Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

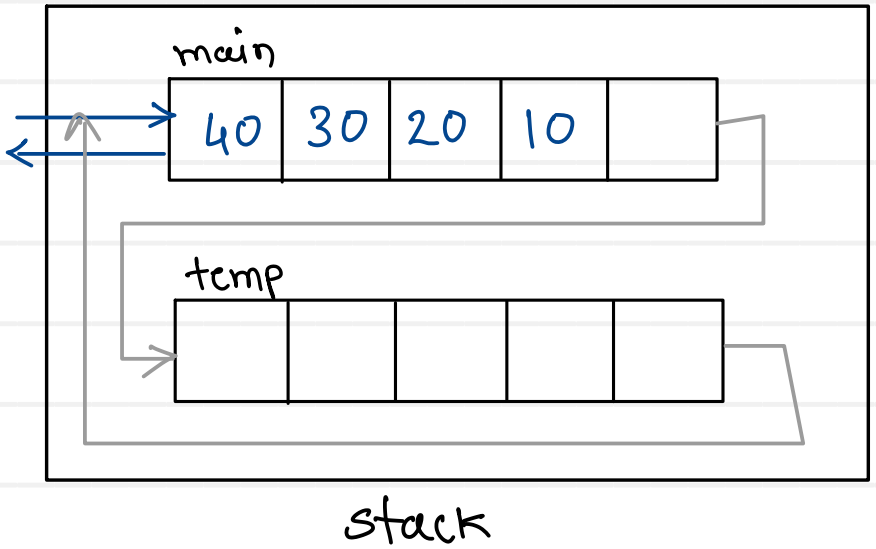
Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Create stack using queue

LIFO FIFO

Push order : 10,20,30,40



Pop order : 40 , 30 , 20 , 10

java.utils

Queue - Interface

```
Queue<> q = new LinkedList<>();
q.offer(_)   <- push op
q.poll()     <- pop op
```

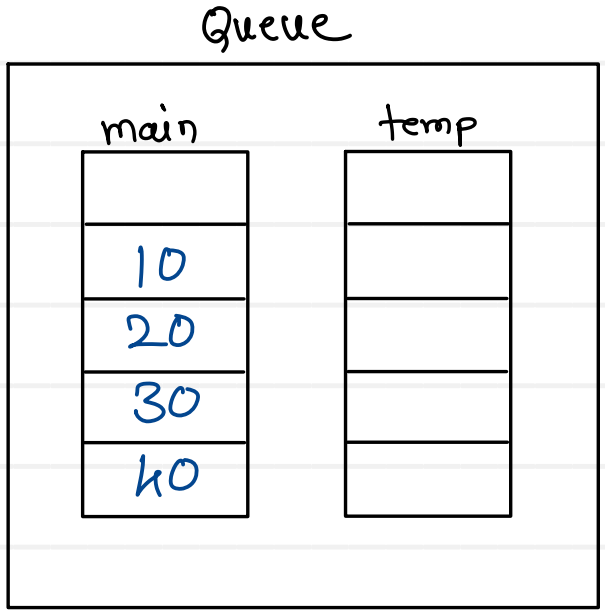
```
push():
    while(!mq.isEmpty())
        tq.offer(mq.poll());
    mq.offer(value);
    while(!tq.isEmpty())
        mq.offer(tq.poll());
pop():
        mq.poll()
peek():
        mq.peek()
```

push:
itrs = n+n
T ∝ 2n
T(n) = O(n)

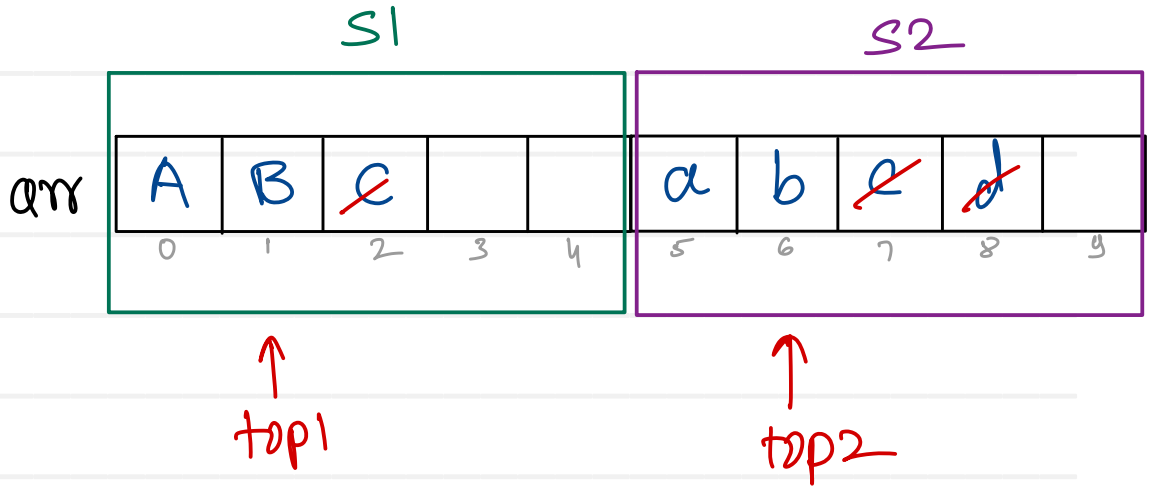
pop & peek:
T(n) = O(1)

Create queue using stack

Push order : 10 ,20, 30, 40



How to Implement two stacks using array efficiently?



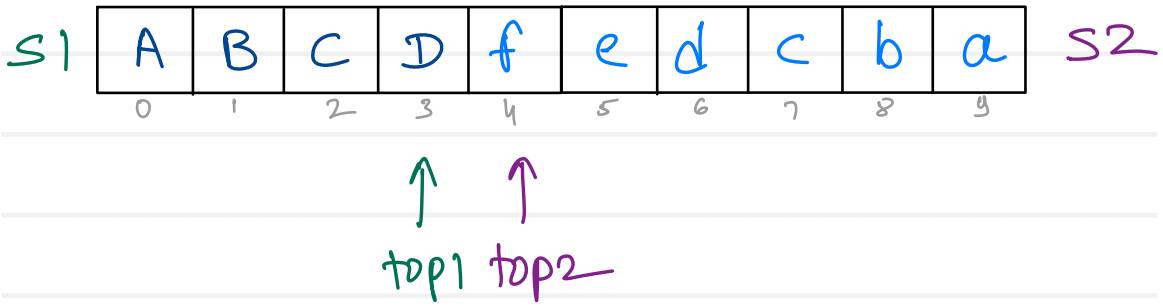
- two stacks with fixed capacity

isEmpty : top1 == -1 top2 == length/2

isFull : top1 == length/2 top2 == length-1

How to Implement two stacks using array efficiently?

- two stacks with variable capacity
- more memory efficient



Stack1

init : top1 = -1
push : top1 ++
arr[top1] = val
pop : top1 --
isEmpty : top1 == -1
isfull : top1 == top2-1

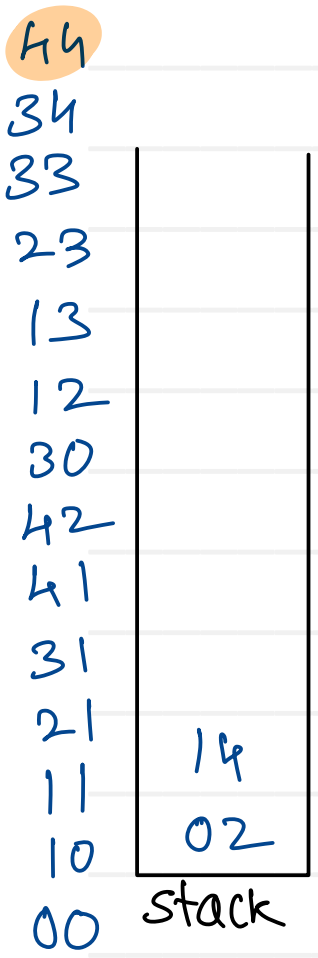
Stack2

init : top2 = length
push : top --
arr[top] = val
pop : top++
isEmpty : top2 == length
isfull : top1 == top2-1

Rat in a maze problem ← backtracking

Algorithm:

1. push src cell on stack & mark it.
2. while stack is not empty :
   - 2.1 pop a cell from stack (cur)
   - 2.2 if(cur cell == dest cell)
     return true;
   - 2.3 push all valid & unmarked neighbor cells on stack.
3. if destination is not reachable
   return false.



maze

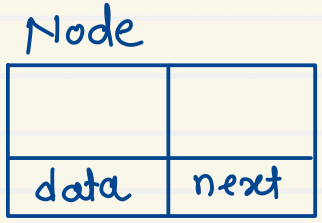
	0	1	2	3	4
0	S 00	01	02	03	04
1	10	11	12	13	14
2	20	21	22	23	24
3	30	31	32	33	34
4	40	41	42	43	d 44

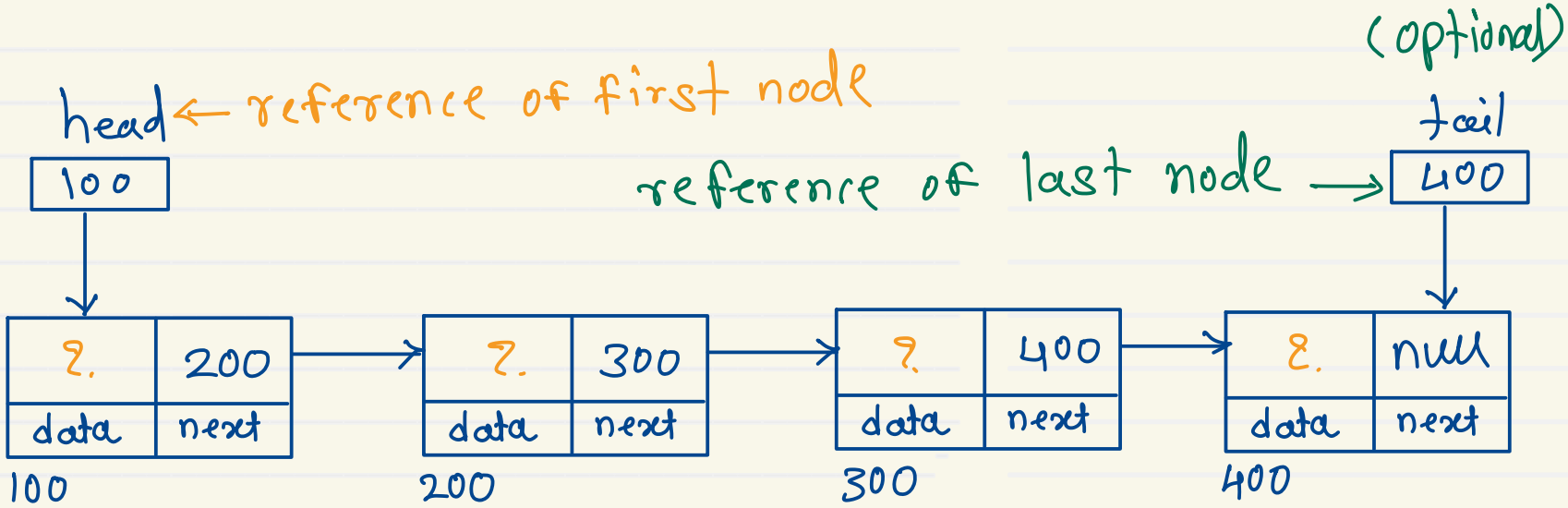
marked → (true/false)

	0	1	2	3	4
0	00	01	02	03	04
1	10	11	12	13	14
2	20	21	22	23	24
3	30	31	32	33	34
4	40	41	42	43	44

Linked List

- Linked list is a linear data structure
- Linked list is list of items linked together
- Each item in linked list is called as Node
- Each node contains data and pointer / reference to the next node



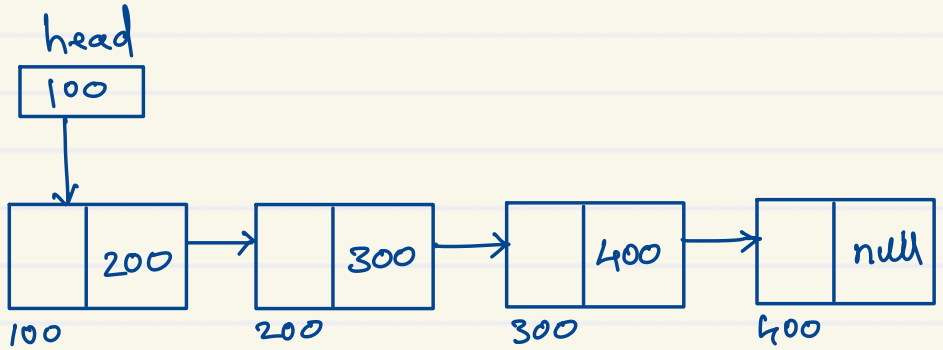


Operations

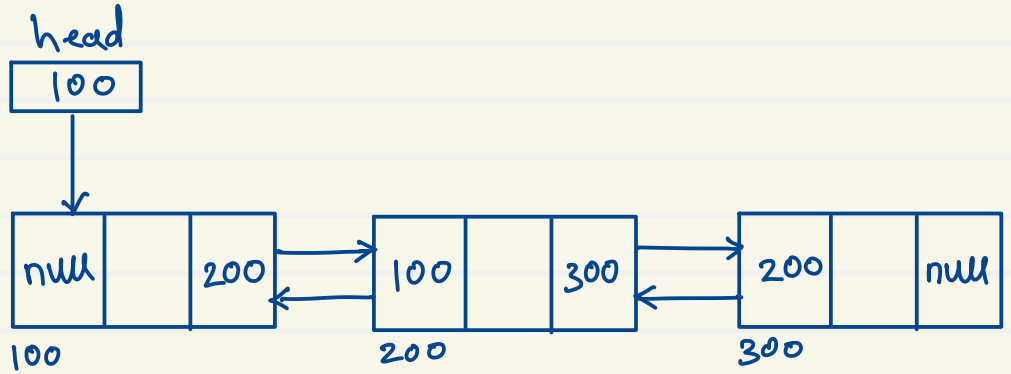
1. Add first
2. Add last
3. Add position
4. Delete first
5. Delete last
6. Delete position
7. Delete all
8. Display

Linked List - Types

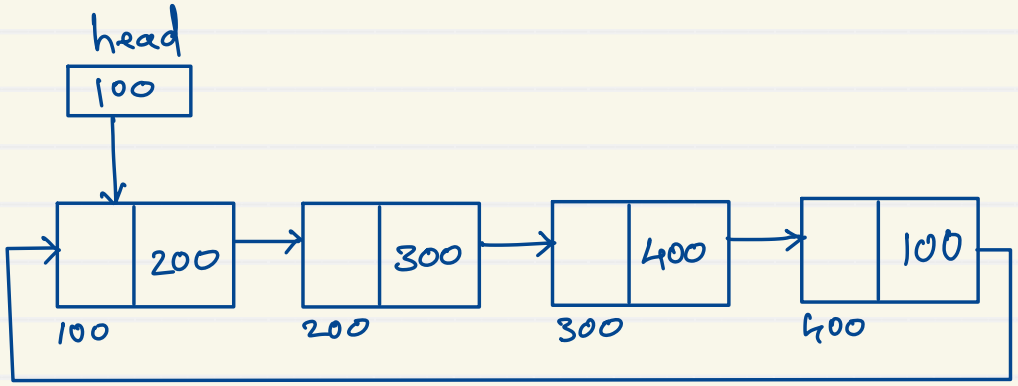
1. Singly linear linked list



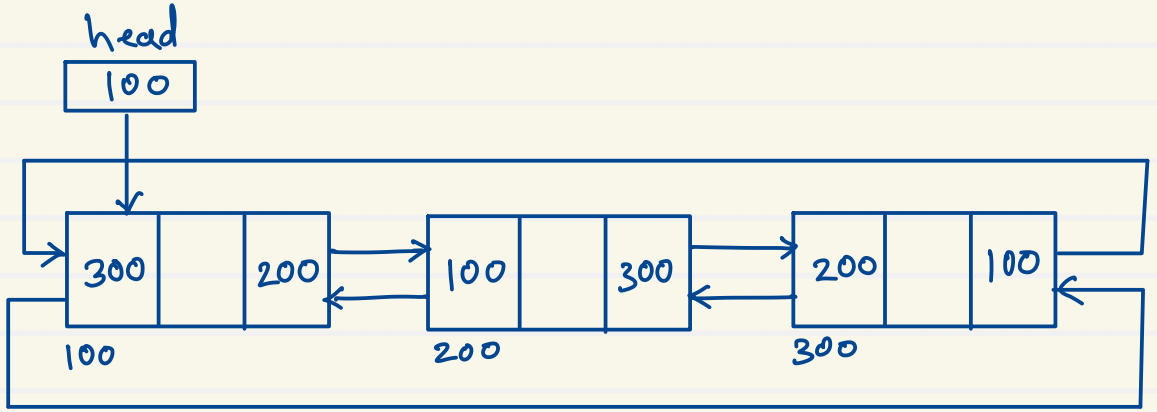
3. Doubly linear linked list



2. Singly circular linked list



4. Doubly circular linked list



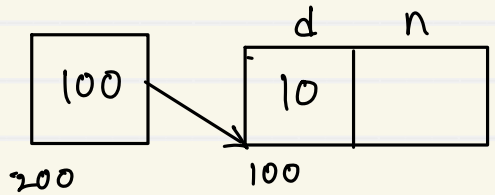
Linked List

Node :
data – int, char, double, class
next – reference of next node

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

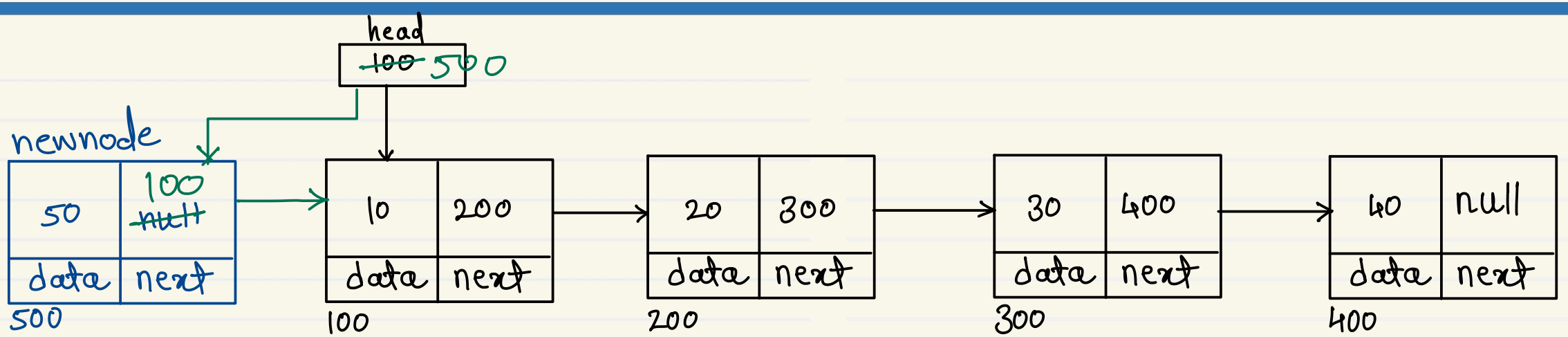
- In java, Node class is inner class of LinkedList class
- In c++, Node class is friend class of LinkedList class.

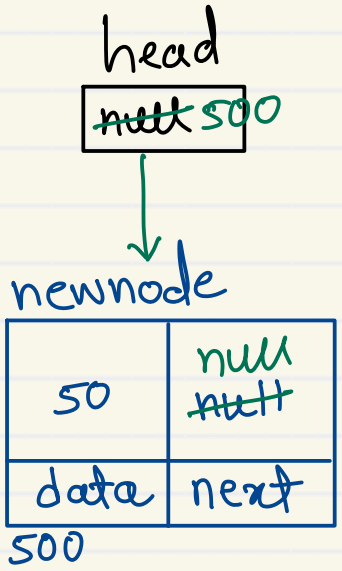
```
Node n = new Node(10)
```




```
class LinkedList {
    Node head;
    Node tail;
    int count;
    public LinkedList() {...}
    public add() {...}
    public delete() {...}
    public display() {...}
    public search() {...}
    public deleteAll() {...}
}
```

Singly linear Linked List - Add first





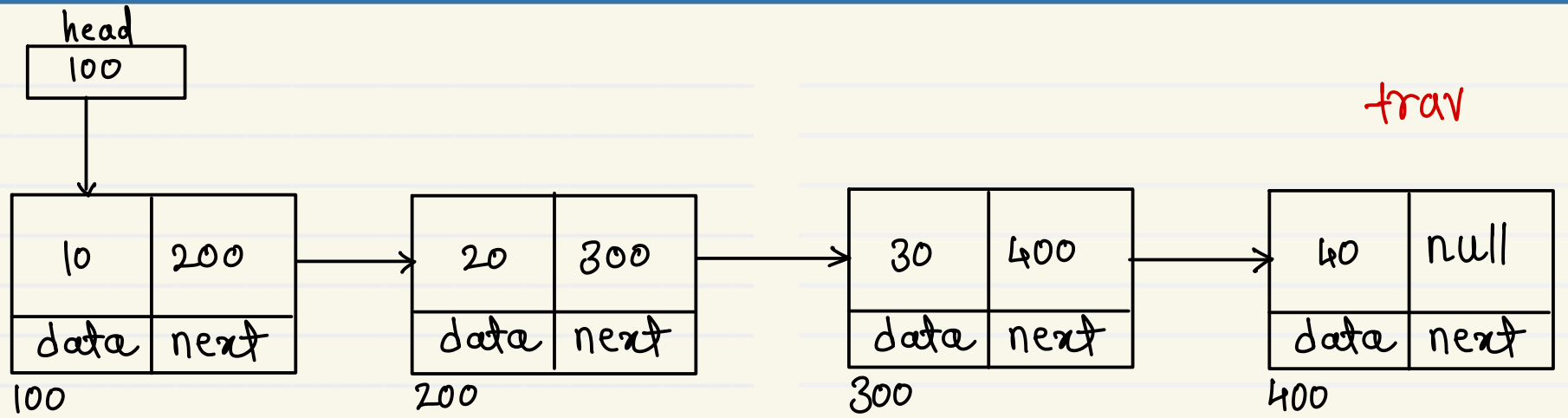
newnode.next = head
head = newnode

1. create a newnode with given data
2. add first node into next of newnode
3. update head by newnode

$T(n) = O(1)$

Singly linear Linked List - Display

traverse – visiting each node at least once

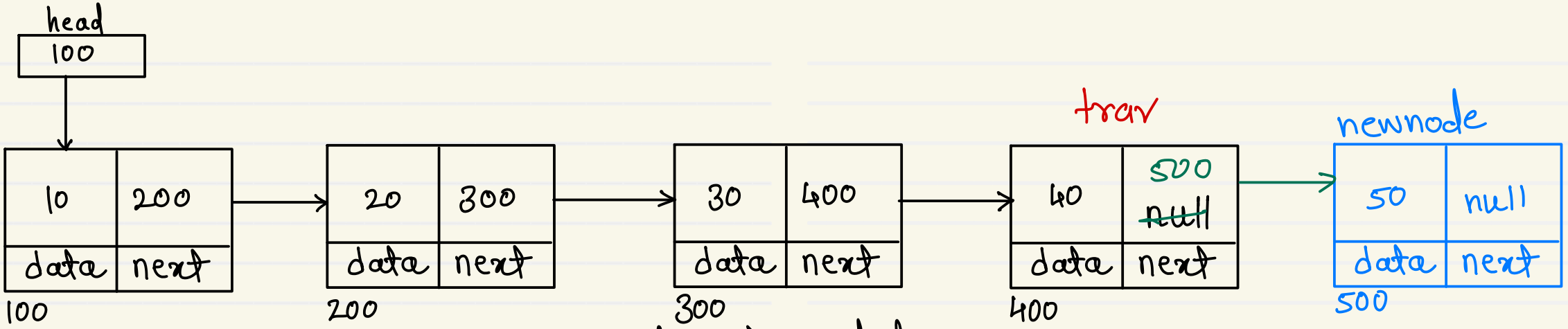


trav	trav.data	trav.next
100	10	200
200	20	300
300	30	400
400	40	null
null		

1. Create reference & start at first node
2. visit/print current node
3. go on next node
4. repeat step 2 & 3 till last node

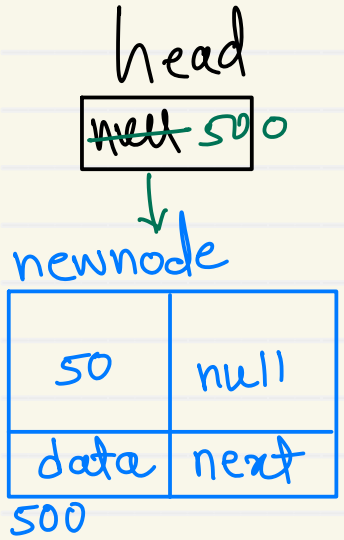
$T(n) = O(n)$

Singly linear Linked List - Add last



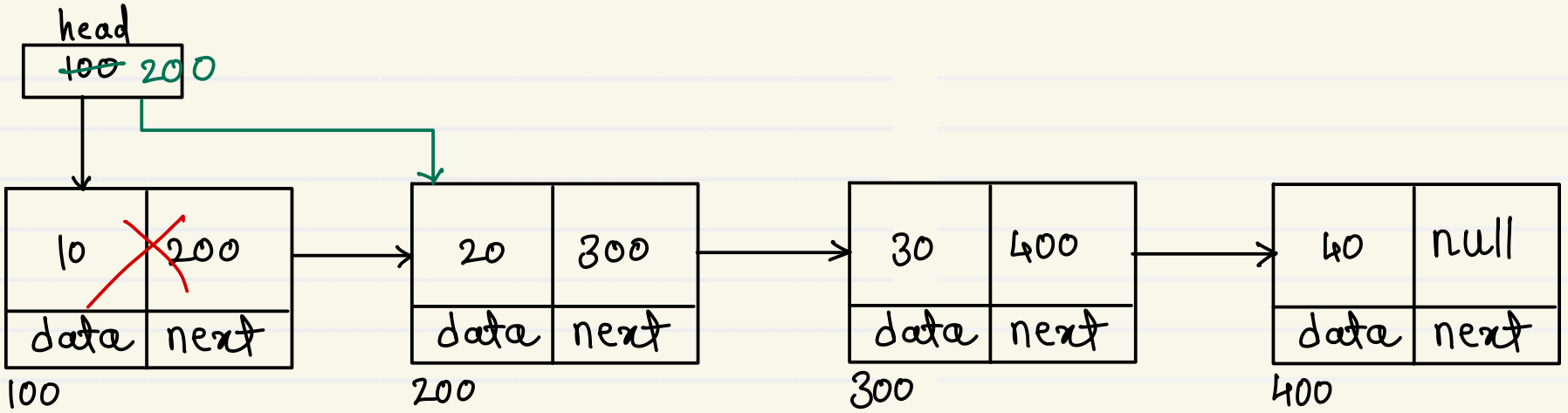
0. create newnode with data
1. if list is empty,
   add newnode into head itself
2. if list is not empty,
   a. traverse till last node
   b. add newnode into next of last node

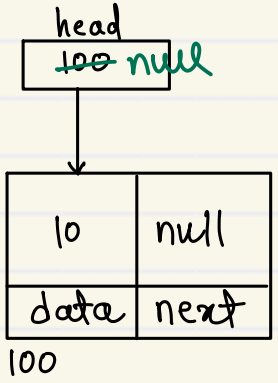
```
trav = head
while (trav.next != null)
    trav = trav.next;
```



$T(n) = O(n)$

Singly linear Linked List - Delete first



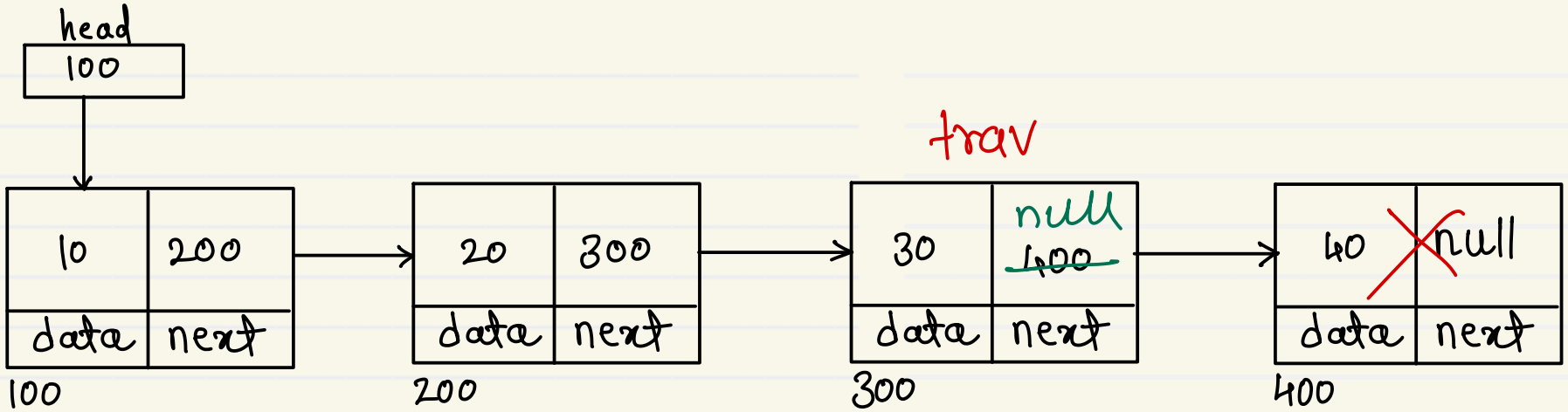


1. if list is empty, return
2. if list is not empty
   a. move head on second node

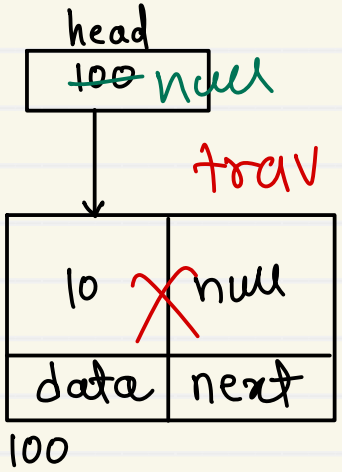
$T(n) = O(1)$

head = head.next

Singly linear Linked List - Delete last




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



1. if list is empty, return
2. if list has single node, delete it
3. if list has multiple nodes
   a. traverse till second last node
   b. make next of second last node null

$T(n) = O(n)$