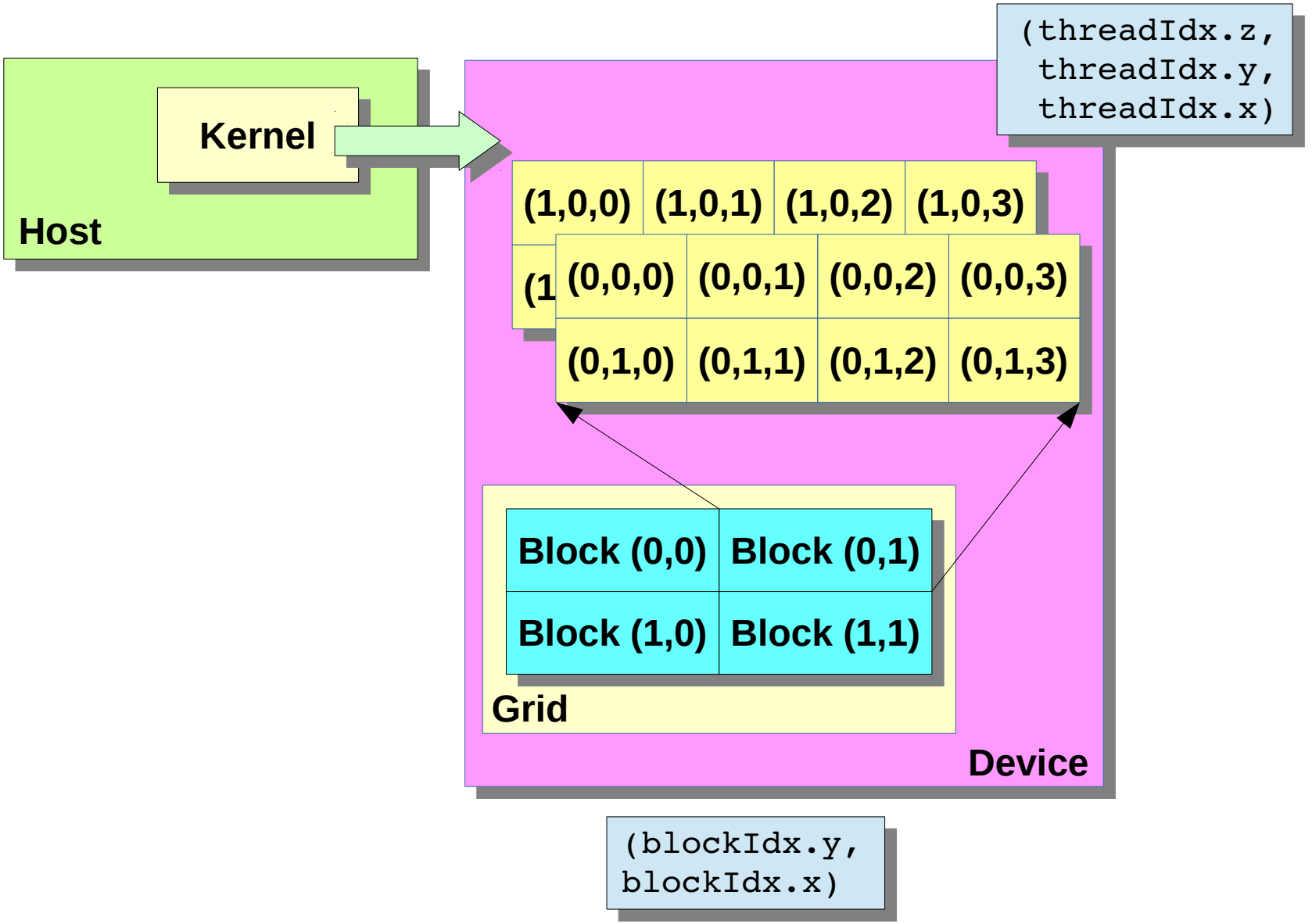
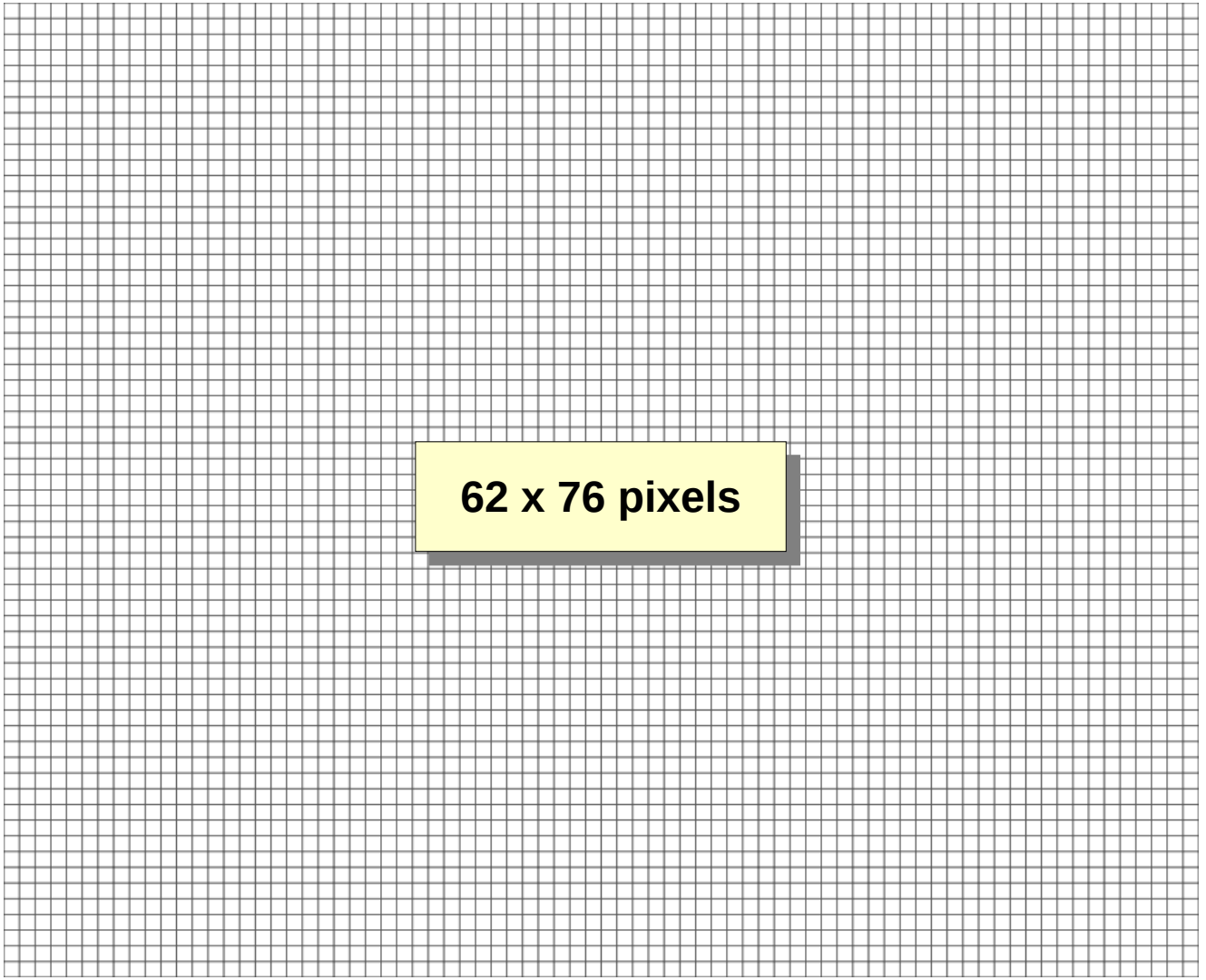
Multidimensional Kernels

Multi-Dimensional Grid

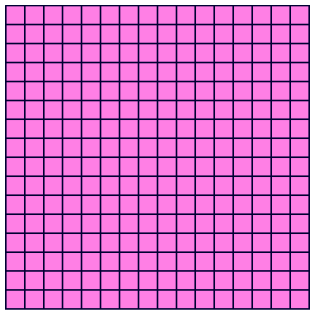


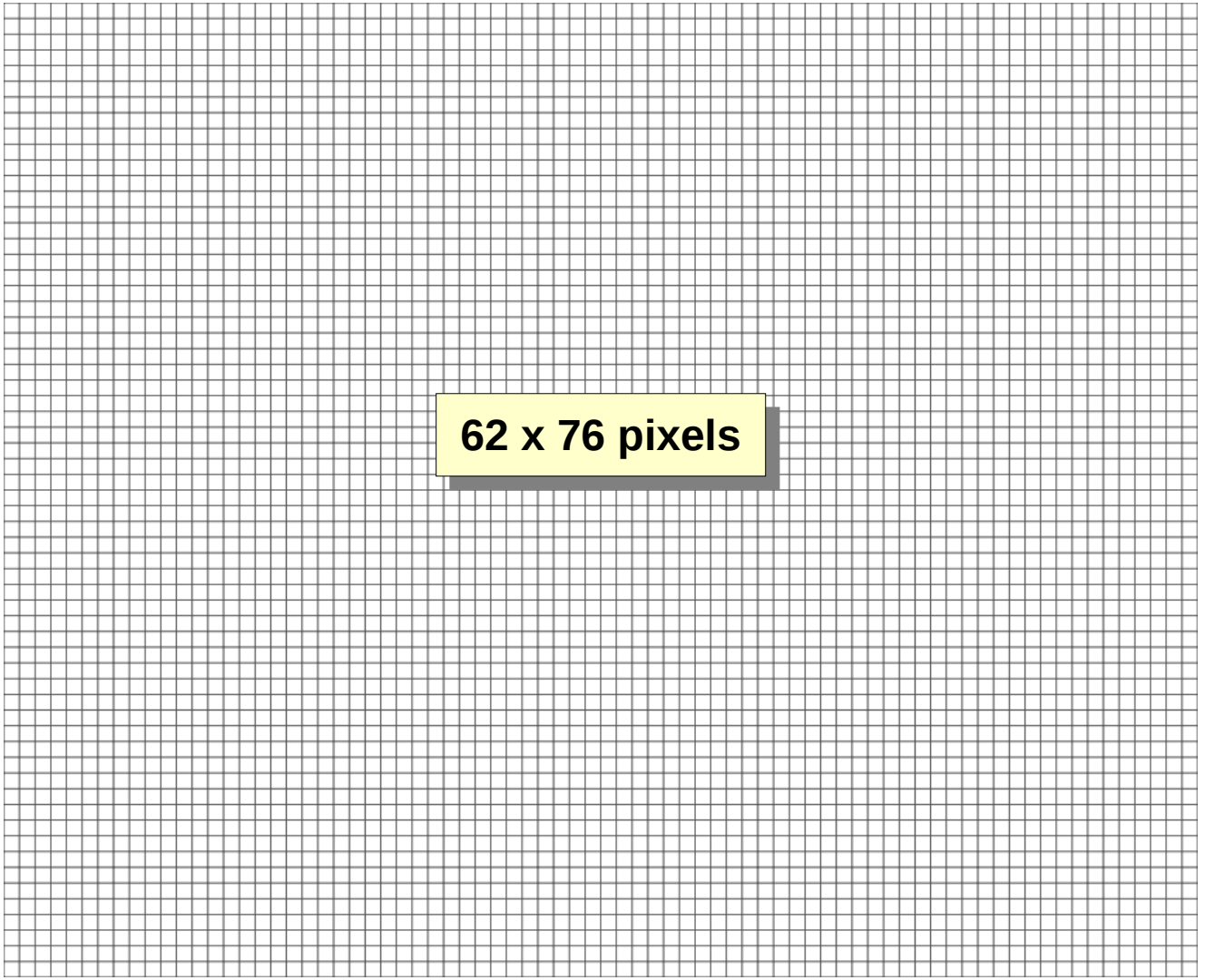
Processing a Picture with a 2D Grid



Processing a Picture with a 2D Grid

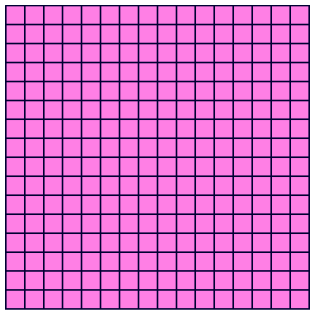
16 x 16 blocks





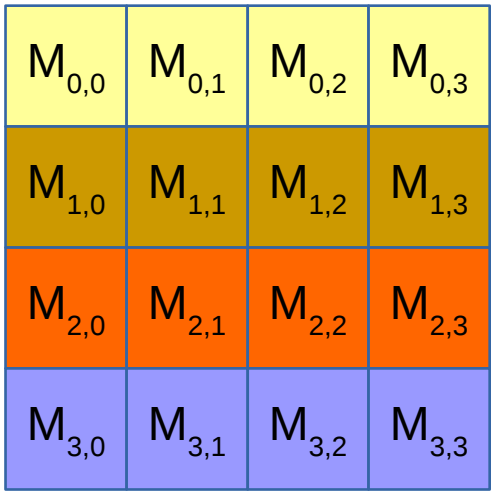
Processing a Picture with a 2D Grid

16 x 16 blocks

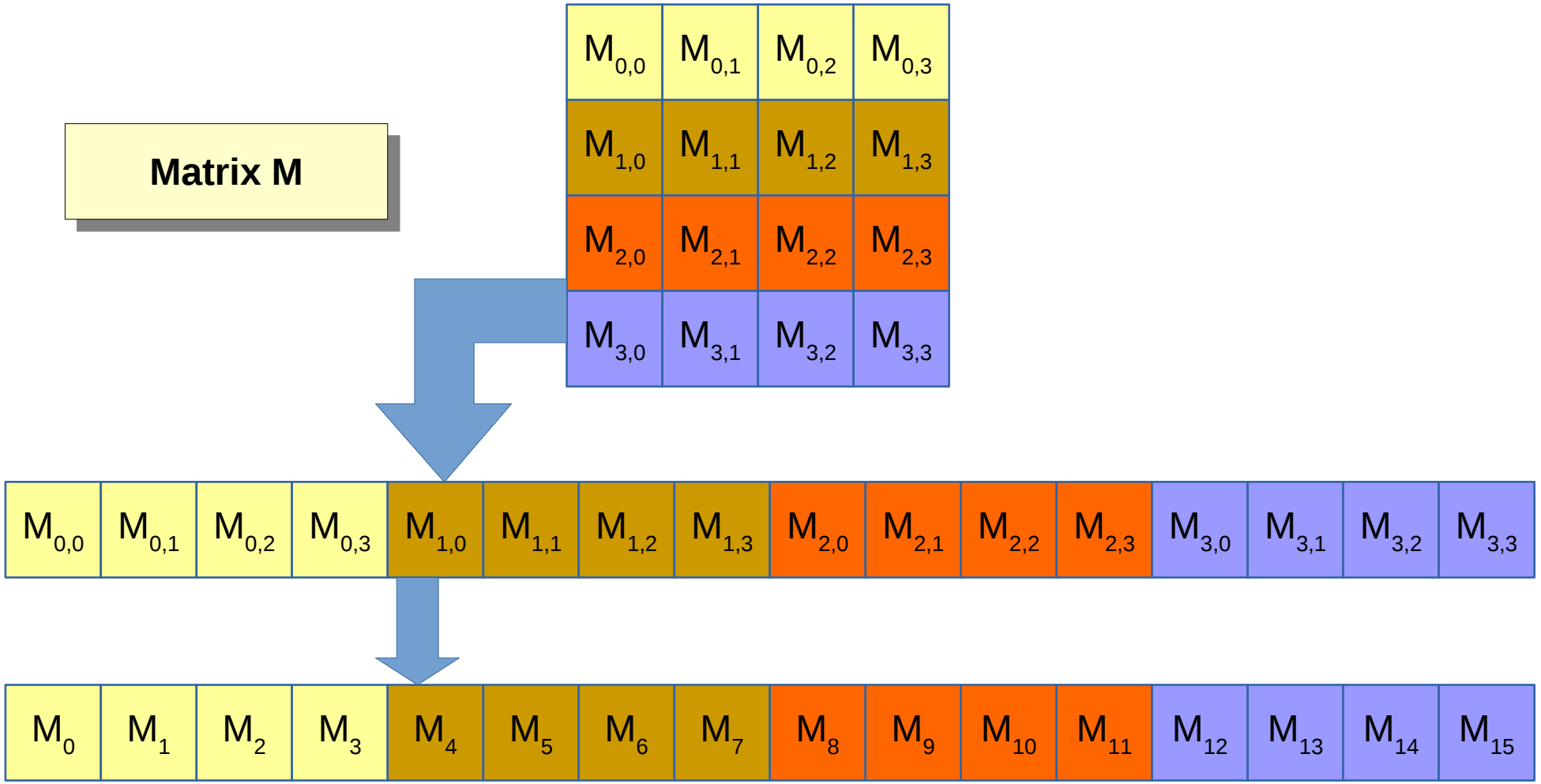


Row Major Layout in C/C++

Matrix M



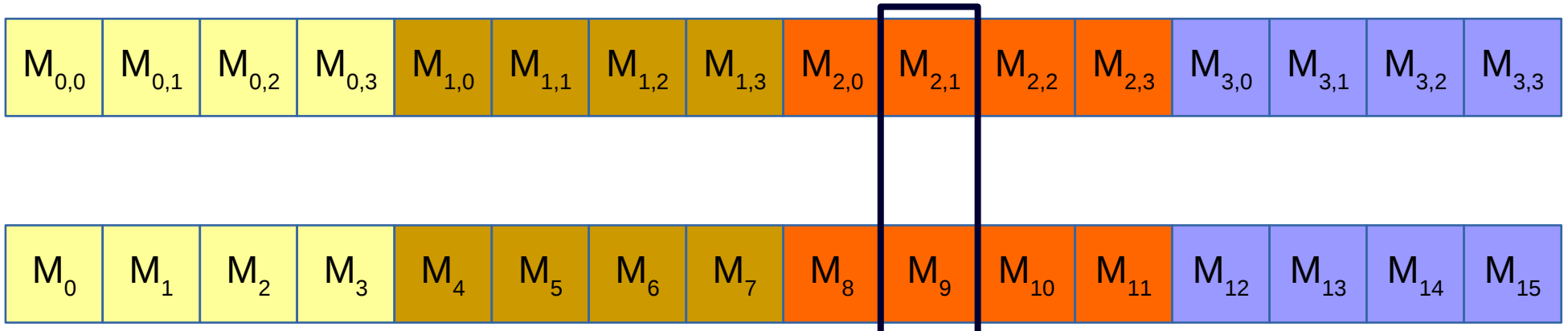
Row Major Layout in C/C++



Row Major Layout in C/C++

Matrix M

$M_{0,0}$	$M_{0,1}$	$M_{0,2}$	$M_{0,3}$
$M_{1,0}$	$M_{1,1}$	$M_{1,2}$	$M_{1,3}$
$M_{2,0}$	$M_{2,1}$	$M_{2,2}$	$M_{2,3}$
$M_{3,0}$	$M_{3,1}$	$M_{3,2}$	$M_{3,3}$



Picture Kernel Source Code

- Task: Scale every pixel value by 2.0
- Picture size = *m* pixels x *n* pixels; Thread block = 16x16 threads.

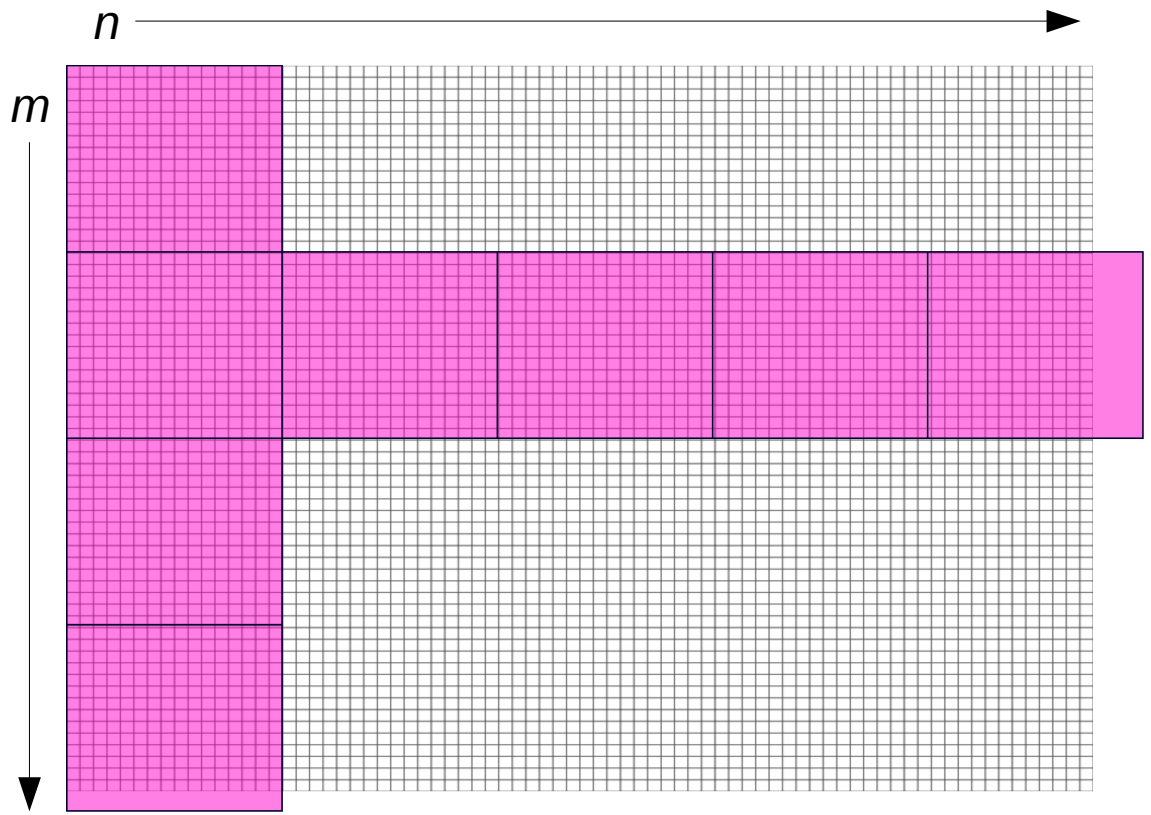
Picture Kernel Source Code

- Task: Scale every pixel value by 2.0
- Picture size = *m* pixels x *n* pixels; Thread block = 16x16 threads.

Picture Kernel Source Code

- Task: Scale every pixel value by 2.0
- Picture size = *m* pixels x *n* pixels; Thread block = 16x16 threads.

No. of thread blocks in the X and Y directions?

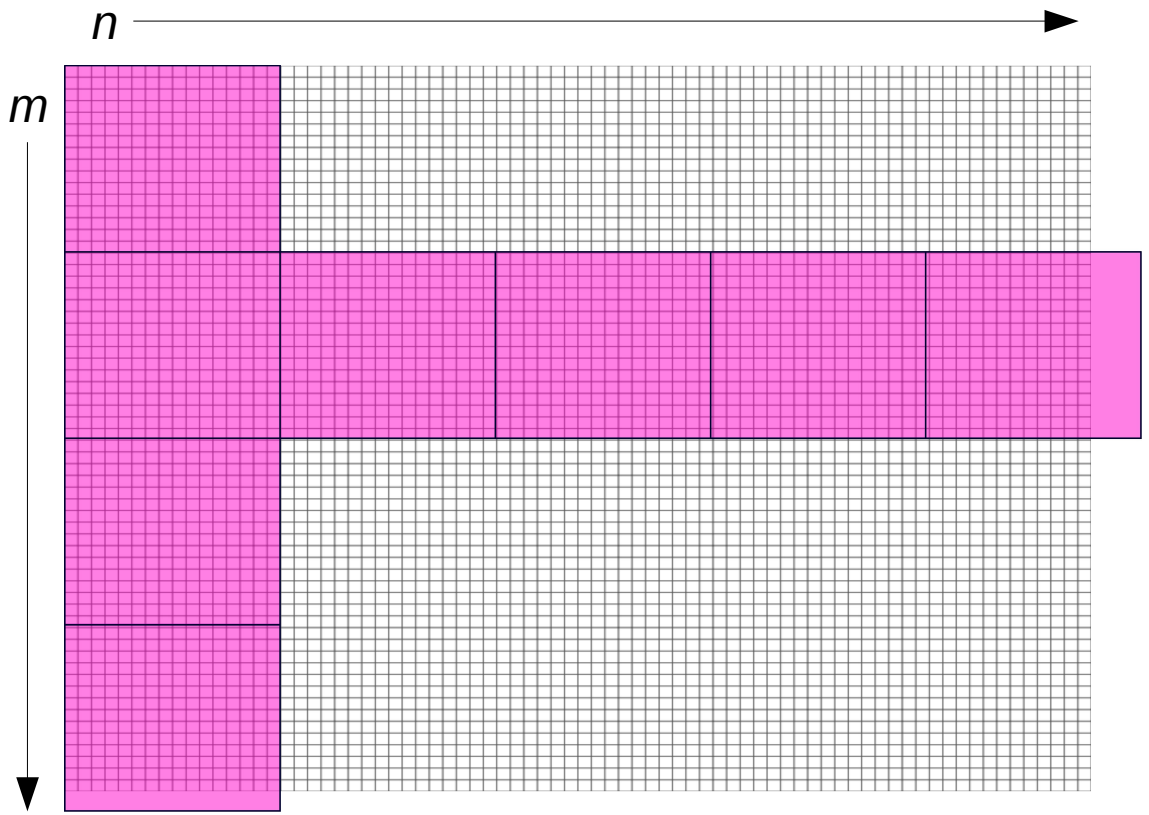


Picture Kernel Source Code

- Task: Scale every pixel value by 2.0
- Picture size = *m* pixels x *n* pixels; Thread block = 16x16 threads.

No. of thread blocks in the X and Y directions?

ceil(n/16), ceil(m/16)



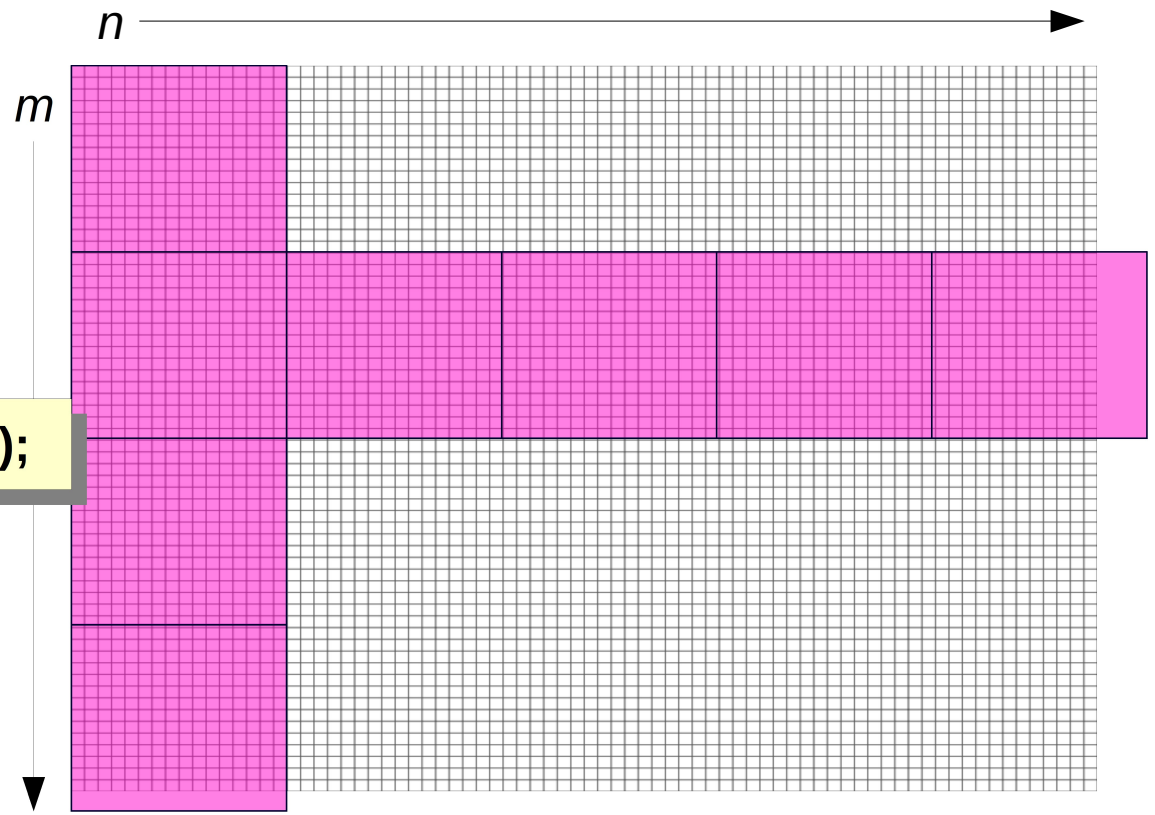
Picture Kernel Source Code

- Task: Scale every pixel value by 2.0
- Picture size = *m* pixels x *n* pixels; Thread block = 16x16 threads.

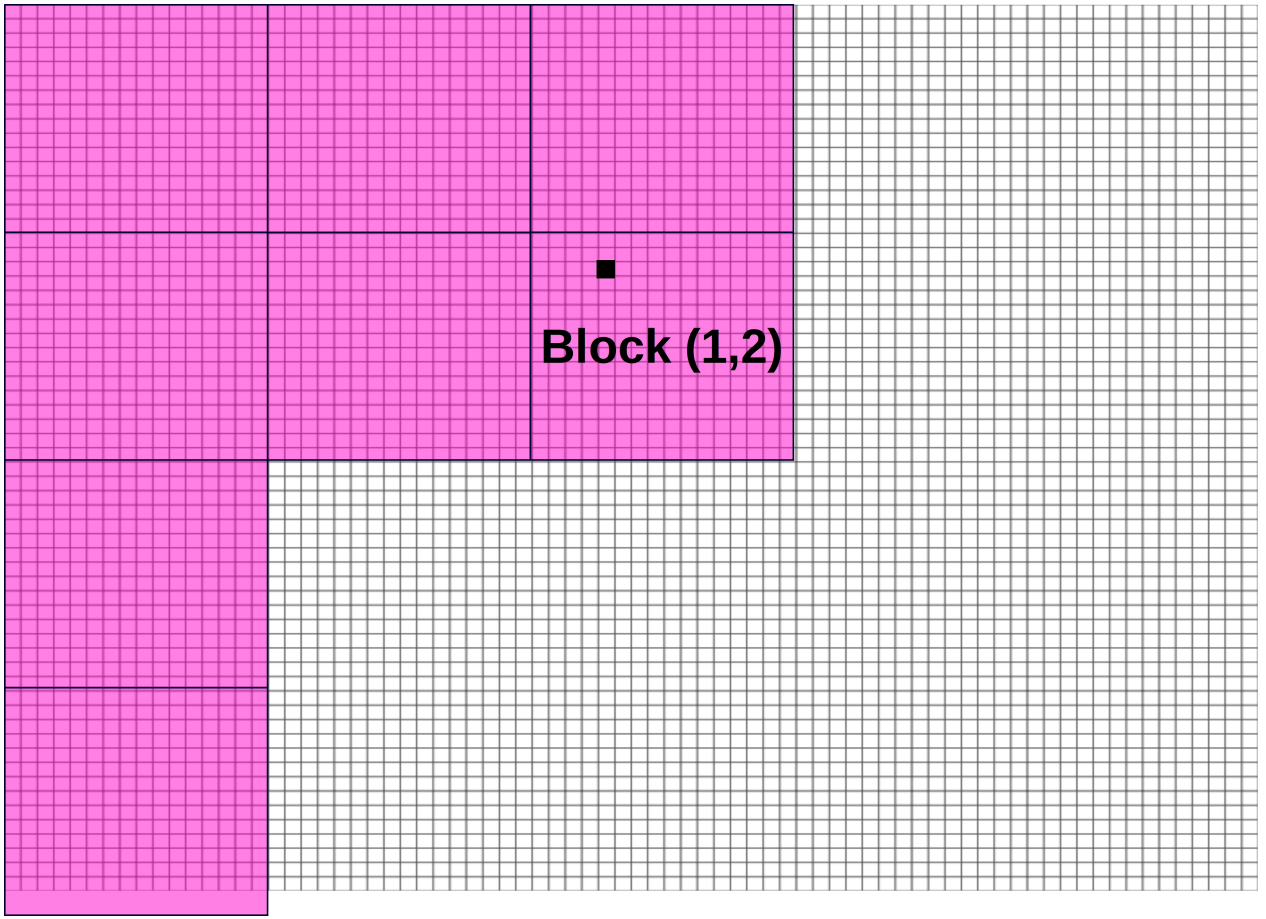
No. of thread blocks in the X and Y directions?

ceil(n/16), ceil(m/16)

```
dim3 DimGrid((n-1)/16+1, ((m-1)/16+1, 1);
```

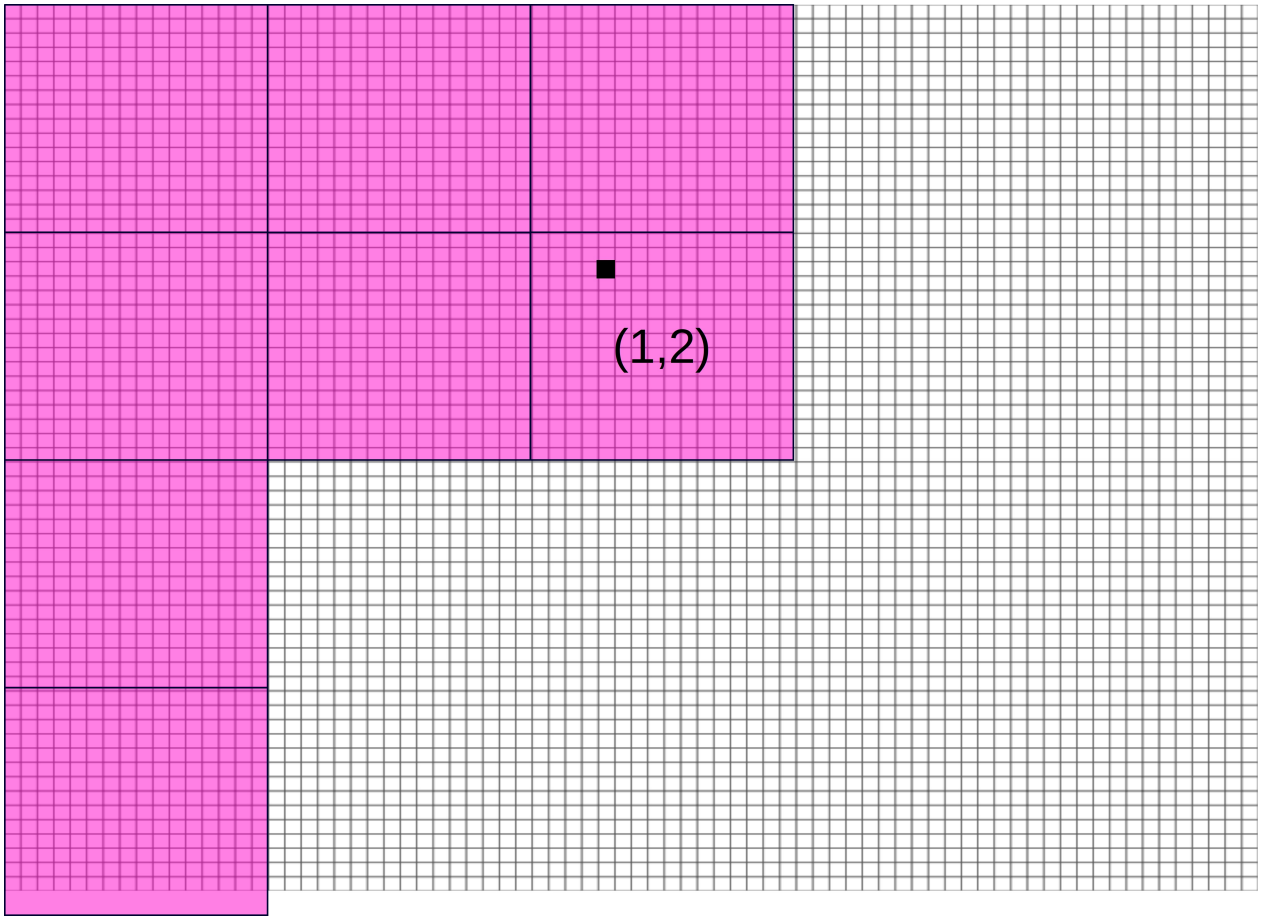


Picture Kernel Source Code



62 x 76 pixels

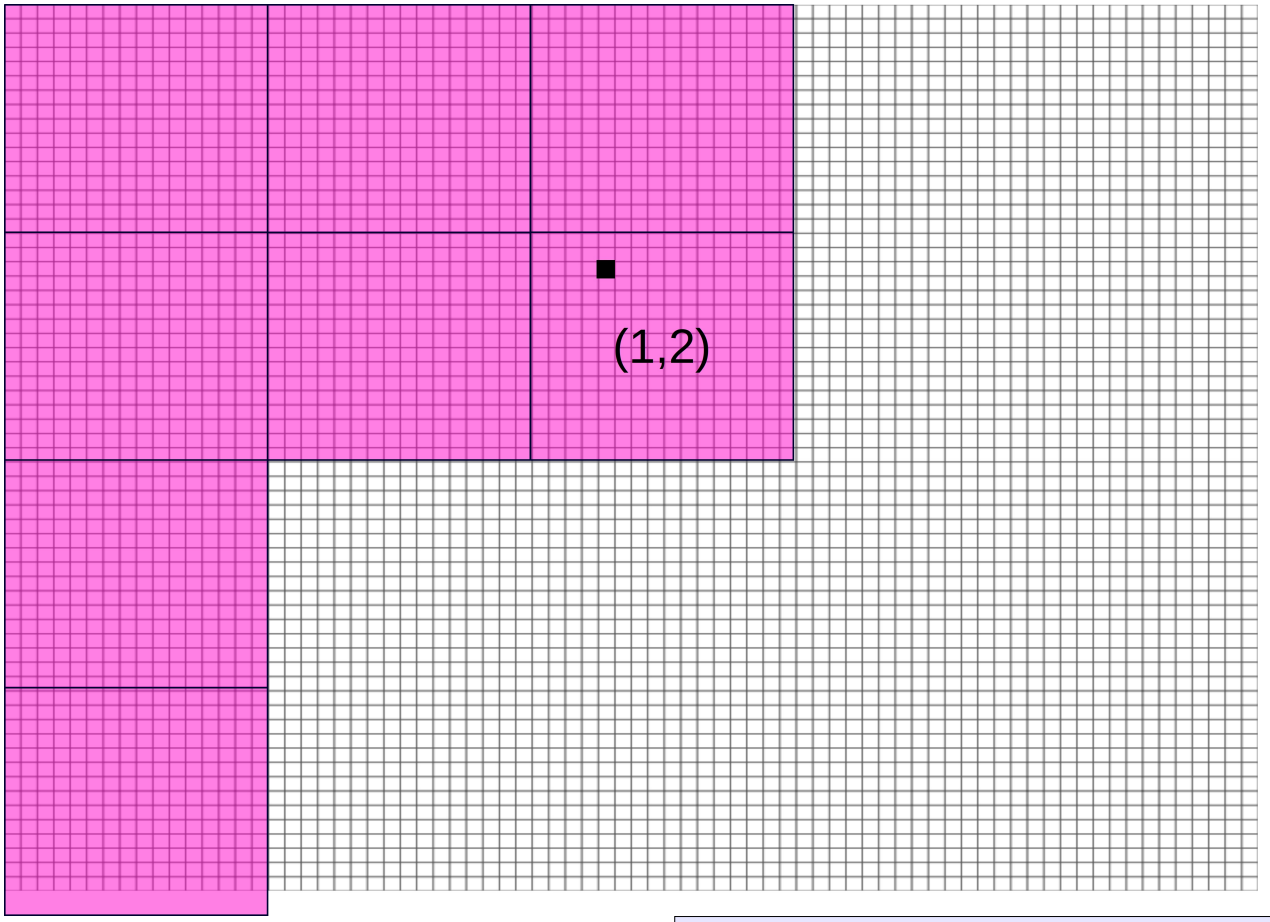
Picture Kernel Source Code



62 x 76 pixels

Which pixel should each thread work on?

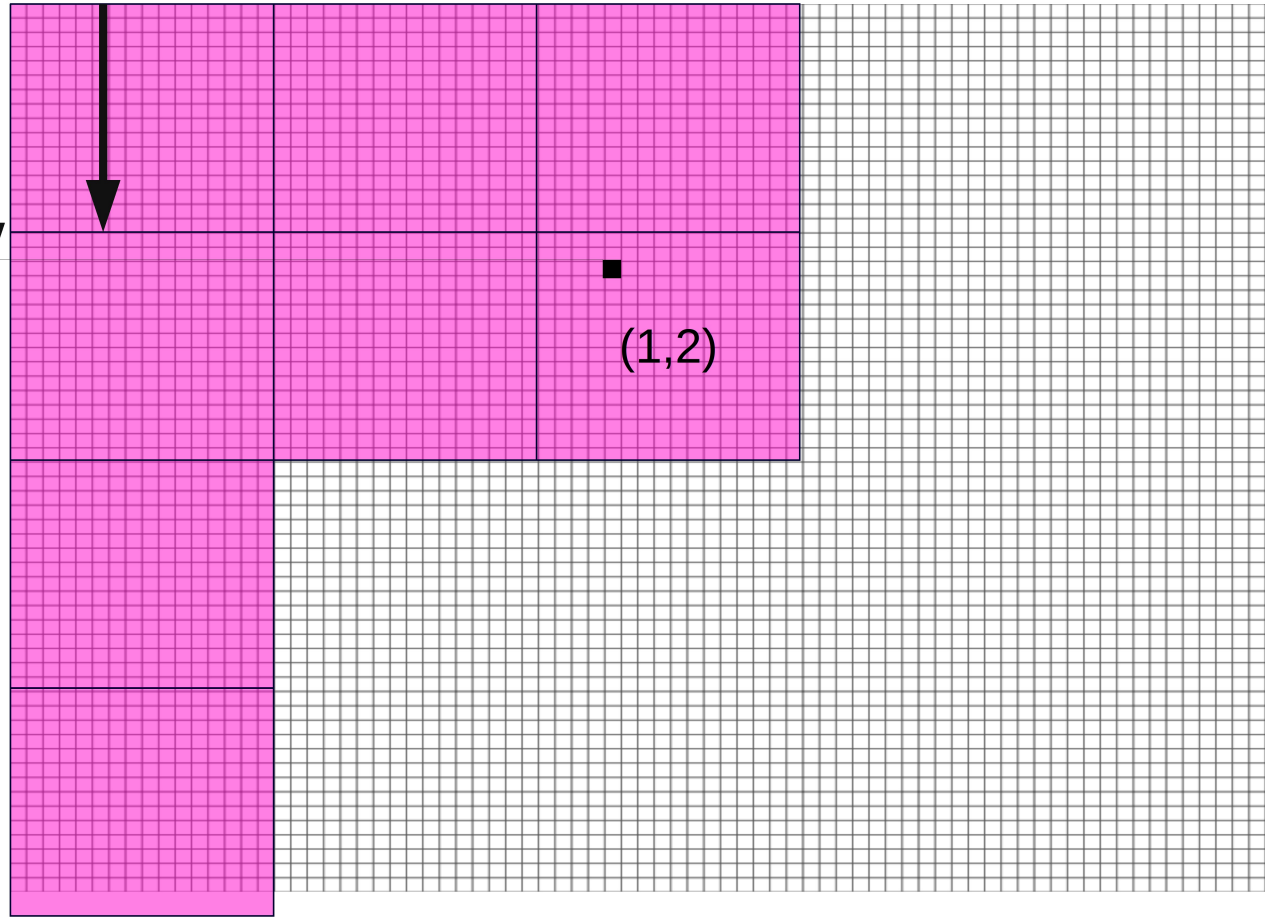
Picture Kernel Source Code



62 x 76 pixels

Thread (threadIdx.x,threadIdx.y),
in Block (blockIdx.x, blockIdx.y)
operates on which (Row,Col) pixel?

Picture Kernel Source Code



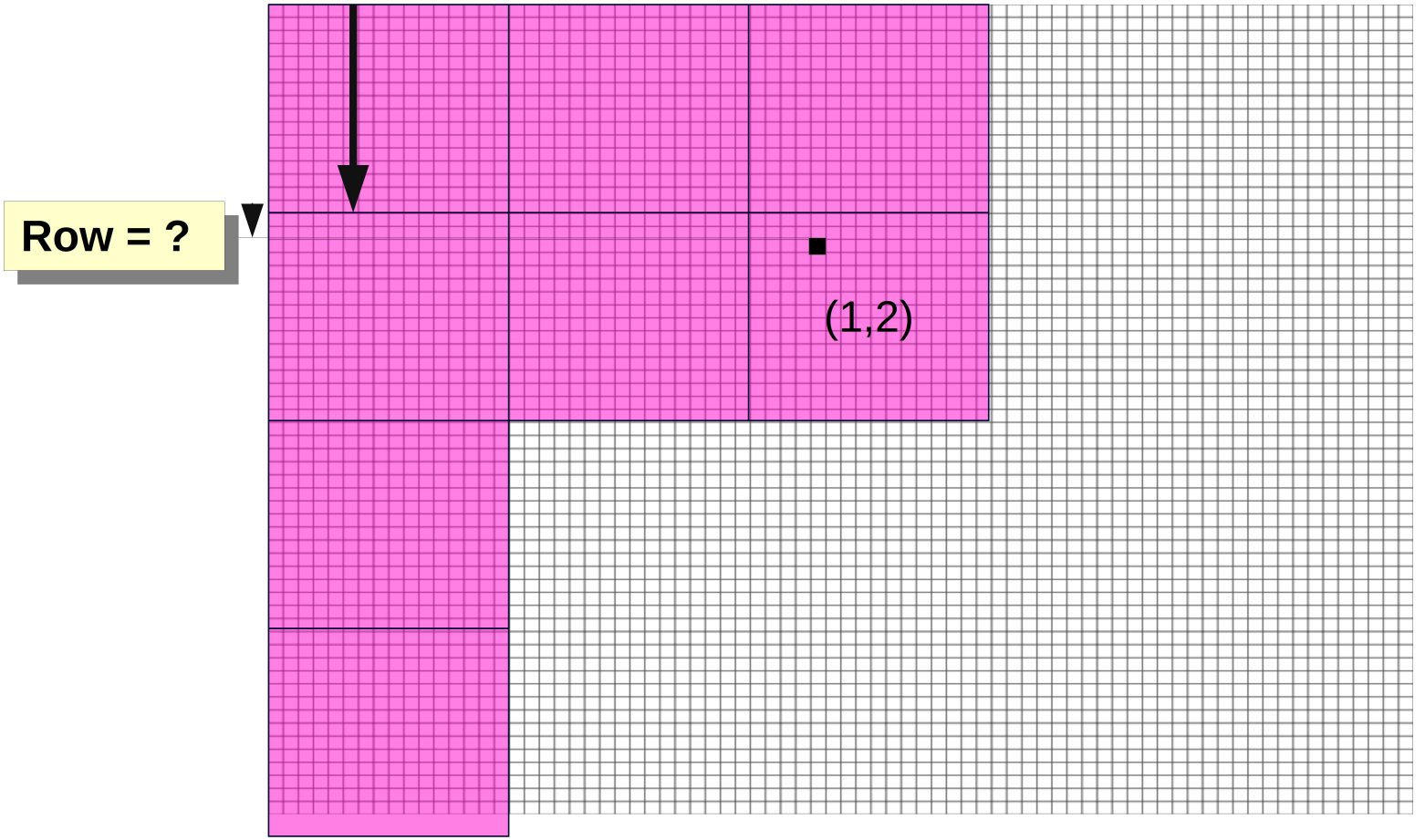
Row = ?

62 x 76 pixels

Row = ?

Thread (threadIdx.x,threadIdx.y),
in Block (blockIdx.x, blockIdx.y)
operates on which (Row,Col) pixel?

Picture Kernel Source Code

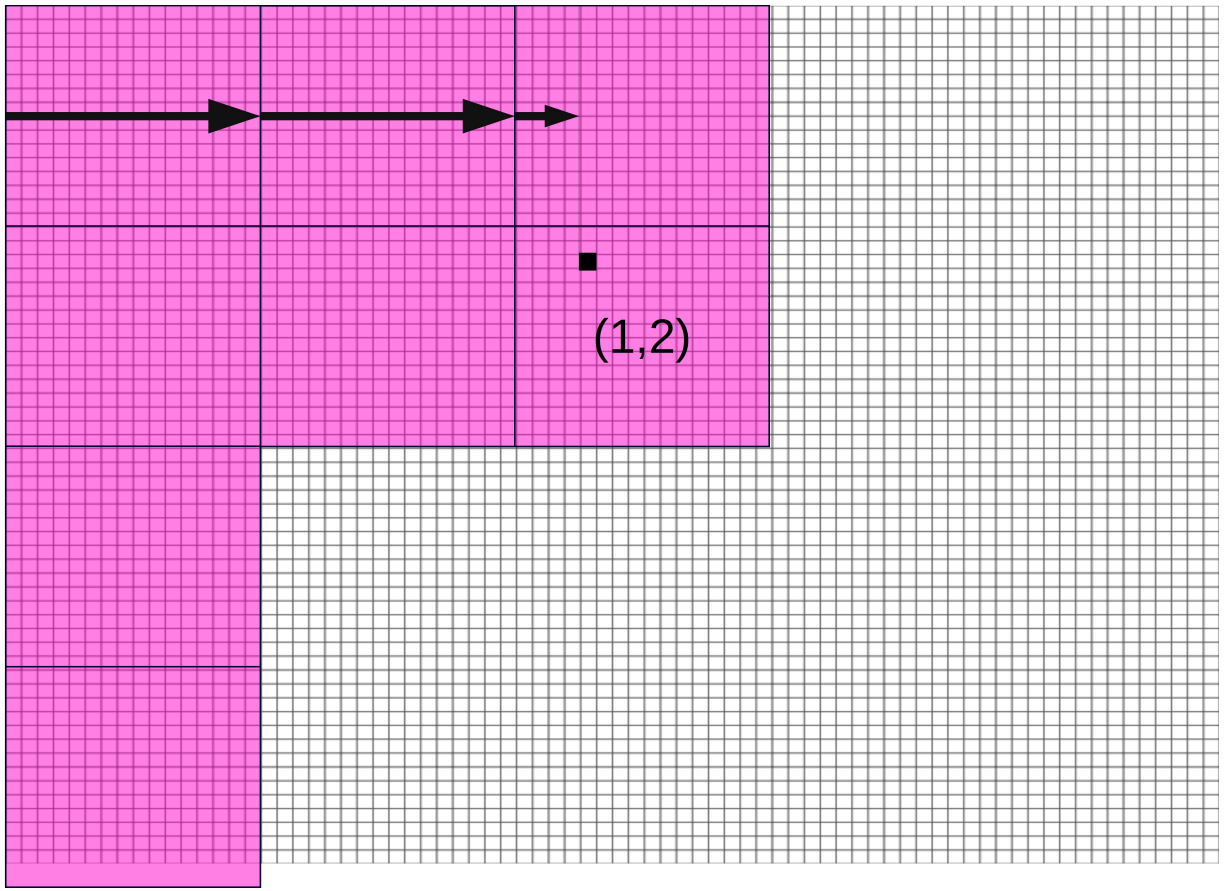


62 x 76 pixels

Row = blockIdx.y*blockDim.y + threadIdx.y;

Picture Kernel Source Code

Col = ?

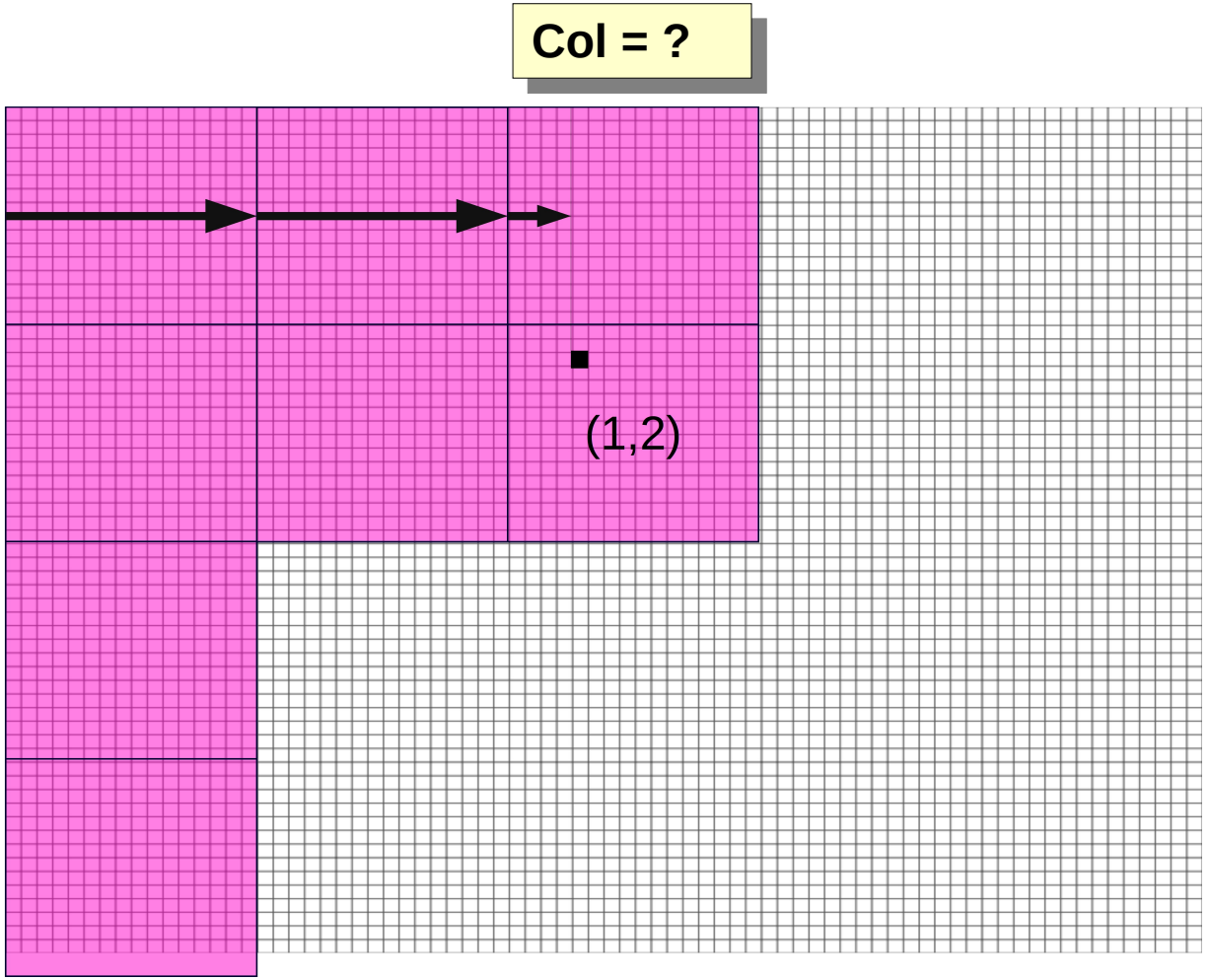


62 x 76 pixels

Row = blockIdx.y*blockDim.y + threadIdx.y;

Col = ?

Picture Kernel Source Code



62 x 76 pixels

```
Row = blockIdx.y*blockDim.y + threadIdx.y;
```

```
Col = blockIdx.x*blockDim.x + threadIdx.x;
```

Picture Kernel Source Code

```
__global__
void PictureKernel(float* d_Pin, float* d_Pout, int n, int m)
{
  ...
}
```

Task: Scale every pixel value by 2.0

Picture Kernel Source Code

```
__global__
void PictureKernel(float* d_Pin, float* d_Pout, int n, int m)
{
    // Calculate the row # of the d_Pin and d_Pout element


    // Calculate the column # of the d_Pin and d_Pout element


    // each thread computes one element of d_Pout if in range

}
```

Task: Scale every pixel value by 2.0

Picture Kernel Source Code

```
__global__
void PictureKernel(float* d_Pin, float* d_Pout, int n, int m)
{
    // Calculate the row # of the d_Pin and d_Pout element
    int Row = blockIdx.y*blockDim.y + threadIdx.y;

    // Calculate the column # of the d_Pin and d_Pout element


    // each thread computes one element of d_Pout if in range

}
```

Task: Scale every pixel value by 2.0

Picture Kernel Source Code

```
__global__
void PictureKernel(float* d_Pin, float* d_Pout, int n, int m)
{
    // Calculate the row # of the d_Pin and d_Pout element
    int Row = blockIdx.y*blockDim.y + threadIdx.y;

    // Calculate the column # of the d_Pin and d_Pout element
    int Col = blockIdx.x*blockDim.x + threadIdx.x;

    // each thread computes one element of d_Pout if in range

}
```

Task: Scale every pixel value by 2.0

Picture Kernel Source Code

```
__global__
void PictureKernel(float* d_Pin, float* d_Pout, int n, int m)
{
    // Calculate the row # of the d_Pin and d_Pout element
    int Row = blockIdx.y*blockDim.y + threadIdx.y;

    // Calculate the column # of the d_Pin and d_Pout element
    int Col = blockIdx.x*blockDim.x + threadIdx.x;

    // each thread computes one element of d_Pout if in range

       d_Pout[Row*n+Col] = 2.0*d_Pin[Row*n+Col];

}
```

Task: Scale every pixel value by 2.0

Picture Kernel Source Code

```
__global__
void PictureKernel(float* d_Pin, float* d_Pout, int n, int m)
{
    // Calculate the row # of the d_Pin and d_Pout element
    int Row = blockIdx.y*blockDim.y + threadIdx.y;

    // Calculate the column # of the d_Pin and d_Pout element
    int Col = blockIdx.x*blockDim.x + threadIdx.x;

    // each thread computes one element of d_Pout if in range
    if ((Row < m) && (Col < n)) {
        d_Pout[Row*n+Col] = 2.0*d_Pin[Row*n+Col];
    }
}
```

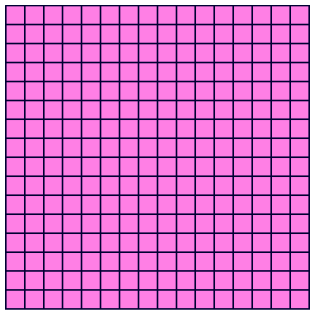
Task: Scale every pixel value by 2.0

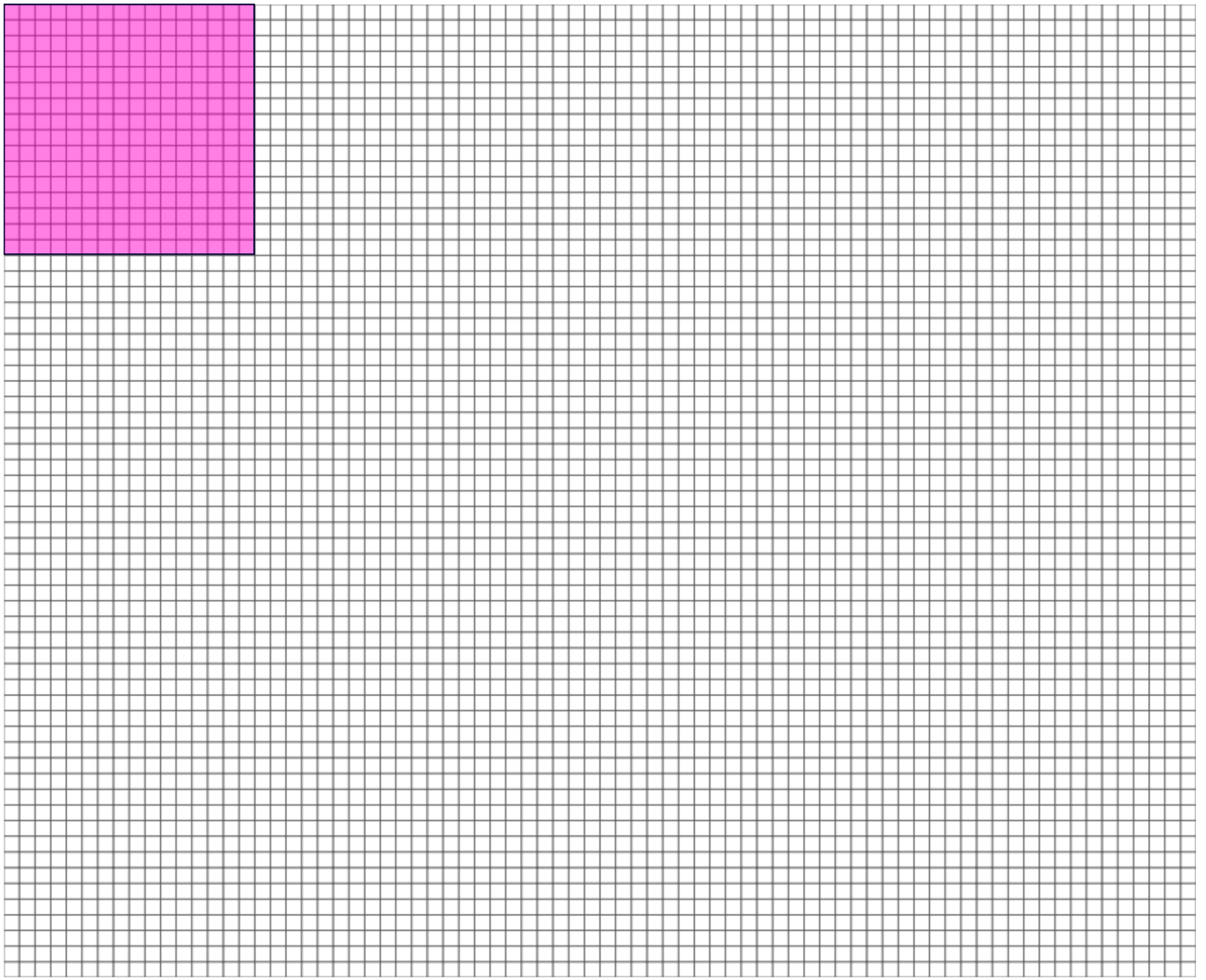
Picture Kernel Call

```
// assume that the picture is mxn,
// m pixels in Y dimension and n pixels in X dimension
// input d_Pin has been allocated on and copied to device
// output d_Pout has been allocated on device
...
dim3 DimGrid((n-1)/16 + 1, ((m-1)/16+1, 1);
dim3 DimBlock(16, 16, 1);
PictureKernel<<<DimGrid,DimBlock>>>(d_Pin, d_Pout, n, m);
...
```

Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks

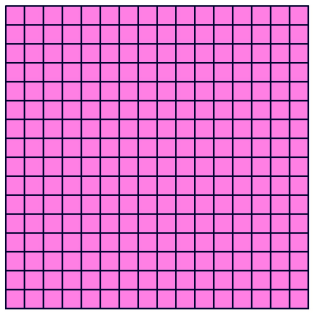




62 x 76 pixels

Covering a 62×76 Picture with 16×16 Blocks

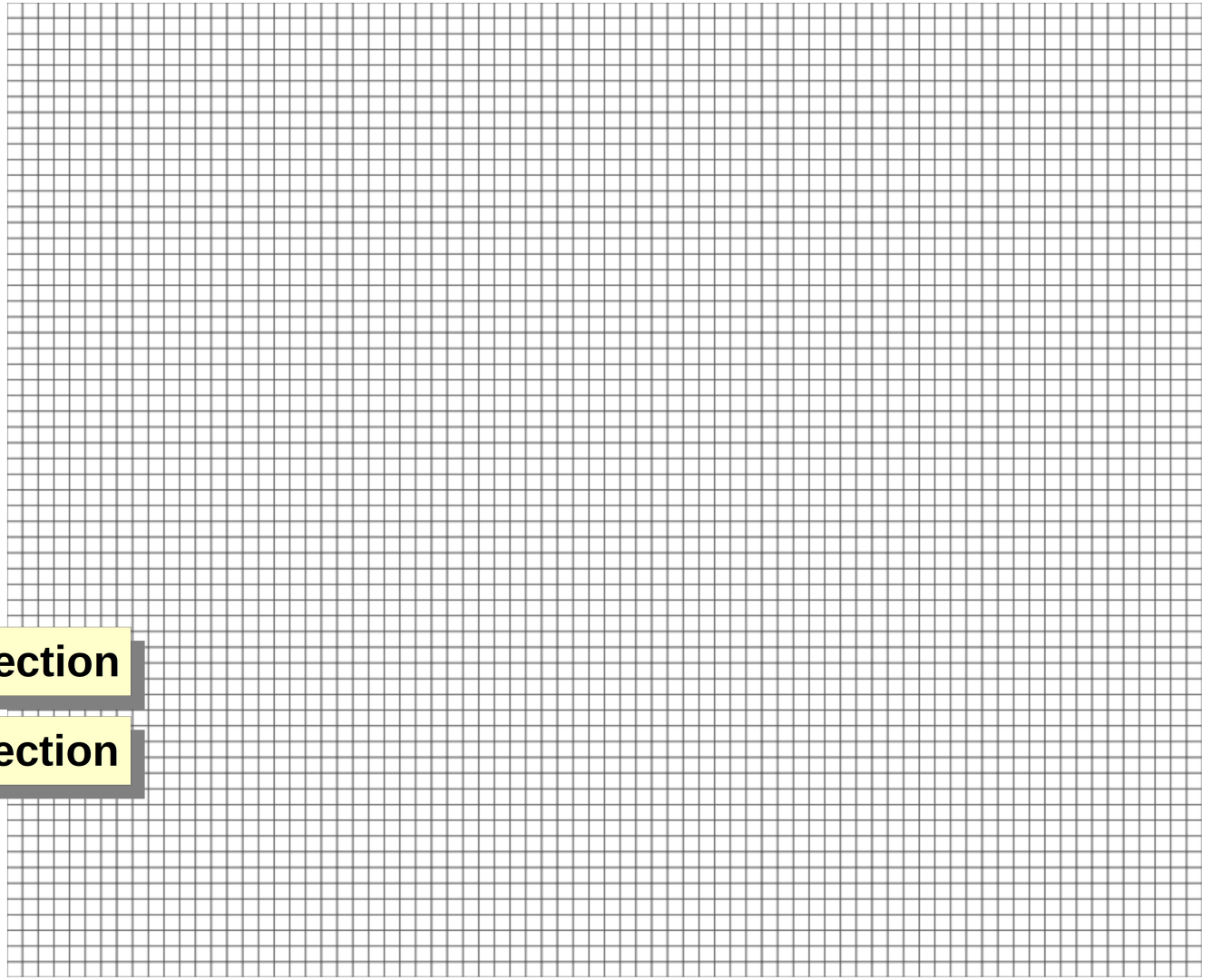
16 x 16 blocks



80 Threads in the X direction

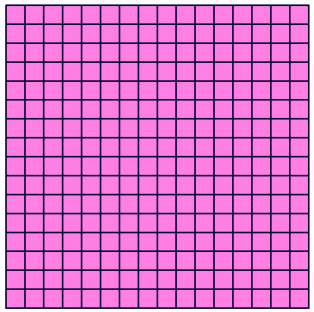
64 Threads in the Y direction

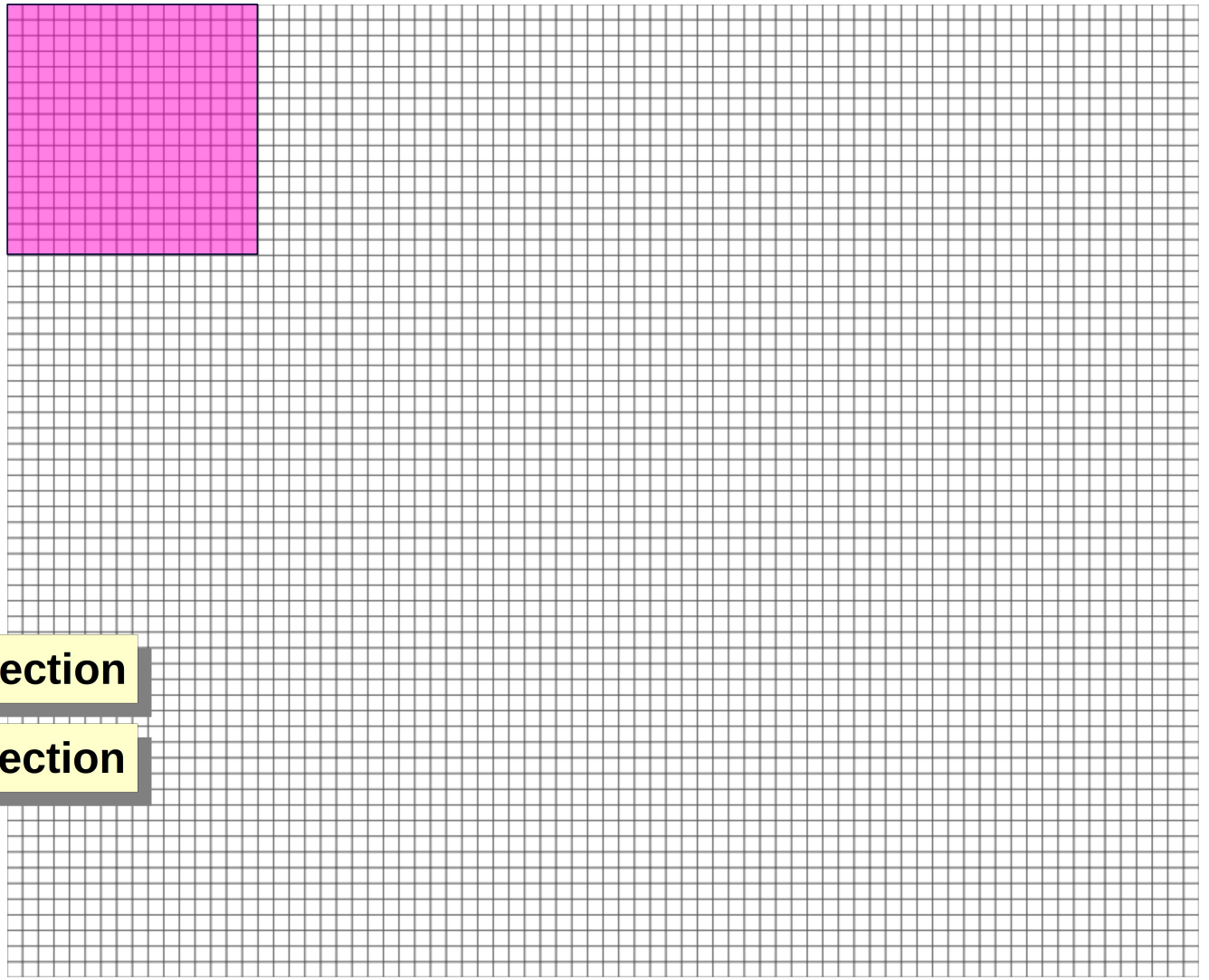
62 x 76 pixels



Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks





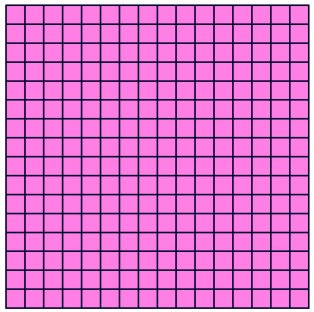
80 Threads in the X direction

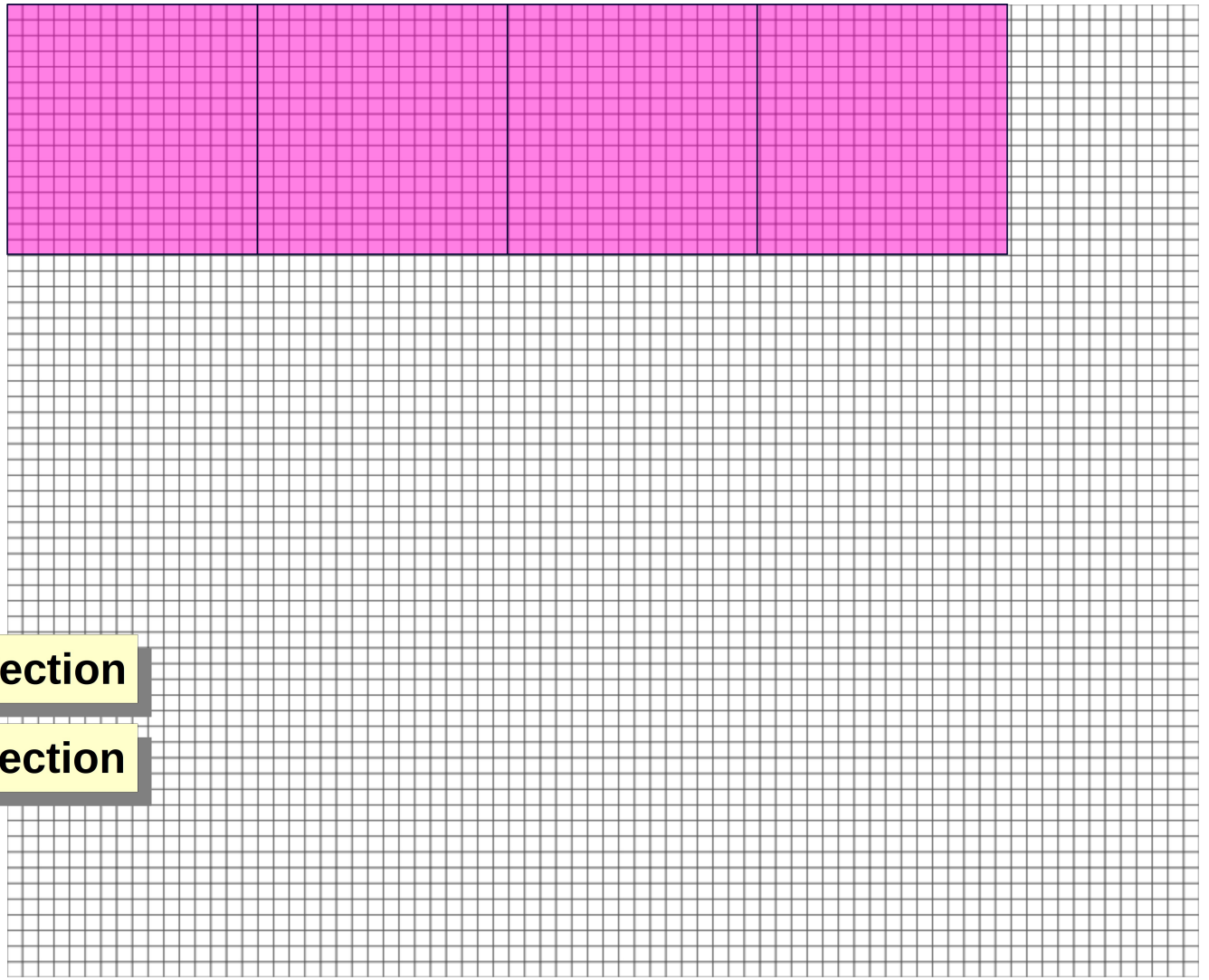
64 Threads in the Y direction

62 x 76 pixels

Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks



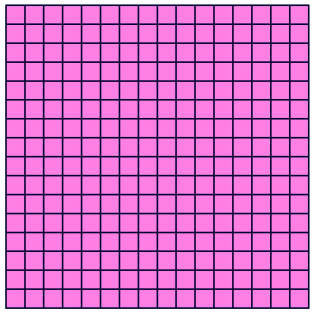


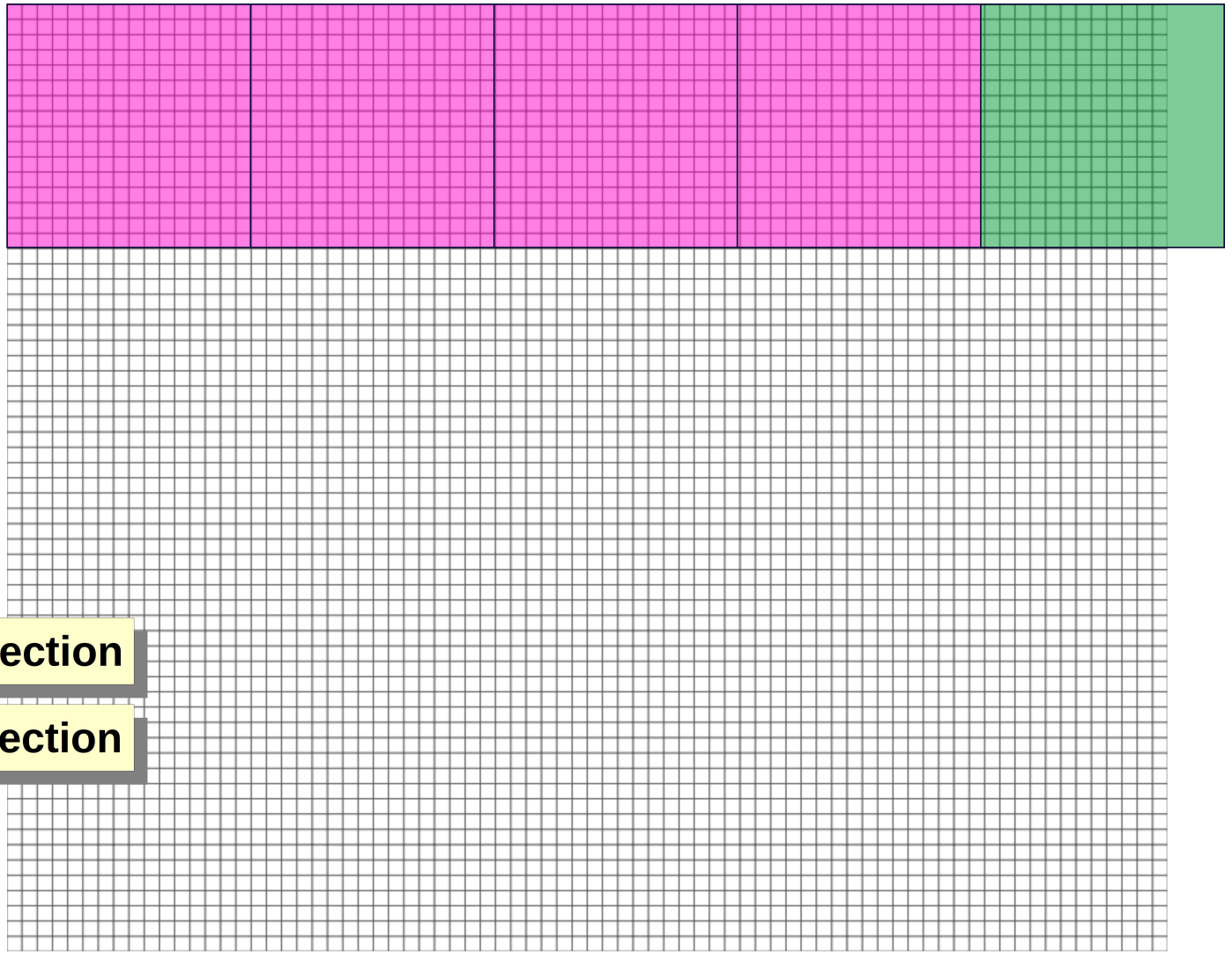
80 Threads in the X direction

64 Threads in the Y direction

Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks



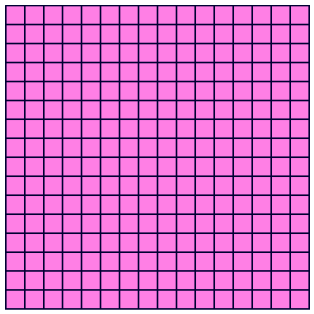


80 Threads in the X direction

64 Threads in the Y direction

Covering a 62×76 Picture with 16×16 Blocks

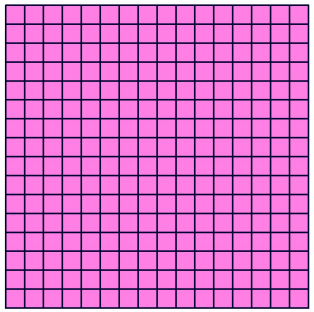
16 x 16 blocks

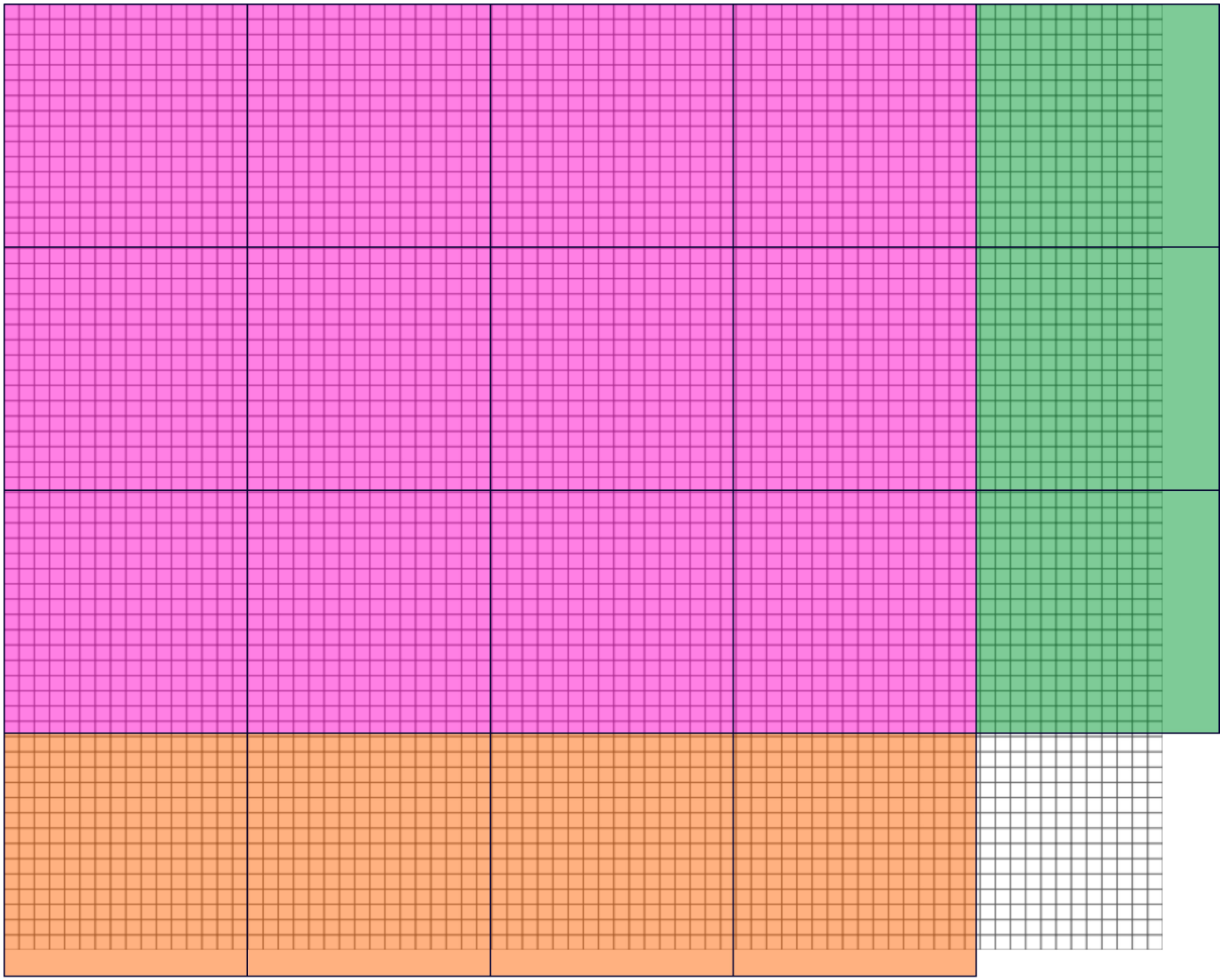


64 Threads in the Y direction

Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks

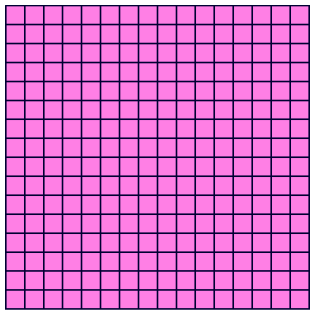


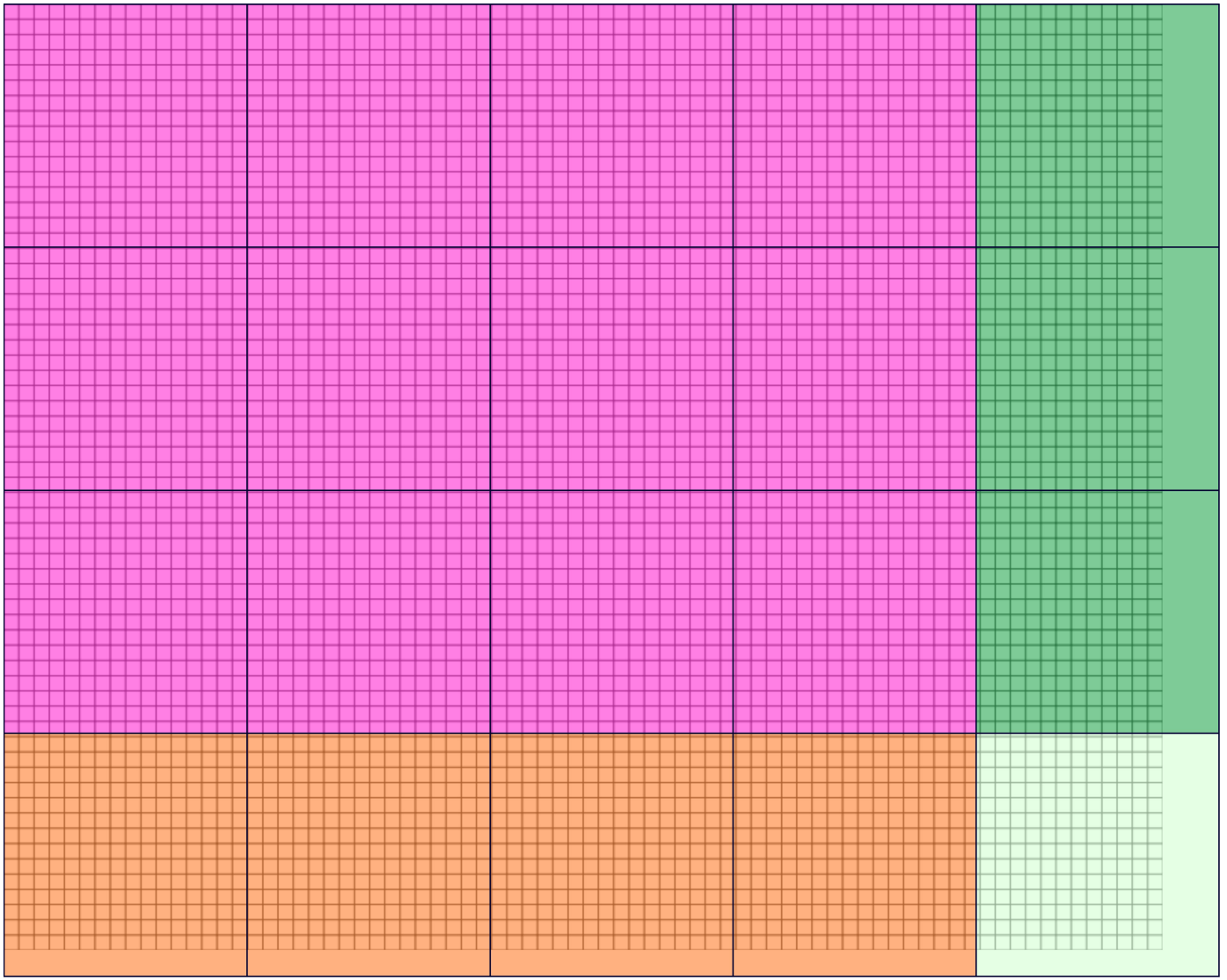


64 Threads in the Y direction

Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks





Covering a 62×76 Picture with 16×16 Blocks

16 x 16 blocks