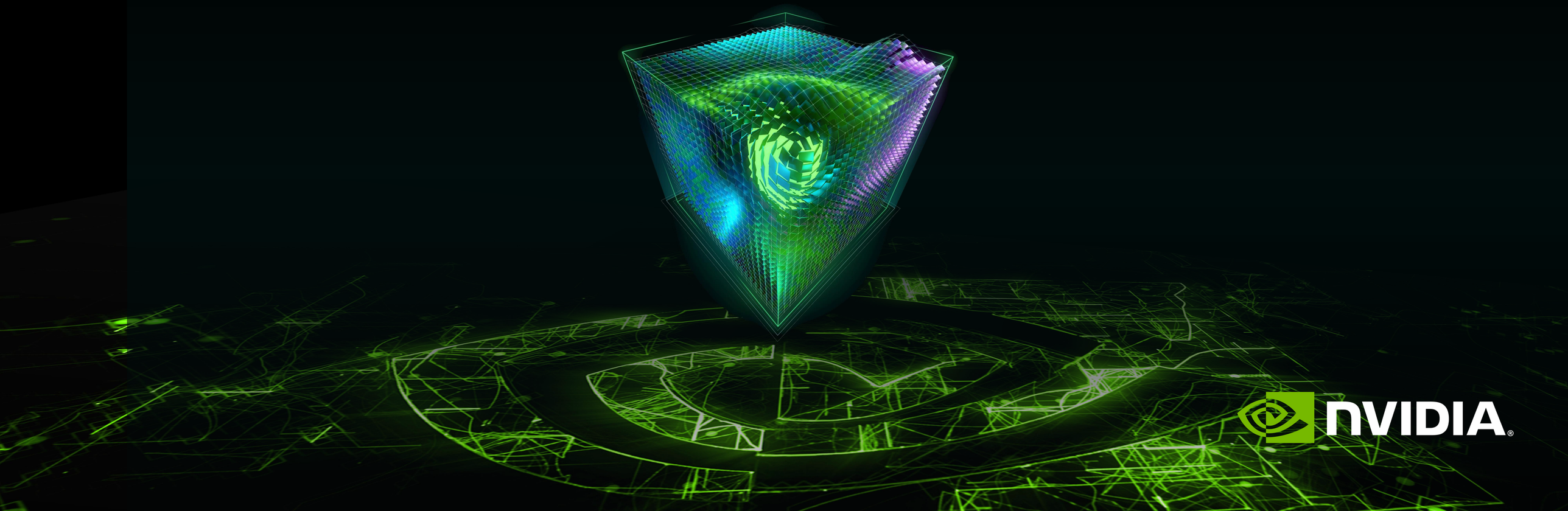
CUDA: New Features and Beyond

Stephen Jones, GTC 2023



Moore’s Law is dead

- NVIDIA CEO Jensen Huang

Moore’s Law is dead

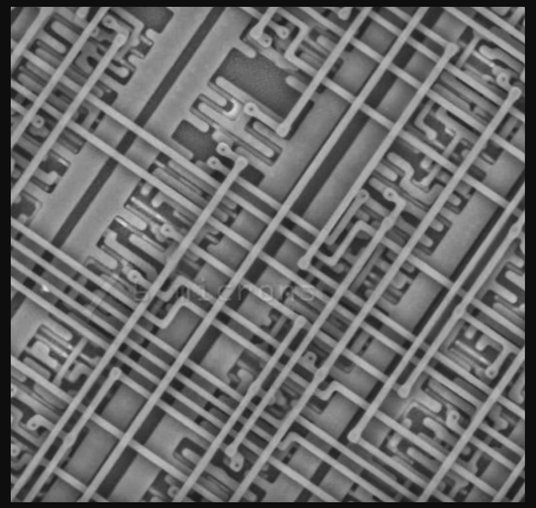
… computing is a not a chip problem
It’s a software and chip problem

- NVIDIA CEO Jensen Huang

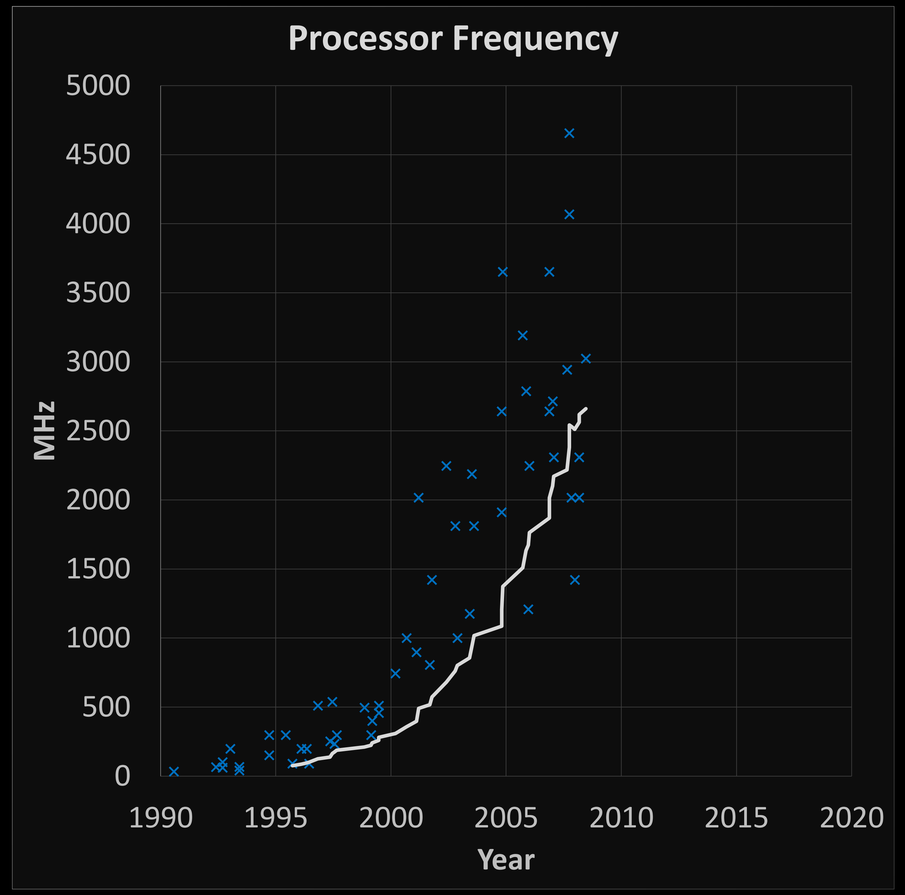
First Phase of Moore’s Law

Dennard Scaling
(until ~2007)

Doubling through smaller line size

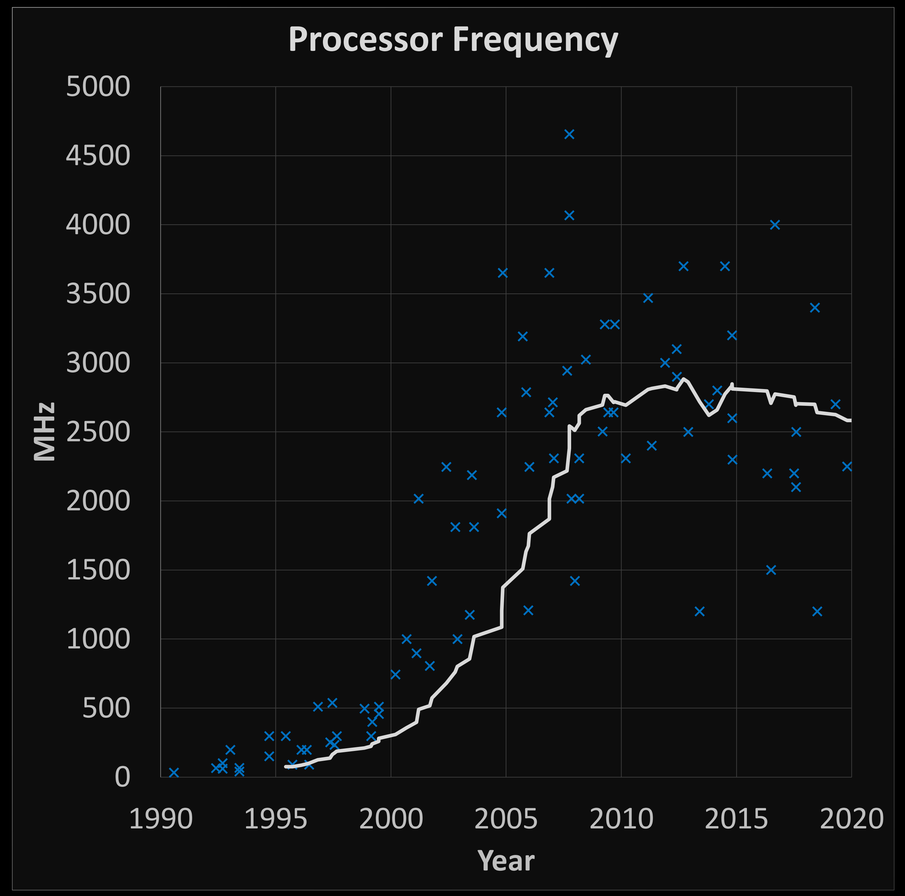


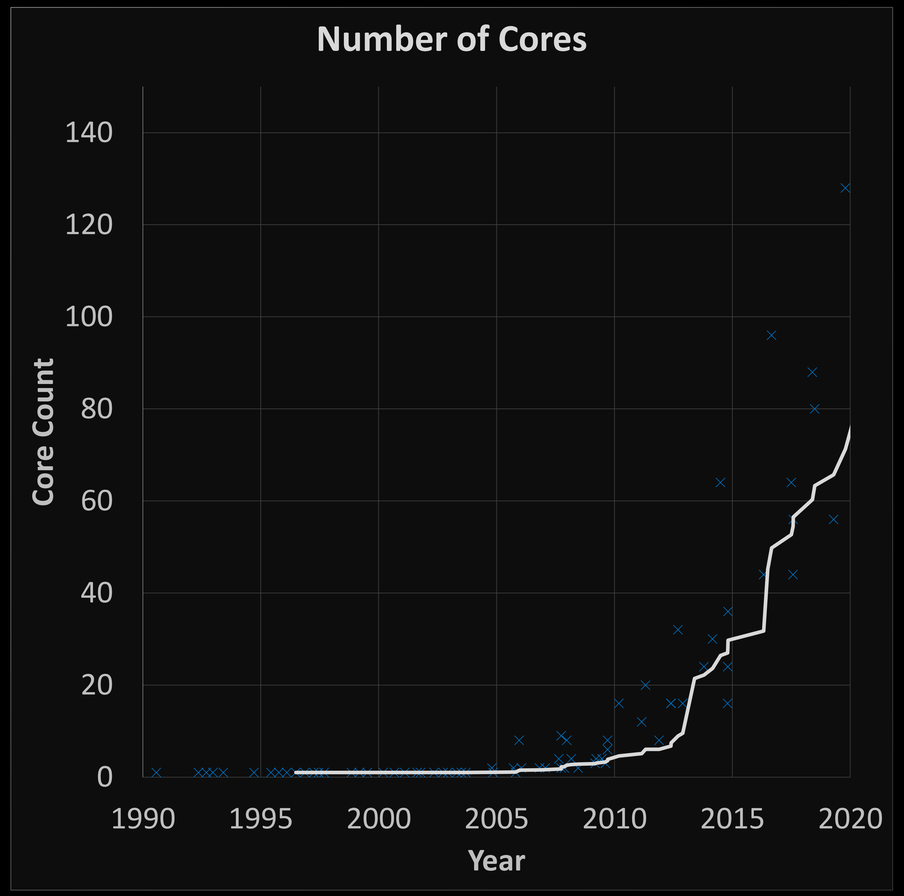
The End of Dennard Scaling



Source: Karl Rupp “40 years of microprocessor trend data”
https://github.com/karlrupp/microprocessor-trend-data

The End of Dennard Scaling



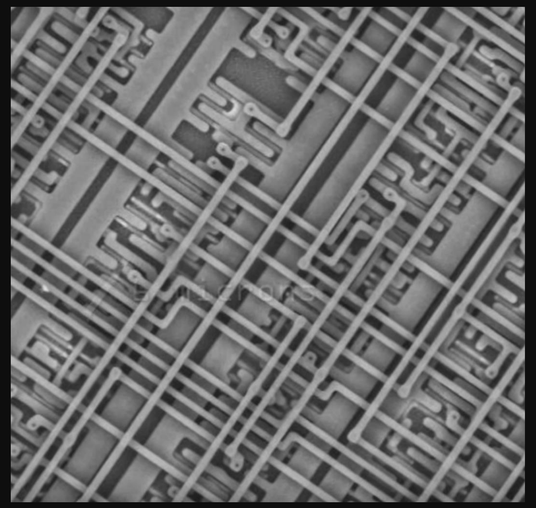


Source: Karl Rupp “40 years of microprocessor trend data”
https://github.com/karlrupp/microprocessor-trend-data

Second Phase of Moore's Law

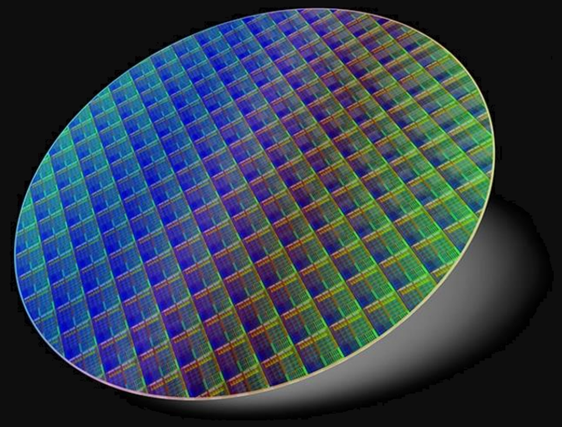
Dennard Scaling
(until ~2007)

Doubling through smaller line size



Single-Die Scaling
(~2007 → Present)

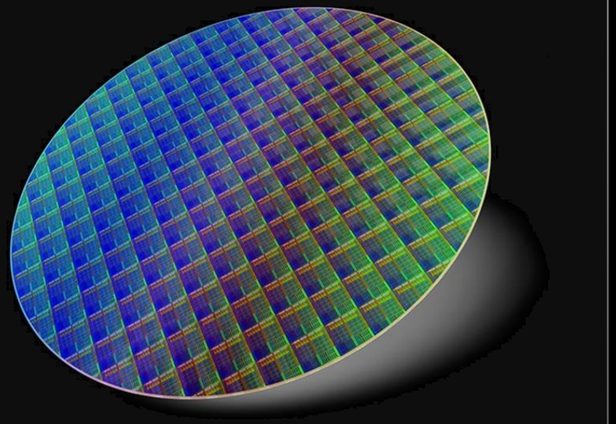
Doubling through increasing core count

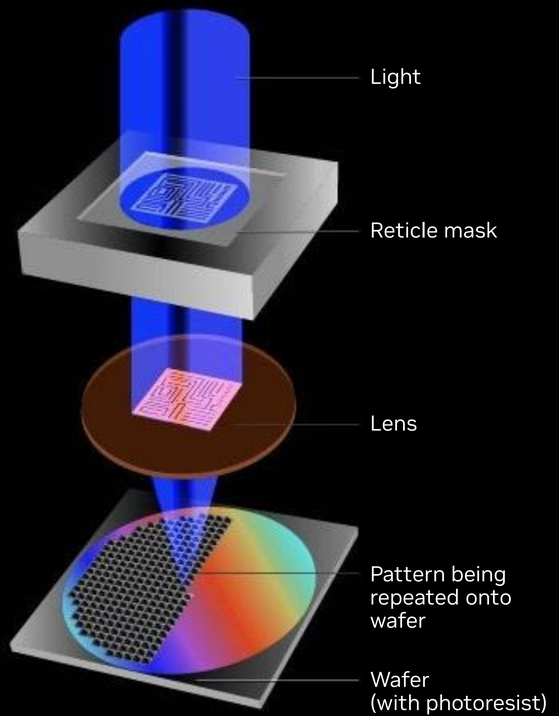


The End of Single-Die Scaling

Single-Die Scaling
(~2007 → Present)

Doubling through increasing core count





Overcoming the Reticle Limit: Heterogeneous Hardware

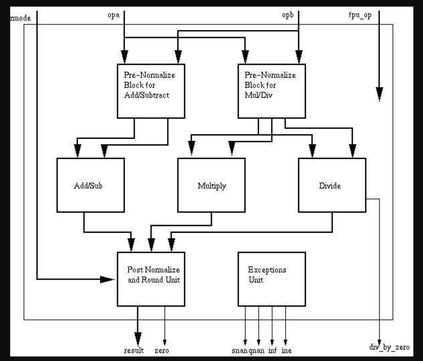
Specialization of hardware components

heterogeneous

adjective US: /ˌhetˌə.roʊˈdʒiː.ni.əs/ UK: /ˌhet.ər.əˈdʒiː.ni.əs/

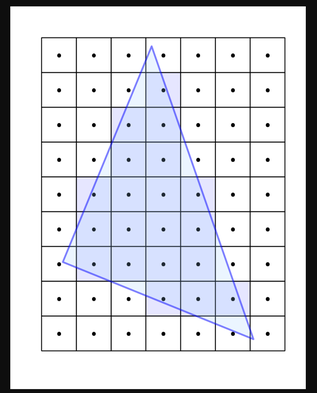
consisting of parts or things that are very different from each other

Floating point unit

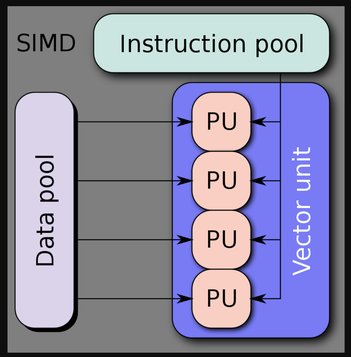


Texture mapping

Rasterization



Vector unit

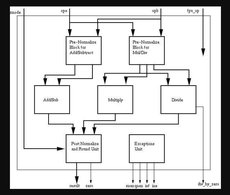


NVIDIA

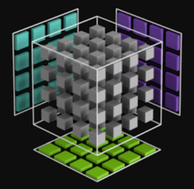
Overcoming the Reticle Limit: Heterogeneous Hardware

NVIDIA Ada microarchitecture

Floating point unit

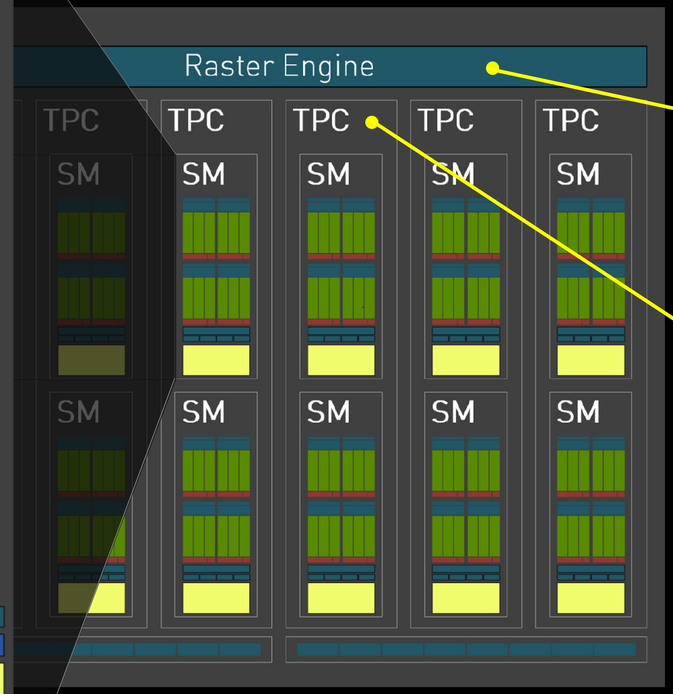


Tensor core



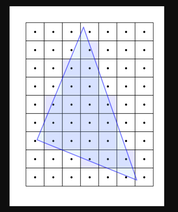
Ray tracing unit

Ada streaming multiprocessor



Ada graphics processing cluster

Rasterization



Texture mapping

Heterogeneous Software

Software is also made up of heterogeneous components written in many different langauges

heterogeneous

adjective US: /ˌhetˌə.roʊˈdʒiː.ni.əs/ UK: /ˌhet.ər.əˈdʒiː.ni.əs/

consisting of parts or things that are very different from each other

Example: Heterogeneous Programming in Python

```
import numpy as np

# Array of 1M random numbers
data = np.random.randint(1, 100, (1024 * 1024))

total = np.sum(data)     # Reduction across array
max = np.amax(data)      # Max in array
average = np.mean(data)  # Average across array
```

← User application is written in Python for productivity

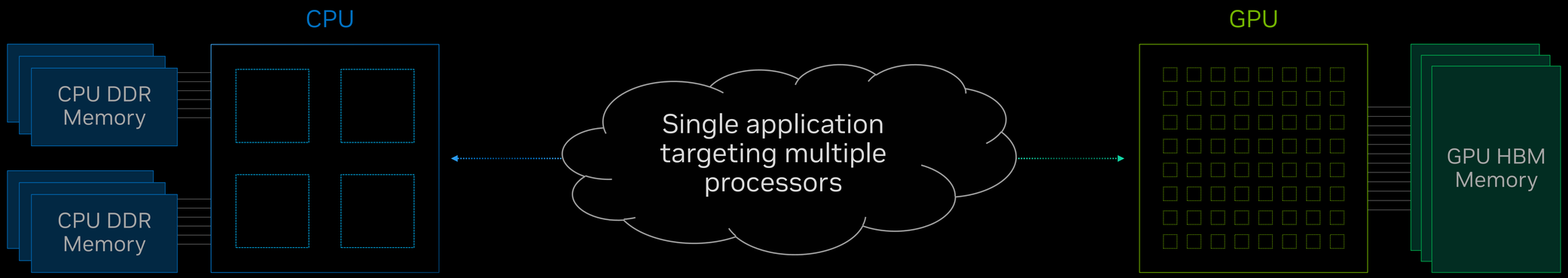
NumPy Library

NumPy library implements fast computation in C & C++

Applications often combine components & libraries written in many different languages

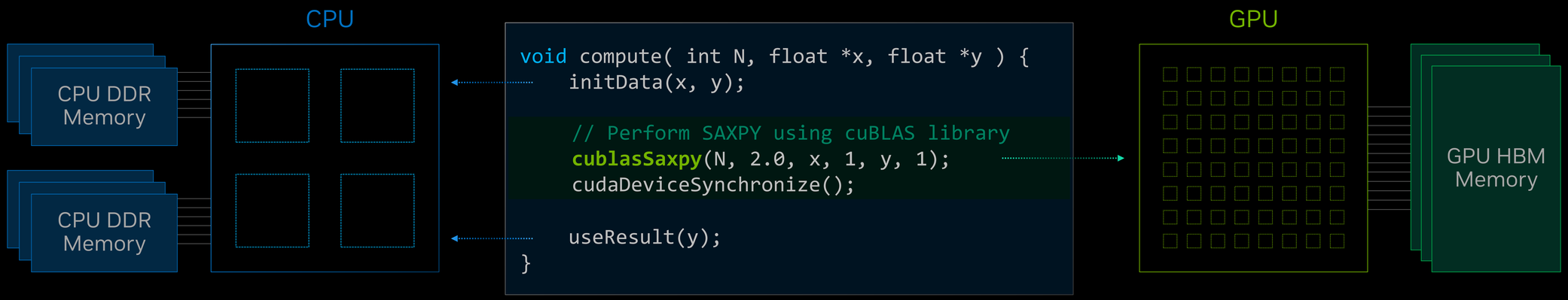
Heterogeneous Computing

Combining processors of different types, each specializing in different types of execution



Heterogeneous Computing

Combining processors of different types, each specializing in different types of execution



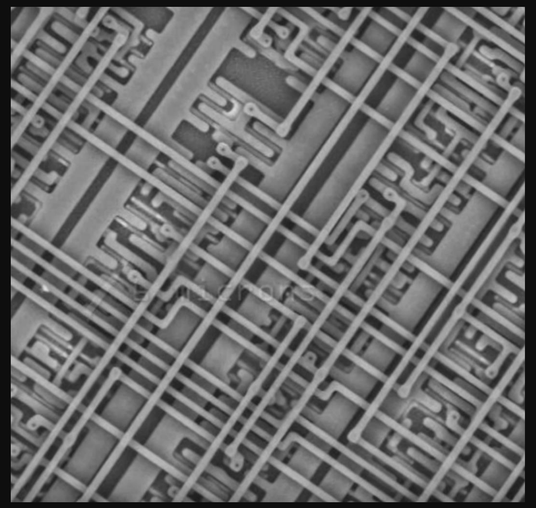
Heterogeneous computing

Calling the GPU-accelerated cuBLAS library

Third Phase of Moore's Law

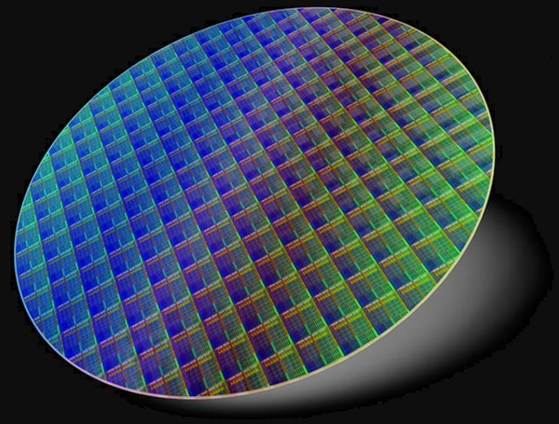
Dennard Scaling
(until ~2007)

Doubling through smaller line size



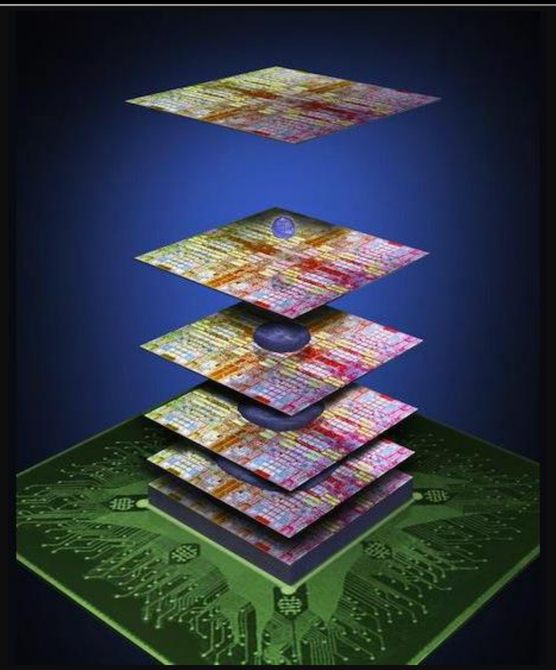
Single-Die Scaling
(~2007 → Present)

Doubling through increasing core count

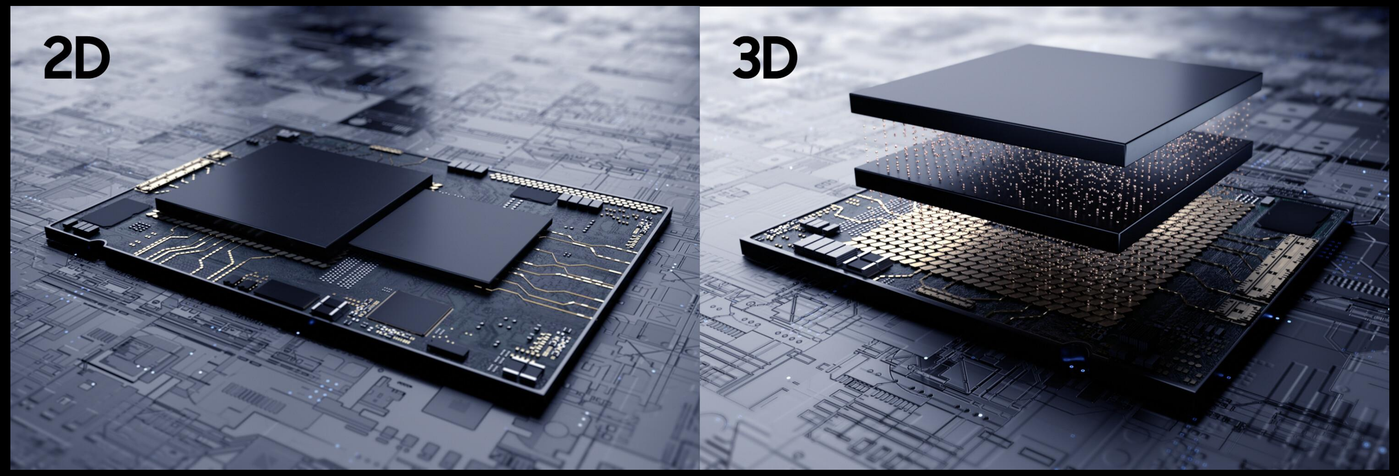


Multi-Die Scaling
(Present → Future)

Doubling through combining dies

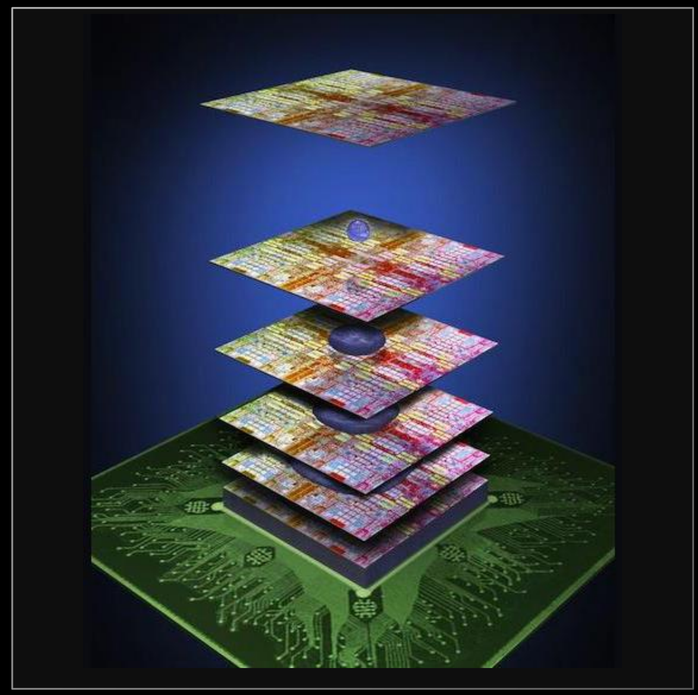


Third Phase of Moore’s Law

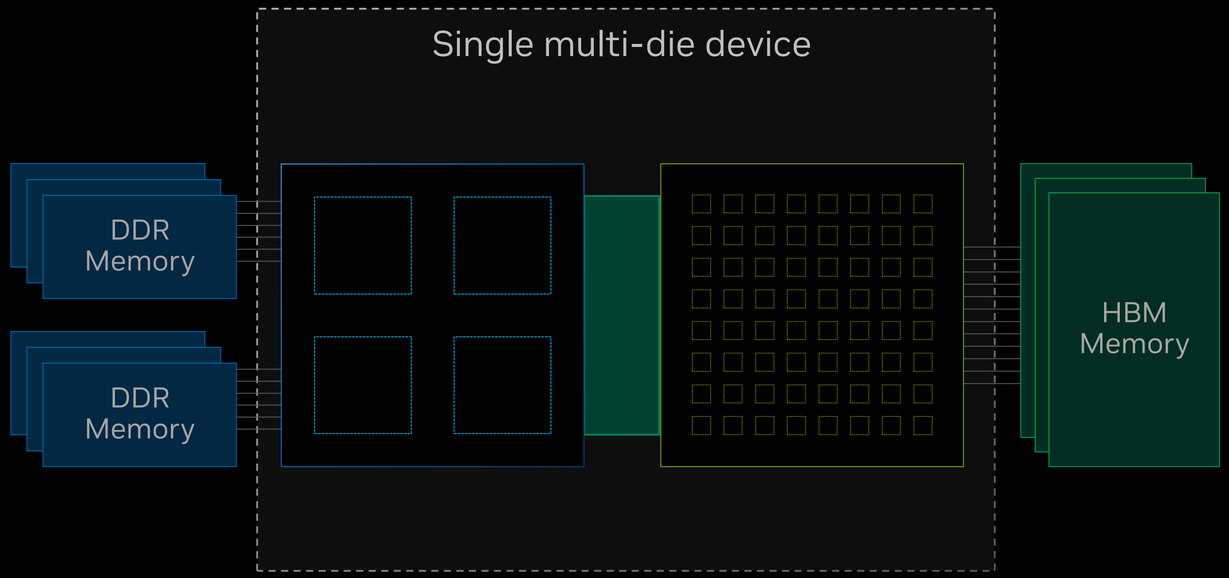


Multi-Die Scaling
(Present → Future)

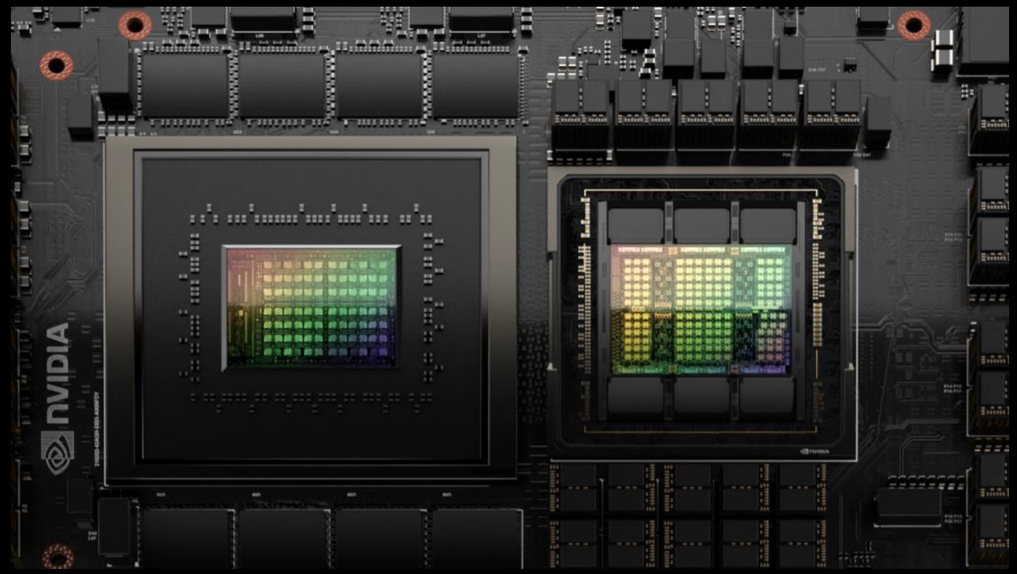
Doubling through combining dies



Multi-Die Heterogeneous Devices

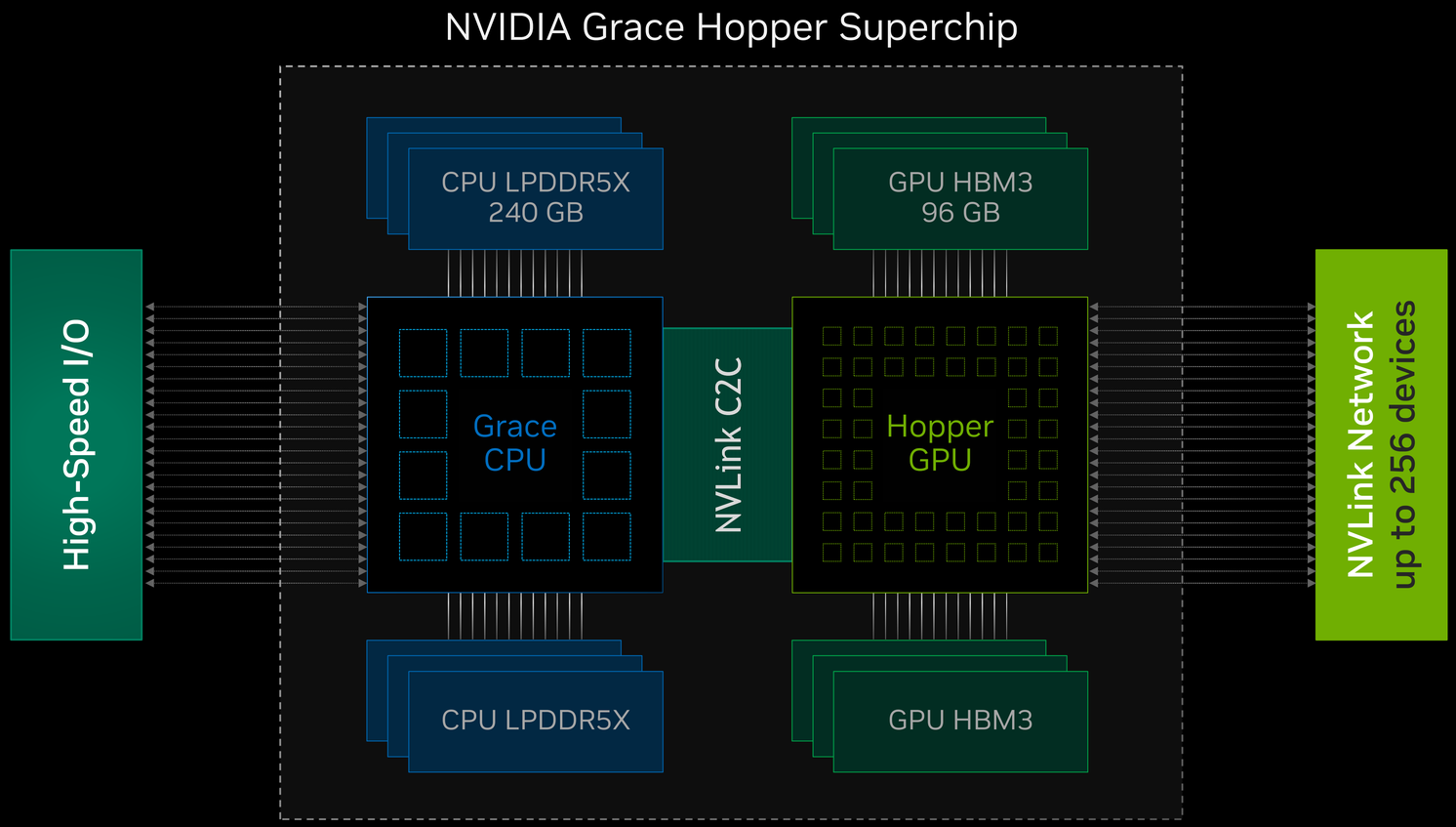


Grace/Hopper Superchip

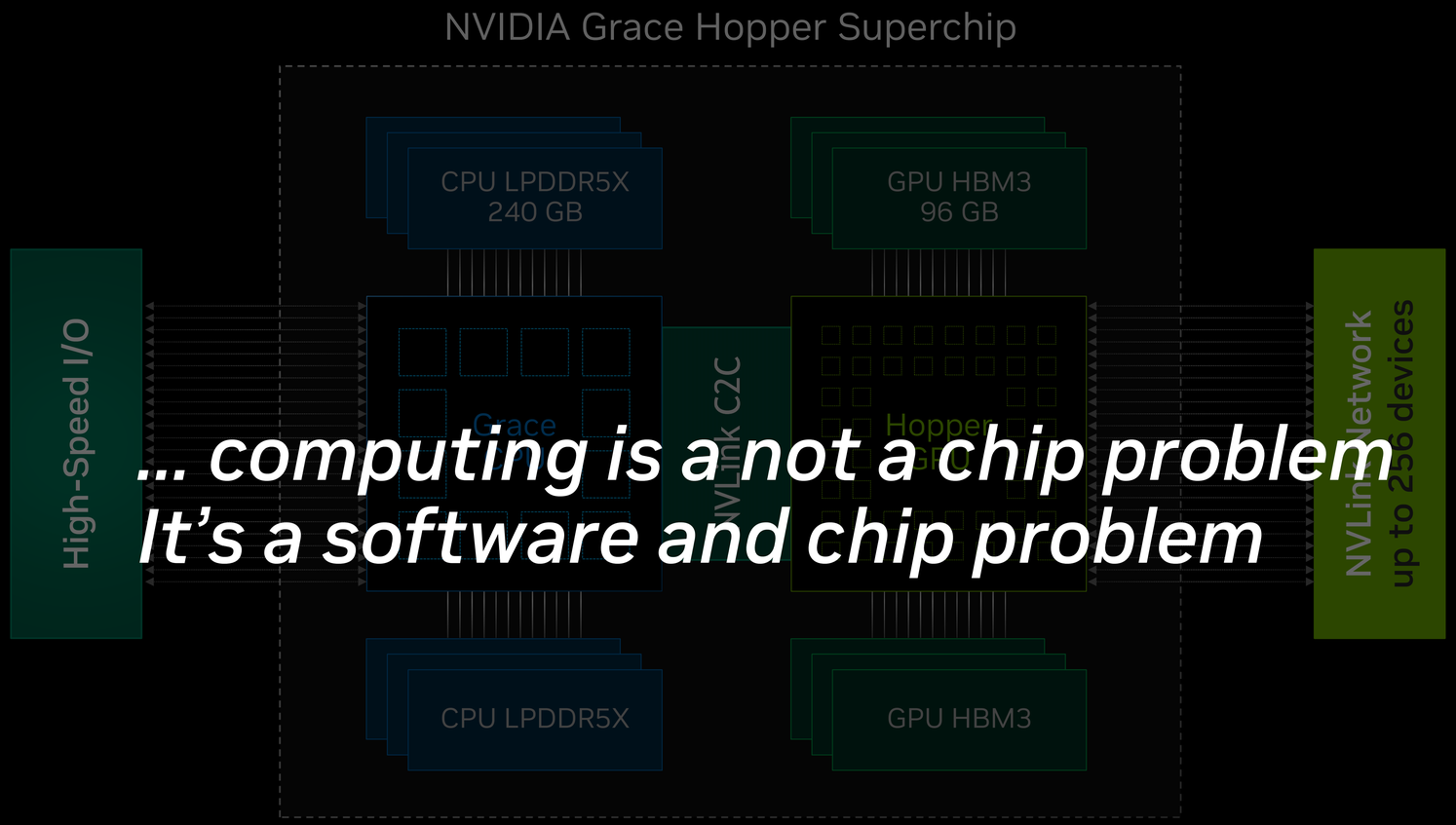


NVIDIA Grace Hopper Superchip Architecture [Paper]

Grace/Hopper Superchip

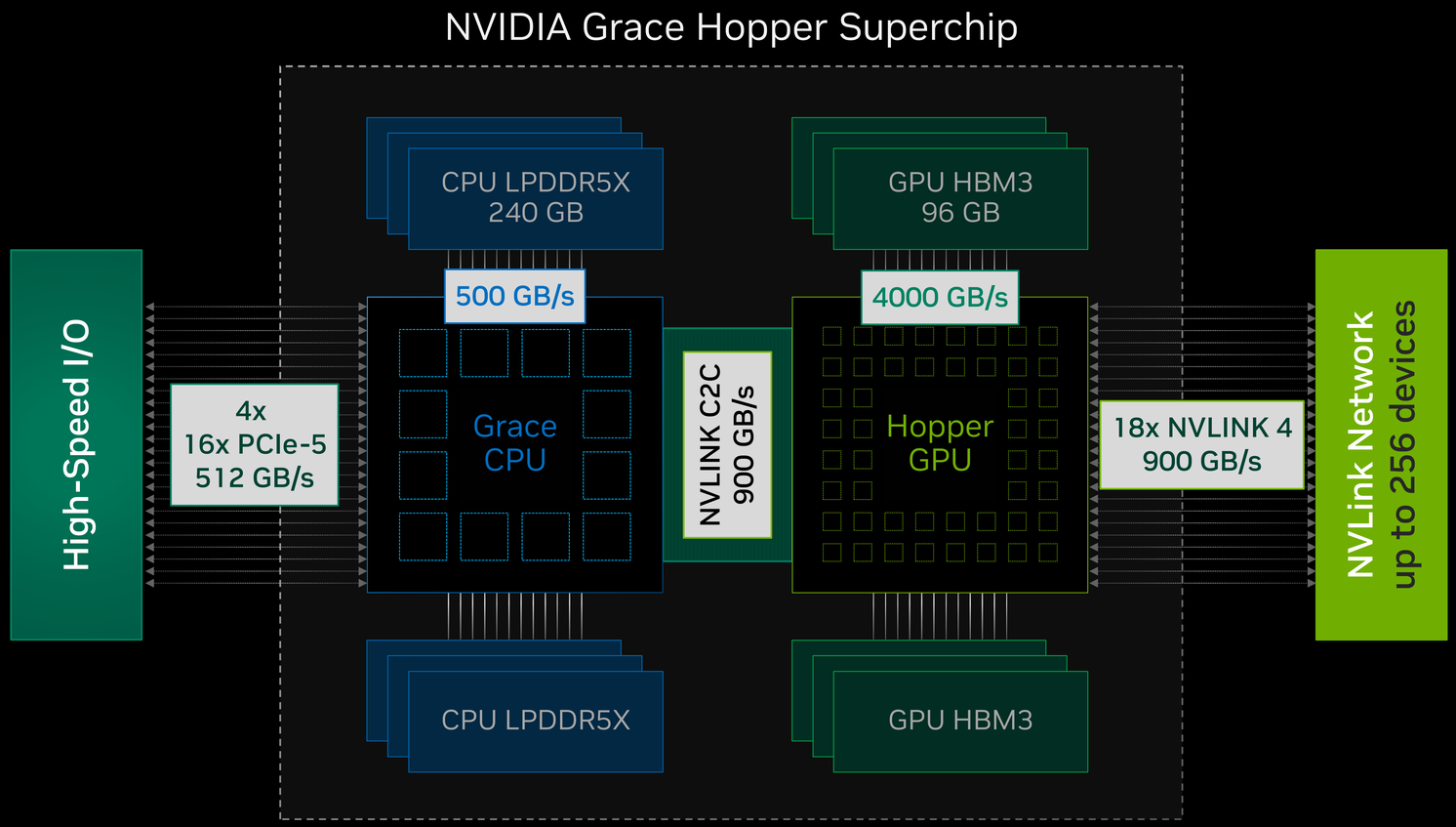


Grace/Hopper Superchip



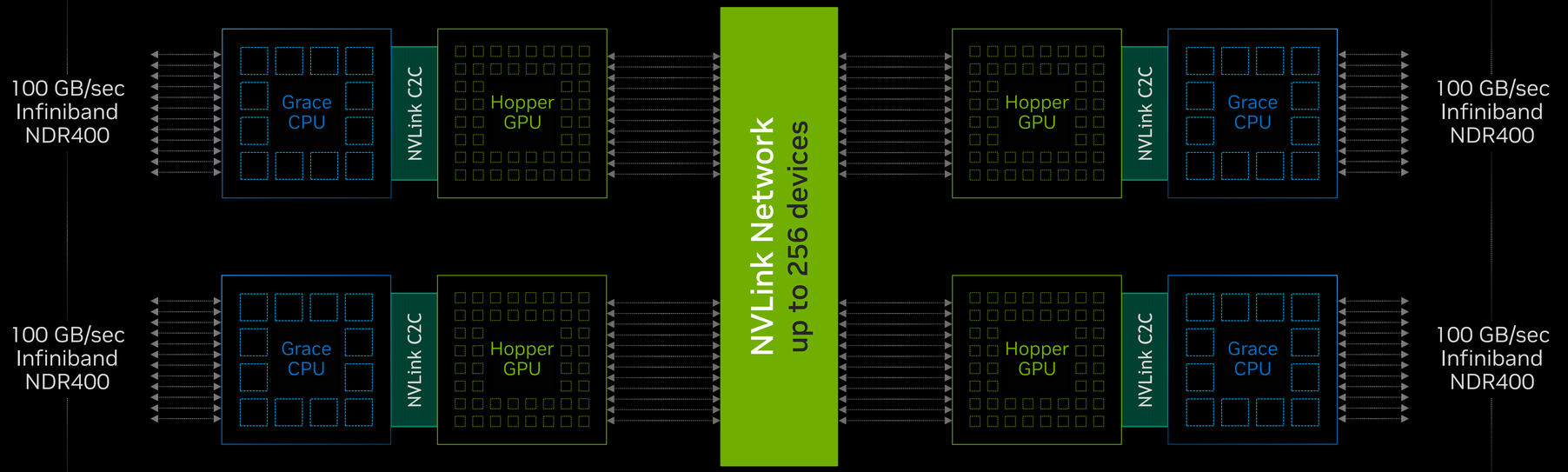
… computing is a not a chip problem
It's a software and chip problem

Grace/Hopper Superchip



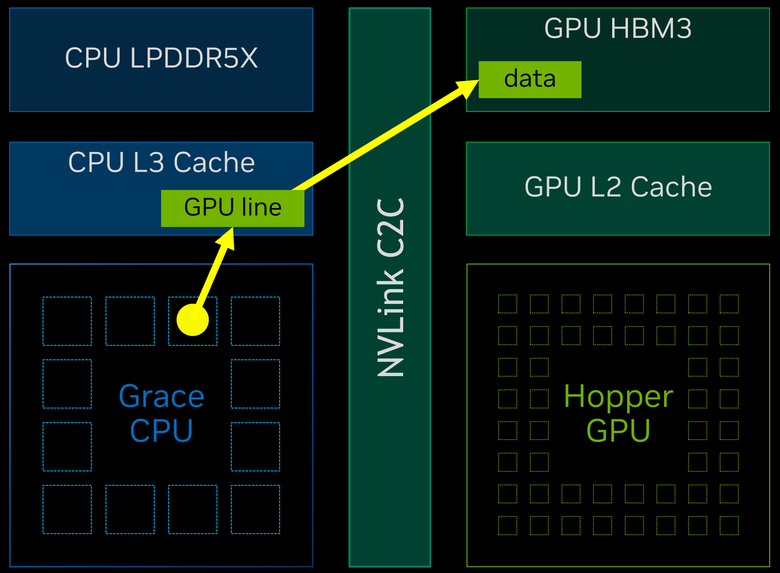
Accelerating HPC applications with ISO C++ on Grace Hopper [S51054]

NVLink Connects Up To 256 Superchips



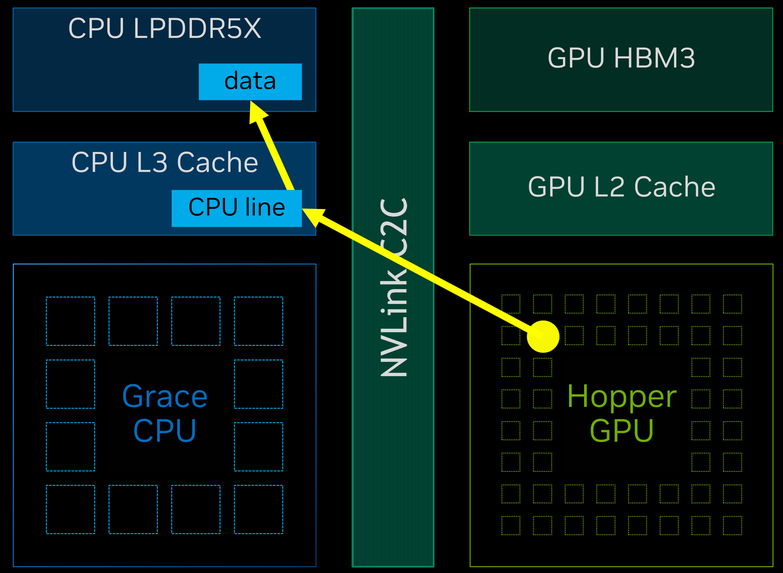
Global Access to All Data

Cache-coherent access via NVLink C2C from either processor to either physical memory



Grace directly reading Hopper's memory

CPU fetches GPU data into CPU L3 cache
Cache remains **coherent** with GPU memory
Changes to GPU memory **evict** cache line

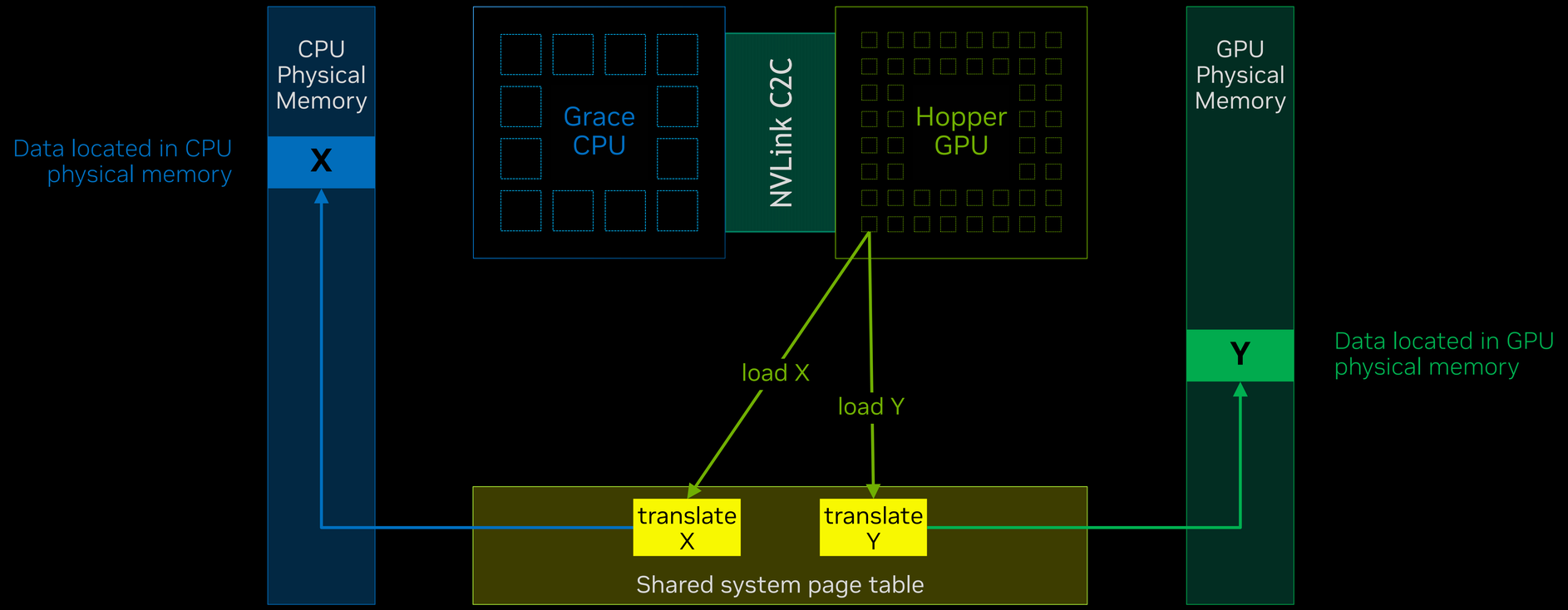


Hopper directly reading Grace's memory

GPU loads CPU data via CPU L3 cache
CPU and GPU **can both hit** on cached data
Changes to CPU memory **update** cache line

Grace/Hopper Unified Memory

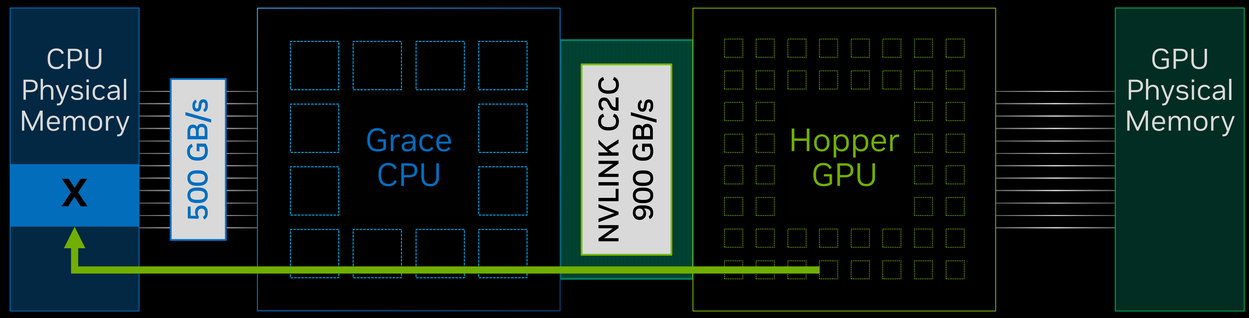
Address Translation Service (ATS) allows full access to all CPU & GPU allocations



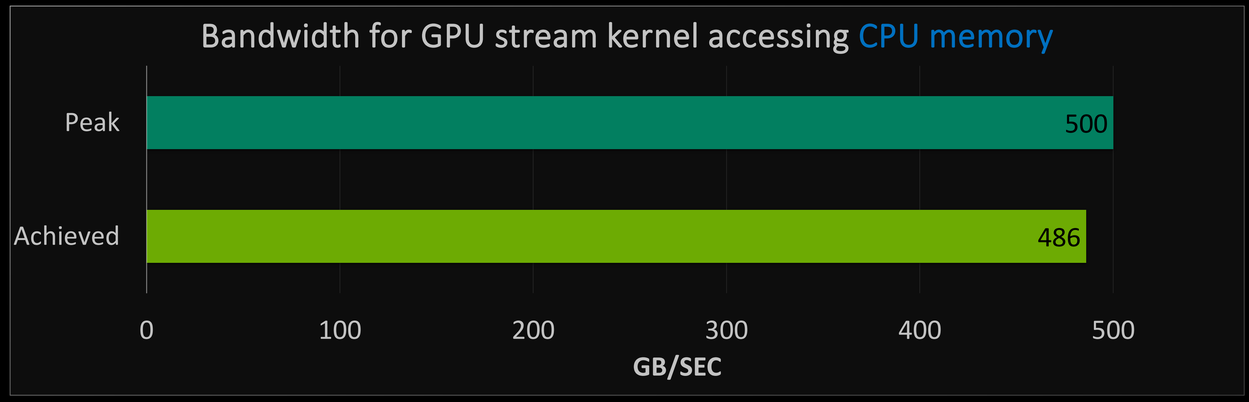
ATS creates a single page table for the whole system

NVLink C2C allows access to all physical memory **without migration**

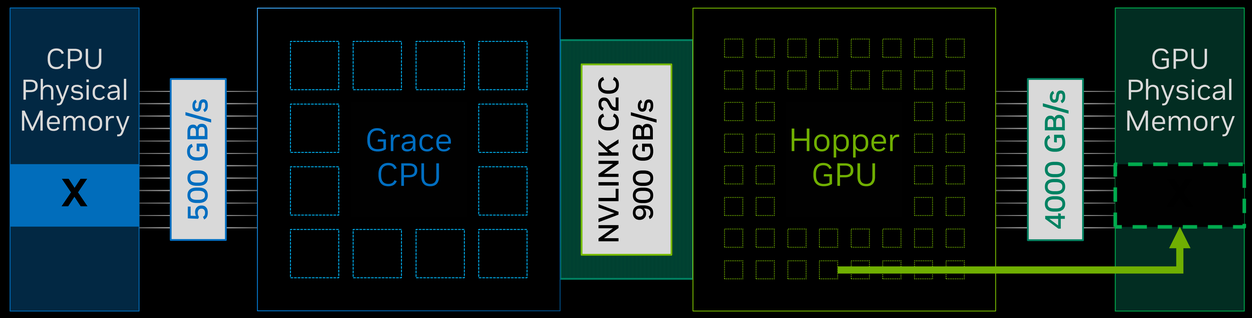
High Bandwidth Memory Access & Automatic Data Migration



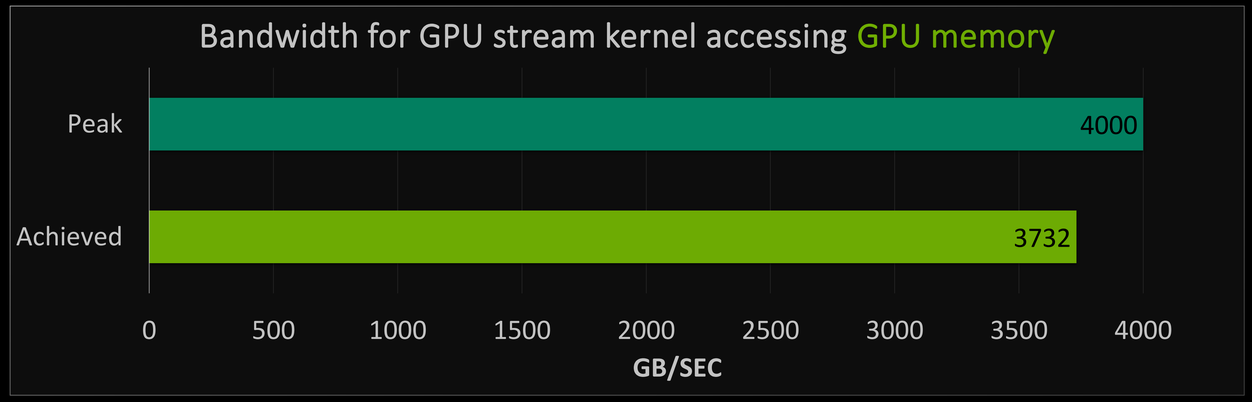
Hopper can access Grace memory
at **full CPU memory speed** of 500 GB/sec



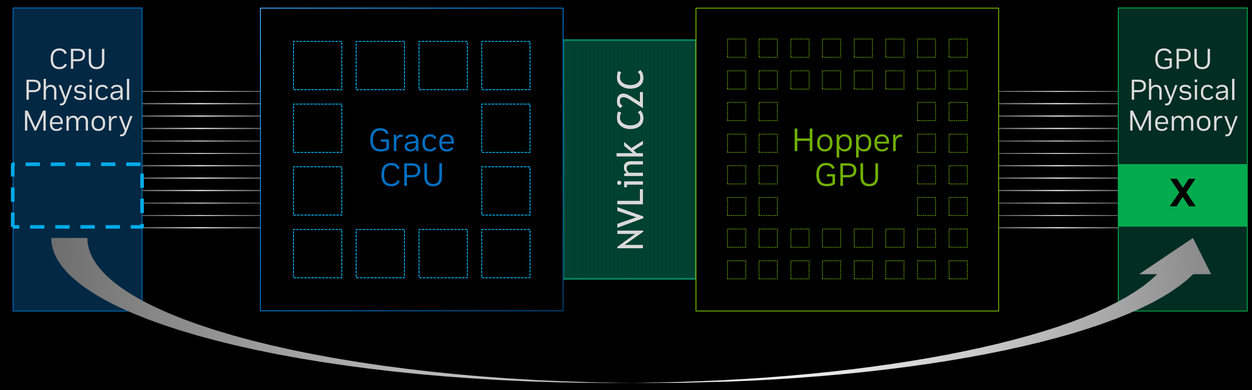
High Bandwidth Memory Access & Automatic Data Migration



But Hopper can access its own memory
at **full HBM speed** of 4000 GB/sec

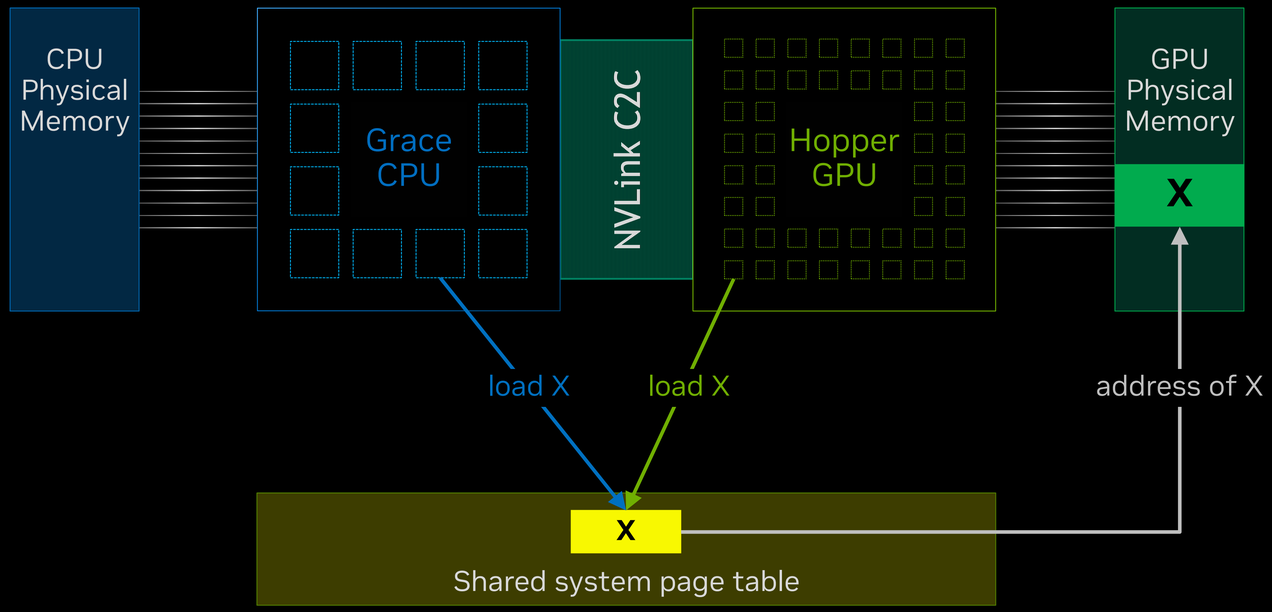


High Bandwidth Memory Access & Automatic Data Migration



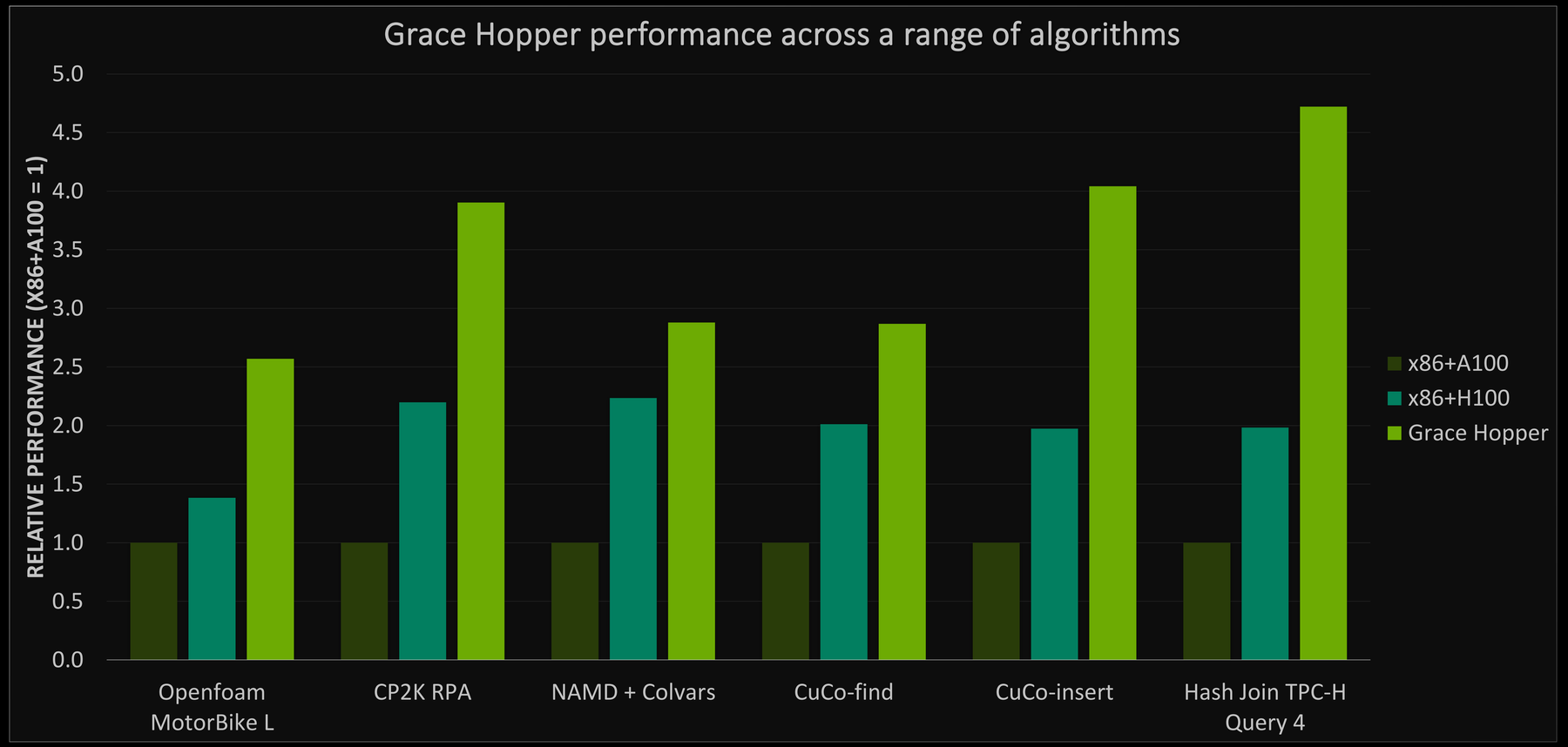
The system can **automatically** migrate
both managed and CPU-allocated memory
in order to optimize access speed

High Bandwidth Memory Access & Automatic Data Migration



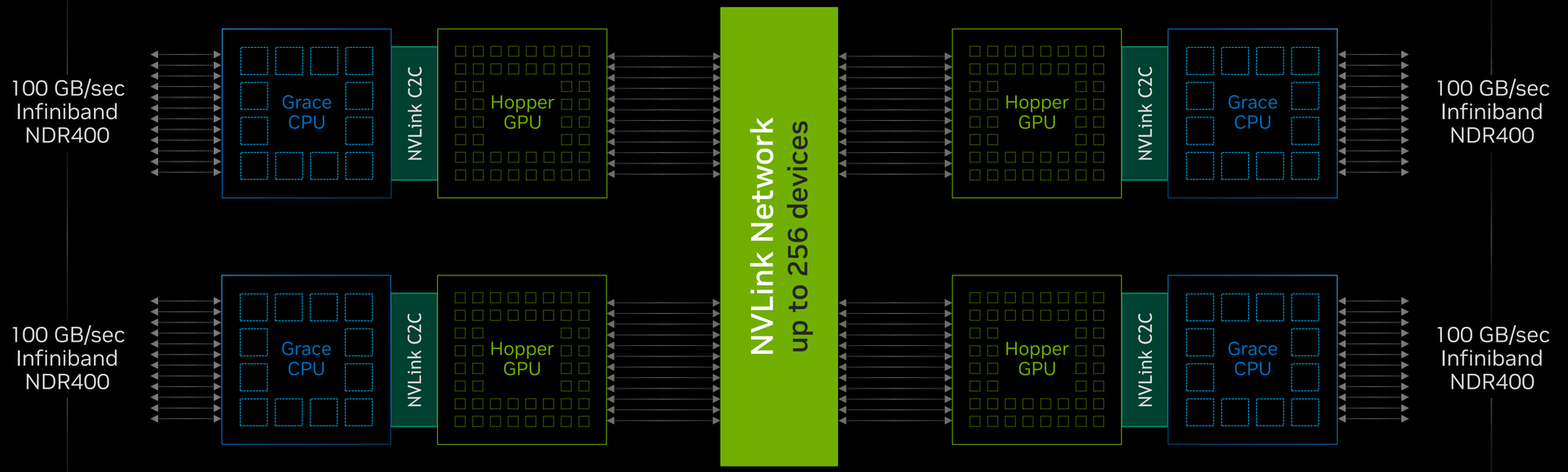
ATS shared page table means that both CPU and GPU automatically access X in its new location after migration

Grace/Hopper SuperChip Performance On Different Workloads

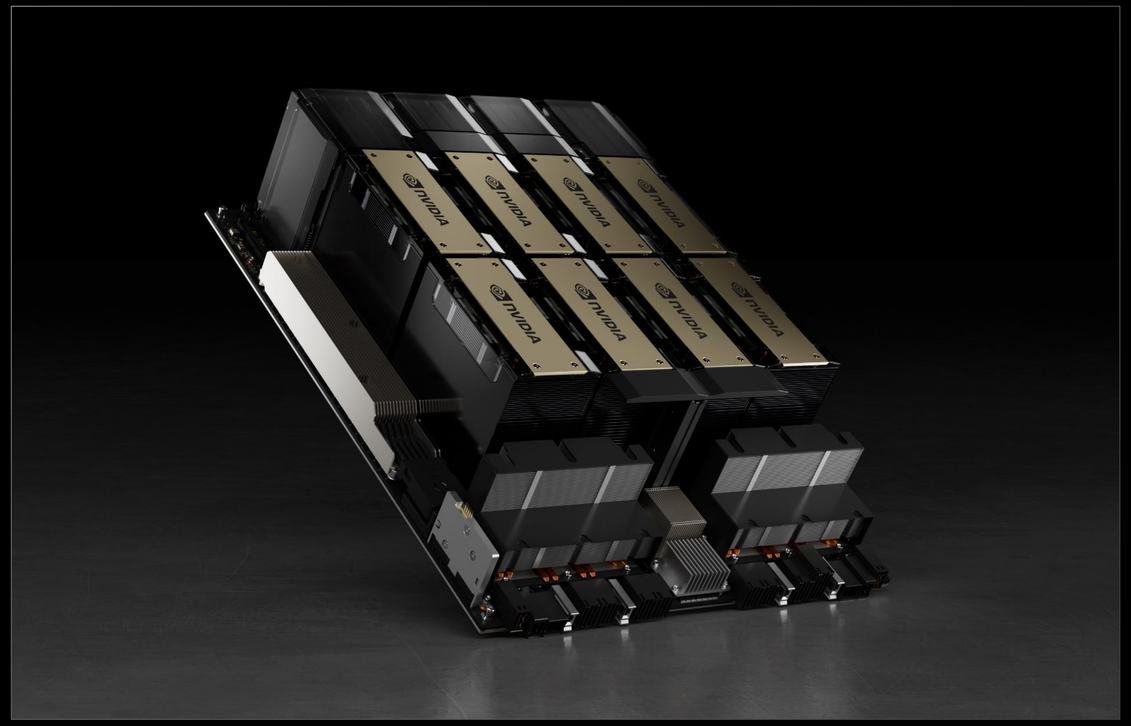


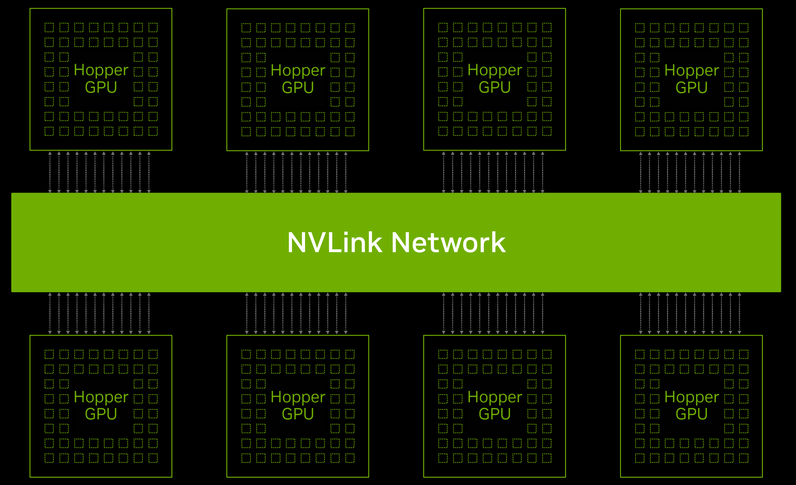
Programming Model and Applications for the Grace Hopper Superchip [S51120]

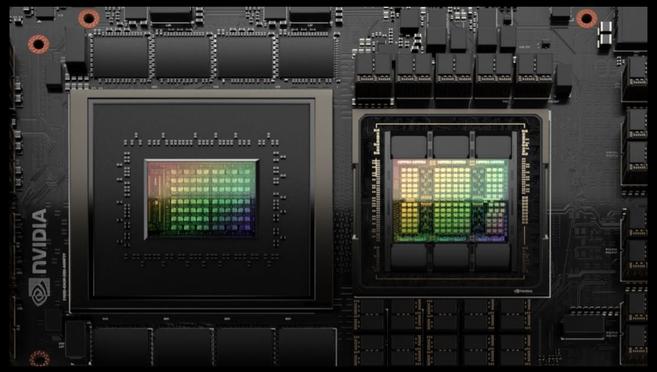
NVLink Connects Up To 256 Superchips



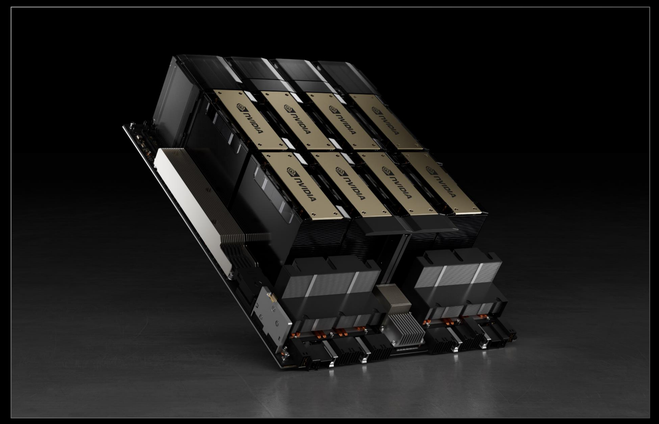
Multi-Chip Systems







Multi-die



Multi-chip

Multi-node

There Is No Future That Is Not Multi-Node

What is CUDA?

CUDA C++

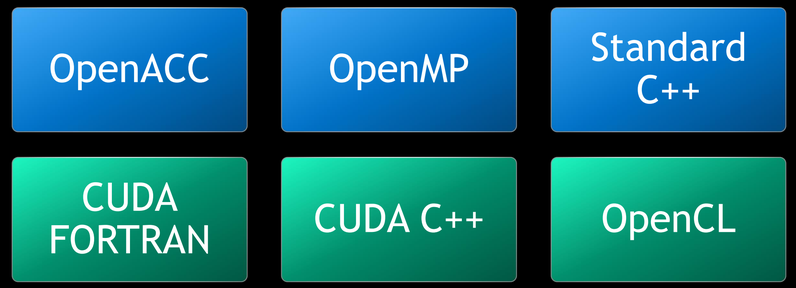
CUDA is More Than CUDA C++

CUDA FORTRAN

CUDA C++

OpenCL

Many Standard Languages Support GPU Computing

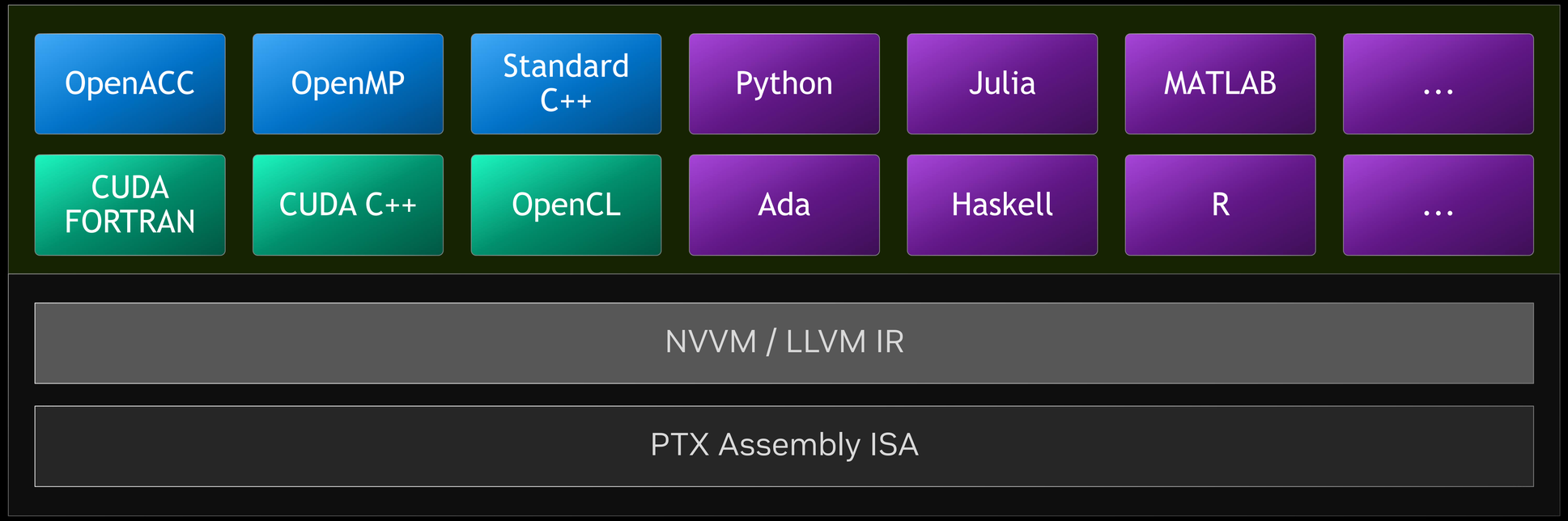


C++ Standard Parallelism [S51755]

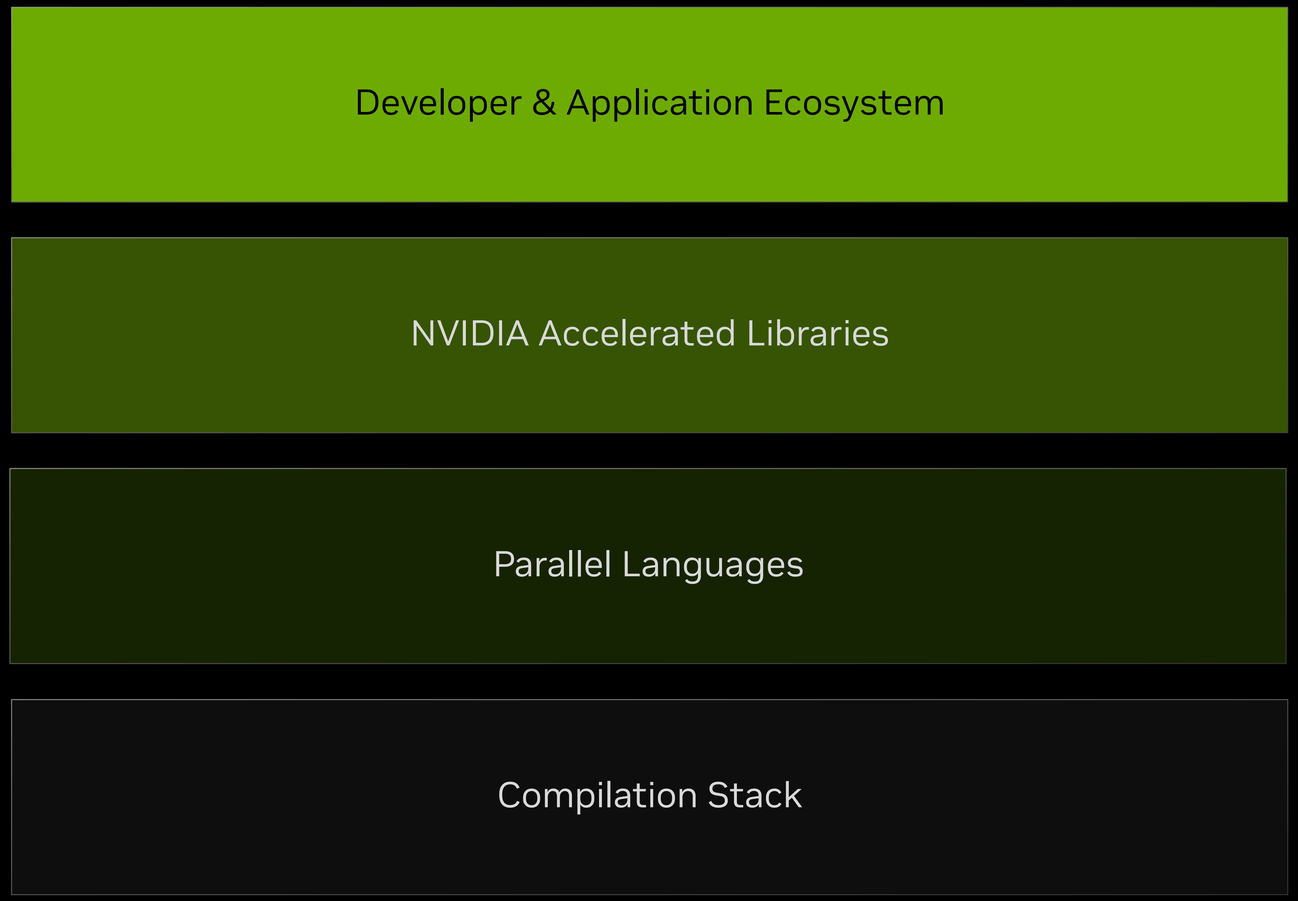
Along With Many Languages Not Supported Directly by NVIDIA

OpenACC	OpenMP	Standard C++	Python	Julia	MATLAB	…
CUDA FORTRAN	CUDA C++	OpenCL	Ada	Haskell	R	…

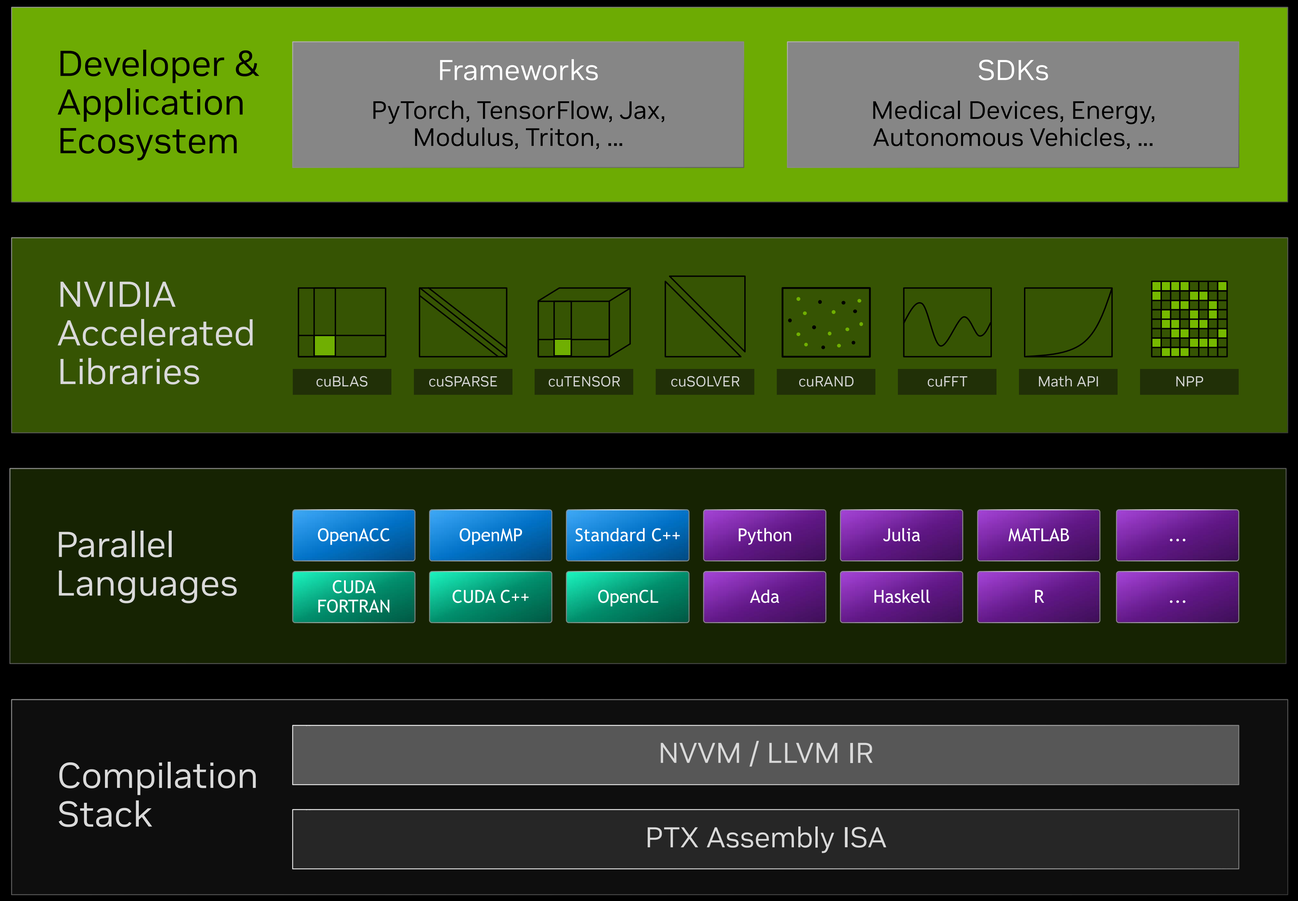
CUDA is a Platform for GPU Computing



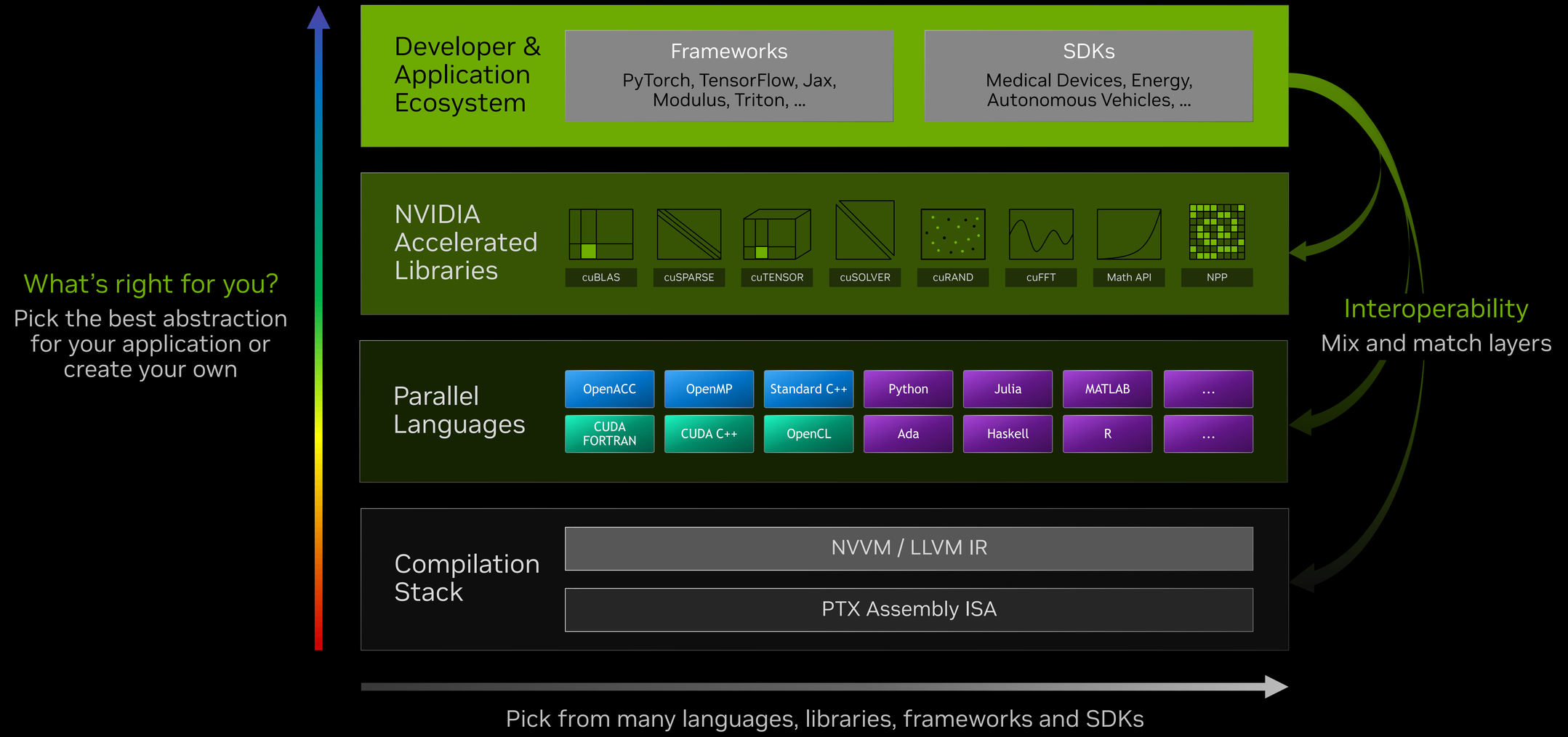
CUDA’s Second Dimension: A Platform of Abstraction Layers



Target the Abstraction Layer That Works Best for Your Application



An Ecosystem With a Spectrum of Choices



How to Write a CUDA Program [S51210]

NVIDIA

Greater productivity

Greater control

The Third Dimension is Scale

Multi-Node Programming Models

Intersect Where it Makes the Most Sense for Your Application

cuNumeric

Implicitly parallel implementations of NumPy APIs

Fully implicit parallelism, transparent to the application

Greater productivity

Greater control

cuNumeric | Legate | Legion | Multi-GPU Libraries | NCCL | CUDA-Aware MPI | NVSHMEM

Multi-Node Programming Models

Intersect Where it Makes the Most Sense for Your Application

cuNumeric

Implicitly parallel implementations of NumPy APIs

Fully implicit parallelism, transparent to the application

NVSHMEM

GPU implementation of OpenSHMEM

Explicitly parallel, load/store to partitioned global address space

Greater productivity

Greater control

cuNumeric | Legate | Legion | Multi-GPU Libraries | NCCL | CUDA-Aware MPI | NVSHMEM

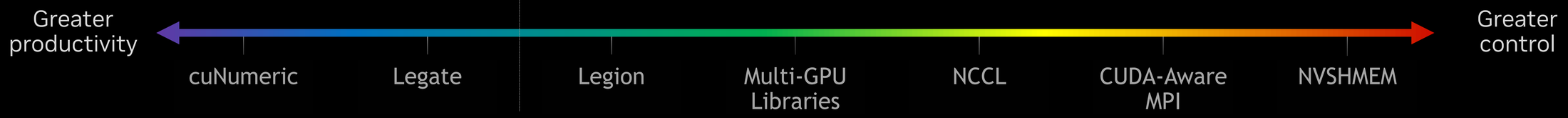
Multi-Node Programming Models

How to Streamline Shared Memory Space With the NVSHMEM Communication Library [S51705]

Developing For Multi-Node Systems

Implicit parallelism

The same code must be **performance portable** to **any accelerator types** and **any machine sizes**



Multi-Node Programming Models

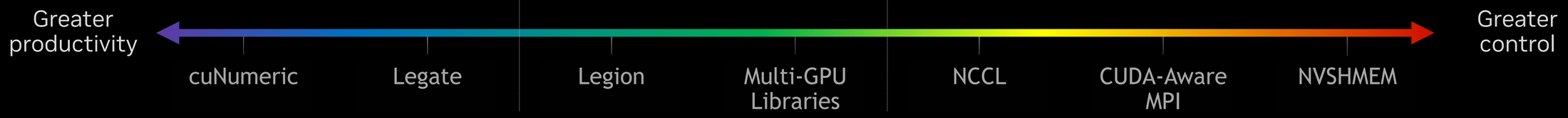
Developing For Multi-Node Systems

Implicit parallelism

The same code must be **performance portable** to **any accelerator types** and **any machine sizes**

Managed parallelism

Applications either provide **source-code mark up** or call into **scalable libraries**, with scaling handled **at runtime**



Multi-Node Programming Models

Developing For Multi-Node Systems

Implicit parallelism

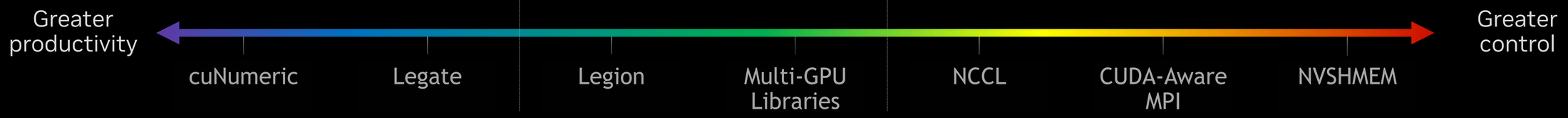
The same code must be **performance portable** to **any accelerator types** and **any machine sizes**

Managed parallelism

Applications either provide **source-code mark up** or call into **scalable libraries**, with scaling handled **at runtime**

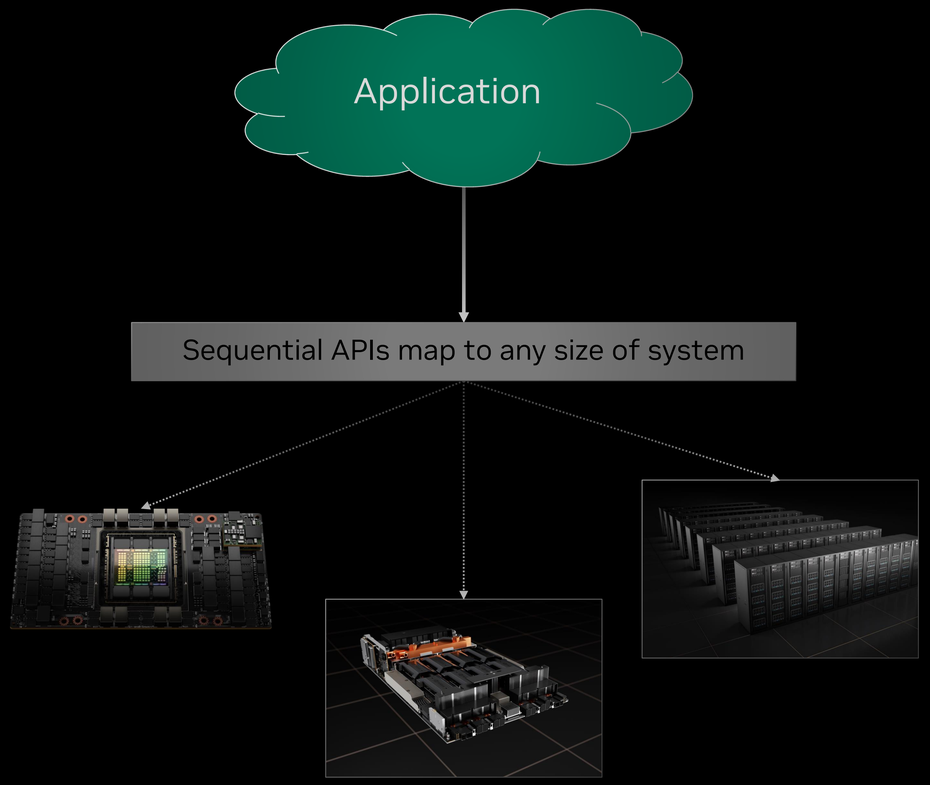
Explicit parallelism

Application **directly manages** data exchange & distribution **by hand** or **via communication APIs**

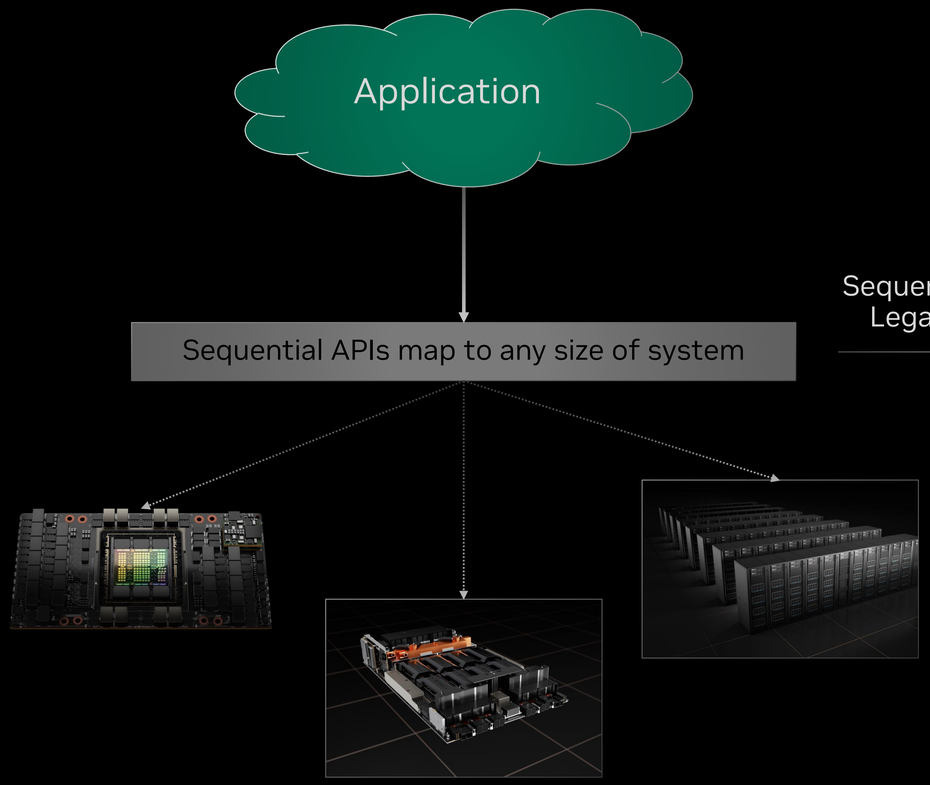


Multi-Node Programming Models

What is “Implicit Parallelism”?



Legate: A Framework for Implicit Parallelism



Sequential API calls
Legate runtime

Legate-Enabled Libraries
Implicitly scaled by Legate runtime system

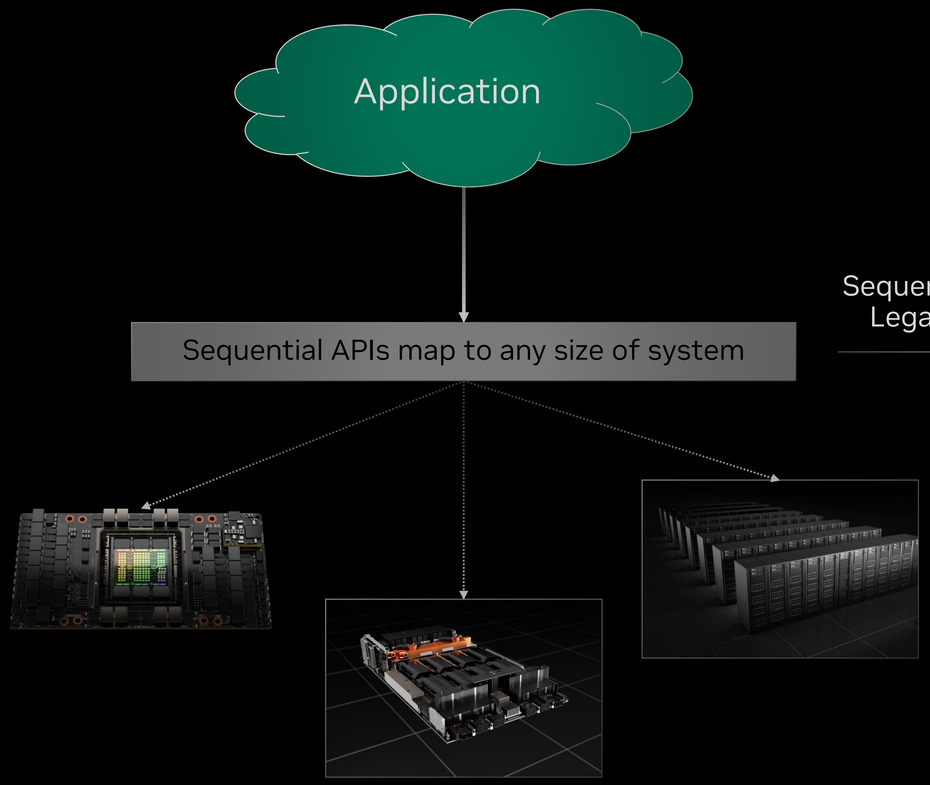
Legate

Productivity / Composability Layer
Accelerates library development

Common Runtime System
Scalable extraction of implicit parallelism

Accelerated Domain Libraries
Maximise single-accelerator performance

Legate: A Framework for Implicit Parallelism



Sequential API calls
Legate runtime

Legate

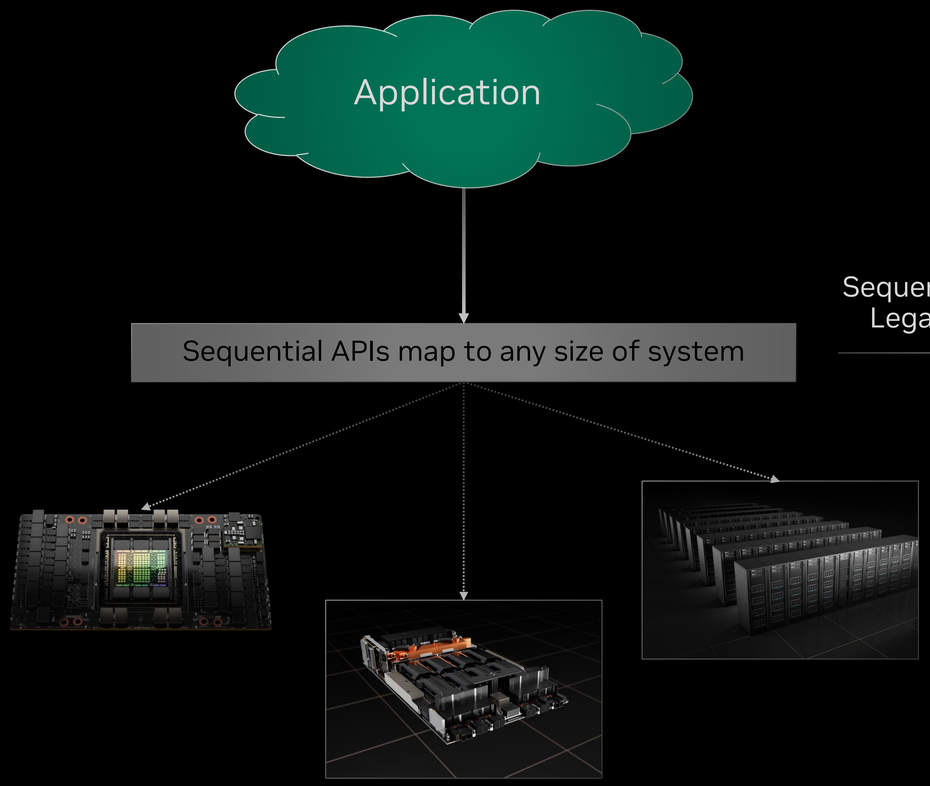
Legate-Enabled Libraries
Implicitly scaled by Legate runtime system

Productivity / Composability Layer
Accelerates library development

Common Runtime System
Scalable extraction of implicit parallelism

Accelerated Domain Libraries
Maximise single-accelerator performance

Legate: A Framework for Implicit Parallelism



Sequential API calls
Legate runtime

Legate

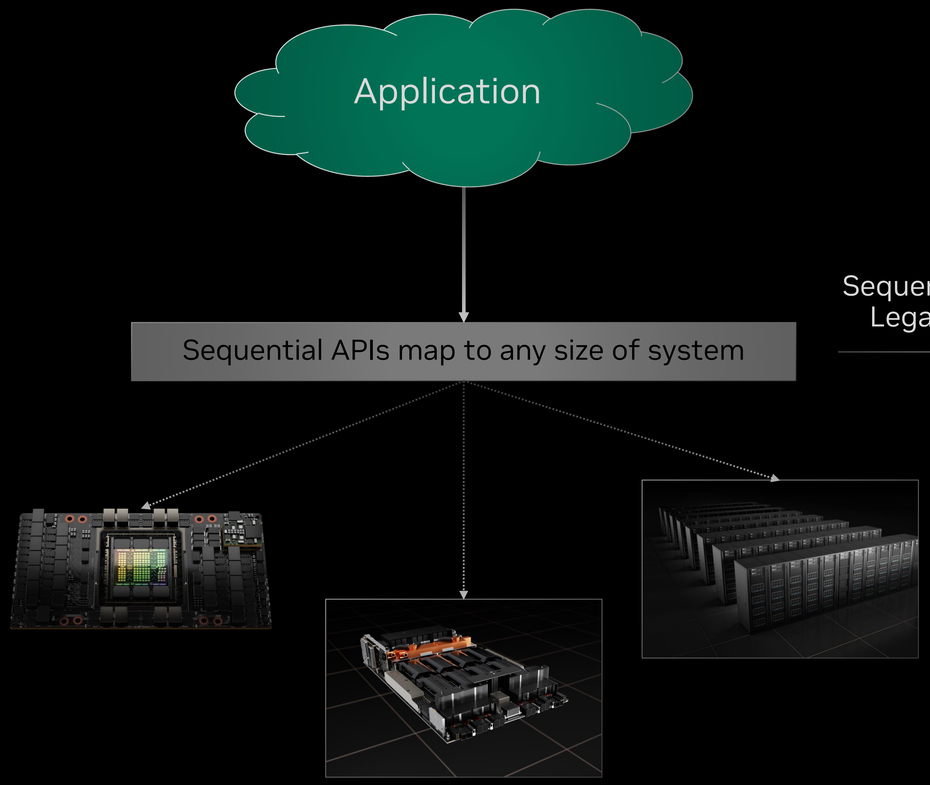
Legate-Enabled Libraries
Implicitly scaled by Legate runtime system

Productivity / Composability Layer
Accelerates library development

Common Runtime System
Scalable extraction of implicit parallelism

Accelerated Domain Libraries
Maximise single-accelerator performance

Legate: A Framework for Implicit Parallelism



Sequential API calls
Legate runtime

Legate

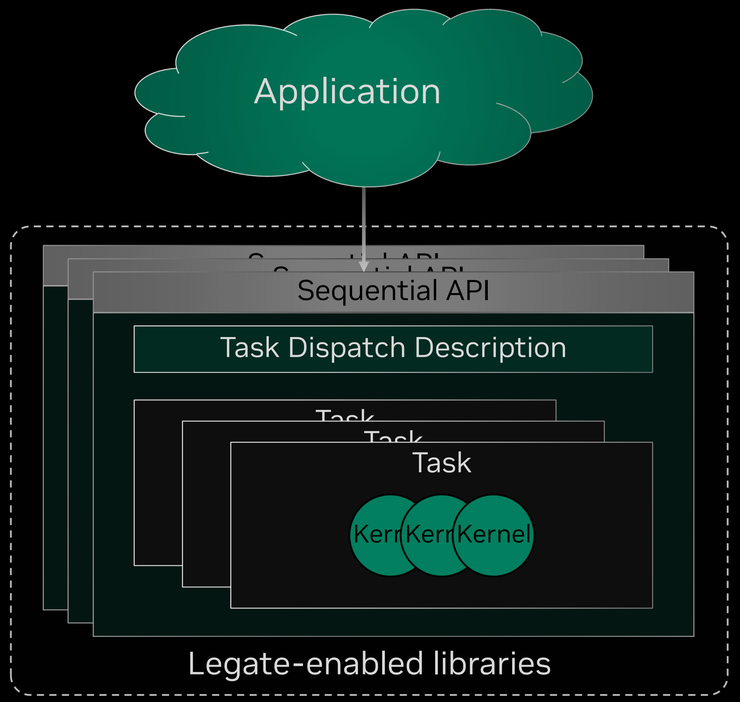
Legate-Enabled Libraries
Implicitly scaled by Legate runtime system

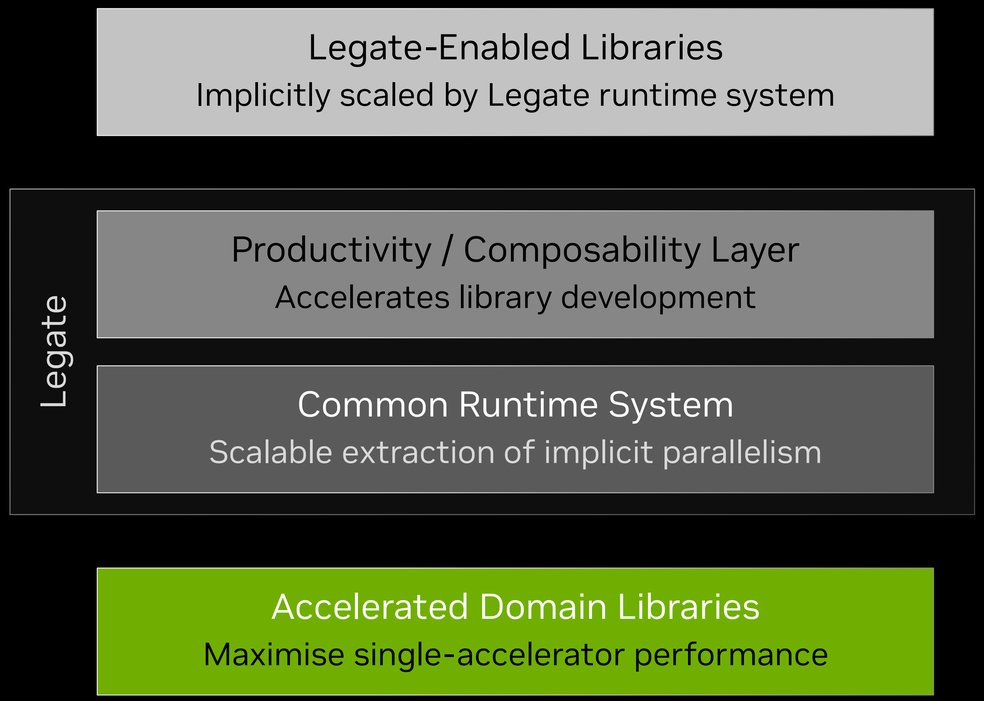
Productivity / Composability Layer
Accelerates library development

Common Runtime System
Scalable extraction of implicit parallelism

Accelerated Domain Libraries
Maximise single-accelerator performance

Legate: A Task-Based Model





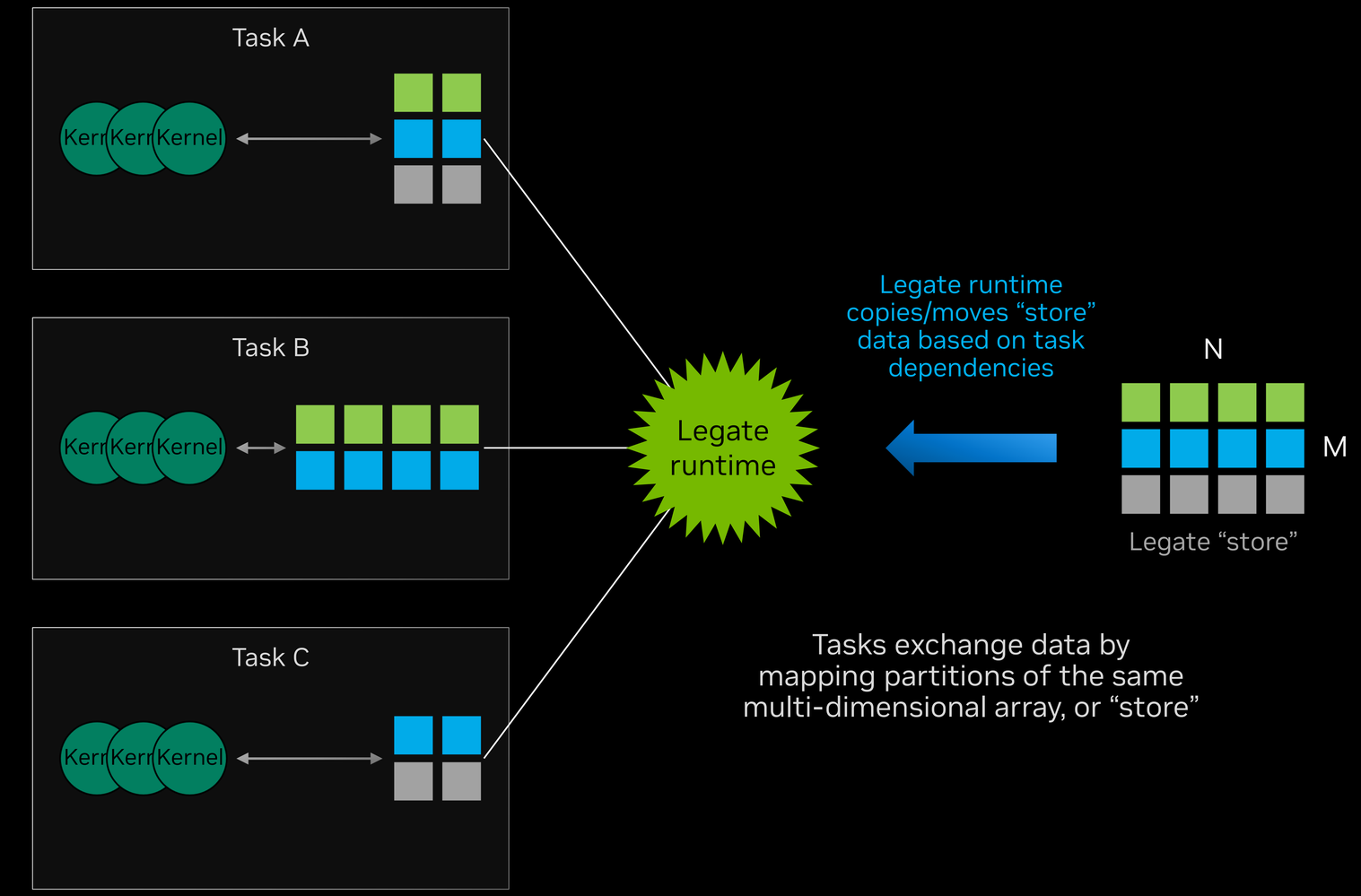
Legate-enabled libraries

Are **free** from explicit synchronization

Are **free** from explicit data movement

Are thereby **composable** with each other

Unified Data Abstraction for True Composability



Tasks exchange data by mapping partitions of the same multi-dimensional array, or “store”

A task may map multiple stores but for convenience a single store is shown here

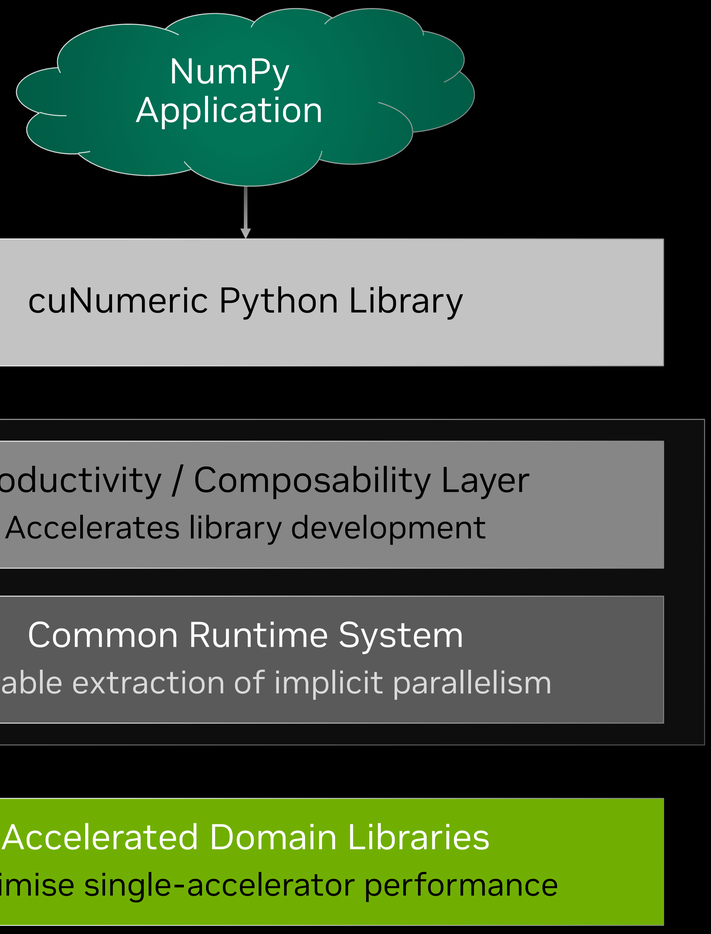
cuNumeric – Implicitly Parallel Implementations of NumPy APIs

Stencil Benchmark

No modifications required to scale to a thousand GPUs

numeric/blob/branch-23.03/examples/stencil.py

```
def run_stencil(N, I, warmup, timing):  # noqa: E741
    grid = initialize(N)

    print("Running Jacobi stencil...")
    center = grid[1:-1, 1:-1]
    north = grid[0:-2, 1:-1]
    east = grid[1:-1, 2:]
    west = grid[1:-1, 0:-2]
    south = grid[2:, 1:-1]

    timer.start()
    for i in range(I + warmup):
        if i == warmup:
            timer.start()
        average = center + north + east + west + south
        work = 0.2 * average
        center[:] = work
    total = timer.stop()

    if timing:
        print(f"Elapsed Time: {total} ms")
    return total
```



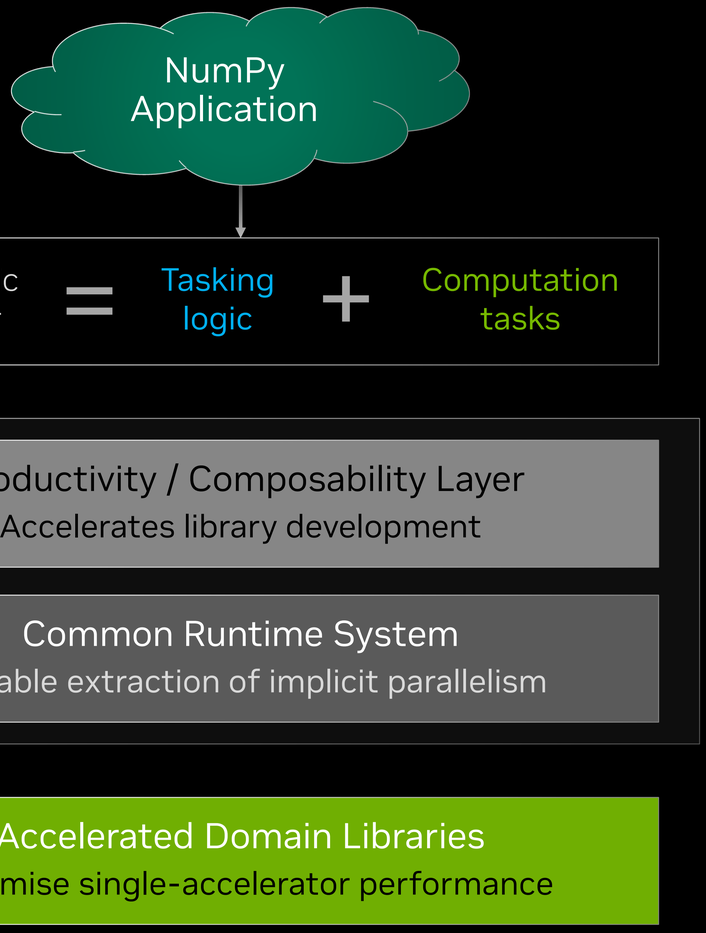
cuNumeric – Implicitly Parallel Implementations of NumPy APIs

Stencil Benchmark

No modifications required to scale to a thousand GPUs

numeric/blob/branch-23.03/examples/stencil.py

```
32
33  def run_stencil(N, I, warmup, timing):  # noqa: E741
34      grid = initialize(N)
35
36      print("Running Jacobi stencil...")
37      center = grid[1:-1, 1:-1]
38      north = grid[0:-2, 1:-1]
39      east = grid[1:-1, 2:]
40      west = grid[1:-1, 0:-2]
41      south = grid[2:, 1:-1]
42
43      timer.start()
44      for i in range(I + warmup):
45          if i == warmup:
46              timer.start()
47          average = center + north + east + west + south
48          work = 0.2 * average
49          center[:] = work
50      total = timer.stop()
51
52      if timing:
53          print(f"Elapsed Time: {total} ms")
54      return total
```



cuNumeric – Implicitly Parallel Implementations of NumPy APIs

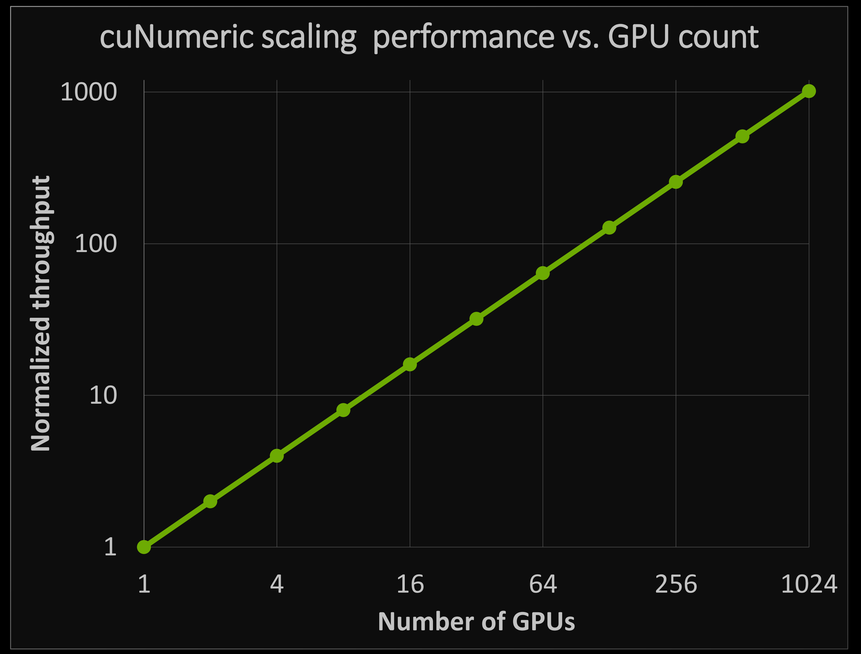
cuNumeric Beta: out March 2023

Stencil Benchmark

No modifications required to scale to a thousand GPUs

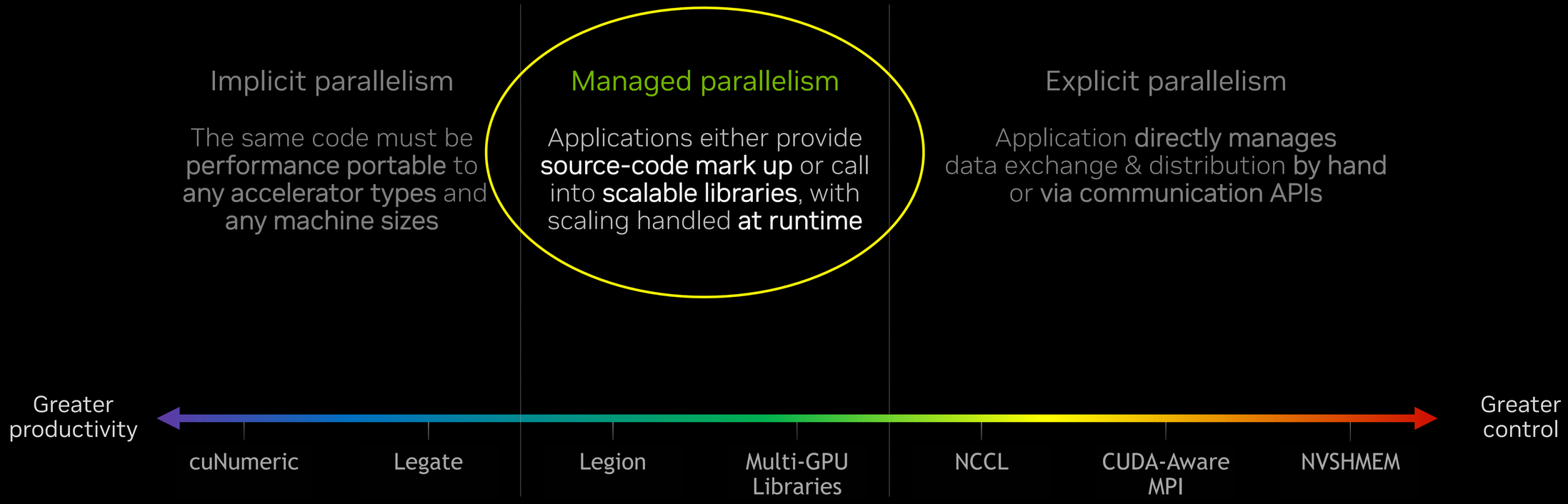
numeric/blob/branch-23.03/examples/stencil.py

```
32
33  def run_stencil(N, I, warmup, timing):  # noqa: E741
34      grid = initialize(N)
35
36      print("Running Jacobi stencil...")
37      center = grid[1:-1, 1:-1]
38      north = grid[0:-2, 1:-1]
39      east = grid[1:-1, 2:]
40      west = grid[1:-1, 0:-2]
41      south = grid[2:, 1:-1]
42
43      timer.start()
44      for i in range(I + warmup):
45          if i == warmup:
46              timer.start()
47          average = center + north + east + west + south
48          work = 0.2 * average
49          center[:] = work
50      total = timer.stop()
51
52      if timing:
53          print(f"Elapsed Time: {total} ms")
54      return total
```



cuNumeric and Legate: How to Create a Distributed GPU Accelerated Library [S51789]

Managed Parallelism Through Accelerated Libraries



Multi-GPU Multi-Node (MGMN) Libraries

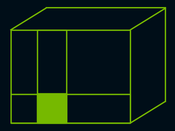
NVIDIA Accelerated Libraries

Host Libraries
cuBLAS, cuFFT, cuSOLVER, cuTENSOR, NPP, cuRAND, ...

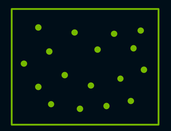
Device Libraries
cuFFTDx, cuBLASDx, CUDA Math API, CUTLASS, ...


cuBLAS


cuSPARSE

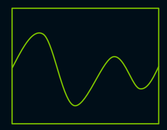

cuTENSOR

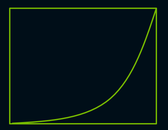

cuSOLVER


cuRAND


cuSOLVERMp

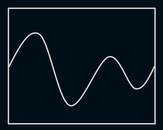

nvJPEG


cuFFT


CUDA Math API


NPP


cuQuantum


cuFFTMp

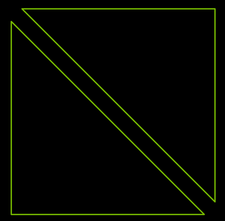
cuSOLVERMp: Distributed Symmetric Eigenvalue Solver

Local matrix size per GPU (32768^2)

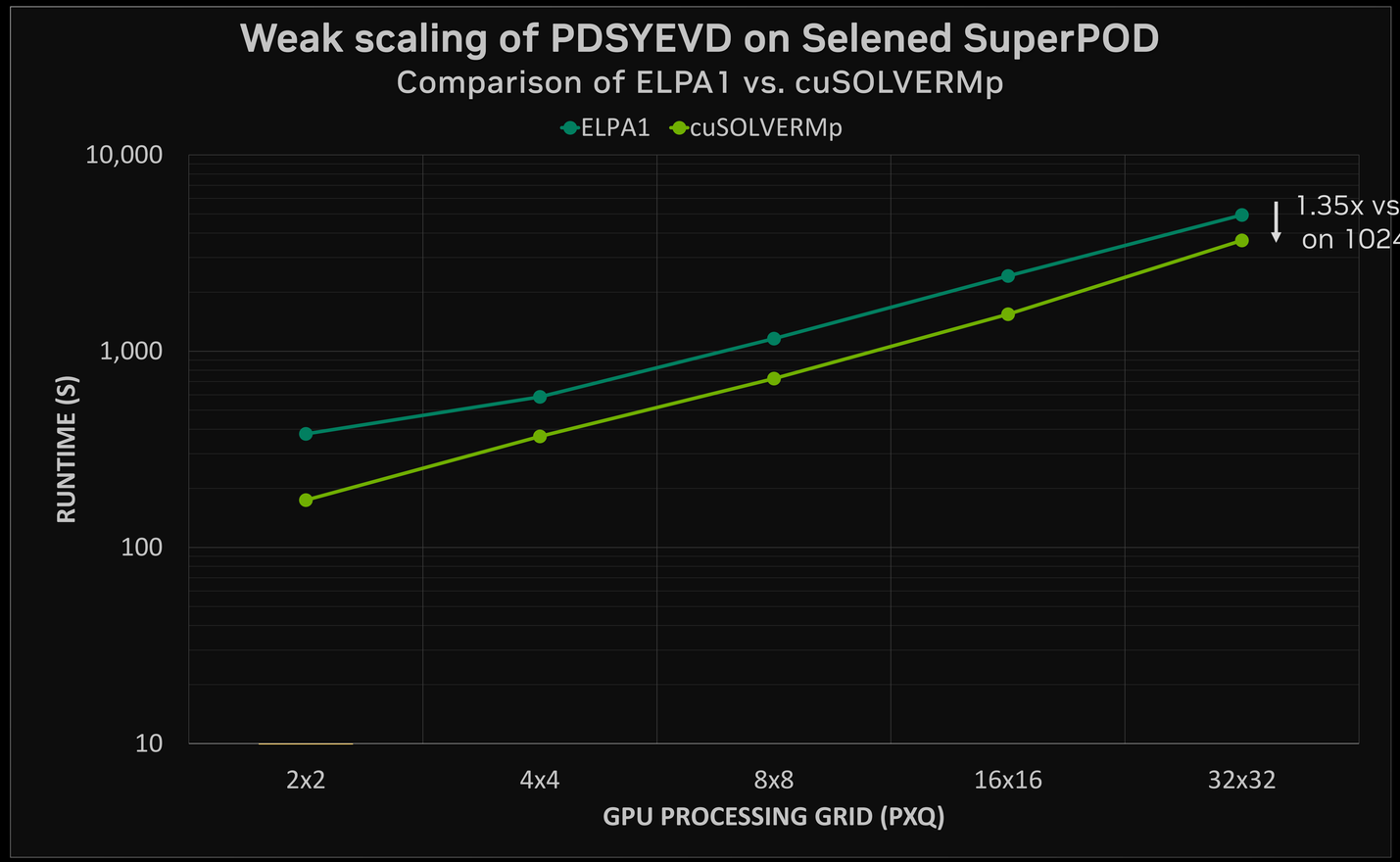
Double precision real data

Speedup of ~1.6x over ELPA on
130K, 262K, 524K sizes

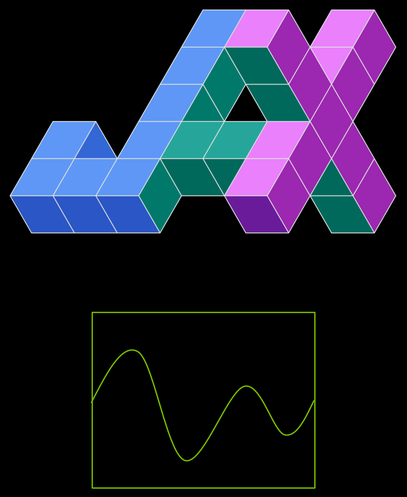
Speedup of 1.35x on $1M^2$ problem

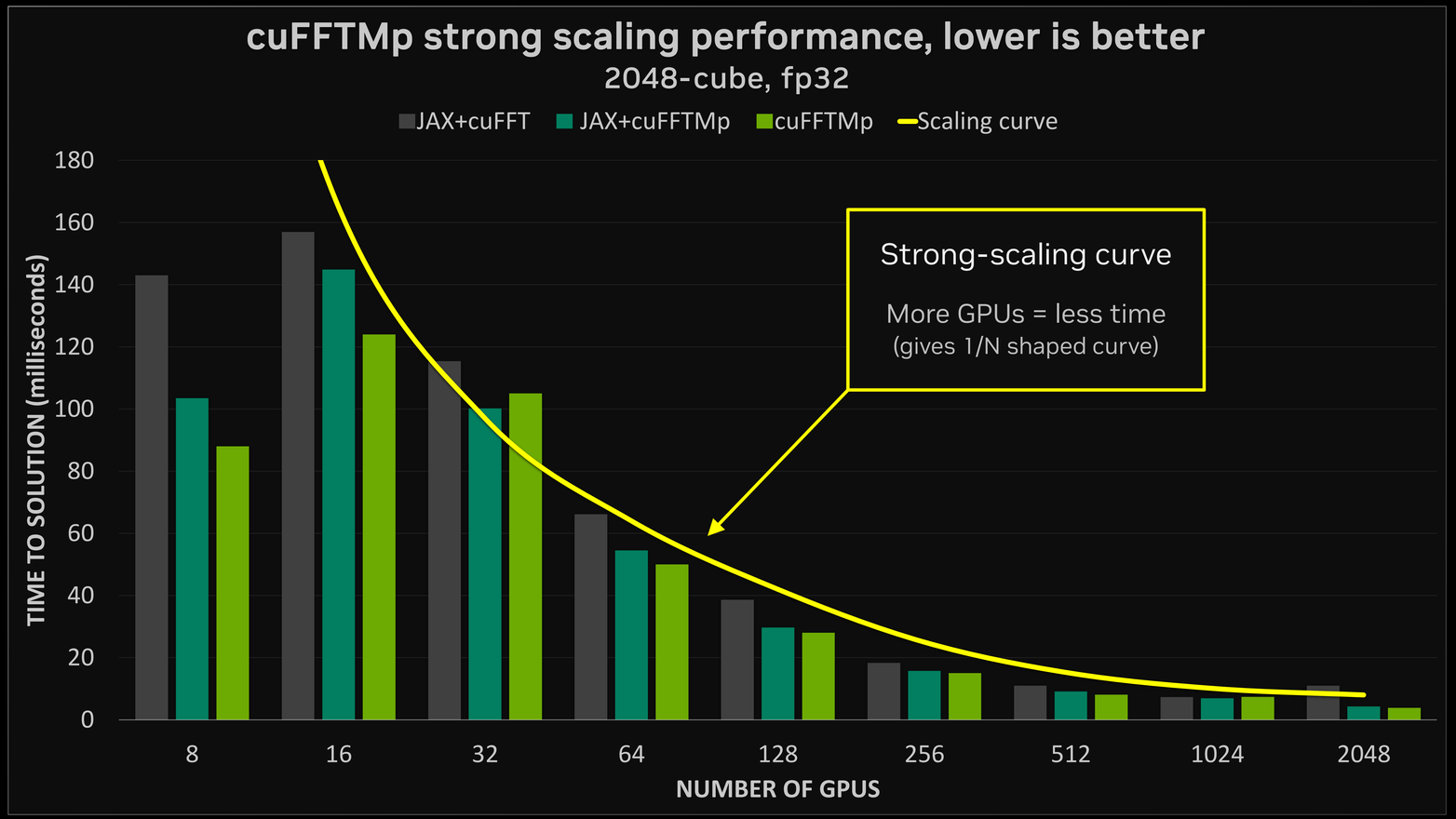


cuSOLVERMp

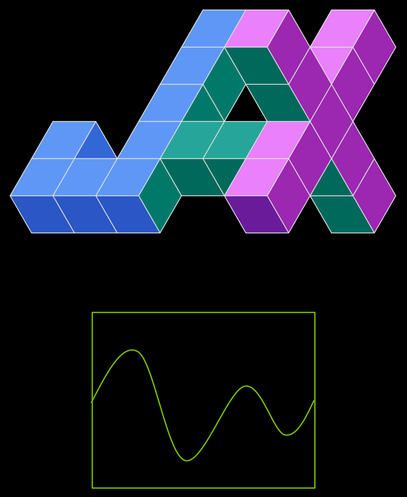


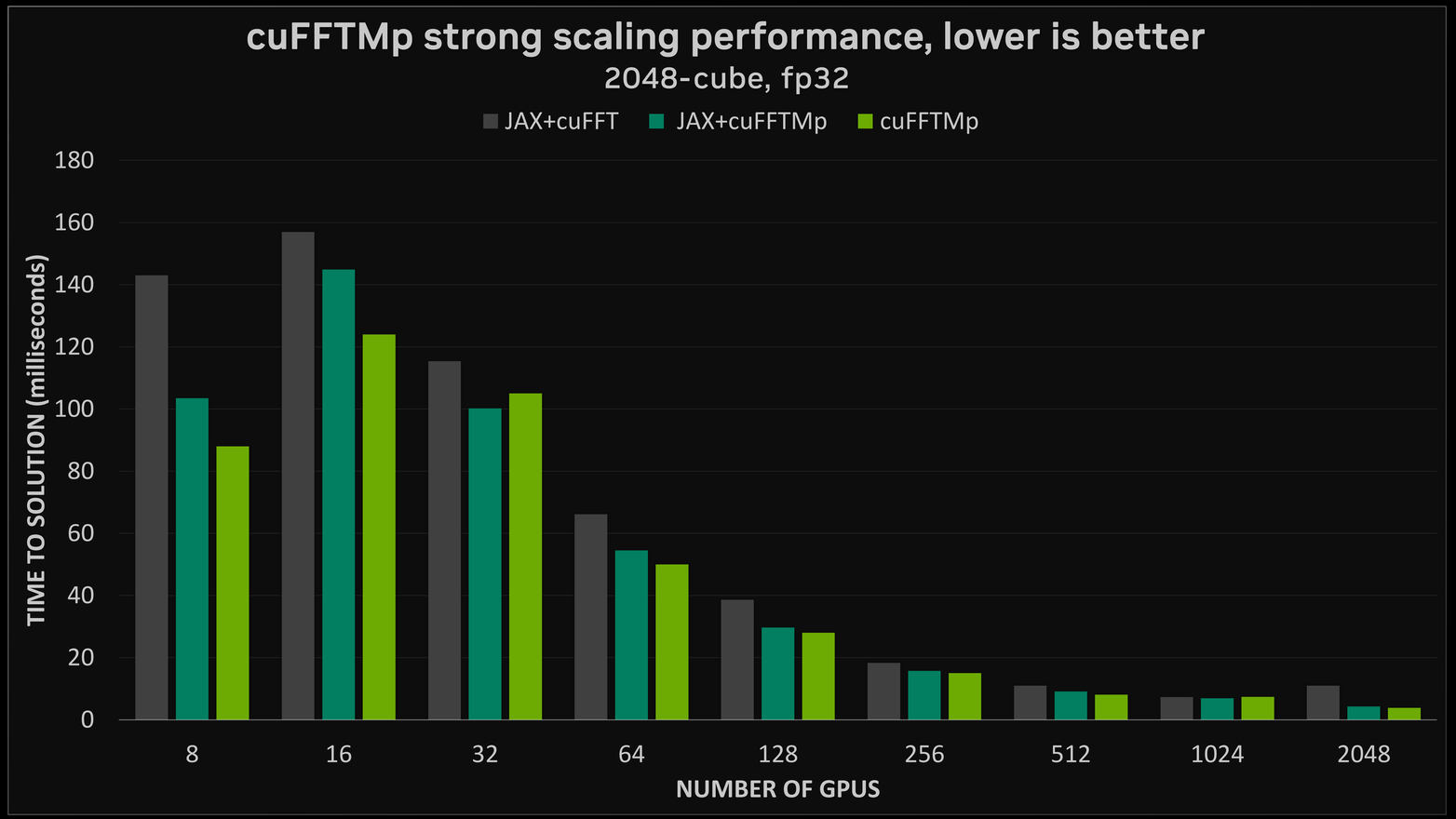
cuFFTMp + JAX


cuFFTMp

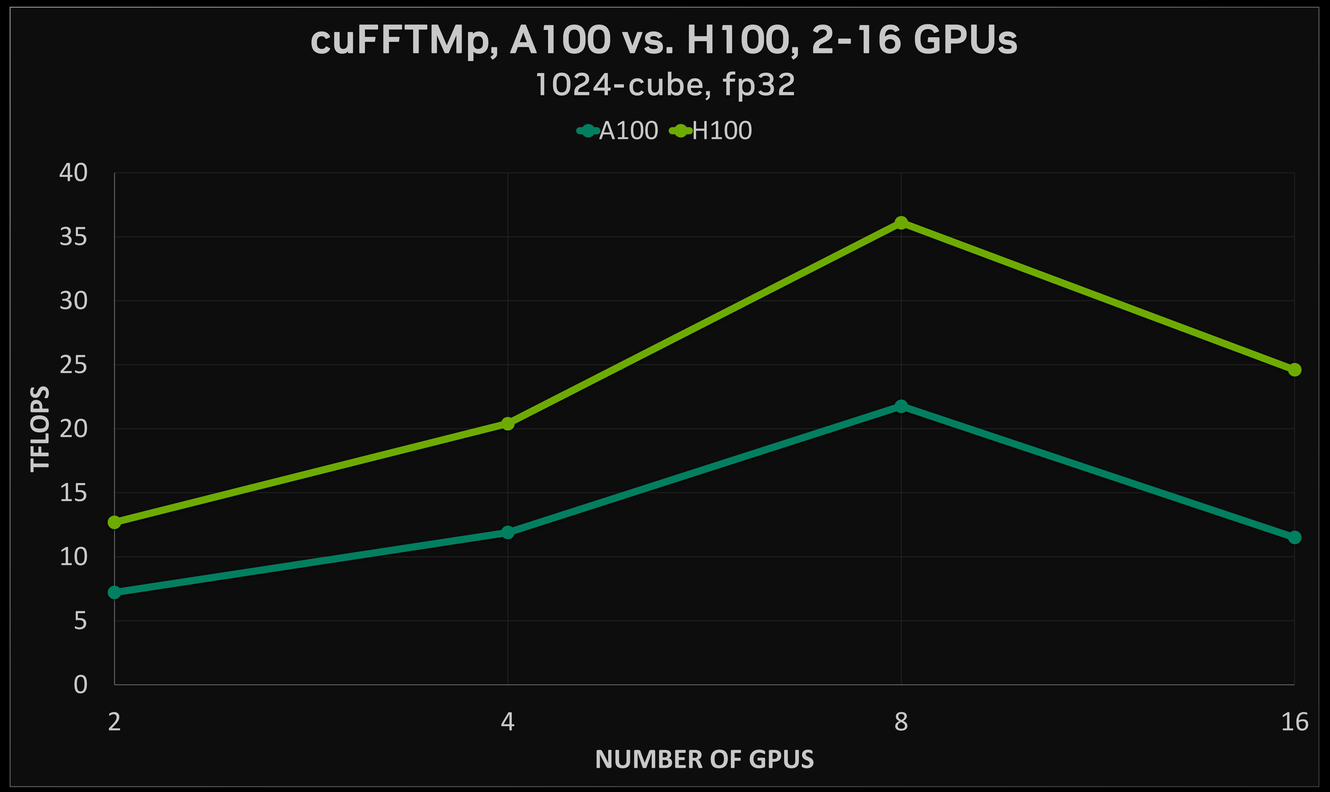

cuFFTMp strong scaling performance, lower is better
2048-cube, fp32
JAX+cuFFT
JAX+cuFFTMp
cuFFTMp
Scaling curve
TIME TO SOLUTION (milliseconds)
180
160
140
120
100
80
60
40
20
0
8
16
32
64
128
256
512
1024
2048
NUMBER OF GPUS
Strong-scaling curve
More GPUs = less time
(gives 1/N shaped curve)

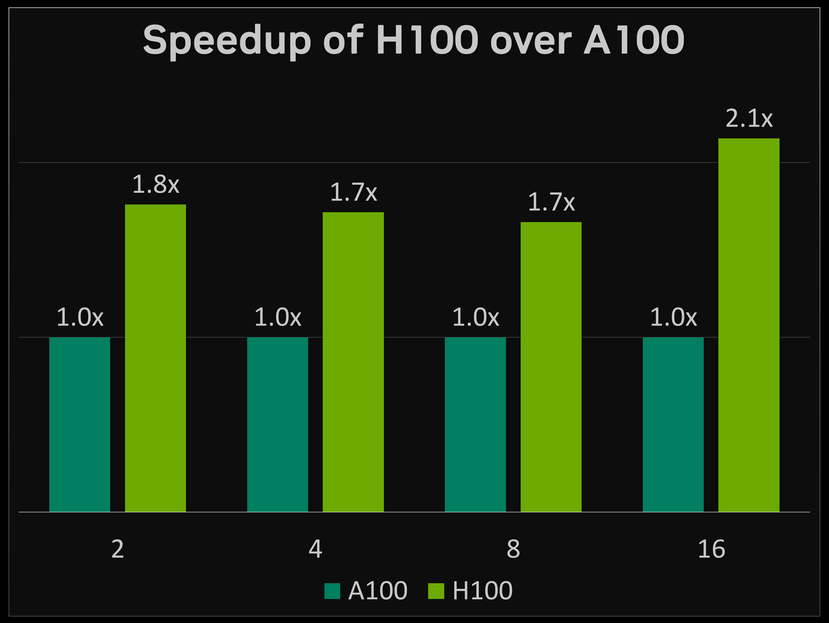
cuFFTMp + JAX



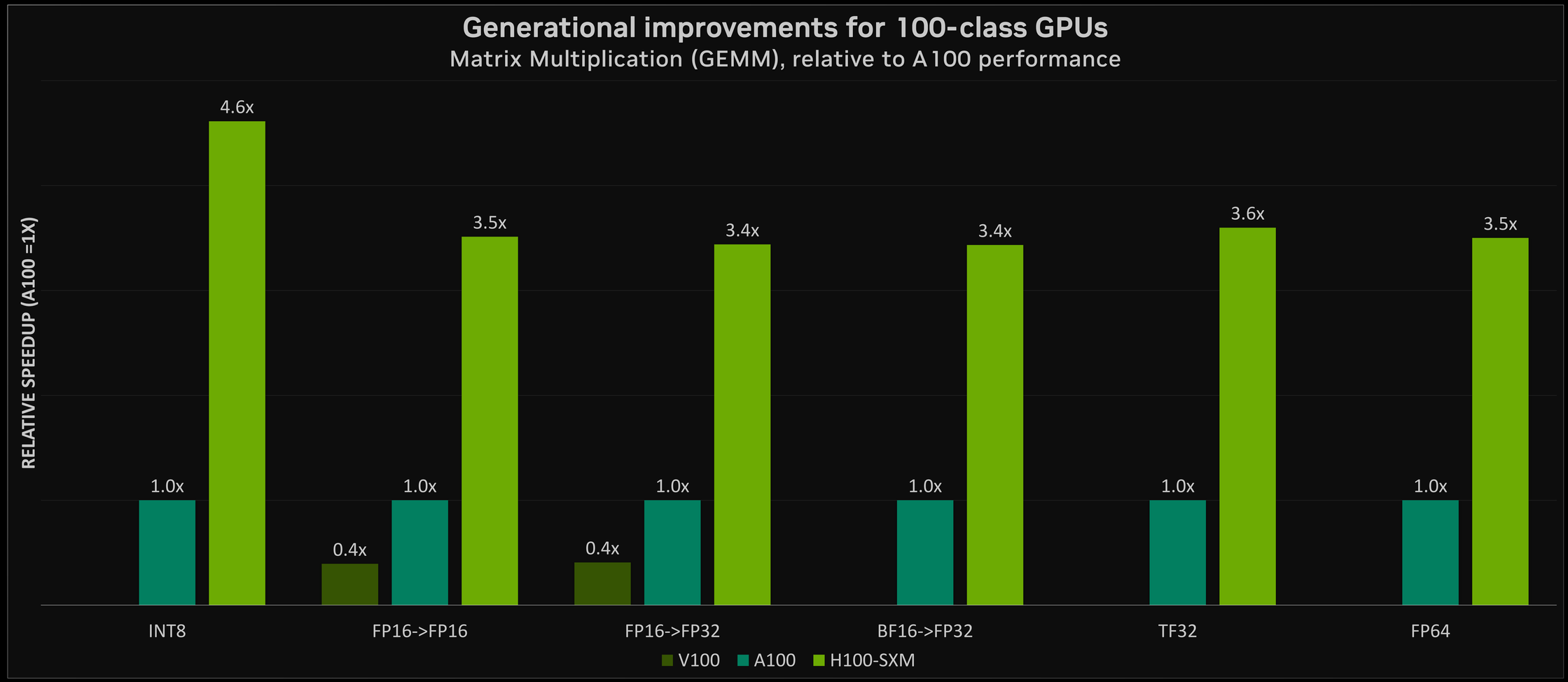


cuFFTMp Performance on H100 GPU vs. A100

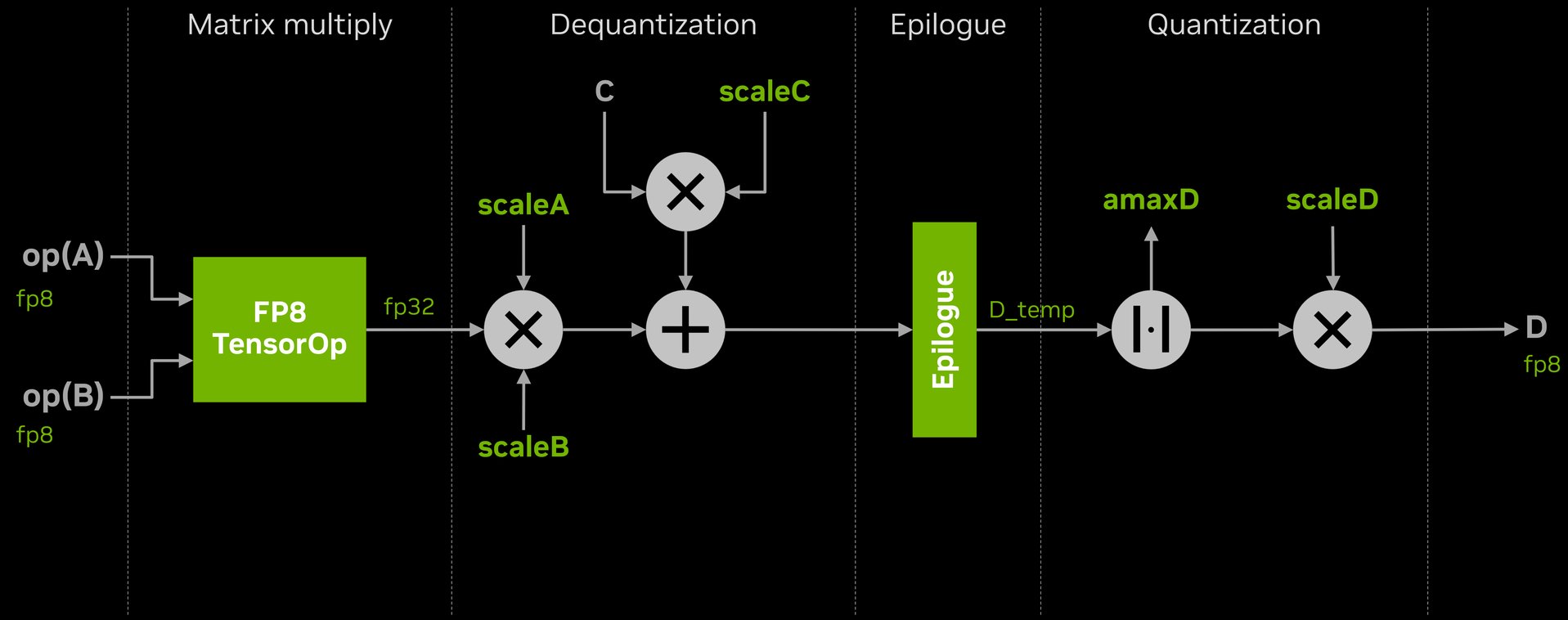




Tensor Core Performance Across Generations

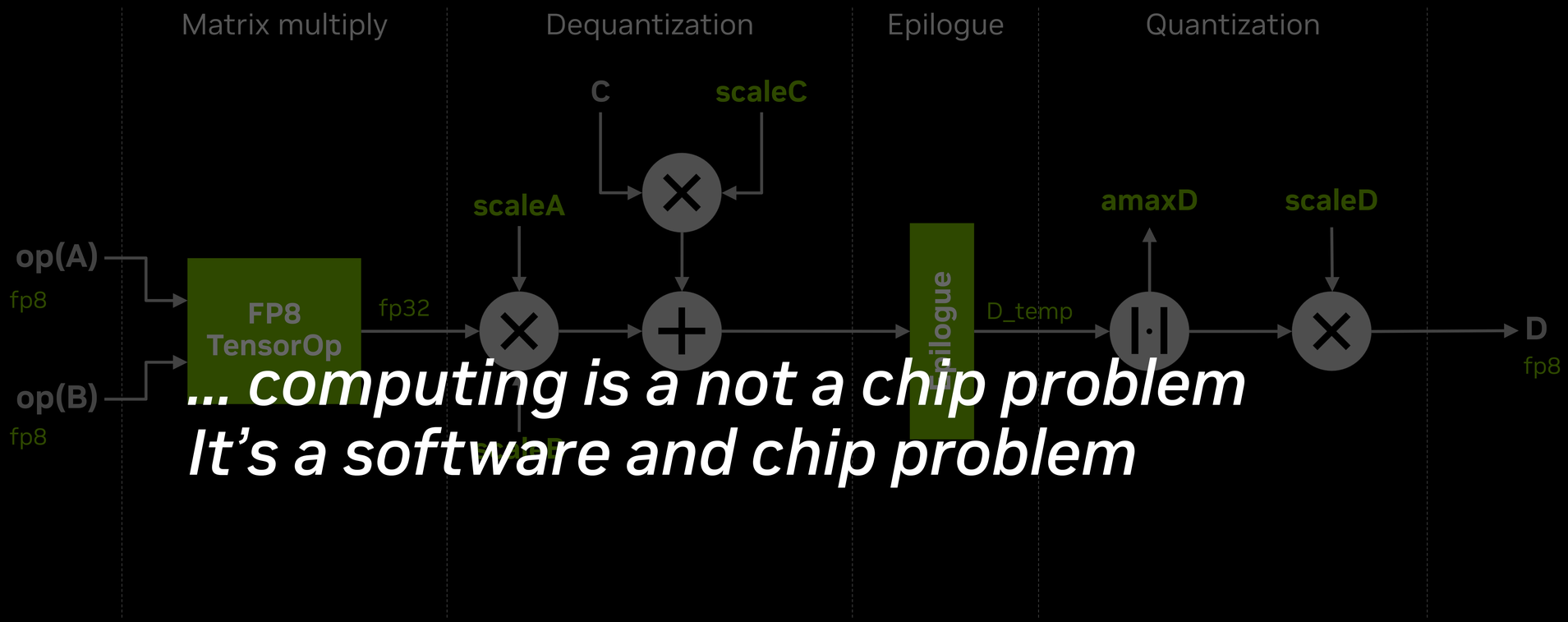


FP8 Matrix Multiplication in cuBLASLt



Block diagram of FP8 matrix multiplication & scaling
(can also directly output fp16, bf16 & fp32)

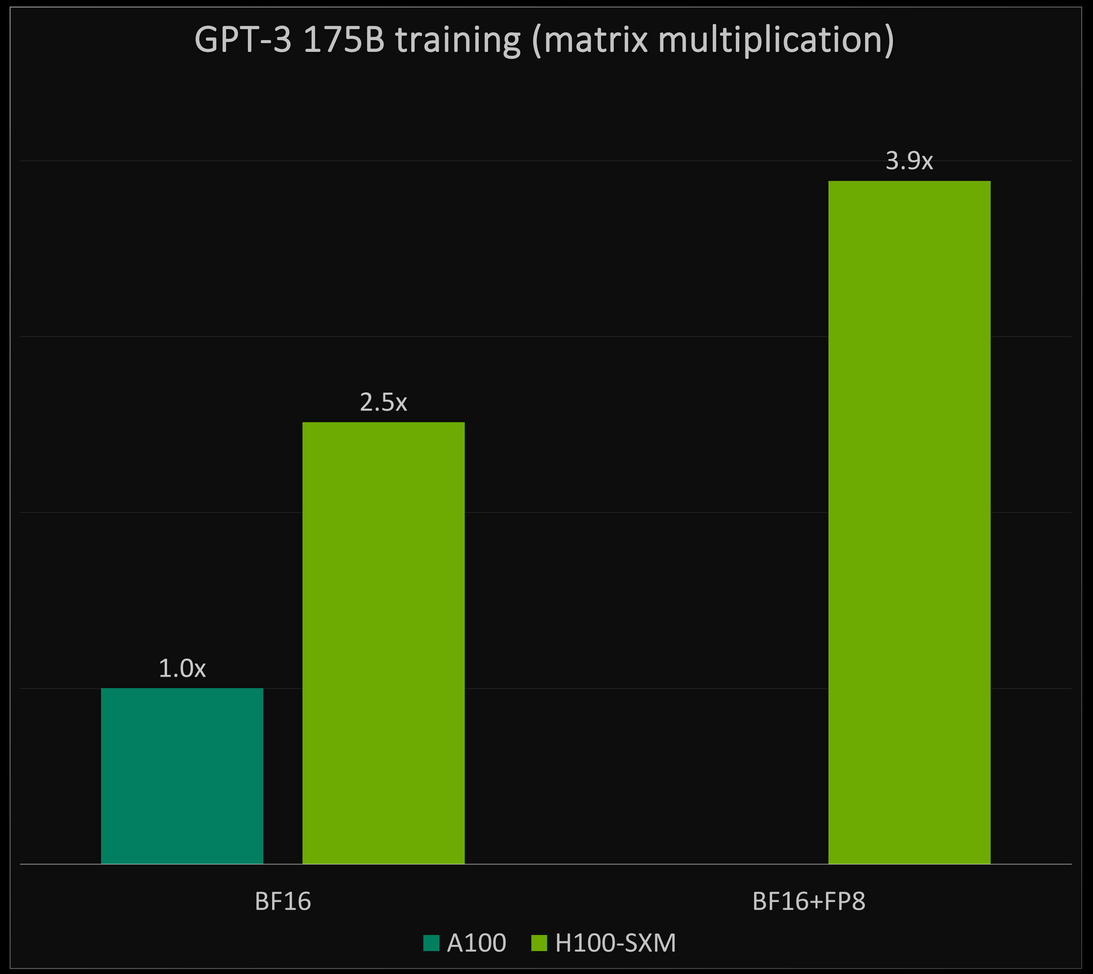
FP8 Matrix Multiplication in cuBLASLt

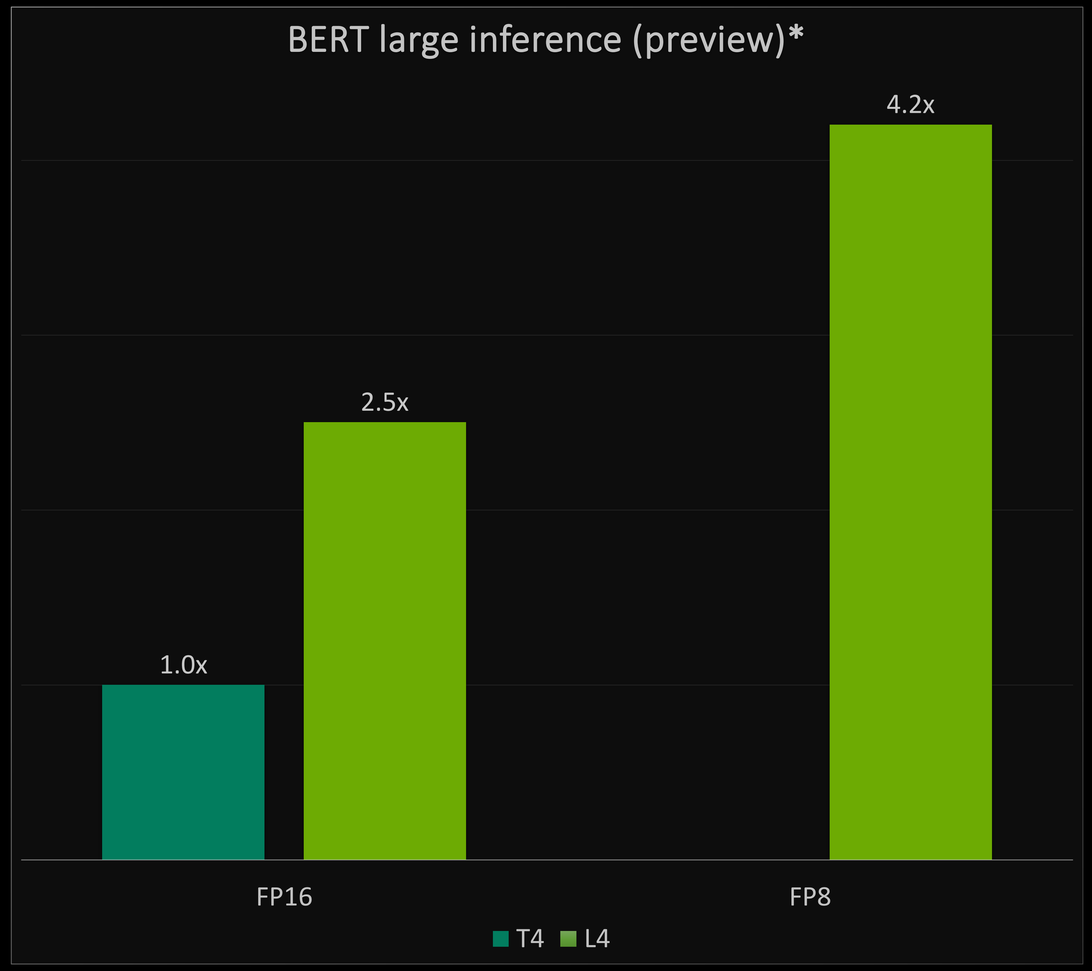


… computing is a not a chip problem
It's a software and chip problem

Block diagram of FP8 matrix multiplication & scaling
(can also directly output fp16, bf16 & fp32)

cuBLAS FP8 on NVIDIA H100 and L4 GPUs

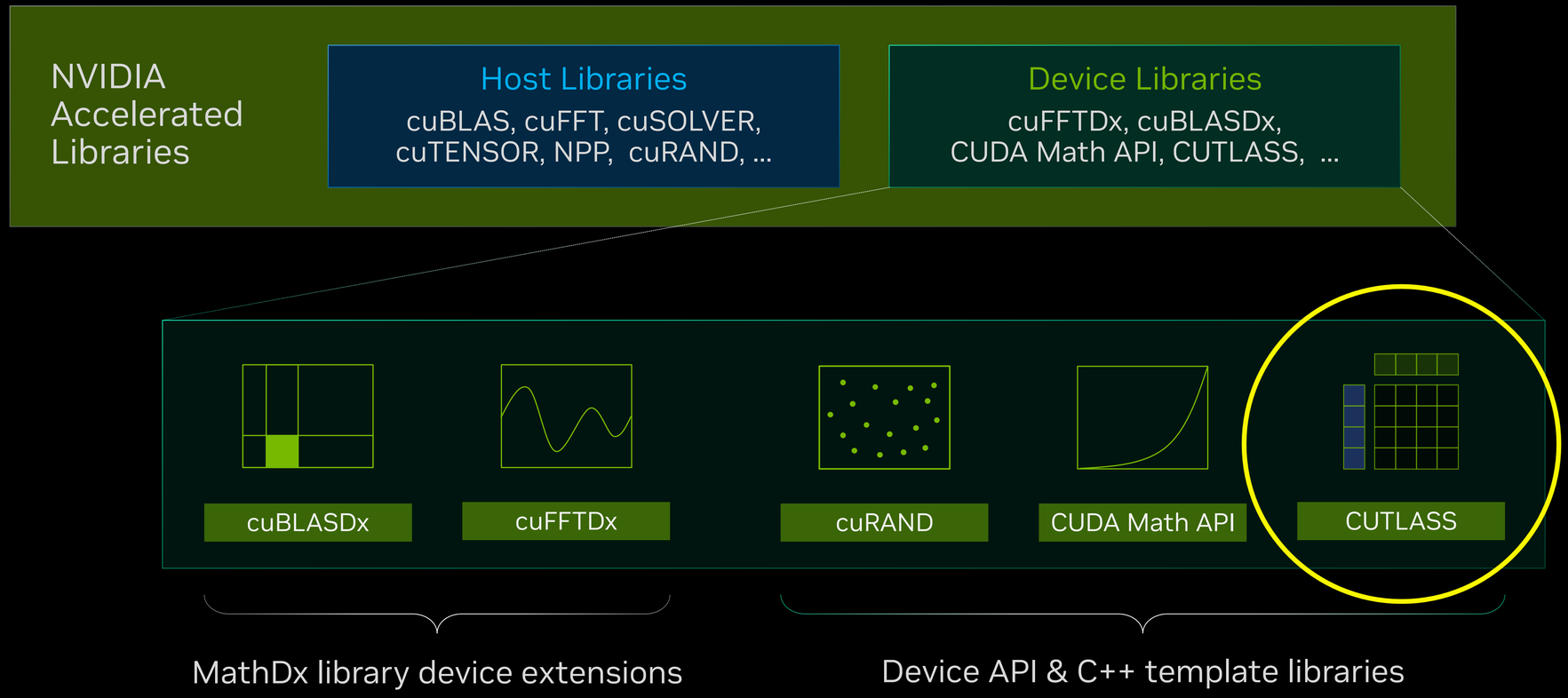




* Average over a range of batch sizes and average sequence lengths

Recent Developments in NVIDIA Math Libraries [S51176]

Device Libraries



Download MathDx Today

CUTLASS 3.0

Tensor Core programming model for linear algebra kernels in CUDA

CUTLASS 3.0

New major version of CUTLASS re-written with **CuTe**

Tiled MMA and Copy micro-kernels represent all GPU arch inner loops

Freely compose micro-kernels with any outer loop schedule

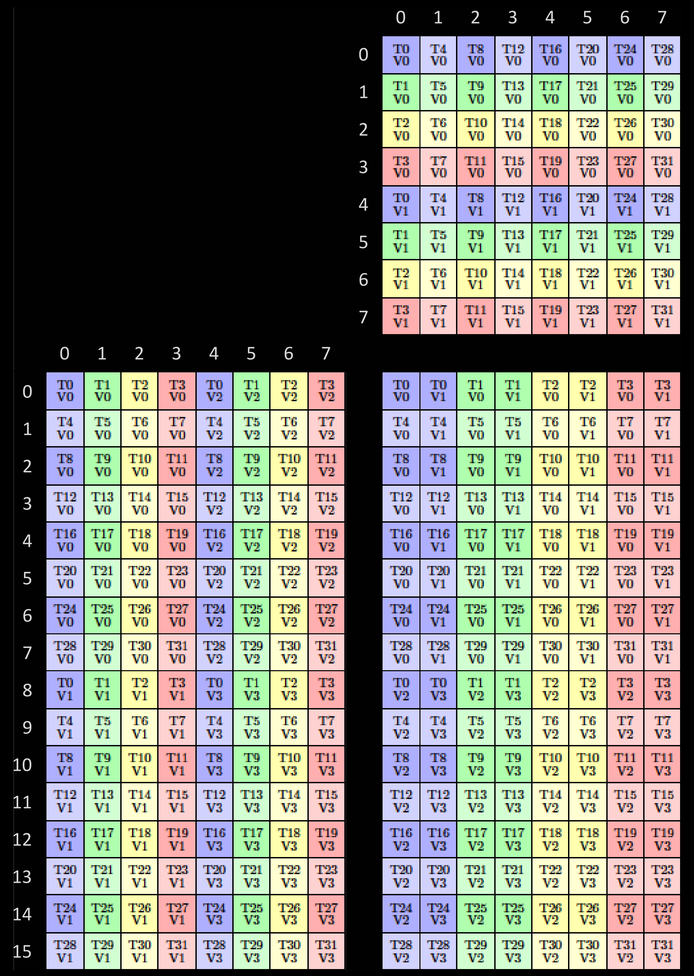
Ideal abstraction for custom kernel development targeting Hopper

Hopper support

Optimal computations for 4th generation Tensor Cores in H100

GEMM kernels targeting Hopper WGMMA + TMA + Threadblock Clusters

Warp-specialized persistent grid scheduling for peak utilization



Layout of threads & data for a single Hopper double precision (FP64) tensor core operation

Download CUTLASS 3.0 from the GitHub repo

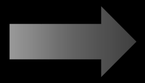
NVIDIA

CuTe: Robust Representation for Thread and Data Layouts

New programming model and API

```
RowMajor
ColumnMajor
RowMajorInterleaved
ColumnMajorInterleaved
PitchLinear
TensorNCHW
VoltaTensorOpMuliplicandCongruous
ColumnMajorVoltaTensorOpMuliplicandCongruous
RowMajorVoltaTensorOpMuliplicandCongruous
VoltaTensorOpMuliplicandBCongruous
ColumnMajorVoltaTensorOpMuliplicandBCongruous
RowMajorVoltaTensorOpMuliplicandBCongruous
VoltaTensorOpMuliplicandCrosswise
ColumnMajorVoltaTensorOpMuliplicandCrosswise
RowMajorVoltaTensorOpMuliplicandCrosswise
Many, many more ...
```

CUTLASS 2.x Layout Definitions



CuTe Layout

Hierarchical and multi-dimensional layout representation

Formal algebra for manipulating layouts

Layouts for both threads and data

Used in CUTLASS 3.0 to build a unified GPU micro-kernel abstraction

```
Layout<Shape, Stride>
```

CUTLASS 3.0 **CuTe** Layout Definition

CUTLASS 3: Towards Better Developer and User Experience

Simplified but powerful kernel instantiation

Hopper-optimized GEMM mainloop

```
using CollectiveOp =
typename gemm::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    half_t, LayoutA, 8,
    half_t, LayoutB, 8,
    float,
    Shape<_128,_128,_64>, Shape<_1, 2,_1>,
    gemm::collective::StageCountAuto,
    gemm::collective::KernelScheduleAuto
  >::CollectiveOp;
```

Hopper-optimized GEMM mainloop
with persistent scheduling and 5 stages

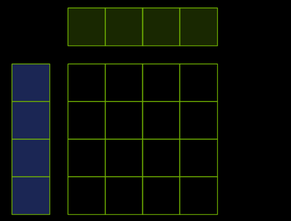
```
using CollectiveOp =
typename gemm::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    half_t, LayoutA, 8,
    half_t, LayoutB, 8,
    float,
    Shape<_128,_128, 64>, Shape< 1, 2,_1>,
    gemm::collective::StageCount<5>,
    gemm::KernelTmaWarpSpecializedPersistent
  >::CollectiveOp;
```

CUTLASS 3: Towards Better Developer and User Experience

Python interface coming in CUTLASS 3.1 (experimental)

Simplified kernel declaration

```
plan = cutlass.GroupedGemm(
    element=torch.float16,
    layout=cutlass.LayoutType.RowMajor)
```



CUTLASS

Simplified porting to DL frameworks

```
cutlass.emit.pytorch(
    plan.construct(), name='grouped_gemm', cc=80)
```

PY | C++ | CU

```
> python setup.py install
```

```
# Import and run generated PyTorch extension
import grouped_gemm
Ds = grouped_gemm.run([A0, A1],
                      [B0, B1])
```

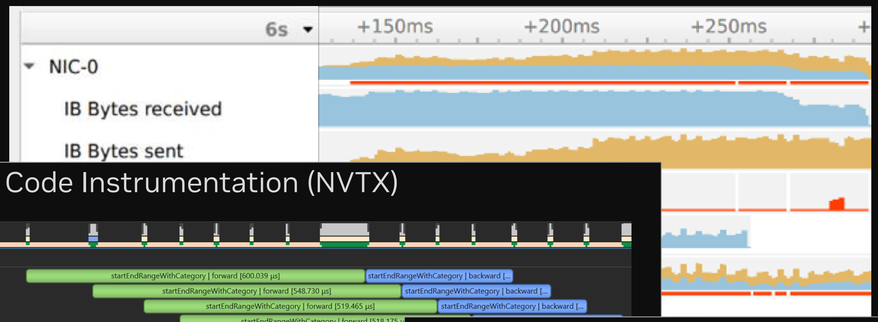
Developing Optimal CUDA Kernels on Hopper Tensor Cores [S51413]

NVIDIA GPU Computing Tools

Correctness Tools

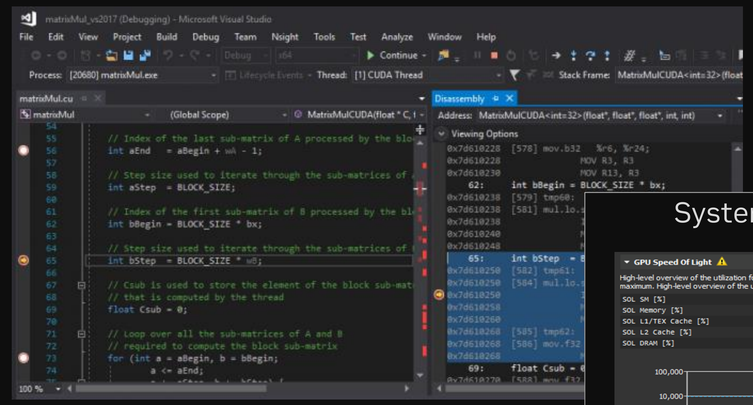
```
$ compute-sanitizer --leak-check full memcheck_demo
========= COMPUTE-SANITIZER
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
Sync: no error
Running out_of_bounds_kernel
Ran out_of_bounds_kernel: no error
Sync: no error
```

Networking & Communications Sampling

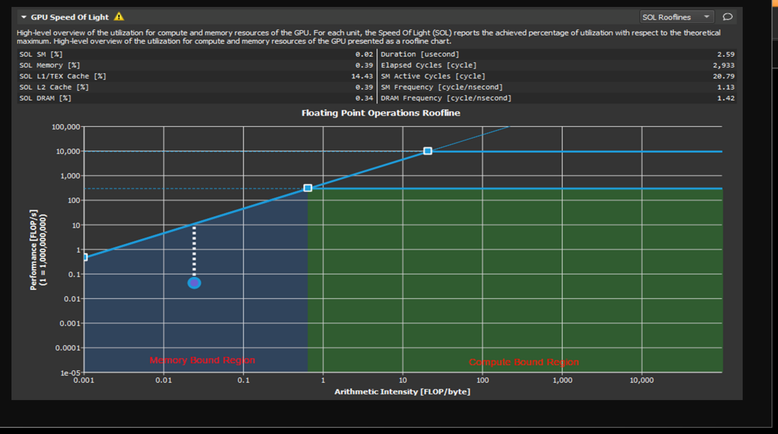


Code Instrumentation (NVTX)

Debuggers, Profilers, IDE Integration



System Profiling, Analysis & Recommendation



Online Monitoring (DCGM)

From the Macro to the Micro: CUDA Developer Tools Find and Fix Problems at Any Scale [S51205]

NVIDIA

Nsight Systems Multi-Node Analysis

Literally tools for your tools

Recipes to analyze existing reports

Collect reports from multiple sources/nodes

Reduces “getting started” time

Analyze thousands of reports

Scale-out processing

Cluster-wide conclusions

Recipes generate results various formats

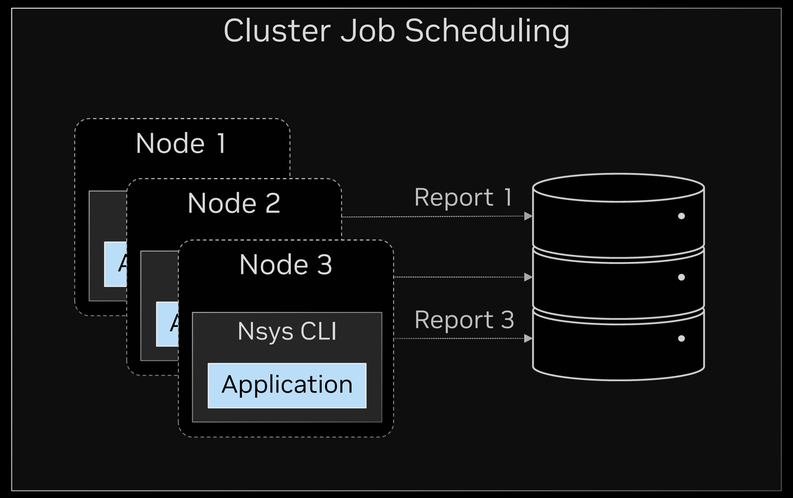
Jupyter Notebook

Parquet, CSV, …

Multi-Report Analysis Workflow

Preview in Nsight Systems 2023.2

Select or create custom recipes
Recipes built on extensible Python library
Set recipe parameters & arguments

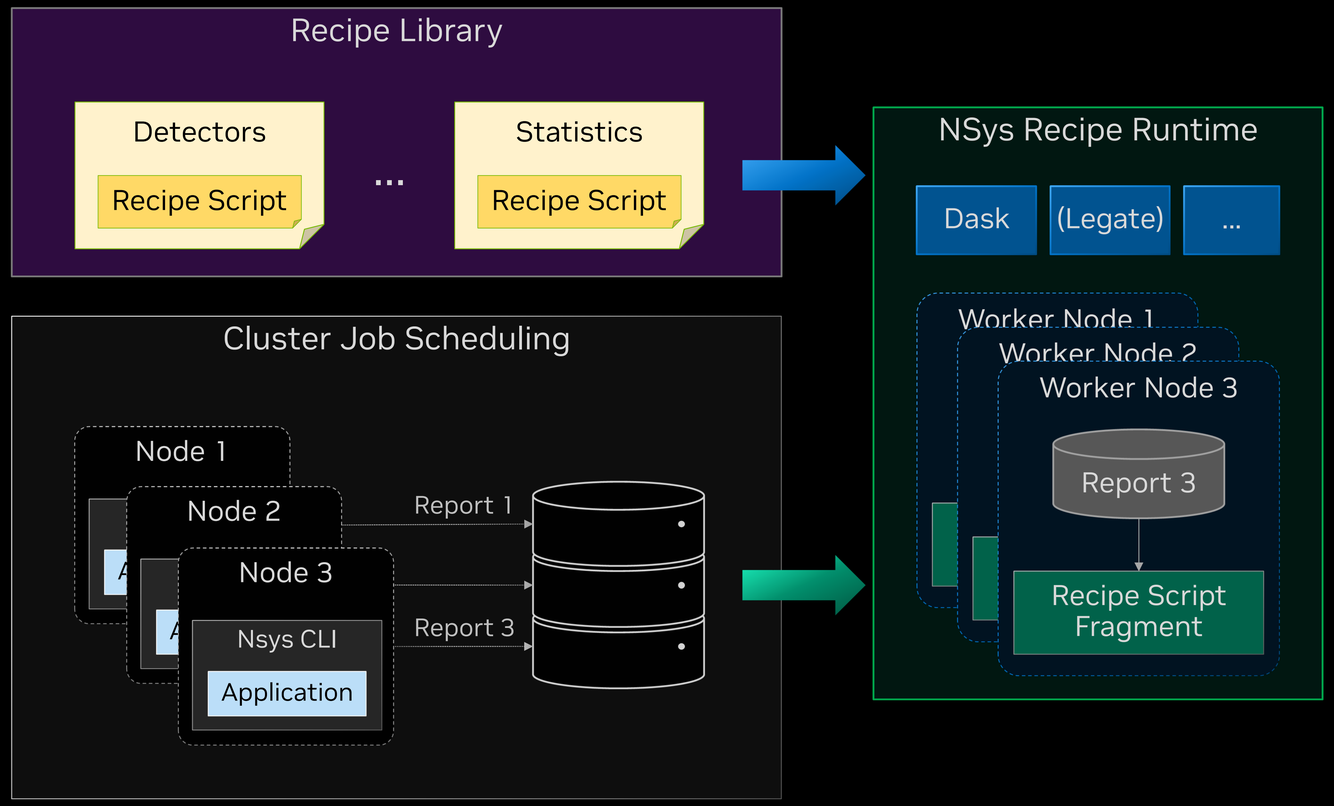


Generate reports per-node / per-rank as usual
Data is output to storage during run
Can schedule for analysis immediately or later

Setup & Report Generation

Multi-Report Analysis Workflow

Preview in Nsight Systems 2023.2



Setup & Report Generation

Launch Analysis

Launch recipe runtime

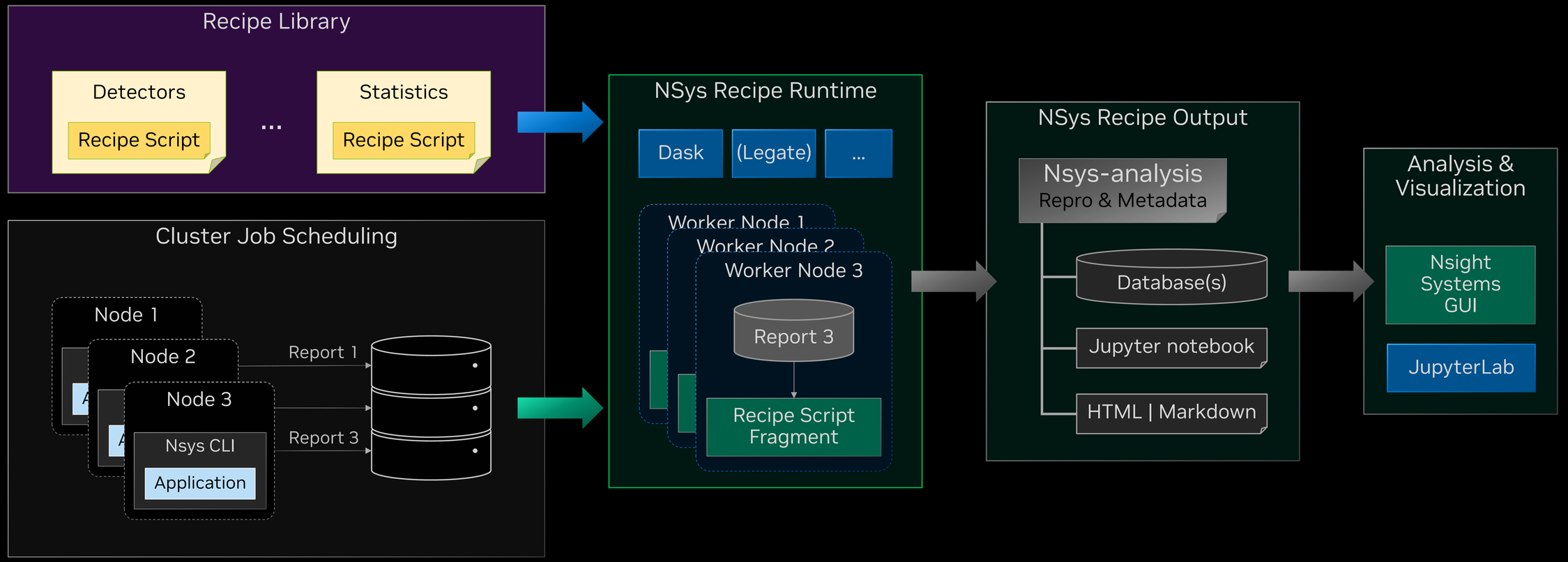
```
$ ./nsys recipe <recipe_name> <args>

$ ./python3 -m nsys_recipe <recipe_name> <args>
```

Several default recipes included with Nsight Systems

- CUDA API Summary
- CUDA Synchronization APIs
- CUDA GPU Kernel Summary
- CUDA GPU MemOps Summary (by Size)
- CUDA GPU MemOps Summary (by Time)
- CUDA Async Memcpy with Pageable Memory
- CUDA Synchronous Memcpy
- CUDA Synchronous Memset
- DX12 Memory Operations
- GPU Gaps
- GPU Time Utilization
- NVTX GPU Trace
- NVTX Range Summary
- OS Runtime SummaryMPI summary
- NCCL summary (options for CPU & GPU)
- NVTX GPU Projection Summary (via CUDA info)
- Iteration pacing via CUDA kernels
- Iteration pacing via NVTX (CPU or GPU)

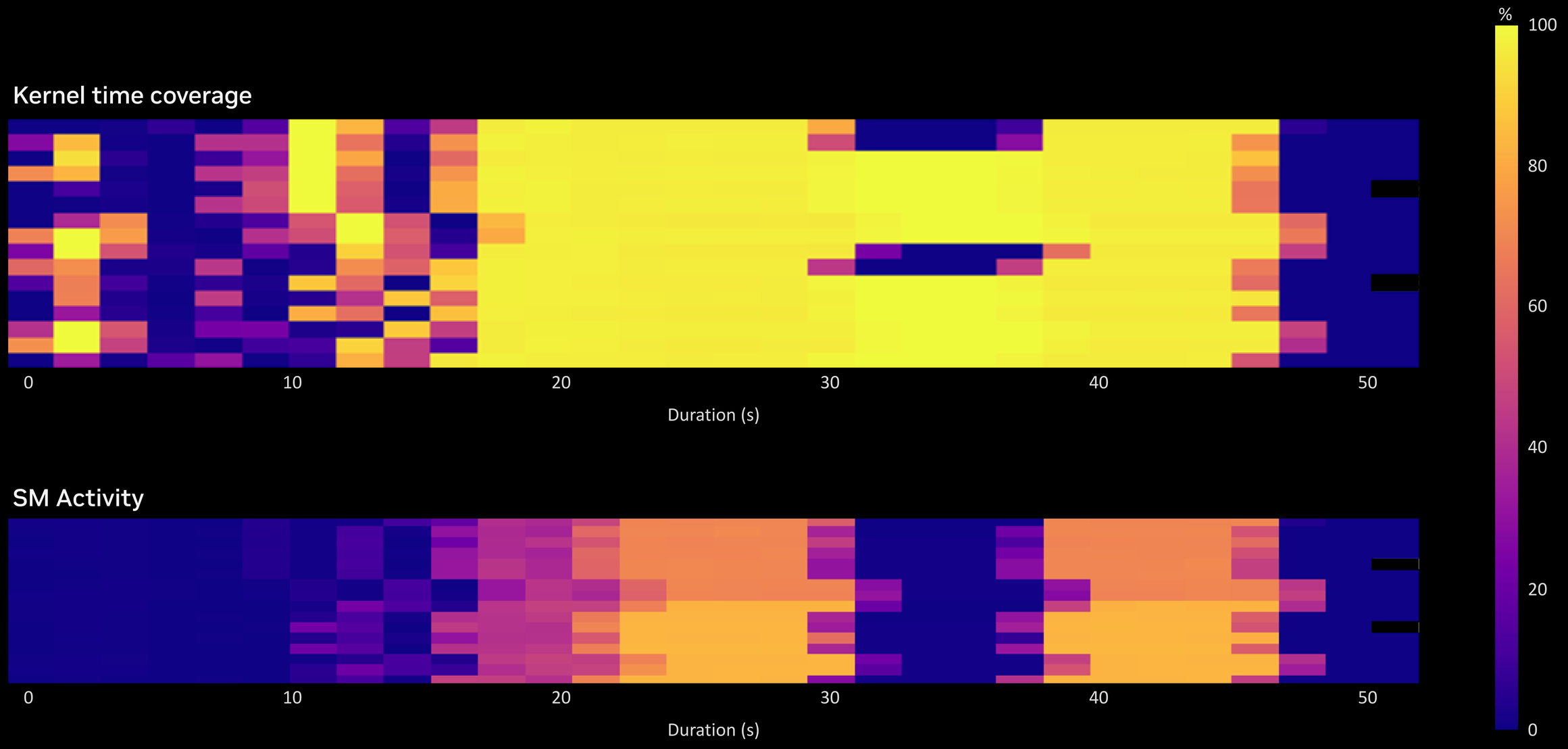
Multi-Report Analysis Workflow

Preview in Nsight Systems 2023.2



Setup & Report Generation | Launch Analysis | Output | View

Example: Heatmap Analysis



Optimizing at Scale: Investigating and Resolving Hidden Bottlenecks for Multi-Node Workloads [S51421]

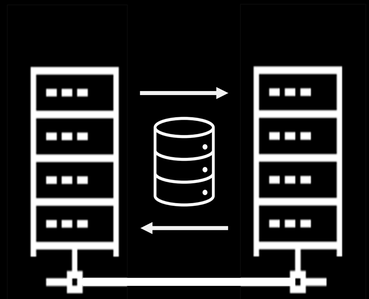
Datacenter Scale Computing

Datacenter Data Security

Data at Rest
(In Storage)

STRONG
PROTECTION

Data in Transit
(Across a Network)



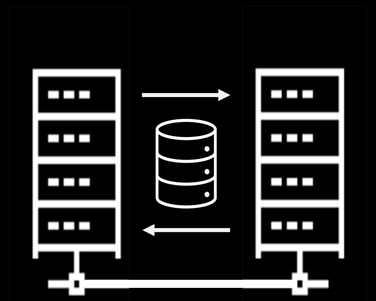
STRONG
PROTECTION

Datacenter Data Security

Data at Rest
(In Storage)

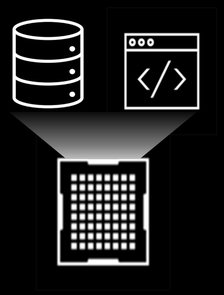
STRONG
PROTECTION

Data in Transit
(Across a Network)

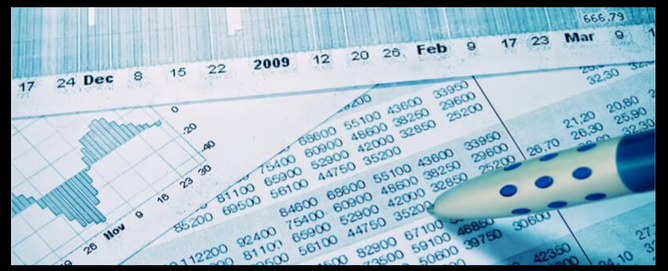


STRONG
PROTECTION

Data in Use
(Being Processed)



WEAK
PROTECTION

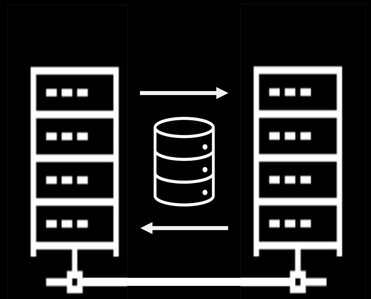


What is Confidental Computing?

Data at Rest
(In Storage)

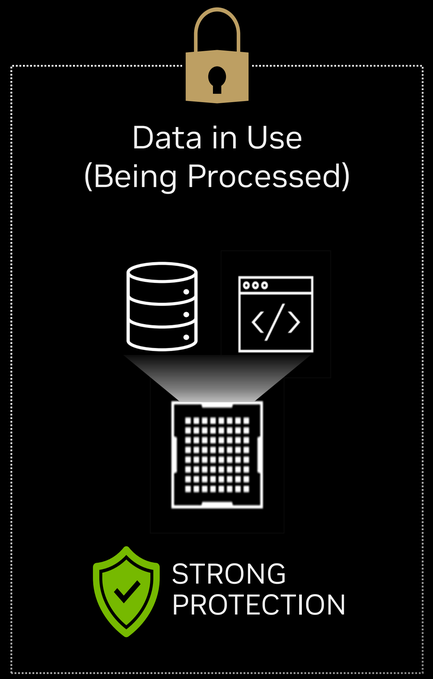
STRONG
PROTECTION

Data in Transit
(Across a Network)



STRONG
PROTECTION

Data in Use
(Being Processed)



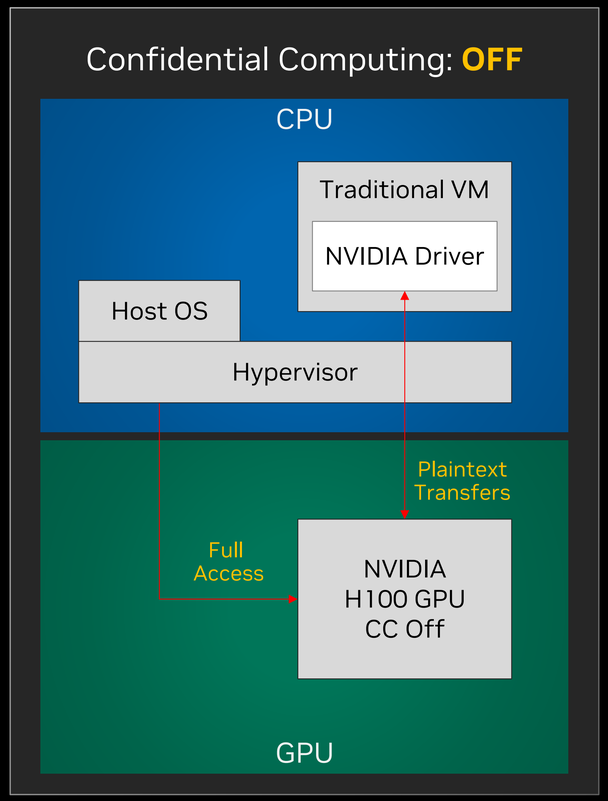
STRONG
PROTECTION

Confidential Computing

Protect data at rest, in transit, **and in use**

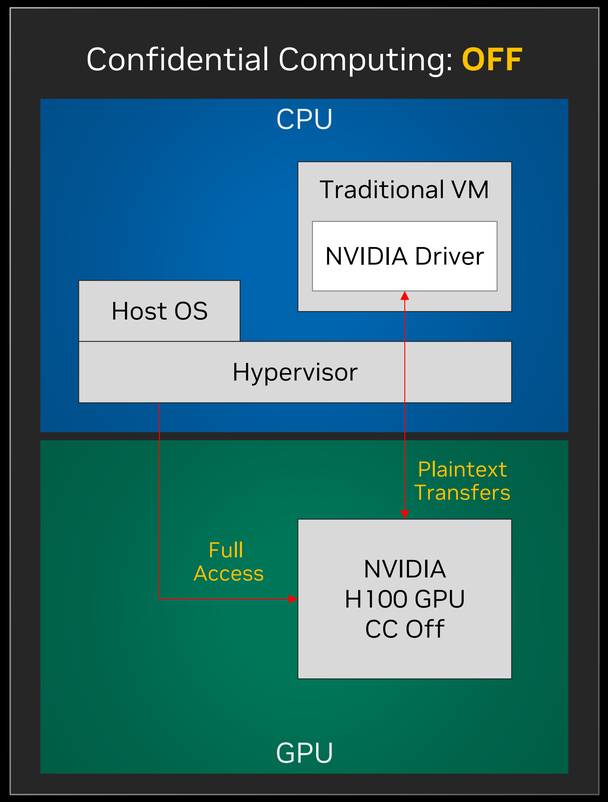
Protect data and code from **other users**

Protect data and code from **machine owners**

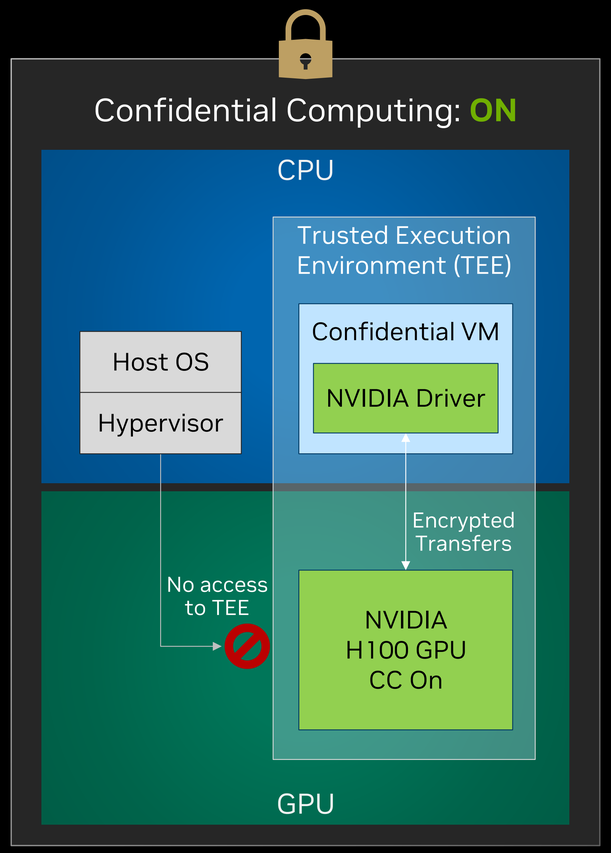


Hypervisor has full access to all of system memory and all of GPU memory

GPU With VM-Based Confidential Computing



Hypervisor has full access to all of system memory and all of GPU memory

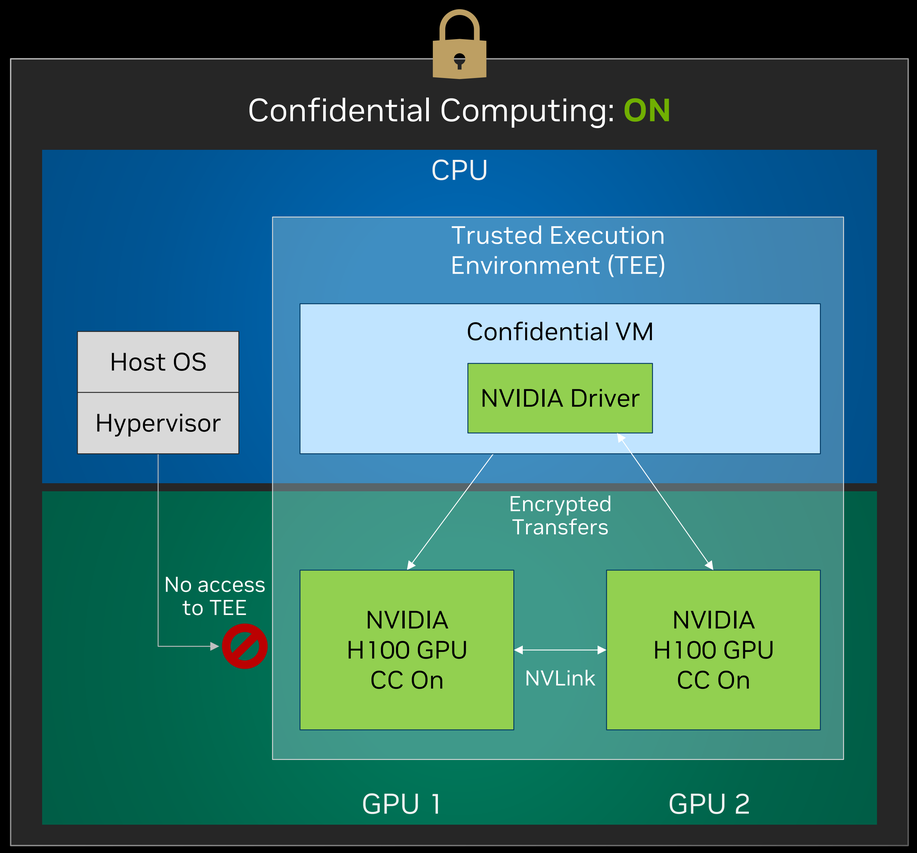


Hypervisor is blocked from accessing the Confidential VM in system memory and blocked from reading GPU memory

GPU With VM-Based Confidential Computing

Applications run unchanged in confidential mode
NVIDIA drivers, libraries, APIs run in the VM
VM-based TEE protects the entire workload

Each GPU is passed through to just one VM
VMs protected from each other
One VM may have multiple GPUs

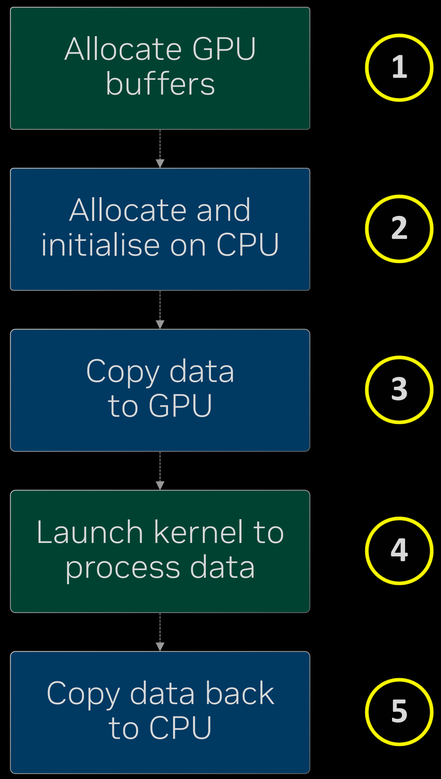


Two NVLink-connected GPUs within the same TEE

Hopper Confidential Computing: How it Works under the Hood [S51709]

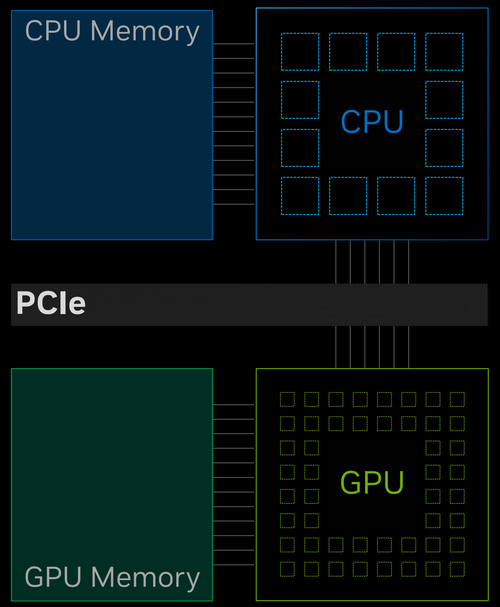
CUDA Applications Run Unmodified Under Confidential Computing

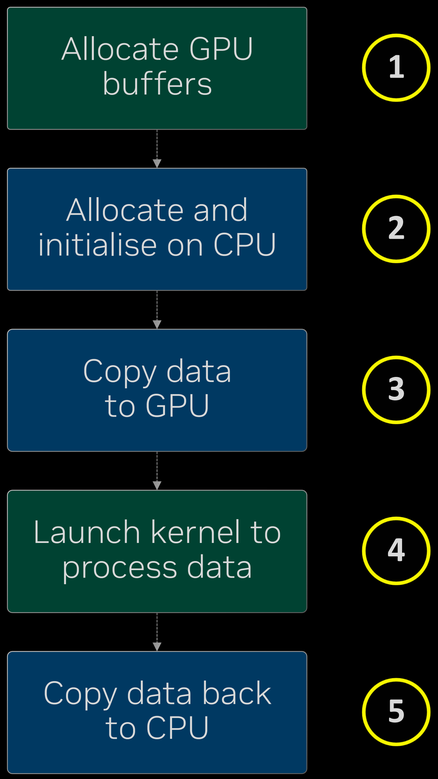
```
int *gpu_compute(size_t size) {
    int *d_data, *d_result, *h_data;

    // Allocate GPU buffers
1   cudaMalloc(&d_data, size);
    cudaMalloc(&d_result, size);

    // Allocate and initialise on CPU
2   cudaMallocHost(h_data, size);
    init(h_data, size);

    // Copy to GPU and process it, then copy back
3   cudaMemcpy(d_data, h_data, size);
4   kernel<<< ... >>>(d_data, d_result);
5   cudaMemcpy(h_data, d_data, size);

    // Return the result
    return h_data;
}
```



CUDA Applications Run Unmodified Under Confidential Computing

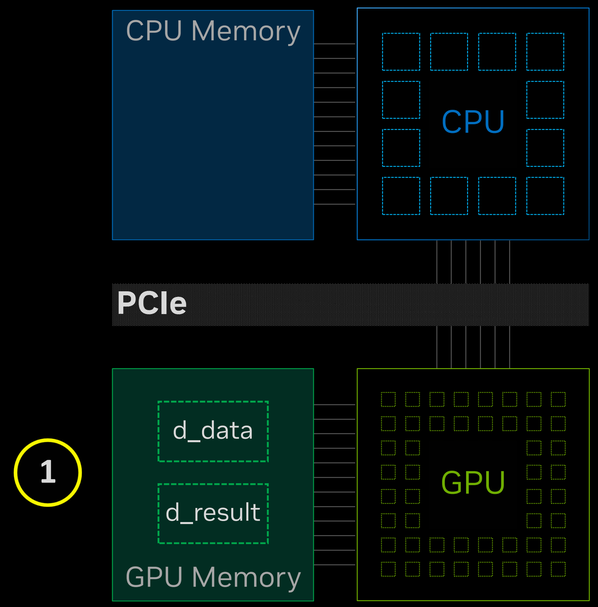
Step	CC Off	CC On



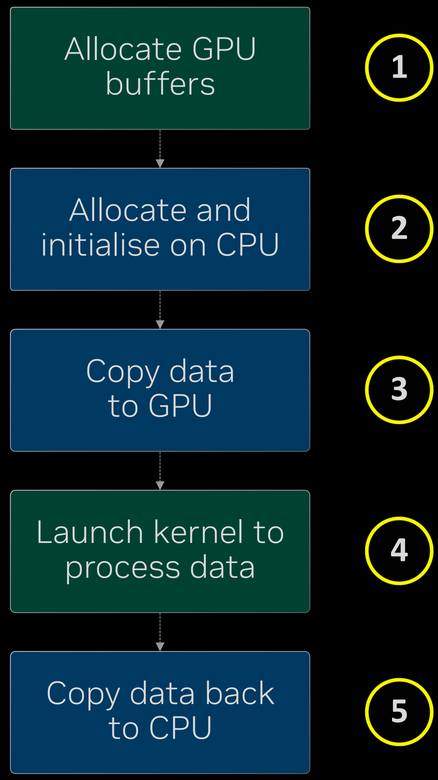


CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	

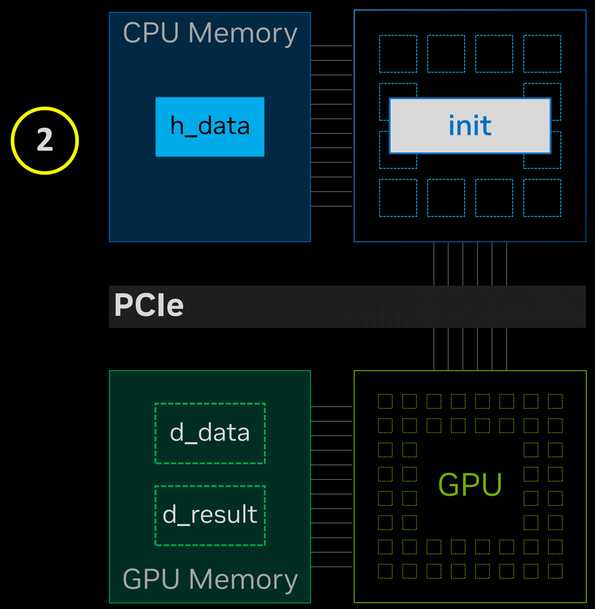



```
cudaMalloc(&d_data, size);
cudaMalloc(&d_result, size);
```

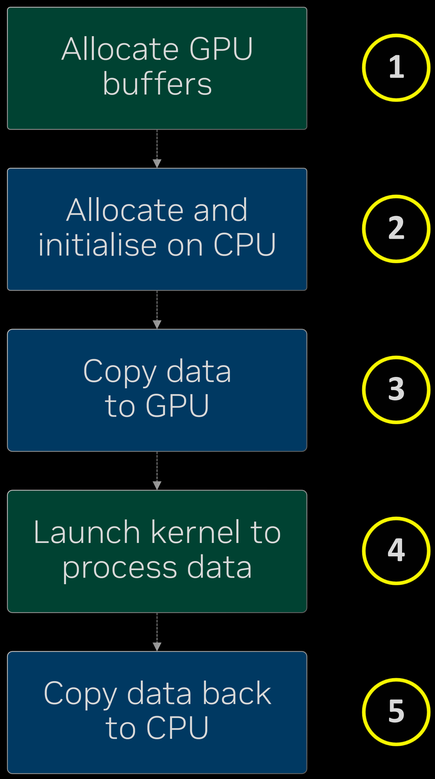


CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	
2	Pinned sysmem allocation & init	

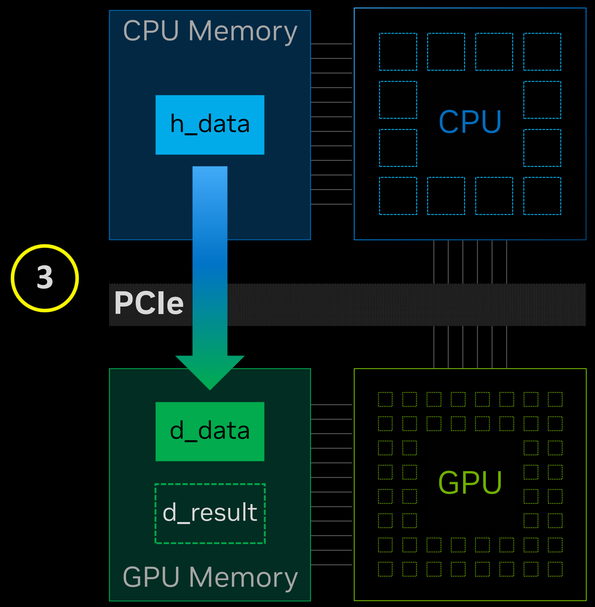



```
cudaMallocHost(h_data, size);
init(h_data, size);
```

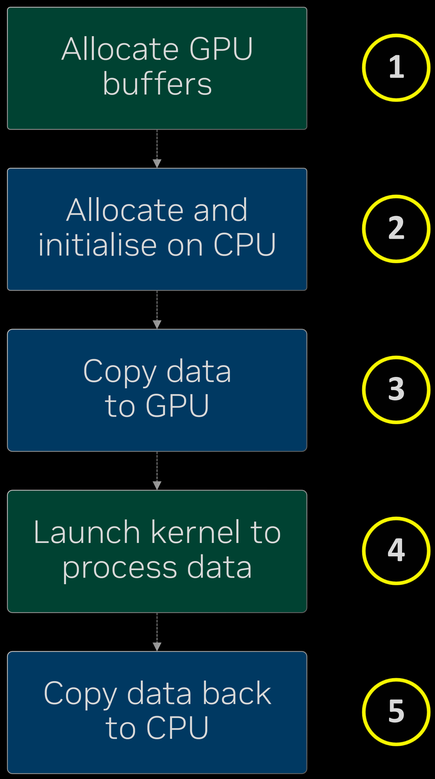


CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	
2	Pinned sysmem allocation & init	
3	DMA transfer	

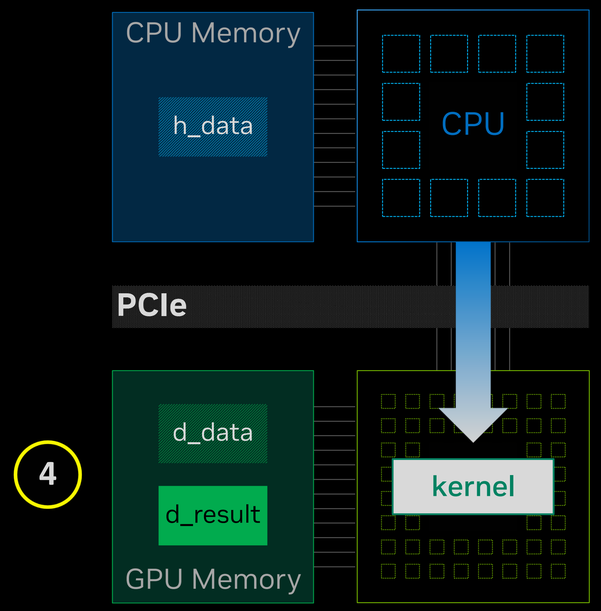


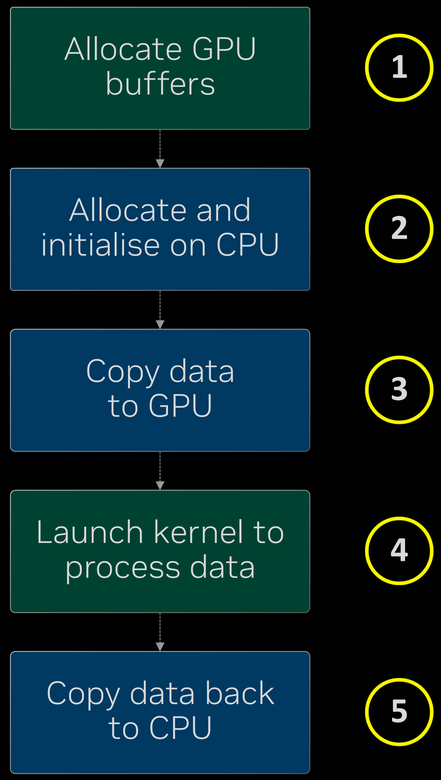

```
cudaMemcpy(d_data, h_data, size);
```



CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	
2	Pinned sysmem allocation & init	
3	DMA transfer	
4	Kernel launch	

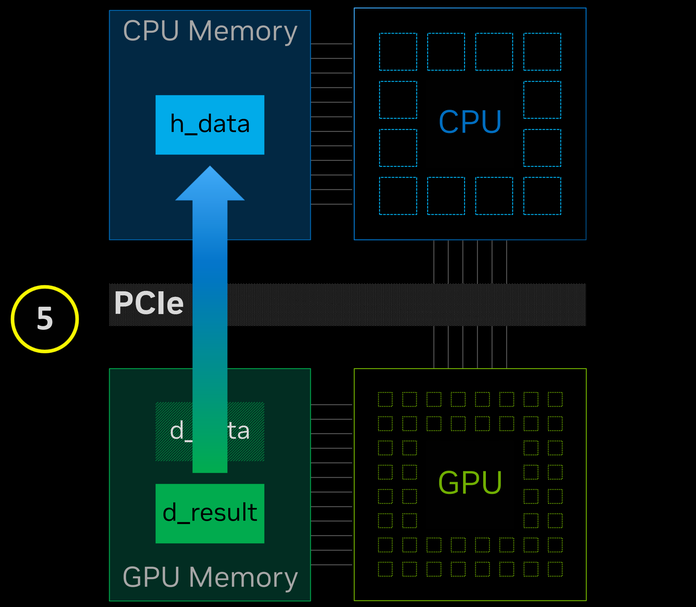




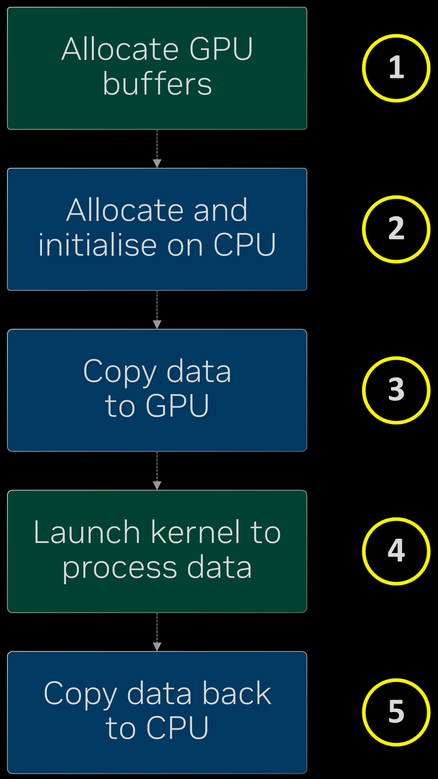

```
kernel<<< ... >>>(d_data, d_result);
```

CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	
2	Pinned sysmem allocation & init	
3	DMA transfer	
4	Kernel launch	
5	DMA transfer	

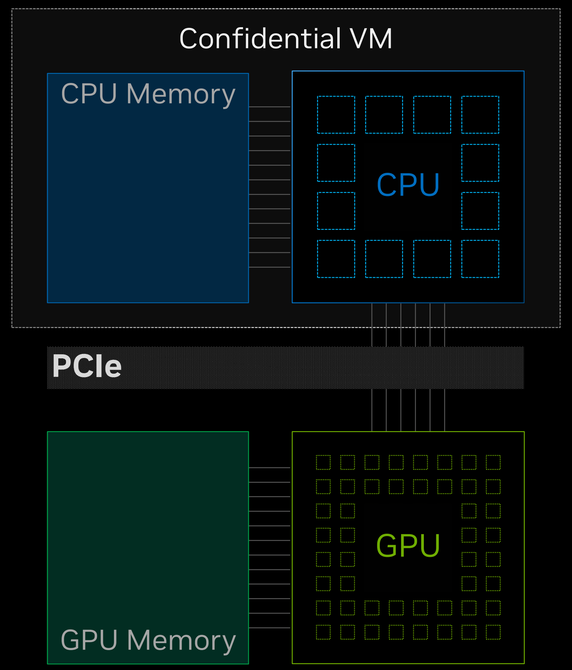


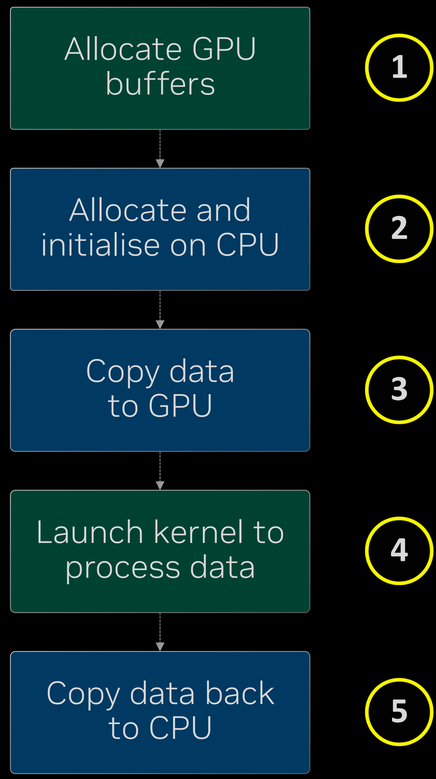

```
cudaMemcpy(h_data, d_data, size);
```



CUDA Applications Run Unmodified Under Confidential Computing

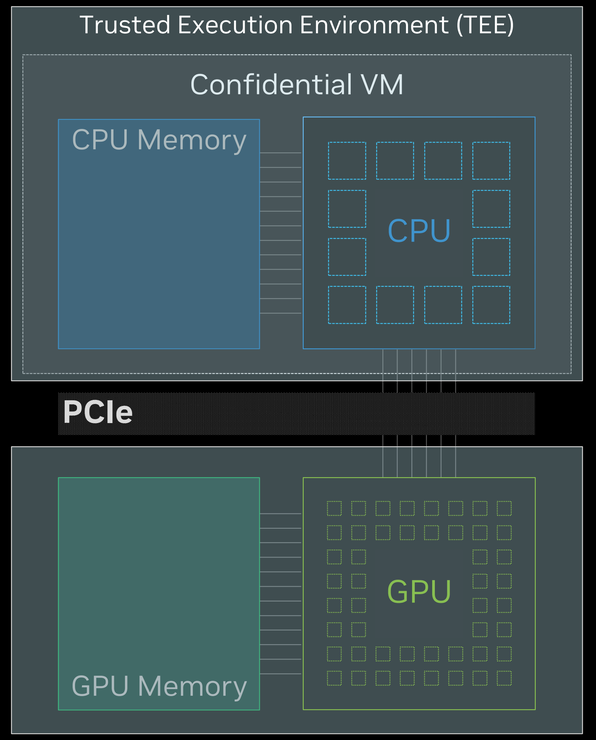
Step	CC Off	CC On
	Normal GPU allocation	
	Pinned sysmem allocation & init	
	DMA transfer	
	Kernel launch	
	DMA transfer	

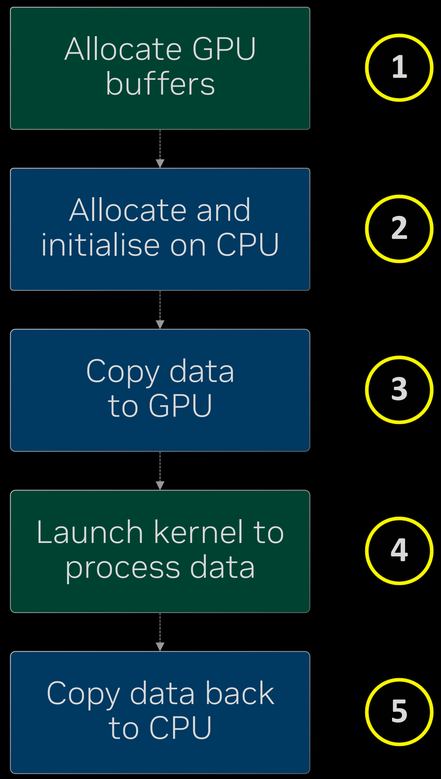




CUDA Applications Run Unmodified Under Confidential Computing

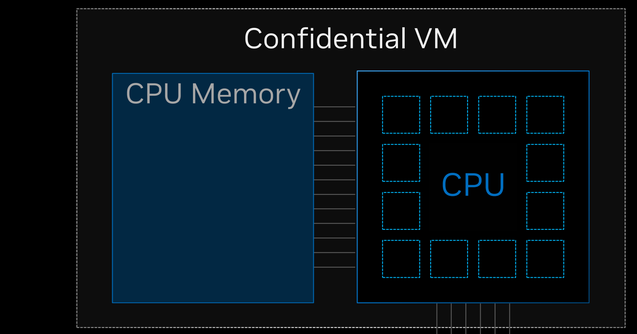
Step	CC Off	CC On
	Normal GPU allocation	
	Pinned sysmem allocation & init	
	DMA transfer	
	Kernel launch	
	DMA transfer	



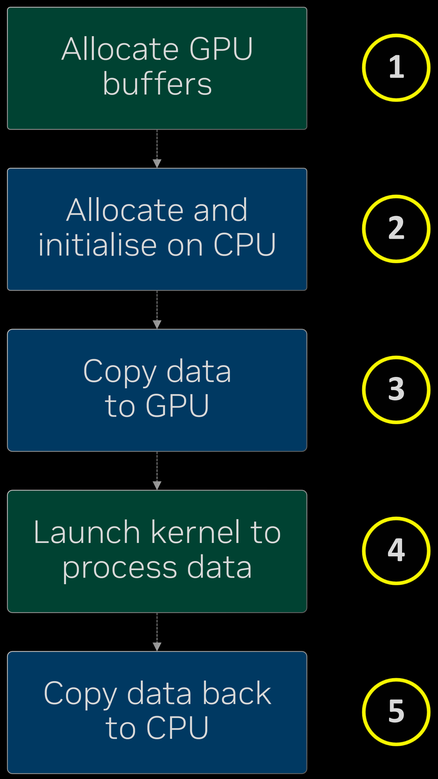


CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	Normal GPU allocation
	Pinned sysmem allocation & init	
	DMA transfer	
	Kernel launch	
	DMA transfer	

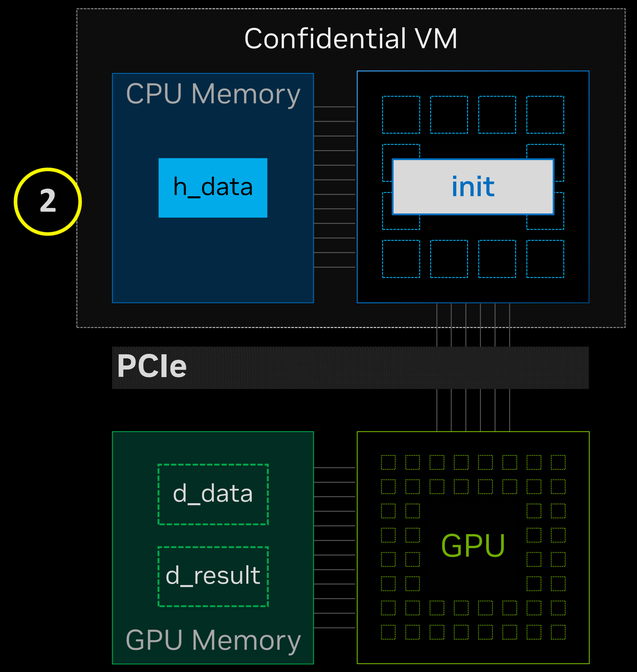


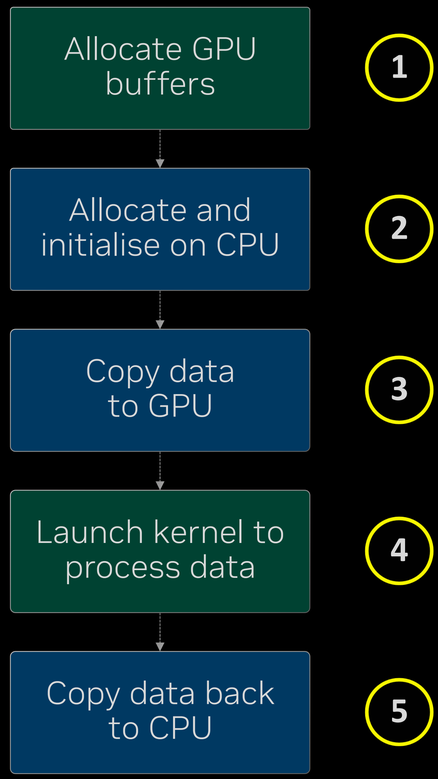

```
cudaMalloc(&d_data, size);
cudaMalloc(&d_result, size);
```



CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	Normal GPU allocation
2	Pinned sysmem allocation & init	**Managed** allocation & init
	DMA transfer	
	Kernel launch	
	DMA transfer	

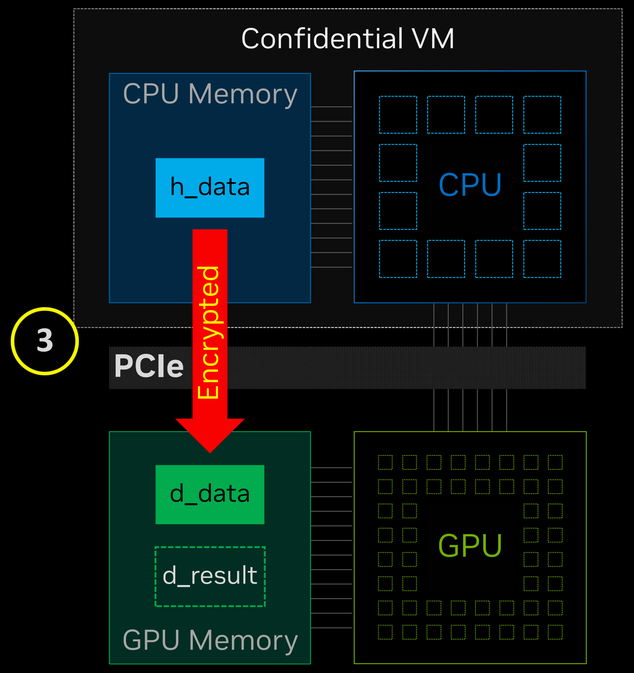




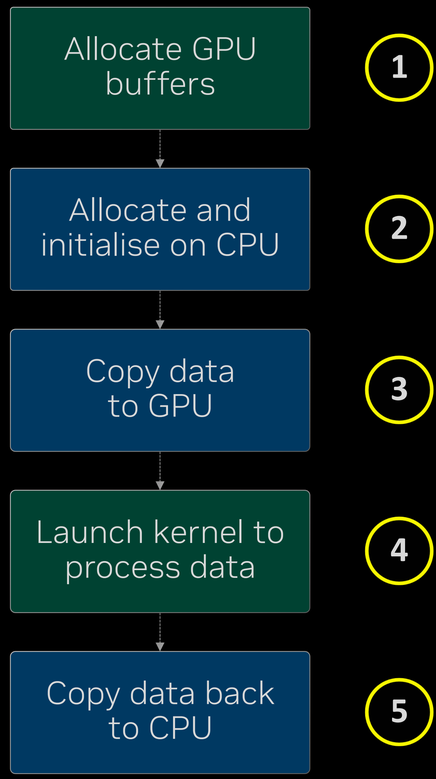

```
cudaMallocHost(h_data, size);
init(h_data, size);
```

CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	Normal GPU allocation
2	Pinned sysmem allocation & init	**Managed** allocation & init
3	DMA transfer	**Encrypted** DMA transfer
	Kernel launch	
	DMA transfer	

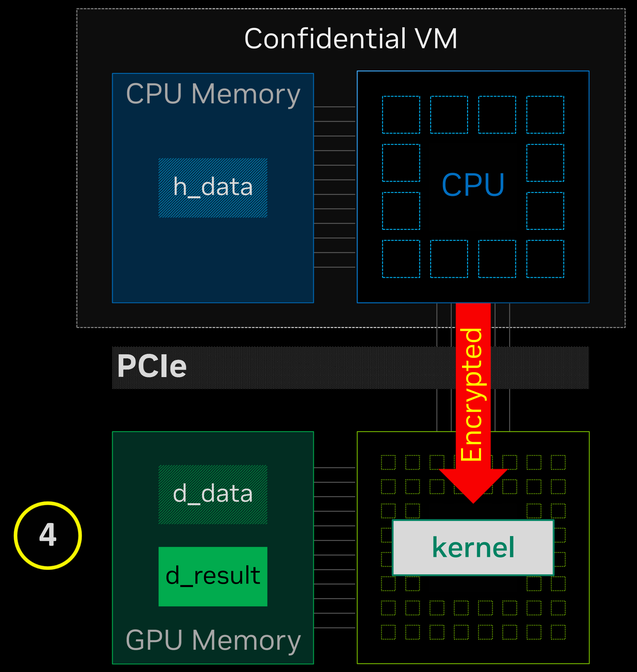


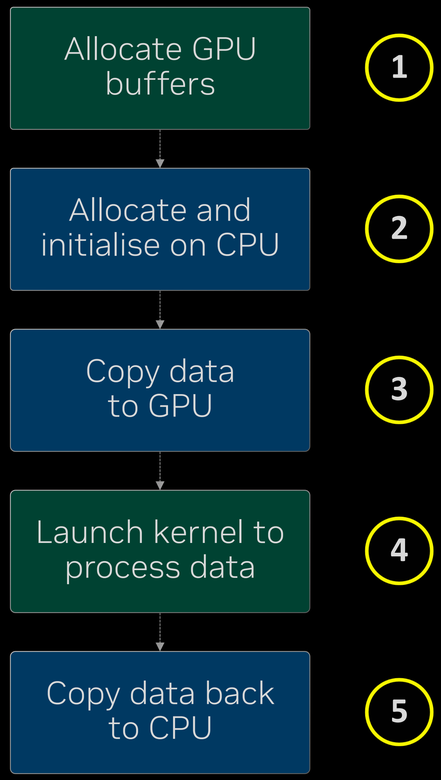

```
cudaMemcpy(d_data, h_data, size);
```



CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	Normal GPU allocation
2	Pinned sysmem allocation & init	**Managed** allocation & init
3	DMA transfer	**Encrypted** DMA transfer
4	Kernel launch	**Encrypted** kernel launch
	DMA transfer	

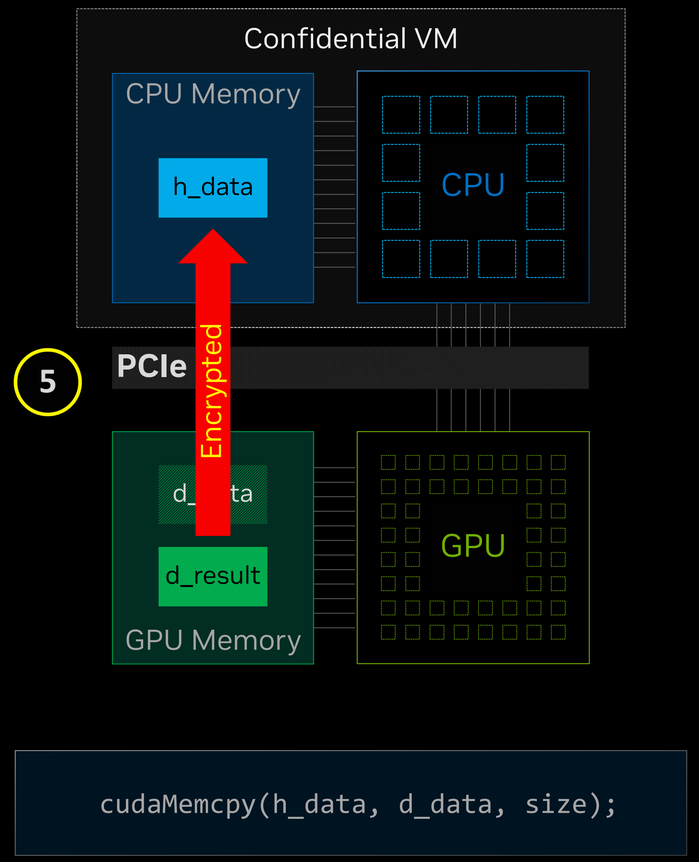


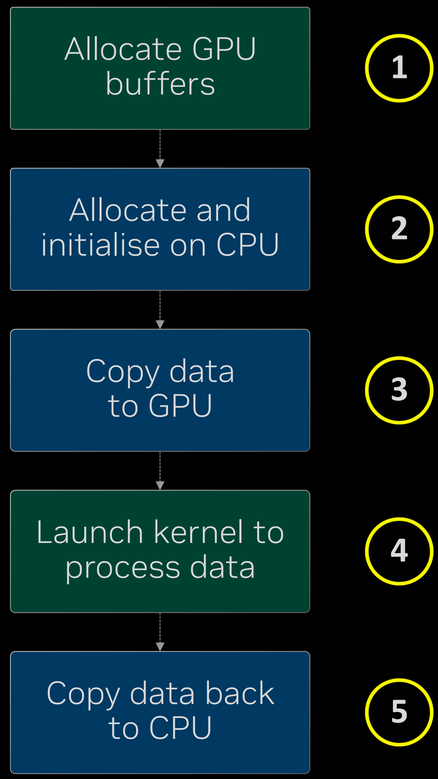



```
kernel<<< ... >>>(d_data, d_result);
```

CUDA Applications Run Unmodified Under Confidential Computing

Step	CC Off	CC On
1	Normal GPU allocation	Normal GPU allocation
2	Pinned sysmem allocation & init	**Managed** allocation & init
3	DMA transfer	**Encrypted** DMA transfer
4	Kernel launch	**Encrypted** kernel launch
5	DMA transfer	**Encrypted** DMA transfer





Confidential Computing: The Developer's View to Secure an Application and Data on NVIDIA H100 [S51684]

References

S51120	Programming Model and Applications for the Grace Hopper Superchip
S51054	Accelerating HPC applications with ISO C++ on Grace Hopper
S51755	C++ Standard Parallelism
S51210	How to Write a CUDA Program
S51705	How to Streamline Shared Memory Space With the NVSHMEM Communication Library
S51789	cuNumeric and Legate: How to Create a Distributed GPU Accelerated Library
S51176	Recent Developments in NVIDIA Math Libraries
S51413	Developing Optimal CUDA Kernels on Hopper Tensor Cores
S51205	From the Macro to the Micro: CUDA Developer Tools Find and Fix Problems at Any Scale
S51421	Optimizing at Scale: Investigating and Resolving Hidden Bottlenecks for Multi-Node Workloads
S51709	Hopper Confidential Computing: How it Works under the Hood
S51684	Confidential Computing: The Developer's View to Secure an Application and Data on NVIDIA H100
S51074	A Deep Dive into the Latest HPC Software
S51880	A Demonstration of AI and HPC Applications for NVIDIA Grace CPU
CWES52036	Connect with the Experts: What's in Your CUDA Toolbox? CUDA Profiling, Optimization, and Debugging Tools for the Latest Architectures
Whitepaper	NVIDIA Grace Hopper Superchip Architecture
Blog	CUDA 12.0 Compiler Support for Runtime LTO Using nvJitLink Library
Package	Download MathDx Today
Repo	Download CUTLASS 3.0 from the GitHub repo