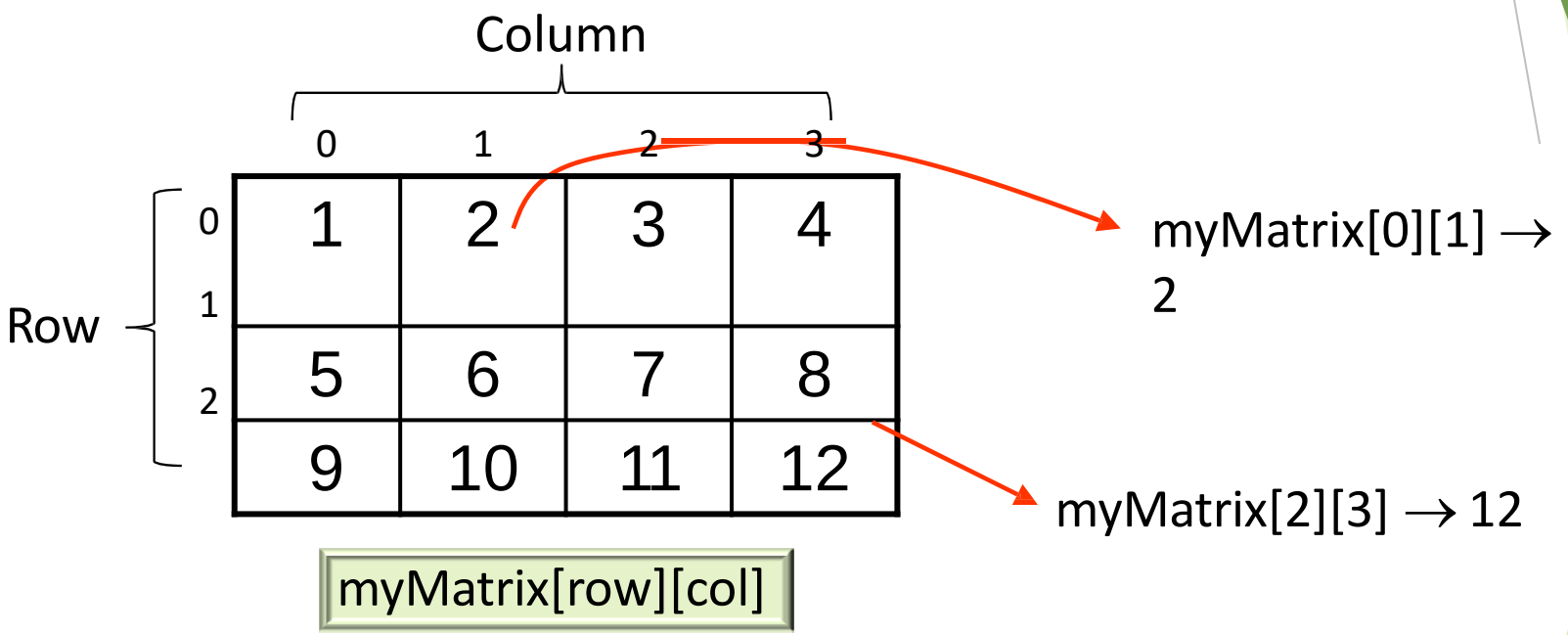
Pointers

Pointers & 2-d Arrays

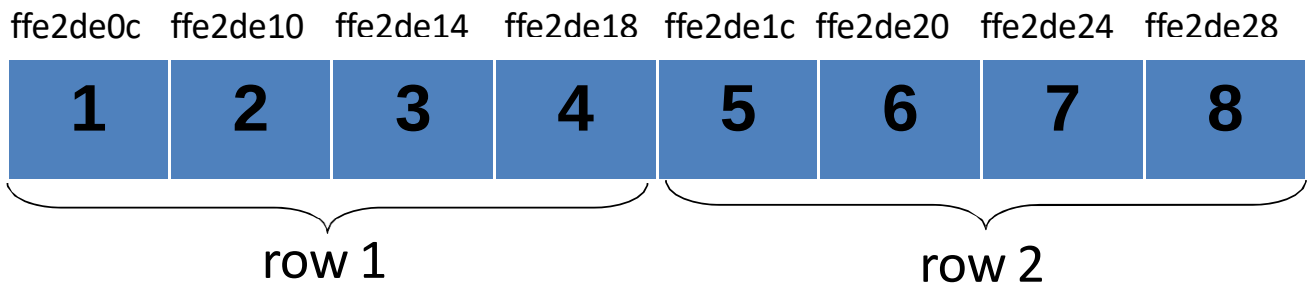
2D Arrays

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



Physically, in one block of memory

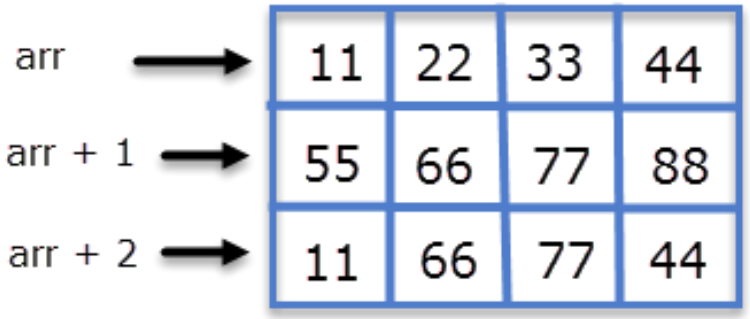
int myMatrix[2][4] = { {1,2,3,4},{5,6,7,8} };



Array elements are stored in *row major* order
Row 1 first, followed by row2, row3, and so on

2-D Arrays and Pointers

- The name *arr* of the array is a constant pointer that points to the 0th element of the array.
  - arr points to 0th 1-D array.
  - (arr + 1) points to 1st 1-D array.
  - (arr + 2) points to 2nd 1-D array.



  - Dereferencing arr we will get *arr, base type of *arr is (int*)
  - *(arr+i) points to the base address of the ith 1-D array.
  - (arr + i) and *(arr+i) points to same address but their base types are completely different. The base type of (arr + i) is a pointer to an array of 4 integers, while the base type of *(arr + i) is a pointer to int or (int*).

2-D array contd..

- *(arr + i) points to the address of the 0th element of the 1-D array. So,
- *(arr + i) + 1 points to the address of the 1st element of the 1-D array
- *(arr + i) + 2 points to the address of the 2nd element of the 1-D array

- Hence we can conclude that:
  - *(arr + i) + j points to the base address of jth element of ith 1-D array.
  - On dereferencing *(arr + i) + j we will get the value of jth element of ith 1-D array.
- By using the expression ***(*(arr + i) + j)** we can find the value of jth element of ith 1-D array.

- The pointer notation *(*(arr + i) + j) is equivalent to the subscript notation arr[i][j].

Program to access values and address of elements of a 2-D array using pointer notation.

```
int main()
{
    int arr[3][4] = { {11,22,33,44}, {55,66,77,88}, {11,66,77,44} };
     int i, j;

    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , *(arr + i));
        for(j = 0; j < 4; j++)
        {
            printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j) );
        }
        printf("\n\n");
    }
    return 0;
}
```

Homework

1. Perform matrix addition using pointer notation. Use functions and pass pointers to the arrays to read, print and add the arrays.