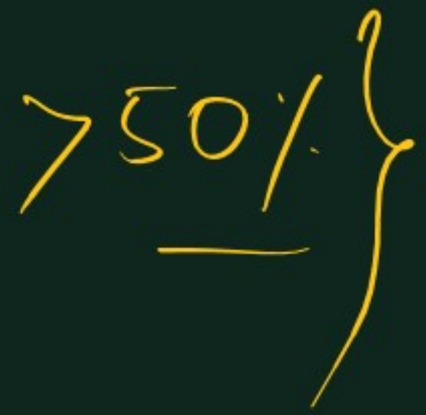
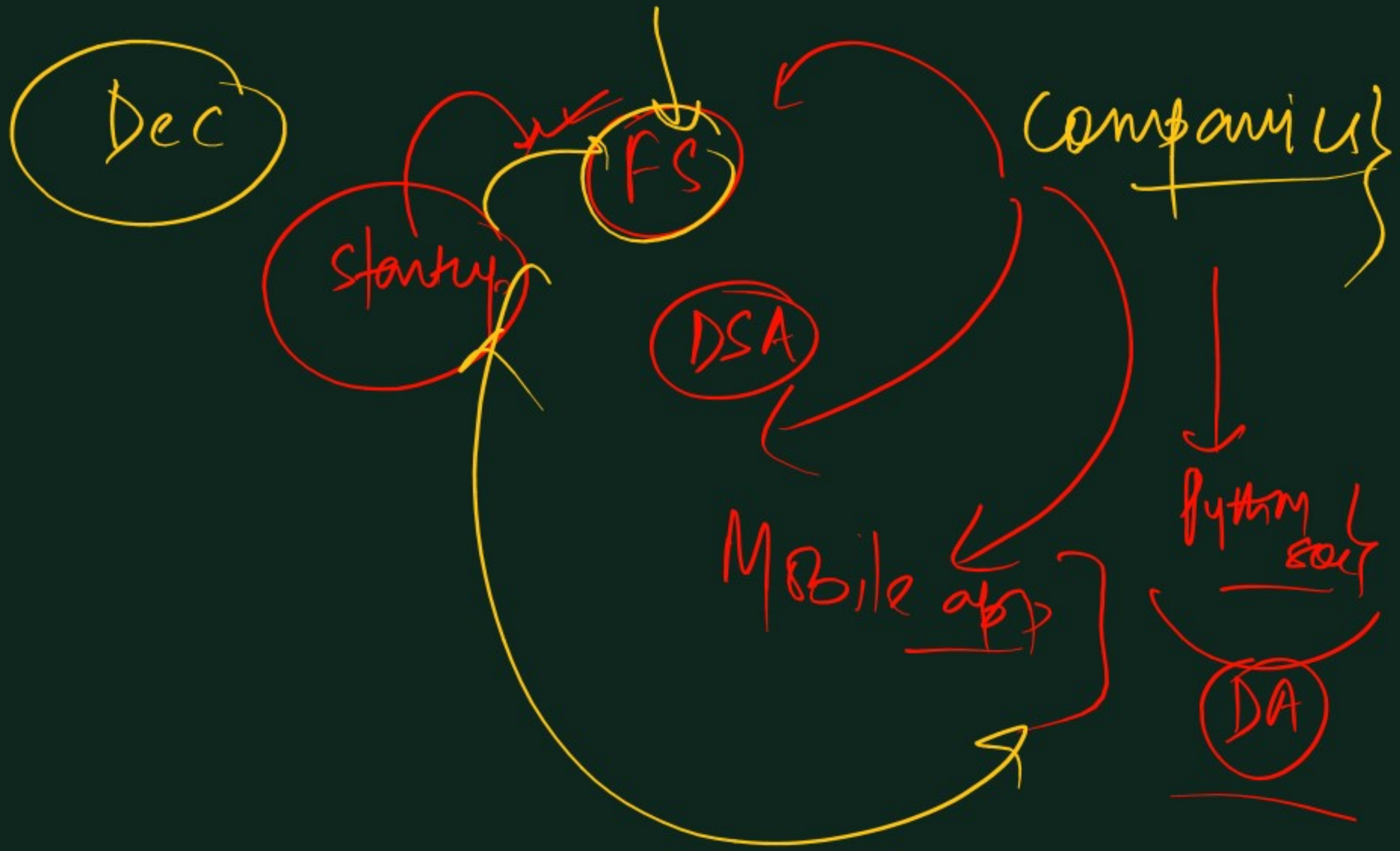
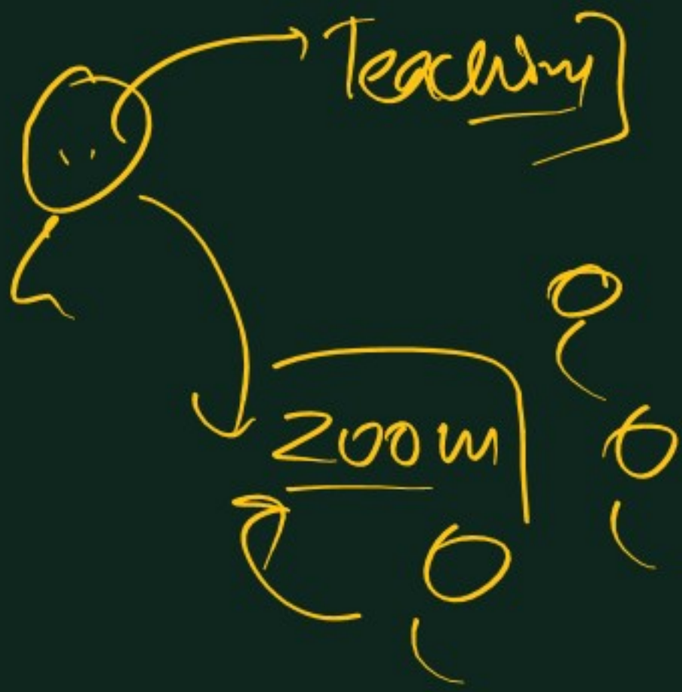
I have no time for anything!

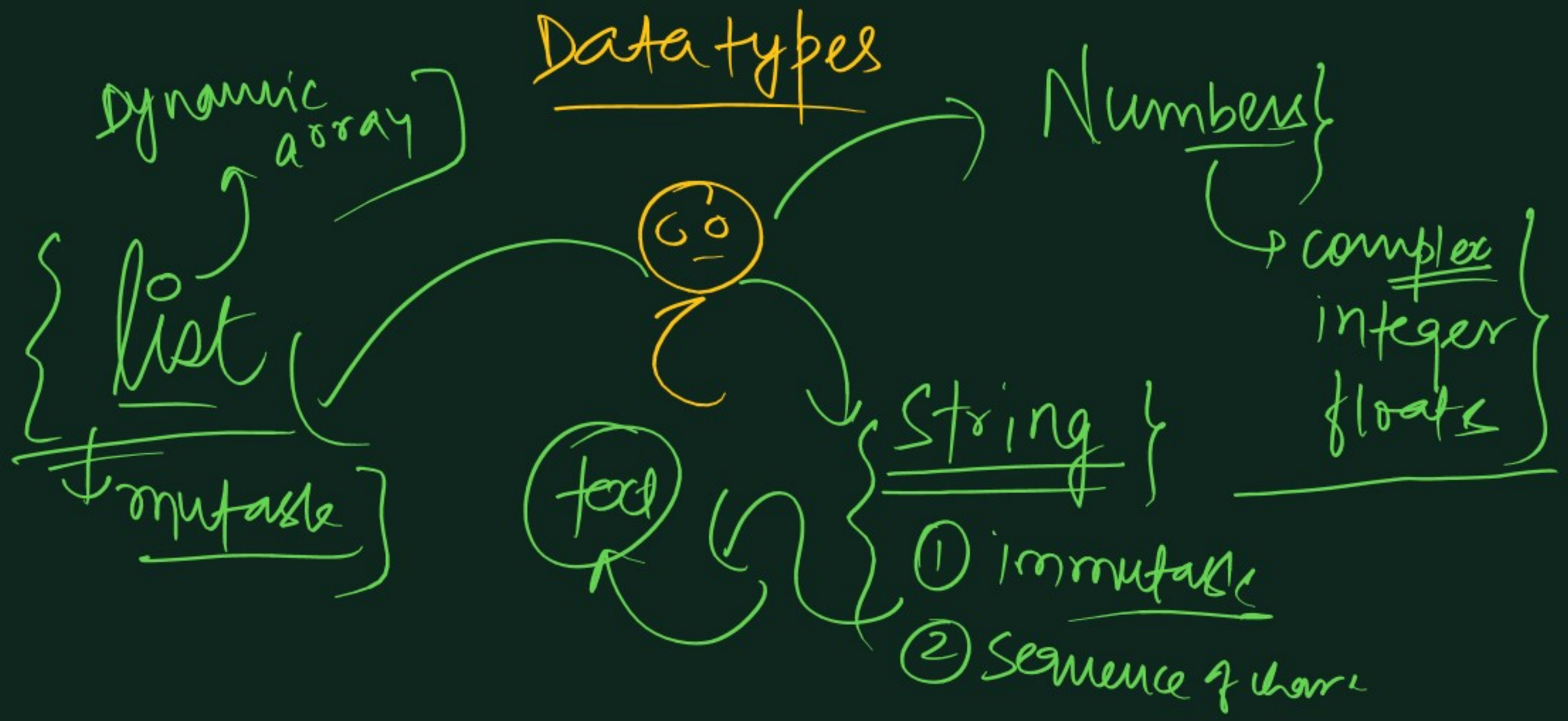

>50%

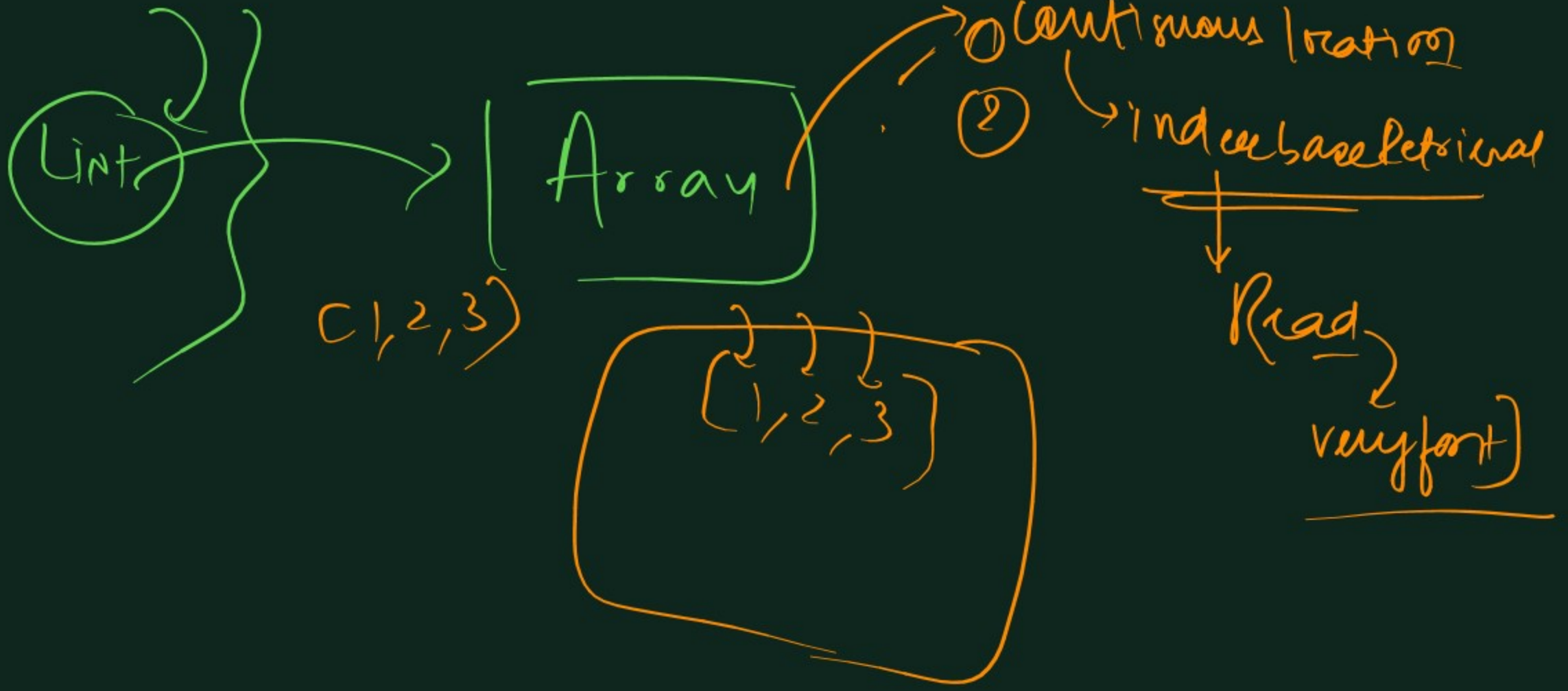

Dec
FS
Company wise
Statistics
DSA
Mobile app
Python
SQL
DA

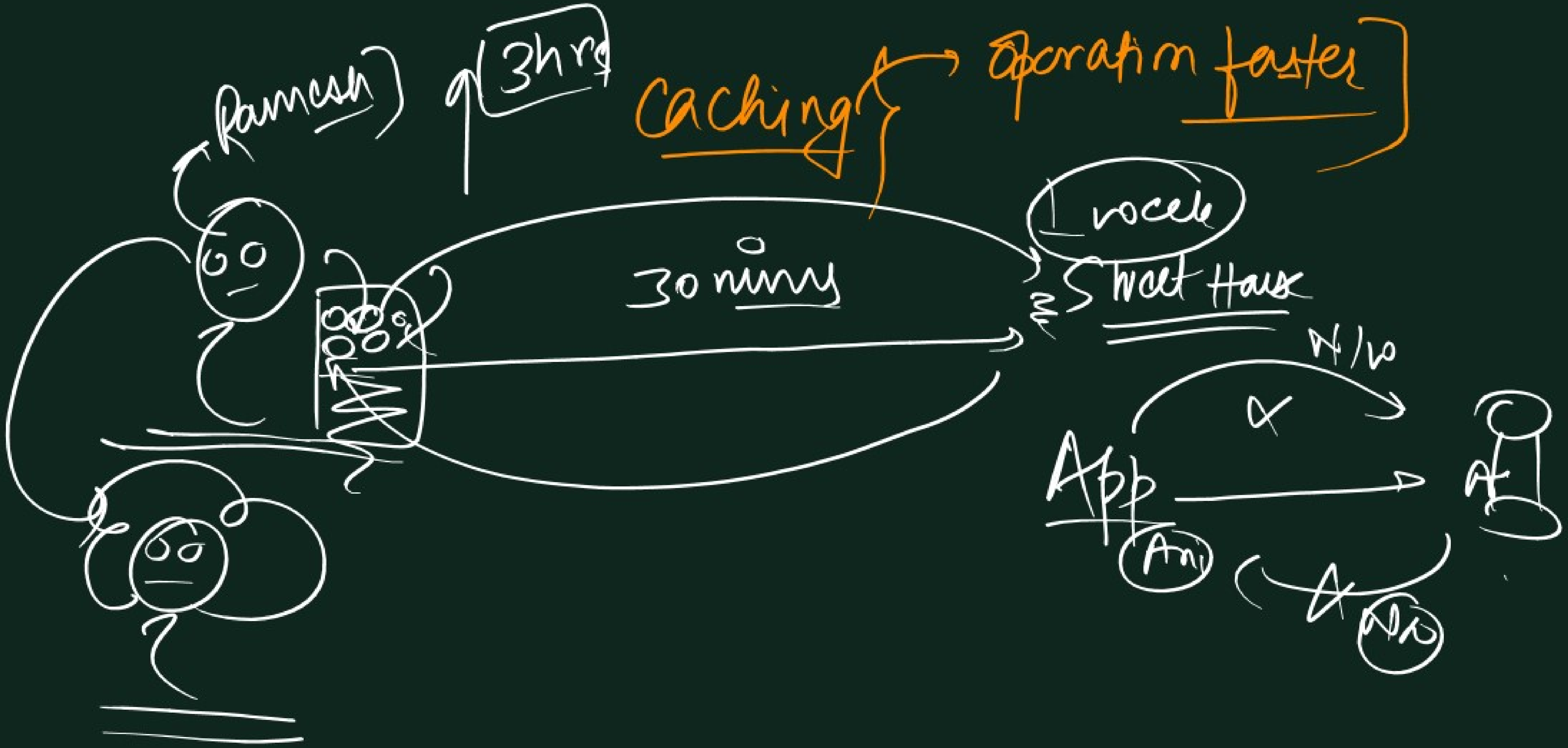
Time is limited



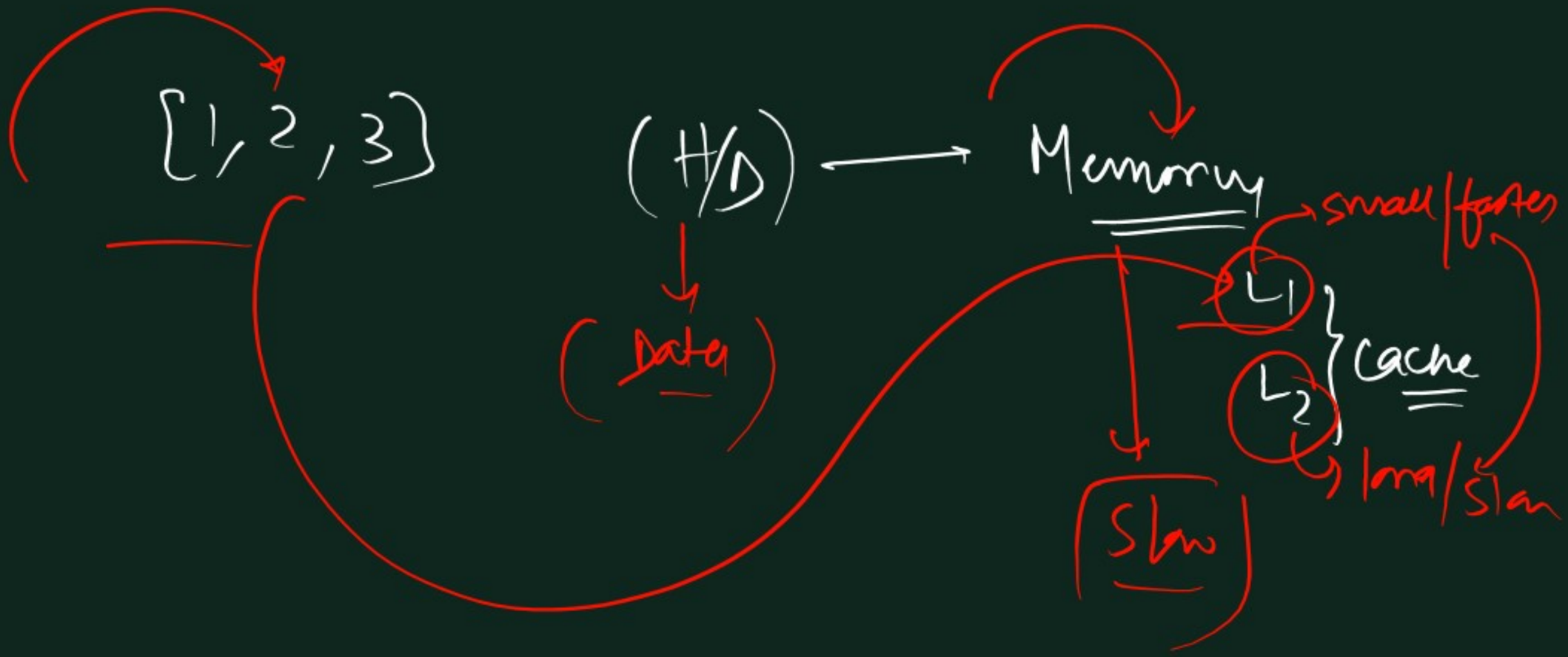

DSA
Language
Netbeans
DSA

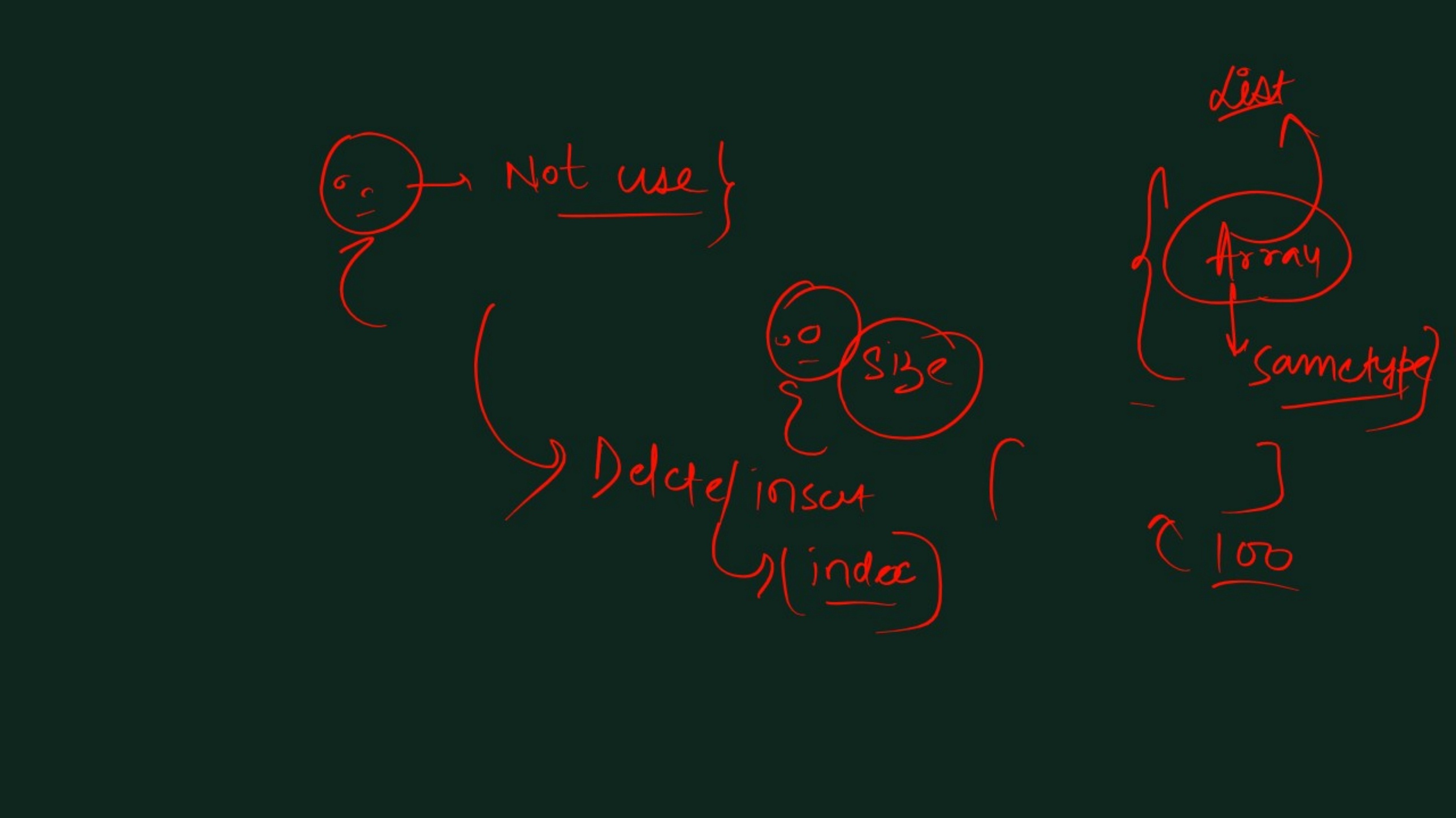

Datatypes
Dynamic array
list
mutable
Numbers
complex
integer
floats
String
food
① immutable
② Sequence of chars

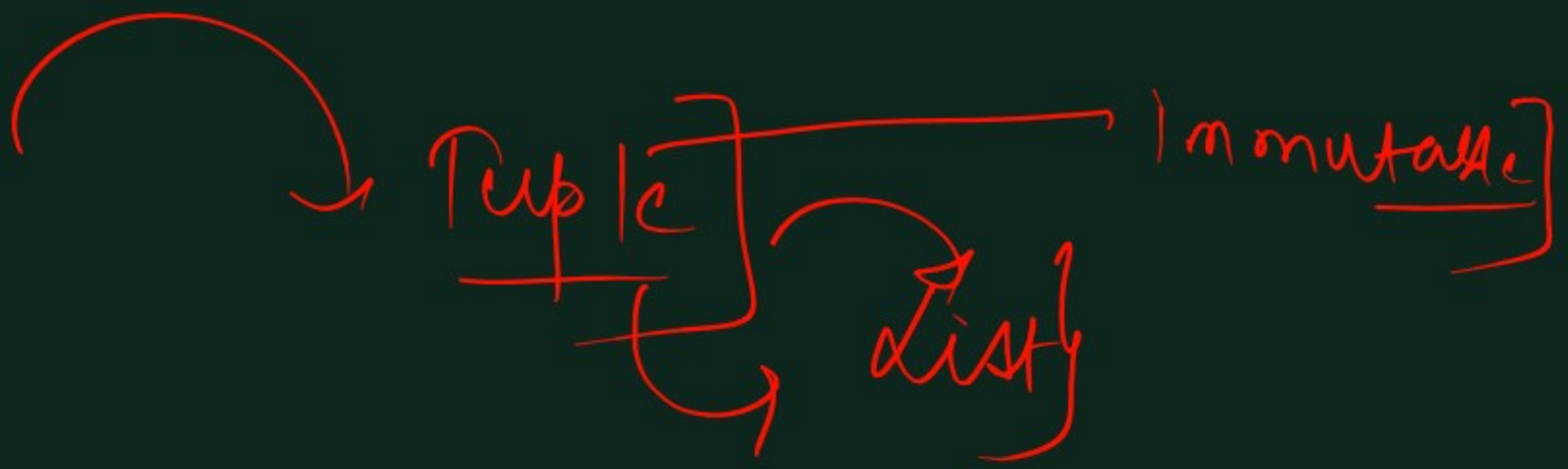

Array
(1,2,3)
[1,2,3]
Read
very fast

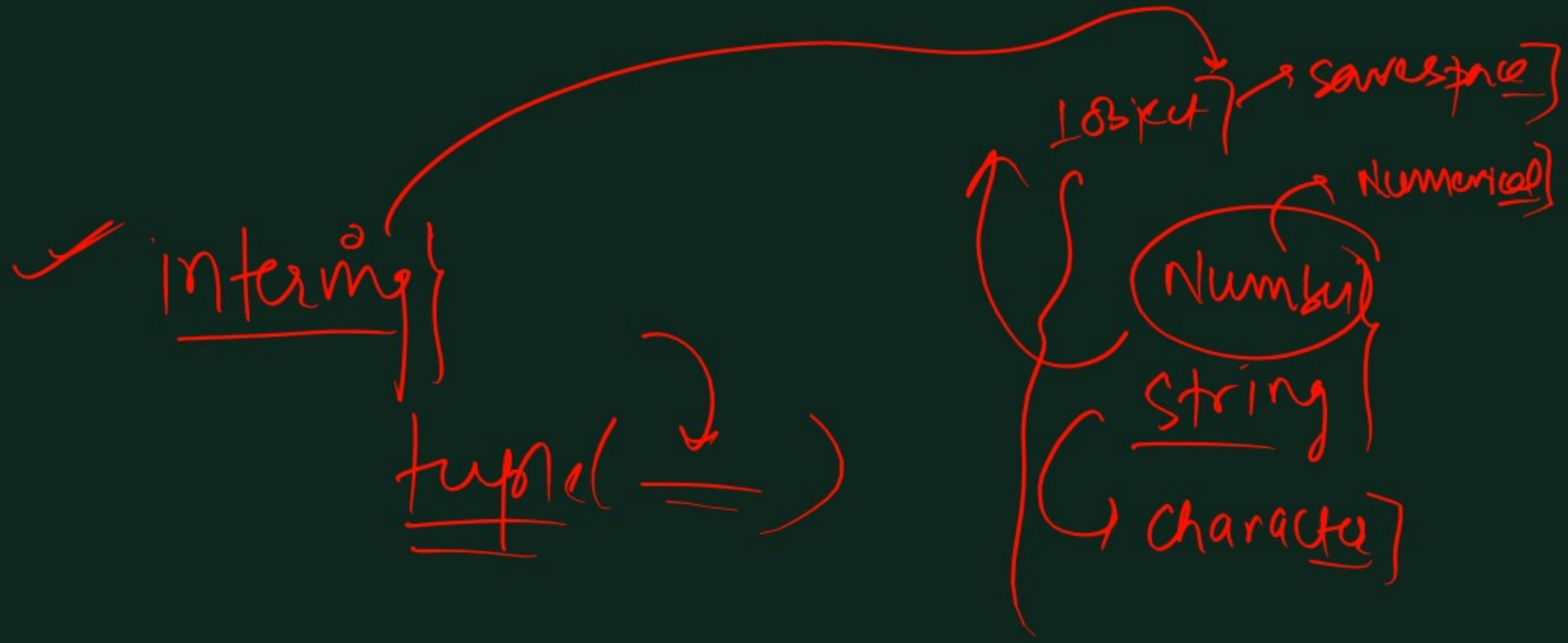


Ramesh
3hrs
Caching
Operation faster
30 mins
Grocery
Sweet House
App


[1, 2, 3]
(H/D)
Memory
Data
small/faster
L1
L2
cache
long/slow
Slow


Not use
List
Array
Same type
size
Delete/insert
(index
100


Tuple
Immutable
List


int
type(=)
Object
Numerical
Number
String
character

Dictionary

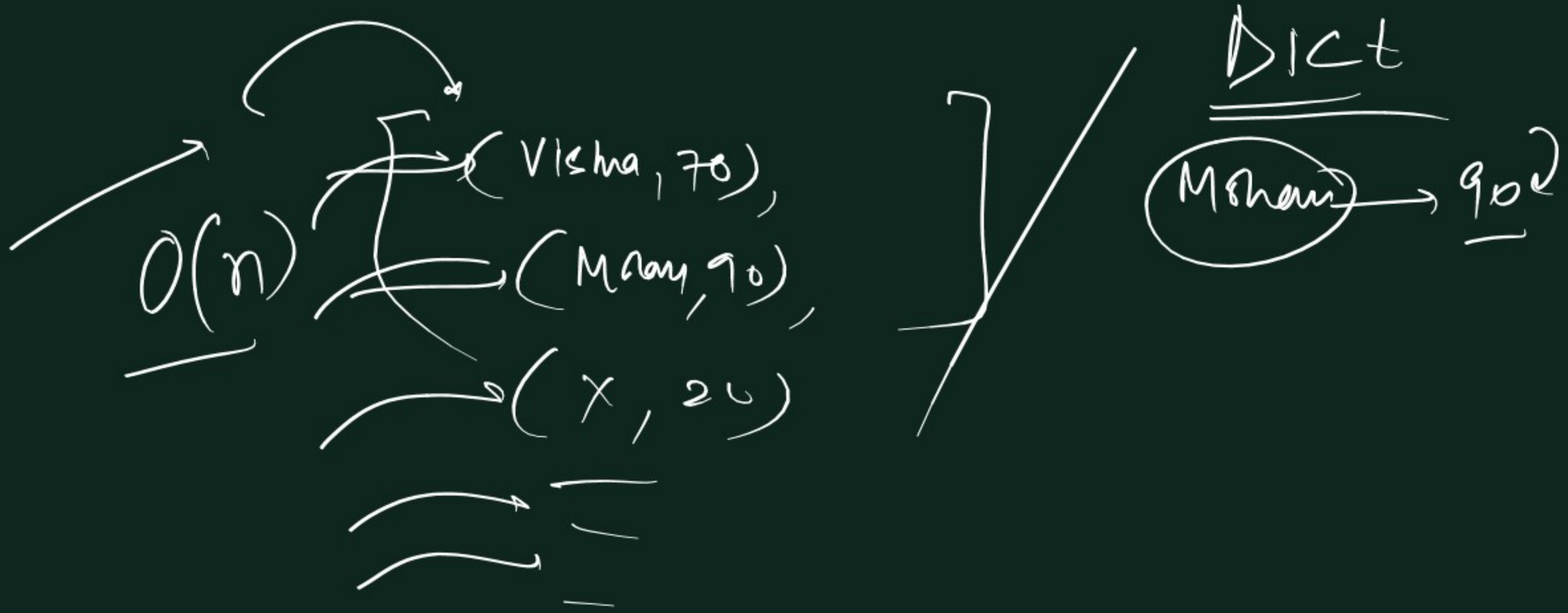
{key, value}

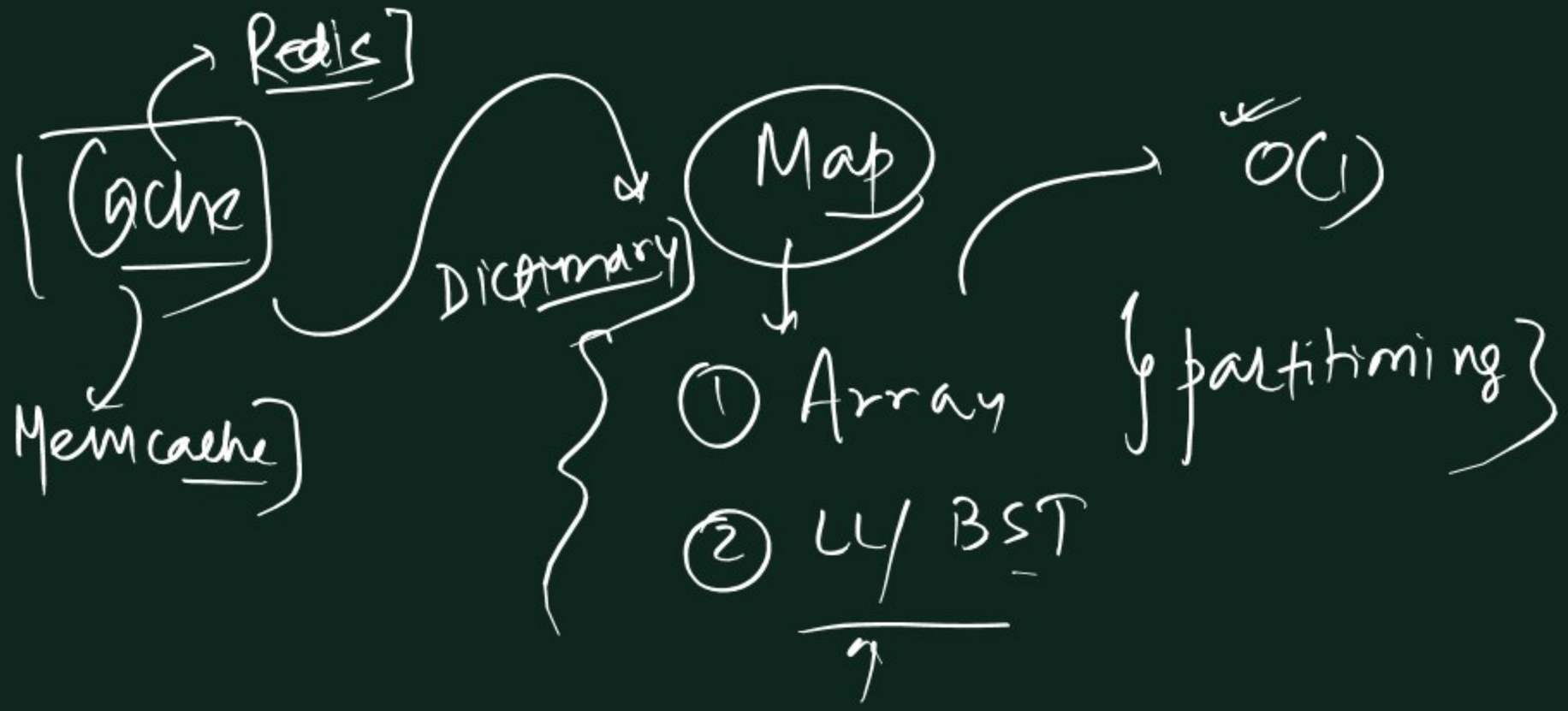
↓ when/Advantages

Name → Salary

Vishnu 70

Maan 75

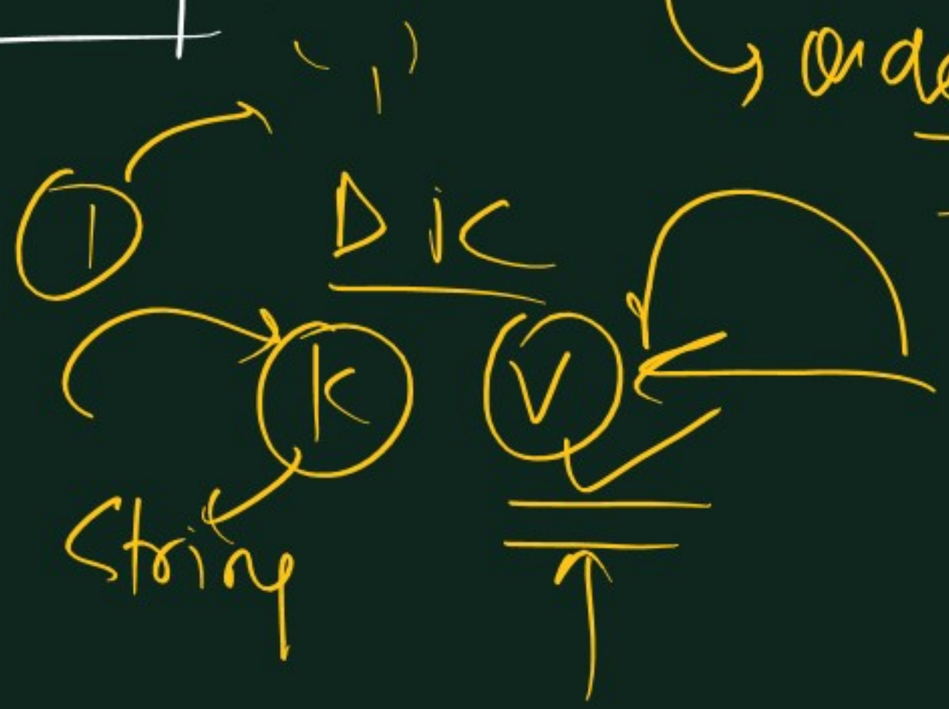

O(n)
(Vishu, 70)
(Manu, 90)
(X, 20)
Dict
Manu → 90


Redis
Cache
Memcache
Dictionary
Map
O(1)
① Array
② LL / BST
partitioning

Challenge:-

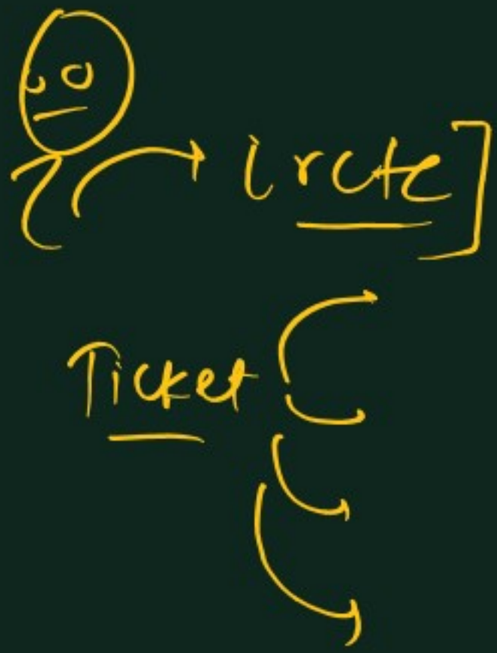
Dictionary {

Order]



Set $\{1, 2, 3\}$

Use case

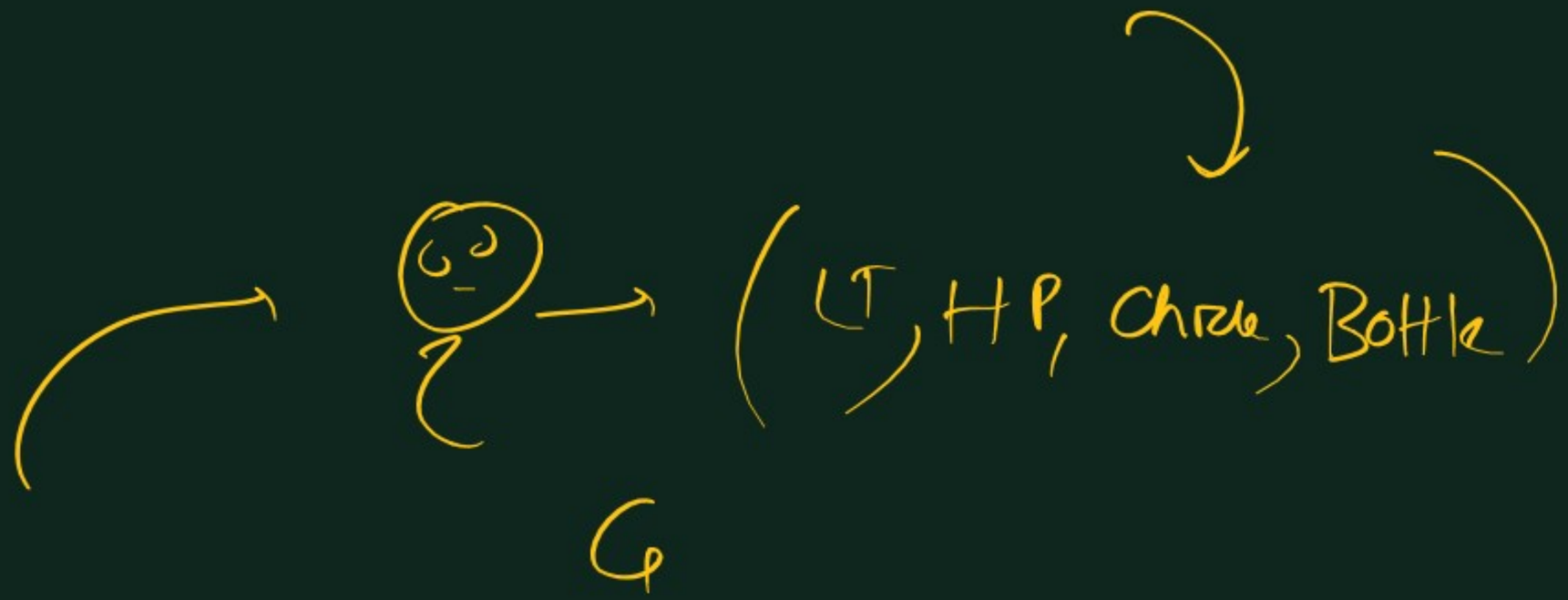



Set
Map
Arrays
LL/BST
Platform

frozenset

set

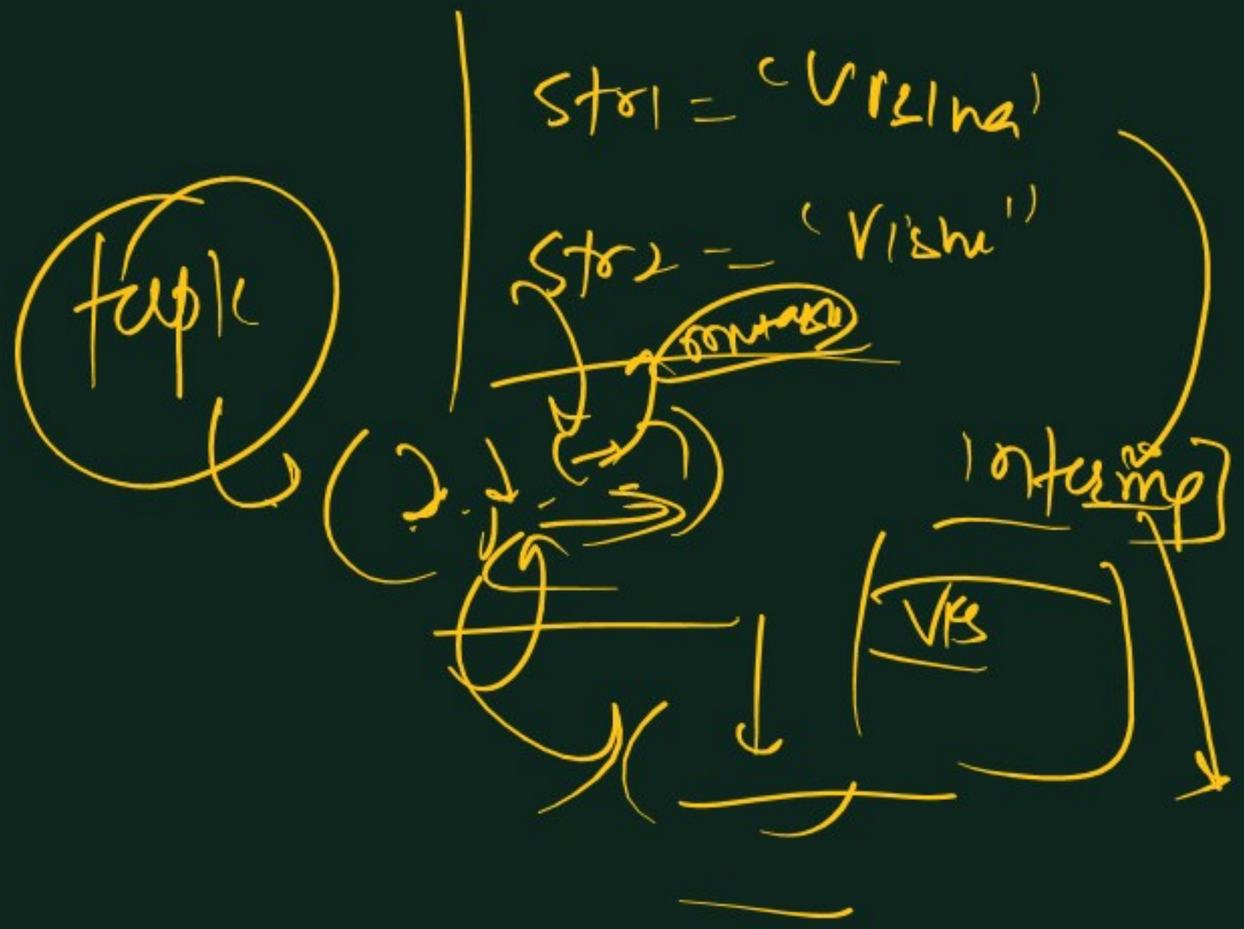
immutable


LT, HP, Chrm, Bottle
G

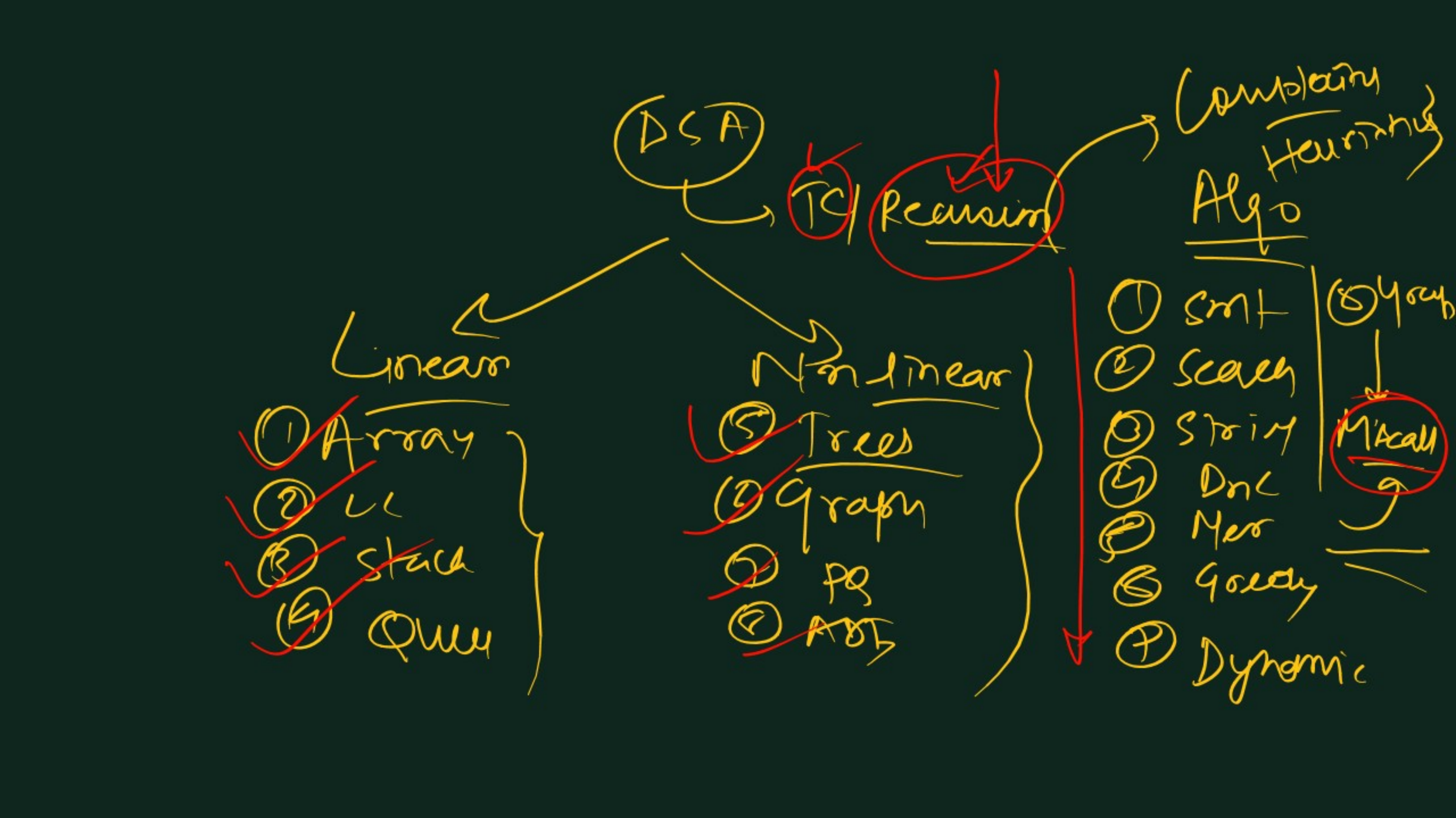
Operators → Unary Op

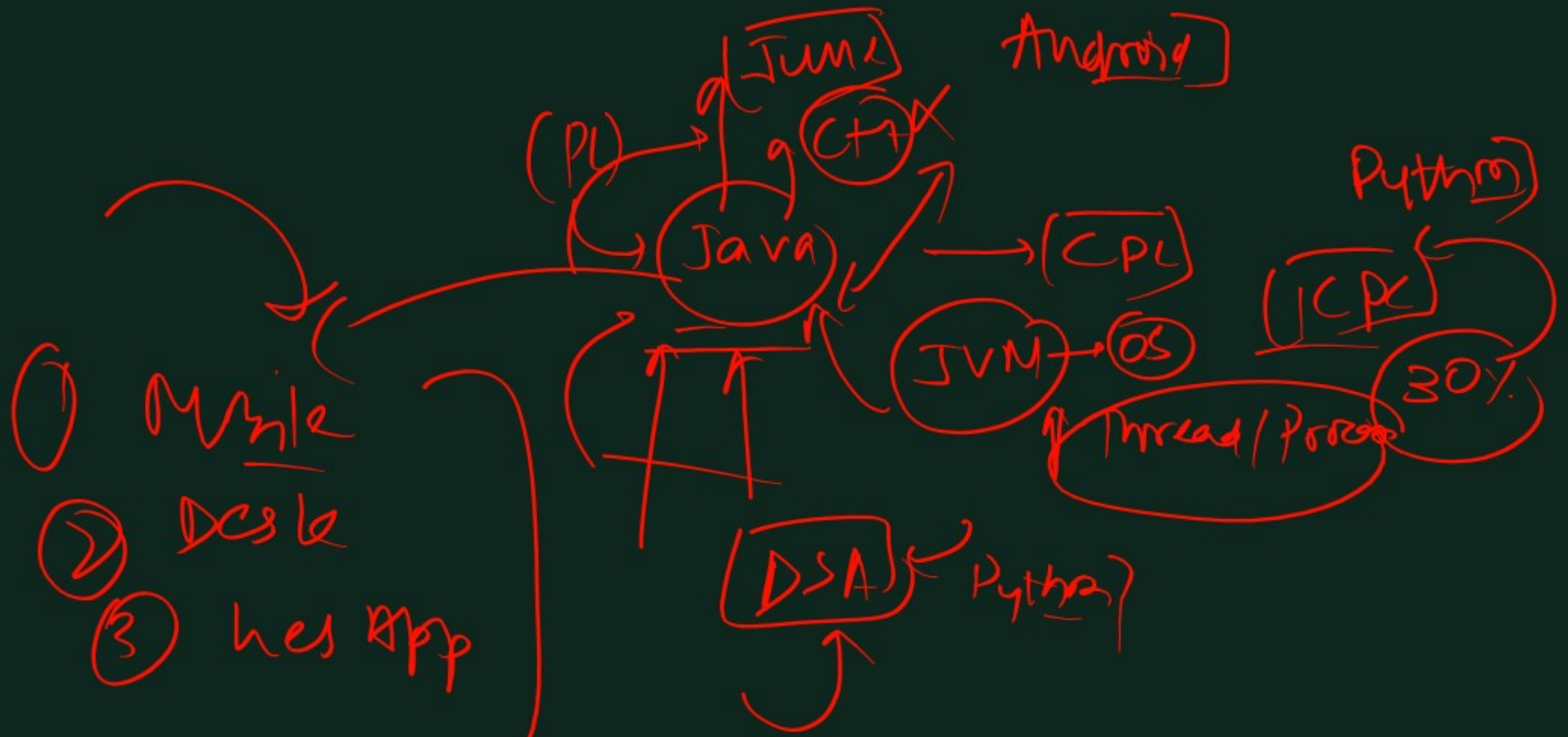
10 ⟶ Binary

1010 ⟶ Decimal No


str1 = 'Vishnu'
str2 = 'Vishu'
tuple
interrup
VB


USA
TO

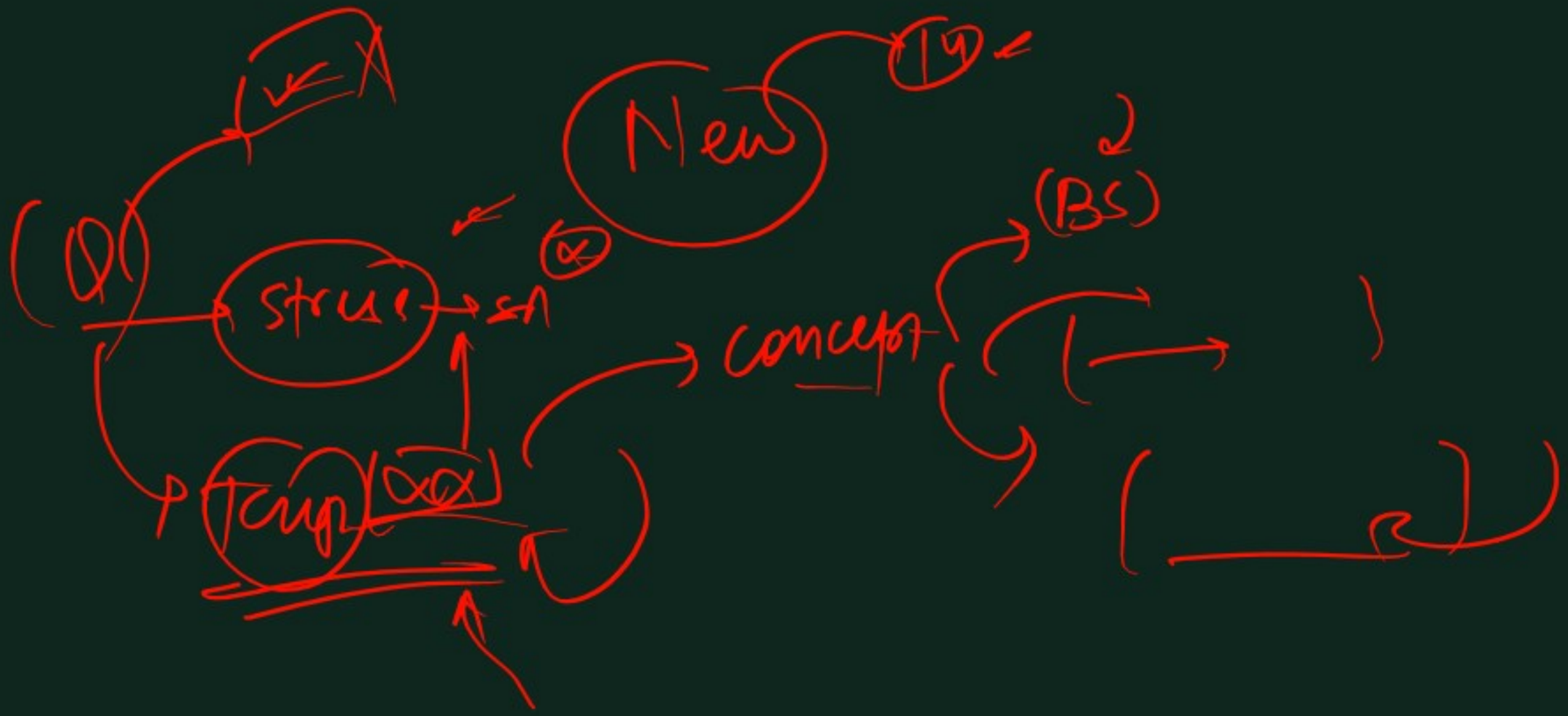

DSA
TC
Recursion
Complexity
Heuristics
Algo
Linear
1 Array
2 LL
3 Stack
4 Queue
Non linear
5 Trees
6 Graph
7 PQ
8 ADT
1 Sort
2 Search
3 String
4 DnC
5 Mer
6 Greedy
7 Dynamic
6 Greedy
MAcall

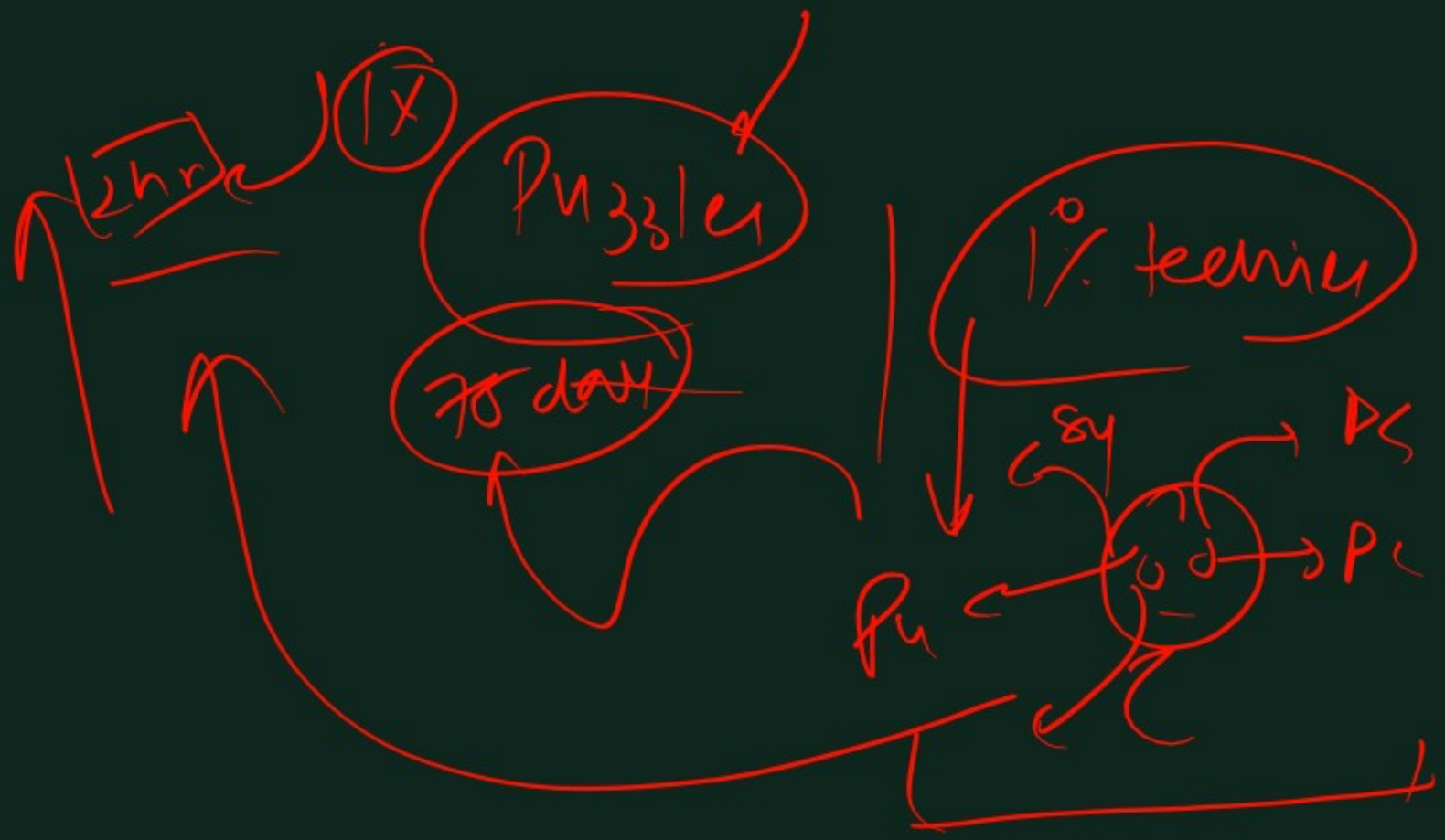

Java
Android
(PL)
C++
CPL
Python
JVM
OS
30%
DSA
Python

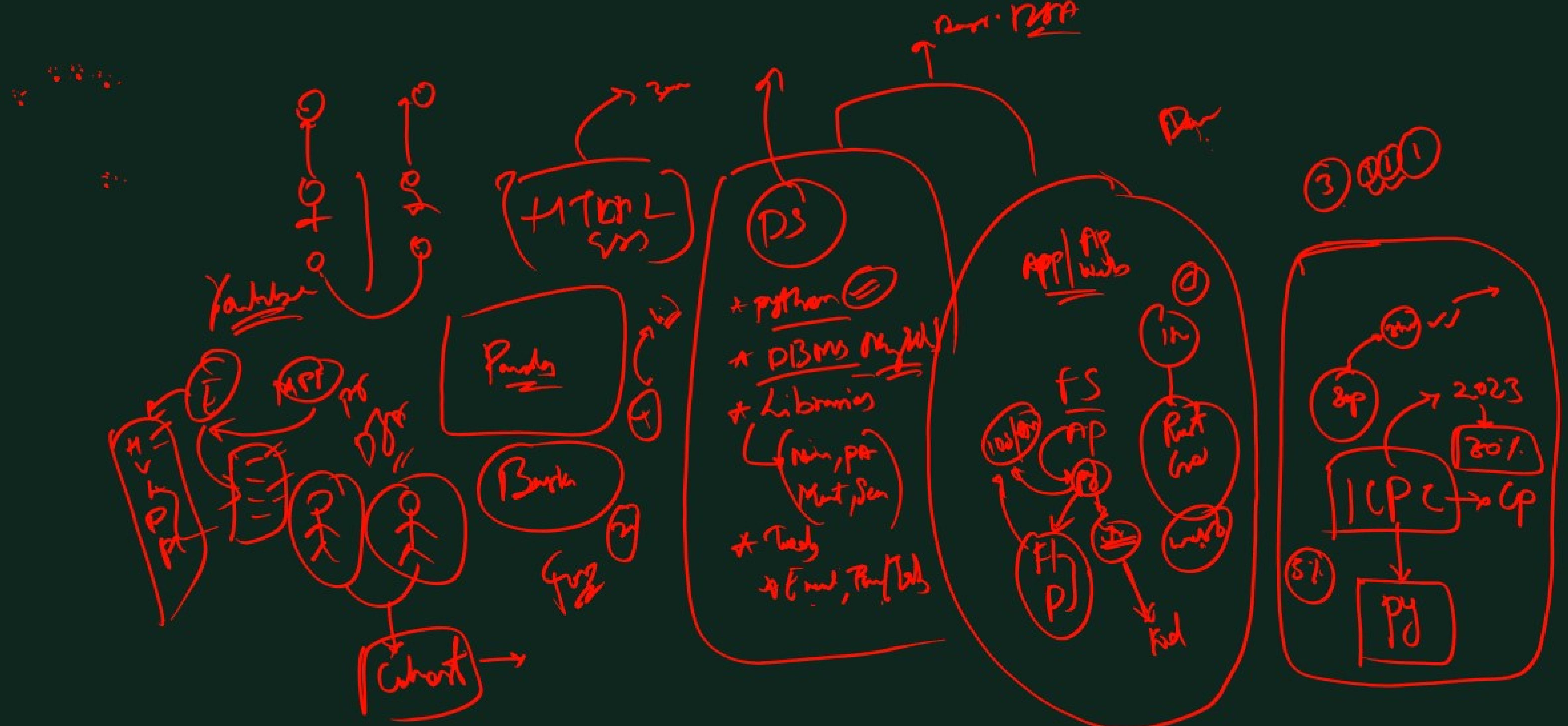
G | F | M | Li } → Language Agnostic

C / C++ / Java / Python

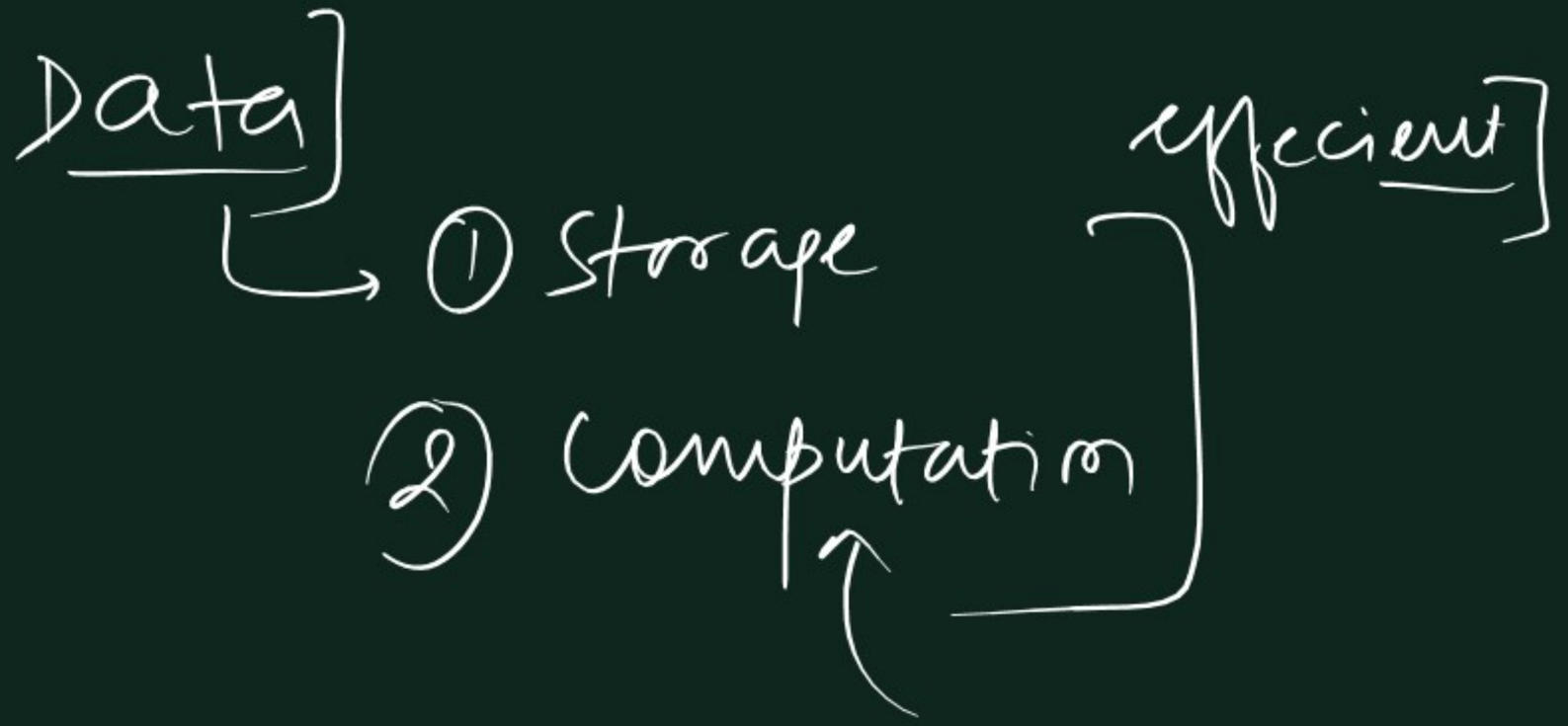
{ , ; }

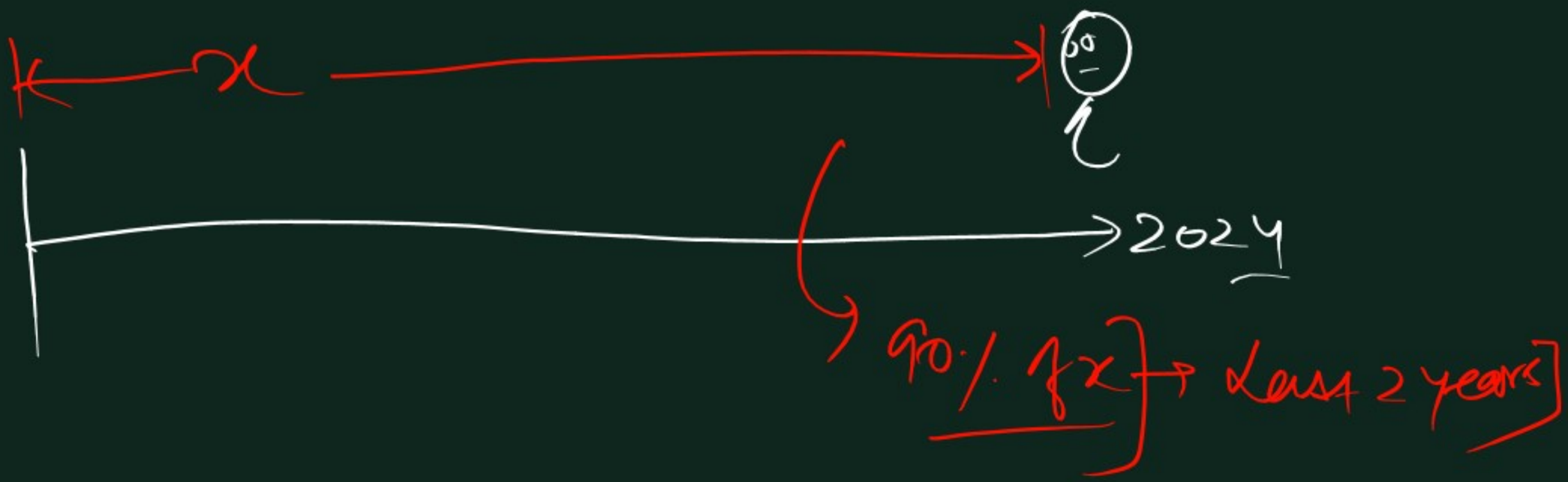

New
struct
concept

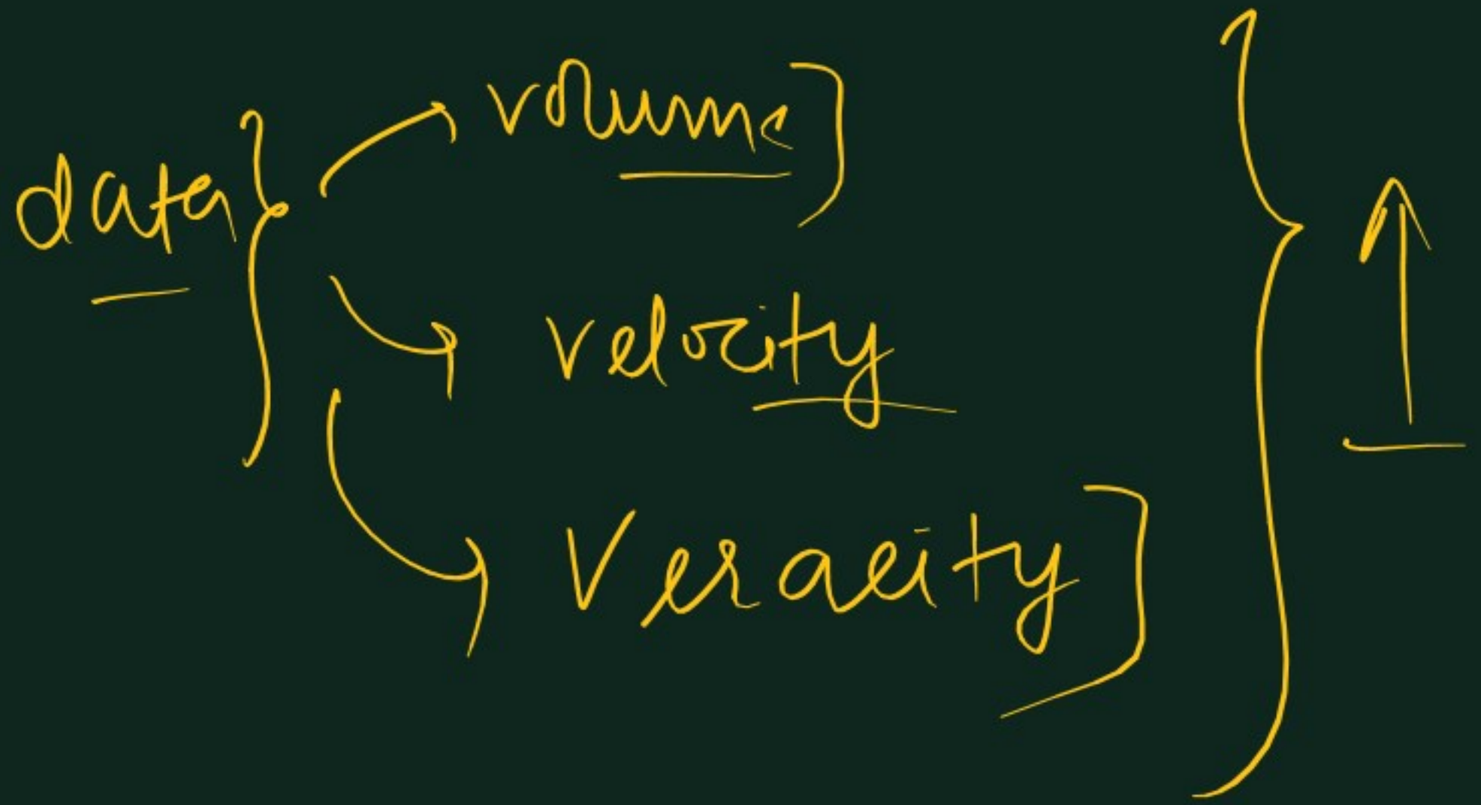

2hr
1x
Puzzles
78 days
1% teens
84
PS
P

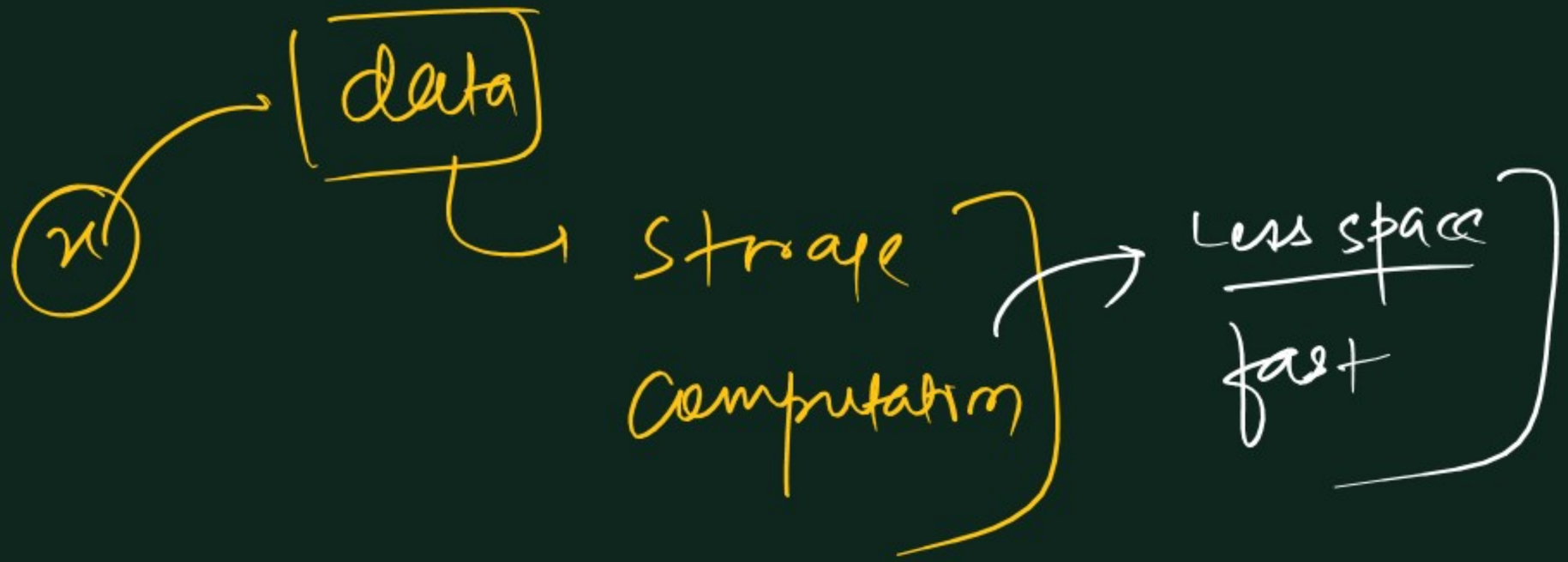

HTML
CSS
DS
python
DBMS
Libraries
Pandas
FS
App
ICPC
CP
2023
30%
py
Cohort

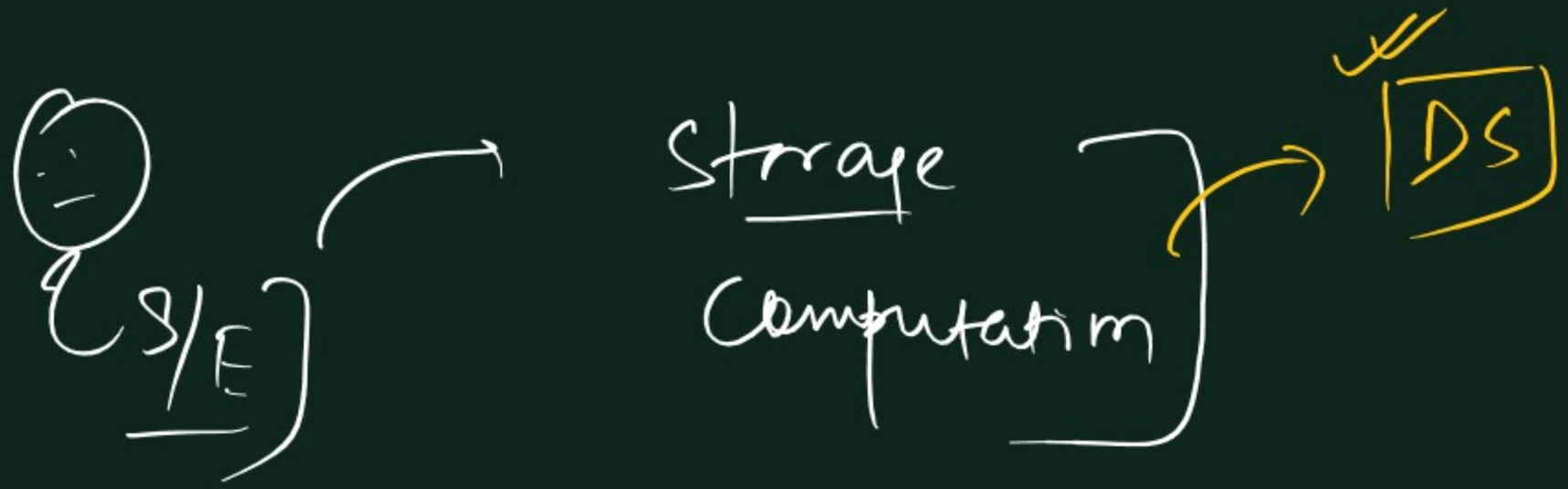
Data Structure }


Data
1 Storage
2 Computation
efficient


x
2024
90% of x → Less 2 years


data
volume
velocity
Veracity


data
x
Storage
Computation
Less space
fast


S/E
Storage
Computation
DS

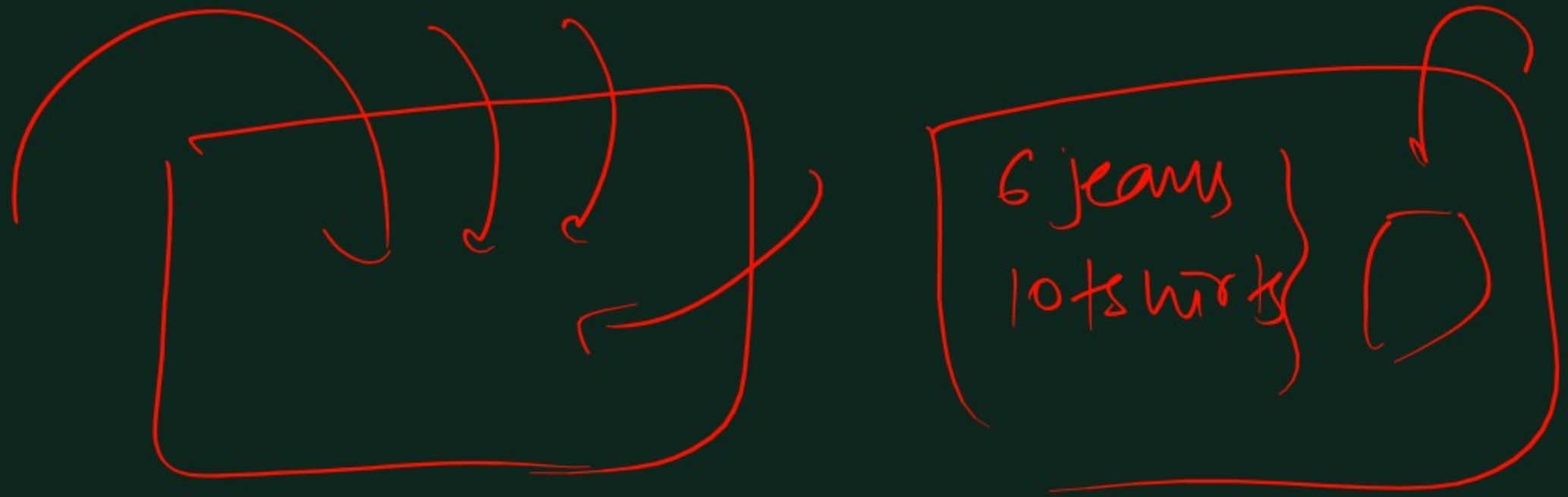
Why?

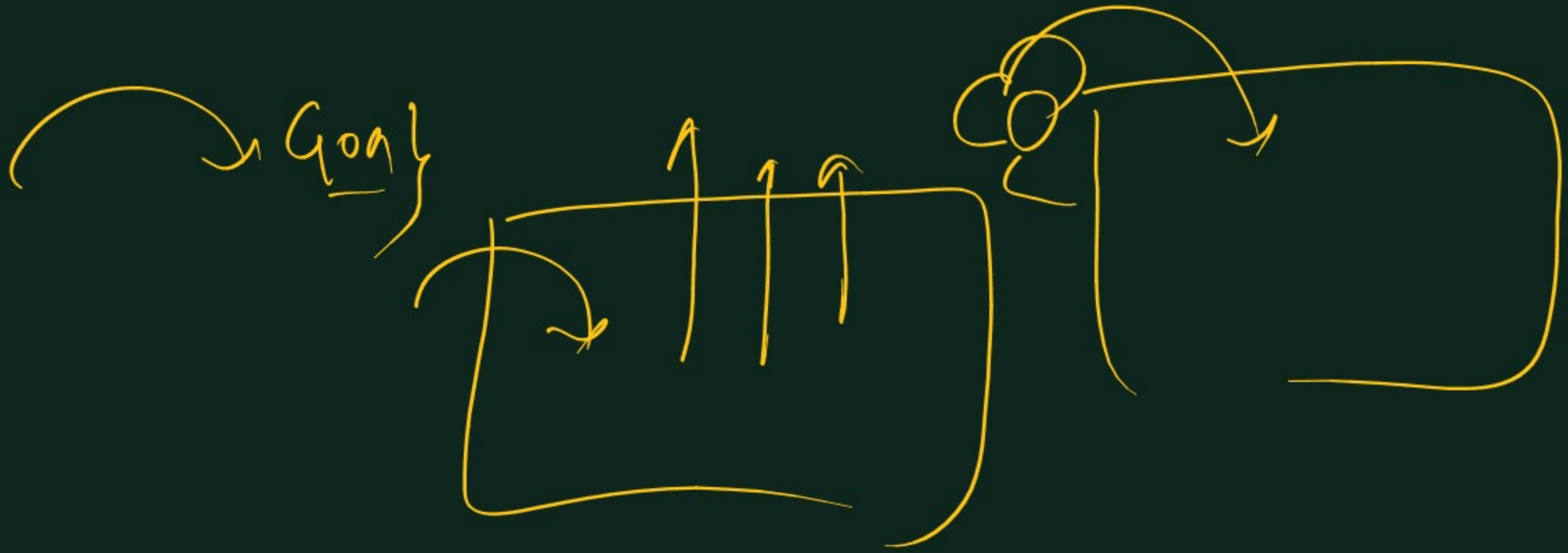
Data Structure

→ structuring Data

→ storage

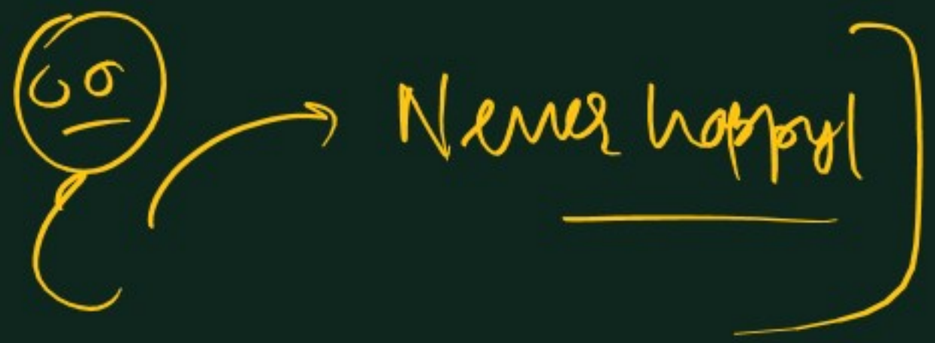
→ computation


6 jeans
10 t-shirts

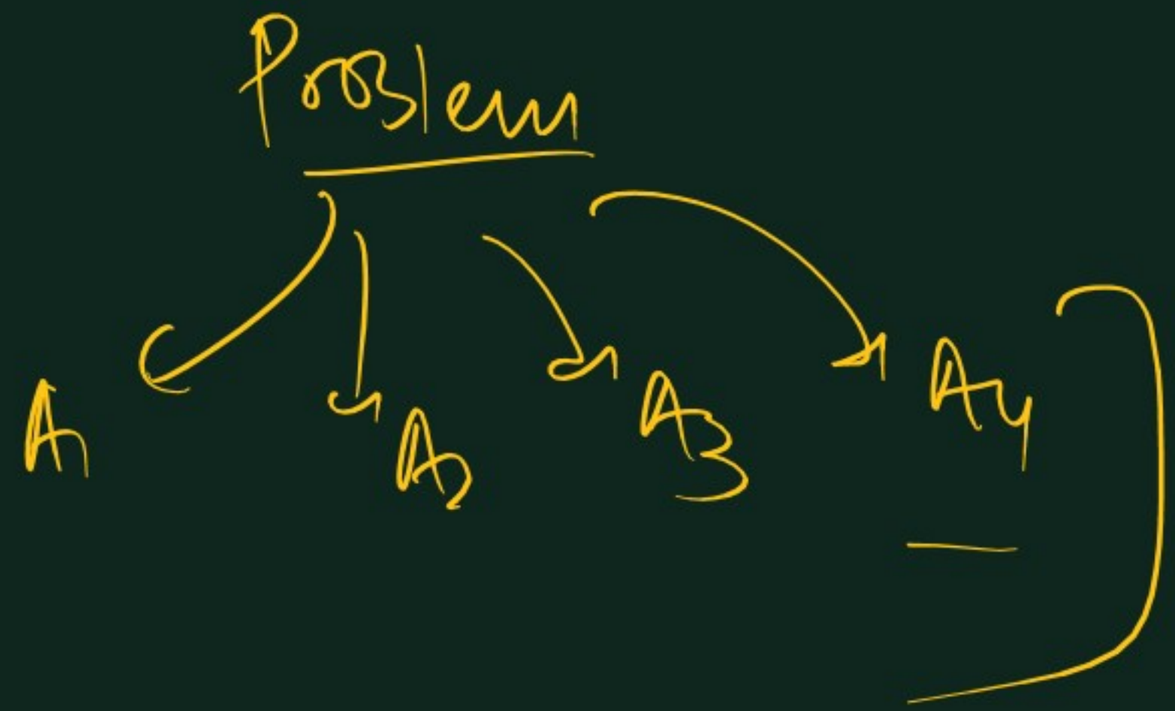

Goal

Algorithms


A1
A2
A3
Airport


Never happy!


Problem
fast
efficient


Problem
A1
A2
A3
A4

A → Less time] Better Algo] → Time complexity]

Time taken (Mac) P

① Logic
② PL → compiled
↳ Interpreted
③ OS / Processors

$A_1 \rightarrow t_1$

$A_2 \rightarrow t_2$

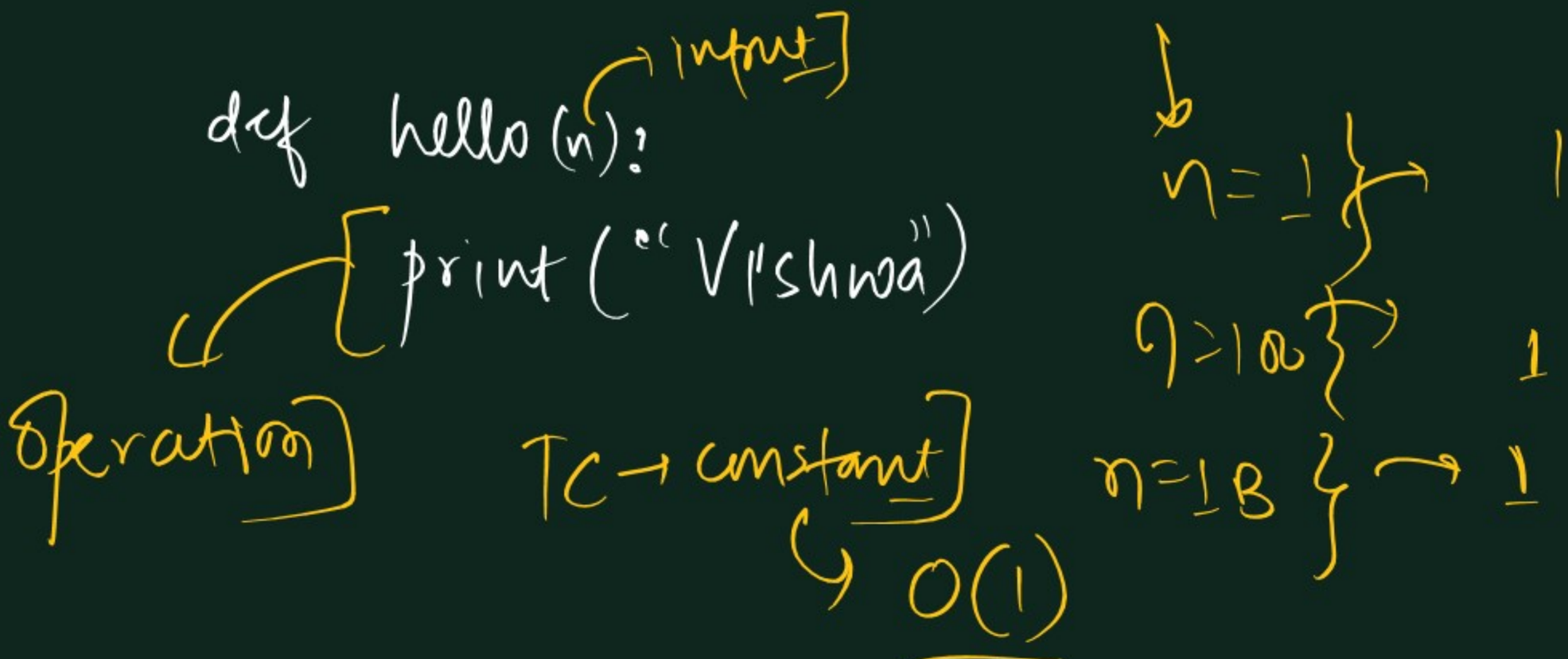
(Python / Windows)

$t_1 < t_2$

A_1 is better than A_2

can't say

Time complexity { No of operations is proportional to input size }

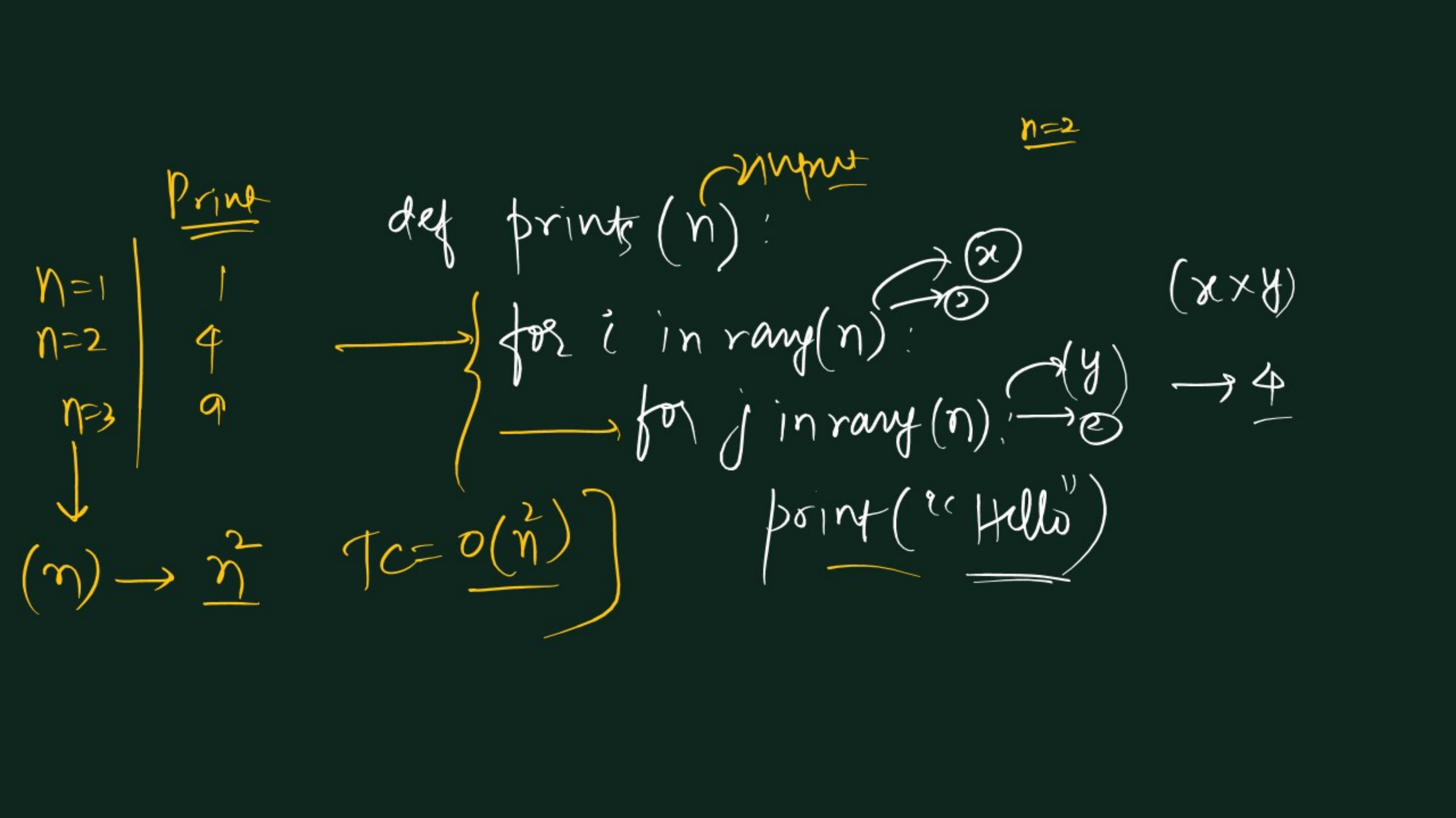

input
def hello(n):
print("Vishwa")
Operation
TC → constant
O(1)
n=1
1
n=10
1
n=1B
1

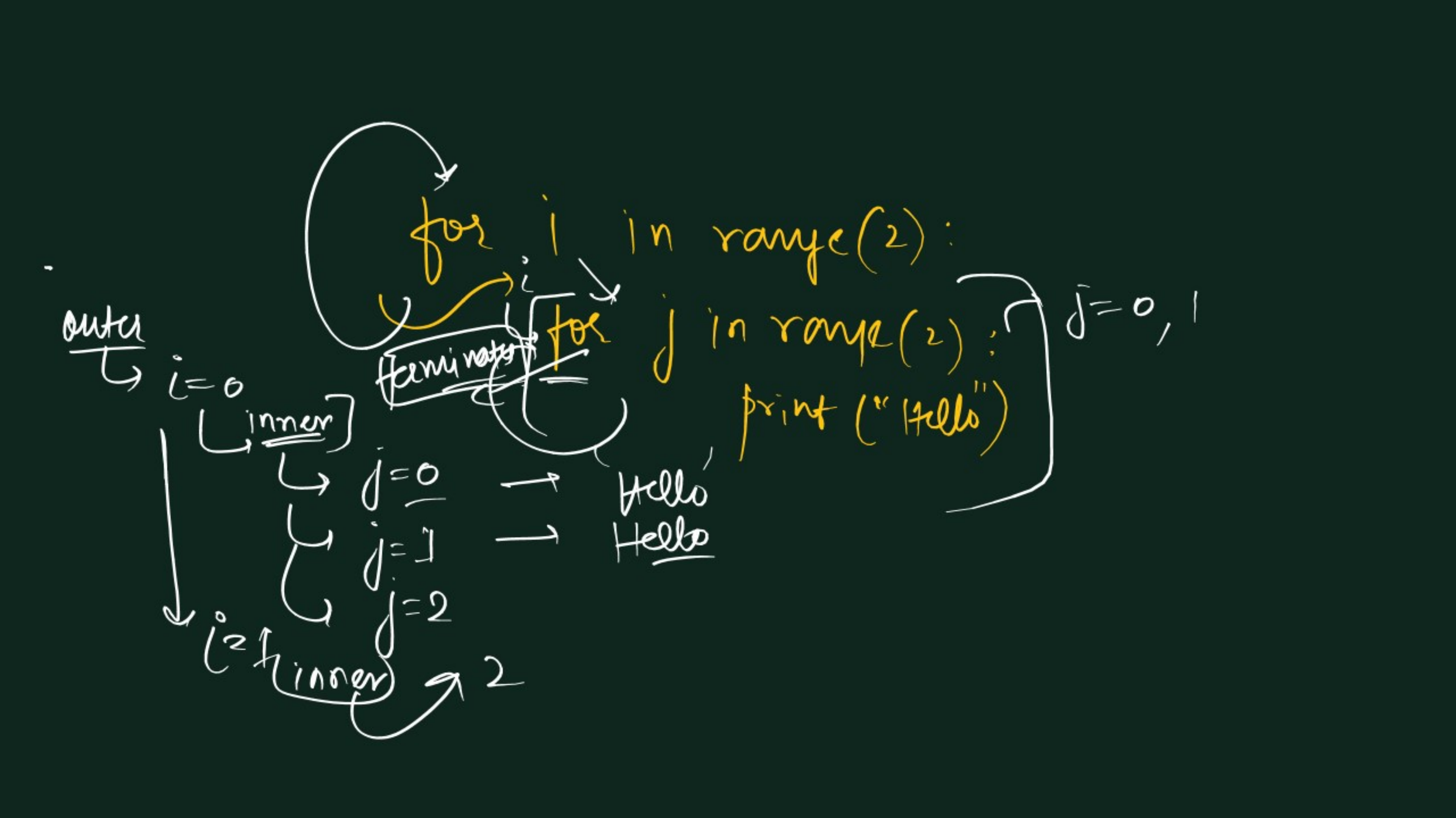
```
def print(n):
    for i in range(n):
        print("hello")
```

Loop ←

n	operations
n = 1	1
n = 100	100
n = 1M	1M

TC = O(n)


n=2
Prints
n=1 | 1
n=2 | 4
n=3 | 9
(n) → n²
def prints(n):
input
for i in range(n):
x
2
for j in range(n):
y
2
(x×y)
→ 4
TC = O(n²)
print("Hello")


for i in range(2):
for j in range(2):
print("Hello")
j=0,1
outer
i=0
inner
j=0 → Hello
j=1 → Hello
j=2
i=2
inner
2
terminate

TC →

```
def myPrint(n):
    i = n
    while (i > 0):
        print("Hello")
        i = i/2
```

kTimes

i <= 1

K in terms of n

i <= 0

i is very small

① $i = n$

② $i = n/2$

③ $i = n/2^2 = \frac{n}{2^{3-1}}$

④ $i = \frac{n}{2^{4-1}}$

$(k) \rightarrow i = \frac{n}{2^{k-1}}$

$(k+1) \rightarrow i = \frac{n}{2^k}$

$i = \frac{n}{2^k}$

$2^k \approx n$

$$i = \frac{n}{2^k}$$

$$a \propto b$$

$$a \propto kb$$

$\Rightarrow i \leq 0$

$\Rightarrow \frac{n}{2^k} \leq 0$

$\frac{n}{2^k} \simeq$ small value

$\Rightarrow \frac{n}{2^k} \simeq$ small value

$\Rightarrow n \simeq$ small value $\times 2^k$

$$n \propto 2^k$$

$$\log_2 n \propto \log_2 2^k$$

No of operation

$$k \propto \log_2 n$$

$i = \frac{n}{2^k}$ $\quad i \leq 1$

$\Rightarrow \frac{n}{2^k} \leq 1$

$\Rightarrow n \leq 2^k$

$2^k \approx n$

$\log_2 2^k \approx \log_2 n \quad TC = O(\log n)$

$\Rightarrow k \propto \log_2 n$

TC=?

O(n)

O(1)

O(n log n)

```
def print(n):
    for i in range(n):
        j = n
        while(j > 1):
            print("Hello")
            j = j/2
        break
```

n × (log n + 1)

TC = O(n log n)