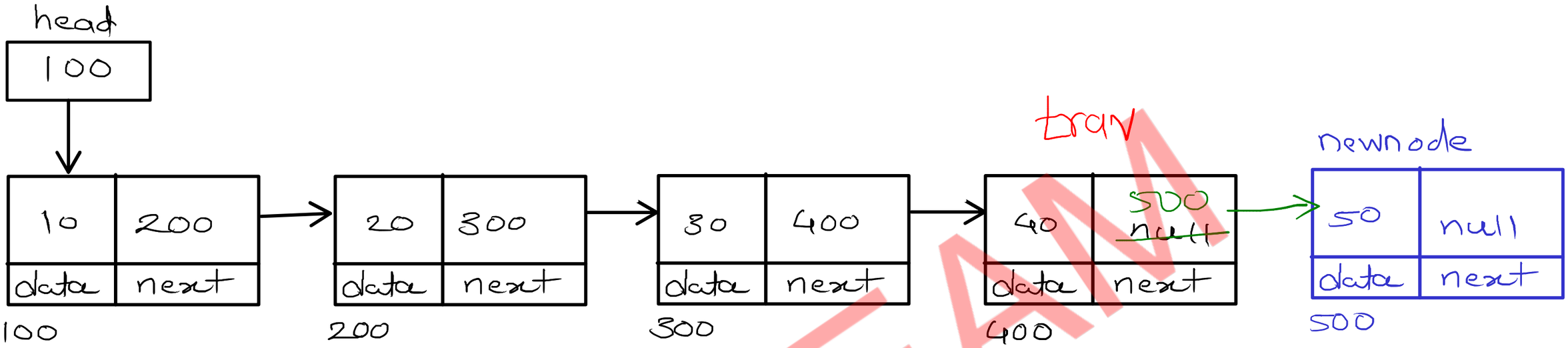


trav	trav.data	trav.next
100	10	200
200	20	300
300	30	400
400	40	null
null		

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

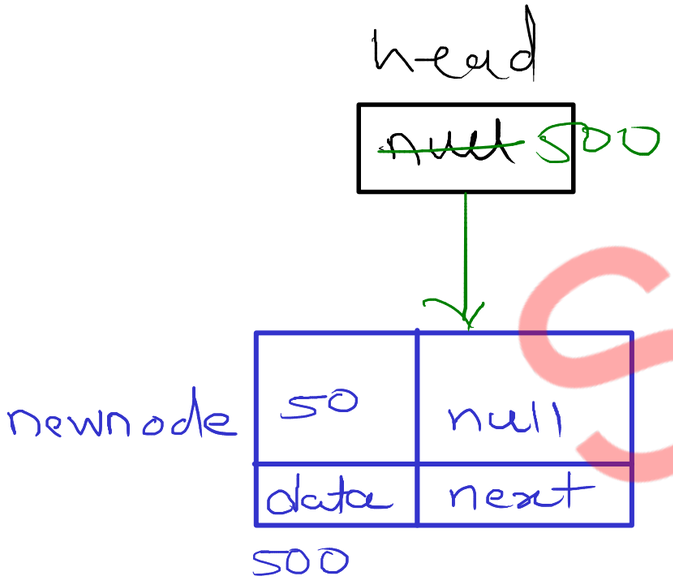
$$T(n) = O(n)$$

Singly Linear Linked List - Add Last




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

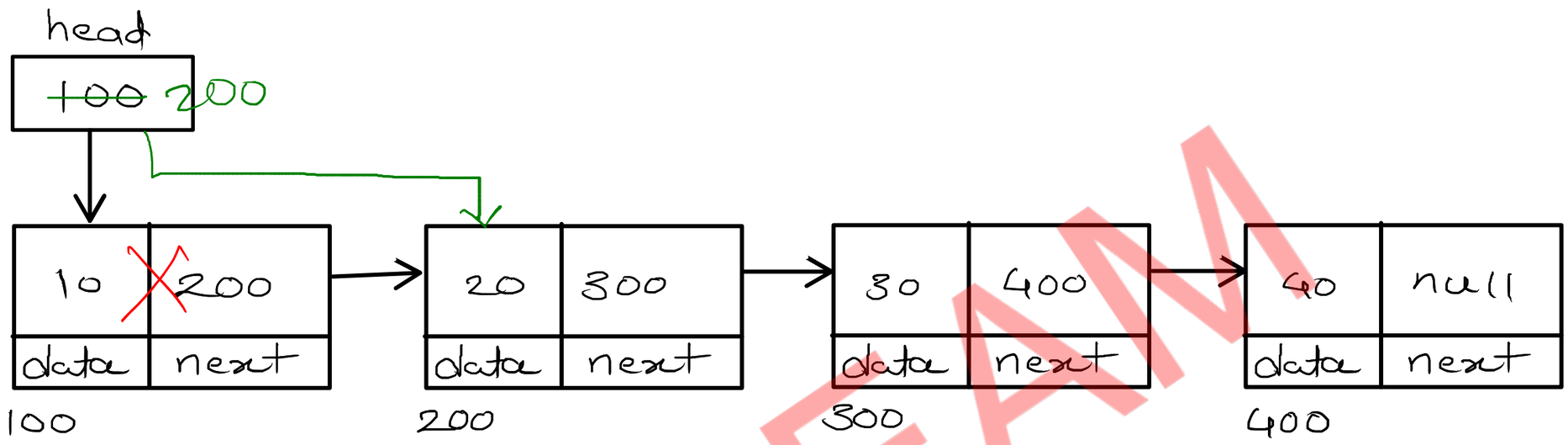
trav	trav.next
100	200
200	300
300	400
400	null

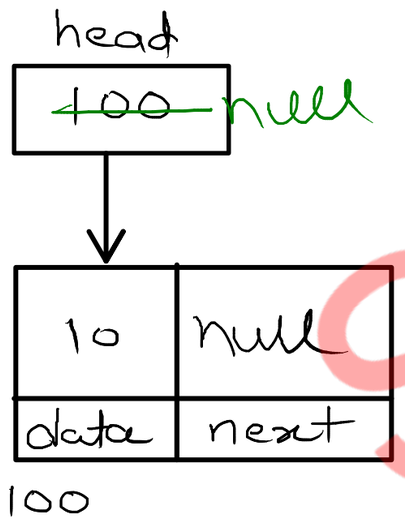


//1. create node with given data
//2. if list is empty
//add newnode into head itself
//3. if list is not empty
//a. traverse till last node
//b. add newnode into next of last node

$$T(n) = O(n)$$

Singly Linear Linked List - Delete First

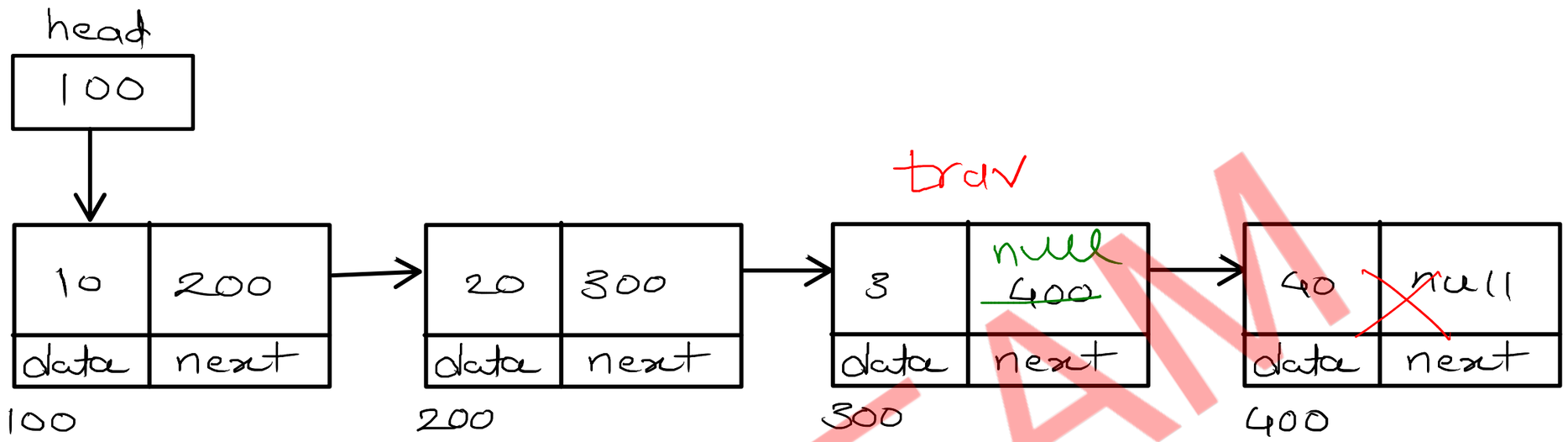





```
//0. if list is empty
   return;
//1. move head on second node
```

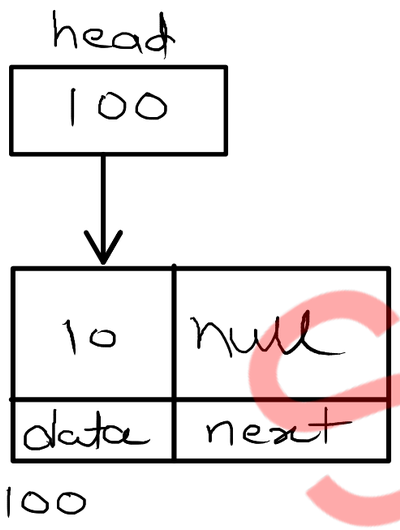
$T(n) = O(1)$

Singly Linear Linked List - Delete Last




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

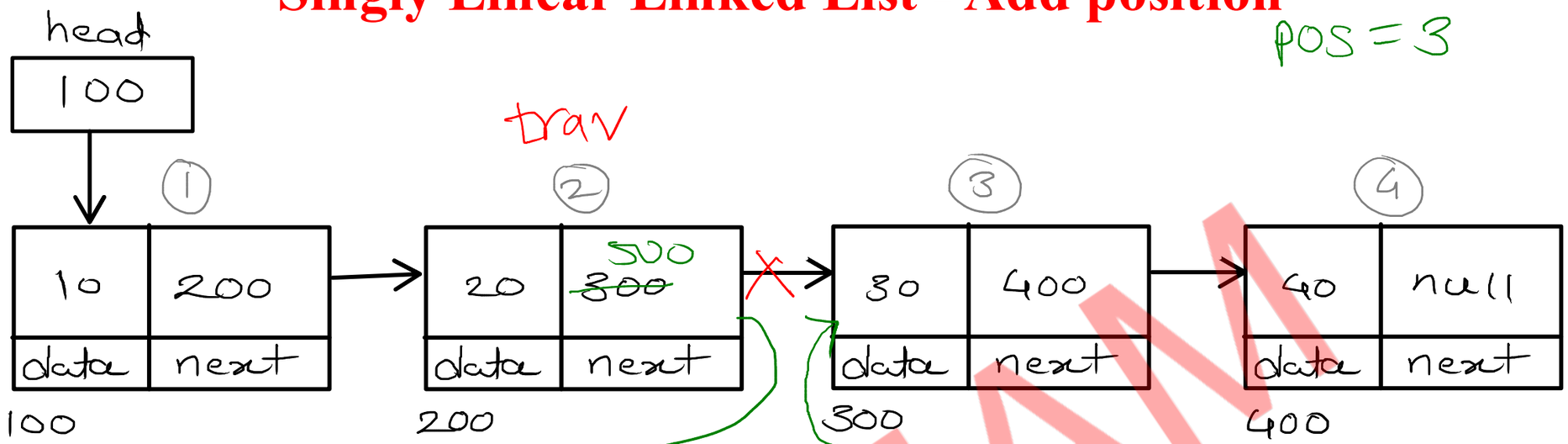
trav	trav.next.next
100	300
200	400
300	null



$T(n) = O(n)$

```
//1. if list is empty
       return;
//2. if list has single node
       head = null;
//3. if list has multiple nodes
       //a. traverse till second last node
       //b. make next of second last node equal to null
```

Singly Linear Linked List - Add position




```
Node 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

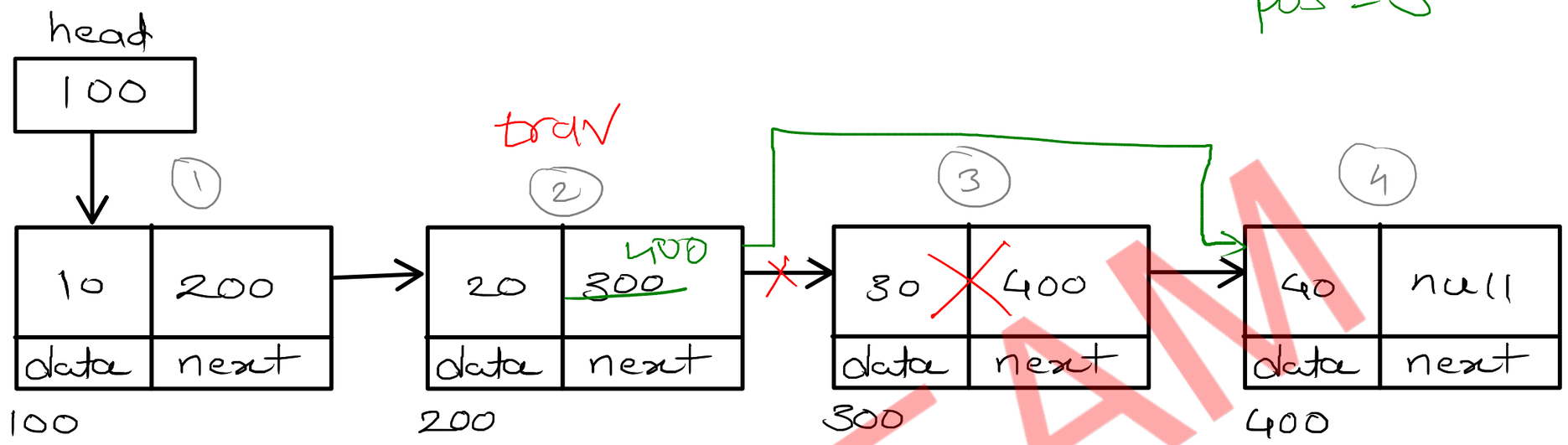
trav	i	i<4
100	1	T
200	2	T
300	3	T
400	4	F

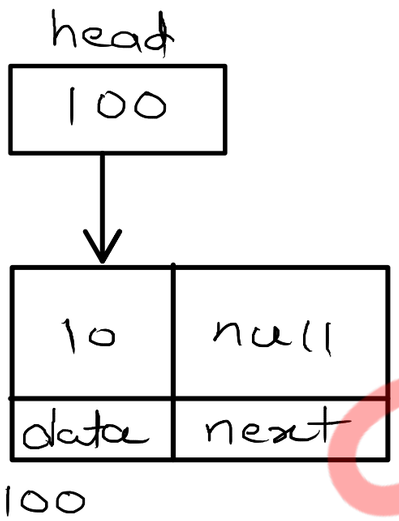
make before break

//1. create newnode with given data
//2. if list is empty
head = newnode;
//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

$T(n) = O(n)$

Singly Linear Linked List - Delete position






```
//1. if list is empty
     return;
//2. if list has single node
     head = null;
//3. if list has multiple node
     //a. traverse till pos -1 node
     //b. add pos + 1 node into next of pos - 1 node
```

T(n) = O(n)