EE382V (17325): Principles in Computer Architecture
Parallelism and Locality
Fall 2007

Lecture 12 – GPU Architecture (NVIDIA G80)

Mattan Erez

The University of Texas at Austin

Outline

- 3D graphics recap and architecture motivation
- Overview and definitions
- Execution core architecture
- Memory architecture
  - Moved to lecture 13

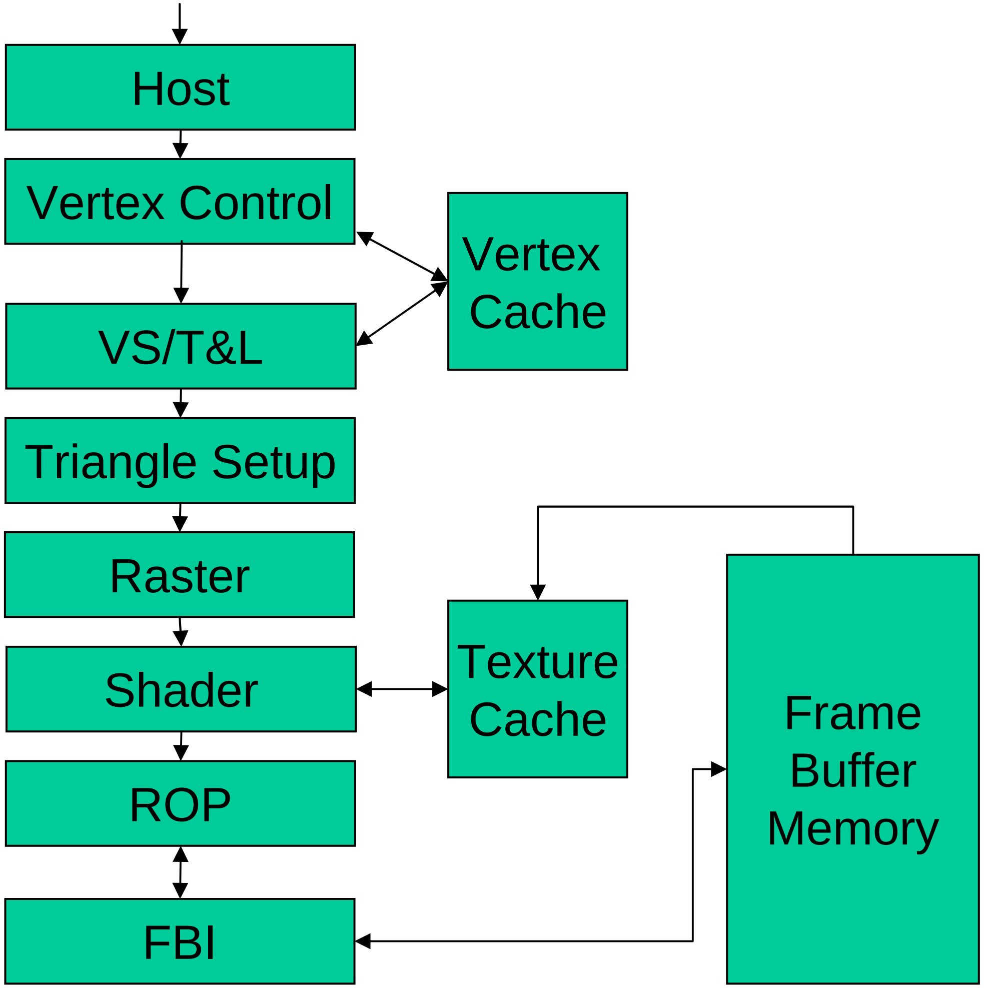
- Most slides courtesy David Kirk (NVIDIA) and Wen-Mei Hwu (UIUC)
  - From The University of Illinois ECE 498AI class
- A few slides courtesy David Luebke (NVIDIA)

A GPU Renders 3D Scenes

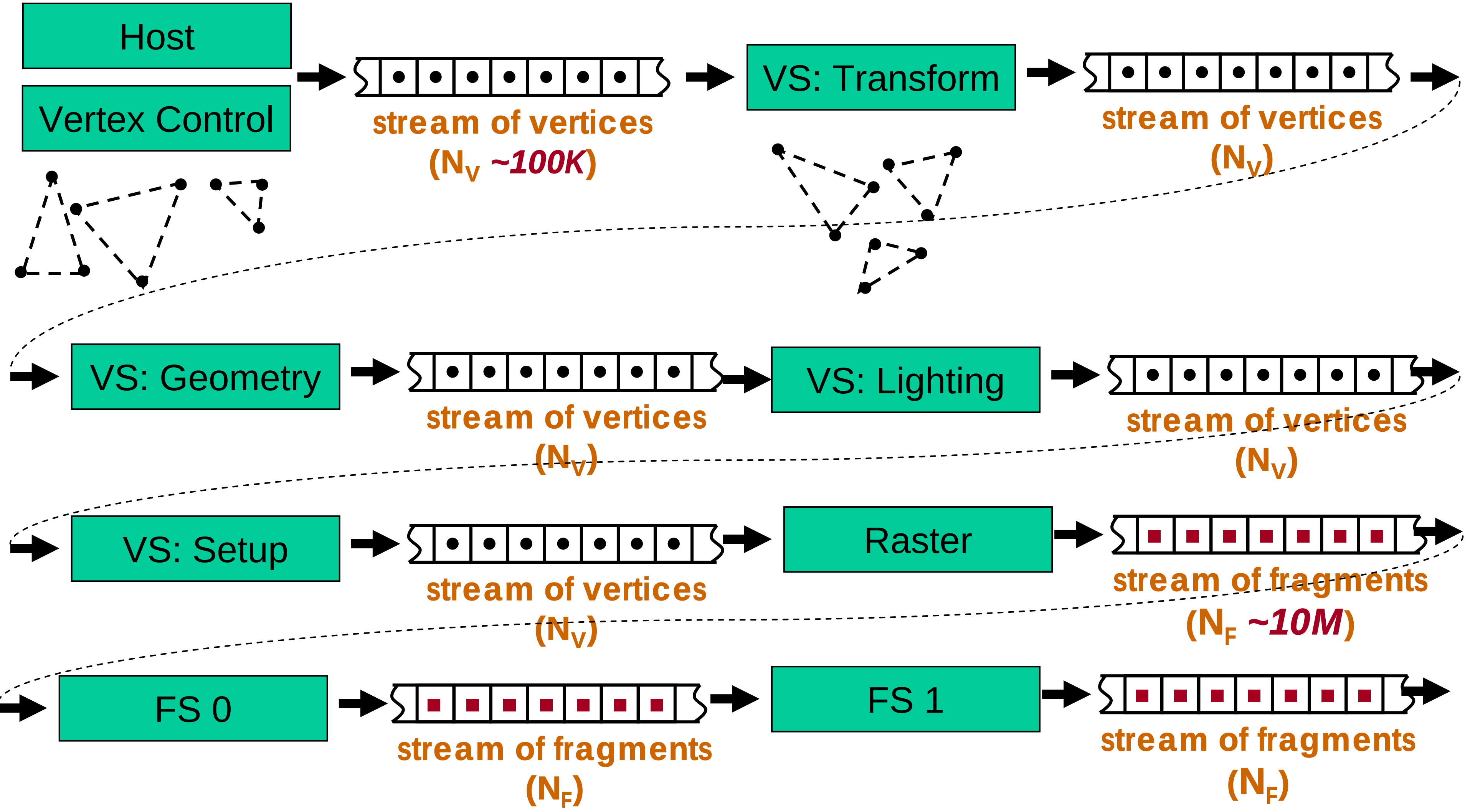
- A ***Graphics Processing Unit (GPU)*** accelerates rendering of 3D scenes
  - Input: description of scene
  - Output: colored pixels to be displayed on a screen
- Input:
  - Geometry (triangles), colors, lights, effects, textures
- Output:

Flight Simulator X

The NVIDIA GeForce Graphics Pipeline



Another View of the 3D Graphics Pipeline



Host

Vertex Control

stream of vertices (N_V *~100K*)

VS: Transform

stream of vertices (N_V)

VS: Geometry

stream of vertices (N_V)

VS: Lighting

stream of vertices (N_V)

VS: Setup

stream of vertices (N_V)

Raster

stream of fragments (N_F *~10M*)

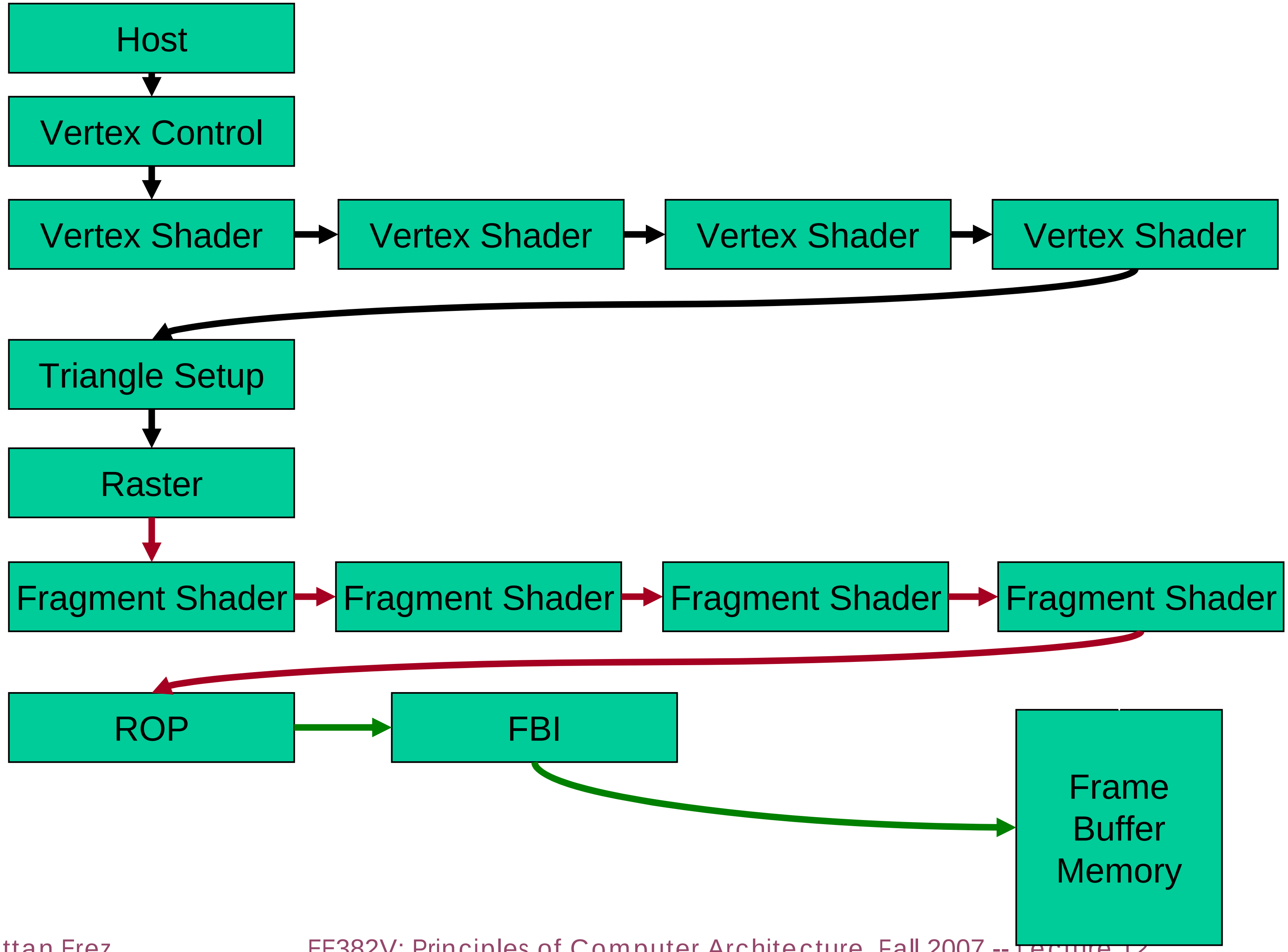
FS 0

stream of fragments (N_F)

FS 1

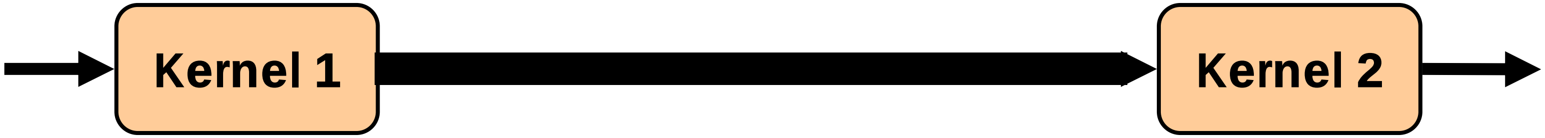
stream of fragments (N_F)

The NVIDIA GeForce Graphics Pipeline



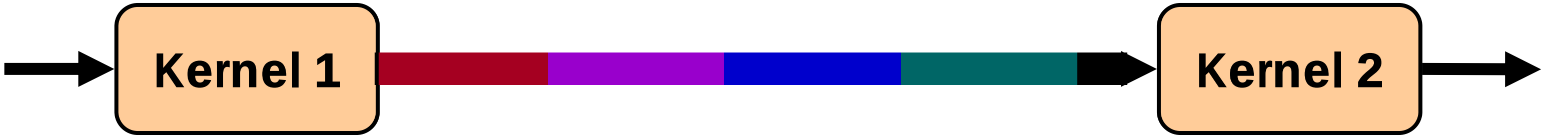
Stream Execution Model

- Data parallel ***streams*** of data
- Processing ***kernels***
  - Unit of Execution is processing of one stream element in one kernel – defined as a ***thread***



Stream Execution Model

- Can partition the streams into chunks
  - Streams are very long and elements are independent
  - Chunks are called ***strips*** or ***blocks***



- Unit of Execution is processing one block of data by one kernel – defined as a ***thread block***

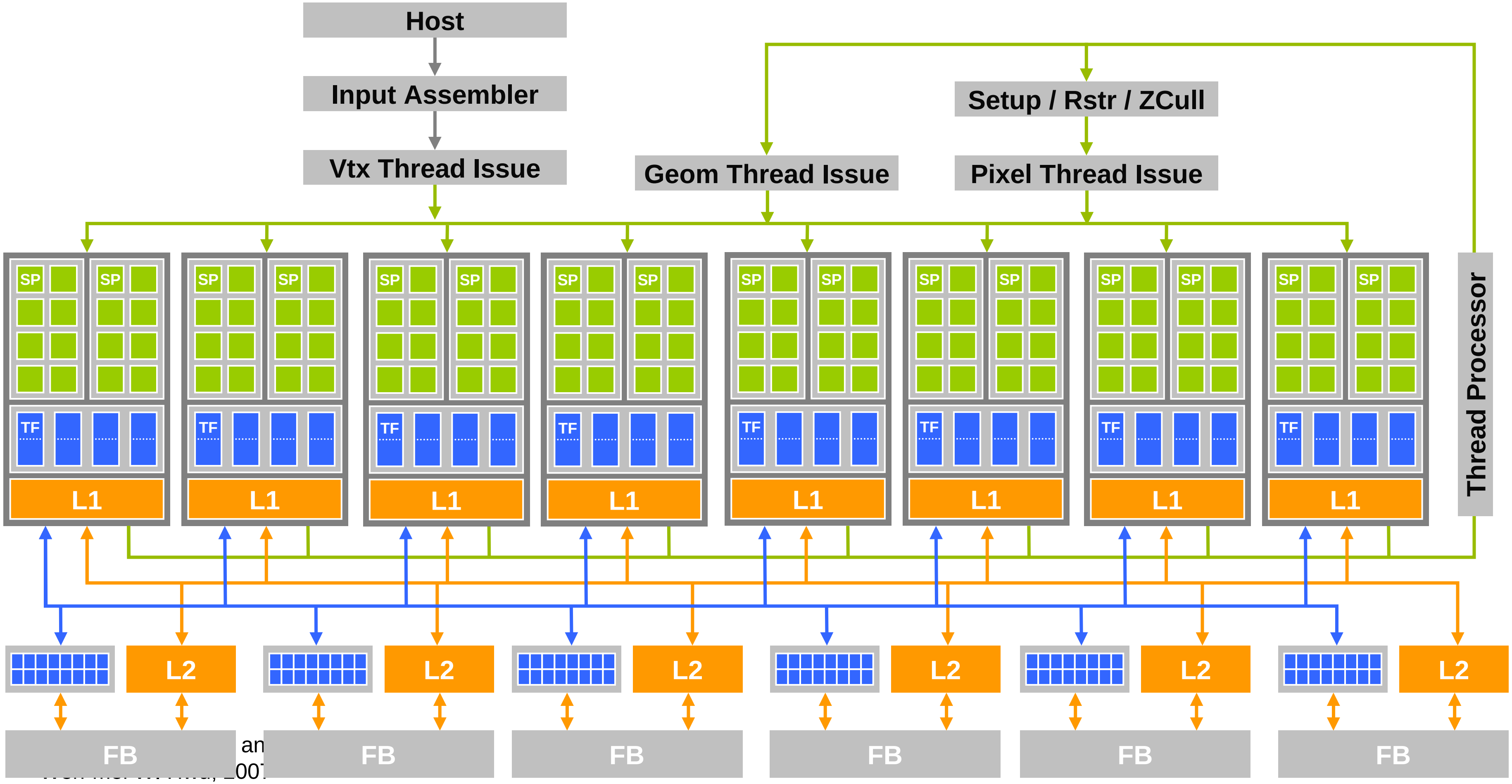
Outline

- 3D graphics recap and architecture motivation
- Overview and definitions
- Execution core architecture
- Memory architecture

- Most slides courtesy David Kirk (NVIDIA) and Wen-Mei Hwu (UIUC)
  - From The University of Illinois ECE 498AI class

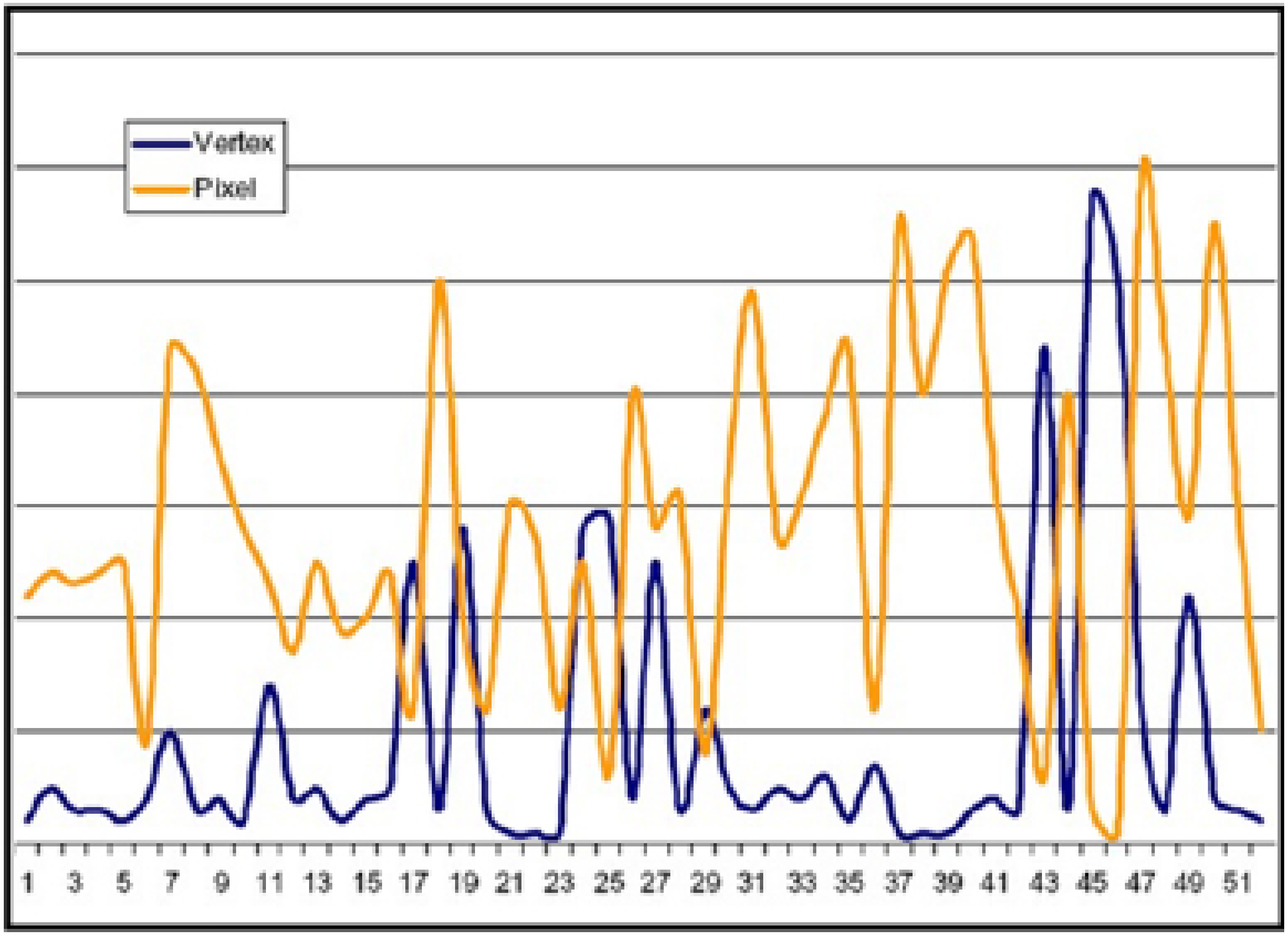
Make the Compute Core The Focus of the Architecture

- The future of GPUs is programmable processing
- So – build the architecture around the processor



an
ECE 498AL, University of Illinois,
Urbana-Champaign

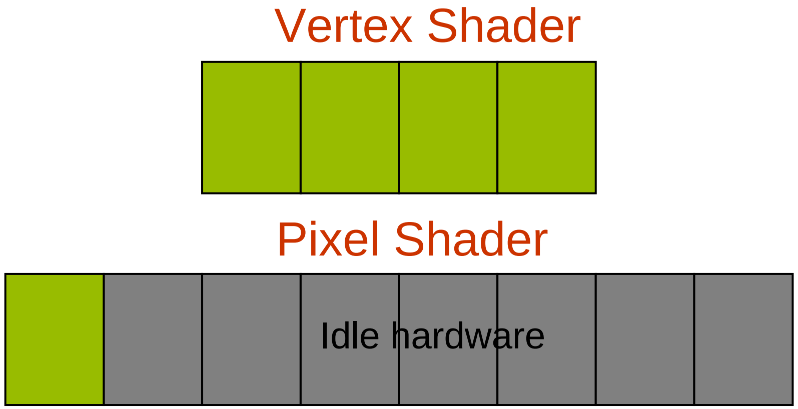
Vertex and Fragment Processing Share Unified Processing Elements



© NVIDIA Corp., 2007

Vertex and Fragment Processing Share Unified Processing Elements

- Load balancing HW is a problem



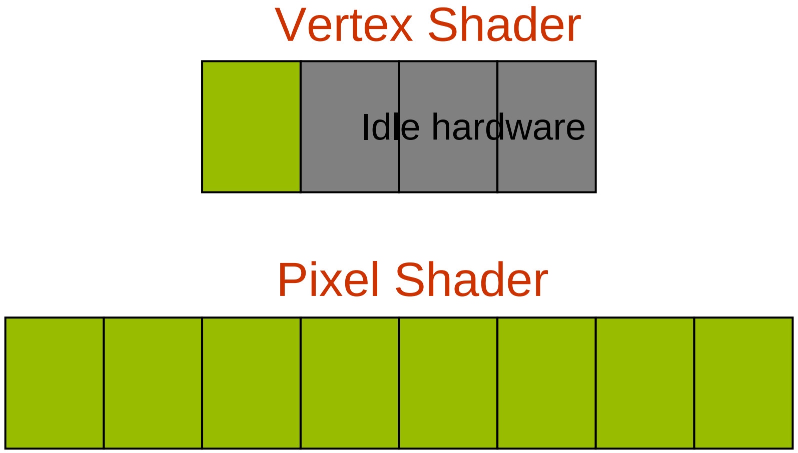
Vertex Shader

Pixel Shader

Idle hardware

Heavy Geometry

Workload Perf = 4



Vertex Shader

Idle hardware

Pixel Shader

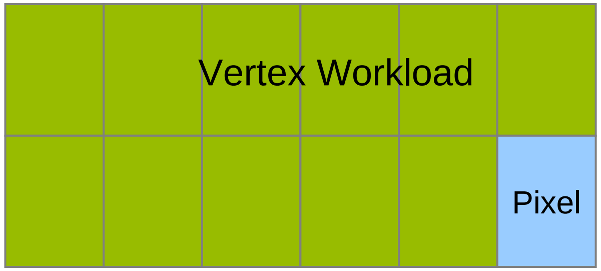
Heavy Pixel

Workload Perf = 8

Vertex and Fragment Processing Share Unified Processing Elements

- Load balancing SW is easier

Unified Shader



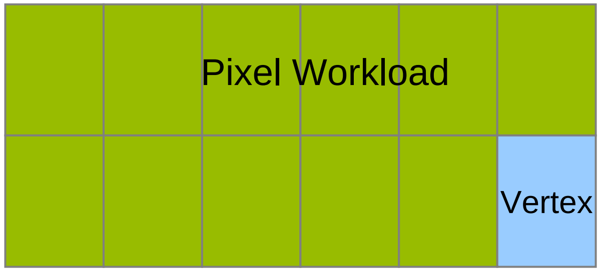
Vertex Workload

Pixel

Heavy Geometry

Workload Perf = 11

Unified Shader



Pixel Workload

Vertex

Heavy Pixel

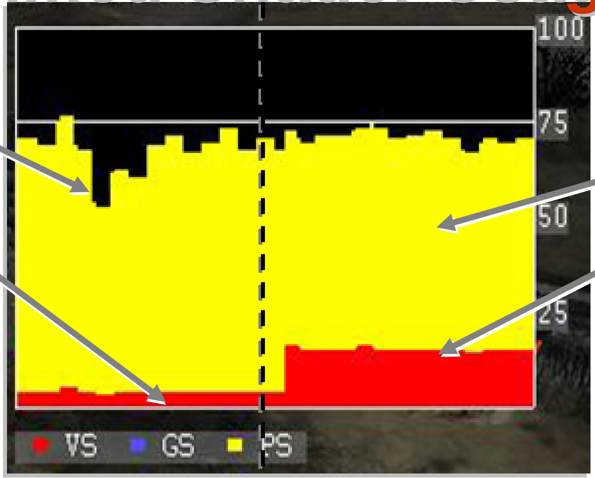
Workload Perf = 11

Vertex and Fragment Processing is Dynamically Load Balanced

Less Geometry

More Geometry

Unified Shader Usage



High pixel shader use

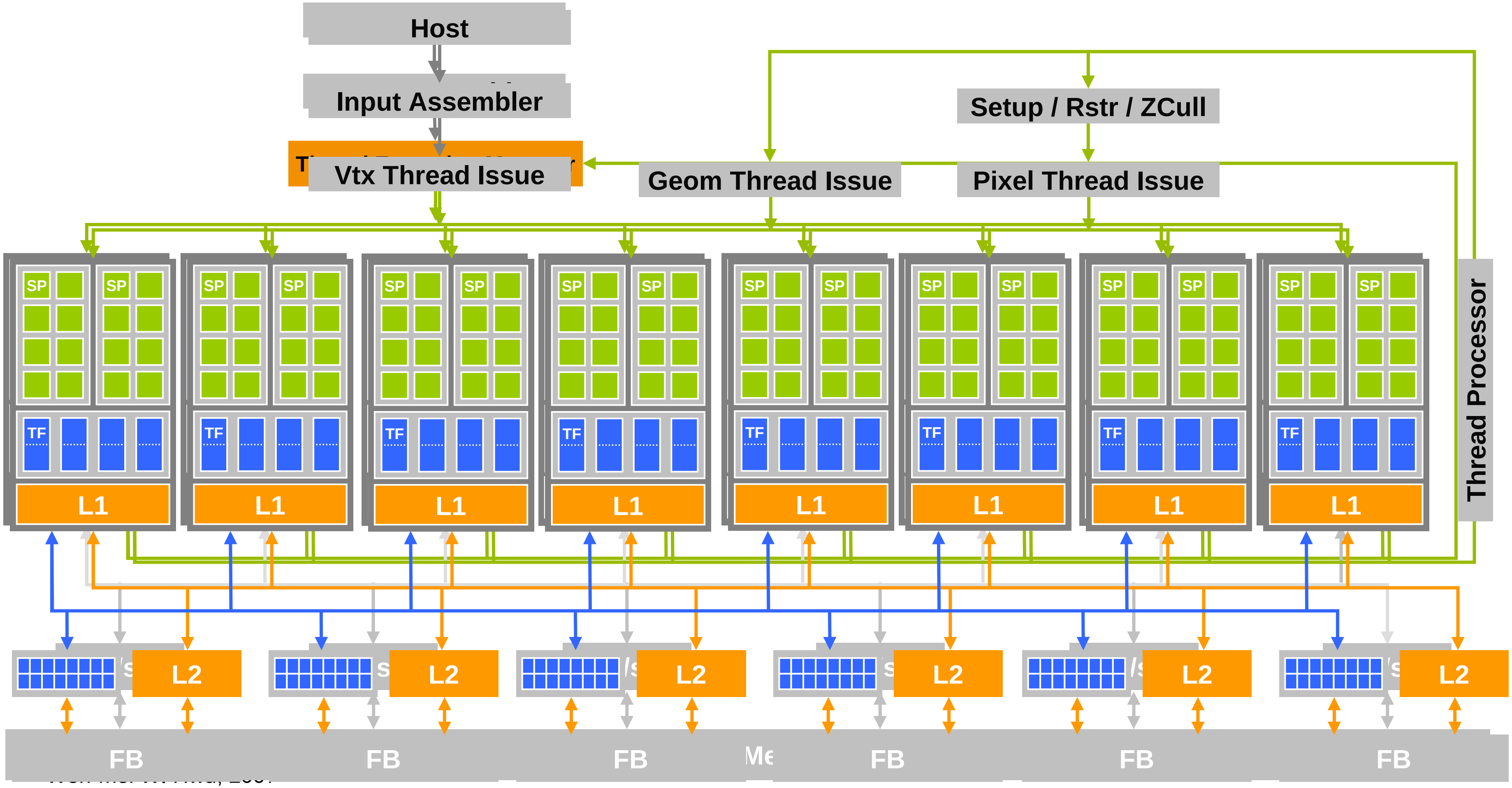
Low vertex shader use

Balanced use of pixel shader and vertex shader

© NVIDIA Corp., 2007

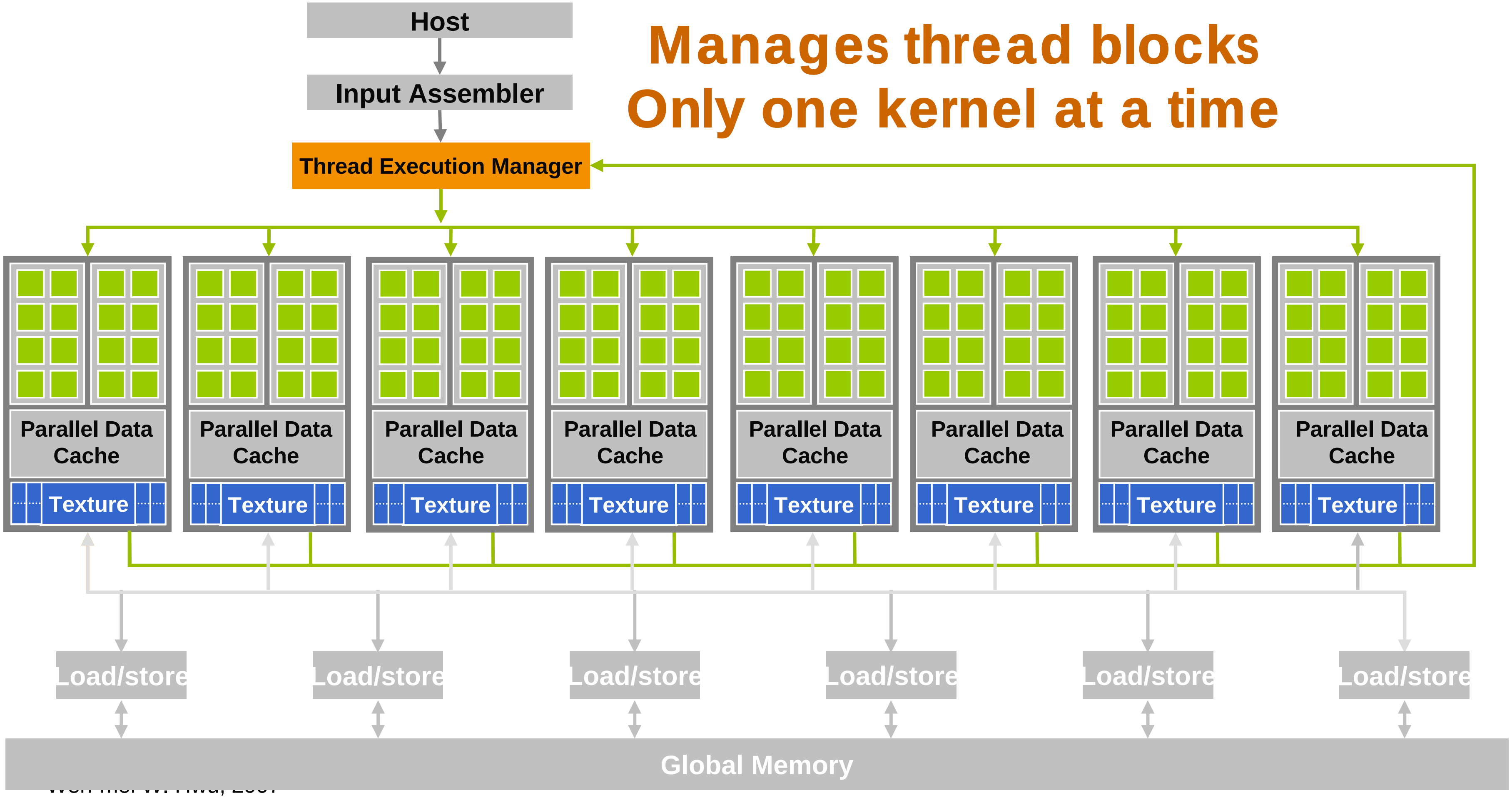
Make the Compute Core The Focus of the Architecture

- The future of GPUs is programmable processing
- Processors execute computing threads
- So – build the architecture around the processor
- Alternative operating mode specifically for computing



Make the Compute Core The Focus of the Architecture

- Processors execute computing threads
- Alternative operating mode specifically for computing



Manages thread blocks

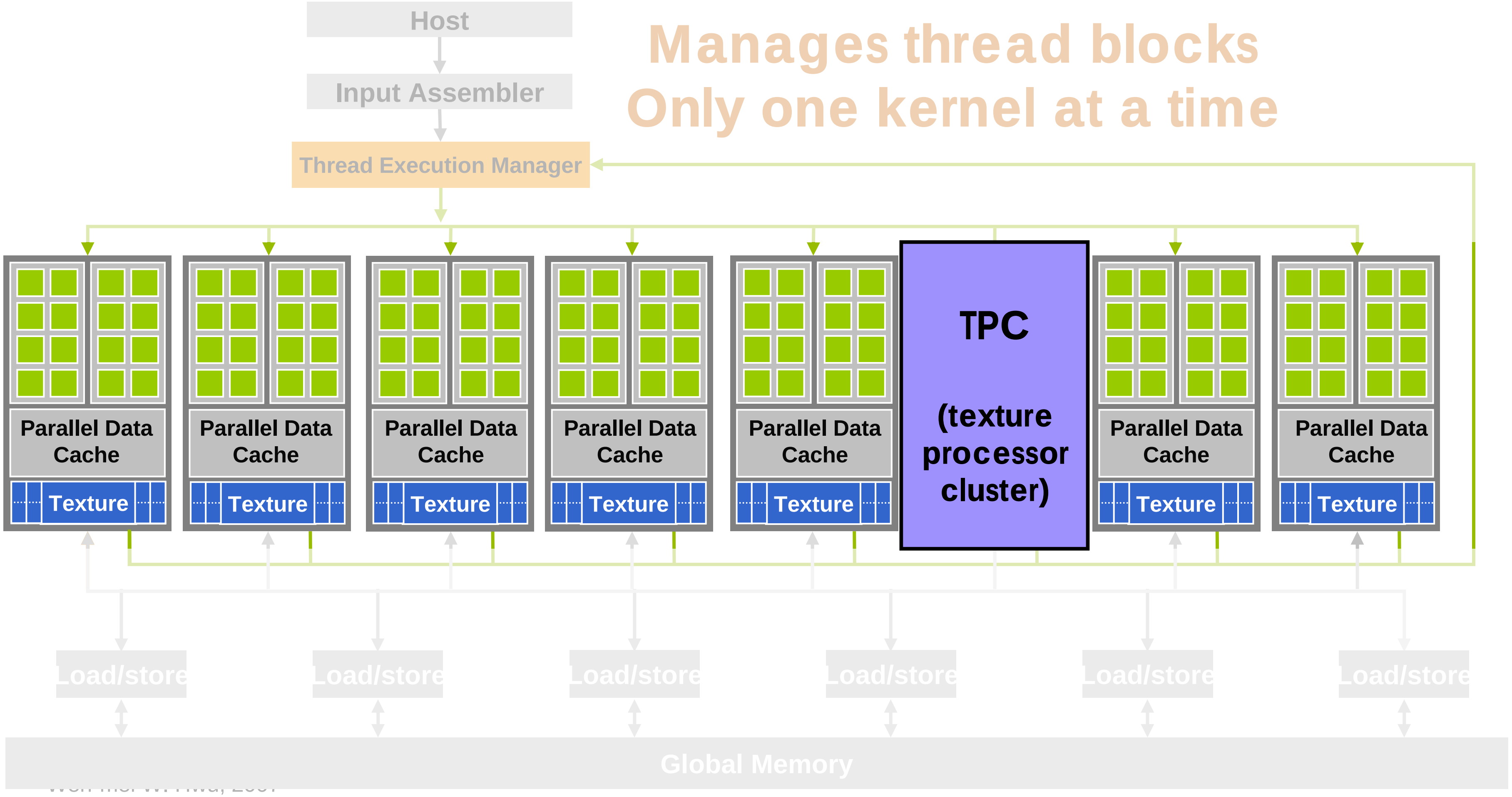
Only one kernel at a time

Outline

- 3D graphics recap and architecture motivation
- Overview and definitions
- Execution core architecture
- Memory architecture

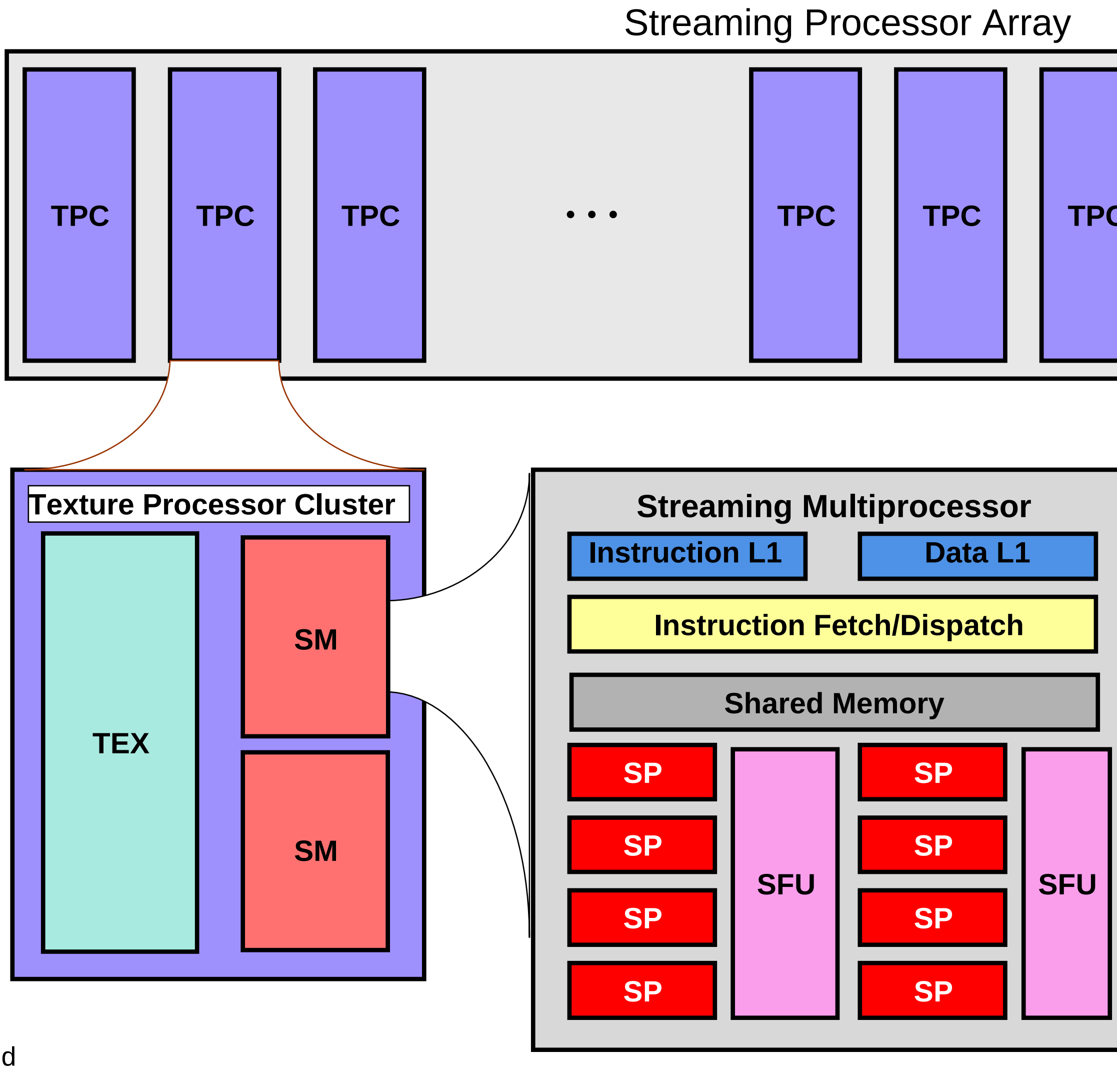
- Most slides courtesy David Kirk (NVIDIA) and Wen-Mei Hwu (UIUC)
  - From The University of Illinois ECE 498AI class

Compute Core



Manages thread blocks
Only one kernel at a time

GeForce-8 Series HW Overview

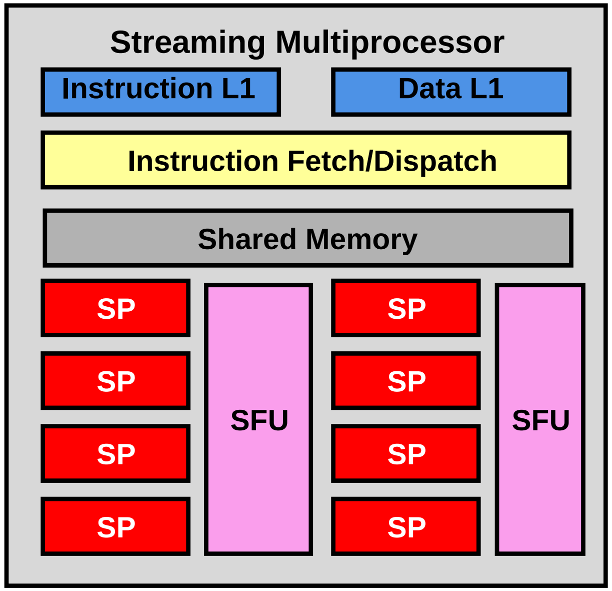


CUDA Processor Terminology

- **SPA** – Streaming Processor Array
  - Array of TPCs
    - 8 TPCs in GeForce8800
- **TPC** – Texture Processor Cluster
  - Cluster of 2 SMs + 1 TEX
    - TEX is a texture processing unit
- **SM** – Streaming Multiprocessor
  - Array of 8 SPs
  - Multi-threaded processor core
  - Fundamental processing unit for a thread block
- **SP** – Streaming Processor
  - Scalar ALU for a single thread
    - With 1K of registers

Streaming Multiprocessor (SM)

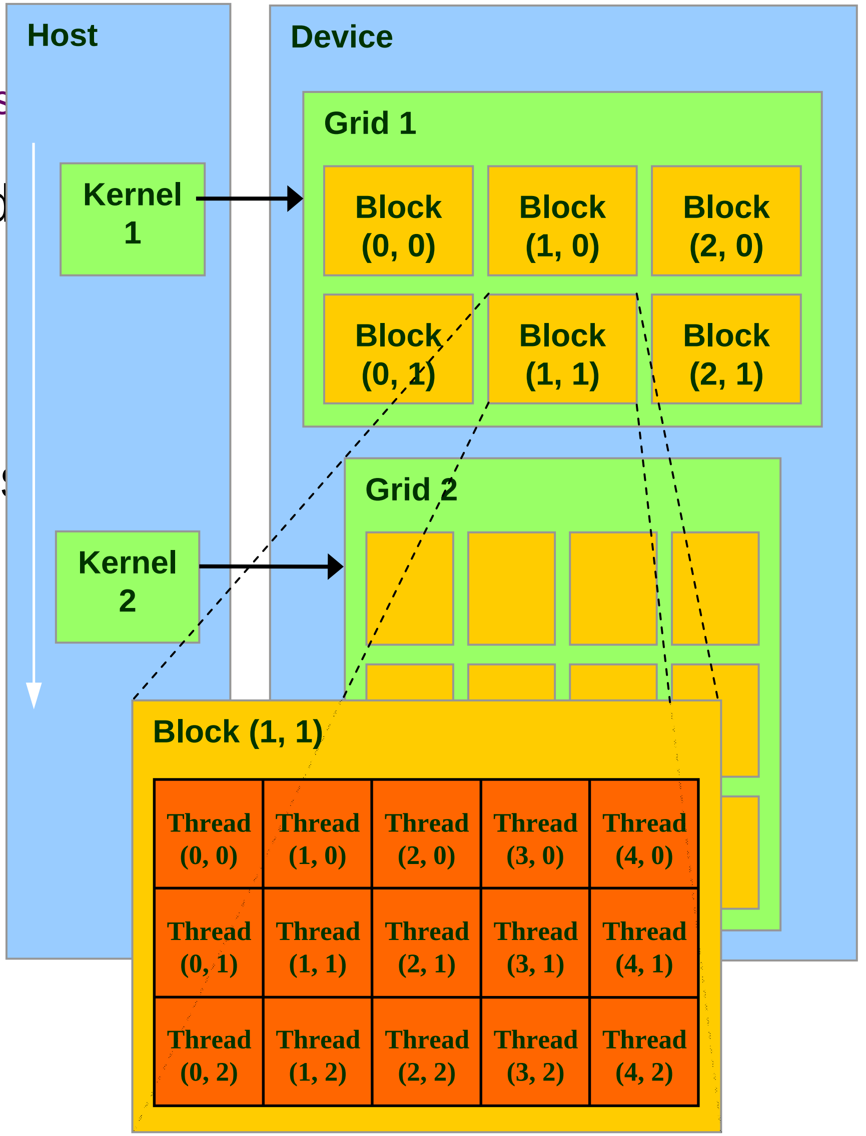
- Streaming Multiprocessor (SM)
  - 8 Streaming Processors (SP)
  - 2 Super Function Units (SFU)
- Multi-threaded instruction dispatch
  - Vectors of 32 threads (***warps***)
  - Up to 16 warps per thread block
    - HW masking of inactive threads in a warp
  - Threads cover latency of texture/memory loads
- 20+ GFLOPS
- 16 KB shared memory
- 32 KB in registers
- DRAM texture and memory access



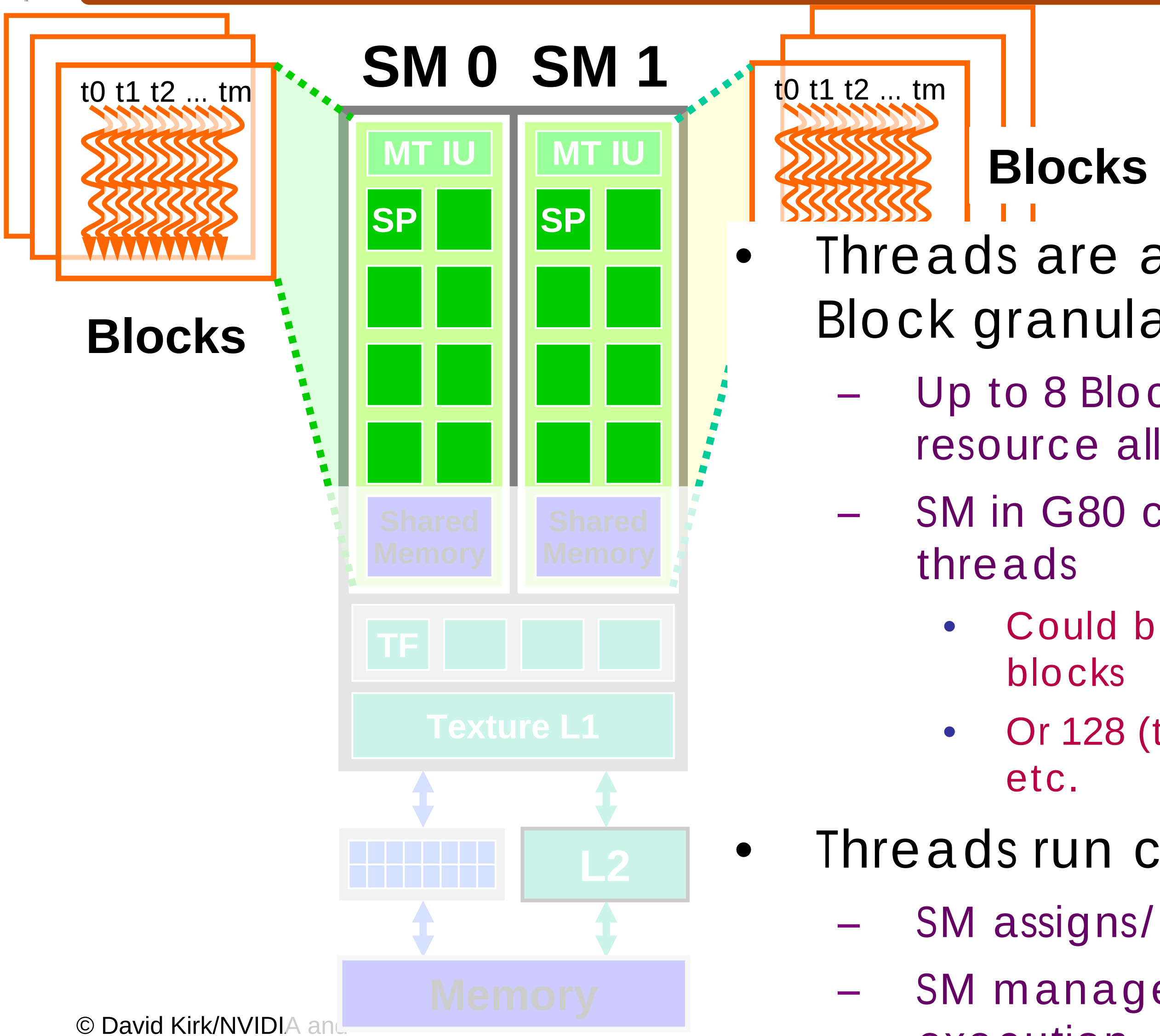
© David Kirk/NVIDIA and
Wen-mei W. Hwu, 2007
ECE 498AL, University of Illinois,
Urbana-Champaign

Thread Life Cycle in HW

- Kernel is launched on the SPA
  - Kernels known as ***grids*** of thread blocks
- Thread Blocks are serially distributed to all the SM's
  - Potentially >1 Thread Block per SM
  - At least 96 threads per block
- 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



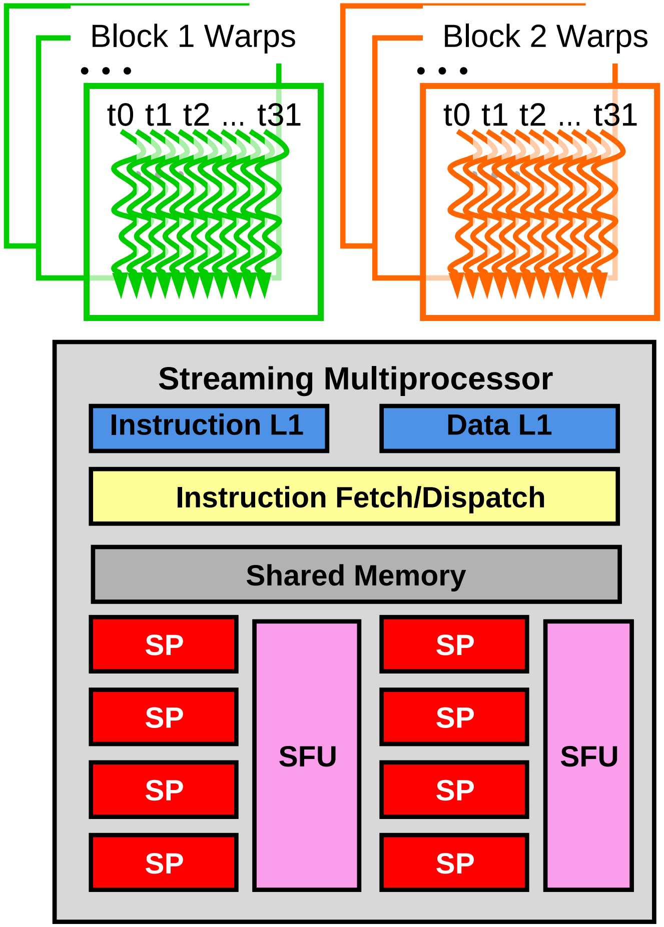
SM Executes Blocks



- 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 (threads/block) * 3 blocks
    - Or 128 (threads/block) * 6 blocks, etc.
- Threads run concurrently
  - SM assigns/maintains thread IDs
  - SM manages/schedules thread execution

Thread Scheduling/Execution

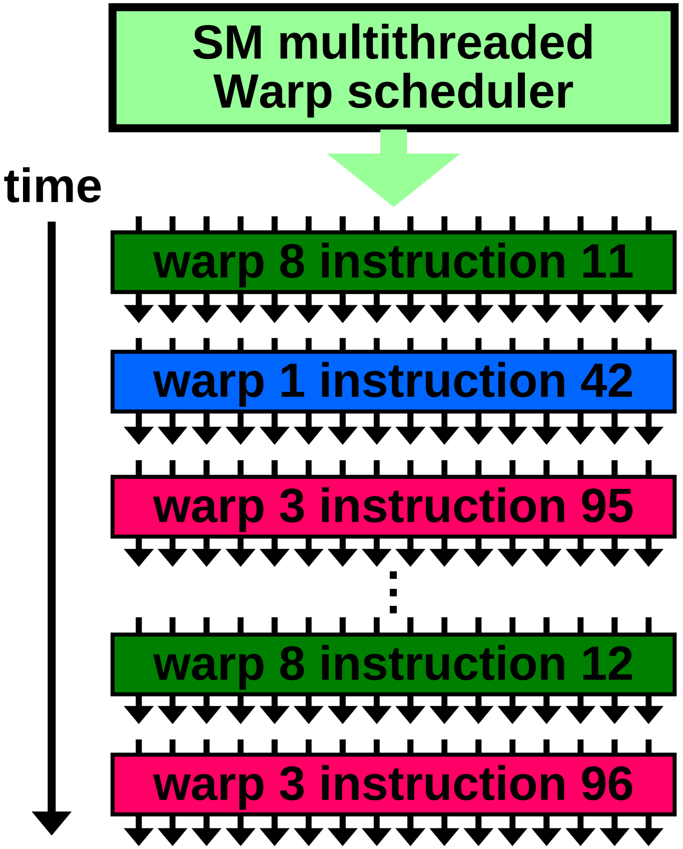
- Each Thread Block is divided into 32-thread Warps
  - This is an implementation decision
- Warps are scheduling units in SM
- If 3 blocks are assigned to an SM and each Block has 256 threads, how many Warps are there in an SM?
  - Each Block is divided into 256/32 = 8 Warps
  - There are 8 * 3 = 24 Warps
  - At any point in time, only one of the 24 Warps will be selected for instruction fetch and execution.



SM Warp Scheduling

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

- 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 latency



SM multithreaded Warp scheduler

time

warp 8 instruction 11

warp 1 instruction 42

warp 3 instruction 95

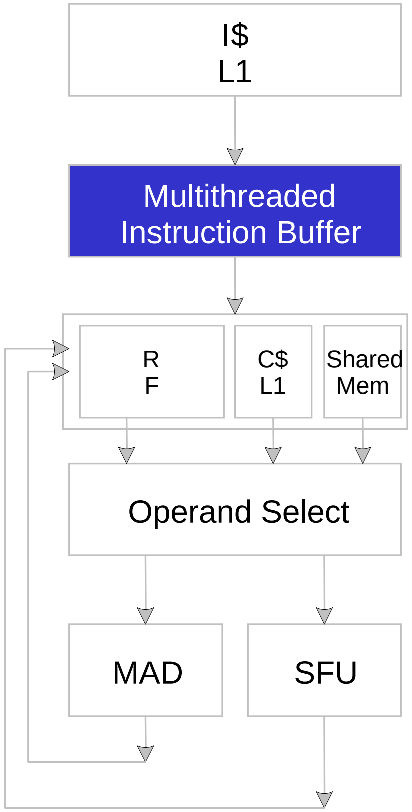
⋮

warp 8 instruction 12

warp 3 instruction 96

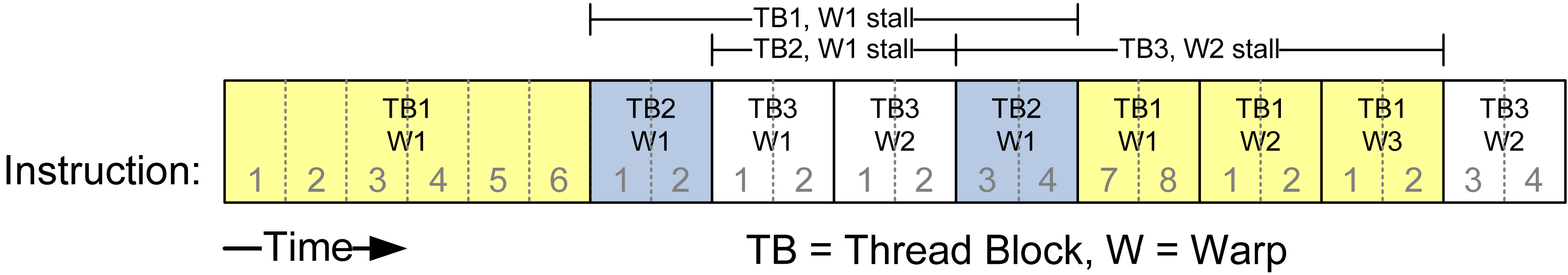
SM Instruction Buffer – Warp Scheduling

- 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
  - Status 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 Memory/Processor ops



Granularity and Resource Considerations

- For Matrix Multiplication, should I use 8X8, 16X16 or 32X32 tiles (1 thread per tile element)?
  - For 8X8, we have 64 threads per Block. Since each SM can take up to 768 threads, it can take up to 12 Blocks. However, each SM can only take up to 8 Blocks, only 512 threads will go into each SM!
  - 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.
  - For 32X32, we have 1024 threads per Block. Not even one can fit into an SM!