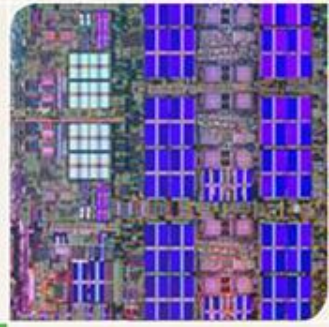
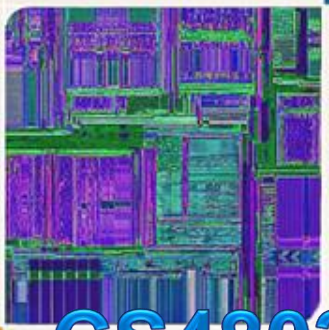





CS4803DGC Design and Programming of Game Consoles

Spring 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing



Debug

- Emulation mode
- Using CUDA-GDB
  - Not supported at Braid Lab ☹
  - All the GDB features are supported
  - Can set a break in kernel
  - Can make a progress only for a singe warp (a set of threads) {focused thread}
  - A previous semester student, Anirudh's blog on CUDA-GDB
    - http://themethodofloci.blogspot.com/

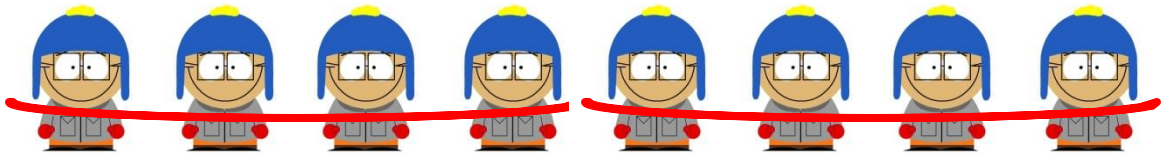


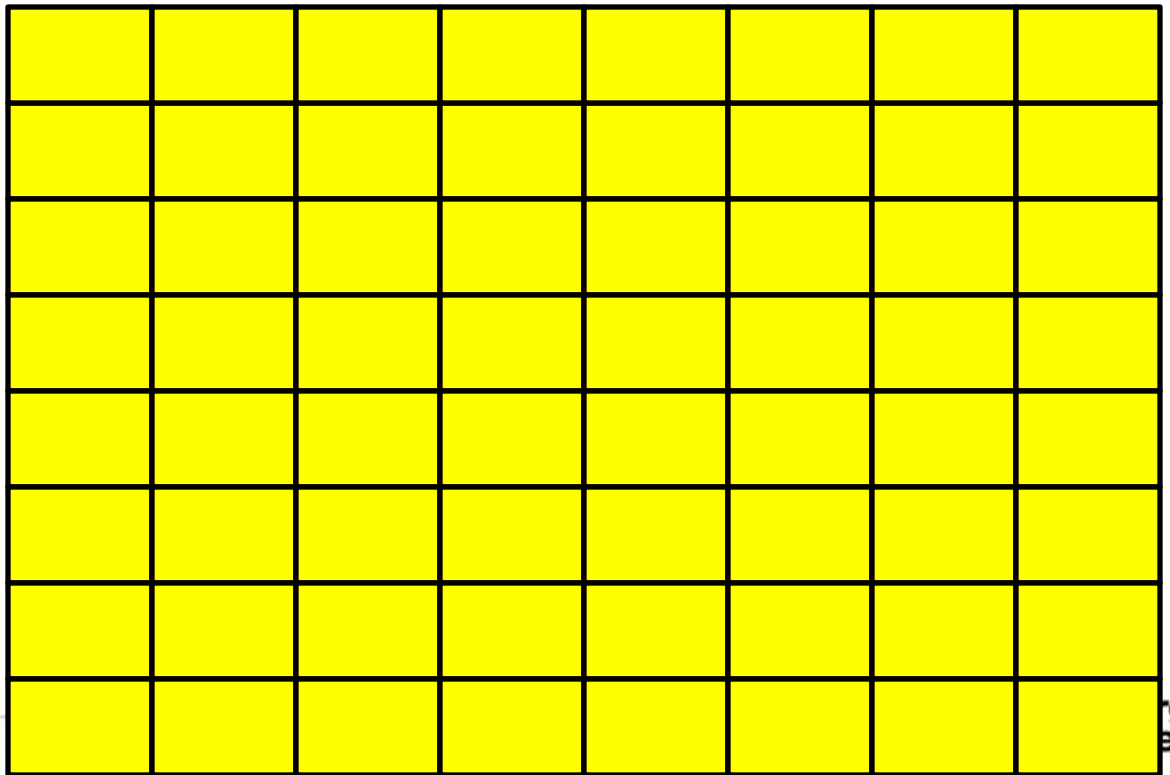
Why MatrixMultiplication?

- Vector format is basic in graphics [xyzw]
- Matrix multiplication is a basic computation
- All the images are represented with matrix

Matrix size = Multiple of block sizes

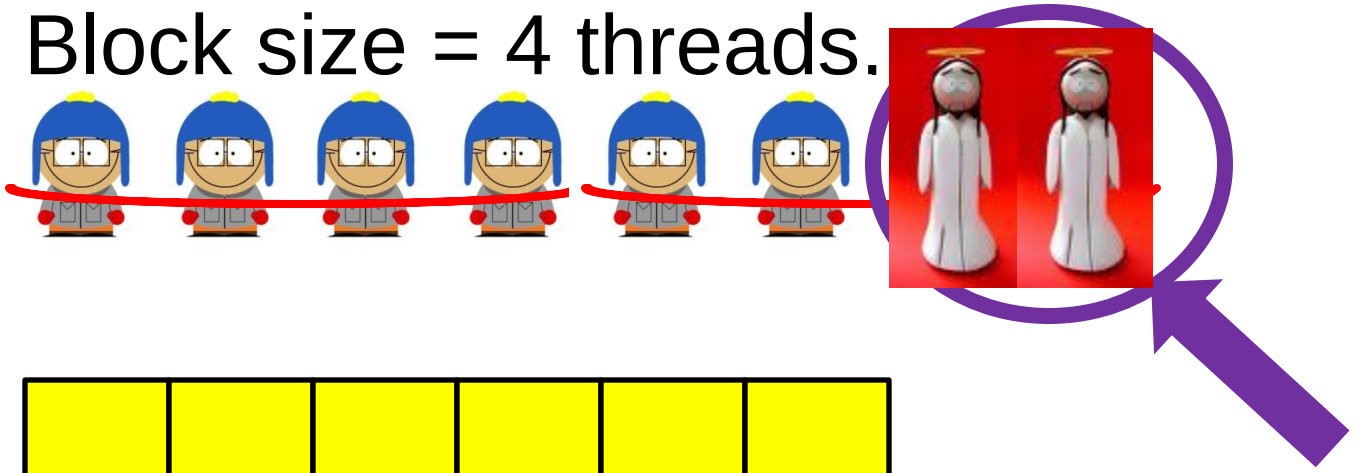
- Block size = 4 threads.

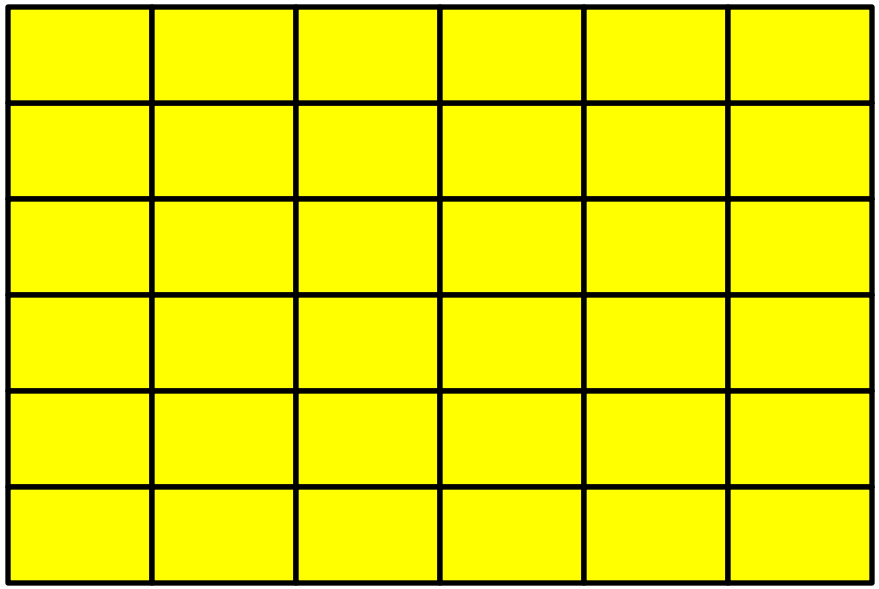




Matrix size != Multiple of block sizes

- Block size = 4 threads.

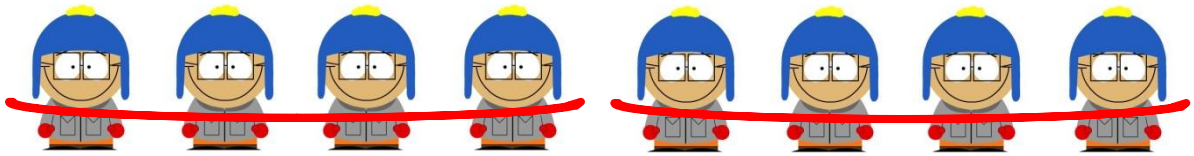




Shouldn't do any work
Why? It might bring a wrong data
And it might add the results into a wrong place
Segmentation faults!

How? Use branch

- Use their names



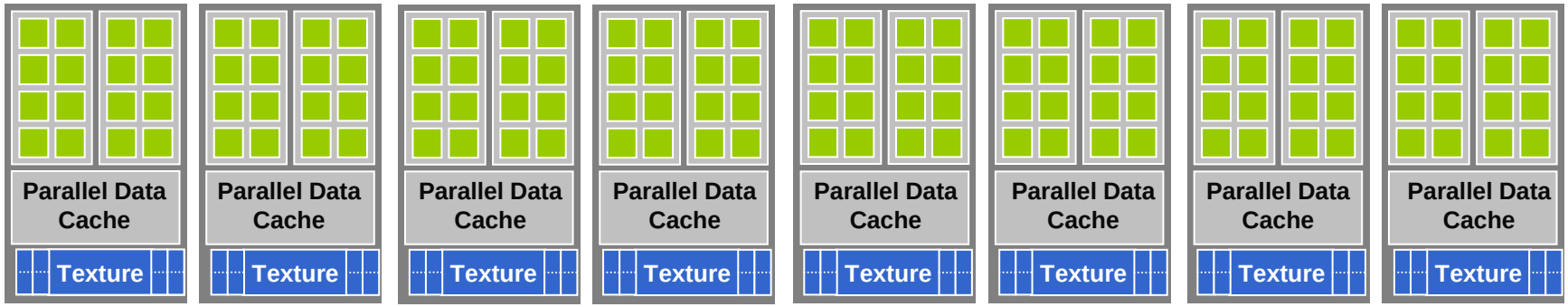
blockIdx.x	0	0	0	0	1	1	1	1
threadIdx.x	0	1	2	3	0	1	2	3

```
Indices  = BLOCK_SIZE & blockidx.x  + threadIdx.x
If(indices < M.width)  {
        // do work
}
```

Then why don't use BLOCK_SIZE=1?

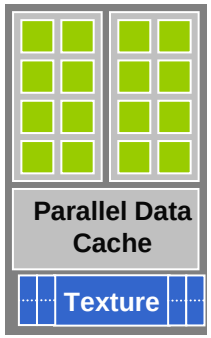


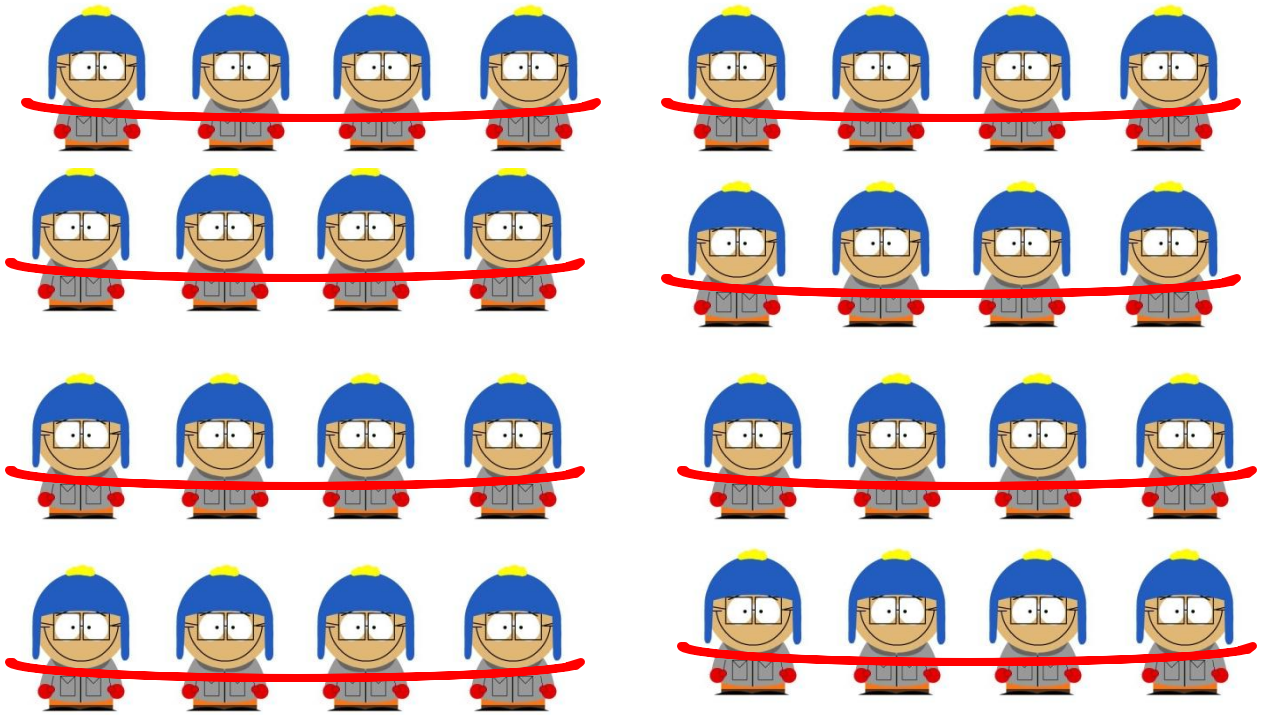
- Resource allocation
- SIMD computation efficiency



Global Memory

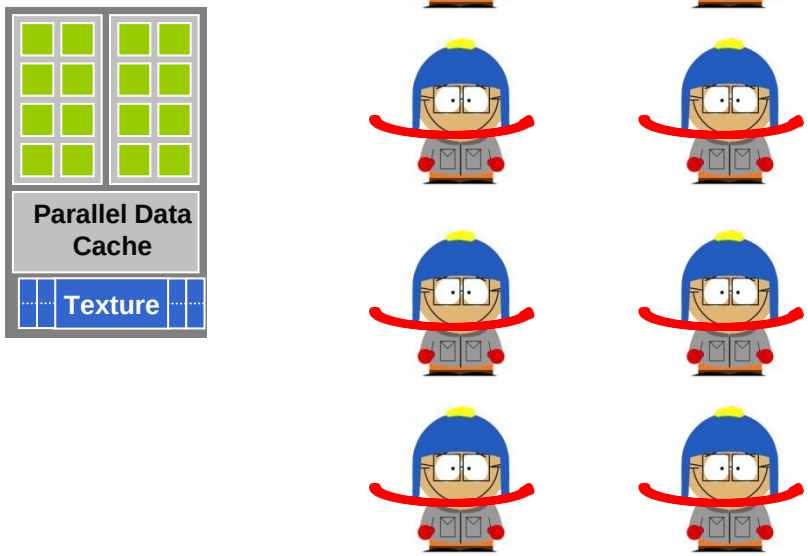
Block size = 4





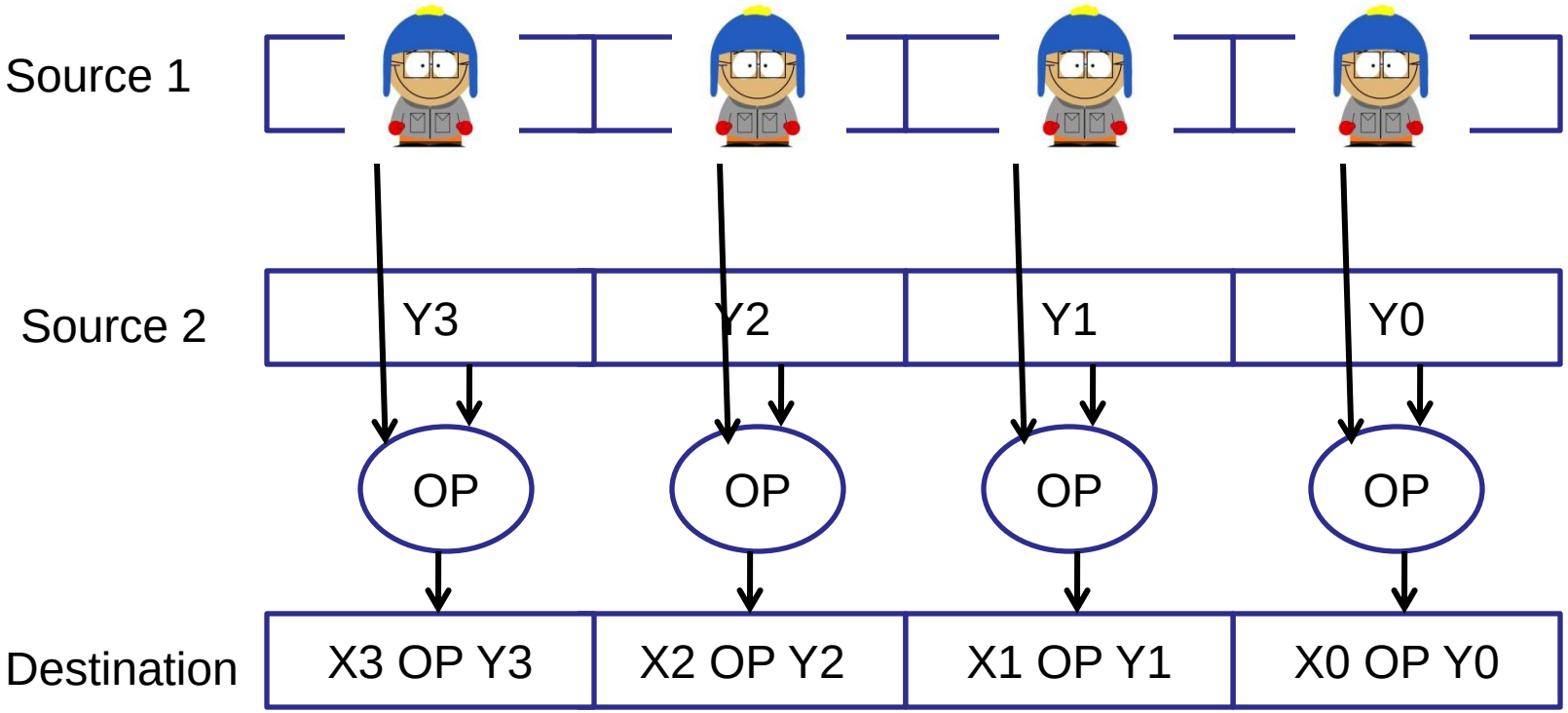
Total thread < 512 or total block < 8

Block size = 1



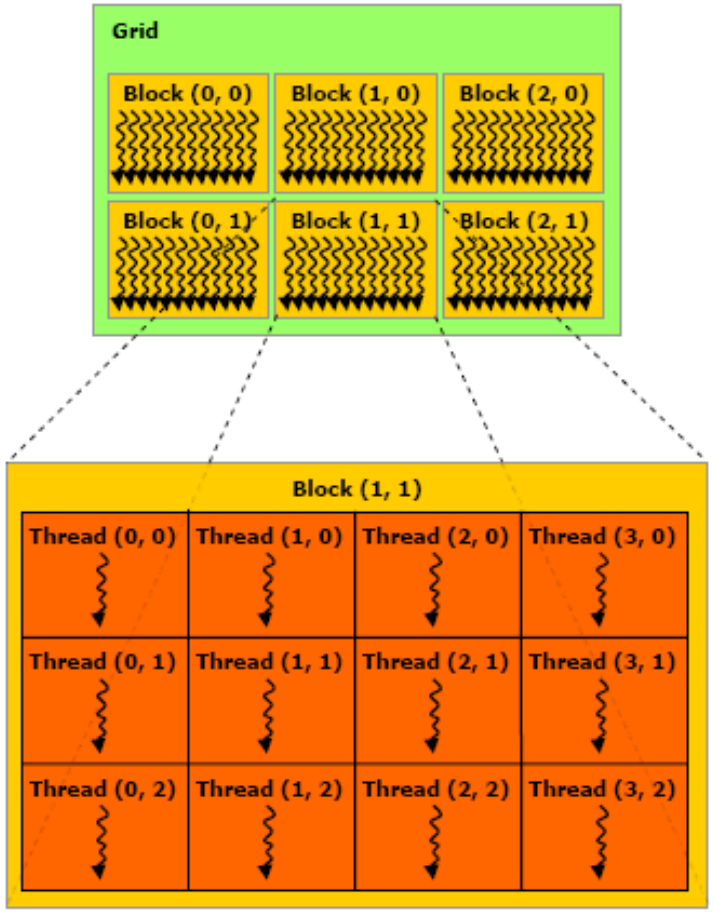
Total thread < 512 or total block < 8

SIMD Execution Model



G80 architecture, SIMID unit size is 8, but 32 threads are handled together
So 32 threads is the best efficient block size

Communication between blocks



- Shared memory cannot be accessible by other blocks

Convolution

- Used by many applications for engineering and mathematics.
- Blur filters or edge detection.

Original Image

Blur convolution filter applied to the source image



Math

- Mathematically, a convolution measures the amount of overlap between two functions.

$$r(i) = (s * k)(i) = \int s(i-n)k(n)dn$$

- Discrete terms

$$r(i) = (s * k)(i) = \sum_n s(i-n)k(n)\,.$$

- Separable convolution (CUDA SDK)

$$r(i) = (s * k)(i,j) = \sum_n \sum_m s(i-n, j-m)k(n,m)$$

Convolution?

Input

23	12	25	36	10
73	26	99	56	2
65	11	5	26	76
83	67	52	32	17
34	84	46	99	32

Kernel

1	0	1
0	1	0
1	0	1

Convolution

23	12	25	36	10
73	26	99	56	2
65	11	5	26	76
83	67	52	32	17
34	84	46	99	32

26	99	56
11	5	26
67	52	32

*

1	0	1
0	1	0
1	0	1

(26 * 1) +
(99 * 0) +
(56 * 1) +
(11 * 0) +
(5 * 1) +
(26 * 0)+
(67 * 1) +
(52 * 0) +
(32 * 1)

23	12	25	36	10
73	26	99	56	2
65	11	18	26	76
83	67	52	32	17
34	84	46	99	32

Boundary

23	12	25	36	10
73	26	99	56	2
65	11	5	26	76
83	67	52	32	17
34	84	46	99	32

0	0	0
0	23	12
0	73	26

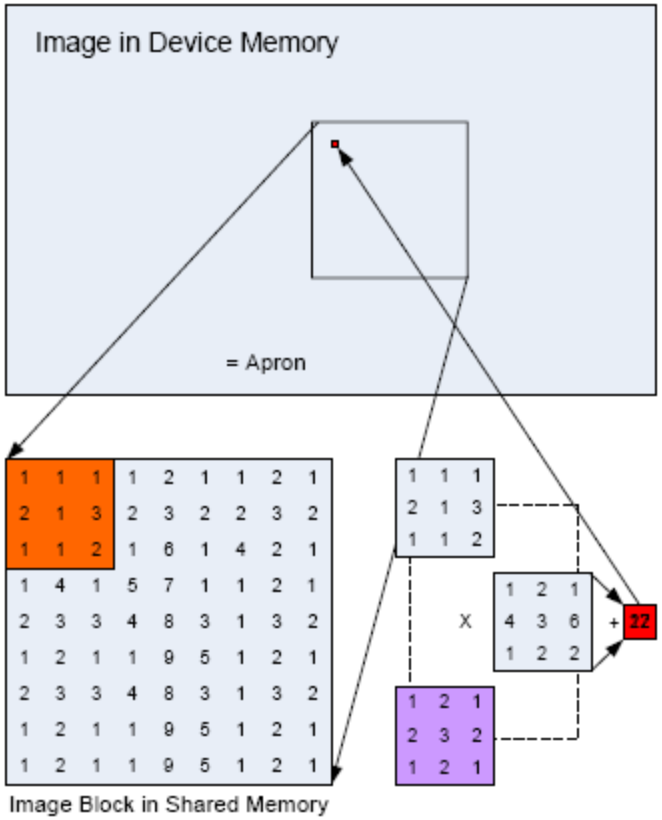
*

1	0	1
0	1	0
1	0	1

(0 * 1) +
(0 * 0) +
(0 * 1) +
(0 * 0) +
(23 * 1) +
(12 * 0)+
(0 * 1) +
(73 * 0) +
(26 * 1)

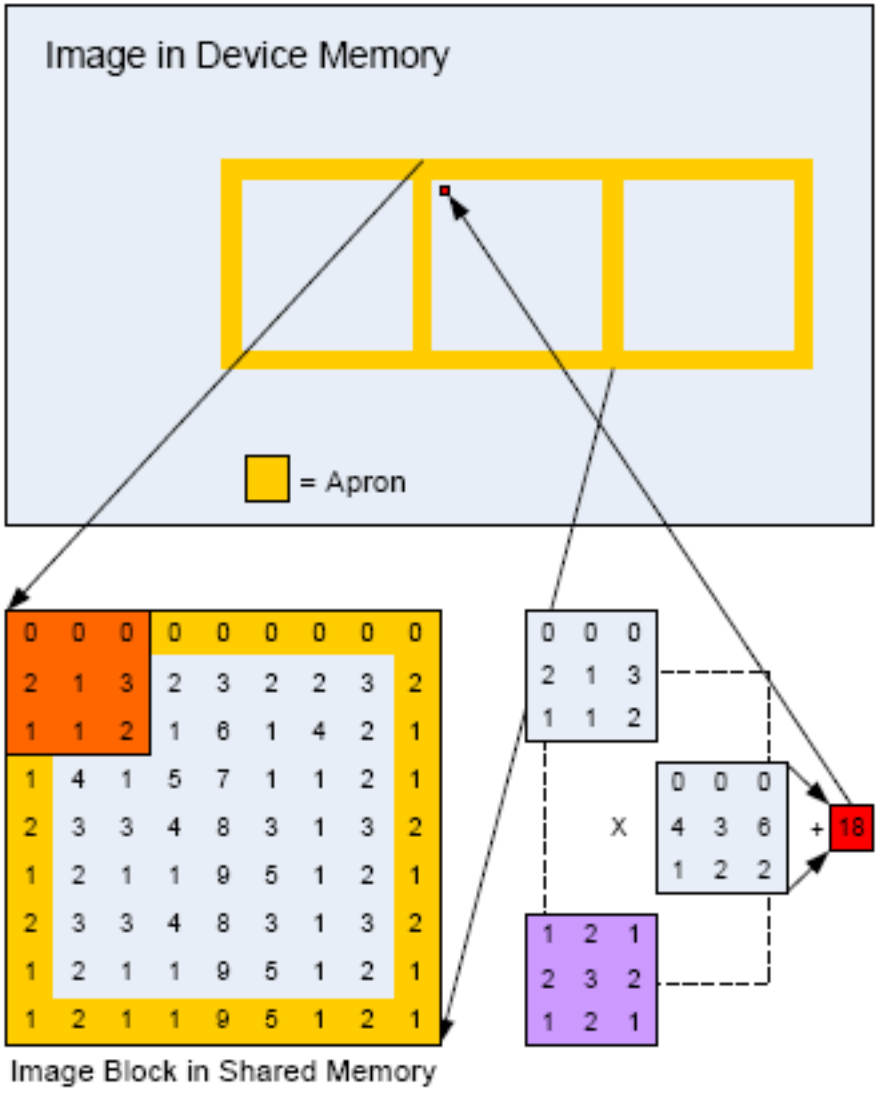
23	12	25	36	10
73	26	99	56	2
65	11	49	26	76
83	67	52	32	17
34	84	46	99	32

A Naïve Implementation



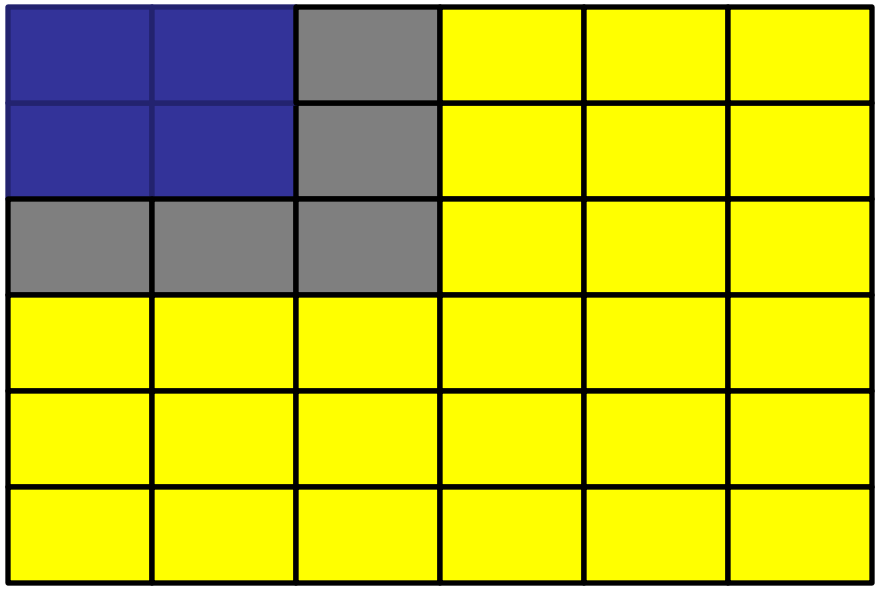
A naïve convolution algorithm. A block of pixels from the image is loaded into an array in shared memory. To process and compute an output pixel (red), a region of the input image (orange) is multiplied element-wise with the filter kernel (purple) and then the results are summed. The resulting output pixel is then written back into the image.

Naïve Implementation: Shared Memory and the Apron

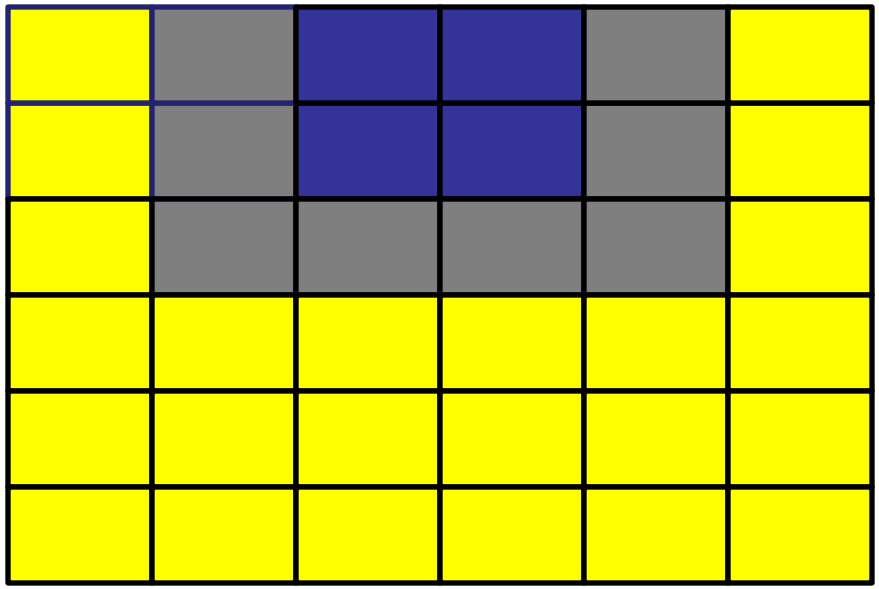


Each thread block must load into shared memory the pixels to be filtered and the apron pixels.

Apron



Apron



Apron