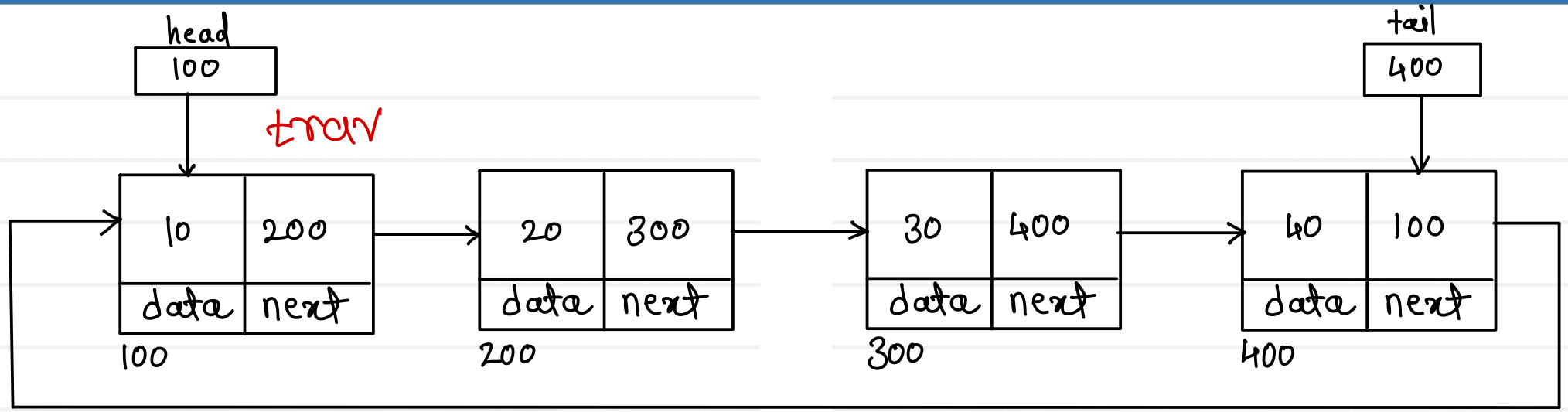
Sunbeam Institute of Information Technology
Pune and Karad

Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Singly Circular Linked List - Display

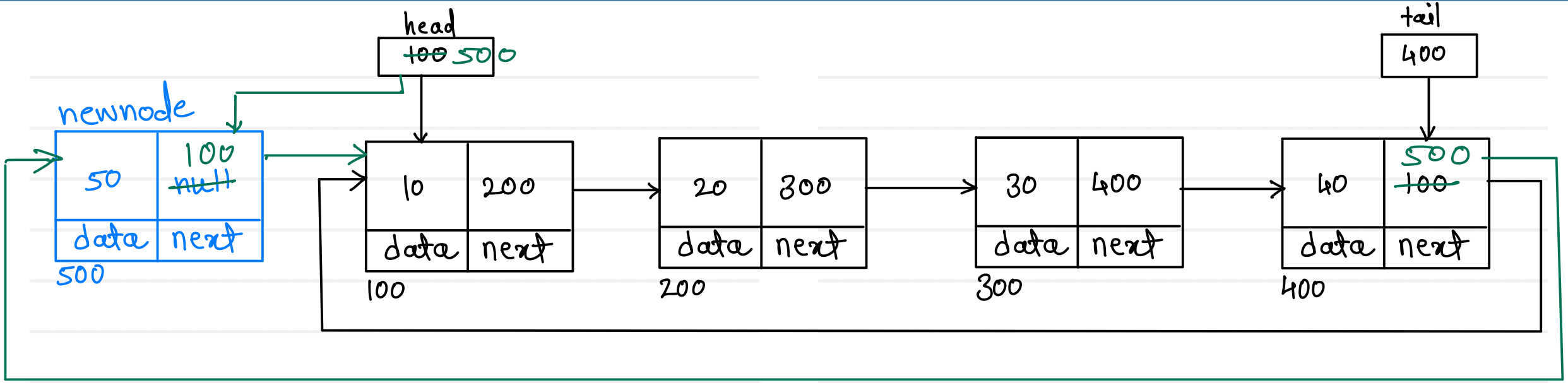


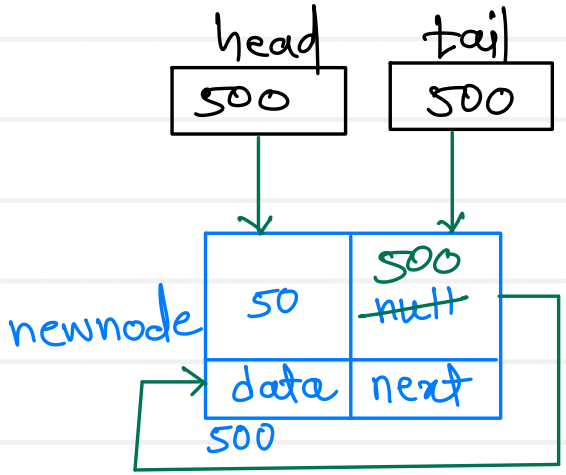
1. create trav & start at head.
2. print current node (trav.data)
3. go on next node (trav.next)
4. repeat above two step till last node

$T(n) = O(n)$

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

Singly Circular Linked List - Add first

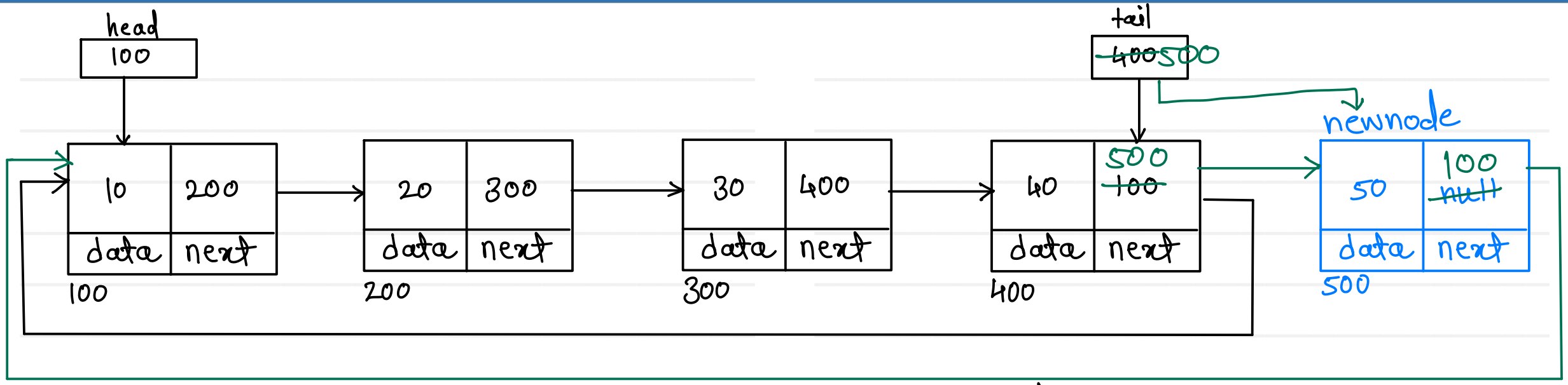


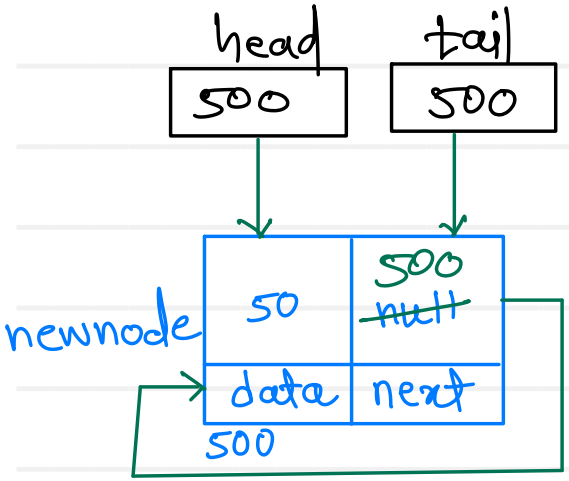


1. create node
2. if list is empty
   a. add newnode into head & tail
   b. make list circular
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into next of last node
   c. move head on newnode

$T(n) = O(1)$

Singly Circular Linked List - Add last

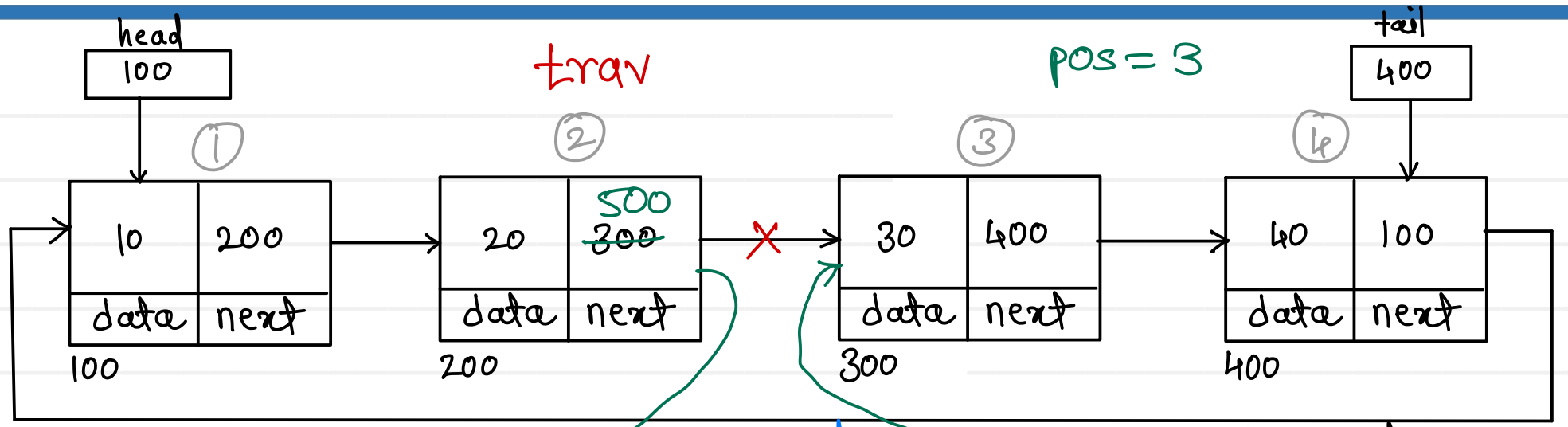




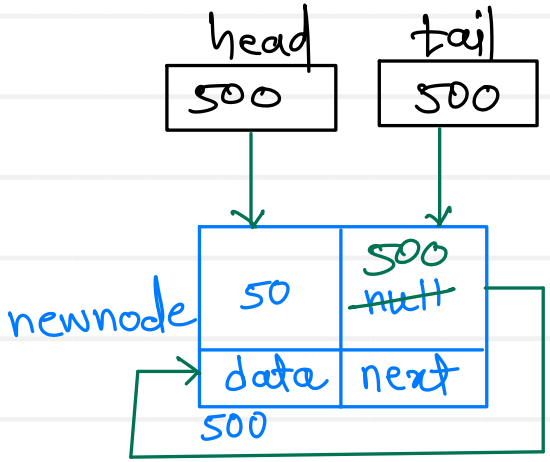
$T(n) = O(1)$

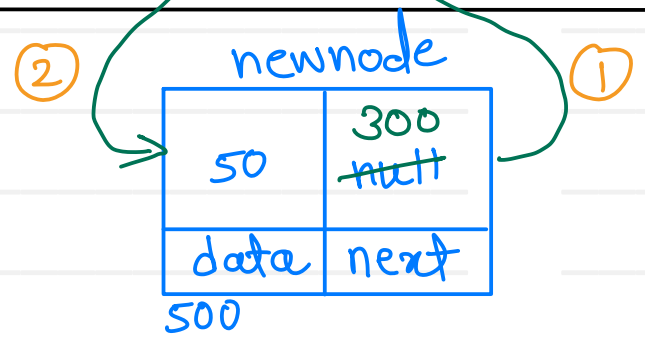
1. create node
2. if list is empty
   a. add newnode into head & tail
   b. make list circular.
3. if list is not empty.
   a. add first node into next of newnode
   b. add newnode into next of last node
   c. move tail on last node.

Singly Circular Linked List - Add position



make before break

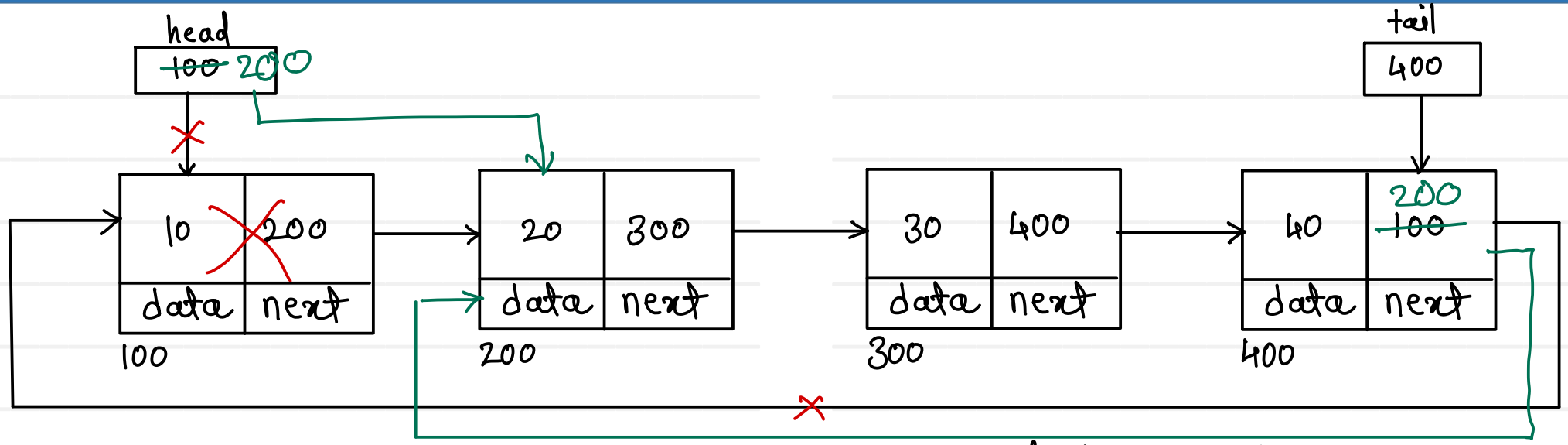


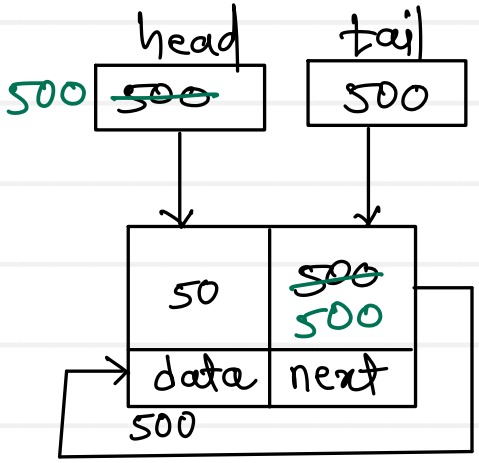


$T(n) = O(n)$

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

Singly Circular Linked List - Delete first



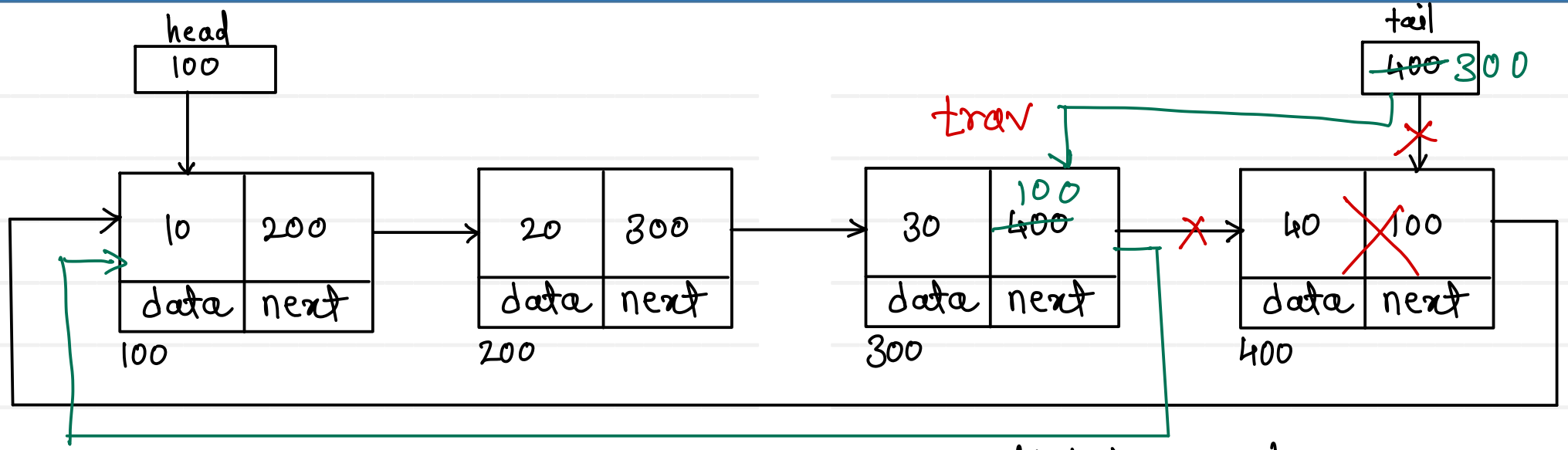


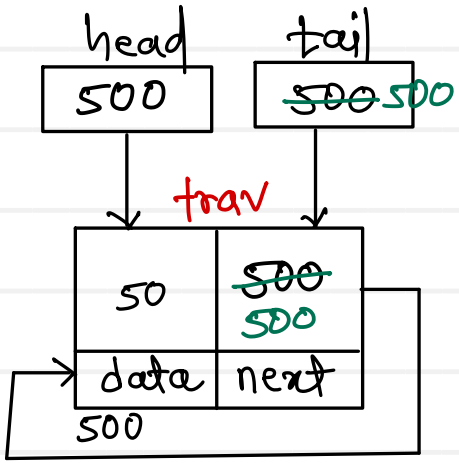

```
tail.next = head.next;
head = head.next
```

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



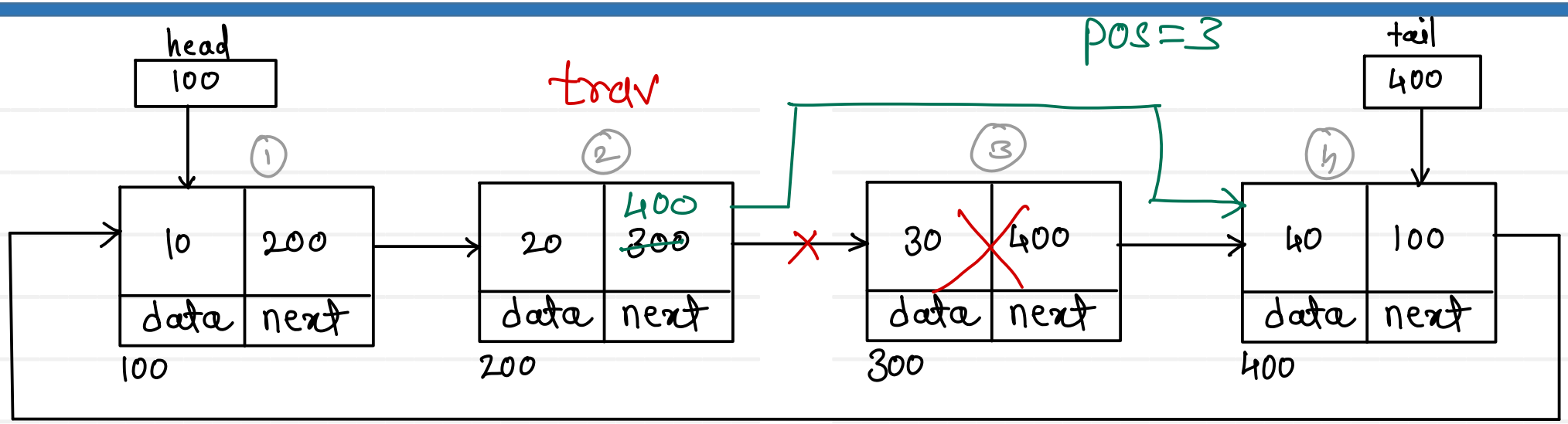



```
Node trav = head
while (trav.next.next != head)
    trav = trav.next;
trav.next = head;
tail = trav.
```

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



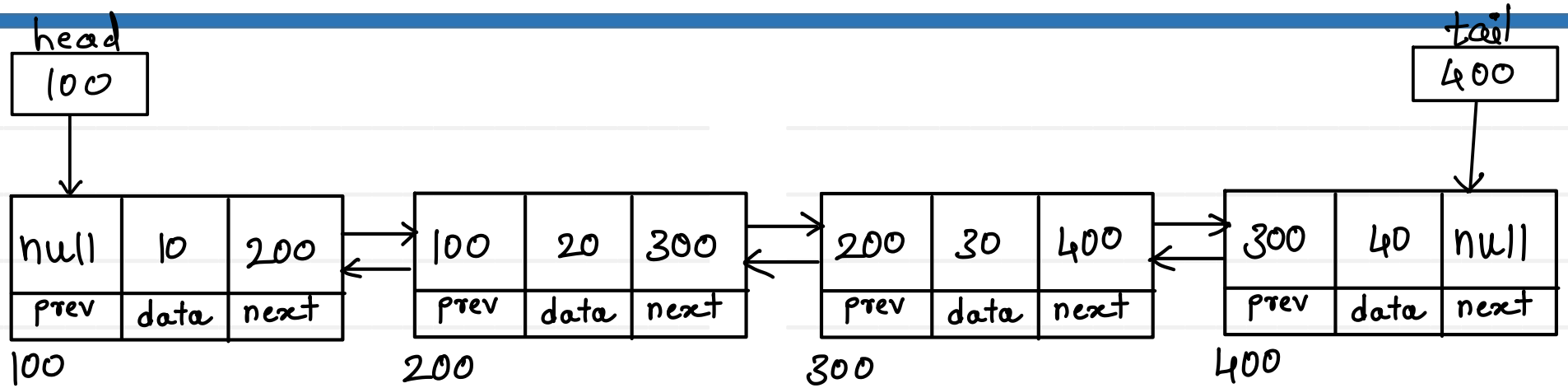
valid positions:
1 to size
Invalid positions:
pos < 1 or pos > size

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

pos=3

trav	i	i < pos-1
100	1	T
200	2	X

Doubly Linear Linked List - Display

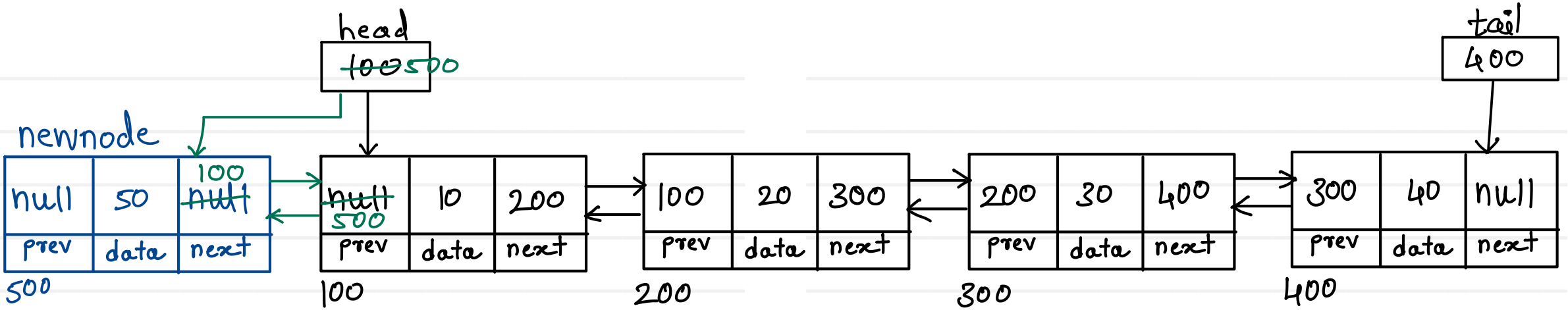


1) create trav & start at head
2) print current node
3) go on next node
4) repeat step 2 & 3 till last node

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

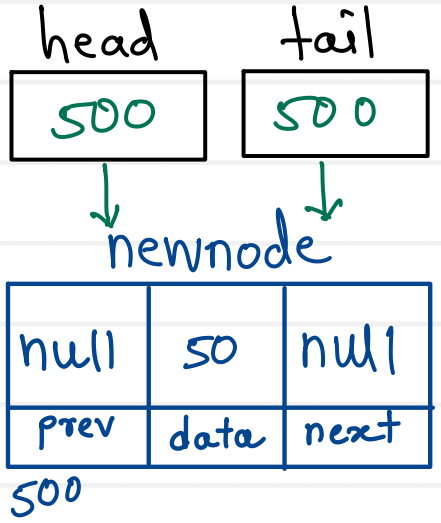
$T(n) = O(n)$

Doubly Linear Linked List - Add first

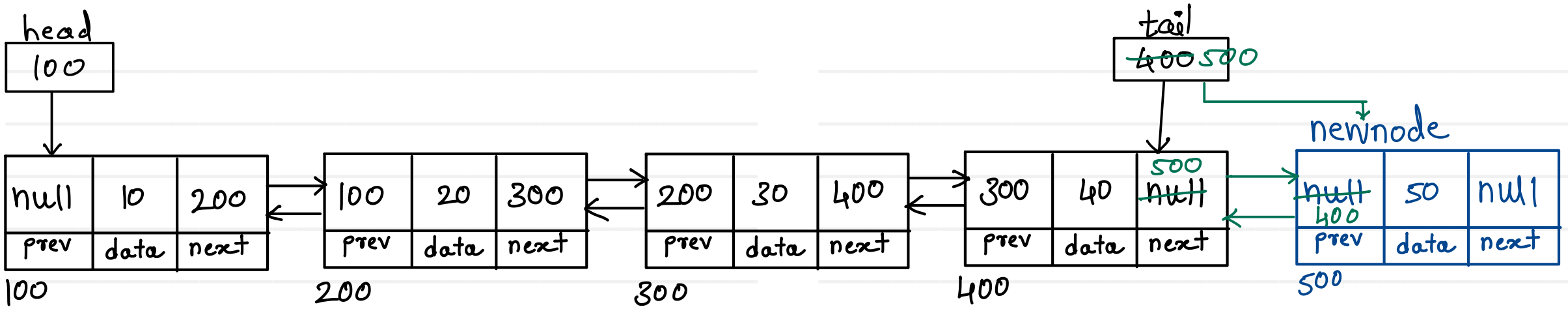


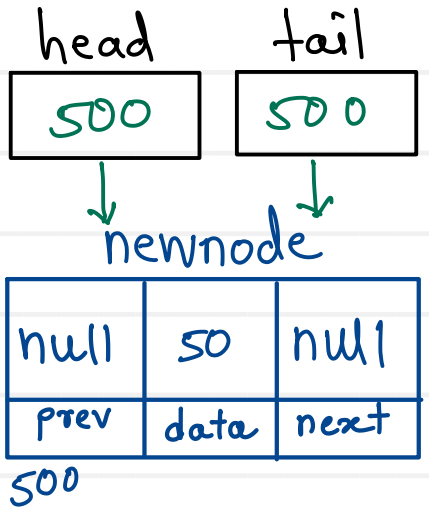
1. create node
2. if list is empty
   head = tail = newnode;
3. if list is not empty
   a. add first node into next of newnode
   b. add newnode into prev of first node
   c. move head on newnode.

$T(n) = O(1)$



Doubly Linear Linked List - Add last

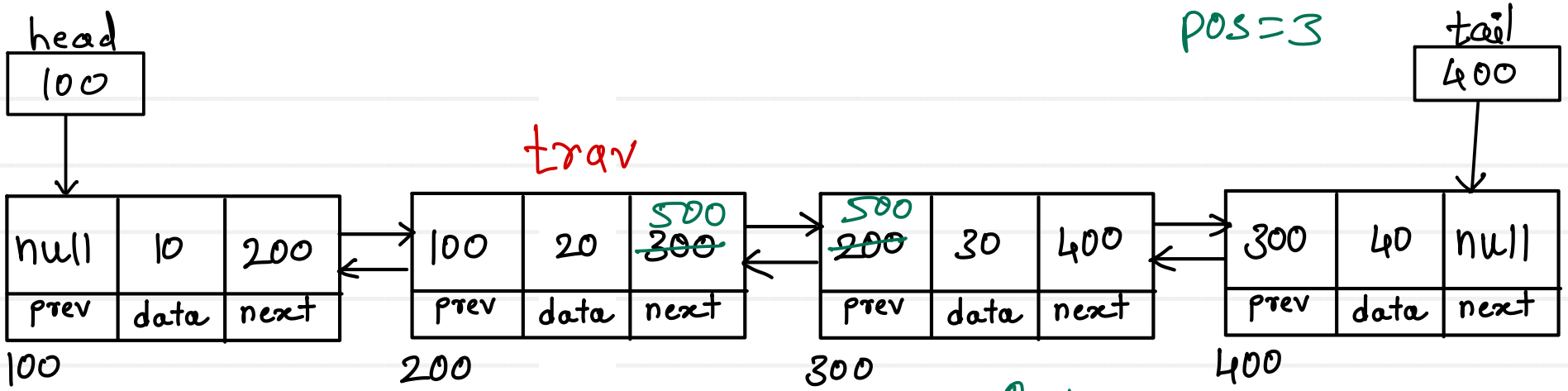




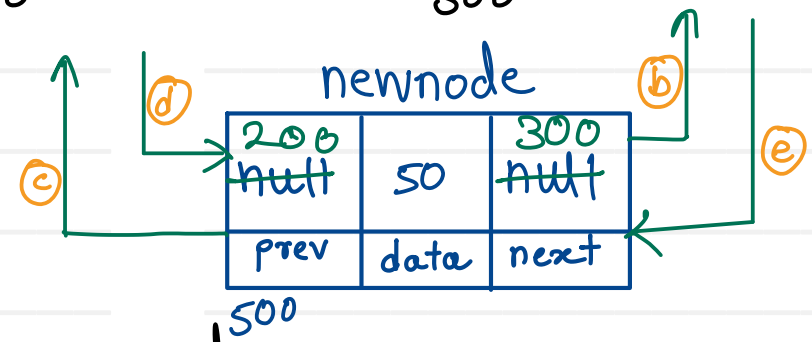
1. Create node
2. if list is empty
   head = tail = newnode;
3. if list is not empty
   a. add last node into prev of newnode
   b. add newnode into next of last node
   c. move tail on newnode

$T(n) = O(1)$

Doubly Linear Linked List - Add position

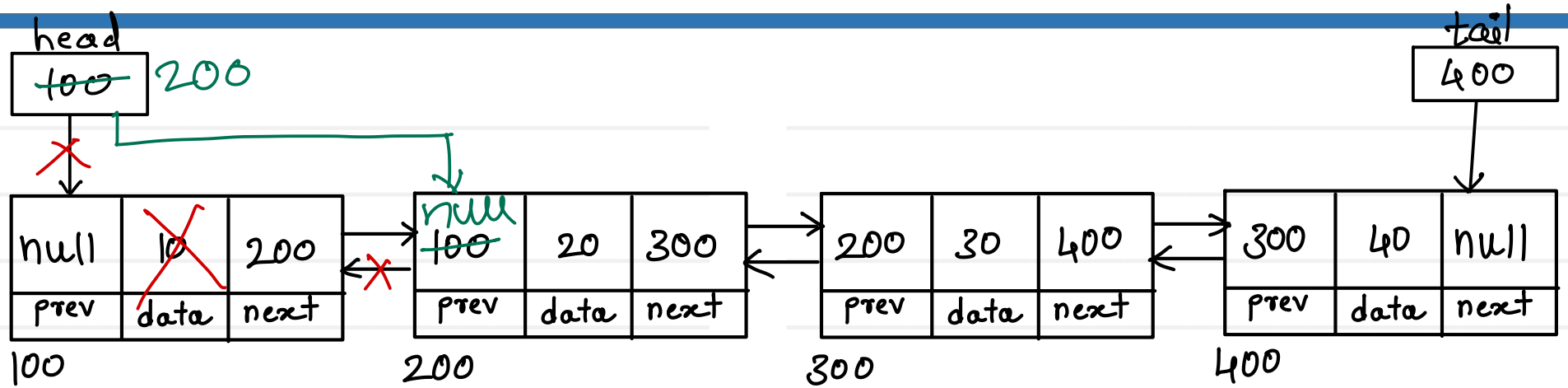


1) Create node
2) if list is empty
   head = tail = newnode;
3) if list is not empty
   a. traverse till pos-1
   b. add pos node into next of newnode
   c. add pos-1 node into prev of newnode
   d. add newnode into next of pos-1 node
   e. add newnode into prev of pos node.

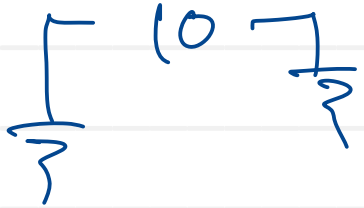


$T(n) = O(n)$

Doubly Linear Linked List - Delete first



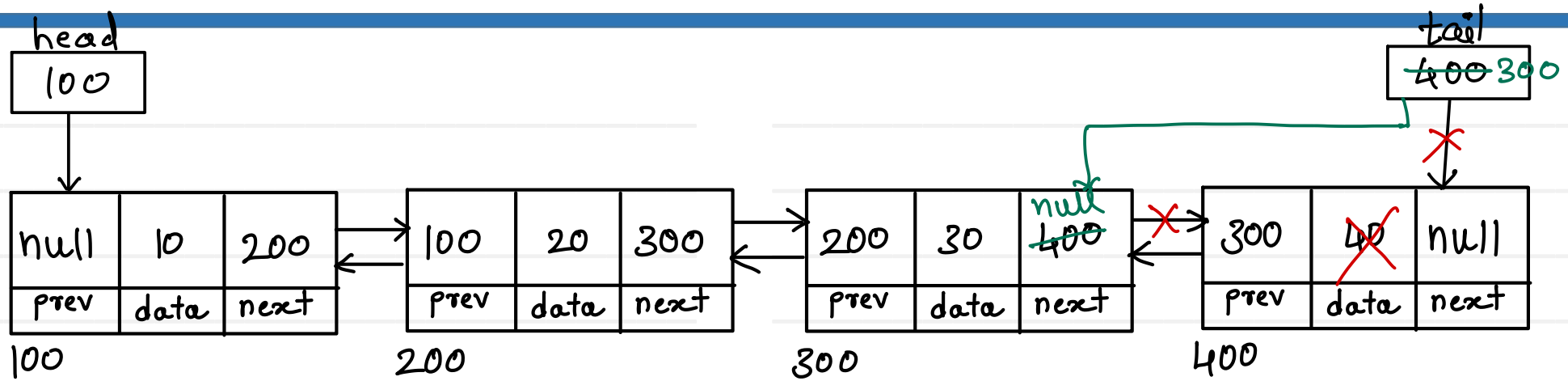
head = head.next
head.prev = null



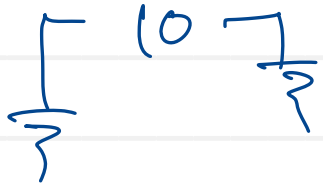
T(n) = O(n)

1. if list is 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

Doubly Linear Linked List - Delete last



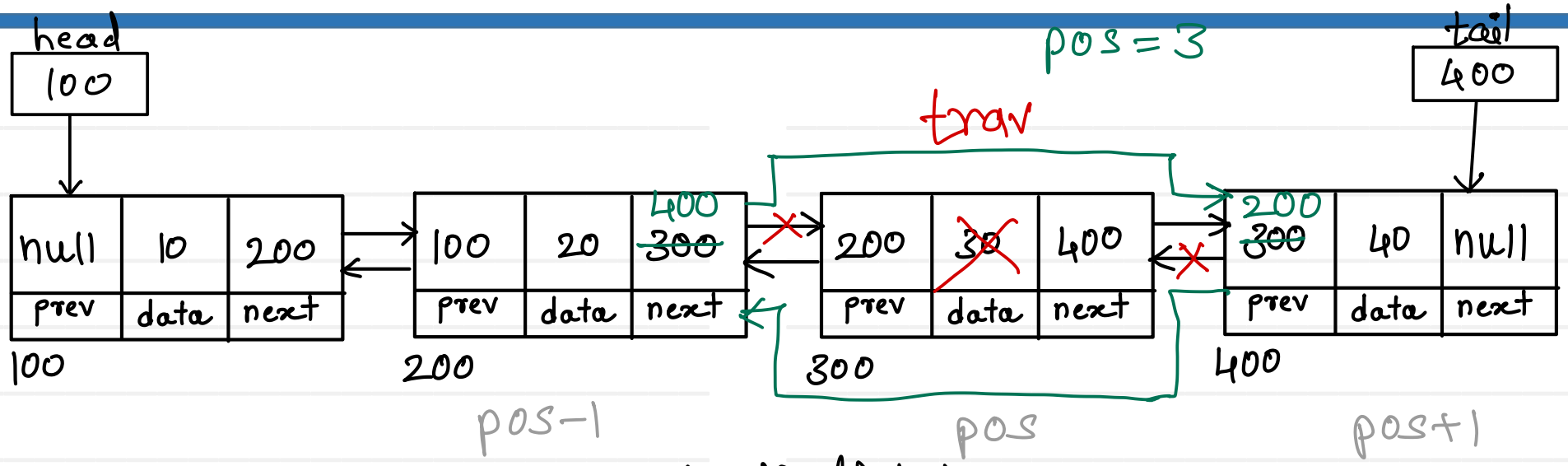
head head = head.next
      head.prev = null



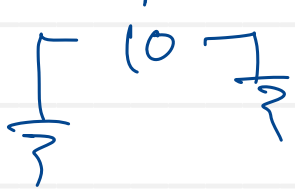
T(n) = O(1)

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

Doubly Linear Linked List - Delete position



head head = head.next
      head.prev = null



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

$T(n) = O(n)$

Linked list - Applications

- linked list is a dynamic data structure because it can grow or shrink at runtime.
- Due to this dynamic nature, linked list is used to implement other data structures like
  1. Stack
  2. Queue
  3. Hash table
  4. Graph

Stack

LIFO

1. Add first
   delete first
2. Add last
   delete last

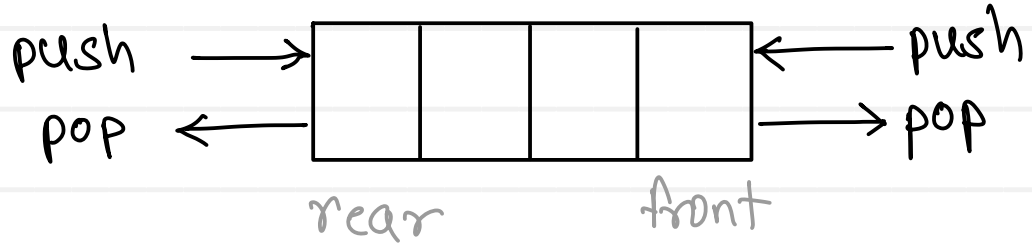
Queue

FIFO

1. Add first
   delete last
2. Add last
   delete first

Deque

(Double Ended Queue)



1. Input restricted deque
   - one input end is closed
2. Output restricted deque
   - one output end is closed

Array Vs Linked list

Array

- Array space inside memory is continuous
- Array can not grow or shrink at runtime
- Random access of elements is allowed
- Insert or delete, needs shifting of array elements
- Array needs less space

Linked list

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

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com