WEEK 3
LECTURE 3

2D - ARRAYS

2D Arrays

→ array of arrays.

→ use case: to work on multiple rows and columns.

→ Total elements - m*n.

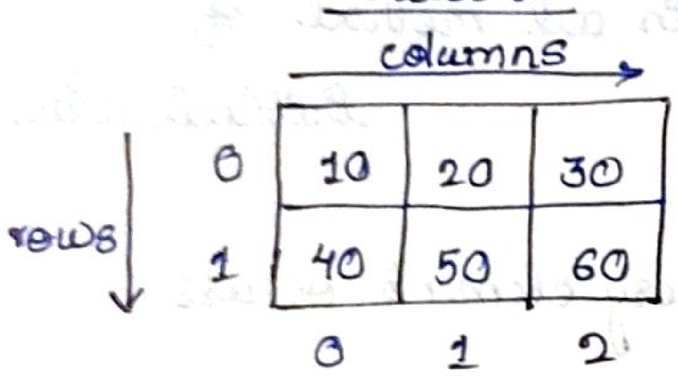
Declaration: int array [m][n];

→ cols → 0 to n-1

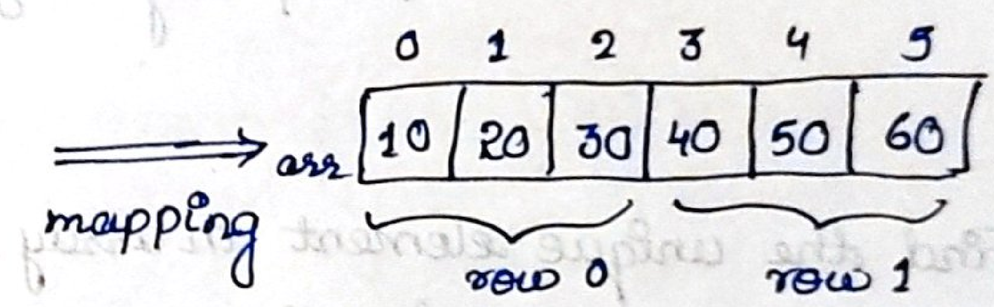
→ rows → 0 to m-1

int arr [m][n];

Visualise-



In memory-



mapping: Linear_Index = c * i + j

no. of columns

row Index

Col Index

i = Linear_Index / c

j = Linear_Index % c

Access ⇒ arr [i][j];

→ Col Index $0 \leq j < n$

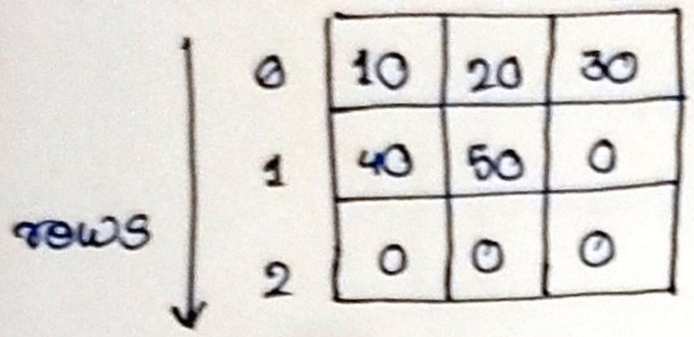
→ row Index $0 \leq i < m$

Initialisation :

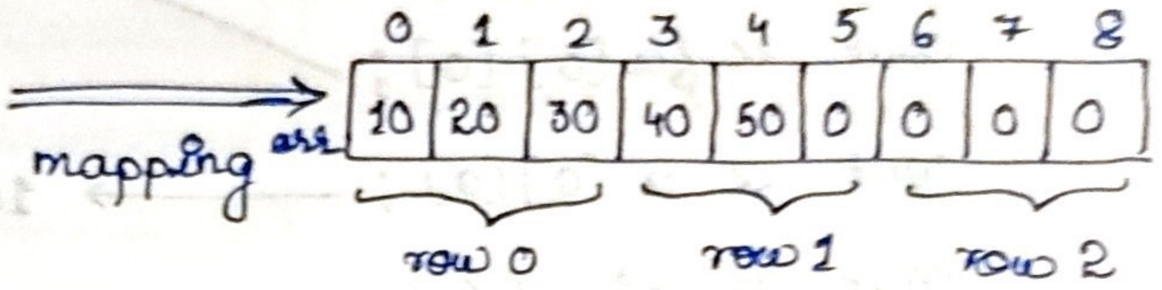
```
int arr [3][3] = { {10,20,30} ,{40,50}}
int arr [3][3] = { 10,20,30,40,50 }
```

same

visualise -



In memory -



Eg : arr [1][0] = 40

c → no of columns = 3

Linear_index = c * i + j

= 3 * 1 + 0

= 3

↳ ∴ 40 is stored at index 3.

2D arrays and function :-

↳ pass by reference.

func (~~int~~ arr [ ][500] , int rows, int cols)

actually arr here is

int (*)[500]

(array of pointers),

not a 2D array.

[cannot leave blank]

↳ if dont know, put the large value.

why?

↳ 500 tells the size of array of pointers.

↳ this value and no of columns in array passed in function call should be same.

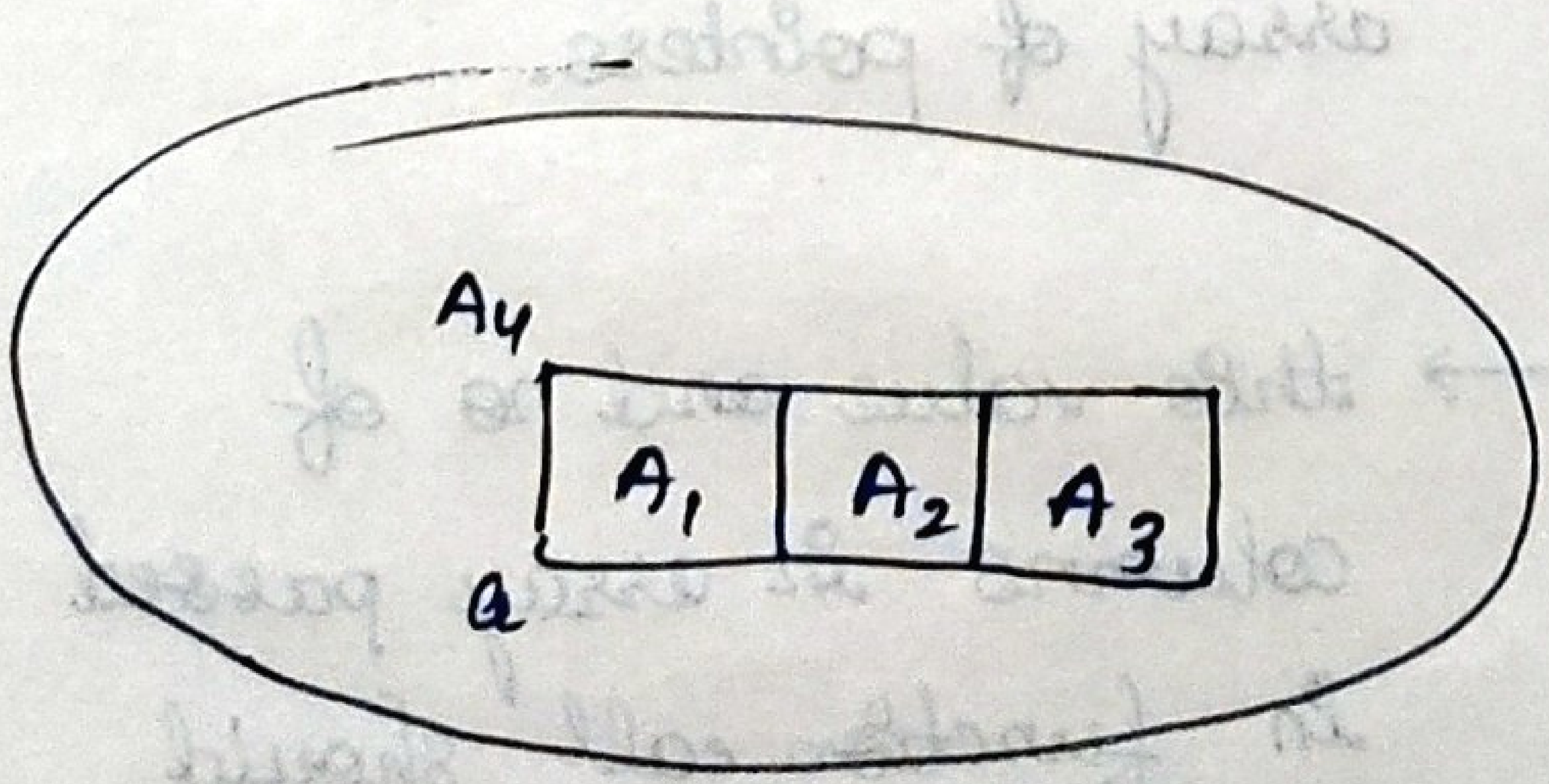
2D array :

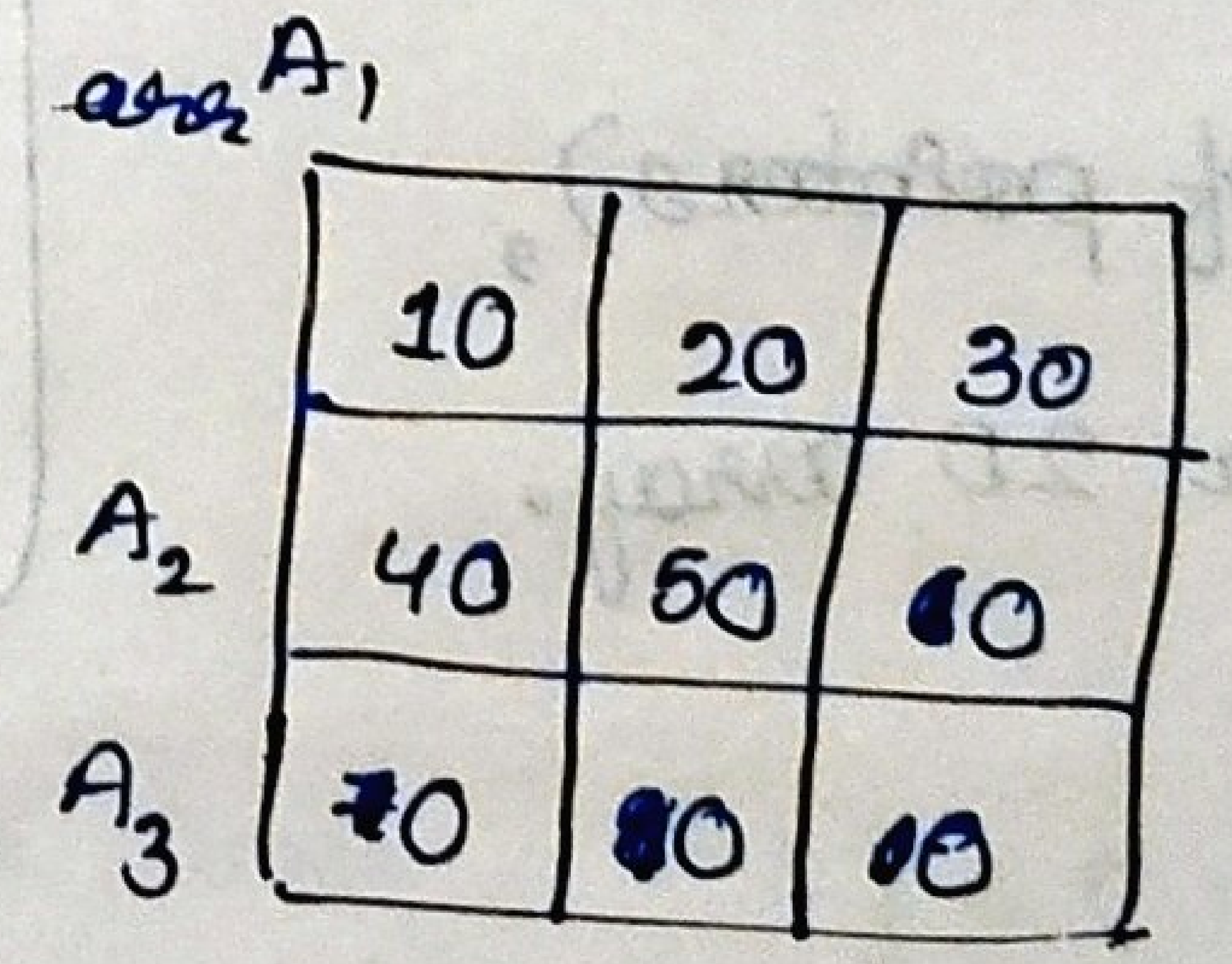
→ not an array of pointers

→ Its array of arrays.

```
func (int a[][3], int rows, int cols)
{
    cout << &a;            → add. of a → A4
    cout << a;             → A1
    cout << a[0];          → A1
    cout << &a[0];         → A1
    cout << &a[0][0];      → A1
    cout << a[0][0];       → 10
    cout << sizeof(a);     → 4
    cout << sizeof(a[0]);  → 12
}

int main ()
{
    int arr [2][3] = {10,20,30,40,50};
    func(arr, 2, 3)
    return 0;
}
```



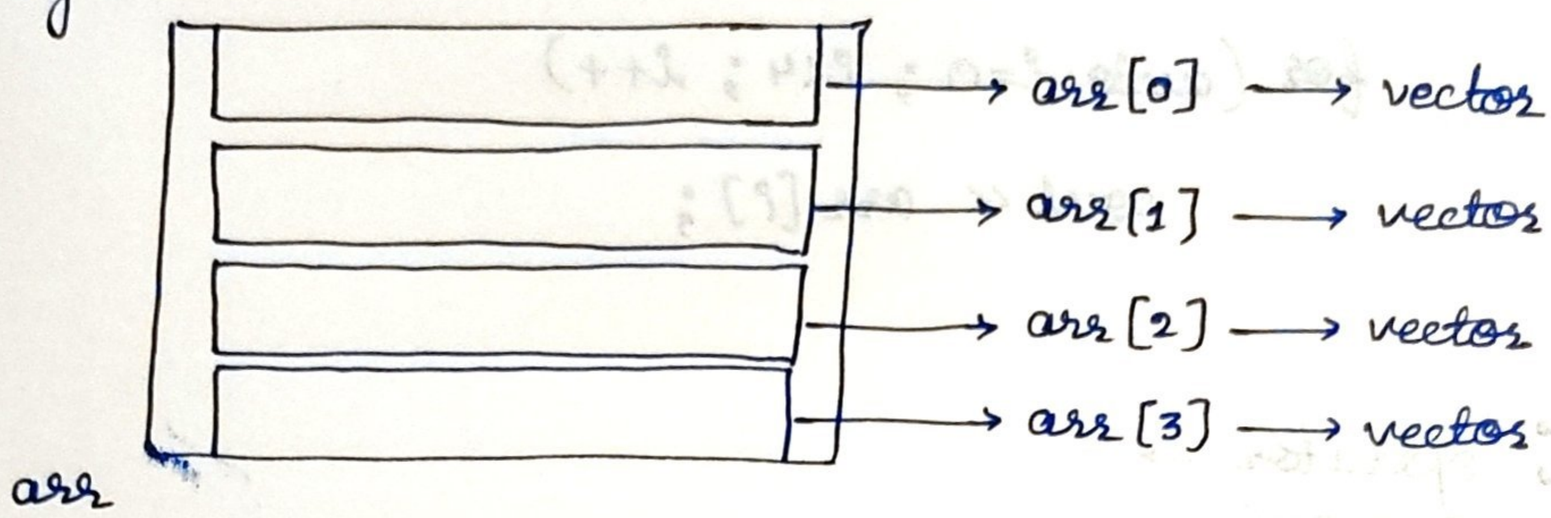


2D - Vector :

Declaration :

```
vector < vector <int> >  arr ;
```

In memory :



⇒ `vector < vector <int> > arr ;`

⇒ `vector < vector <int> > arr (m) ;` → no of rows

no of rows

```
arr.size();
```

no of columns

```
arr[i].size();
```

→ in i^{th} row

→ size of i^{th} row

Initialisation :

```
vector < vector <int> > arr (rows , (vector <int> (cols , value)) ;
```

(size ↑ rows; ↑ initialisation of 1D vector in array)

rows → no. of rows in arr

cols → no. of columns in arr
→ size of 1D vector

value → initial value of all elements of all the 1D vectors

Eg: `vector <vector <int>> arr (2 , vector <int> (4 , 101));`

101	101	101	101
101	101	101	101

arr

auto keyword :

↳ automatically replace with required data type.

```
int arr[4];

for (auto i=0 ; i<4 ; i++)

    cout << arr[i];
```

: operator →

↳ belongs to operators

↳ generally used for sequential access.

↳ map, set, array, vector

Eg : `vector<int> arr {1, 2, 3};`

```
for (int i: arr)
    cout << i;          ⟶ 1 2 3

for (int auto i: mapping)
    cout << i;

for (auto i: set)
    cout << i;
```