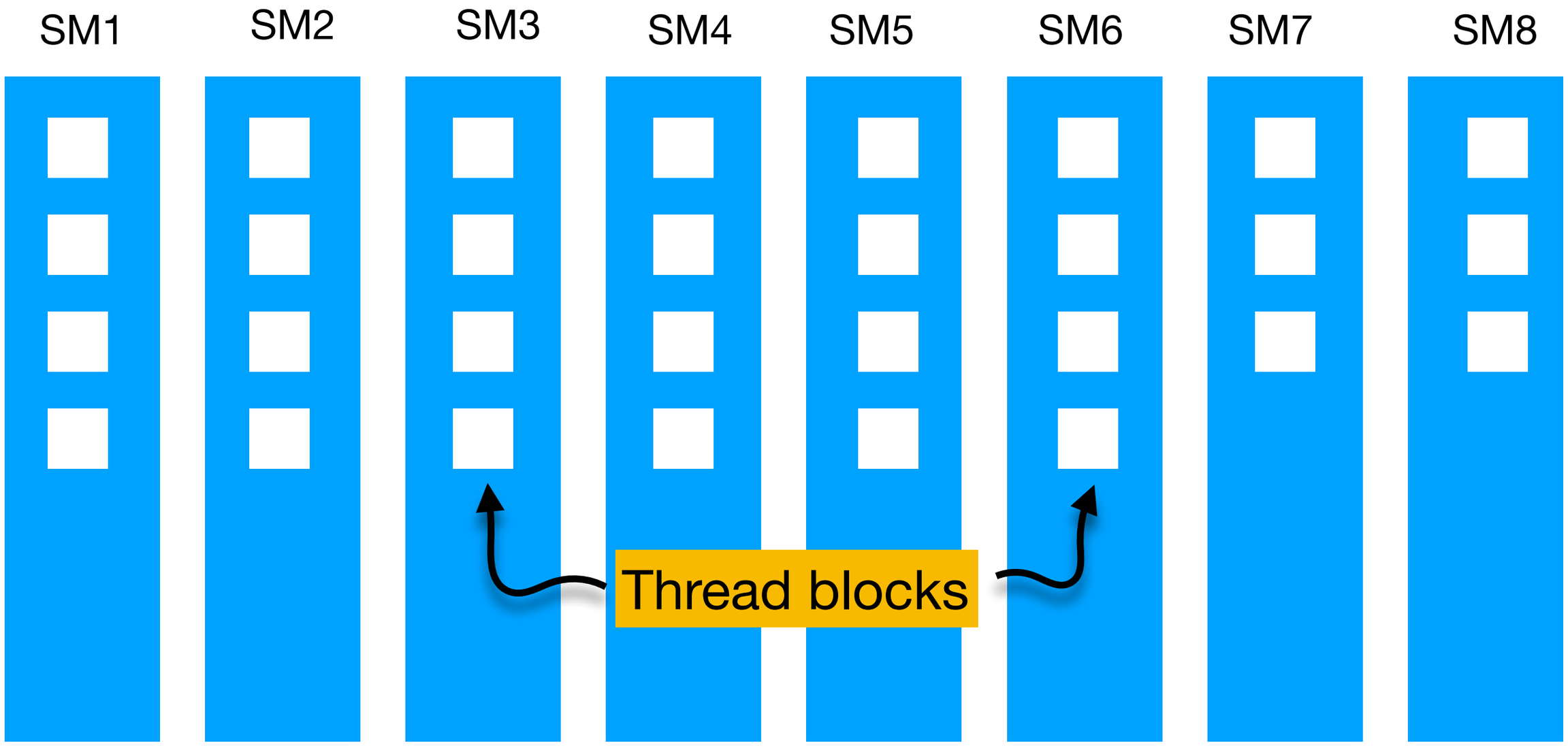
Streaming Multiprocessor

The heart of the GPU is the “Streaming Multiprocessor” or SM.



GPU capabilities

GPUs	Compute Capability	Number of SMs	Blocks per SM	Warps per SM	Threads per SM
GeForce TITAN	3.5 (Kepler)	14	16	64	2048
Tesla K20c	3.5 (Kepler)	13	16	64	2048
Tesla K40c	3.5 (Kepler)	15	16	64	2048
GeForce GTX Titan X	5.2 (Maxwell)	24	32	64	2048

- Blocks are distributed to SMs “round-robin” style.
- The SM is responsible for partitioning registers and shared memory among the thread blocks
- The “warp scheduler” (2-4 per SM) schedules *warps (= 32 threads in a thread block)* on each SM.

Fundamental idea : Warp

- The basic unit of execution on the GPU is a *warp.* A warp consists of 32 threads from a single block and represents the granularity of work processed simultaneously in a “single-instruction-multiple-data” (SIMD) fashion by an SM.
- The SM partitions shared memory among blocks resident in an SM and partitions registers among threads.
- The number of warps that can be active at any one time depend on how resource-demanding each block and thread is.
- A key to optimizing GPU codes is understanding how to configure the kernel.

Configuring Kernels

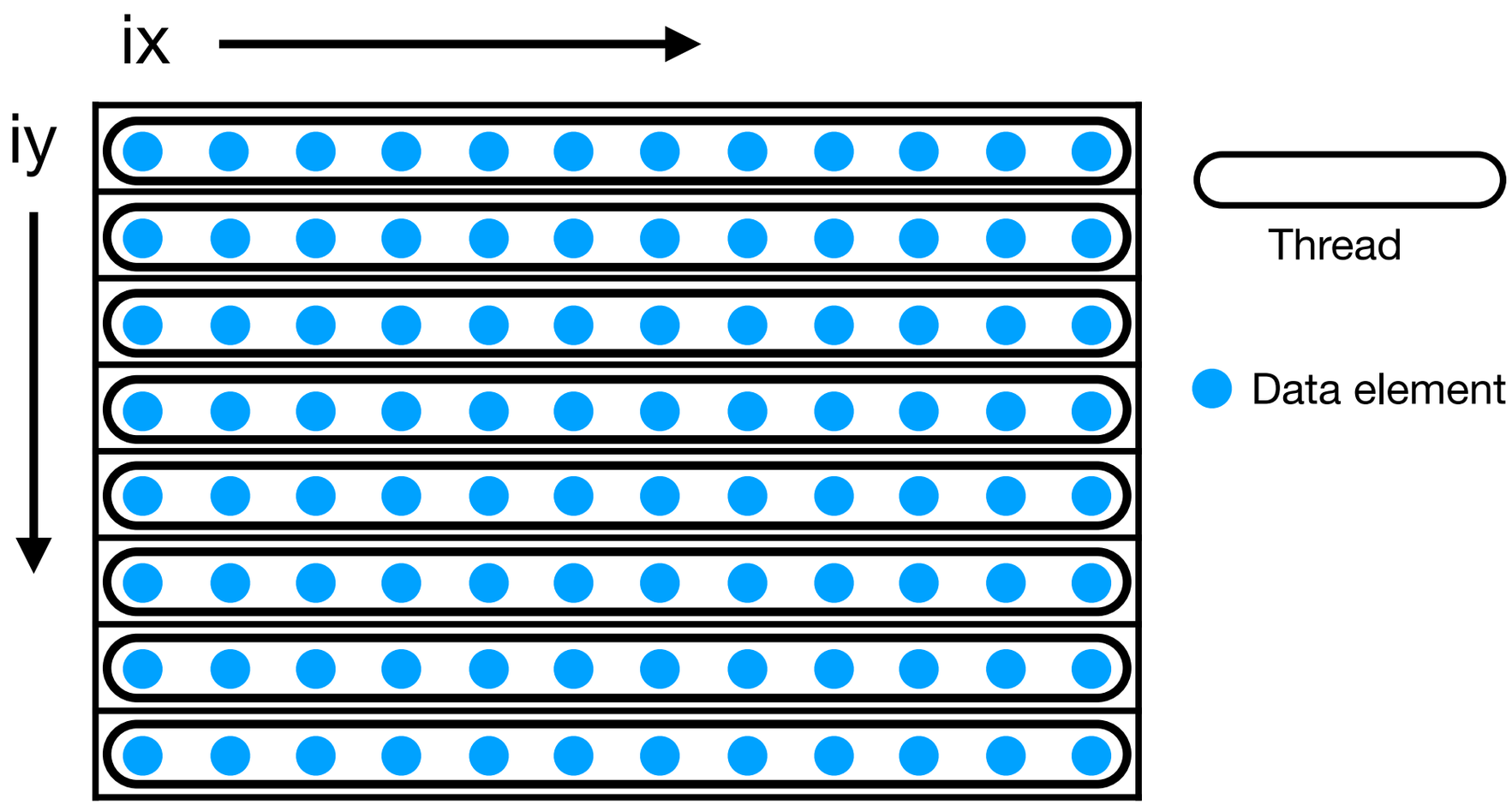
Configuring a kernel : Choose number of threads per block, grid dimensions, and shared memory.

How do we configure a kernel for optimal performance? The number of registers, amount of local and shared memory is limited on each SM. So ...

Kernel configuration will depend on :

- How much work you want each thread to do (e.g. number of floating point operations)
- How much memory each thread requires (e.g. local memory, constant memory, and shared memory)
- How the threads will traverse the memory
-

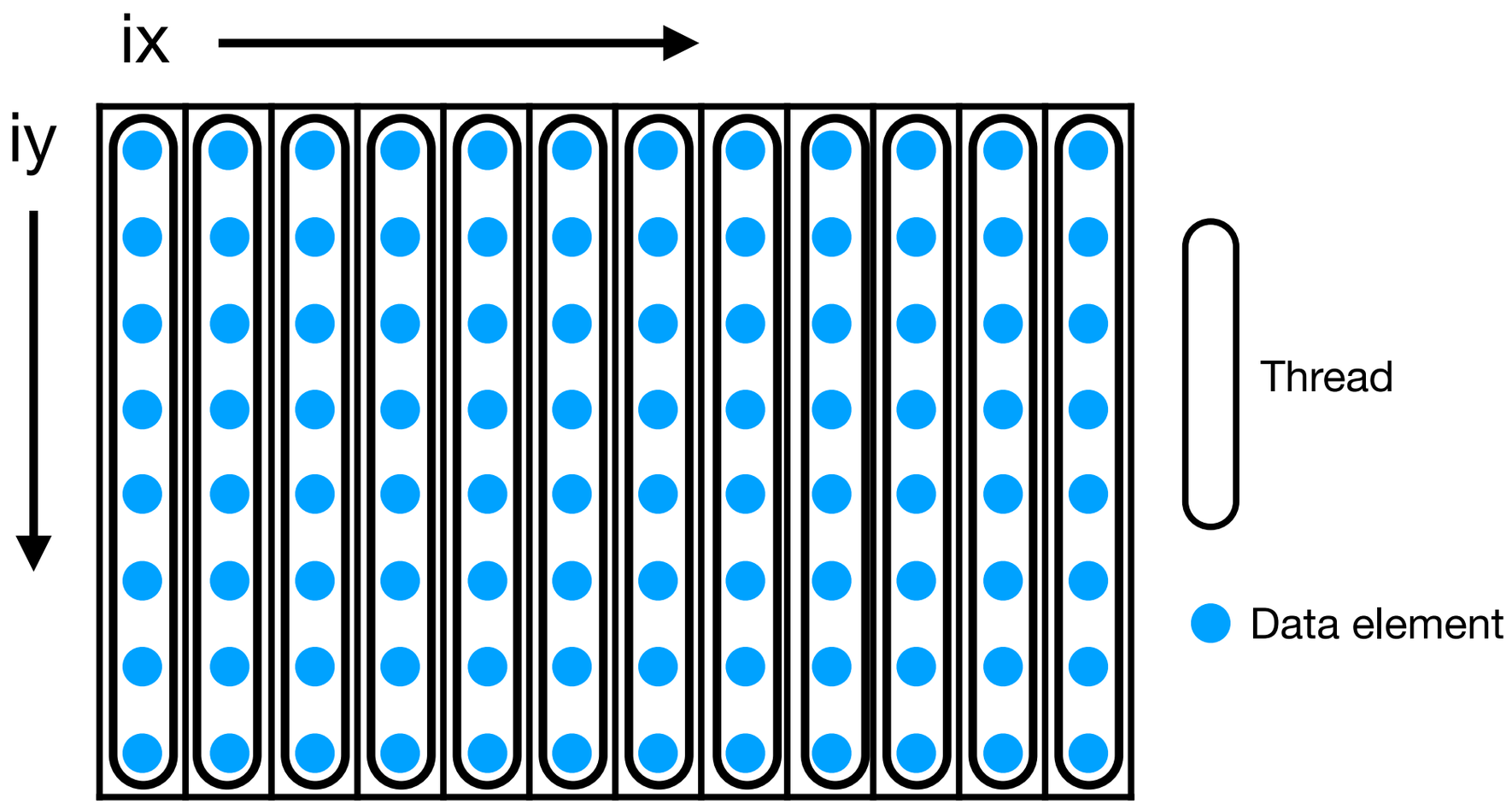
GPU Execution Model



What are the grid dimensions? 1 x 8

What are the block dimensions? 1 x 1

GPU Execution Model



What are the grid dimensions? 12 x 1

What are the block dimensions? 1 x 1

CPU Memory Access

Version 1

Time : **2.22 (s)**

Access entries in the order they appear in memory

```
void copymat_host_x(int m, int n, int* A, int *B)
{
    int ix,iy,idx;
    for(iy = 0; iy < n; iy++)
       for(ix = 0; ix < m; ix++)
        {
            idx = iy*m + ix;
            B[idx] = A[idx];
        }
}
```

Version 2

Time : **9.67 (s)**

Entries separated by a stride of m.

```
void copymat_host_y(int m, int n, int* A, int *B)
{
    int ix,iy,idx;
    for(ix = 0; ix < m; ix++)
        for(iy = 0; iy < n; iy++)
        {
            idx = iy*m + ix;
            B[idx] = A[idx];
        }
}
```

see Week_12/Wednesday/host.cu

GPU Memory access

Version 1

Time : **0.264 (s)**

Access entries in the order they appear in memory

```
__global__ void copymat_x(int m, int n, int* A, int *B)
{
    int idx, ix;
    int iy = threadIdx.y + blockIdx.y*blockDim.y;
    if (iy < n)
        for(ix = 0; ix < P; ix++) {
            idx  = iy*m + ix;
            B[idx]   = A[idx];
        }
}
```

Version 2

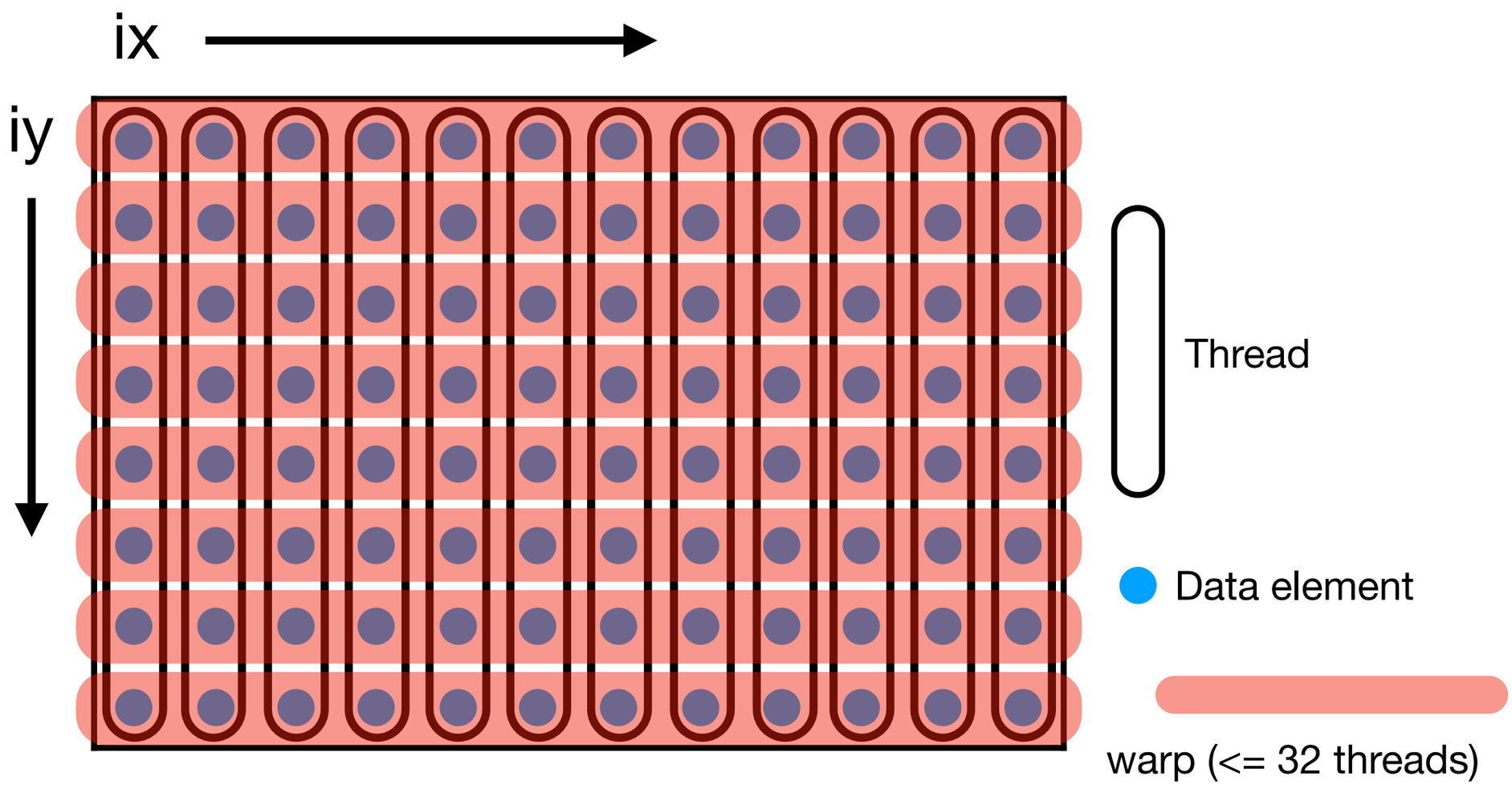
Time : **0.0115 (s)**

Warp is able to access entries in contiguous memory

```
__global__ void copymat_y(int m, int n, int* A, int *B)
{
    int ix = threadIdx.x + blockIdx.x*blockDim.x;
    int idx, iy;
    if (ix < m)
        for(iy = 0; iy < P; iy++) {
            idx  = iy*m + ix;
            B[idx]   = A[idx];
        }
}
```

see Week_12/Wednesday/blockdim.cu

Warp memory access



What are the grid dimensions? 1 x 1

What are the block dimensions? 12 x 1

Profiling with nvprof

```
$ nvprof --metrics achieved_occupancy,sm_efficiency sm
```

Metrics (%)	Description
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor

```
$ nvprof --events warps_launched,sm_cta_launched sm
```

Events (counter)	Description
warps_launched	Number of warps launched.
sm_cta_launched	Number of blocks launched

Profiling : Results (Titan)

GPU	Compute Capability	Number of SMs	Blocks per SM	Warps per SM	Threads per SM
GeForce GTX Titan X	5.2 (Maxwell)	24	32	64	2048

Block	Grid	Blocks/SM	Warps/ block	Warps/SM	Threads/SM	Total Threads	Time (s)
1	1	1	1	1	1	1	0.959
1	12	1	1	1	1	12	1.003
1	24	1	1	1	1	24	1.003
8	24	1	1	1	8	192	1.005
16	24	1	1	1	16	384	1.046
32	24	1	1	1	32	768	1.001
256	24	1	8	8	256	6144	1.022
512	24	1	16	16	512	12288	1.014
1024	24	1	32	32	1024	24576	1.018
256	192	8	8	**64**	**2048**	49152	1.046
512	96	4	16	**64**	**2048**	49152	1.006
1024	48	2	32	**64**	**2048**	49152	1.011

Profiling : Results (Titan)

GPU	Compute Capability	Number of SMs	Blocks per SM	Warps per SM	Threads per SM
GeForce GTX Titan X	5.2 (Maxwell)	24	32	64	2048

Block	Grid	sm_efficiency	warp_execution_efficiency	achieved_occupancy	warps_launched	Time (s)
1	1	**4.17%**	3.12%	0.015625	1	0.959
1	12	**49.63%**	3.12%	0.015625	12	1.003
1	24	**99.24%**	3.12%	0.015625	24	1.003
8	24	99.25%	**25%**	0.015625	24	1.005
16	24	99.28%	**50%**	0.015625	24	1.046
32	24	99.35%	**100%**	0.015625	24	1.001
256	24	98.74%	100%	**0.125000**	192	1.022
512	24	98.87%	100%	**0.250000**	768	1.014
1024	24	99.17%	100%	**0.500000**	1536	1.018
256	192	98.71%	100%	1.000000	1536	1.046
512	96	99.09%	100%	1.000000	1536	1.006
1024	48	99.06%	100%	1.000000	1536	1.011

Exposing parallelism

GPU	Compute Capability	Number of SMs	Blocks per SM	Warps per SM	Threads per SM
GeForce GTX Titan X	5.2 (Maxwell)	24	32	64	2048

Block	Grid	Blocks/SM	Warps/ block	Warps/SM	Threads/SM	Total Threads	Time (s)
1024	48	2	32	**64**	**2048**	49152	1.011
512	96	4	16	**64**	**2048**	49152	1.006
256	192	8	8	**64**	**2048**	49152	1.046
128	384	16	4	**64**	**2048**	49152	1.006
64	768	**32**	2	**64**	**2048**	49152	1.033

Find three examples of simulations that take 2 seconds, but launch fewer than 49152 threads.

Exposing parallelism

GPU	Compute Capability	Number of SMs	Blocks per SM	Warps per SM	Threads per SM
GeForce GTX Titan X	5.2 (Maxwell)	24	32	64	2048

Block	Grid	Blocks/SM	Warps/ block	Warps/SM	Threads/SM	Total Threads	Time (s)
1	769	**33**	1	33	33	769	1.963
704	49	3	22	**66**	**2112**	34496	1.949
33	769	**33**	2	**66**	1089	25377	1.950

Block	Grid	sm_efficiency	warp_execution_ efficiency	achieved_ occupancy	warps_ launched	Time (s)
1	769	52.27%	3.12%	0.480624	769	1.963
704	49	52.29%	**100%**	0.673750	1078	1.949
33	769	51.86%	51.60%	0.961032	1538	1.950

More Profiling

```
$ nvprof --query-metrics
```

```
$ nvprof --query-events
```

Lists all events and metrics that can be profiled

```
$ nvprof --metrics all
```

Profiles all metrics and/or events

```
$ nvprof --events all
```

Note : Profiling your code will affect timing results, so it may be a good idea to time your kernels separately. Also, some metrics and events will conflict with each other giving erroneous results. Better to run queries separately.