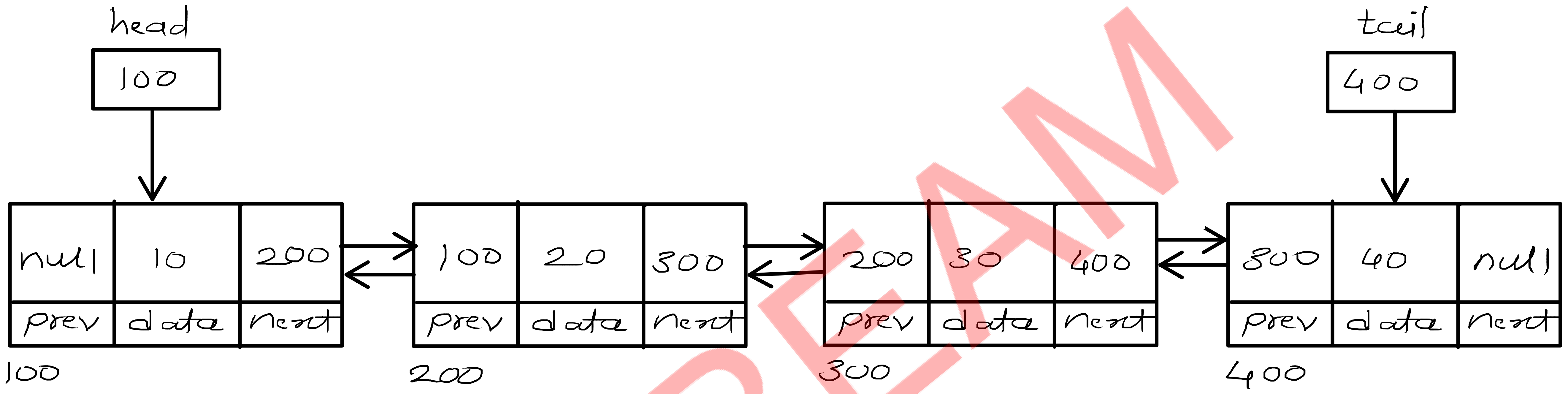
Doubly Linear Linked List - Display

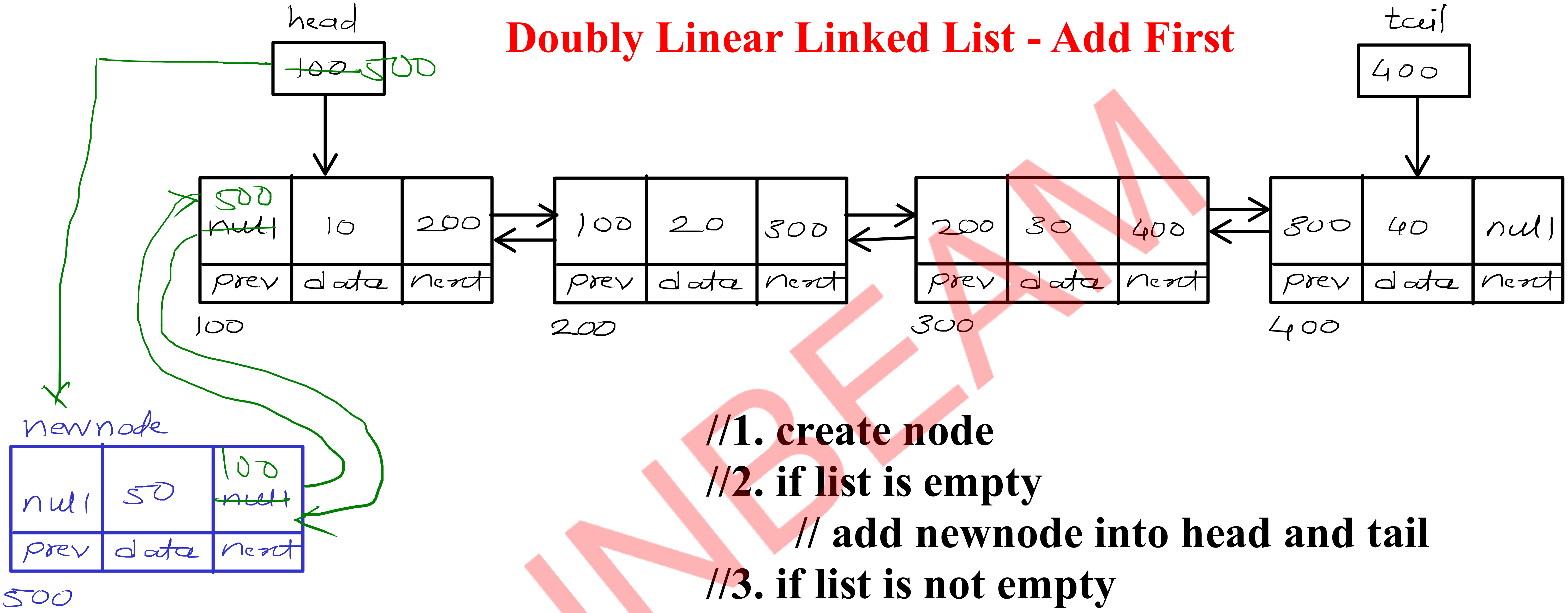


// forward traversal
//1. create trav and start at head
//2. visit/print data of current node
//3. go on next node
//4. repeat step 2 and 3 till last node

// reverse traversal
//1. create trav and start at tail
//2. visit/print data of current node
//3. go on prev node
//4. repeat step 2 and 3 till first node

$T(n) = O(n)$

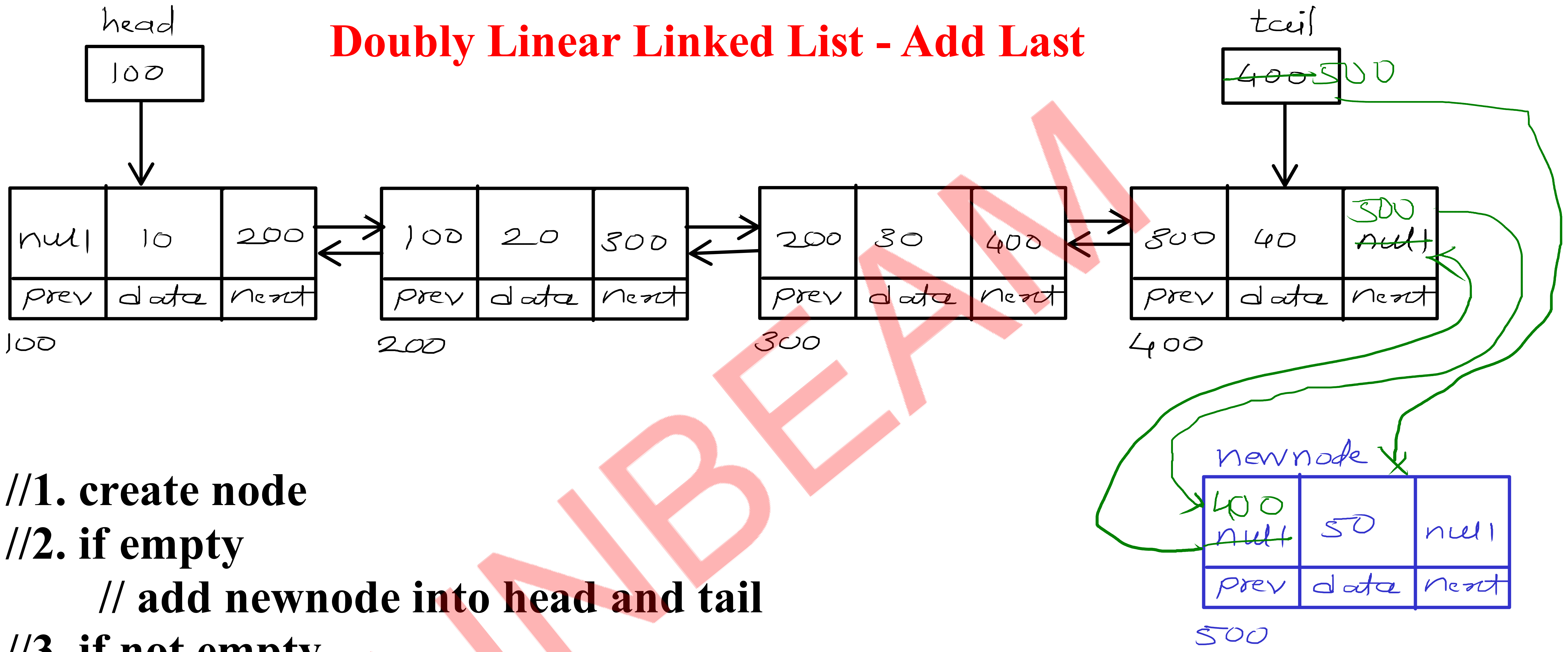
Doubly Linear Linked List - Add First



```
//1. create node
//2. if list is empty
        // 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
```

T(n) = O(1)

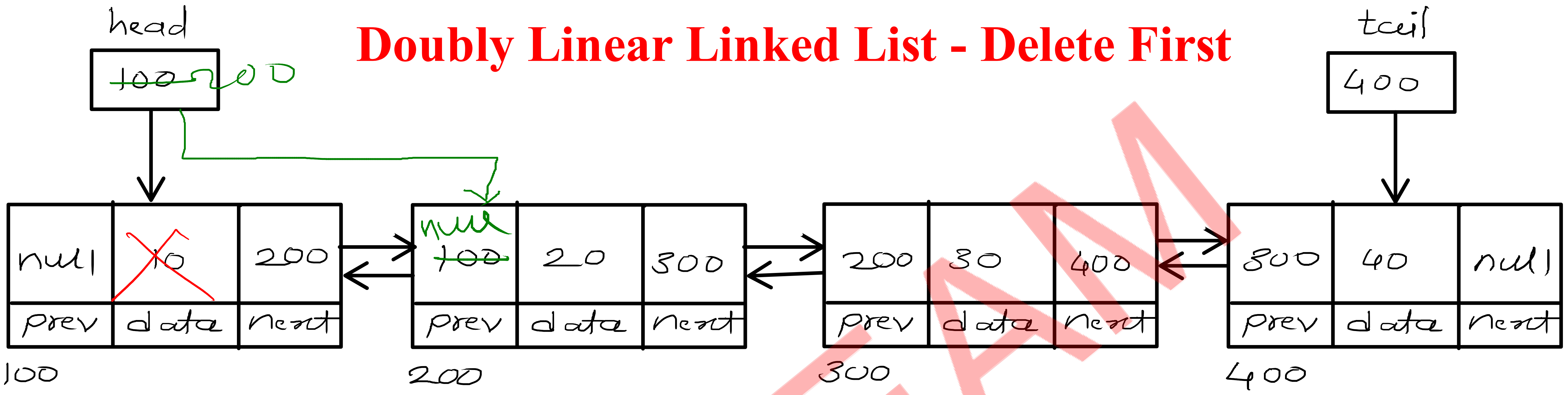
Doubly Linear Linked List - Add Last



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

$T(n) = O(1)$

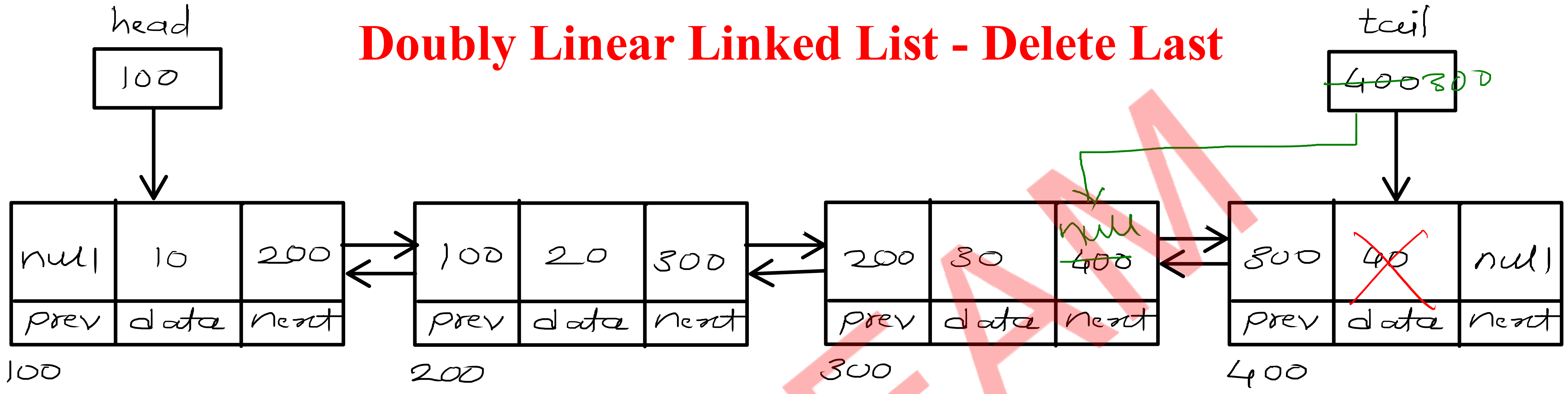
Doubly Linear Linked List - Delete First



```
//1. if empty
        return;
//2. if single node
        head = tail = null;
//3. if multiple nodes
        //a. move head on second node
        //b. make prev of second node null
```

$T(n) = O(1)$

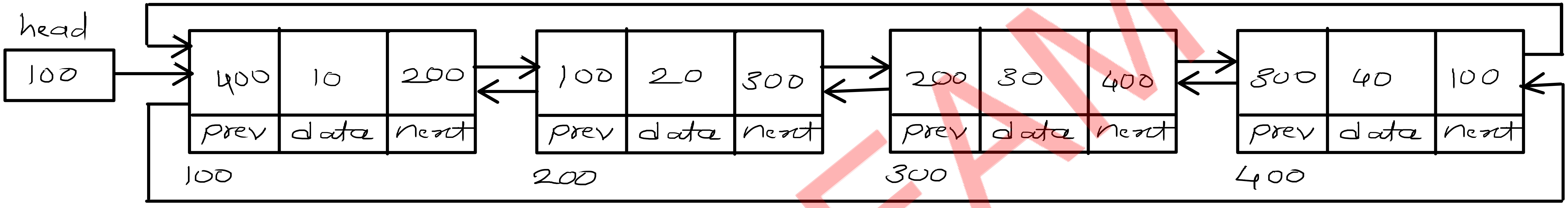
Doubly Linear Linked List - Delete Last



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

$T(n) = O(1)$

Doubly Circular Linked List - Display

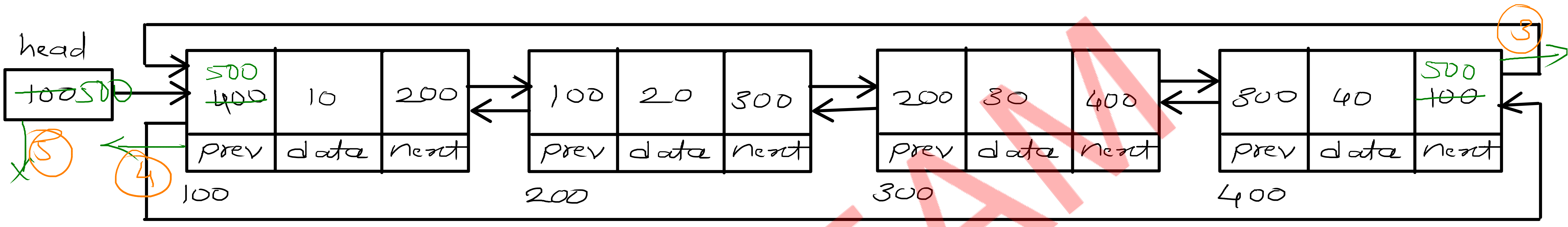


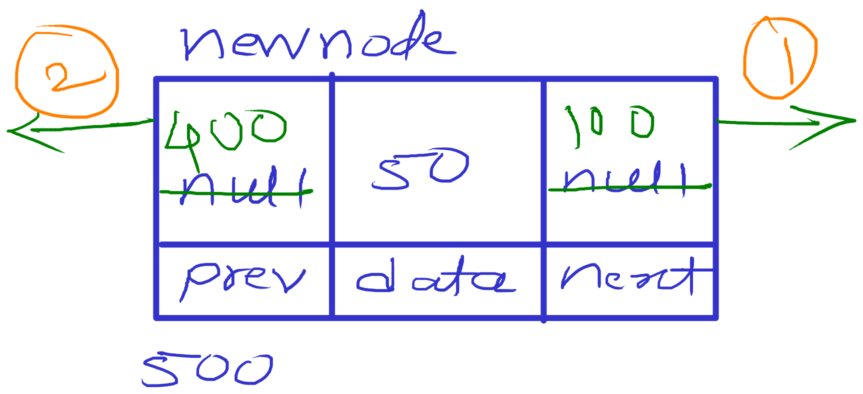
//1. create a trav and start at first node
//2. print data of current node (trav.data)
//3. go on next node
//4. repeat step 2 and 3 till last node

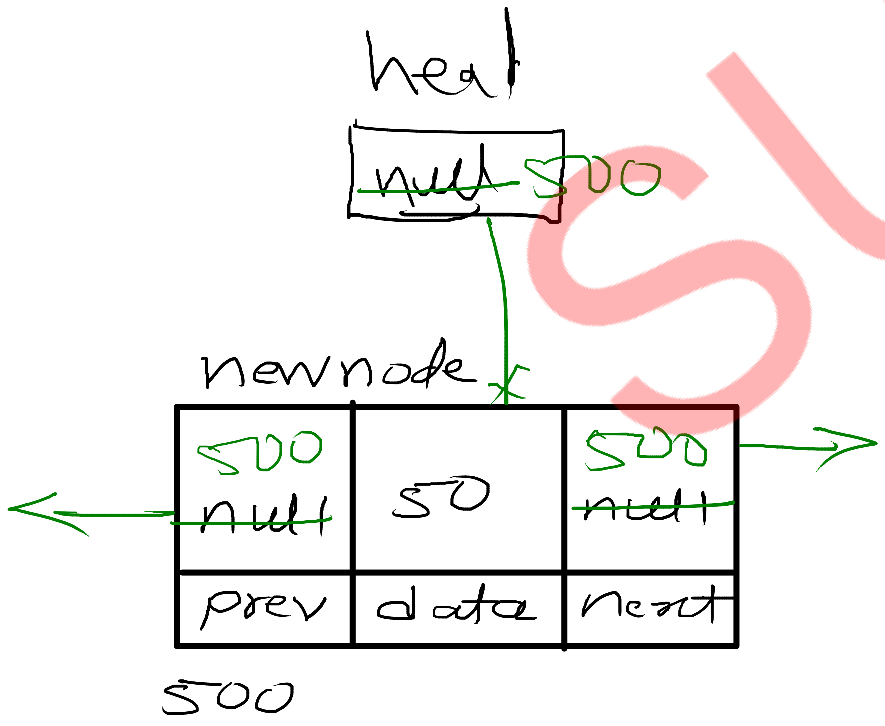
//1. create a trav and start at last node
//2. print data of current node (trav.data)
//3. go on prev node
//4. repeat step 2 and 3 till first node

$T(n) = O(n)$

Doubly Circular Linked List - Add first



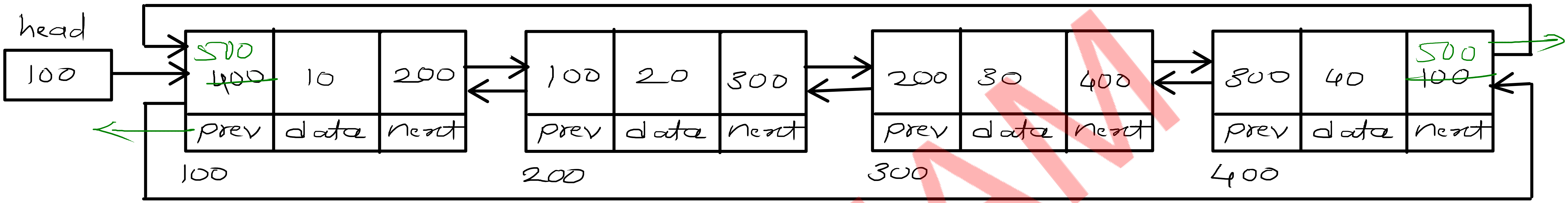




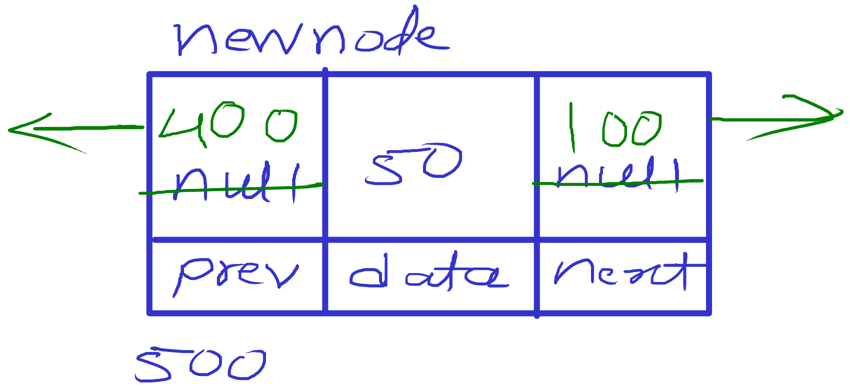
```
//1. create node
//2. if empty
        //a. add newnode into head
        //b. make list circular
//3. if not empty
        //a. add first node into next of newnode
        //b. add last node into prev of newnode
        //c. add newnode into next of last node
        //d. add newnode into prev of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

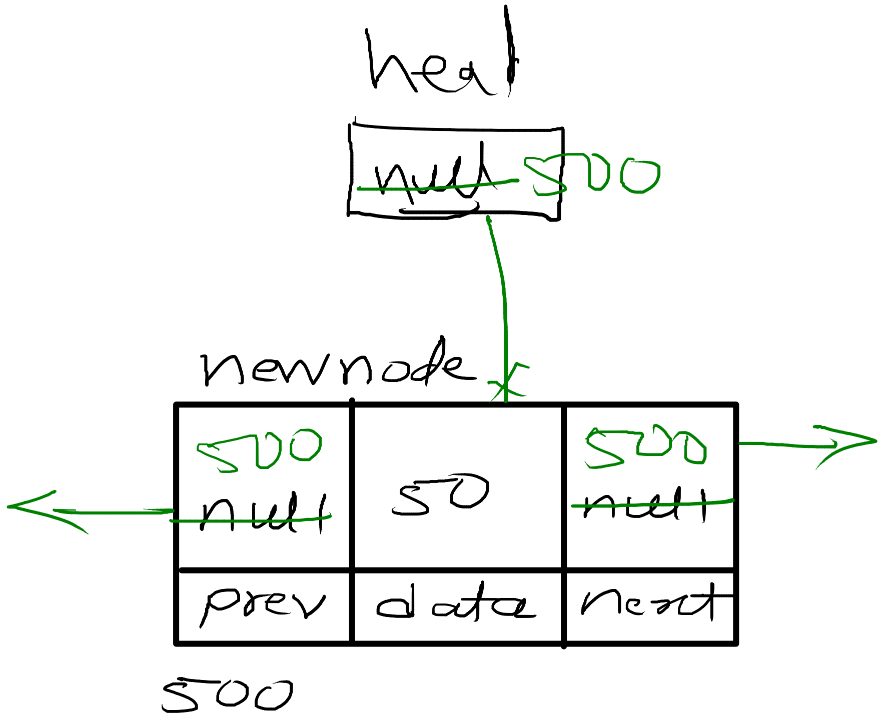
$T(n) = O(1)$

Doubly Circular Linked List - Add Last



```
//1. create node
//2. if empty
        //a. add newnode into head
        //b. make list circular
//3. if 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
```

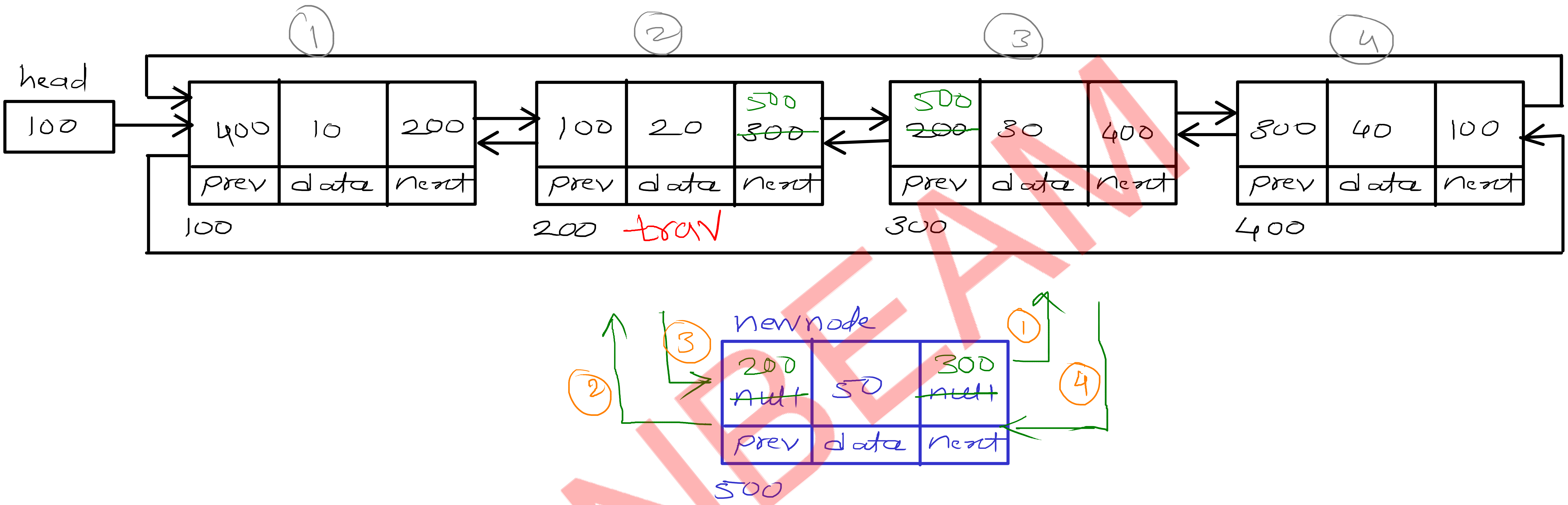




$T(n) = O(1)$

Doubly Circular Linked List - Add position

pos=3

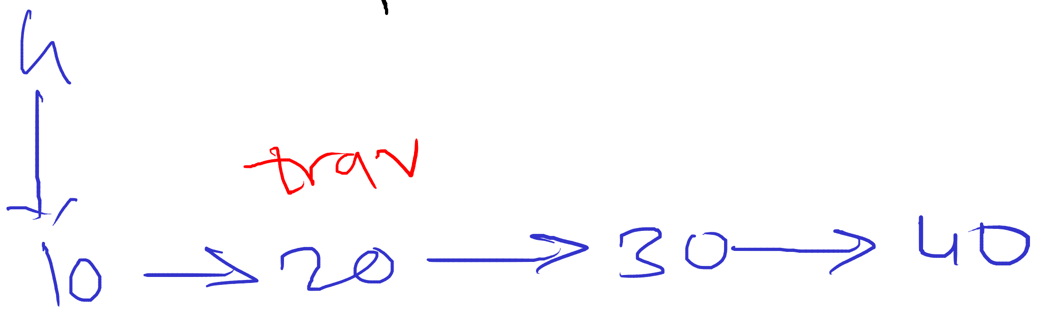


```
//1. if empty
    //a. add newnode into head
    //b. make list circular
//2. if not empty
    //a. traverse till pos-1 node
    //b. add pos node into next of newnode
    //c. add pos-1 node into prev of newnode
    //d. add newnode into next of pos-1node
    //e. add newnode into prev of pos node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

T(n) = O(n)

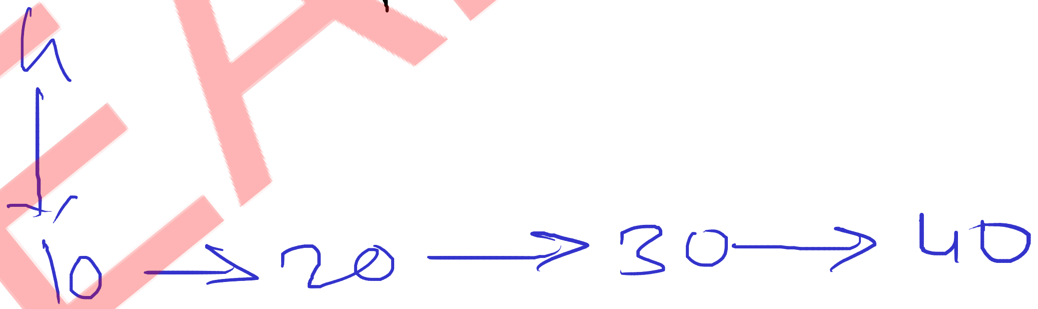
```
trav = head;
for(int i=1 ; i< pos-1 && trav.next != head ; i++)
                    trav=trav.next;
```

pos=3



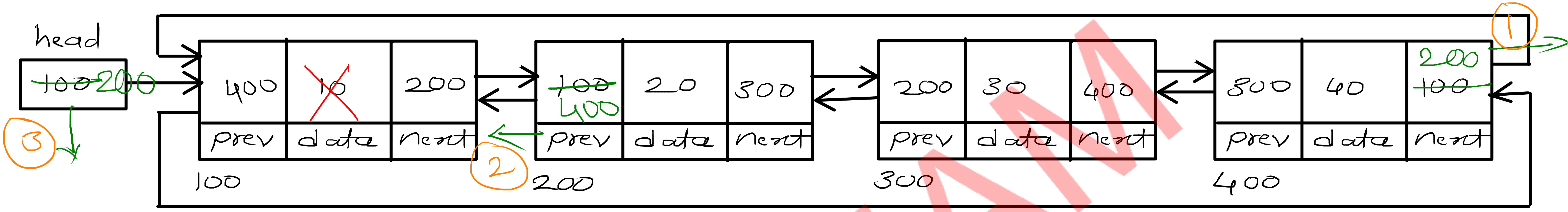
trav	i	i<2
100	1	T
200	2	F

pos=6



trav	i	i<5
100	1	T
200	2	T
300	3	T
400	4	T
100		

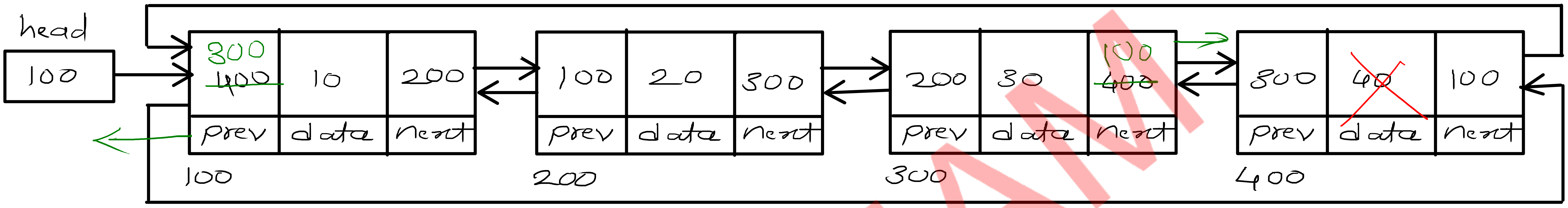
Doubly Circular Linked List - Delete First



```
//1. if empty
        return;
//2. if single node
        head = null;
//3. if 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)

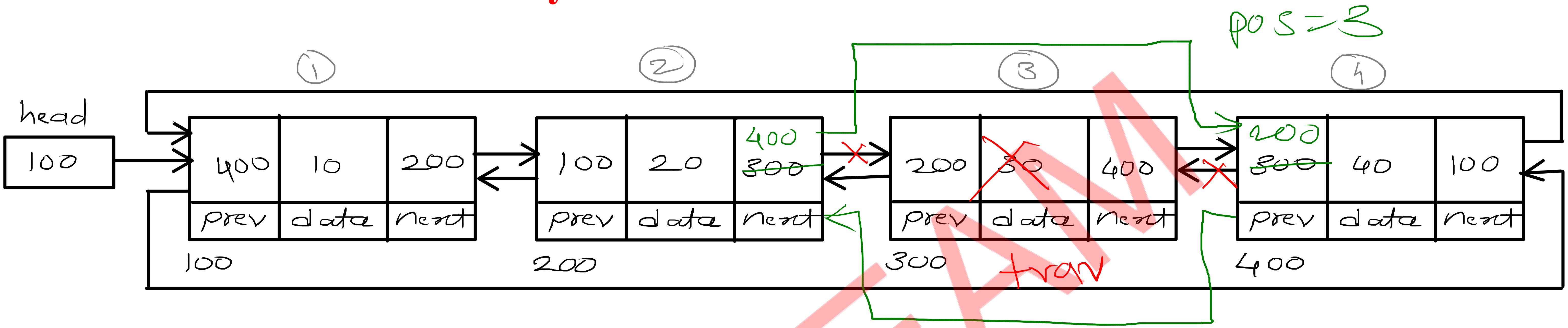
Doubly Circular Linked List - Delete Last



```
//1. if empty
        return;
//2. if single node
        head = null;
//3. if 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)

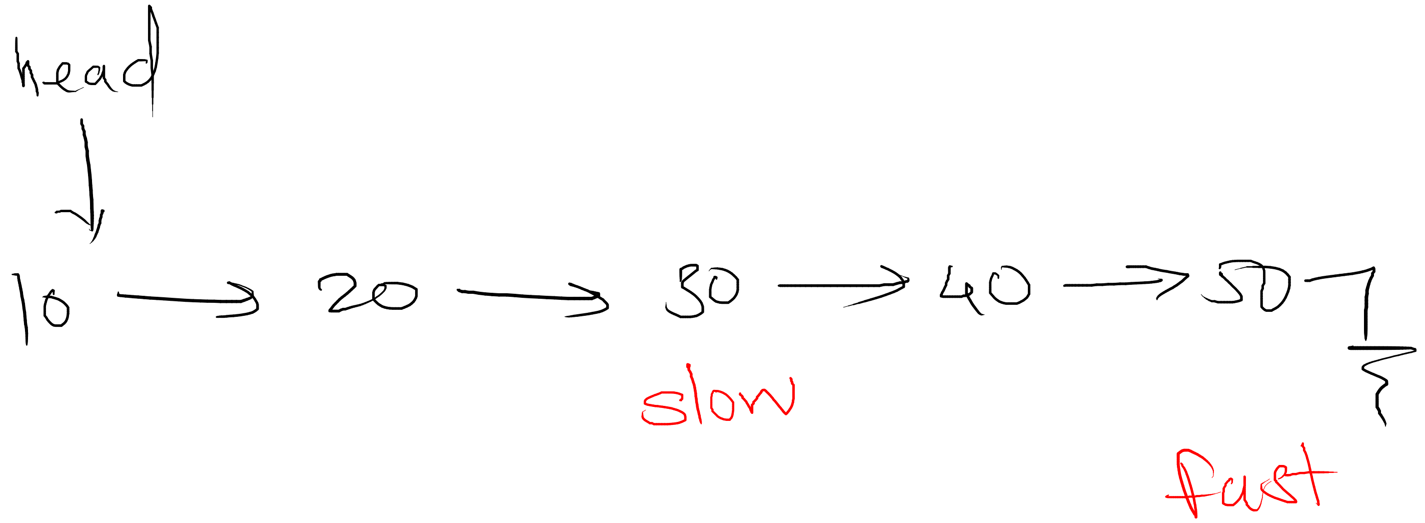
Doubly Circular Linked List - Delete Position

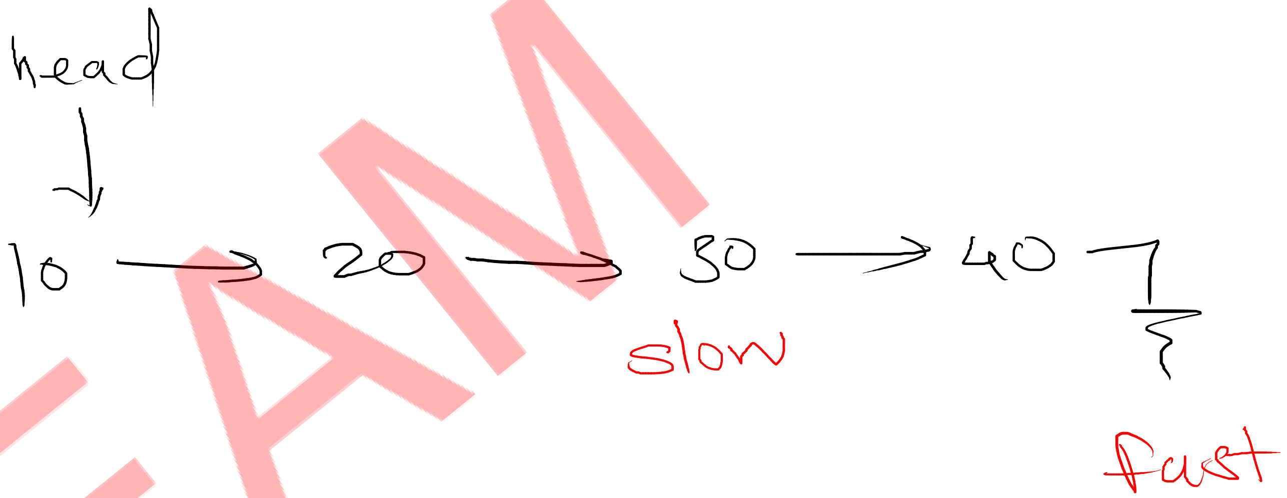


```
//1. if empty
        return;
//2. if single node
        head = null;
//3. if multiple nodes
        //a. traverse till pos node
        //b. add pos+1 node into next of pos-1 node
        //c. add pos-1 node into prev of pos+1 node
```

$T(n) = O(n)$

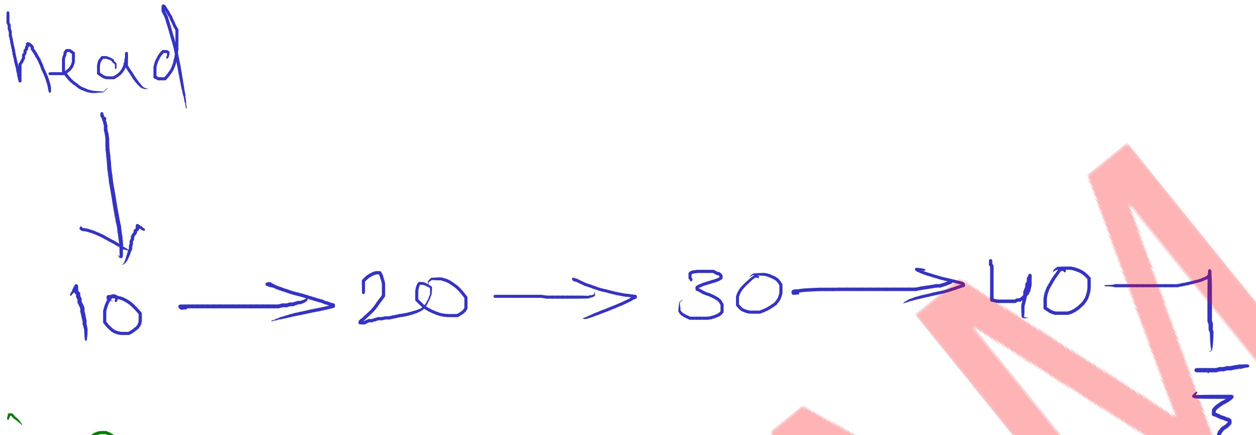
Singly Linear Linked List - Find mid





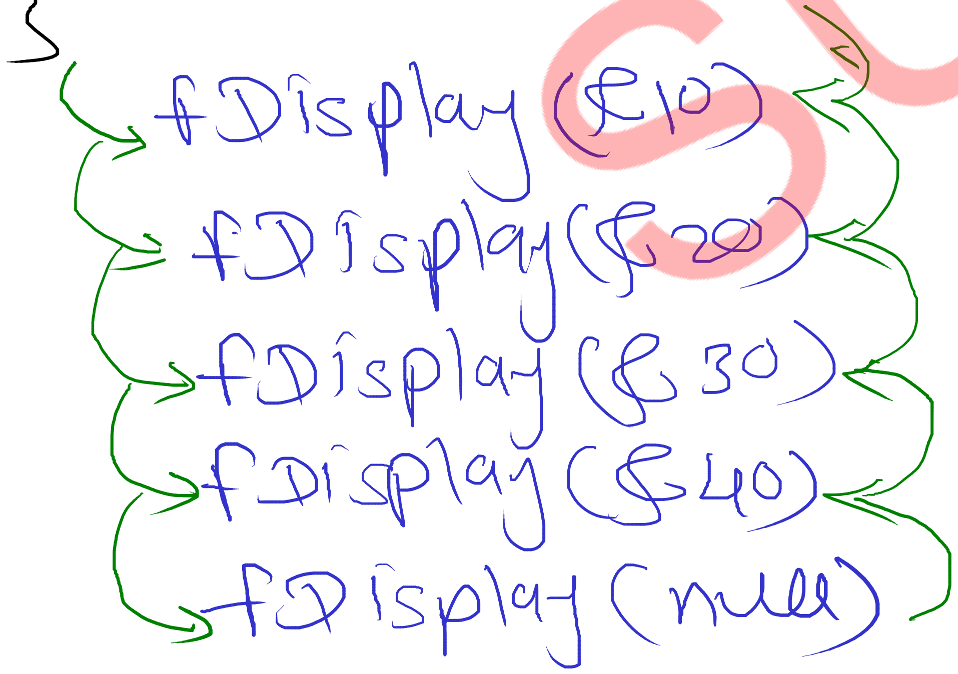
```
Node findMid( ){
        Node fast = head;
        Node slow = head;
        while(fast != null && fast.next != null){
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow;
}
```

Singly Linear Linked List - Reverse Display



Tail Recursion

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

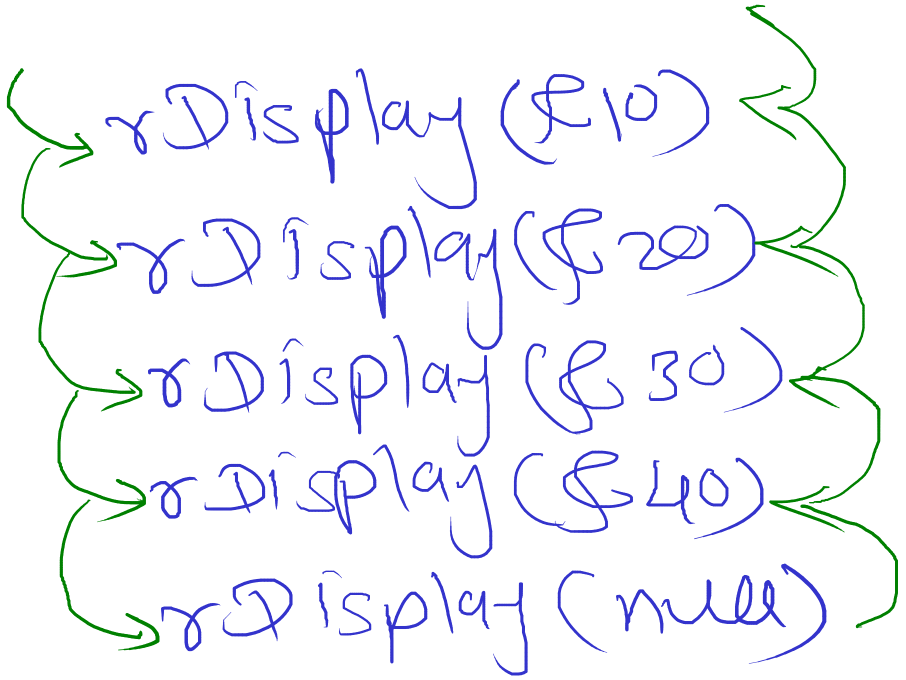


fDisplay(&10)
fDisplay(&20)
fDisplay(&30)
fDisplay(&40)
fDisplay(null)

10, 20, 30, 40

Non-tail Recursion

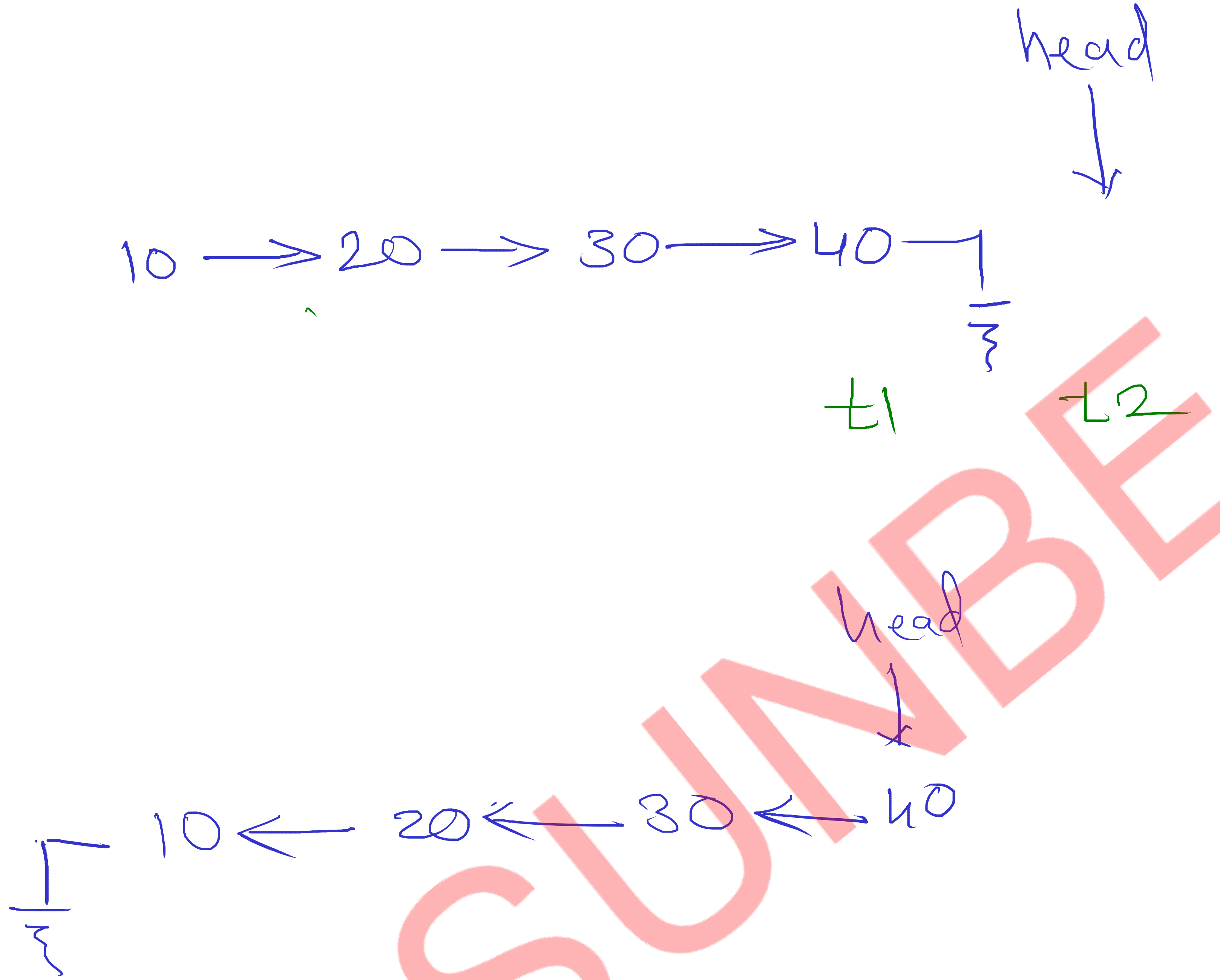
```
void rDisplay(Node trav)
    if(trav == null)
        return;
    rDisplay(trav.next);
    sysout(trav.data)
}
```



rDisplay(&10)
rDisplay(&20)
rDisplay(&30)
rDisplay(&40)
rDisplay(null)

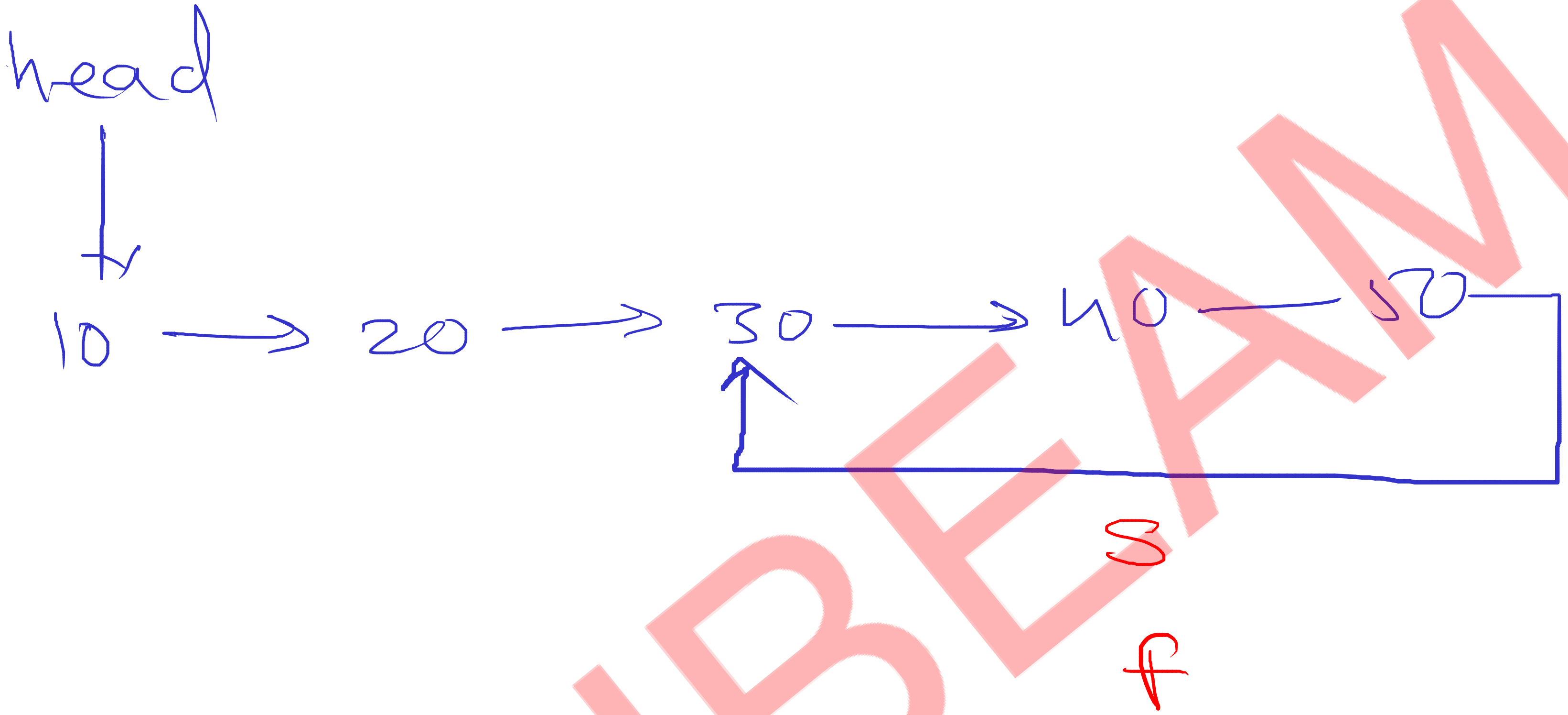
40, 30, 20, 10

Singly Linear Linked List - Reverse List



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

Detect loop inside linked list



```
Node fast = head;
Node slow = head;
while(fast != null && fast.next != null){
    fast = fast.next.next;
    slow = slow.next;
    if(slow == fast)
        return true; ---> loop is detected
}
```

Linked List Applications

- linked list is a dynamic data structure (grow or shrink at any time)
- due to this dynamic nature, linked list is used to implement other data structures like:

1. stack
2. queue
3. hash tables
4. graph

Stack

(LIFO)

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

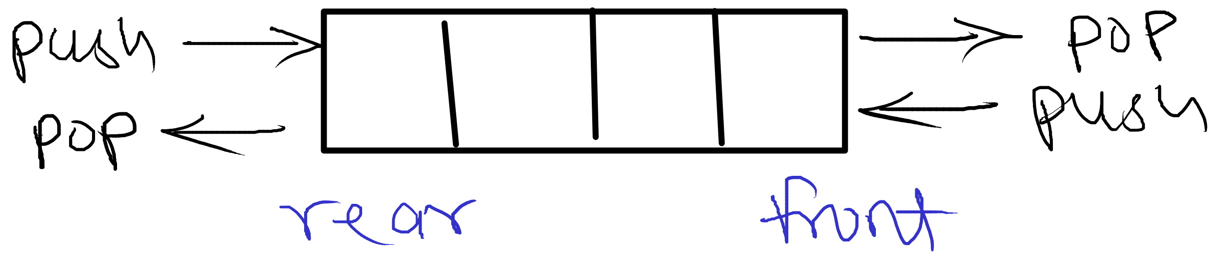
Queue

(FIFO)

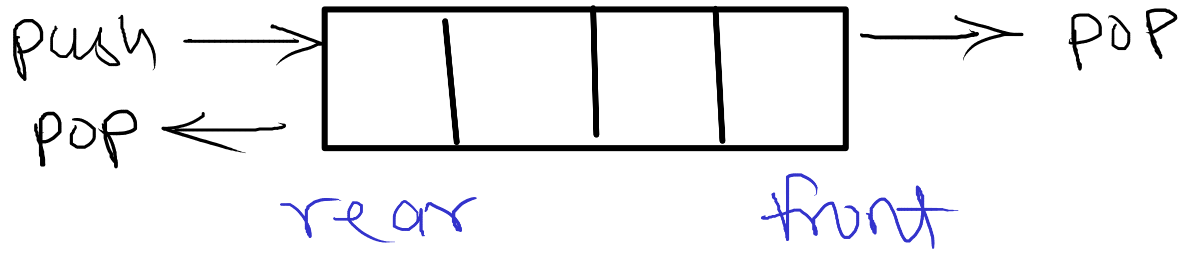
1. Add First
Delete Last
2. Add Last
Delete First

Deque

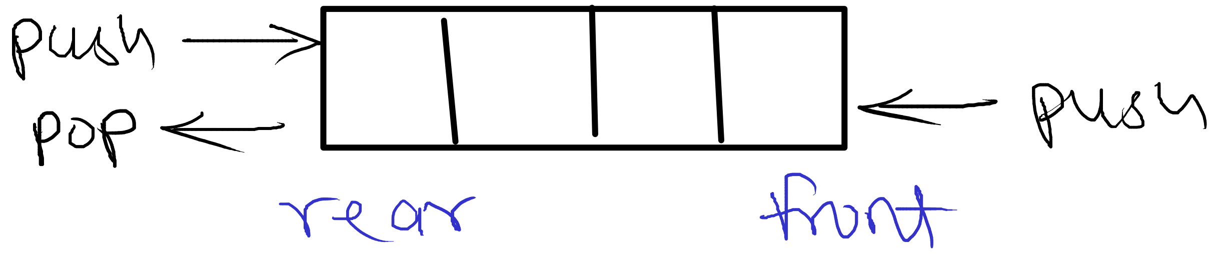
(Double Ended Queue)



Input Restricted Deque



Output Restricted Deque

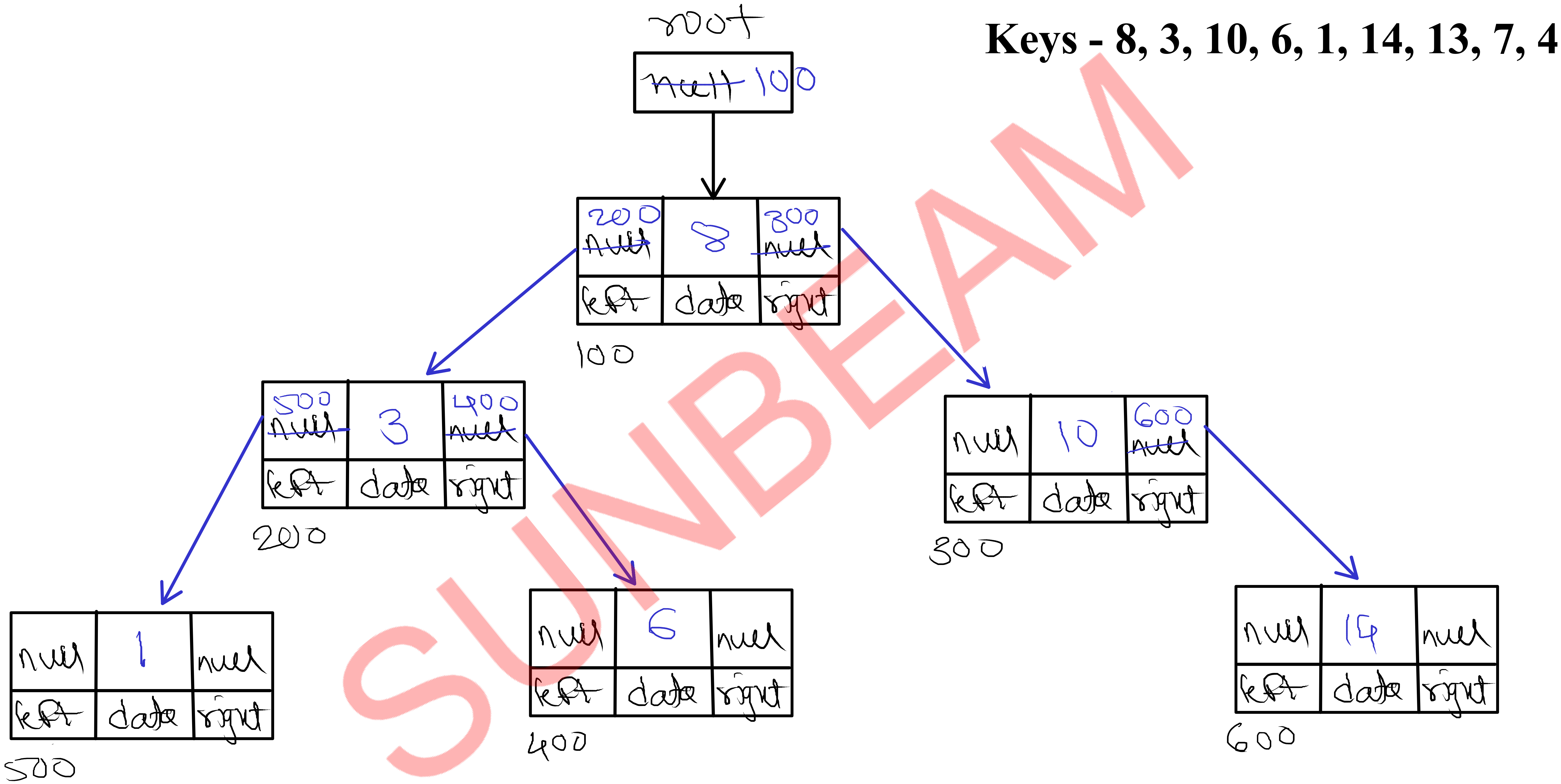


Array Vs Linked List

Array	Linked List
1. Array space in memory is contiguous	1. Linked list space in memory is not contiguous
2. Array can not grow or shrink at runtime	2. Linked list can grow or shrink at runtime
3. Random access of elements is allowed	3. Random access of elements is not allowed(sequential)
4. Insert or Delete, needs shifting of array elements	4. Insert or Delete, do not need shifting of nodes
5. Array needs less space	5. Linked lists need more space

BST - Add Node

Keys - 8, 3, 10, 6, 1, 14, 13, 7, 4



BST - Add Node

```
//1. create node for given value
//2. if tree is empty
        // add newnode into root itself
//3. if tree is not empty
        //3.1 create trav and start at root node
        //3.2 check if value is less than current data
                //3.2.1 if left of current node is empty
                        // add newnode into left of current node
                //3.2.2 if left of current node is not empty
                        // go to the left of current node
        //3.3 check if value is greater or equal to data
                //3.3.1 if right of current node is empty
                        // add newnode into right of current node
                //3.3.2 if right of current node is not empty
                        // go to the right of current node
        //3.4 repeat step 3.2 and 3.3 untill node is added
```