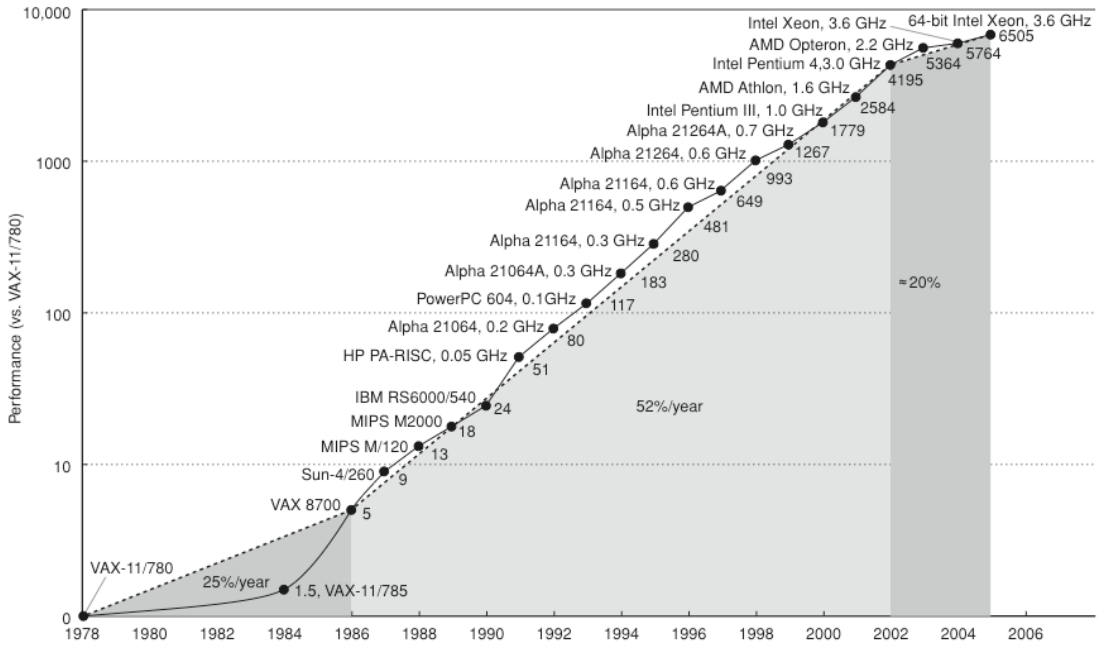
Experiences with GPU Computing

John Owens
Assistant Professor, Electrical and Computer Engineering
Institute for Data Analysis and Visualization
University of California, Davis

The Right-Hand Turn

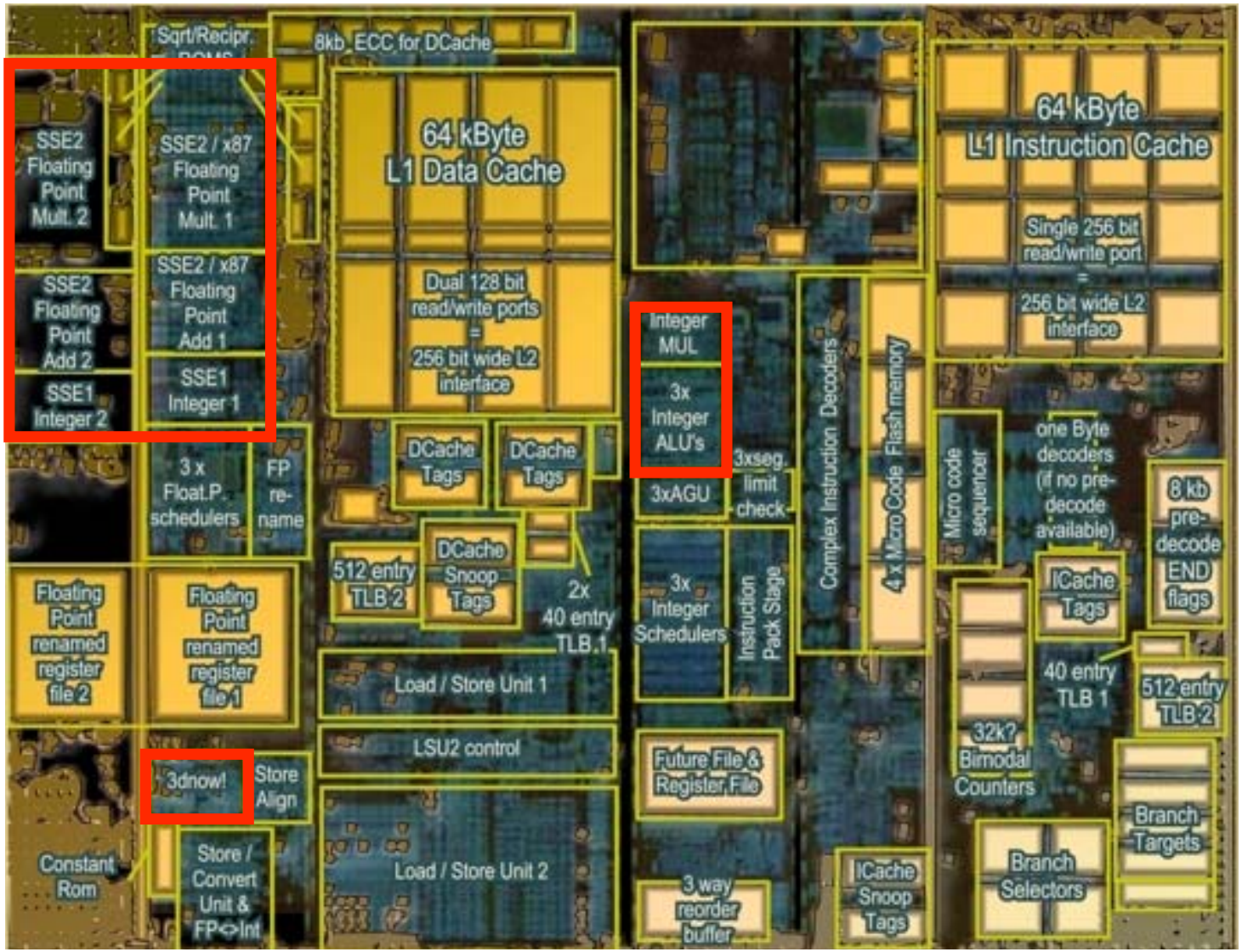


[H&P Figure 1.1]

Why?

- ILP increasingly difficult to extract from instruction stream
- Control hardware dominates μprocessors
  - Complex, difficult to build and verify
  - Takes substantial fraction of die
  - Scales poorly
    - Pay for max throughput, sustain average throughput
    - Quadratic dependency checking
  - Control hardware doesn't do any math!

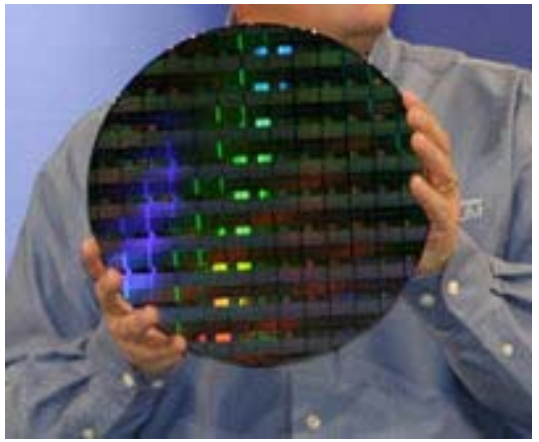
AMD “Deerhound” (K8L)



chip-architect.com

Go Parallel

- Time of architectural innovation
- Major CPU vendors supporting multicore
- Interest in general-purpose programmability on GPUs
- Universities must teach thinking in parallel



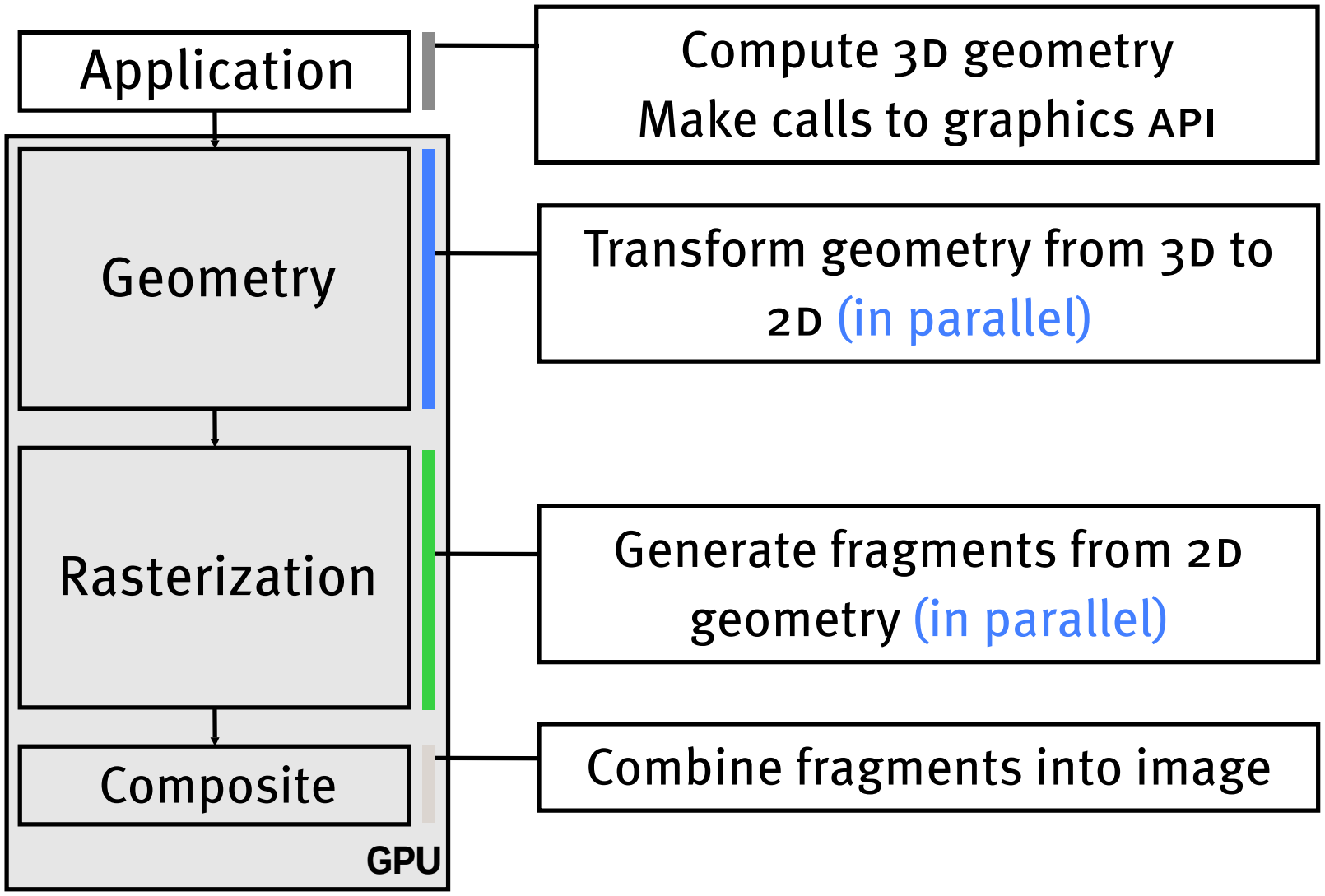
Outline

- The graphics pipeline
- Graphics programmability begets GPGPU
- NVIDIA's CUDA GPGPU environment
- Our lessons from GPGPU
- Motivating future architectures
- Concluding thoughts

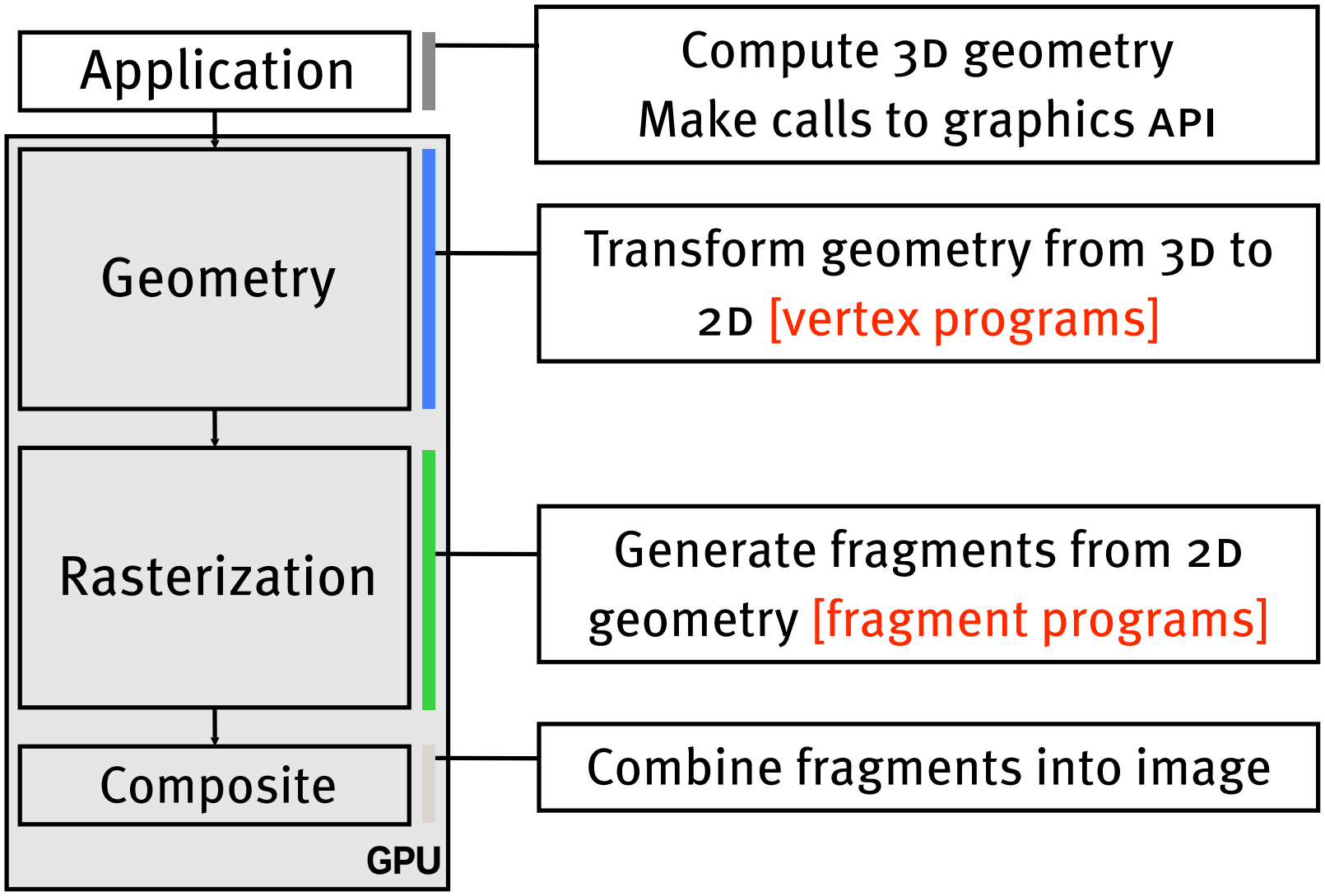
What's Different about the GPU?

- The future of the desktop is parallel
  - We just don't know what kind of parallel
- GPUs and multicore are different
  - Multicore: Coarse, heavyweight threads, better performance per thread
  - GPUs: Fine, lightweight threads, single-thread performance is poor
- A case for the GPU
  - Interaction with the world is visual
  - GPUs have a well-established programming model
  - NVIDIA shipped 100M units last year, Intel even more, 500M+ total/year

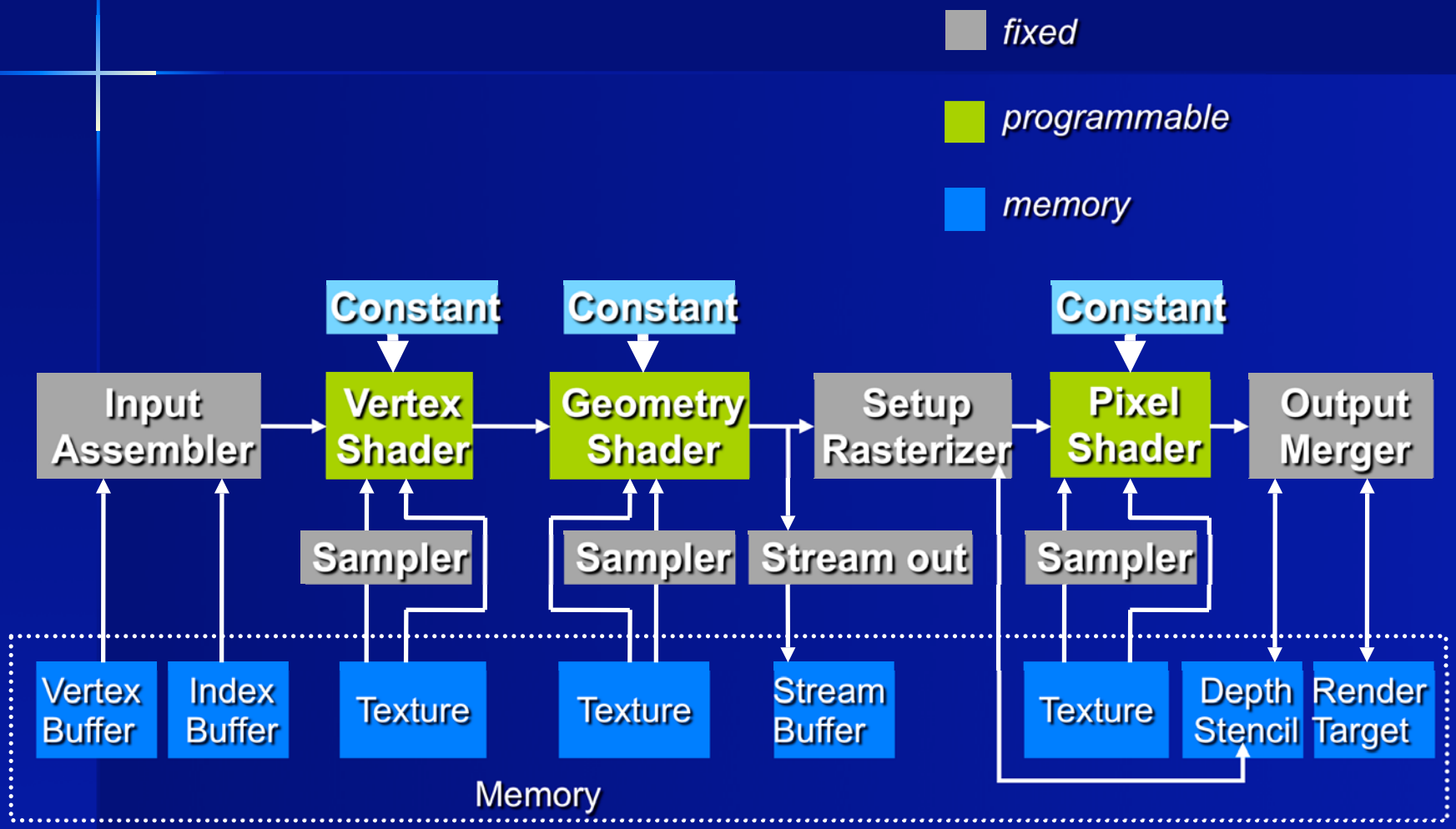
The Rendering Pipeline



The Programmable Pipeline



DirectX 10 Pipeline

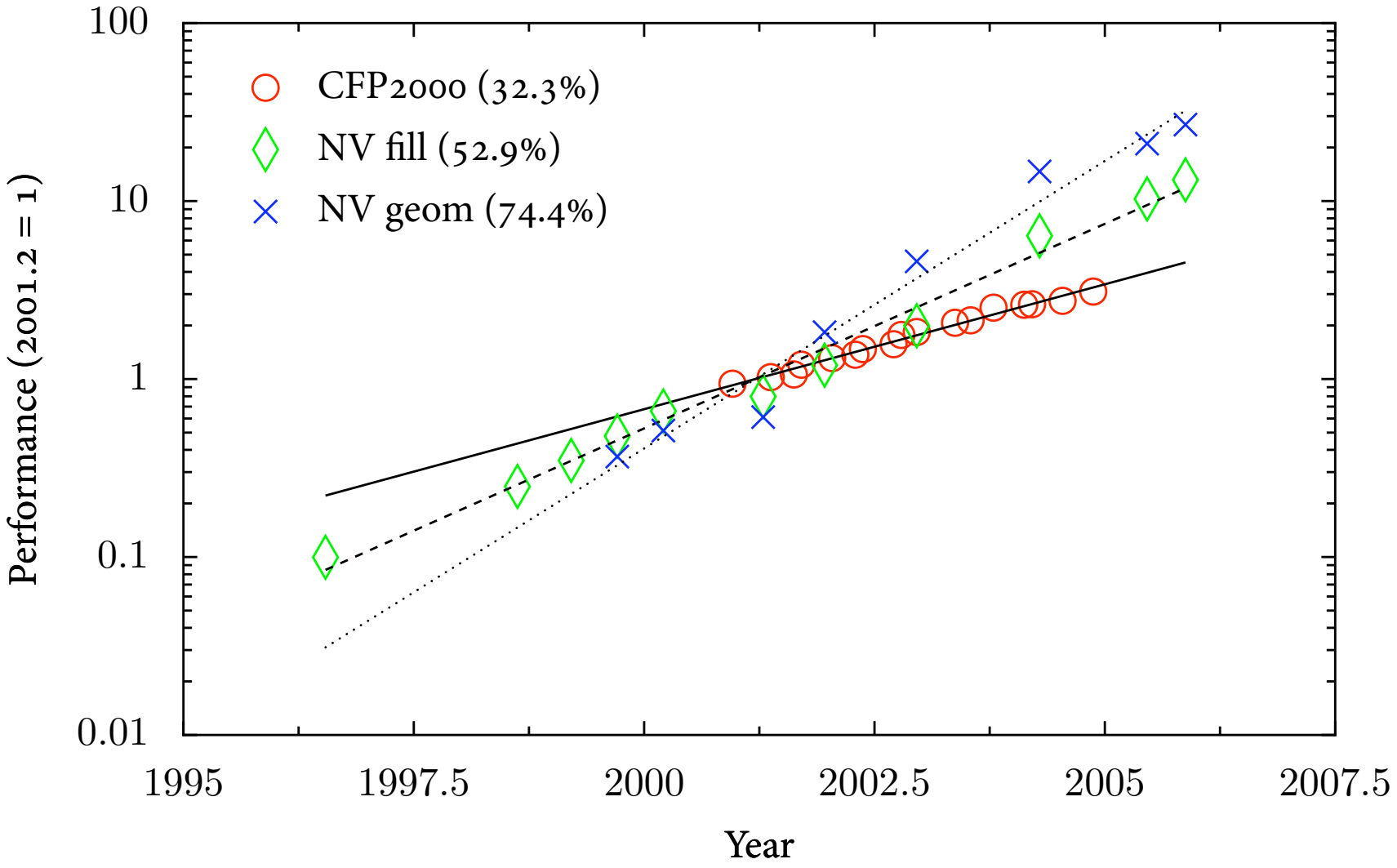


Courtesy David Blythe, Microsoft

Characteristics of Graphics

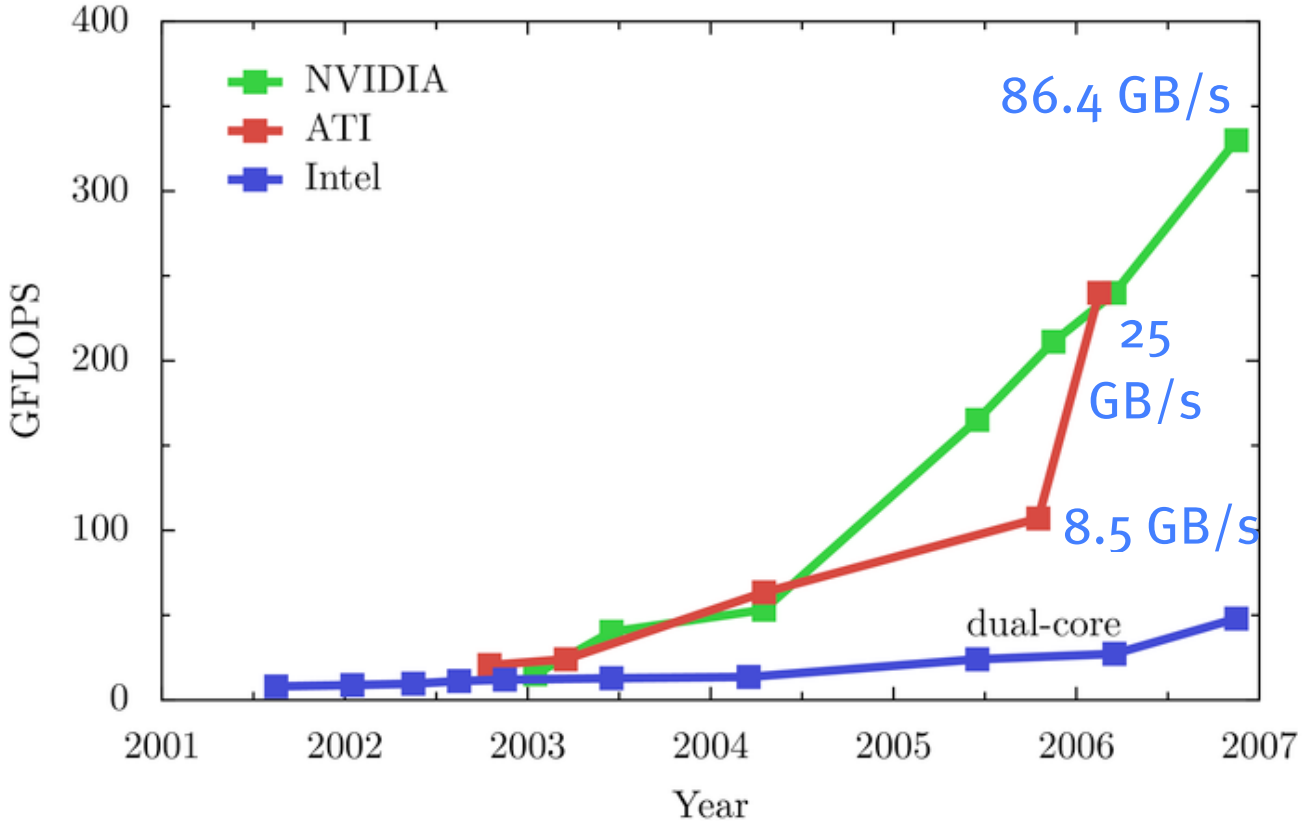
- Large computational requirements
- Massive parallelism
  - Graphics pipeline designed for independent operations
- Long latencies tolerable
- Deep, feed-forward pipelines
- Hacks are OK—can tolerate lack of accuracy
- GPUs are good at *parallel*, *arithmetically intense*, *streaming-memory* problems

Long-Term Trend: CPU vs. GPU



Recent GPU Performance Trends

Programmable 32-bit FP operations per second



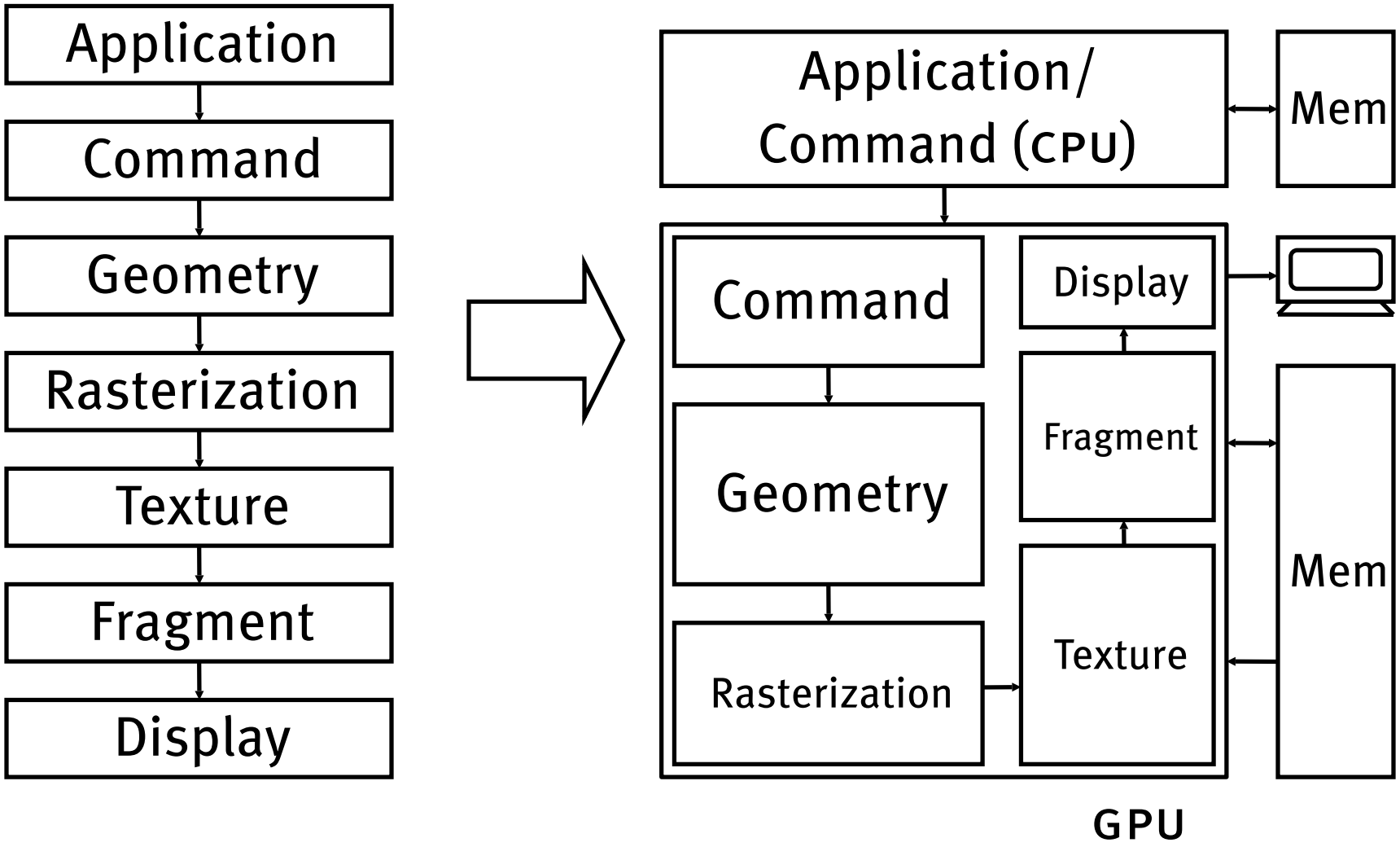
$548
8800GTX

$195
X1950

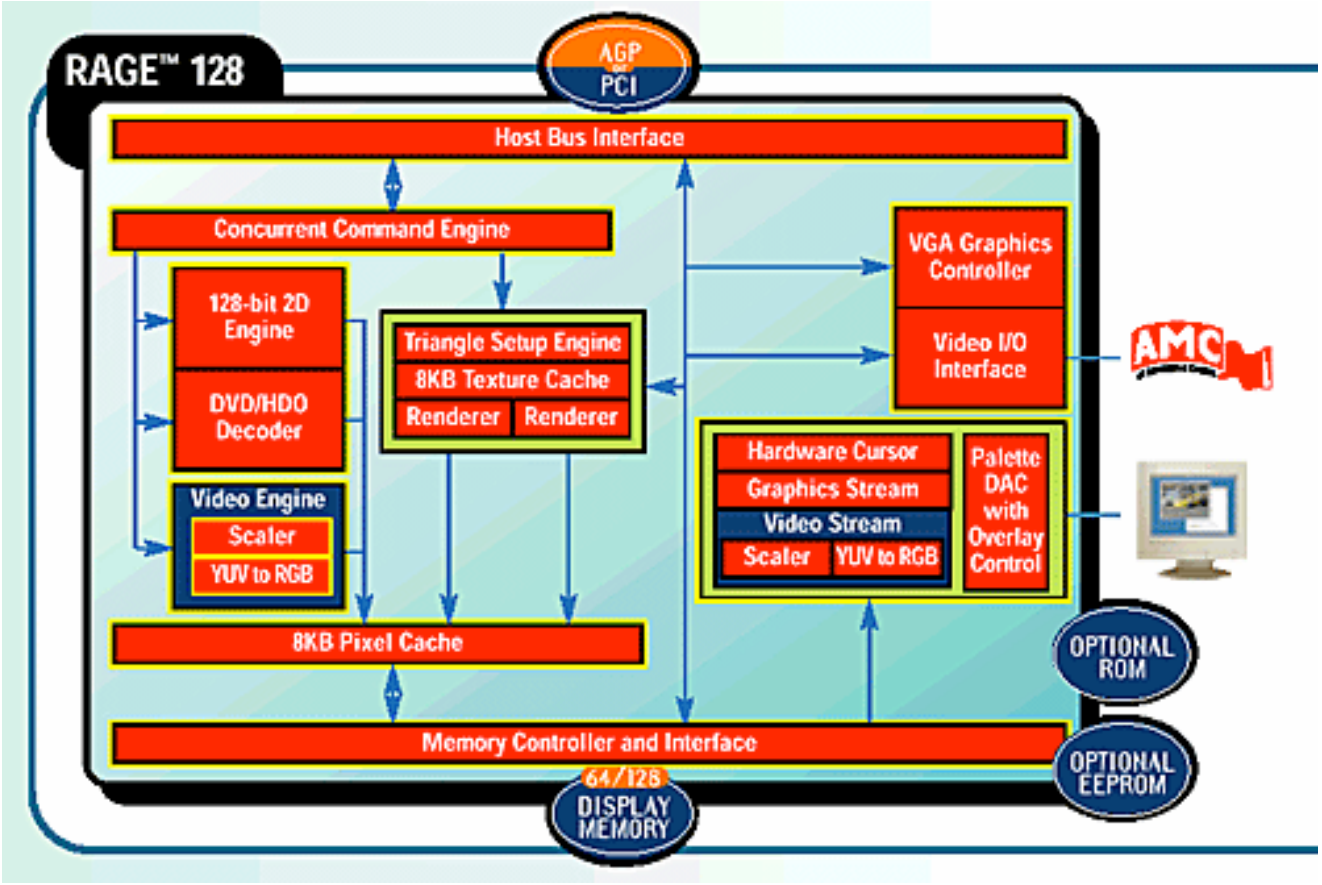
$240
P4X3.6

Early data courtesy Ian Buck; from Owens et al. 2007 [CGF]

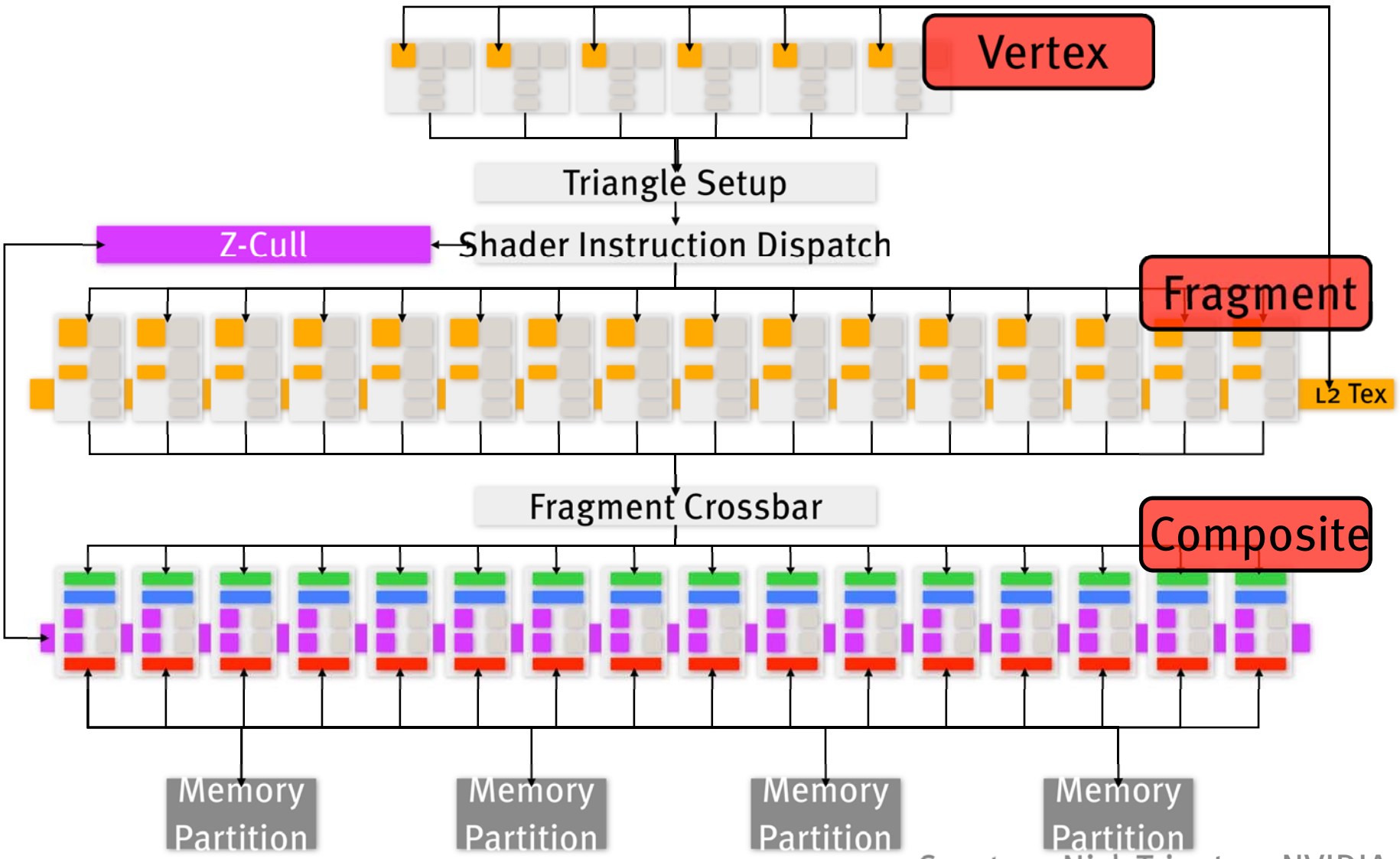
Graphics Hardware—Task Parallel



Rage 128

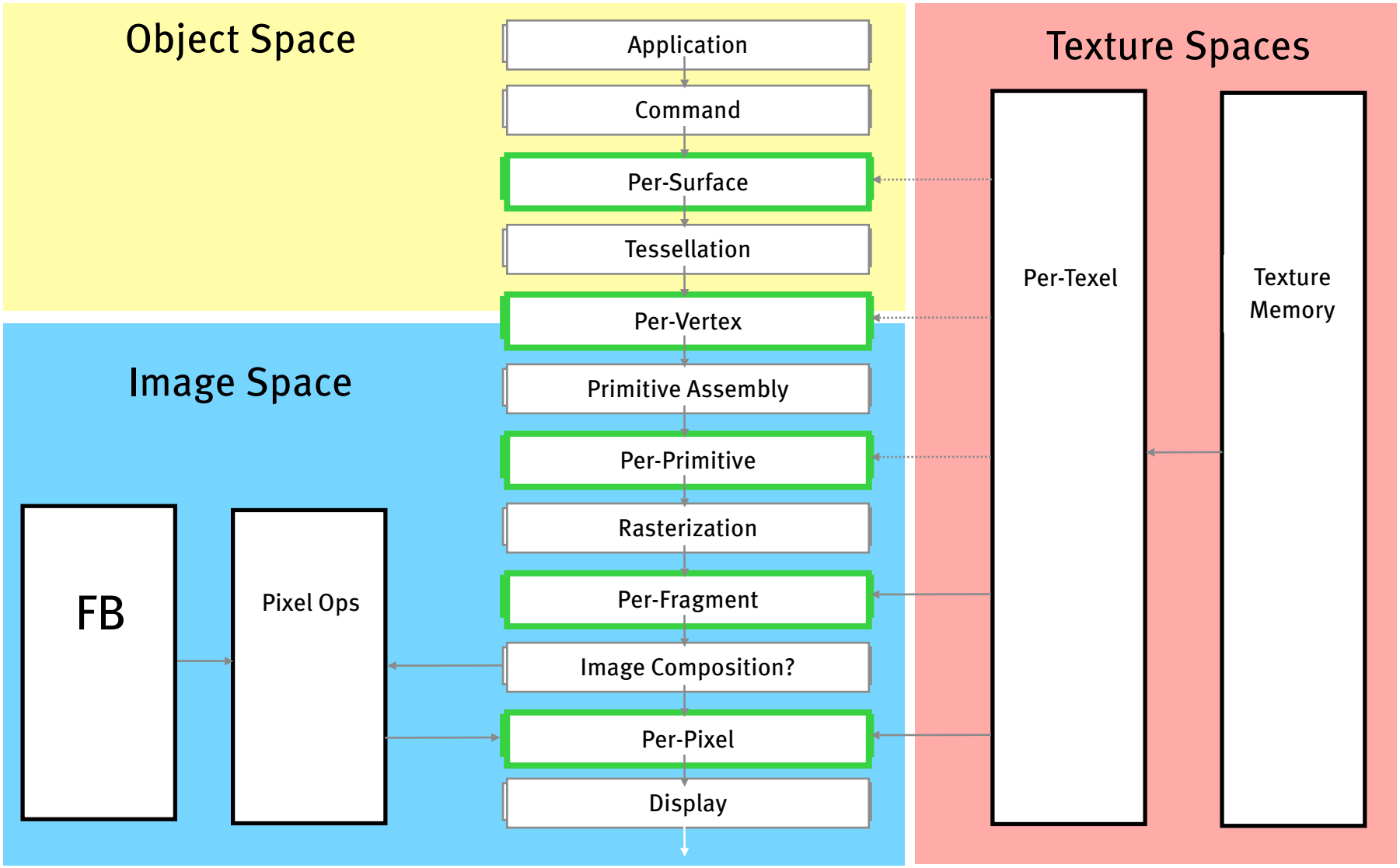

RAGE™ 128
AGP or PCI
Host Bus Interface
Concurrent Command Engine
128-bit 2D Engine
DVD/HDO Decoder
Video Engine
Scaler
YUV to RGB
Triangle Setup Engine
8KB Texture Cache
Renderer
Renderer
8KB Pixel Cache
VGA Graphics Controller
Video I/O Interface
Hardware Cursor
Graphics Stream
Video Stream
Scaler
YUV to RGB
Palette DAC with Overlay Control
AMC
OPTIONAL ROM
OPTIONAL EEPROM
Memory Controller and Interface
64/128
DISPLAY MEMORY

NVIDIA GeForce 6800 3D Pipeline



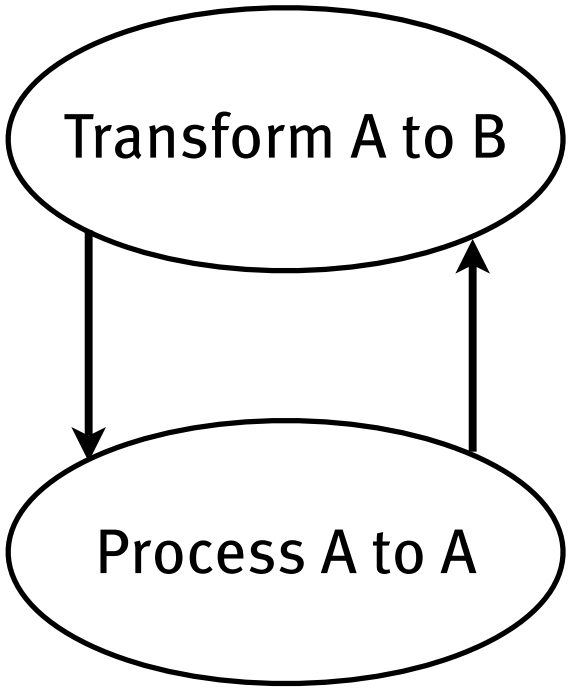
Courtesy Nick Triantos, NVIDIA

Programmable Pipeline



[From Akeley and Hanrahan, Real-Time Graphics Architectures]

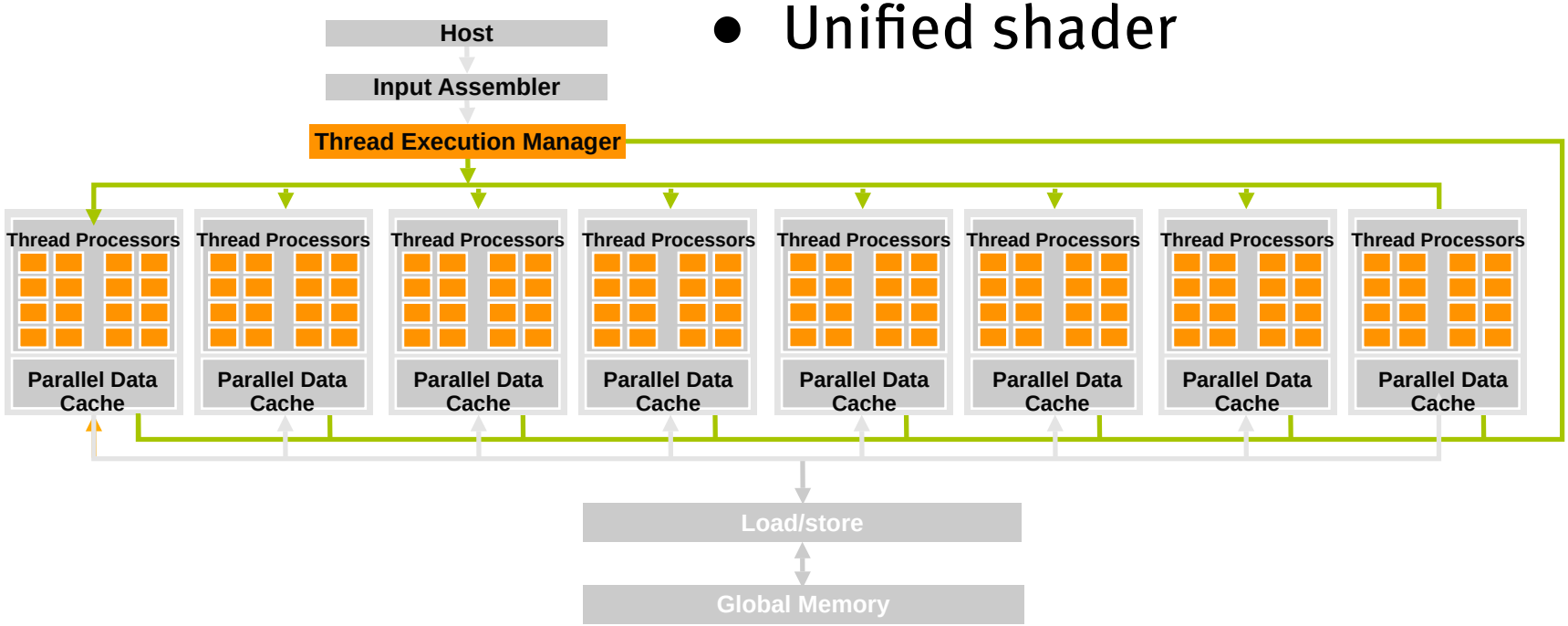
Generalizing the Pipeline



- Transform A to B
  - Ex: Rasterization (triangles to fragments)
  - Historically fixed function
- Process A to A
  - Ex: Fragment program
  - Recently programmable

GeForce 8800 GPU

- Built around programmable units
- Unified shader



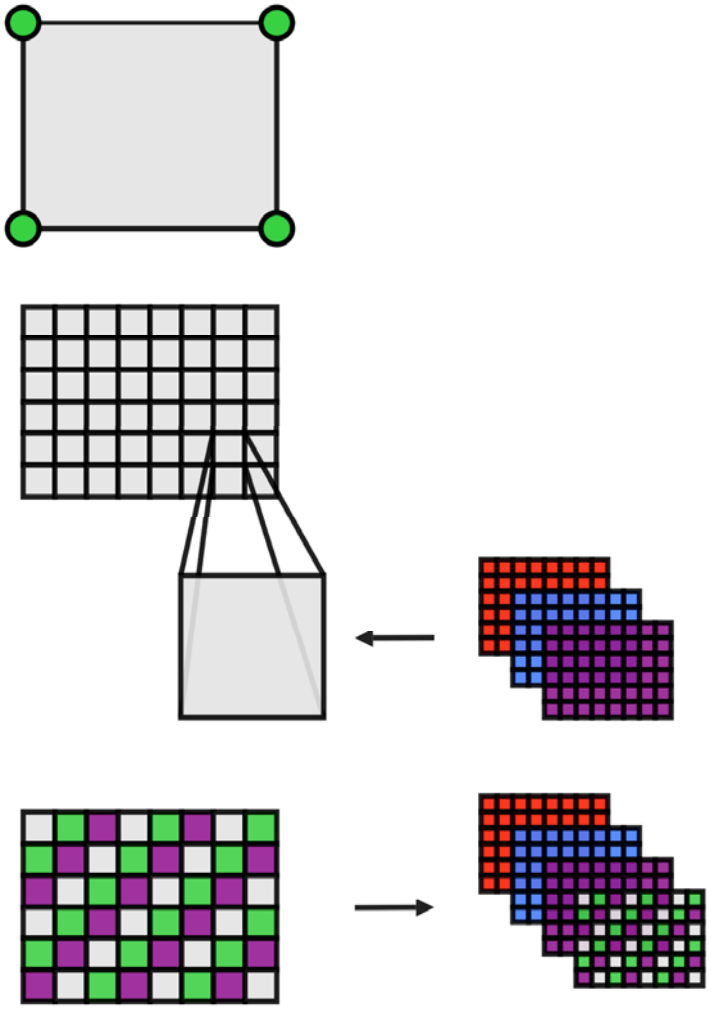
[courtesy of Ian Buck, NVIDIA]

Towards Programmable Graphics

- Fixed function
  - Configurable, but not programmable
- Programmable shading
  - Shader-centric
  - Programmable shaders, but fixed pipeline
- Programmable graphics
  - Customize the *pipeline*
  - Neoptica asserts the major obstacle is programming models and tools

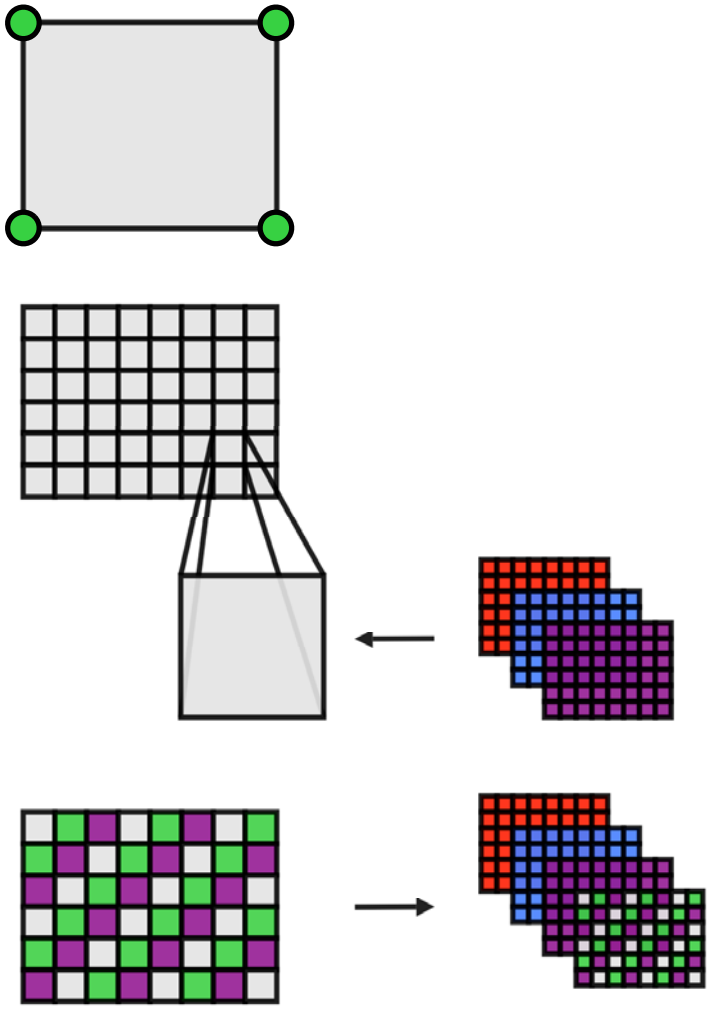
http://www.neoptica.com/NeopticaWhitepaper.pdf
http://www.graphicshardware.org/previous/www_2006/presentations/pharr-keynote-gh06.pdf

Programming a GPU for Graphics



- Application specifies geometry -› rasterized
- Each fragment is shaded w/ SIMD program
- Shading can use values from texture memory
- Image can be used as texture on future passes

Programming a GPU for GP Programs

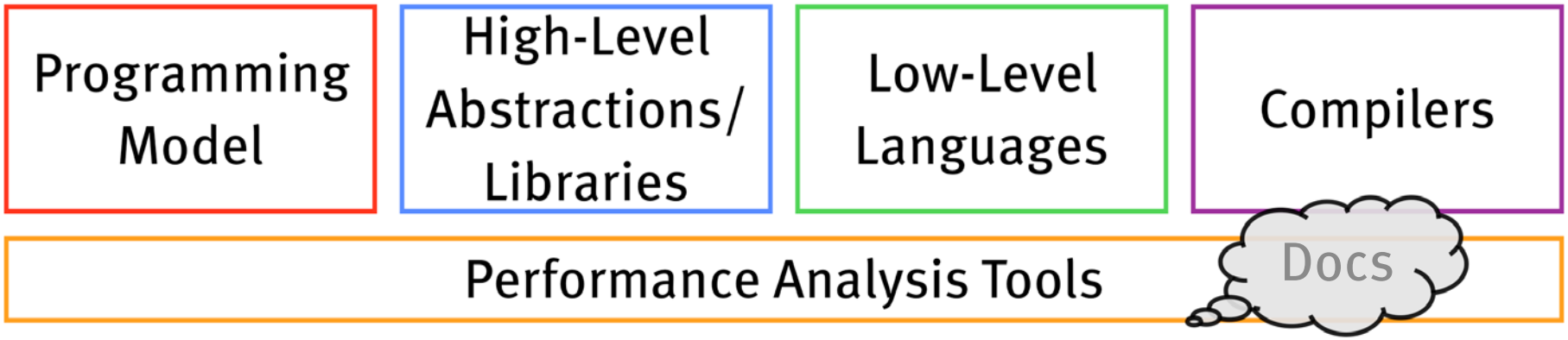


- Draw a screen-sized quad
- Run a SIMD program over each fragment
- “Gather” is permitted from texture memory
- Resulting buffer can be treated as texture on next pass

3 Generations of GPGPU

- Making it work at all
  - Functionality and tools extremely primitive
  - Comparisons not rigorous
- Making it work better
  - Functionality improving
  - Better understanding of what we do well/poorly
  - Solid comparisons
  - Software development cycle still primitive—“horizontal” model of development
- Doing it right
  - How do we build stable, portable, modular building blocks and applications?

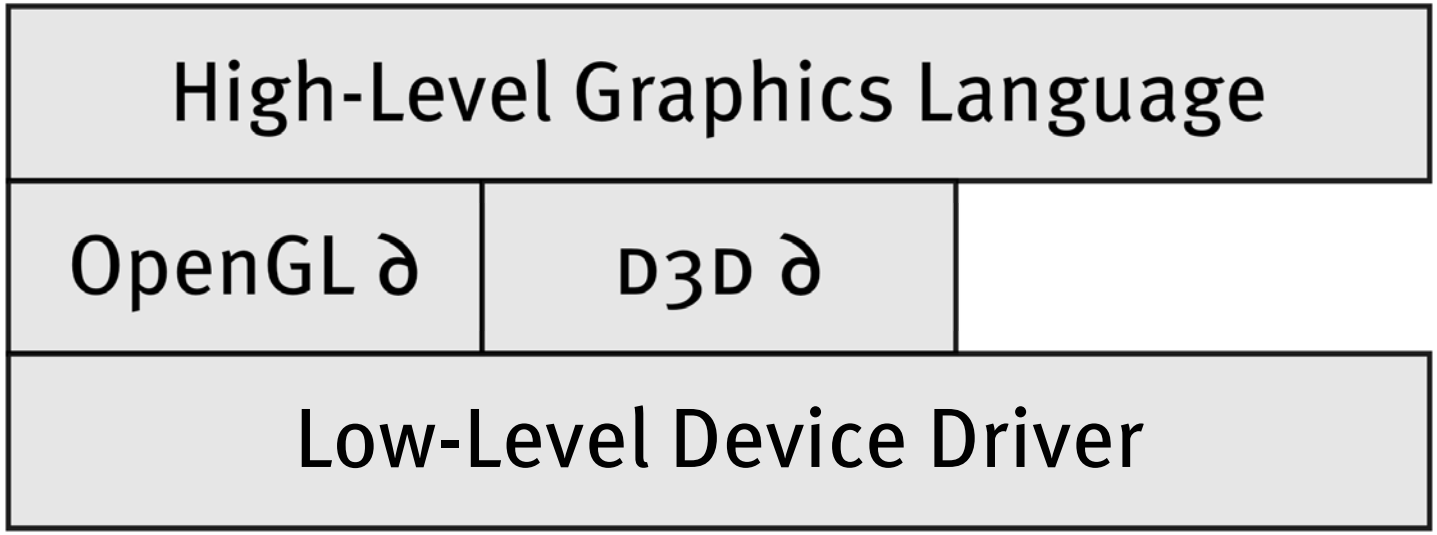
Challenge: Programming Systems



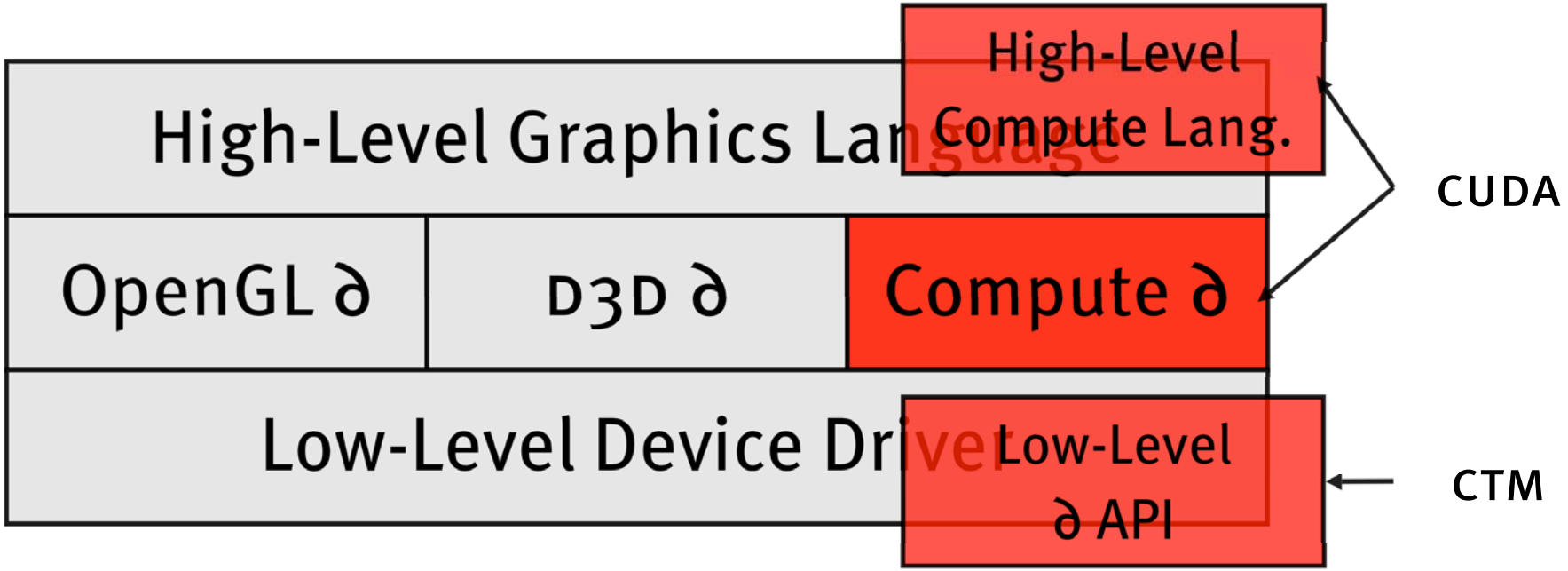
- CPU
- Scalar
- STL, GNU SL, MPI, …
- C, Fortran, …
- gcc, vendor-specific, …
- gdb, vtune, Purify, …
- Lots
- … applications

- GPU
- Stream? Data-Parallel?
- Brook, Scout, sh, Glift -› PS, RM
- GLSL, Cg, HLSL, CUDA/CTM …
- Vendor-specific
- Shadesmith, NVPerfHUD
- None
- … kernels

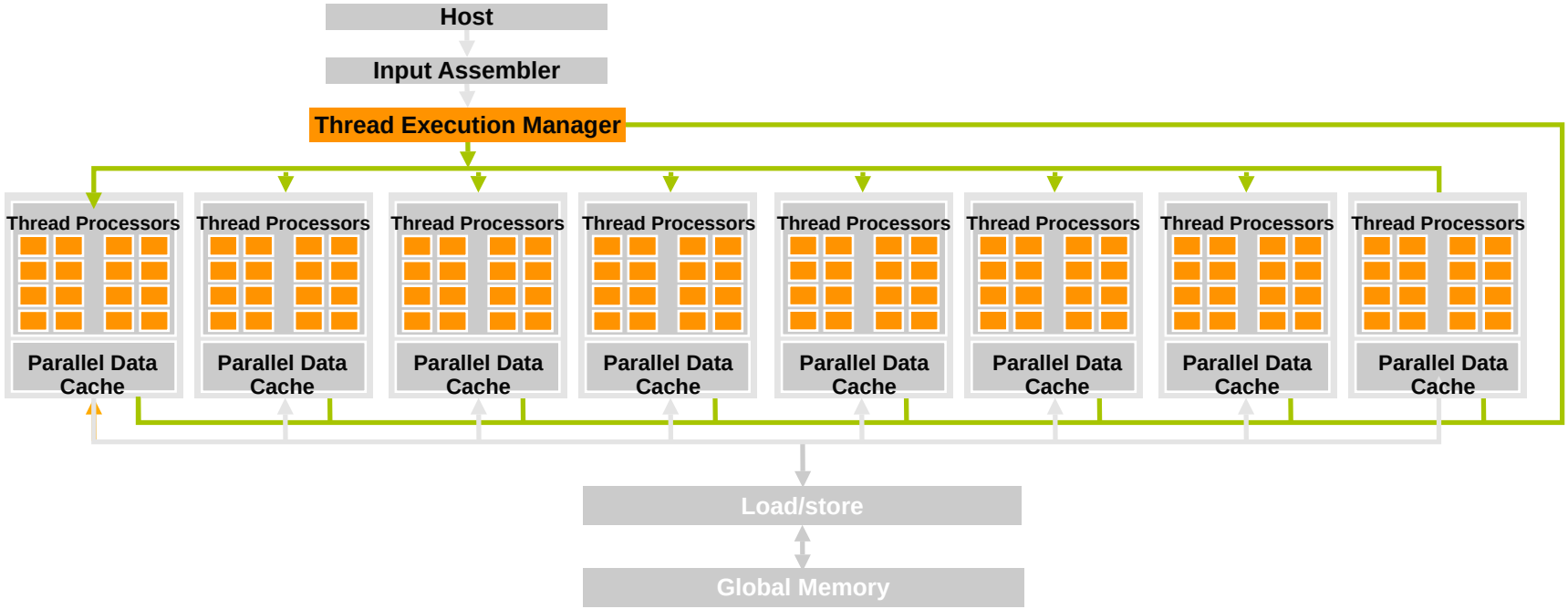
Yesterday's Vendor Support



Today's New Vendor Support

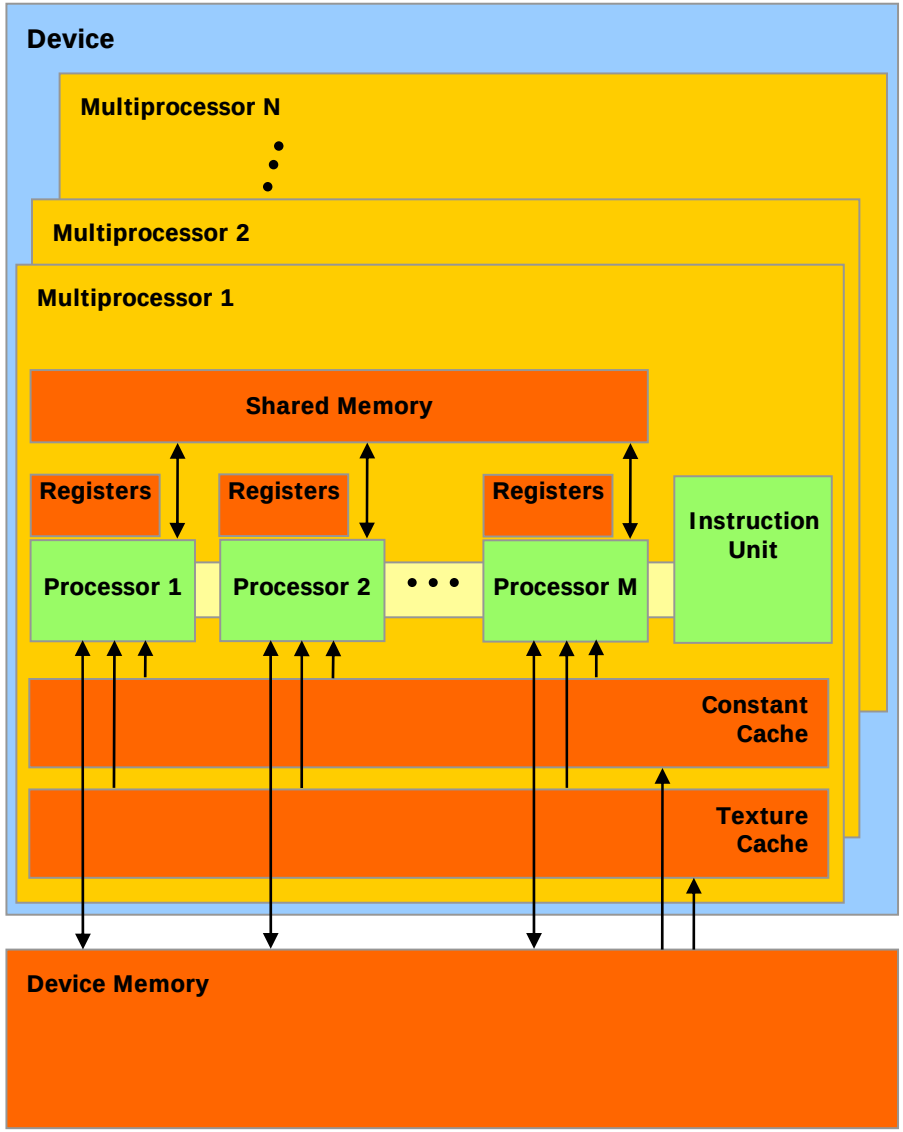


GeForce 8800 GPU

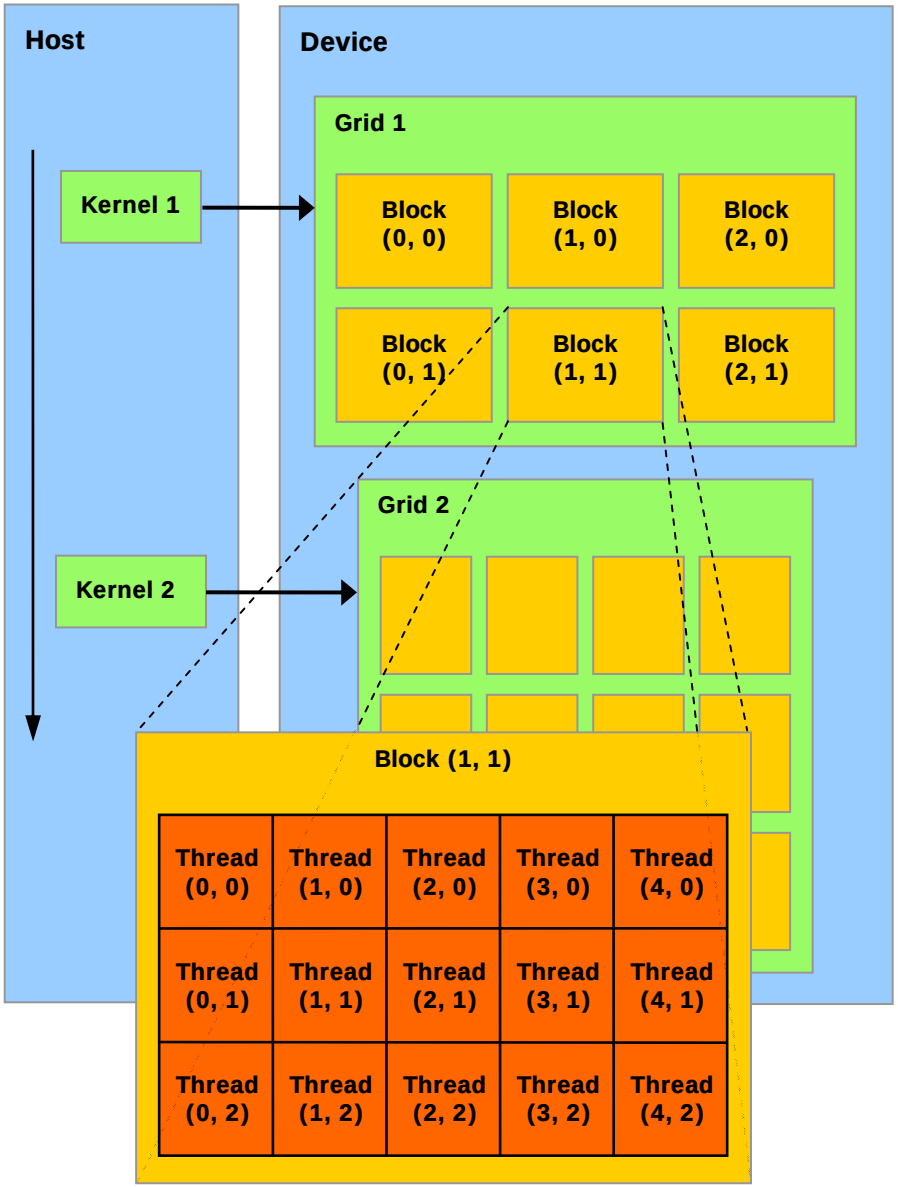


[courtesy of Ian Buck, NVIDIA]

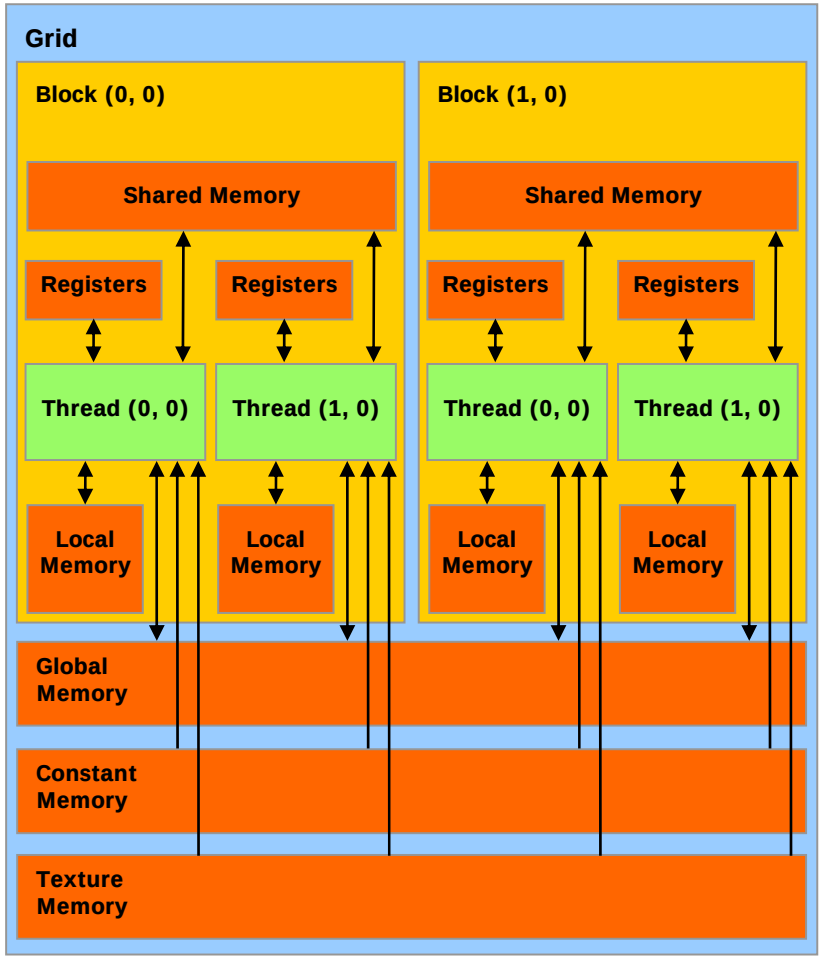
CUDA Hardware Abstraction



CUDA Program Structure



CUDA Memory Model



The CUDA Abstraction

- Computation organized into grids of thread blocks and threads within a thread block
- Kernels run over thread blocks
  - Kernels are SIMD
- Threads have arbitrary access to memory
  - Threads within a thread block share 16 KB shared memory on chip
- CUDA maps thread blocks to hardware
- Programmer responsible for CPU<->GPU communication, synchronization
- Interoperability of CUDA & OpenGL, D3D

What I Like

- Mapping of problems into blocks
  - Blocks comprise 1D, 2D, 3D grids
    - Expect scalability will likely target “more blocks” rather than “more threads per block” or “faster/more capable threads”
    - Good match to hardware

What I Like

- Software environment
  - C-like language with extensions
  - Emulation mode is great
  - Profiling doesn’t give us much (yet) but gives us what’s most important
    - Would like register usage information

Memory Consistency

- No guarantee of consistency between threads
- This is a deep question

What I Don’t Like (HW)

- Memory coalescing seems fairly restrictive
- Currently no concurrency between transfer and compute
- Bank conflicts are a pain to deal with

What I Don't Like (SW)

- Shared memory declarations are awkward
  - Multiple buffers, multiple types of memory
- Pointers into shared memory are awkward
- Manual synchronization is biggest source of bugs
- Bug identification is tough …
  - CUDA? Driver? Our code?

GPGPU: Bottom Up

- Scan, a parallel primitive originally for APL, used in Connection Machine
  - Practically interesting only in parallel contexts
- Efficient GPU implementation
- Good for problems that require global communication
  - Quicksort, radix sort, sparse matrix ops, tridiagonal solvers, trees & graphs, geometry manipulation …

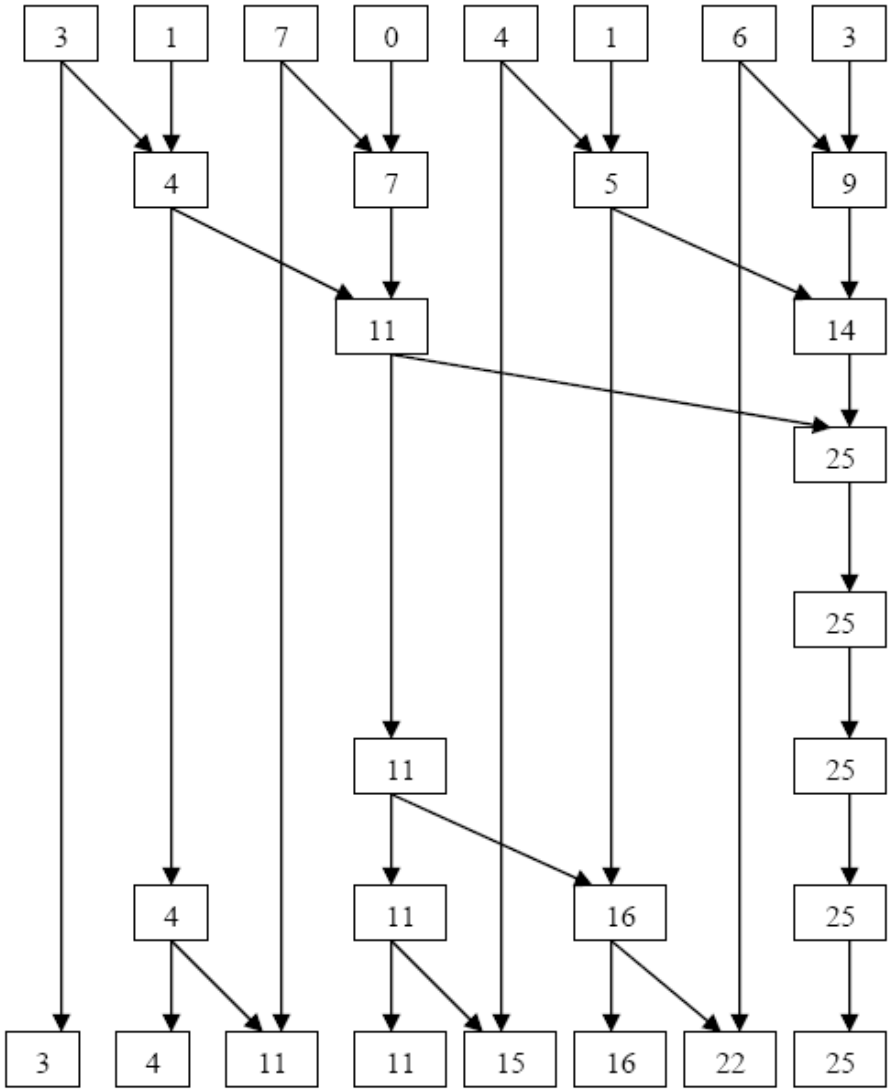
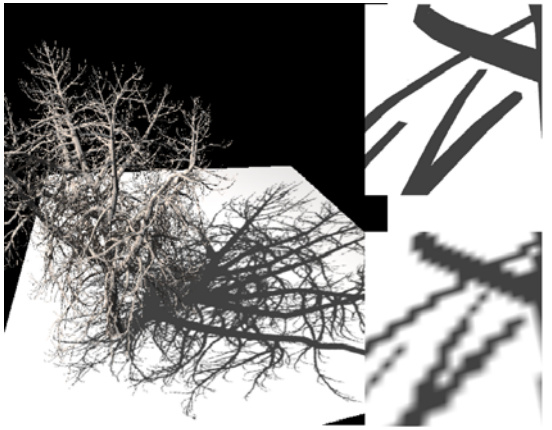
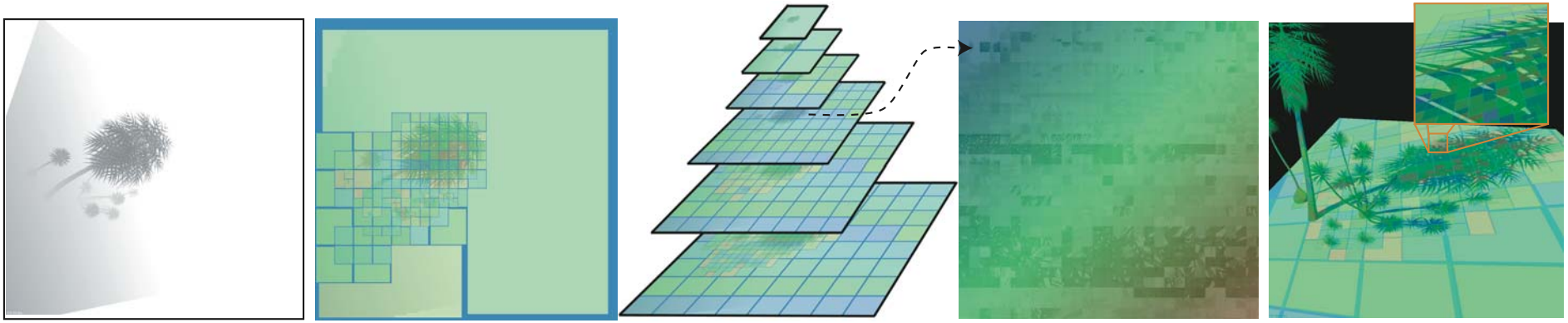


Figure courtesy Shubhabrata Sengupta

GPGPU: Top Down

- Goal: Adaptive, multiresolution grid
- Strategy: Page-table formulation, most work done on GPU
- Result: First GPU support of adaptive shadow maps





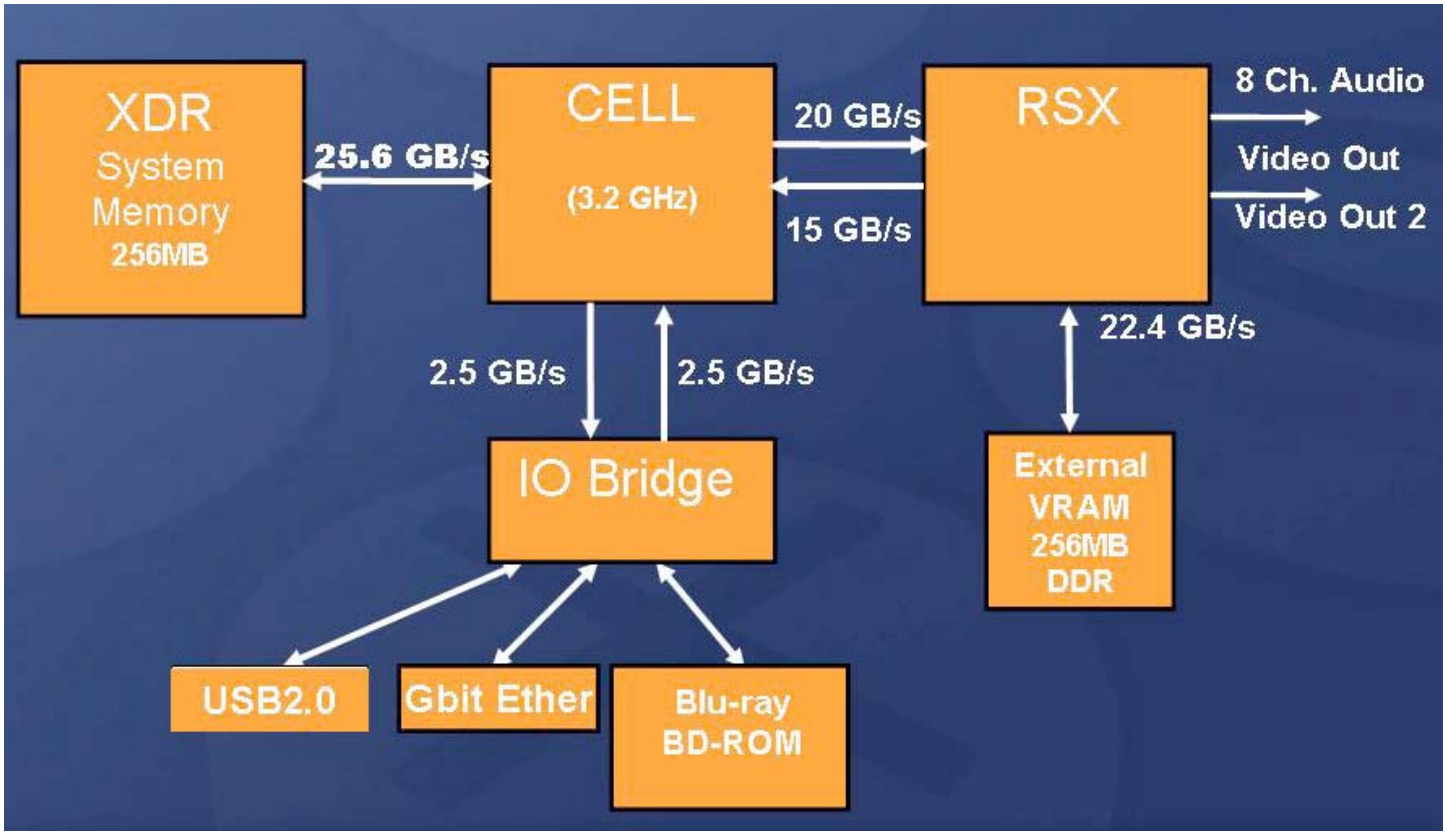
(a) Virtual Domain (b) Adaptive Tiling (c) Page Table (d) Physical Memory (e) Adaptive Shadow Map

Lefohn et al., "Glift: An Abstraction for Generic, Efficient GPU Data Structures." ACM TOG Jan 2006.

Lessons Learned

- Lack of abstractions, APIs, libraries is a critical problem
  - "Vertical" development needs to become "horizontal"
- Division of labor
  - Data structure creation, access, change, update … who's responsible?
  - Today crippled by low-bandwidth path between CPU and GPU
  - Lessons of PS3 are useful here

Sony PS3



- CPUs are good at creating & manipulating data structures
- GPUs are good at accessing & updating data structures

http://www.watch.impress.co.jp/game/docs/20060329/3dps303.jpg

Belief 1.

One size does not fit all.

Future computing systems will feature both coarse-grained thread parallelism and fine-grained data parallelism.

Belief 2.

SIMD is a strong candidate for fine-grained parallelism.

Belief 3.

Fine-grained parallel units should have efficient support for reductions and expansions.

Belief 4.

Future heterogeneous architectures must be able to share data between heterogeneous units with high bandwidth.

Belief 5.

Throughput is more important than latency.

Belief 6.

New architectures will allow explicit programmer control over the memory hierarchy.

Belief 7.

It’s the software, stupid.

The successful future computing systems will be the ones that best allow their programming environments to exploit parallelism.

Belief 8.

Future programs must exploit parallelism to continue to achieve performance gains.

I believe those programs must be written in explicitly parallel languages or with e.p. libraries.

Belief 9.

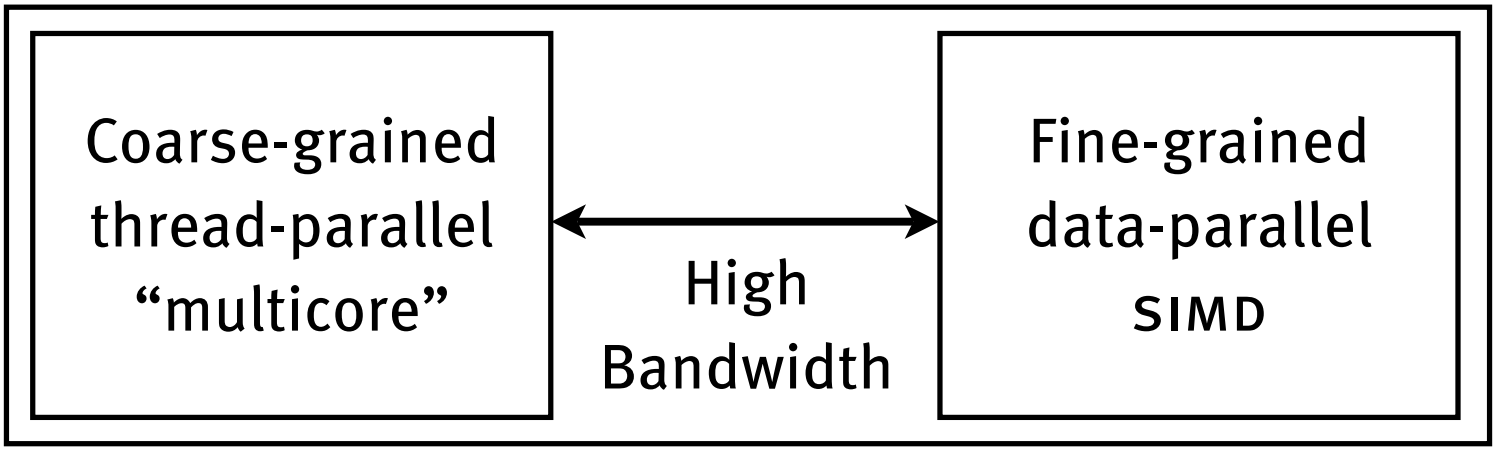
It is necessary to develop parallel APIs and abstractions that work across different architectures and programming systems.

Industry is probably not the place to do this.

Belief 10.

The most important new thing we must teach students in computing is how to think in parallel.

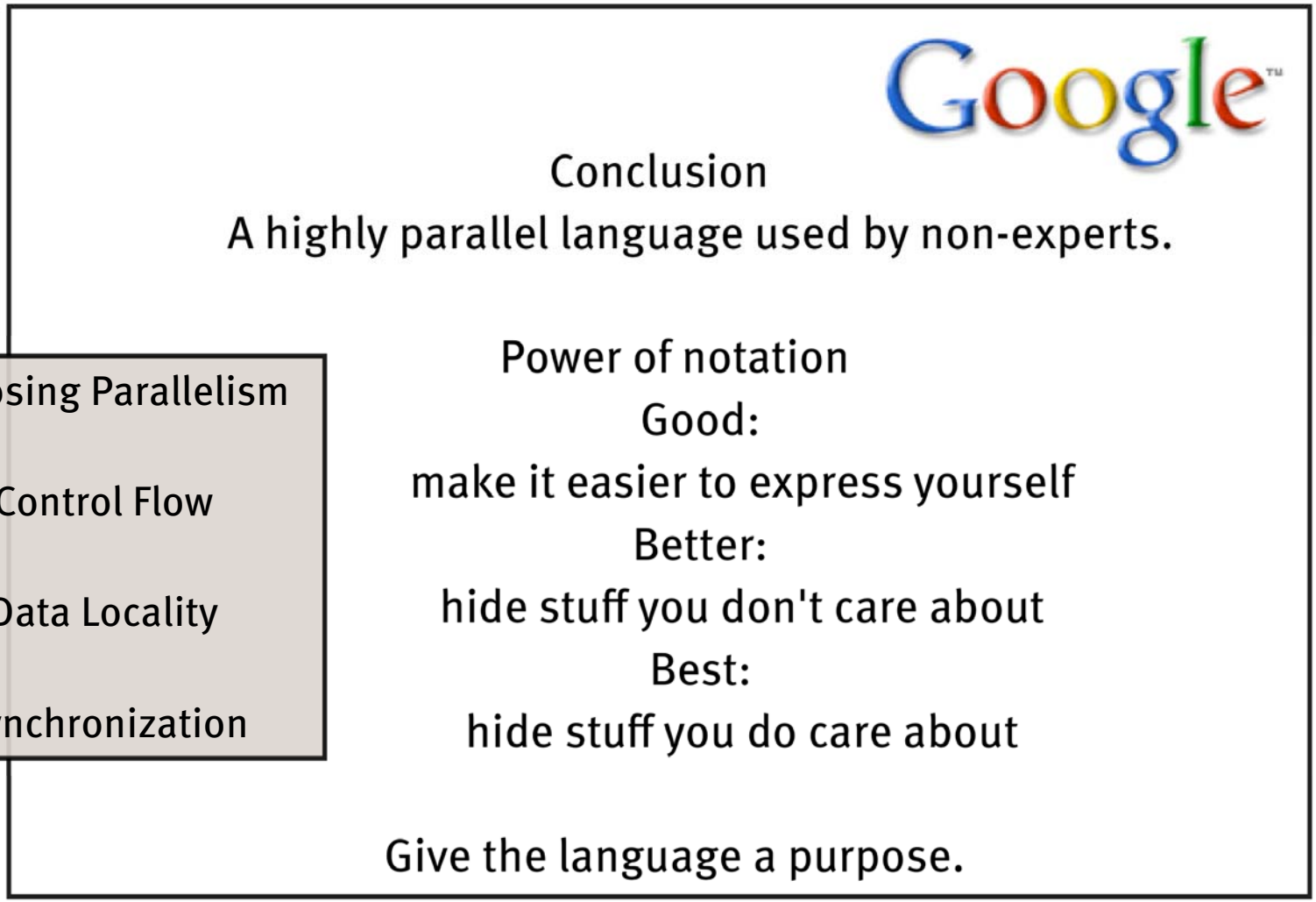
Summary



Software interface

- Explicitly parallel
- Explicit management of memory hierarchy
- Works with other hw (Cell, GPU, …)

Rob Pike on Languages



My GPGPU Top Ten

- The Killer App
- Programming models and tools
- GPU in tomorrow's computer?
- Data conditionals
- Relationship to other parallel hw/sw
- Managing rapid change in hw/sw (roadmaps)
- Performance evaluation and cliffs
- Philosophy of faults and lack of precision
- Broader toolbox for computation / data structures
- Wedding graphics and GPGPU techniques

Acknowledgements

- Collaborators at UC Davis: Aaron Lefohn, Andy Riffel, Shubhabrata Sengupta, Adam Moerschell, Yao Zhang
- Pat McCormick (Los Alamos)
- Mark Harris, David Luebke, Nick Triantos, Craig Kolb (NVIDIA)
- Mark Segal (AMD)
- Ian Buck, Tim Purcell, Pat Hanrahan, Bill Dally (Stanford)
- Funding: DOE Office of Science, DOE SciDAC Institute for Ultrascale Visualization, NSF, Los Alamos National Laboratory, Lockheed Martin, Chevron, UC MICRO, UCD

The Research Landscape

- Thank you for supporting the University of California.
- DOD funding for computing research in universities was cut in half between 2001 and 2005.
- NSF CISE award rates in FY2004 were 16%, lowest of all directorates.
- Intel's help in both lobbying for federal funding and directly supporting interesting research is both welcomed and essential.

For more information …

- GPGPU home: http://www.gpgpu.org/
  - Mark Harris, UNC/NVIDIA
- GPU Gems (Addison-Wesley)
  - Vol 1: 2004; Vol 2: 2005; Vol 3: 2007
  - Vol 2 has NVIDIA GeForce 6800
- Survey paper ("A Survey of General-Purpose Computation on Graphics Hardware", Owens et al., Computer Graphics Forum March 2007)
- Conferences: Siggraph, Graphics Hardware, GP2, EDGE
  - Course notes: Siggraph '05–07, IEEE Visualization '04–05, Supercomputing '06
- University research: Caltech, CMU, Duisberg, Illinois, Purdue, Stanford, SUNY Stonybrook, Texas, TU München, Utah, UBC, UC Davis, UNC, Virginia, Waterloo