Sunbeam Institute of Information Technology
Pune and Karad

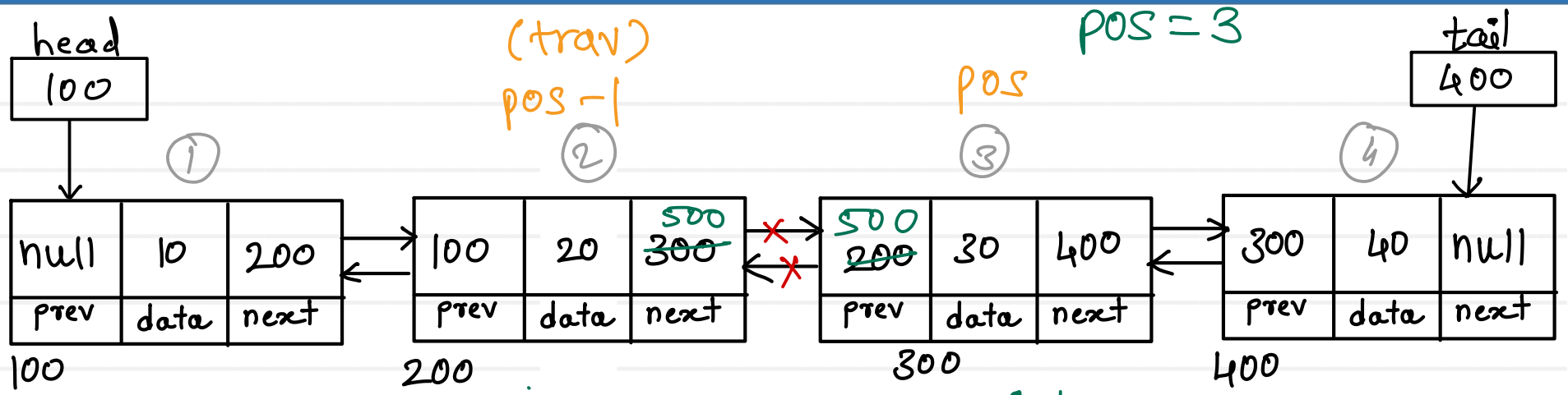
Algorithms and Data structures

Trainer - Devendra Dhande

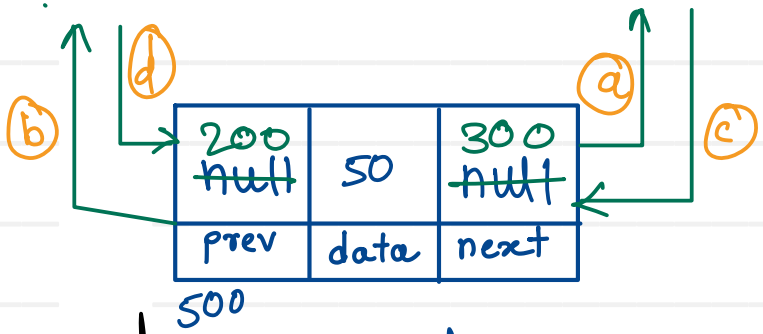
Email – devendra.dhande@sunbeaminfo.com

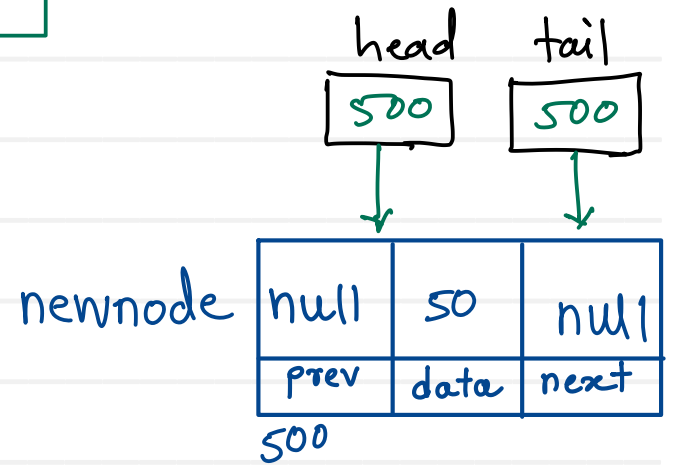
Doubly Linear Linked List - Add position

size = 4
valid positions:
1 → size+1
invalid positions:
< 1
> size+1

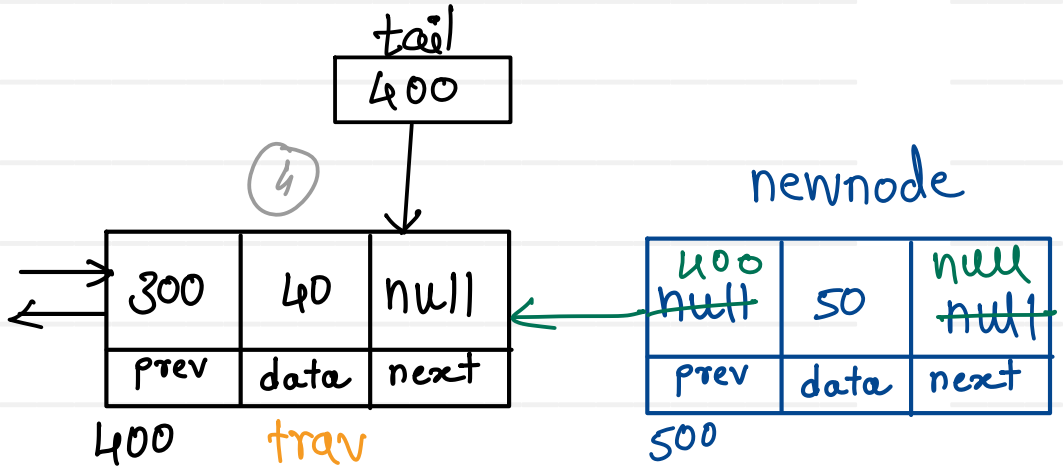


1. Create node
2. if list is empty
   add newnode in head & tail
3. if list is not empty
   traverse till pos-1 node
   a. add pos node into next of newnode
   b. add pos-1 node into prev of newnode
   c. add newnode into prev of pos node
   d. add newnode into next of pos-1 node





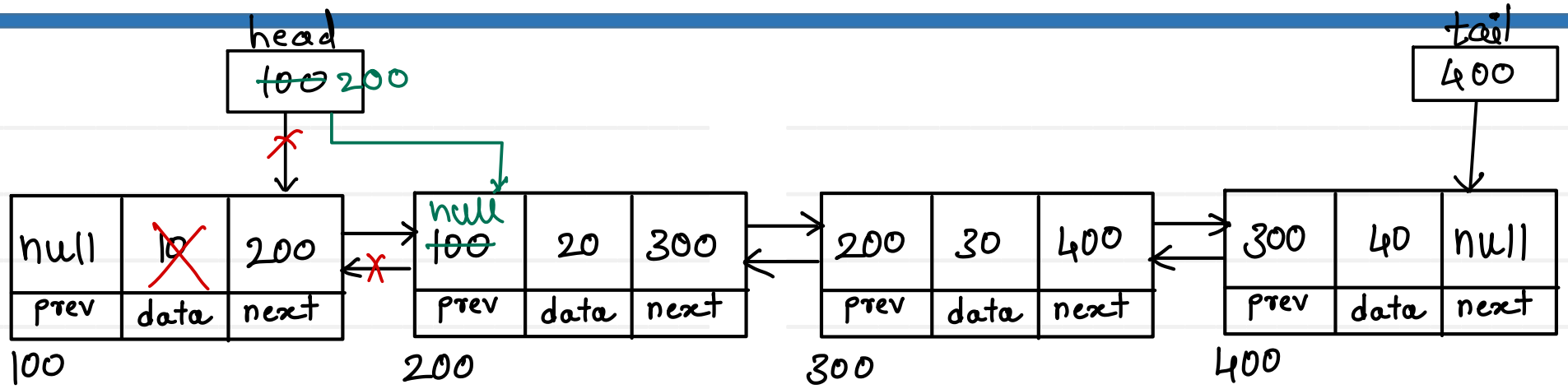
pos = 5

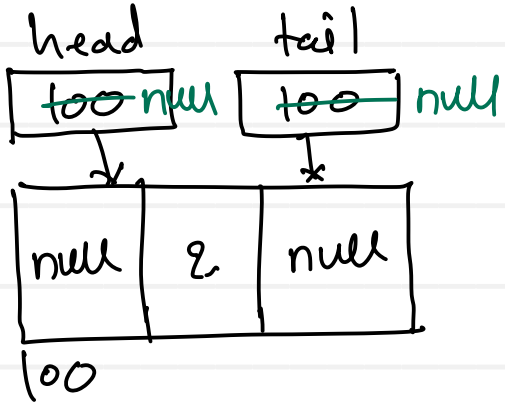



```
newnode.next = trav.next
newnode.prev = trav
trav.next.prev = newnode
trav.next = newnode
```

null pointer exception →

Doubly Linear Linked List - Delete first



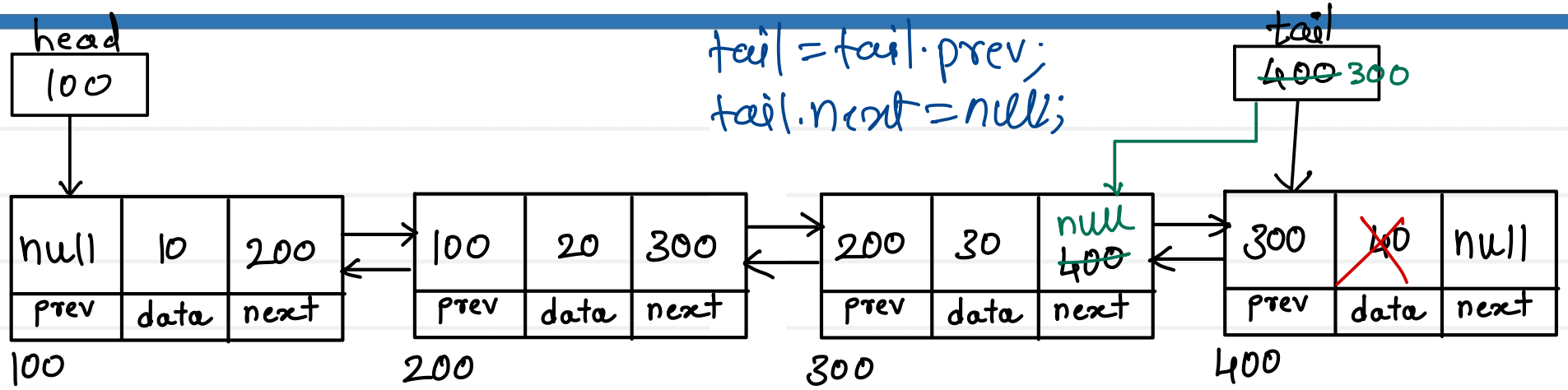



```
head.next.prev = null;
head = head.next;
```

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

$T(n) = O(1)$

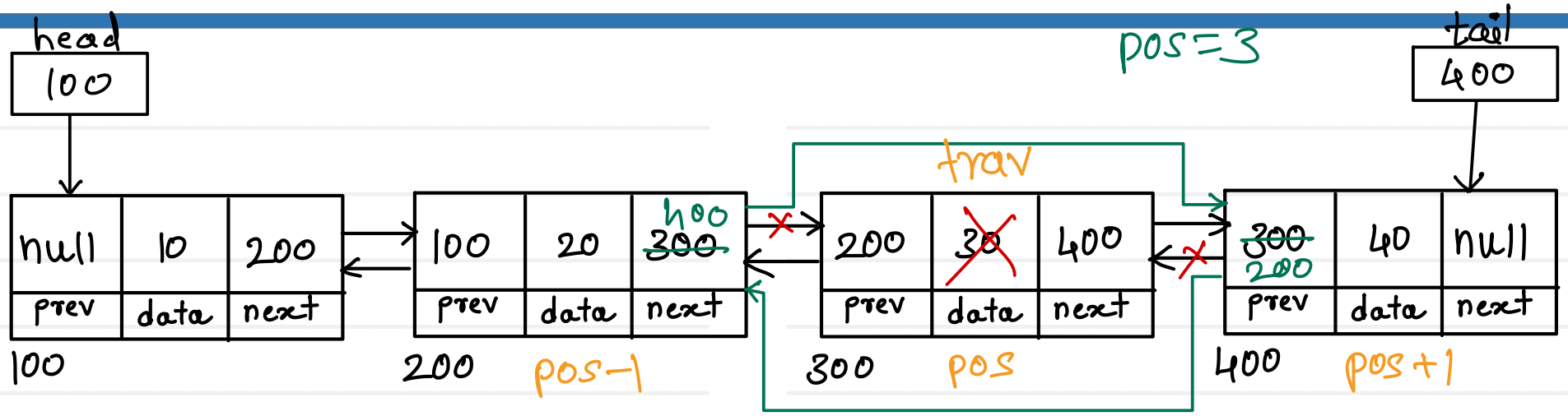
Doubly Linear Linked List - Delete last



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

$T(n) = O(1)$

Doubly Linear Linked List - Delete Position

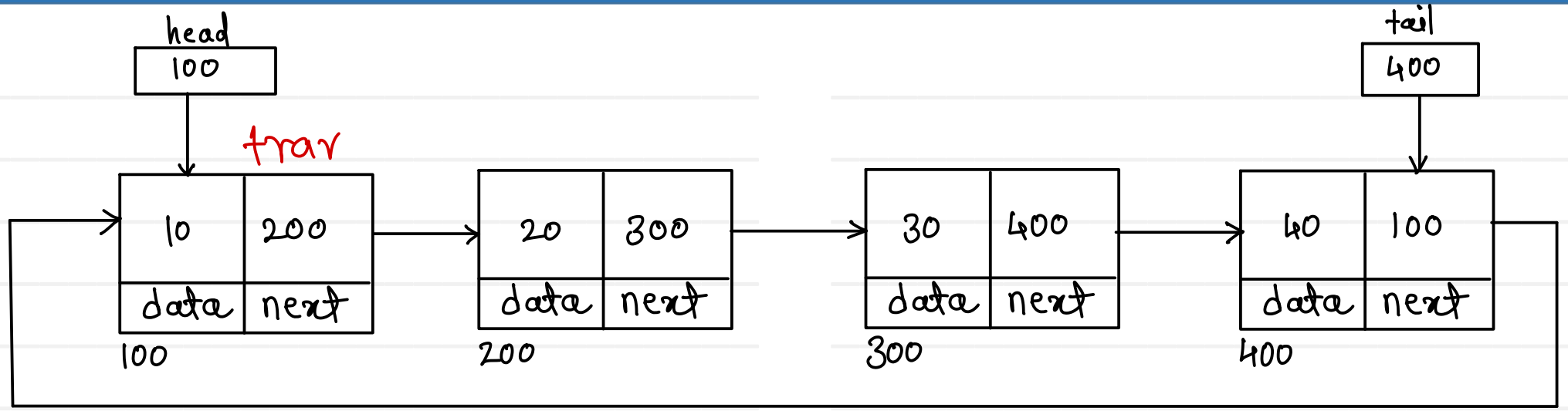


1. if list is empty, return
2. if list is not empty,
   a. traverse till pos node
   b. add pos+1 node into next of pos-1 node
   c. add pos-1 node into prev of pos+1 node

size = 4
valid positions :
1 to size
invalid positions :
< 1
> size

$T(n) = O(n)$

Singly Circular Linked List - Display



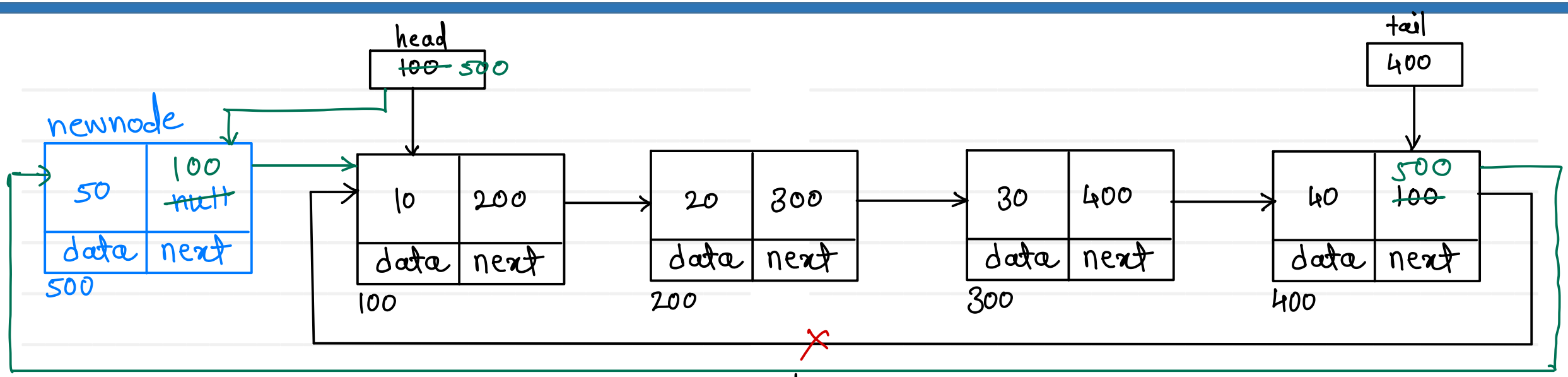
1. create trav & start at first node
2. visit / print current data
3. go on next node
4. repeat above two steps till last node

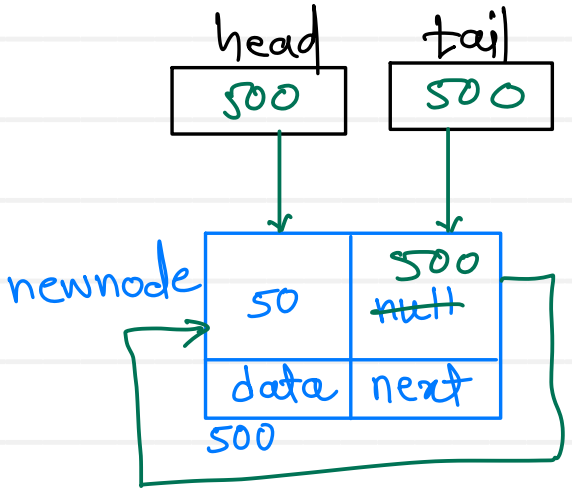
$T(n) = O(n)$

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

trav
100 ✓
200 ✓
300 ✓
400 ✓
100 ✗

Singly Circular Linked List - Add first

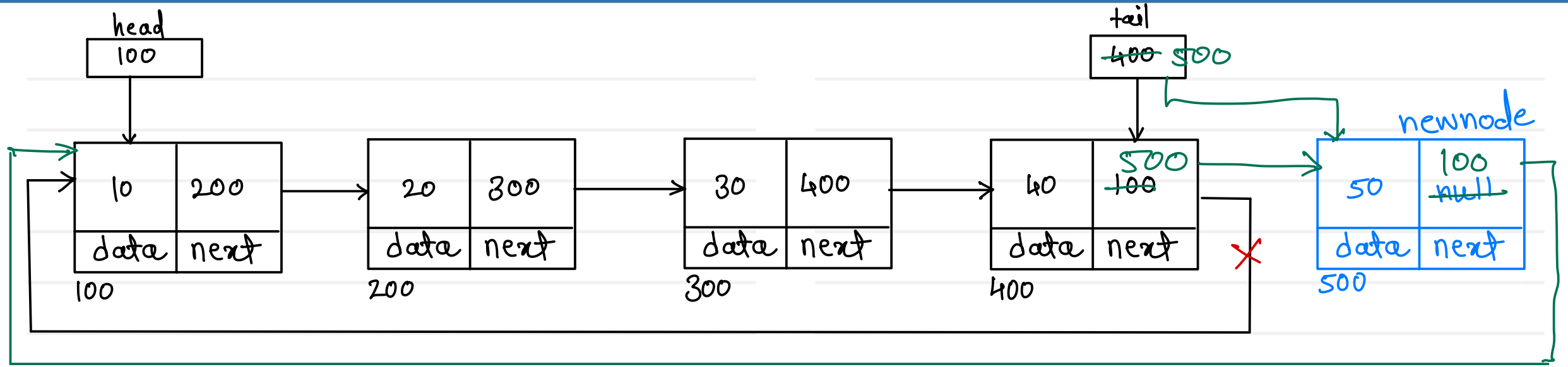


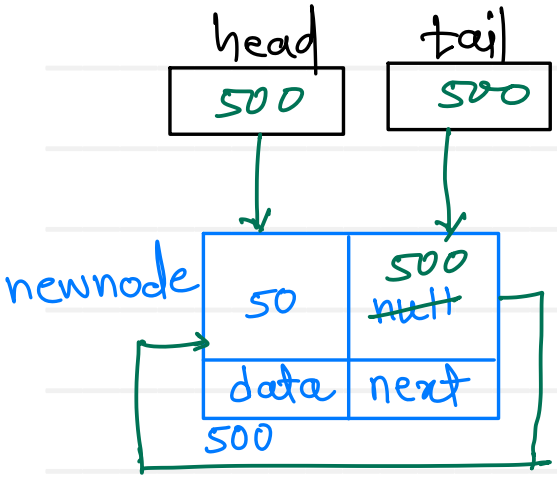


1. create newnode
2. if list is empty,
   a. add newnode into head & tail
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into next of last node
   c. move head on newnode

$T(n) = O(1)$

Singly Circular Linked List - Add last

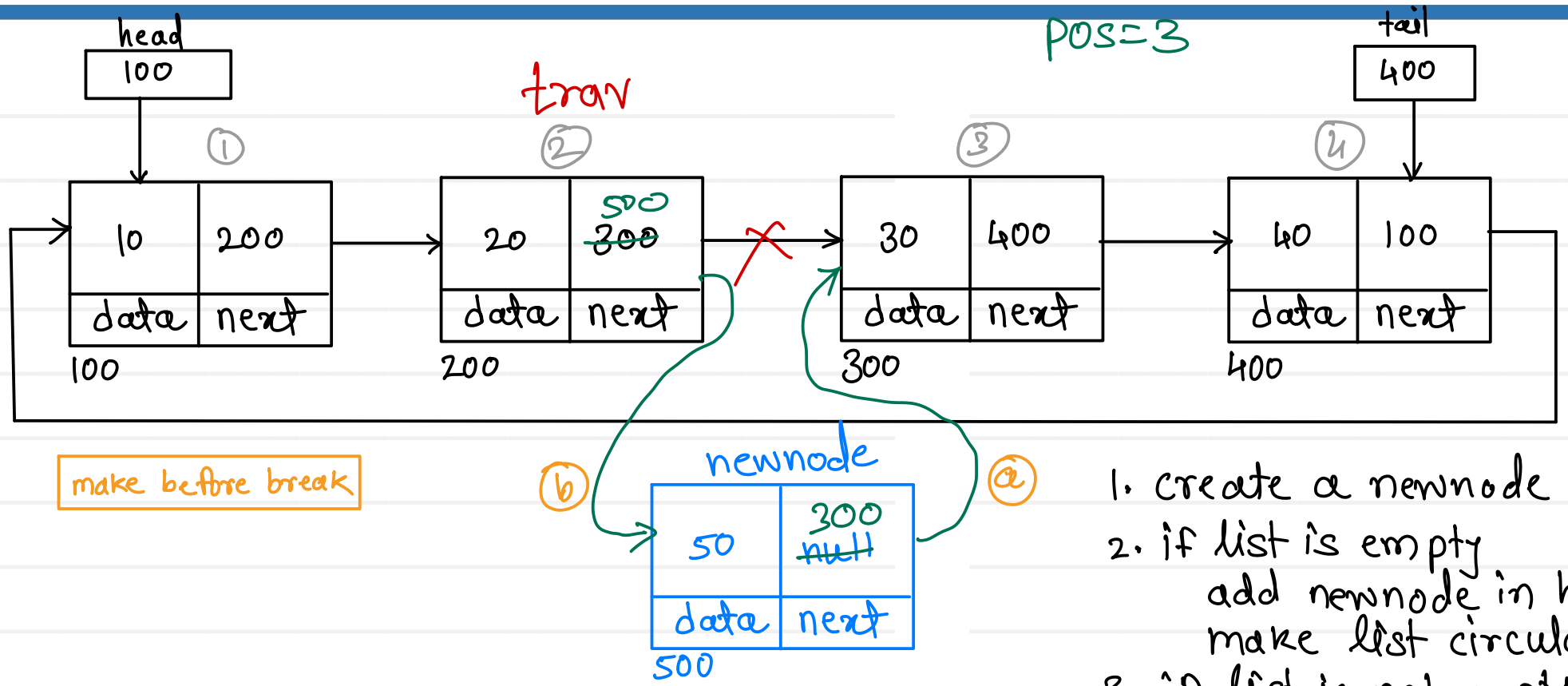




1. create newnode
2. if list is empty,
   a. add newnode into head & tail
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into next of last node
   c. move tail on newnode

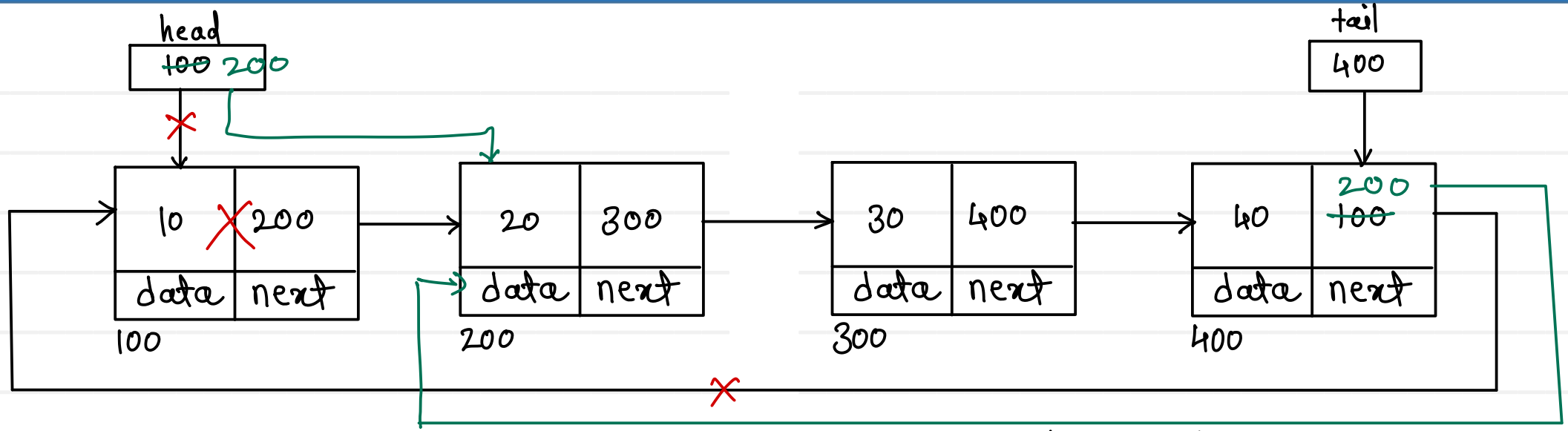
$T(n) = O(1)$

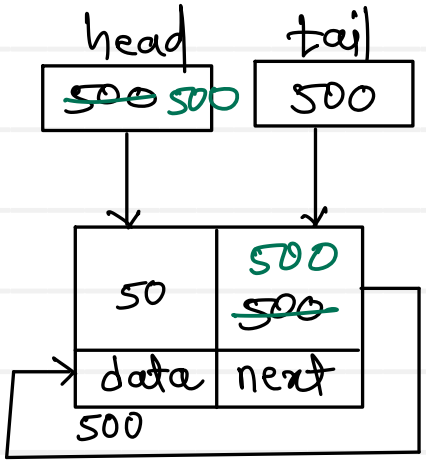
Singly Circular Linked List - Add position



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

Singly Circular Linked List - Delete first



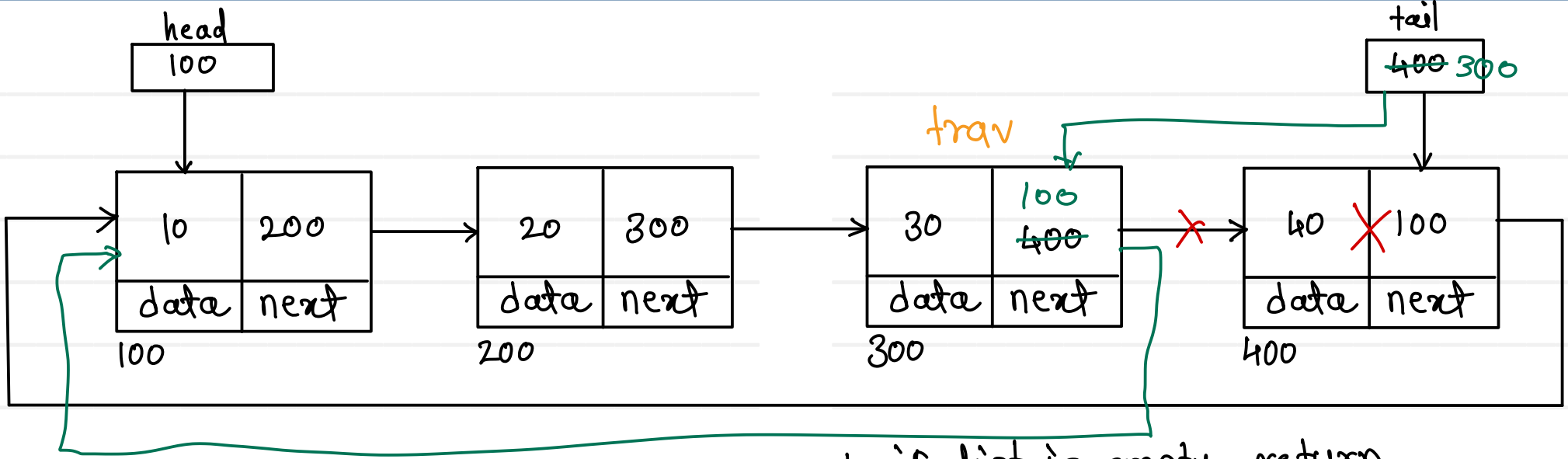


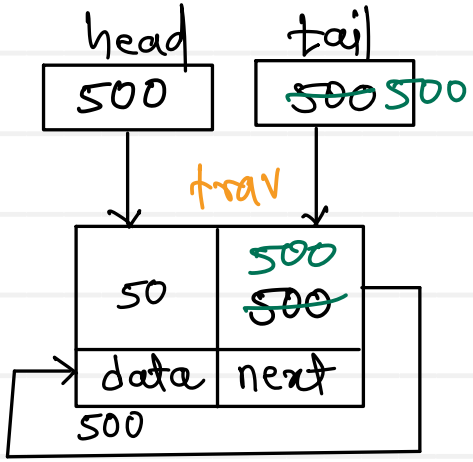

```
tail.next = head.next;
head = head.next;
```

1. if list is empty, return
2. if list has single node
   head = tail = null
3. if list has 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



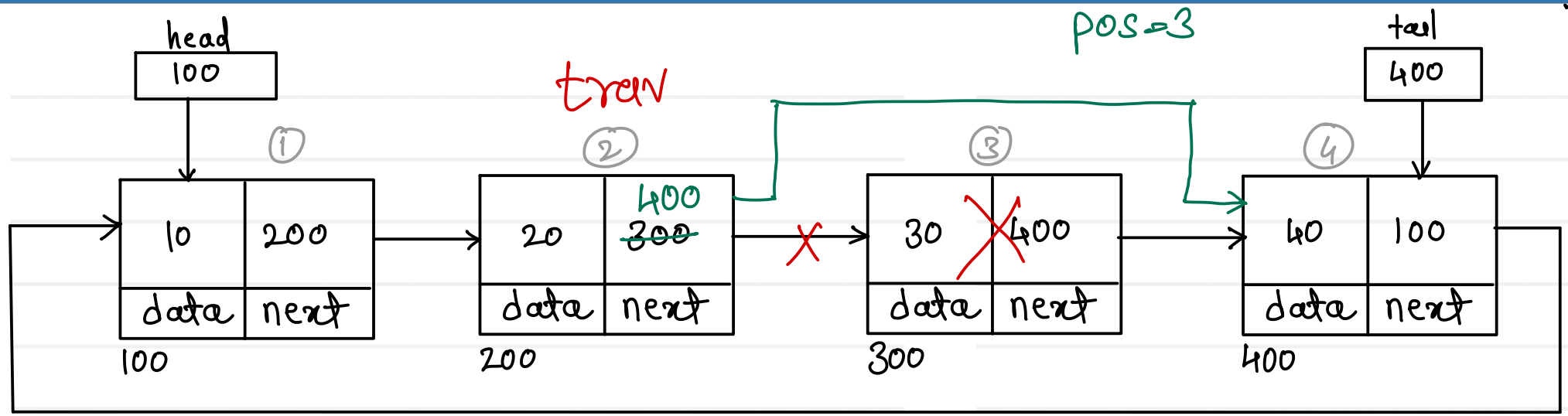



```
while(trav.next.next != head)
    trav = trav.next;
trav.next = head;
tail = trav;
```

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

$T(n) = O(n)$

Singly Circular Linked List - Delete position

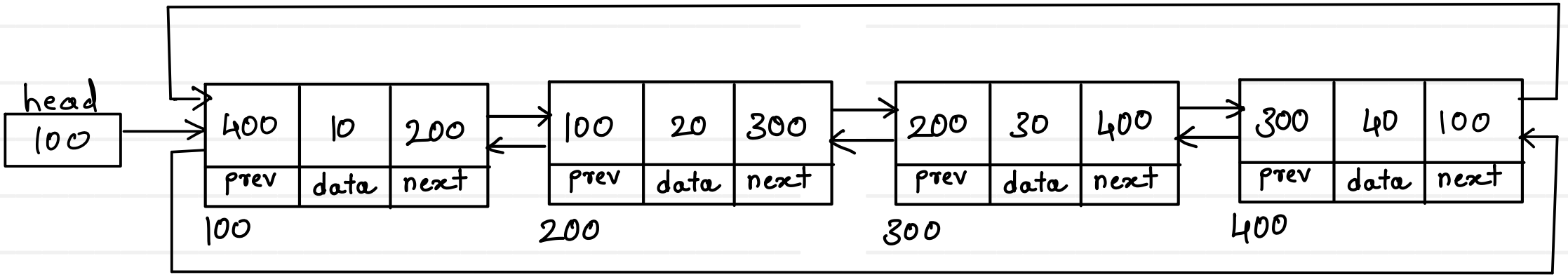


valid positions :
1 to size
invalid positions :
pos < 1
pos > size

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

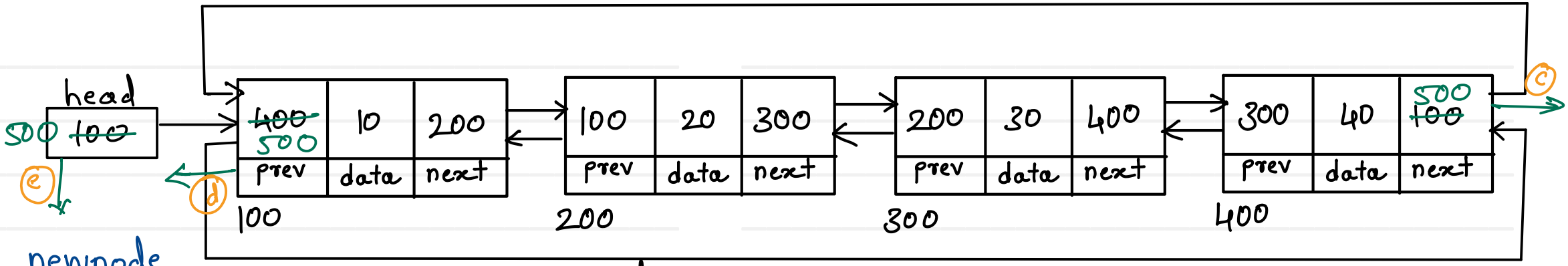


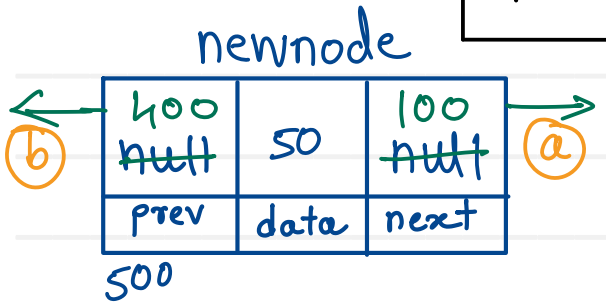
1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

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

$T(n) = O(n)$

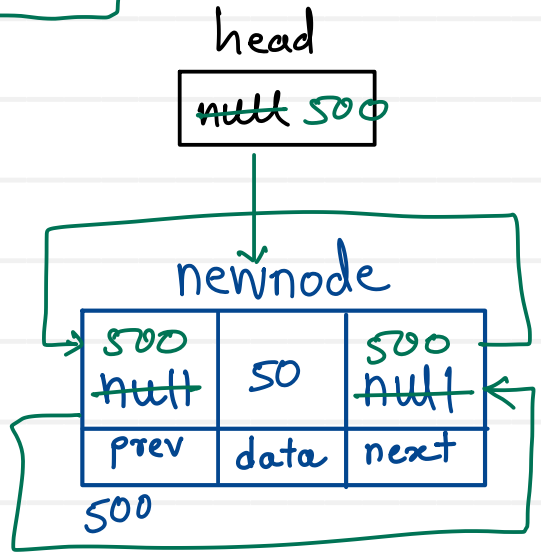
Doubly Circular Linked List - Add first



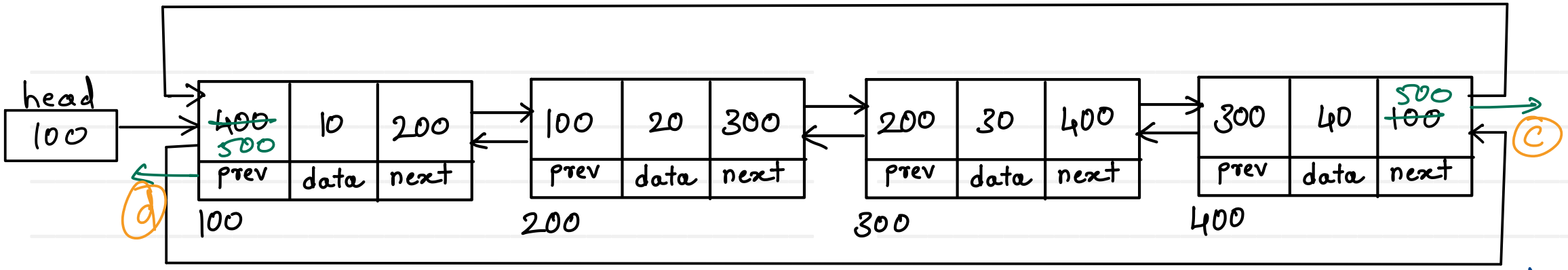


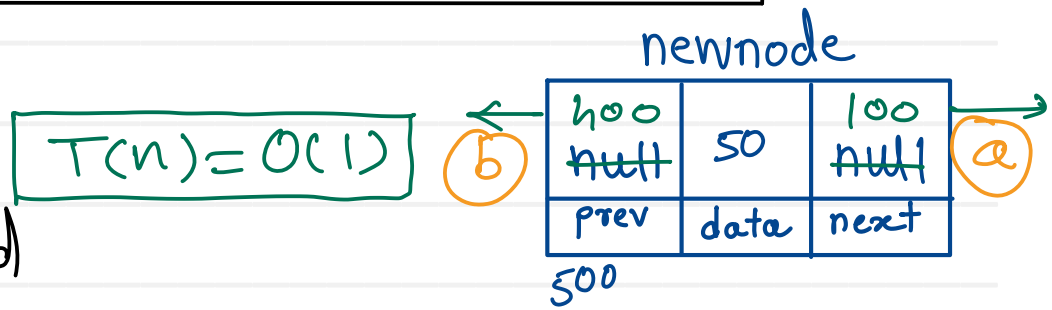
1. Create newnode
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add last node into prev of newnode
   c. add newonode into next of last node
   d. add newnode into prev of first node
   e. move head on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

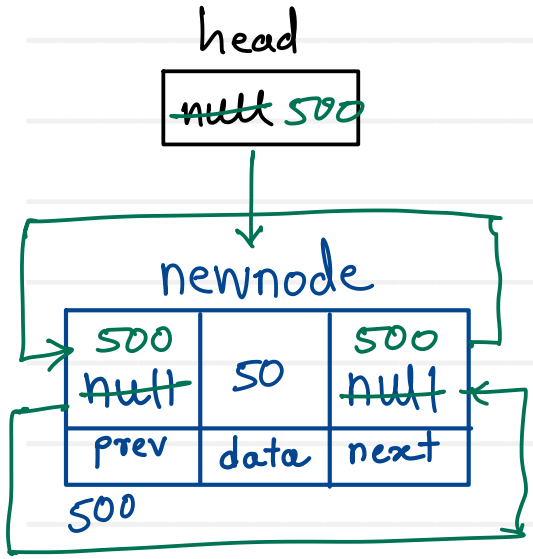
$T(n) = O(1)$



Doubly Circular Linked List - Add last

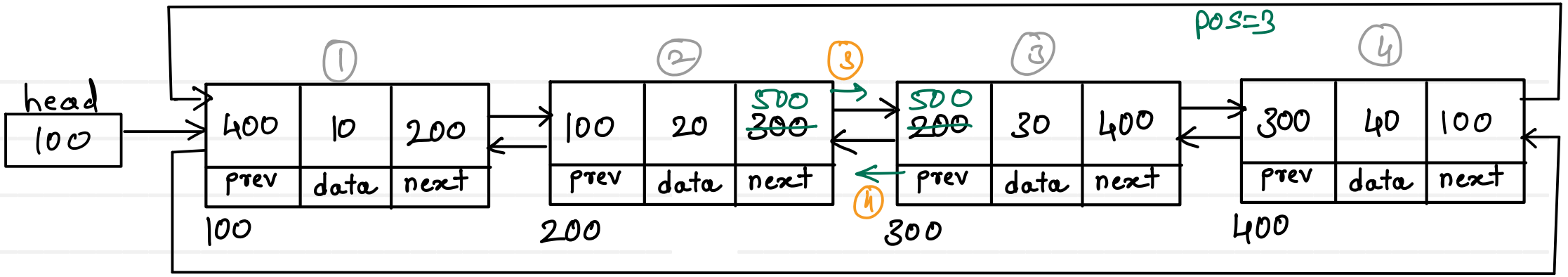




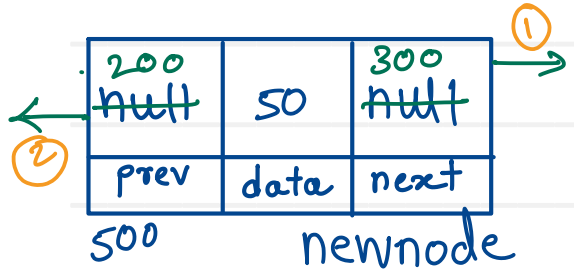


1. Create newnode
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add last node into prev of newnode
   c. add newonode into next of last node
   d. add newnode into prev of first node

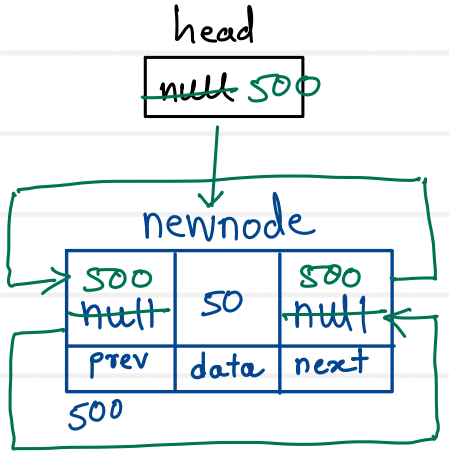
Doubly Circular Linked List - Add position



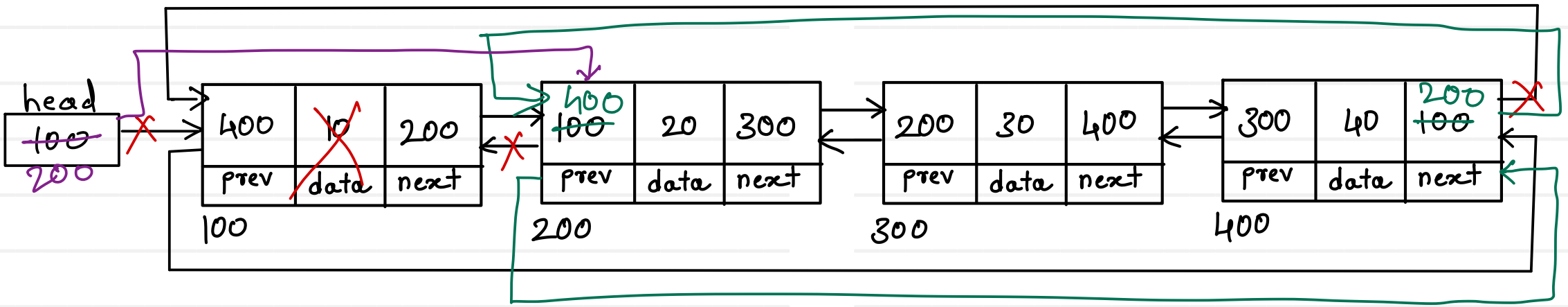
1. create node
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty.
   a. traverse till pos-1 node
   b. add pos node into next of newnode
   c. add pos-1 node into prev of newnode
   d. add newnode into next of pos-1 node
   e. add newnode into prev of pos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



$T(n) = O(1)$



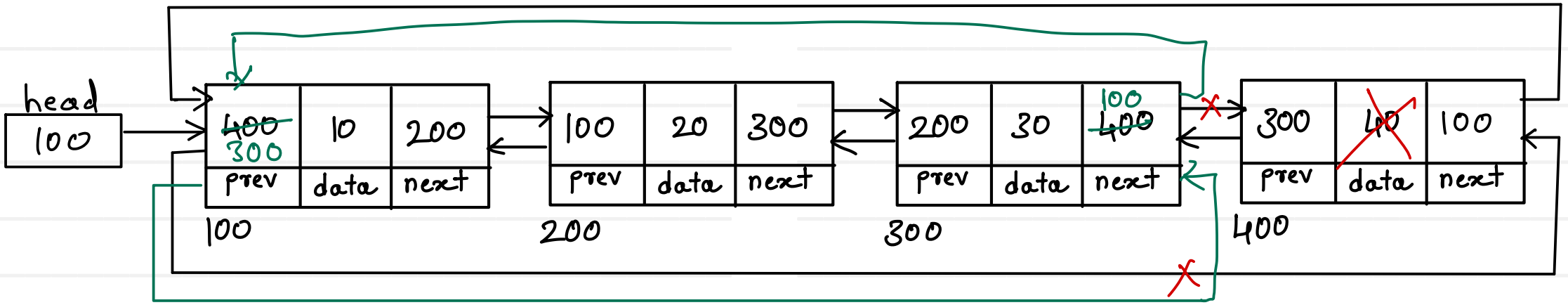
Doubly Circular Linked List - Delete first



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

$T(n) = O(1)$

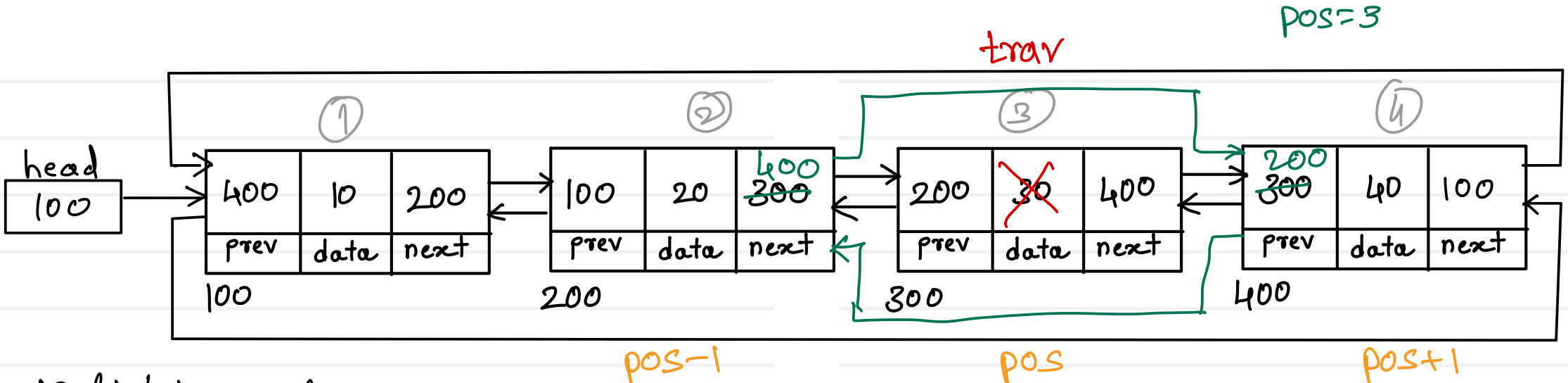
Doubly Circular Linked List - Delete last



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
   b. add second last node into prev of first node

$T(n) = O(1)$

Doubly Circular Linked List - Delete position

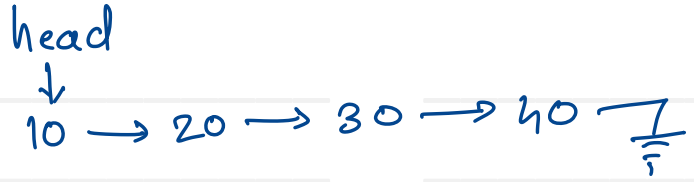


1. if list is empty
   return;
2. if list has single node
   head = tail = null;
3. if list has multiple nodes
   a. traverse till pos node
   b. add pos+1 node into next of pos-1 node
   c. add pos-1 node into prev of pos+1 node

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

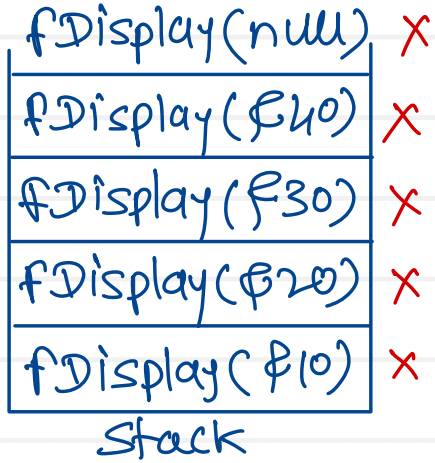
$T(n) = O(n)$

Singly linear linked list - Display reverse



Tail Recursion

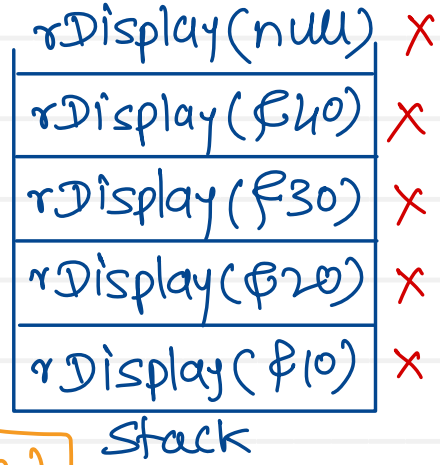
```
void fDisplay(Node trav) {
    if(trav == null)
        return;
    sysout(trav.data);
    fDisplay(trav.next);
}
```



10, 20, 30, 40

Non tail Recursion

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



40, 30, 20, 10

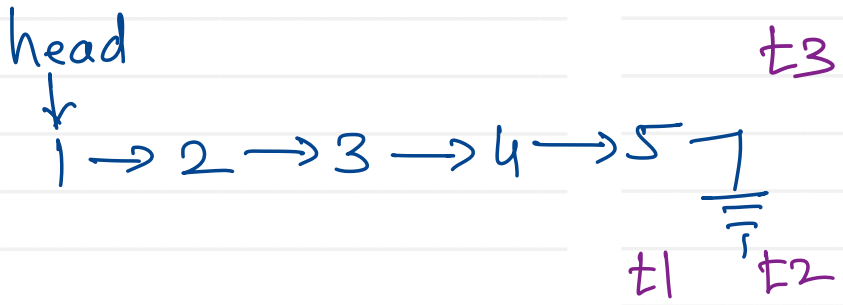
T(n) = O(n)

S(n) = O(n)

Reverse Linked List

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

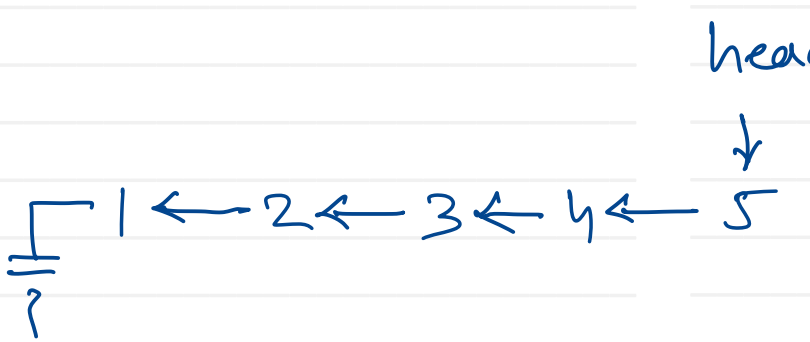
Example 1:
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]



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

Example 3:

Input: head = []
Output: []



$T(n) = O(n)$

$S(n) = O(1)$

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

Middle of the Linked List

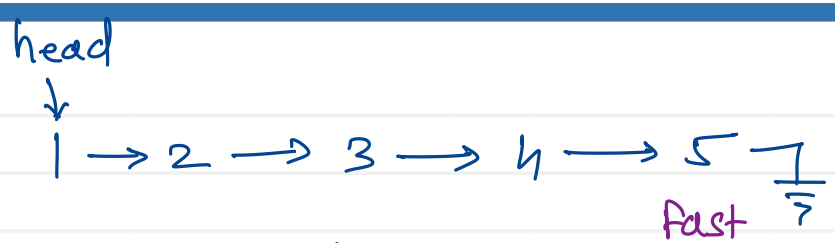
Given the head of a singly linked list, return the middle node of the linked list.

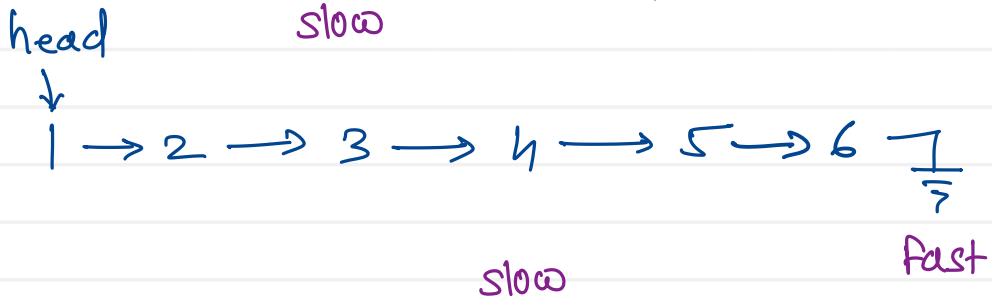
If there are two middle nodes, return the second middle node.

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.

$T(n) = O(n)$





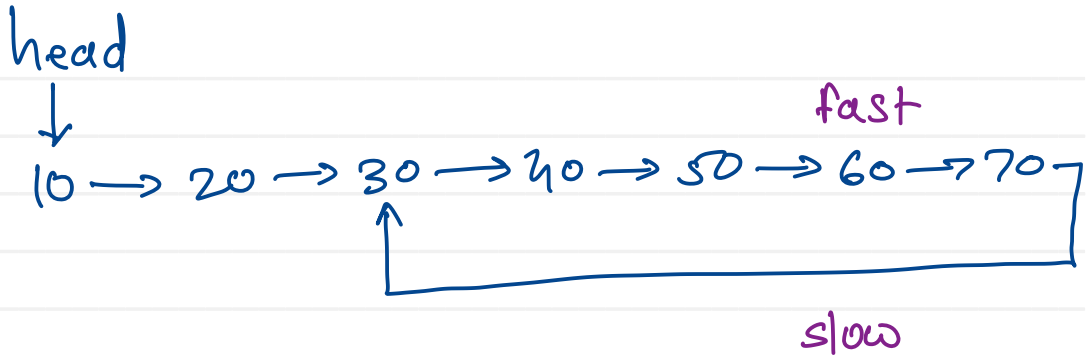

```
Node findMid(Node head) {
    Node slow = head, fast = head;
    while(fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}
```

Linked List Cycle

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

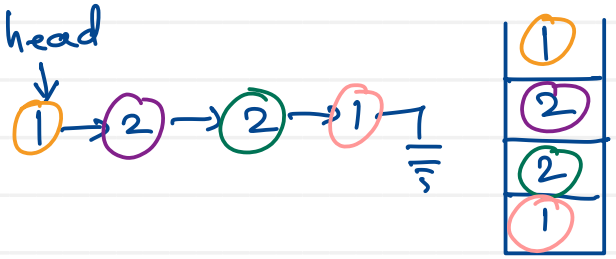
```
boolean hasCycle(Node head) {
    Node slow = head, fast = head;
    while( fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
        if(fast == slow)
            return true;
    }
    return false;
}
```

T(n) = O(n)

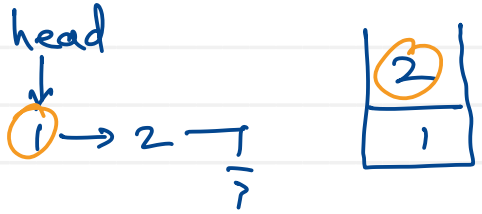
Palindrome Linked List

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

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



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



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

```
boolean isPalindrome(Node head) {
    Stack<Integer> st = new Stack<>();
    Node trav = head;
    while (trav != null) {
        st.push(trav.data);
        trav = trav.next;
    }
    Node trav = head;
    while(!st.isEmpty()) {
        if(trav.data != st.pop())
            return false;
        trav = trav.next;
    }
    return true;
}
```

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com