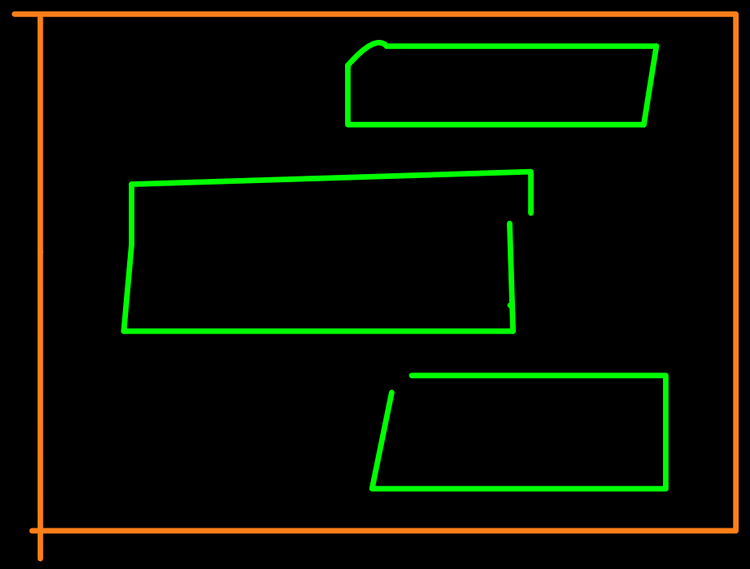
Problems With Arrays

↳ 1) An array always consumes contiguous memory location



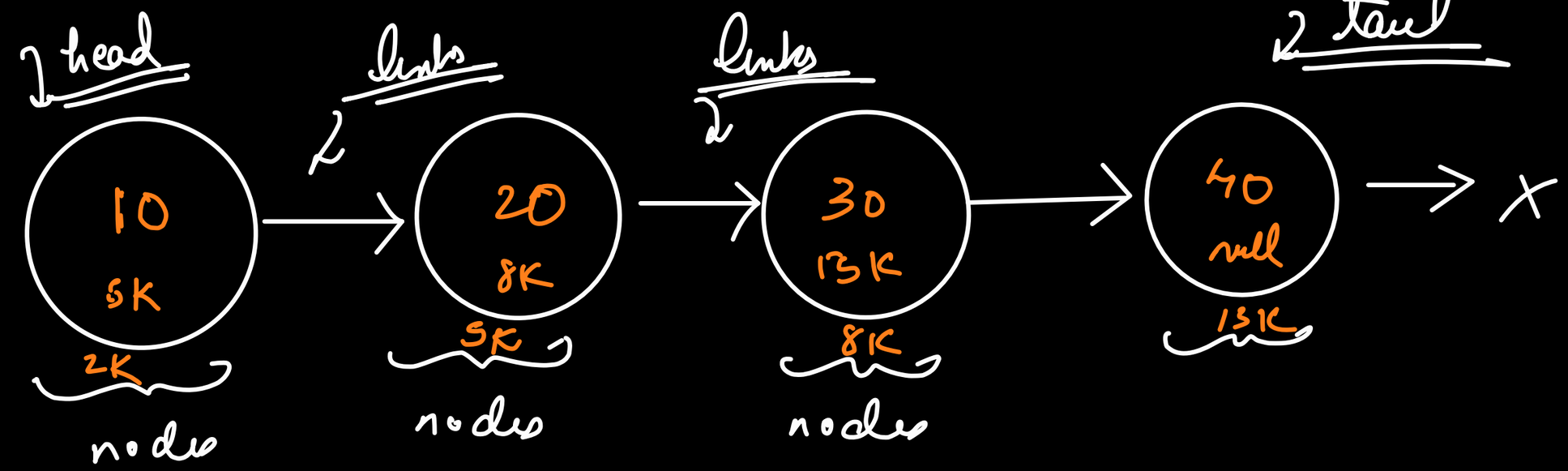
2) When we actually add a new element to an array it creates copies of itself for oddly new element

3) If we want to add an element in the start of an array → O(n)

Linked Lists

→ linear data structures

→ they form chain like structure

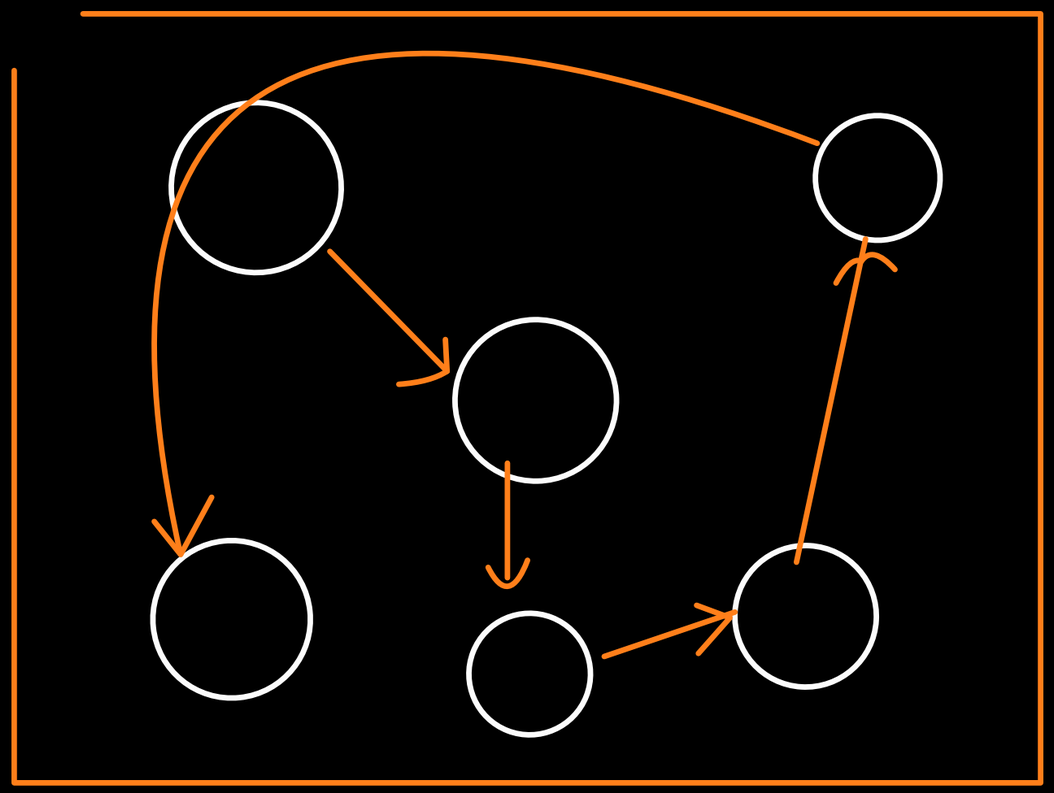


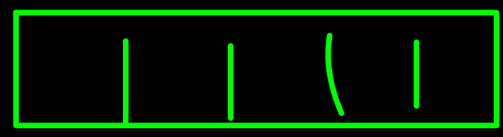
head → first node of the list

tail → last node of the list

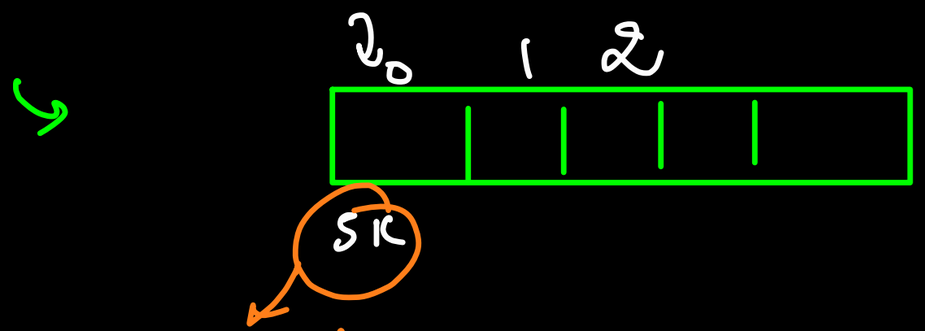
2 Node

in array, we used to have contiguous memory occupied.





linked list nodes are just simple objects that gets created at random avail memory loc. So they don't need to be contiguous.

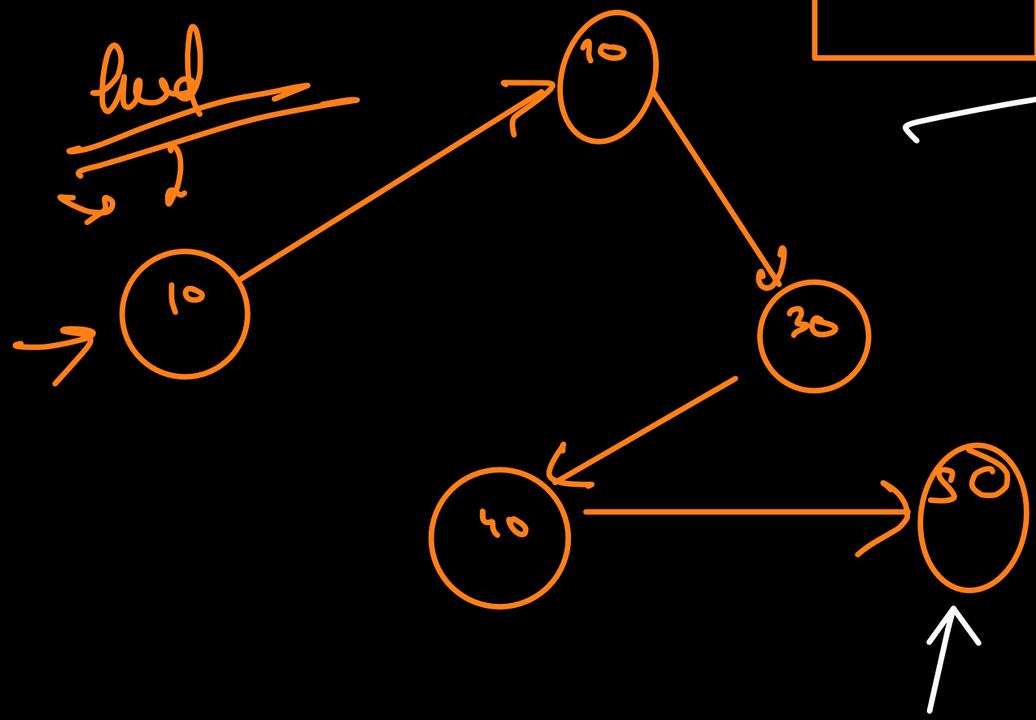


→ random access

→ arr[1] → O(1)

arr[2]

→ no random access



→ O(n)

head ← {

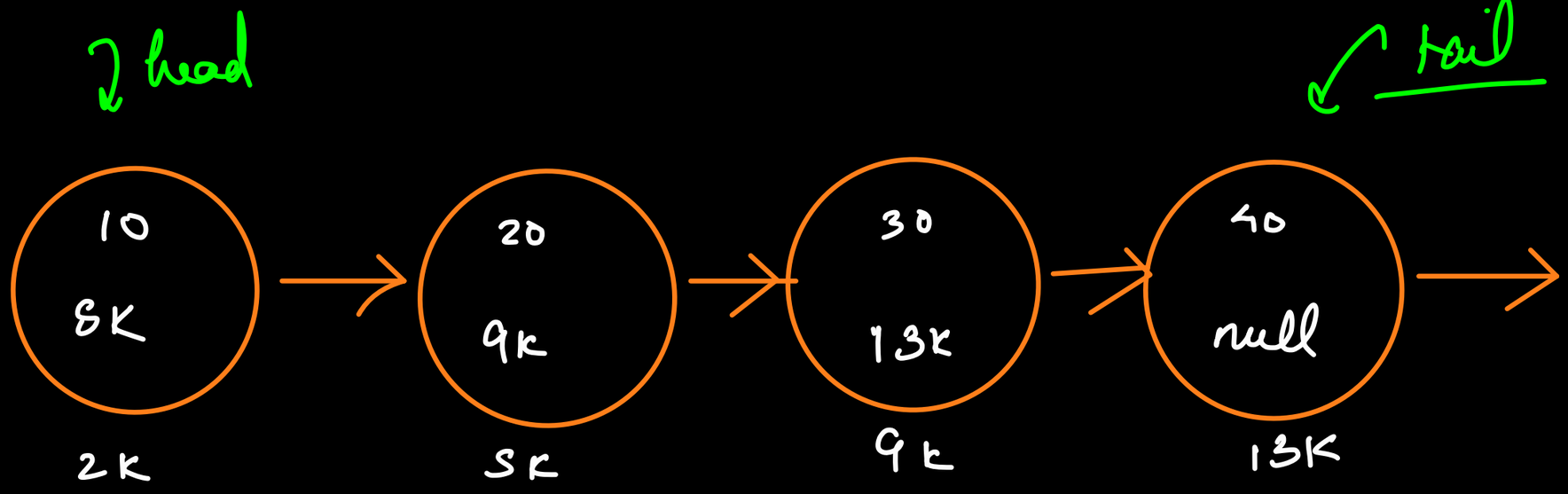
data : 10

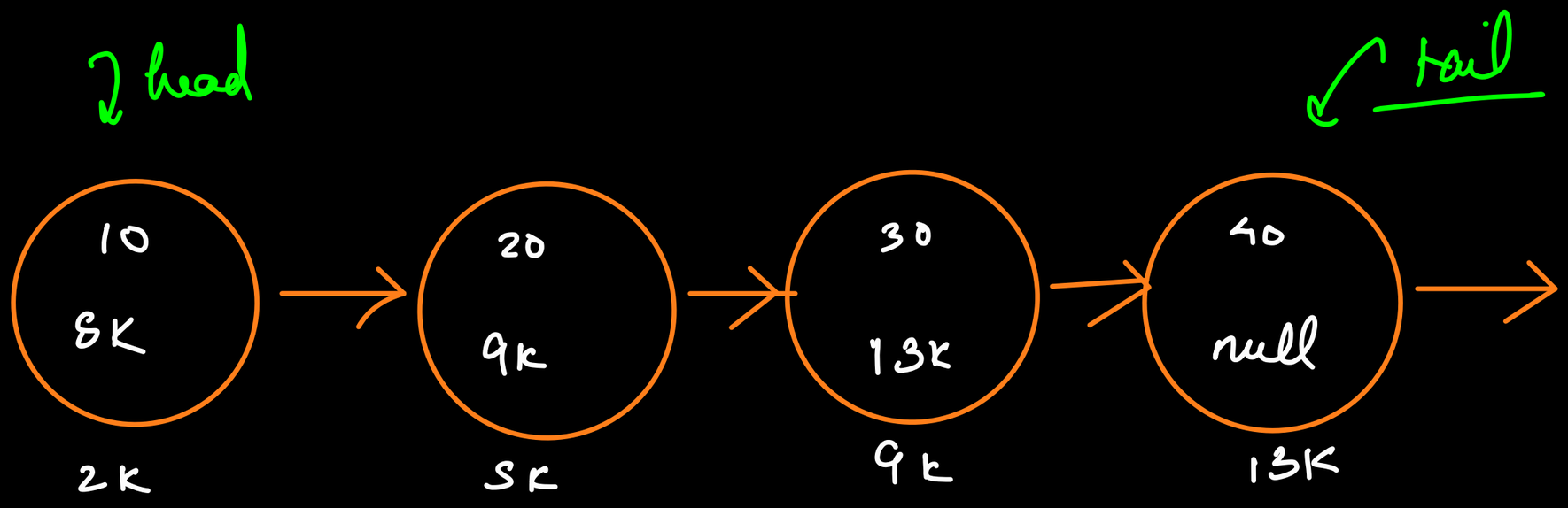
next : ——

}

- Add
  - addAtHead
  - addAtTail
  - addAt
- Remove
  - removeAtHead
  - removeATTail
  - removeAt
- access
  - getHead
  - getTail
  - getAt

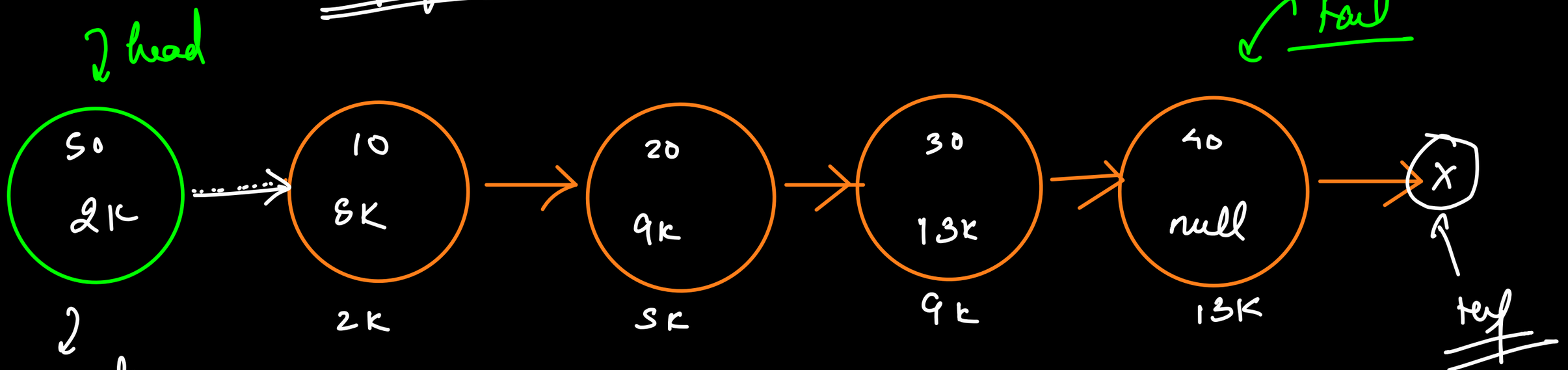
display





addAT Head → we want to add a new node behind the head & make this new node as the head of the list.

display



temp.data
temp = temp.next

addATHead (50)

→ Create a new node object with data as 50 & next property as null.

→ set the next property of the new_node as head.
(new_node.next = head)

→ Update the head. (head = new_node)

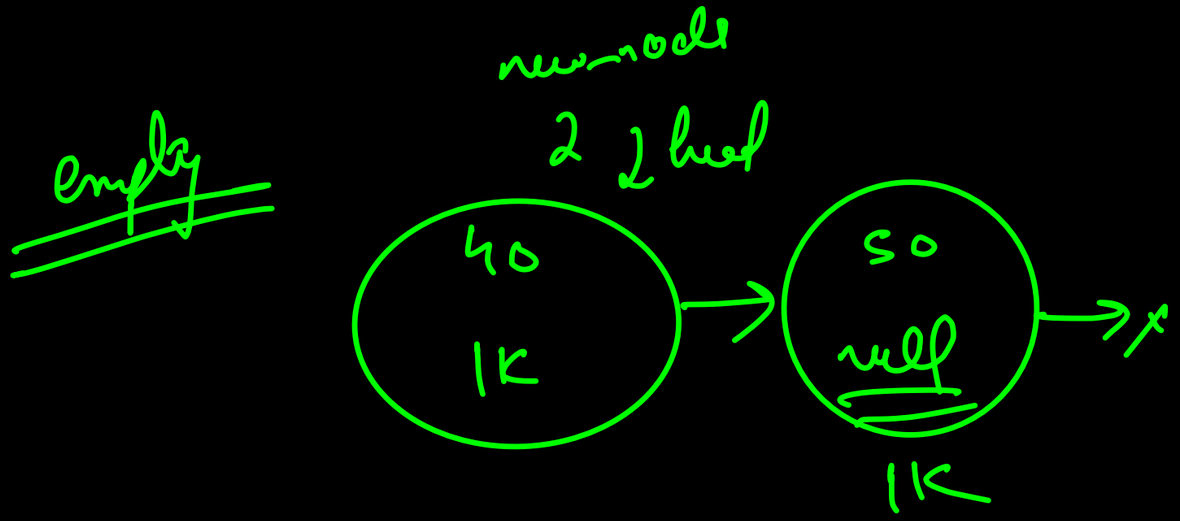
addAtHead (head, data) → O(1)

```
    new_node = createNode (data);

    new_node.next = head;

    head = new_node;

}
```



obj = { a : 10
b : 30 }

obj1 = obj

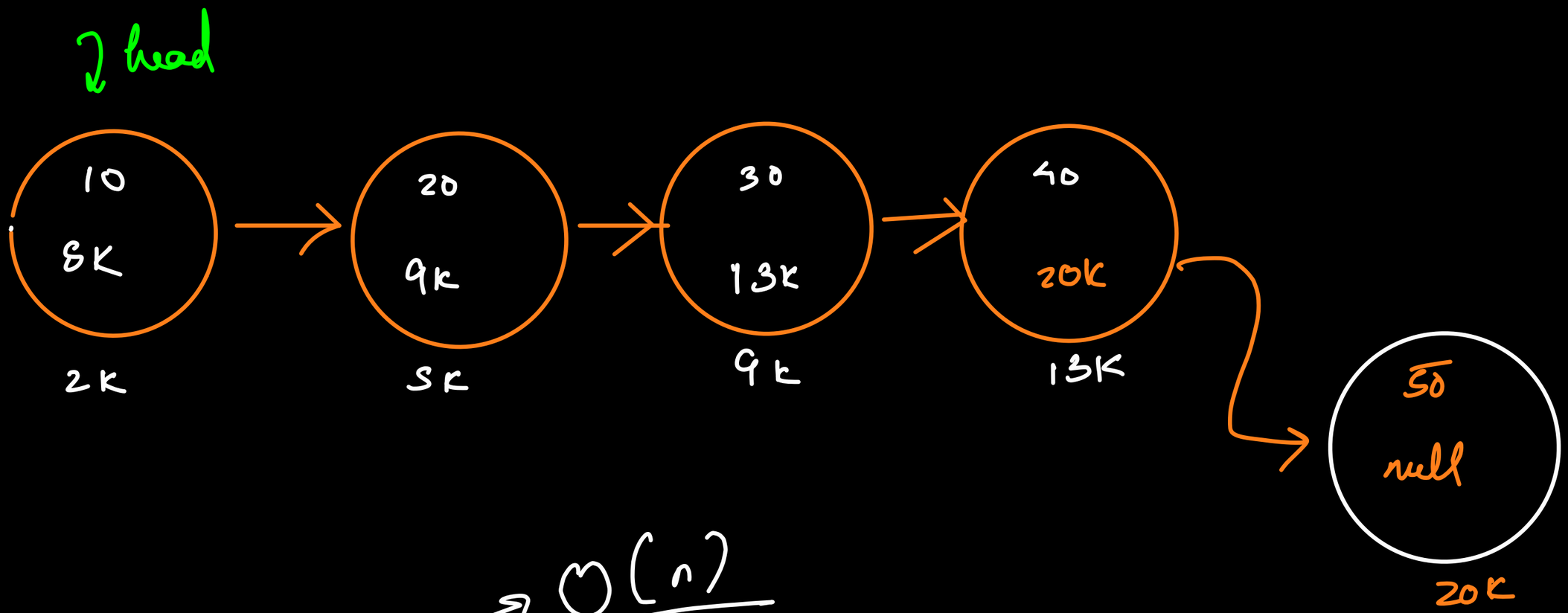
obj.b = 30

{ x : 10
y : 20 }

y

x = 10;
y = 100;
x = y =

```
display (head) {                          → O(n)
    temp = head;
    while (temp != null) {
        print (temp.data);
        temp = temp.next;
    }
}
```



addAtTail ( ) → O(n)

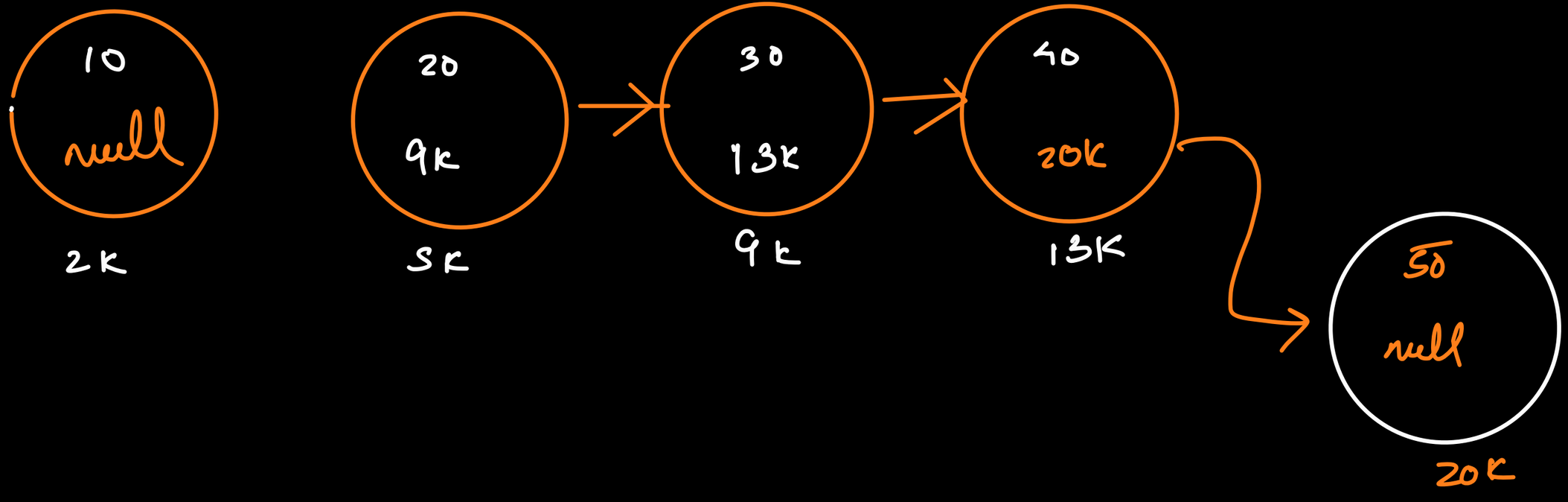
1) Create a new node.
2) attach the new_node after tail

temp.next = new_Node;

1 step

2 head

2 newhead

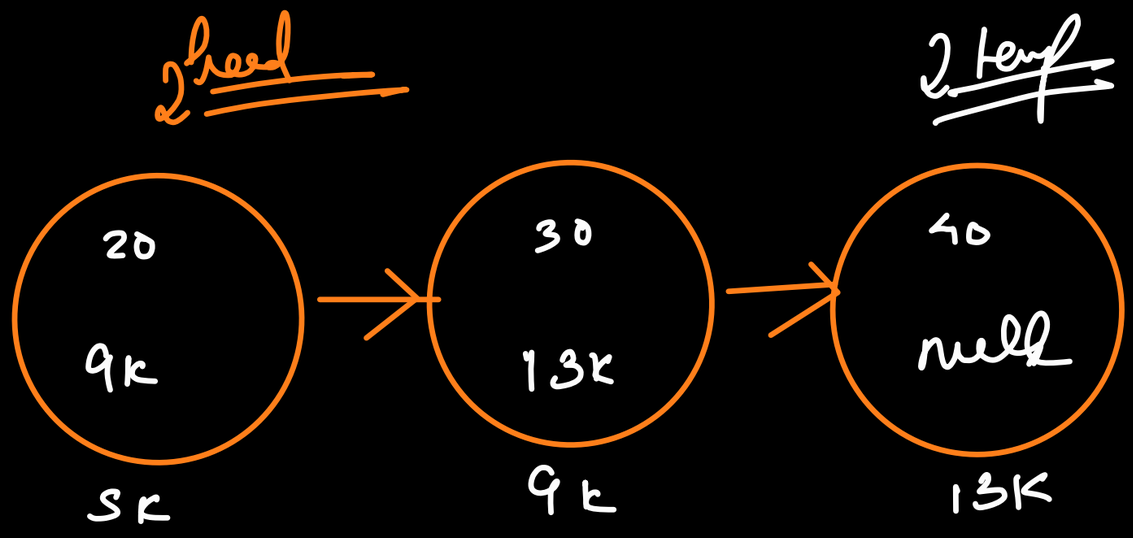


1) Store the new_head
(newHead = head.next)

2) detach the original head from the linked list.
head.next = null

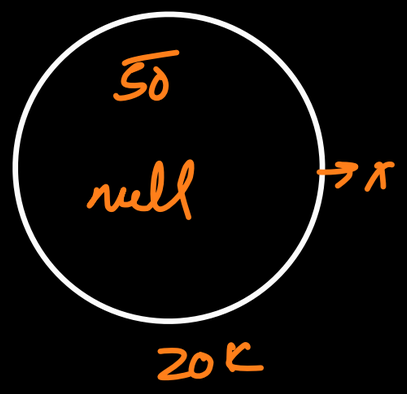
Step

```
remove head (head ) {
    if ( head == null)
        return null;
    newhead = head.next
    head.next = null
    return newhead;
}
```



→ empty ll

→ single node

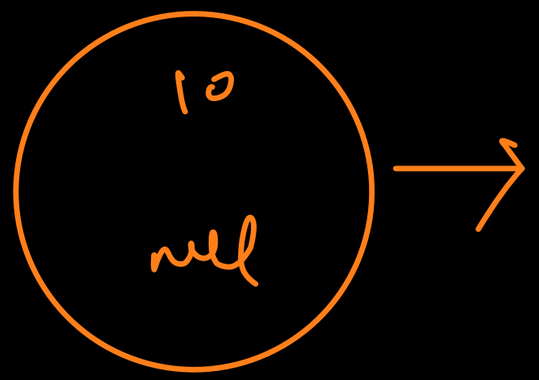


How to identify 2nd last node.

↳ temp.next.next == null

Break the connection of the 2nd last & last node.

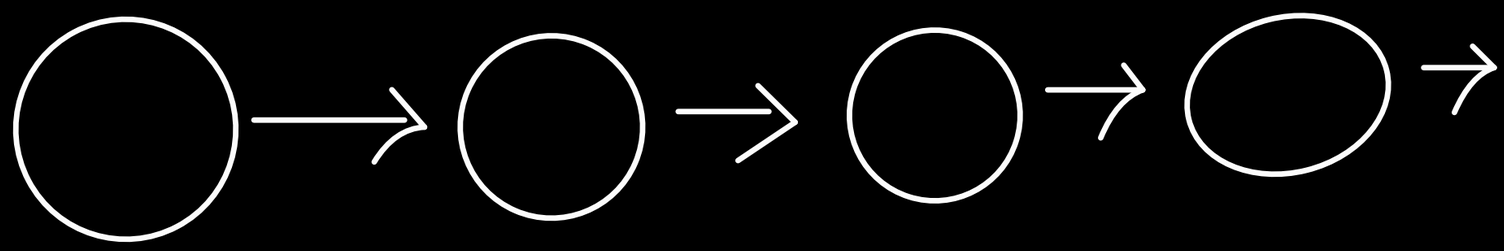
temp.next = null



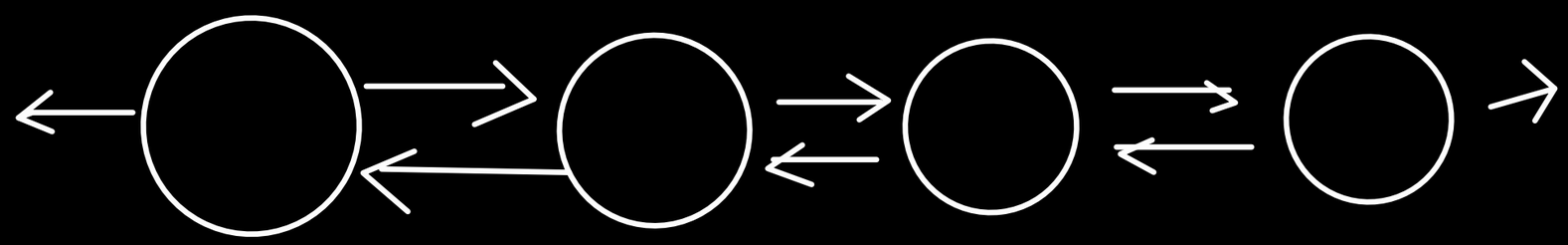

```
if ( head . next == null) {
        return null;
}
```

Types Of Linked Lists

1) Singly Linked List



2) Doubly Linked List



3) Circular Linked List

4) Skip List