
DSA
TC/SC
Recursion
Sorting
Bubble
selection
insertion
search
linear
binary
Backtracking
List
Array

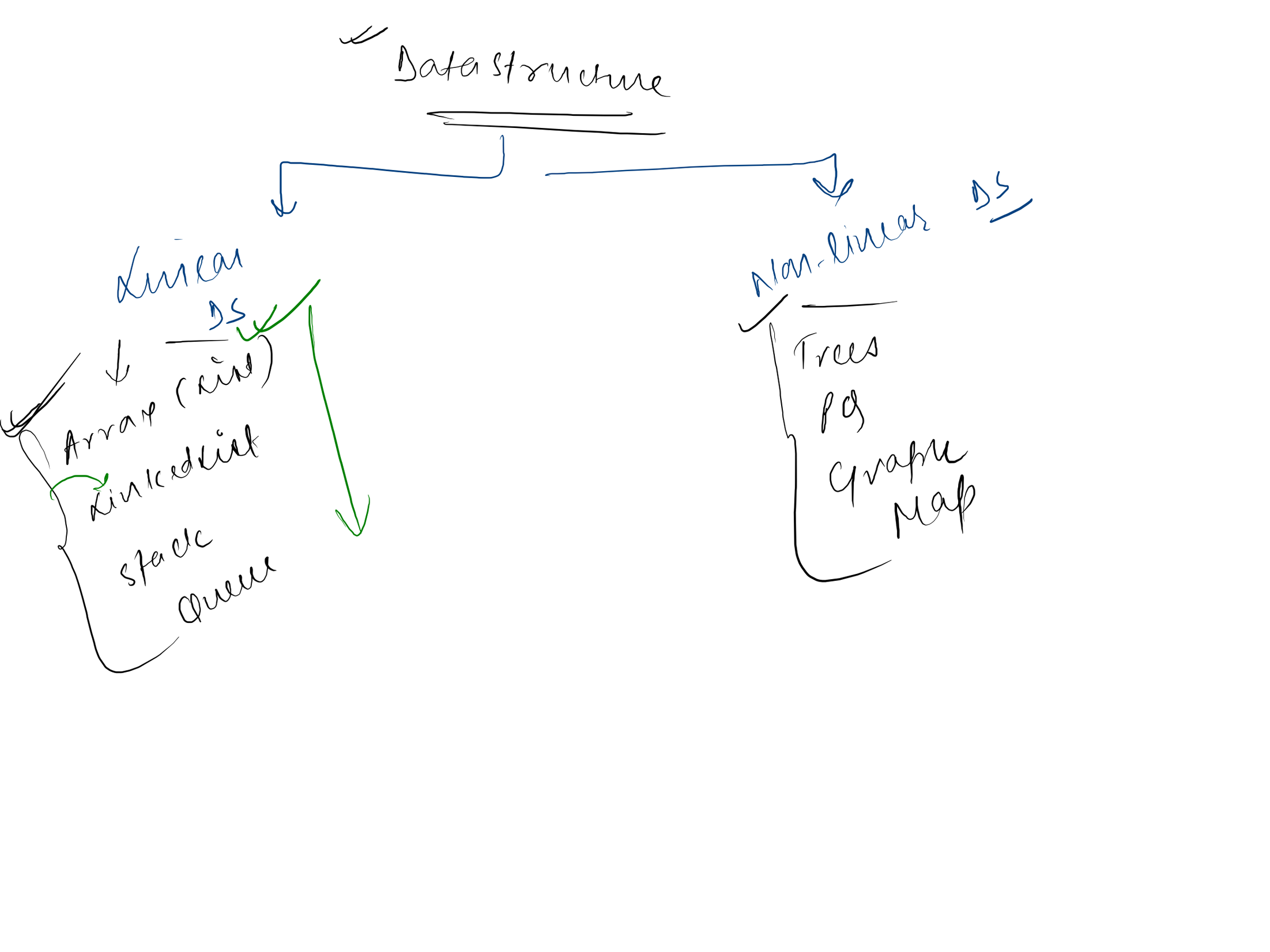
Language

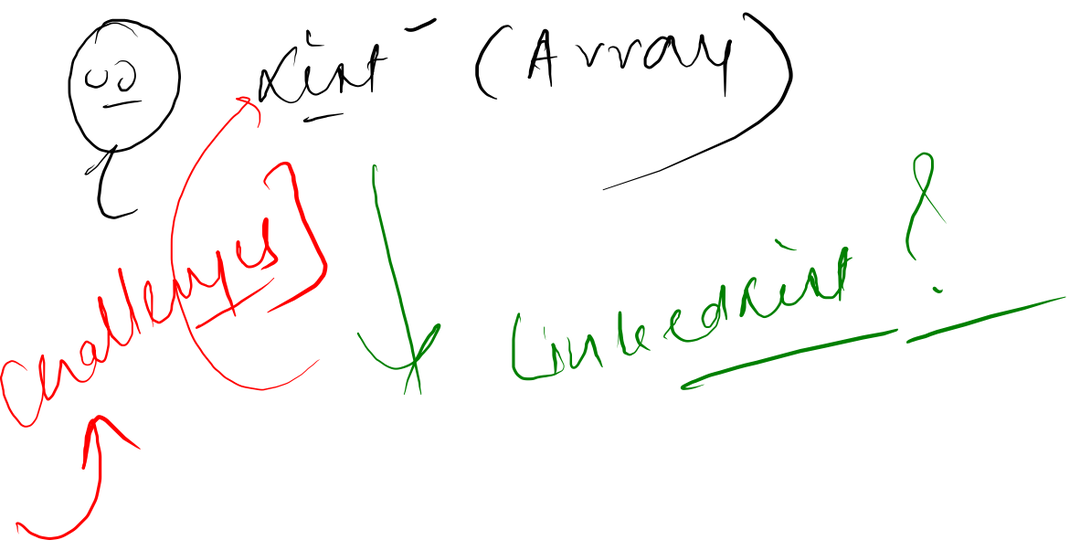
Java → [3, 1, 2, 9, 4]

Python → dict → [3, 1, 2, 9, 4]

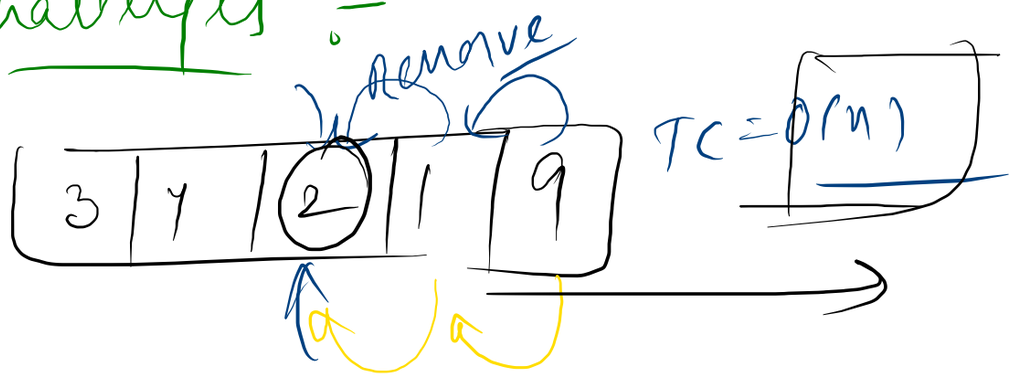
LinkedList }

[List
↑ Array]


Data structure
Linear DS
Array (1D)
LinkedList
Stack
Queue
Non-linear DS
Trees
BST
Graph
Map


int (Array)
challenges
Linkedin !

Arrays → challenges :-

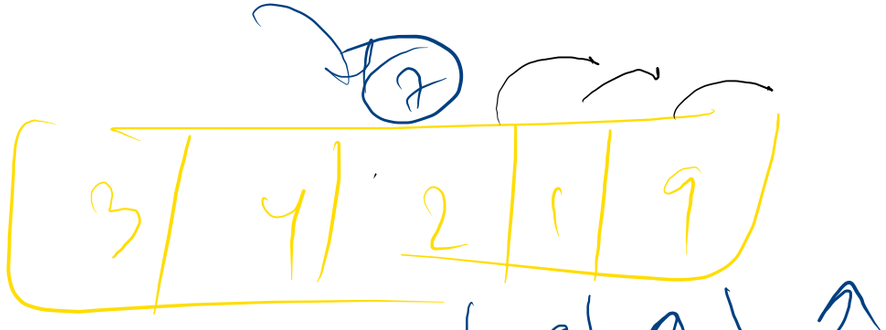


DSA

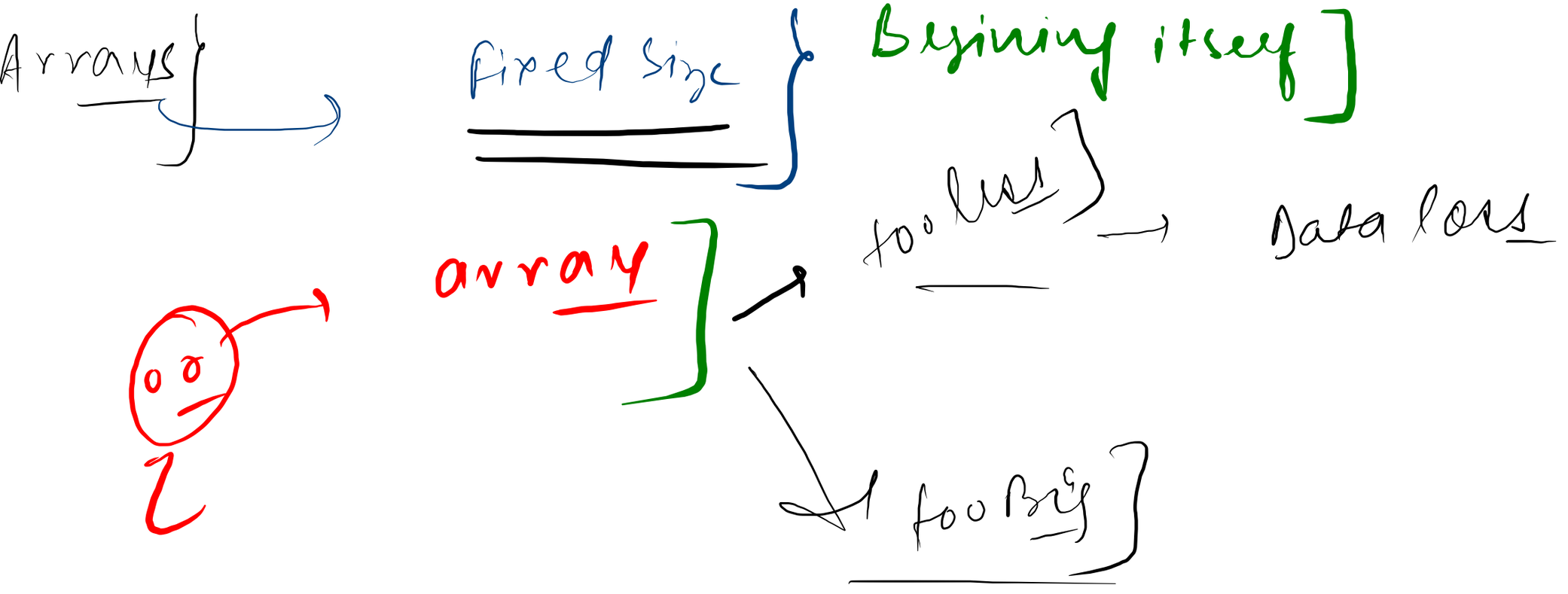
↑ aptitude & PL

Removal / Addition

Expensive Operation



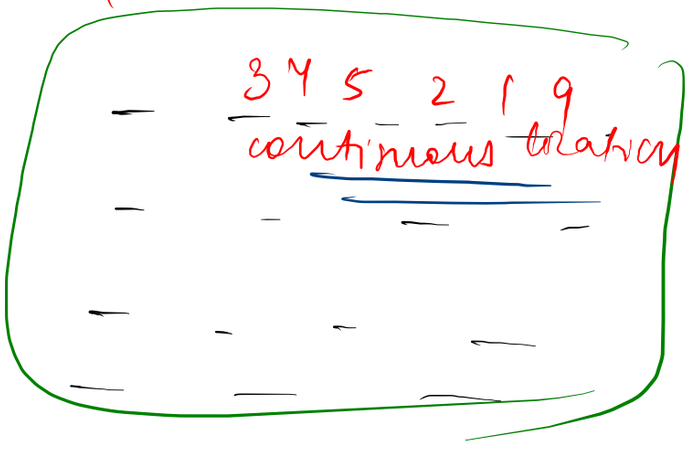
[3, 4, 2, 1, 9]

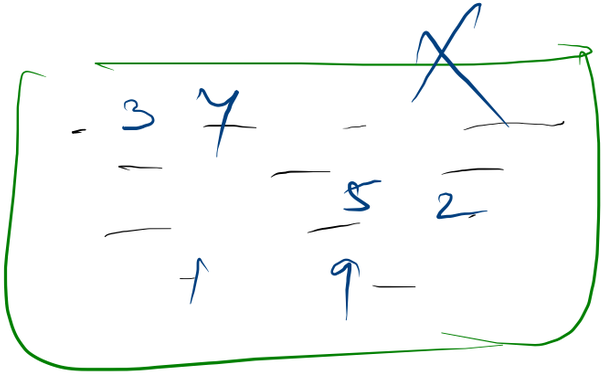

Arrays
fixed size
Beginning itself
array
foo list
Data lors
foo Big

Array

All elements are stored in continuous location in memory

[3, 4, 5, 2, 1, 9]





O(1)

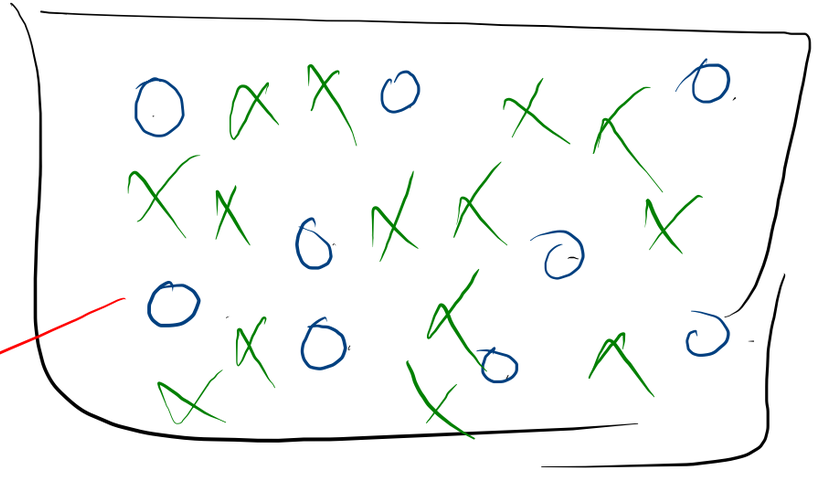
Retrieve elements

Industry

[3, 2, 4, 1, 5]

004

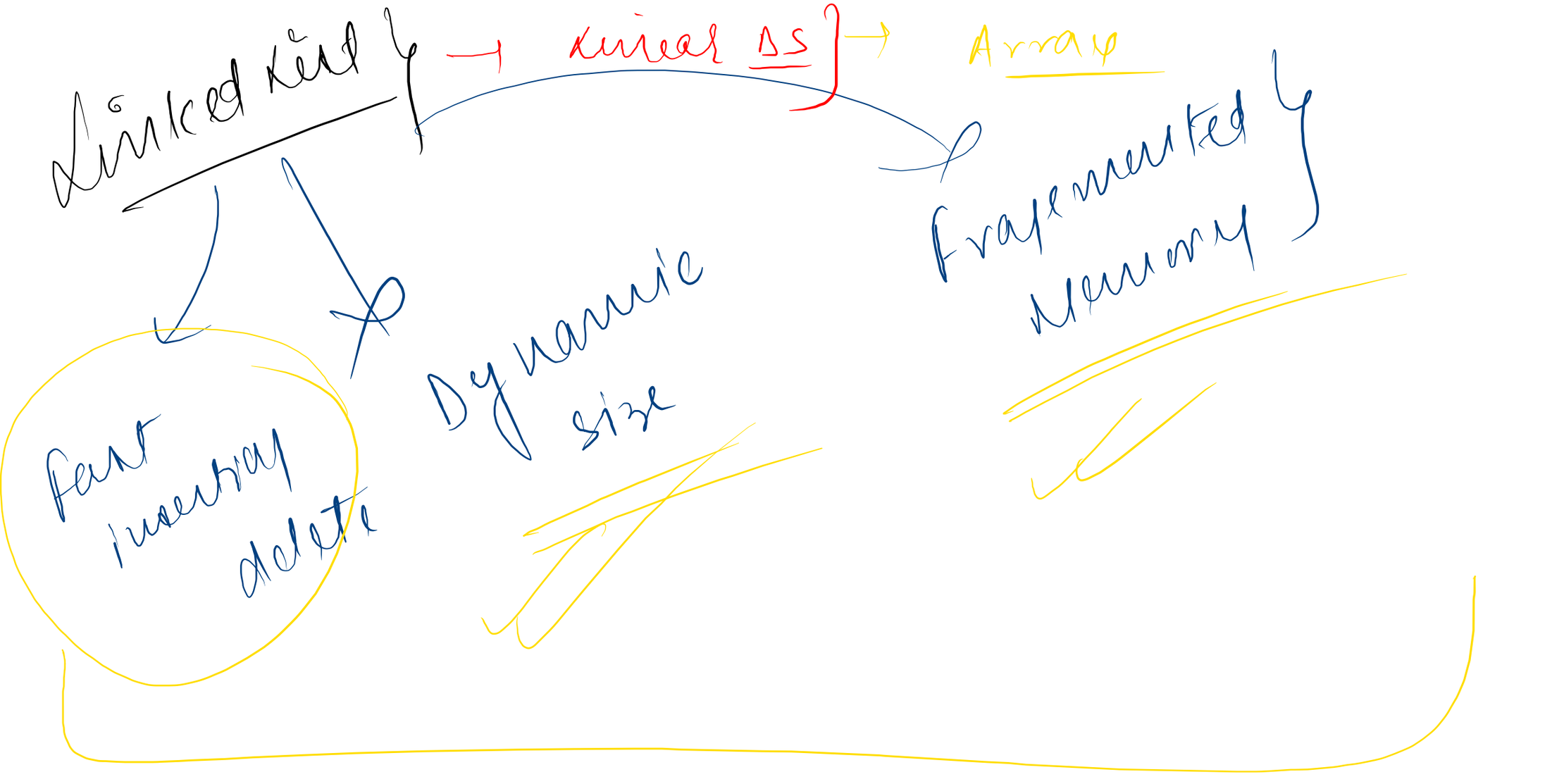
q - vacant



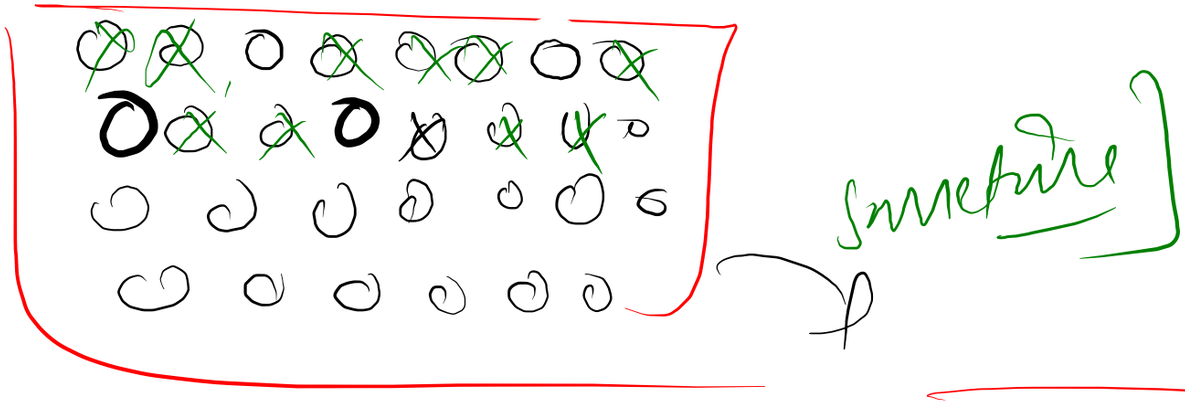
Fragmented Memory

0 → free space

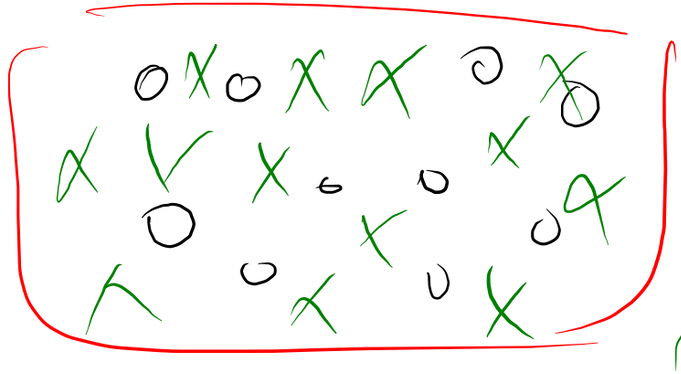
x → occupied


Linked List
Linear DS
Array
Fragmented Memory
Dynamic size
Fast insertion deletion

Fragmented Memory



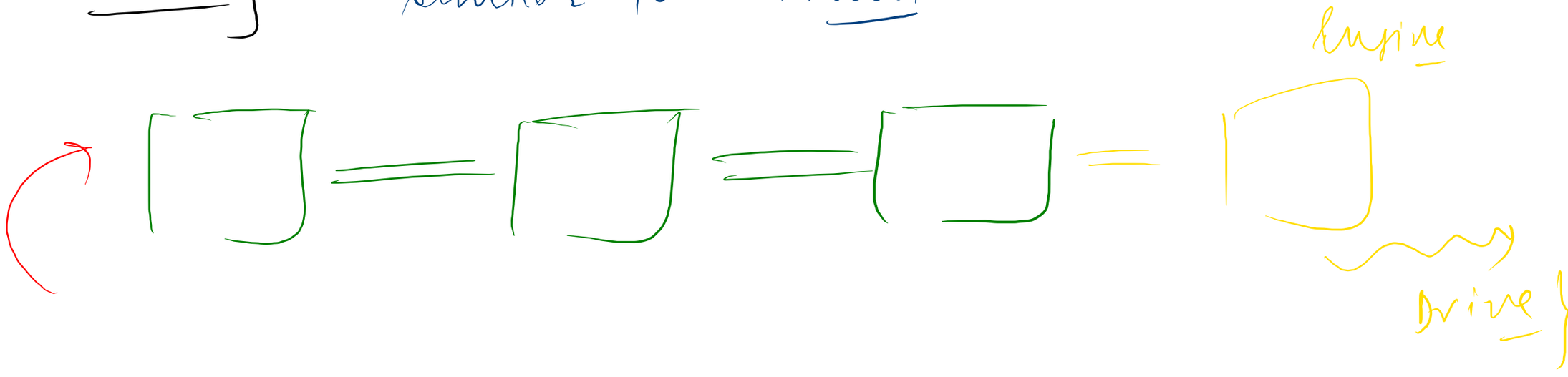
Swapping



Fragmented Memory

What LL?

similar to Traits



Linked List

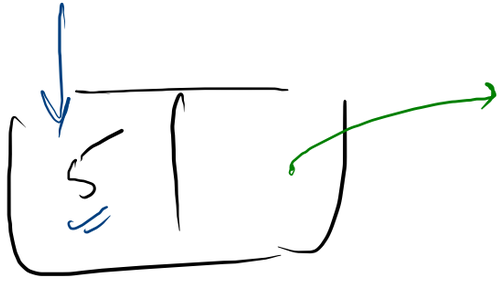
Collection of List Nodes

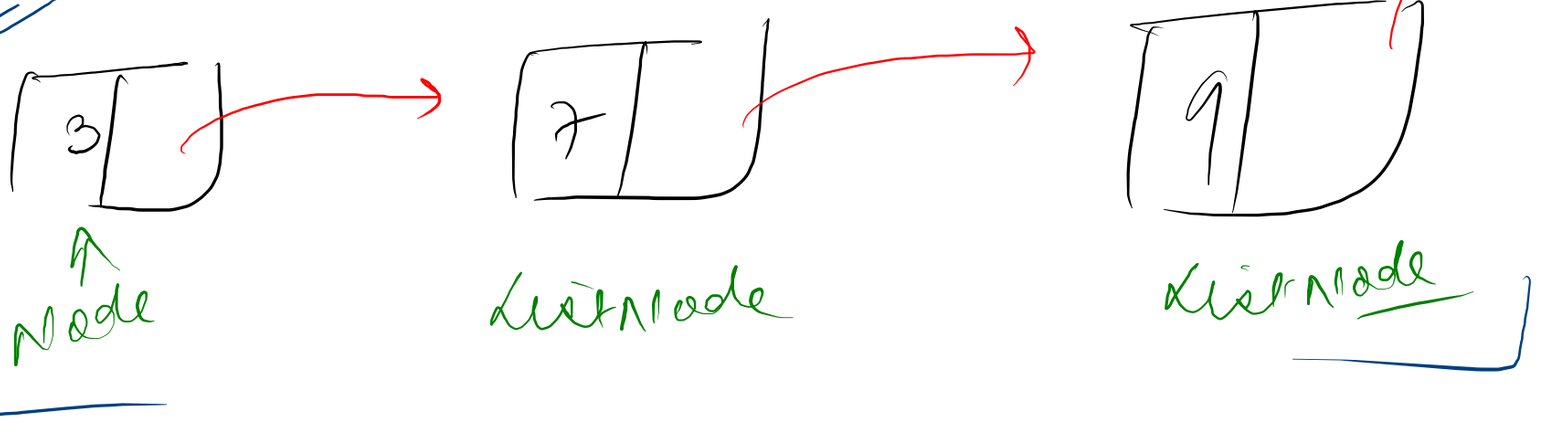
List Node

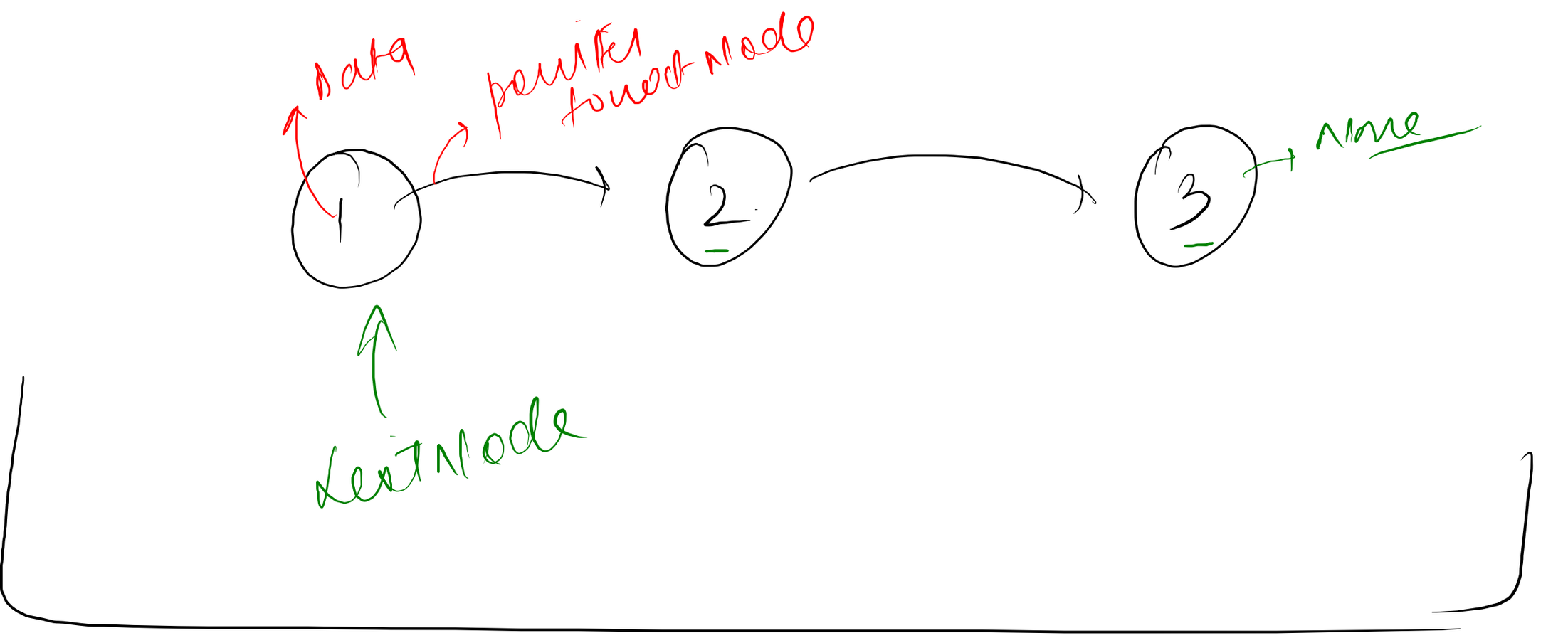
Data

Reference to the next node

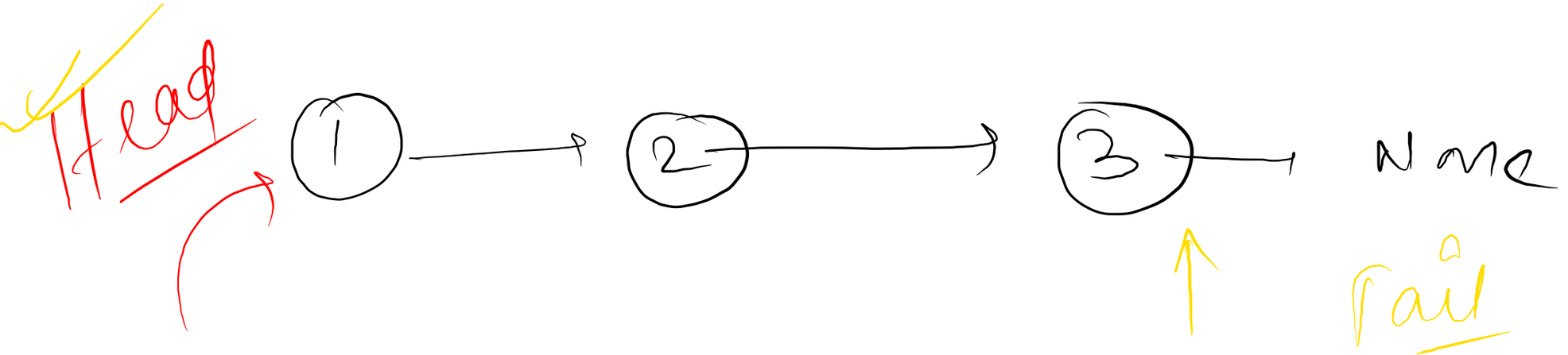
Next Node

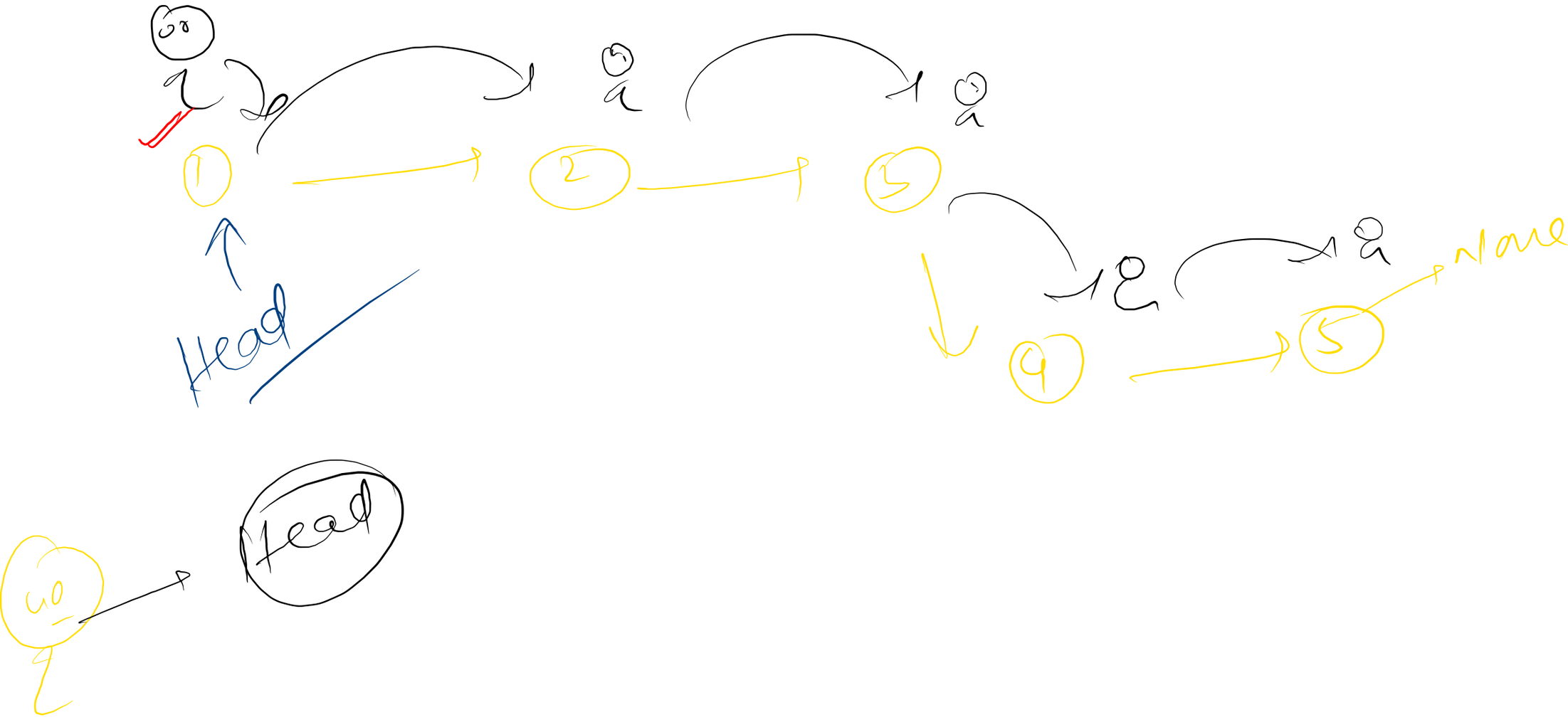






Linked List


Head
1
2
3
None
tail


1
2
5
9
5
Head
Head
None

Linked List → ① collection of ListNode

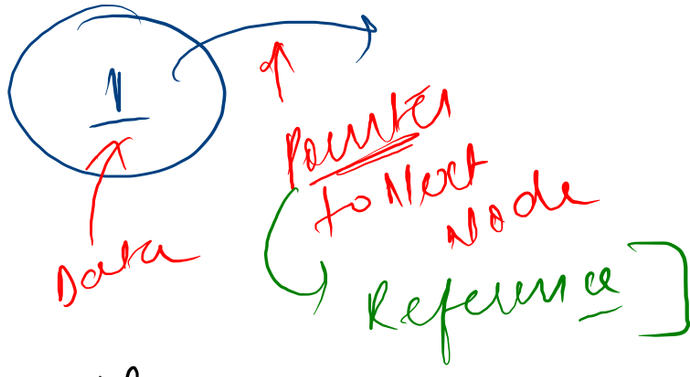
What

Why

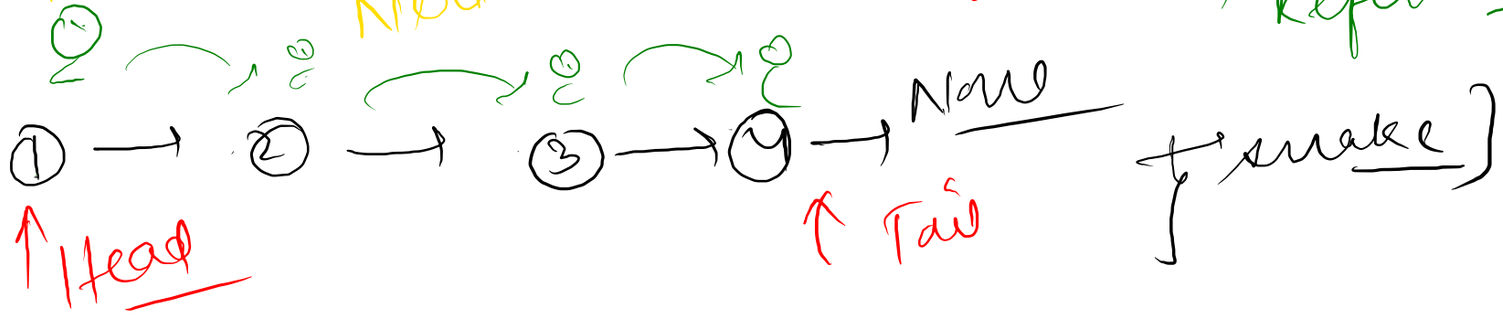
② ListNode

Data

Reference to Next Node

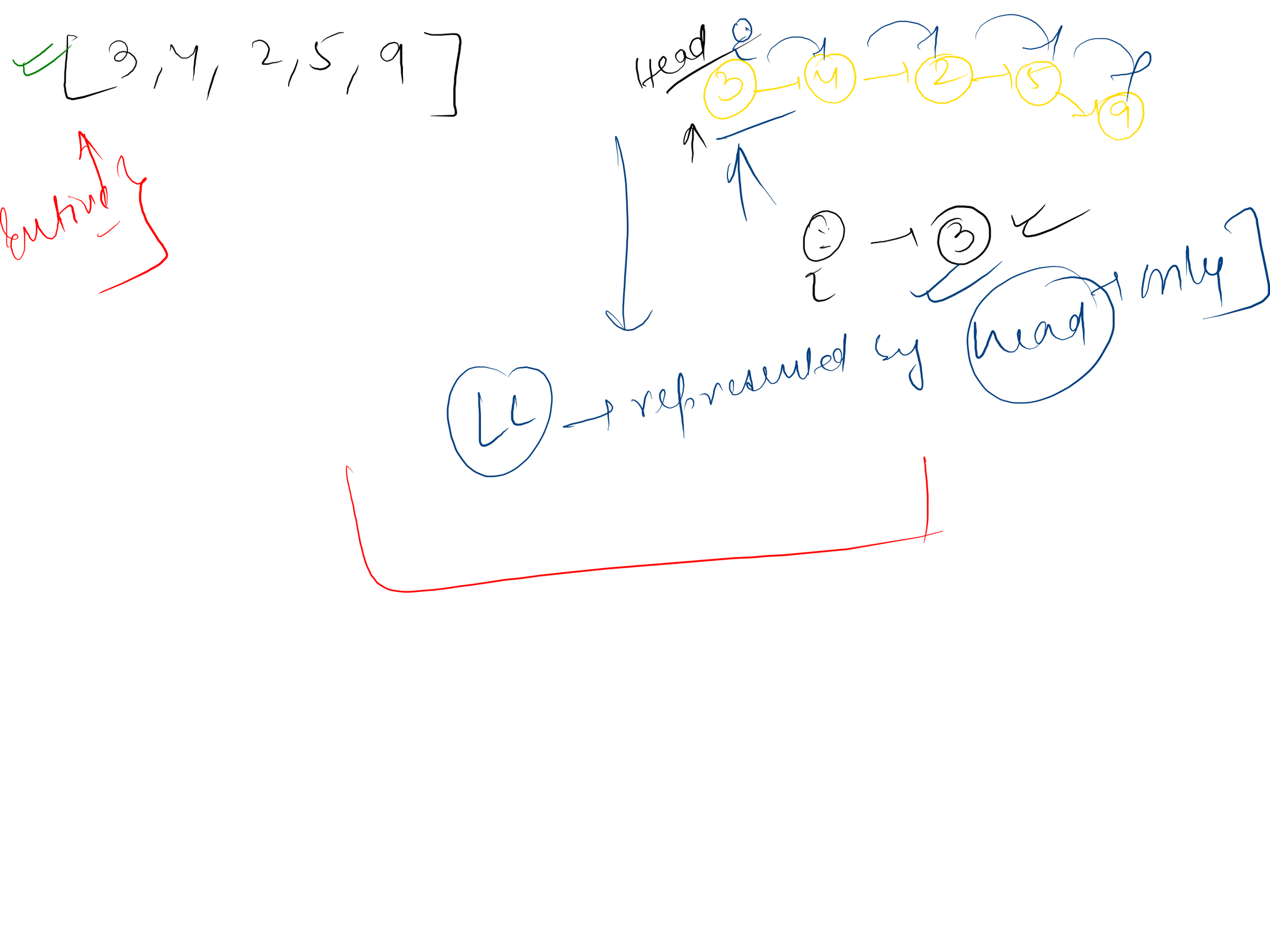


③

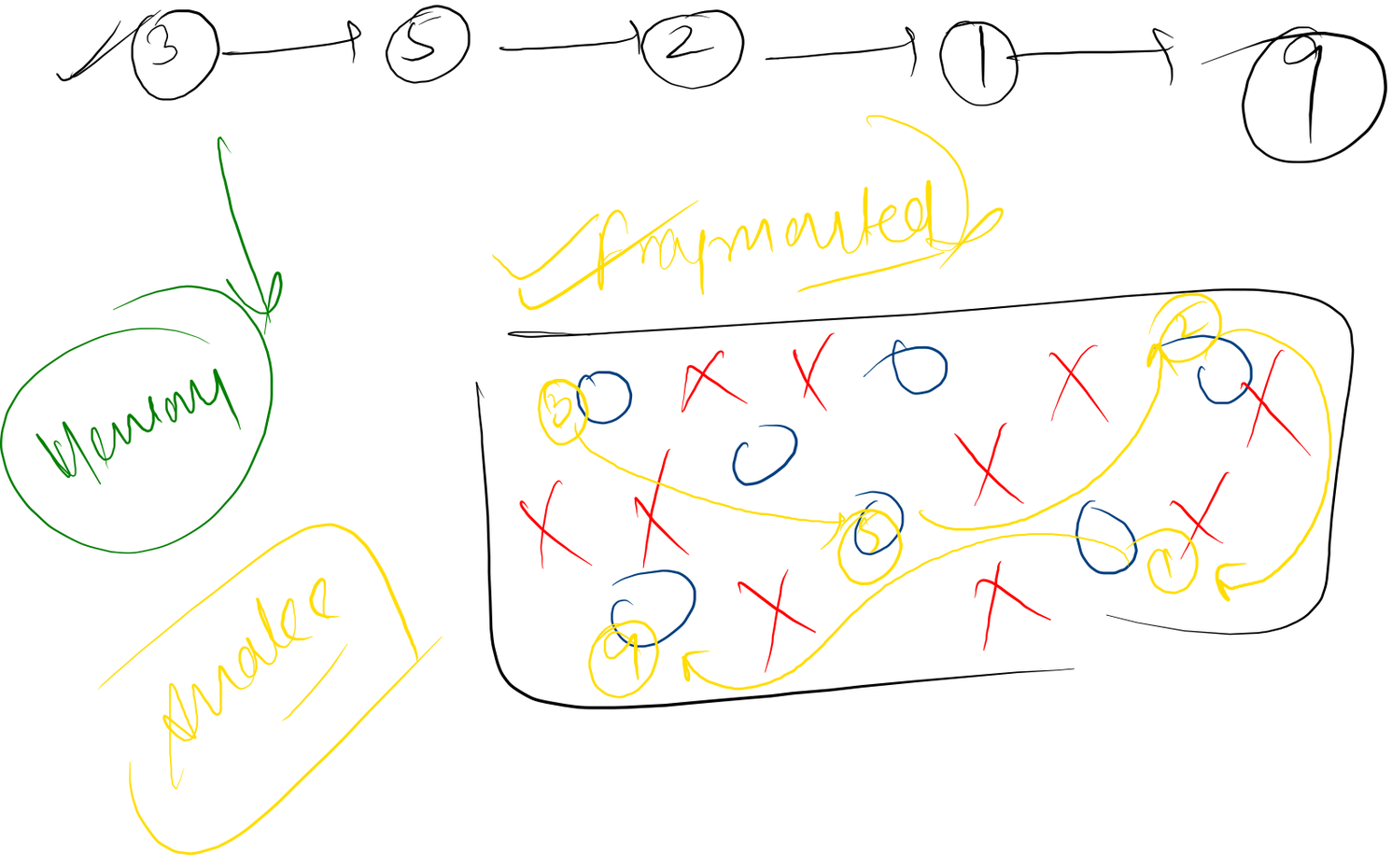


④ → know head → Entire LL

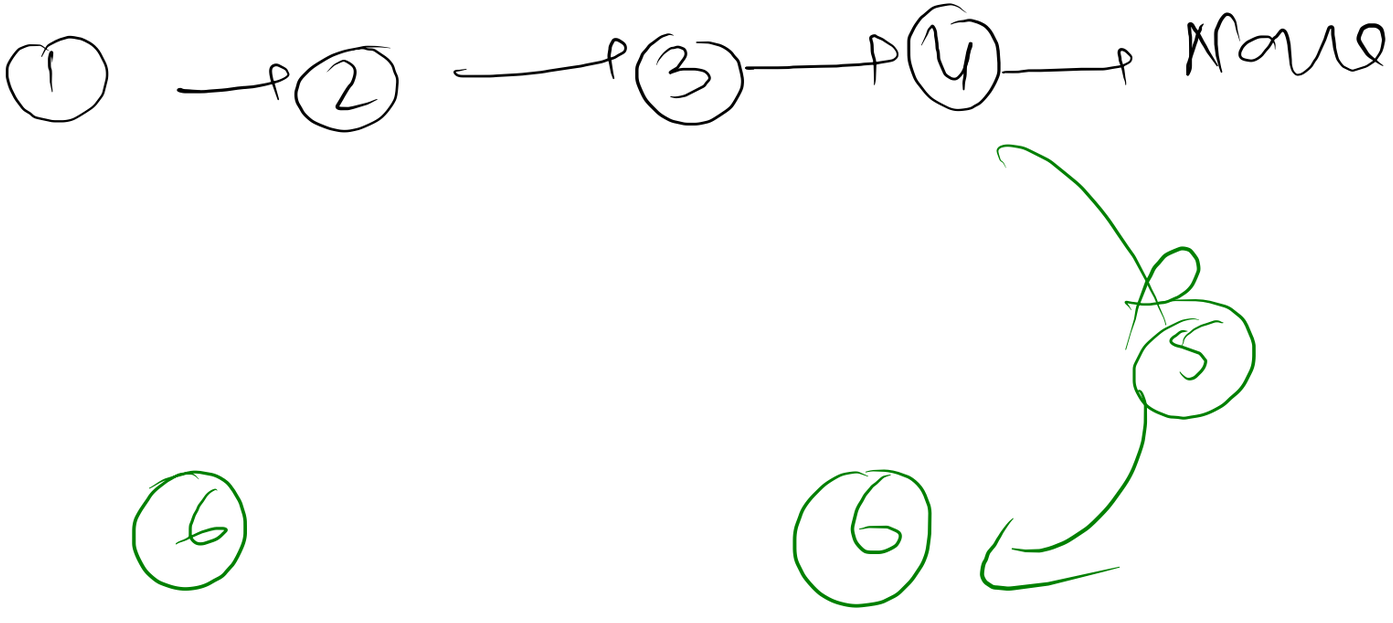

LL
collection of list nodes
Head


[3, 4, 2, 5, 9]
Head
3
4
2
5
9
Linked
LL → represented by
Head
only

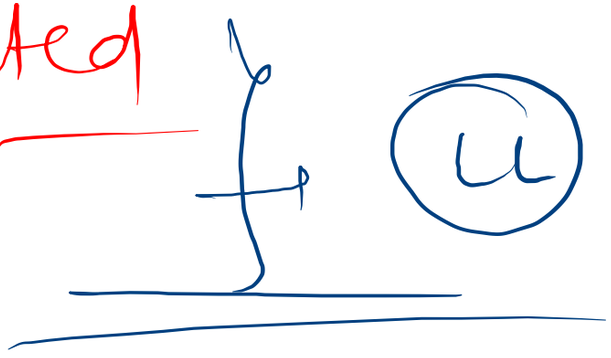
Single Directed → Single Linked List

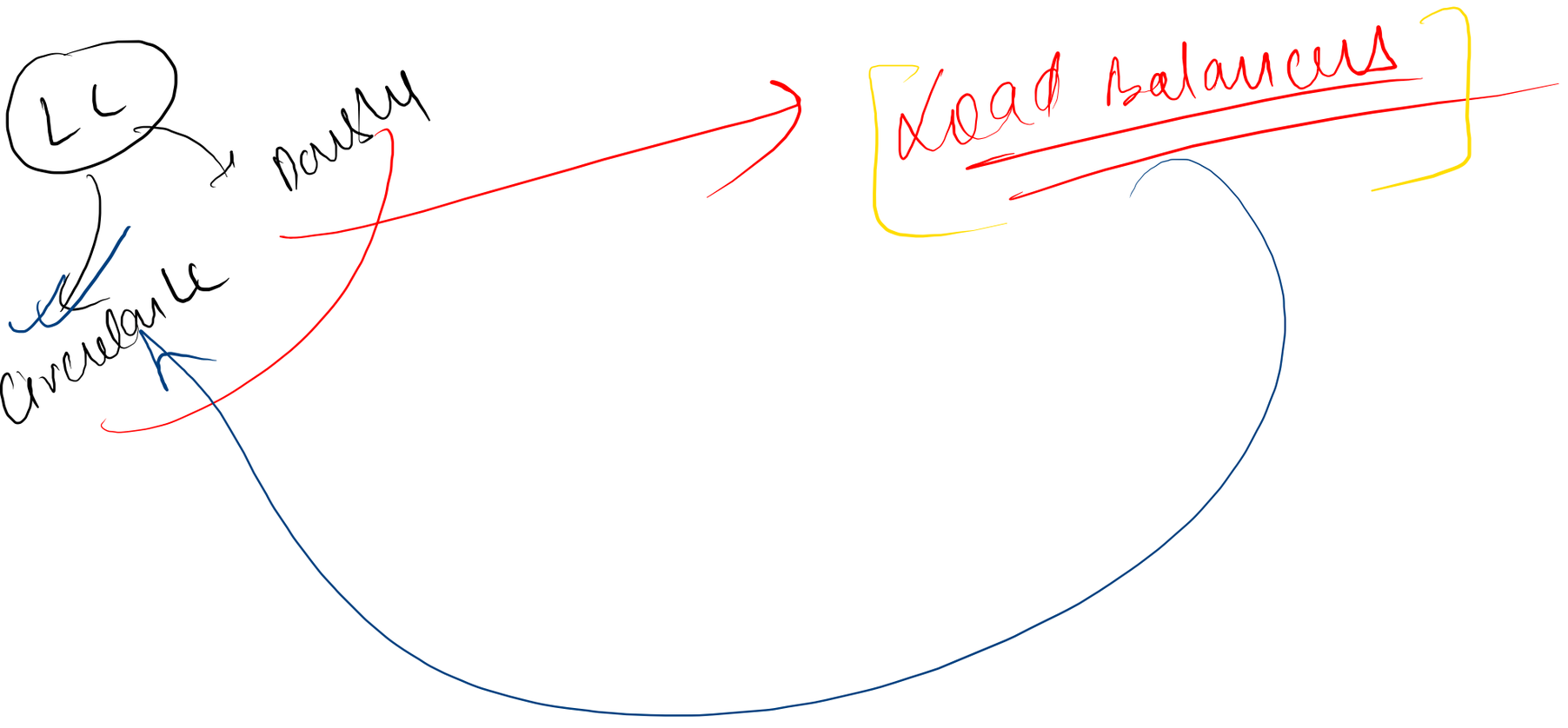

3
5
2
1
9
Memory
Fragmented
Holes
0 → free
X → occupied

Vectors → LL → Dynamic size



(2) Fragmented

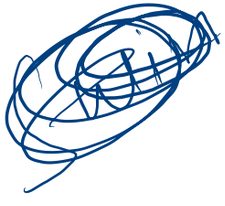



LC
Load Balancer

OS → Resource scheduling → Circular dependency

↓

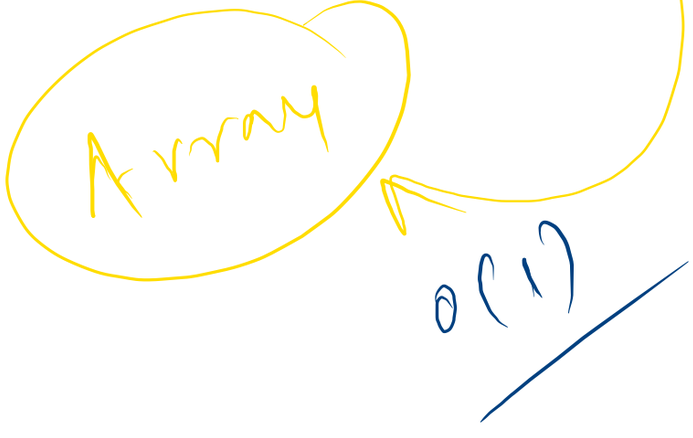
Spring → says no loop → Circular dependency

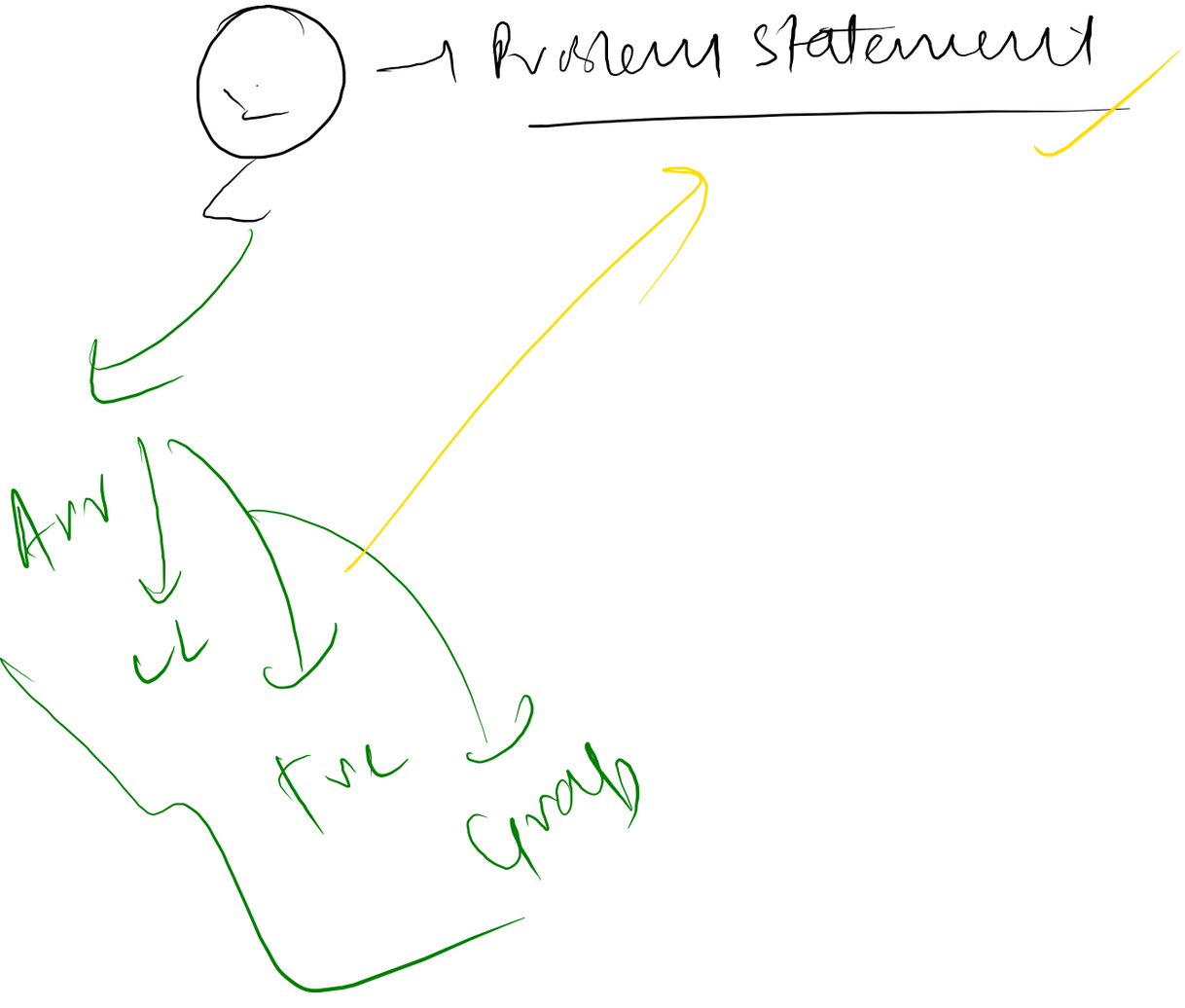


L9 → challenges

① Extra Memory → Address of next

② No index based retrieval → O(n)



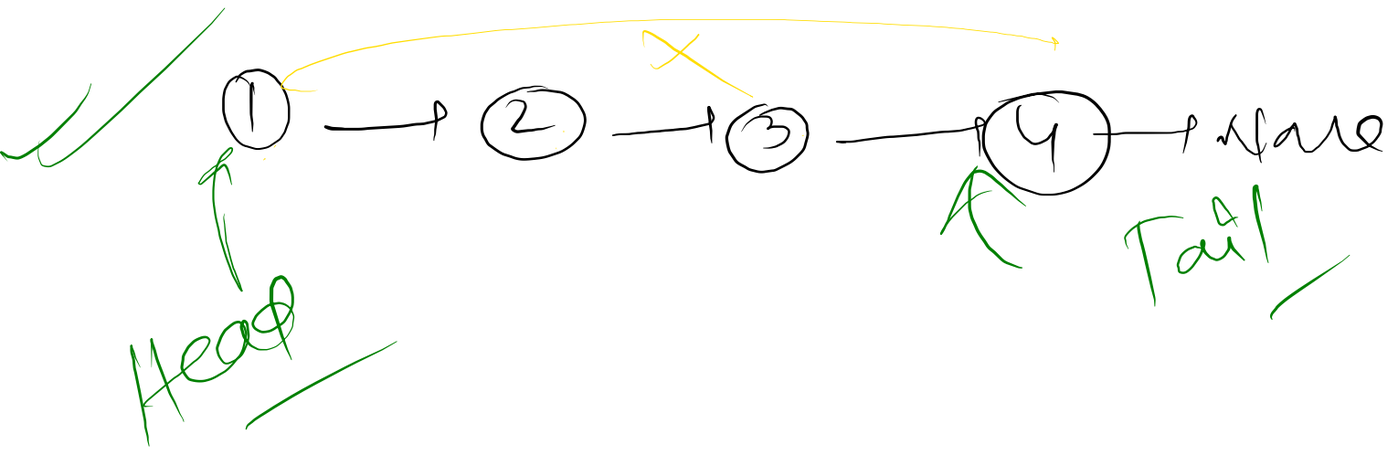

Problem Statement
Arr
ul
Tuc
Group

Array ⟶ Fast index base data]

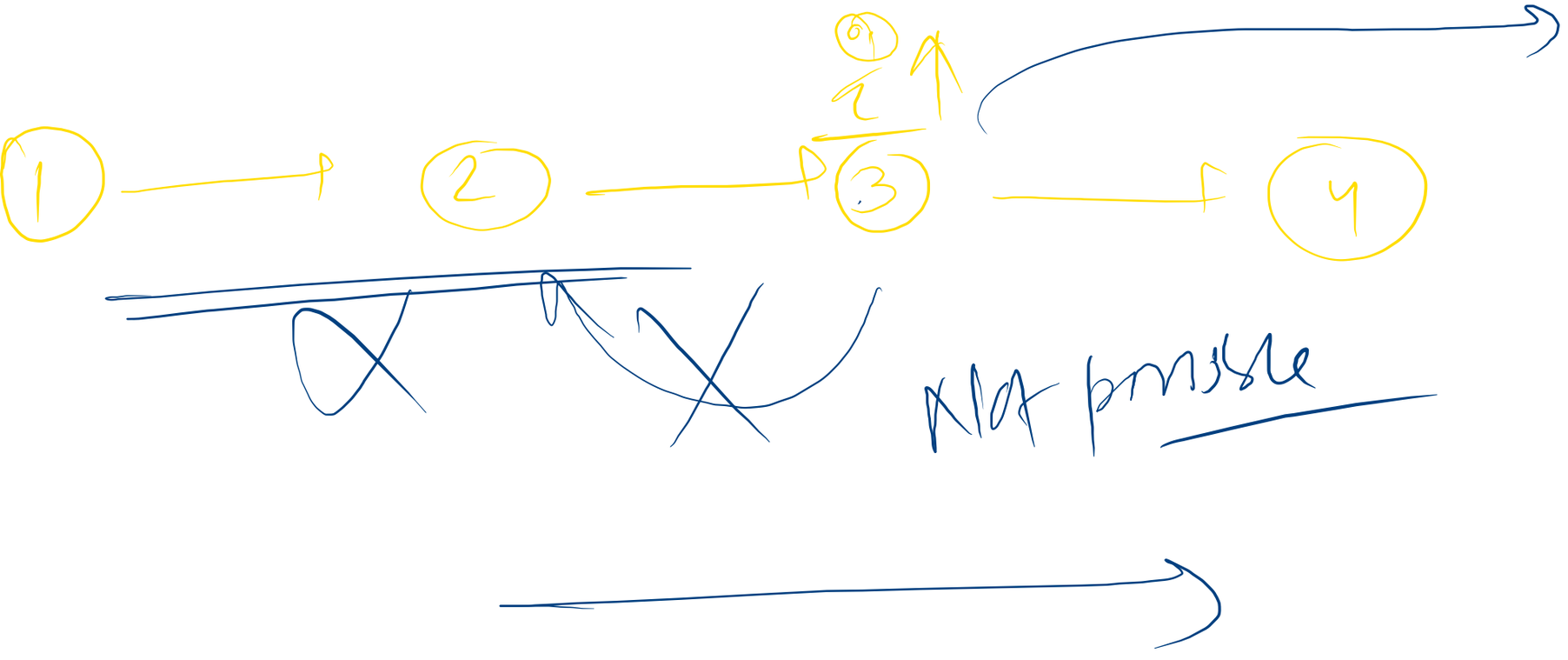
LL

↓ Dynamic size

Fragmented Memory


1
2
3
4
Null
Head
Tail


3
4
5
2


1
2
3
4
NIA promise

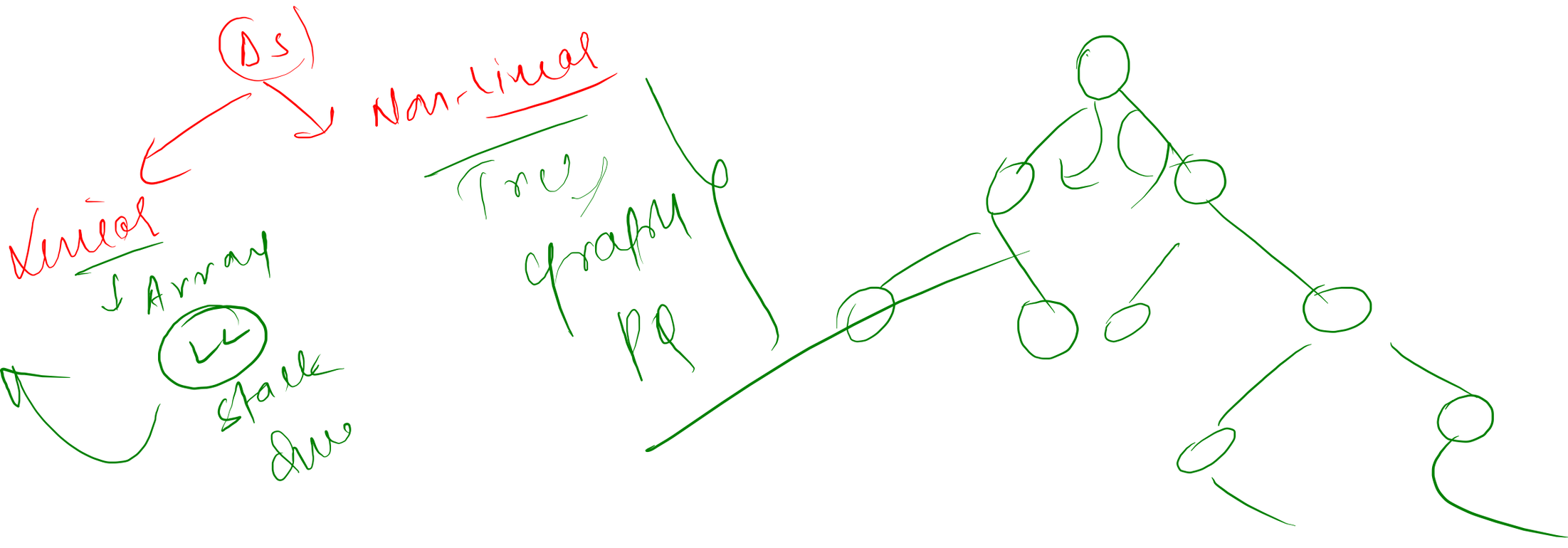

Array

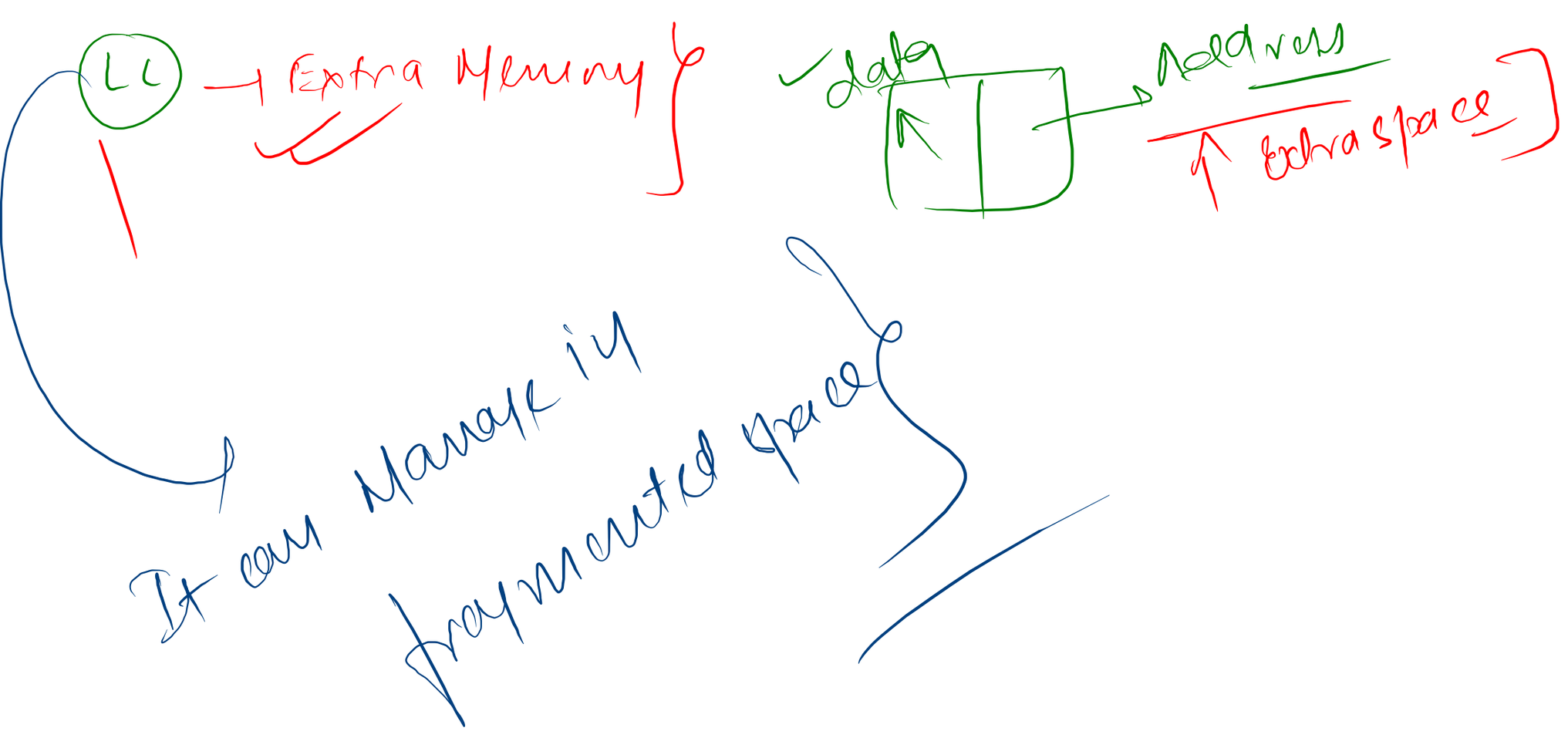

Return
Cart

LC

↑ can work in fragmented Memory also


Heap

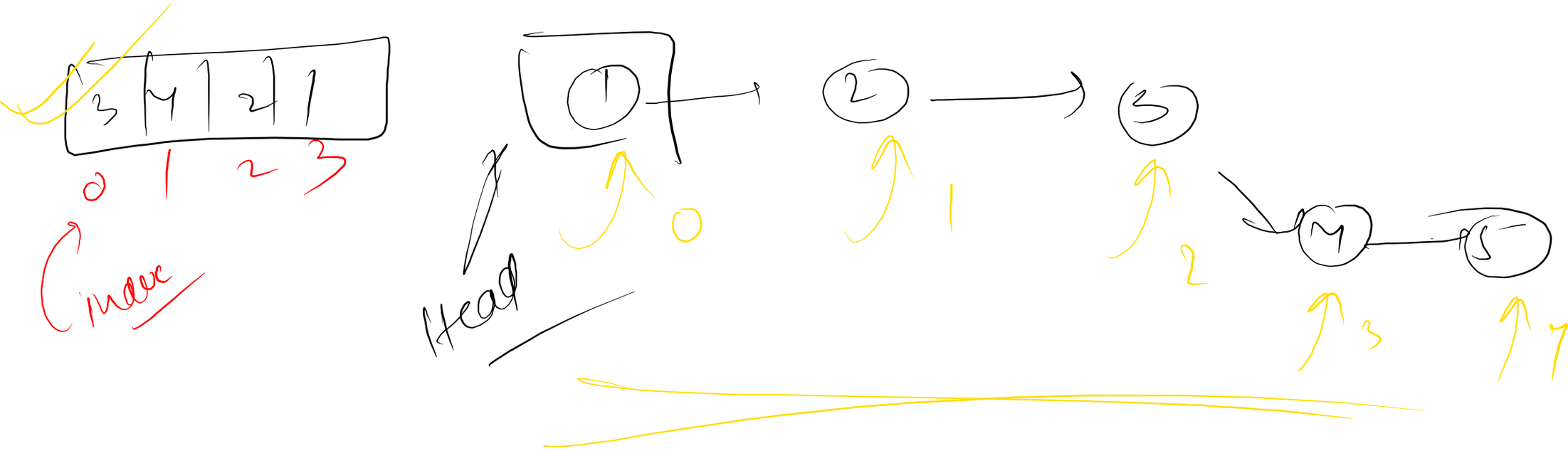

DS
Non-Linear
Linear
Array
LL
Stack
Que
Tree
graph
PQ

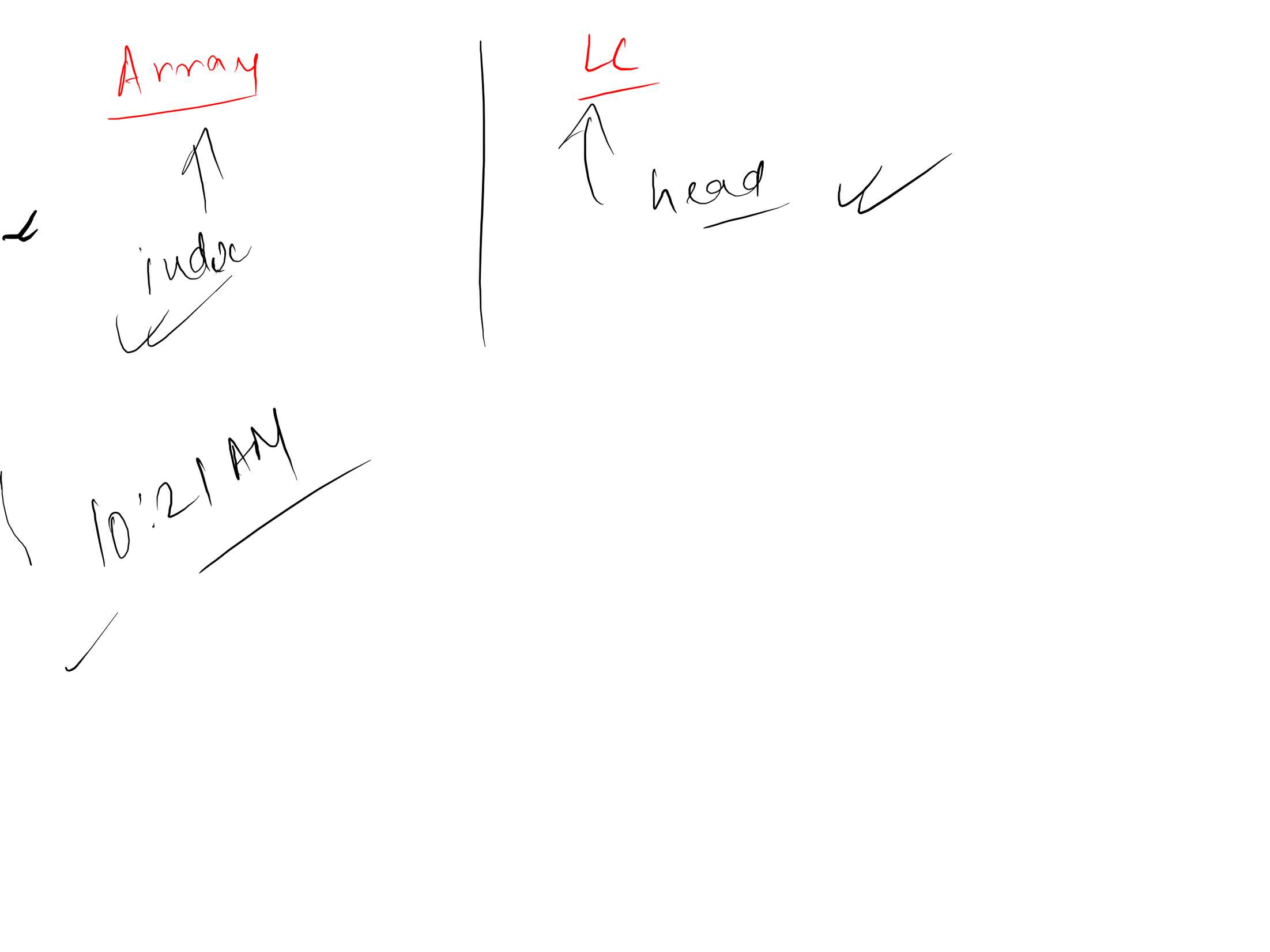

LL
Extra Memory
data
address
extra space
It can Manage in
fragmented space

Python → Linked list → Linked list = collection of Node objects

1. Iterate
2. Length
3. Insertion
4. Deletion

} basic operations → Problems


3 4 2 1
0 1 2 3
index
Head
1
2
3
4
5
0
1
2
3


Array
index
LL
head
10:21 AM

```
class Node :

    def __init__(self,data=None, next=None):
        self.data = data
        self.next = next
```

object

Provide the methods

setting data / setting data

next

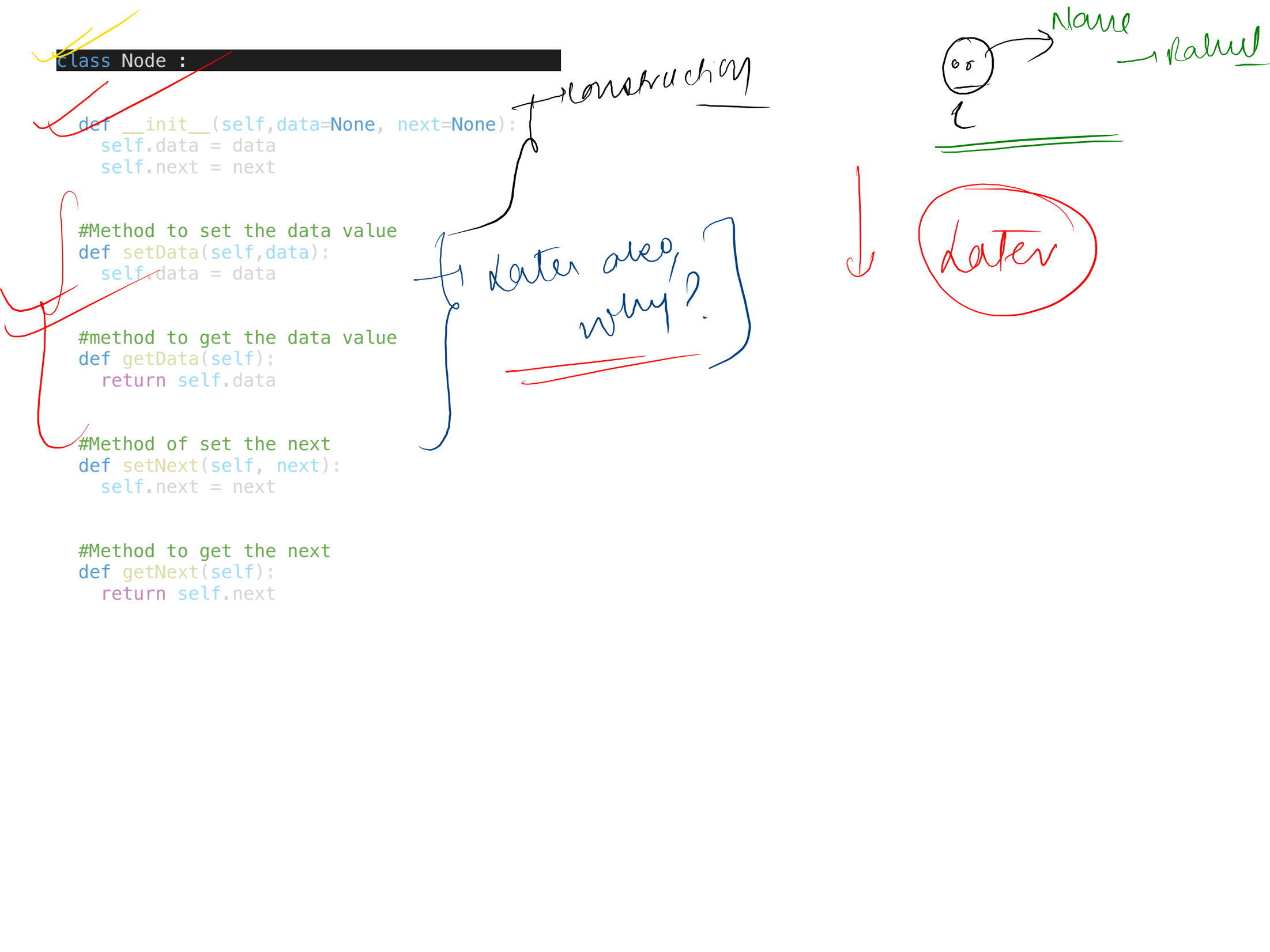
Node object

class

curr Node

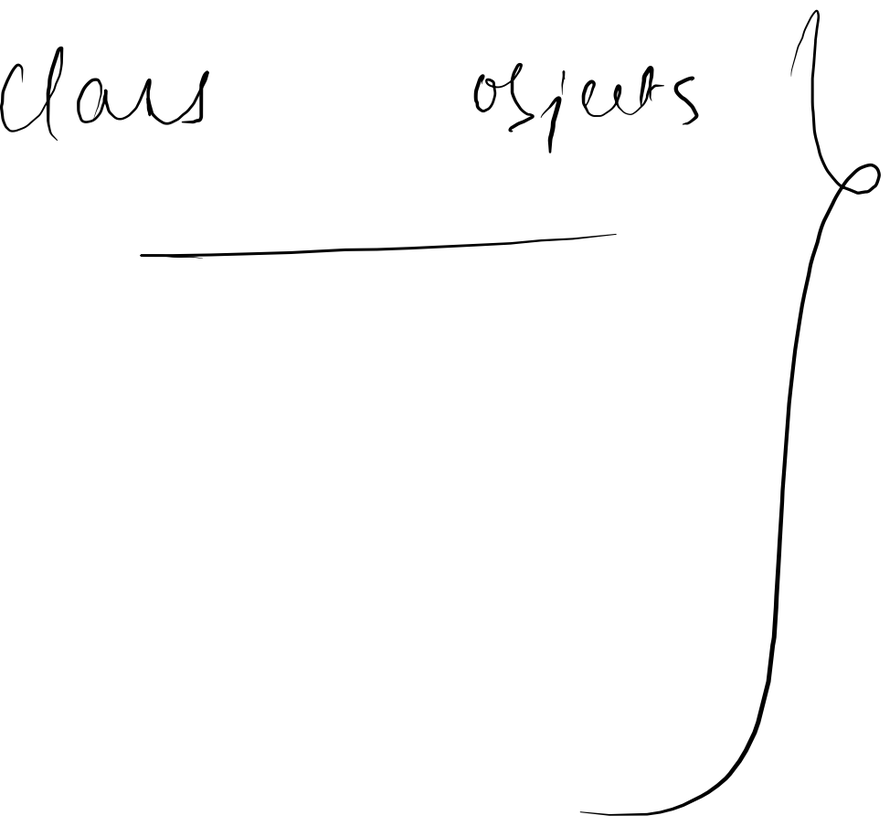
data

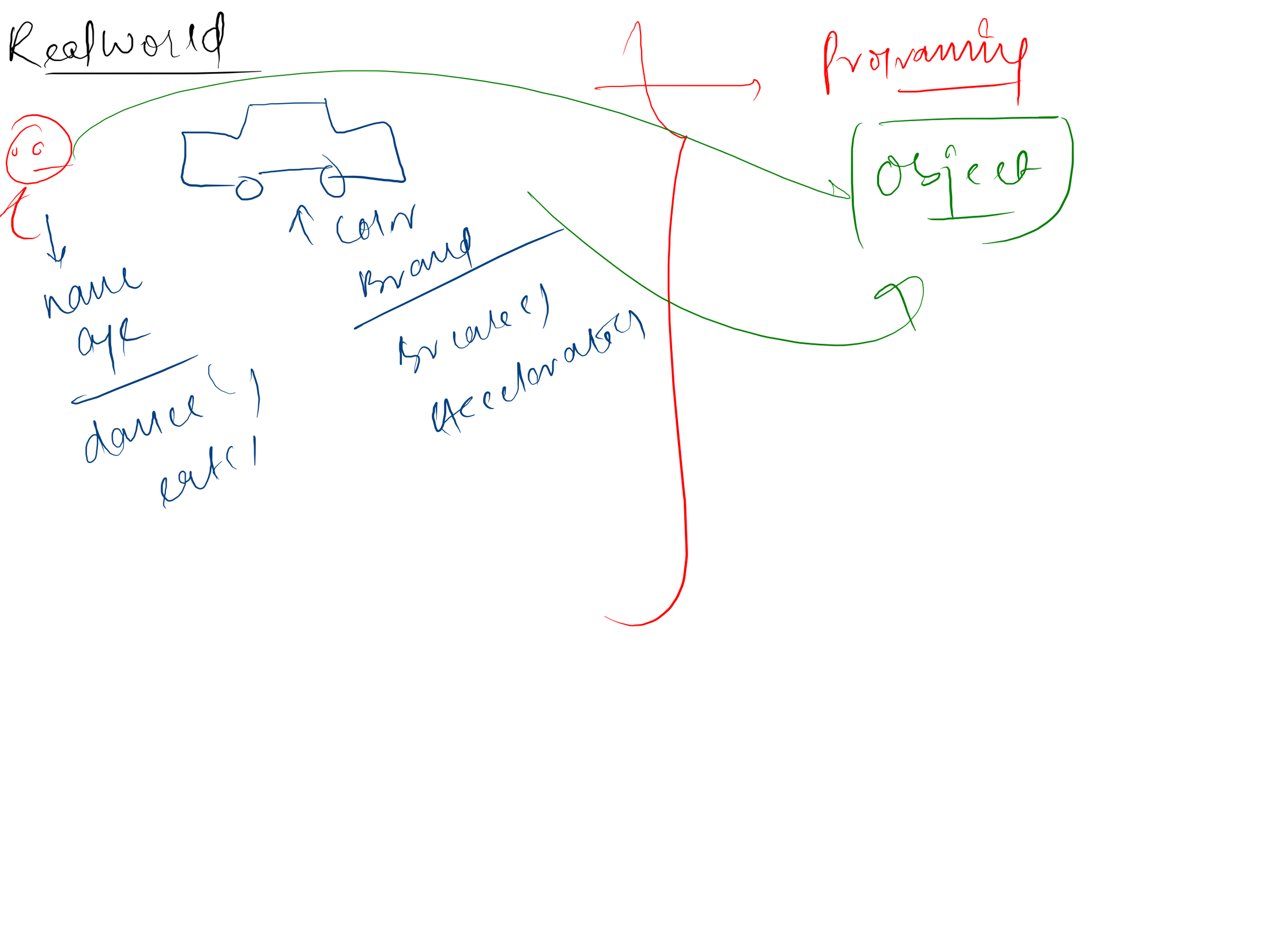
```
class Node :

    def __init__(self,data=None, next=None):
        self.data = data
        self.next = next

    #Method to set the data value
    def setData(self,data):
        self.data = data

    #method to get the data value
    def getData(self):
        return self.data

    #Method of set the next
    def setNext(self, next):
        self.next = next

    #Method to get the next
    def getNext(self):
        return self.next
```



class | objects

↑ Python


class
objects


Real world
Programming
Object
color
Brand
name
age
dance()
eat()
break()
Accelerate()

Class

↑ Blueprint

EveryHuman

↑ name
age
gender

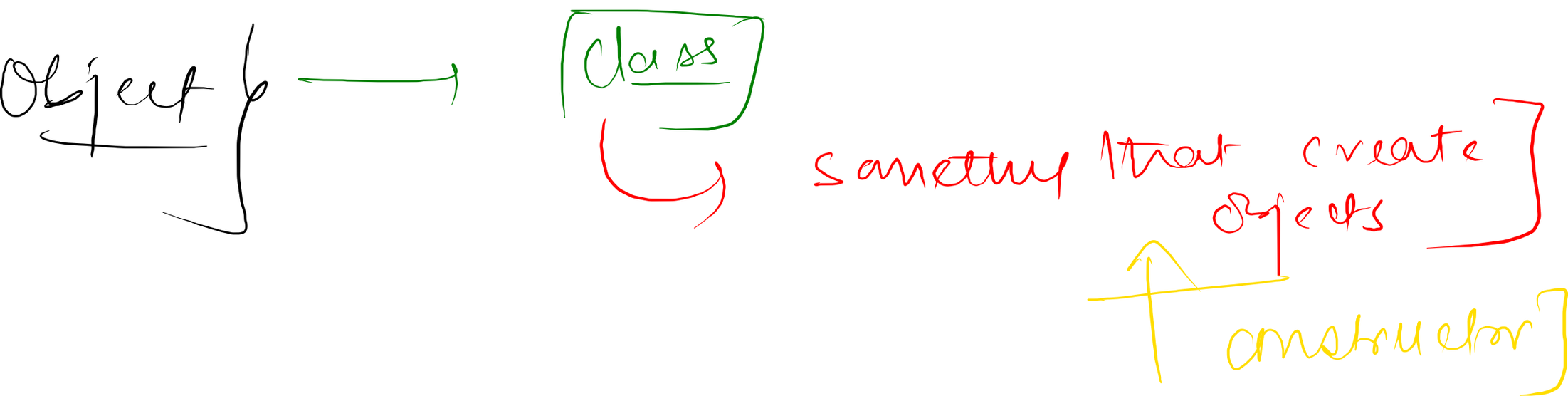
Vishnu | 99 | M

saadhvi | 94 | F

vanmathi | 93 | F

Class

Objects


Object
Class
something that create Objects
constructor

class Node

↳ objects } → Linked List }

```
#Create a linked list -> collection of list nodes


head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)


#Creating the linkage
head.setNext(node2)
node2.setNext(node3)
node3.setNext(node4)
```

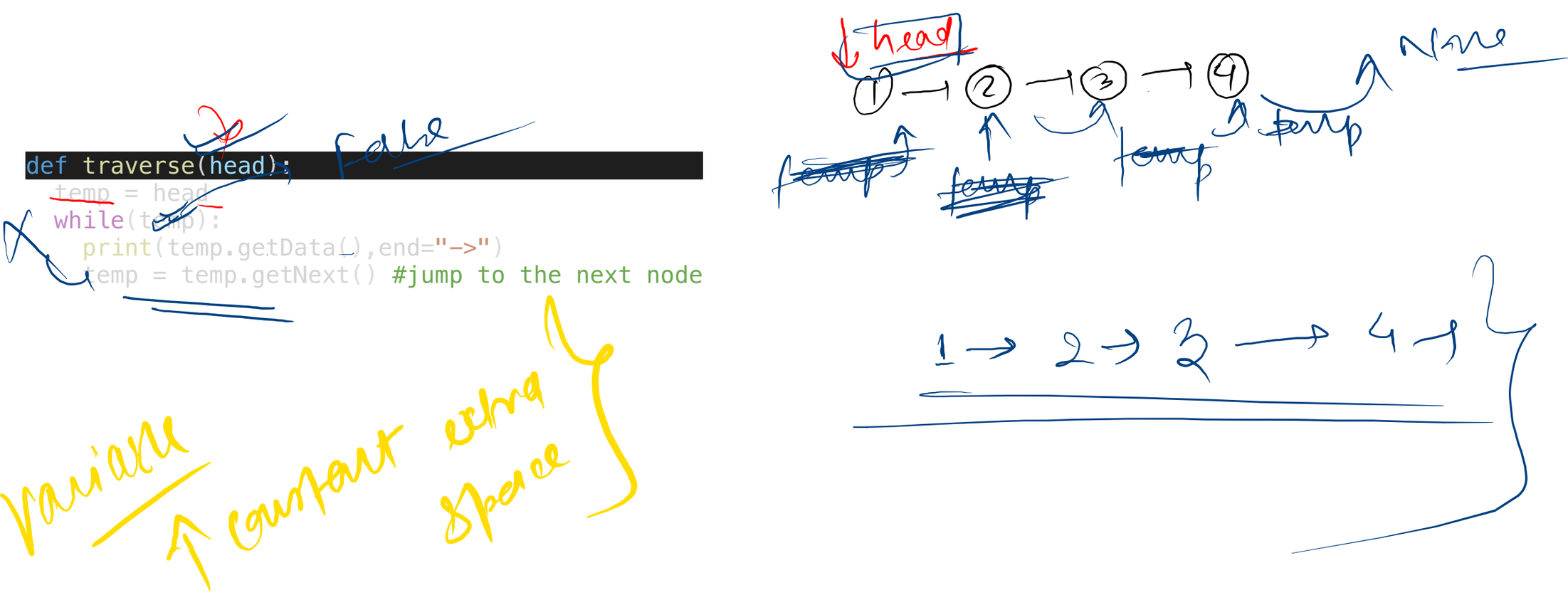
```
class Node :


  def __init__(self,data=None, next=None):
    self.data = data
    self.next = next


  #Method to set the data value
  def setData(self,data):
    self.data = data


  #method to get the data value
  def getData(self):
    return self.data


  #Method of set the next
  def setNext(self, next):
    self.next = next


  #Method to get the next
  def getNext(self):
    return self.next
```

Traverse this LL

Print all the elements of a LL

1 2 3 4

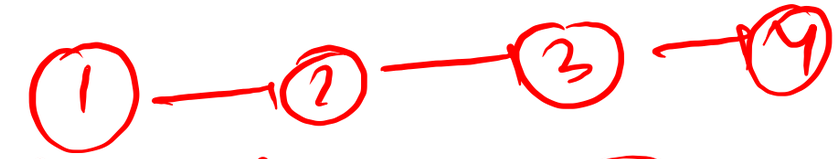
```
def traverse(head):
    temp = head
    while(temp):
        print(temp.getData(),end="->")
        temp = temp.getNext() #jump to the next node
```



Variable → constant extra space

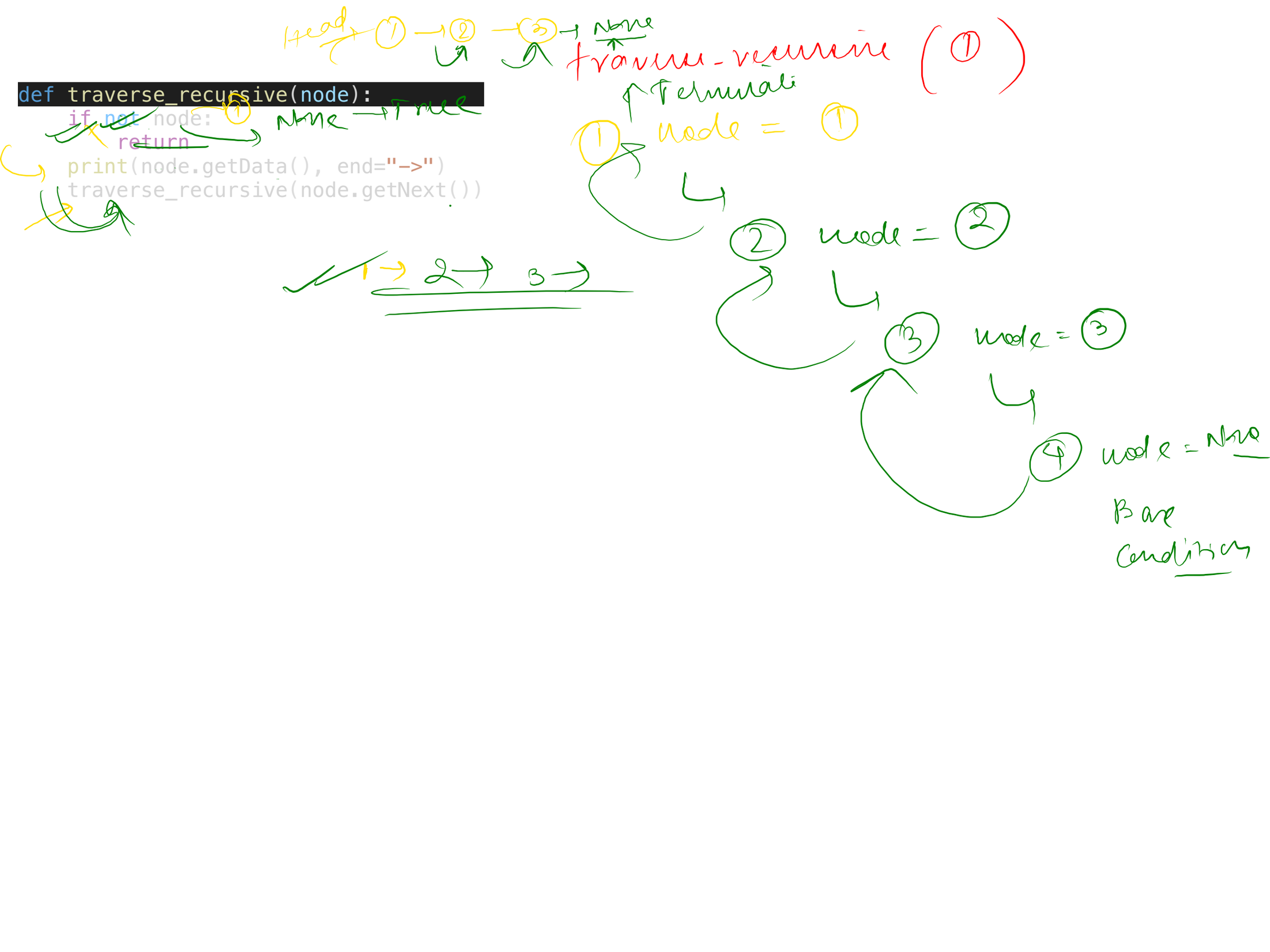
[3, 5, 2, 1, 4]

Traverse → Loop } O(n)



↑ Head

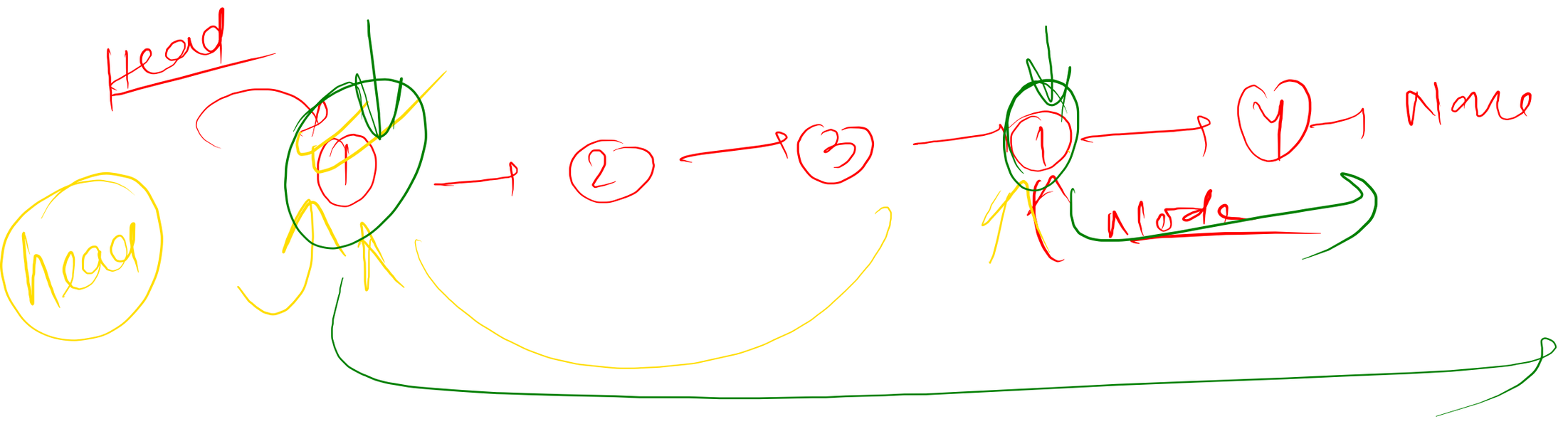
Traverse { Loop } O(n)

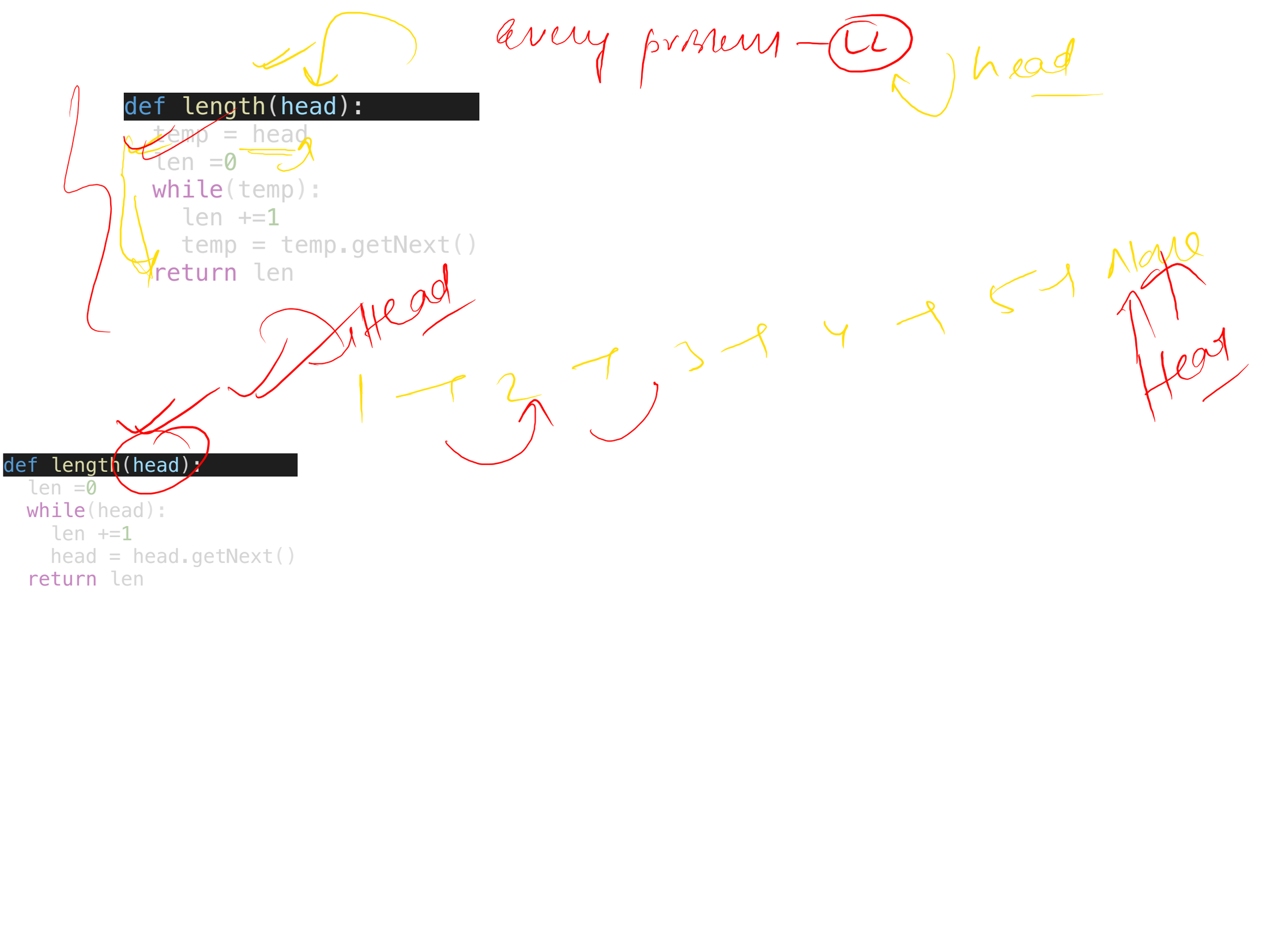

def traverse_recursive(node):
if not node:
return
print(node.getData(), end="->")
traverse_recursive(node.getNext())
head
None
traverse-recursive (1)
terminal
node = 1
node = 2
node = 3
node = None
Base
Condition
1 → 2 → 3 →

of → head of the linked list

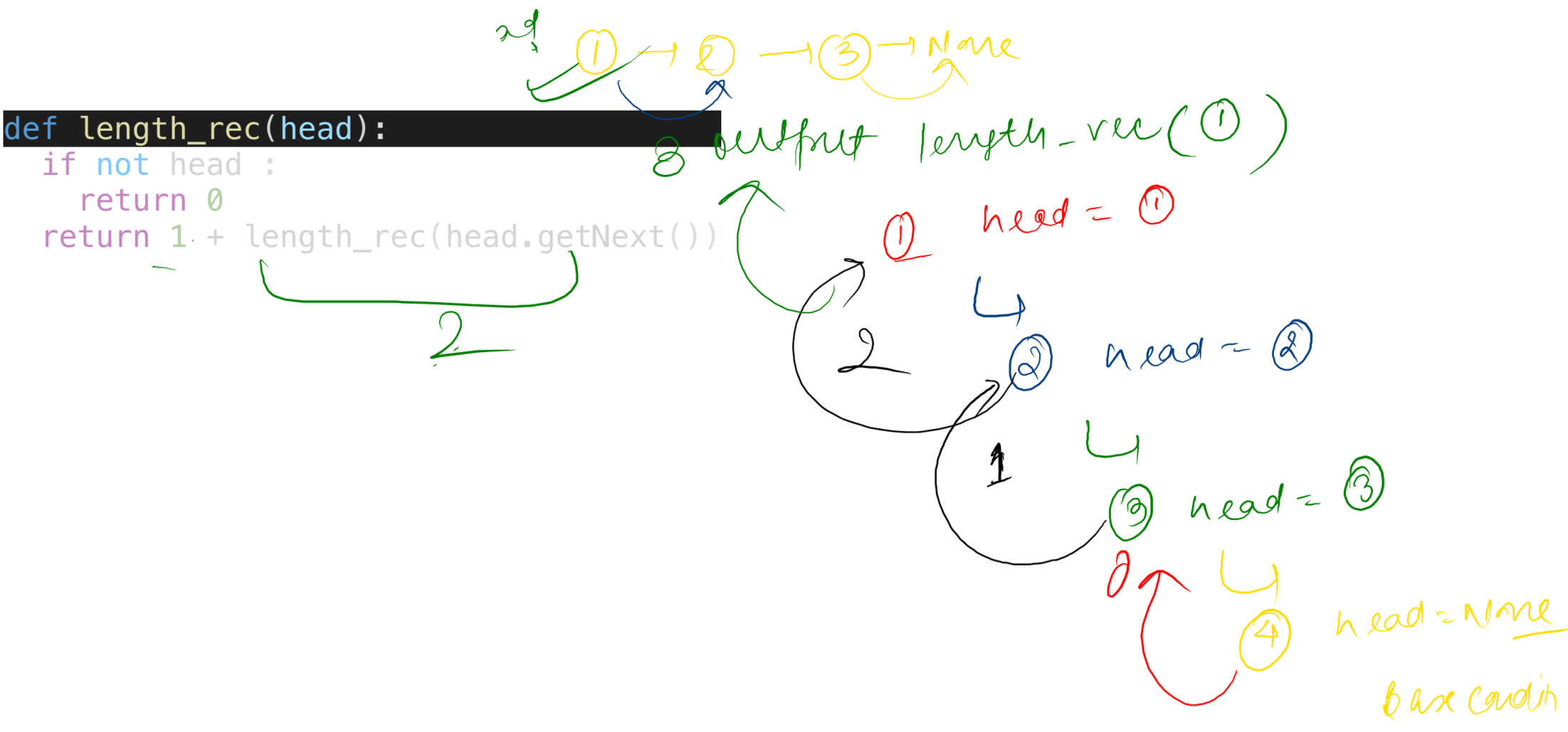
↑ find me the length

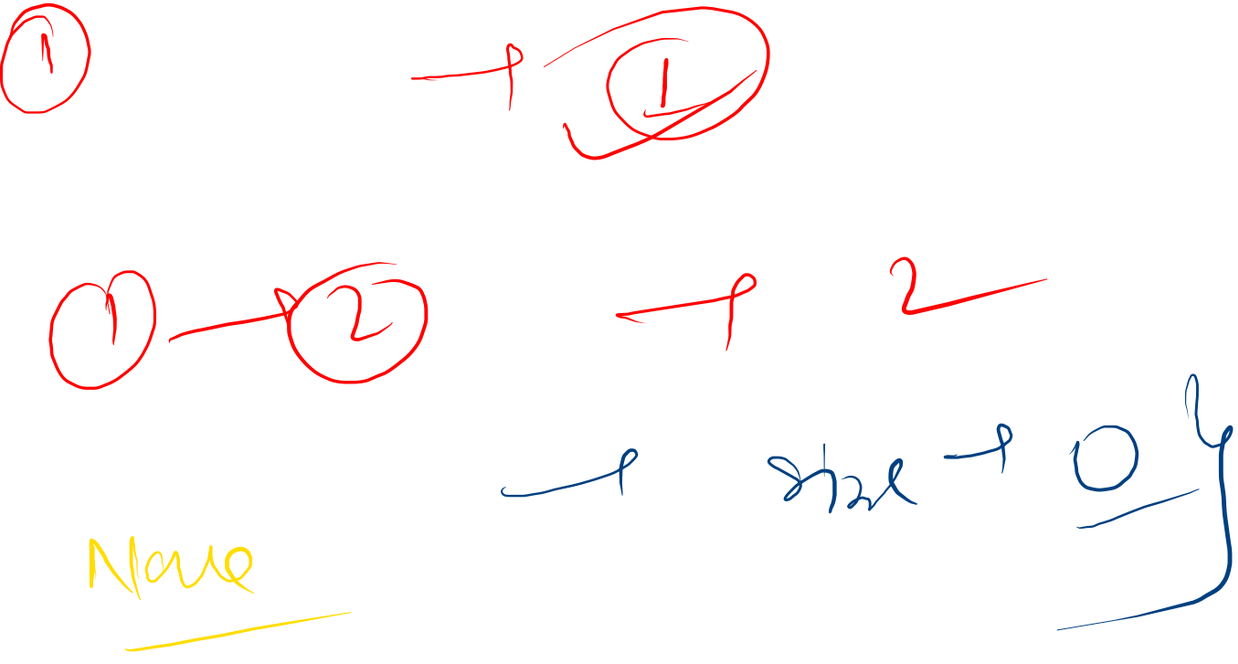
↑ No of nodes are there


Head
1
2
3
1
4
Name
Node
Head

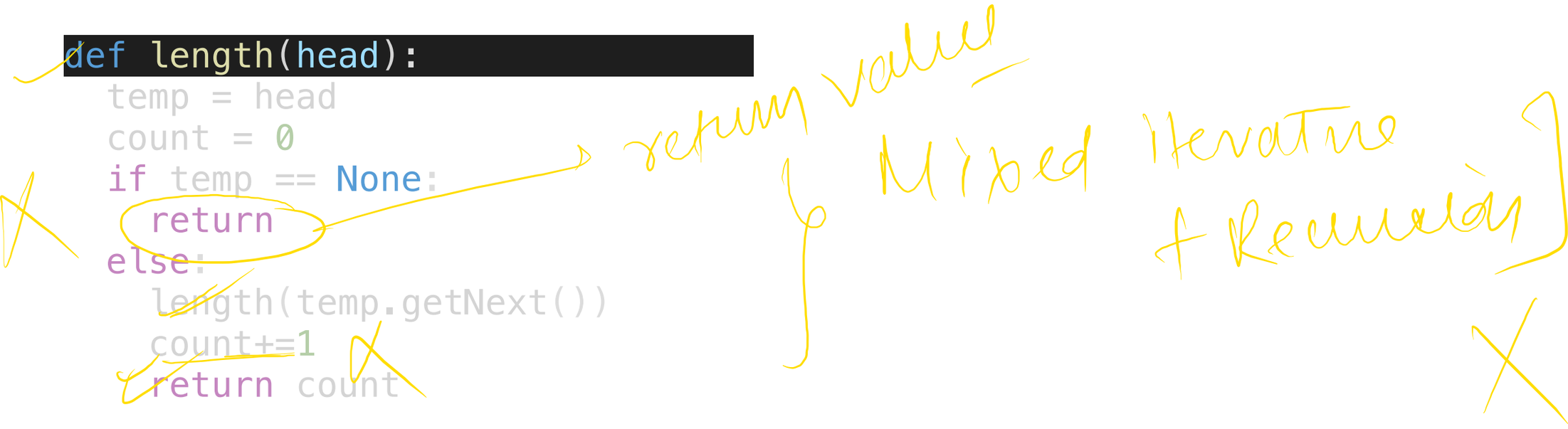

every problem — LL
head
def length(head):
temp = head
len =0
while(temp):
len +=1
temp = temp.getNext()
return len
Dhead
1
2
3
4
5
None
Head
def length(head):
len =0
while(head):
len +=1
head = head.getNext()
return len

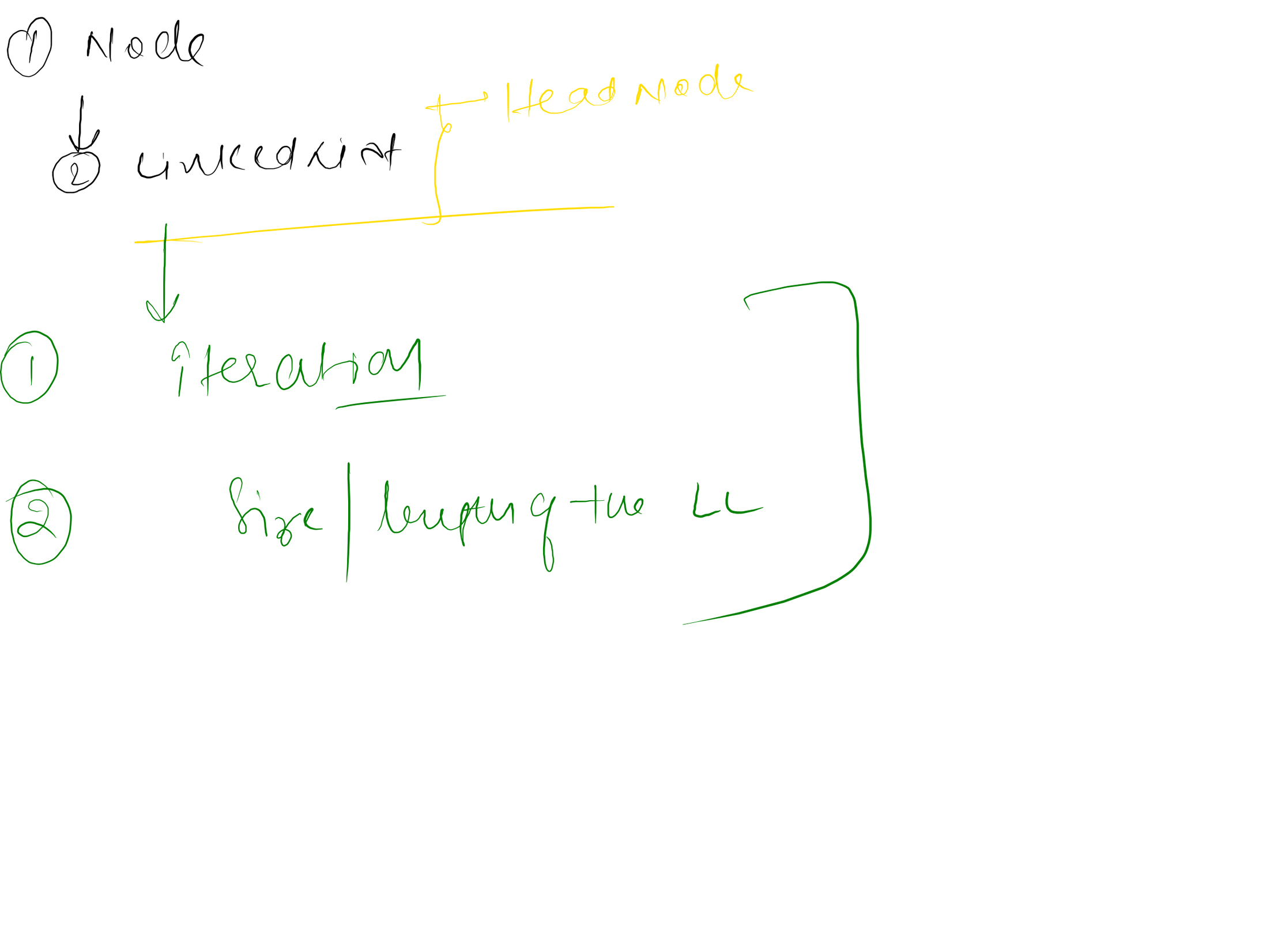
```python
def length_rec(head):
  if not head :
    return 0
  return 1 + length_rec(head.getNext())
```




1
1
1
2
2
size
0
None

```
def length(head):
    temp = head
    count = 0
    if temp == None:
        return
    else:
        length(temp.getNext())
        count+=1
        return count
```

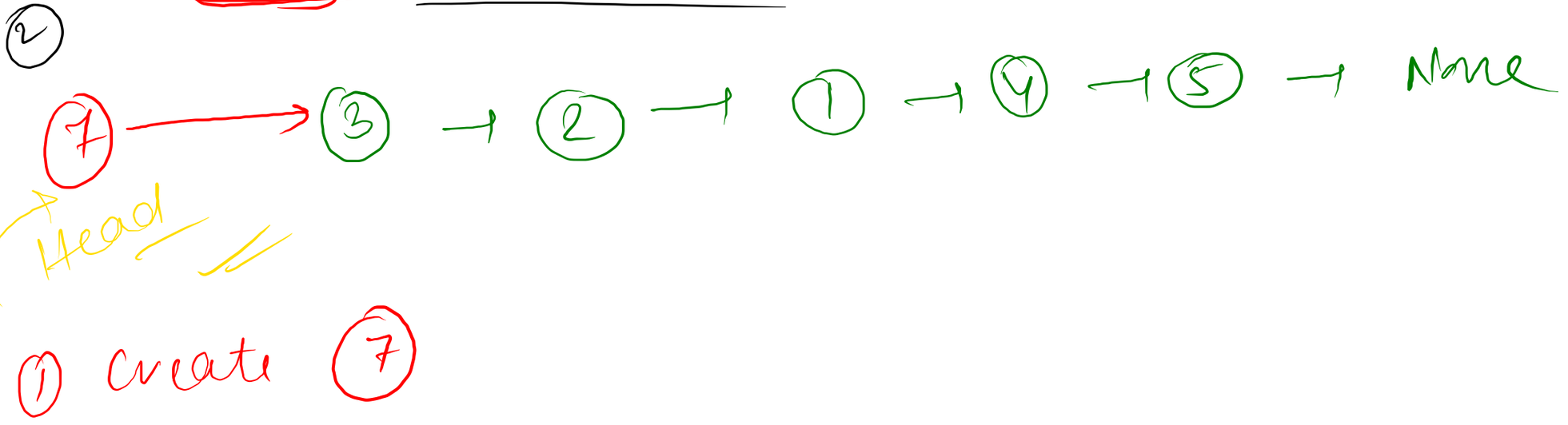



1 Node
2 LinkedList
Head Node
1 Iteration
2 Size | keeping the LL

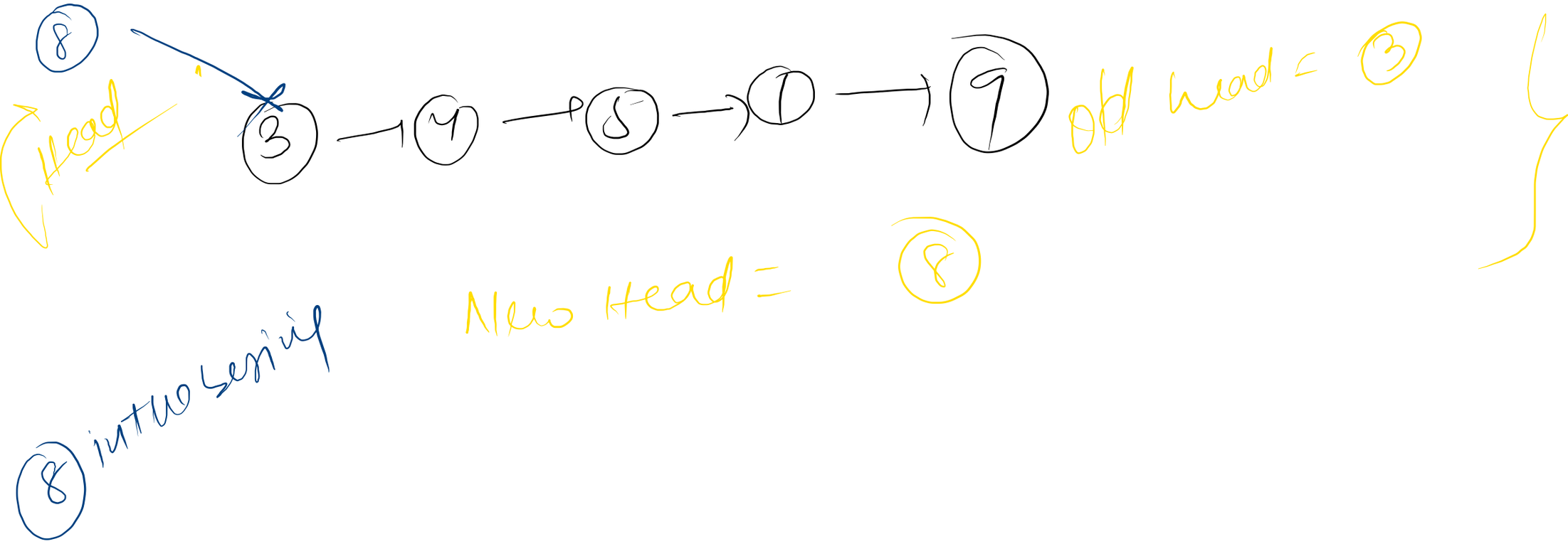
Insertion

7 insert

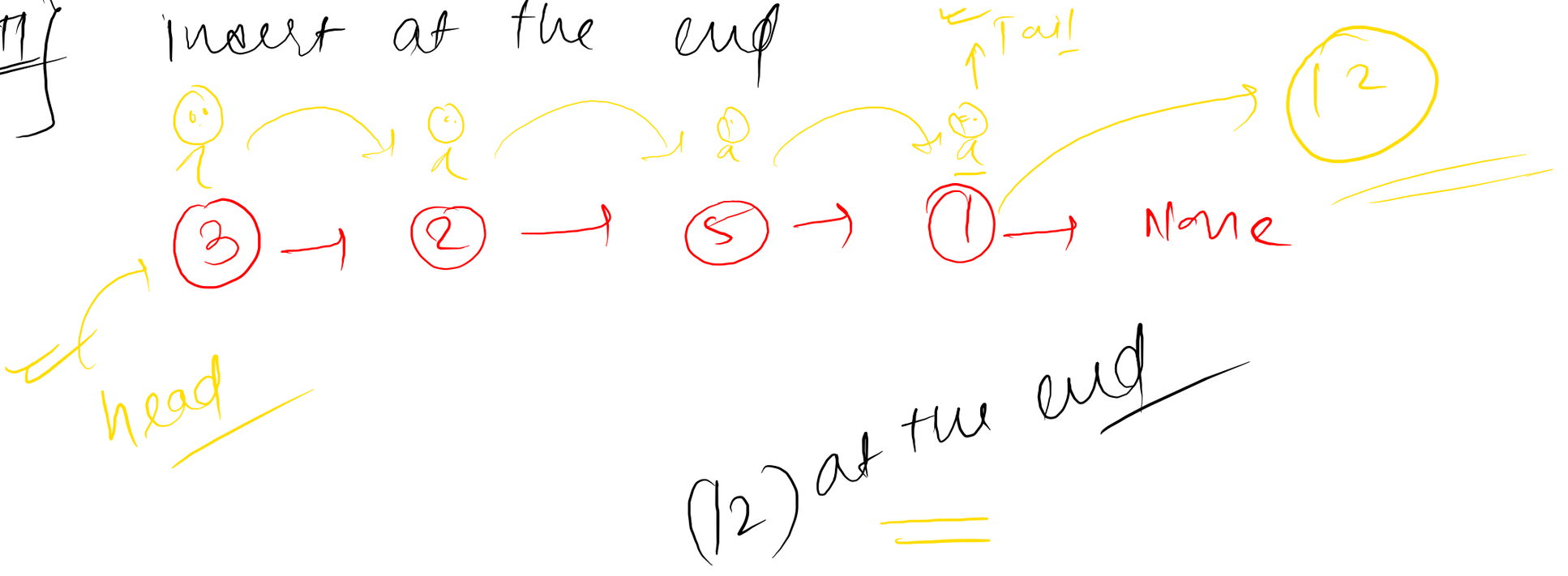
insert 7 at the Beginning



1) Create 7


8
Head
3
4
5
1
9
old Head = 3
New Head = 8
8 initialising

Case II) Insert at the end

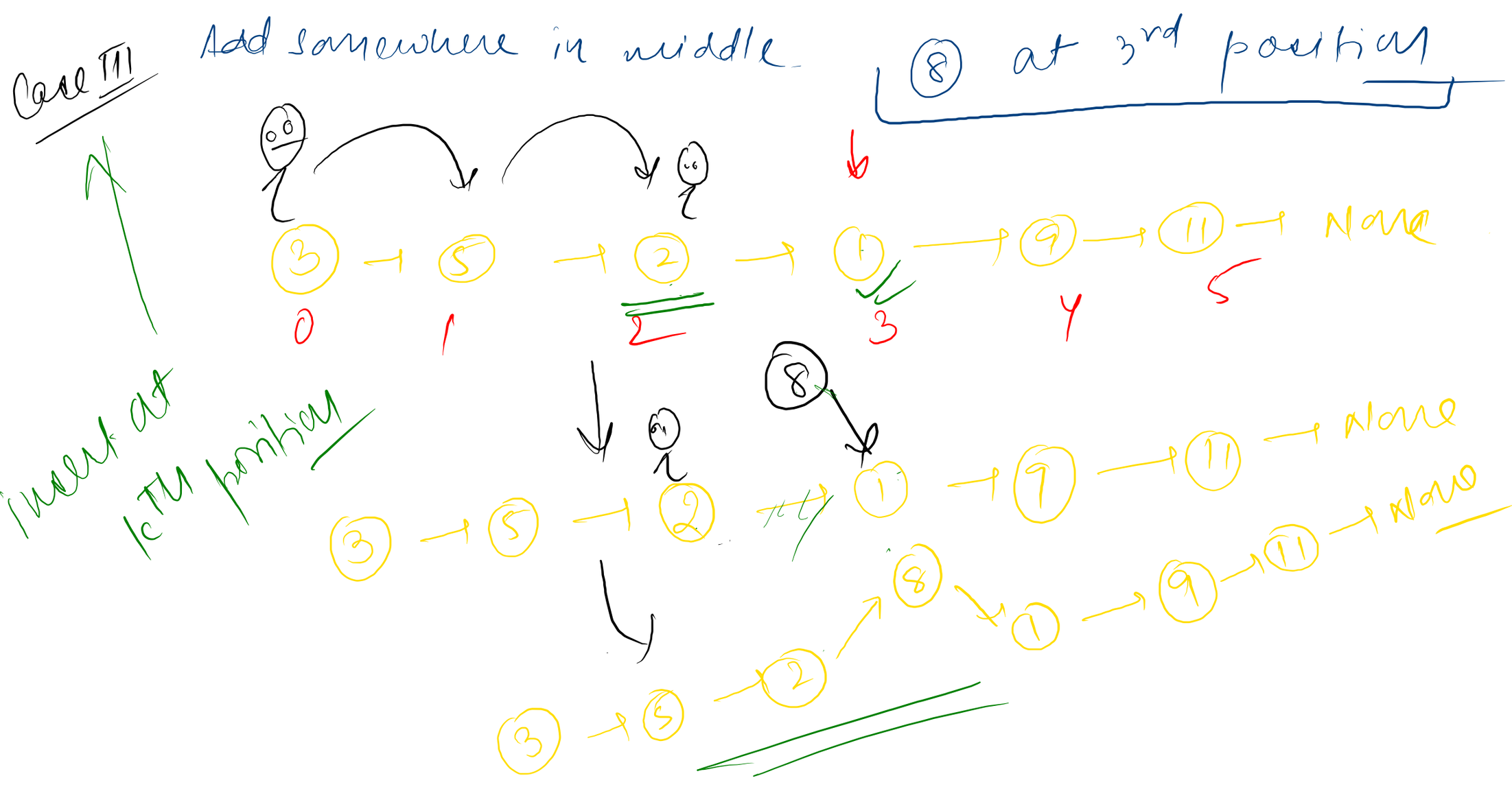


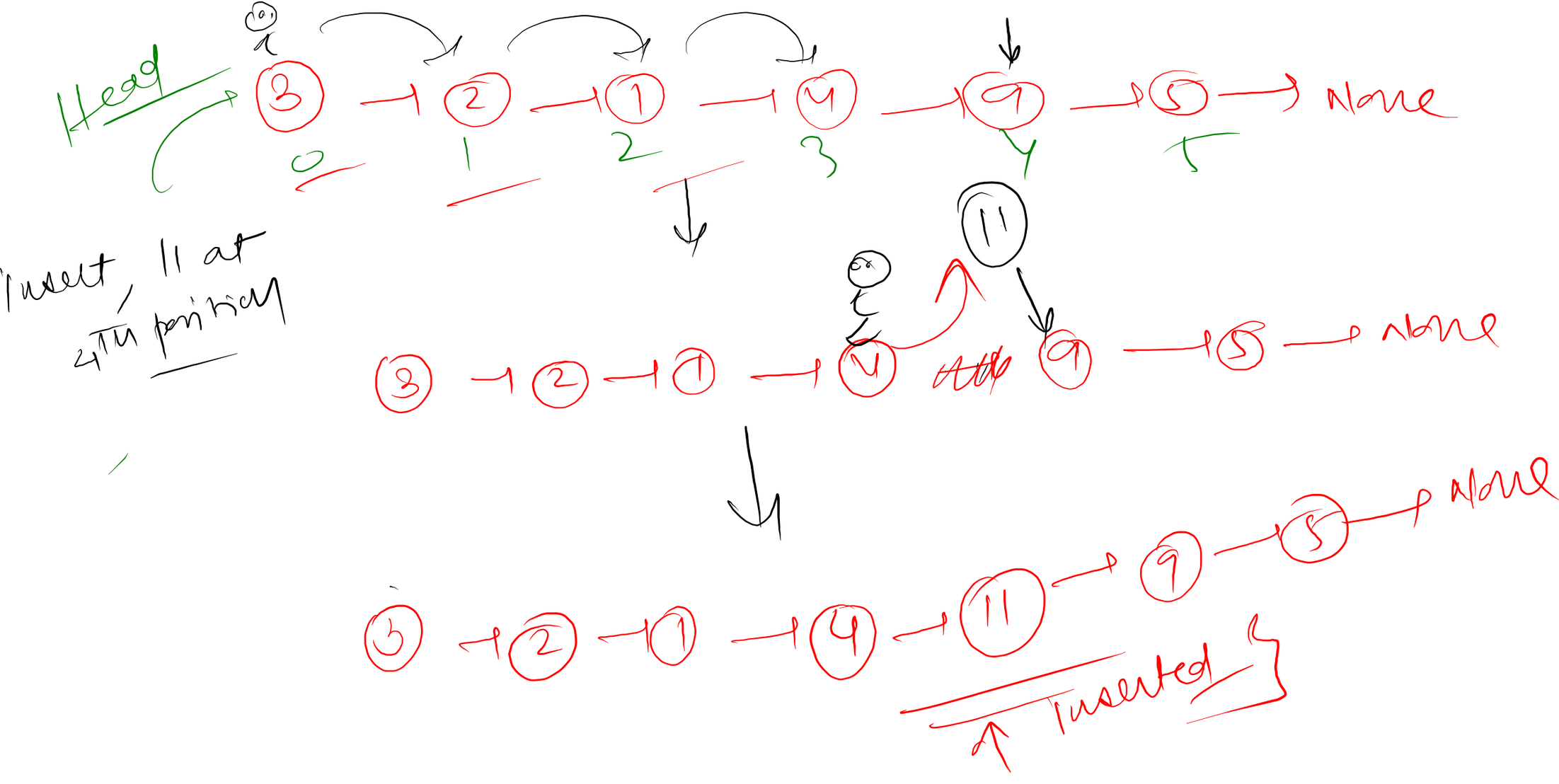
(12) at the end

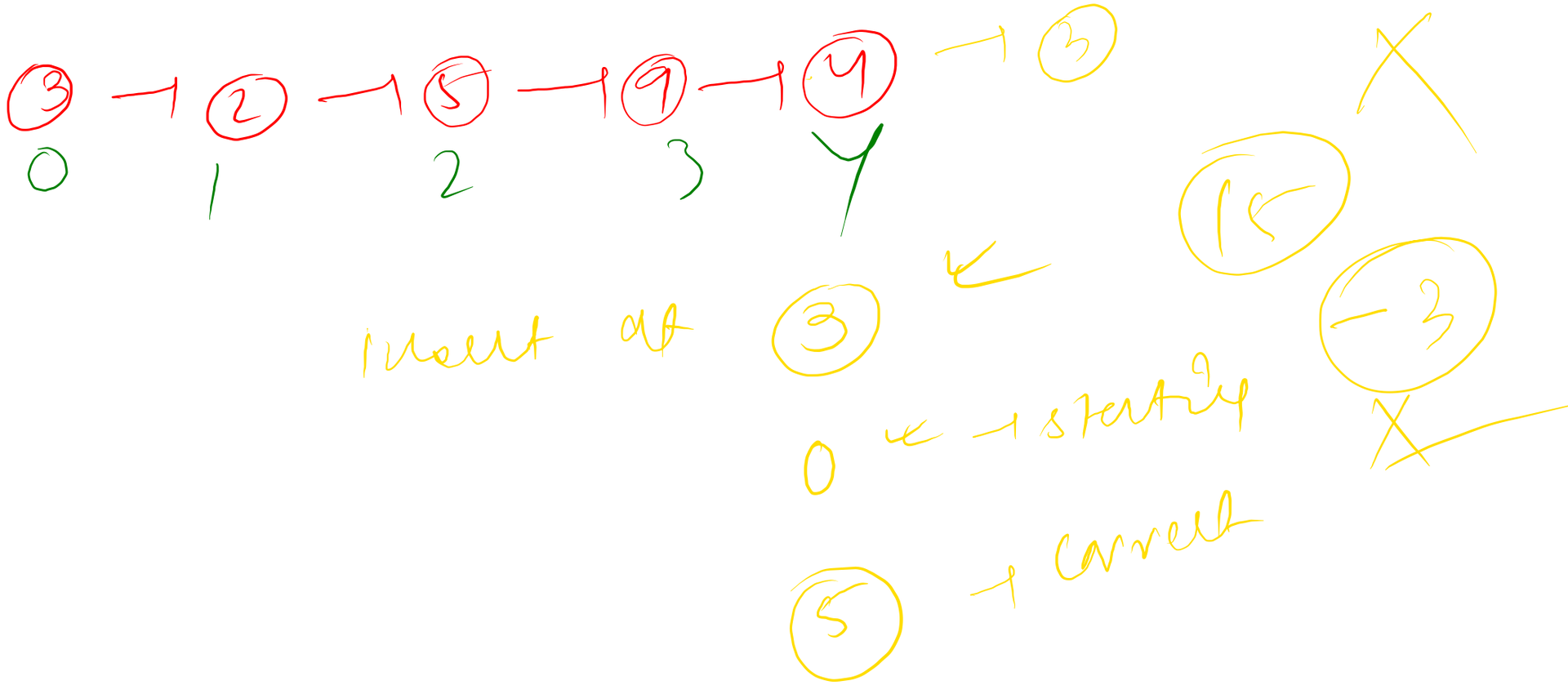
Case III

Add somewhere in middle

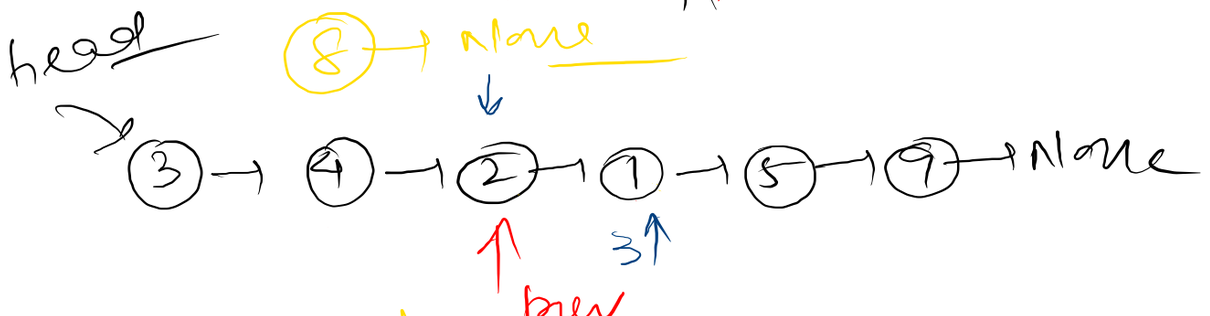
(8) at 3rd position

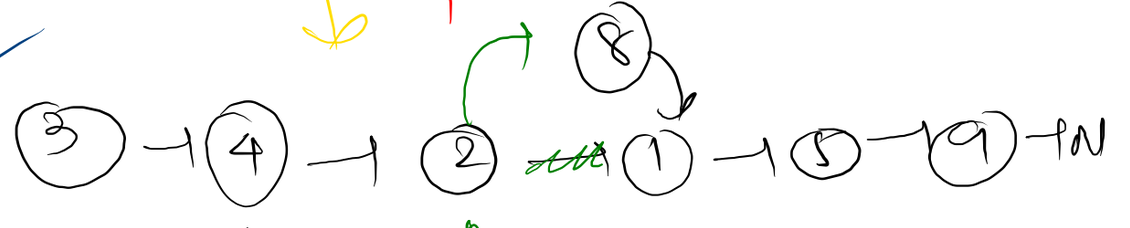


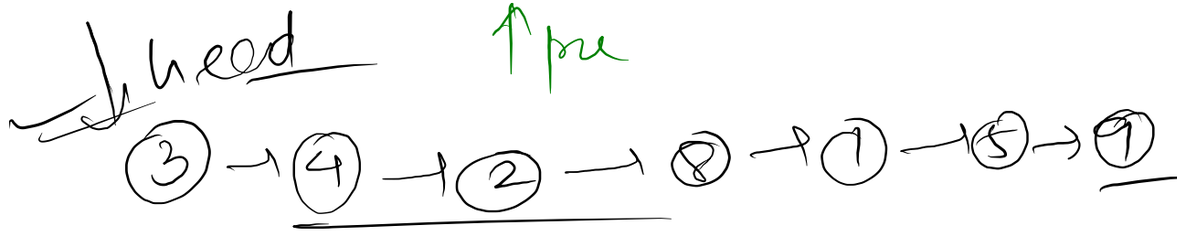

Head
3
2
1
4
9
5
None
0
1
2
3
4
5
Insert 11 at 4TH position
11
3
2
1
4
9
5
None
3
2
1
4
11
9
5
None
Inserted


3
2
5
9
4
0
1
2
3
4
insert of
3
0
5
15
-3
X
X

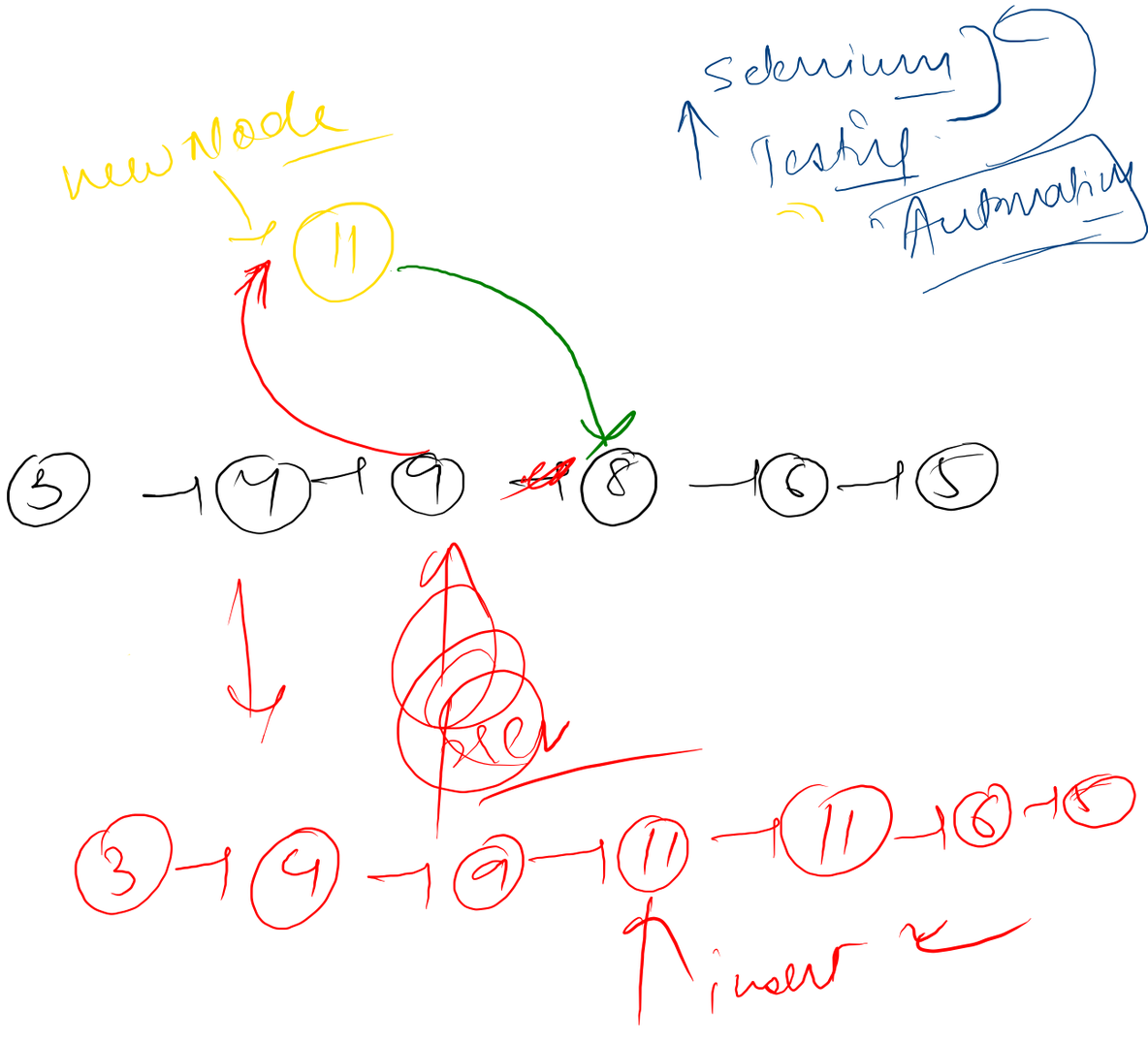
```
def insertNode(head,data,k):
  #If the k is valid
  if (k > length(head) or k <0):
    print("Argument k passed is not valid")
    return head


  #Create new node for data
  newNode = Node(data) #Create the new node object


  if( k==0):
    #We need to insert at the begining
    #We need to update the head
    newNode.setNext(head)
    head = newNode
  else :
    #When not in begining
    # We need to jump to the prev node of the postion
    prev = head
    i=0
    while(i<k-1):
      prev = prev.getNext()
      i+=1


  #prev will be one position left of kth position
  newNode.setNext(prev.getNext())
  prev.setNext(newNode)
  return head
```

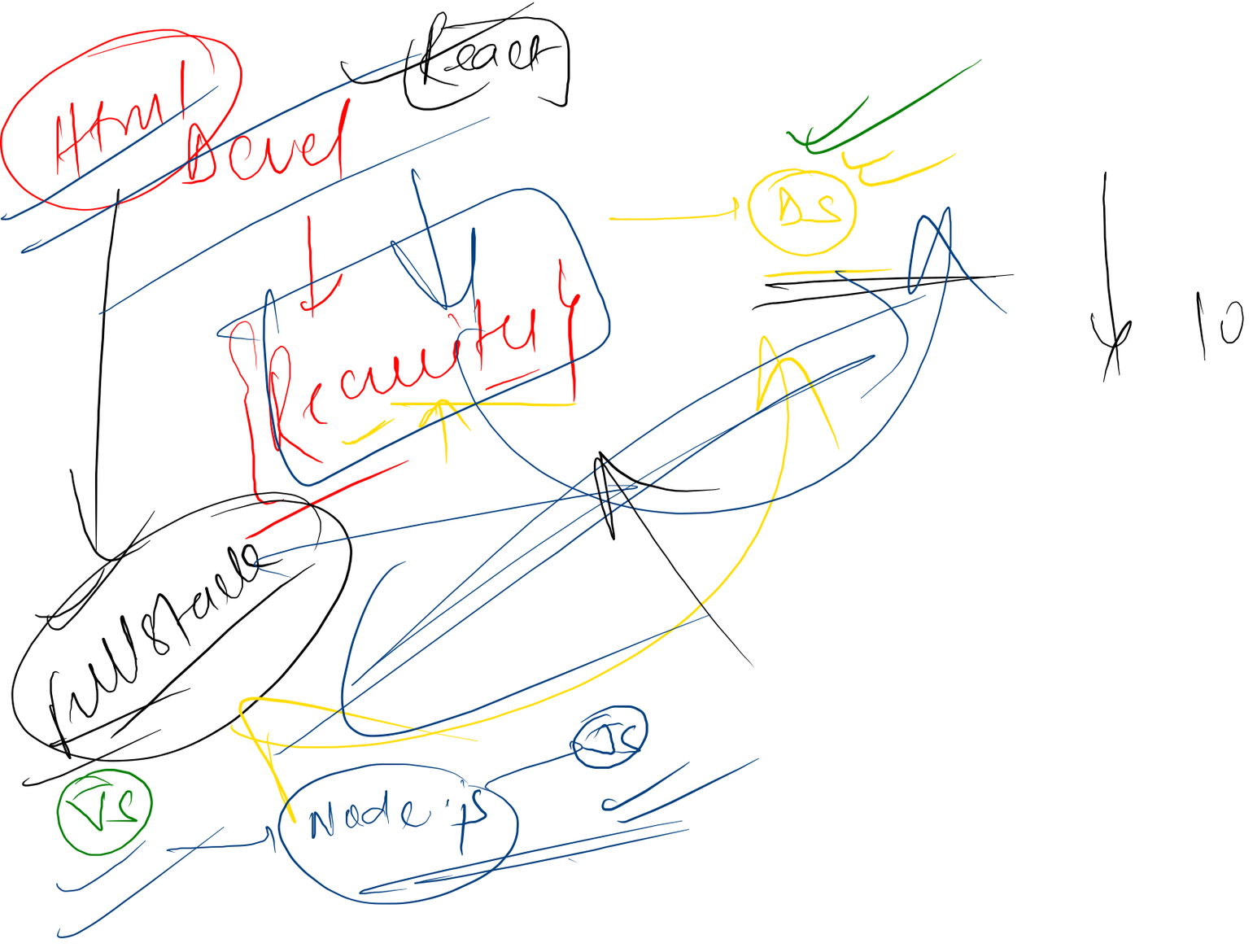


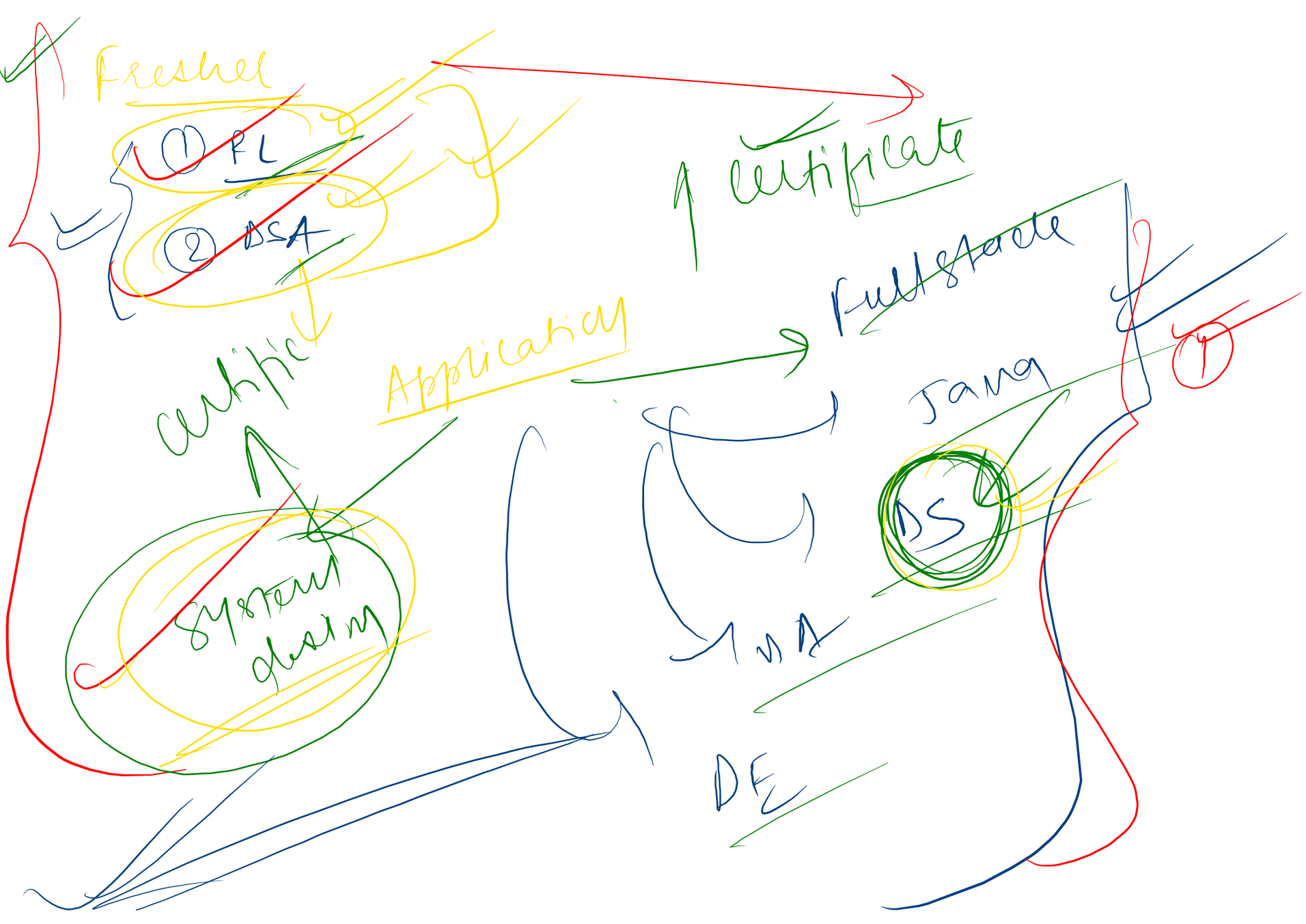


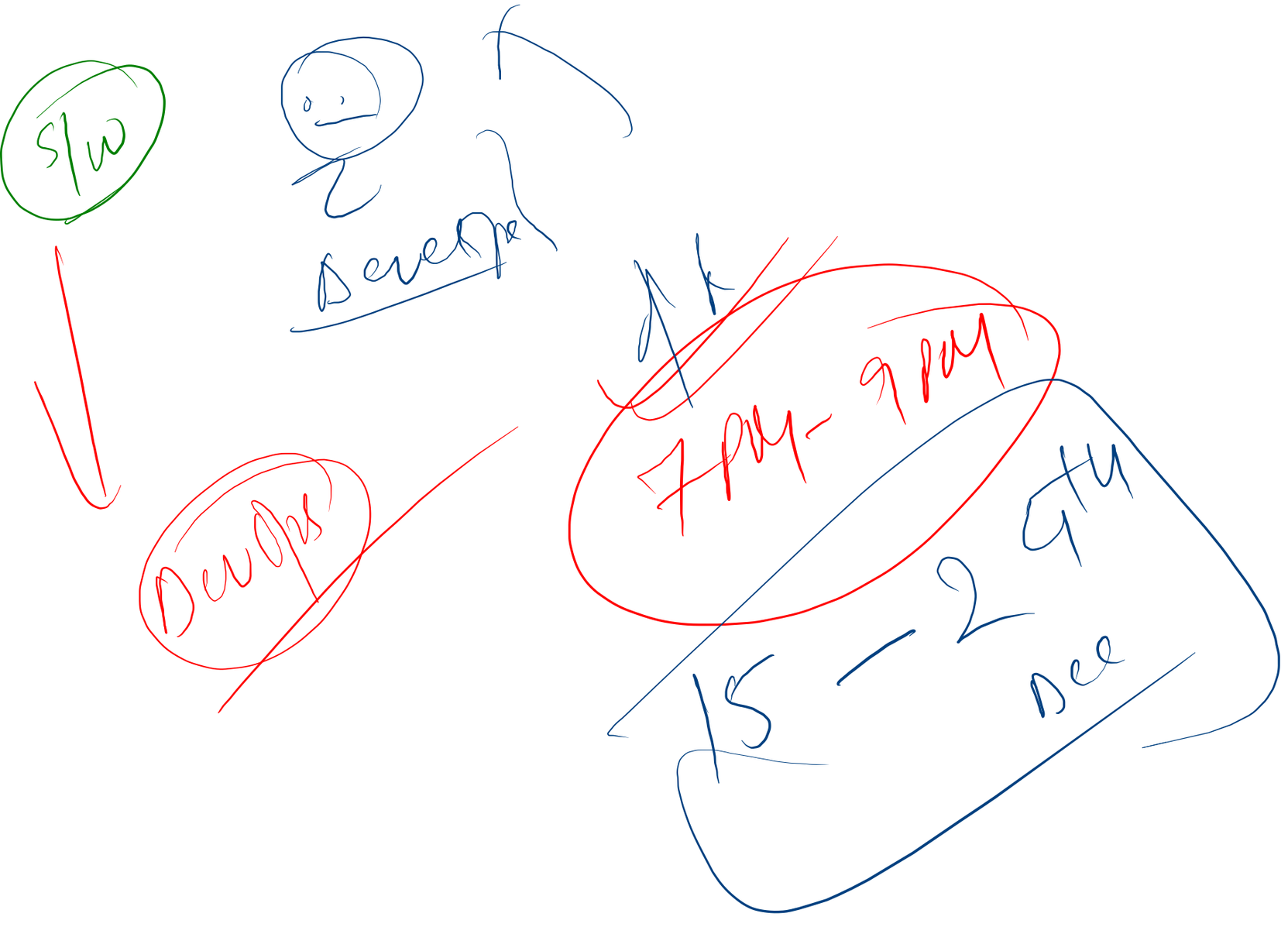


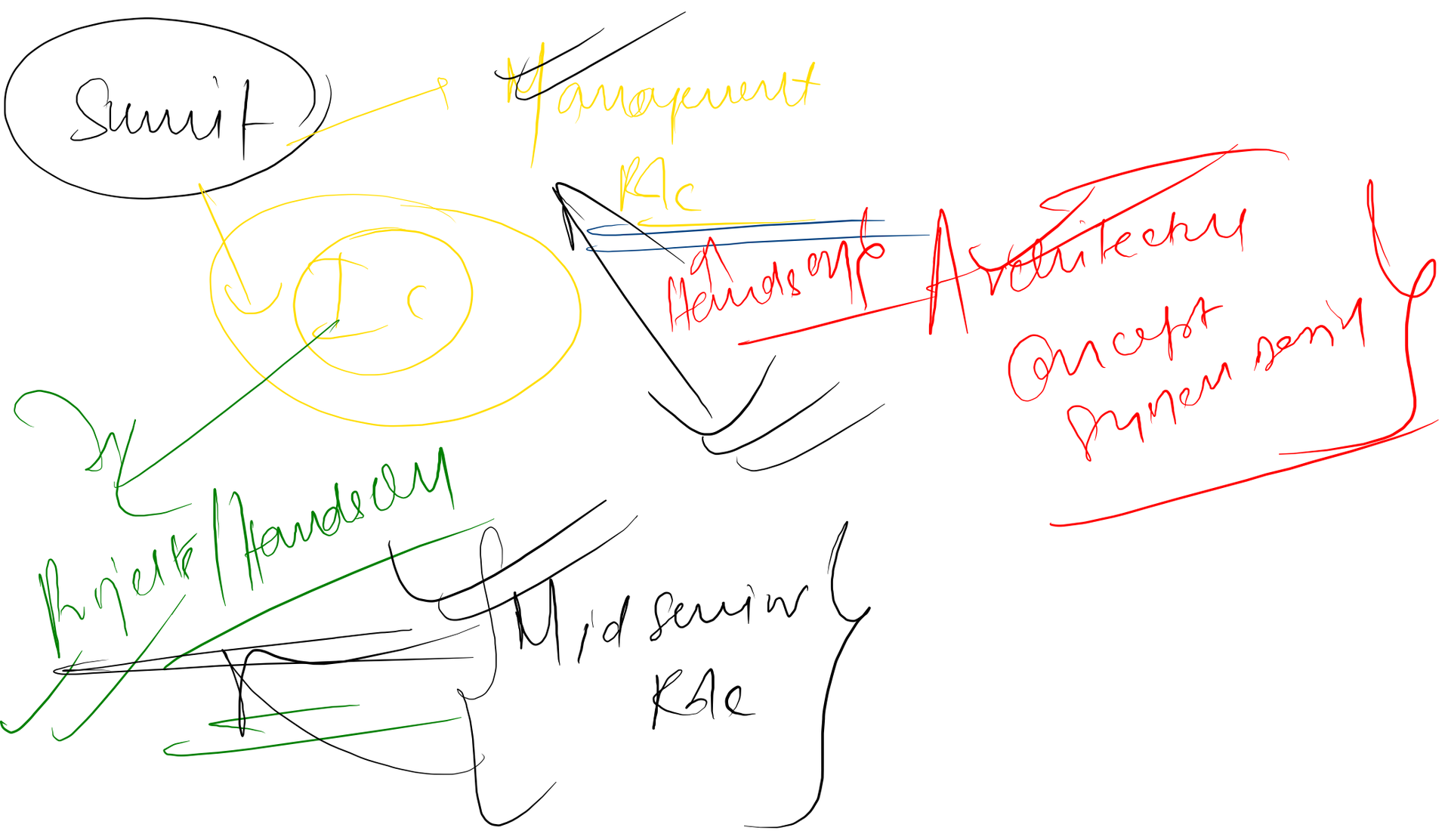
```
def insertNode(head,data,k):
  #If the k is valid
  if (k > length(head) or k <0):
    print("Argument k passed is not valid")
    return head


  #Create new node for data
  newNode = Node(data) #Create the new node object


  if( k==0):
    #We need to insert at the begining
    #We need to update the head
    newNode.setNext(head)
    head = newNode
  else :
    #When not in begining
    # We need to jump to the prev node of the postion
    prev = head
    i=0
    while(i<k-1):
      prev = prev.getNext()
      i+=1


    #prev will be one position left of kth position
    newNode.setNext(prev.getNext())
    prev.setNext(newNode)
  return head
```

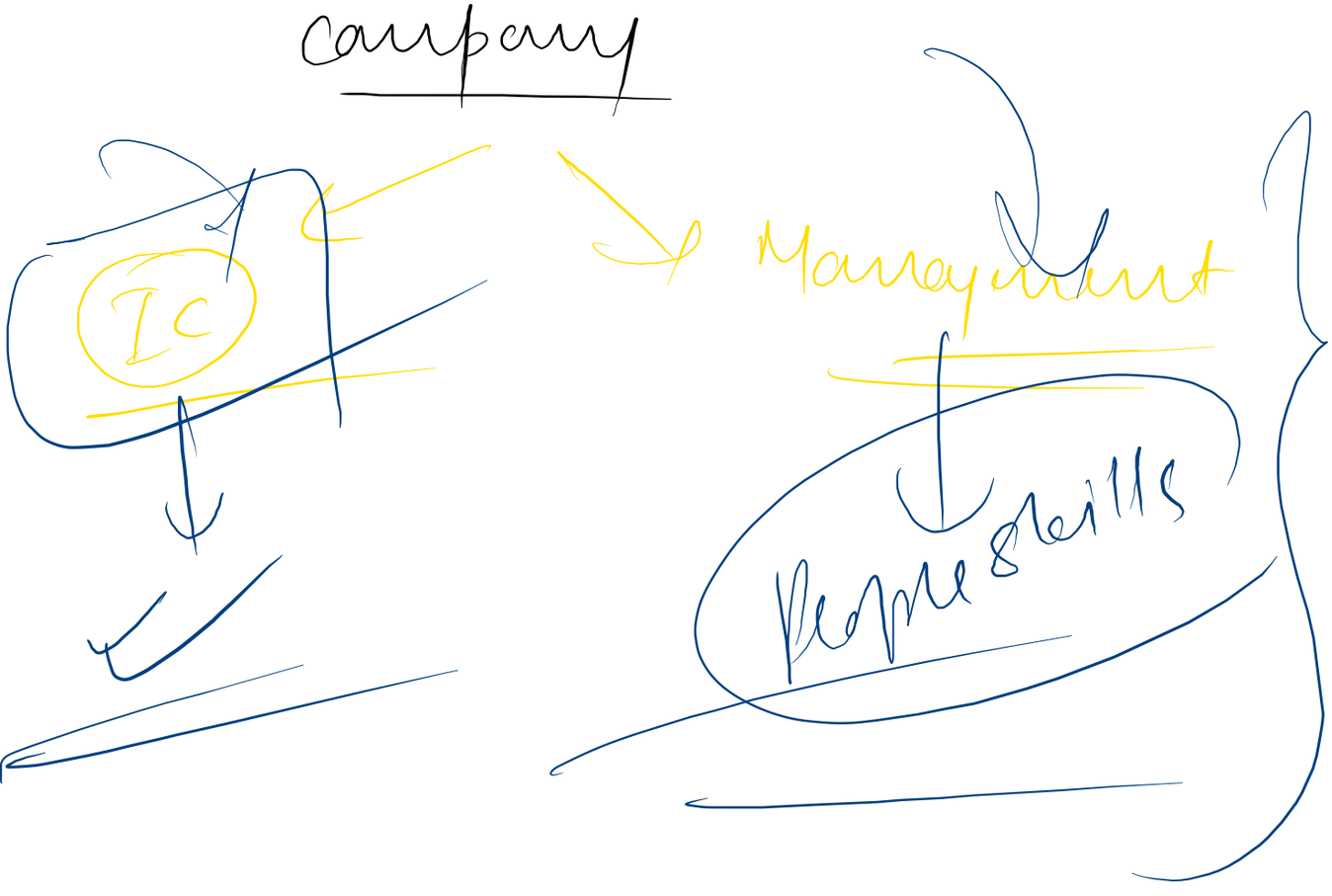



Html
React
Redux
DS
IO
Node.js


Freshel
① FL
② DSA
Application
Fullstack
Java
DS
DSA
DE


S/W
Devops
7PM - 9PM
15 - 2 Dec


Summit
Management
RAC
Architecture
Concept
Project
Project / Handson
Mid senior
RAC


Company
IC
Management
People skills

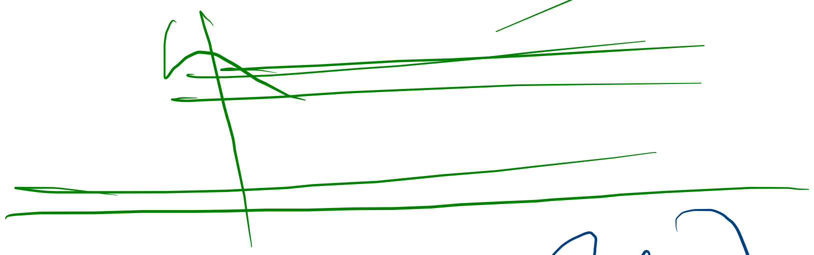
FAANG

Management

Strong in concepts

Track Record

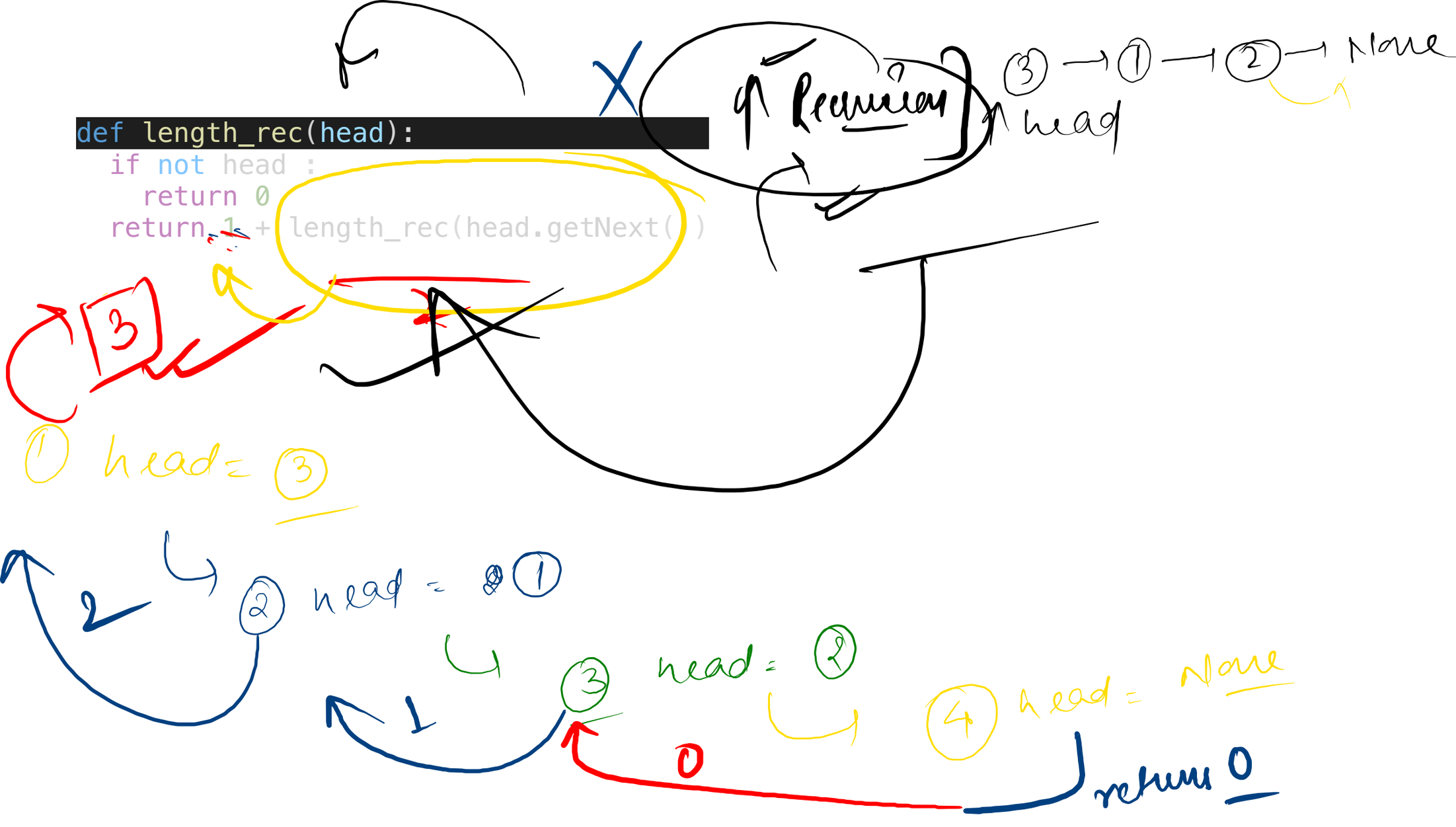
good handson

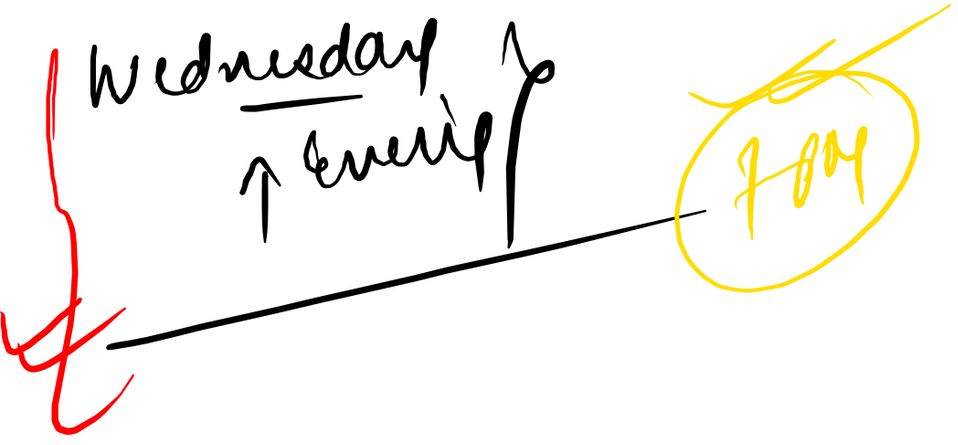


Univ

LMS portal

+ Kube

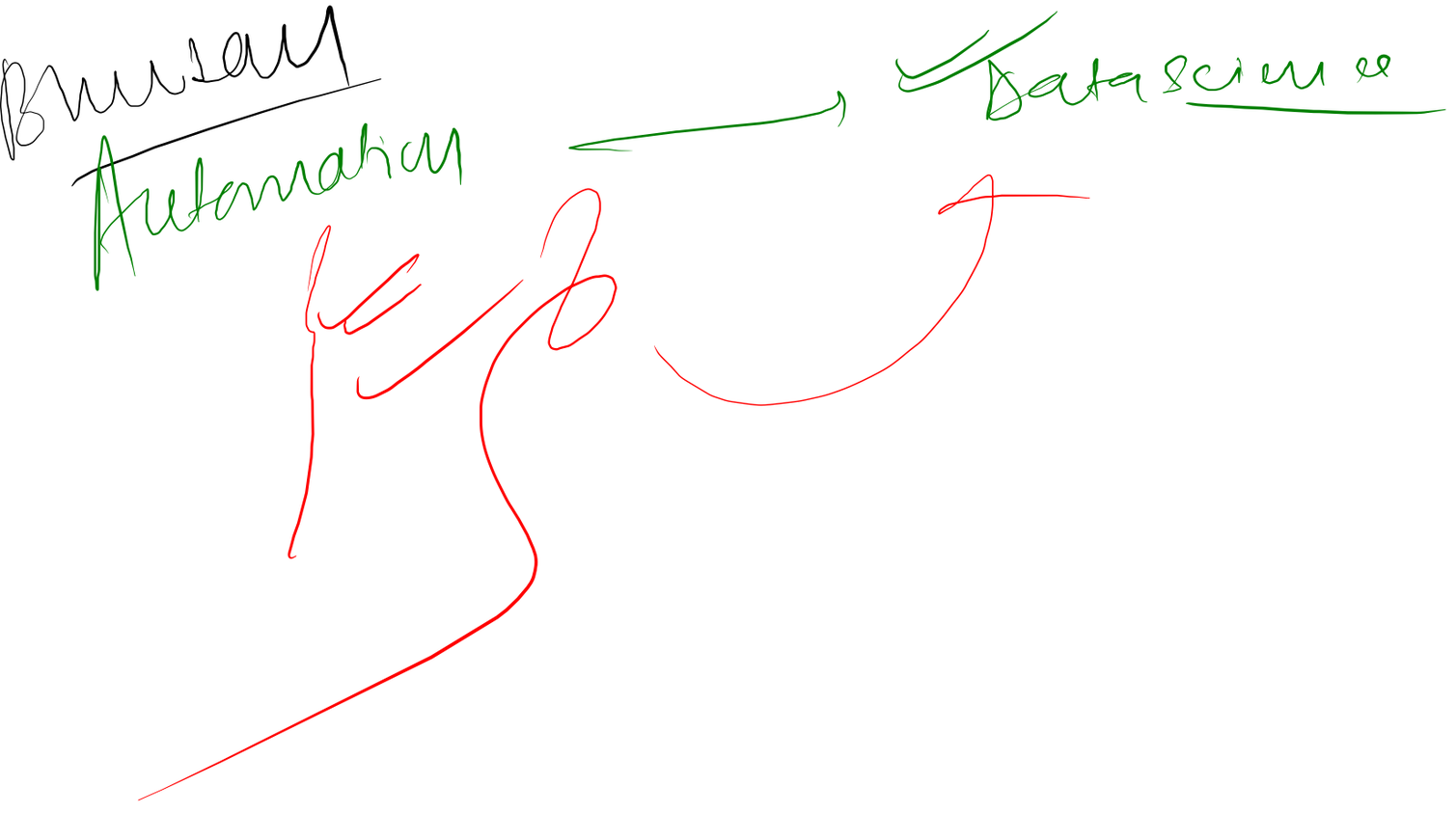

def length_rec(head):
if not head :
return 0
return 1 + length_rec(head.getNext())
Recursion
head
3 → 1 → 2 → None
3
head = 3
2
head = 1
1
head = 2
0
head = None
return 0

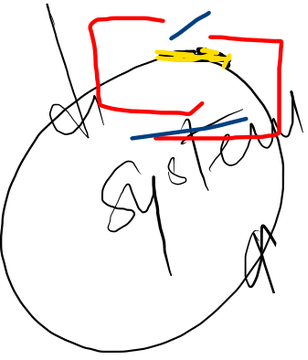

Wednesday
every
7pm

Monday } Video } → links

PDF

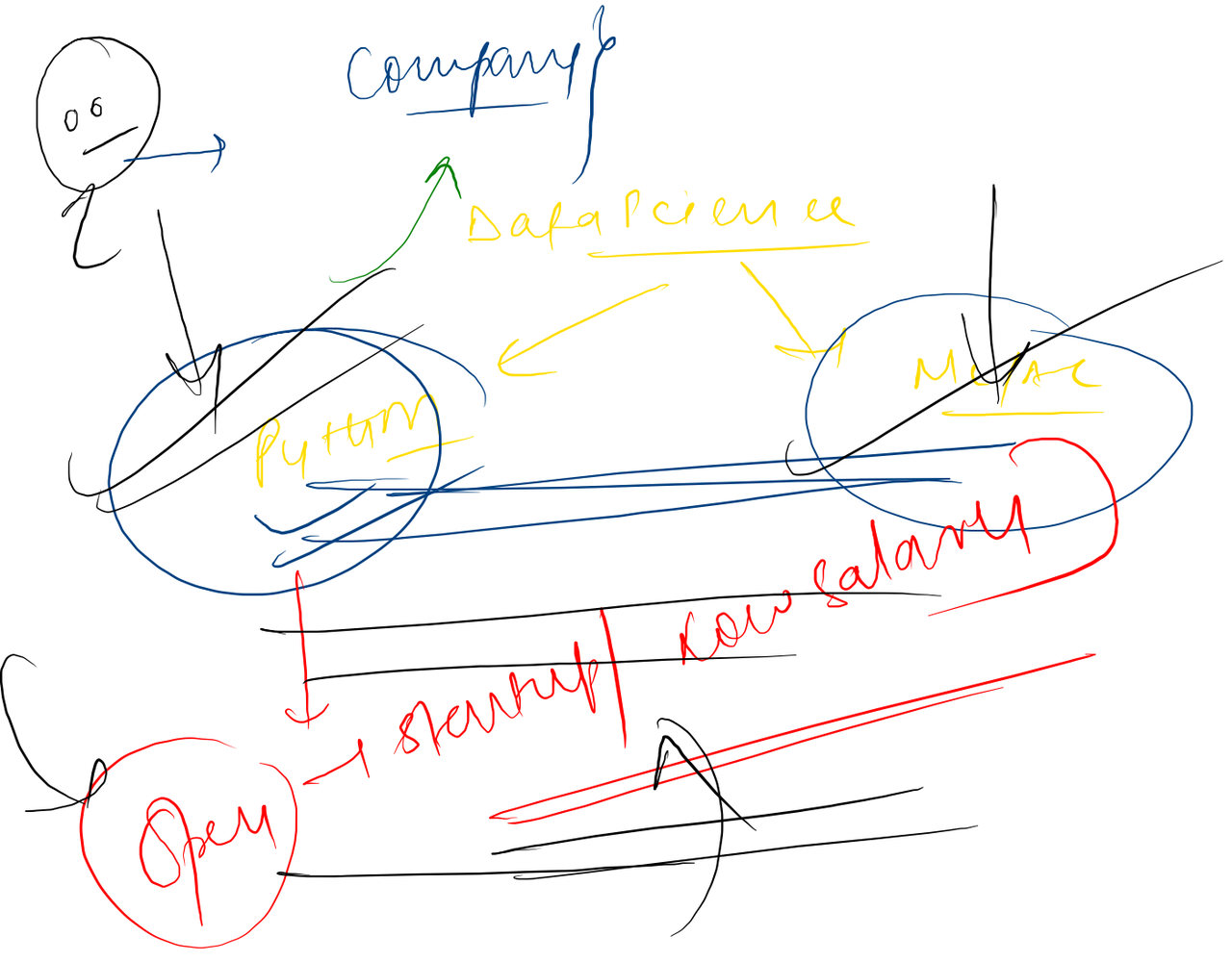
Assignments }

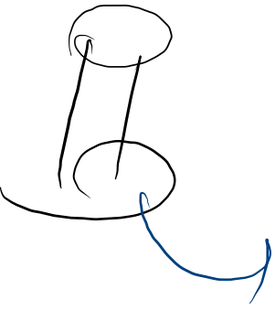

Business
Automation
Data science


System

BE

Electrical

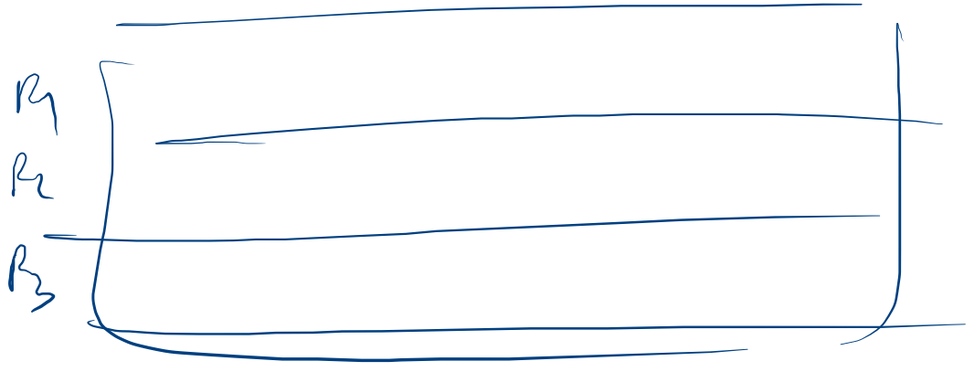

Company
Data Science
Python
MySQL
Query
Statistics
low salary



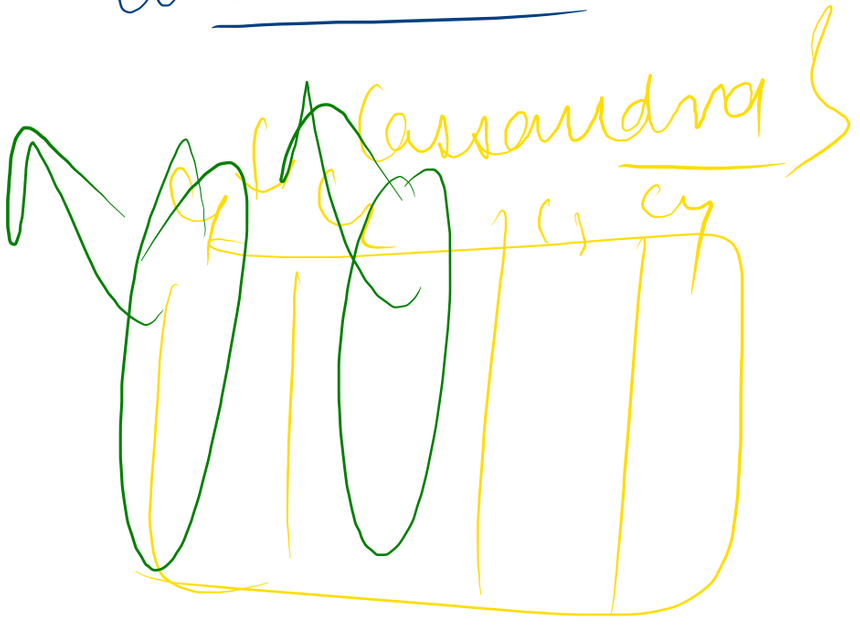
Mercury

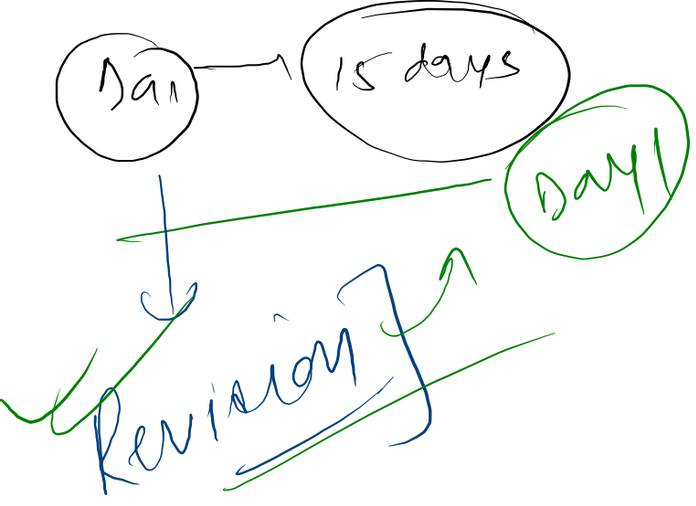
Rawbased DB

↳ Mysql / oracle



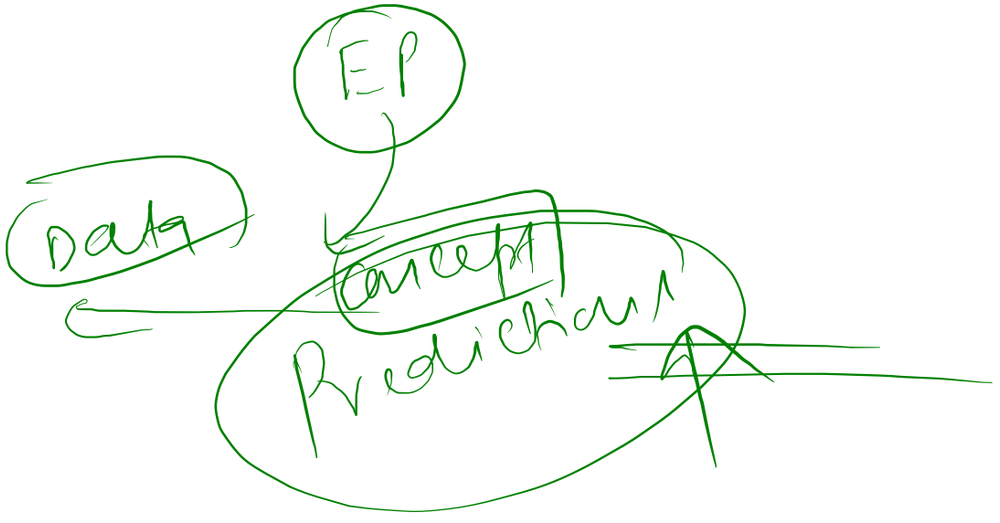
Columnar

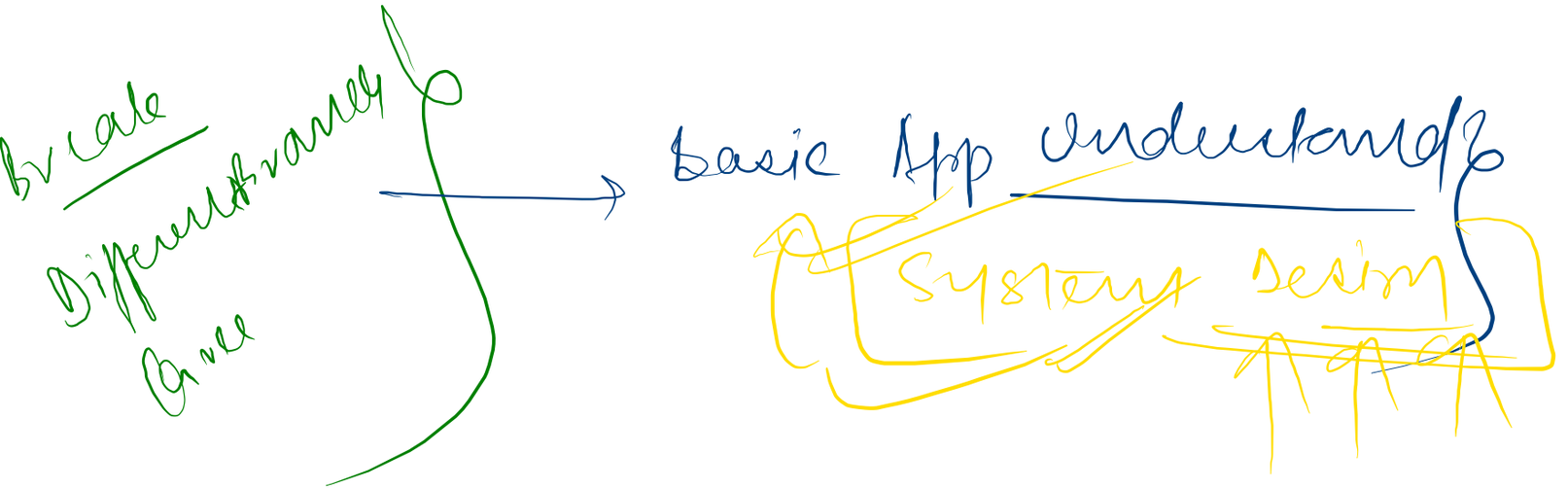


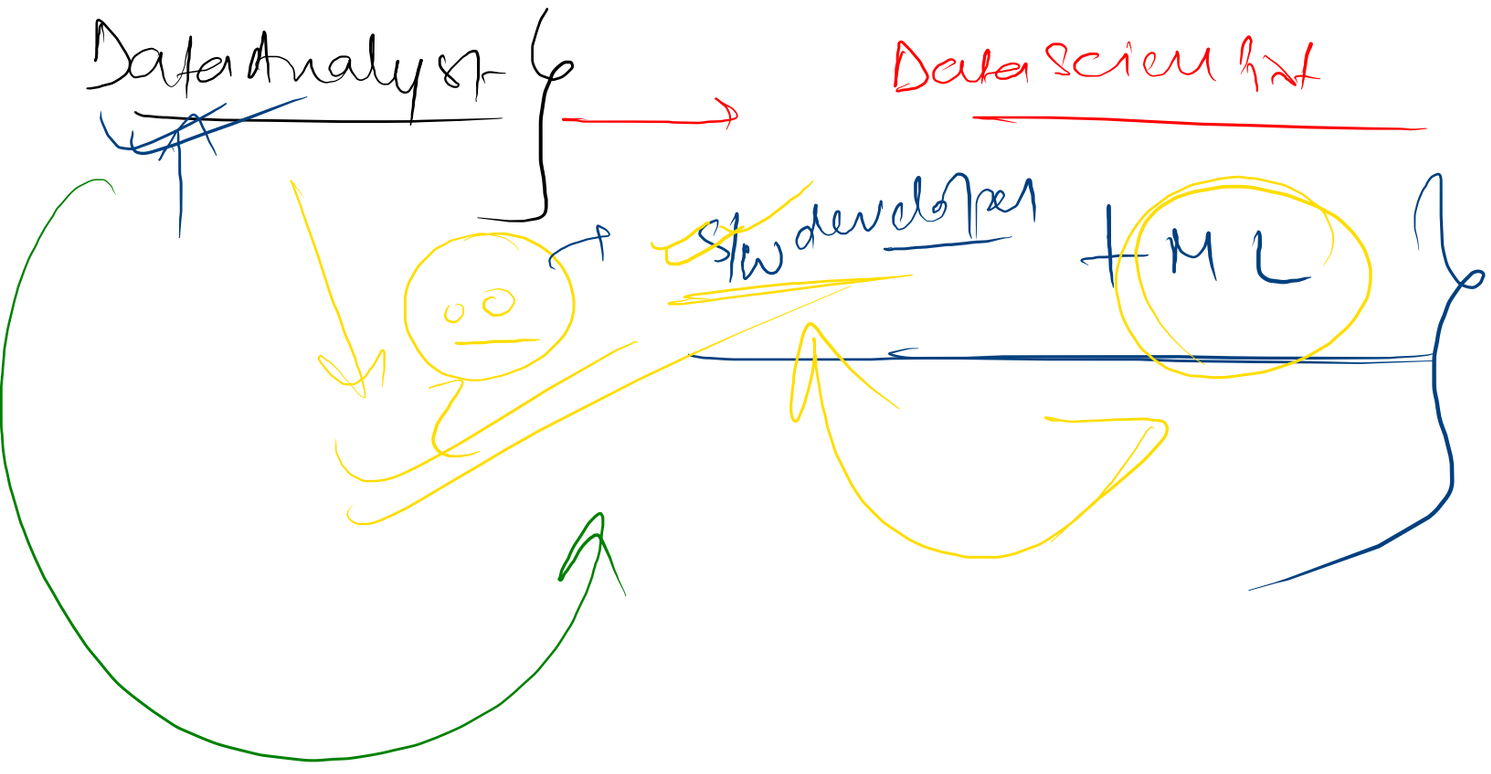

Day
15 days
Day1
Revision

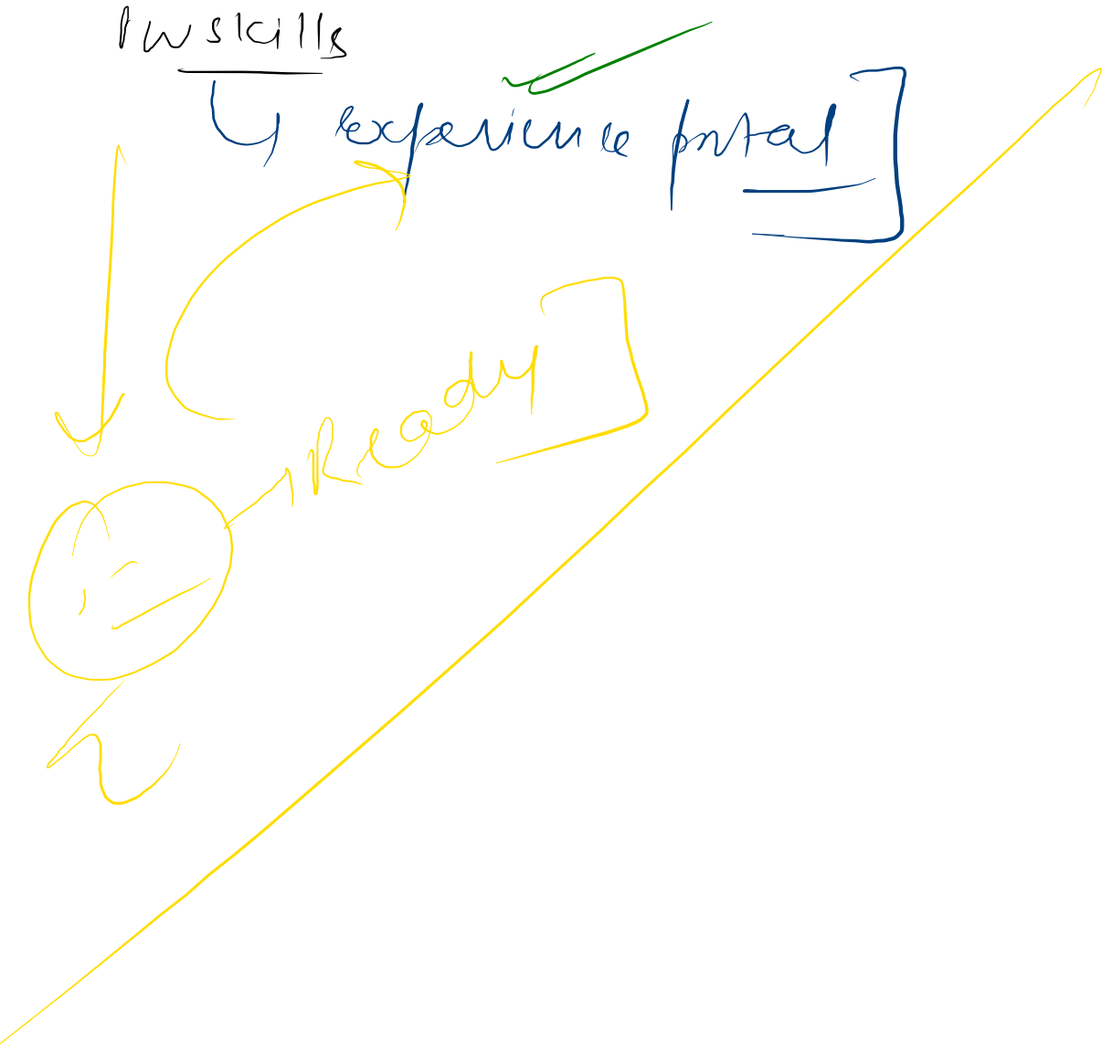
UG → 2nd year

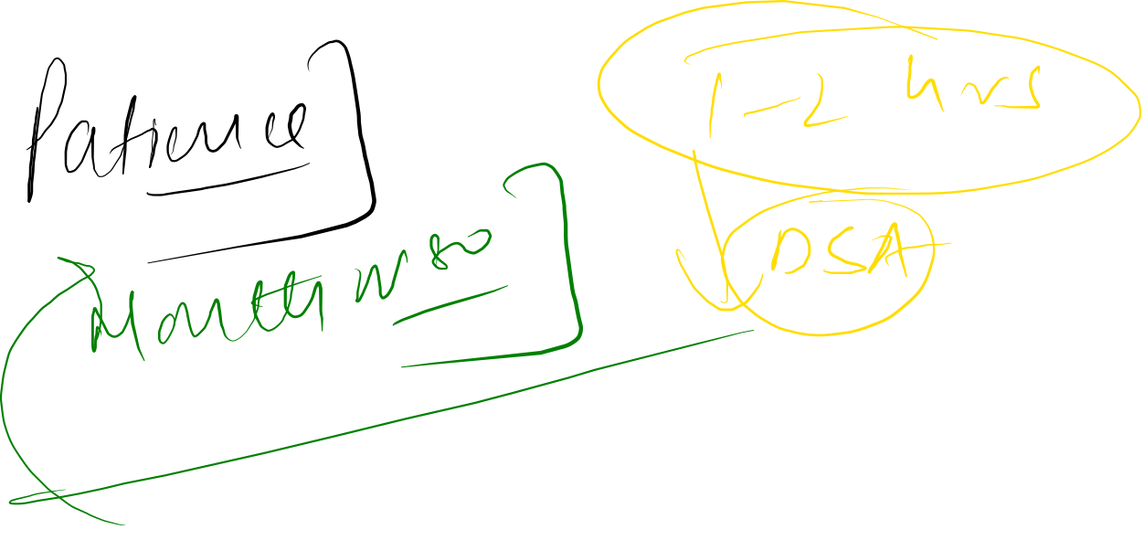
ML

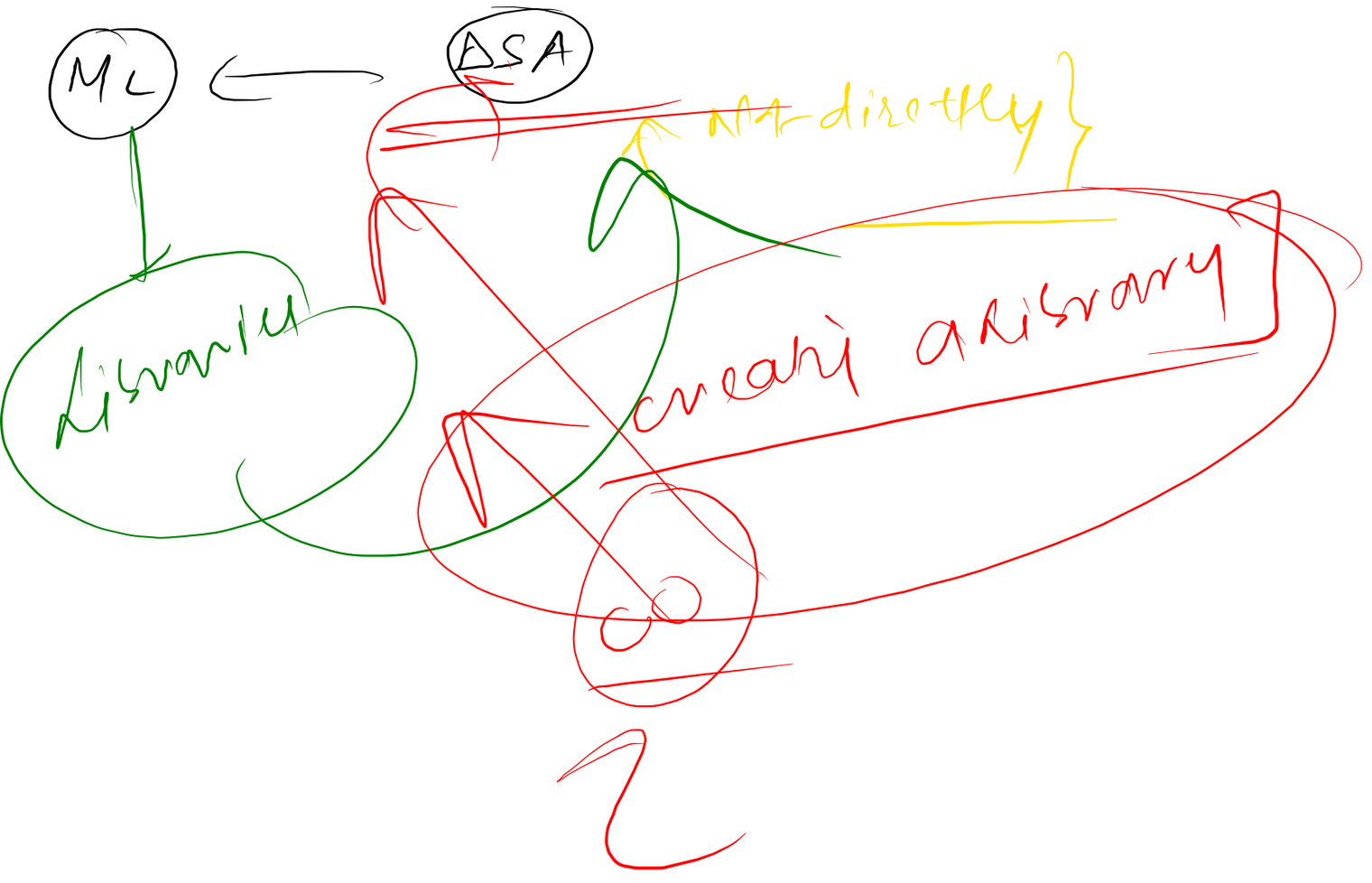


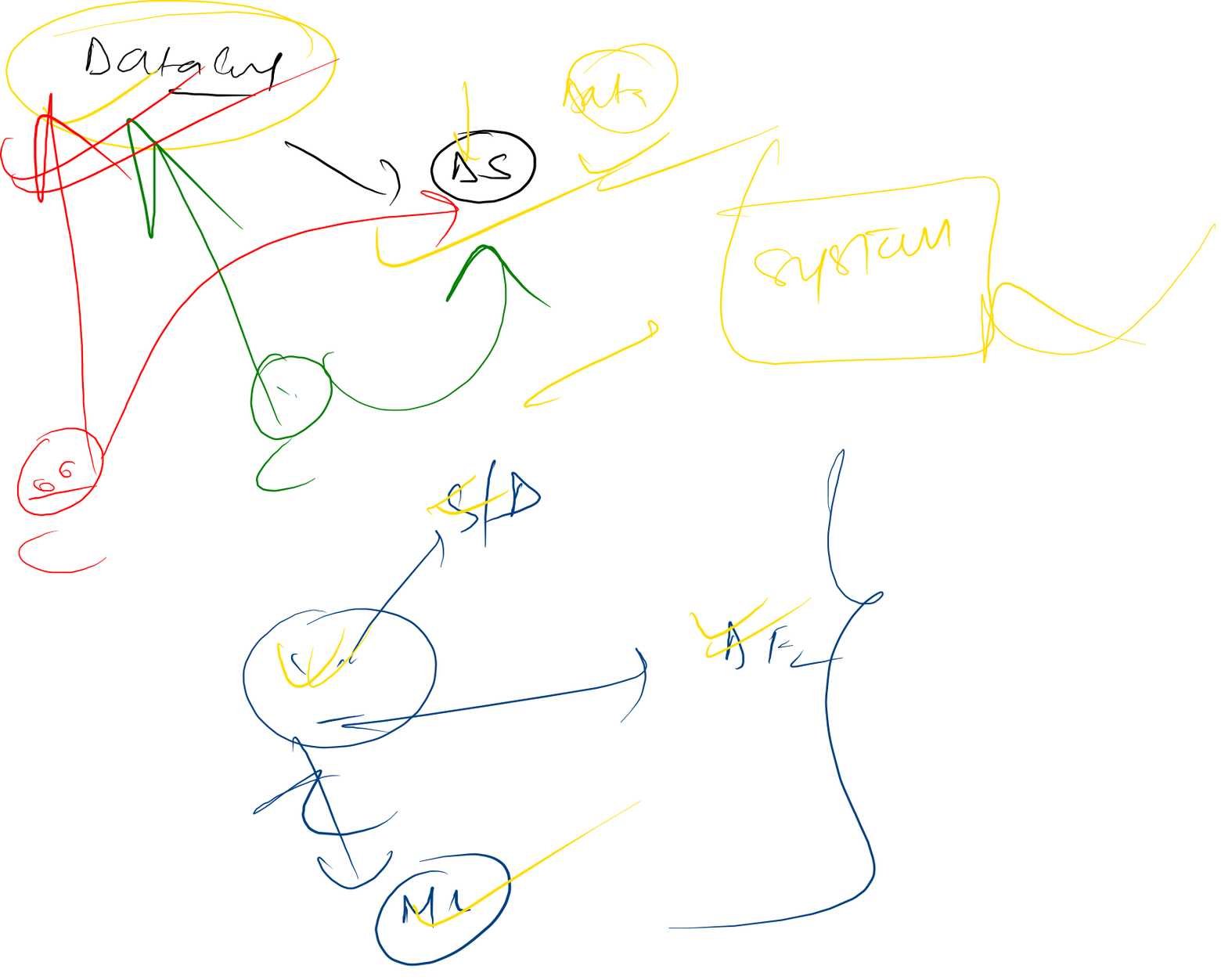

Scale
Differentiate
Grow
basic App understanding
System Design

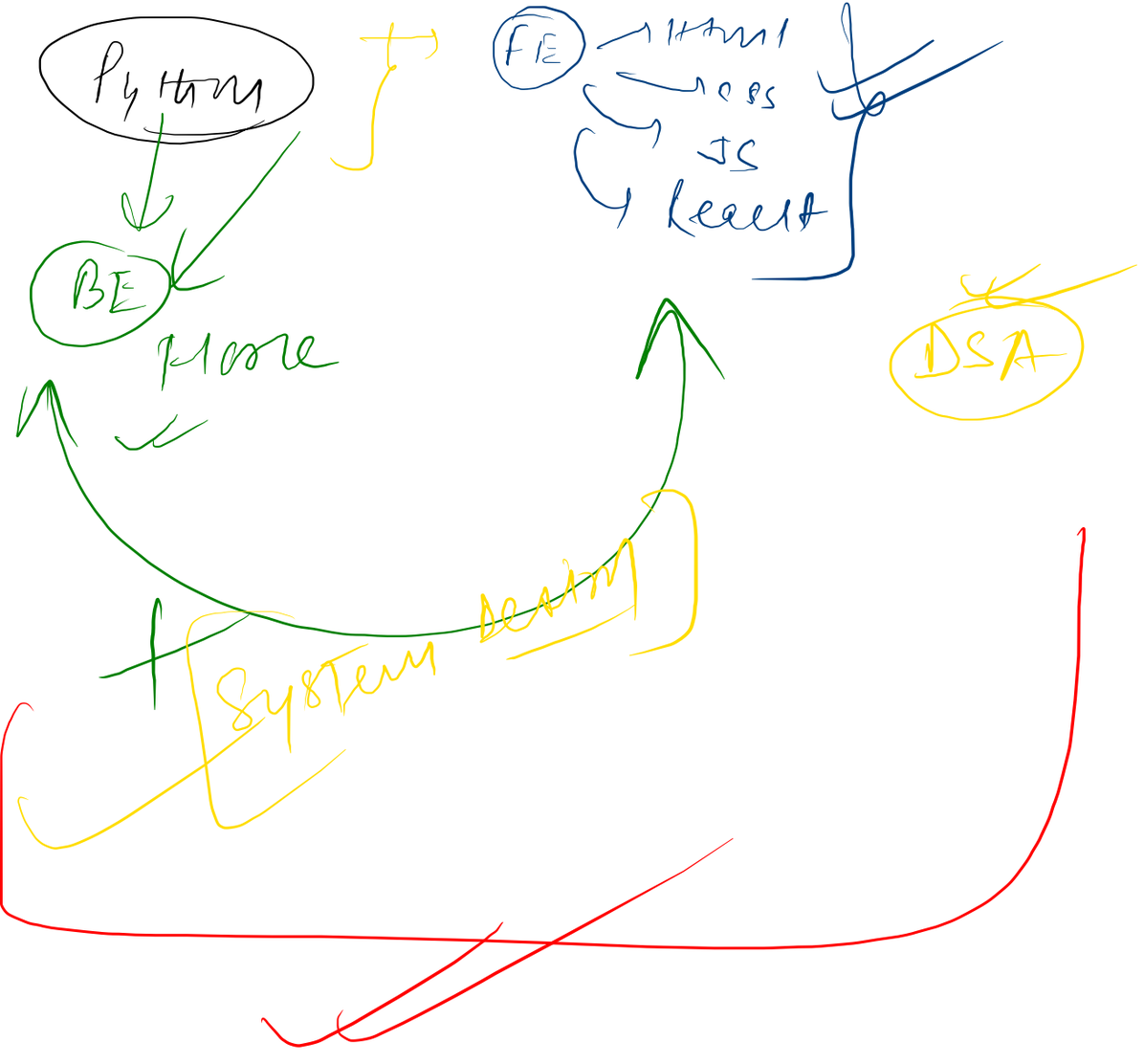

Data Analyst
Data Scientist
s/w developer
+ ML

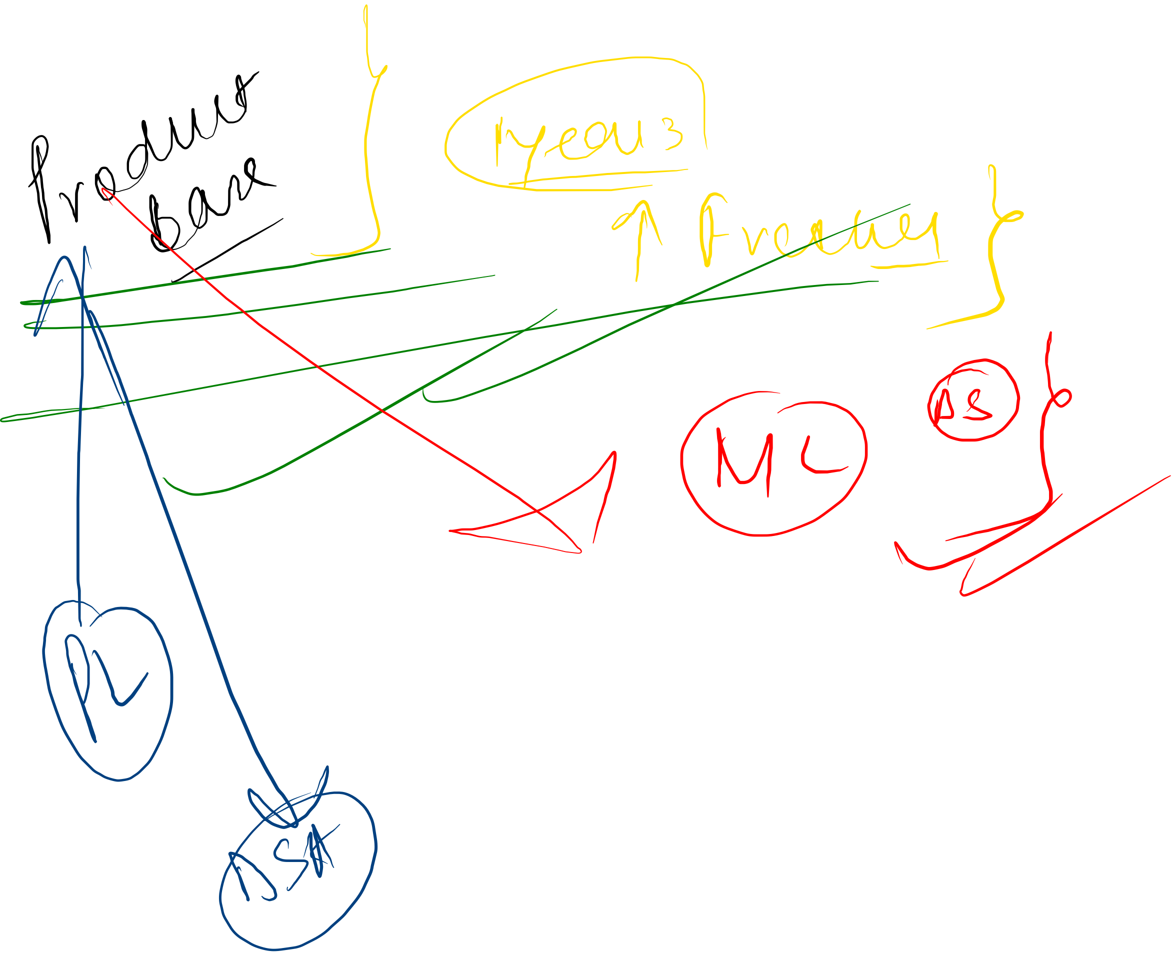

Pw skills
experience portal
Ready

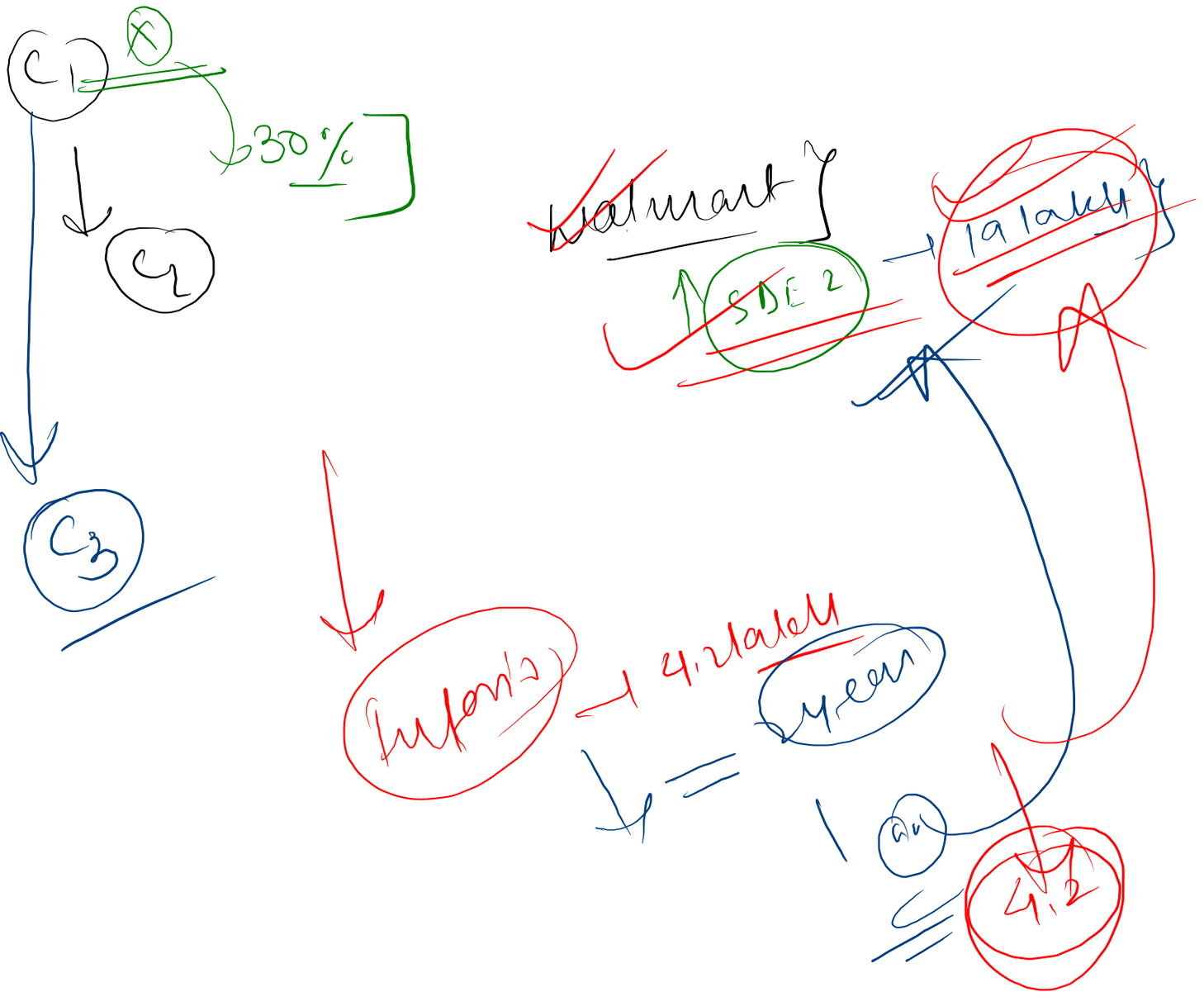

Patience
Monitoring
1-2 hrs
DSA

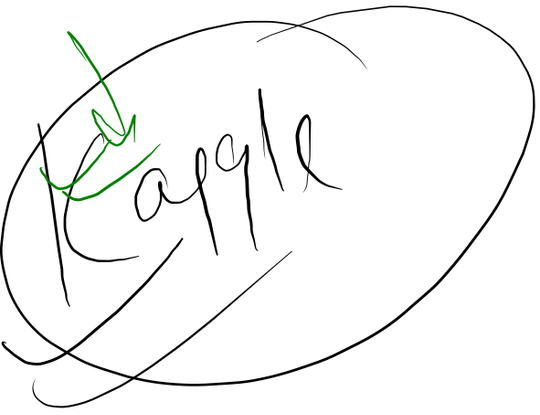

ML
DSA
directly
Libraries
create arrays

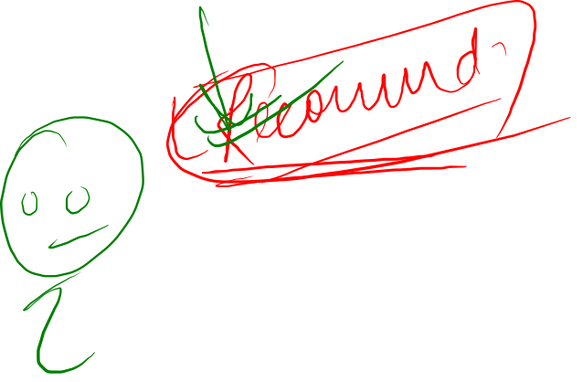

Dataset
System

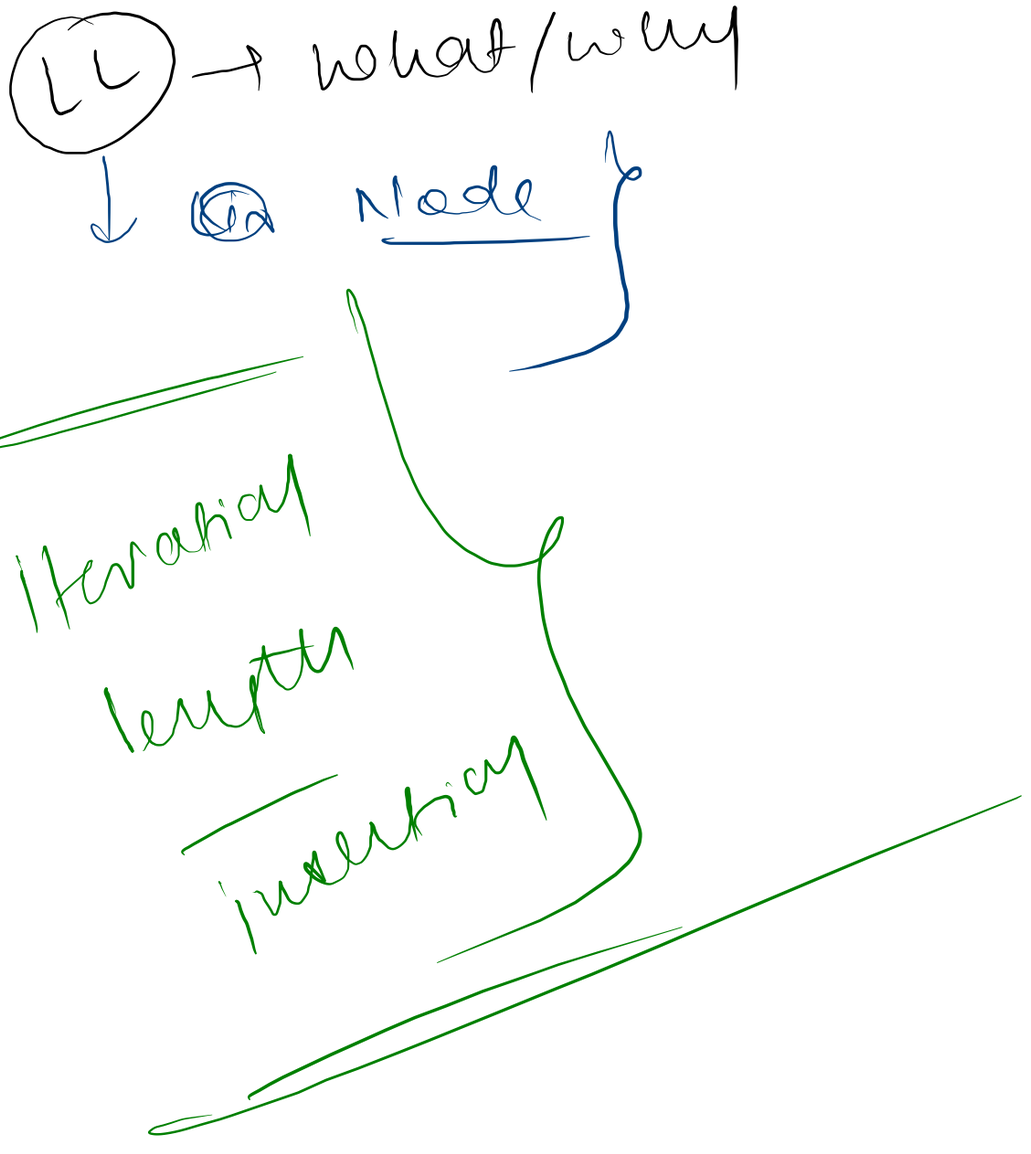

Python
FE
HTML
CSS
JS
React
BE
more
DSA
System Design

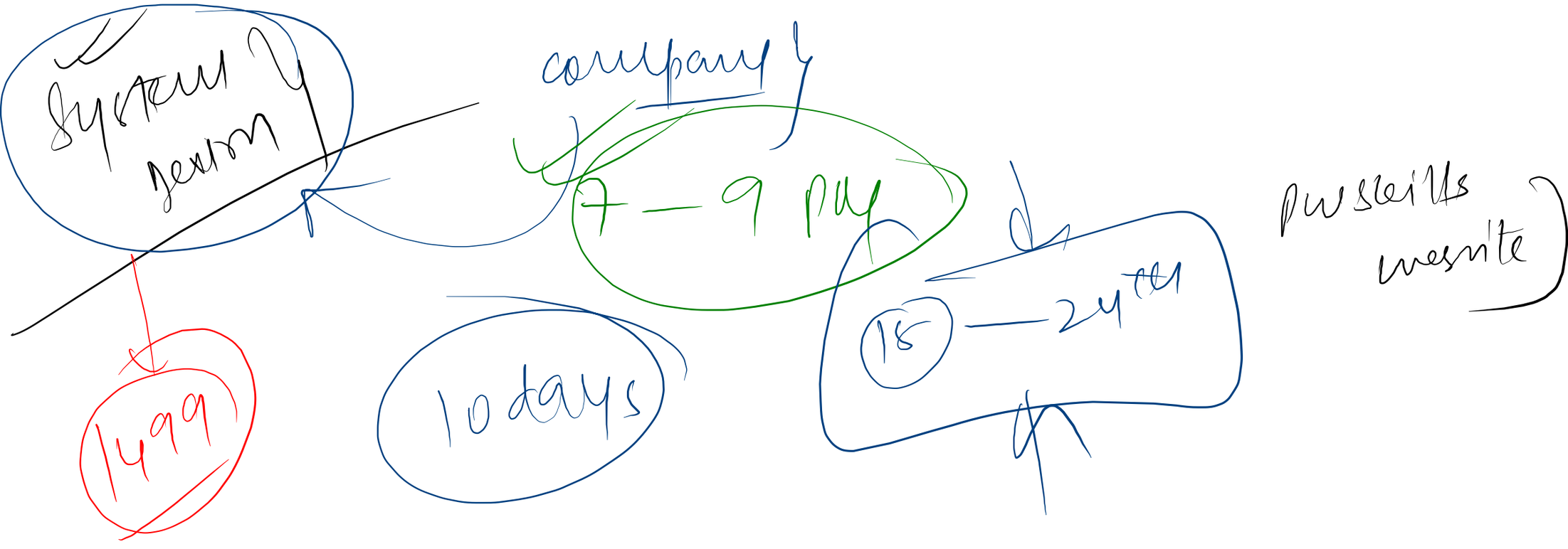

ML
DS
DL
DSA


C1
X
30%
C2
C3
Walmart
SDE 2
19 lakh
Infosys
4.2 lakh
4 yrs
AI
4.2


Kaggle

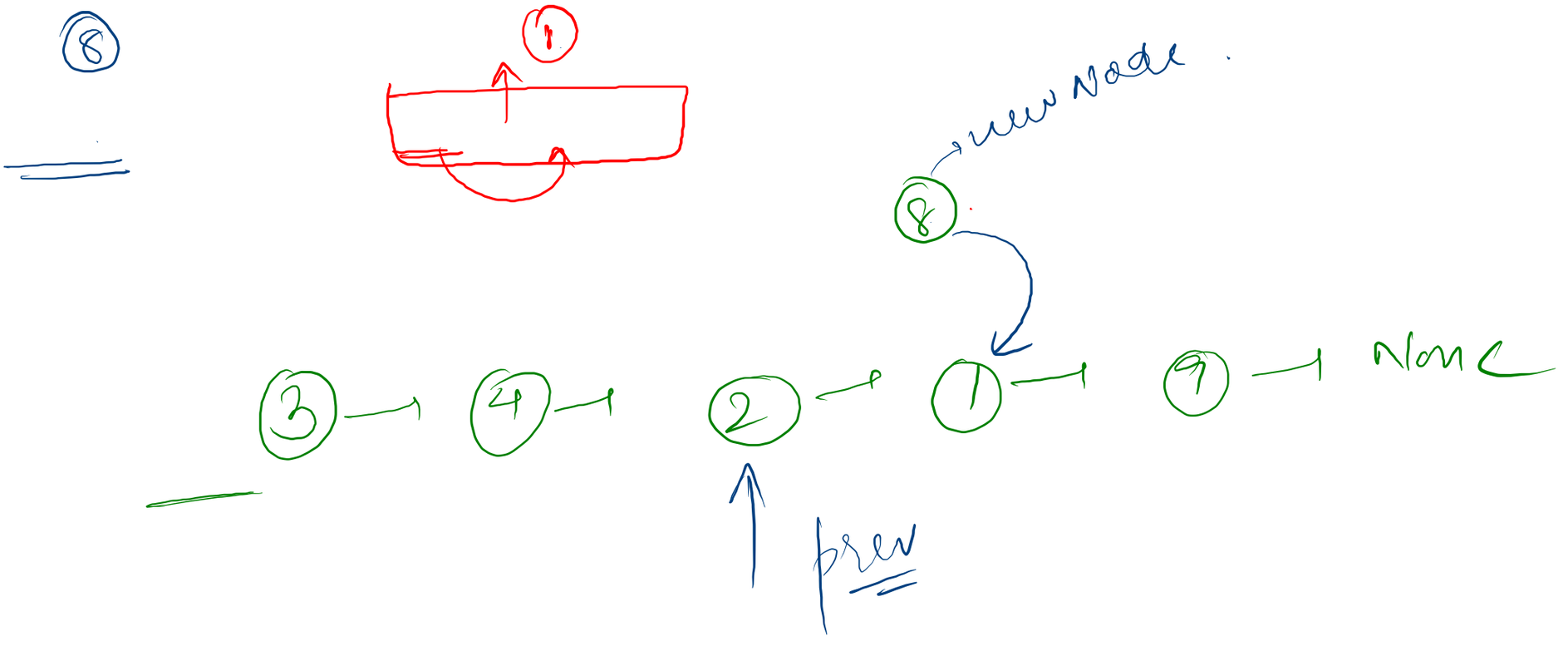

Round

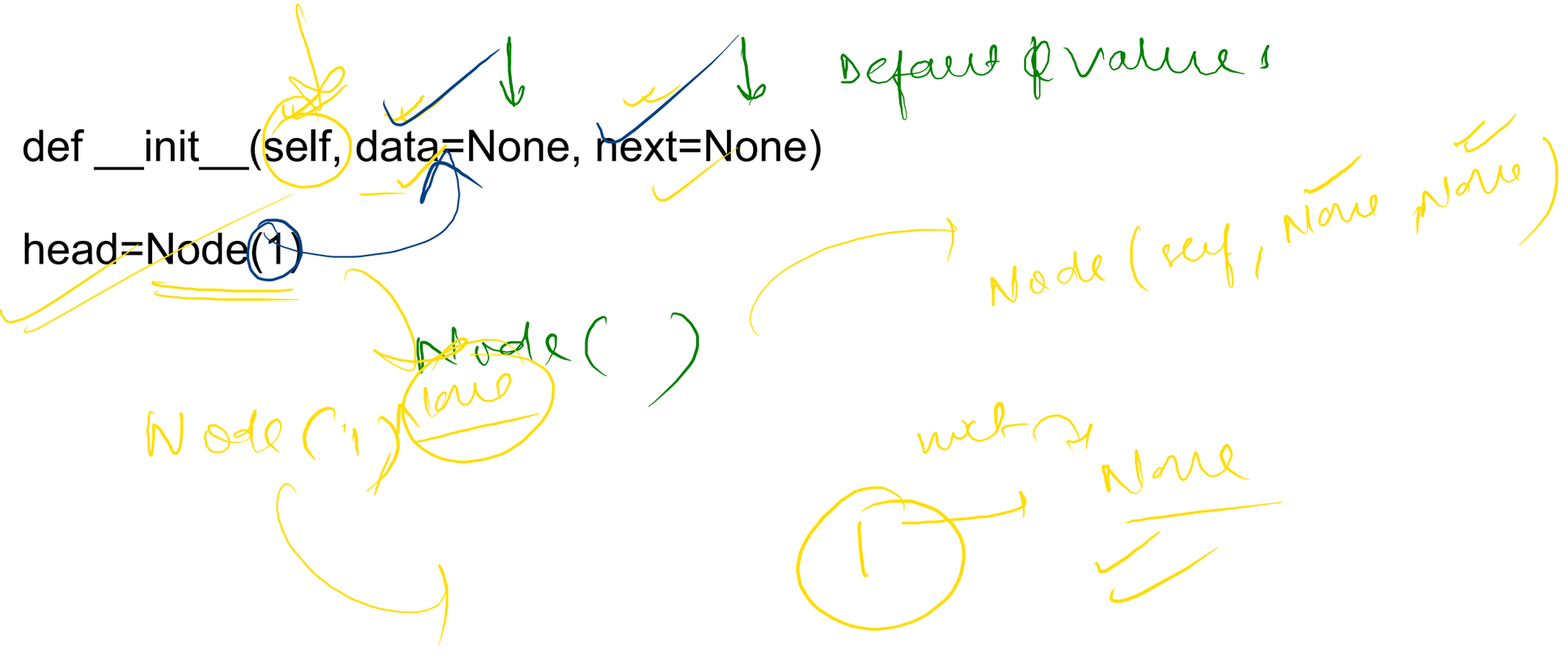

LL
what/why
Node
Iterative
length
insertion

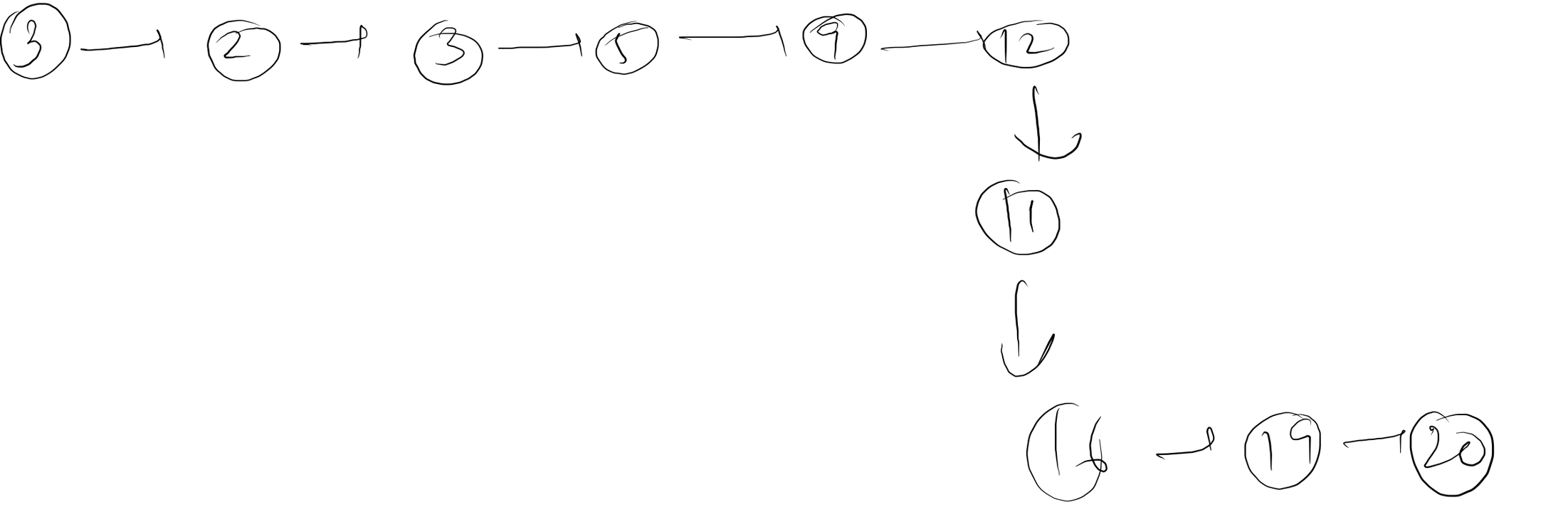

System design
company
7 — 9 LPA
1499
10 days
18 — 24 LPA
Pursuits
Internship

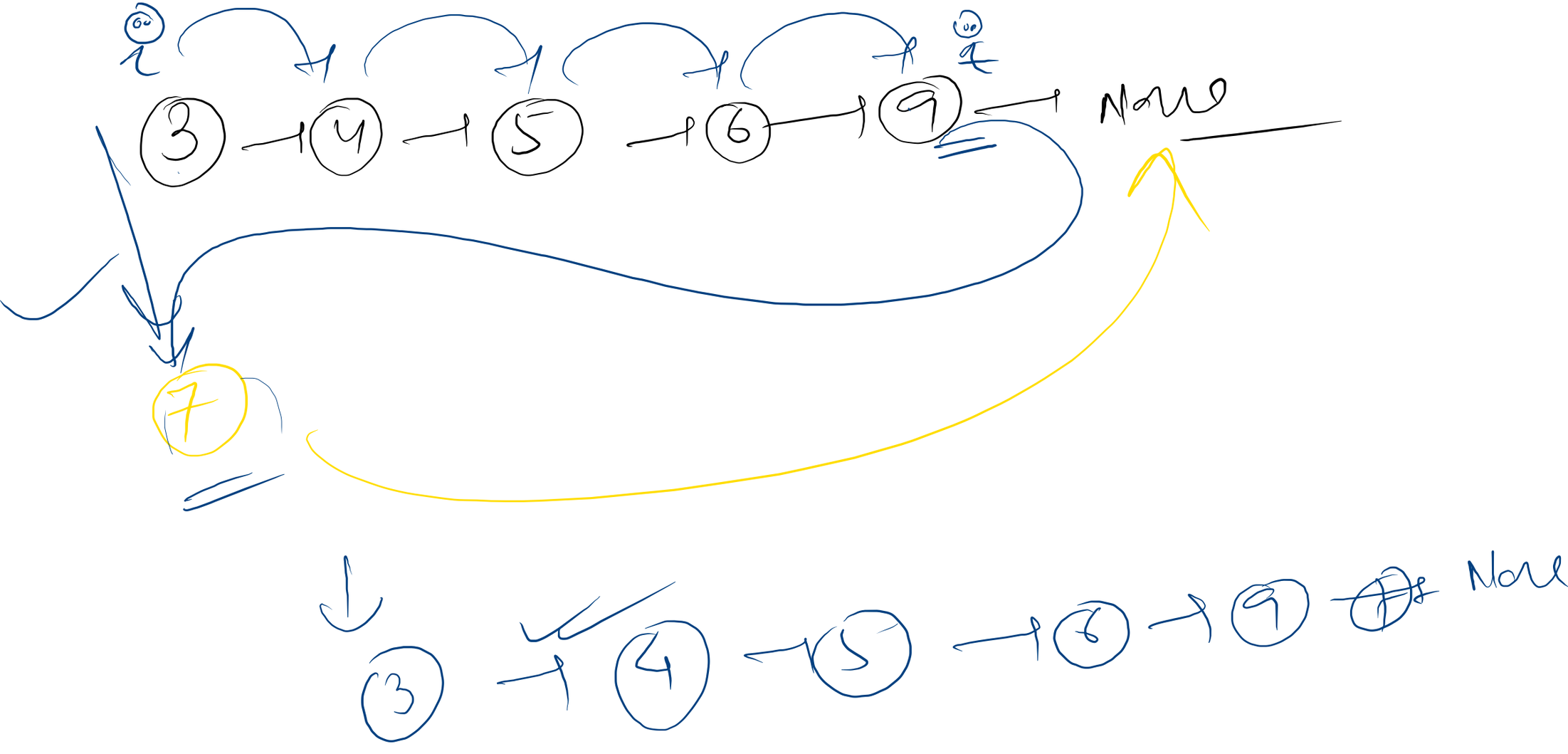
Thanks!

12:30 Your service starts !!!


8
1
new Node
8
3
4
2
1
9
None
prev


def __init__(self, data=None, next=None)
default value
head=Node(1)
Node( )
Node('1')
value
Node (self, None None)
next
None
1


3
2
3
5
9
12
11
14
19
20


3
4
5
6
9
Null
7
3
4
5
6
9
Null