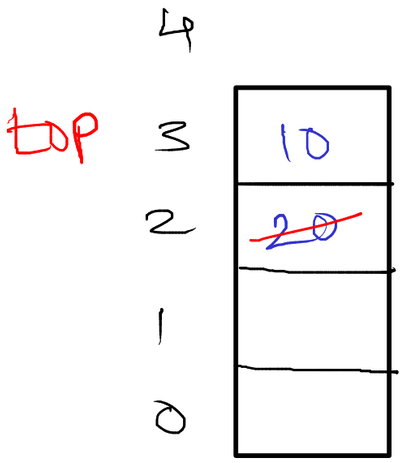
Descending Stack

top = size



Push:

```
top--;
arr[top] = value;
```

Pop:

```
top++;
```

Peek:

```
return arr[top]
```

Empty:

```
top == size
```

Full:

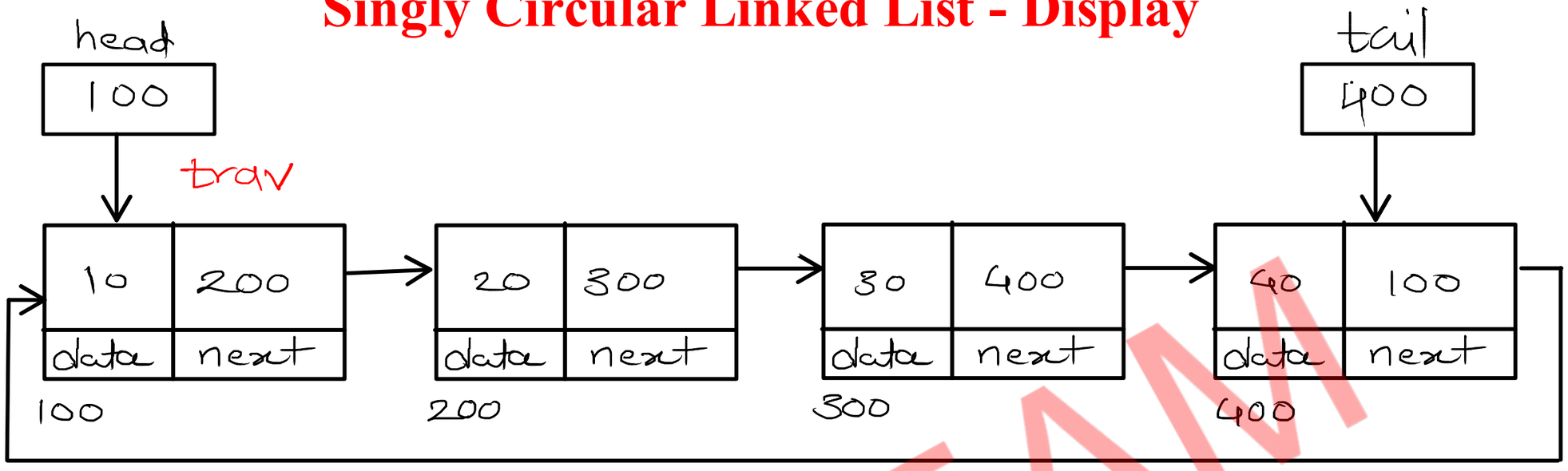
```
top == 0
```

```
class Employee {
    int empid;
    String name;
    String address;
}

class Node {
    Employee data;
    Node next;
}
```

SUNBEAM

Singly Circular Linked List - Display



//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

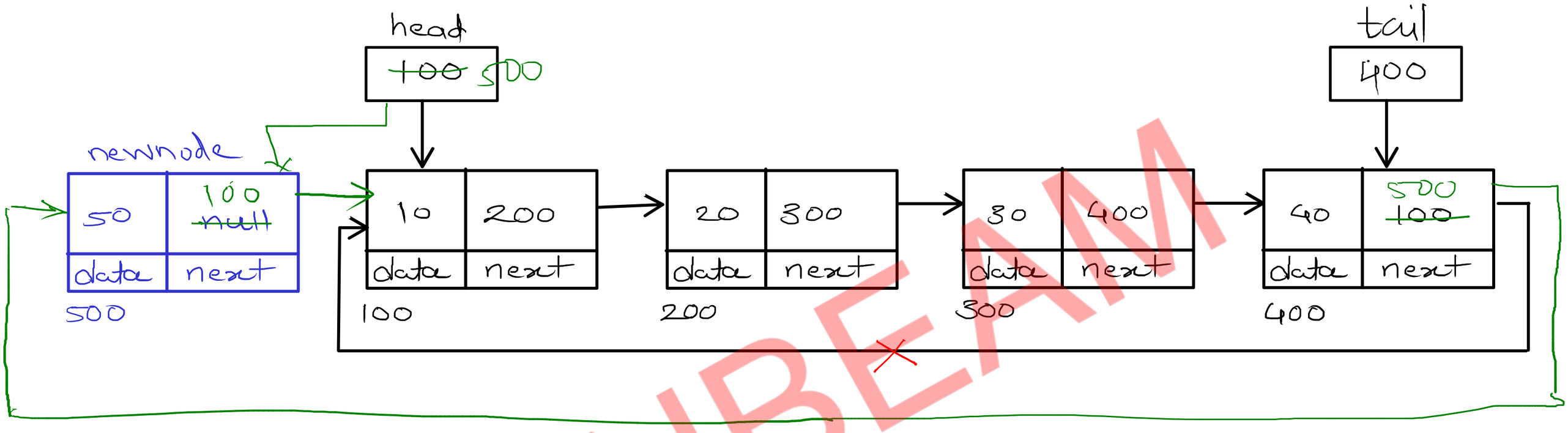
trav	trav.data	trav.next
100	10	200
200	20	300
300	30	400
400	40	100
100		

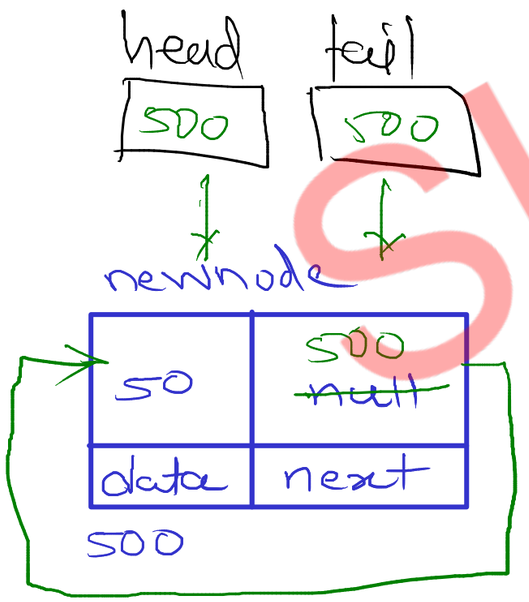
```
Node trav = head;
do {
    sysout(trav.data);
    trav = trav.next;
} while(trav != head);
```

T(n) = O(n)

SUNBEAM

Singly Circular Linked List - Add First



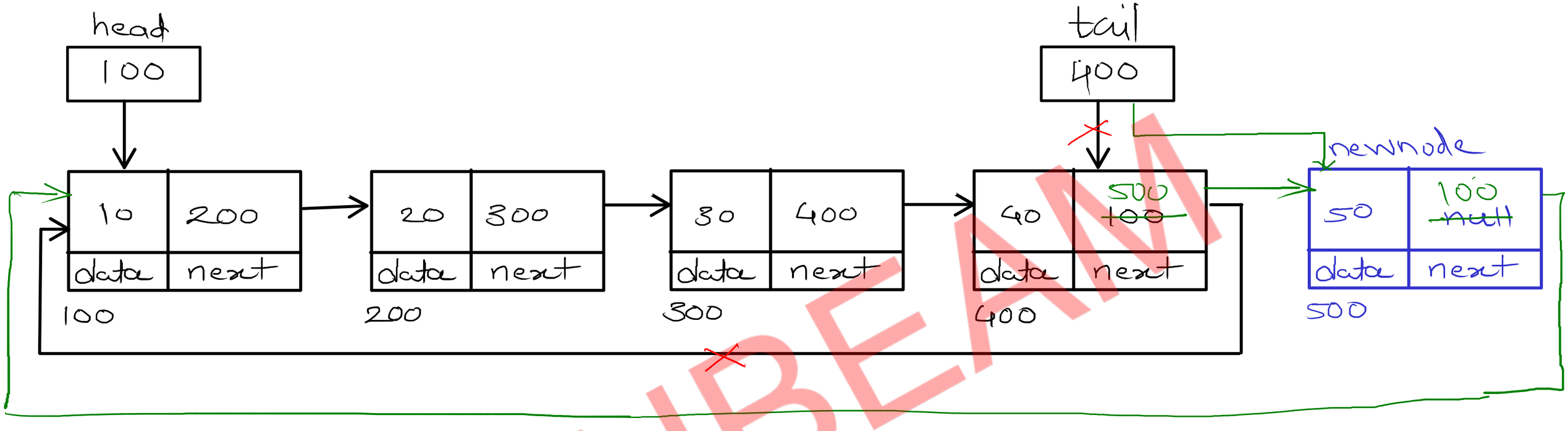


T(n) = O(1)

```
//1. create a newnode
//2. if list is empty
      //a. add newnode into head and 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
```

SUNBEAM

Singly Circular Linked List - Add Last

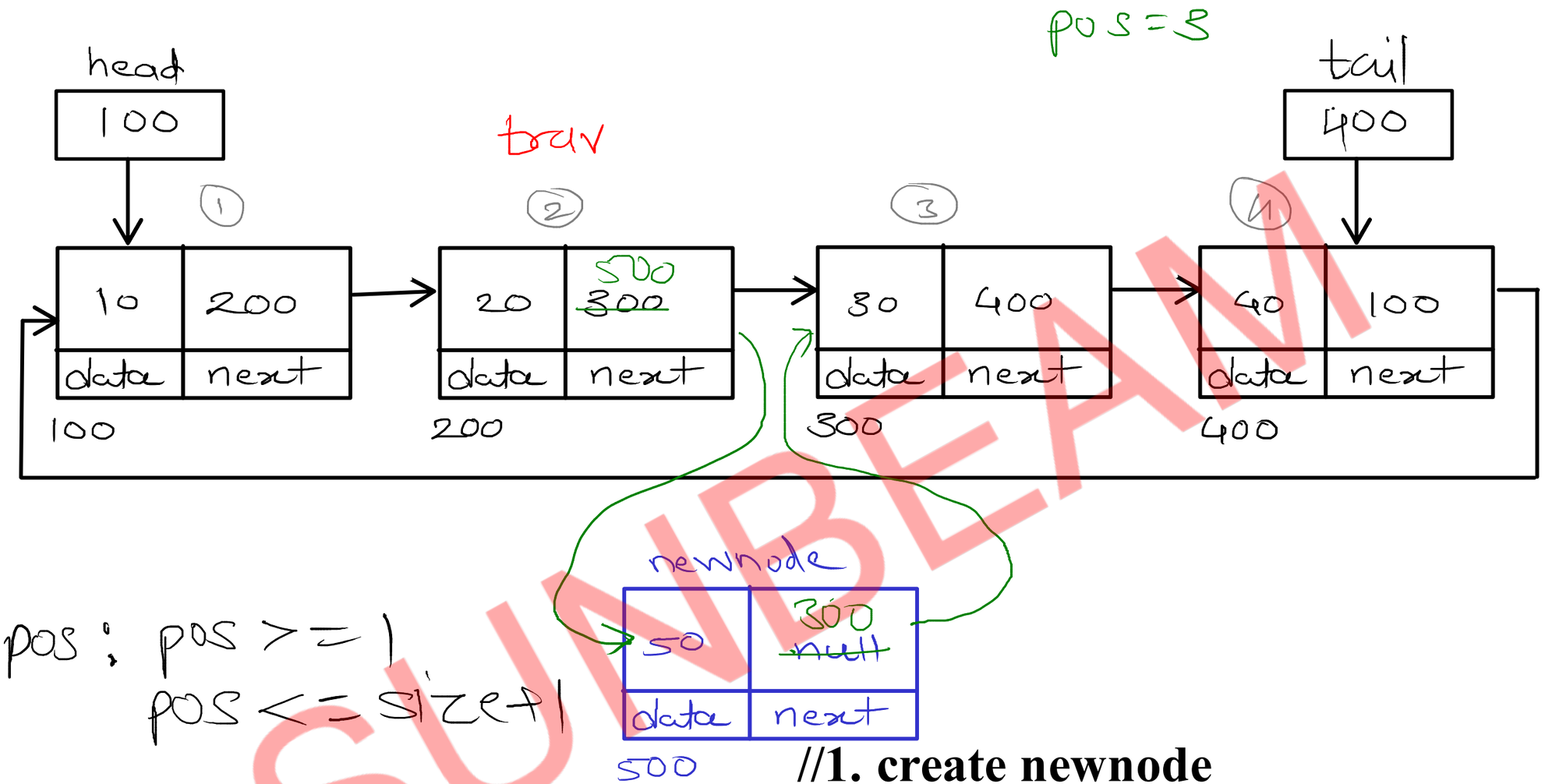


SUNBEAM

```
//1. create a newnode
//2. if list is empty
        //a. add newnode into head and 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



valid pos : pos >= 1
pos <= size+1

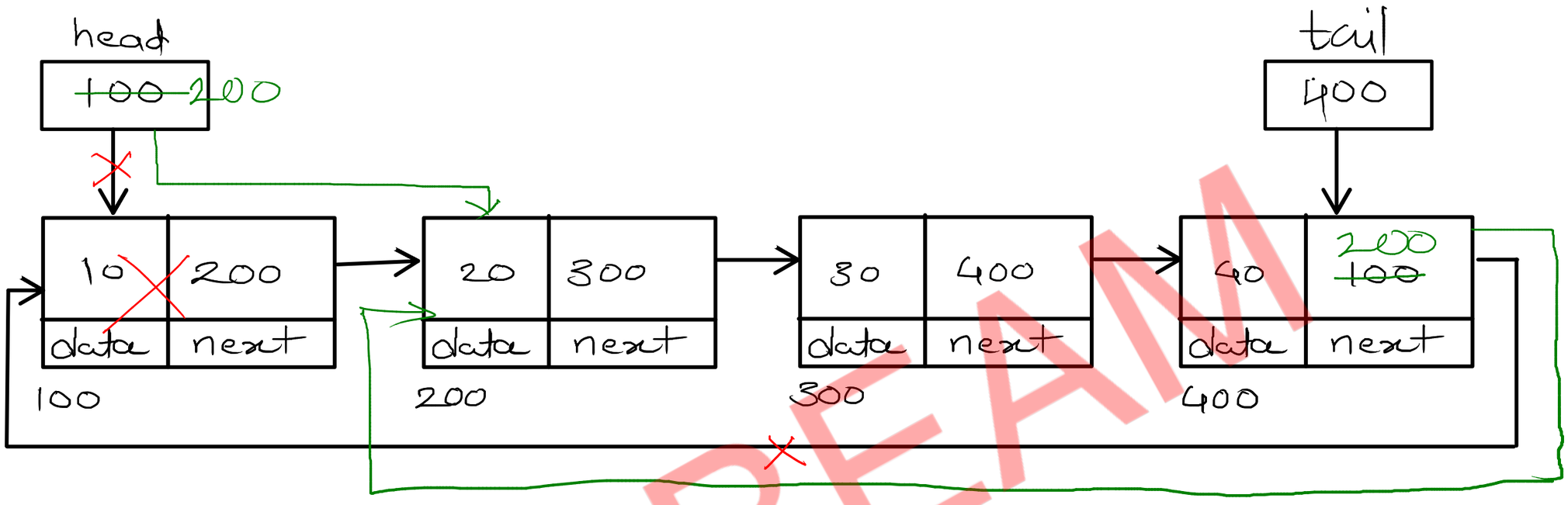
invalid pos : pos < 1
pos > size+1

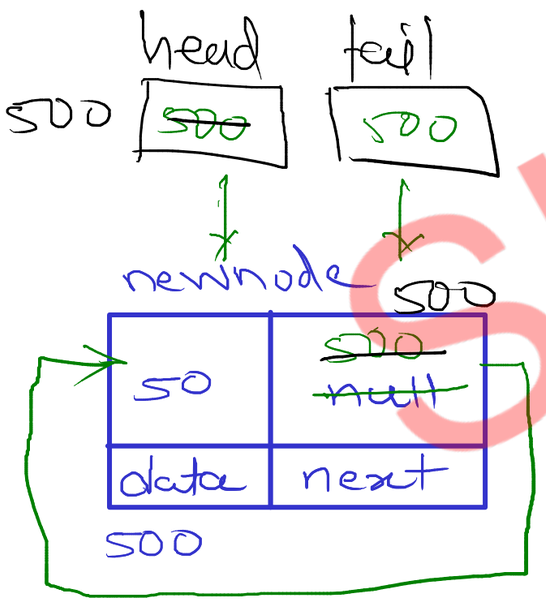
T(n) = O(n)

//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. traverse till pos-1
//b. add pos node into next of newnode
//c. add newnode into next of pos -1 node

SUNBEAM

Singly Circular Linked List - Delete first





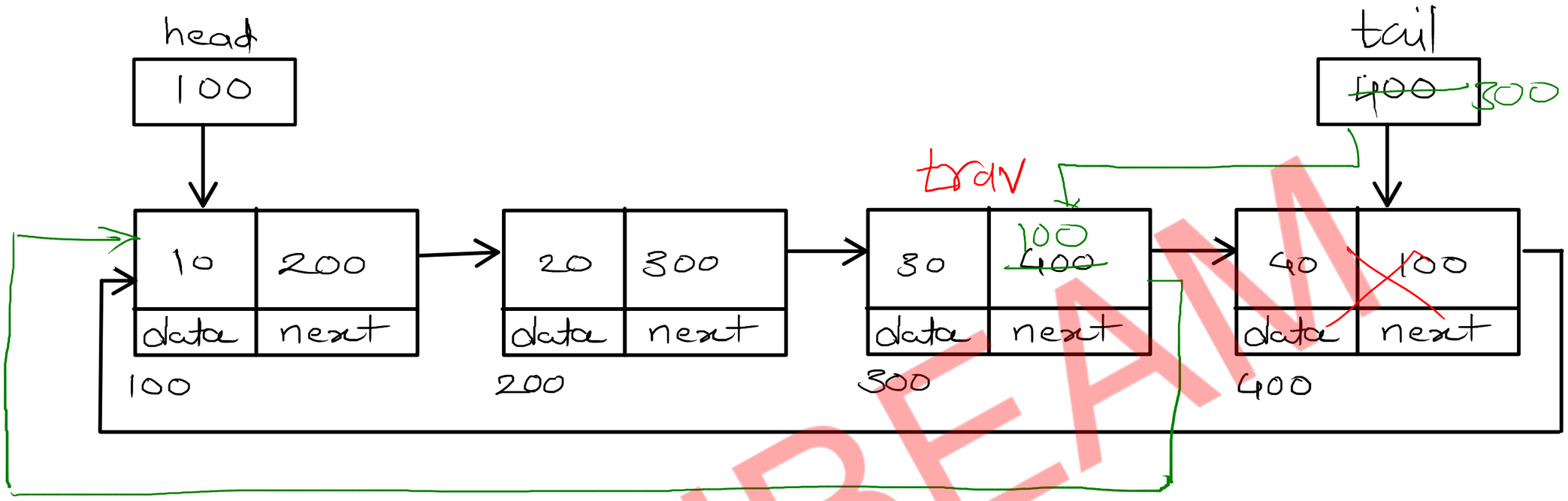

```
//1. if list is empty
      return;
//2. if list has single node
      head = tail = null;
//3. if list has multiple nodes
      //a. add second node into next of last node
      //b. move head on second node
```

tail.next = head.next
head = head.next

T(n) = O(1)

SUNBEAM

Singly Circular Linked List - Delete Last

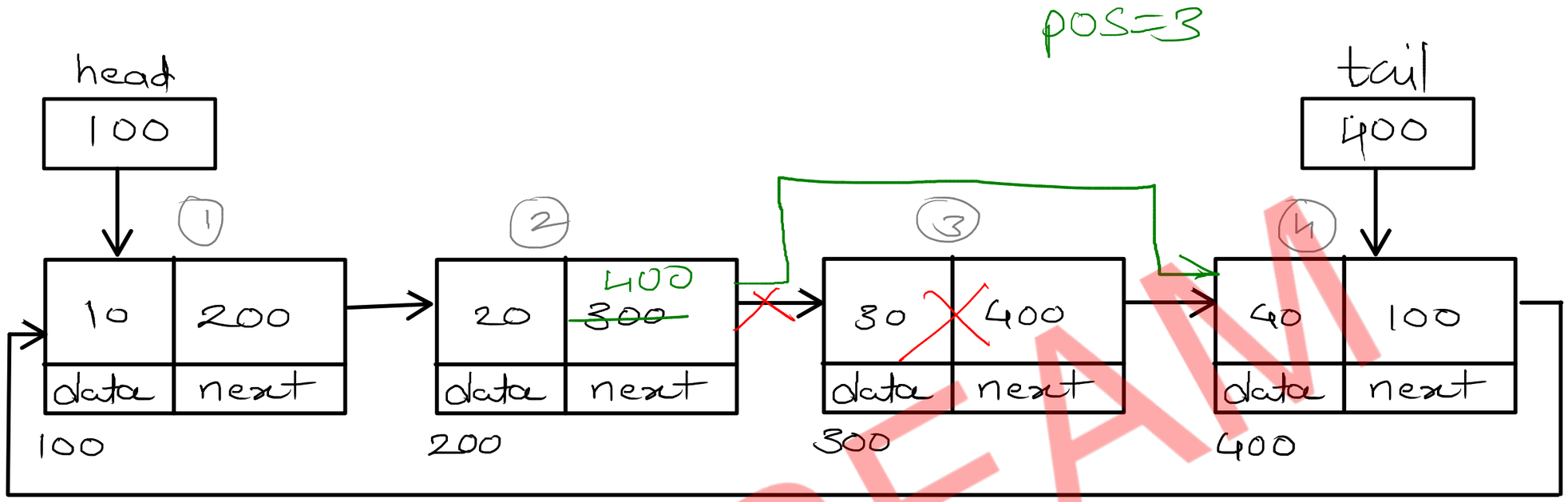


T(n) = O(n)

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

SUNBEAM

Singly Circular Linked List - Delete Position



valid pos : pos >= 1
pos <= size

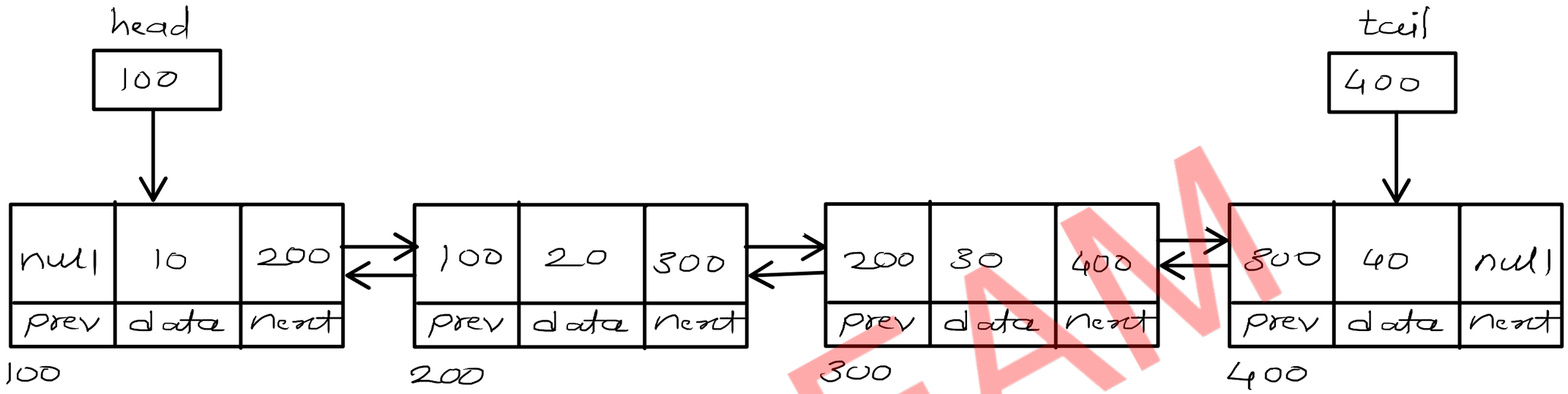
invalid pos : pos < 1
pos > size

T(n) = O(n)

```
//1. if list is empty
      return;
//2. if list has single
      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
```

SUNBEAM

Doubly Linear Linked List - Display



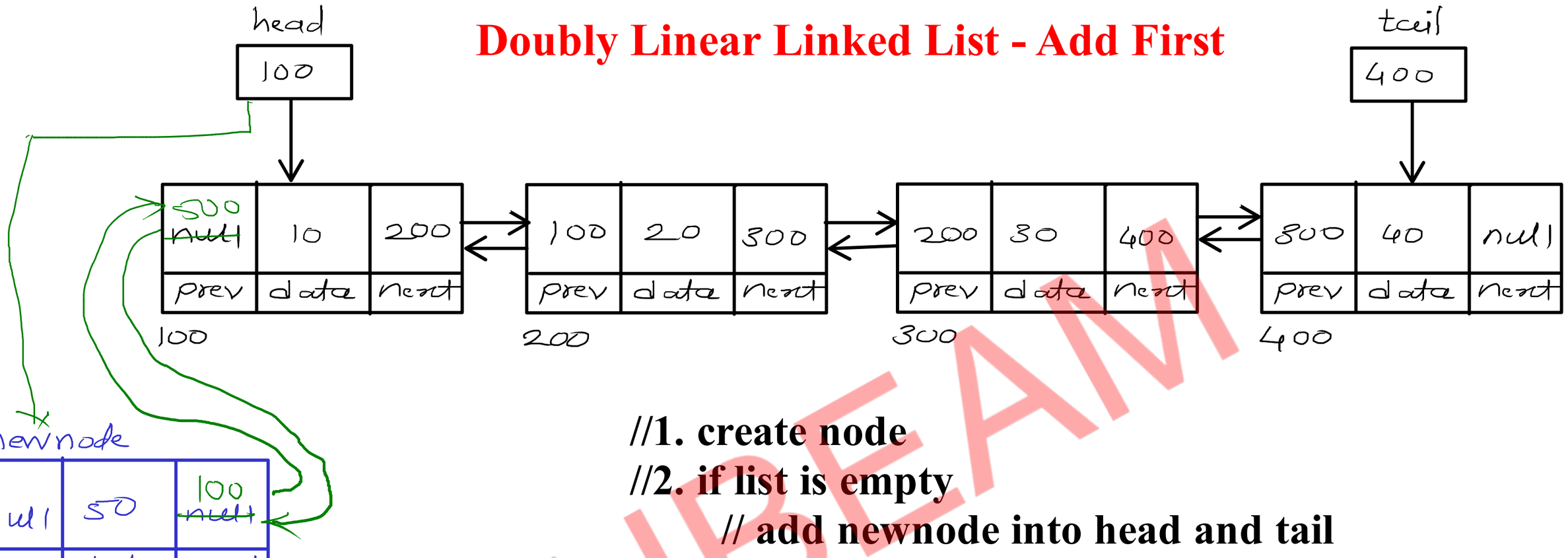
// forward traversal
//1. start at head
//2. print current node
//3. go on next node
//4. repeat step 2 and 3 till last node

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

$T(n) = O(n)$

SUNBEAM

Doubly Linear Linked List - Add First

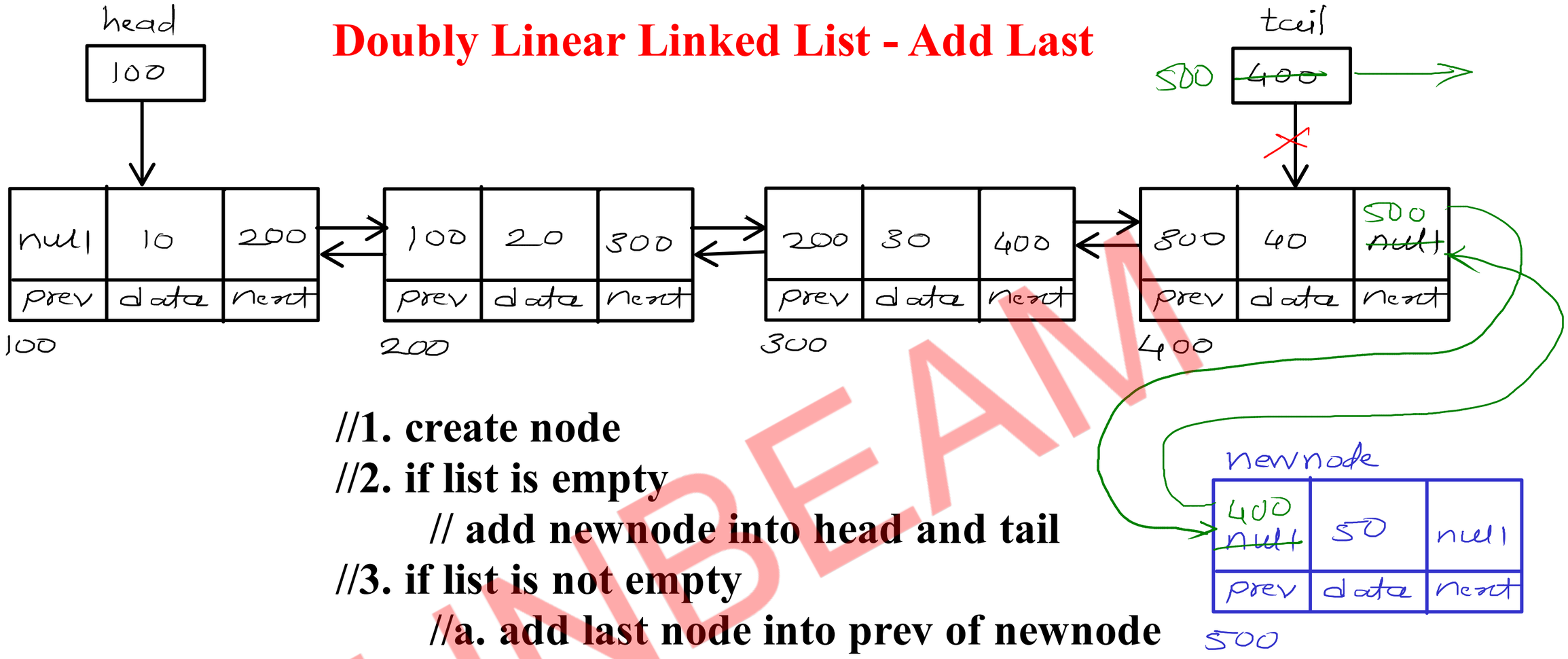



```
//1. create node
//2. if list is empty
        // add newnode into head and tail
//3. if list is not empty
   //a. add first node into next of newnode
   //b. add newnode into prev of first node
   //c. move head on newnode
```

T(n) = O(1)

SUNBEAM

Doubly Linear Linked List - Add Last

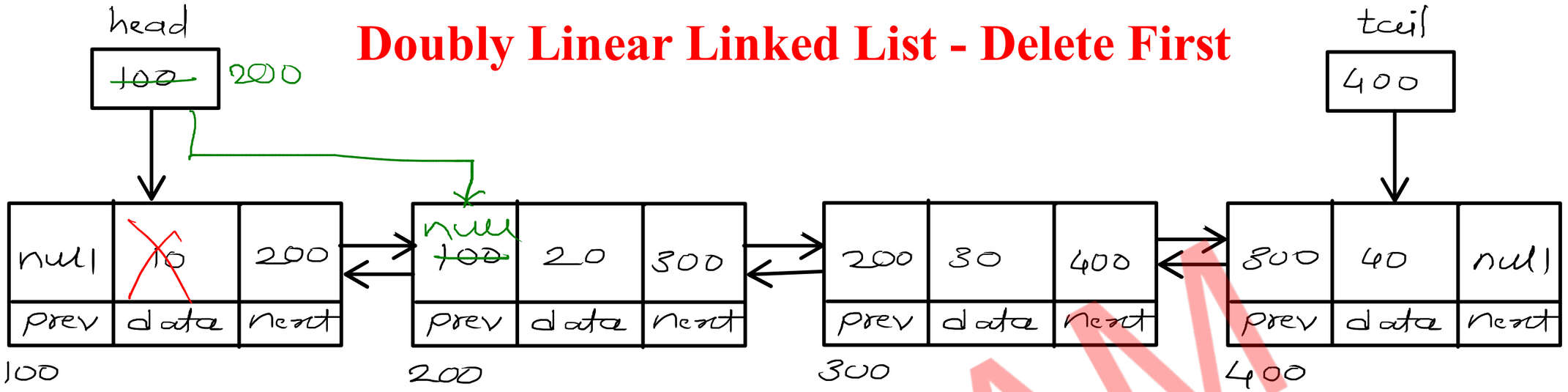


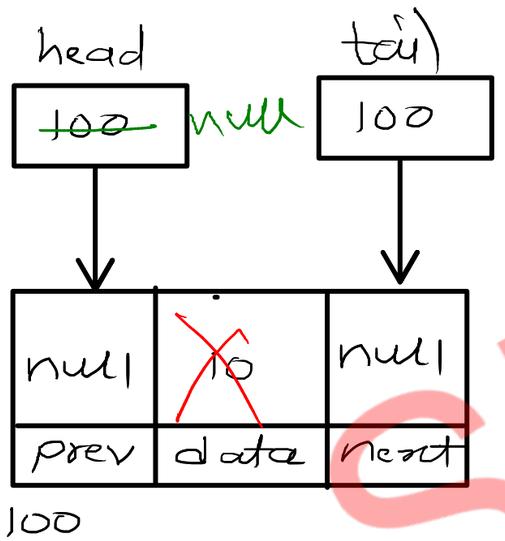
//1. create node
//2. if list is empty
// add newnode into head and tail
//3. if list is not empty
//a. add last node into prev of newnode
//b. add newnode into next of last node
//c. move tail on newnode

$T(n) = O(1)$

SUNBEAM

Doubly Linear Linked List - Delete First





head = head.next

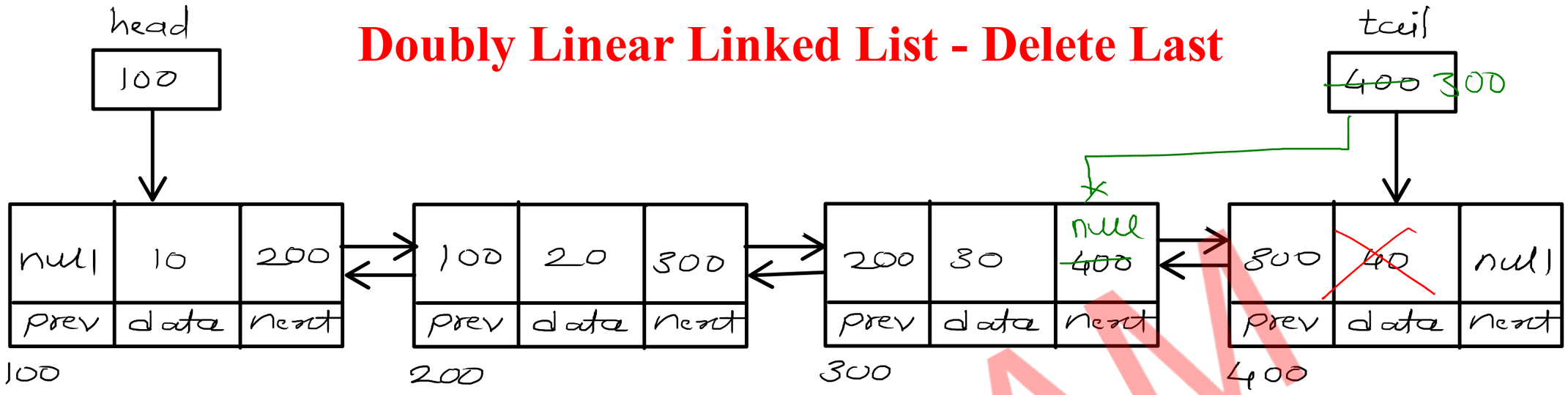
head.prev = null

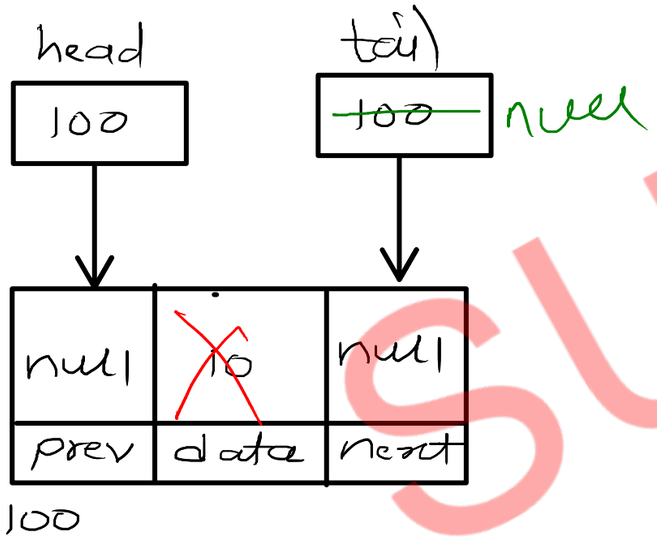
```
//1. if list is empty
       return;
//2. if list has single node
       head = tail = null;
//3. if list has multiple node
     //a. move head on second node
     //b. make prev of second node equal to null
```

T(n) = O(1)

SUNBEAM

Doubly Linear Linked List - Delete Last





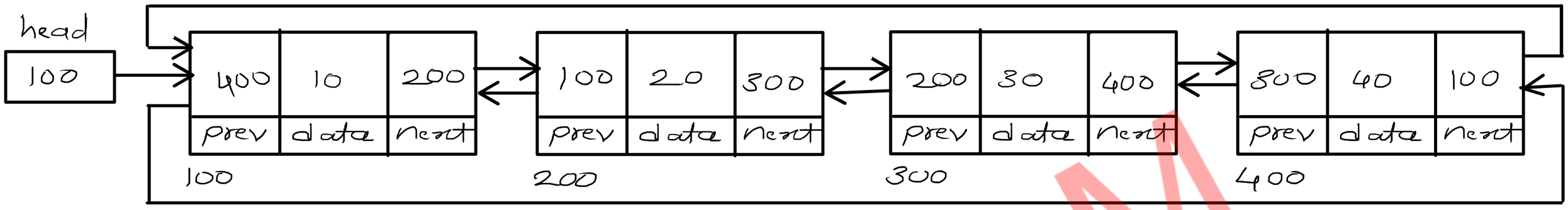

```
tail = tail.prev;
tail.next = null;
```

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

T(n) = O(1)

SUNBEAM

Doubly Circular Linked List - Display



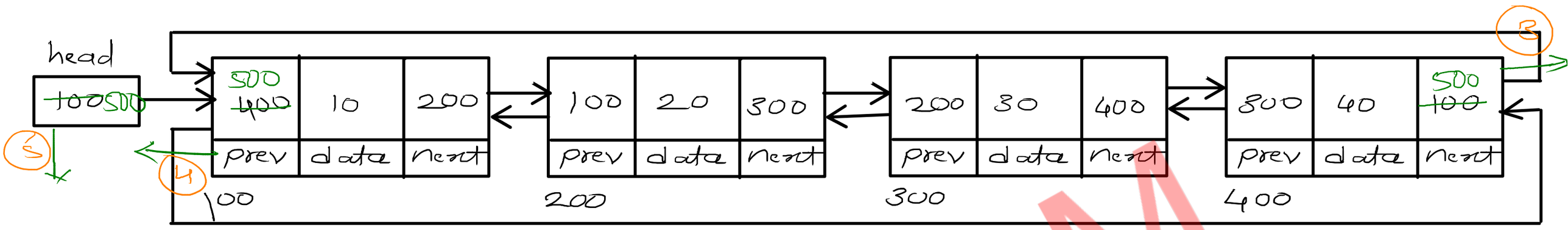
// forward traversal
//1. start at head
//2. print current node
//3. go on next node
//4. repeat step 2 and 3 till last node

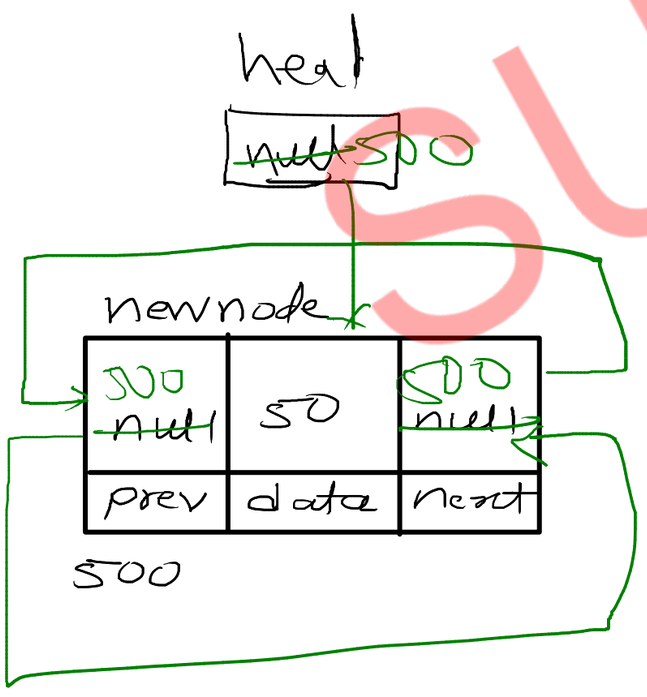
// reverse traversal
//1. start at tail
//2. print current node
//3. go on prev node
//4. repeat step 2 and 3 till first node

$T(n) = O(n)$

SUNBEAM

Doubly Circular Linked List - Add first



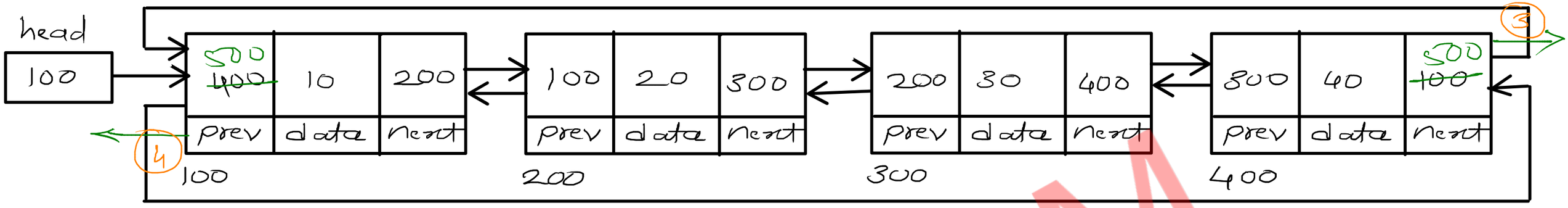


//a. create a newnode
//b. if list is empty
//1. add newnode into head
//2. make list circular
//c. if list is not empty
//1. add first node into next of newnode
//2. add last node into prev of newnode
//3. add newnode into next of last node
//4. add newnode into prev of first node
//5. move head on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

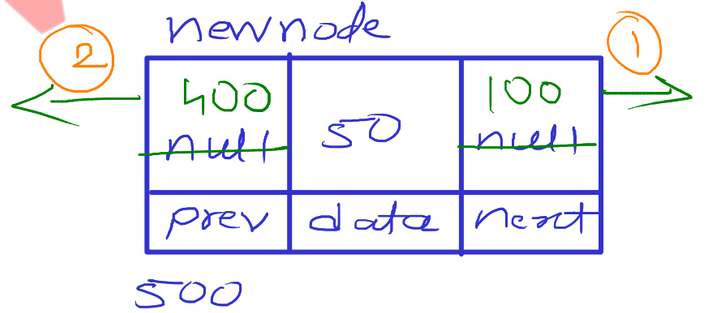
$T(n) = O(1)$

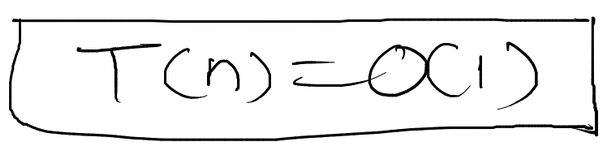
SUNBEAM

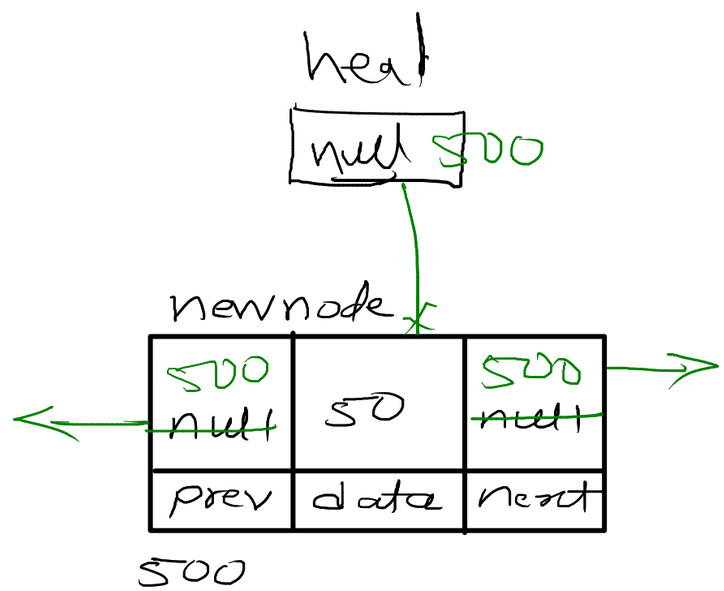
Doubly Circular Linked List - Add Last



//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 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

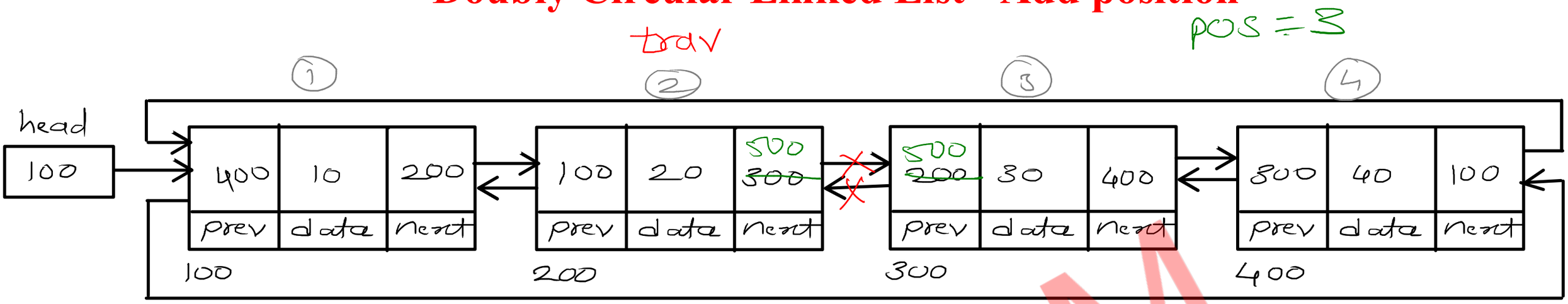


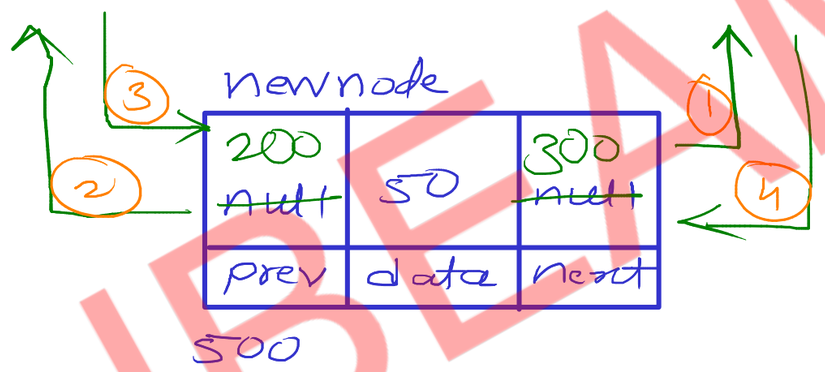




SUNBEAM

Doubly Circular Linked List - Add position

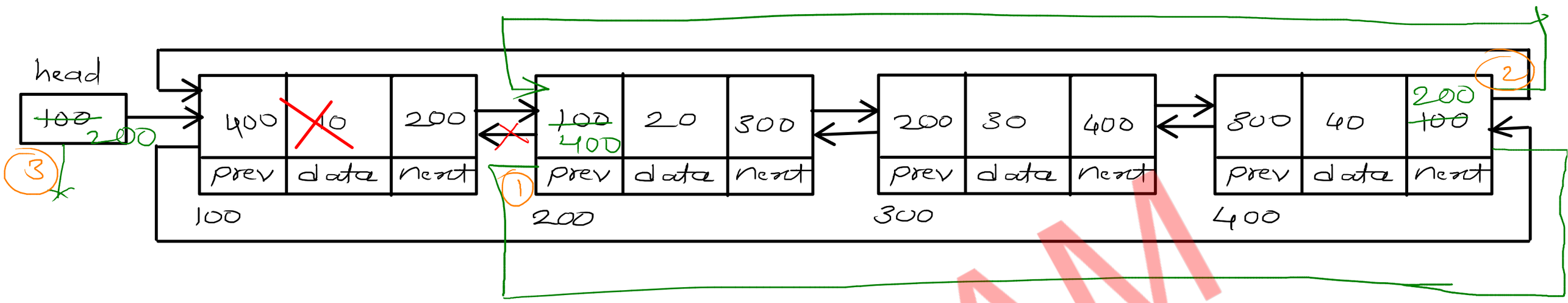


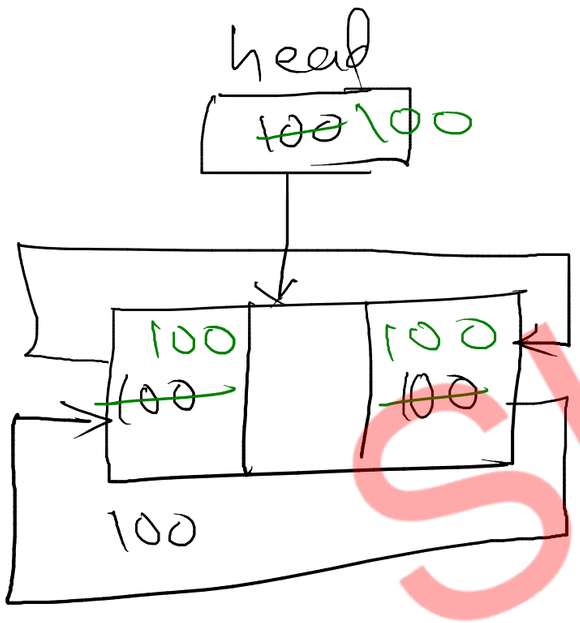


//a. create a newnode
//b. if list is empty
//1. add newnode into head
//2. make list circular
//c. if list is not empty
// traverse till pos-1 node
//1. add pos node into next of newnode
//2. add pos-1 node into prev of newnode
//3. add newnode into next of pos-1 node
//4. add newnode into prev of pos node

SUNBEAM

Doubly Circular Linked List - Delete First



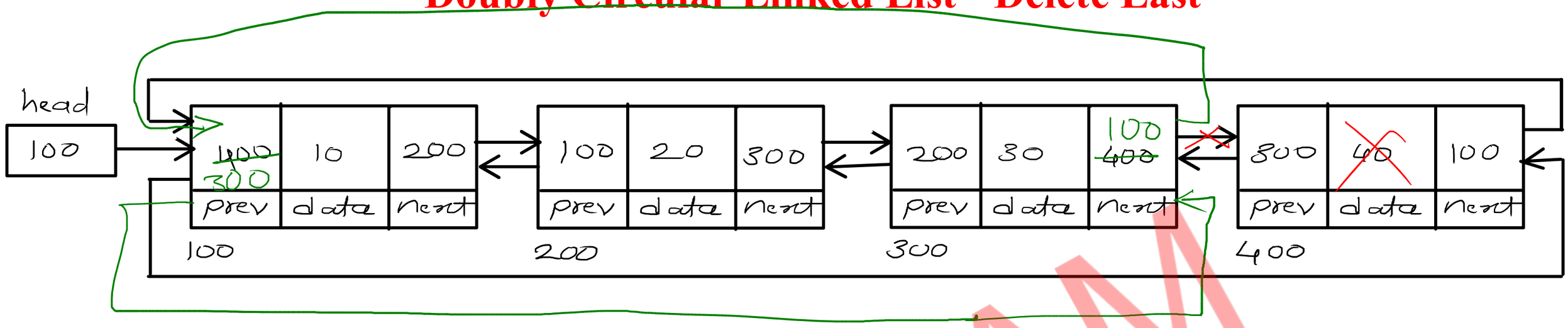


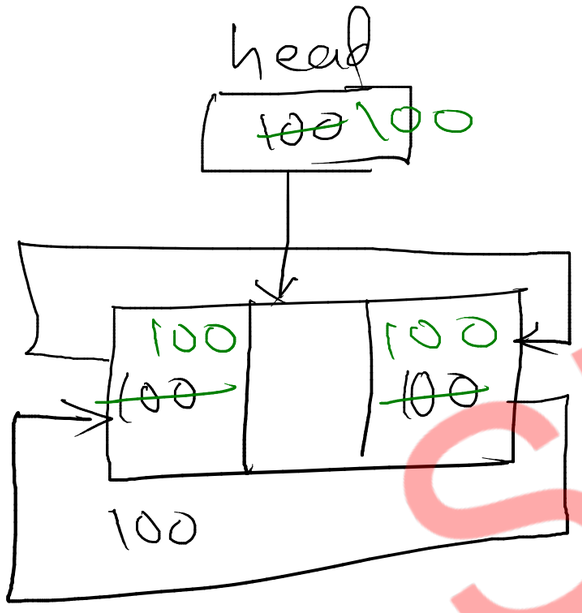
//1. if list is empty
return;
//2. if list has single node
head = null;
//3. if list has multiple node
//a. add laast node into prev of second node
//b. add second node into next of last node
//c. move head on second node

T(n) = O(1)

SUNBEAM

Doubly Circular Linked List - Delete Last





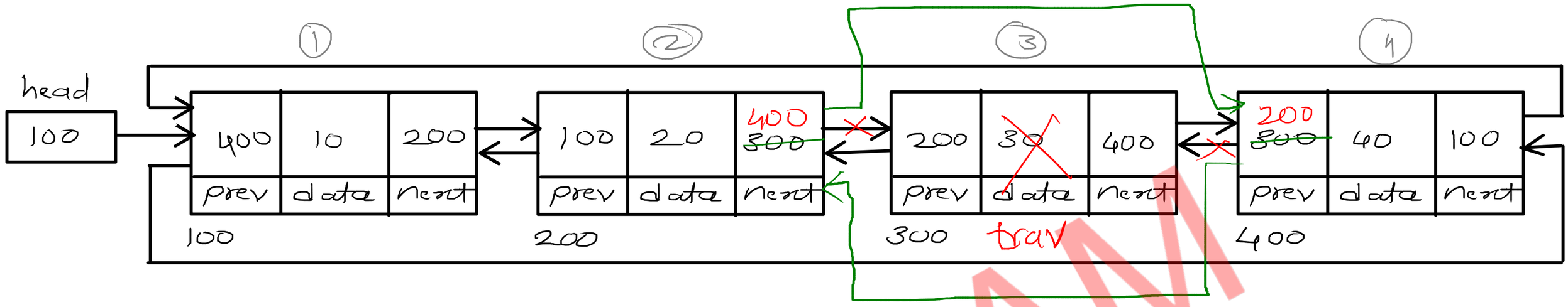
//1. if list is empty
return;
//2. if list has single node
head = null;
//3. if list has multiple node
//a. add first node into next of second last node
//b. add second last node into prev of first node

T(n) = O(1)

SUNBEAM

Doubly Circular Linked List - Delete Position

pos = 3




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

T(n) = O(n)

SUNBEAM

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

SUNBEAM

Stack

LIFO

1. Add first
   Delete first
2. Add last
   Delete lats

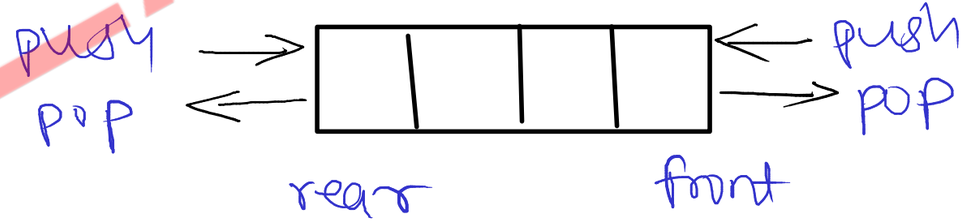
Queue

FIFO

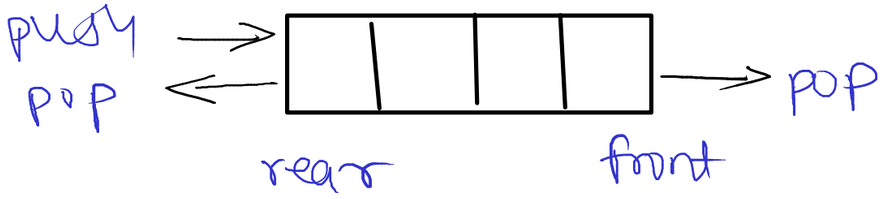
1. Add first
   Delete last
2. Add last
   Delete first

Deque

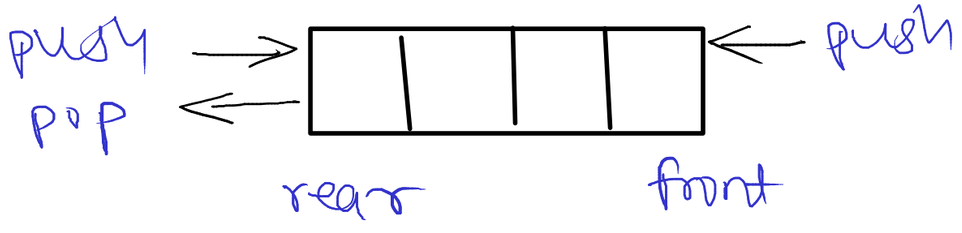
(Double Ended Queue)



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

SUNBEAM