



ECE 498AL

Lectures 8: Threading Hardware in G80

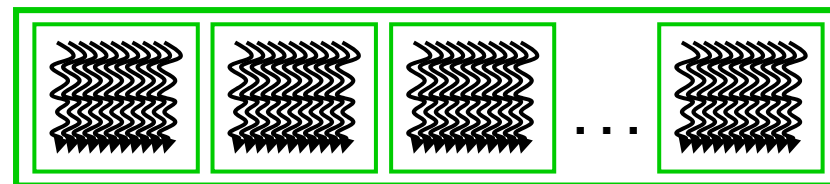
Single-Program Multiple-Data (SPMD)

- CUDA integrated CPU + GPU application C program
 - Serial C code executes on CPU
 - Parallel Kernel C code executes on GPU thread blocks

CPU Serial Code

GPU Parallel Kernel

KernelA<<< nBlk, nTid >>>(args);

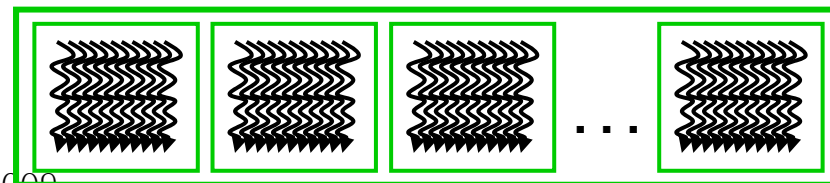


Grid 0

CPU Serial Code

GPU Parallel Kernel

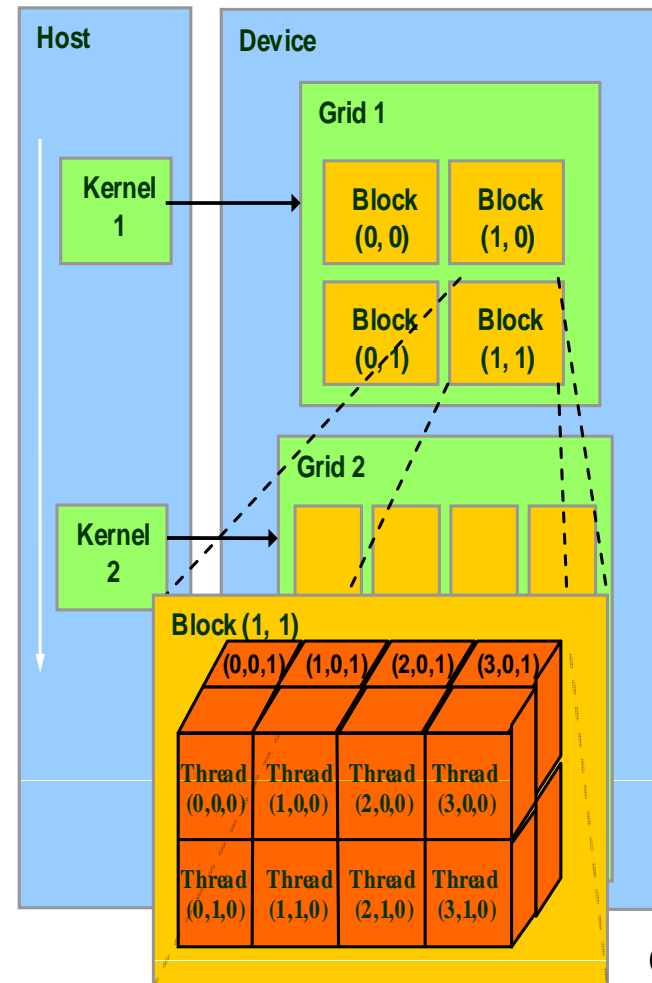
KernelB<<< nBlk, nTid >>>(args);



Grid 1

Grids and Blocks

- A kernel is executed as a grid of thread blocks
 - All threads share global memory space
- A thread block is a batch of threads that can cooperate with each other by:
 - Synchronizing their execution using barrier
 - Efficiently sharing data through a low latency shared memory
 - Two threads from two different blocks cannot cooperate

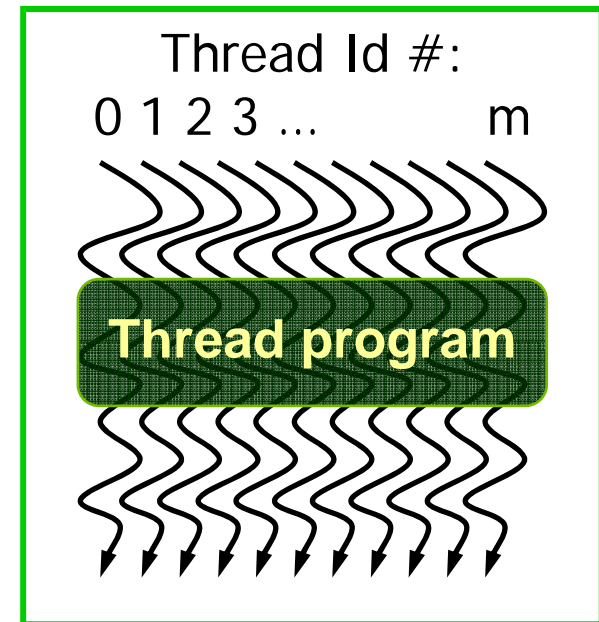


Courtesy: NDVIA

CUDA Thread Block: Review

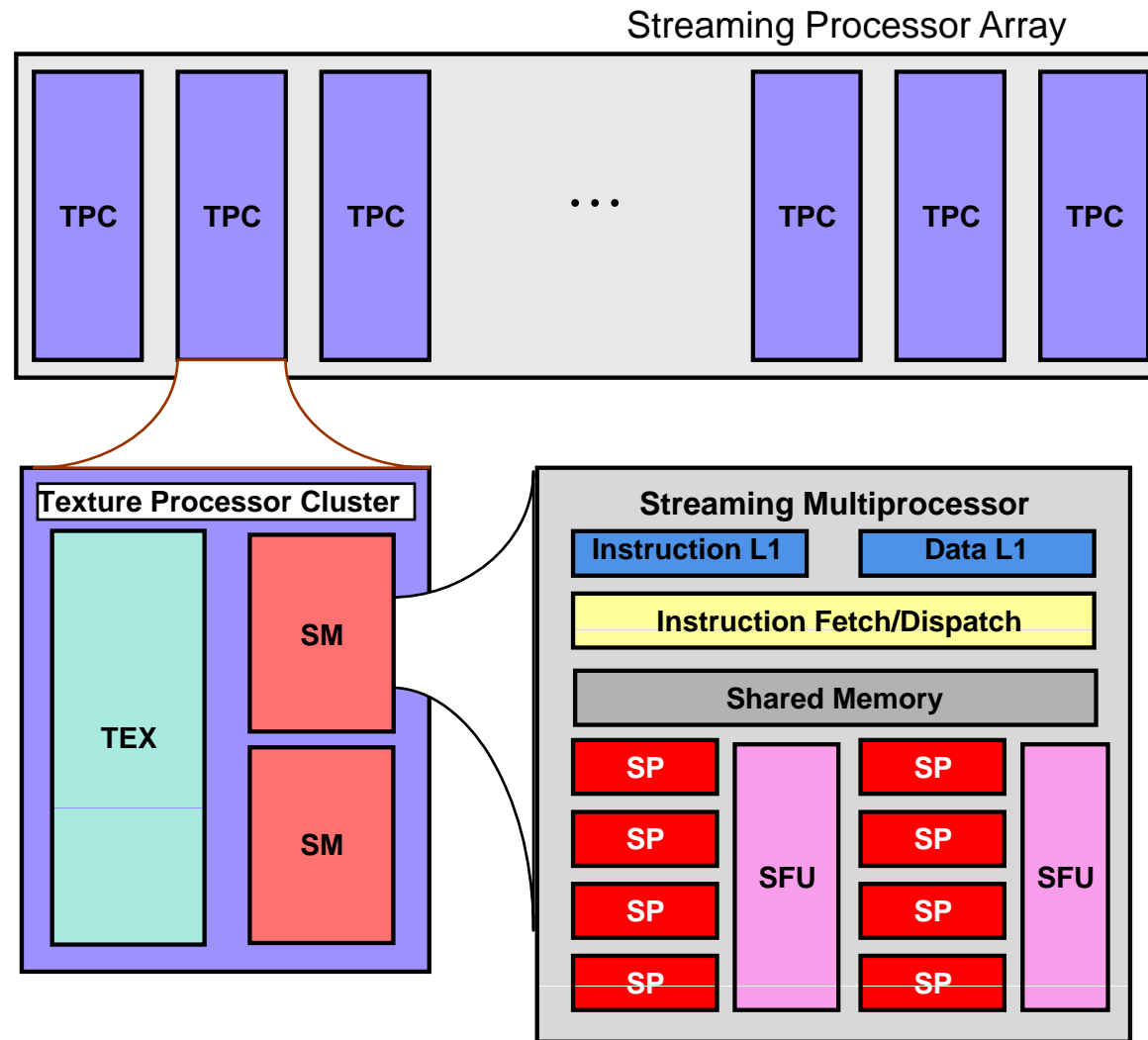
- Programmer declares (Thread) Block:
 - Block size 1 to **512** concurrent threads
 - Block shape 1D, 2D, or 3D
 - Block dimensions in threads
- All threads in a Block execute the same thread program
- Threads share data and synchronize while doing their share of the work
- Threads have **thread id** numbers within Block
- Thread program uses **thread id** to select work and address shared data

CUDA Thread Block



Courtesy: John Nickolls,
NVIDIA

GeForce-8 Series HW Overview

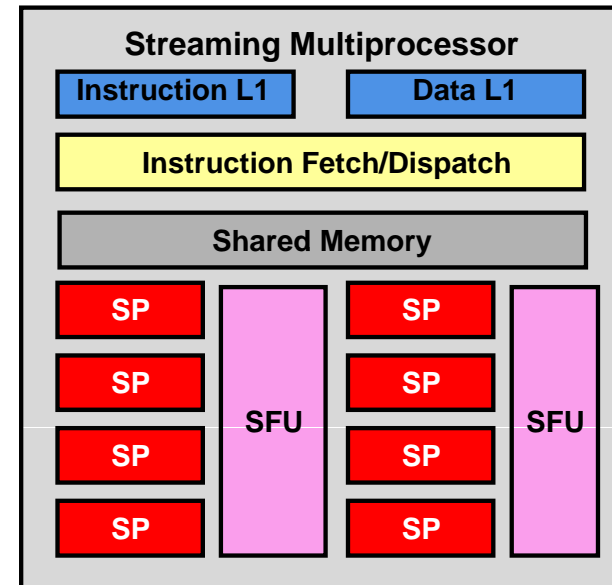


CUDA Processor Terminology

- SPA
 - Streaming Processor Array (variable across GeForce 8-series, 8 in GeForce8800)
- TPC
 - Texture Processor Cluster (2 SM + TEX)
- SM
 - Streaming Multiprocessor (8 SP)
 - Multi-threaded processor core
 - Fundamental processing unit for CUDA thread block
- SP
 - Streaming Processor
 - Scalar ALU for a single CUDA thread

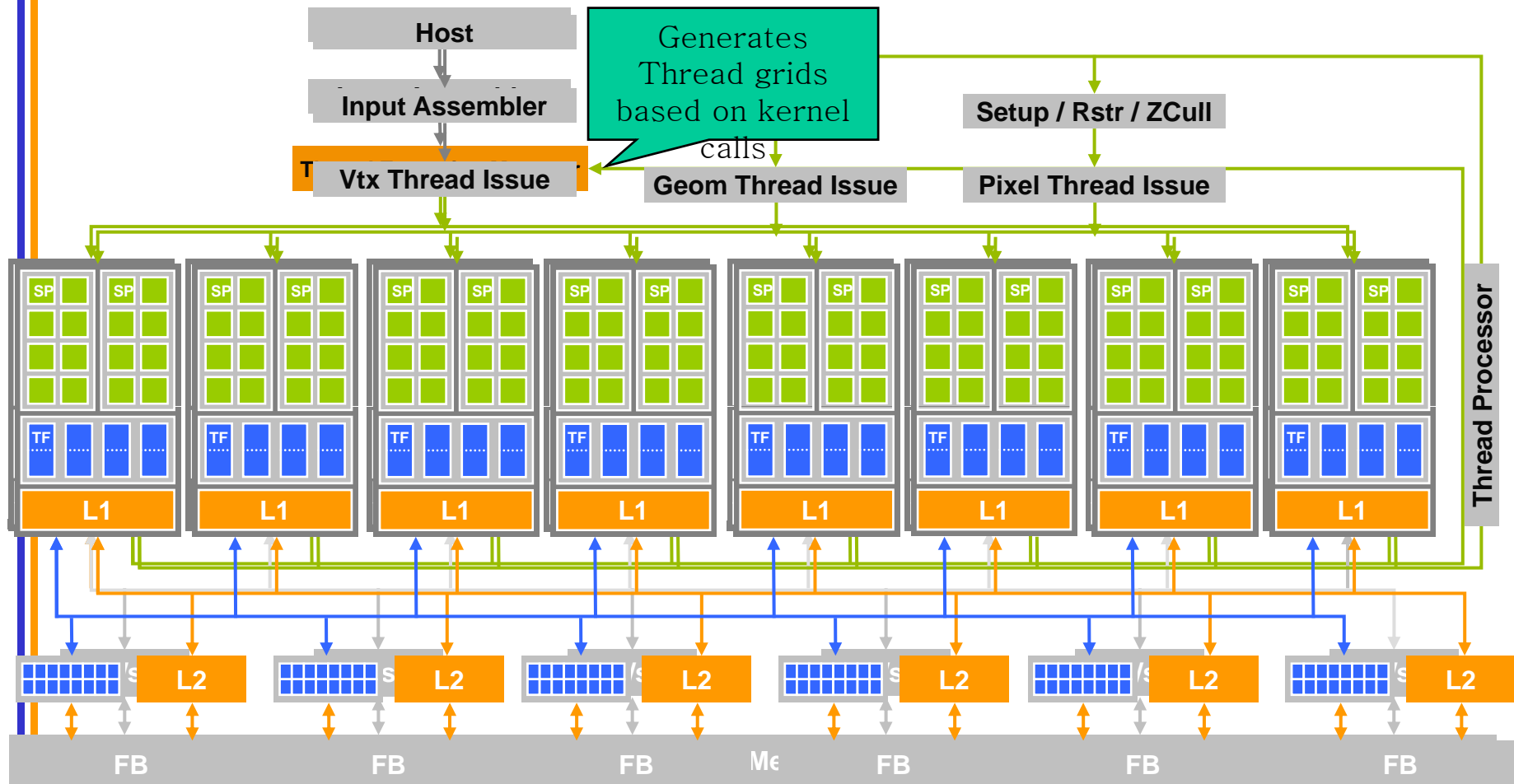
Streaming Multiprocessor (SM)

- Streaming Multiprocessor (SM)
 - 8 Streaming Processors (SP)
 - 2 Super Function Units (SFU)
- Multi-threaded instruction dispatch
 - 1 to 512 threads active
 - Shared instruction fetch per 32 threads
 - Cover latency of texture/memory loads
- 20+ GFLOPS
- 16 KB shared memory
- texture and global memory access



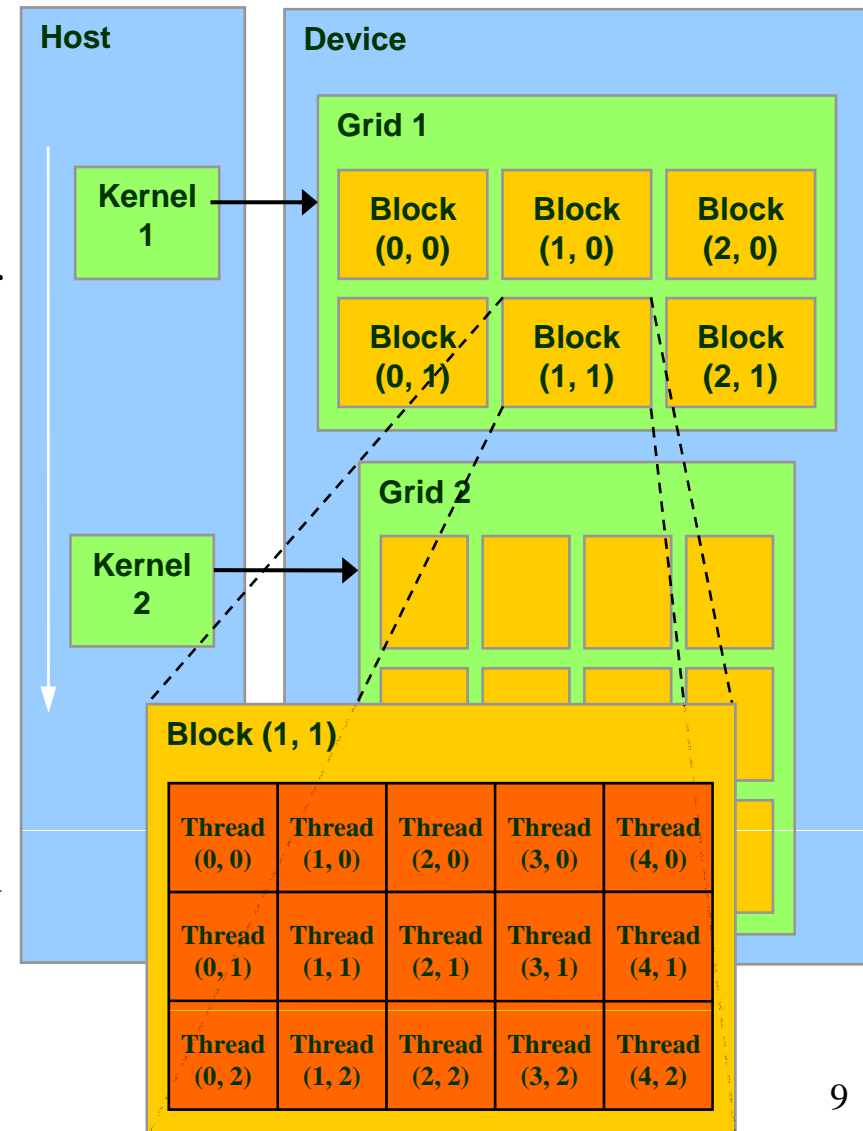
G80 Thread Computing Pipeline

- Precisions of CEs is programmable for threads processing
- Softer build to operating more as parallel processor computing

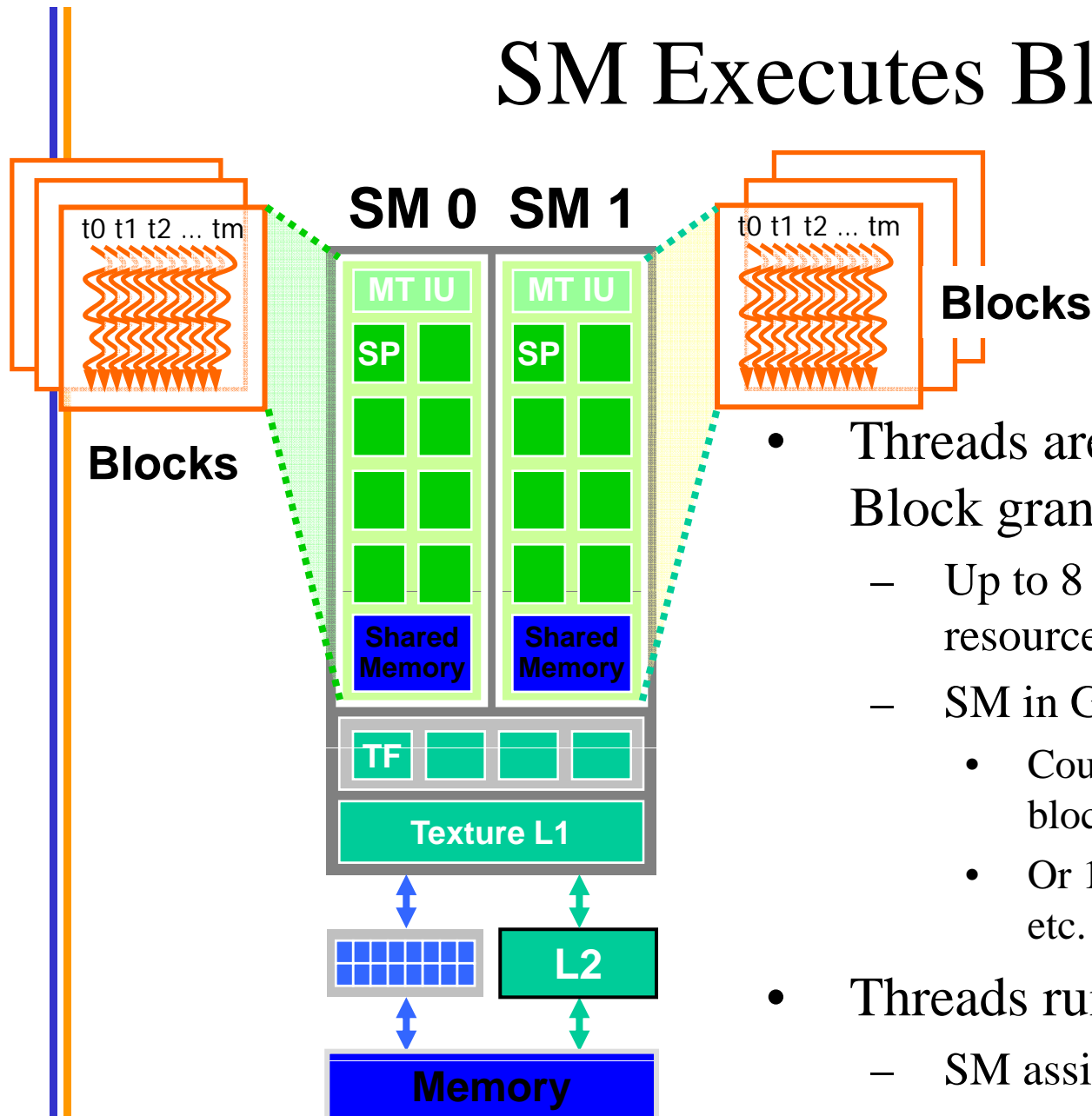


Thread Life Cycle in HW

- Grid is launched on the SPA
- Thread Blocks are serially distributed to all the SM's
 - Potentially >1 Thread Block per SM
- Each SM launches Warps of Threads
 - 2 levels of parallelism
- SM schedules and executes Warps that are ready to run
- As Warps and Thread Blocks complete, resources are freed
 - SPA can distribute more Thread Blocks



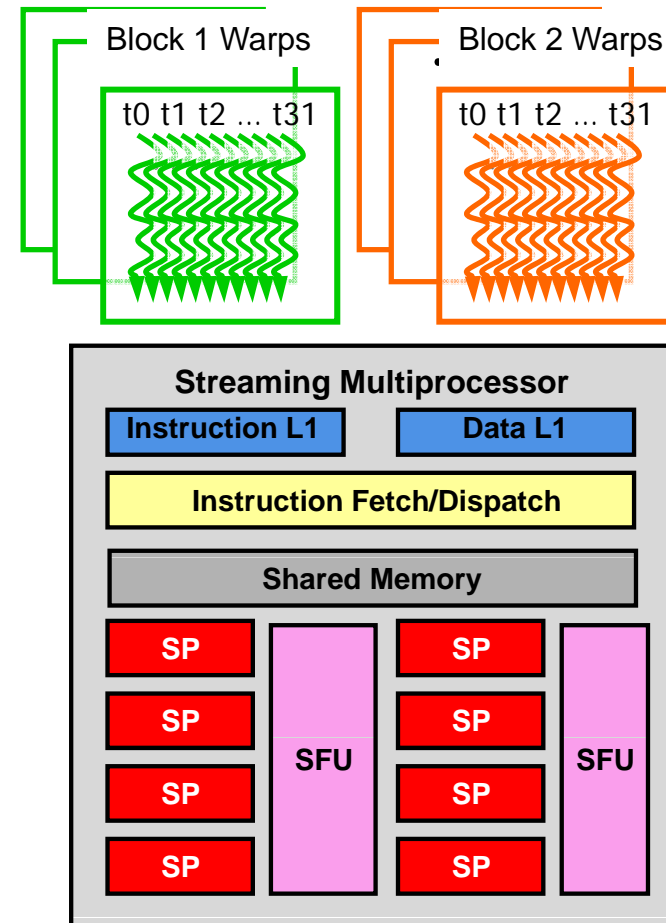
SM Executes Blocks



- Threads are assigned to SMs in Block granularity
 - Up to 8 Blocks to each SM as resource allows
 - SM in G80 can take up to 768 threads
 - Could be $256 \text{ (threads/block)} * 3 \text{ blocks}$
 - Or $128 \text{ (threads/block)} * 6 \text{ blocks, etc.}$
- Threads run concurrently
 - SM assigns/maintains thread id #s
 - SM manages/schedules thread execution

Thread Scheduling/Execution

- Each Thread Blocks is divided in 32-thread Warps
 - This is an implementation decision, not part of the CUDA programming model
- Warps are scheduling units in SM
- If 3 blocks are assigned to an SM and each Block has 256 threads, how many Warps are there in an SM?
 - Each Block is divided into $256/32 = 8$ Warps
 - There are $8 * 3 = 24$ Warps
 - At any point in time, only one of the 24 Warps will be selected for instruction fetch and execution.

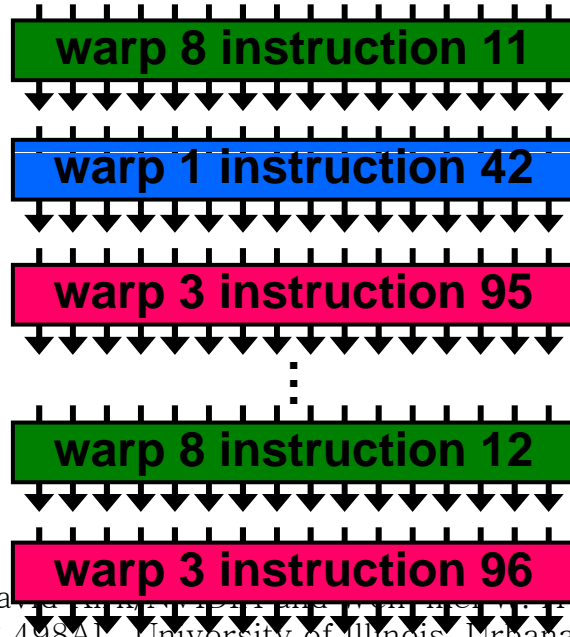


SM Warp Scheduling



SM multithreaded
Warp scheduler

time



- SM hardware implements zero-overhead Warp scheduling
 - Warps whose next instruction has its operands ready for consumption are eligible for execution
 - Eligible Warps are selected for execution on a prioritized scheduling policy
 - All threads in a Warp execute the same instruction when selected

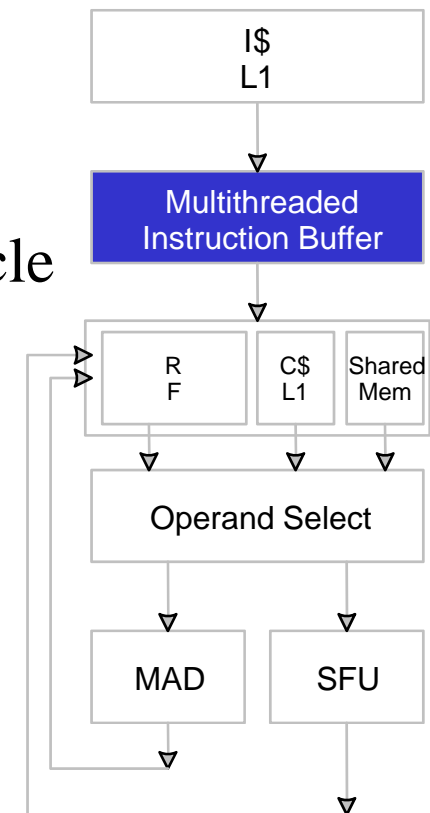
4 clock cycles needed to dispatch the same instruction for all threads in a Warp in G80

- If one global memory access is needed for every 4 instructions
- A minimal of 13 Warps are needed to

fully tolerate 200-cycle memory latency¹²

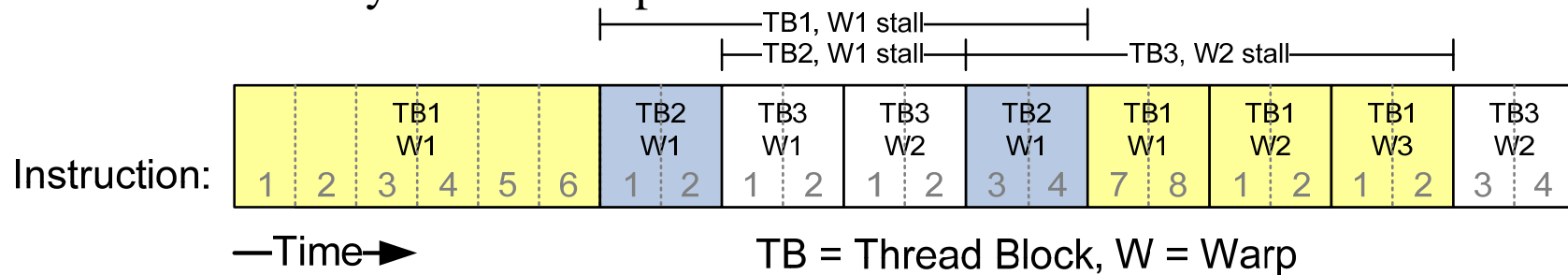
SM Instruction Buffer – Warp Scheduling

- Fetch one warp instruction/cycle
 - from instruction L1 cache
 - into any instruction buffer slot
- Issue one “ready-to-go” warp instruction/cycle
 - from any warp - instruction buffer slot
 - operand scoreboarding used to prevent hazards
- Issue selection based on round-robin/age of warp
- SM broadcasts the same instruction to 32 Threads of a Warp



Scoreboarding

- All register operands of all instructions in the Instruction Buffer are scoreboarded
 - Instruction becomes ready after the needed values are deposited
 - prevents hazards
 - cleared instructions are eligible for issue
- Decoupled Memory/Processor pipelines
 - any thread can continue to issue instructions until scoreboarding prevents issue
 - allows Memory/Processor ops to proceed in shadow of other waiting Memory/Processor ops



Granularity Considerations

- For Matrix Multiplication, should I use 4X4, 8X8, 16X16 or 32X32 tiles?
 - For 4X4, we have 16 threads per block, Since each SM can take up to 768 threads, the thread capacity allows 48 blocks. However, each SM can only take up to 8 blocks, thus there will be only 128 threads in each SM!
 - There are 8 warps but each warp is only half full.
 - For 8X8, we have 64 threads per Block. Since each SM can take up to 768 threads, it could take up to 12 Blocks. However, each SM can only take up to 8 Blocks, only 512 threads will go into each SM!
 - There are 16 warps available for scheduling in each SM
 - Each warp spans four slices in the y dimension
 - For 16X16, we have 256 threads per Block. Since each SM can take up to 768 threads, it can take up to 3 Blocks and achieve full capacity unless other resource considerations overrule.
 - There are 24 warps available for scheduling in each SM
 - Each warp spans two slices in the y dimension
 - For 32X32, we have 1024 threads per Block. Not even one can fit into an SM!