Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

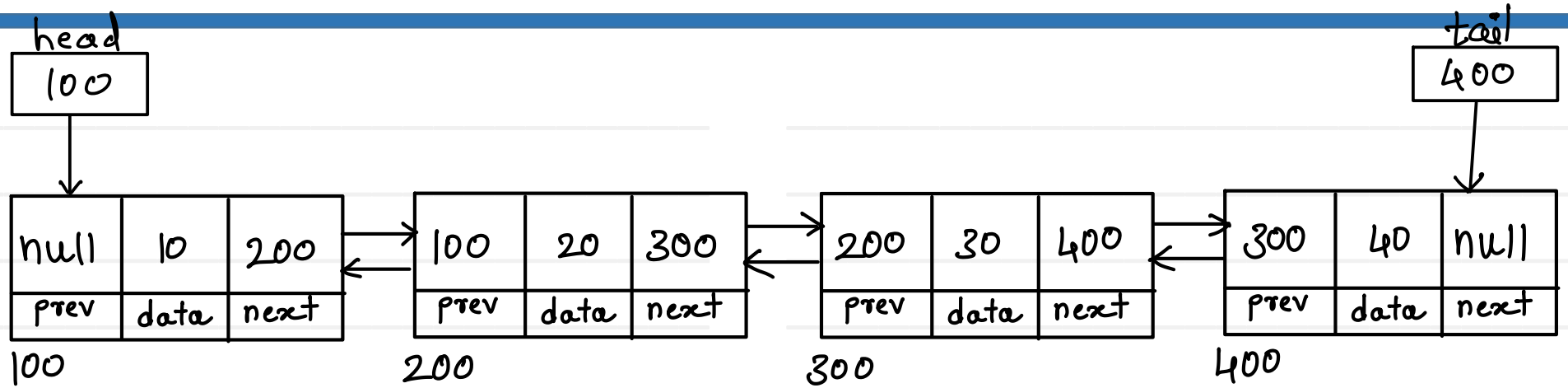
Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Array Vs Linked list

Array	Linked list
• Array space inside memory is continuous	• Linked list space inside memory is not continuous
• Array can not grow or shrink at runtime	• Linked list can grow or shrink at runtime
• Random access of elements is allowed	• Random access of elements is not allowed
• Insert or delete, needs shifting of array elements	• Insert or delete, need not shifting of linked list elements
• Array needs less space	• Linked list needs more space

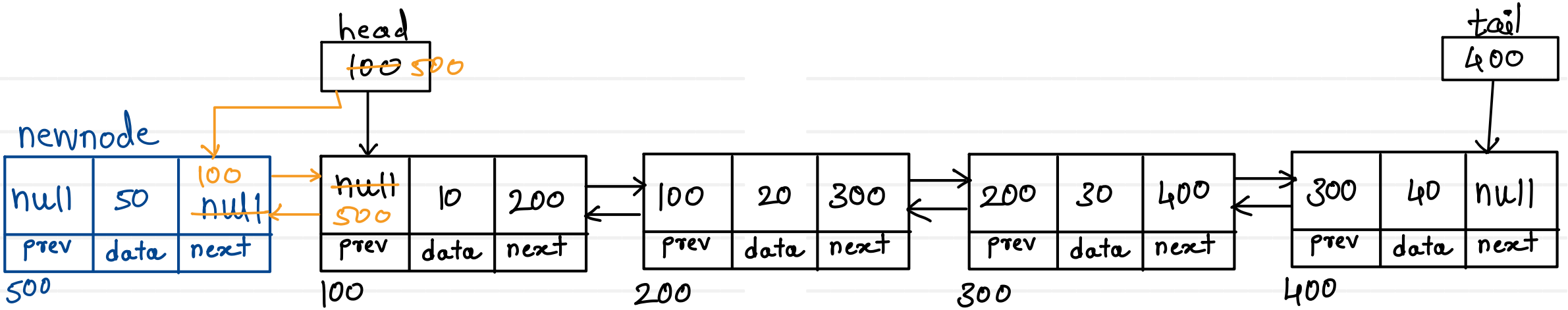
Doubly Linear Linked List - Display

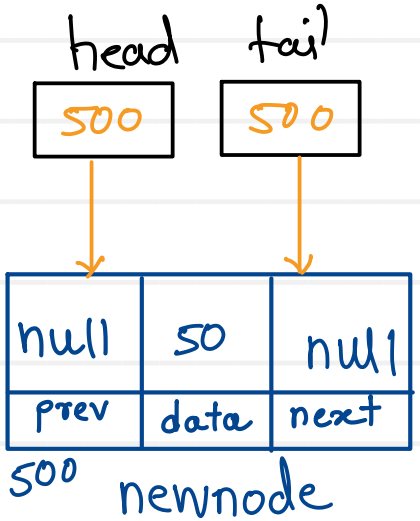


1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

1. Create trav & start at last node
2. print current node
3. go on prev node
4. repeat step 2 & 3 till first node

Doubly Linear Linked List - Add first



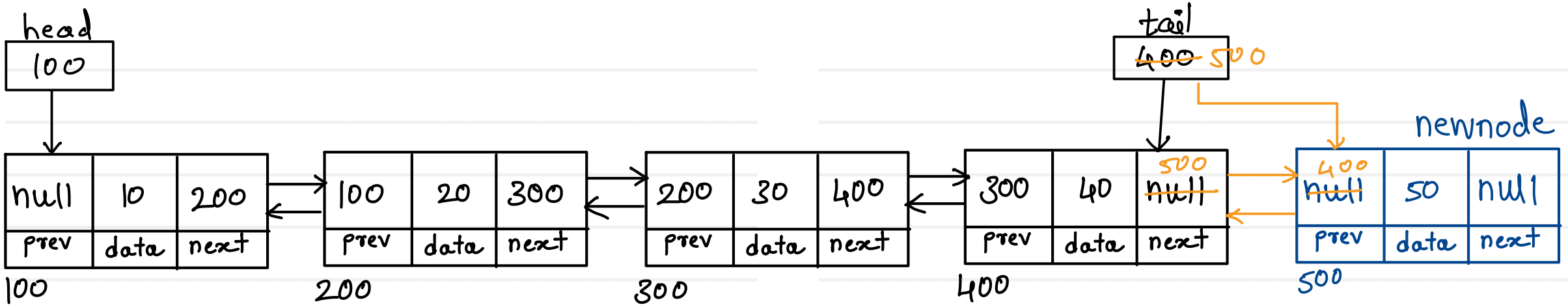


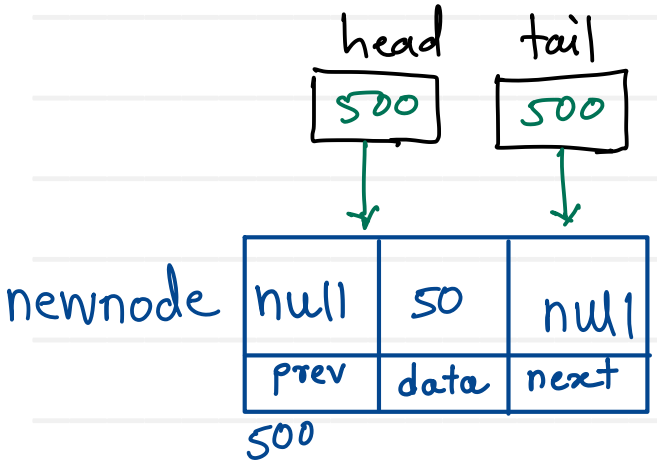
if list is empty,
then add newnode into head & tail

if list is not empty,
a. add first node into next of newnode
b. add newnode into prev of first
c. move head on newnode

$T(n) = O(1)$

Doubly Linear Linked List - Add last



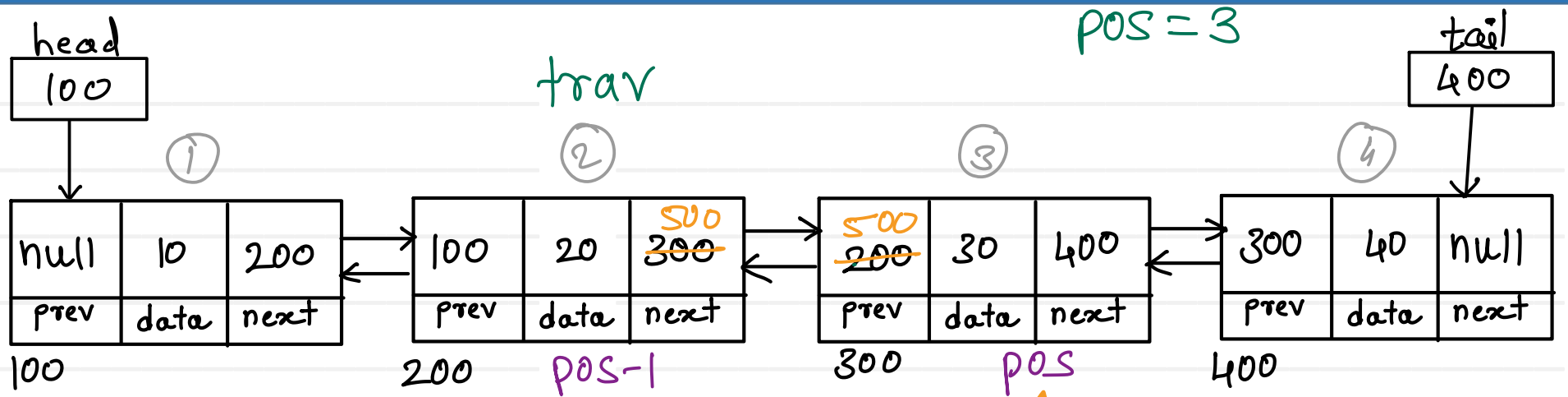


if list is empty,
then add newnode
into head & tail

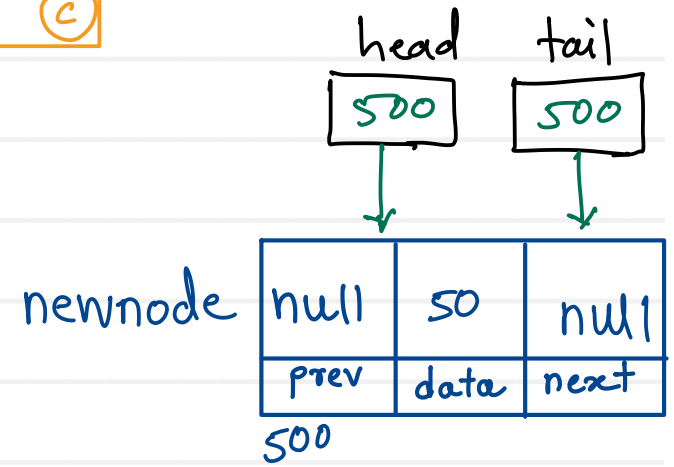
1. add last node into prev of newnode
2. add newnode into next of last node
3. move tail on newnode

$T(n) = O(1)$

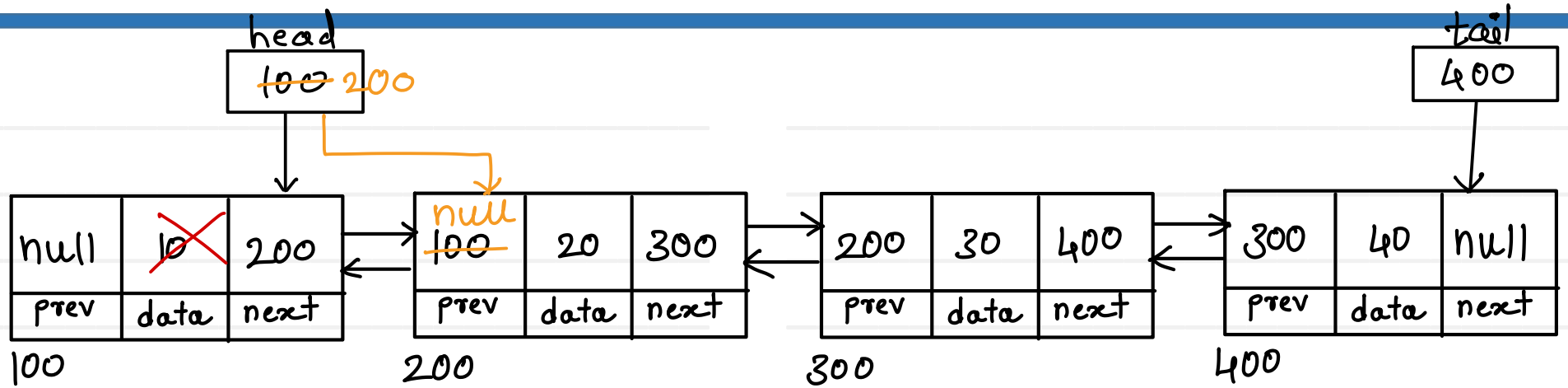
Doubly Linear Linked List - Add position

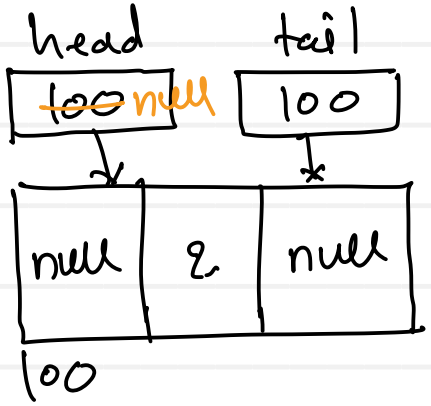


traverse till pos-1 node
a. add pos node into next of newnode
b. add pos-1 node into prev of newnode
c. add newnode into prev of pos node
d. add newnode into next of pos-1 node



Doubly Linear Linked List - Delete first



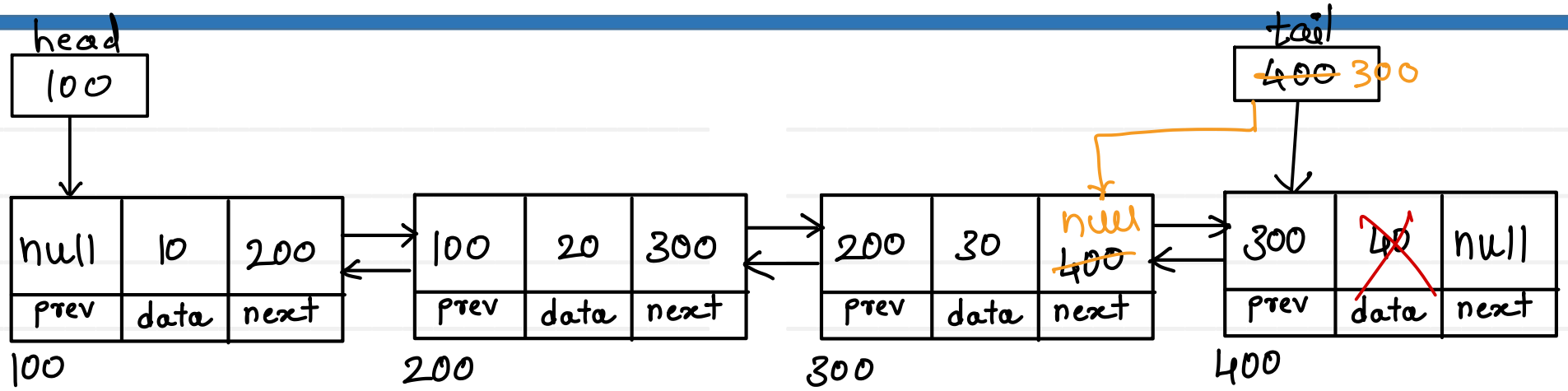


head = head.next

head.prev = null

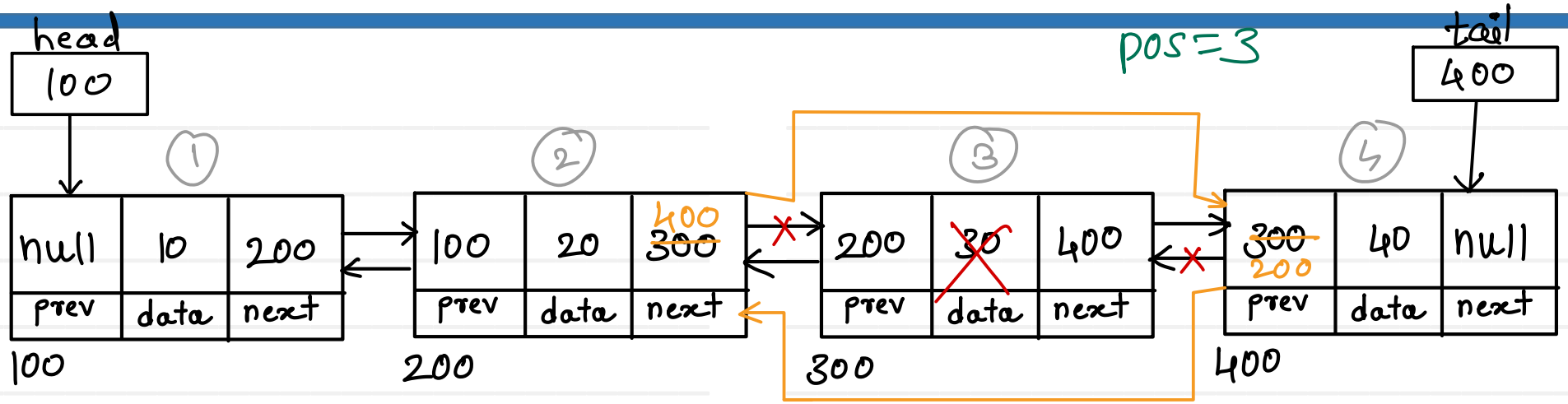
1. if list is empty, return
2. if list has single node, delete it
   head = tail = null
3. if list has multiple nodes,
   a. move head on second node
   b. add null into prev of second node

Doubly Linear Linked List - Delete last



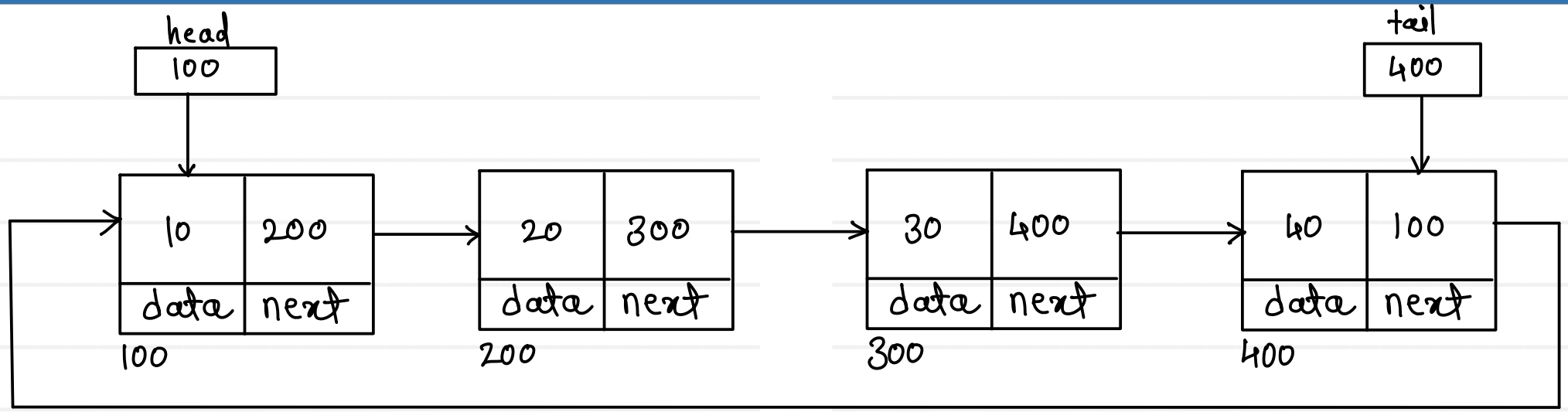
1. if list is empty, return
2. if list has single node, delete it
   head = tail = null
3. if list has multiple nodes,
   a. move tail on second last node
   b. add null into next of second last node

Doubly Linear Linked List - Delete Position



1. if list is empty, return trav.prev
2. if list has single node, delete it
   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

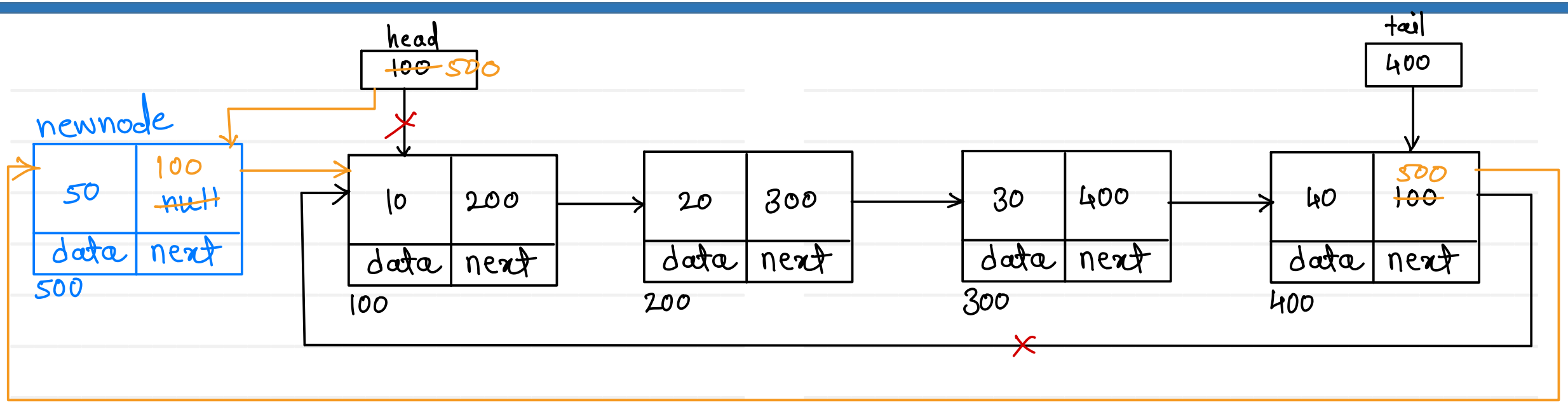
Singly Circular Linked List - Display

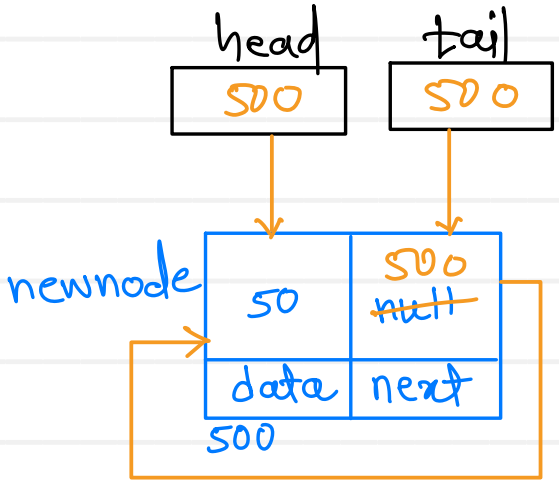


1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeate above 2 steps for each node

```
Node trav = head;
do {
    sysout(trav.data);
    trav = trav.next;
} while (trav != head );
```

Singly Circular Linked List - Add first



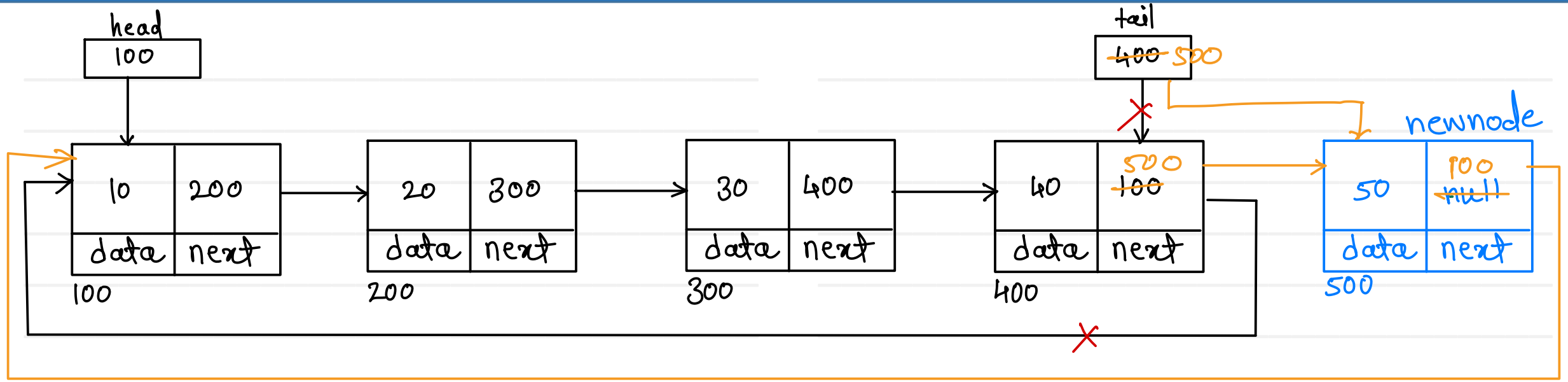


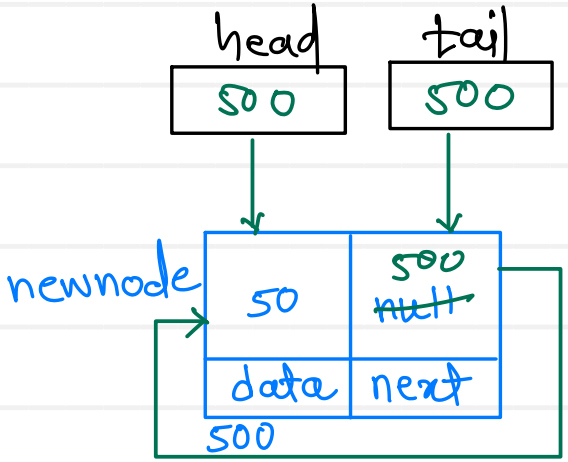
if list is empty,
1. add newnode into head & tail
2. make list circular

if list is not empty,
a. add first node into next of newnode
b. add newnode into next of last node
c. move head on newnode

$T(n) = O(1)$

Singly Circular Linked List - Add last



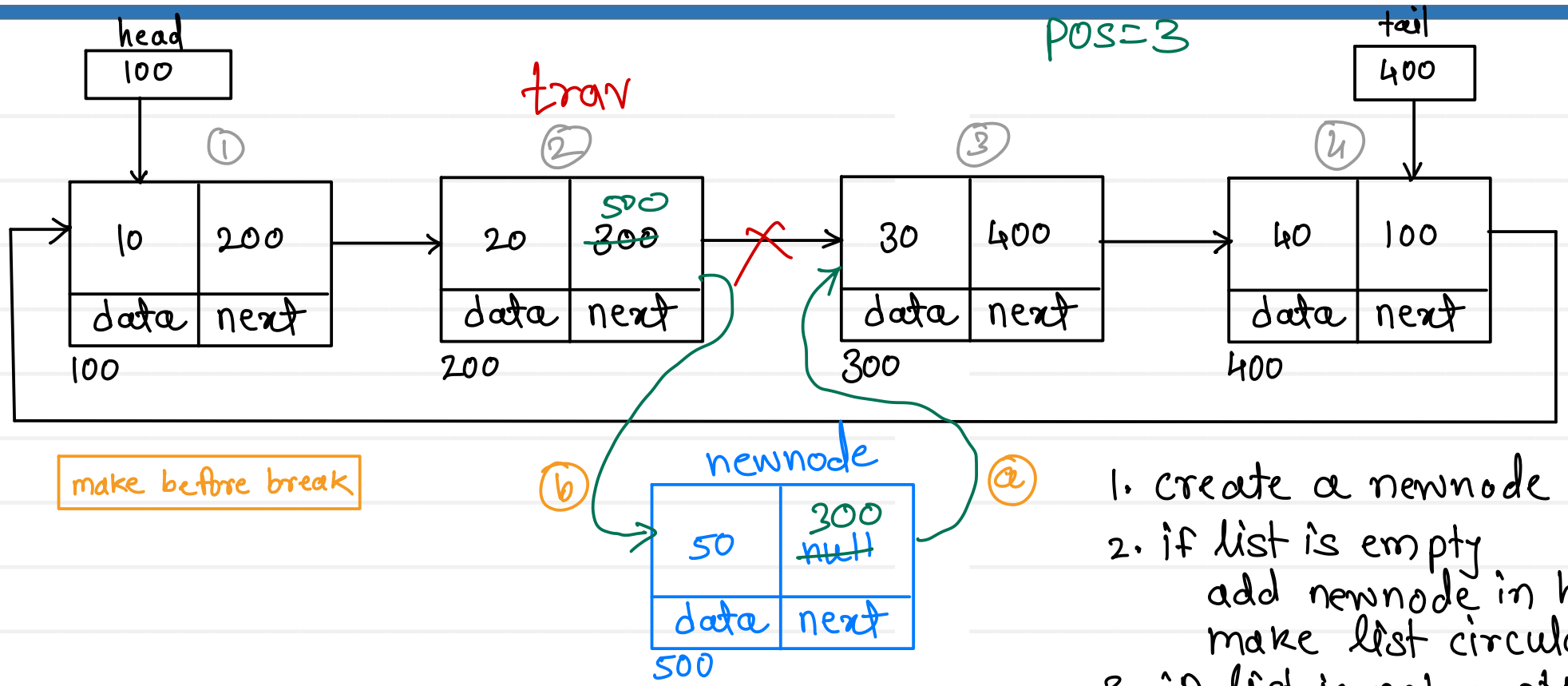


if list is empty,
1. add newnode into head & tail
2. make list circular

if list is not empty,
a. add first node into next of newnode
b. add newnode into next of last node
c. move tail on newnode

$T(n) = O(1)$

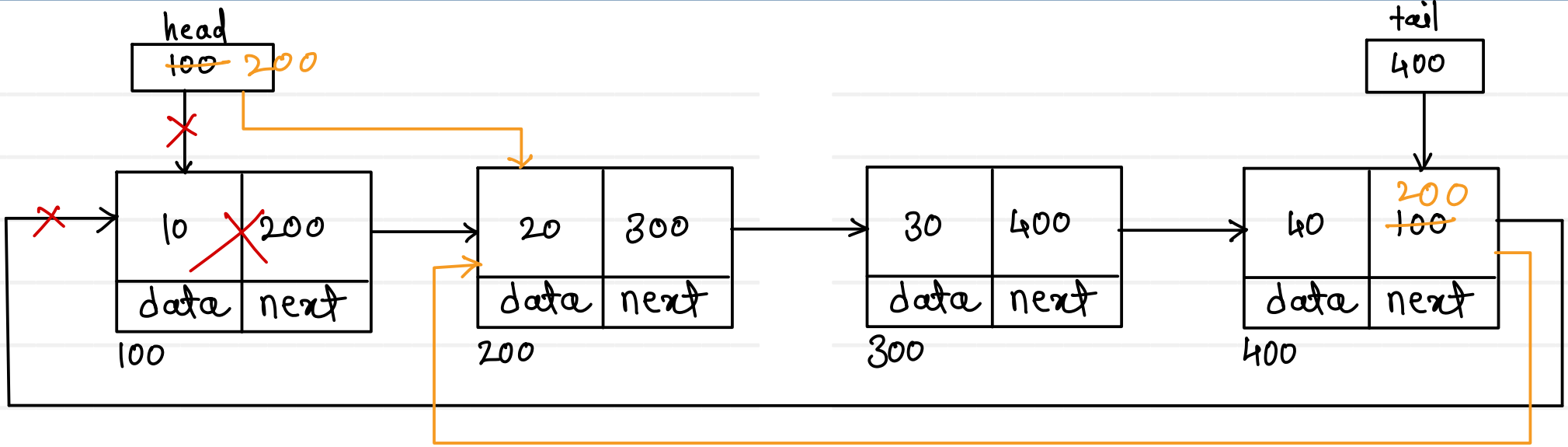
Singly Circular Linked List - Add position

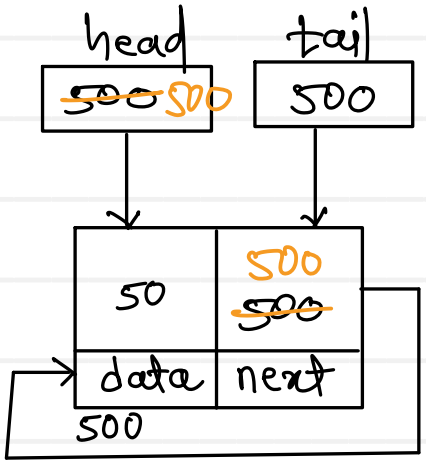


1. create a newnode
2. if list is empty
   add newnode in head & tail.
   make list circular.
3. if list is not empty
   a. traverse till pos-1 node
   b. add pos node into next of newnode
   c. add newnode into next of pos-1 node

$T(n) = O(n)$

Singly Circular Linked List - Delete first



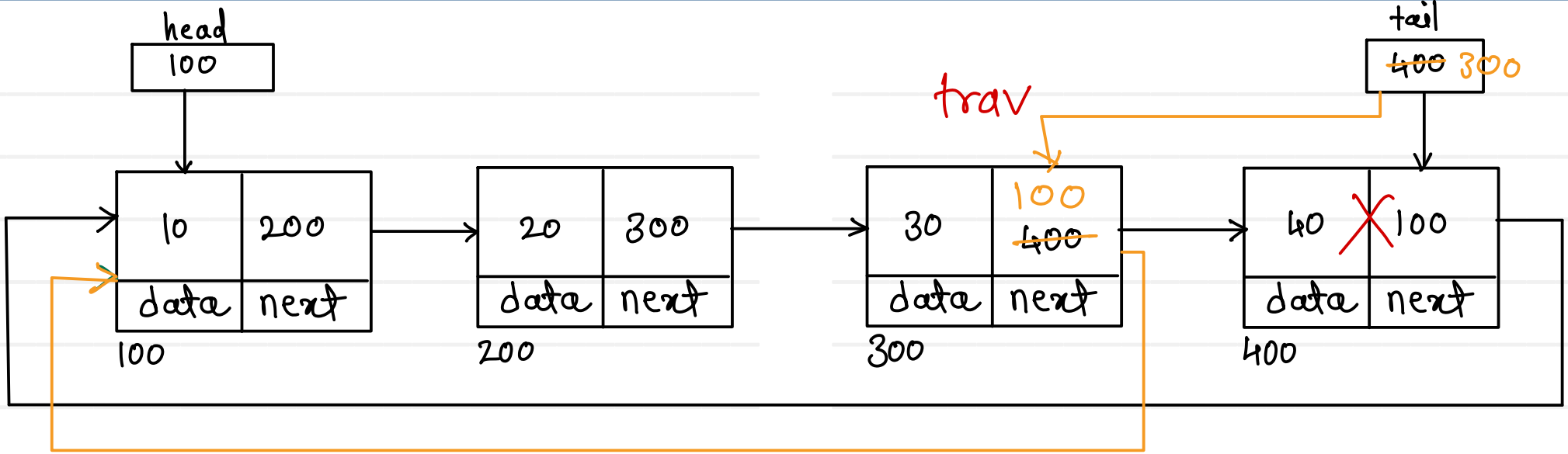


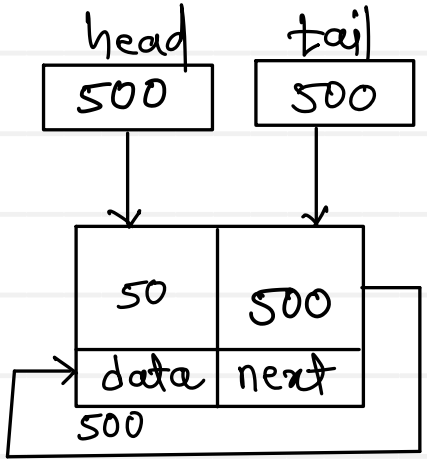
tail.next = head.next
head = head.next

1. if empty, return
2. if single node, delete it
3. if multiple nodes
   a. add second node into next of last node
   b. move head on second node

$T(n) = O(1)$

Singly Circular Linked List - Delete last

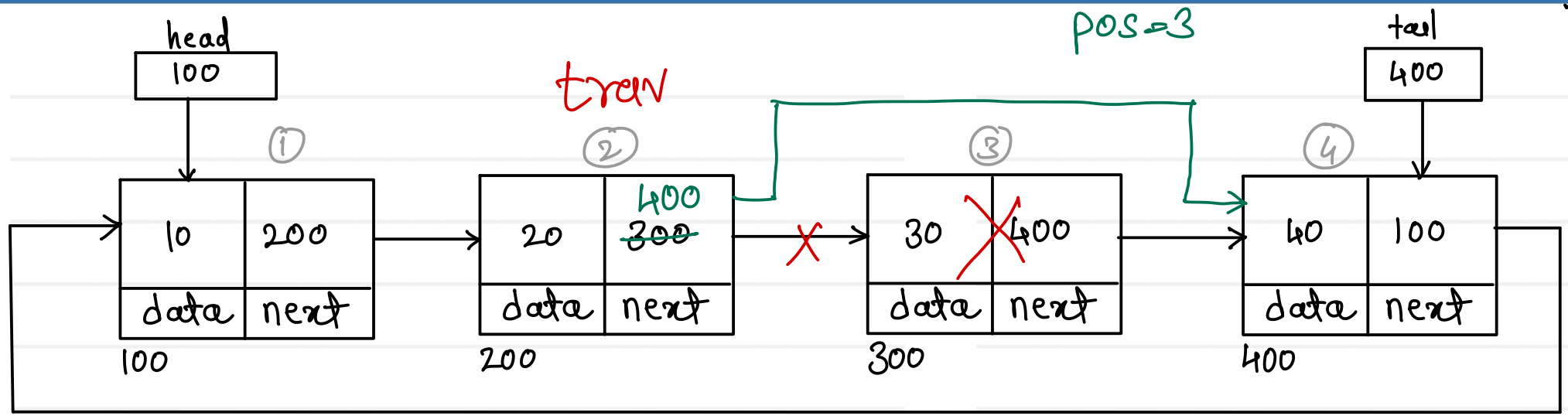




$T(n) = O(n)$

1. if list is empty, return
2. if list has single node, delete it.
3. if list has multiple nodes
   a. traverse till second last node
   b. add first into next of second last node
   c. move tail on second last node

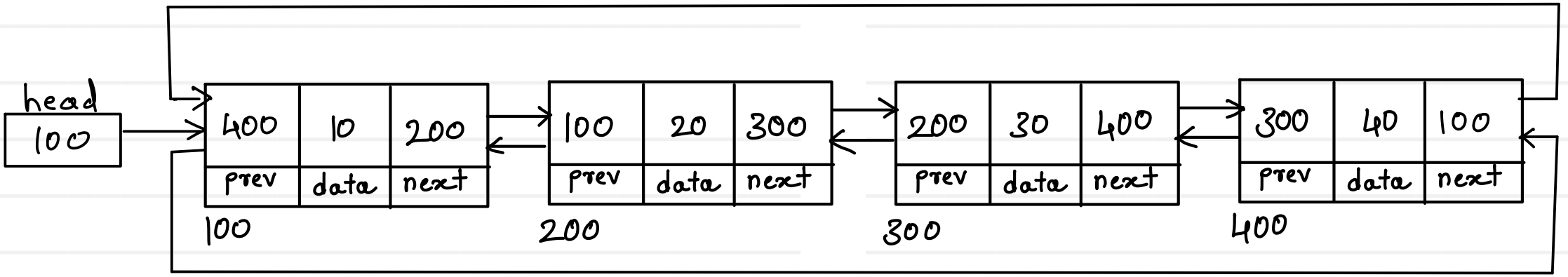
Singly Circular Linked List - Delete position



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

$T(n) = O(n)$

Doubly Circular Linked List - Display

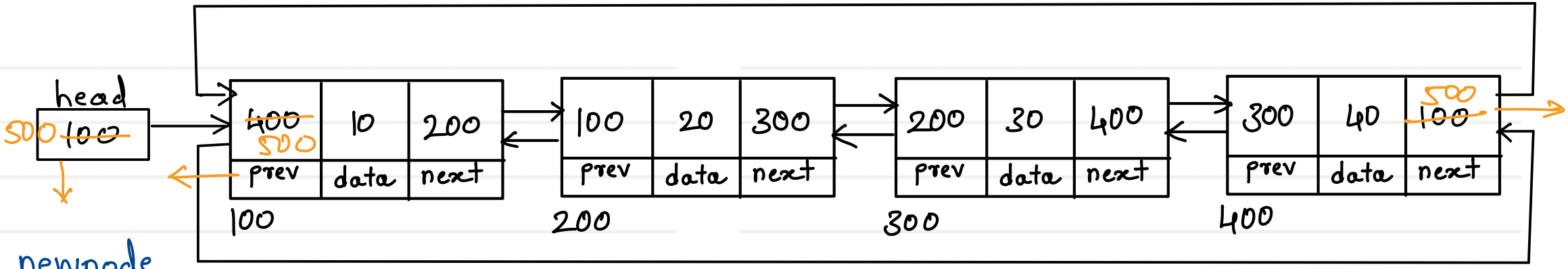


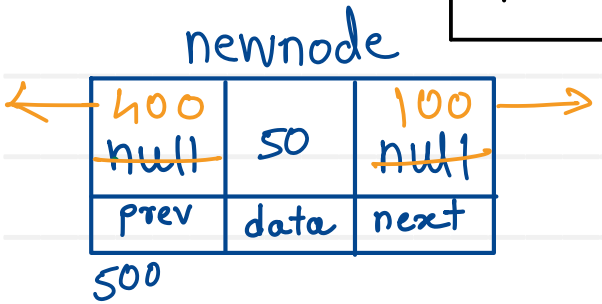
1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

1. Create trav & start at last node
2. print current node
3. go on prev node
4. repeat step 2 & 3 till first node

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

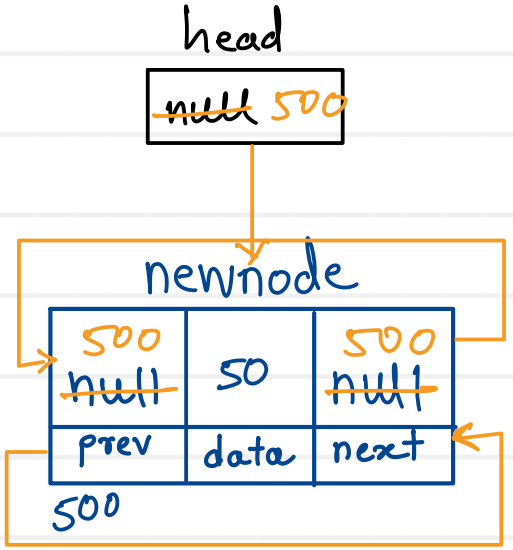
Doubly Circular Linked List - Add first





if not empty,

1. add first node into next of newnode
2. add last node into prev of newnode
3. add newnode into next of last node
4. add newnode into prev of first node
5. move head on newnode

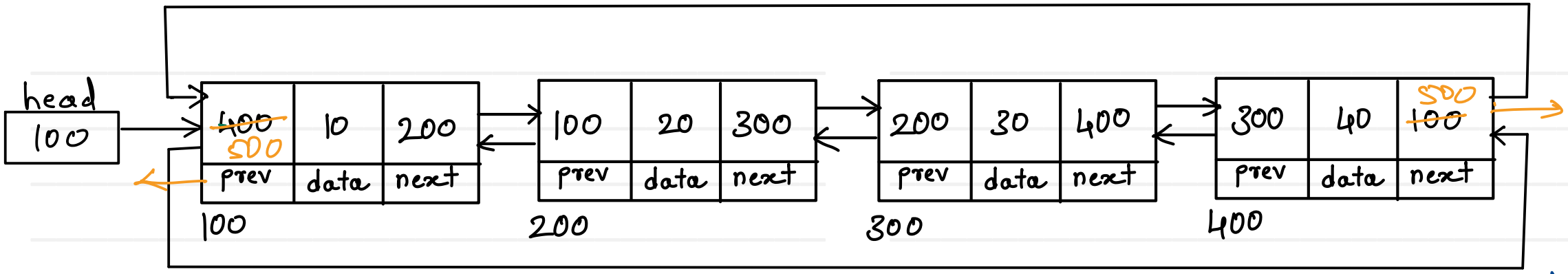


$T(n) = O(1)$

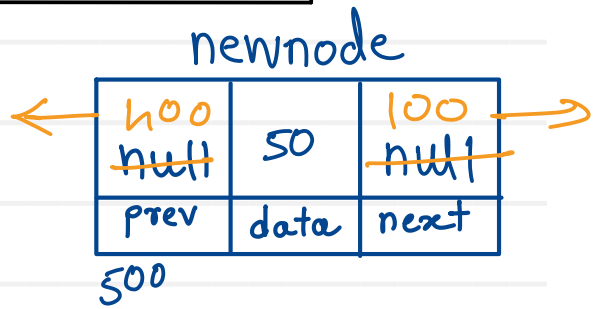
if list empty,

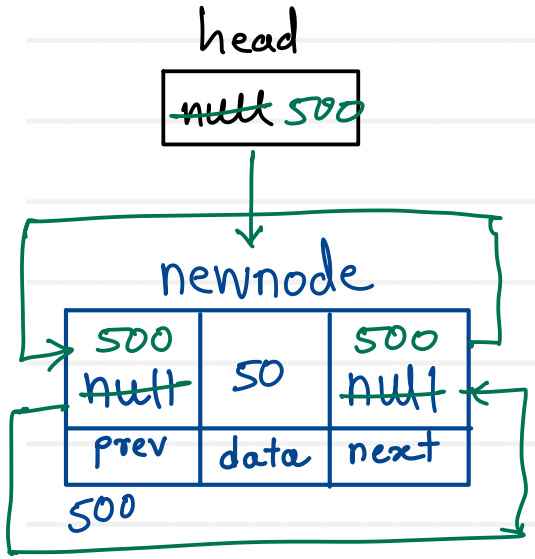
a. add newnode into head
b. make list circular

Doubly Circular Linked List - Add last



$T(n) = O(1)$

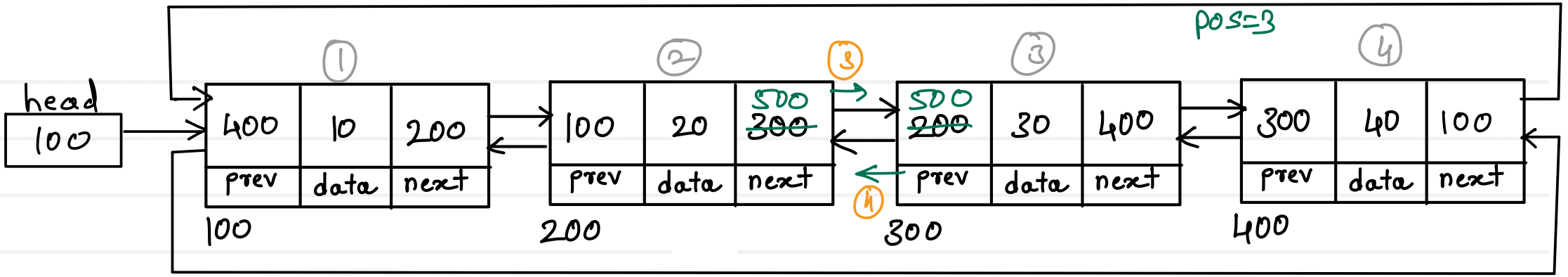




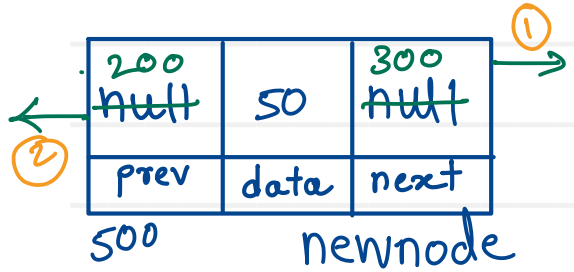
if list empty,
a. add newnode into head
b. make list circular

if not empty,
1. add first node into next of newnode
2. add last node into prev of newnode
3. add newnode into next of last node
4. add newnode into prev of first node

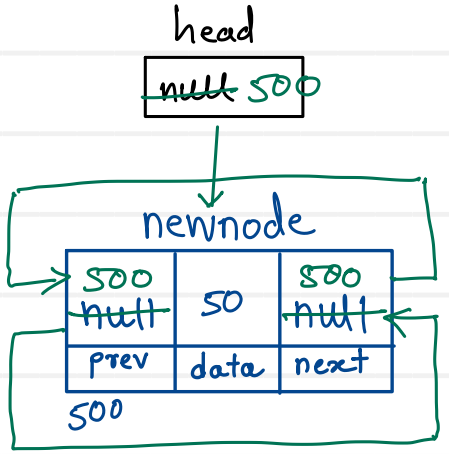
Doubly Circular Linked List - Add position



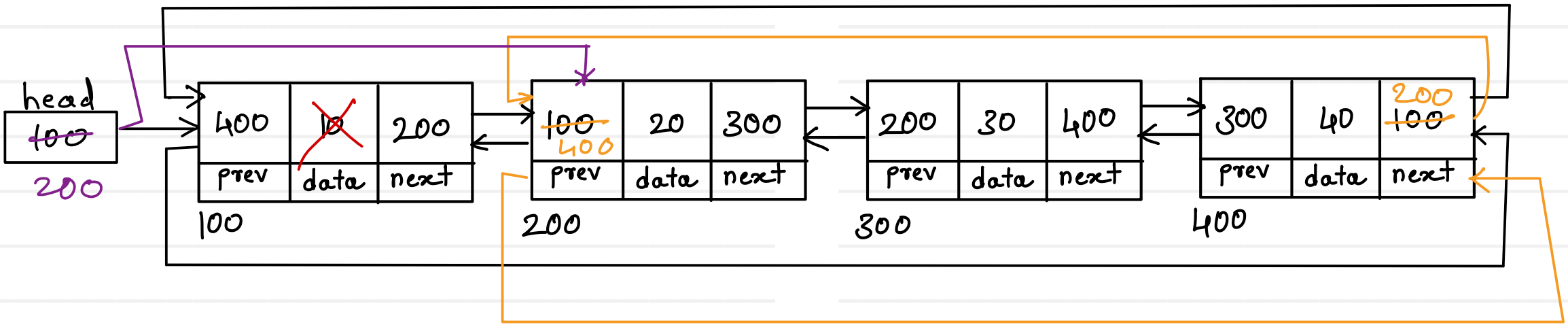
1. create node
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty.
   a. traverse till pos-1 node
   b. add pos node into next of newnode
   c. add pos-1 node into prev of newnode
   d. add newnode into next of pos-1 node
   e. add newnode into prev of pos node



$T(n) = O(1)$



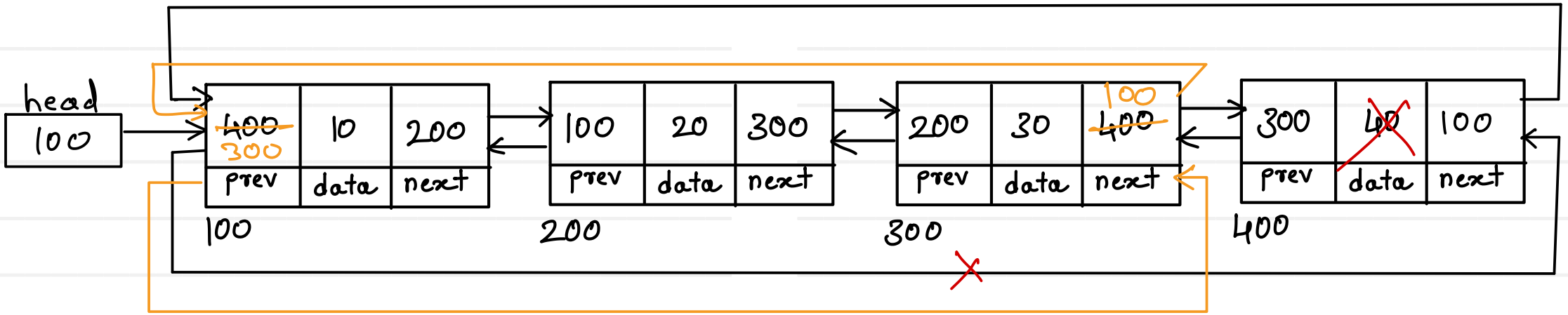
Doubly Circular Linked List - Delete first



1. if list is empty, return
2. if list has single node, delete it
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

$T(n) = O(1)$

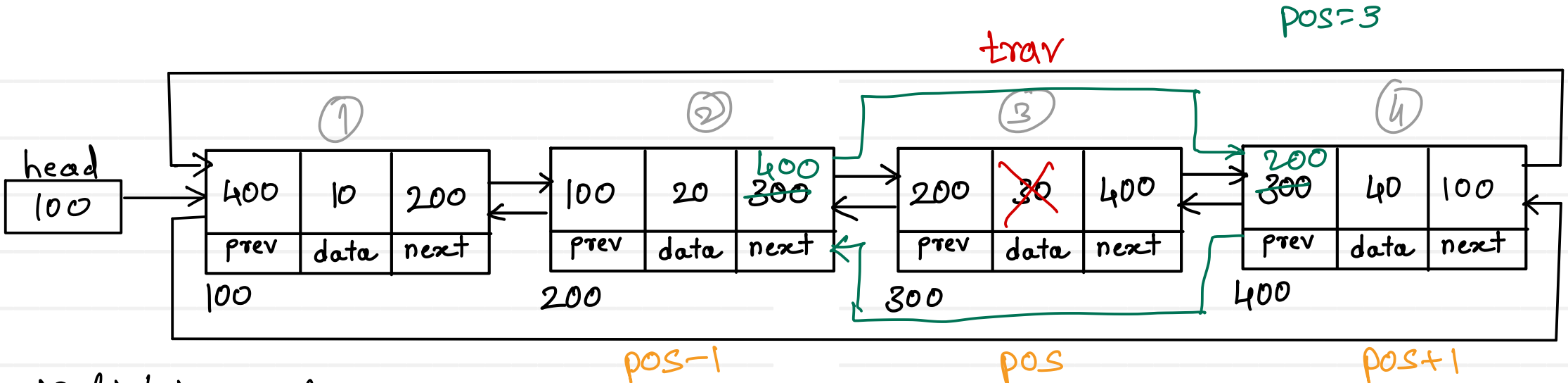
Doubly Circular Linked List - Delete last



1. if list is empty, return
2. if list has single node, delete it
3. if list has multiple nodes
   a. add second last node into prev of first node
   b. add first node into next of second last node

$T(n) = O(1)$

Doubly Circular Linked List - Delete position



1. if list is empty
   return;
2. if list has single node
   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

trav → pos
trav.prev → pos-1
trav.next → pos+1

$T(n) = O(n)$

Iterative approach

- loops are used to implement algorithms

Time $\propto$ No. of iterations of the loop

```
int factorial ( int n) {
    int fact = 1
    for( i=1; i<=n; i++)
        fact *= i;
    return fact;
}
```

No. of iterations = n

Time $\propto$ n

$T(n) = O(n)$

Extra space = i, fact

$S(n) = O(1)$

Recursive approach

- recursion is used to implement algorithms

Time $\propto$ No. of recursive calls of a function

```
int factorial ( int n) {
    if( n==1)
        return 1;
    return n * factorial (n-1);
}
```

No. of recursive calls = n

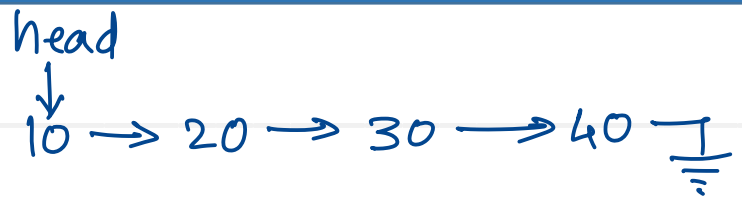
Time $\propto$ n

$T(n) = O(n)$

n-FARs will be created on stack

$S(n) = O(n)$

Singly linear linked list - Display reverse

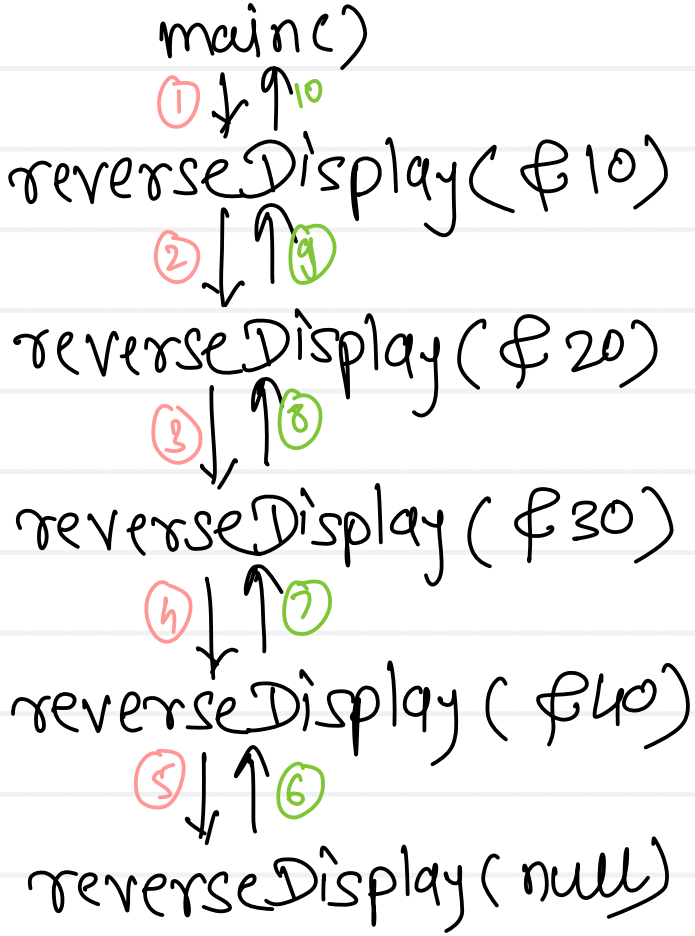


Output : 40 30 20 10

```
void reverseDisplay(Node curr)
    if(curr == null)
        return;
    reverseDisplay(curr.next);
    sysout(curr.data);
}
```

$T(n) = O(n)$

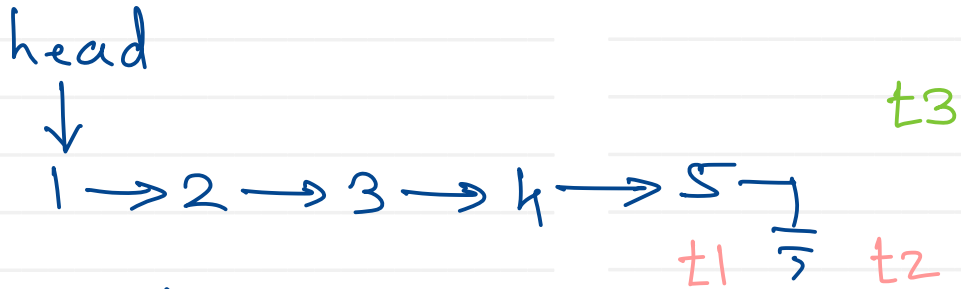
$S(n) = O(n)$



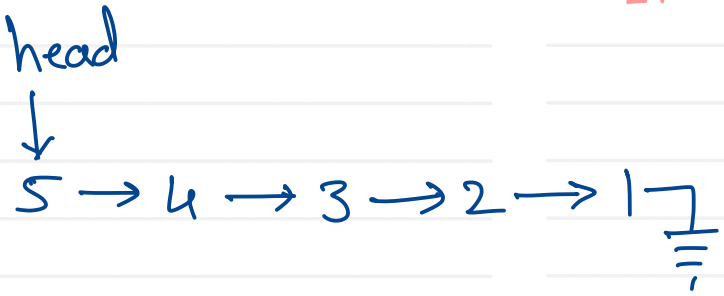
Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Example 1:
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]



Example 2:
Input: head = [1,2]
Output: [2,1]



Example 3:

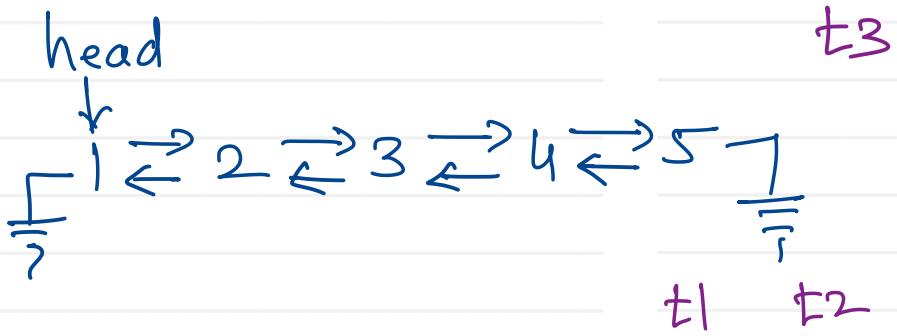
Input: head = []
Output: []

```
Node reverseList(Node head) {
    Node t1=null, t2=head, t3;
    while(t2!=null) {
        t3 = t2.next;
        t2.next = t1;
        t1 = t2;
        t2 = t3;
    }
    head = t1;
    return head;
}
```

Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Example 1:
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]



Example 2:
Input: head = [1,2]
Output: [2,1]

Example 3:

Input: head = []
Output: []

T(n) = O(n)

S(n) = O(1)

```
Node reverseList(head){
    Node t1 = head;
    Node t2 = head.next;
    Node t3;
    t1.next = null;
    while(t2 != null){
        t3 = t2.next;
        t2.next = t1;
        t1.prev = t2;
        t1 = t2;
        t2 = t3;
    }
    head = t1;
    return head;
}
```

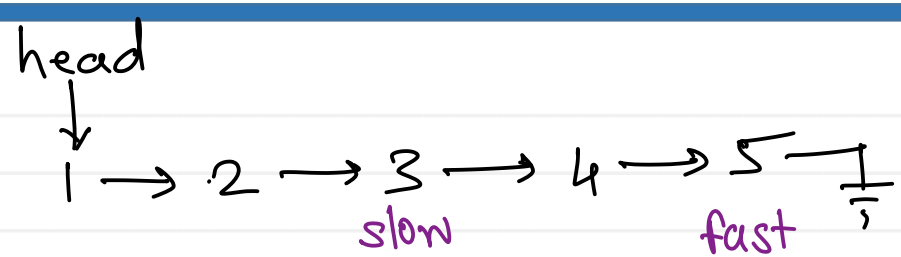
Middle of the Linked List

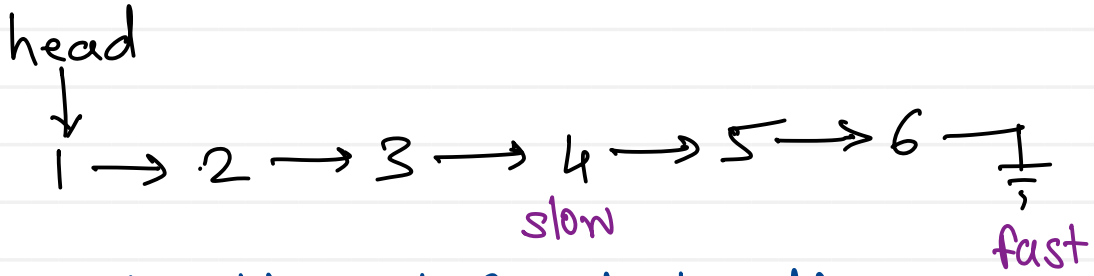
Given the head of a singly linked list, return the middle node of the linked list.

If there are two middle nodes, return the second middle node.

Example 1:
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.

Example 2:
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.






```
Node findMiddleNode(Node head){
    Node fast = head, slow = head;
    while(fast != null && fast.next != null){
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}
```

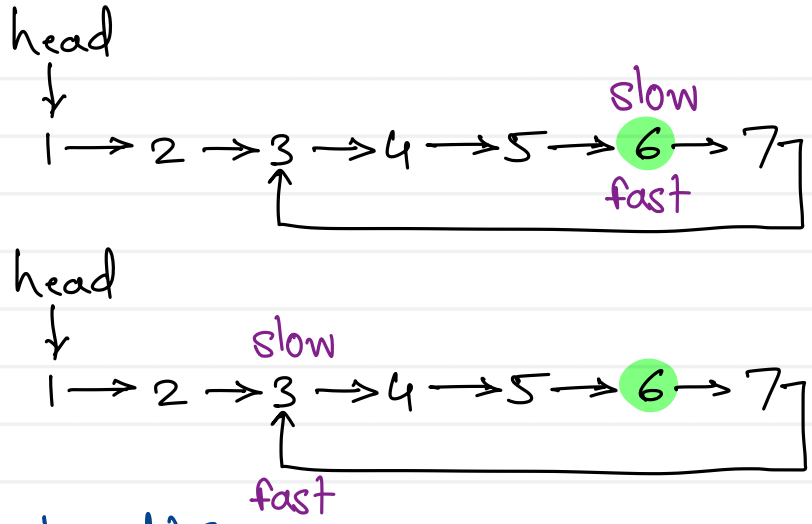
Linked List Cycle

(Floyd Cyclic Detection)

Given head, the head of a linked list, determine if the linked list has a cycle in it.

Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.



Example 1:
Input: head = [3,2,0,-4], pos = 1
Output: true

Example 2:
Input: head = [1,2], pos = 0
Output: true

Example 3:
Input: head = [1], pos = -1
Output: false

```
boolean hasCycle(Node head){
    Node fast = head, slow = head;
    while(fast != null && fast.next != null){
        fast = fast.next.next;
        slow = slow.next;
        if(fast == slow)
            return true;
    }
    return false;
}
```

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com