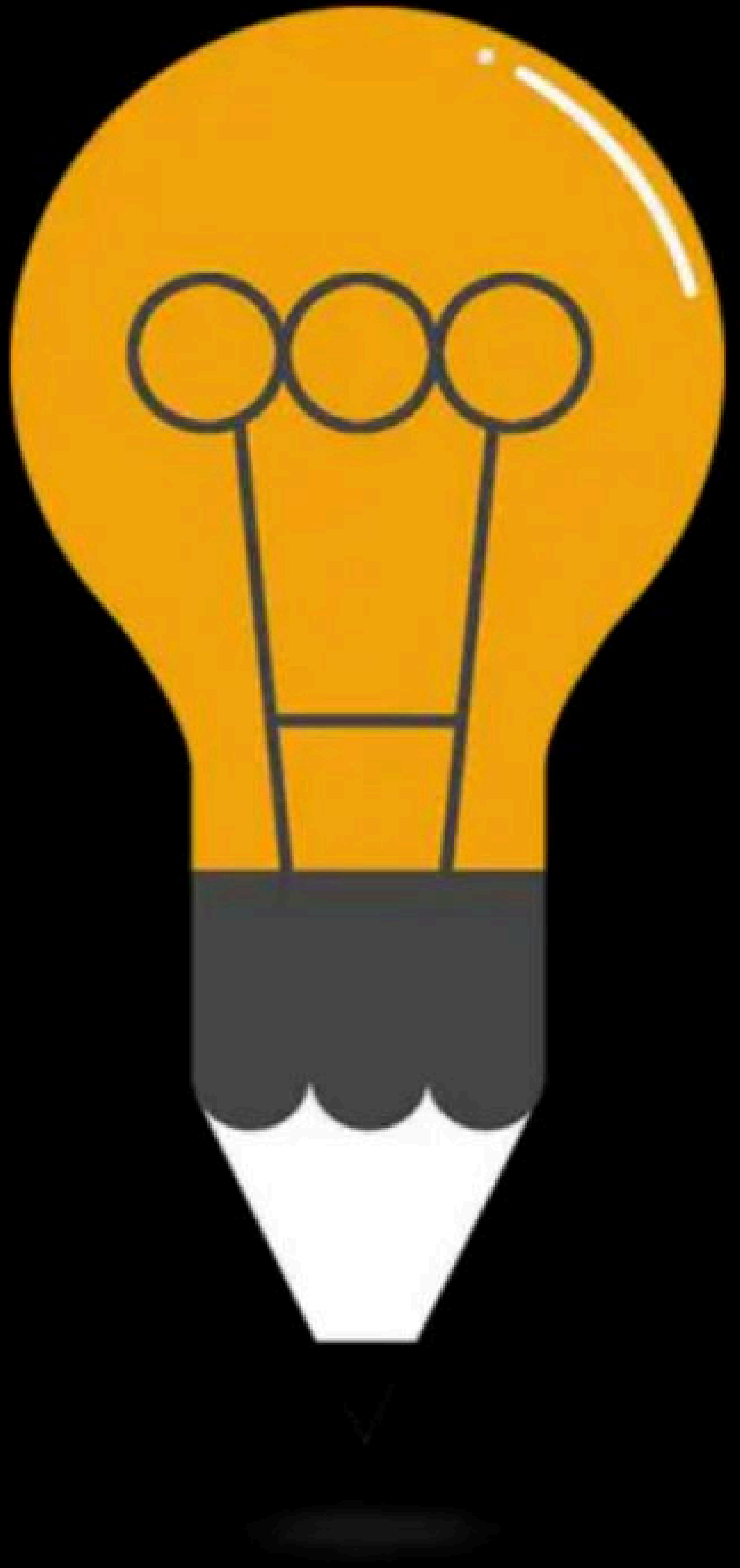
Array With Pointers

Course on C-Programming & Data Structures: GATE - 2024 & 2025

Vishvadeep Gothi • Lesson 7 • Dec 2, 2022



Pointers & Array

By: Vishvadeep Gothi

Address of First Element

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

0	1	2	3	4
1	3	5	6	2
200	202	204	206	208

printf("%u", A); base add. of array => 206

printf("%u", A+0); 200

printf("%d", *A); 1

printf("%d", *(A+0)); 1

printf("%d", *(A+1)); => *(200 + 1*2) => *202

↳ 3

printf(" %d", *(A+3)); ⇒ *(200 + 3 * 2) = *206 = 6

A[3]   printf(" %d", A[3]);

Array A[]

index[A] ⟺ *(index+A)

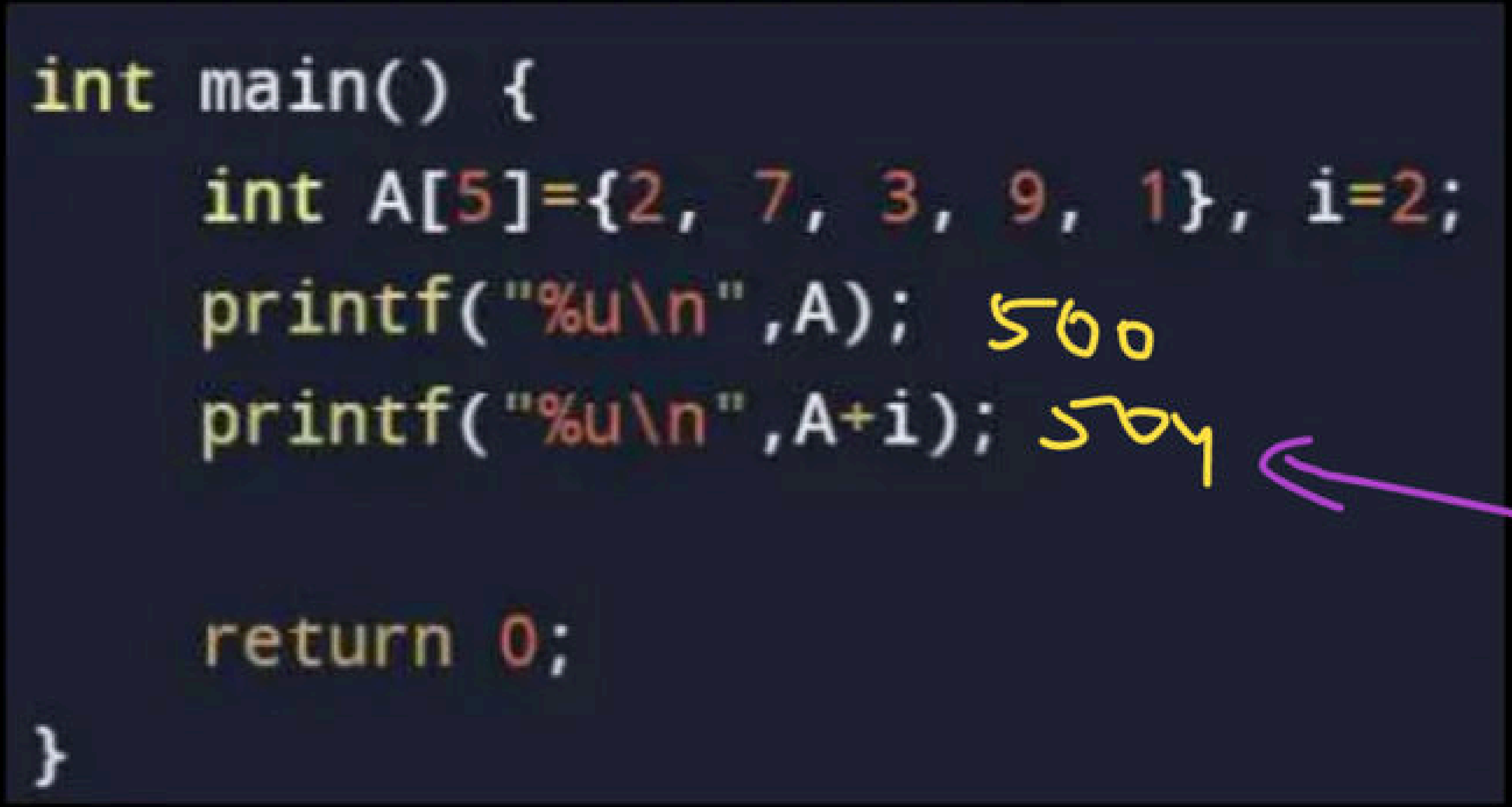
A[index] ⟺ *(A+index)

---

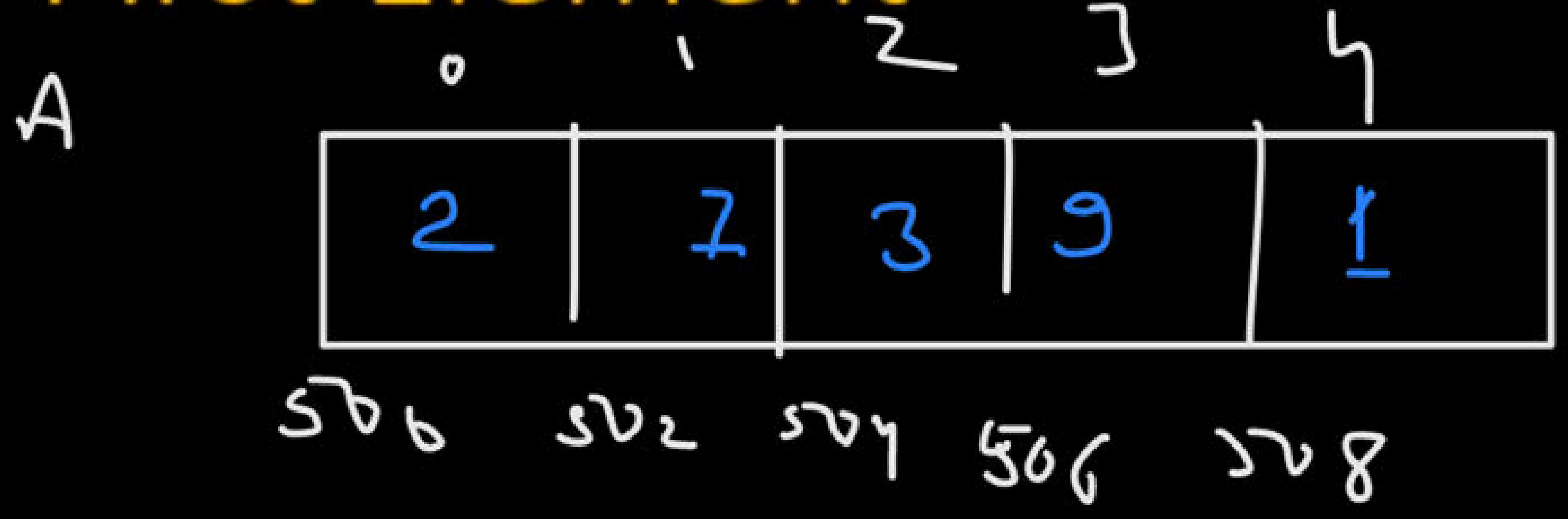
printf(%d", *(2+A)); 5

printf(" %d", 2[A]); 5

Address of First Element



```
int main() {
    int A[5]={2, 7, 3, 9, 1}, i=2;
    printf("%u\n",A);   500
    printf("%u\n",A+i); 504

    return 0;
}
```



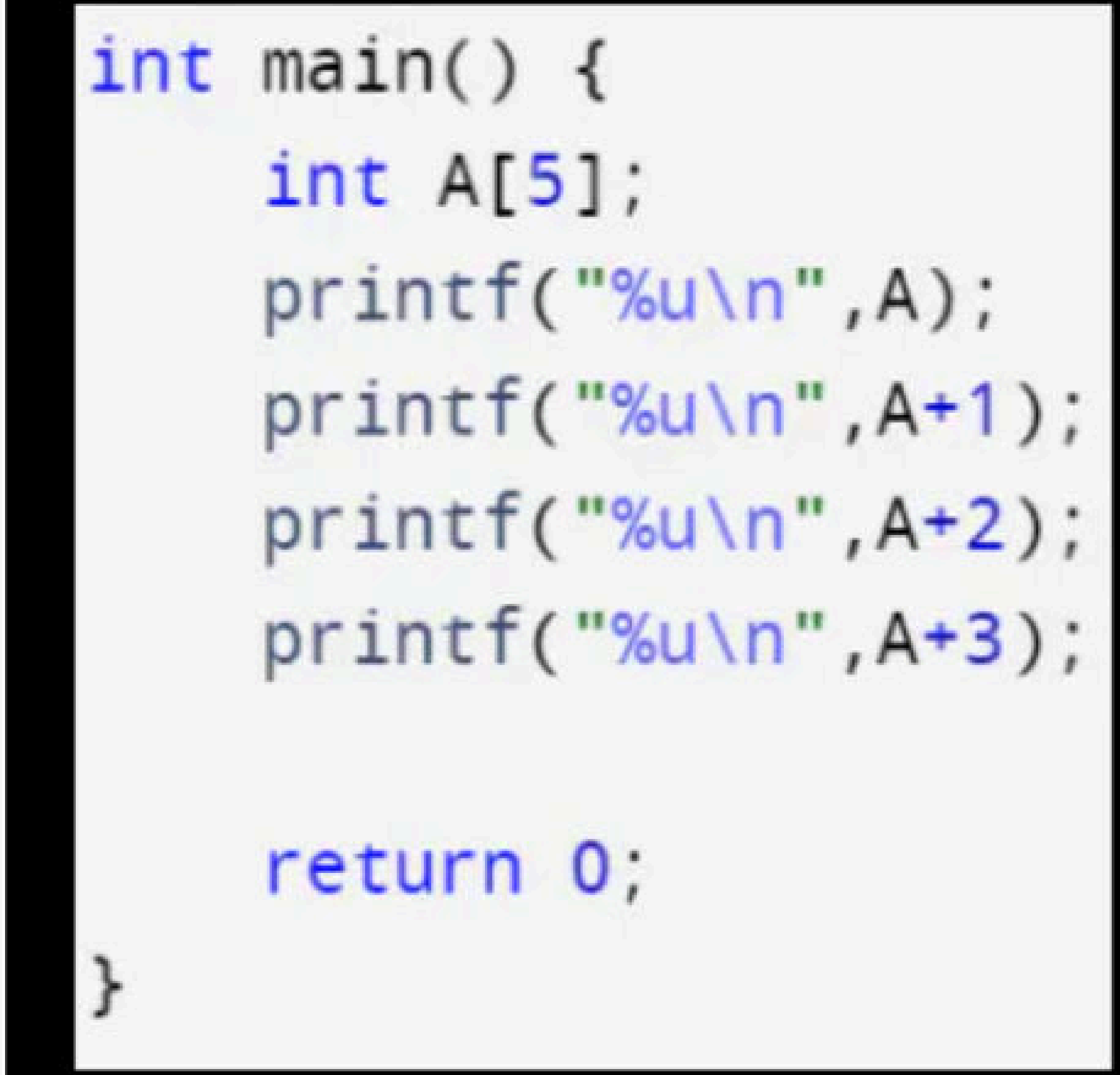
A

0	1	2	3	4
2	7	3	9	1
500	502	504	506	508

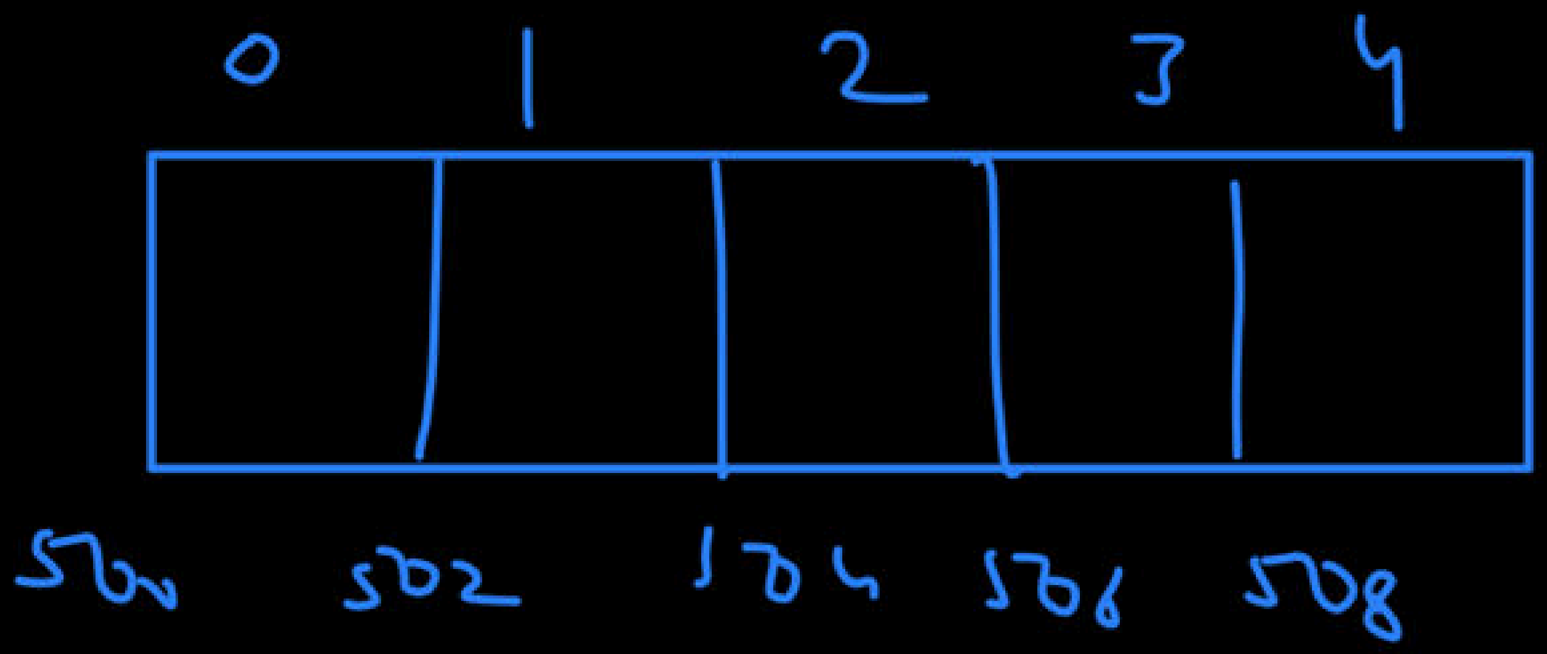
500 + 2 * 2 = 504

printf(" %d", $\boxed{*A}$+3);   5

Address of First Element



```
int main() {
    int A[5];
    printf("%u\n",A);
    printf("%u\n",A+1);
    printf("%u\n",A+2);
    printf("%u\n",A+3);

    return 0;
}
```



500

502

504

506

```
int A[5] = {8, 4, 5, 1, 3};
int *p = A;

printf("%d", p[3]); 1

printf("%d", *(p+2)); 5

p++;
printf("%d", *(p+3)); 3
```

	0	1	2	3	4
A	8	4	5	1	3
	500	502	504	506	508

p = ~~500~~
502

*(502 + 3 * 2)

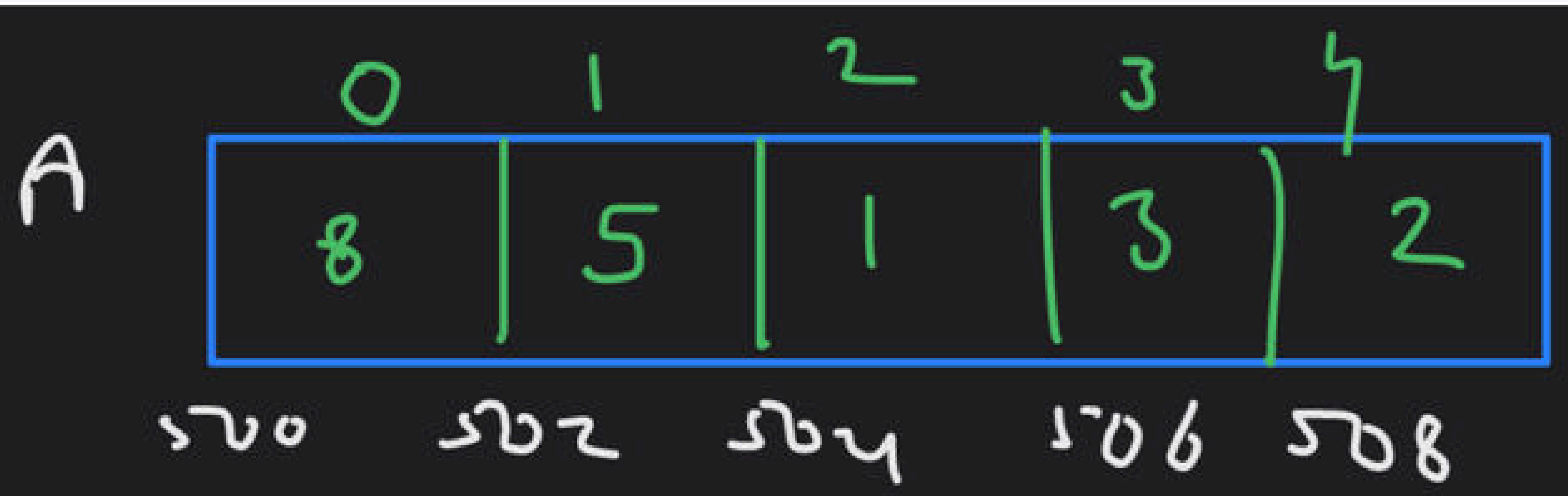
* 508

3

```
int   A[5] = {8, 5, 1, 3, 2};

int  *p = &A[0];

printf("%d",   *(p + p[4] - p[2]));
```

5

p = 500



$500 + \boxed{2 - 1}$

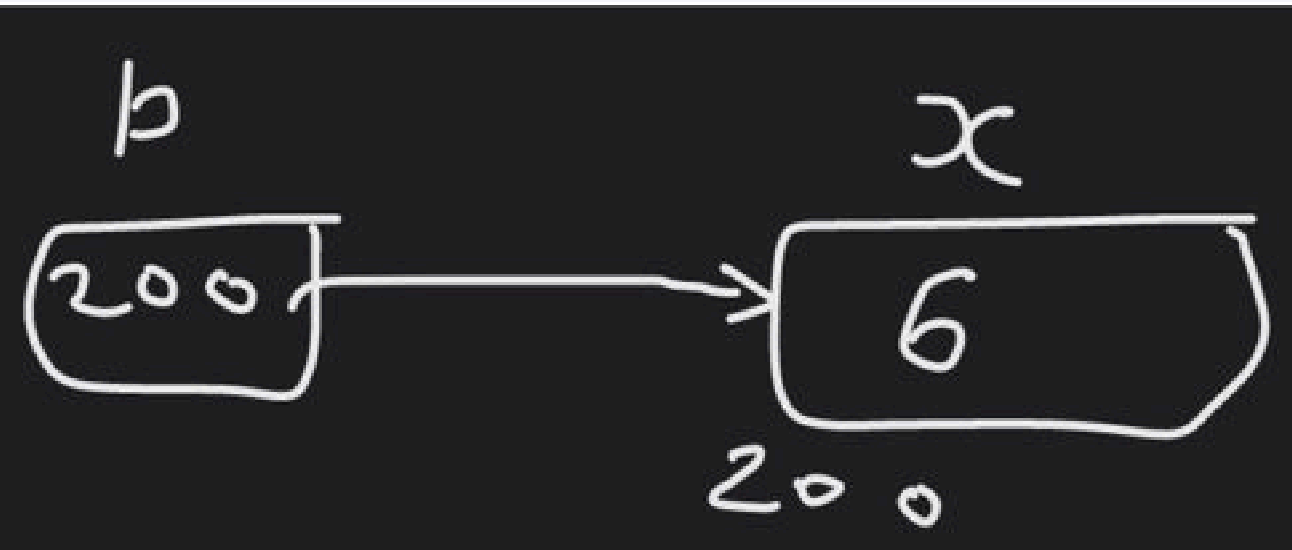
$500 + \boxed{1}$

$500 + 1 * \text{size}$

$500 + 1*2$

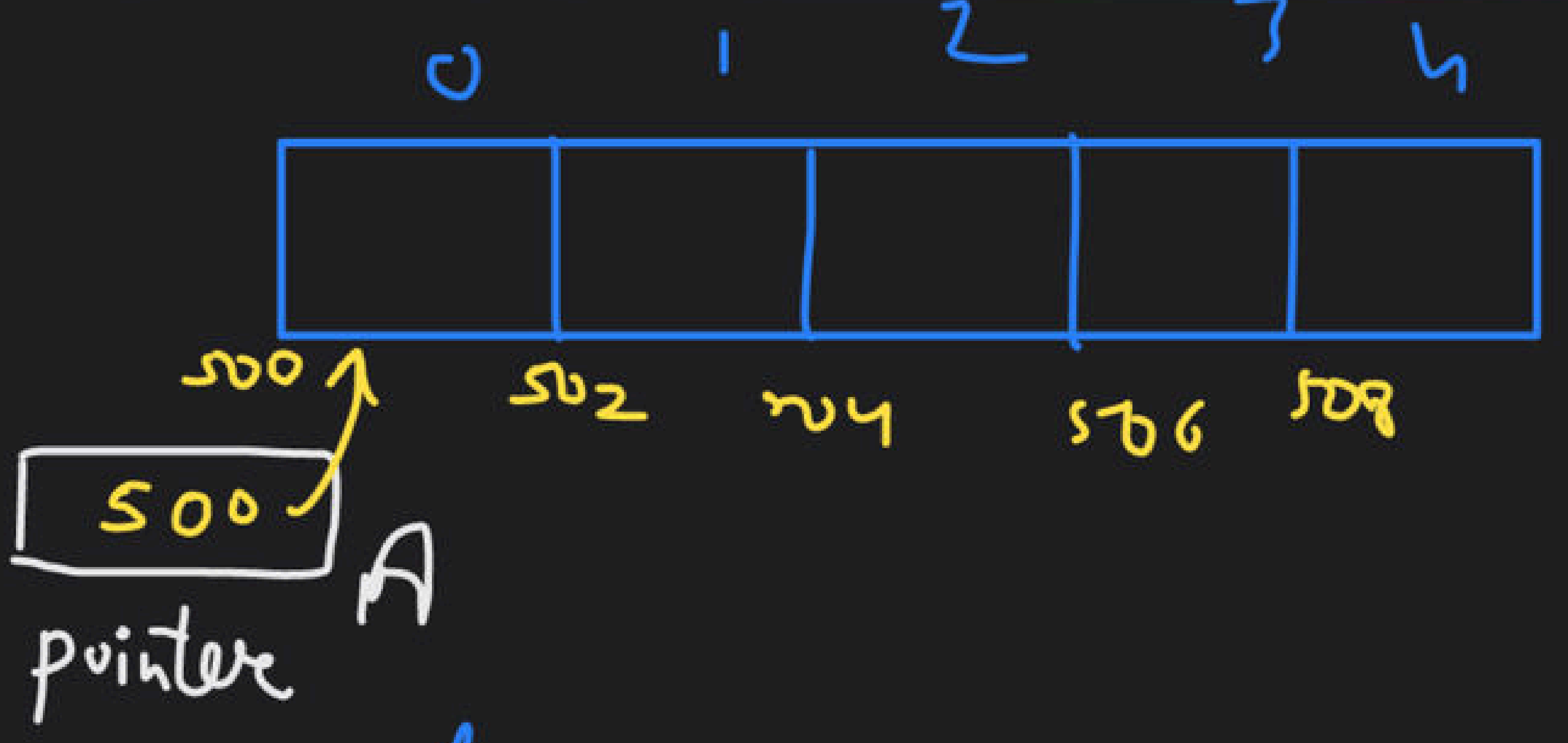
502

```
int x = 6;
int *p = &x;
```



```
printf("%d", x); ⇒ 6
printf("%d", &(x)); ⇒ add. of x
printf("%d", &p); ⇒ add. of p
```

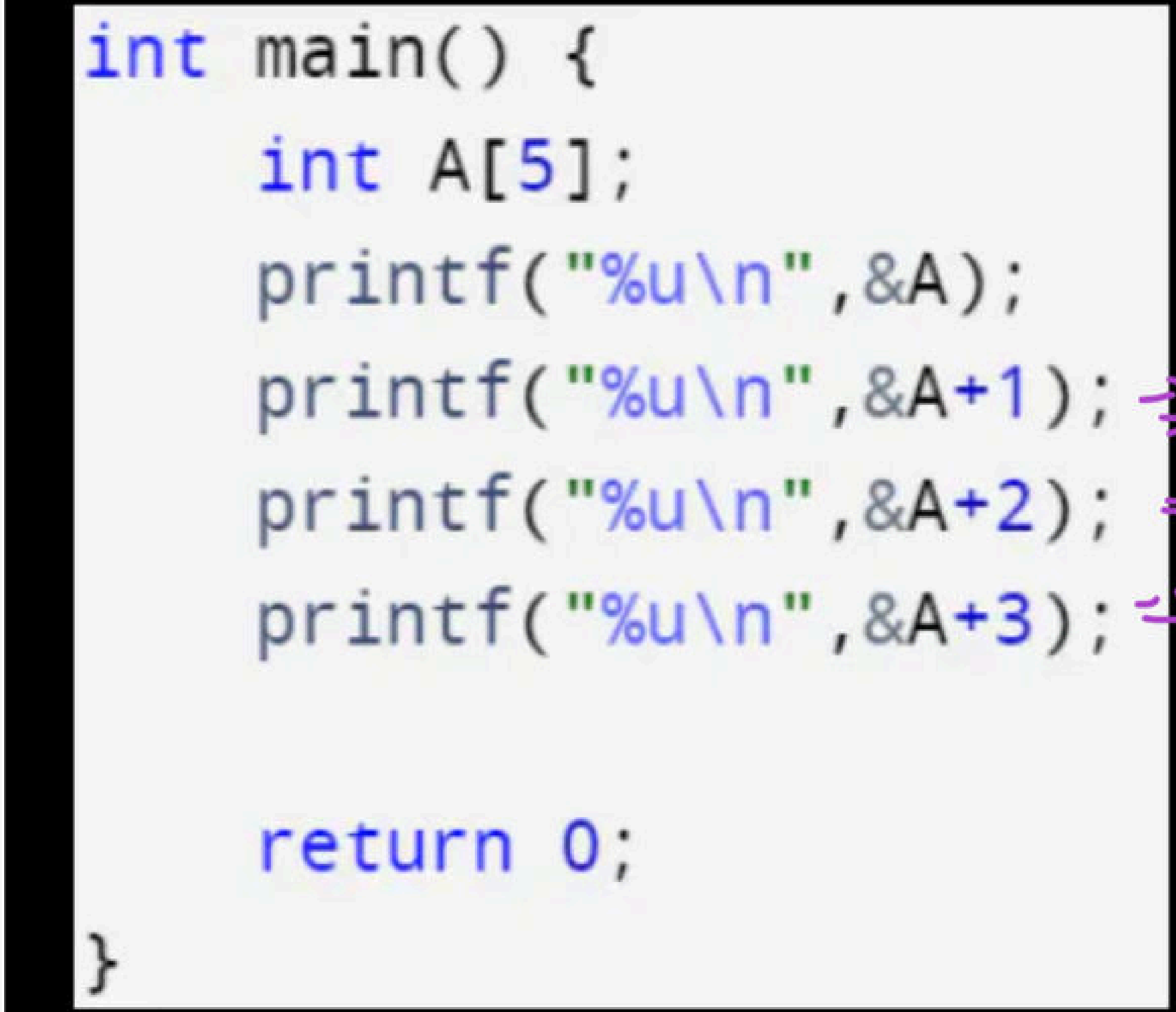
```
int A[5];
```



```
printf("%u", A); 500
printf("%u", &A); add. of constant pointer A.
```

&A[0]
⇓
base add. of array

Address of Array Name



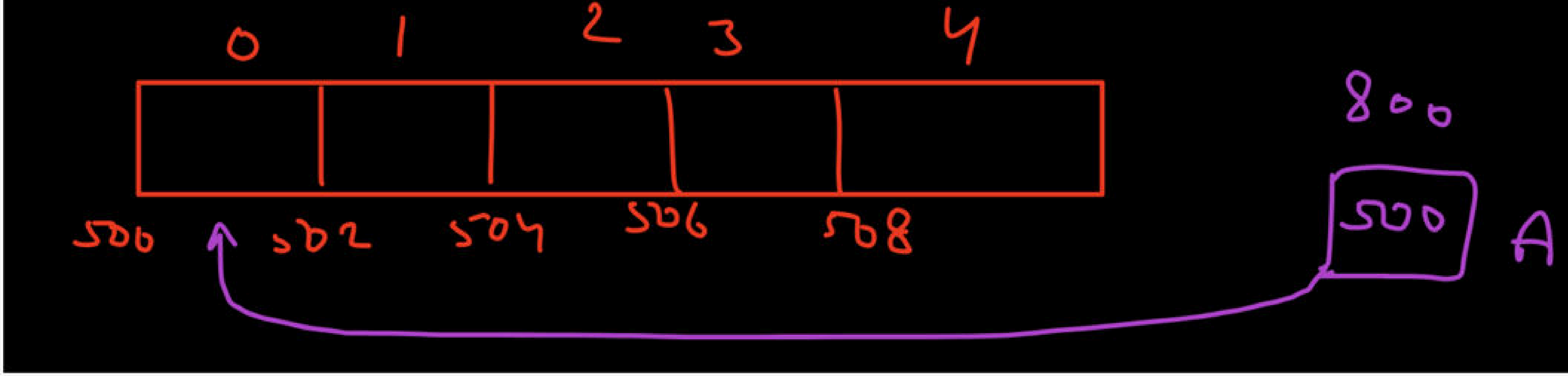
```
int main() {
    int A[5];
    printf("%u\n",&A);
    printf("%u\n",&A+1);
    printf("%u\n",&A+2);
    printf("%u\n",&A+3);

    return 0;
}
```

800

⇒ increment by size of entire array ⇒ 810
(10 bytes)

⇒ 820

⇒ 830



int x ;

int A[5] = {5, 3, 8, 6, 1};

x = A; ⇐ error

printf("%u", x);

x = (int)A

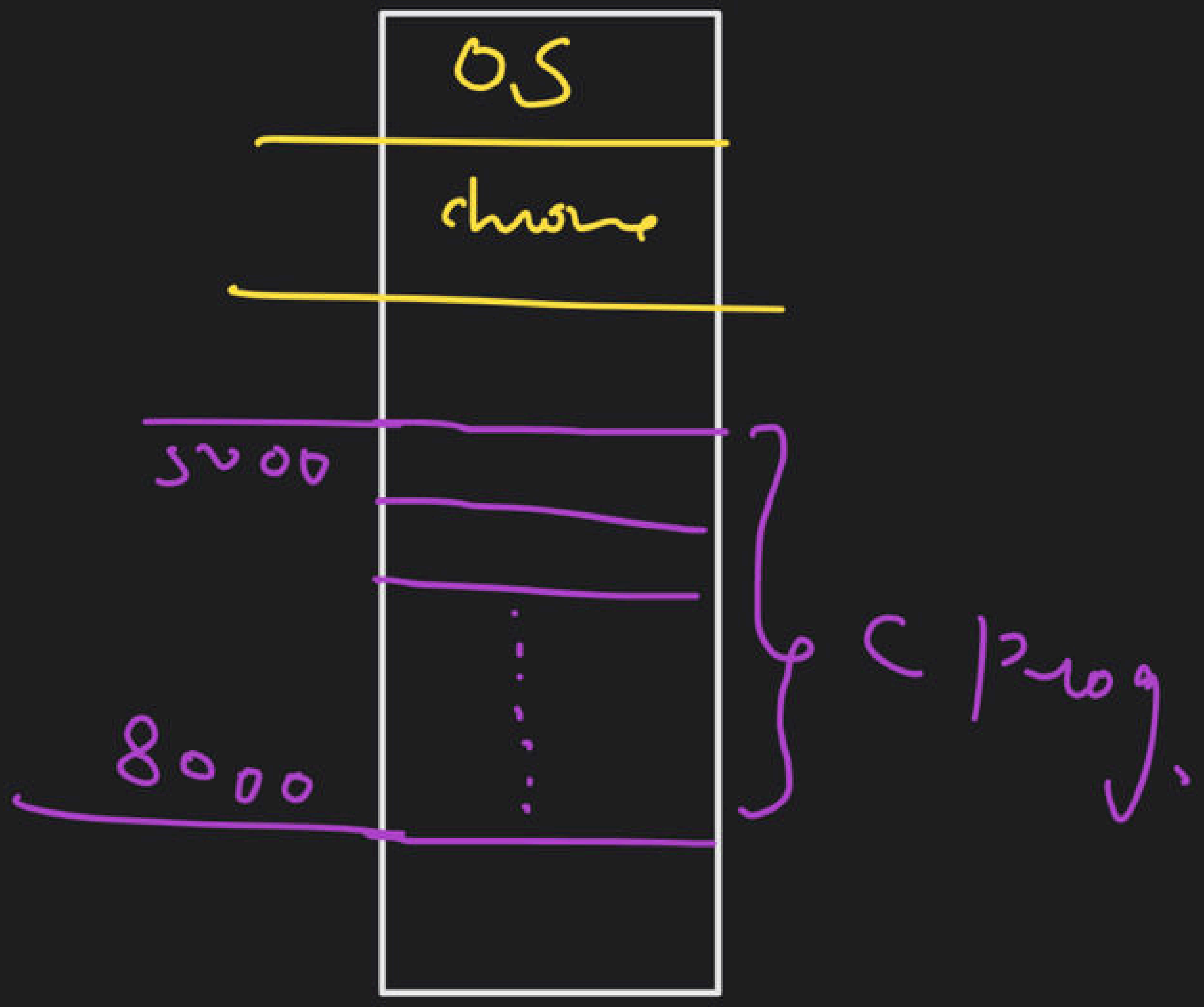
int variable = base add. of array

-type not matching

x = 'A'; ⇐ 'A' ASCII value 65 implicitly converted into int and stored in x .

```
int *p;

p = 1010;  ← error

printf("%d", *p);
```

```
int A[5];
int *p;
int x;

x = (int) A;
p = (int *) x;   ?
```



2-D array

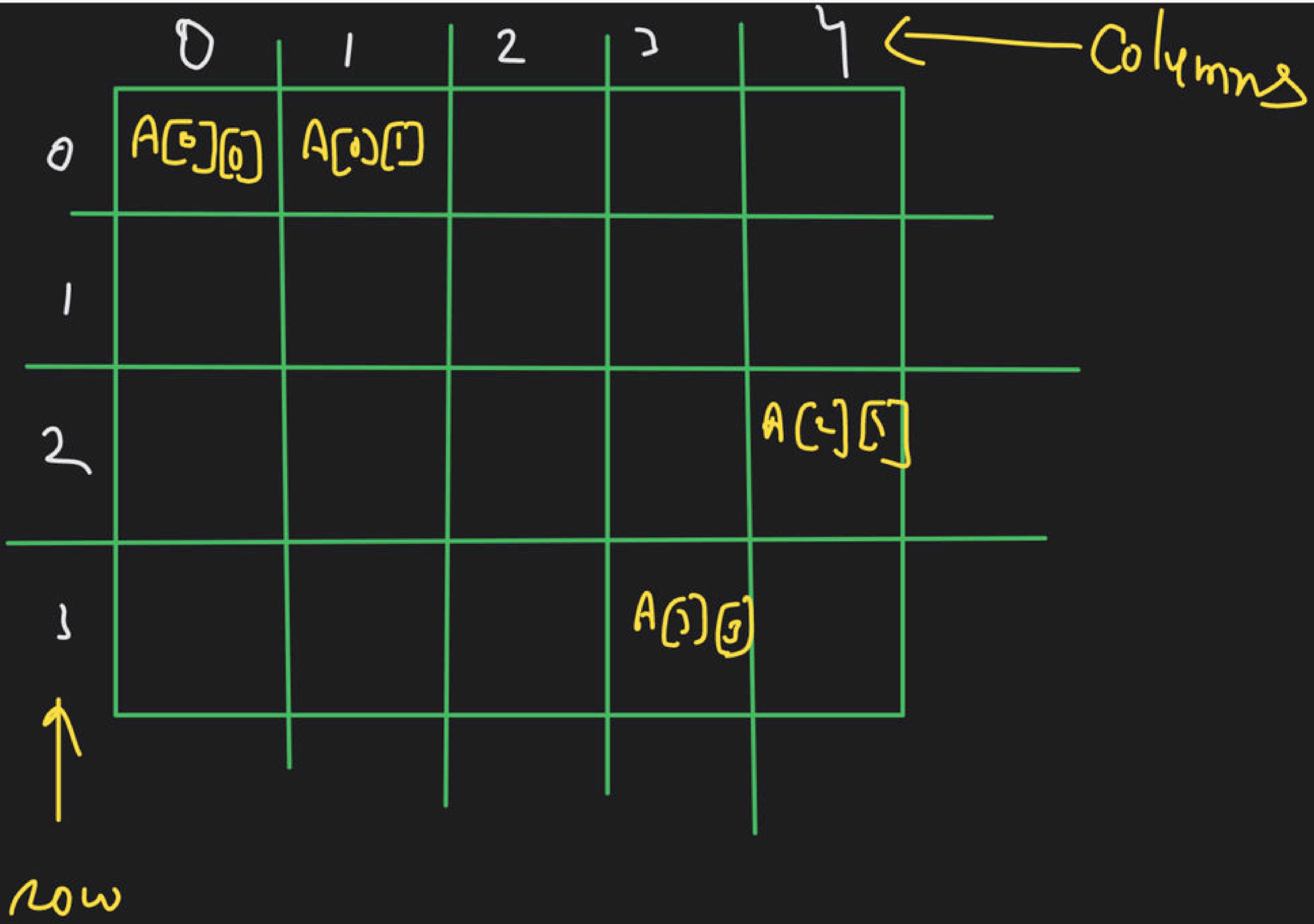
collection of 1-D array.

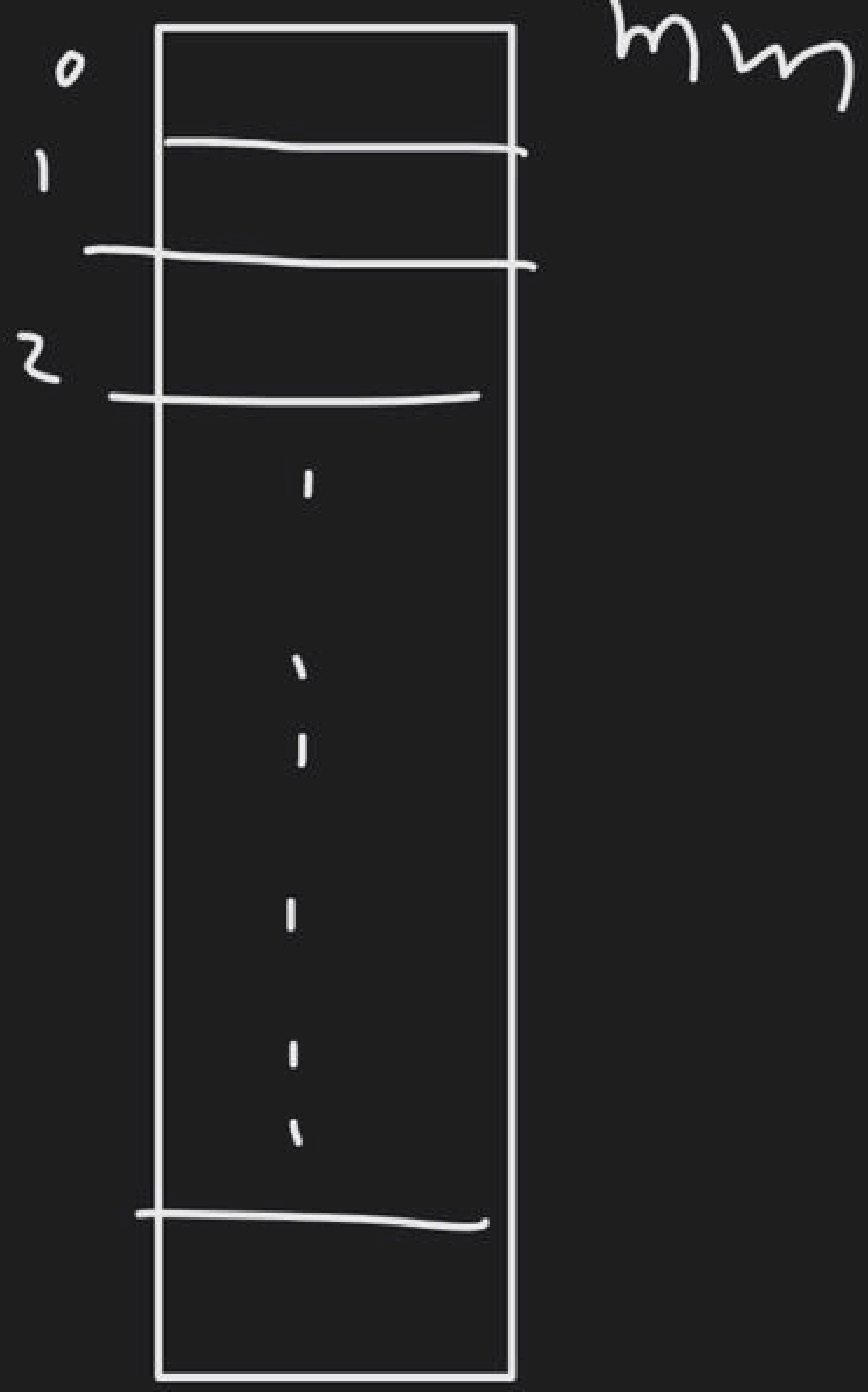
Declare:-

datatype name[no. of arrays] [each array size]

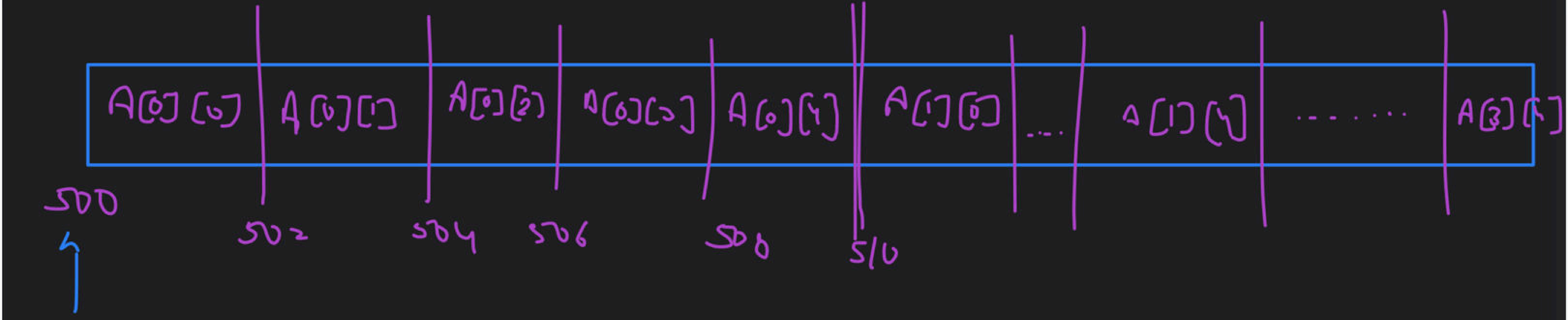
int A[4][5]

	0	1	2	3	4
0					
1					
2					
3					





stored in mm row wise (Row major ordere)



base add. of array

add. of A[i][j] = Base + size of each element $\left(i * \text{no. of columns} + j \right)$

$A[2][4] = 500 + 2 * (2 * 5 + 4)$

$= 528$

int A [4][3] ;

rows => 0, 1, 2, 3

columns => 0, 1, 2

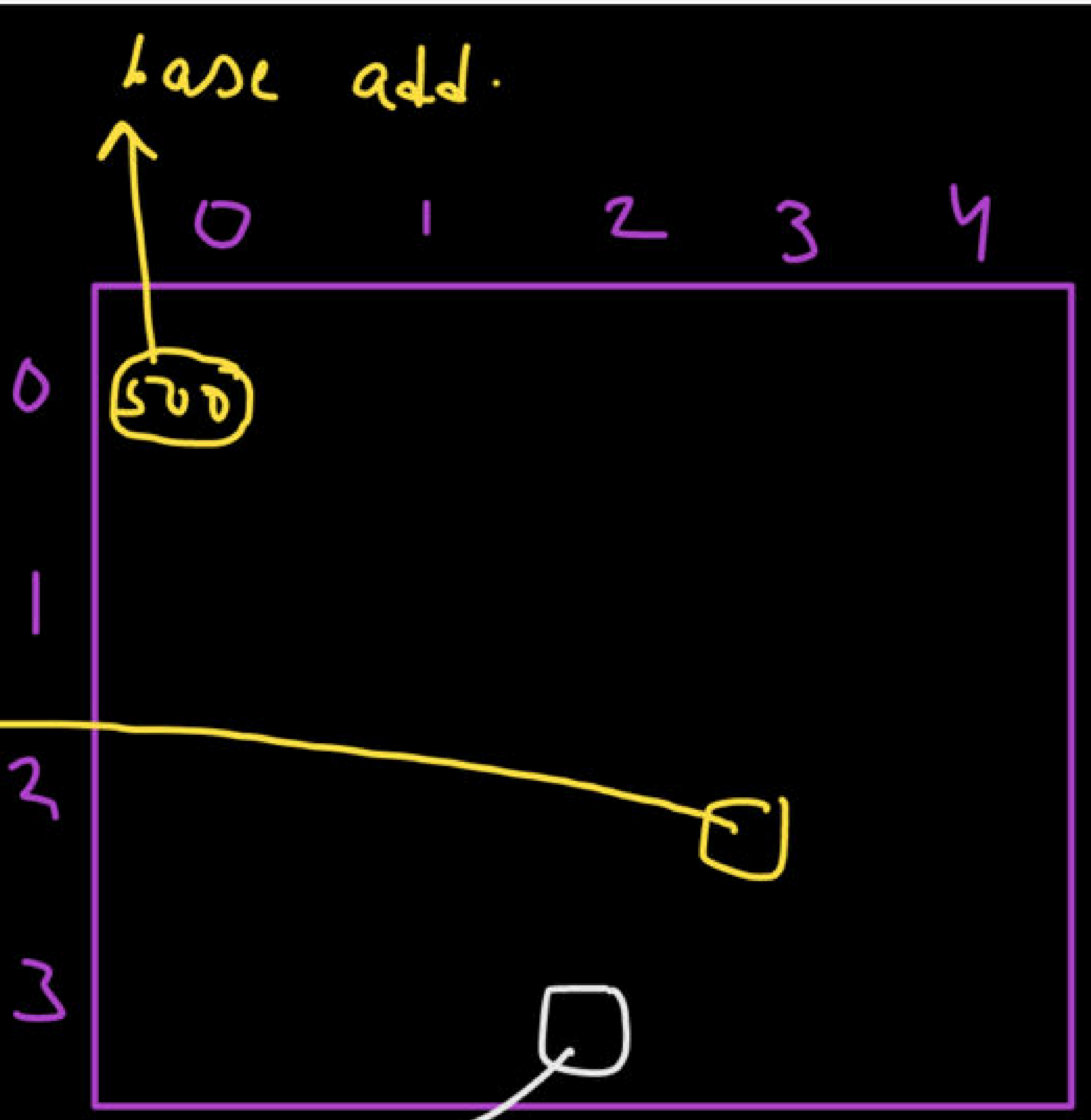
printf("%u", A); => base addr. 500

printf("%u", &A[0][0]) => 500

		0	1	2
A+0	0	500		
A+1	1			
A+2	2			
A+3	3			

int A[4][5]

2-D array



```
A          → base add. of array
A+0        → name of the 0th row
&A         → add. of pointer A
&A[0] + 1  → ?
&A[2][3]   → add. of A[2][3] ⇒ 526
A[3][2]    → element (value) at row 3 column 2
```

Happy Learning.!