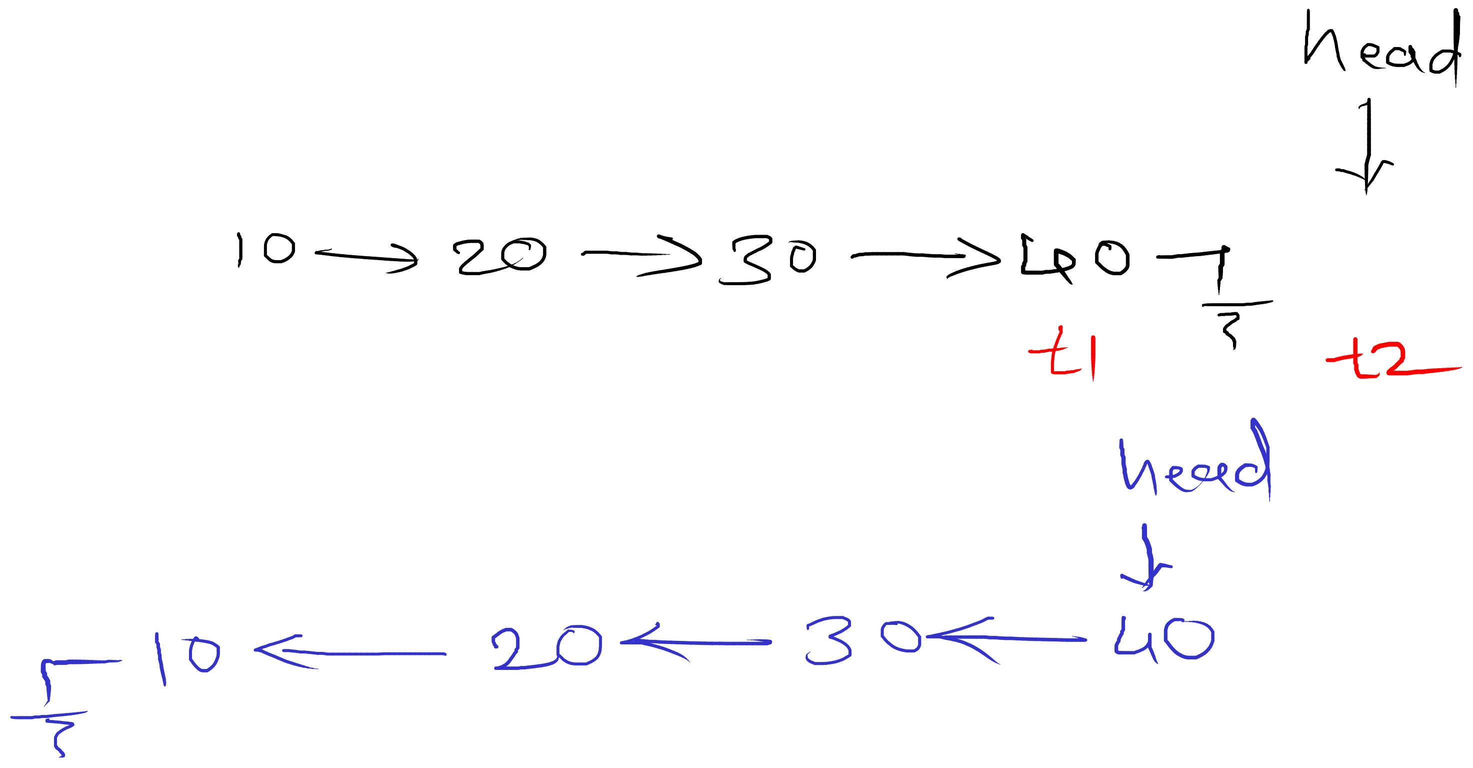
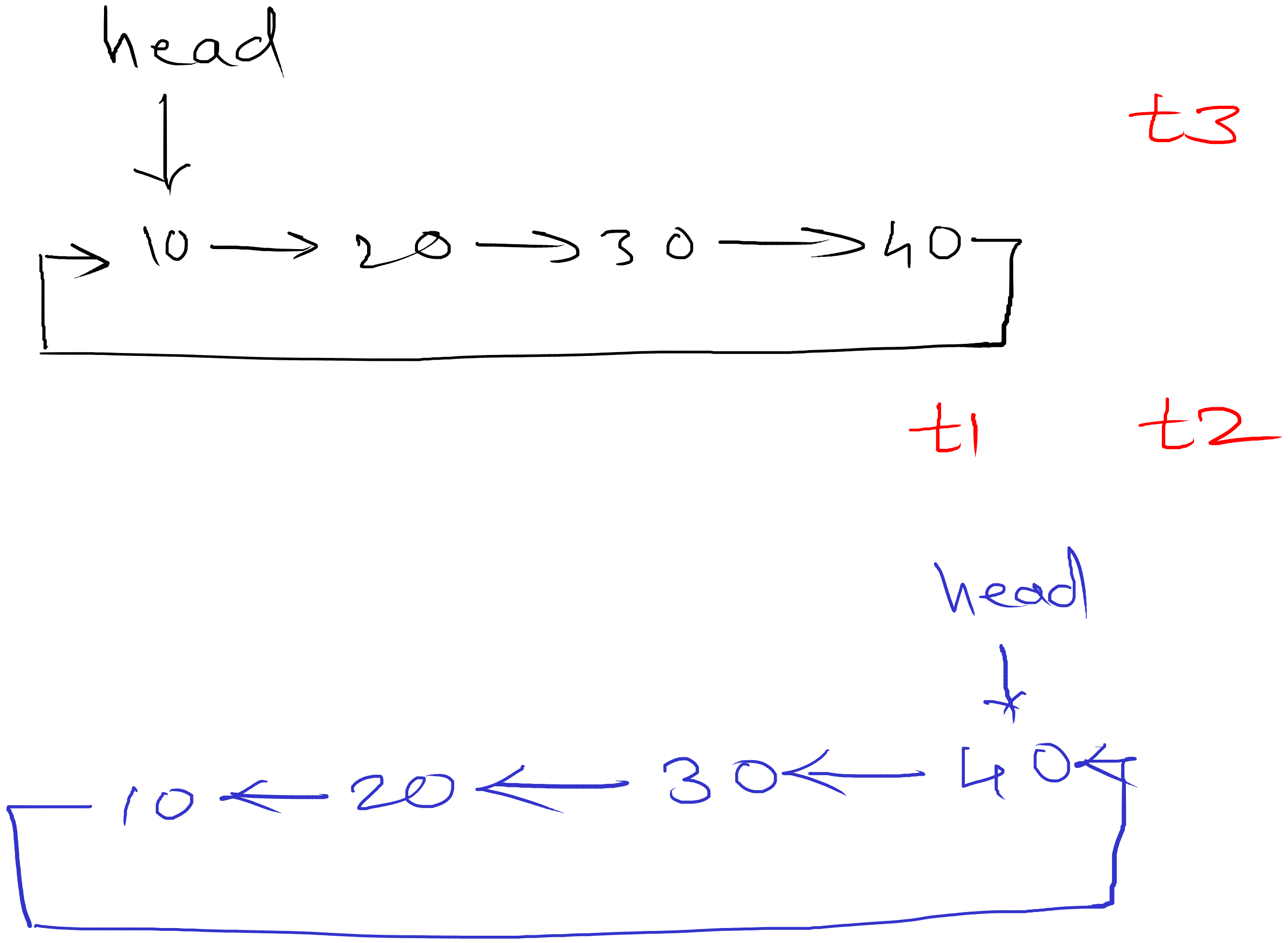
Singly Linear Linked List - Reverse List



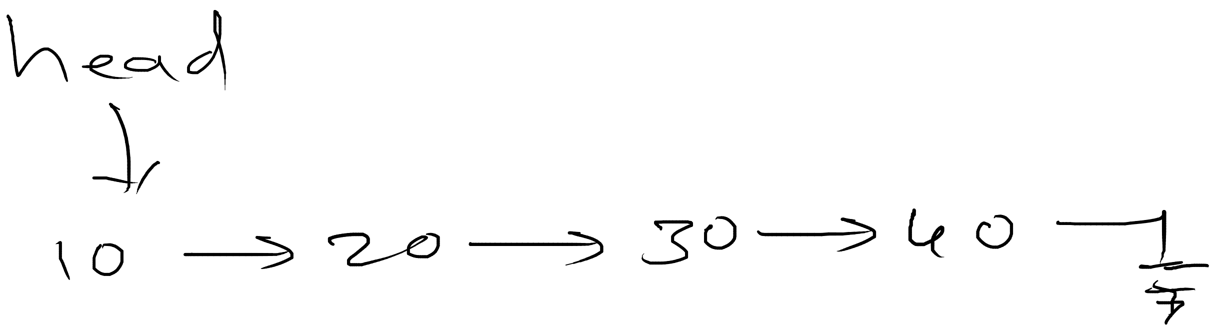
```
t1 = head;
t2 = head.next;
head.next = null;
while(head != null) {
        head = t2.next;
        t2.next = t1;
        t1 = t2;
        t2 = head
}
head = t1;
```

Singly Circular Linked List - Reverse List



```
t1 = head;
t2 = head.next;
while(t2 != head) {
      t3 = t2.next;
      t2.next = t1;
      t1 = t2;
      t2 = t3;
}
head.next = t1;
head = t1;
```

Singly Linear Linked List - Reverse Display



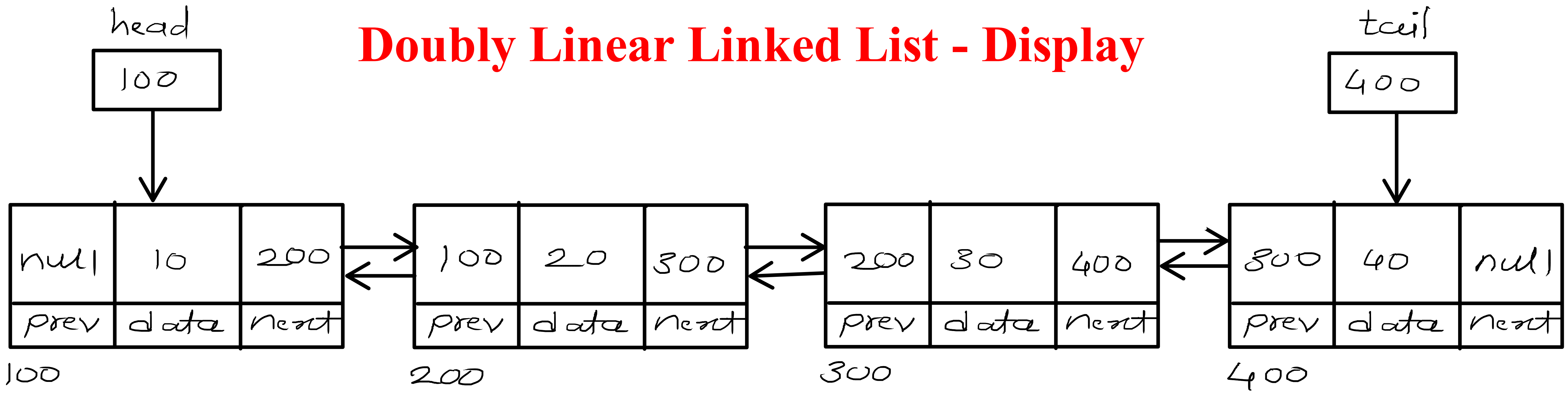
```
void  revDisplay(Node trav){
        if(trav == null)
            return;

        revDisplay(trav.next);
        sysout(trav.data);

}
```

Non-tail Recursion

```
void  forDisplay(Node trav){
        if(trav==null)
            return;
        sysout(trav.data);
        forDisplay(trav.next);
}
```

Tail Recursion

Doubly Linear Linked List - Display



```
//1. create trav and start at head
//2. visit current node
//3. go on next node
//4. repeat step 2 and 3 till last node
```

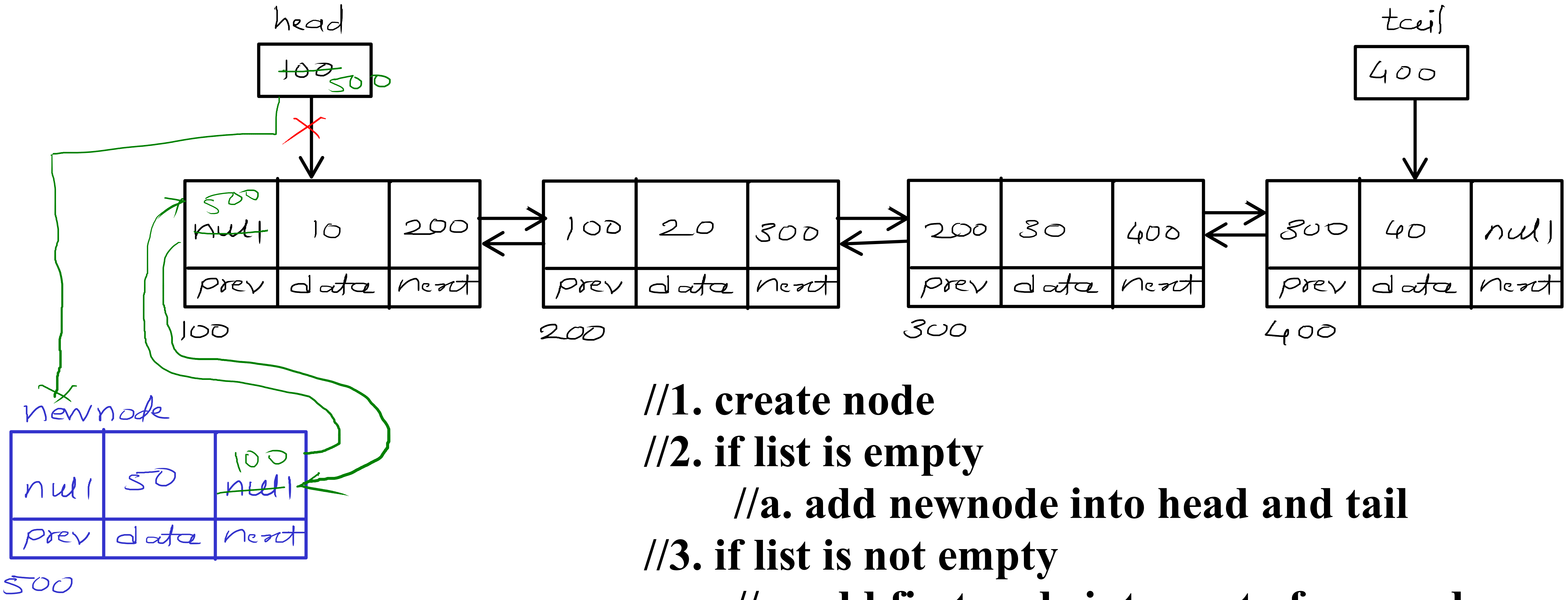
Forward

```
//1. create trav and start at tail
//2. visit current node
//3. go on prev node
//4. repeat step 2 and 3 till first node
```

Reverse

Time Complexity : O(n)

Doubly Linear Linked List - Add First



//1. create node

//2. if list is empty

    //a. add newnode into head and tail

//3. if list is not empty

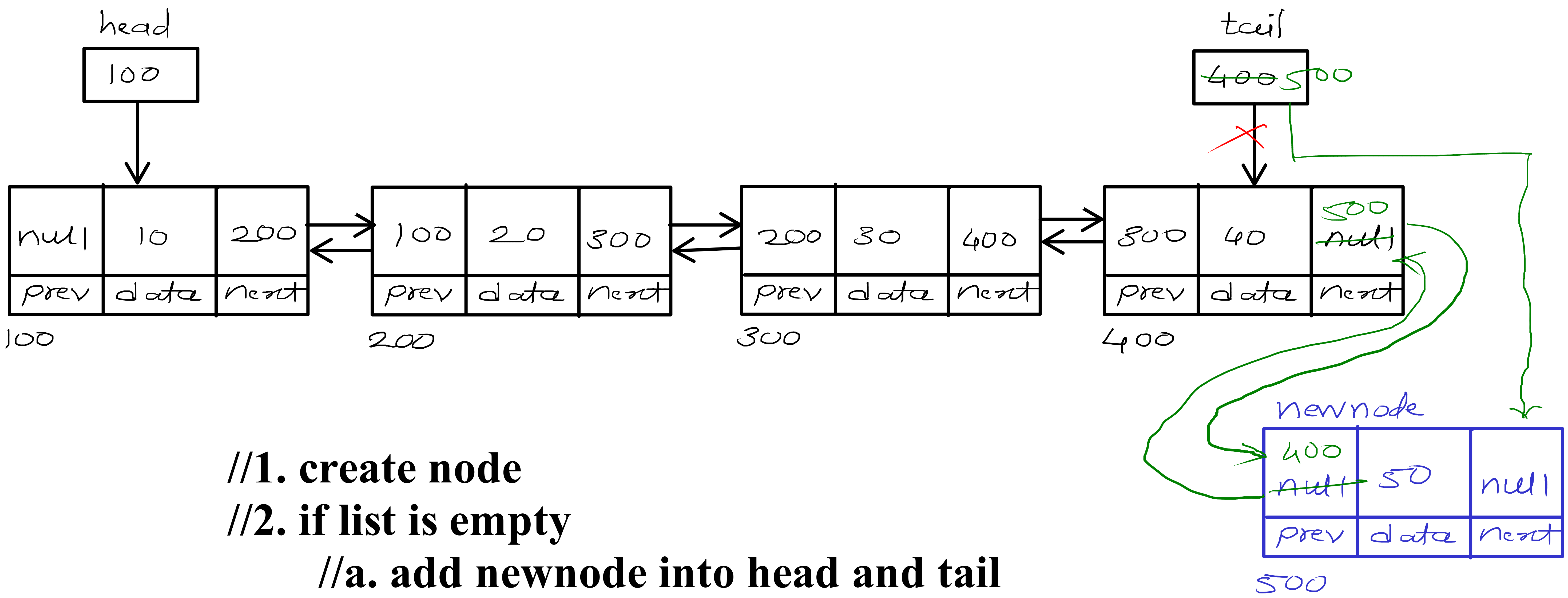
    //a. add first node into next of newnode

    //b. add newnode into prev of first node

    //c. move head on newnode

Time Complexity : O(1)

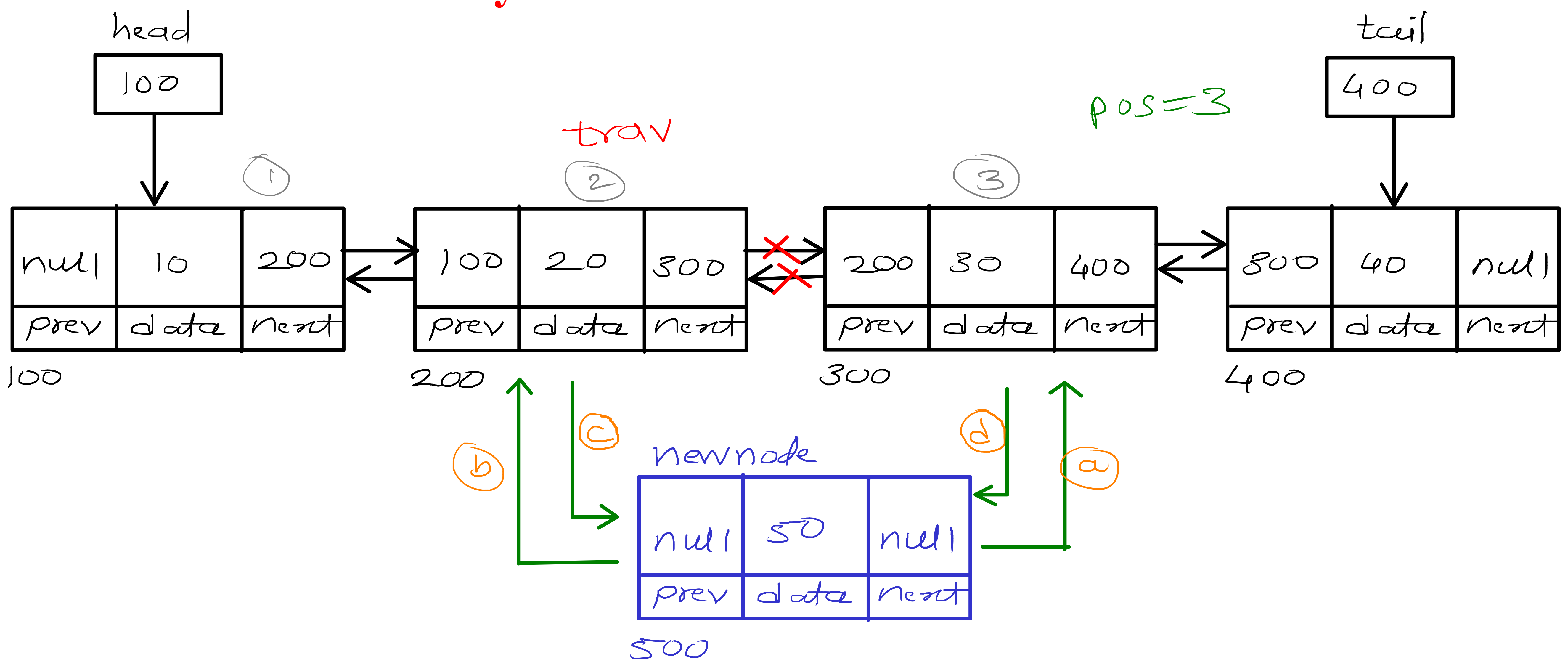
Doubly Linear Linked List - Add Last



```
//1. create node
//2. if list is empty
        //a. add newnode into head and tail
//3. if list is not empty
        //a. add last node into prev of newnode
        //b. add newnode into next of last node
        //c. move tail on newnode
```

Time Complexity : O(1)

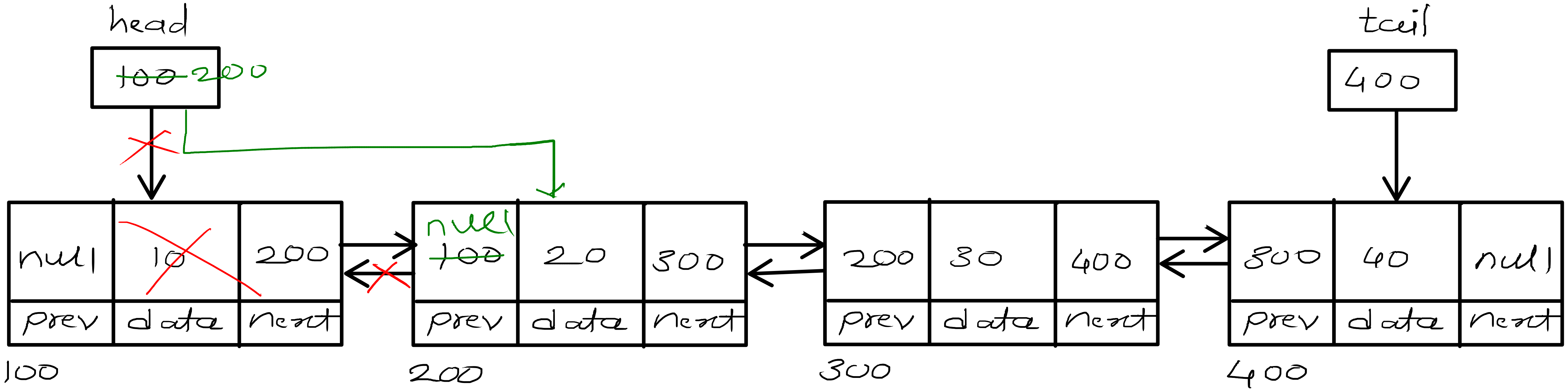
Doubly Linear Linked List - Add Position



```
//1. create node
//2. if list is empty
        //a. add newnode into head and tail
//3. if list is not empty
        //3.1 traverse till pos -1 node
        //a. add pos node into next of newnode
        //b. add pos-1 node into prev of newnode
        //c. add newnode into next of pos-1 node
        //d. add newnode into prev of pos node
```

Time Complexity : O(n)

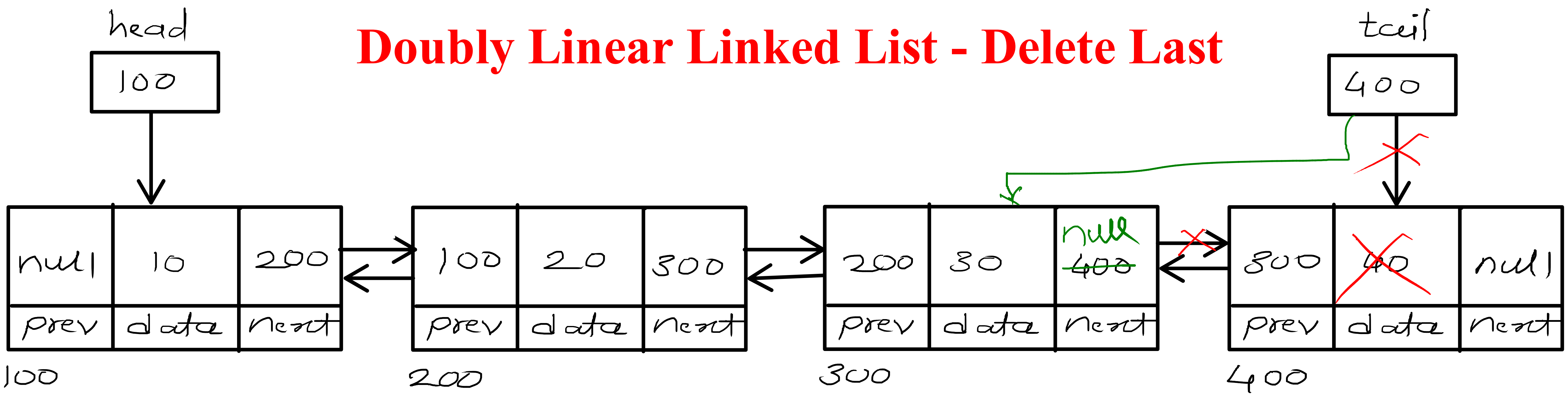
Doubly Linear Linked List - Delete First



```
//1. if list is empty
        // print msg
//2. if list has single node
        // head = tail = null
//3. if list has multiple node
        //a. move head on second node
        //b. add null into prev of second node
```

Time Complexity : O(1)

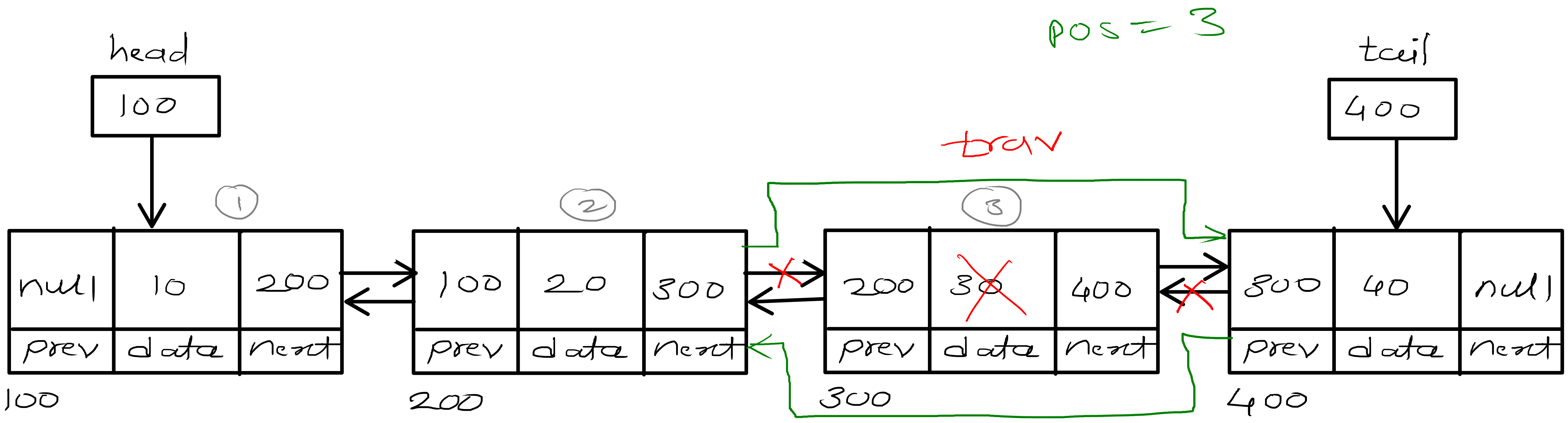
Doubly Linear Linked List - Delete Last



```
//1. if list is empty
        return;
//2. if list has single node
        head = tail = null;
//3. if list has multiple nodes
        //a. move tail on second last node
        //b. add null into next of second last node
```

Time Complexity : O(1)

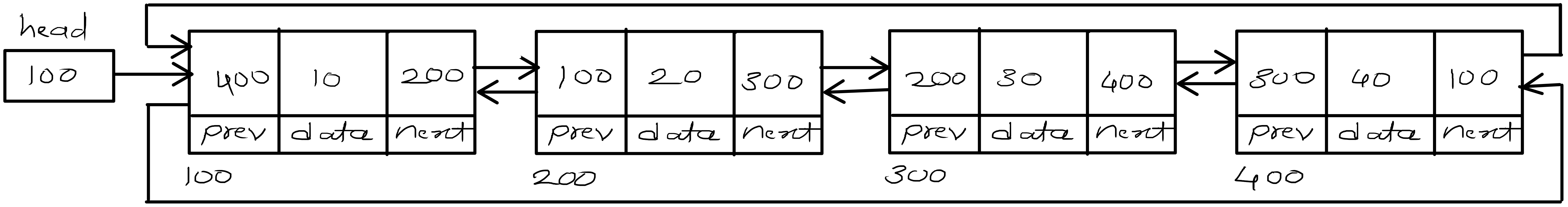
Doubly Linear Linked List - Delete Position



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

Time Complexity : O(n)

Doubly Circular Linked List - Display

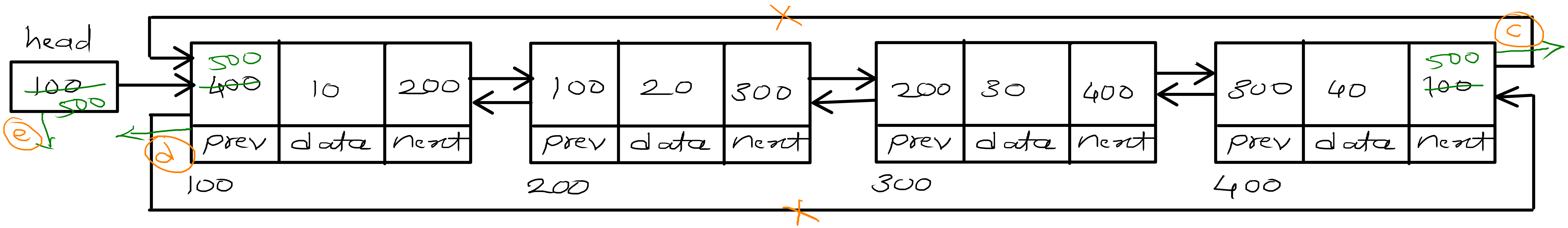


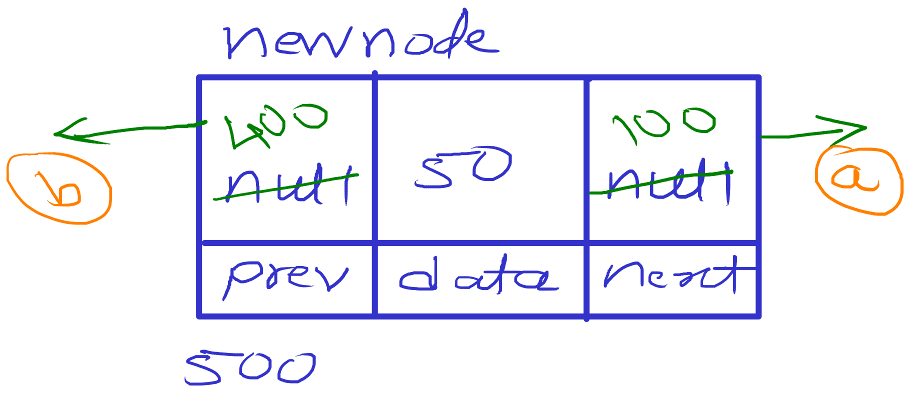
//1. create trav and start at head
//2. visit current node
//3. go on next node
//4. repeat step 2 and 3 till last node

//1. create trav and start at last node
//2. visit current node
//3. go on prev node
//4. repeat step 2 and 3 till first node

Time Complexity : O(n)

Doubly Circular Linked List - Add First

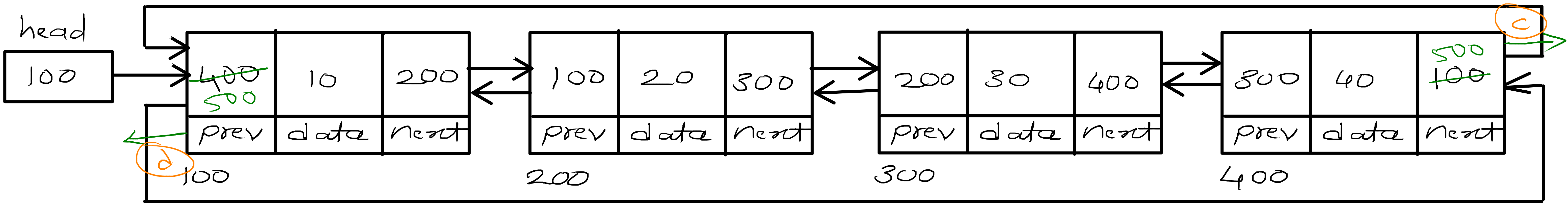


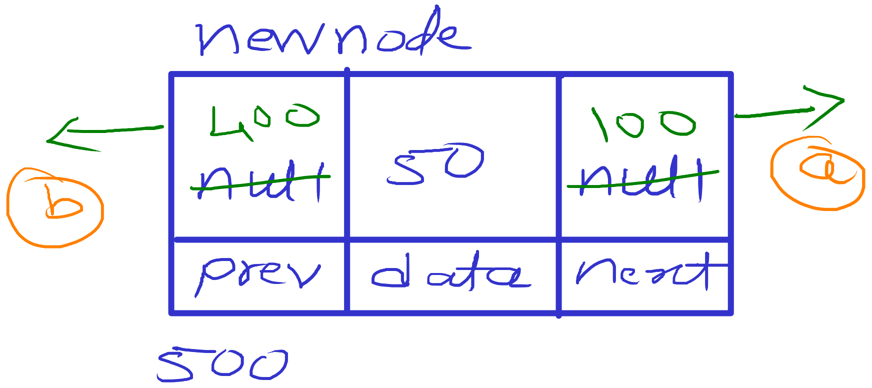


```
//1. create node
//2. if list is empty
        //a. add newnode into head
        //b. make list circular
//3. if list is not empty
        //a. add first node into next of newnode
        //b. add last node into prev of newnode
        //c. add newnode into next of last node
        //d. add newnode into prev of first node
        //e. move head on newnode
```

Time Complexity : O(1)

Doubly Circular Linked List - Add Last

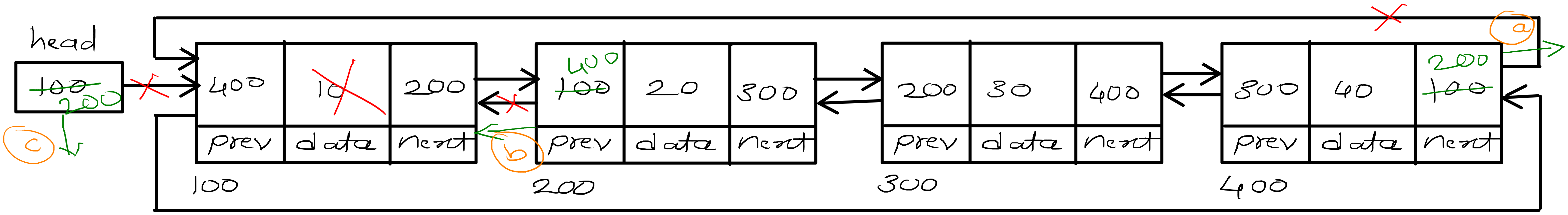




```
//1. create node
//2. if list is empty
        //a. add newnode into head
        //b. make list circular
//3. if list is not empty
        //a. add first node into next of newnode
        //b. add last node into prev of newnode
        //c. add newnode into next of last node
        //d. add newnode into prev of first node
```

Time Complexity : O(1)

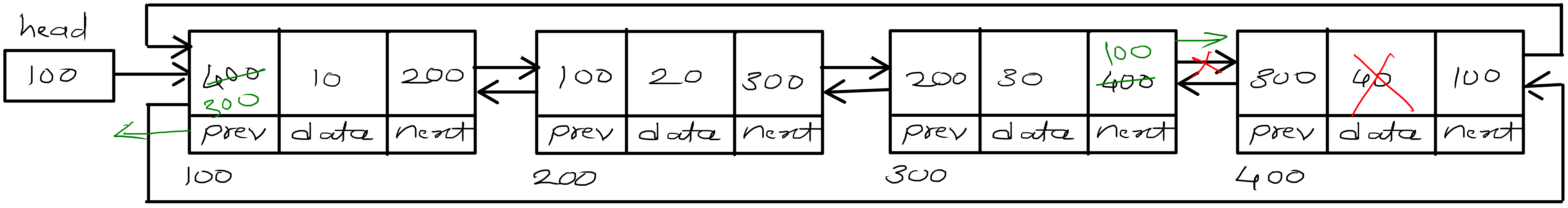
Doubly Circular Linked List - Delete First



```
//1. if list is empty
        return;
//2. if has single node
        head = null;
//3. if list has multiple nodes
        //a. add second node into next of last node
        //b. add last node into prev of second node
        //c. move head on second node
```

Time Complexity : O(1)

Doubly Circular Linked List - Delete Last



```
//1. if list is empty
        return;
//2. if list has single node
        head = null;
//3. if list has multiple nodes
        //a. add first node into next of second last node
        head.prev.prev.next = head;
        //b. add second last node into prev of first node
        head.prev = head.prev.prev;
```

Time Complexity : O(1)

Linked List Applications

- dynamic data structure (grow or shrink at runtime)
- due to this dynamic nature, it is used to implement other data strutcures like
    - stack
    - queue
    - hash table (seperate chaining)
    - graph (adjacency list)

Stack

(push/pop)

1.

Add First()
Delete First()

2.

Add Last()
Delete Last()

Queue

(push/pop)

1.

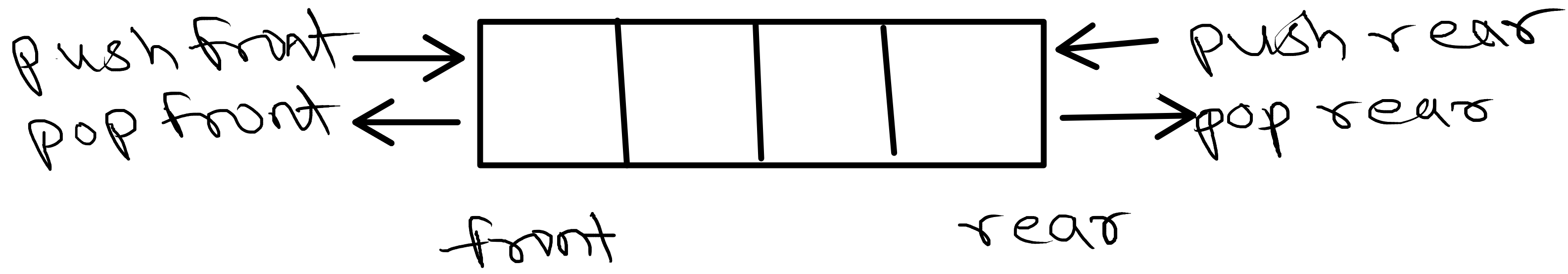
Add First()
Delete Last()

2.

Add Last()
Delete First()

Deque

(Double Ended Queue)



Types:

1. Input Restricted deque
    - insert/push is allowed from only one end
2. Output Restricted deque
    - remove/pop is alloed from only one end

Array Vs Linked List

Array

1. Array space in memory is contiguous

2. Array can not grow or shrink at runtime

3. Random access of elements is allowed

4. Insert or Delete, needs shifting of array elements

5. Array needs less space

Linked List

1. Linked list space in memory is not contiguous

2. Linked list can grow or shrink at runtime

3. Random access of elements is not allowed(sequential)

4. Insert or Delete, do not need shifting of nodes

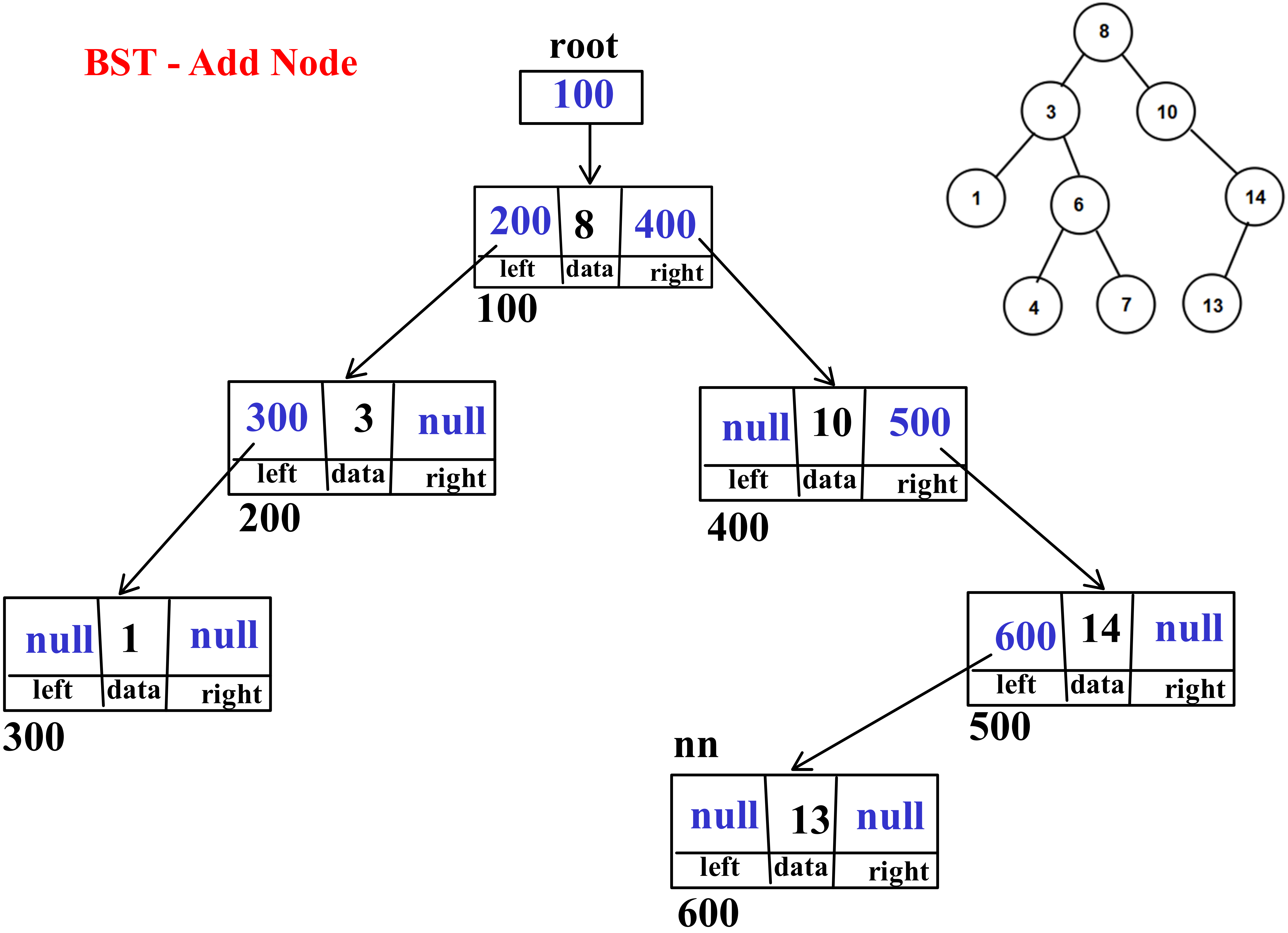
5. Linked lists need more space

BST - Add Node

```
//1. create node with given data
//2. if tree is empty
    //a. add newnode into root itself
//3. if tree is not empty
    //3.1 create trav and start at root
    //3.2 if value is less than current node data
        //3.2.1 if current node left is empty
            //add value(node) in left of current node
        //3.2.2 if current node left is not empty
            // go into left
    //3.3 if value is greater than current node data
        //3.3.1 if current node right is empty
            //add value(node) in right of current node
        //3.2.2 if current node right is not empty
            // go into right
    //3.4 repeat step 3.2 and 3.3 till node is not added
```

Time Complexity : O(h) / O(log n)

BST - Add Node


root
100
200
8
400
left
data
right
100
300
3
null
left
data
right
200
null
10
500
left
data
right
400
null
1
null
left
data
right
300
600
14
null
left
data
right
500
nn
null
13
null
left
data
right
600
8
3
10
1
6
14
4
7
13