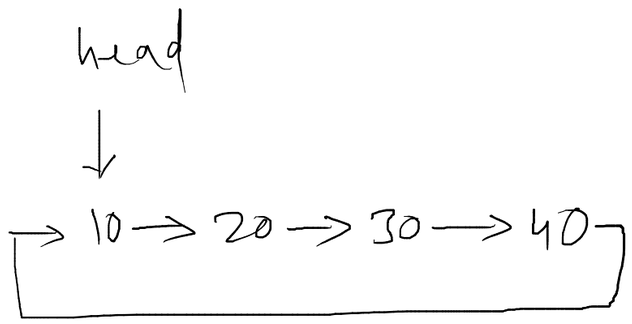
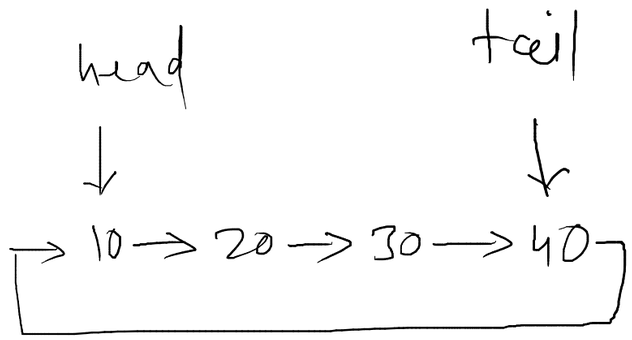
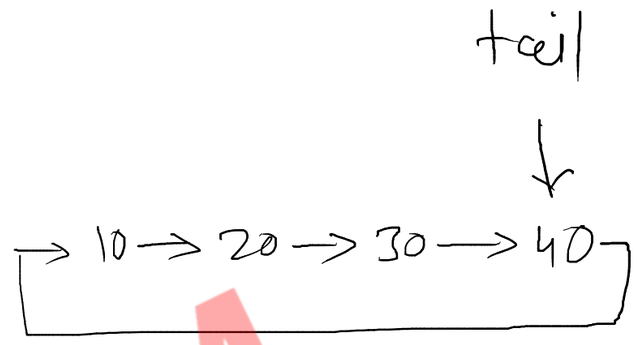


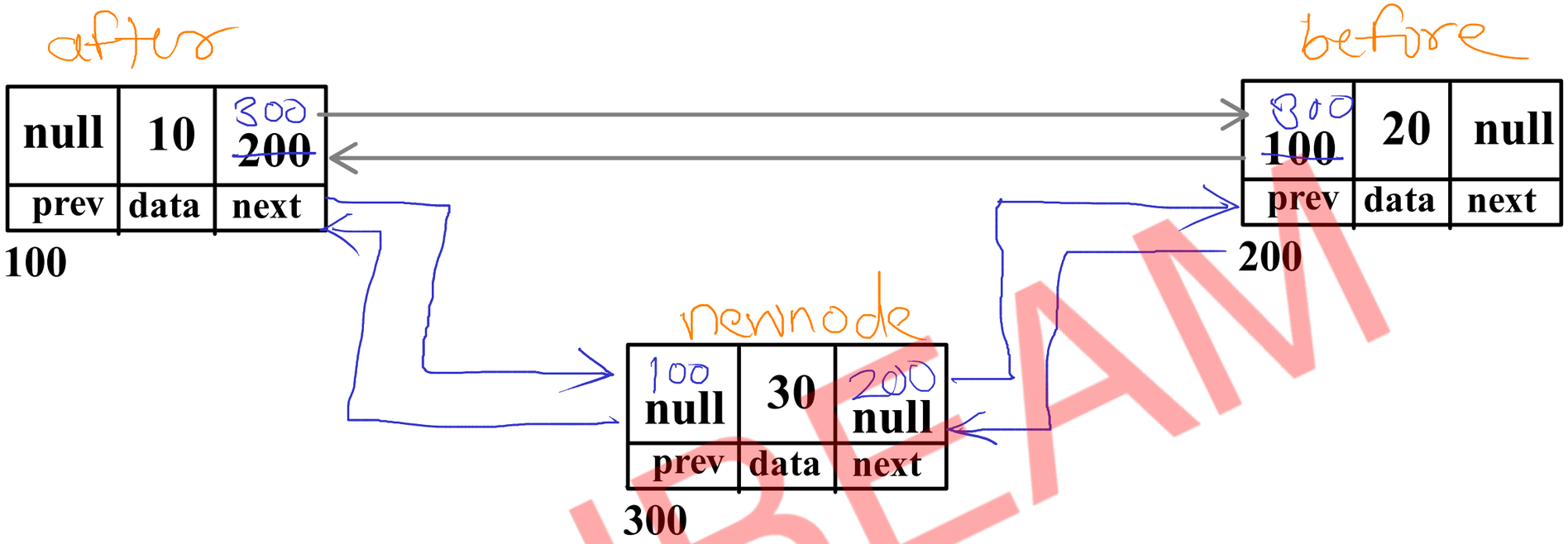
Addfirst → O(n)
Add Last → O(n)
Add Pos → O(n)
Delete first → O(n)
Delete Last → O(n)
delete Pos → O(n)
Traversal → O(n)



Addfirst → O(1)
Add Last → O(1)
Add Pos → O(n)
Delete first → O(1)
Delete Last → O(n)
Delete Pos → O(n)
Traversal → O(n)



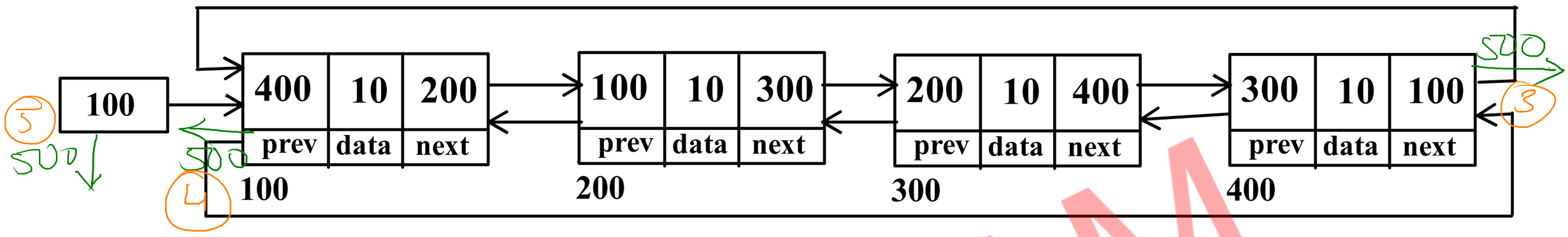
Addfirst → O(1)
Add Last → O(1)
Add Pos → O(n)
Delete first → O(1)
Delete Last → O(n)
delete Pos → O(n)
Traversal → O(n)

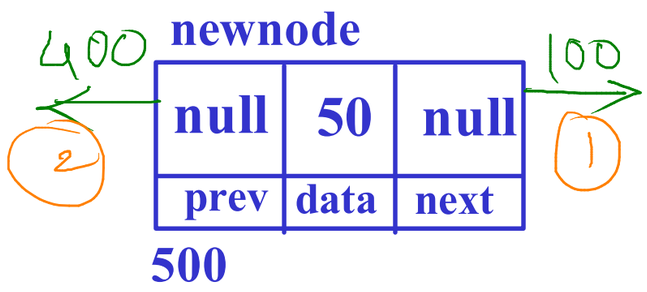


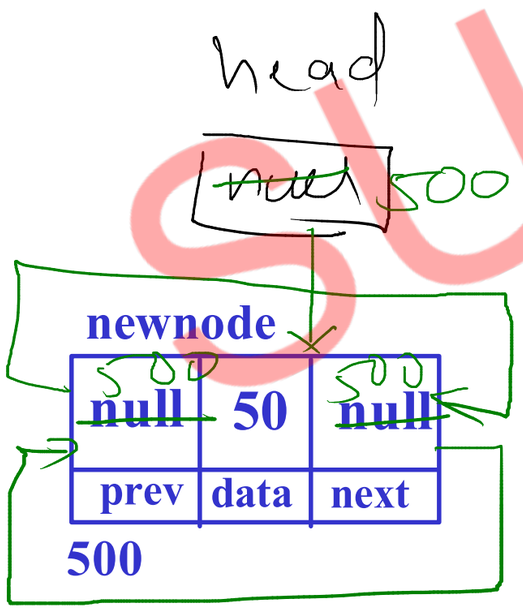
//1. add before node into next of newnode
//2. add after node into prev of newnode
//3. add newnode into next of after node
//4. add newnode into prev of before node

```
newnode.next = before;
newnode.prev = after;
after.next = newnode;
before.prev = newnode;
```

Doubly Circular Linked List - Add First



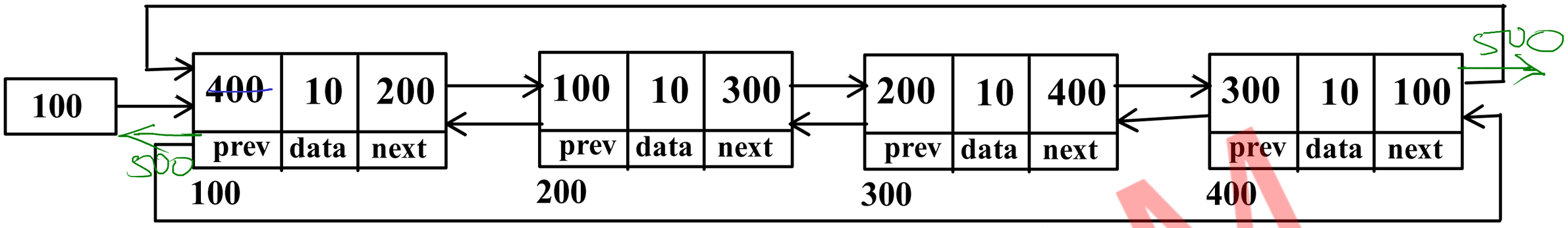


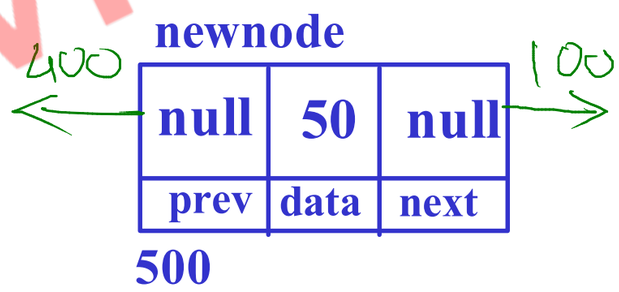


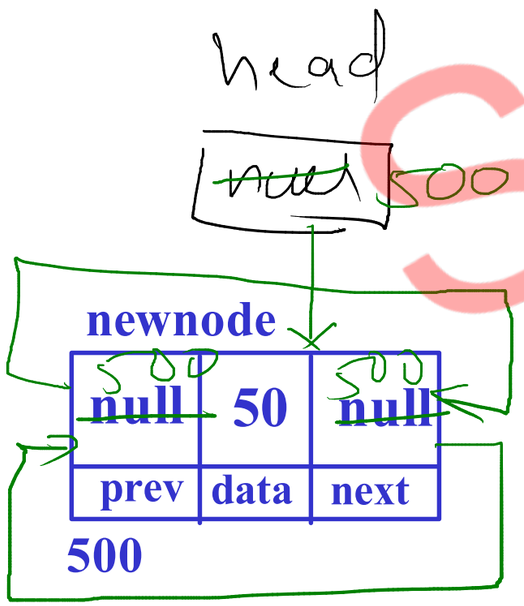

```
//1. create newnode
//2. if list is empty
        //a. add newnode into head
        //b. make list circular
//3. if list is not empty
        //a. call insert method
        //b. move head on newnode
```

T(n) = O(1)

Doubly Circular Linked List - Add Last





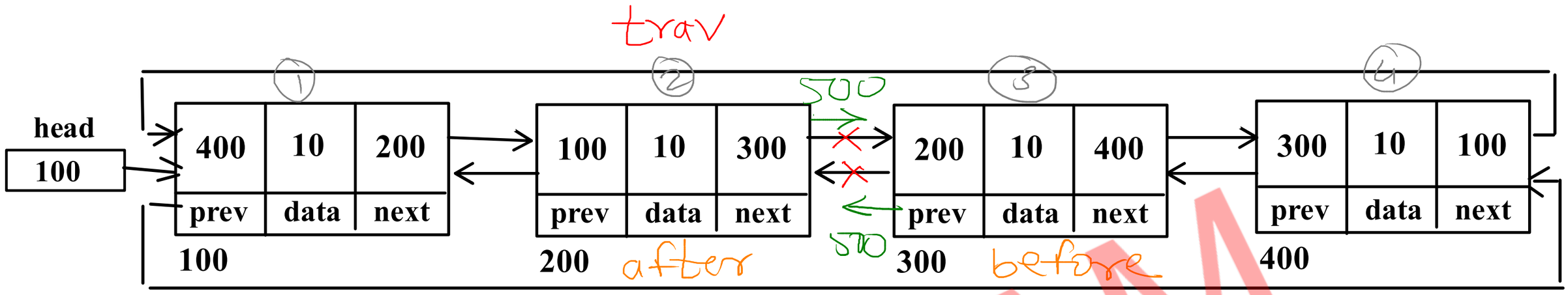


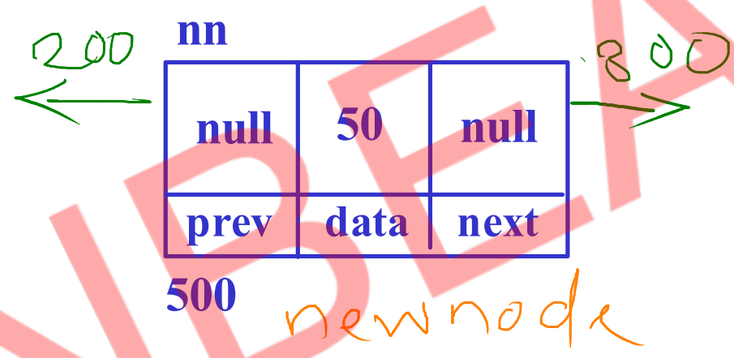

```
//1. create newnode
//2. if list is empty
        //a. add newnode into head
        //b. make list circular
//3. if list is not empty
        //a. call insert method
```

T(n) = O(1)

Doubly Circular Linked List - Add pos

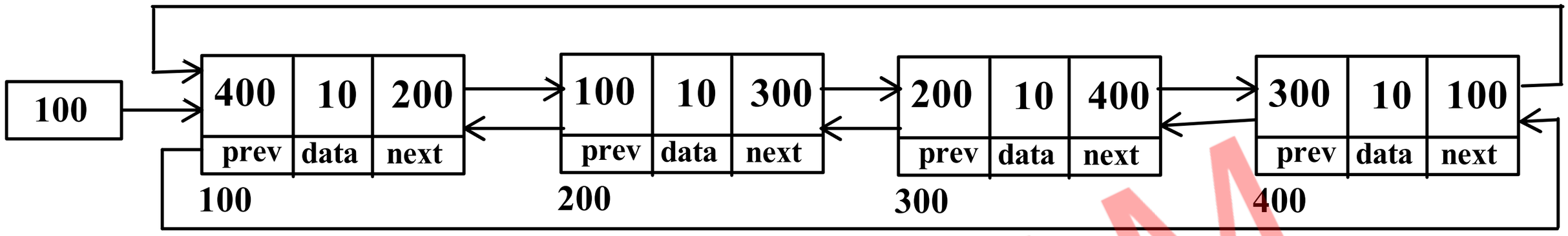
pos=3






```
//1. create newnode
//2. if list is empty
        //a. add newnode into head
        //b. make list circular
//3. if list is not empty
        //a. traverse till pos-1 node
        //b. call insert method
```

Doubly Circular Linked List - Display



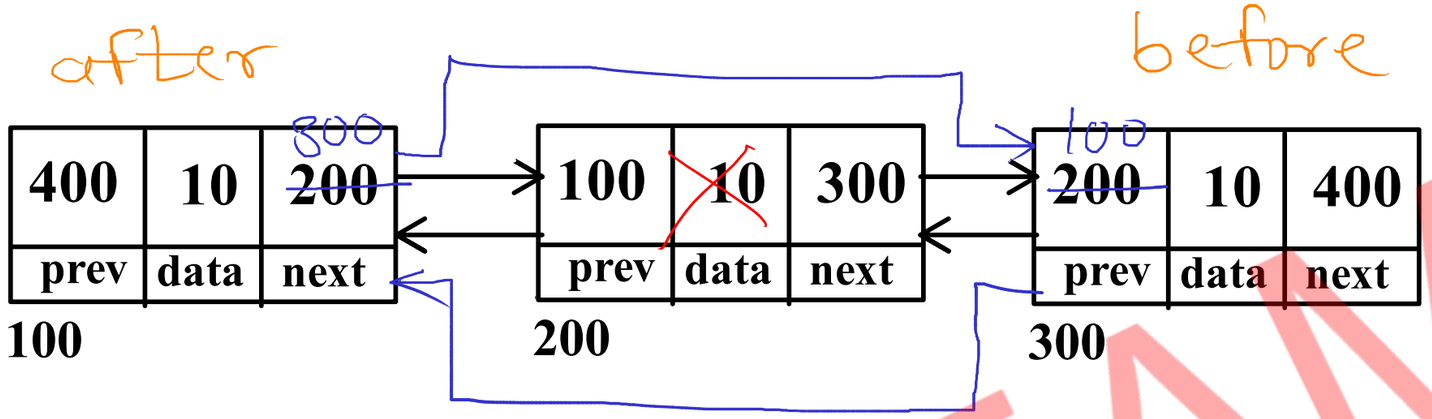
Forward Display

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

Reverse Display

//1. start at last node
//2. print current node (trav.data)
//3. go on prev node (trav.prev)
//4. repeat step 2 and 3 till first node

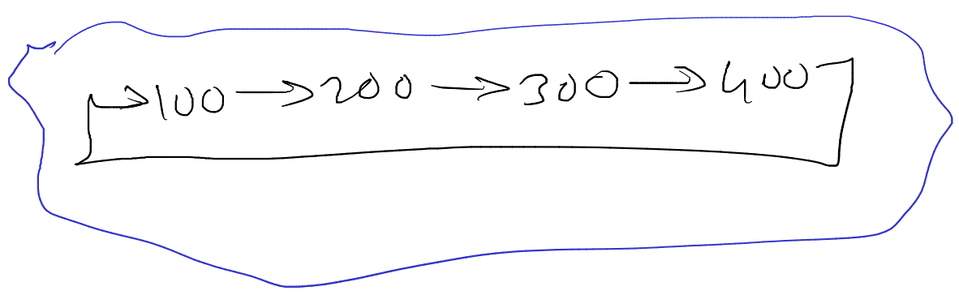
$T(n) = O(n)$



//1. add before node into next of after node
//2. add after node into prev of before node

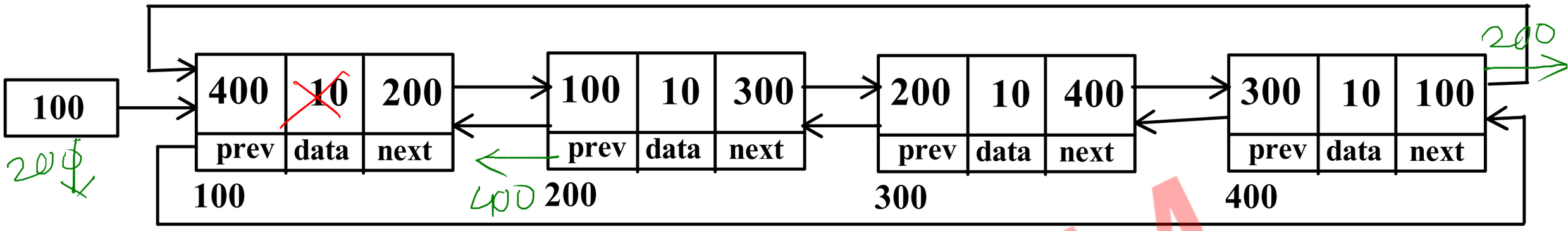
after.next = before
before.prev = after

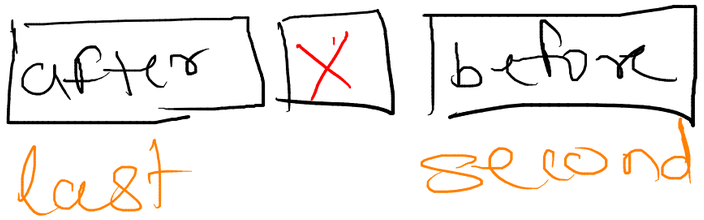
head



Irland of Isolation

Doubly Circular Linked List - Delete First

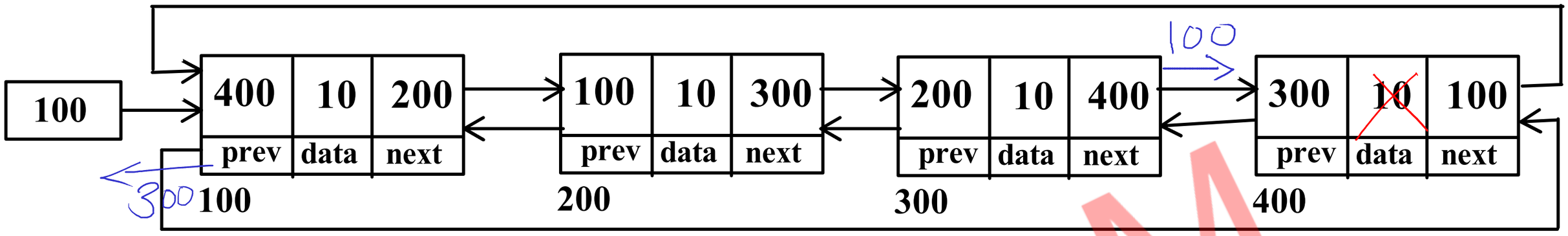


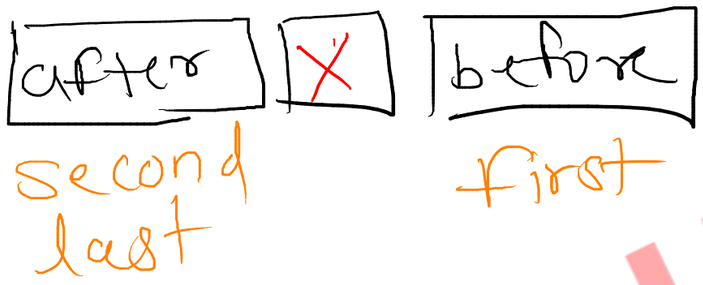



```
//1. if list is emtpy
       return;
//2. if list has single node
       head = null;
//3. if list has multiple nodes
     //a. call delete method
     //b. move head on second node
```

T(n) = O(1)

Doubly Circular Linked List - Delete Last



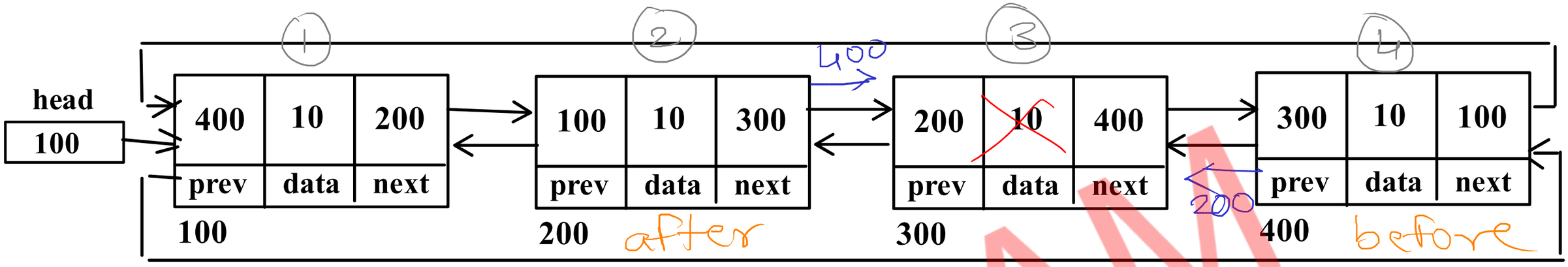


T(n) = O(1)

```
//1. if list is emtpy
        return;
//2. if list has single node
        head = null;
//3. if list has multiple nodes
        //a. call delete method
```

Doubly Circular Linked List - Del Pos

pos = 3



T(n) = O(n)

```
//1. if list is emtpy
      return;
//2. if list has single node
      head = null;
//3. if list has multiple nodes
      //a. traverse till pos node
      //a. call delete method
```

Linked List Applications

- dynamic data structure - grow / shrink at runtime
- due to this dynamic nature, it is used to implement other data structures
  1. Stack
  2. Queue
  3. Hash Table (Seperate chaining)
  4. Graph (Adjacency list)

- Operating system - job queue, ready queue, waiting queues
  (Doubly circular linked list)
- Ring topology in Networks - (Doubly circular linked list)

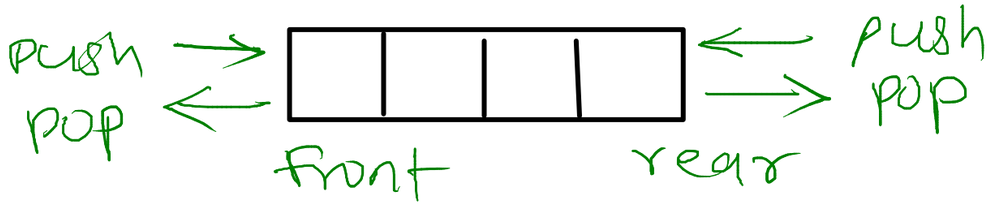
Stack(LIFO)

1. Add First
   Delete First
2. Add Last
   Delete Last

Queue(FIFO)

1. Add First
   Delete Last
2. Add Last
   Delete First

Deque(Double Ended Queue)



push front
add first

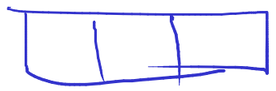
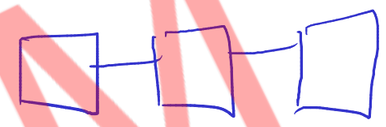
push rear
add last

pop front
delete first

pop rear
delet (last

Types of dequeue
1) input restricted dequeue
2) output restricted dequeue

Array Vs Linked List

Array	Linked List
1. Array space in memory is contiguous 	1. Linked list space in memory is not contiguous 
2. Array can not grow or shrink at runtime	2. Linked list can grow or shrink at runtime
3. Random access of elements is allowed	3. Random access of elements is not allowed(sequential)
4. Insert or Delete, needs shifting of array elements	4. Insert or Delete, do not need shifting of nodes
5. Array needs less space	5. Linked lists need more space