Array Operator Equivalence

- We have the following equivalences:

```
int a[20];
a[i]          - is equivalent to
*(a+i)        - is equivalent to
*(&a[0]+i)    - is equivalent to
*((int*)((char*)&a[0]+i*sizeof(int)))
```

- You may substitute array indexing `a[i]` by `*((int*)((char*)&a[0]+i*sizeof(int)))` and it will work!
- ***C was designed to be machine independent assembler***

2D Array. 1st Implementation

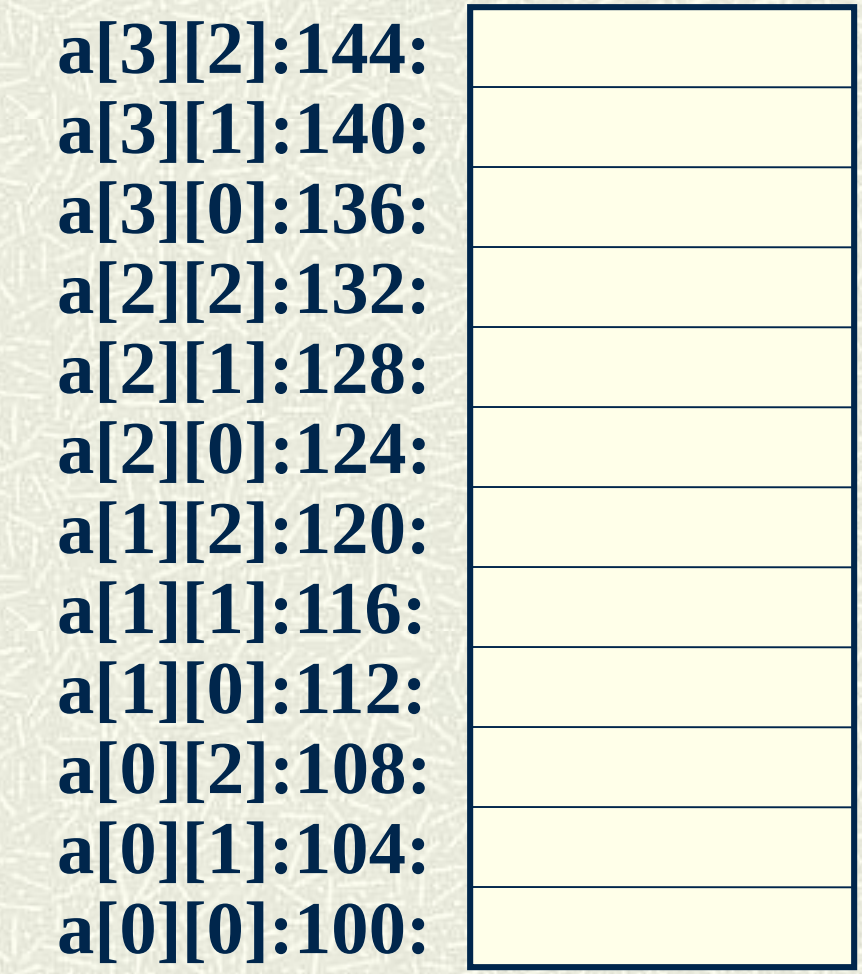
- 1st approach

Normal 2D array.

```
int a[4][3];
```

```
a[i][j] ==
*(int*)((char*)a +
i*3*sizeof(int) +
j*sizeof(int))
```

a:



2D Array 2nd Implementation (Jagged Arrays)

- 2nd approach

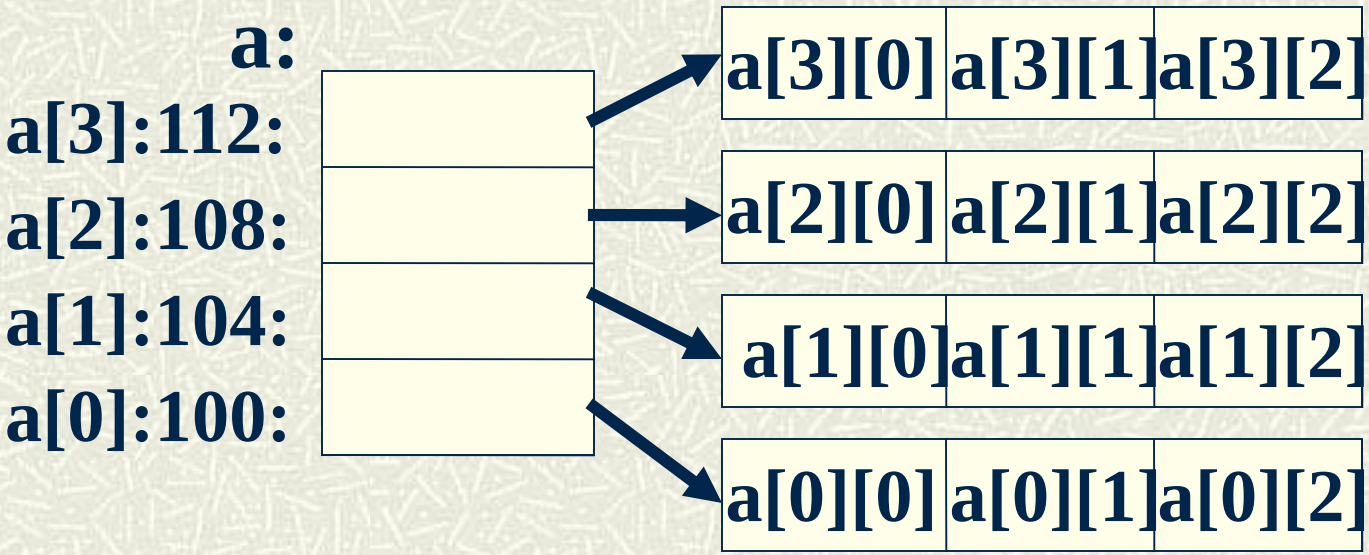
Array of pointers to rows

```
int*(a[4]);
for(int i=0; i<4; i++){
     a[i]=(int*)malloc(sizeof(int)*3);
     assert(a[i]!=NULL);
}
```

2D Array 2nd Implementation

- 2nd approach

Array of pointers to rows (cont)




```
int*(a[4]);

a[3][2]=5
```

2D Array 3rd Implementation (Jagged Arrays)

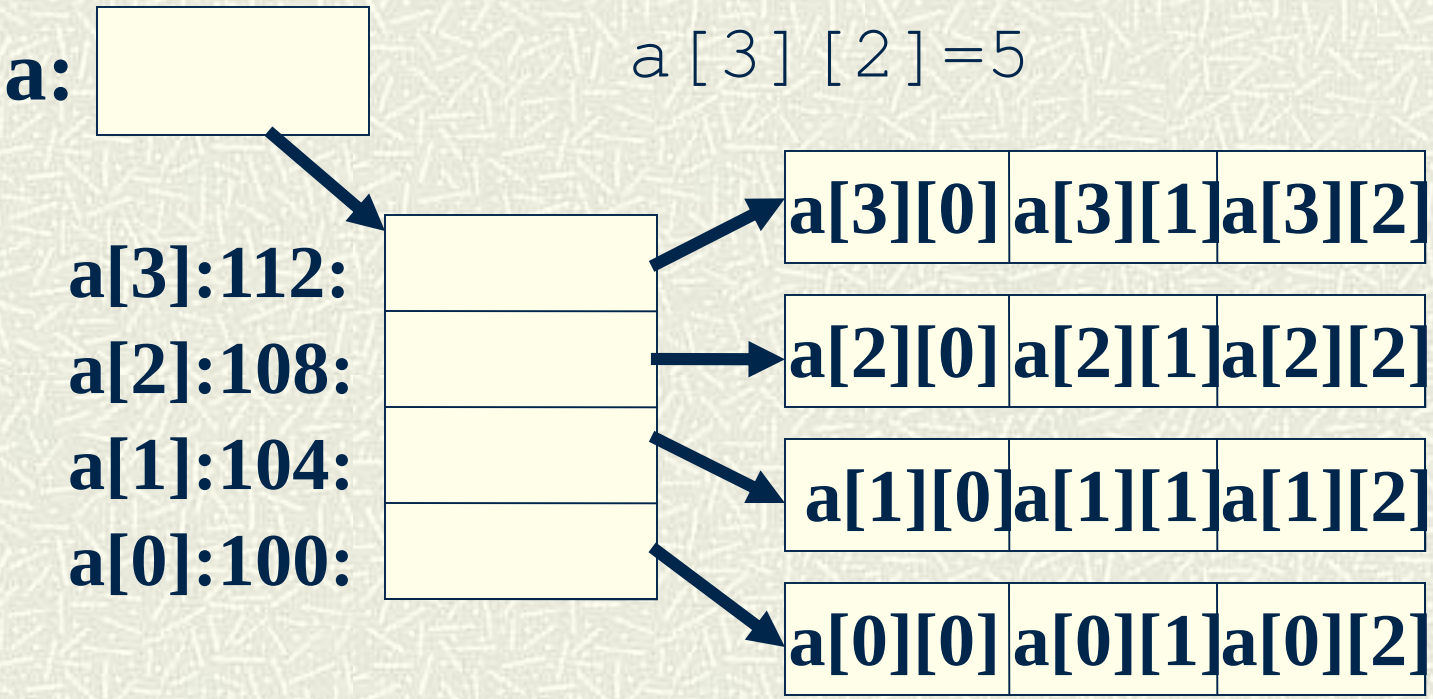
- 3rd approach. a is a pointer to an array of pointers to rows.

```
int **a;
a=(int**)malloc(4*sizeof(int*));
assert( a!= NULL)
for(int i=0; i<4; i++)
{
  a[i]=(int*)malloc(3*sizeof(int));
  assert(a[i] != NULL)
}
```

2D Array 3rd Implementation

- a is a pointer to an array of pointers to rows. (cont.)

```
int **a;
a[3][2]=5
```



Advantages of Pointer Based Arrays

- You don’t need to know in advance the size of the array (dynamic memory allocation)
- You can define an array with different row sizes

Advantages of Pointer Based Arrays

- Example: Triangular matrix