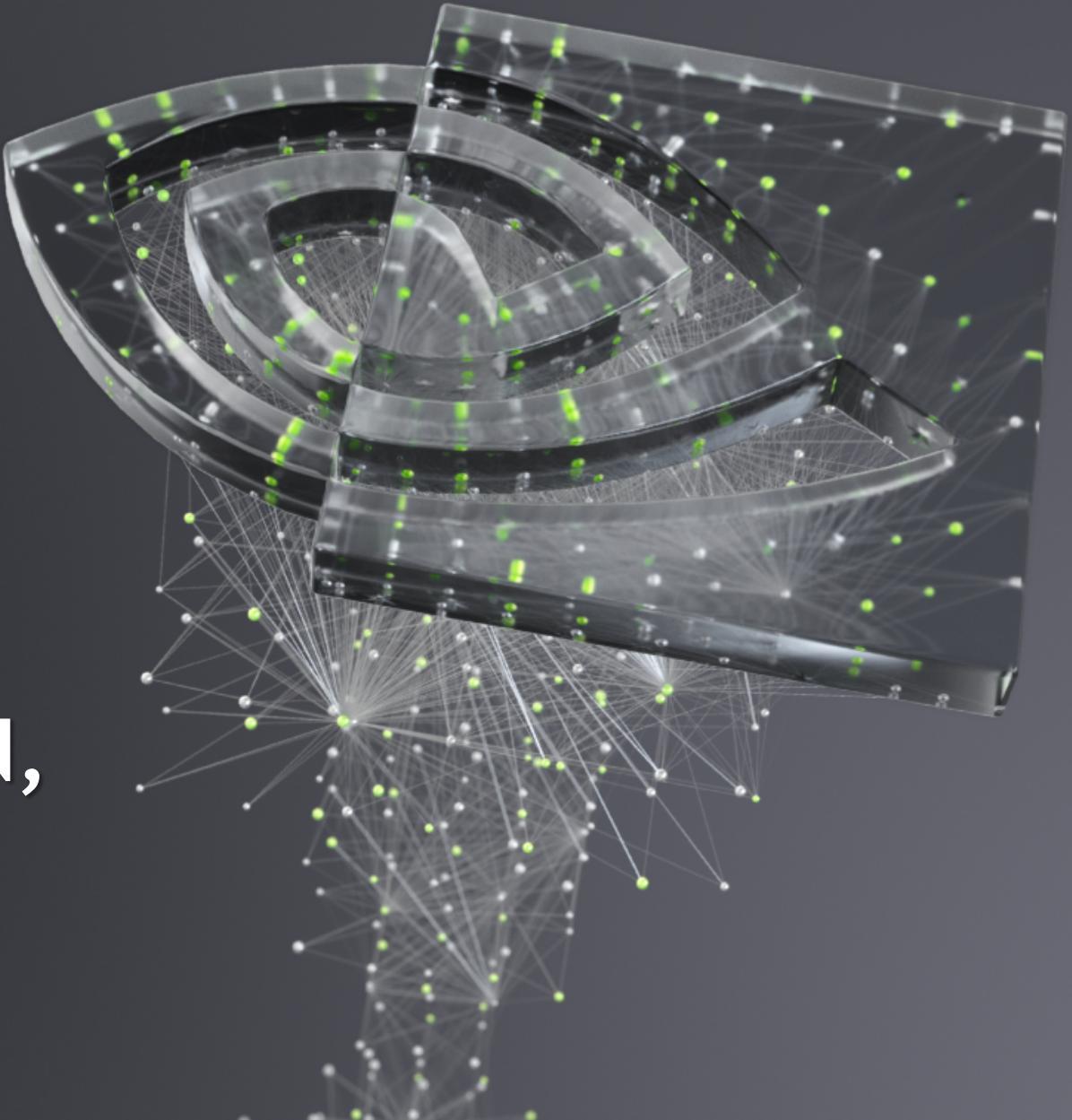




# CUDA OPTIMIZATION, PART 1

NVIDIA Corporation



# OUTLINE

- ▶ Architecture:
  - Kepler/Maxwell/Pascal/Volta
- ▶ Kernel optimizations
  - ▶ Launch configuration (use lots of threads)
- ▶ Part 2 (next session):
  - ▶ Global memory throughput (use memory efficiently)
  - ▶ Shared memory access

Most concepts in this presentation apply to *any language or API* on NVIDIA GPUs

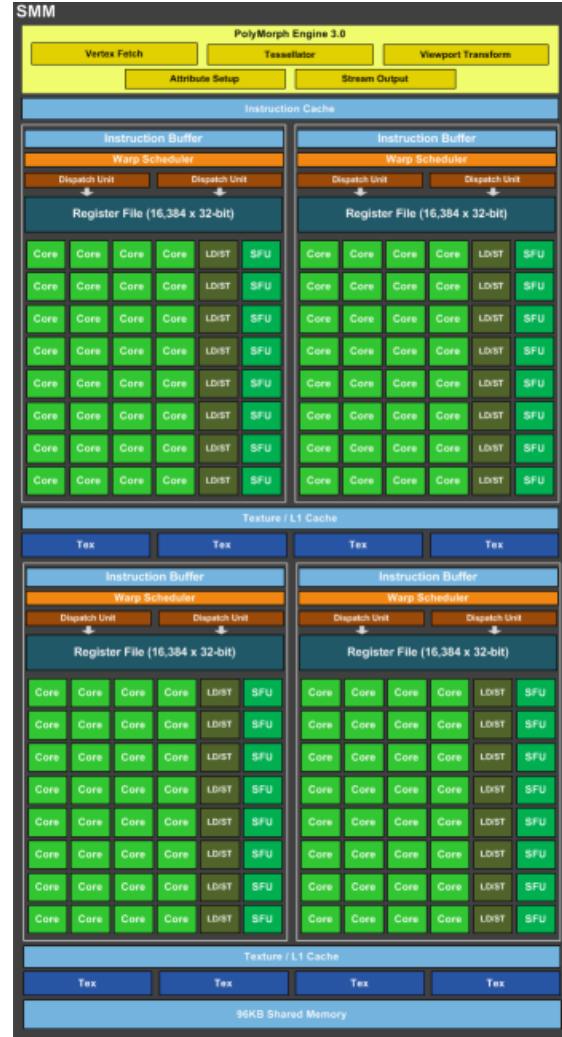
# KEPLER CC 3.5 SM (GK110)

- ▶ “SMX” (enhanced SM)
- ▶ 192 SP units (“cores”)
- ▶ 64 DP units
- ▶ LD/ST units, 64K registers
- ▶ 4 warp schedulers
- ▶ Each warp scheduler is dual-issue capable
- ▶ K20: 13 SMX’s, 5GB
- ▶ K20X: 14 SMX’s, 6GB
- ▶ K40: 15 SMX’s, 12GB



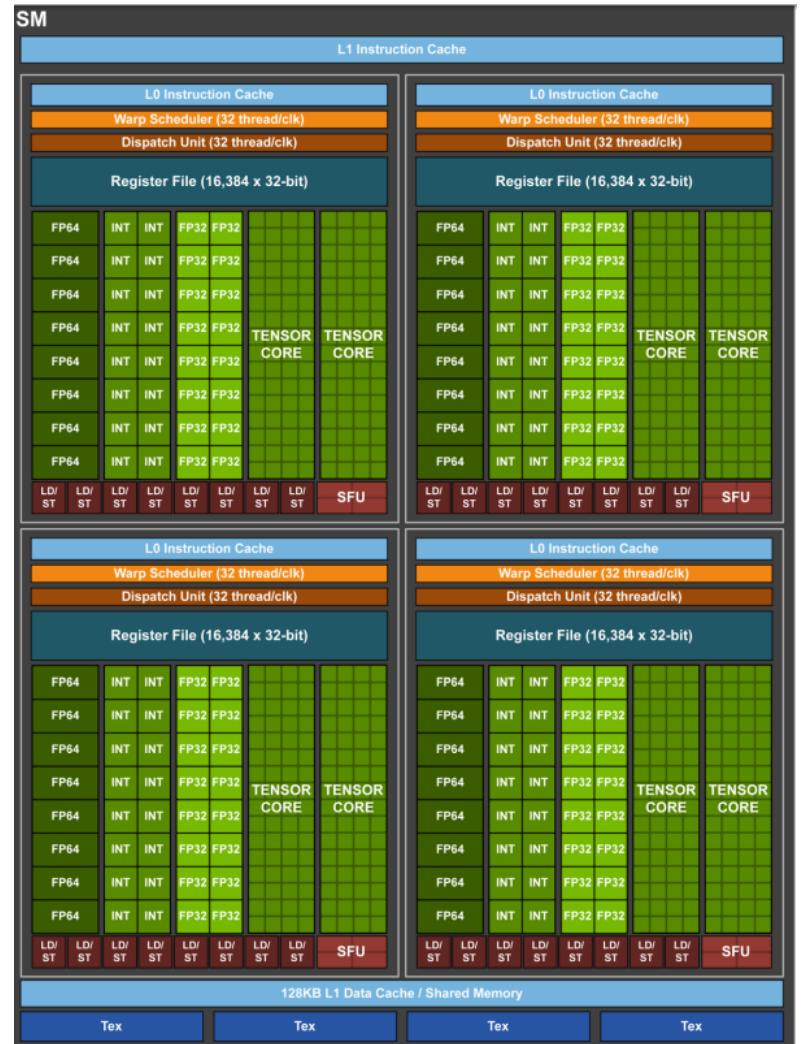
# MAXWELL/PASCAL CC5.2, CC6.1 SM

- ▶ “SMM” (enhanced SM)
- ▶ 128 SP units (“cores”)
- ▶ 4 DP units
- ▶ LD/ST units
- ▶ cc 6.1: INT8
- ▶ 4 warp schedulers
- ▶ Each warp scheduler is dual-issue capable
- ▶ M40: 24 SMM’s, 12/24GB
- ▶ P40: 30 SM’s, 24GB
- ▶ P4: 20 SM’s, 8GB



# PASCAL/VOLTA CC6.0/7.0

- ▶ 64 SP units (“cores”)
- ▶ 32 DP units
- ▶ LD/ST units
- ▶ FP16 @ 2x SP rate
- ▶ cc7.0: TensorCore
- ▶ P100/V100 2/4 warp schedulers
- ▶ Volta adds separate int32 units
- ▶ P100: 56 SM's, 16GB
- ▶ V100: 80 SM's, 16/32GB

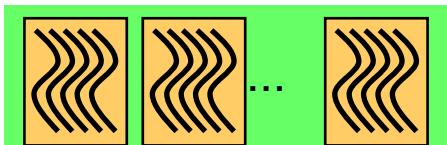


# EXECUTION MODEL

## Software

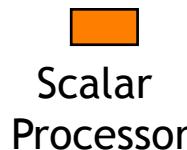


Thread Block

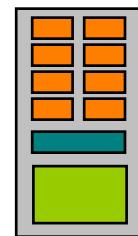


Grid

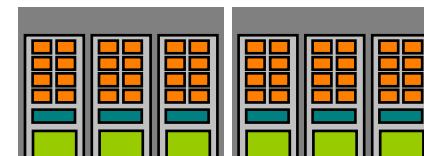
## Hardware



Scalar Processor



Multiprocessor



Device

Threads are executed by scalar processors

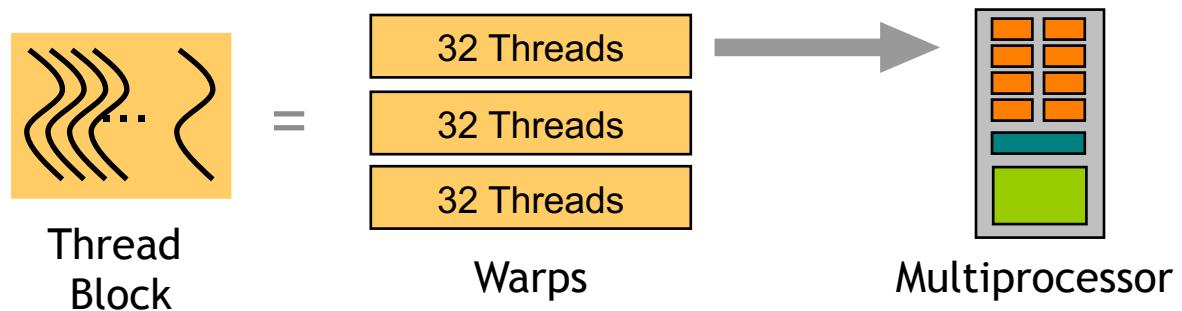
Thread blocks are executed on multiprocessors

Thread blocks do not migrate

Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)

A kernel is launched as a grid of thread blocks

# WARPS

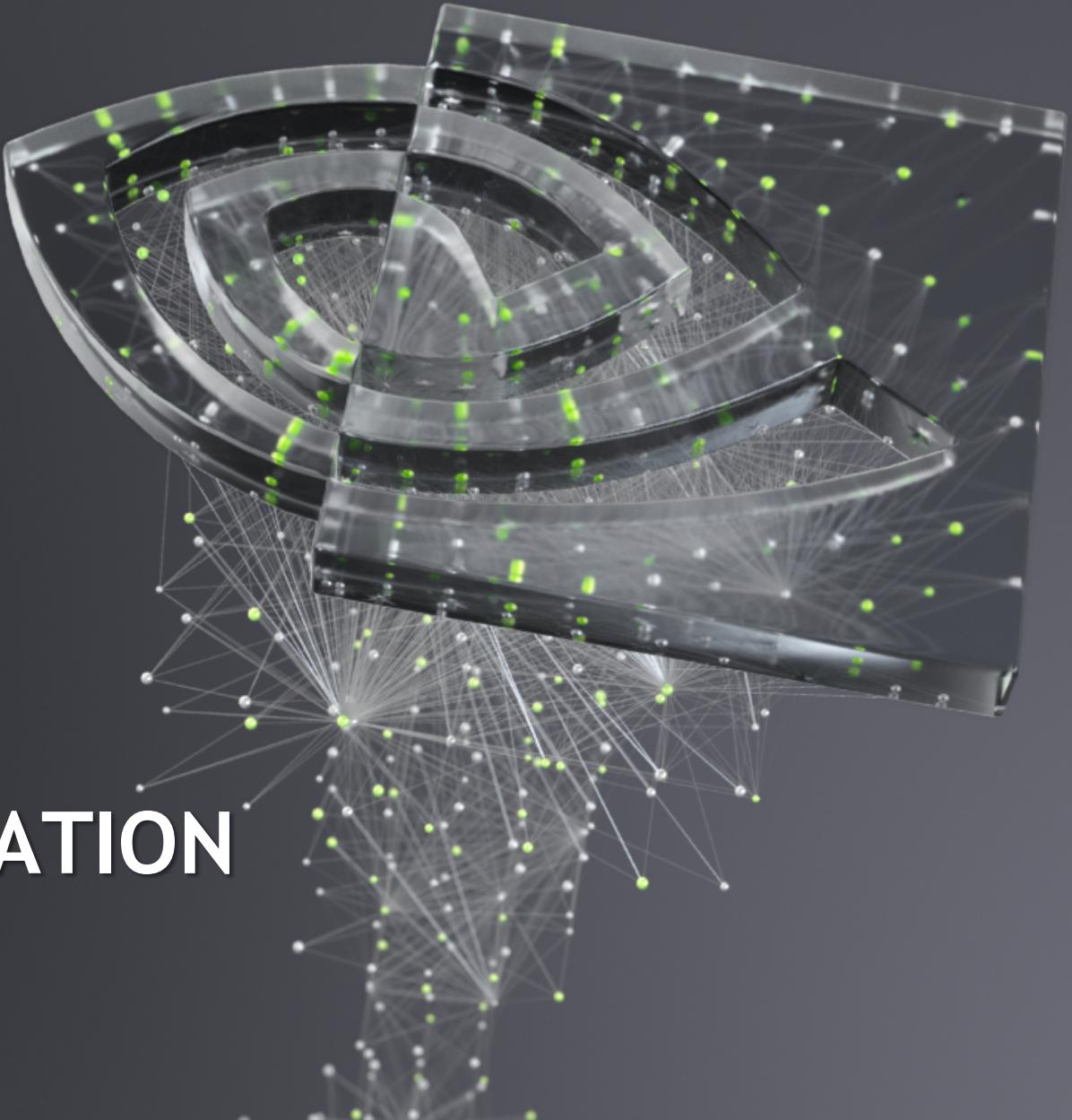


A thread block consists of 32-thread warps

A warp is executed physically in parallel (SIMD) on a multiprocessor



# LAUNCH CONFIGURATION



# LAUNCH CONFIGURATION

- ▶ Key to understanding:
  - ▶ Instructions are issued in order
  - ▶ A thread stalls when one of the operands isn't ready:
    - ▶ Memory read by itself doesn't stall execution
  - ▶ Latency is hidden by switching threads
    - ▶ GMEM latency: >100 cycles (varies by architecture/design)
    - ▶ Arithmetic latency: <100 cycles (varies by architecture/design)
- ▶ How many threads/threadblocks to launch?
- ▶ Conclusion:
  - ▶ Need enough threads to hide latency

# GPU LATENCY HIDING

- ▶ In CUDA C source code:
  - ▶ `int idx = threadIdx.x+blockDim.x*blockIdx.x;`
  - ▶ `c[idx] = a[idx] * b[idx];`
  
- ▶ In machine code:
  - ▶ I0: LD R0, a[idx];
  - ▶ I1: LD R1, b[idx];
  - ▶ I2: MPY R2,R0,R1

# GPU LATENCY HIDING - INSIDE THE SM

- ▶ I0: LD R0, a[idx];
  - ▶ I1: LD R1, b[idx];
  - ▶ I2: MPY R2,R0,R1
- clock cycles:
- C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0:

W1:

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

# GPU LATENCY HIDING - INSIDE THE SM

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: I0 I1

W1:

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

# GPU LATENCY HIDING - INSIDE THE SM

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: I0 I1

W1:

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

# GPU LATENCY HIDING - INSIDE THE SM

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: I0 I1

W1: I0

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

# GPU LATENCY HIDING - INSIDE THE SM

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: I0 I1

W1: I0 I1

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

# GPU LATENCY HIDING - INSIDE THE SM

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: 

W1: 

W2: 

W3:

W4:

W5:

W6:

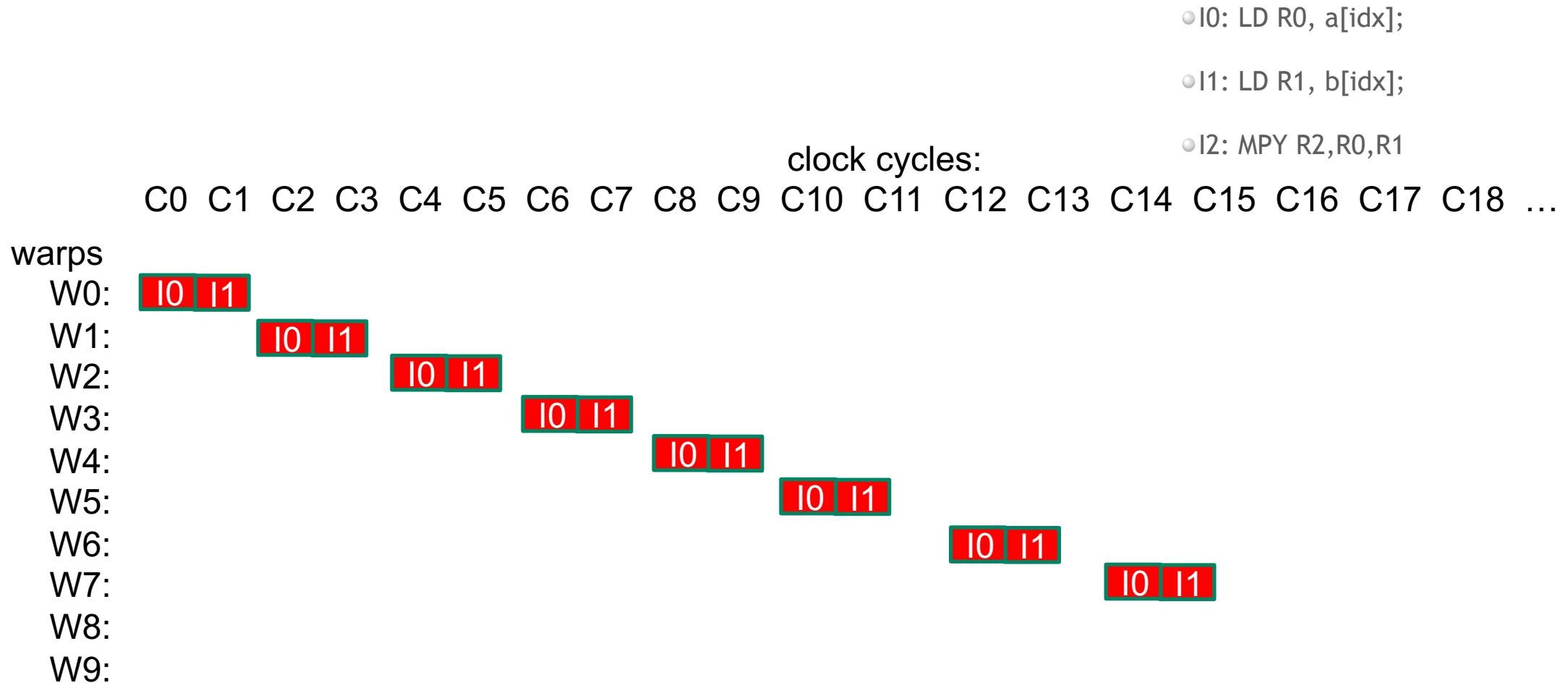
W7:

W8:

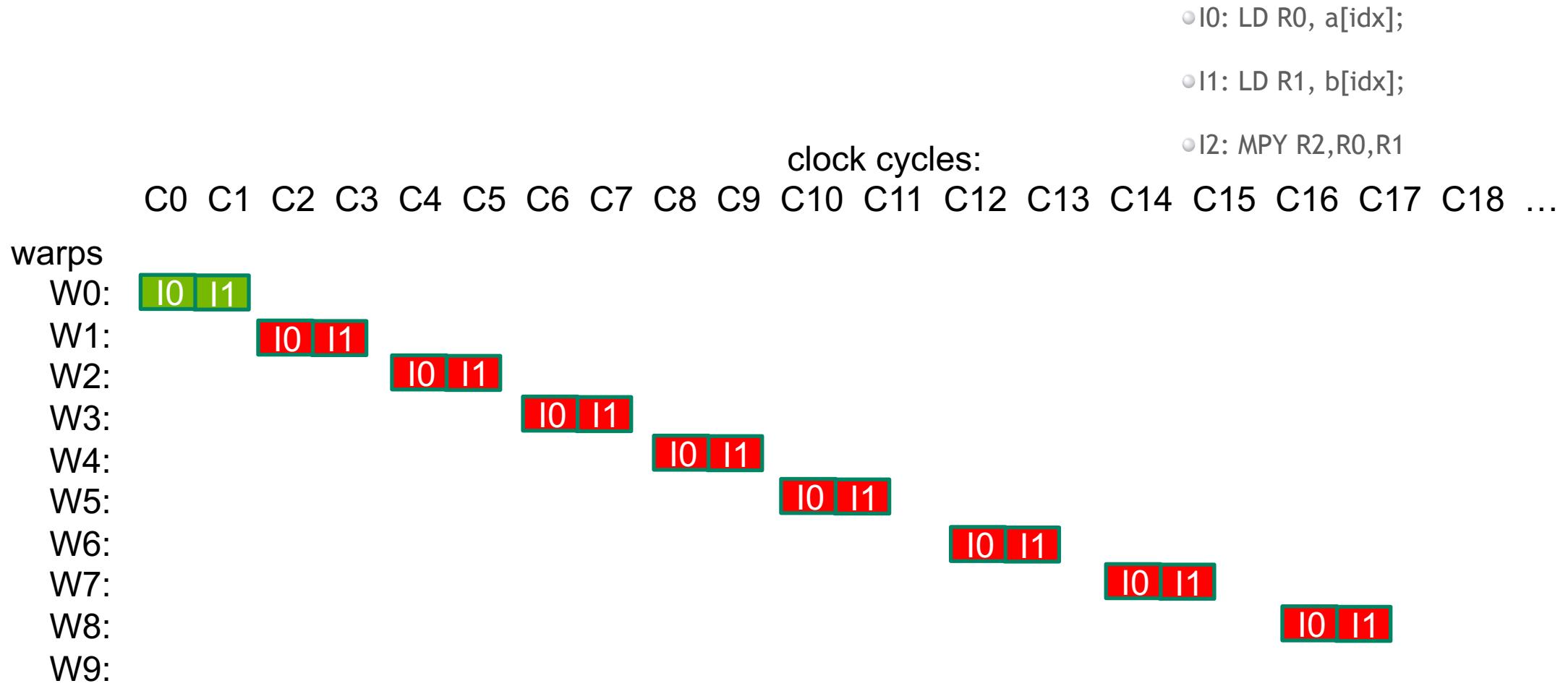
W9:

...

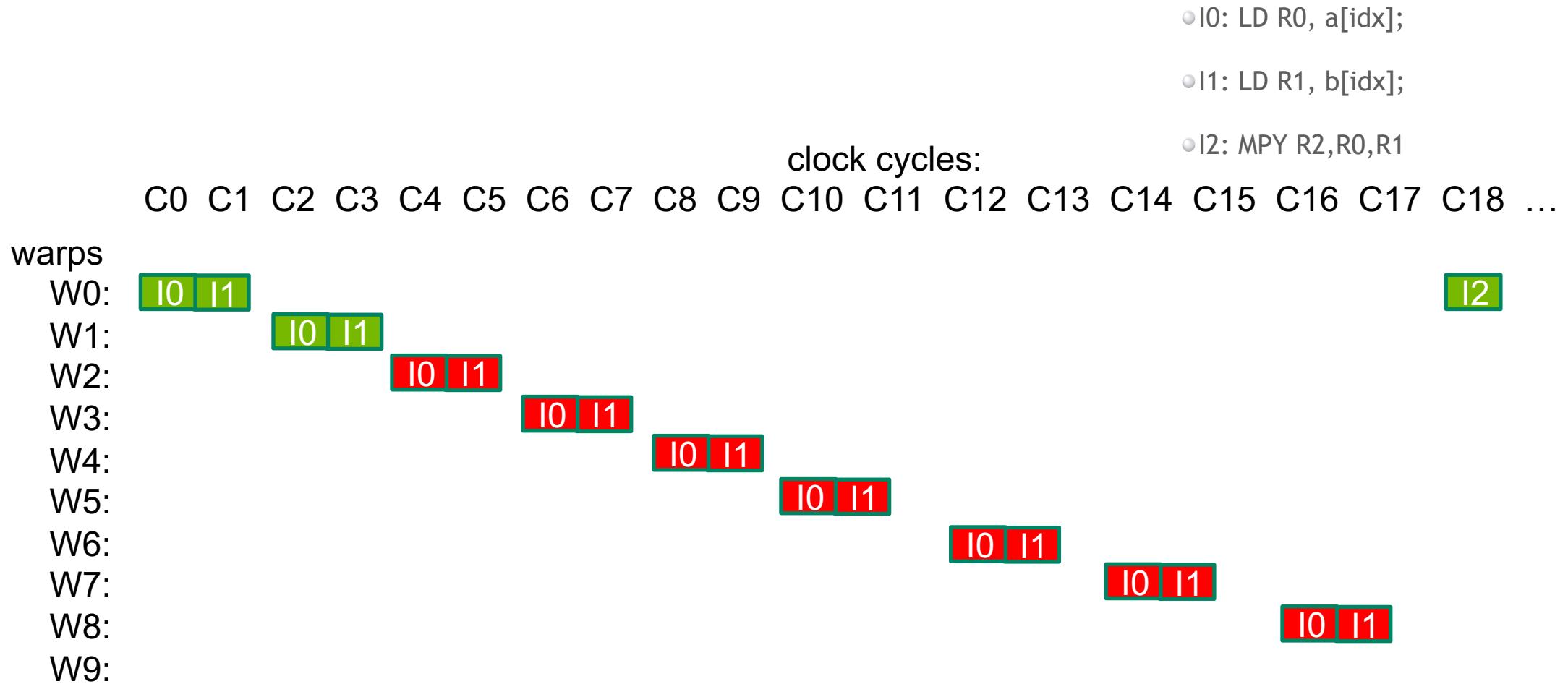
# GPU LATENCY HIDING - INSIDE THE SM



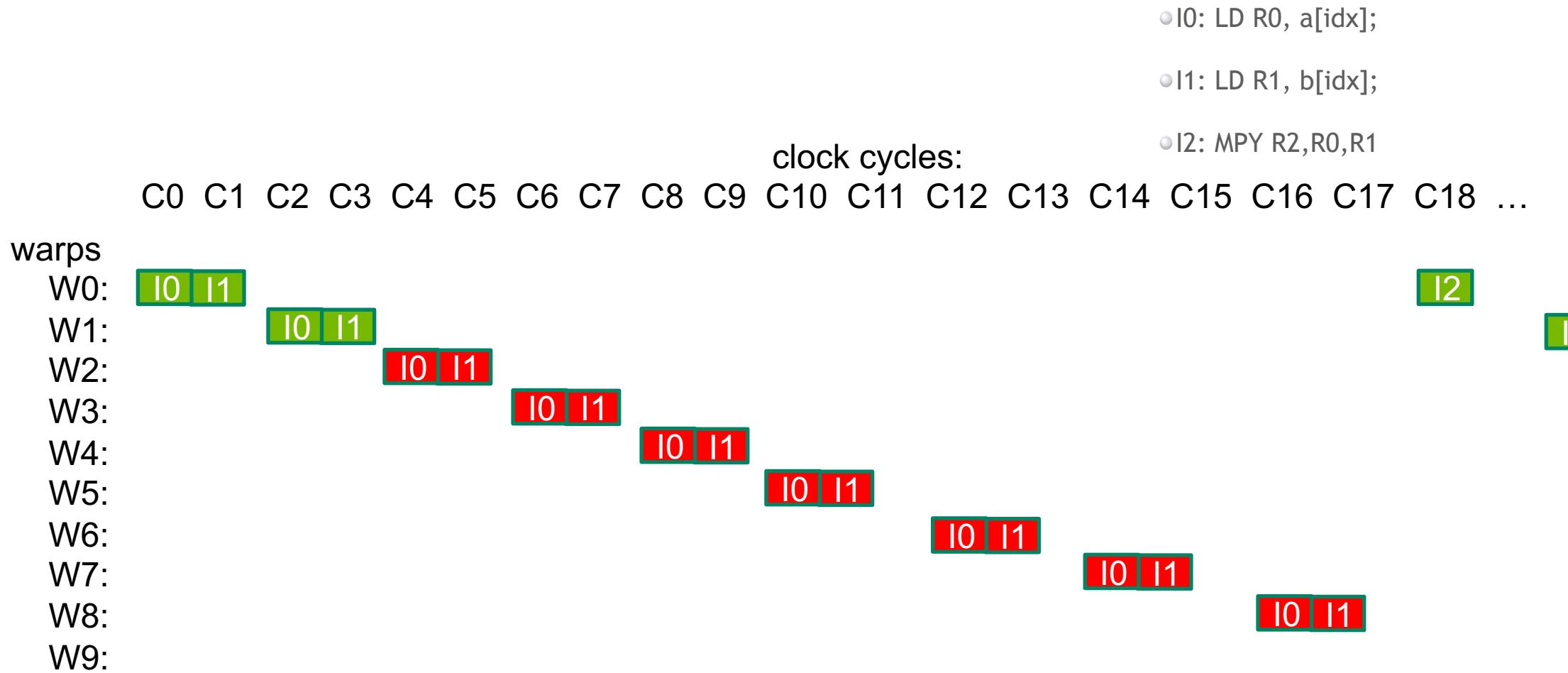
# GPU LATENCY HIDING - INSIDE THE SM



# GPU LATENCY HIDING - INSIDE THE SM



# GPU LATENCY HIDING - INSIDE THE SM



# LAUNCH CONFIGURATION

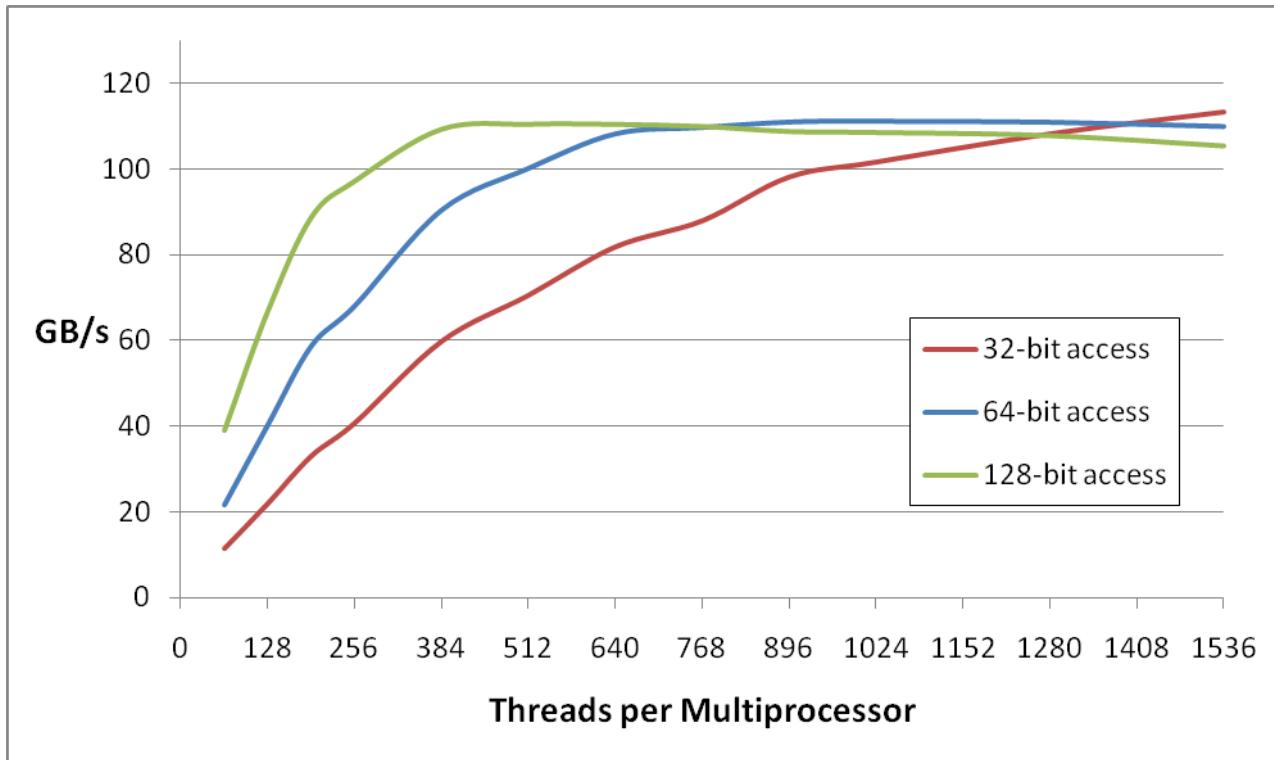
- ▶ Hiding arithmetic latency:
  - ▶ Need ~10's warps (~320 threads) per SM
  - ▶ Or, latency can also be hidden with independent instructions from the same warp
    - ▶ ->if instructions never depends on the output of preceding instruction, then only 5 warps are needed, etc.
- ▶ Maximizing global memory throughput:
  - ▶ Depends on the access pattern, and word size
  - ▶ Need enough memory transactions in flight to saturate the bus
    - ▶ Independent loads and stores from the same thread
    - ▶ Loads and stores from different threads
    - ▶ Larger word sizes can also help (float2 is twice the transactions of float, for example)

# MAXIMIZING MEMORY THROUGHPUT

Increment of an array of 64M elements

Two accesses per thread (load then store) - dependent, so really 1 access per thread at a time

theoretical bandwidth: ~120 GB/s



Several independent smaller accesses have the same effect as one larger one.

For example:

Four 32-bit ~ = one 128-bit

# LAUNCH CONFIGURATION: SUMMARY

- ▶ Need enough total threads to keep GPU busy
  - ▶ Typically, you'd like **512+ threads** per SM (aim for 2048 - maximum “occupancy”)
    - ▶ More if processing one fp32 element per thread
    - ▶ Of course, exceptions exist
- ▶ Threadblock configuration
  - ▶ Threads per block should be a **multiple of warp size (32)**
  - ▶ SM can concurrently execute **at least 16** thread blocks (Maxwell/Pascal/Volta: 32)
    - ▶ Really small thread blocks prevent achieving good occupancy
    - ▶ Really large thread blocks are less flexible
    - ▶ Could generally use **128-256 threads/block**, but use whatever is best for the application

# ASIDE: WHAT IS OCCUPANCY?

- ▶ A measure of the actual thread load in an SM, vs. peak theoretical/peak achievable
- ▶ CUDA includes an occupancy calculator spreadsheet
- ▶ Achievable occupancy is affected by limiters to occupancy
- ▶ Primary limiters:
  - ▶ Registers per thread (can be reported by the profiler, or can get at compile time)
  - ▶ Threads per threadblock
  - ▶ Shared memory usage

# SUMMARY

- ▶ GPU is a massively thread-parallel, latency hiding machine
- ▶ Kernel Launch Configuration:
  - ▶ Launch enough threads per SM to hide latency
  - ▶ Launch enough threadblocks to load the GPU
- ▶ Use analysis/profiling when optimizing:
  - ▶ “Analysis-driven Optimization” (future session)
  - ▶ -> Nsight Compute can show you information about whether you’ve saturated the compute subsystem or the memory subsystem.

# FUTURE SESSIONS

- ▶ Fundamental Optimization, Part 2
- ▶ Atomics, Reductions, Warp Shuffle
- ▶ Using Managed Memory
- ▶ Concurrency (streams, copy/compute overlap, multi-GPU)
- ▶ Analysis Driven Optimization
- ▶ Cooperative Groups

# FURTHER STUDY

- ▶ Optimization in-depth:
  - ▶ <http://on-demand.gputechconf.com/gtc/2013/presentations/S3466-Programming-Guidelines-GPU-Architecture.pdf>
- ▶ Analysis-Driven Optimization:
  - ▶ <http://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf>
- ▶ CUDA Best Practices Guide:
  - ▶ <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html>
- ▶ CUDA Tuning Guides:
  - ▶ <https://docs.nvidia.com/cuda/index.html#programming-guides>  
(Kepler/Maxwell/Pascal/Volta)

# HOMEWORK

- ▶ Log into Summit (ssh [username@home.ccs.ornl.gov](mailto:username@home.ccs.ornl.gov) -> ssh summit)
- ▶ Clone GitHub repository:
  - ▶ Git clone [git@github.com:olcf/cuda-training-series.git](https://github.com/olcf/cuda-training-series.git)
- ▶ Follow the instructions in the readme.md file:
  - ▶ <https://github.com/olcf/cuda-training-series/blob/master/exercises/hw3/readme.md>
- ▶ Prerequisites: basic linux skills, e.g. ls, cd, etc., knowledge of a text editor like vi/emacs, and some knowledge of C/C++ programming



# QUESTIONS?

