



CS 354

GPU Architecture

Mark Kilgard
University of Texas
March 6, 2012

Today's material

- In-class quiz
- Lecture topic
 - *Architecture of Graphics Processing Units (GPUs)*
- Course work
 - Homework #4 due today
 - Review textbook reading
 - Chapter 5, 6, and 7
 - Project #2 on texturing, shading, & lighting is coming
 - **Remember: Midterm in-class on March 8**

My Office Hours

- Tuesday, before class
 - Painter (PAI) 5.35
 - 8:45 a.m. to 9:15
- Thursday, after class
 - ACE 6.302
 - 11:00 a.m. to 12
- Randy's office hours
 - Monday & Wednesday
 - 11 a.m. to 12:00
 - Painter (PAI) 5.33



Last time, this time

- Last lecture, we discussed
 - Programmable shading
 - Graphics hardware shading languages
- This lecture
 - How do GPUs work?

Daily Quiz

On a sheet of paper

- Write your EID, name, and date
- Write #1, #2, #3, #4 followed by its answer

1.

Pick the best choice: Shade trees are

- a) fractal trees with shadows
- b) OpenGL commands
- c) hierarchical arrangements of shading computations
- d) fractal patterns of all sorts

2.

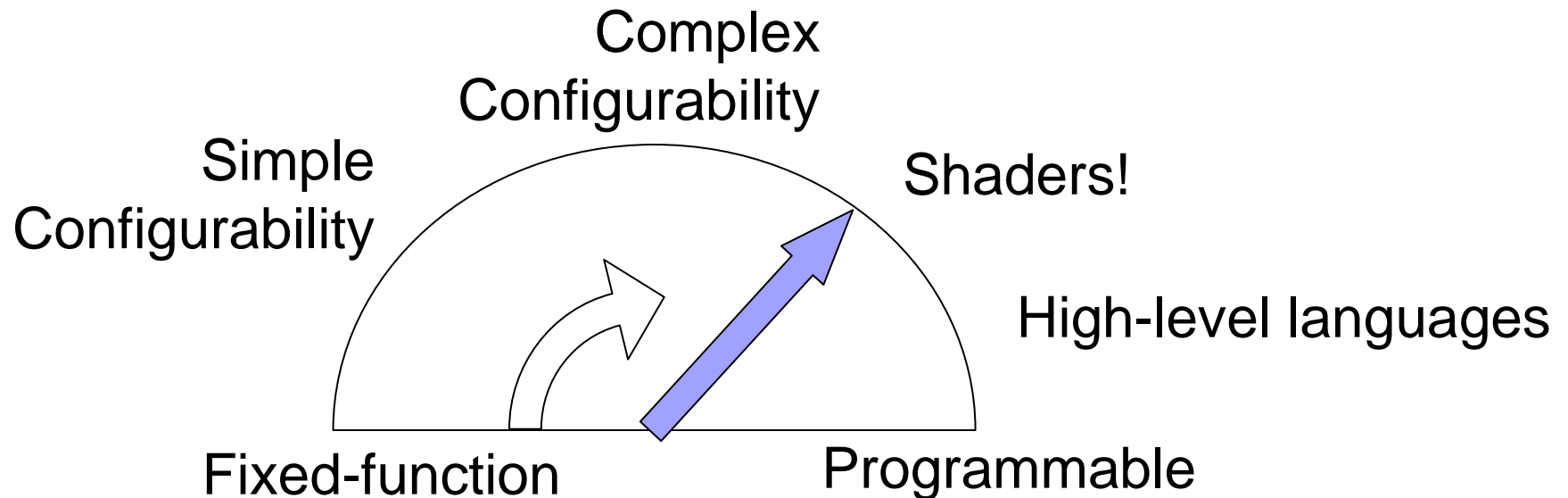
Name one general purpose programming language that GLSL borrows from.

3.

Multiple choice: The GLSL standard has built-in data types for

- a) vectors
- b) matrices
- c) texture samplers
- d) floating-point values
- e) pointers to malloc'ed memory
- f) a through e
- g) a through d

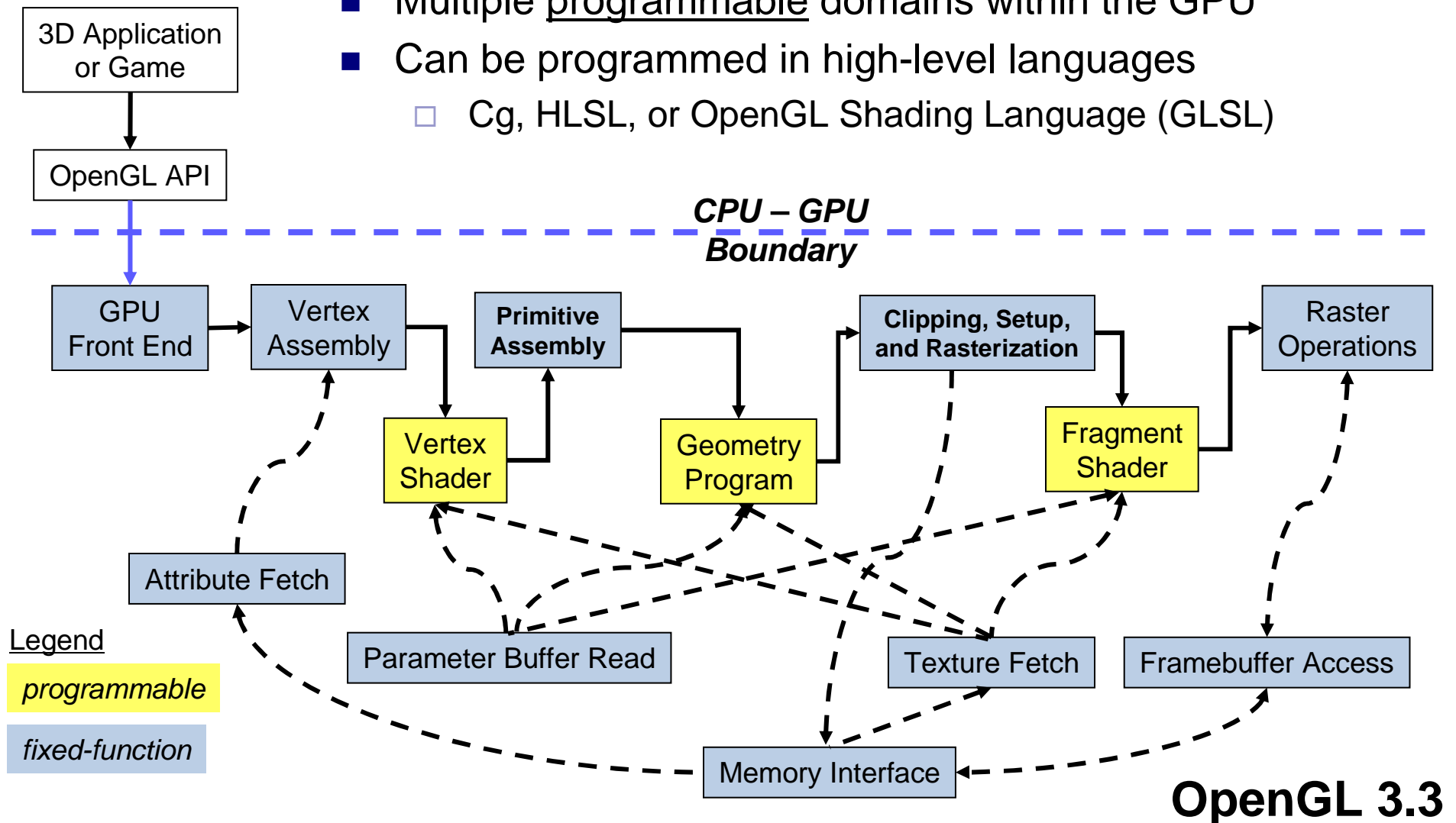
Key Trend in OpenGL Evolution



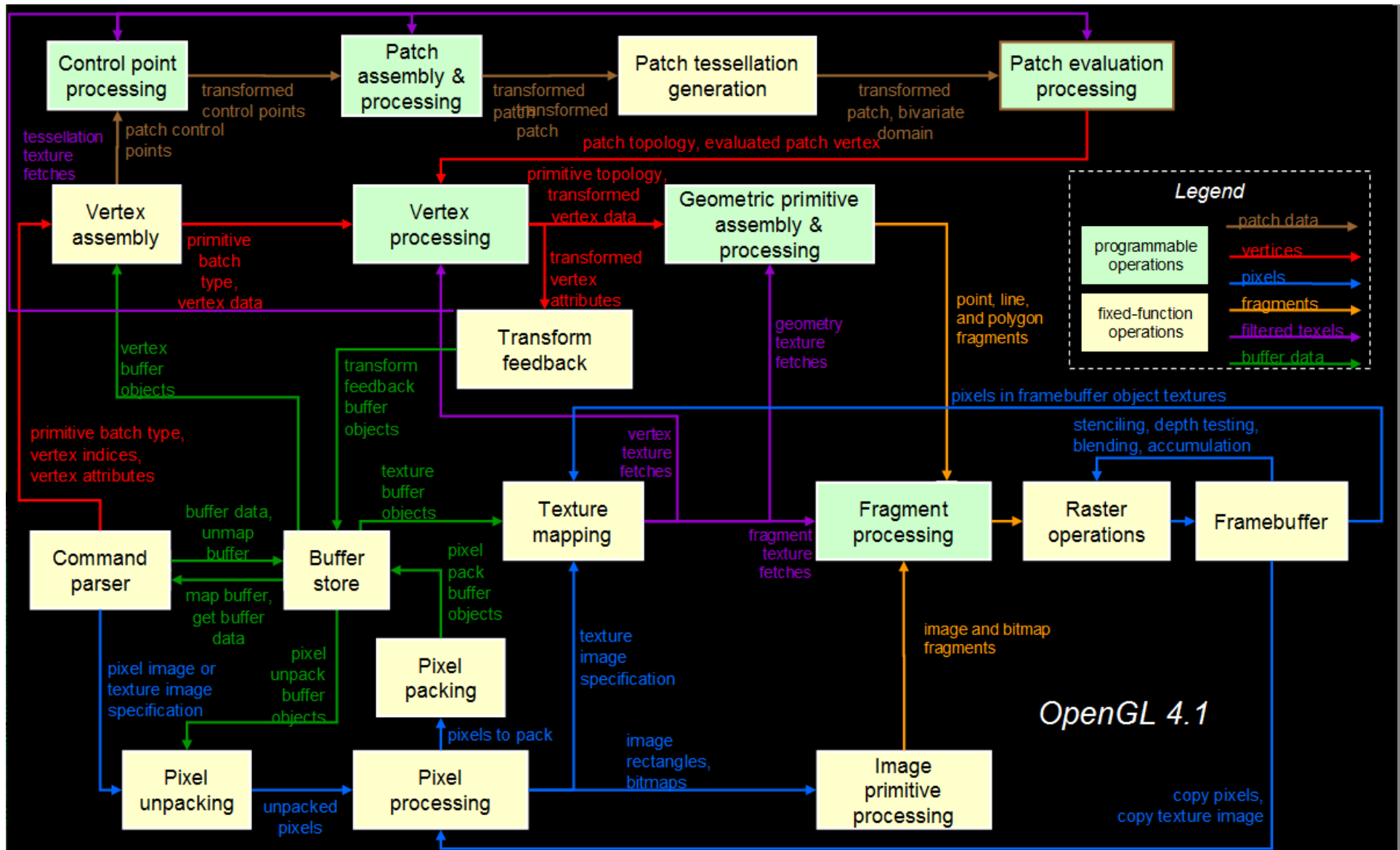
- Direct3D follows the same trend
- Also reflects trend in GPU architecture
 - API and hardware co-evolving

Programming Shaders **inside** GPU

- Multiple programmable domains within the GPU
- Can be programmed in high-level languages
 - Cg, HLSL, or OpenGL Shading Language (GLSL)



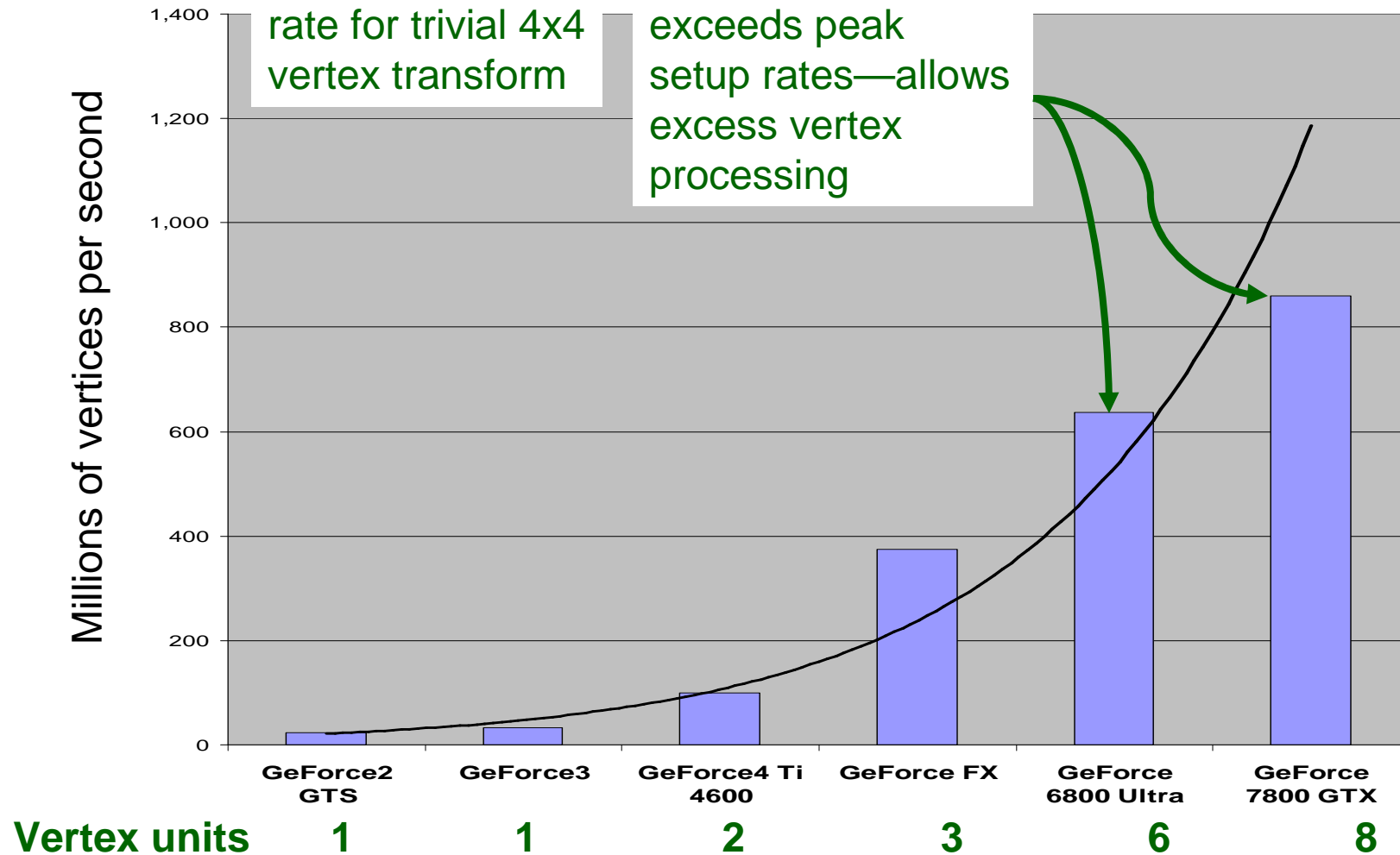
Complex OpenGL Data Flow



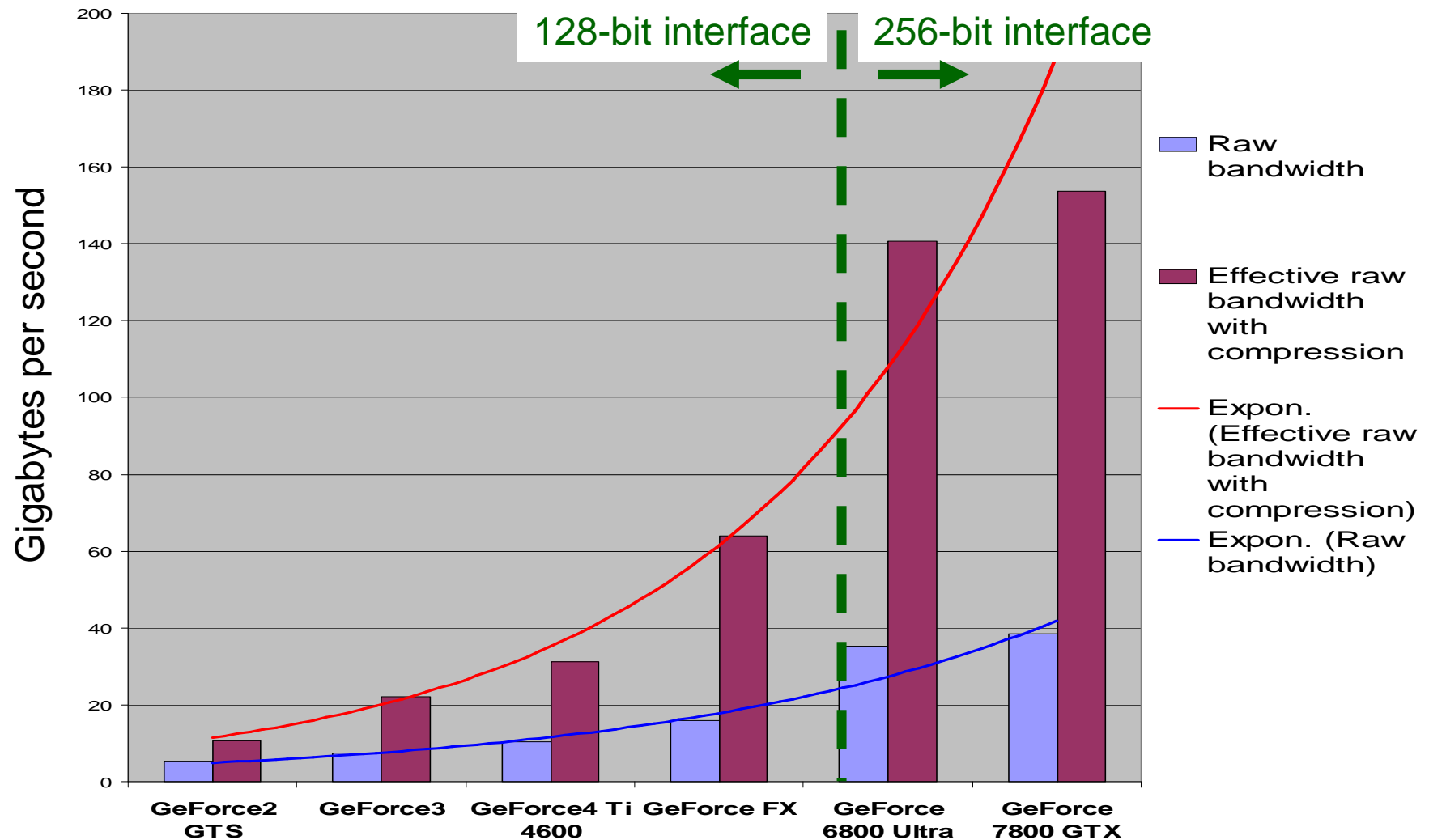
Six Years of GPU Architecture

Product		New Features	OpenGL Version	Direct3D Version
2000	GeForce 256	Hardware transform & lighting, configurable fixed-point shading, cube maps, texture compression, anisotropic texture filtering	1.3	DX7
2001	GeForce3	Programmable vertex transformation, 4 texture units, dependent textures, 3D textures, shadow maps, multisampling, occlusion queries	1.4	DX8
2002	GeForce4 Ti 4600	Early Z culling, dual-monitor	1.4	DX8.1
2003	GeForce FX	Vertex program branching, floating-point fragment programs, 16 texture units, limited floating-point textures, color and depth compression	1.5	DX9
2004	GeForce 6800 Ultra	Vertex textures, structured fragment branching, non-power-of-two textures, generalized floating-point textures, floating-point texture filtering and blending	2.0	DX9c
2005	GeForce 7800 GTX	Transparency antialiasing	2.0	DX9c

GeForce Peak Vertex Processing Trends



GeForce Peak Memory Bandwidth Trends



Effective GPU Memory Bandwidth

■ **Compression schemes**

- Lossless depth and color (when multisampling) compression
- Lossy texture compression (S3TC / DXTC)
- Typically assumes 4:1 compression

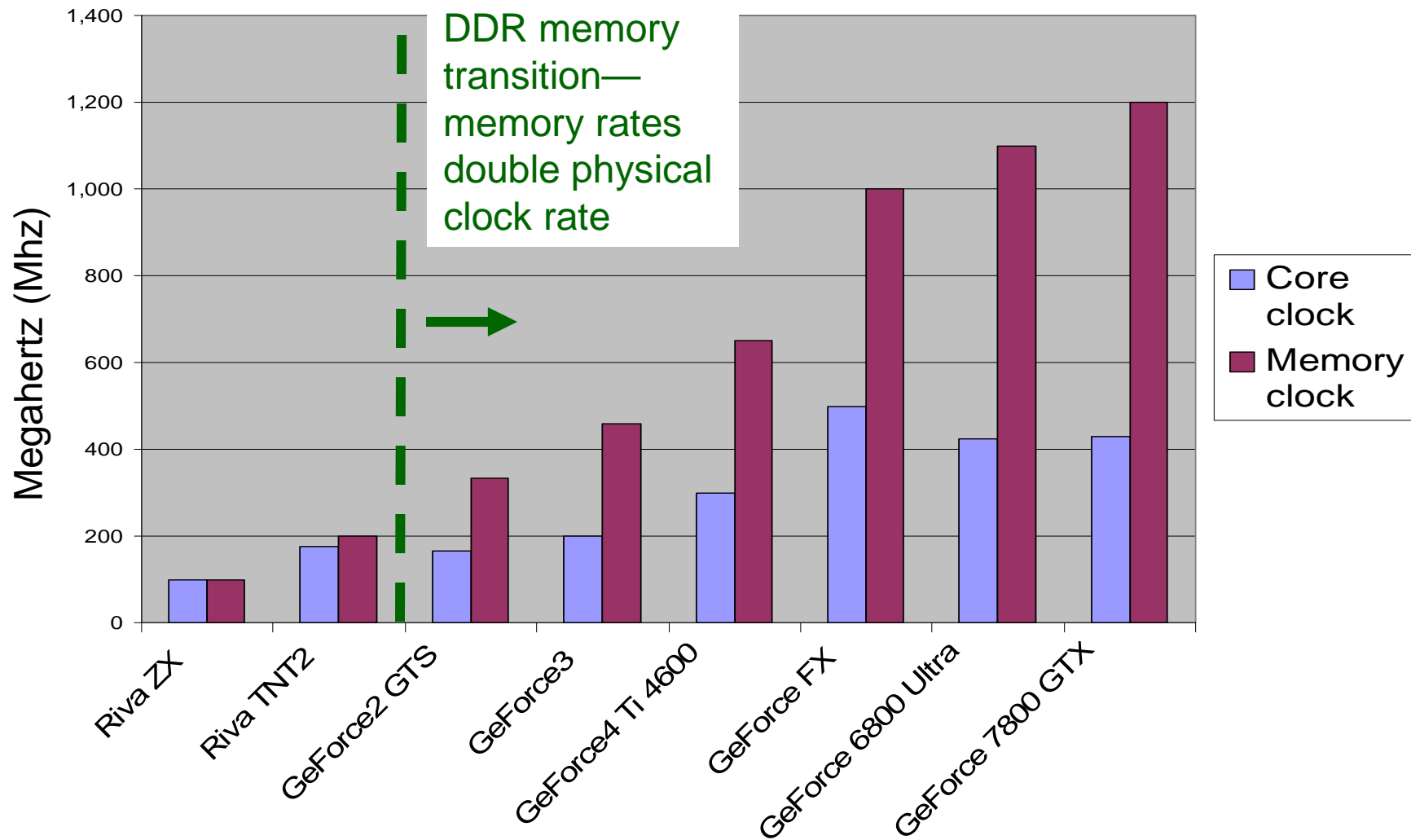
■ **Avoidance useless work**

- Early killing of fragments (Z cull)
- Avoiding useless blending and texture fetches

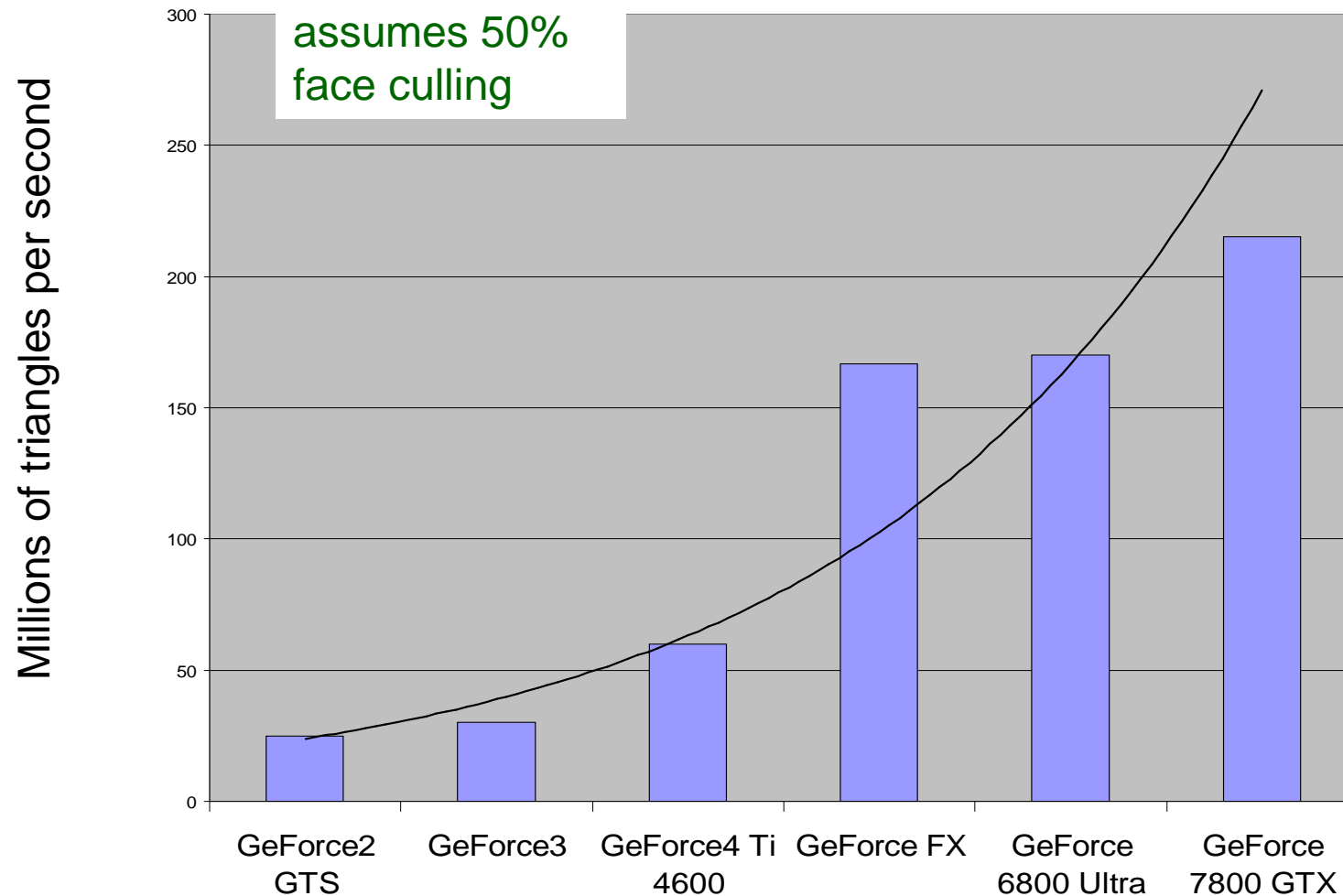
■ **Very clever memory controller designs**

- Combining memory accesses for improved coherency
- Caches for texture fetches

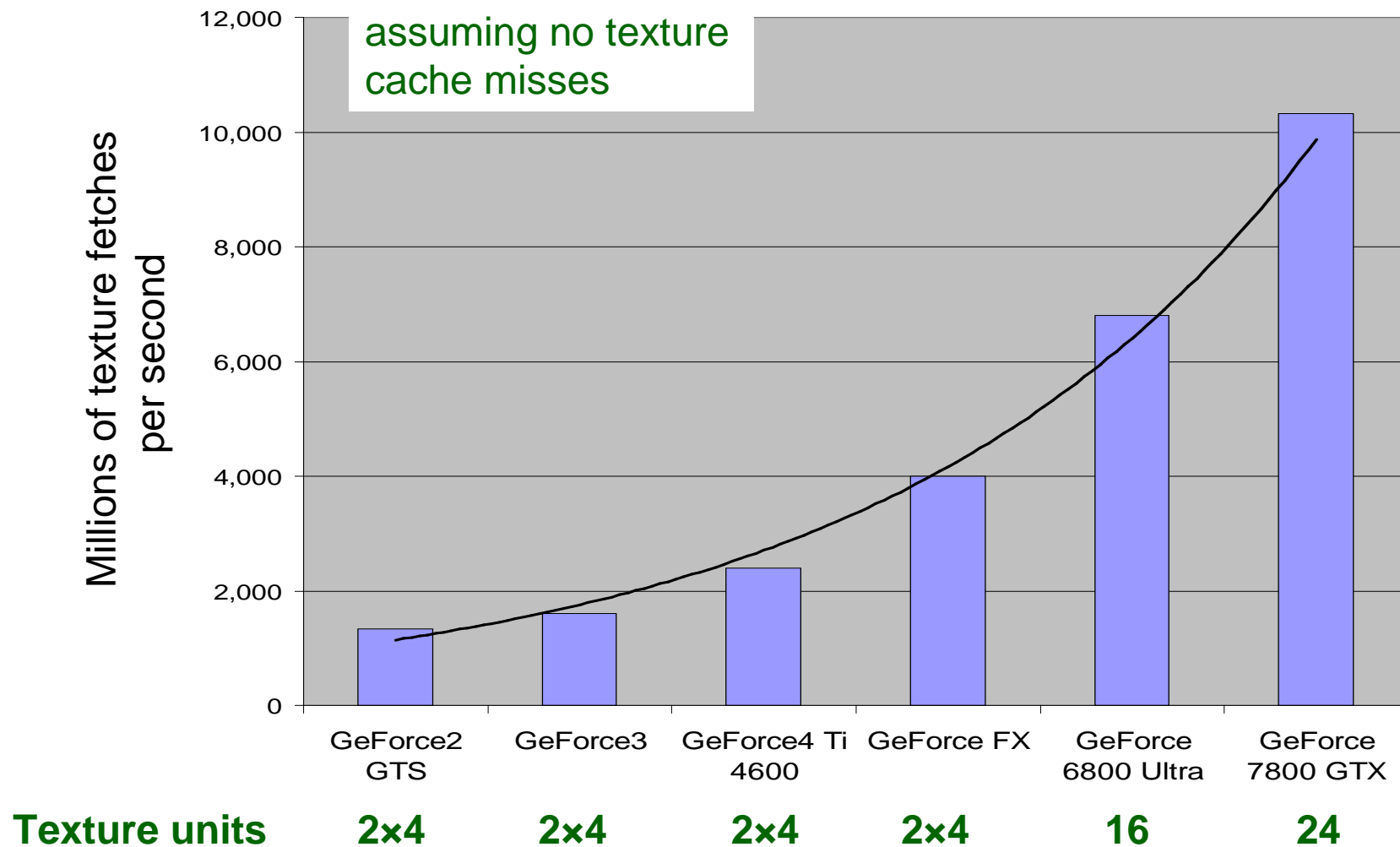
GeForce Core and Memory Clock Rates



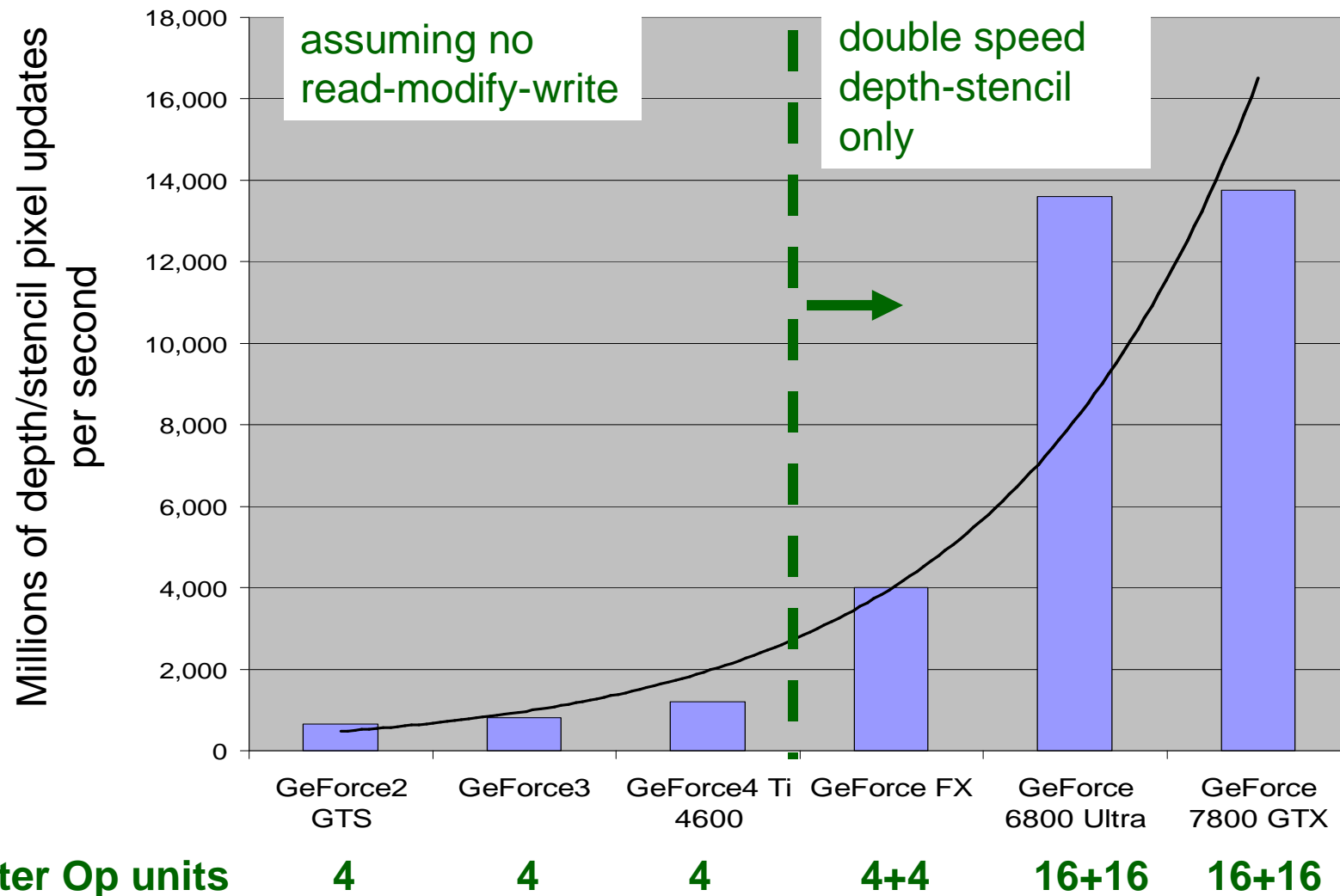
GeForce Peak Triangle Setup Trends



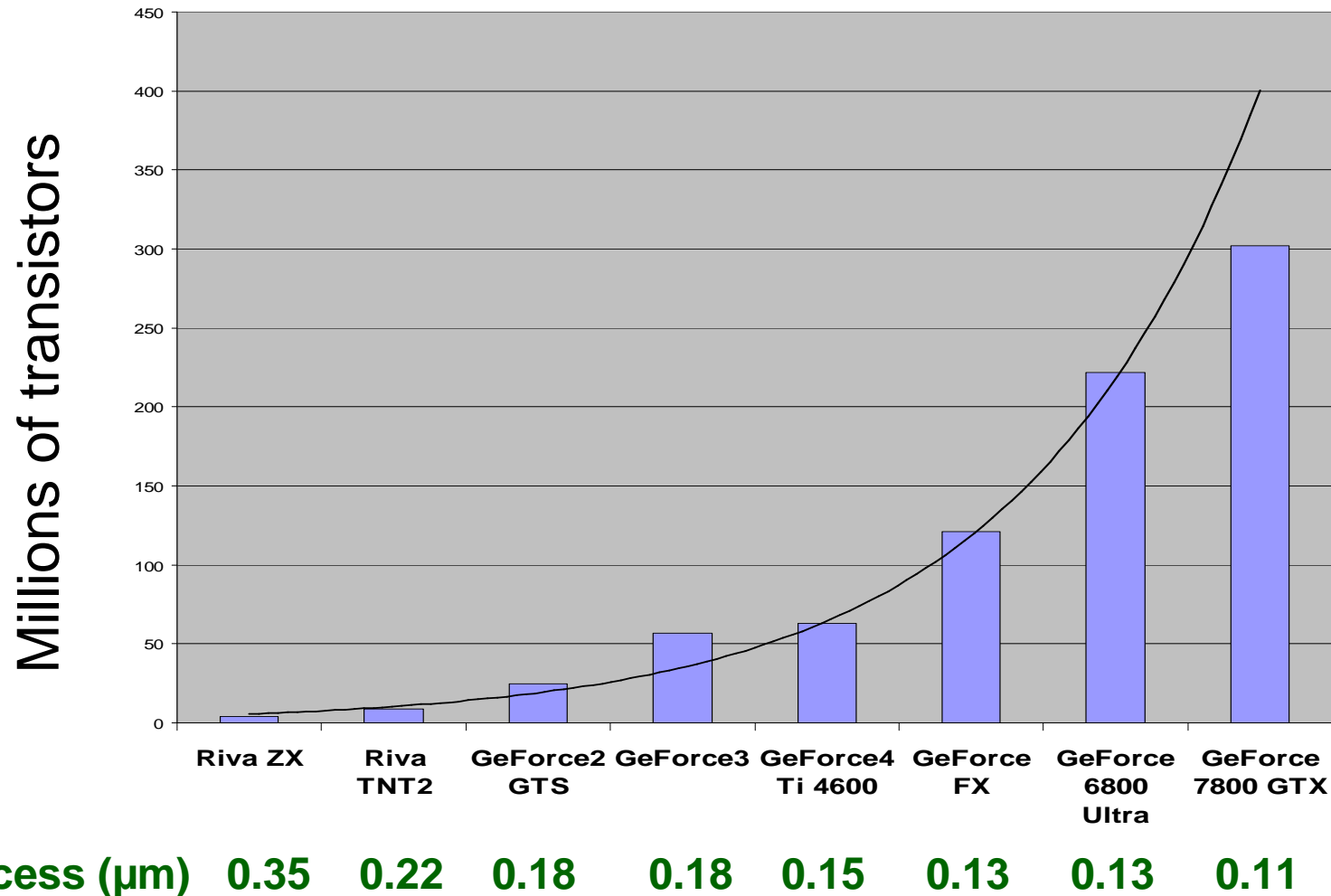
GeForce Peak Texture Fetch Trends

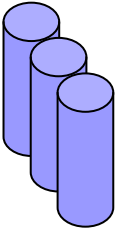
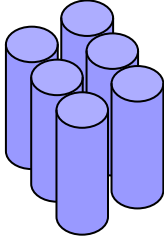
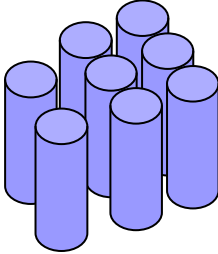
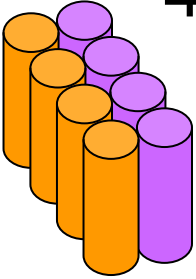
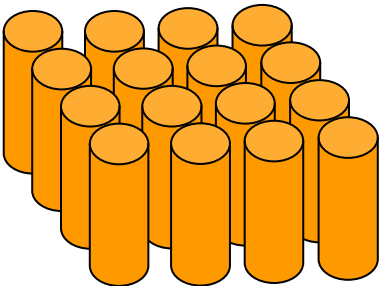
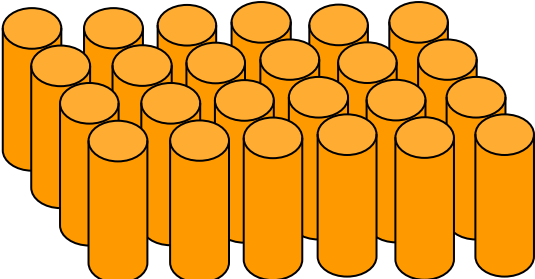
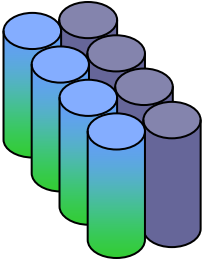
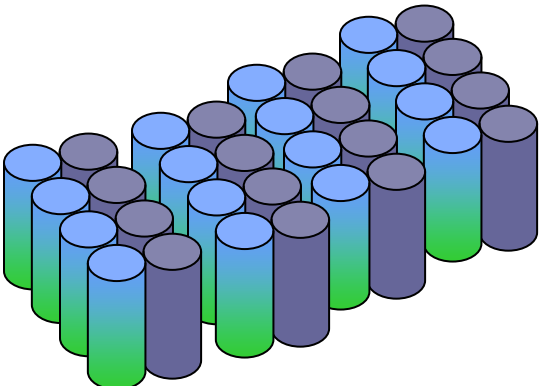
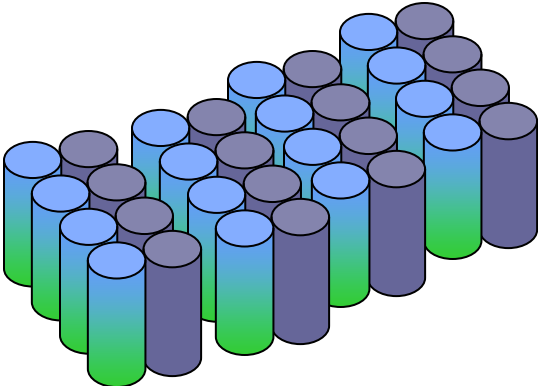


GeForce Peak Depth/Stencil-only Fill

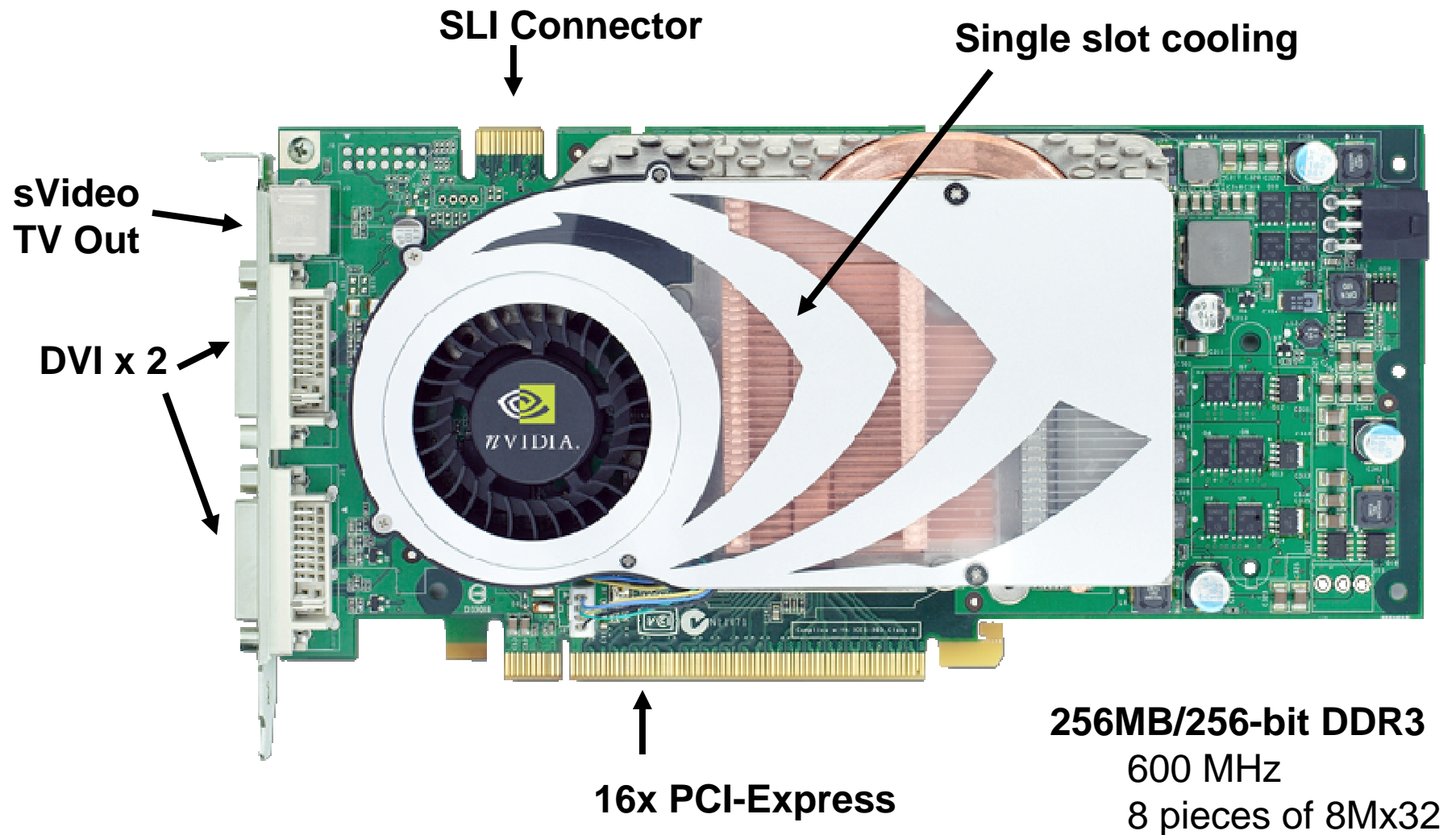


GeForce Transistor Count and Semiconductor Process



Hardware Unit	GeForce FX 5900	GeForce 6800 Ultra	GeForce 7800 GTX
Vertex	 3	 6	 8
Fragment 2 nd Texture Fetch	 4+4	 16	 24
Raster Color Raster Depth	 4+4	 16+16	 16+16

GeForce 7800 GTX Board Details



GeForce 7800 GTX

GPU Details

302 million transistors
430 MHz core clock
256-bit memory interface

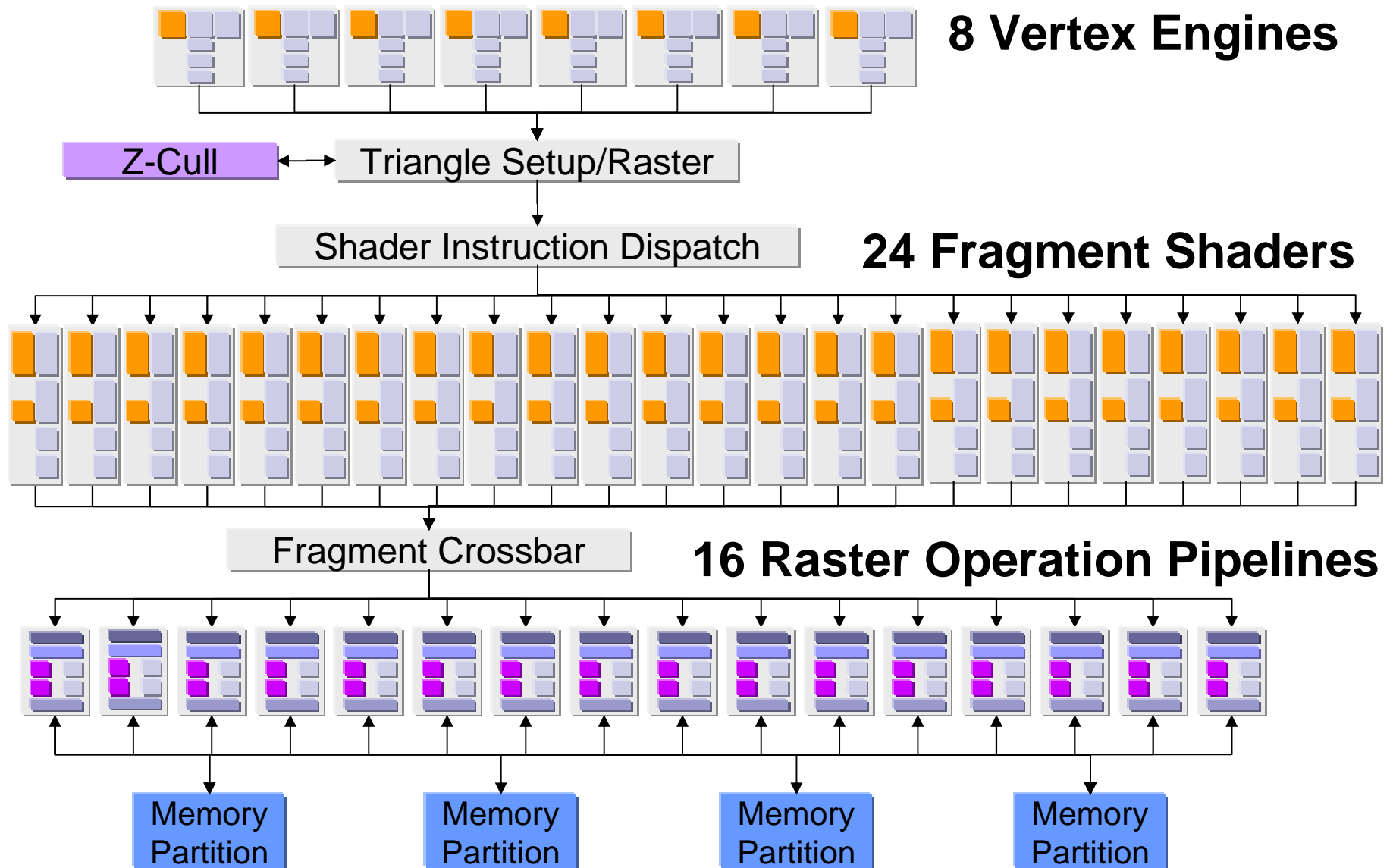


Notable Functionality

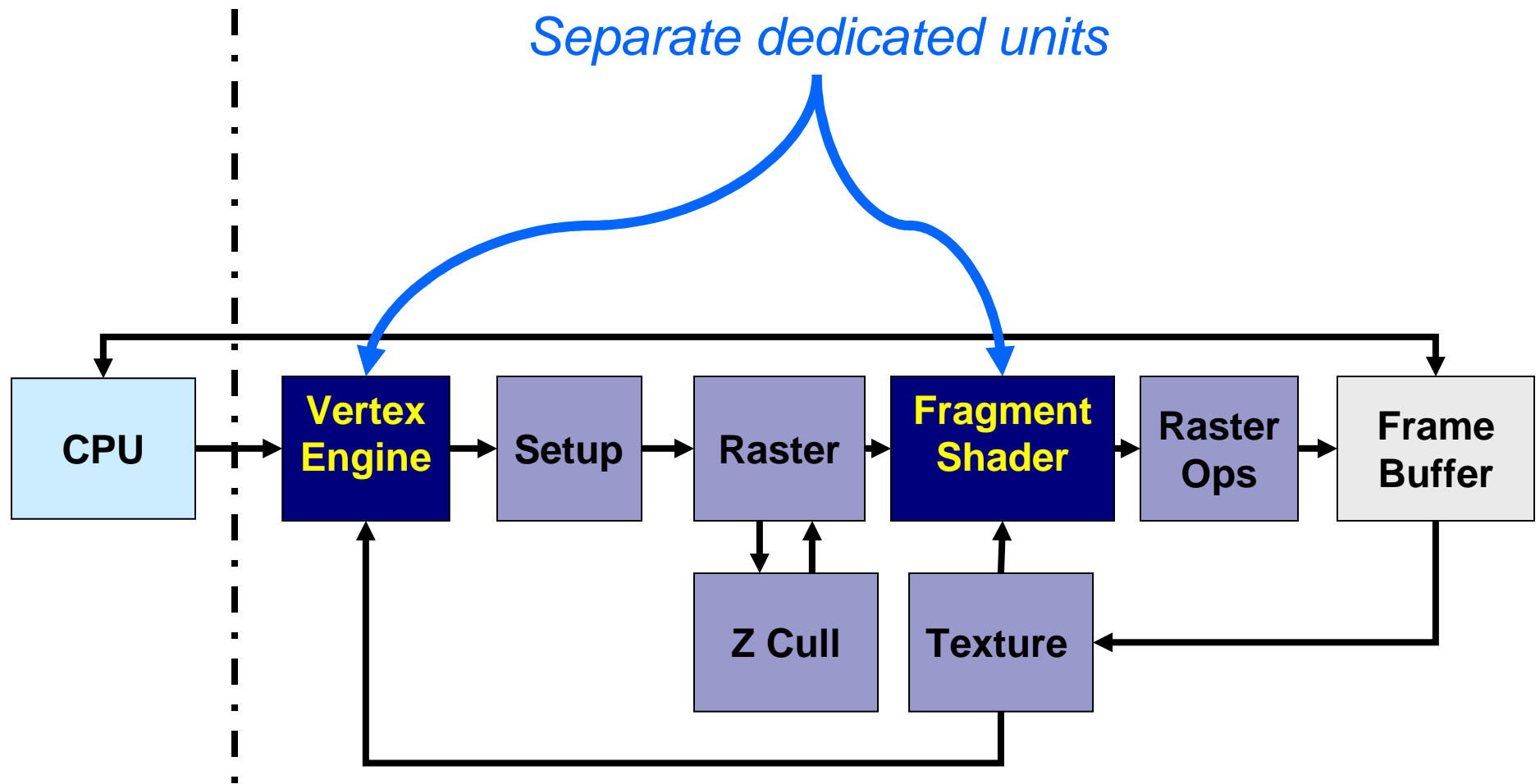
- Non-power-of-two textures with mipmaps
- Floating-point (fp16) blending and filtering
- sRGB color space texture filtering and frame buffer blending
- Vertex textures
- 16x anisotropic texture filtering
- Dynamic vertex *and* fragment branching
- Double-rate depth/stencil-only rendering
- Early depth/stencil culling
- Transparency antialiasing

GeForce 7800 GTX

Parallelism

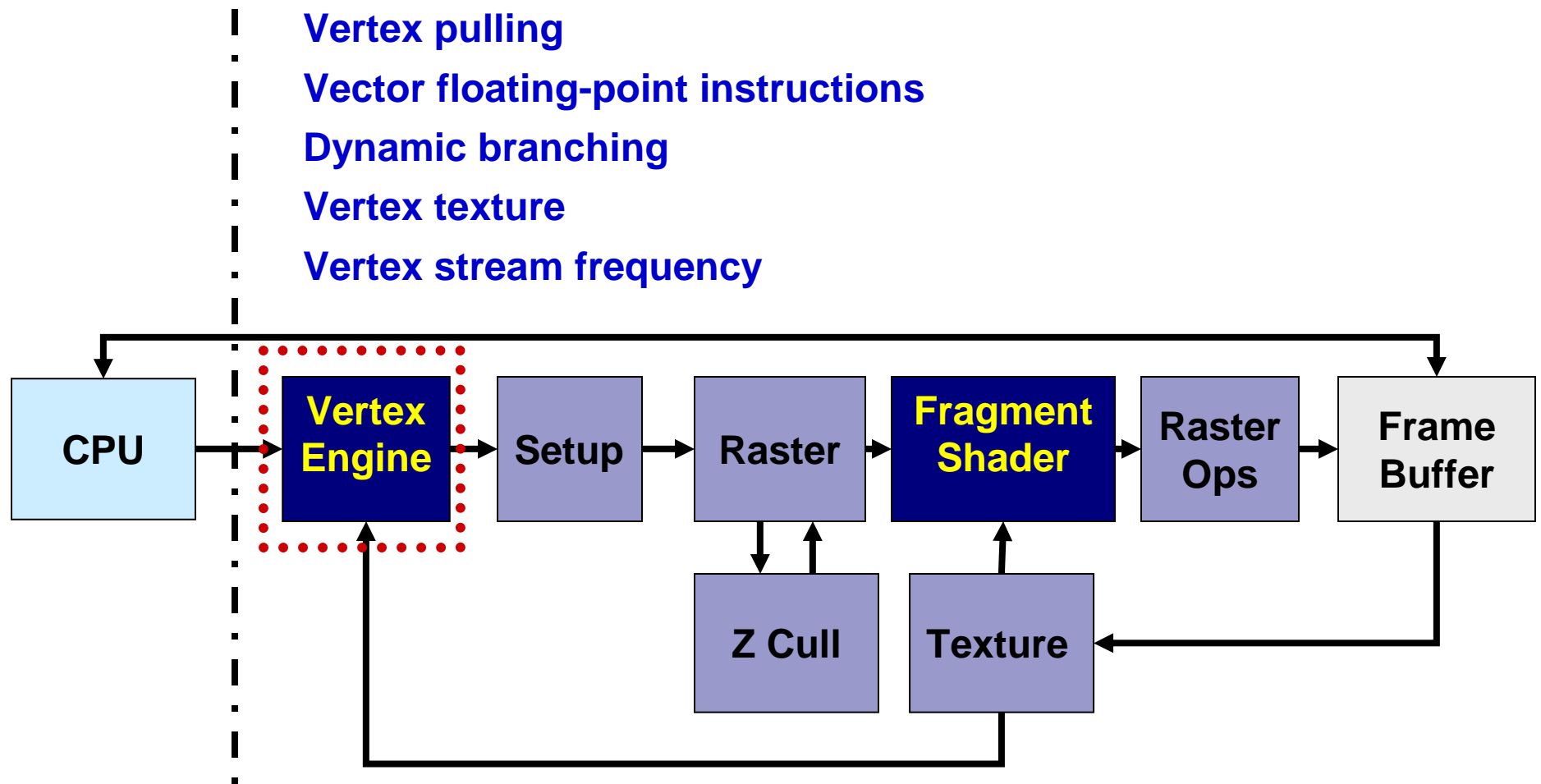


GeForce Graphics Pipeline

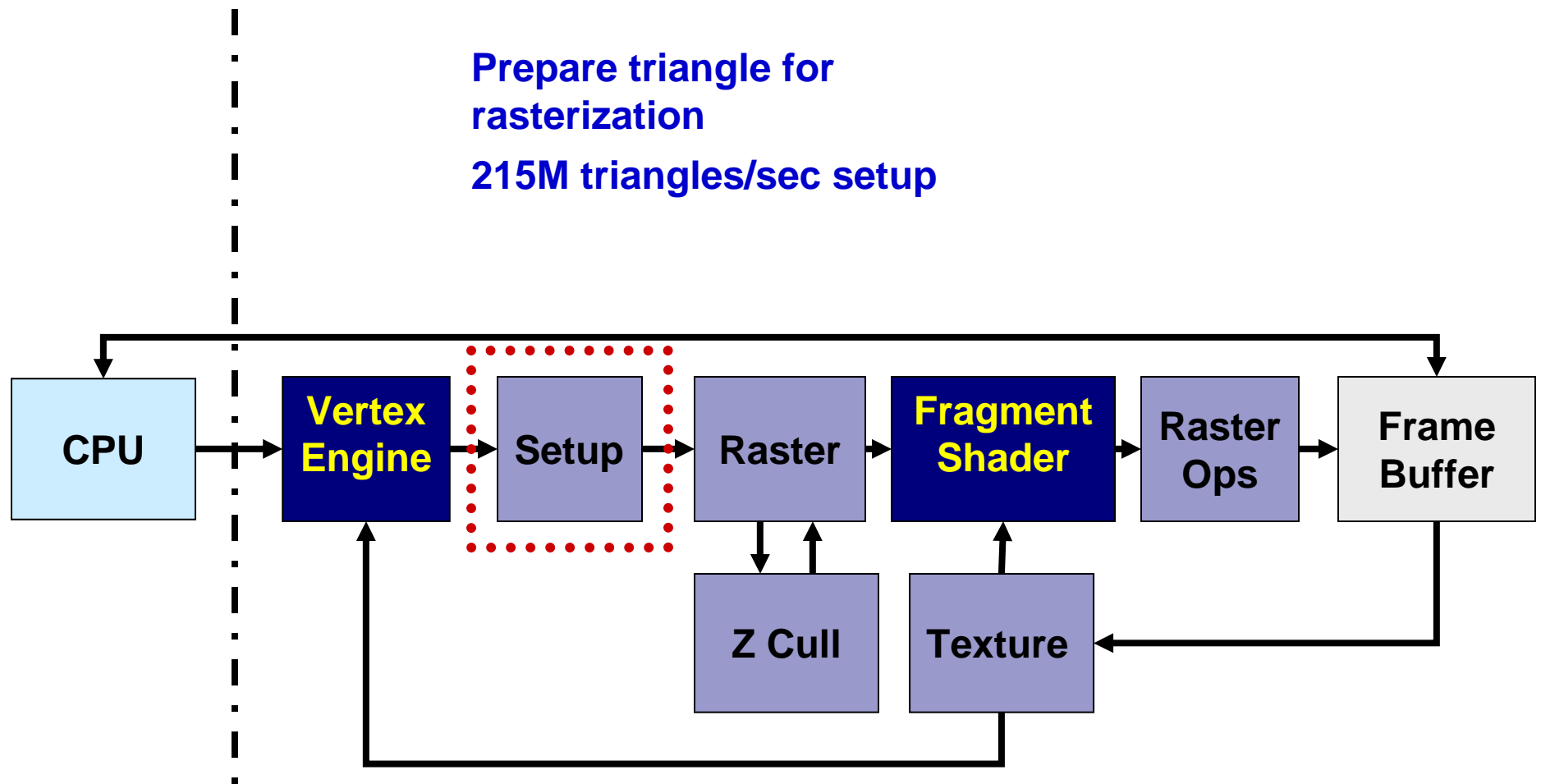


GeForce Graphics Pipeline

Vertex Engine

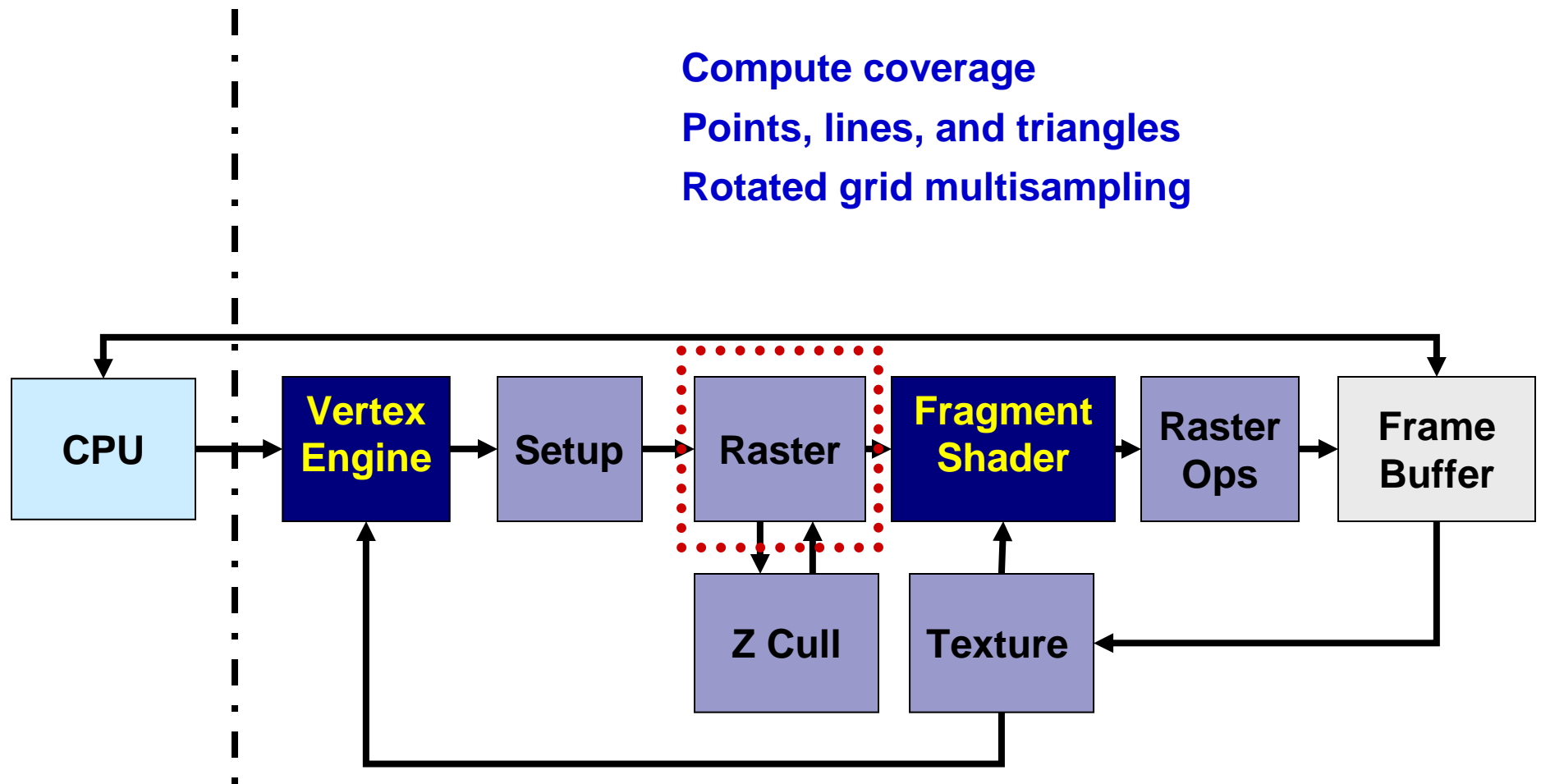


GeForce Graphics Pipeline Setup



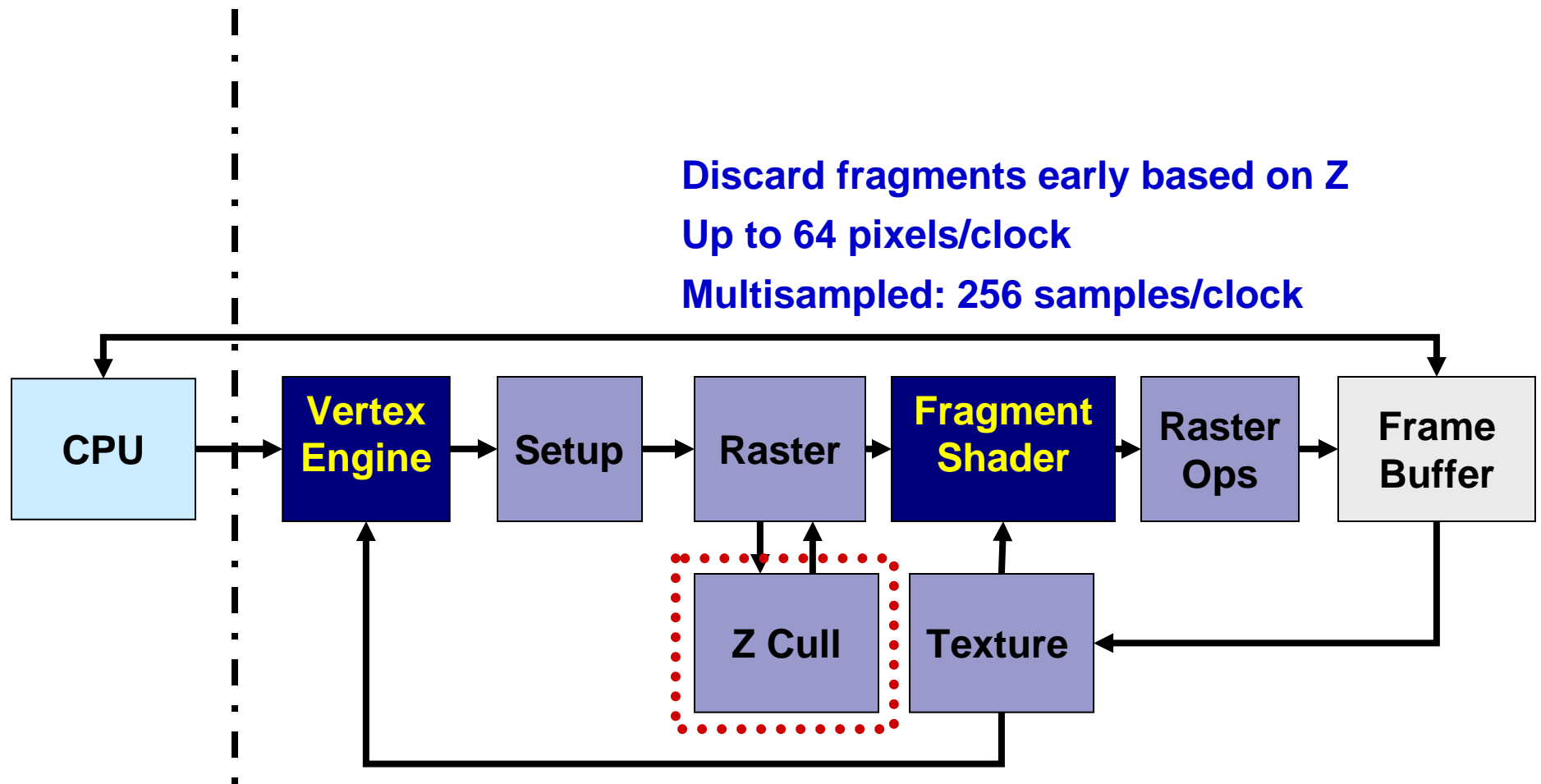
GeForce Graphics Pipeline

Raster



GeForce Graphics Pipeline

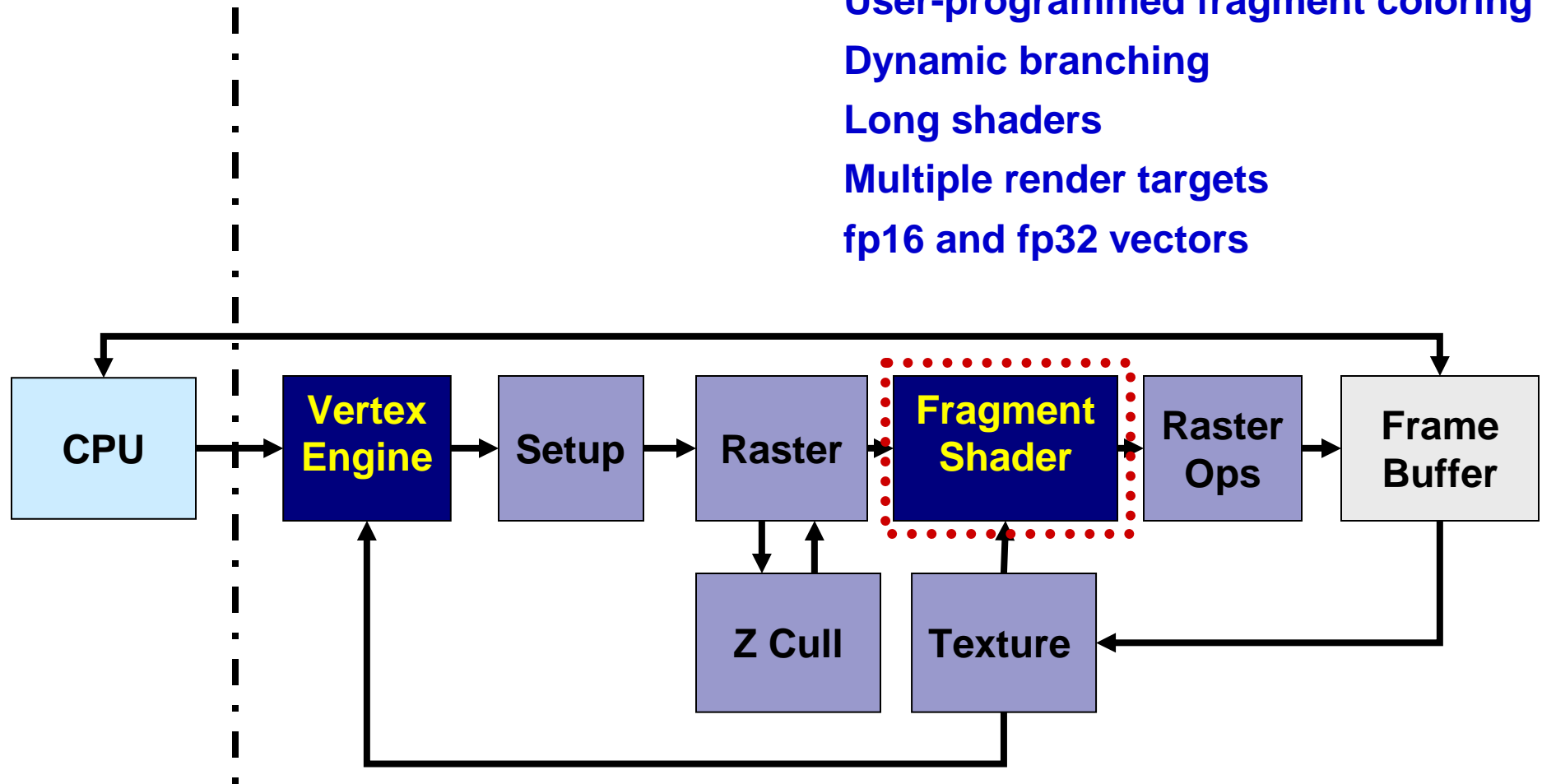
Z Cull



GeForce Graphics Pipeline

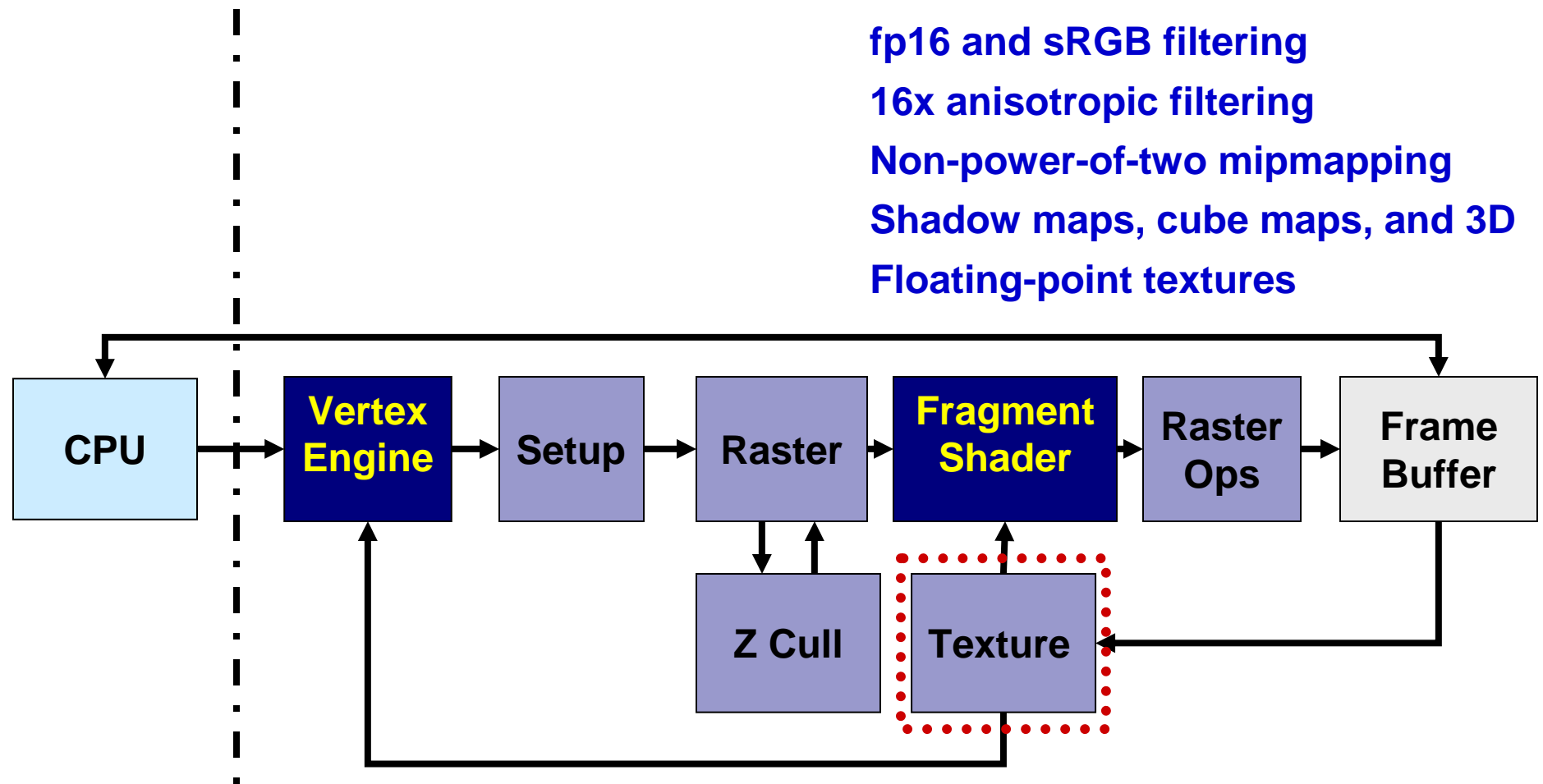
Fragment Shader

User-programmed fragment coloring
Dynamic branching
Long shaders
Multiple render targets
fp16 and fp32 vectors



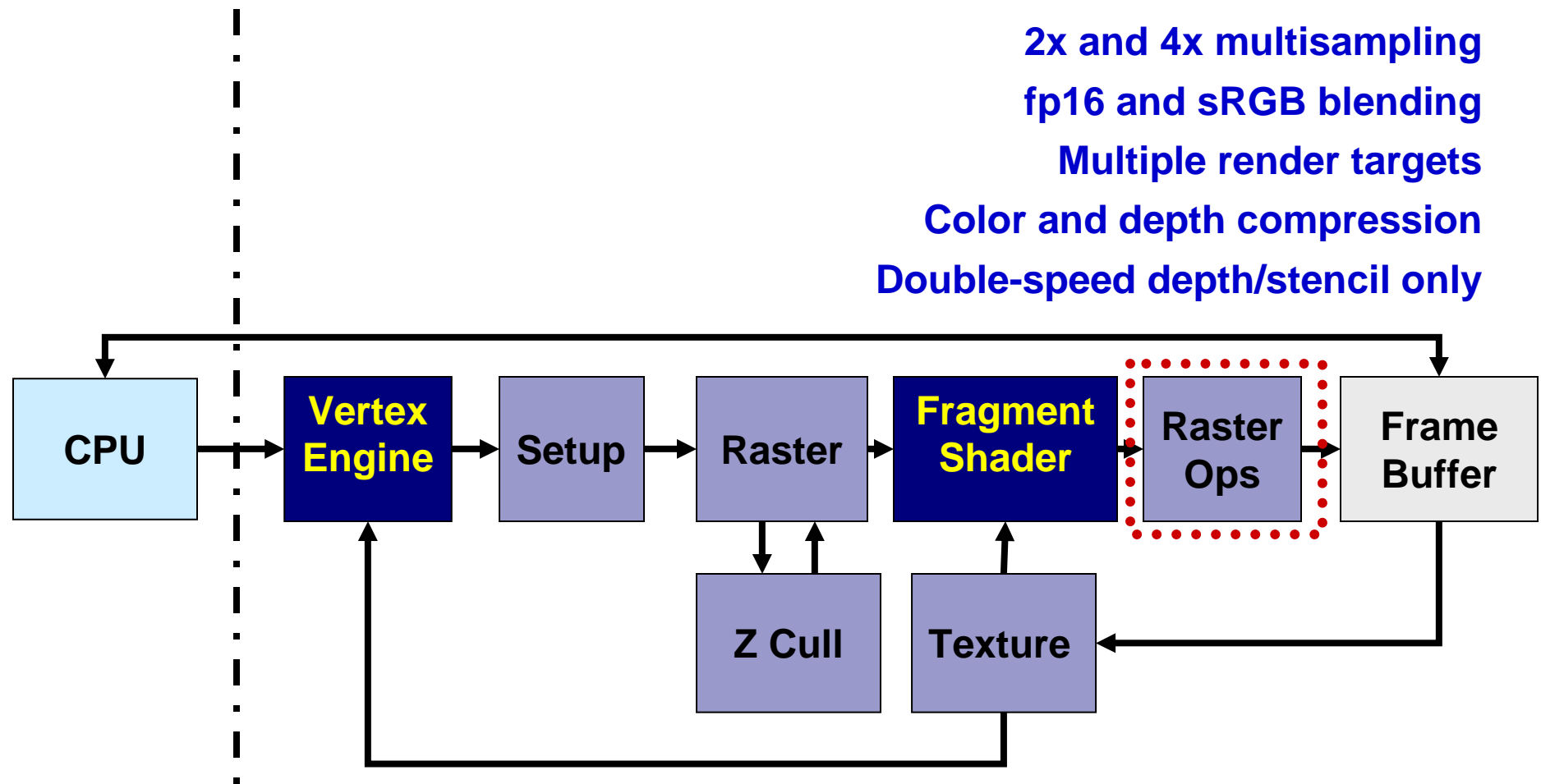
GeForce Graphics Pipeline

Texture

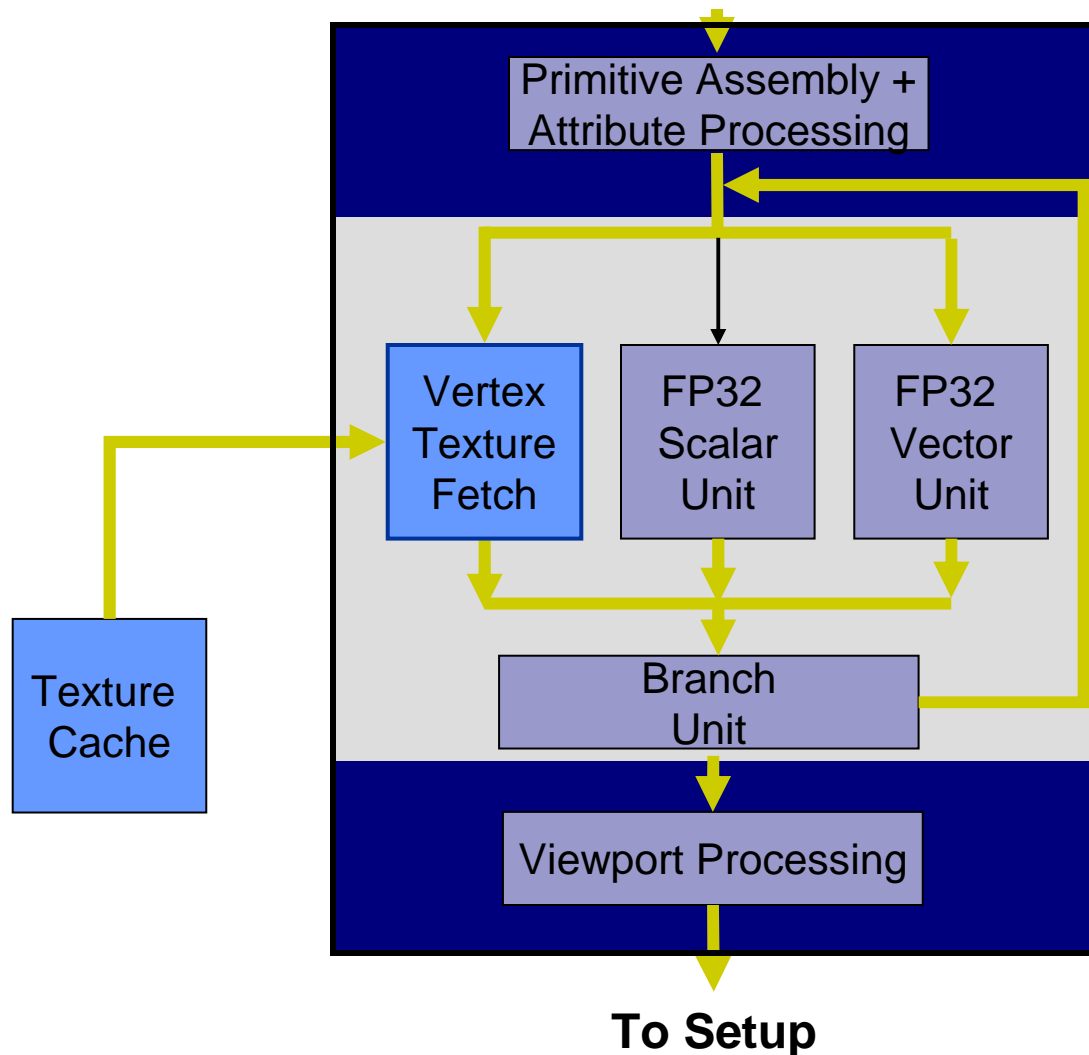


GeForce Graphics Pipeline

Texture



Single GeForce 7800 Vertex Unit



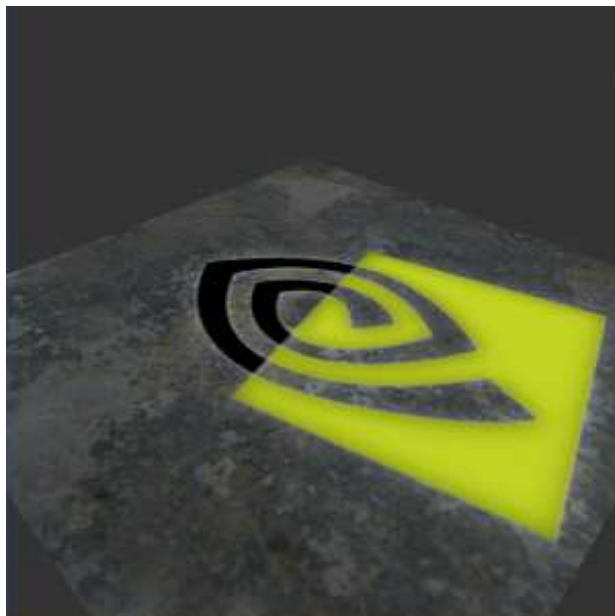
Vertex Processing Engine

- MIMD Architecture
- Dual Issue
- Low-penalty branching
- Shader Model 3.0
- 32 vector registers
- 512 static instructions per program
- Indexed input and output registers

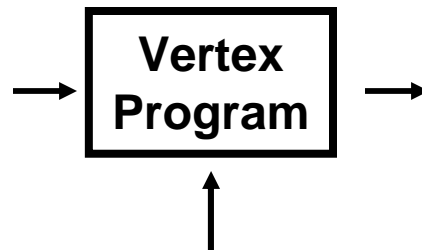
Vertex Texture Fetch

- Non-stalling
- Up to 4 texture units
 - Unlimited fetches
- Mipmapping, no filtering

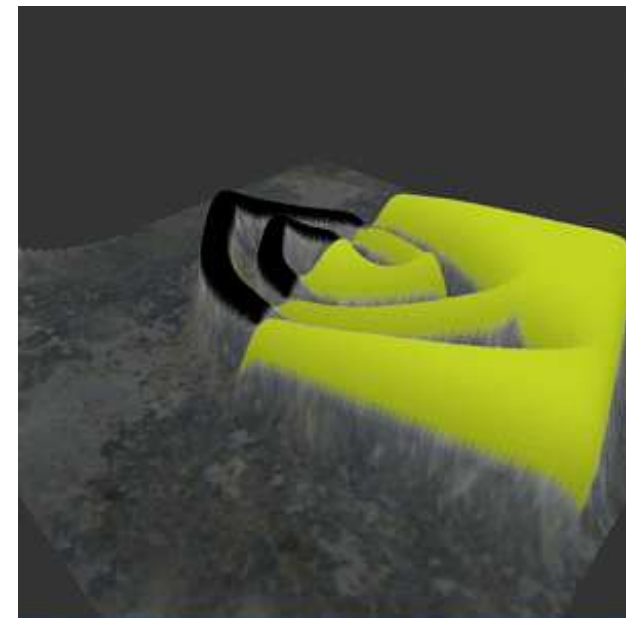
Vertex Texturing Example



Flat tessellated mesh



**Height field
texture**

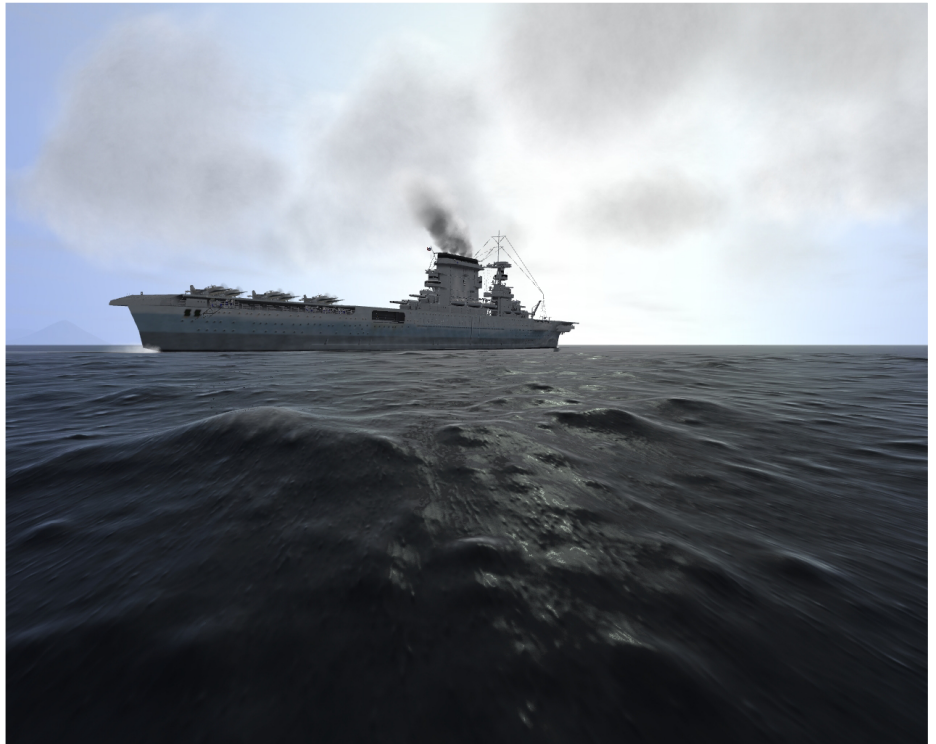


Displaced mesh

Vertex Textures for Dynamic Displacement Mapping



Without Vertex Textures

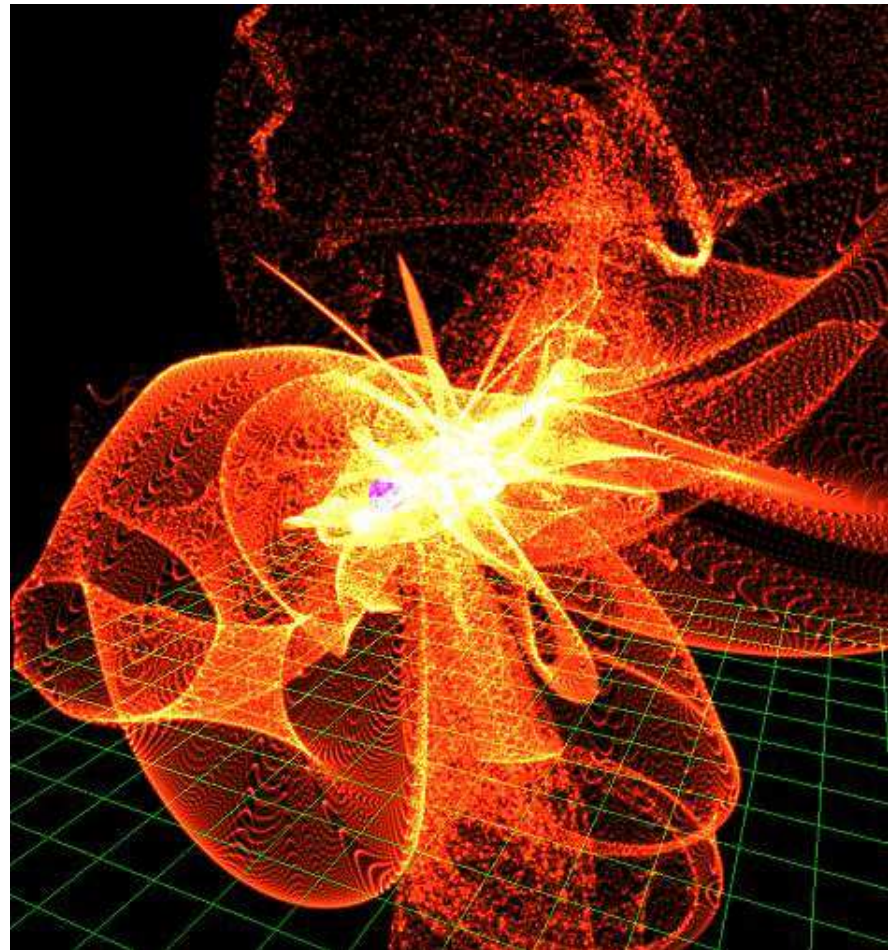


With Vertex Textures

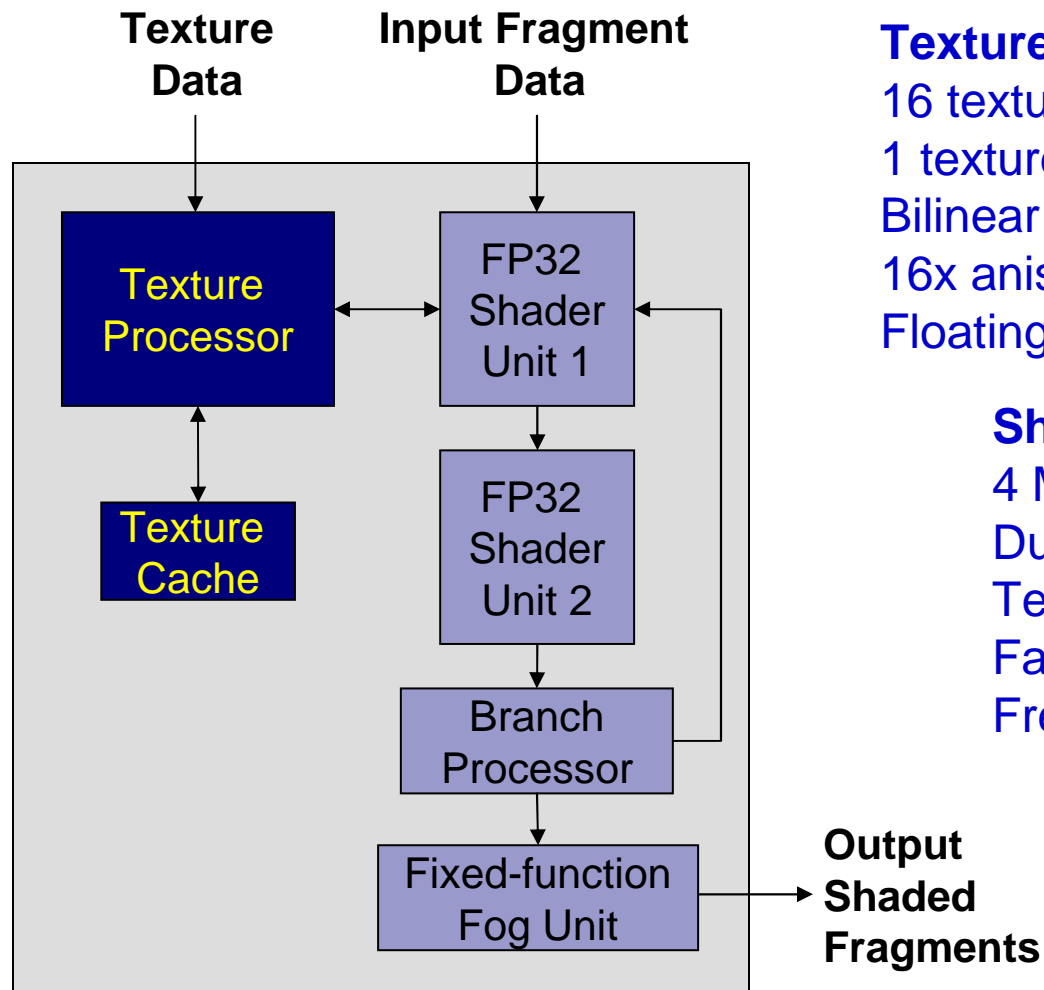
Images used with permission from *Pacific Fighters*. © 2004 Developed by 1C:Maddox Games.
All rights reserved. © 2004 Ubi Soft Entertainment.

Vertex Textures to Drive Particle Systems

- Render-to-texture
 - Simulation runs in floating-point frame buffer, also usable as texture
- Vertex textures
 - Determines particle location with vertex texture fetch



Single GeForce 7800 Fragment Shader Pipeline



Texture Processor

16 texture units
1 texture fetch at full speed
Bilinear or tri-linear filtering
16x anisotropic filtering
Floating-point (fp16) texture filtering

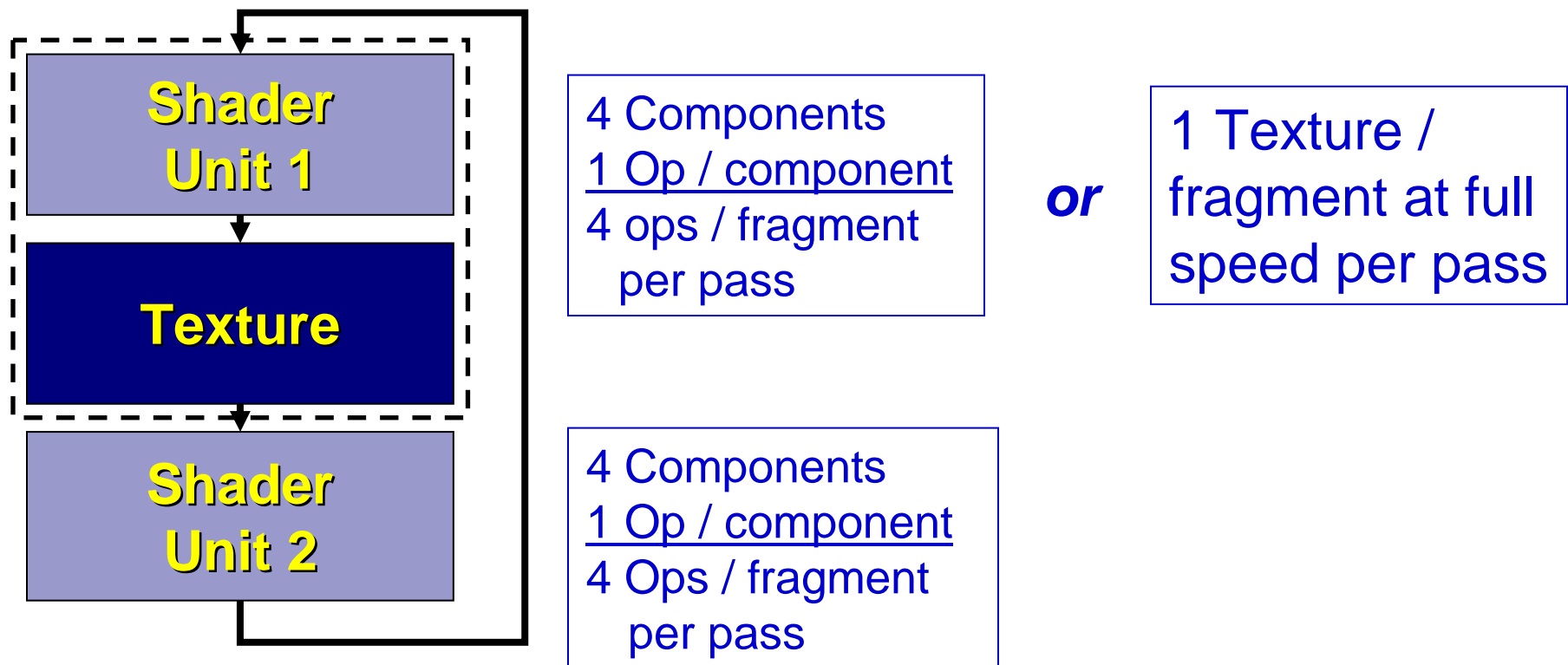
Shader Unit 1

4 MULs + RCP
Dual Issue
Texture address calculation
Fast fp16 normalize
Free: negate, abs, condition codes

Shader Unit 2

4 MADs or DP4
Dual Issue
Free: negate, abs, condition codes

Operations Per Fragment Shader Pass

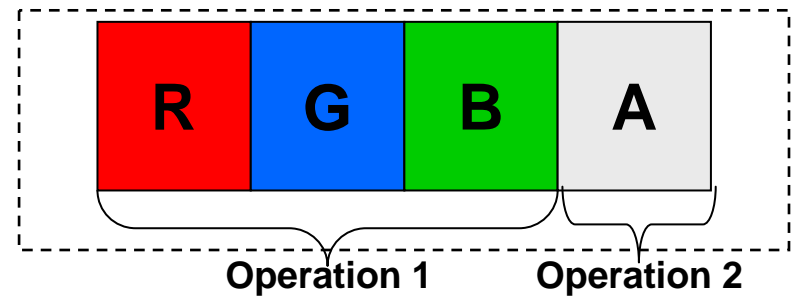


8 Operations / fragment per pass

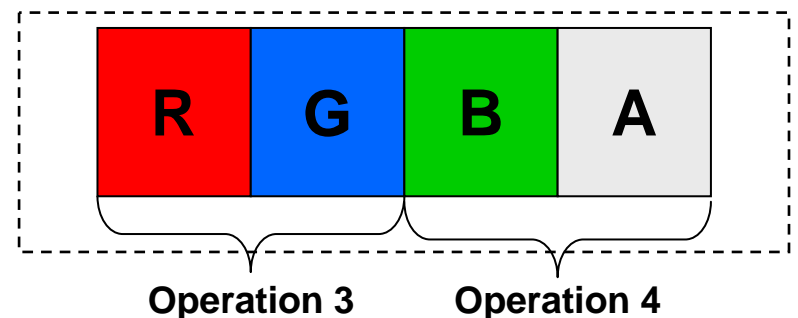
Fragment Shader Component Co-issue

- Use 4 components various ways
 - RGBA all together
 - RGB and A
 - RG and GB
- Both shader units
- Two operations per shader unit

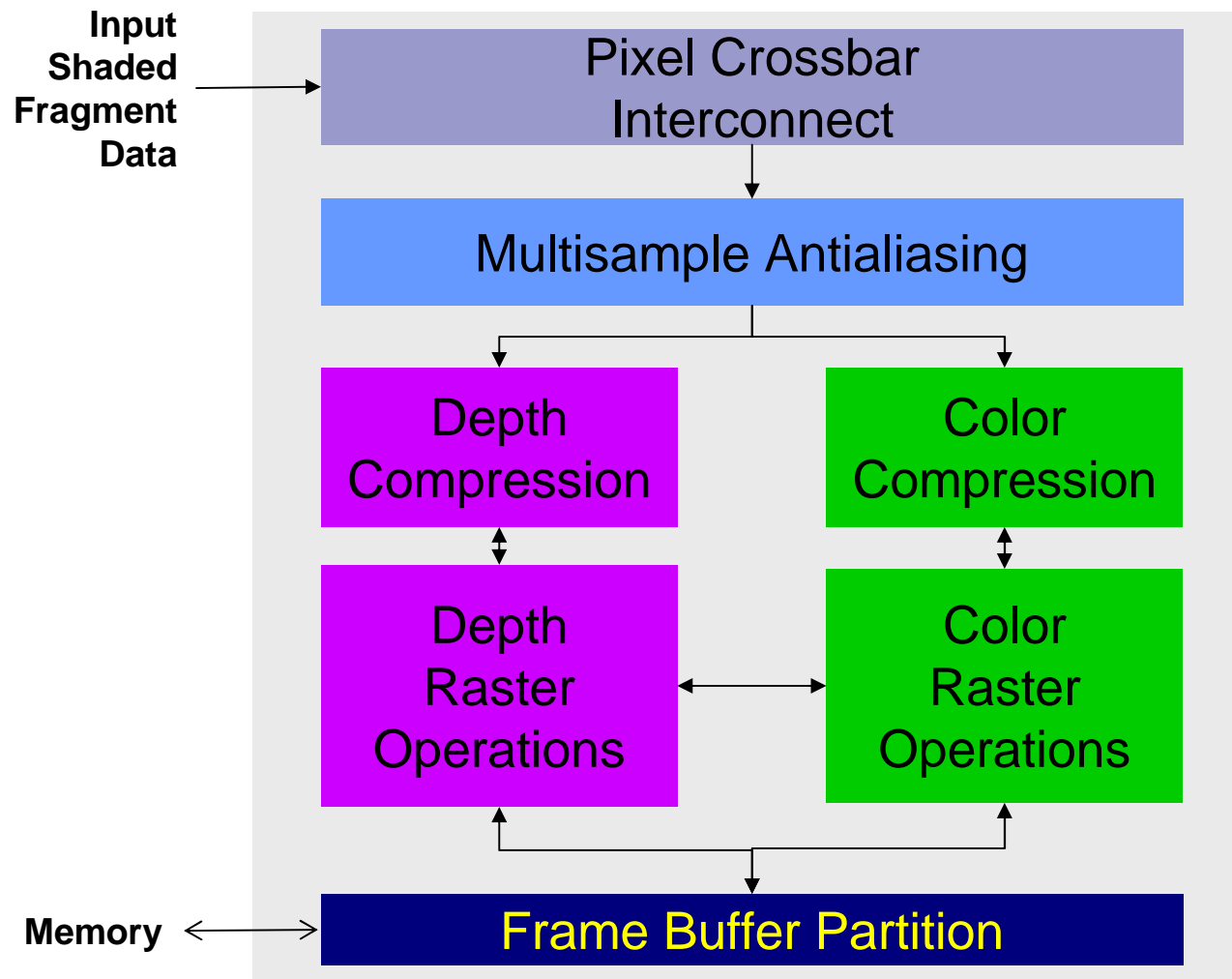
Shader
Unit 1



Shader
Unit 2



Single GeForce 7800 Raster Operations Pipeline



Functionality

- OpenEXR floating-point blending
- sRGB blending
- 4x rotated grid multisampling
- Lossless color and depth compression
- Multiple render targets

GeForce 7800

Transparency Antialiasing



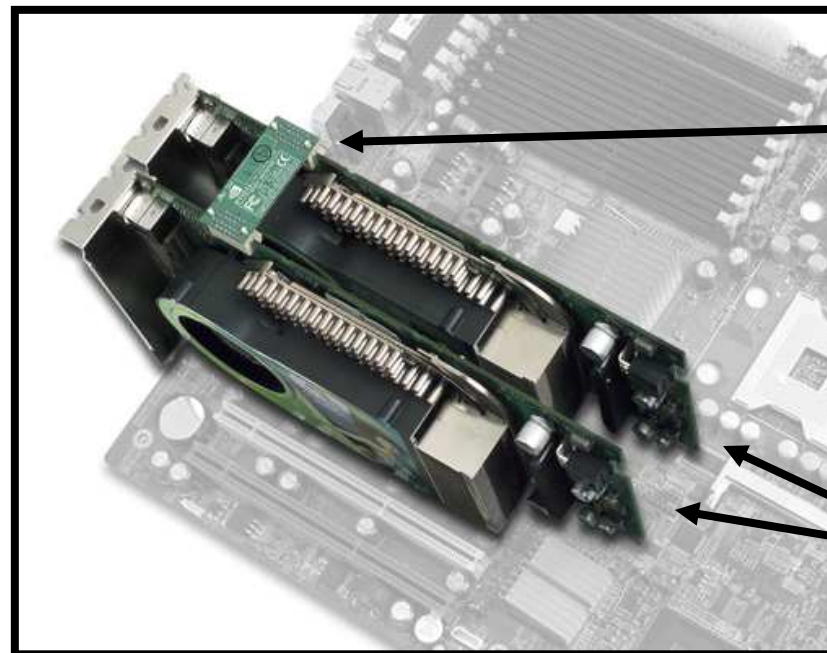
**Conventional 4x antialiasing
with alpha tested context**



**Transparency antialiasing
with alpha tested context**

Scalable Link Interface (SLI)

- Gang two GeForce 6600, 6800, or 7800 graphics boards together
 - Can almost double your performance



SLI
Connector

Two 6800 Ultras
pictured

SLI Rendering Modes

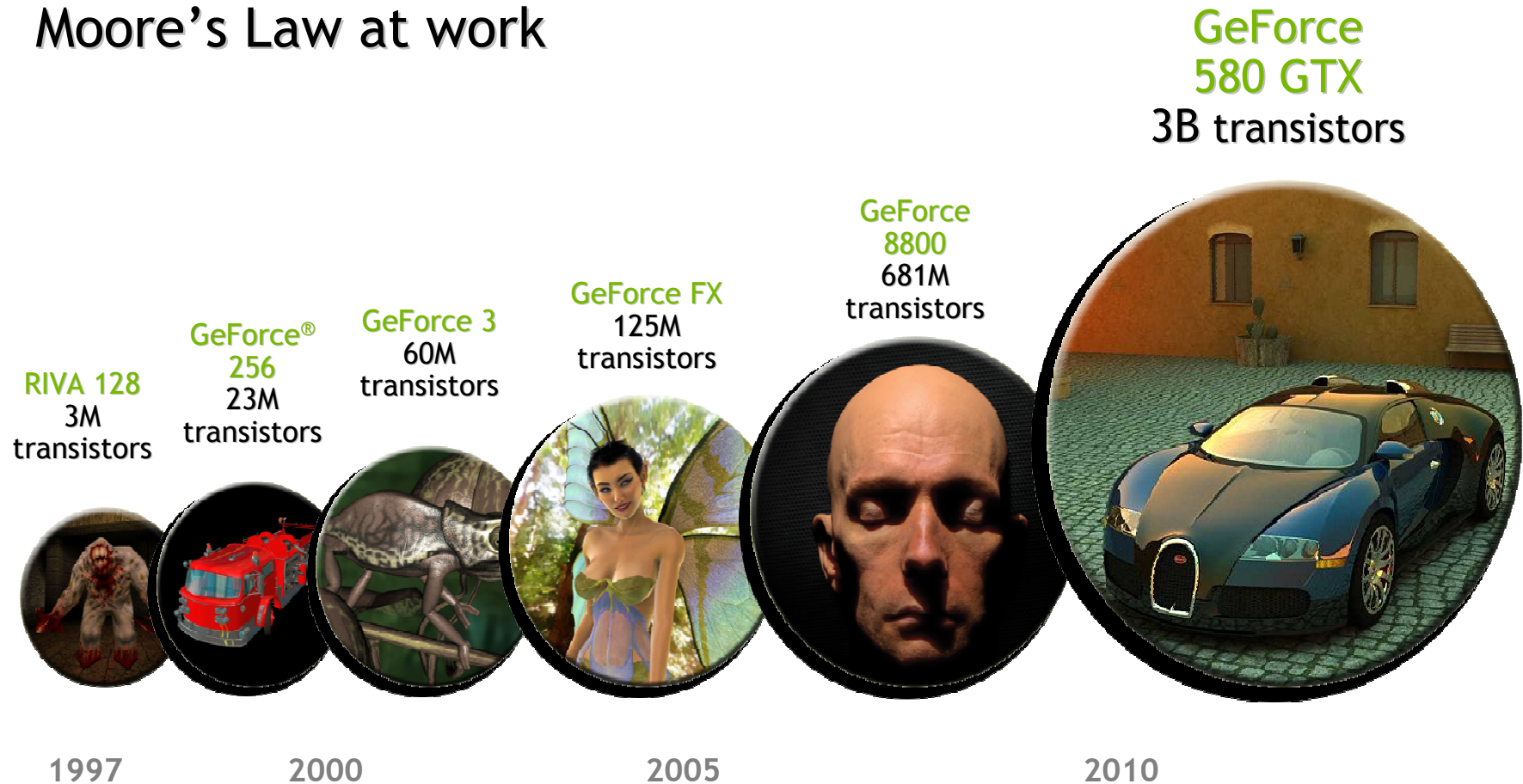
- Split Frame Rendering (SFR)
 - One GPU renders top of screen; other renders the bottom
 - Scales fragment processing but not vertex processing
- Alternate Frame Rendering (AFR)
 - Scales both vertex and fragment processing
 - Adds frame latency
 - Rendering must be free of CPU synchronization
- SLI Antialiasing: SLI8x and SLI16x
 - Better antialiasing quality rather than performance
 - Each card renders with slightly different sub-pixel offset

PC Graphics Hardware Evolution

Viable economics: 650 million GeForce GPUs since 1999

1,000x complexity since 1995

Moore's Law at work



Current High-end “Fermi” GPU

- Current high-end graphics card
- 512 graphics “cores”
- 1.5Gb memory
- System power: 600W
- OpenGL 4.2 / DirectX 11 functionality

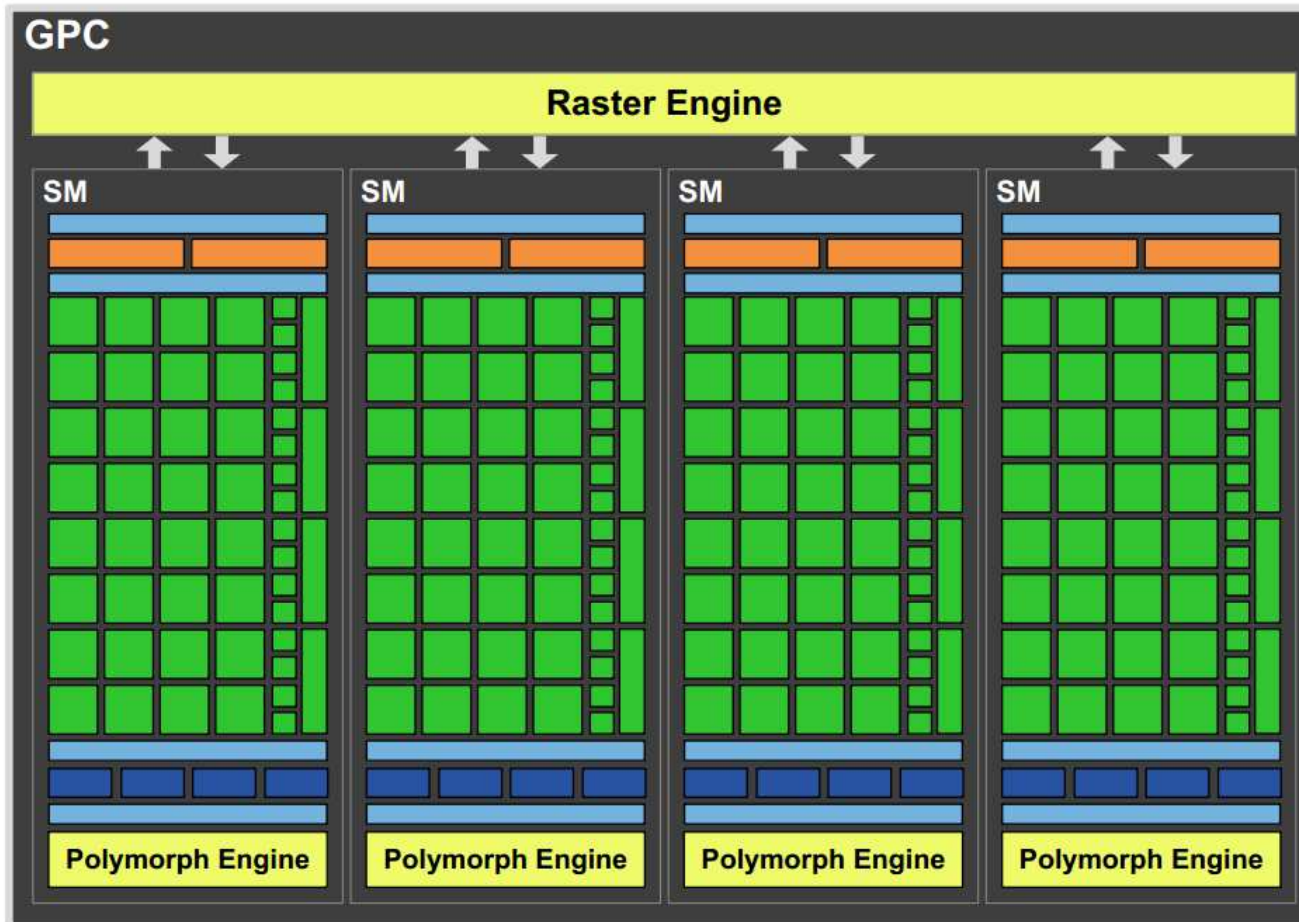


High-level “Fermi” Architecture



- GF100
- Four Graphics Processor Clusters (GPCs)
 - Each is self-contained graphics pipeline
 - Smaller chips have fewer GPCs
- Shared L2 cache
- 6 Memory Controllers
 - 1.5 Gb

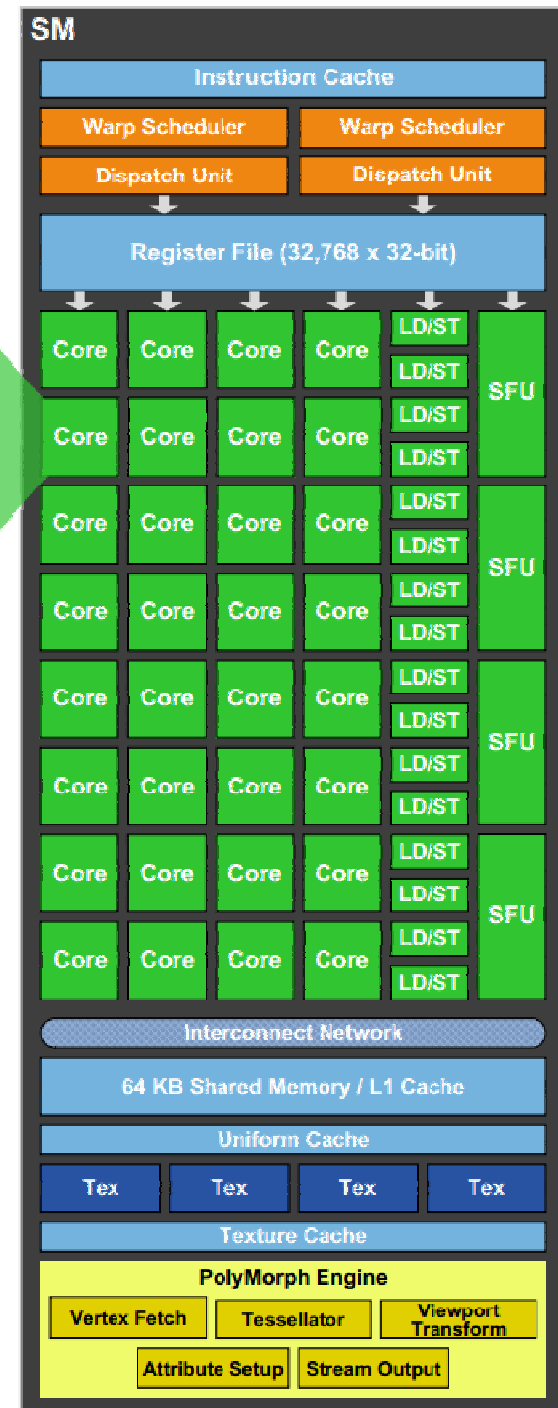
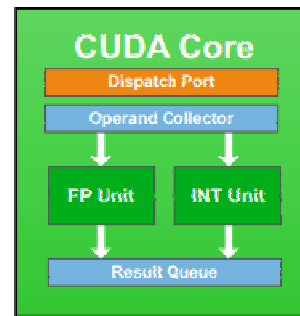
Inside Each Graphics Processing Cluster



- Raster engine
- Four SMs
 - Streaming Multiprocessor
- Texture fetch resources
- Tessellation and vertex processing resources
 - Polymorph Engine

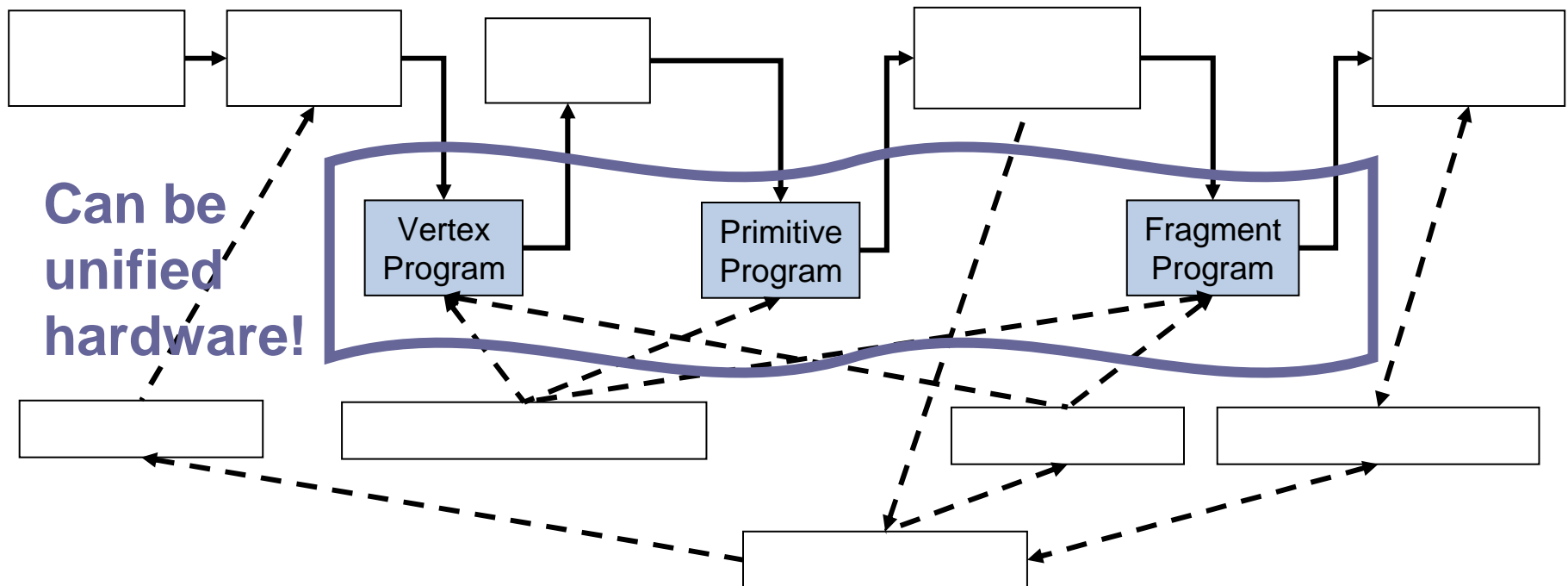
Streaming Multiprocessor (SM)

- Multi-processor execution unit
 - 32 scalar processor cores
 - Warp is a unit of thread execution of up to 32 threads
- Two workloads
 - Graphics
 - Vertex shader
 - Tessellation
 - Geometry shader
 - Fragment shader
 - Compute

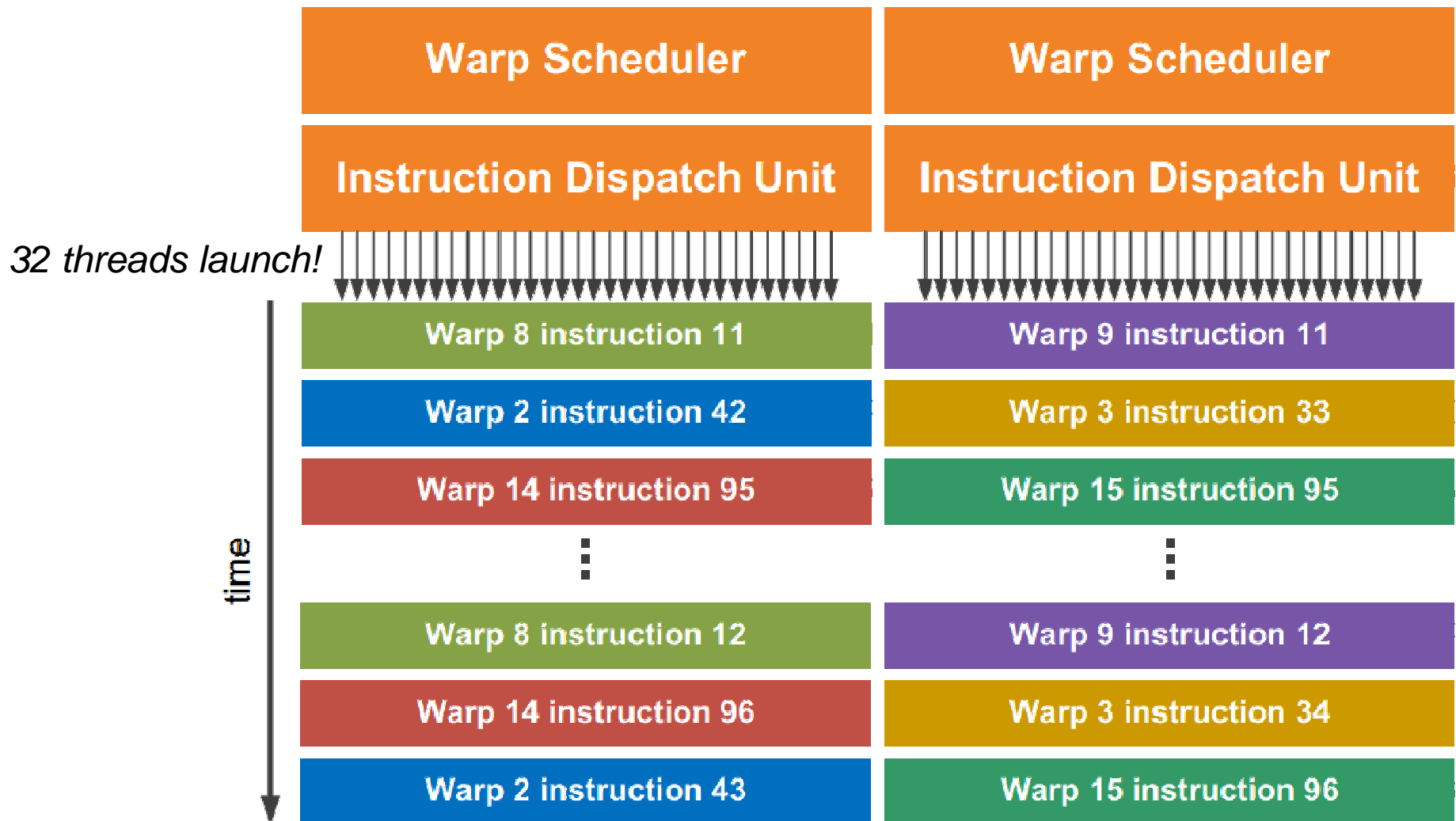


OpenGL Pipeline Programmable Domains run on Unified Hardware

- Unified Streaming Processor Array (SPA) architecture means same capabilities for all domains
 - Plus **tessellation + compute** (*not shown below*)

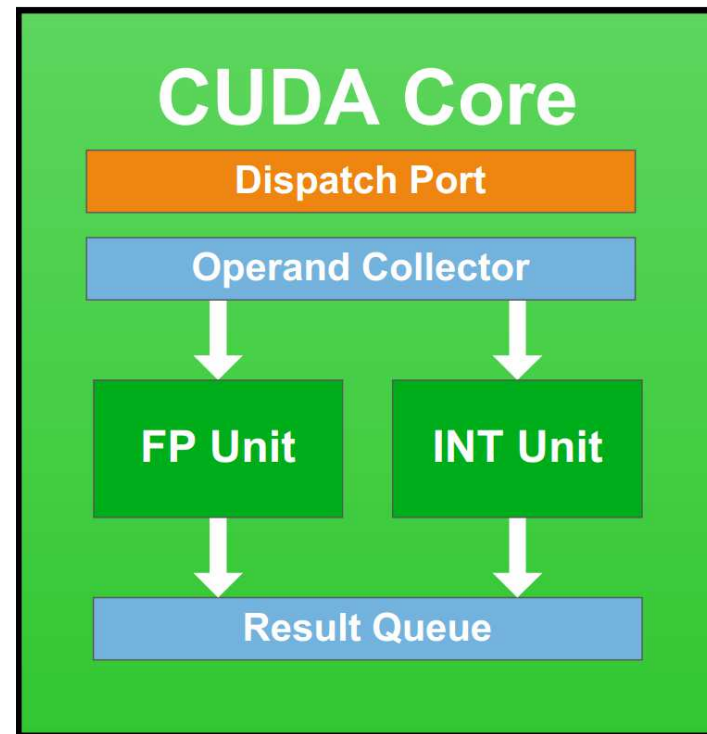


Dual Warp Scheduling



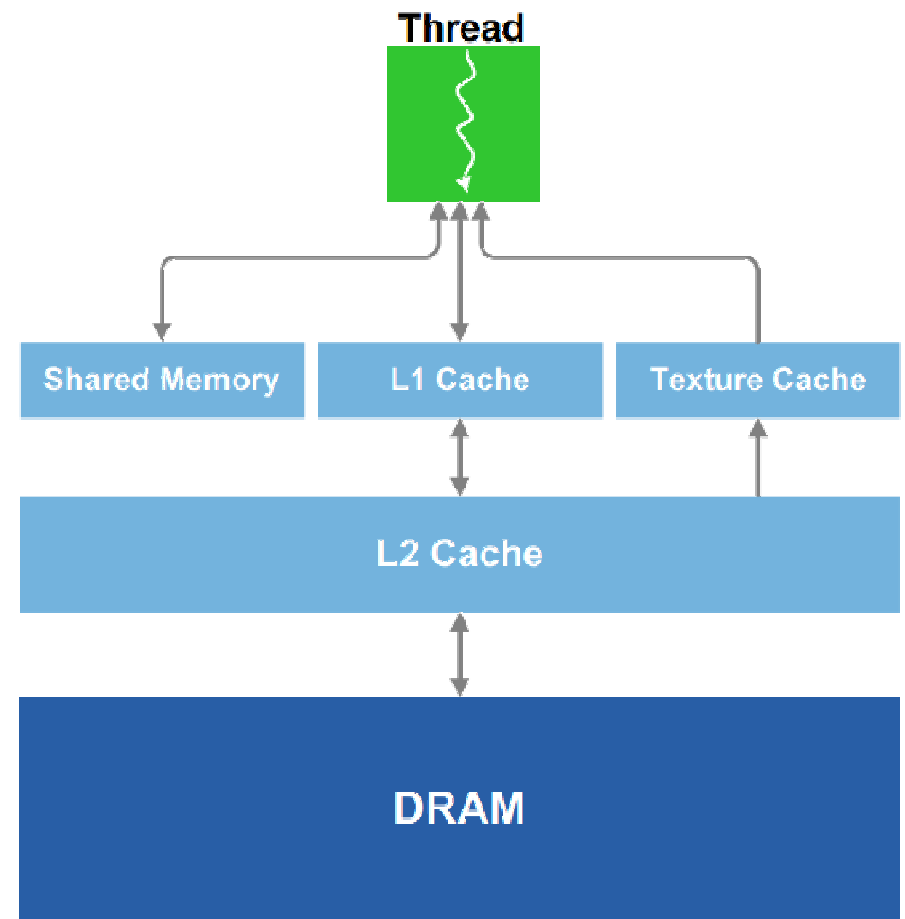
Shader or CUDA Core, Same Unit but Two Personalities

- Execution unit
 - Scalar floating-point
 - Scalar integer



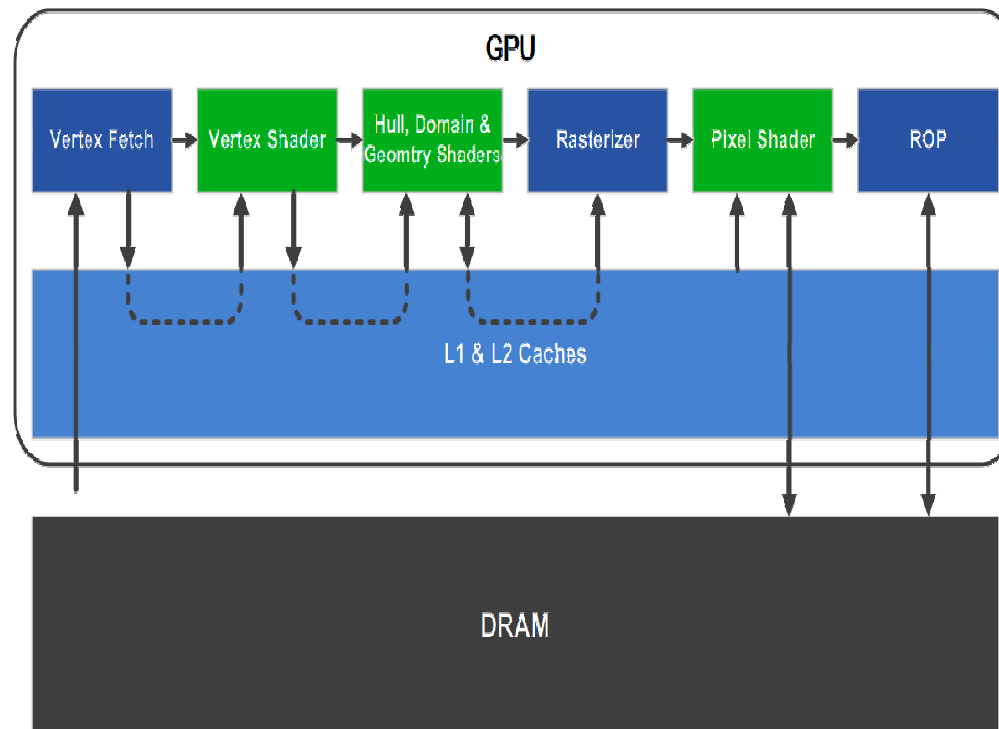
Levels of Caching in Fermi GPU

- 12 KB L1 Texture cache
 - Per texture unit
- SM 64 K cache
 - Split into dedicated 16K or 48K Load/Store cache
 - Shared memory 48K or 16K
- L2 unifies texture cache, raster operation cache, and internal buffering in prior generation
 - 768 K
 - Read / write
 - Fully coherent



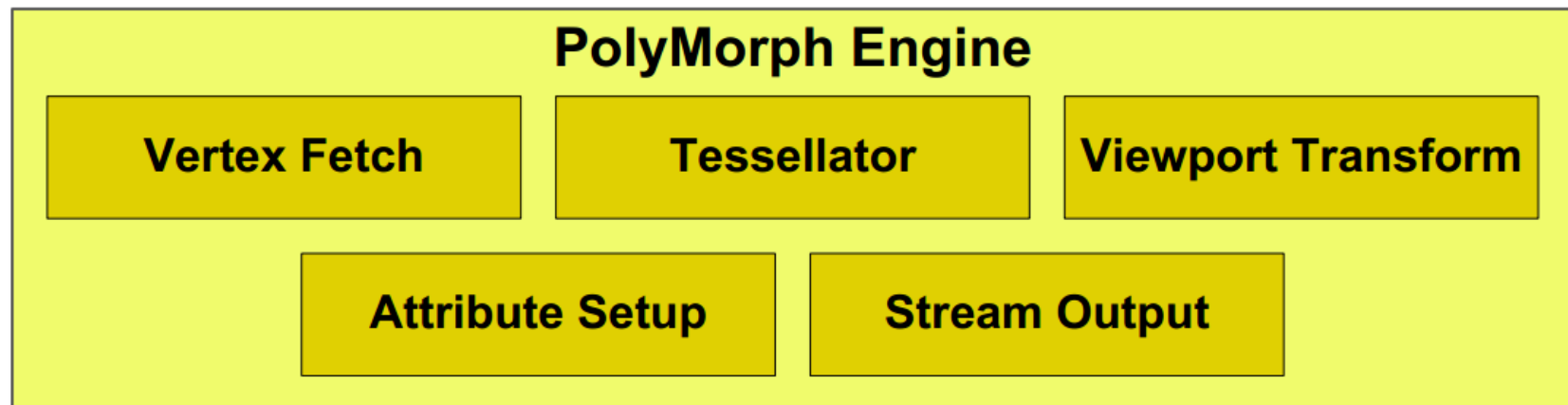
Cache Use Strategies in Fermi GPU

- Pipeline stages can communicate efficiently through GPU's L1 and L2 caches
 - Buffering between stages stays all on chip
 - Only vertex, texel, and pixel read/writes need to go to DRAM



Vertex and Tessellation Processing Tasks

- Fixed-function graphics engines
 - Pull attributes and assemble vertex
 - Manage tessellation control and domain shader evaluation
 - Viewport transform
 - Attribute setup of plane equations for rasterization
 - Stream out vertices into buffers



Rasterization Tasks

- Turns primitives into fragments
 - Computes edge equations
 - Two-stage rasterization
 - Coarse raster finds tiles the primitive could be in
 - Fine raster evaluates sample positions within tiles
 - Zcull efficiently eliminates occluded fragments

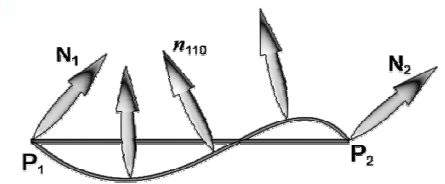
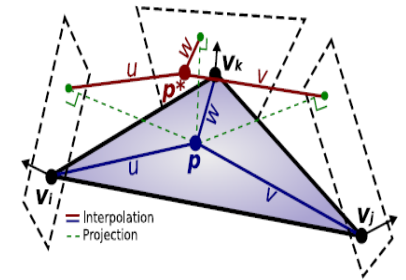
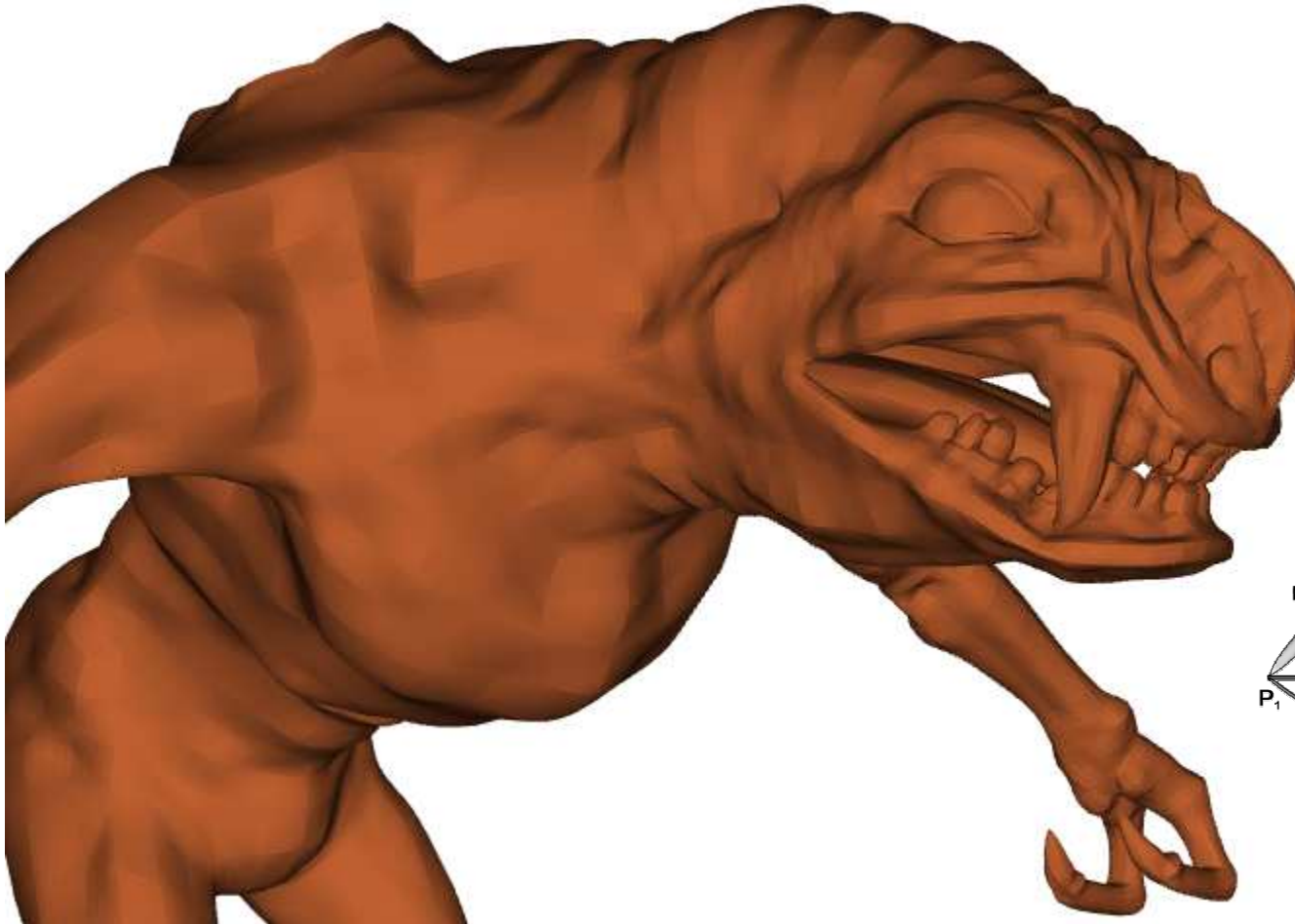


Base Input Mesh



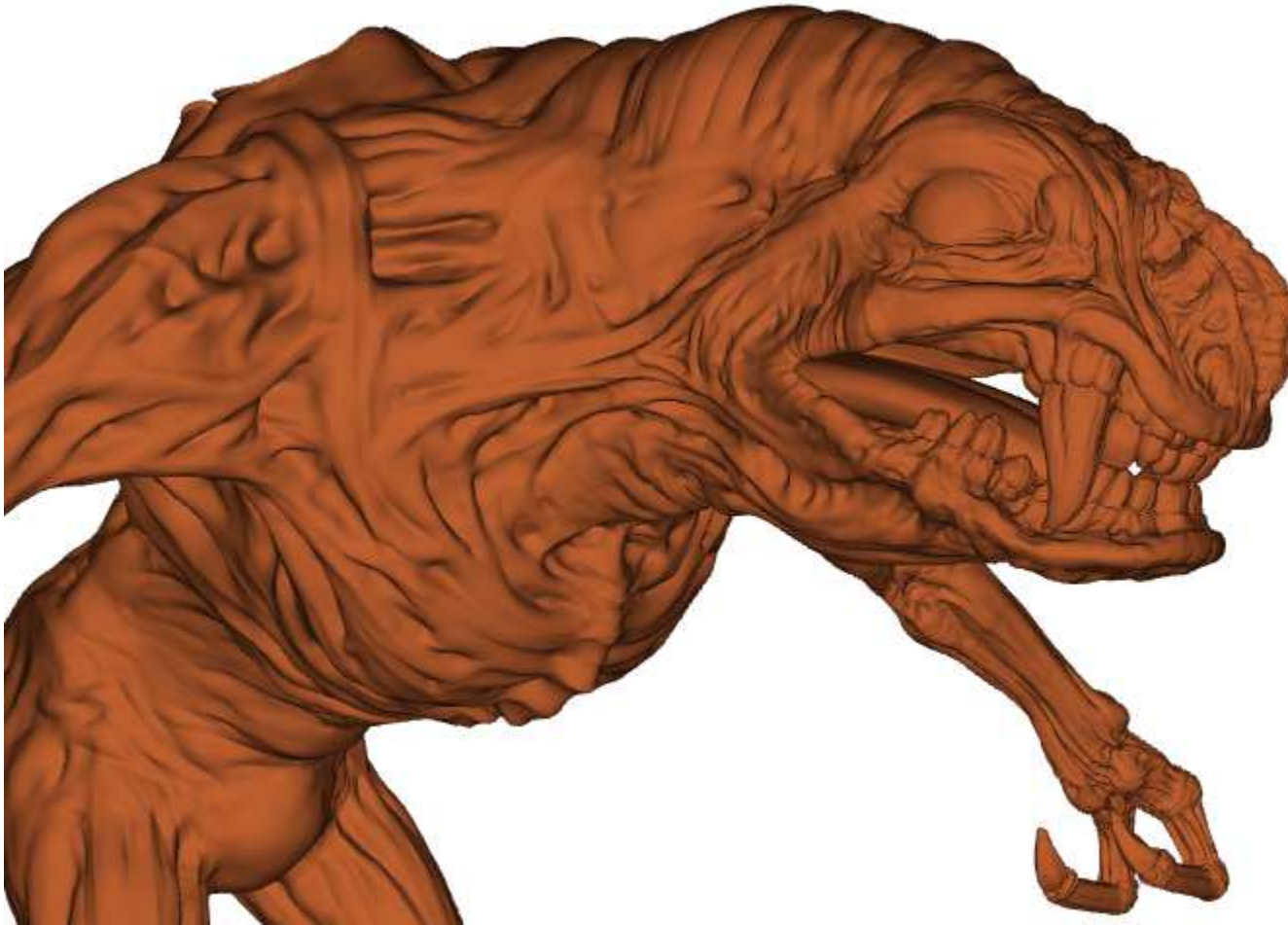
From *Metro 2033*, © THQ and 4A Games

Apply Phong Tessellation



From *Metro 2033*, © THQ and 4A Games

Add Displacement Mapping



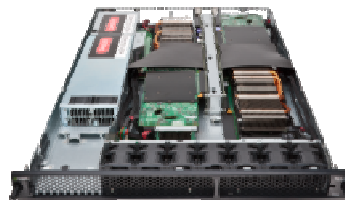
From *Metro 2033*, © THQ and 4A Games

GPUs as Compute Nodes

- Architecture of GPU has evolved into a high-performance, high-bandwidth compute node



**Small form factor
Compute**



**Integrated CPU-GPU
Servers & Blades**



**OEM CPU Server +
Compute 1U**



**Workstations
2 to 4 Tesla
GPUs**

Compute Programming Model

- Cooperative Thread Array (CTA)
 - Single Program, Multiple Data
 - Organized in 1D, 2D, or 3D
- Programming APIs
 - CUDA, OpenCL, DirectCompute
 - APIs + language = parallel processing system
 - OpenGL or Direct3D through shaders
 - Cg, HLSL, GLSL

Now in World's Fastest Supercomputers

Tianhe-1A

2.507 Petaflop

**7168 Tesla M2050
GPUs**

**National Supercomputing Center
in Tianjin**



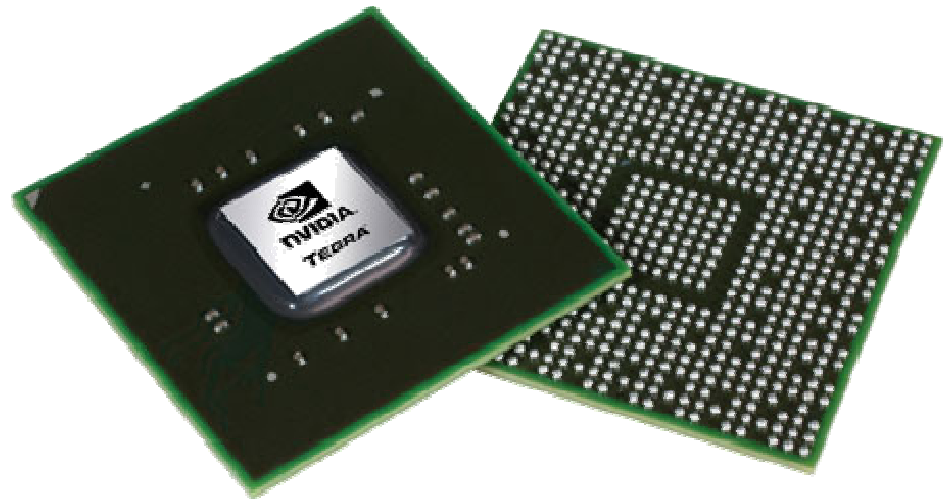
Opposite direction: Consumer mobile devices



Low-power Mobile System on a Chip (SoC)

Complete system on a chip

- 4 ARM cores
- Integrated graphics
 - OpenGL ES 2.0
- Power <1W



Mid-term Next Class

■ Mid-term

- ☐ Similar in format to the homeworks
- ☐ 15% of your final grade
- ☐ Arrive on time
- ☐ Open textbook. Open notes, including lecture slides.
- ☐ Calculators allowed/encouraged.
- ☐ No smart phones, no computers, no Internet access.
- ☐ Show your work to justify your answer and provide a basis for partial credit.

■ What to study

- ☐ All material in lecture slides
- ☐ Review in-class quiz questions
- ☐ Study homeworks
- ☐ Responsible for textbook readings

■ Have a relaxing spring break

- ☐ Next lecture: Shadows
- ☐ Come back to Project 2