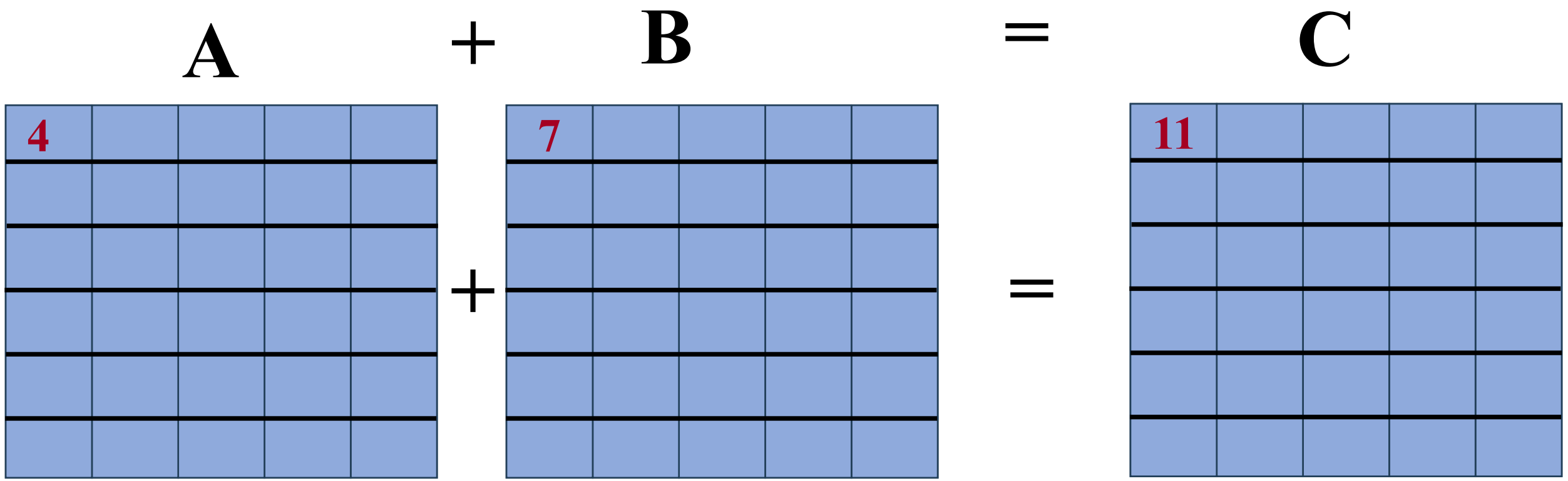
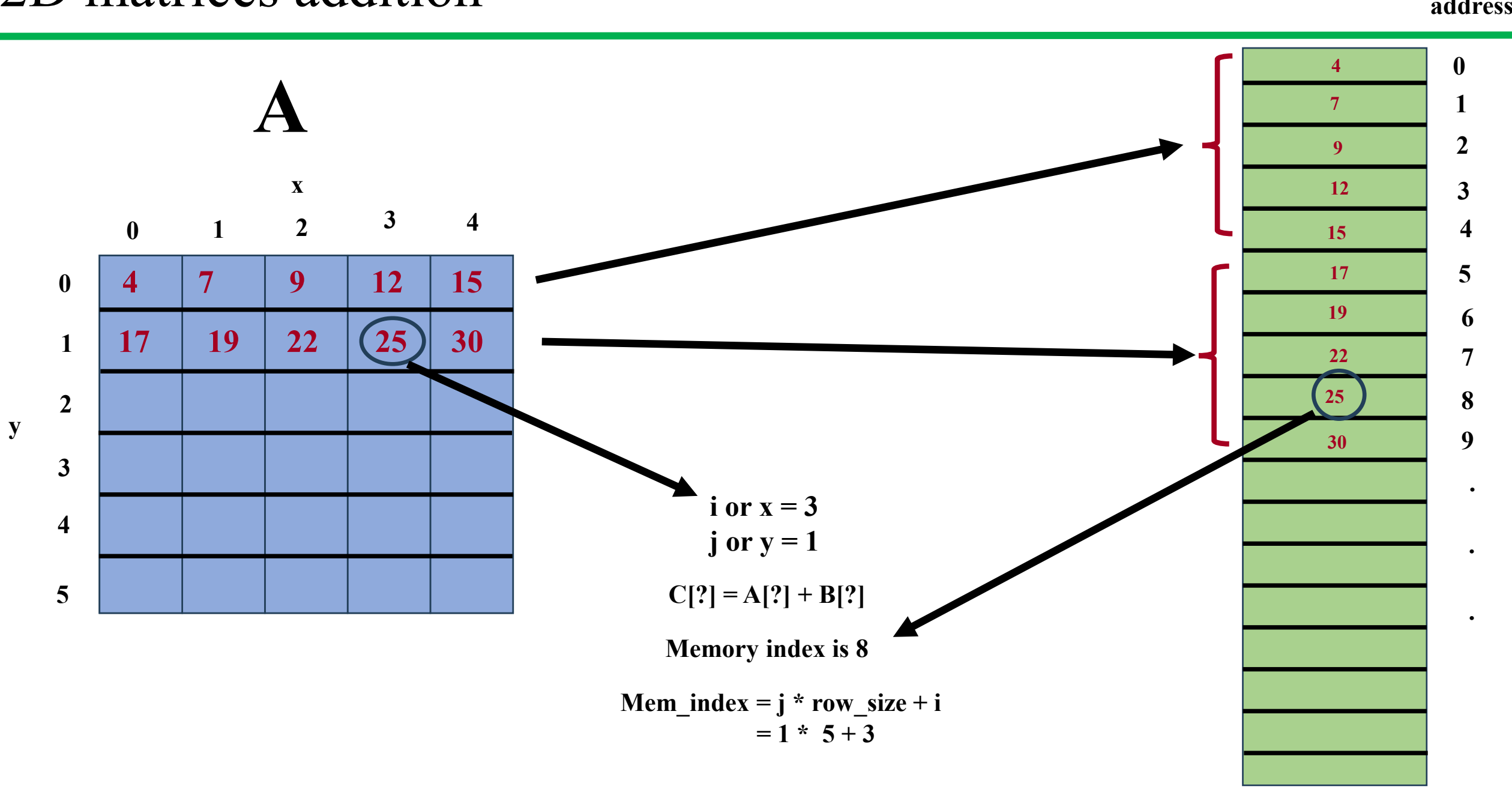
2D matrices addition

- The second level of this course.
- Previously, we focused on 1D threads and blocks.
- We applied that on adding two vectors.
- Block size = blockDimx = number of threads per block = 1024
- Grid size = GridDimx = 1024 blocks
- Block size = (32,32) = blockDimx and blockDimy
- Grid size = (1024*1024) = GridDimx and GridDimy

2D matrices addition

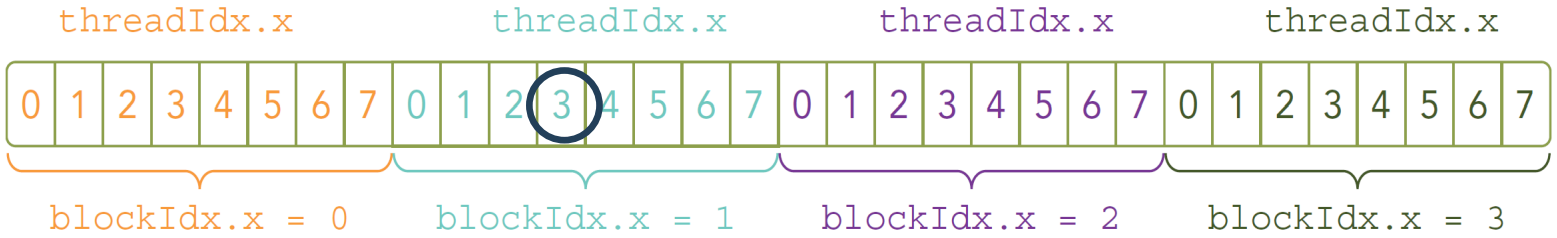


2D matrices addition



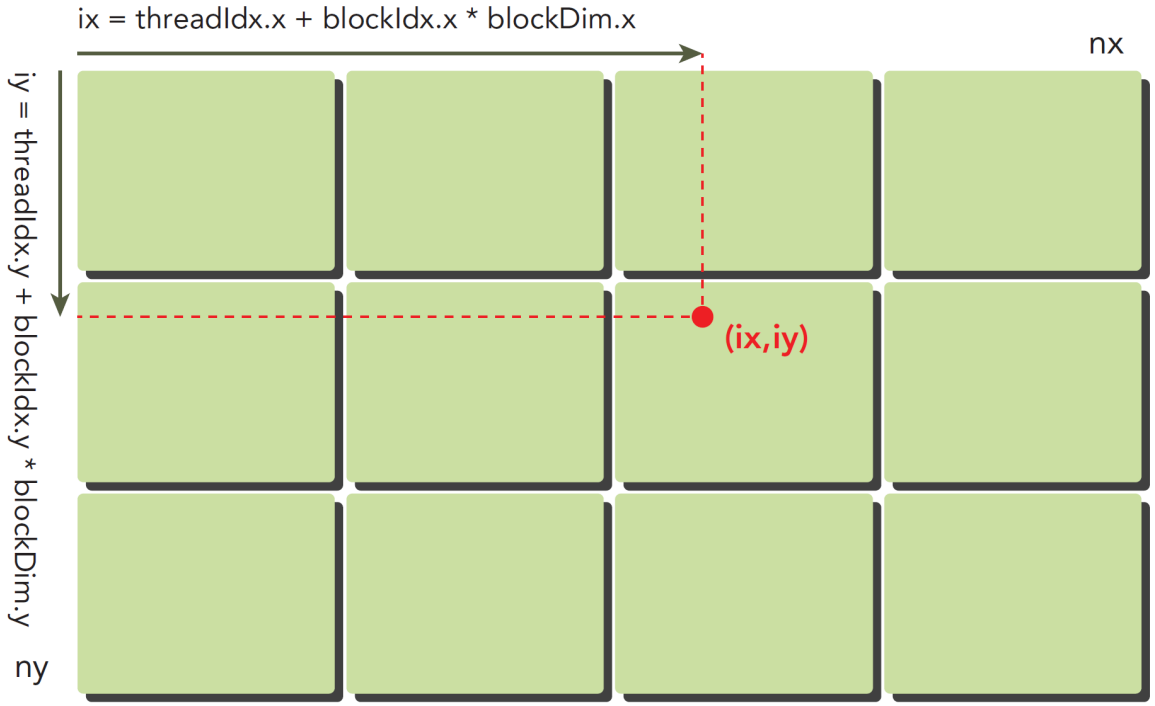
Mapping from thread IDs and Block IDs to memory index

address




```
ix = threadIdx.x + blockIdx.x * blockDim.x
```

10 = 3 + 1 * 8




```cpp
__global__ void vectorAdd(int *A, int *B, int *C, int n) {
    int ix = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < n) {
        C[ix] = A[ix] + B[ix];
    }
}
```

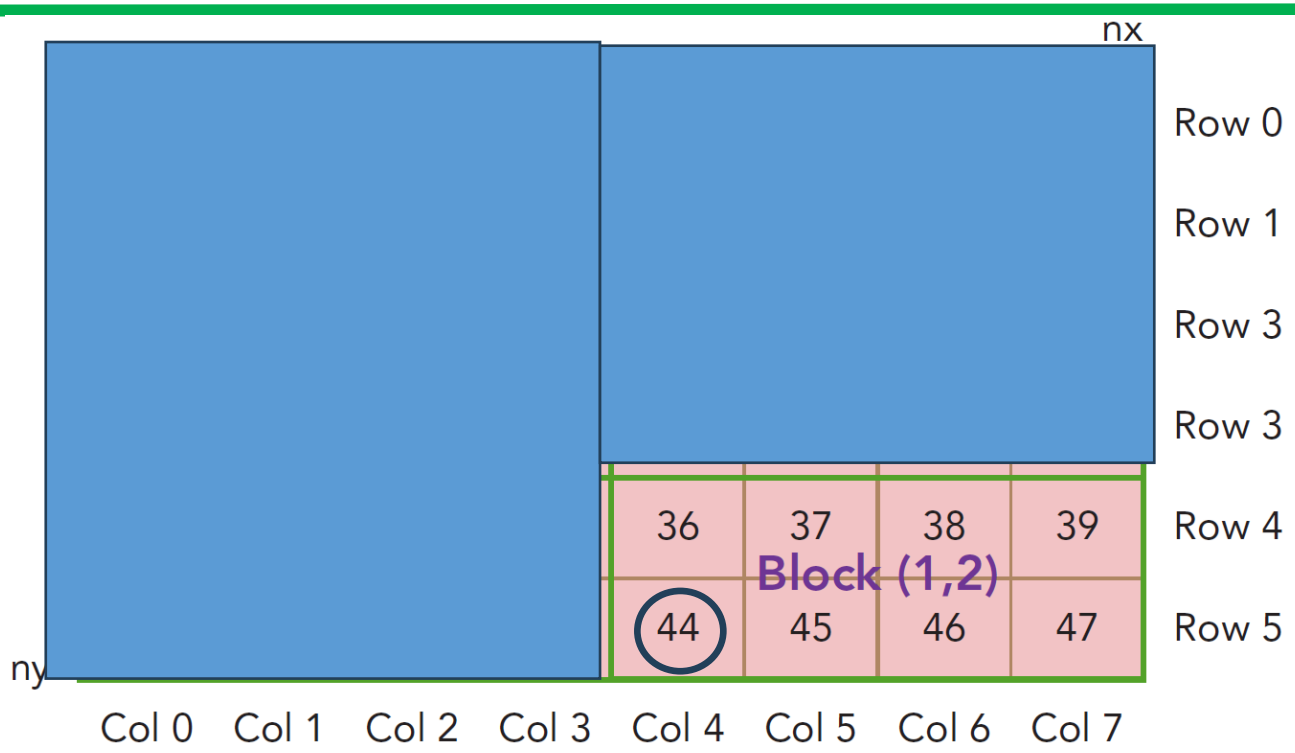
```
ix = threadIdx.x + blockIdx.x * blockDim.x
iy = threadIdx.y + blockIdx.y * blockDim.y

idx = iy * nx + ix
```

Mapping from thread IDs and Block IDs to memory index

We have 3 things to understand

- Matrix x and y values.
  - Rows = 6 (0 to 5)
  - Cols = 8 (0 to 7)
- 2D Block and 2D thread indices
  - 2D_Grid_size = Blocks = 3,2 (6 blocks)
  - 2D_Block_size = T_per_block = 2*4 (8 threads)
  - dimx=4 – dimy=2
- Memory address (index)
  - 48 elements (0 to 47)




```
ix = threadIdx.x + blockIdx.x * blockDim.x
```

ix =

```
iy = threadIdx.y + blockIdx.y * blockDim.y
```

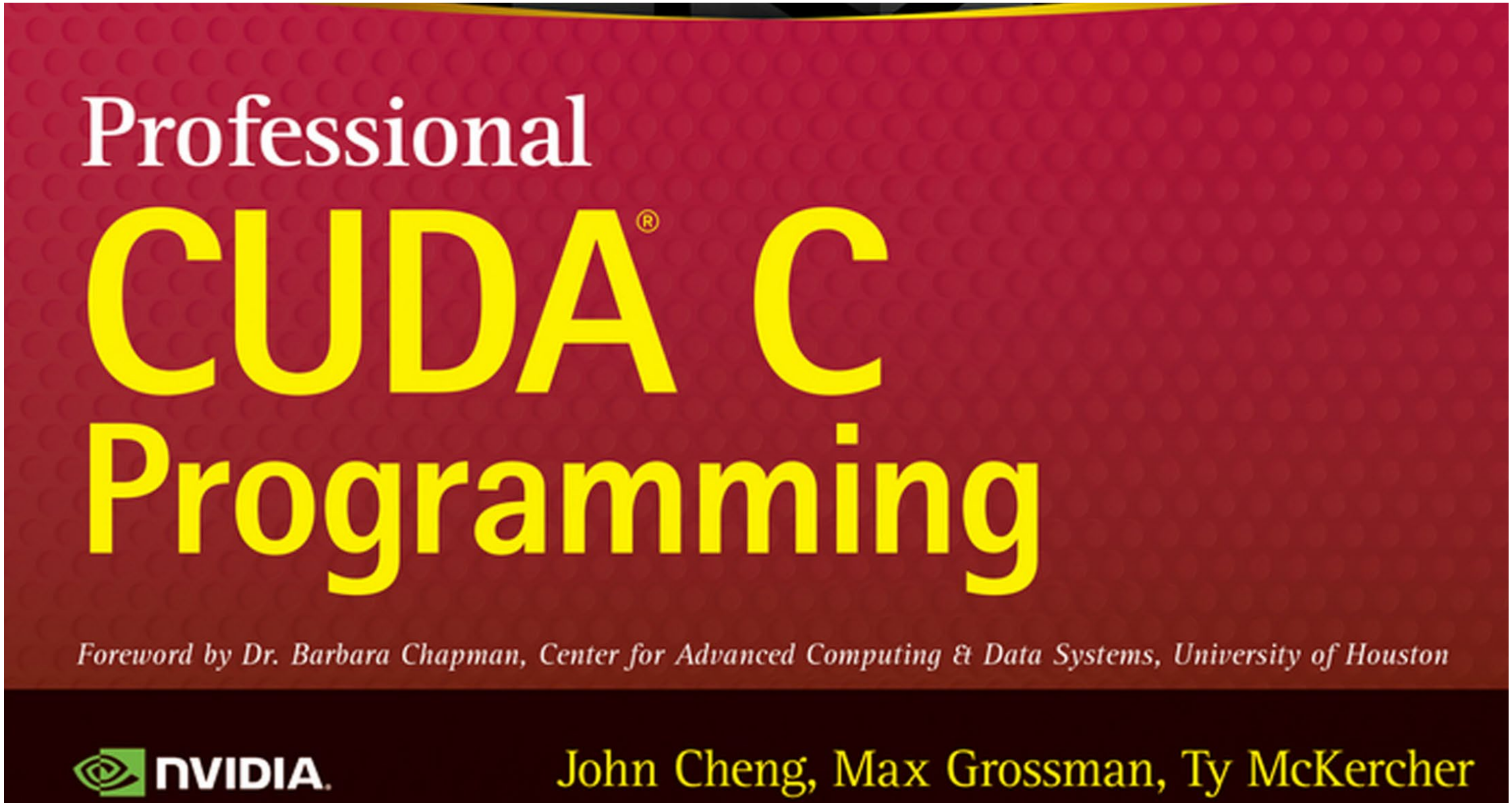
iy =

```
idx = iy * nx + ix
```

idx =

We need TIDs and BIDs to get ix and iy
We need the matrix size (number of cols) to get the memory index

Mapping from thread IDs and Block IDs to memory index



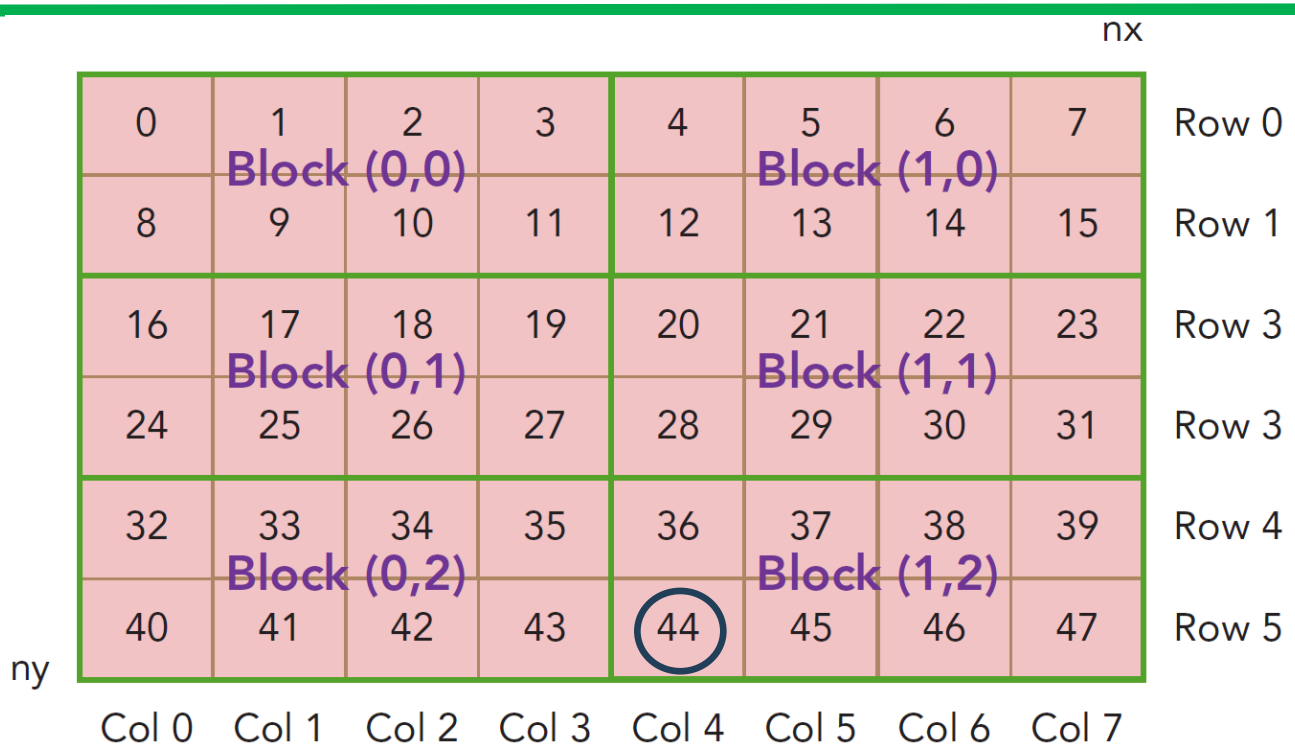
Mapping from thread IDs and Block IDs to memory index

Hits happen when we access same location.
A[0] will be accessed once and the same applies to all elements.
Because this addition not matrix multiplication.

Mapping from thread IDs and Block IDs to memory index

We have 3 things to understand

- Matrix x and y values.
  - Rows = 6 (0 to 5)
  - Cols = 8 (0 to 7)
- 2D Block and 2D thread indices
  - 2D_Grid_size = Blocks = 3,2 (6 blocks)
  - 2D_Block_size = T_per_block = 2*4 (8 threads)
  - dimx=4 – dimy=2
- Memory address (index)
  - 48 elements (0 to 47)




```
ix = threadIdx.x + blockIdx.x * blockDim.x
```

ix =

```
iy = threadIdx.y + blockIdx.y * blockDim.y
```

iy =

```
idx = iy * nx + ix
```

idx =

We need TIDs and BIDs to get ix and iy
We need the matrix size (number of cols) to get the memory index

```
// Define block and grid dimensions
dim3 blockDim(32, 32);  // 16x16 threads per block
dim3 gridDim((M + blockDim.x - 1) / blockDim.x, (N + blockDim.y - 1) / blockDim.y);
```

Matrix size = 4096 * 4096

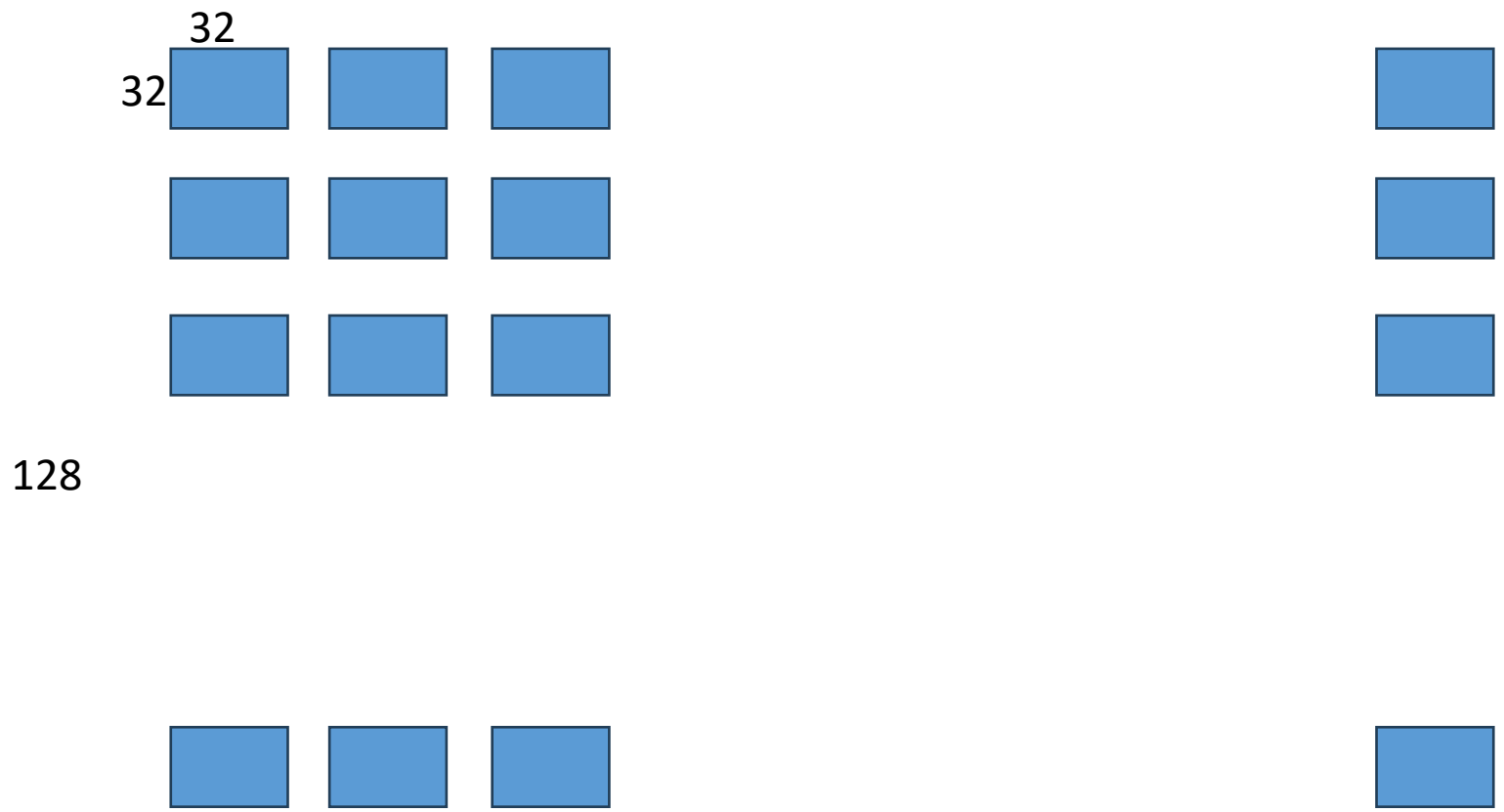
```
matrixAdd(float *, float *, float *, int, int) (128, 128, 1)x(32, 32, 1),
```

Why L1 has 0 hitrate ?

L1 has a cache line of 128 bytes.

All cache lines will be missed in l1 cache.

It is per cache line not element because each warp will execute the load operation for the 32 threads in parallel, so they are all considered misses.



- Matrix x and y values.
  - Rows = 6 (0 to 5)
  - Cols = 8 (0 to 7)
- 2D Block and 2D thread indices
  - 2D_Grid_size = griddim = 3,2 (6 blocks)
  - 2D_Block_size = blockdim = 1*64 (64 threads)
  - dimx=4 – dimy=2
- Memory address (index)
  - 48 elements (0 to 47)

128

128