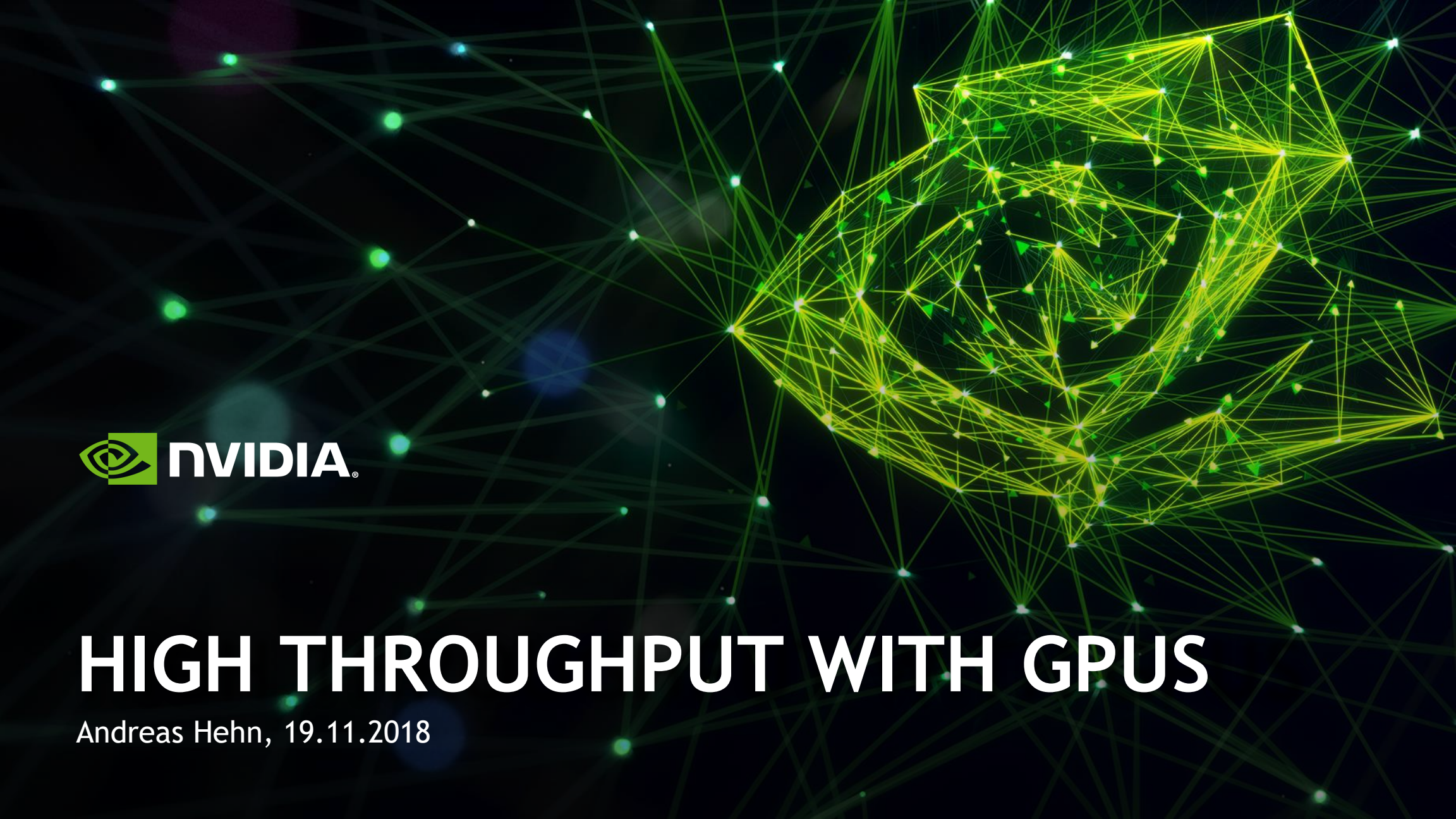
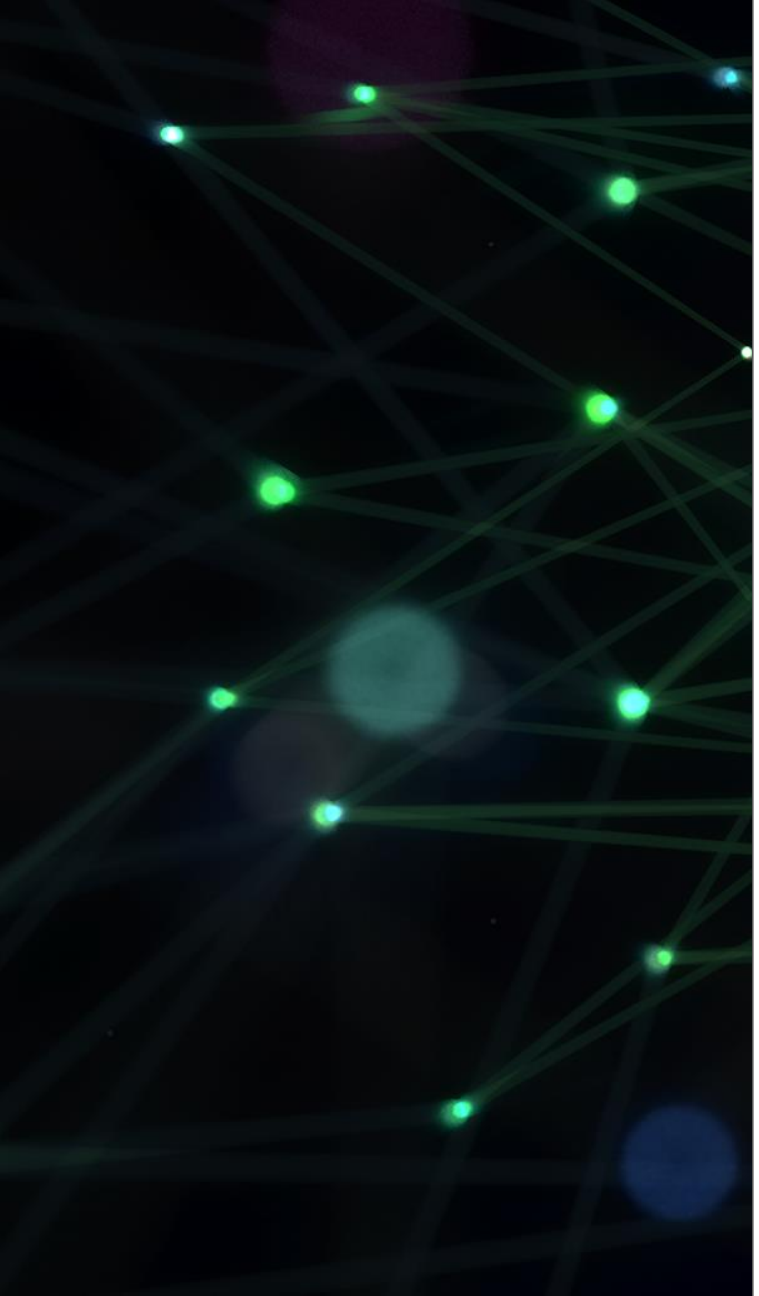


HIGH THROUGHPUT WITH GPUS

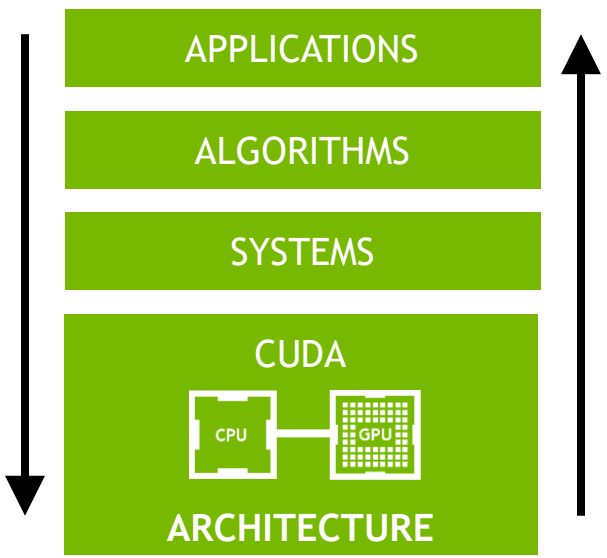
Andreas Hehn, 19.11.2018

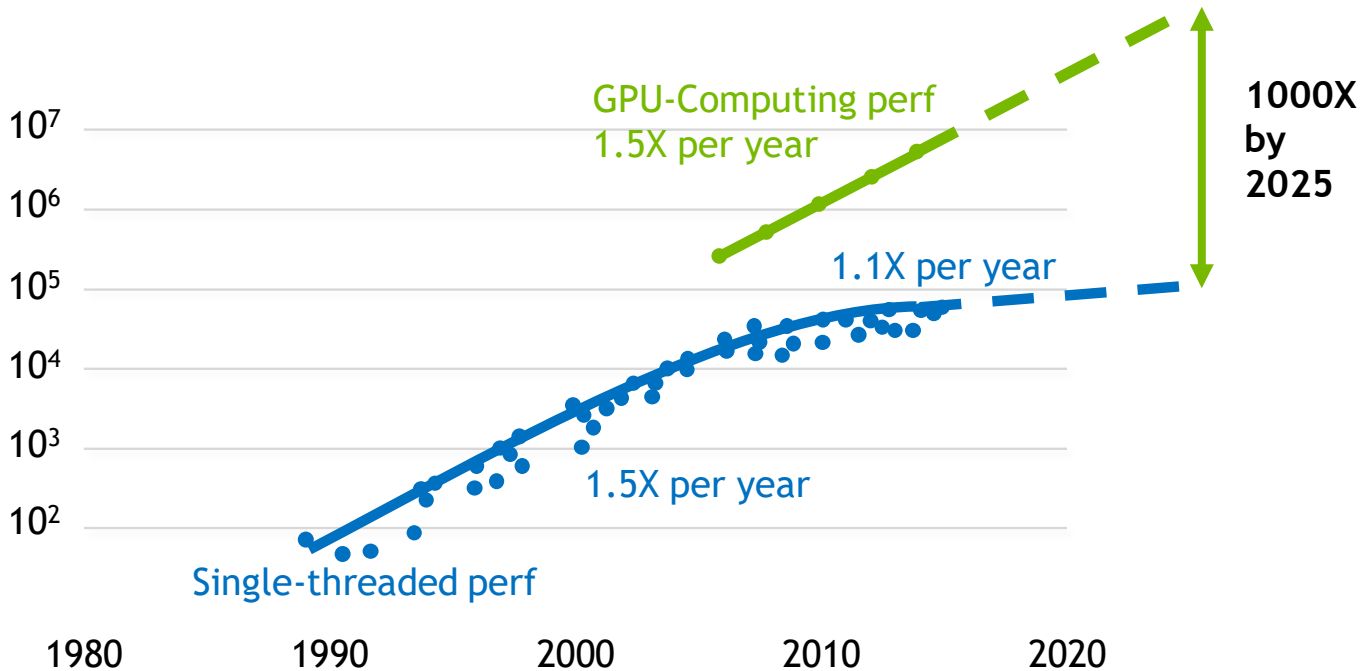


- WHY GPUS?
- HOW?
- DRIVING DEVELOPMENTS


WHY GPUS?

RISE OF GPU COMPUTING

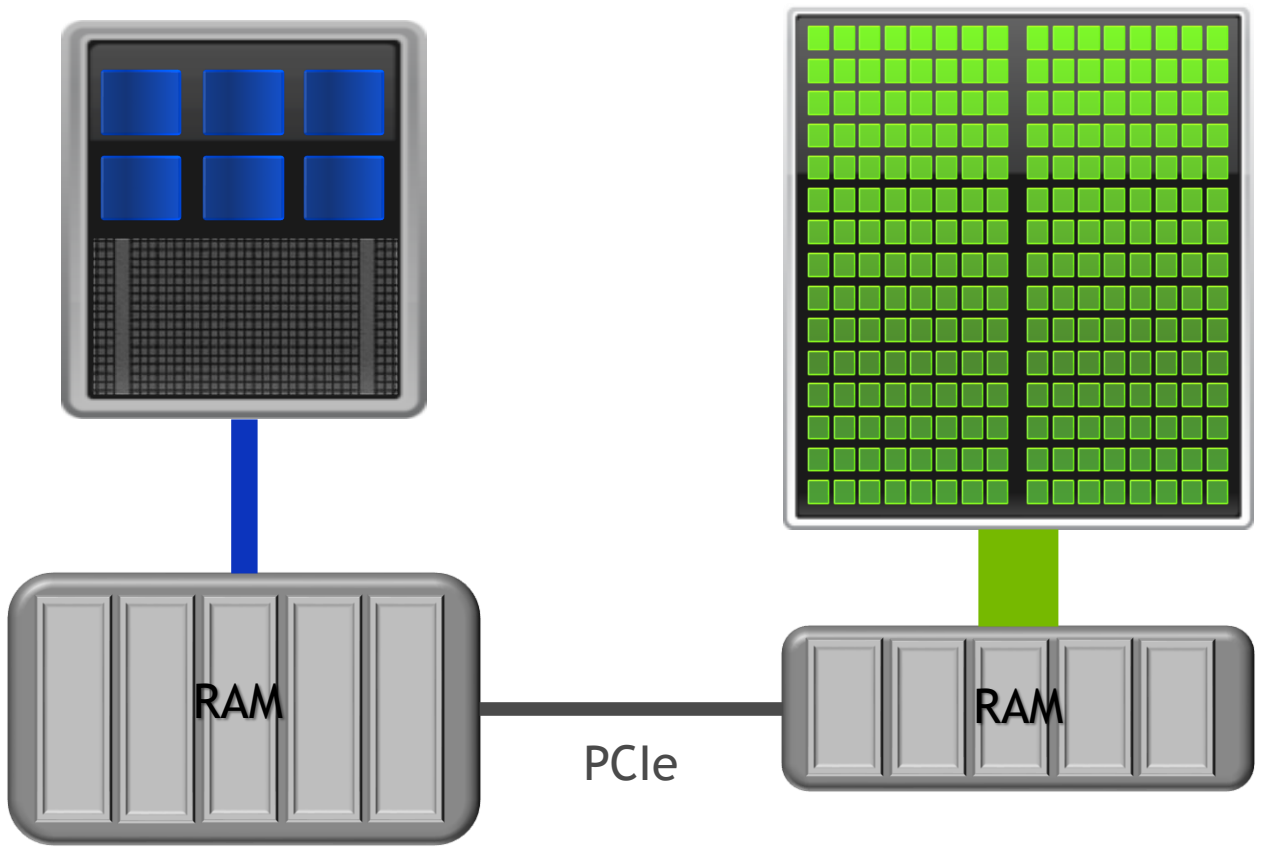




Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

GPU COMPUTING

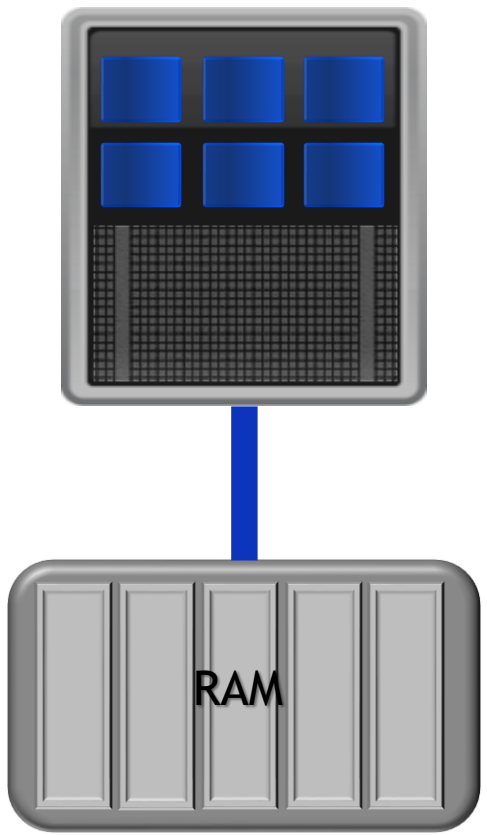
CPU
Optimized for
Serial Tasks



GPU
Optimized for
Parallel Tasks

GPU COMPUTING

CPU
Optimized for
Serial Tasks



Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- few threads, can run very quickly

Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

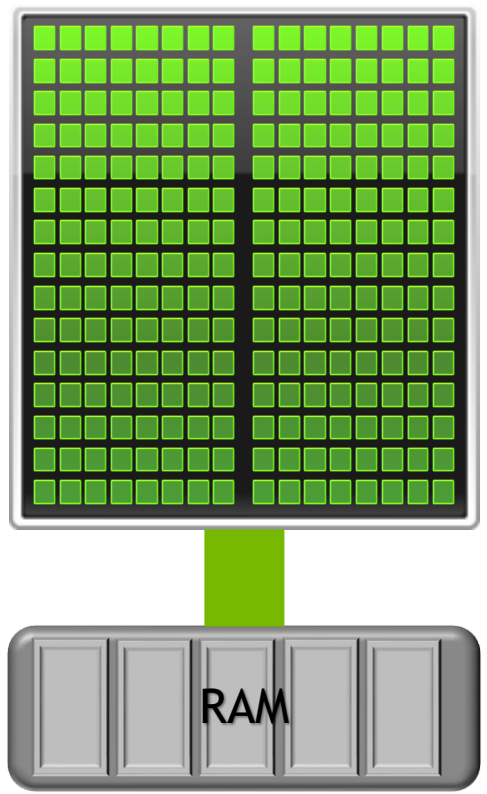
GPU COMPUTING

Strengths

- High bandwidth main memory
- Latency tolerant via parallelism
- Significantly more compute resources
- High throughput
- High performance/watt

Weaknesses

- Relatively low memory capacity
- Low per-thread performance

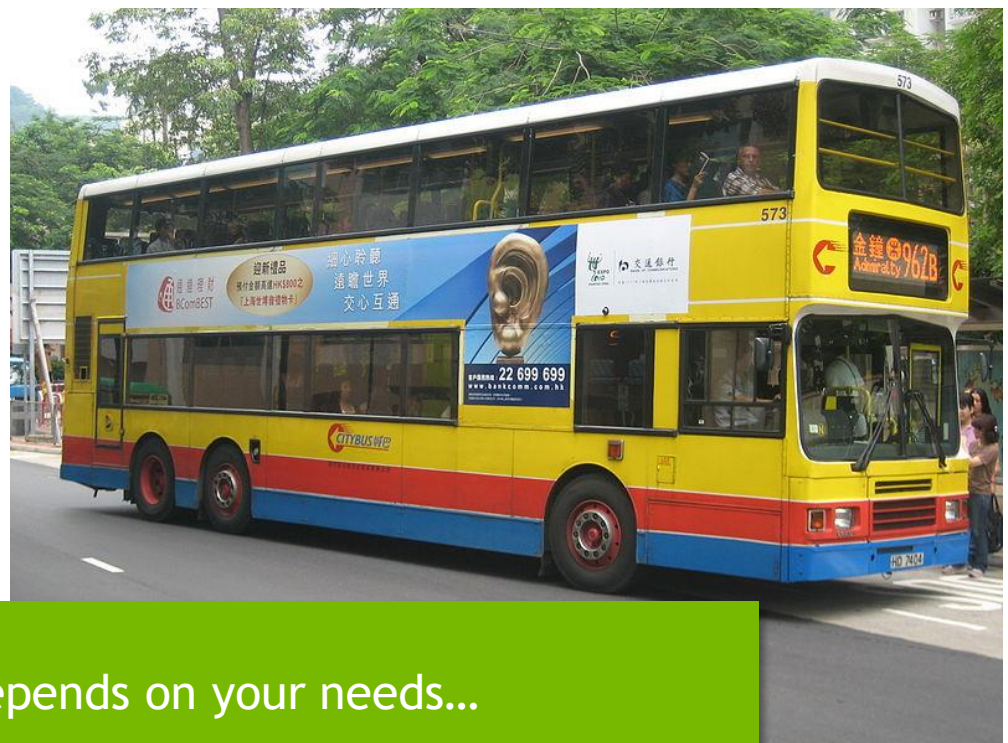


GPU
Optimized for Parallel Tasks

Speed vs Throughput

Speed

Throughput



Which is better depends on your needs…

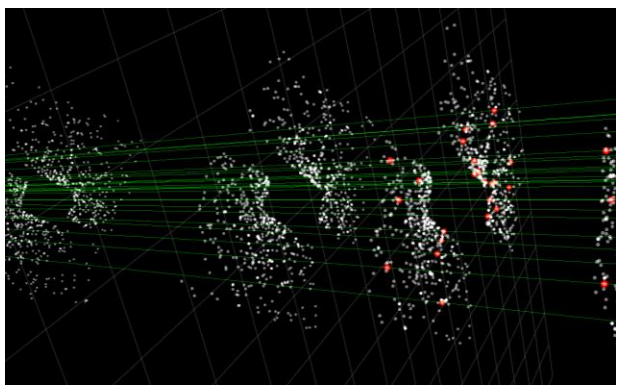
*Images from Wikimedia Commons via Creative Commons

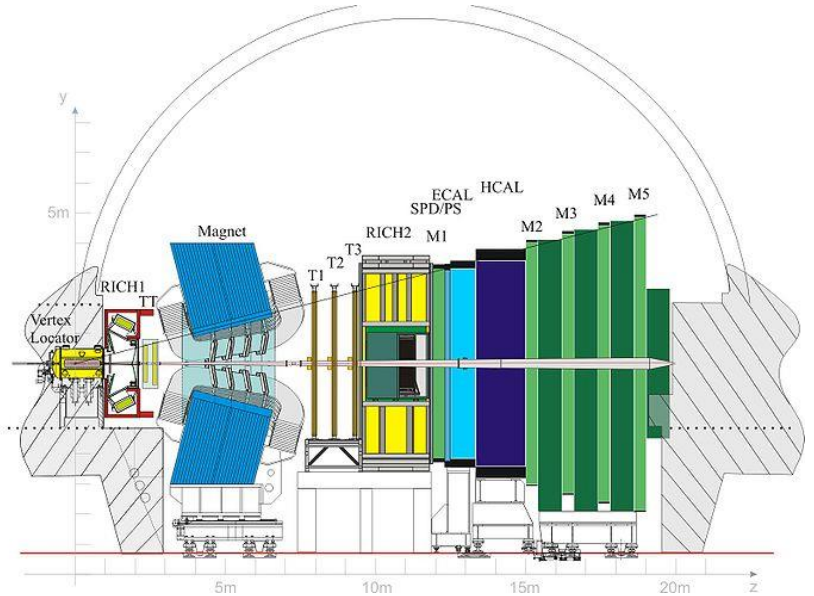
TRACK RECONSTRUCTION

Example - LHCb

- Data rate 5TB/s
- ~30 000 000 independent events/s
- Velo: ~300 hits / layer

~200 track candidates

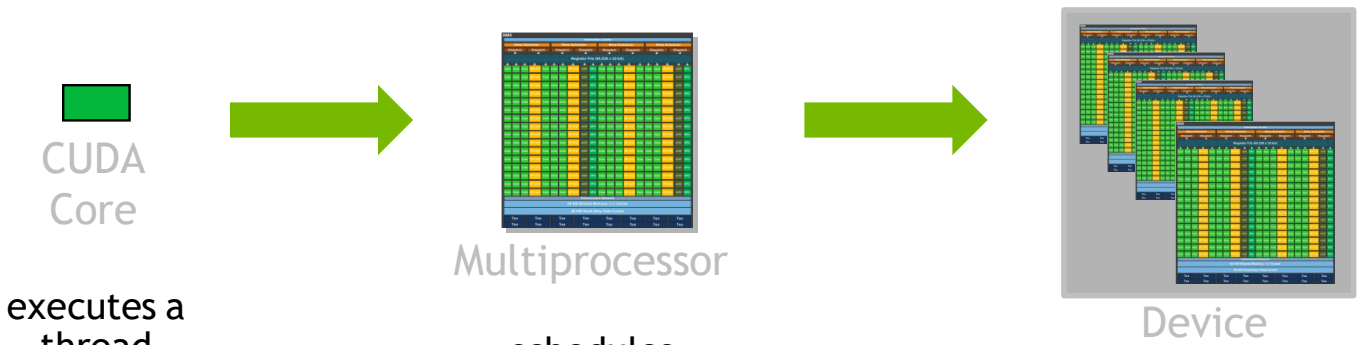




* LHCb collaboration 08 *JINST* **3** S08005

Lots of parallelism
High Throughput needed

THE GPU



TESLA V100

FUSING AI AND HIGH PERFORMANCE COMPUTING

80 Streaming Multiprocessors = 5,120 CUDA cores
15.7 FP32 TFLOPS | 7.8 FP64 TFLOPS | 125 Tensor TFLOPS
16GB / 32GB HBM2 @ 900GB/s | 300GB/s NVLink


HOW?

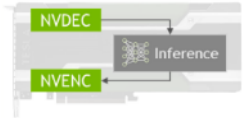
GPU ACCELERATED LIBRARIES

"Drop-in" Acceleration for Your Applications

DEEP LEARNING

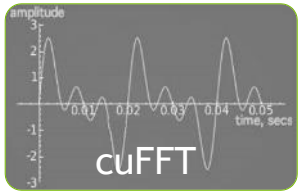
cuDNN


TensorRT



DeepStream SDK

SIGNAL, IMAGE & VIDEO



cuFFT

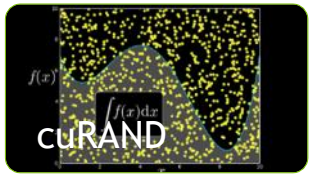

NVIDIA NPP


CODEC SDK

LINEAR ALGEBRA


cuBLAS

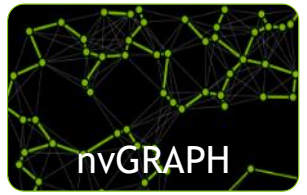

cuSPARSE

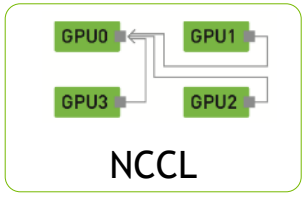

cuRAND


cuSOLVER


CUDA Math library

PARALLEL ALGORITHMS


nvGRAPH



NCCL

PROGRAM GPUS

NVIDIA CUDA

CPU

```
void saxpy_serial(int n,
                  float a,
                  float* x,
                  float* y)
{


  for (int i = 0; i < n; i++)
    y[i] = a * x[i] + y[i];
}


// Perform SAXPY on 1M elements
saxpy_serial(4096 * 256, 2.0, x, y);
```

GPU

```
__global__
void saxpy_parallel(int n,
                    float a,
                    float* x,
                    float* y)
{
  int i = blockIdx.x * blockDim.x +
          threadIdx.x;
  if (i < n) y[i] = a * x[i] + y[i];
}


// Perform SAXPY on 1M elements
saxpy_parallel<<<4096, 256>>>(n, 2.0, x, y);
```

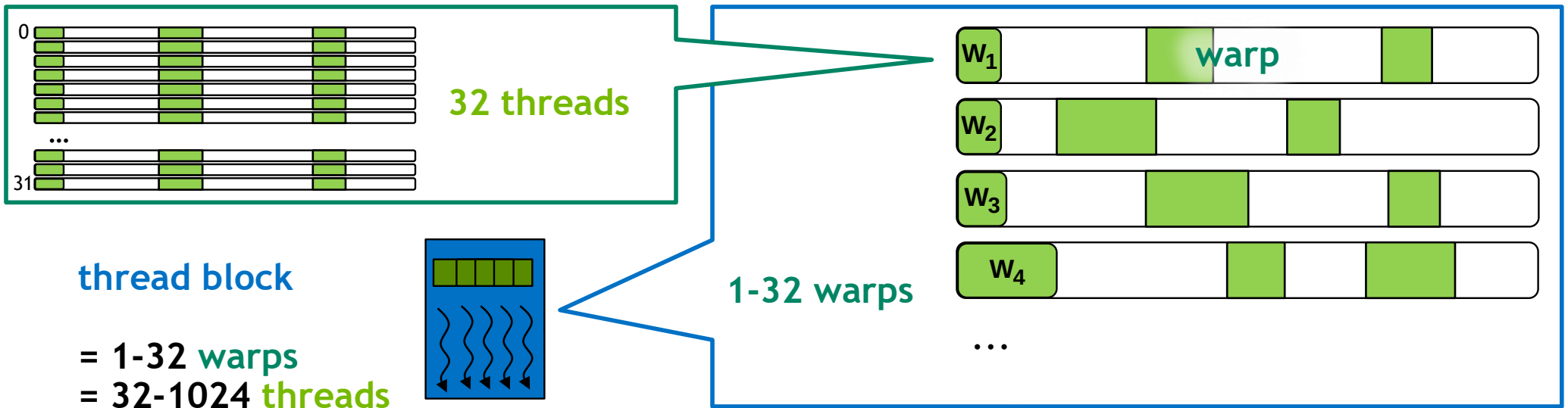
GPU PROGRAMMING FUNDAMENTALS

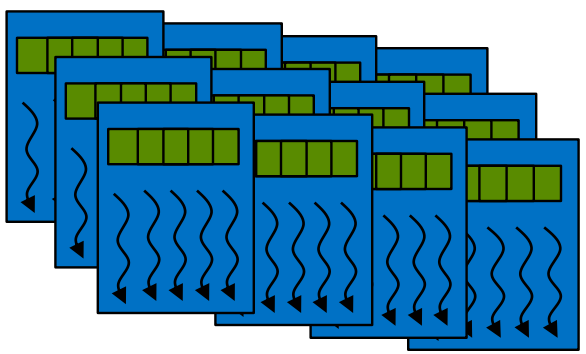
3 Important Rules

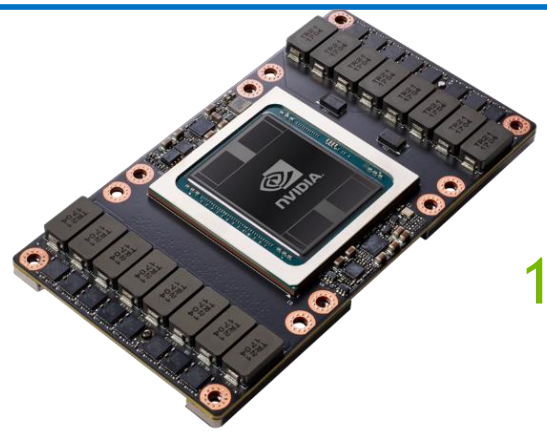
1. Think parallel! Feed 1000s of **threads**
2. Minimize and overlap **CPU<->GPU transfers**
3. GPU-friendly **data-layout**

THOUSANDS OF THREADS

Some Structure







Tesla V100

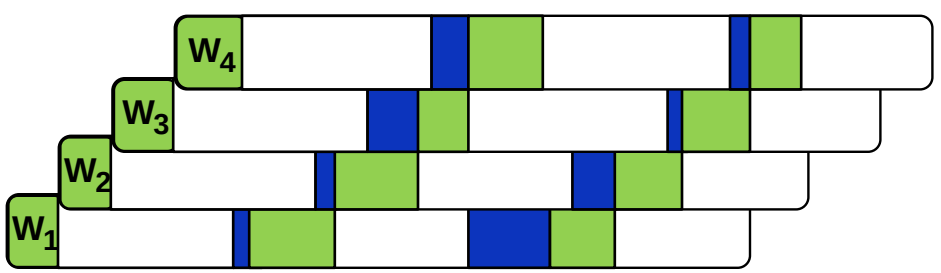
160’000 threads!

LOW LATENCY OF HIGH THROUGHPUT?

CPU architecture must **minimize latency** within each thread

CPU core – Low Latency Processor

GPU architecture **hides latency**
with computation from other threads (warps)



GPU Stream Multiprocessor – High Throughput Processor

Computation Thread/Warp

Processing

Waiting for data

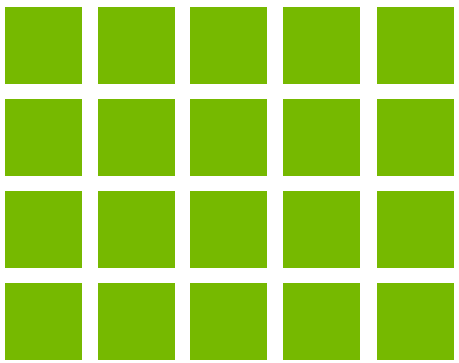
Ready to be processed

Context switch

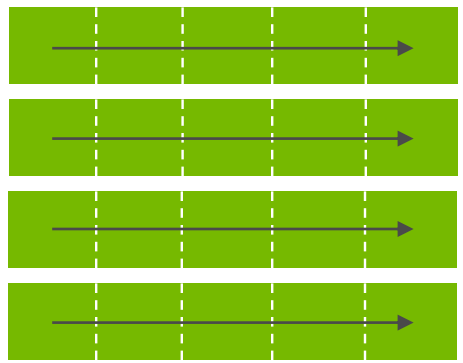
MAXIMIZE THREADS

Must expose enough parallelism to saturate the GPU

GPU threads are slower than CPU threads

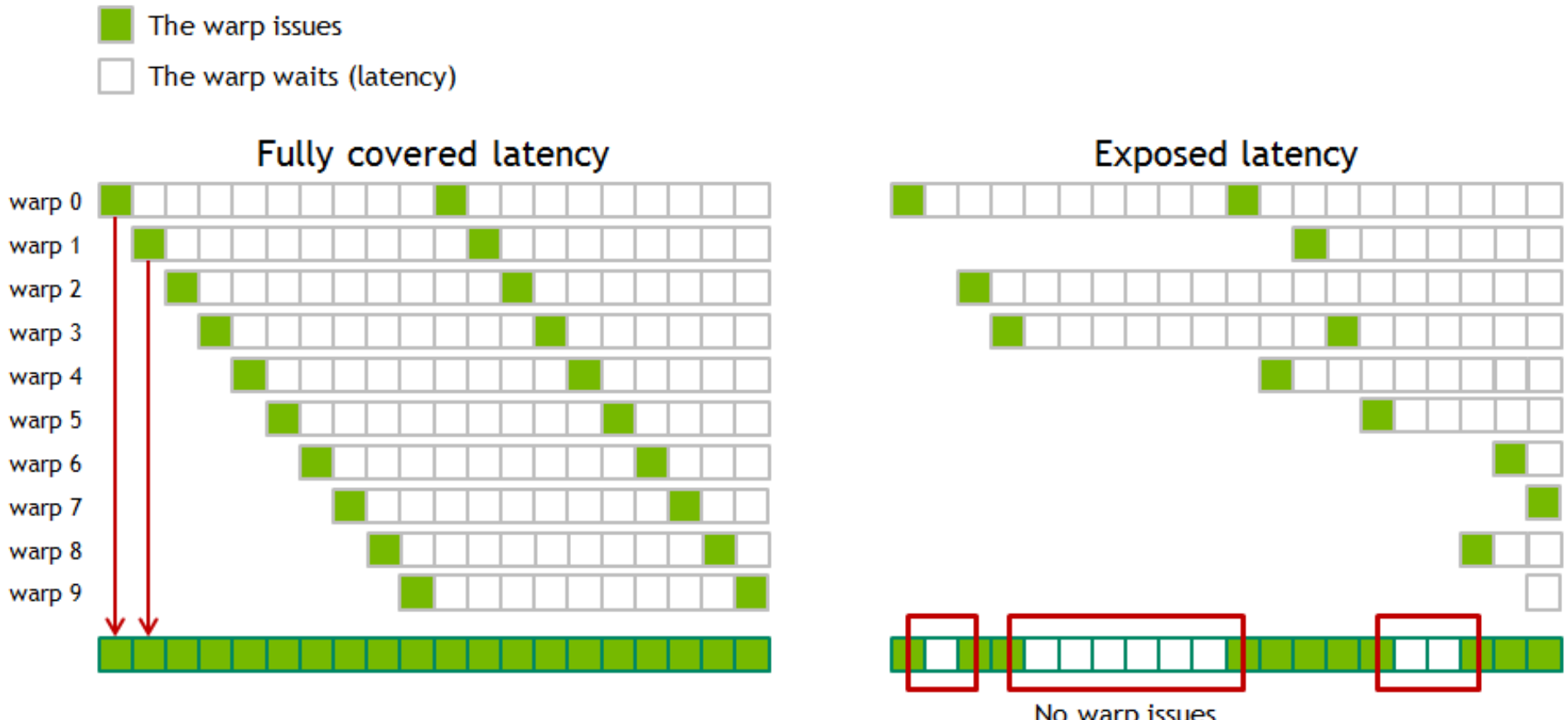


Fine-grained parallelism is good



Coarse-grained parallelism is bad

GPU PROGRAMMING FUNDAMENTALS

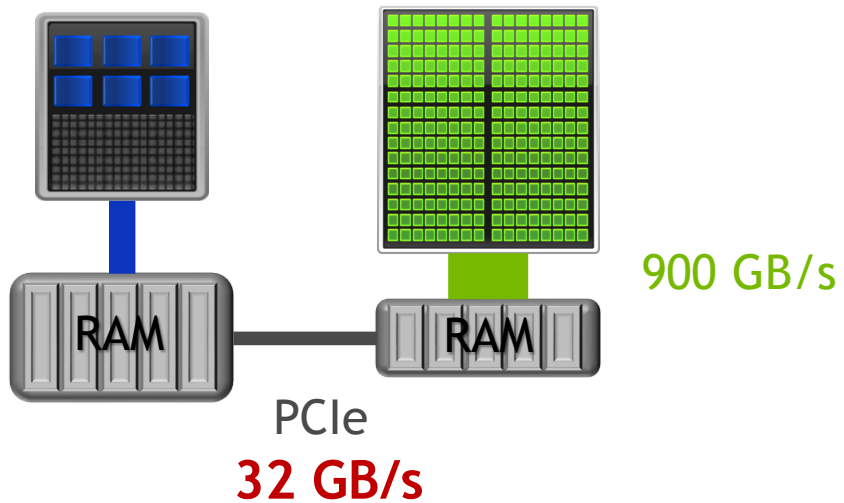


GPU PROGRAMMING FUNDAMENTALS

3 Important Rules

1. Think parallel! Feed 1000s of **threads**
2. Minimize and overlap **CPU<->GPU transfers**
3. GPU-friendly **data-layout**

MINIMIZE MEMORY TRANSFERS



- Transfer only what is necessary
- Keep data on GPU as long as possible
- Use asynchronous memcpys and keep CPUs + GPUs busy.

GPU PROGRAMMING FUNDAMENTALS

3 Important Rules

1. Think parallel! Feed 1000s of **threads**
2. Minimize and overlap **CPU<->GPU transfers**
3. GPU-friendly **data-layout**

MEMORY LAYOUT

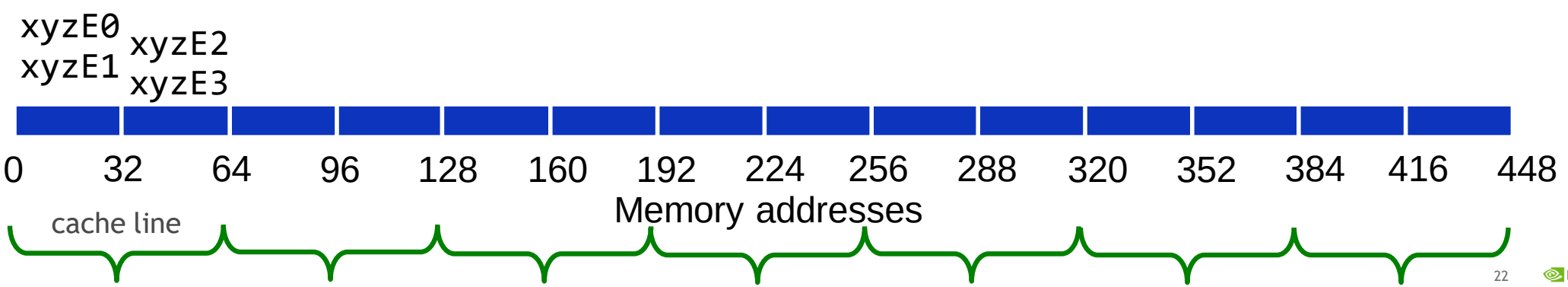
Typical CPU code

CPU Data Layout

```
struct hit {
    float x,y,z,E;
};
array<hit> data;
```

CPU Application Code

```
for(int i=0; i<n; ++i) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```



MEMORY LAYOUT

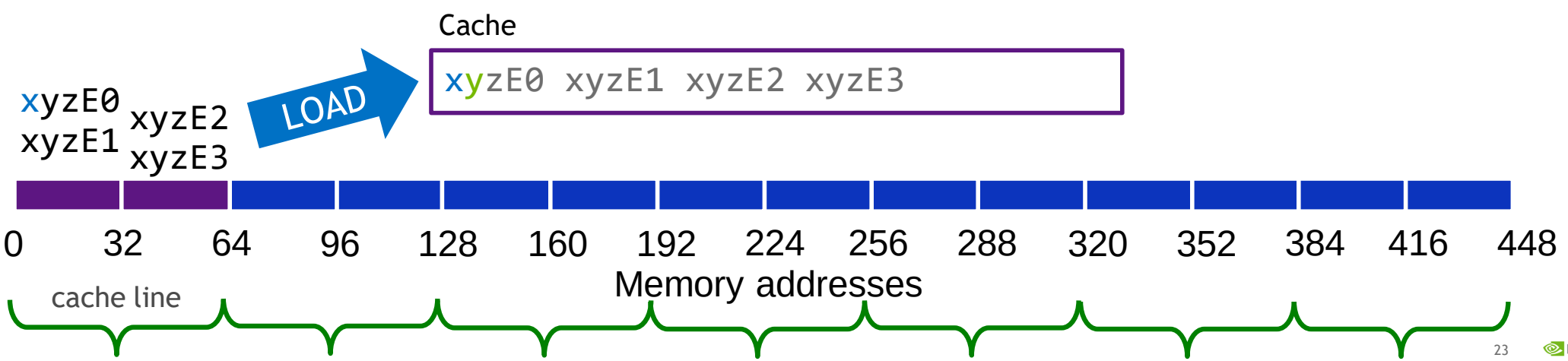
Typical CPU code

CPU Data Layout

```
struct hit {
    float x,y,z,E;
};
array<hit> data;
```

CPU Application Code

```
for(int i=0; i<n; ++i) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```



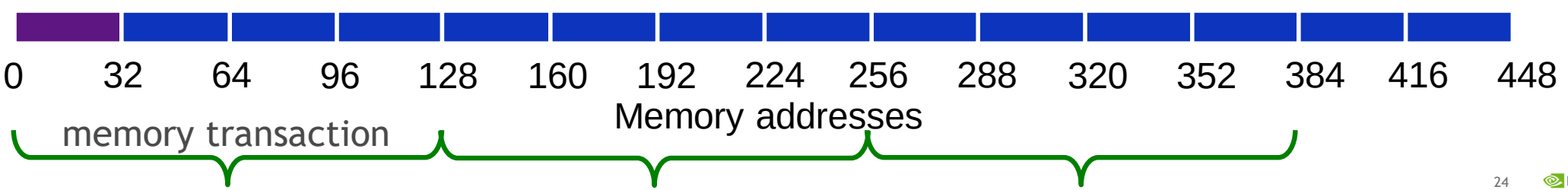
MEMORY LAYOUT

- loads are 128 Byte wide
- (almost) no cache
- loads executed per warp (32 threads)
  - we need values x[0:32] now

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```

32 threads run in sync



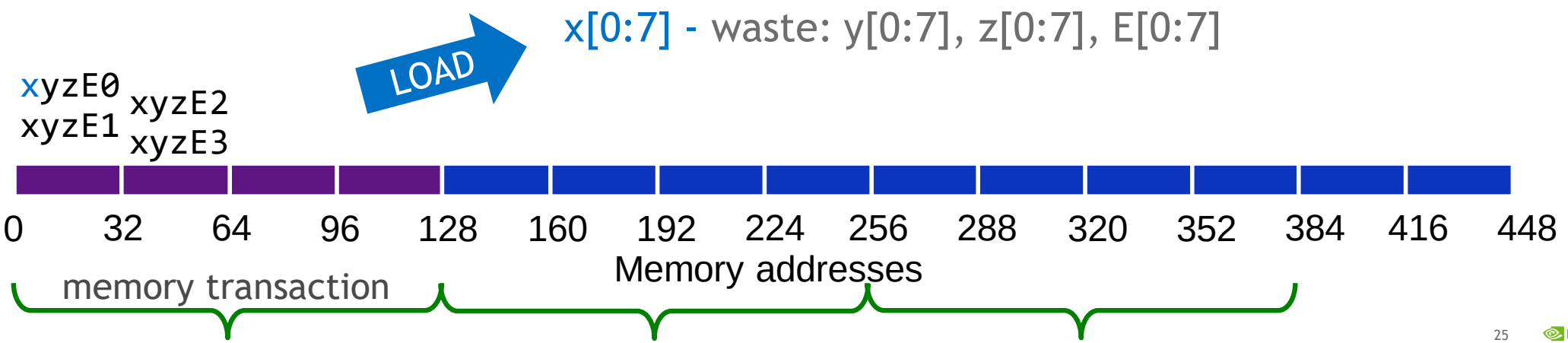
MEMORY LAYOUT

CPU Data Layout

```
struct hit {
    float x,y,z,E;
};
array<hit> data;
```

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```



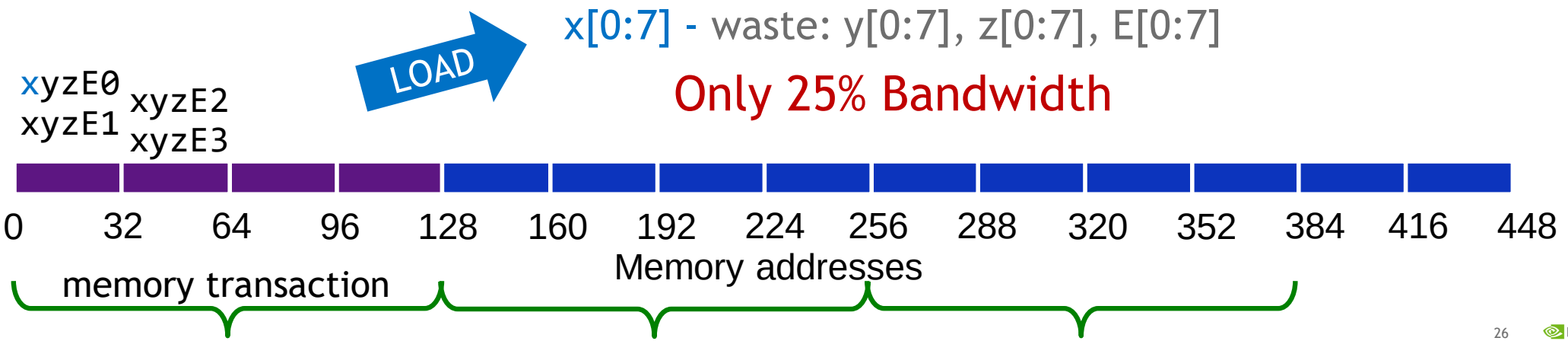
MEMORY LAYOUT

CPU Data Layout

```
struct hit {
    float x,y,z,E;
};
array<hit> data;
```

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```



MEMORY LAYOUT

… GPU friendly

CPU Data Layout

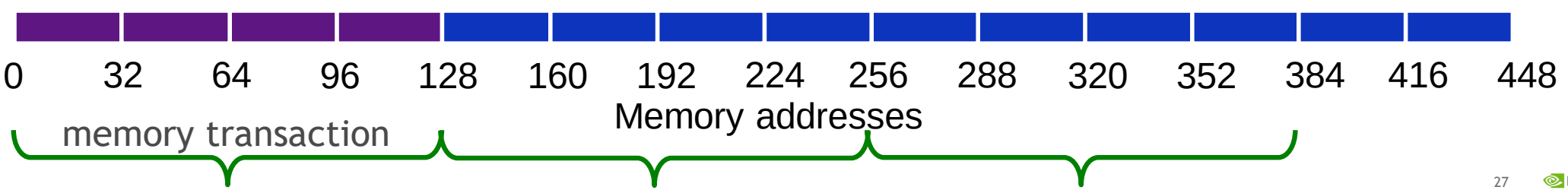
```
struct hit {
    float x,y,z,E;
};
array<hit> data;
```

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
    norm2 = x[i]*x[i]
          + y[i]*y[i] + …
```

GPU Data Layout

```
struct hit_cloud {
    array<float> x; array<float> y;
    array<float> z; array<float> E;
};
hit_cloud data;
```



MEMORY LAYOUT

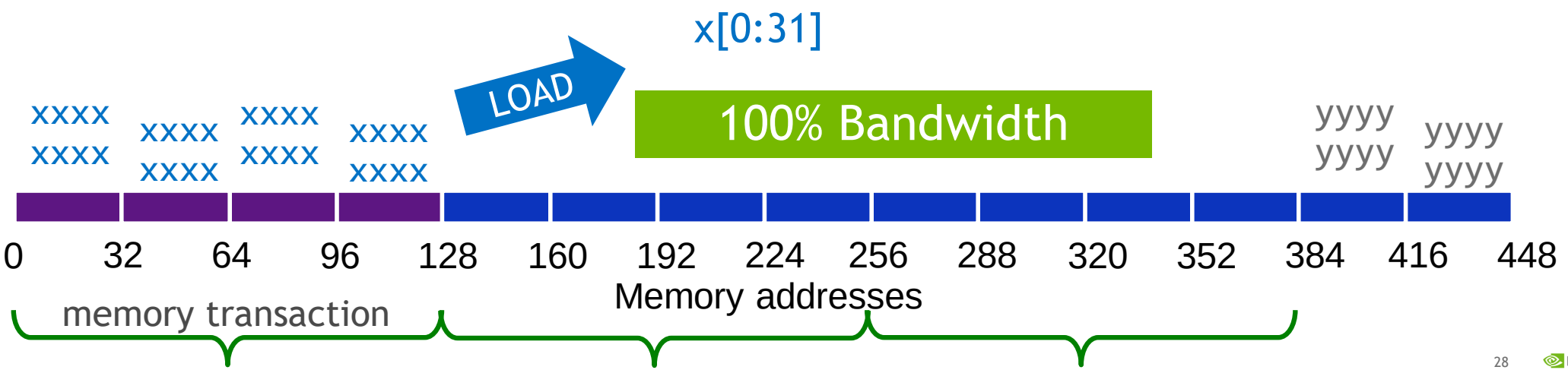
… GPU friendly

GPU Data Layout

```
struct hit_cloud {
    array<float> x; array<float> y;
    array<float> z; array<float> E;
};
hit_cloud data;
```

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```



MEMORY LAYOUT

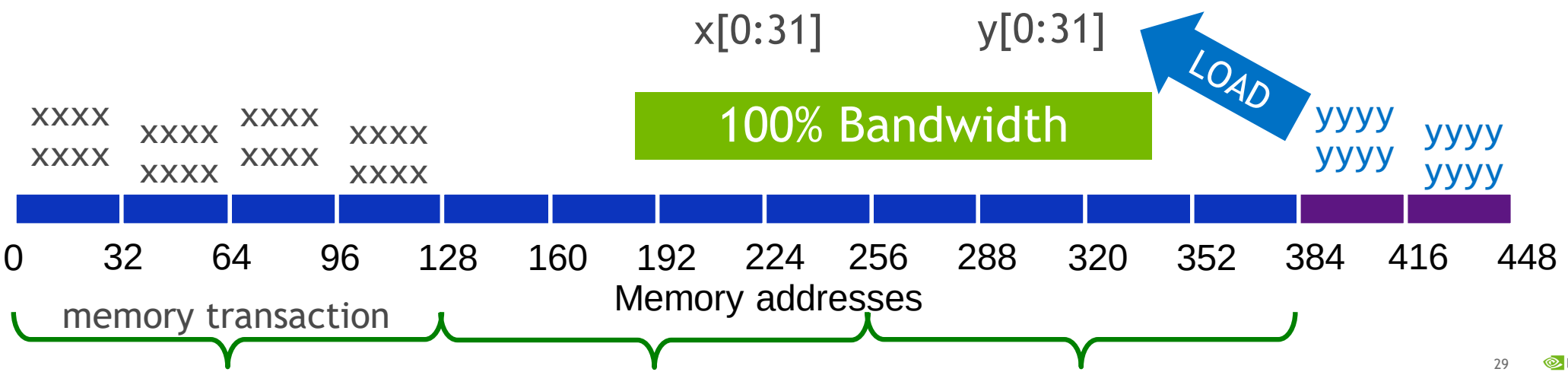
… GPU friendly

GPU Data Layout

```
struct hit_cloud {
    array<float> x; array<float> y;
    array<float> z; array<float> E;
};
hit_cloud data;
```

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
  float norm2 = x[i]*x[i]
                + y[i]*y[i] + …
}
```



MEMORY LAYOUT

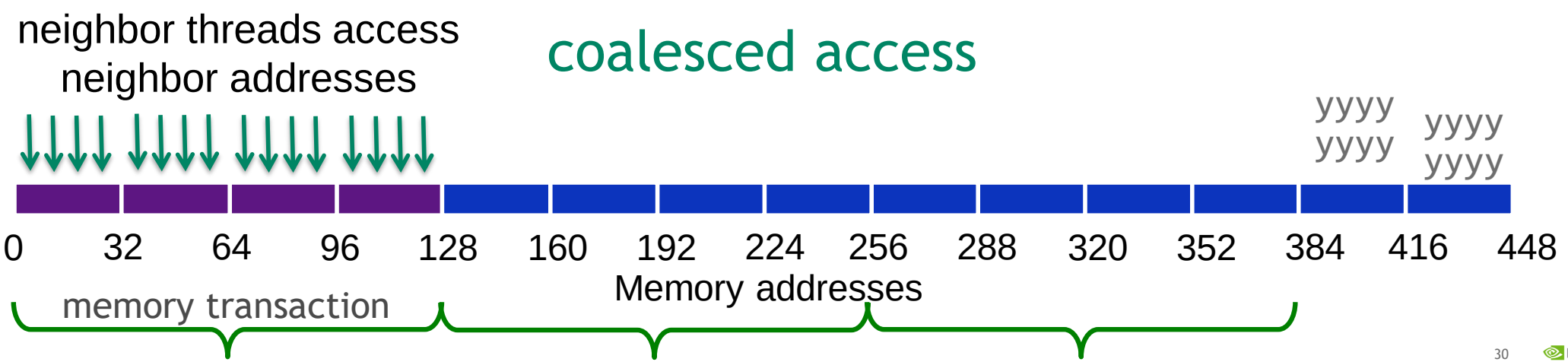
… GPU friendly

GPU Data Layout

```
struct hit_cloud {
    array<float> x; array<float> y;
    array<float> z; array<float> E;
};
hit_cloud data;
```

GPU Application Code

```
int i = threadIdx.x;
if(i < n) {
  float norm2 = x[i]*x[i]
              + y[i]*y[i] + …
}
```



MEMORY LAYOUT

… GPU friendly

CPU Data Layout

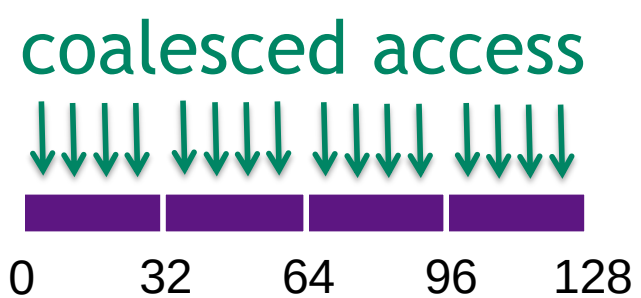
```
struct hit {
    float x,y,z,E;
};
array<hit> data;
```

Array of Structures (AoS)

GPU Data Layout

```
struct hit_cloud {
    array<float> x; array<float> y;
    array<float> z; array<float> E;
};
hit_cloud data;
```

Structure of Arrays (SoA)



GPU PROGRAMMING FUNDAMENTALS

3 Important Rules

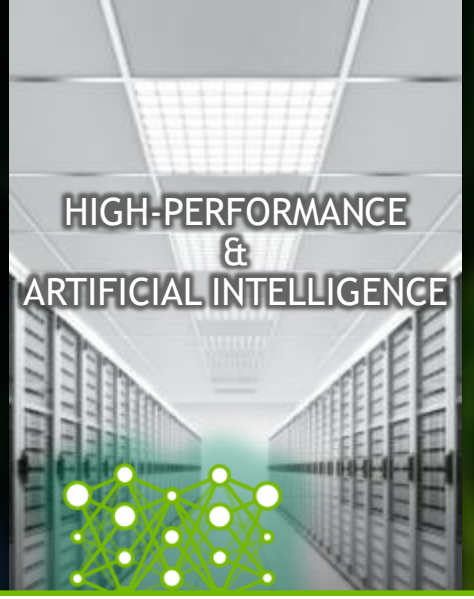
1. Think parallel! Feed 1000s of **threads**
2. Minimize and overlap **CPU<->GPU transfers**
3. GPU-friendly **data-layout**


DRIVING
DEVELOPMENTS


GAMING
&
GRAPHICS

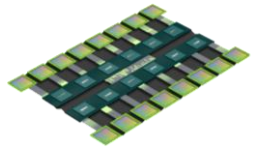

HIGH-PERFORMANCE
&
ARTIFICIAL INTELLIGENCE


AUTONOMOUS
&
EMBEDDED



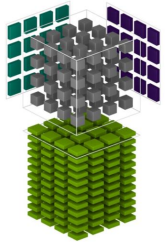
NVLink

NVSwitch



RT CORE

TENSOR CORE

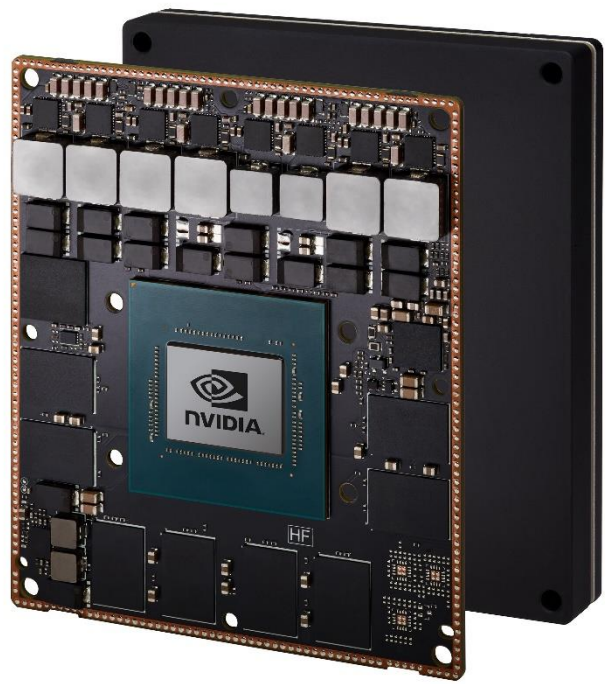


XAVIER SOC

XAVIER SOC

System on a Chip for Autonomous Machines

	JETSON AGX XAVIER
GPU	512 Core Volta @ 1.37GHz 64 Tensor Cores
CPU	8 core Carmel ARM CPU @ 2.26GHz (4x) 2MB L2 + 4MB L3
DL Accelerator	(2x) NVDLA
Vision Accelerator	(2x) 7-way VLIW Processor
Memory	16GB 256-bit LPDDR4x @ 2133MHz 137 GB/s
Storage	32GB eMMC
Video Encode	(4x) 4Kp60 / (8x) 4Kp30 HEVC
Video Decode	(2x) 8Kp30 / (6x) 4Kp60 12-bit support
Camera	16 lanes MIPI CSI-2 | 8 lanes SLVS-EC D-PHY 40Gbps / C-PHY 109Gbps
PCI Express	16 lanes PCIe Gen 4 1x8 + 1x4 + 1x2 + 2x1
Mechanical	100mm x 87mm 699 pin connector
Power	10W / 15W / 30W



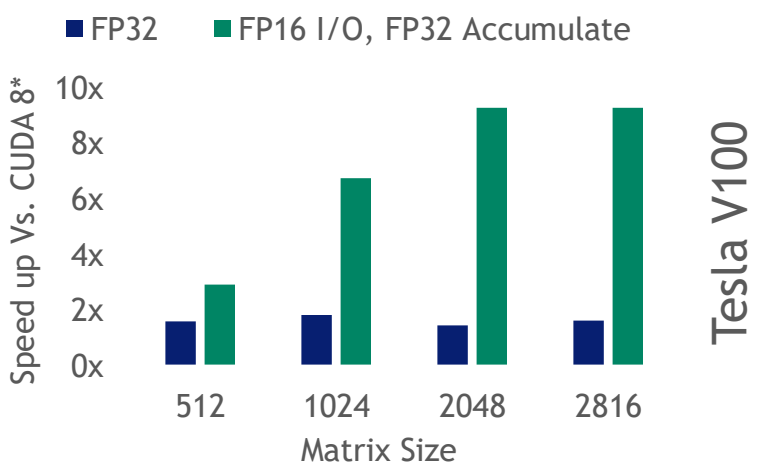
TENSOR CORES

Mixed-precision matrix-matrix multiplication

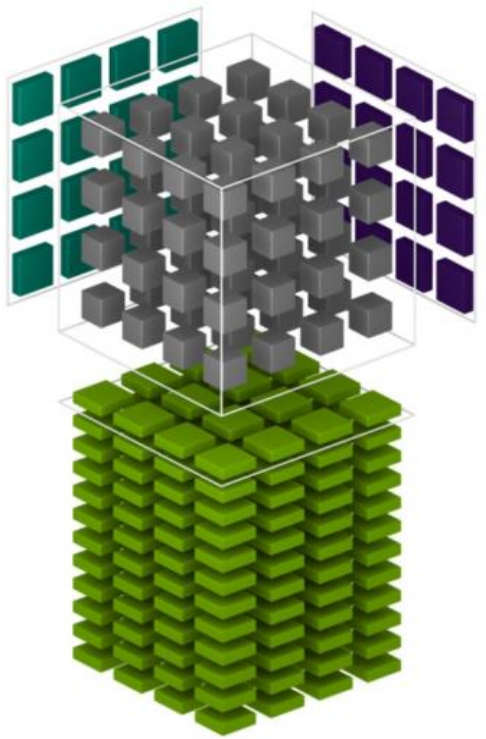
$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 | FP16 | FP16 | FP16 or FP32

5x - 9x faster GEMM operations



Tesla V100



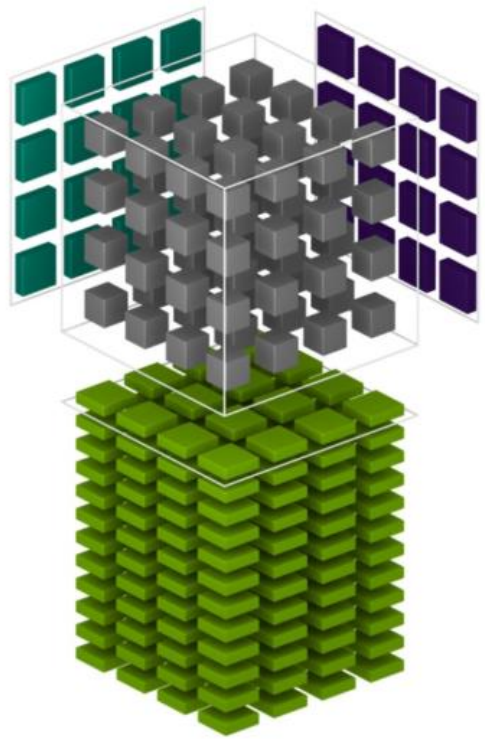
NVIDIA

TENSOR CORES

Mixed-precision matrix-matrix multiplication

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 — FP16 — FP16 — FP16 or FP32

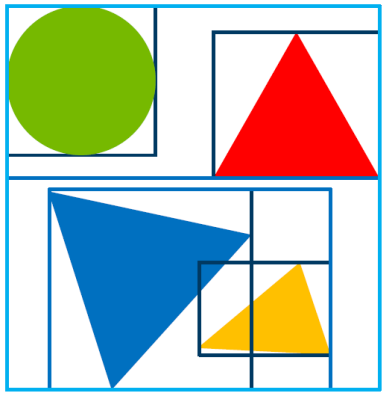


GPU	SMs	Total	Peak Half FLOPS	PEAK INT8 OPS	PEAK INT4 OPS	PEAK Binary OPS
V100	80	640	125 TFLOPS	N.A.	N.A.	N.A.
TU102	72	576	130.5 TFLOPS	261 TOPS	522 TOPS	2088 TOPS

Used via libraries, exposed in CUDA 10

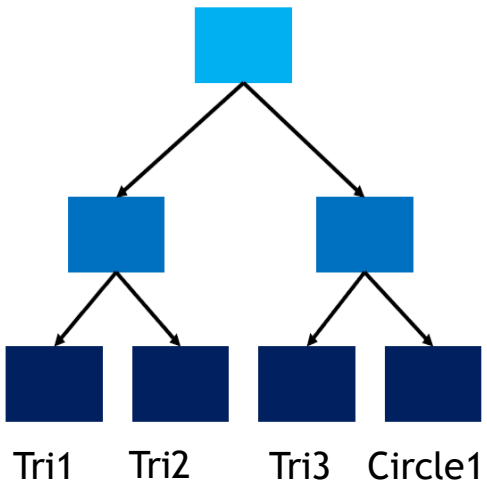
RT CORES

Ray-Tracing Acceleration



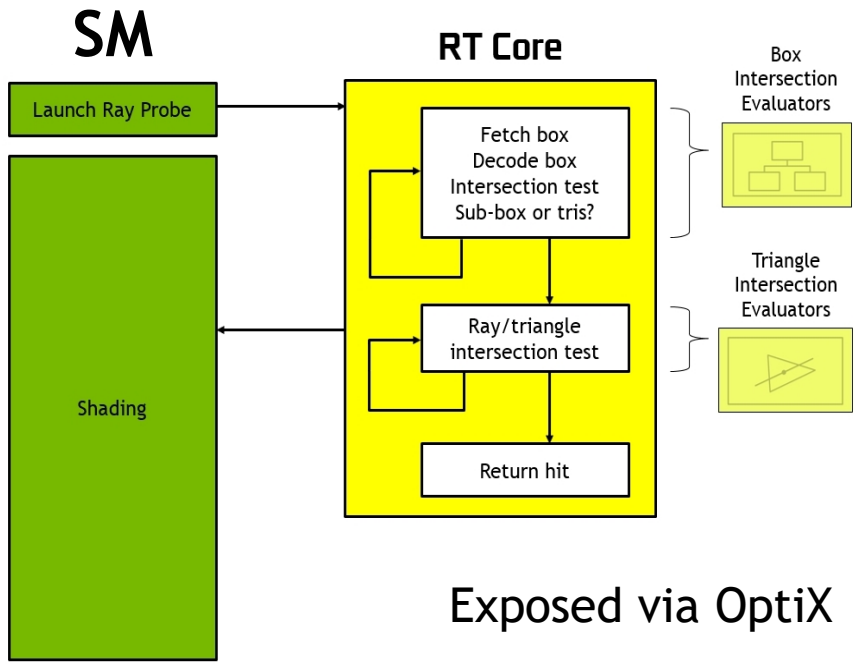
RT Cores perform

- Ray-Bounding Volume Hierarchy Traversal
- Ray-Triangle Intersection



Return to SM for

- Multi-level Instancing
- Custom Intersection
- Shading



Exposed via OptiX

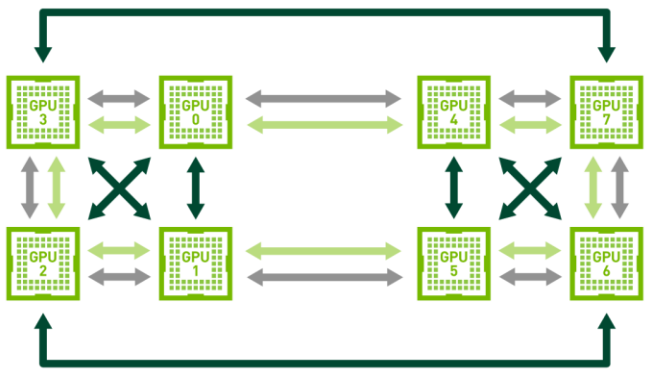
NVLINK & NVSWITCH

Fast GPU Interconnect

NVLink

High-Speed Interconnect

- 25 GB/sec, each direction, per Link
- V100: 6 Links = 300GB/s
- Link to POWER9



NVSwitch

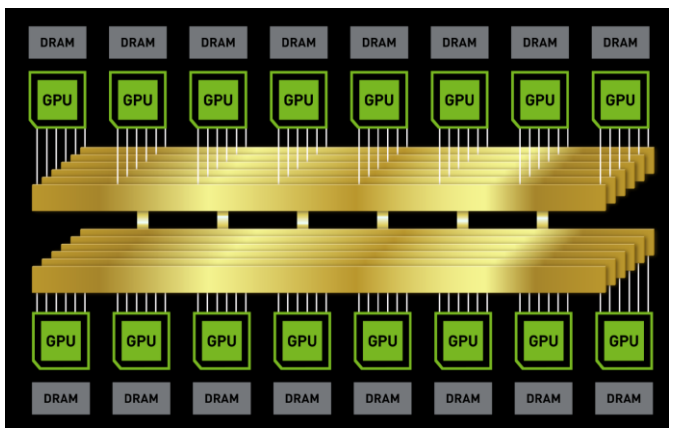
World's highest Bandwidth on-node switch

- 900 GB/sec
- 18 NVLink ports | 50GB/s per port bi-directional

Enables The World's Largest GPU

- 16 Tesla V100 32GB
  - 2 petaFLOPS
  - Unified 512GB HBM2 GPU Memory Space

NVIDIA DGX-2



CONCLUSION

- GPUs are high-throughput processors
- Programming GPUs requires some rethinking (parallel, transfers, data layout)

- Track Reconstruction should be well suited for GPUs

- New features for Graphics, AI, Autonomous Vehicles

  → Explore how to exploit them for reconstruction


NVIDIA