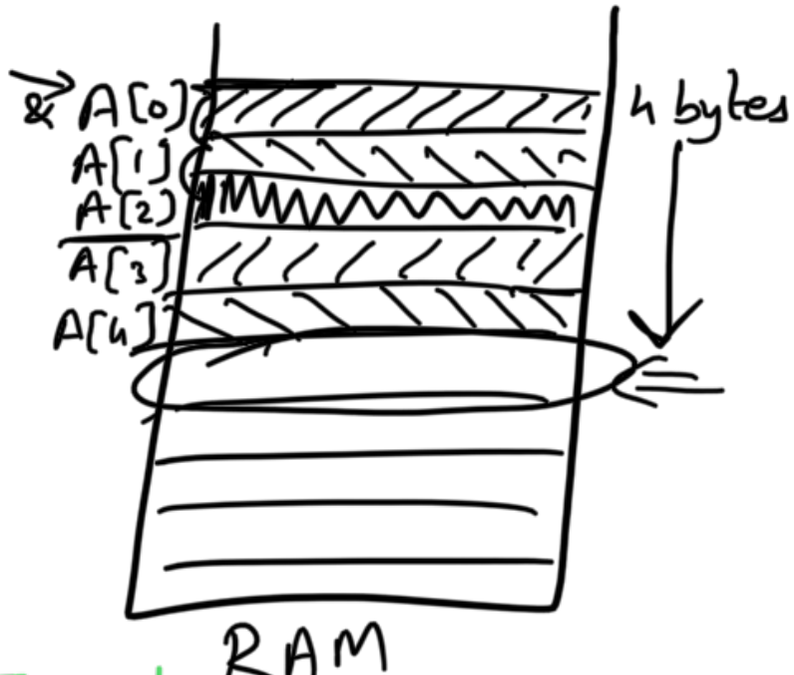
Linked List basics

arrays → O(1) random access

A[i] → i^{th} element

⇓

Contiguous memory locations.



A[2]. int

&A[0] + 4*2

static

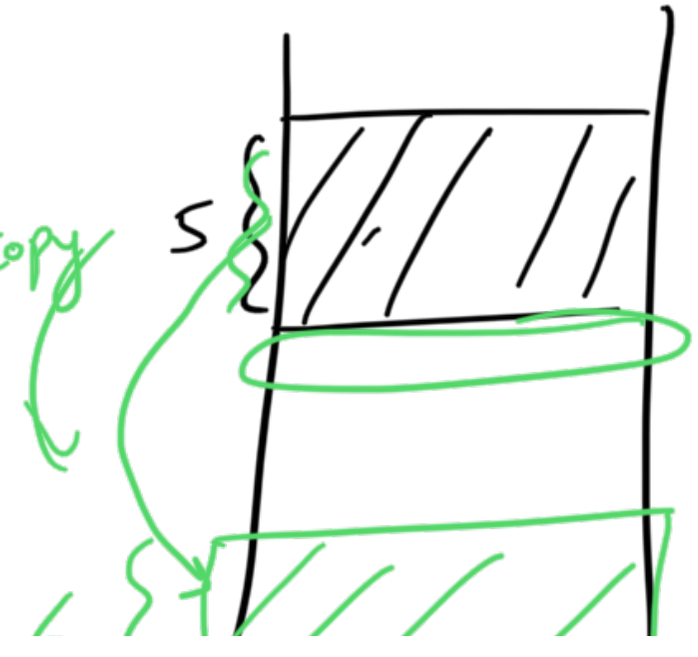
Size → Fixed ⇒ O(N) time ← Insert

Arraylist, vectors → Dynamic Array

A[5] ← add

A[6]

O(N) copy



insertion in Dynamic Array: ⇒ O(1) Amortized

eg.

Rs.80 → 1 week.

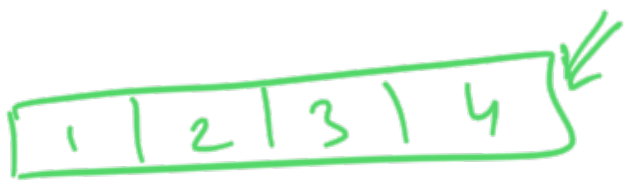
1 year.

10 × 52 * 80 > ...

upfront

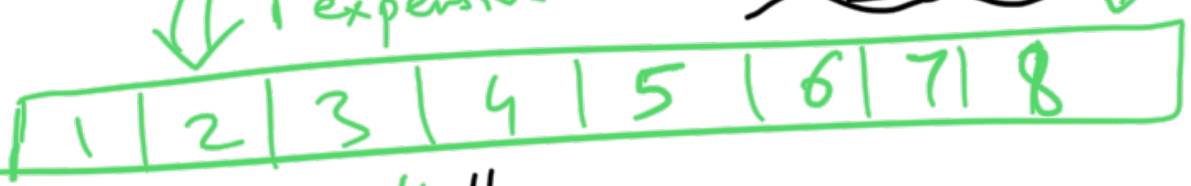
10 yrs → 10K

add



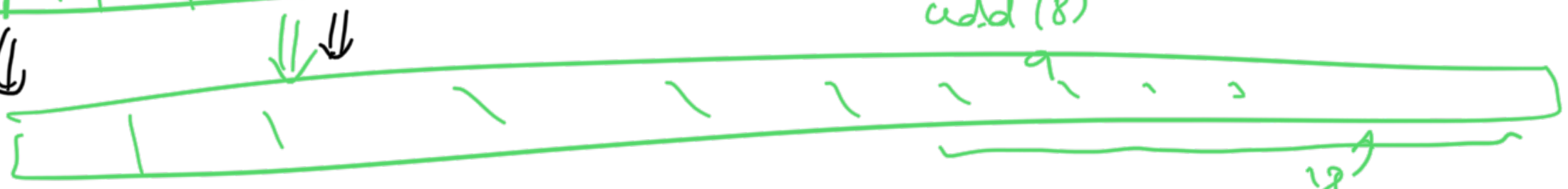
1 expensive

O(1)



add (6)
add (7)
add (8)

O(N).



8

H.W

Insert beginning, end

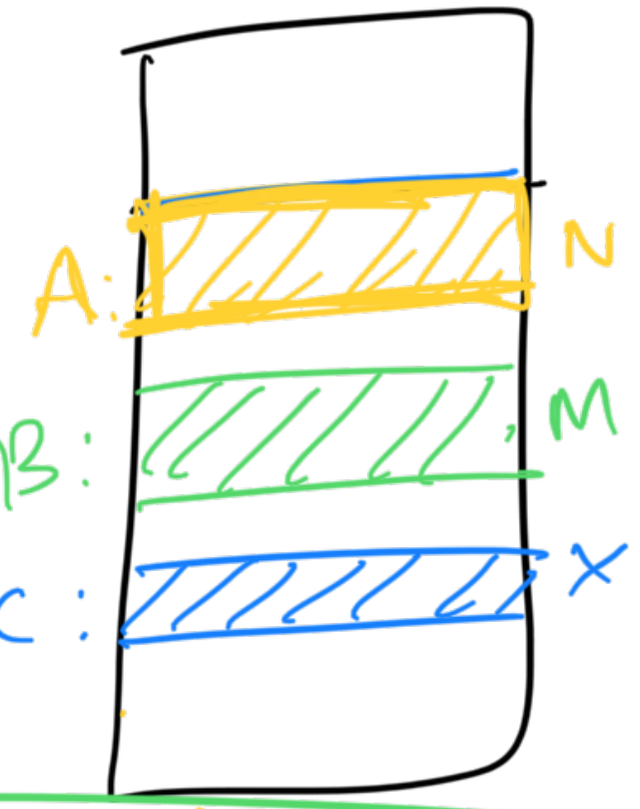
delete

random access.

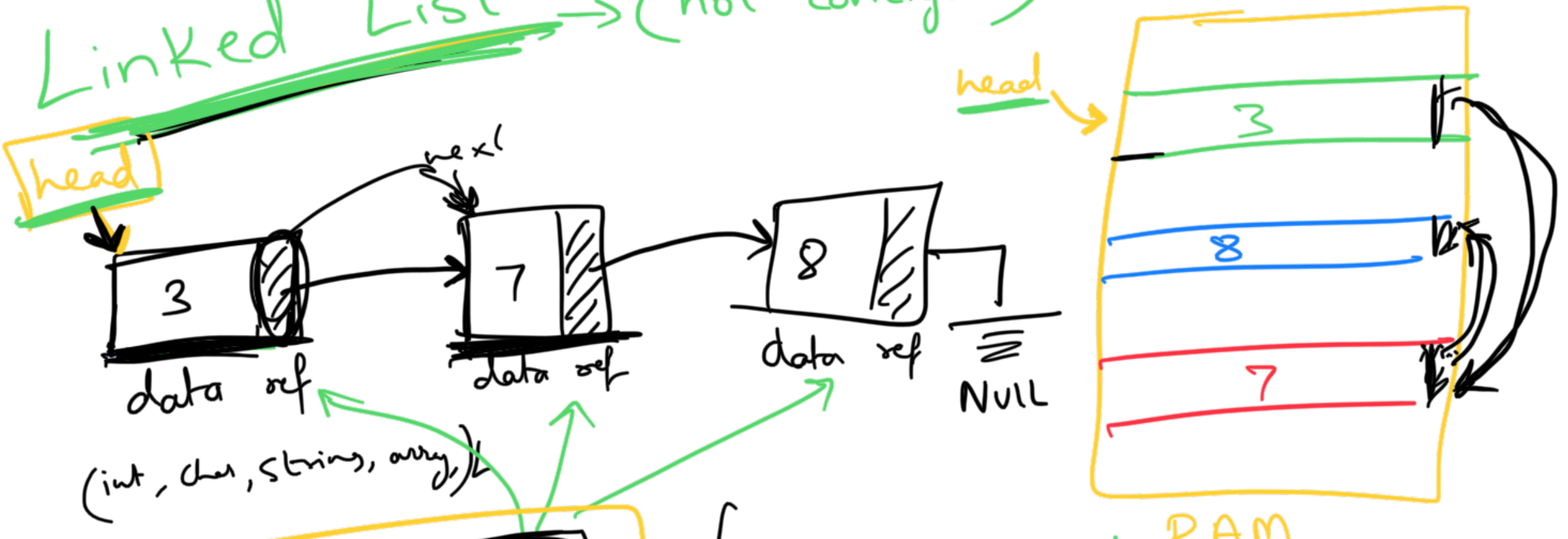
If we don't care about O(1) random

Linked List

add.



Linked List → (not contiguous)



(int, char, String, array, ...)

Node

class INode
{
int data,
Node next
}

Java

class Node
{
int data
Node* next
}

C++

OOP based language

Node mynode = new Node()

construct

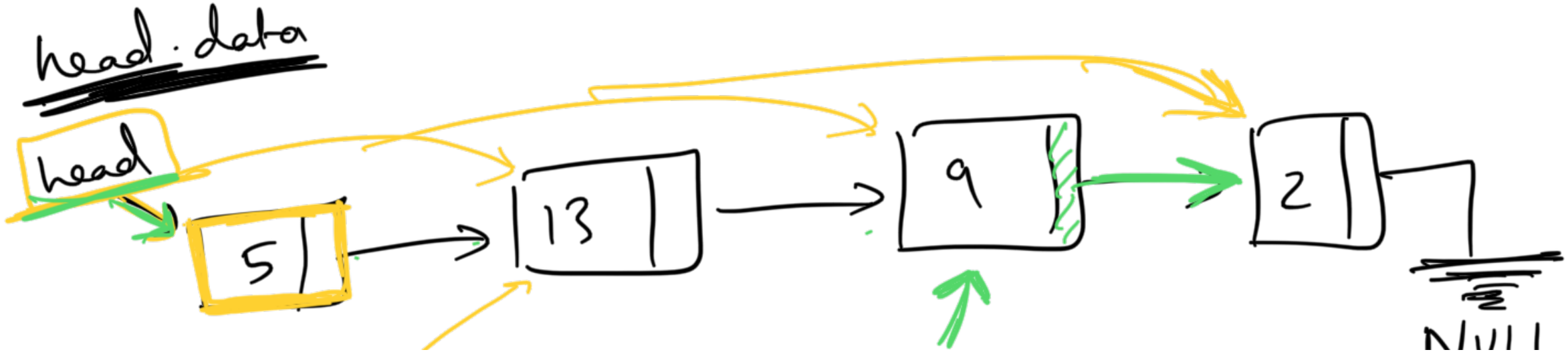
mynode.data = 3

mynode.next =

mydata.data = 3

mynod.next = &

head.data



NULL

head.next.data = 13

Temp

while (T)
{ head = head.next
}

struct node
{

struct node*

};

C++.

head → next

How to create a LL?

Node head = new Node()

head.data = 0/garbage

head.next = null/garbage.

Constructor

Node head = new Node(10);

3

10 → null

3, 5, 9, 4

Node head = new Node(3)

Node temp = new Node(5)

head.next = temp

head → 3 → null, 5 → null

L.L → 5 Nodes → [1, 2, 3, 4, 5]

int [ ]

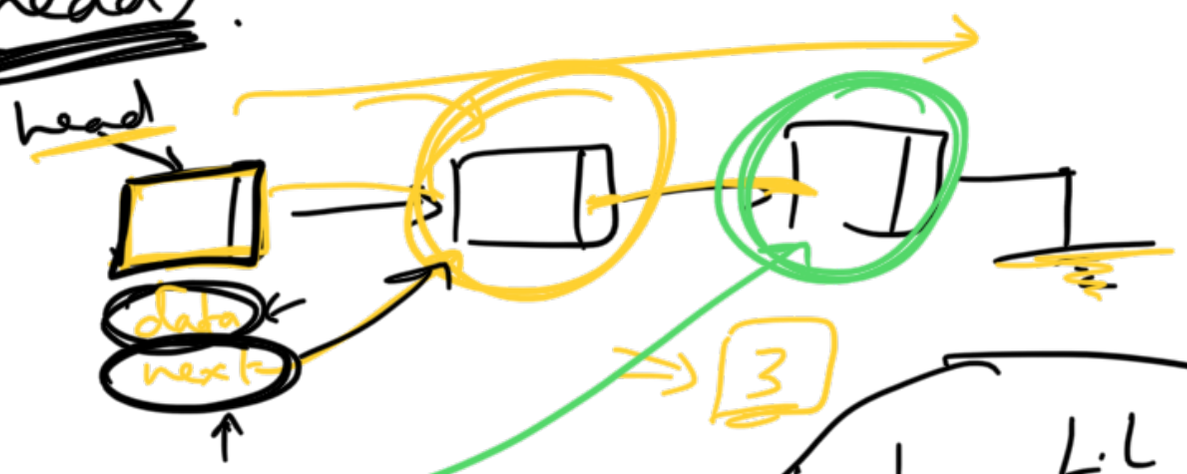
head → 1 → 2 → 3 → 4 → 5 → NULL

O(1)

N node

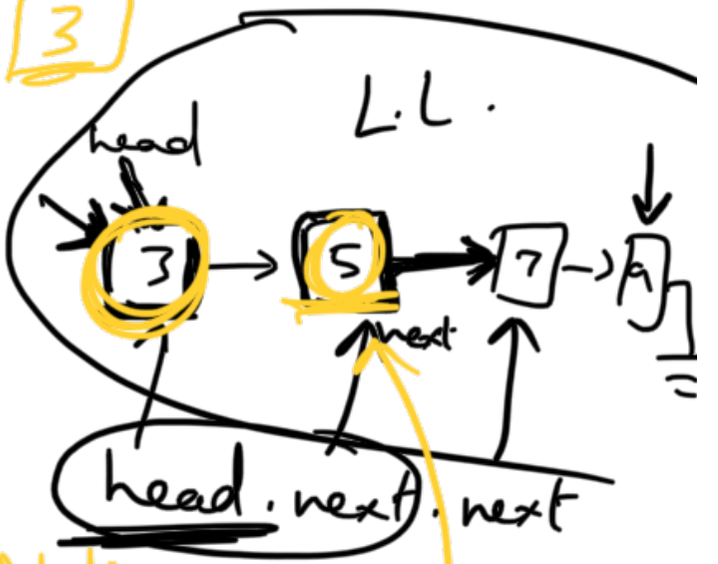
O(N) time

Q.1 Given a Linked List (head). Find the length of LL

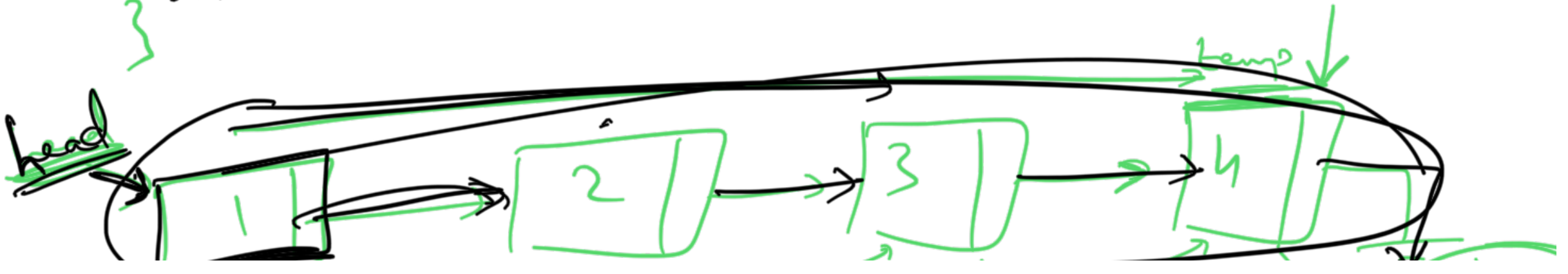


head.next

head.next.next




```
int getlength(Node head)
{   int length = 0;
While (head != NULL)
{   length++;
    head = head.next;
}
return length.
}
```

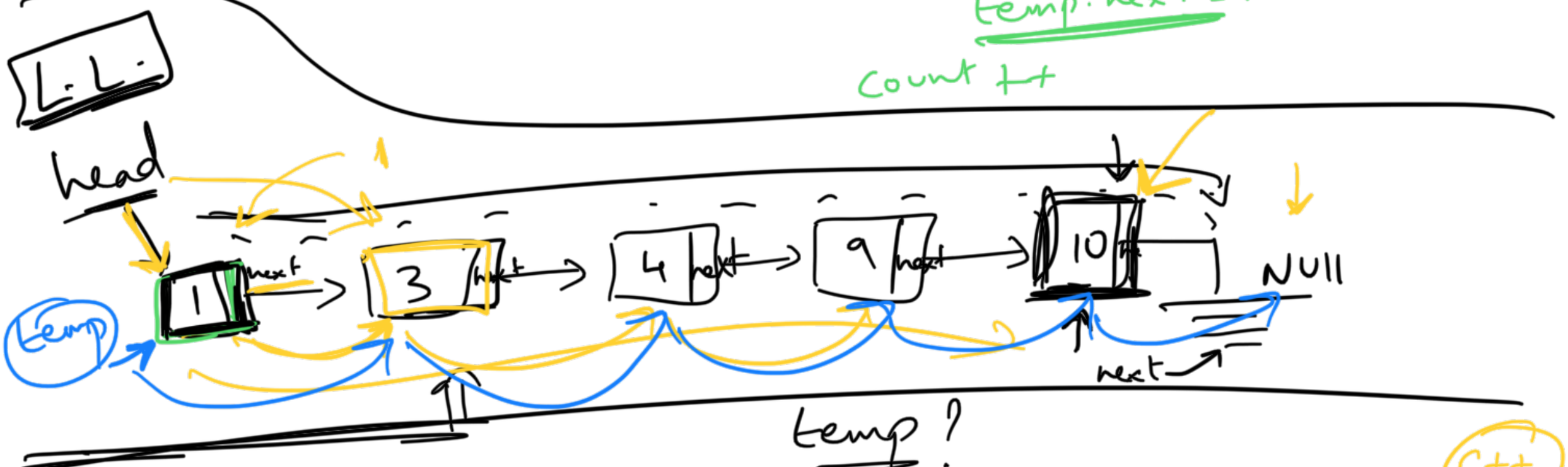


NULL

head.next.next.... = NULL

temp.next = NULL

count ++



temp ?

C++

```
int getlen(Node head)
{
    int len = 0
    while (head != NULL)
    {
        head = head.next
        len++;
    }
}
```

*head.data

*head.next

return len;

head

Bug

Class Node

char int data

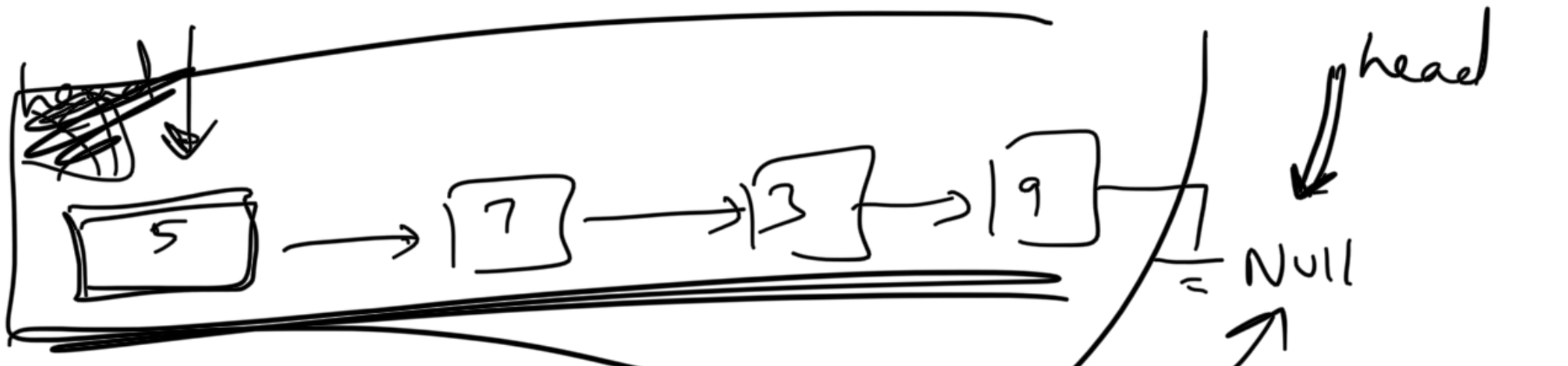
Node next

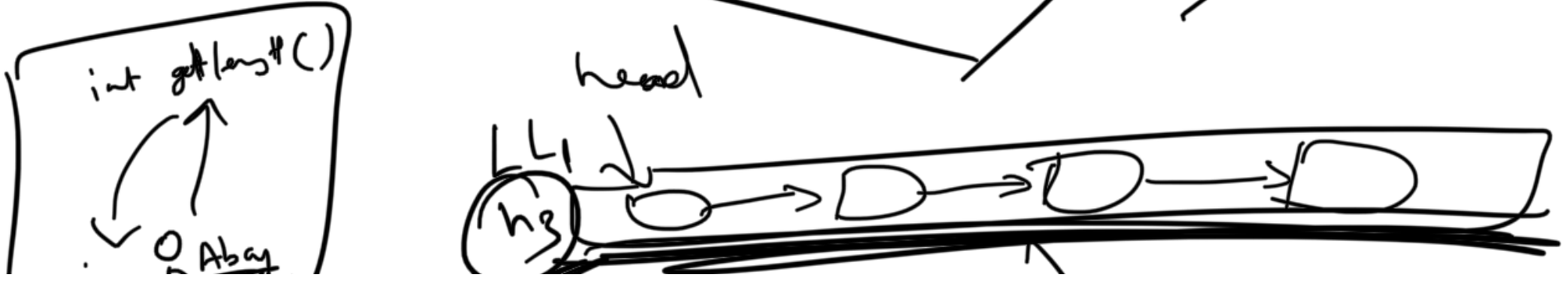
object

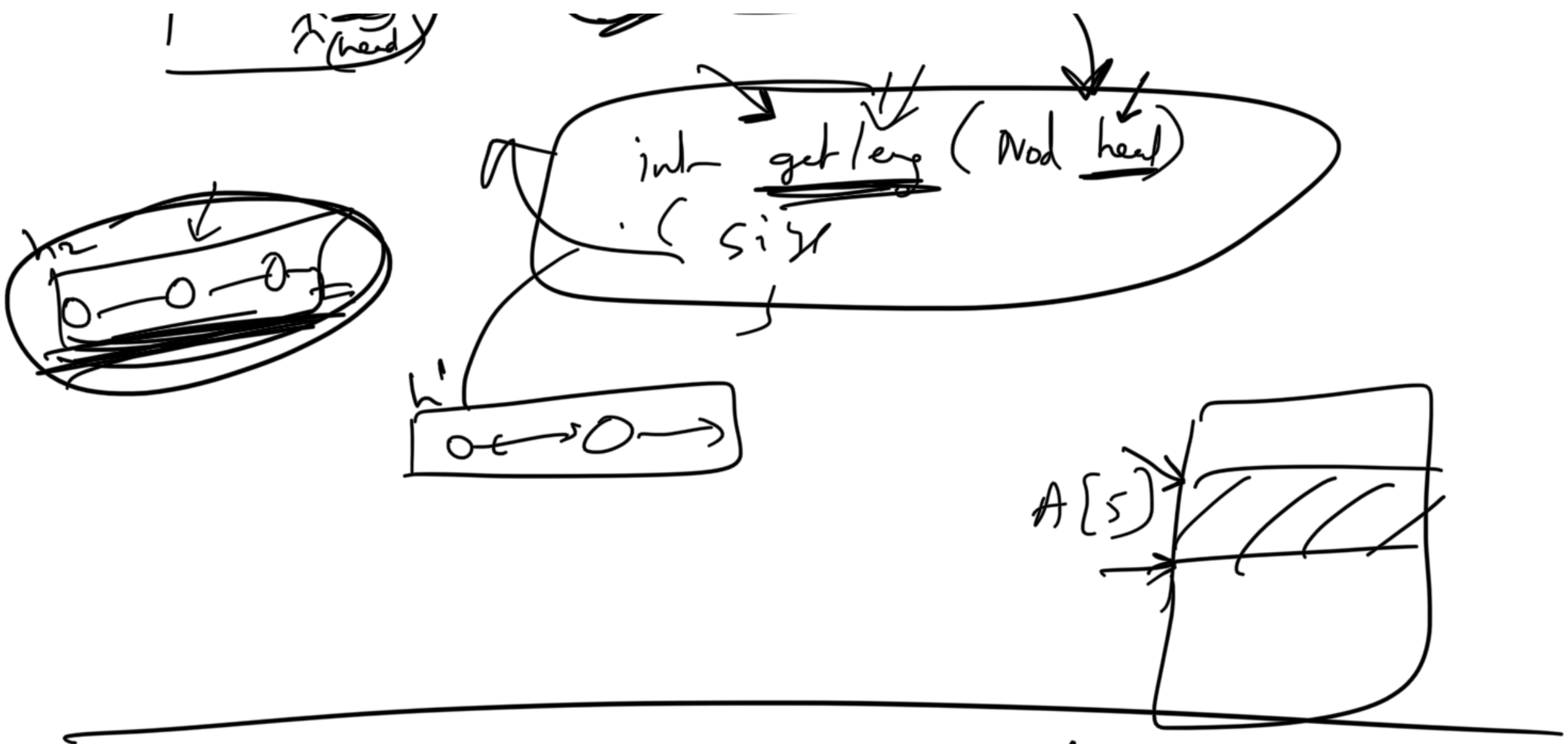
Node h = new Node()

h.data = 5

h.next = null







We never modify head, unless asked

C++

Node *temp = head

```
int getLength (Node head)
{
    int len = 0
    Node temp = head
    while (temp != NULL)
    {
        temp = temp.next
```

len++

}

return len;

temp

head

len

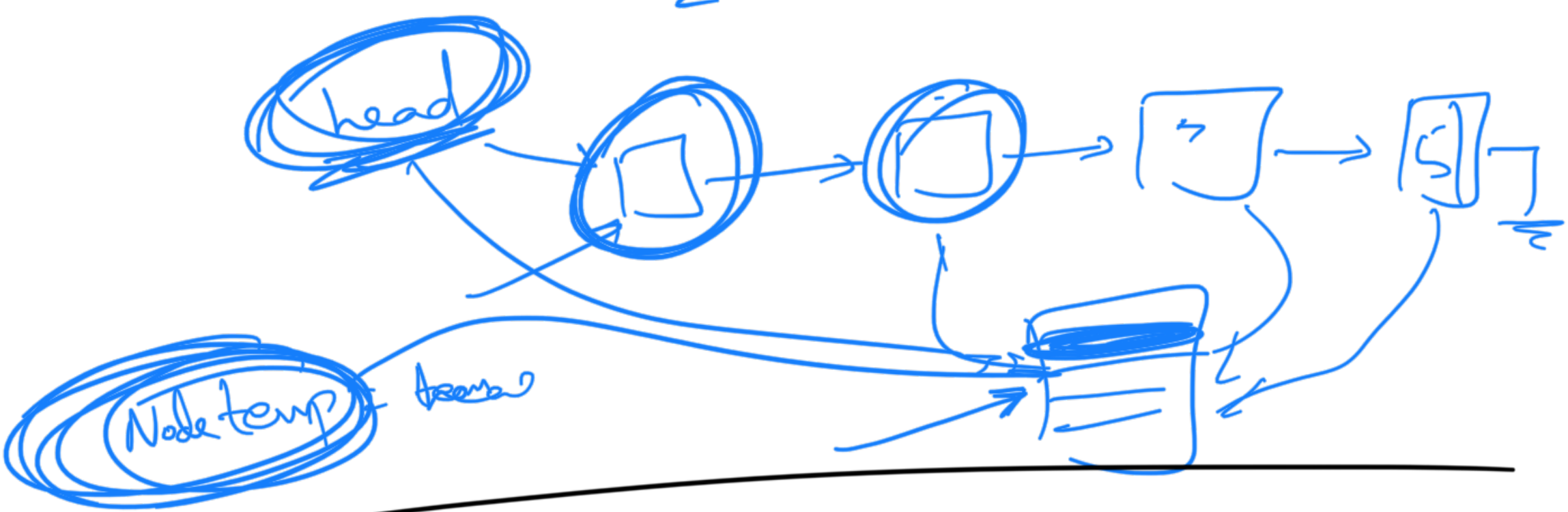
LL

head

getLen(head)

head => NULL

2 variable



~~Arraylist~~

Elastic Search / Solar / Lucene

Inverted index (skip list)

Key → doc1, doc2, doc3 ....

doc1, doc

Key?

iphone	doc1, doc2, doc3 ....
black	doc3, doc9

Hashmap

Key

Collision

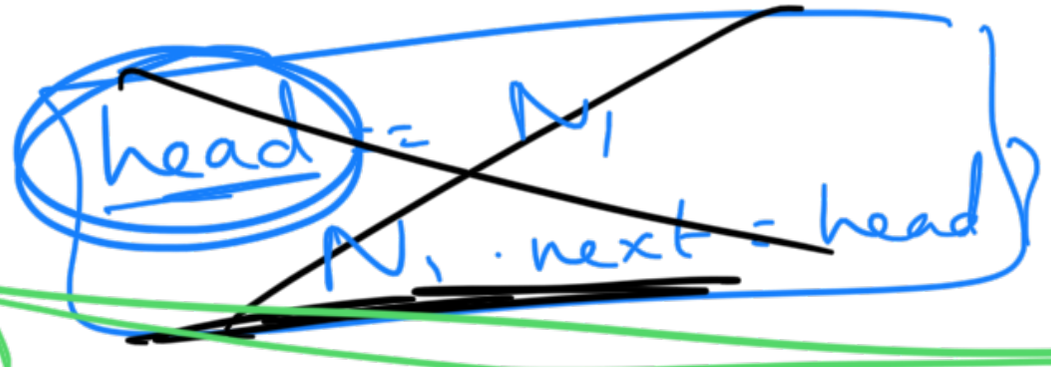
O(1)?

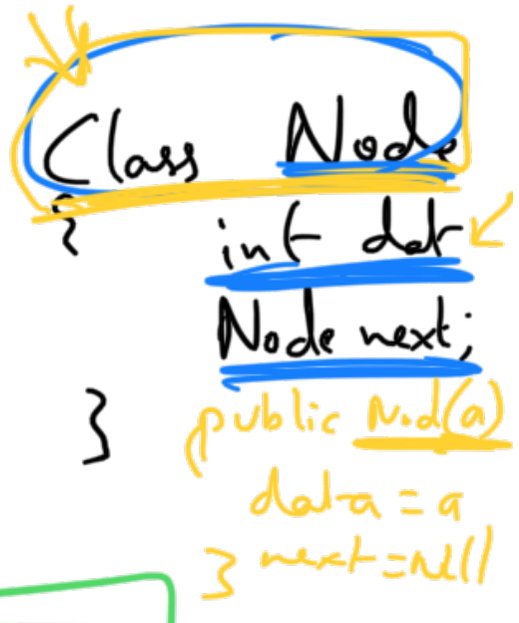
insert-

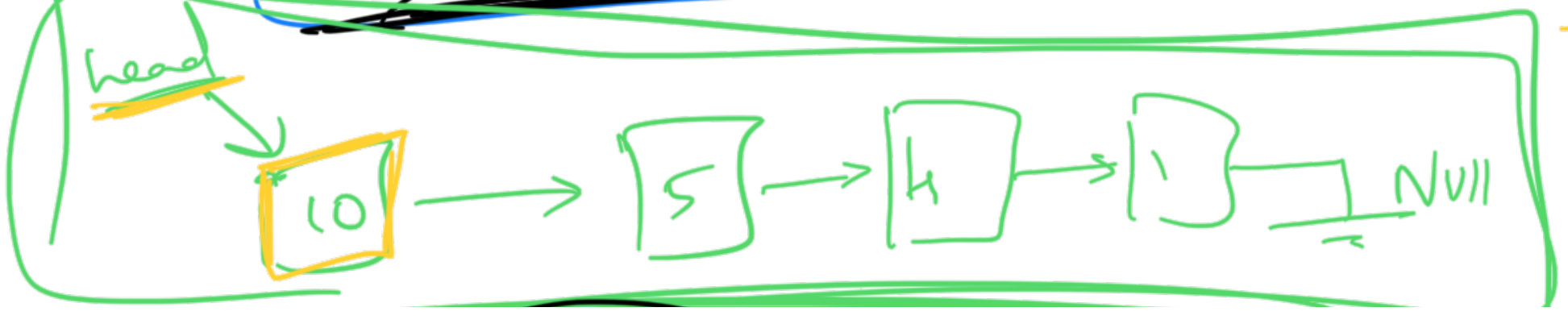
head

Given a LL, add some data at start

10





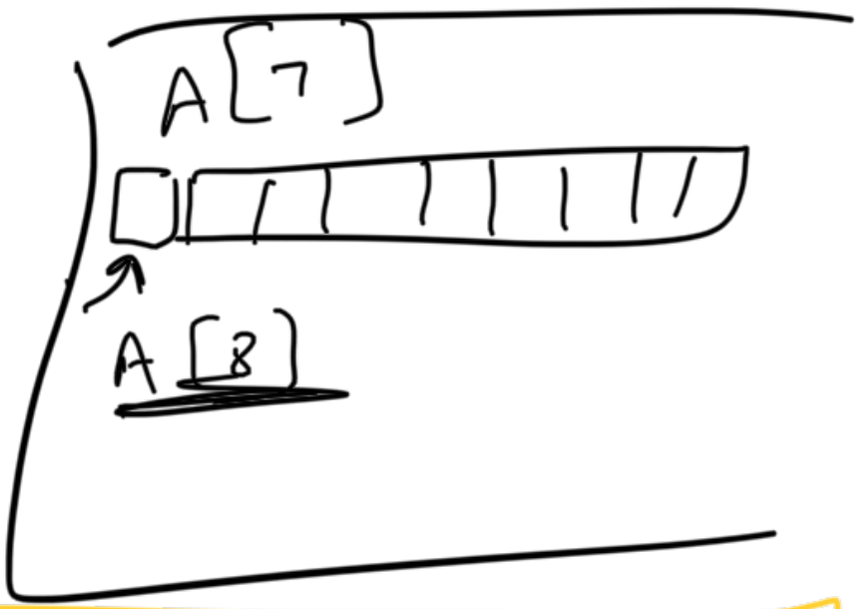


N_1 . next = head

head = N_1

T.C ⇒ O(1)

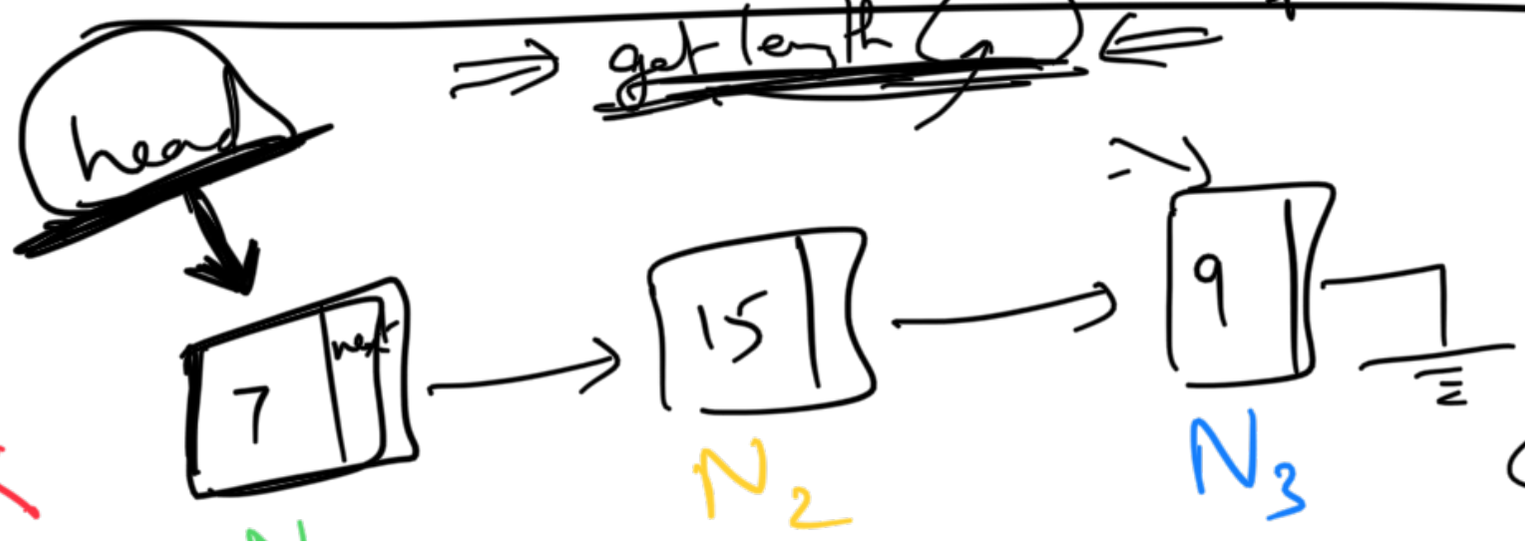
V.S. O(N) for array



At any point if we lose head → LL is lost

Break till 10:50

get length



N_1

Node X = new Node(5)

C++

Node *X = new

&

head →

Address		
1089	5	next
2012	7	
2019	9	next

N_3

RAM

Class P

Class Node (a) (a address)

{

data

next

};

P *

N_2

N_1

2012

off

1021

1 if 20

hexadecimal

3

(2) At end:

head

tail

Java

head → reference to the Node

temp

head → pointer to the node

Node N_1 = new Node()

temp = head

while (temp.next != null)

{

temp = temp.next

}

N_1

null

tail

T.C ⇒ O(N)

tail → temp

temp.next = N_1

N_1.next = NULL

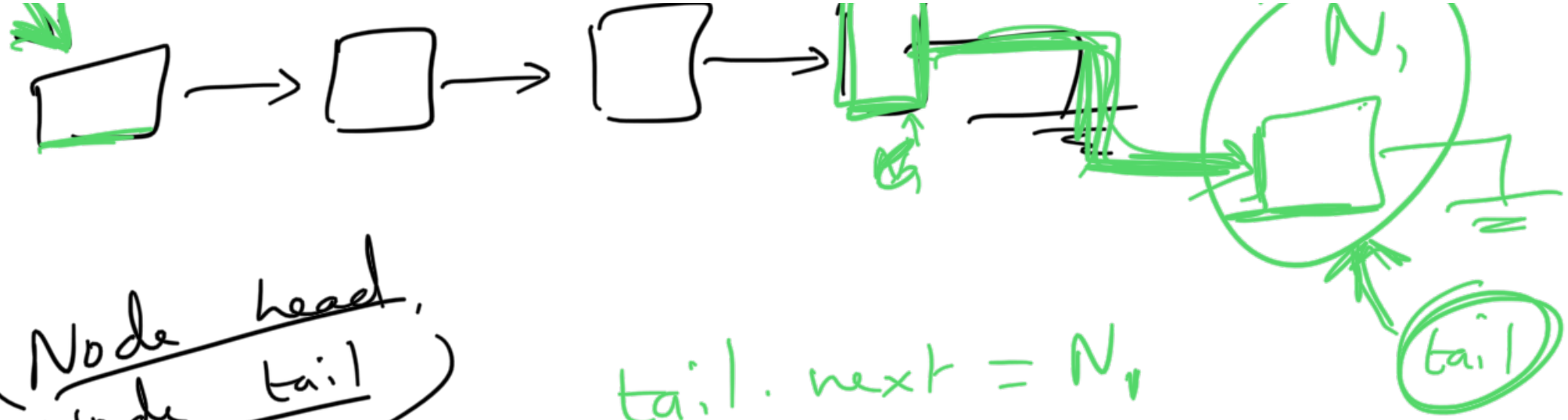
If tail is stored → insert at end bcoms O(1)

tail.next = N_1

tail = N_1

head

tail



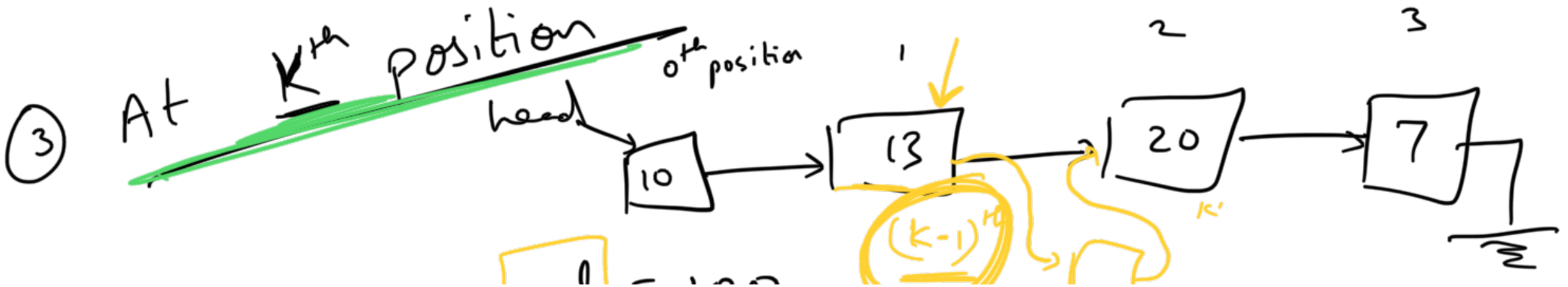
(Node head,
Node tail)

tail.next = N1

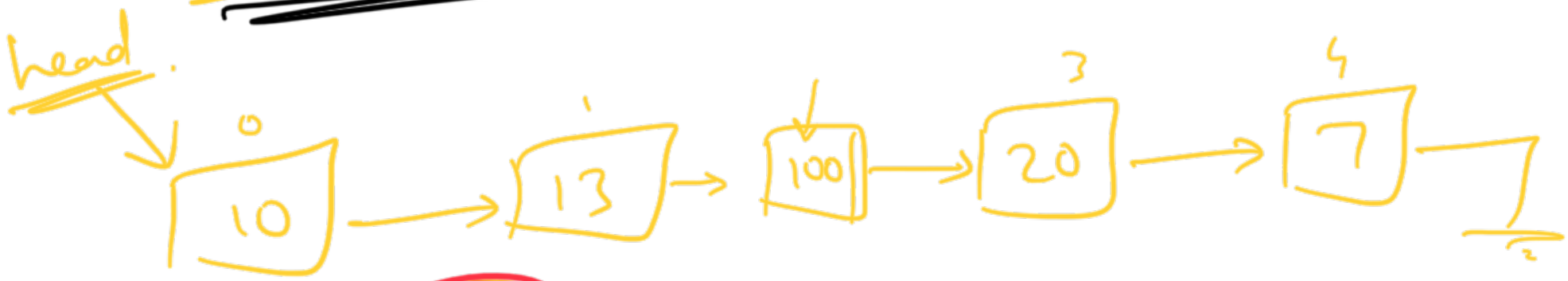
tail = N1

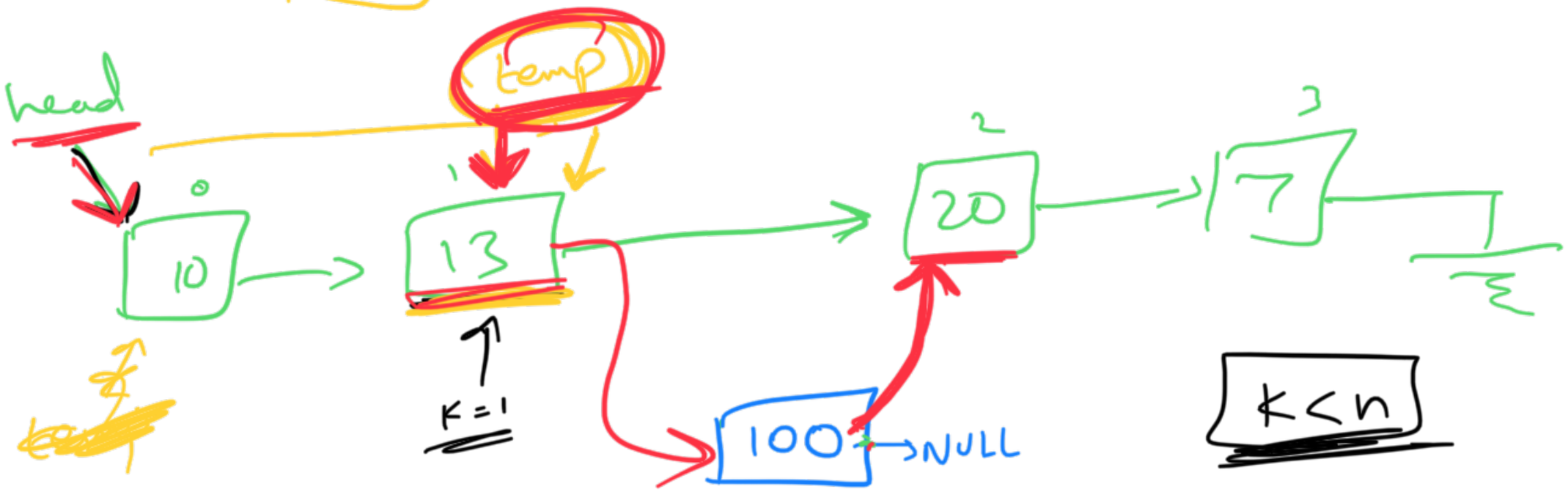
tail.next = N1

tail = N1



k = 2, val = 100 .





k<n

c = 0

temp = head

```
While ( c < k-1 )
{
    temp = temp.next
    c++
}
```

k = 2 .

0 < 2 !

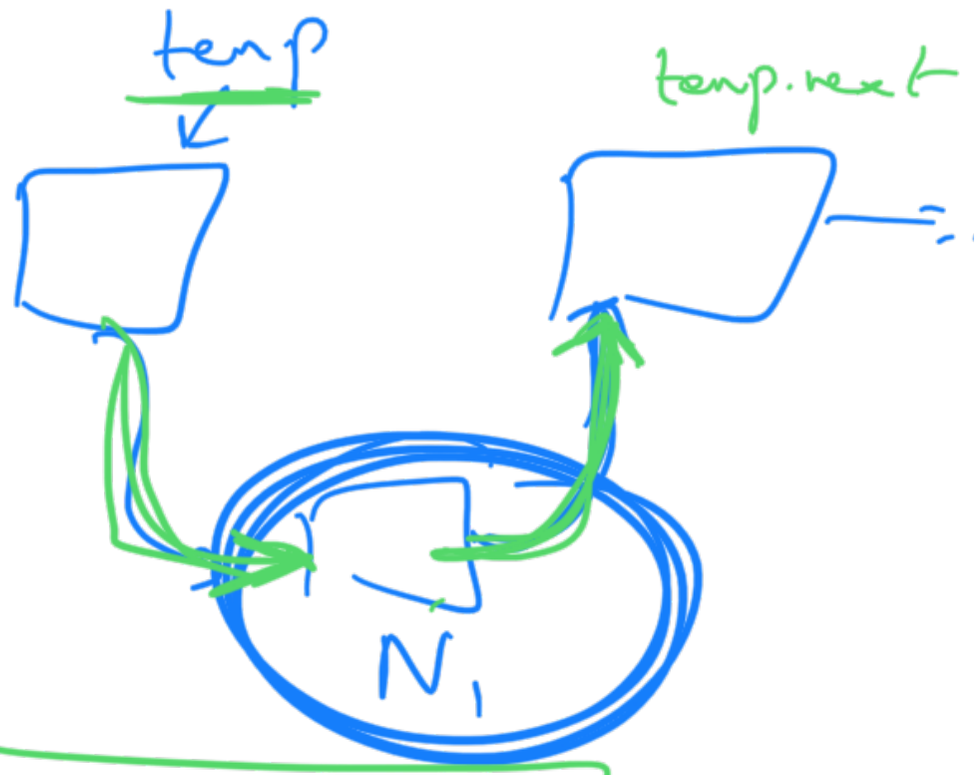
c = 1

Node (100)

Node N_1 = new

N_1.next = temp.next

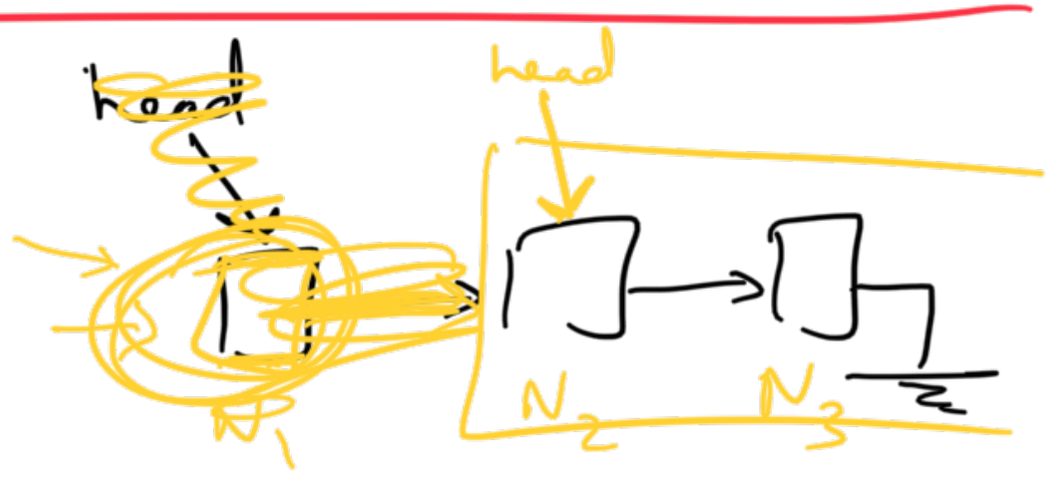
temp.next = N_1



T.C => O(N)

V.S. for array T.C => O(N)

Delete ① From front.

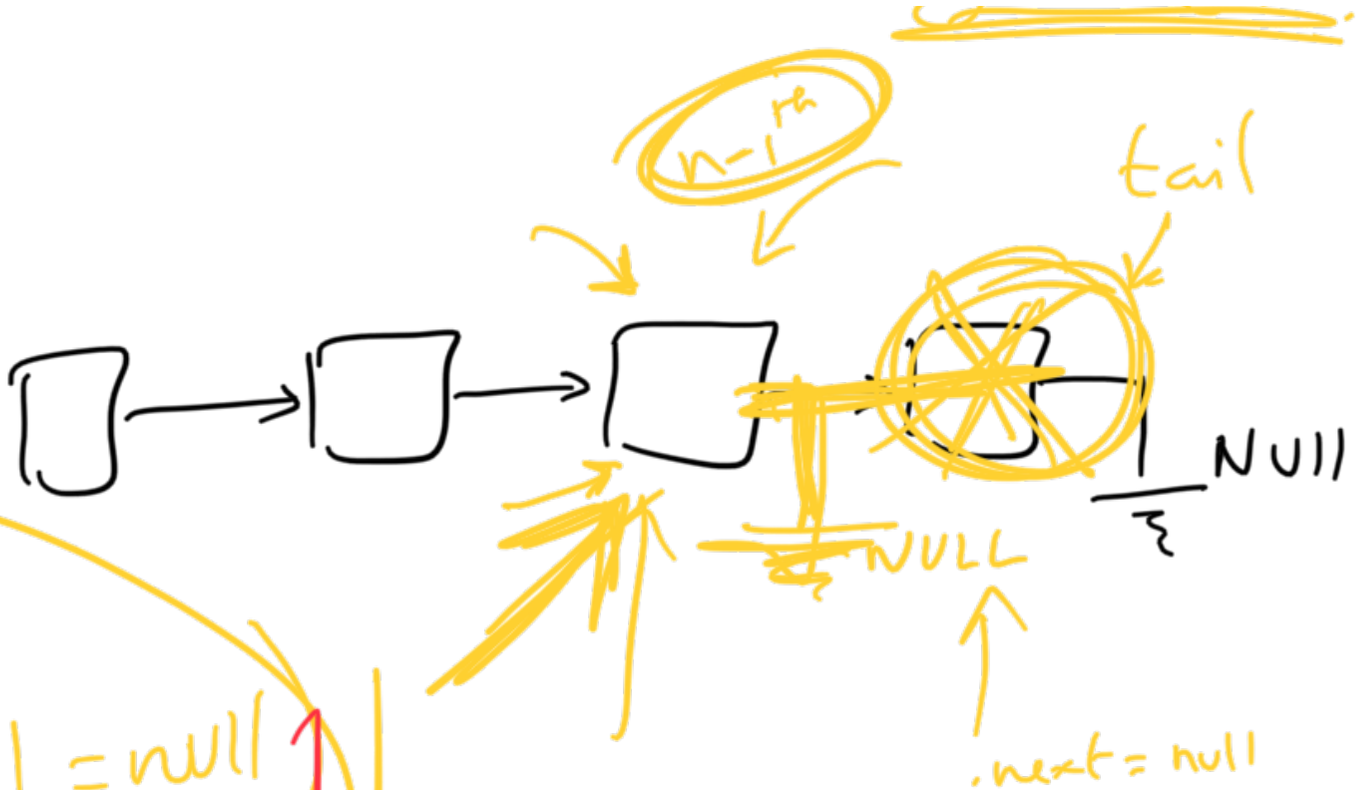


head = head.next

T.C => O(1)

V.S. array O(N)

② From back.



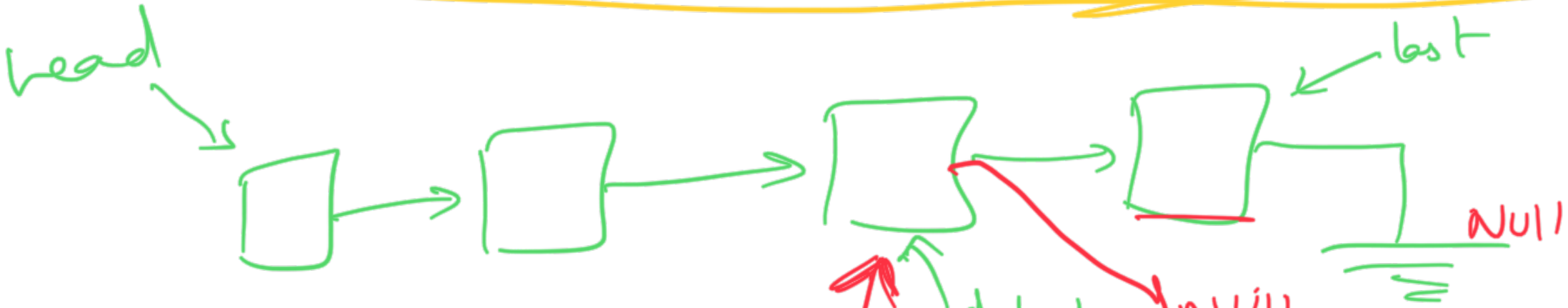

```
temp = head
While (temp.next.next != null
{
    temp = temp.next
}
temp.next = null
tail = temp
```

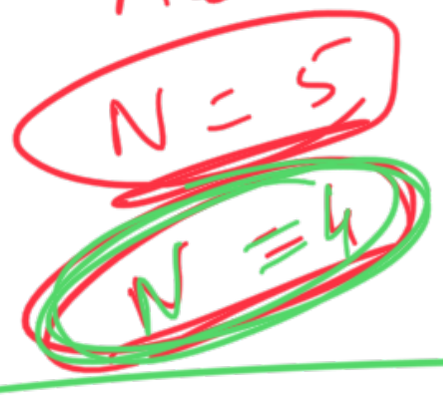
T.C ⇒ O(N).

V.S. array

O(1).

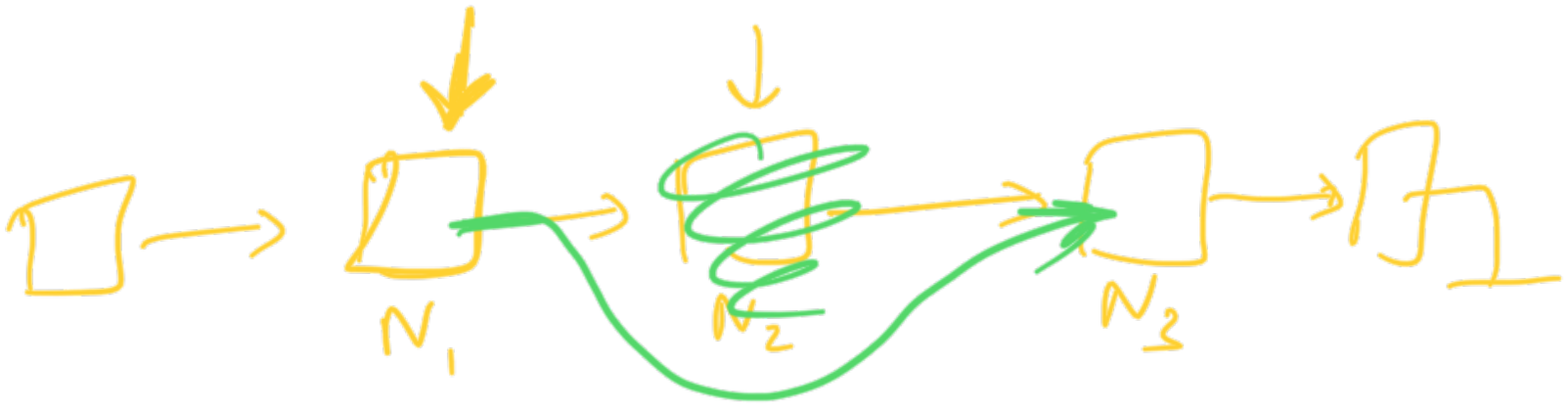
array → (n-1).





A for (i = 0 → N)

③ H.W How to delete from K^{th} pos



$N_1.next = N_1.next.next$

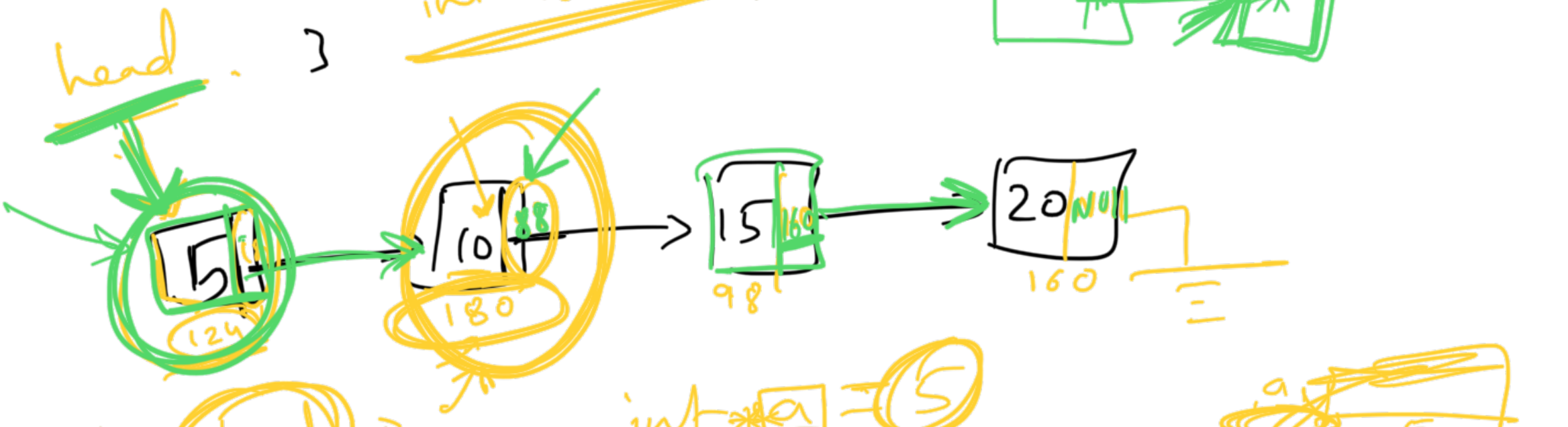
End

Doubts

Class
{
Node
int data;
Node next
180
int address = 180
}

Class

next → object of type Node



Node head →

int *a = 5

Node X = head.data
head.next

Java
inbuilt

X.data

X.next → in

.next

Link

Class Node

{

String data

Node next;

Node prev;

Node random

}

Node

size

public int

head

size

tail

add()

delet

size

s

Sing

Node * head = address

*head

Node head = new obj