
Accelerated Python
Mark Ebersole, NVIDIA
CUDA Educator
NVIDIA

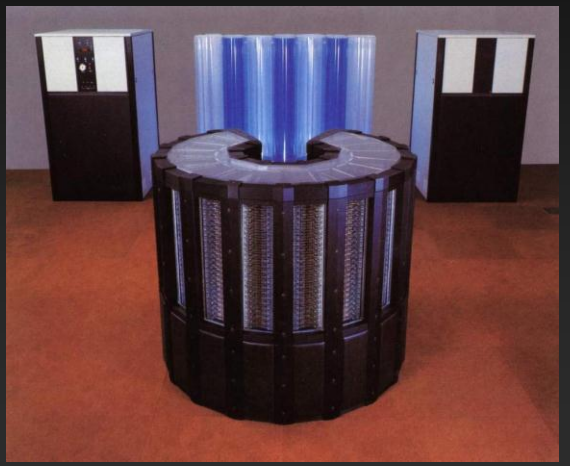
Once upon a time…

Past Massively Parallel Supercomputers

Thinking Machine

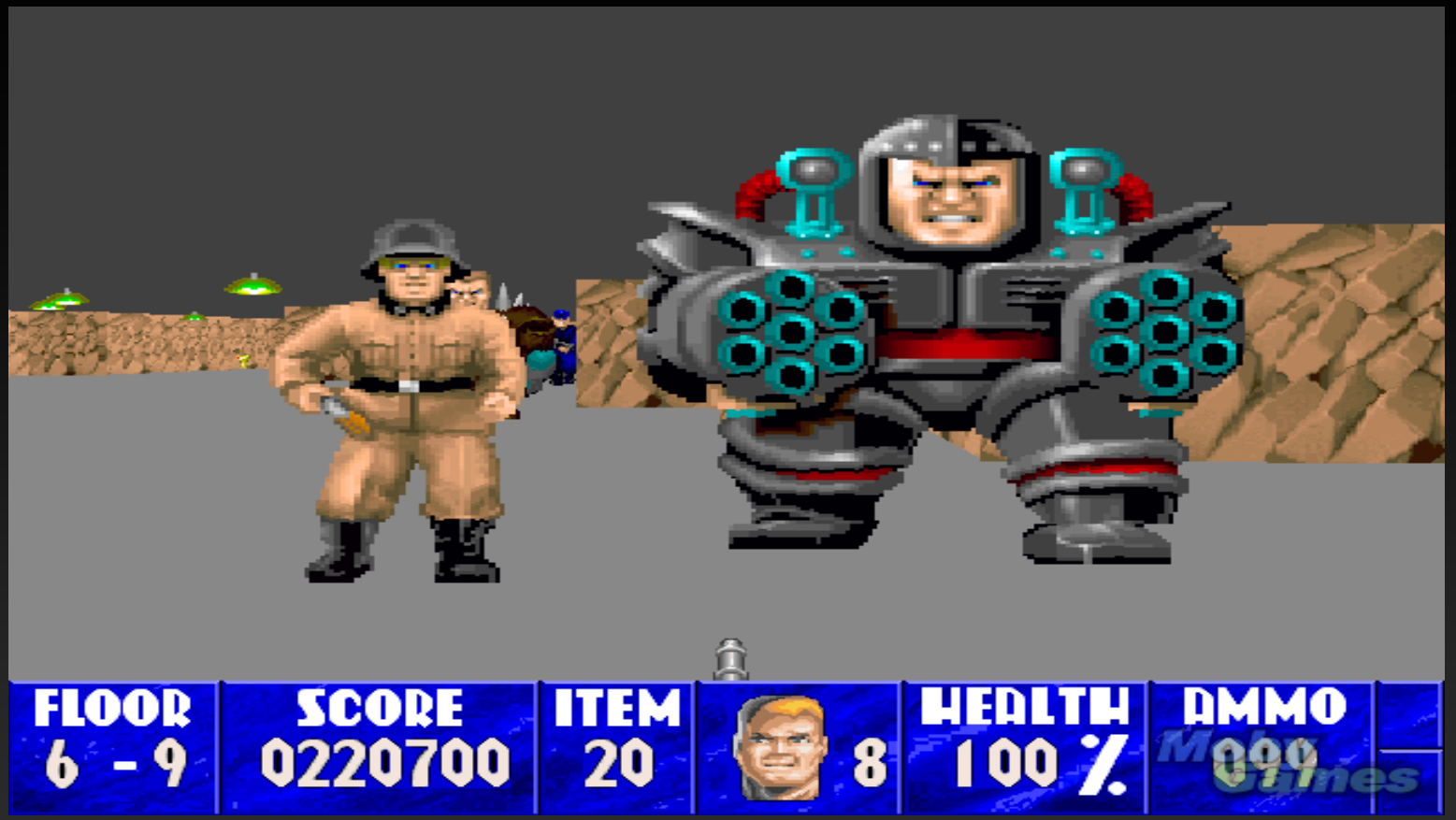
Goodyear MPP

MasPar

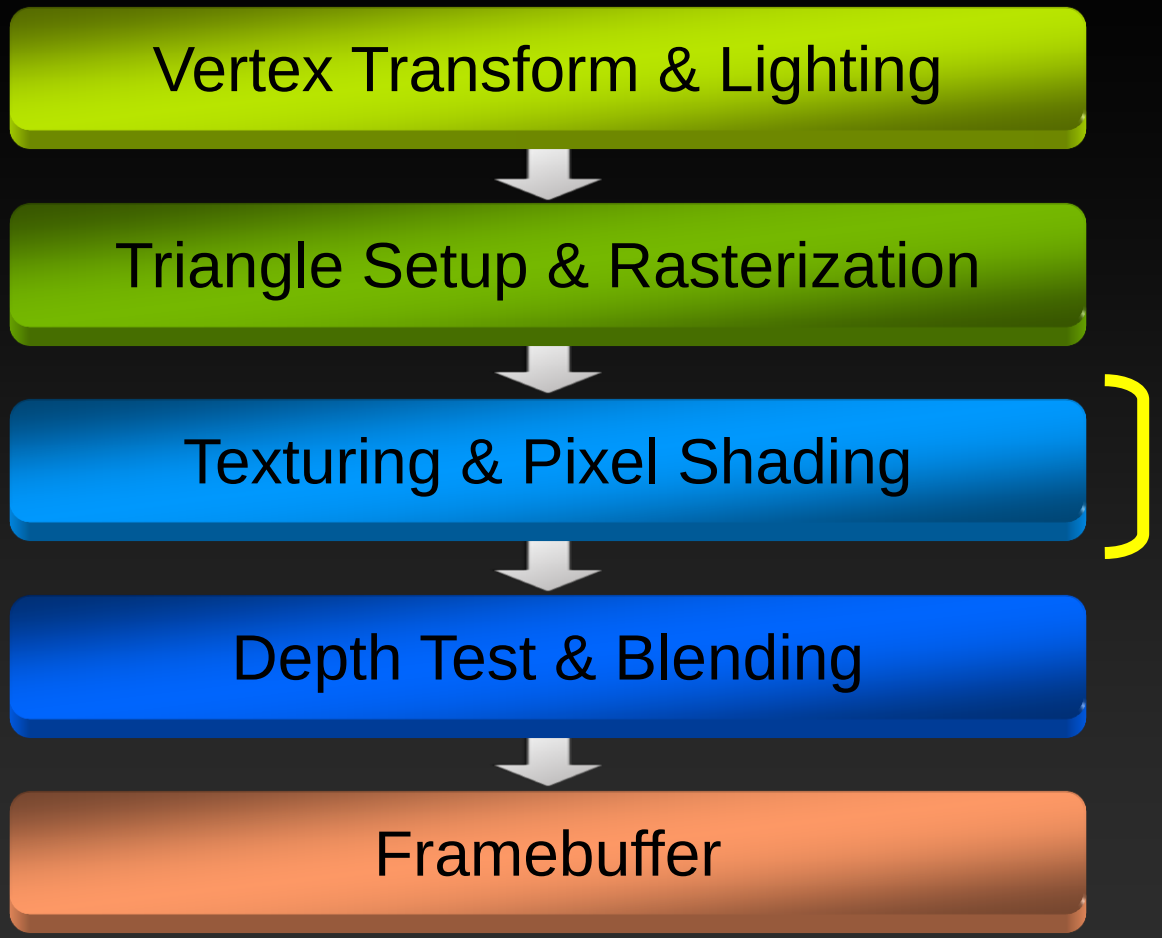


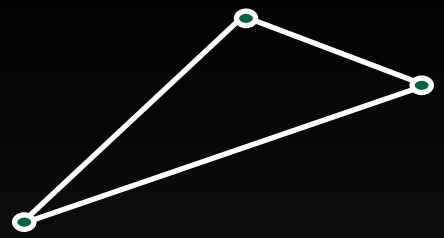
Cray 2

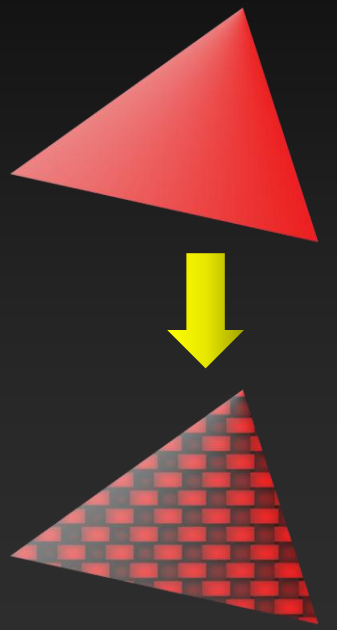
1.31 TFLOPS on DGEMM


FLOOR
6 - 9
SCORE
0220700
ITEM
20
8
HEALTH
100 %
AMMO
099

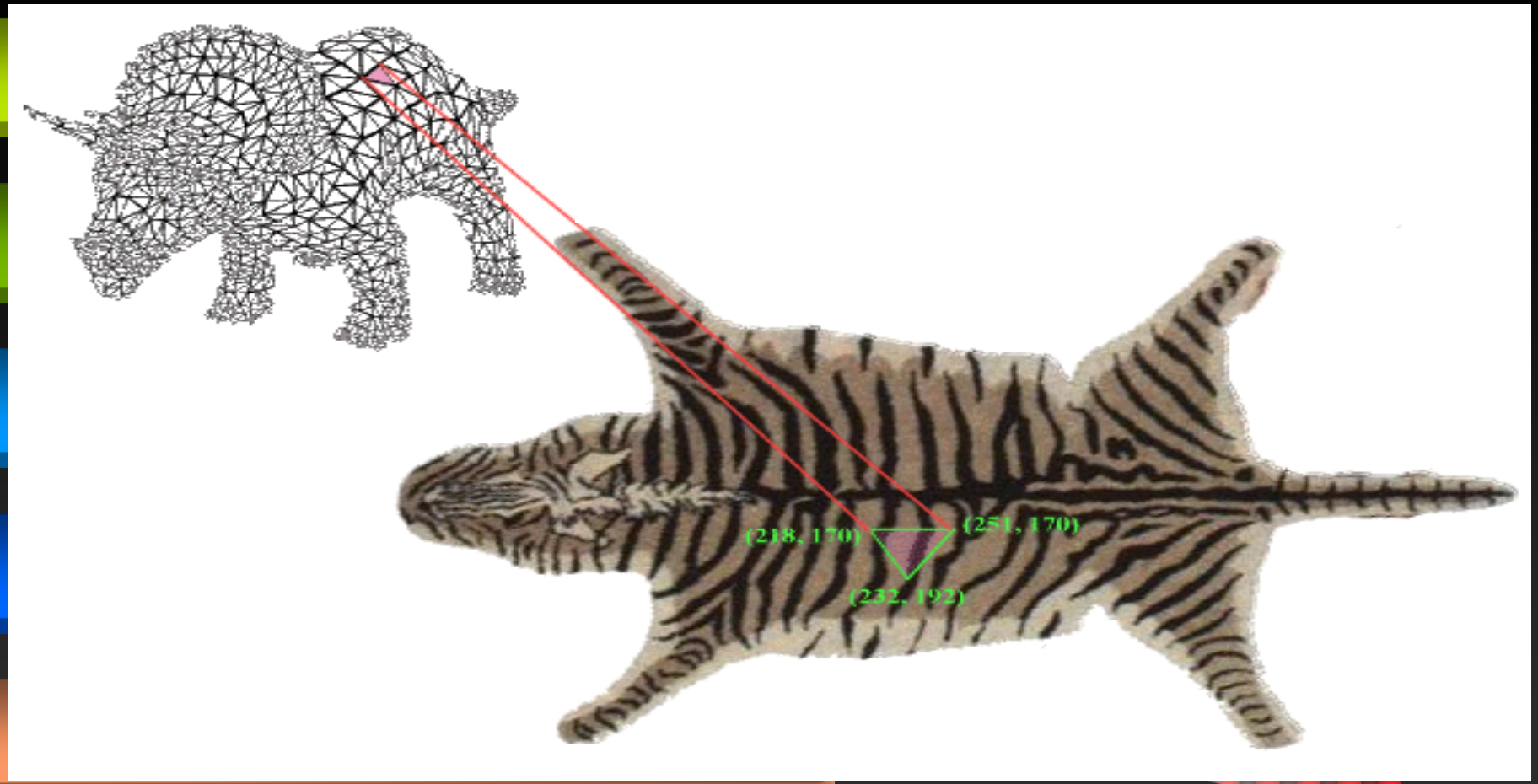
The Graphics Pipeline



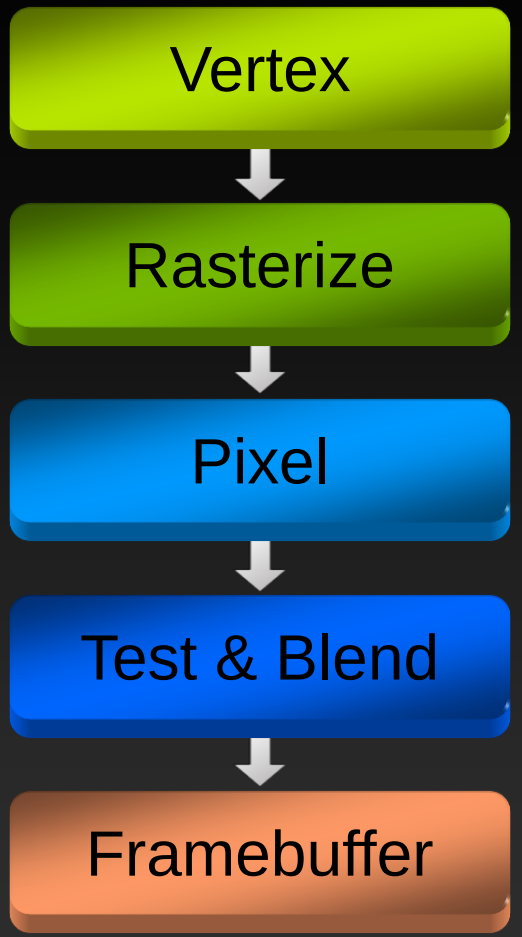




The Graphics Pipeline

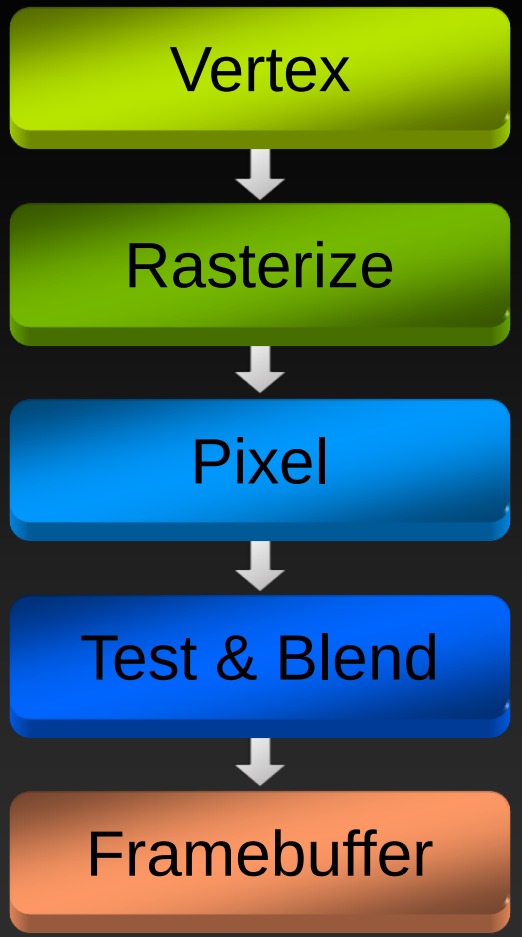


The Graphics Pipeline



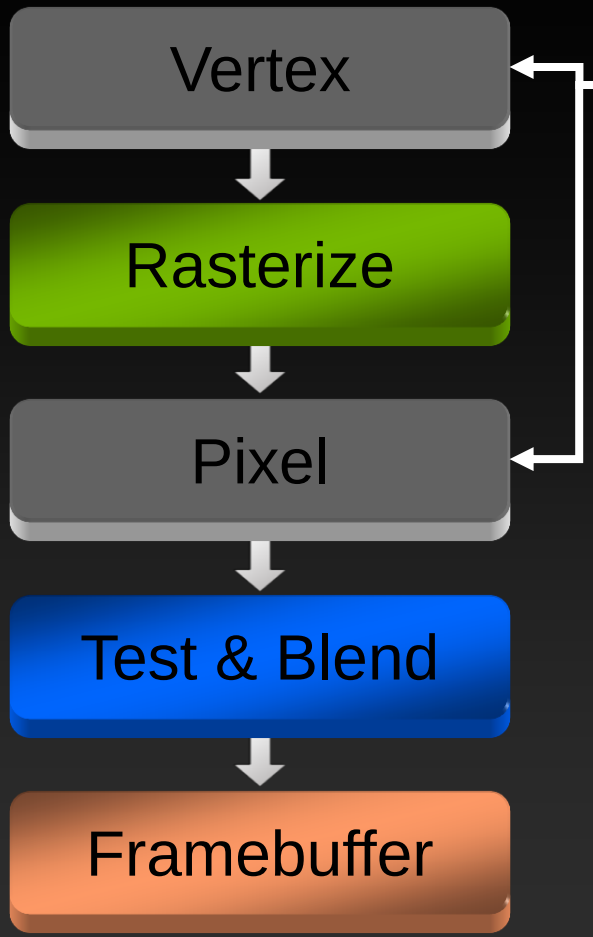
- Key abstraction of real-time graphics
- Hardware used to look like this
- One chip/board per stage
- Fixed data flow through pipeline

The Graphics Pipeline



- Remains a useful abstraction
- Hardware used to look like this

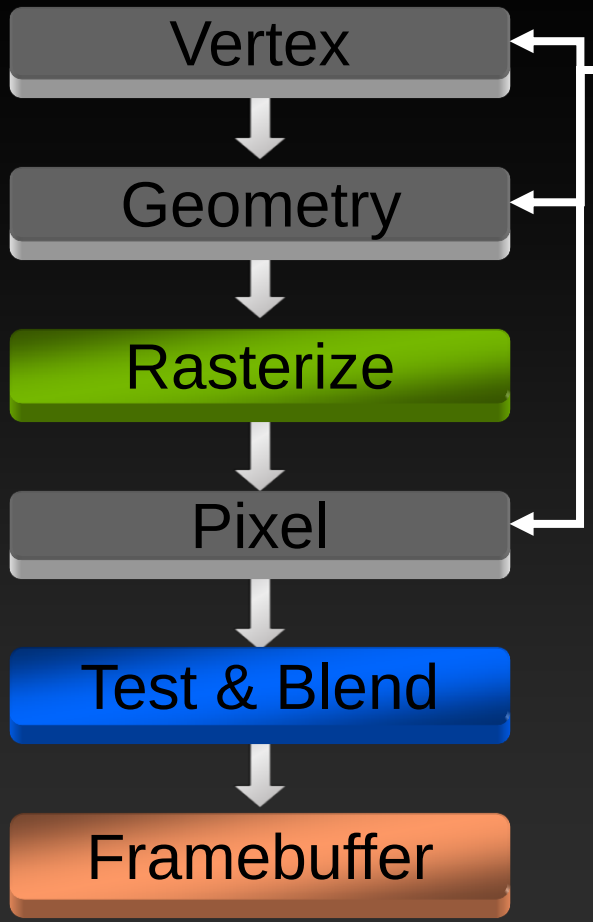
The Graphics Pipeline




```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

- Hardware **used to** look like this:
  - Vertex, pixel processing became programmable

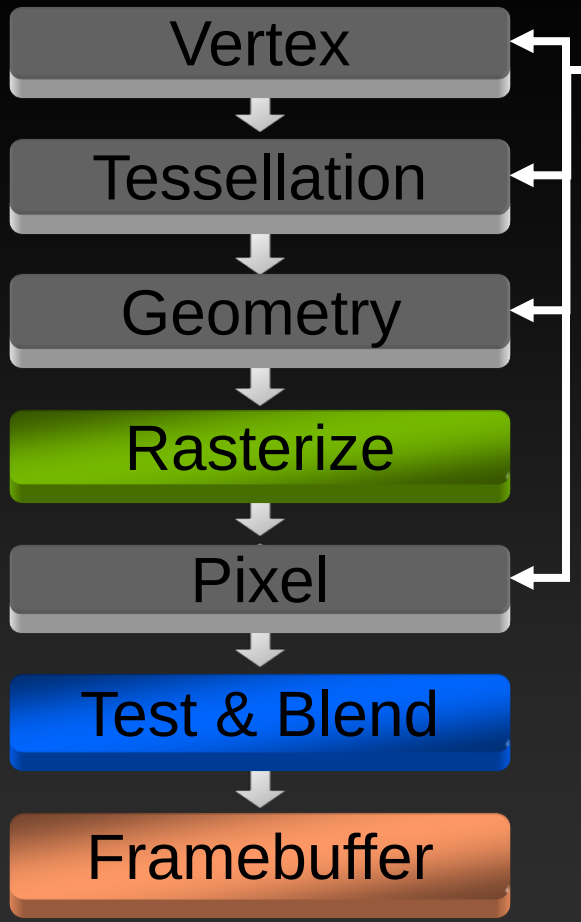
The Graphics Pipeline




```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

- Hardware **used to** look like this
  - Vertex, pixel processing became programmable
  - New stages added

The Graphics Pipeline

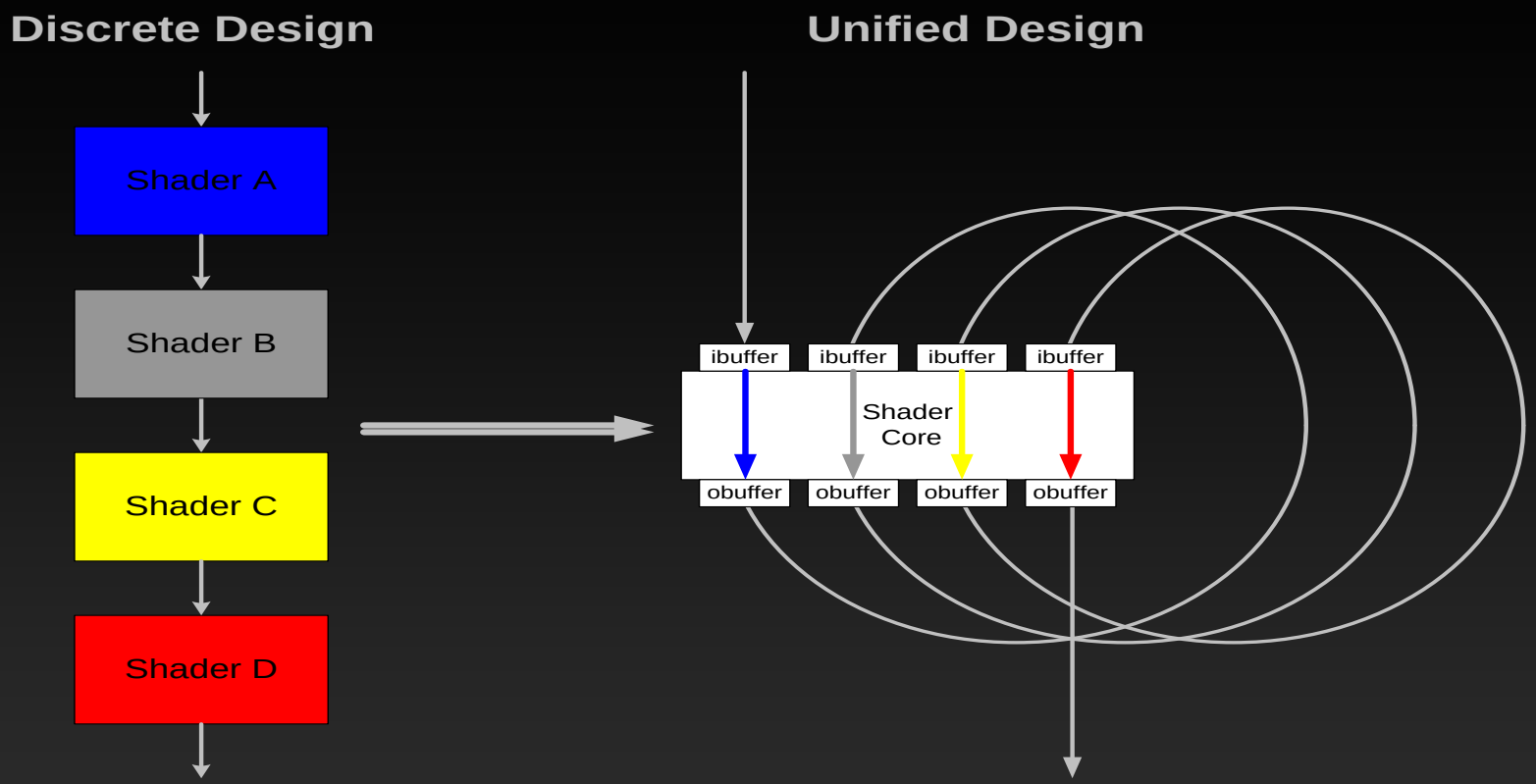



```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

- Hardware **used to** look like this
  - Vertex, pixel processing became programmable
  - New stages added

GPU architecture increasingly centers around shader execution

Modern GPUs: Unified Design



Vertex shaders, pixel shaders, etc. become *threads* running different programs on a flexible core

GeForce 8: First Fully Programmable GPU

Stunning Graphics Realism

Lush, Rich Worlds

Crysis © 2006 Crytek / Electronic Arts

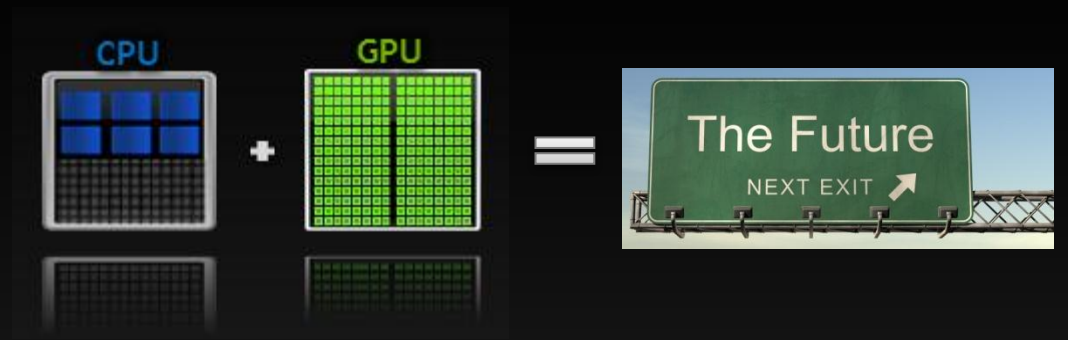
Incredible Physics Effects

Core of the Definitive Gaming Platform

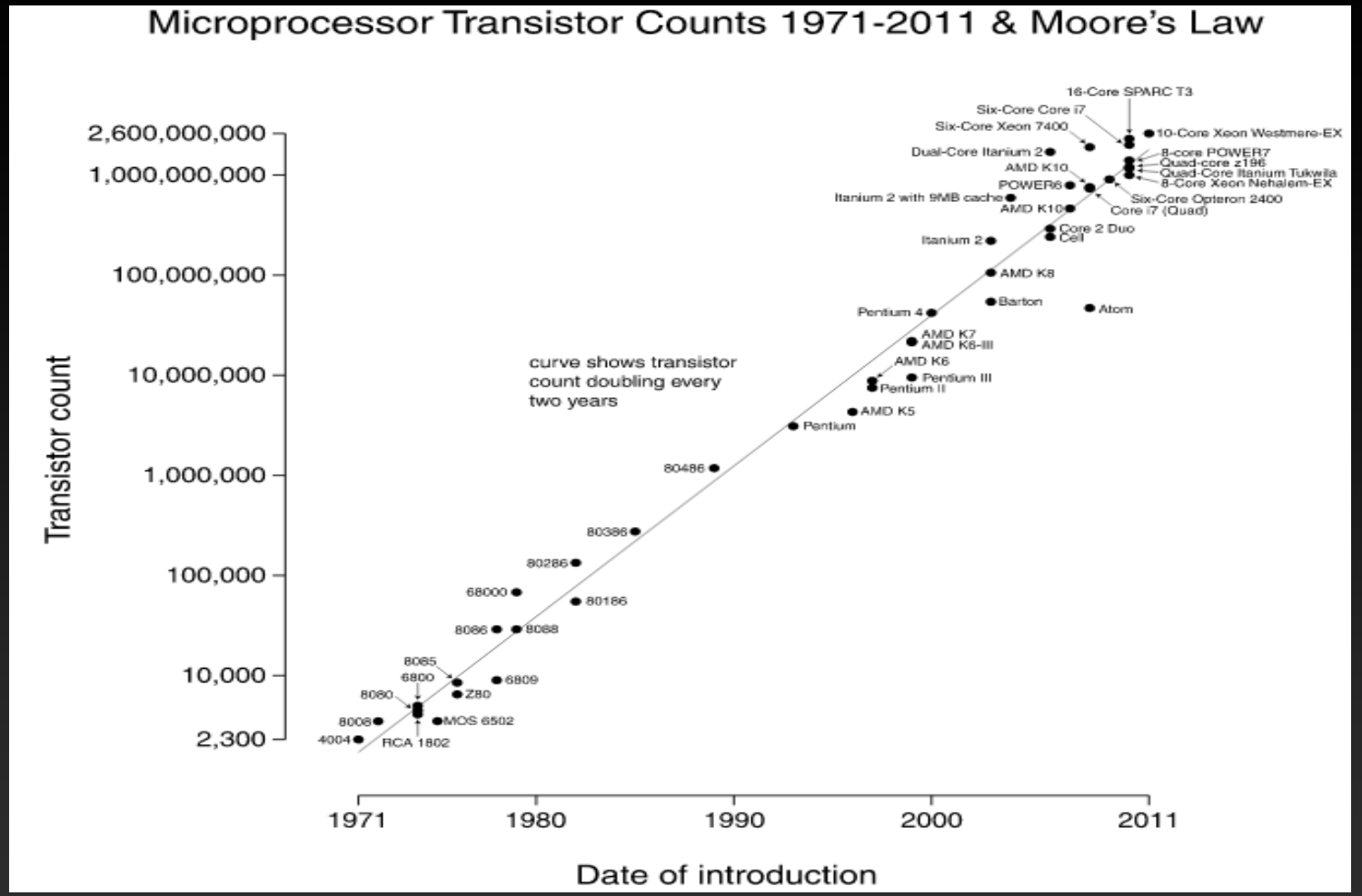
Hellgate: London © 2005-2006 Flagship Studios, Inc. Licensed by NAMCO BANDAI Games America, Inc.

Full Spectrum Warrior: Ten Hammers © 2006 Pandemic Studios, LLC. All rights reserved. © 2006 THQ Inc. All rights reserved.

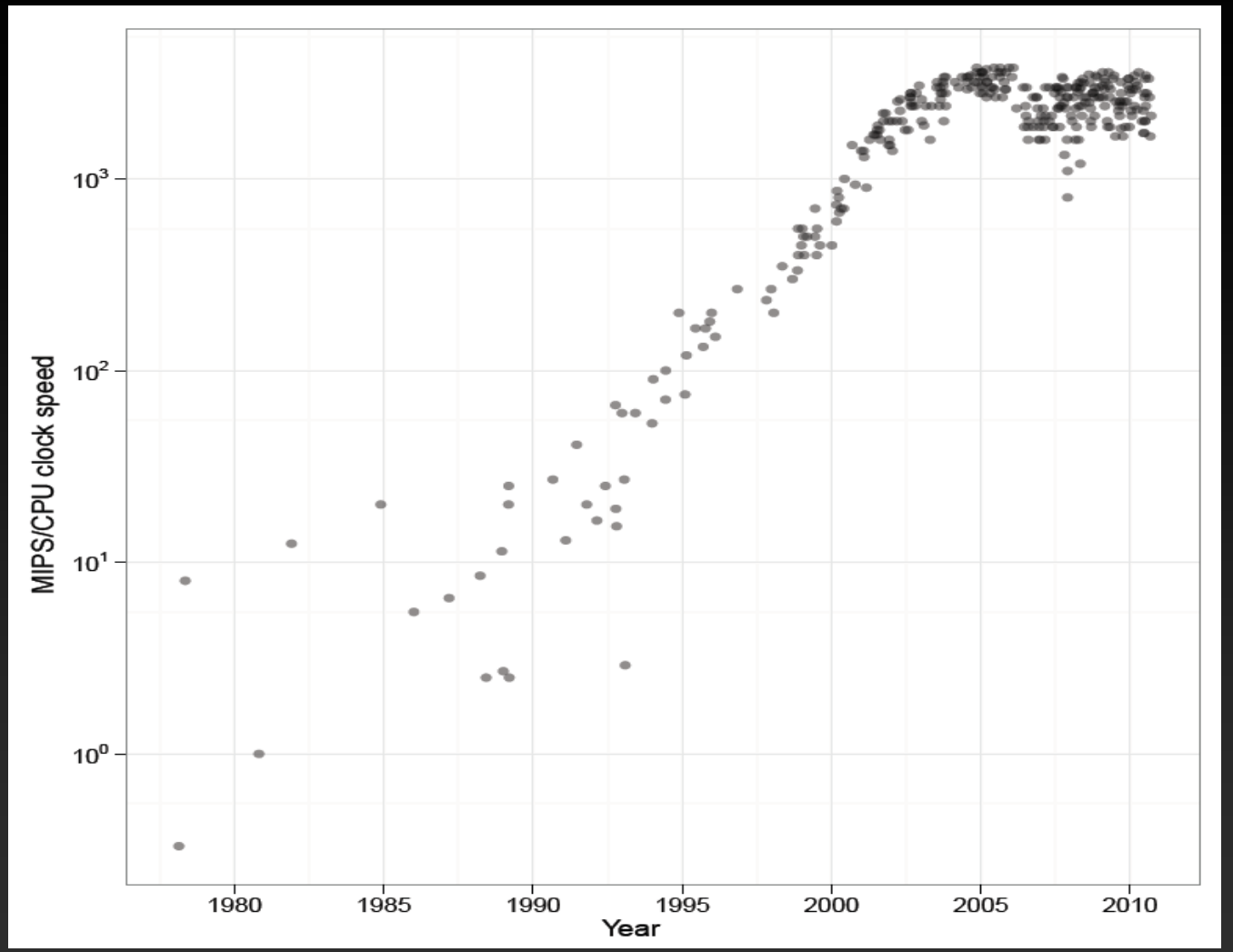
My Three Points



Moore's Law

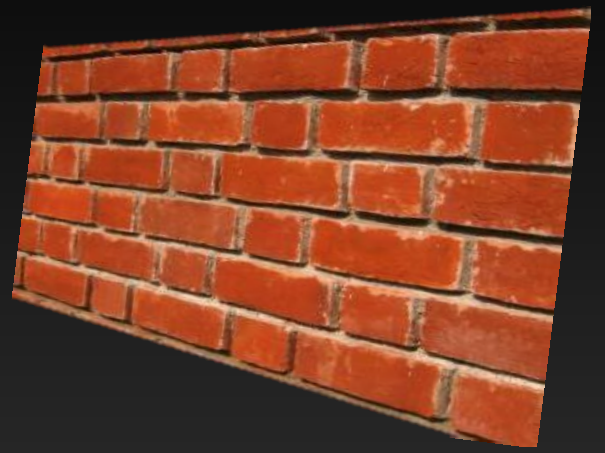

Microprocessor Transistor Counts 1971-2011 & Moore's Law
Transistor count
2,600,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
curve shows transistor count doubling every two years
16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
AMD K10
Quad-Core Itanium Tukwila
POWER6
8-Core Xeon Nehalem-EX
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Cell
Itanium 2
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K8-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium
80486
80386
80286
68000
80186
8086
8088
8085
6800
6809
Z80
8080
8008
MOS 6502
4004
RCA 1802
1971
1980
1990
2000
2011
Date of introduction

Moore's Law

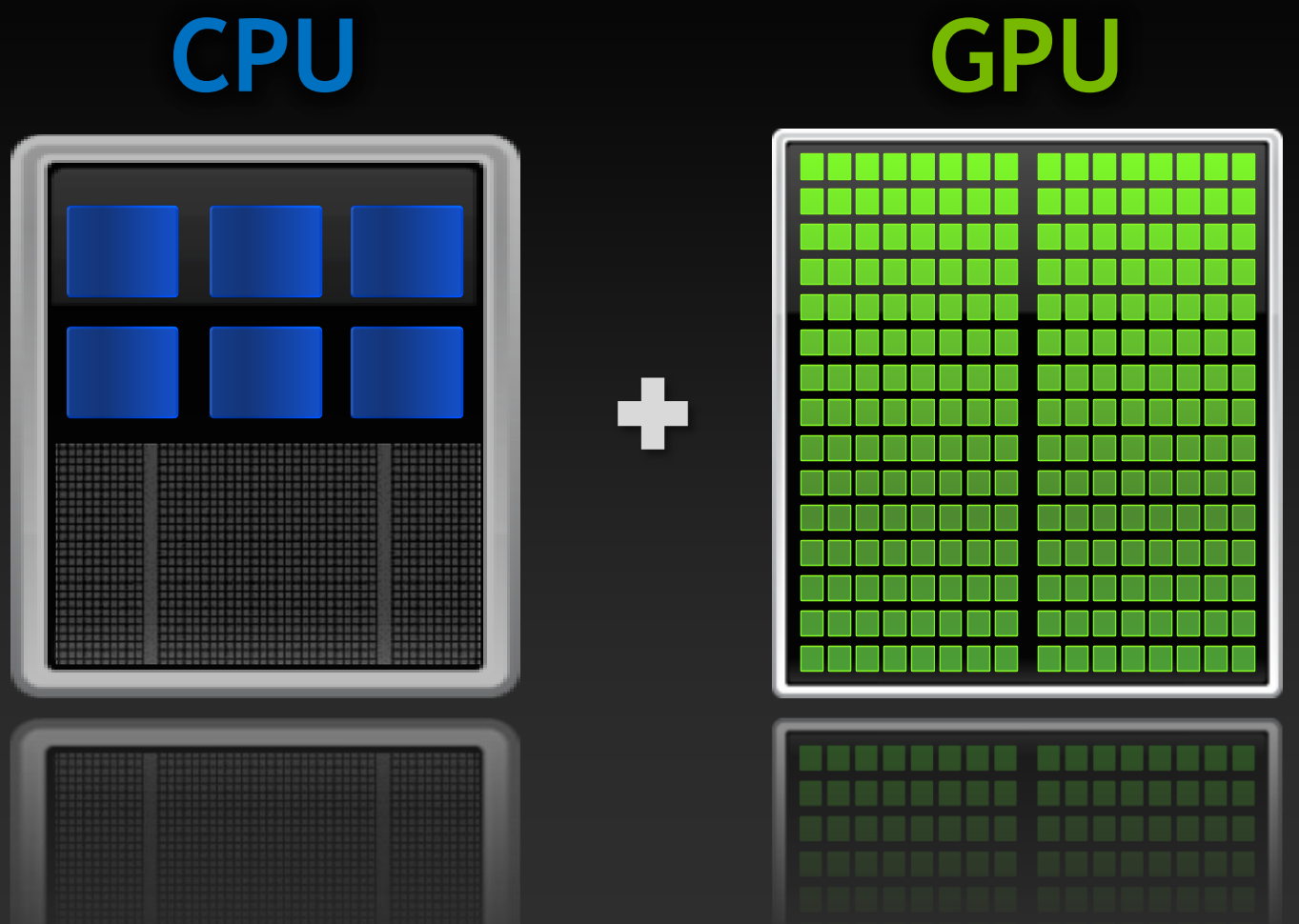

MIPS/CPU clock speed
10^3
10^2
10^1
10^0
1980
1985
1990
1995
2000
2005
2010
Year

What makes up the brick wall?

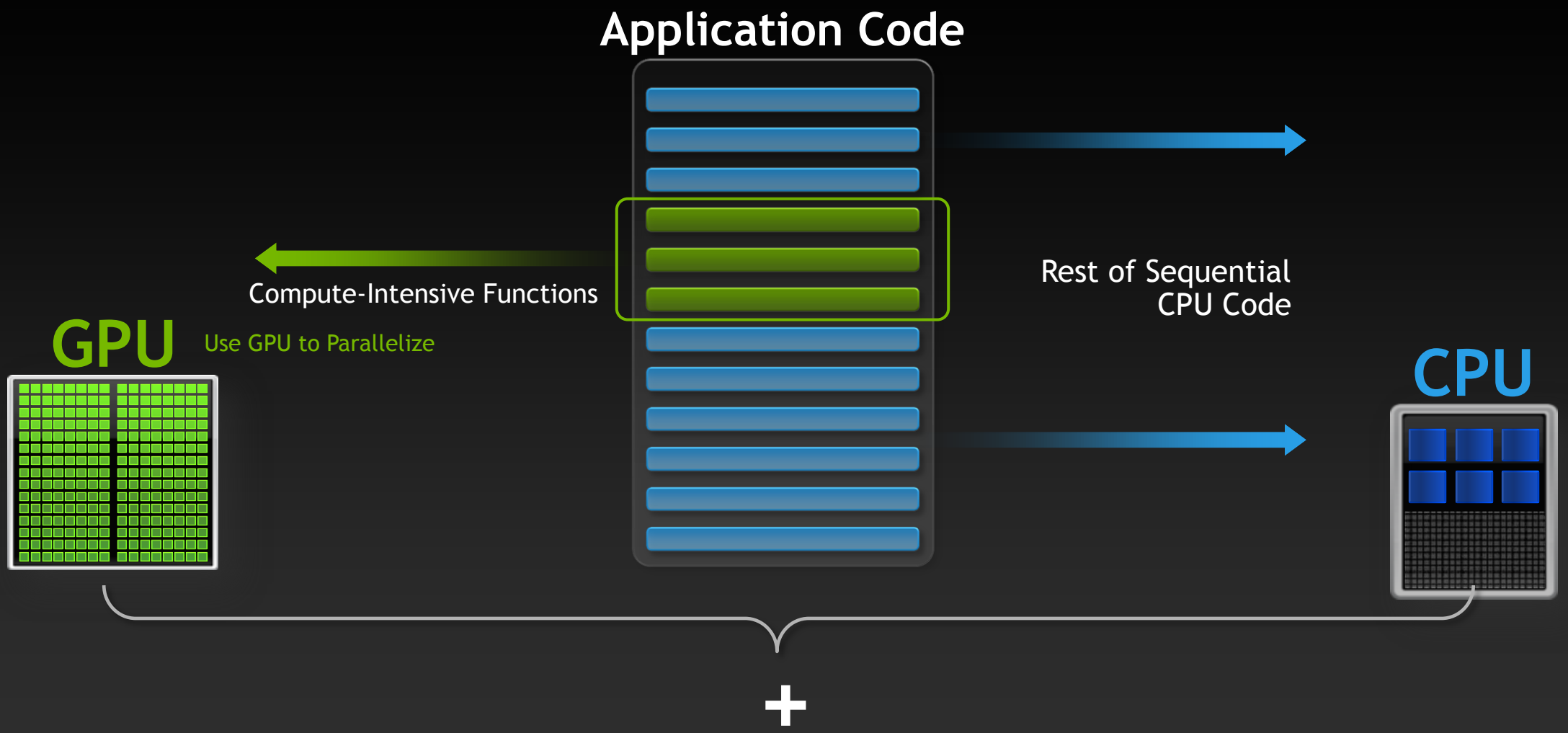
Power Wall
Memory Wall
\+ ILP Wall
= Brick Wall



Heterogeneous Computing

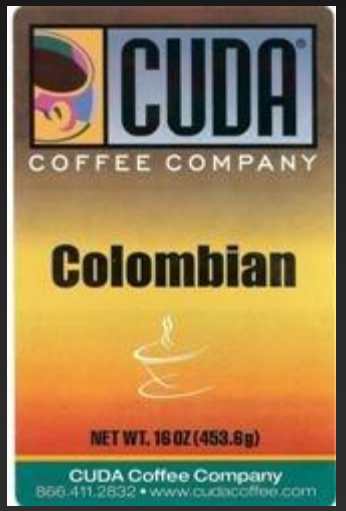


The basic idea



What is CUDA?

- Programming language?
- Compiler?
- Classic car?
- Beer?
- Wine?
- Coffee?



CUDA Parallel Computing Platform

www.nvidia.com/getcuda

Programming Approaches

- **Libraries** — "Drop-in" Acceleration
- **OpenACC Directives** — Easily Accelerate Apps
- **Programming Languages** — Maximum Flexibility

Development Environment

Nsight IDE
Linux, Mac and Windows
GPU Debugging and Profiling

CUDA-GDB debugger
NVIDIA Visual Profiler

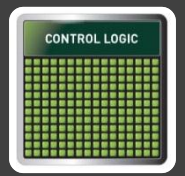
Open Compiler Tool Chain

LLVM COMPILER INFRASTRUCTURE

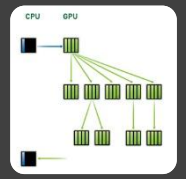
Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

Hardware Capabilities

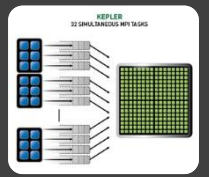
SMX



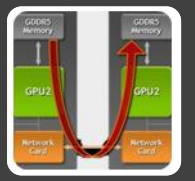
Dynamic Parallelism



HyperQ

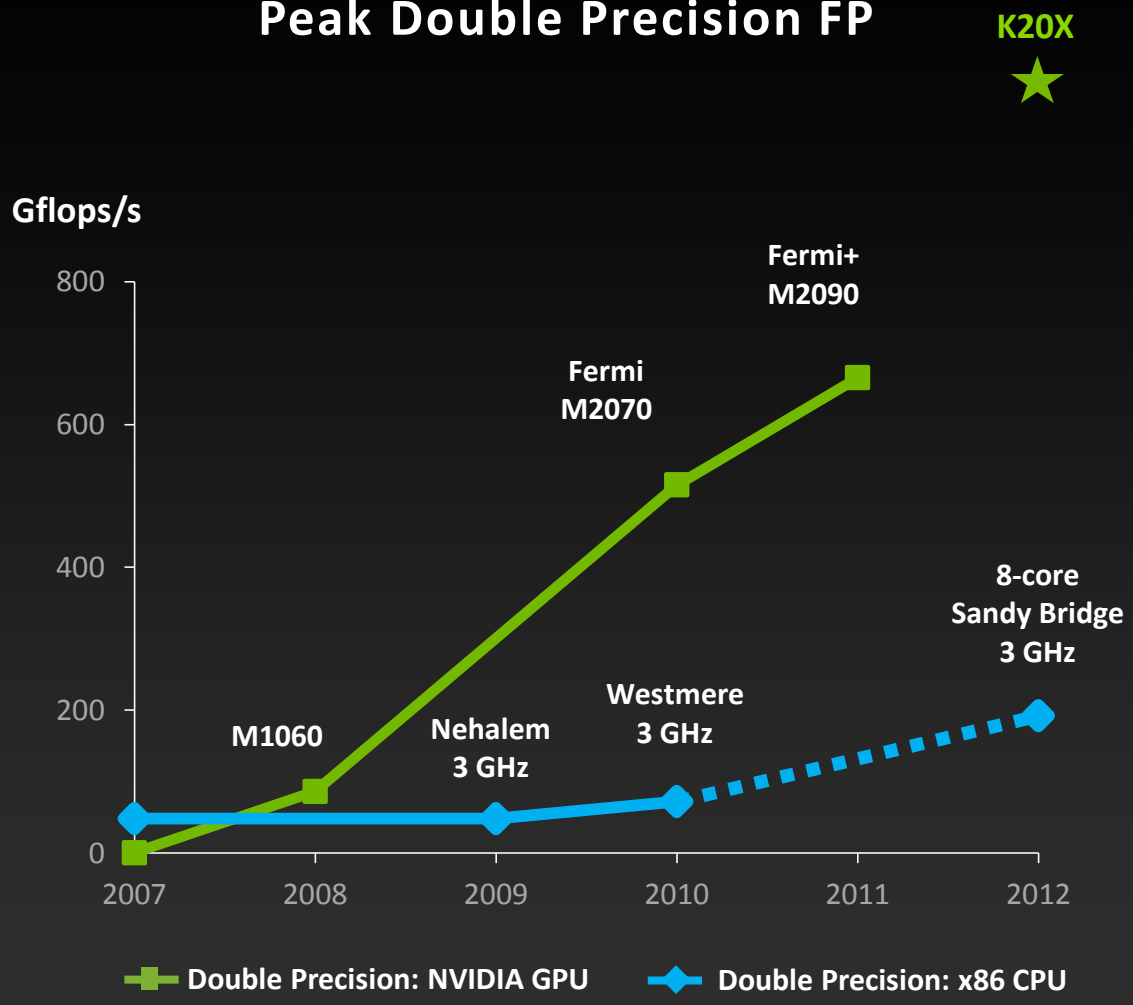


GPUDirect

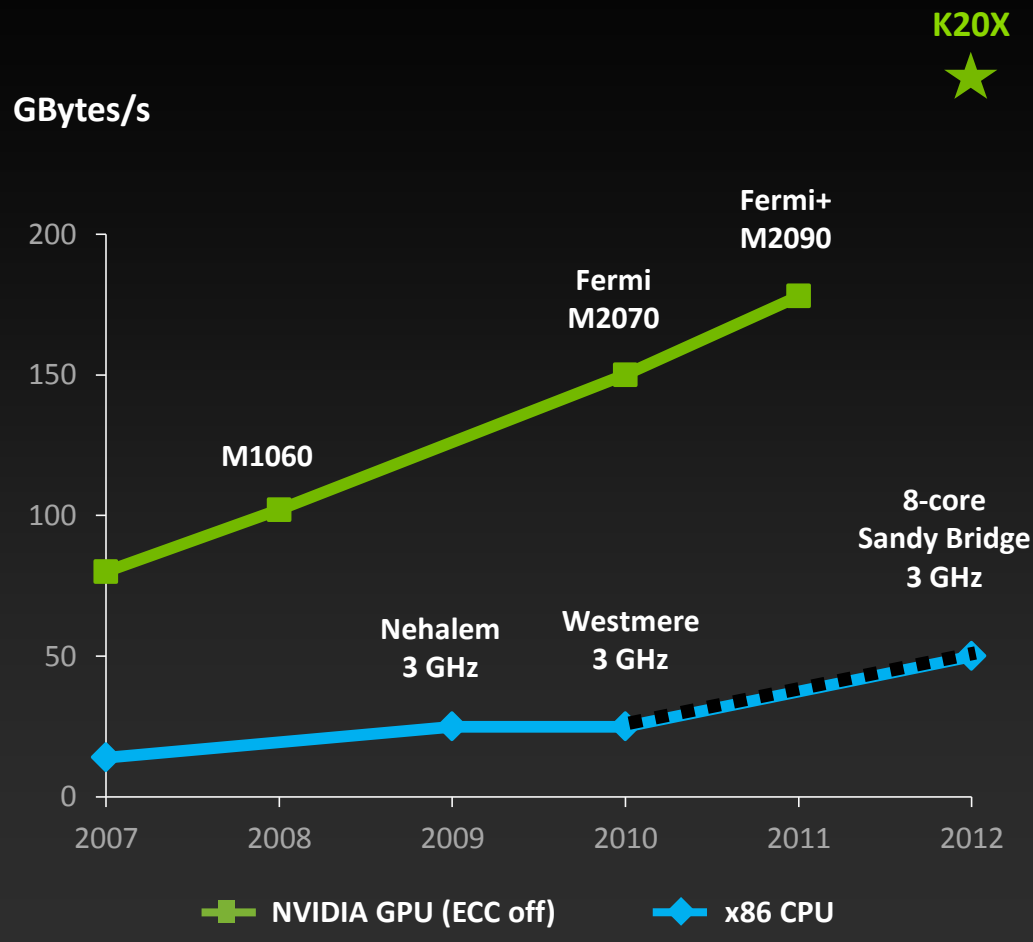


Why CUDA?

Peak Double Precision FP

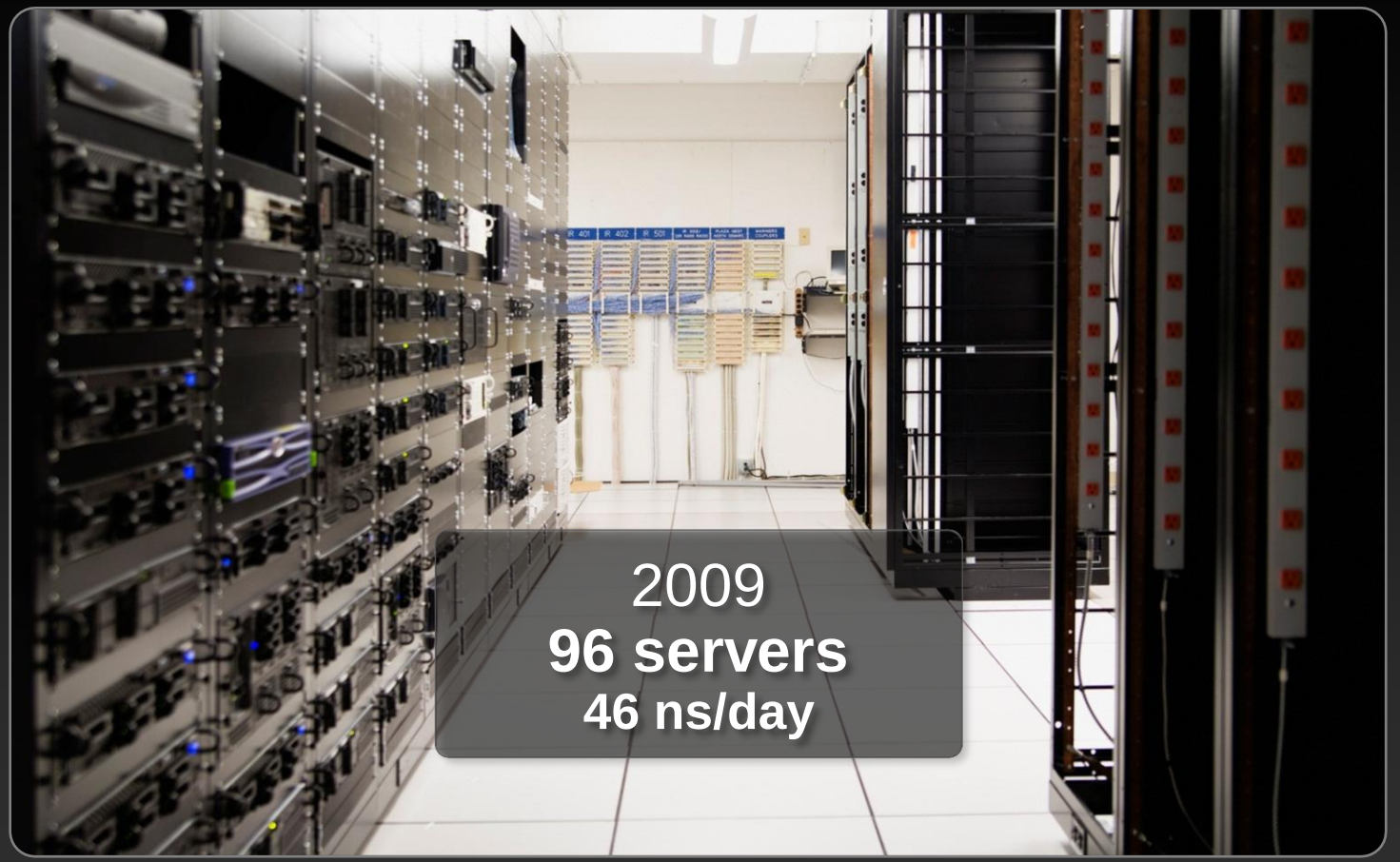


Peak Memory Bandwidth



Kepler: Unprecedented Value to Science

AMBER-JAC NVE Benchmark



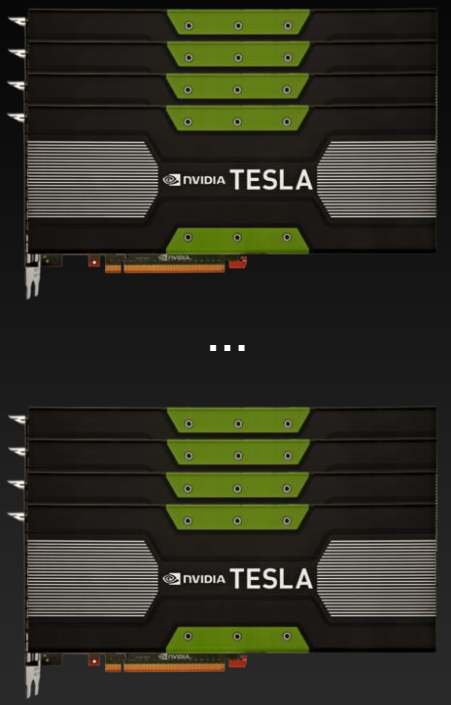
2009
96 servers
46 ns/day

2012
1 Tesla K20 GPU
81 ns/day

JANUS vs. GPUs

JANUS: 184 Peak TFLOPs
(2010)

≈



141 K20x GPUs
(2012)

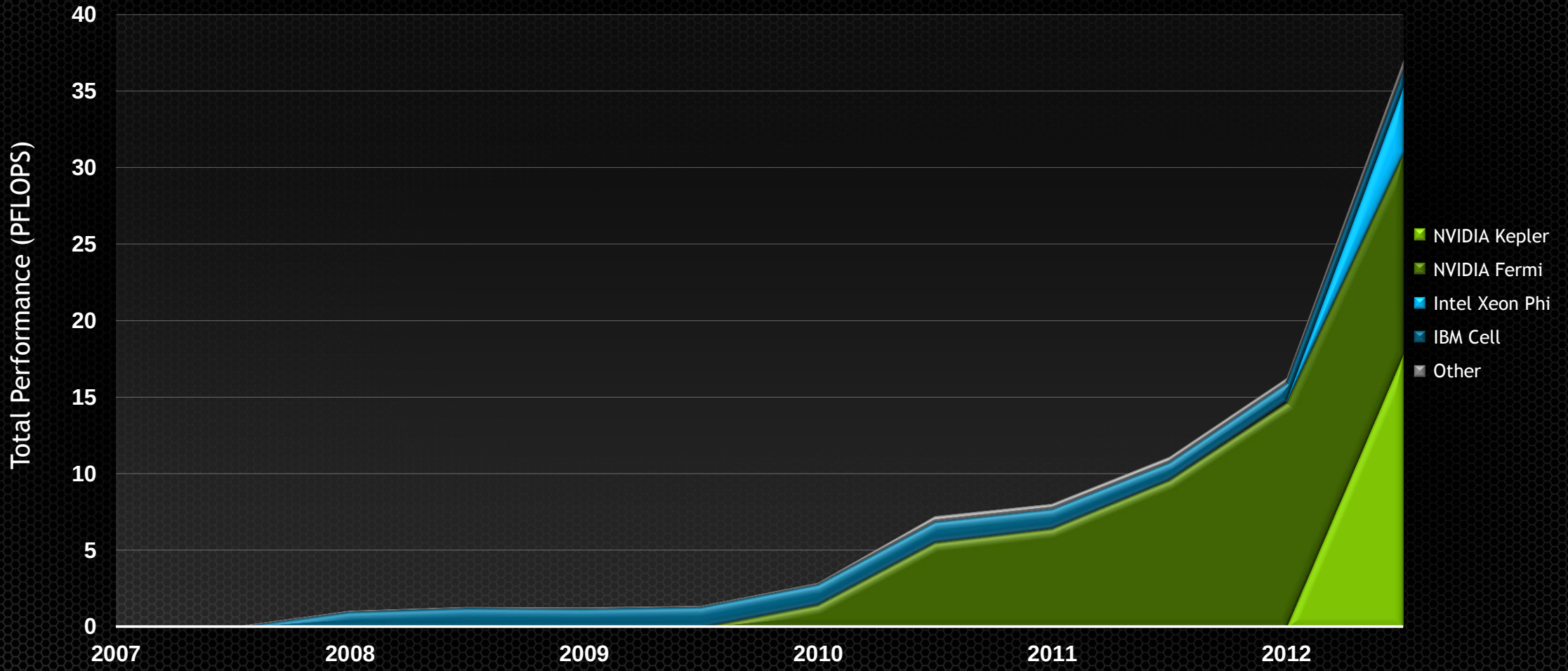
≈

2 Server Racks

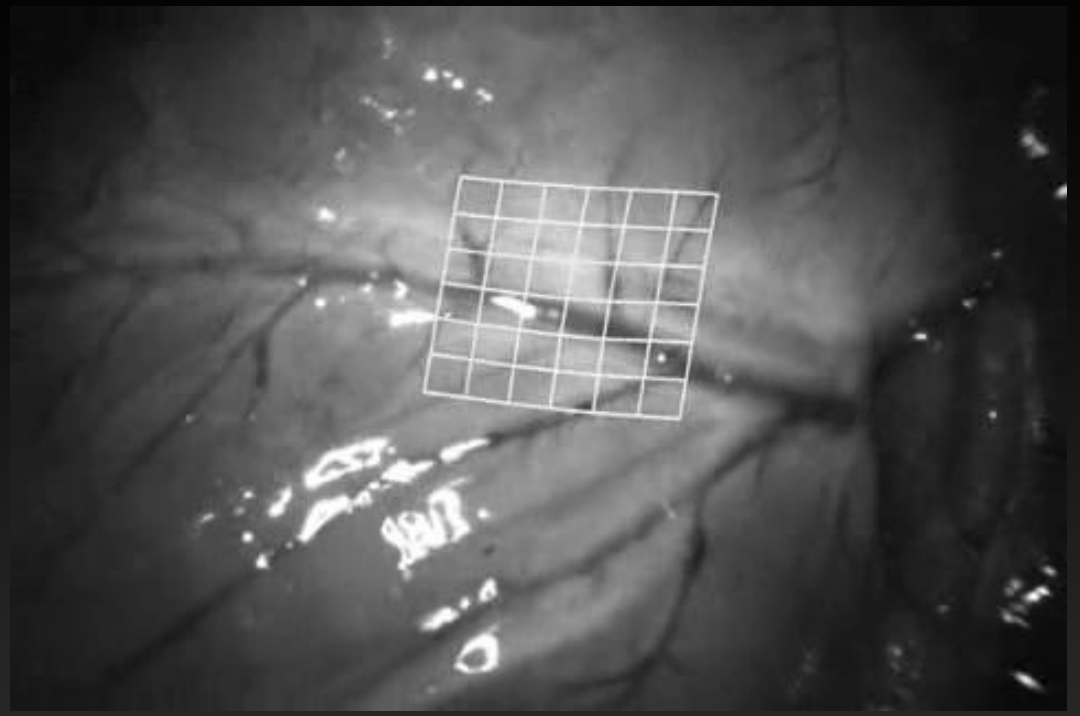
CUDA Accelerating

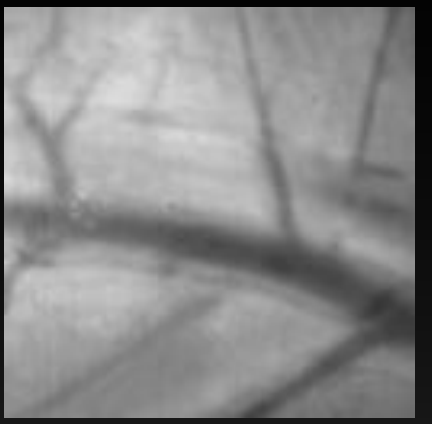
19% of FLOPS from GPU Systems

TOP500®



Operating on a Beating Heart





Only 2% of surgeons will operate on a beating heart

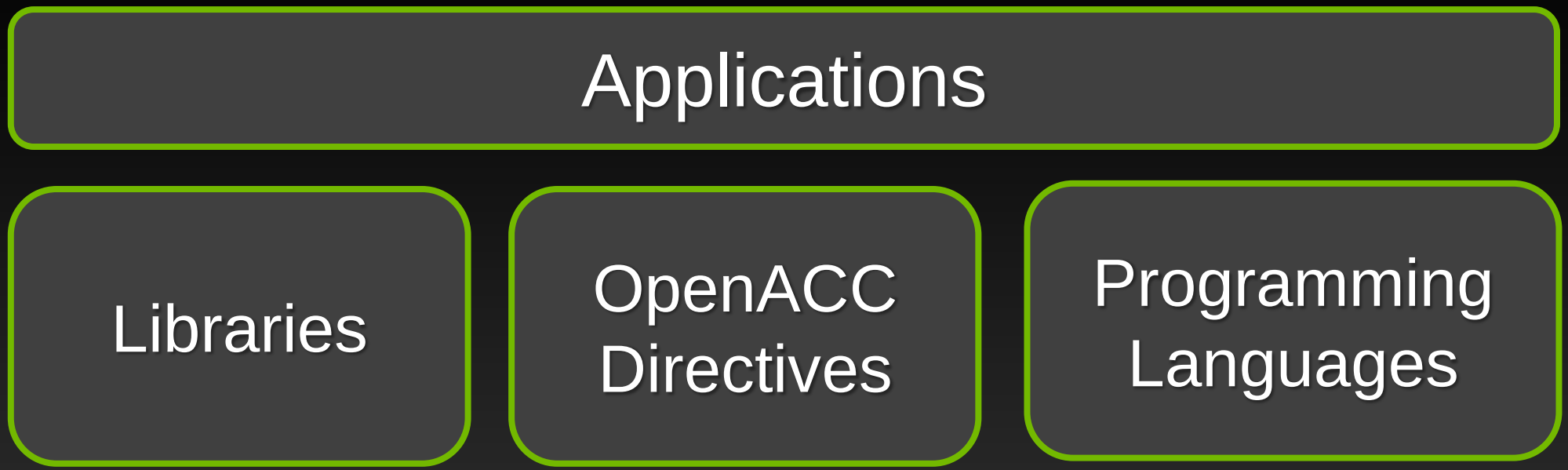
Patient stands to lose 1 point of IQ every10 min with heart stopped

GPU enables real-time motion compensation to virtually stop beating heart for surgeons

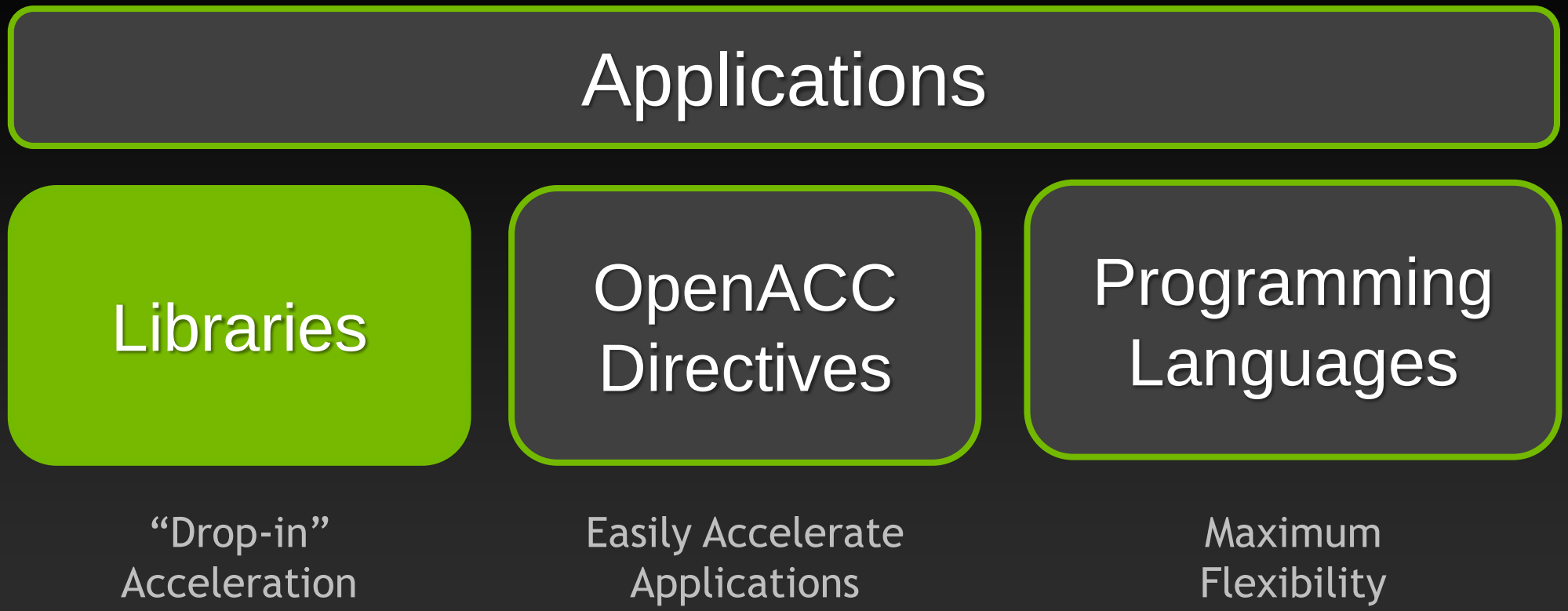
Courtesy Laboratoire d'Informatique de Robotique et de Microelectronique de Montpellier

Getting Started with CUDA

3 Ways to Accelerate Applications



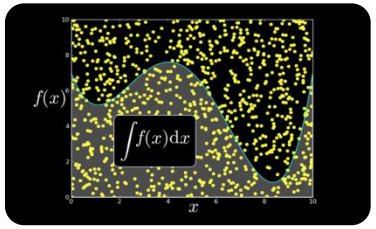
3 Ways to Accelerate Applications



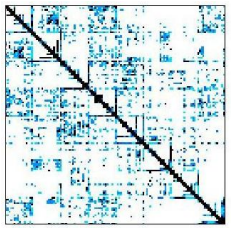
GPU Accelerated Libraries

"Drop-in" Acceleration for your Applications

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

CULA tools

GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore

open source initiative

NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

open source initiative

Thrust

C++ STL Features for CUDA

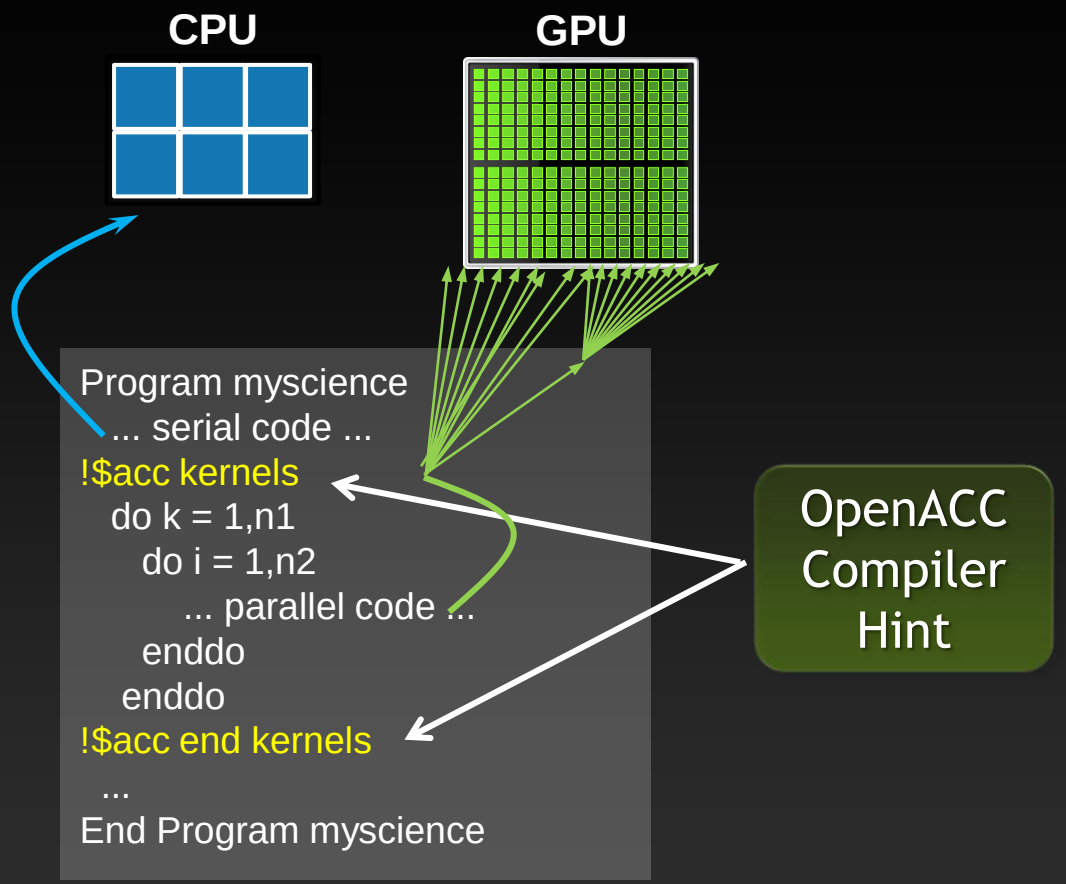
open source initiative

3 Ways to Accelerate Applications

Applications

Libraries	OpenACC Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility

OpenACC Directives

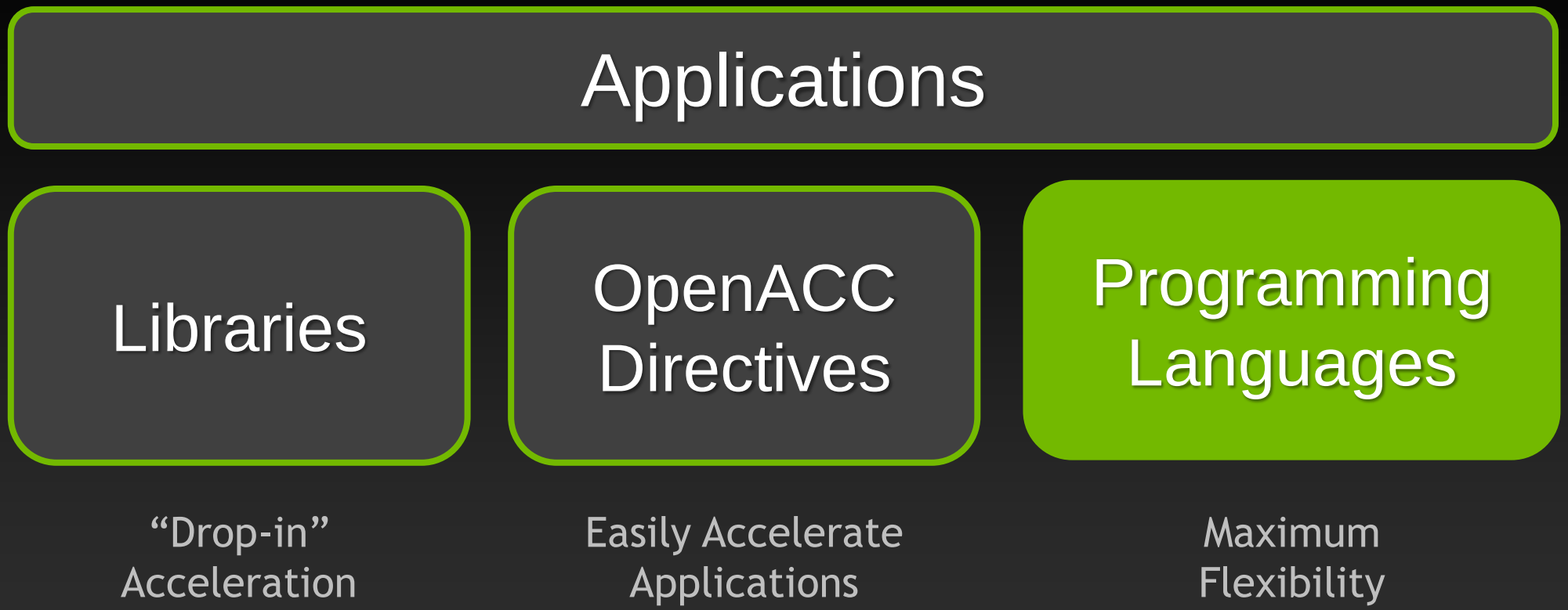


Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

3 Ways to Accelerate Applications



GPU Programming Languages

Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	OpenACC, CUDA Fortran
C	OpenACC, CUDA C
C++	Thrust, CUDA C++
Python	PyCUDA, Copperhead, CUDA Python
F#	Alea.cubase

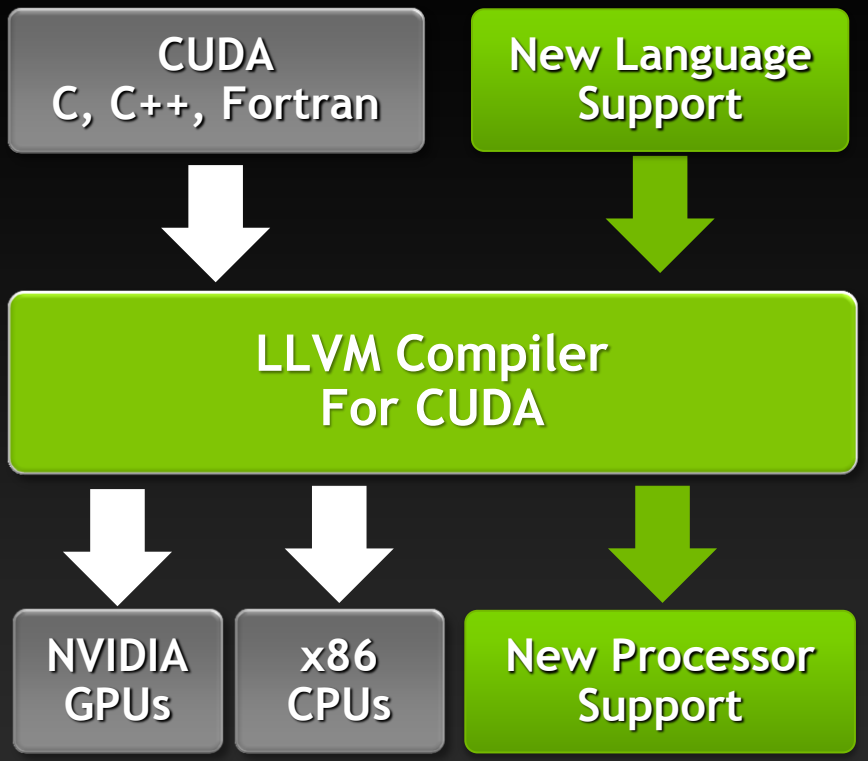
Enabling More Programming Languages

Developers want to build front-ends for

Python, Java, R, DSLs …

Target other processors like

ARM, FPGAs, GPUs, x86 …



LLVM COMPILER INFRASTRUCTURE

Enabling More Programming Languages

CONTINUUM ANALYTICS

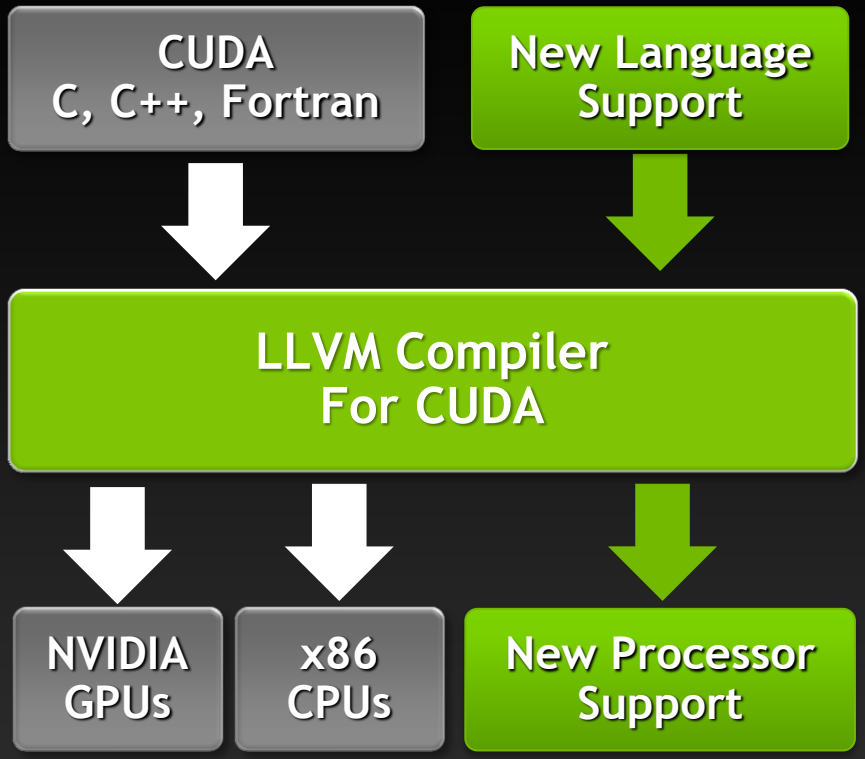
QuantAlea

double negative visual effects

PERVASIVE PARALLELISM LABORATORY PPL

Mozilla Rust

Halide (http://halide-lang.org/)



LLVM COMPILER INFRASTRUCTURE

python™

NVIDIA

Rapid Development

Powerful Libraries

NumPy

SciPy.org

matplotlib

Large Community

PyCon
Connecting The Python Community

Commercial Support

CONTINUUM ANALYTICS

ENTHOUGHT
SCIENTIFIC COMPUTING SOLUTIONS

Is Python Fast Enough for HPC?

Python apps often implement
performance critical functions in C/C++.

PyCUDA

- Provides access to CUDA API from Python
  - Convenience classes for device data and code management
- Kernels must still be written in CUDA C
  - Can be dynamically generated and compiled on the fly by PyCUDA
- Open Source project

Copperhead

Standard Python

```
import numpy as np


def saxpy(a, x, y):
  return [a * xi + yi
          for xi, yi in zip(x, y)]

x = np.arange(2**20, dtype=np.float32)
y = np.arange(2**20, dtype=np.float32)




cpu_result = saxpy(2.0, x, y)
```

http://numpy.scipy.org

Copperhead: Parallel Python


```
from copperhead import *
import numpy as np

@cu
def saxpy(a, x, y):
  return [a * xi + yi
          for xi, yi in zip(x, y)]

x = np.arange(2**20, dtype=np.float32)
y = np.arange(2**20, dtype=np.float32)

with places.gpu0:
  gpu_result = saxpy(2.0, x, y)

with places.openmp:
  cpu_result = saxpy(2.0, x, y)
```

http://copperhead.github.com

Compile Python for Parallel Architectures

- Anaconda Accelerate from Continuum Analytics
  - NumbaPro array-oriented compiler for Python & NumPy
  - Compile for CPUs or GPUs (uses LLVM + NVIDIA Compiler SDK)
- Fast Development + Fast Execution: Ideal Combination

Free Academic License

http://continuum.io

NumbaPro Demos

- Mandelbrot iPython Notebook
  - https://github.com/harrism/numbapro_examples (download)
  - http://nbviewer.ipython.org/f5707335f40af9463c43 (view)
- Monte Carlo iPython Notebook
  - https://github.com/ContinuumIO/numbapro-examples/tree/master/monte_carlo_pricer (download)
  - http://nbviewer.ipython.org/835a8ca39ced77fe751d (view)

JANUS GPUs

GPU nodes

We currently provide two GPU-enabled nodes for visualization and GPU-accelerated applications. These nodes are accessed via the crc-gpu queue.

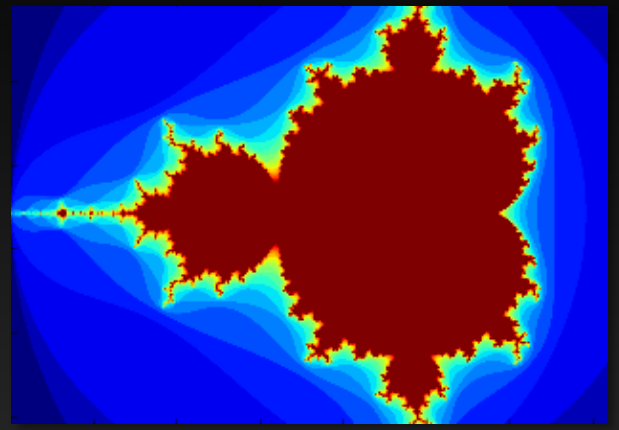
Their specifications are:

- 128 GB RAM
- 2 1-TB local disks
- 12 physical cores (24 with hyperthreading), Intel Xeon X5660 processors (2.8 GHz)
- 1 NVIDIA Tesla M2070 GPU

The crc-gpu queue has a maximum walltime of 4 hours.

CUDA Python

NumbaPro

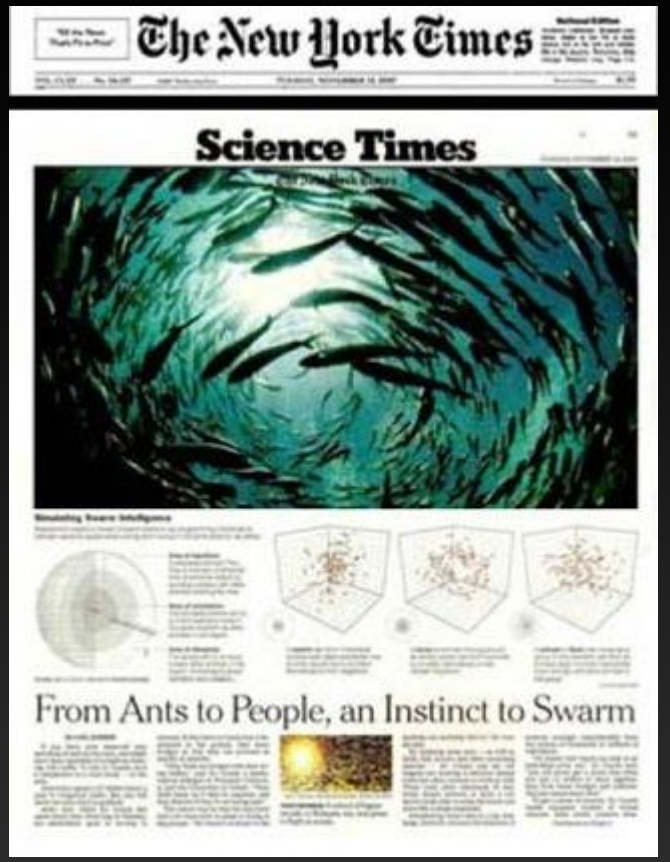


```
@cuda.jit(restype=uint32, argtypes=[f8, f8, uint32], device=True)
def mandel(x, y, max_iters):
  c = complex(x, y)
  z = 0.0j
  for i in range(max_iters):
    z = z*z + c
    if (z.real*z.real + z.imag*z.imag) >= 4:
      return i
  return max_iters

@cuda.jit(argtypes=[uint8[:,:], f8, f8, f8, f8, uint32])
def mandel_kernel(img, xmin, xmax ymin, ymax, iters):
  x, y = cuda.grid(2)
  if x < img.shape[0] and y < img.shape[1]:
    img[y, x] = mandel(min_x+x*((max_x-min_x)/img.shape[0]),
                       min_y+y*((max_y-min_y)/img.shape[1]), iters)

gimage = np.zeros((1024, 1024), dtype = np.uint8)
d_image = cuda.to_device(gimage)
mandel_kernel[(32,32), (32,32)](d_image, -2.0, 1.0, -1.0, 1.0, 20)
d_image.to_host()
```

CUDA Programming, Python Syntax

	1024^2 Mandelbrot Time	Speedup v. Pure Python
Pure Python	4.85s	--
NumbaPro (CPU)	0.11s	44x
CUDA Python (K20)	.004s	1221x

CUDA Breeds Academic Success



Facial Recognition with CUDA

- NICTA: National Information and Communication Technology Australia
  - Specialize in security research

With fricken laserbeams!

- Created by Intellectual Ventures to help fight malaria in third world countries
- Image detection and targeting is done with NVIDIA GPUs

Links to get started

- Get CUDA: www.nvidia.com/getcuda
- Anaconda Accelerate: store.continuum.io/cshop/accelerate/
- Programming Guide/Best Practices…
  - docs.nvidia.com
- Questions:
  - NVIDIA Developer forums devtalk.nvidia.com
  - Search or ask on www.stackoverflow.com/tags/cuda
- General: www.nvidia.com/cudazone

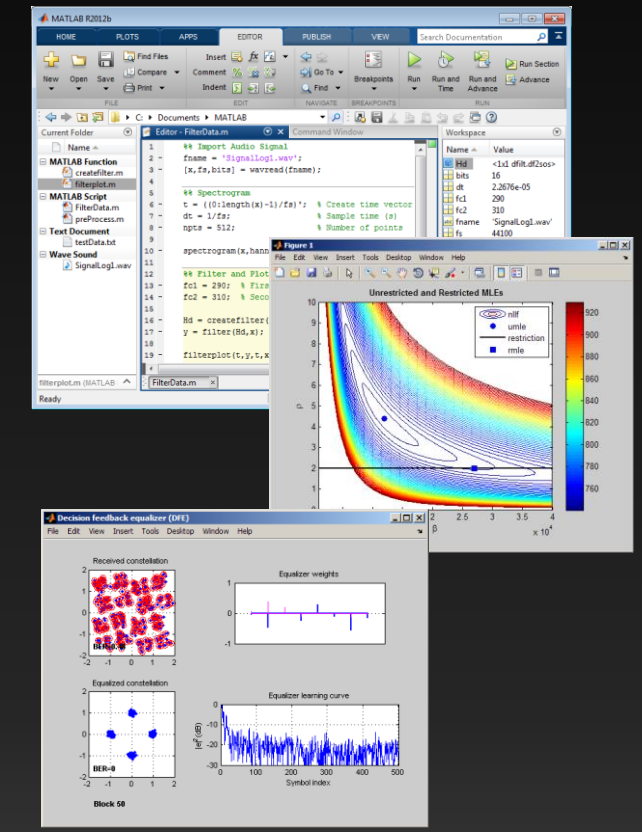
Parallel Computing with MATLAB

On-ramp to GPU Computing

MathWorks®

MATLAB

- Over 200 of the most popular MATLAB functions on GPUs
  Including:
  - Random number generation
  - FFT
  - Matrix multiplications
  - Solvers
  - Convolutions
  - Min/max
  - SVD
  - Cholesky and LU factorization
- MATLAB Compiler support (GPU acceleration in stand-alone applications)
- GPU features in
  - Communication Systems Toolbox
  - Phased Array System Toolbox
  - Signal Processing Toolbox
  - Neural Network Toolbox



Udacity Parallel Programming Course

UDACITY

Learn. Think. Do.

Invent your future through free interactive college classes.

20,000+ Students Already Taking the course!

IT'S FREE!

Udacity.com