# Timeline for CPUs

**80's: CPU and system same speed. Zero wait states.**

**1993: CPUs faster than the rest of the system. Rapid raise of frequency.**

**Late 90's to present: Multi-CPU systems, multi-core CPUs.**

**CPUs are still improving, but going for higher frequency is not as obvious as before.**

# Meanwhile, at the graphics dept

**80’s: Hardware sprites. Push pixels with low-level code.**

**1993: Textured 3D games: Wolf3D, Doom.**

**Early 90’s: Professional 3D boards.**

**1996: 3dfx Voodoo1!**

**2001: Programmable shaders.**

**2006: G80, unified architecture. CUDA**

**2009: OpenCL.**

**2010: Fermi architecture**

**2012: Kepler architecture**

| | 1995 | 2005 | |
|---|---|---|---|
| CPU Frequency (GHz) | .1 | 3.2 | 32x |
| Memory Frequency (GHz) | .03 | 1.2 | 40x |
| Bus Bandwidth (GB/sec) | .1 | 4 | 40x |
| Hard Disk Size (GB) | .5 | 200 | 400x |

| | 1995 | 2005 | |
|---|---|---|---|
| CPU Frequency (GHz) | .1 | 3.2 | 32x |
| Memory Frequency (GHz) | .03 | 1.2 | 40x |
| Bus Bandwidth (GB/sec) | .1 | 4 | 40x |
| Hard Disk Size (GB) | .5 | 200 | 400x |
| Pixel Fill Rate (GPixels/sec) | .0004 | 3.3 | 8250x |
| Vertex Rate (GVerts/sec) | .0005 | .35 | 700x |
| Graphics flops (GFlops/sec) | .001 | 40 | 40000x |
| Graphics Bandwidth (GB/sec) | .3 | 19 | 63x |
| Frame Buffer Size (MB) | 2 | 256 | 128x |

# How about 2005-2011?

| | 2005 | 2011 | |
|---|---|---|---|
| CPU Frequency (GHz) | 3.2 | 3.8 | 1.18x |
| Memory Frequency (GHz) | 1.2 | 2.0 | 1.67x |
| Bus Bandwidth (GB/sec) | 4 | 31 | 7.75x |
| Hard Disk Size (GB) | 200 | 4000 | 20x |

| | 2005 | 2011 | | |
|---|---|---|---|---|
| CPU Frequency (GHz) | 3.2 | 3.8 | 1.18x | x cores? |
| Memory Frequency (GHz) | 1.2 | 2.0 | 1.67x | |
| Bus Bandwidth (GB/sec) | 4 | 31 | 7.75x | |
| Hard Disk Size (GB) | 200 | 4000 | 20x | |
| Pixel Fill Rate (GPixels/sec) | 3.3 | 59 | 18x | |
| Vertex Rate (GVerts/sec) | .35 | ? | ? | |
| Graphics flops (GFlops/sec) | 40 | 2488 | 62x | |
| Graphics Bandwidth (GB/sec) | 19 | 327.7 | 17x | |
| Frame Buffer Size (MB) | 256 | 3000 | 12x | |

**And what about 2011-2015 then?**

**CPU frequency: No change**
**Hard disk size: 8-16 TB**
**GPU GFLOPS: Up from 2500 to 7000**

**many other data tricky to find**

**But is this a fair comparison?**
**Let us compare apples with apples:**
**GFLOPS for both!**

| | GPU | CPU |
|---|---|---|
| 1995: | 0.001 | 0.09 |
| 2005: | 40 | 5.6 |
| 2011: | 2488 | 91 |
| 2015: | 7000 | 176 |

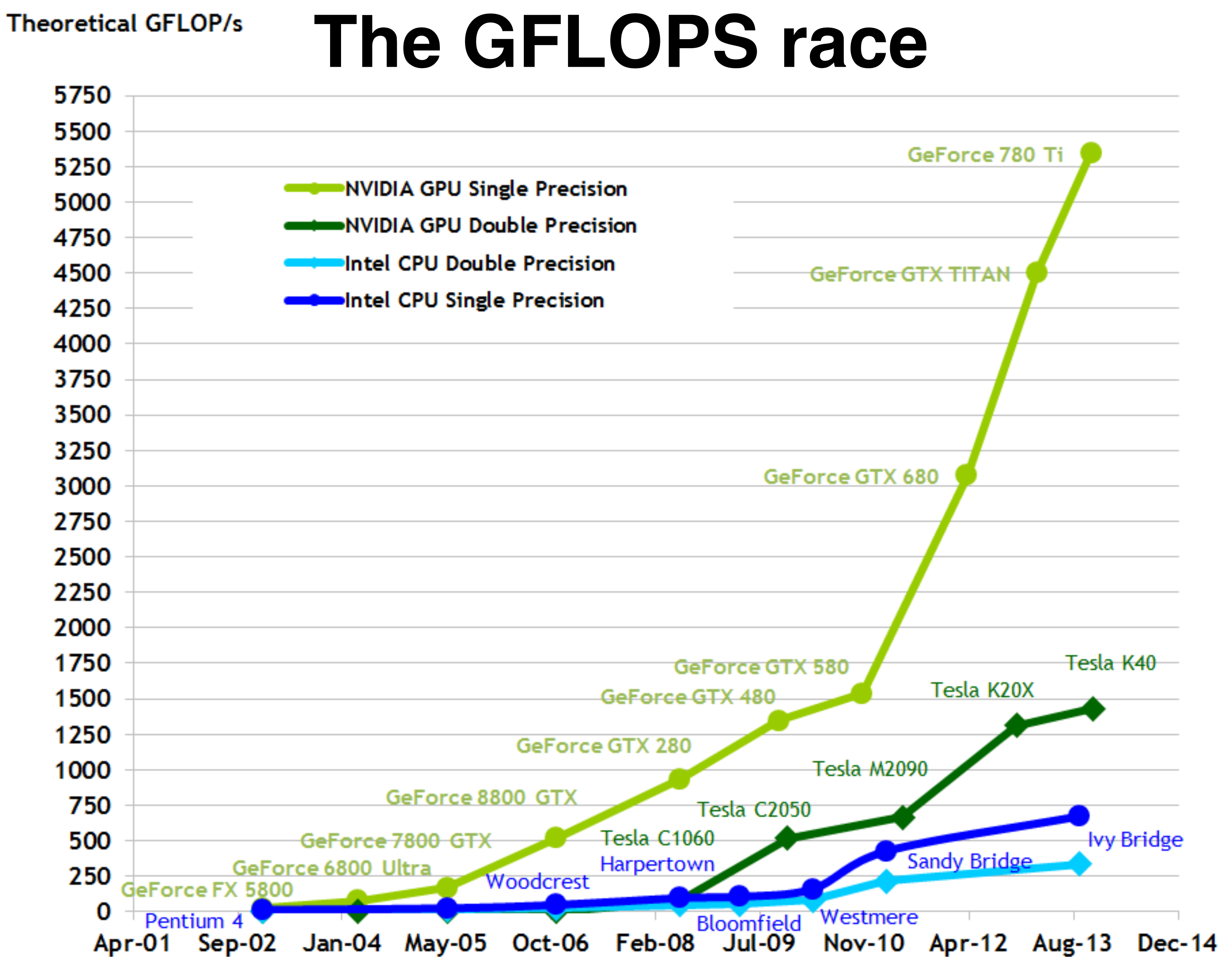
**(Various sources)**

# How about economy: GFLOPS per dollar?

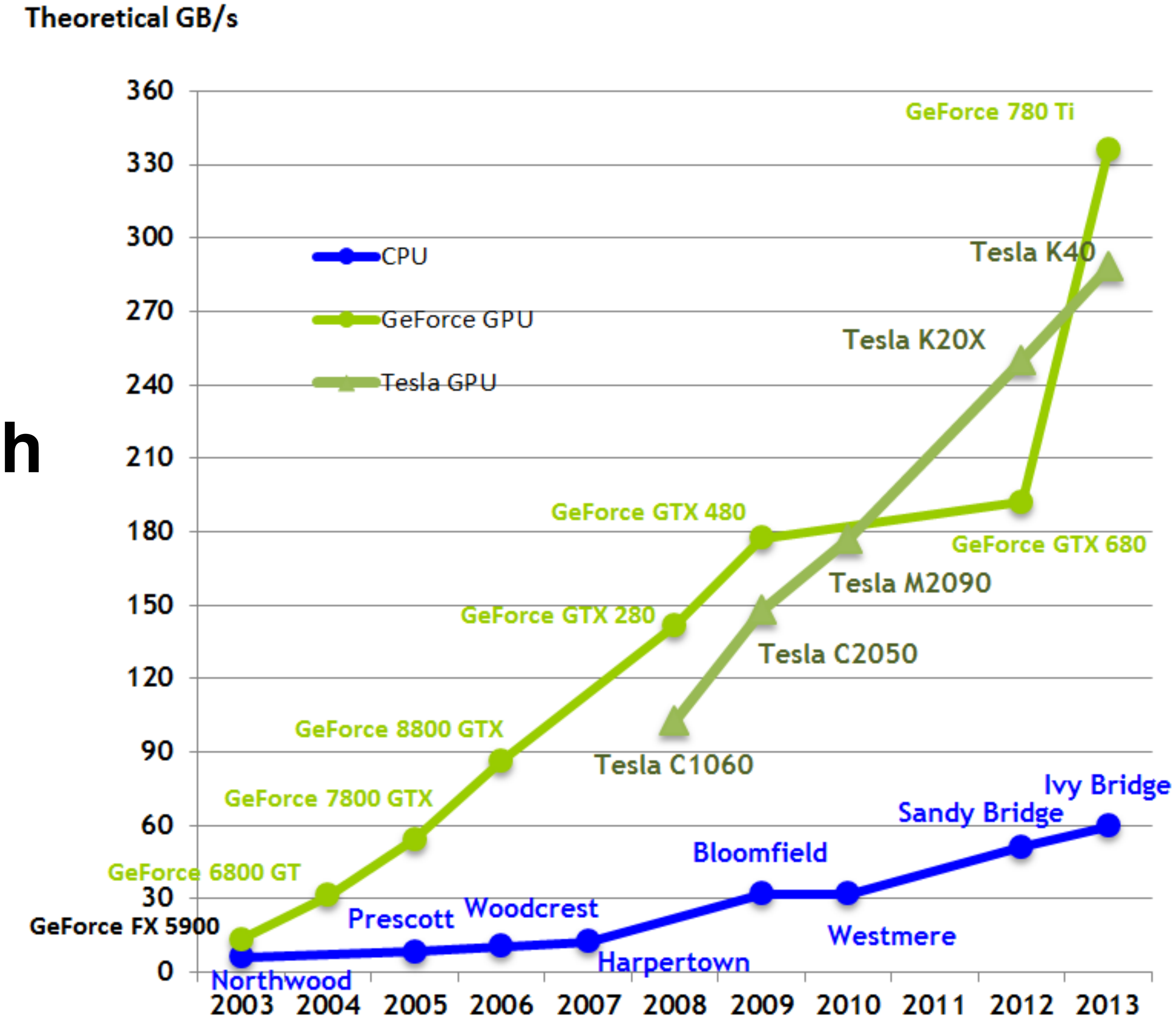
| | |
|---|---|
| 1961: | 8.3 trillion |
| 1984: | 42 million |
| 1997: | 42000 (CPU cluster) |
| 2000: | 836-1300 |
| 2007: | 52 |
| 2012: | 0.73 (AMD 7970) |
| 2013: | 0.22 (PS4) |
| 2015: | 0.08 (Radeon R9 295) |

(Wikipedia)

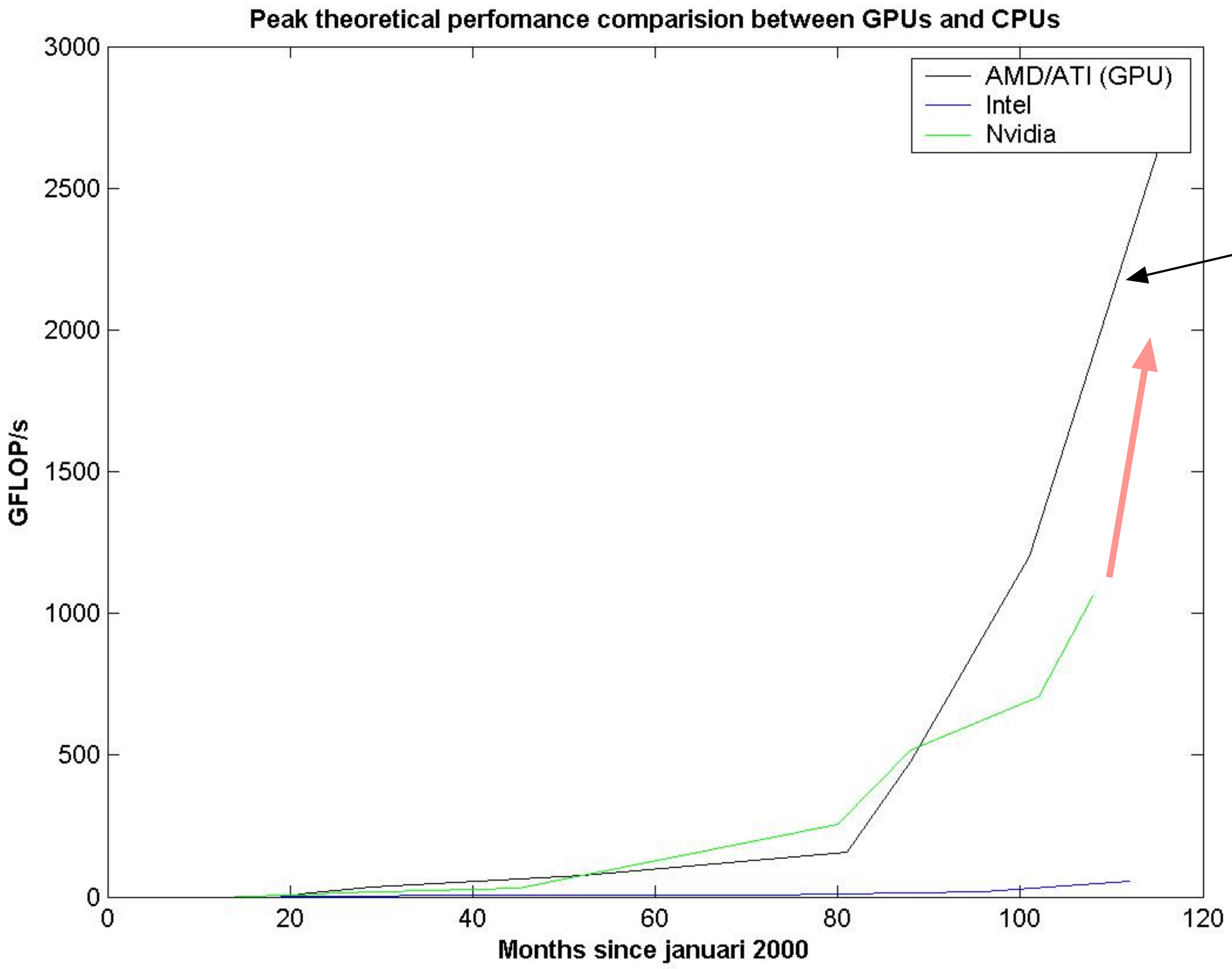
# The GFLOPS race



# The bandwidth race



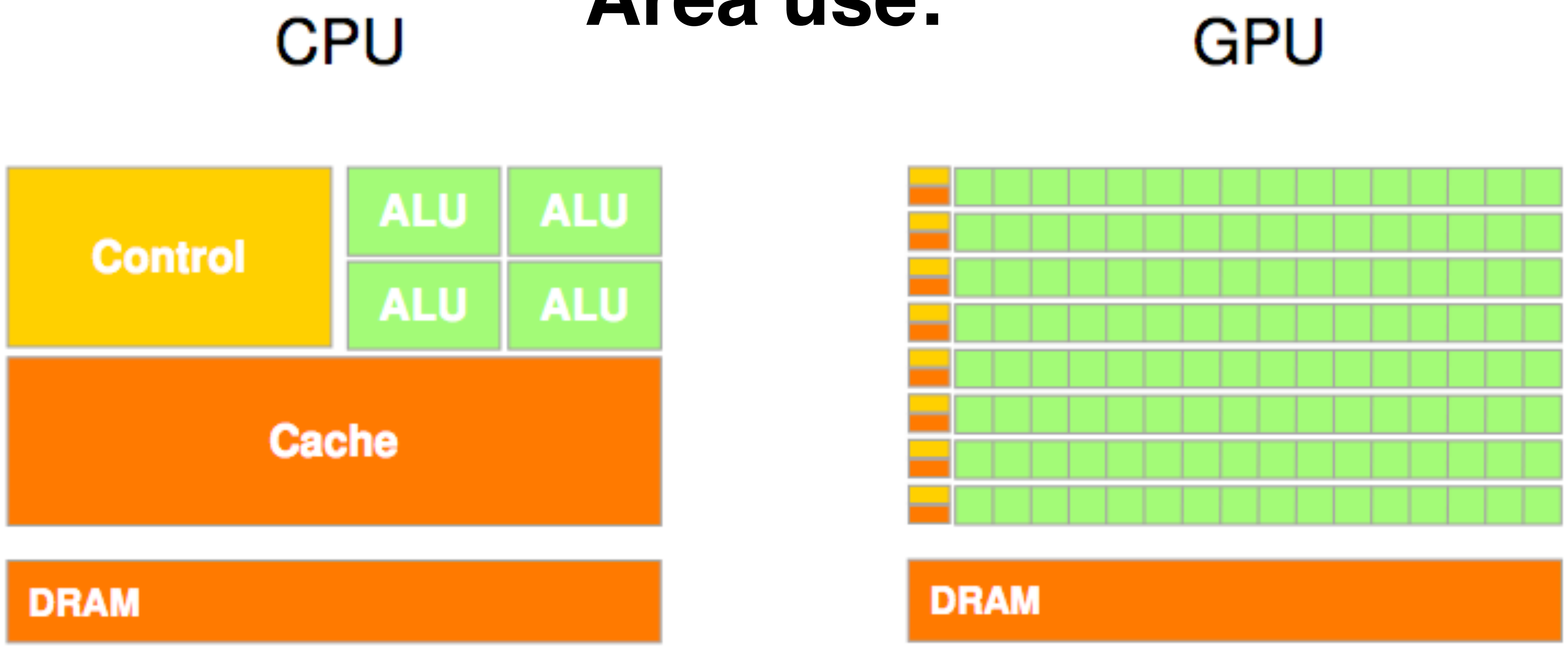
# Another graph, including ATI/AMD




**AMD took the lead in single precision while NVidia was chasing for double with Fermi**

# How is this possible?

## Area use:




**But in particular: SIMD architecture**

# Flynn's taxonomy

| | |
|---|---|
| **SISD**<br>Single instruction, single data<br>Old single-core systems | **MISD**<br>Multiple instruction, single data<br>Multiple for redundance |
| **SIMD**<br>Single instruction, multiple data<br>GPUs, vector processors | **MIMD**<br>Multiple instruction, multiple data<br>Multi-core CPUs |

Plus SIMT, single instruction, multiple threads

# SIMD

Single instruction, multiple data

Simplifies instruction handling. All cores get the same instruction.

Excellent for operations where one operation must be made on many data elements.

Is that so common? Yes!

Data best in stored arrays.

# Data Oriented Programming

DOP optimizes for performance.

Data structures selected to fit the computations, instead of the programmer!

Optimize for the end user instead for the programmer!

Popular in the game industry - why not elsewhere?

# SIMT - Single Instruction, Multiple Thread

A variant of SIMD.

Parallelism expressed as threads.

A programming model, but also demands that the hardware can handle threads very fast.

Threads dependent - executed in a SIMD processor!

So, why does SIMT fit a graphics processor so well?

# Why did GPUs get so much performance?

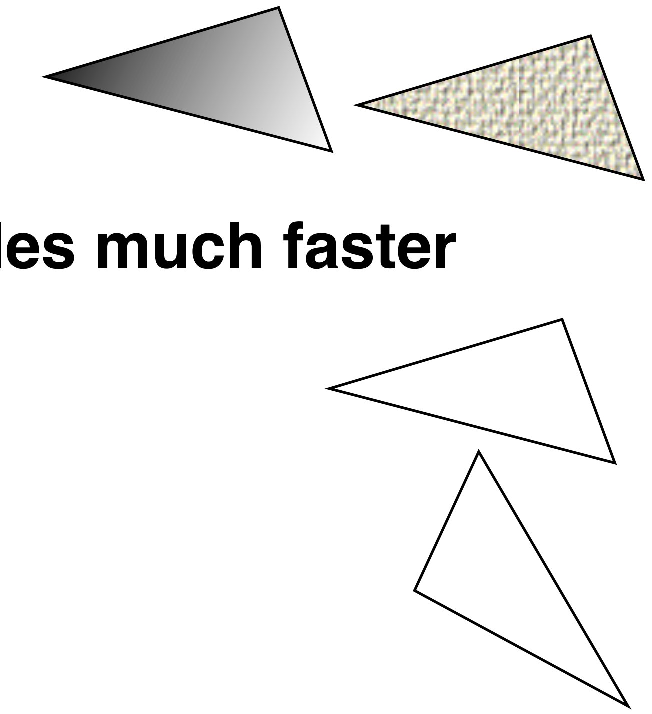
Early problem with large amounts of data. (Complex geometry, millions of output pixels.)

Graphics pipeline designed for parallelism!

Hiding memory latency by parallelism

Volume. 3D graphics boards central component in game industry. Everybody wants one!

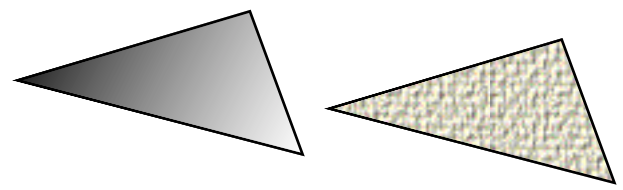
New games need new impressive features. Many important advancements started as game features.



***Must process many pixels fast!***

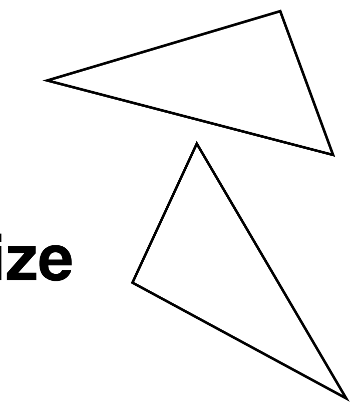
**Early GPUs could draw textured, shaded triangles much faster than the CPU.**

***Must process many pixels fast!***

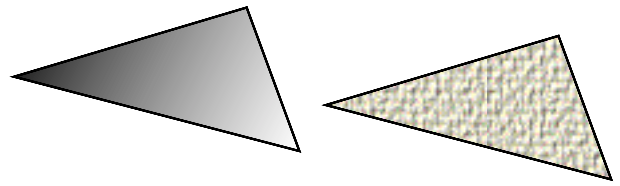


**Early GPUs could draw textured, shaded triangles much faster than the CPU.**

***Must do matrix multiplication and divisions fast.***

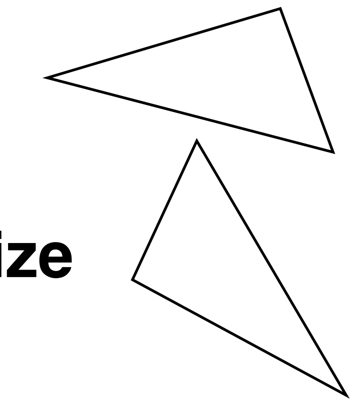


**Next generation could transform vertices and normalize vectors.**



***Must process many pixels fast!***

**Early GPUs could draw textured, shaded triangles much faster than the CPU.**



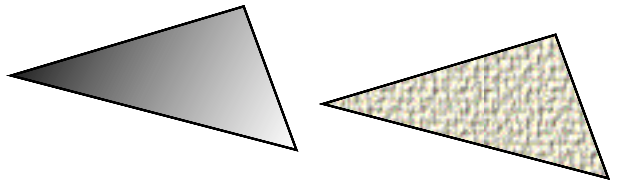
***Must do matrix multiplication and divisions fast.***

**Next generation could transform vertices and normalize vectors.**

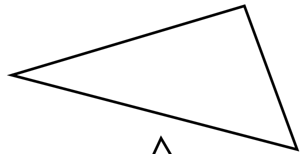
***Must have programmable parts.***

**This was added to make Phong shading and bump mapping.**

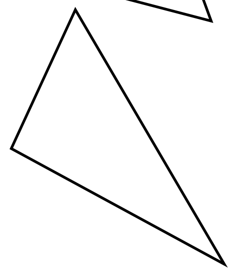
*Must process many pixels fast!*



Early GPUs could draw textured, shaded triangles much faster than the CPU.



*Must do matrix multiplication and divisions fast.*



Next generation could transform vertices and normalize vectors.

*Must have programmable parts.*

This was added to make Phong shading and bump mapping.

*Must work in floating-point!*

This was for light effects, HDR.

# So a GPU should

- process vertices, many in parallel, applying the same transformations on each

- process pixels (fragments) in parallel, applying the same color/light/texture calculations on each

**SIMD friendly problem!**

**Less control, control many calculations instead of one**

# A different kind of threads

**SIMD threads, all run the same program**

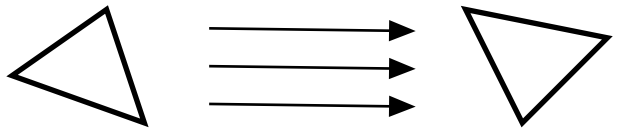
**Group-wise, they execute in parallel, SIMD-style**

**Made for graphics operations: Shader threads calculate one pixel or one vertex**

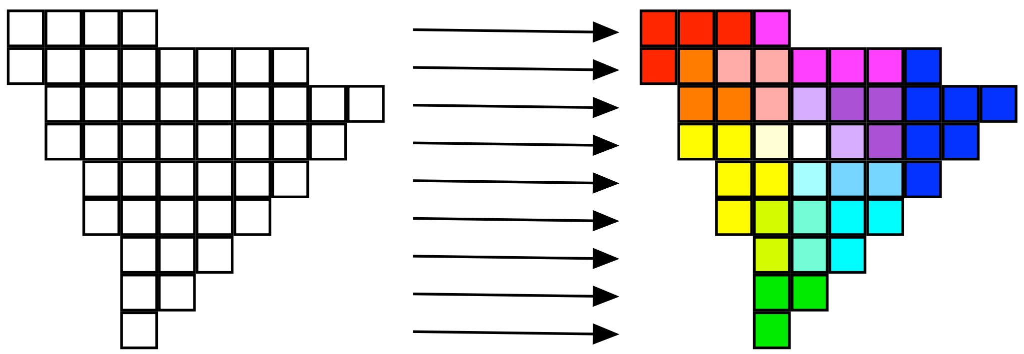
**CUDA/OpenCL threads may calculate anything, but typically one part of the output - in order**

## The main tasks in rendering graphics:

One thread per vertex
Same operations, same kernel, different data



One thread per pixel (fragment)
Same operations, same kernel, different data



# The 3D pipeline in the GPU

**Low-level operations from vertices to pixel data**