CS & IT ENGINEERING

Programming in C

Arrays and Pointers 02

Lec- 02

By- Pankaj Sharma sir

Arrays-II

① Array-name → constant
→ Address of first element.

⟹ Invalid

++Array-name,
Array-name++,
Array-name--;
--Array-name;
Array-name = 

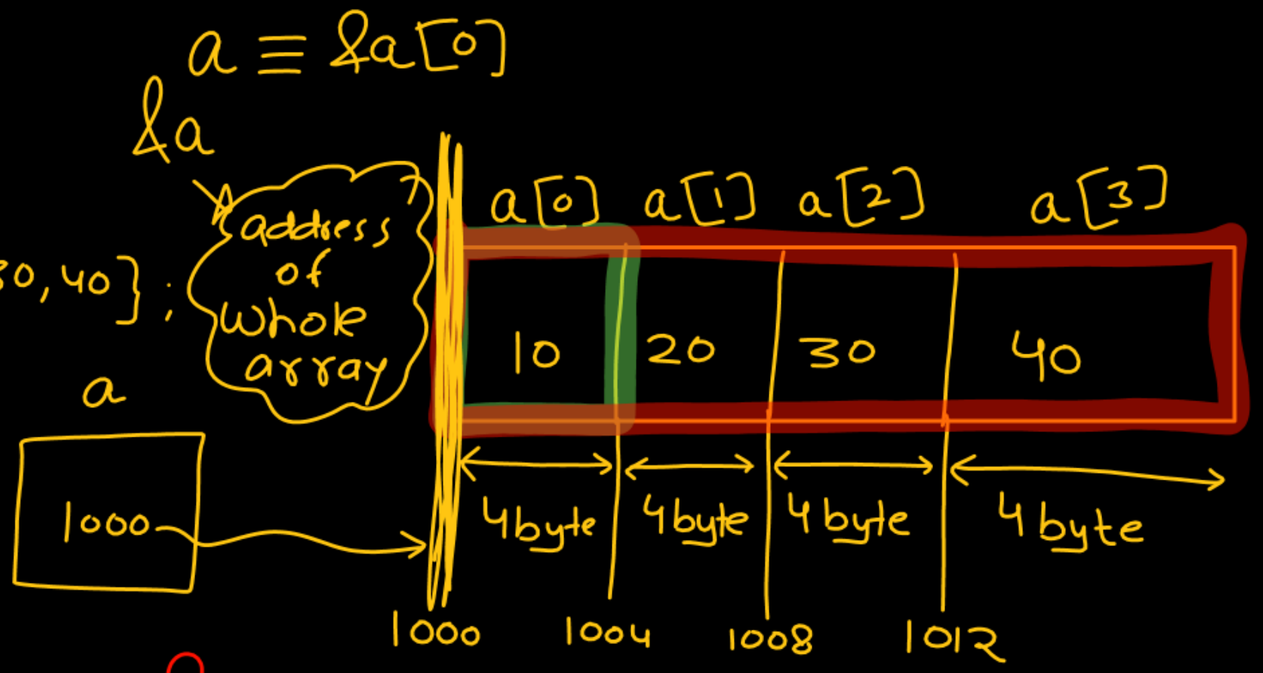
② Relative addressing

```
void main(){
    int a[4] = {10,20,30,40};
    printf("%u", a);
    printf("%u", &a[0]);
}
```

1000

1000

$a \equiv \&a[0]$



$a \equiv \&a[0]$

(3) Name of array does not represent address with 2 operators: & and sizeof

$\&a \Rightarrow$ address of whole array

$\&a[0]$
$\&a[1]$
$\&a[2]$
$\&a[3]$

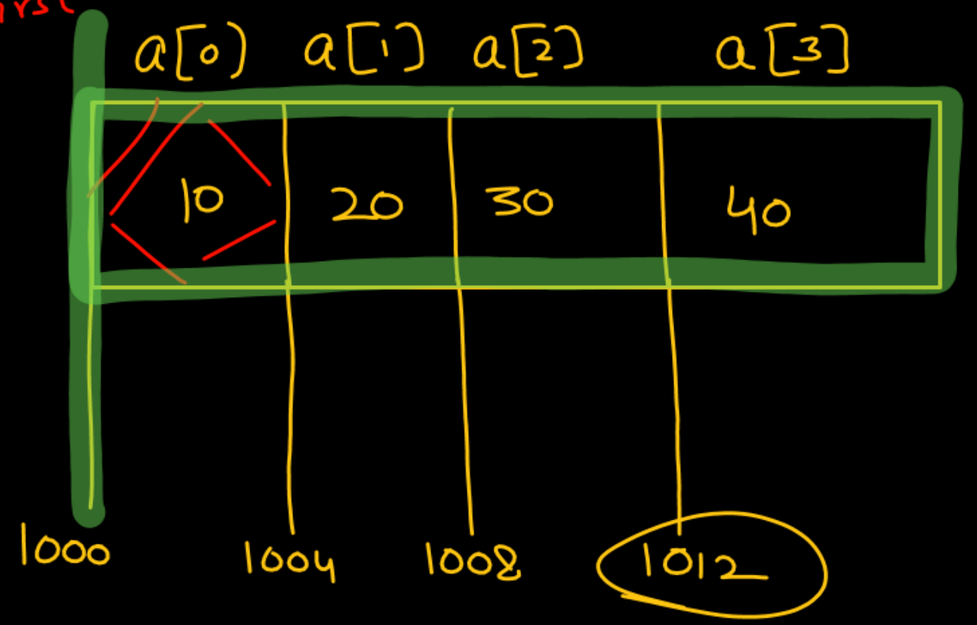
4byte ⇒

16 byte ⇒ Whole array

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

```
printf("%u", a);   → 1000   (add. of first ele.)
printf("%u", &a);  → 1000
printf("%u", &a[0]); → 1000
```

Numerical value same



(4) (a) : Numerical value of a is 100

a+1 ?

value ← Simple var.

a ⇒ address ∃
⇒ pointer var. ∃

```
int a = 100;
printf("%d", a+1);
```

101

Value + value = value
arithmetic

Addresss. arith.

[address + value ⇒ address
value + address ⇒ address]

address + adress ⇒ Invalid

(5.) If declaration of array is having n-dimensions and

(i) Anywhere in prog. , u provide exactly n-dimension then it is an element.

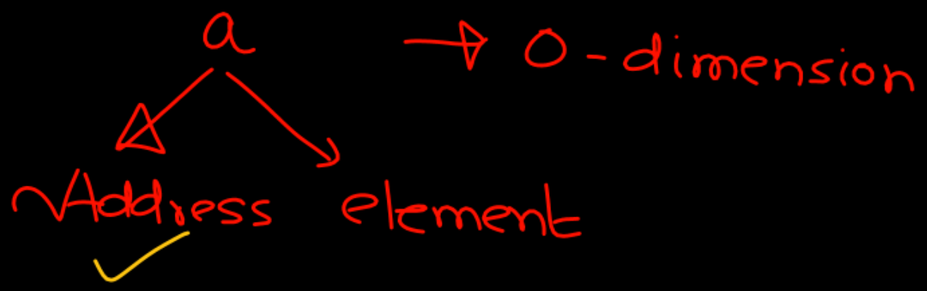
(ii) Anywhere in prog, u provide less than n-dim. then it is an address.

1-dimension

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

=

=



a[0] → element

a[1] → ele

a[2] → ele

2-dimension

```
int a[2][3] = {1,2,3,6,7,8};
```

a → Address ✓ (0-dimension)
a → element X

a[0] → Address ✓ (1-dimesion)
a[0] → element X

a[1] → Address (1-dim.)

a[1][1] → element

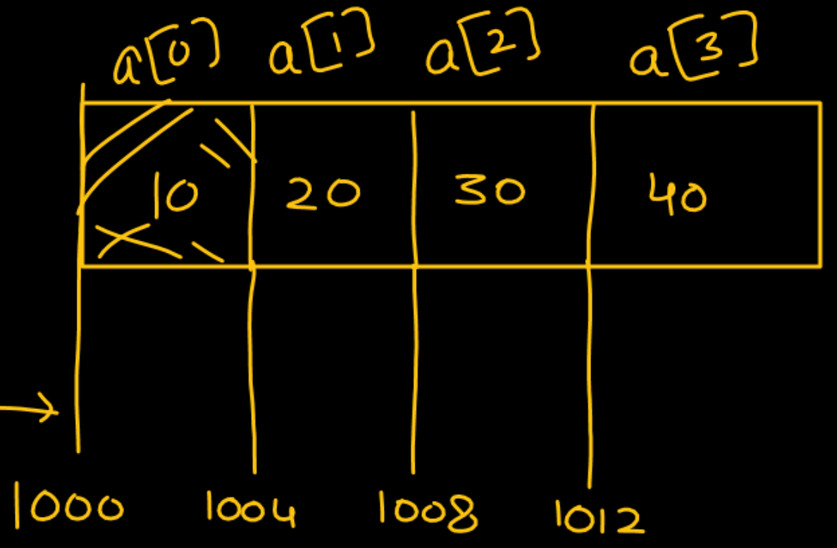
a + 1

1) What is a? → value → dance
                → Address

2) किसका address है | (Whose address we are talking about)

3) उसका size क्या है | (Find the size of that Object)

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



a+1

(i) a → address ✓
a → element

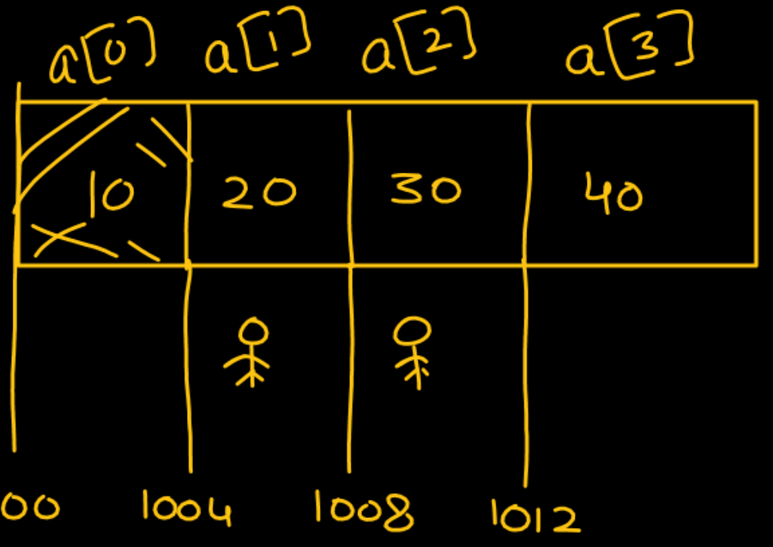
(ii) Which object is pointed by a.
Whose address is rep. by a.

&a[0]

address of a[0]

(iii) size of a[0] = 4bytes

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



a+1

$\Rightarrow$ &a[0] + 1 X 4

$\Rightarrow$ 1000 + 4 $\Rightarrow$ 1004

a+1 $\equiv$ (Memory location 1004) $\equiv$ &a[1]

a+2 $\equiv$ (Memory location 1008) $\equiv$ &a[2]

a+2 → (i) a → Address ✓ / element x

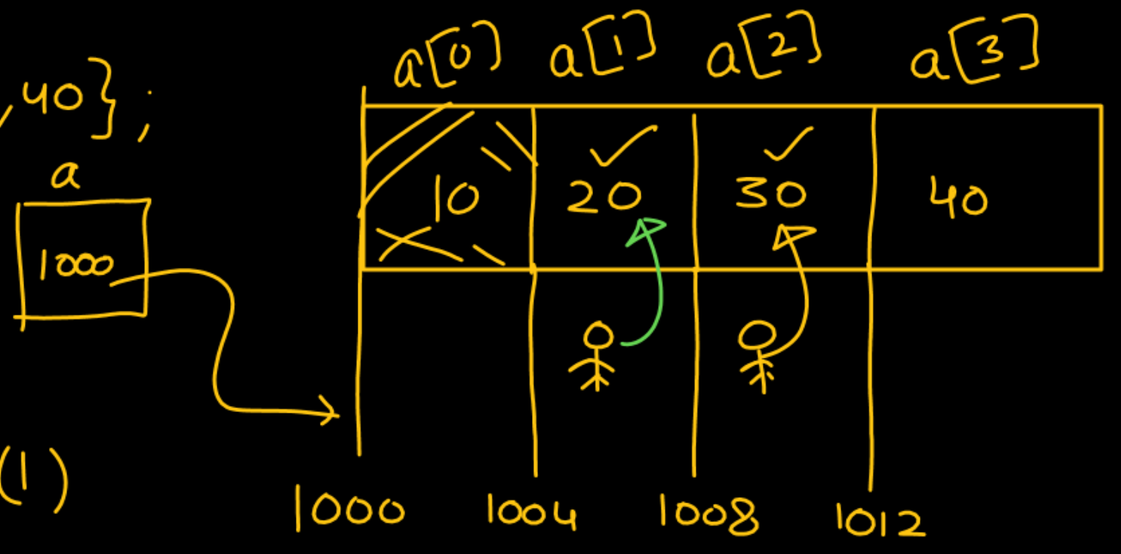
(ii) a = &a[0]

(iii) size of a[0] = 4 byt

&a[0] + 2 X 4

1000 + 8 = 1008

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



$a+1 \equiv (\text{Memory location } 1004) \equiv \&a[1]$ —(1)

$a+2 \equiv (\text{Memory location } 1008) \equiv \&a[2]$ —(2)

$*(a+1) \equiv \text{value at}(\text{Memory location } 1004) \equiv *\&a[1]$

$*(a+2) \equiv \text{value at}(\text{Memory location } 1008) \equiv *\&a[2]$

$*(a+1) \equiv 20 \equiv a[1]$ —(3)

$*(a+2) \equiv 30 \equiv a[2]$ —(4)

$*(a+i) \equiv a[i]$

```
void main(){
    int a[4] = {10,20,30,40};
    printf("%d",a[1]);
    printf("%d",*(a+1));
}
```

Same

20

Address + val ⇒ Address

val + Address ⇒ Address

$2+3 \iff 3+2$

(Addition commutative

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

$$= i[a]$$

```
Void main(){
    int a[4] = {10,20,30,40};
    printf("%d",a[1]);
    printf("%d",1[a]);
    printf("%d",*(a+1));
    printf("%d",*(1+a));
    }
```

declaration
↳ compiler info
Save

Invalid

Invalid

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

Rules/Naming

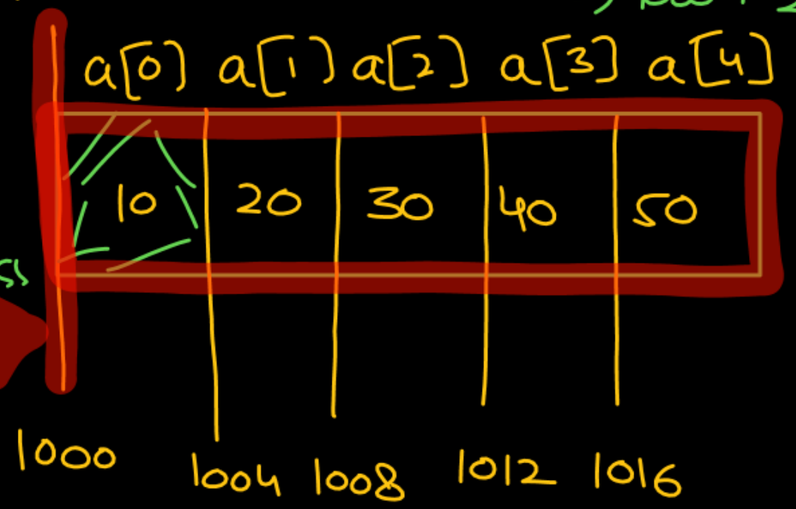
```
data_type Id_name[size] ;
```

digit X

```
int 4[a]
```

-, alphabet

```
void main(){
    int a[5] = {10,20,30,40,50};
    printf("%u", a);      // 1000 ← address &a[0]
    printf("%u", &a);     // 1000 ← पूरे array का address
    printf("%u", a+1);    // 1004 ← &a[1]
    printf("%u", &a+1);   // 1020
    printf("%u", *(a+1)); // 20 ⇓ a[1]
}
```



&a+1 ⇒ (i) &a ↗ add ✓ ↘ elem.

(ii) whole array ⇒ add

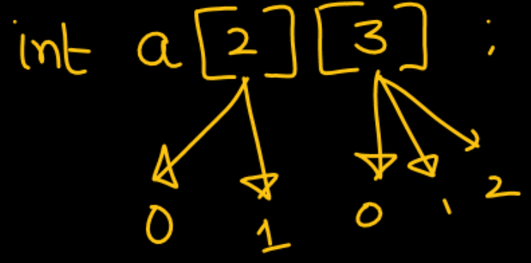
(iii) 20 bytes &a+1×20 ⇒ 1000+20 ⇒ 1020

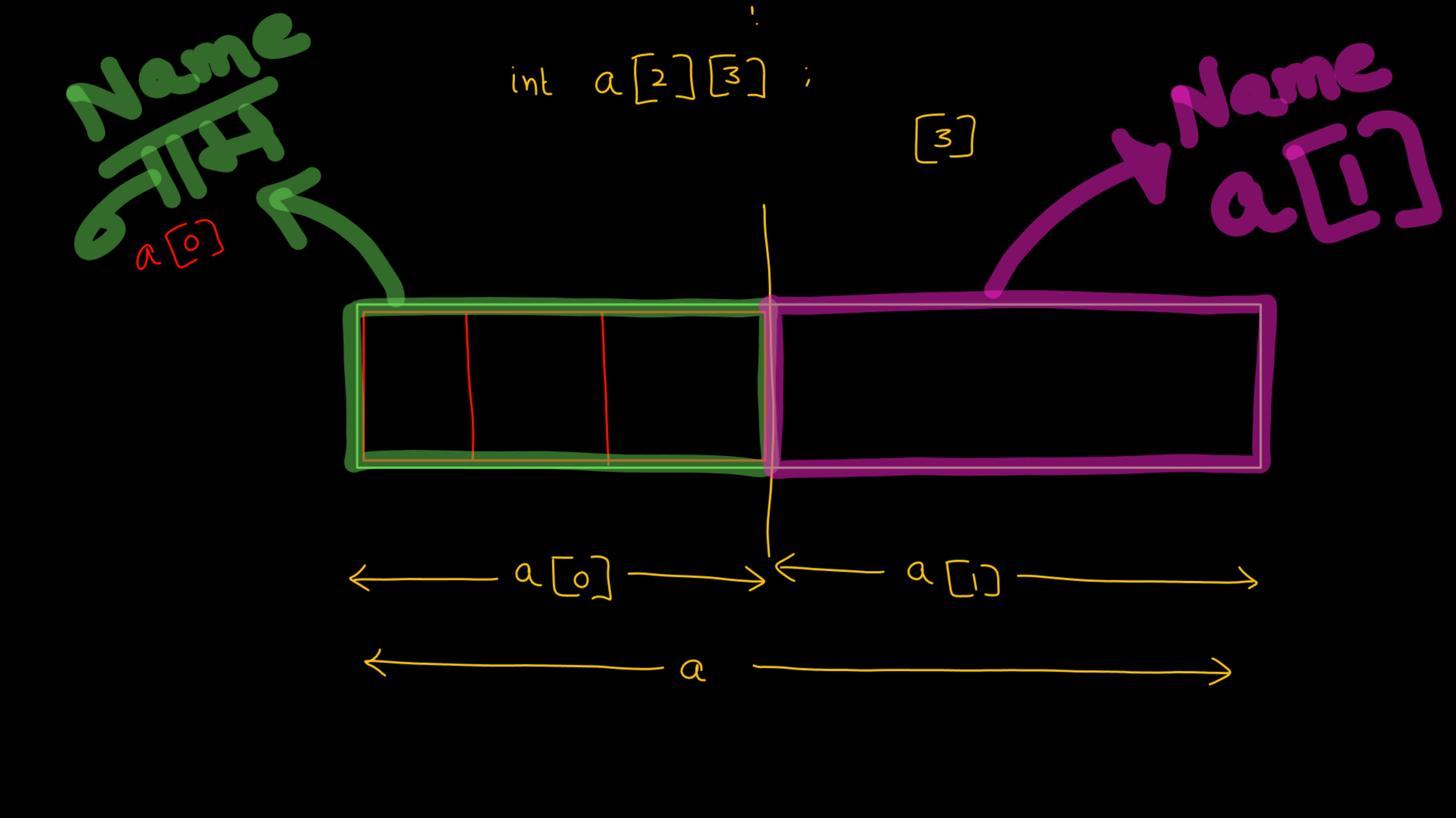
a+1 → (i) a ? ↗ add ✓ ↘ ele

&a[0]+1×4 (ii) &a[0] (iii) a[0] का size ⇒ 4 byte

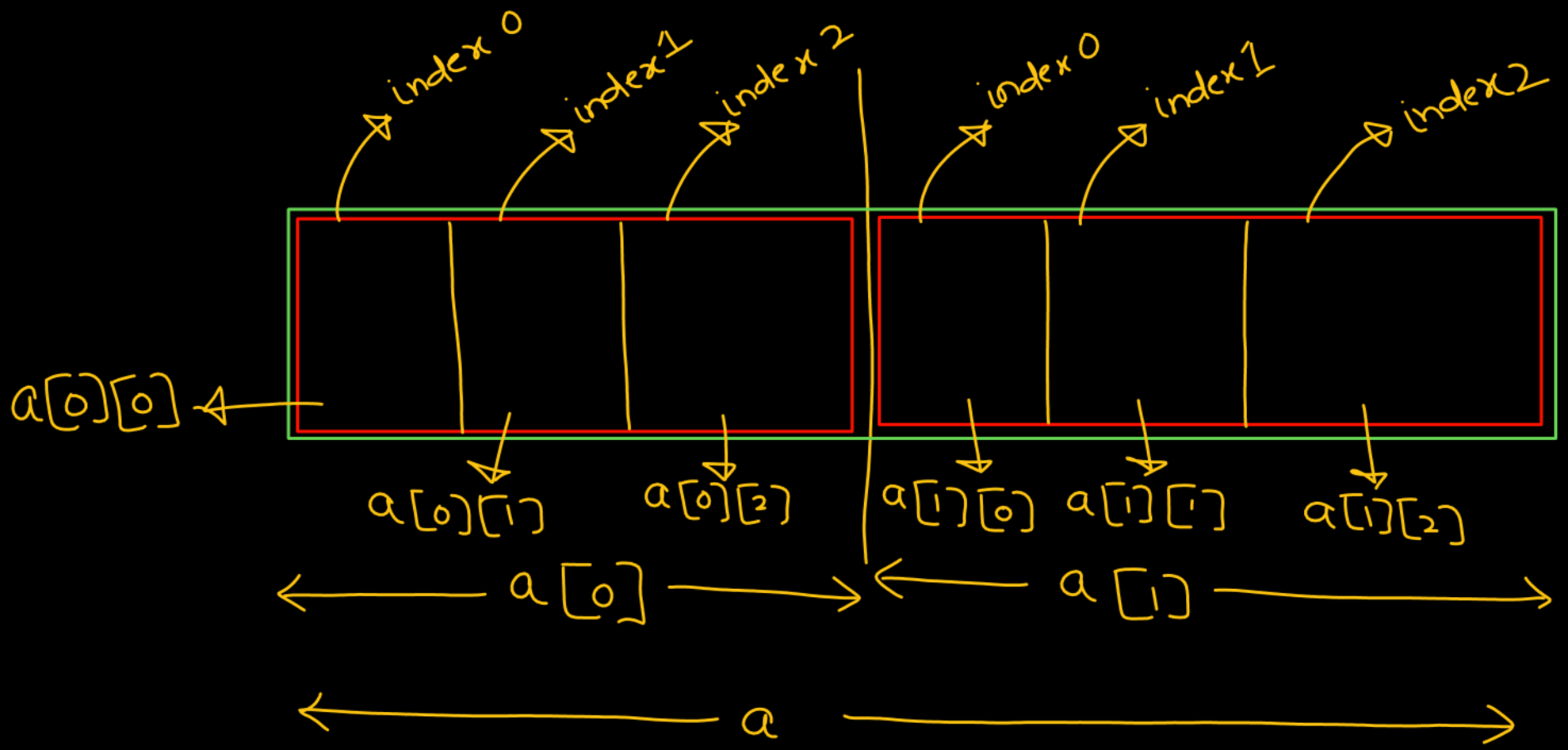
1000+4 ⇒ 1004

2-D array



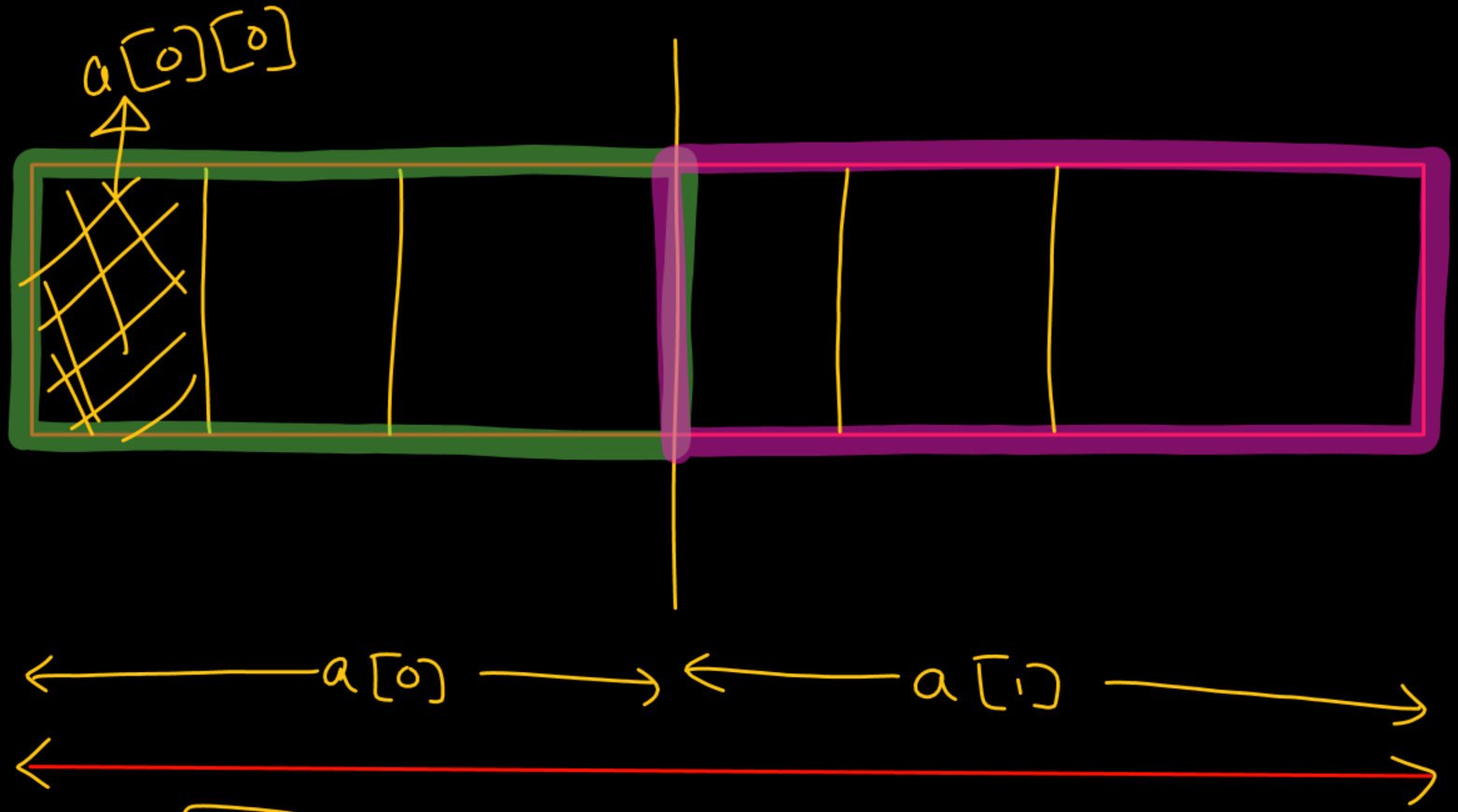

int a[2][3];
Name
a[0]
[3]
Name
a[1]
a[0]
a[1]
a

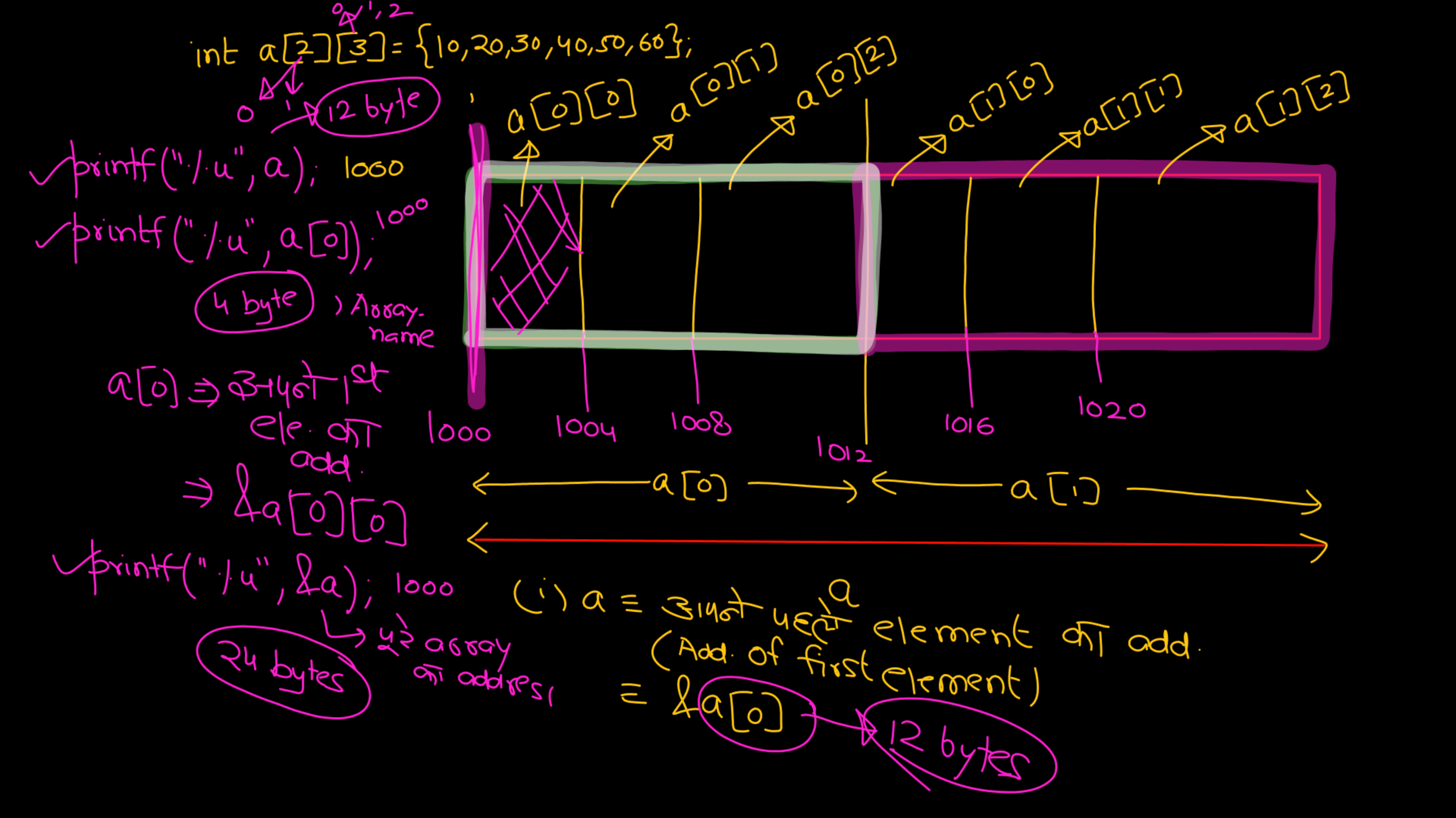
int a[2][3];


index 0
index 1
index 2
index 0
index 1
index 2
a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]
a[0]
a[1]
a

int a[2][3] = {10,20,30,40,50,60};

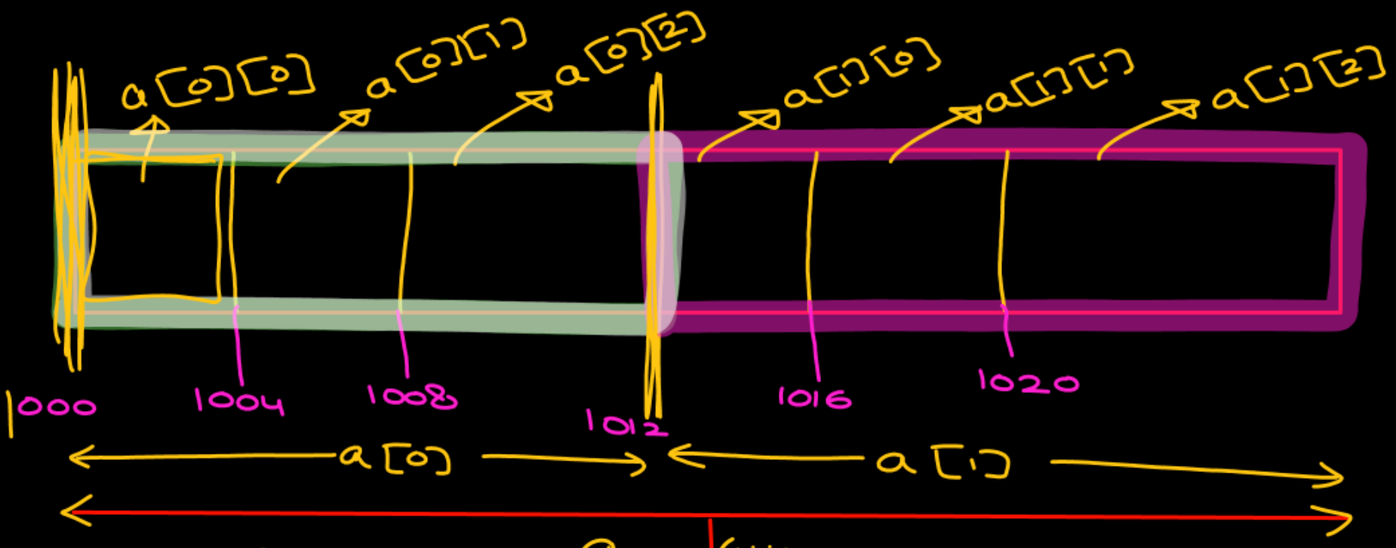
1. a ≡ &a[0]
2. a[0] ≡ &a[0][0]
3. &a ⇒ Add. of Whole array (24 byte)




int a[2][3] = {10,20,30,40,50,60};
0,1,2
0
1
12 byte
printf("%u", a); 1000
printf("%u", a[0]); 1000
4 byte
Array name
a[0] ⇒ अपने 1st ele. का add.
⇒ &a[0][0]
printf("%u", &a); 1000
24 bytes
पूरे array का address
a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]
1000
1004
1008
1012
1016
1020
a[0]
a[1]
(i) a ≡ अपने पहले element का add.
(Add. of first element)
≡ &a[0]
12 bytes

```
int a[2][3] = {10,20,30,40,50,60};
```

```
printf("%u", a+1)      1012
printf("%u", &a+1);    1024
printf("%u", a[0]+1);  1004
```



(i) a+1 — a ⇒ address ✓ / element ✗

a) add
b) a ≡ &a[0]
c) Size of a[0] ⇒ 12 byte
a+1 ⇒ &a[0] + 1×12
1000+12
(a+1) ⇒ 1012

(ii) &a+1 ↓ ⇒ array का add.
&a + 1×24
1000 + 24
⇒ 1024

(iii) a[0]+1
a) a[0] ⇔ add. ✓ / elem.
b) a[0] ≡ &a[0][0]
c) sizeof a[0][0] ⇒ 4 byte
&a[0][0] + 1×4
1000 + 4
⇒ 1004


THANK YOU GW
SOLDIERS !