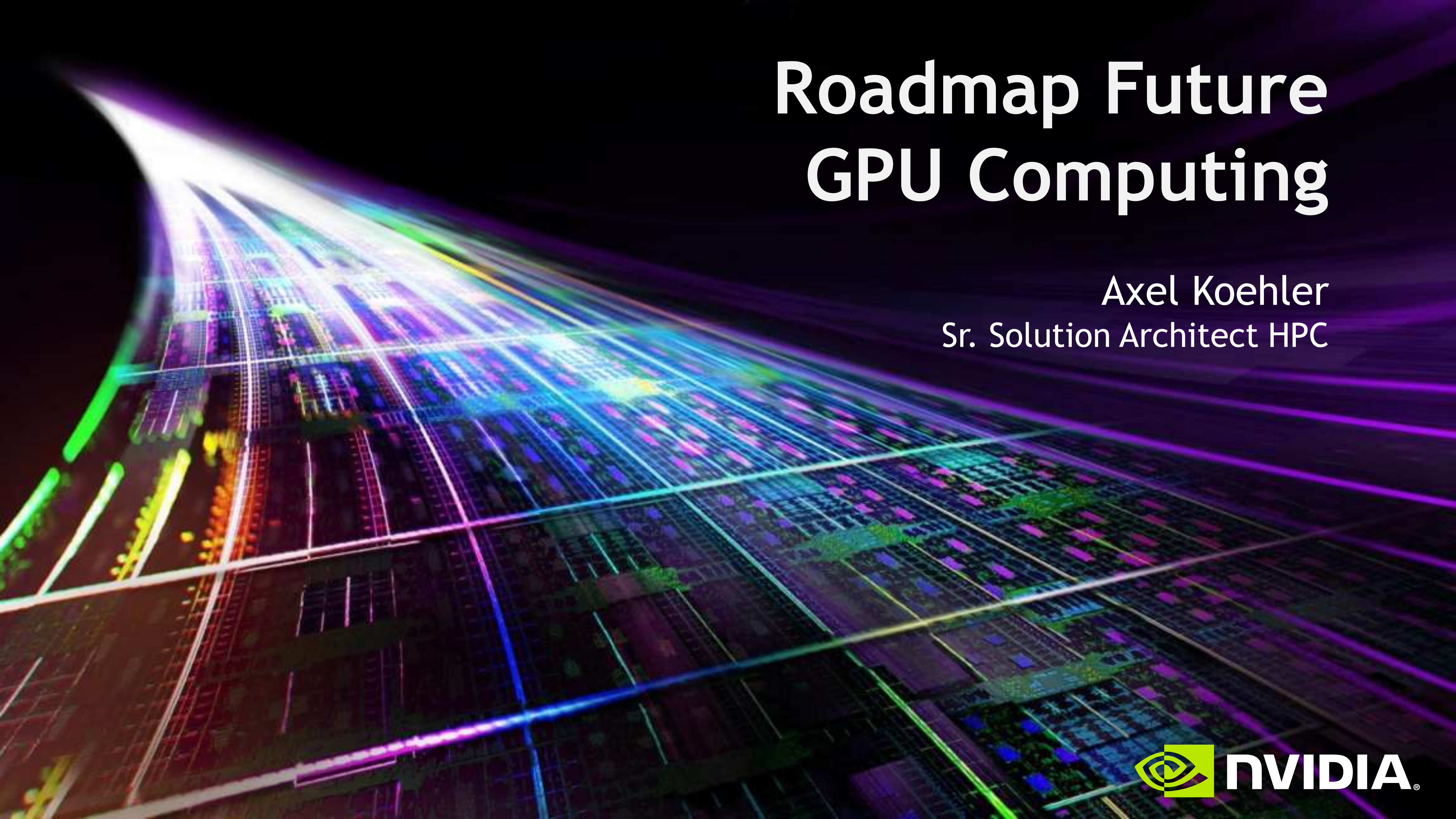
Roadmap Future GPU Computing



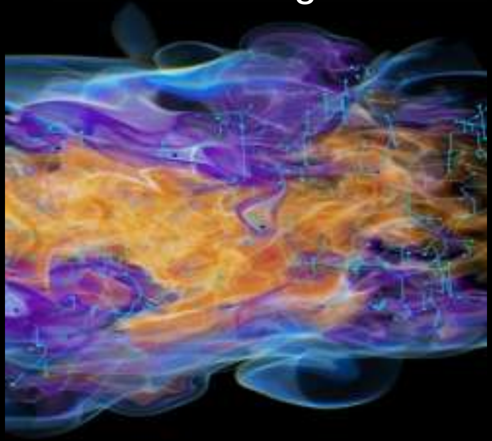
Axel Koehler

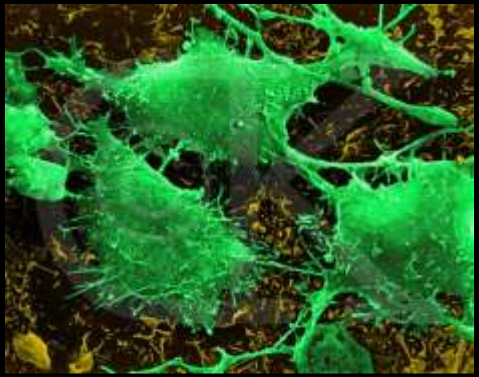
Sr. Solution Architect HPC

Continued Demand for Compute Power

Comprehensive Earth System Model at 1KM scale, enabling modeling of cloud convection and ocean eddies.

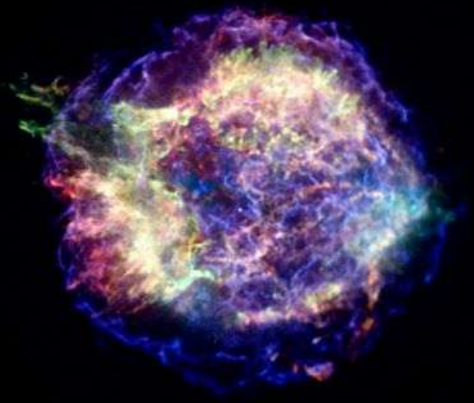
First-principles simulation of combustion for new high-efficiency, low-emision engines.



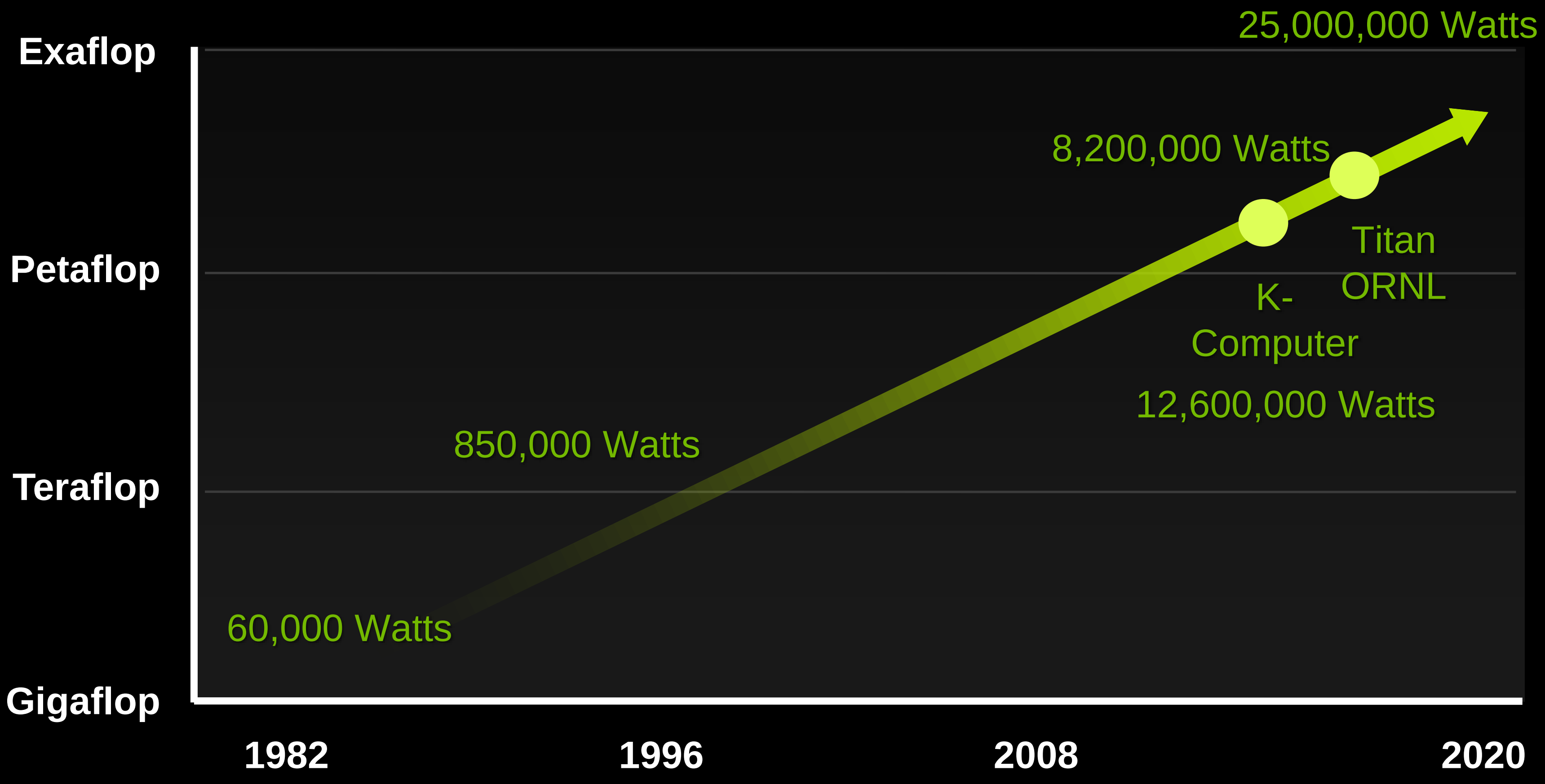


Coupled simulation of entire cells at molecular, genetic, chemical and biological levels.

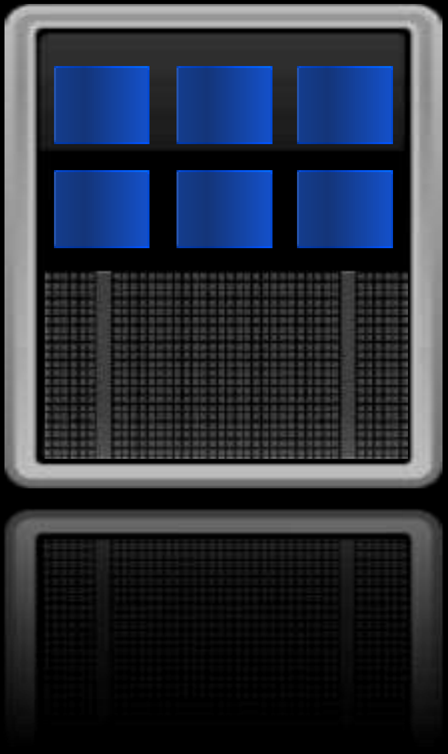
Predictive calculations for thermonuclear and core-collapse supernovae, allowing confirmation of theoretical models.



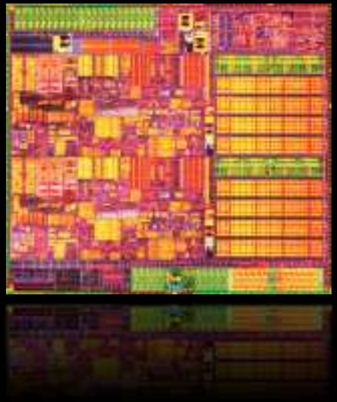
Power Crisis in (Super)computing



Multi-core CPUs



- Industry has gone multi-core as a first response to power issues
  - Performance through parallelism, not frequency
- But CPUs are fundamentally designed for single thread performance rather than energy efficiency
  - Fast clock rates with deep pipelines
  - Data and instruction caches optimized for latency
  - Superscalar issue with out-of-order execution
  - Lots of predictions and speculative execution
  - Lots of instruction overhead per operation

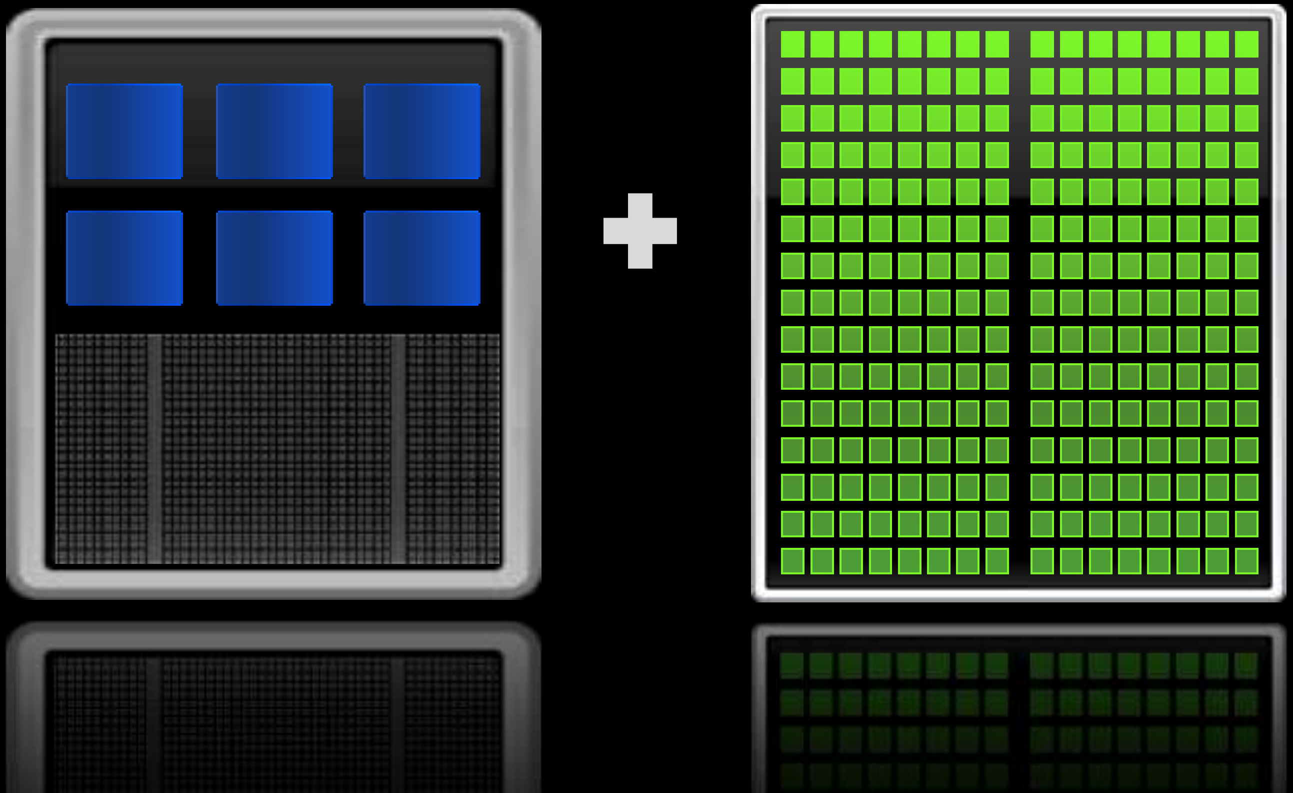


Less than 2% of chip power today goes to flops.

Accelerated Computing

Add GPUs: Accelerate Applications

CPUs: designed to run a few tasks quickly.

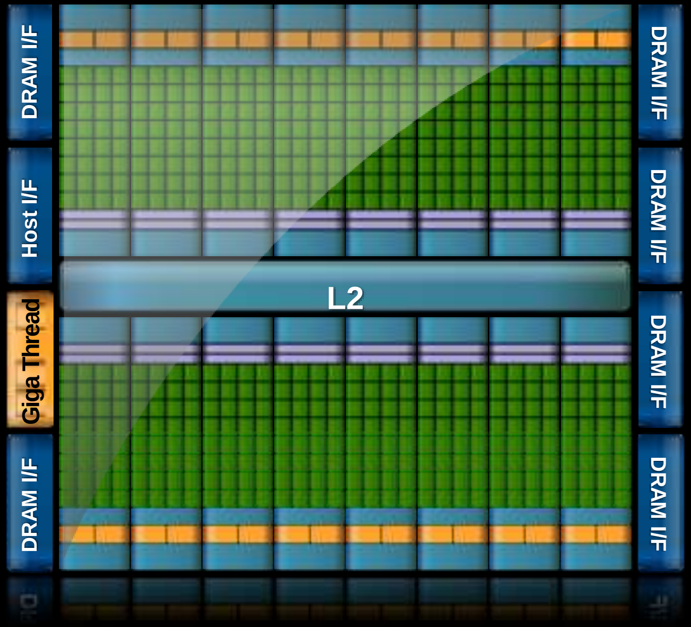


GPUs: designed to run many tasks *efficiently*.

Energy efficient GPU

Performance = Throughput

- Fixed function hardware
  - Transistors are primarily devoted to data processing
  - Less leaky cache
- SIMT thread execution
  - Groups of threads formed into warps which always executing same instruction
  - Some threads become inactive when code path diverges
- Cooperative sharing of units with SIMT
  - eg. fetch instruction on behalf of several threads or read memory location and broadcast to several registers
- Lack of speculation reduces overhead
- Minimal Overhead
  - Hardware managed parallel thread execution and handling of divergence



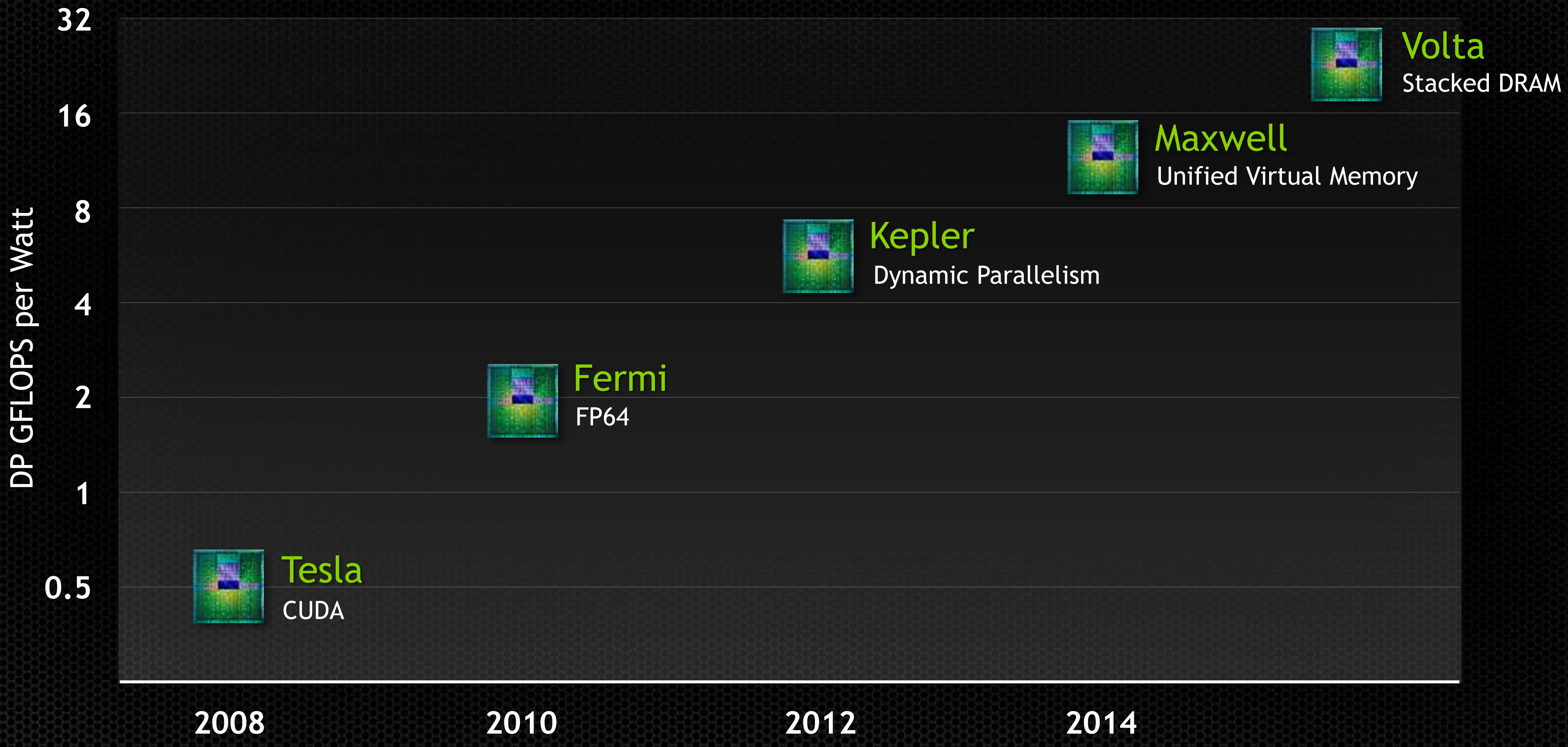
Overarching Goals for GPU Computing

Power
Efficiency

Ease of
Programming
And Portability

Application
Space
Coverage

GPU Roadmap


32
16
8
4
2
1
0.5
DP GFLOPS per Watt
Volta
Stacked DRAM
Maxwell
Unified Virtual Memory
Kepler
Dynamic Parallelism
Fermi
FP64
Tesla
CUDA
2008
2010
2012
2014

Kayla Development Platform

- Kepler-class GPU
  - SM35 → adds dynamic parallelism and other features
  - 2 SMX, 384 CUDA cores
  - Comes in MXM and PCIe form factor
  - Capability approaching Logan SoC (Integrated solution will be more power-efficient)
  - CUDA and OpenGL 4.3 support
- Carrier board: Seco mini-ITX GPU devkit
  - NVidia Tegra 3 CPU on Q7 module
  - NVidia PCIe GPU (eg. gf108, gk107, gk104, and Kayla GPU)
  - Carrier provides I/O connectors (eg. Gigabit, SATA, USB)

https://developer.nvidia.com/kayla-platform

CUDA on ARM roadmap

- Software
  - CUDA releases starting with CUDA 5.5 and 319.xy include ARM support
  - Native ARM compiler architecture (no longer x86 cross development needed)
  - cuda-gdb: native ARM and client-server
- Long term plans for CUDA on the ARM platform
  - Logan, Tegra with integrated Kepler class GPU
  - ARMv8 64-bit platform support, starting with Parker
  - Enable other partners and industry support

Which Takes More Energy?

Performing a 64-bit floating-point FMA:

893,500.288914668

× 43.90230564772498

= 39,226,722.78026233027699

+ 2.02789331400154

= 39,226,724.80815564

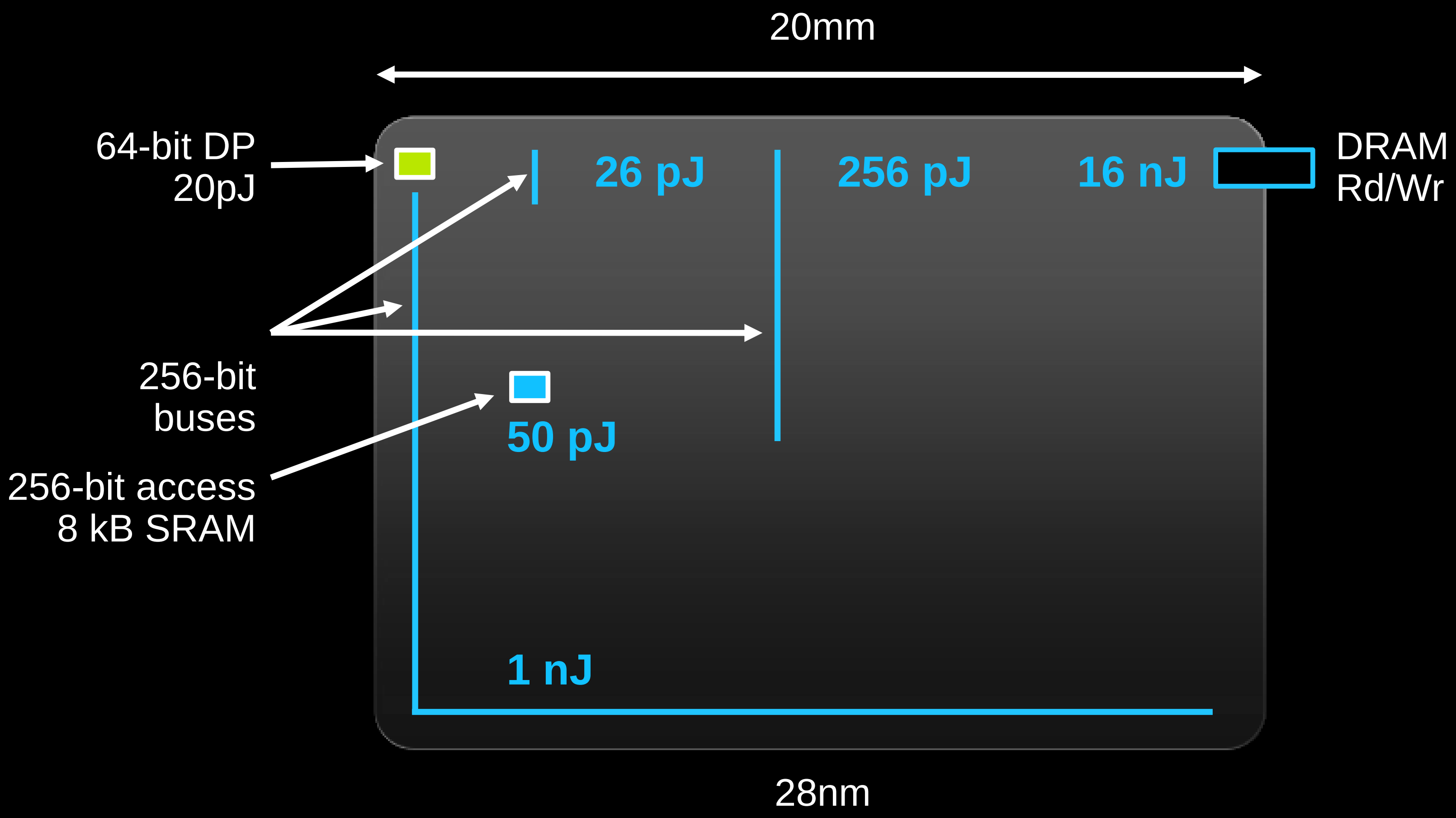
Or moving the three 64-bit operands 20 mm across the die:

This one takes over 4.7x the energy today (40nm)!

It’s getting worse: in10nm, relative cost will be 17x!

Loading the data from off chip takes >> 100x the energy.

Power is the problem

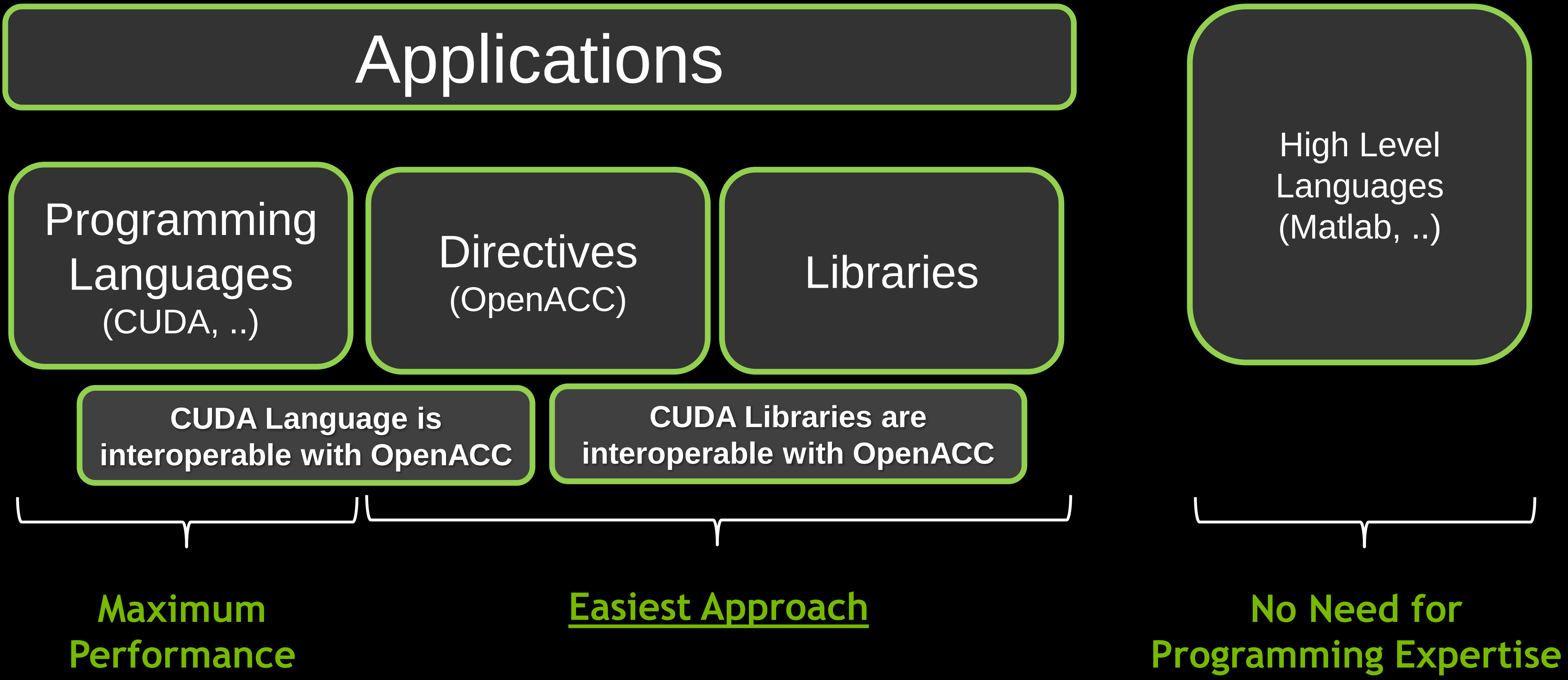


Fetching operands costs more than computing on them

What is important for the future?

- Its not about the FLOPS
- Its about data movements
- Algorithms should be designed to perform more work per unit data movement
- Programming systems should further optimize this data movement
- Architectures should facilitate this by providing an exposed hierarchy and efficient communication

Ways to Accelerate Applications



Applications

Programming Languages (CUDA, ..)

Directives (OpenACC)

Libraries

High Level Languages (Matlab, ..)

CUDA Language is interoperable with OpenACC

CUDA Libraries are interoperable with OpenACC

Maximum Performance

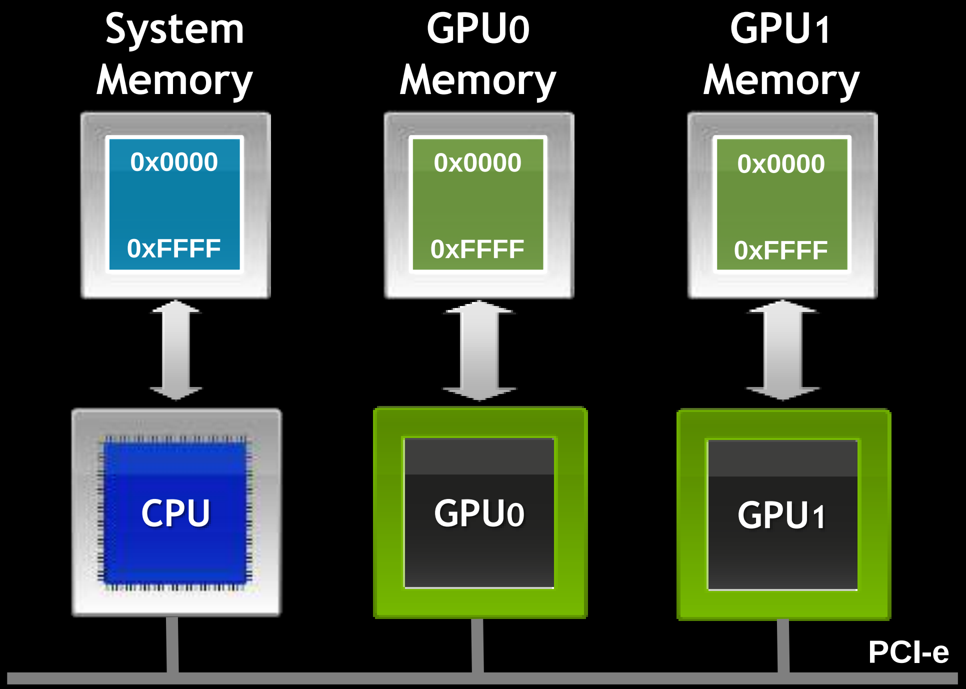
Easiest Approach

No Need for Programming Expertise

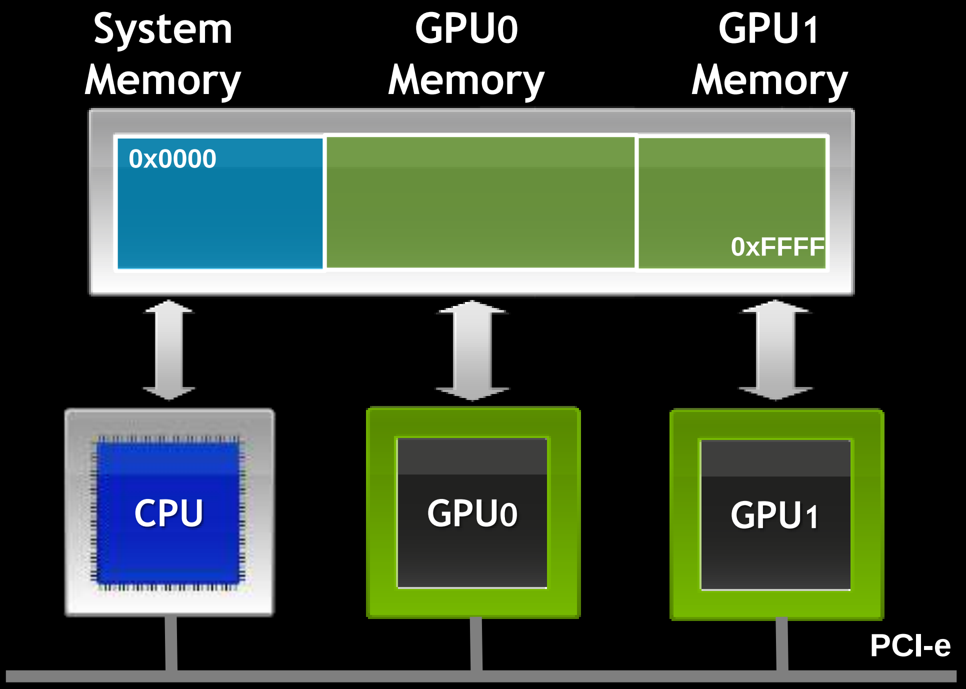
Unified Virtual Addressing

Easier to Program with Single Address Space

No UVA: Multiple Memory Spaces



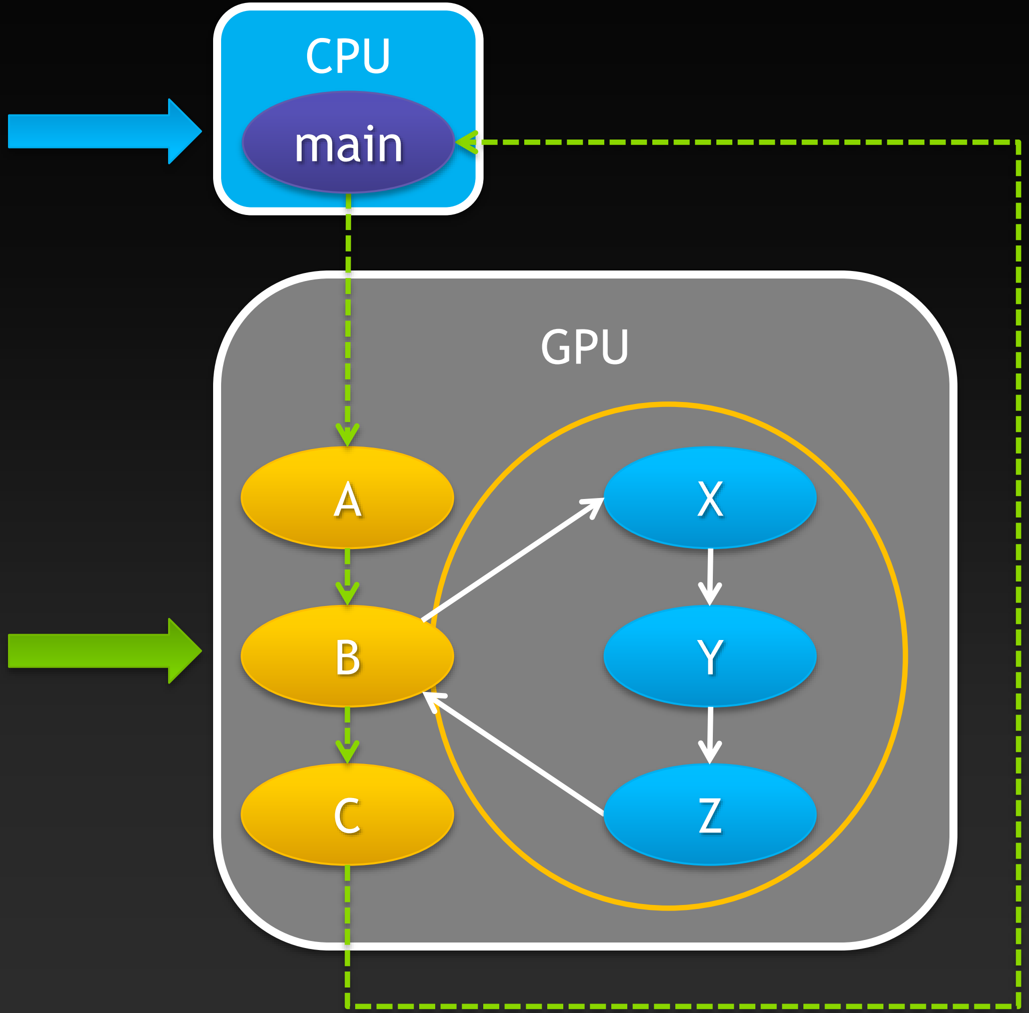
UVA : Single Address Space



Unified Runtime Interface

```
int main() {
    float *data;
    setup(data);

    A <<< ... >>> (data);
    B <<< ... >>> (data);
    C <<< ... >>> (data);

    cudaDeviceSynchronize();
    return 0;
}
```

```
__global__ void B(float *data)
{
    do_stuff(data);

    X <<< ... >>> (data);
    Y <<< ... >>> (data);
    Z <<< ... >>> (data);
    cudaDeviceSynchronize();

    do_more_stuff(data);
}
```



Dynamic Parallelism

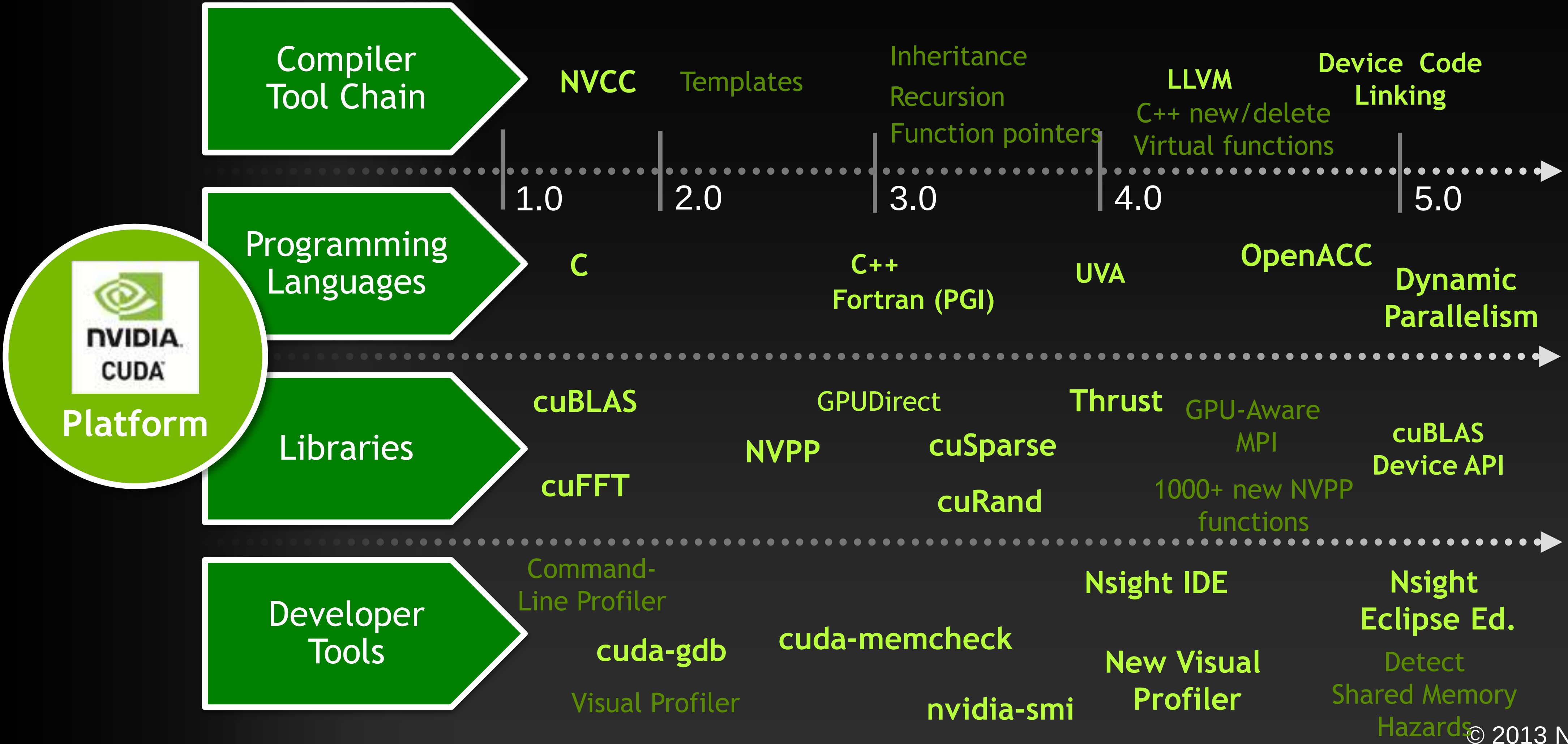
Unified Virtual Memory

```
void sortfile(FILE *fp, int N) {
  char *data = (char*)malloc(N);
  char *sorted = (char*)malloc(N);
  fread(data, 1, N, fp);

  char *d_data, *d_sorted;
  cudaMalloc(&d_data, N);
  cudaMalloc(&d_sorted, N);
  cudaMemcpy(d_data, data, N, ...);

  parallel_sort<<< ... >>>(  sorted,   data, N);

  cudaMemcpy(sorted, d_sorted, N, ...);
  cudaFree(d_data);
  cudaFree(d_sorted);

  use_data(sorted);
  free(data); free(sorted);
}
```

Unified Virtual Memory

```
void sortfile(FILE *fp, int N) {
  char *data = (char*)malloc(N);
  char *sorted = (char*)malloc(N);
  fread(data, 1, N, fp);

  ~~char *d_data, *d_sorted;~~
  ~~cudaMalloc(&d_data, N);~~
  ~~cudaMalloc(&d_sorted, N);~~
  ~~cudaMemcpy(d_data, data, N, ...);~~

  parallel_sort<<< ... >>>(  sorted,   data, N);

  ~~cudaMemcpy(sorted, d_sorted, N, ...);~~
  ~~cudaFree(d_data);~~
  ~~cudaFree(d_sorted);~~

  use_data(sorted);
  free(data); free(sorted);
}
```

Platform for Parallel Computing



NVIDIA CUDA Platform

Compiler Tool Chain

NVCC

Templates

Inheritance

Recursion

Function pointers

LLVM

C++ new/delete

Virtual functions

Device Code Linking

1.0

2.0

3.0

4.0

5.0

Programming Languages

C

C++

Fortran (PGI)

UVA

OpenACC

Dynamic Parallelism

Libraries

cuBLAS

cuFFT

NVPP

GPUDirect

cuSparse

cuRand

Thrust

GPU-Aware MPI

1000+ new NVPP functions

cuBLAS Device API

Developer Tools

Command-Line Profiler

cuda-gdb

Visual Profiler

cuda-memcheck

nvidia-smi

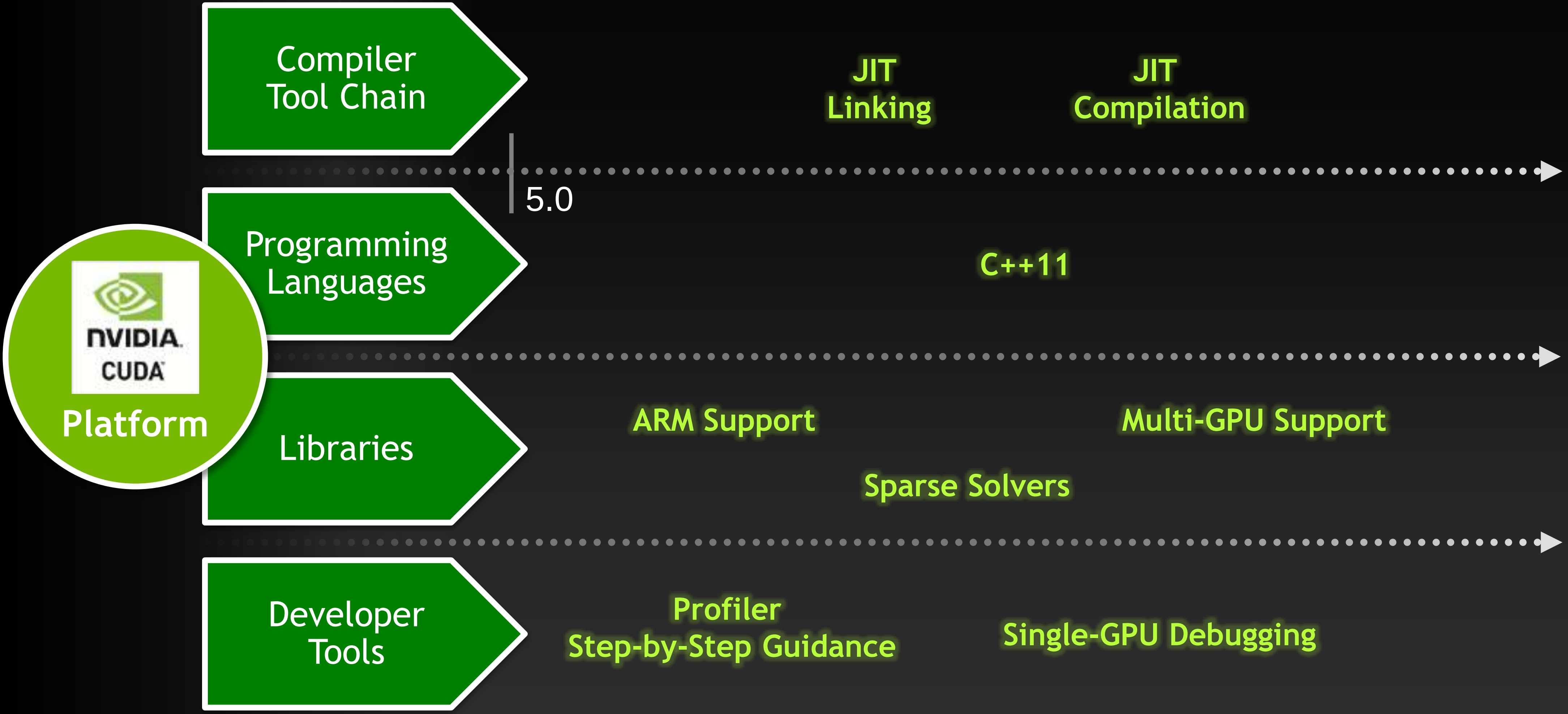
Nsight IDE

New Visual Profiler

Nsight Eclipse Ed.

Detect Shared Memory Hazards

Platform for Parallel Computing



NVIDIA CUDA

Platform

Compiler Tool Chain

5.0

JIT Linking

JIT Compilation

Programming Languages

C++11

Libraries

ARM Support

Multi-GPU Support

Sparse Solvers

Developer Tools

Profiler Step-by-Step Guidance

Single-GPU Debugging

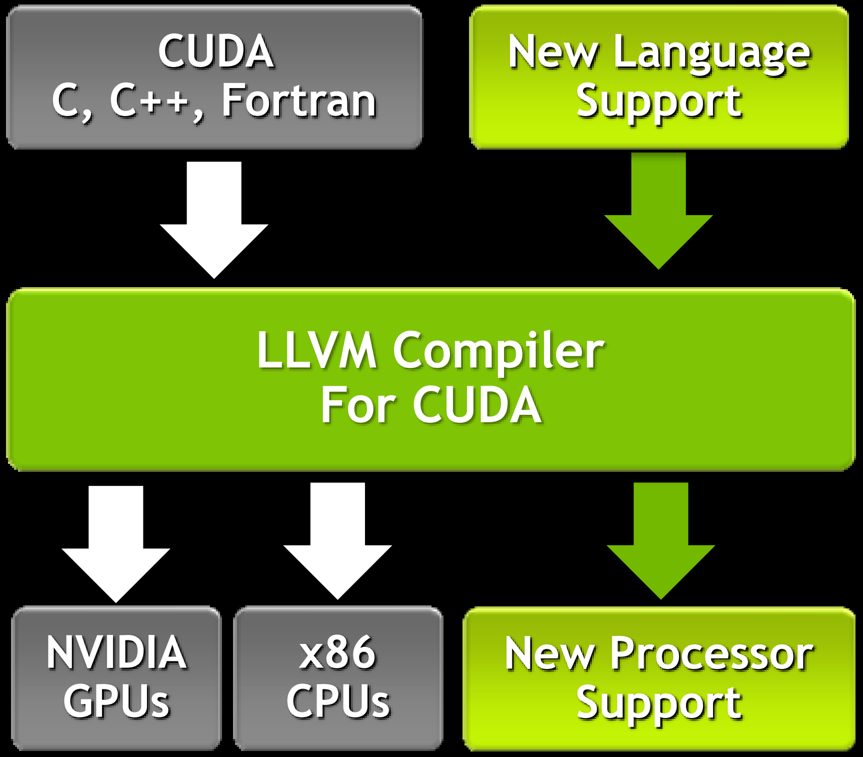
CUDA Compiler Contributed to Open Source LLVM

Developers want to build front-ends for

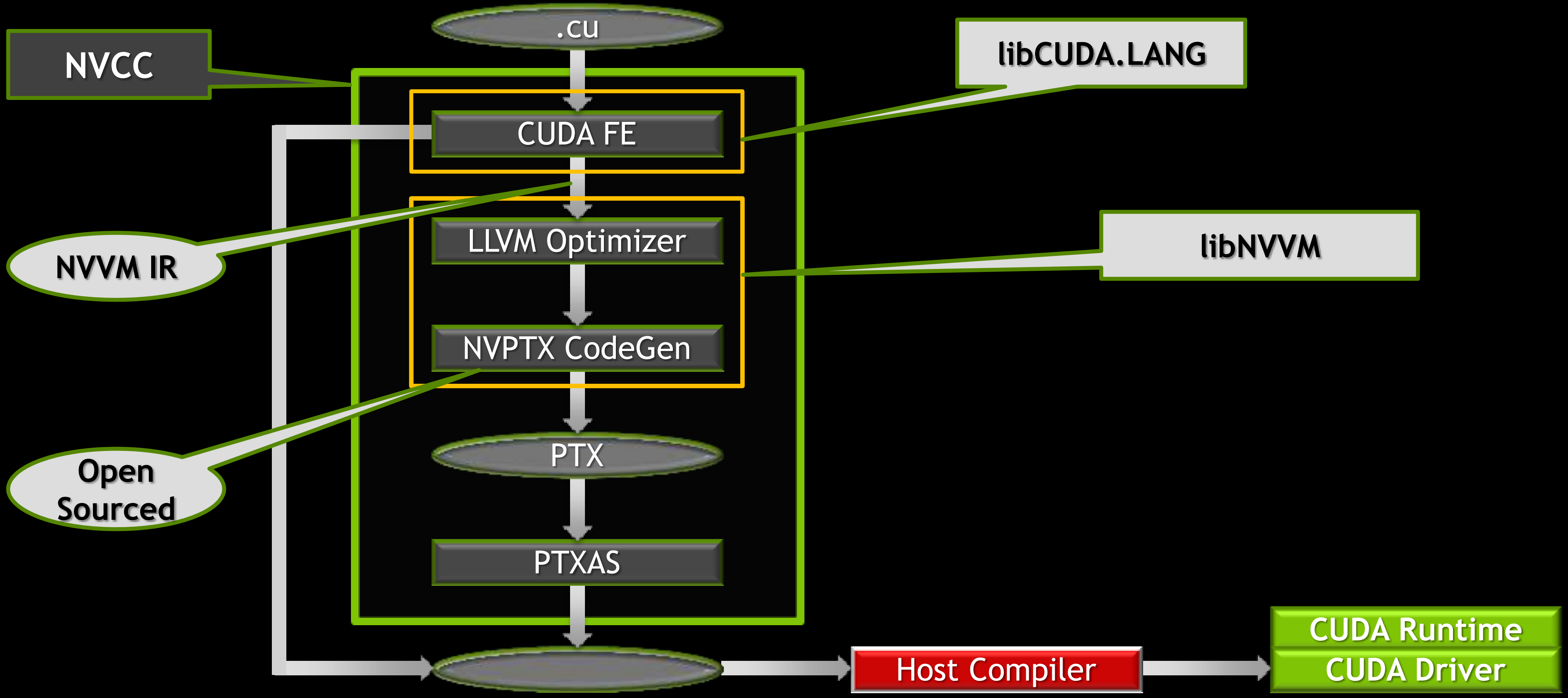
Java, Python, R, DSLs

Target other processors like

ARM, FPGA, GPUs, x86

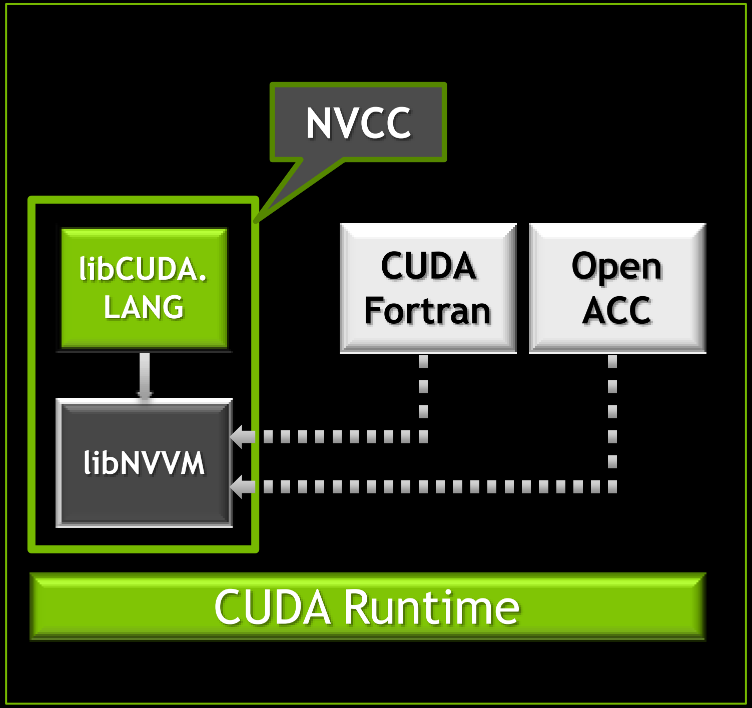


Open Compiler Architecture

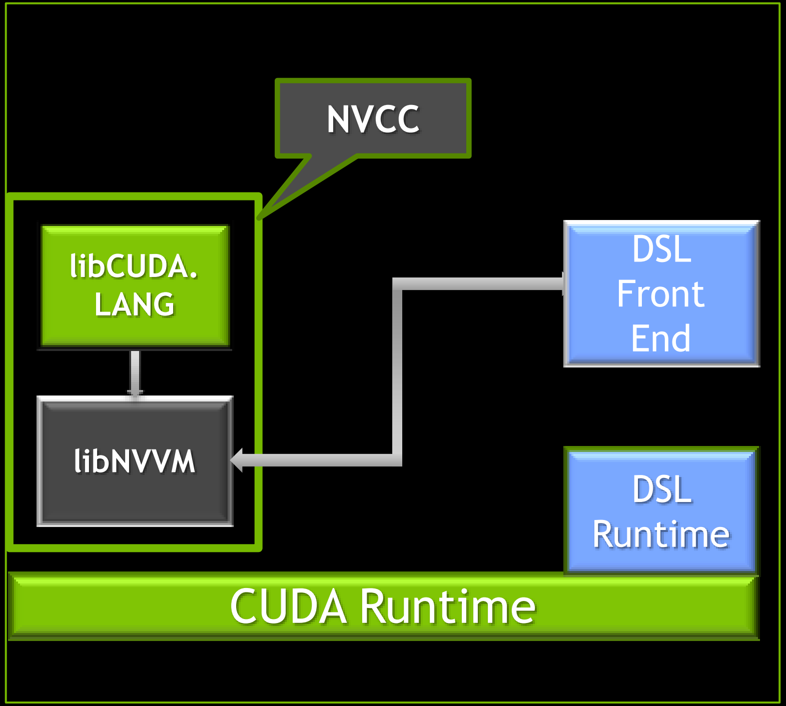


https://developer.nvidia.com/cuda-llvm-compiler

Scenarios for the Compiler SDK



Building Production Quality Compilers



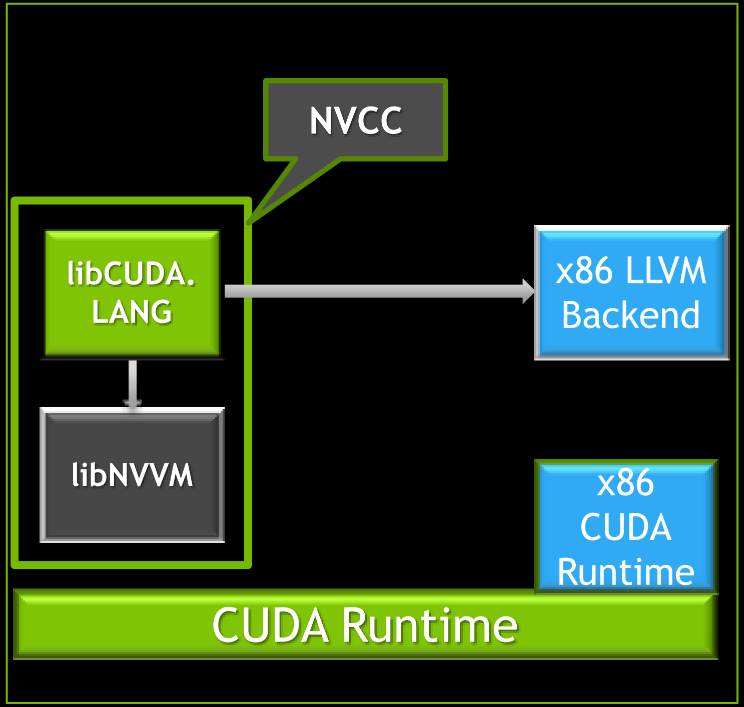
Building Domain Specific Languages (DSL)

MATLAB

R

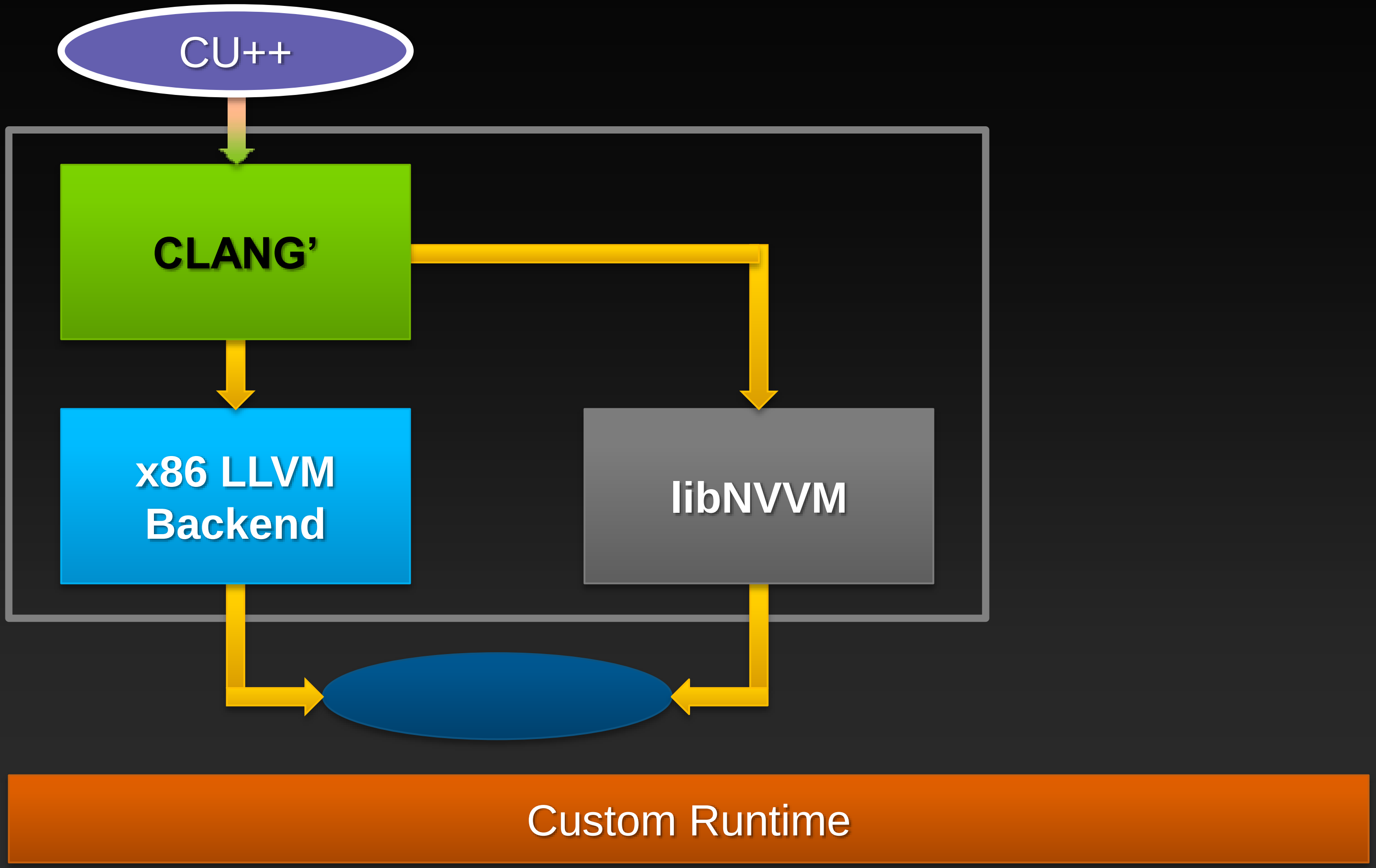
JET

HALIDE

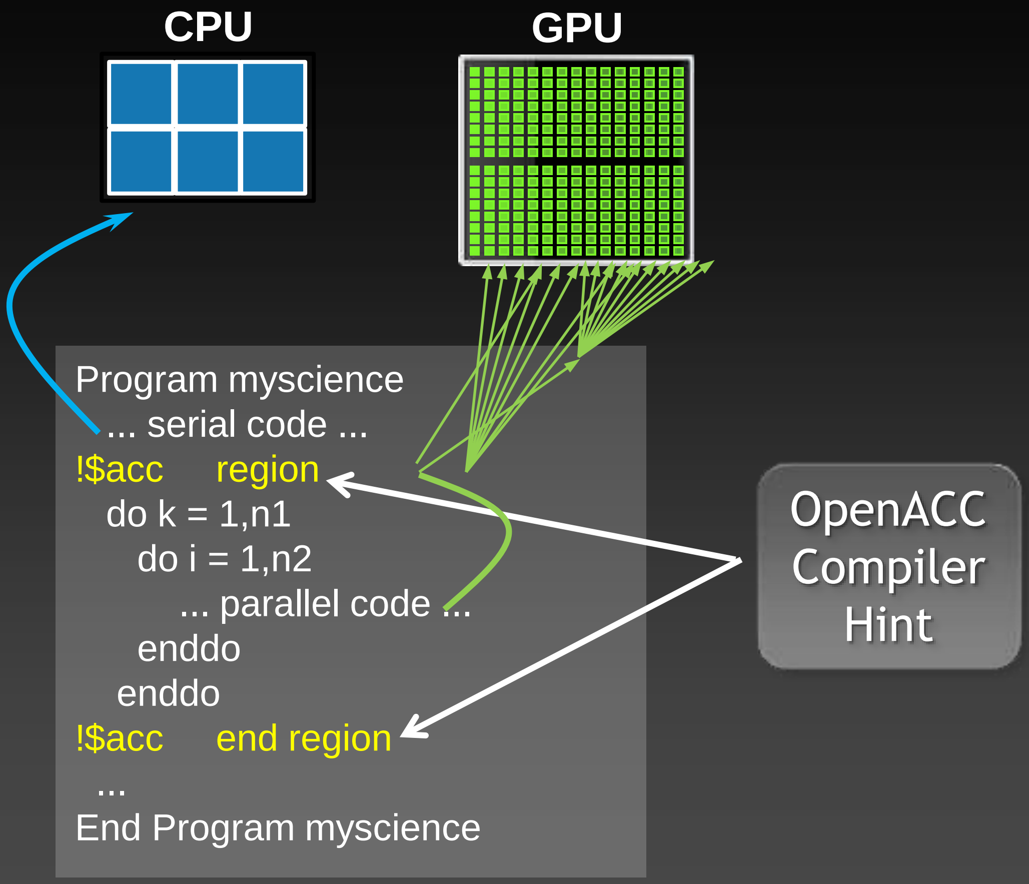


Enabling Other Platforms

Enabling Research in GPU Computing



OpenACC Directives



```
Program myscience
 ... serial code ...
!$acc     region
 do k = 1,n1
   do i = 1,n2
       ... parallel code ...
   enddo
  enddo
!$acc     end region
 ...
End Program myscience
```

Your original Fortran or C code

Easy, Open, Powerful

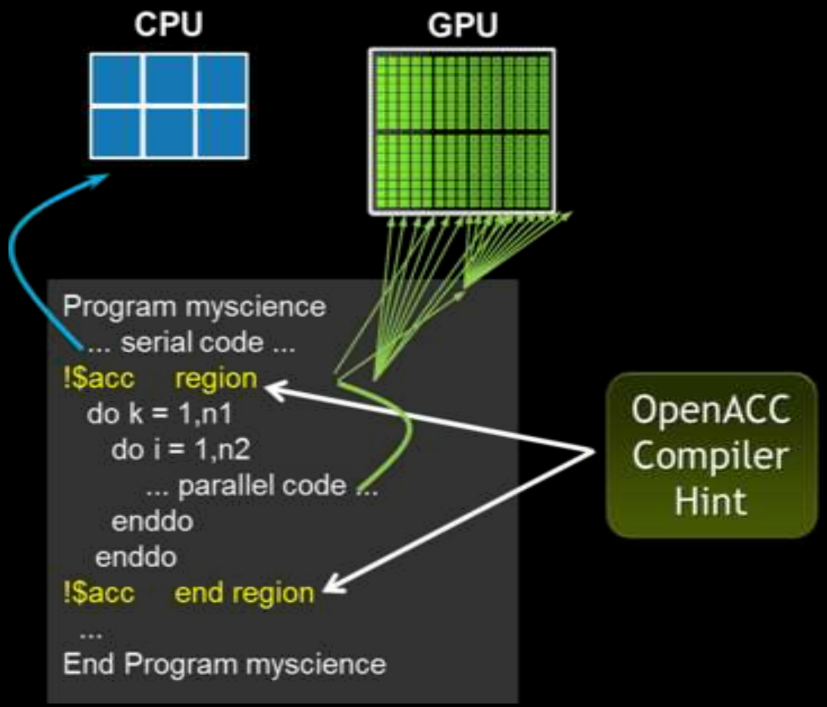
- Simple Compiler hints
- Works on multicore CPUs & many core GPUs
- Compiler Parallelizes code
- Future Integration into OpenMP standard planned

http://www.openacc.org

NVIDIA. CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

Proposed Additions for OpenACC 2.0

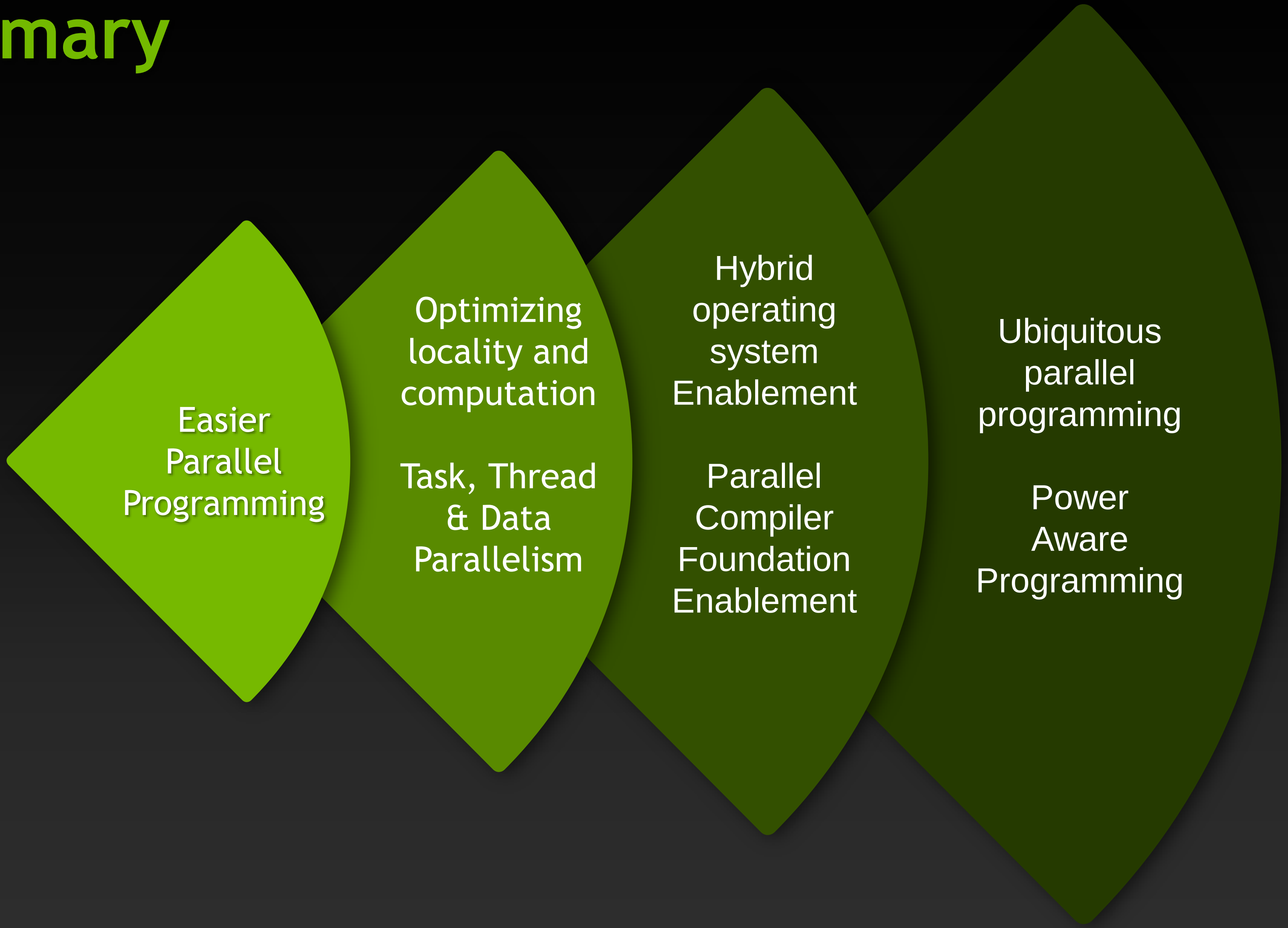
- Address ambiguities in existing spec
- List of 30+ features to be added
- Nested parallelism
- Separate compilation
- Function calls
- Data directives for control, unstructured data, deep copy for C++ structures, non-contiguous memory
- Multiple devices
- Profiling interface
- Certification – OpenACC test suite



```
Program myscience
  ... serial code ...
!$acc   region
   do k = 1,n1
      do i = 1,n2
          ... parallel code ...
      enddo
   enddo
!$acc   end region
   ...
End Program myscience
```

http://www.openacc.org

Summary



Today

Easier Parallel Programming

Optimizing locality and computation

Task, Thread & Data Parallelism

Hybrid operating system Enablement

Parallel Compiler Foundation Enablement

Ubiquitous parallel programming

Power Aware Programming

Thank you.
Questions?

Axel Koehler
akoehler@nvidia.com