GPU BOOTCAMP

WELCOME

Agenda

- Introduction to GPU Computing
- N-Ways to GPU Programming
- Demo: OpenACC

INTRODUCTION TO GPU COMPUTING

What to expect?

- Broad view on GPU Stack
- Fundamentals of GPU Architecture
- Ways to GPU Computing
- Good starting point

WHAT IS PARALLEL PROGRAMMING?

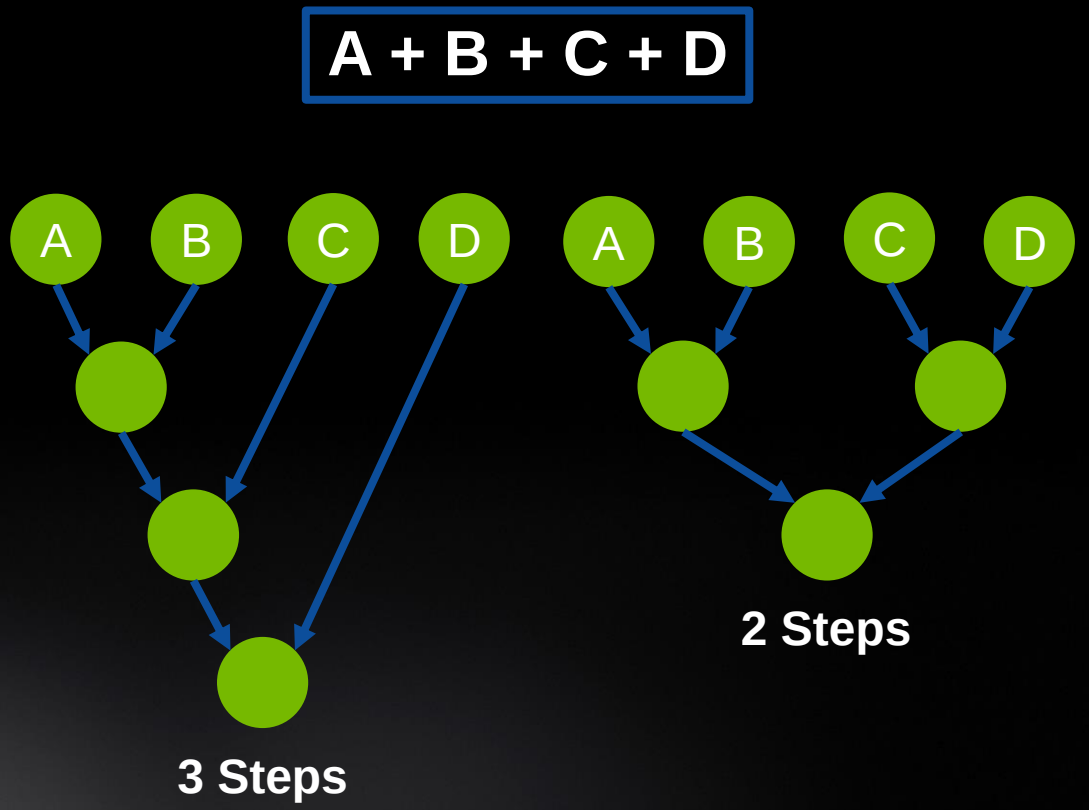
“Performance Programming”

Parallel programming involves exposing an algorithm’s ability to execute in parallel

This may involve breaking a large operation into smaller tasks (task parallelism)

Or doing the same operation on multiple data elements (data parallelism)

Parallel execution enables better performance on modern hardware



WHAT IS PARALLEL PROGRAMMING?

A real world example

A professor and his 3 teaching assistants (TA) are grading 1,000 student exams

This exam has 8 questions on it

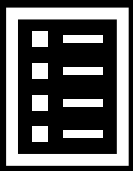
Let’s assume it takes 1 minute to grade 1 question on 1 exam

To maintain fairness, if someone grades a question (for example, question #1) then they must grade that question on all other exams

The following is a sequential version of exam grading

Prof

TA



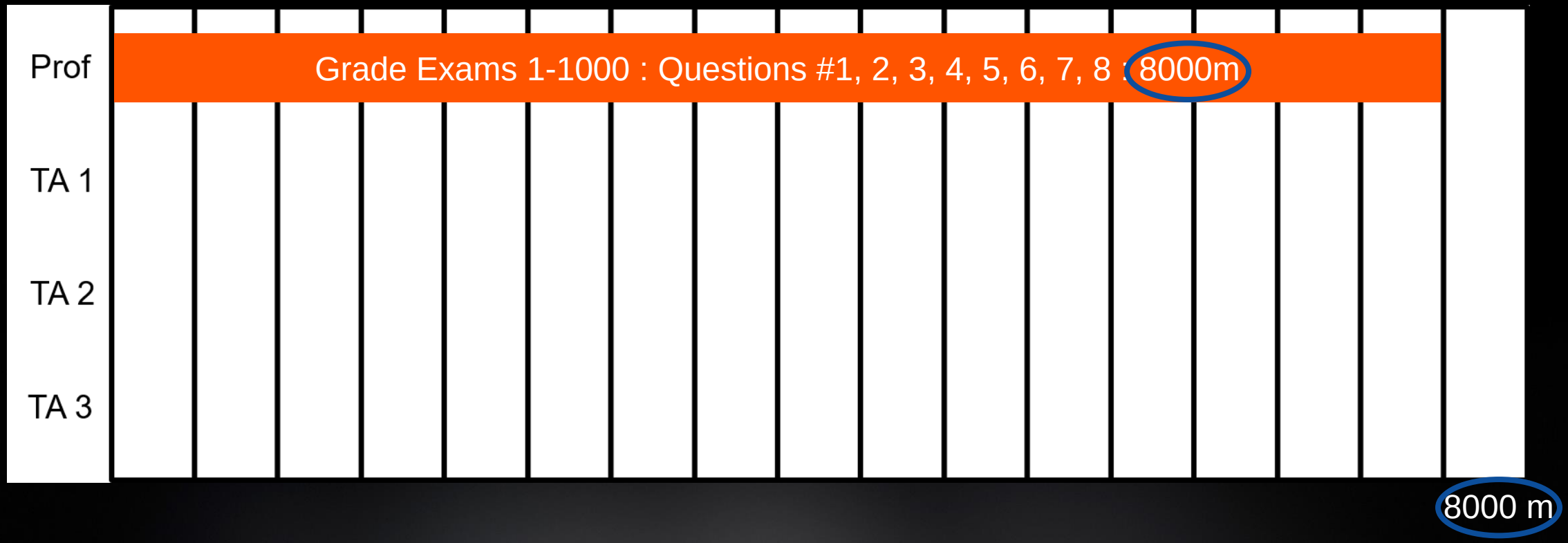
x1000

8 questions per exam

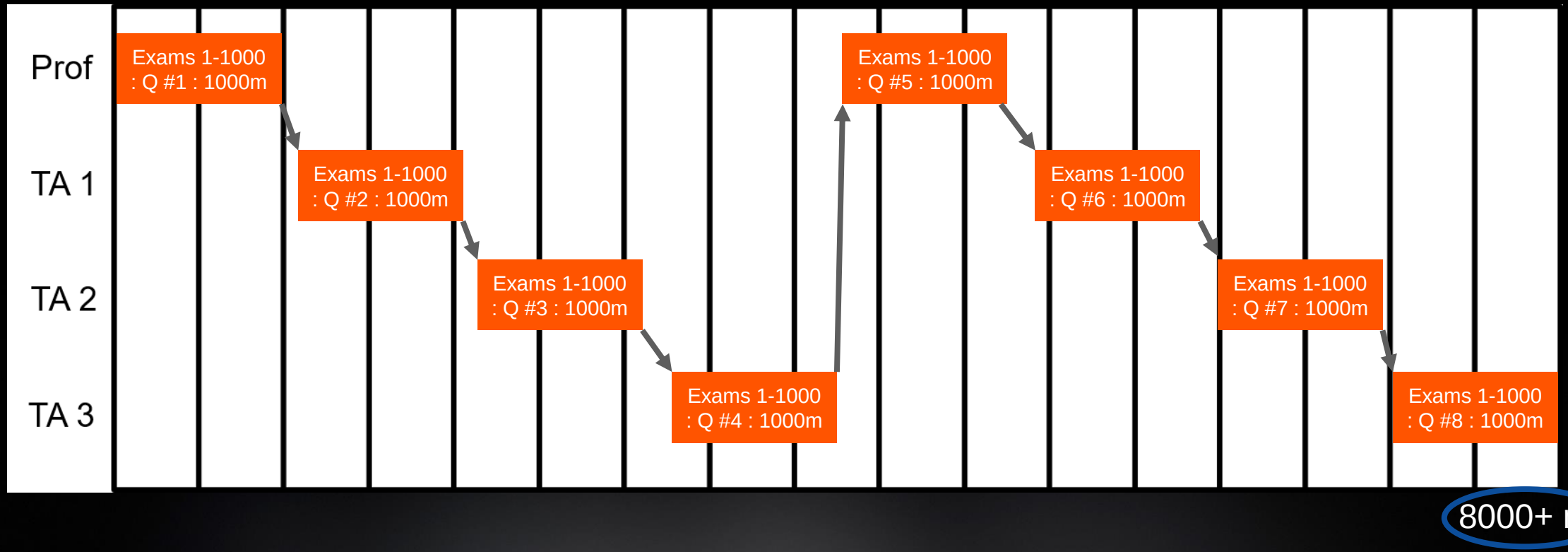
8,000 questions in total

1 minute per question

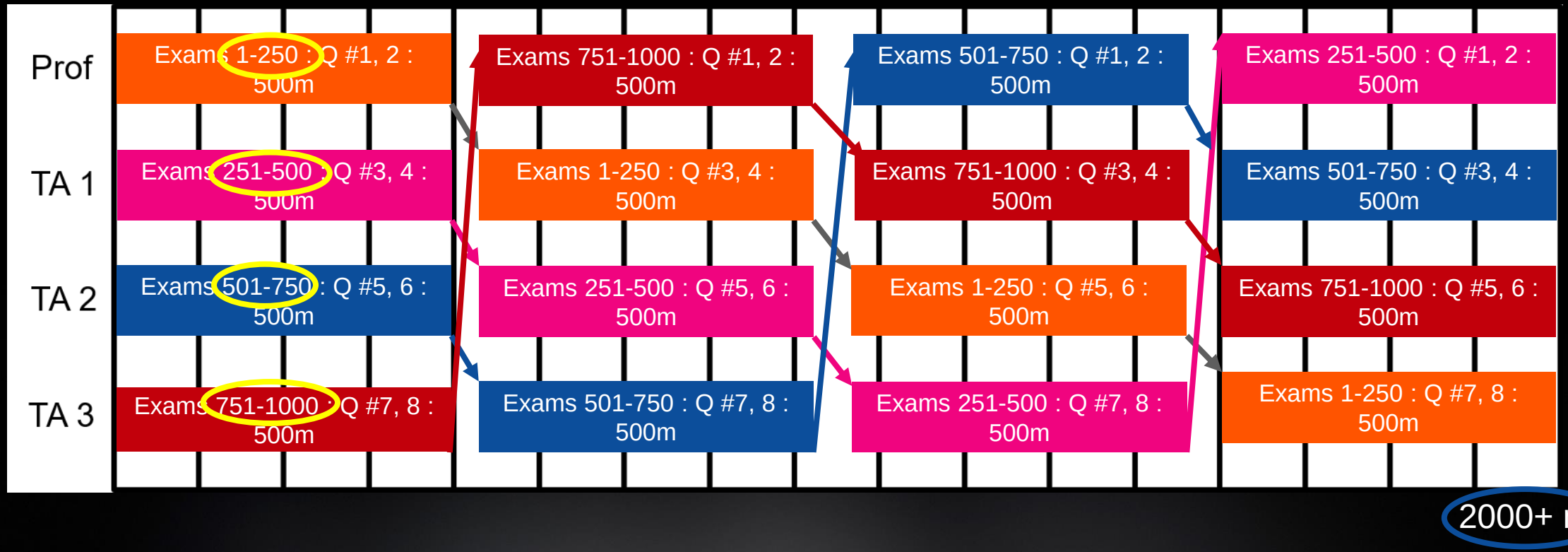
SEQUENTIAL SOLUTION



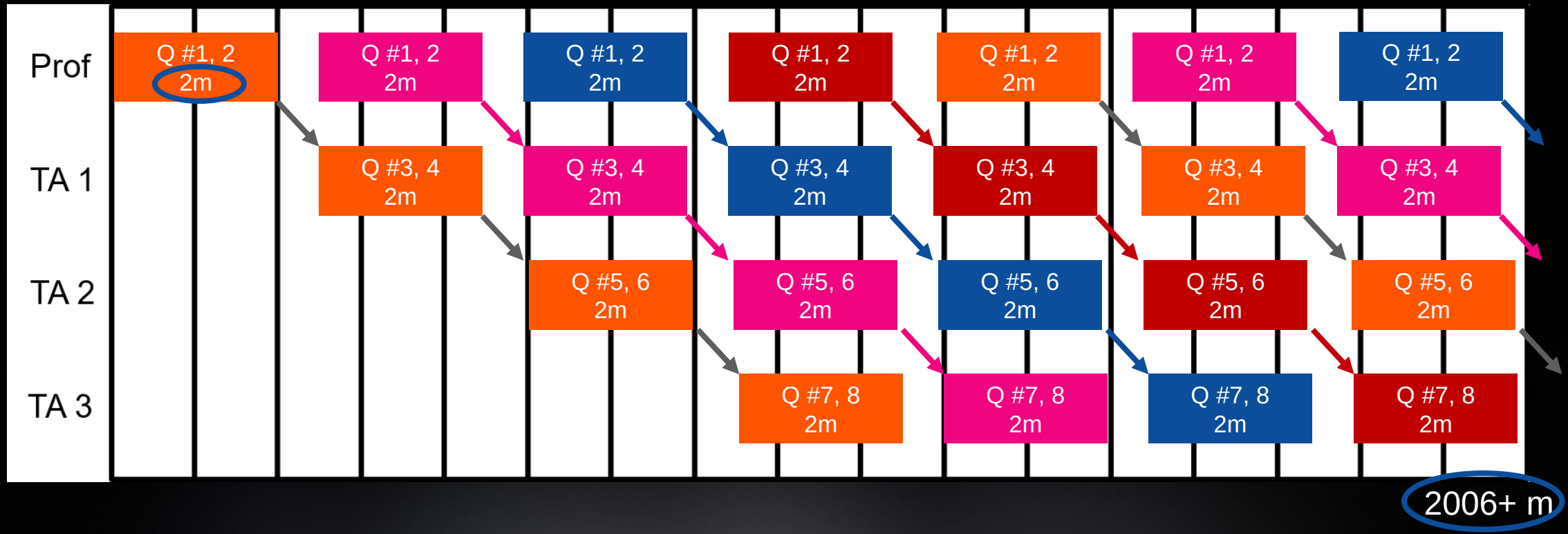
SEQUENTIAL SOLUTION



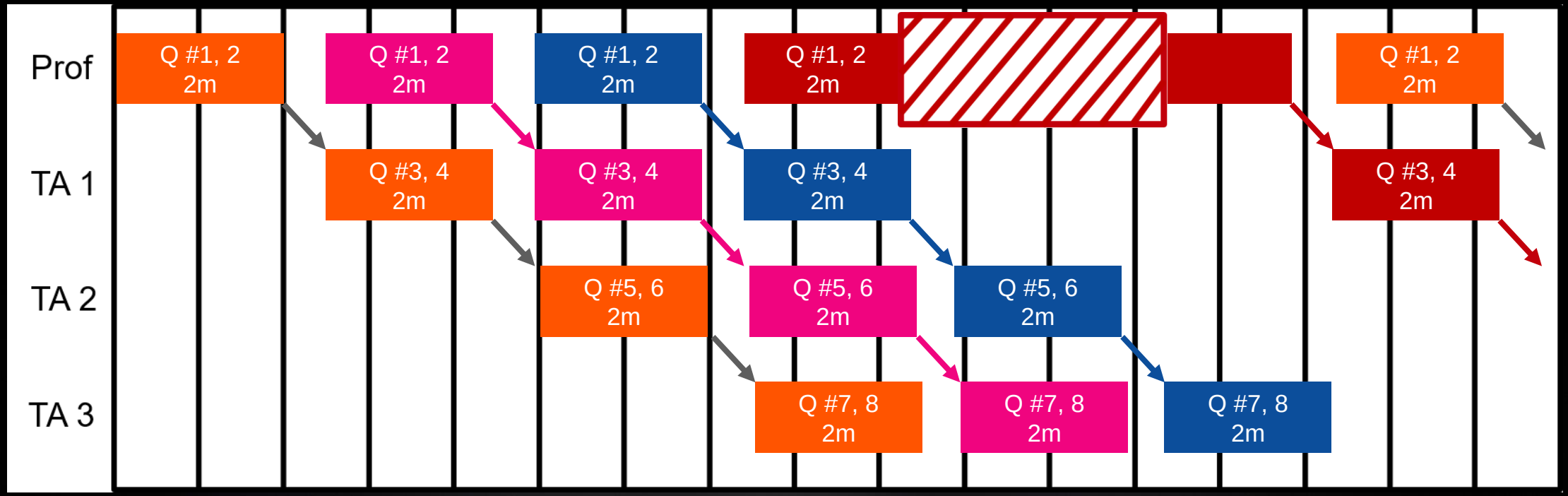
PARALLEL SOLUTION



PIPELINE



PIPELINE STALL



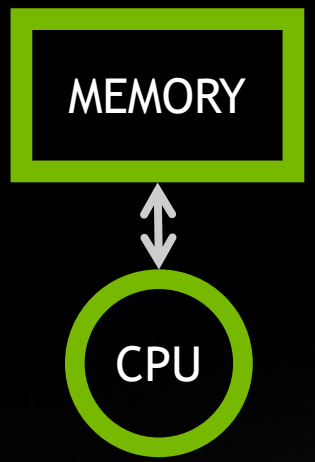
STATE OF THE ART 2019

CORAL Summit System
5-10x Faster
1/5th the Nodes,
Same Energy Use as Titan

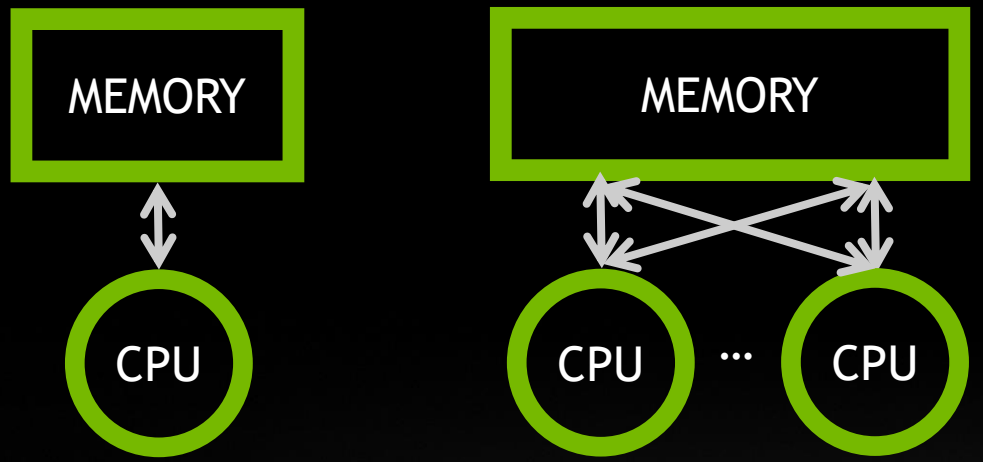
Earth Simulator
40.96 TF
Top500 #1 for 3 years

Just 1 Node in Summit
Is the same performance as the
Earth Simulator in 2002

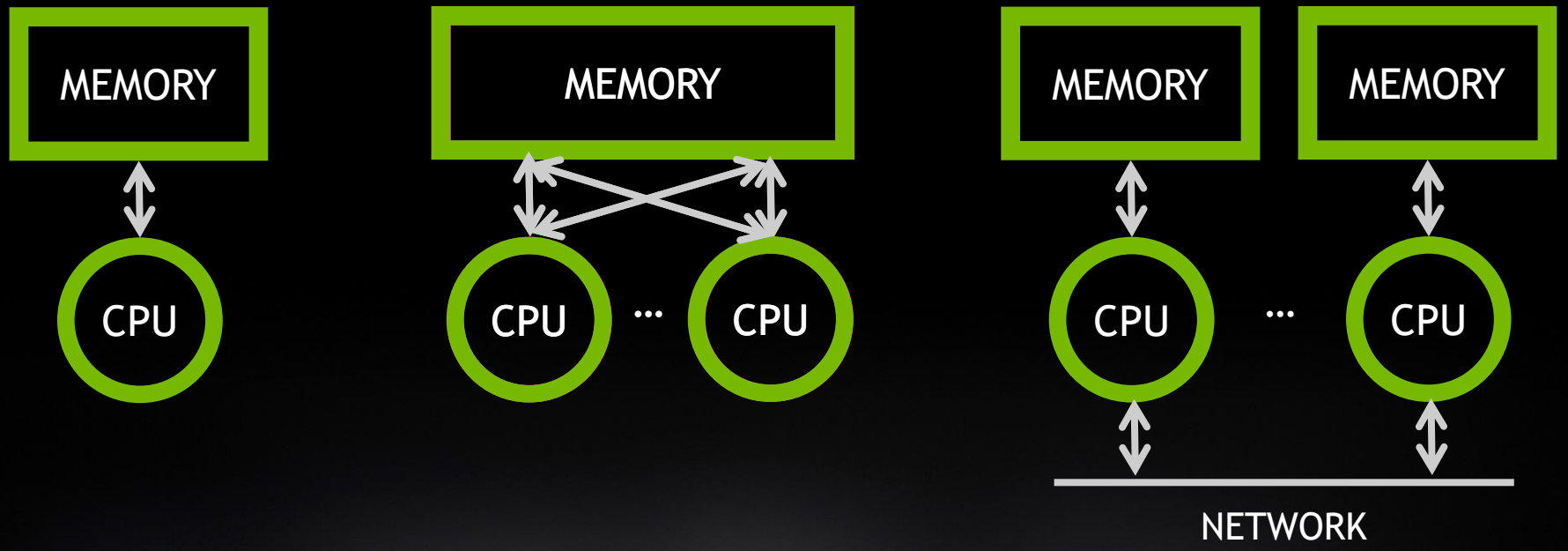
HPC SYSTEM EVOLUTION



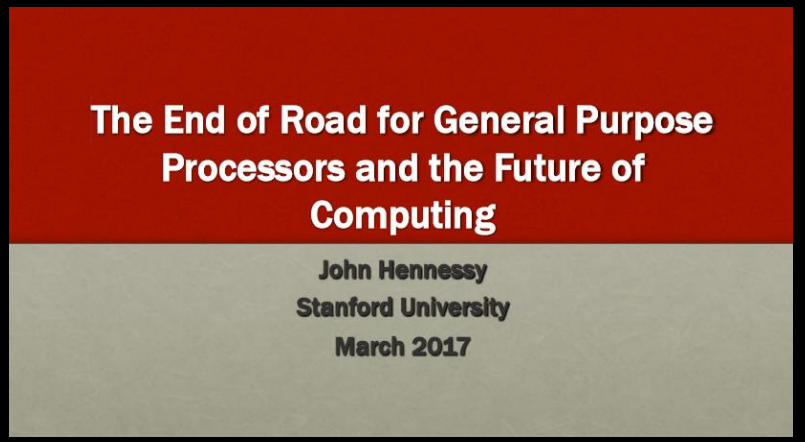
HPC SYSTEM EVOLUTION

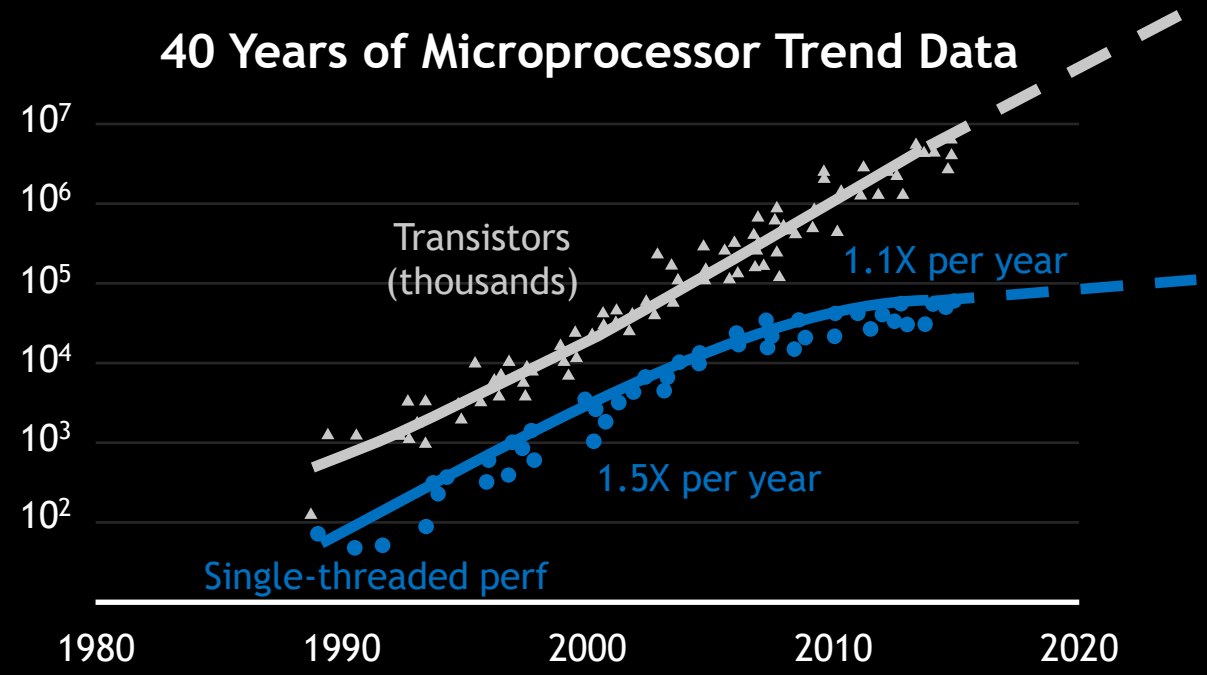


HPC SYSTEM EVOLUTION



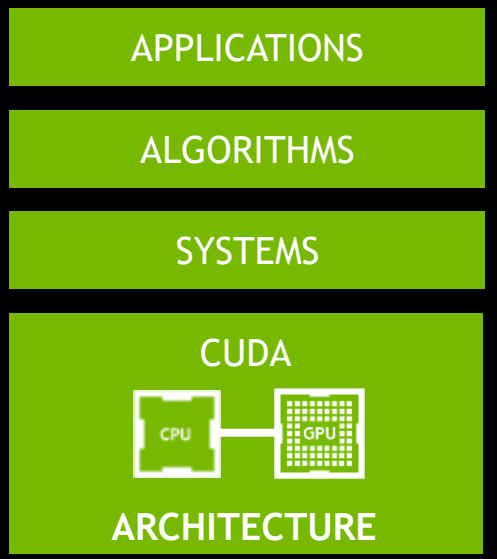
LIFE AFTER MOORE'S LAW

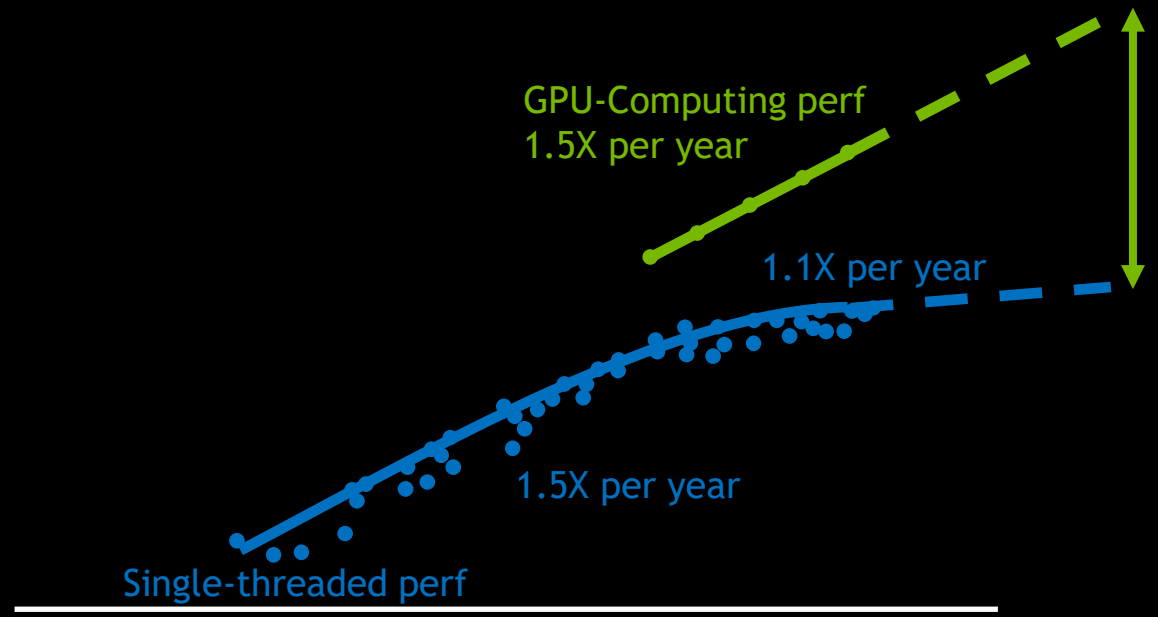




Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

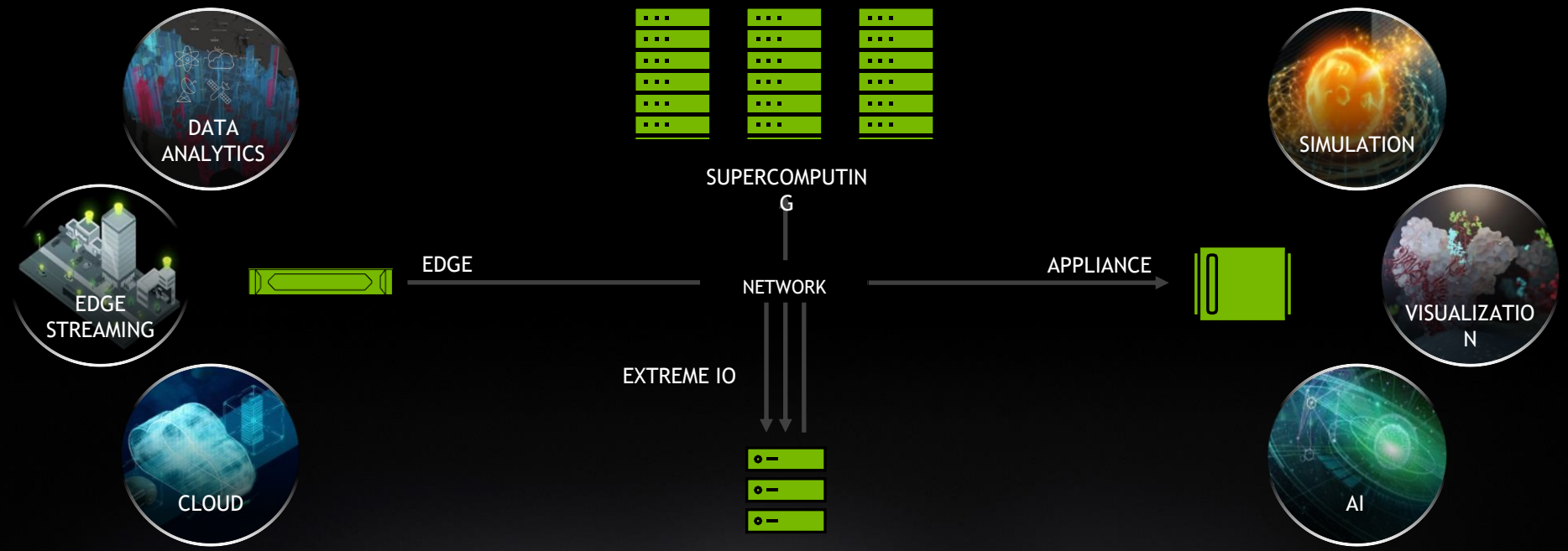
RISE OF GPU COMPUTING





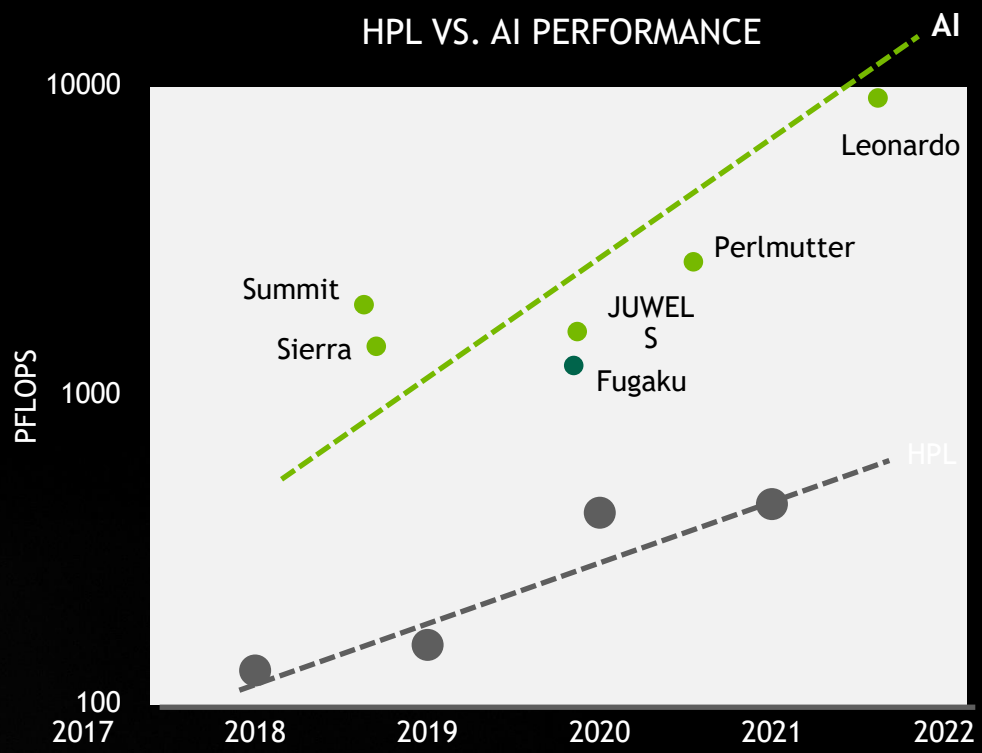
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

EXPANDING UNIVERSE OF SCIENTIFIC COMPUTING



THE ERA OF EXASCALE AI SUPERCOMPUTING

AI is the New Growth Driver for Modern HPC

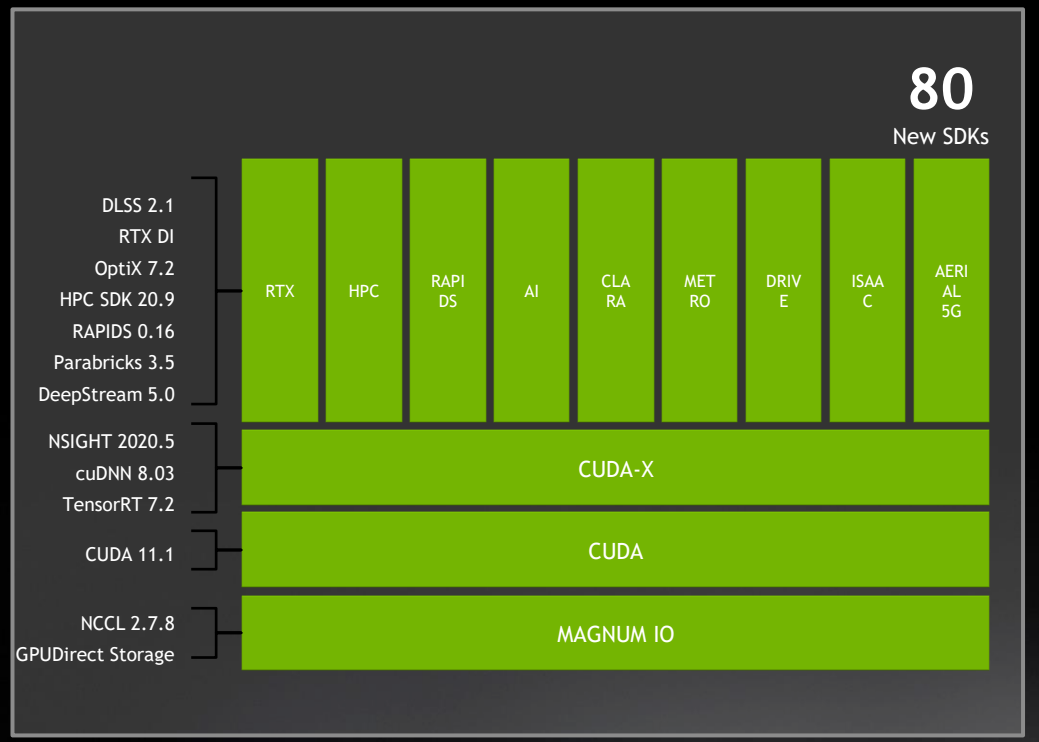


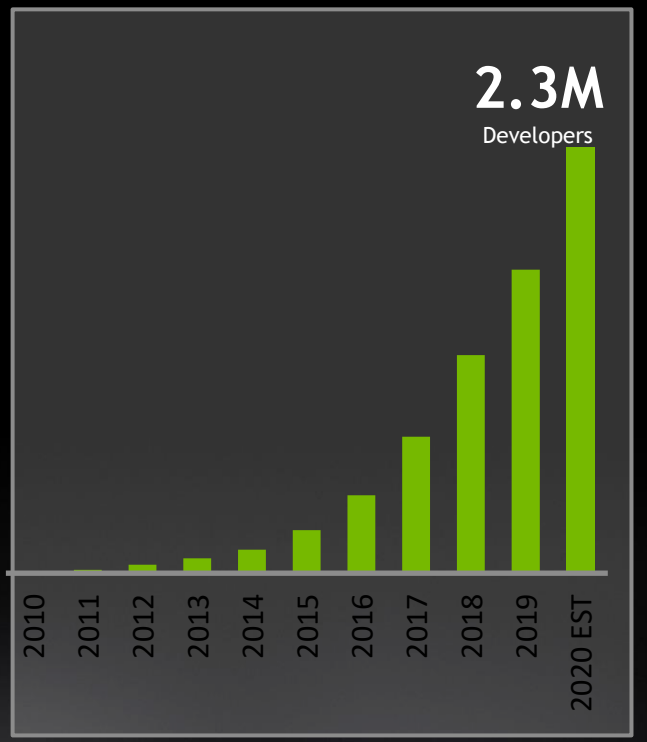
HPL: Based on #1 system in June Top500
AI: Peak system FP16 FLOPS

NVIDIA.

AMAZING EXPANSION OF NVIDIA ECOSYSTEM

Apps for Every Industry Reaching Billions of Users





6M
CUDA Downloads in 2020

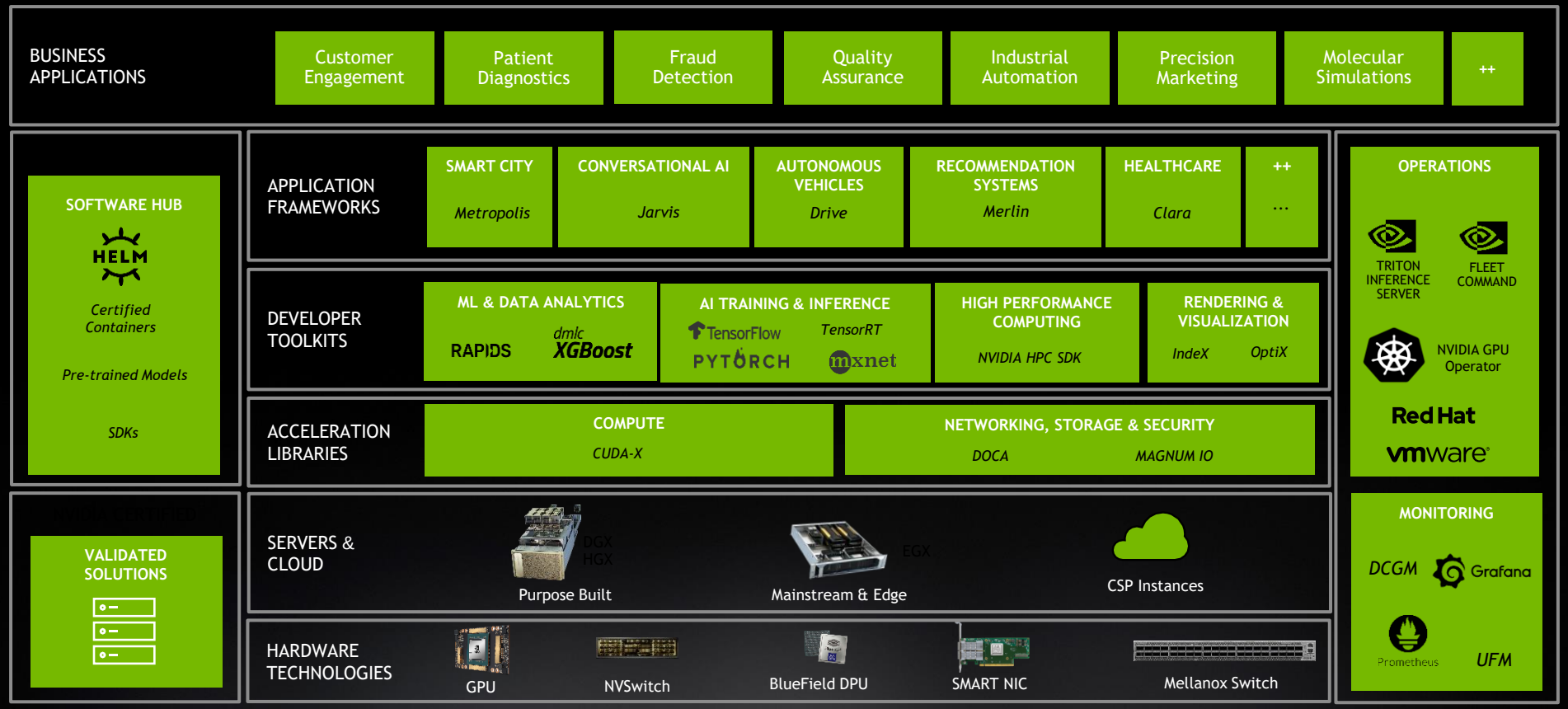
1,800
GPU-Accelerated Applications

6,500
AI Startups

COMPLETE SOFTWARE STACK

GROWING ECOSYSTEM

NVIDIA DATACENTER PLATFORM

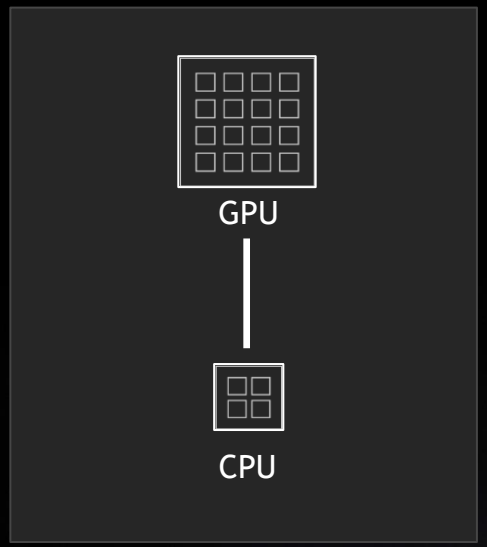


INDIA FASTEST SUPERCOMPUTER

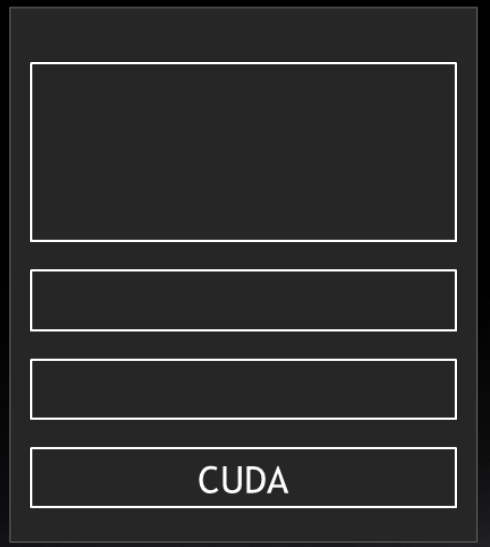
PARAM SIDDHI

- Ranked 62 on the TOP500 ranking
- NVIDIA DGX A100 Systems connected with NVIDIA Mellanox HDR InfiniBand Network
- Offers a performance of 4.6 petaflops sustained and 210 AI petaflops

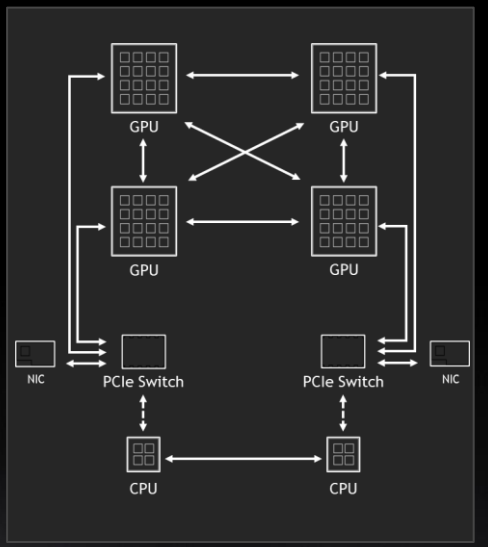
25 YEARS OF ACCELERATED COMPUTING



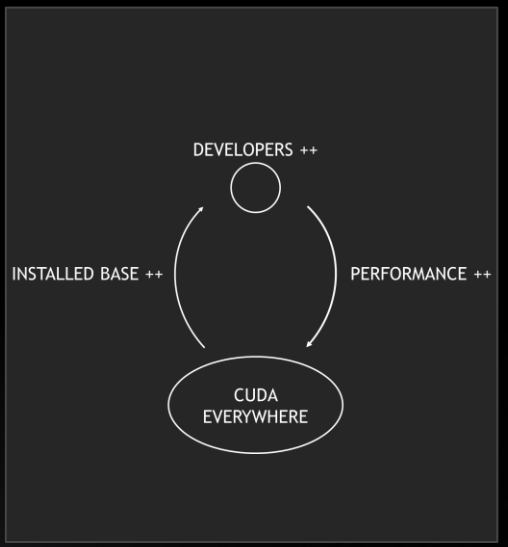
X-FACTOR SPEED UP



FULL STACK

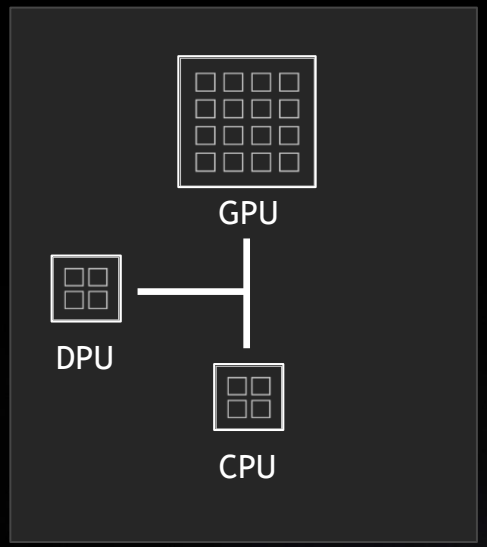


SYSTEMS

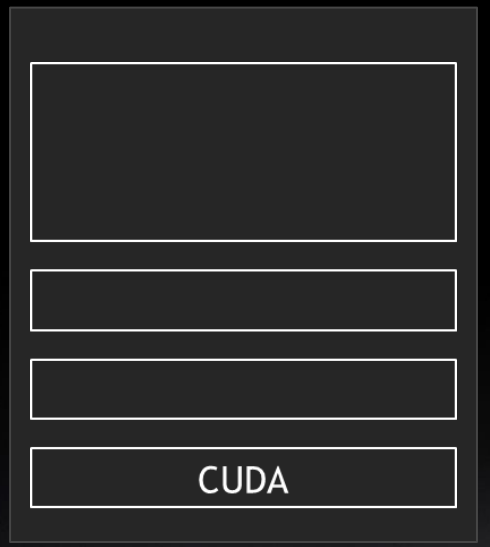


ONE ARCHITECTURE

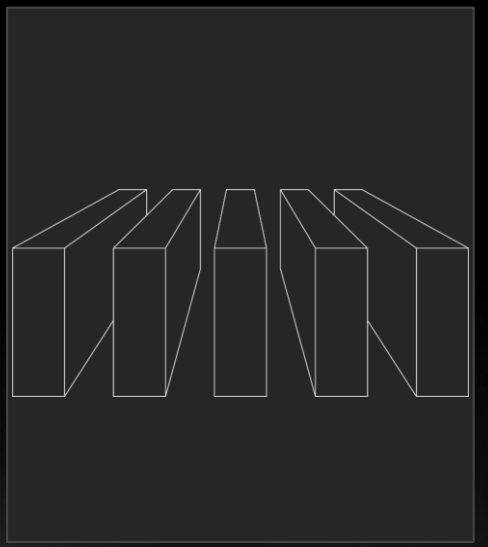
25 YEARS OF ACCELERATED COMPUTING



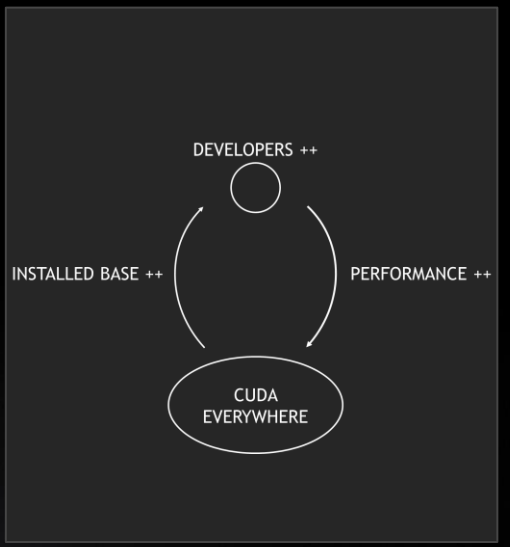
X-FACTOR SPEED UP



FULL STACK



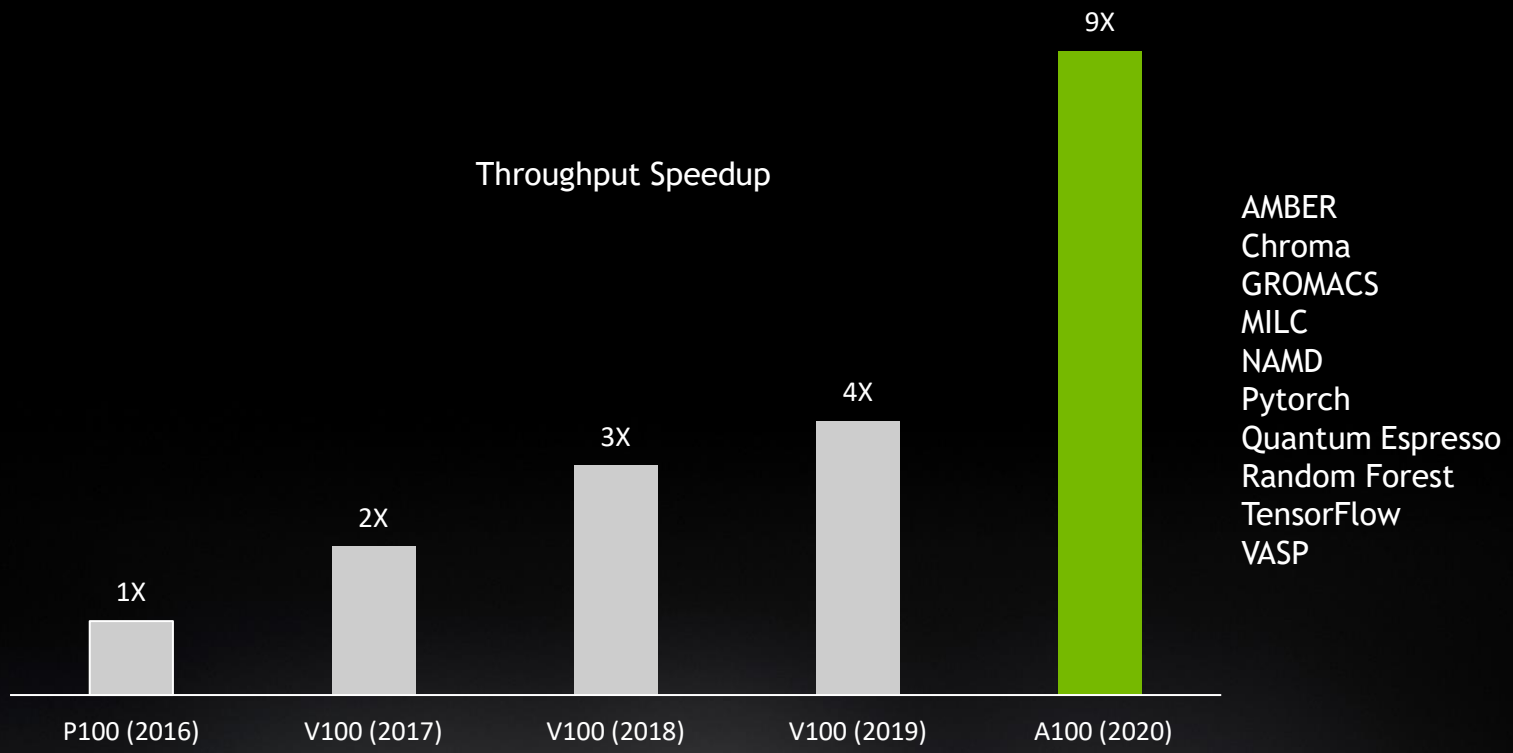
DATA-CENTER SCALE



ONE ARCHITECTURE

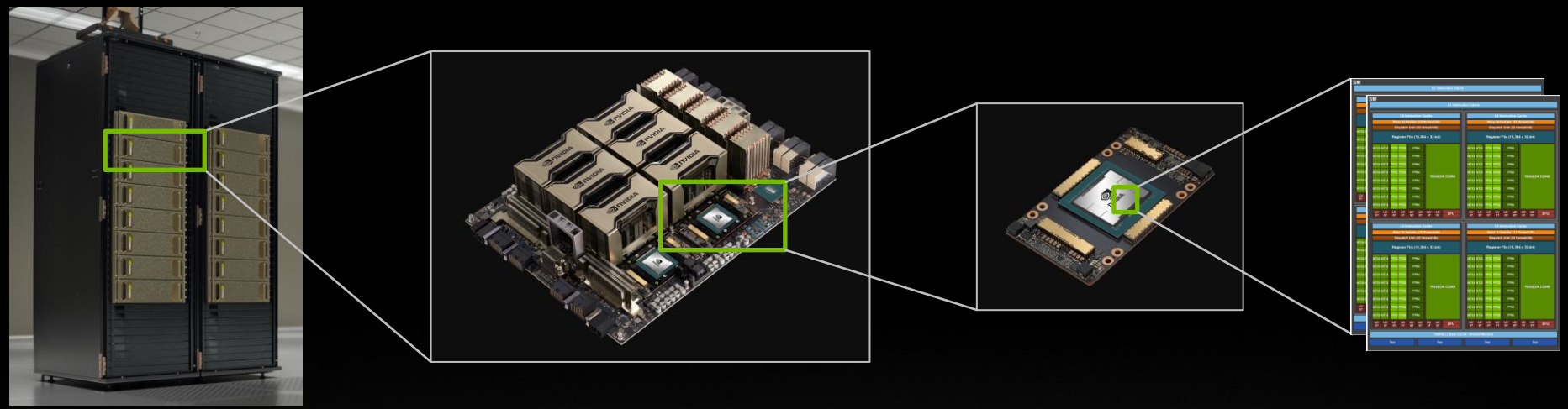
9X MORE PERFORMANCE IN 4 YEARS

Beyond Moore's Law With Full Stack Innovation



Geometric Mean of application speedups vs. P100 : Benchmark Application: Amber [PME-Cellulose_NVE], Chroma [szscl21_24_128], GROMACS [ADH Dodec], MILC [Apex Medium], NAMD [stmv_nve_cuda], PyTorch (BERT Large Fine Tuner], Quantum Espresso [AUSURF112-jR]; Random Forest FP32 [make_blobs (160000 x 64 : 10)], TensorFlow [ResNet-50], VASP 6 [Si Huge], |GPU node: with dual-socket CPUs with 4x P100, V100, or A100 GPUs.

HIERARCHY OF SCALES



Multi-System Rack
Unlimited Scale

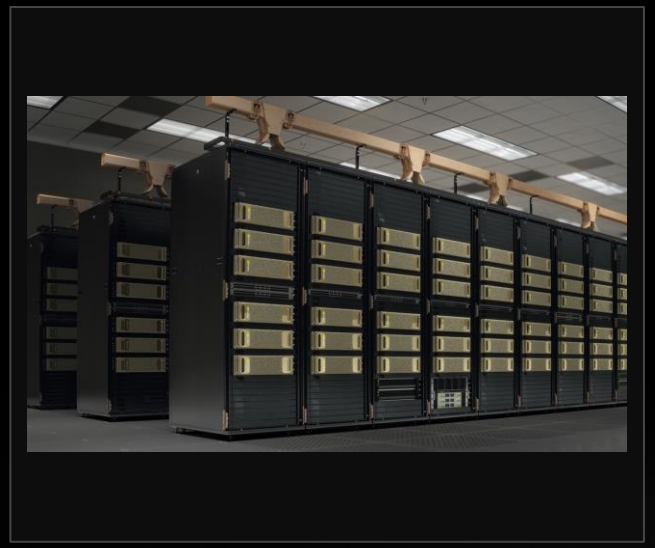
Multi-GPU System
8 GPUs

Multi-SM GPU
108 Multiprocessors

Multi-Core SM
2048 threads

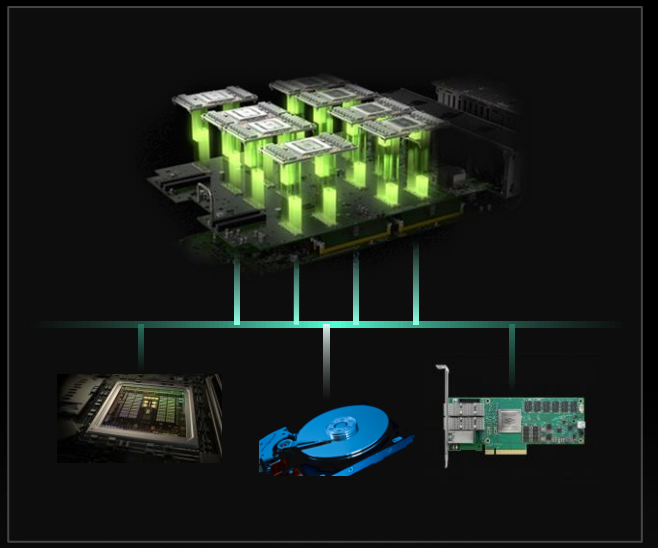
CUDA PLATFORM: TARGETS EACH LEVEL OF THE HIERARCHY

The CUDA Platform Advances State Of The Art From Data Center To The GPU



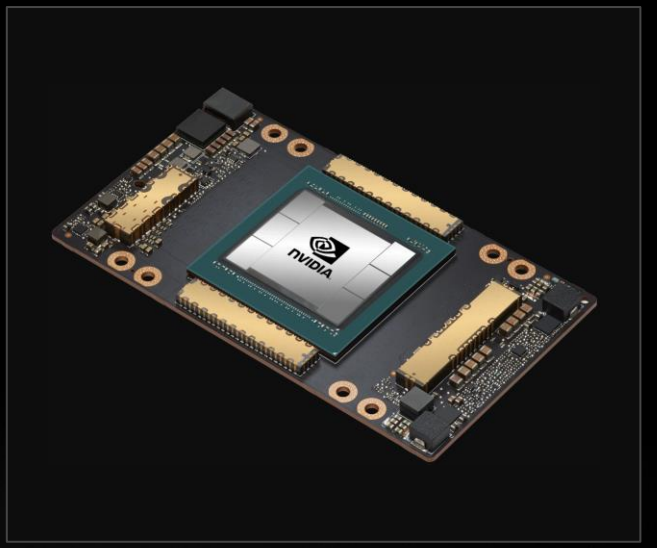
System Scope

FABRIC MANAGEMENT
DATA CENTER OPERATIONS
DEPLOYMENT
MONITORING
COMPATIBILITY
SECURITY



Node Scope

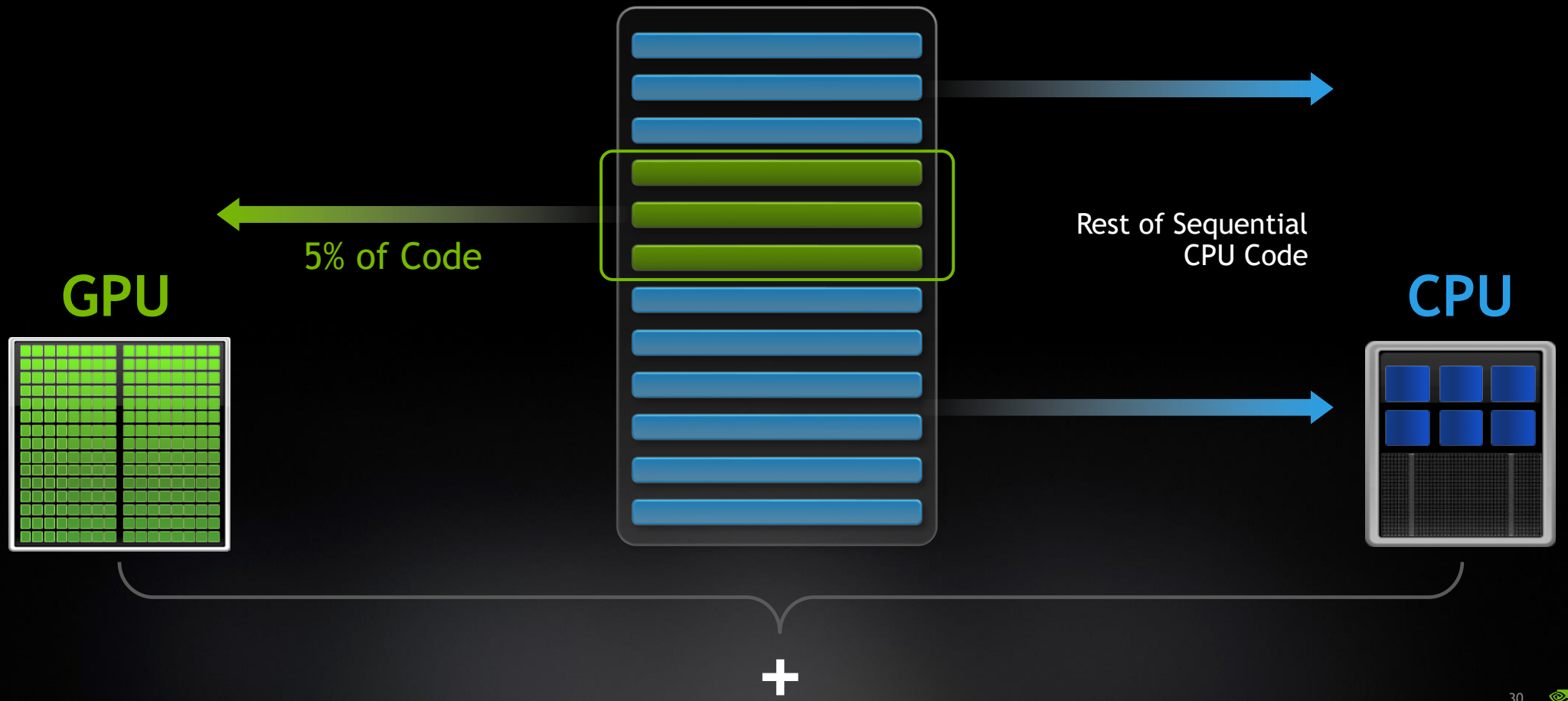
GPU-DIRECT
NVLINK
LIBRARIES
UNIFIED MEMORY
ARM
MIG



Program Scope

CUDA C++
OPENACC
STANDARD LANGUAGES
SYNCHRONIZATION
PRECISION
TASK GRAPHS

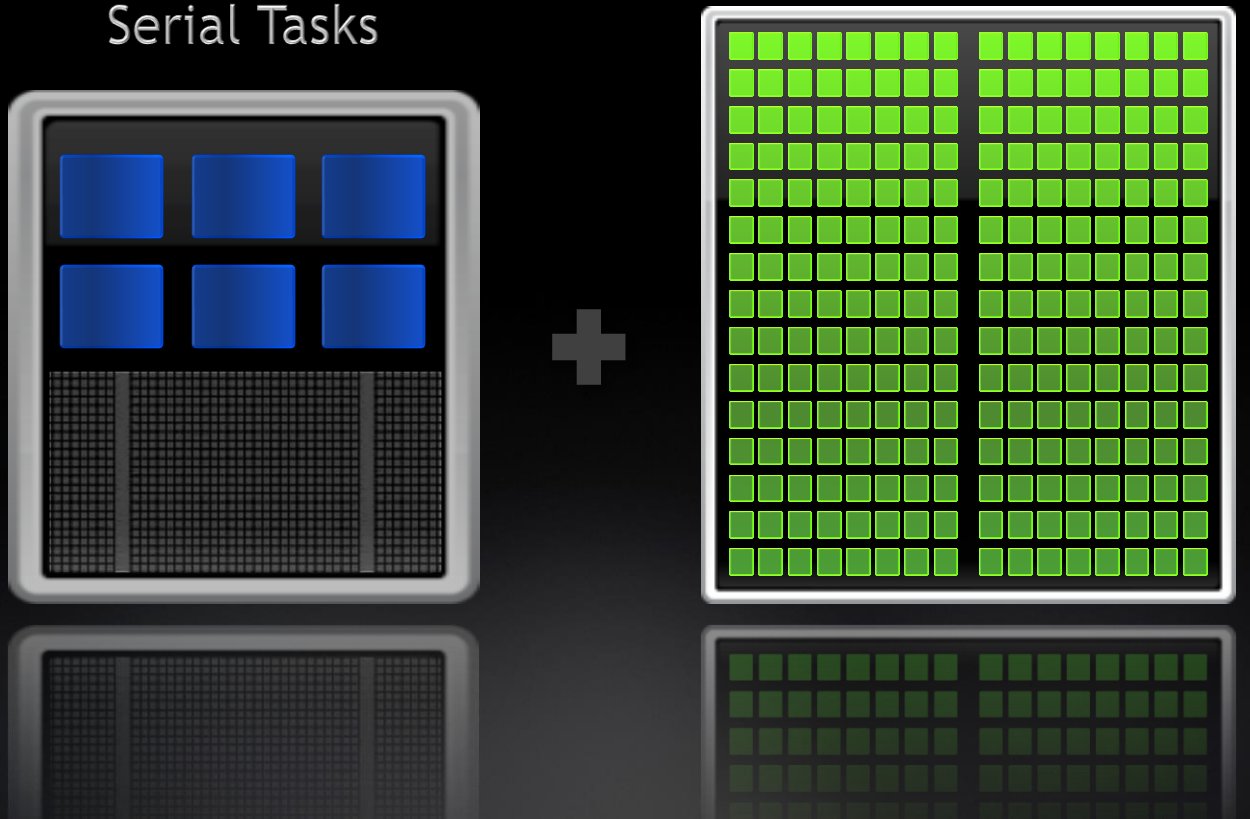
HOW GPU ACCELERATION WORKS



ACCELERATED COMPUTING

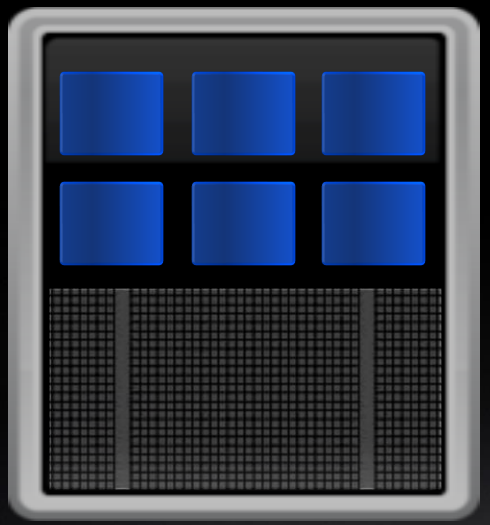
CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks



CPU IS A LATENCY REDUCING ARCHITECTURE

CPU
Optimized for
Serial Tasks



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

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

GPU IS ALL ABOUT HIDING LATENCY

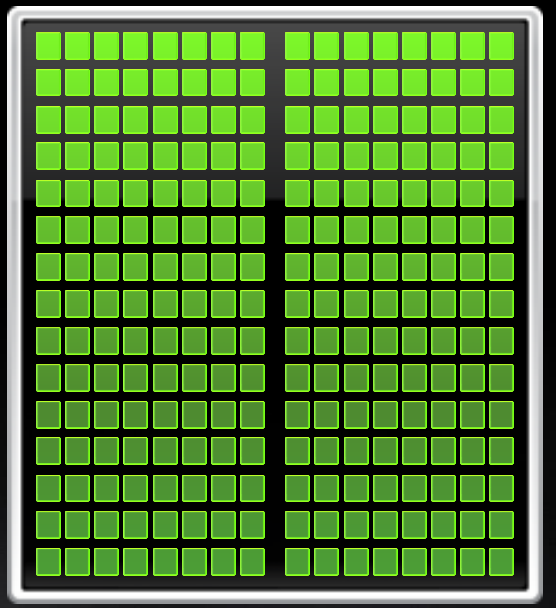
GPU Strengths

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

GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

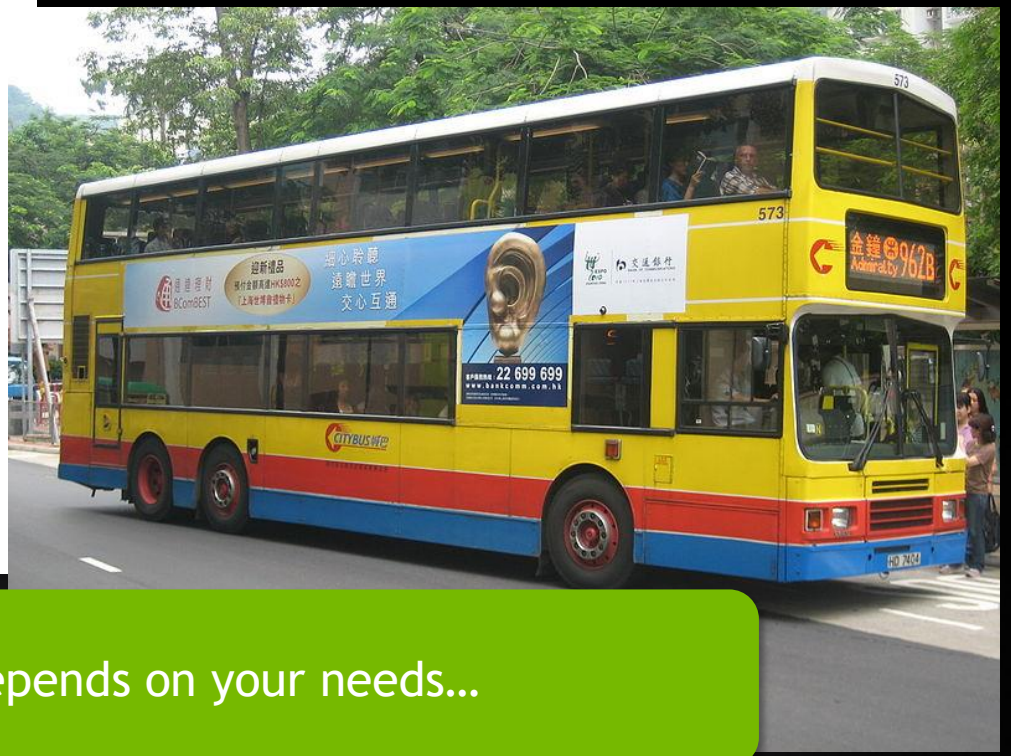
GPU Accelerator
Optimized for
Parallel Tasks



SPEED V. THROUGHPUT

Speed

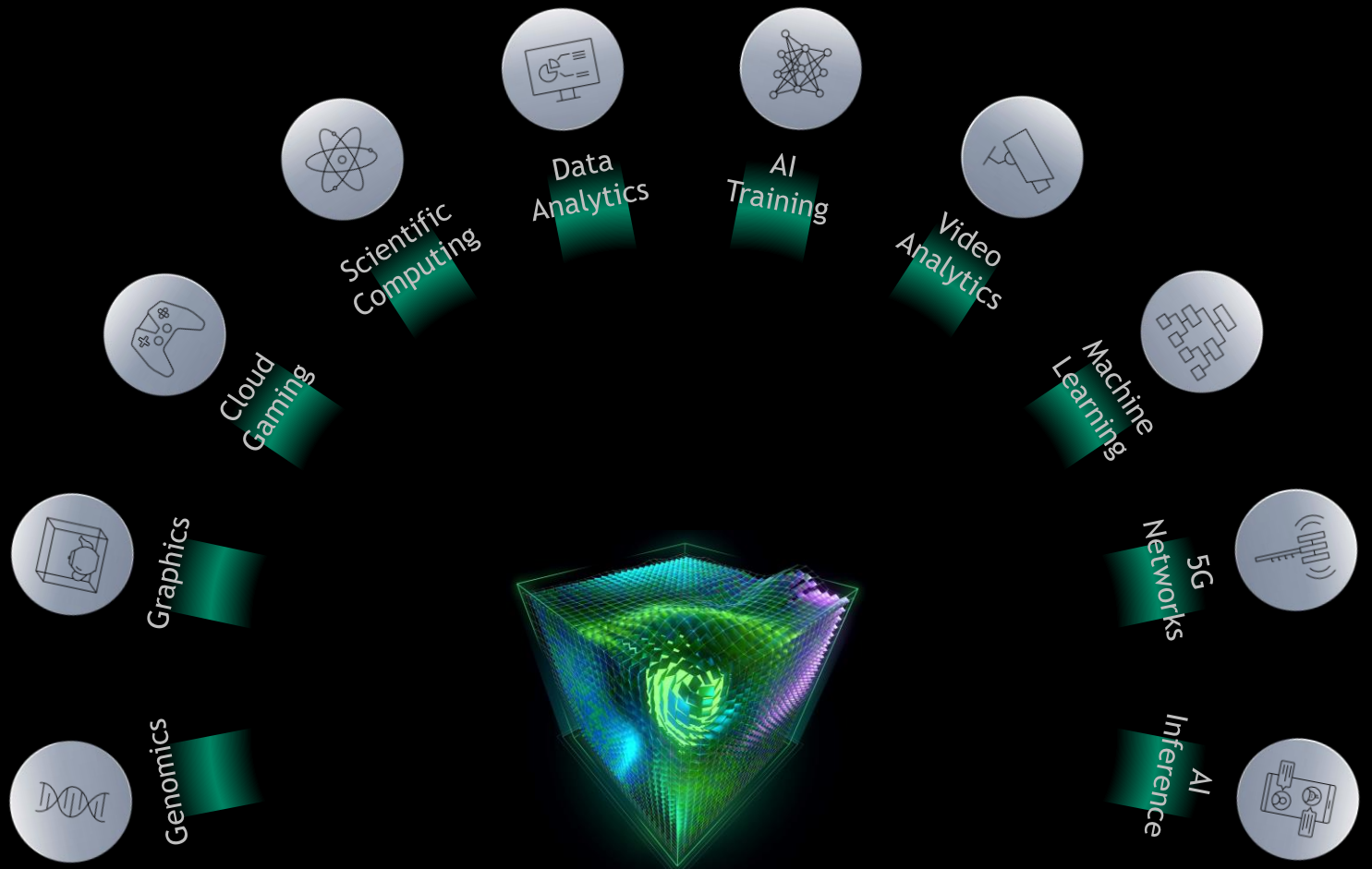
Throughput



Which is better depends on your needs…

Images from Wikimedia Commons via Creative Commons

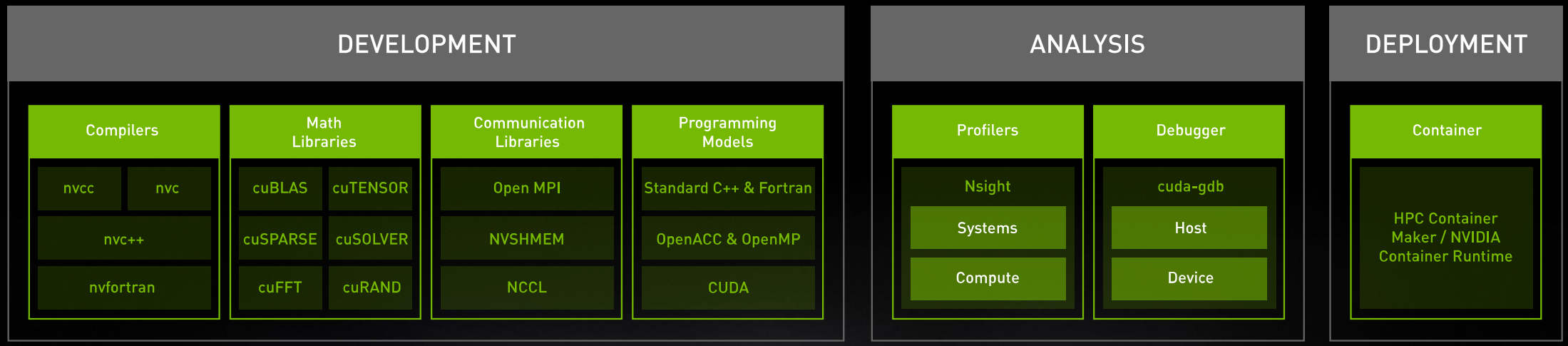
HUGE BREADTH OF PLATFORMS, SYSTEMS, LANGUAGES



INTRODUCING: THE NVIDIA HPC SDK

Download at developer.nvidia.com/hpc-sdk

NVIDIA HPC SDK



Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA

Compatible with 99% of Top500

GPU PROGRAMMING IN 2020 AND BEYOND

Math Libraries | Standard Languages | Directives | CUDA

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

GPU Accelerated
C++ and Fortran

```
#pragma acc data copy(x,y)
{

...

std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});

...

}
```

Incremental Performance
Optimization with Directives

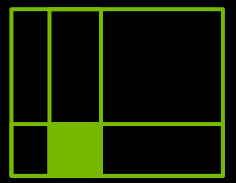
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  cudaMallocManaged(&x, ...);
  cudaMallocManaged(&y, ...);
  ...
  saxpy<<<(N+255)/256,256>>>(...,x, y)
  cudaDeviceSynchronize();
  ...
}
```

Maximize GPU Performance with
CUDA C++/Fortran

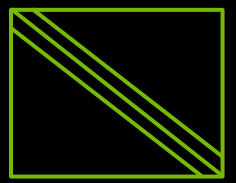
GPU Accelerated Math Libraries

GPU ACCELERATED MATH LIBRARIES



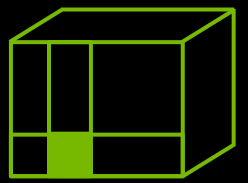
cuBLAS

BF16, TF32 and FP64 Tensor Cores



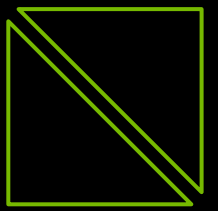
cuSPARSE

Increased memory BW, Shared Memory & L2



cuTENSOR

BF16, TF32 and FP64 Tensor Cores

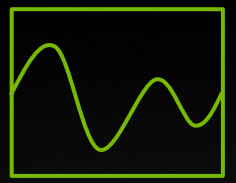


cuSOLVER

BF16, TF32 and FP64 Tensor Cores

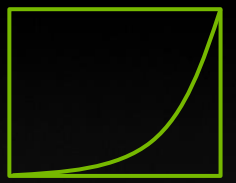
nvJPEG

Hardware Decoder



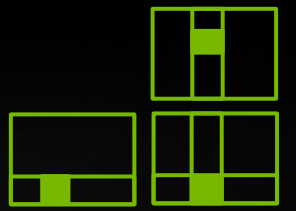
cuFFT

BF16, TF32 and FP64 Tensor Cores



CUDA Math API

Increased memory BW, Shared Memory & L2



CUTLASS

BF16 & TF32 Support

SINGLE PRECISION ALPHA X PLUS Y (SAXPY)

GPU SAXPY in multiple languages and libraries

Part of Basic Linear Algebra Subroutines (BLAS) Library

$$z = \alpha x + y$$

x, y, z : vector

α : scalar

SAXPY: OPENACC COMPILER DIRECTIVES

Parallel C Code

```
void saxpy(int n,
           float a,
           float *x,
           float *y)
{
#pragma acc kernels
  for (int i = 0; i <
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Parallel Fortran Code

```
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
!$acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
!$acc end kernels
end subroutine saxpy


...
! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
```

www.openacc.org

SAXPY: CUBLAS LIBRARY

Serial BLAS Code

```
int N = 1<<20;

...

// Use your choice of blas library

// Perform SAXPY on 1M elements
blas_saxpy(N, 2.0, x, 1, y, 1);
```

Parallel cuBLAS Code

```
int N = 1<<20;


cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasShutdown();
```

You can also call cuBLAS from Fortran, C++, Python, and other languages:
http://developer.nvidia.com/cublas

SAXPY: CUDA C

Standard C

```
void saxpy(int n, float a,
      float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;



// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

Parallel C

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

int N = 1<<20;
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

http://developer.nvidia.com/cuda-toolkit

SAXPY: CUDA FORTRAN

Standard Fortran

```
module mymodule contains
  subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i)+y(i)
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy(2**20, 2.0, x, y)

end program main
```

Parallel Fortran

```
module mymodule contains
  attributes(global) subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    attributes(value) :: a, n
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i)+y(i)
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x_d(2**20), y_d(2**20)
  x_d = 1.0, y_d = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)

end program main
```

http://developer.nvidia.com/cuda-fortran

SAXPY: PYTHON

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

Numba: Parallel Python

```
import numpy as np
from numba import vectorize

@vectorize(['float32(float32, float32,
float32)'], target='cuda')
def saxpy(a, x, y):
    return a * x + y

N = 1048576

# Initialize arrays
A = np.ones(N, dtype=np.float32)
B = np.ones(A.shape, dtype=A.dtype)
C = np.empty_like(A, dtype=A.dtype)

# Add arrays onGPU
C = saxpy(2.0, X, Y)
```

https://numba.pydata.org

SAXPY: PSTL

Serial C++ Code (with STL and Boost)

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...



// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
           2.0f * _1 + _2);
```

www.boost.org/libs/lambda

Parallel C++ Code

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...



// Perform SAXPY on 1M elements
std::transform(std::execution::par
, x.begin(), x.end(),
                 y.begin(), y.end(),
           2.0f * _1 + _2);
```

https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9770-c++17-parallel-algorithms-for-nvidia-gpus-with-pgi-c++.pdf

SAXPY: MATLAB

Parallel C Code

```
void saxpy(int n,
           float a,
           float *x,
           float *y)
{
#pragma acc kernels
  for (int i = 0; i <
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Parallel C++ Code

```
<<initialize>>
p = parpool

parfor i = 1:numel(N)
   y(i) = 2.0 * x(i) + y(i)
end

<<post process>
delete(p)
```

SAXPY: MATLAB

- 500+ GPU-enabled MATLAB functions
- Additional GPU-enabled Toolboxes
  - Neural Networks
  - Image Processing and Computer Vision
  - Communications
  - Signal Processing
  - Stats Toolbox

Transfer Data To GPU From

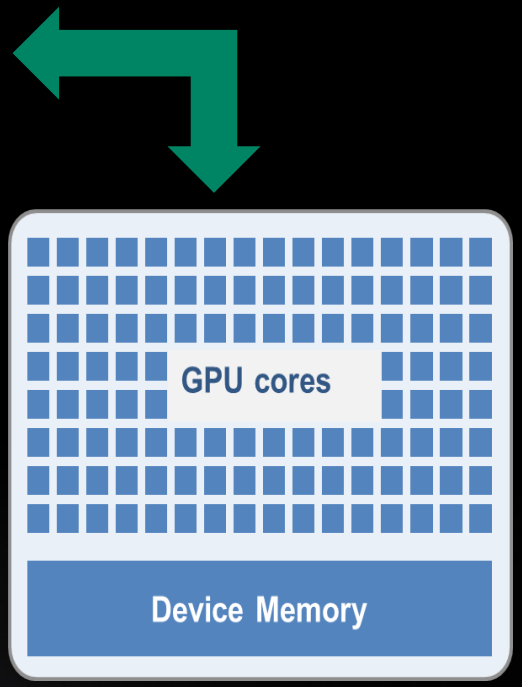
```
x=gpuArray(x);
```

Perform Calculation on GPU

```
X=saxpy(N,2.0,x,0,1,y,0,1);
```

Gather Data or Plot

```
y=gather(y)
```



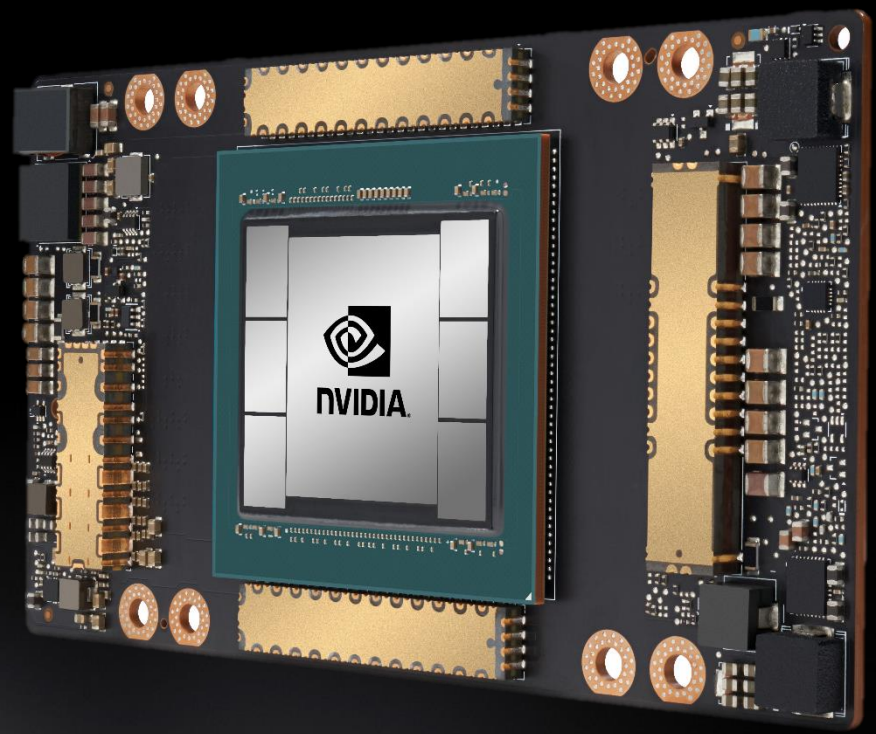

THANK YOU
NVIDIA

CHOOSING THE RIGHT DATA CENTER GPU

	Applications Workloads	Availability
A100	Training, Inference, HPC	Today: DGX A100 Starting Q3: DGX A100/Cloud/OEM Servers
V100	Training, HPC	Broadly Available
T4	Dedicated Inference, Low Power Servers	Broadly Available
RTX 6K/8K	Graphics Intensive	Broadly Available

NVIDIA A100 DETAILED SPECS

	Peak Performance
Transistor Count	54 billion
Die Size	826 mm^2
FP64 CUDA Cores	3,456
FP32 CUDA Cores	6,912
Tensor Cores	432
Streaming Multiprocessors	108
FP64	9.7 teraFLOPS
FP64 Tensor Core	19.5 teraFLOPS
FP32	19.5 teraFLOPS
TF32 Tensor Core	156 teraFLOPS | 312 teraFLOPS*
BFLOAT16 Tensor Core	312 teraFLOPS | 624 teraFLOPS*
FP16 Tensor Core	312 teraFLOPS | 624 teraFLOPS*
INT8 Tensor Core	624 TOPS | 1,248 TOPS*
INT4 Tensor Core	1,248 TOPS | 2,496 TOPS*
GPU Memory	40 GB
Interconnect	NVLink 600 GB/s PCIe Gen4 64 GB/s
Multi-Instance GPUs	Various Instance sizes with up to 7MIGs @5GB
Form Factor	4/8/16 SXM GPUs in HGX A100
Max Power	400W (SXM)



* Includes Sparsity