

---

# A Data-Parallel Genealogy: The GPU Family Tree

---



John Owens

Department of Electrical and Computer Engineering  
Institute for Data Analysis and Visualization  
University of California, Davis

# Outline

---

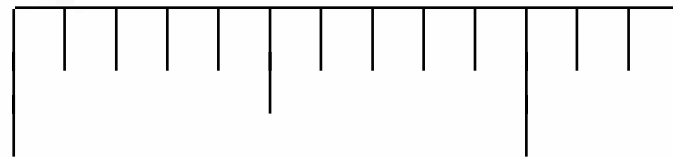
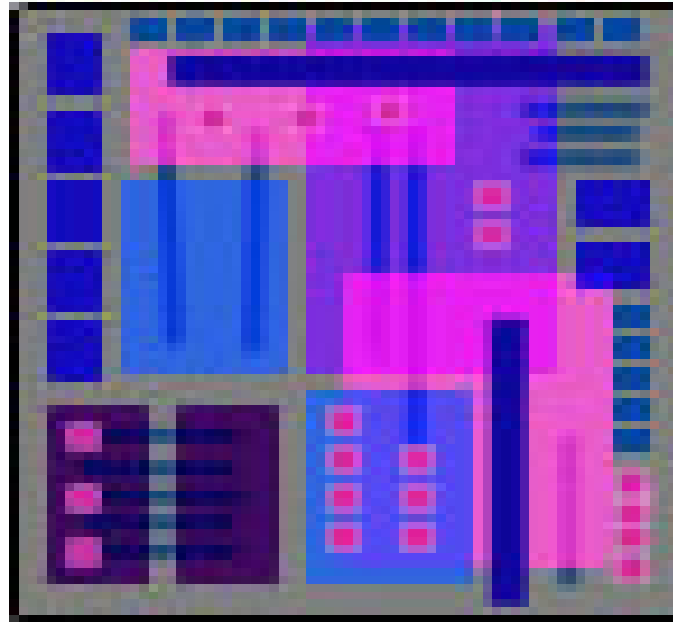
- Moore's Law brings opportunity
  - Gains in performance ...
  - ... and capabilities.
  - What has 20+ years of development brought us?
- How can we use those transistors?
  - Microprocessors?
  - Stream processors
    - Today's commodity stream processor: the GPU



# The past: 1987

---

20 MIPS CPU  
1987



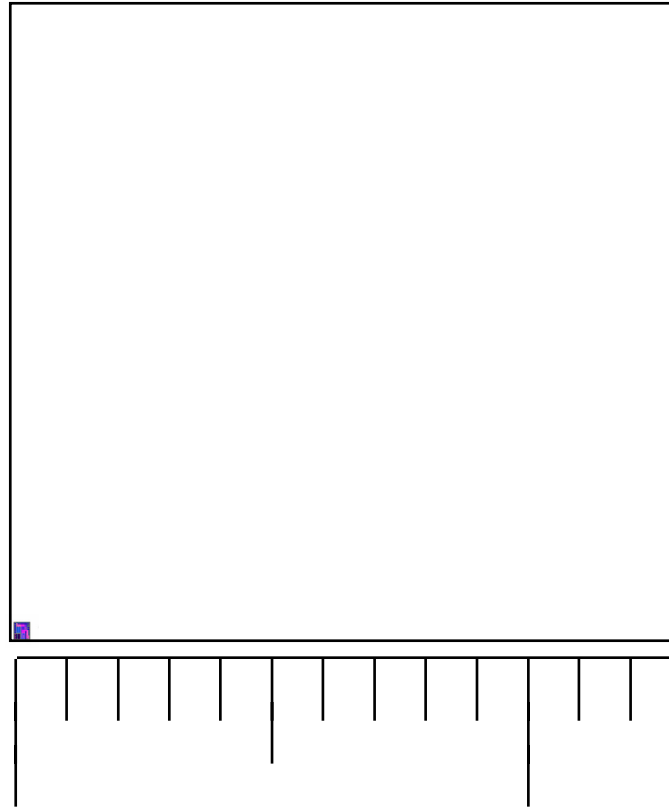
[courtesy Anant Agarwal]



# The future: 2007

---

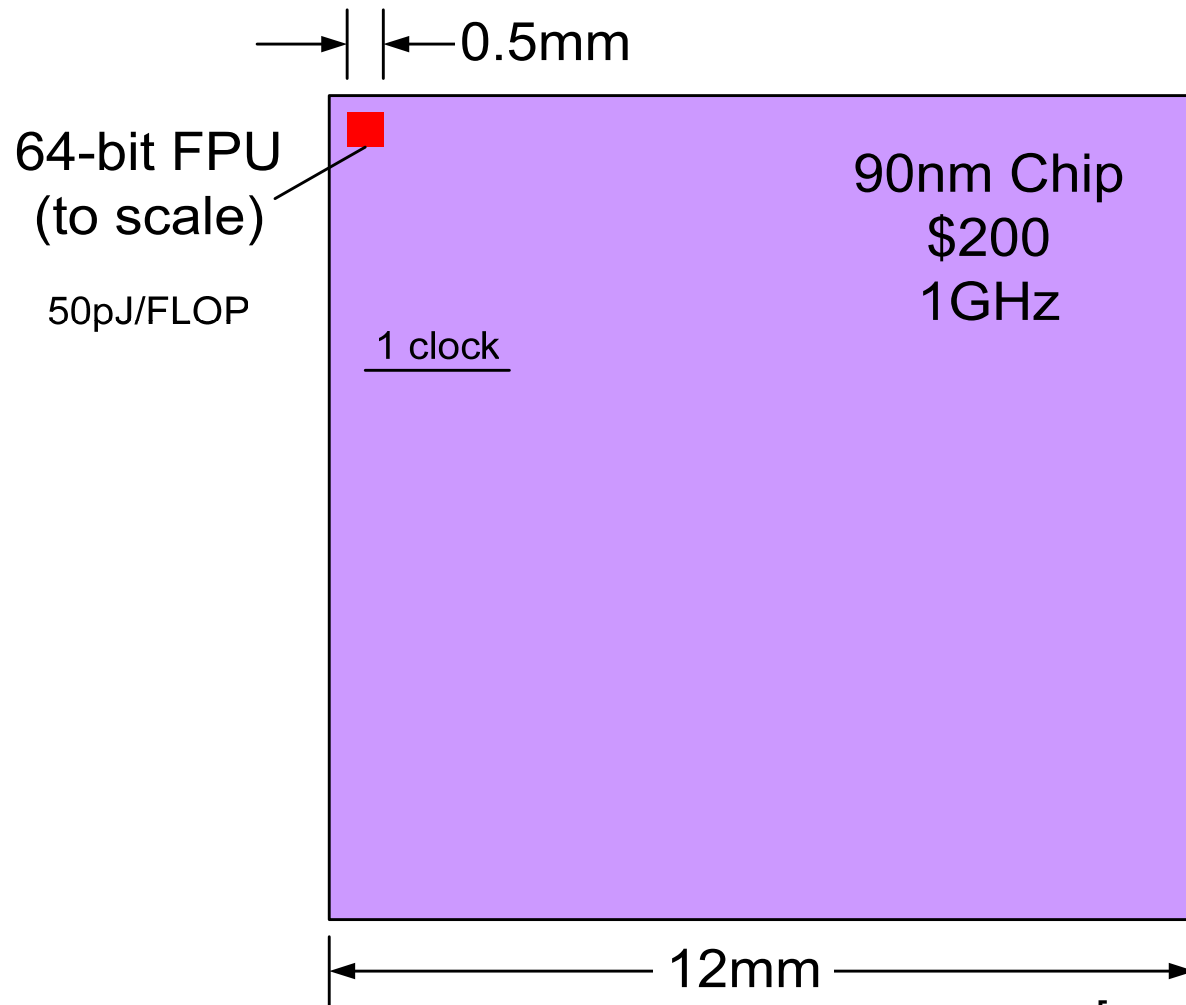
1 Billion Transistors  
2007



[courtesy Anant Agarwal]



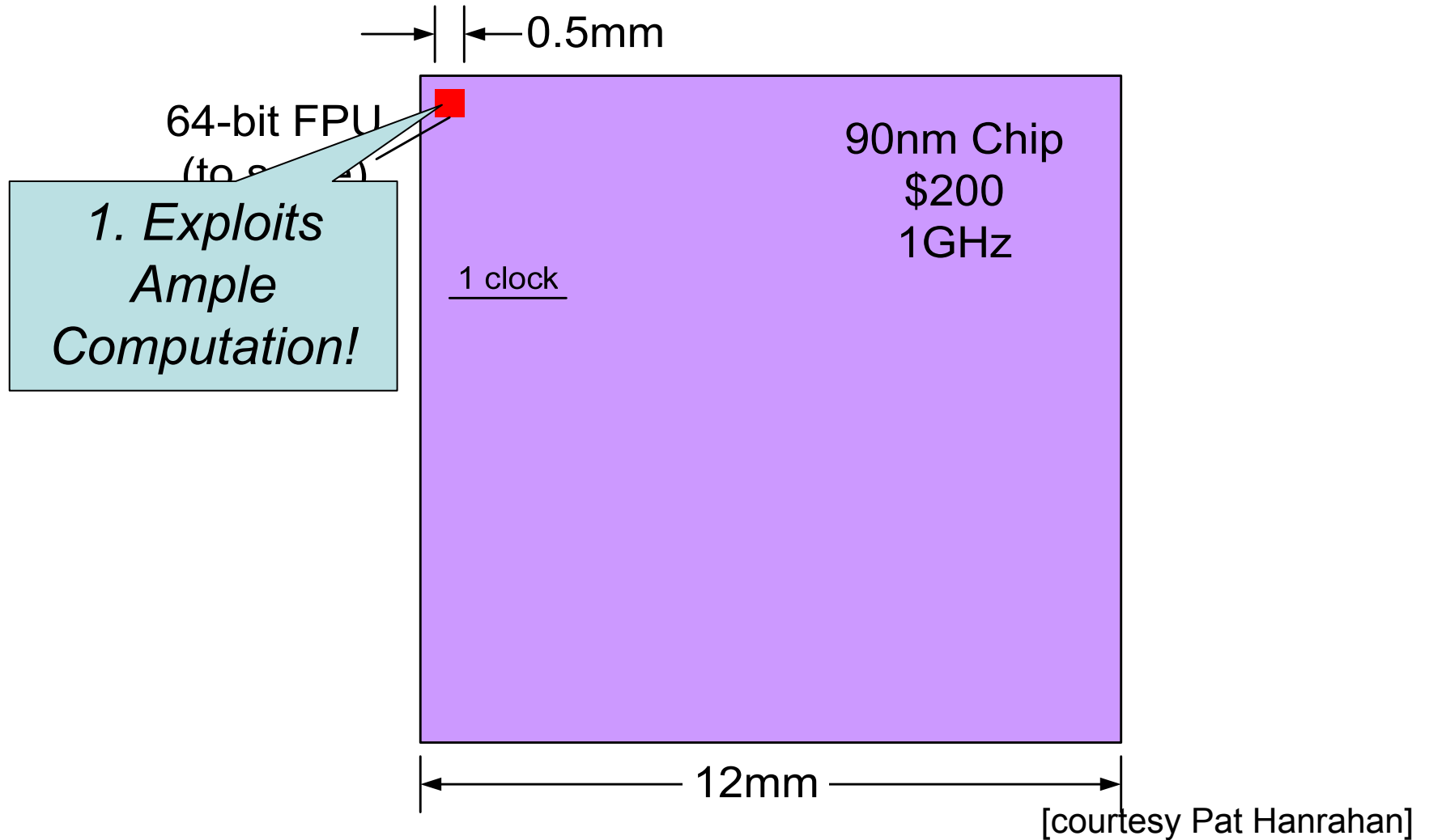
# Today's VLSI Capability



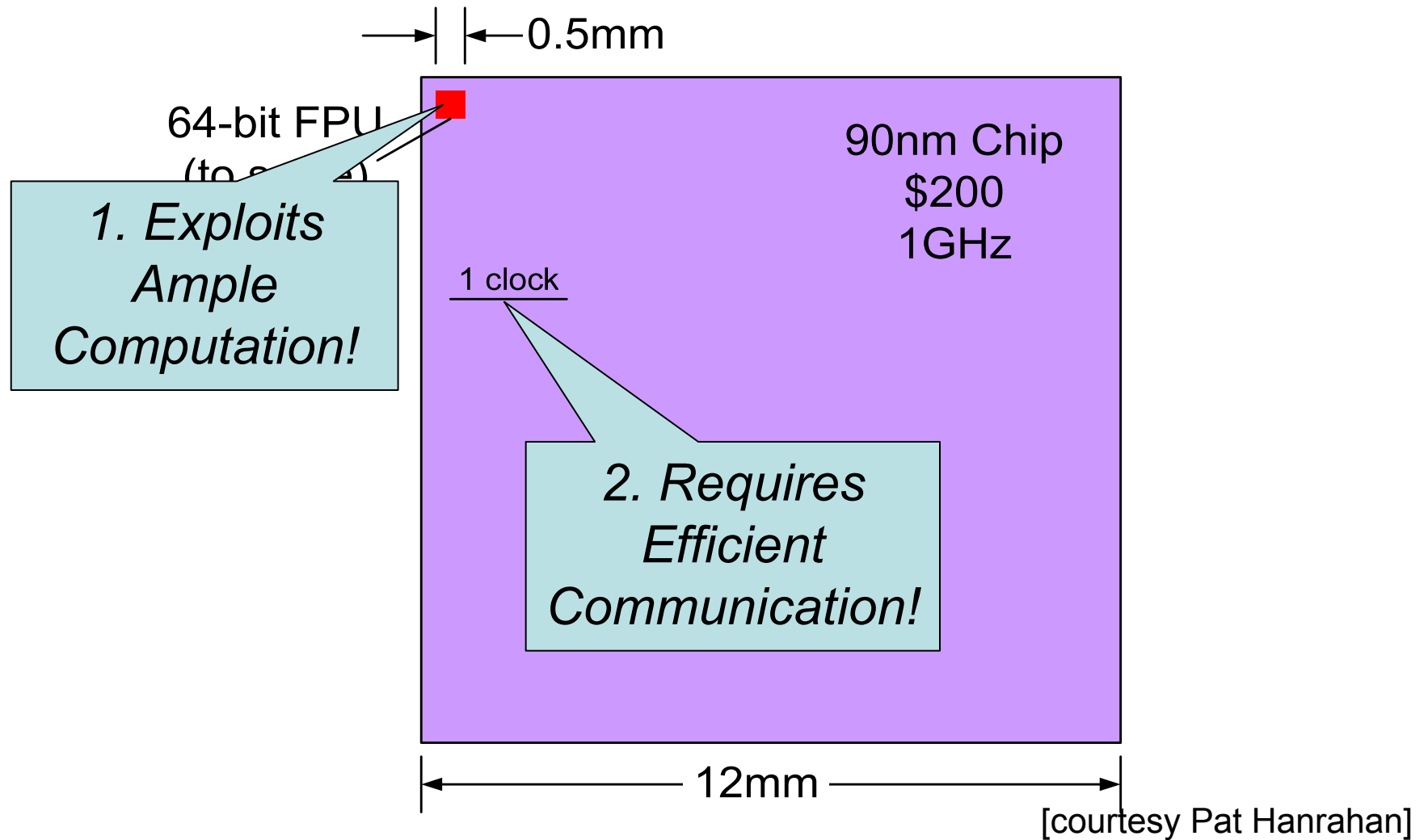
[courtesy Pat Hanrahan]



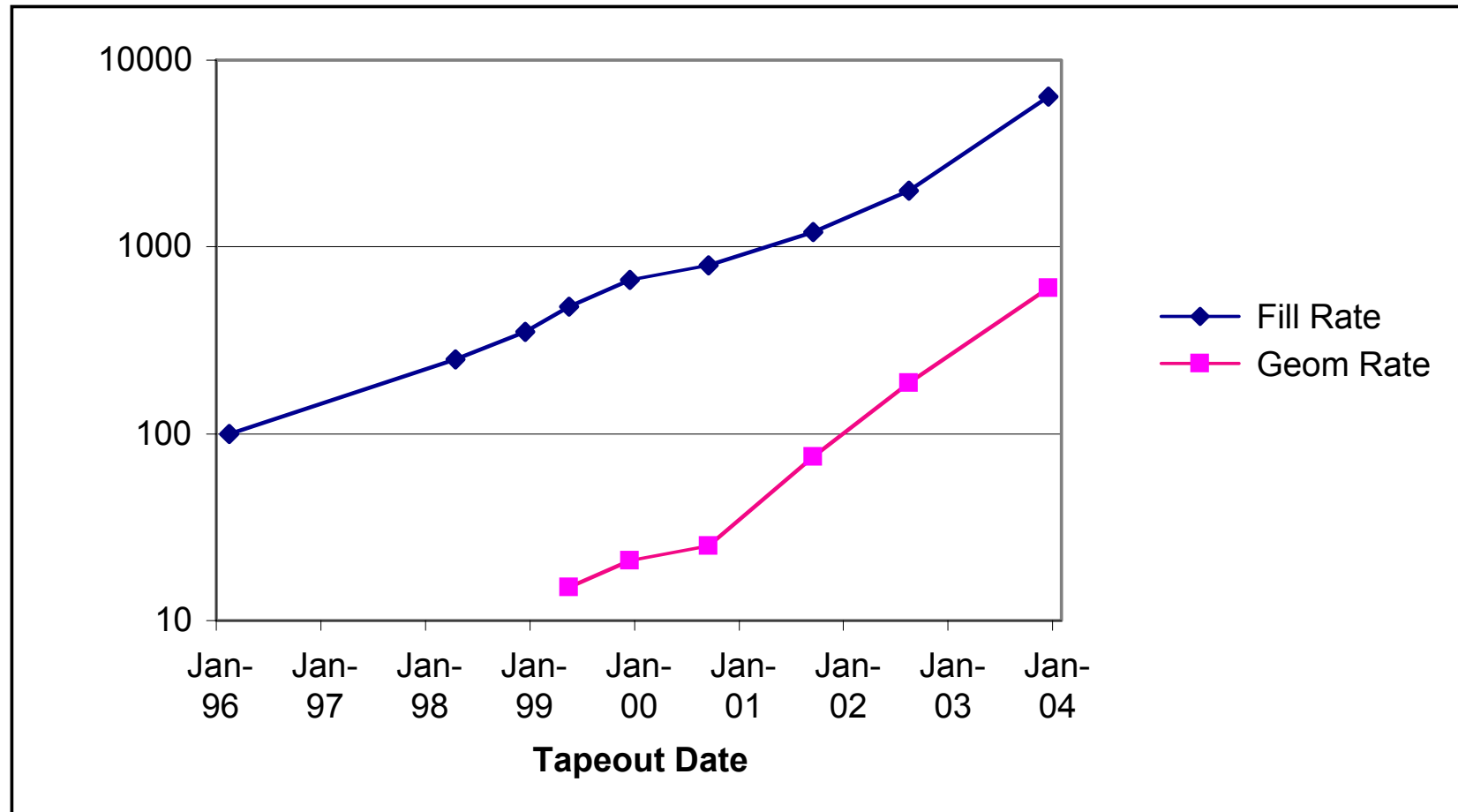
# Today's VLSI Capability



# Today's VLSI Capability



# NVIDIA Historicals

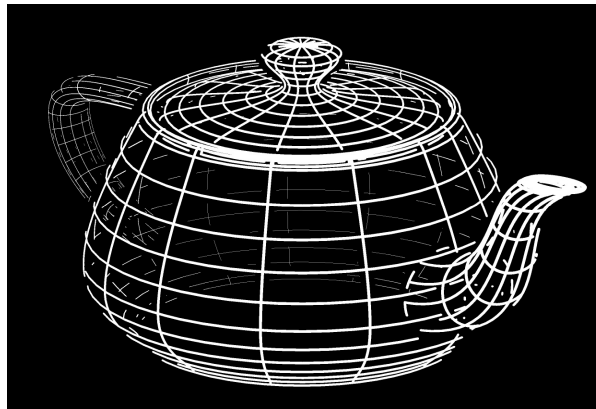


# GPU History: Features

---



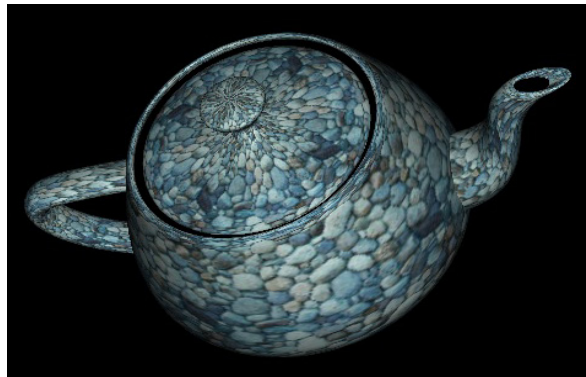
< 1982



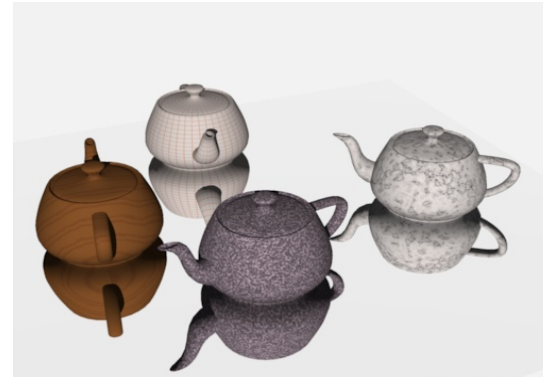
1982-87



1987-92



1992-2000



2000-



# Outline

---

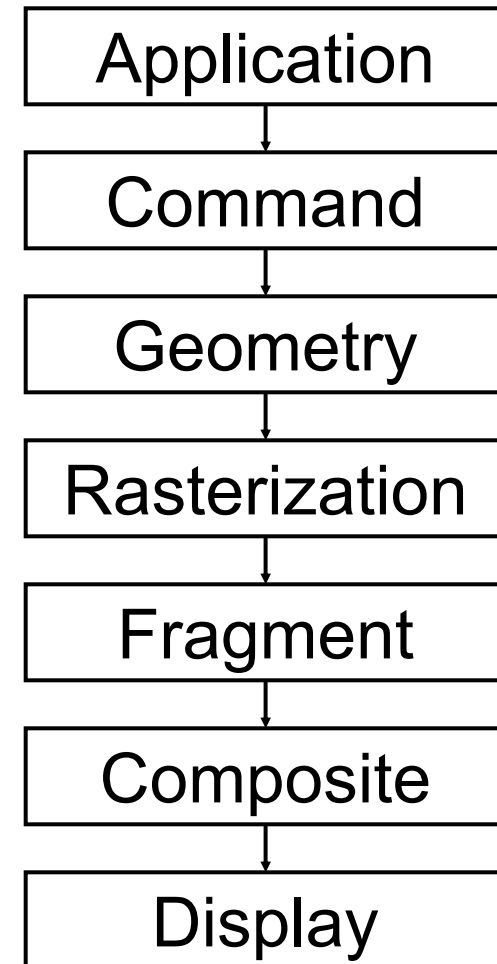
- Moore's Law brings opportunity
  - Gains in performance ...
  - ... and capabilities.
  - What has 20+ years of development brought us?
- How can we use those transistors?
  - Microprocessors?
  - Stream processors
    - Today's commodity stream processor: the GPU



# Characteristics of Graphics Apps

---

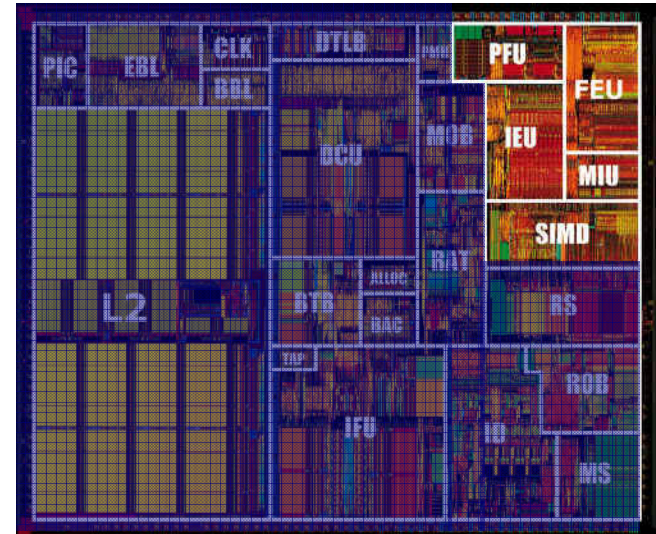
- Lots of arithmetic
- Lots of parallelism
- Simple control
- Multiple stages
- Feed forward pipelines
- Latency-tolerant / deep pipelines
  
- What other applications have these characteristics?



# Microprocessors: A Solution?

---

- Microprocessors address a different application space
  - Scalar programming model with no native data parallelism
    - Excel at control-heavy tasks
    - Not so good at data-heavy, regular applications
  - Few arithmetic units – little area
  - Optimized for low latency not high bandwidth
- Maybe the scalar processing model isn't the best fit



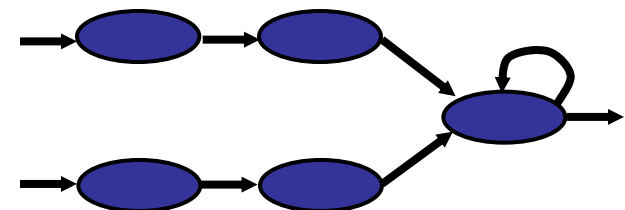
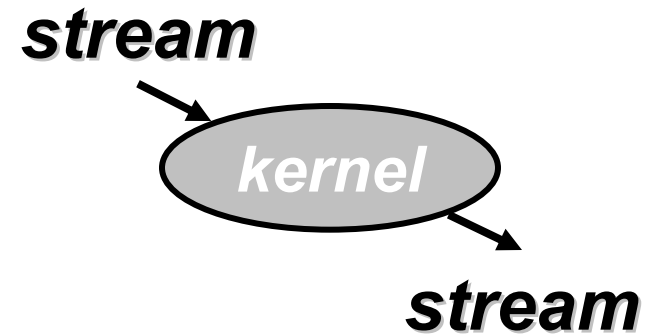
Pentium III – 28.1M T



# Stream Programming Abstraction

---

- Let's think about our problem in a new way
- Streams
  - Collection of data records
  - All data is expressed in streams
- Kernels
  - Inputs/outputs are streams
  - Perform computation on streams
  - Can be chained together



# Why Streams?

---

- Ample computation by exposing parallelism
  - Streams expose data parallelism
    - Multiple stream elements can be processed in parallel
  - Pipeline (task) parallelism
    - Multiple tasks can be processed in parallel
  - Kernels yield high arithmetic intensity
- Efficient communication
  - Producer-consumer locality
  - Predictable memory access pattern
    - Optimize for throughput of all elements, not latency of one
    - Processing many elements at once allows latency hiding



# Taxonomy of Streaming Processors

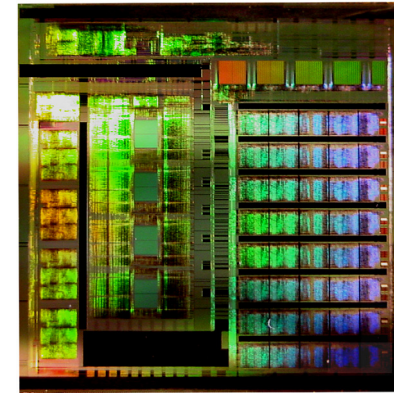
---

- In common:
  - Exploit parallelism for high computation rate
    - Each stage processes many items in parallel (d.p.)
    - Several stages can run at the same time (t.p.)
  - Efficient communication
    - Minimize memory traffic
    - Optimized memory system
- What's different?
  - Mapping of processing units to tasks in graphics pipeline

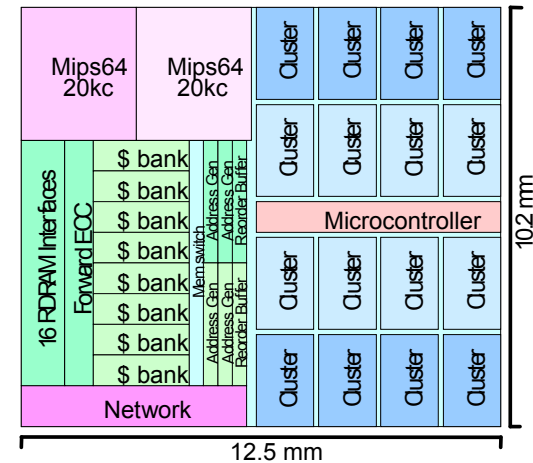


# Stream Processors

- Fewer processing units than tasks
  - “Time multiplexed” organization
  - Each stage fully programmable
- Stanford Imagine
  - 32b stream processor for image, signal, and graphics processing (2001)
- Stanford Merrimac
  - 64b stream processor for scientific computing (2004)
  - Core of Stanford Streaming Supercomputer
- Challenge:
  - Efficiently mapping all tasks to one processing unit - no specialization



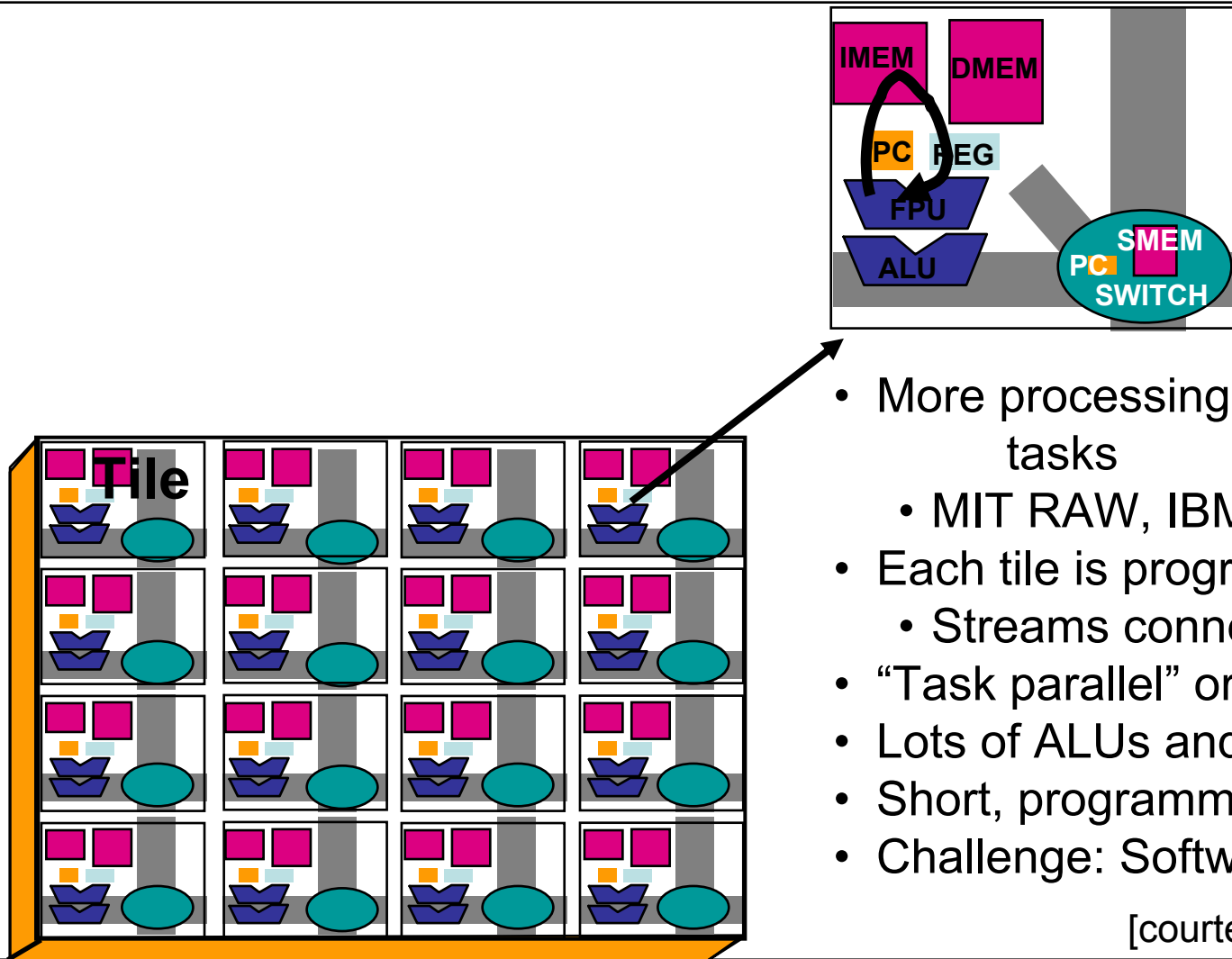
[Stanford Imagine - 2001]



[Stanford Merrimac - 2004]



# MIT RAW: Tiled Processor Architecture

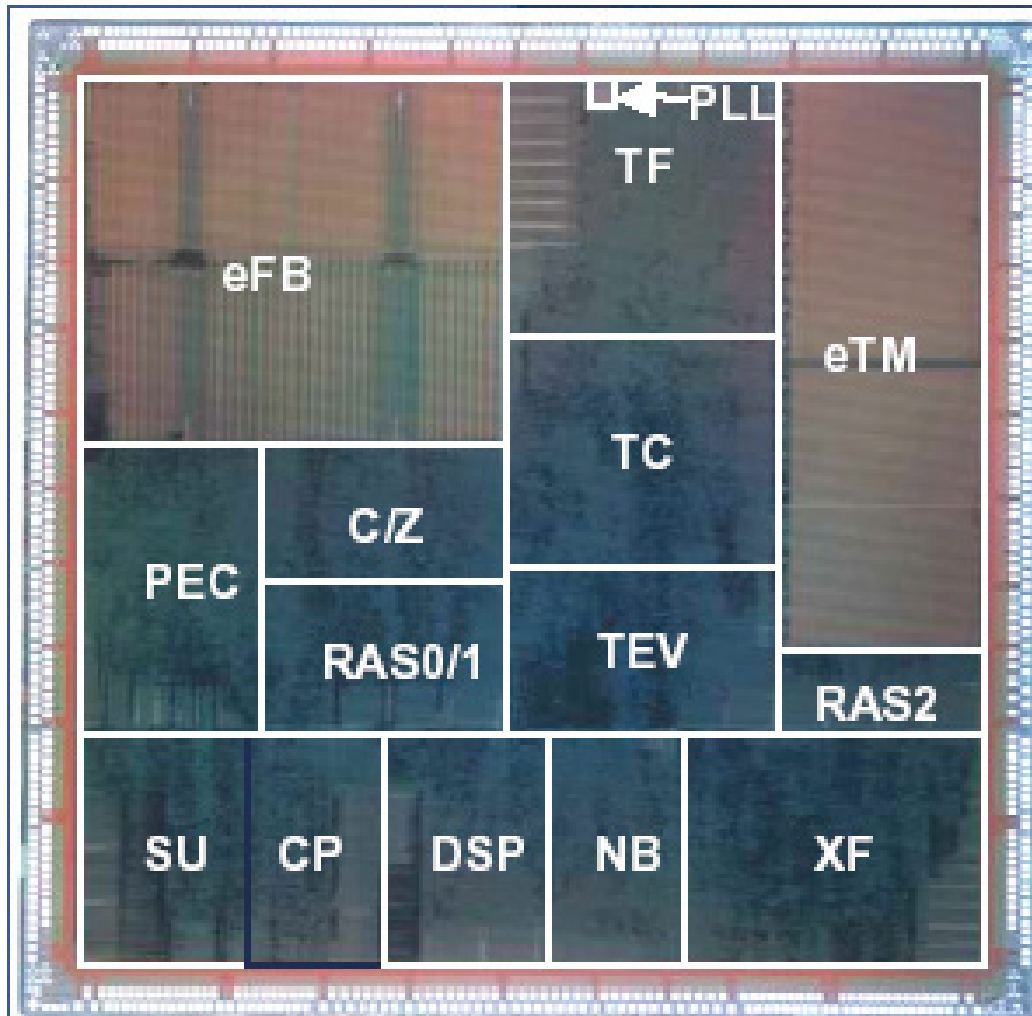


- More processing units than tasks
  - MIT RAW, IBM Cell
- Each tile is programmable
  - Streams connect tiles
- “Task parallel” organization
- Lots of ALUs and registers
- Short, programmable wires
- Challenge: Software support

[courtesy Anant Agarwal]



# GPU: Special-Purpose Graphics Hardware



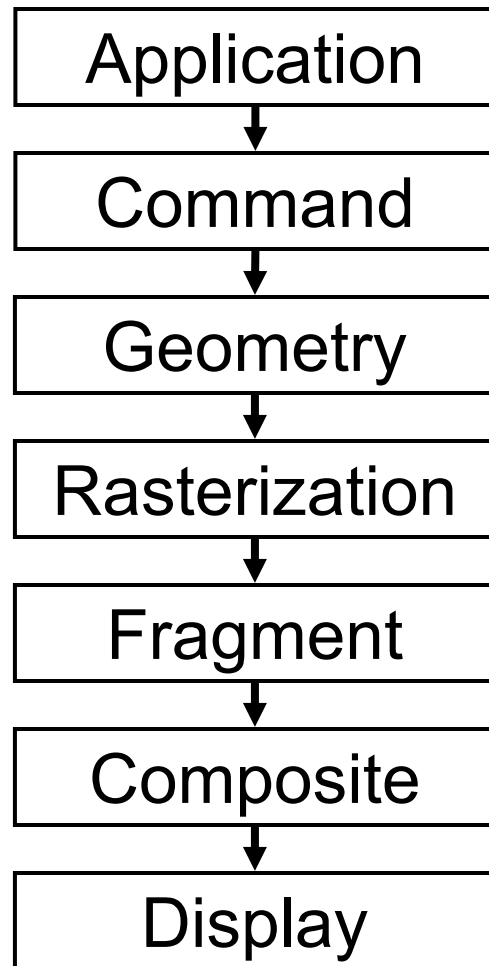
- Task-parallel organization
- Assign each task to processing unit
- Hardwire each unit to specific task - huge performance advantage!
- Provides ample computation resources
- Efficient communication patterns
- Dominant graphics architecture

[ATI Flipper – 51M T]



# Today's Graphics Pipeline

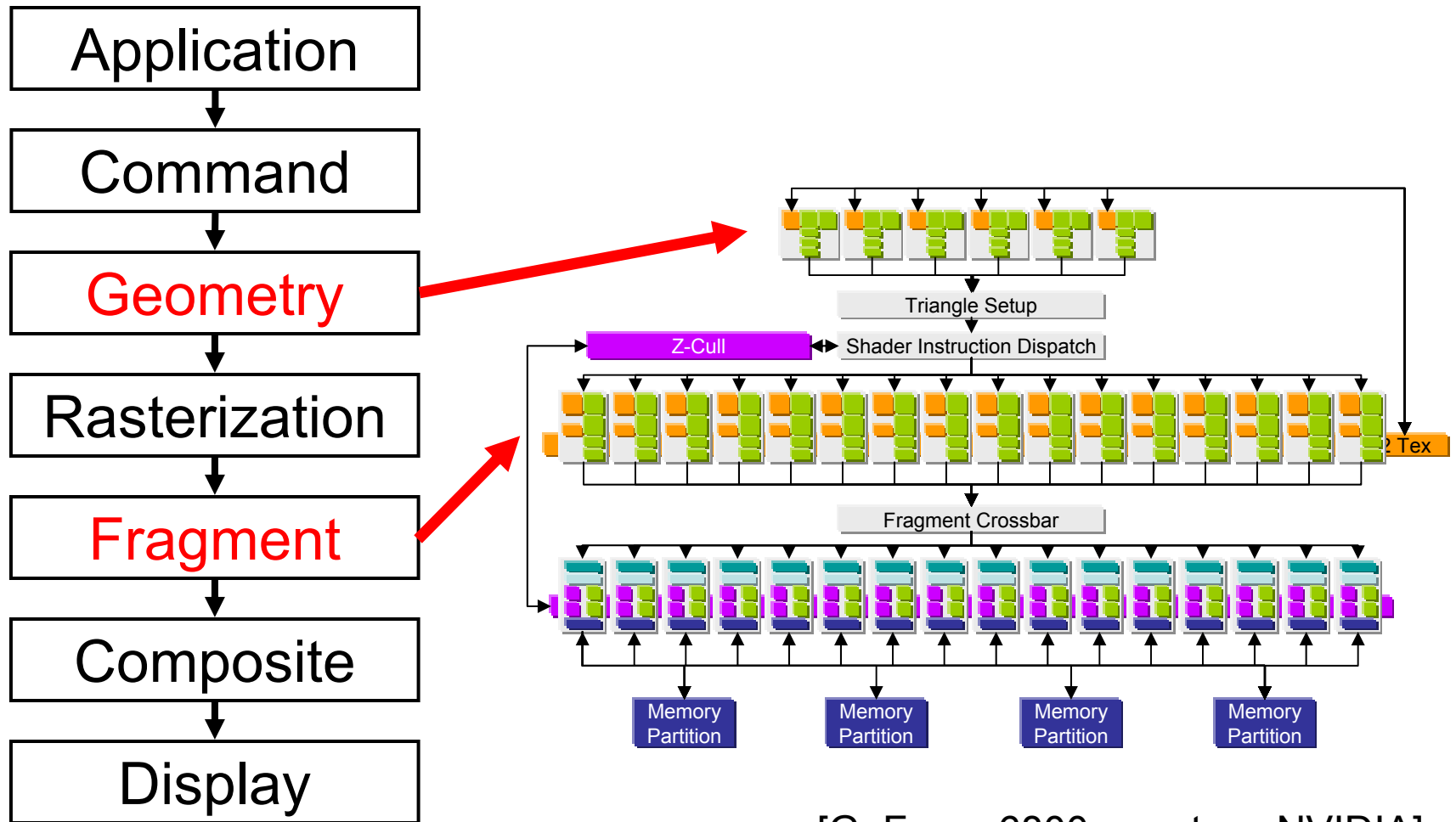
---



- Graphics is well suited to:
  - The stream programming model
  - Stream hardware organizations
    - GPUs are a commodity stream processor!
- What if we could apply these techniques to more general-purpose problems?
  - GPUs should excel at tasks that require ample computation and efficient communication.
- What's missing?



# The Programmable Pipeline



[GeForce 6800, courtesy NVIDIA]



# Conclusions

---

- Adding programmability to GPUs is exciting!
  - Confluence: Lots of computation; expertise in harnessing that computation; commodity production; programmability
  - GPUs have great performance
    - Computation & communication
  - Programmability allows them to address many interesting problems
- Many challenges remain ...
  - Algorithms, programming models, architecture, languages, tools ...
- Next speaker:
  - Aaron Lefohn
  - “The GPGPU Programming Model”

