CMPSCI 691AD – General Purpose Computation on the GPU

Spring 2009

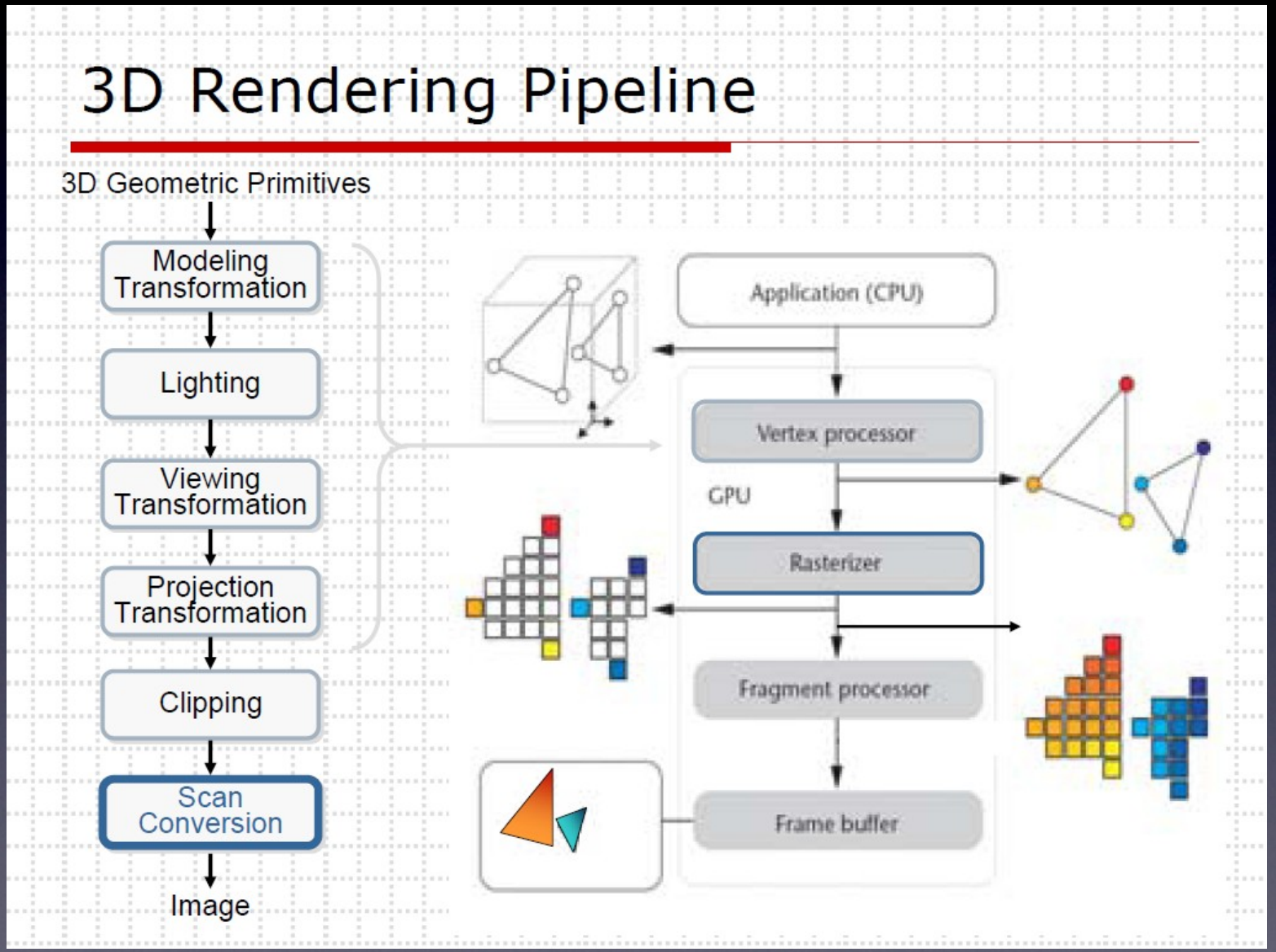
Lecture 12: Graphics Pipeline and Programming Language

Rui Wang

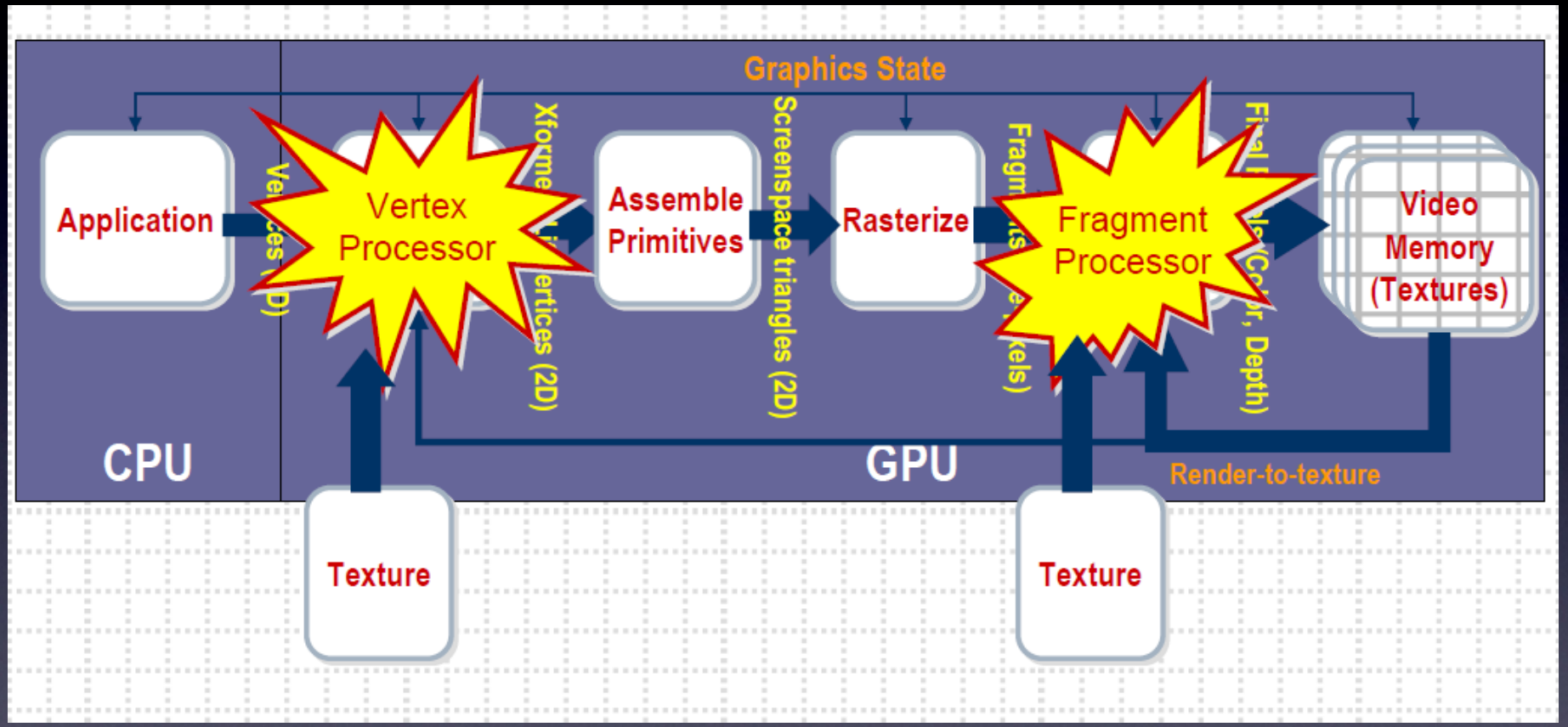
The View of GPU for Graphics Applications

- Rendering: 3D models → 2D images
- GPUs use rasterization-based algorithms
  - Rasterization vs. Ray Tracing

Fixed-Function Graphics Pipeline



Programmable Graphics Pipeline

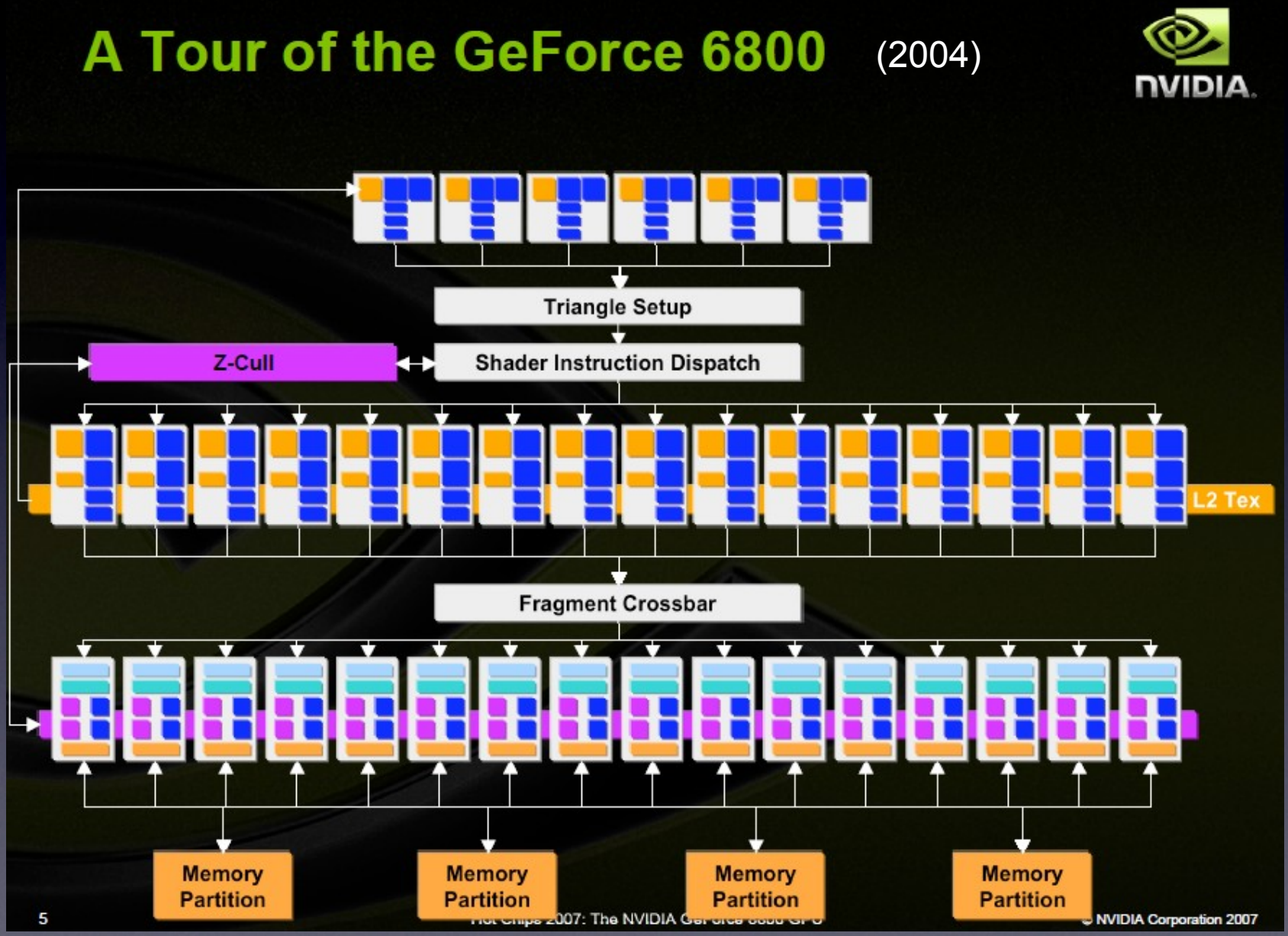


- Load balancing of Vertex vs. Fragment processor?
  - Compare the amount of work of both

Programming Graphics Pipeline

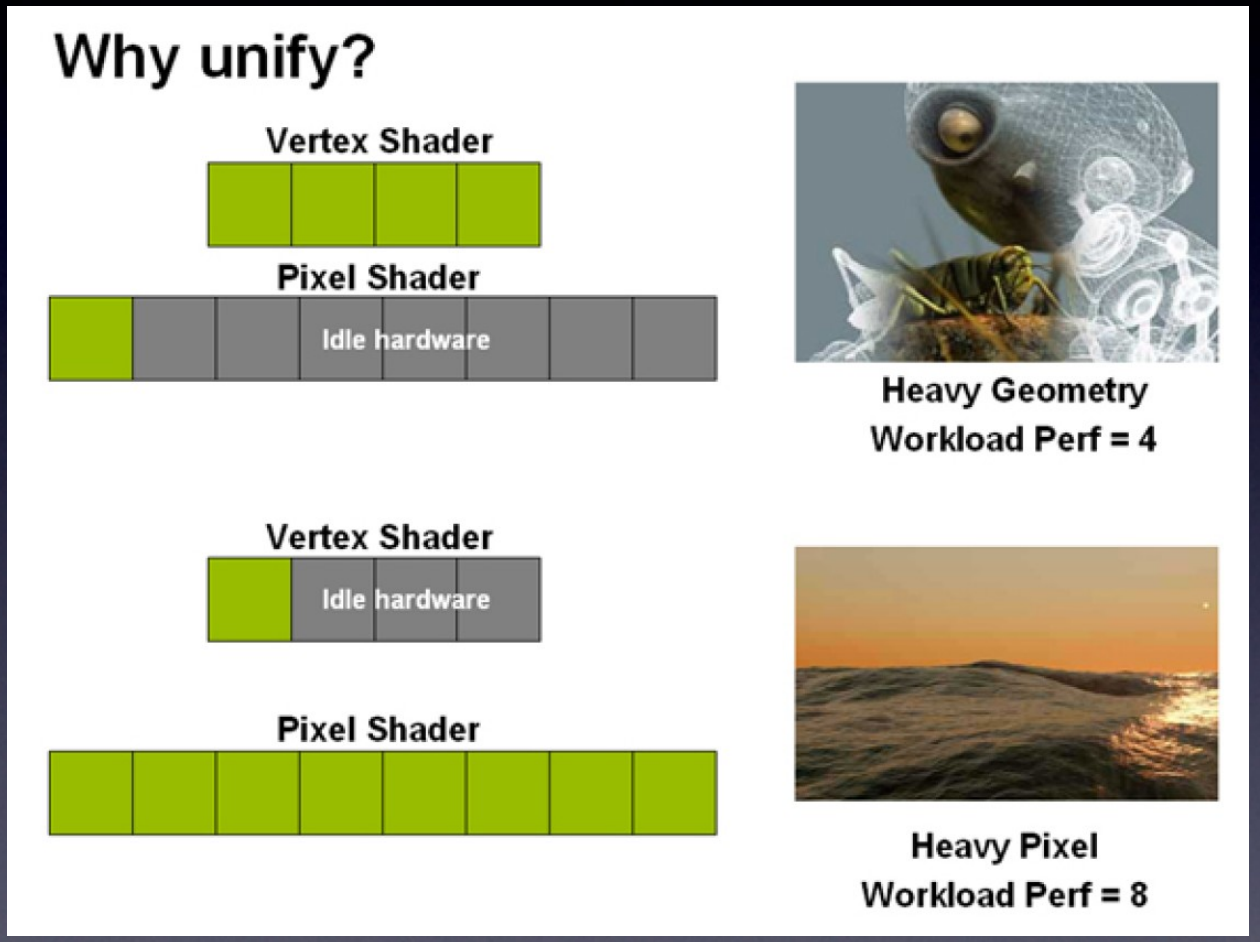
- For a very long time, GPU architecture design is built around this standard graphics pipeline
- Stream SIMD processing model
- The design allocates a dedicated number of stream processors for each stage.
  - Each shader processor has different instruction set
  - Each supports particular input and output data elements
  - Mostly vector processors
  - Load balancing is predetermined.

Dedicated Processor Design



http://www.hotchips.org/archives/hc19/2_Mon/HC19.02/HC19.02.01.pdf

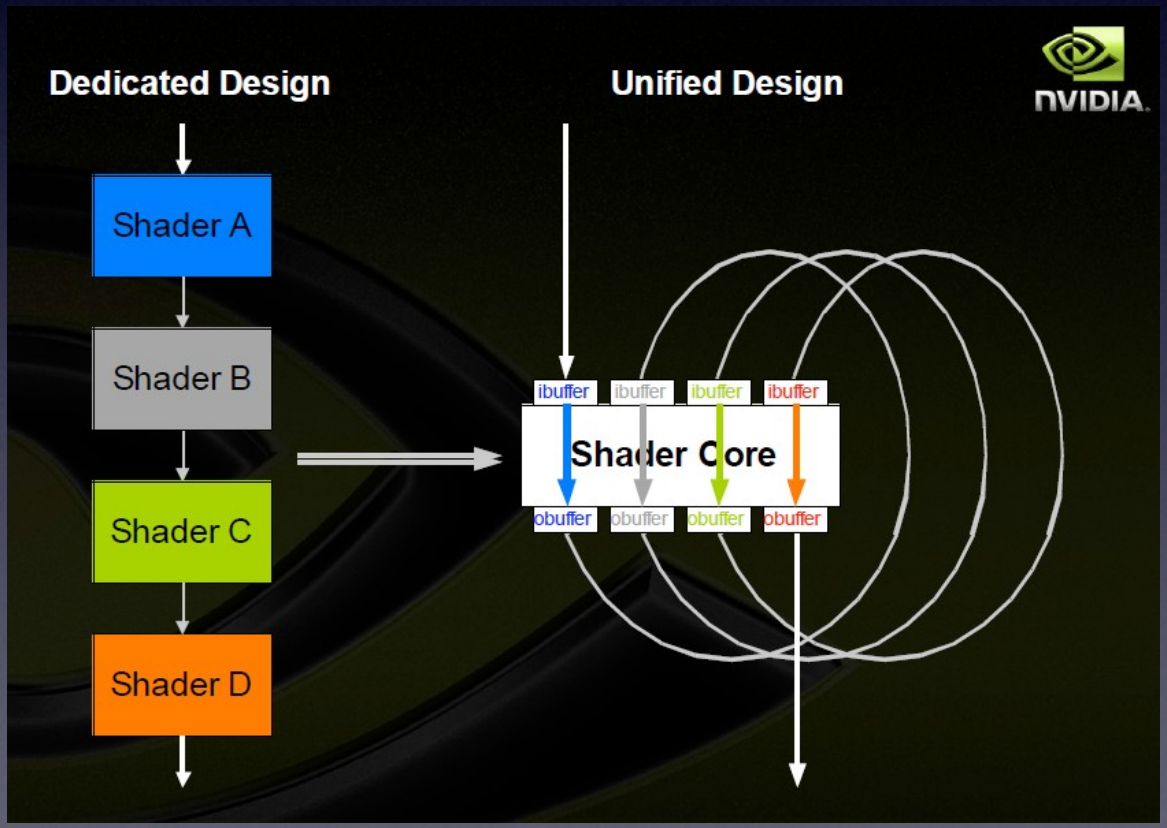
Dedicated Processor Design



http://www.hotchips.org/archives/hc19/2_Mon/HC19.02/HC19.02.01.pdf

Unified Shader Processor Design

- GeForce 8800 uses unified processor architecture
  - All shaders use the same instruction set, execute on the same units
  - Permits much better, dynamic load balancing



http://www.hotchips.org/archives/hc19/2_Mon/HC19.02/HC19.02.01.pdf

Dedicated Processor Design

Why unify?

Unified Shader

Heavy Geometry
Workload Perf =12

Unified Shader

Heavy Pixel
Workload Perf = 12

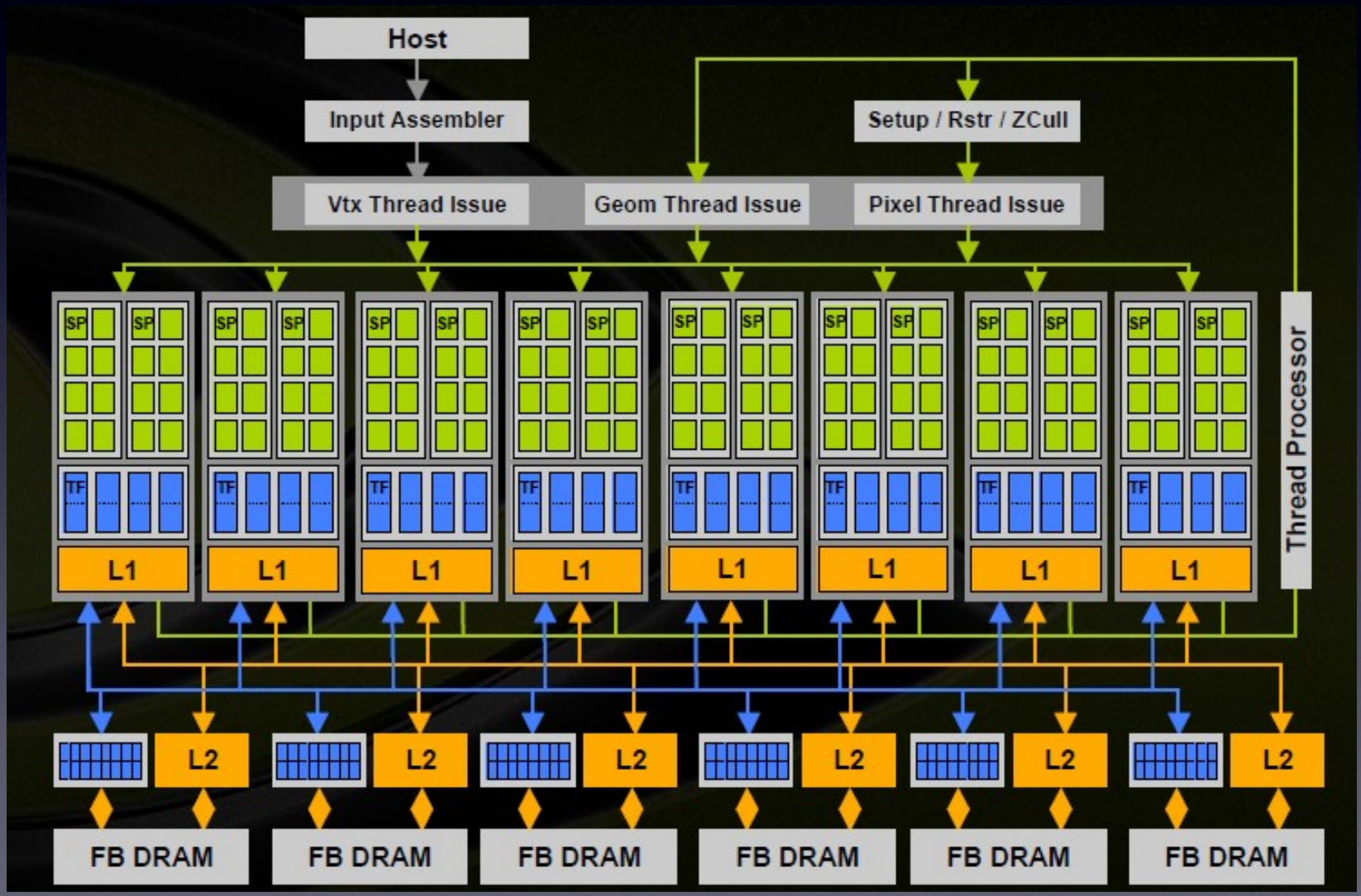
http://www.hotchips.org/archives/hc19/2_Mon/HC19.02/HC19.02.01.pdf

Unified Shader Processor Design

- Also enables additional, more versatile shader processors
- Geometry Shader
  - Executed after vertex shader
  - With adjacency information
  - Generate new triangles from those sent to the beginning of the pipeline
- Physics Engine
- More...

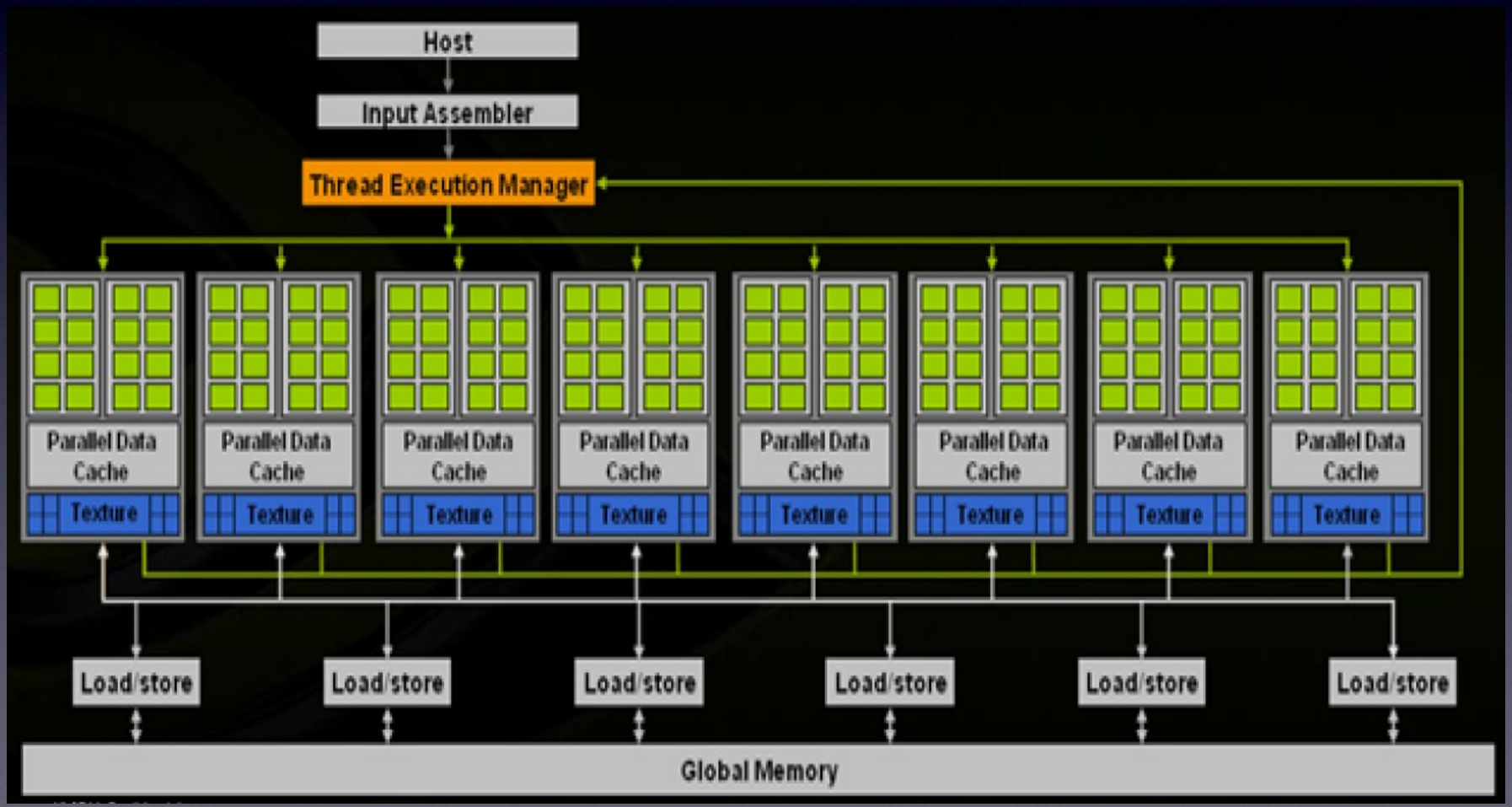
Unified Shader Processor Design

- 8800 Architecture (2006)



Unified Shader Processor Design

- 8800 Architecture (2006) – CUDA view
  - A result of evolution of modern graphics hardware



Graphics Data Elements

- Vertex Data
  - Vertex Buffer Objects (VBO)
- Pixel Data
  - Pixel Buffer Objects (PBO)
- Textures
- Frame Data
  - Frame Buffer Objects (FBO)
  - Can attach textures, depth buffer etc.
- You can image these buffer objects chuncks of GPU that store data elements (pointers are not allowed in graphics API)

Graphics Programming Language

- OpenGL (latest version: 3.0)
  - GLSL
- DirectX (latest version: 10.1)
  - HLSL

CUDA Advantages over Graphics Programming Languages

- Random access global memory
  - Threads can access any global memory location, and shared memory of the same block
- Unlimited access to memory
  - Threads can read/write as many locations as needed
  - Indirection
- Block-wise synchronization
- Lower learning curve
  - A few extensions to C
  - No knowledge of graphics pipeline required

Some Graphics Concepts

- Texture Linear Interpolation
- Texture Filtering
  - Mipmaps
  - Summed Area Table
    - How to efficiently generate? (say, if you have a huge texture)