Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

Trainer - Devendra Dhande

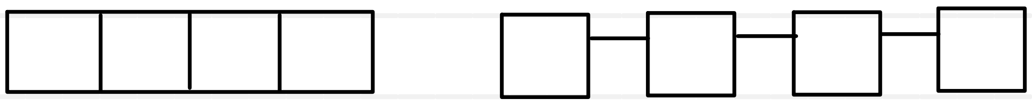
Email – devendra.dhande@sunbeaminfo.com

Data Structure

- organising data inside memory for efficient processing along with operations like add, delete, search, etc which can be performed on data.
- eg stack - push/pop/peek

Physical Data structures

Array   Linked List



Logical Data structure

stack, Queue, tree, heap, Graph

- data structures are used to achieve
  - Abstraction
    - data & orgnisation of data is hidden from outside
    - Abstract Data Types (ADT)
  - Reusability
    - data structures can be used to implement other data structures & to solve few algorithms
  - Efficiency
    - efficiency is measured in two parameters
      time - required to execute
      space - required to execute inside memory.

Types of data structures

(Basic)

Linear data structures

- data is organised sequentially/ linearly

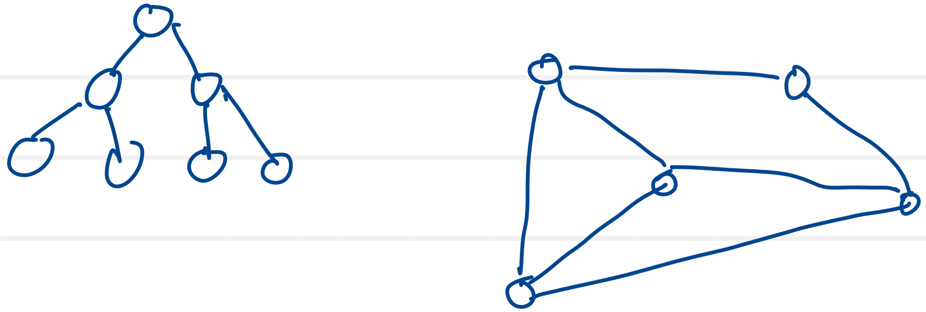
- data can be accessed sequentially

e.g. Array, struct/class, Stack, Queue, Linked List

(Advanced)

Non linear data structures

- data is organised in multiple levels (hierarchy)



- data can not be accessed sequentially

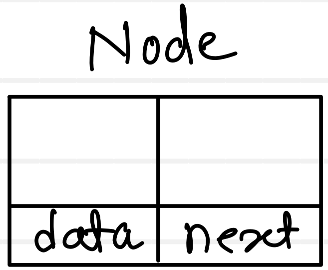
e.g. tree, Graph, heap

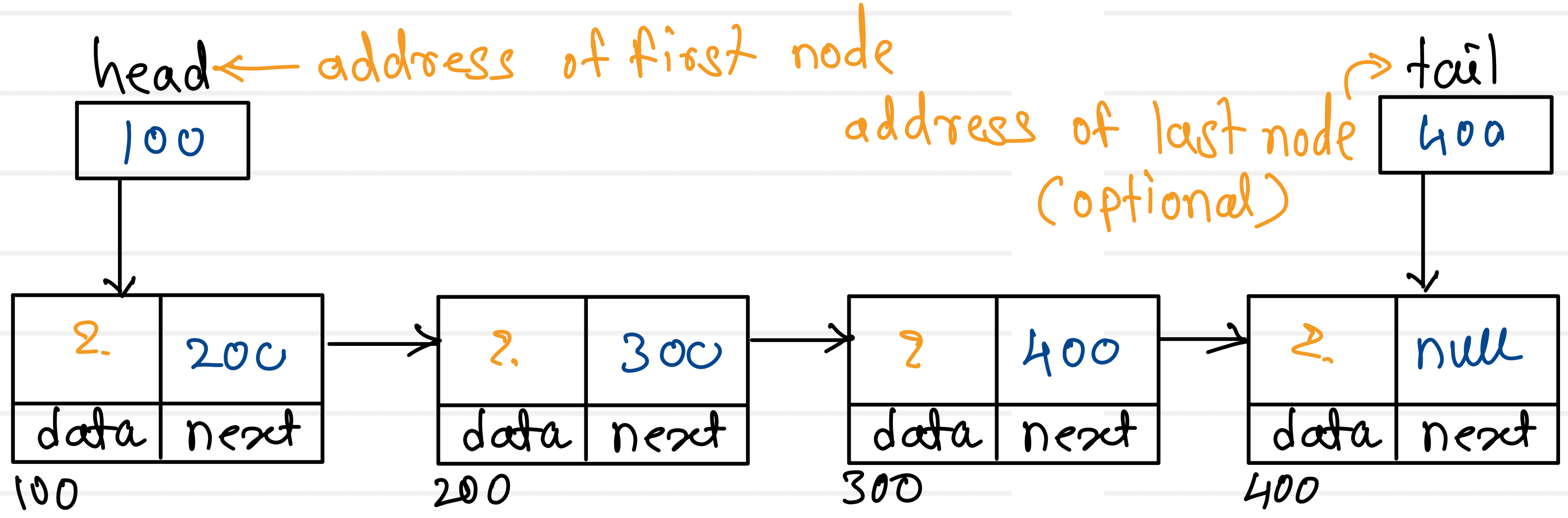
Hash table/Map

Linked List

- Linked list is a linear data structure
- collection of similar type of data
- every data keeps address of next data
- element of linked list is called as "Node".

- every node has two parts:
  - data : actual data
  - link/next : address of next data





Linked List

Operations

1. Add first
2. Add last
3. Add position (insert)

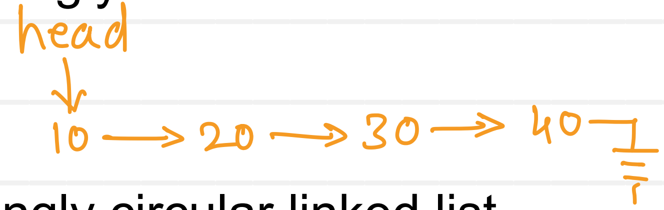
1. Delete first
2. Delete last
3. Delete position

1. Display (traverse) (forward/ backward)

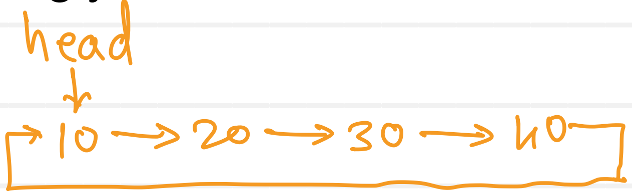
1. Search
2. Sort
3. Reverse

Types

1. Singly linear linked list

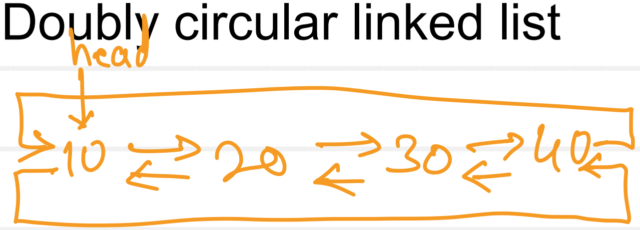


2. Singly circular linked list



3. Doubly linear linked list

4. Doubly circular linked list



Linked List

Node :
data : int, double, string, class, enum....
next : reference of next node

```
class Node {   ← self referential class
    int data;
    Node next;
}
```

why inner ?
- private fields of inner class (Node) will be directly accessible into List class

why static ?
- to restrict access of private fields of outer class into inner class

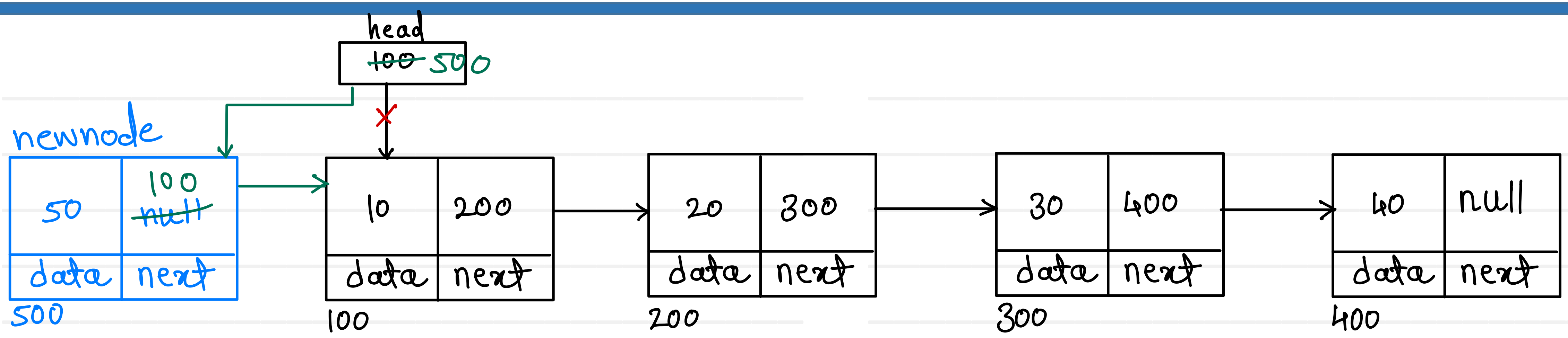
```
class List {
    static class Node {
        int data;
        Node next;
    }
    Node head, tail;
    int size ;
    public List( ) { ... }
    public void addNode( ) { ... }
    public void deleteNode( ) { .... }
    public void traverse( ) { .... }
    public Node searchNode( ) { ... }
}
```

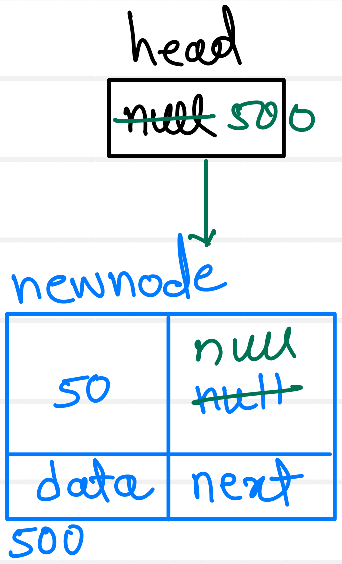
```
class List {
    static class Node {
        -
        -
    }
    head, tail;
    . . .
    class Iterator {
        Node curr;
    }
};
```

Why static ?

↳ don't have any dependency of outer class to create object of inner.

Singly linear Linked List - Add first

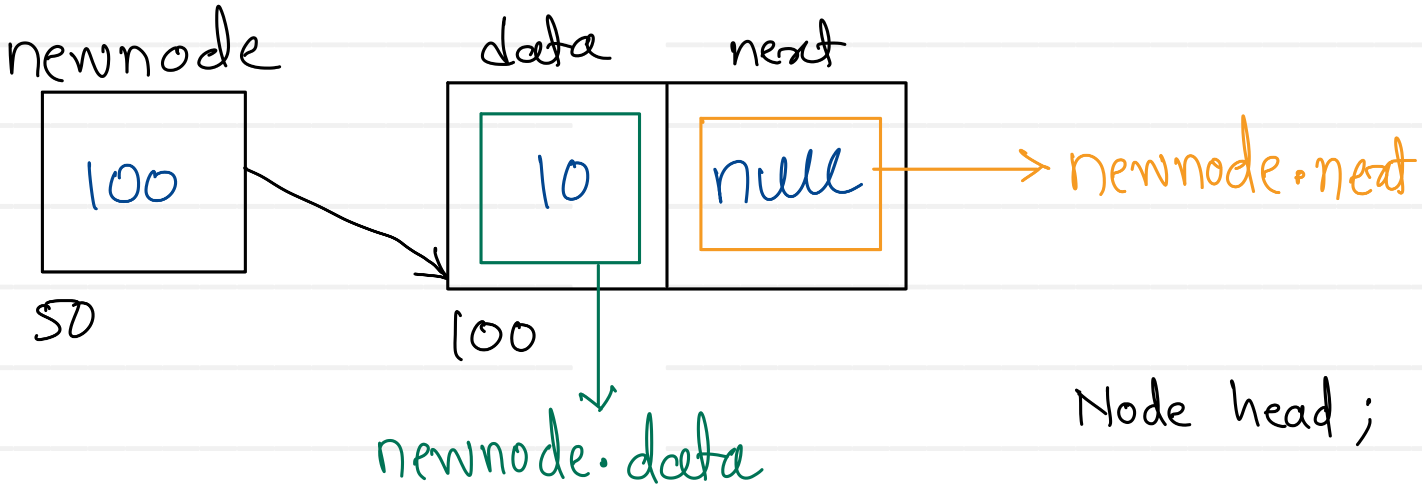




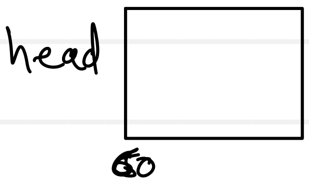
newnode.next = head
head = newnode

1. create a newnode
2. add first node into next of newnode
3. move head on newnode

Node newnode = new Node(10);



Node head;

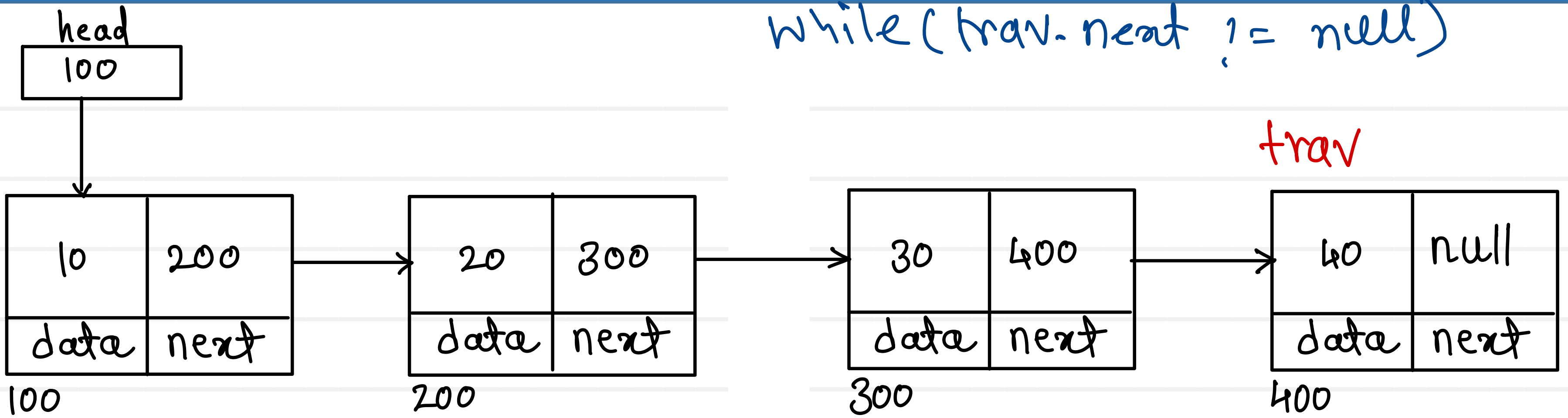


newnode.data = 2. ← writing

2. = newnode.data ← reading

Singly linear Linked List - Display

while(trav.next != null)

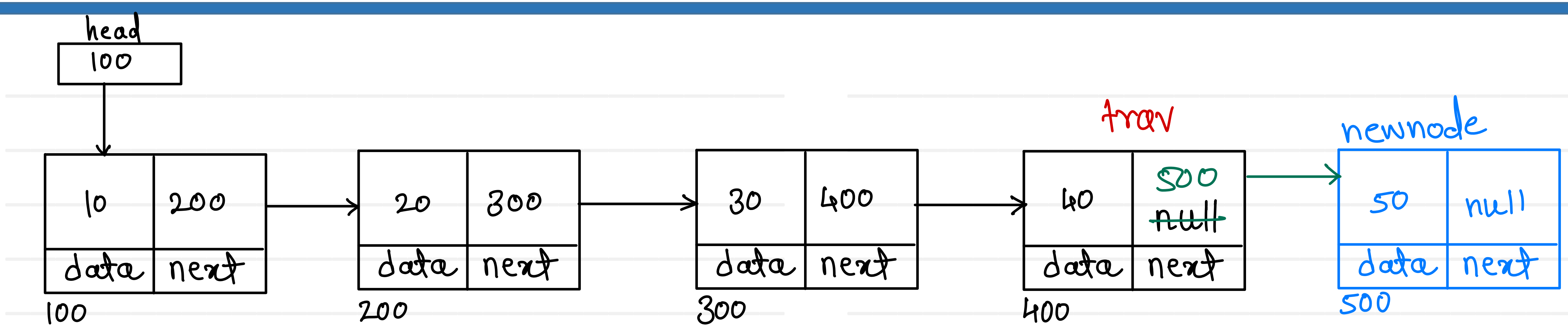


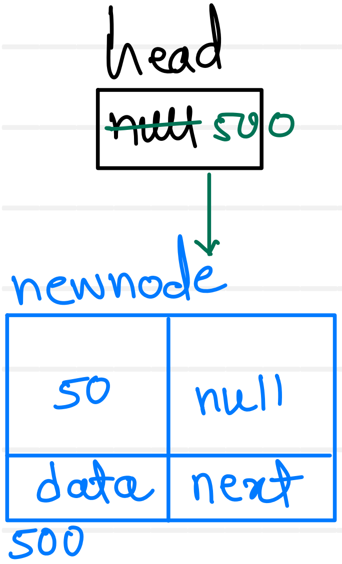
1. create trav & start at head (first node)
2. visit/print data of current node
3. go on next node
4. repeat above two steps for every node

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

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

Singly linear Linked List - Add last

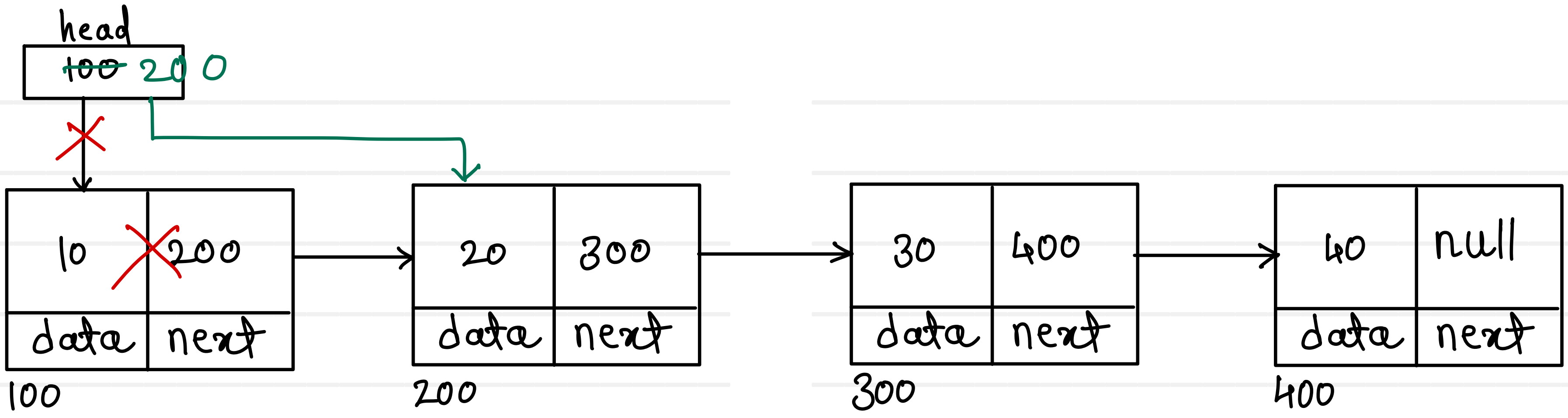


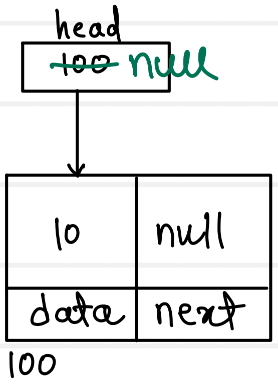



```
Node trav = head;
while(trav.next != null)
    trav = trav.next
```

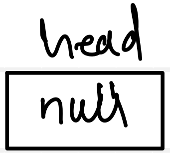
1. Create a newnode
2. if list is empty,
   add newnode into head itself
3. if list is not empty,
   a. traverse till last node
   b. add newnode into next of last node

Singly linear Linked List - Delete first



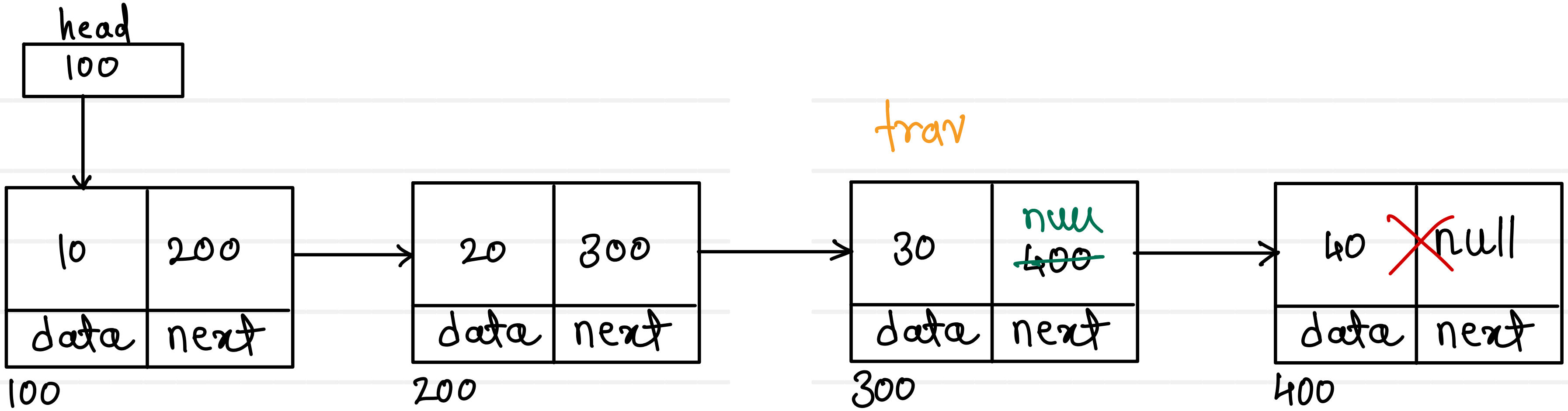


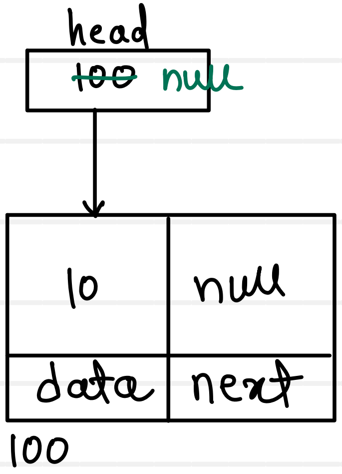
1. if list is empty , return
2. if list is not empty,
   a. move head on second node



head = head.next

Singly linear Linked List - Delete last





trav	trav.next	trav.next.next
100	200	300
200	300	400
300	400	null

address of second last node

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

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

→ trav = null

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

→ trav = last node

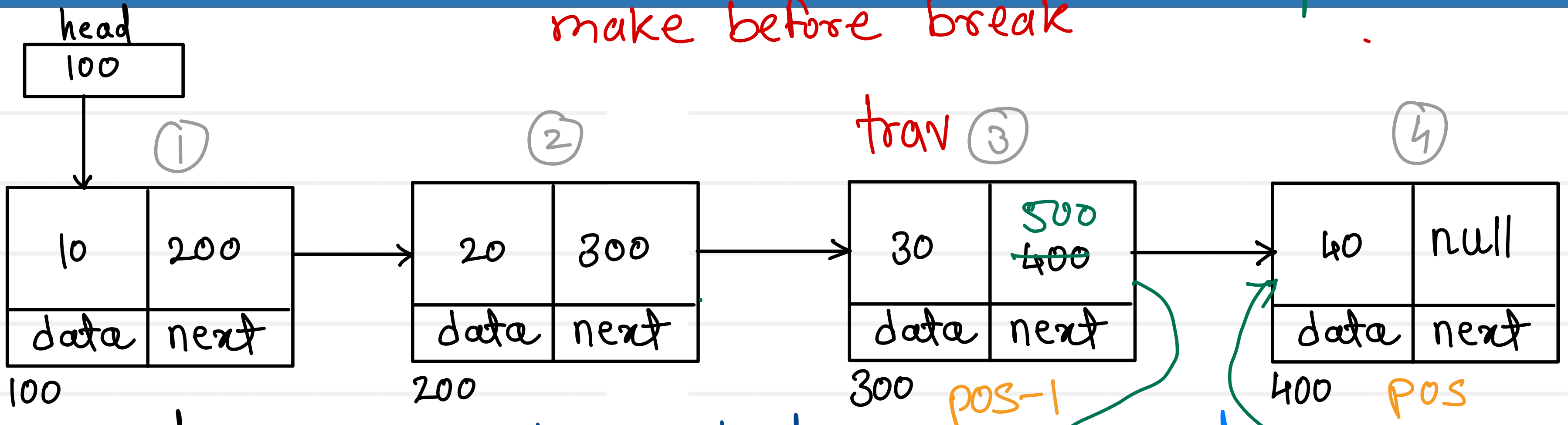
```
Node trav = head;
while (trav.next.next != null)
    trav = trav.nc -;     xt
```

→ trav = second last node

Singly linear Linked List - Add position

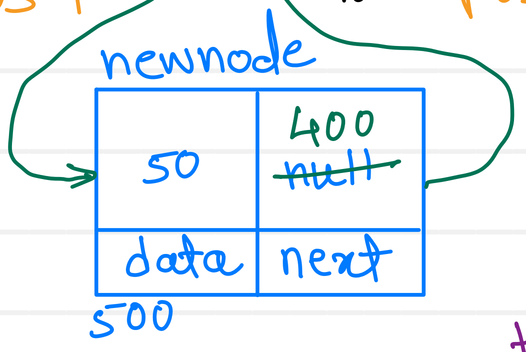
make before break

pos = 4



1. traverse till pos-1 node
2. add pos node into next of newnode
3. add newnode into next of pos-1 node

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



pos = 4

trav	i	i<3
100	1	T
200	2	T
300	3	F

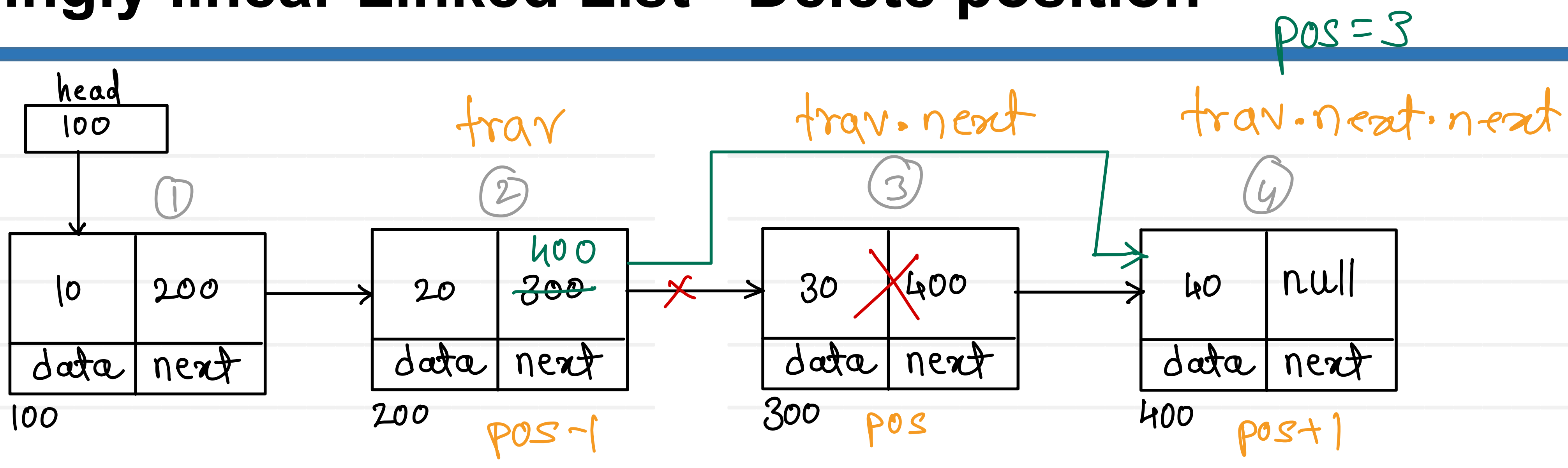
pos = 5

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

pos = 6

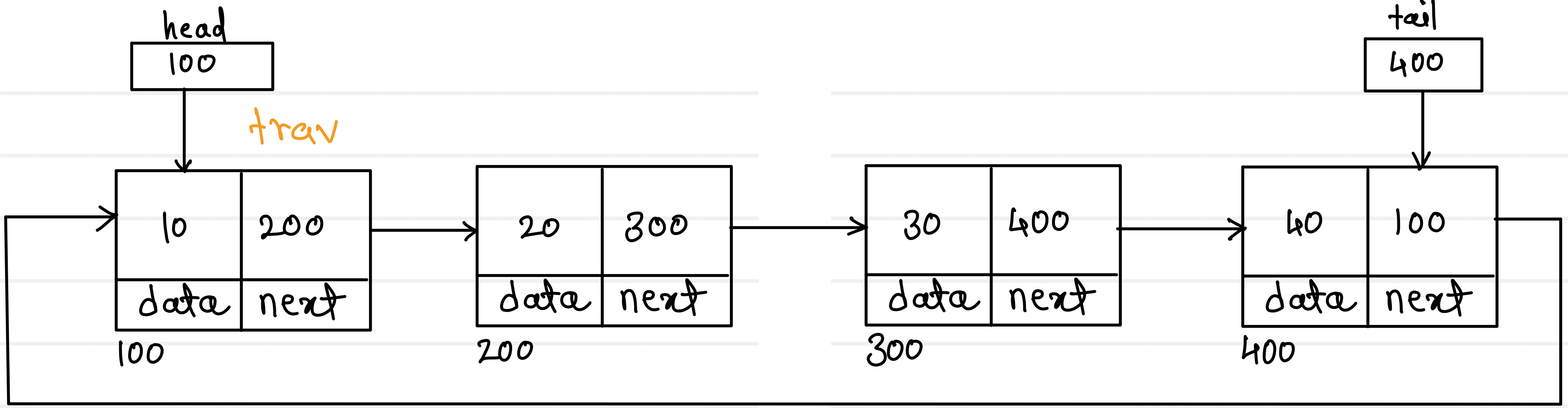
trav	i	i<5
100	1	T
200	2	T
300	3	T
400	4	T
null	5	F

Singly linear Linked List - Delete position



1. traverse till pos-1 node
2. add pos+1 node into next of pos-1 node

Singly Circular Linked List - Display

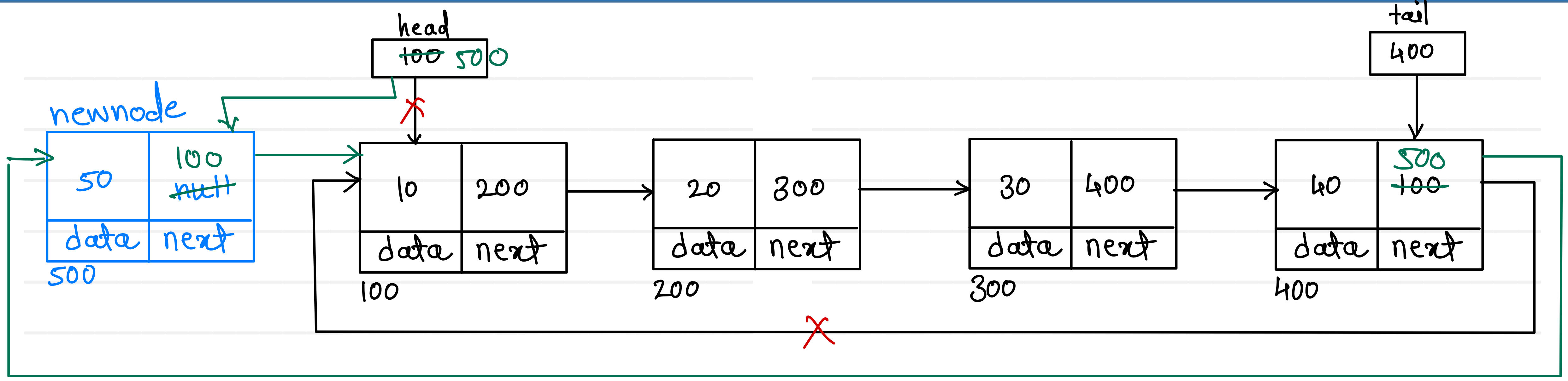


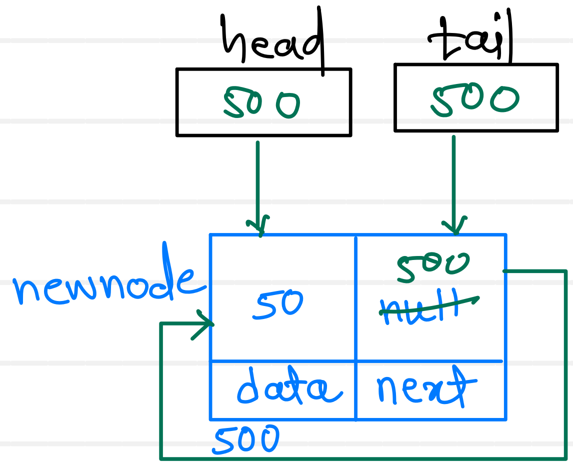
1. create trav & start at first node
2. print current node data
3. go on next node
4. repeate above 2 steps for each node

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

trav
100
200
300
400
~~100~~

Singly Circular Linked List - Add first

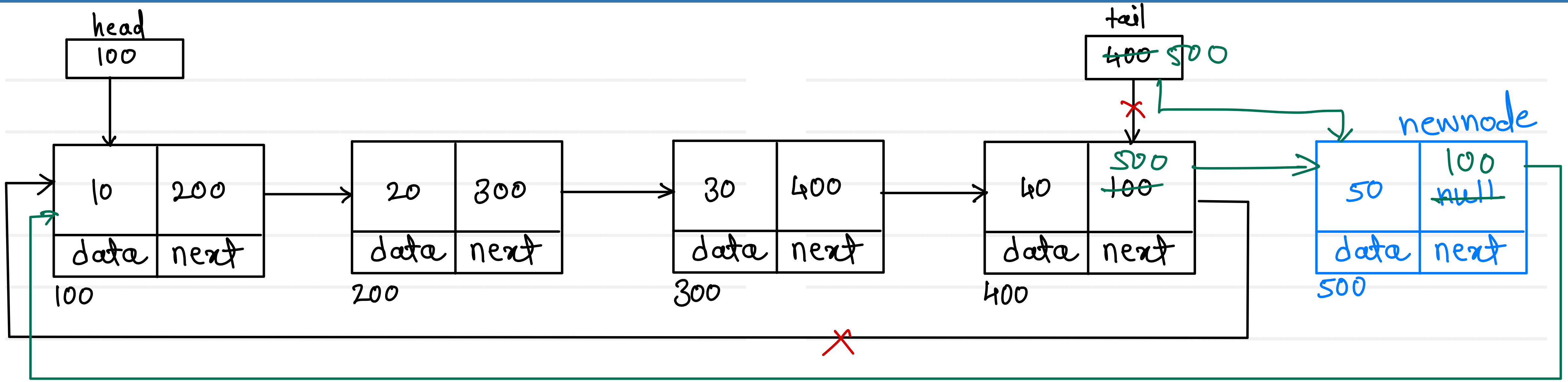


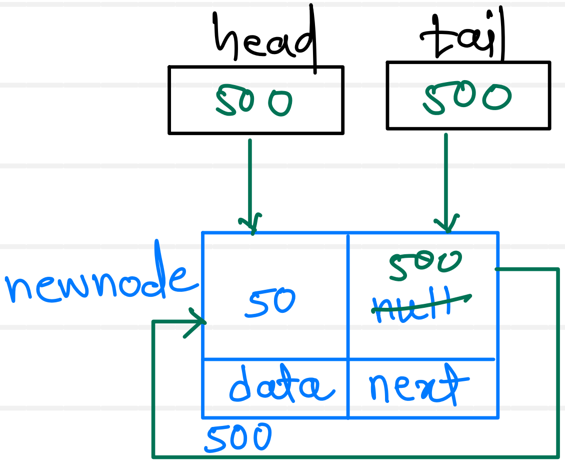


if list is empty,
a. add newnode into head & tail
b. make list circular

if list is not empty,
a. add first into next of newnode
b. add newnode into next of last node
c. move head on newnode

Singly Circular Linked List - Add last

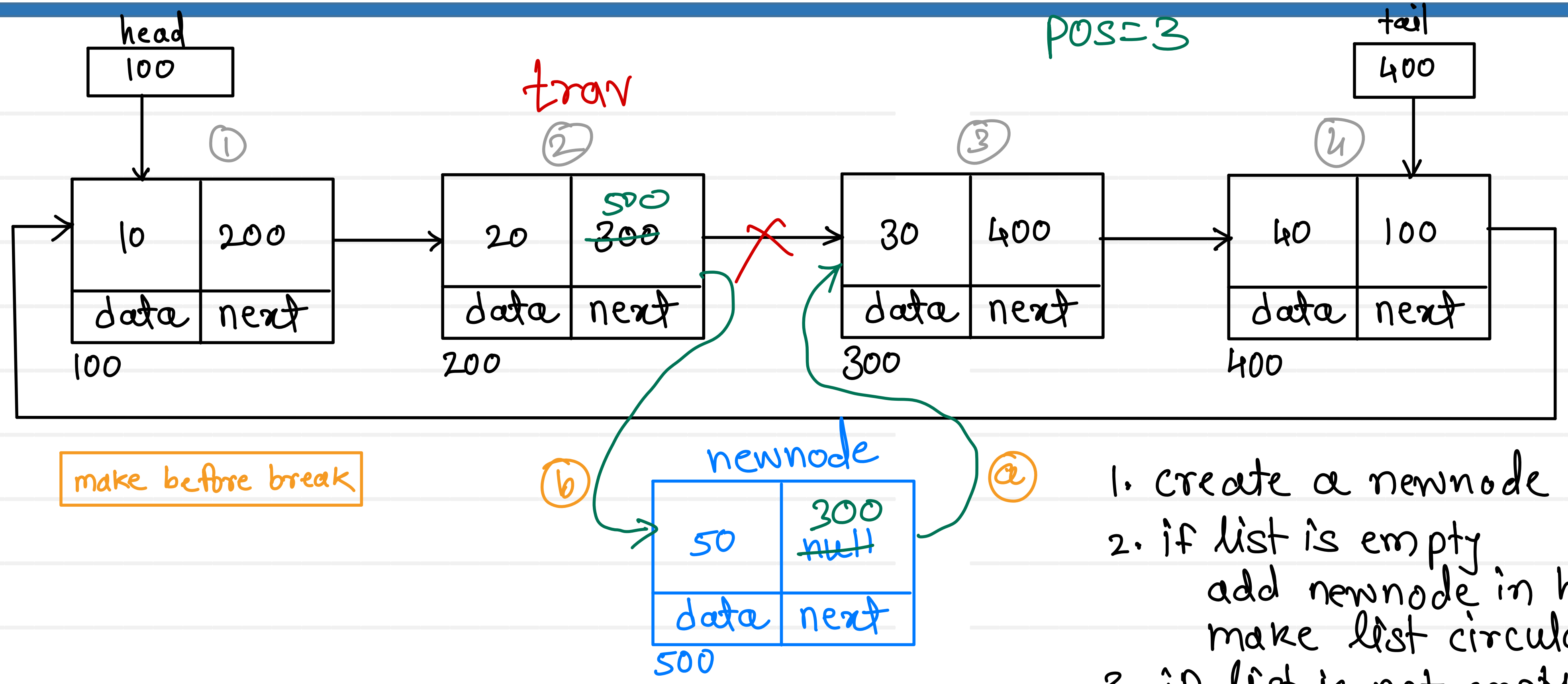




if list is empty,
a. add newnode into head & tail
b. make list circular

if list is not empty,
a. add first into next of newnode
b. add newnode into next of last node
c. move tail on newnode

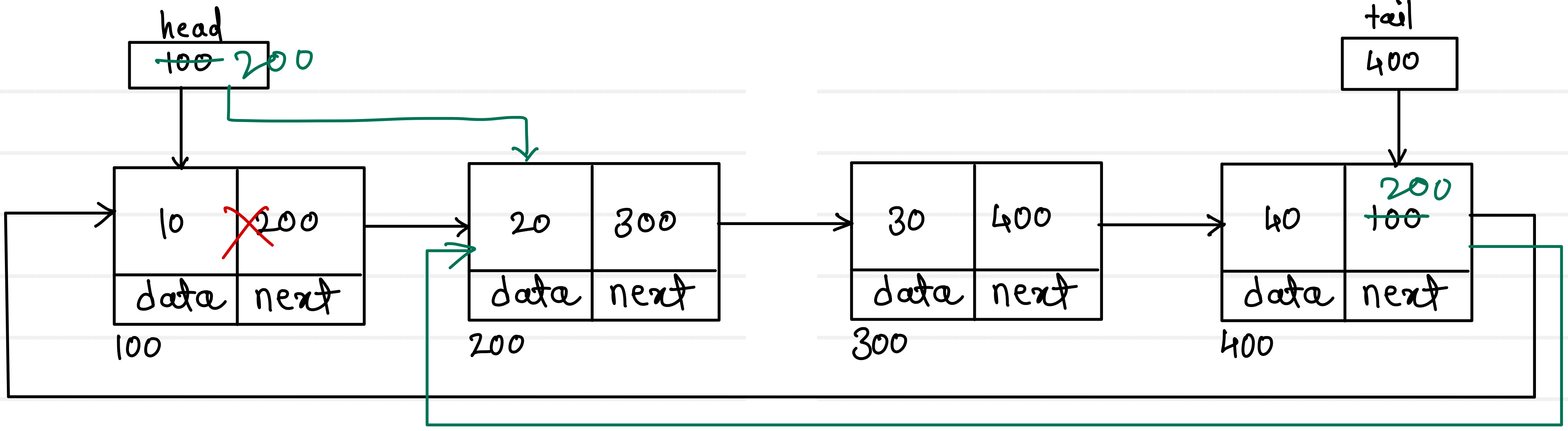
Singly Circular Linked List - Add position

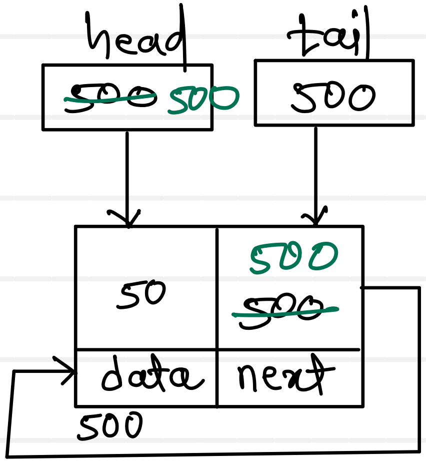


valid positions:
1 to size+1
invalid positions:
pos < 1
pos > size+1

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

Singly Circular Linked List - Delete first

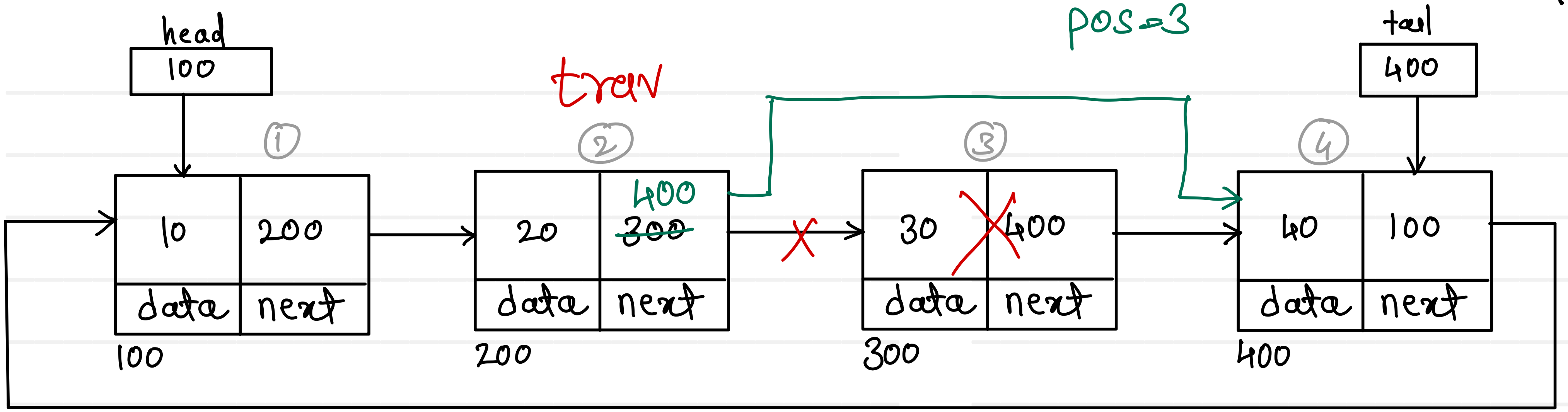




tail.next = head.next;
head = head.next;

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

Singly Circular Linked List - Delete position



valid positions :
1 to size
invalid positions :
pos < 1
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

SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com