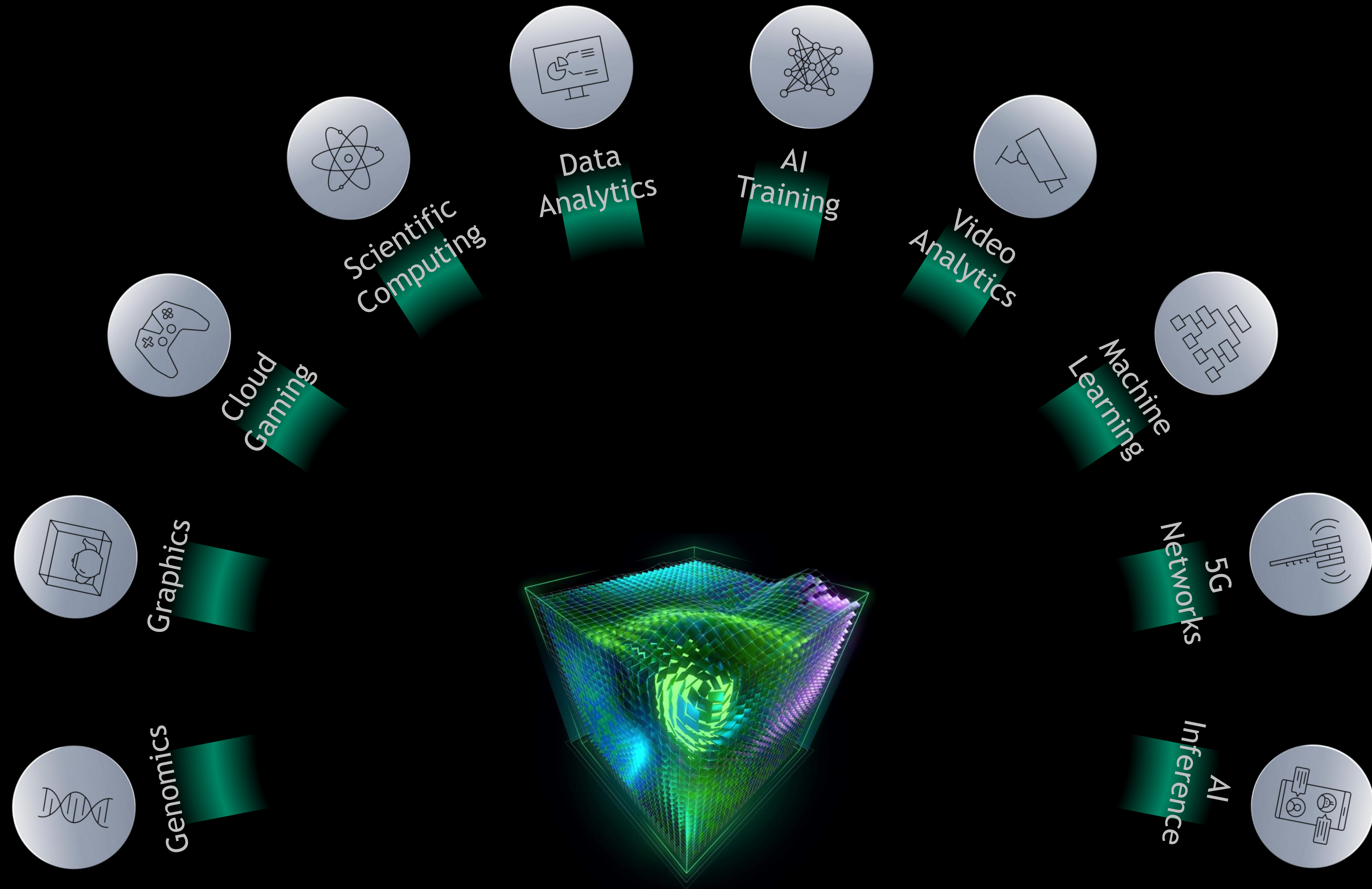
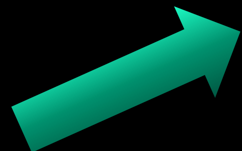
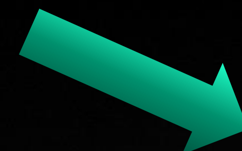
CUDA 11: NEW FEATURES AND BEYOND

Stephen Jones, GTC 2020

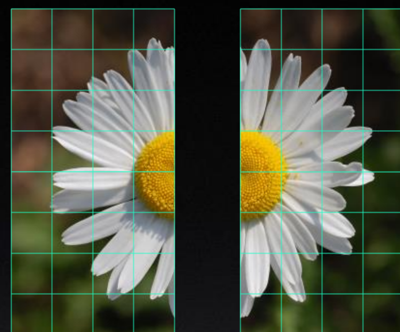
HUGE BREADTH OF PLATFORMS, SYSTEMS, LANGUAGES

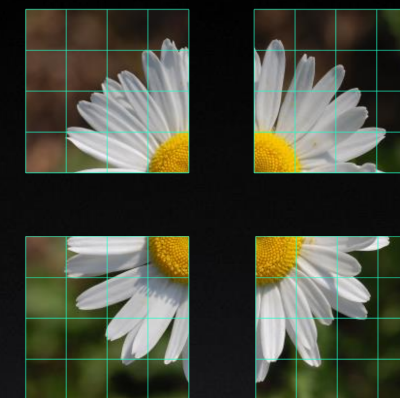




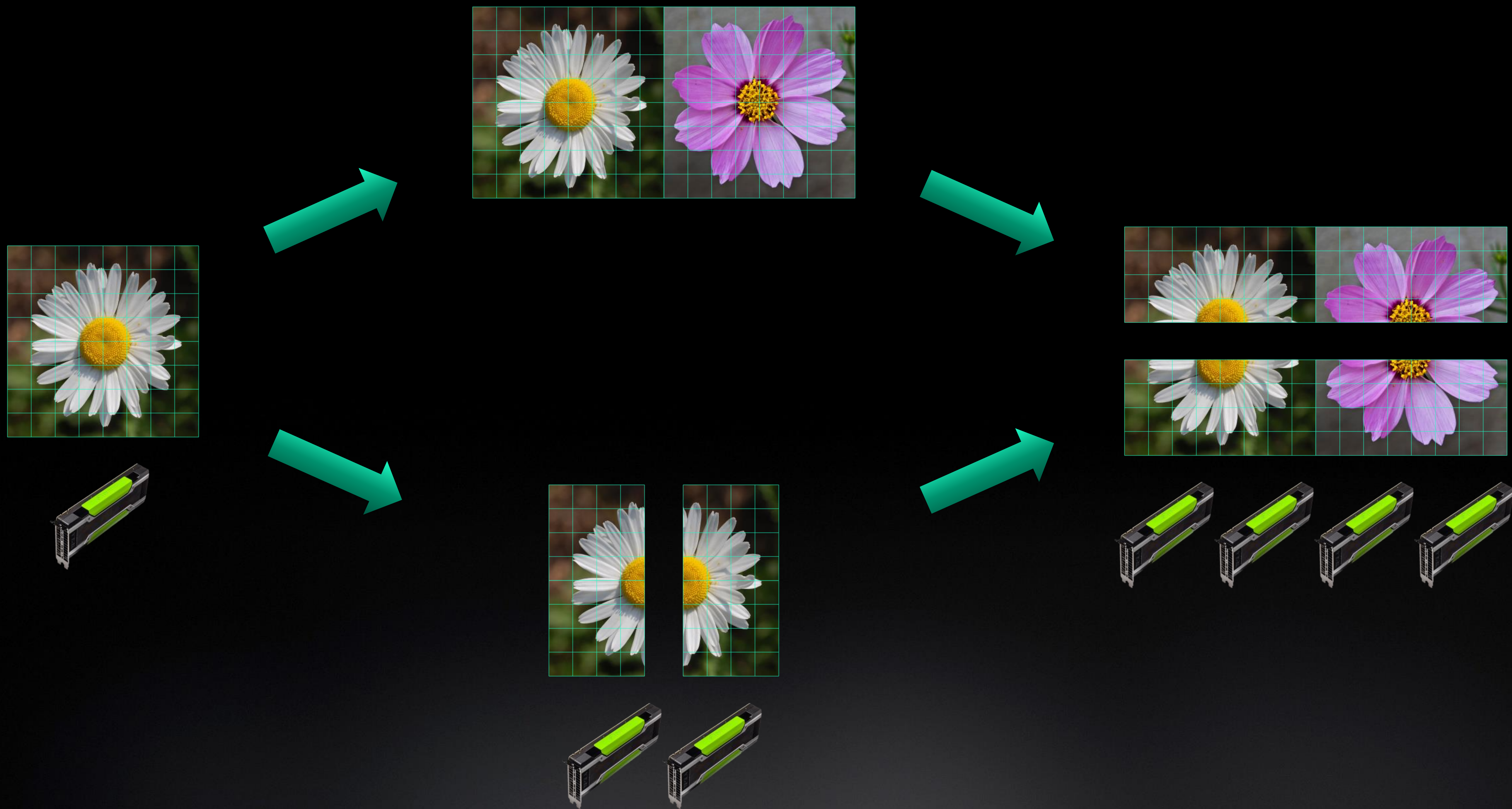


Weak Scaling
Larger Problem





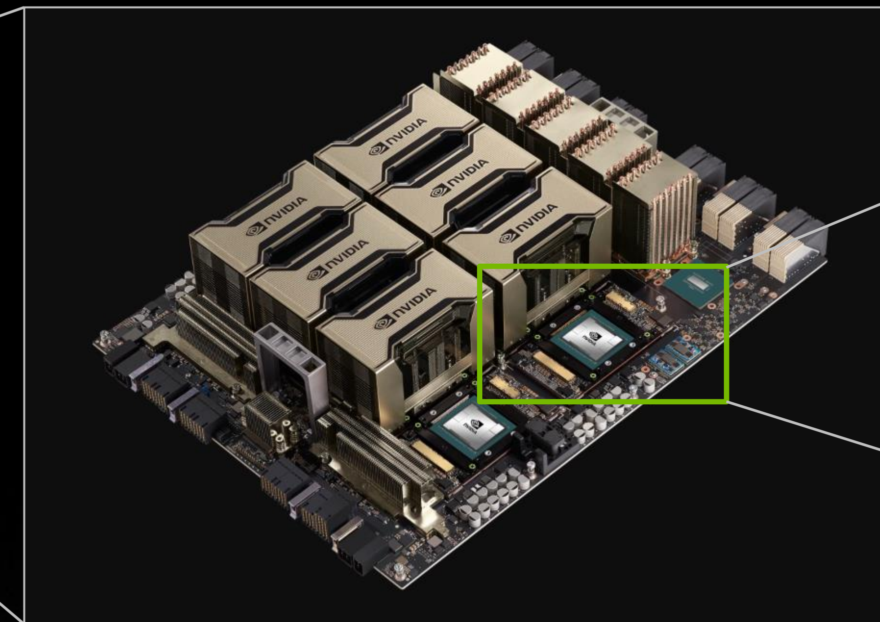
Strong Scaling
Faster Solution

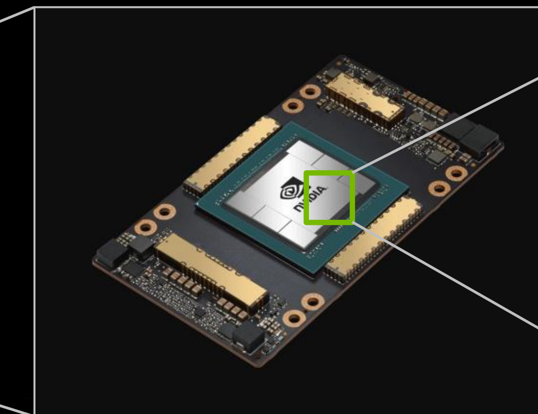


Mixed Scaling

Larger & Faster

HIERARCHY OF SCALES





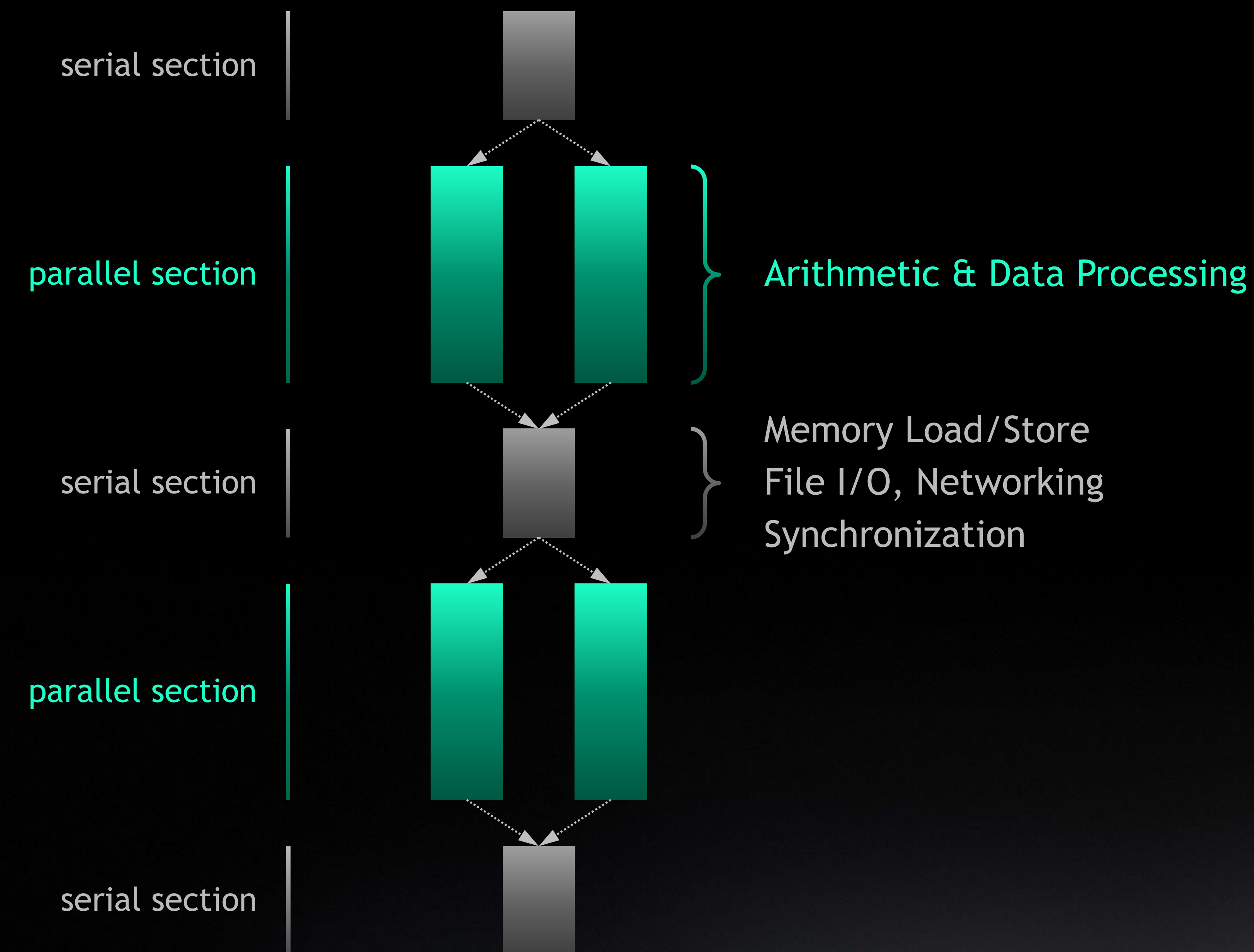
Multi-System Rack
Unlimited Scale

Multi-GPU System
8 GPUs

Multi-SM GPU
108 Multiprocessors

Multi-Core SM
2048 threads

AMDAHL'S LAW



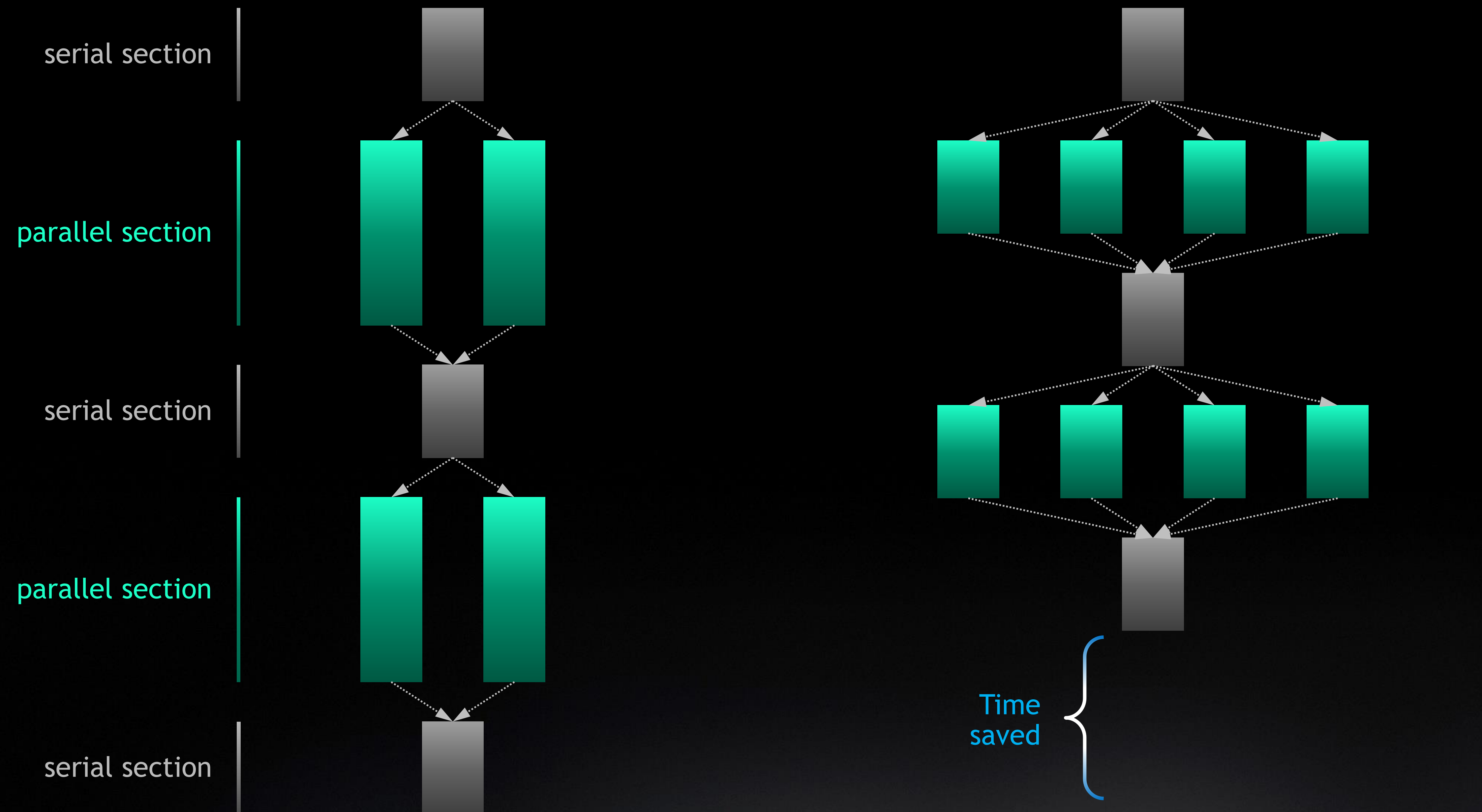
Arithmetic & Data Processing

Memory Load/Store
File I/O, Networking
Synchronization

Some Parallelism

Program time =
sum(serial times + parallel times)

AMDAHL'S LAW



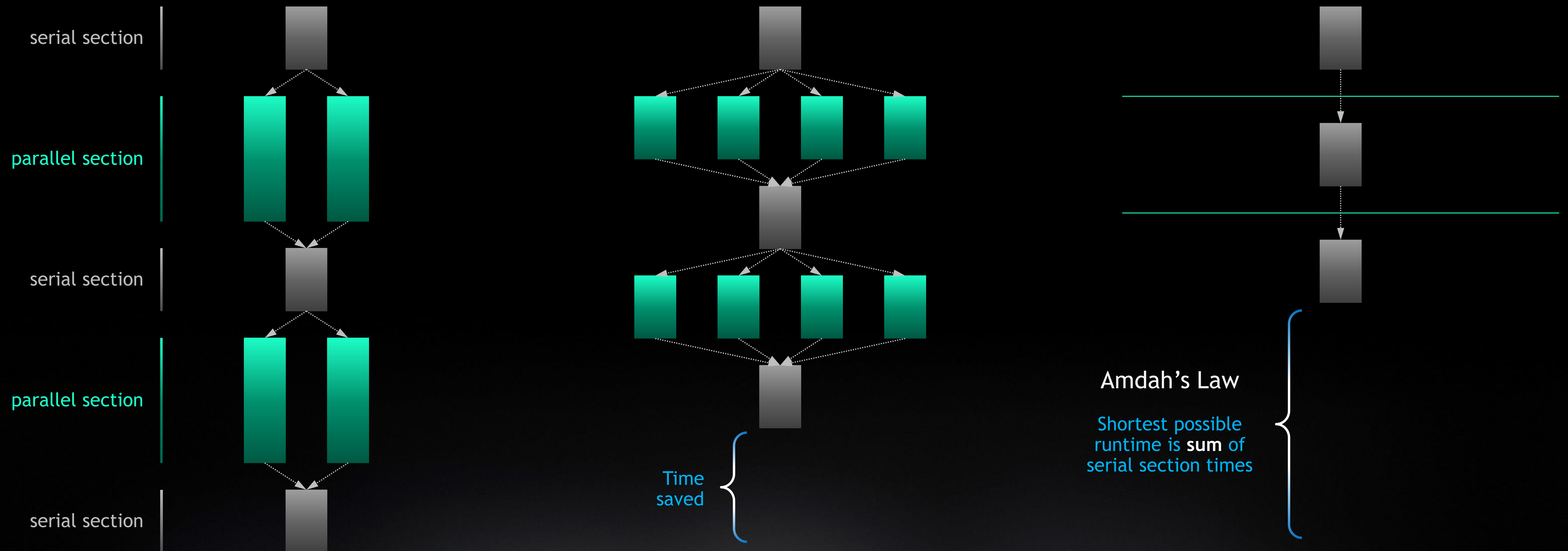
Some Parallelism

Program time =
sum(serial times + parallel times)

Increased Parallelism

Parallel sections take less time
Serial sections take same time

AMDAHL'S LAW



Some Parallelism

Program time =
sum(serial times + parallel times)

Increased Parallelism

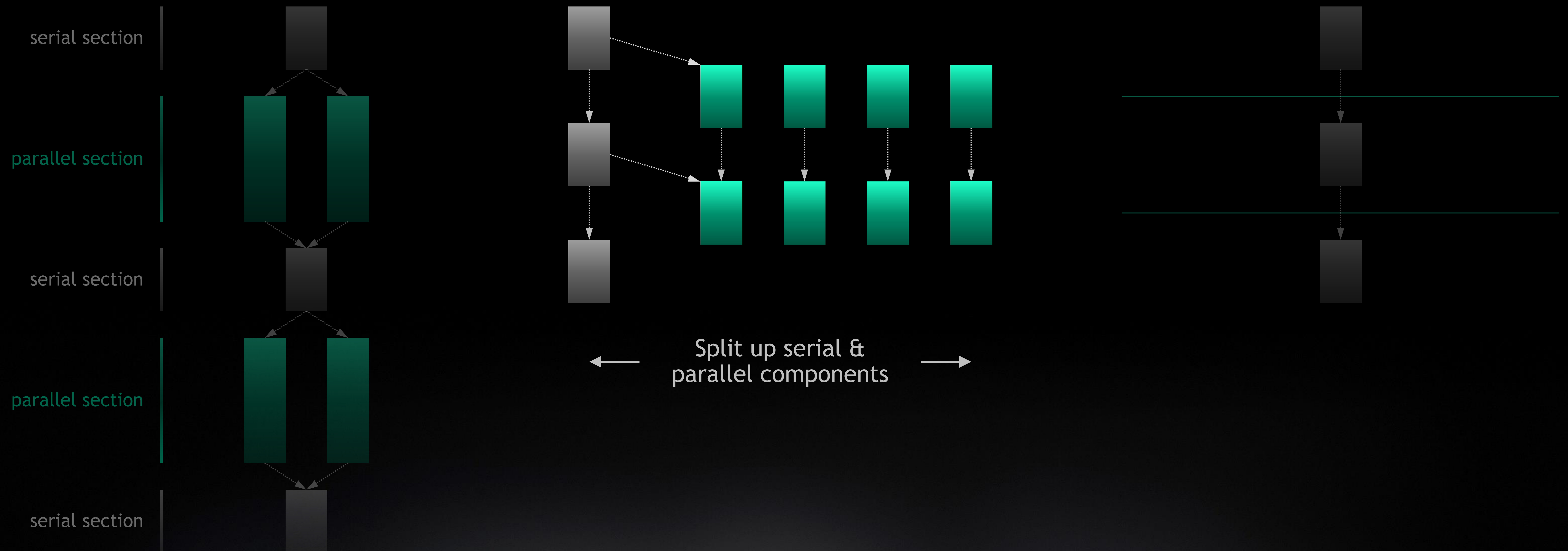
Parallel sections take less time
Serial sections take same time

Infinite Parallelism

Parallel sections take no time
Serial sections take same time

NVIDIA

OVERCOMING AMDAHL: ASYNCHRONY & LATENCY



Some Parallelism

Program time =
sum(serial times + parallel times)

Task Parallelism

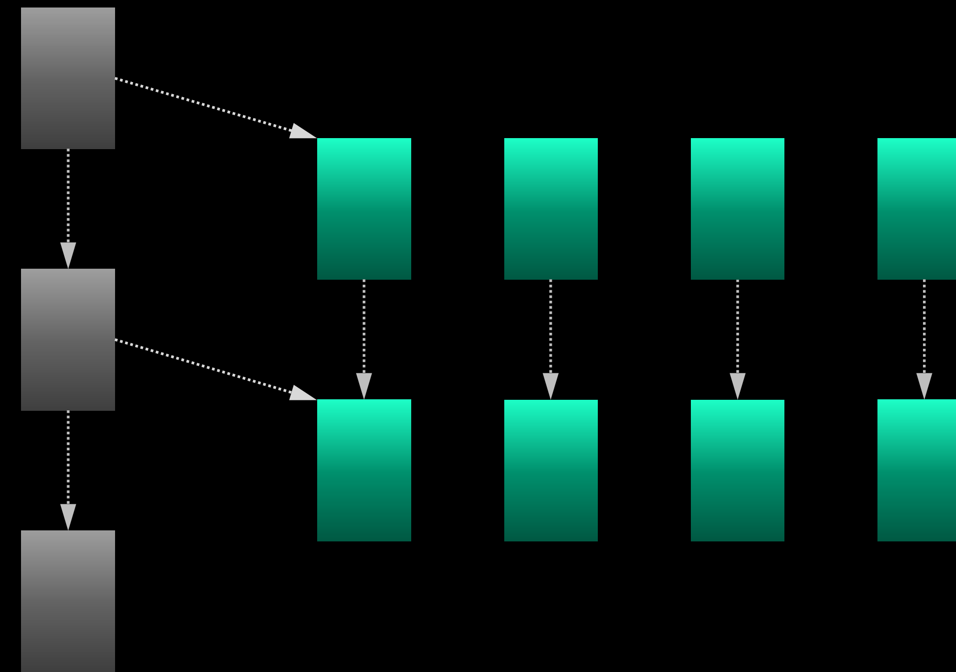
Parallel sections overlap with serial sections

Infinite Parallelism

Parallel sections take no time
Serial sections take same time

NVIDIA.

OVERCOMING AMDAHL: ASYNCHRONY & LATENCY

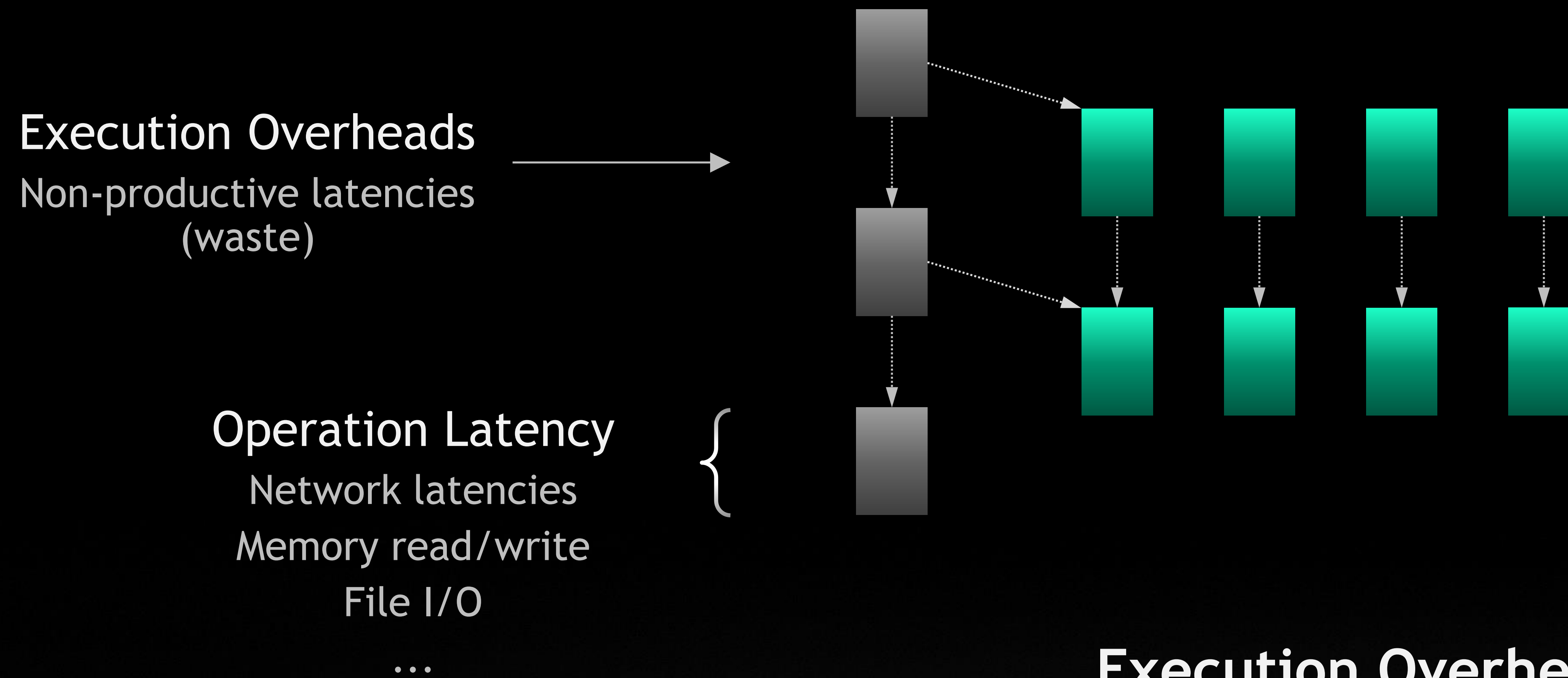


CUDA Concurrency Mechanisms At Every Scope

CUDA Kernel	Threads, Warps, Blocks, Barriers
Application	CUDA Streams, CUDA Graphs
Node	Multi-Process Service, GPU-Direct
System	NCCL, CUDA-Aware MPI, NVSHMEM

NVIDIA.

OVERCOMING AMDAHL: ASYNCHRONY & LATENCY



Execution Overheads are waste

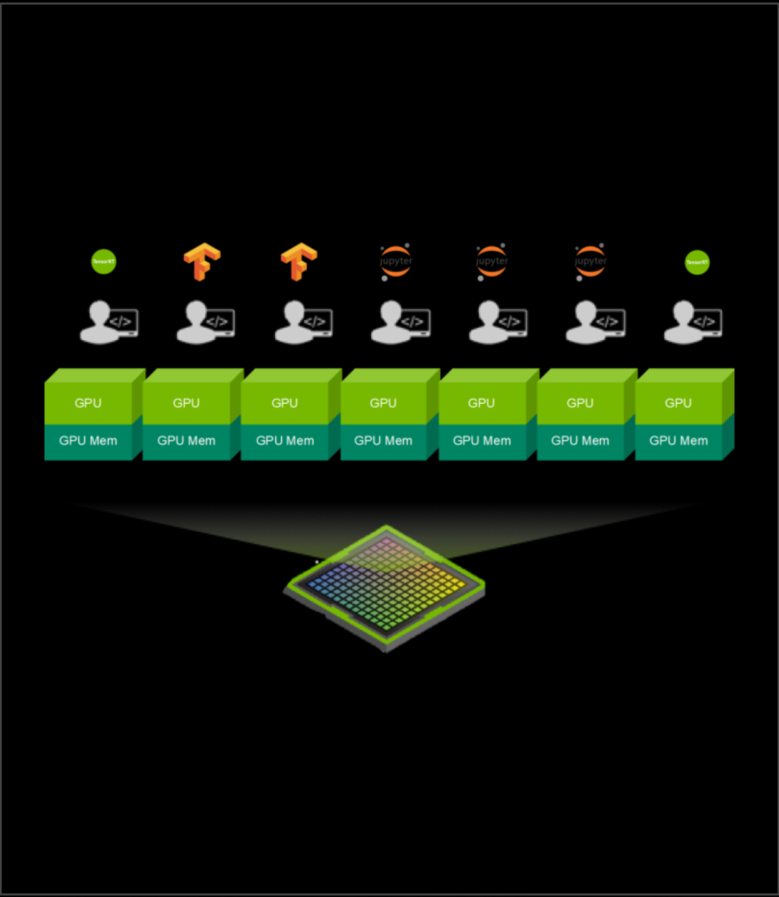
Reduced through hardware & system efficiency improvements

Operation Latencies are the cost of doing work

Improve through hardware & software optimization

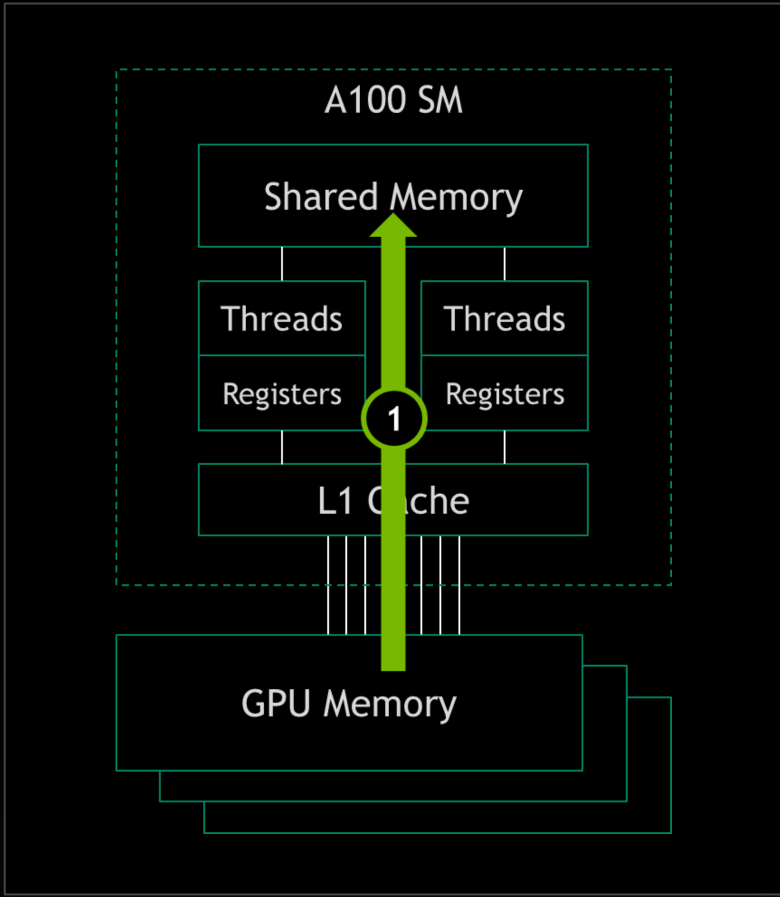
NVIDIA

CUDA KEY INITIATIVES



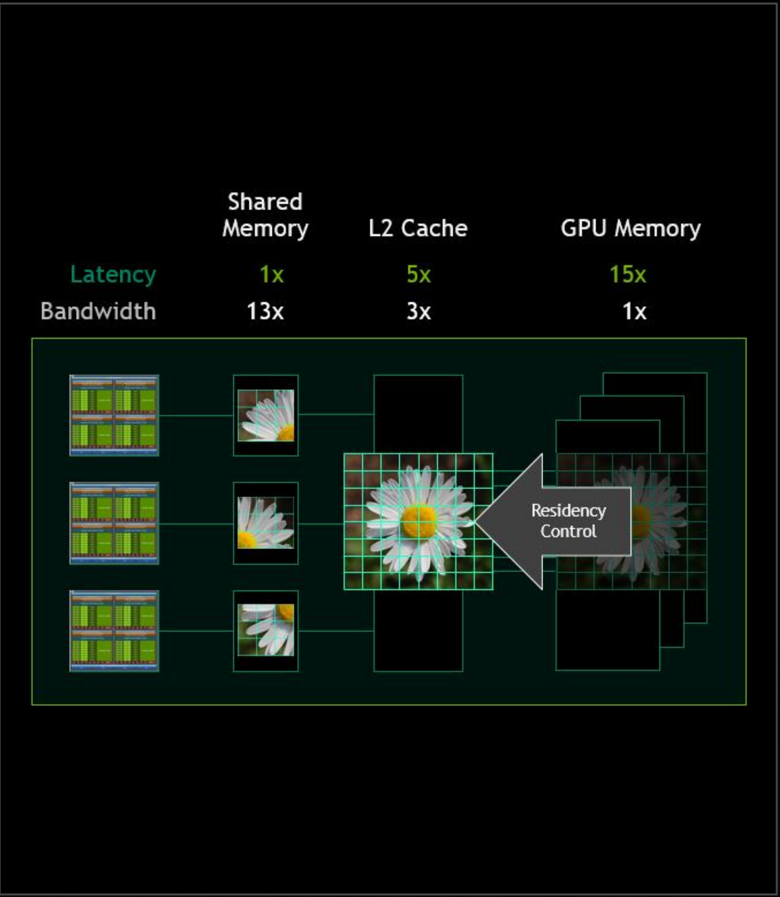
Hierarchy

Programming and running systems at every scale



Asynchrony

Creating concurrency at every level of the hierarchy



Latency

Overcoming Amdahl with lower overheads for memory & processing

```
// ISO C++, __host__ only
#include <atomic>
std::atomic<int> x;

// CUDA C++, __host__ __device__
// Strictly conforming to the ISO C++
#include <cuda/std/atomic>
cuda::std::atomic<int> x;

// CUDA C++, __host__ __device__
// Conforming extensions to ISO C++
#include <cuda/atomic>
cuda::atomic<int, cuda::thread_scope_block> x;
```

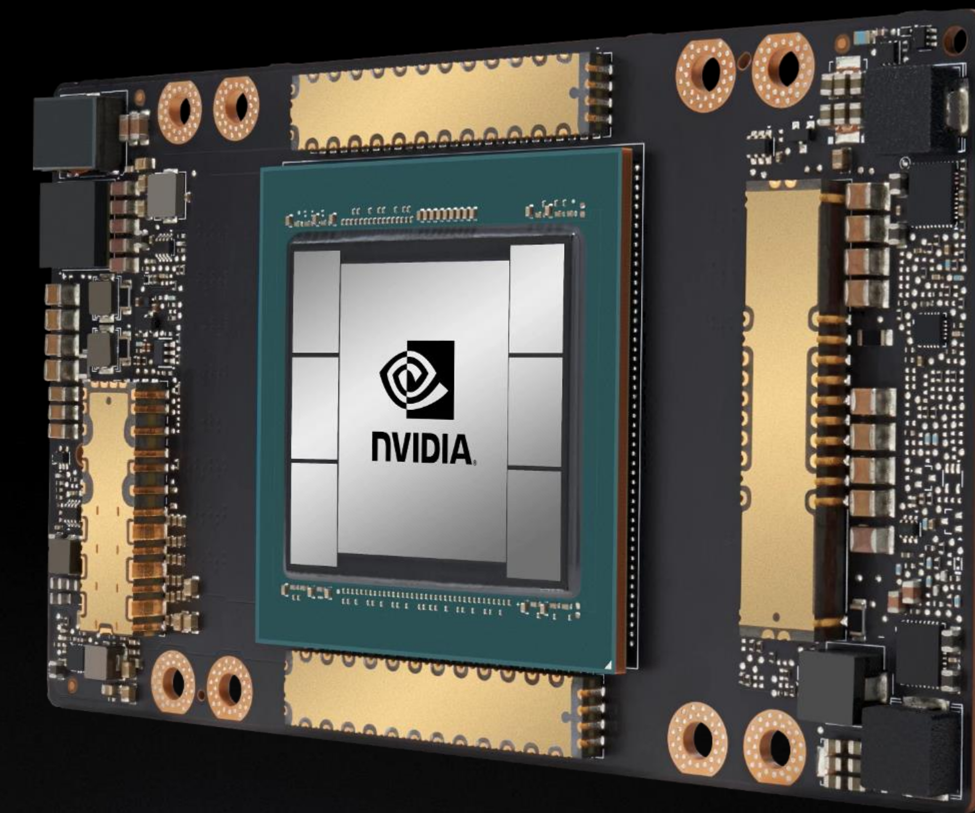
Language

Supporting and evolving Standard Languages

NVIDIA

ANNOUNCING THE NVIDIA AMPERE GPU ARCHITECTURE

	V100	A100
SMs	80	108
Tensor Core Precision	FP16	FP64, TF32, BF16, FP16, I8, I4, B1
Shared Memory per Block	96 kB	160 kB
L2 Cache Size	6144 kB	40960 kB
Memory Bandwidth	900 GB/sec	1555 GB/sec
NVLink Interconnect	300 GB/sec	600 GB/sec

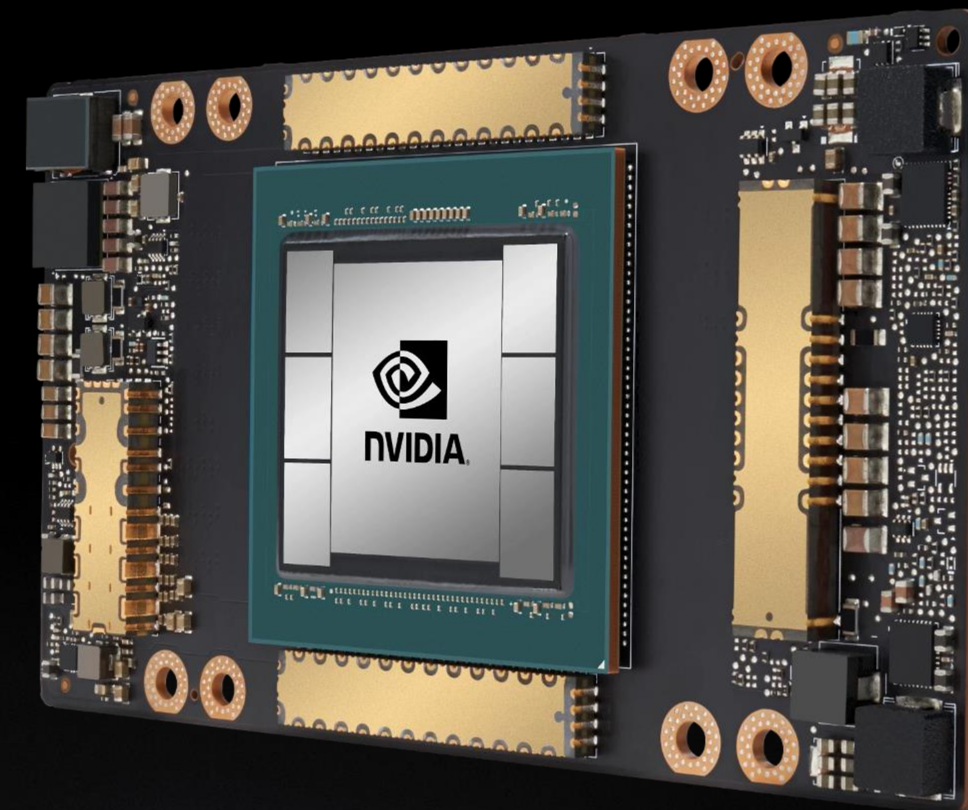


For more information see: S21730 - Inside the NVIDIA Ampere Architecture && www.nvidia.com/nvidia-ampere-architecture-whitepaper

ANNOUNCING THE NVIDIA AMPERE GPU ARCHITECTURE

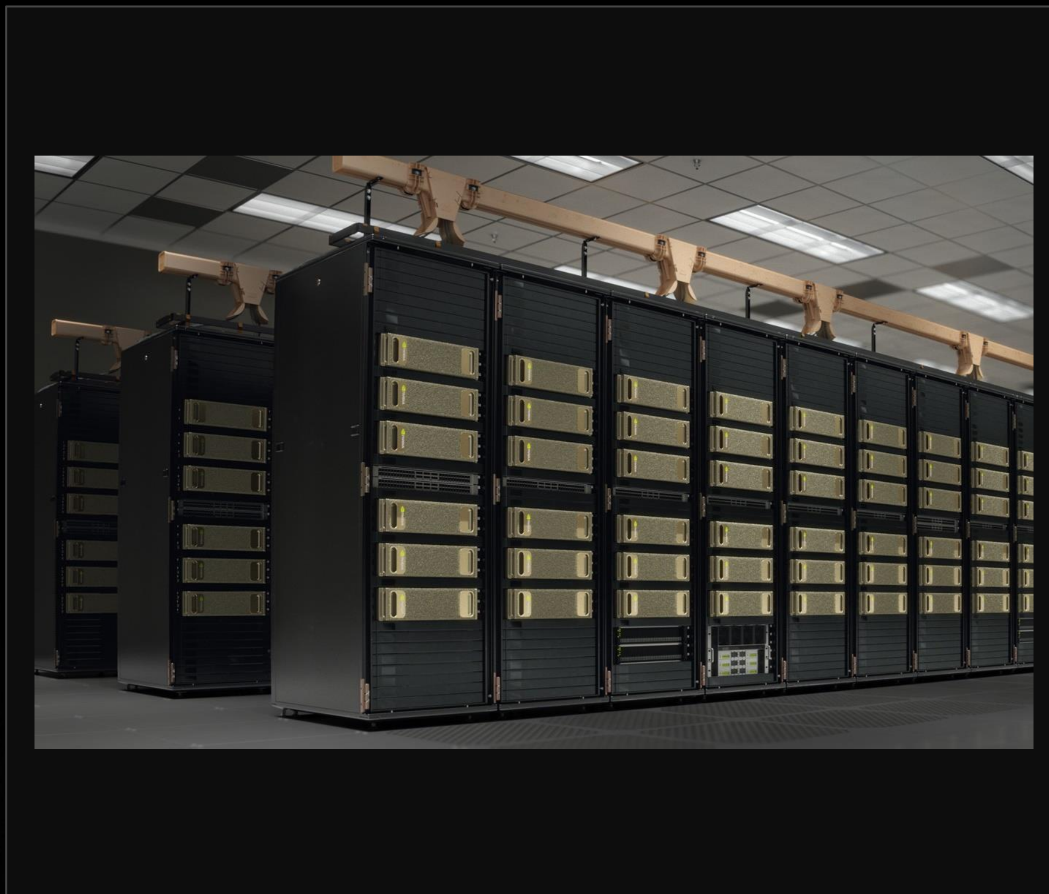
NVIDIA GA100 Key Architectural Features

- Multi-Instance GPU
- Advanced barriers
- Asynchronous data movement
- L2 cache management
- Task graph acceleration
- New Tensor Core precisions



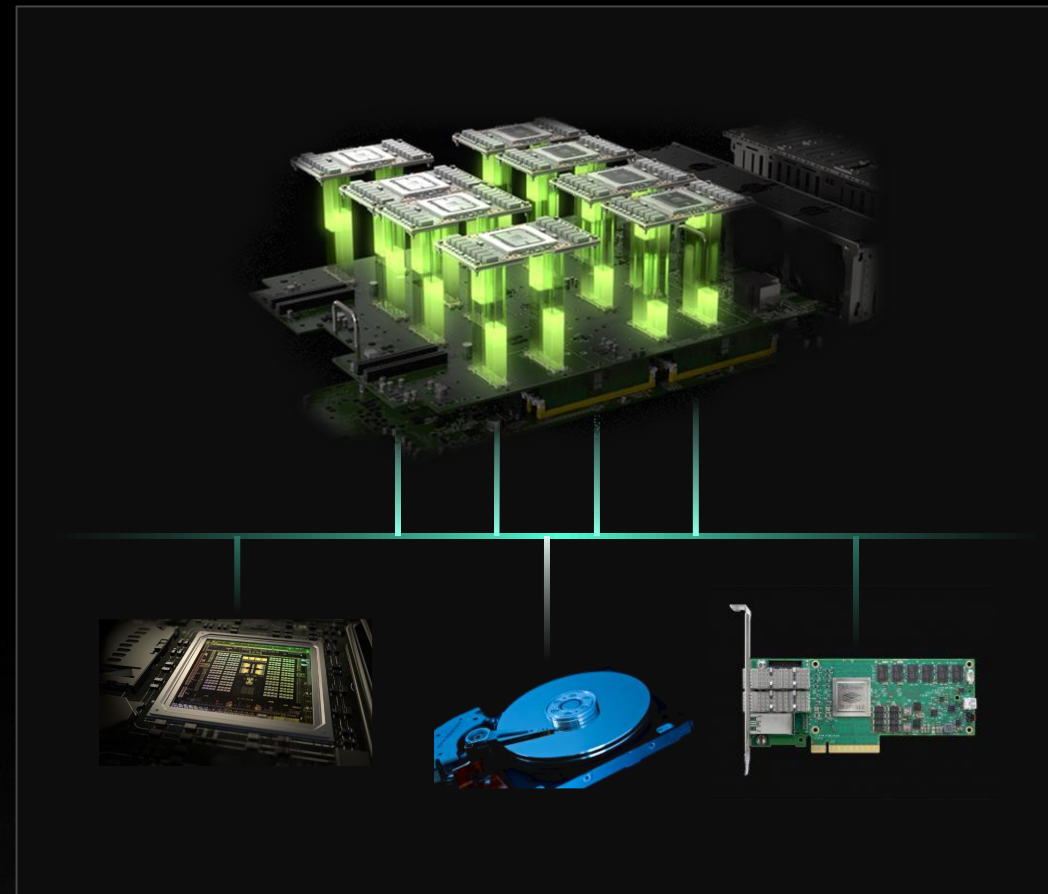
CUDA PLATFORM: TARGETS EACH LEVEL OF THE HIERARCHY

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



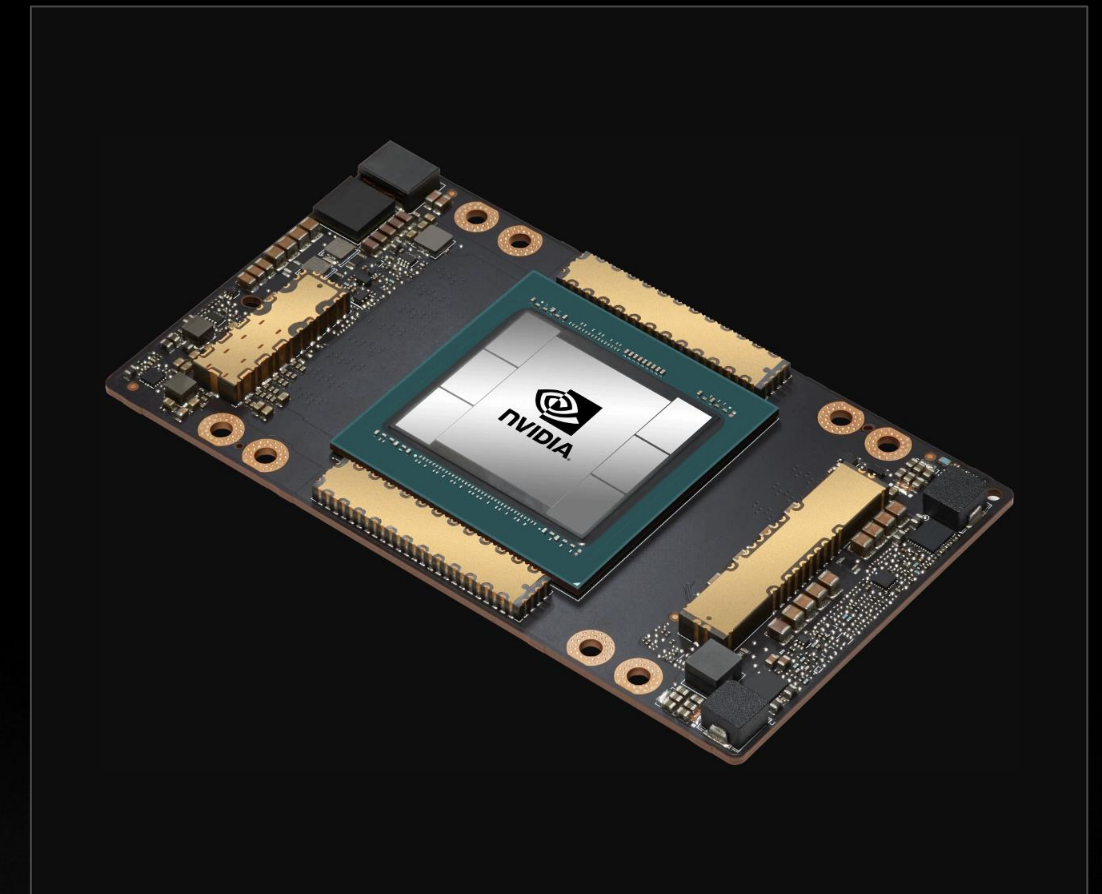
System Scope

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

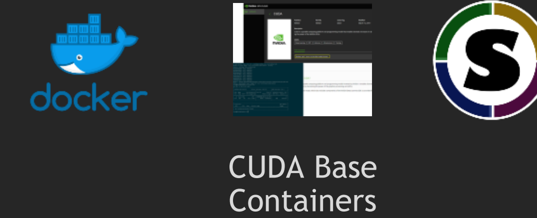
CUDA ON ARM

Technical Preview Release – Available for Download

HPC APP and vis CONTAINERS

LAMMPS
GROMACS
MILC
NAMD
HOOMD-blue
VMD
Paraview

NGC TensorFlow



CUDA Base Containers

CUDA-X LIBRARIES

cuBLAS
cuSPARSE
cuFFT
cuRAND
cuSOLVER
Math API
Thrust
libcu++

GRAPHICS

NVIDIA OPTIX

NVIDIA IndeX

CUDA TOOLKIT

GCC 8.3

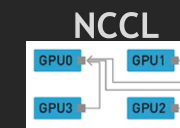
Arm C/C++
nvc++ (PGI)

COMPILERS

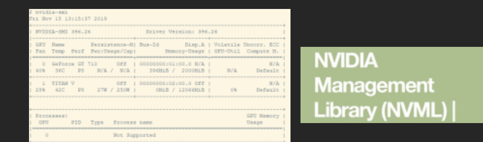
Debugger:
Nsight Systems
Profilers:
CUPTIv2 Tracing APIs, Metrics
Nsight Compute

COMMS LIBRARIES

NCCL



CUDA Aware MPI



OPERATING SYSTEMS

RHEL 8.0 for Arm

Ubuntu 18.04.3 LTS

OEM SYSTEMS

HPE Apollo 70

Gigabyte R281

GPUs

Tesla V100

DATA CENTER GPU MANAGER (DCGM)

GPU Management in the Accelerated Data Center

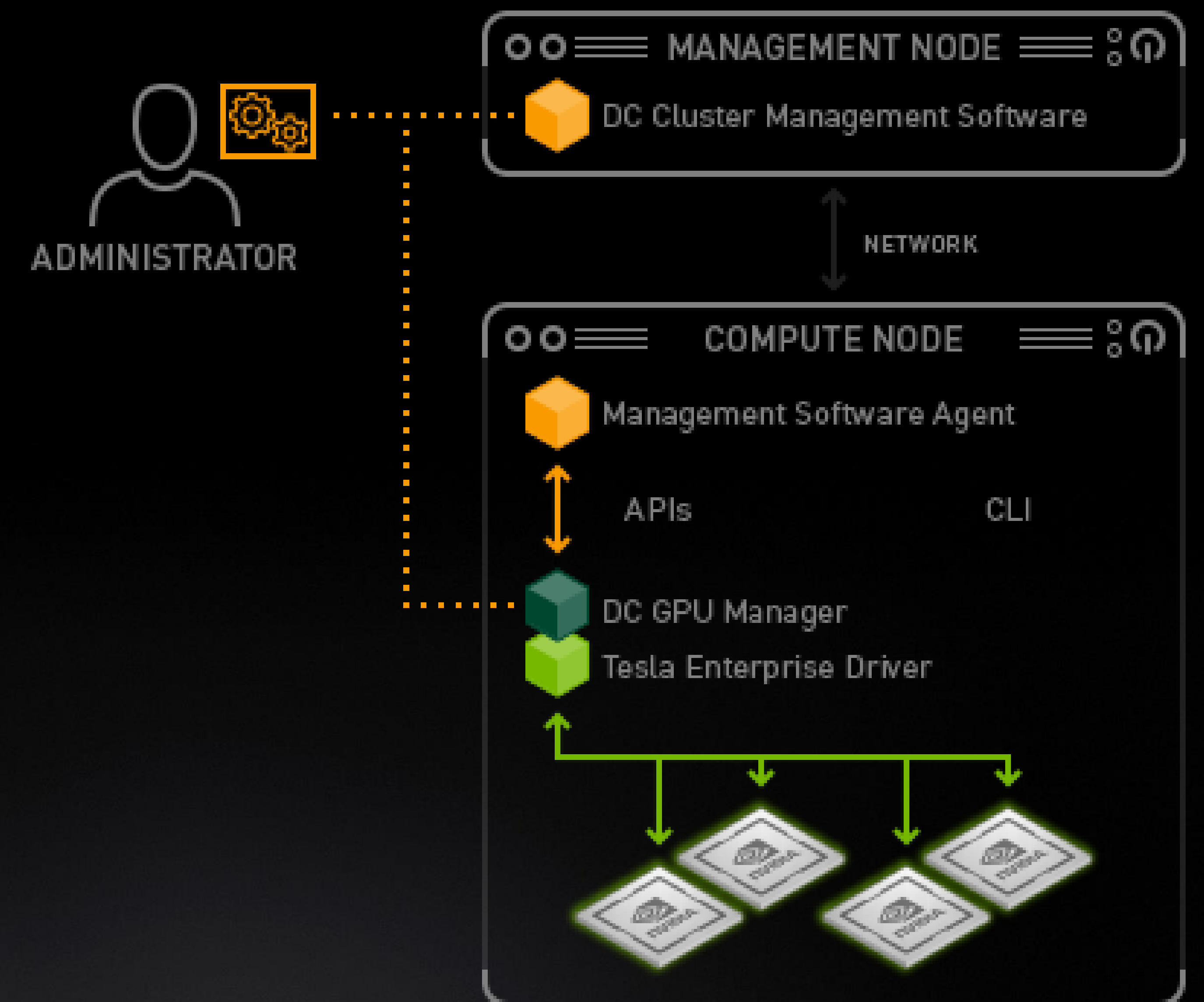
Intended for

- Online monitoring of Data Center GPUs in production
- Production line testing/pre-production testing of servers

Active health monitoring	GPU Metrics
NVSwitch management	Comprehensive diagnostics
System alerts	Governance policies

Supports Data Center SKUs (Kepler+) on Linux x86_64, POWER architectures

https://developer.nvidia.com/dcgm



NVIDIA.

06:59:58
JOB RUNTIME

84%
TIME GPUS ACTIVE 8

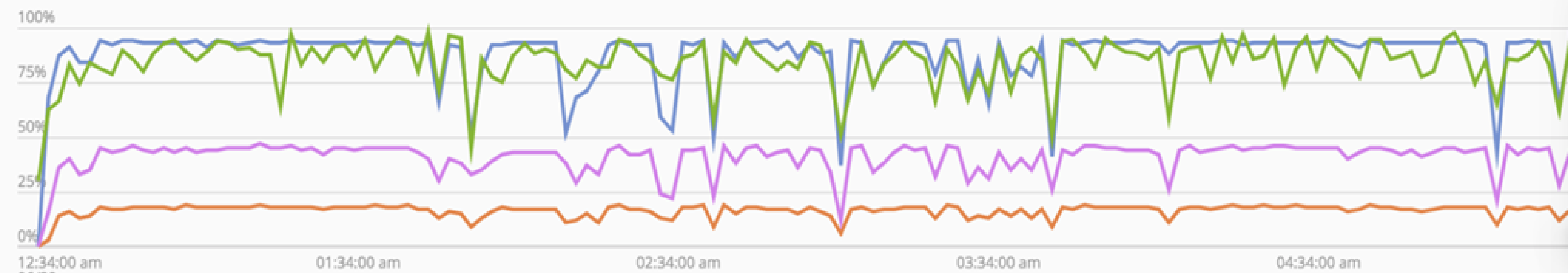
89%
GPU UTILIZATION

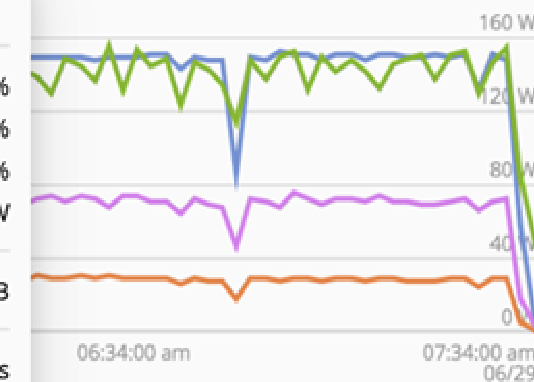
Sample Resolution at 2 minutes

Download CSV   Reset Focus

GPU Active Tensor Cores Active GPU Memory Active GPU Power

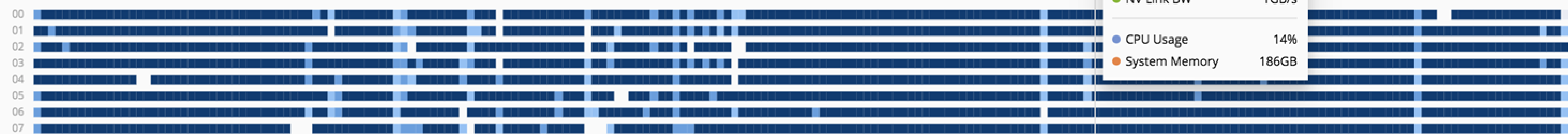
Mean   Data Selection



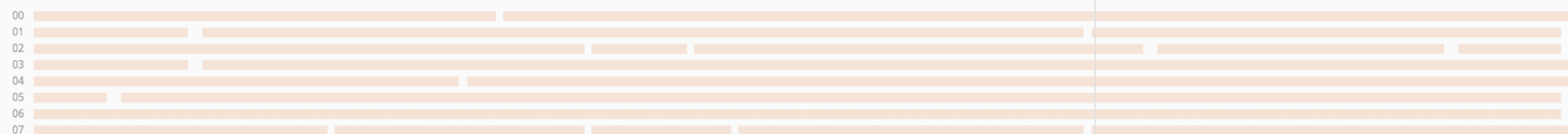


Collapse Heatmap

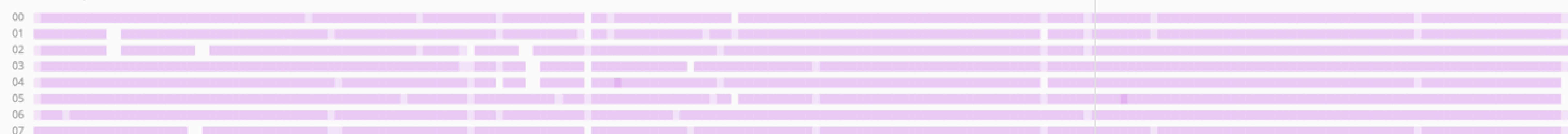
GPU Active



Tensor Cores Active



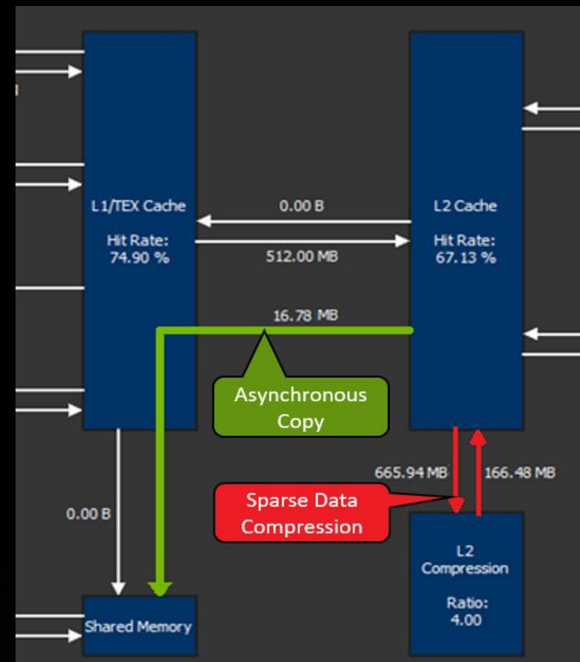
GPU Memory Active



GPU Power

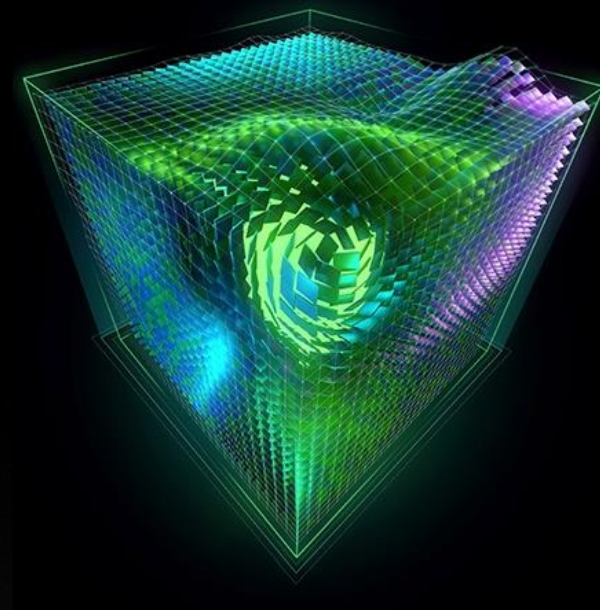
NVIDIA.

GPU & PLATFORM SUPPORT ACROSS DEVELOPER TOOLS

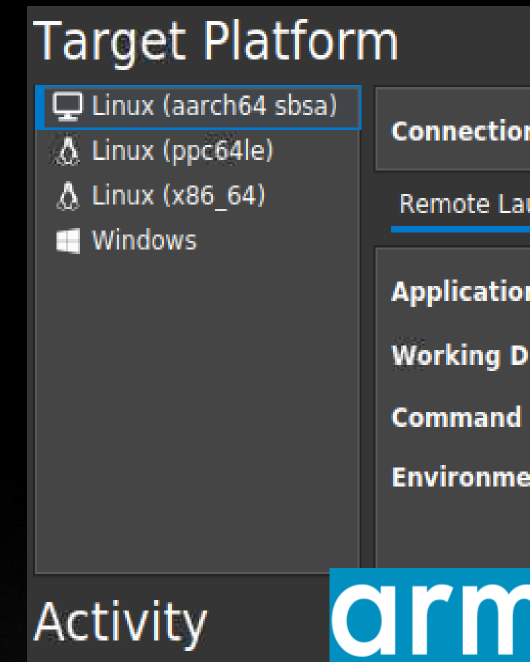


Chips Update

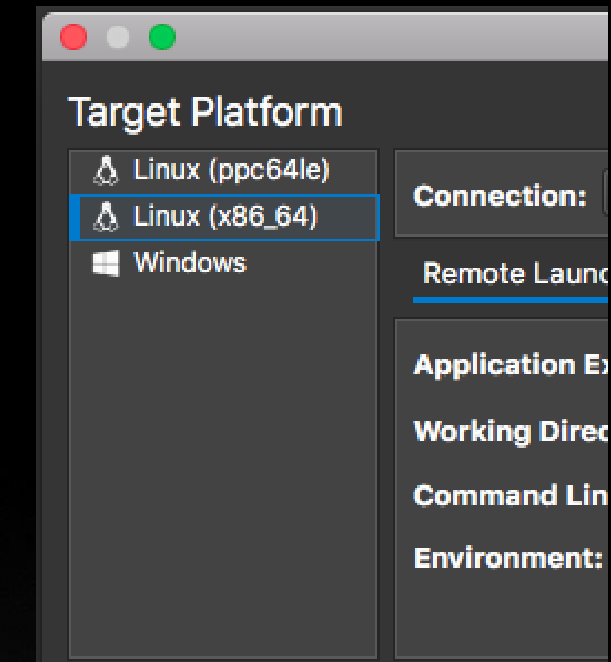
A100 GPU Support



CUDA 11.0 support



Arm SBSA support



OS support updates

POWER9 support

MacOSX host platform only

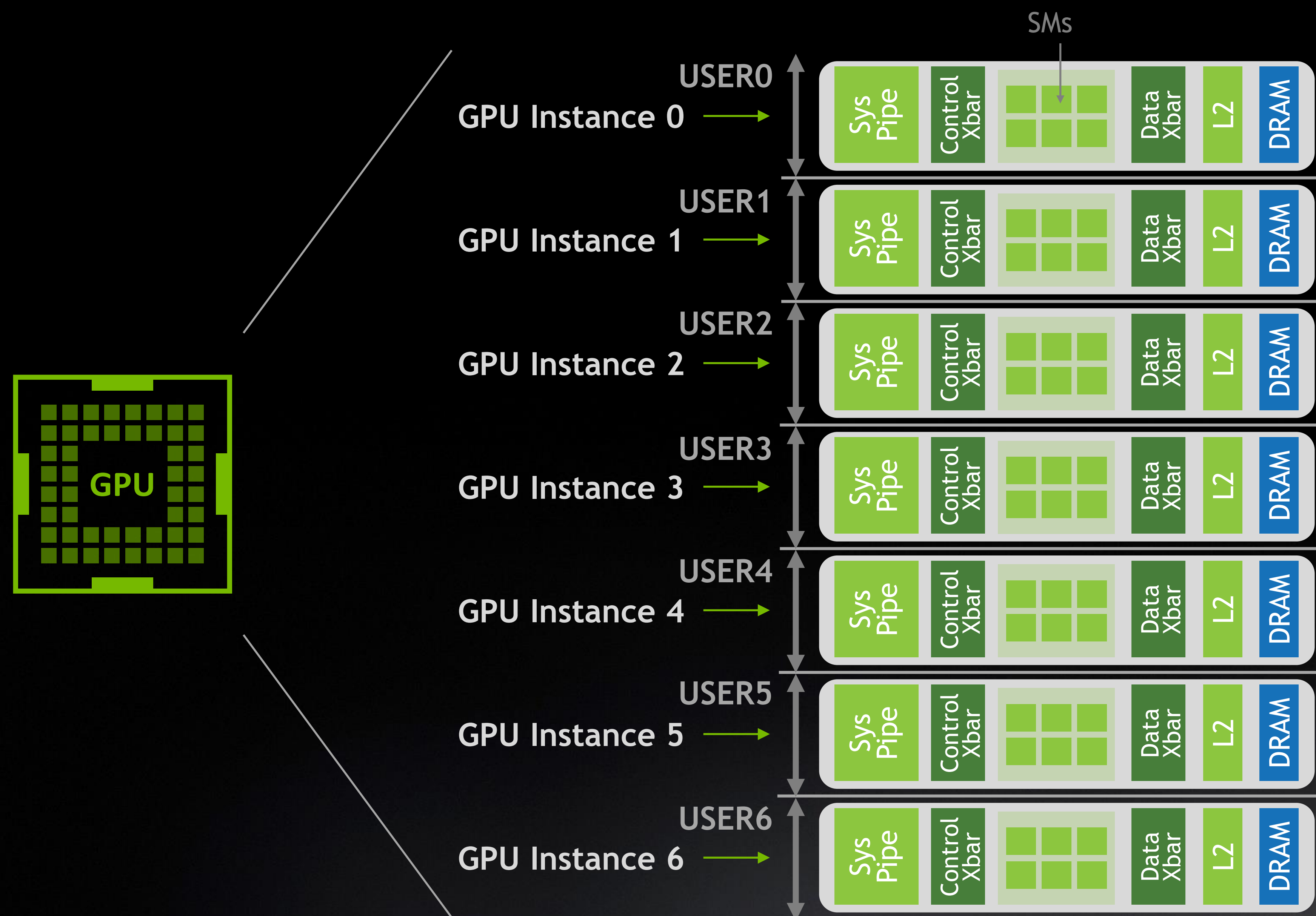
Removal of Windows 7 support

For more information see: S22043 – CUDA Developer Tools: Overview and Exciting New Features

NVIDIA.

NEW MULTI-INSTANCE GPU (MIG)

Divide a Single GPU Into Multiple *Instances*, Each With Isolated Paths Through the Entire Memory System



Up To 7 GPU Instances In a Single A100

Full software stack enabled on each instance, with dedicated SM, memory, L2 cache & bandwidth

Simultaneous Workload Execution With Guaranteed Quality Of Service

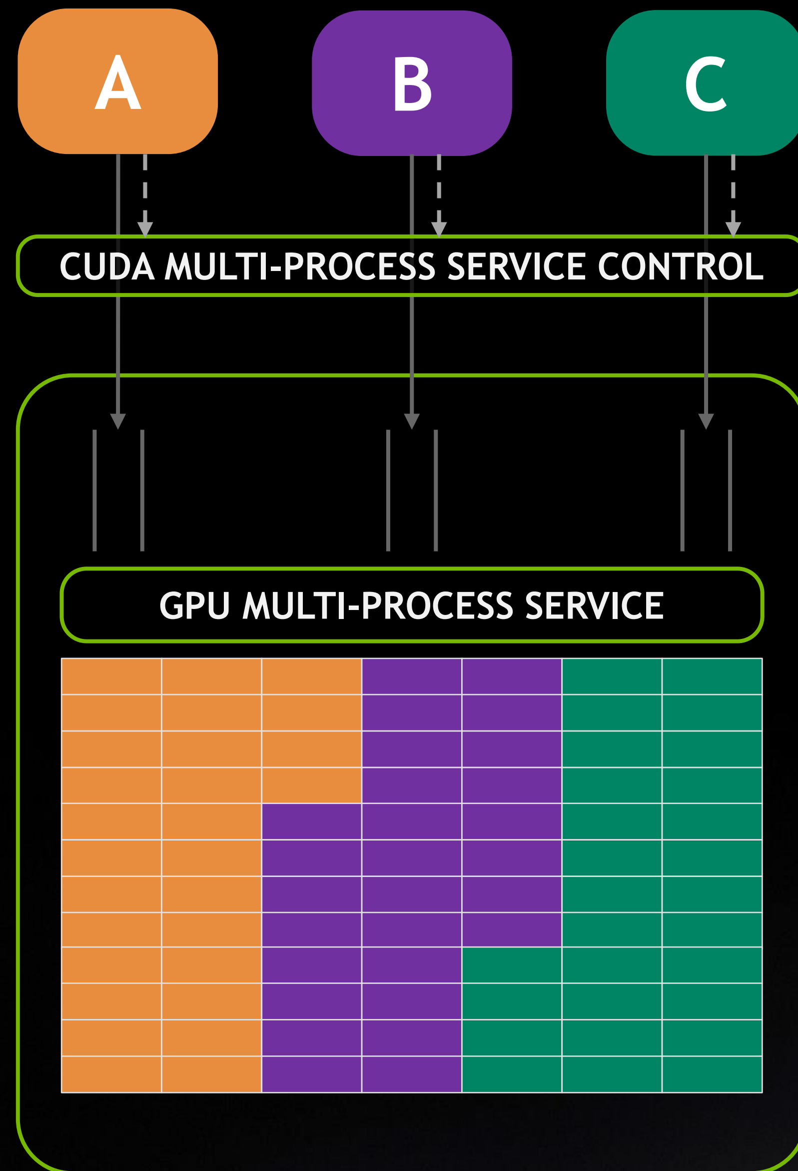
All MIG instances run in parallel with predictable throughput & latency, fault & error isolation

Diverse Deployment Environments

Supported with Bare metal, Docker, Kubernetes Pod, Virtualized Environments

NVIDIA.

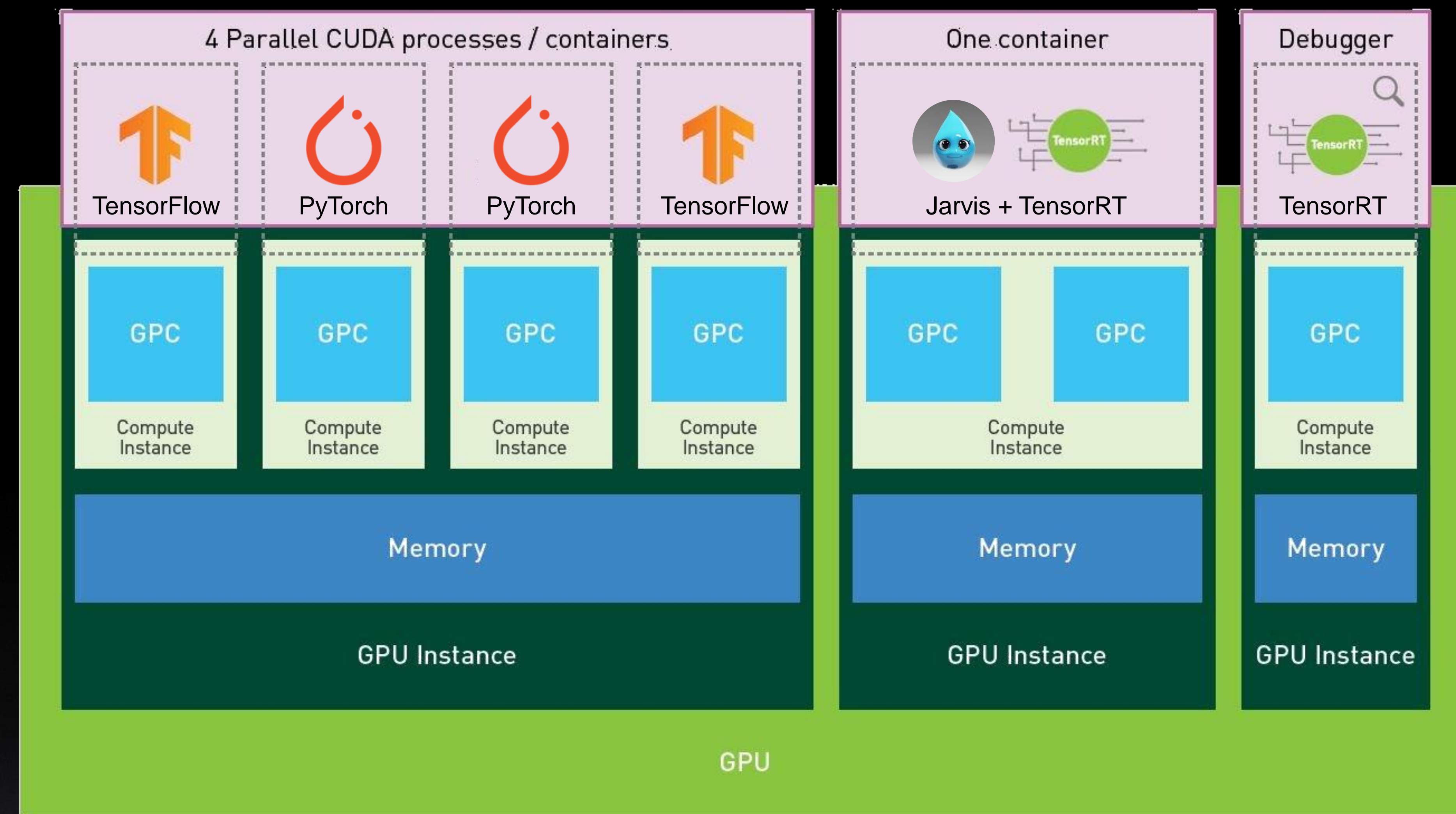
LOGICAL VS. PHYSICAL PARTITIONING



Multi-Process Service

Dynamic contention for GPU resources

Single tenant



Multi-Instance GPU

Hierarchy of instances with guaranteed resource allocation

Multiple tenants

CUDA CONCURRENCY MECHANISMS

	Streams	MPS	MIG
Partition Type	Single process	Logical	Physical
Max Partitions	Unlimited	48	7
Fractional Provisioning	No	Yes	Yes
Memory Protection	No	Yes	Yes
Memory Bandwidth QoS	No	No	Yes
Fault Isolation	No	No	Yes
Cross-Partition Interop	Always	IPC	Limited IPC
Reconfigure	Dynamic	Process launch	When idle

For more information see: S21975 - Inside NVIDIA's Multi-Instance GPU Feature

CUDA VIRTUAL MEMORY MANAGEMENT

Breaking Memory Allocation Into Its Constituent Parts

1. Reserve Virtual Address Range
   `cuMemAddressReserve/Free`
2. Allocate Physical Memory Pages
   `cuMemCreate/Release`
3. Map Pages To Virtual Addresses
   `cuMemMap/Unmap`
4. Manage Access Per-Device
   `cuMemSetAccess`

Control & reserve address ranges

Can remap physical memory

Fine-grained access control

Manage inter-GPU peer-to-peer sharing on a per-allocation basis

Inter-process sharing

For more information see: https://devblogs.nvidia.com/introducing-low-level-gpu-virtual-memory-management/

CUDA VIRTUAL MEMORY MANAGEMENT

Basic Memory Allocation Example

1. Reserve Virtual Address Range
   `cuMemAddressReserve/Free`
2. Allocate Physical Memory Pages
   `cuMemCreate/Release`
3. Map Pages To Virtual Addresses
   `cuMemMap/Unmap`
4. Manage Access Per-Device
   `cuMemSetAccess`

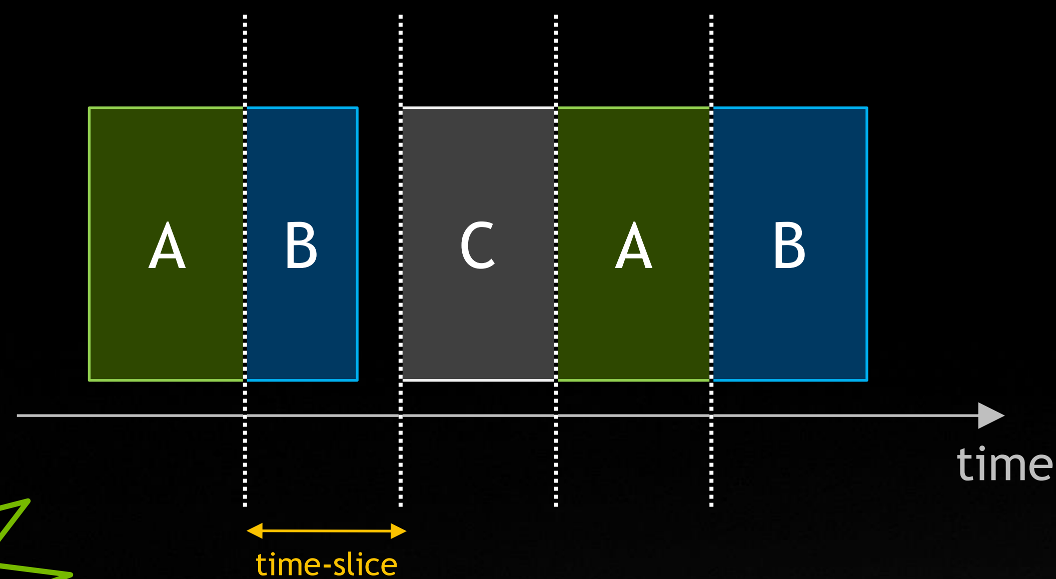
```
// Allocate memory
cuMemCreate(&handle, size, &allocProps, 0);

// Reserve address range
cuMemAddressReserve(&ptr, size, alignment,
                    fixedVa, 0);

// Map memory to address range
cuMemMap(ptr, size, offset, handle, 0);

// Make the memory accessible on all devices
cuMemSetAccess(ptr, size, rwOnDeviceSet,
               deviceCount);
```


NVIDIA

EXECUTION SCHEDULING & MANAGEMENT

Pre-emptive scheduling

Processes share GPU through time-slicing
Scheduling managed by system

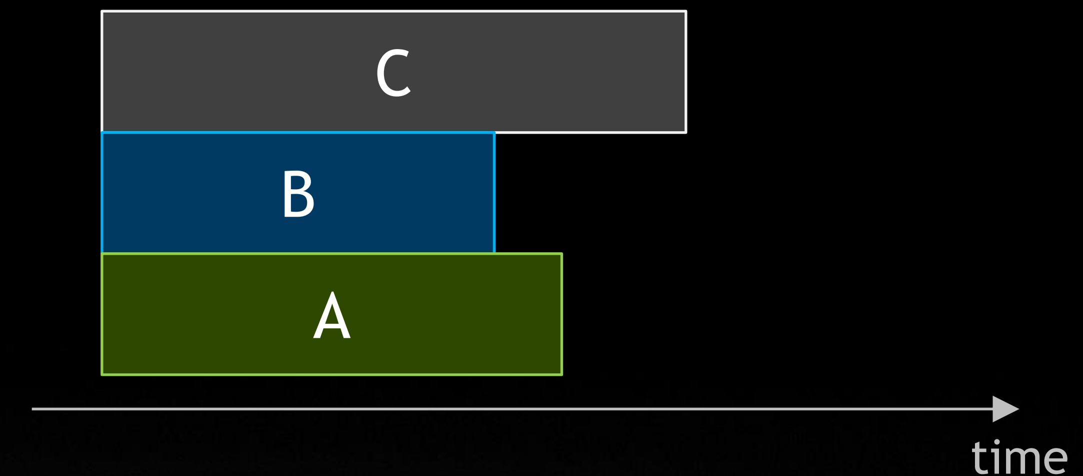



```
$ nvidia-smi compute-policy
  --set-timeslice={default, short, medium, long}
```

Time-slice configurable via **nvidia-smi**

Concurrent scheduling

Processes run on GPU simultaneously
User creates & manages scheduling streams



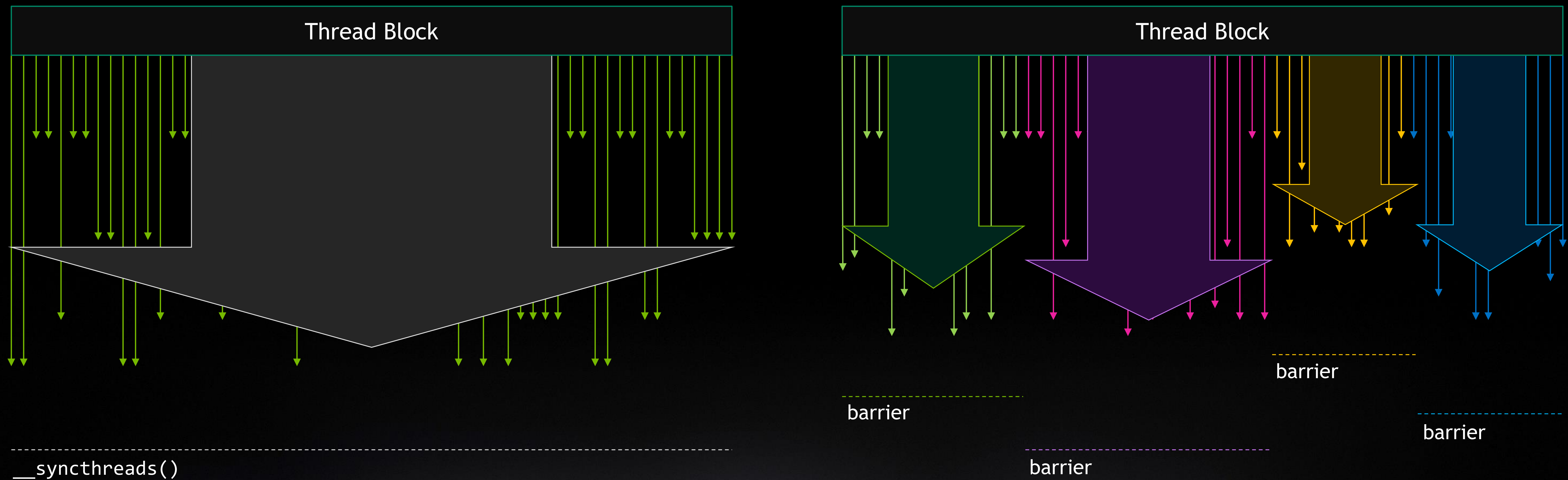

```
cudaStreamCreateWithPriority(pStream, flags, priority);
cudaDeviceGetStreamPriorityRange(leastPriority, greatestPriority);
```

CUDA 11.0 adds a new **stream priority** level

NVIDIA

FINE-GRAINED SYNCHRONIZATION

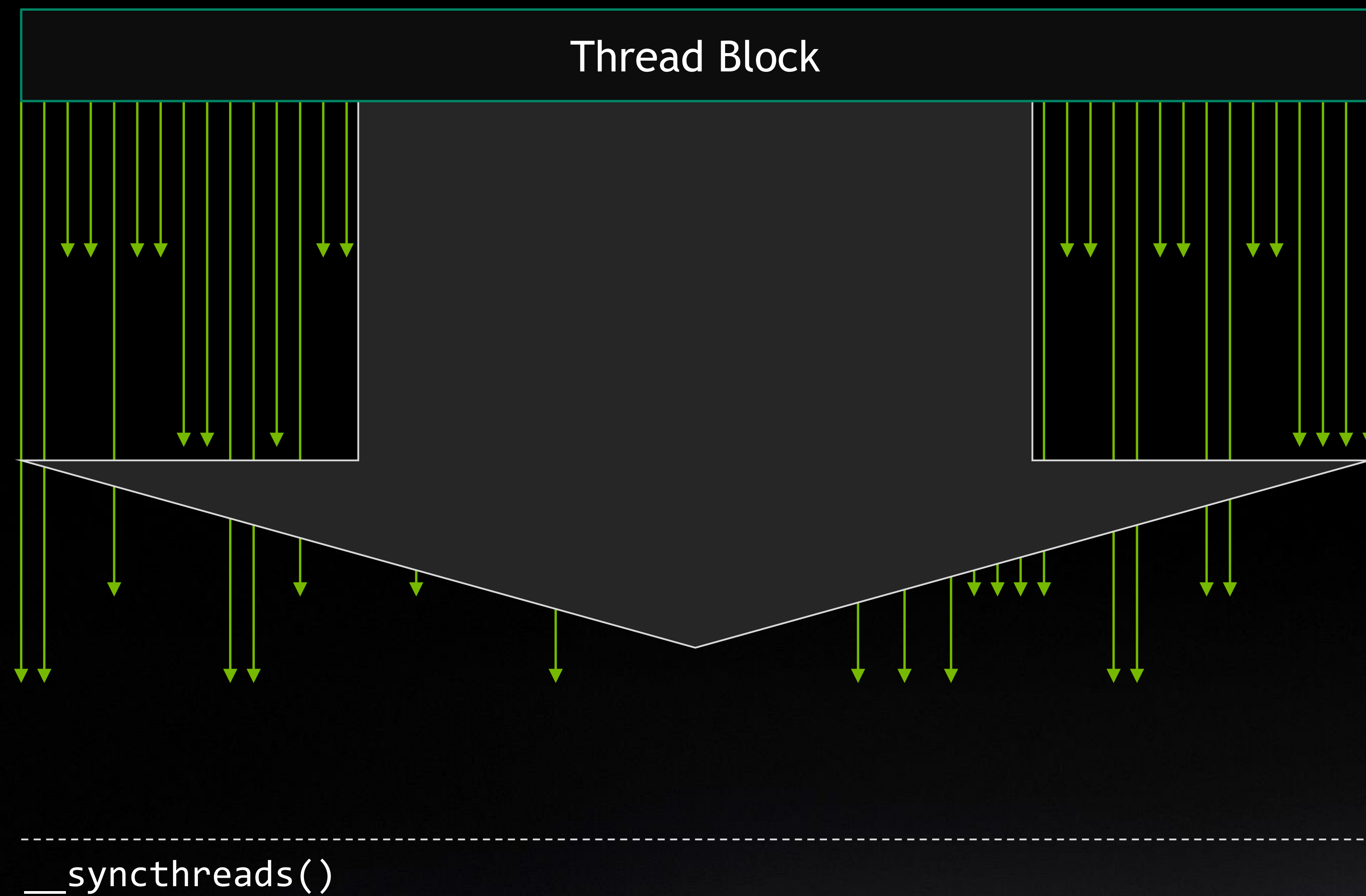
NVIDIA Ampere GPU Architecture Allows Creation Of Arbitrary Barriers

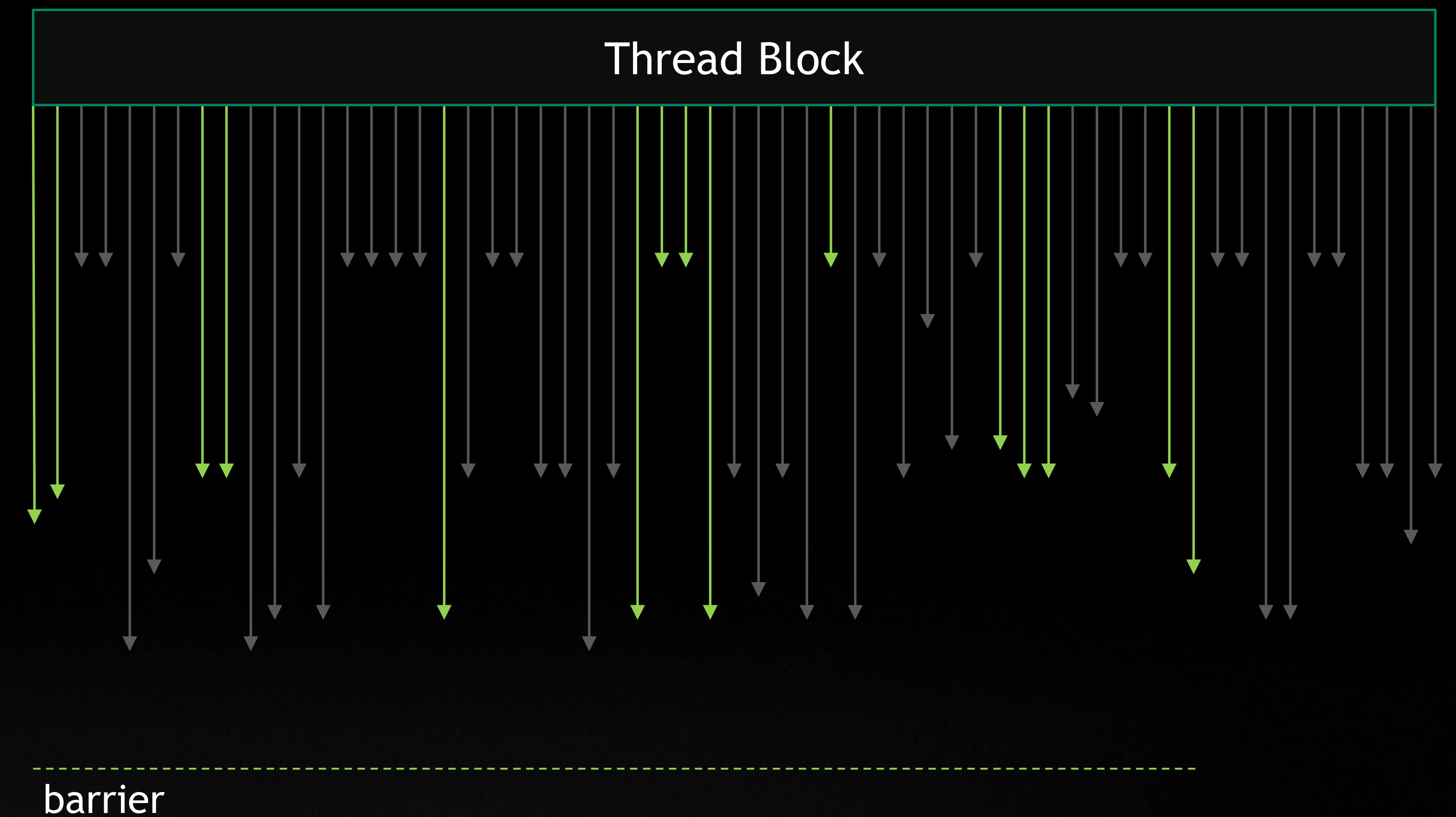


NVIDIA

FINE-GRAINED SYNCHRONIZATION

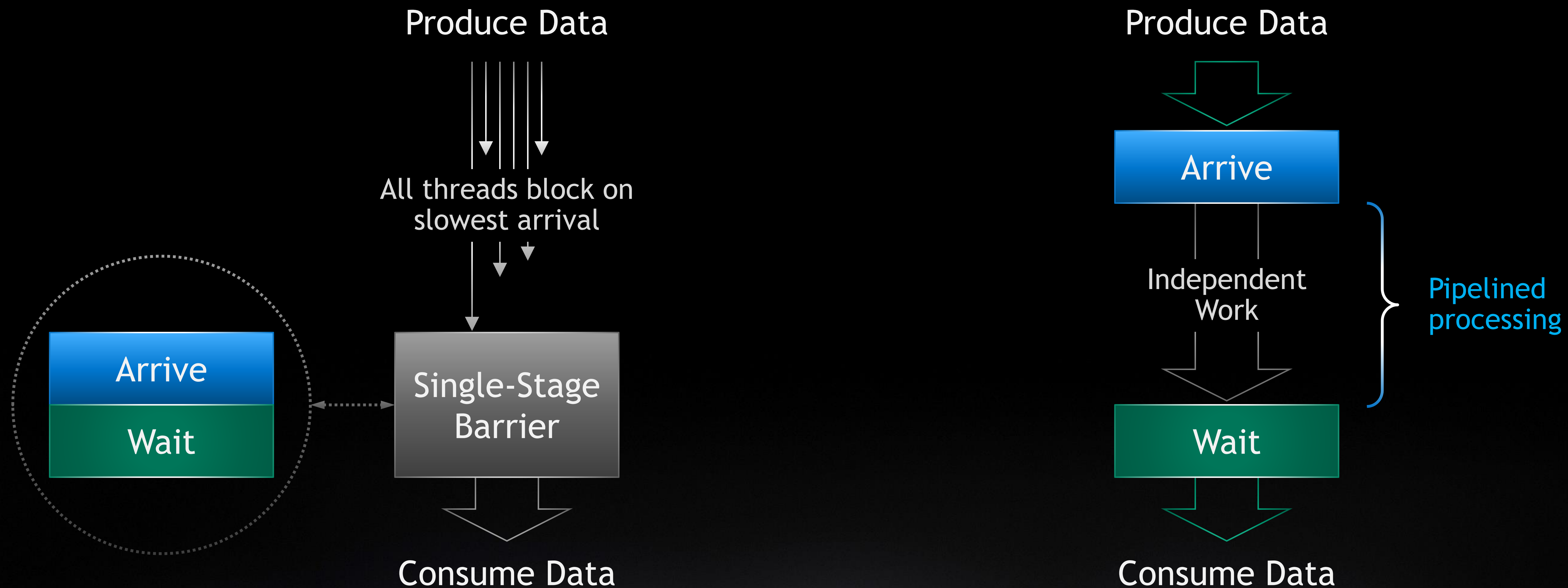
NVIDIA Ampere GPU Architecture Allows Creation Of Arbitrary Barriers





NVIDIA.

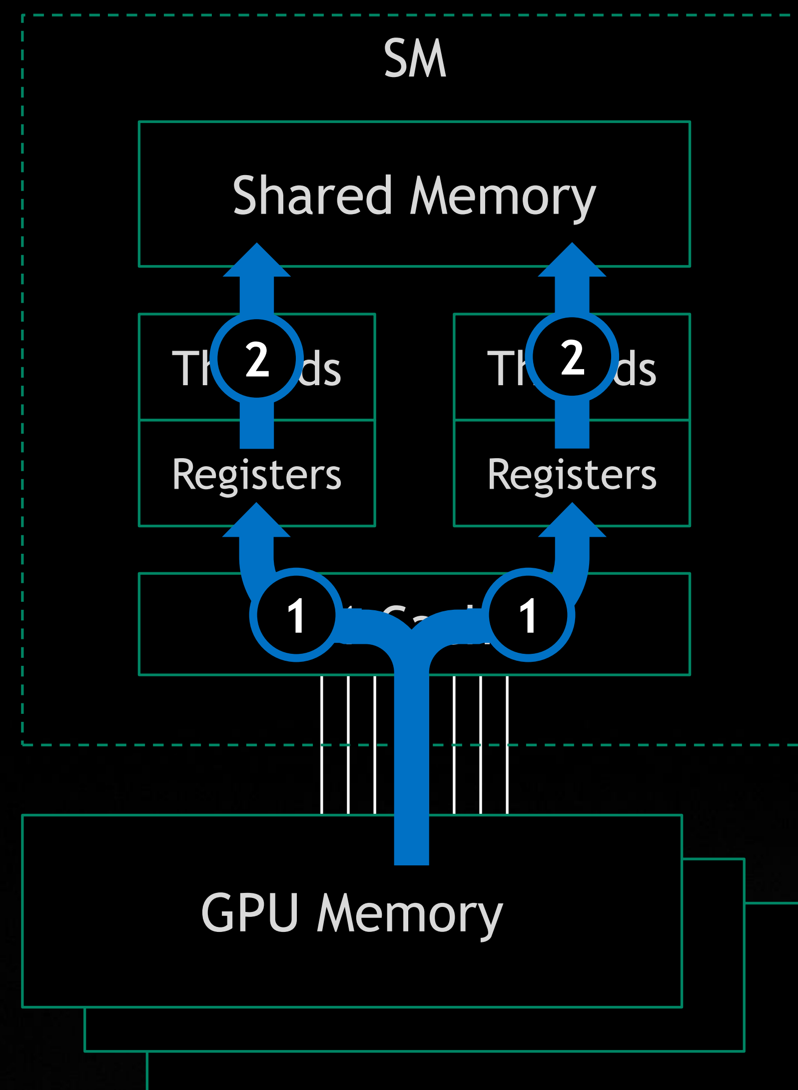
ASYNCHRONOUS BARRIERS

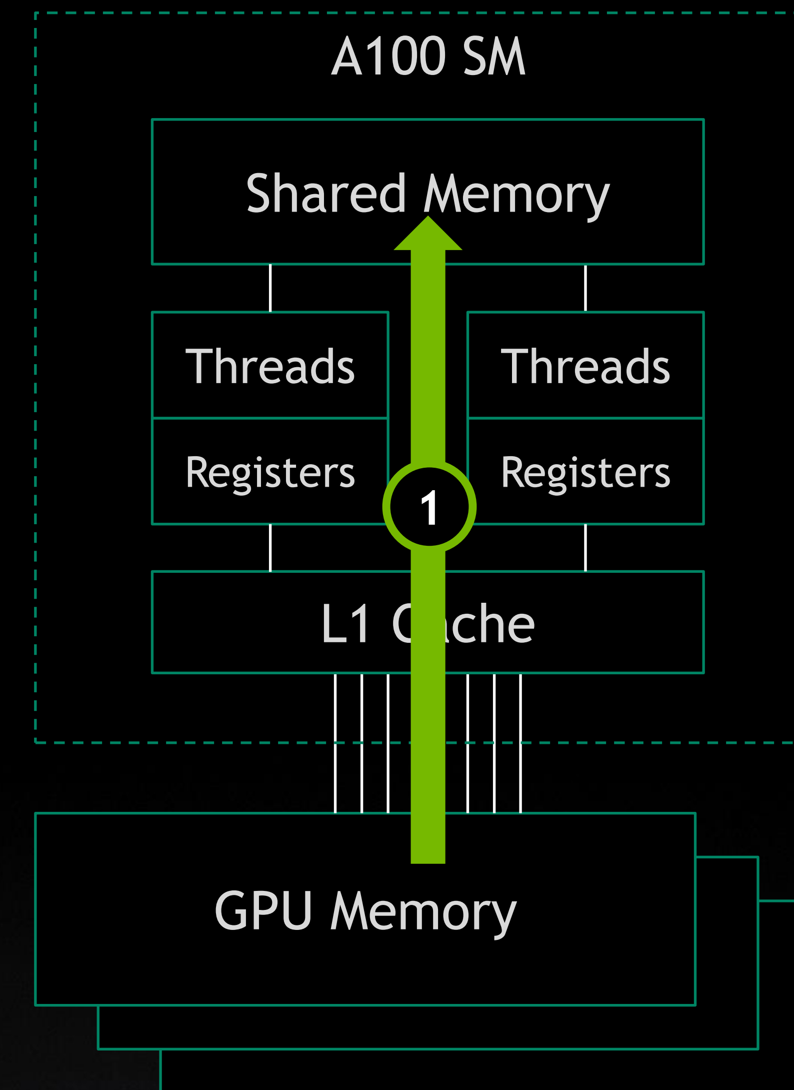


Single-Stage barriers combine back-to-back arrive & wait

Asynchronous barriers enable pipelined processing

ASYNC MEMCOPY: DIRECT TRANSFER INTO SHARED MEMORY





Two step copy to shared memory via registers

1. Thread loads data from GPU memory into registers
2. Thread stores data into SM shared memory

Asynchronous direct copy to shared memory

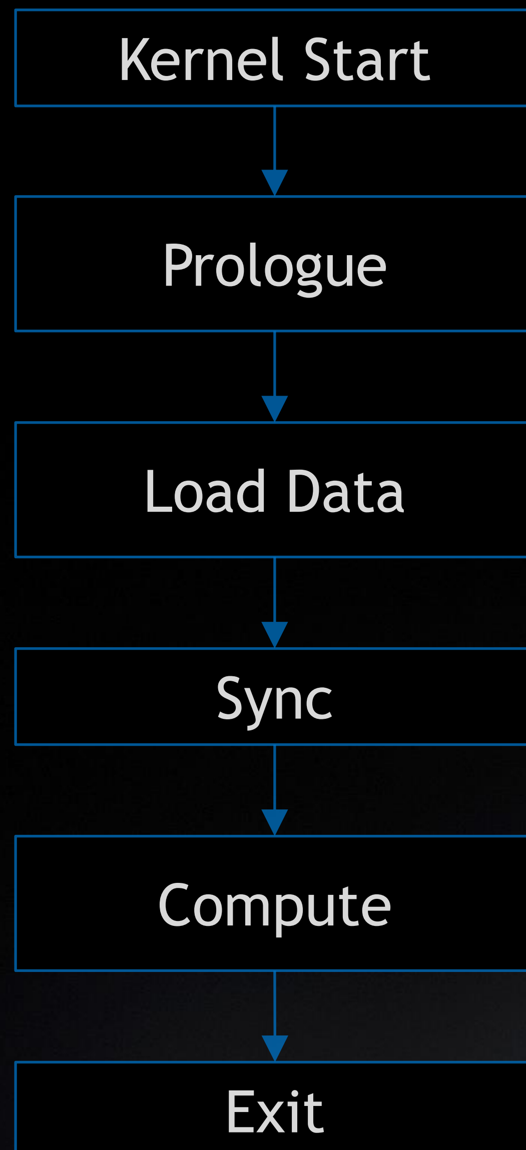
1. Direct transfer into shared memory, **bypassing** thread resources

NVIDIA.

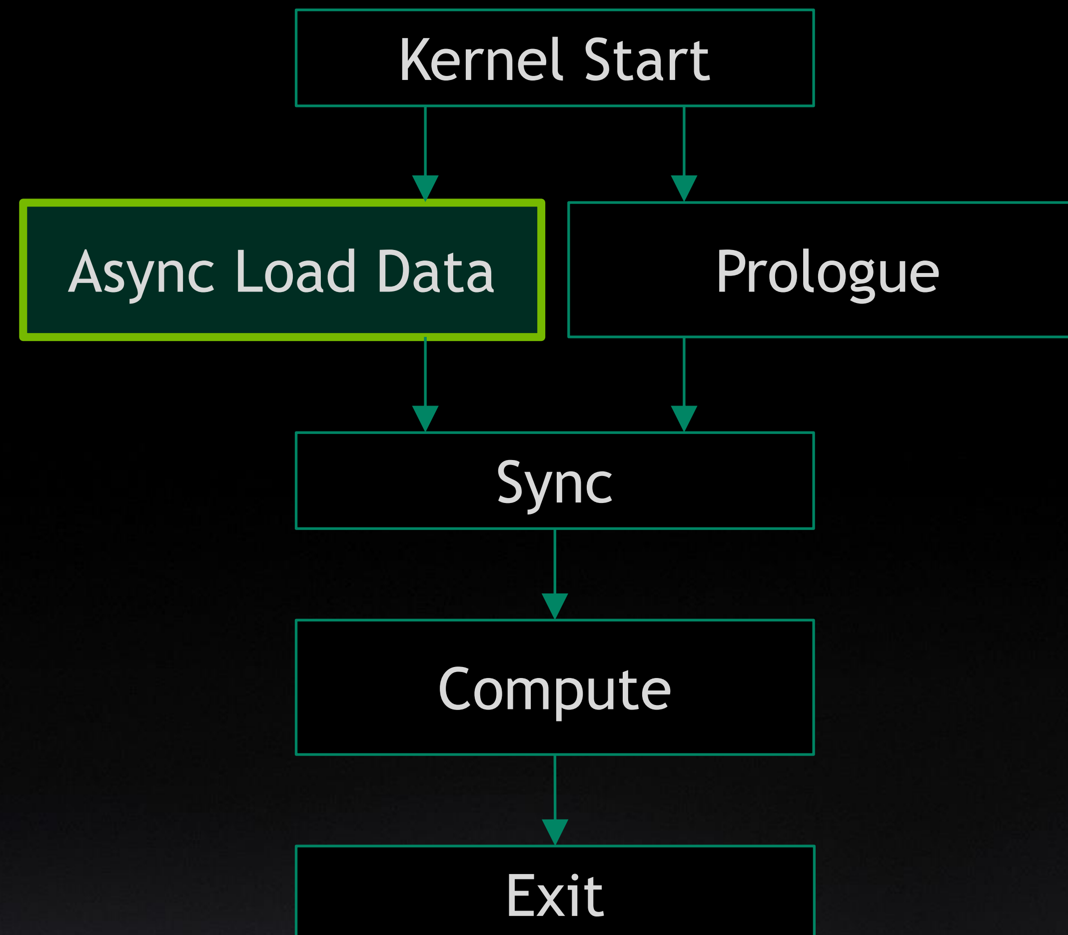
THIS FEATURE WILL CHANGE EVERY KERNEL THAT I WRITE

Free performance, fewer resources, cleaner code

All my programs look like this



Now they will look like this



Use **fewer resources** (reduced register pressure)

Increase **occupancy**

“Prologue” is now **free**

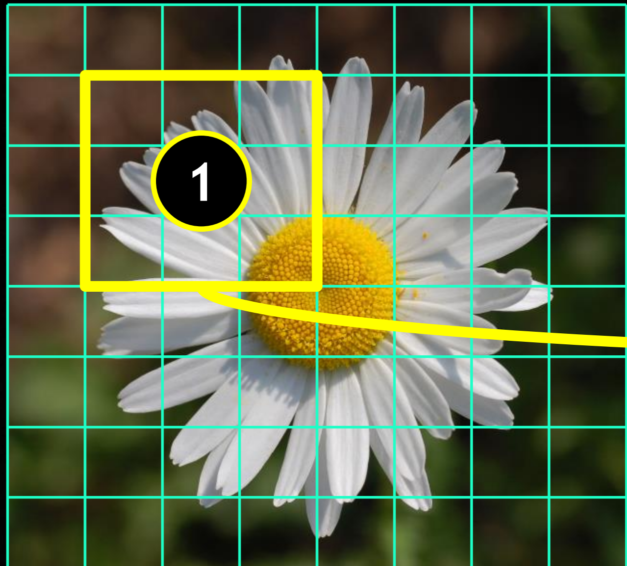
Enables **Pipelined** iteration with split barriers (see upcoming)

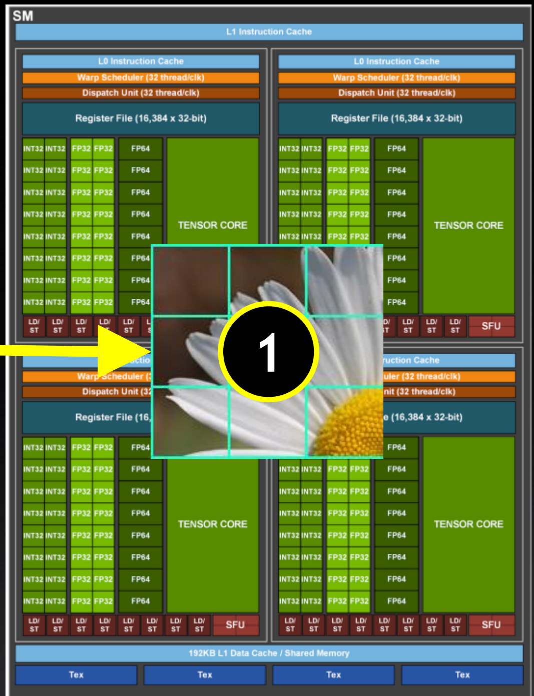
NVIDIA

SIMPLE DATA MOVEMENT

Shared Memory

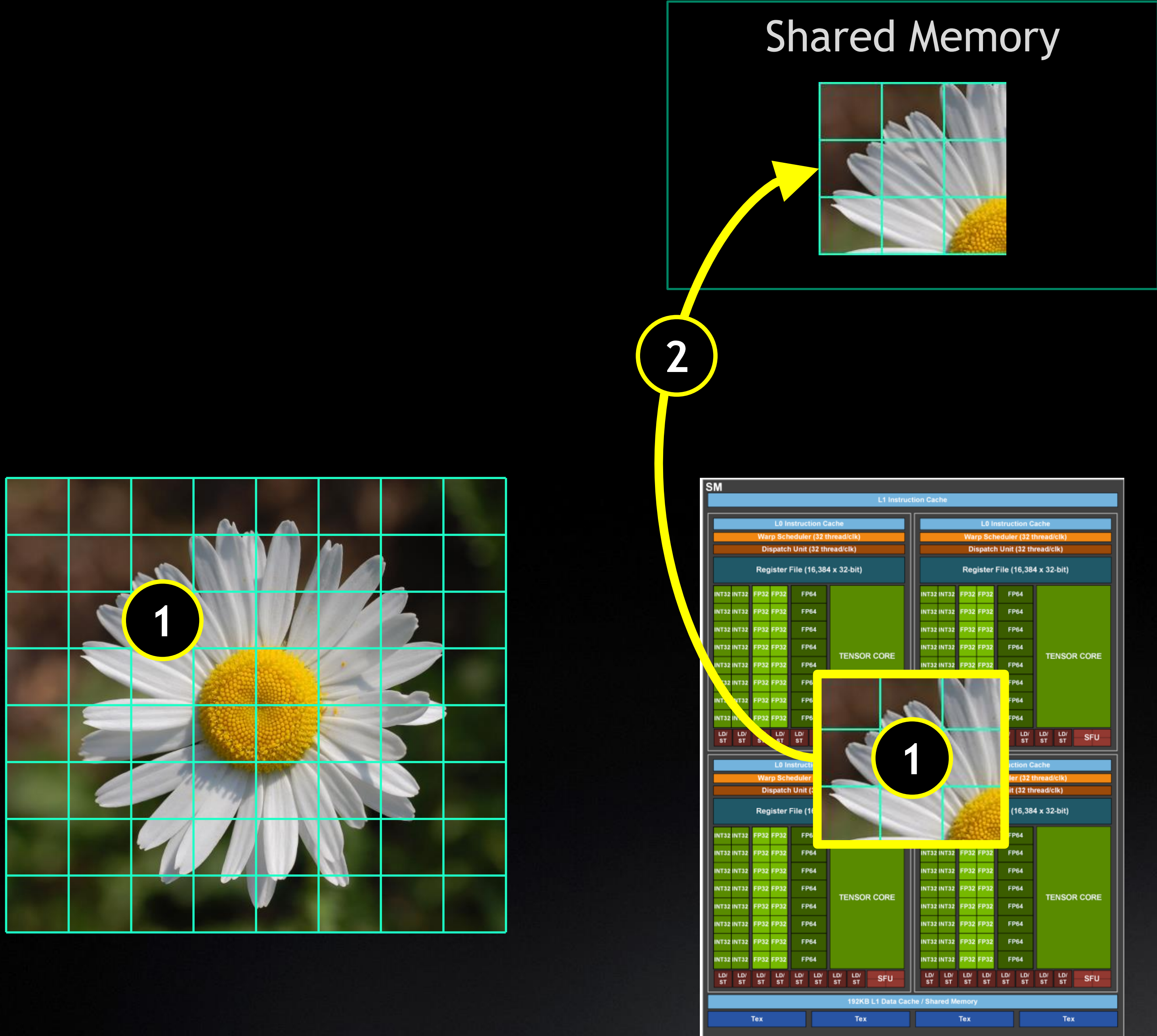
1 Load image element into registers

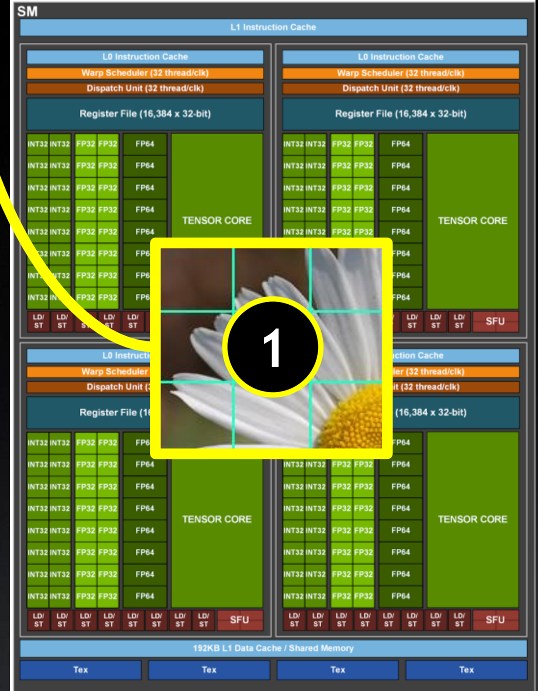




NVIDIA.

SIMPLE DATA MOVEMENT



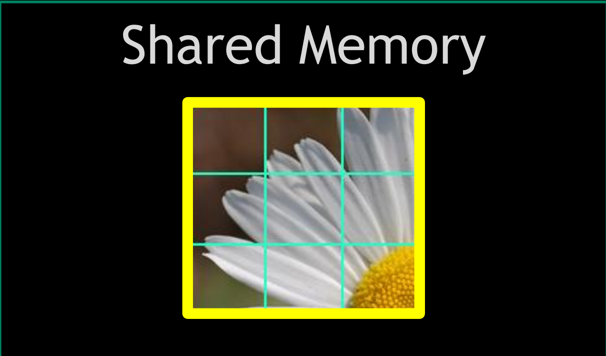


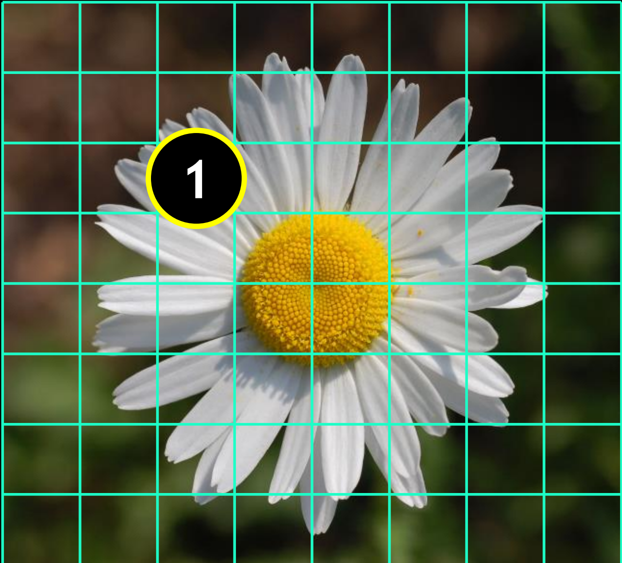
1. Load image element into registers
2. Store image element into shared memory

NVIDIA.

SIMPLE DATA MOVEMENT

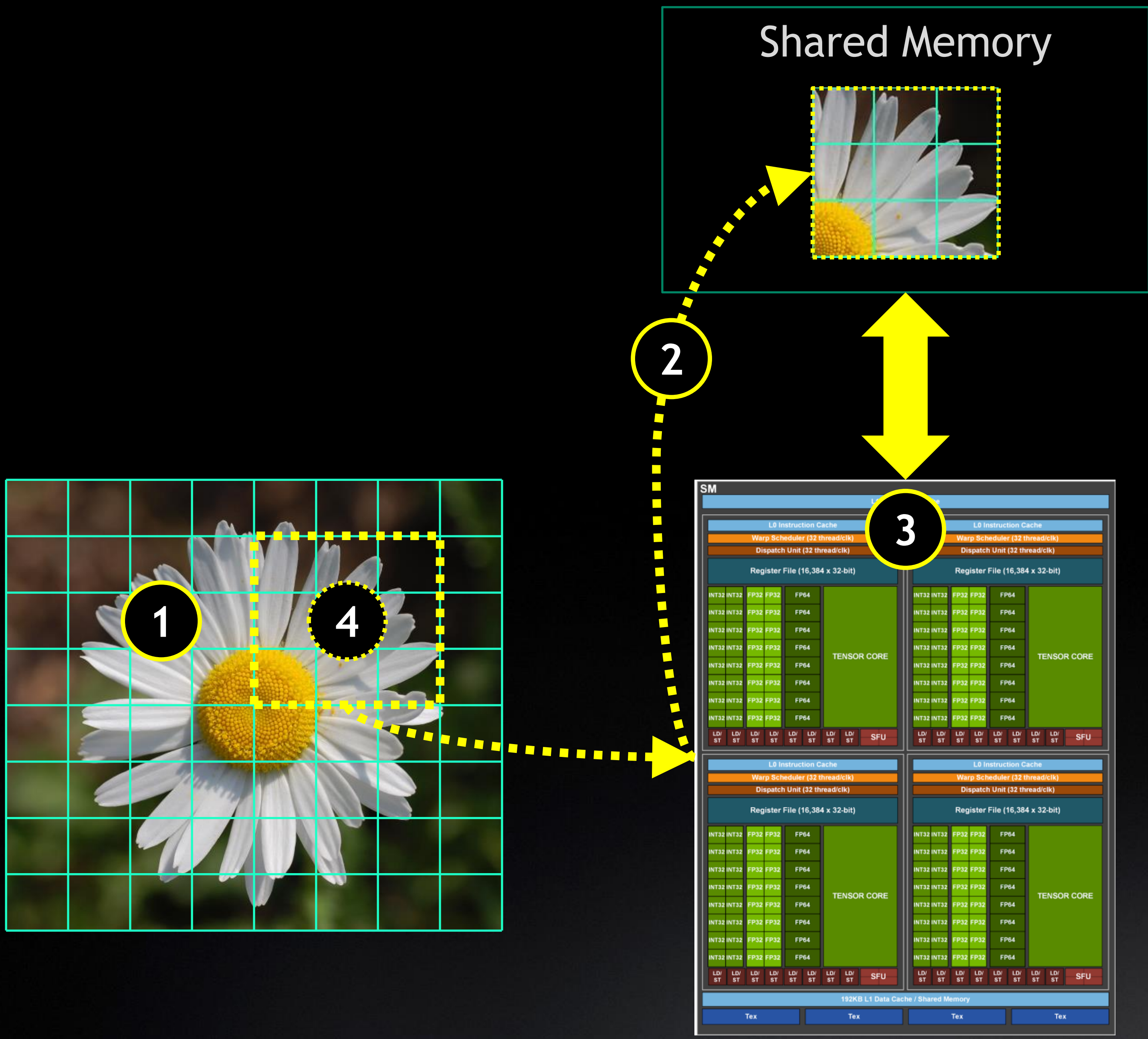
Shared Memory





1. Load image element into registers
2. Store image element into shared memory
3. Compute using shared memory data

SIMPLE DATA MOVEMENT



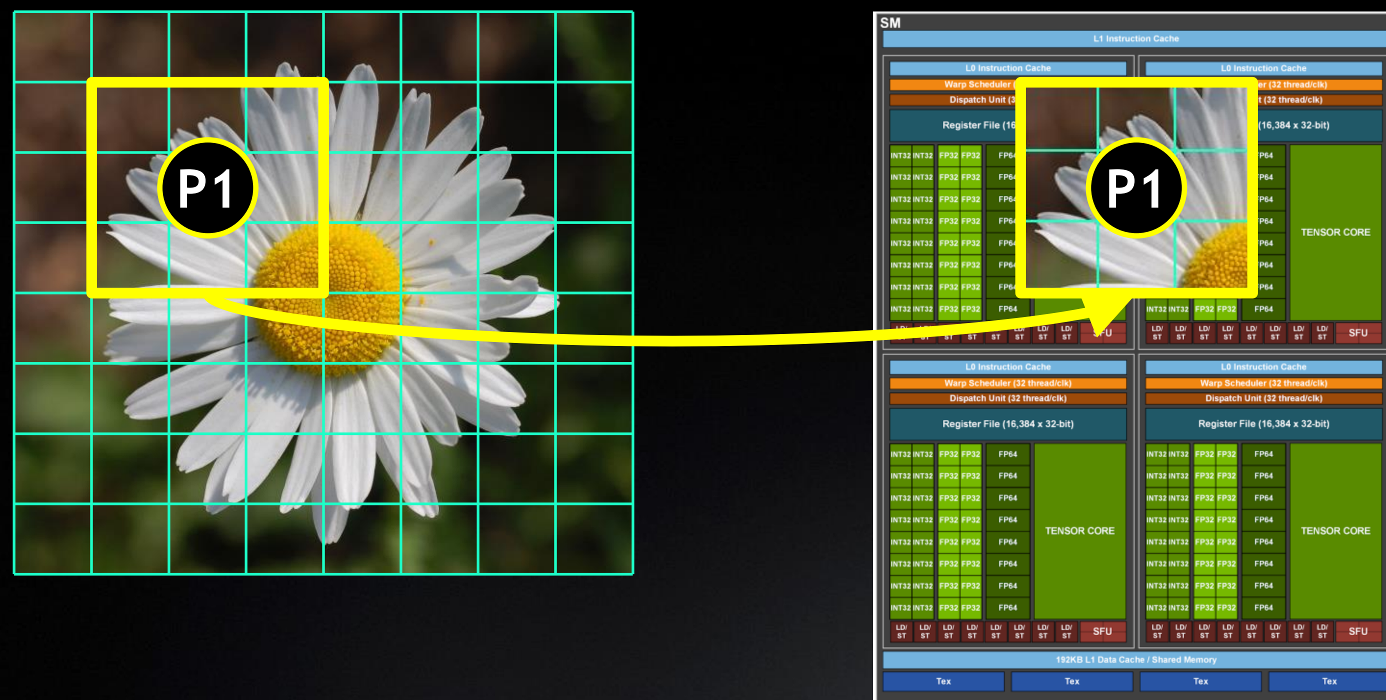
1. Load image element into registers
2. Store image element into shared memory
3. Compute using shared memory data
4. Repeat for next element

NVIDIA.

DOUBLE-BUFFERED DATA MOVEMENT

Shared Memory

P1 Prefetch initial image element into registers



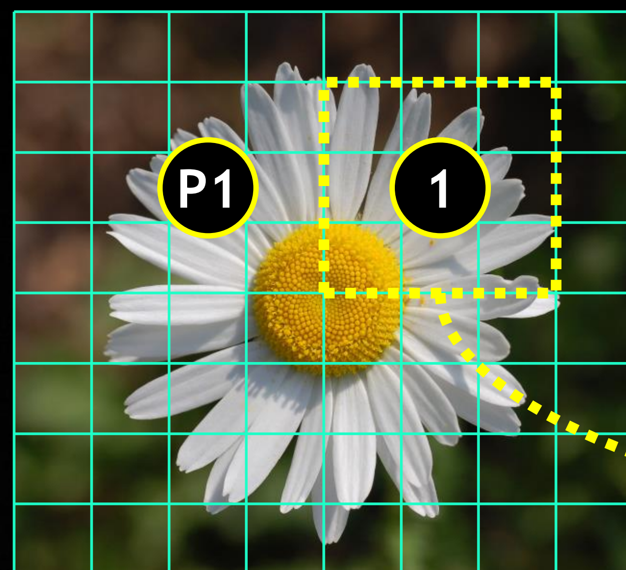
NVIDIA.

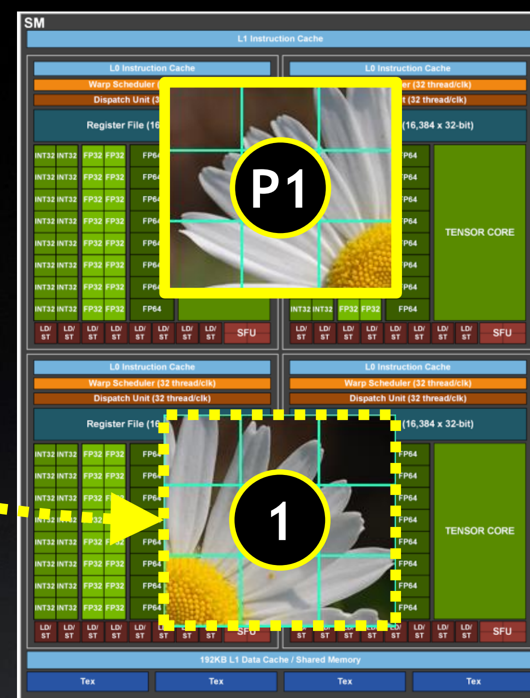
DOUBLE-BUFFERED DATA MOVEMENT

Shared Memory

P1 Prefetch initial image element into registers

1 Prefetch **next** element into **more** registers





NVIDIA.

DOUBLE-BUFFERED DATA MOVEMENT

Shared Memory

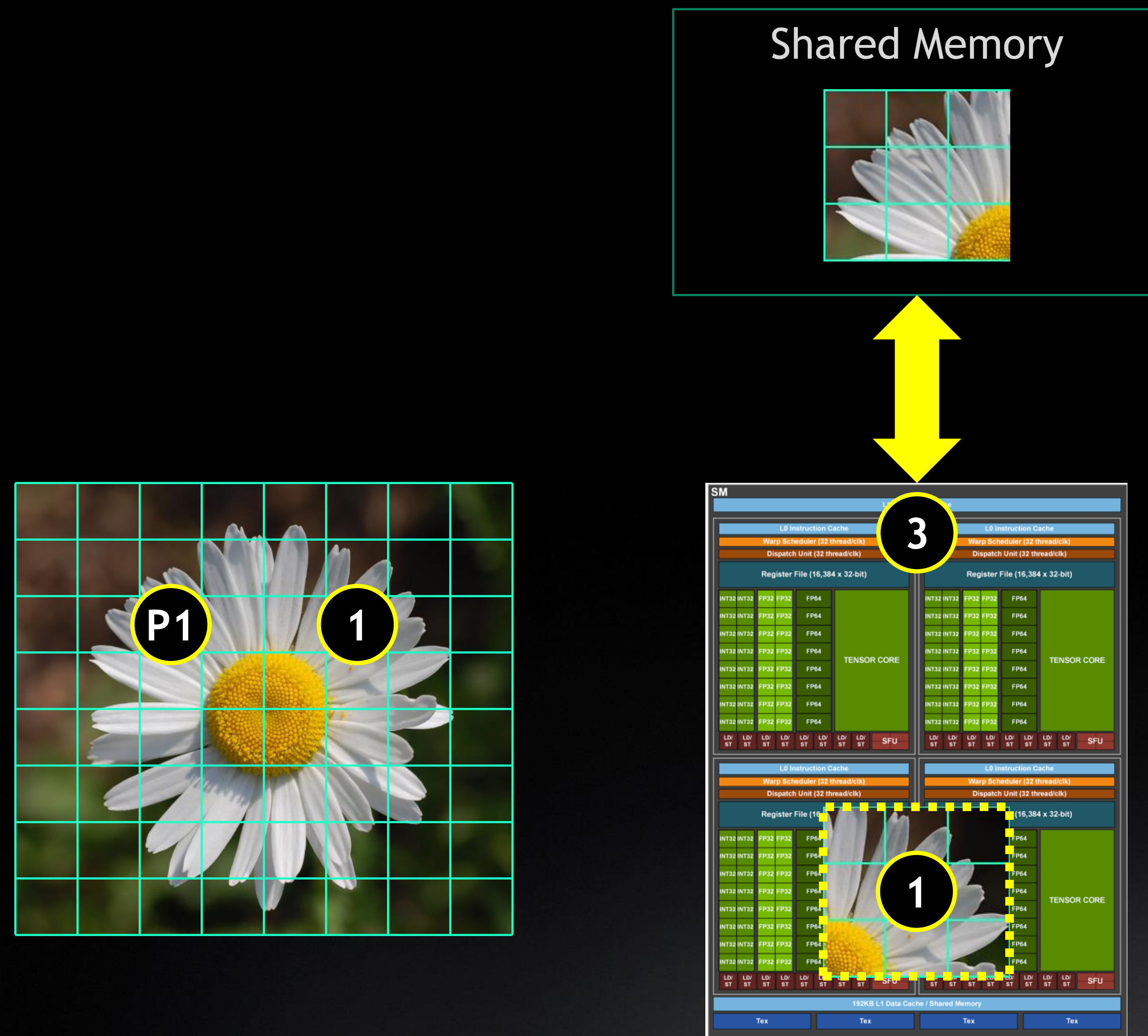
P1 Prefetch initial image element into registers

1 Prefetch **next** element into **more** registers

2 Store **current** element into shared memory

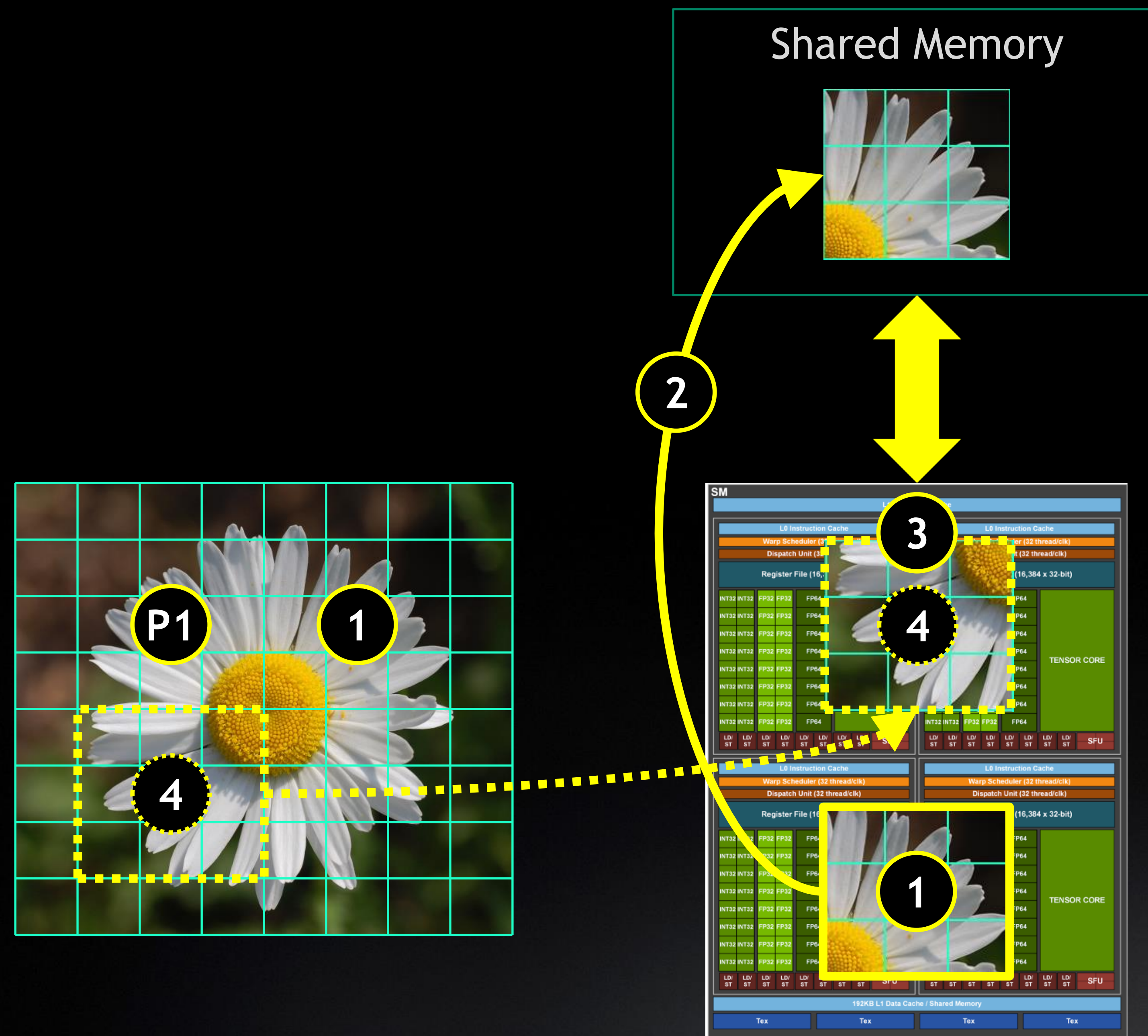
NVIDIA.

DOUBLE-BUFFERED DATA MOVEMENT



- (P1) Prefetch initial image element into registers
- (1) Prefetch **next** element into **more** registers
- (2) Store **current** element into shared memory
- (3) Compute using shared memory data

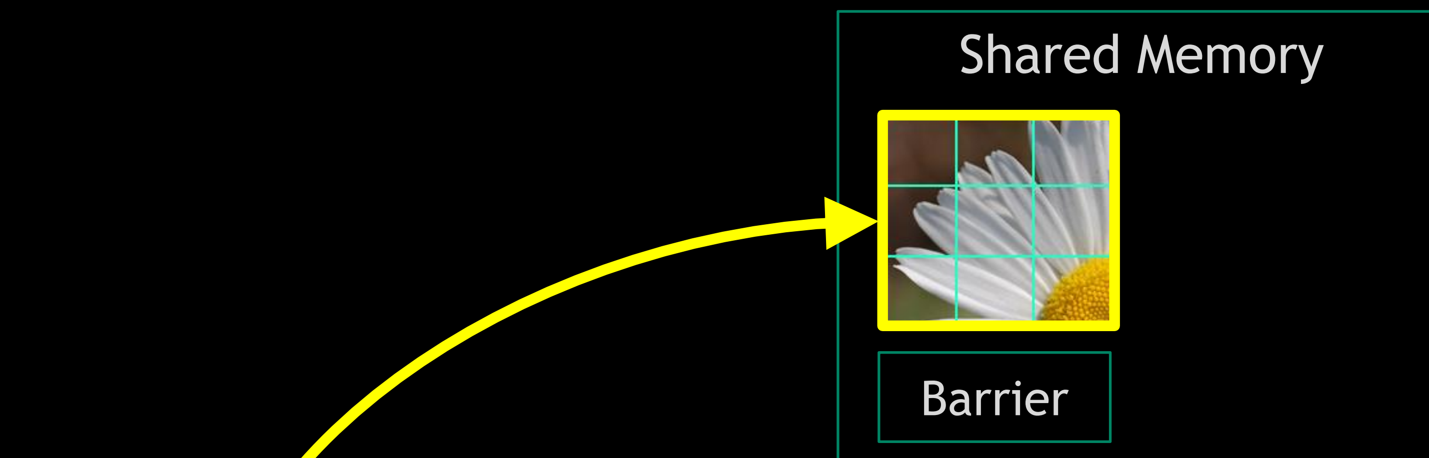
DOUBLE-BUFFERED DATA MOVEMENT

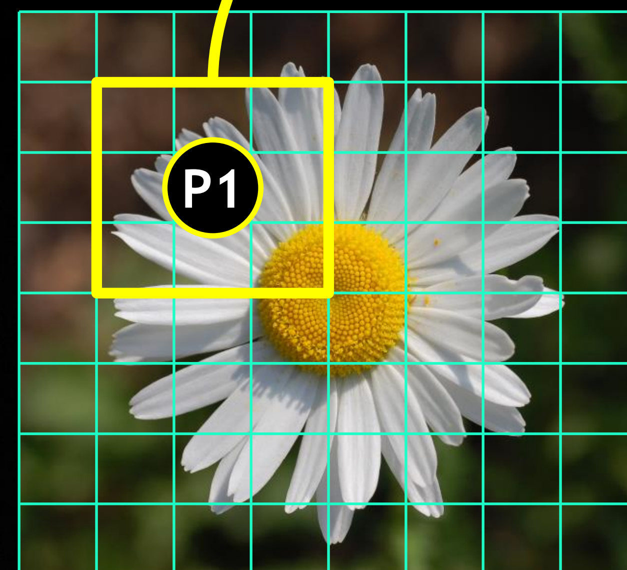


- (P1) Prefetch initial image element into registers
- (1) Prefetch **next** element into **more** registers
- (2) Store **current** element into shared memory
- (3) Compute using shared memory data
- (4) Repeat for next element

NVIDIA.

ASYNCHRONOUS DIRECT DATA MOVEMENT

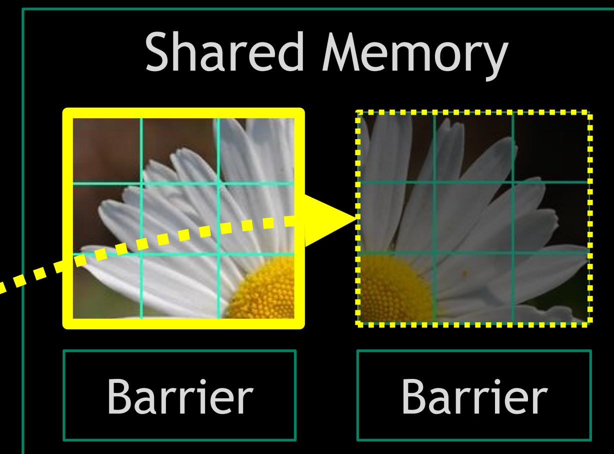




P1 **Async copy** initial element into shared memory

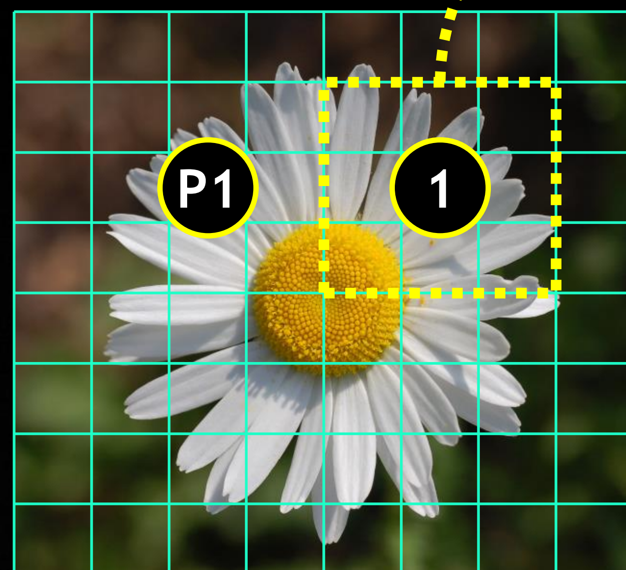
ASYNCHRONOUS DIRECT DATA MOVEMENT

Shared Memory



Barrier

Barrier

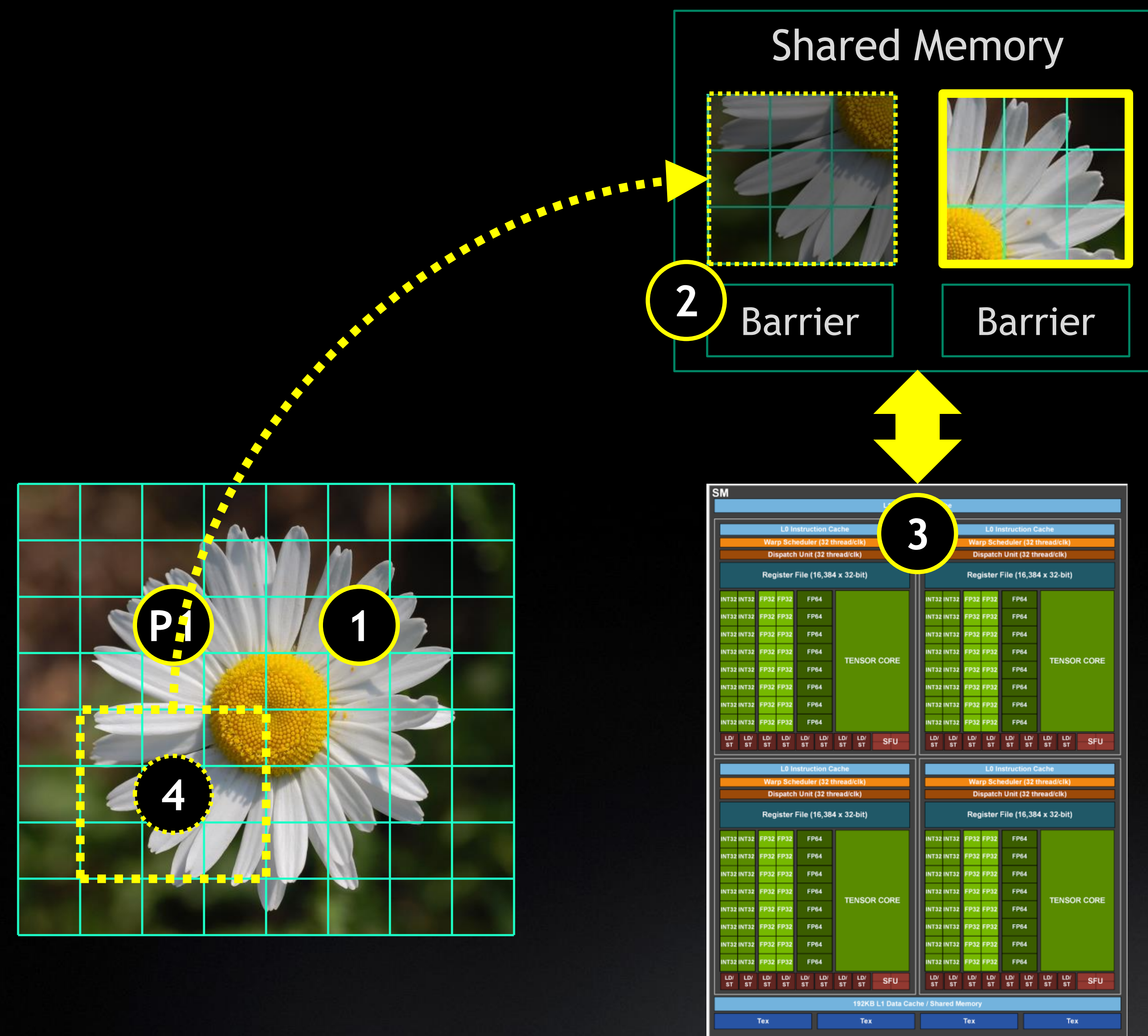


P1 **Async copy** initial element into shared memory

1 **Async copy** **next** element into shared memory

NVIDIA

ASYNCHRONOUS DIRECT DATA MOVEMENT

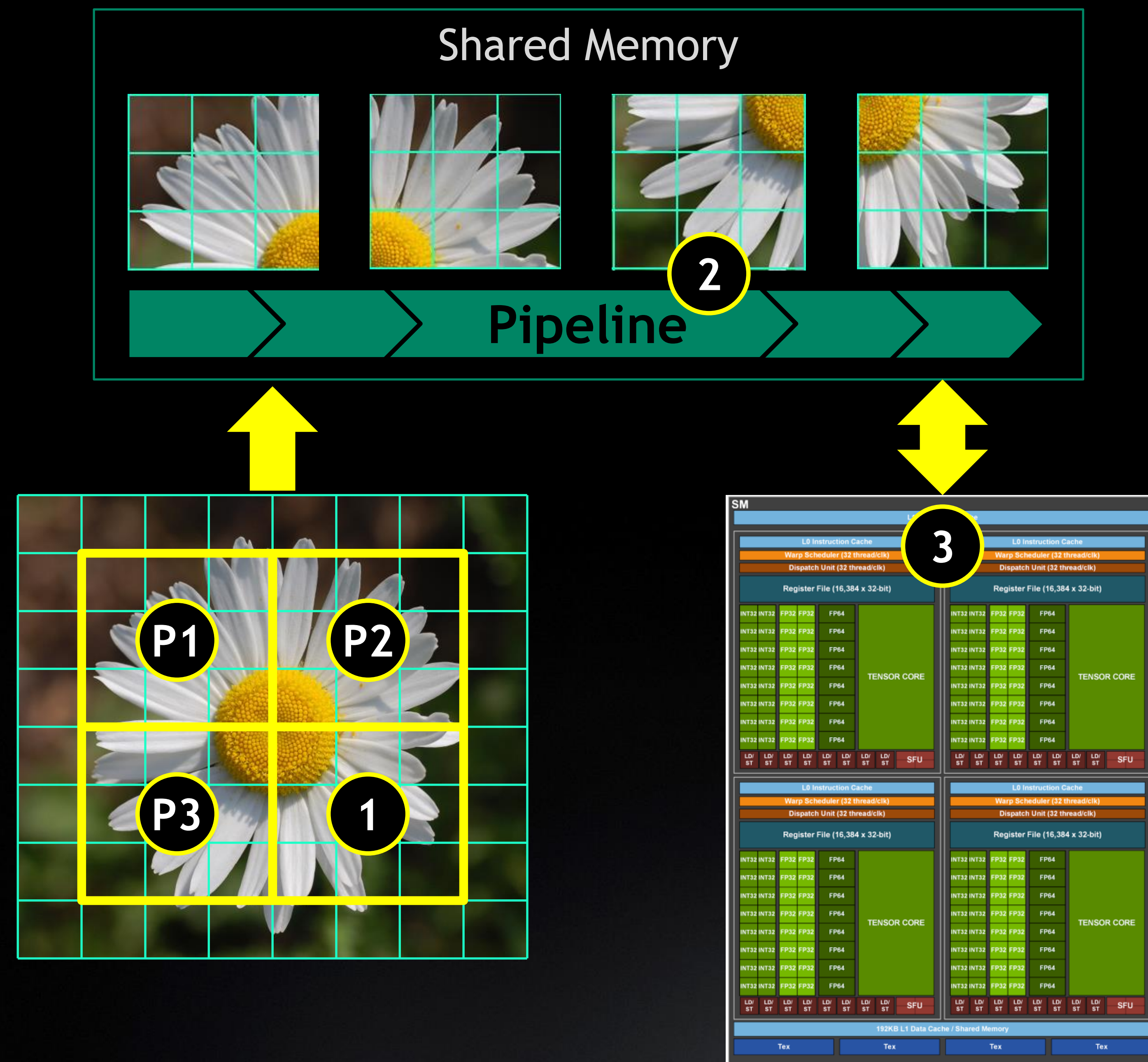


- **(P1)** Async copy initial element into shared memory
- **(1)** Async copy **next** element into shared memory
- **(2)** Threads synchronize with **current** async copy
- **(3)** Compute using shared memory data
- **(4)** Repeat for next element

NVIDIA

ASYNCHRONOUS COPY PIPELINES

Prefetch multiple images in a continuous stream



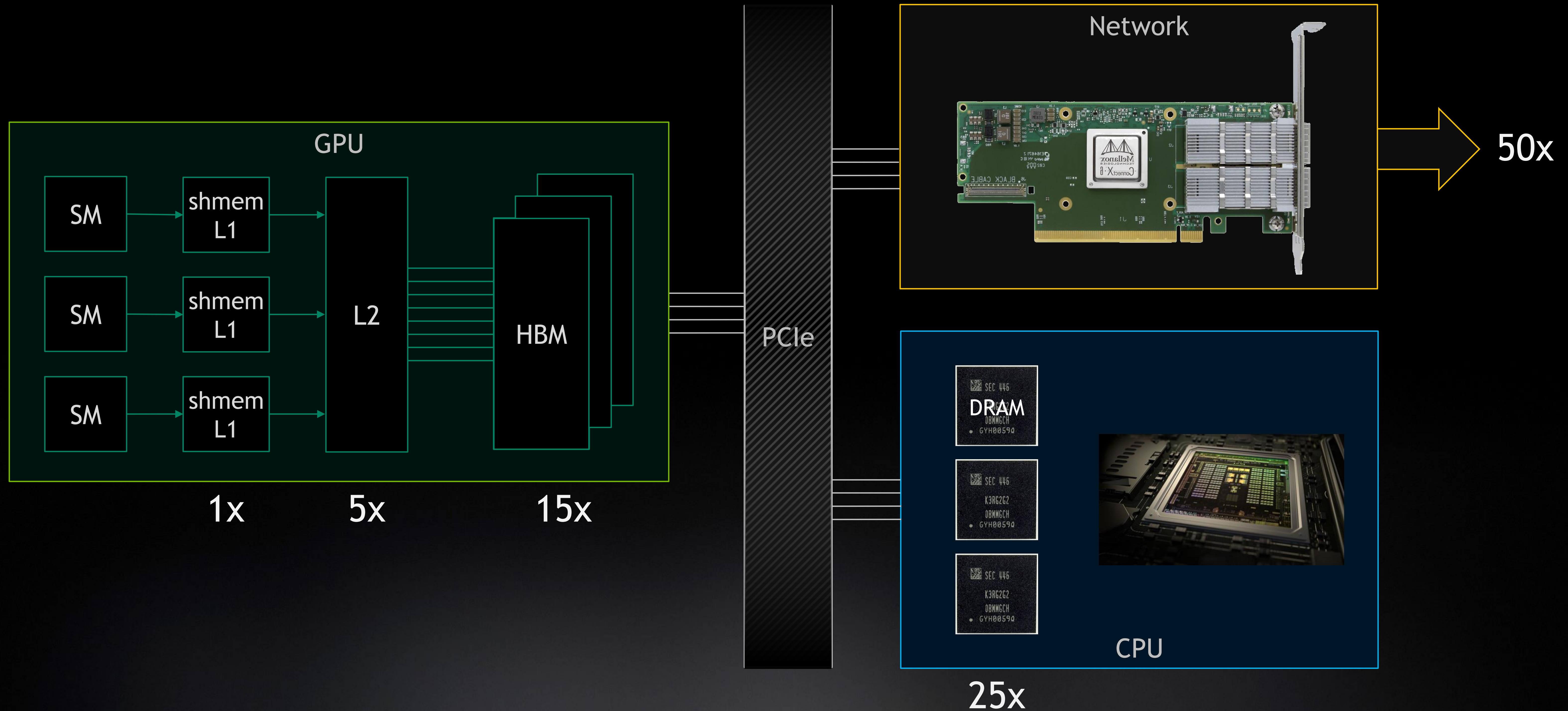
P1 P2 P3 **Async copy** multiple elements into shared memory

1. **Async copy next** element into shared memory
2. Threads **synchronize** with **oldest** pipelined copy
3. Compute using shared memory data
4. Repeat for next element

For more information see: S21170 - CUDA on NVIDIA GPU Ampere Architecture, Taking your algorithms to the next level of performance

NVIDIA

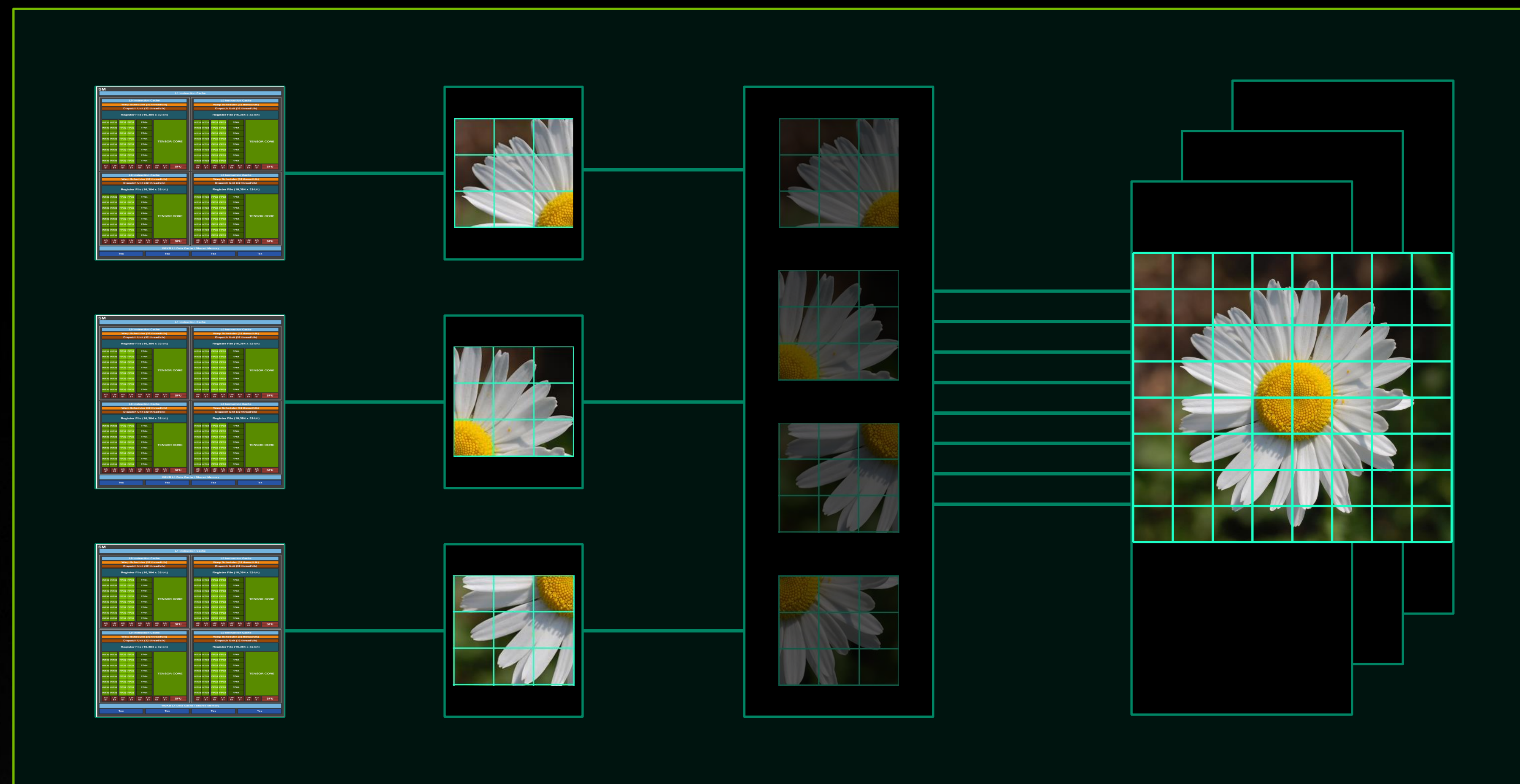
HIERARCHY OF LATENCIES



NVIDIA

MANAGING LATENCY: L2 CACHE RESIDENCY CONTROL

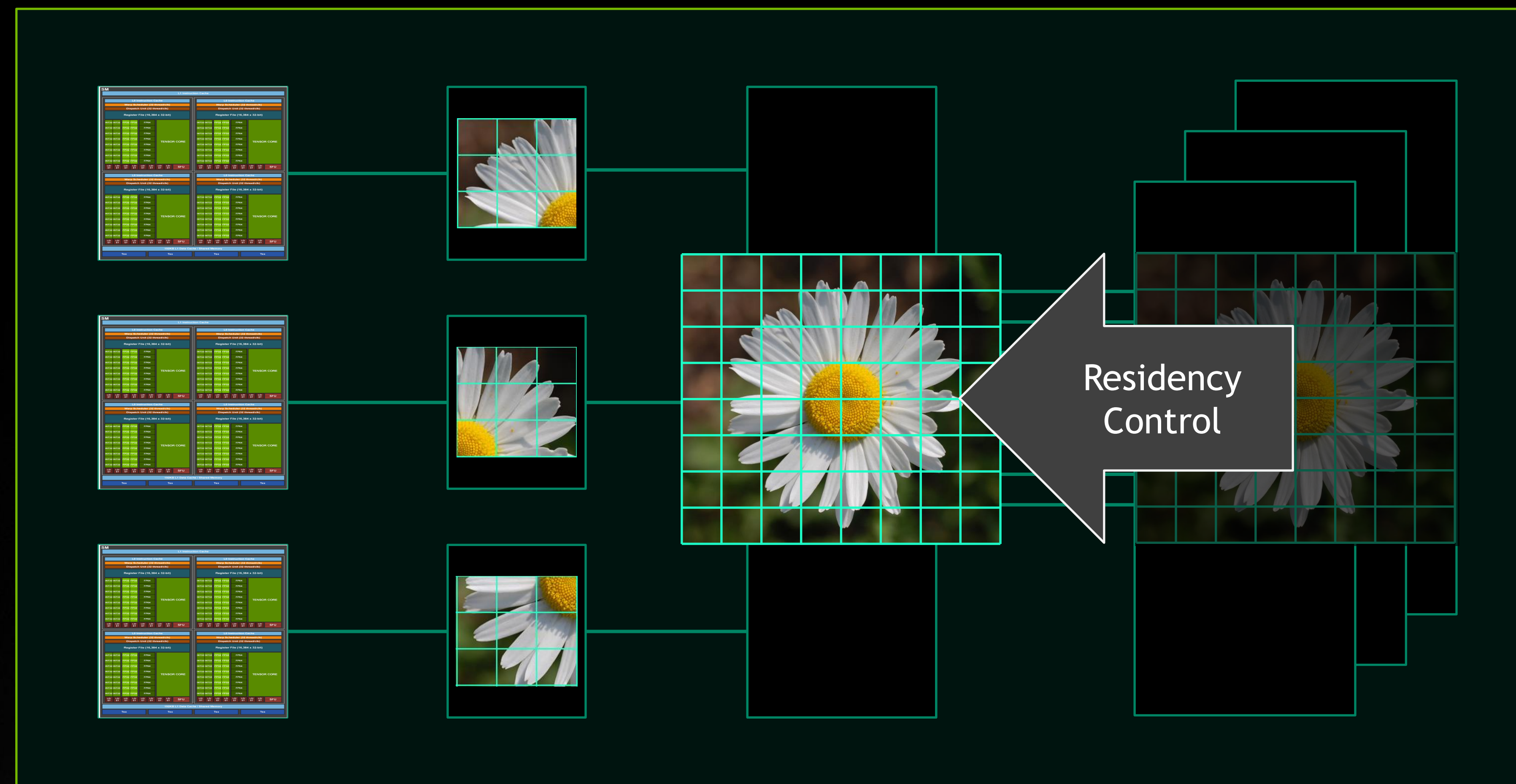
	Shared Memory	L2 Cache	GPU Memory
Latency	1x	5x	15x
Bandwidth	13x	3x	1x



NVIDIA

MANAGING LATENCY: L2 CACHE RESIDENCY CONTROL

	Shared Memory	L2 Cache	GPU Memory
Latency	1x	5x	15x
Bandwidth	13x	3x	1x



L2 Cache Residency Control

Specify address range up to 128MB for persistent caching

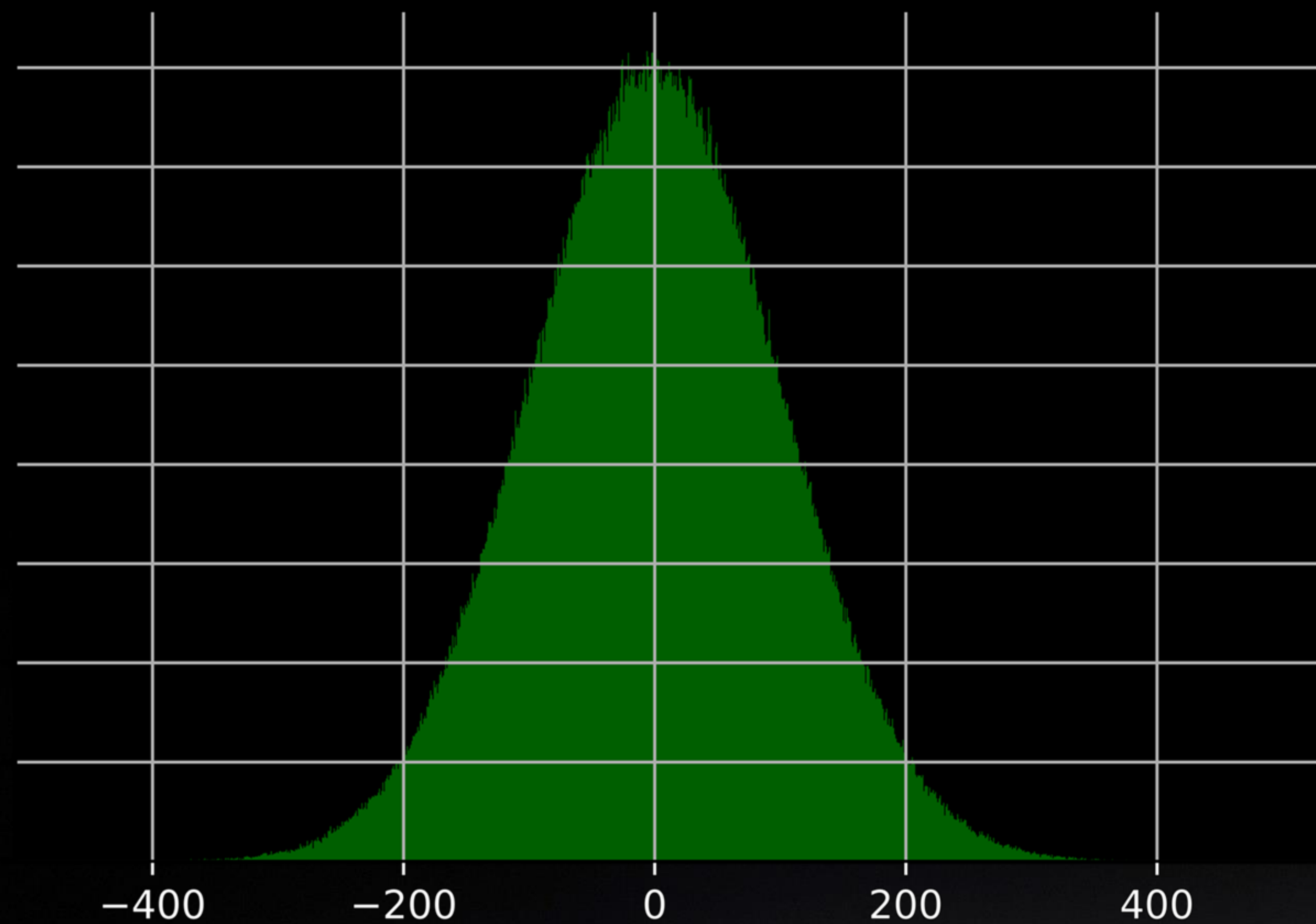
Normal & streaming accesses cannot evict persistent data

Load/store from range persists in L2 even between kernel launches

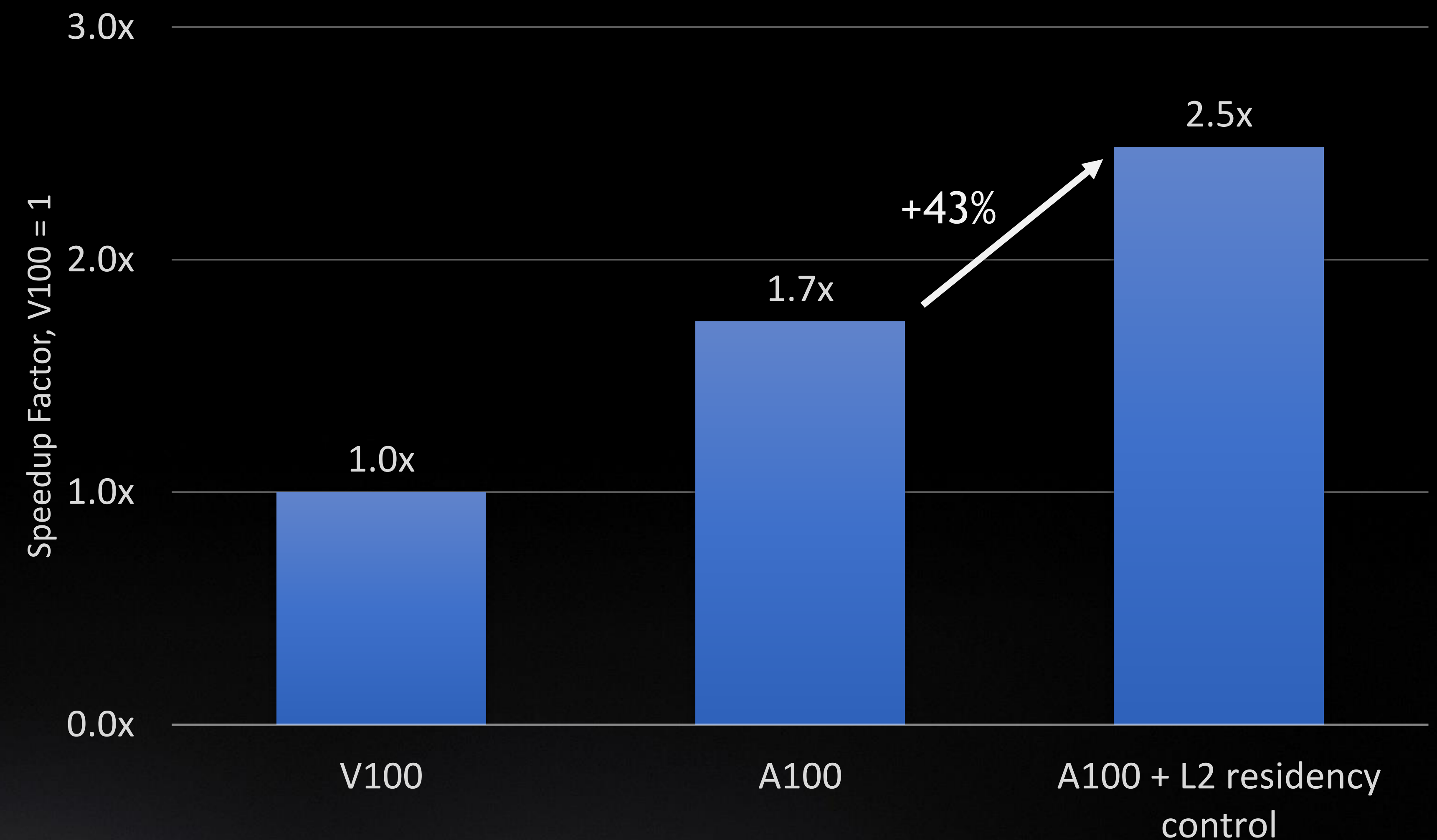
Normal accesses can still use entire cache if no persistent data is present

NVIDIA.

MANAGING LATENCY: L2 CACHE RESIDENCY CONTROL

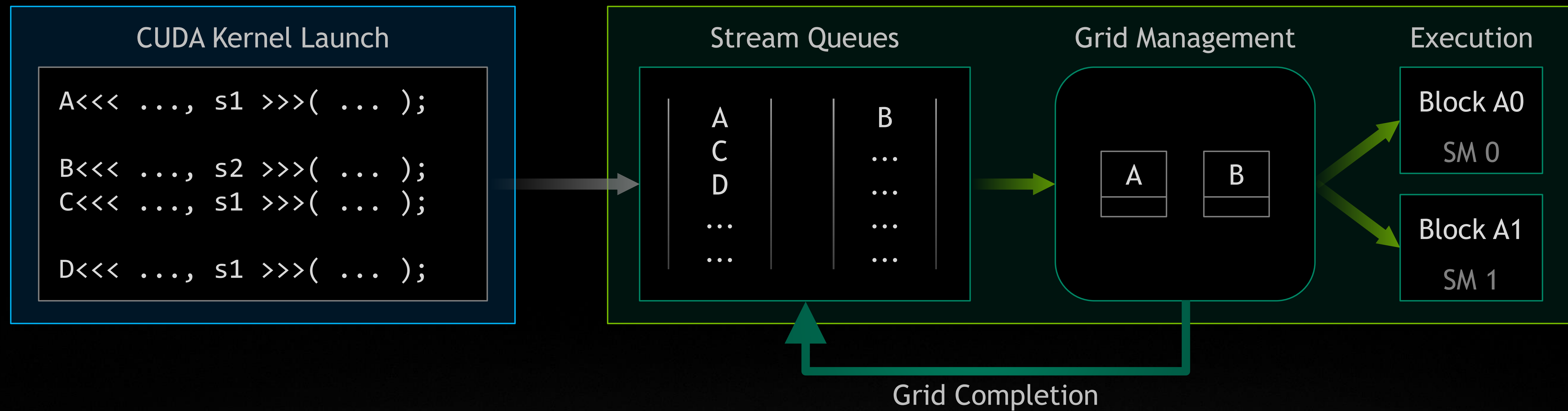


256 million items counted into 5 million histogram bins

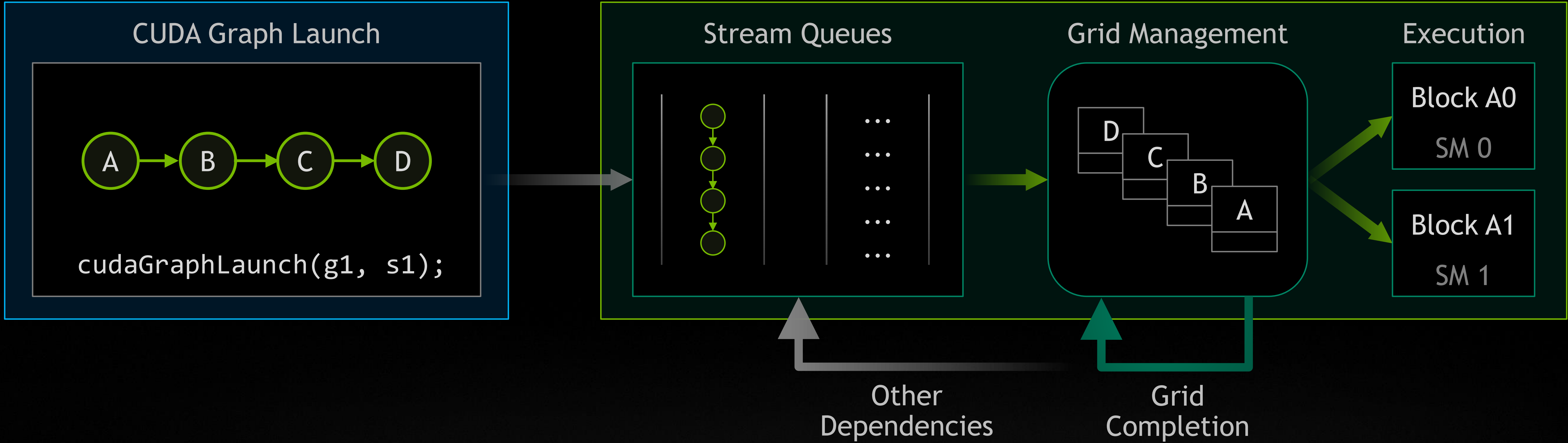


For more information see: S21819 - Optimizing Applications for NVIDIA Ampere GPU Architecture

ANATOMY OF A KERNEL LAUNCH



ANATOMY OF A GRAPH LAUNCH

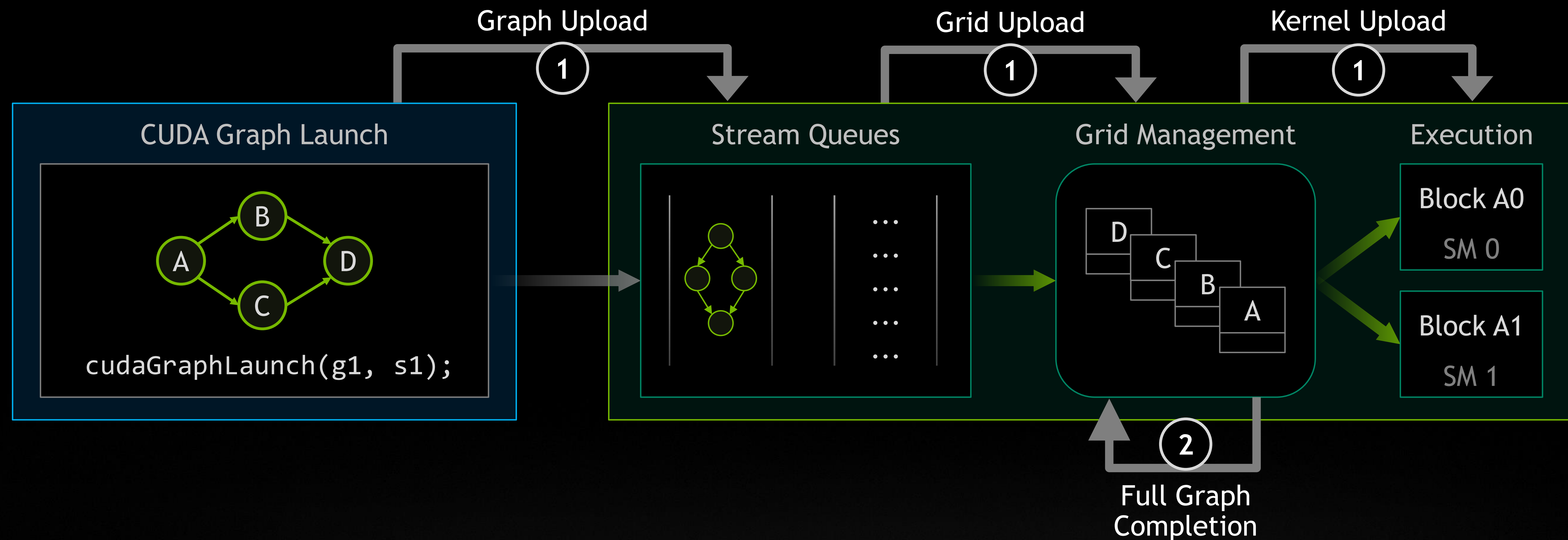


Graph allows launch of multiple kernels in a **single operation**

Graph pushes multiple grids to Grid Management Unit allowing **low-latency dependency resolution**

NVIDIA.

A100 ACCELERATES GRAPH LAUNCH & EXECUTION



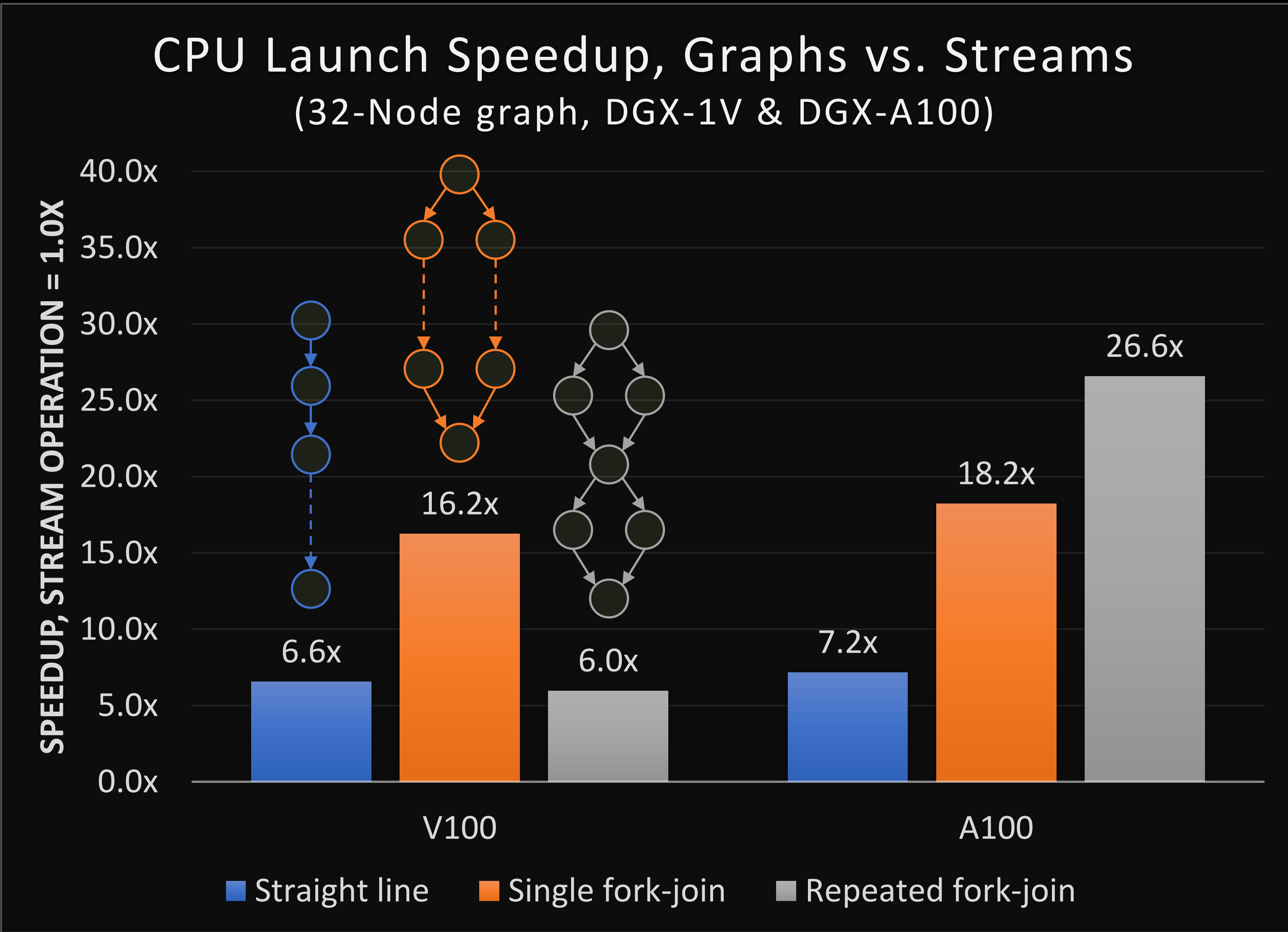
New A100 Execution Optimizations for Task Graphs

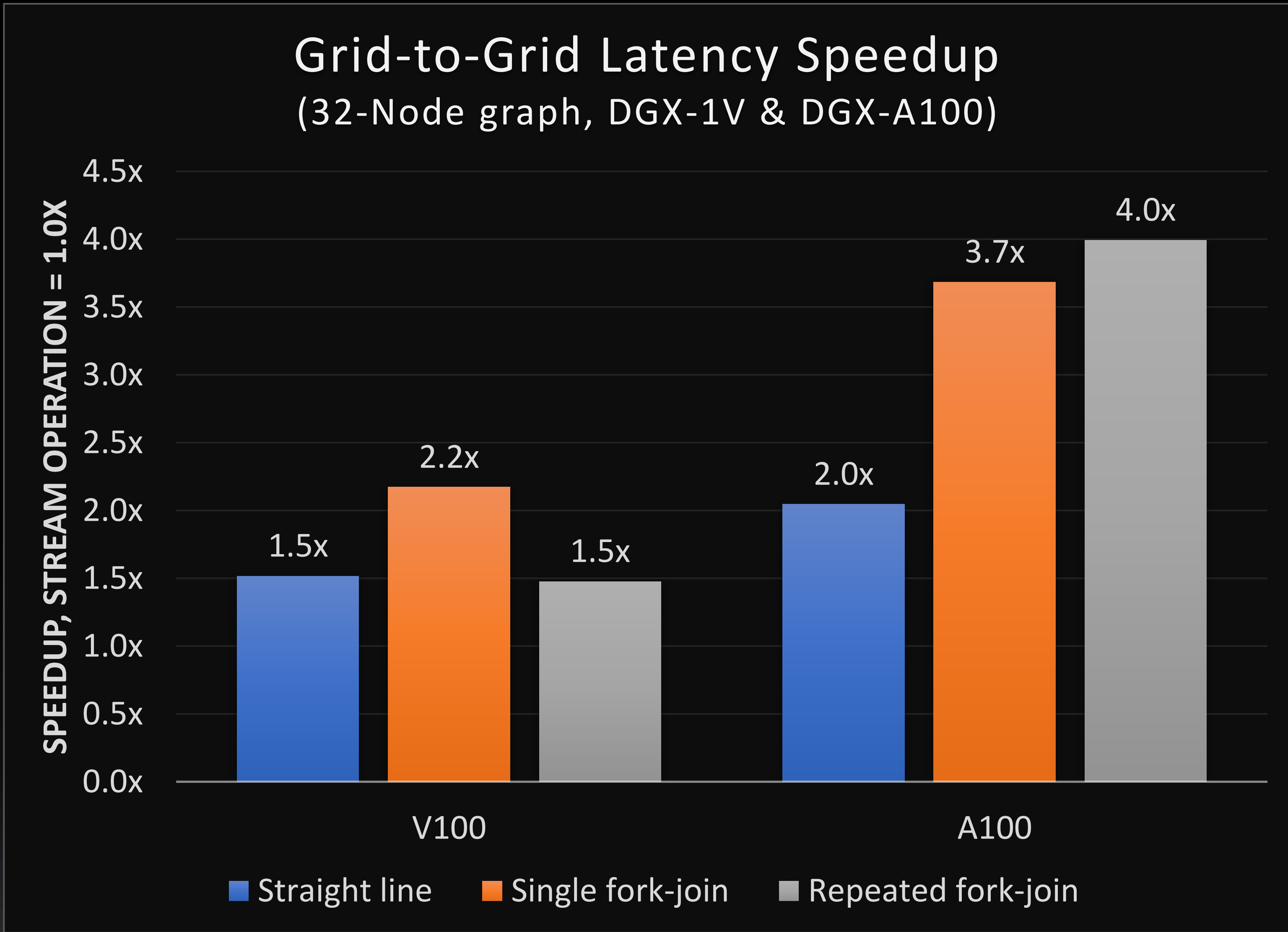
1. Grid launch latency reduction via whole-graph upload of grid & kernel data
2. Overhead reduction via accelerated dependency resolution

NVIDIA

LATENCIES & OVERHEADS: GRAPHS vs. STREAMS

Empty Kernel Launches – Investigating System Overheads





Note: Empty kernel launches – timings show reduction in latency only

NVIDIA.

GRAPH PARAMETER UPDATE

Fast Parameter Update When Topology Does Not Change

Graph Update

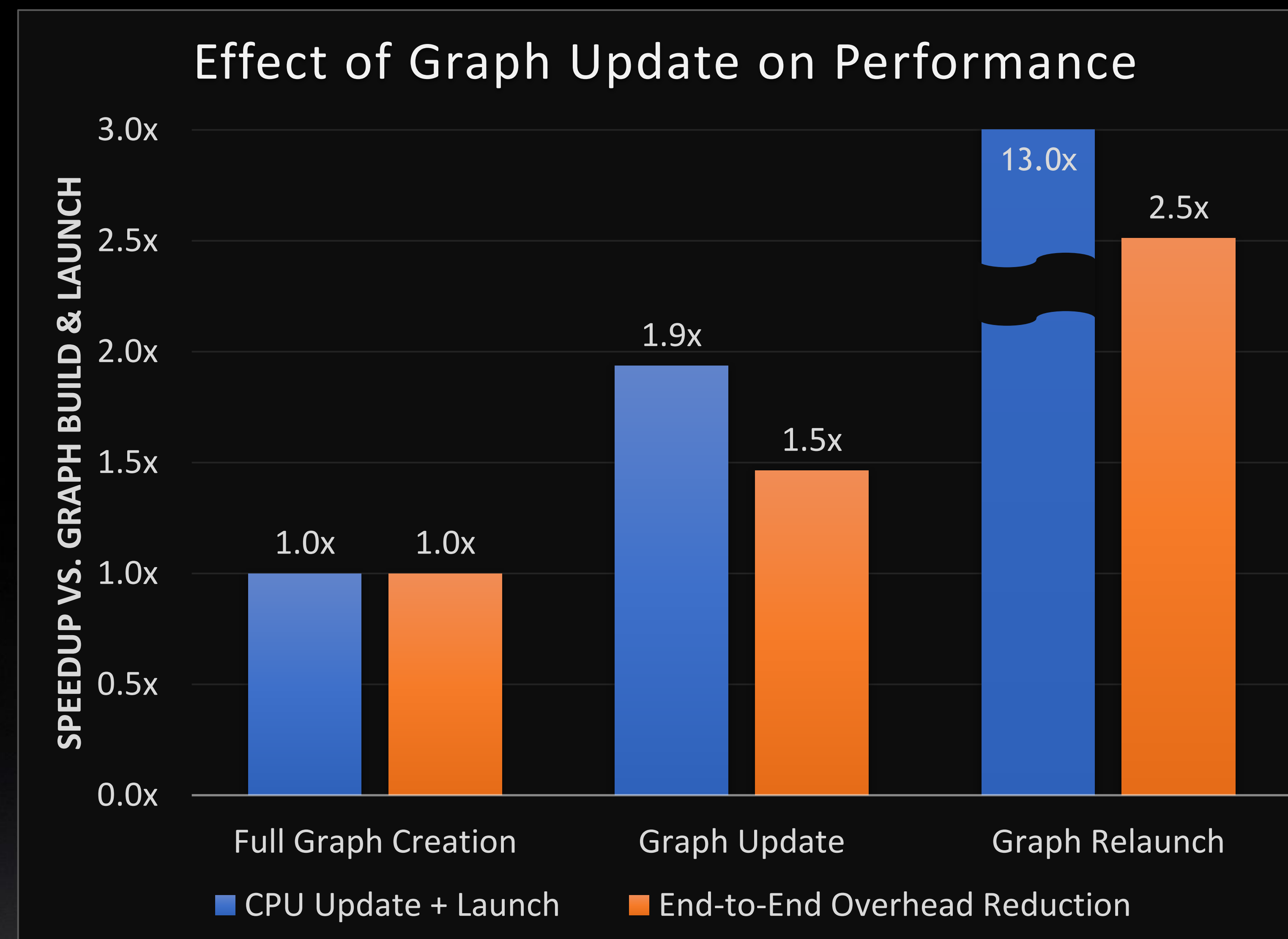
Modify parameters without rebuilding graph

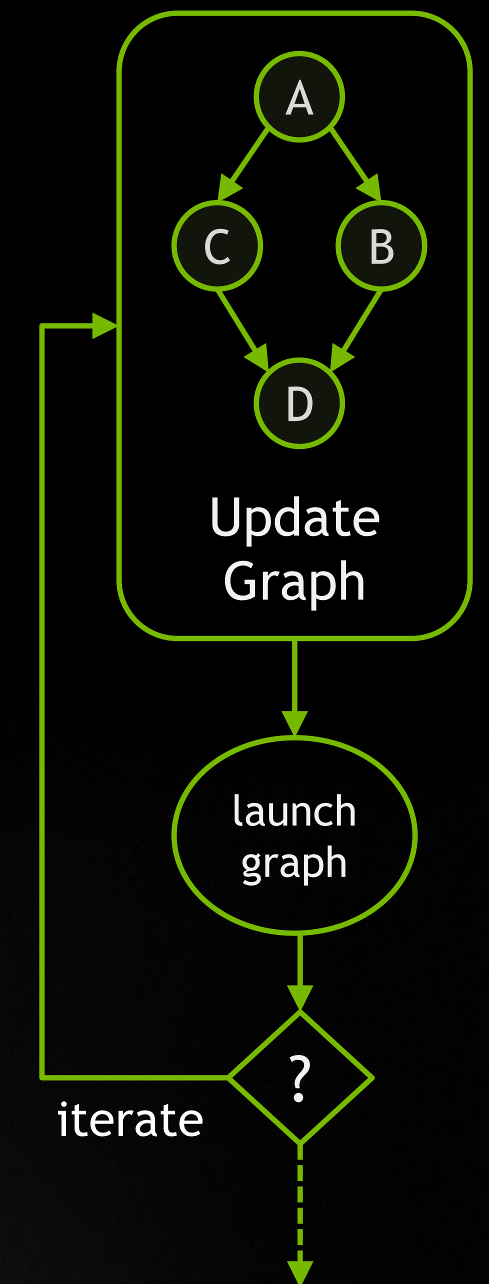
Change launch configuration, kernel parameters, memcopy args, etc.

Topology of graph may not change

Nearly 2x speedup on CPU

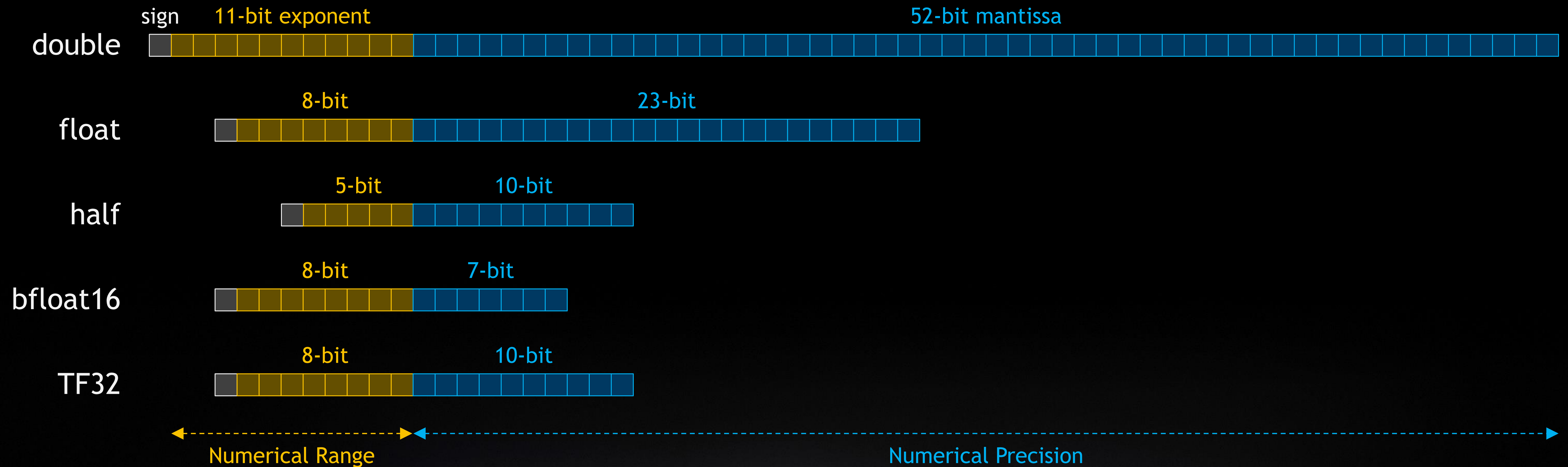
50% end-to-end overhead reduction





NVIDIA

FLOATING POINT FORMATS & PRECISION



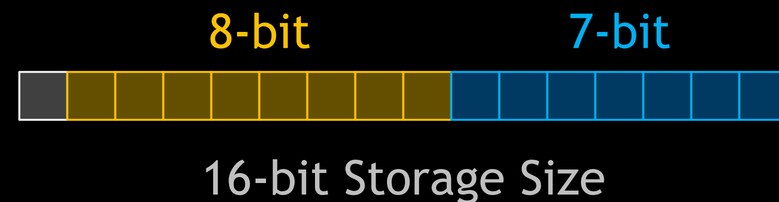
$$\text{value} = (-1)^{\text{sign}} \times 2^{\text{exponent}} \times (1 + \text{mantissa})$$

NVIDIA

NEW FLOATING POINT FORMATS: BF16 & TF32

Both Match fp32 8-bit Exponent: Covers The Same Range of Values

bfloat16

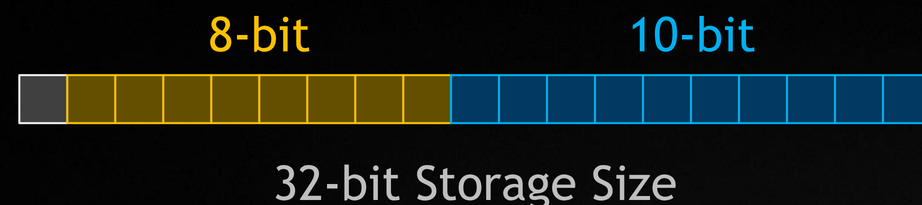


Available in CUDA C++ as nv_bfloat16 numerical type

Full CUDA C++ numerical type - `#include <cuda_fp16.h>`

Can use in both host & device code, and in templated functions*

TF32



Tensor Core math mode for single-precision training

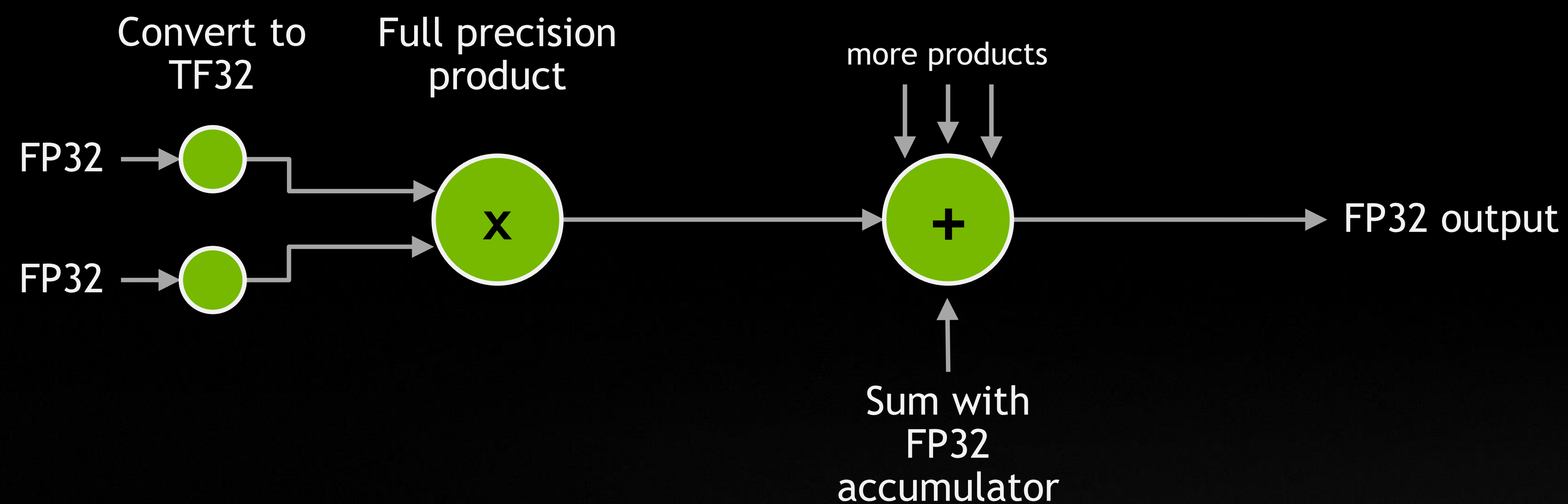
Not a numerical type - tensor core inputs are rounded to TF32

CUDA C++ programs use `float` (fp32) throughout

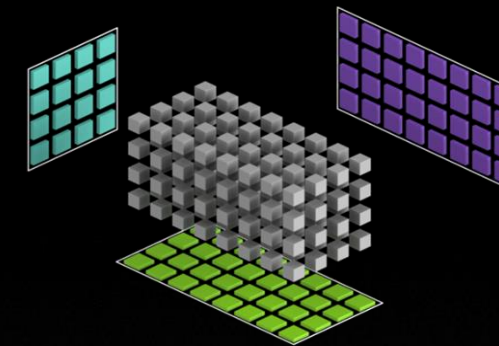
*(similar to CUDA's IEEE-FP16 "`half`" type)

TENSOR FLOAT 32 - TENSOR CORE MODE

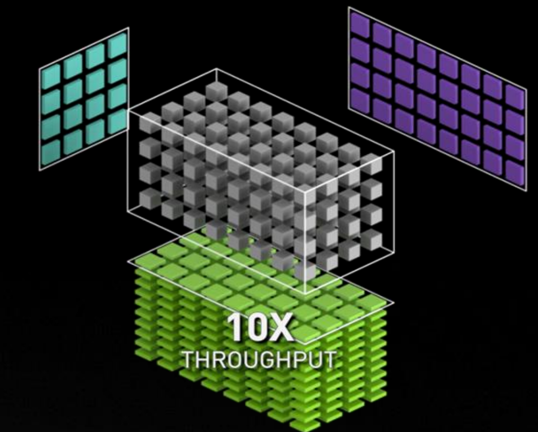
A100 Tensor Core Input Precision
All Internal Operations Maintain Full FP32 Precision



NVIDIA V100 FP32



NVIDIA A100 Tensor Core TF32



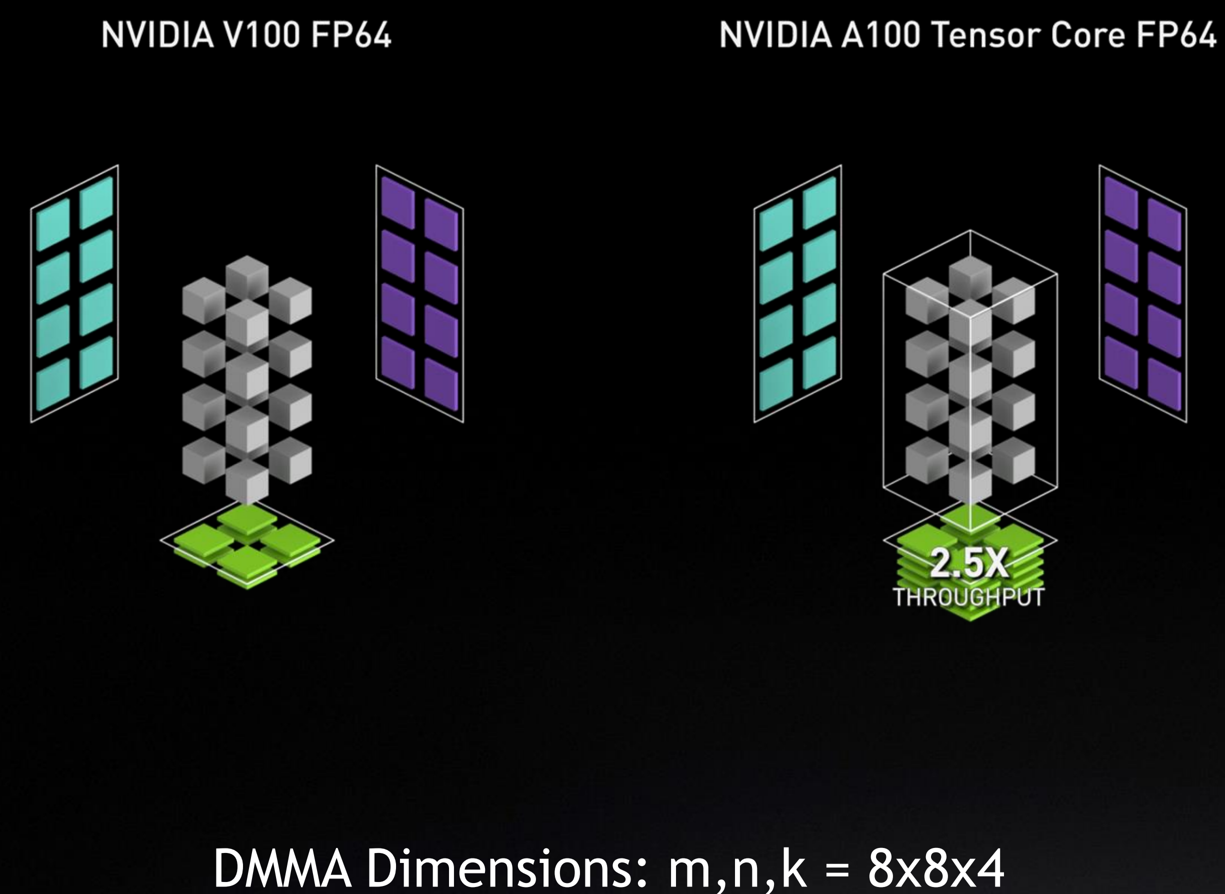
TF32 MMA Dimensions: m,n,k = 16x8x8

For more information see: S22082: Mixed-Precision Training of Neural Networks

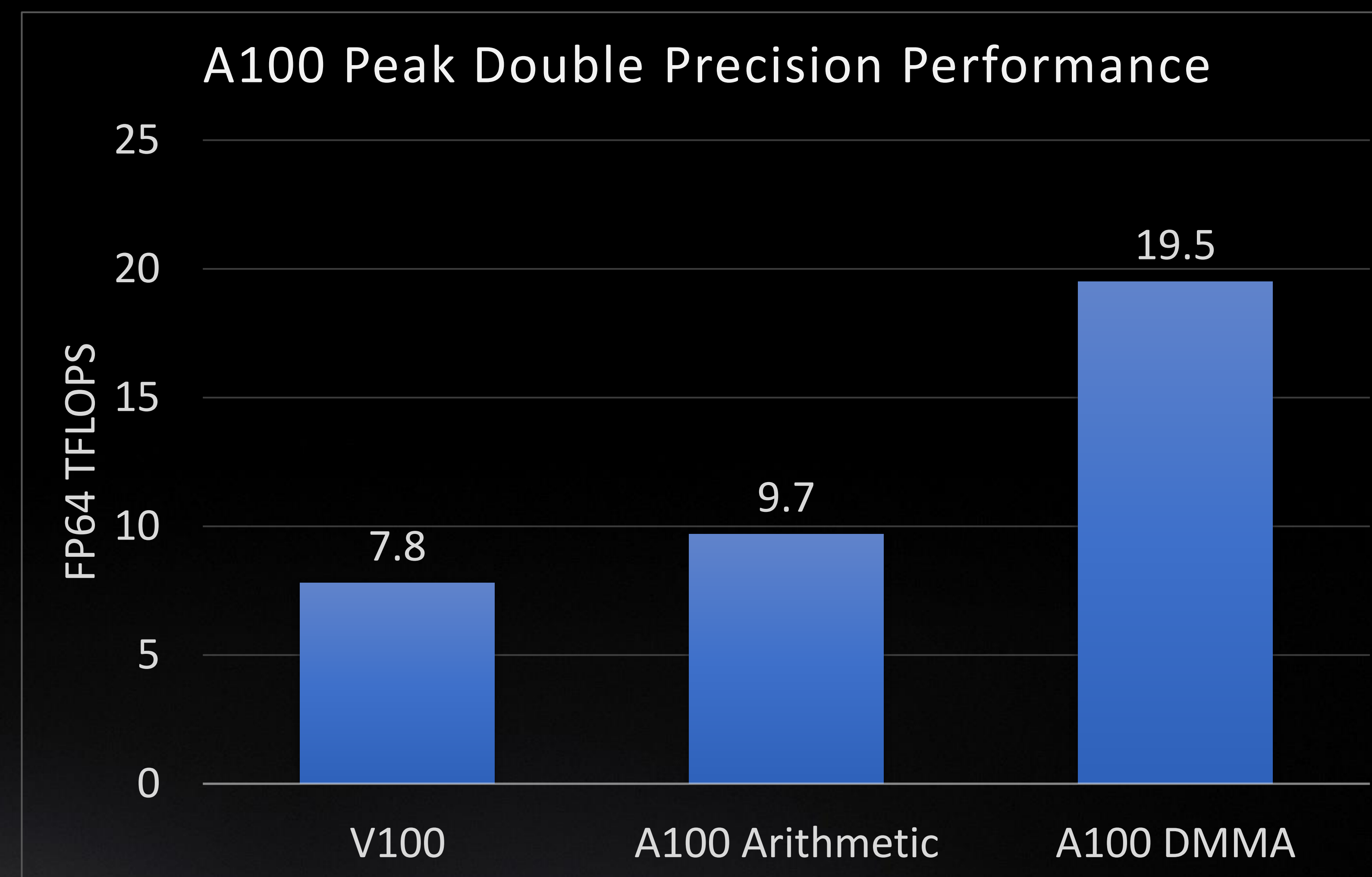
NVIDIA

A100 INTRODUCES DOUBLE PRECISION TENSOR CORES

All A100 Tensor Core Internal Operations Maintain Full FP64 Precision

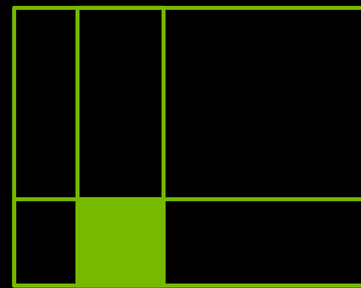


DMMA Dimensions: m,n,k = 8x8x4



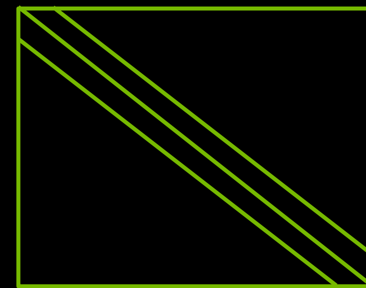
NVIDIA

A100 GPU ACCELERATED MATH LIBRARIES IN CUDA 11.0



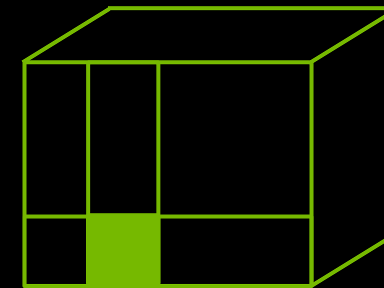
cuBLAS

BF16, TF32 and FP64 Tensor Cores



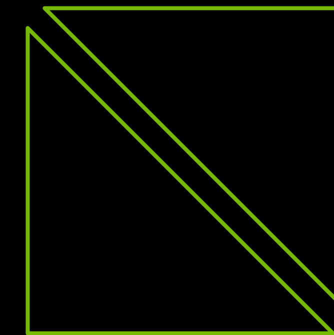
cuSPARSE

Increased memory BW, Shared Memory & L2



cuTENSOR

BF16, TF32 and FP64 Tensor Cores

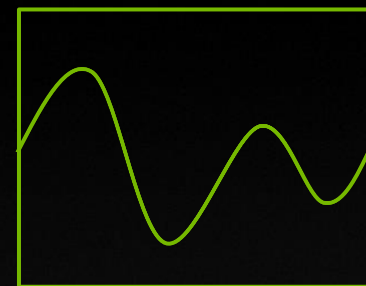


cuSOLVER

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

For more information see: S21681 - How CUDA Math Libraries Can Help You Unleash the Power of the New NVIDIA A100 GPU

CUTLASS – TENSOR CORE PROGRAMMING MODEL

Warp-Level GEMM and Reusable Components for Linear Algebra Kernels in CUDA

CUTLASS 2.2

Optimal performance on NVIDIA Ampere microarchitecture

New floating-point types: `nv_bfloat16, TF32, double`

Deep software pipelines with async memcopy

CUTLASS 2.1

BLAS-style host API

CUTLASS 2.0

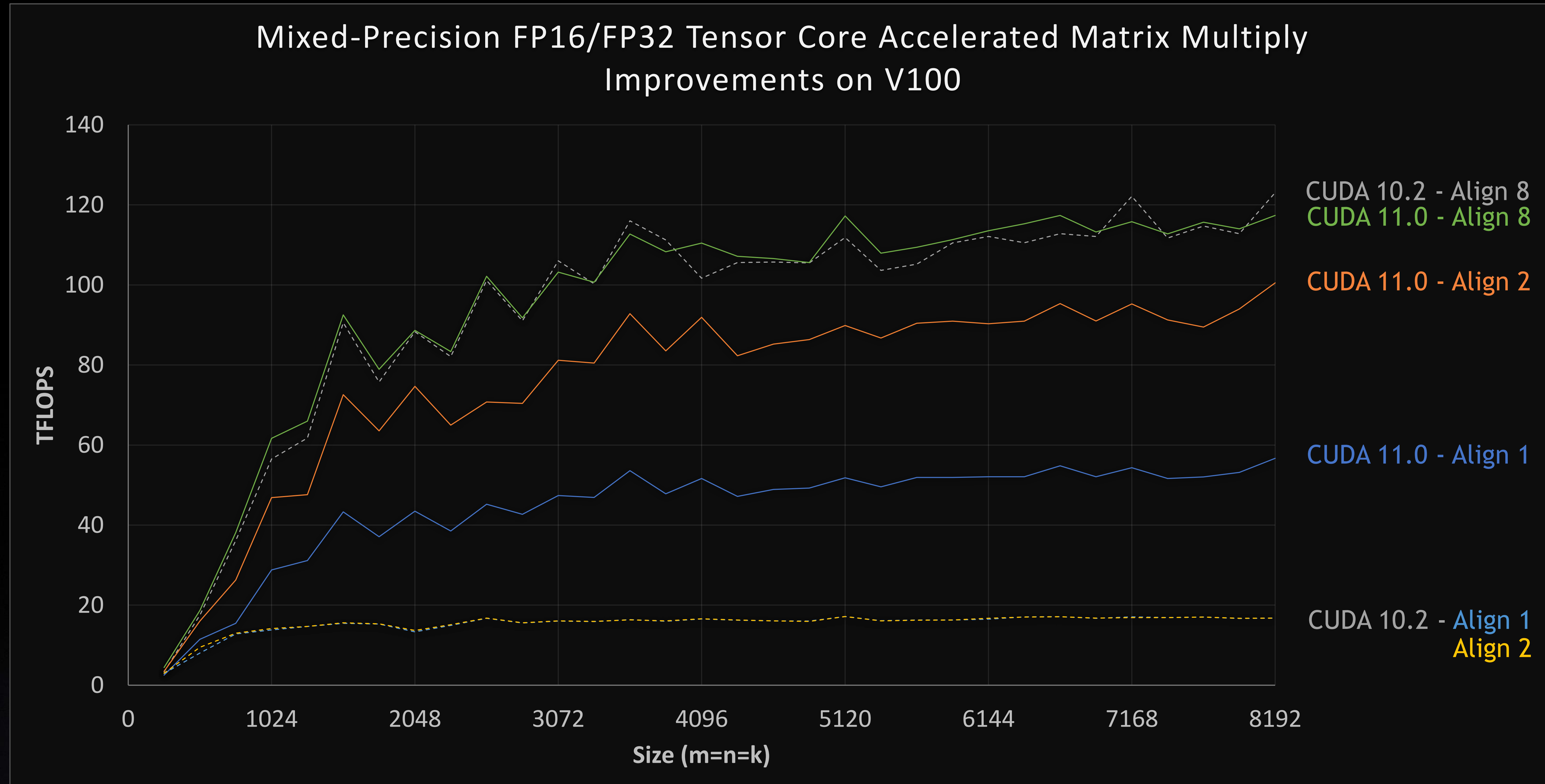
Significant refactoring using modern C++11 programming

```
using Mma = cutlass::gemm::warp::DefaultMmaTensorOp<
  GemmShape<64, 64, 16>,
  half_t, LayoutA,                   // GEMM A operand
  half_t, LayoutB,                   // GEMM B operand
  float, RowMajor                    // GEMM C operand
>;

__shared__ ElementA smem_buffer_A[Mma::Shape::kM * GemmK];
__shared__ ElementB smem_buffer_B[Mma::Shape::kN * GemmK];

// Construct iterators into SMEM tiles
Mma::IteratorA iter_A({smem_buffer_A, lda}, thread_id);
Mma::IteratorB iter_B({smem_buffer_B, ldb}, thread_id);

Mma::FragmentA frag_A;
Mma::FragmentB frag_B;
Mma::FragmentC accum;

Mma mma;

accum.clear();

#pragma unroll 1
for (int k = 0; k < GemmK; k += Mma::Shape::kK) {


  iter_A.load(frag_A);  // Load fragments from A and B matrices
  iter_B.load(frag_B);

  ++iter_A; ++iter_B;   // Advance along GEMM K to next tile in A
                        //   and B matrices

                        // Compute matrix product
  mma(accum, frag_A, frag_B, accum);
}
```

For more information see: S21745 - Developing CUDA Kernels to Push Tensor Cores to the Absolute Limit

cuBLAS

Eliminating Alignment Requirements To Activate Tensor Cores for MMA



AlignN means alignment to 16-bit multiples of N. For example, align8 are problems aligned to 128bits or 16 bytes.

NVIDIA

MATH LIBRARY DEVICE EXTENSIONS

Introducing cuFFTDx: Device Extension

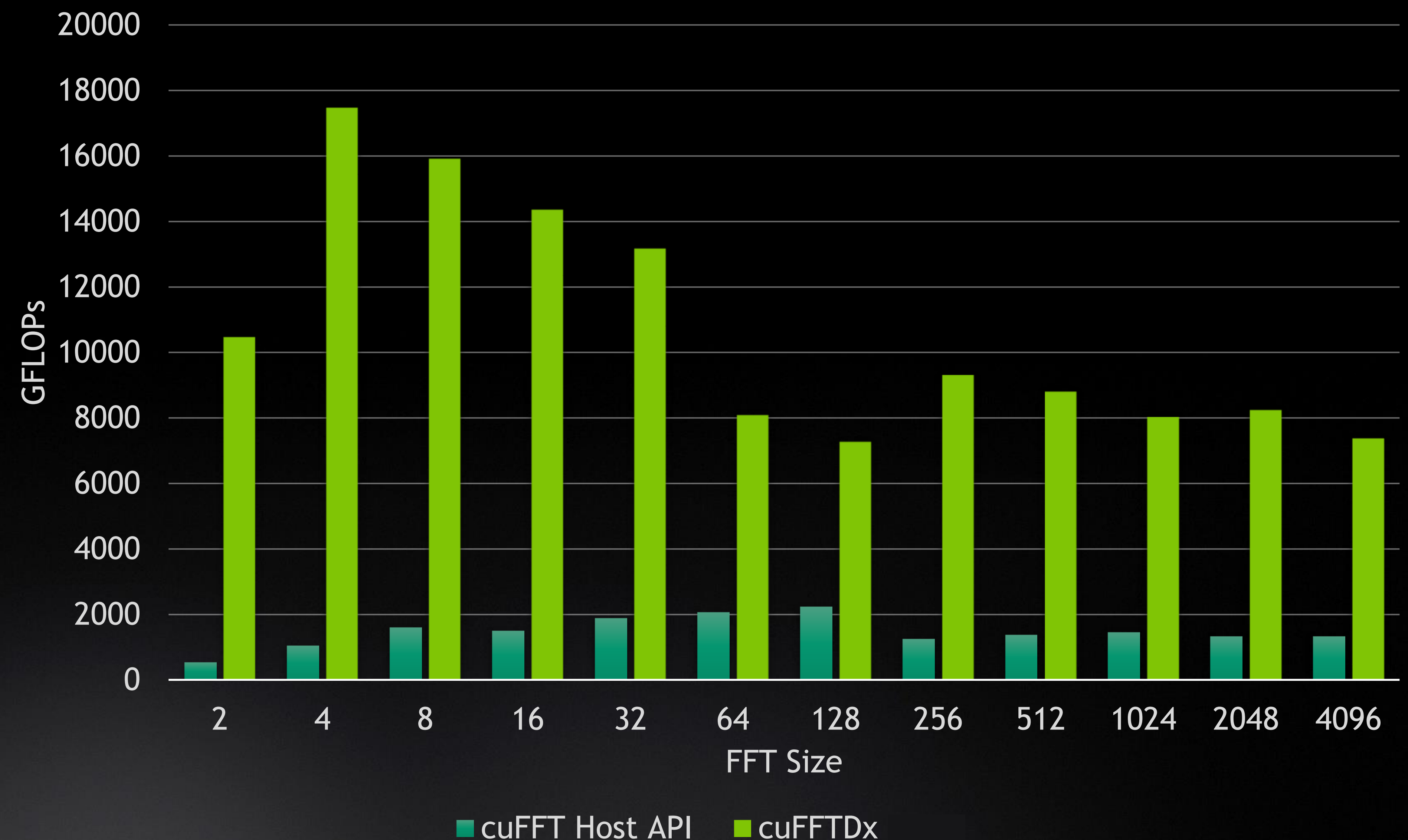
Available in Math Library EA Program

Device callable library

Retain and reuse on-chip data

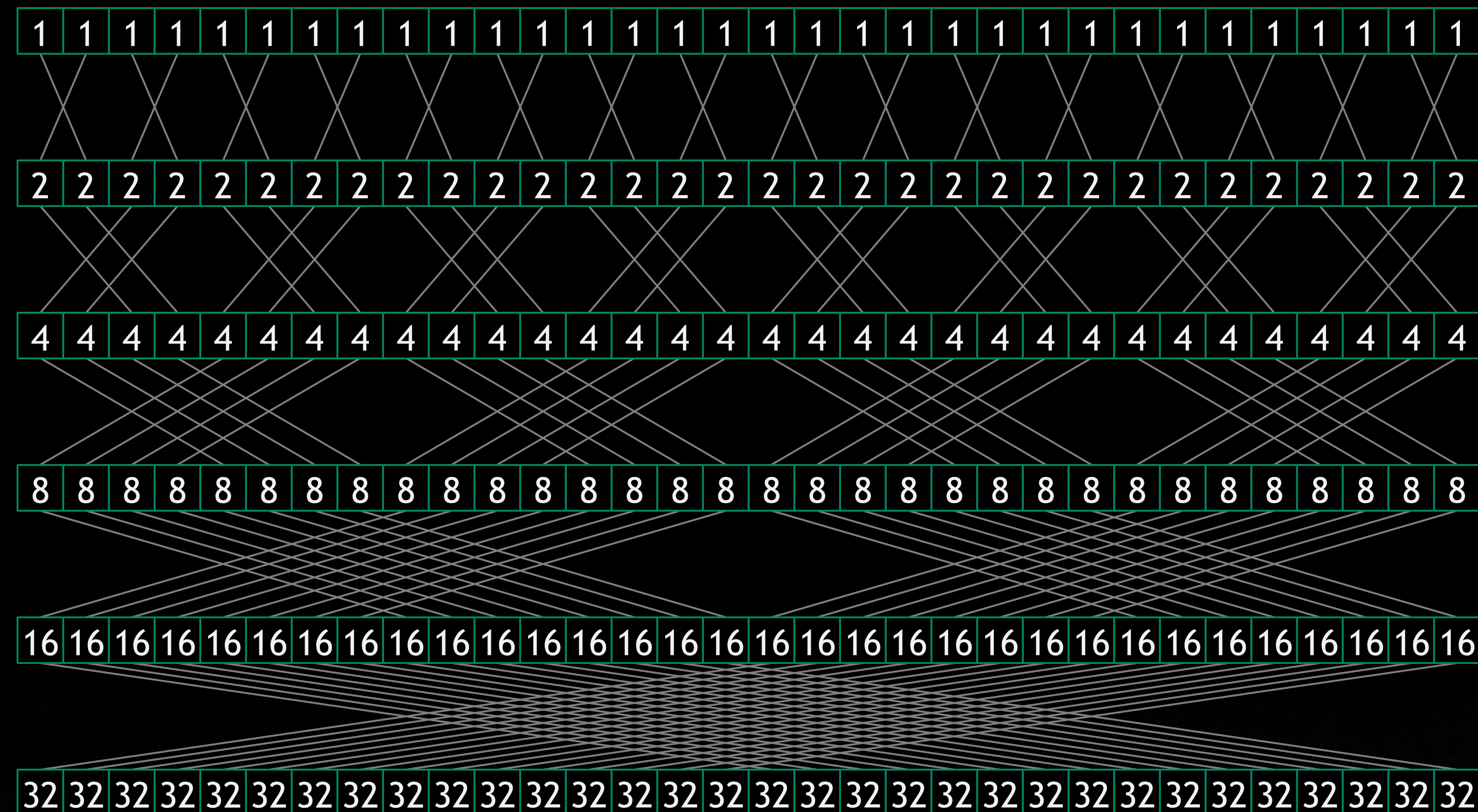
Inline FFTs in user kernels

Combine multiple FFT operations



https://developer.nvidia.com/CUDAMathLibraryEA

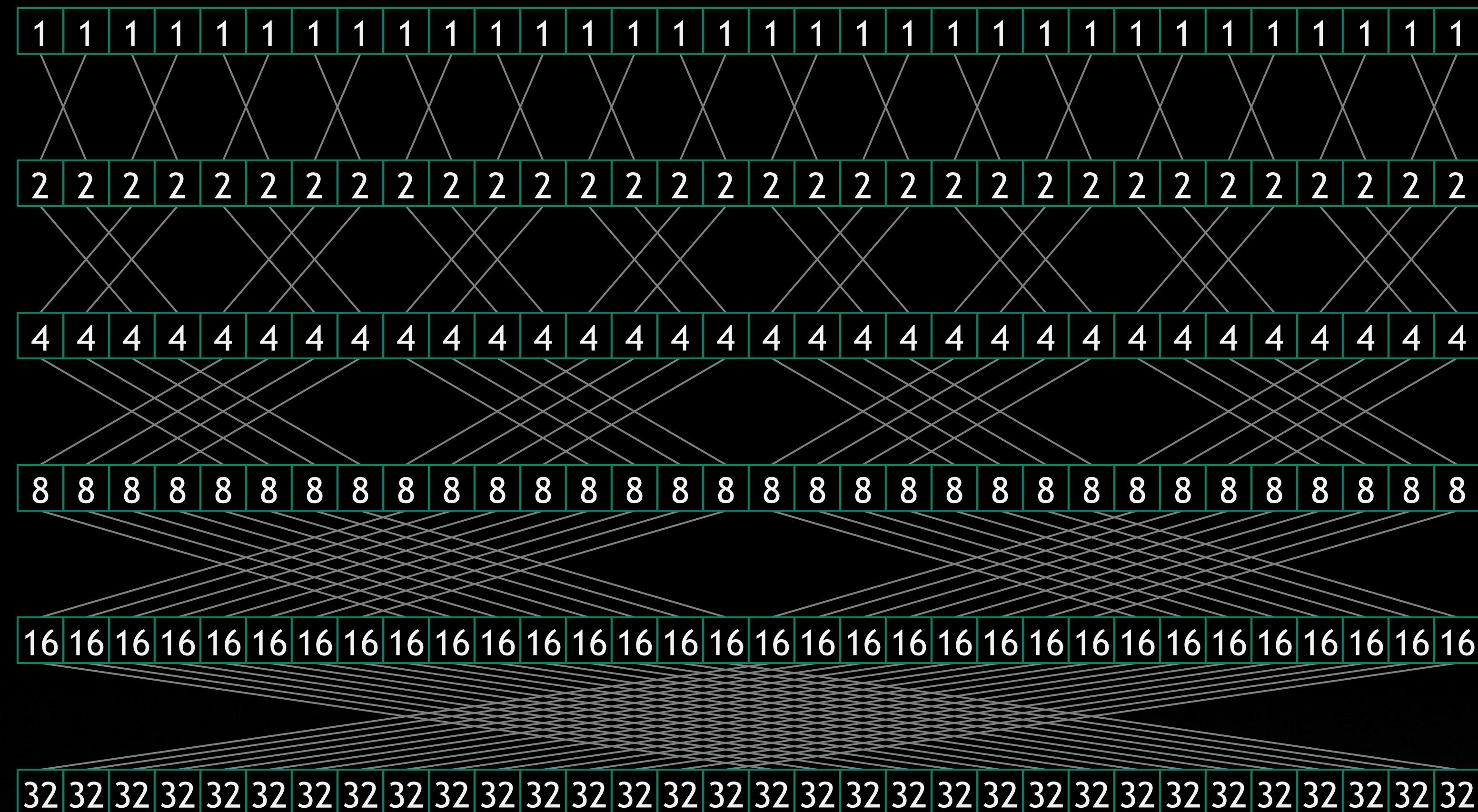
WARP-WIDE REDUCTION USING __shfl

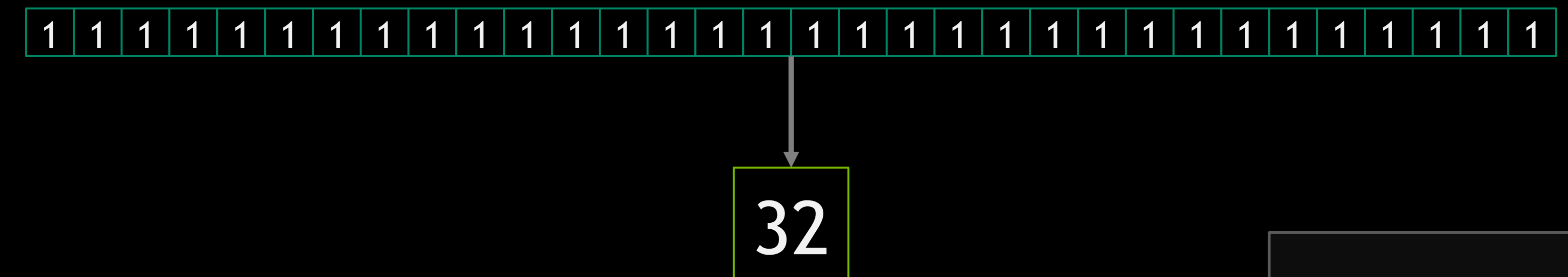



```
__device__ int reduce(int value) {
      value += __shfl_xor_sync(0xFFFFFFFF, value, 1);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 2);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 4);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 8);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 16);

      return value;
}
```


NVIDIA.

WARP-WIDE REDUCTION IN A SINGLE STEP





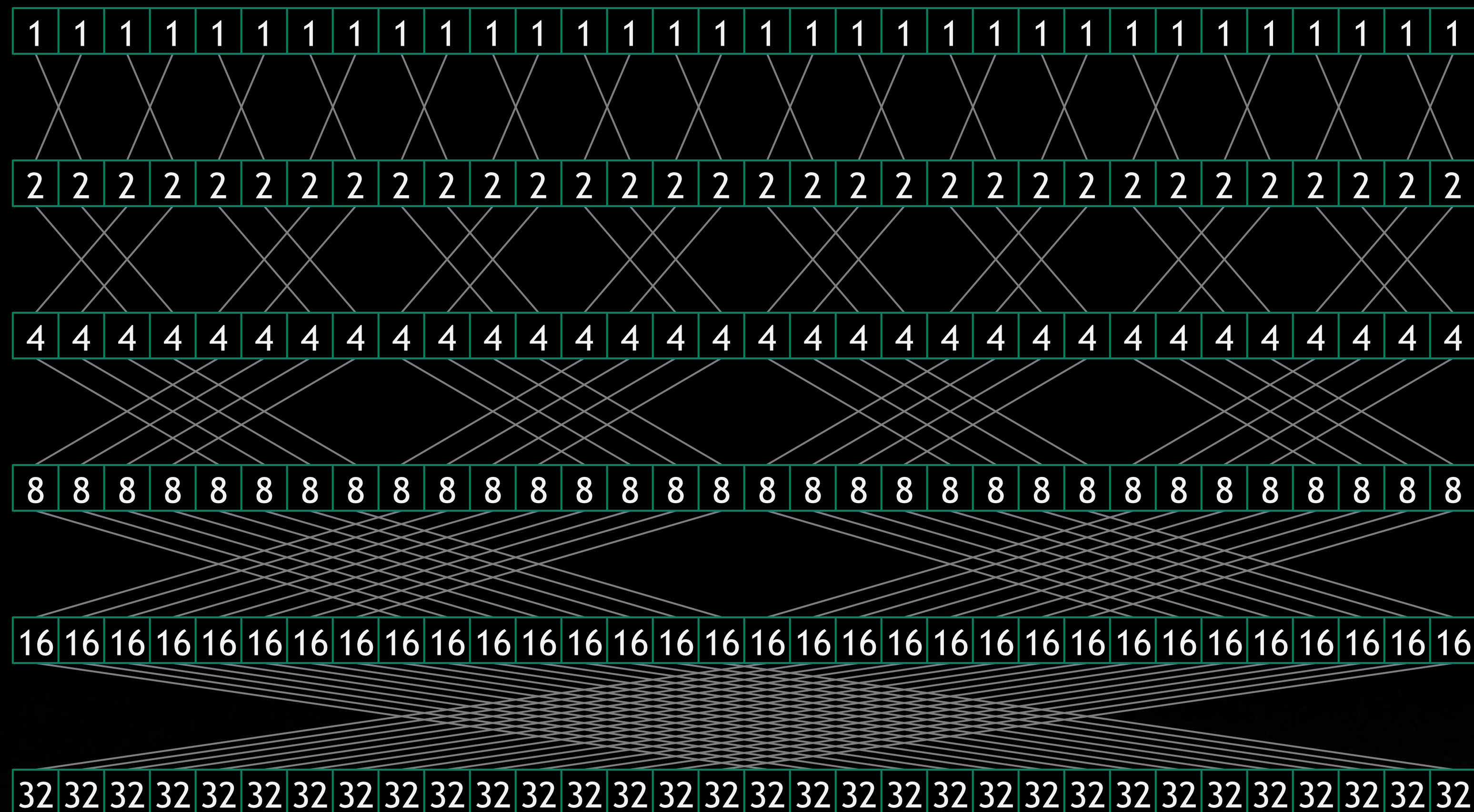
Supported operations

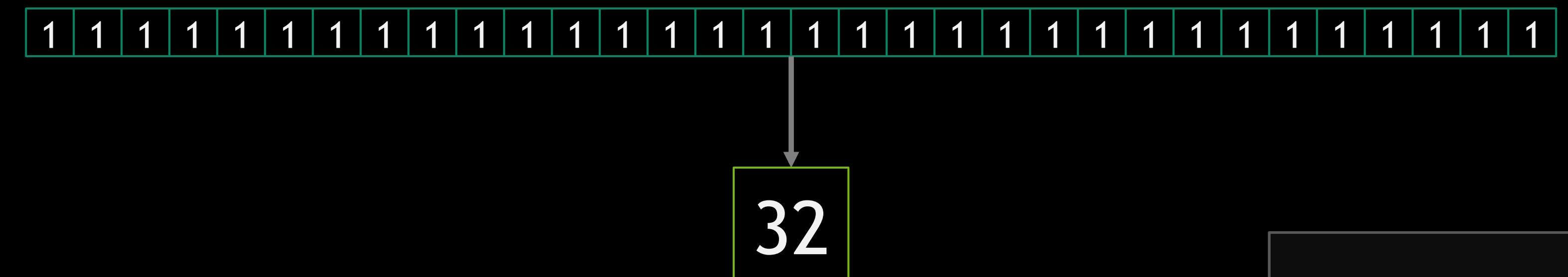
add
min
max
and
or
xor

```
__device__ int reduce(int value) {
      value += __shfl_xor_sync(0xFFFFFFFF, value, 1);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 2);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 4);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 8);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 16);

      return value;
}
```

```
int total = __reduce_add_sync(0xFFFFFFFF, value);
```

WARP-WIDE REDUCTION IN A SINGLE STEP





Supported operations

add
min
max
and
or
xor

```
__device__ int reduce(int value) {
      value += __shfl_xor_sync(0xFFFFFFFF, value, 1);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 2);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 4);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 8);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 16);

      return value;
}
```

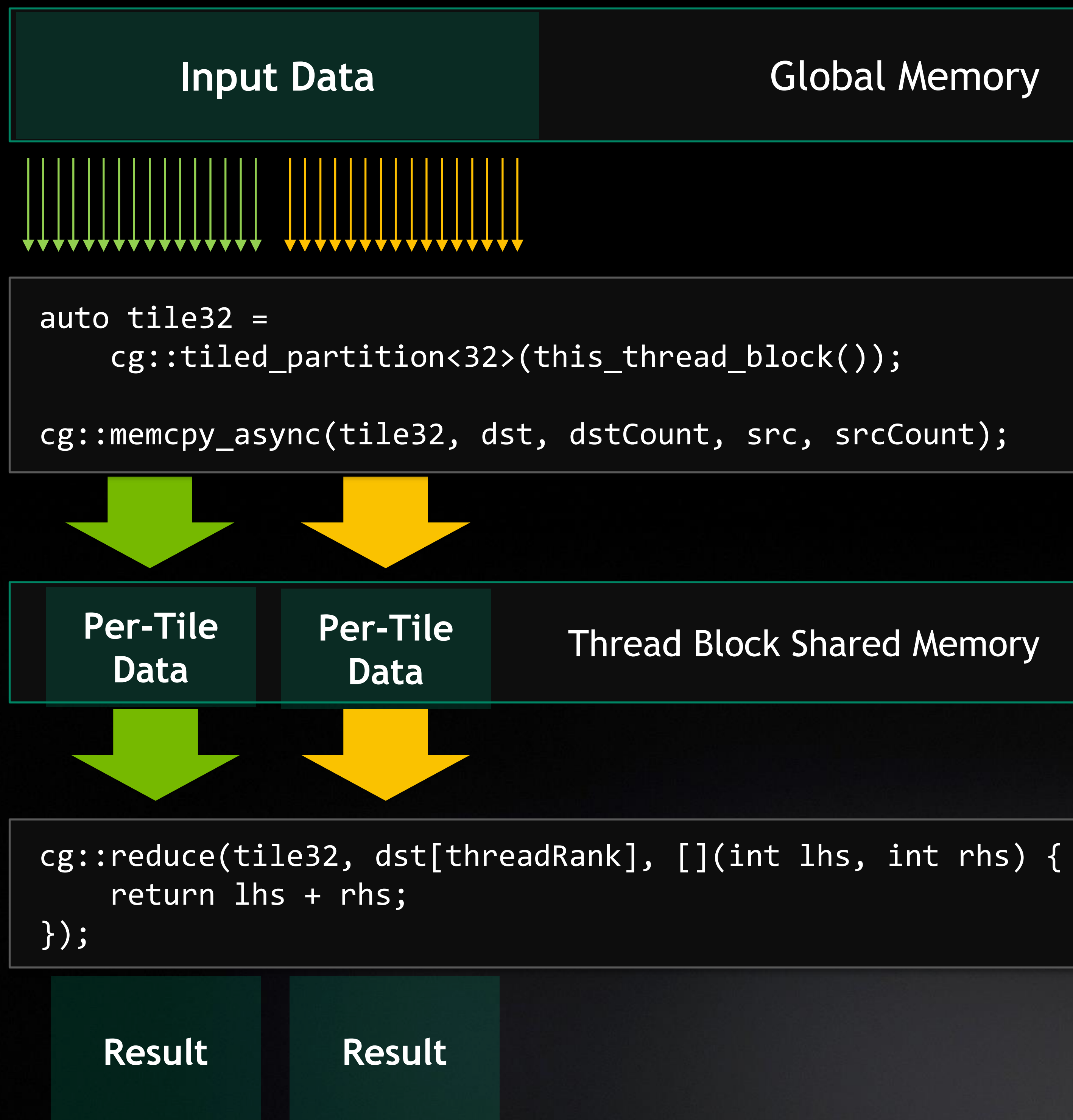
```
int total = __reduce_add_sync(0xFFFFFFFF, value);
```

```
thread_block_tile<32> tile32 =
            tiled_partition<32>(this_thread_block());

// Works on all GPUs back to Kepler
cg::reduce(tile32, value, cg::plus<int>());
```

NVIDIA

COOPERATIVE GROUPS

Cooperative Groups Features Work On All GPU Architectures (incl. Kepler)



Cooperative Groups Updates

No longer requires separate compilation

30% faster grid synchronization

New platforms Support (Windows and Linux + MPS)

Can now capture cooperative launches in a CUDA graph

cg::reduce also accepts C++ lambda as reduction operation

NVIDIA.

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

For more information see: S21766 - Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing

ISO C++ == Language + Standard Library

NVIDIA.

ISO C++ == Language + Standard Library

CUDA C++ == Language

NVIDIA.

libcu++ : THE CUDA C++ STANDARD LIBRARY

ISO C++ == Language + Standard Library

CUDA C++ == Language + **libcu++**

Strictly conforming to ISO C++, plus conforming extensions

Opt-in, Heterogeneous, Incremental

NVIDIA.

`cuda::std::`

Opt-in	Does not interfere with or replace your host standard library
Heterogeneous	Copyable/Movable objects can migrate between host & device Host & Device can call all member functions Host & Device can concurrently use synchronization primitives*
Incremental	A subset of the standard library today Each release adds more functionality

*Synchronization primitives must be in managed memory and be declared with `cuda::std::thread_scope_system`

NVIDIA.

libcu++ NAMESPACE HIERARCHY

```
// ISO C++, __host__ only
#include <atomic>
std::atomic<int> x;

// CUDA C++, __host__ __device__
// Strictly conforming to the ISO C++
#include <cuda/std/atomic>
cuda::std::atomic<int> x;

// CUDA C++, __host__ __device__
// Conforming extensions to ISO C++
#include <cuda/atomic>
cuda::atomic<int, cuda::thread_scope_block> x;
```

For more information see: S21262 - The CUDA C++ Standard Library

NVIDIA

CUDA C++ HETEROGENEOUS ARCHITECTURE

Thrust
Host code Standard Library-inspired primitives
e.g: *for_each*, *sort*, *reduce*

CUB
Re-usable building blocks, targeting 3 layers of abstraction

libcu++
Heterogeneous ISO C++ Standard Library

CUB is now a fully-supported component of the CUDA Toolkit. Thrust integrates CUB’s high performance kernels.

NVIDIA.

CUB: CUDA UNBOUND

Reusable Software Components for Every Layer of the CUDA Programming Model

Device-wide primitives

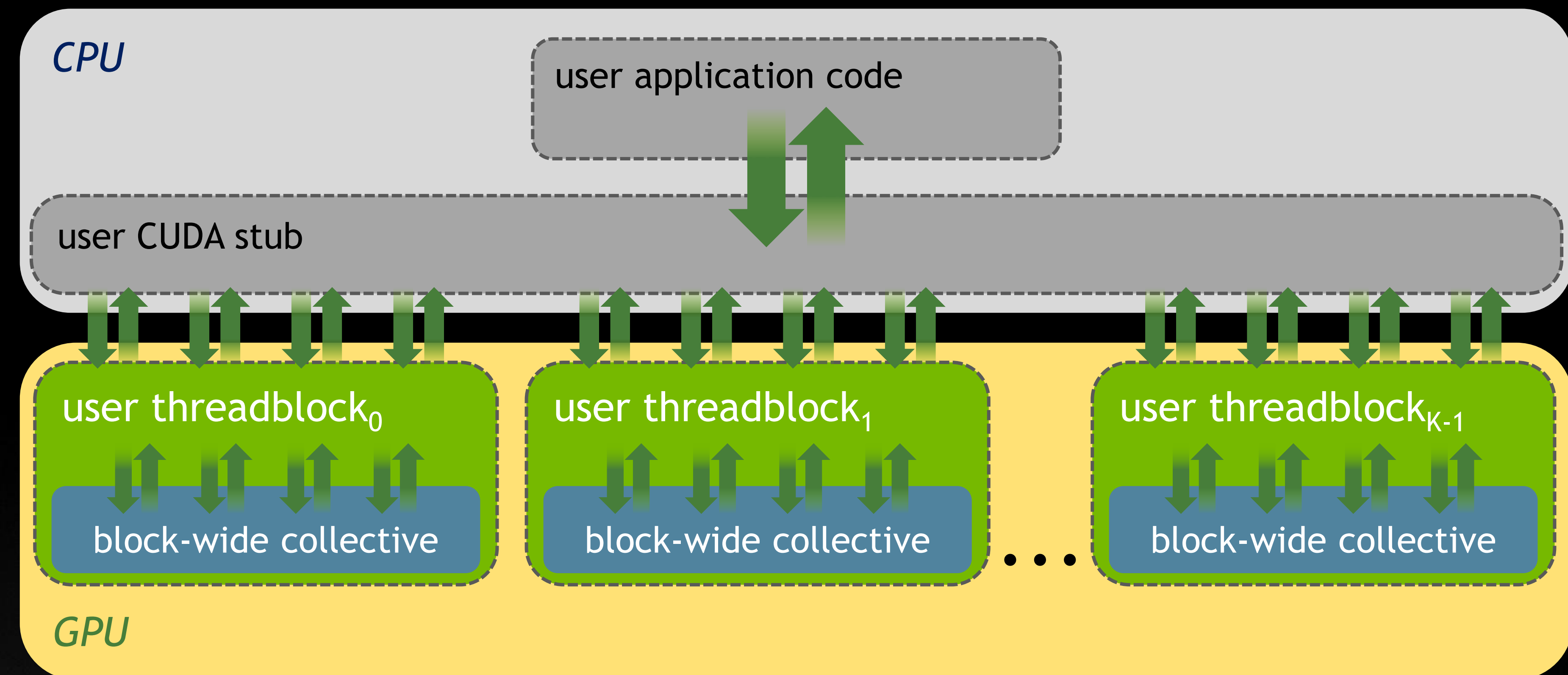
Parallel sort, prefix scan, reduction, histogram, etc.
Compatible with CUDA dynamic parallelism

Block-wide "collective" primitives

Cooperative I/O, sort, scan, reduction, histogram, etc.
Compatible with arbitrary thread block sizes and types

Warp-wide "collective" primitives

Cooperative warp-wide prefix scan, reduction, etc.
Safely specialized for each underlying CUDA architecture



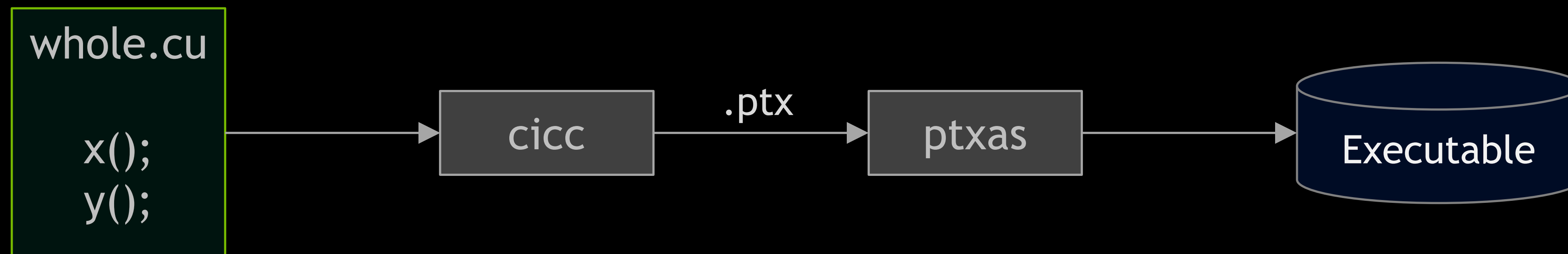
NVIDIA

NVCC HIGHLIGHTS IN CUDA 11.0 TOOLKIT

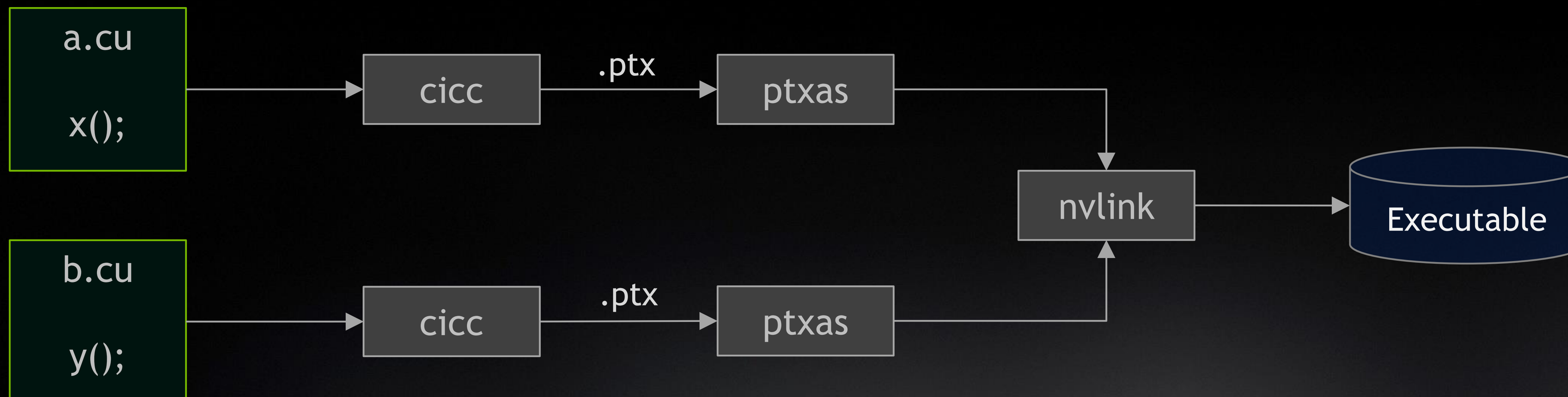
Key Features	
ISO C++ 17 CUDA Support	*Preview feature*
Link-Time Optimization	*Preview feature*

New in CUDA 11.0	
Accept duplicate CLI options across all NVCC sub-components	
Host compiler support for GCC 9, clang 9, PGI 20.1	
Host compiler version check override option	--allow-unsupported-compiler
Native AArch64 NVCC binary with ARM Allinea Studio 19.2 C/C++ and PGI 20 host compiler support	

LINK-TIME OPTIMIZATION



Whole-Program Compilation



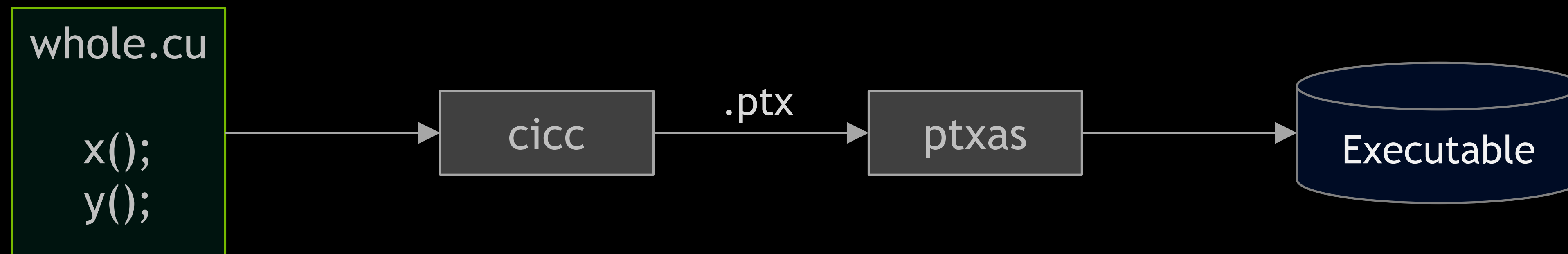
Separate Compilation

All cross-compilation-unit calls must link via ABI, e.g:

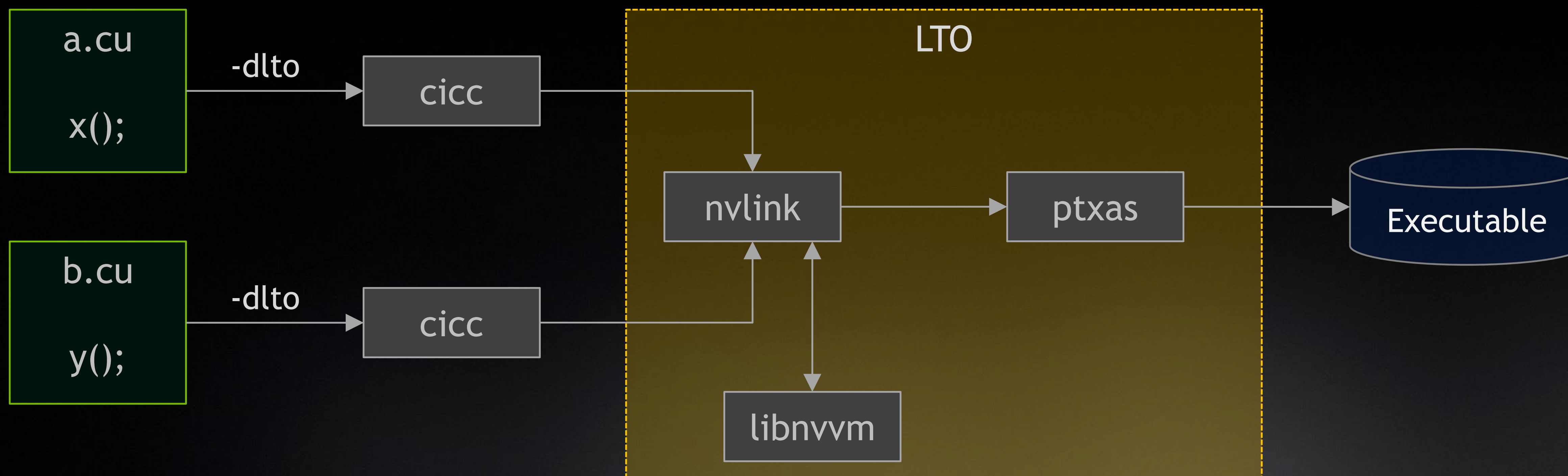
x() → y()

ABI calls incur call overheads

LINK-TIME OPTIMIZATION



Whole-Program Compilation



Link-Time Optimization

Permits inlining of device functions across modules

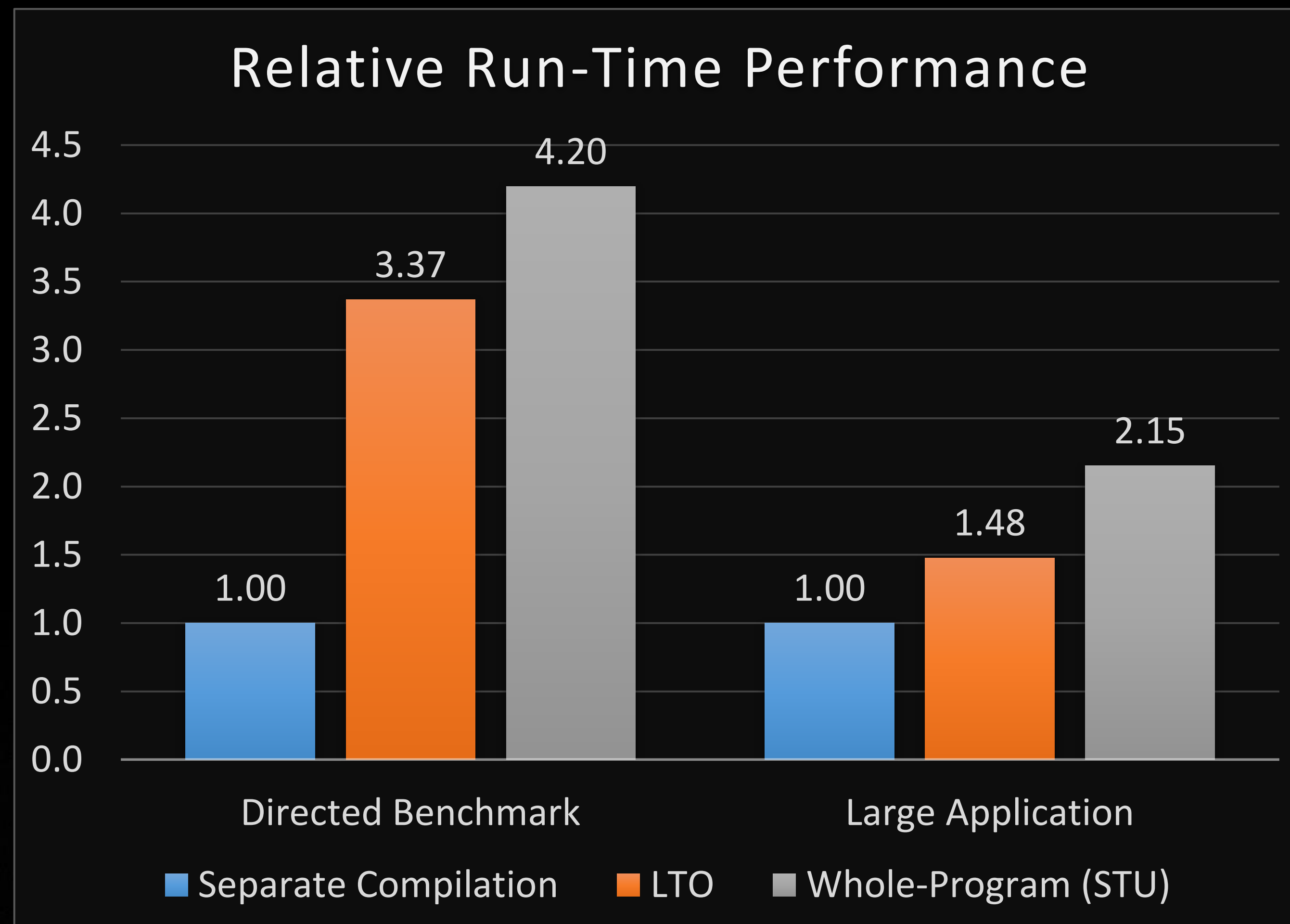
Mitigates ABI call overheads

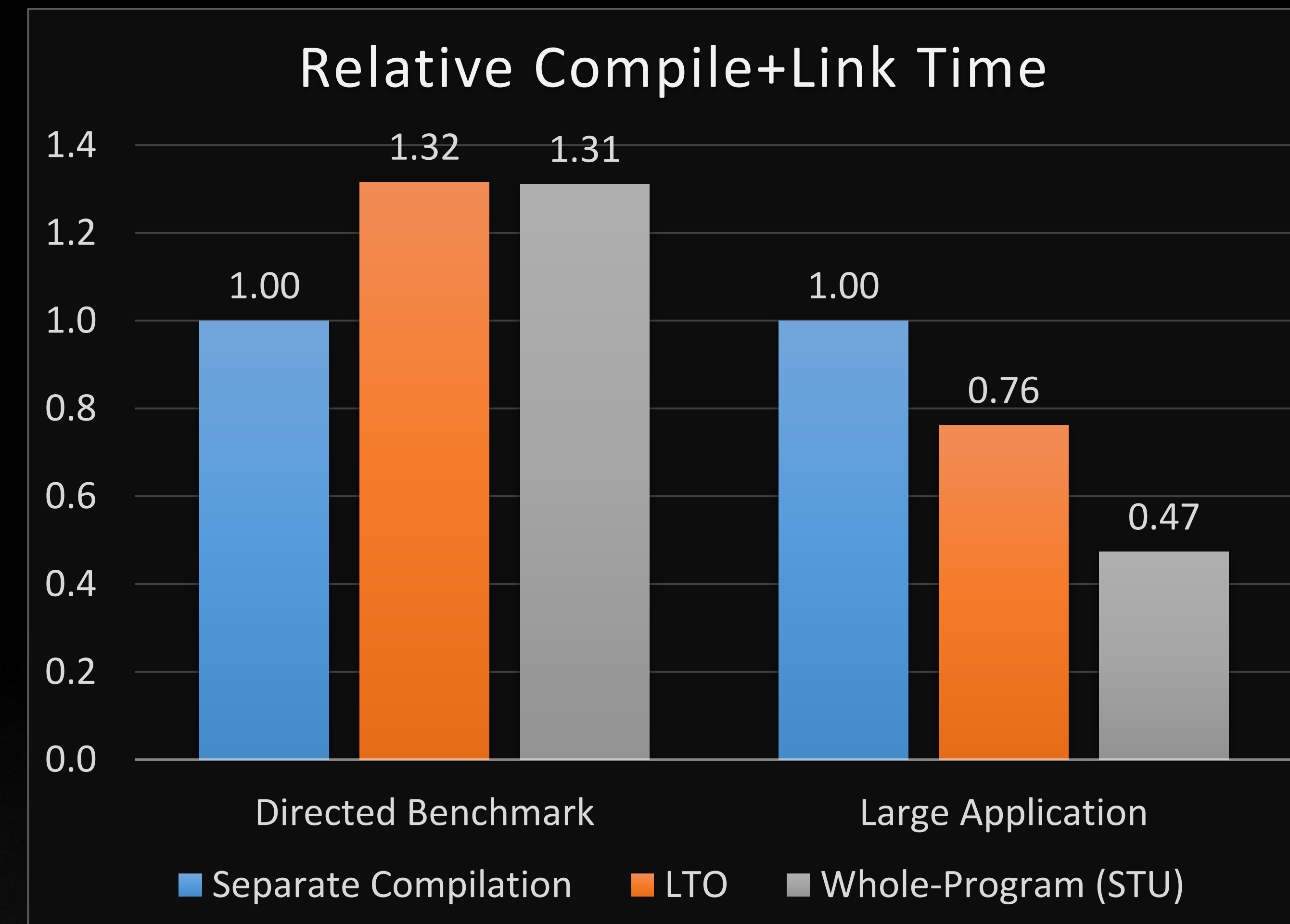
Facilitates Dead Code Elimination

NVIDIA.

LINK-TIME OPTIMIZATION

Preview Release in CUDA 11.0



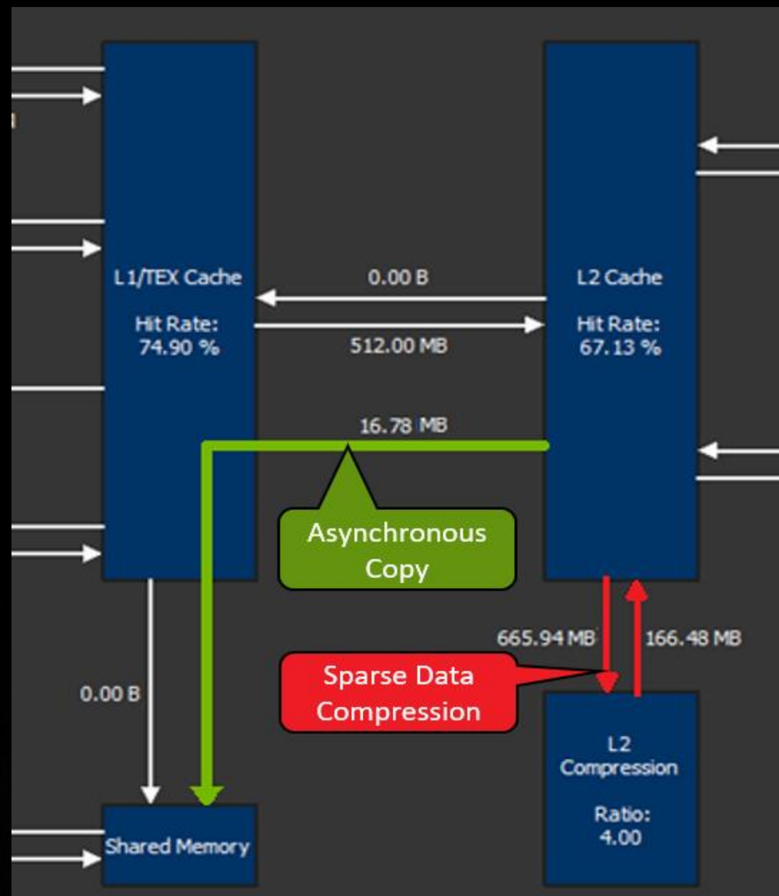


Enabled through ***-dlto*** option for compile and link steps

Partial LTO (mix of separate compilation & LTO) supported

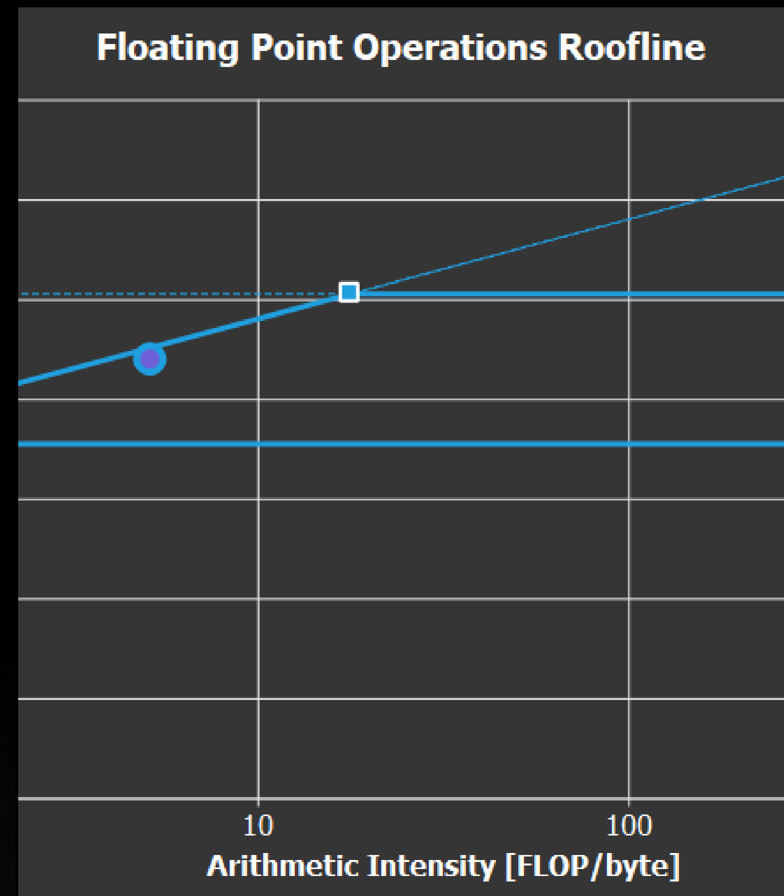
NVIDIA

NSIGHT COMPUTE 2020.1



Chips Update

A100 GPU Support



Advanced Analysis

Roofline
New Memory Tables

Sampling Data (All)

	Value
.cu:76 (0xc0118edd0 in GPUBlackScholesCallPut)	1,152
.cu:77 (0xc0118ef30 in GPUBlackScholesCallPut)	1,152
.cu:106 (0xc0118fac0 in GPUBlackScholesCallPut)	65
.cu:77 (0xc0118ef40 in GPUBlackScholesCallPut)	49
.cu:106 (0xc0118faf0 in GPUBlackScholesCallPut)	39

Sampling Data (Not Issued)

	Value
.cu:77 (0xc0118ef30 in GPUBlackScholesCallPut)	870
.cu:76 (0xc0118edd0 in GPUBlackScholesCallPut)	850
.cu:106 (0xc0118fac0 in GPUBlackScholesCallPut)	40
.cu:77 (0xc0118ef40 in GPUBlackScholesCallPut)	37
.cu:106 (0xc0118faf0 in GPUBlackScholesCallPut)	23

Most Instructions Executed

	Value
.cu:106 (0xc0118fb70 in GPUBlackScholesCallPut)	122,880
.cu:106 (0xc0118faf0 in GPUBlackScholesCallPut)	122,880

Workflow Improvements

Hot Spot Tables
Section Links

ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e3a0
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e430
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e450
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e4e0
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e530
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e540
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e560
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e640
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e670
ected 8192 transactions, got 10240 (1.25x) at PC 0x7f5762f3e690

Other Changes

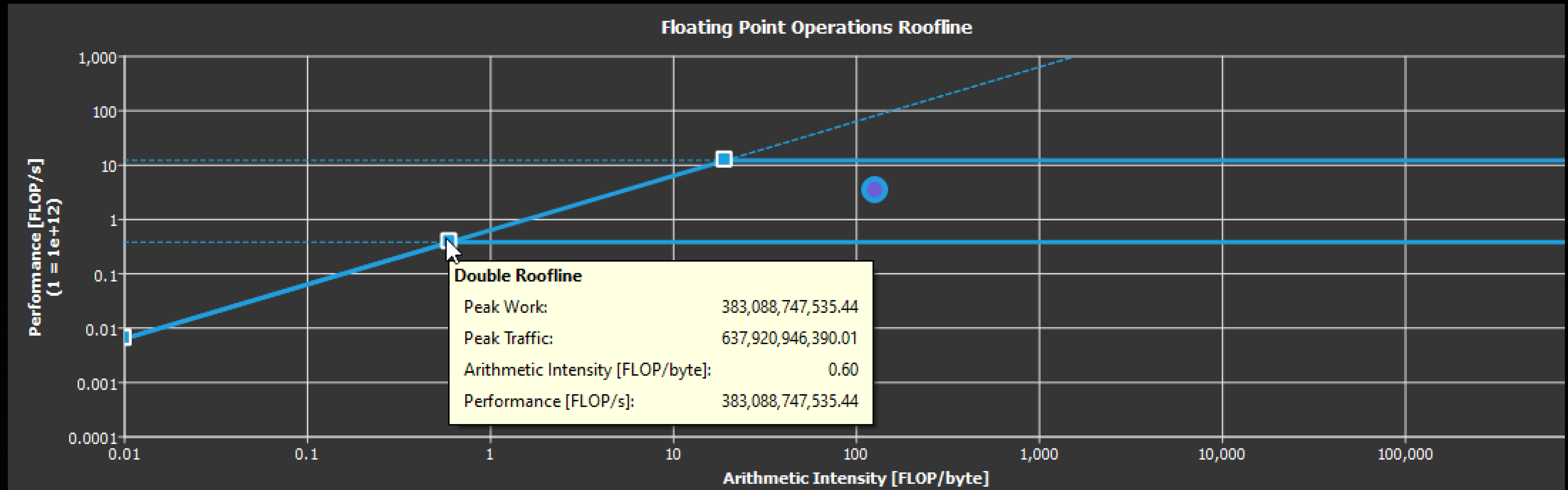
New Rules, Names

For more information see: S21771 - Optimizing CUDA kernels using Nsight Compute

NSIGHT COMPUTE 2020.1

New Roofline Analysis

Efficient way to evaluate kernel characteristics, quickly understand potential directions for further improvements or existing limiters



Inputs
Arithmetic Intensity (FLOPS/bytes)
Performance (FLOPS/s)

Ceilings
Peak Memory Bandwidth
Peak FP32/FP64 Performance

NVIDIA.

COMPUTE-SANITIZER

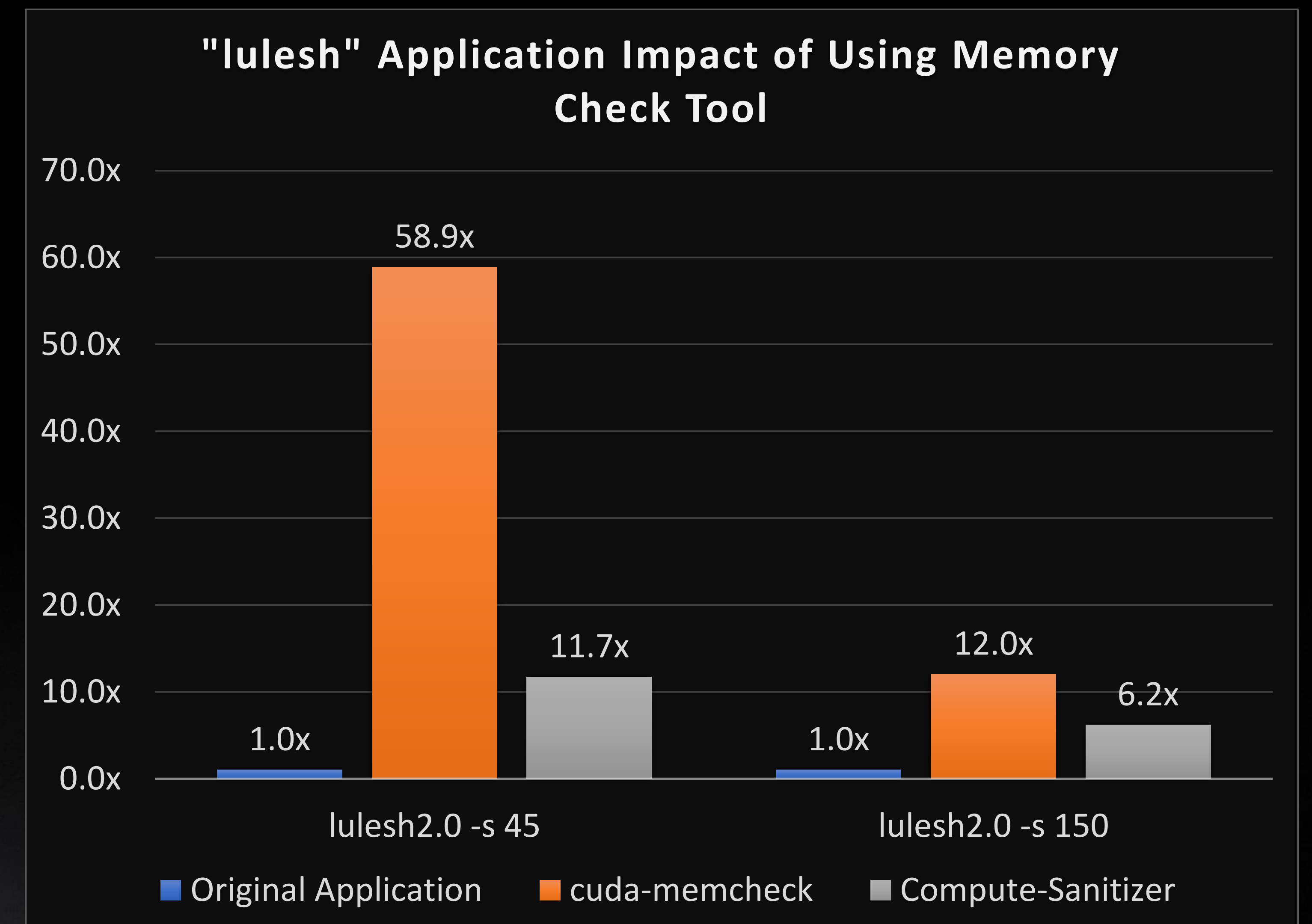
Command Line Interface (CLI) Tool Based On The Sanitizer API

Next-Gen Replacement Tool for `cuda-memcheck`

Significant performance improvement of 2x - 5x compared with `cuda-memcheck` (depending on application size)

Performance gain for applications using libraries such as CUSOLVER, CUFFT or DL frameworks

`cuda-memcheck` still supported in CUDA 11.0 (does not support Arm SBSA)



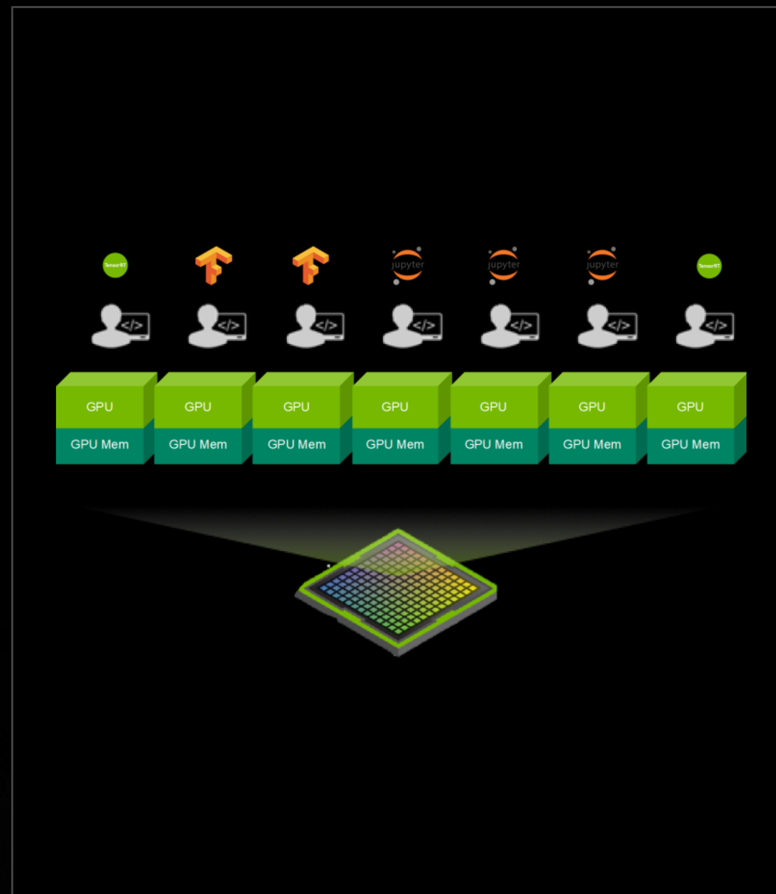
https://docs.nvidia.com/cuda/compute-sanitizer

For more information see: S22043 – CUDA Developer Tools: Overview and Exciting New Features

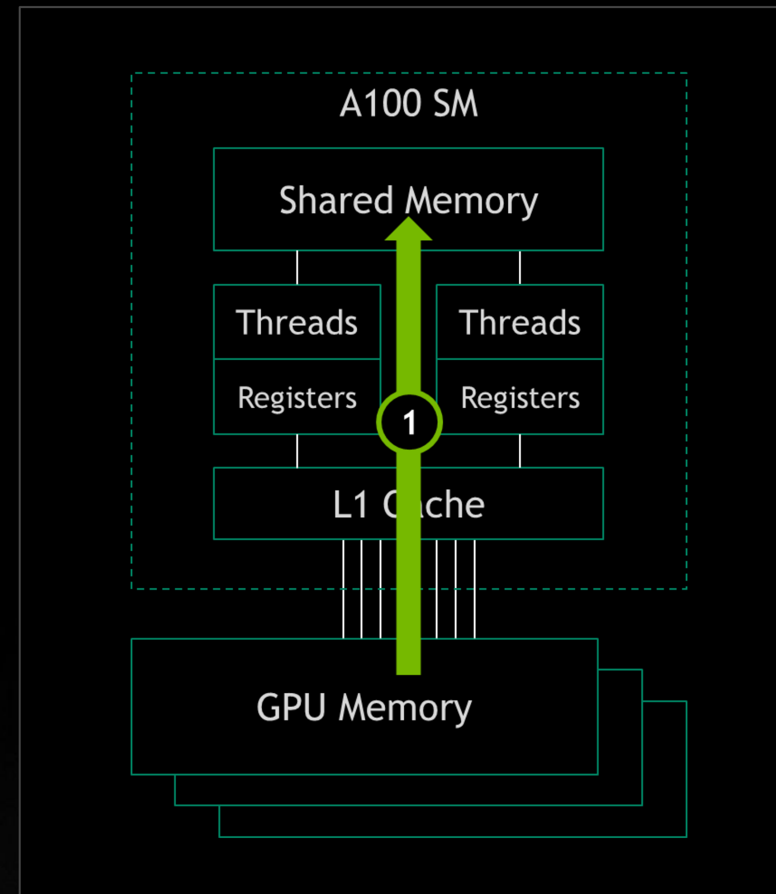
NVIDIA

CUDA 11.0: AVAILABLE FOR DOWNLOAD SOON

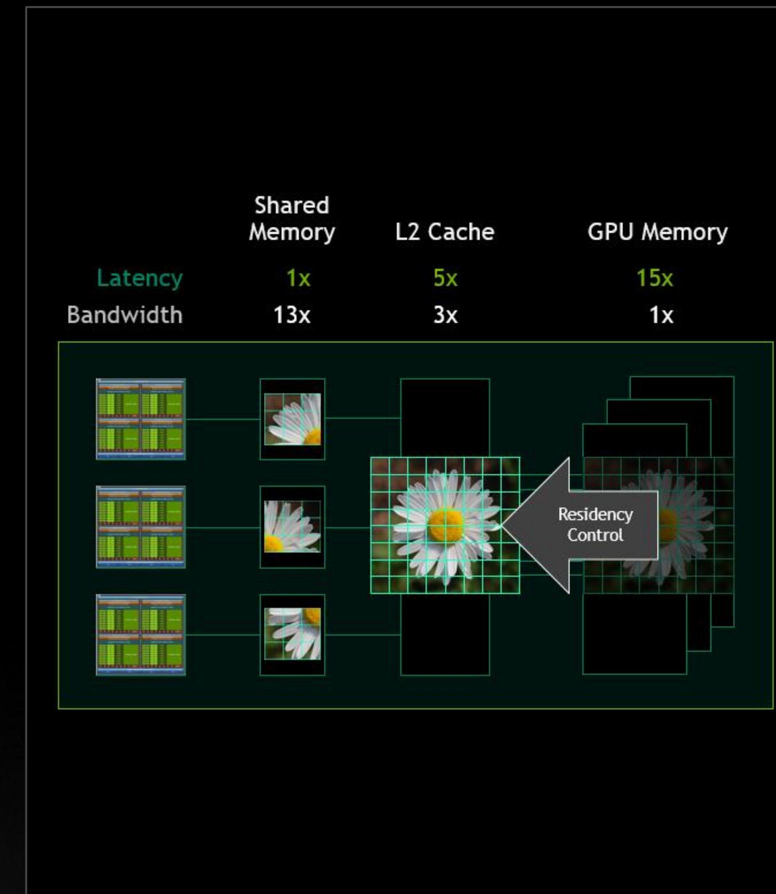
https://developer.nvidia.com/cuda-downloads



Hierarchy



Asynchrony



Latency

```
// ISO C++, __host__ only
#include <atomic>
std::atomic<int> x;

// CUDA C++, __host__ __device__
// Strictly conforming to the ISO C++
#include <cuda/std/atomic>
cuda::std::atomic<int> x;

// CUDA C++, __host__ __device__
// Conforming extensions to ISO C++
#include <cuda/atomic>
cuda::atomic<int, cuda::thread_scope_block> x;
```

Language

REFERENCES

Deep dive into any of the topics you've seen by following these links

S21730	Inside the NVIDIA Ampere Architecture
Whitepaper	https://www.nvidia.com/nvidia-ampere-architecture-whitepaper
S22043	CUDA Developer Tools: Overview and Exciting New Features
Developer Blog	https://devblogs.nvidia.com/introducing-low-level-gpu-virtual-memory-management/
S21975	Inside NVIDIA's Multi-Instance GPU Feature
S21170	CUDA on NVIDIA GPU Ampere Architecture, Taking your algorithms to the next level of...
S21819	Optimizing Applications for NVIDIA Ampere GPU Architecture
S22082	Mixed-Precision Training of Neural Networks
S21681	How CUDA Math Libraries Can Help You Unleash the Power of the New NVIDIA A100 GPU
S21745	Developing CUDA Kernels to Push Tensor Cores to the Absolute Limit
S21766	Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing
S21262	The CUDA C++ Standard Library
S21771	Optimizing CUDA kernels using Nsight Compute

https://developer.nvidia.com/cuda-downloads