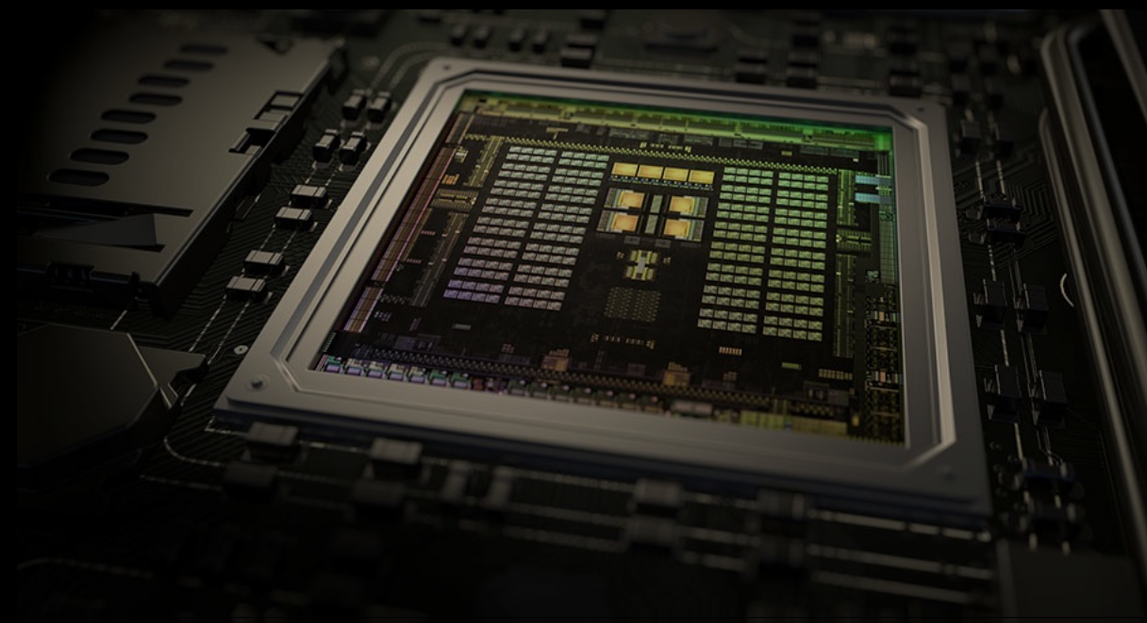


CUDA Programming Model

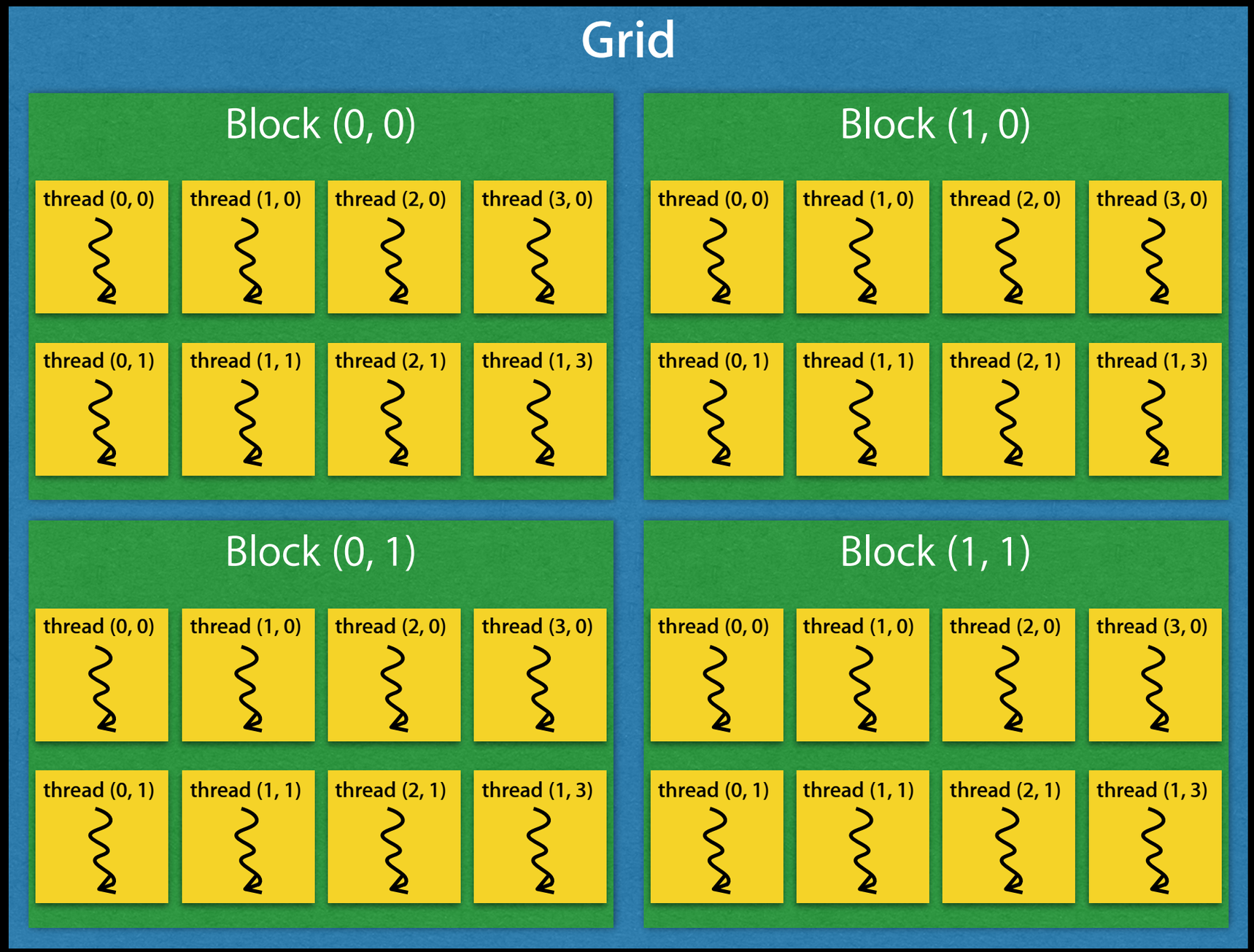
Ming Yang
Apr 5, 2016

Thread
Occupancy
Block
Warp
Local memory
Grid
Shared memory
Kernel
Register
Global memory

???

- What are the scheduling units in Streaming Multiprocessor (SM)??
  - warps. How are they scheduled?
- How is the occupancy computed??
  - anything to do with block/thread/registers/shared memory? Yes! All of them.

Thread hierarchy



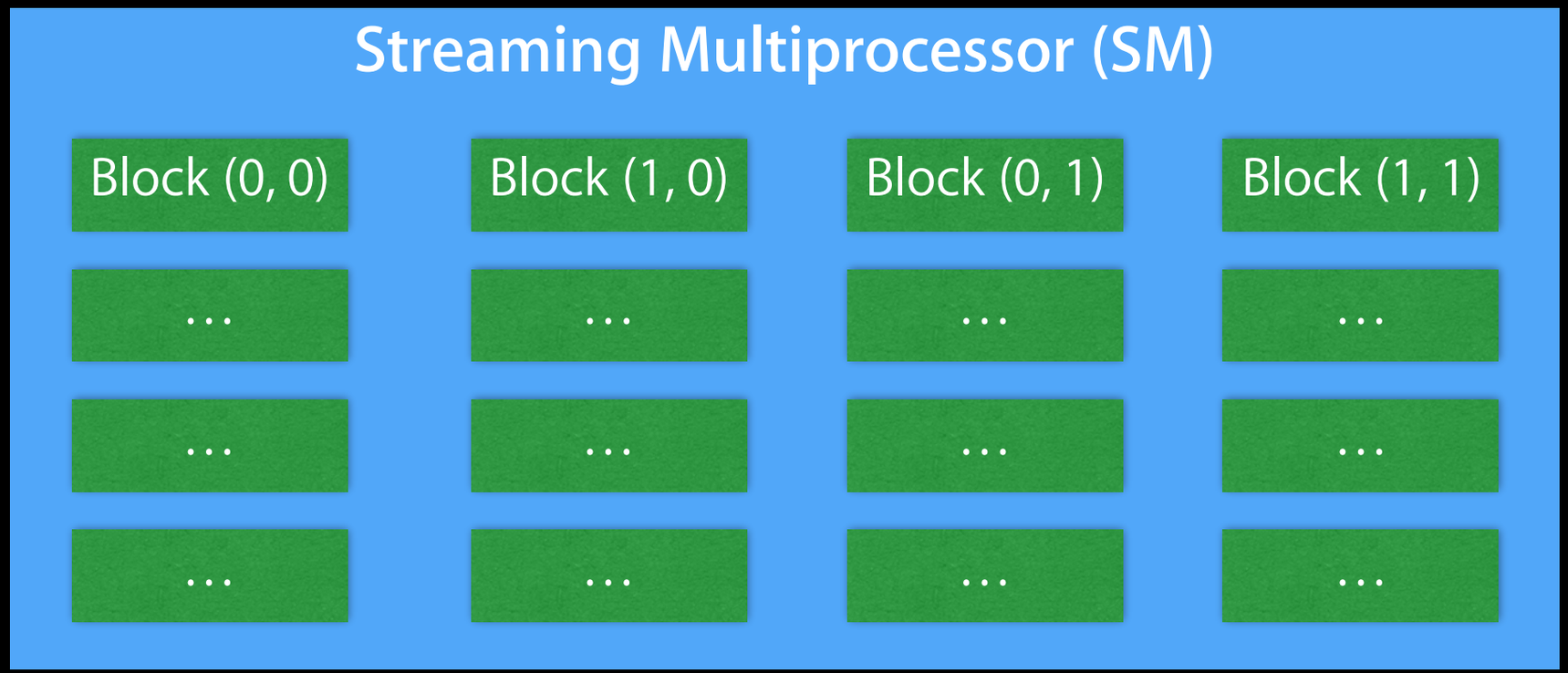
# of threads:

(2*2) * (4*2) = 32

```
…
dim3 dimGrid(2, 2, 1);
dim3 dimBlock(4, 2, 1);
vectorAdd<<<dimGrid, dimBlock>>>(a, b, c);
…
```

How are these threads assigned to the SMs??

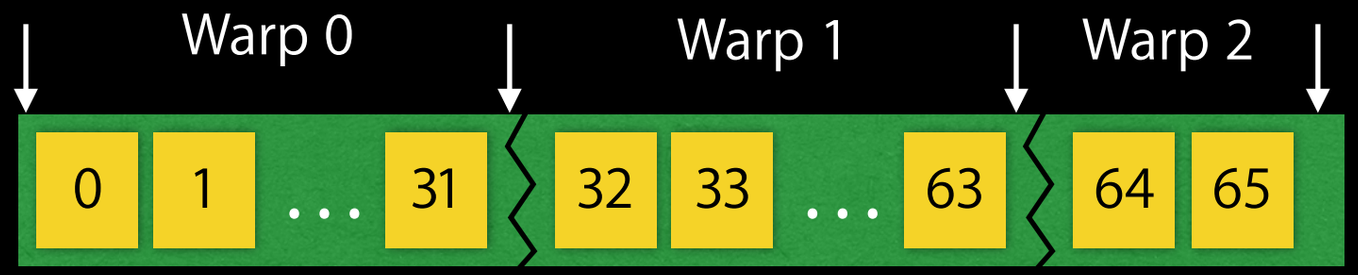
Thread Blocks Assignment



- Threads are assigned to SM in block granularity
- Blocks in one grid can be assigned to different SMs
- SM manages/schedules thread execution.
  - how??

Warps as Scheduling Units

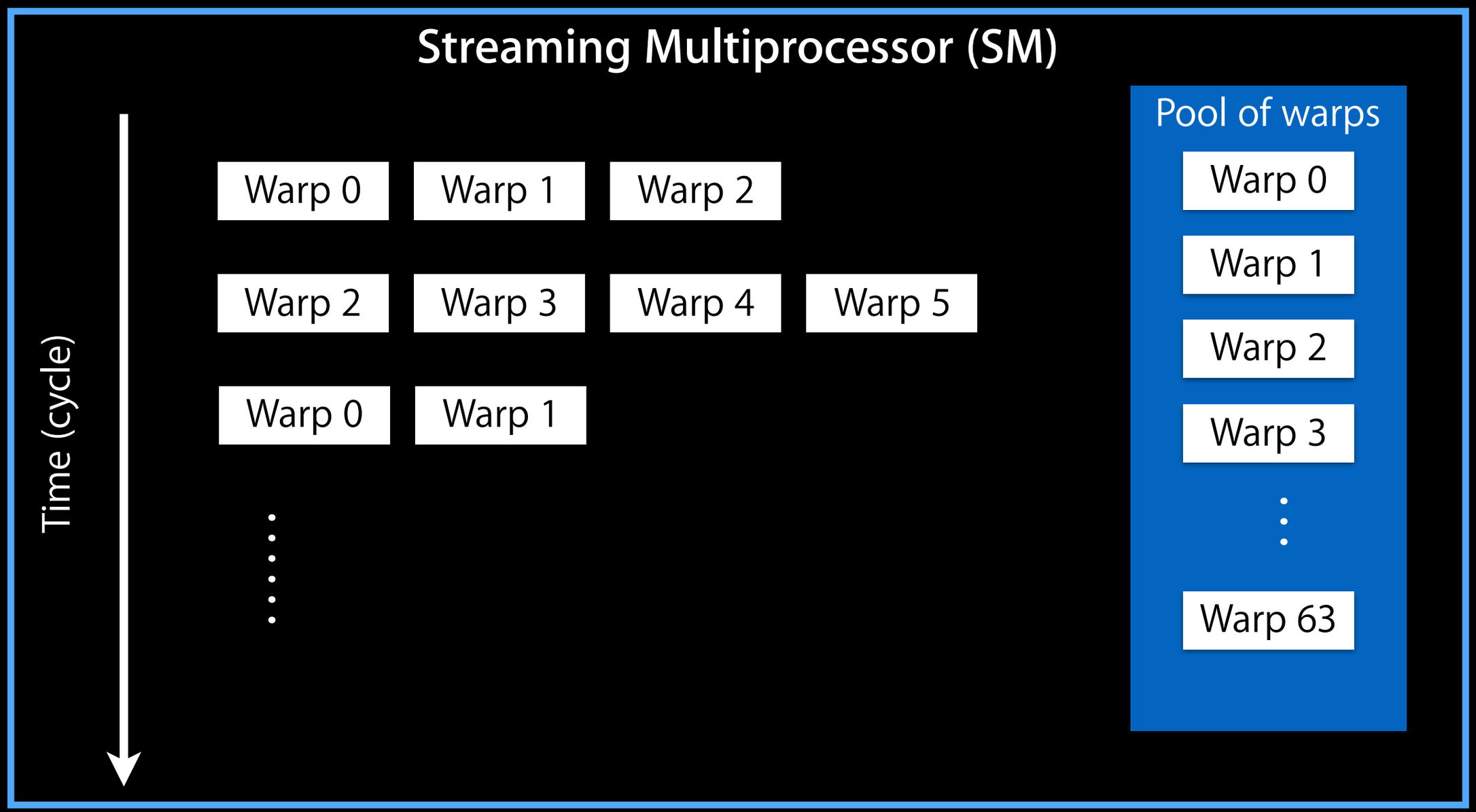
- Each block is executed as 32-thread warps



- Warps are scheduling units in SM
  - how are they scheduled?
- Threads in a warp execute in SIMT
  - what is SIMT (Single Instruction Multiple Thread)?
  - What about control divergence?

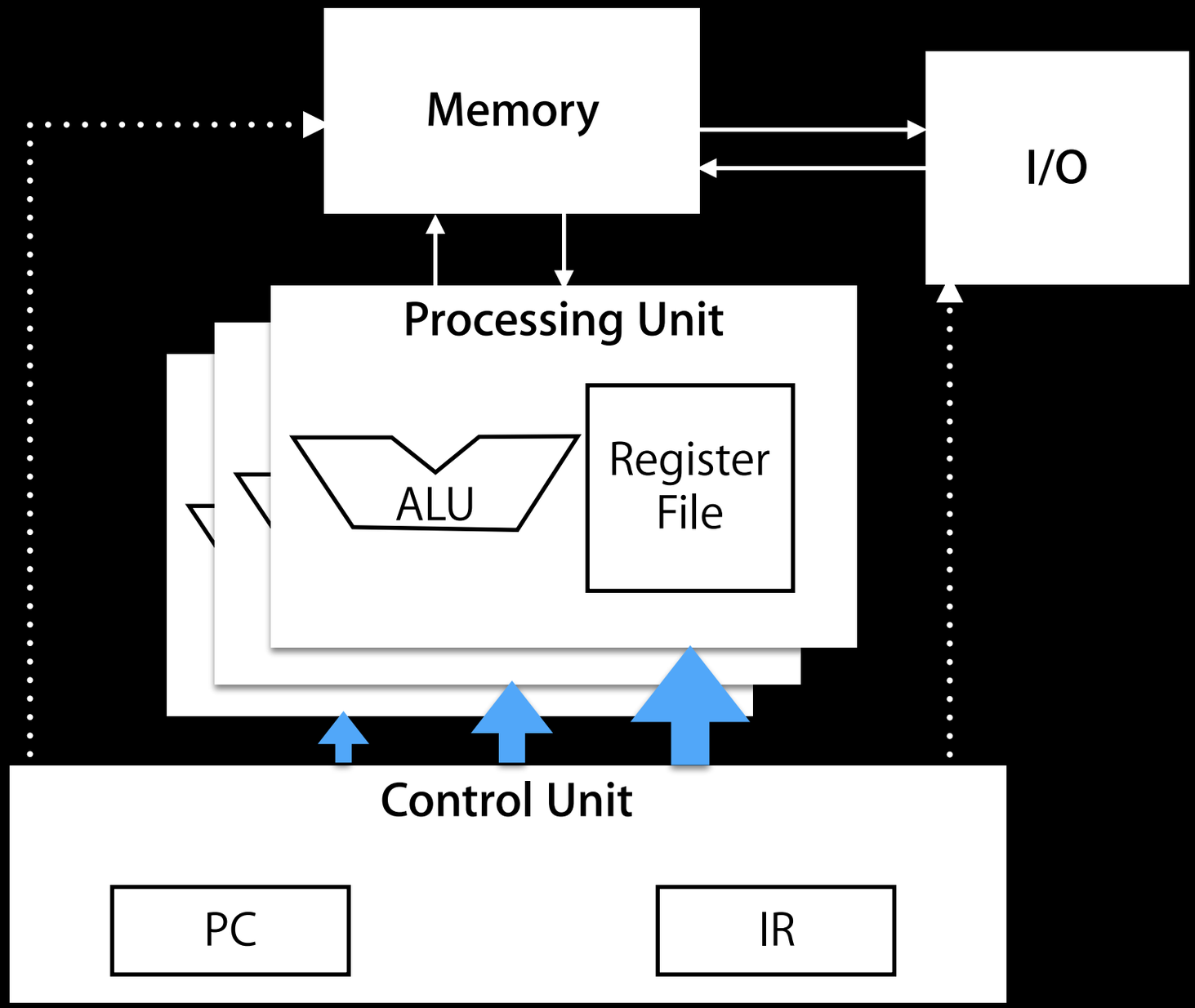
Warps as Scheduling Units (cont.)

- Warps are scheduling units in SM



Warps as Scheduling Units (cont.)

- Threads in a warp execute in SIMT



Review

- Threads are organized by block/grid
- Threads are assigned to SM in block granularity
- Threads are scheduled in the unit of warp, and in the way of SIMD

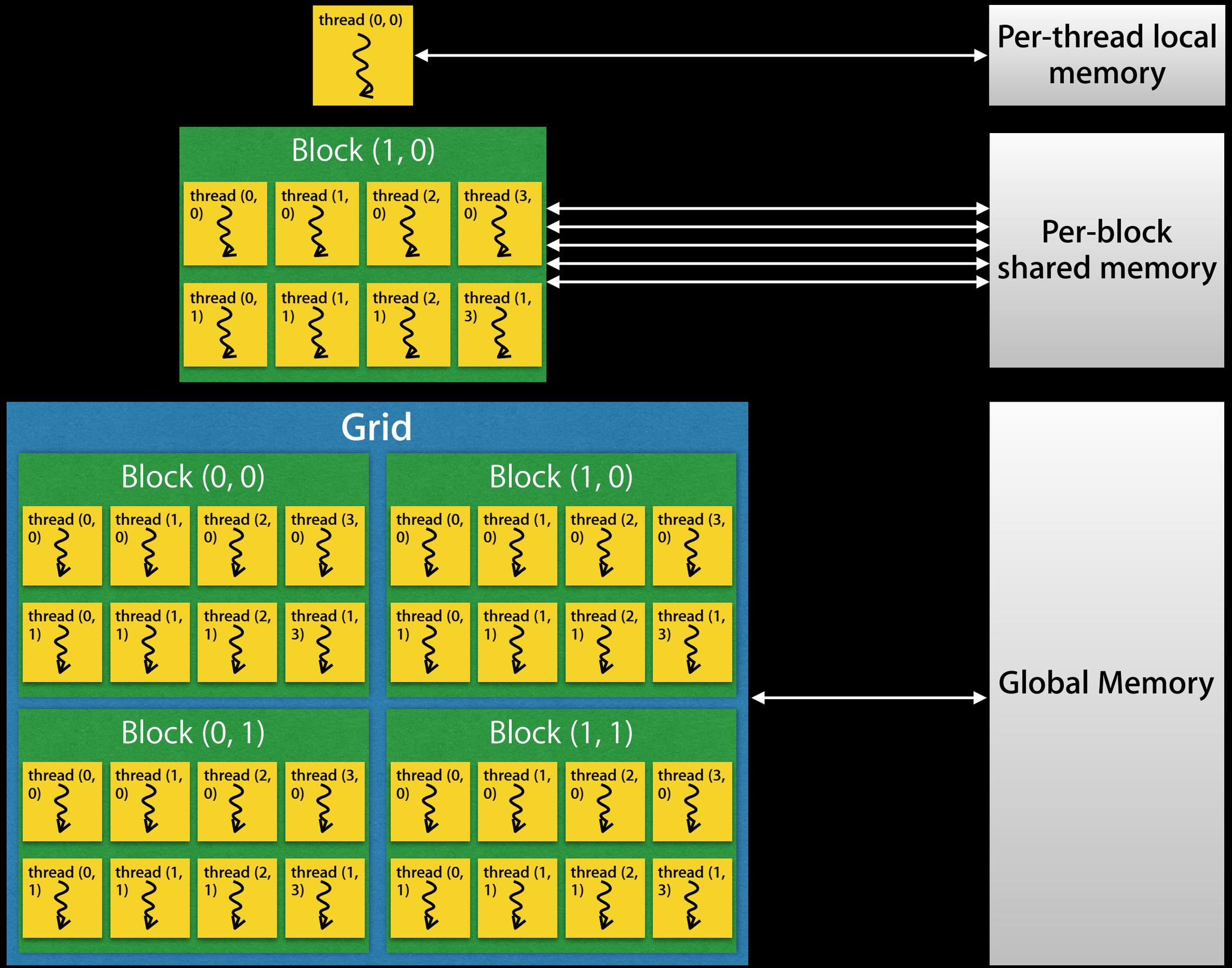
Occupancy

- Occupancy = # of active warps / Maximum number of resident warps per SM

Technical Specifications	Compute Capabilities							
	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3
Maximum number of resident warps per SM	48	64						

- Occupancy limiters:
  - Register usage
  - Shared memory usage
  - Block size

Memory hierarchy



Occupancy limiter: Register usage

Technical Specifications	Compute Capabilities							
	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3
Maximum number of 32-bit registers per thread block	32 K	**64 K**						32 K
Maximum number of resident threads per SM	1536	**2048**						

- Example 1 (capability = 3.0)
  - Kernel uses 21 registers per thread
  - # of active threads = 64K / 21 ≈ 3121
    - > 2048 thus an occupancy of 100%

Occupancy limiter: Register usage (cont.)

Technical Specifications	Compute Capabilities							
	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3
Maximum number of 32-bit registers per thread block	32 K	**64 K**						32 K
Maximum number of resident threads per SM	1536	**2048**						
Maximum number of resident warps per SM	48	**64**						

- Example 2 (capability = 3.0)
  - Kernel uses 64 registers per thread
  - # of Active threads = 64K / 64 = 1024
    - # of warps = 1024 / 32 = 32
    - Occupancy = 32 / 64 = 50%

Occupancy limiter: Shared memory

Technical Specifications	Compute Capabilities							
	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3
Maximum amount of shared memory per SM	**48 KB**				112 KB	64 KB	96 KB	64 KB
Maximum number of resident threads per SM	1536	**2048**						
Maximum number of resident warps per SM	48	**64**						

- Example 1 (capability = 3.0)
  - Kernel uses 16 bytes of shared memory per thread
  - # of Active threads = 48K / 16 = 3072
    - > 2048 thus an occupancy of 100%

Occupancy limiter: Shared memory (cont.)

Technical Specifications	Compute Capabilities							
	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3
Maximum amount of shared memory per SM	48 KB				112 KB	64 KB	96 KB	64 KB
Maximum number of resident threads per SM	1536	2048						
Maximum number of resident warps per SM	48	64						

- Example 2 (capability = 3.0)
  - Kernel uses 32 bytes of shared memory per thread
  - # of Active threads = 48K / 32 = 1536
    - # of warps = 1536 / 32 = 48
    - Occupancy = 48 / 64 = 75%

Occupancy limiter: Block size

Technical Specifications	Compute Capabilities							
	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3
Maximum number of resident blocks per multiprocessor	8	16				32		
Maximum number of resident threads per SM	1536	2048						
Maximum number of resident warps per SM	48	64						

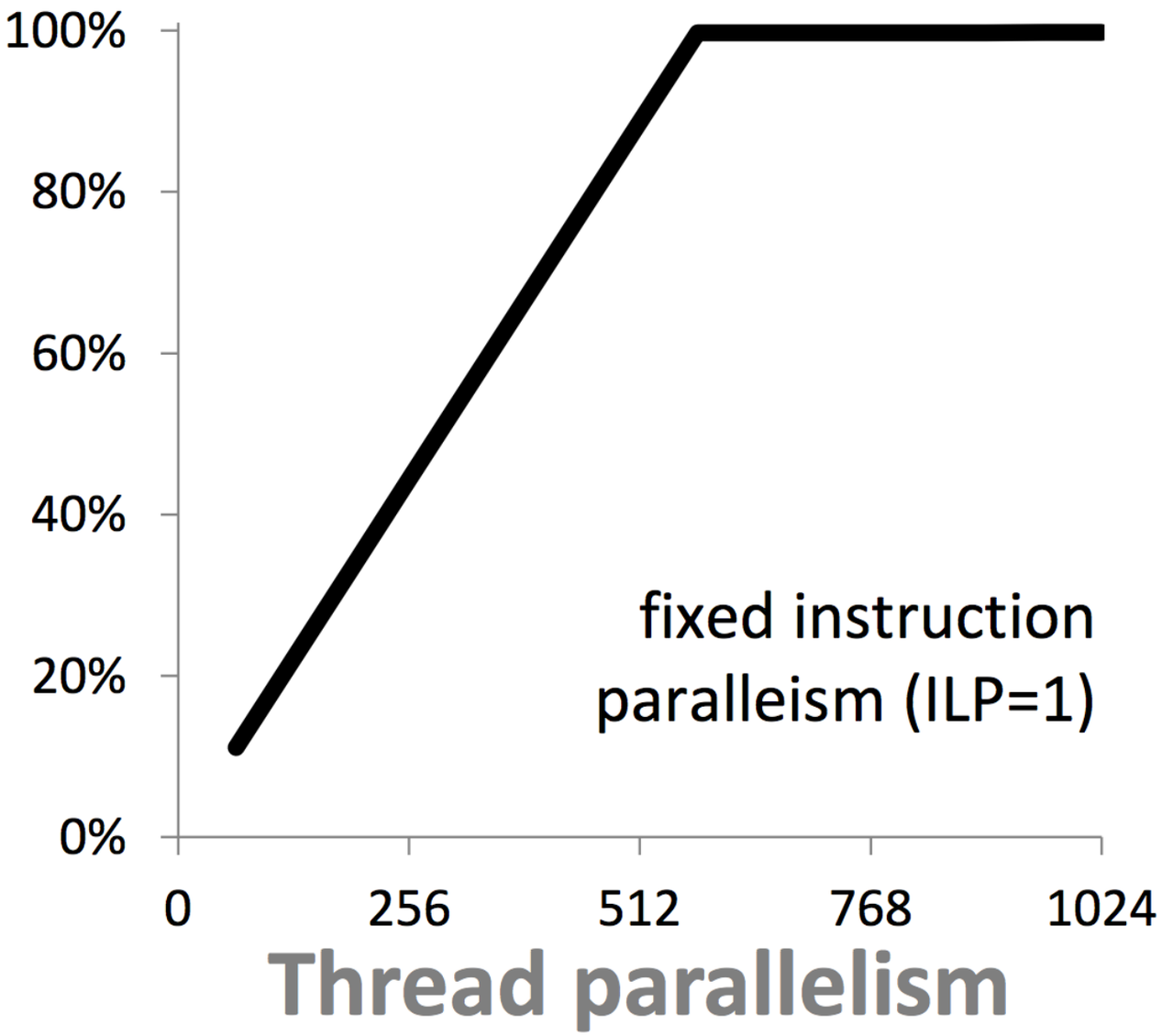
- capability = 3.0

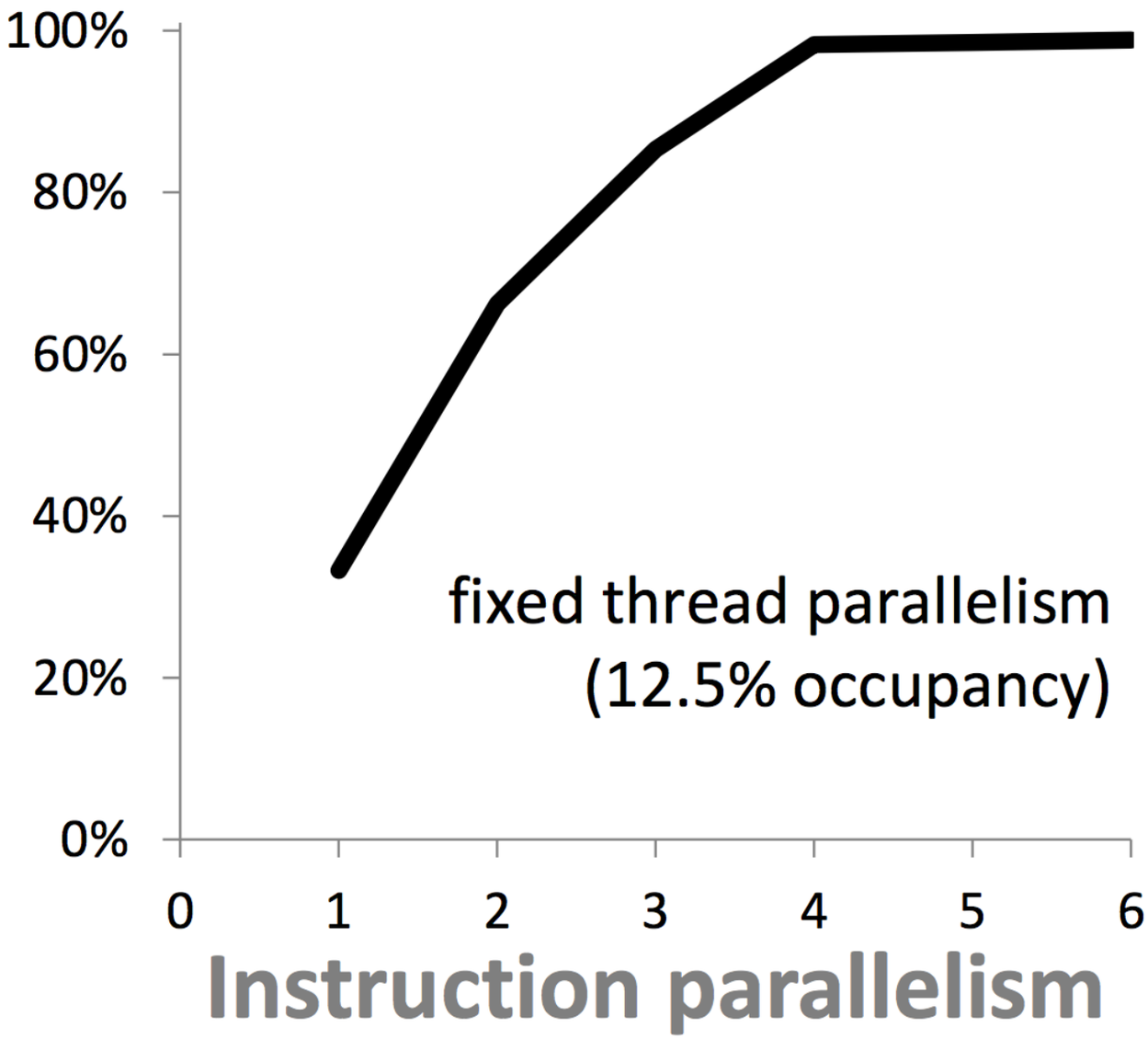
Warp size=32

Block size	Active threads	Active warps	Occupancy
32	32 * 16 = 512	512 / 32 = 16	16 / 64 = 25%
64	1024	32	50%
128	2048	64	100%
192	3072 (2048)	64	100%
256	4096 (2048)	64	100%

Occupancy

- Do we want higher occupancy?
  - Maybe yes. Latency (of memory op. and algorithmic op.) can be hidden with more threads running.
- Is occupancy a metric of performance?
  - No!! It's just one of the contributing factors.





Reference:
http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf
http://on-demand.gputechconf.com/gtc/2010/video/S12238-Better-Performance-at-Lower-Occupancy.mp4

Review

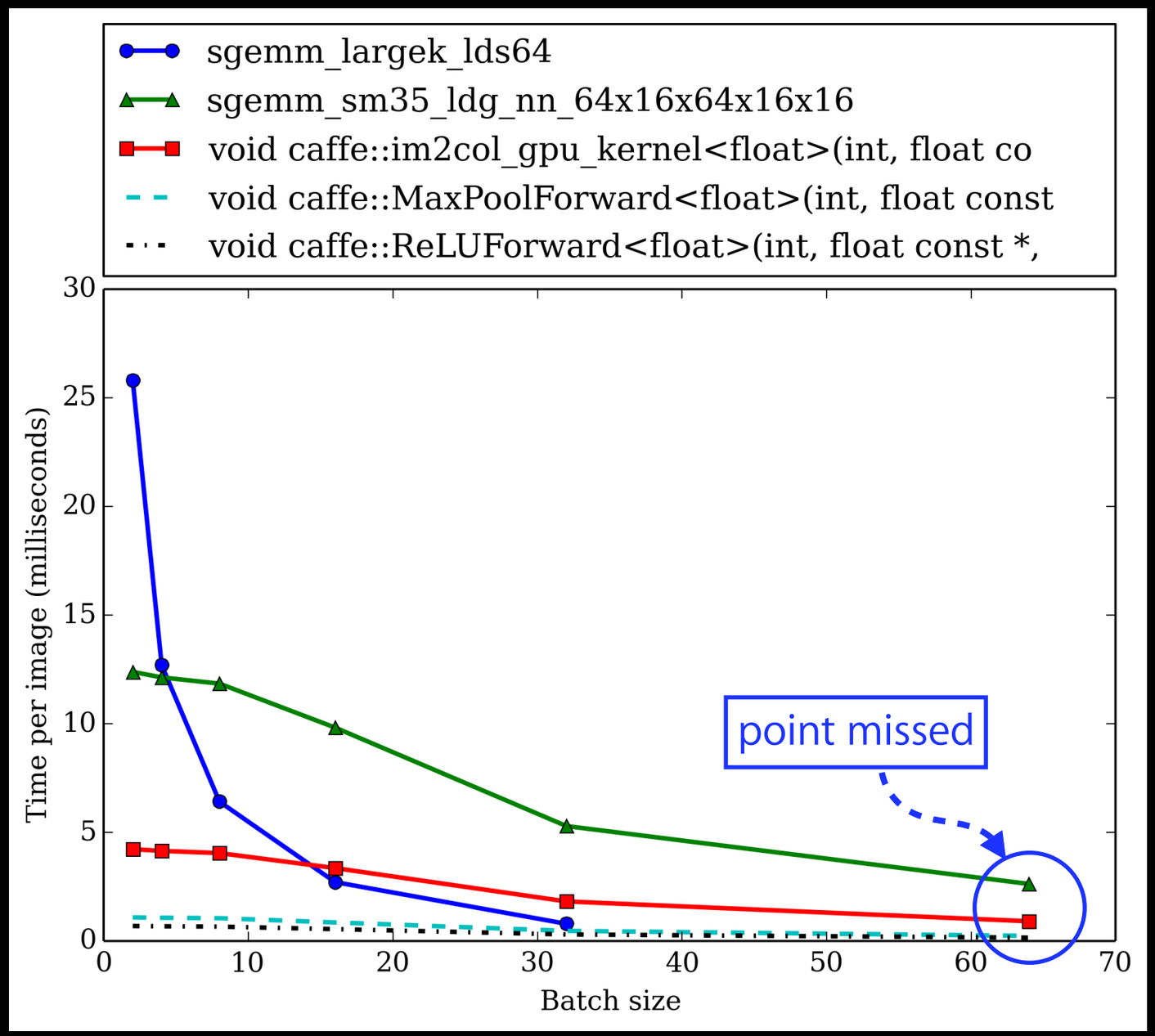
- Calculation formula for occupancy
  - # of active warps / maximum number of warps per SM
- Occupancy limiters:
  - register, shared memory, block size
- Understanding of occupancy
  - occupancy is not equivalent to performance
  - but we still want higher occupancy usually

Case study: cublasSgemm

- Matrix multiplication of single-precision real number
  - SGEMM performs one of the matrix-matrix operations
    - C := alpha*op(A)*op(B) + beta*C
      - alpha: 1.0
      - beta: 0.
    - where op(X) is one of
      - **op(X) = X** or op(X) = X**T (transposed)
        - always this one in our case
- It's used by the fully-connected (fc) layer in Caffe (when batch size is larger than 1)

Reasons of case-studying cublasSgemm

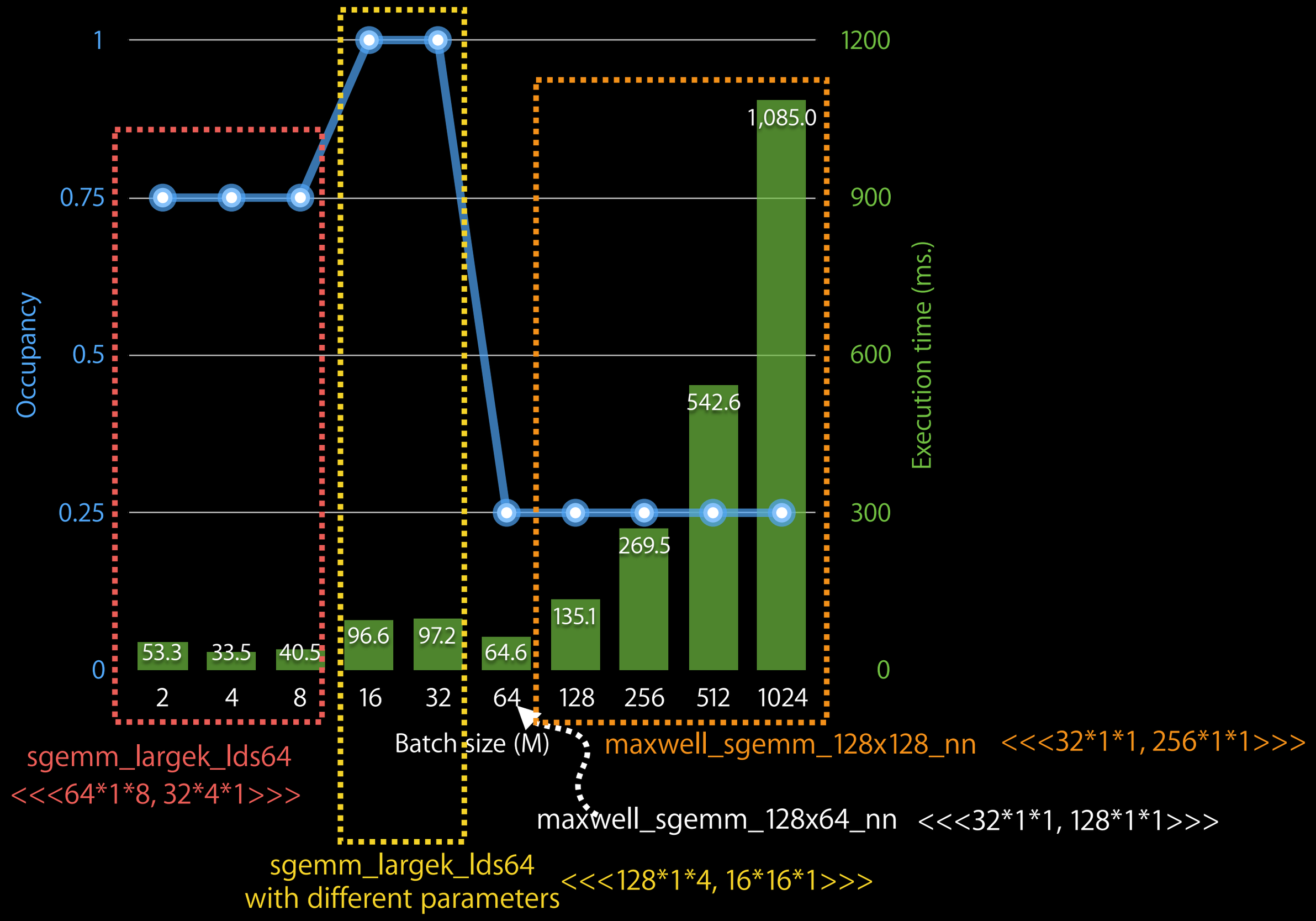
- sgemm_largek_lds64
  - it's the kernel used by cublasSgemm
- it decreases fastest with batch size increasing
- it's the only kernel I observed of which occupancy changes with different batch sizes



Experiment

- Use cublasSgemm:
  - Inputs: Matrix A (M*K), B (K*N)
  - Output: Matrix C (M*N) = A*B
- Variables used here are consistent with the usage in the fully-connected layer in Caffe)
  - M: batch size (2, 4, 8, …, 1024)
  - K: 9216/**4096**/4096
  - N: 4096/**4096**/1000

Results


Occupancy
1
0.75
0.5
0.25
0
Execution time (ms.)
1200
900
600
300
0
53.3
33.5
40.5
96.6
97.2
64.6
135.1
269.5
542.6
1,085.0
2
4
8
16
32
64
128
256
512
1024
Batch size (M)
sgemm_largek_lds64
<<<64*1*8, 32*4*1>>>
sgemm_largek_lds64
with different parameters
<<<128*1*4, 16*16*1>>>
maxwell_sgemm_128x128_nn <<<32*1*1, 256*1*1>>>
maxwell_sgemm_128x64_nn <<<32*1*1, 128*1*1>>>

Summary

- Thread hierarchy
- Streaming multiprocessor scheduling
- Memory hierarchy
- Occupancy
- Case study on `cublasSgemm`

References

- (Coursera class) Heterogeneous Parallel Programming by Wen-mei W. Hwu (https://class.coursera.org/hetero-004)

- http://docs.nvidia.com/cuda/cuda-c-programming-guide/

- http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf

Backup slides (about stream and concurrency) after this

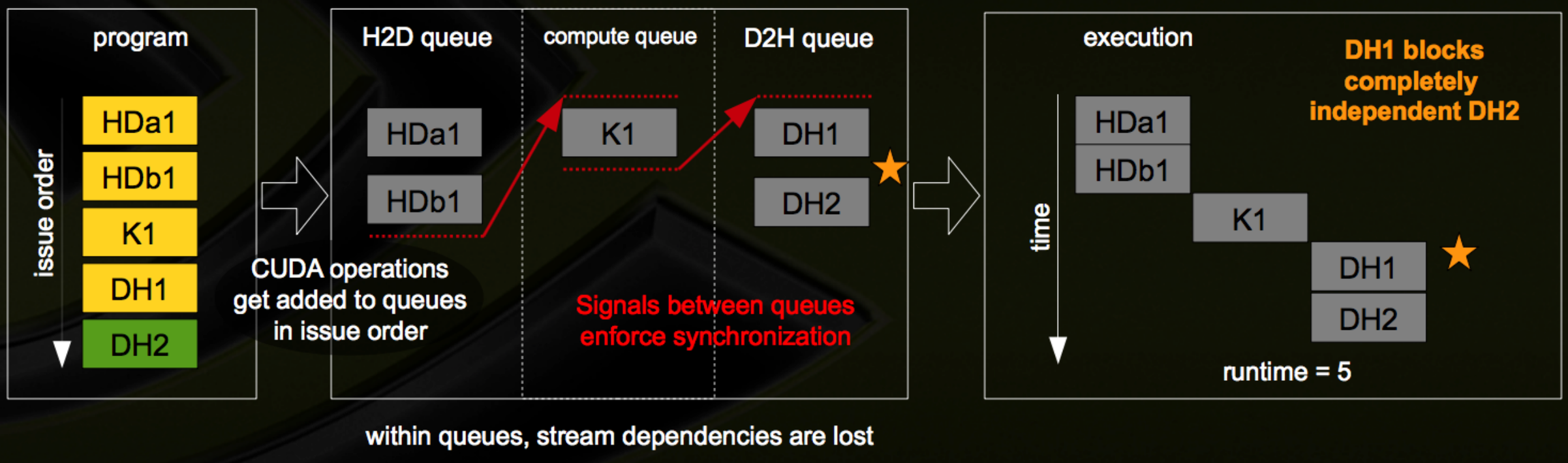
They’re basically copied from
http://on-demand.gputechconf.com/gtc-express/2011/presentations/StreamsAndConcurrencyWebinar.pdf

Streams

- A sequence of operations that execute in issue-order on the GPU
- Programming model used to effect concurrency
  - CUDA operations in different streams may run concurrently
  - CUDA operations from different streams may be interleaved
- Rules:
  - A CUDA operation is dispatched from the engine queue if:
    - Preceding calls in the same stream have completed,
    - Preceding calls in the same queue have been dispatched, and
    - Resources are available

Example

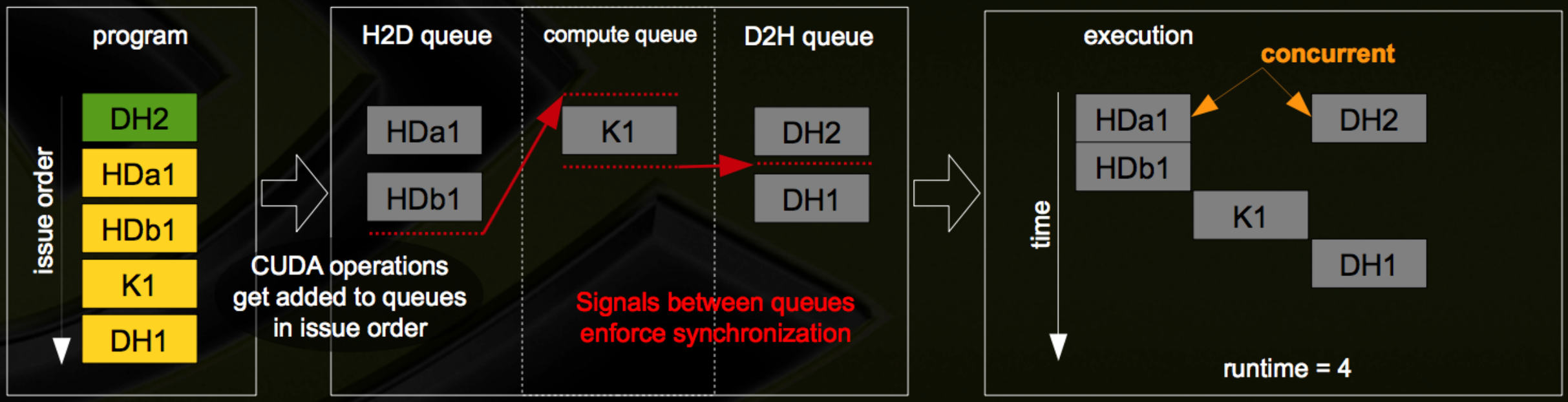
- **Two streams, stream 1 is issued first**
  - **Stream 1 : HDa1, HDb1, K1, DH1 (issued first)**
  - **Stream 2 : DH2 (completely independent of stream 1)**



Example

- **Two streams, stream 2 is issued first**
  - **Stream 1 : HDa1, HDb1, K1, DH1**
  - **Stream 2 : DH2 (issued first)**

issue order matters!



within queues, stream dependencies are lost