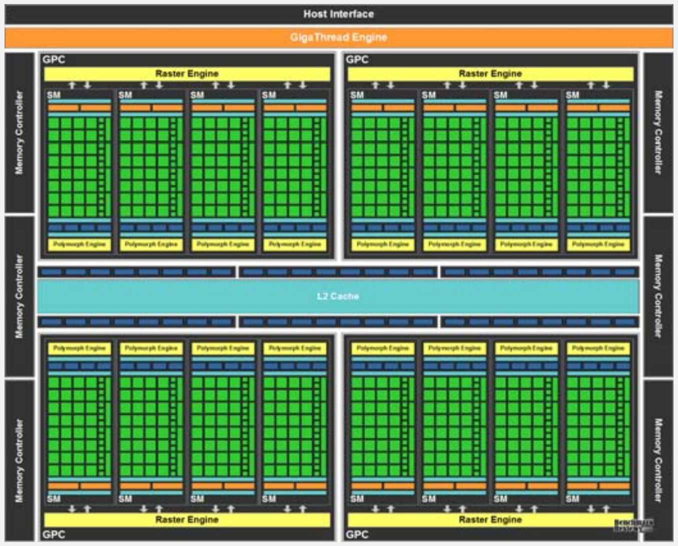
GPU 101

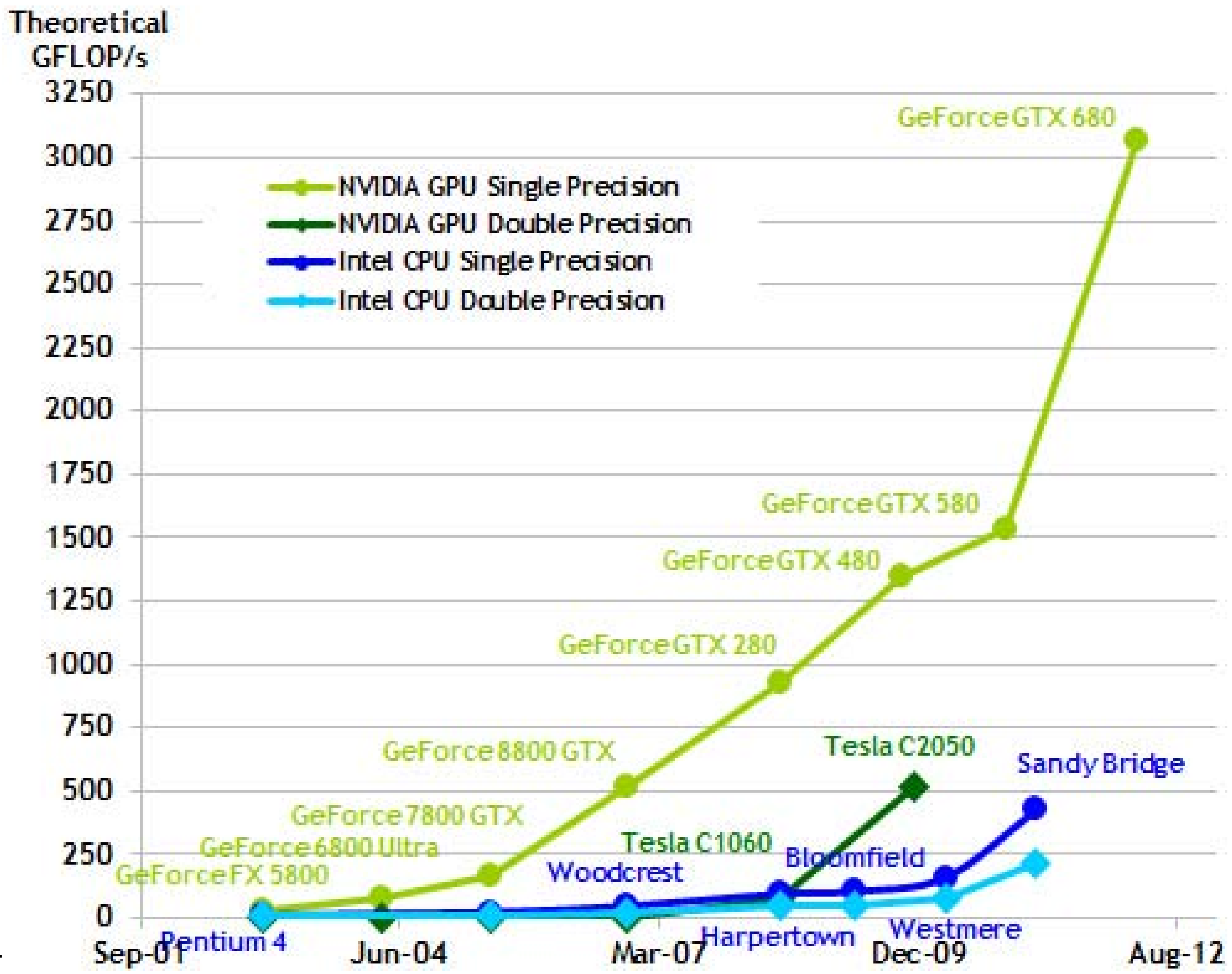
Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University



Why do we care about GPU Programming?
A History of GPU Performance vs. CPU Performance

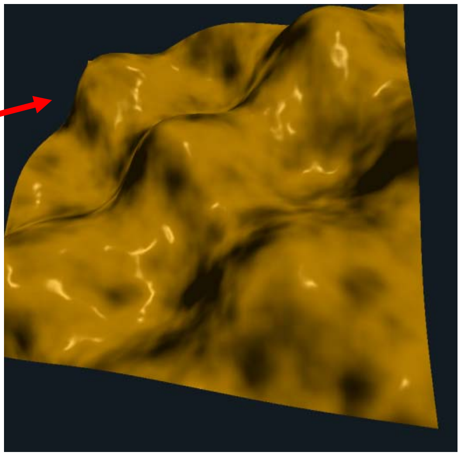


Source: NVIDIA

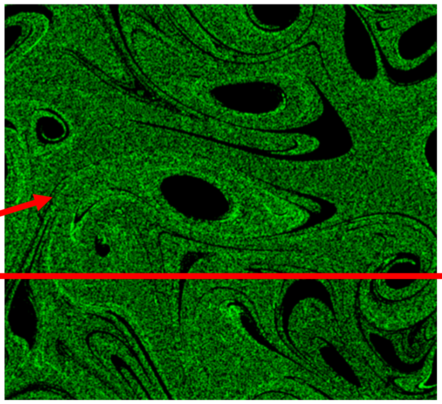
How Can You Gain Access to GPU Power?

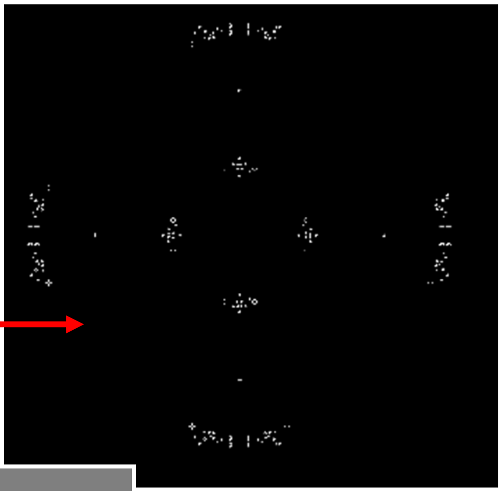
There are three ways:

1. Write a graphics display program (≥ 1985)



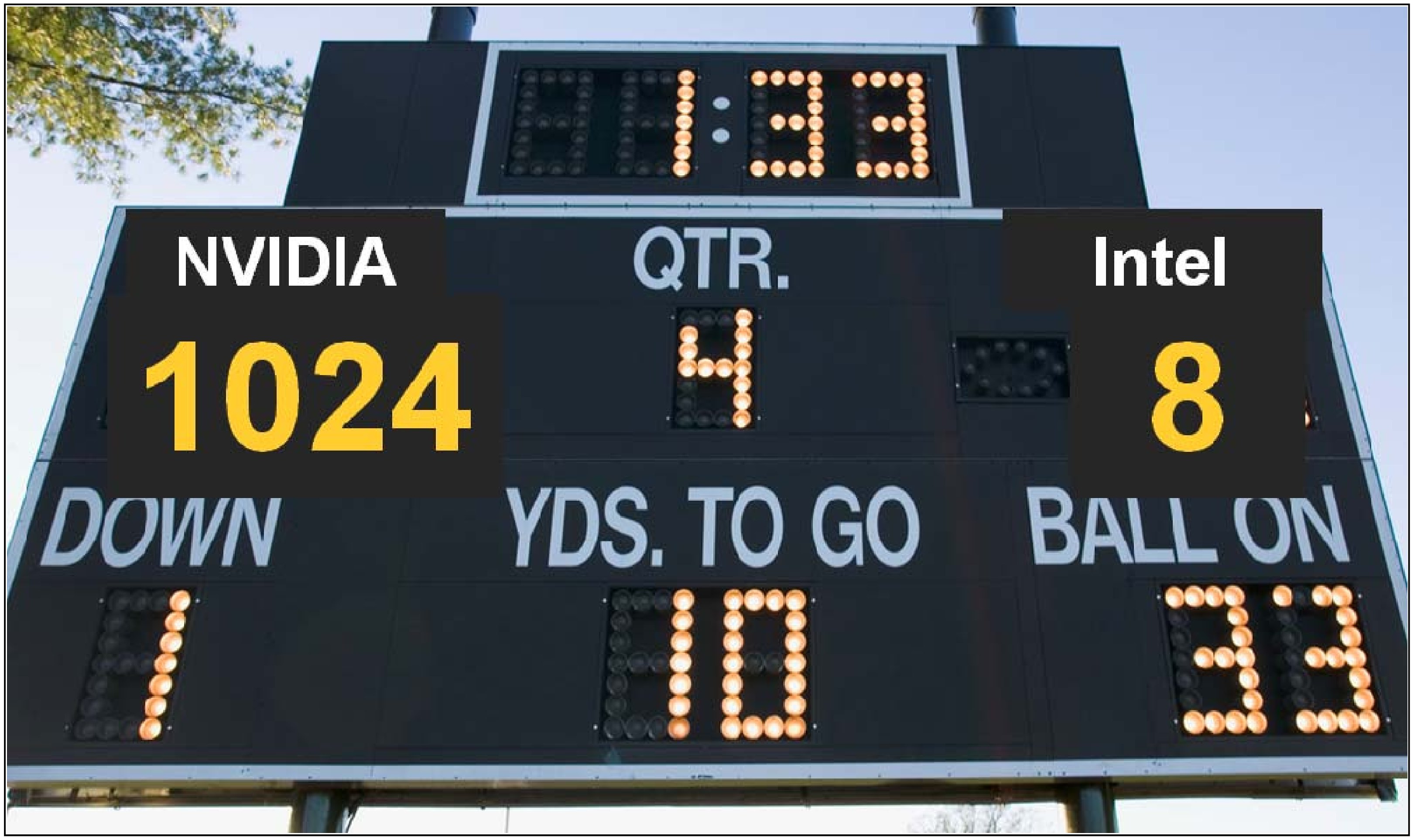
2. Write an application that looks like a graphics display program, but uses the fragment shader to do some computation (≥ 2002)





3. Write in OpenCL (or CUDA), which looks like C++ (≥ 2006)

The "Core-Score". How can this be?



Why have GPUs Been Outpacing CPUs in Performance?

Due to the nature of graphics computations, GPU chips are customized to handle **streaming data**.

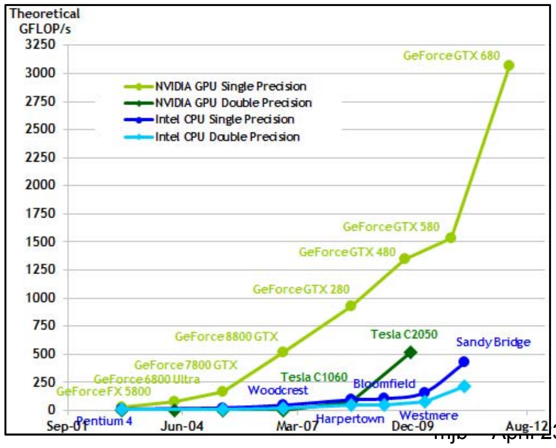
Another reason is that GPU chips do not need the significant amount of **cache** space that occupies much of the real estate on general-purpose CPU chips. The GPU die real estate can then be re-targeted to hold more cores and thus to produce more processing power.

Another reason is that general CPU chips contain on-chip logic to do **branch prediction.** This, too, takes up chip die space.

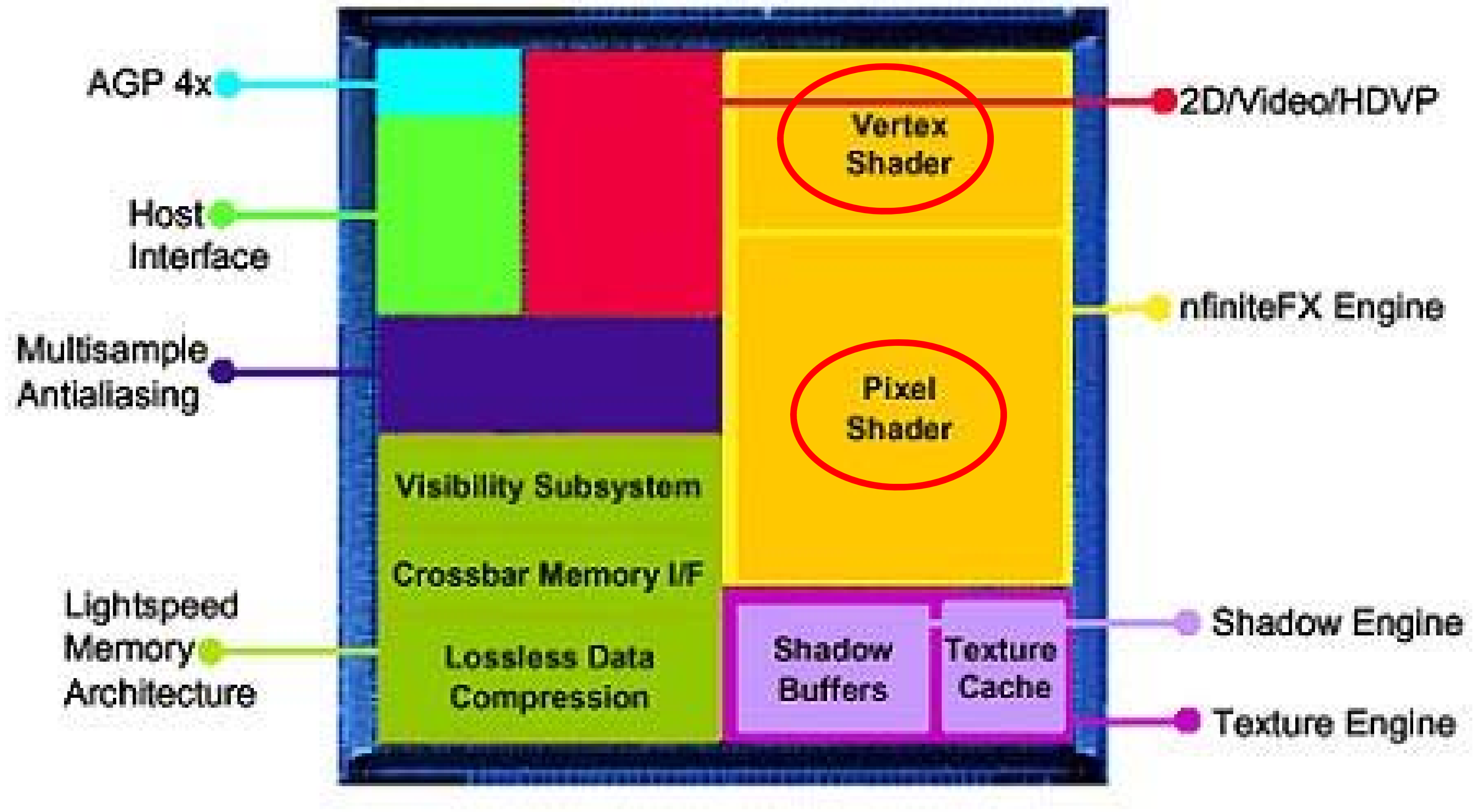
Another reason is that general CPU chips contain on-chip logic to process instructions **out-of-order** if the CPU is blocked and is waiting on something (e.g., a memory fetch). This, too, takes up chip die space.

So, which is better, CPU or GPU?

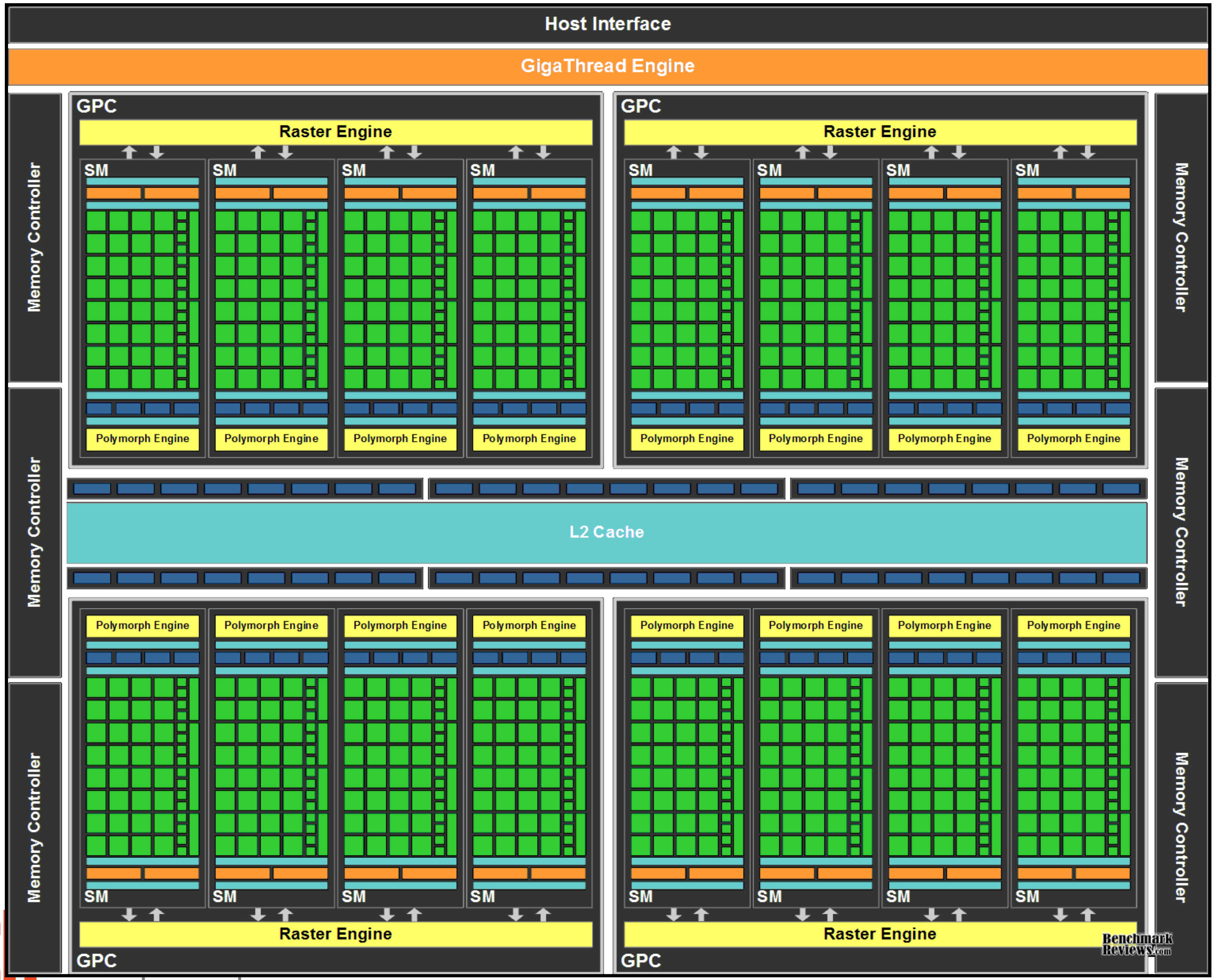
It depends on what you are trying to do!



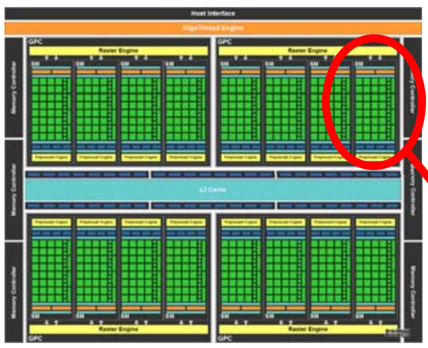
Originally, GPU Devices were very task-specific

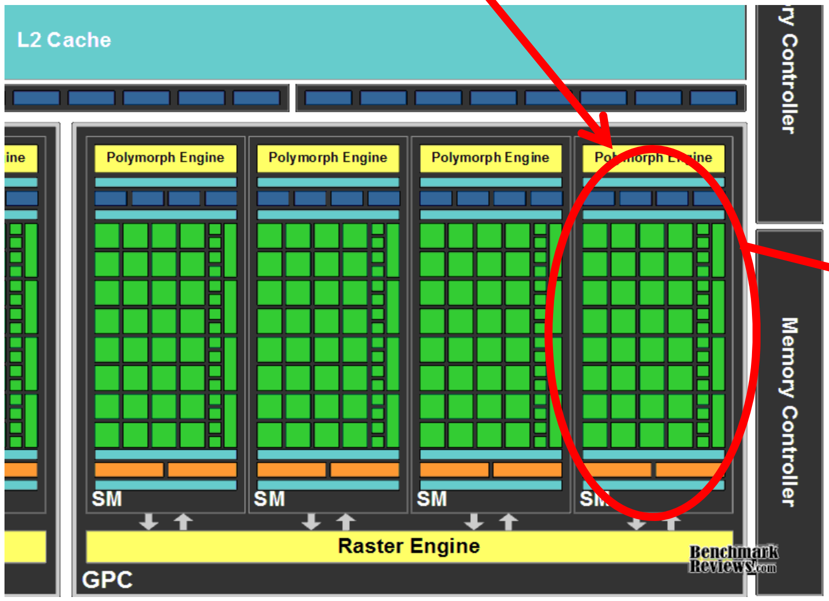


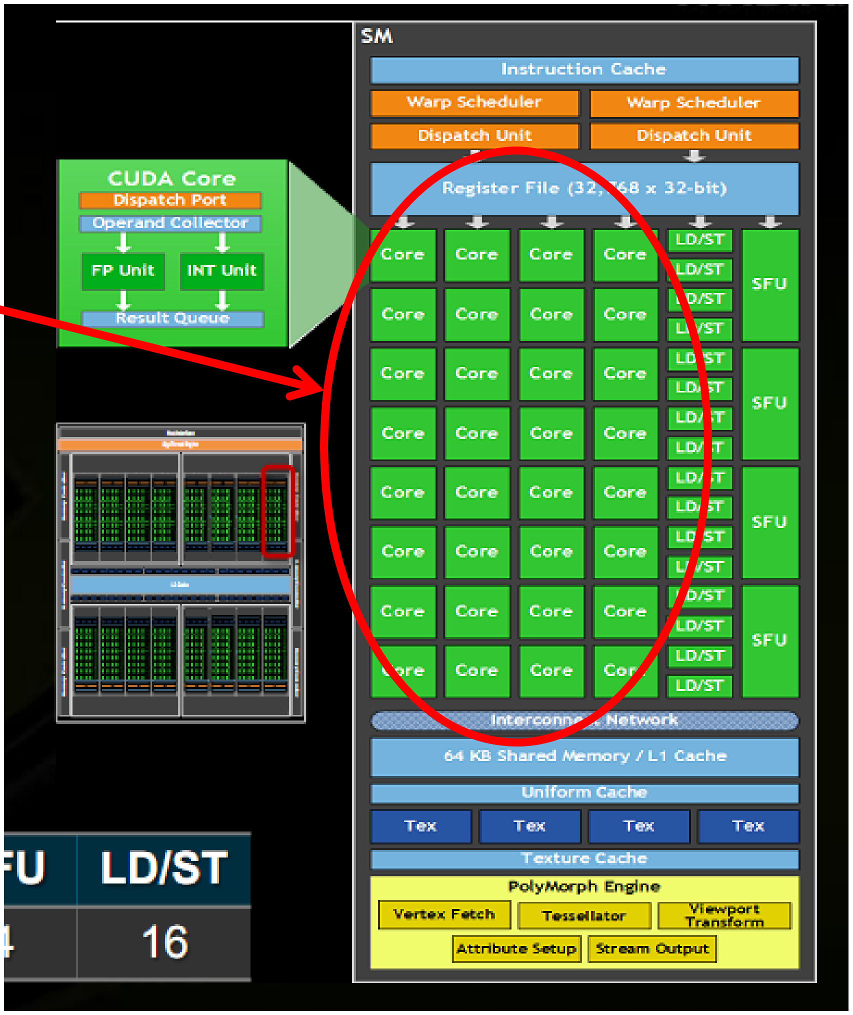
Today's GPU Devices are less task-specific



Consider the architecture of the NVIDIA Titan Black we have in our *rabbit* System





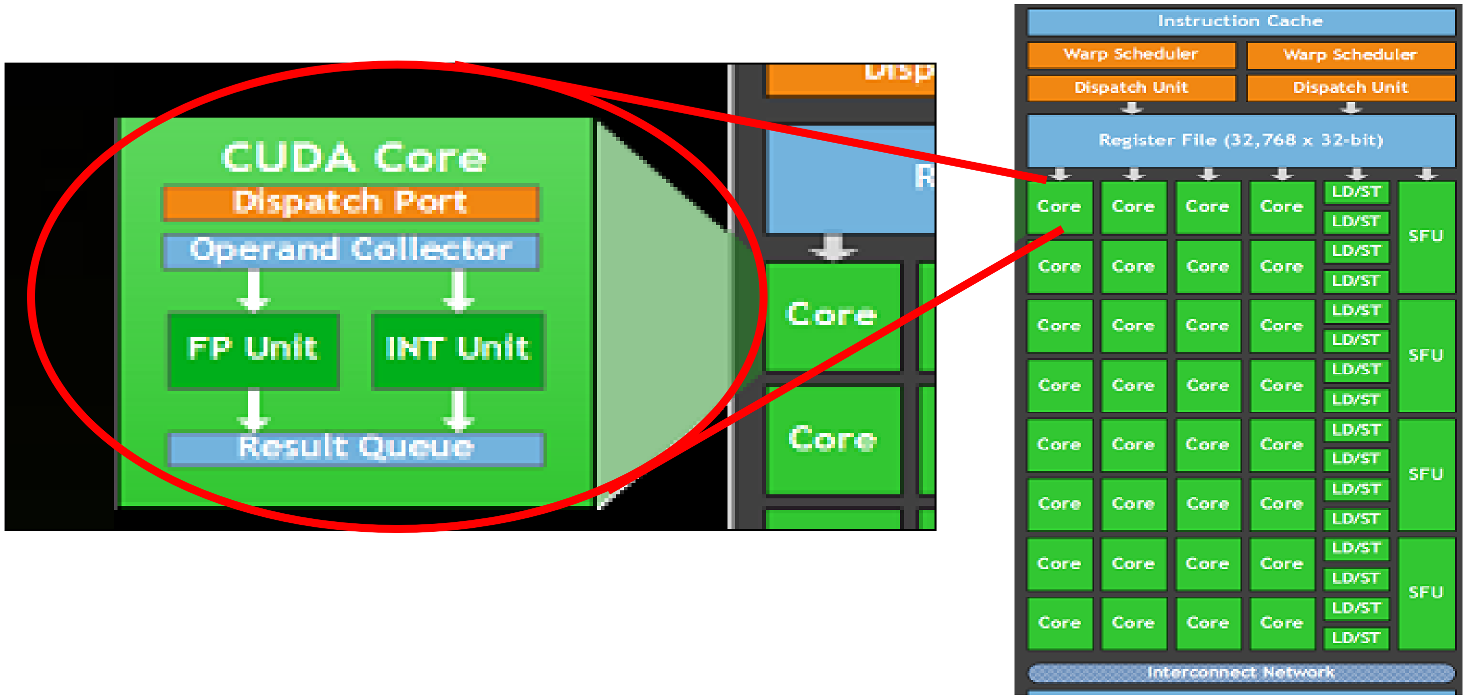


15 Streaming Multiprocessors (SMs) / chip

192 cores / SM

Wow! 2880 cores / chip? Really?

What is a "Core" in the GPU Sense?

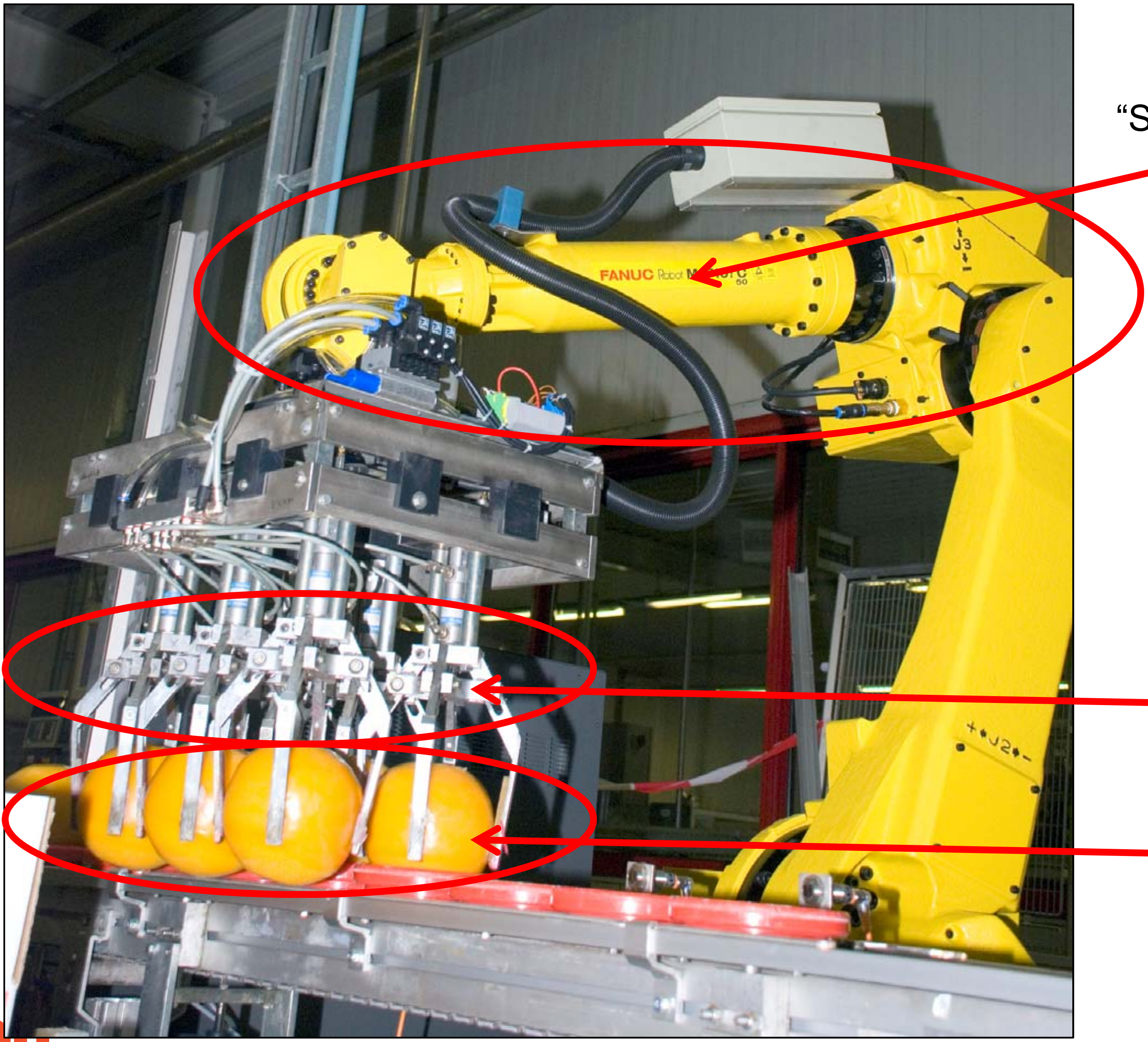


Look closely, and you'll see that NVIDIA really calls these "CUDA Cores"

Look even more closely and you'll see that these CUDA Cores have no control logic – they are **pure compute units**. (The surrounding SM has the control logic.)

Other vendors refer to these as "Lanes". You might also think of them as 192-way SIMD.

A Mechanical Equivalent…



"Streaming Multiprocessor"

"CUDA Cores"

"Data"

http://news.cision.com

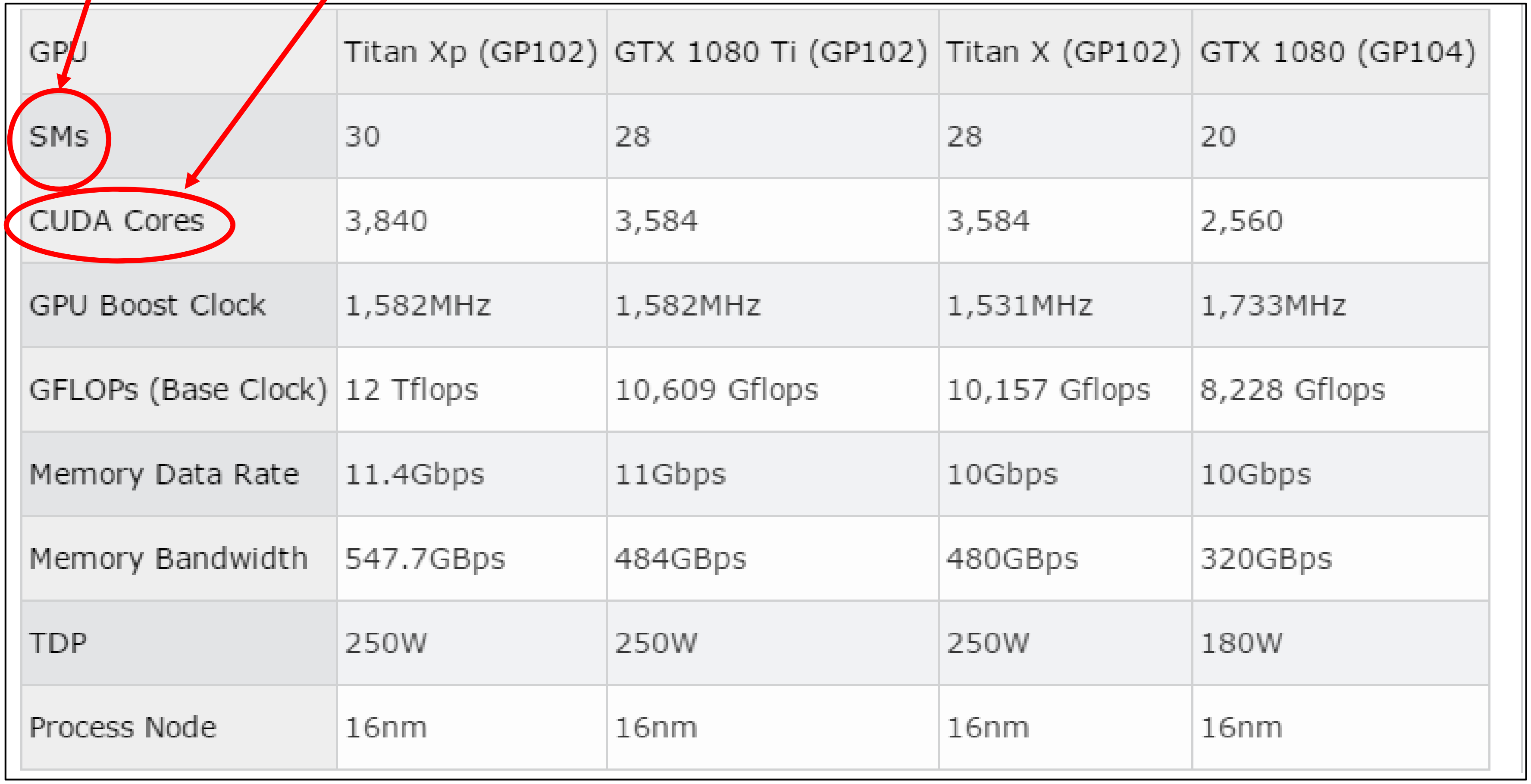
How Many Robots Do You See Here?

12? 72? Depends what you count as a “robot”.

A Spec Sheet Example

Streaming Multiprocessors

128 CUDA Cores per SM



GPU	Titan Xp (GP102)	GTX 1080 Ti (GP102)	Titan X (GP102)	GTX 1080 (GP104)
SMs	30	28	28	20
CUDA Cores	3,840	3,584	3,584	2,560
GPU Boost Clock	1,582MHz	1,582MHz	1,531MHz	1,733MHz
GFLOPs (Base Clock)	12 Tflops	10,609 Gflops	10,157 Gflops	8,228 Gflops
Memory Data Rate	11.4Gbps	11Gbps	10Gbps	10Gbps
Memory Bandwidth	547.7GBps	484GBps	480GBps	320GBps
TDP	250W	250W	250W	180W
Process Node	16nm	16nm	16nm	16nm

Source: Tom's Hardware

The Bottom Line is This

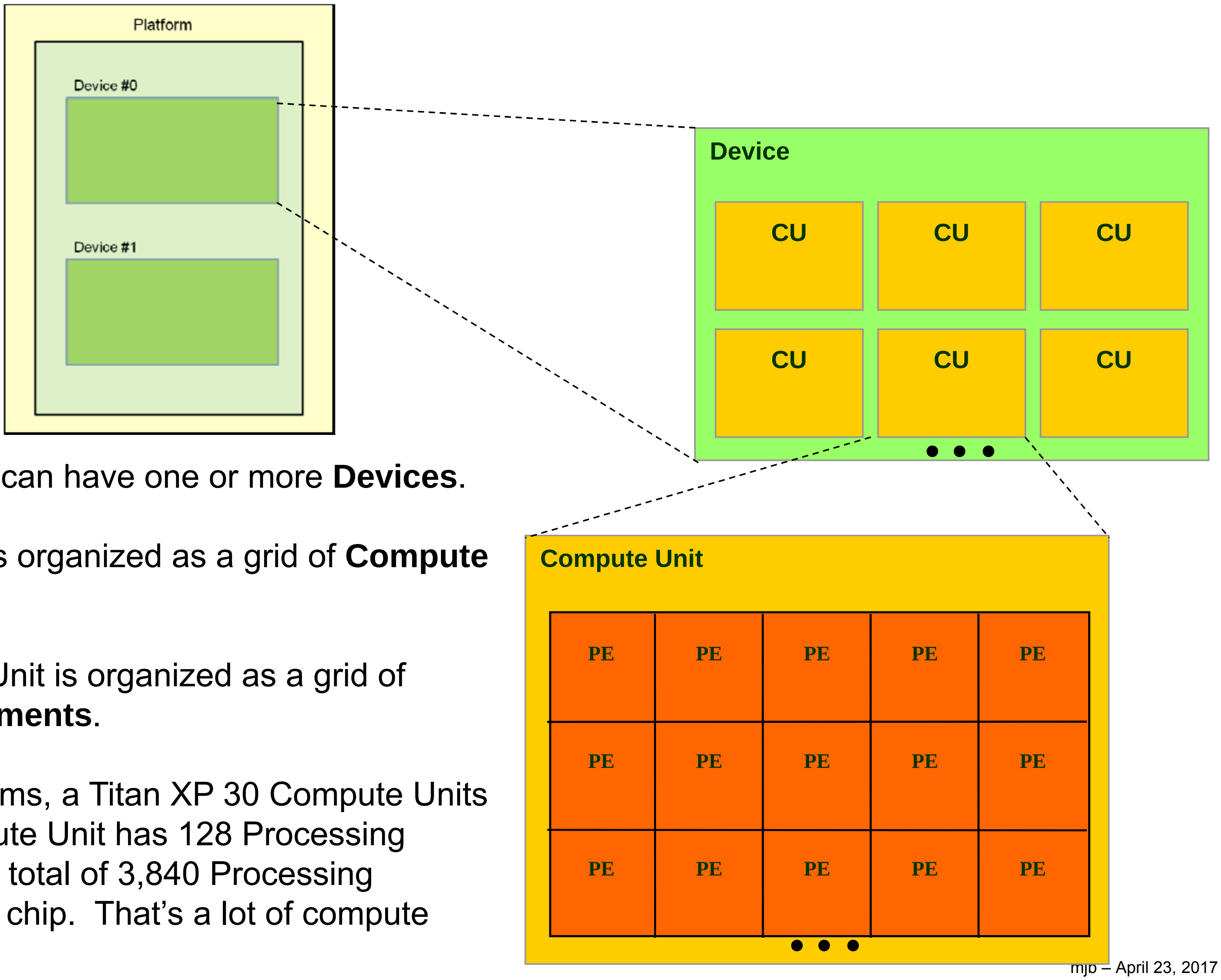
So, the Titan Xp has 30 processors per chip, each of which is optimized to do 128-way SIMD. This is an amazing achievement in computing power. But, it is obvious that it is difficult to *directly* compare a CPU with a GPU. They are optimized to do different things.

So, let's use the information about the architecture as a way to consider what CPUs should be good at and what GPUs should be good at

CPU	**GPU**
General purpose programming	Data parallel programming
Multi-core under user control	Little user control
Irregular data structures	Regular data structures
Irregular flow control	Regular Flow Control

BTW,
The general term in the OpenCL world for an SM is a **Compute Unit**.
The general term in the OpenCL world for a CUDA Core is a **Processing Elemen**t.

Compute Units and Processing Elements are Arranged in Grids



A GPU Platform can have one or more **Devices**.

A GPU **Device** is organized as a grid of **Compute Units.**

Each Compute Unit is organized as a grid of **Processing Elements**.

So in NVIDIA terms, a Titan XP 30 Compute Units and each Compute Unit has 128 Processing Elements – for a total of 3,840 Processing Elements on the chip. That's a lot of compute power!

Thinking ahead to OpenCL…

How can GPUs execute General C Code Efficiently?

• Ask them to do what they do best. Unless you have a very intense **Data Parallel** application, don't even think about using GPUs for computing.

• GPU programs expect you to not just have a few threads, but to have ***thousands*** of them!

• Each thread executes the same program (called the *kernel*), but operates on a different small piece of the overall data

• Thus, you have many, many threads, all waking up at about the same time, all executing the same kernel program, all hoping to work on a small piece of the overall problem.

• OpenCL has built-in functions so that each thread can figure out which thread number it is, and thus can figure out what part of the overall job it's supposed to do.

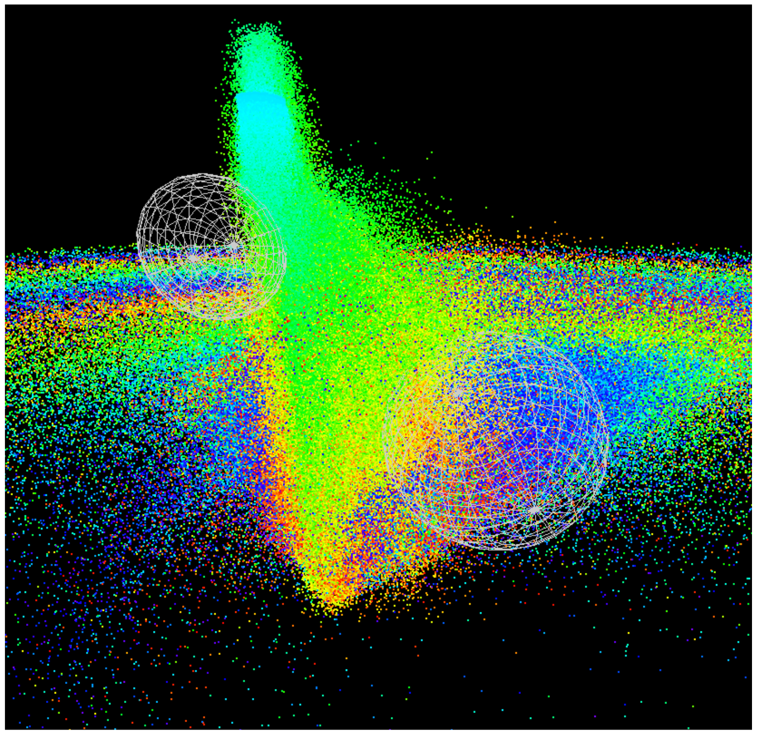
• When a thread gets blocked somehow (a memory access, waiting for information from another thread, etc.), the processor switches to executing another thread to work on.

So, the Trick is to Break your Problem into Many, Many Small Pieces

Particle Systems are a great example.

1. Have one thread per *each particle*.
2. Put all of the initial parameters into an array in GPU memory.
3. Tell each thread what the current **Time** is.
4. Each thread then computes its particle's position, color, etc. and writes it into arrays in GPU memory.
5. The CPU program then initiates OpenGL drawing of the information in those arrays.

Note: once setup, the data never leaves GPU memory!


Ben Weiss