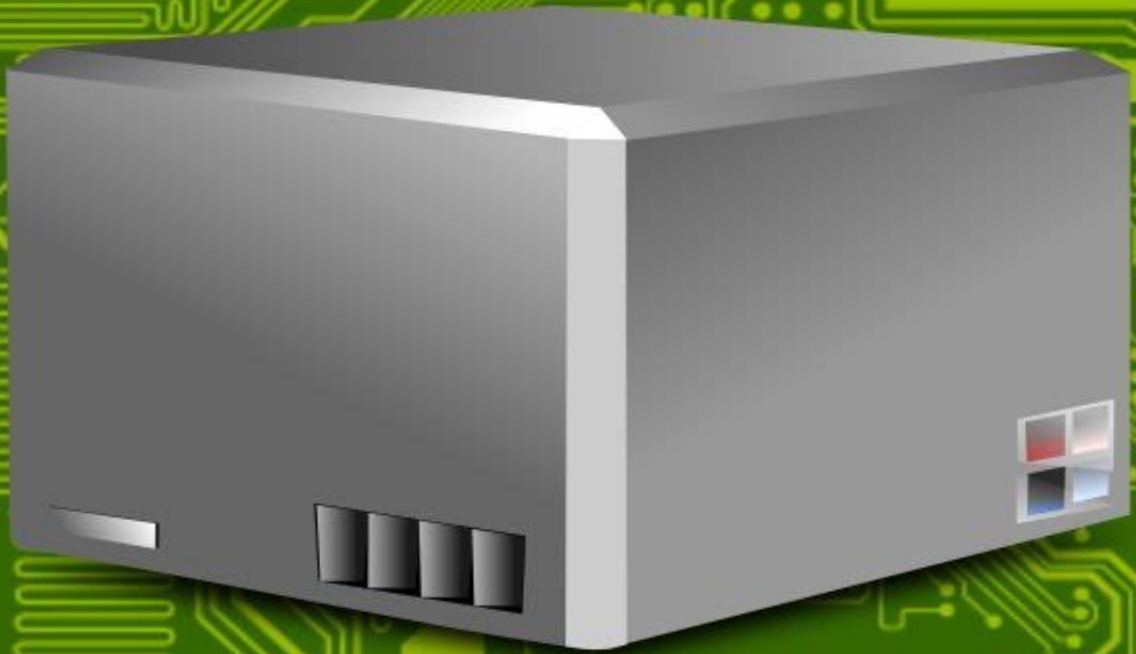
GPU Computing

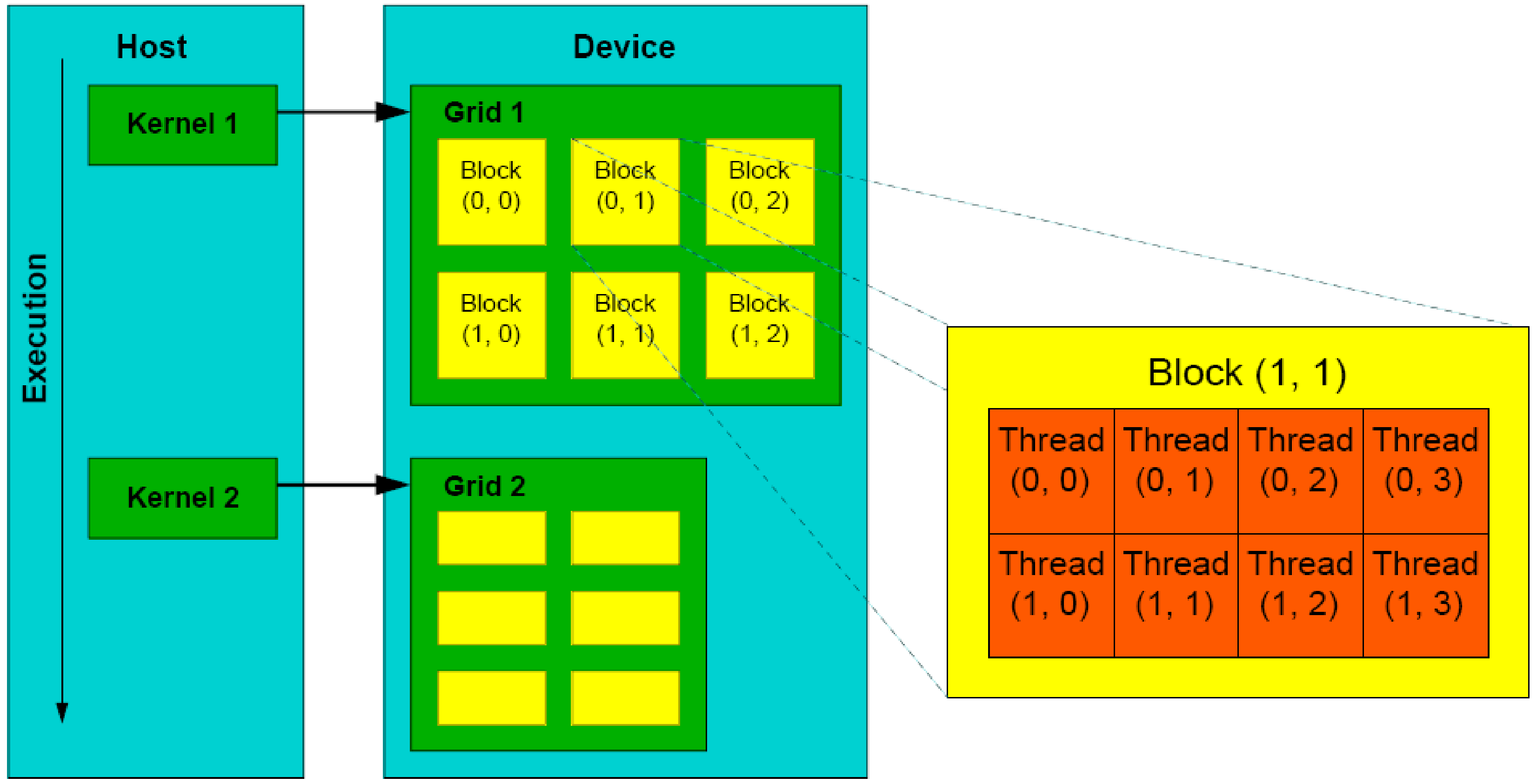


GPU 硬件架构综述

Hui Liu
Email: hui.sc.liu@gmail.com

CUDA Refresher

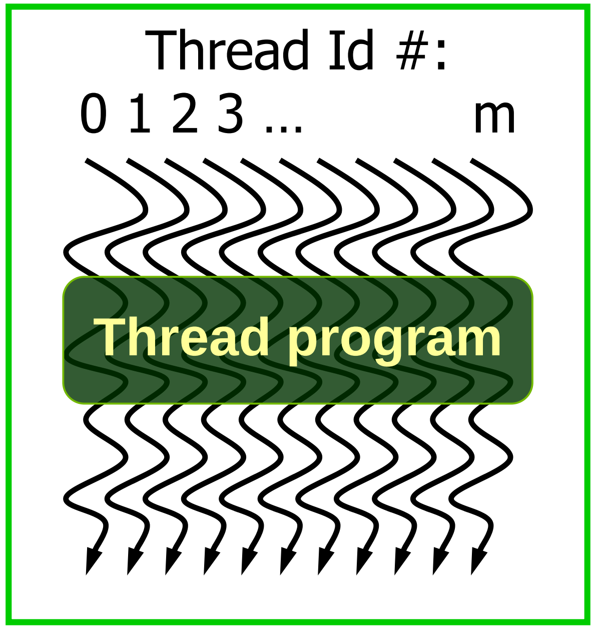
- Grids of Blocks
- Blocks of Threads



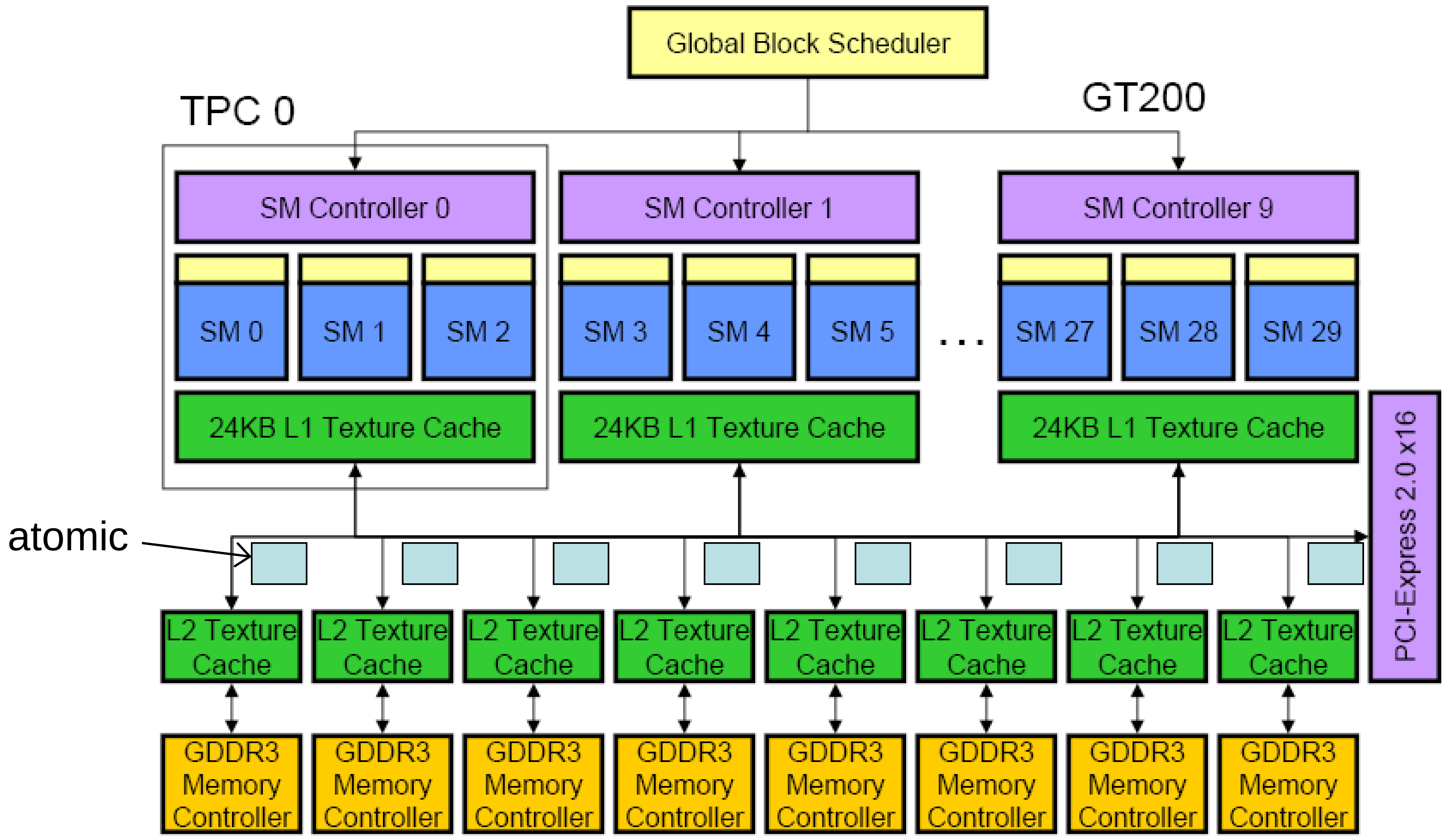
Thread Blocks Refresher

- Programmer declares (Thread) Block:
  - Block size 1 to **512 (1024, 2048)** concurrent threads
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads

- All threads in a Block execute the same thread program
- Threads have thread id numbers within Block
- Threads share data and synchronize while doing their share of the work
- Thread program uses thread id to select work and address shared data



GT200 Architecture Overview

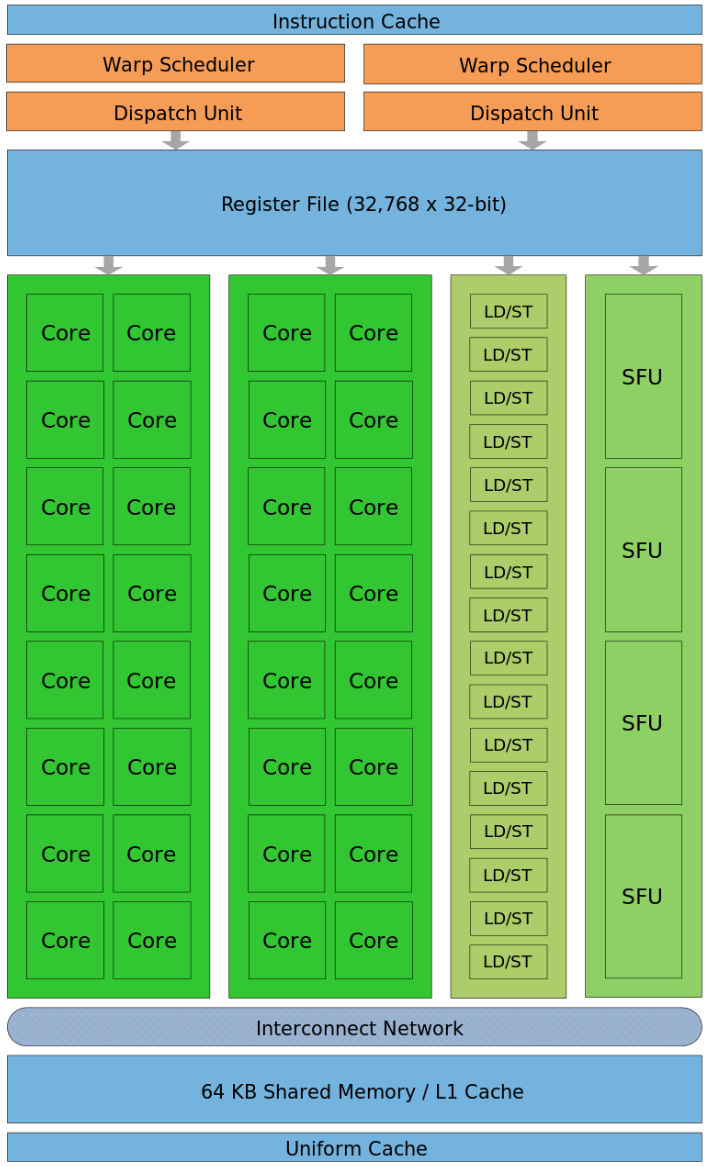


Terminology

- **SPA**
  - Streaming Processor Array
- **TPC/GPC**
  - Texture (Graphics) Processor Cluster
    - 3 SM + TEX
- **SM**
  - Streaming Multiprocessor (32 SP)
  - Multi-threaded processor core
  - Fundamental processing unit for CUDA thread block
- **SP (CUDA Core)**
  - Streaming Processor

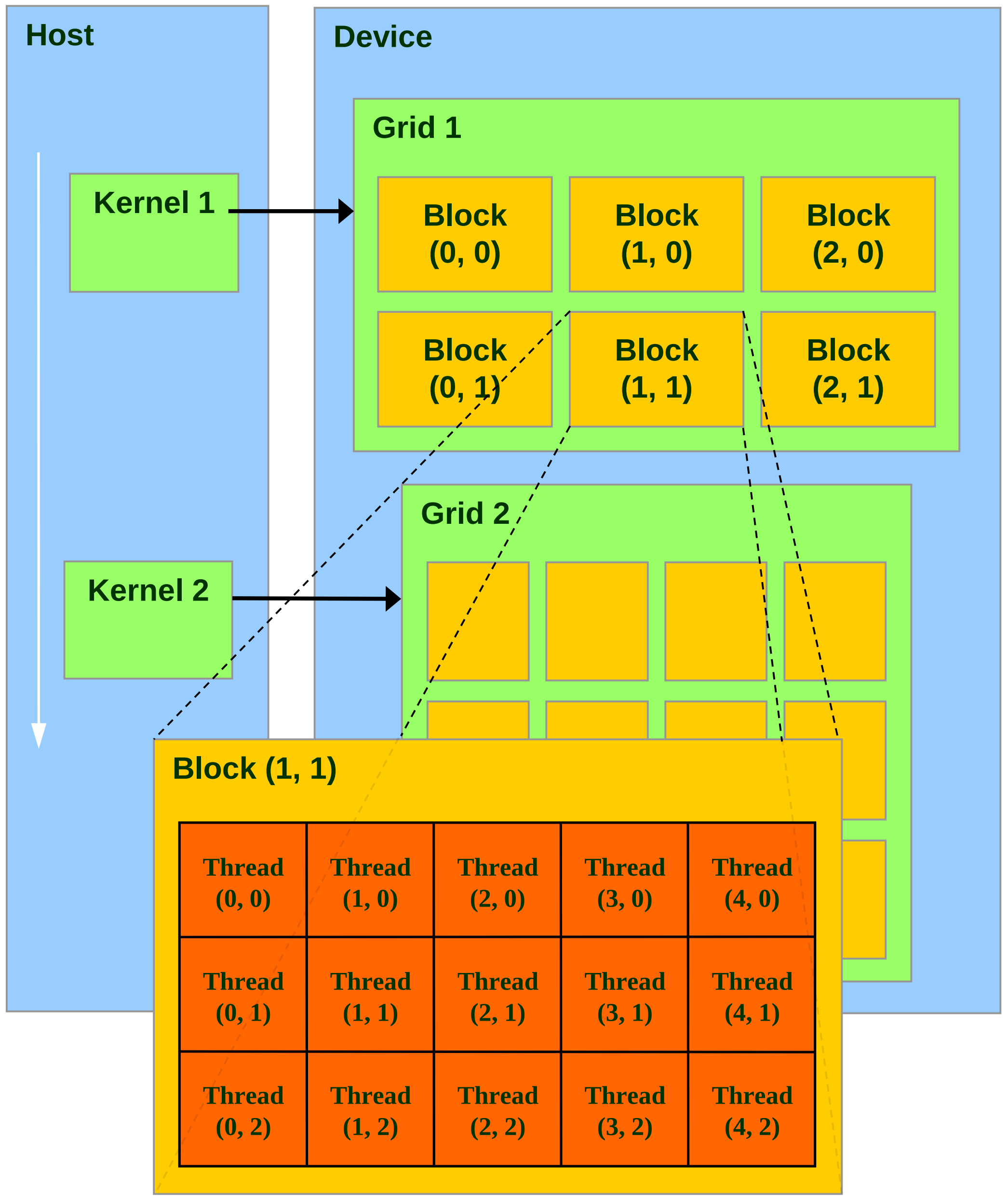
Stream Multiprocessor Overview

- Streaming Multiprocessor (SM)
  - 32 Streaming Processors (SP)
  - 4 Super Function Units (SFU)
- Multi-threaded instruction dispatch
  - 1 to 512 (1024, 2048) threads active
  - Shared instruction fetch per 32 threads
  - Cover latency of texture/memory loads
- 64KB shared memory / L1 cache
- DRAM texture and memory access



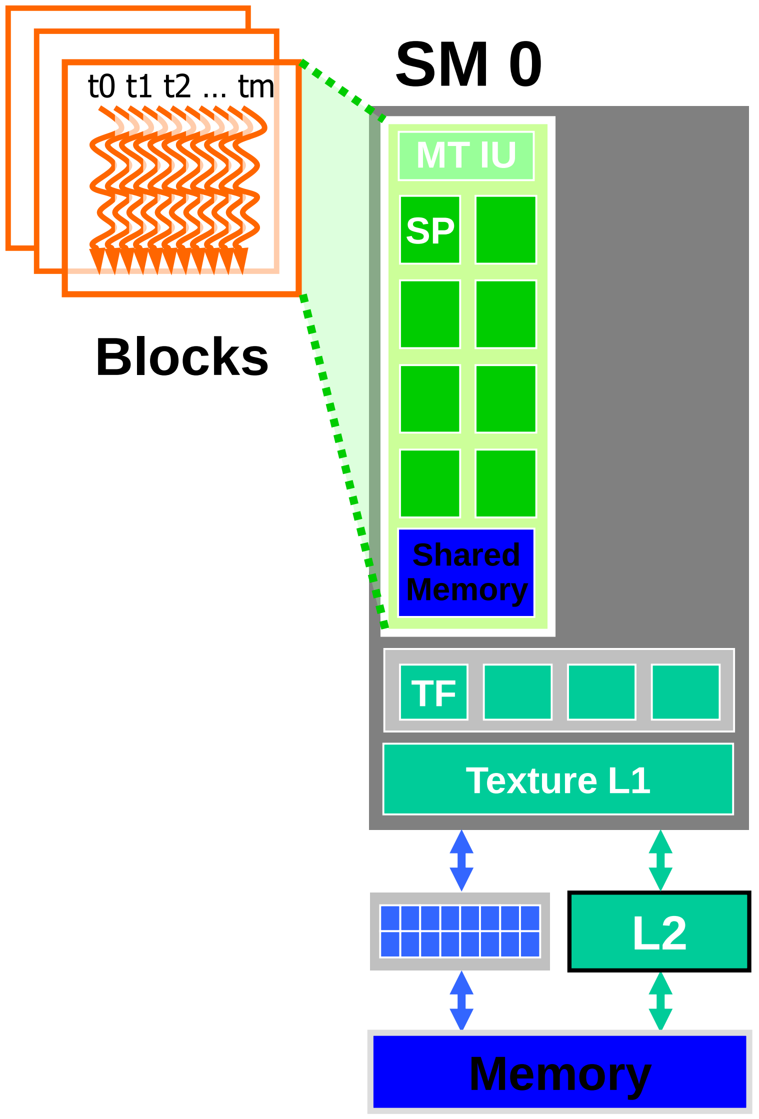
Thread Life

- 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



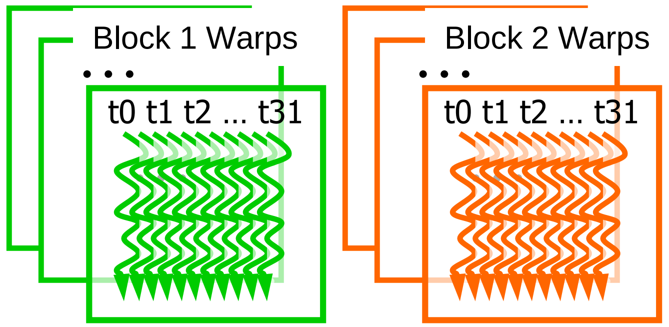
Stream Multiprocessors Execute Blocks

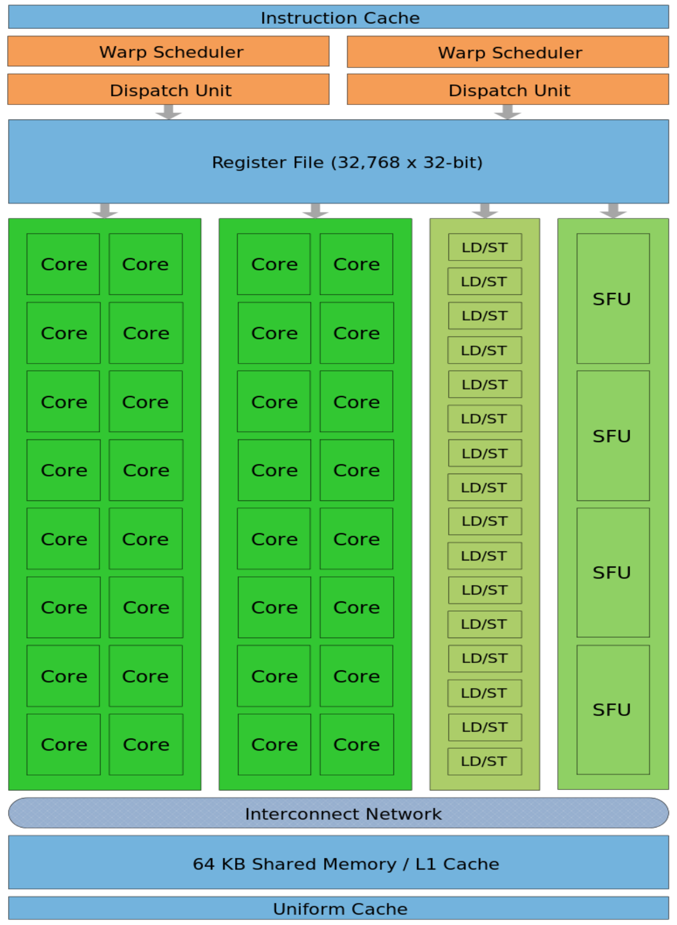
- Threads are assigned to SMs in Block granularity
  - Up to 8 (and more) Blocks to each SM as resource allows
  - SM in G200 can take up to 1K threads
    - Could be 256 (threads/block) * 4 blocks
    - Or 128 (threads/block) * 8 blocks, etc.
- Threads run concurrently
  - SM assigns/maintains thread id #s
  - SM manages/schedules thread execution



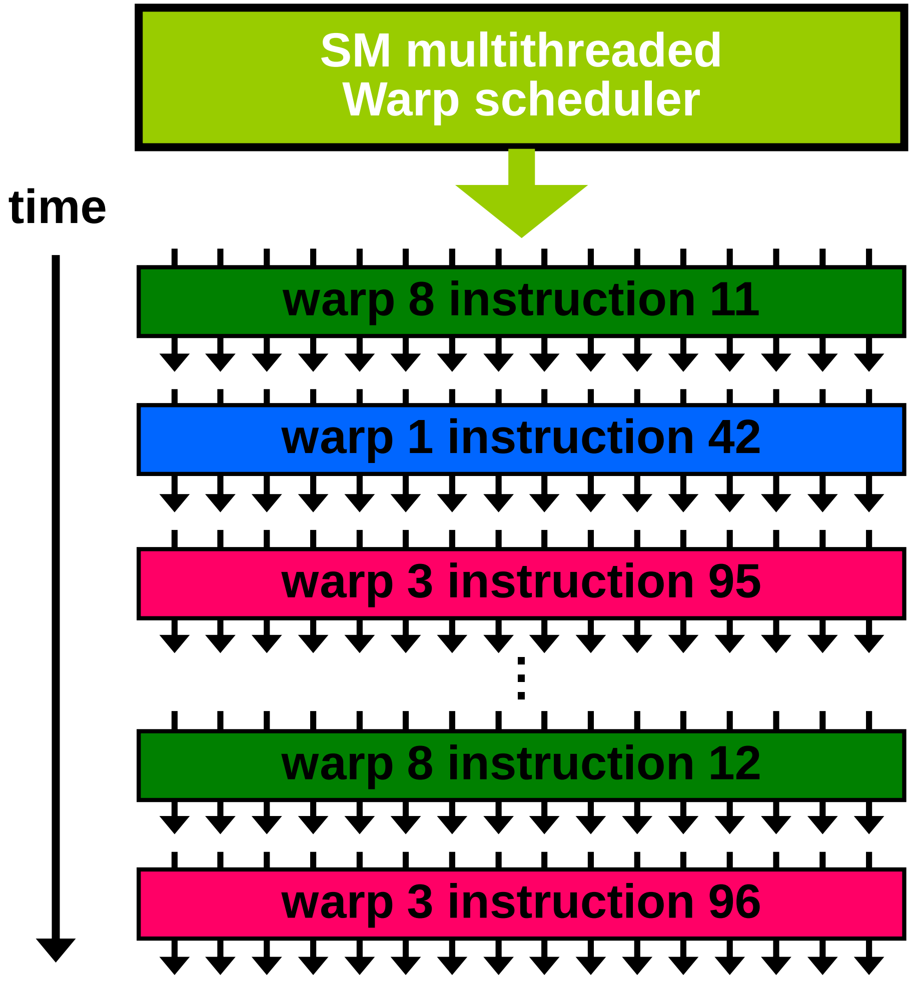
Thread Scheduling and Execution

- Each Thread Blocks is divided in 32-thread Warps
  - This is an implementation decision, not part of the CUDA programming model
- Warp: primitive scheduling unit
- All threads in warp:
  - same instruction
  - control flow causes some to become inactive





Warp Scheduling



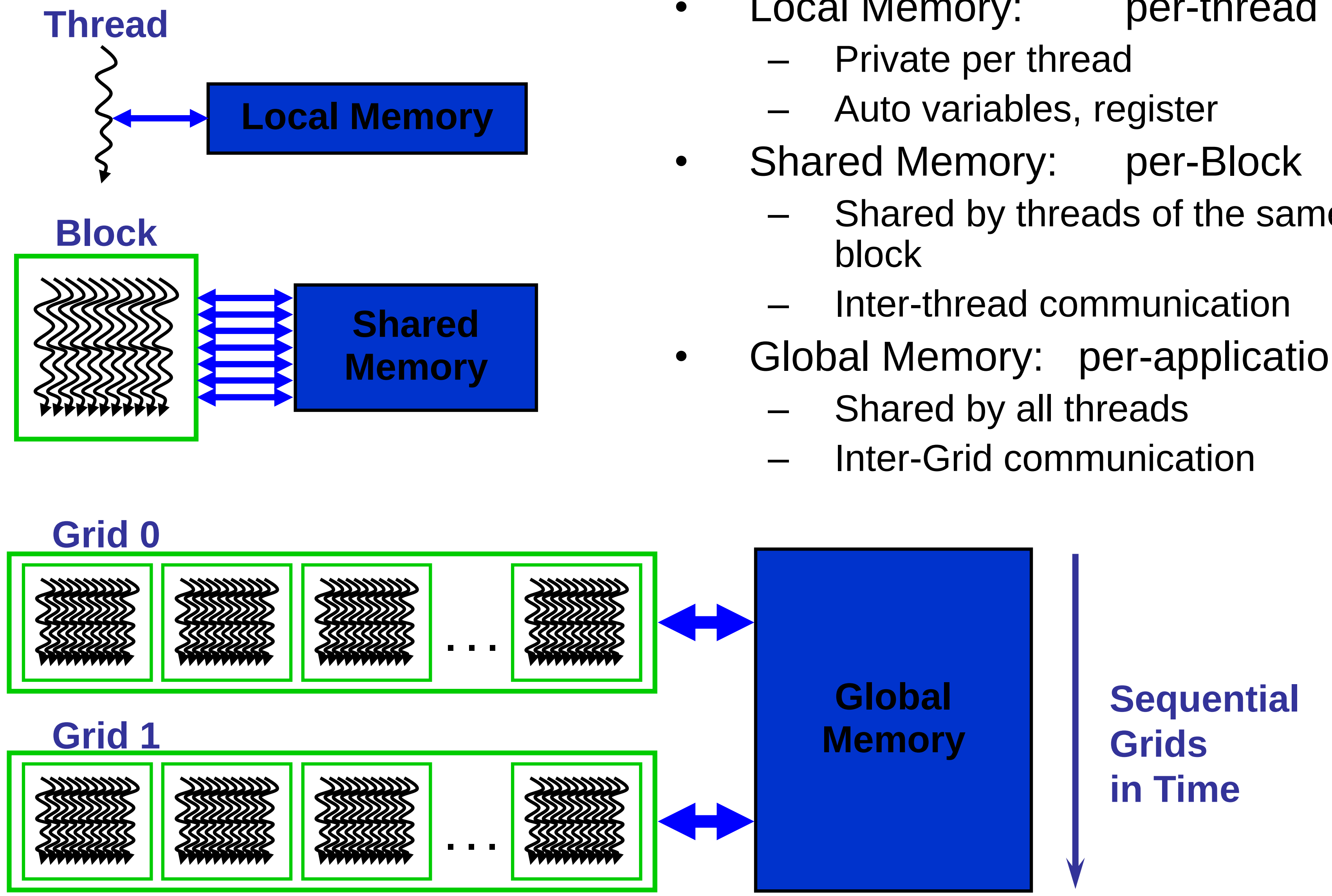
- SM hardware implements zero-overhead Warp scheduling
  - Warps whose next instruction has its operands ready for consumption are eligible for execution
  - Eligible Warps are selected for execution on a prioritized scheduling policy
  - All threads in a Warp execute the same instruction when selected

Memory System Goals

- High-Bandwidth (730 GB/s, 900 GB/s)
- As much parallelism as possible
- wide. 512 pins in G200 / Many DRAM chips
- max data rate per pin.
- maximize utilization
  - Multiple bins of memory requests
  - Coalesce requests to get as wide as possible
  - Goal to use every cycle to transfer from/to memory

- Caches where it makes sense. Small (L1, L2)

Parallelism in the Memory System



- Local Memory: per-thread
  - Private per thread
  - Auto variables, register
- Shared Memory: per-Block
  - Shared by threads of the same block
  - Inter-thread communication
- Global Memory: per-application
  - Shared by all threads
  - Inter-Grid communication

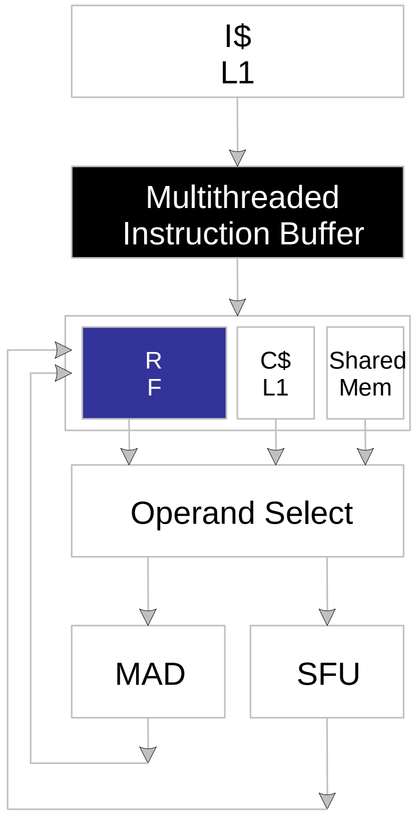
SM Memory Architecture

- Threads in a Block share data & results
  - In Memory and Shared Memory
  - Synchronize at barrier instruction

- Per-Block Shared Memory Allocation
  - Keeps data close to processor
  - Minimize trips to global Memory
  - SM Shared Memory dynamically allocated to Blocks, one of the limiting resources

SM Register File

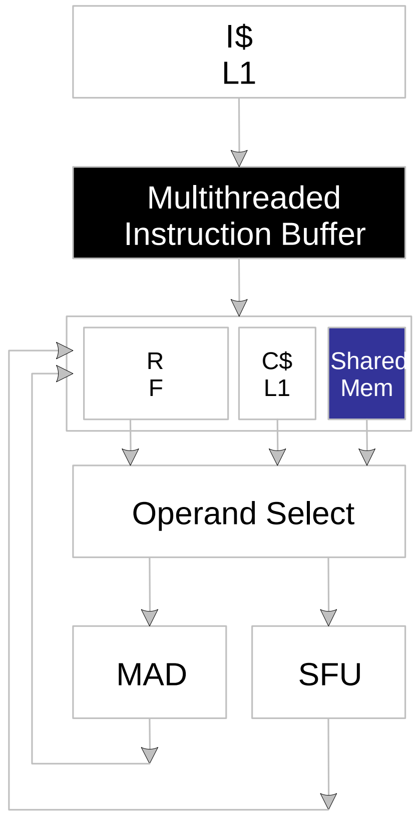
- Register File (RF)
  - 64 KB (depending on arch)
  - 16K 32-bit registers
  - Provides 4 operands/clock
- TEX pipe can also read/write RF
  - 3 SMs share 1 TEX
- Load/Store pipe can also read/write RF



Shared Memory

- Each SM has 64 (?) KB of Shared Memory
  - 16 banks of 32bit words
- CUDA uses Shared Memory as shared storage visible to all threads in a thread block
  - read and write access

- Key Performance Enhancement
- Move data in Shared memory
- Operate in there

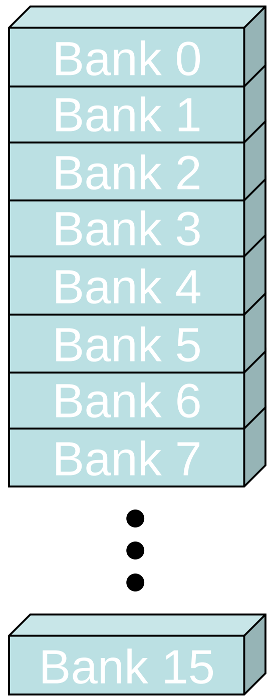


Parallel Memory Architecture

- In a parallel machine, many threads access memory
  - Therefore, memory is divided into banks
  - Essential to achieve high bandwidth

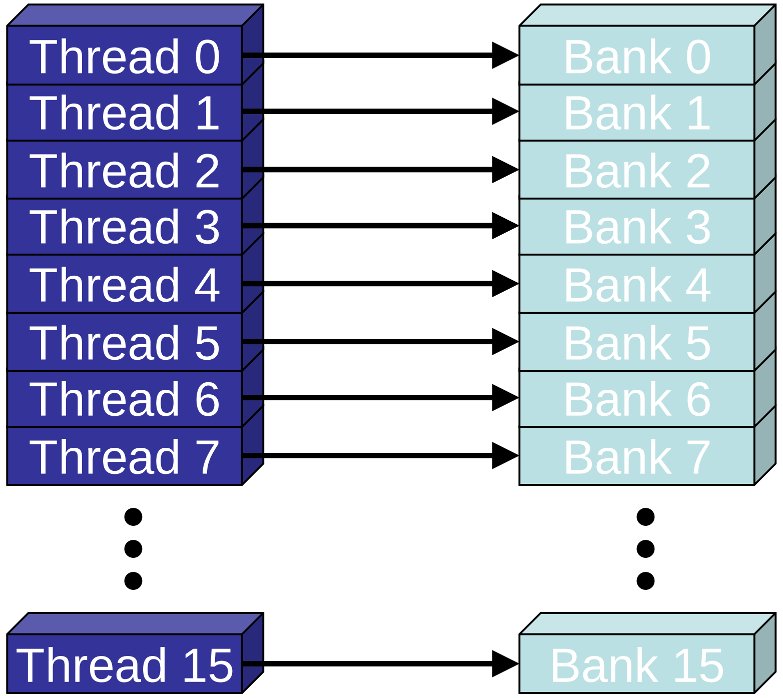
- Each bank can service one address per cycle
  - A memory can service as many simultaneous accesses as it has banks

- Multiple simultaneous accesses to a bank result in a bank conflict
  - Conflicting accesses are serialized

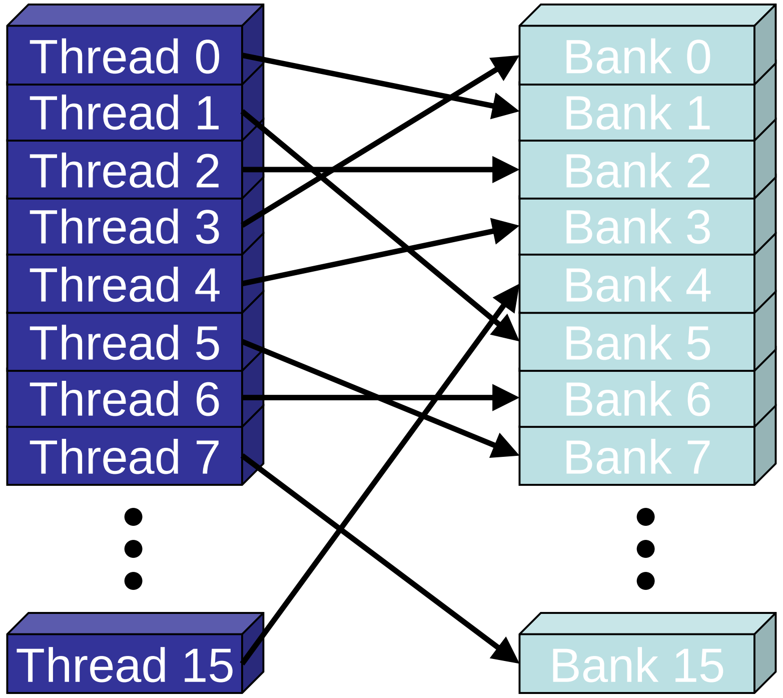


Bank Addressing Examples

- No Bank Conflicts
  - Linear addressing stride == 1

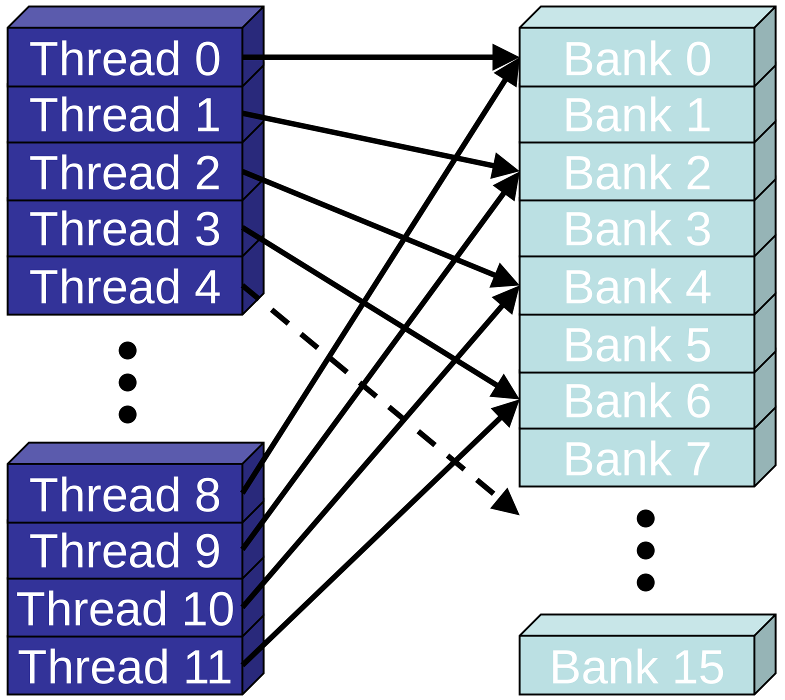


- No Bank Conflicts
  - Random 1:1 Permutation

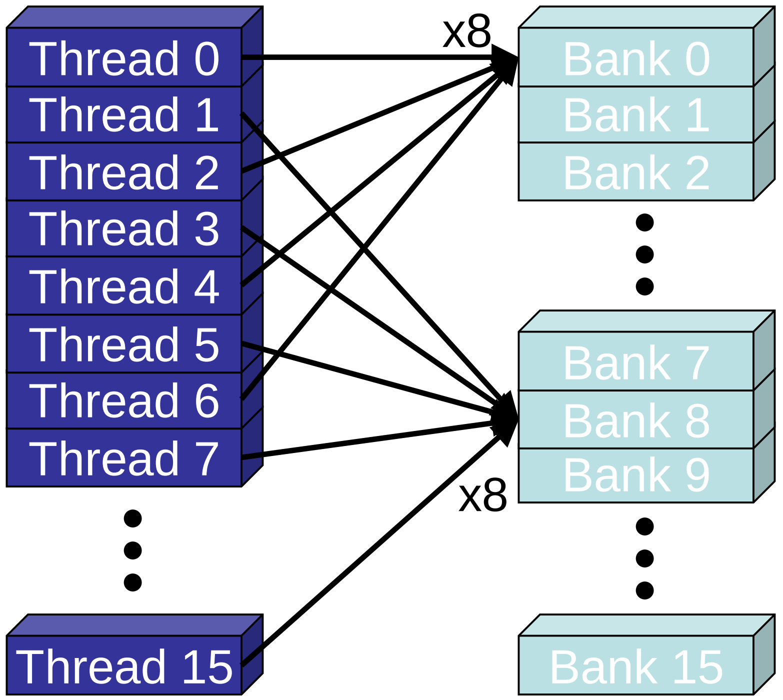


Bank Addressing Examples

- 2-way Bank Conflicts
  - Linear addressing stride == 2



- 8-way Bank Conflicts
  - Linear addressing stride == 8

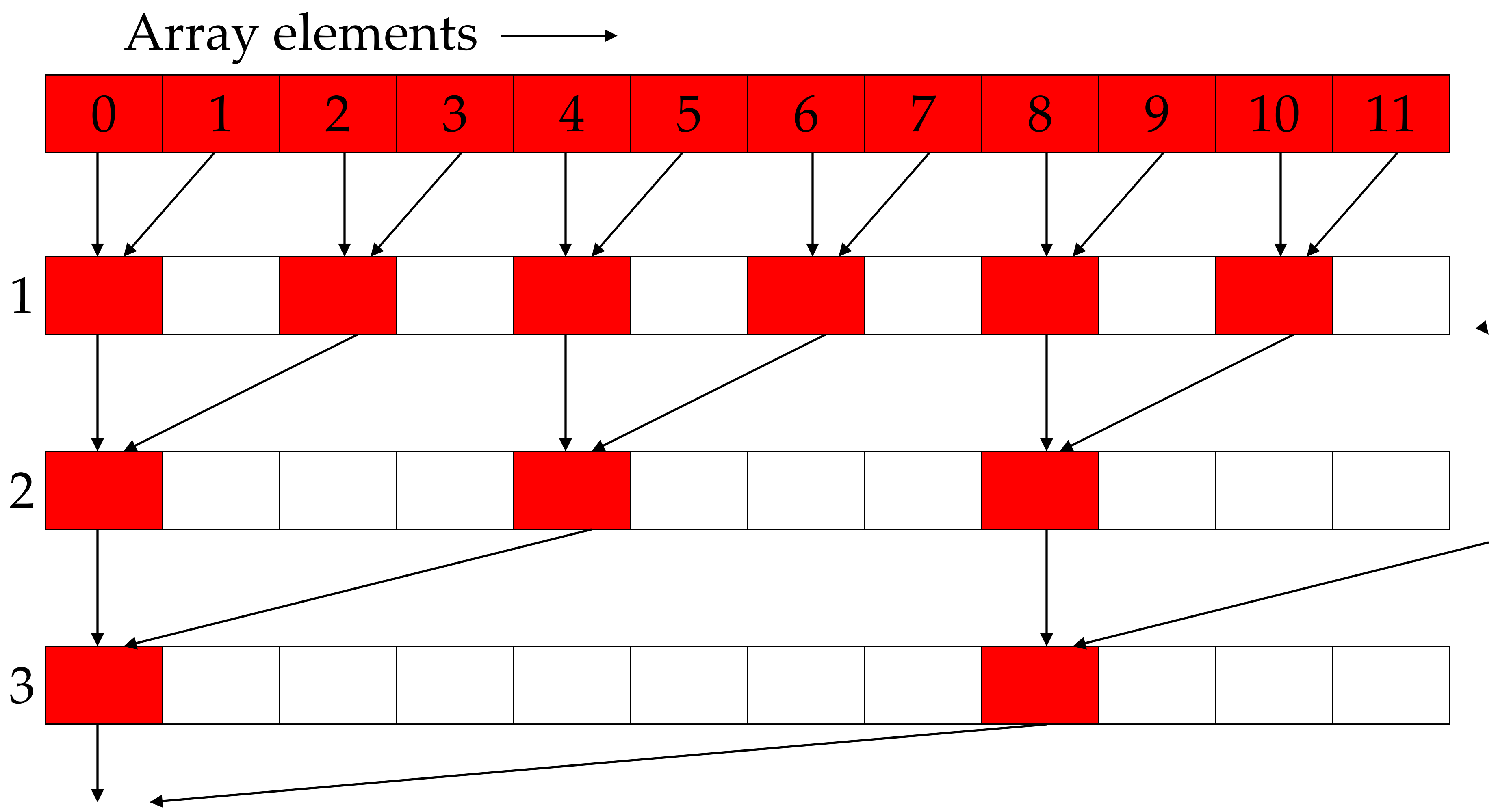


Shared memory bank conflicts

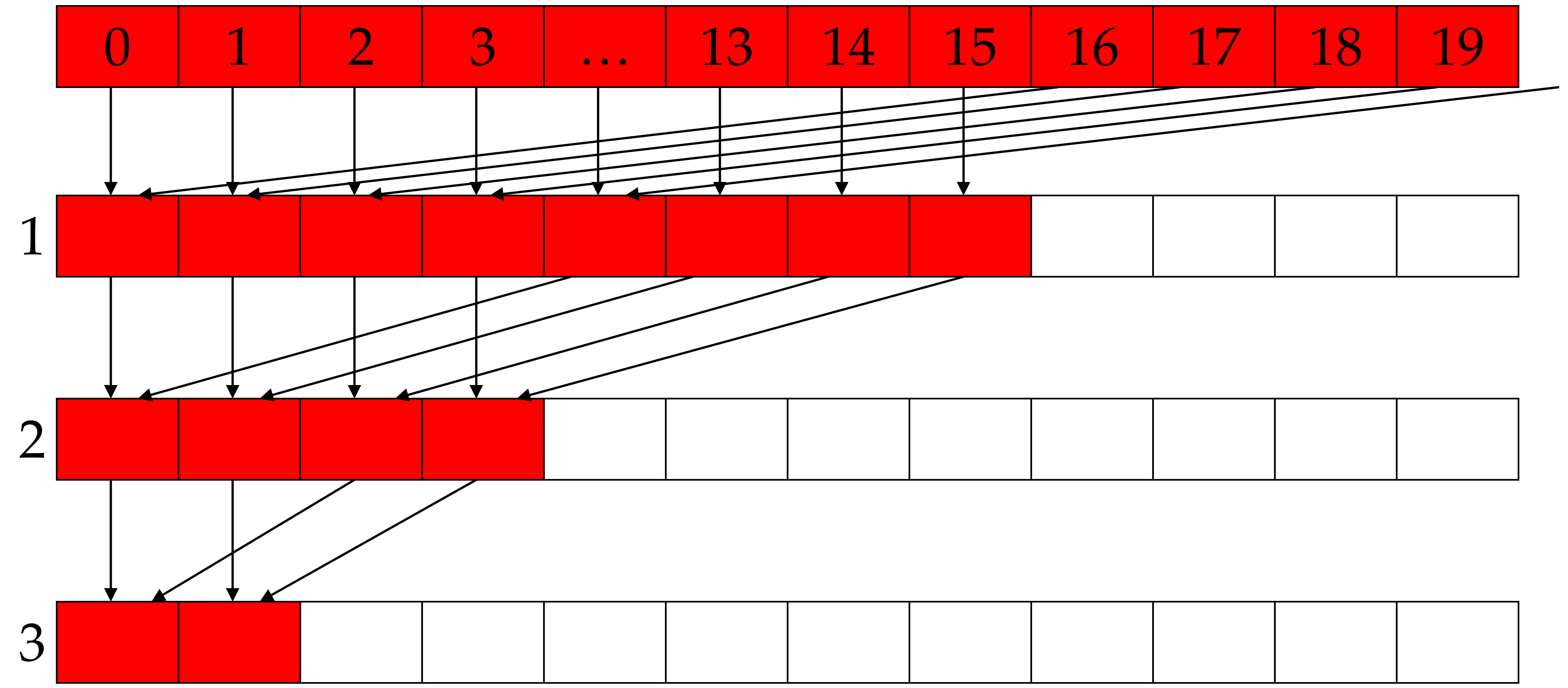
- Shared memory is as fast as registers if there are no bank conflicts

- The fast case:
  - If all threads of a half-warp access different banks, there is no bank conflict
  - If all threads of a half-warp access the identical address, there is no bank conflict (broadcast)
- The slow case:
  - Bank Conflict: multiple threads in the same half-warp access the same bank
  - Must serialize the accesses (or broadcast)
  - Cost = max # of simultaneous accesses to a single bank

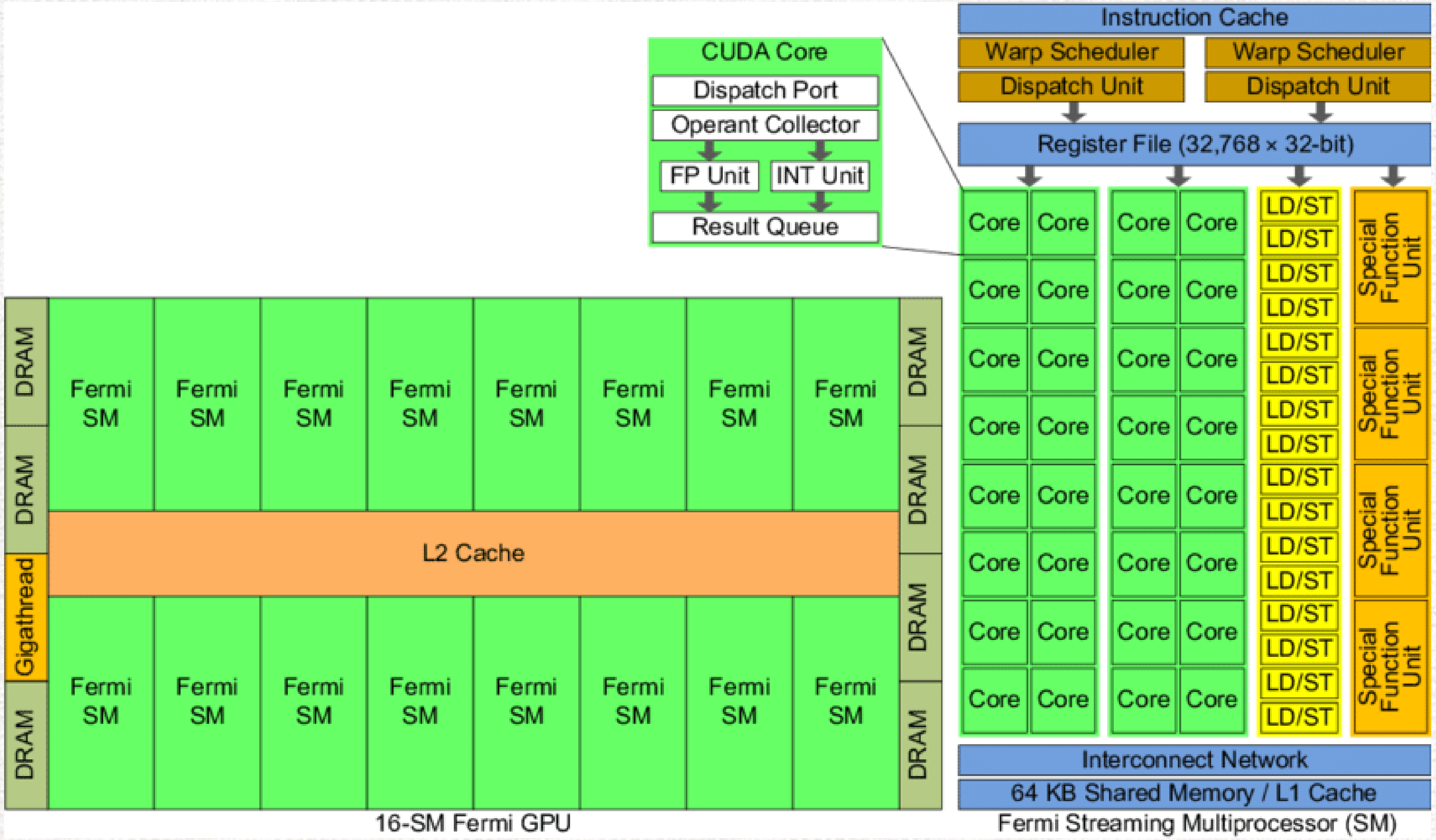
Vector Reduction with Bank Conflicts



No Bank Conflicts



GPU 架构


CUDA Core
Dispatch Port
Operant Collector
FP Unit
INT Unit
Result Queue
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 × 32-bit)
Core
LD/ST
Special Function Unit
Interconnect Network
64 KB Shared Memory / L1 Cache
Fermi Streaming Multiprocessor (SM)
DRAM
Gigathread
Fermi SM
L2 Cache
16-SM Fermi GPU

THANK YOU