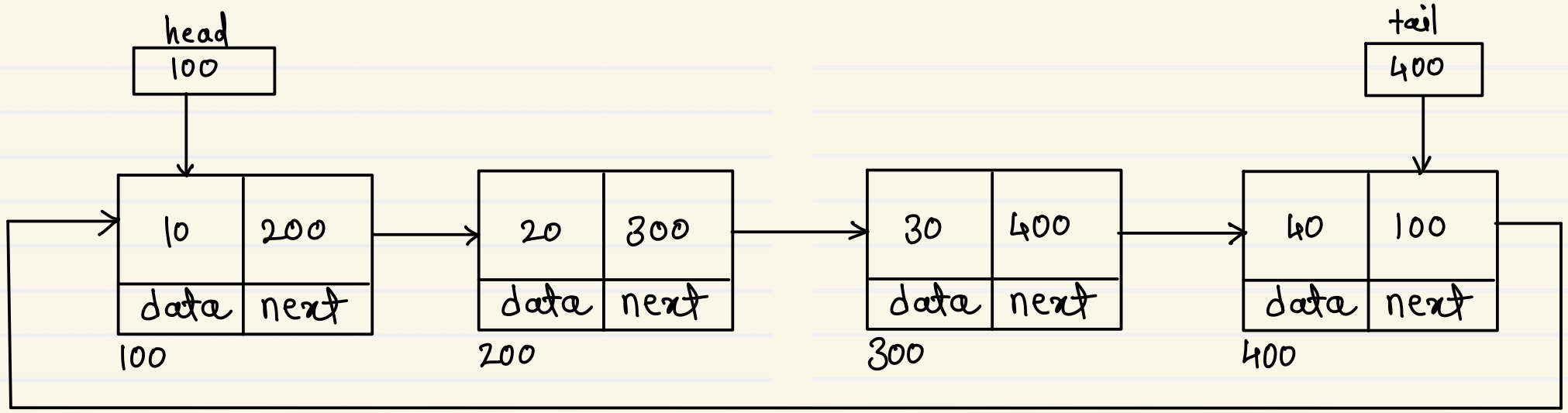
Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

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

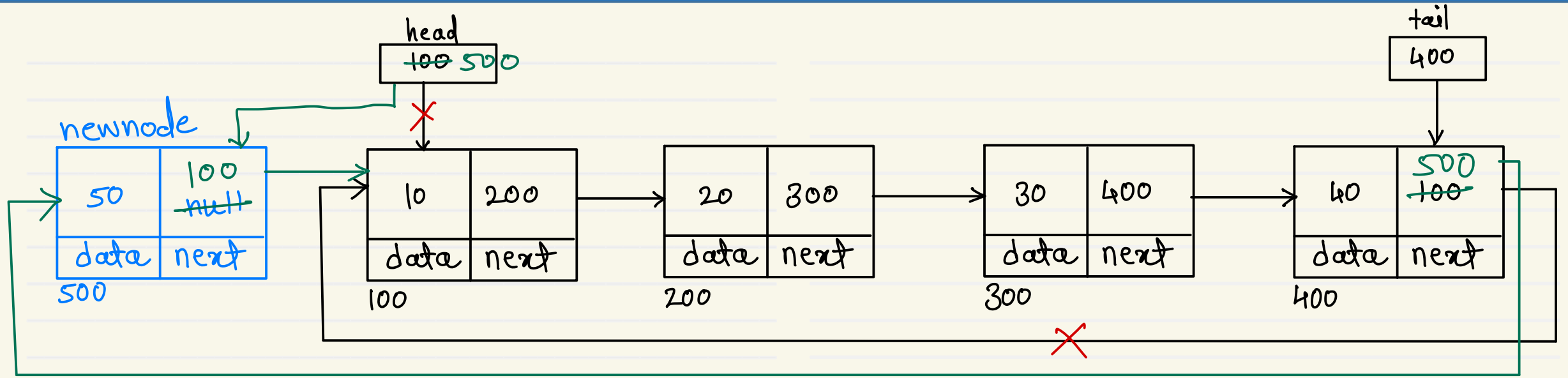
$T(n) = O(n)$

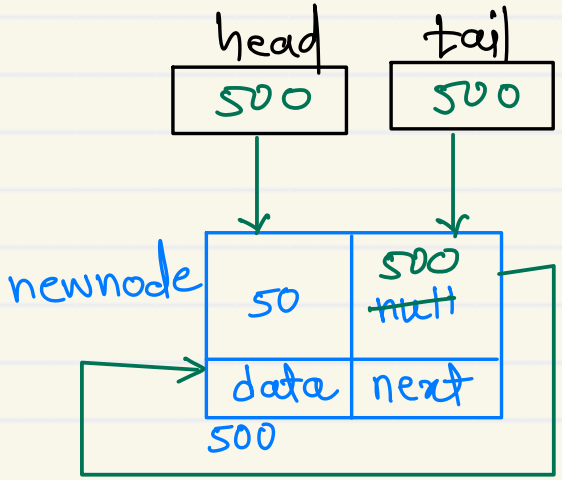
```
Node trav = head;
do {
    sysout ( trav.data);
    trav = trav.next;
} while ( trav != head )
```

trav
100
200
300
400
100

output : 10 20 30 40

Singly Circular Linked List - Add first

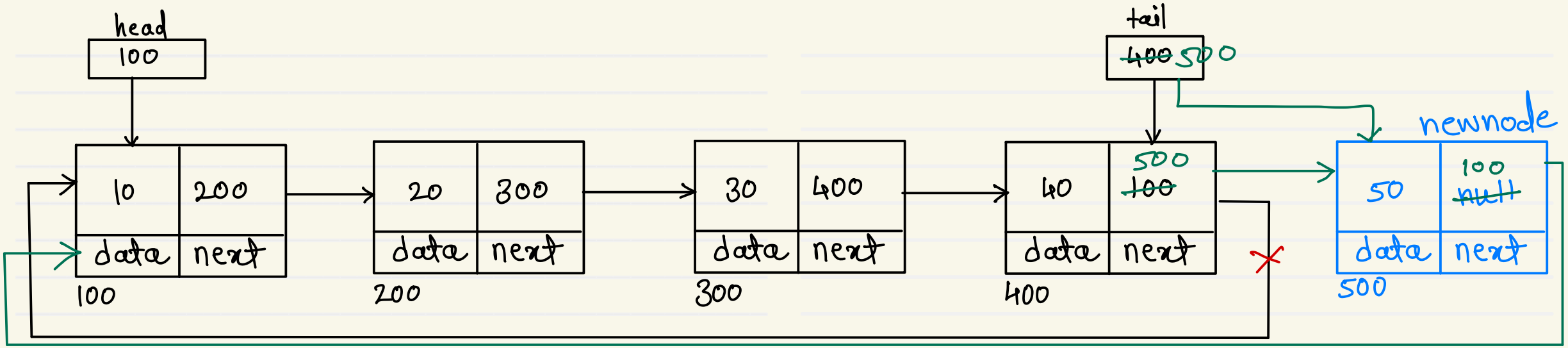


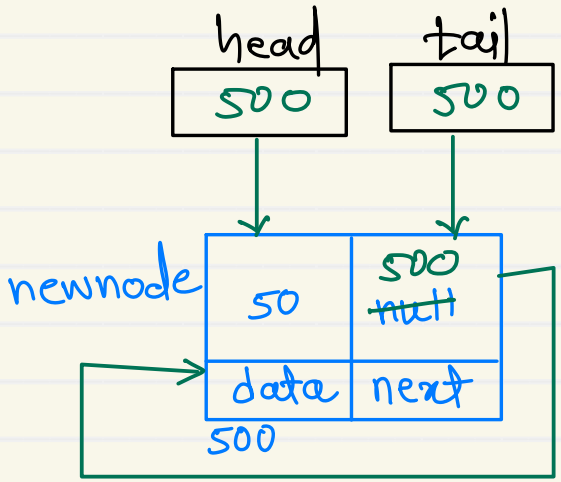


1. create a newnode
2. if list is empty,
   a. add newnode into head & tail
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into next of last node
   c. move head on newnode

$T(n) = O(1)$

Singly Circular Linked List - Add last

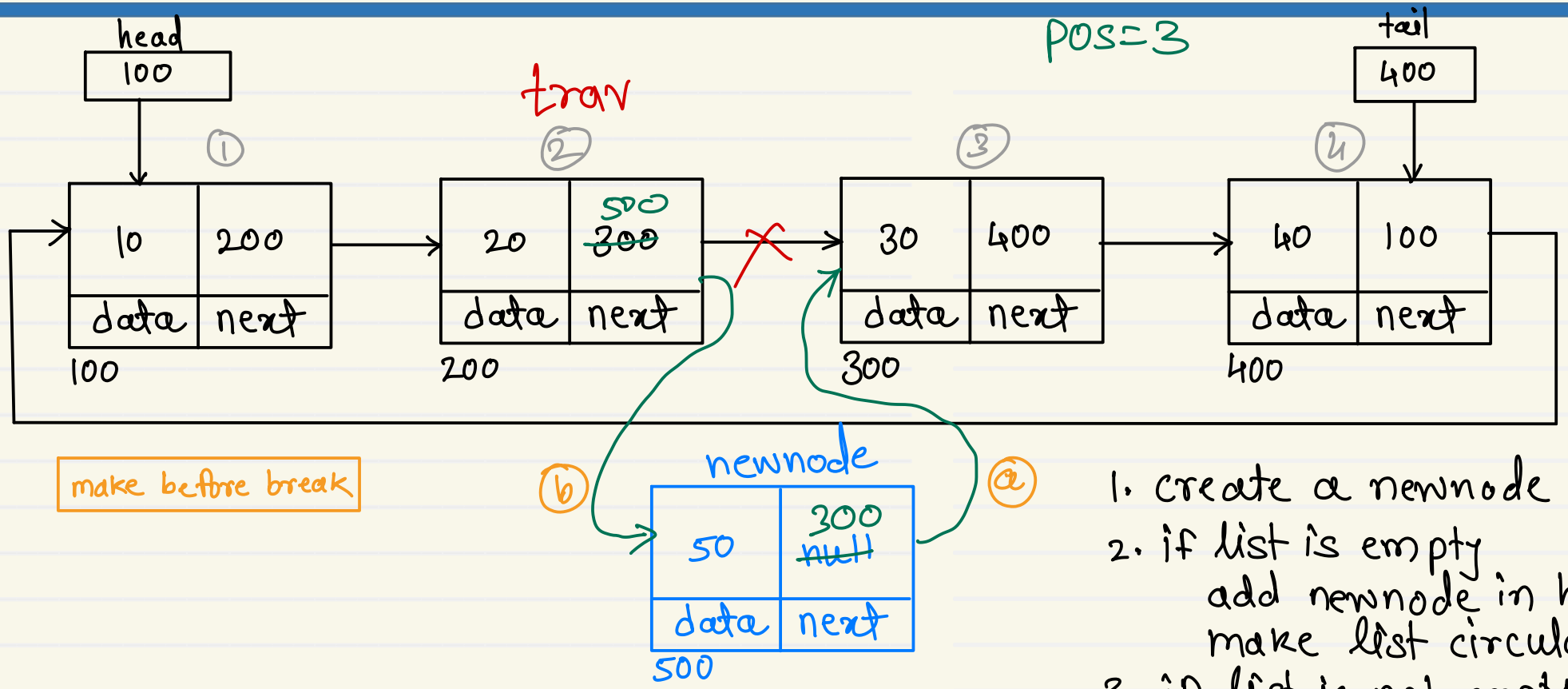




1. create a newnode
2. if list is empty,
   a. add newnode into head & tail
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into next of last node
   c. move tail on newnode

$T(n) = O(1)$

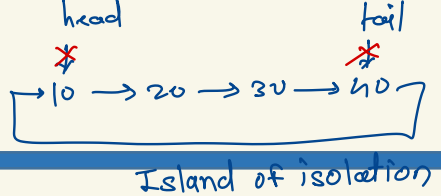
Singly Circular Linked List - Add position

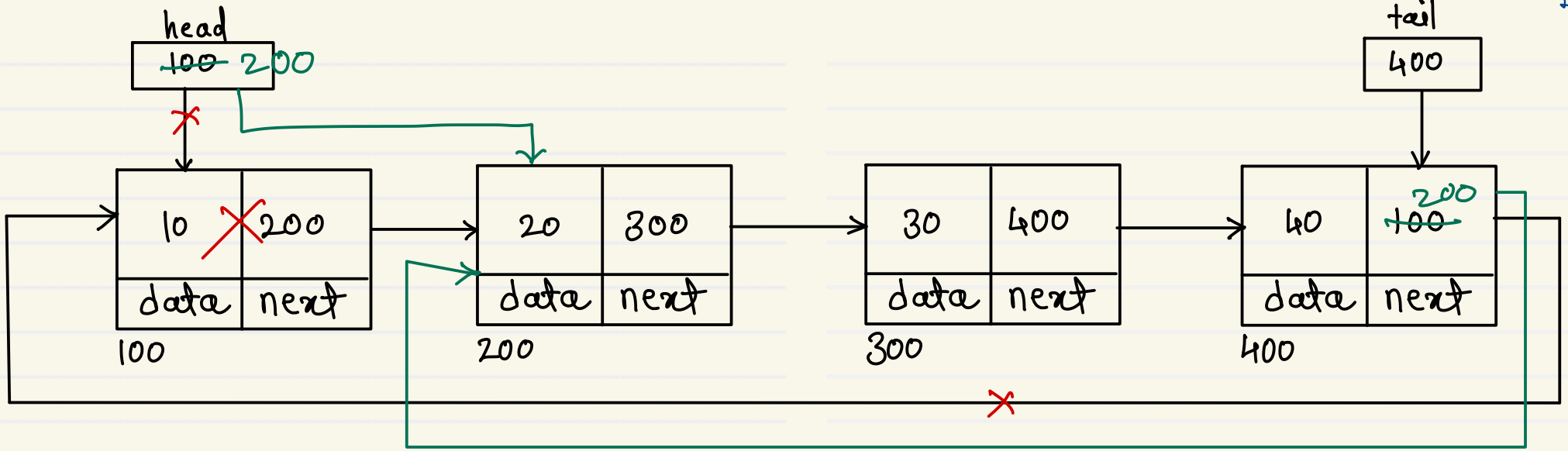


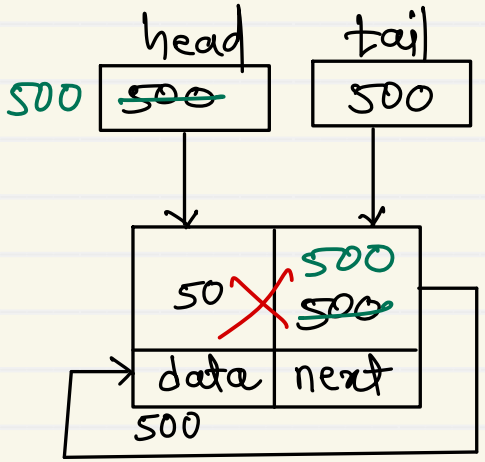
1. create a newnode
2. if list is empty
   add newnode in head & tail.
   make list circular.
3. if list is not empty
   a. traverse till pos-1 node
   b. add pos node into next of newnode
   c. add newnode into next of pos-1 node

$T(n) = O(n)$

Singly Circular Linked List - Delete first





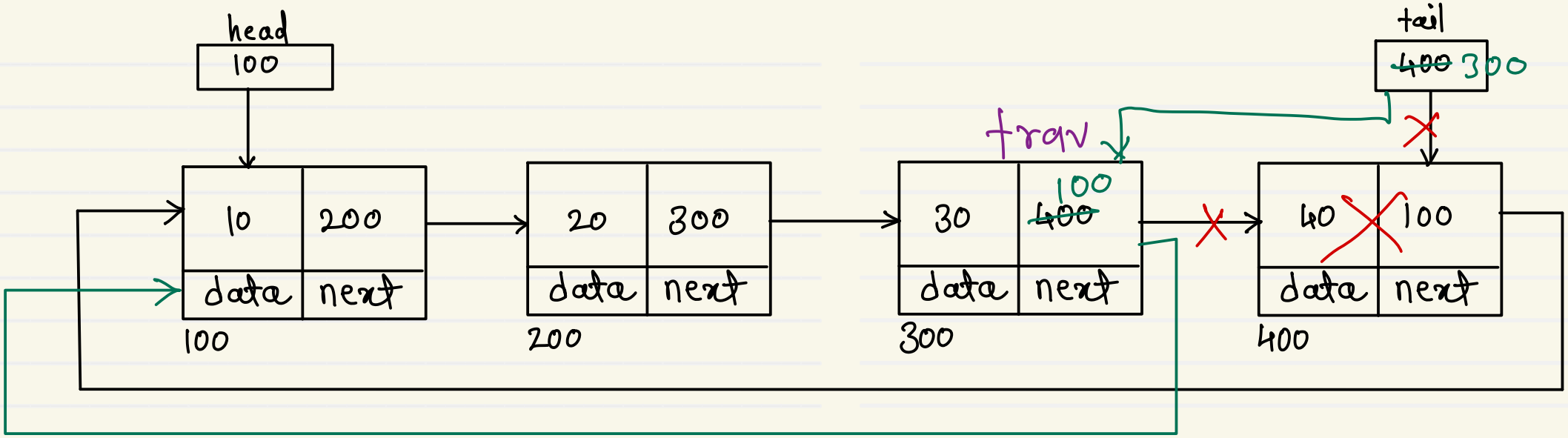


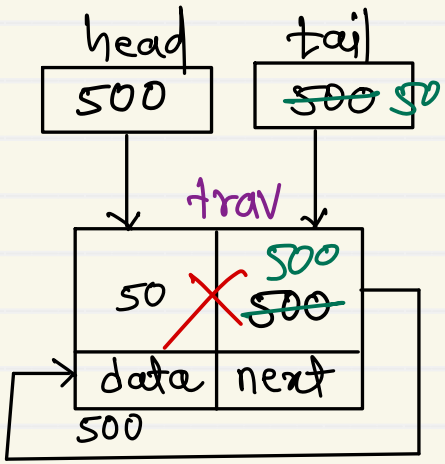

```
tail.next = head.next;
head = head.next;
```

1. if list is empty, return
2. if list has single node,
   make head & tail equal to 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



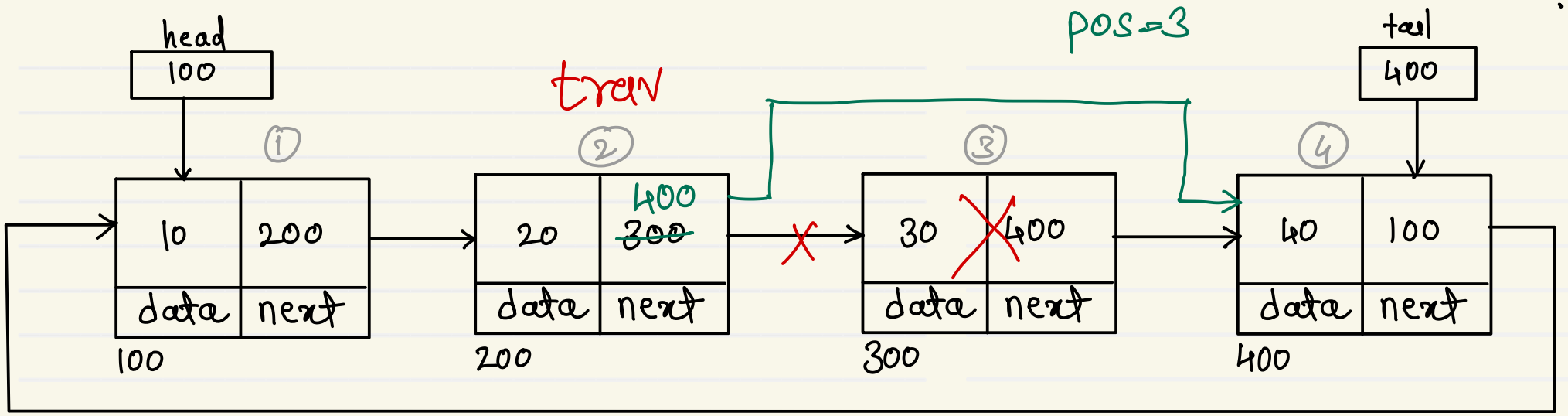



```
while(trav.next.next != head)
    trav = trav.next;

or

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

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

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

$T(n) = O(n)$

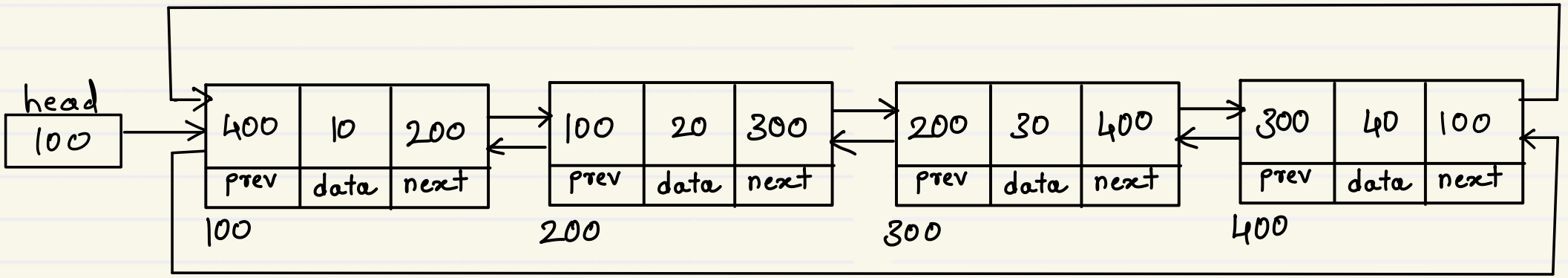
Singly Circular Linked List - Delete position



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

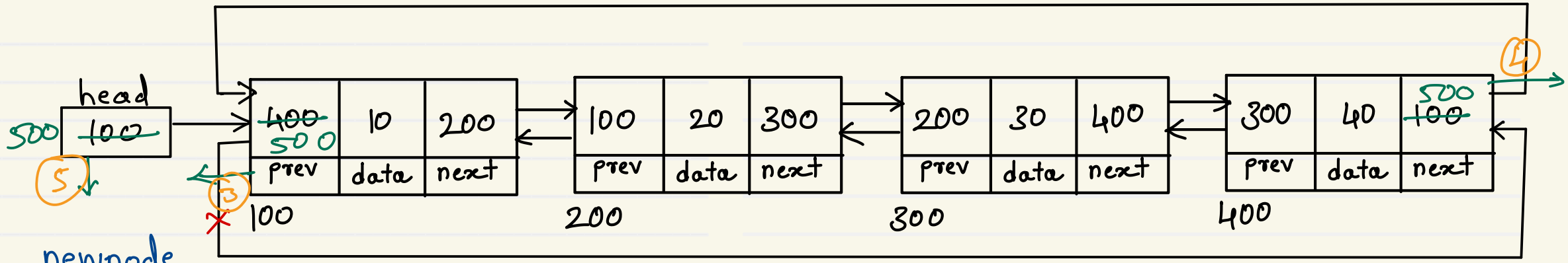


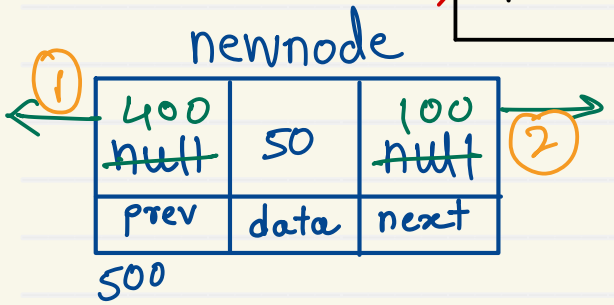
1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

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

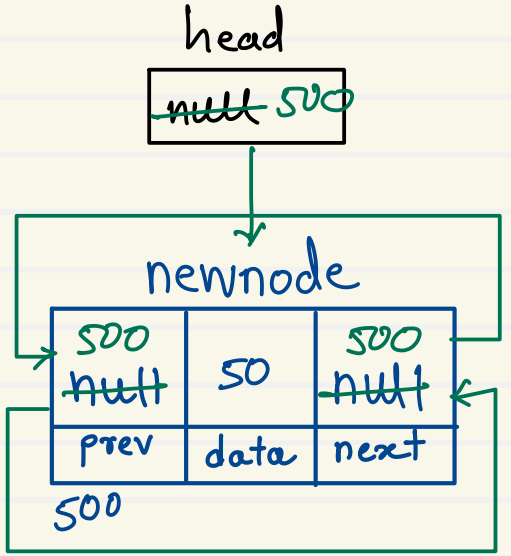
$$T(n) = O(n)$$

Doubly Circular Linked List - Add first



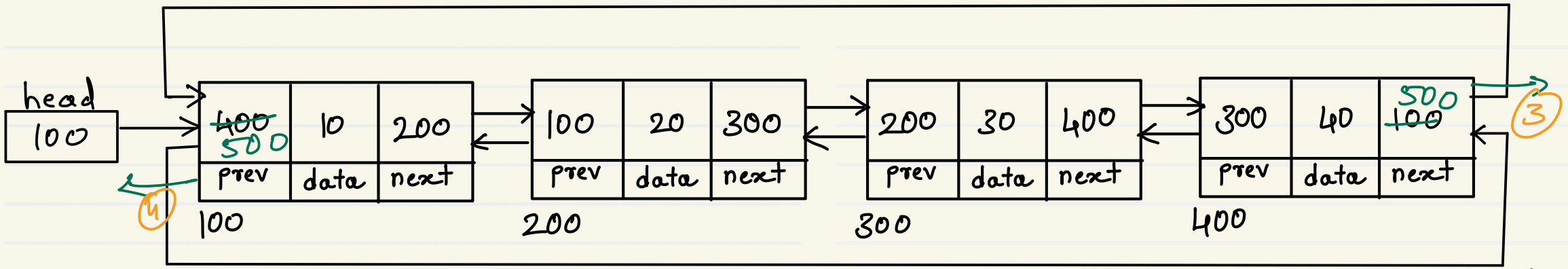


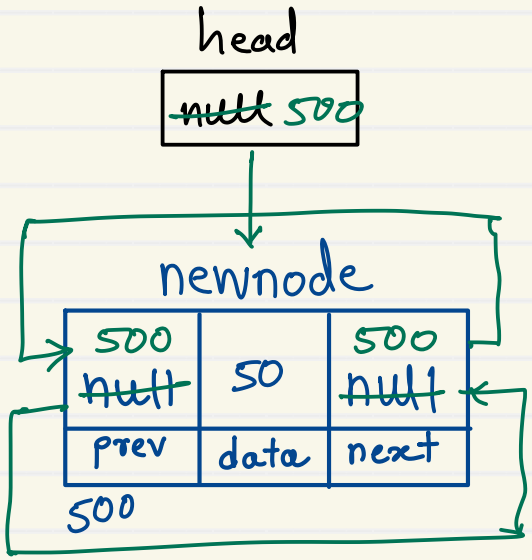
a. Create a newnode
b. if list is empty,
   1. add newnode into head
   2. make list circular
c. if list is not empty,
   1. add last node into prev of newnode
   2. add first node into next of newnode
   3. add newnode into prev of firstnode
   4. add newnode into next of last node
   5. move head on newnode

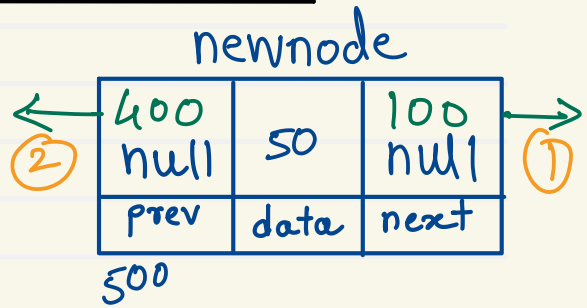


$T(n) = O(1)$

Doubly Circular Linked List - Add last



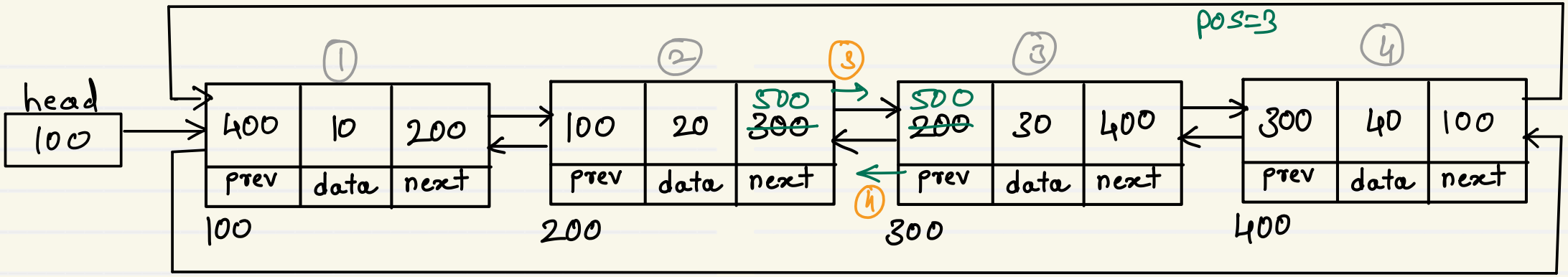




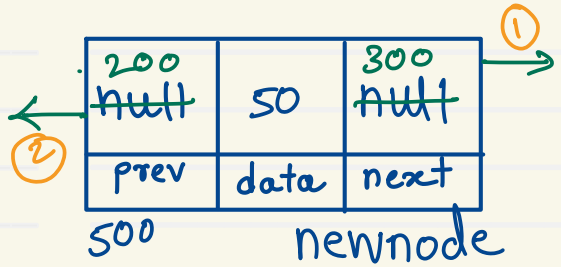
1. Create node
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add last node into prev of newnode
   c. add newnode into next of last node
   d. add newnode into prev of first node

$T(n) = O(1)$

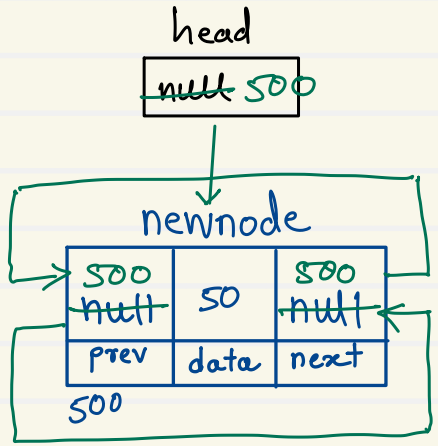
Doubly Circular Linked List - Add position



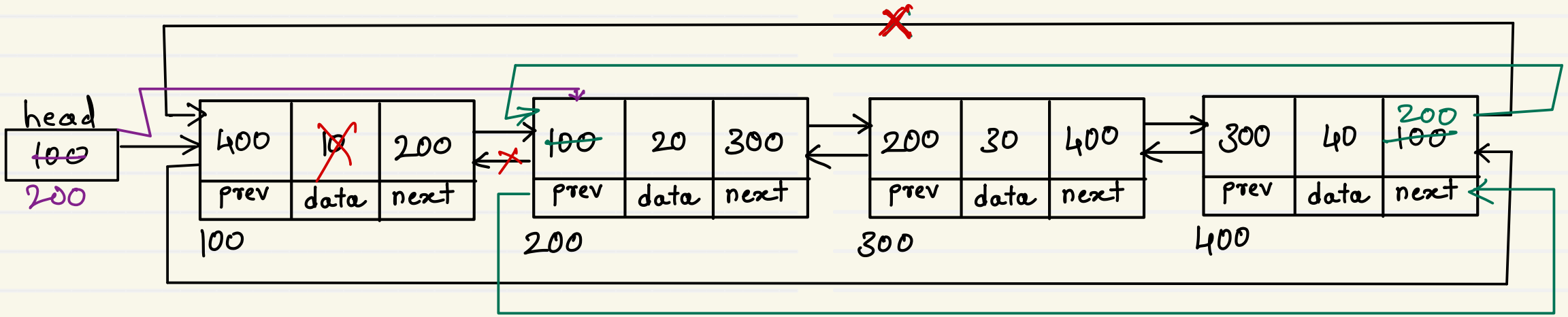
1. create node
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty.
   a. traverse till pos-1 node
   b. add pos node into next of newnode
   c. add pos-1 node into prev of newnode
   d. add newnode into next of pos-1 node
   e. add newnode into prev of pos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



$T(n) = O(1)$



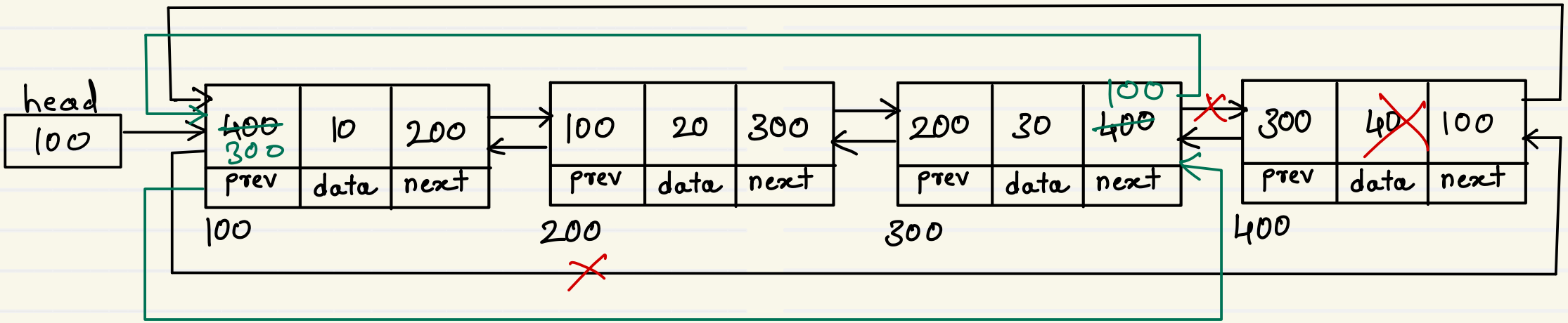
Doubly Circular Linked List - Delete first



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

$T(n) = O(1)$

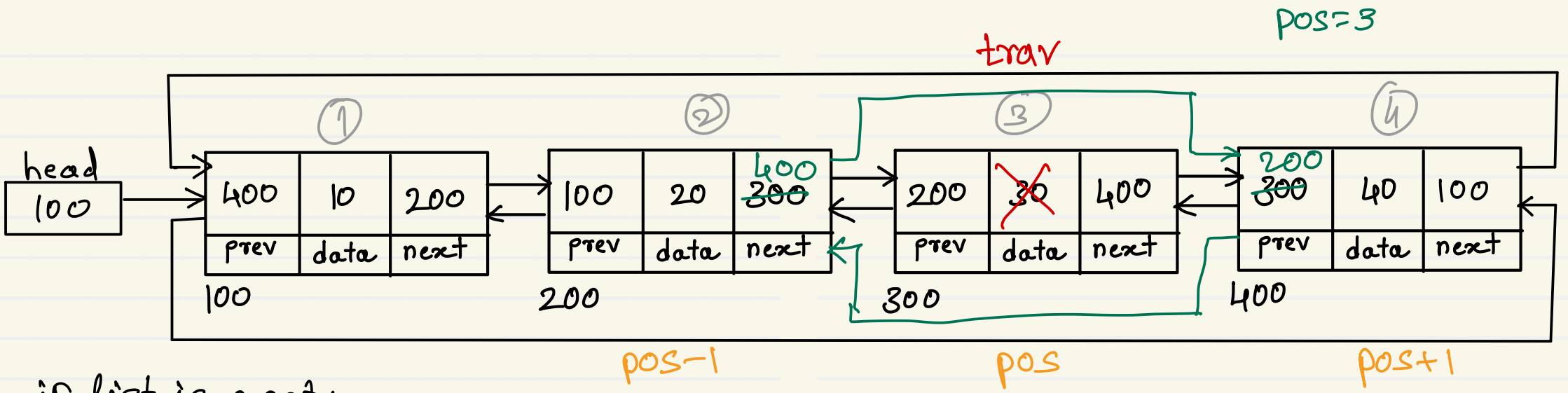
Doubly Circular Linked List - Delete last



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

$T(n) = O(n)$

Stack and Queue using linked list

Stack (LIFO)

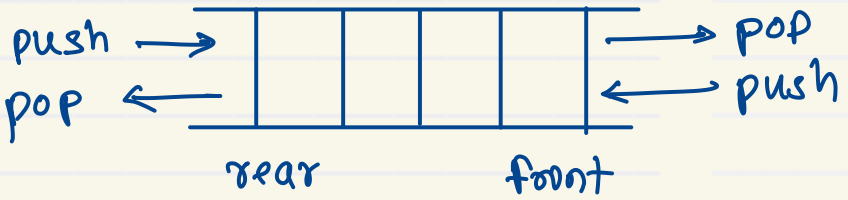
1. Add First
   Delete First
2. Add Last
   Delete Last

Queue (FIFO)

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

Deque (Double Ended Queue)

- insertion & deletion is allowed from both the ends (rear & front)



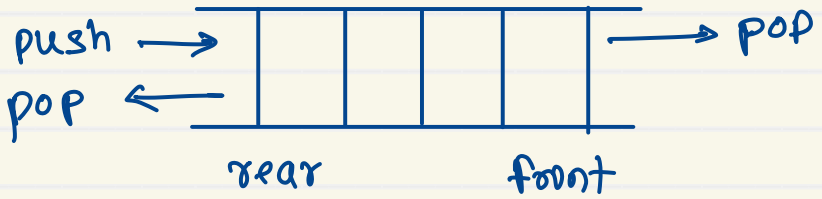
- to implement, circular array or linked list is used.
- in deque "FIFO" principle is not followed.
- stack & queue are efficiently implemented with deque.

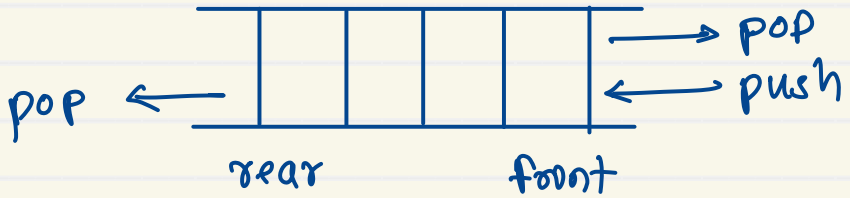
operation:

1. pushRear() → addFirst()
2. popRear() → deleteFirst()
3. peekRear()
4. pushFront() → addLast()
5. popFront() → deleteLast()
6. peekFront()
7. isEmpty
8. isFull (in case of array)

Types of deque

Input Restricted Deque





Output Restricted Deque