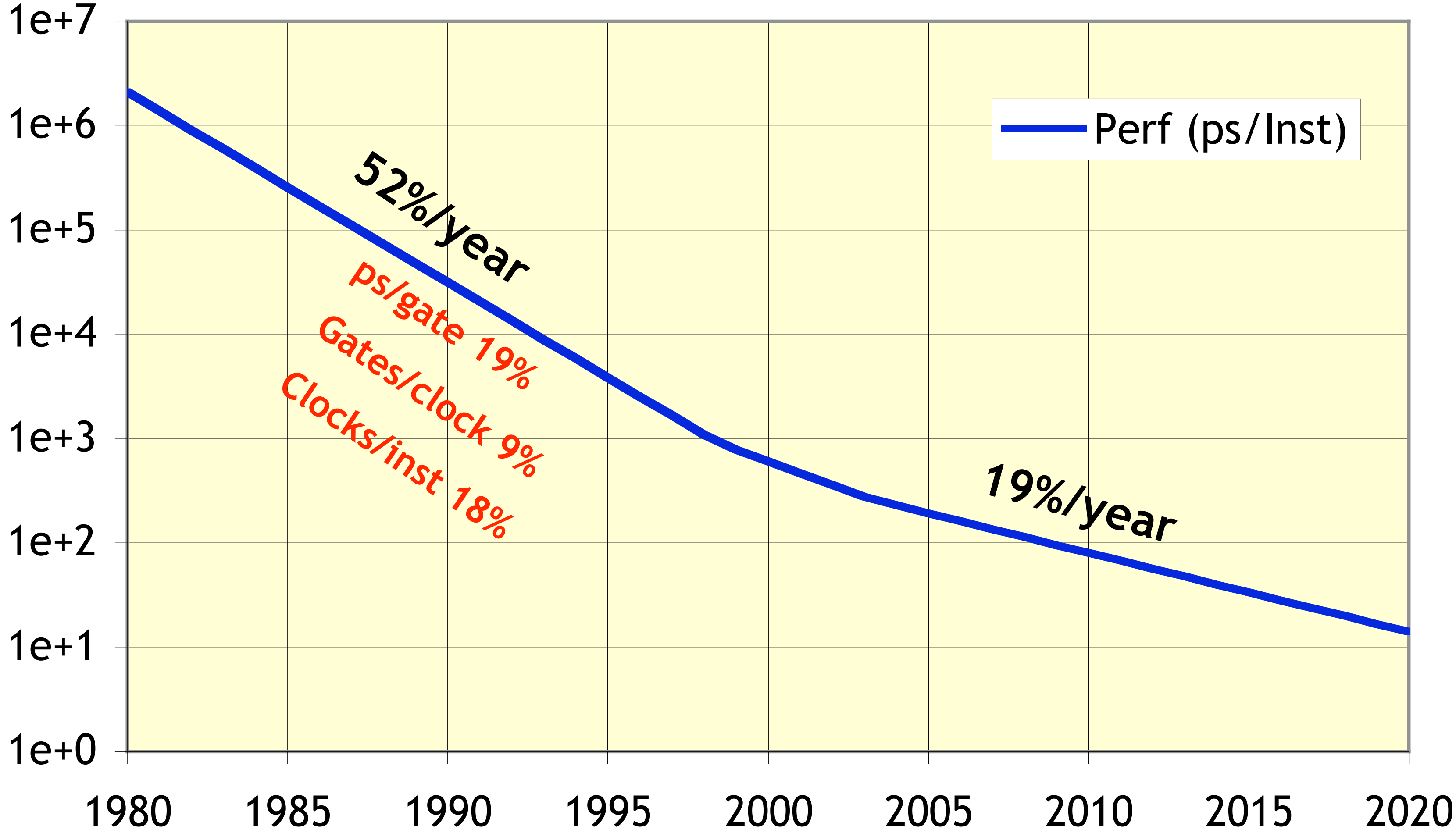
# What’s New with GPGPU?

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

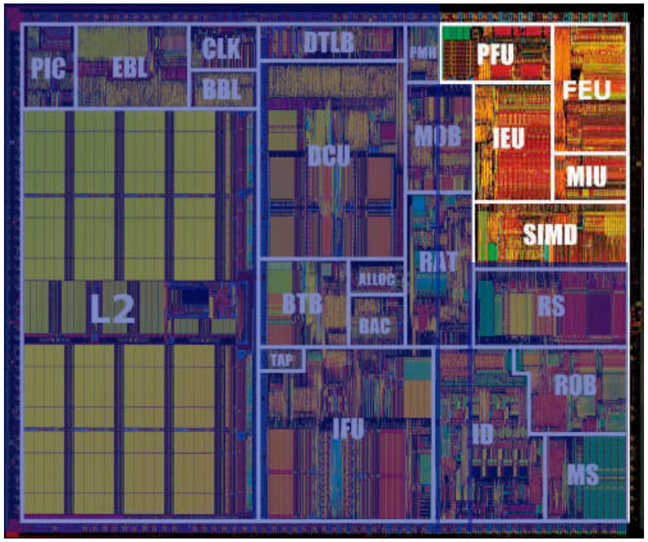
# Microprocessor Scaling is Slowing




[courtesy of Bill Dally]

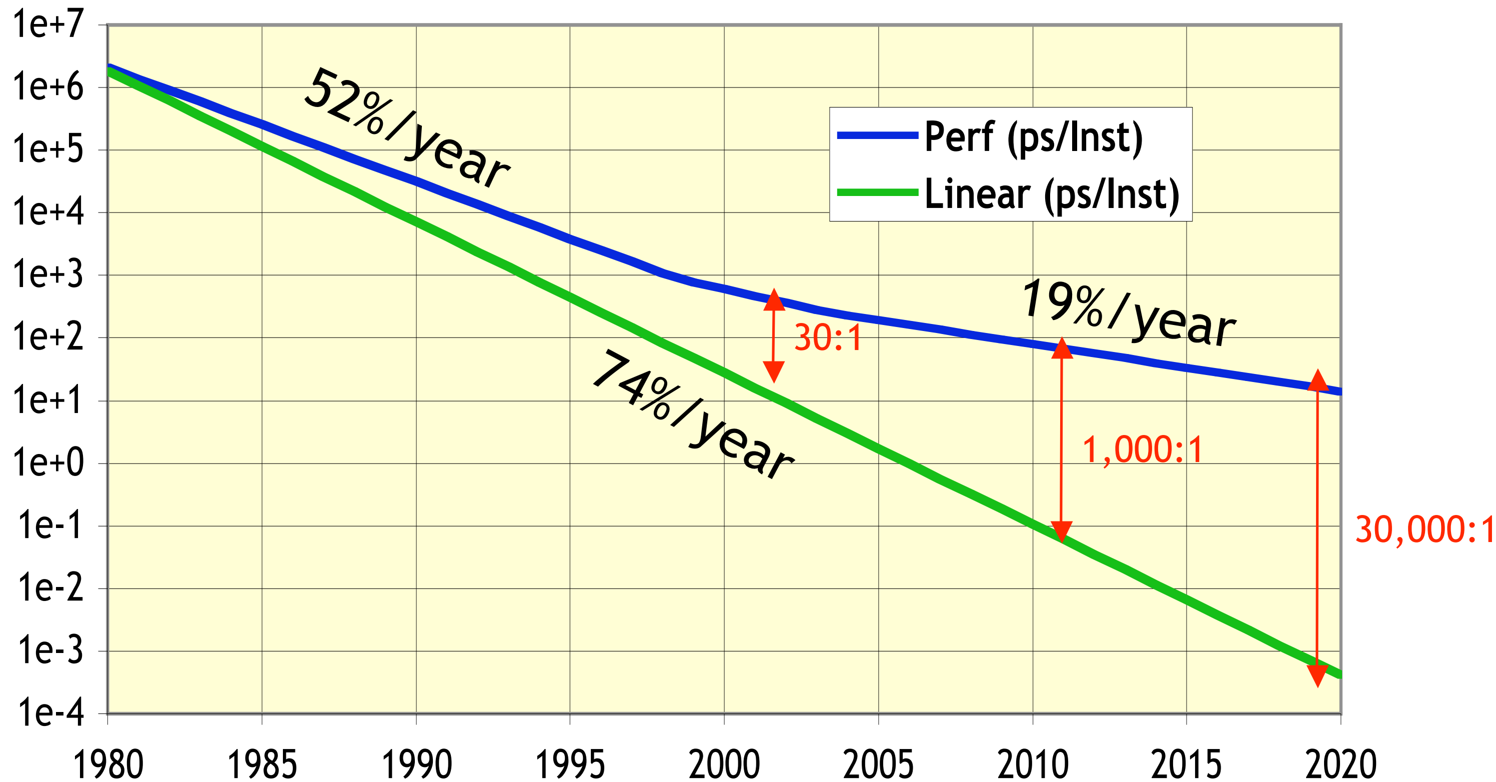
# Today's Microprocessors

- **Scalar programming model with no native data parallelism**
  - SSE is the exception
- **Few arithmetic units - little area**
- **Optimized for complex control**
- **Optimized for *low latency* not *high bandwidth***
- **Result: poor match for many apps**




Pentium III – 28.1M T

# Future Potential is Large



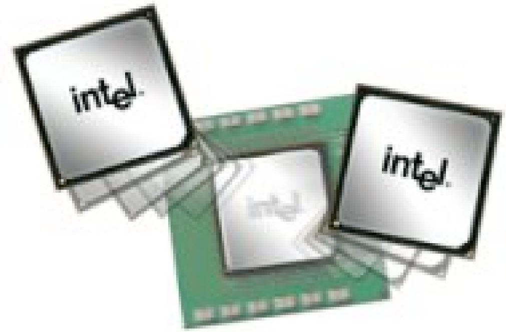

- **2001: 30:1**
- **2011: 1000:1**

[courtesy of Bill Dally]

# Parallel Processing is the Future

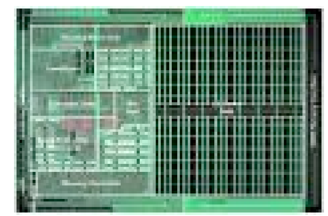
Major vendors supporting multicore

- Intel, AMD

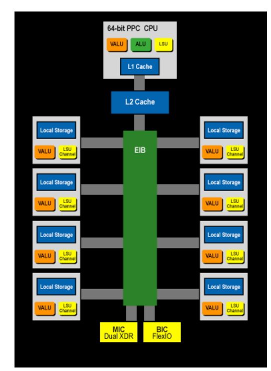



Excitement about IBM Cell

Hardware support for threads

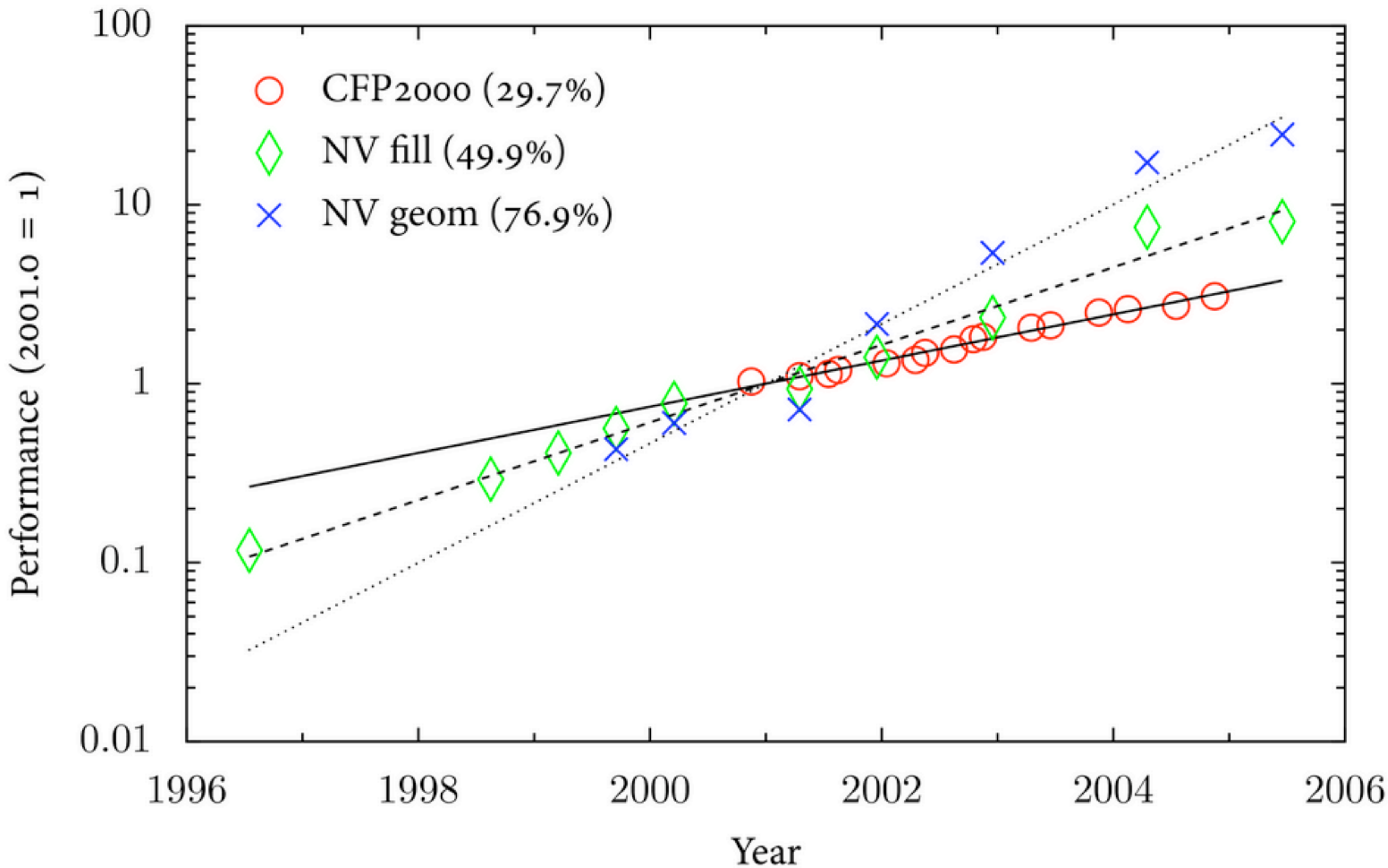


Interest in general-purpose programmability on GPUs



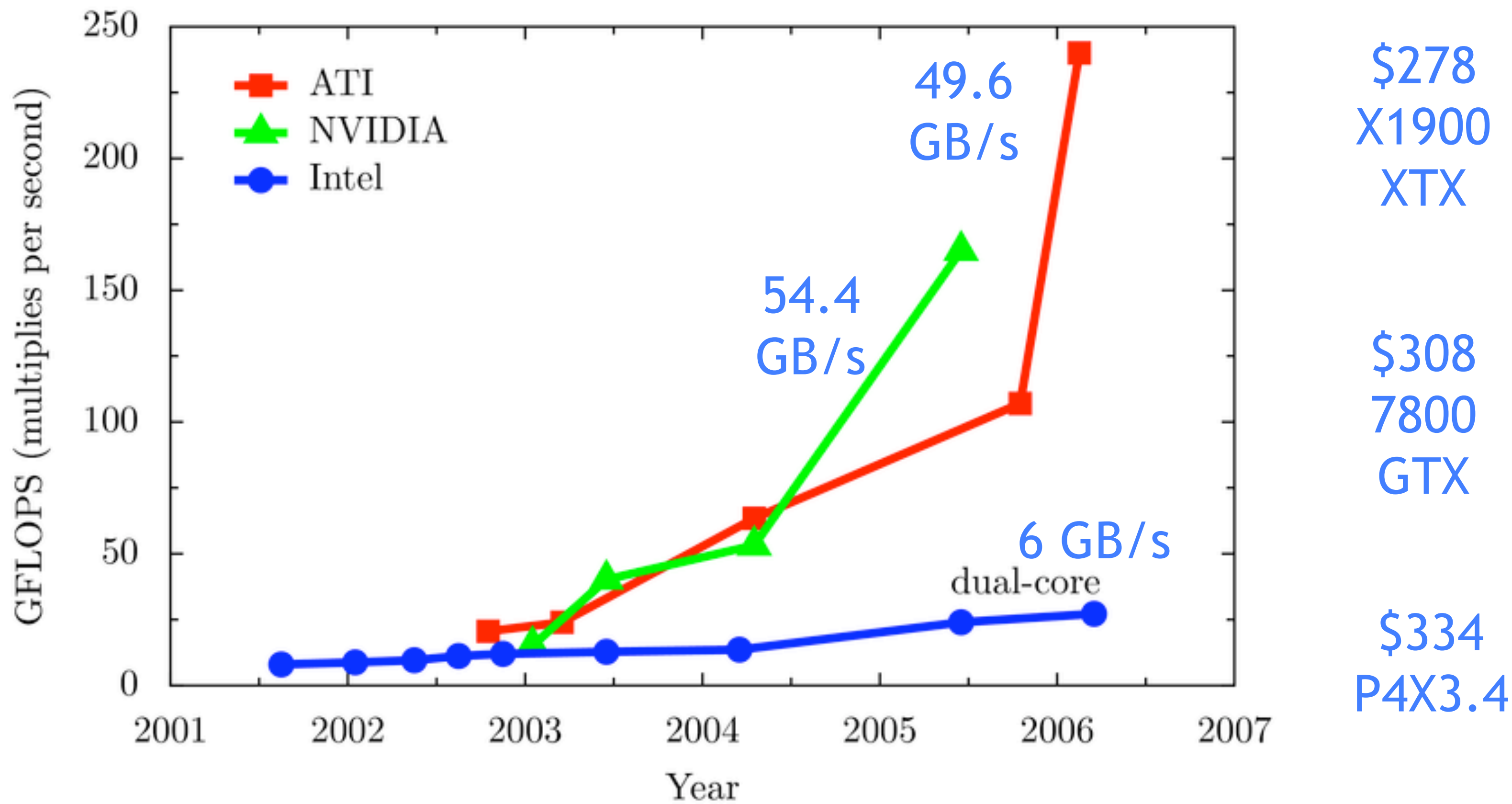

Universities must teach thinking in parallel

# Long-Term Trend: CPU vs. GPU



# Recent GPU Performance Trends

Programmable 32-bit FP multiplies per second




*Data courtesy Ian Buck; from Owens et al. 2005 [EG STAR]*

# Functionality Improves Too!

10 years ago:

- Graphics done in software


5 years ago:

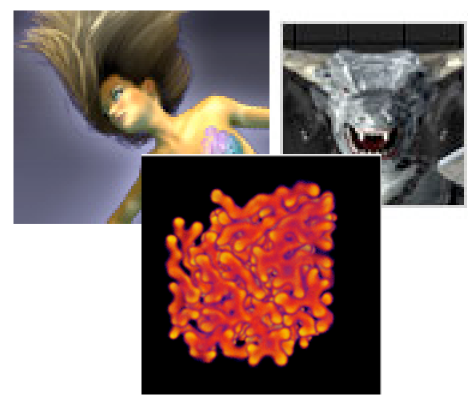
- Full graphics pipeline

Today:

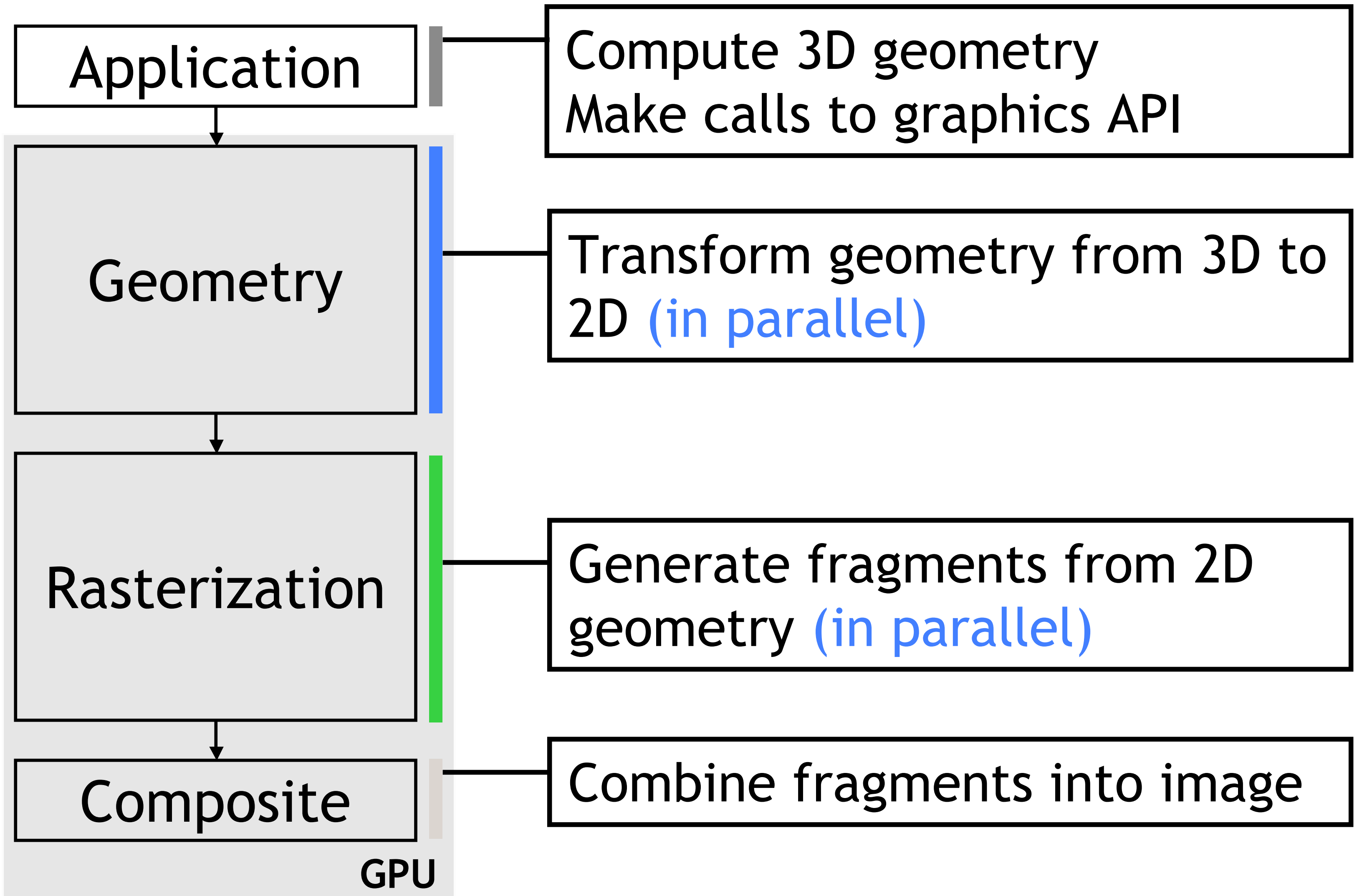
- 40x geometry, 13x fill vs. 5 yrs ago
- Programmable!

Programmable, data parallel processing on every desktop

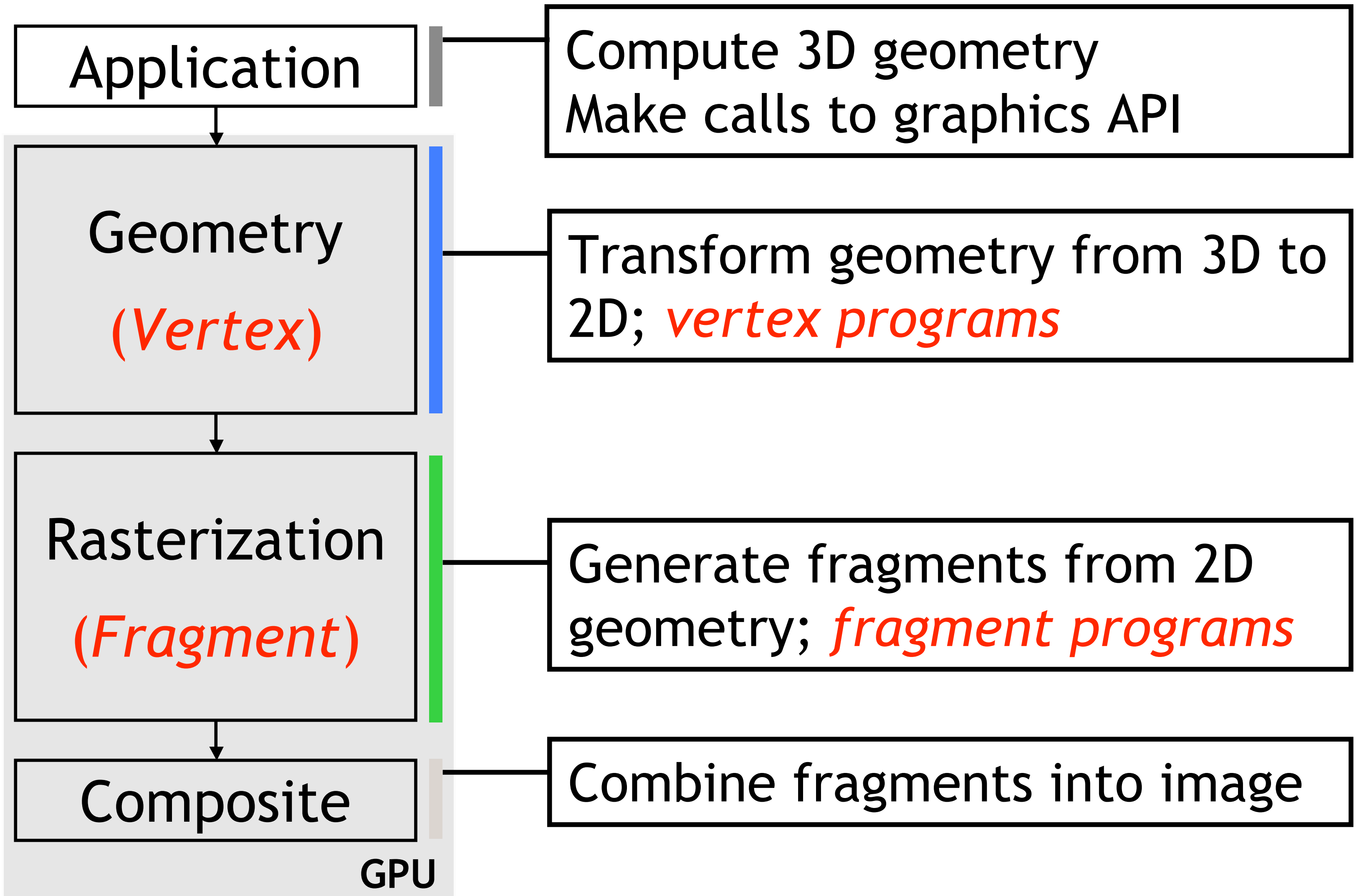
The GPU is the first commercial data-parallel processor



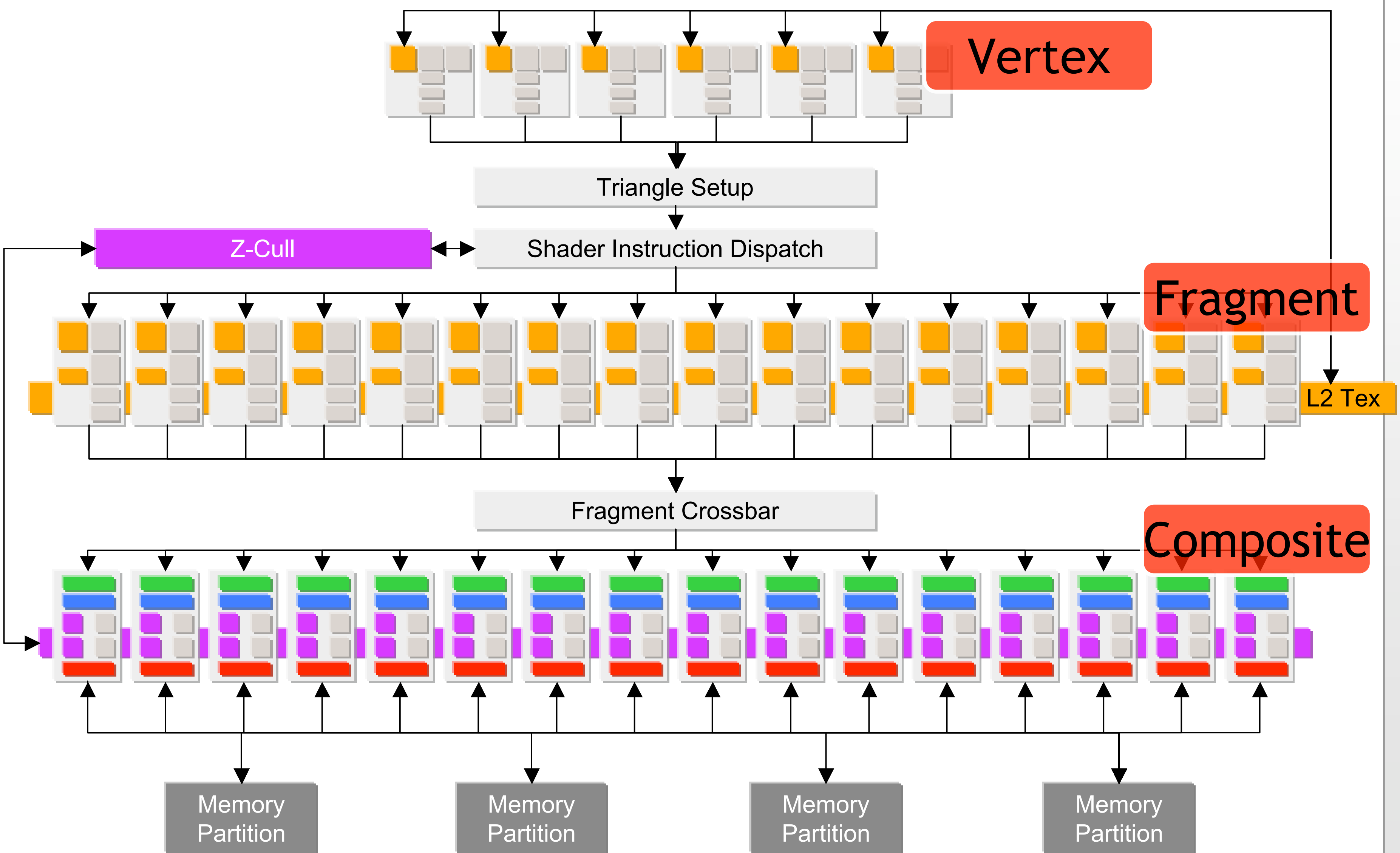
# The Rendering Pipeline



# The Programmable Rendering Pipeline

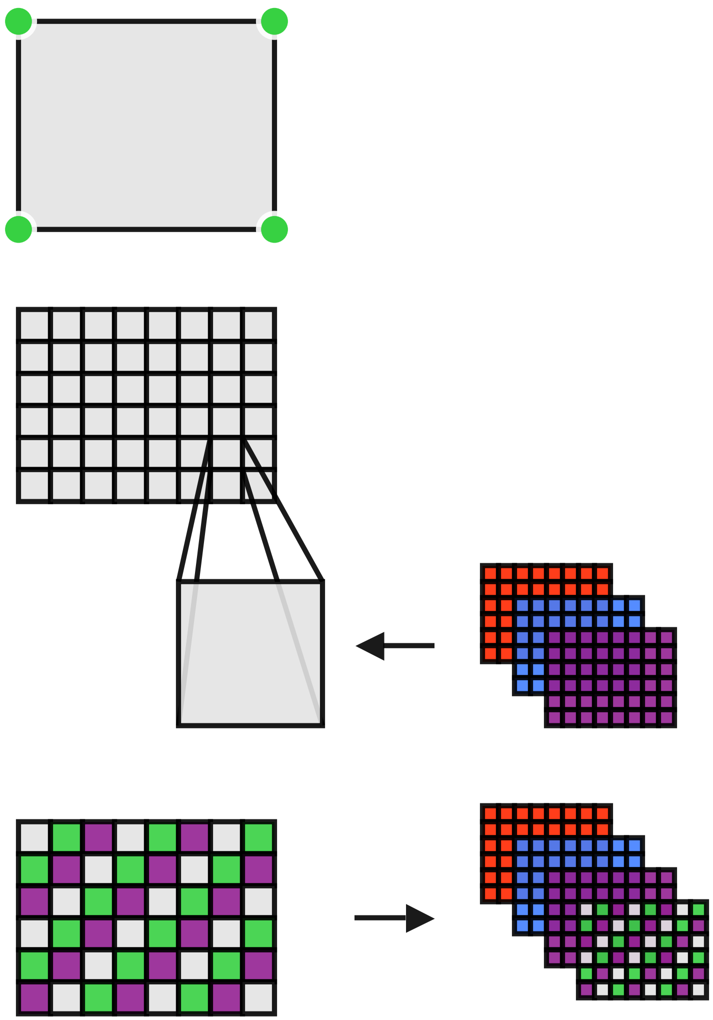


# NVIDIA GeForce 6800 3D Pipeline



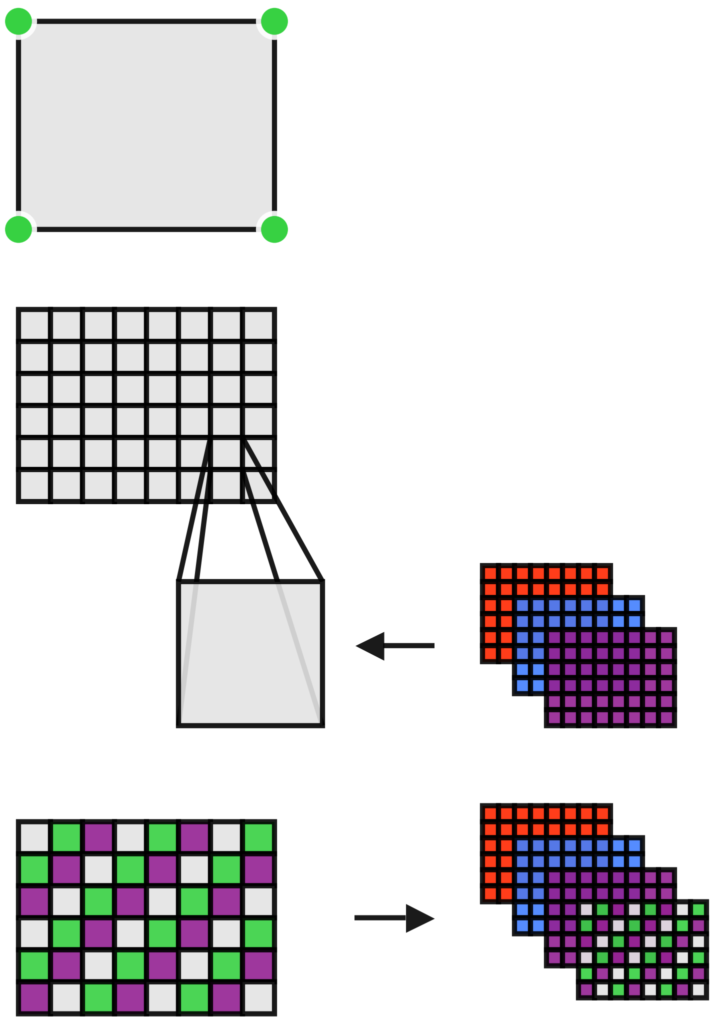

*Courtesy Nick Triantos, NVIDIA*

# 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

# GPUs are fast (why?) …

**Characteristics of computation permit efficient hardware implementations**

- High amount of parallelism …
- … exploited by graphics hardware
- High latency tolerance and feed-forward dataflow …
- … allow very deep pipelines
- … allow optimization for bandwidth not latency

**Simple control**

- Restrictive programming model

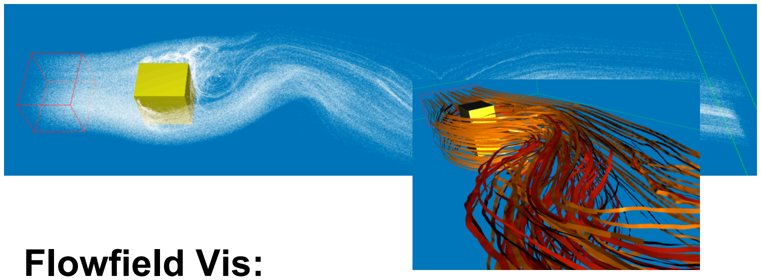
**Competition between vendors**

# … but GPU programming is hard

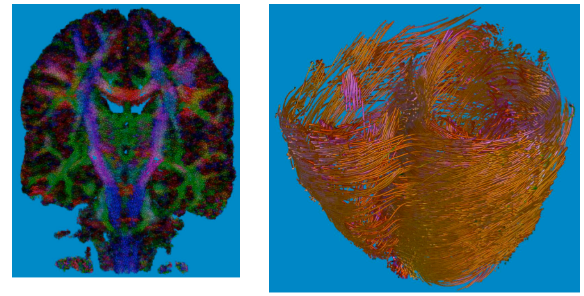
- Must think in graphics metaphors
- Requires parallel programming (CPU-GPU, task, data, instruction)
- Restrictive programming models and instruction sets
- Primitive tools
- Rapidly changing interfaces

Big picture: Every time I say “GPU”, substitute “parallel processor”

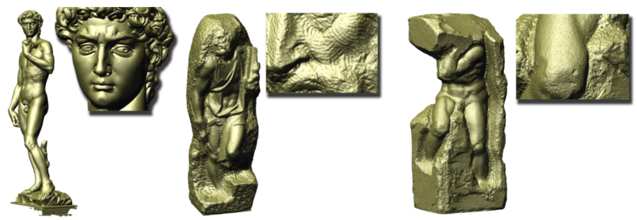
# GPGPU in Scientific Visualization



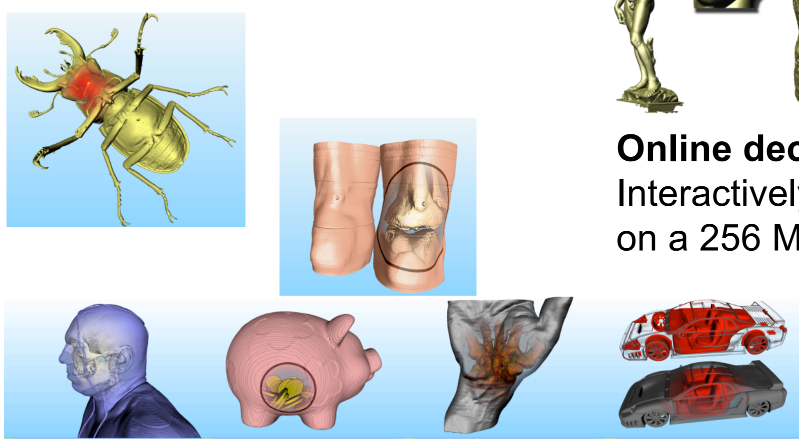
**Flowfield Vis:**
“a million particles with ease” @ 60 fps



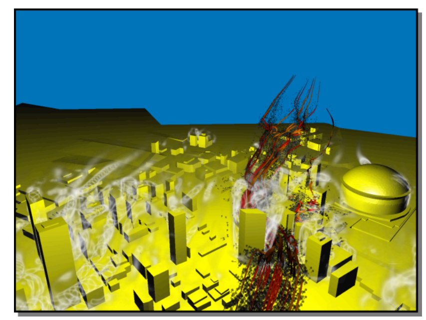
**Tensorfield Vis:**
CPU – GPU Speedup = 1 : 150 (!!!)



**Online decompression:**
Interactively Visualize 20 GB of data
on a 256 MB GPU



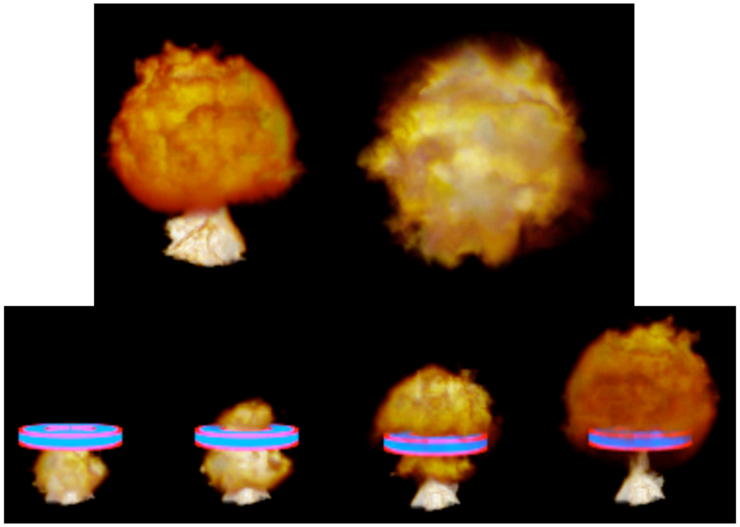
**Focus and Context Raycasting:**
High Quality interactive visualization
of large datasets on $400 hardware



**GPGPU and VIS**
**a winning team**
(winner of IEEE 2005 VIS Competition)

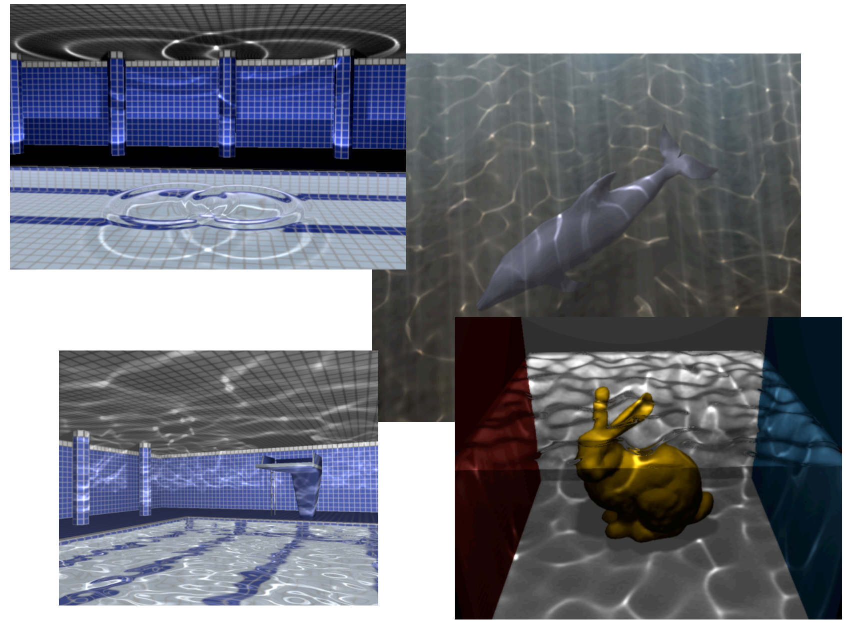
*Courtesy Jens Krüger (TU München)*

# GPGPU Effects



**Fire**

**Smoke**



**Water**

**Let your Virtual Reality come to live with interactive GPGPU effects!**
(all off these effects are simulated and rendered on the GPU in realtime)

*Courtesy Jens Krüger (TU München)*

# Challenge: Programming Systems

| Programming Model | High-Level Abstractions/ Libraries | Low-Level Languages | Compilers |
|---|---|---|---|

Performance Analysis Tools

Docs

| CPU | GPU |
|---|---|
| Scalar | Stream? Data-Parallel? |
| STL, GNU SL, MPI, … | Brook, Scout, sh, Glift |
| C, Fortran, … | GLSL, Cg, HLSL, … |
| gcc, vendor-specific, … | Vendor-specific |
| gdb, vtune, Purify, … | Shadesmith, NVPerfHUD |
| Lots | None |
| → *applications* | → *kernels* |

# Glift: Data Structures for GPUs

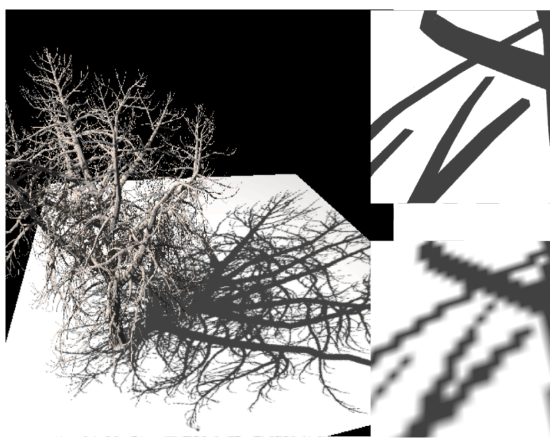
## Goal

- Simplify creation and use of random-access GPU data structures for graphics and GPGPU programming

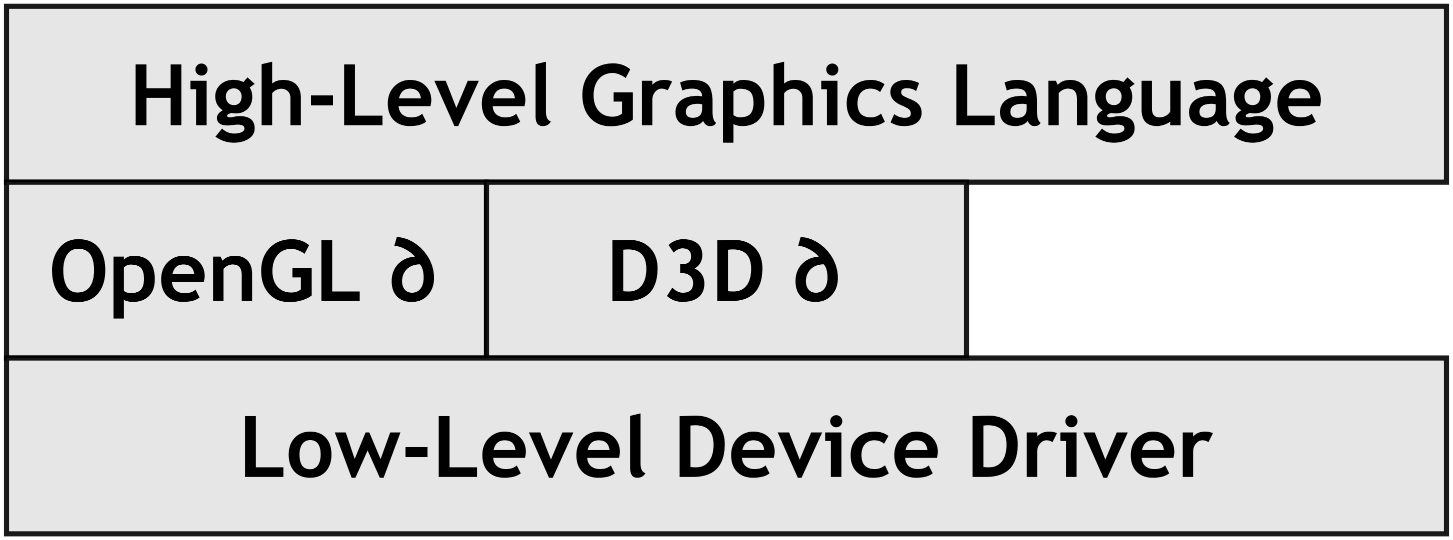
## Contributions

- Abstraction for GPU data structures
- Glift template library
- Iterator computation model for GPUs

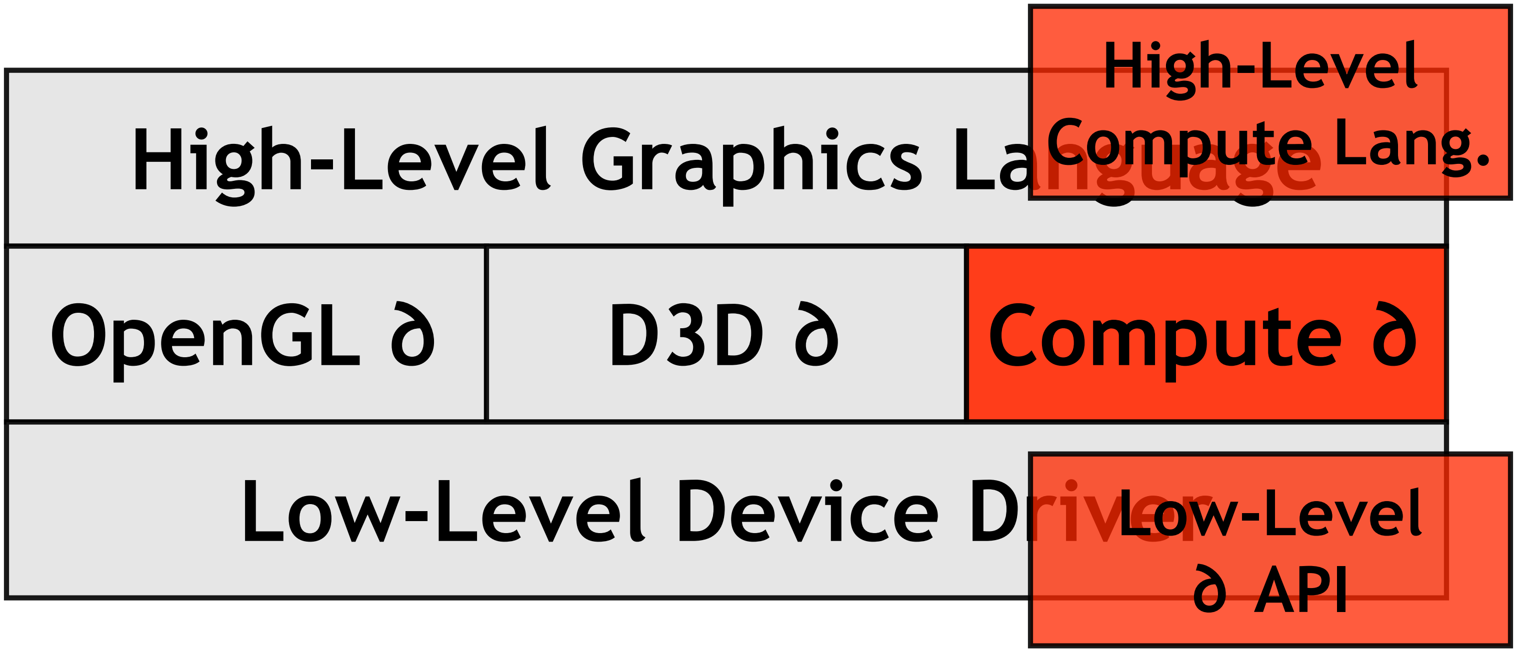
Aaron E. Lefohn, Joe Kniss, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. “Glift: An Abstraction for Generic, Efficient GPU Data Structures”'. ACM TOG Jan 2006.



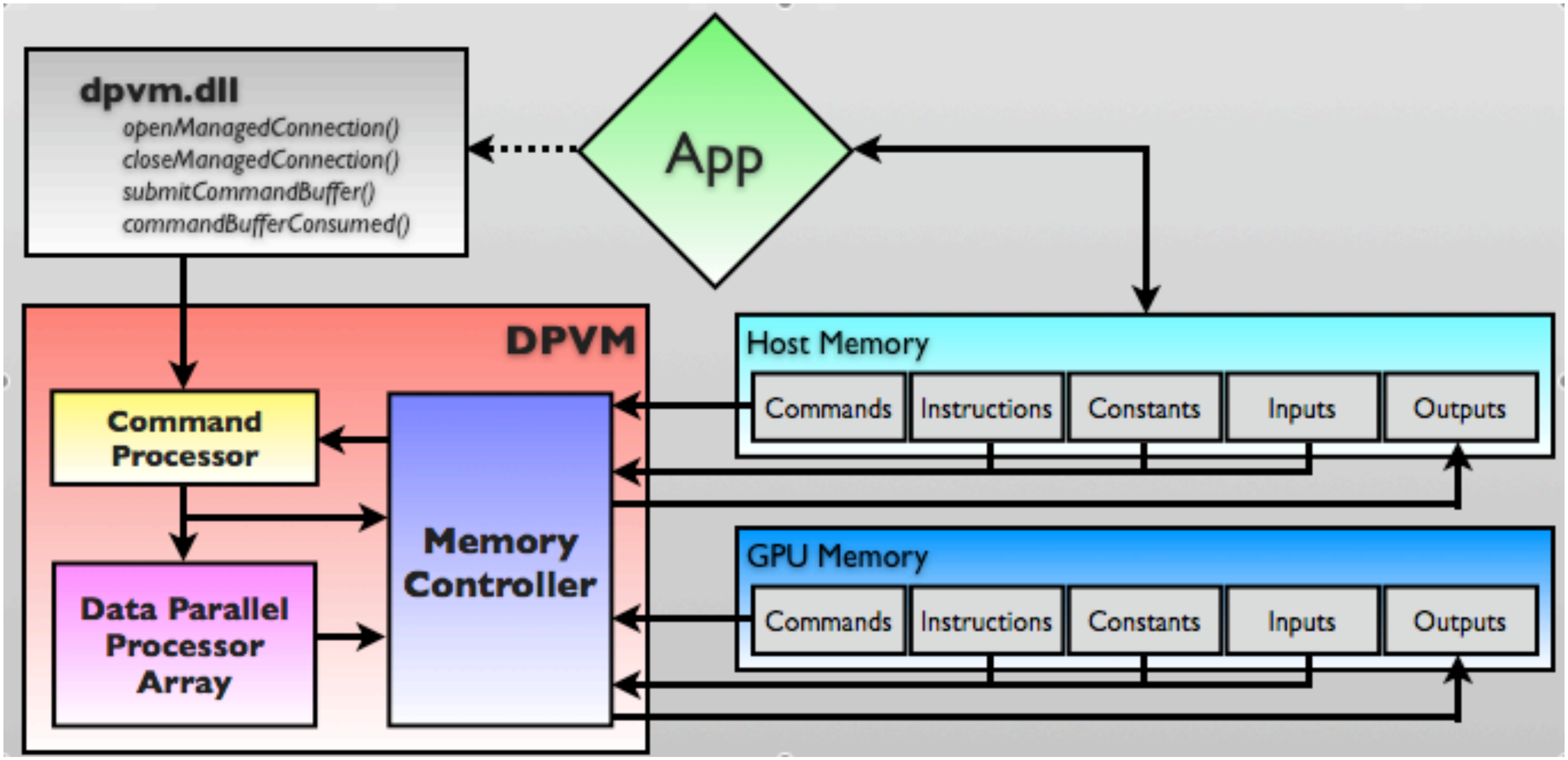
# Today’s Vendor Support



# Possible Future Vendor Support



# ATI CTM




## Low-level interface to GPU

# Big Picture Research Targets

We don't know what architecture will win. But we know it will be parallel.

- **Data structures**
  - Top-down approach rather than bottom-up … what SHOULD we support?
  - Interaction of algorithms and data structures
  - Export to multiple architectures
- **Self-tuning code**
- **Communication between GPUs**
- **Programming systems for *multiple* parallel architectures**
  - Major obstacle: difficulty of programming. Danger of fragmentation! Opportunity for education as well.
  - Learn from past
  - Explore portable primitives

**What we're good at: using GPUs as first class computing resources**

# Rob Pike on Languages

Conclusion

A highly parallel language used by non-experts.

Power of notation

Good:

make it easier to express yourself

Better:

hide stuff you don't care about

Best:

hide stuff you do care about

Give the language a purpose.

**Exposing Parallelism**

**Control Flow**

**Data Locality**

**Synchronization**

# Moving Forward …

What will DX10 give us?

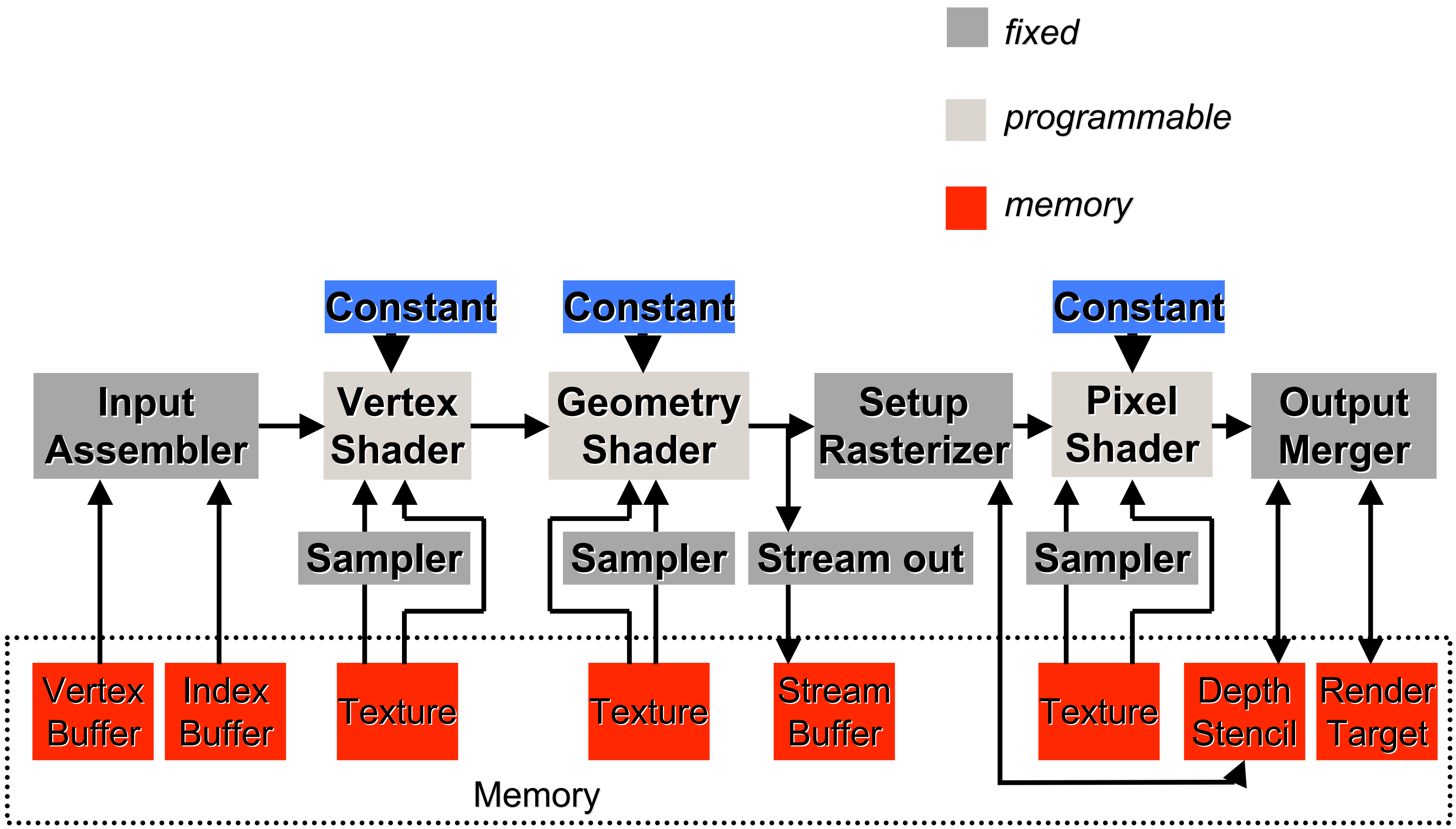
What works well now?

What doesn’t work well now?

What will improve in the future?

What will continue to be difficult?

# The New DX10 Pipeline




[courtesy David Blythe]

# What Runs Well on GPUs?

## GPUs win when …

- *Limited data reuse*

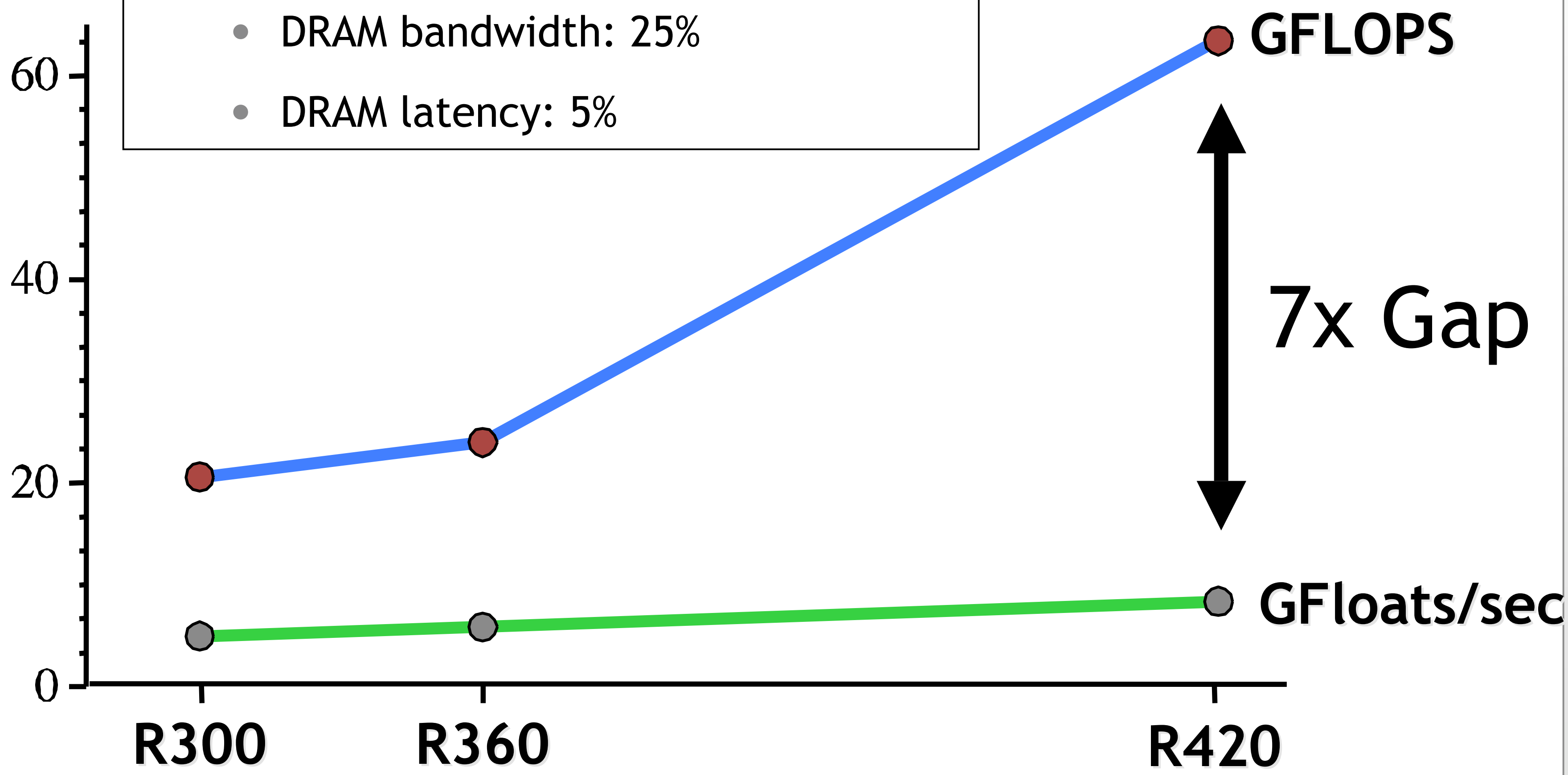
| | Memory BW | Cache BW |
|---|---|---|
| P4 3GHz | 6 GB/s | 44 GB/s |
| NV GF 6800 | 36 GB/s | -- |

- *High arithmetic intensity*: Defined as math operations per memory op
  - Attacks the memory wall - are all mem ops necessary?
- *Common error*: Not comparing against optimized CPU implementation

# Arithmetic Intensity

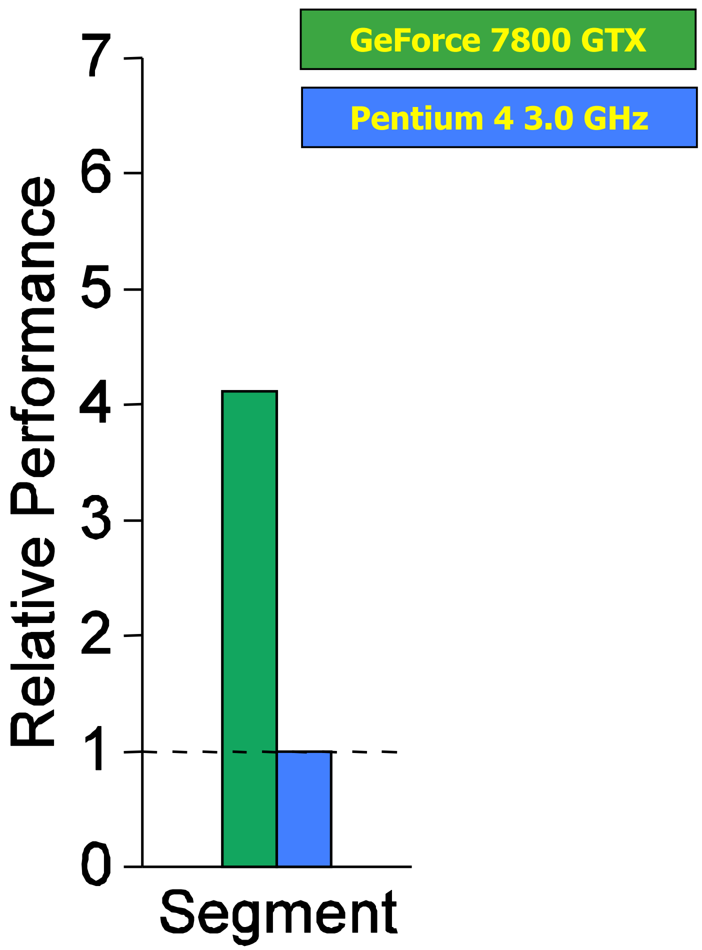
**Historical growth rates (per year):**

- Compute: 71%
- DRAM bandwidth: 25%
- DRAM latency: 5%




[courtesy Ian Buck]

# Arithmetic Intensity




## GPU wins when…

### Arithmetic intensity

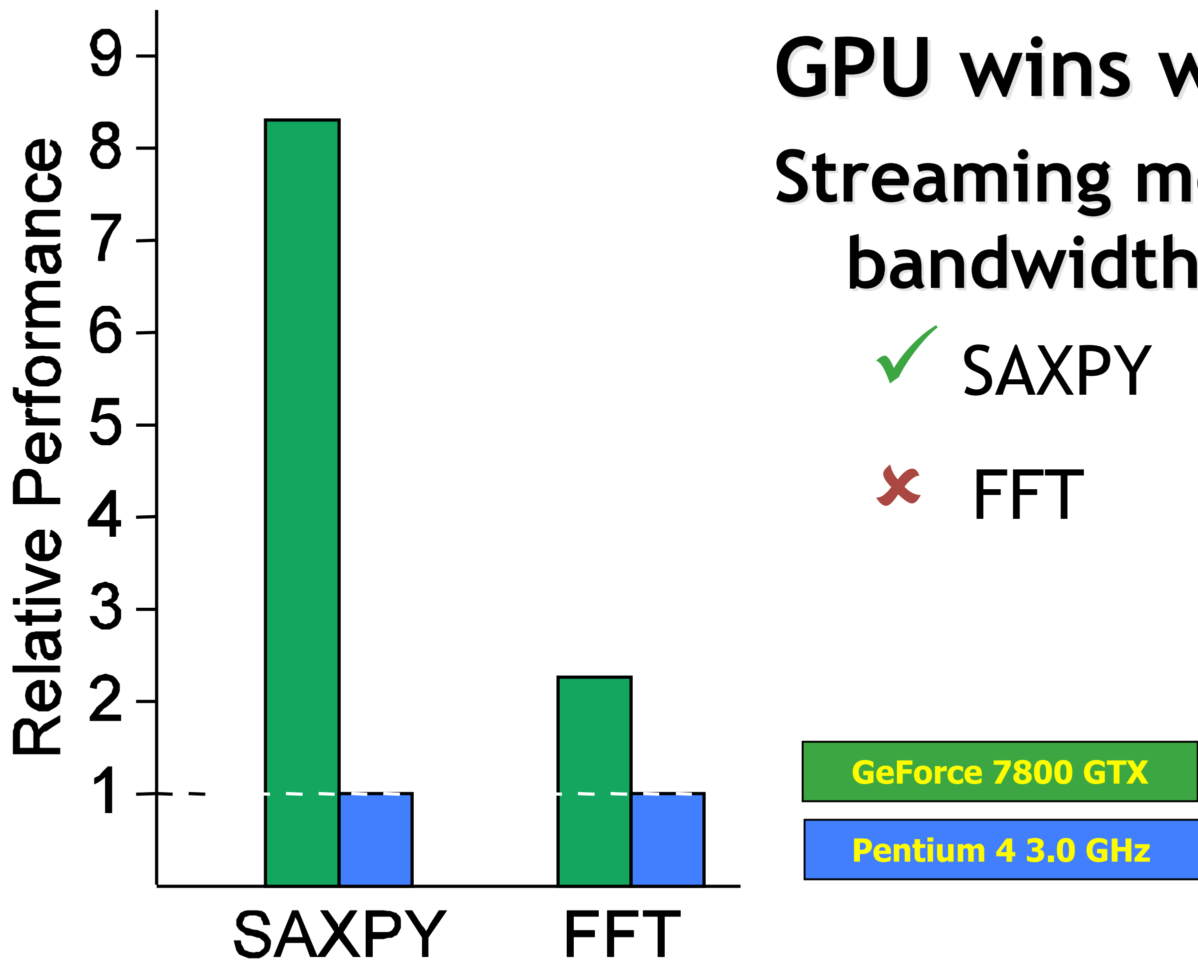
✓ Segment

3.7 ops per word

11 GFLOPS

[courtesy Ian Buck]

# Memory Bandwidth



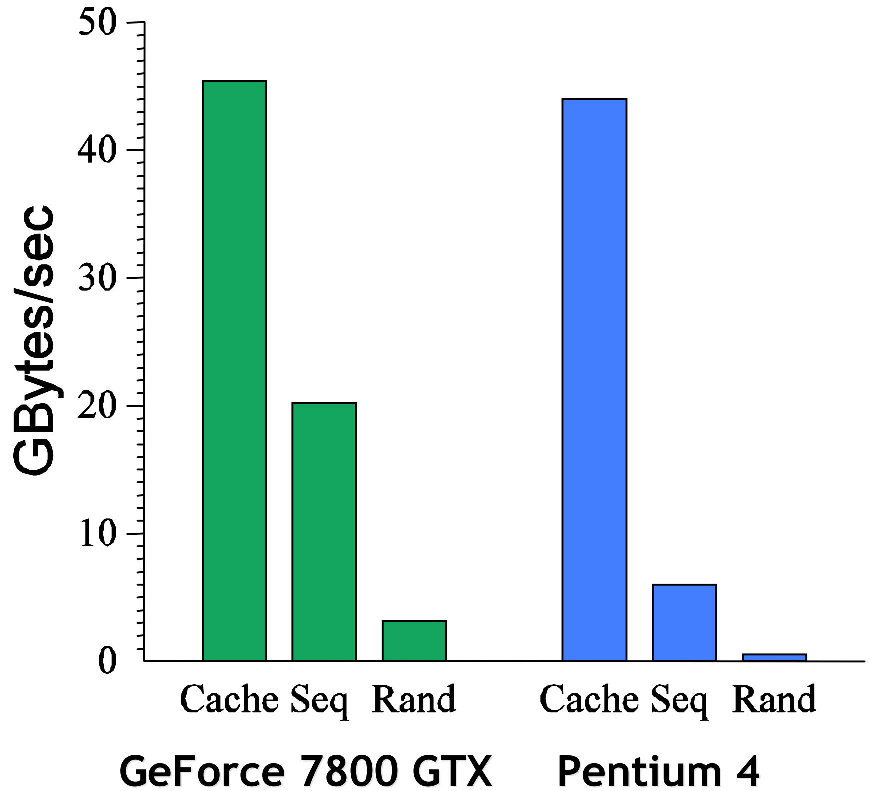

## GPU wins when…

**Streaming memory bandwidth**

- ✓ SAXPY
- ✗ FFT

[courtesy Ian Buck]

# Memory Bandwidth




## Streaming Memory System

- Optimized for sequential performance

## GPU cache is limited

- Optimized for texture filtering
- Read-only
- Small

## Local storage

- CPU >> GPU

[courtesy Ian Buck]

# What Will (Hopefully) Improve?

## Orthogonality

- Instruction sets
- Features
- Tools

## Stability

## Interfaces, APIs, libraries, abstractions

- Necessary as graphics and GPGPU converge!

# What Won't Change?

**Rate of progress**

**Precision (64b floating point?)**

**Parallelism**

- Won't sacrifice performance

**Difficulty of programming parallel hardware**

- … but APIs and libraries may help

**Concentration on entertainment apps**

# 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