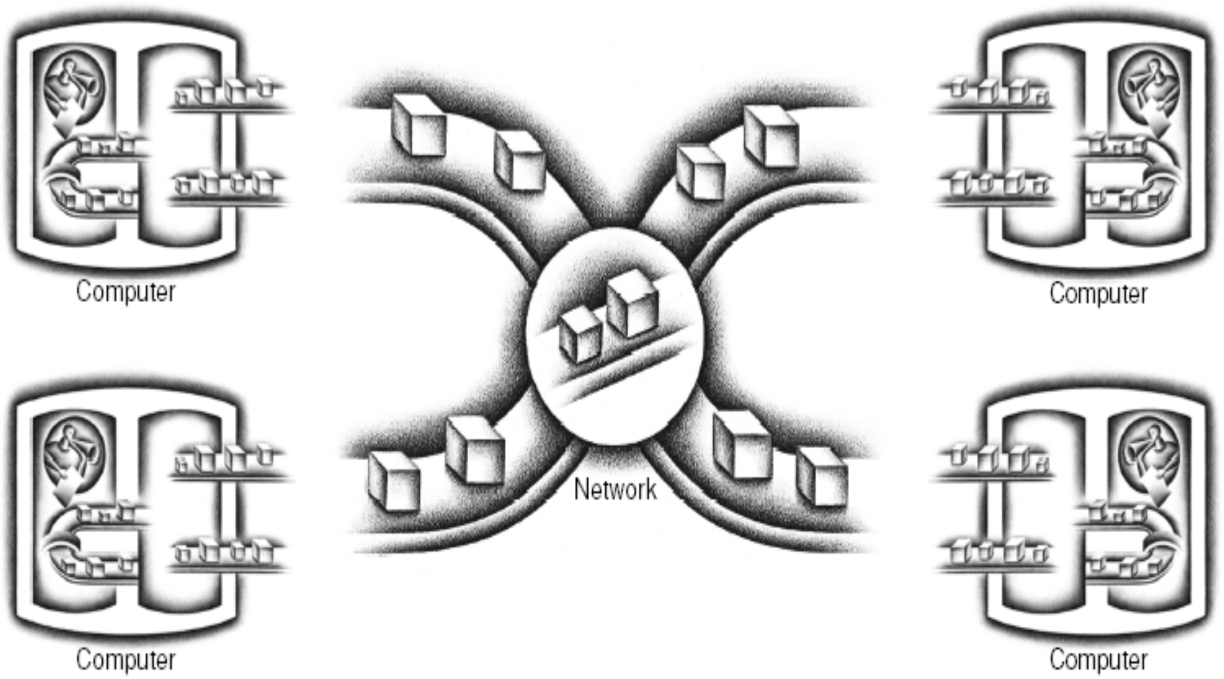
Lecture 14 Multicores, Multiprocessor and Cluster (Part II)



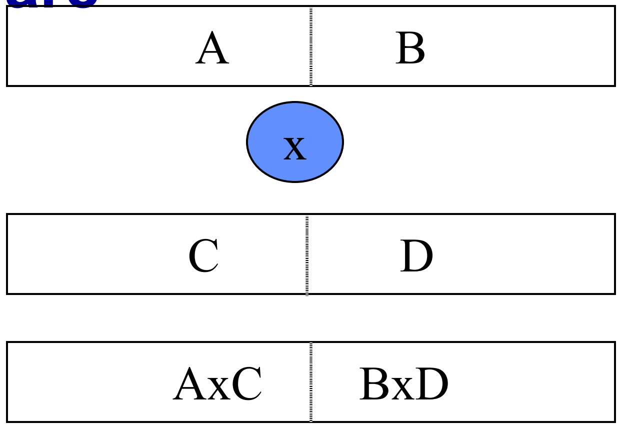
Computing Device Classification: Instruction and Data Streams

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE instructions of x86
	Multiple	**MISD**: No examples today	**MIMD**: Intel Xeon e5345

- SPMD: Single Program Multiple Data
  - A parallel program on a MIMD computer
  - Conditional code for different processors

SIMD

- Operate elementwise on vectors of data
  - E.g., MMX and SSE instructions in x86
    - » Multiple data elements in 128-bit wide registers
- All processors execute the same instruction at the same time
  - Each with different data address, etc.
- Simplifies synchronization
- Reduced instruction control hardware
- Works best for highly data-parallel applications



Vector Processors

- Highly pipelined function units
- Stream data from/to vector registers to units
  - Data collected from memory into registers
  - Results stored from registers to memory
- Example: Vector extension to MIPS
  - 32 × 64-element registers (64-bit elements)
  - Vector instructions
    - `lv`, `sv`: load/store vector
    - `addv.d`: add vectors of double
    - `addvs.d`: add scalar to each element of vector of double
- Significantly reduces instruction-fetch bandwidth

Example: DAXPY (Y = a × X + Y)

double

- **Conventional MIPS code**

```
      l.d   $f0,a($sp)      ;load scalar a
      addiu r4,$s0,#512     ;upper bound of what to
load
loop: l.d   $f2,0($s0)      ;load x(i)
      mul.d $f2,$f2,$f0     ;a × x(i)
      l.d   $f4,0($s1)      ;load y(i)
      add.d $f4,$f4,$f2     ;a × x(i) + y(i)
      s.d   $f4,0($s1)      ;store into y(i)
      addiu $s0,$s0,#8      ;increment index to x
      addiu $s1,$s1,#8      ;increment index to y
      subu  $t0,r4,$s0      ;compute bound
      bne   $t0,$zero,loop  ;check if done
```

- **Vector MIPS code**

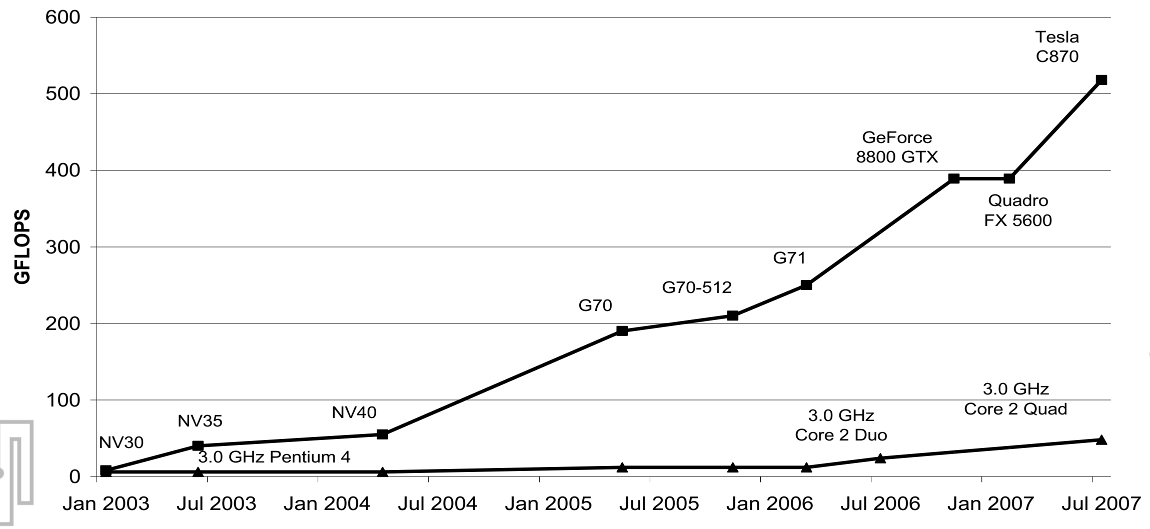
```
      l.d     $f0,a($sp)    ;load scalar a
      lv      $v1,0($s0)    ;load vector x
      mulvs.d $v2,$v1,$f0   ;vector-scalar multiply   aX
      lv      $v3,0($s1)    ;load vector y
      addv.d  $v4,$v2,$v3   ;add y to product
      sv      $v4,0($s1)    ;store the result
```

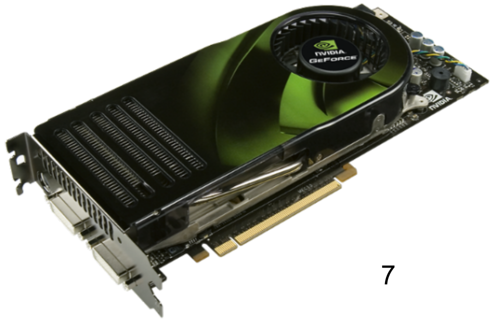
Vector vs. Scalar

- Vector architectures and compilers
  - Simplify data-parallel programming
  - Explicit statement of absence of loop-carried dependences
    - Reduced checking in hardware
  - Regular access patterns benefit from interleaved and burst memory
  - Avoid control hazards by avoiding loops
- More general than ad-hoc media extensions (such as MMX, SSE)
  - Better match with compiler technology

What is GPU?

- Graphics Processing Units
- GPU is a device to compute massive vertices, pixels, and general purpose data
- Feature
  - High availability
  - High computing performance
  - Low price of computing capability





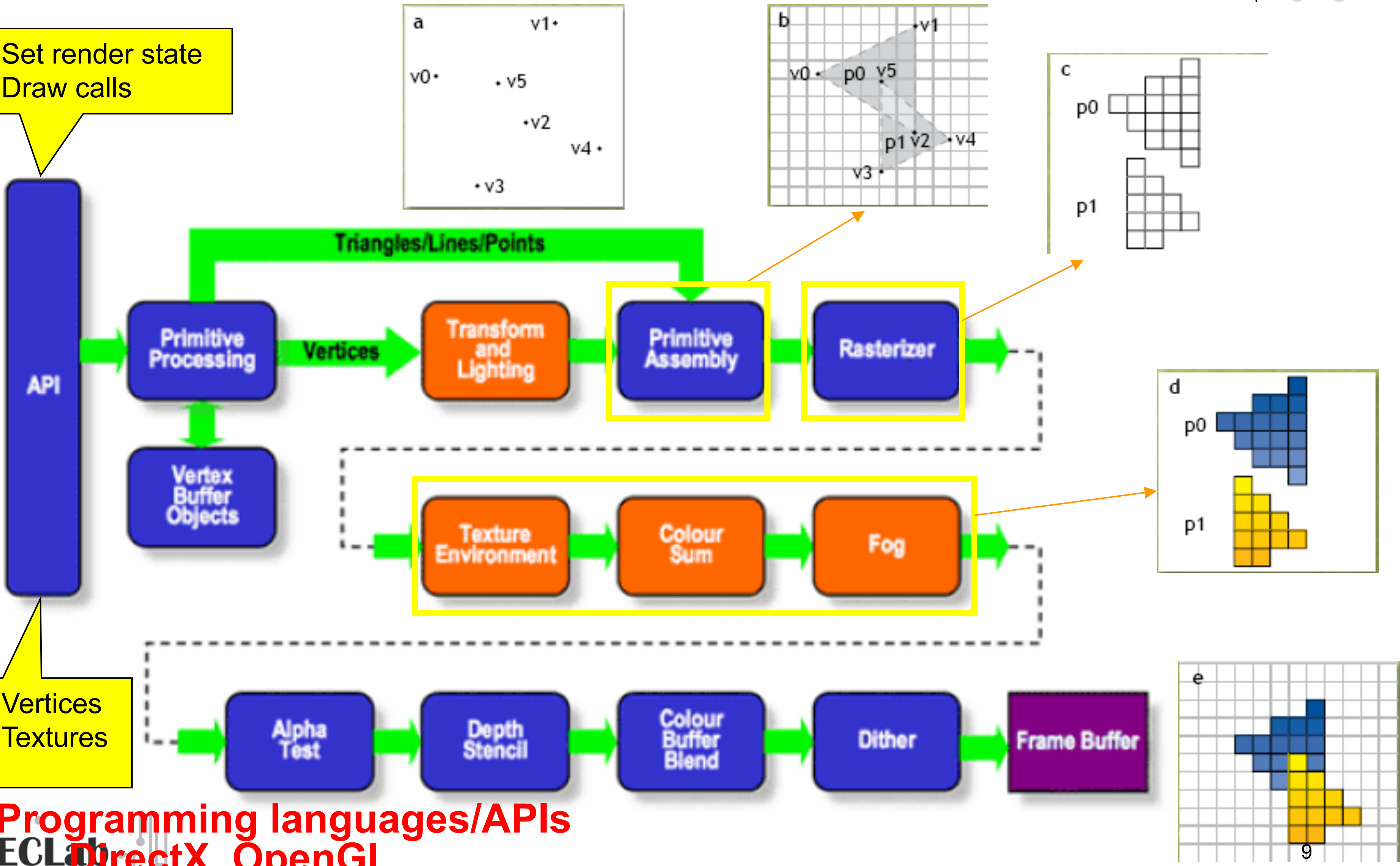
GPU's History and Evolution

- Early History
  - In early 90's, graphics are only performed by a video graphics array (VGA) controller
  - In 1997, VGA controllers start to incorporate 3D acceleration functions
  - In 2000, the term GPU is coined to denote that the graphics devices had become a processor
- GPU Evolution
  - Fixed-Function → Programmable
    - 1999, NVIDIA *Geforce 256*, Fixed-function vertex transform and pixel pipeline
    - 2001, NVIDIA *Geforce 3*, 1st programmable vertex processor
    - 2002, ATI *Radeon 9700*, 1st programmable pixel(fragment) processor
  - Non-unified Processor → Unified Processor
    - 2005, Microsoft *XBOX 360*, 1st unified shader architecture
  - Tesla GPU series released in 2007
  - Fermi Architecture released in 2009
  - Kepler Architecture released in 2012
  - Maxwell Architecture released in 2014
  - Pascal Architecture released in 2016 (16 nm FinFET process)
  - Volta Architecture released in 2017 – Tensor Core for AI (12 nm proess)

energy

Fixed-function 3D Graphics Pipeline

- Set render state
- Draw calls



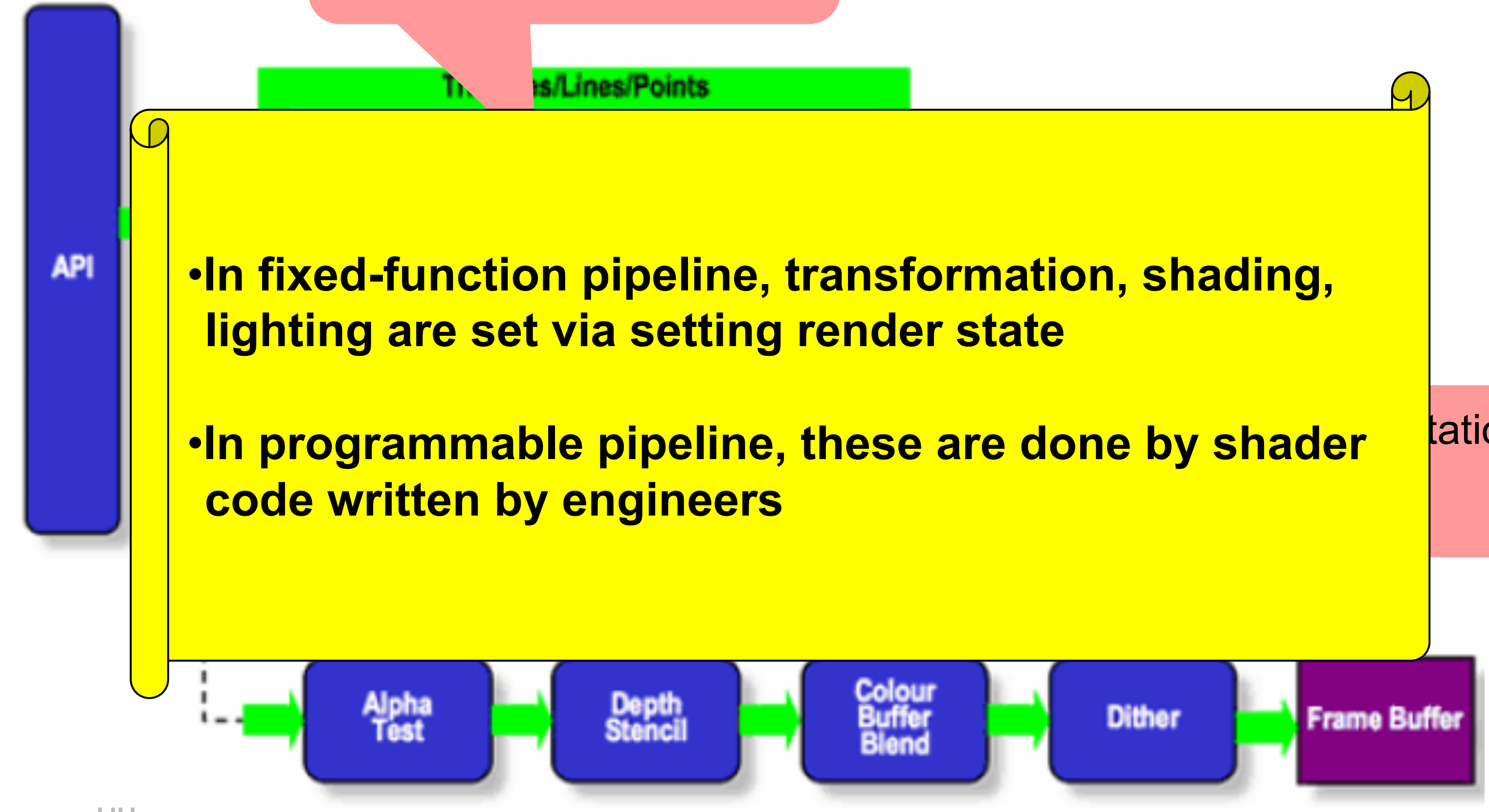
- Vertices
- Textures

Programming languages/APIs
DirectX, OpenGL

Programmable 3D Graphics Pipeline

High Level Shader Language (HLSL)

- Vertex computation



tation

•In fixed-function pipeline, transformation, shading, lighting are set via setting render state

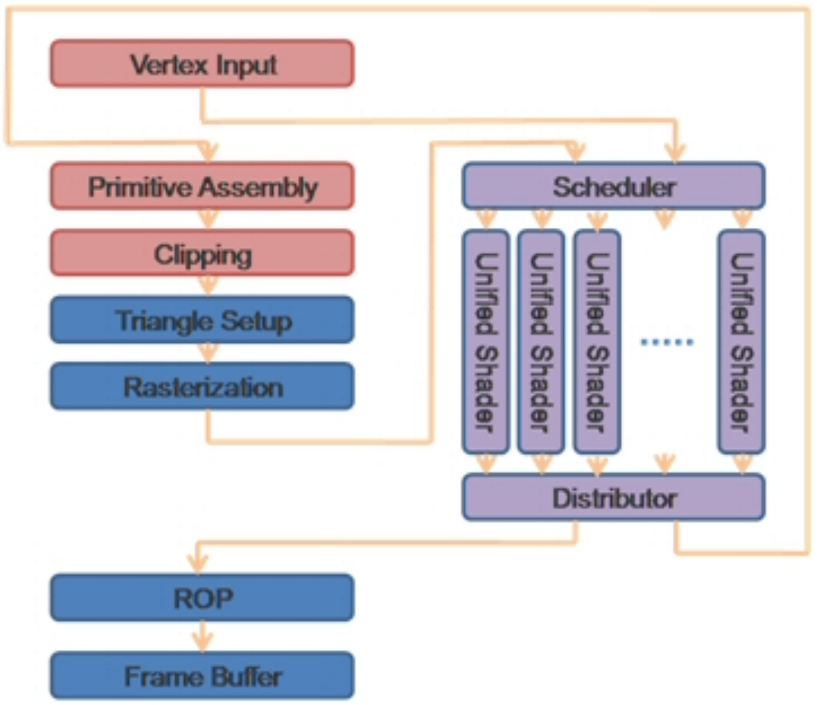
•In programmable pipeline, these are done by shader code written by engineers

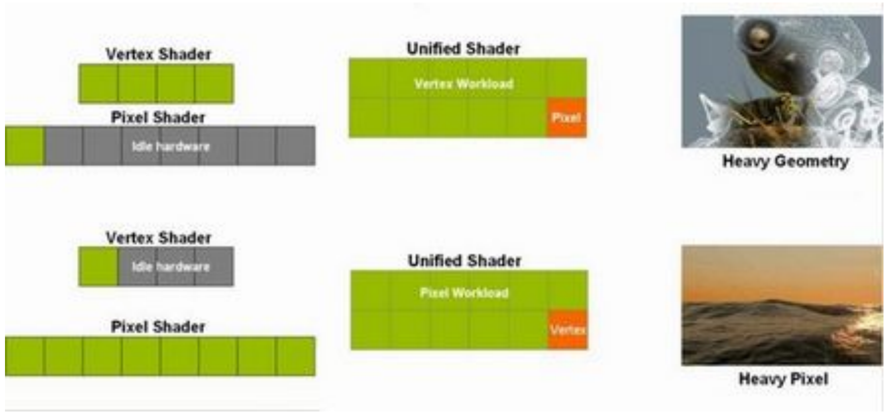
ECLab

Embedded Computing Lab.

Unified Shader Architecture

- Use the same shader processors for all types of computation
  - Vertex threads
  - Pixel threads
  - Computation threads
- Advantage
  - Better resource utilization
  - Lower hardware complexity





Embedded Computing Lab.

Modern GPUs: A Computing Device

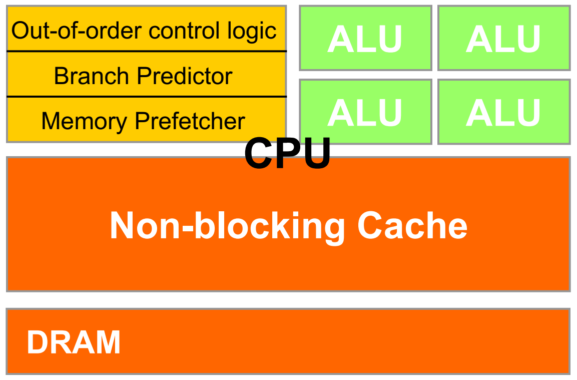
- GPUs have orders of magnitudes more computing power than CPU
- General-purpose tasks with high-degree of data level parallelism running on GPU outperform those on CPU => General-Purpose computing on GPU (GPGPU)
- GPGPU programming models
  - NVIDIA's CUDA
  - AMD's StreamSDK
  - OpenCL

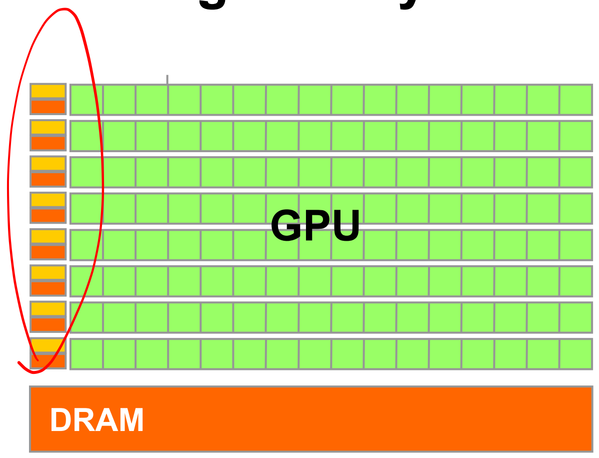
GPGPU Performance	
Medical Imaging	300X
Molecular Dynamics	150X
SPICE	130X
Fourier Transform	130X
Fluid Dynamics	100X

ECLab
Embedded Computing Lab.

Fundamental Architectural Differences between CPU & GPU

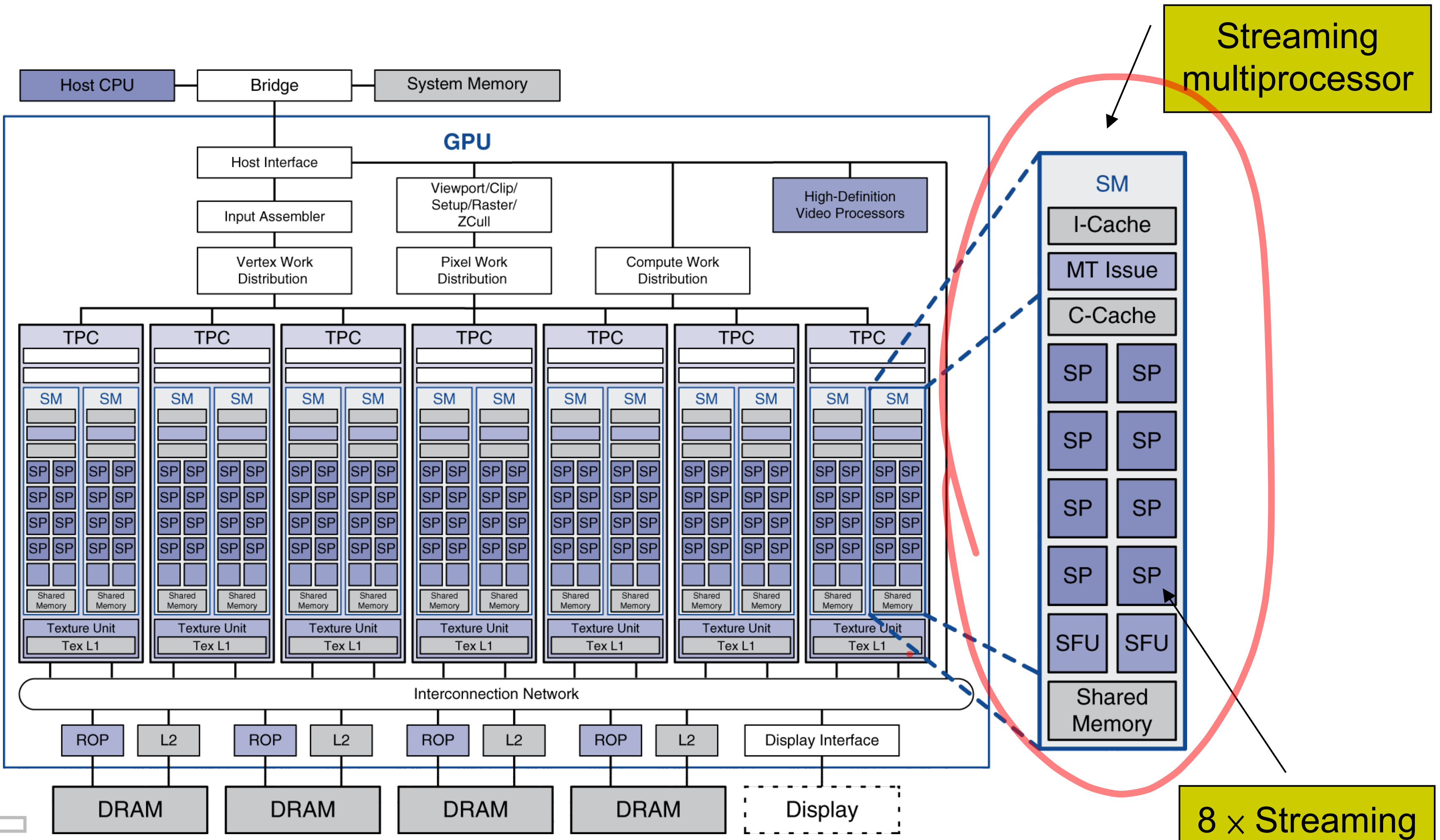
- Multi-core CPU
  - Coarse-grain, heavyweight threads
  - Memory latency is resolved though large on-chip caches & out-of-order execution
- Modern GPU
  - Fine-grain, lightweight threads
  - Exploit thread-level parallelism for hiding latency





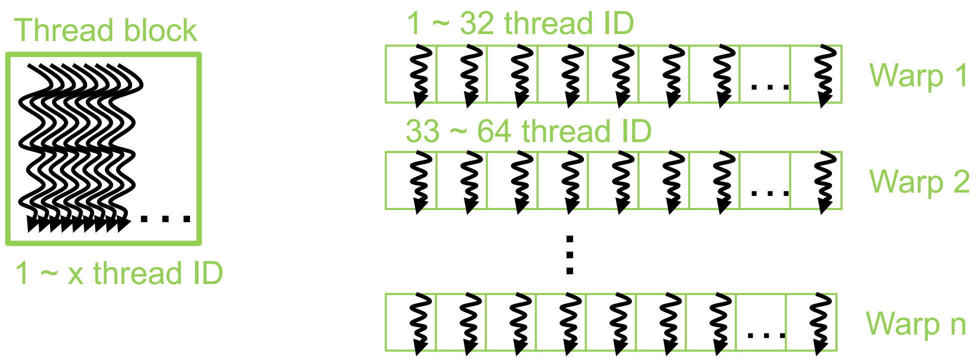
ECLab
Embedded Computing Lab.

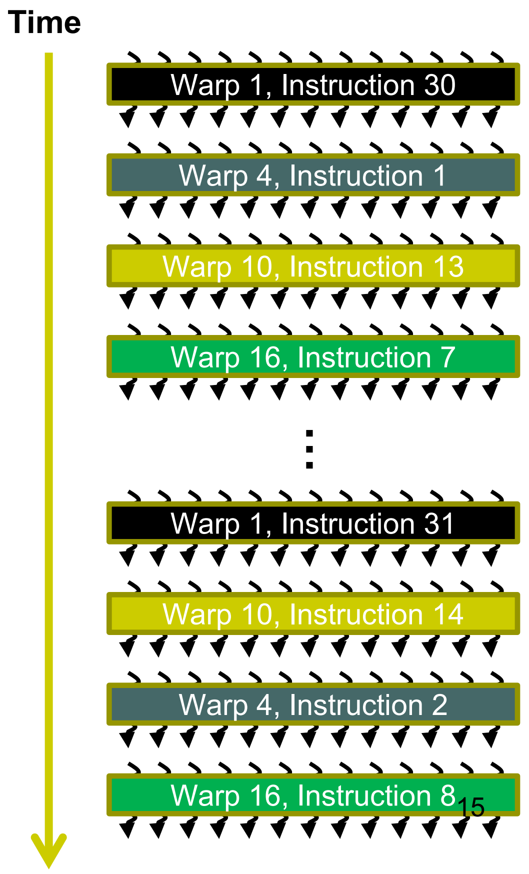
Example: NVIDIA Tesla



SIMT Execution Model of GPUs

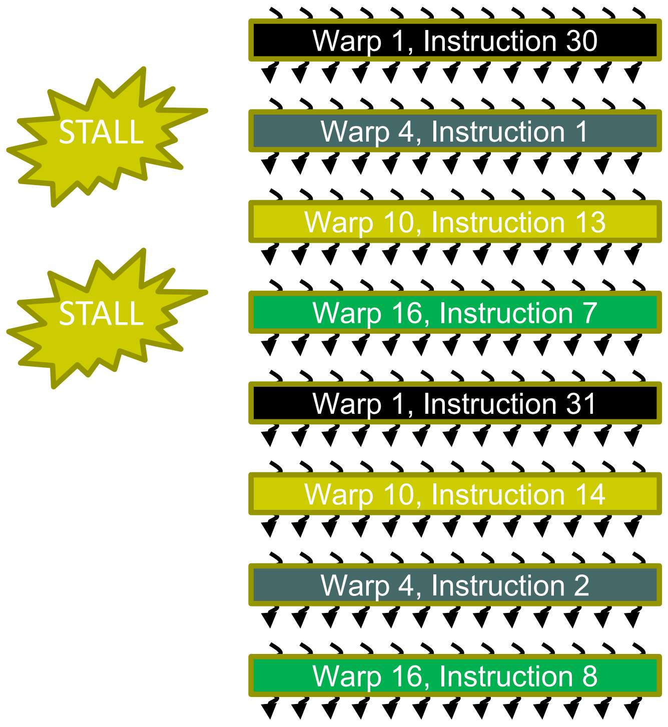
- SIMT (Single Instruction Multiple Threads)
  - Warp
    - A group of threads (pixel, vertex, compute…)
    - Basic scheduling/execution unit
    - Common PC value

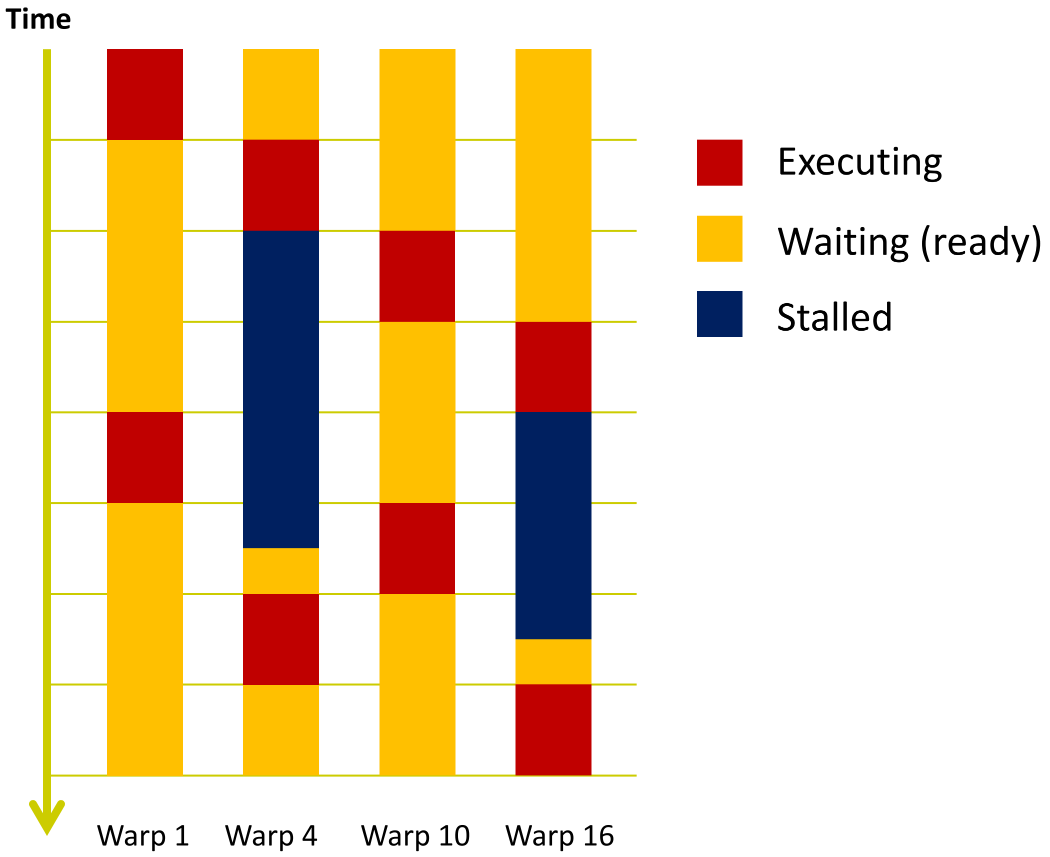




Latency Hiding

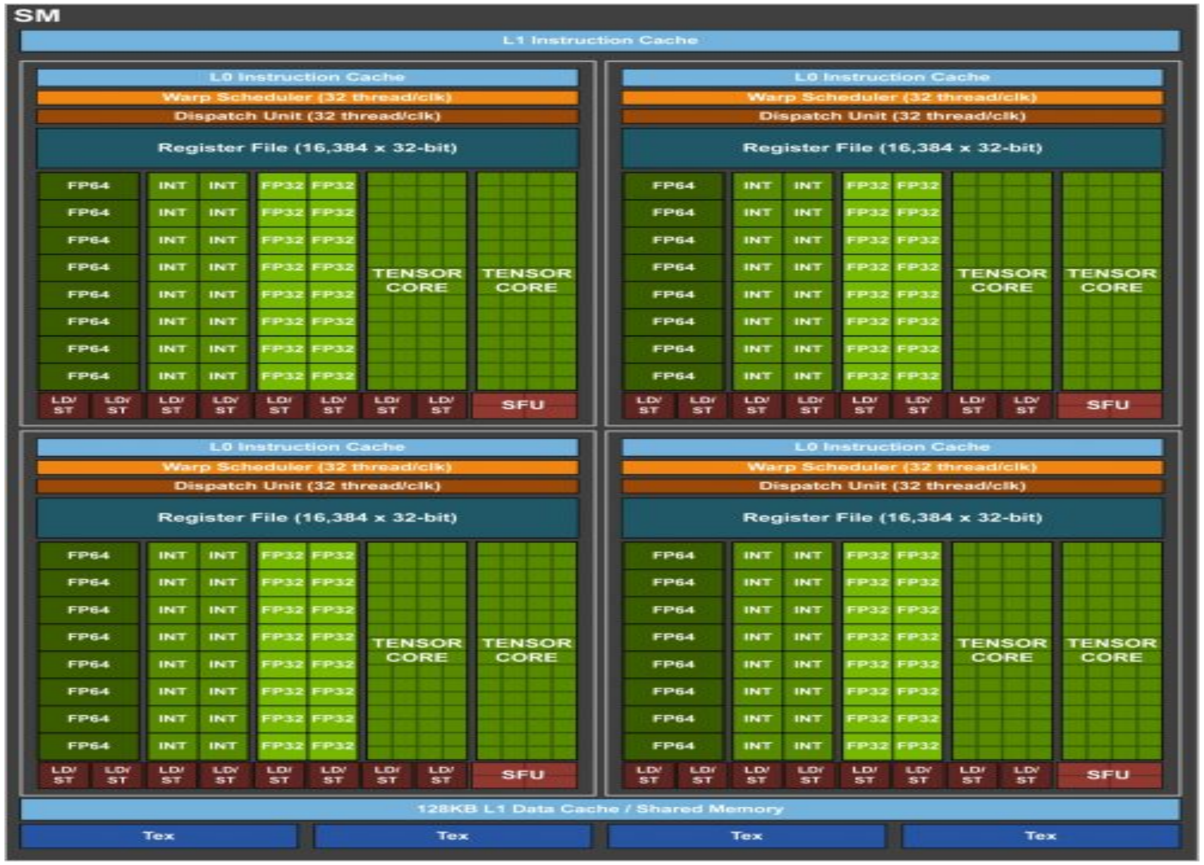
- **Interleaved warp execution**

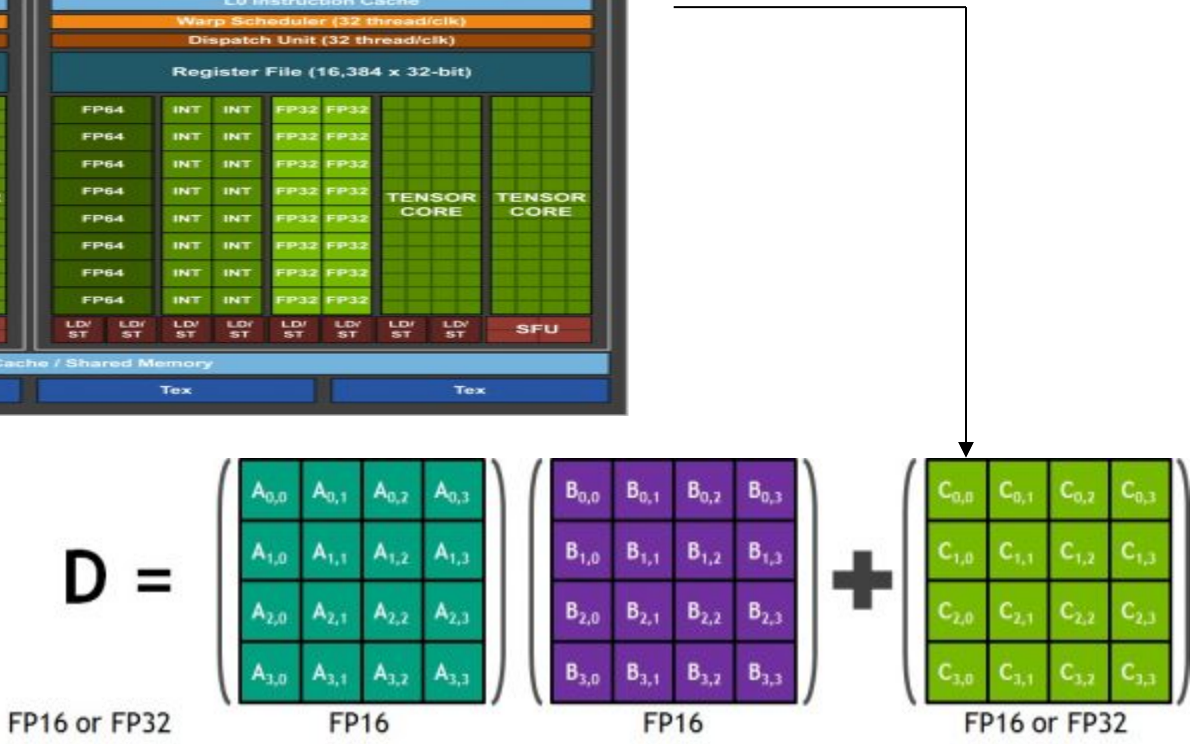




Embedded Computing Lab.

Volta

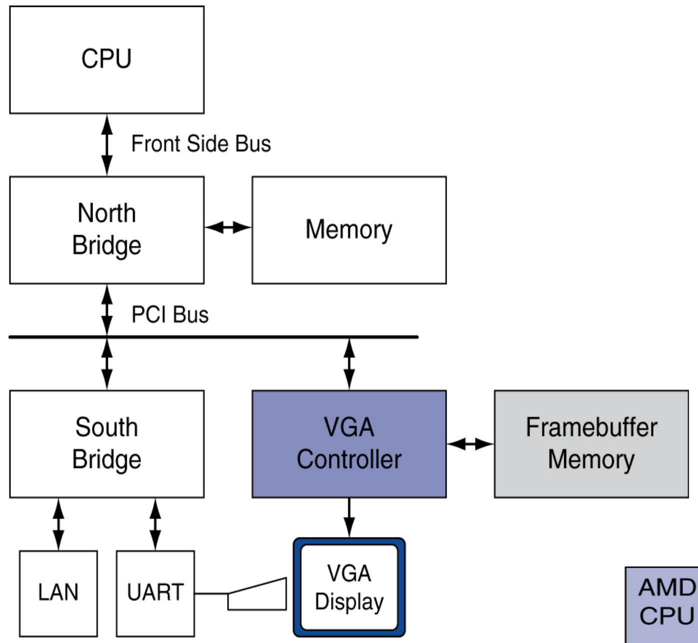

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

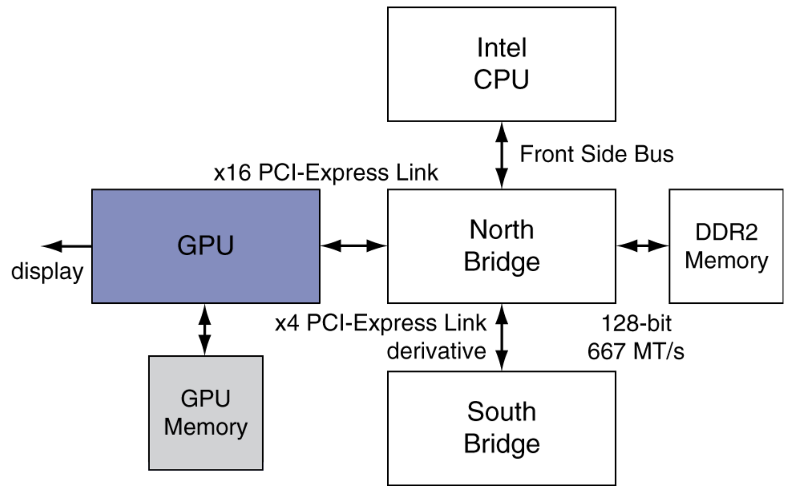

D =
$A_{0,0}$ $A_{0,1}$ $A_{0,2}$ $A_{0,3}$
$A_{1,0}$ $A_{1,1}$ $A_{1,2}$ $A_{1,3}$
$A_{2,0}$ $A_{2,1}$ $A_{2,2}$ $A_{2,3}$
$A_{3,0}$ $A_{3,1}$ $A_{3,2}$ $A_{3,3}$
$B_{0,0}$ $B_{0,1}$ $B_{0,2}$ $B_{0,3}$
$B_{1,0}$ $B_{1,1}$ $B_{1,2}$ $B_{1,3}$
$B_{2,0}$ $B_{2,1}$ $B_{2,2}$ $B_{2,3}$
$B_{3,0}$ $B_{3,1}$ $B_{3,2}$ $B_{3,3}$
+
$C_{0,0}$ $C_{0,1}$ $C_{0,2}$ $C_{0,3}$
$C_{1,0}$ $C_{1,1}$ $C_{1,2}$ $C_{1,3}$
$C_{2,0}$ $C_{2,1}$ $C_{2,2}$ $C_{2,3}$
$C_{3,0}$ $C_{3,1}$ $C_{3,2}$ $C_{3,3}$
FP16 or FP32
FP16
FP16
FP16 or FP32

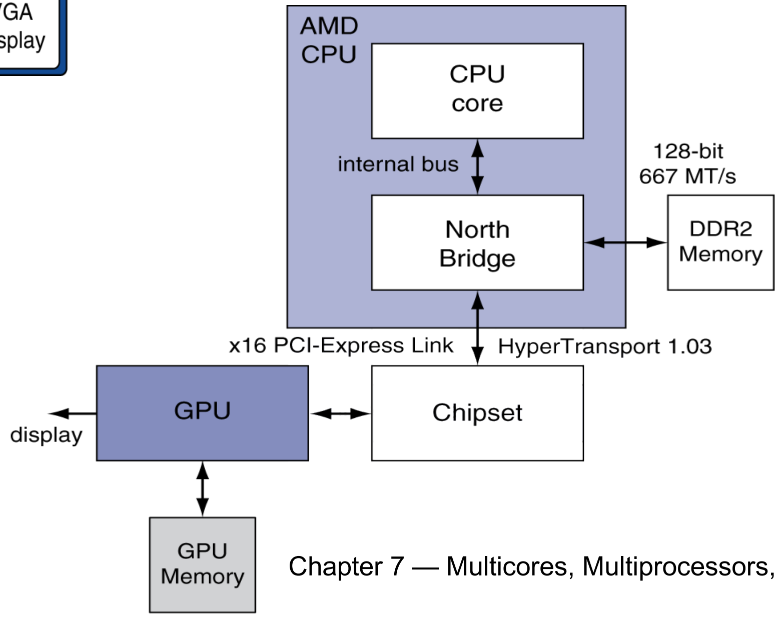
ECLab

Embedded Computing Lab.

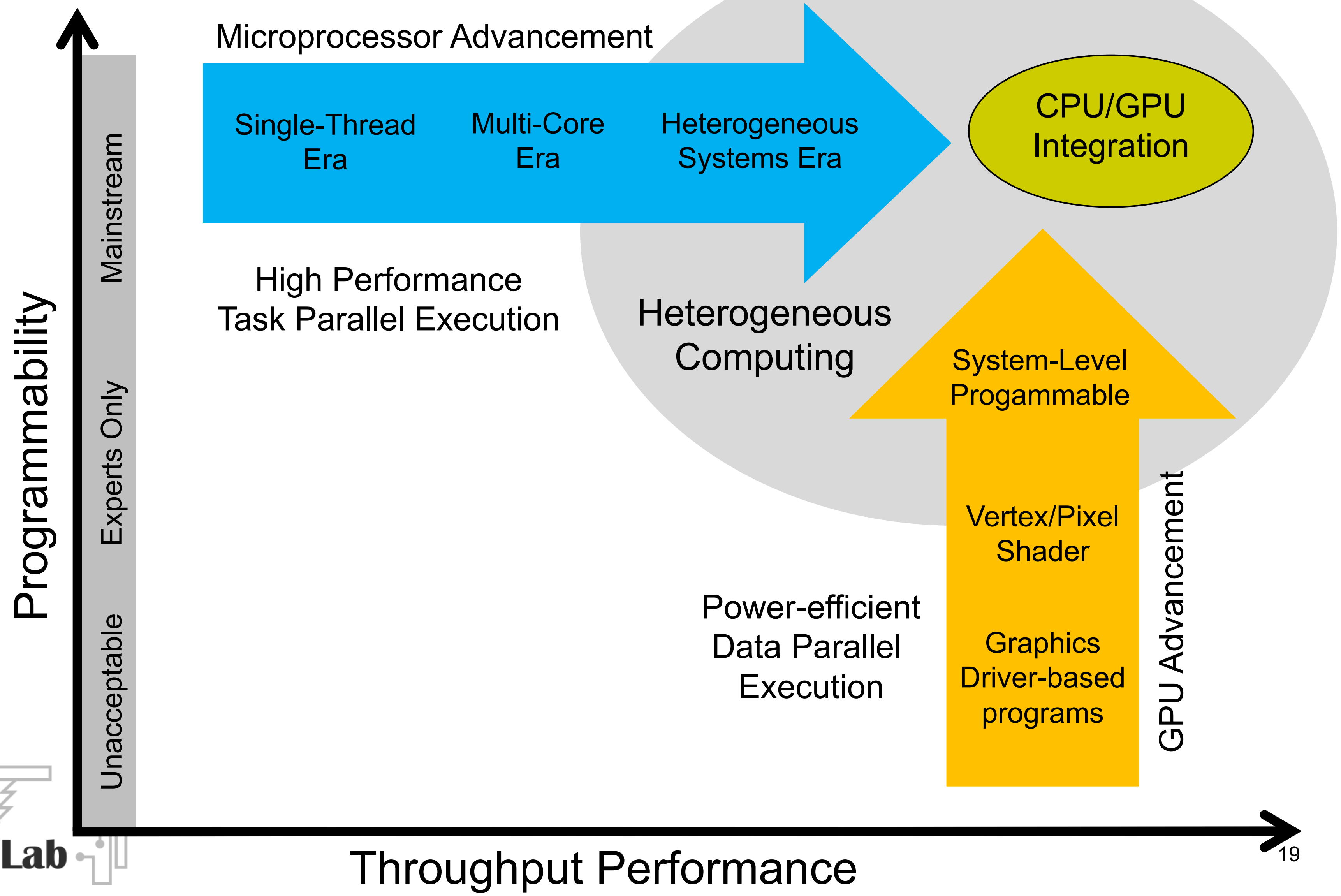
Graphics in the System







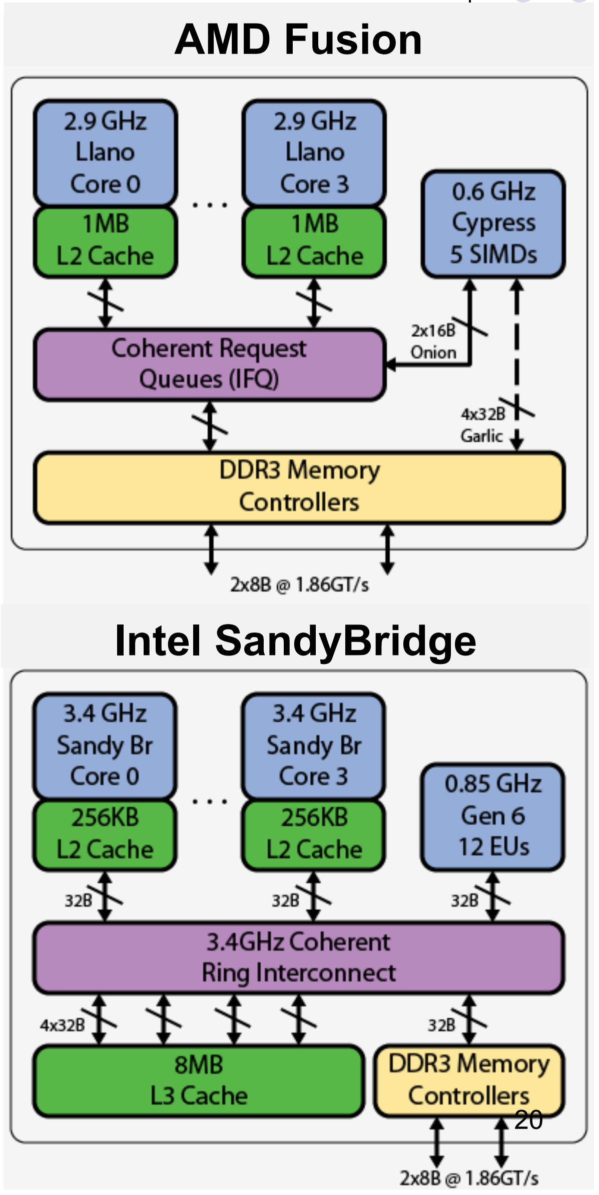
CPU/GPU Integration: CPU's Advancement Meets GPU's



ECLab
Embedded Computing Lab.

Heterogeneous Computing ~ 2011

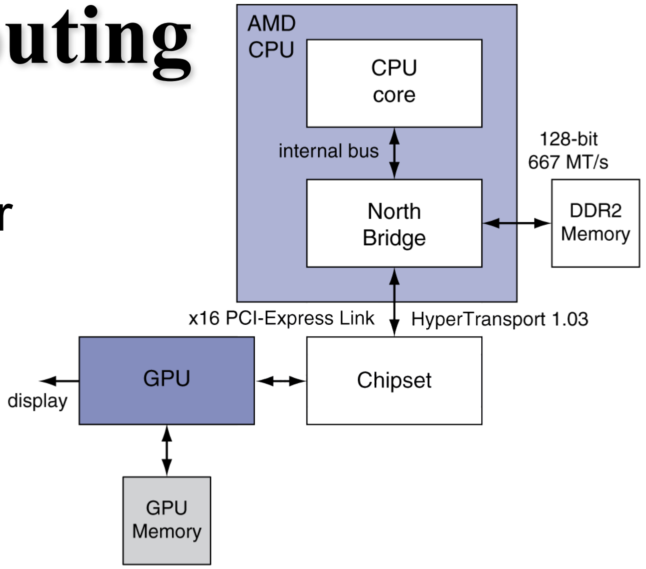
- Intel SandyBridge
  - Shared last-level cache (LLC) and main memory
- AMD Fusion APU (Accelerated Processing Unit)
  - Shared main memory

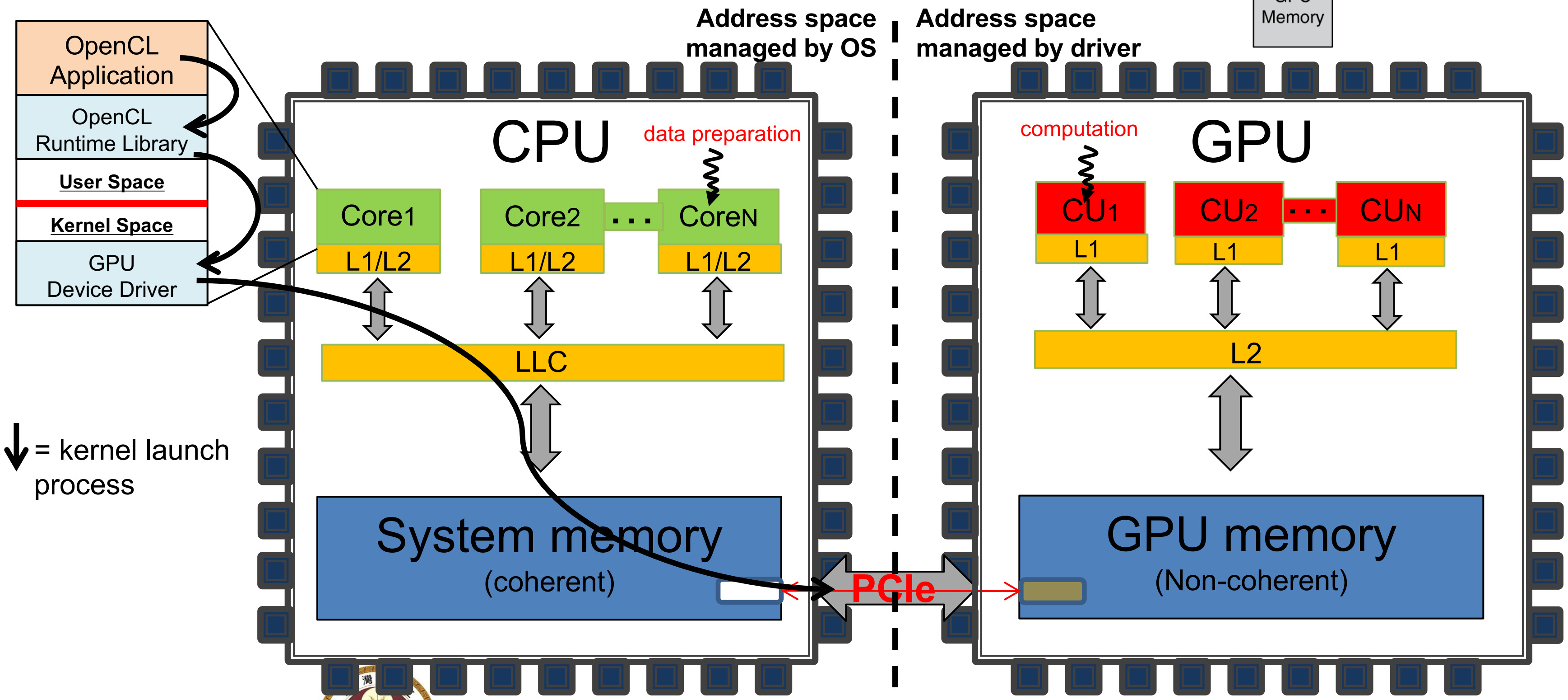


ECLab
Embedded Computing Lab.

Evolution of Heterogeneous Computing

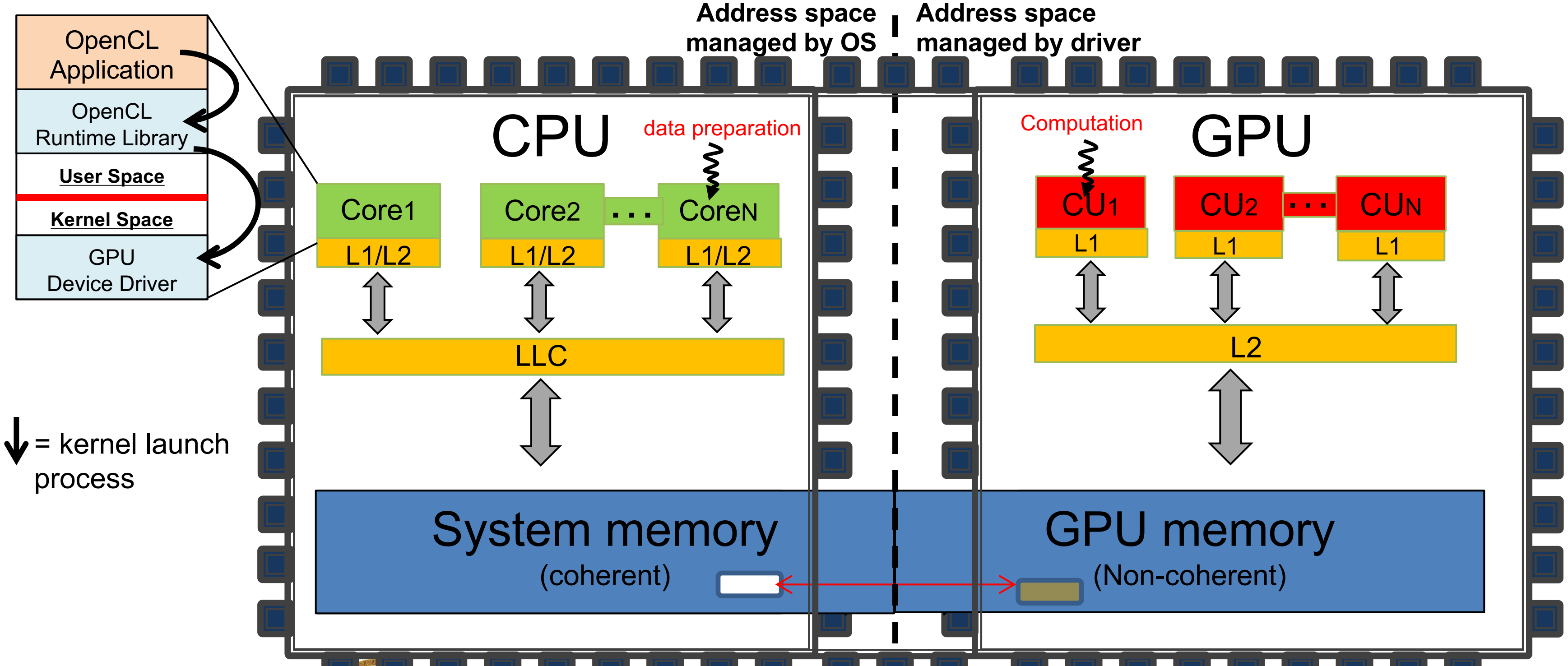
- **Dedicated GPU**
  - GPU kernel is launched through the device driver
  - Separate CPU/GPU address space
  - Separate system/GPU memory
  - Data copy between CPU/GPU via PCIe





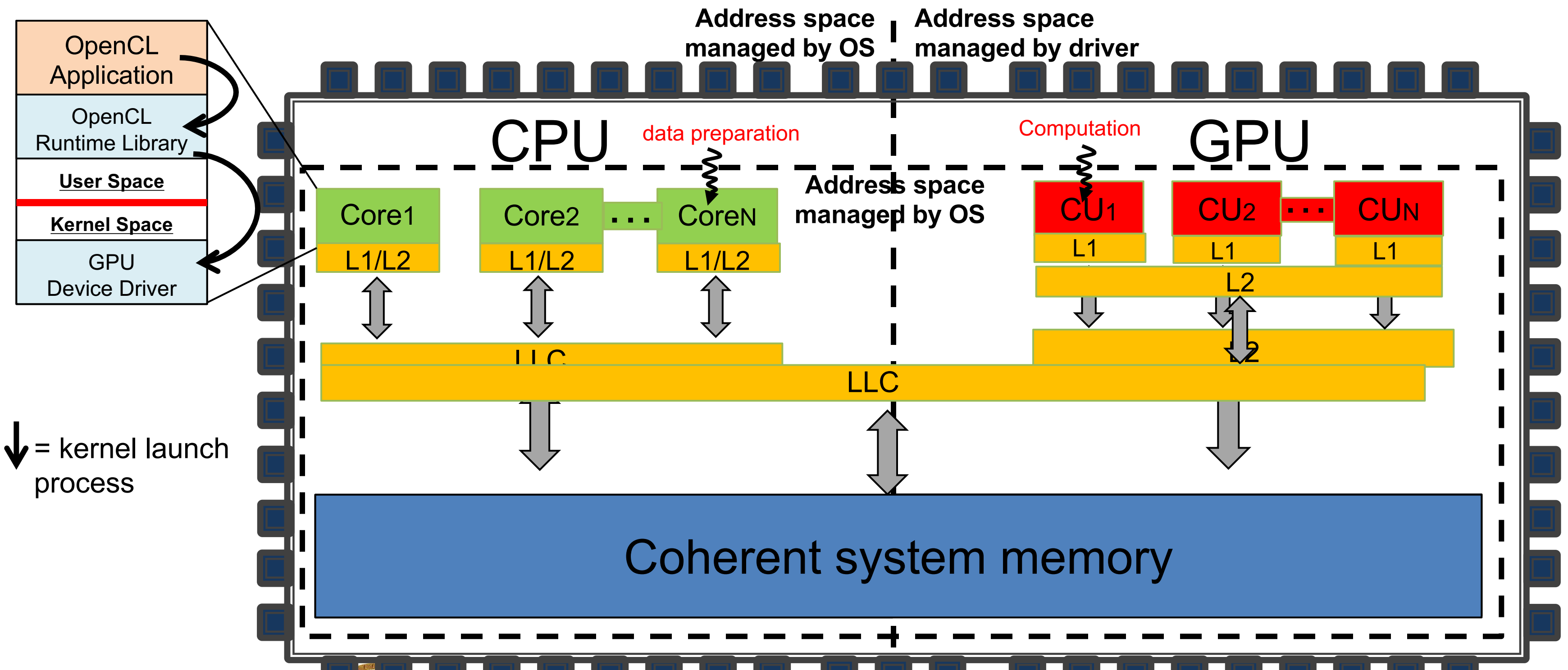
Evolution of Heterogeneous Computing

- **Integrated GPU architecture**
  - GPU kernel is launched through the device driver
  - Separate CPU/GPU address space
  - Separate system/GPU memory
  - Data copy between CPU/GPU *via memory bus*



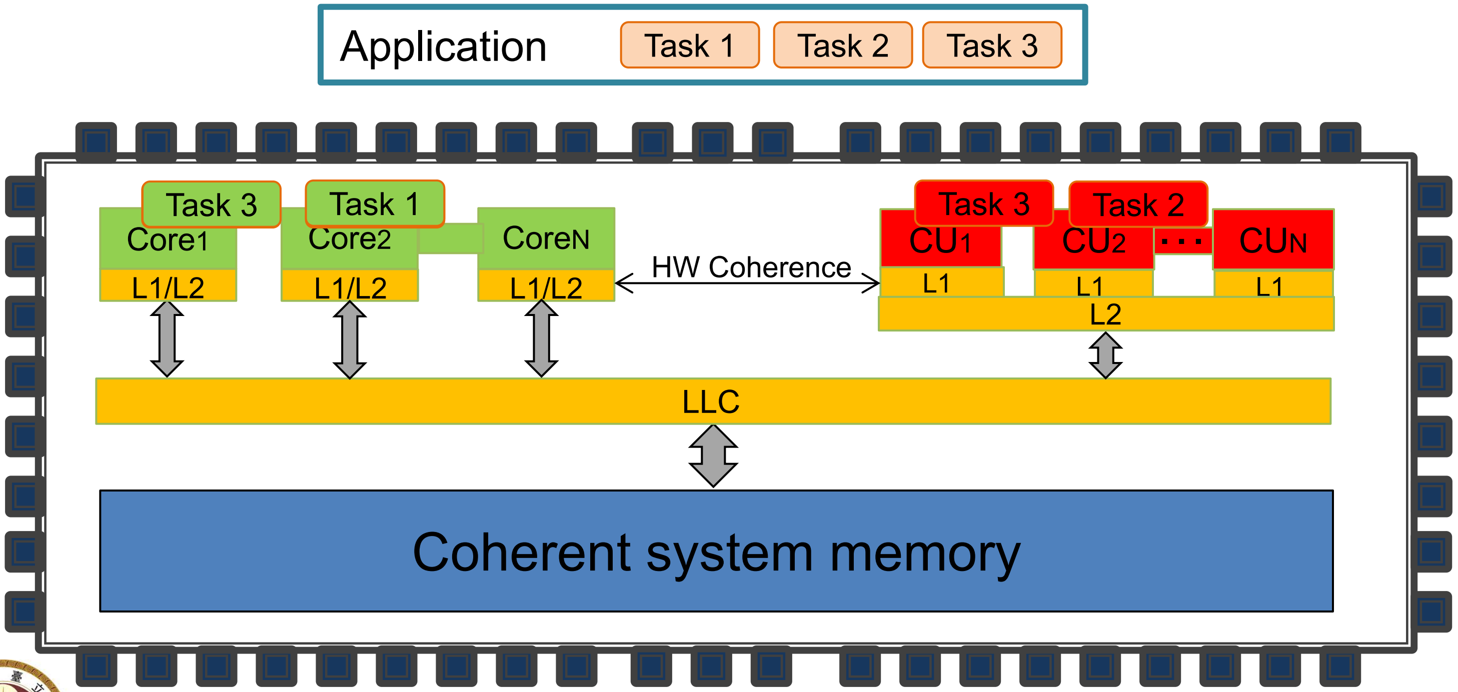
Evolution of Heterogeneous Computing

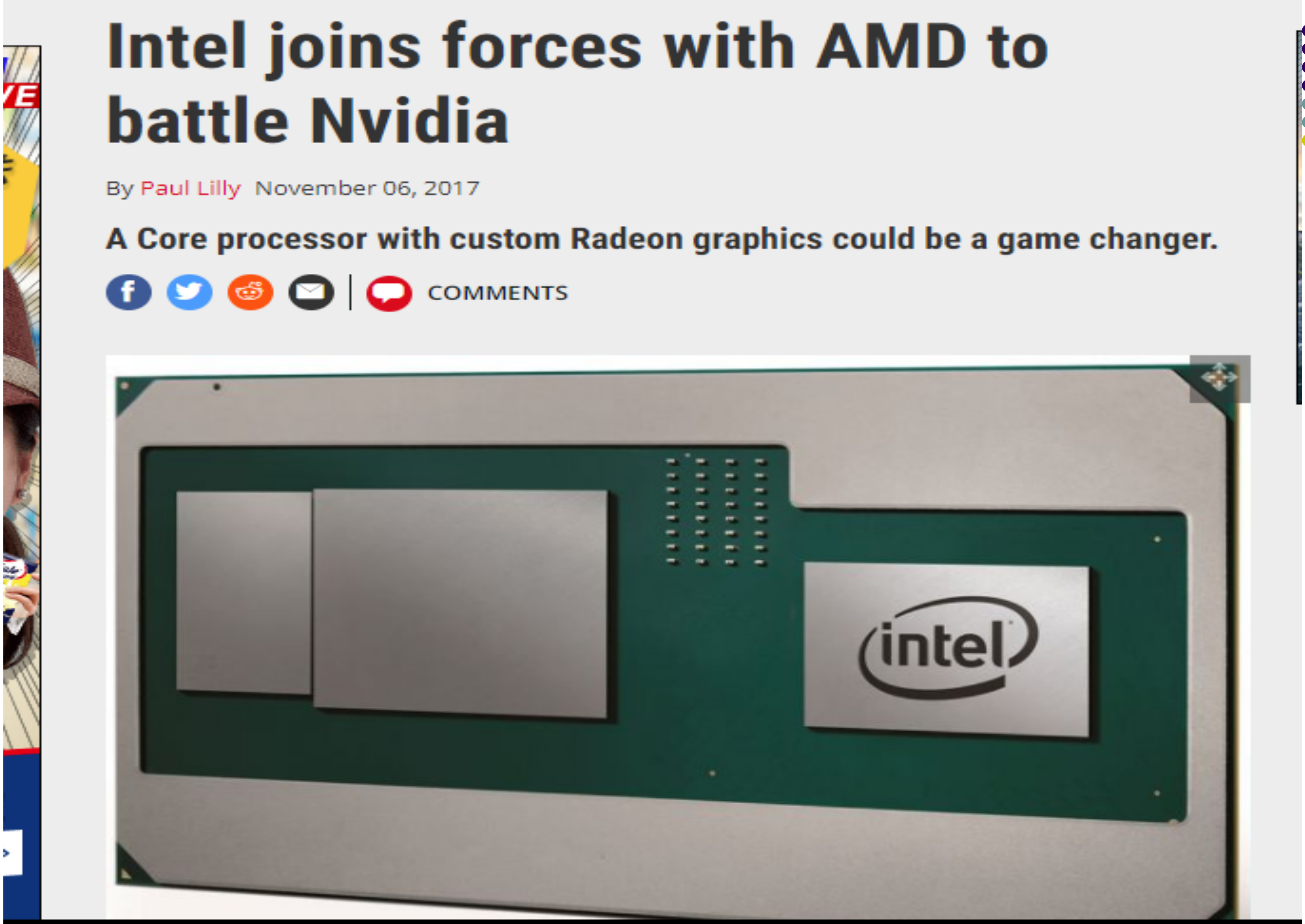
- **Integrated GPU architecture**
  - GPU kernel is launched through the device driver
  - Unified CPU/GPU address space (managed by OS)
  - Unified system/GPU memory
  - No data copy - data can be retrieved by pointer passing



Utopia World of Heterogeneous Computing

- Processors are architected to operate cooperatively
  - Tasks in an application are executed on different types of core
  - Unified coherent memory enables data sharing across all processors
- Designed to enable the applications to run on different processors at different time
  - Capability to translate from high-level language to target binary at run-time
  - User-level task dispatch
  - Decision making module





Intel joins forces with AMD to battle Nvidia

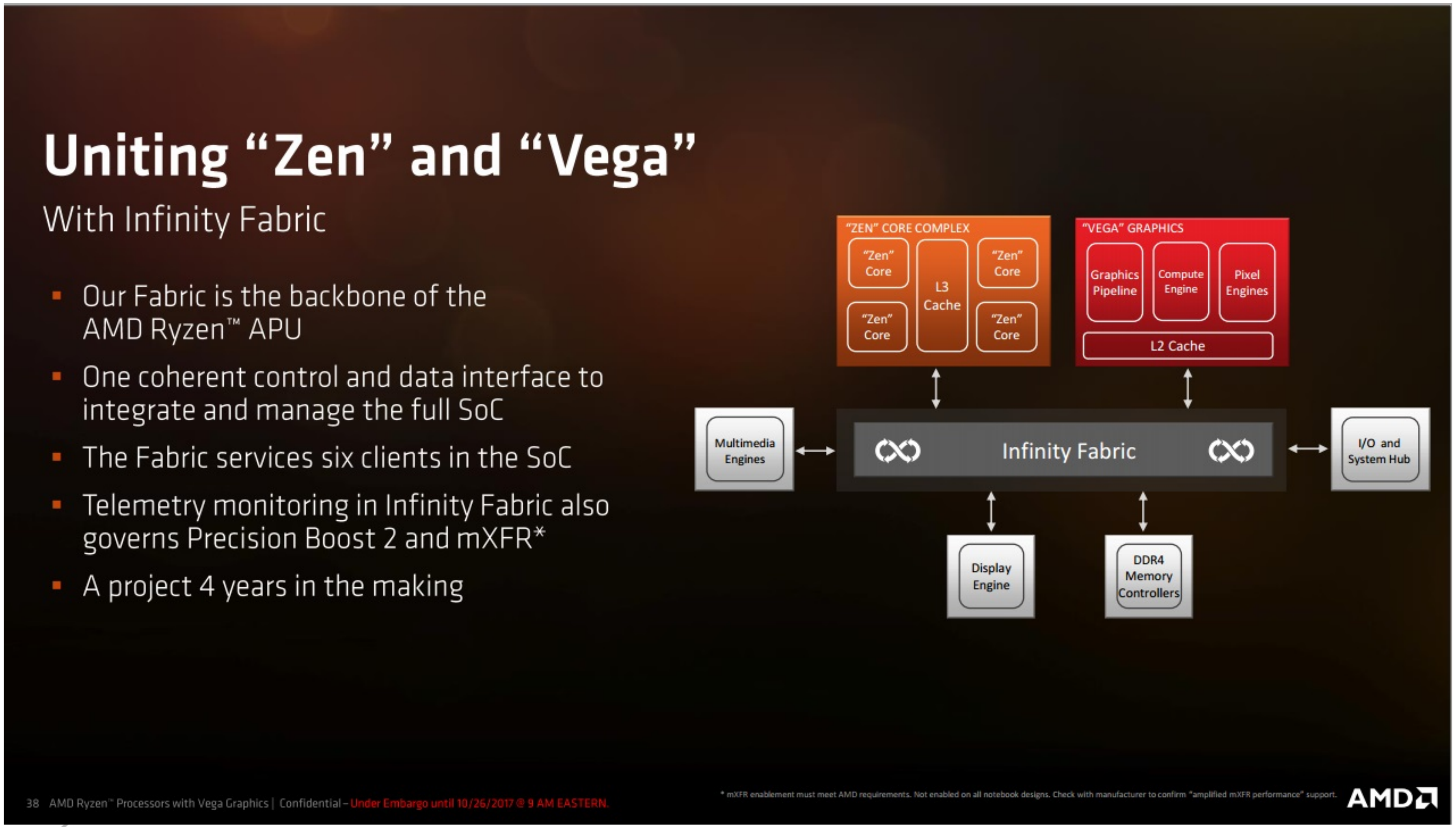
By Paul Lilly November 06, 2017

A Core processor with custom Radeon graphics could be a game changer.

COMMENTS

- https://www.youtube.com/watch?time_continue=84&v=gaHs_guCp2o

AMD Ryzen



Uniting "Zen" and "Vega"

With Infinity Fabric

- Our Fabric is the backbone of the AMD Ryzen™ APU
- One coherent control and data interface to integrate and manage the full SoC
- The Fabric services six clients in the SoC
- Telemetry monitoring in Infinity Fabric also governs Precision Boost 2 and mXFR*
- A project 4 years in the making

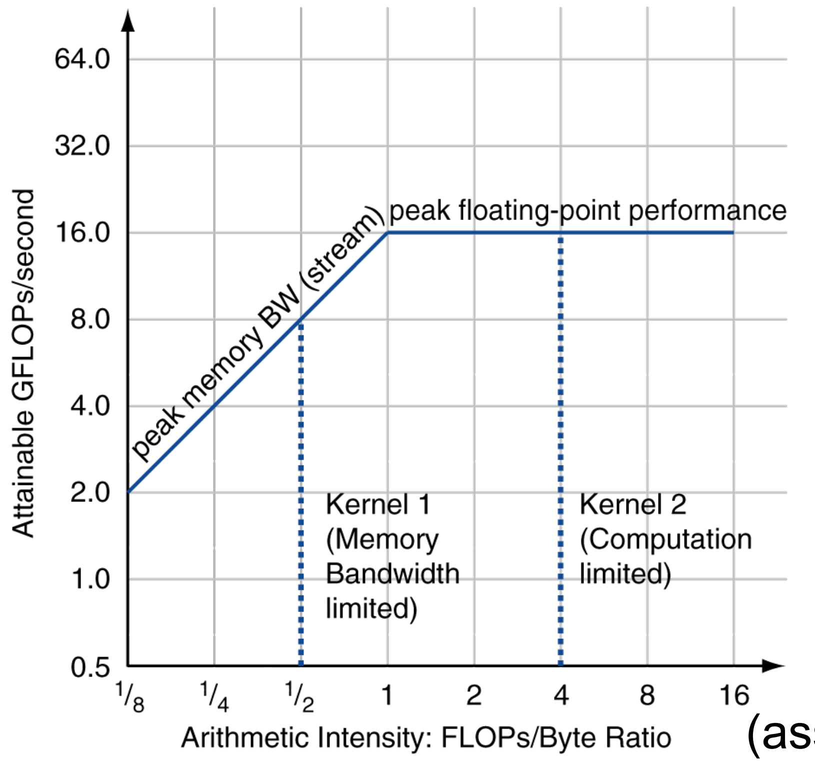
"ZEN" CORE COMPLEX
"Zen" Core
"Zen" Core
L3 Cache
"Zen" Core
"Zen" Core
"VEGA" GRAPHICS
Graphics Pipeline
Compute Engine
Pixel Engines
L2 Cache
Multimedia Engines
Infinity Fabric
I/O and System Hub
Display Engine
DDR4 Memory Controllers

* mXFR enablement must meet AMD requirements. Not enabled on all notebook designs. Check with manufacturer to confirm "amplified mXFR performance" support.

Parallel Benchmarks

- Linpack: matrix linear algebra
- SPECrate: parallel run of SPEC CPU programs
  - Job-level parallelism
- SPLASH: Stanford Parallel Applications for Shared Memory
  - Mix of kernels and applications, strong scaling
- NAS (NASA Advanced Supercomputing) suite
  - computational fluid dynamics kernels
- PARSEC (Princeton Application Repository for Shared Memory Computers) suite
  - Multithreaded applications using Pthreads and OpenMP

Roofline : A Simple Performance Model



$$\frac{\text{Floating-Point Ops/sec}}{\text{Floating-Point Ops/ byte}} = \text{Bytes/sec}$$

Arithmetic Intensity

(assuming 16GB/sec peak bandwidth)

Attainable GFLOPs/sec
= Min (Peak Memory BW × Arithmetic Intensity, Peak FP Performance)