CS & IT ENGINEERING

Data Structures

Introduction to Data Structures

Lec- 05

By- Pankaj Sharma sir

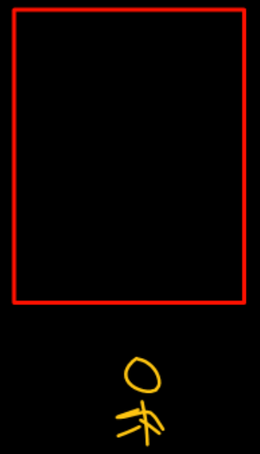
Introduction-5

Array

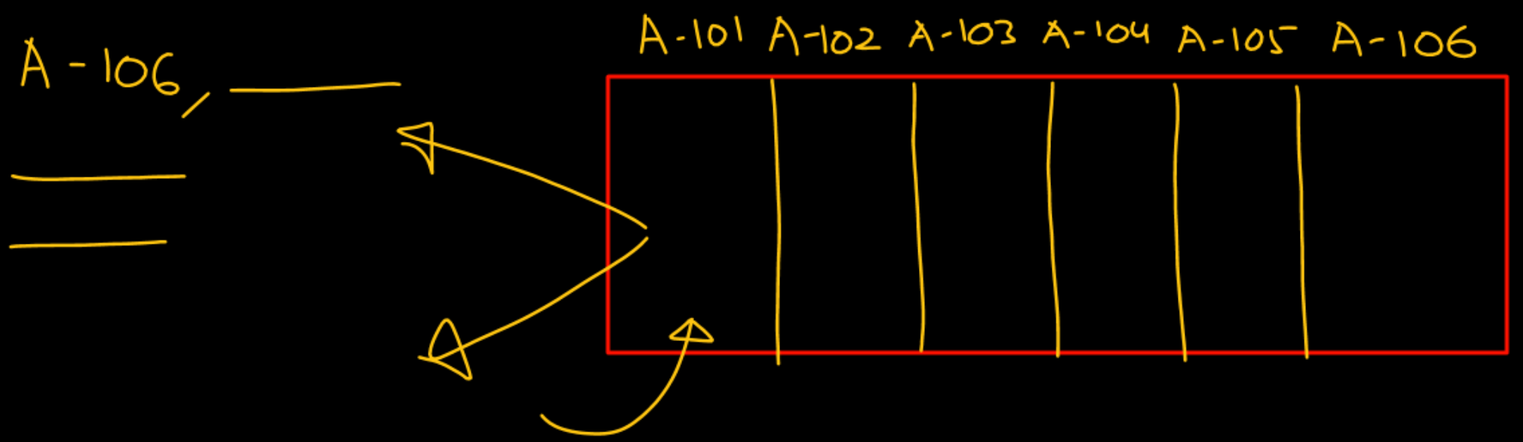
Address :

A-106, Krishna Nagar

Mathura (U.P.)



2 types → Abs. address
2 types → relative address

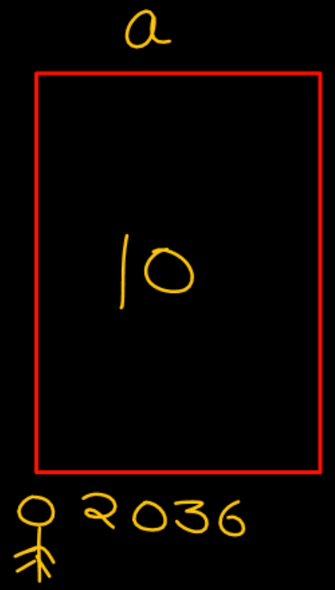


How to find add. of some variable: & (address of operator)

scanf में हमेशा use होता है।

```
int a = 10;
```

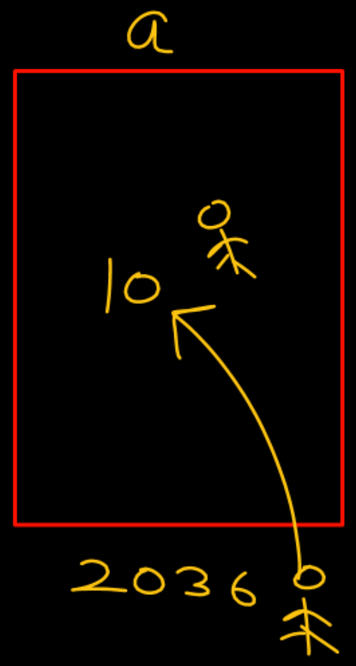
&a ≡ Memory loc. 2036



* ⇒ value at operator

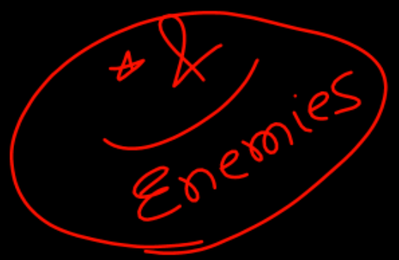
int a = 10;

&a ≡ Mem. location 2036



*(&a) ⇒ value at (Memory location 2036)

*&a = 10



~~*&a = a~~

int a = 10;

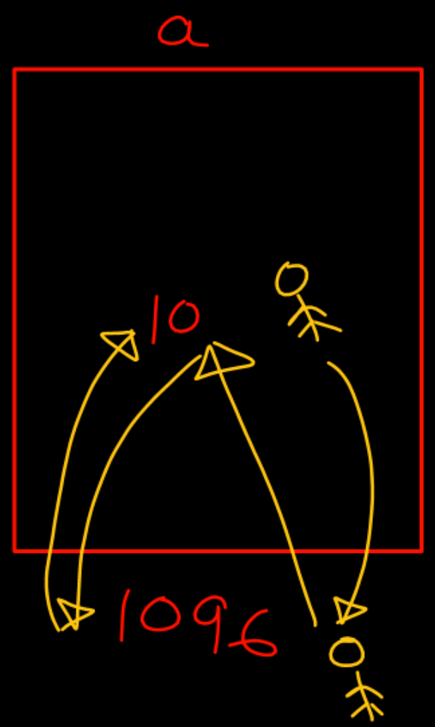
&a ⇒ Mem. location

1096

*(&a) ⇒ a

printf("%d", *&*&*&a)

↳ 10

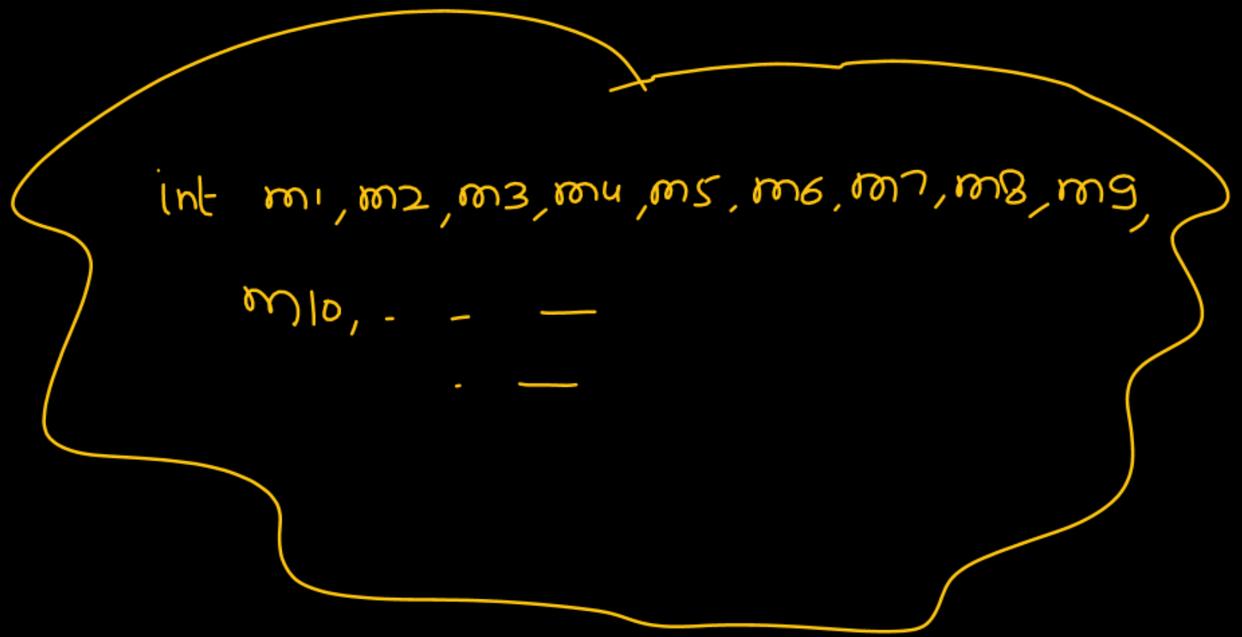


*( )

Address

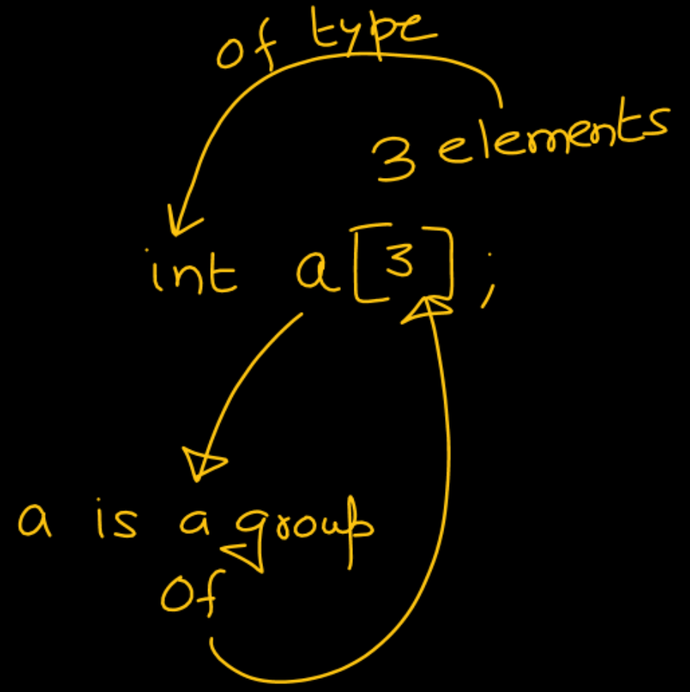
Why array ?

```
#——
Void main(){
    int m1,m2,m3;
    float avg;
    ==
    ==
    ==
}
```



```
int a, b, c;
```

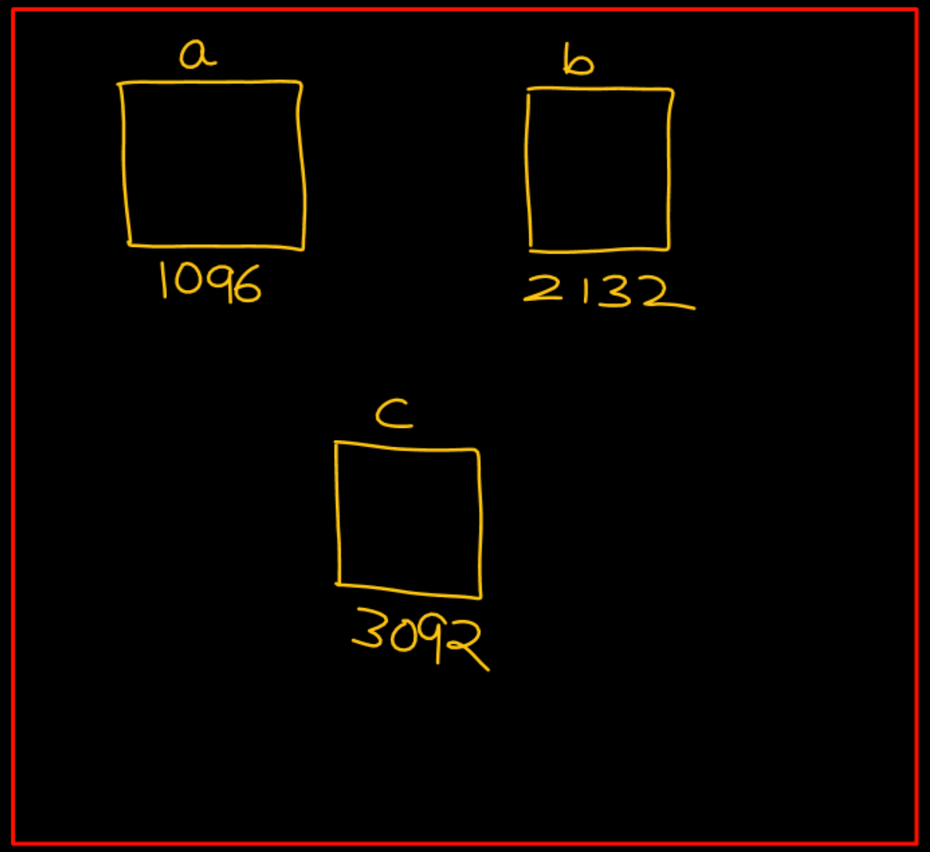
3 variable



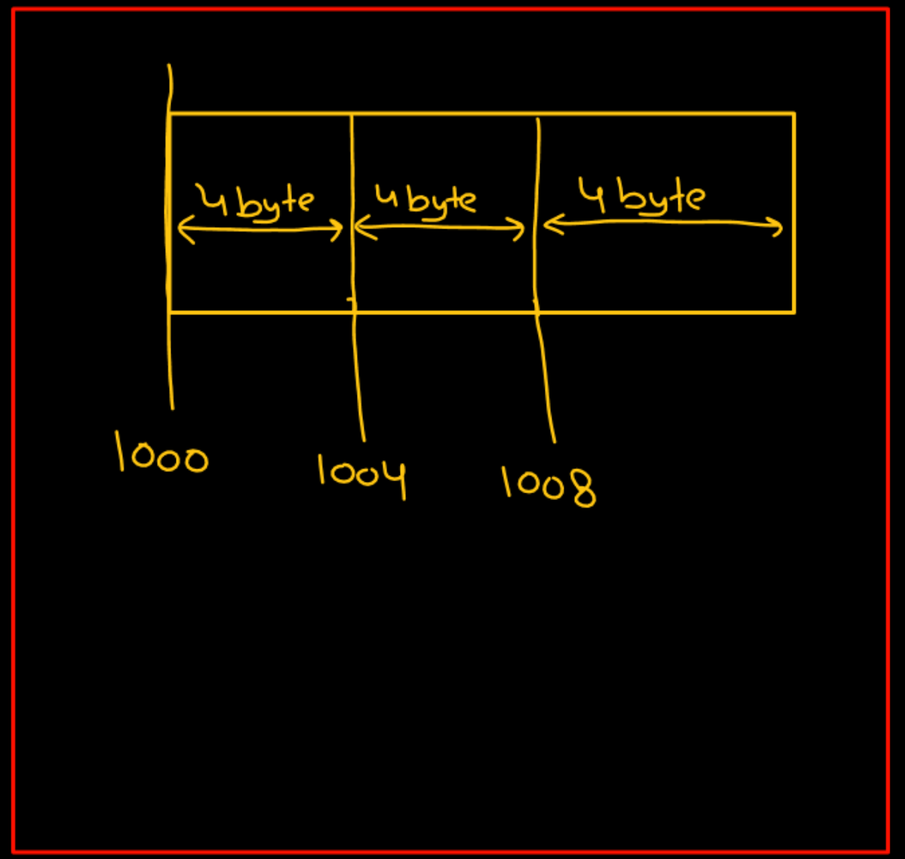

```
int a[50];
float b[100];
```

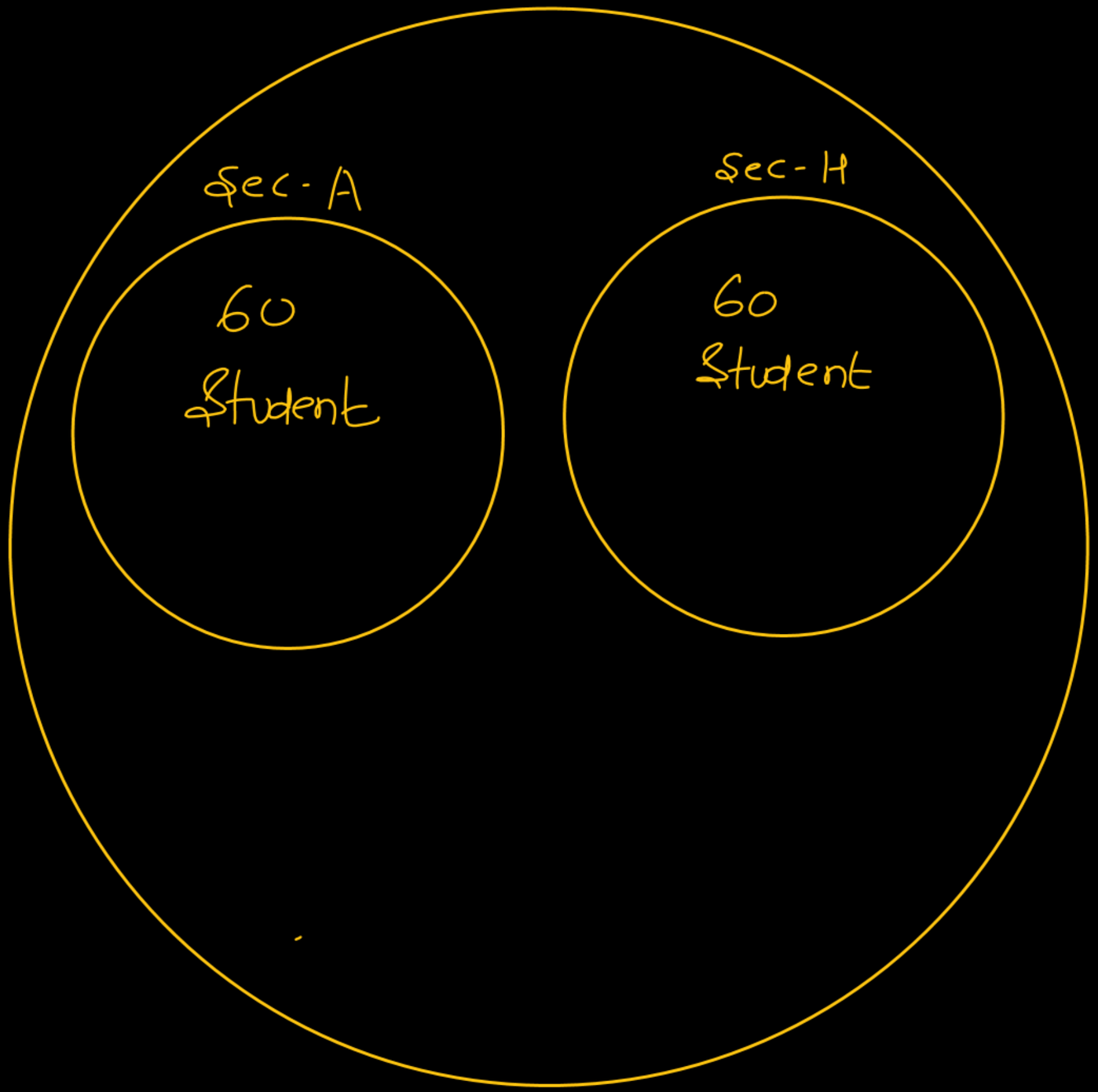
① Seq. stored ⇒ one after another

```
int a, b, c;
```

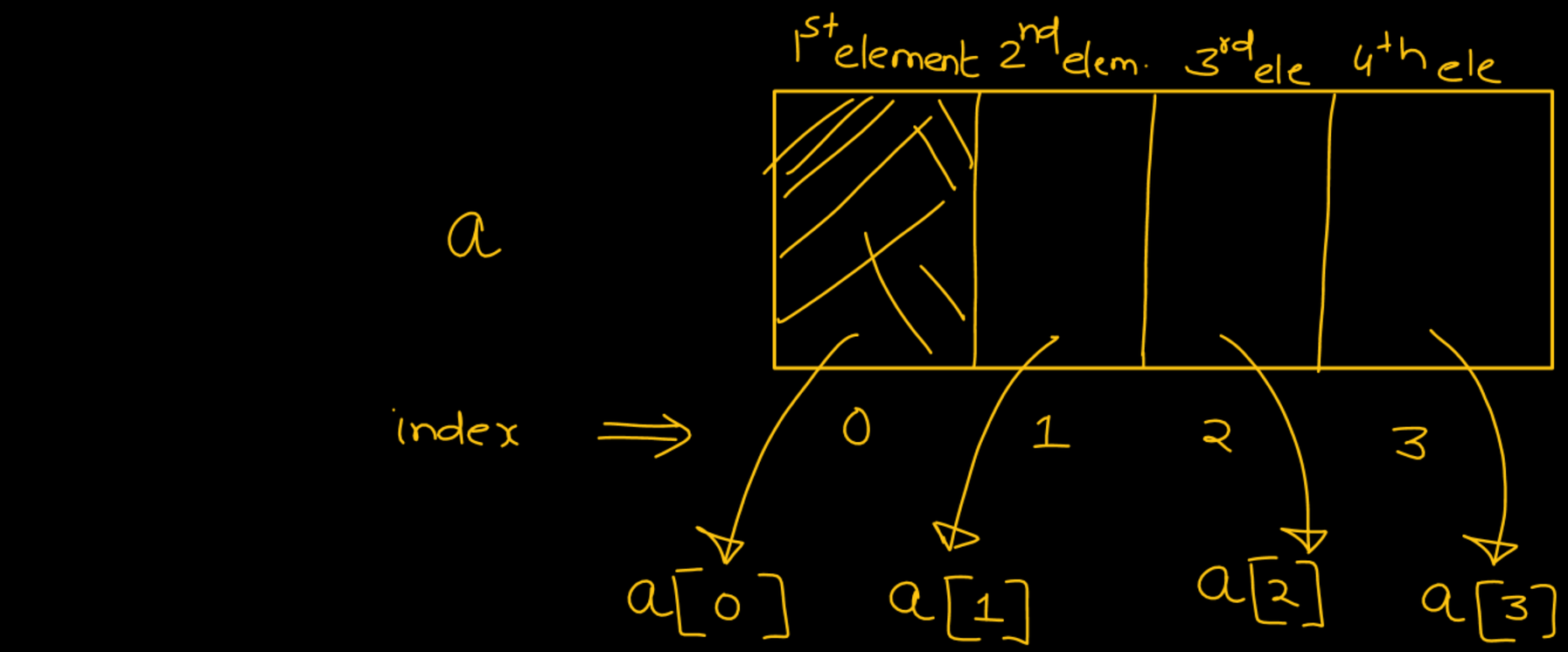



```
int a[3];
```



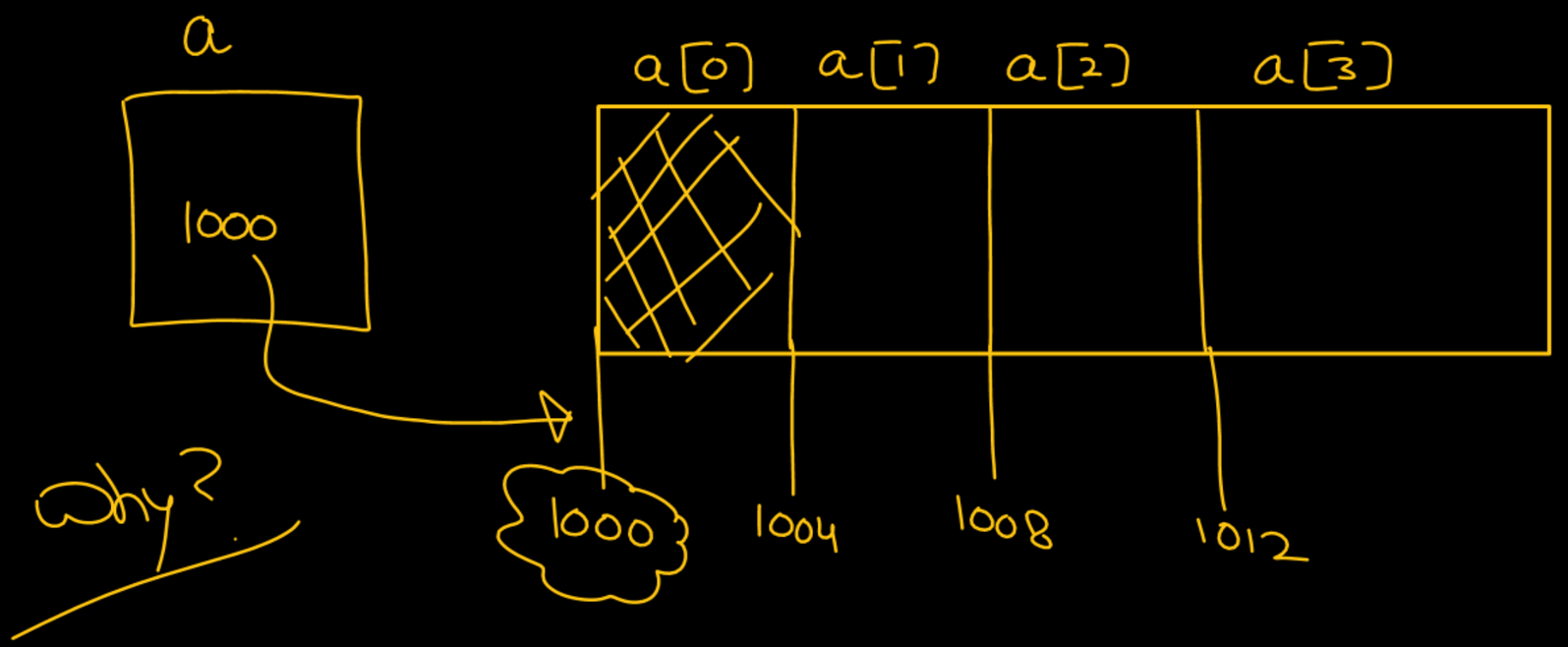

Sec - A
60
Student
Sec - H
60
Student

```
int a[4];
```

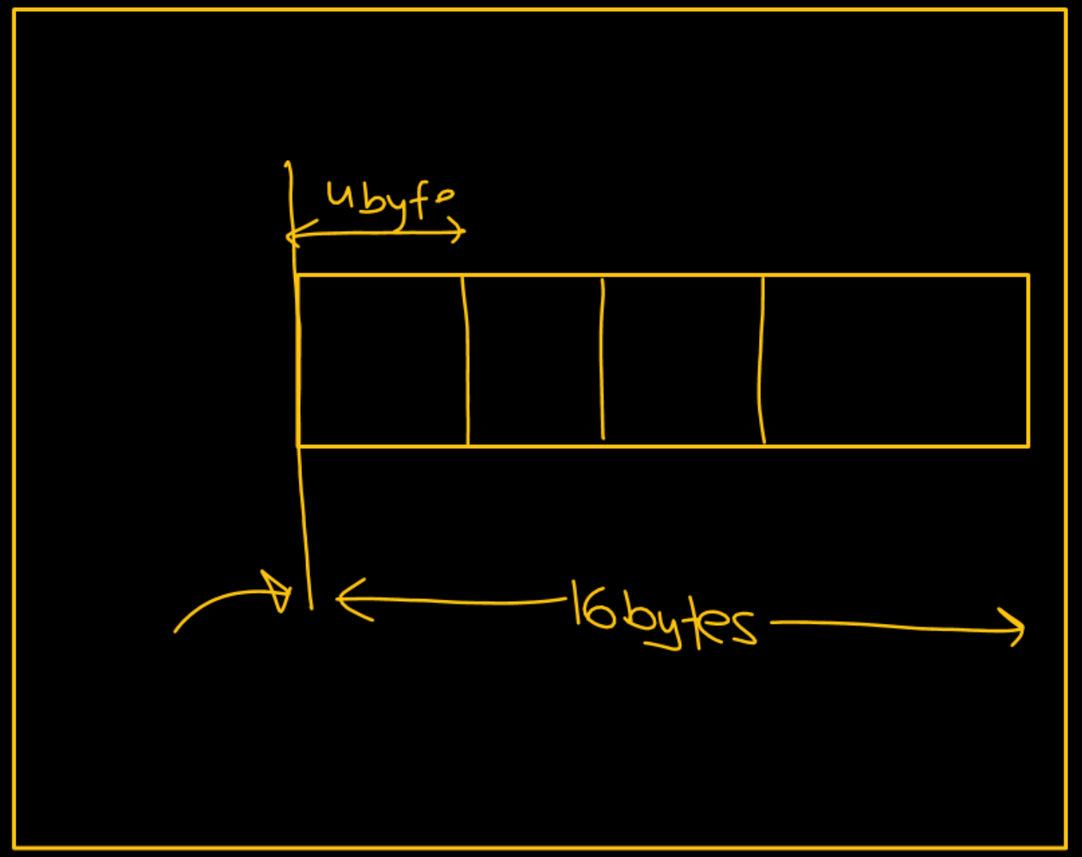


int a,b,c,d ;

int a[4] ;



```
int a[4];
```



a : Numerical value 100

a+1 ? ⟹ 101 ✗

What is a ?

simple var. / value ⟹ value + value = value

101 ✓

int a = 100;

a+1 ⟹ 101 ✓

address

add: asithm.

(address + Int: value ⟹ add)

1: Name of an array ⟹ represent address of its first element.

2: Name of an array ⇒ constant

Lvalue = Rvalue ;

2 = 3 ;

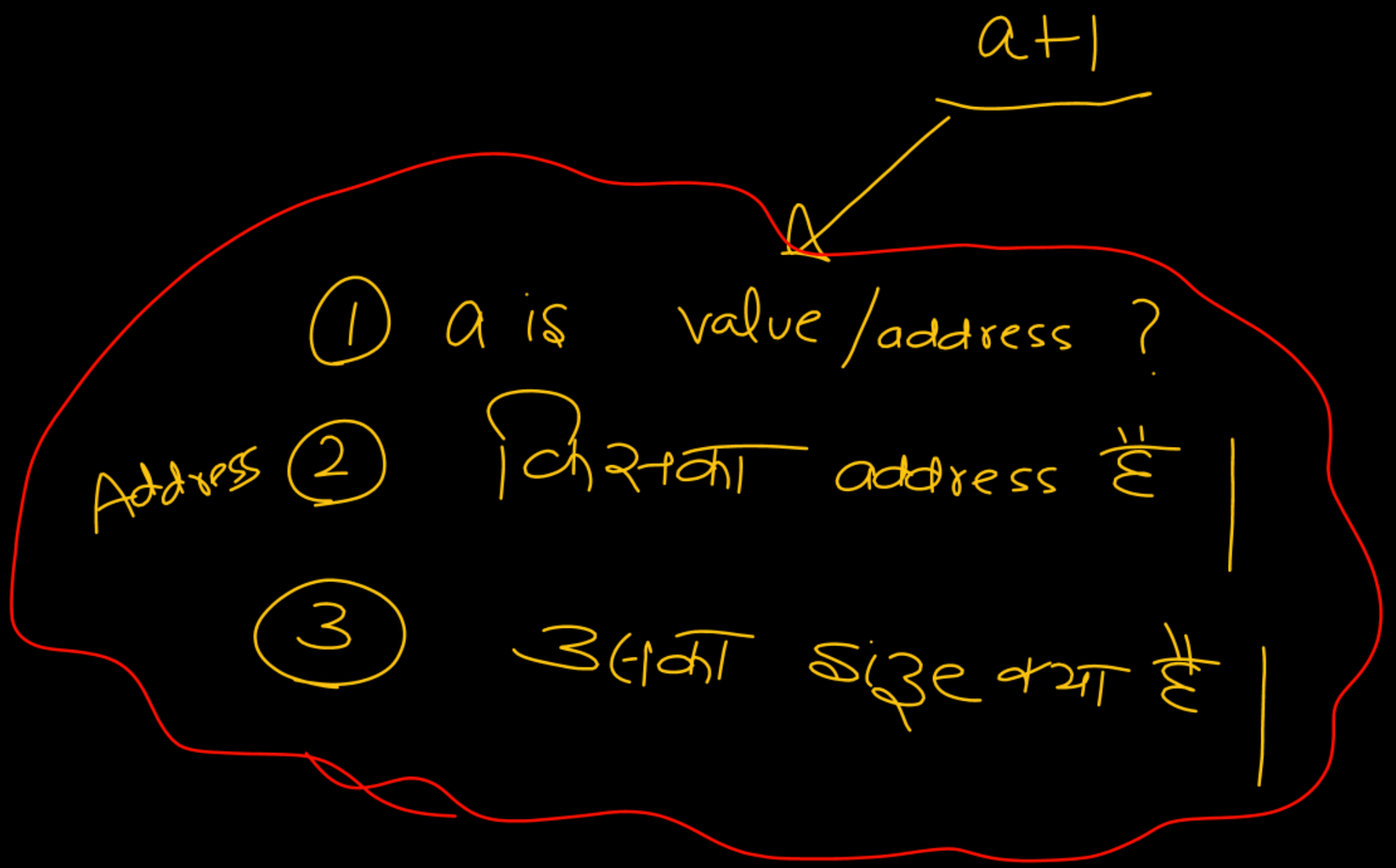
2 = a ;

Array_Name = ;

Lvalue

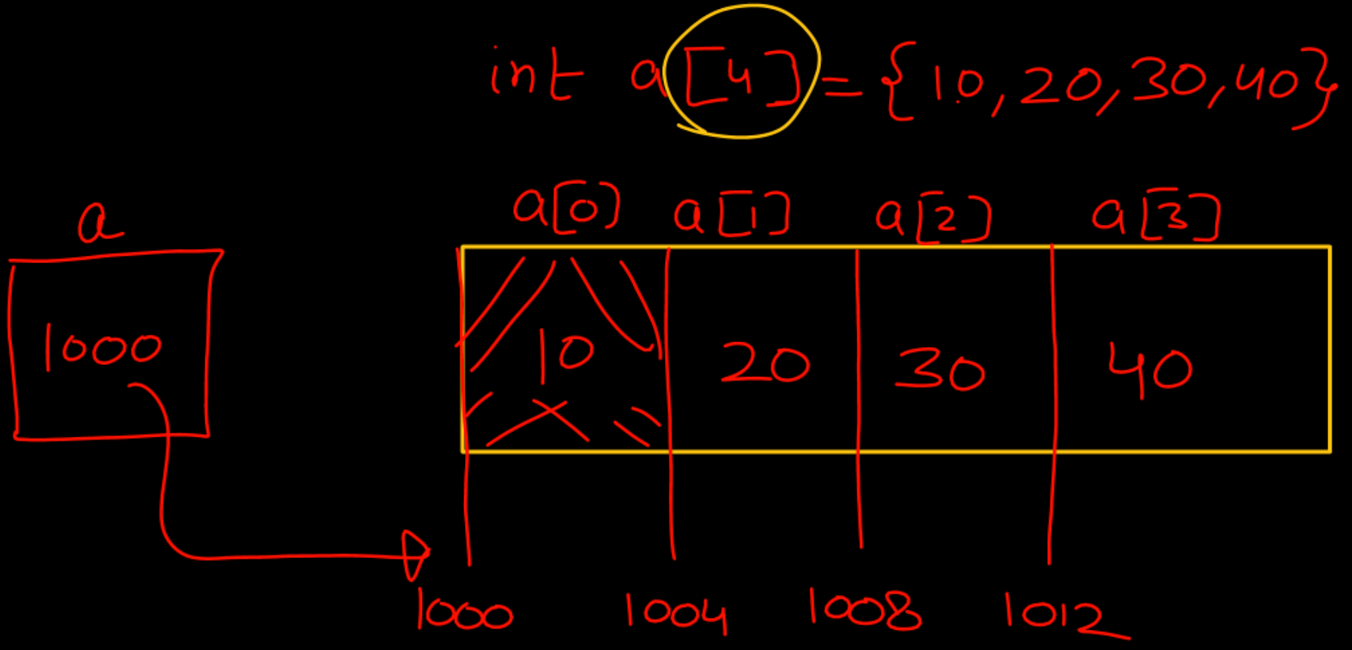
++2 ;
2++ ;
--3 ;
3-- ;
} Invalid

Array_name++ ;
++ Array_name ;
-- Array_name ;
Array_name-- ;
} Invalid


a+1
① a is value / address ?
Address ② किसका address है |
③ उसका size क्या है |

a[0] — ele
a[1] — ele
a[2] — ele
a[3] — ele

int a[4] = {10, 20, 30, 40}

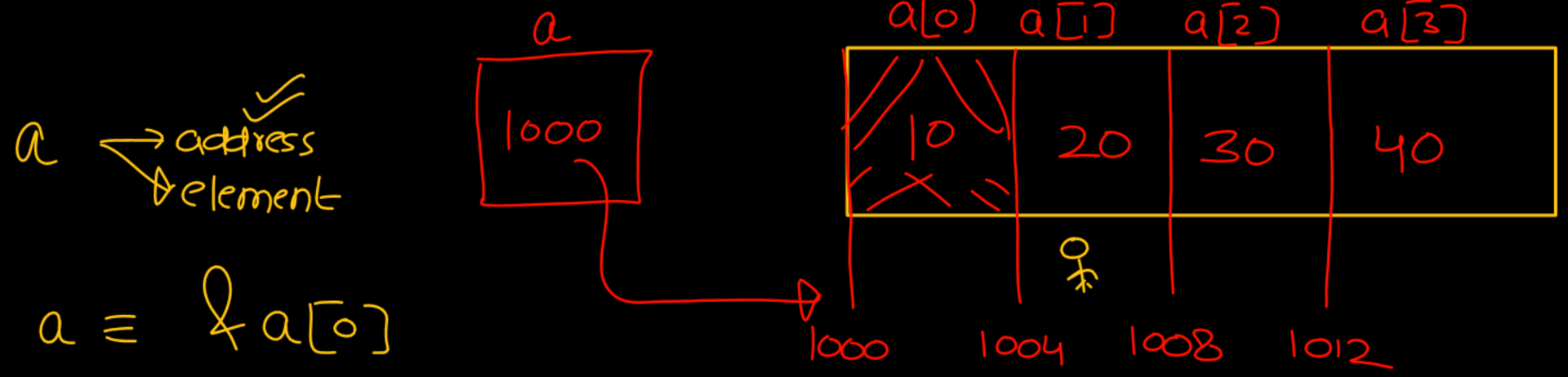


① decl. ⇒ n-dimension [ ]

n-dim ⇒ element

less than n-dim ⇒ address

int a[4] = {10, 20, 30, 40}



(i) $a \equiv \&a[0]$

a+1 ?

(i) a → address / element

(ii) जिसका address $a \equiv \&a[0]$

(iii) a[0] ⇒ 4 byte

⇒ $a+1 \Rightarrow \&a[0] + 1 \times 4$

⇒ 1000 + 4

⇒ 1004

(a+1) ⇒ Memory loc. 1004 ⇒ &a[1]

a+2 ⇒

a ⇒ addr

&a[0] ⇒ 4 bytes

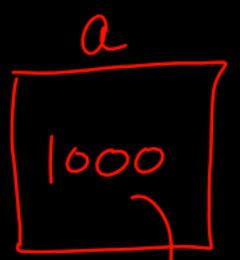
a+2 = &a[0] + 2×4

= 1000 + 8 = 1008

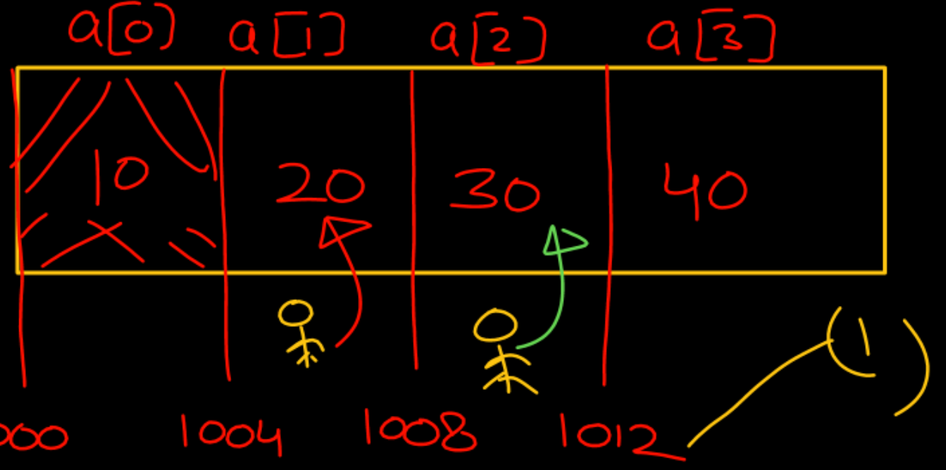
a+2 = (Mem. loc. 1008) = &a[2] —(1)

*(a+2) = val. at (Mem. loc. 1008) = *&a[2]

*(a+2) = 30 = a[2]



int a[4] = {10, 20, 30, 40}



(a+1) ⇒ Memory loc. 1004 ⇒ &a[1]

*(a+1) = value at (Mem. loc. 1004) = *&a[1]

*(a+1) = 20 = a[1]

100

$*(a+2) = 30 = a[2]$

$*(a+1) = 20 = a[1]$

$*(a+i) = a[i]$

```
int a[4] = {10,20,30,40};
printf("%d", *(a+2));
printf("%d", a[2]);
```

30

$a[i] = *(a+i) = a[i]$

$*(i+a) = i[a]$

Addition is commutative

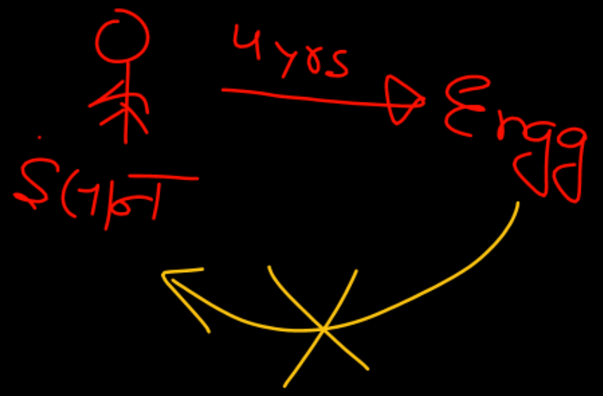
$$a[i] = *(a+i) = *(i+a) = i[a]$$

~~int 4[a] = {10,20,30,40};~~

int a[4] = {10,20,30,40};

```
printf("%d", a[2]);
printf("%d", 2[a]);
printf("%d", *(a+2));
printf("%d", *(2+a));
```

So

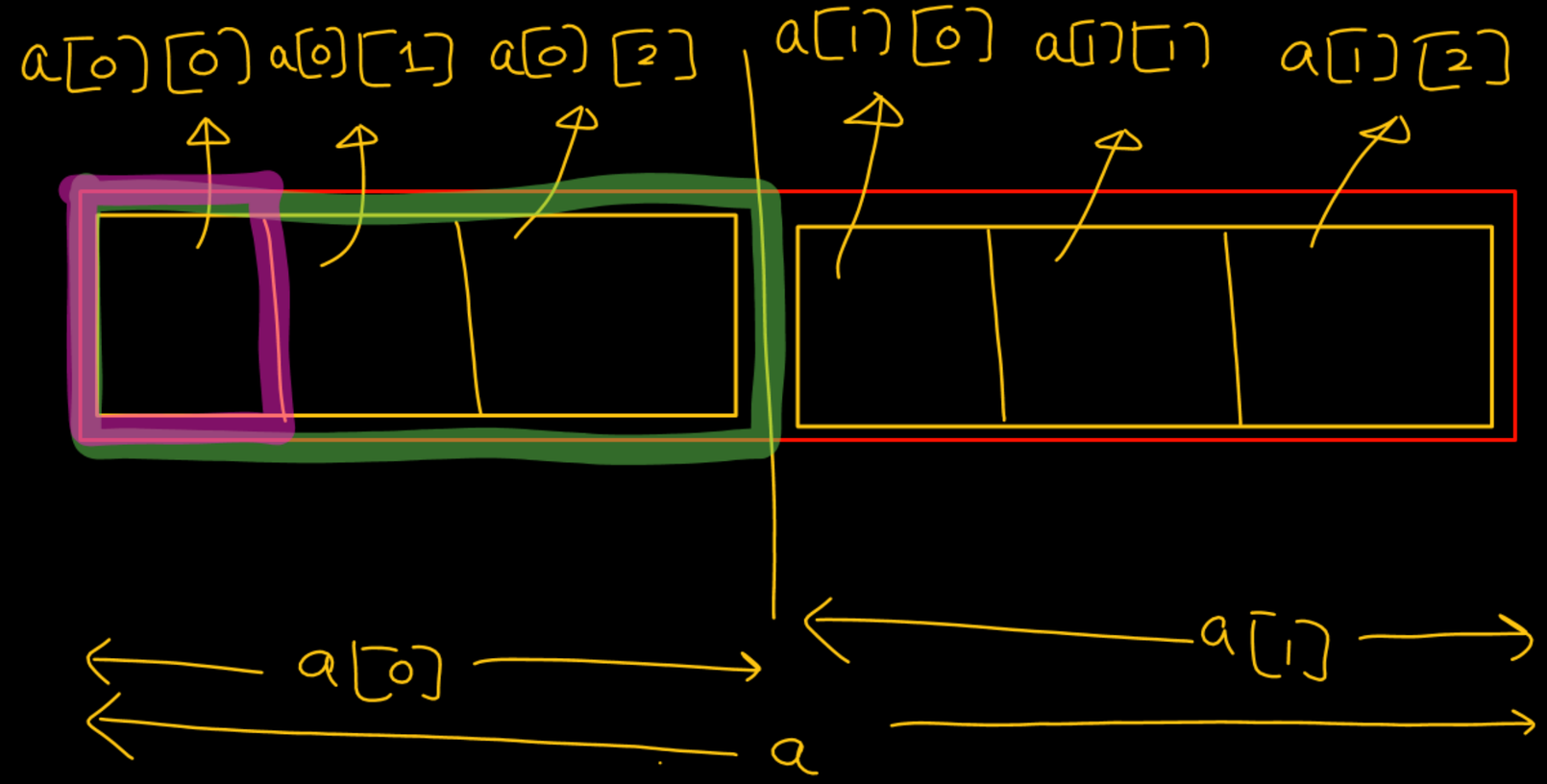


1st element of a ⇒ a[0] 12 bytes

1st element of a[0] ⇒ a[0][0]

2-D array

int a[2][3] = {1,2,3,4,5,6};

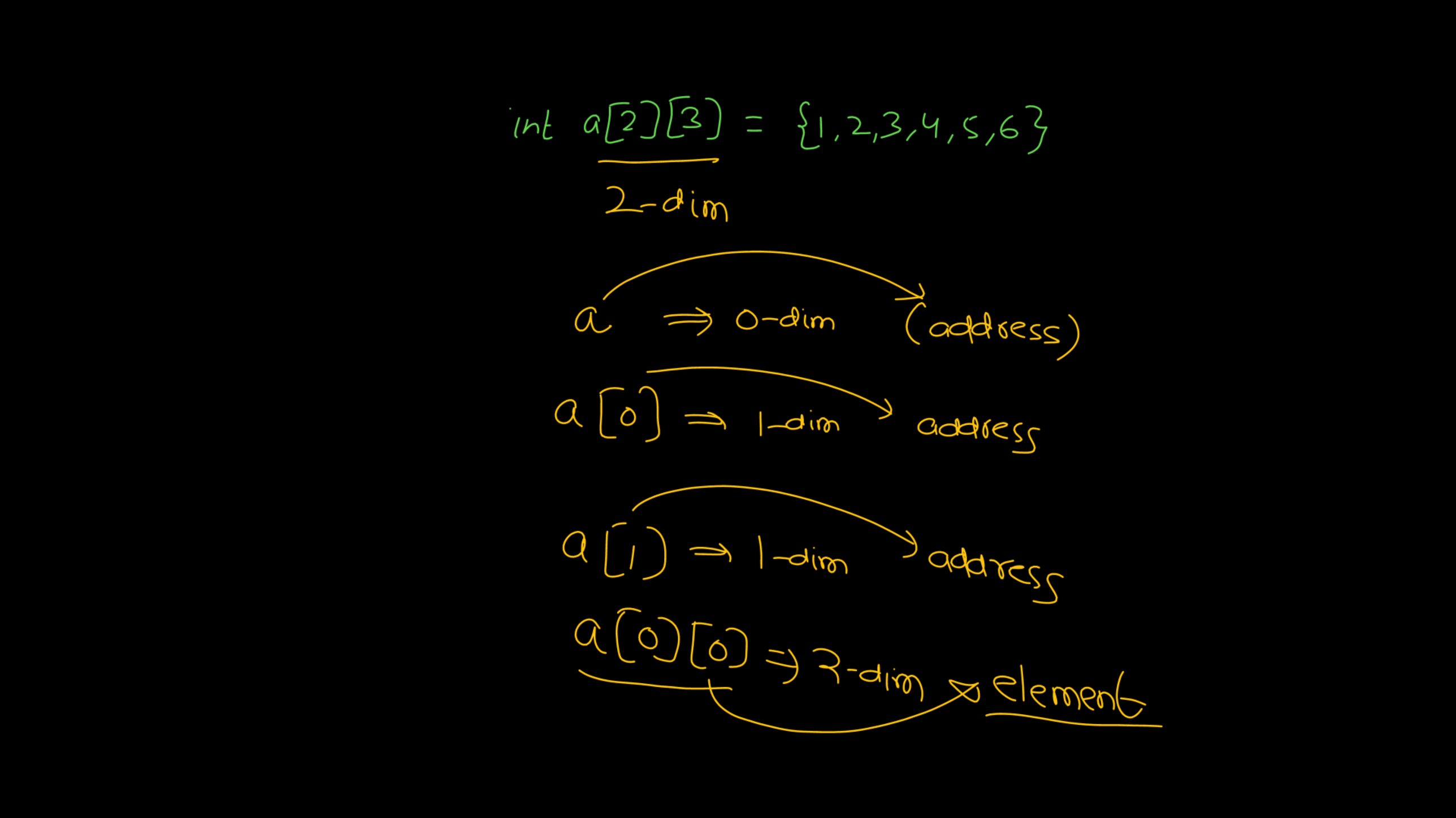


① array name ⇒ 1st element address

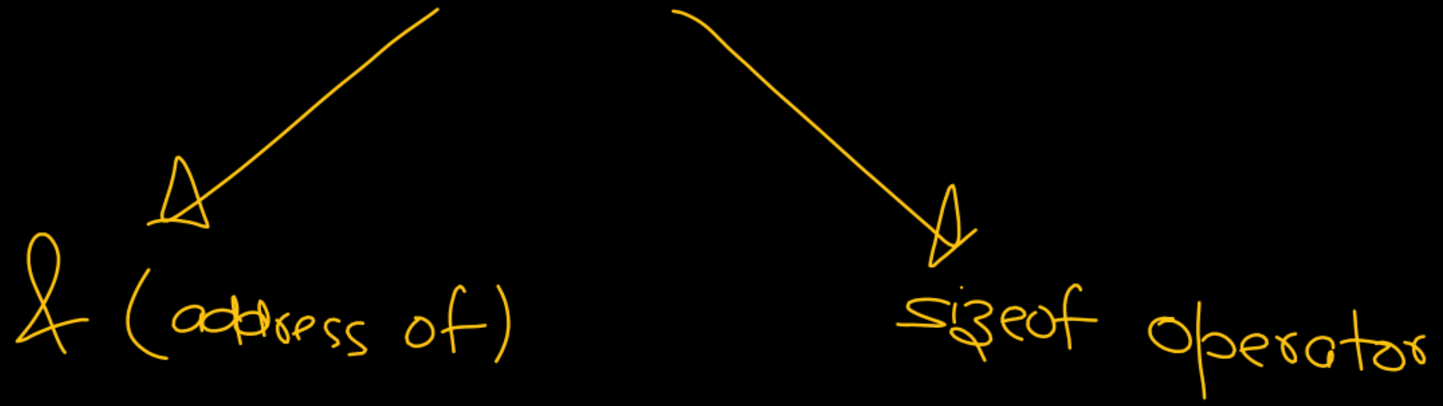
② array dec ⇒ n-dimension

a) Anywhere other than dec if we provide exactly n-dimension
⇒ then we are working on elements

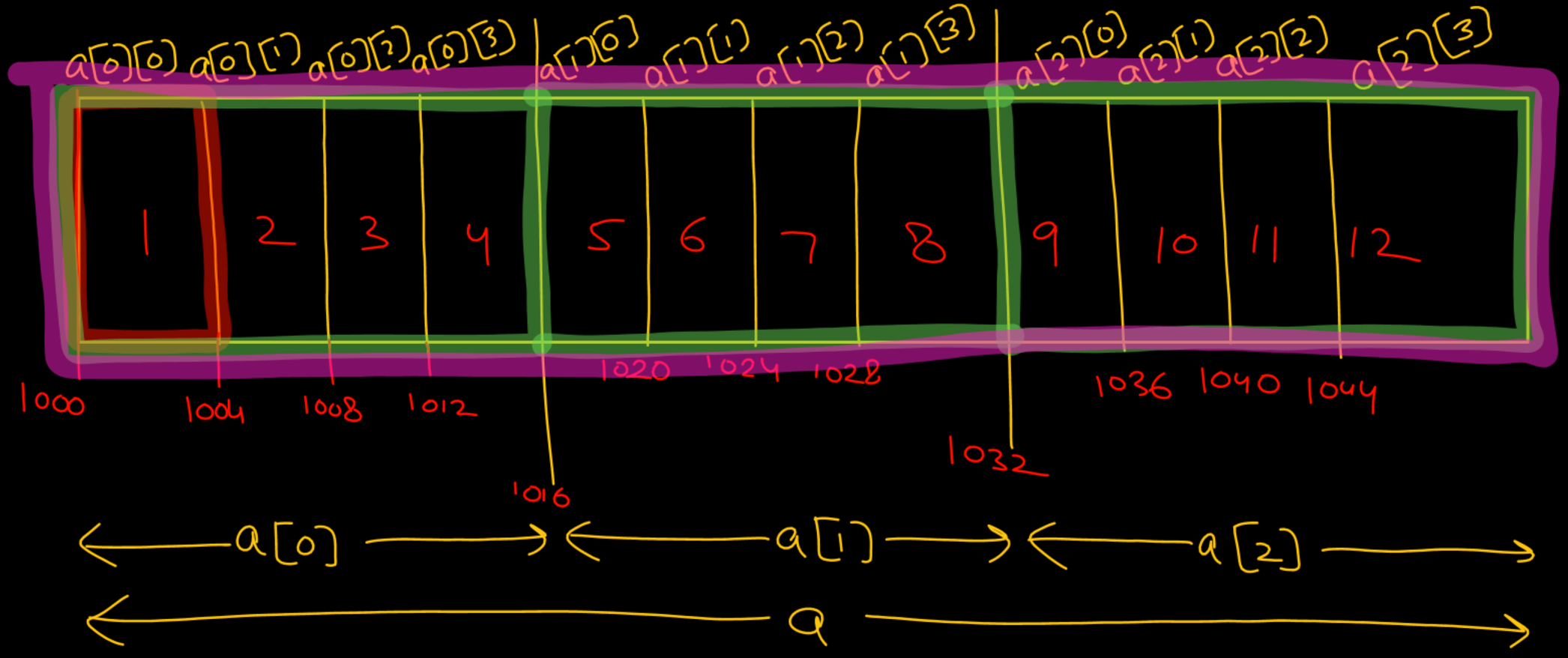
b) if we provide less than n-dim
⇒ addresses


int a[2][3] = {1,2,3,4,5,6}
2-dim
a ⇒ 0-dim (address)
a[0] ⇒ 1-dim address
a[1] ⇒ 1-dim address
a[0][0] ⇒ 2-dim element

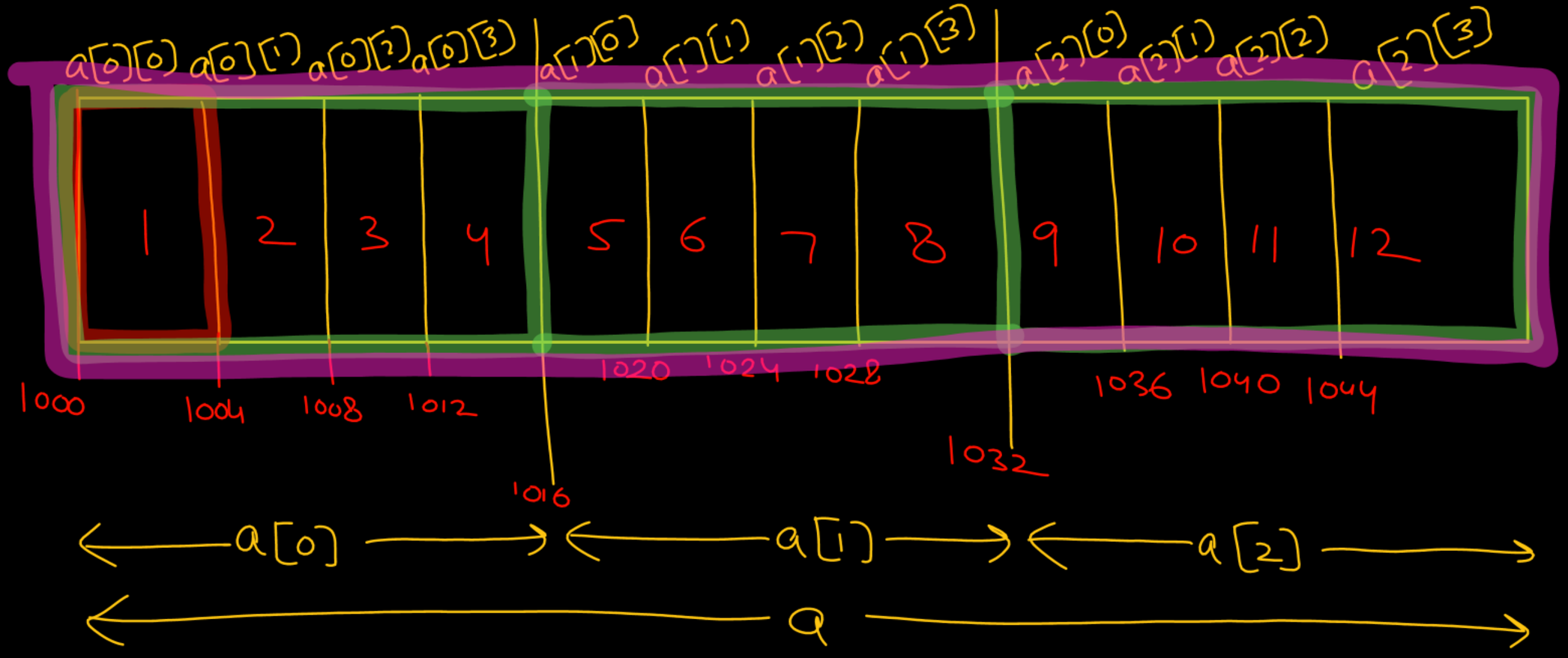
③ Name of an array does not represent an add. with 2 operators

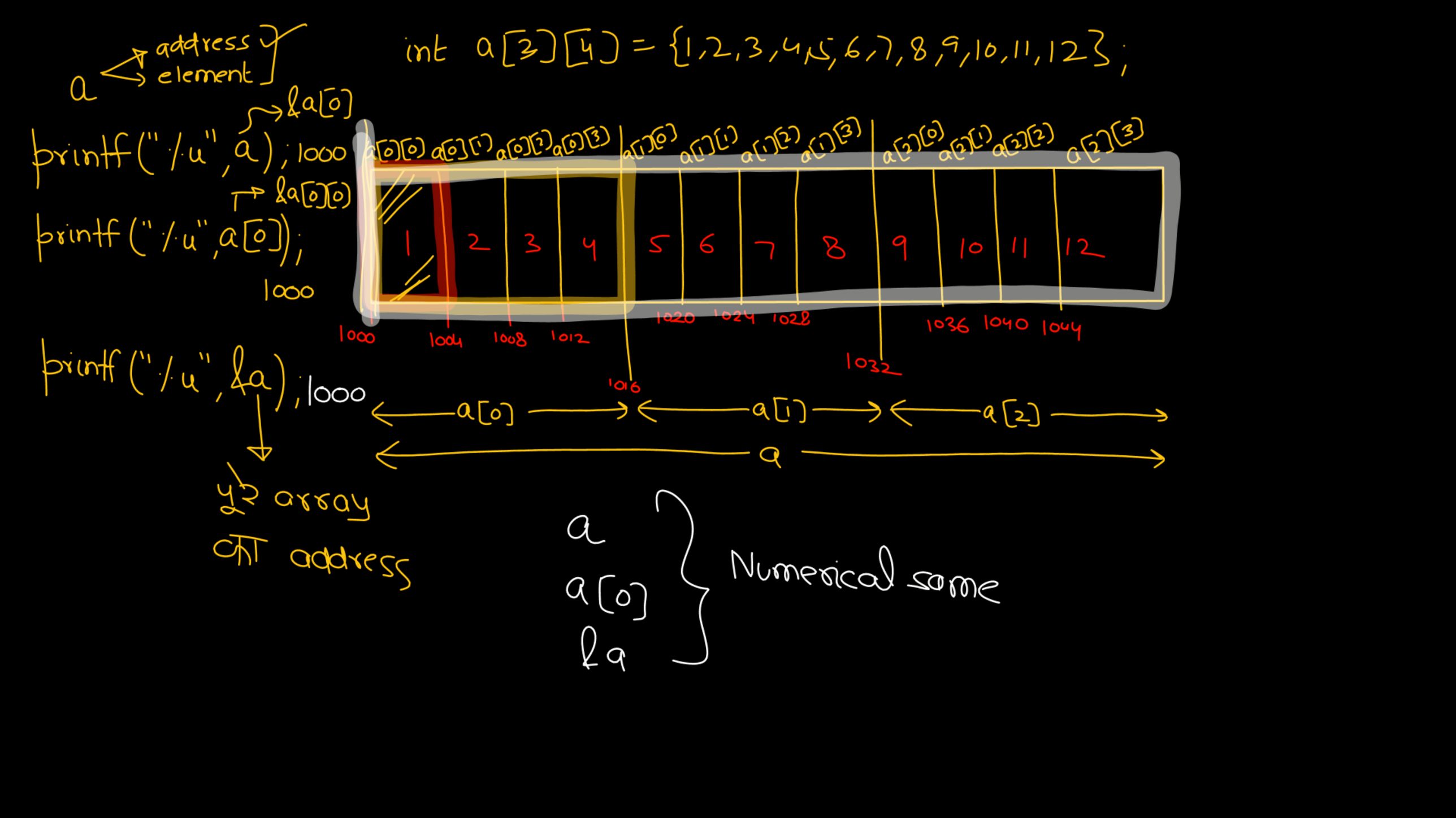


```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```



```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```




a
address
element
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
&a[0]
printf("%u", a); 1000
&a[0][0]
printf("%u", a[0]); 1000
printf("%u", &a); 1000
a[0][0] a[0][1] a[0][2] a[0][3] a[1][0] a[1][1] a[1][2] a[1][3] a[2][0] a[2][1] a[2][2] a[2][3]
1 2 3 4 5 6 7 8 9 10 11 12
1000 1004 1008 1012 1016 1020 1024 1028 1032 1036 1040 1044
a[0] a[1] a[2]
a
2-D array का address
a
a[0]
&a
Numerical same

```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

```
printf("%u", a+1);
```

1016

```
printf("%u", &a+1);
```

पूरे array $&a+1 \times 48$

$1000+48 = 1048$

```
printf("%u", a[0]+1);
```

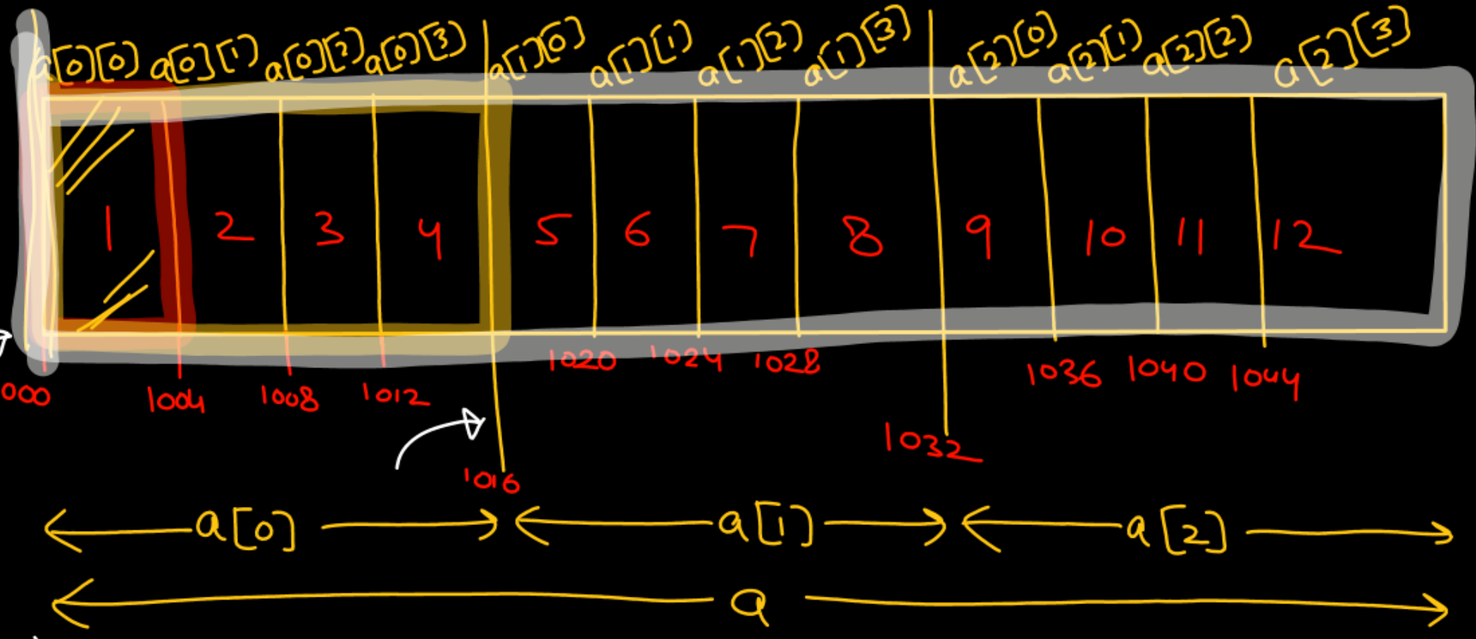
1004

$&a[0][0]+1$

4 byte

$= &a[0][0]+1 \times 4$

$1000+4 = 1004$



address + value

① किसका add. है

② size ?

@+1 $\Rightarrow &a[0]+1$

size = 16

$&a[0]+1 \times 16 = 1000 + 16 = 1016$

int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};

1) a+1 ?

a का मतलब – Address

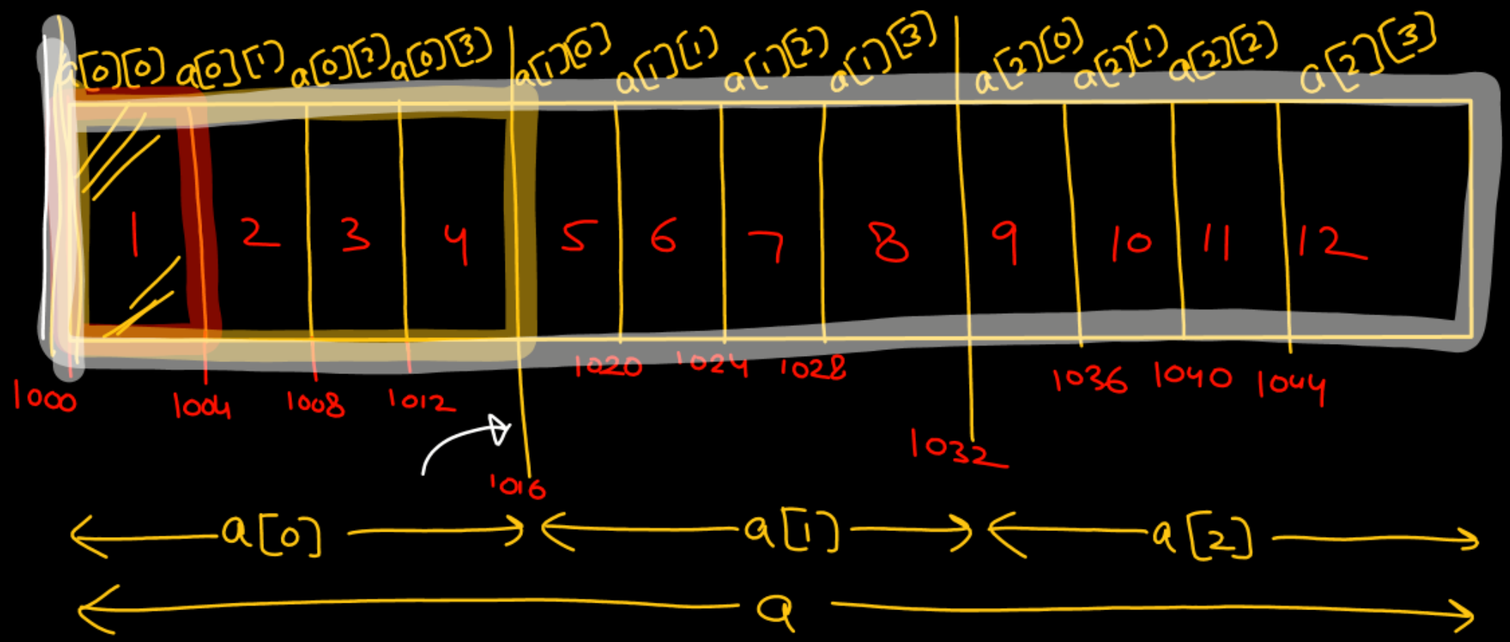
a ≡ &a[0]

a+1 ≡ &a[0]+1

&a[0]+1x16

1000+16

a+1 = 1016



a का first element ⇒ a[0]

a[0] का first element ⇒ a[0][0]

&a ⇒ Whole array (48 byte)

③ a[0]+1

a[0] ⇒ Address

&a[0][0]

a[0]+1 = &a[0][0]+1

&a[0][0]+1x4

⇒ 1004

⇒ &a[0][1]

② &a+1

Whole array size = 48

&a+1x48

1000+48 = 1048

L - 2, 3, 4

Chapter - 4

doubts

Arrays & pointer - 1

- 2
- 3
- 4
- 5
- 6
- 7
- 8

H.W

Strings - 1

Notice


THANK YOU GW
SOLDIERS !