CS 758: Advanced Topics in Computer Architecture

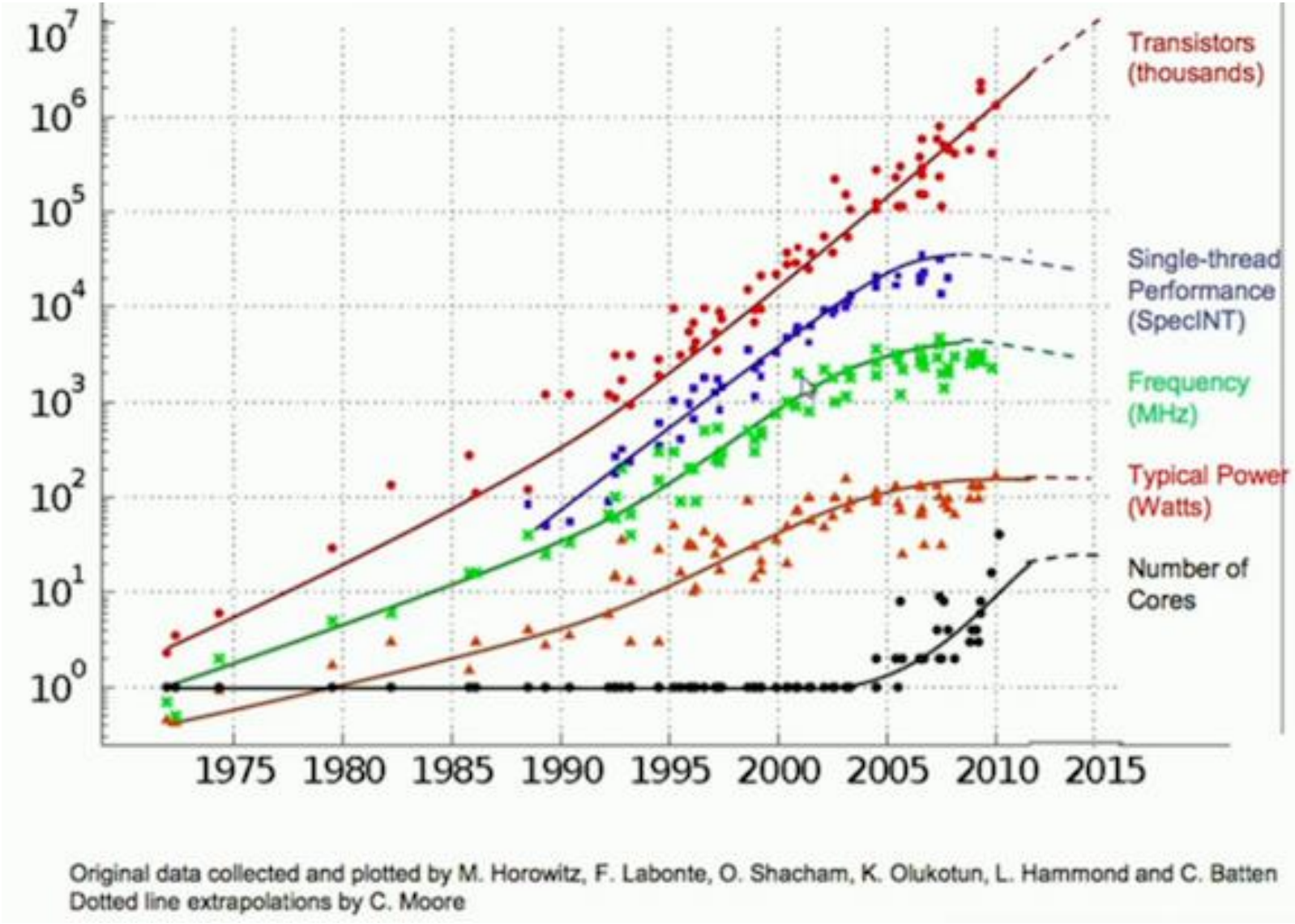
Lecture #2: Dark Silicon & Intro to GPUs

Professor Matthew D. Sinclair

Some of these slides were developed by Tim Rogers at the Purdue University, Tor Aamodt at the University of British Columbia, and Wen-mei Hwu & David Kirk at the University of Illinois at Urbana-Champaign.

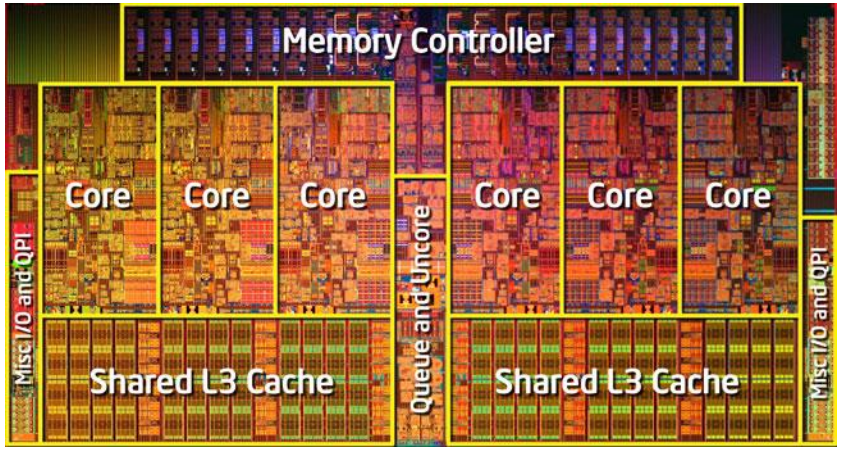
Slides enhanced by Matt Sinclair

Moore’s Law

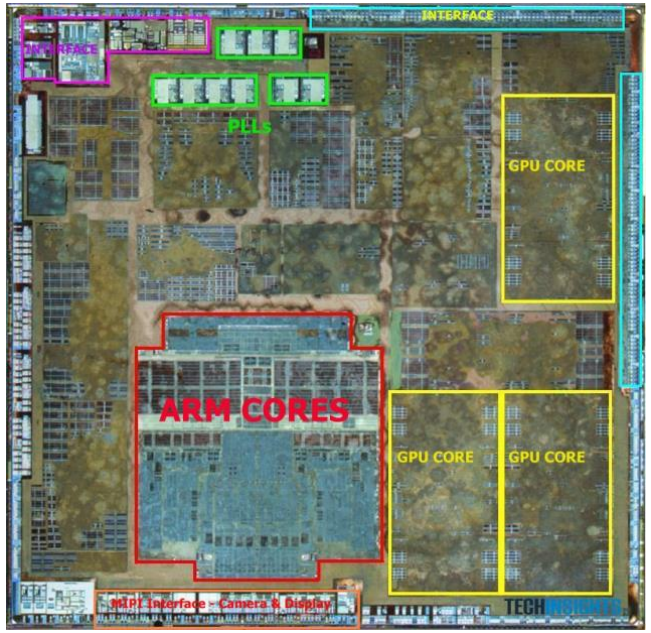

10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1975 1980 1985 1990 1995 2000 2005 2010 2015
Transistors (thousands)
Single-thread Performance (SpecINT)
Frequency (MHz)
Typical Power (Watts)
Number of Cores
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Silver Bullet for Moore's Law?

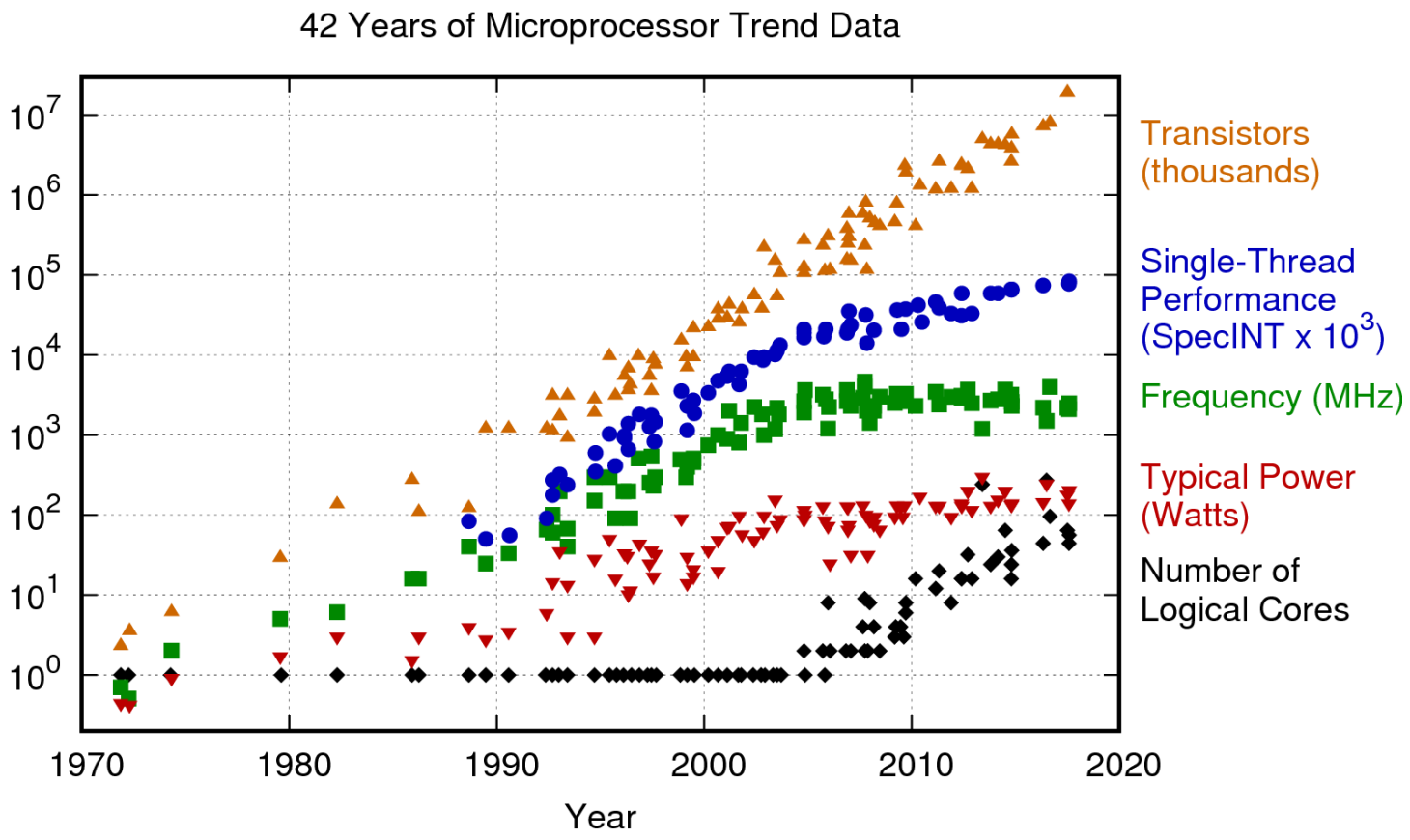
Parallelism



Specialization



Updated Moore's Law Plot



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Dark Silicon Key Takeaways

- Can’t rely on multi-core parallelism to save us
  - Many “real” applications don’t have enough parallelism
  - Even if they do, power limits them

Can’t reach Moore’s Law scaling

- GPUs help …
  - … but only to a point
  - Eventually power or parallelism limits them

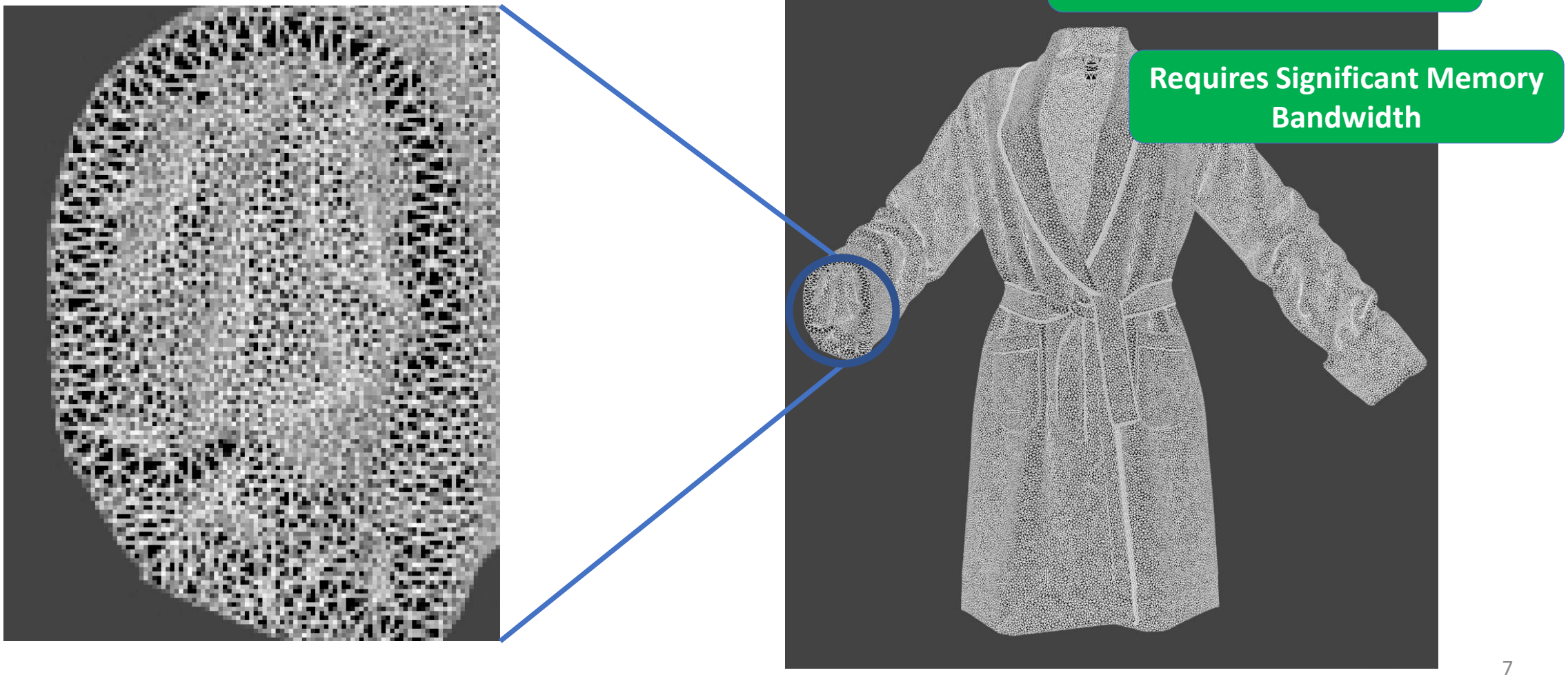
Tremendous opportunity for innovation

What **was** a GPU?

- GPU = Graphics Processing Unit
  - Accelerator for raster-based graphics (OpenGL, DirectX, Vulkan)
  - Highly programmable
  - Commodity hardware
  - 100’s of ALUs; 10000’s of concurrent threads

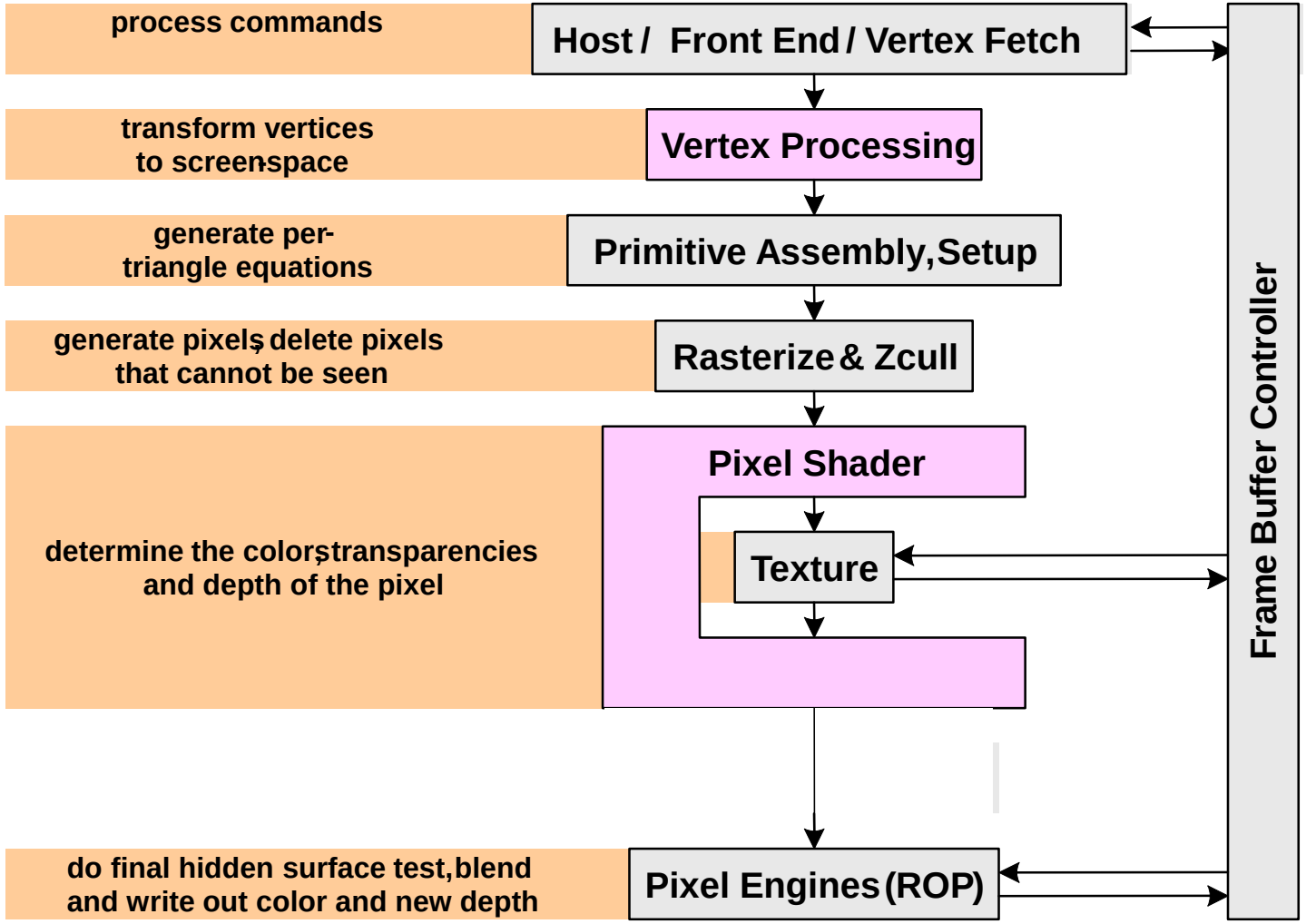
Today the name GPU is not really meaningful.
In reality they are highly parallel, highly programmable vector supercomputers.

Modern GPUs: Good at drawing triangles



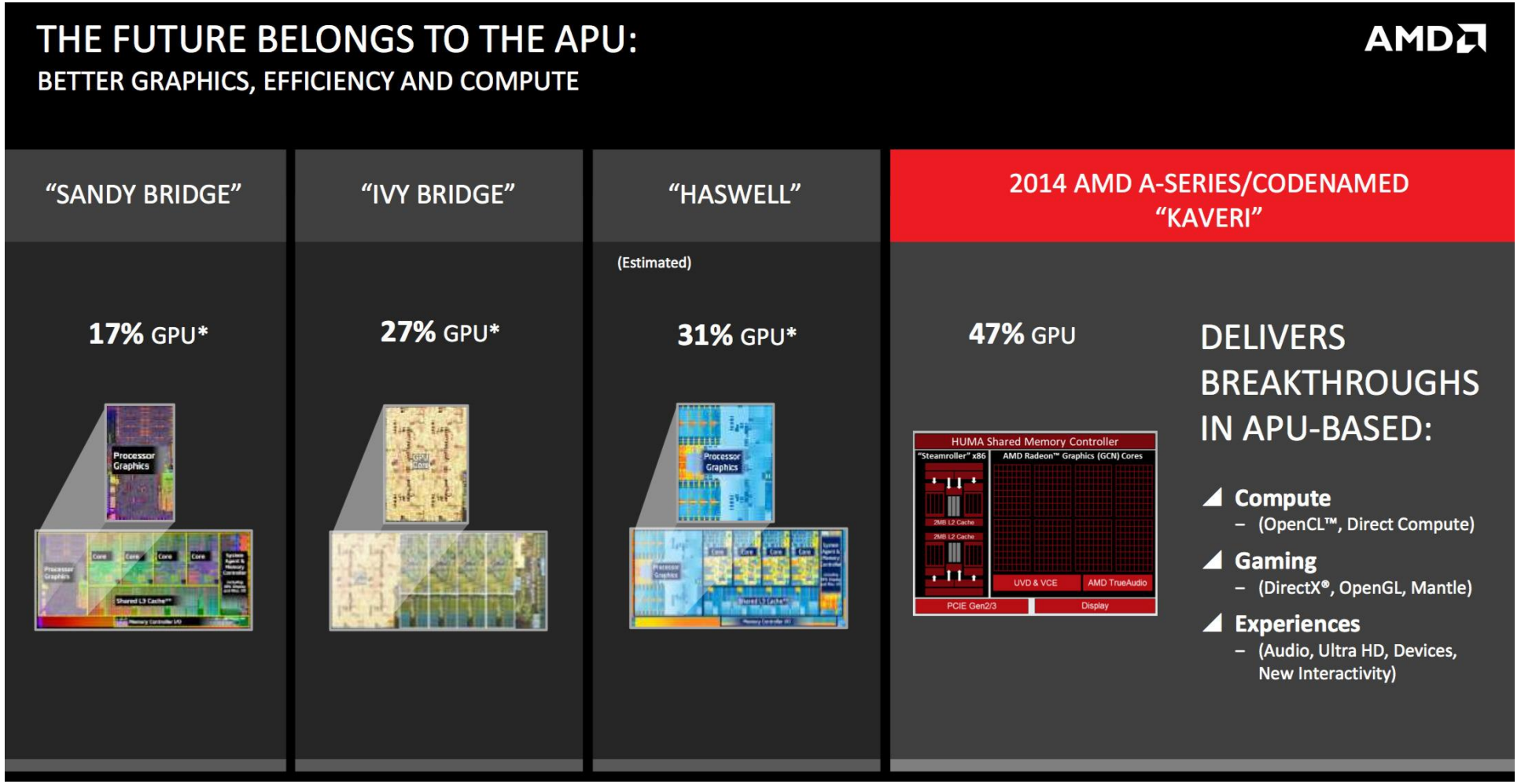
pixel color result of running “shader” program

GPU: The Life of a Triangle



[David Kirk / Wen-mei Hwu]

Today: GPUs are Ubiquitous



[APU13 keynote]

Flynn's Taxonomy

- Focus: Data parallel workloads
  - Independent, identical computation on multiple data inputs
- MIMD (Multiple Instruction, Multiple Data):
  - Split independent work over multiple processors
  - Subcategory: SPMD (Single Program, Multiple Data)
    - Only if work is identical (same program)
- SIMD (Single Instruction, Multiple Data):
  - Split identical, independent work over multiple execution units
  - More efficient: eliminate redundant fetch/decode vs. SPMD/MIMD
  - Use single PC and single register file

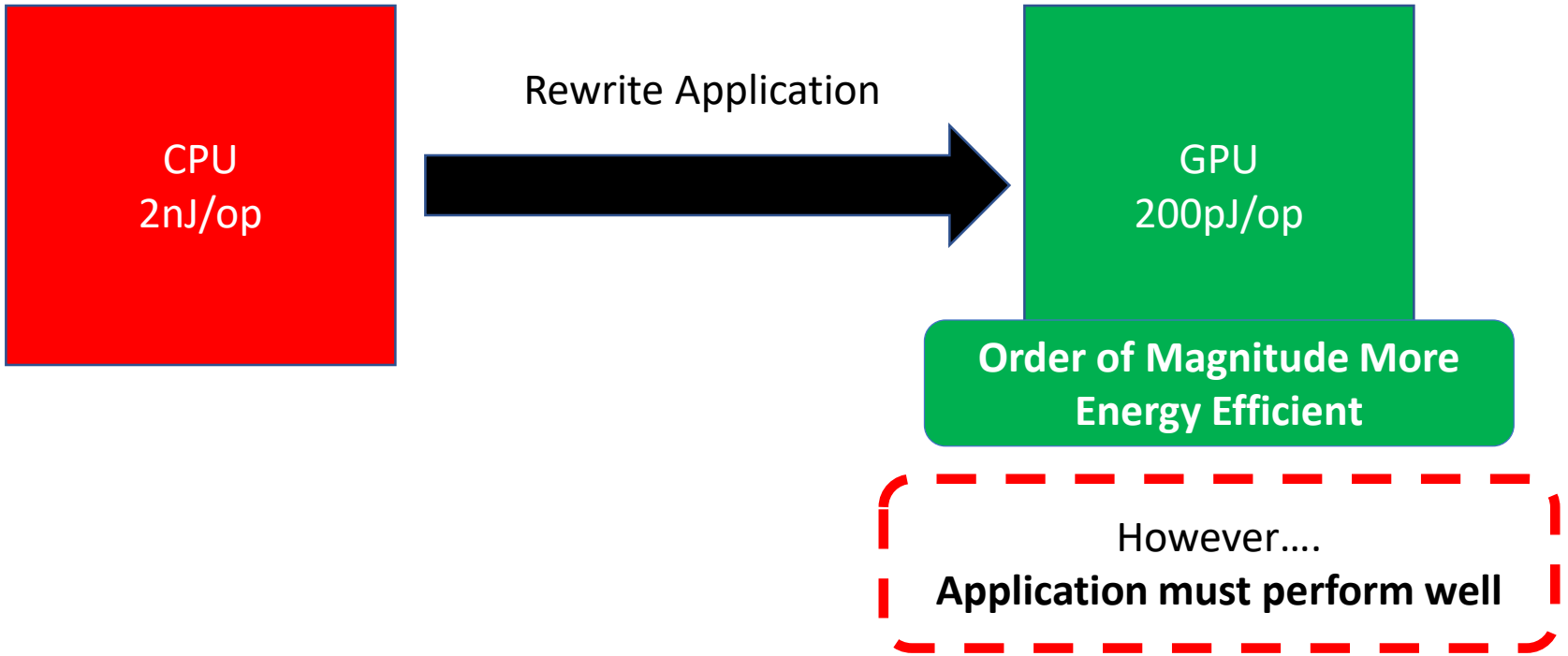
Flynn's Taxonomy (Cont.)

- SIMD's cousin: SIMT (Single Instruction, Multiple Thread)
  - Split identical, independent work over multiple lockstep threads
  - One PC for group of lockstep threads, but multiple register files
  - This is what GPUs do today
  - Work well for **streaming** applications

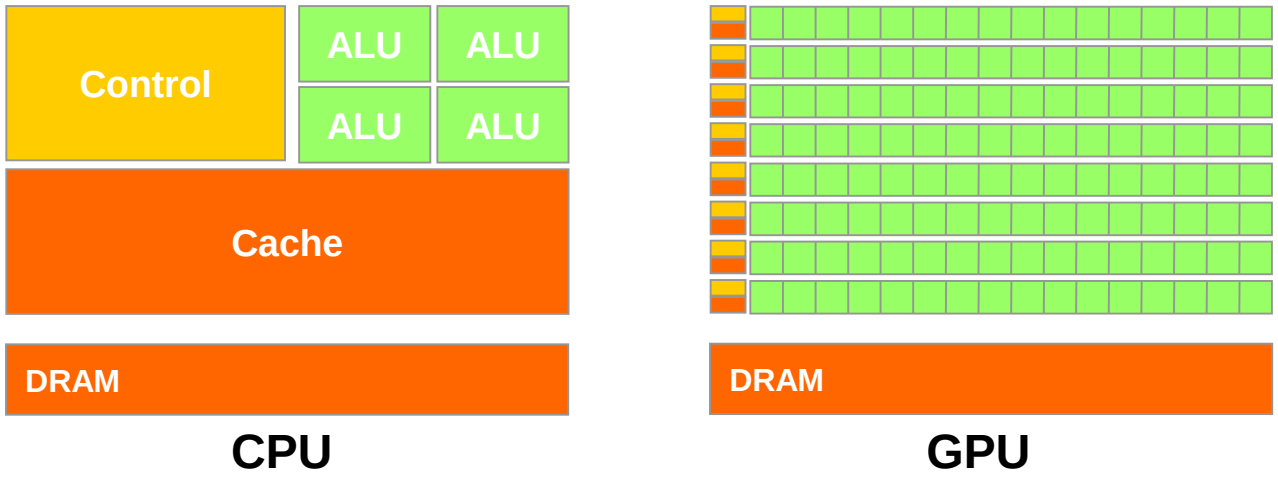
- Sidenote:
  - People use SIMT and SIMD somewhat interchangeably
  - They do have differences though

Why use a GPU for computing?

- GPU uses larger fraction of silicon for computation than CPU.
- At peak performance GPU uses order of magnitude less energy per operation than CPU.



GPU uses larger fraction of silicon for computation than CPU?



[NVIDIA]

Growing Interest in GPGPU

- Supercomputing – Green500.org Nov 2014

  “the top three slots of the Green500 were powered by three different accelerators with number one, L-CSC, being powered by AMD FirePro™ S9150 GPUs, powered by NVIDIA K20x GPUs. Beyond these top three, the next 20 supercomputers were also accelerator-based.”

- Deep Belief Networks map *very* well to GPUs (e.g., Google keynote at 2015 GPU Tech Conf.)

  http://blogs.nvidia.com/blog/2015/03/18/google-gpu/

  http://www.ustream.tv/recorded/60071572

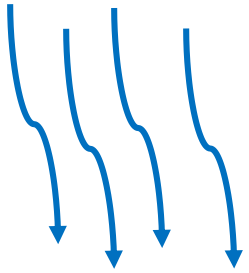
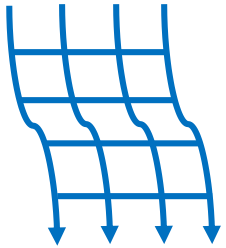
“Machine learning is the manna sent to GPUs from heaven”
- Industry Researcher

GPGPUs vs. Vector Processors

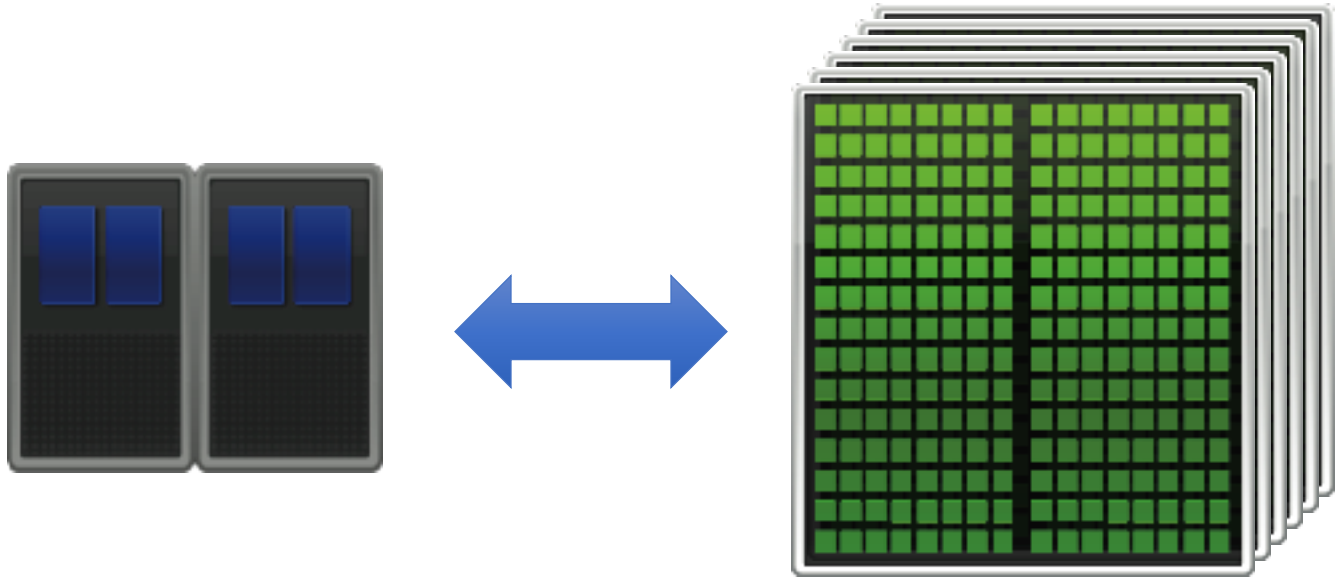
- Similarities at hardware level between GPU and vector processors.

- (Arguably) SIMT programming model moves hardest parallelism detection problem from compiler to programmer.

Execution Model Comparison

	MIMD/SPMD	SIMD/Vector	SIMT
			
Example	Multicore CPUs	x86 SSE/AVX	GPUs
Pros	More general: better support for TLP	Able to mix serial and parallel code	Easier to program, Scatter & Gather operations
Cons	Inefficient for data parallelism	Gather/Scatter implementations more complicated	Divergence kills performance

GPU Compute Programming Model

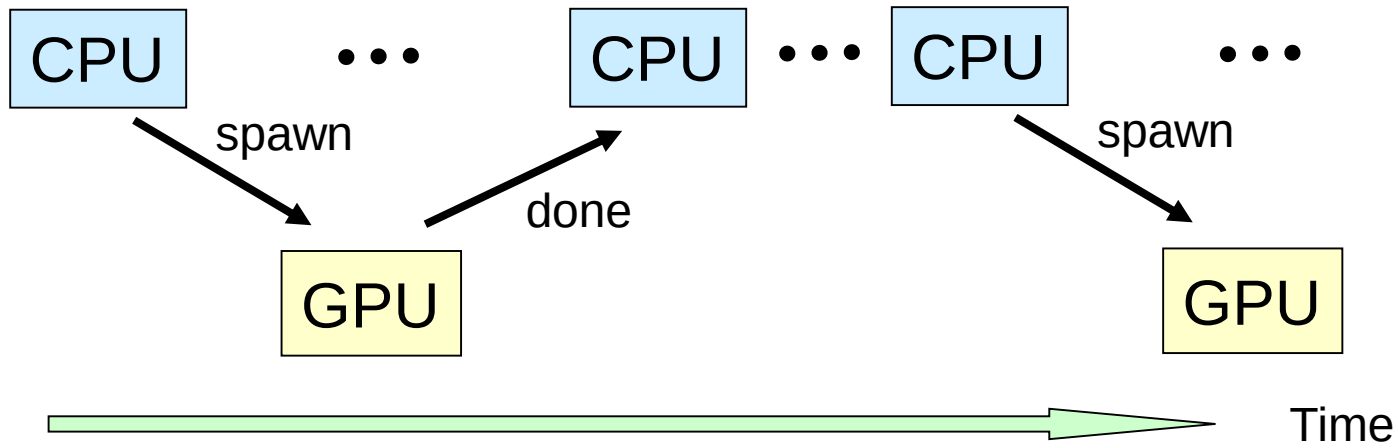


CPU GPU

How is this system programmed (today)?

GPGPU Programming Model

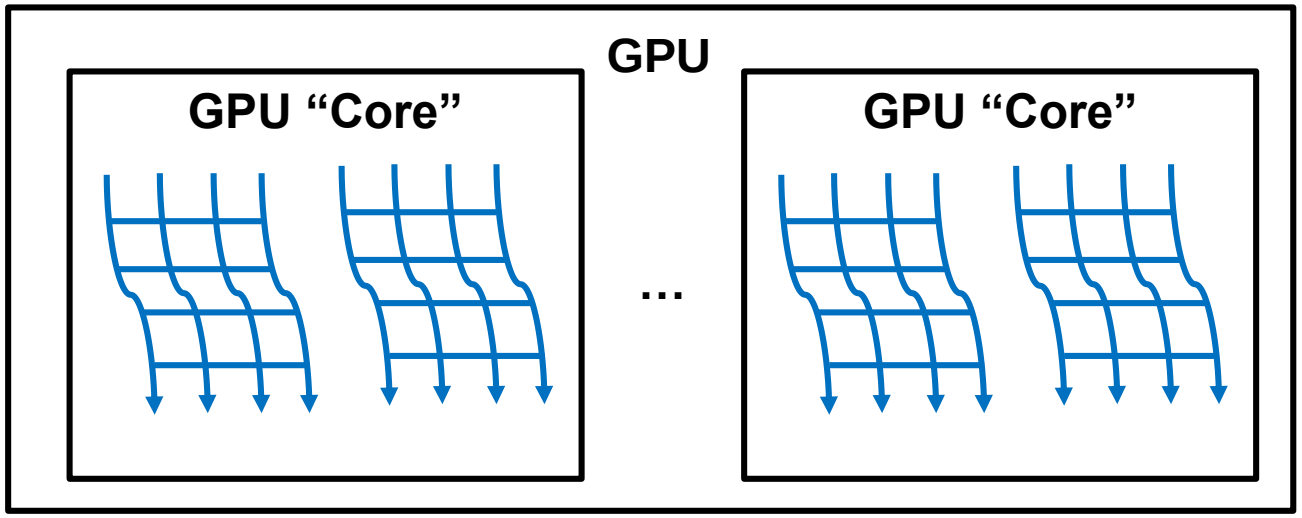
- CPU “Off-load” parallel kernels to GPU



  - Transfer data to GPU memory
  - GPU HW spawns threads
  - Need to transfer result data back to CPU main memory

SIMT Execution Model

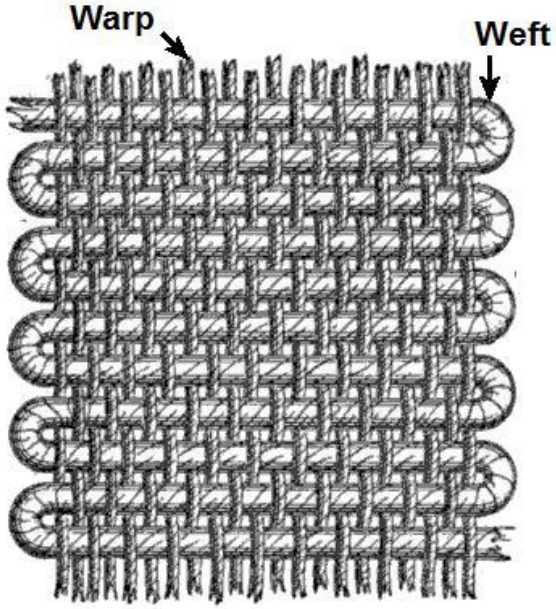
- Group SIMT “threads” together on a GPU “core”
- SIMT threads are grouped together for efficiency
  - Loose analogy: SIMT thread group ≈ one CPU SMT thread
  - Difference: GPU threads are **exposed** to the programmer
- Execute different SIMT thread groups simultaneously
  - On a single GPU “core” per-cycle SIMT thread groups swaps
  - Execute different SIMT thread groups on different GPU “cores”



SIMT Execution Model (Cont.)

- Programmers sees MIMD threads (scalar)
- GPU bundles threads into warps (wavefronts) and runs them in lockstep on SIMD hardware
- An NVIDIA warp groups 32 consecutive threads together (AMD wavefronts group 64 threads together)

- Aside: Why "Warp"? In the textile industry, the term "warp" refers to "the threads stretched lengthwise in a loom to be crossed by the weft" [Oxford Dictionary].



[https://en.wikipedia.org/wiki/Warp_and_woof]

For Next Class

- Read GPGPA Chapters 1-2

Final Thought

- Who has heard of Spectre and Meltdown?

- Are GPUs affected by these?