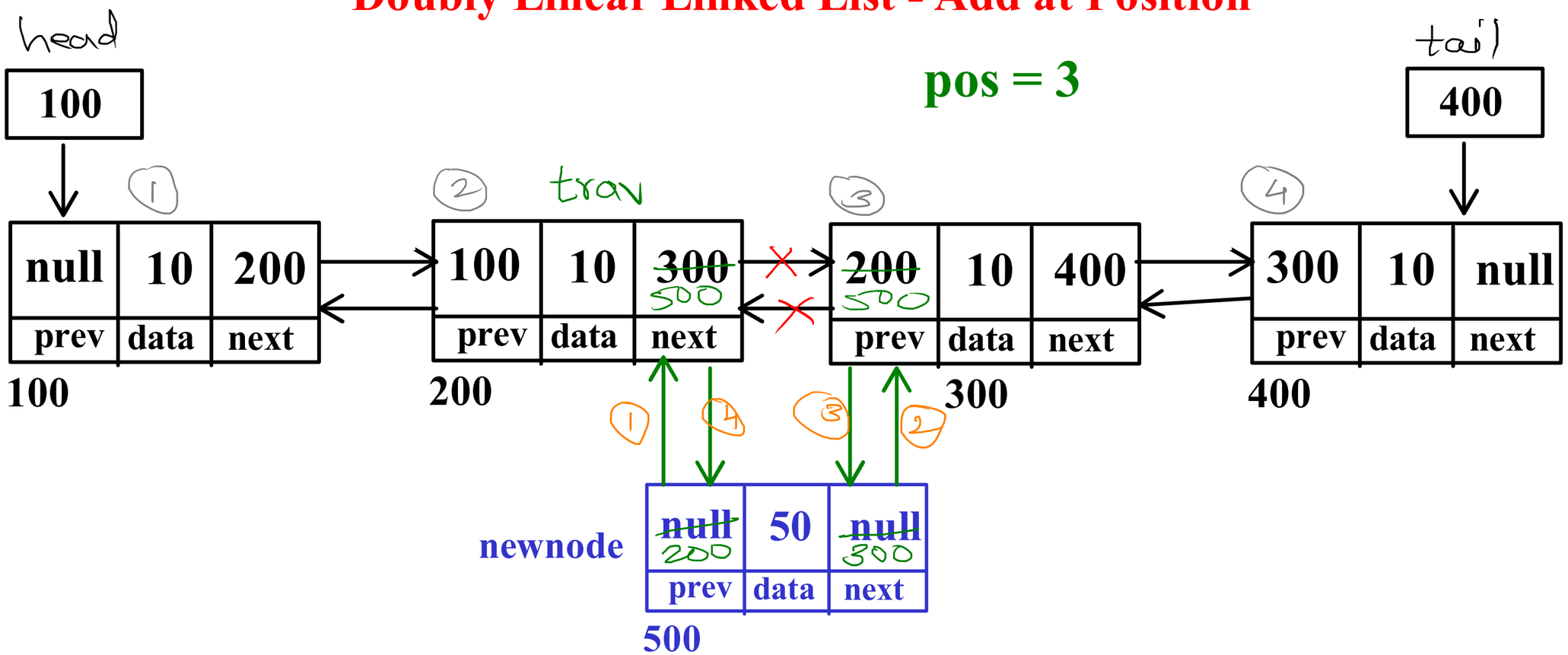
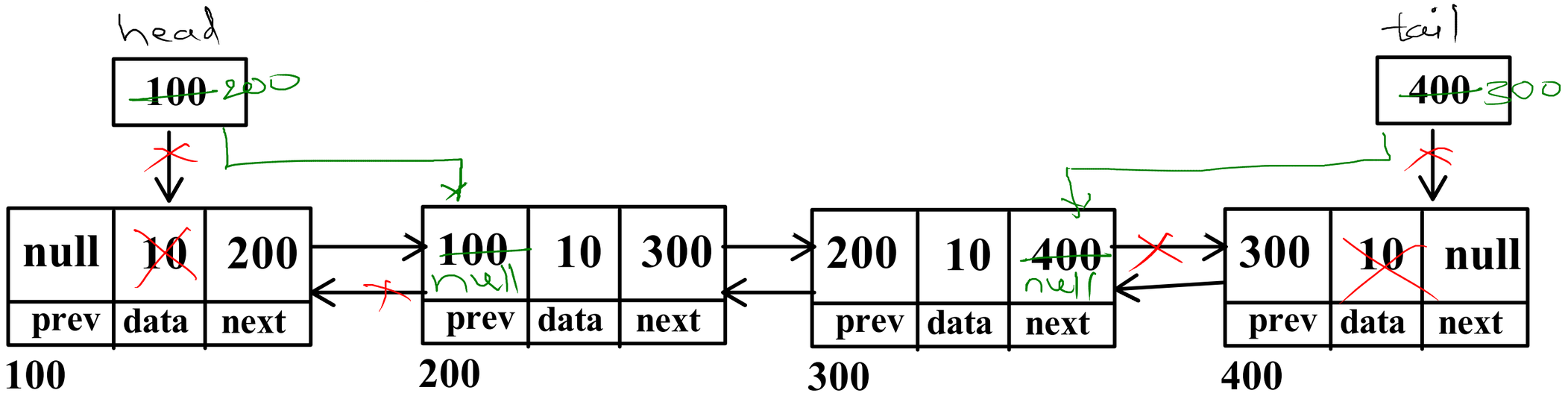
Doubly Linear Linked List - Add at Position




```
//1. create node with given value
//2. if list is empty
        //a. add newnode into head and tail
//3. if list is not empty
        //a. traverse till pos-1 node
        //b. add pos-1 node into prev of newnode
        //c. add pos node into next of newnode
        //d. add nenwode into prev of pos node
        //e. add newnode into next of pos-1 node
```

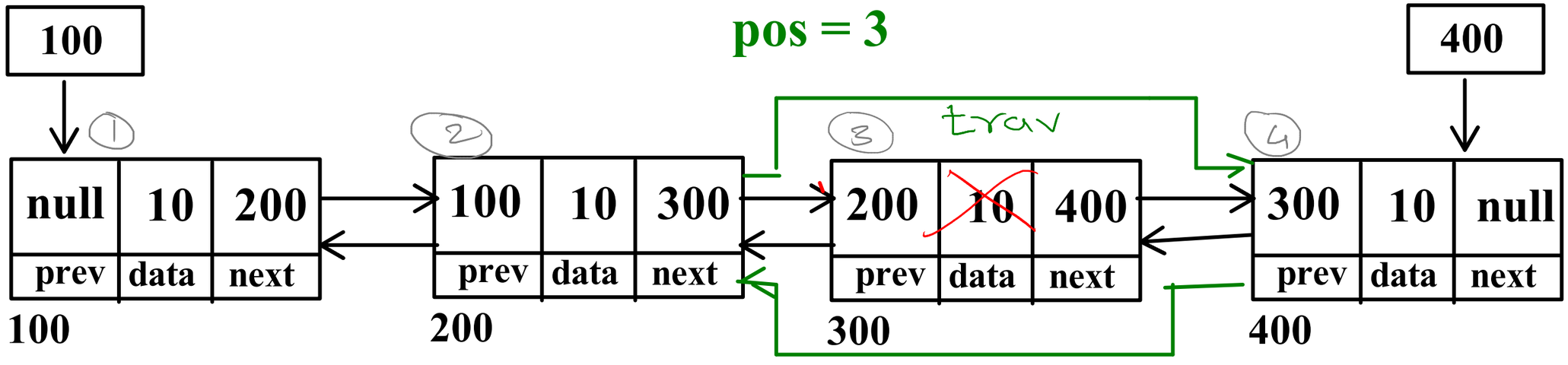
Doubly Linear Linked List - Delete First and Last




```
//1.  if list is empty
//2. if list has single node
//3. if list has multiple nodes
        //a. move head on second node
        //b. make prev of second node equal to null
```

```
//1.  if list is empty
//2. if list has single node
//3. if list has multiple nodes
        //a. move tail on second last node
        //b. make next of second last node equal to null
```

Doubly Linear Linked List - Delete Position



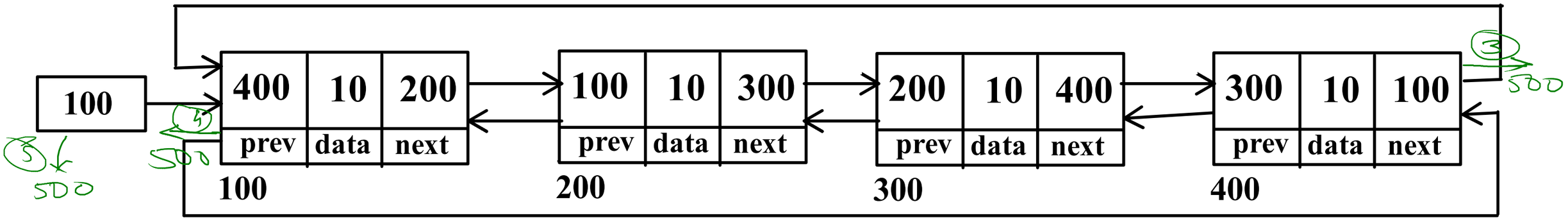
pos = trav

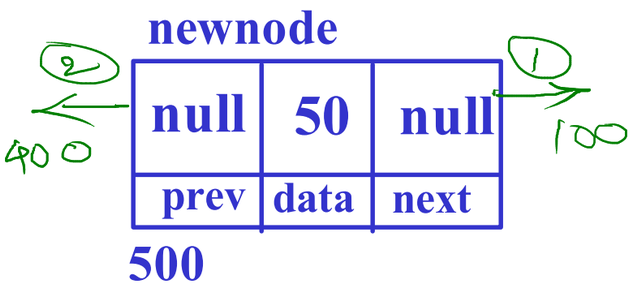
pos+1 = trav.next

pos+1 = trav.prev

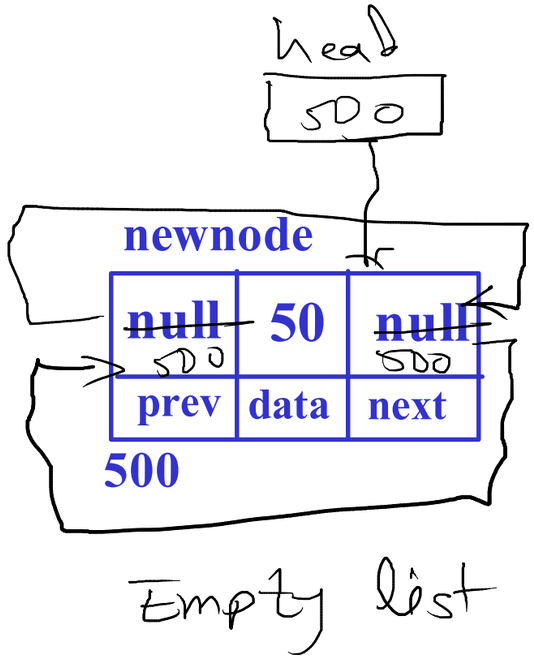
```
//1. if list is empty
//2. if list has single node
//3. if list has multiple nodes
        //a. traverse till pos node
        //b. add pos+1 node into next of pos-1 node
        //c. add pos-1 node into prev of pos+1 node
```

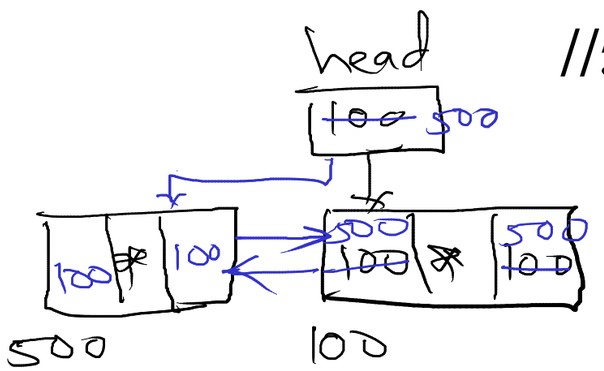
Doubly Circular Linked List - Add First



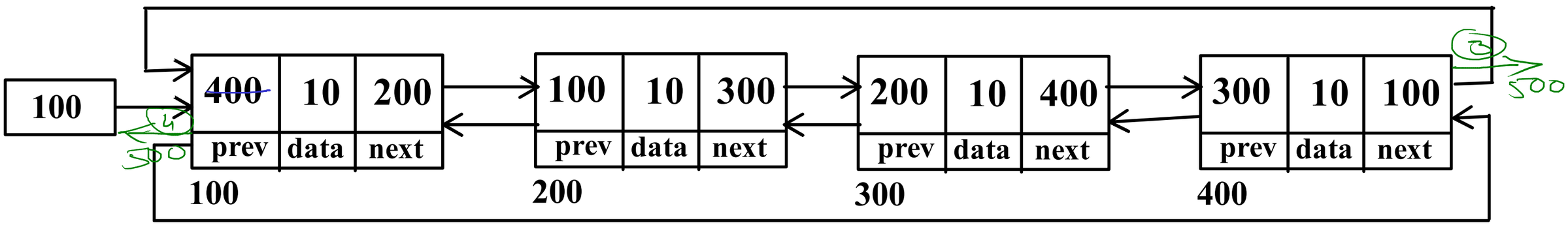


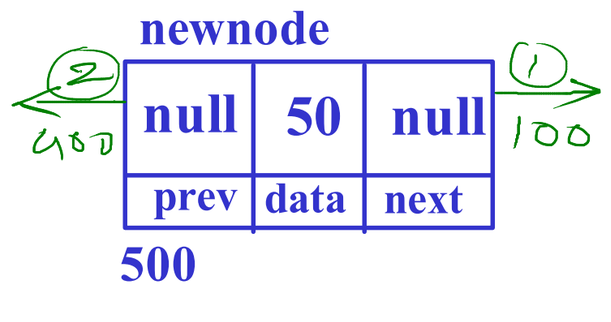

```
//a. create a node usig given value
//b. if list is empty
        //1. add newnode into head
        //2. make list circular
//c. if list is not empty
        //1. add first node into next of newnode
        //2. add last node into prev of newnode
        //3. add newnode into next of last node
        //4. add newnode into prev of first node
        //5. add newnode into head
```





Doubly Circular Linked List - Add Last

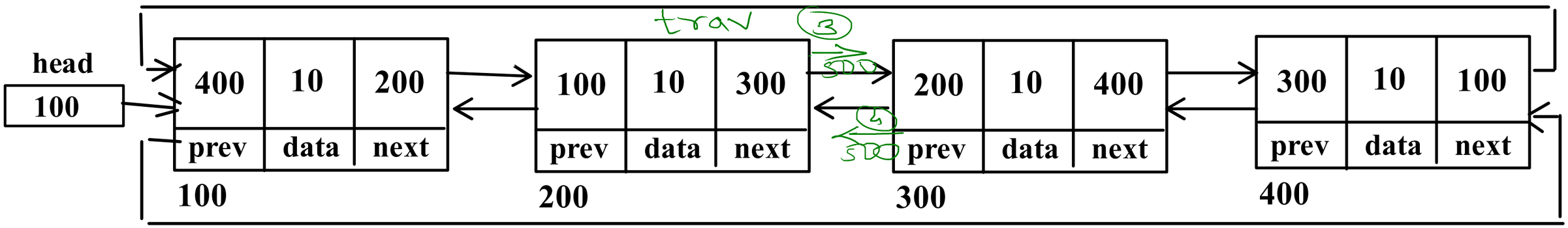


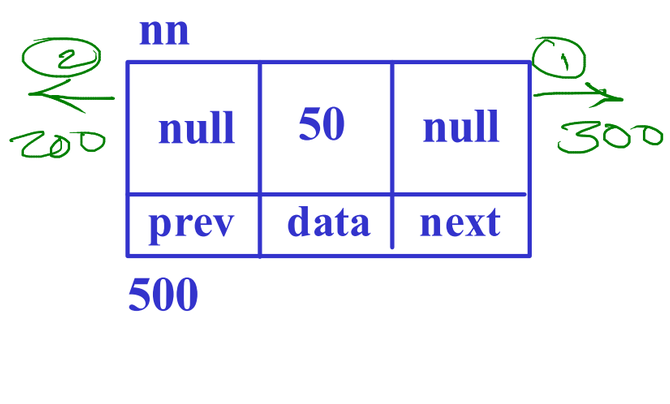



```
//a. create node
//b. if list is empty
        //1. add newnode into head
        //2. make list circular
//c. if list is not empty
        //1. add first node into next of newnode
        //2. add last node into prev of newnode
        //3. add newnode into next of last node
        //4. add nenwode into prev of first node
```

Doubly Circular Linked List - Add pos

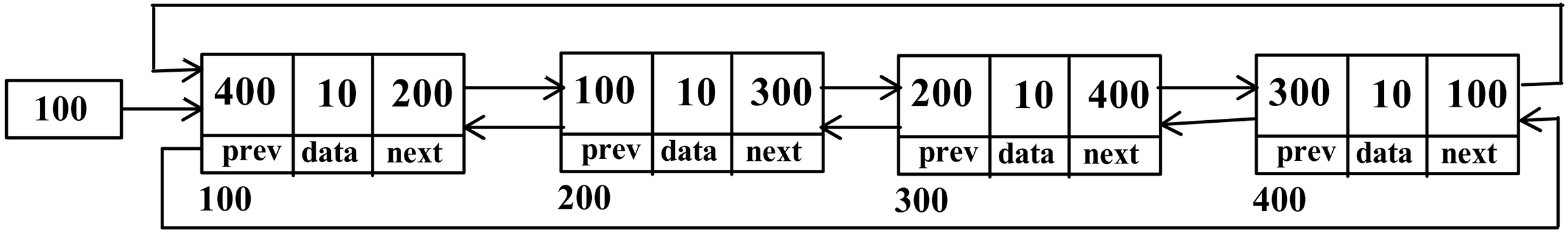
pos=3






```
//1. create node
//2. if list is empty
        //add newnode into head
        //make list circular
//3. if list is not empty
        //traverse till pos-1 node
        //add pos node into next of nn
        //add pos-1 node into prev of nn
        //add nn into next of pos-1 node
        //add nn into prev of pos node
```

Doubly Circular Linked List - Display



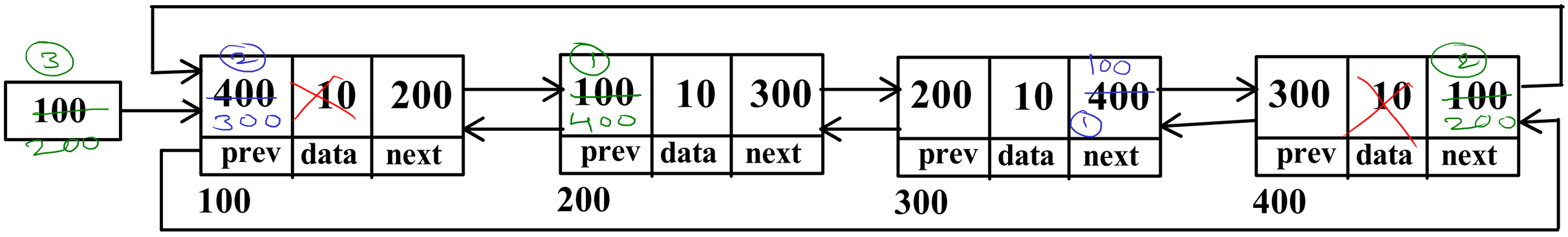
Forward

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

Backward

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

Doubly Circular Linked List - Delete First and Last

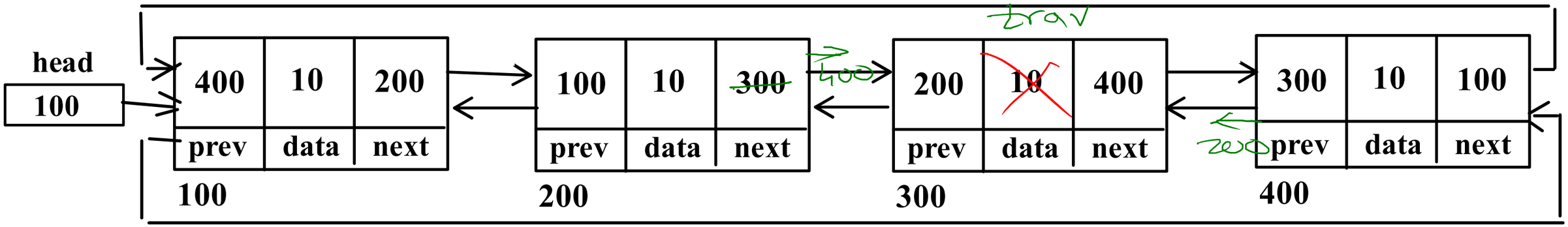


//1. if list is empty
//2. if list has single node
//3. if list has multiple nodes
//a. add last node into prev of second node
//b. add second node into next of last node
//c. add second node into head

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

Doubly Circular Linked List - Del Pos

pos=3




```
//0. special 1 - invalid position : pos < 1 or pos > cnt + 1
//0. special 2 - pos == 1
//1. if list is empty
//2. if list has single node
//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
```

Time Complexity Analysis of Linked List

	SLLL	SCLL	DLLL	DCLL
1. Add First	O(1)	O(n)	O(1)	O(1)
2. Add Last	O(n)	O(n)	O(n)	O(1)
3. Add Position	O(n)	O(n)	O(n)	O(n)
4. Del First	O(1)	O(n)	O(1)	O(1)
5. Del Last	O(n)	O(n)	O(n)	O(1)
6. Del Position	O(n)	O(n)	O(n)	O(n)
7. Display(traverse)	O(n)	O(n)	O(n)	O(n)

Efficient Linked List

Linked List Applications

- dynamic data structure - grow / shrink at runtime
- due to this dynamic nature, it is used to implement other data structures
    1. Stack
    2. Queue
    3. Hash Table (Seperate chaining)
    4. Graph (Adjacency list)

- Operating system - job queue, ready queue, waiting queues
  (Doubly circular linked list)
- Ring topology in Networks - (Doubly circular linked list)

Stack(LIFO)

1. Add First
2. Del First

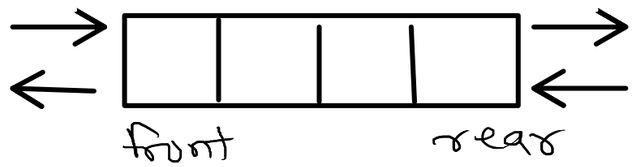
1. Add Last
2. Del Last

Queue(FIFO)

1. Add First
2. Del Last

1. Add Last
2. Del First

Deque (Double Ended Queue)



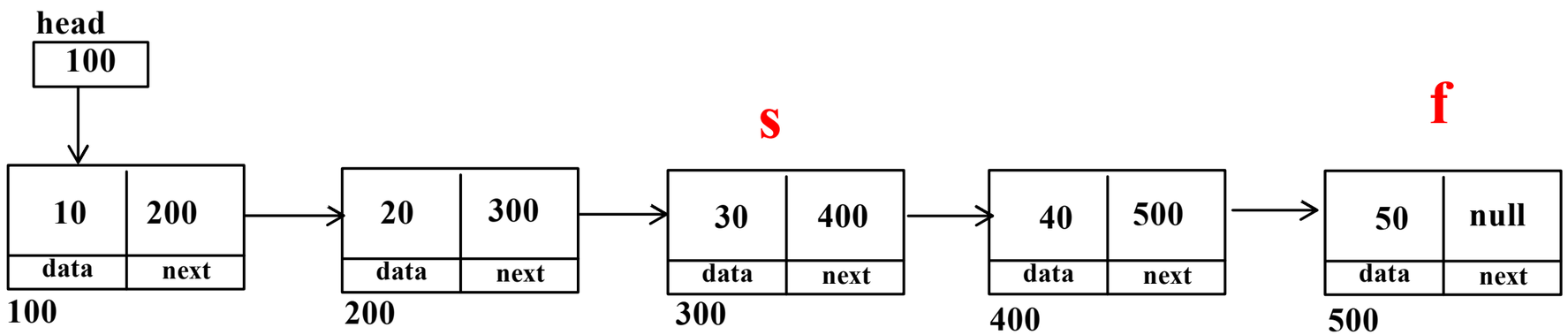
1. Push front
   Add first
2. Push rear
   Add last
3. Pop front
   Del First
4. Pop rear
   Del Last

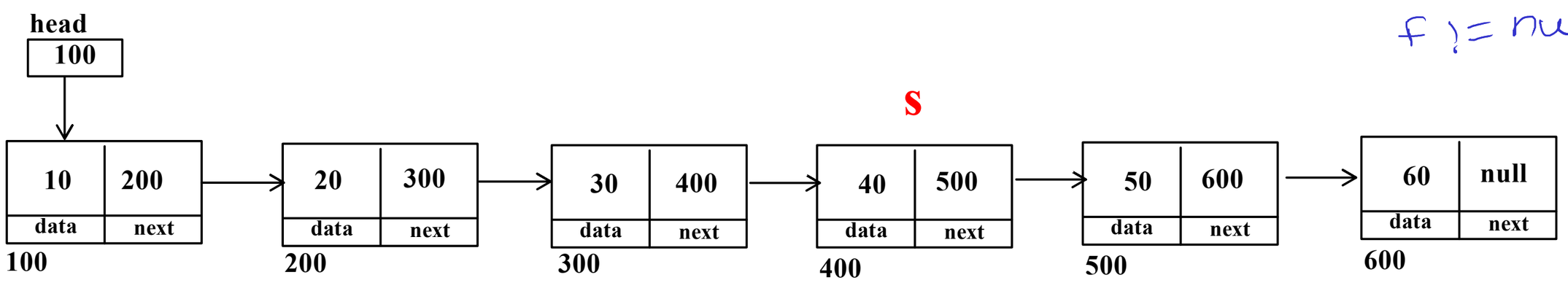
Types:
1. Input Restricted Deque
2. Output Restricted Deque

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

Linked List - Find Mid






```
void findMid(){
Node s = head, f = head;
while( f!=null && f.next!=null){
    s=s.next;
    f=f.next.next;
}
sysout("mid" + s.data);
}
```

iter = n

T ∝ n

T(n) = O(n)

Find middle of singly linear linked list using single pointer.

1. traverse till last node & count nodes.
2. traverse till count/2 and print middle nod

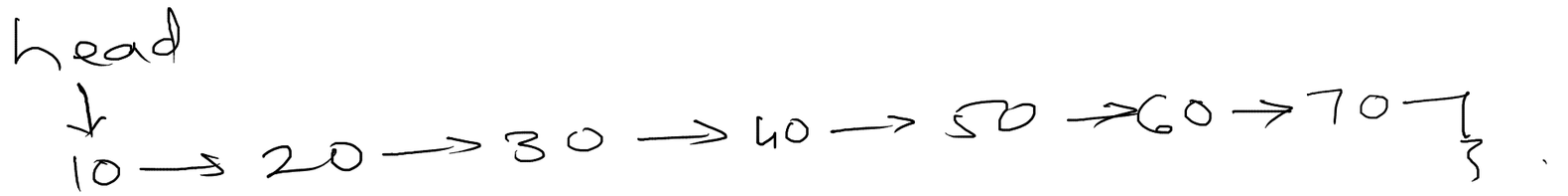
```
void findMid( ){
    count=0;
    trav = head;
    while(trav != null){
        count++;
        trav=trav.next;
    }
    trav=head;
    for(i=1; i<count/2; i++)
        trav = trav.next;
    sysout("mid :" + trav.data);
}
```

$its = n + n/2$

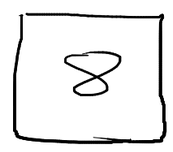
$T \propto n$

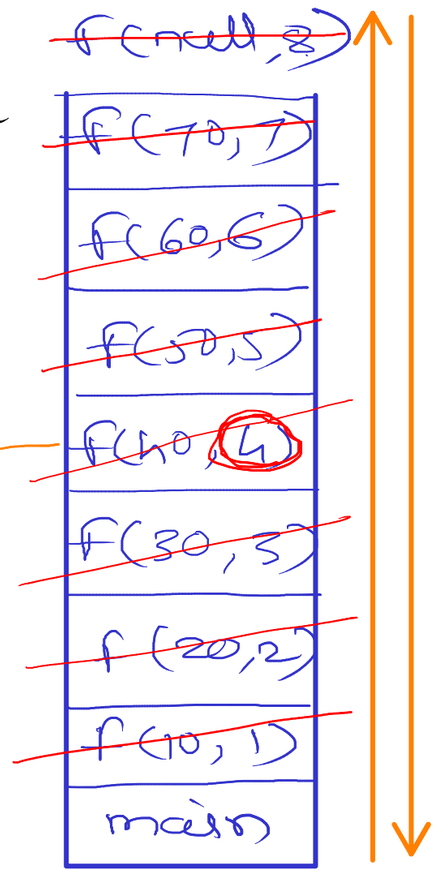
$T(n) = O(n)$

Find middle of singly linear linked list using single pointer and recursion

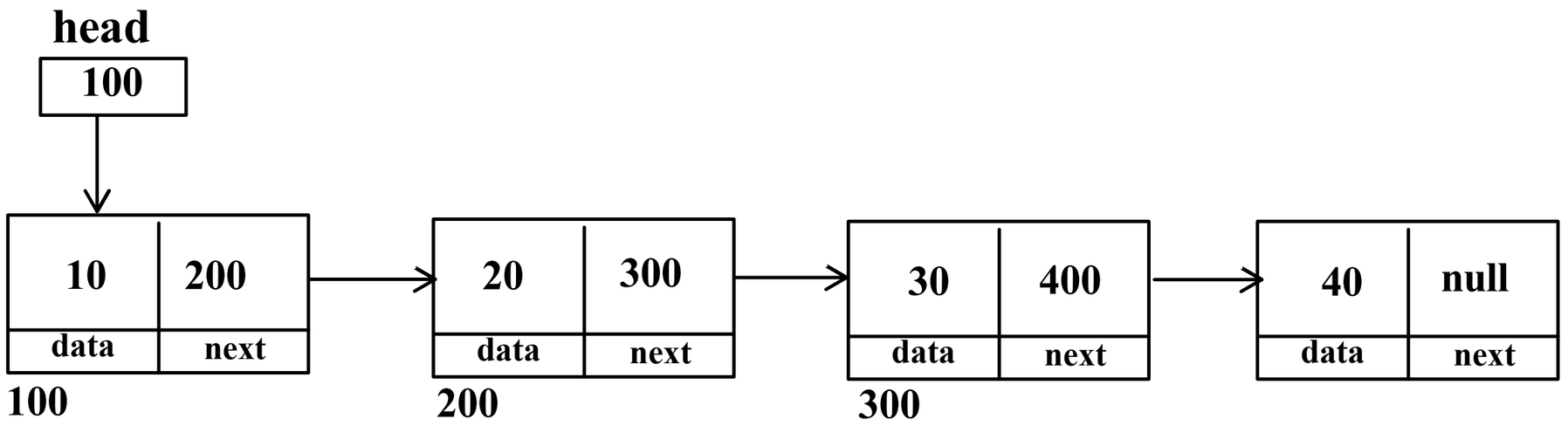



```
int max; //global
void findMid(INode trav, int no) {
    if(trav == null) {
        max = no;
        return;
    }
    findMid(trav.next, no+1);
    if(no == max/2)
        sysout(trav.data);
}
main -> findMid(head, 1);
```





Linked List - Display Reverse



① store all elements of linked into stack one by one
② pop elements from stack one by one and display them.

```
void displayReverse() {
    Stack<Integer> s = new Stack<>();
    trav = head;
    while(trav != null)
        s.push(trav.data);
    while(!s.isEmpty())
        sysout(s.pop());
}
```

iters = 2 * n

time ∝ n

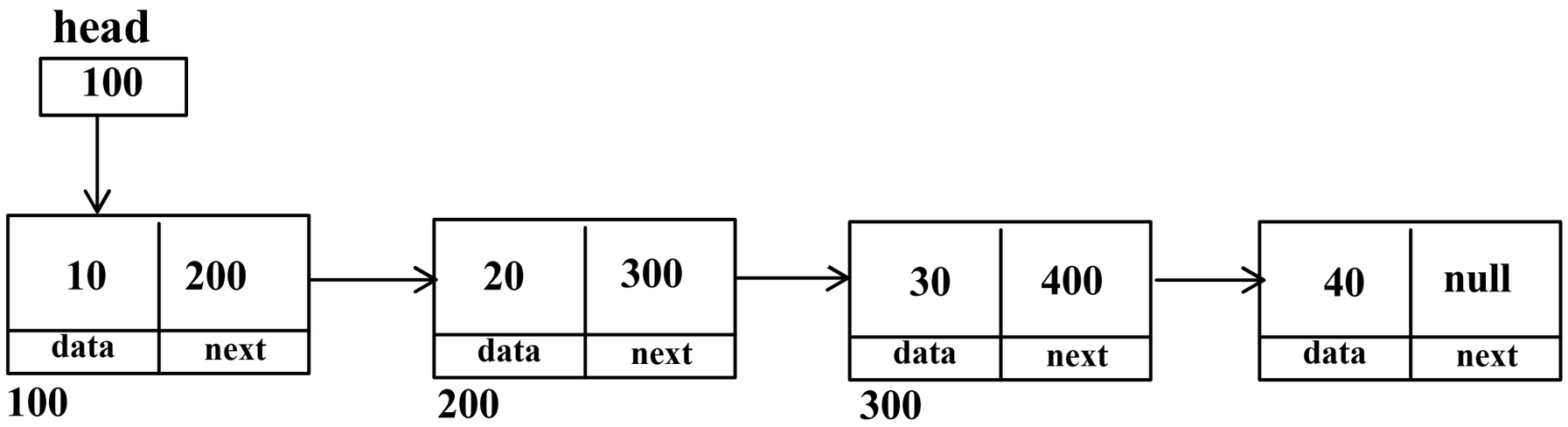
T(n) = O(n)

Auxillary space = n

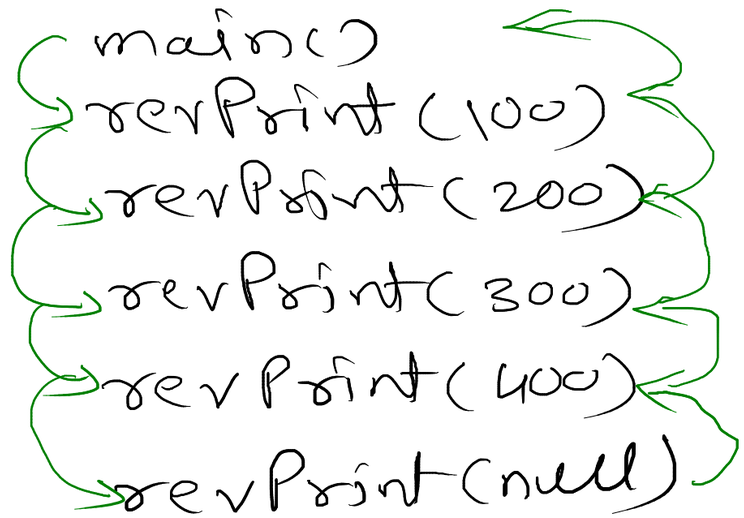
S(n) = O(n)

↑ additional space of stack

Display singly linked list in reverse order.




```
void revPrint(INode trav){
    if(trav == null)
        return;
    revPrint(trav.next);
    sysout(trav.data);
}
main -> revPrint(head);
```



40, 30, 20, 10

$T(n) = O(n)$