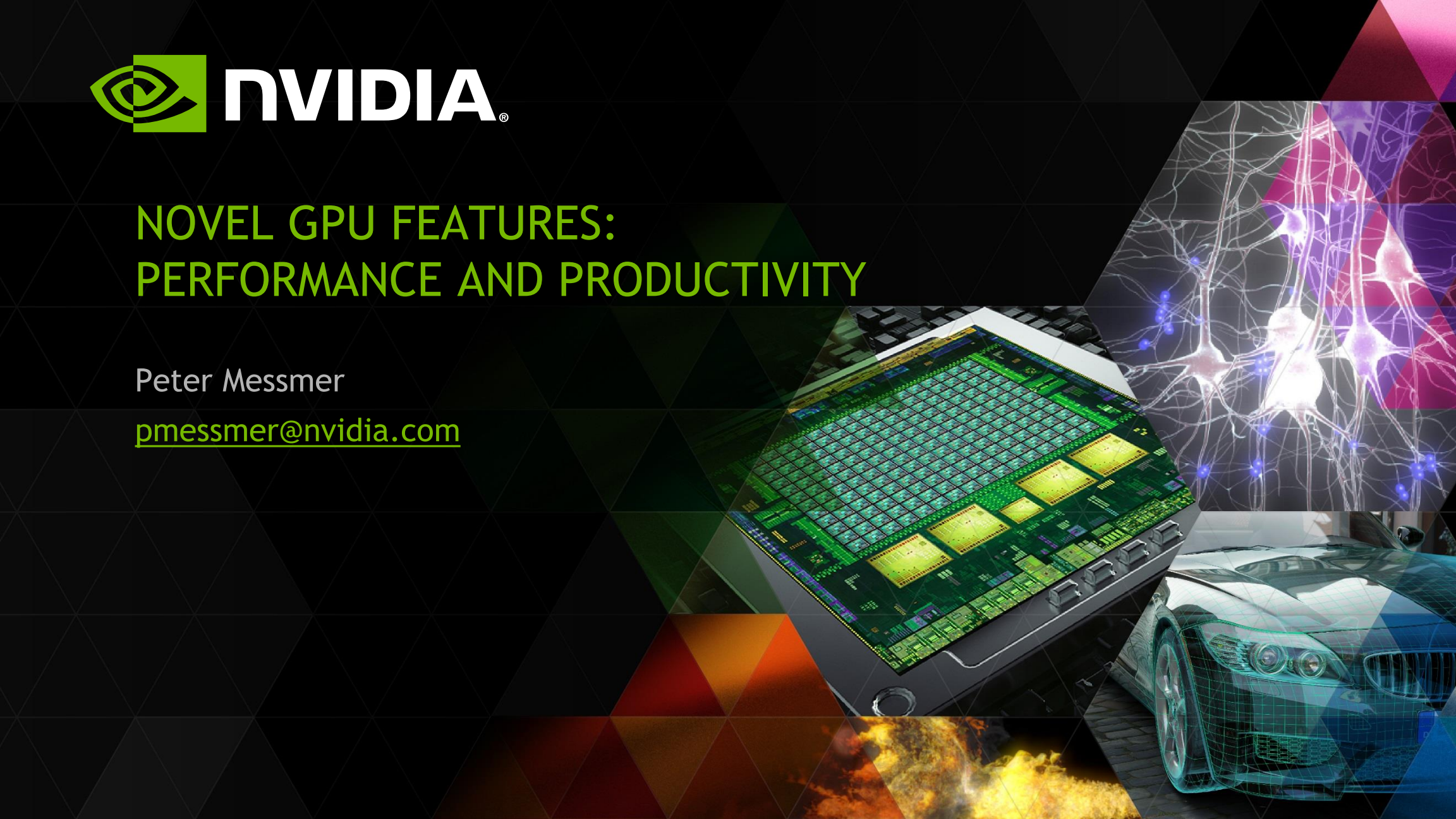


NOVEL GPU FEATURES: PERFORMANCE AND PRODUCTIVITY

Peter Messmer

pmessmer@nvidia.com

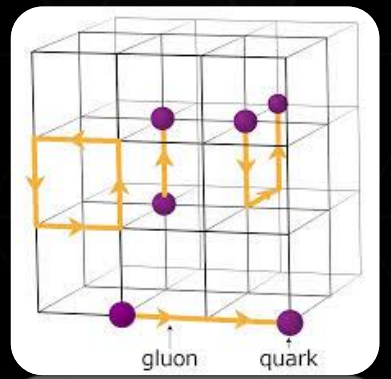
COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger

High-Level Trigger

Monte Carlo Analysis

Lattice QCD



COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger

Connectivity
Latency control
Power limited

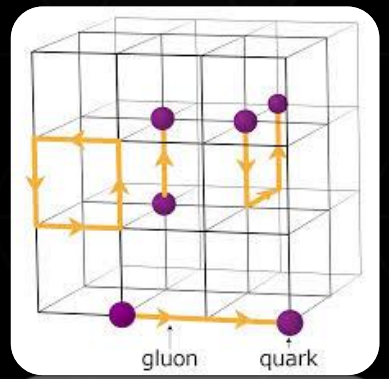
High-Level Trigger

Data volume
Computer vision
Unsupervised learning

Monte Carlo Analysis

Portability
Legacy codes
Limited parallelism
Geometry processing

Lattice QCD



Connectivity
Low latency
Bandwidth,
bandwidth,
bandwidth,..

COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger

Connectivity
Latency control
Power limited

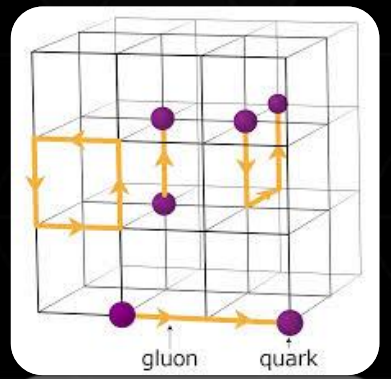
High-Level Trigger

Data volume
Computer vision
Unsupervised learning

Monte Carlo Analysis

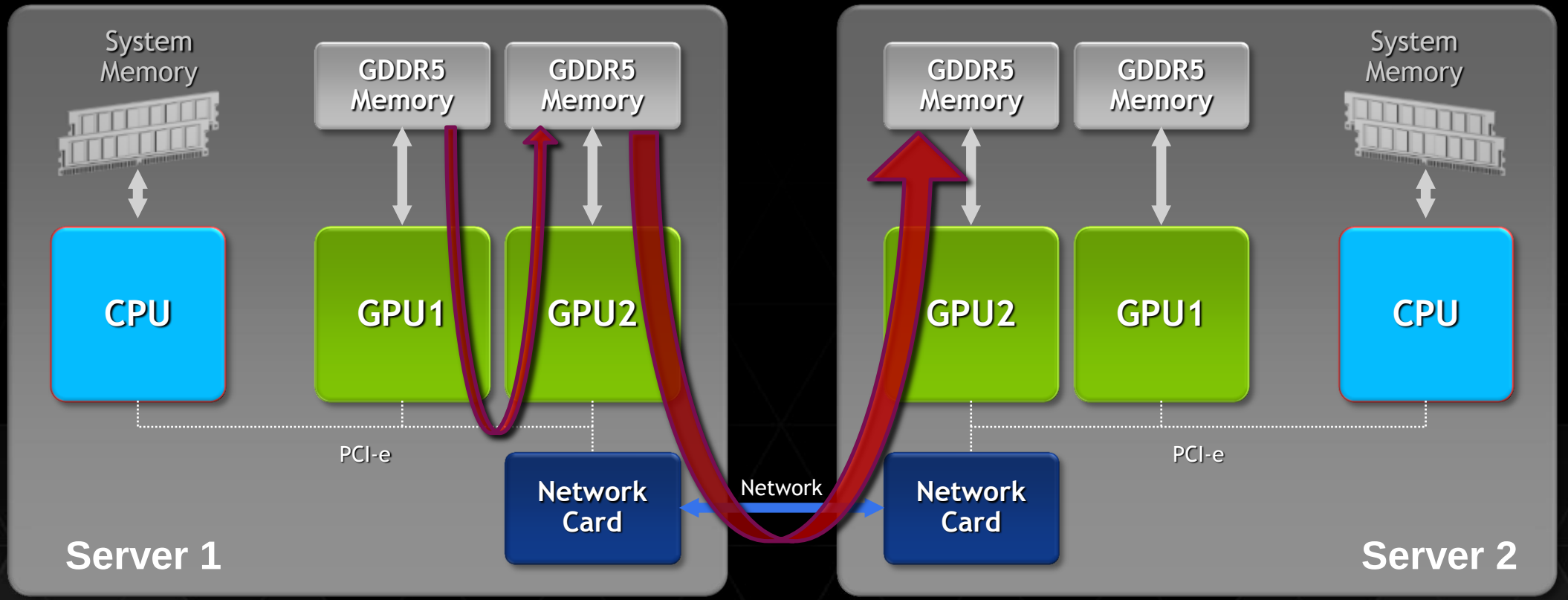
Portability
Legacy codes
Limited parallelism
Geometry processing

Lattice QCD



Connectivity
Low latency
Bandwidth,
bandwidth,
bandwidth,..

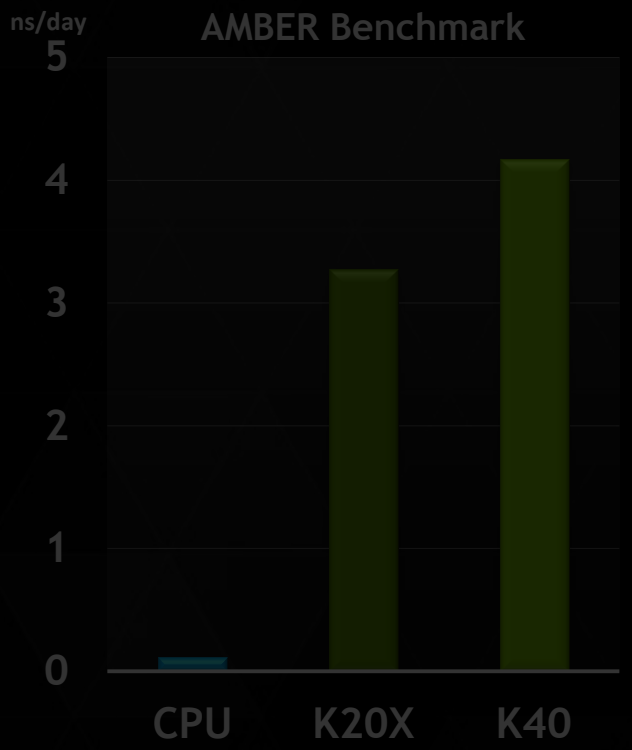
KEPLER ENABLES NVIDIA GPUDIRECT™ RDMA



http://docs.nvidia.com/cuda/gpudirect-rdma

TESLA K40
WORLD'S FASTEST ACCELERATOR

FASTER
1.4 TF| 2880 Cores | 288 GB/s



LARGER
2x Memory Enables More Apps

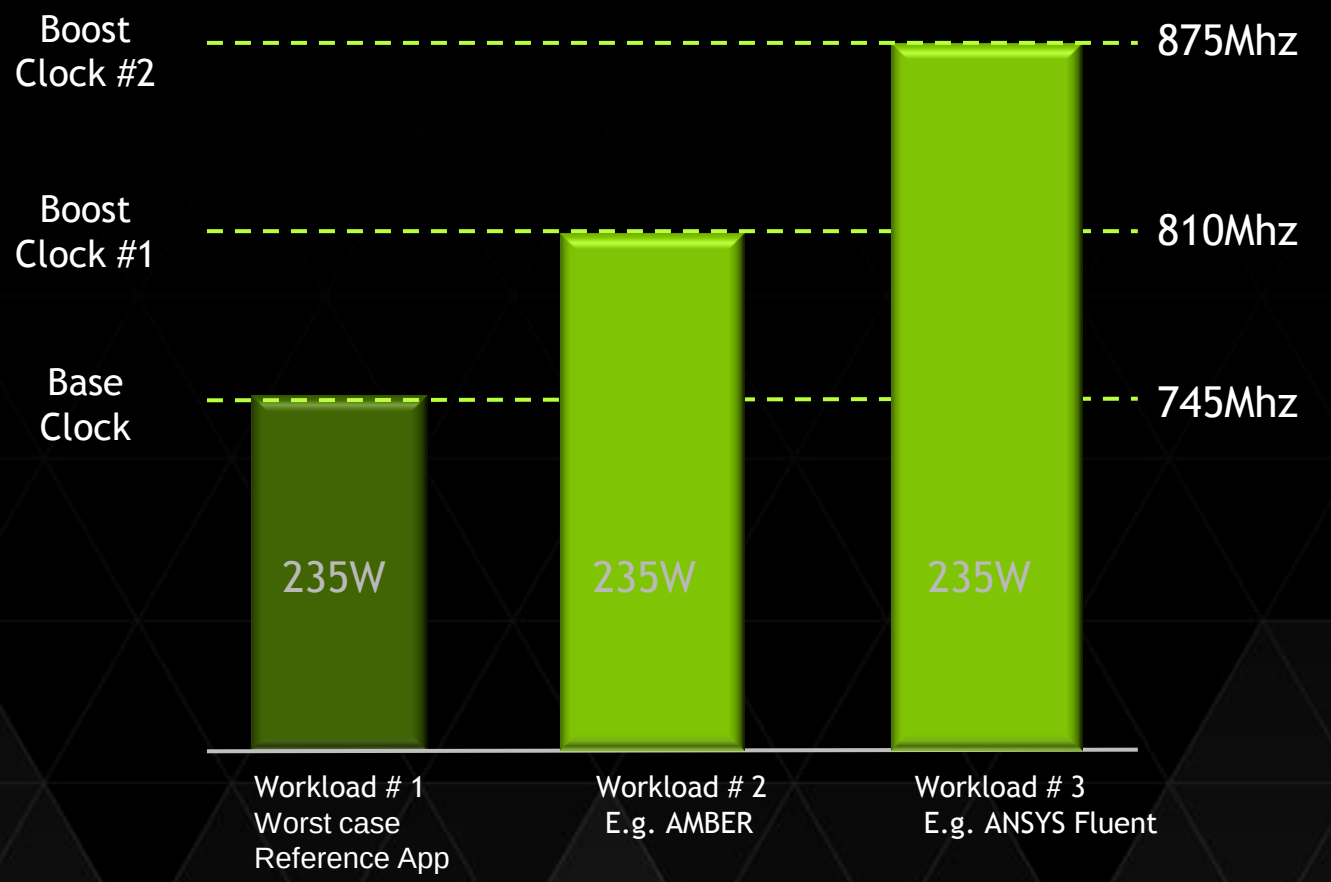
SMARTER
Unlock Extra Performance
Using Power Headroom

GPU Boost

AMBER Benchmark: SPFP-Nucleosome
CPU: Dual E5-2687W @ 3.10GHz, 64GB System Memory, CentOS 6.2, GPU systems: Single Tesla K20X or Single Tesla K40

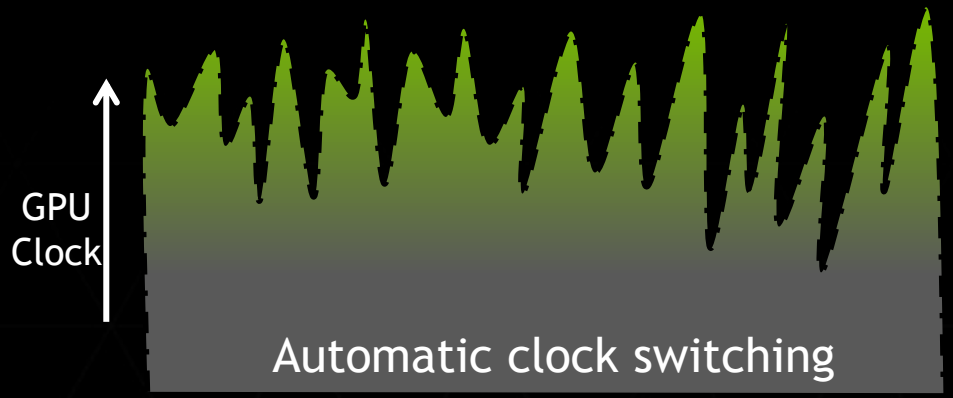
GPU BOOST ON TESLA K40

Convert Power Headroom to Higher Performance



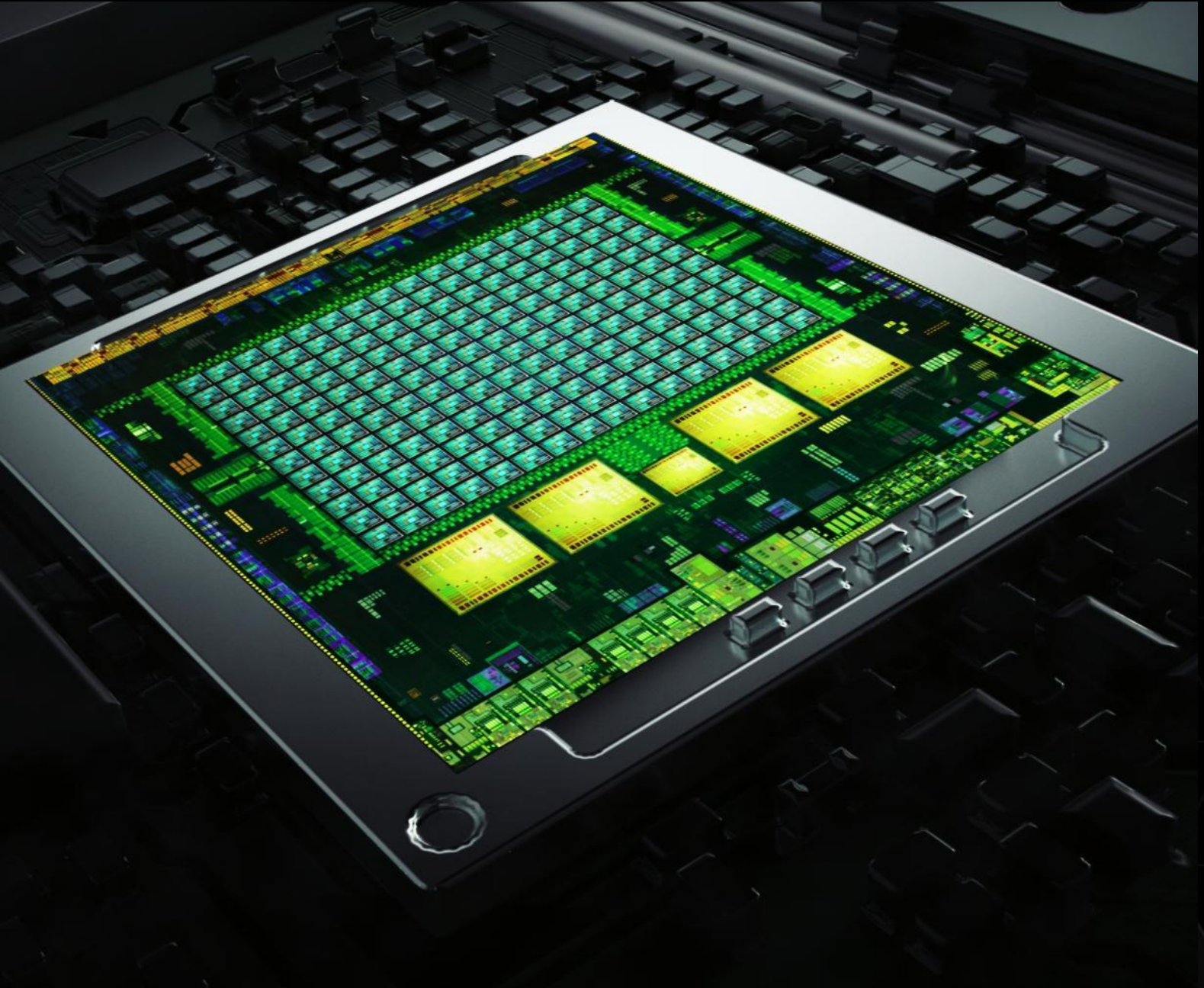
WORKLOAD BEHAVIOR WITH GPU BOOST

Non-Tesla



Tesla K40

Default	Boost	Base
Preset Options	Lock to base clock	3 Levels: Base, Boost1 or Boost2
Boost Interface	Control Panel	NV-SMI, NVML nvidia-smi -q -d CLOCK,SUPPORTED_CLOCKS nvidia-smi -ac <MEM clock, Graphics clock>
Target duration for boost clocks	~50% of run-time	100% of workload run time Must-have for HPC workload



TEGRA K1

IMPOSSIBLY ADVANCED

NVIDIA Kepler Architecture

4-Plus-1 Quad-Core A15

192 NVIDIA CUDA Cores

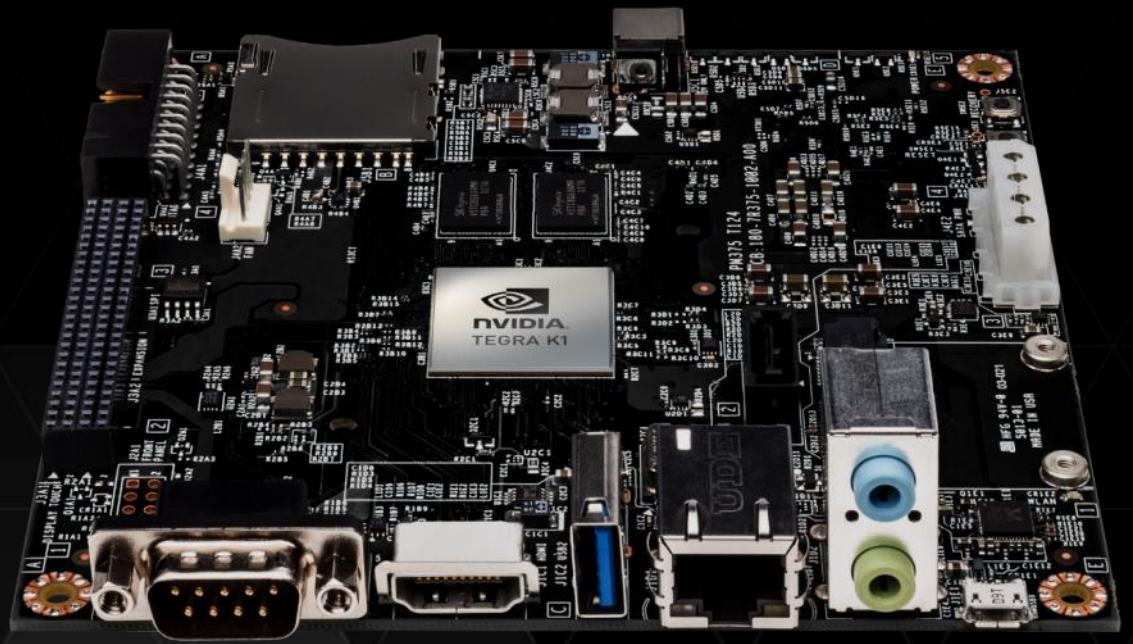
Compute Capability 3.2

326 GFLOPS

5 Watts

JETSON TK1

THE WORLD'S 1st EMBEDDED SUPERCOMPUTER



Development Platform for Embedded
Computer Vision, Robotics, Medical

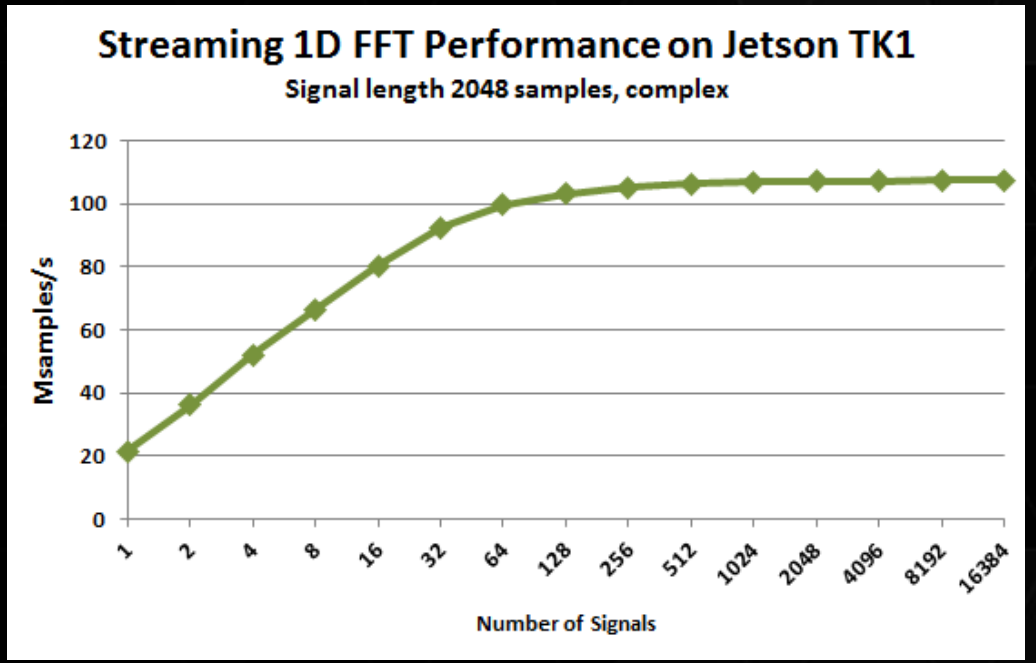
192 Cores · 326 GFLOPS

CUDA Enabled

Available Now

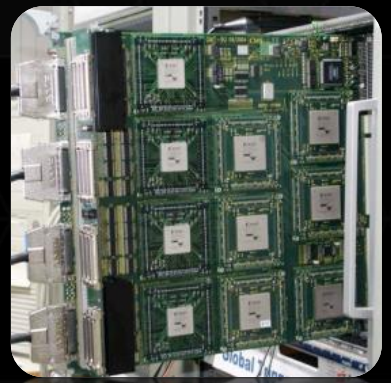
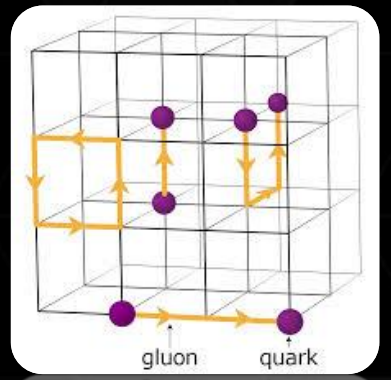
BATCHED 1D FFT PERFORMANCE ON JETSON

- Low power processor on sensor
- Host/Dev Bandwidth: 6 GB/s



- Dev/Dev FFT up to 780 Msamples/s
- Streaming FFT off host: 110 Msamples/s sustained
- About 1/13th of K20 performance

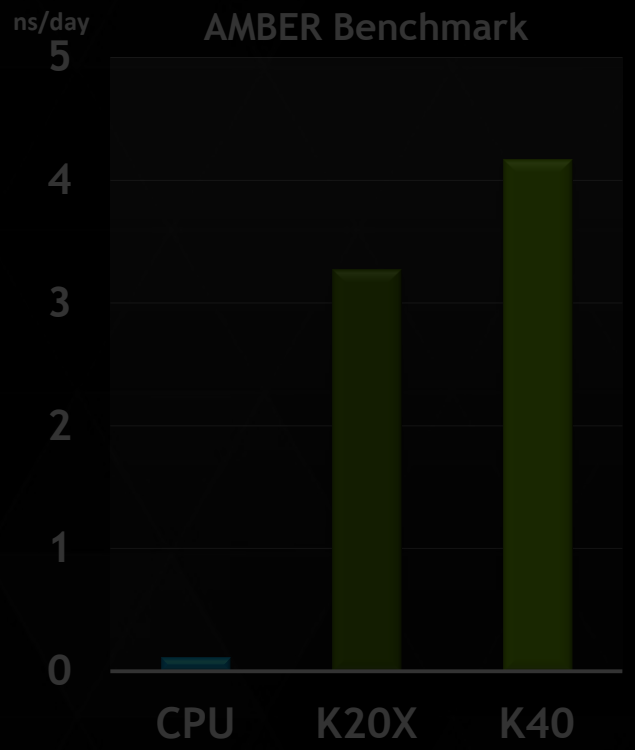
COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger	High-Level Trigger	Monte Carlo Analysis	Lattice QCD
			
Connectivity Latency control Power limited	Data volume Computer vision Unsupervised learning	Portability Legacy codes Limited parallelism Geometry processing	Connectivity Low latency Bandwidth, bandwidth, bandwidth,.

TESLA K40
WORLD'S FASTEST ACCELERATOR

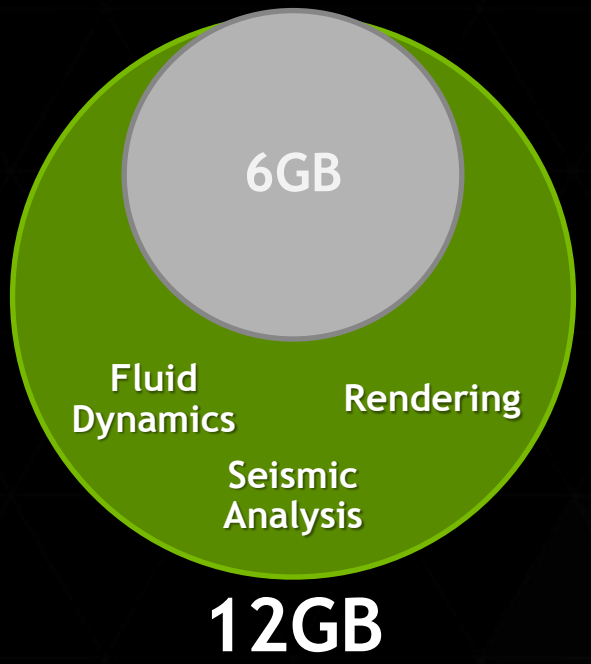
FASTER

1.4 TF| 2880 Cores | 288 GB/s



LARGER

2x Memory Enables More Apps



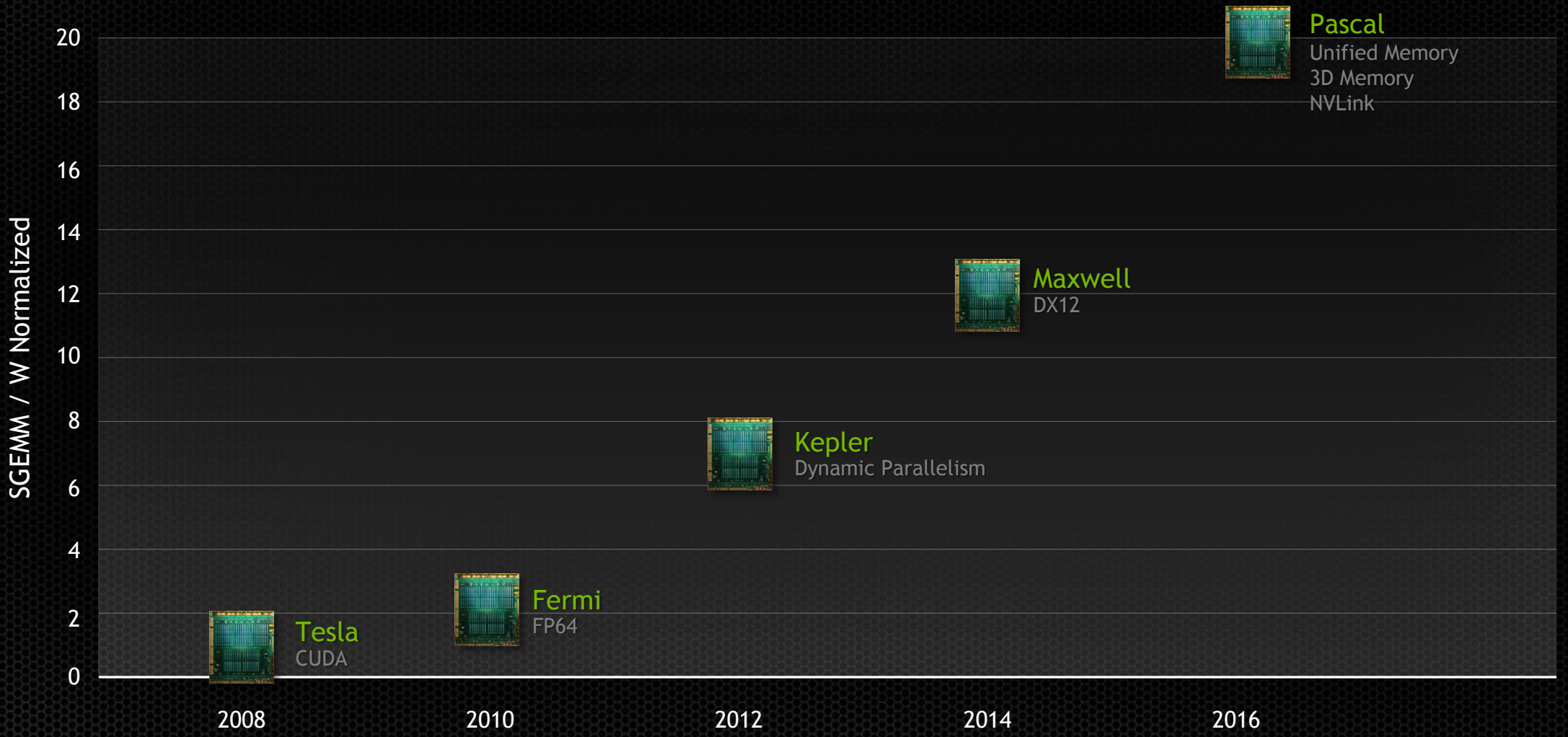
SMARTER

Unlock Extra Performance Using Power Headroom

GPU Boost

AMBER Benchmark: SPFP-Nucleosome
CPU: Dual E5-2687W @ 3.10GHz, 64GB System Memory, CentOS 6.2, GPU systems: Single Tesla K20X or Single Tesla K40

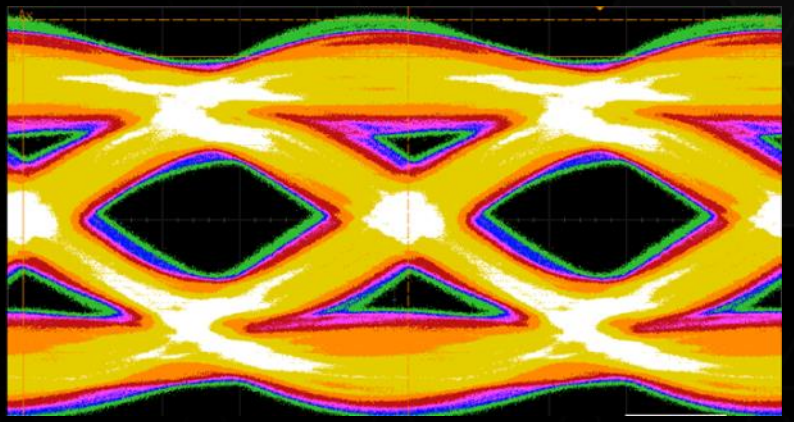
STRONG CUDA GPU ROADMAP



INTRODUCING NVLINK AND STACKED MEMORY

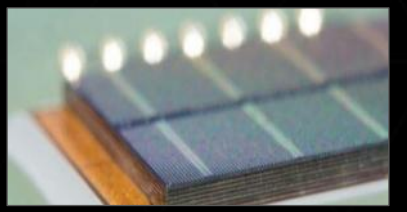
NVLINK

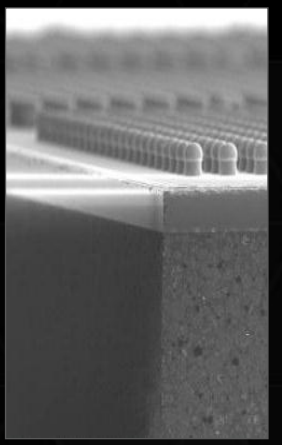
- GPU high speed interconnect
- 80-200 GB/s
- Planned support for POWER CPUs



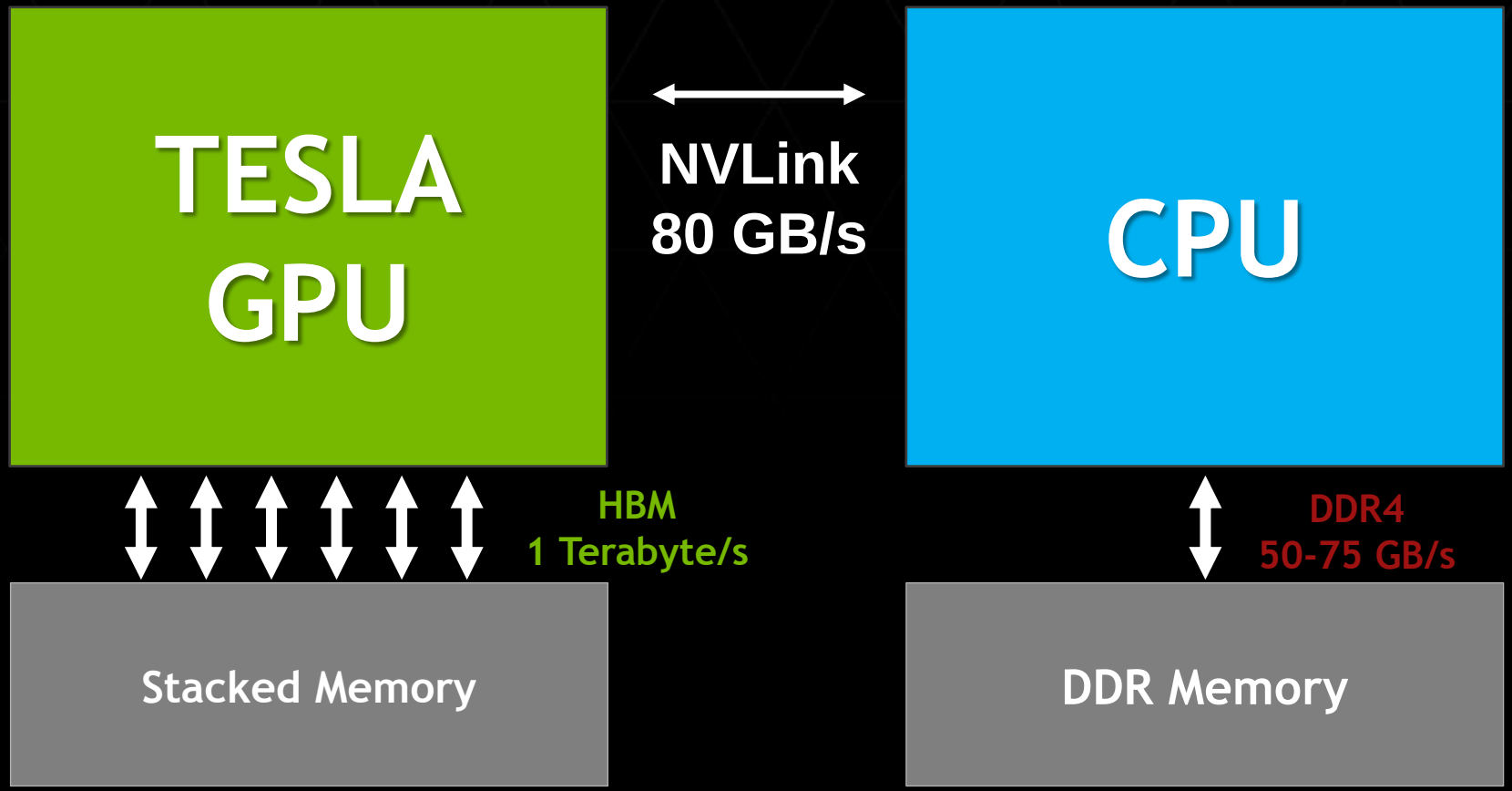
Stacked Memory

- 4x Higher Bandwidth (~1 TB/s)
- 3x Larger Capacity
- 4x More Energy Efficient per bit



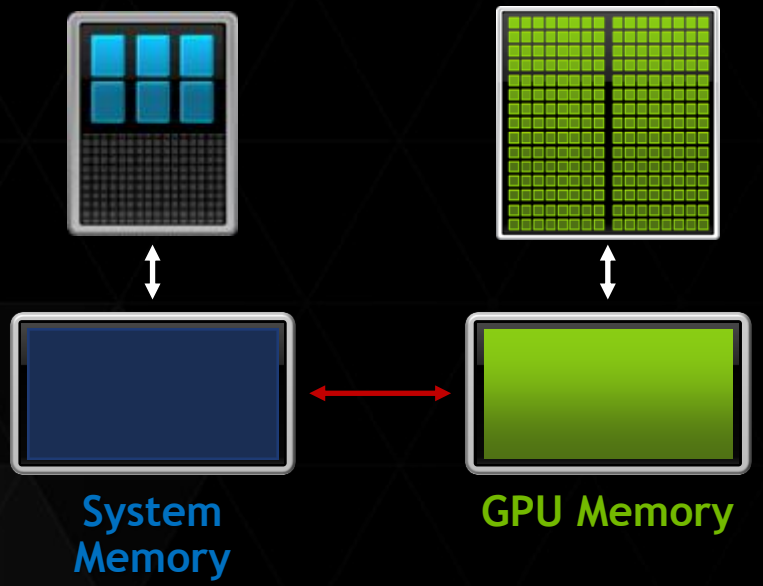


NVLINK ENABLES DATA TRANSFER AT SPEED OF CPU MEMORY

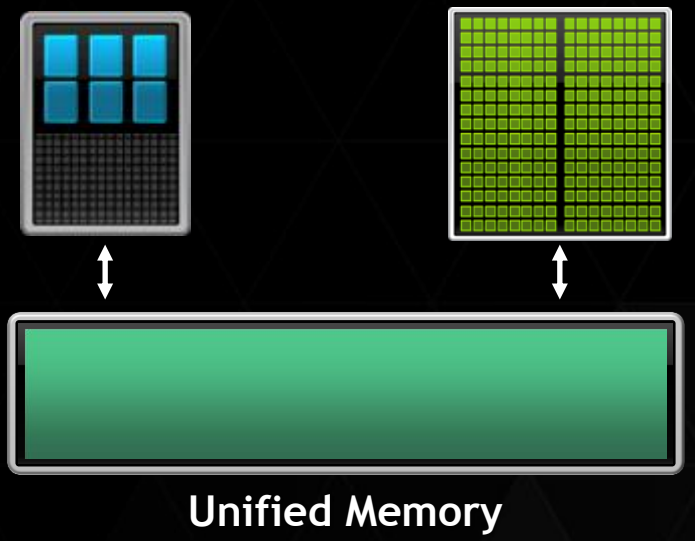


UNIFIED MEMORY
DRAMATICALLY LOWER DEVELOPER EFFORT

Developer View Today



Developer View With Unified Memory



SUPER SIMPLIFIED MEMORY MANAGEMENT CODE

CPU Code

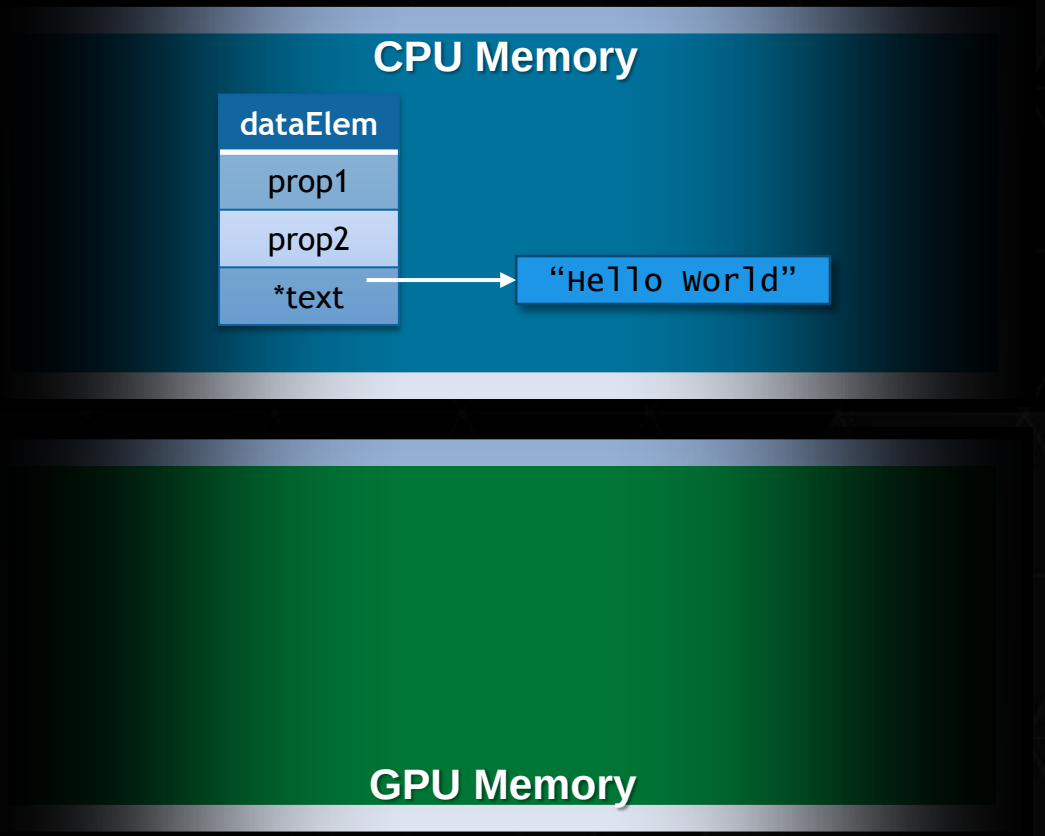
```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

CUDA 6 Code with Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

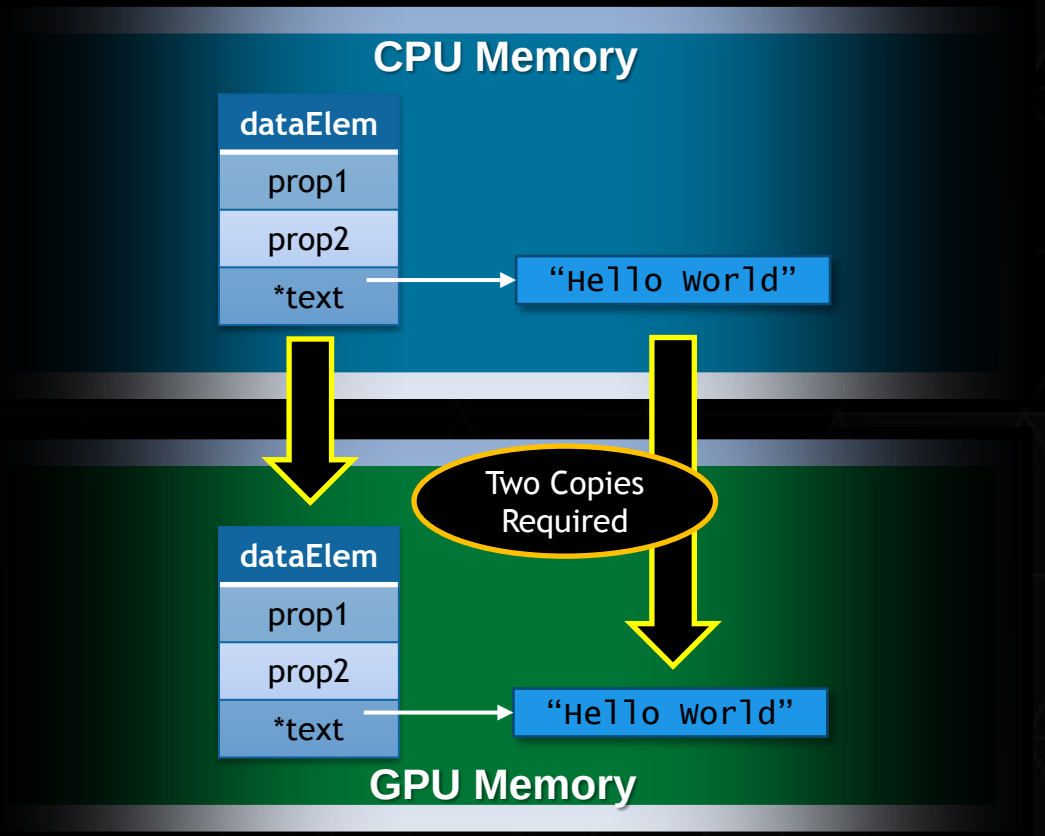
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
struct dataElem
{
    int prop1;
    int prop2;
    char *text;
};
```



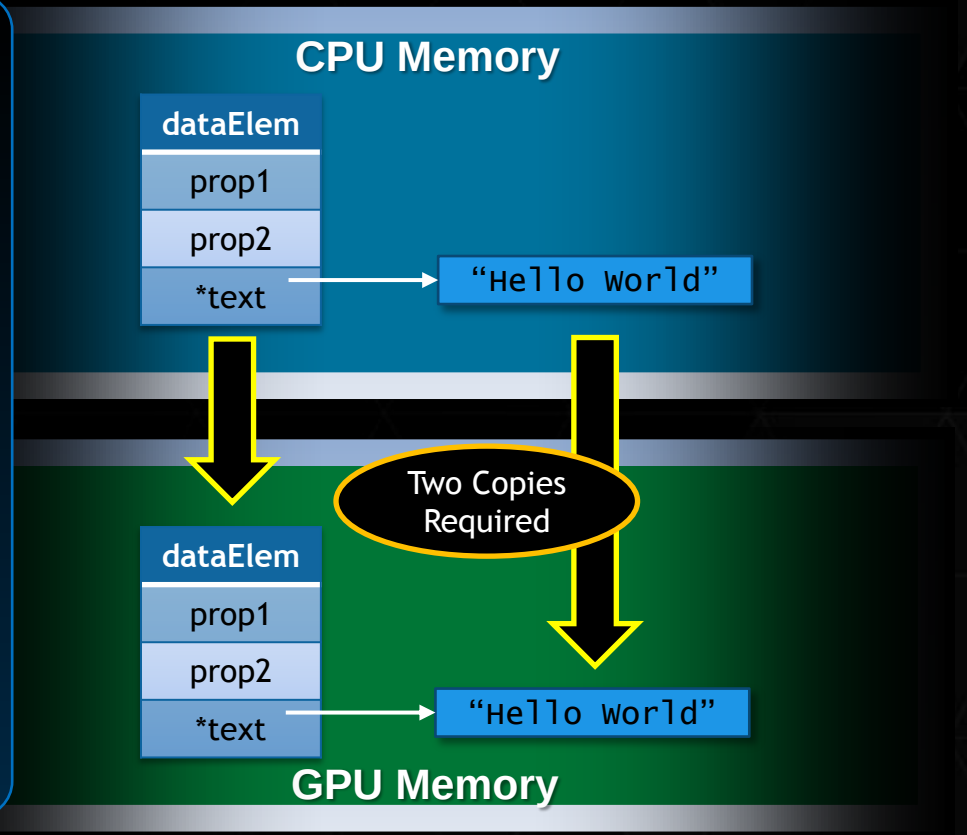
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
struct dataElem
{
    int prop1;
    int prop2;
    char *text;
};
```



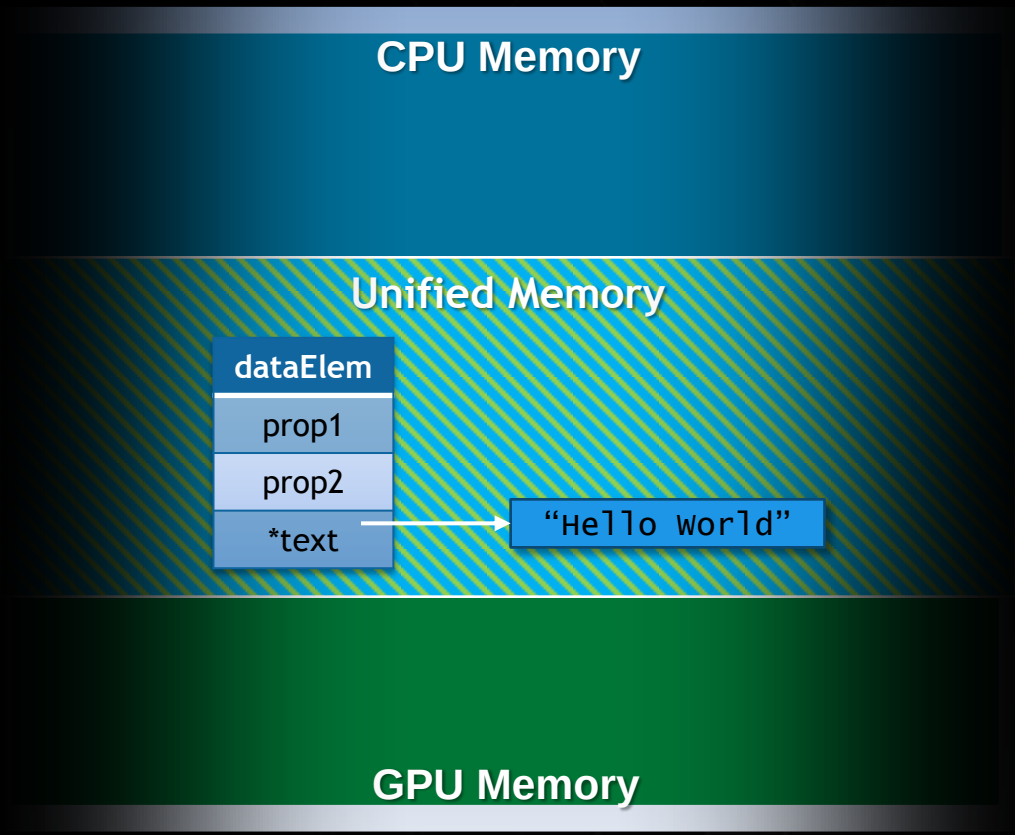
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
void launch(dataElem *elem) {
    dataElem *g_elem;
    char *g_text;

    int textlen = strlen(elem->text);

    // Allocate storage for struct and text
    cudaMalloc(&g_elem, sizeof(dataElem));
    cudaMalloc(&g_text, textlen);

    // Copy up each piece separately, including
    // new "text" pointer value
    cudaMemcpy(g_elem, elem, sizeof(dataElem));
    cudaMemcpy(g_text, elem->text, textlen);
    cudaMemcpy(&(g_elem->text), &g_text,
                                sizeof(g_text));

    // Finally we can launch our kernel, but
    // CPU & GPU use different copies of "elem"
    kernel<<< ... >>>(g_elem);
}
```



SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```
void launch(dataElem *elem) {
    kernel<<< ... >>>(elem);
}
```



UNIFIED MEMORY

Call Sort on CPU

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

    sort(data, N);

    fwrite(data, 1, N, out);
    free(data);
}
```

Call Sort on Kepler

```
void sortfile(FILE *in, FILE *out, int N)
{
    char *data = (char *)cudaMallocManaged(N);
    fread(data, 1, N, in);

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

    fwrite(data, 1, N, out);
    cudaFree(gpu_data);
}
```

Memory Management Becomes Performance Optimization

Call Sort on Pascal

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

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

    fwrite(data, 1, N, out);
    free(gpu_data);
}
```

No need for opt-in allocator

C++11 IN CUDA 6.5

- Experimental release in CUDA 6.5

```
nvcc -std=c++11 my_cpp11_code.cu
```

- Support for all C++11 features offered by host compiler in host code

- Currently no support for lambdas passed from host to device

THRUST: STL-LIKE CUDA TEMPLATE LIBRARY

- GPU(device) and CPU(host) vector class

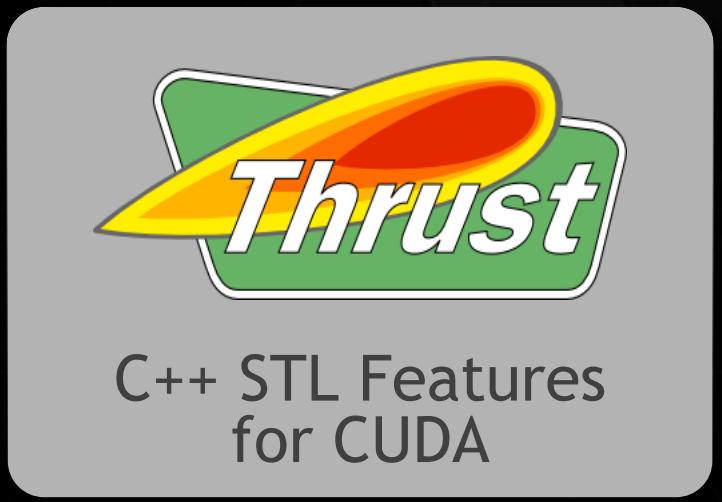
```
thrust::host_vector<float> H(10, 1.f);

thrust::device_vector<float> D = H;
```

- Iterators

```
thrust::fill(D.begin(), D.begin()+5, 42.f);

float* raw_ptr = thrust::raw_pointer_cast(D);
```

- Algorithms
  - Sort, reduce, transformation, scan, ..

```
thrust::transform(D1.begin(), D1.end(), D2.begin(), D2.end(),

 thrust::plus<float>());    // D2 = D1 + D2
```



CUB – CUDA UNBOUND

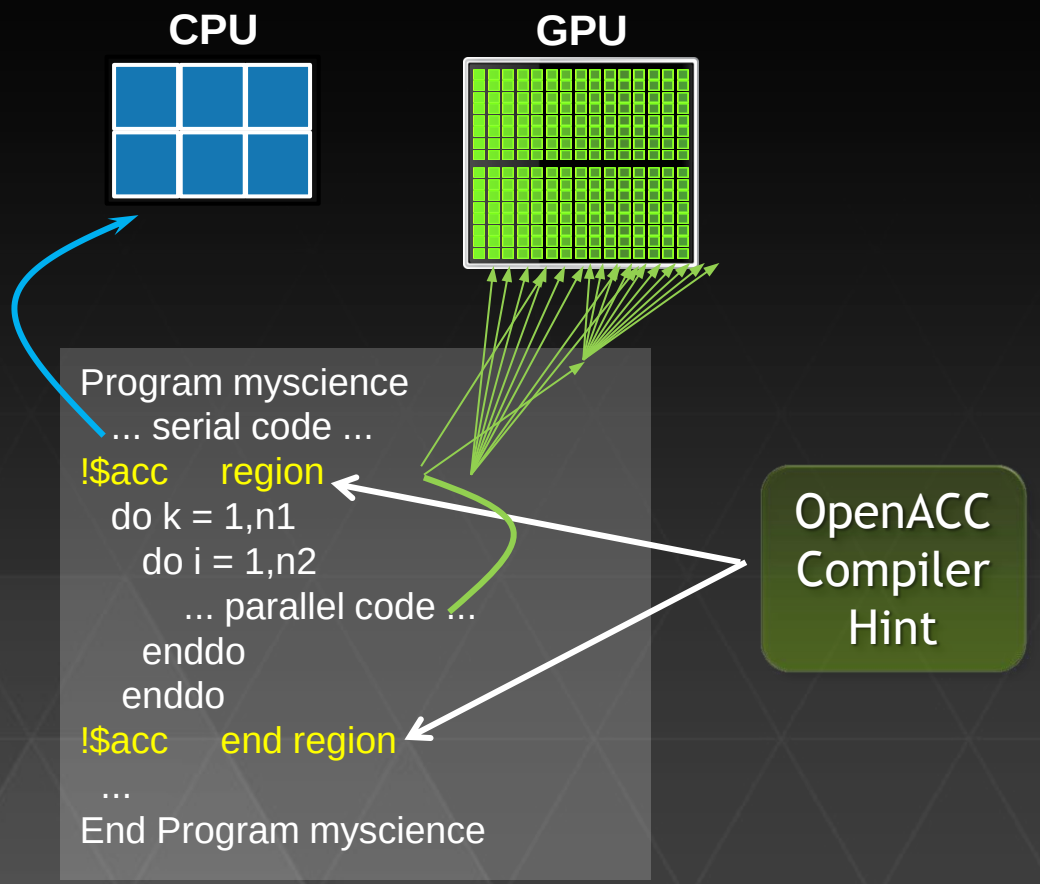
- Building blocks for kernels via C++ header library
- Data parallel primitives for thread blocks

- Collective primitives
  - scan, sort, histogram, ..
  - global memory transfer

- http://nvlabs.github.io/cub

```
1)  template<int BLOCK_SIZE, BlockReduceAlgorithm ALGORITHM>
2)  __global__
3)  void maxKernel(int* max, int* input)
4)  {
5)   int id=blockIdx.x*blockDim.x + threadIdx.x;
6)   typedef cub::BlockReduce<int,BLOCK_SIZE,ALGORITHM> BlockReduceT;
7)
8)   // Allocate temporary storage in shared memory
9)   __shared__ typename BlockReduceT::Temp temp_storage;
10)
11)
12)  int block_max=BlockReduceT(temp_storage).Reduce(val,cub::Max());
13)
14)  // update global max value
15)  if(threadIdx.x == 0) atomicMax(max,block_max);
16)
17)  return;
18) }
```

OPENACC DIRECTIVES



Easy, Open, Powerful

- Simple Compiler hints
- Works on multicore CPUs & many core GPUs
- Compiler Parallelizes code

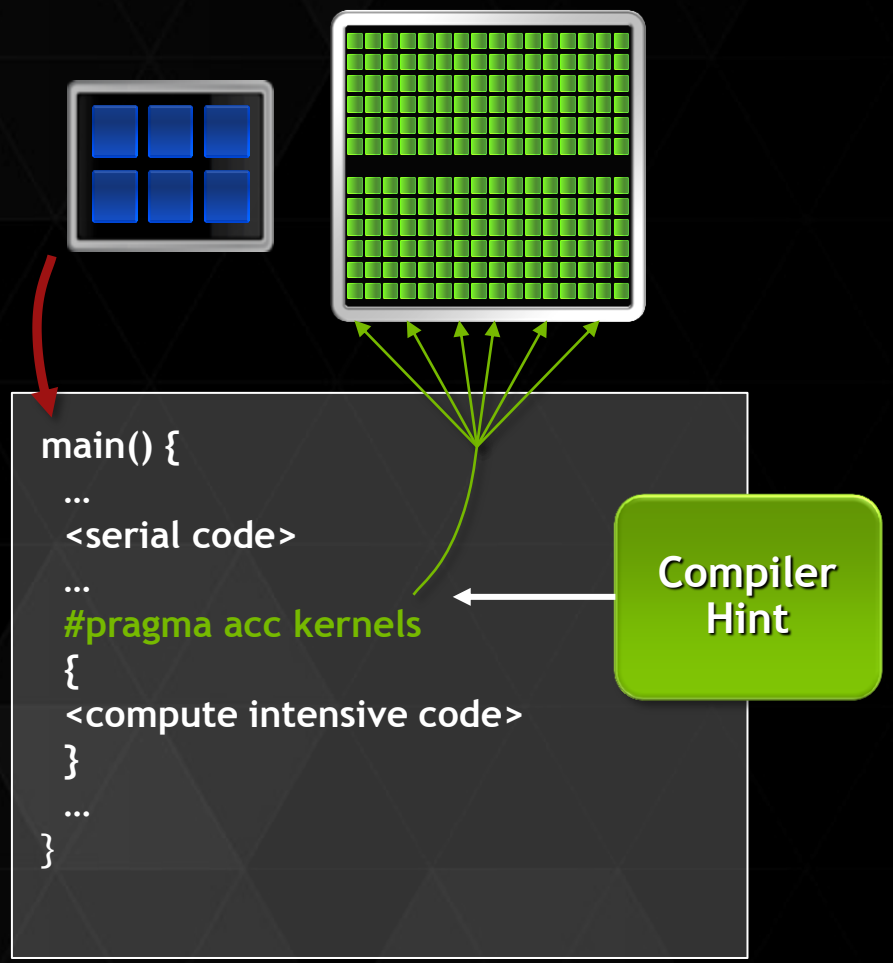
http://www.openacc.org

NVIDIA

PGI

CRAY THE SUPERCOMPUTER COMPANY

OPENACC: OPEN, SIMPLE, PORTABLE



- Open Standard
- Easy, Compiler-Driven Approach
- Portable on GPUs and Xeon Phi

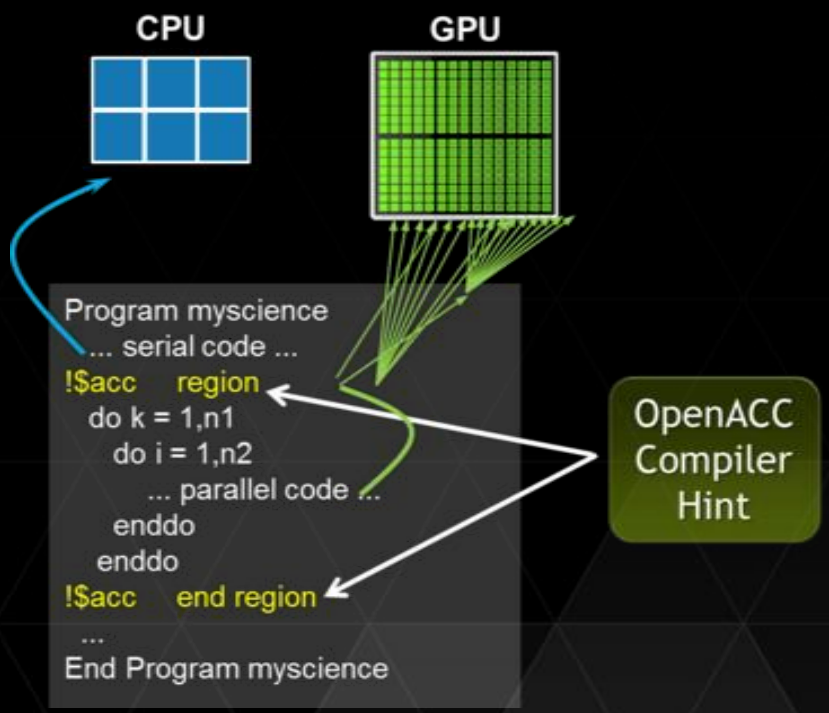
CAM-SE Climate
6x Faster on GPU
Top Kernel: 50% of Runtime

ADDITIONS FOR OPENACC 2.0

OpenACC
DIRECTIVES FOR ACCELERATORS

- Procedure calls
- Separate compilation
- Nested parallelism
- Device-specific tuning, multiple devices
- Data management features and global data
- Multiple host thread support
- Loop directive additions
- Asynchronous behavior additions
- New API routines for target platforms

(CUDA, OpenCL, Intel Coprocessor Offload Infrastructure)



COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger

Connectivity
Latency control
Power limited

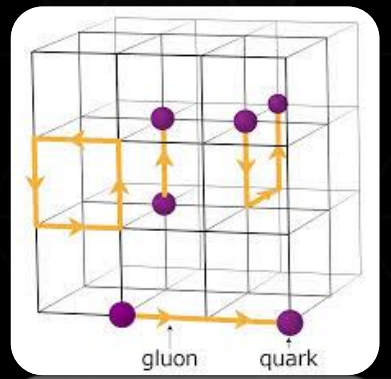
High-Level Trigger

Data volume
Computer vision
Unsupervised learning

Monte Carlo Analysis

Portability
Legacy codes
Limited parallelism
Geometry processing

Lattice QCD

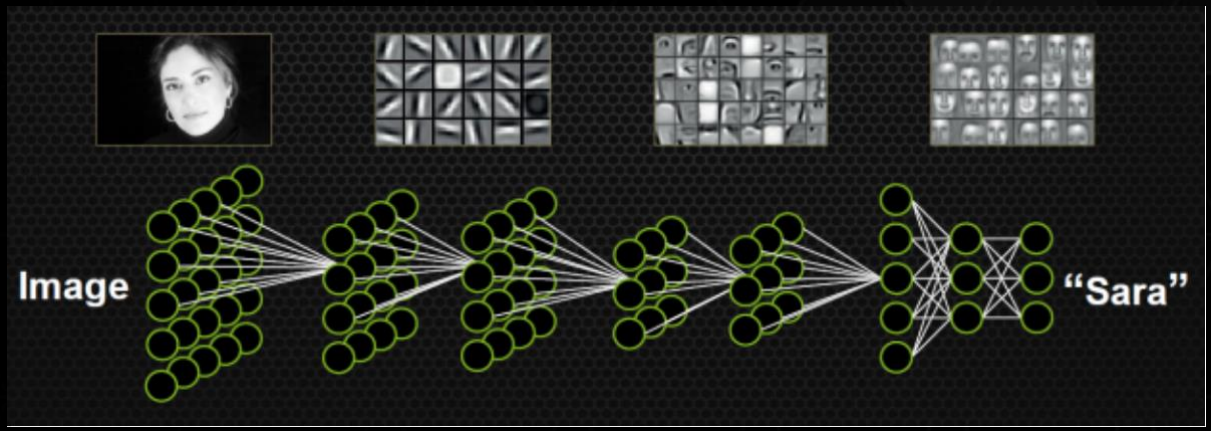


Connectivity
Low latency
Bandwidth,
bandwidth,
bandwidth,..

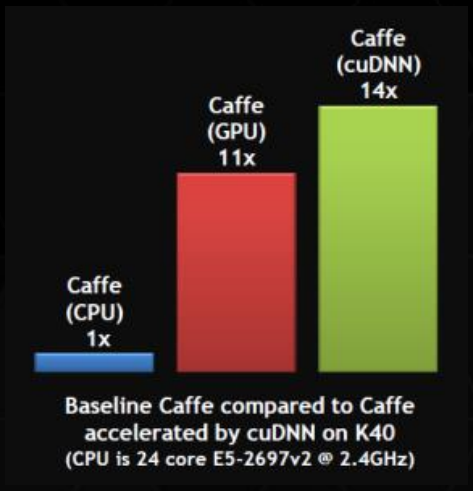
CUDNN

- Deep Neural Network Library
- Pre-packaged kernels for
  - convolution, pooling, softmax
  - Activations
  - ..

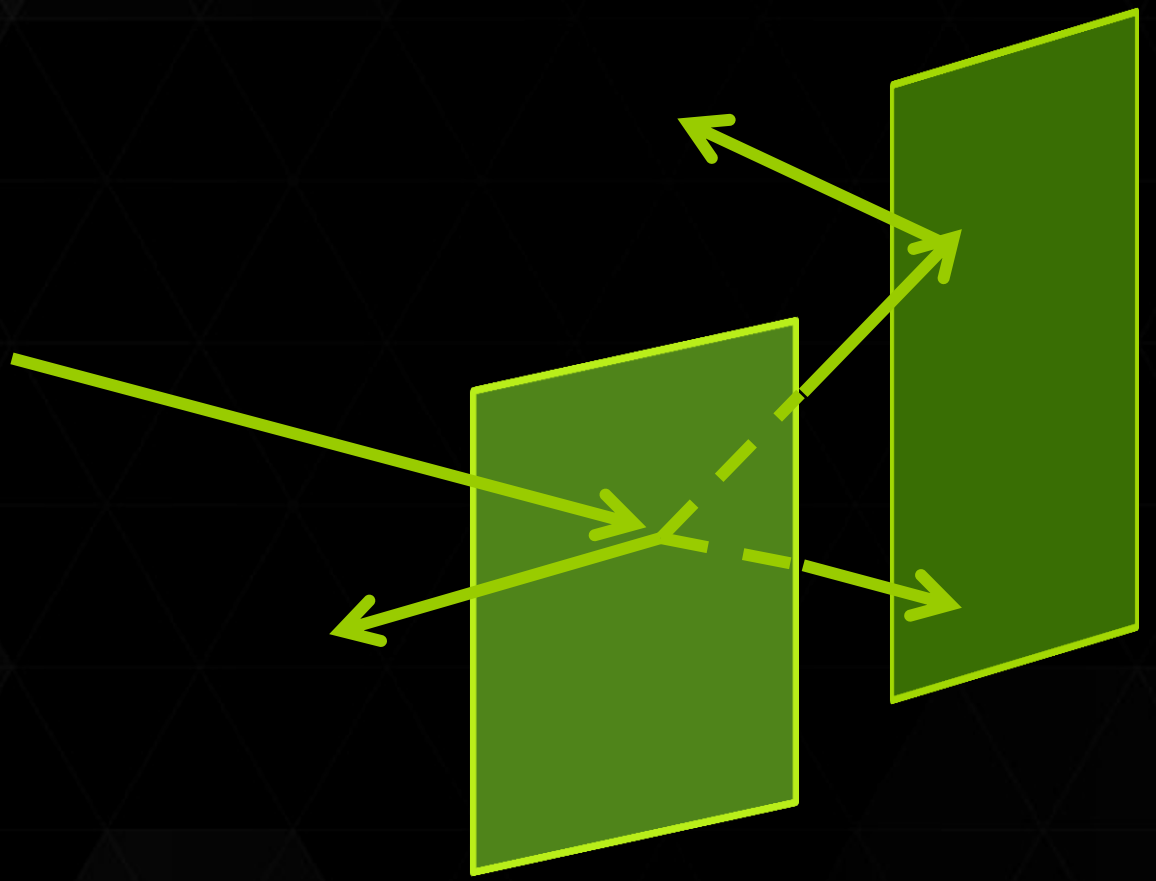
developer.nvidia.com/cuDNN




```
/* Set decriptors */
cudnnSetTensor4dDescriptor(InputDesc, CUDNN_TENSOR_NCHW, 128, 96, 221
cudnnSetFilterDescriptor(FilterDesc, 256, 96, 7, 7 );
cudnnSetConvolutionDescriptor(convDesc, InputDesc, FilterDesc,
                              pad_x, pad_y, 2, 2, 1, 1, CUDNN_CONVOLU

/* query output layout */
cudnnGetOutputTensor4dDim(convDesc, CUDNN_CONVOLUTION_FWD, &n_out, &c
                          &h_out, &w_out);
```



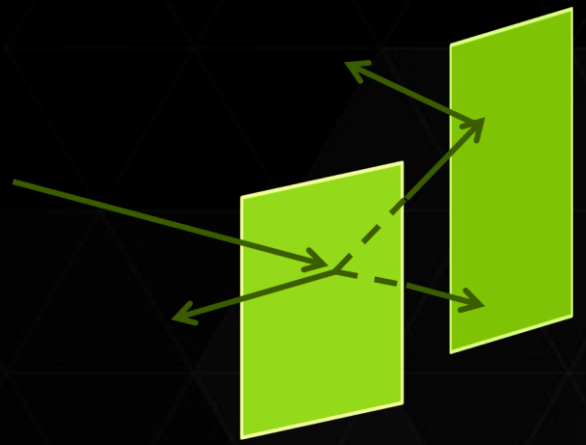
IF YOUR APPLICATION LOOKS LIKE THIS..



.. YOU MIGHT BE INTERESTED IN OPTIX

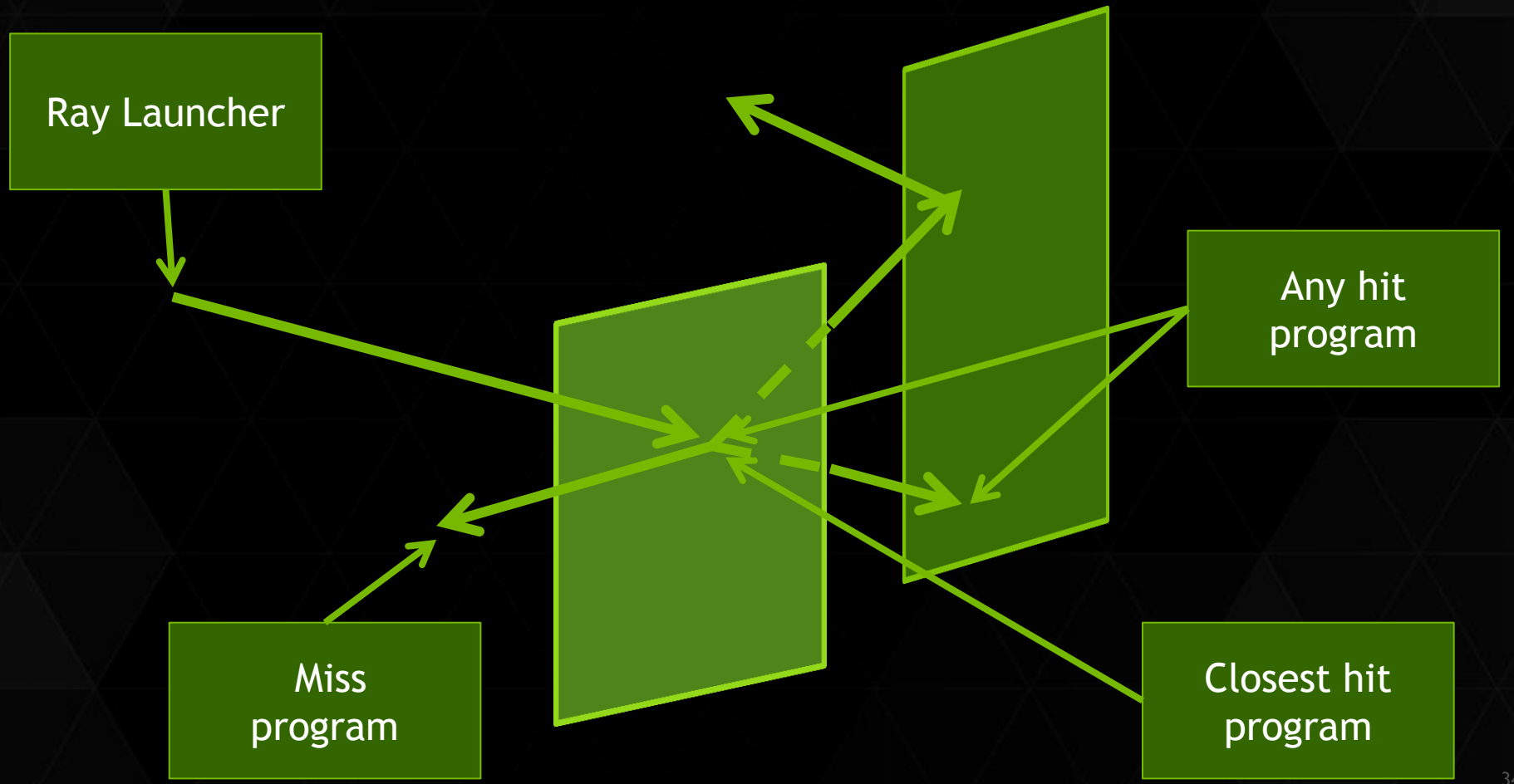
- Ray-tracing framework
  - Build your own RT application

- Generic Ray-Geometry interaction
  - Rays with arbitrary payloads



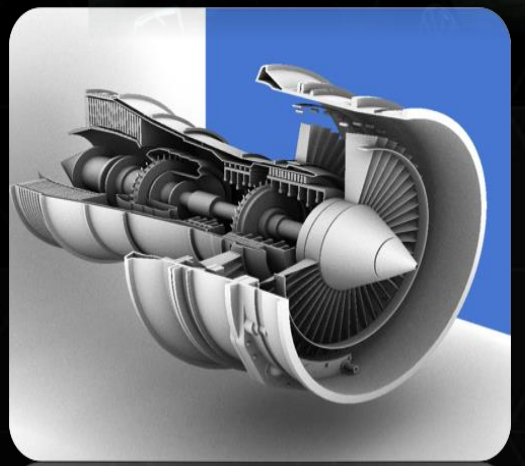
- Multi-GPU support

DIFFERENT PROGRAMS GET INVOKED FOR DIFFERENT RAYS



OPTIX PRIME: LOW-LEVEL RAY TRACING API

- OptiX simplifies implementation of RT apps
  - Manages memory, data transfers etc
- Sometimes an overkill for simple scene queries
  - E.g. just need visibility of triangulated geometries
- OptiX Prime: Low-Level Tracing API



SUMMARY

- Connectivity
  - GPU Direct RDMA
- Memory limit is non-issue
  - Large memory boards, Unified memory, NVLink
- Portability via abstraction
  - Thrust, CUB, C++11, OpenACC
- Frameworks and libraries for HEP tasks
  - OptiX, cuDNN, …
- GPUs are ready for HEP tasks!