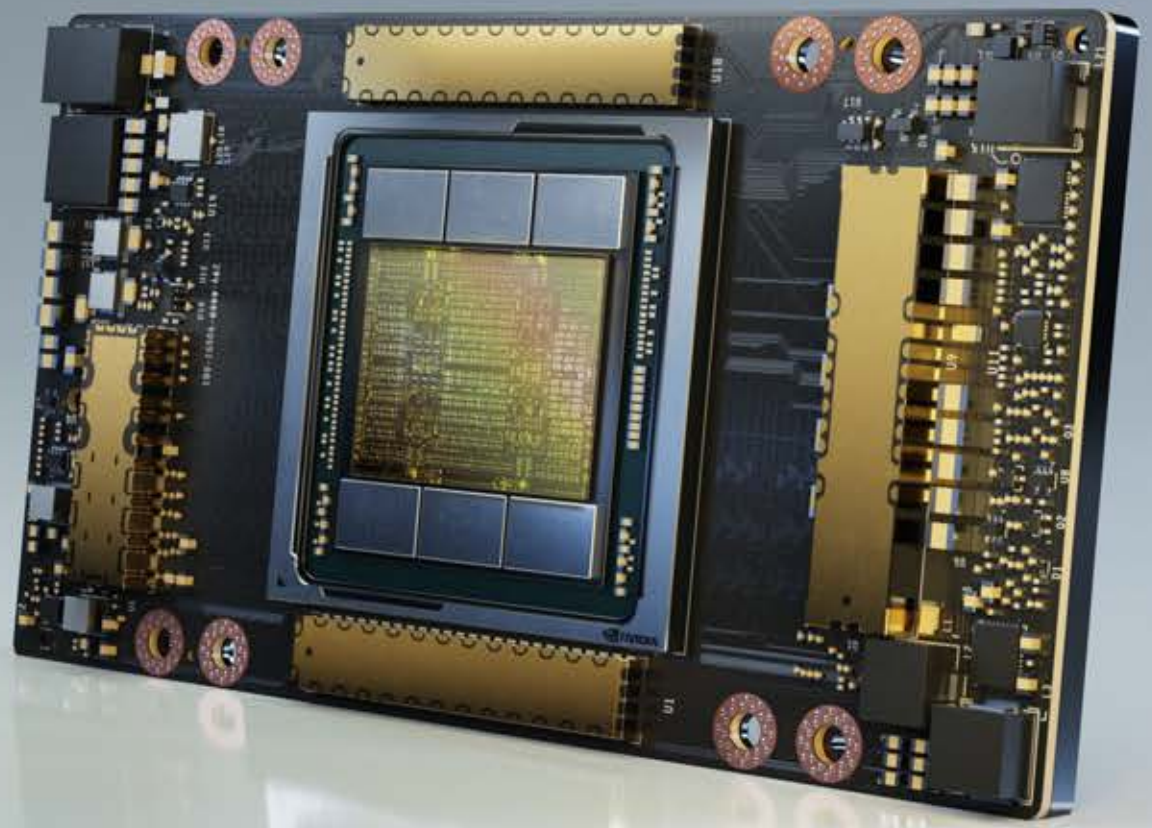
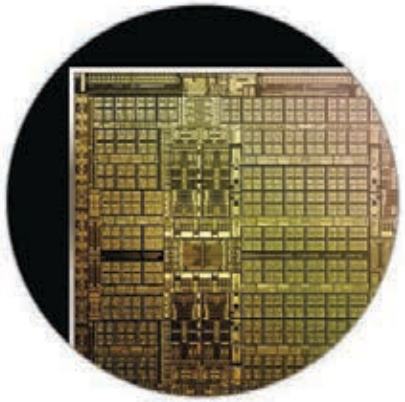
Inside the NVIDIA Ampere Architecture

Ronny Krashinsky, Olivier Giroux
GPU Architects

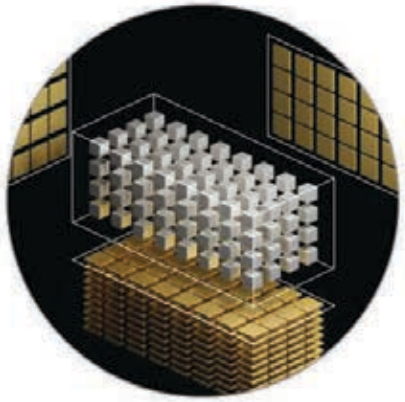
GTC 2020



UNPRECEDENTED ACCELERATION AT EVERY SCALE

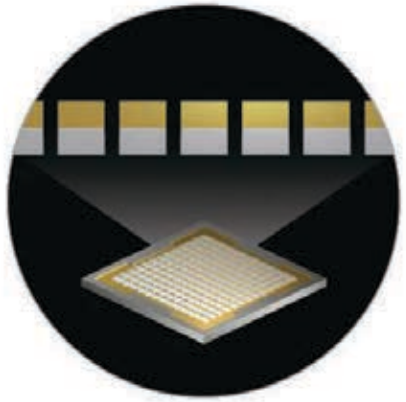


54 BILLION XTORS

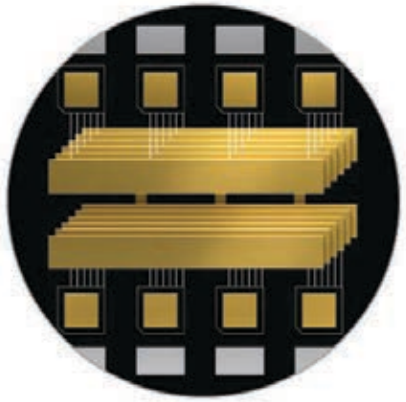


3rd GEN TENSOR CORES

SPARSITY ACCELERATION



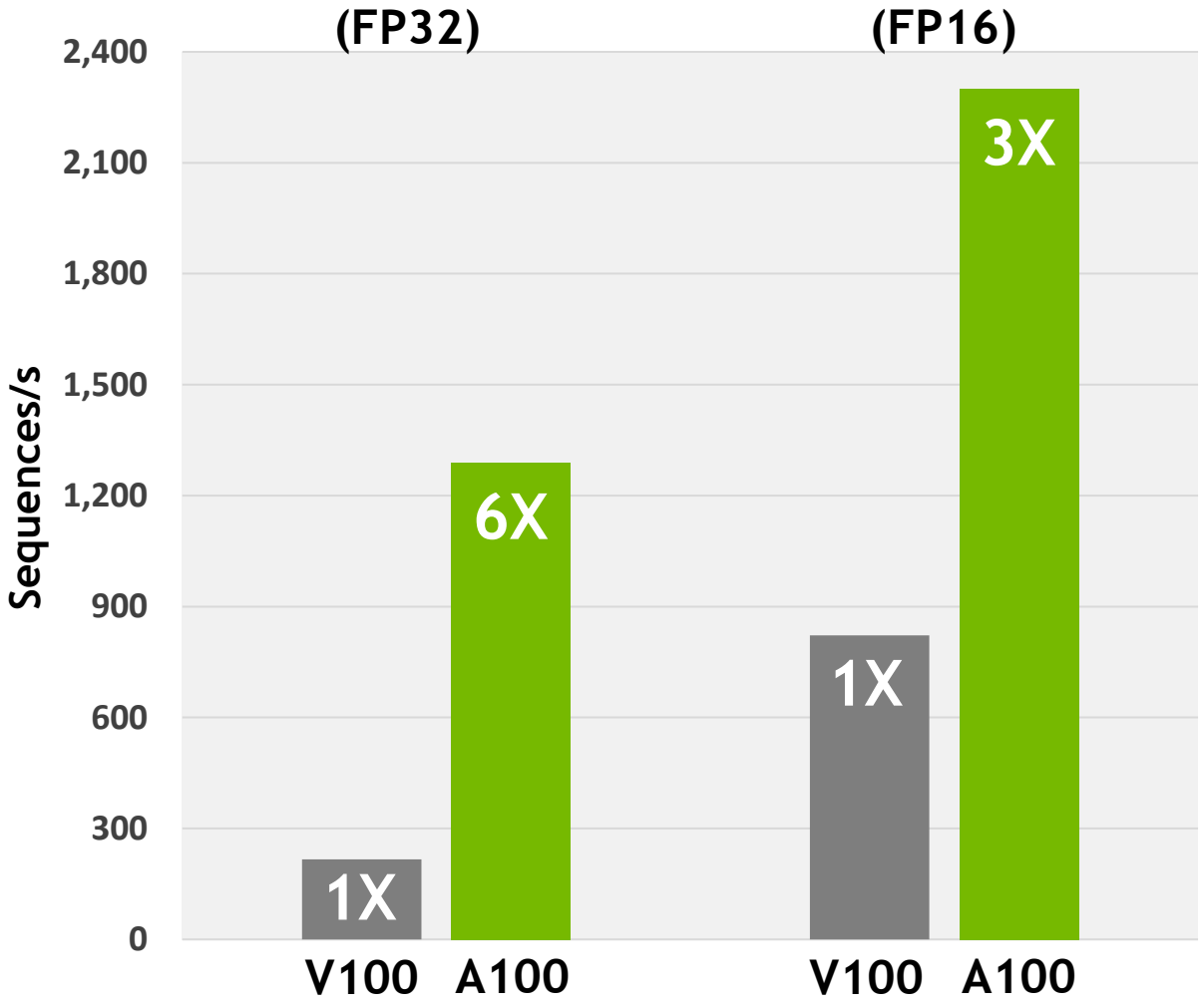
MIG



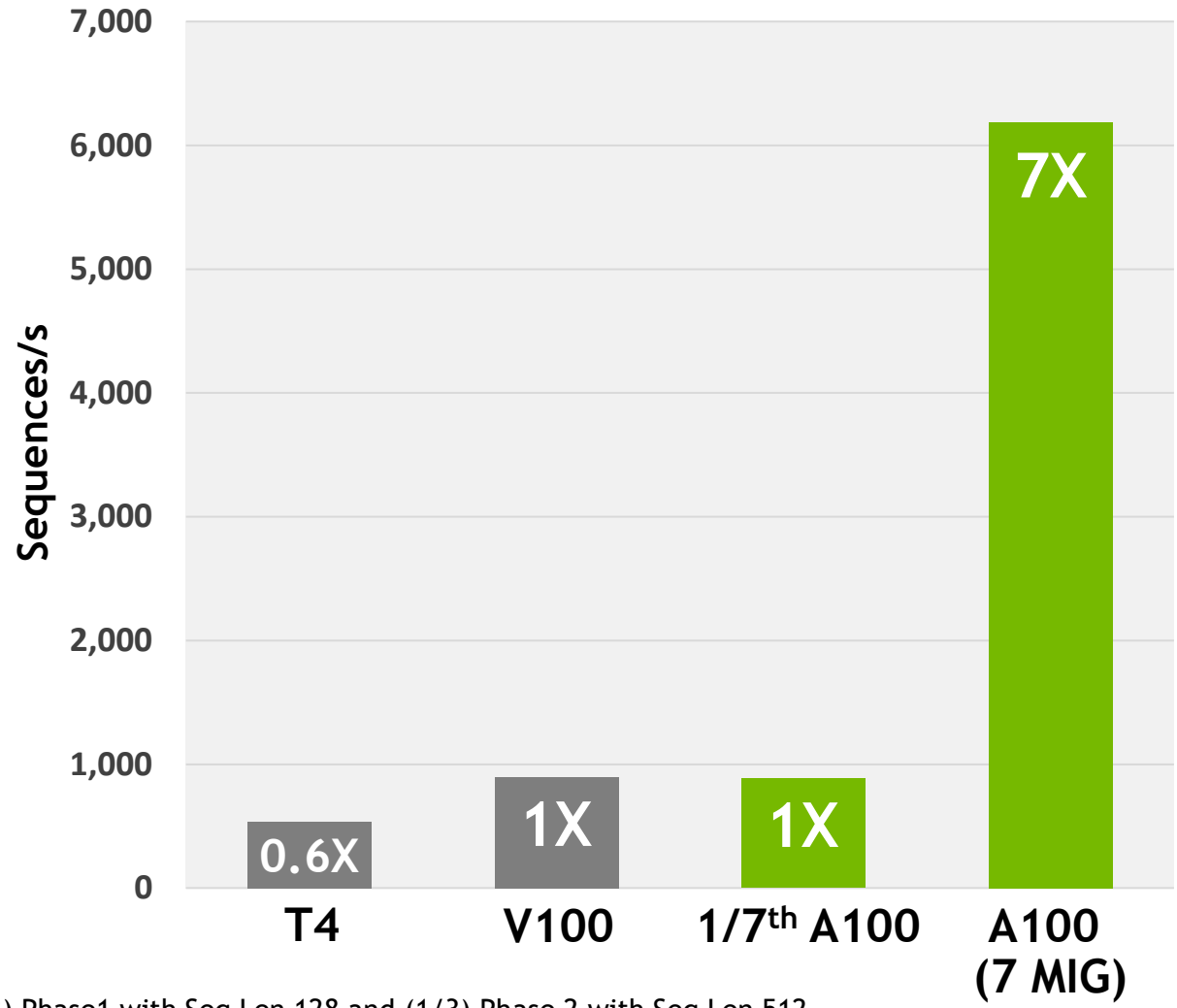
3rd GEN NVLINK & NVSWITCH

UNIFIED AI ACCELERATION

BERT-LARGE TRAINING

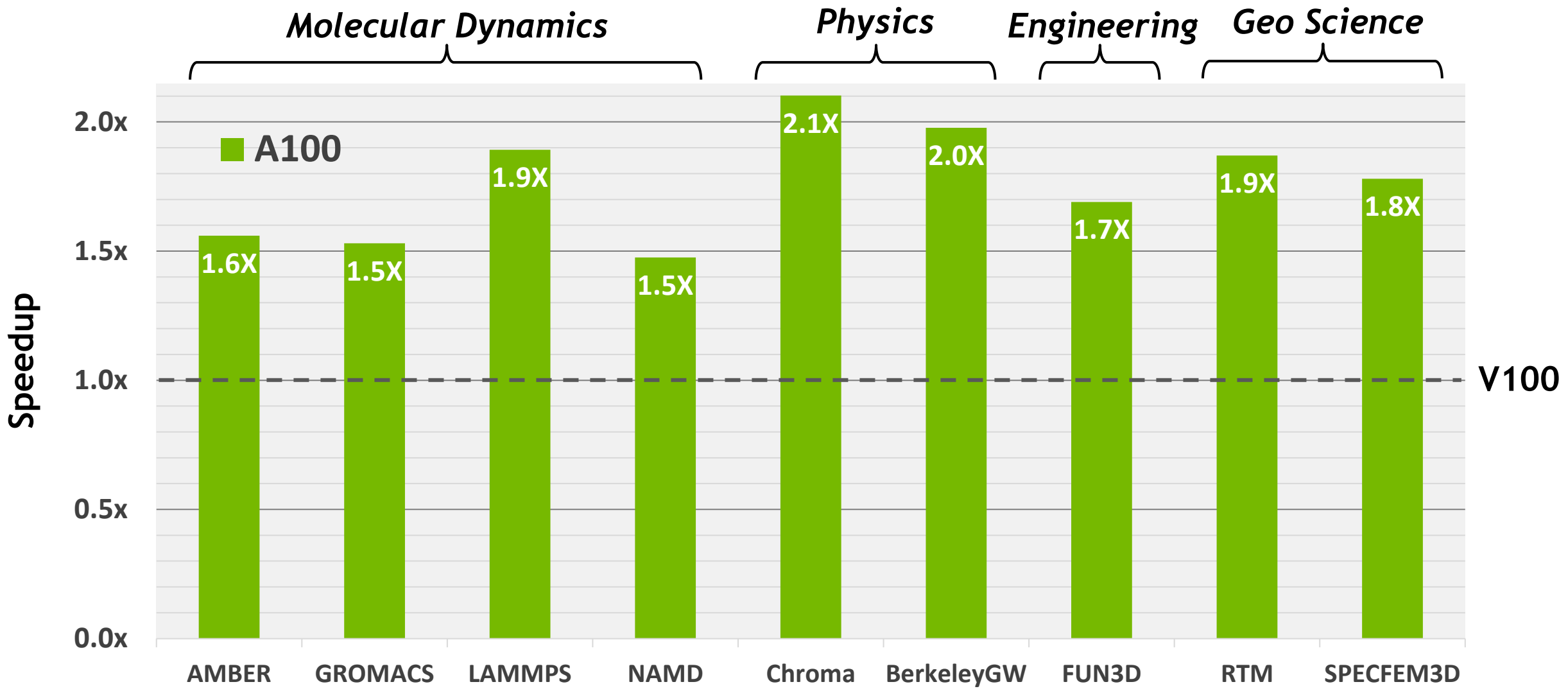


BERT-LARGE INFERENCE



All results are measured
BERT Large Training (FP32 & FP16) measures Pre-Training phase, uses PyTorch including (2/3) Phase1 with Seq Len 128 and (1/3) Phase 2 with Seq Len 512, V100 is DGX1 Server with 8xV100, A100 is DGX A100 Server with 8xA100, A100 uses TF32 Tensor Core for FP32 training
BERT Large Inference uses TRT 7.1 for T4/V100, with INT8/FP16 at batch size 256. Pre-production TRT for A100, uses batch size 94 and INT8 with sparsity

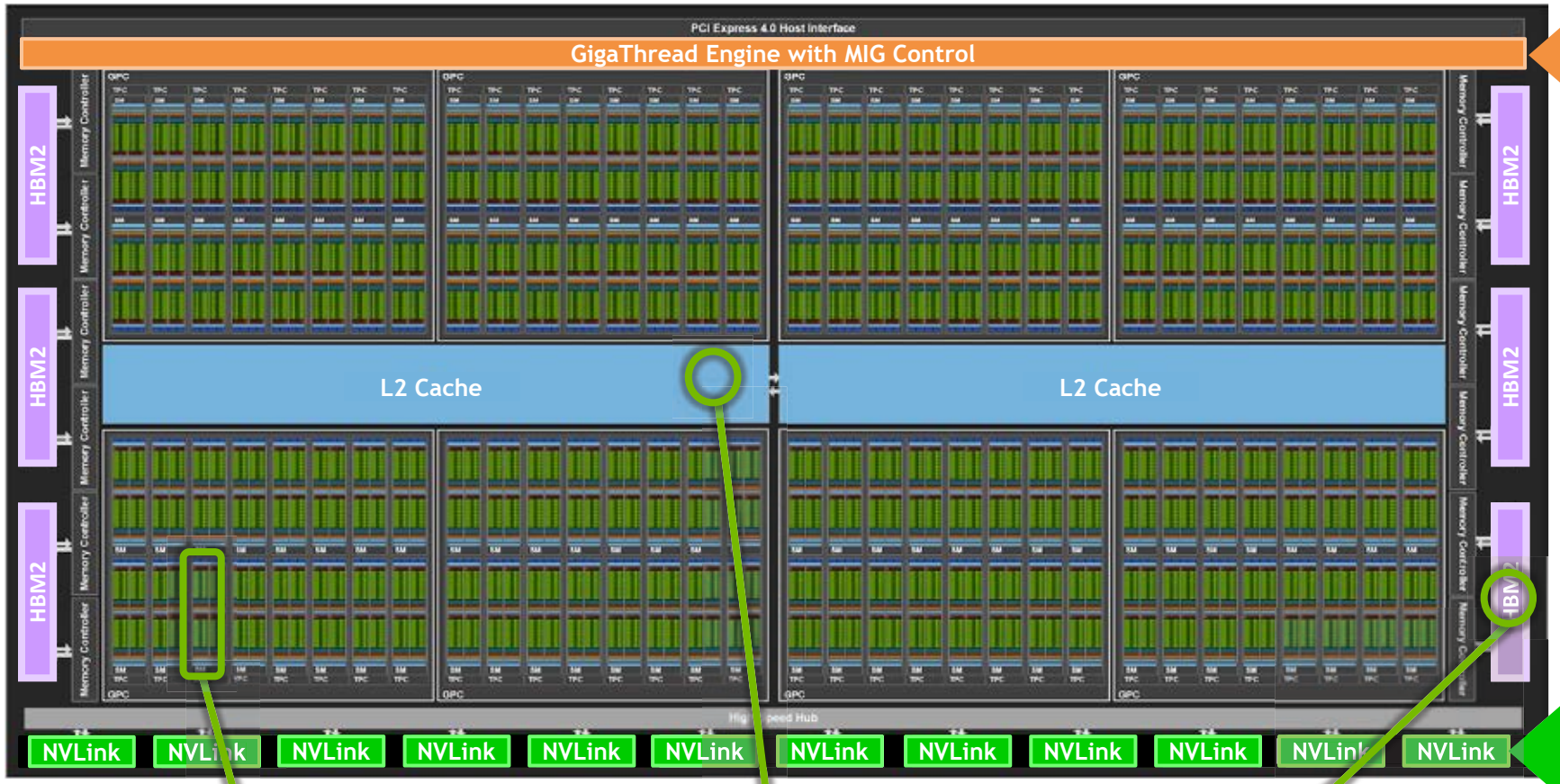
ACCELERATING HPC



All results are measured
Except BerkeleyGW, V100 used is single V100 SXM2. A100 used is single A100 SXM4
More apps detail: AMBER based on PME-Cellulose, GROMACS with STMV (h-bond), LAMMPS with Atomic Fluid LJ-2.5, NAMD with v3.0a1 STMV_NVE
Chroma with szscl21_24_128, FUN3D with dpw, RTM with Isotropic Radius 4 1024^3, SPECFEM3D with Cartesian four material model
BerkeleyGW based on Chi Sum and uses 8xV100 in DGX-1, vs 8xA100 in DGX A100

A100 TENSOR-CORE GPU

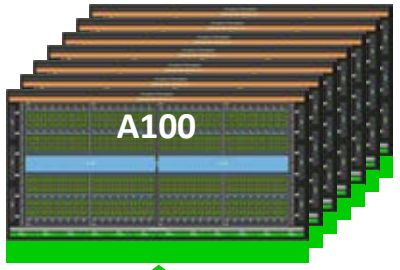
54 billion transistors in 7nm



7x

Scale OUT

Multi-Instance GPU



2x BW

Scale UP

3rd gen. NVLINK

108 SMs
6912 CUDA Cores

40MB L2
6.7x capacity

1.56 TB/s HBM2
1.7x bandwidth

A100 SM

Third-generation Tensor Core

Faster and more efficient
Comprehensive data types
Sparsity acceleration

Asynchronous data movement and synchronization

Increased L1/SMEM capacity


1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity


1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity

V100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA
V100	FP32	FP32	15.7	1x
	FP16	FP32	125	8x

V100

125 TOPS FF16/FP32 Mixed-precision

8x vs. FFMA

A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA
V100	FP32	FP32	15.7	1x
	FP16	FP32	125	8x
A100	FP32	FP32	19.5	1x
	FP16	FP32	312	16x

V100→A100

2.5x TOPS FF16/FP32 Mixed-precision

2x TOPS/SM

A100 TENSOR CORE

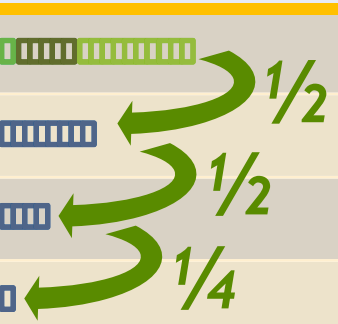
	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA
V100	FP32	FP32	15.7	1x
	FP16	FP32	125	8x
A100	FP32	FP32	19.5	1x
	TF32	FP32	156	8x
	FP16	FP32	312	16x
	BF16	FP32	312	16x

TF32 accelerates FP32 in/out data → 10x vs. V100 FP32

BFloat16 (BF16) at same rate as FP16

A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA
V100	FP32	FP32	15.7	1x
	FP16	FP32	125	8x
A100	FP32	FP32	19.5	1x
	TF32	FP32	156	8x
	FP16	FP32	312	16x
	BF16	FP32	312	16x
	FP16	FP16	312	16x
	INT8	INT32	624	32x
	INT4	INT32	1248	64x
	BINARY	INT32	4992	256x



TOPS track operand width

Inference data types

A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-

With Sparsity *another* 2x, INT8/INT4 reach *petaops*

A100 TENSOR CORE

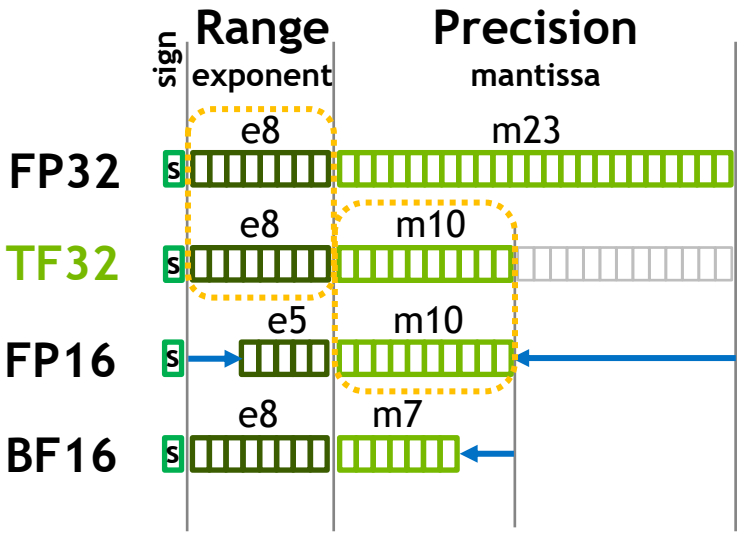
	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x		
	IEEE FP64		19.5	1x		

V100→A100
2.5x FLOPS for HPC

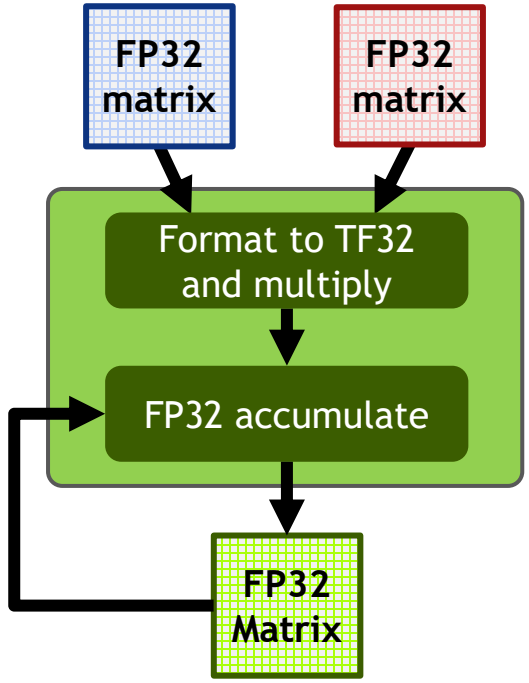
A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-
	IEEE FP64		19.5	1x	-	-

INSIDE A100 TensorFloat-32 (TF32)



Range of FP32 with precision of FP16



FP32 input/output

FP32 storage and math for all activations, gradients, …
everything outside tensor cores

Out-of-the-box tensor core acceleration for DL

Easy step towards maximizing tensor core performance with mixed-precision (FP16, BF16)

Up to 4x speedup on linear solvers for HPC

→S22082: Mixed-Precision Training of Neural Networks, 5/20 2:45pm PDT

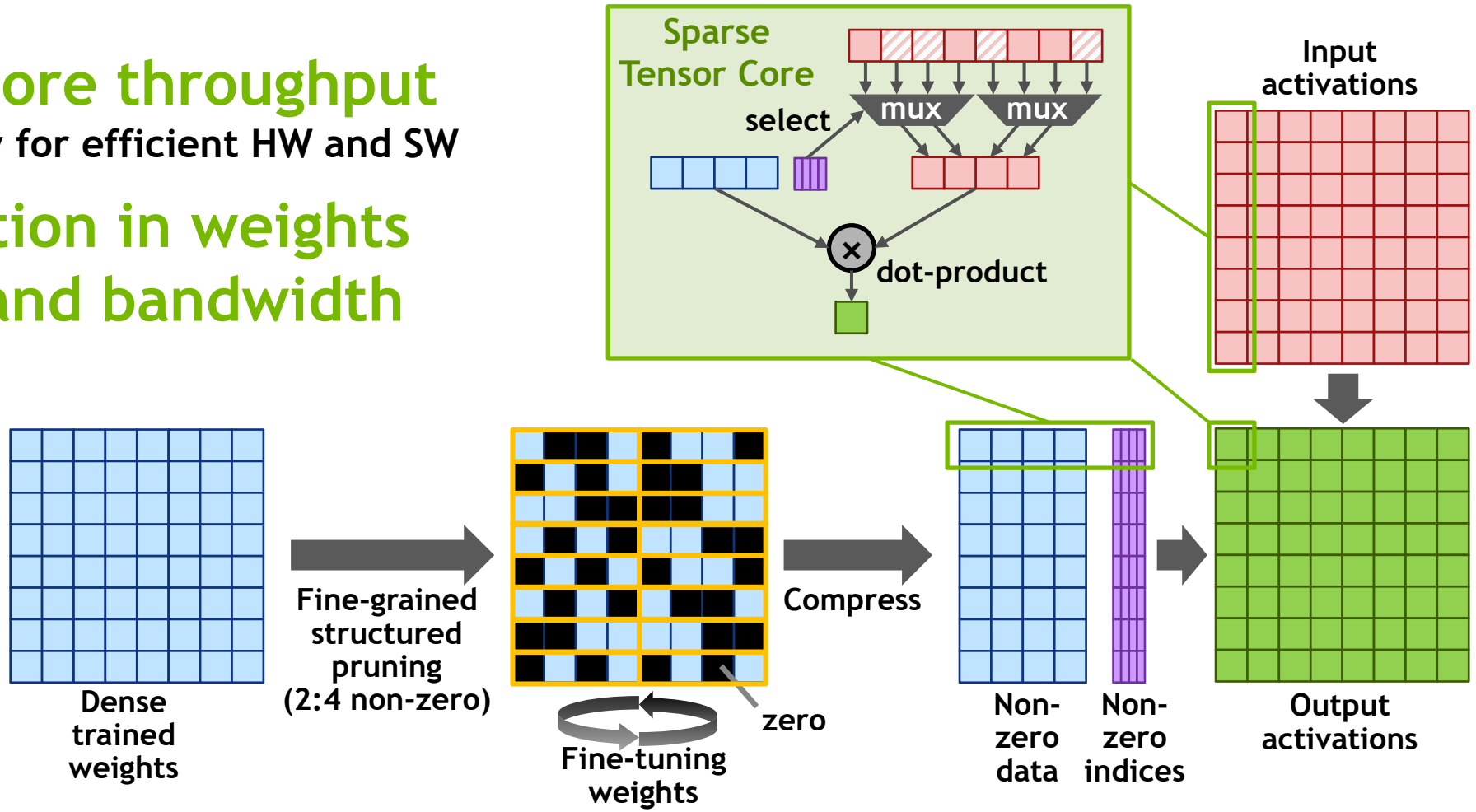
→S21681: How CUDA Math Libraries can help you unleash the power of the new NVIDIA A100 GPU (recording available)

INSIDE A100 SPARSE TENSOR CORE

2x Tensor Core throughput

Structured-sparsity for efficient HW and SW

~2x reduction in weights footprint and bandwidth



~No loss in inferencing accuracy

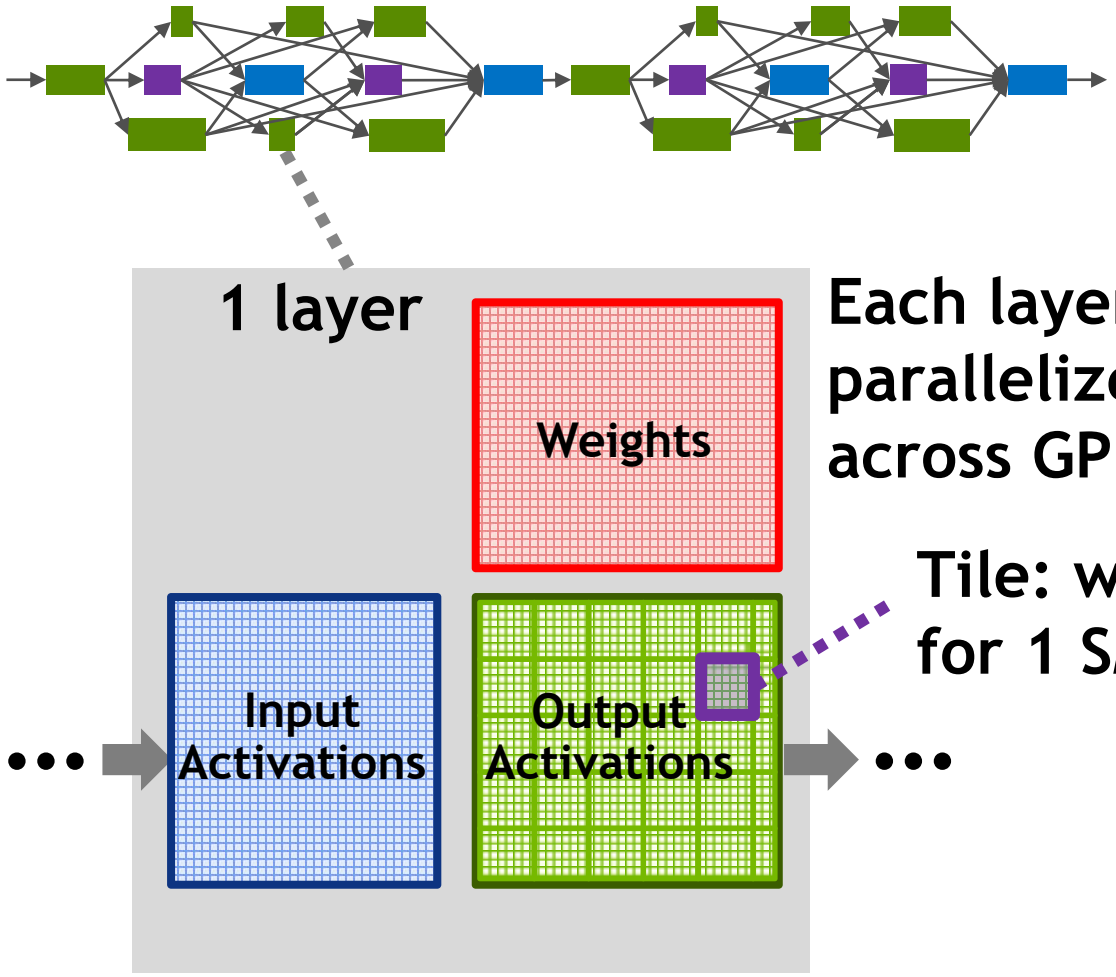
Evaluated across dozens of networks: vision, object detection, segmentation, natural language modeling, translation

→ S22085: Accelerating Sparsity in the NVIDIA Ampere Architecture, 5/20 1:30pm PDT

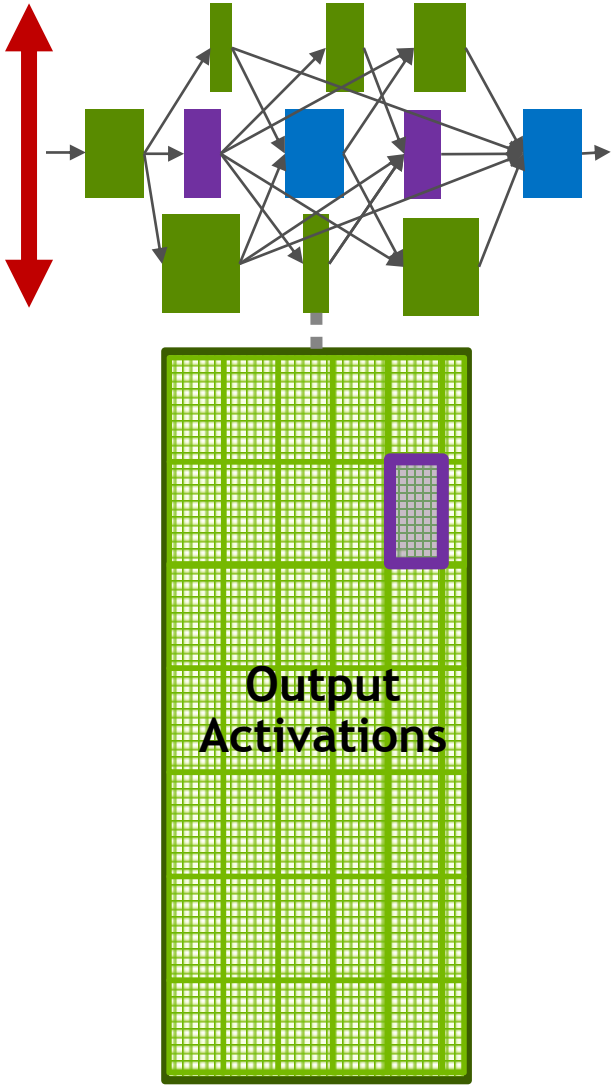
1. New Tensor Core
2. **Strong Scaling**
3. Elastic GPU
4. Productivity

DL STRONG SCALING

DL networks:
Long chains of sequentially-dependent compute-intensive layers

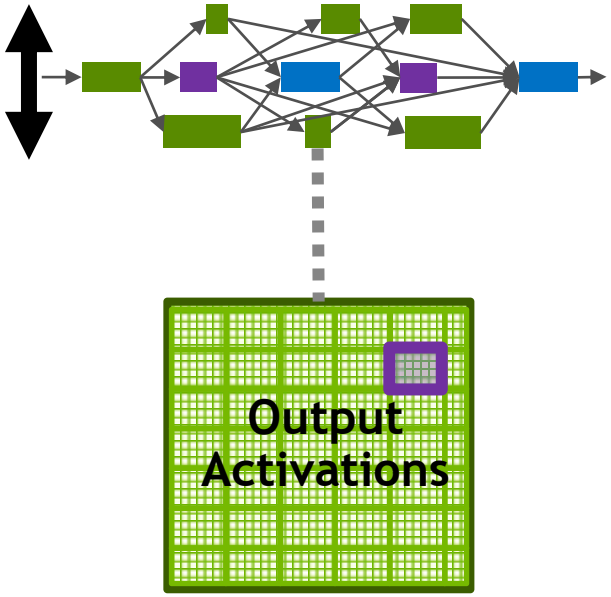


Weak scaling



~2.5x larger network runs in same time

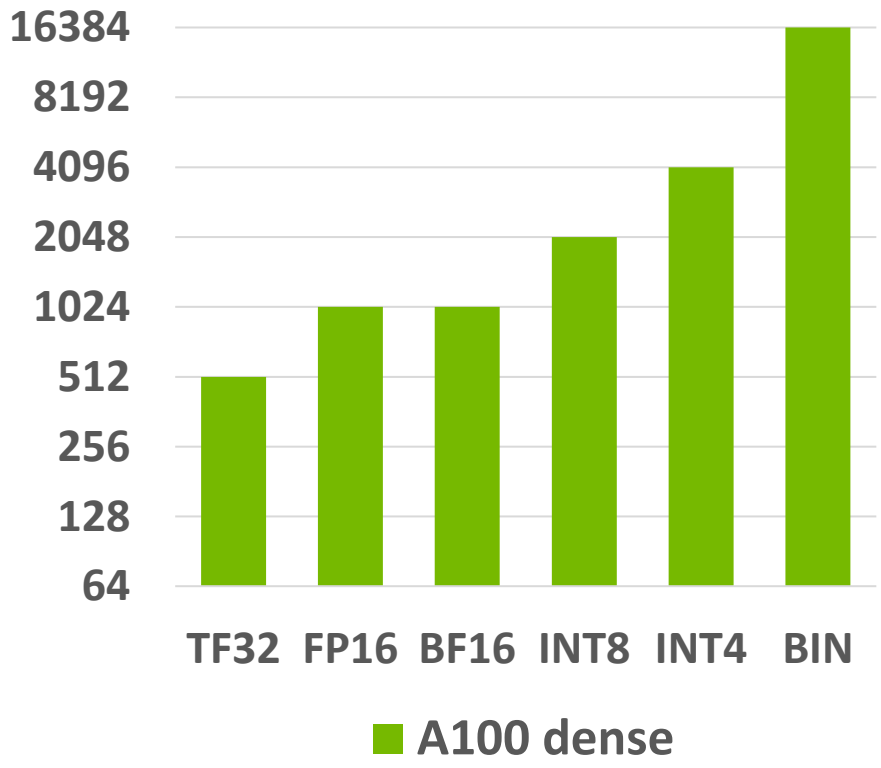
Strong scaling



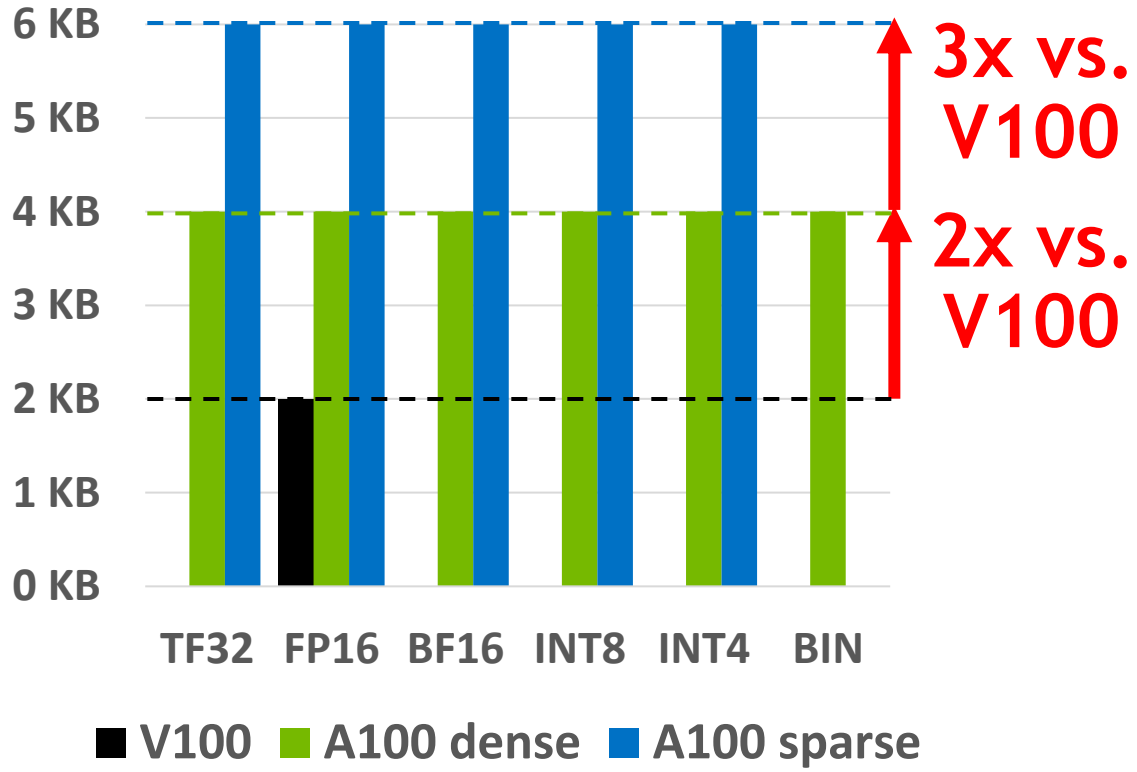
Fixed network runs ~2.5x faster

HOW TO KEEP TENSOR CORES FED?

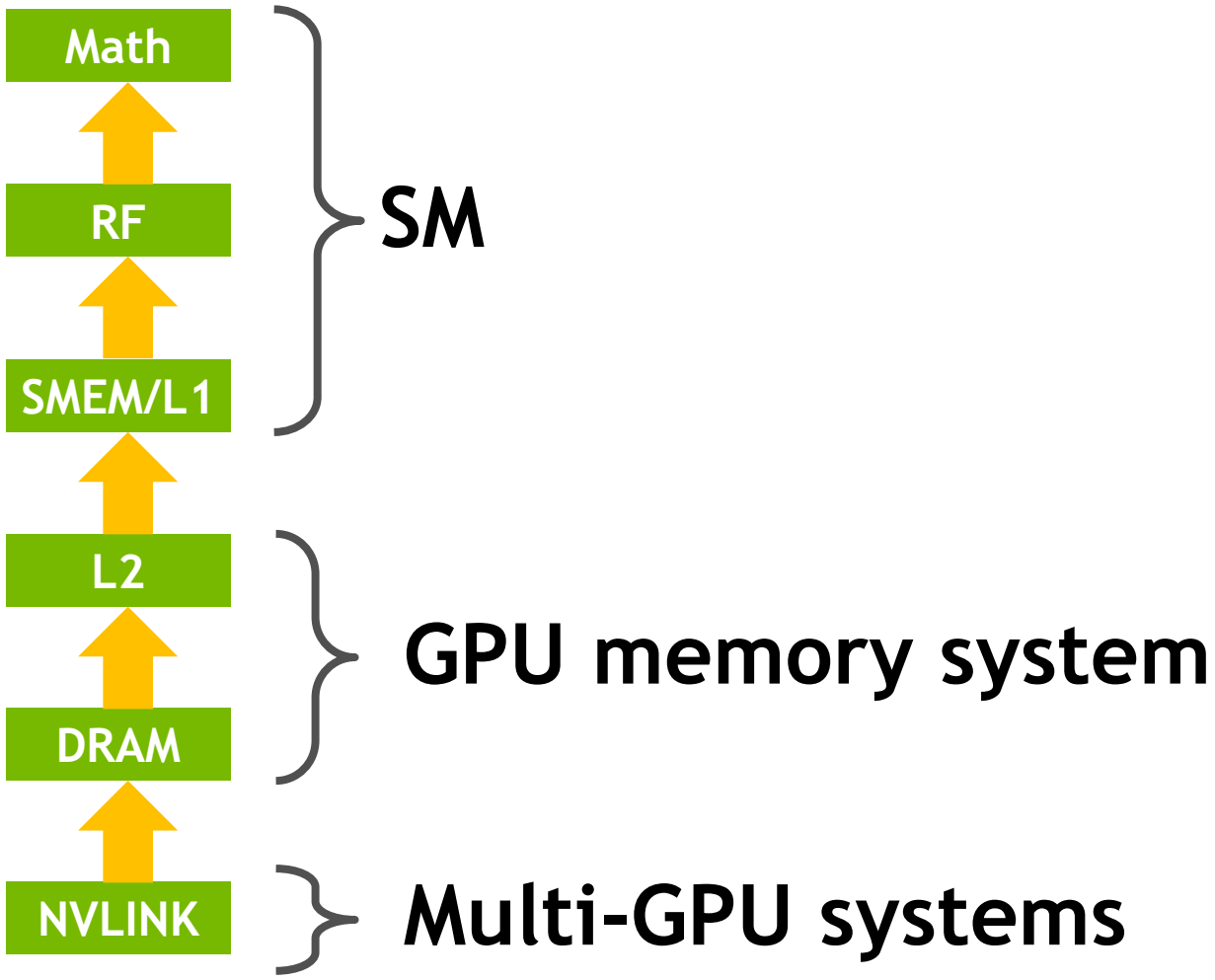
Math bandwidth
(MACs/clock/SM)



Required data bandwidth
(A+B operands, B/clock/SM)



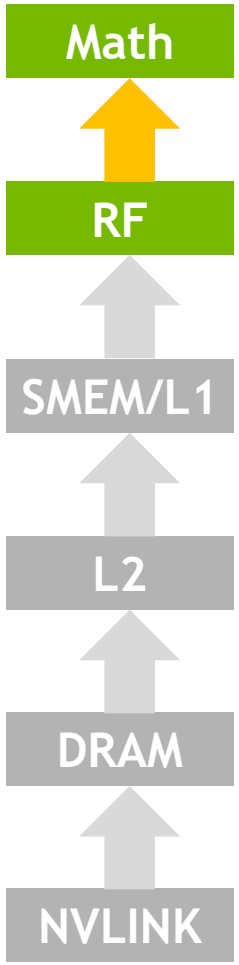
A100 STRONG SCALING INNOVATIONS



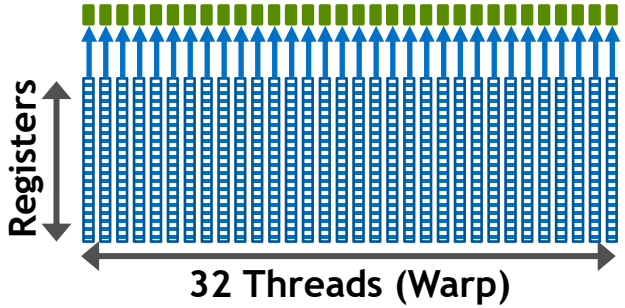
Improve speeds & feeds and efficiency across all levels of compute and memory hierarchy

A100 TENSOR CORE

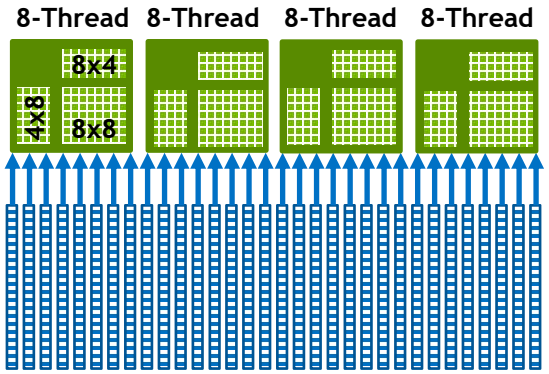
2x throughput vs. V100, >2x efficiency



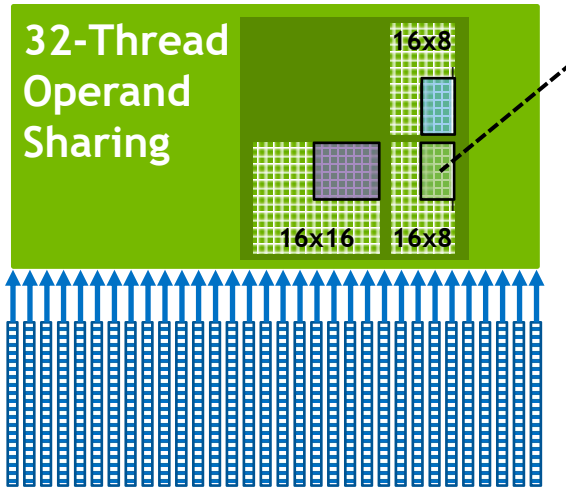
FFMA
(32 MACs, 2 cycles)



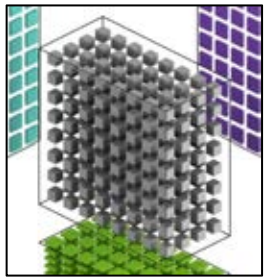
V100 TC Instruction
(1024 MACs, 8 cycles)



A100 TC Instruction
(2048 MACs, 8 cycles)



A100 TC
(1 cycle)

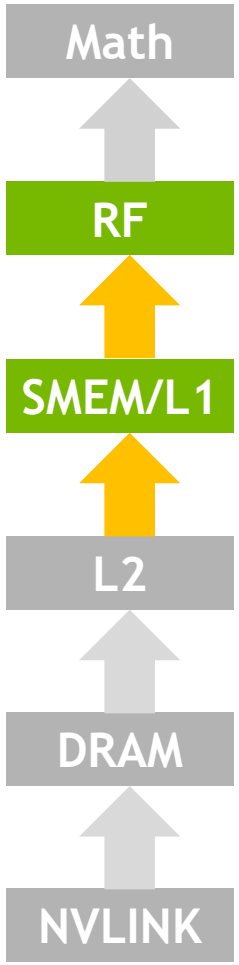


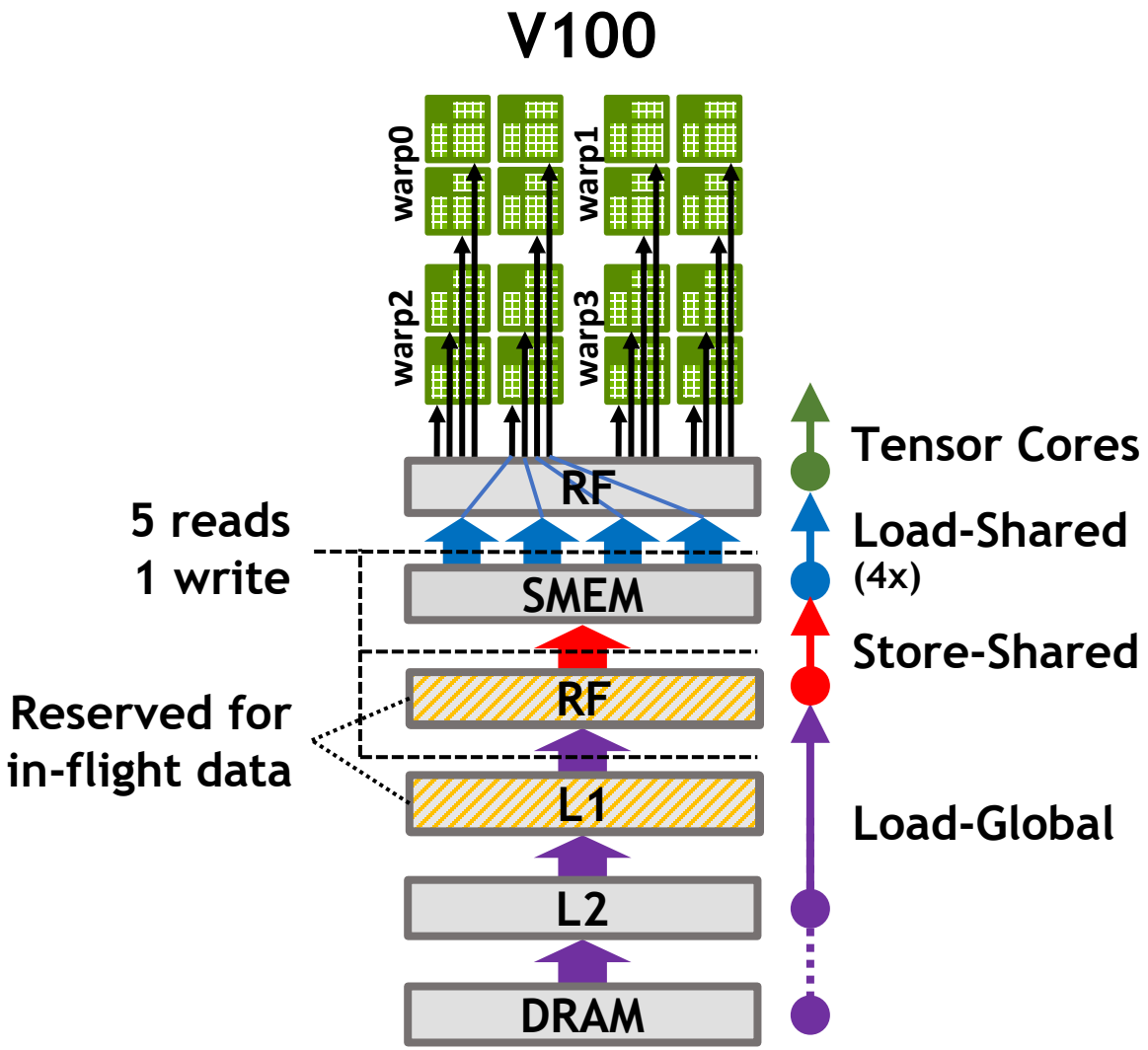
16x16x16 matrix multiply	FFMA	V100 TC	A100 TC	A100 vs. V100 (improvement)	A100 vs. FFMA (improvement)
Thread sharing	1	8	32	4x	32x
Hardware instructions	128	16	2	8x	64x
Register reads+writes (warp)	512	80	28	2.9x	18x
Cycles	256	32	16	2x	16x

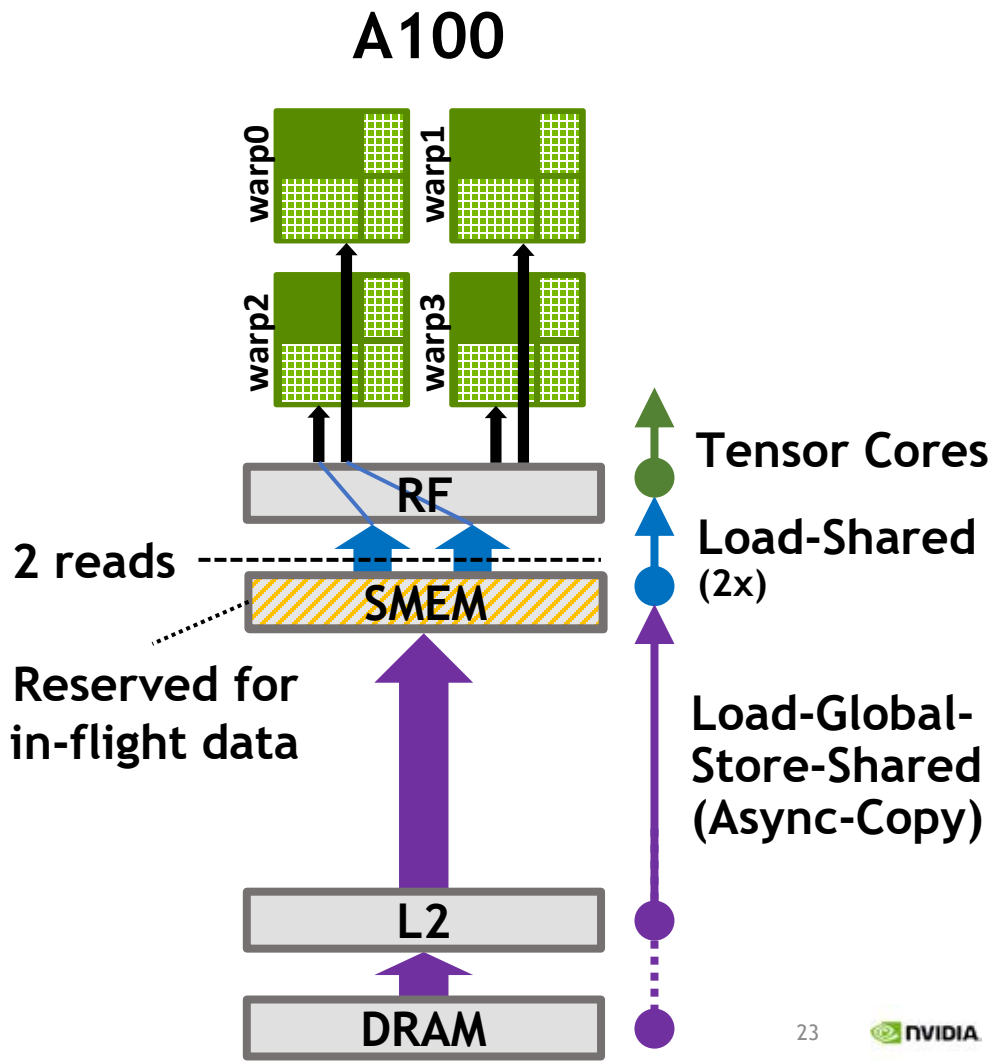
Tensor Cores assume FP16 inputs with FP32 accumulator, V100 Tensor Core instruction uses 4 hardware instructions

A100 SM DATA MOVEMENT EFFICIENCY

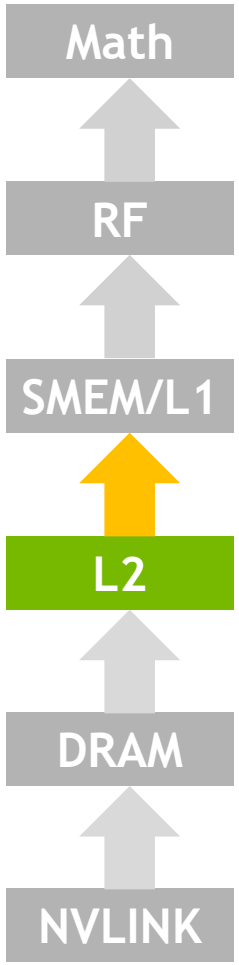
3x SMEM/L1 bandwidth, 2x in-flight capacity

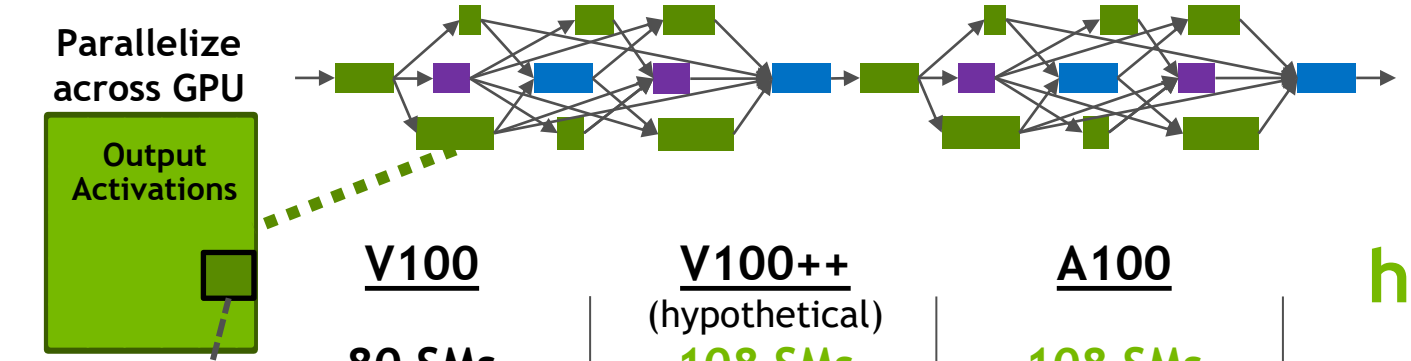






A100 L2 BANDWIDTH

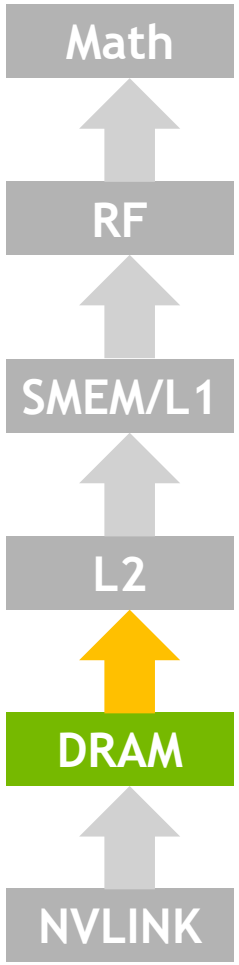




	V100	V100++ (hypothetical)	A100
	80 SMs V100 TC 64 L2 slices 32 B/clk/slice	108 SMs A100 TC 64 L2 slices 32 B/clk/slice	108 SMs A100 TC 80 L2 slices 64 B/clk/slice
	12 B/clk/SM 47%	24 B/clk/SM 127%	24 B/clk/SM 51%
	16 B/clk/SM 63%	32 B/clk/SM 169%	32 B/clk/SM 68%
	24 B/clk/SM 94%	48 B/clk/SM 253%	48 B/clk/SM 101%

Split L2 with hierarchical crossbar - 2.3x increase in bandwidth over V100, lower latency

A100 DRAM BANDWIDTH



Faster HBM2

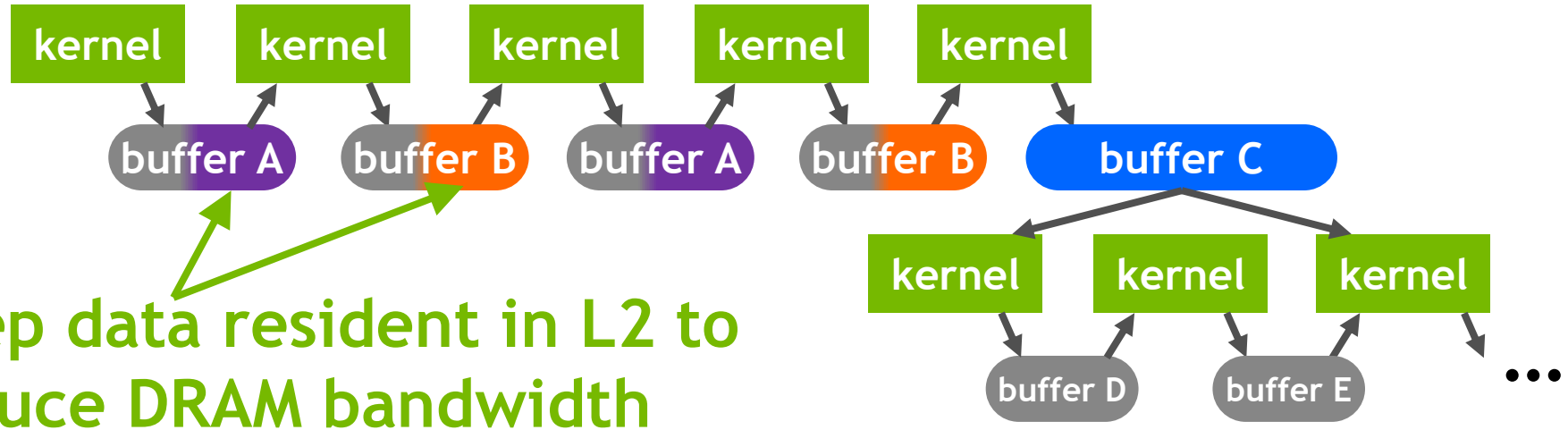
25% more pins, 38% faster clocks

→ 1.6 TB/s, 1.7x vs. V100

Larger and smarter L2

40MB L2, 6.7x vs. V100

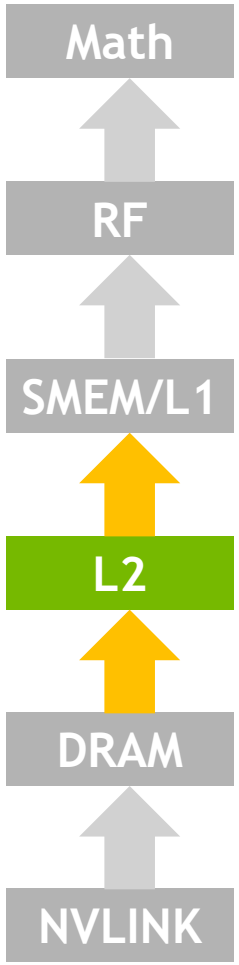
L2-Residency controls



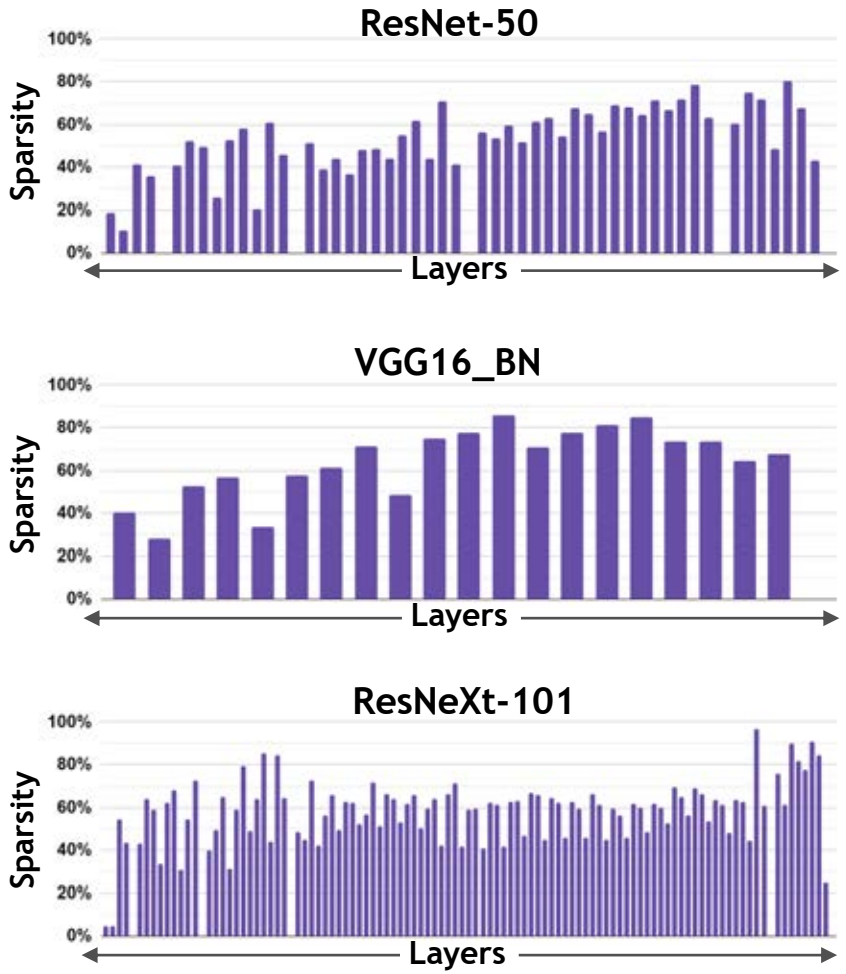
Keep data resident in L2 to reduce DRAM bandwidth

→ S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT

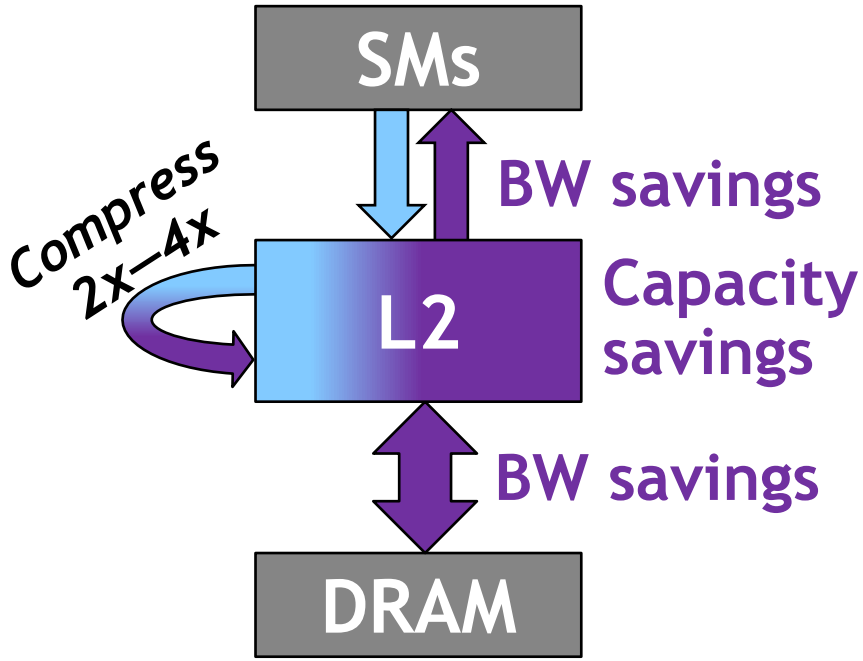
A100 COMPUTE DATA COMPRESSION



Activation sparsity due to ReLU



Up to 4x DRAM+L2 bandwidth and 2x L2 capacity for fine-grained unstructured sparsity



→ S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT

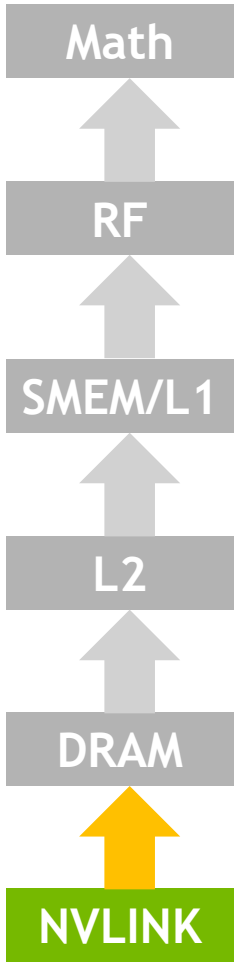
A100 NVLINK BANDWIDTH

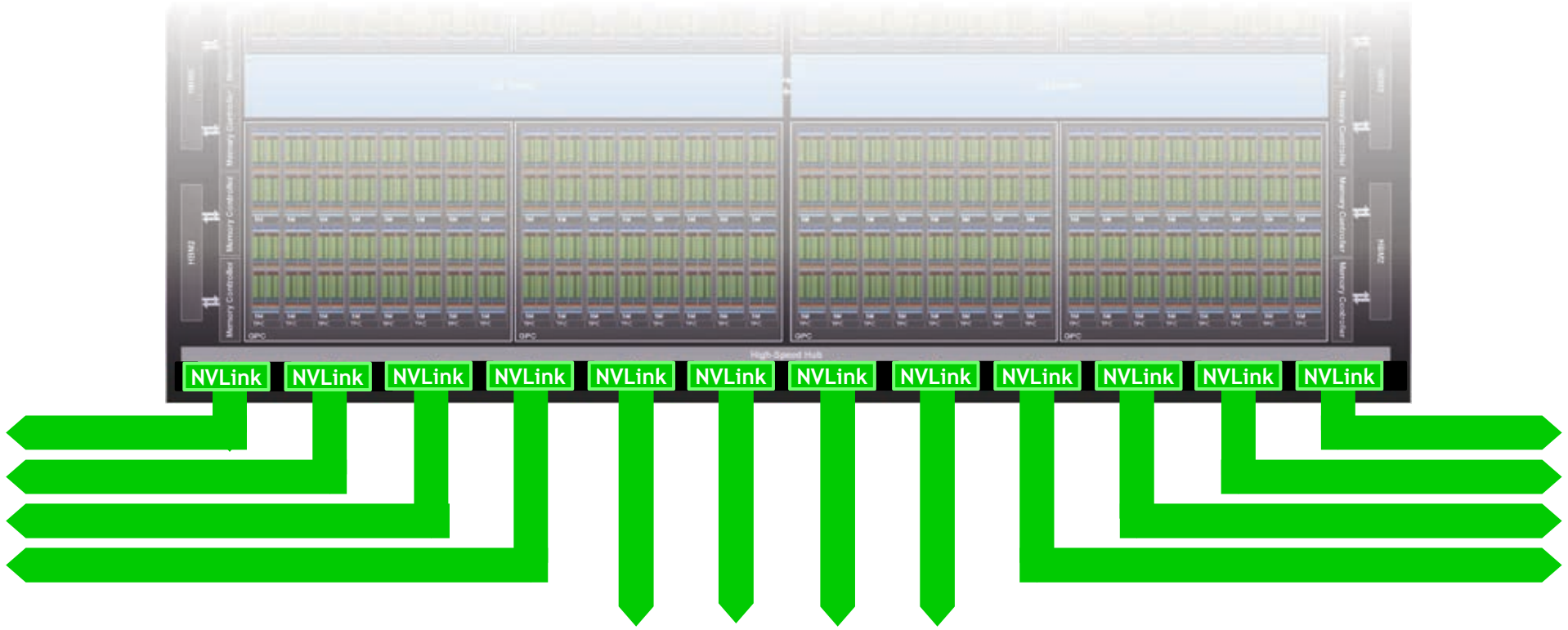
Third Generation NVLink

50 Gbit/sec per signal pair

12 links, 25 GB/s in/out, 600 GB/s total

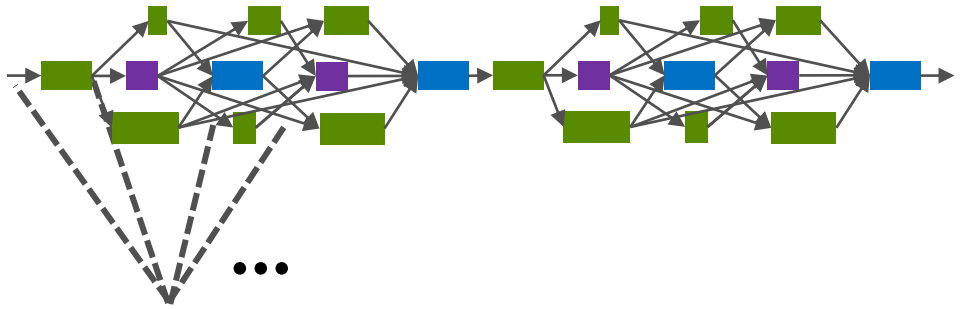
2x vs. V100





→S21884: Under the Hood of the new DGX A100 System Architecture (recording available soon)

A100 ACCELERATES CUDA GRAPHS

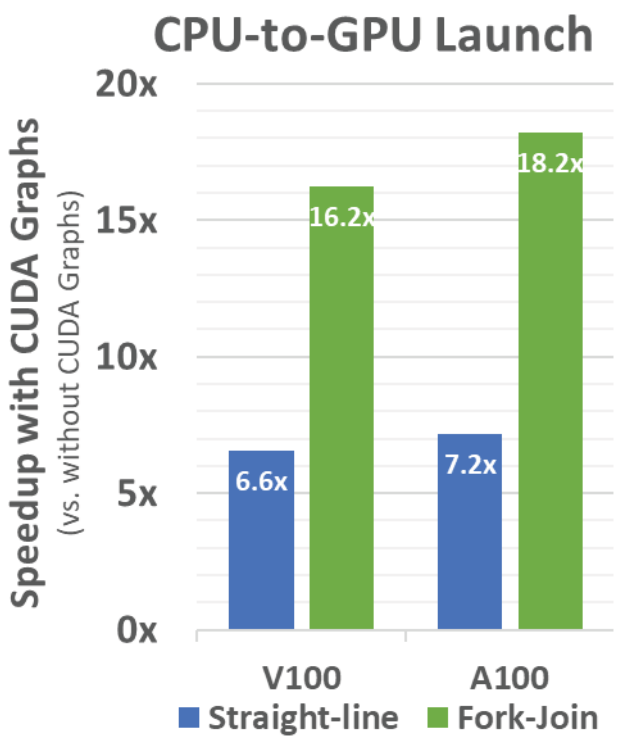


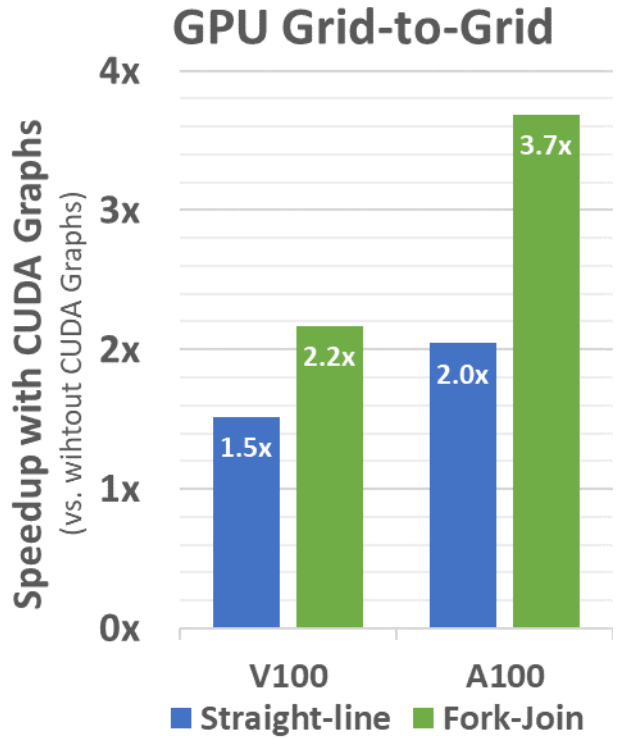
Grid launches:

- *CPU-to-GPU*
- *GPU grid-to-grid*

With strong scaling CPU and grid launch overheads become increasingly important

(Amdahl's law)





32-node graphs of empty grids, DGX1-V, DGX-A100

One-shot CPU-to-GPU graph submission and graph reuse

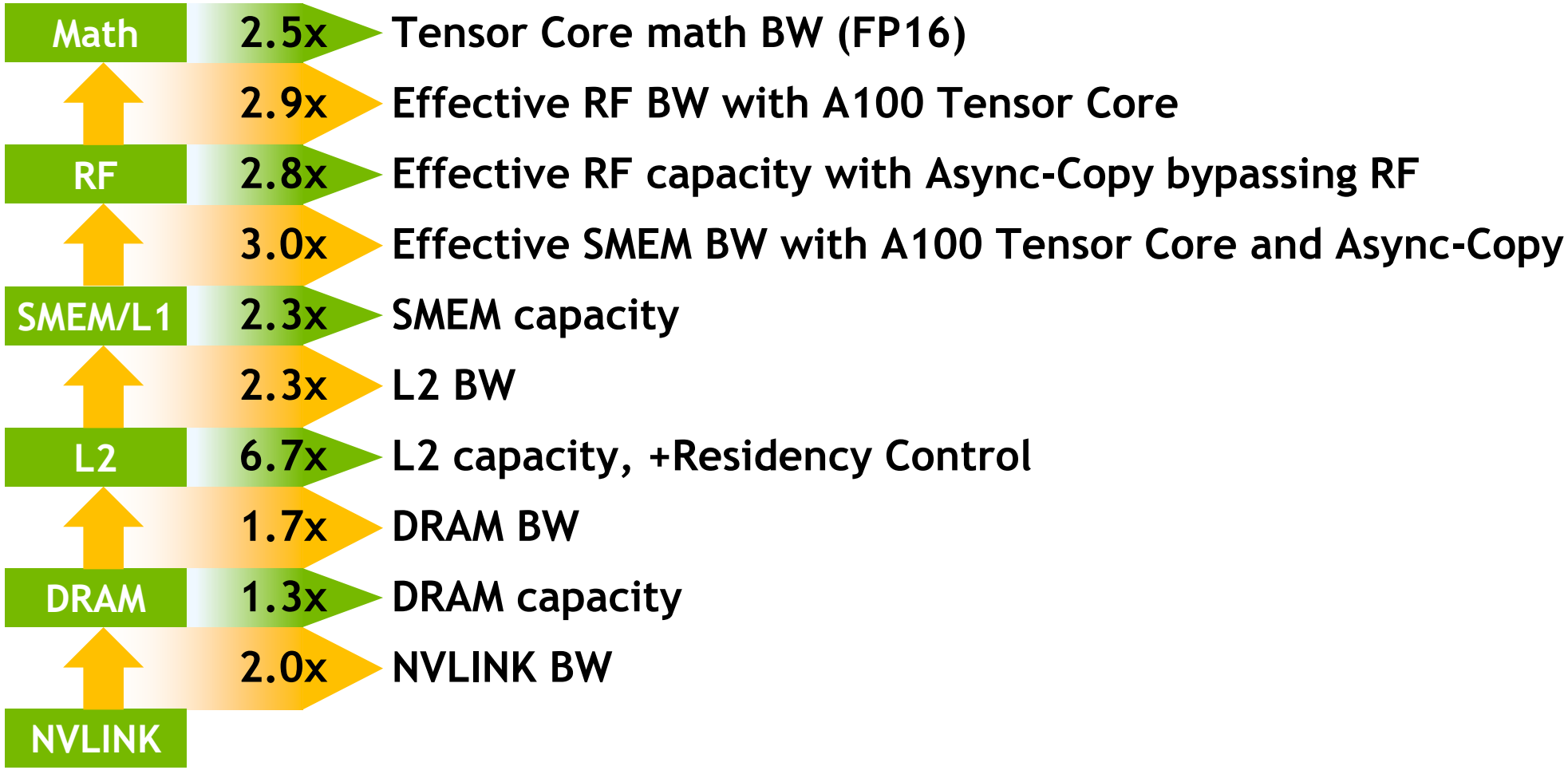
Microarchitecture improvements for grid-to-grid latencies

→S21760: CUDA New Features And Beyond, 5/19 10:15am PDT

A100 STRONG SCALING INNOVATIONS

Delivering unprecedented levels of performance

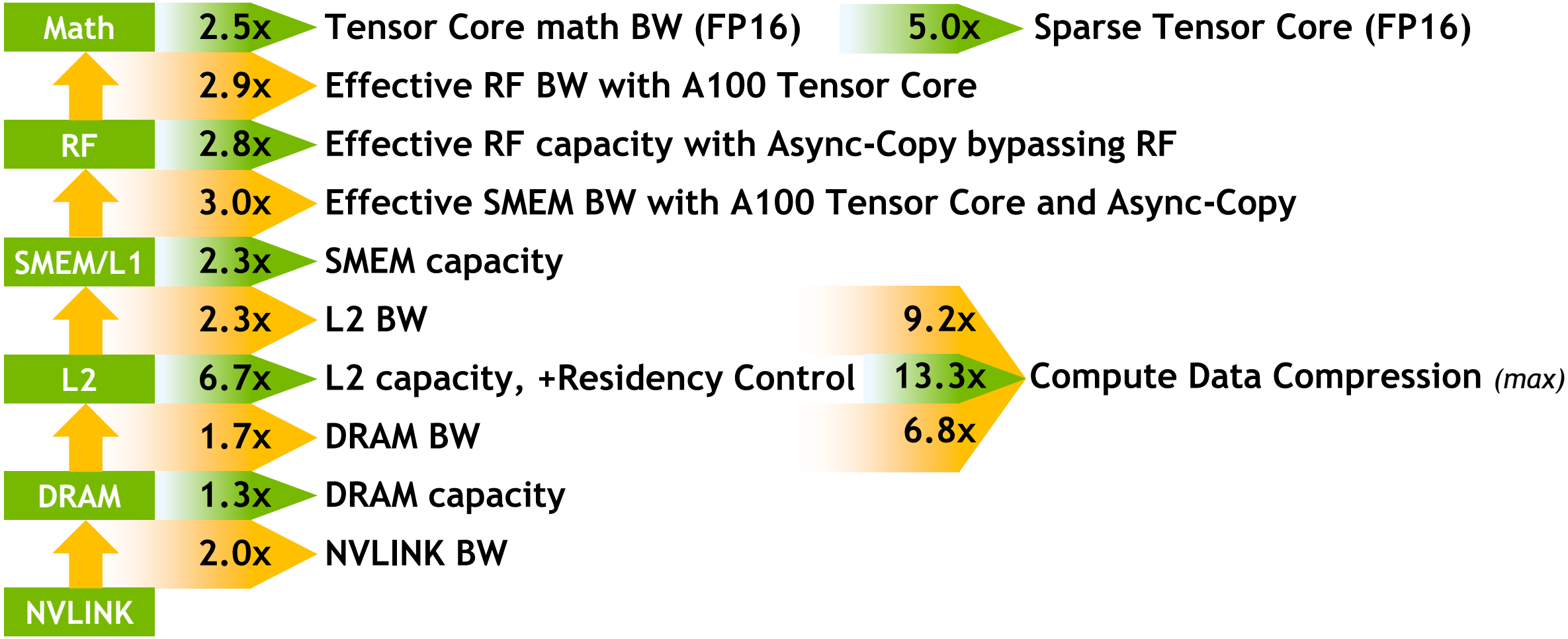
A100 improvements over V100



A100 STRONG SCALING INNOVATIONS

Delivering unprecedented levels of performance

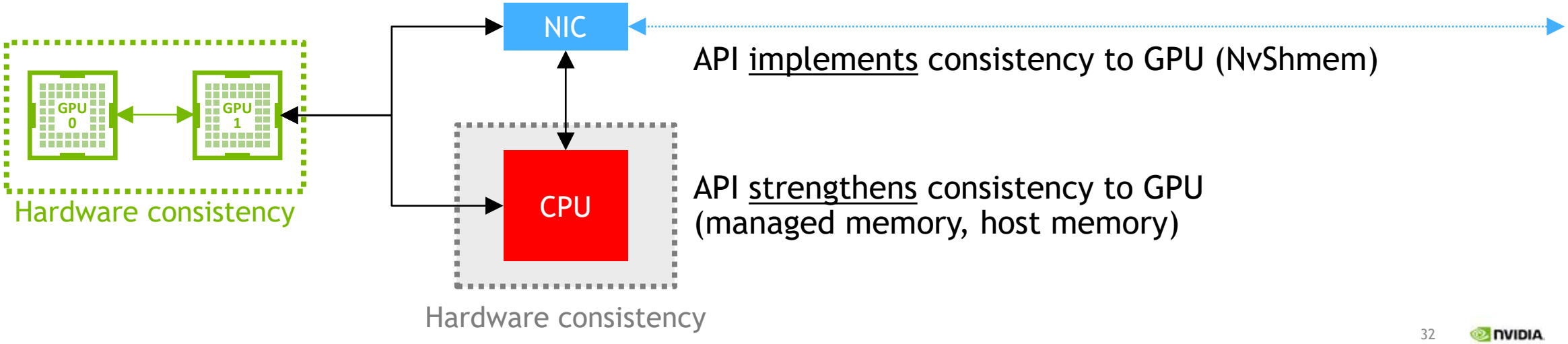
A100 improvements over V100



1. New Tensor Core
2. Strong Scaling
3. **Elastic GPU**
4. Productivity

NVLINK: ONE BIG GPU

- **InfiniBand/Ethernet**: travels a long distance, consistency is the responsibility of software
- **PCI Express**: hardware consistency for I/O, not for programming language memory models
- **NVLINK**: hardware consistency for programming language memory models, like system bus

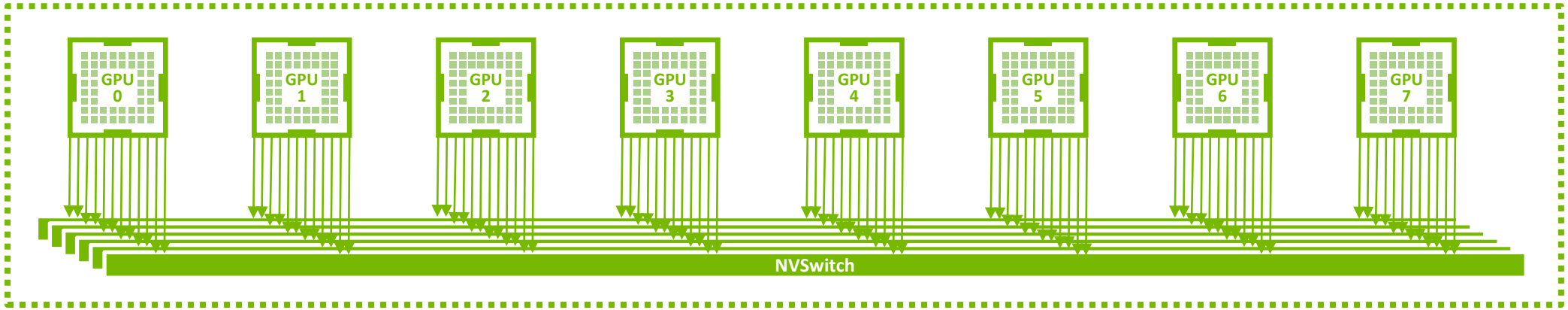


HGX A100: 3RD GEN NVLINK

- **HGX A100 4-GPU:** fully-connected system with 100GB/s all-to-all BW

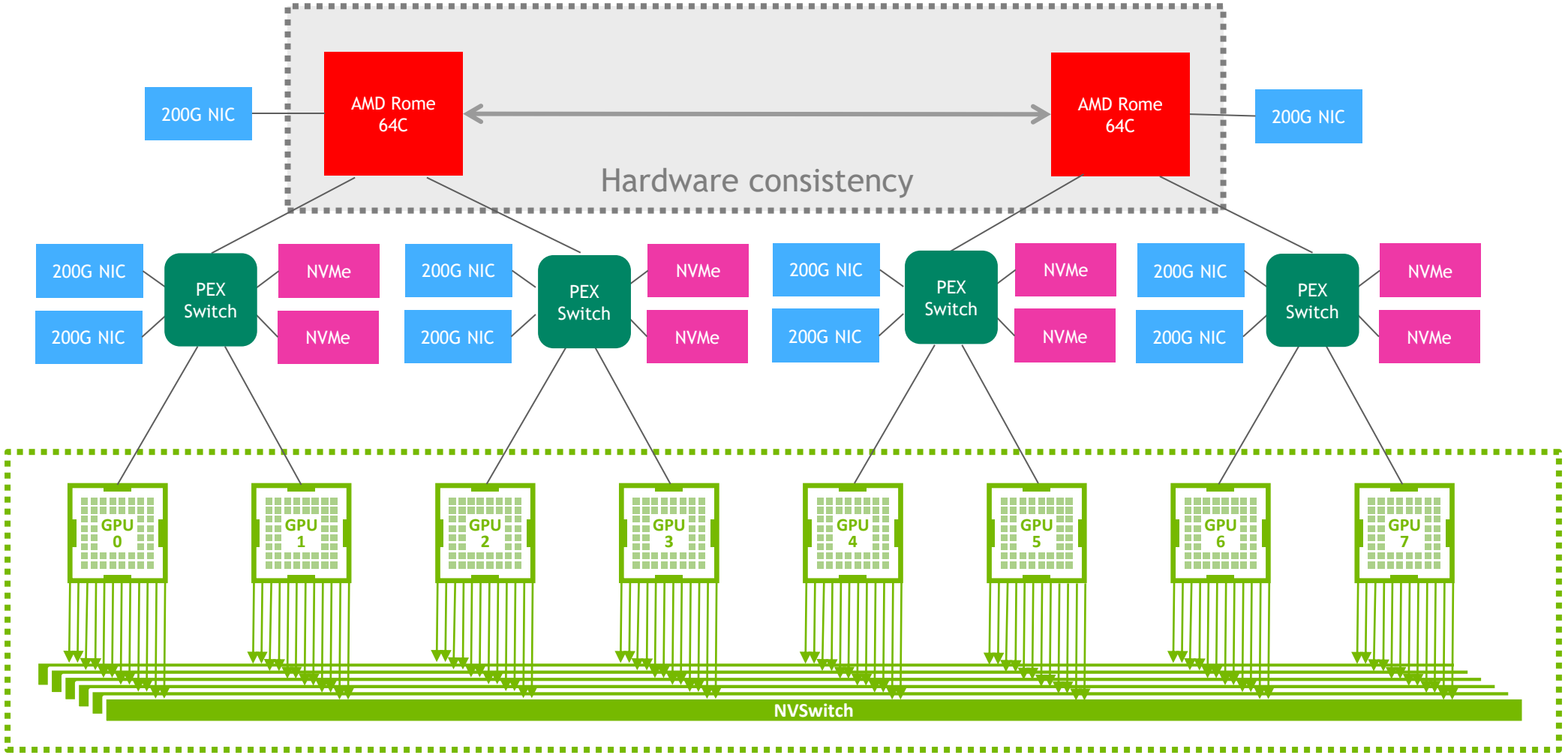
HGX A100: 3RD GEN NVLINK & SWITCH

- **HGX A100 4-GPU**: fully-connected system with 100GB/s all-to-all BW
- **New NVSwitch**: 6B transistors in TSMC 7FF, 36 ports, 25GB/s each, per direction
- **HGX A100 8-GPU**: 6x NVSwitch in a fat tree topology, 2.4TB/s full-duplex bandwidth



Hardware consistency

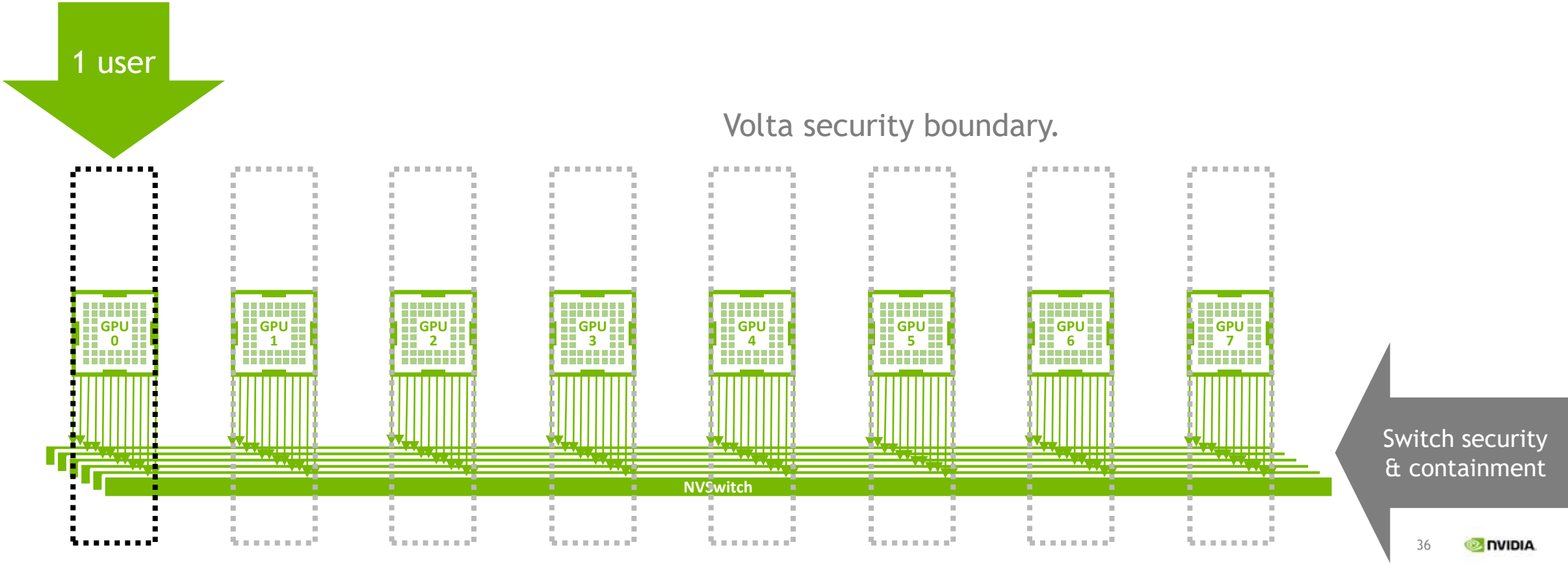
DGX A100: PCIE4 CONTROL & I/O



Hardware consistency

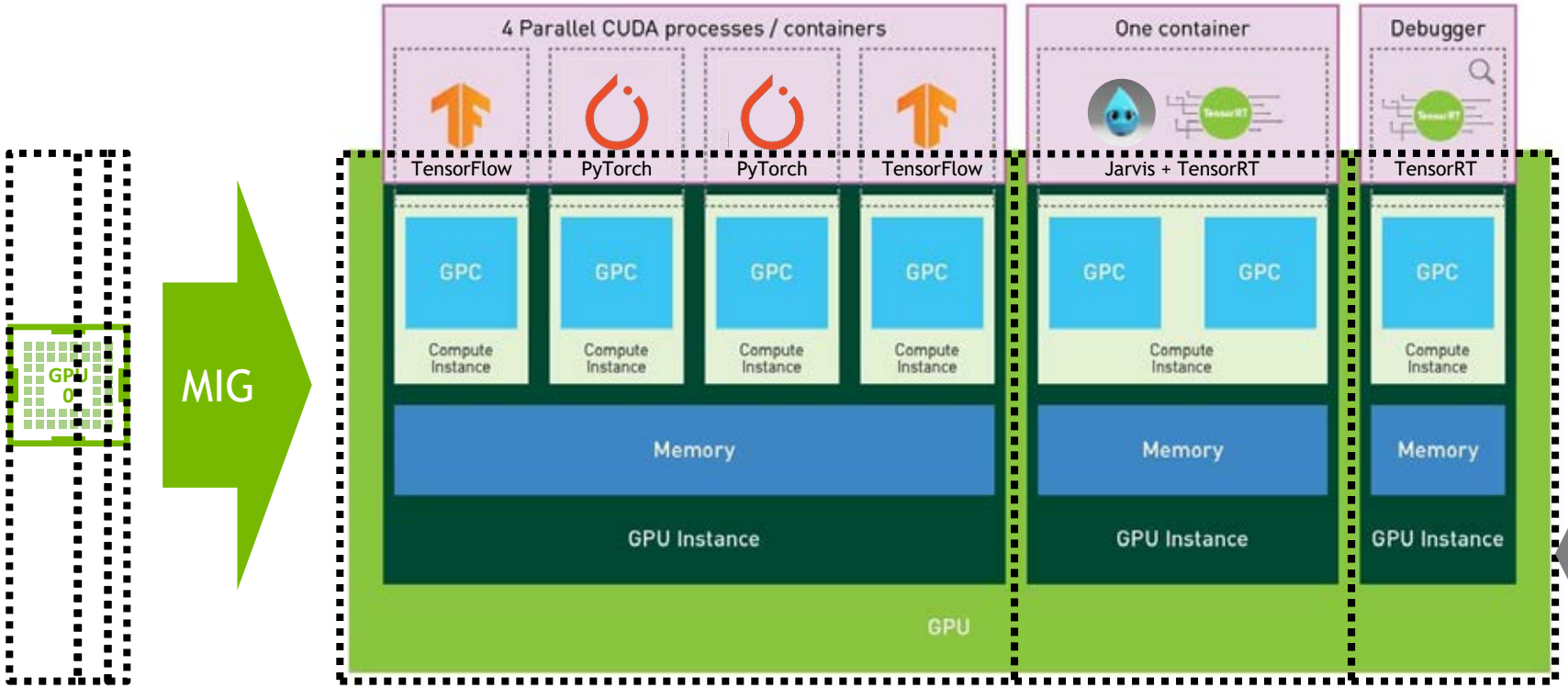
CLOUD SMALL INSTANCE USAGE

- Small workloads can under-utilize GPU cloud instances, provisioned at whole GPU level
- CSPs can't use MPS for GPU space-sharing, because it doesn't provide enough isolation



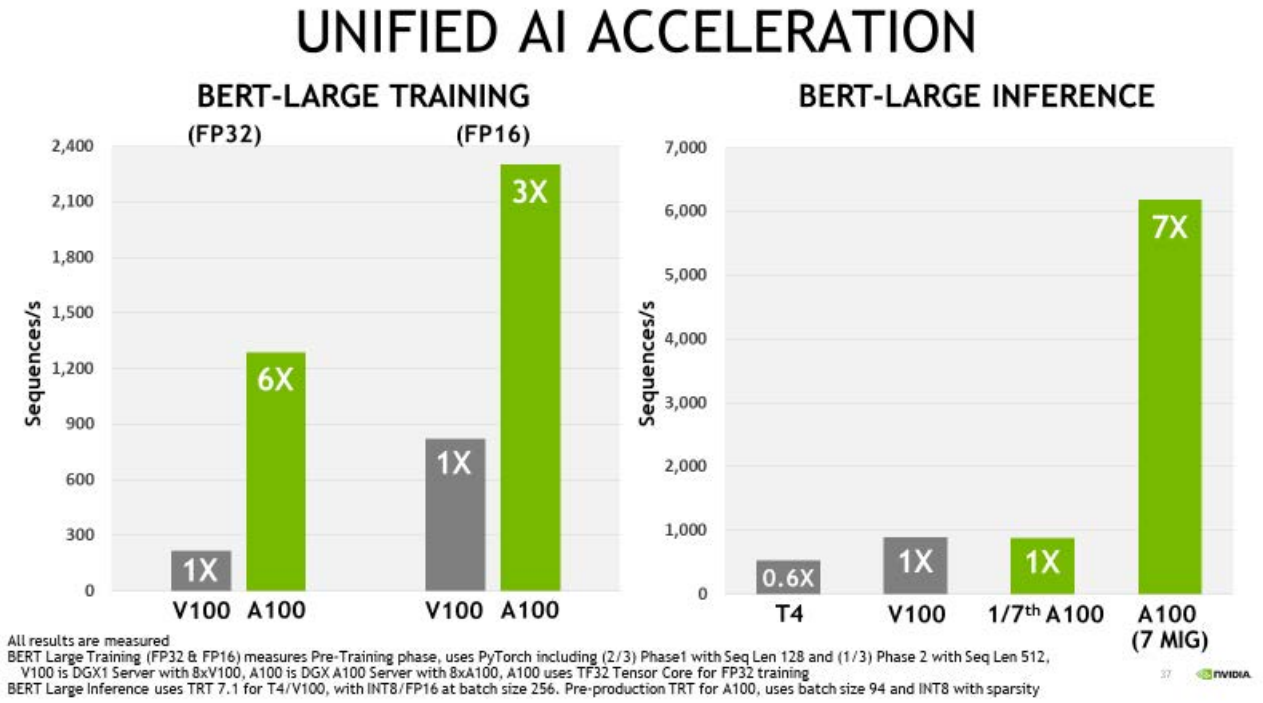
NEW: MULTI-INSTANCE GPU (MIG)

- Up to 7 instances total, dynamically reconfigurable
- **Compute instances:** compute/fault isolation, but share/compete for memory
- **GPU instances:** separate and isolated paths through the entire memory system



ELASTIC GPU COMPUTING

- Each A100 is 1 to 7 GPUs
- Each DGX A100 is 1 to 56 GPUs
- Each GPU can serve a different user, with full memory isolation and QoS



→S21975: Inside NVIDIA's Multi-Instance GPU Feature (recording available)

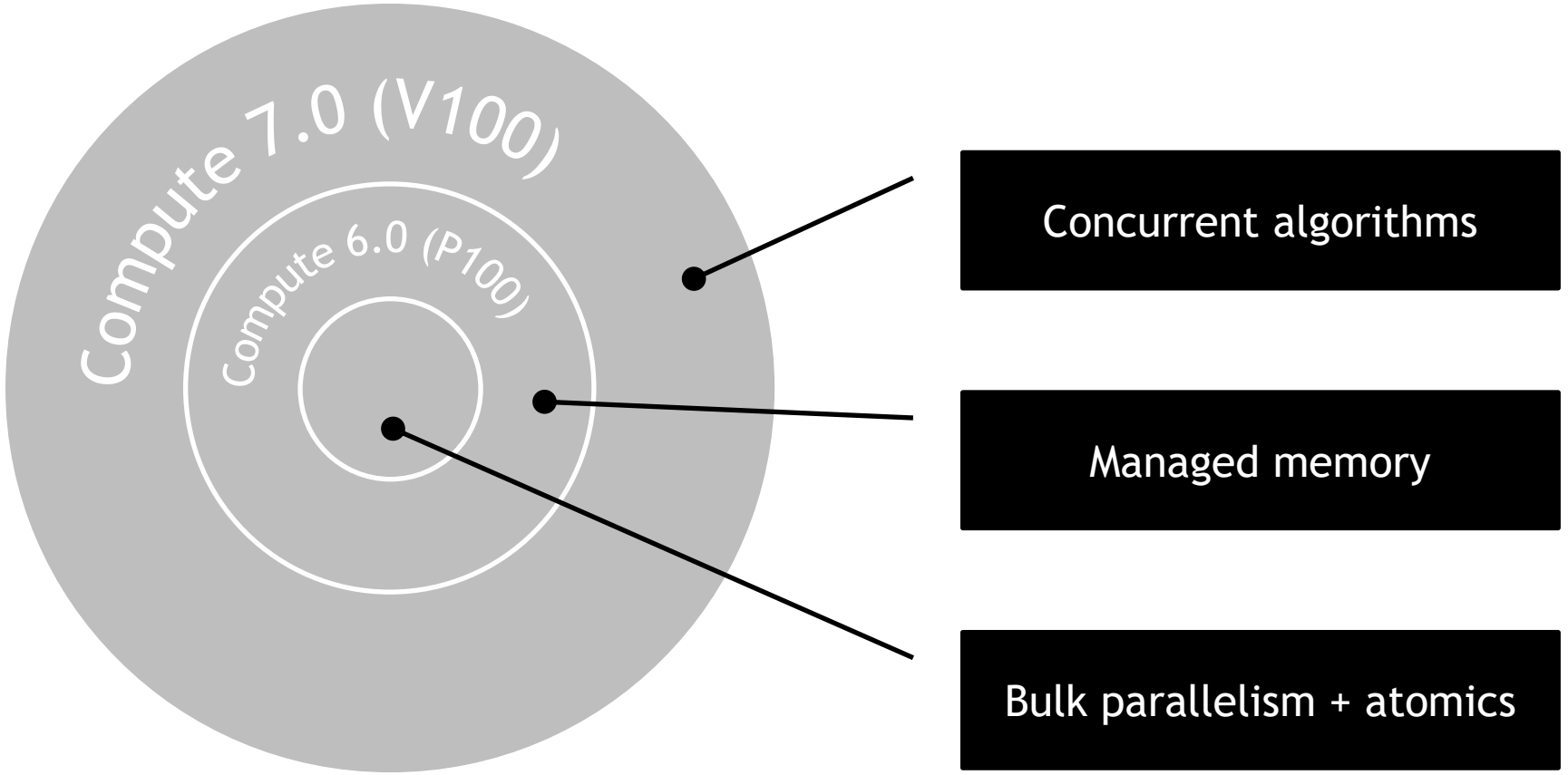
→S21884: Under the Hood of the new DGX A100 System Architecture (recording available soon)

→S21702: Introducing NVIDIA DGX A100: The Universal AI System for Enterprise, 5/20 9:00am PDT

1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. **Productivity**

COMPUTE CAPABILITY

Programming Model Development at 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) {
  ...
  cudaMemcpy(d_x, x, ...);
  cudaMemcpy(d_y, y, ...);

  saxpy<<<(N+255)/256,256>>>(...);

  cudaMemcpy(y, d_y, ...);
```

Maximize GPU Performance with
CUDA C++/Fortran

GPU Accelerated Math Libraries

→S21766 Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing, 5/19 1:30PM PST

PROGRAMMING MODEL WANTED

Software pipelining to hide latency is hard.

```
__device__ void exhibit_A1()
{
  memcpy(/* ... */); //< blocks here
  /* more work */

  compute();         //< needed here
  /* more work */
}
```

Data

```
__device__ void exhibit_B1()
{
  compute_head();
  __syncthreads(); //< blocks here
  /* more work */

  compute_tail();  //< needed here
  /* more work */
}
```

Compute

PROGRAMMING MODEL WANTED

Software pipelining to hide latency is hard.

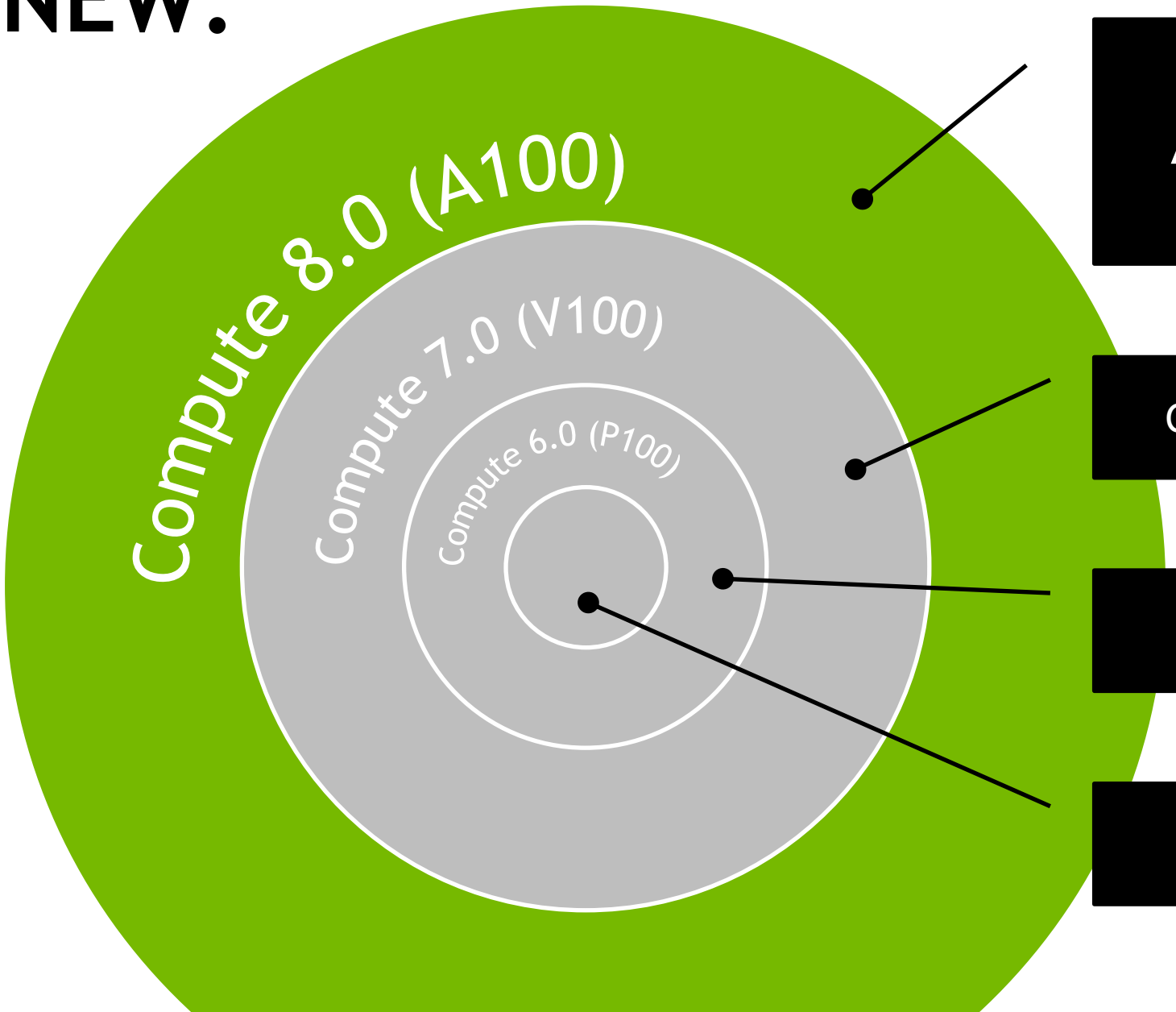
```
__device__ void exhibit_A2()
{
  memcpy(/* ... */); //< blocks here
  /* memcpy( ... ); */

  compute();         //< needed here
  /* compute(); */
}
```

Data

```
__device__ void exhibit_B2()
{
  compute_head();
  __syncthreads(); //< blocks here
  /* compute_head();
     __syncthreads(); */
  compute_tail();  //< needed here
  /* compute_tail(); */
}
```

Compute

NEW:



- Asynchronous algorithms
- Concurrent algorithms
- Managed memory
- Bulk parallelism

CO-DESIGNED: A100 & C++20 barrier

Key to asynchronous programming in compute_80

```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;
```

```
class barrier {  // synopsis
  //...
  void arrive_and_wait();
  arrival_token arrive(ptrdiff_t = 1);
  void wait(arrival_token &&) const;
  //...
};
```

Nonblocking

ASYNCHRONOUS COPY + BARRIER

Capability	PTX ISA	CUDA C++ API
Asynchronous barrier	`mbarrier.{`*<basis functions>*`}`	`cuda::barrier<…>`
Asynchronous copy	`cp.async.ca +` `cp.async.mbarrier.arrive`	`cuda::memcpy_async(…)`
+Cache-bypass	`cp.async.cg`	CUDA 11 preview library in `experimental::` namespace
+Zero-fill ragged edge	`cp.async.* … wr-size, rd-size;`	
+User-level tracking	`cp.async.mbarrier.arrive.noinc`	
+Single-threaded mode	`cp.async.{commit_group, wait_group}`	

ASYNCHRONOUS PROGRAMMING MODEL

```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;
```

```
__device__ void exhibit_A3()
{
  __shared__ barrier b1, b2;
  // ^^initialization omitted
   cuda::memcpy_async(/* ... */, b1);
    cuda::memcpy_async(/* ... */, b2);
  b1.arrive_and_wait();
  compute();
    b2.arrive_and_wait();
    compute();
}
```

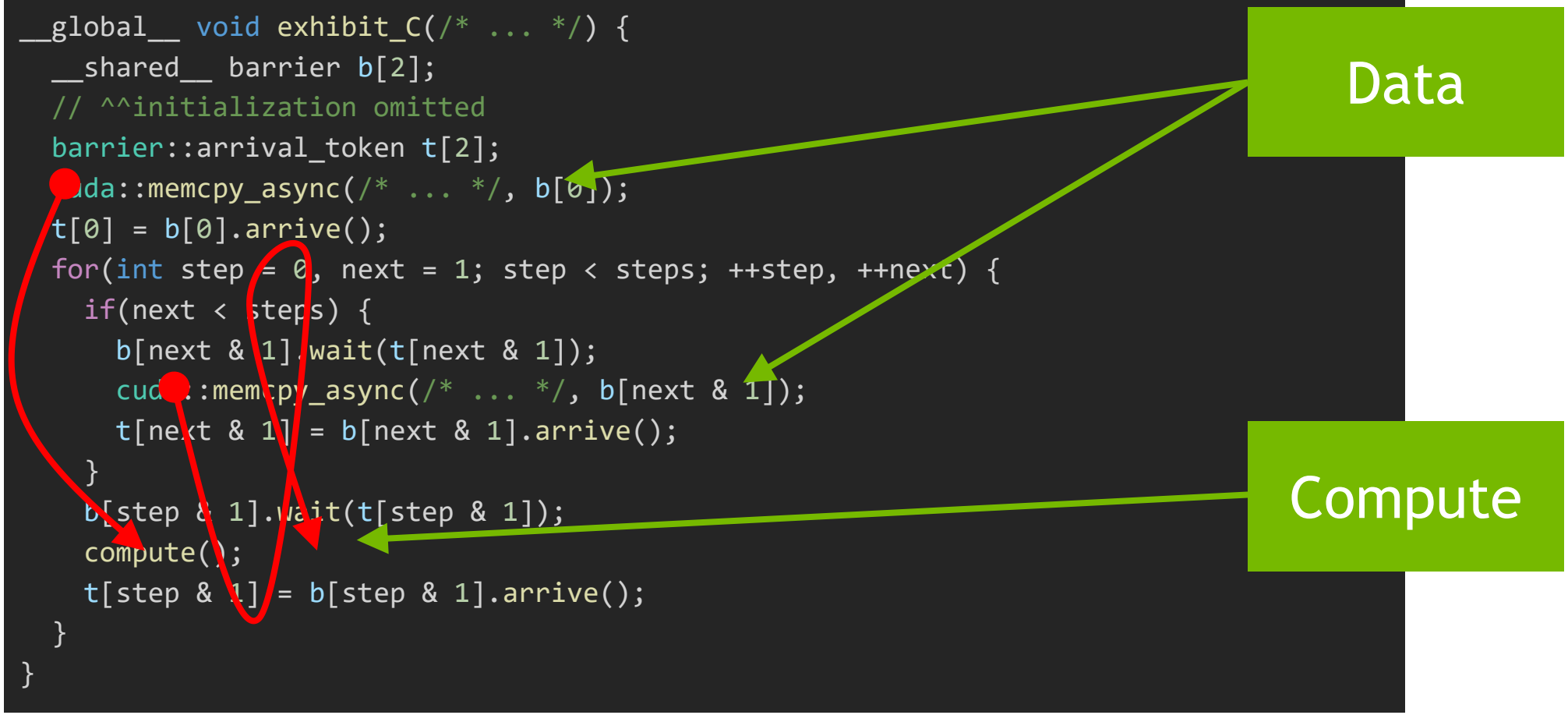
Data

```
__device__ void exhibit_B3()
{
  __shared__ barrier b1, b2;
  // ^^initialization omitted
  compute_head();
  auto t1 = b1.arrive();
    compute_head();
    auto t2 = b2.arrive();
  b1.wait(t1);
  compute_tail();
    b2.wait(t2);
    compute_t
}
```

Compute

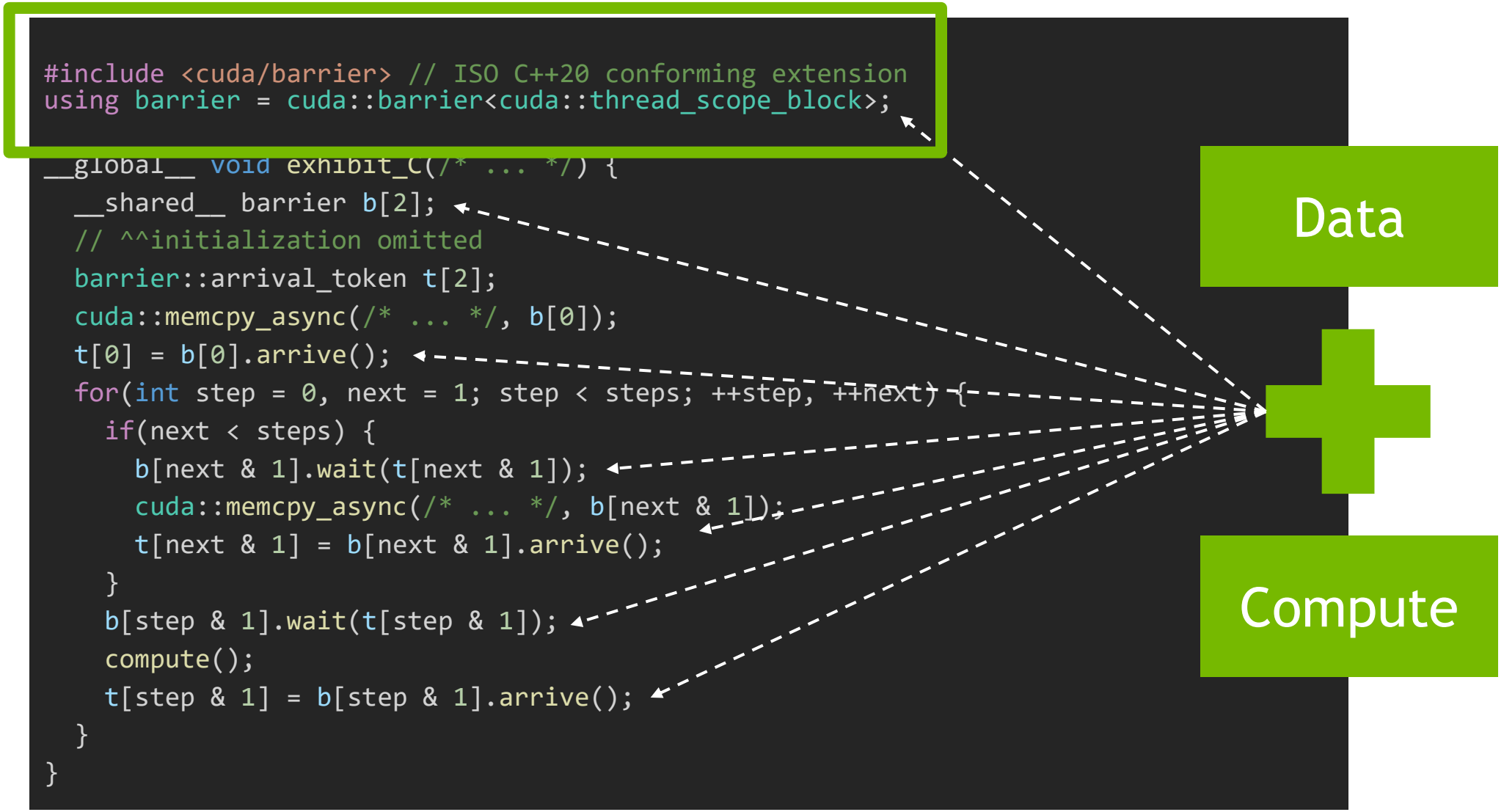
MULTI-BUFFERING PIPELINES IN C++

```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;

__global__ void exhibit_C(/* ... */) {
  __shared__ barrier b[2];
  // ^^initialization omitted
  barrier::arrival_token t[2];
  cuda::memcpy_async(/* ... */, b[0]);
  t[0] = b[0].arrive();
  for(int step = 0, next = 1; step < steps; ++step, ++next) {
    if(next < steps) {
      b[next & 1].wait(t[next & 1]);
      cuda::memcpy_async(/* ... */, b[next & 1]);
      t[next & 1] = b[next & 1].arrive();
    }
    b[step & 1].wait(t[step & 1]);
    compute();
    t[step & 1] = b[step & 1].arrive();
  }
}
```



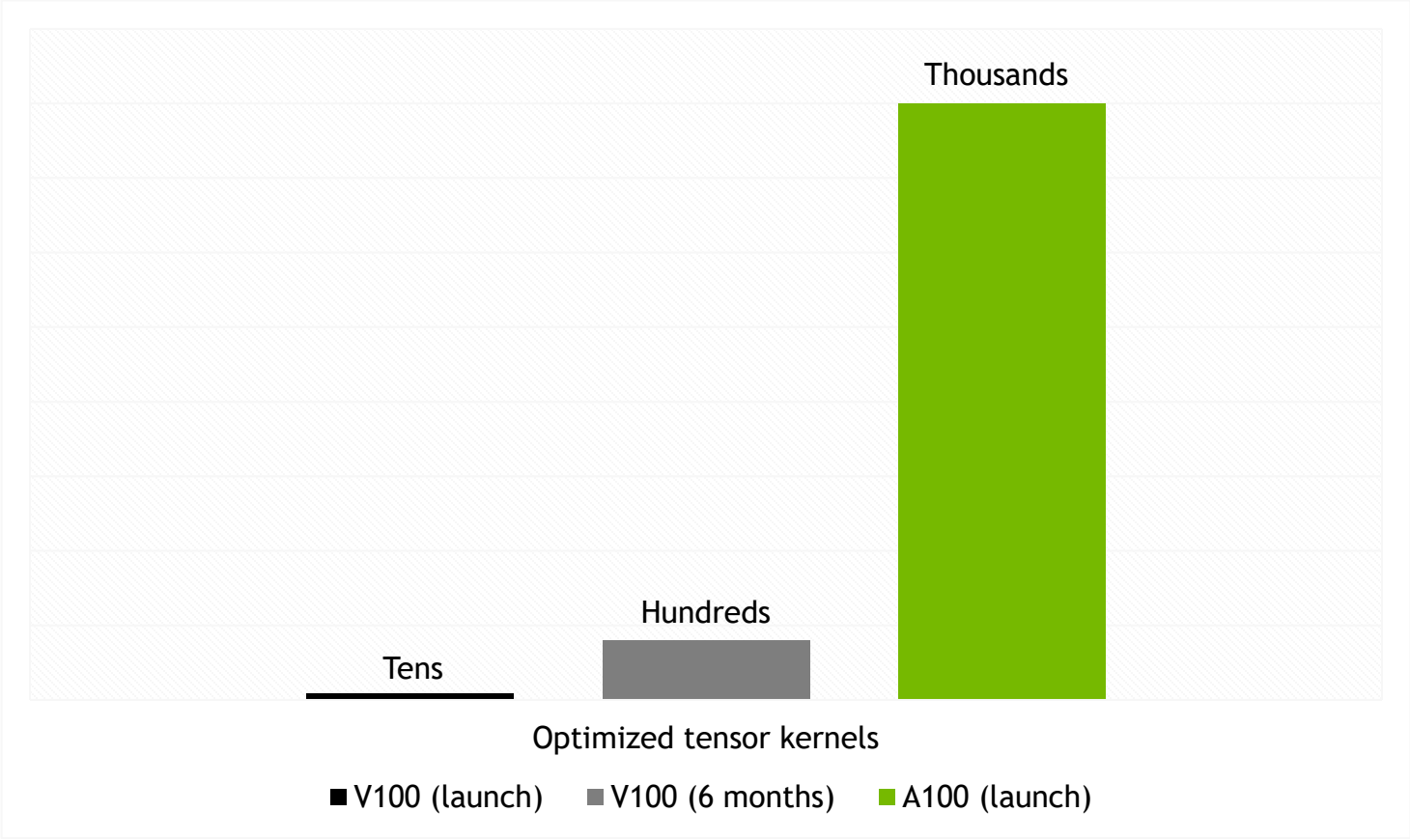
MULTI-BUFFERING PIPELINES IN C++

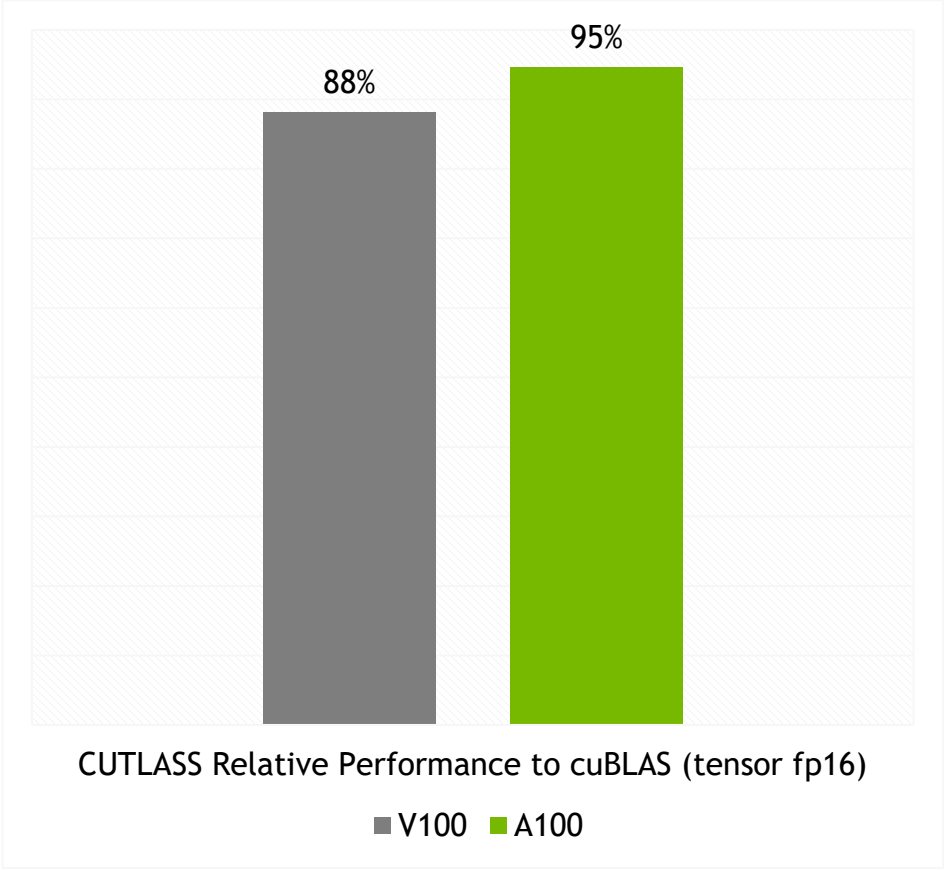
```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;

__global__ void exhibit_C(/* ... */) {
  __shared__ barrier b[2];
  // ^^initialization omitted
  barrier::arrival_token t[2];
  cuda::memcpy_async(/* ... */, b[0]);
  t[0] = b[0].arrive();
  for(int step = 0, next = 1; step < steps; ++step, ++next) {
    if(next < steps) {
      b[next & 1].wait(t[next & 1]);
      cuda::memcpy_async(/* ... */, b[next & 1]);
      t[next & 1] = b[next & 1].arrive();
    }
    b[step & 1].wait(t[step & 1]);
    compute();
    t[step & 1] = b[step & 1].arrive();
  }
}
```



→S21760: CUDA New Features And Beyond, 5/19 10:15am PDT

OUR PRODUCTIVITY GAINS FROM A100

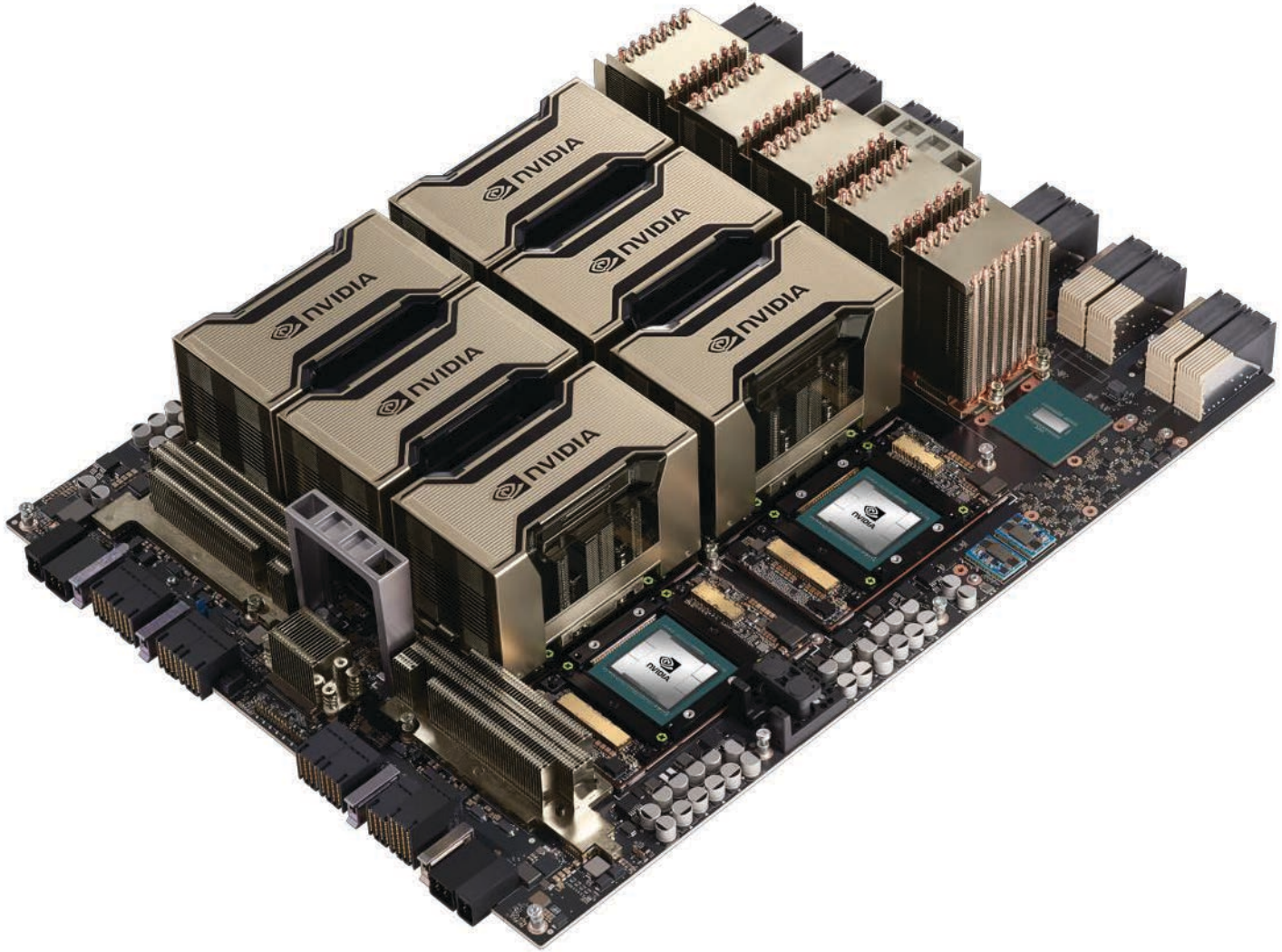


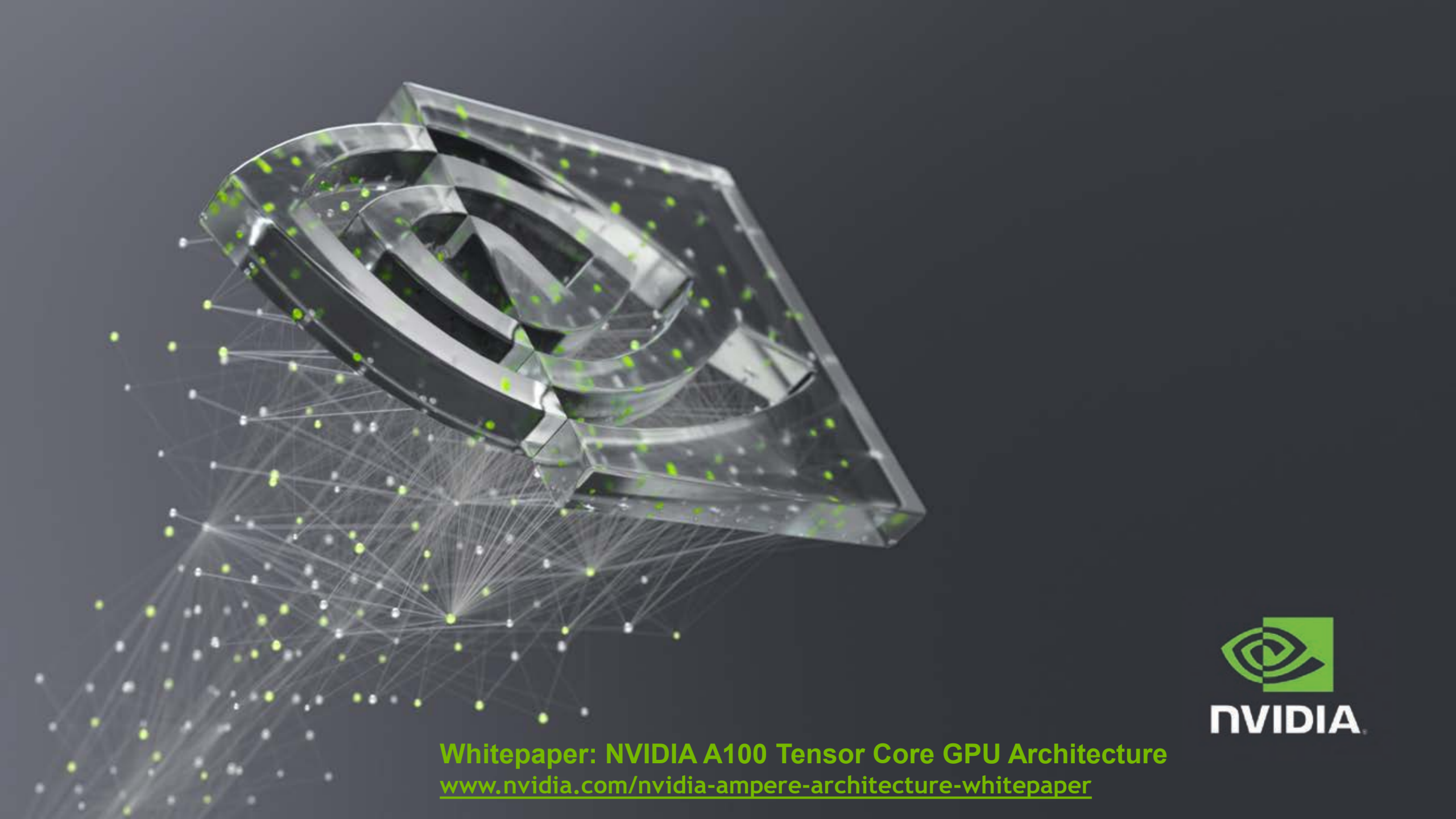


→S21745: Developing CUDA kernels to push Tensor Cores to the Absolute Limit on NVIDIA A100, 5/21 11:30AM PST

CLOSING

UNPRECEDENTED ACCELERATION AT EVERY SCALE





Whitepaper: NVIDIA A100 Tensor Core GPU Architecture

www.nvidia.com/nvidia-ampere-architecture-whitepaper

CS 295: Modern Systems
GPU Computing Introduction

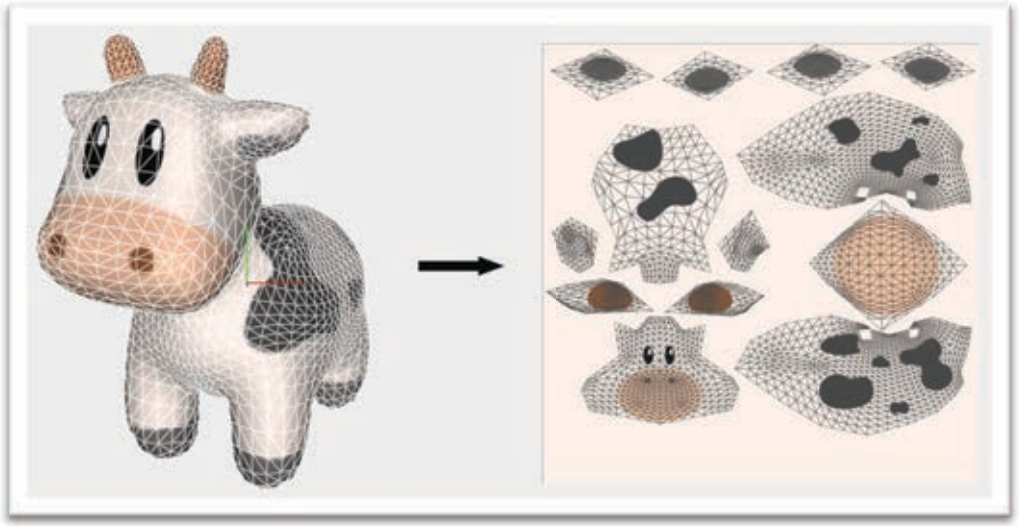
Sang-Woo Jun

Spring 2019

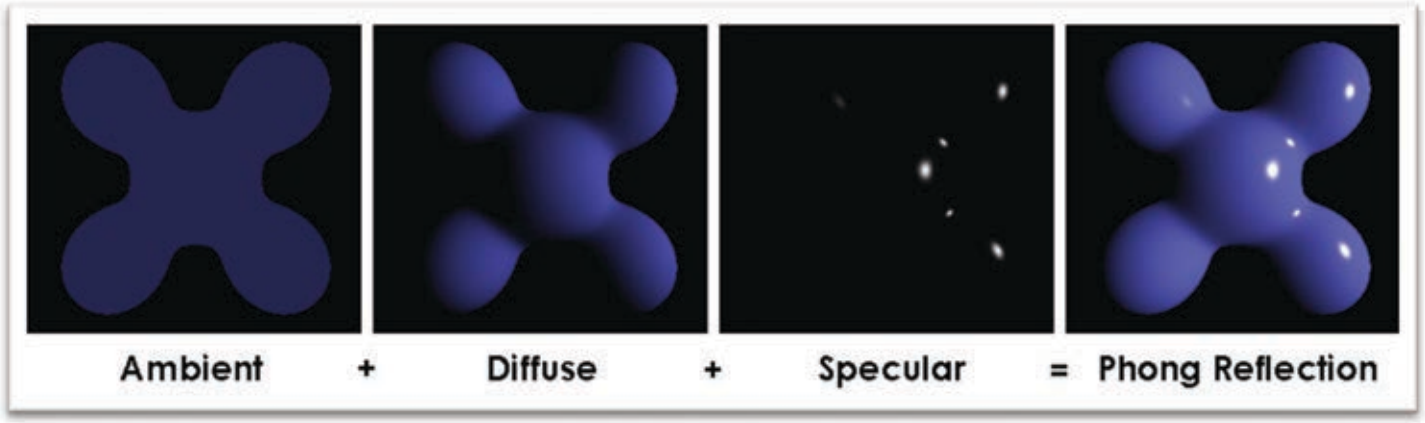
UCI

Graphic Processing – Some History

- 1990s: Real-time 3D rendering for video games were becoming common
  - Doom, Quake, Descent, … (Nostalgia!)
- 3D graphics processing is immensely computation-intensive



Texture mapping



Shading

Warren Moore, "Textures and Samplers in Metal," Metal by Example, 2014

Gray Olsen, "CSE 470 Assignment 3 Part 2 - Gourad/Phong Shading," grayolsen.com, 2018

Graphic Processing – Some History

- Before 3D accelerators (GPUs) were common
- CPUs had to do all graphics computation, while maintaining framerate!
  - Many tricks were played

Doom (1993) : “Affine texture mapping”

- Linearly maps textures to screen location, disregarding depth
- Doom levels did not have slanted walls or ramps, to hide this

Graphic Processing – Some History

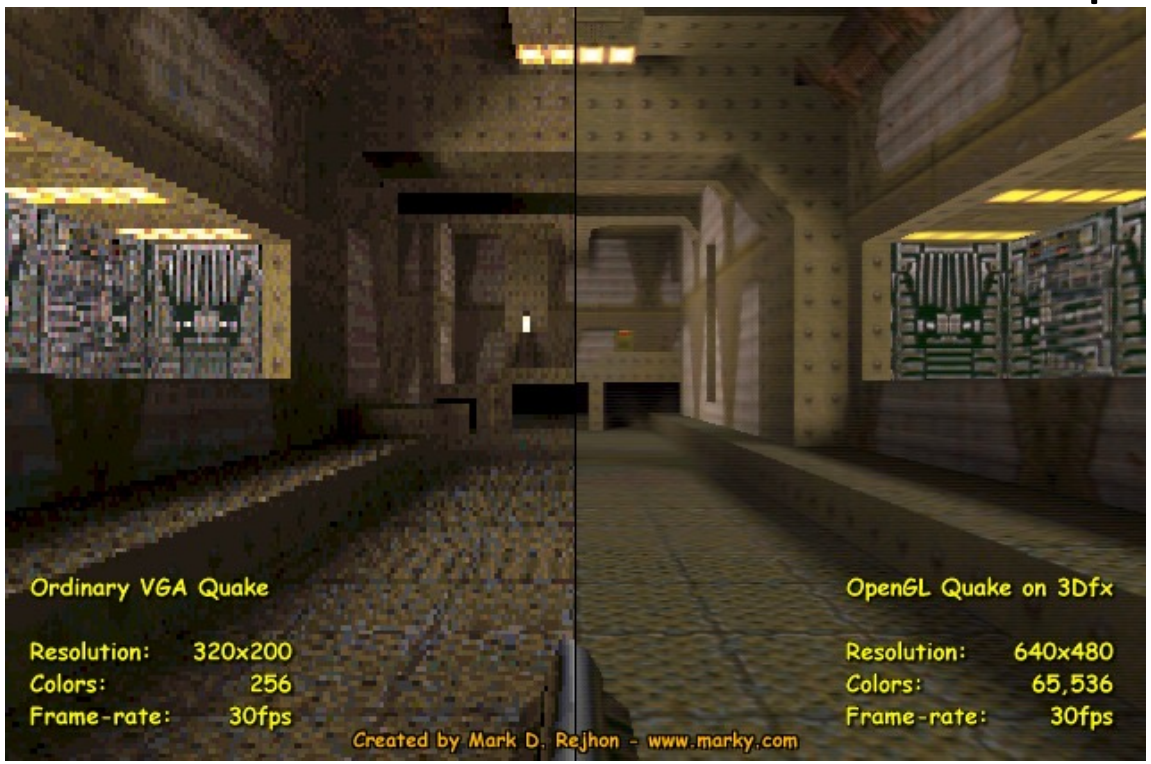
- Before 3D accelerators (GPUs) were common
- CPUs had to do all graphics computation, while maintaining framerate!
  - Many tricks were played

Quake III arena (1999) : “Fast inverse square root” magic!

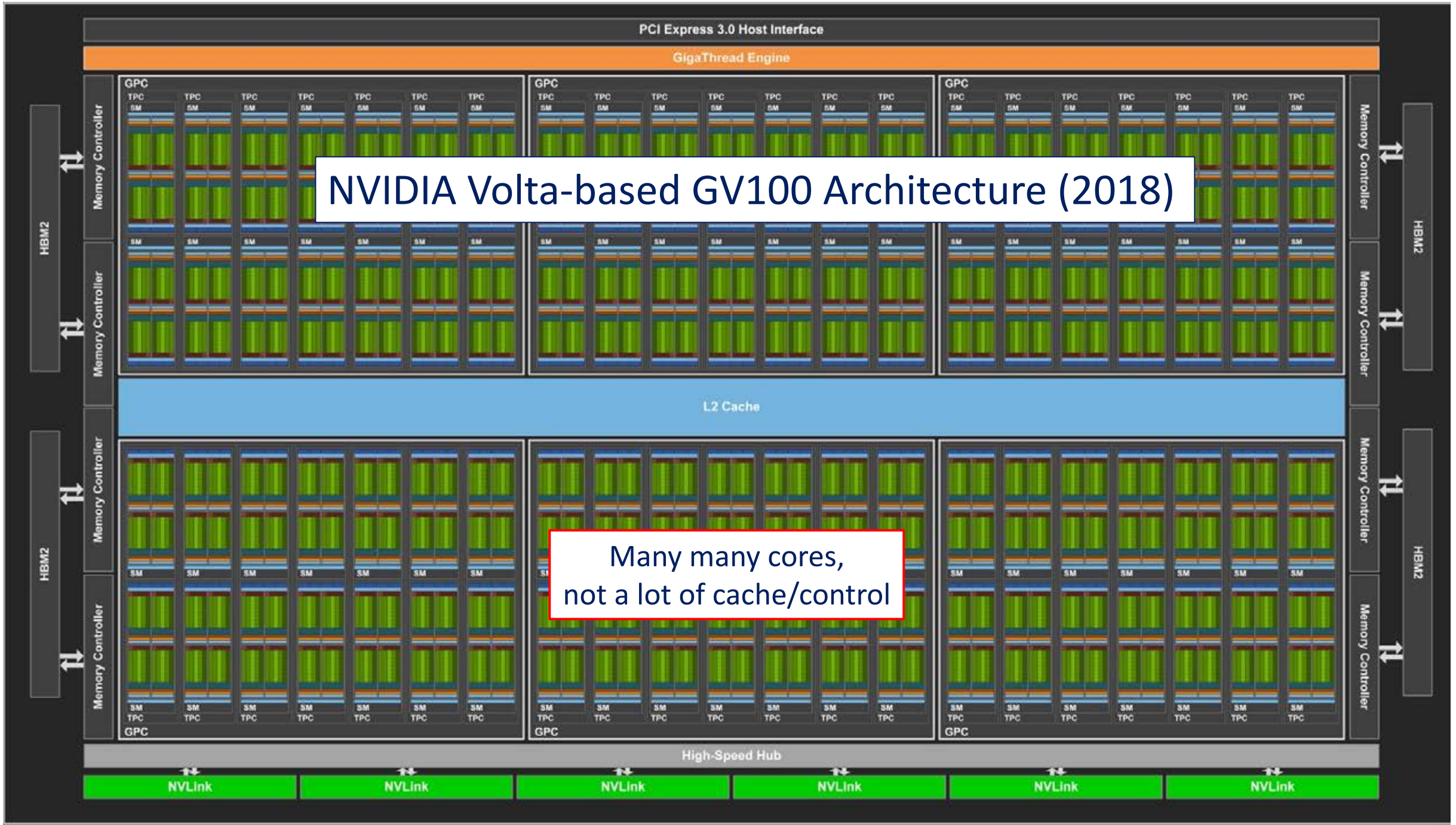
```
float Q_rsqrt( float number )
{
    const float x2 = number * 0.5F;
    const float threehalfs = 1.5F;

    union {
        float f;
        uint32_t i;
    } conv = {number}; // member 'f' set to value of 'number'.
    conv.i  = 0x5f3759df - ( conv.i >> 1 );
    conv.f  *= ( threehalfs - ( x2 * conv.f * conv.f ) );
    return conv.f;
}
```

Introduction of 3D Accelerator Cards

- Much of 3D processing is short algorithms repeated on a lot of data
  - pixels, polygons, textures, …
- Dedicated accelerators with simple, massively parallel computation



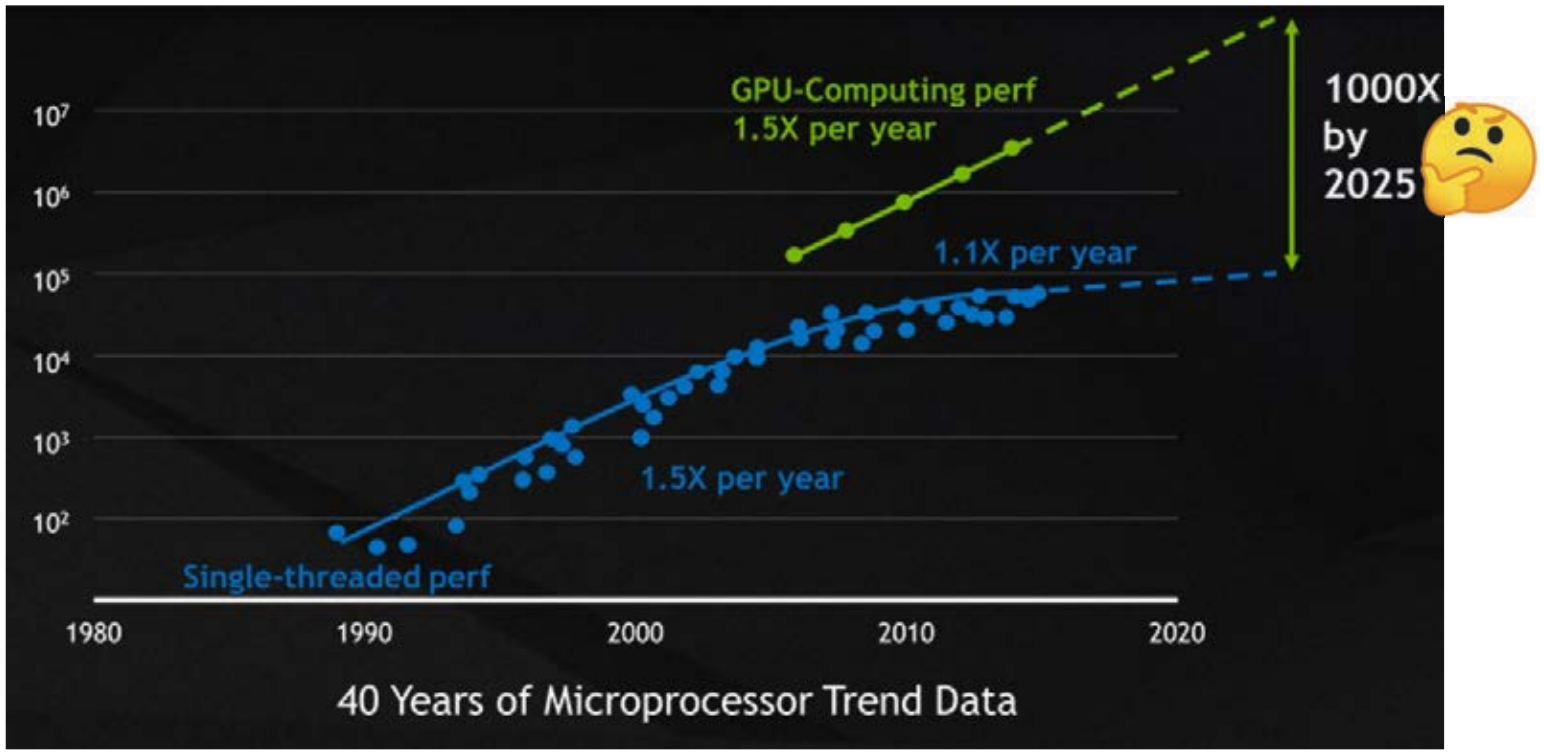
A Diamond Monster 3D, using the Voodoo chipset (1997)
(Konstantin Lanzet, Wikipedia)


PCI Express 3.0 Host Interface
GigaThread Engine
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink
NVIDIA Volta-based GV100 Architecture (2018)
Many many cores,
not a lot of cache/control

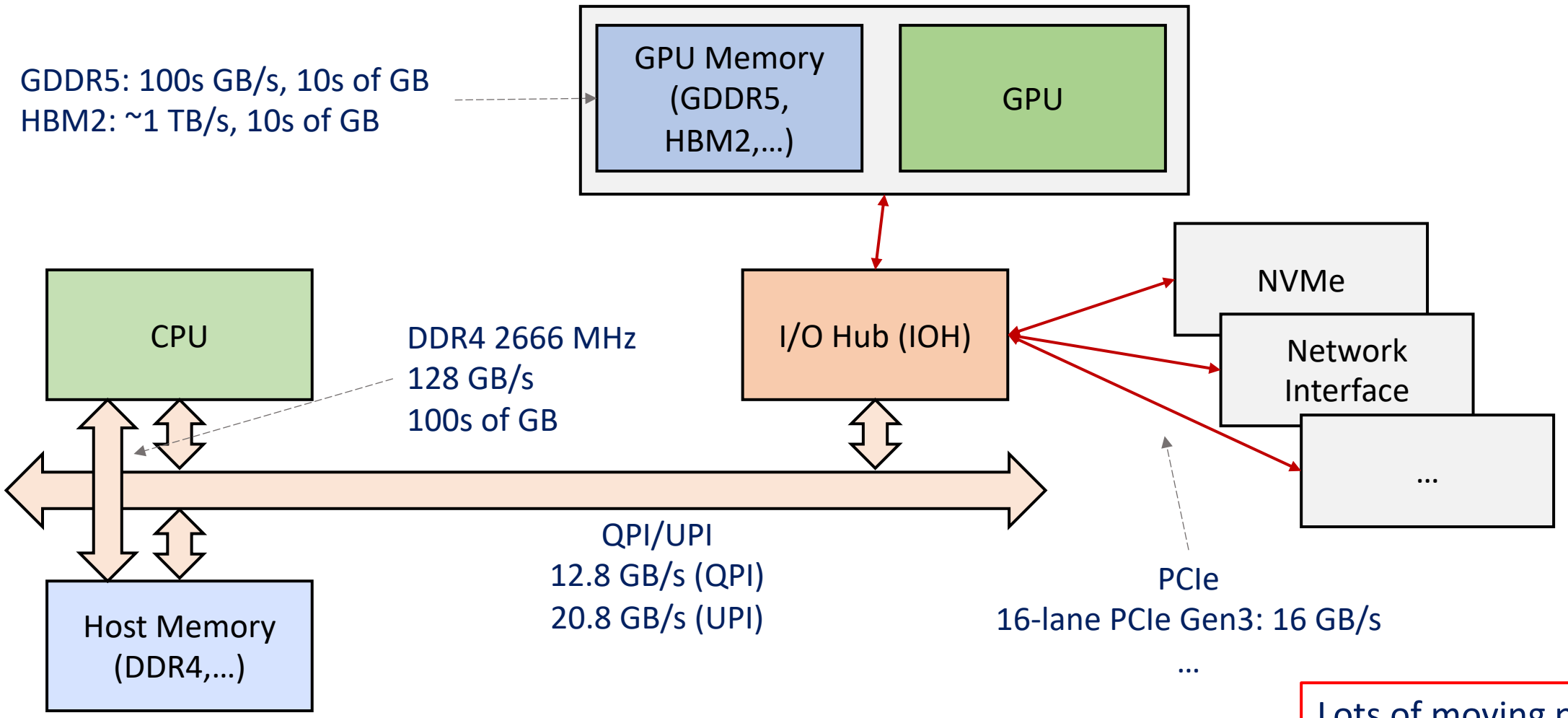
Peak Performance vs. CPU

	Throughput	Power	Throughput/Power
Intel Skylake	128 SP GFLOPS/4 Cores	100+ Watts	~1 GFLOPS/Watt
NVIDIA V100	15 TFLOPS	200+ Watts	~75 GFLOPS/Watt

Also,

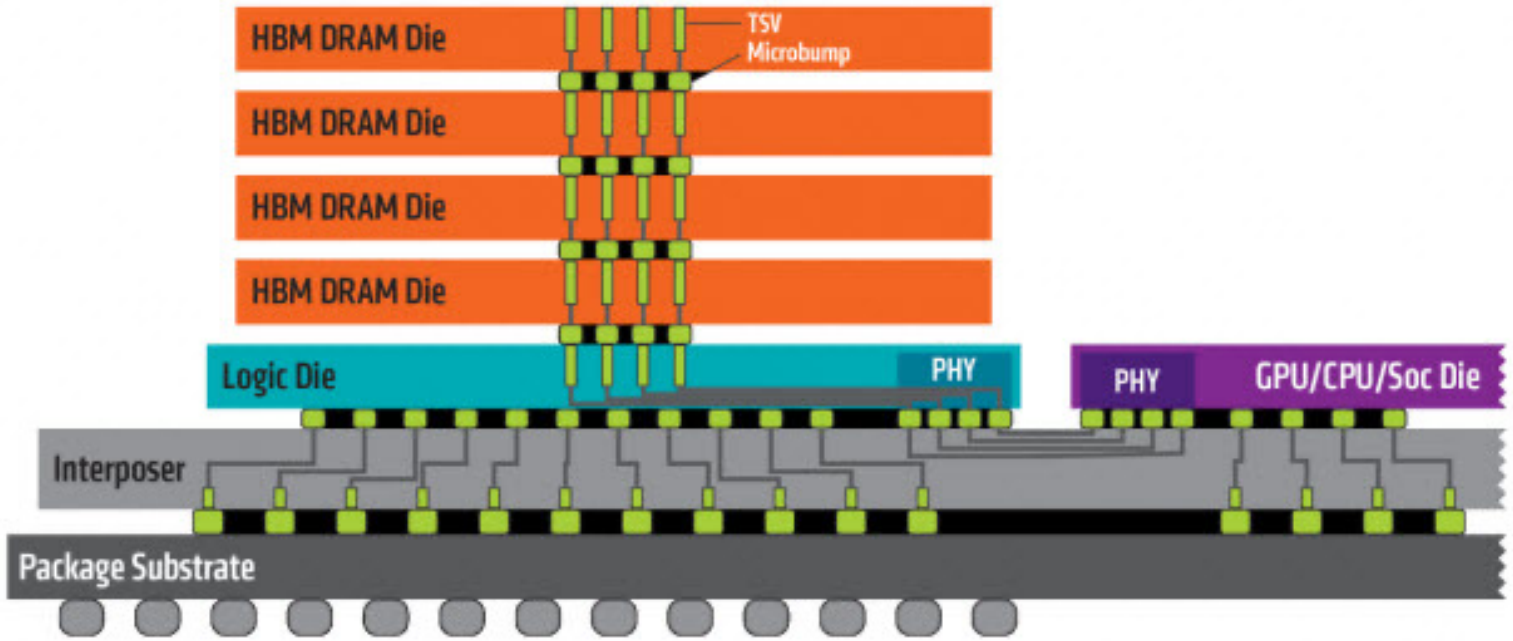


System Architecture Snapshot With a GPU (2019)



High-Performance Graphics Memory

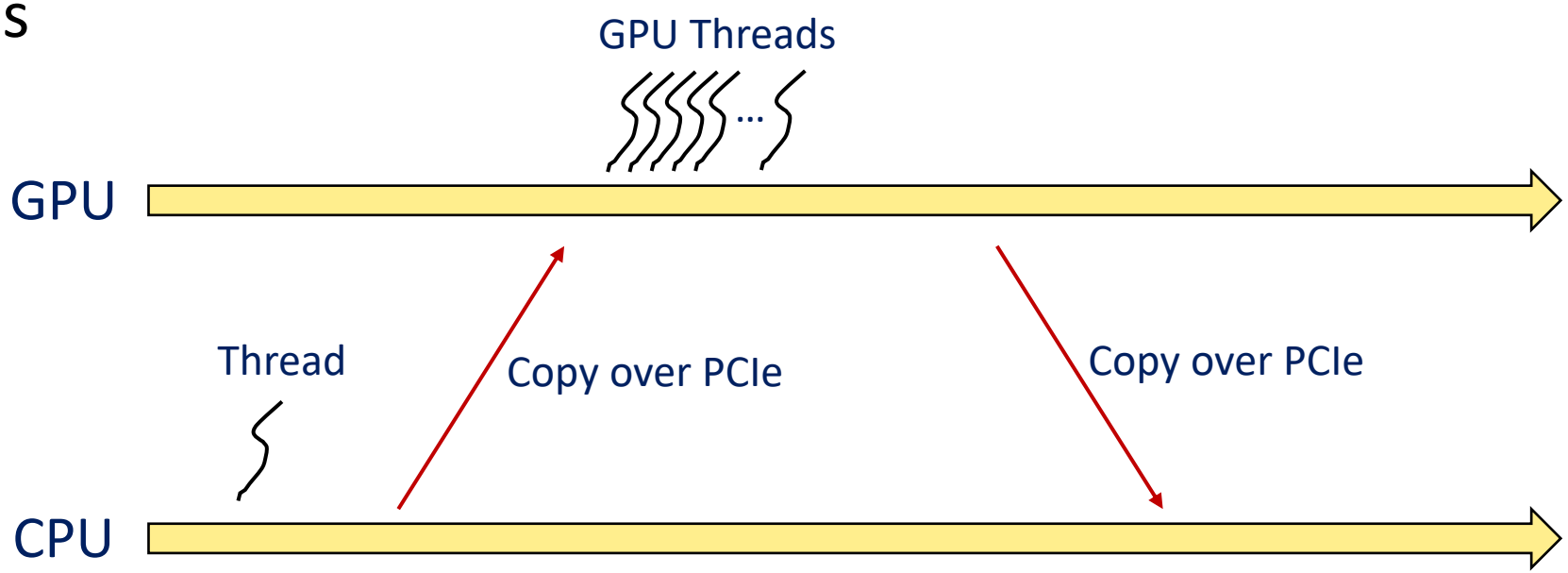
- Modern GPUs even employing 3D-stacked memory via silicon interposer
  - Very wide bus, very high bandwidth
  - e.g., HBM2 in Volta



Graphics Card Hub, “GDDR5 vs GDDR5X vs HBM vs HBM2 vs GDDR6 Memory Comparison,” 2019

Massively Parallel Architecture For Massively Parallel Workloads!

- NVIDIA CUDA (Compute Uniform Device Architecture) – 2007
  - A way to run custom programs on the massively parallel architecture!
- OpenCL specification released – 2008
- Both platforms expose synchronous execution of a massive number of threads



CUDA Execution Abstraction

- Block: Multi-dimensional array of threads
  - 1D, 2D, or 3D
  - Threads in a block can synchronize among themselves
  - Threads in a block can access shared memory
  - CUDA (Thread, Block) ~= OpenCL (Work item, Work group)
- Grid: Multi-dimensional array of blocks
  - 1D or 2D
  - Blocks in a grid can run in parallel, or sequentially
- Kernel execution issued in grid units
- Limited recursion (depth limit of 24 as of now)

Simple CUDA Example

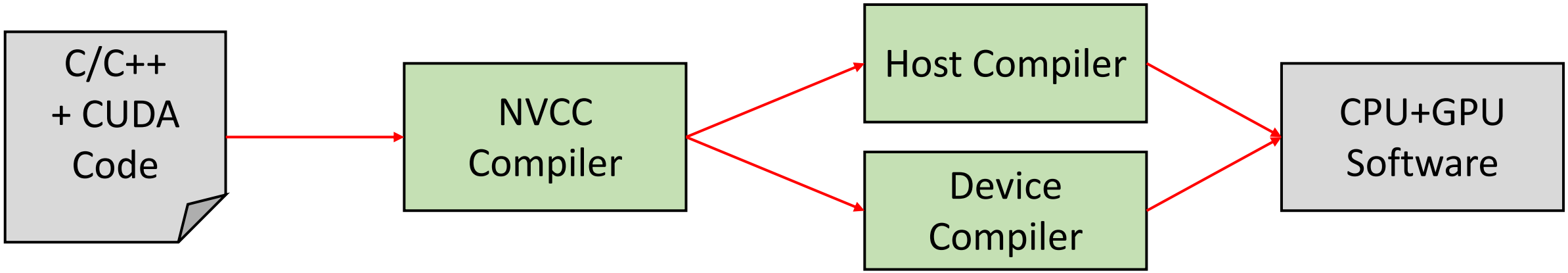
Asynchronous call

CPU side

```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

GPU side

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```



Simple CUDA Example

```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
}
```

1 block

N threads per block

Should wait for kernel to finish

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

Only void allowed

Which of N threads am I?
See also: blockIdx

__global__:
In GPU, called from host/GPU
__device__:
In GPU, called from GPU
__host__:
In host, called from host

N instances of VecAdd spawned in GPU

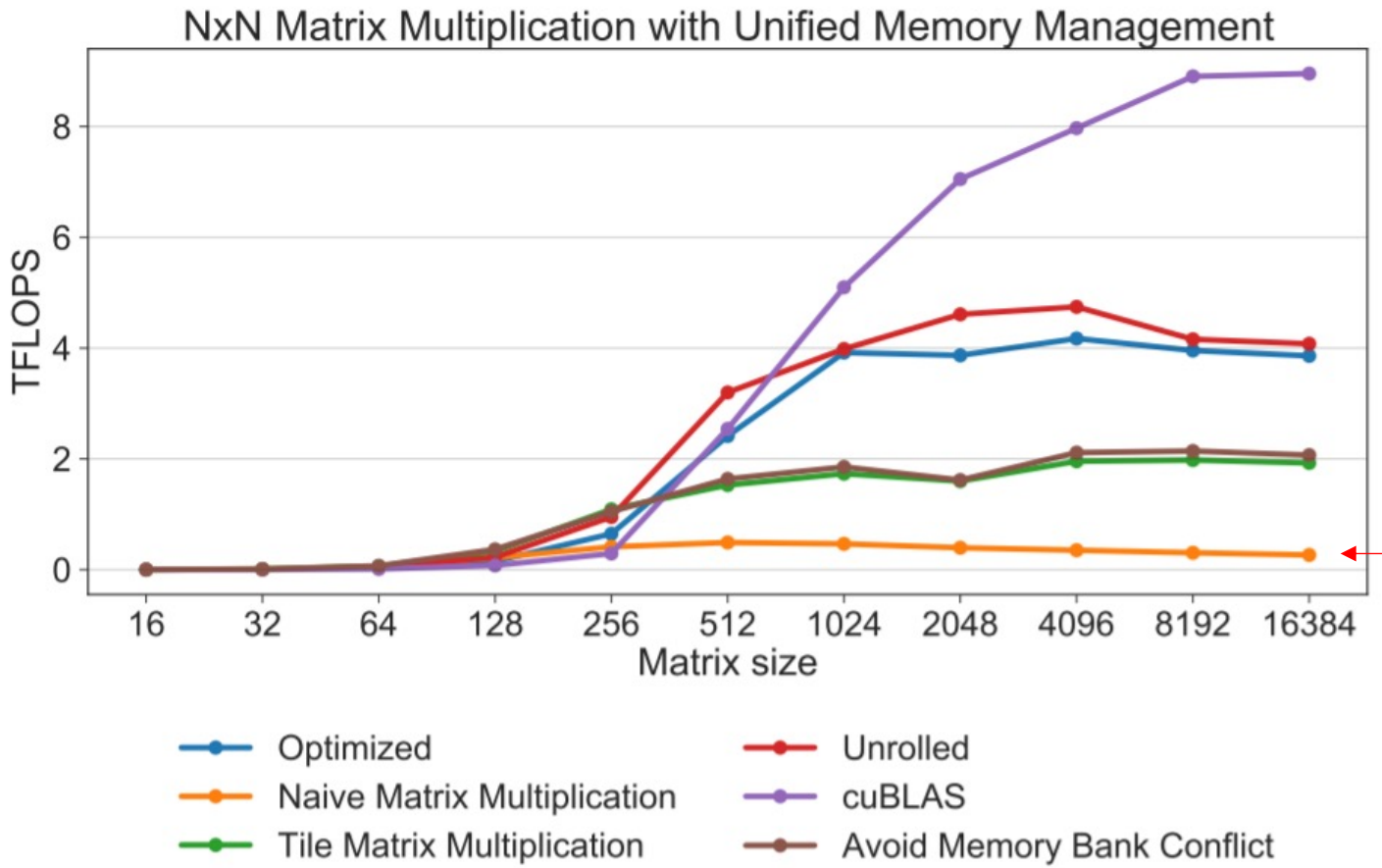
One function can be both

More Complex Example: Picture Blurring

- Slides from NVIDIA/UIUC Accelerated Computing Teaching Kit
- Another end-to-end example
  https://devblogs.nvidia.com/even-easier-introduction-cuda/

- Great! Now we know how to use GPUs – Bye?

Matrix Multiplication Performance Engineering

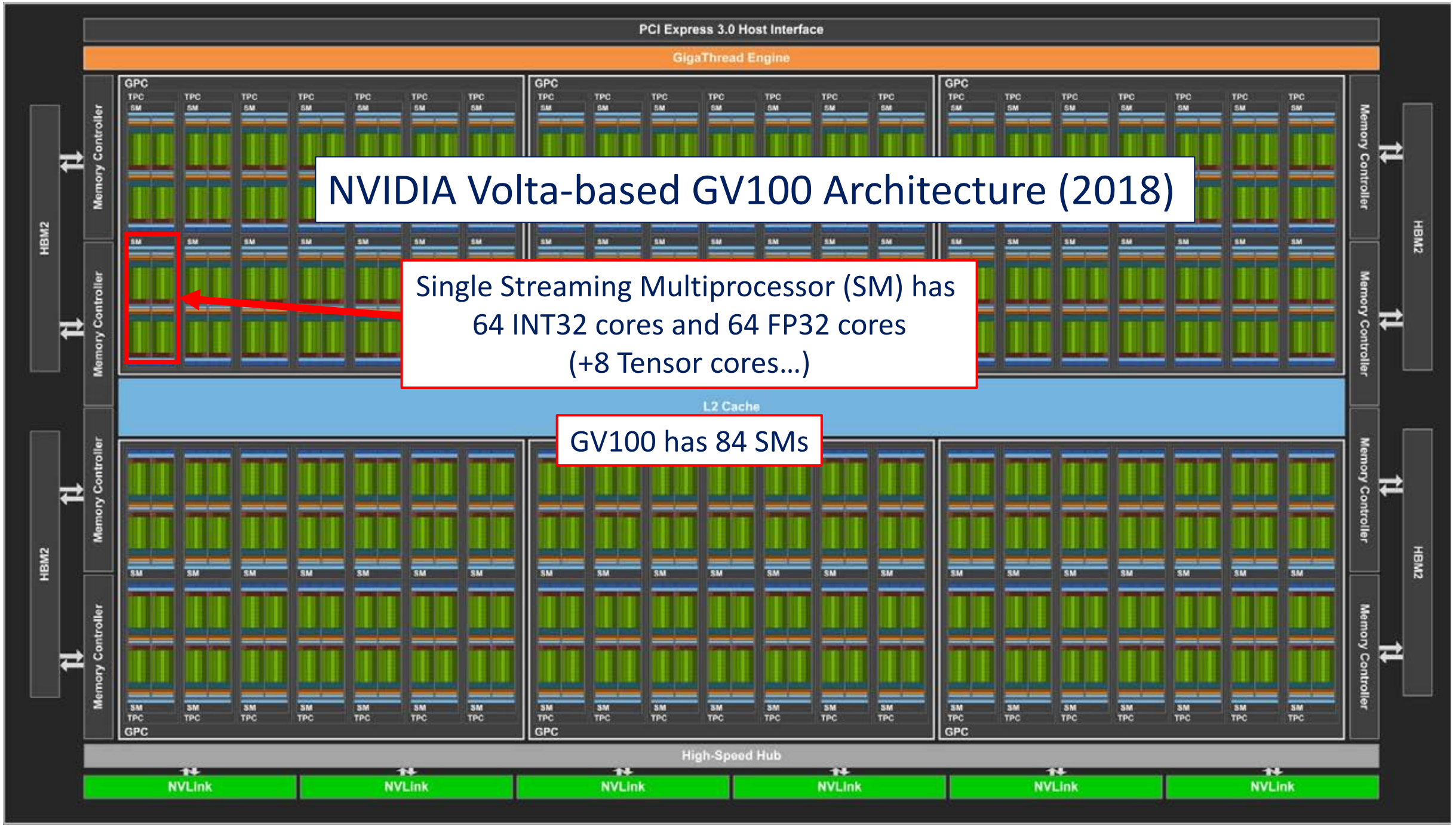


No faster than CPU

Results from NVIDIA P100

Coleman et. al., "Efficient CUDA," 2017

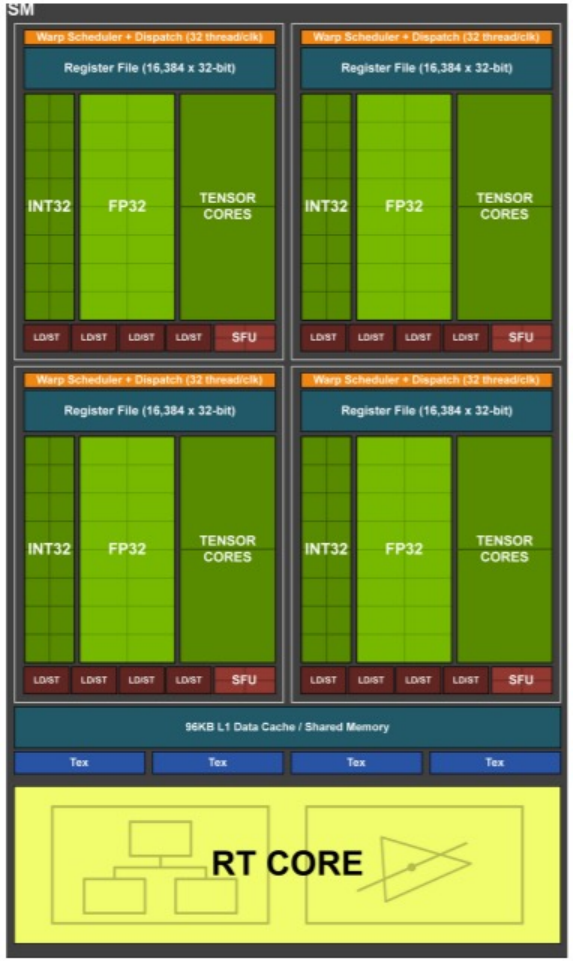
Architecture knowledge is needed (again)


PCI Express 3.0 Host Interface
GigaThread Engine
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink
NVIDIA Volta-based GV100 Architecture (2018)
Single Streaming Multiprocessor (SM) has
64 INT32 cores and 64 FP32 cores
(+8 Tensor cores…)
GV100 has 84 SMs

Volta Execution Architecture

- 64 INT32 Cores, 64 FP32 Cores, 4 Tensor Cores, Ray-tracing cores..
  - Specialization to make use of chip space…?
- Not much on-chip memory per thread
  - 96 KB Shared memory
  - 1024 Registers per FP32 core
- Hard limit on compute management
  - 32 blocks AND 2048 threads AND 1024 threads/block
  - e.g., 2 blocks with 1024 threads, or 4 blocks with 512 threads
  - Enough registers/shared memory for all threads must be available (all context is resident during execution)

More threads than cores – Threads interleaved to hide memory latency



Resource Balancing Details

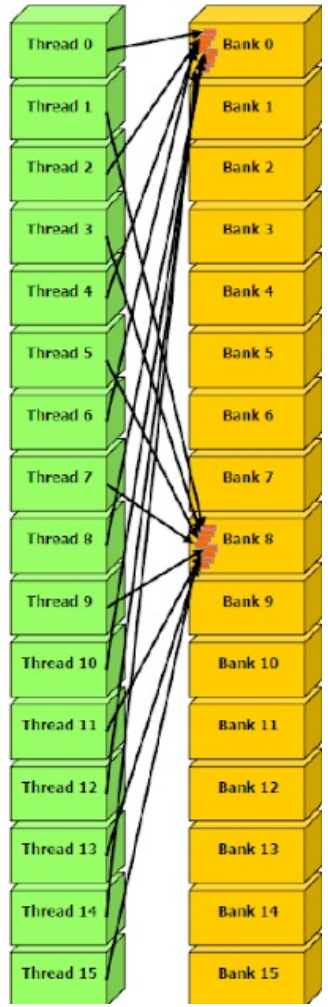
- How many threads in a block?
- Too small: 4x4 window == 16 threads
  - 128 blocks to fill 2048 thread/SM
  - SM only supports 32 blocks -> only 512 threads used
    - SM has only 64 cores… does it matter? Sometimes!
- Too large: 32x48 window == 1536 threads
  - Threads do not fit in a block!
- Too large: 1024 threads using more than 64 registers
- Limitations vary across platforms (Fermi, Pascal, Volta, …)

Warp Scheduling Unit

- Threads in a block are executed in 32-thread “warp” unit
  - Not part of language specs, just architecture specifics
  - A warp is SIMD – Same PC, same instructions executed on every core
- What happens when there is a conditional statement?
  - Prefix operations, or control divergence
  - More on this later!
- Warps have been 32-threads so far, but may change in the future

Memory Architecture Caveats

- Shared memory peculiarities
  - Small amount (e.g., 96 KB/SM for Volta) shared across all threads
  - Organized into banks to distribute access
  - Bank conflicts can drastically lower performance
- Relatively slow global memory
  - Blocking, caching becomes important (again)
  - If not for performance, for power consumption…



8-way bank conflict
1/8 memory bandwidth

Are Mobile DNN Accelerators Accelerating DNNs?

Qingqing Cao*, Alexandru E. Irimiea‡, Mohamed Abdelfattah◊,
Aruna Balasubramanian*, Nicholas D. Lane†◊

*

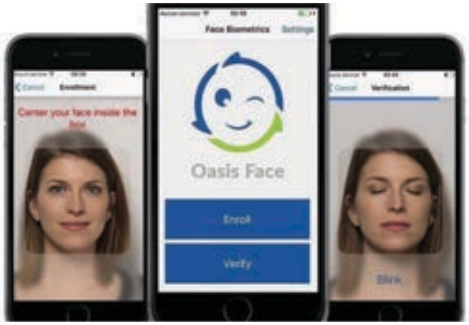
‡

†

◊

5th International Workshop on Embedded and Mobile Deep Learning (EMDL 2021)

Deep Learning Applications Exploding on Mobile



Challenges in Running **DNNs** on Mobile Processors

Mobile processors often run on battery-powered devices

Mobile processors often have limited processing power

Credit: OFA, ICLR 2020

Are Mobile DNN Accelerators the Solutions?

Vision processing unit
(VPU)

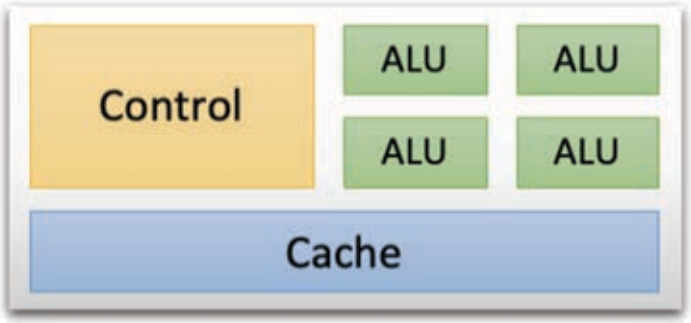
Tensor processing unit
(EdgeTPU)

Intelligence processing unit
(IPU)

Many customized DNN accelerators, xPUs: designed for better energy efficiency and greater DNN processing power

Background: Architecture of Mobile CPUs

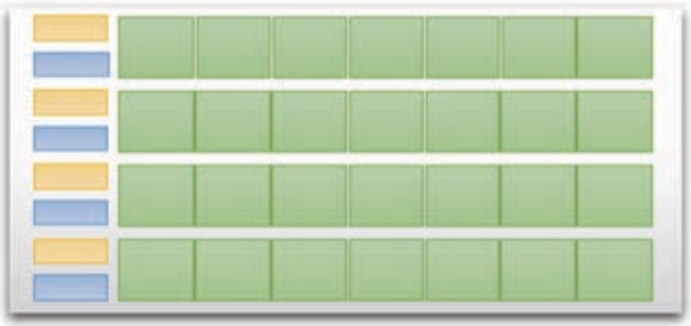
CPU



- Single instruction multiple data (SIMD) capabilities to parallelize compute intensive operations
- Designed for more general tasks, complex control logic and lower compute density

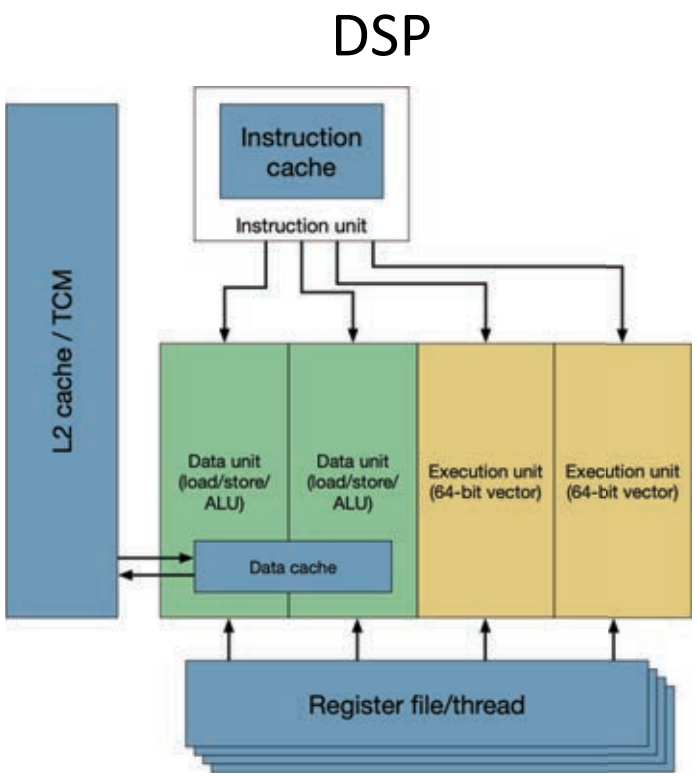
Background: Architecture of Mobile GPUs

GPU



- Stronger SIMD capabilities (many GPU shader cores/more ALUs), high compute density
- Mobile GPUs often share memory with CPUs, memory accesses are energy-consuming

Background: Architecture of DSPs



- Enhance SIMD by introducing vector execution unit in addition to ALUs
- Integer based operations, hardware-assisted multithreading, more energy efficient

Background: Differences between CPU, GPU and DSP

CPU: general purpose, SIMD instructions for parallel processing

GPU: originally designed for graphics processing, massive parallelism (more SIMD processing units), power hungry

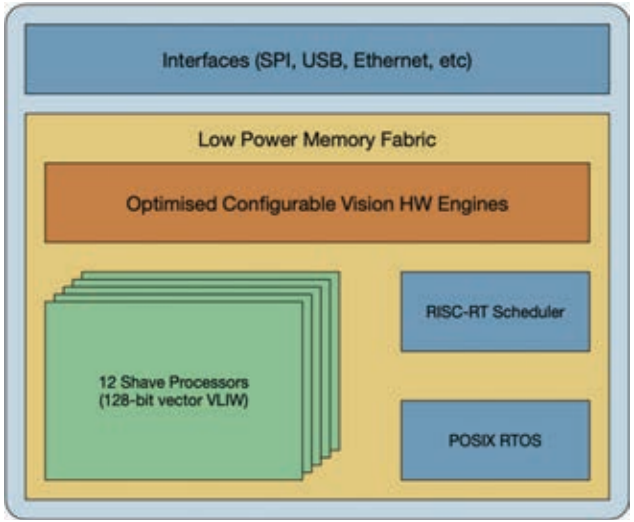
DSP: originally for photo/video/audio processing, integer-only vector operations, energy efficient

Goal of Mobile DNN Accelerator Empirical Study

Understanding the DNN performance on mobile accelerators in comparison to conventional processors like CPU, GPU, and DSP architectures

Mobile DNN Accelerators: The architecture of NCS VPU

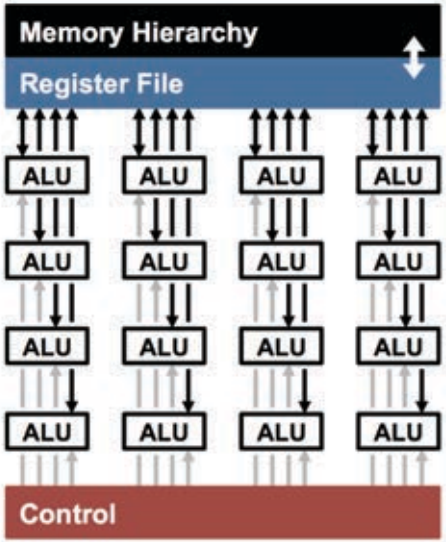
VPU of Neural Compute Stick (NCS)



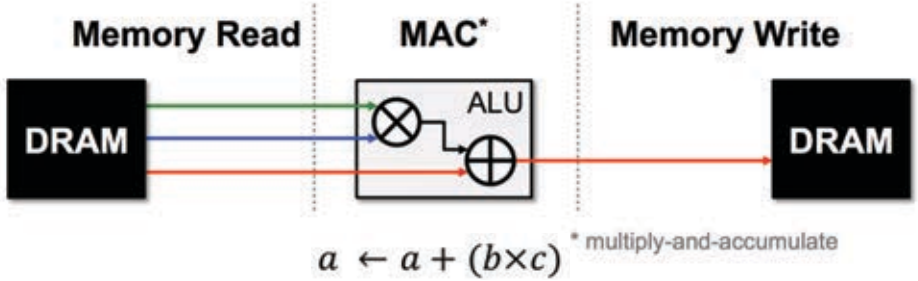
- 128-bit vector processing cores (SHAVE processors), SIMD support controlled via VLIW instructions
- NCS1: 12 SHAVE cores, 2MB on-chip memory (SRAM)
- NCS2: 16 SHAVE cores, 2.5 MB SRAM, additional neural compute engine dedicated for DNN workloads with higher efficiency

Why NCS Matters? Relationships to DNN Accelerators

CPUs and GPUs
(SIMD/SIMT)



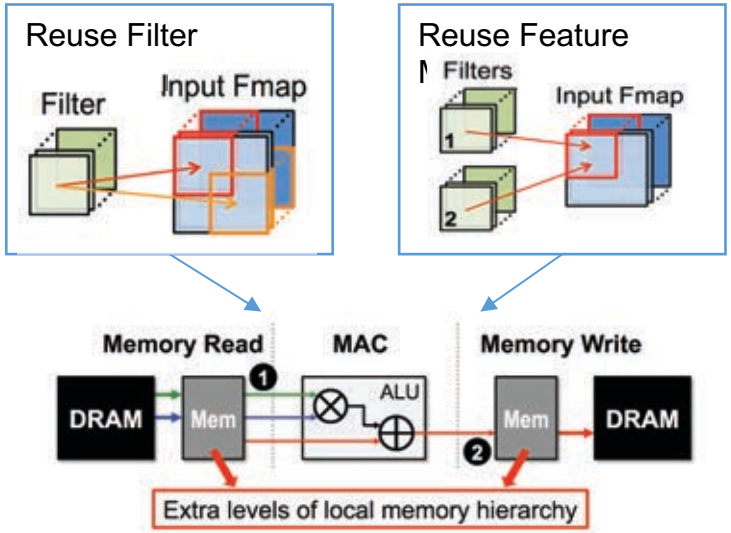
Memory access is the bottleneck for CPUs and GPUs

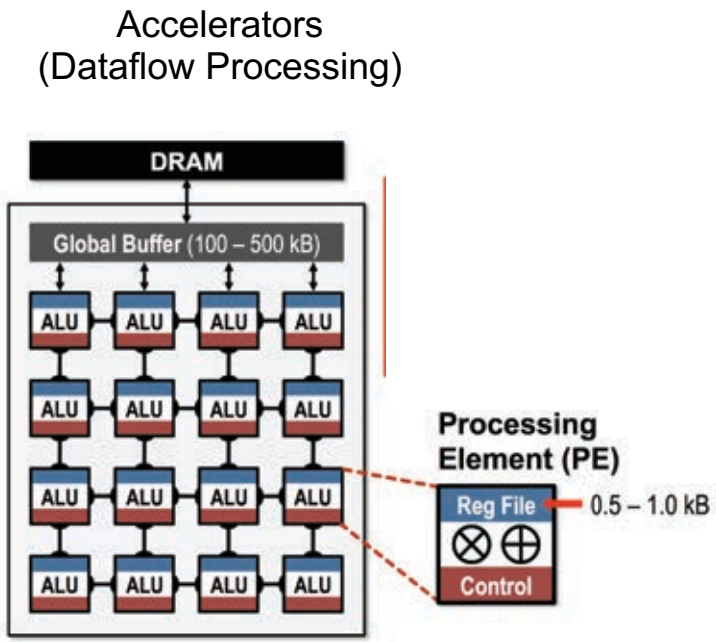


Example: AlexNet has **724M** MACs , but **2896M** DRAM accesses required!

Credit: Eyeriss [ISCA 2016]

Why NCS Matters? Relationships to DNN Accelerators





1. Data reuse reduces DRAM reads of filter/fmap by up to 500x
2. Partial sum accumulation does NOT have to access DRAM

Credit: Eyeriss [ISCA 2016]

DNN Accelerators Should Run Faster and Be More Energy-Efficient

Device	Maximum On-Chip GOPs/s	Ext. Memory (kB)	Maximum BW (GB/s)	Power (W)
CPU	17.6/core	1536	27.8	5
GPU	519	1024		
DSP	128	512		
NCS1	1000	2048	7	1
NCS2	4000	2560	11.9	1

Study Setup

Hardware:

- Android Smartphone (OnePlus 3),
- NCS1 and NCS2,
- Jetson TX2 board (w/ CUDA GPU)

Software:

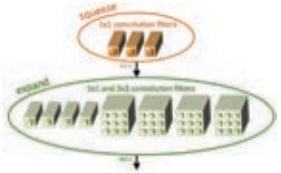
- OpenVINO SDK for NCS
- TensorFlow Lite for CPU
- Snapdragon Neural Processing Engine(SNPE) for GPU and DSP

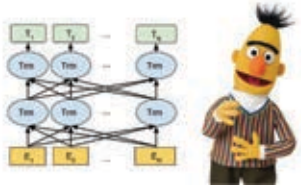
Power measurements:

- Monsoon power monitor for OnePlus 3
- MakerHawk UM34C for NCS on PC
- On-board power module for TX2

Study Methodology

DNN Workloads





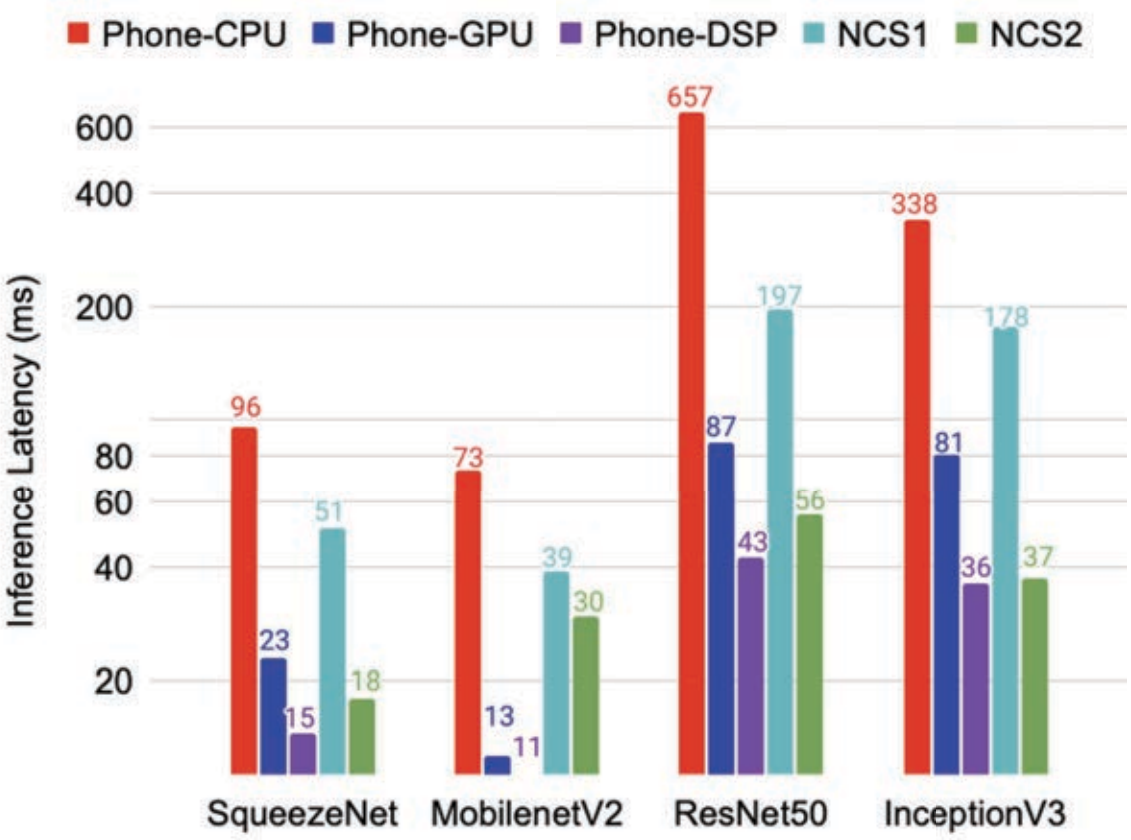
➔ 4 Popular CNNs: SqueezeNet, MobileNetV2, ResNet50, InceptionV3

➔ 4 NLP Transformer (BERT) models: BERT-tiny, BERT-mini, BERT-small, BERT-medium

Measurement Setup

- Measure inference latency and energy and average across 10 runs
- Scale the voltage and current measurements to make them comparable

GPU and DSP Are Faster than NCS1 for CNN Workloads

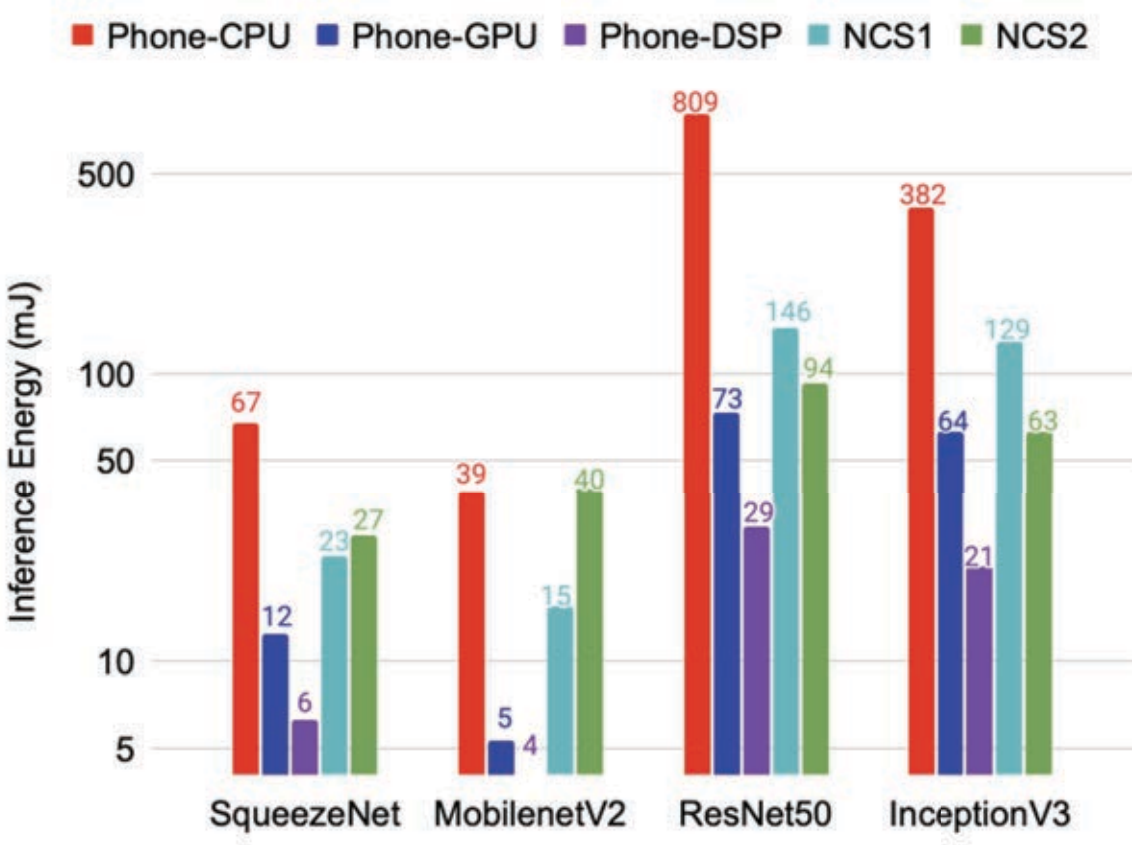


NCS1 is slower than GPU and DSP, likely because the SHAVE cores are not optimized for CNNs.

Due to dedicated neural compute engine, NCS2 consistently beats NCS1 as expected, and is often faster than GPUs.

DSP is still the fastest accelerator for all studied CNN workloads.

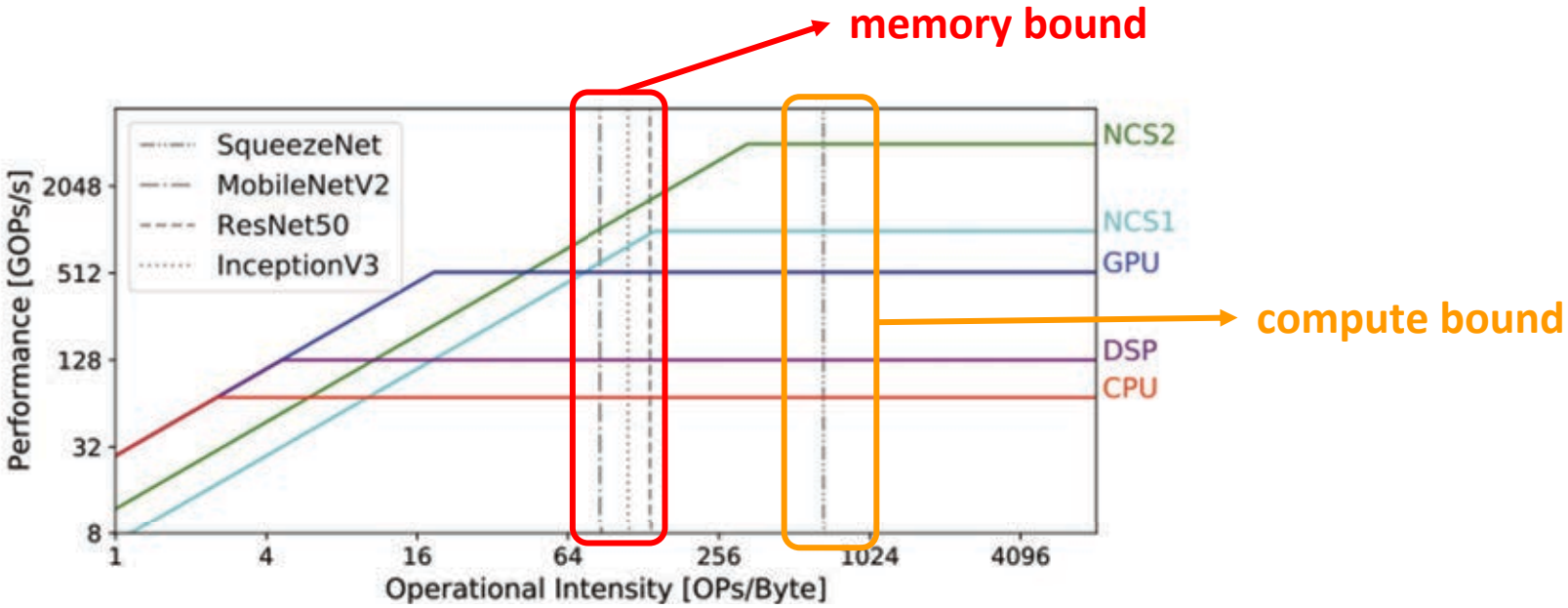
CNN: GPU and DSP Are More Energy-Efficient than NCS



The NCS1/2 consume more energy than the GPU, but much less than the CPU in all cases.

DSP consumes least energy for all studied CNN workloads.

Roofline Analysis of CNNs

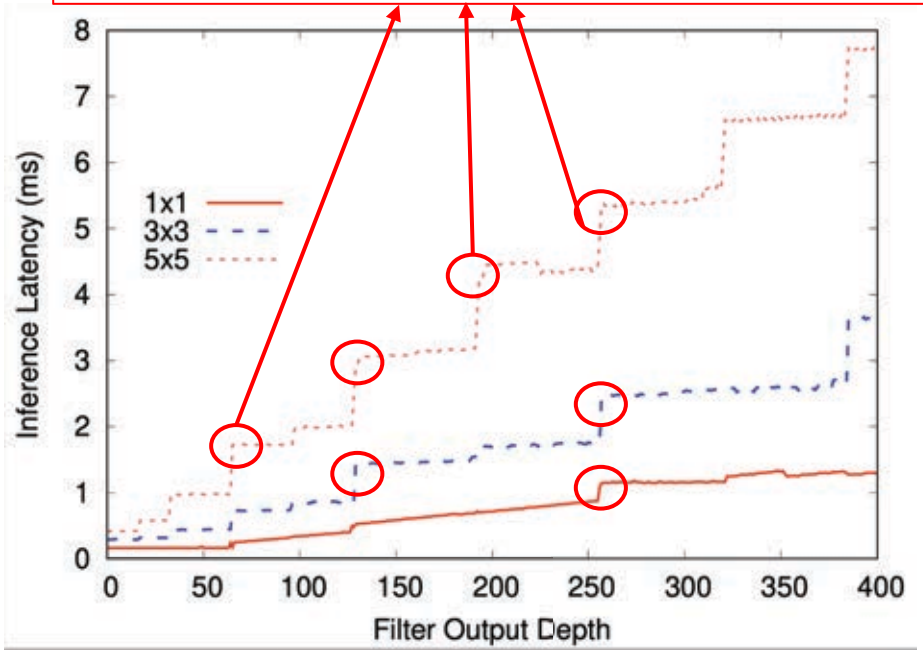


NCS devices are memory bound (except SqueezeNet)

- Existing models cannot fit in memory (even with better memory architecture)
- Parallelization does not help because the workload is memory bound

Can changing models help: NCS2 “Vectorization” Effects

Abrupt increases in latency at specific depths



Reason: CNN filter size exceeds the vector instruction parallelism in the NCS2

The effects happen much less on other devices such as CPU/GPU
(hence our focus on NCS2)

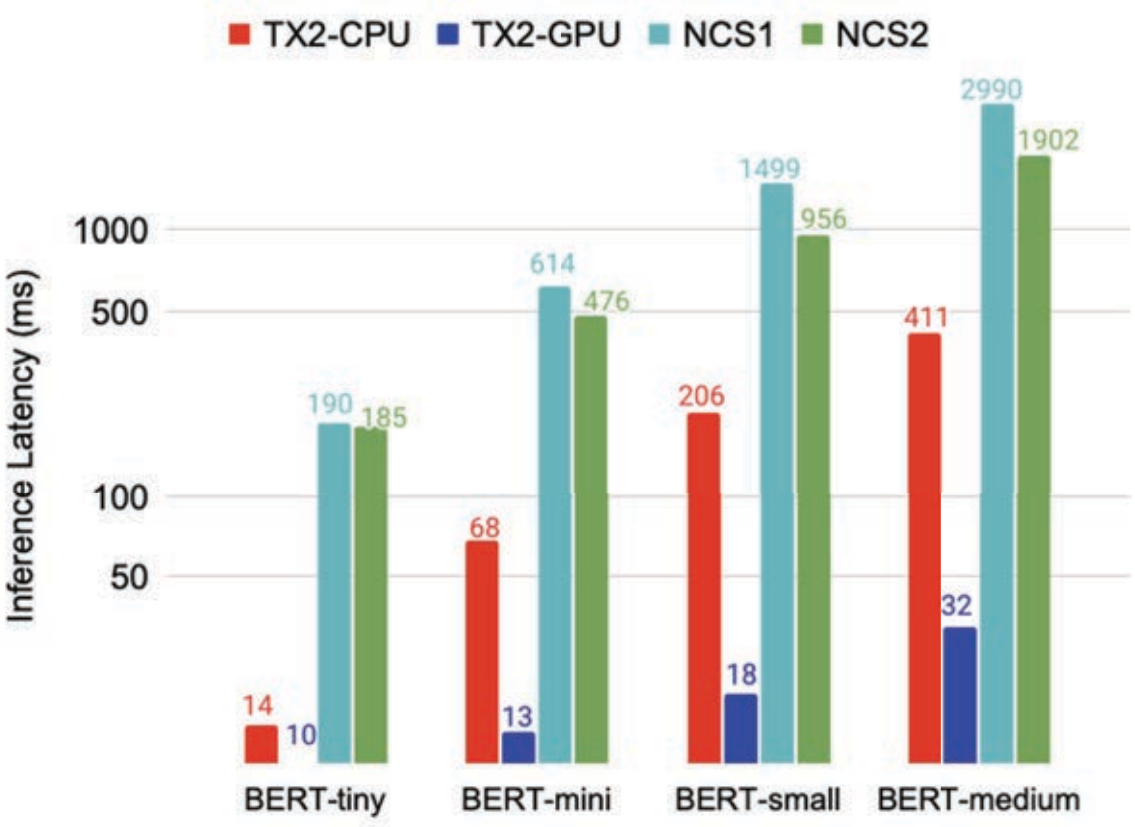
NCS-Aware CNN Optimizations

We decreased the output filter depths in InceptionV3 to the closest power of two and then retrained the model

		Latency	Power	Perf/Watt
Original	NCS2	37 ms	633 mW	43 fps/W
	DSP	**36 ms**	**587 mW**	**47 fps/W**
Pruned	NCS2	**20 ms**	592 mW	84 fps/W
	DSP	23 ms	**478 mW**	**91 fps/W**

NCS2 becomes 15% faster than DSP after pruning, even though it was 3% before pruning.

NCS Devices Are Slower than CPU/GPU for NLP Models



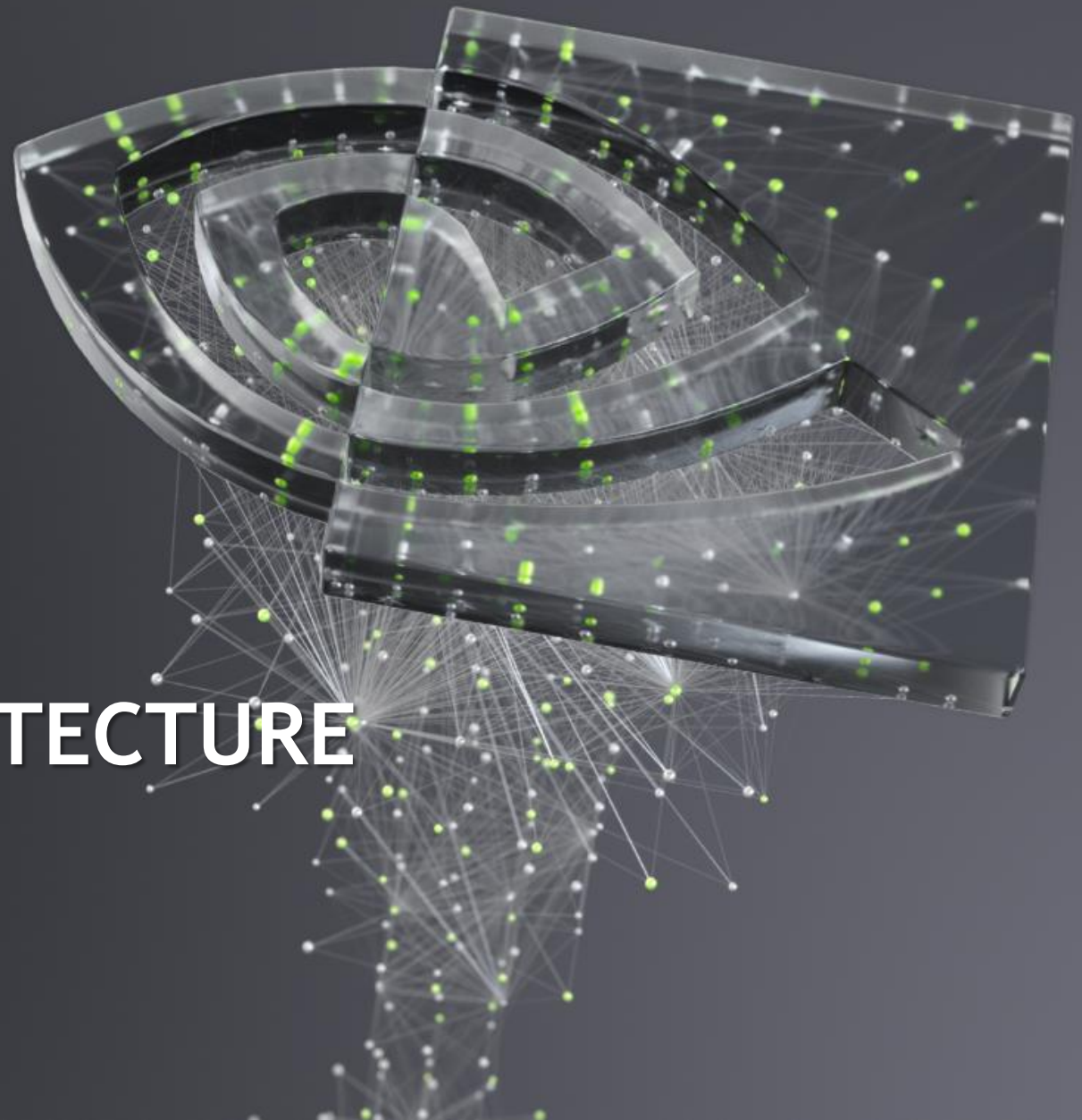
NCS1 and NCS2 are 5 ~ 10x slower than the CPU and are 20 ~ 60 slower than the GPU

Hypothesis: NCS accelerators are not optimized for NLP models like BERT;

BERT models use attention mechanism which has different properties compared to CNNs

Summary

- Performance Characterization of DNNs on commodity mobile DNN accelerators -- NCS
- Development of Preliminary Accelerator Specific Neural Network Inference Optimizations
- Major Takeaway: Mobile DNN Accelerators are not ready yet to accelerator DNNs, and are just like conventional mobile processors, only as good as the software and algorithms that drive them.



NVIDIA CUDA ARCHITECTURE

João Paulo Navarro – Solutions Architect
jpnavarro@nvidia.com

- NVIDIA history
- Why GPU and accelerated computing
- GPU architecture
- GPU data-center architecture

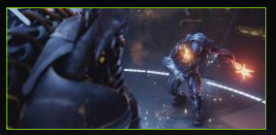
NVIDIA

GRAPHICS

GAMING

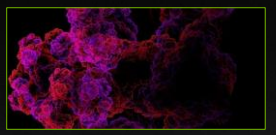
DESIGN

RENDERING



HPC

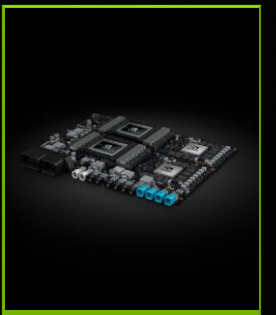
SUPERCOMPUTING



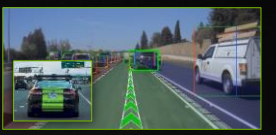
AI

AI TRAINING

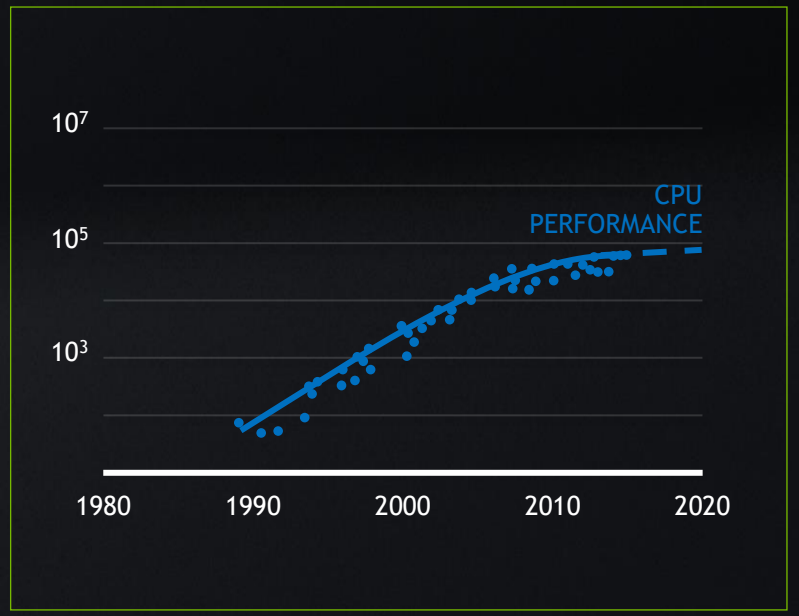
AI INFERENCE



ROBOTICS

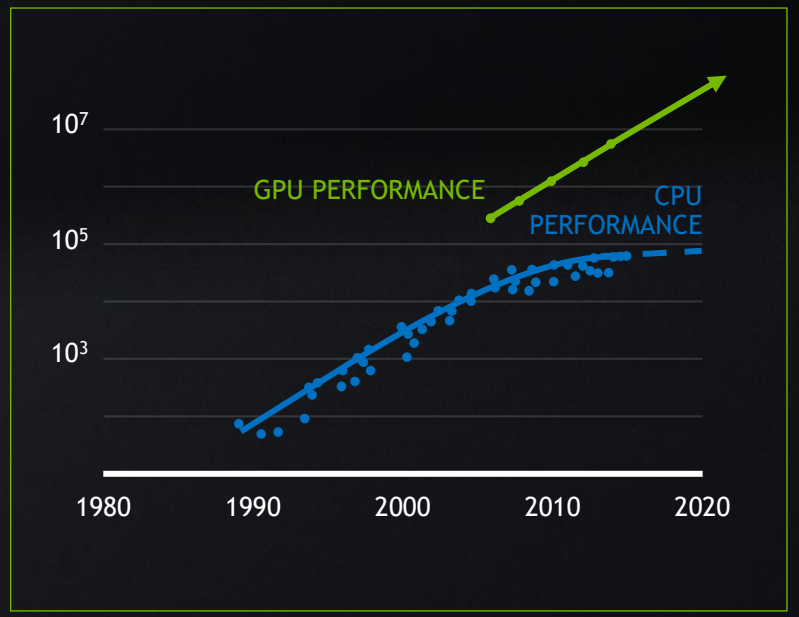


FORCES SHAPING COMPUTING

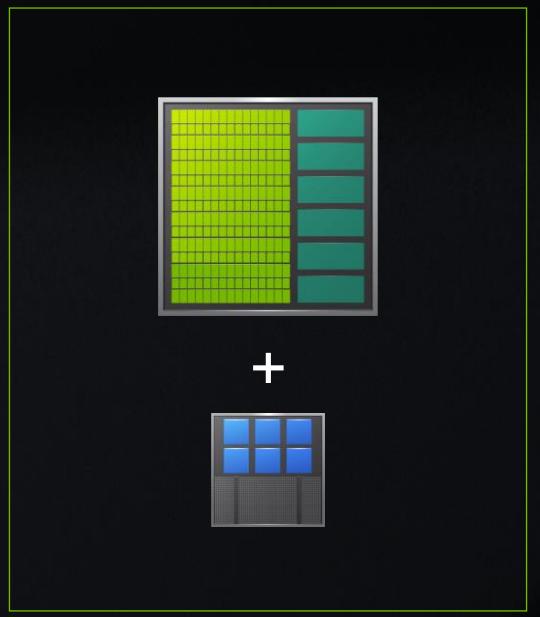


BEYOND MOORE'S LAW

FORCES SHAPING COMPUTING

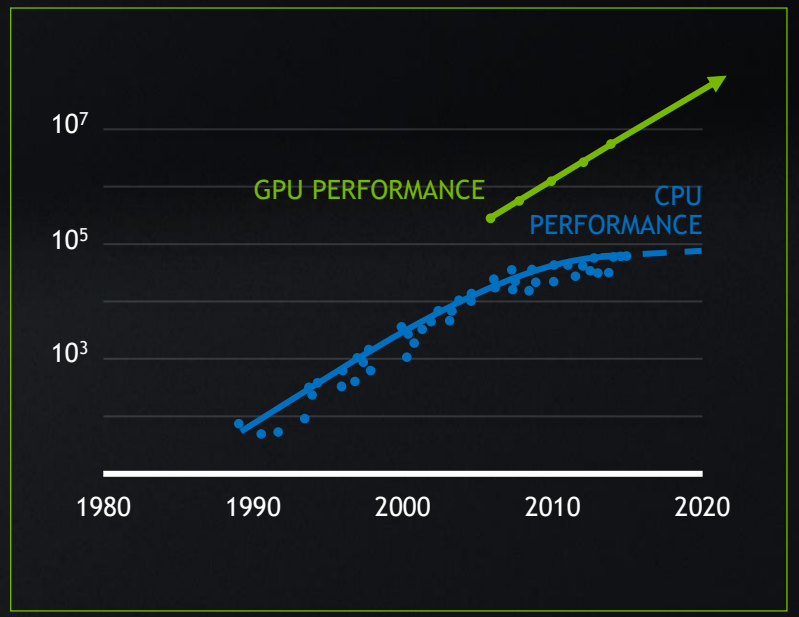


BEYOND MOORE'S LAW – 1000X EVERY 10 YEARS

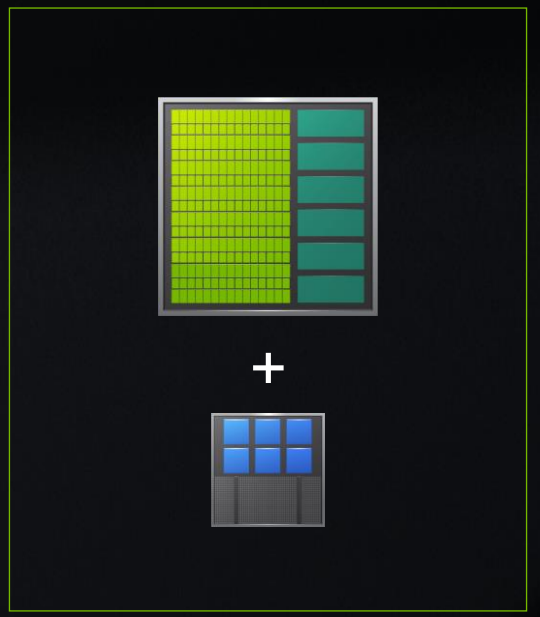


ACCELERATED COMPUTING

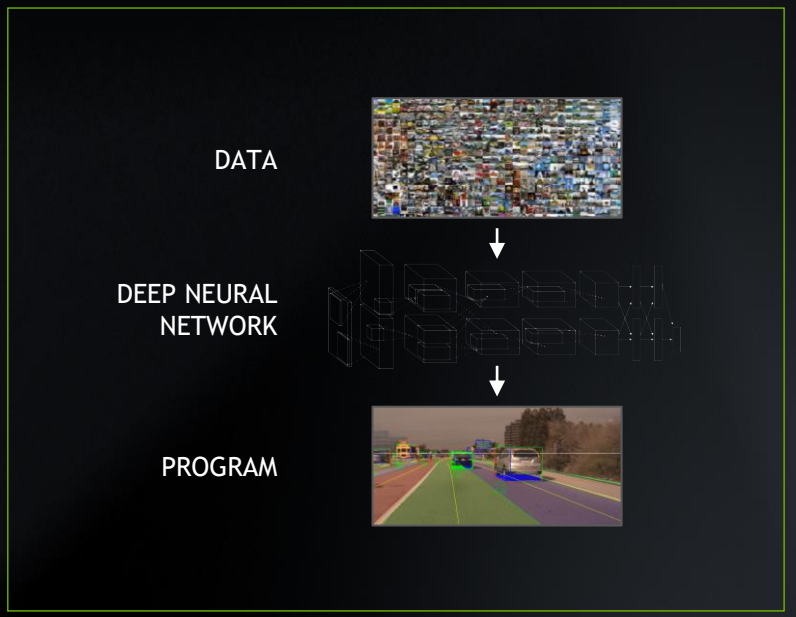
FORCES SHAPING COMPUTING



BEYOND MOORE'S LAW – 1000X EVERY 10 YEARS

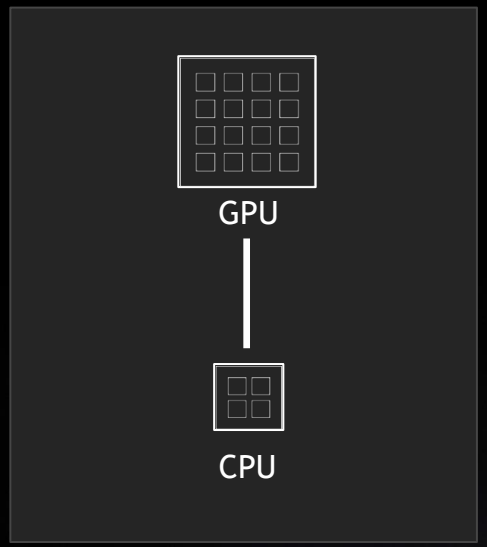


ACCELERATED COMPUTING

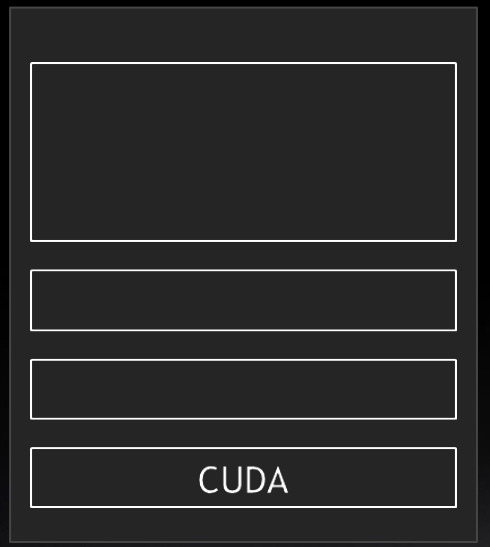


COMPUTERS WRITING SOFTWARE

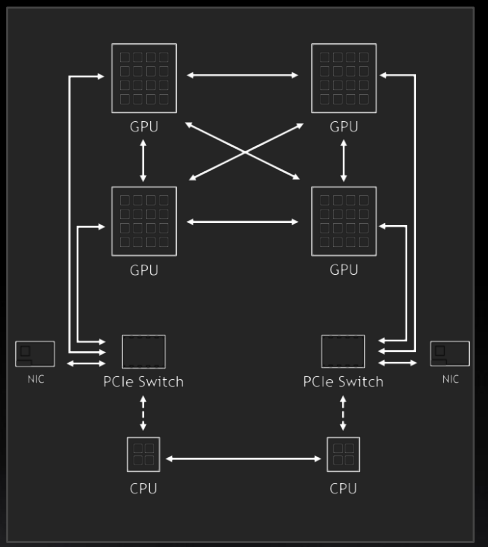
25 YEARS OF ACCELERATED COMPUTING



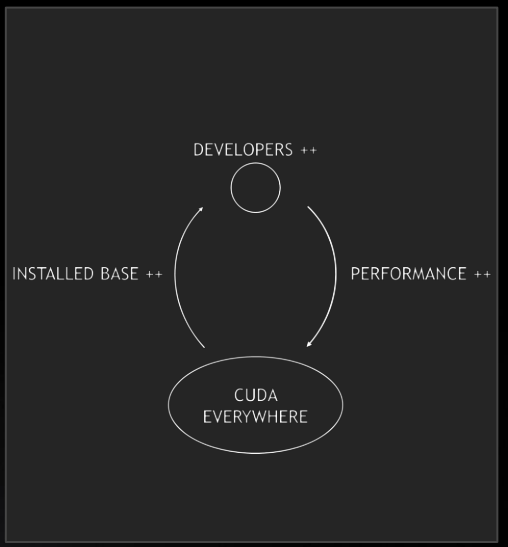
X-FACTOR SPEED UP



FULL STACK

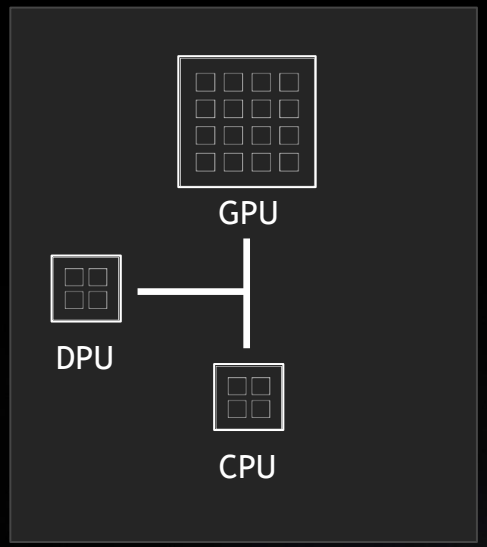


SYSTEMS

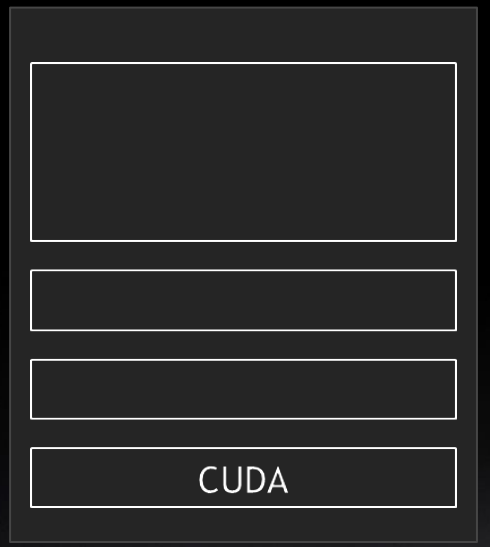


ONE ARCHITECTURE

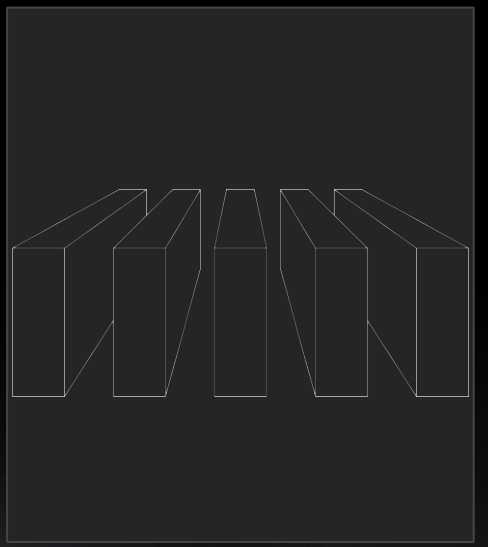
25 YEARS OF ACCELERATED COMPUTING



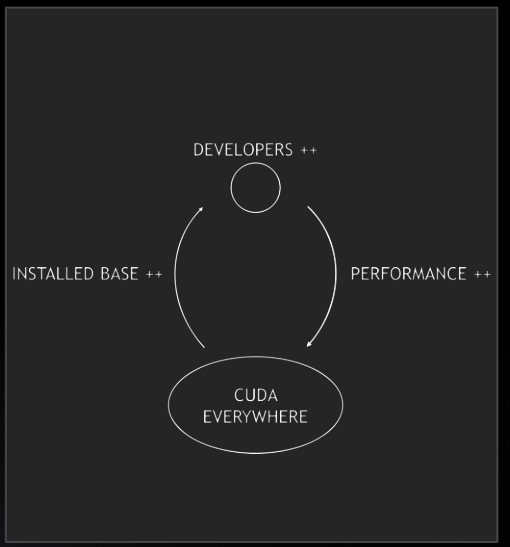
X-FACTOR SPEED UP



FULL STACK



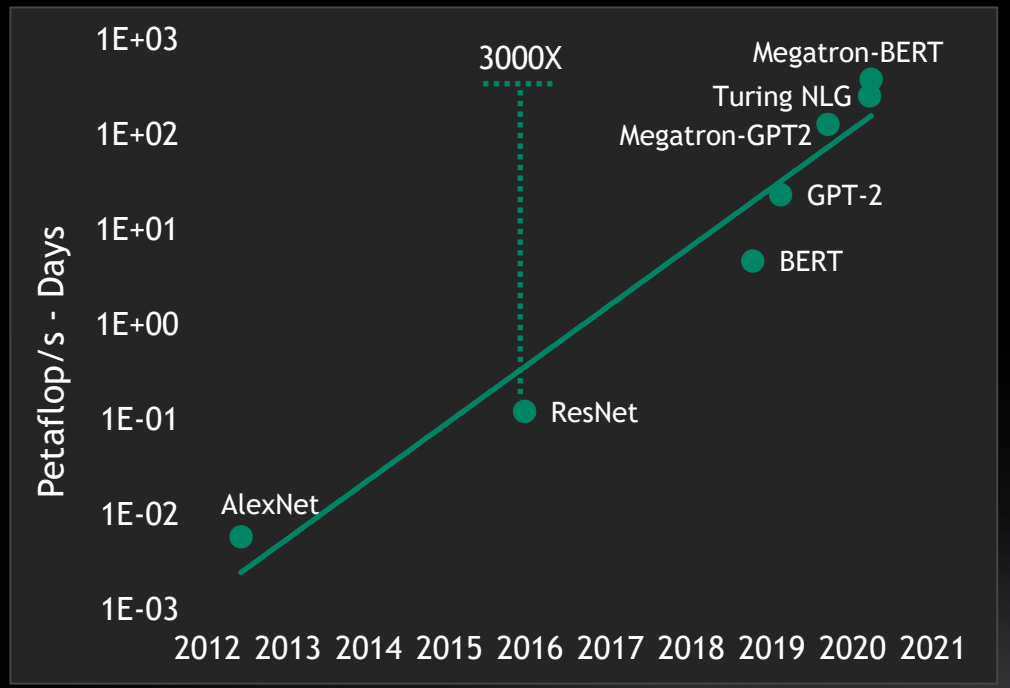
DATA-CENTER SCALE



ONE ARCHITECTURE

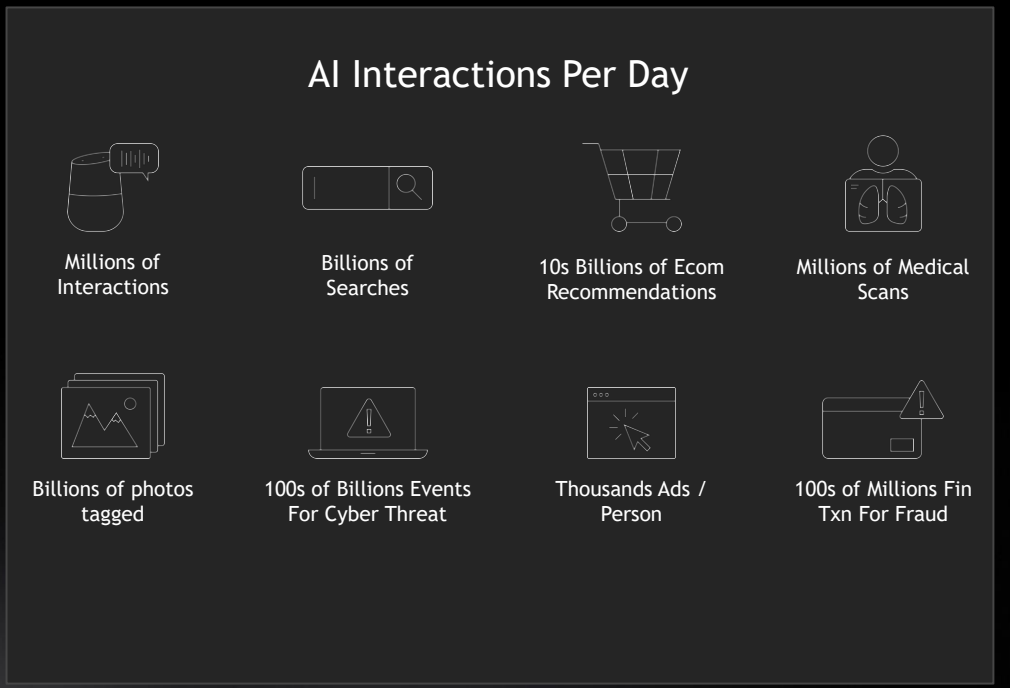
CHALLENGES: ACCELERATING BIG AND SMALL

AI Advances Demand Exponentially Higher Compute



3000X Higher Compute Required to Train Largest Models Since Volta

AI Applications Demand Distributed Pervasive Acceleration



Every AI Powered Interaction Needs Varying Amount of Compute

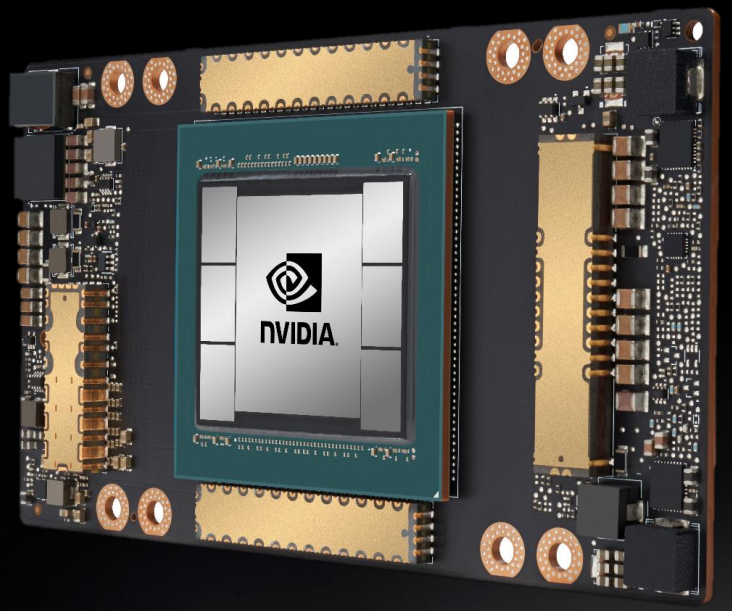
Source: OpenAI, NVIDIA

TODAY'S HYPERCONVERGED DATA CENTER

Impossible to Optimally Design Server Mix for Unpredictable Demand

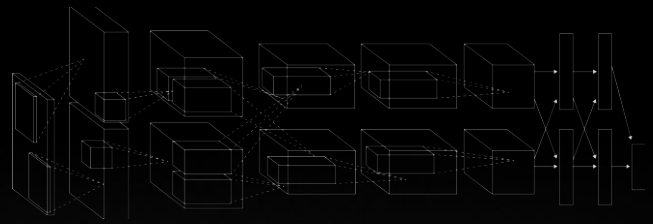
REIMAGINING THE GPU

Three Breakthroughs to Fuel the Next Era of Modern Accelerated Data Centers



20X

A GIANT LEAP IN PERFORMANCE



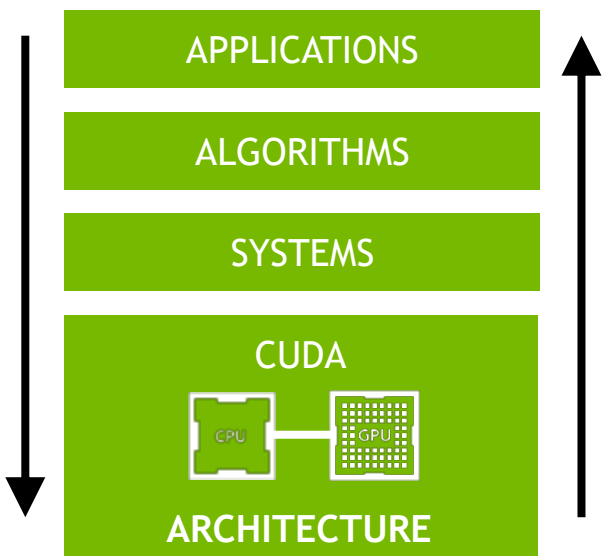
UNIFIED AI TRAINING AND INFERENCE ACCELERATION

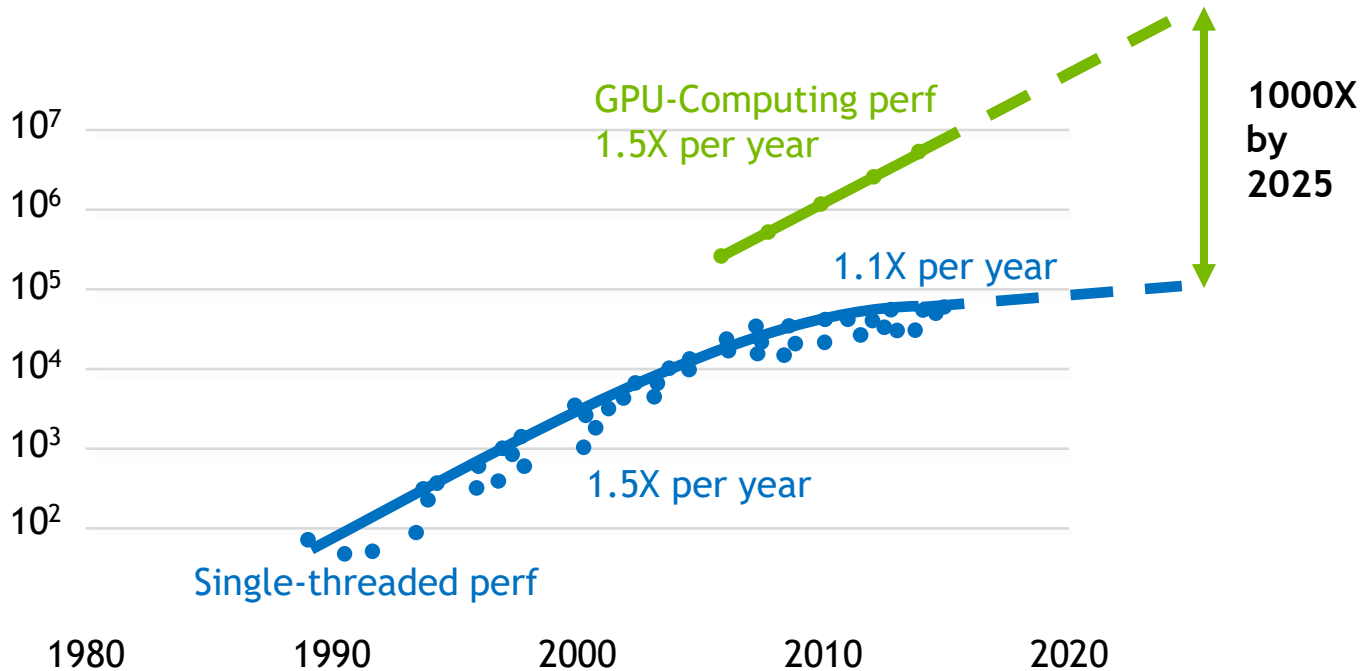
1-50

SCALABILITY FOR THE ELASTIC DATACENTER


HIGH-PERFORMANCE
COMPUTING WITH NVIDIA

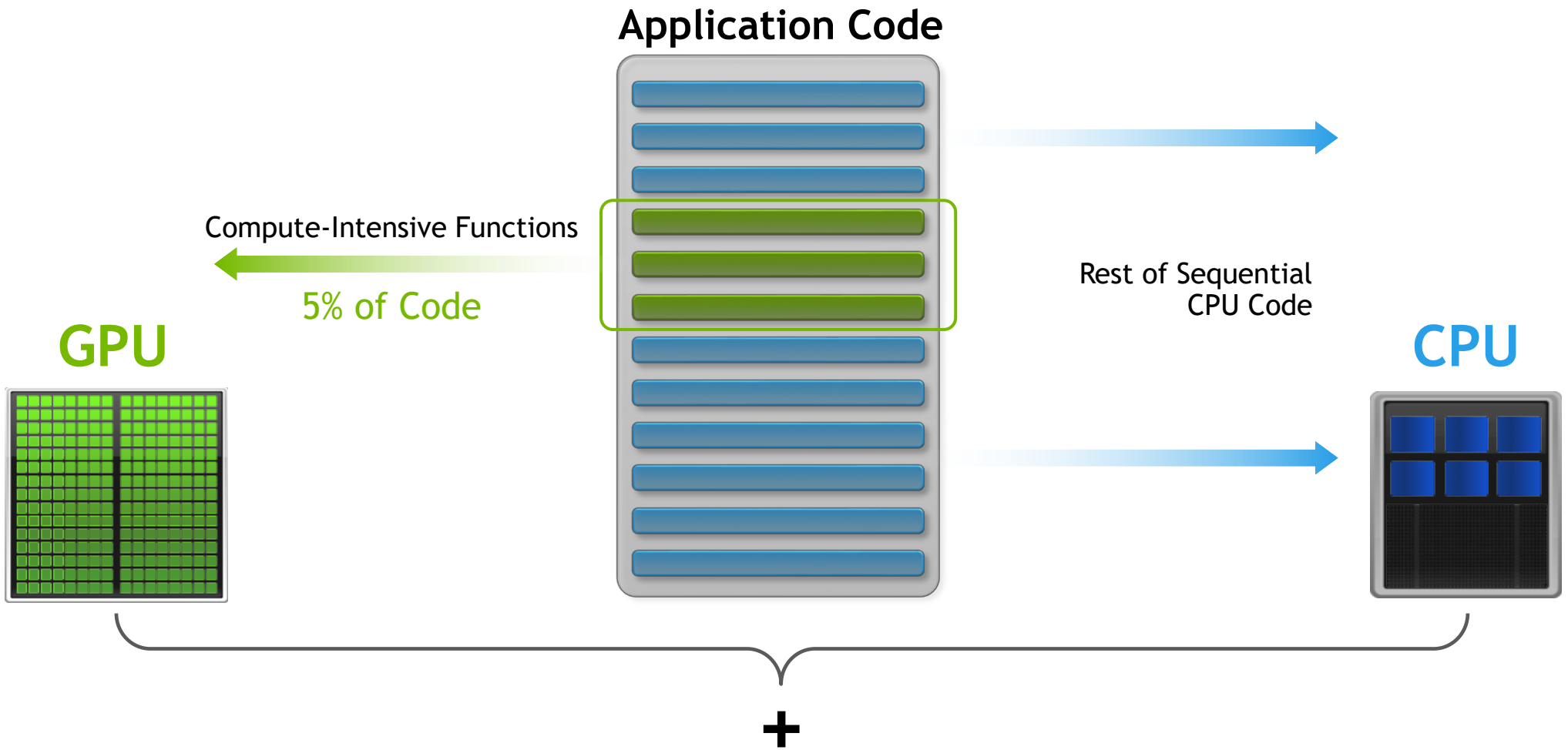
RISE OF GPU COMPUTING





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

HOW GPU ACCELERATION WORKS



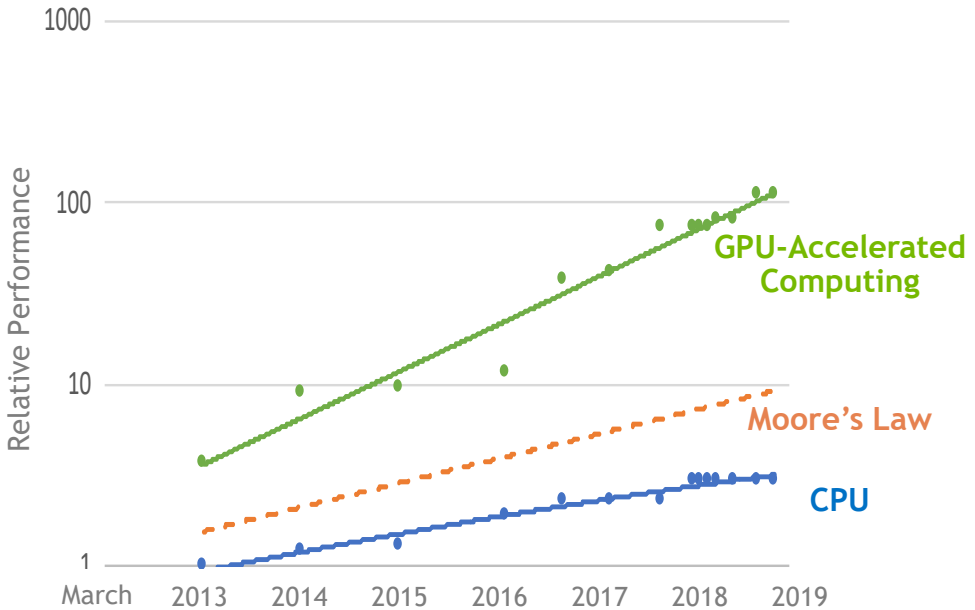
BEYOND MOORE'S LAW

Progress Of Stack In 6 Years

2013

- cuBLAS: 5.0
- cuFFT: 5.0
- cuRAND: 5.0
- cuSPARSE: 5.0
- NPP: 5.0
- Thrust: 1.5.3
- CUDA: 5.0
- Resource Mgr: r304
- Base OS: CentOS 6.2

Accelerated Server With Fermi



Measured performance of Amber, CHROMA, GTC, LAMMPS, MILC, NAMD, Quantum Espresso, SPECFEM3D

2019

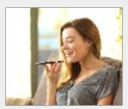
- cuBLAS: 10.0
- cuFFT: 10.0
- cuRAND: 10.0
- cuSOLVER: 10.0
- cuSPARSE: 10.0
- NPP: 10.0
- Thrust: 1.9.0
- CUDA: 10.0
- Resource Mgr: r384
- Base OS: Ubuntu 16.04

Accelerated Server with Volta

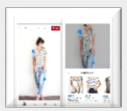
NVIDIA DATA CENTER PLATFORM

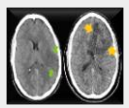
Single Platform Drives Utilization and Productivity

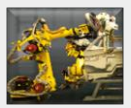
CUSTOMER USE CASES


Speech

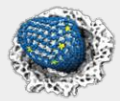
Translate

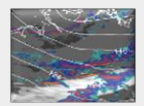

Recommender

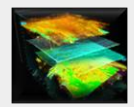

Healthcare


Manufacturing


Finance


Molecular Simulations


Weather Forecasting


Seismic Mapping


Creative & Technical


Knowledge Workers

CONSUMER INTERNET & INDUSTRY APPLICATIONS

SCIENTIFIC APPLICATIONS

VIRTUAL GRAPHICS

APPS & FRAMEWORKS

python · TensorFlow · mxnet · Chainer · ONNX · RAPIDS · PYTORCH

Amber · NAMD · +600 Applications

CATIA · Ps · AUTODESK 3DS MAX · Windows 10

CUDA-X & NVIDIA SDKs

MACHINE LEARNING: cuDF | cuML | cuGRAPH

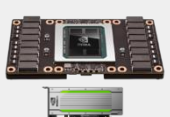
DEEP LEARNING: cuDNN | CUTLASS | TensorRT

HPC: OpenACC | cuFFT

VIRTUAL GPU: vDWS | vPC | vAPPS

CUDA & CORE LIBRARIES - cuBLAS | NCCL

TESLA GPUs & SYSTEMS


TESLA GPU


NVIDIA DGX FAMILY

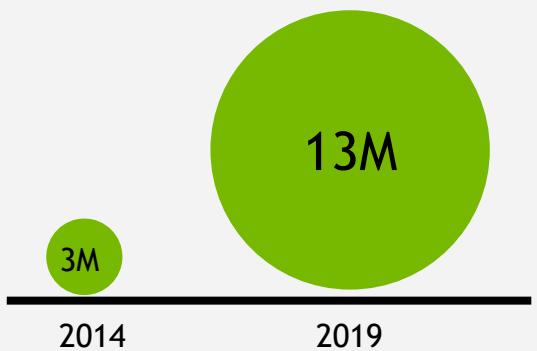

NVIDIA HGX


EVERY OEM


EVERY MAJOR CLOUD

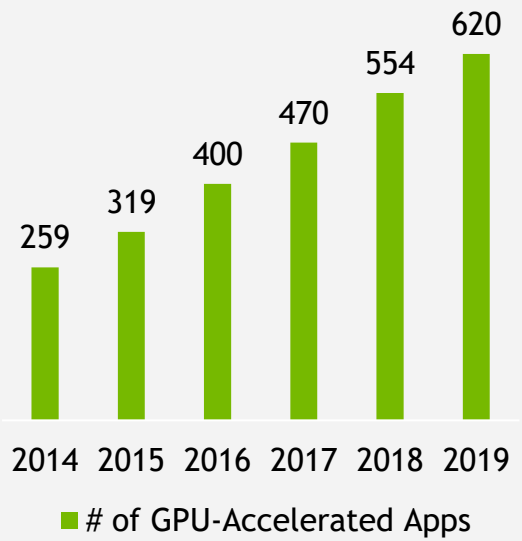
MOST ADOPTED PLATFORM FOR ACCELERATING HPC

13M CUDA Downloads



4X IN 5 YEARS

600+ Applications Accelerated



ALL TOP 15 APPLICATIONS ACCELERATED

125 Systems on Top 500

World's #1 Summit: 149 PF

World's #2 Sierra: 95 PF

Europe's #1 Piz Daint: 21 PF

Japan's #1 ABCI: 20 PF

Industrial #1 Total Pangea 3: 18 PF

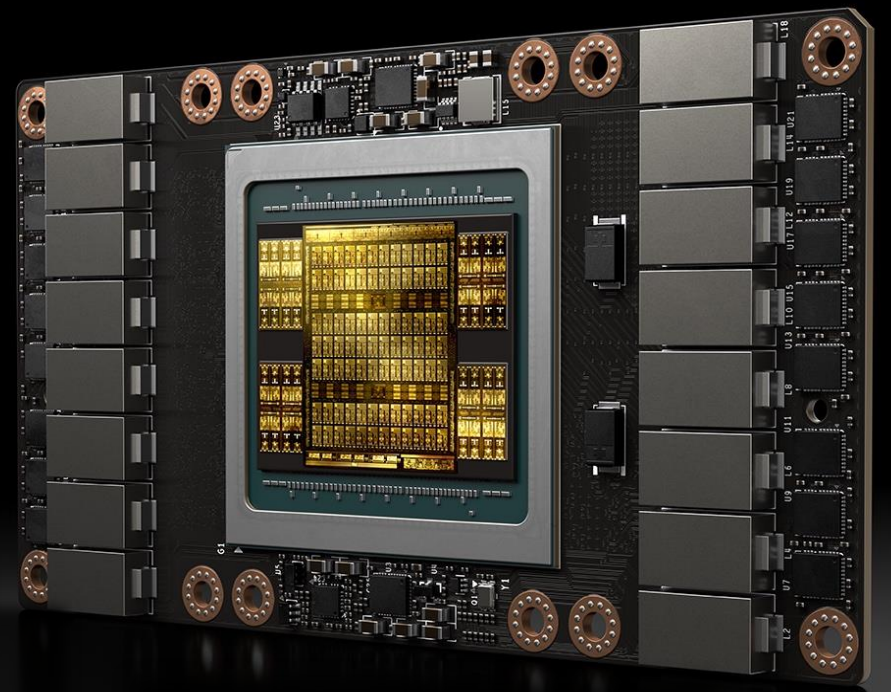
NEW HIGHS IN TOP 500 LIST

NVIDIA POWERS WORLD'S FASTEST SUPERCOMPUTER

Summit Becomes First System To Scale The 100 Petaflops Milestone

122 PF HPC

3 EF AI



27,648
Volta Tensor Core GPUs

NVIDIA POWERS FASTEST SUPERCOMPUTERS IN US, EUROPE, JAPAN, INDUSTRY

17 of World's 20 Most Energy-efficient Supercomputers

ORNL Summit
World's Fastest
27,648 GPUs| 122 PF

LLNL Sierra
US 2nd Fastest
17,280 GPUs| 72 PF

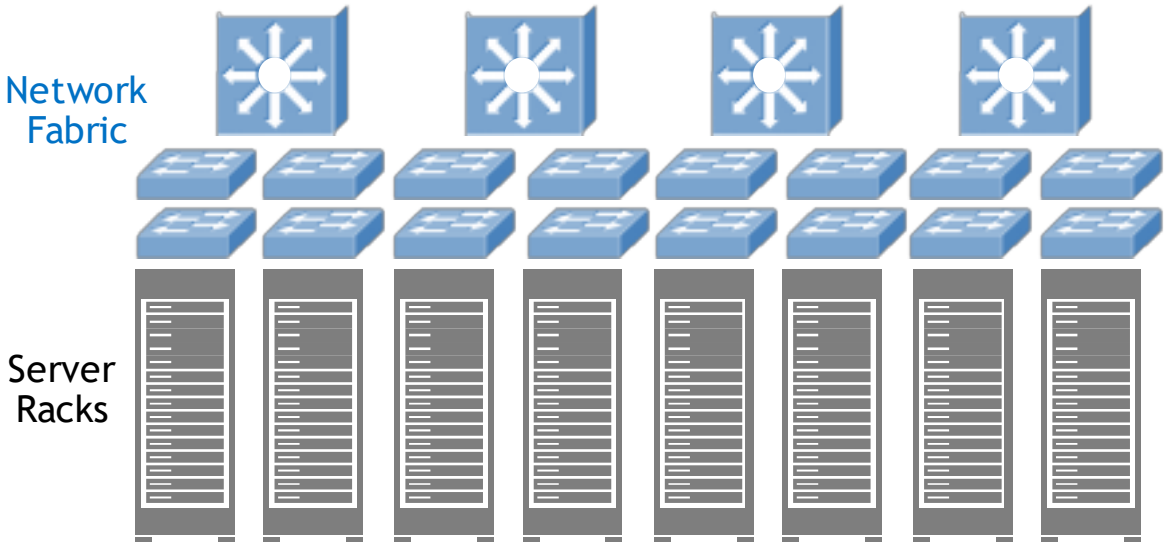
ABCI
Japan's Fastest
4,352 GPUs| 20 PF

Piz Daint
Europe's Fastest
5,320 GPUs| 20 PF

ENI HPC4
Fastest Industrial
3,200 GPUs| 12 PF

WEAK NODES

Lots of Nodes Interconnected with
Vast Network Overhead



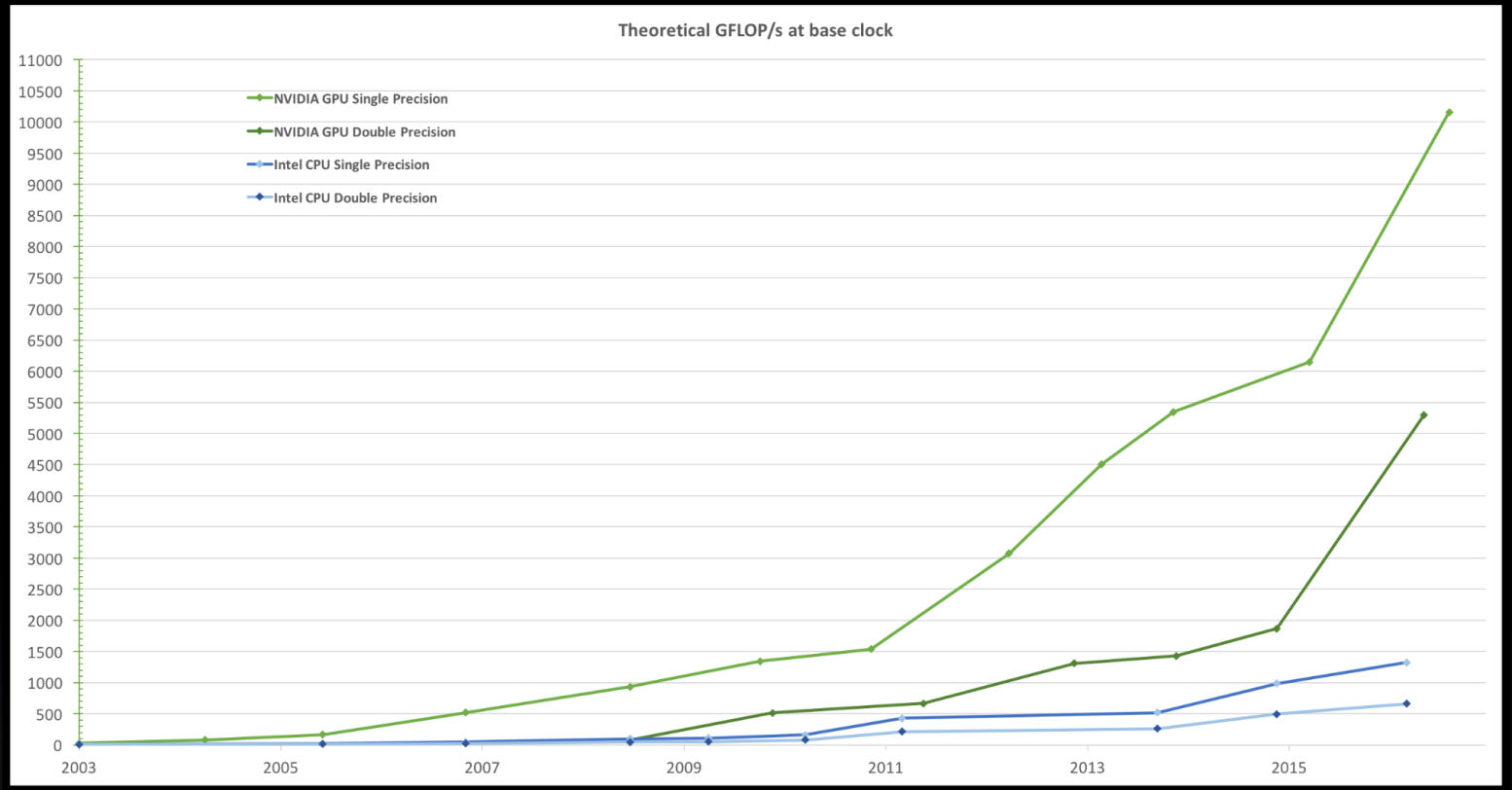
STRONG NODES

Few Lightning-Fast Nodes with
Performance of Hundreds of Weak Nodes


GPU ARCHITECTURE

SINGLE AND DOUBLE PRECISION THROUGHPUT

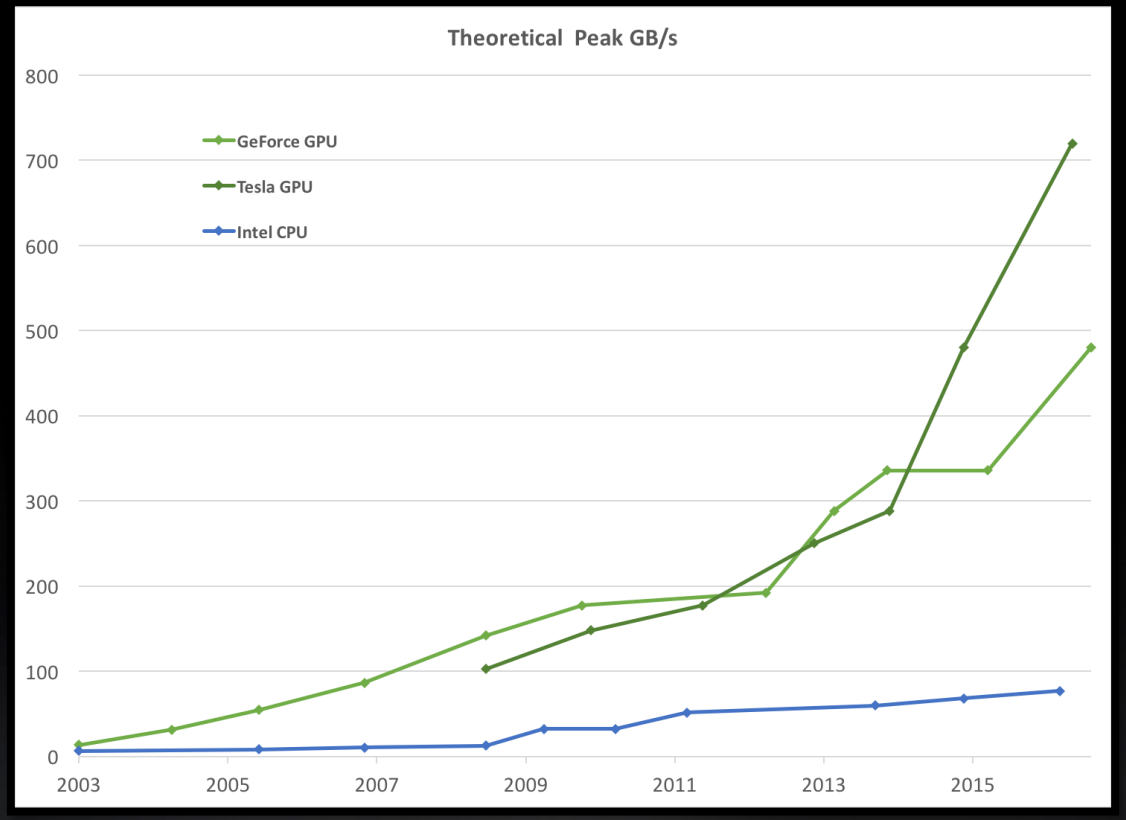
Floating-Point Operations per Second for the CPU and GPU



NVIDIA

SINGLE AND DOUBLE PRECISION THROUGHPUT

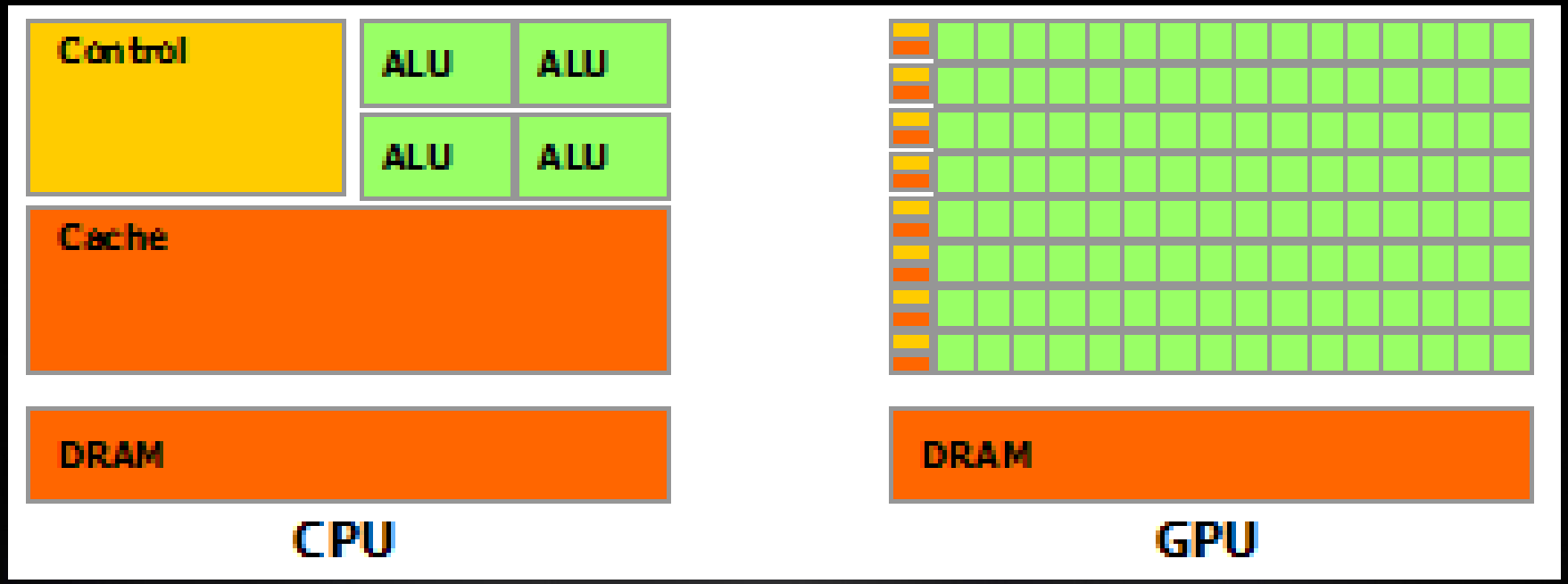
Memory Bandwidth for the CPU and GPU



Why does GPU have memory?

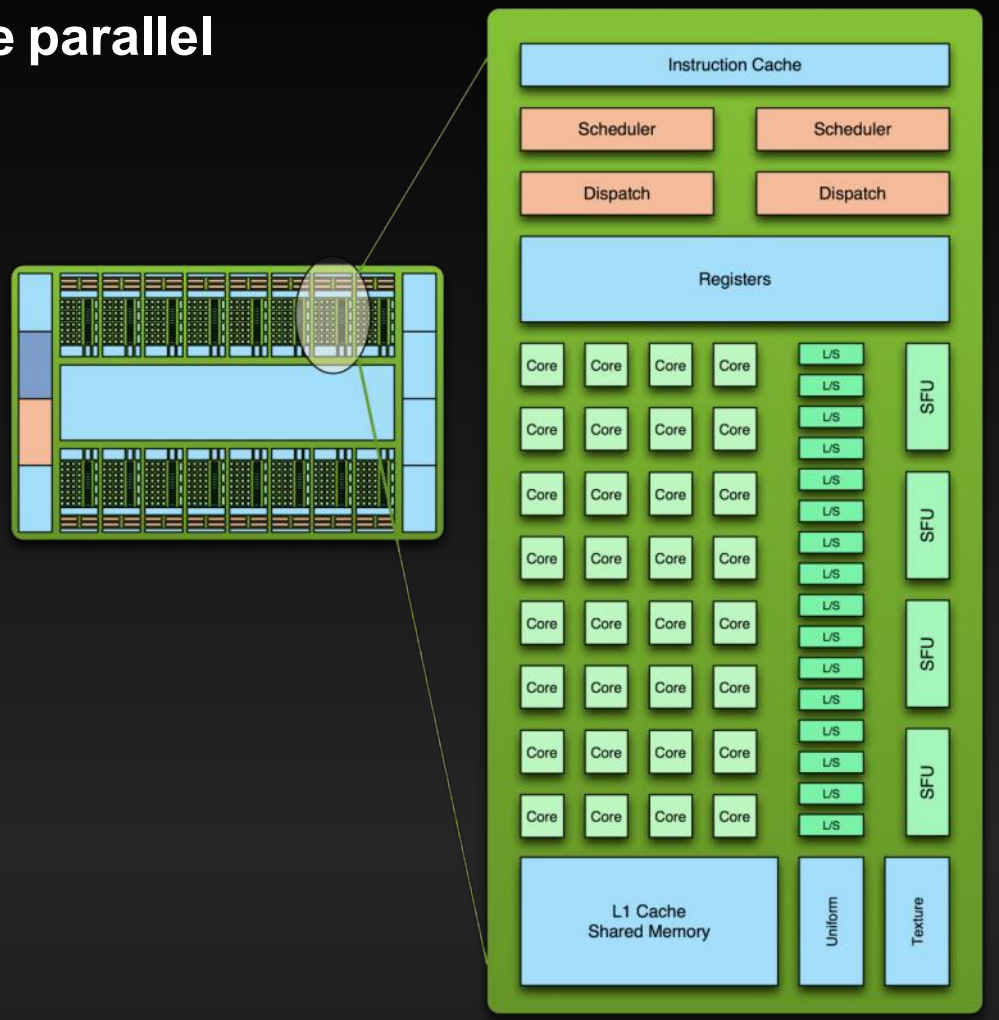
CPU VS GPU

The GPU Devotes More Transistors to Data Processing



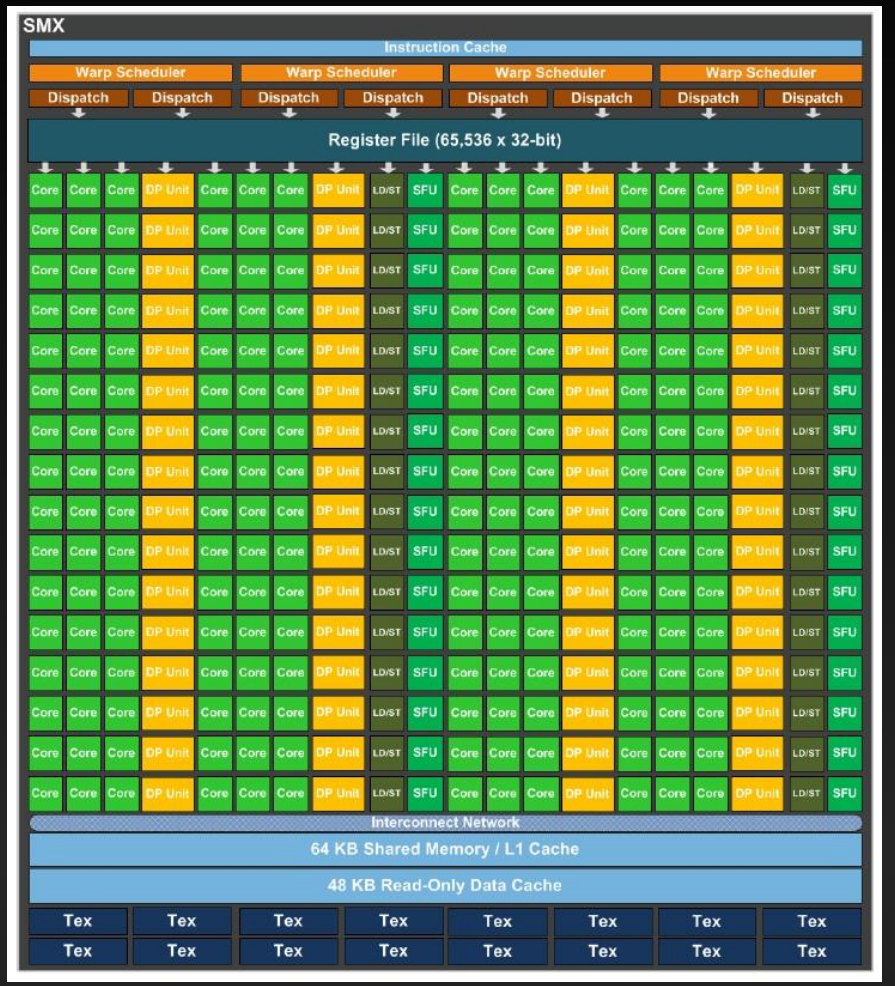
20-Series Architecture (Fermi)

- 512 *Scalar Processor (SP) cores* execute parallel thread instructions
- 16 Streaming *Multiprocessors (SMs)* each contains
  - 32 scalar processors
    - 32 fp32 / int32 ops / clock,
    - 16 fp64 ops / clock
  - 4 Special Function Units (SFUs)
  - Shared register file (128KB)
  - *48 KB / 16 KB Shared memory*
  - 16KB / 48 KB L1 data cache



Kepler cc 3.5 SM (GK110)

- “SMX” (enhanced SM)
- 192 SP units (“cores”)
- 64 DP units
- LD/ST units, 64K registers
- 4 warp schedulers
- Each warp scheduler is dual-issue capable
- K20: 13 SMX’s, 5GB
- K20X: 14 SMX’s, 6GB
- K40: 15 SMX’s, 12GB



Maxwell/Pascal cc5.2, cc6.1 SM

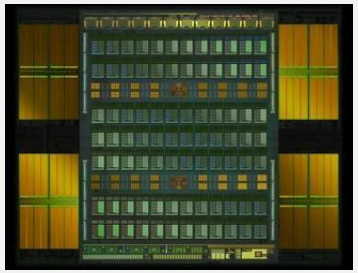
- "SMM" (enhanced SM)
- 128 SP units ("cores")
- 4 DP units
- LD/ST units
- cc 6.1: INT8
- 4 warp schedulers
- Each warp scheduler is dual-issue capable
- M40: 24 SMM's, 12/24GB
- P40: 30 SM's, 24GB
- P4: 20 SM's, 8GB

Pascal/Volta cc6.0/7.0

- 64 SP units (“cores”)
- 32 DP units
- LD/ST units
- FP16 @ 2x SP rate
- cc7.0: TensorCore
- 4 warp schedulers
- Each warp scheduler is dual-issue capable
- P100: 50 SM’s, 16GB
- V100: 80 SM’s, 16/32GB

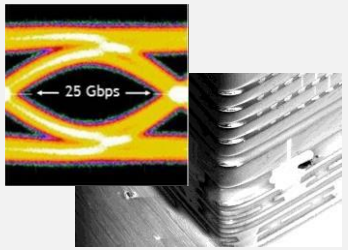
INTRODUCING TESLA V100

Volta Architecture



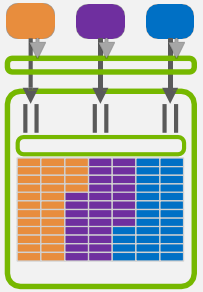
Most Productive GPU

Improved NVLink & HBM2



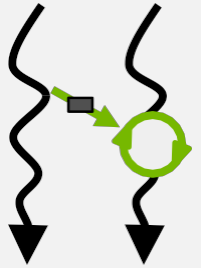
Efficient Bandwidth

Volta MPS



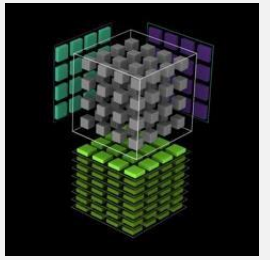
Inference Utilization

Improved SIMT Model



New Algorithms

Tensor Core



120 Programmable TFLOPS Deep Learning

The Fastest and Most Productive GPU for Deep Learning and HPC

TESLA V100

21B transistors
815 mm²

80 SM
5120 CUDA Cores
640 Tensor Cores

16 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

*full GV100 chip contains 84 SMs

VOLTA GV100 SM

	GV100
FP32 units	64
FP64 units	32
INT32 units	64
Tensor Cores	8
Register File	256 KB
Unified L1/Shared memory	128 KB
Active Threads	2048

VOLTA GV100 SM

REDESIGNED FOR PRODUCTIVITY

Twice the schedulers

Simplified Issue Logic

Large, fast L1 cache

Improved SIMT model

Tensor acceleration

=

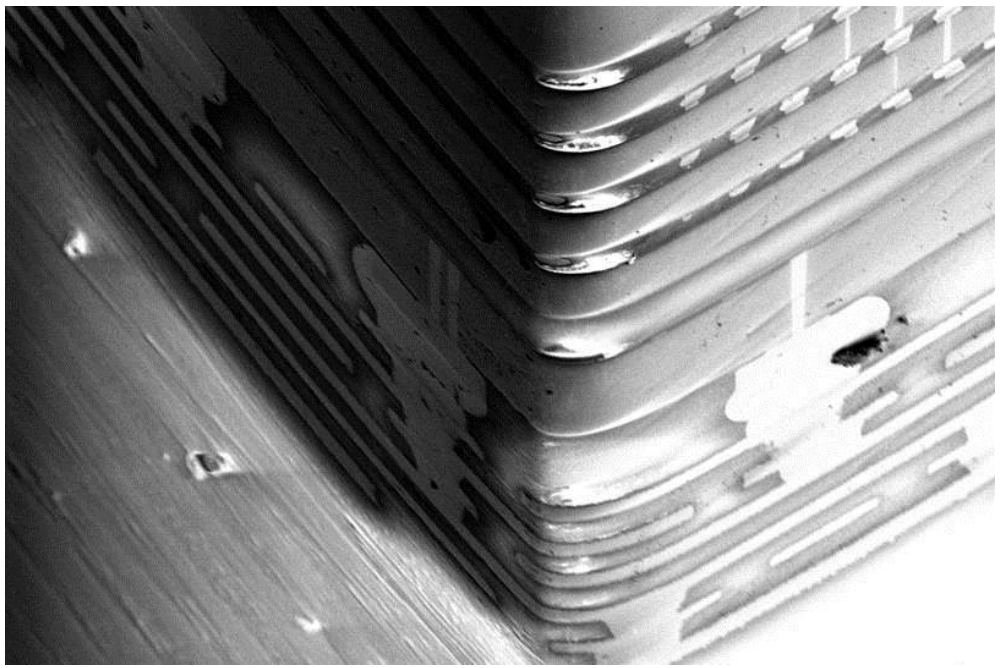
The easiest SM to program yet

GPU PERFORMANCE COMPARISON

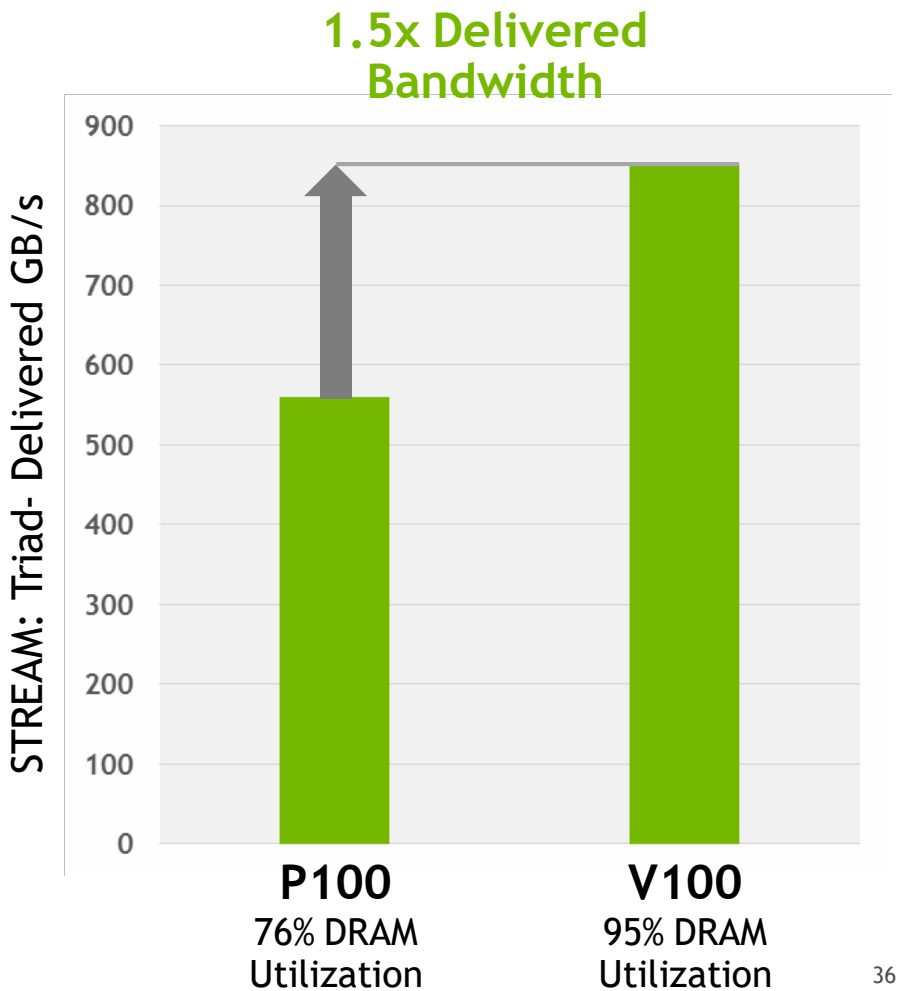
	P100	V100	Ratio
Training acceleration	10 TOPS	120 TOPS	12x
Inference acceleration	21 TFLOPS	120 TOPS	6x
FP64/FP32	5/10 TFLOPS	7.5/15 TFLOPS	1.5x
HBM2 Bandwidth	720 GB/s	900 GB/s	1.2x
NVLink Bandwidth	160 GB/s	300 GB/s	1.9x
L2 Cache	4 MB	6 MB	1.5x
L1 Caches	1.3 MB	10 MB	7.7x

MEMORY HIERARCHY

NEW HBM2 MEMORY ARCHITECTURE



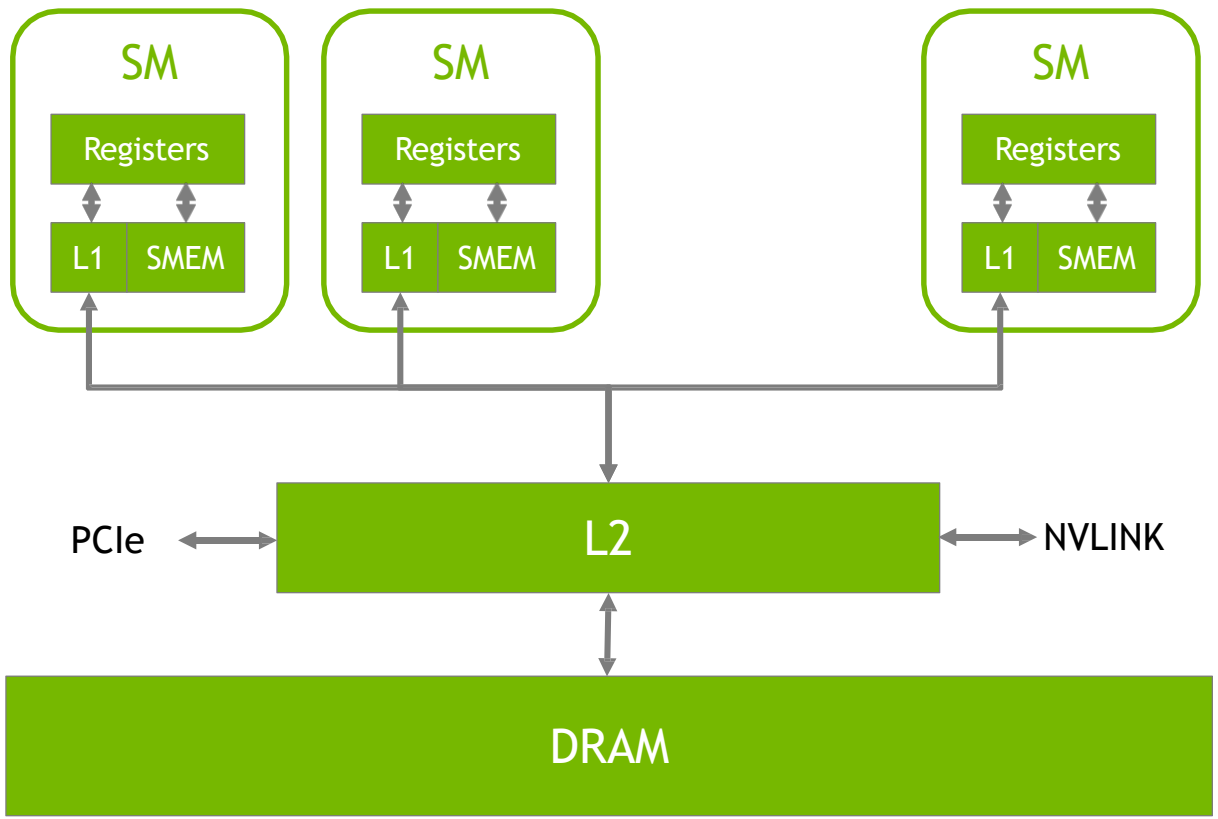
HBM2 stack



V100 measured on pre-production hardware.

VOLTA MEMORY SUBSYSTEM

Tesla V100



80 Streaming Multiprocessors
256KB register file (20 MB)

Unified Shared Mem / L1 Cache
128KB, variable split (10MB Total,
14 TB/s), Volta caches L1 writes

6 MB L2 Cache, L2 is write back

16/32 GB HBM2 (900 GB/s)

L1, L2 CACHES

Why do GPU have caches?

Caches on GPUs can help with:

"Smoothing" irregular, unaligned access patterns

Caching common data accessed by many threads

Faster register spills, local memory

Can help in codes that don't use shared memory

SHARED MEMORY

Scratch-pad memory on each SM
User-managed cache, hardware does not evict data
Data written to SMEM stays there until this the code overwrites the data or threadblock finishes execution

Useful for:
Storing frequently-accessed data, to reduce DRAM accesses
Communication among threads of a threadblock

Performance benefits compared to DRAM:
20-40x lower latency
~15x higher bandwidth

UNIFIED SHARED MEM / L1 CACHE

Variable split

Volta: 6 possible smem / L1 splits
96KB / 32KB
64KB / 64KB
32KB / 96KB
16KB / 112KB
8KB / 120KB
0KB /128 KB

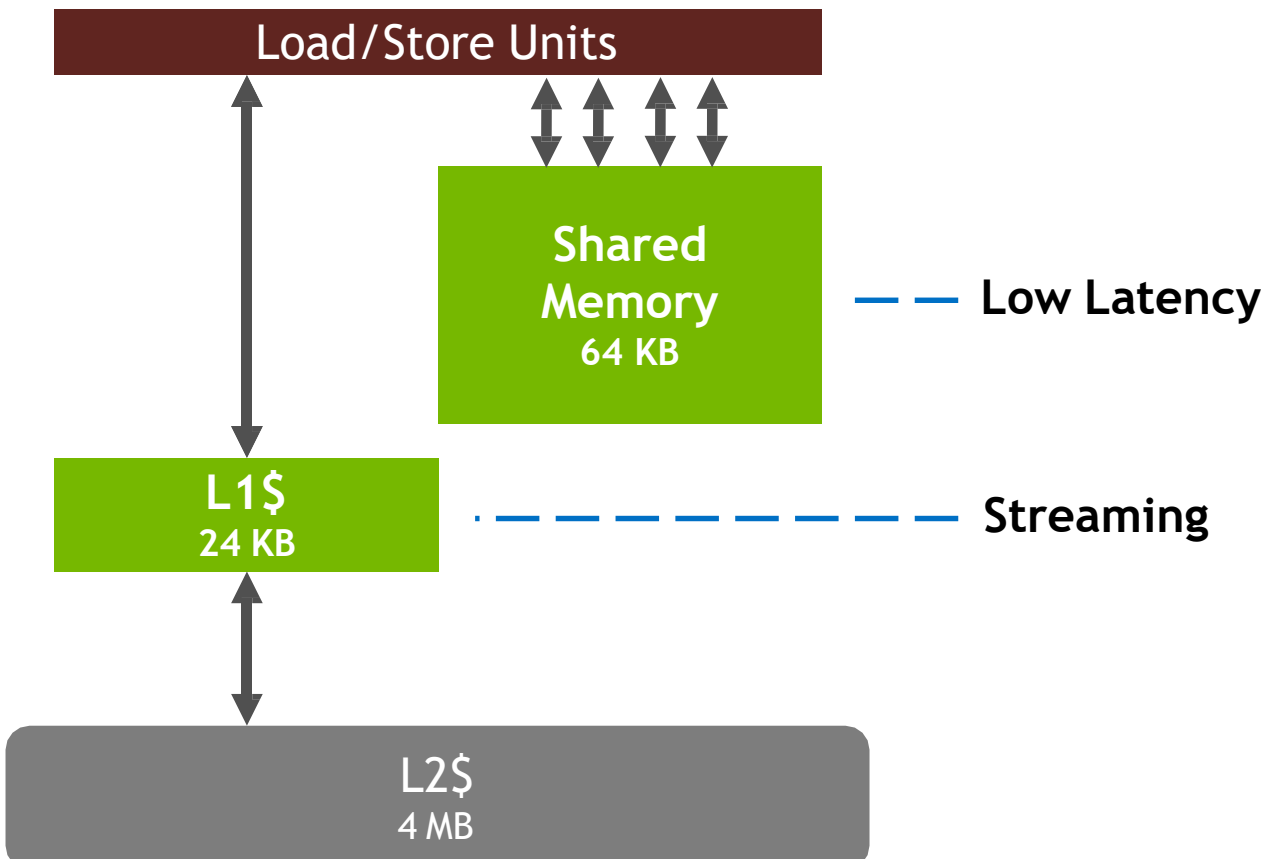
Turing: 2 possible smem / L1 splits
64KB / 32KB
32KB / 64KB

How to specify the L1 / Smem split:
cudaFuncSetAttribute (MyKernel, cudaFuncAttributePreferredSharedMemoryCarveout, carveout);

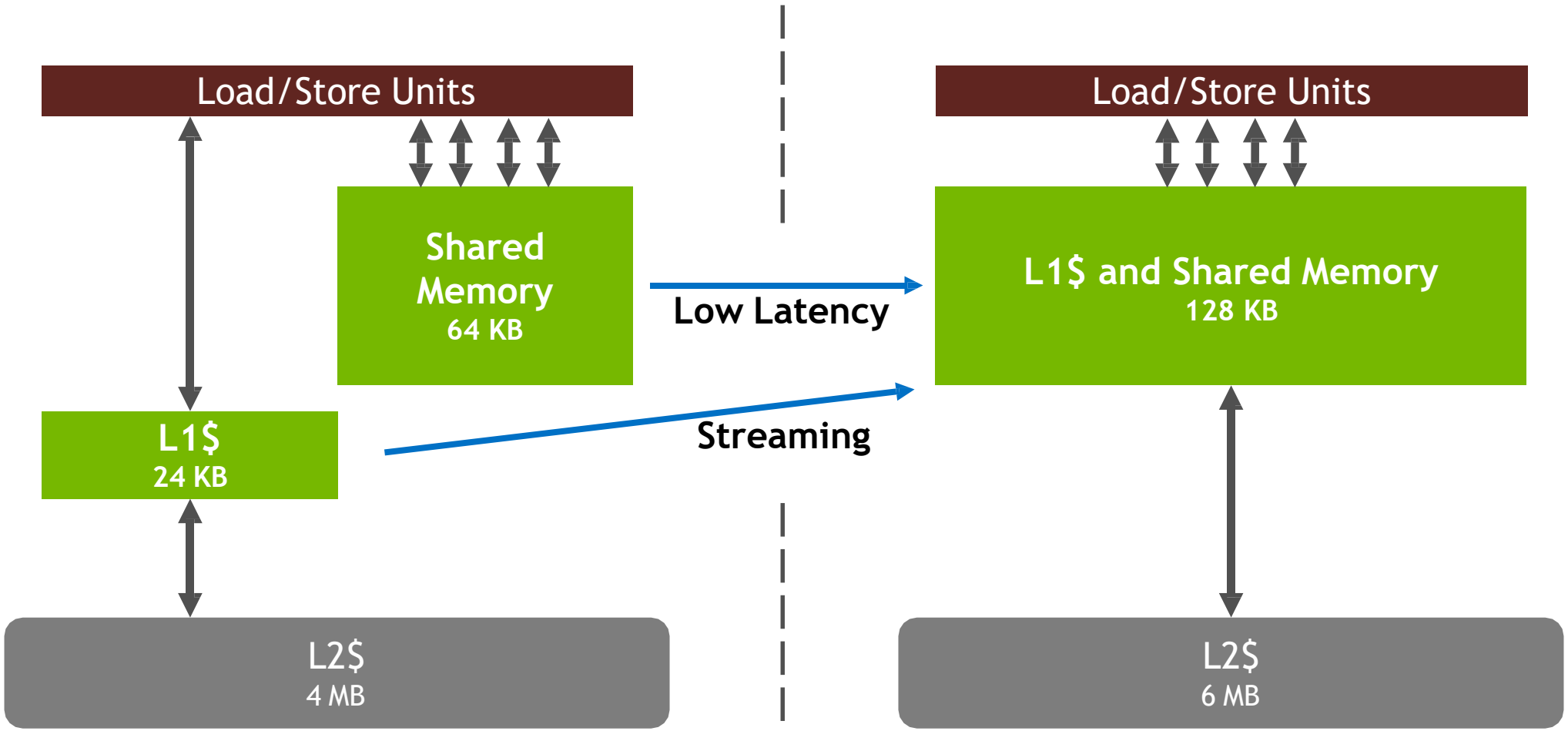
The driver usually does a pretty good job at choosing the right split.

To overcome 48 KB per threadblock limitation call:
cudaFuncSetAttribute (MyKernel, cudaFuncAttributeMaxDynamicSharedMemorySize, maxsize);

RECAP: PASCAL L1 AND SHARED MEMORY



UNIFYING KEY TECHNOLOGIES



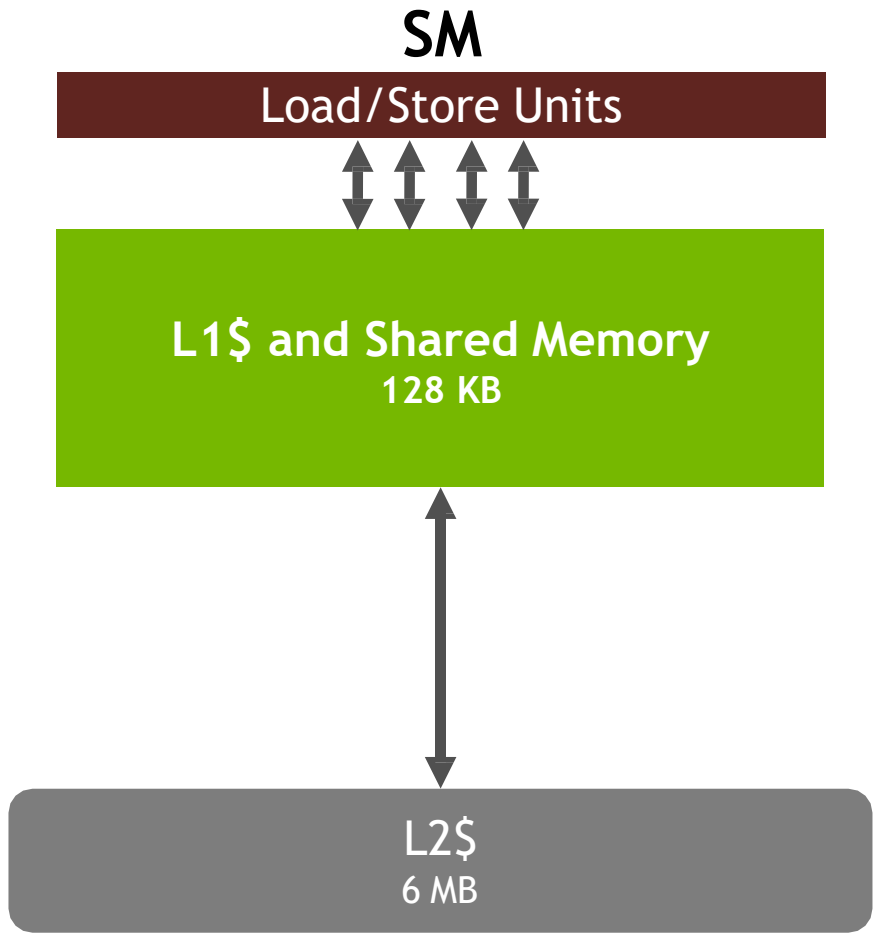
VOLTA L1 AND SHARED MEMORY

Volta Streaming L1$:

Low cache hit latency
4x more bandwidth
5x more capacity

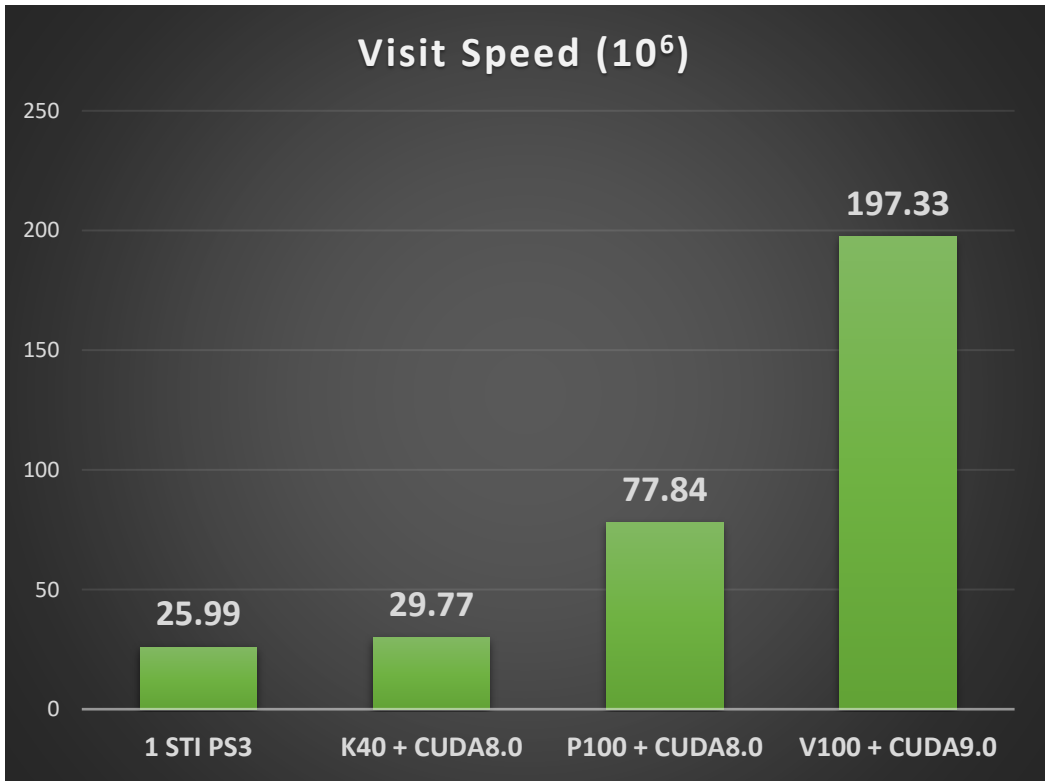
Volta Shared Memory :

Unified storage with L1
Configurable up to 96KB



“SCALABILITY OF CPU AND GPU SOLUTIONS OF THE PRIME ELLIPTIC CURVE DISCRETE LOGARITHM PROBLEM”

Jairo Panetta (ITA), Paulo Souza (ITA), Luiz Laranjeira (UnB), Carlos Teixeira Jr (UnB)



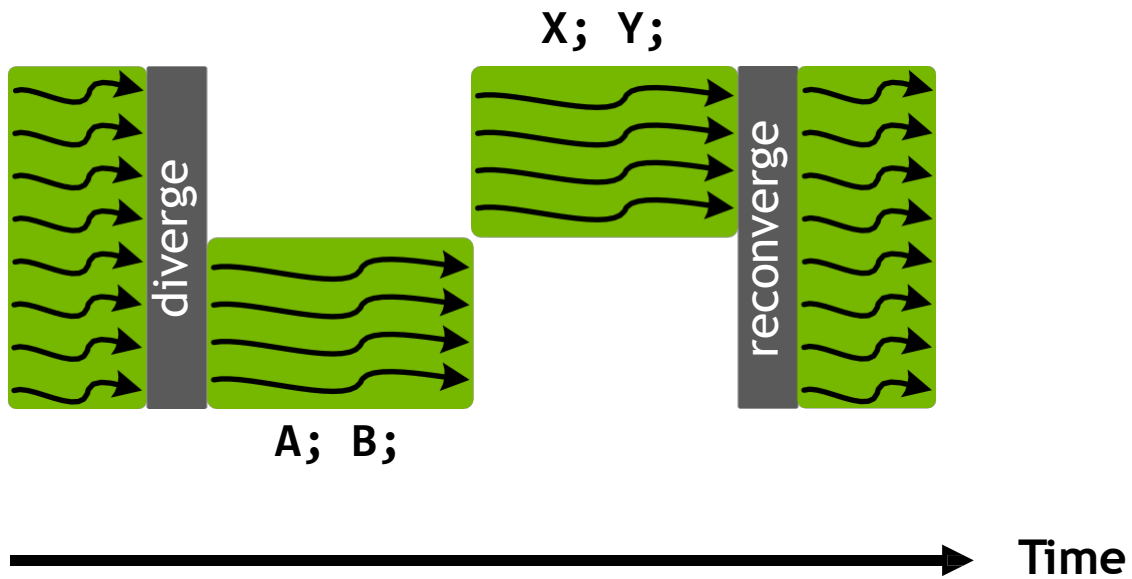
SBAC-PAD

RENASIC
Rede Nacional em Segurança da Informação e Criptografia

VOLTA WARP EXECUTION MODEL

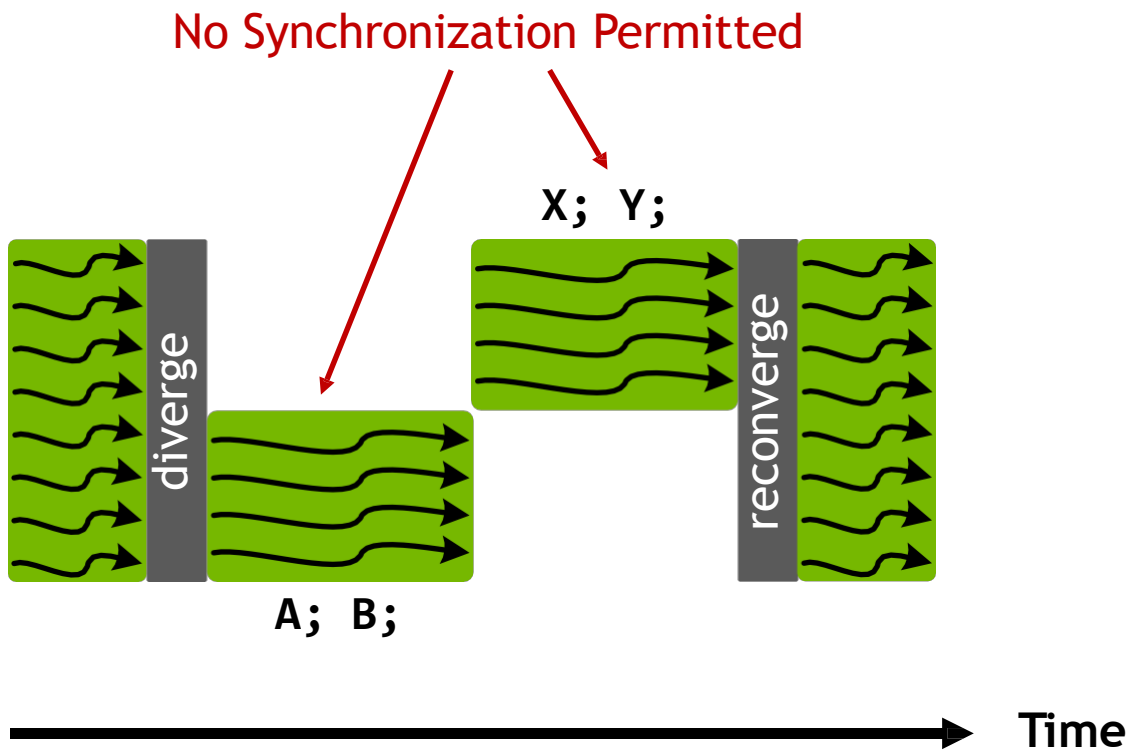
PASCAL WARP EXECUTION MODEL

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
```

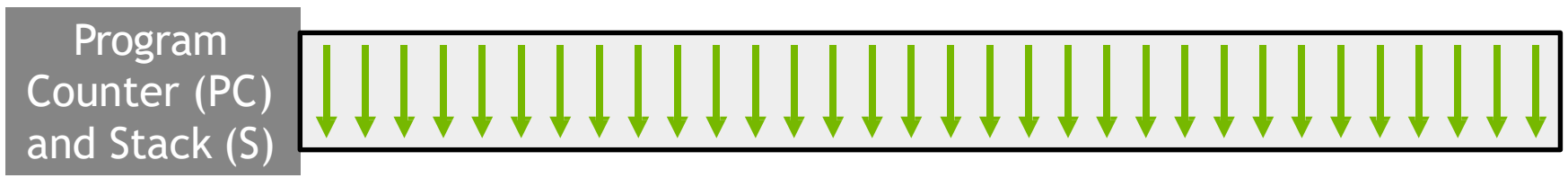


PASCAL WARP EXECUTION MODEL

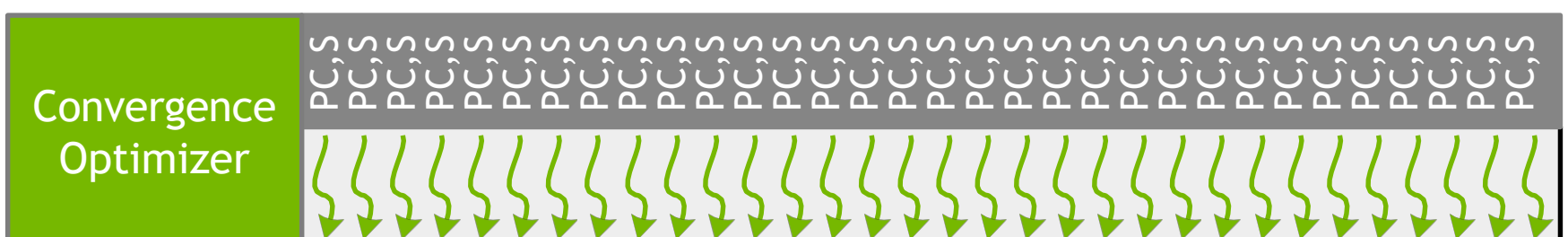
```
if (threadIdx.x < 4) {
    A;
    __syncwarp();
    B;
} else {
    X;
    __syncwarp();
    Y;
}
```



WARP IMPLEMENTATION



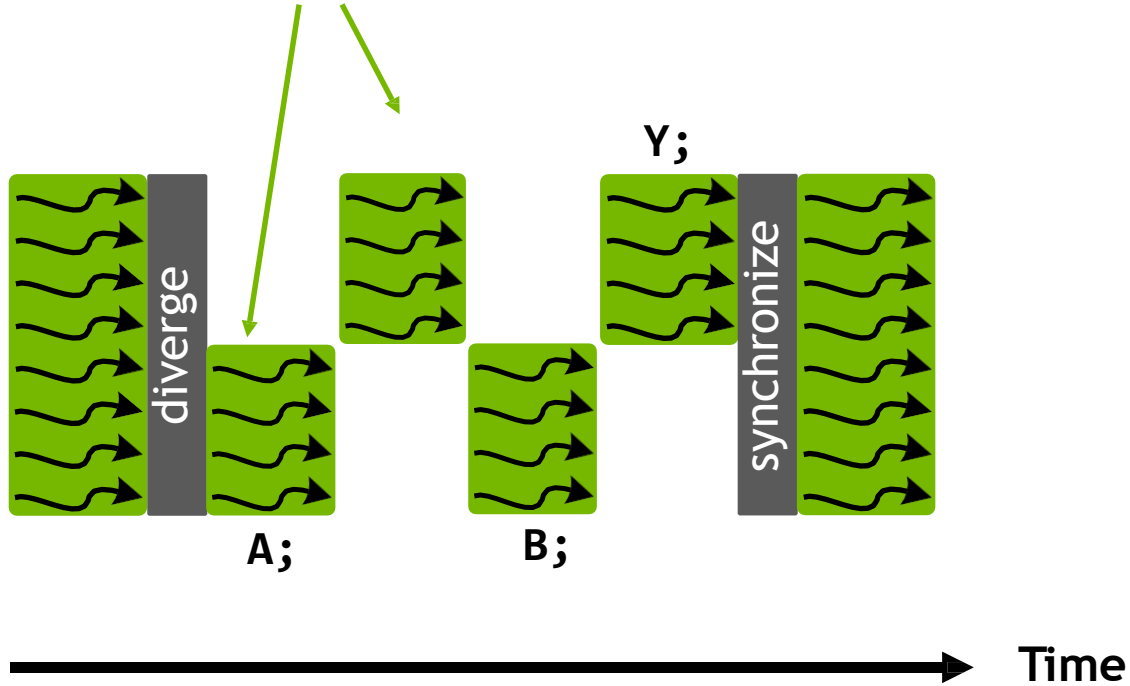
Volta



VOLTA WARP EXECUTION MODEL

Synchronization may lead to interleaved scheduling!

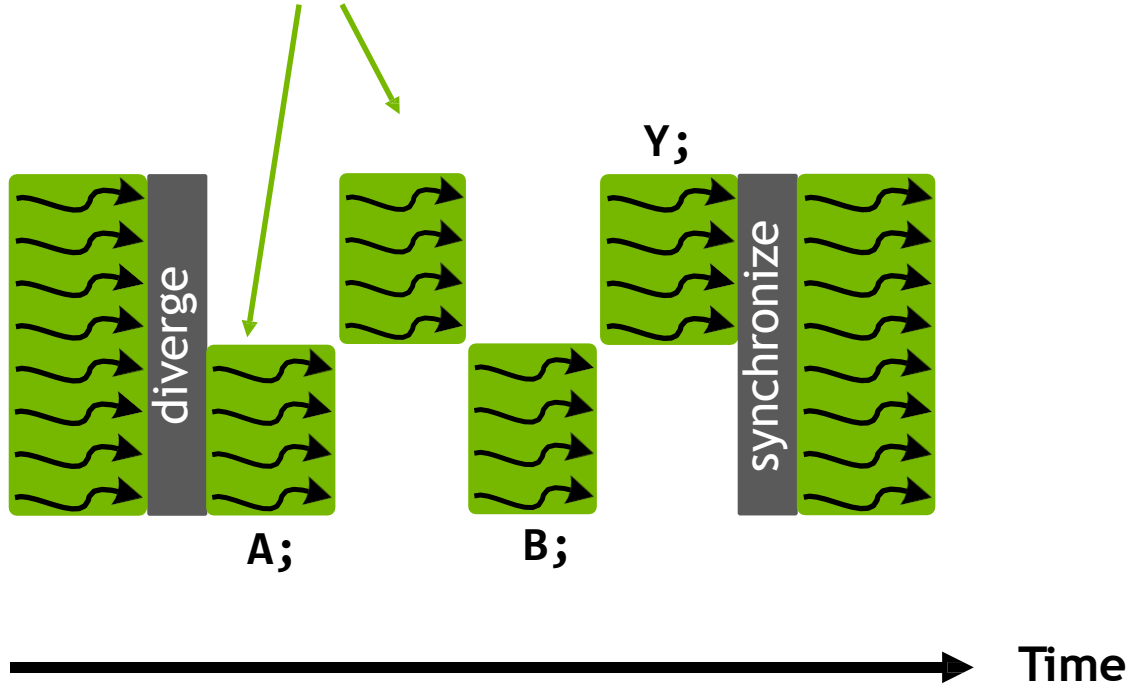
```
if (threadIdx.x < 4) {
    A;
    __syncwarp();
    B;
} else {
    X;
    __syncwarp();
    Y;
}
__syncwarp();
```



VOLTA WARP EXECUTION MODEL

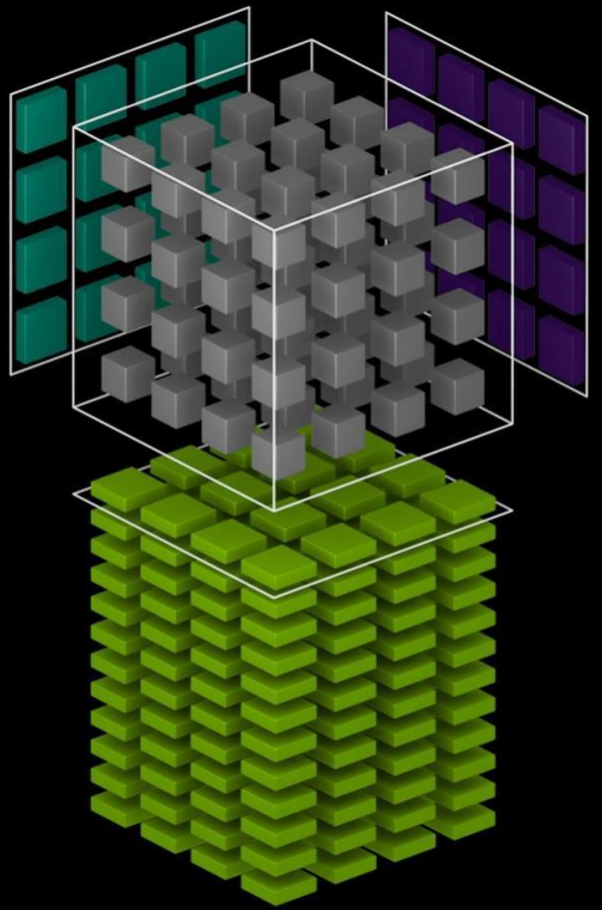
Synchronization may lead to interleaved scheduling!

```
if (threadIdx.x < 4) {
    A;
    __syncwarp();
    B;
} else {
    X;
    __syncwarp();
    Y;
}
__syncwarp();
```



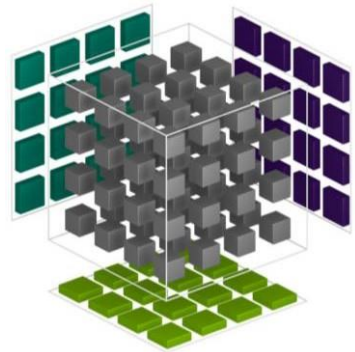
Software synchronization also supported, e.g. locks for doubly-linked list!

VOLTA TENSOR CORE



TENSOR CORE

Mixed Precision Matrix Math
4x4 matrices



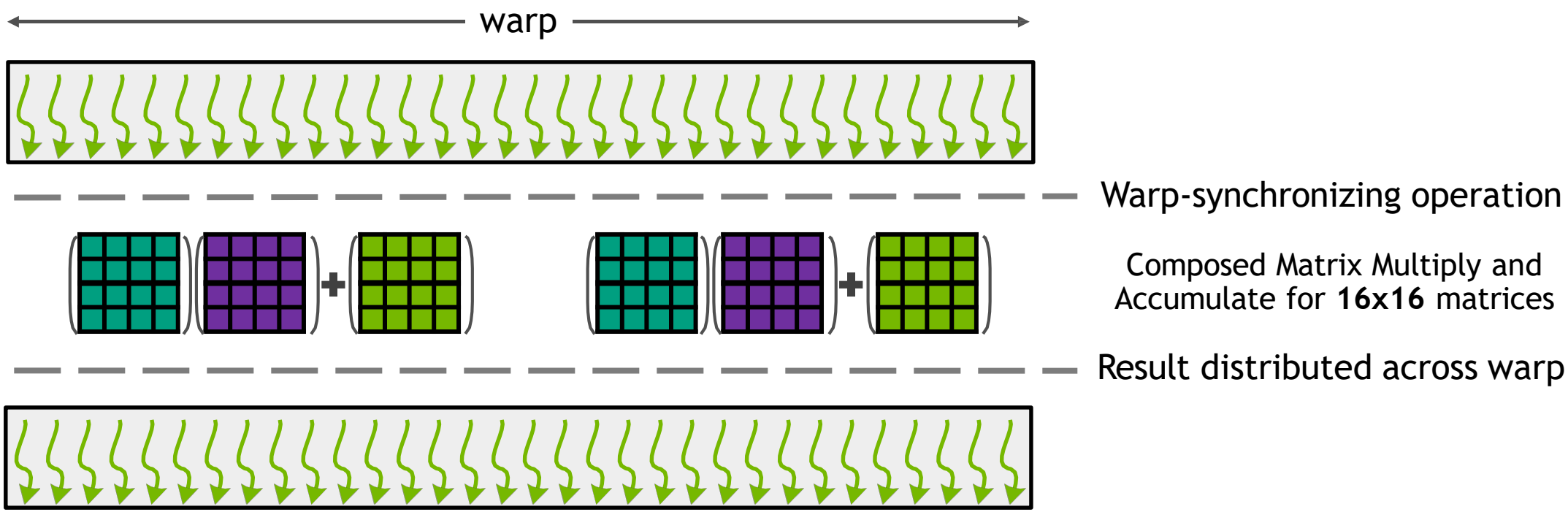
$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0}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

FP16 or FP32 — FP16 — FP16 — FP16 or FP32

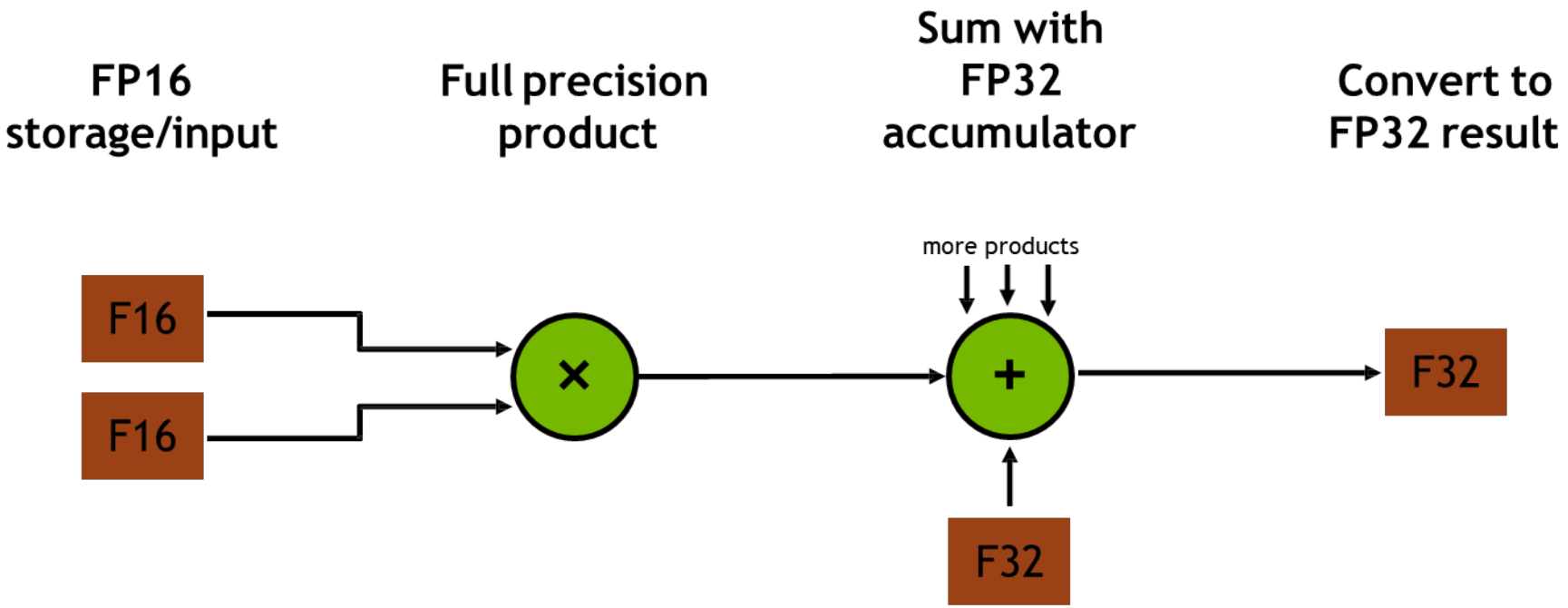
$$D = AB + C$$

TENSOR SYNCHRONIZATION

FULL WARP 16X16 MATRIX MATH

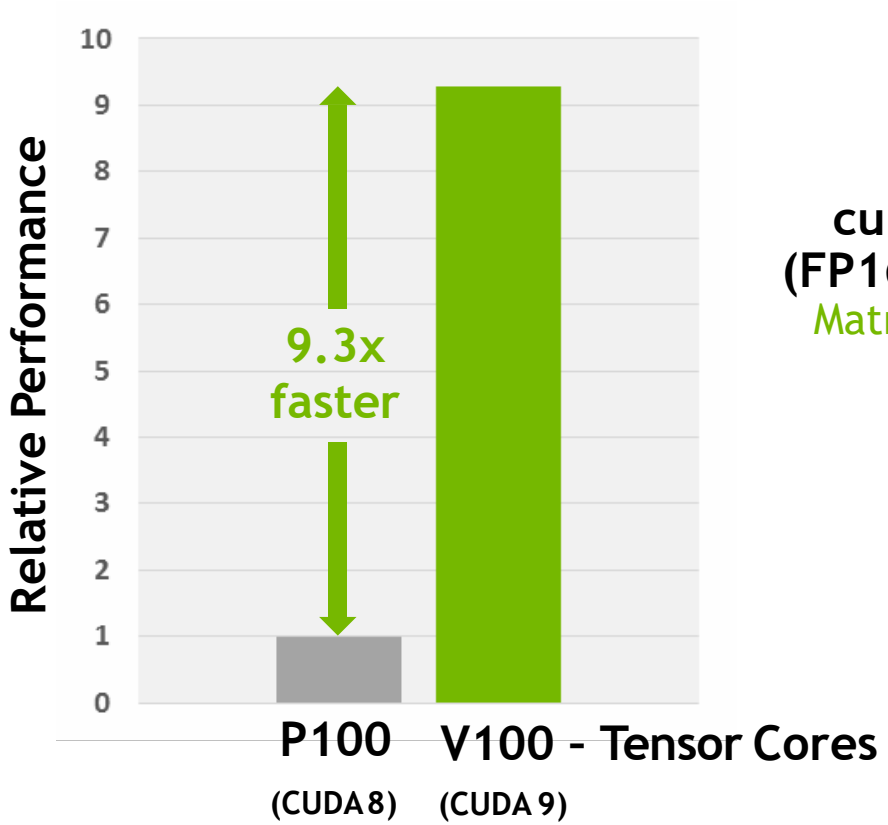


VOLTA TENSOR OPERATION



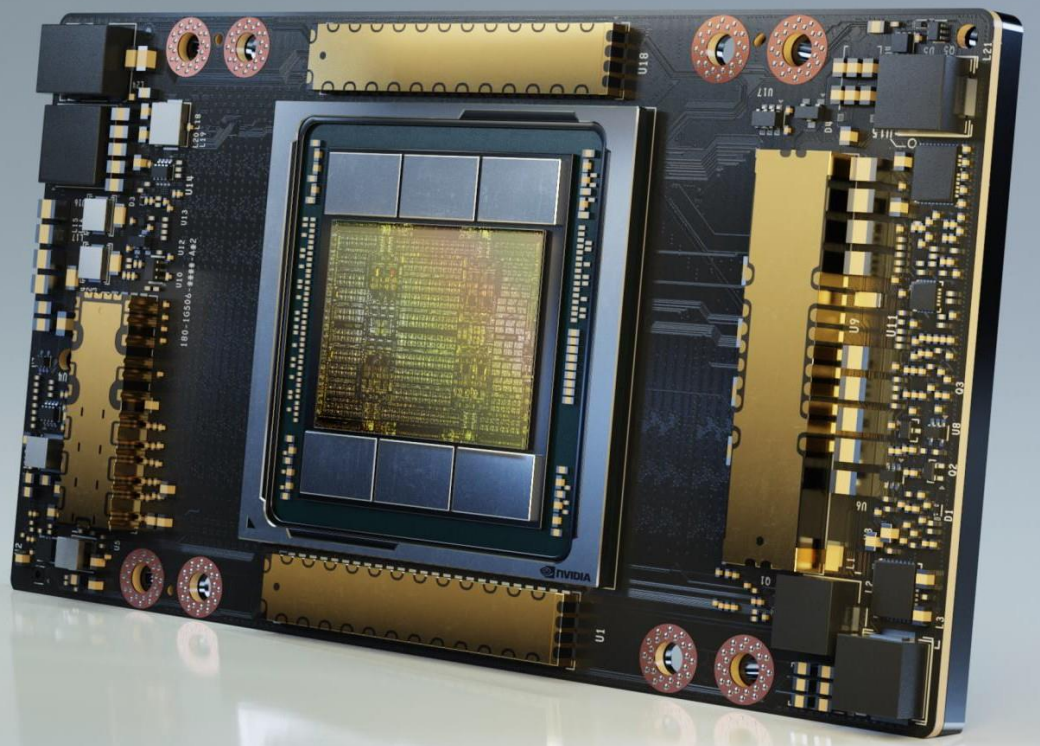
Also supports FP16 accumulator mode for inferencing

A GIANT LEAP FOR DEEP LEARNING

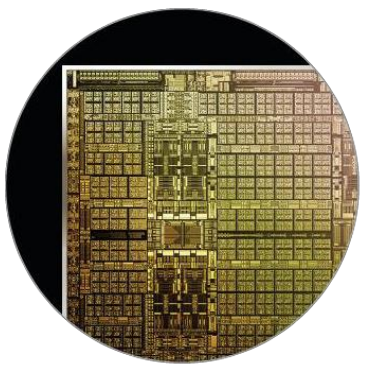


cuBLAS Mixed Precision
(FP16 input, FP32 compute)
Matrix Multiply (M=N=K=2048)

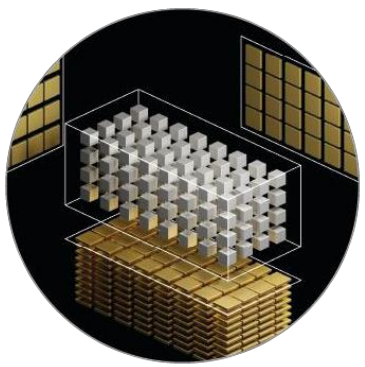
Inside the NVIDIA Ampere Architecture



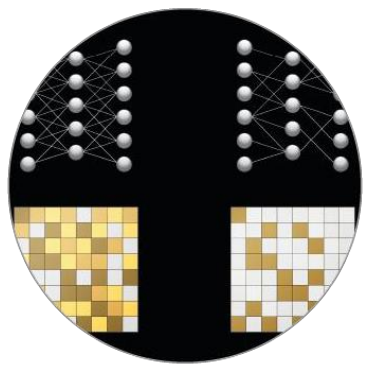
UNPRECEDENTED ACCELERATION AT EVERY SCALE



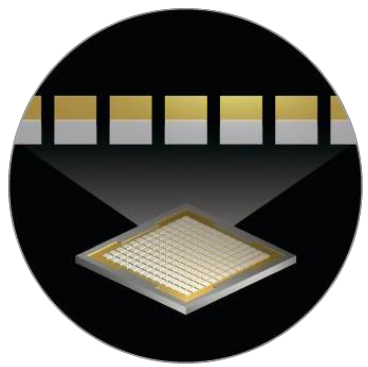
54 BILLION XTORS



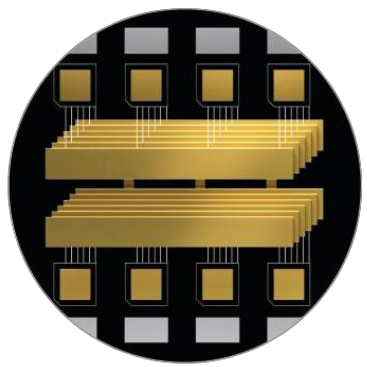
3rd GEN TENSOR CORES



SPARSITY ACCELERATION



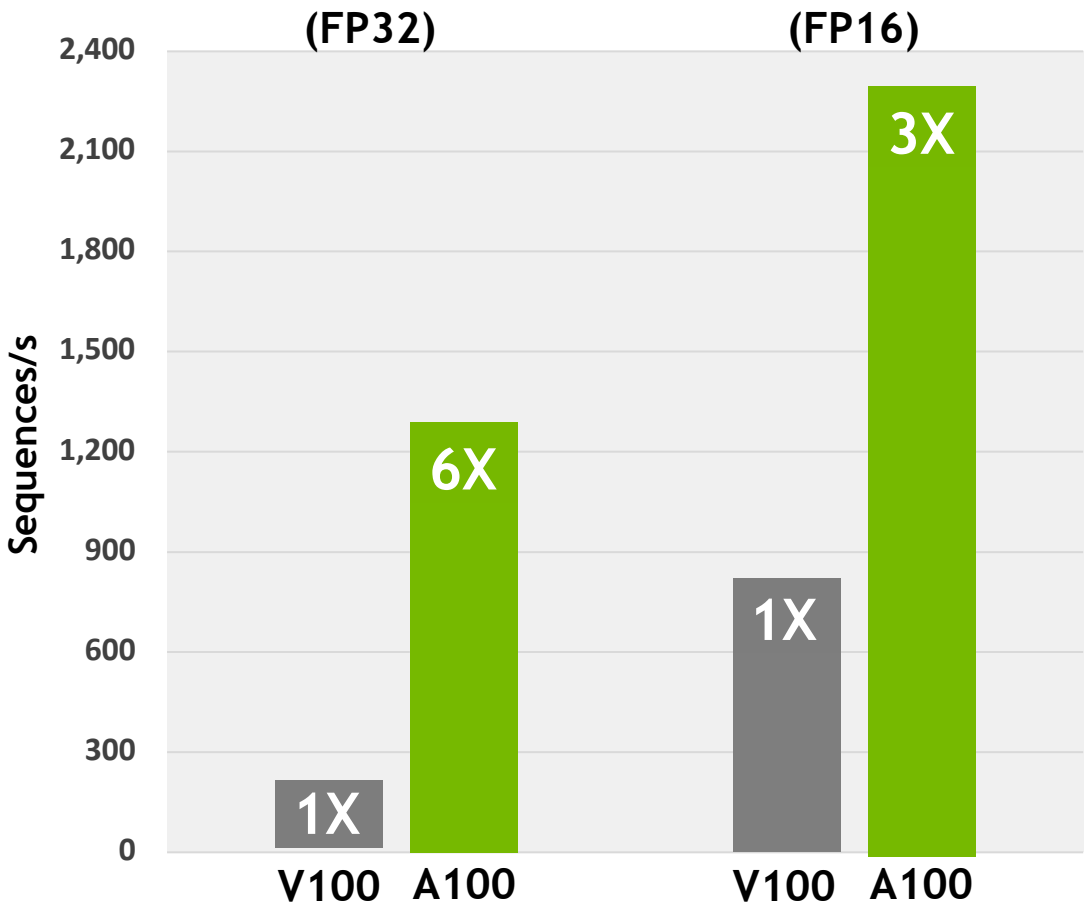
MIG



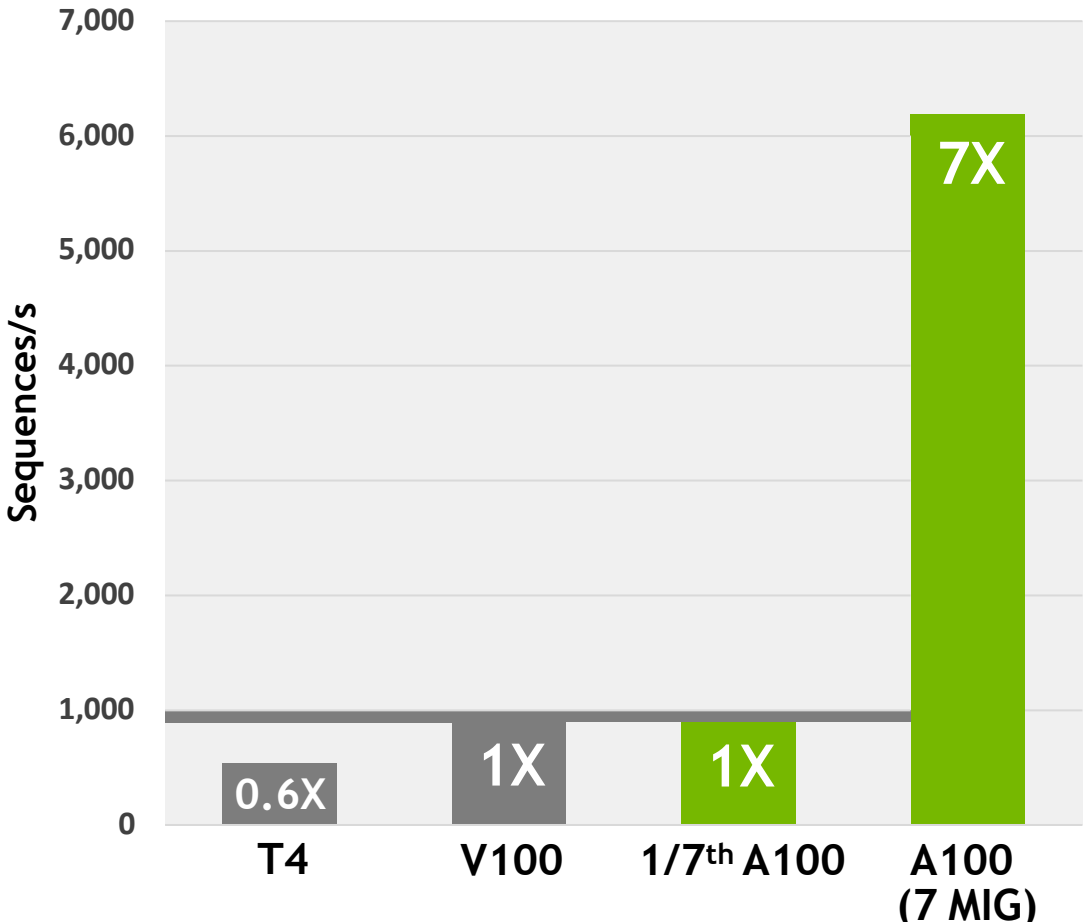
3rd GEN NVLINK & NVSWITCH

UNIFIED AI ACCELERATION

BERT-LARGE TRAINING

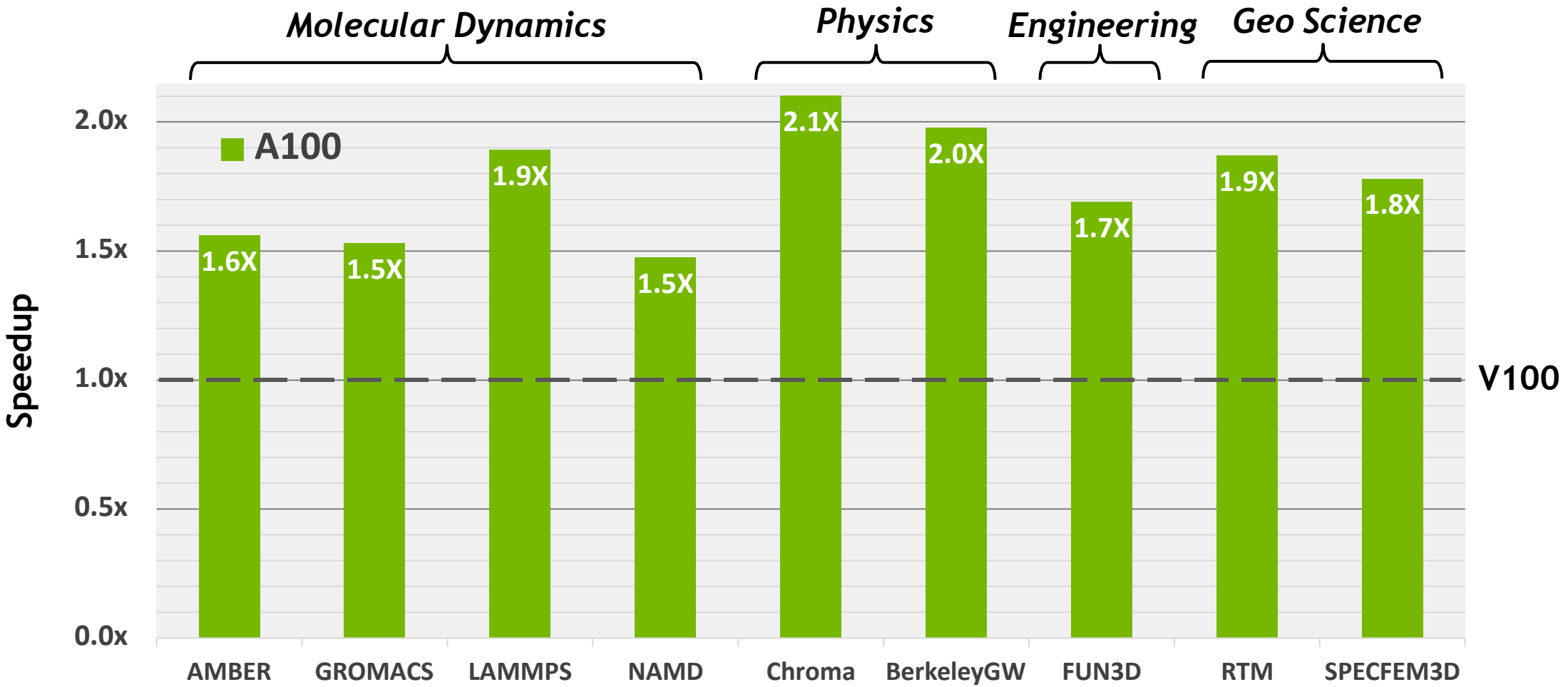


BERT-LARGE INFERENCE



All results are measured
BERT Large Training (FP32 & FP16) measures Pre-Training phase, uses PyTorch including (2/3) Phase1 with Seq Len 128 and (1/3) Phase 2 with Seq Len 512, V100 is DGX1 Server with 8xV100, A100 is DGX A100 Server with 8xA100, A100 uses TF32 Tensor Core for FP32 training
BERT Large Inference uses TRT 7.1 for T4/V100, with INT8/FP16 at batch size 256. Pre-production TRT for A100, uses batch size 94 and INT8 with sparsity

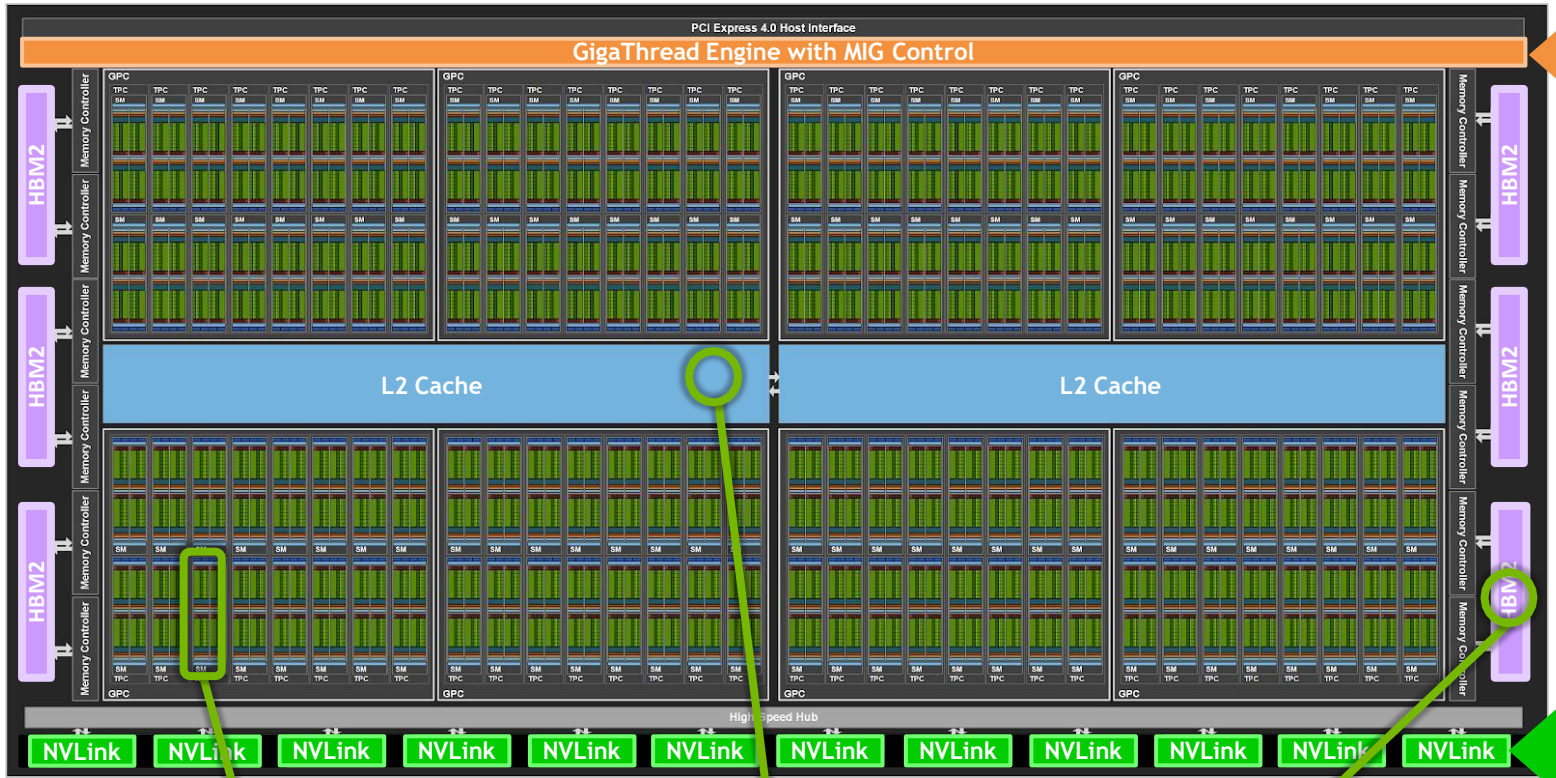
ACCELERATING HPC



All results are measured
Except BerkeleyGW, V100 used is single V100 SXM2. A100 used is single A100 SXM4
More apps detail: AMBER based on PME-Cellulose, GROMACS with STMV (h-bond), LAMMPS with Atomic Fluid LJ-2.5, NAMD with v3.0a1 STMV_NVE
Chroma with szscl21_24_128, FUN3D with dpw, RTM with Isotropic Radius 4 1024^3, SPECFEM3D with Cartesian four material model
BerkeleyGW based on Chi Sum and uses 8xV100 in DGX-1, vs 8xA100 in DGX A100

A100 TENSOR-CORE GPU

54 billion transistors in 7nm



7x

Scale OUT

Multi-Instance GPU

2x BW

Scale UP

3rd gen. NVLINK

108 SMs
6912 CUDA Cores

40MB L2
6.7x capacity

1.56 TB/s HBM2
1.7x bandwidth

A100 SM

Third-generation Tensor Core

Faster and more efficient
Comprehensive data types
Sparsity acceleration

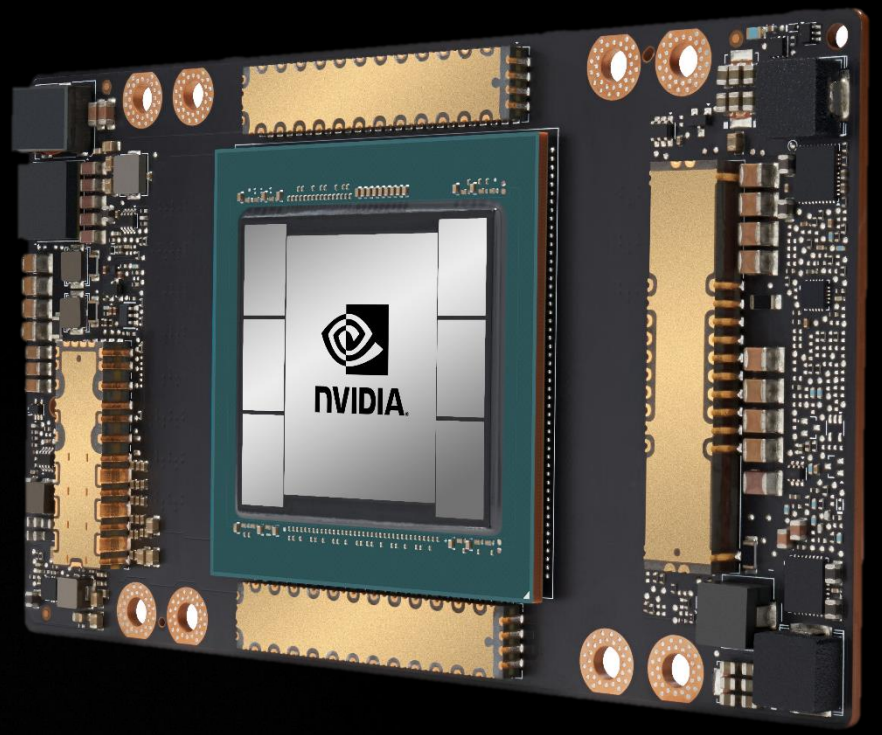
Asynchronous data movement and synchronization

Increased L1/SMEM capacity

INTRODUCING NVIDIA A100

Greatest Generational Leap – 20X Volta

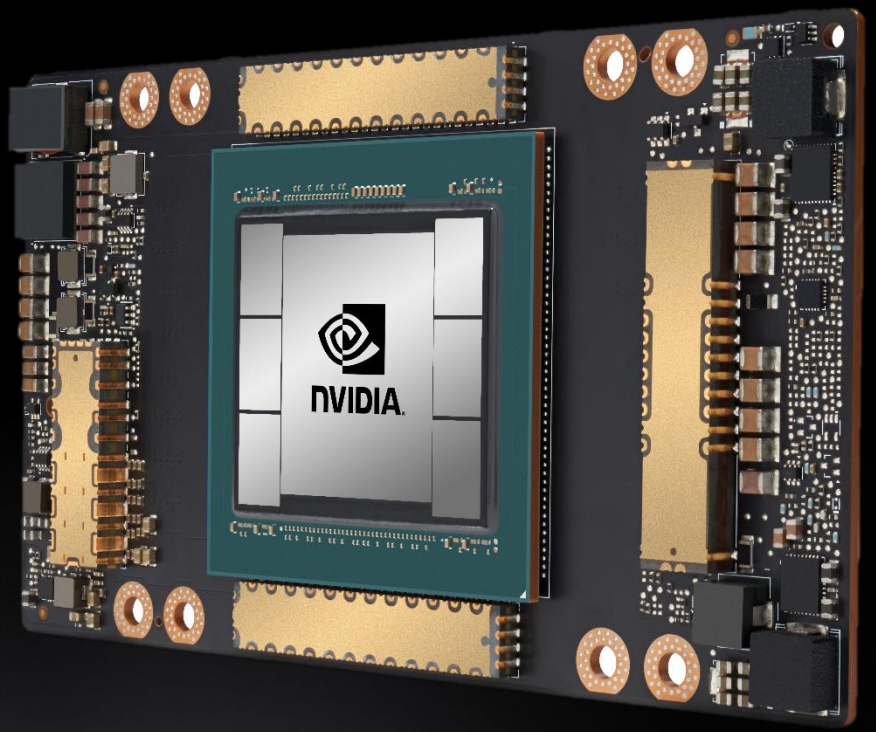
	Peak		Vs Volta
FP32 TRAINING	312	TFLOPS	20X
INT8 INFERENCE	1,248	TOPS	20X
FP64 HPC	19.5	TFLOPS	2.5X
MULTI INSTANCE GPU			7X GPUs



54B XTOR | 826mm2 | TSMC 7N | 40GB Samsung HBM2 | 600 GB/s NVLink

NVIDIA A100 DETAILED SPECS

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

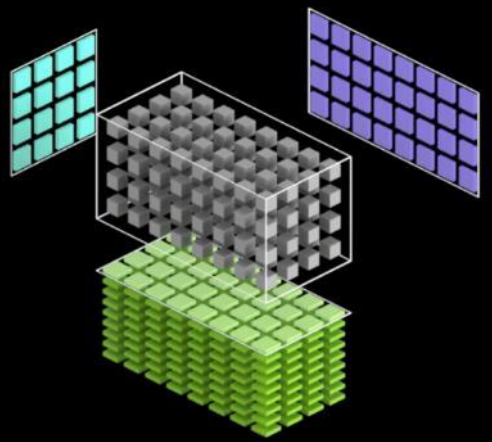


* Includes Sparsity

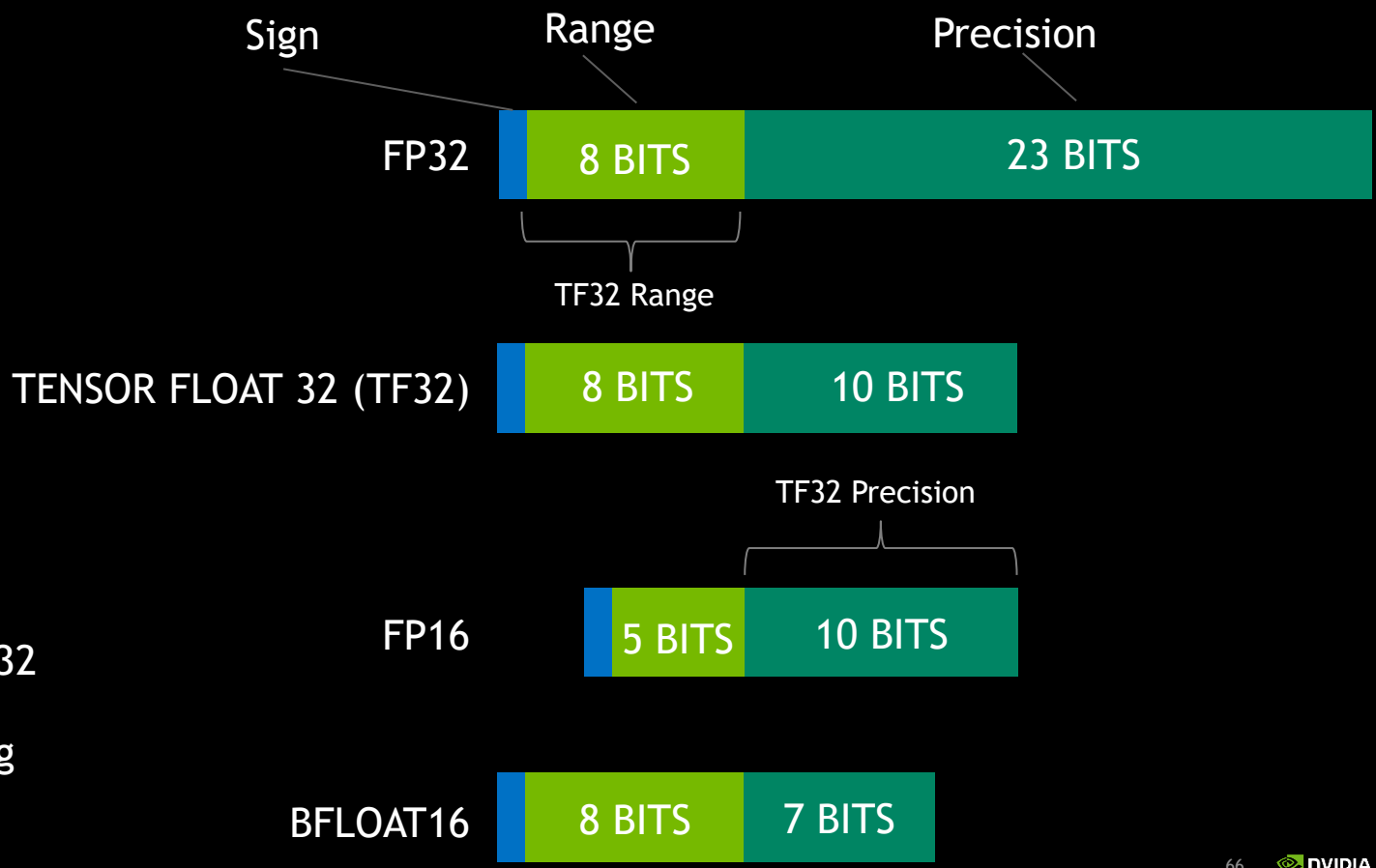

1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity

1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity

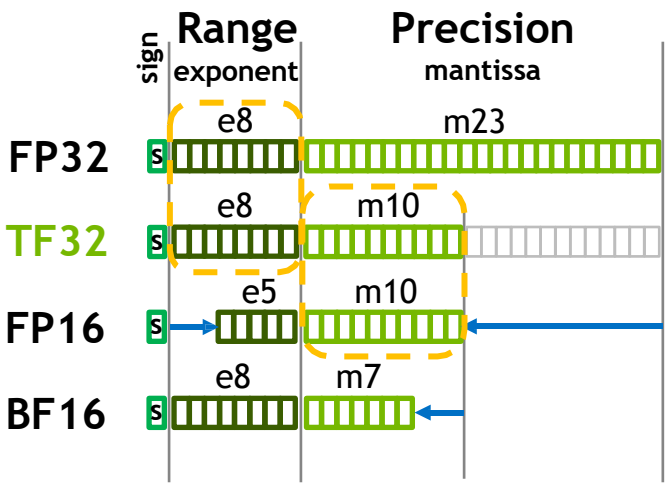
NEW TF32 TENSOR CORES



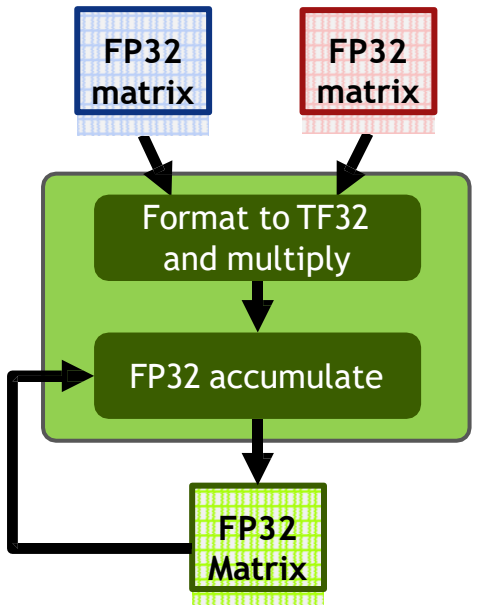
- Range of FP32 and Precision of FP16
- Input in FP32 and Accumulation in FP32
- No Code Change Speed-up for Training



INSIDE A100 TensorFloat-32 (TF32)



Range of FP32 with precision of FP16



FP32 input/output

FP32 storage and math for all activations, gradients, …
everything outside tensor cores

Out-of-the-box tensor core acceleration for DL

Easy step towards maximizing tensor core performance with mixed-precision (FP16, BF16)

Up to 4x speedup on linear solvers for HPC

S22082: Mixed-Precision Training of Neural Networks, 5/20 2:45pm PDT

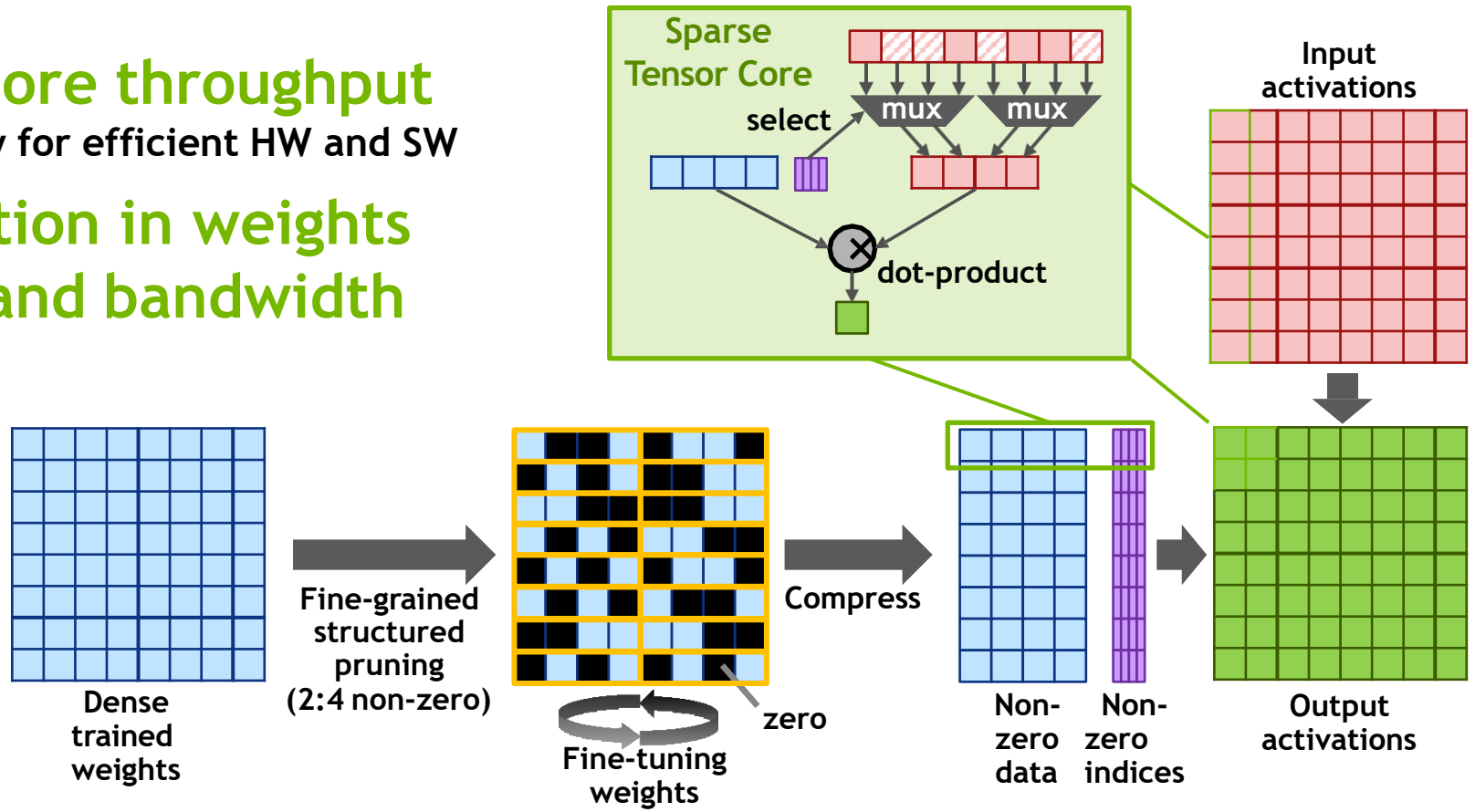
S21681: How CUDA Math Libraries can help you unleash the power of the new NVIDIA A100 GPU (recording available)

INSIDE A100 SPARSE TENSOR CORE

2x Tensor Core throughput

Structured-sparsity for efficient HW and SW

~2x reduction in weights footprint and bandwidth



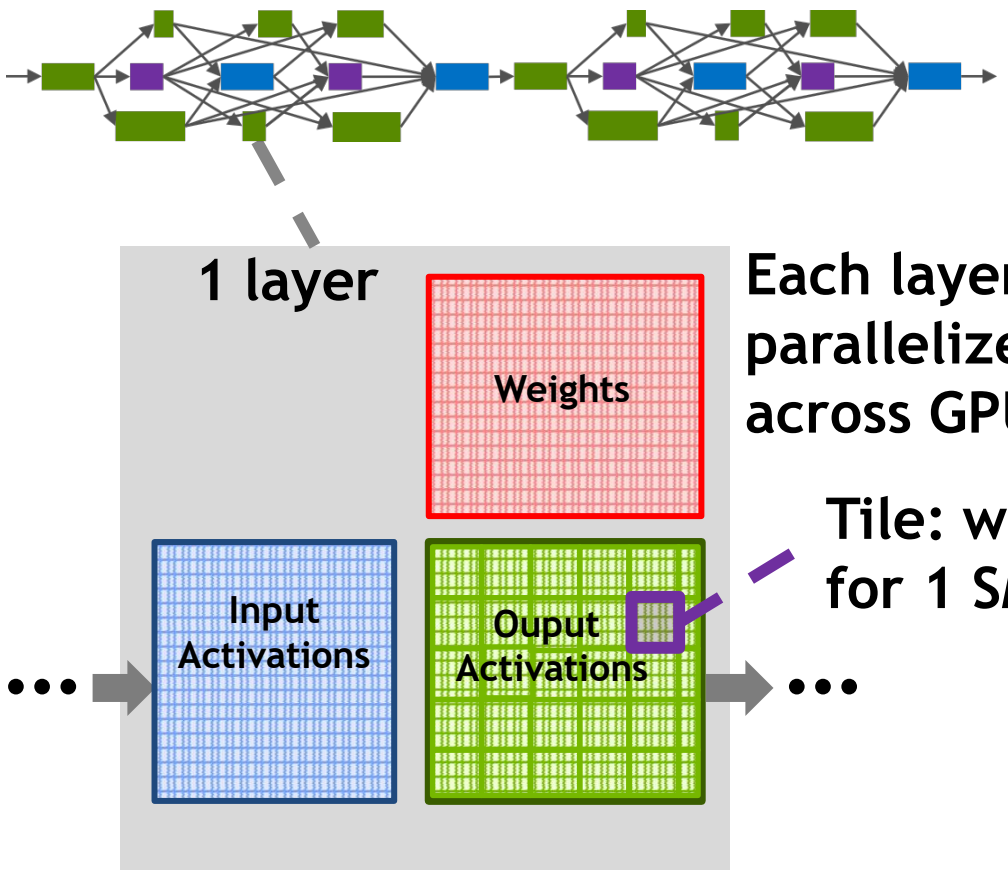
~No loss in inferencing accuracy

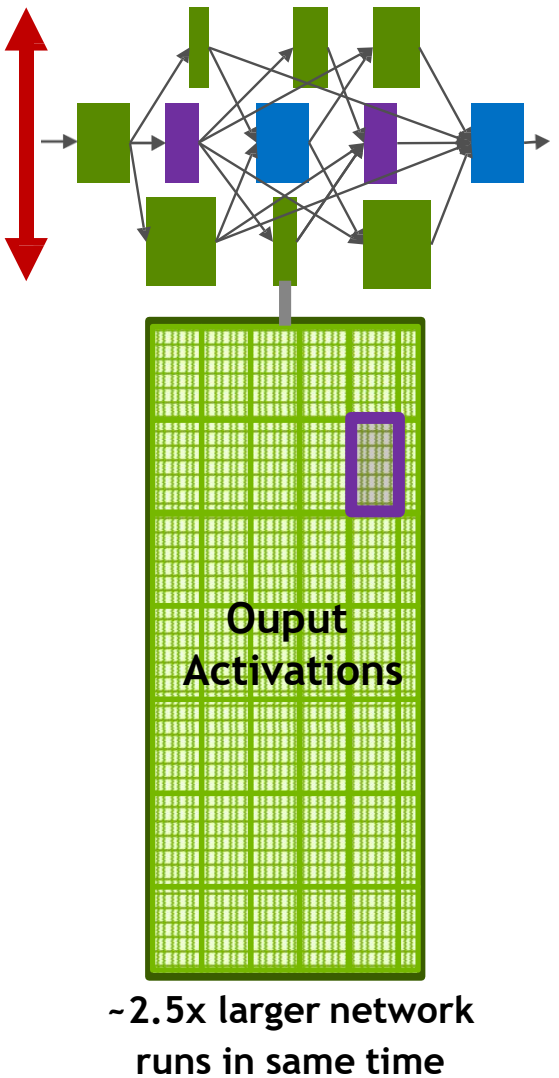
Evaluated across dozens of networks: vision, object detection, segmentation, natural language modeling, translation

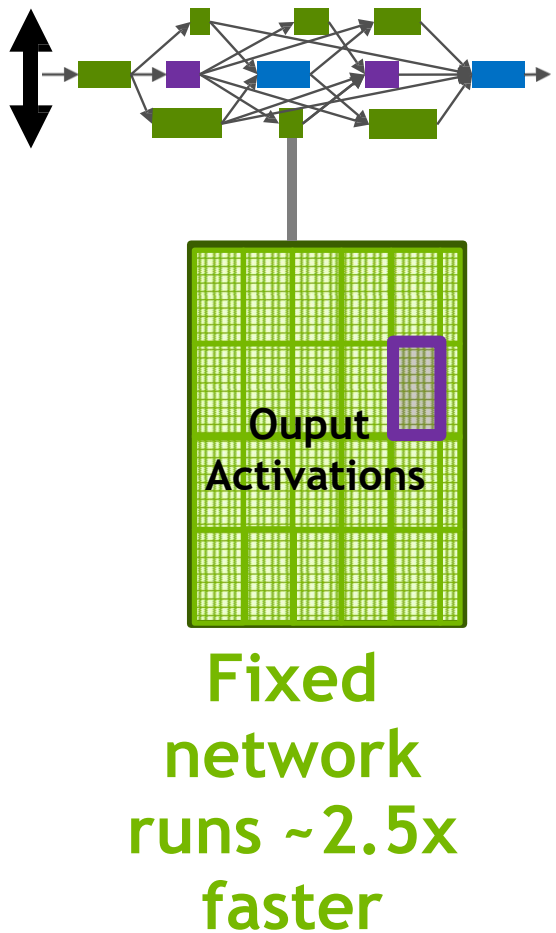
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. Productivity

DL STRONG SCALING

DL networks:
Long chains of sequentially-dependent compute-intensive layers



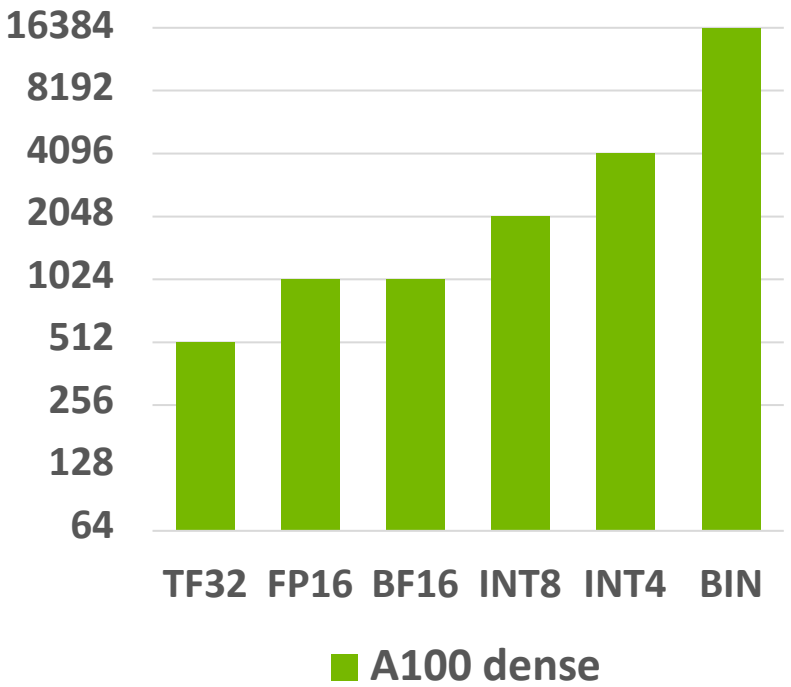




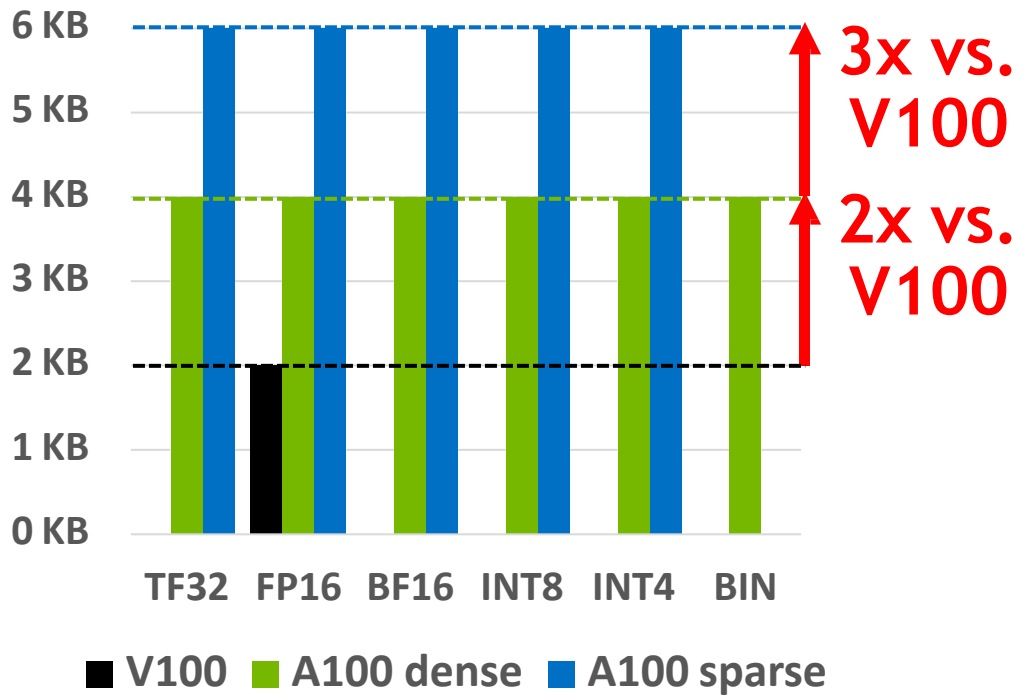
NVIDIA

HOW TO KEEP TENSOR CORES FED?

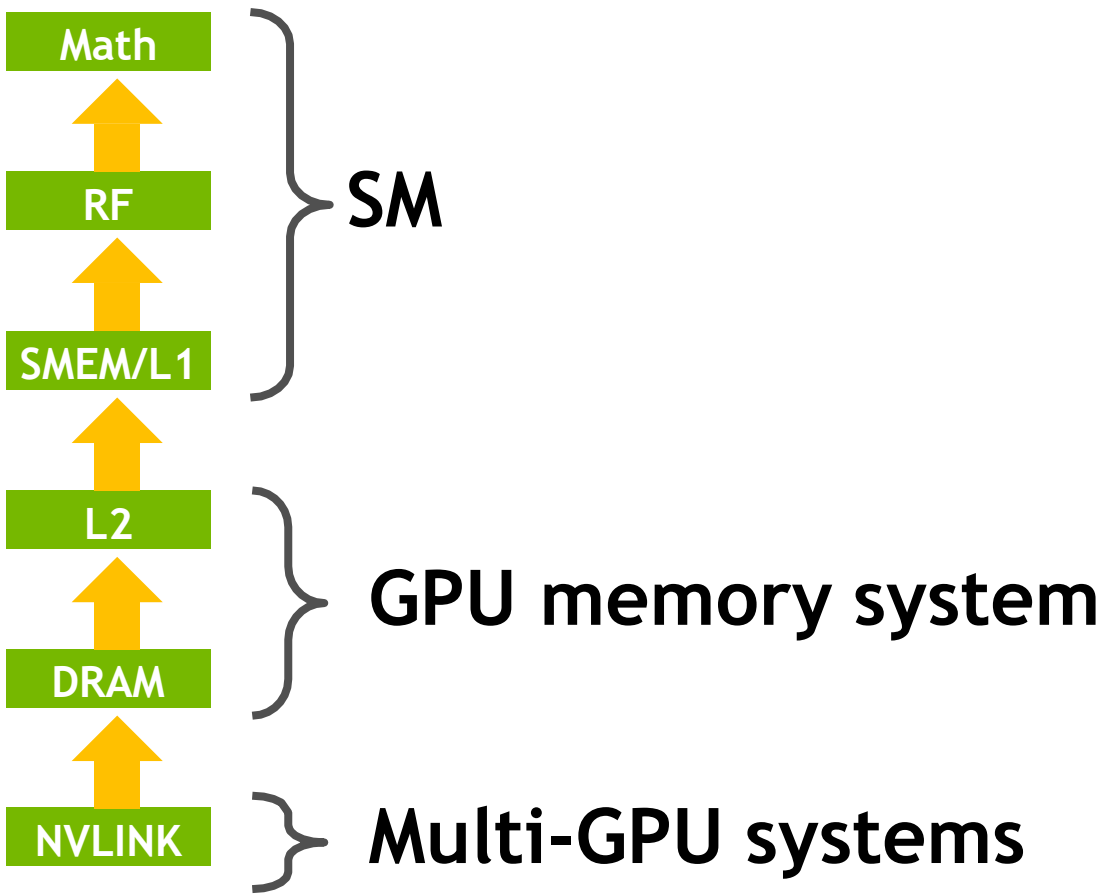
Math bandwidth
(MACs/clock/SM)



Required data bandwidth
(A+B operands, B/clock/SM)



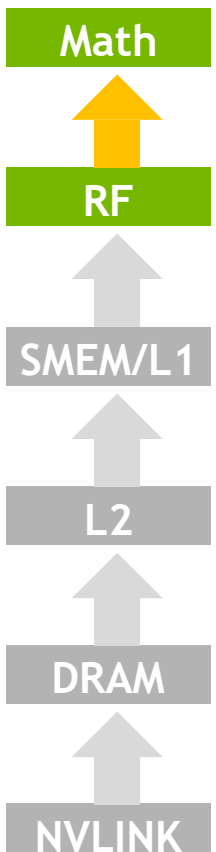
A100 STRONG SCALING INNOVATIONS

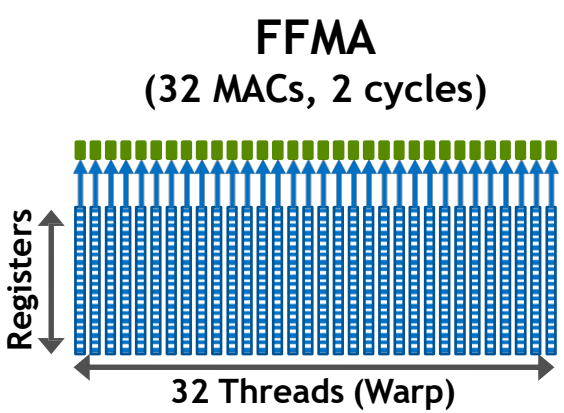


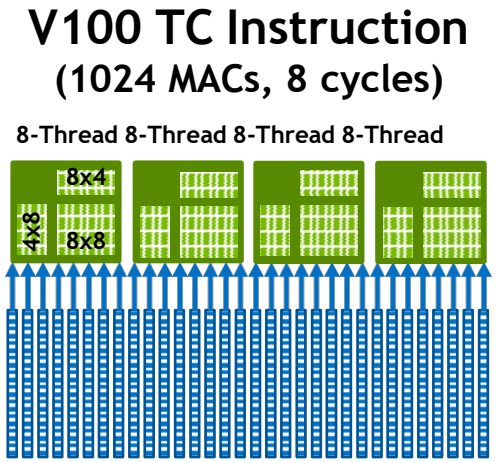
Improve speeds & feeds and efficiency across all levels of compute and memory hierarchy

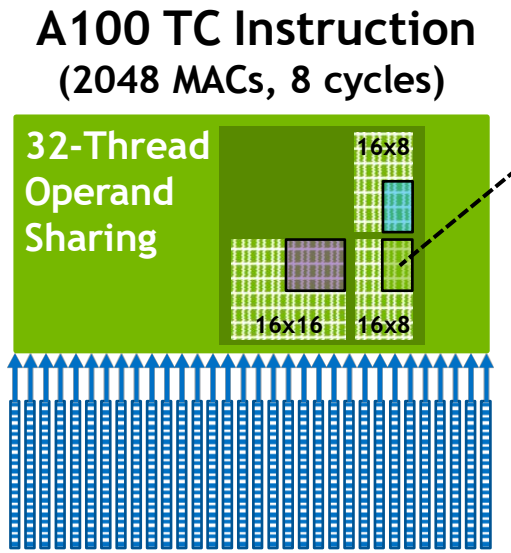
A100 TENSOR CORE

2x throughput vs. V100, >2x efficiency







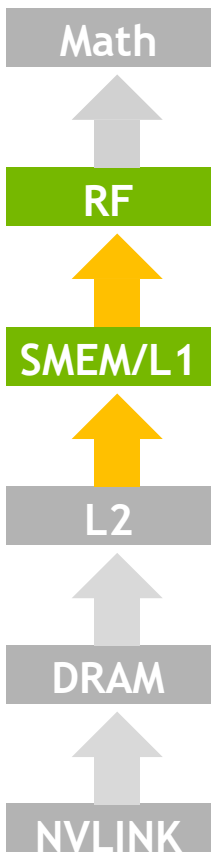


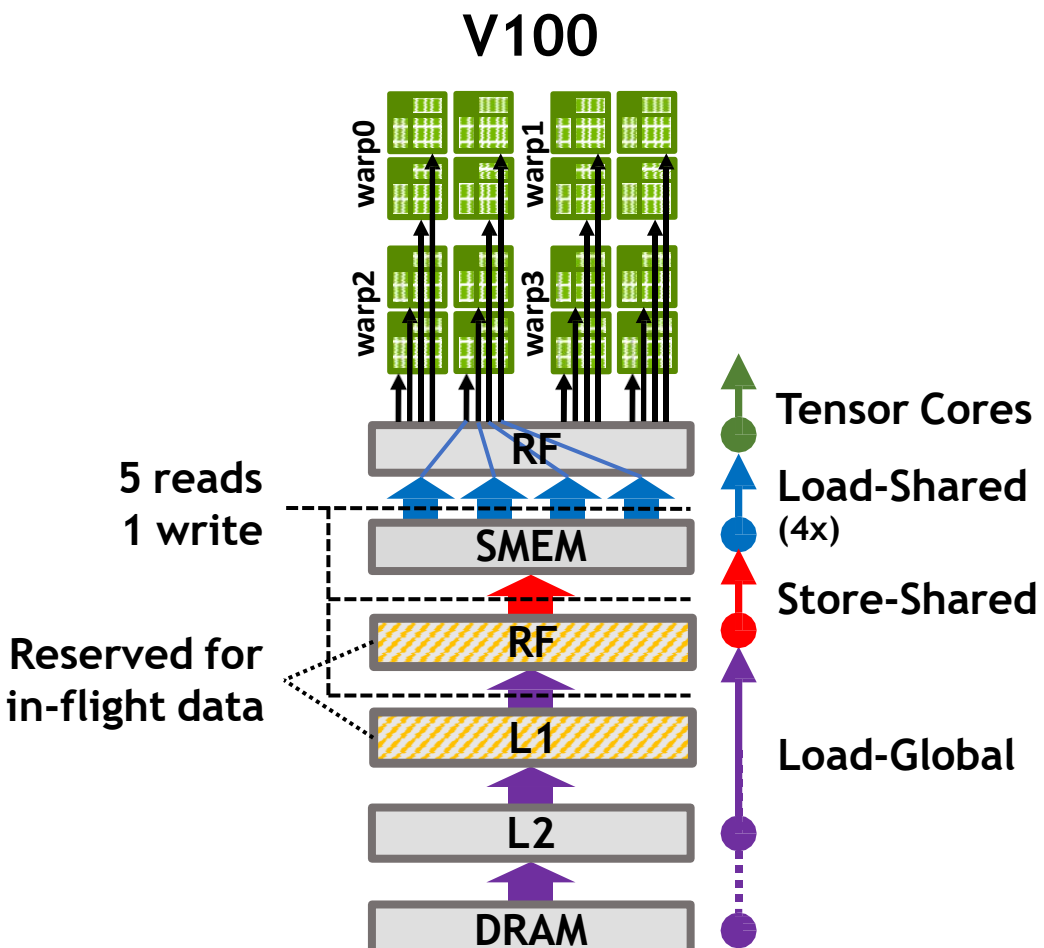
16x16x16 matrix multiply	FFMA	V100 TC	A100 TC	A100 vs. V100 (improvement)	A100 vs. FFMA (improvement)
Thread sharing	1	8	32	4x	32x
Hardware instructions	128	16	2	8x	64x
Register reads+writes (warp)	512	80	28	2.9x	18x
Cycles	256	32	16	2x	16x

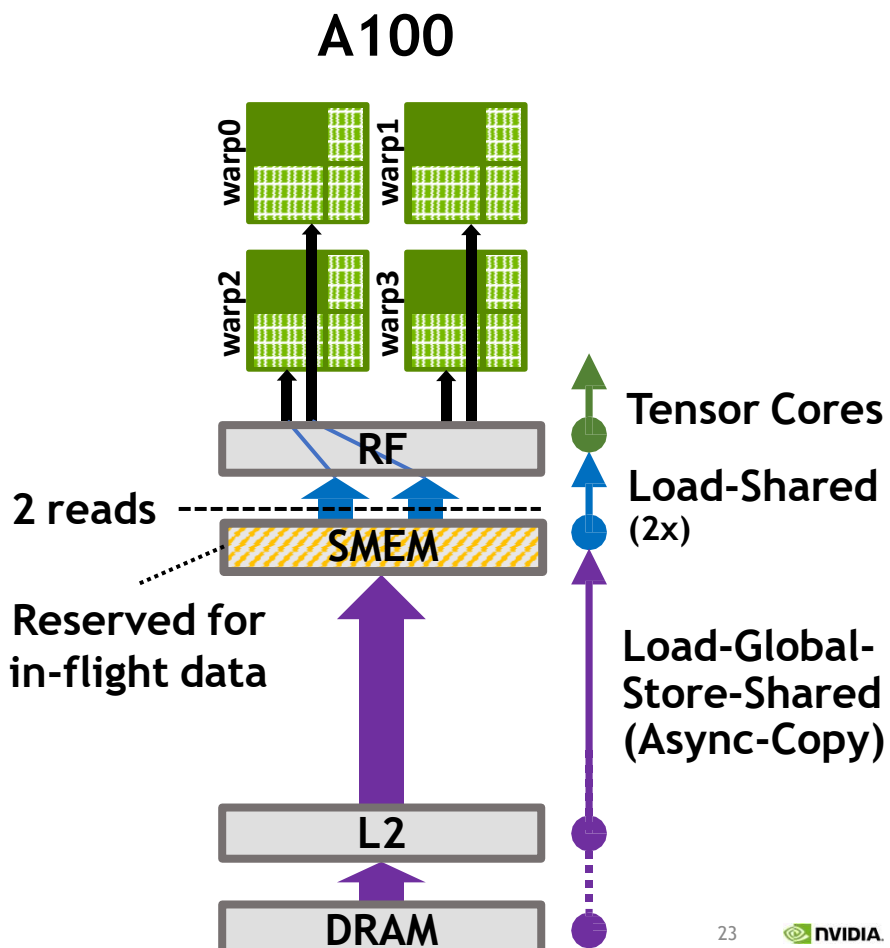
Tensor Cores assume FP16 inputs with FP32 accumulator, V100 Tensor Core instruction uses 4 hardware instructions

A100 SM DATA MOVEMENT EFFICIENCY

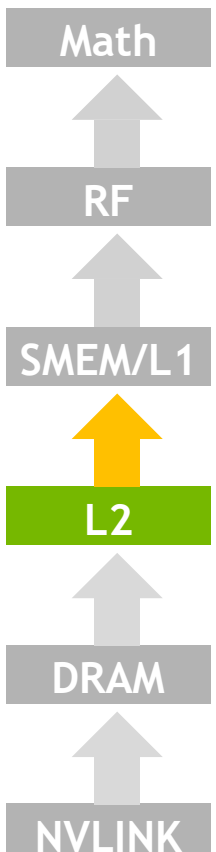
3x SMEM/L1 bandwidth, 2x in-flight capacity

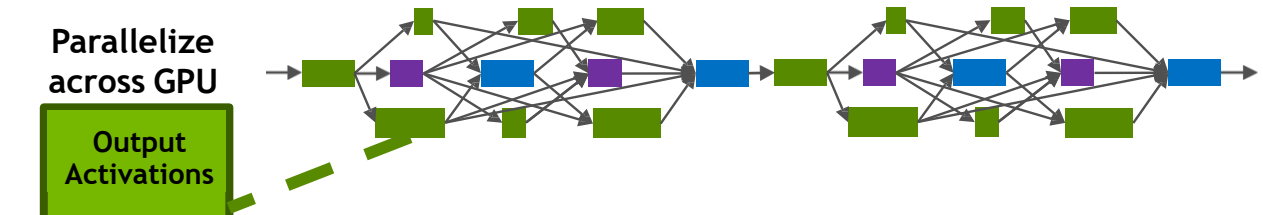


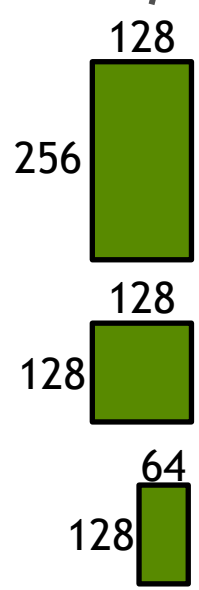




A100 L2 BANDWIDTH

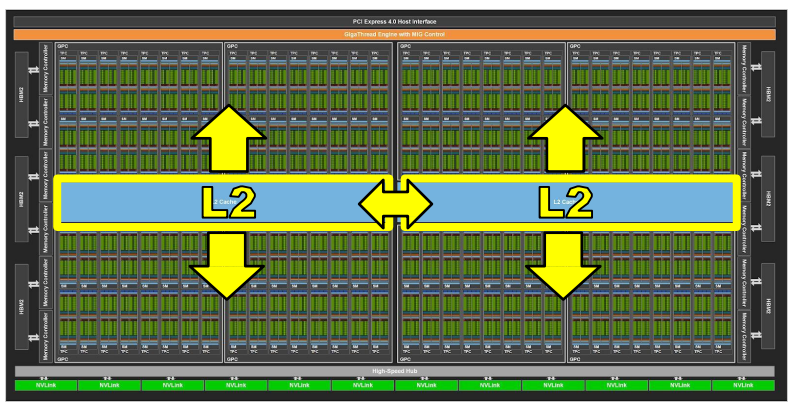






	V100	V100++ (hypothetical)	A100
	80 SMs V100 TC 64 L2 slices 32 B/clk/slice	108 SMs A100 TC 64 L2 slices 32 B/clk/slice	108 SMs A100 TC 80 L2 slices 64 B/clk/slice
	12 B/clk/SM 47%	24 B/clk/SM 127%	24 B/clk/SM 51%
	16 B/clk/SM 63%	32 B/clk/SM 169%	32 B/clk/SM 68%
	24 B/clk/SM 94%	48 B/clk/SM 253%	48 B/clk/SM 101%

Split L2 with hierarchical crossbar - 2.3x increase in bandwidth over V100, lower latency



A100 DRAM BANDWIDTH

Faster HBM2

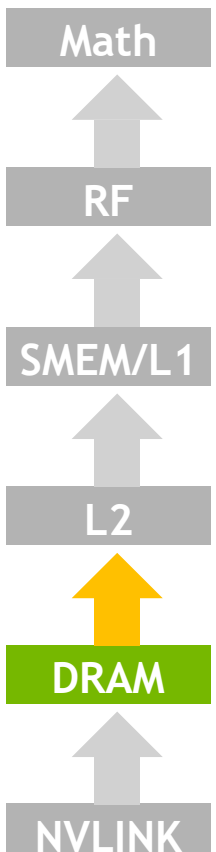
25% more pins, 38% faster clocks

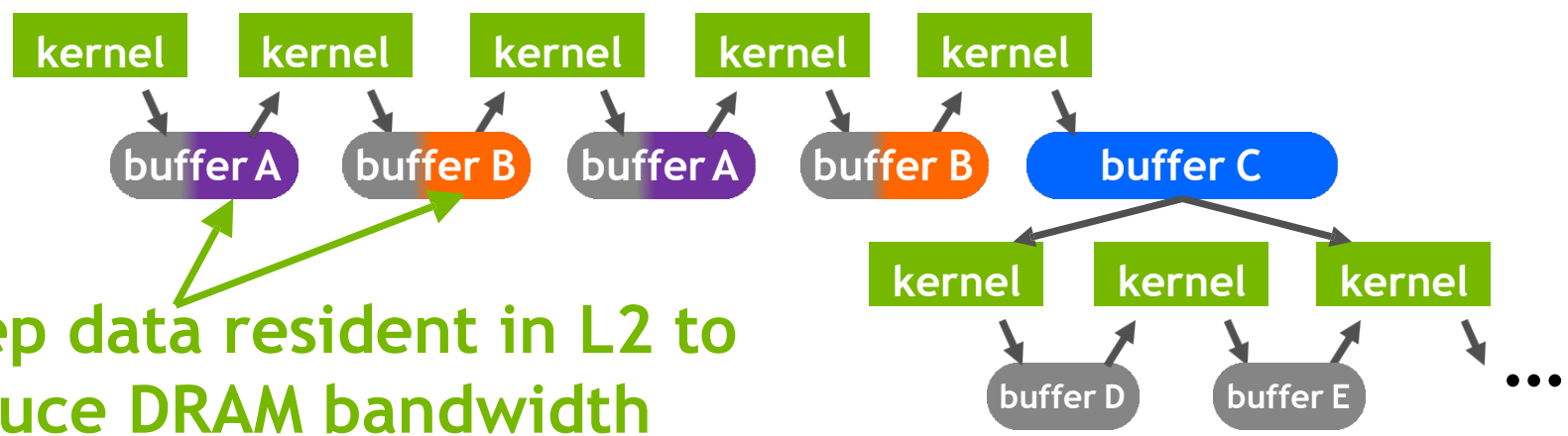
❼ 1.6 TB/s, 1.7x vs. V100

Larger and smarter L2

40MB L2, 6.7x vs. V100

L2-Residency controls

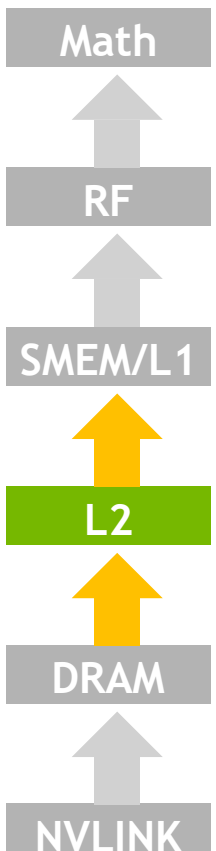




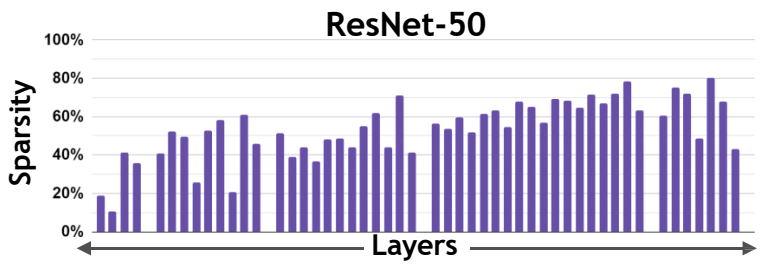
Keep data resident in L2 to reduce DRAM bandwidth

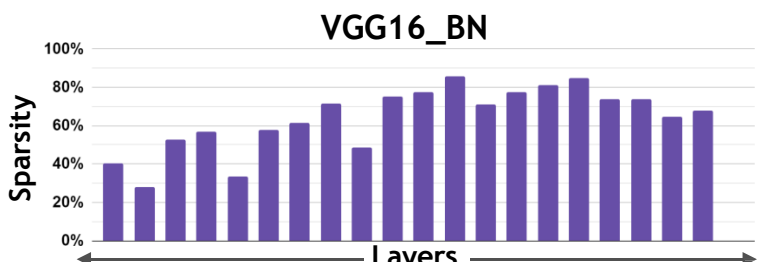
❼ S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT

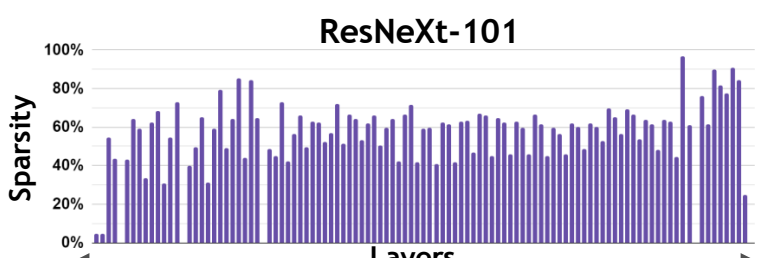
A100 COMPUTE DATA COMPRESSION



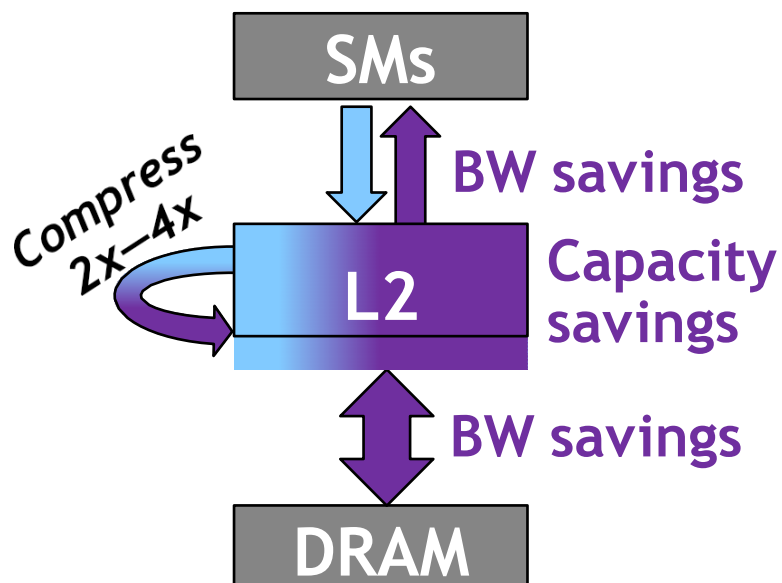
Activation sparsity due to ReLU







Up to 4x DRAM+L2 bandwidth and 2x L2 capacity for fine-grained unstructured sparsity



❼ S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT

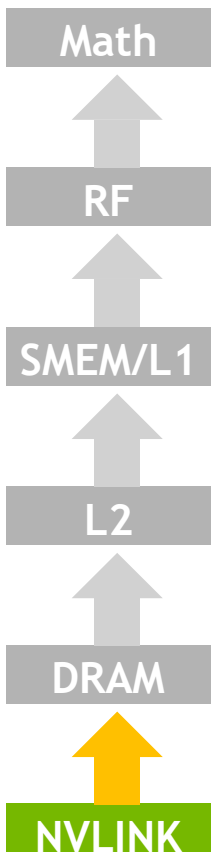
A100 NVLINK BANDWIDTH

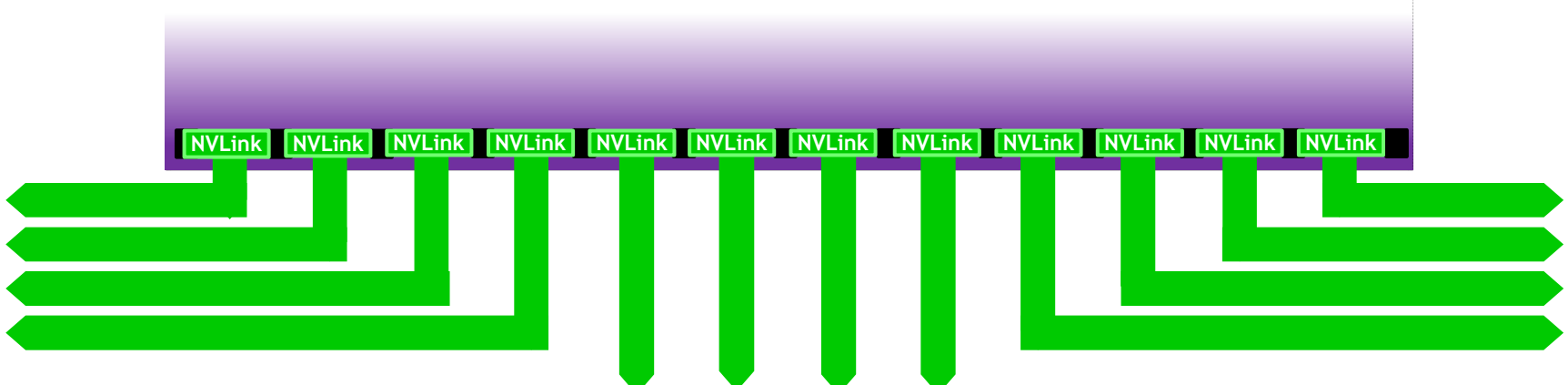
Third Generation NVLink

50 Gbit/sec per signal pair

12 links, 25 GB/s in/out, 600 GB/s total

2x vs. V100

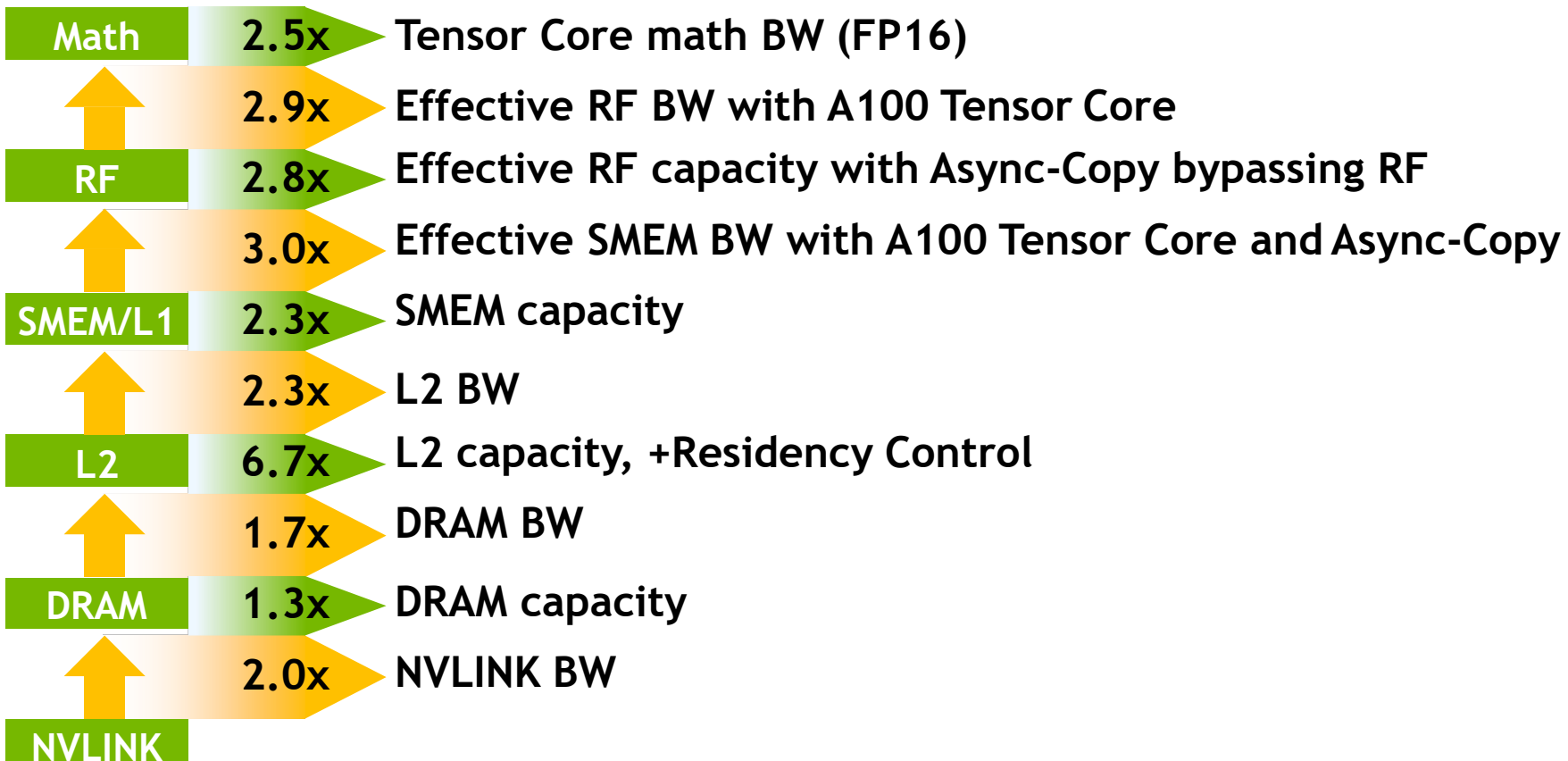




A100 STRONG SCALING INNOVATIONS

Delivering unprecedented levels of performance

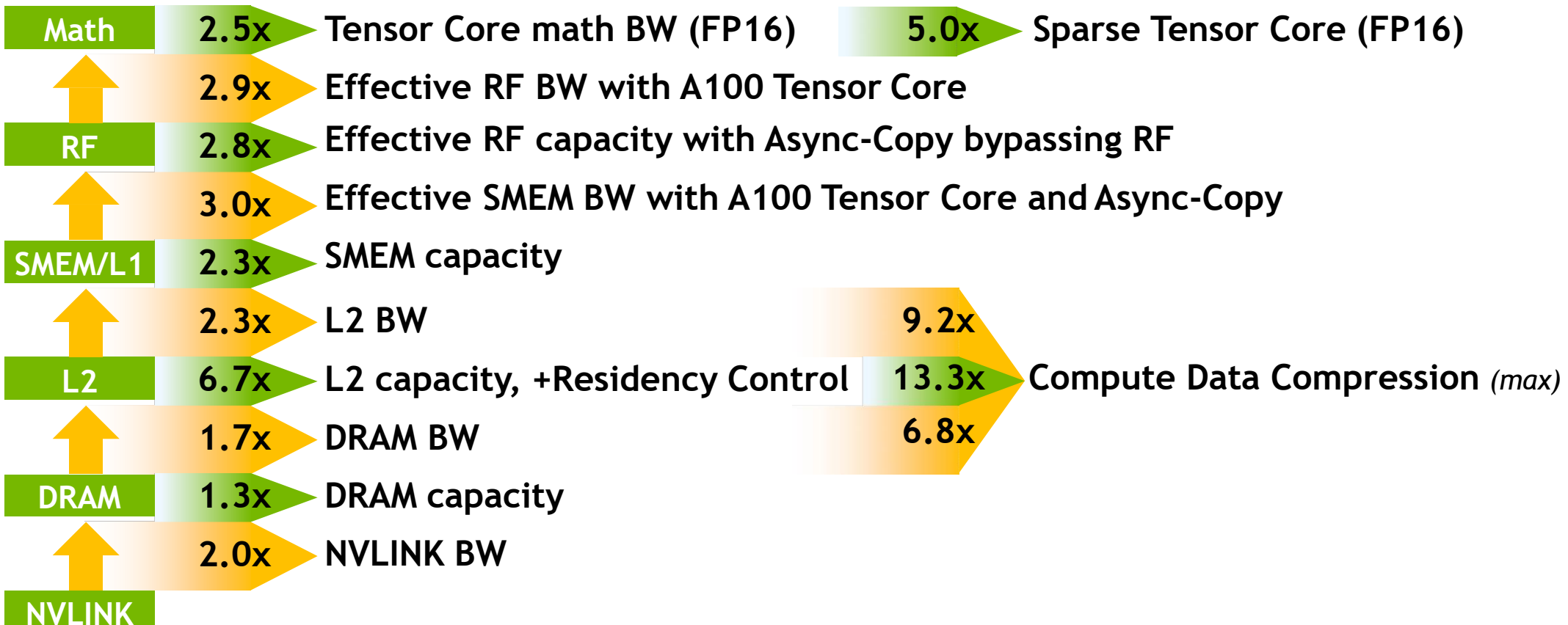
A100 improvements over V100



A100 STRONG SCALING INNOVATIONS

Delivering unprecedented levels of performance

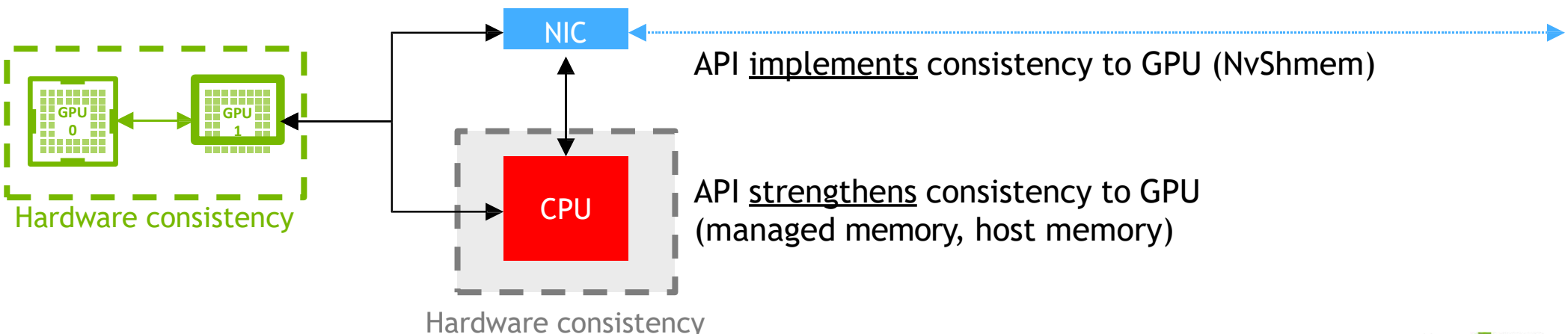
A100 improvements over V100



1. New Tensor Core
2. Strong Scaling
3. **Elastic GPU**
4. Productivity

NVLINK: ONE BIG GPU

- **InfiniBand/Ethernet**: travels a long distance, consistency is the responsibility of software
- **PCI Express**: hardware consistency for I/O, not for programming language memory models
- **NVLINK**: hardware consistency for programming language memory models, like system bus

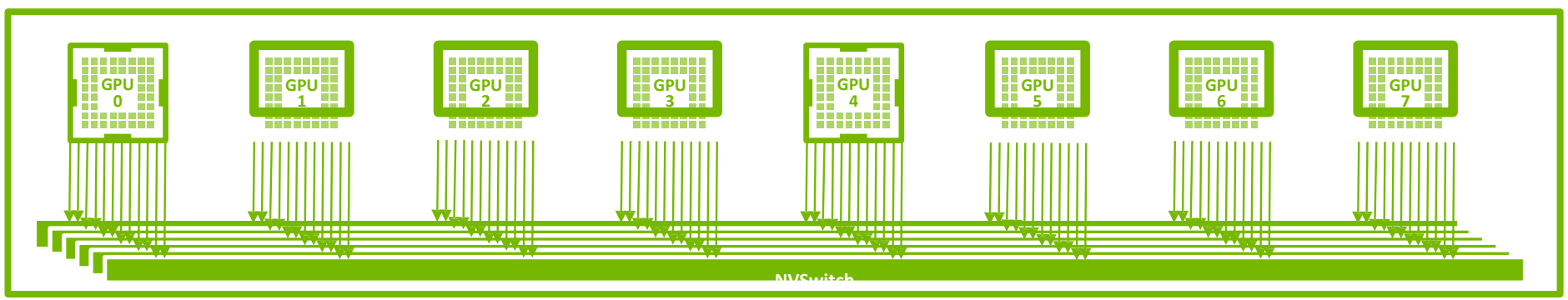


HGX A100: 3RD GEN NVLINK

- **HGX A100 4-GPU:** fully-connected system with 100GB/s all-to-all BW

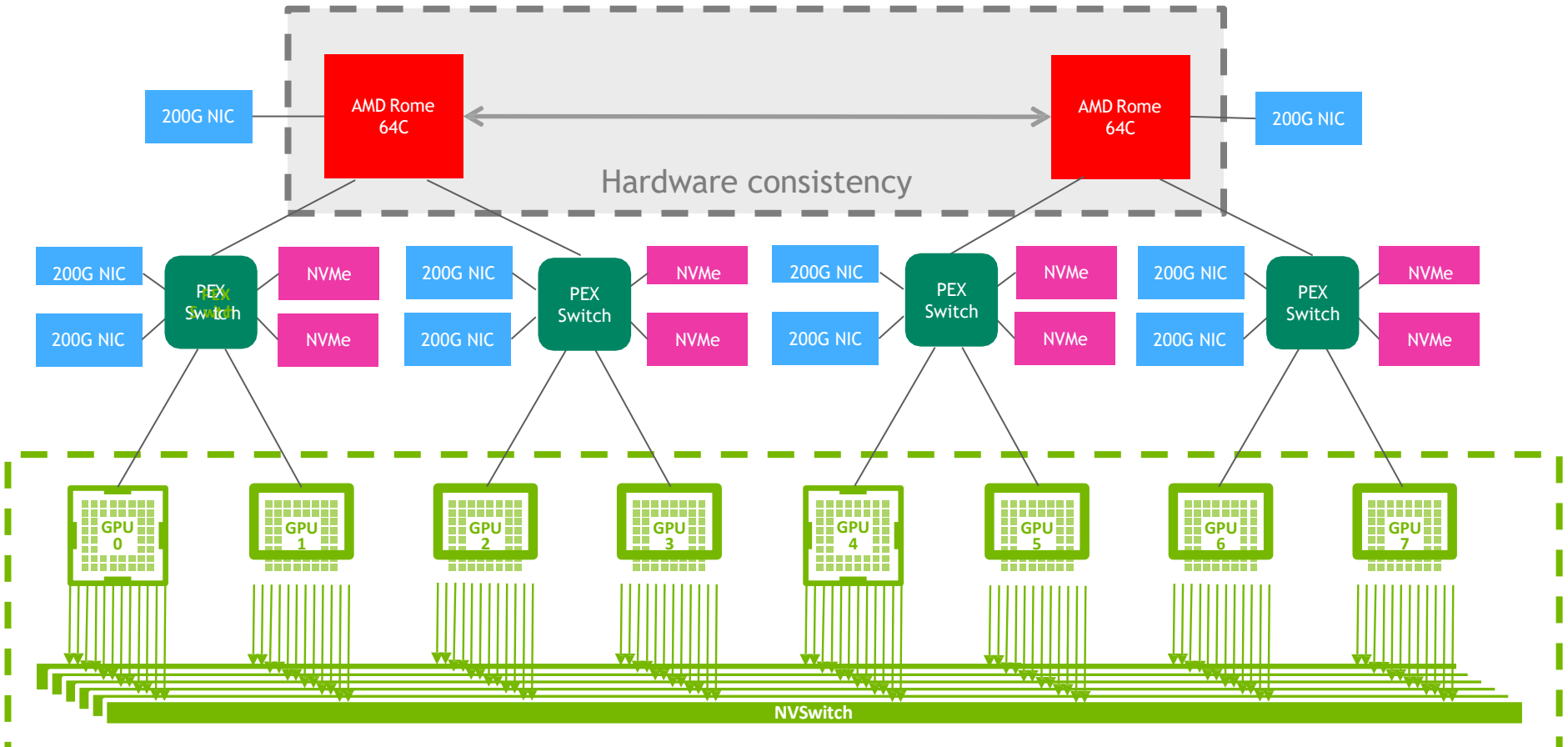
HGX A100: 3RD GEN NVLINK & SWITCH

- **HGX A100 4-GPU:** fully-connected system with 100GB/s all-to-all BW
- **New NVSwitch:** 6B transistors in TSMC 7FF, 36 ports, 25GB/s each, per direction
- **HGX A100 8-GPU:** 6x NVSwitch in a fat tree topology, 2.4TB/s full-duplex bandwidth



Hardware consistency

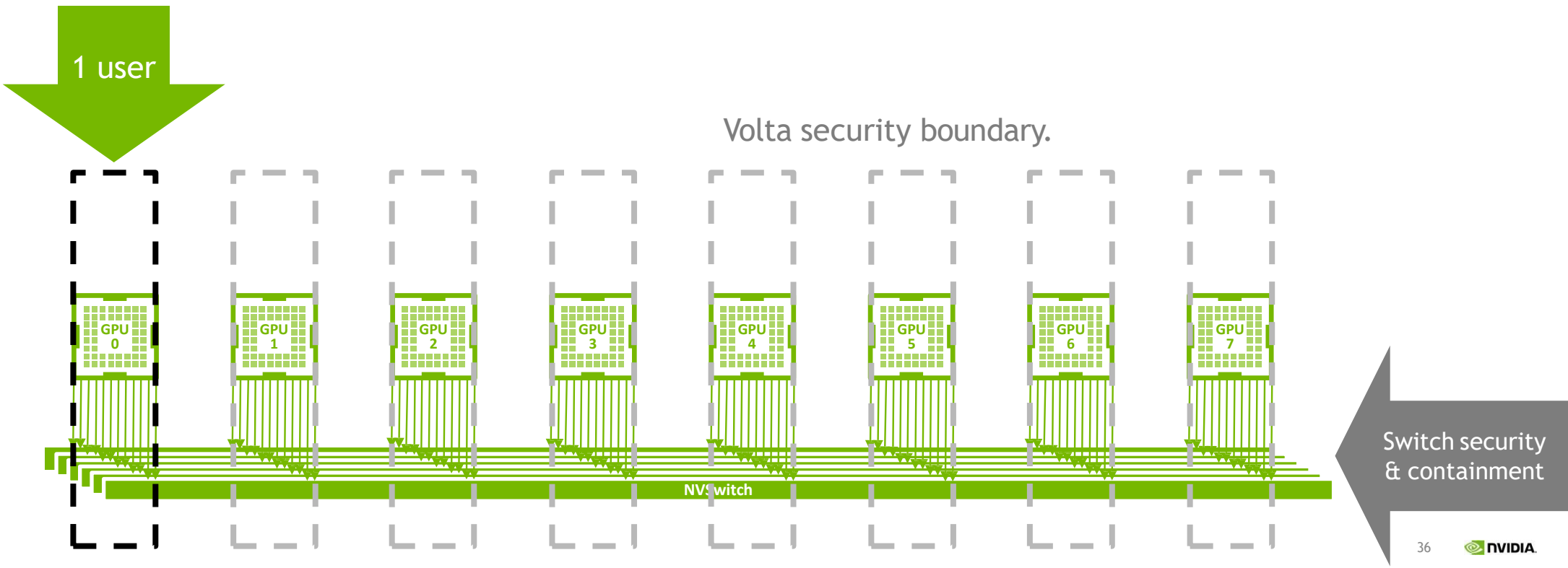
DGX A100: PCIE4 CONTROL & I/O



Hardware consistency

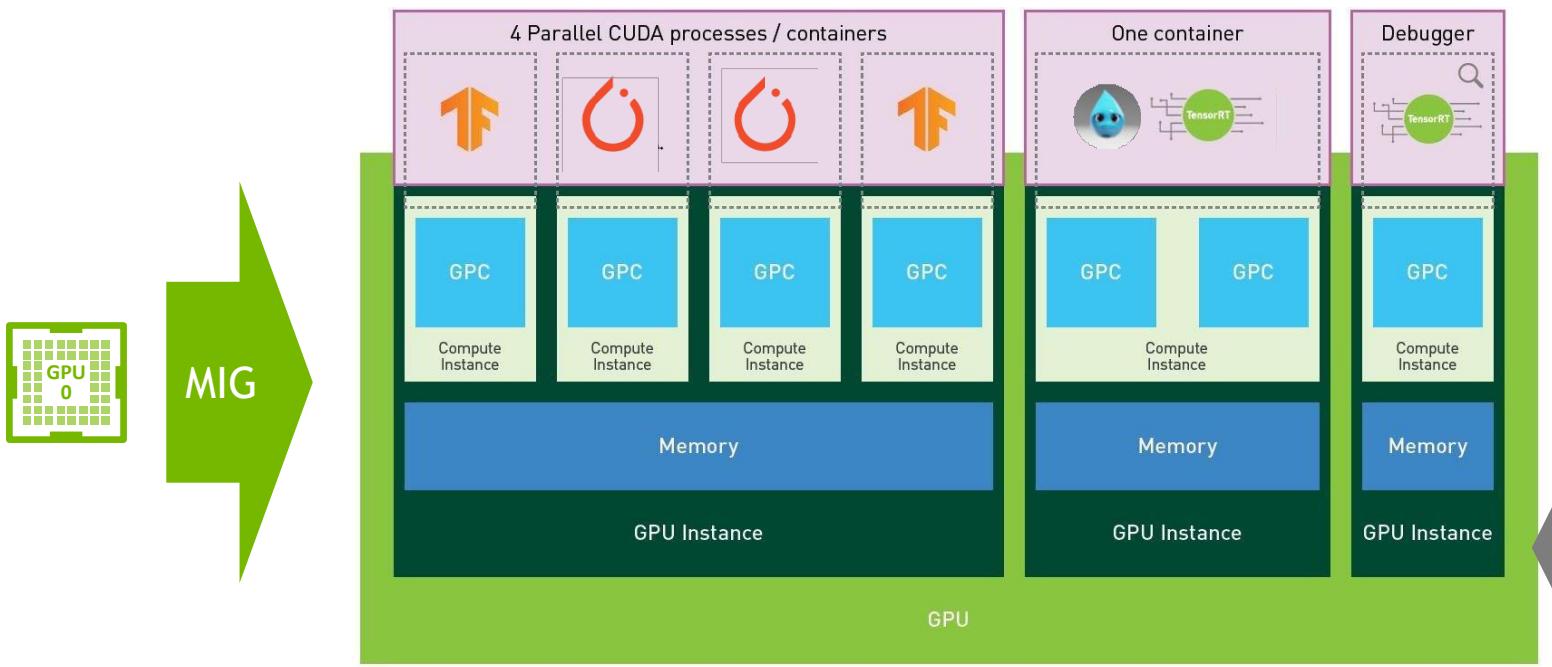
CLOUD SMALL INSTANCE USAGE

- Small workloads can under-utilize GPU cloud instances, provisioned at whole GPU level
- CSPs can't use MPS for GPU space-sharing, because it doesn't provide enough isolation



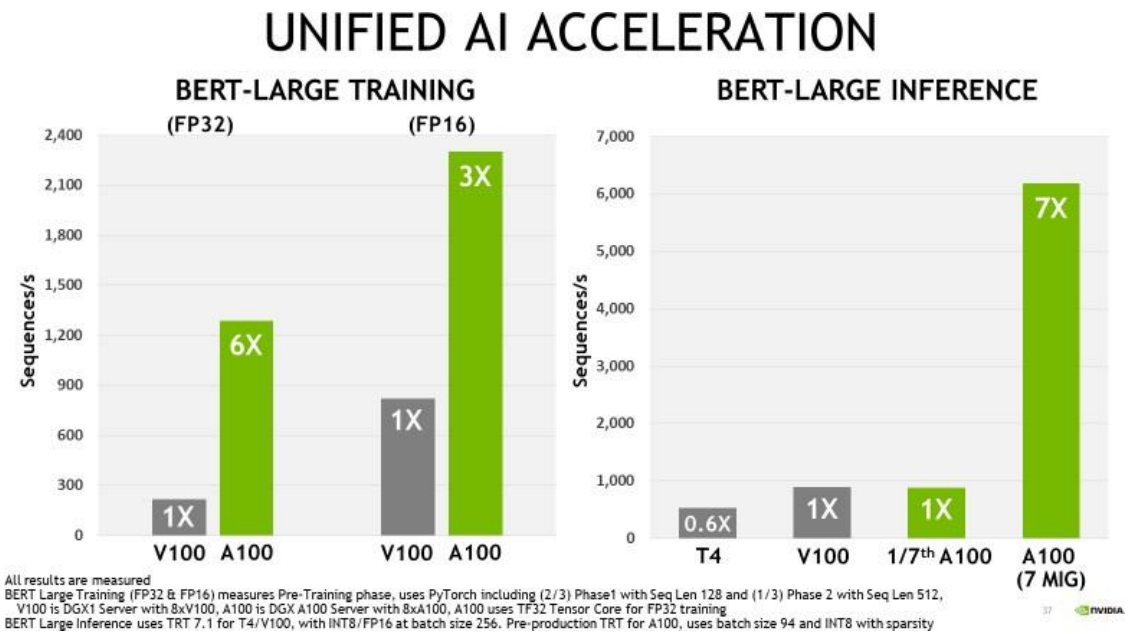
NEW: MULTI-INSTANCE GPU (MIG)

- Up to 7 instances total, dynamically reconfigurable
- **Compute instances:** compute/fault isolation, but share/compete for memory
- **GPU instances:** separate and isolated paths through the entire memory system



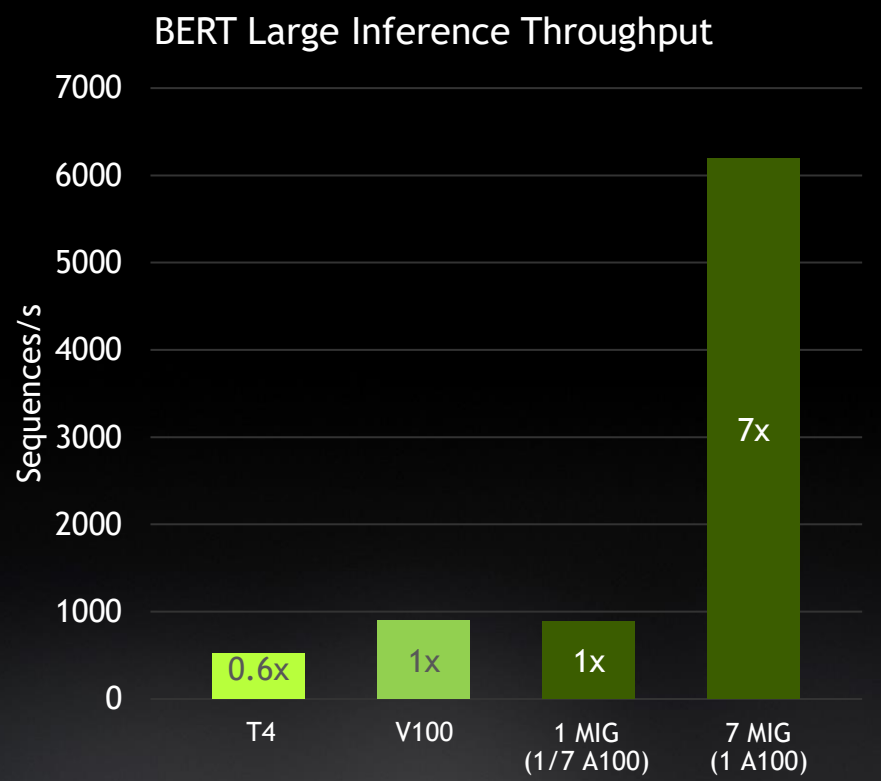
ELASTIC GPU COMPUTING

- Each A100 is 1 to 7 GPUs
- Each DGX A100 is 1 to 56 GPUs
- Each GPU can serve a different user, with full memory isolation and QoS



- S21975: Inside NVIDIA's Multi-Instance GPU Feature (recording available)
- S21884: Under the Hood of the new DGX A100 System Architecture (recording available soon)
- S21702: Introducing NVIDIA DGX A100: The Universal AI System for Enterprise, 5/20 9:00am PDT

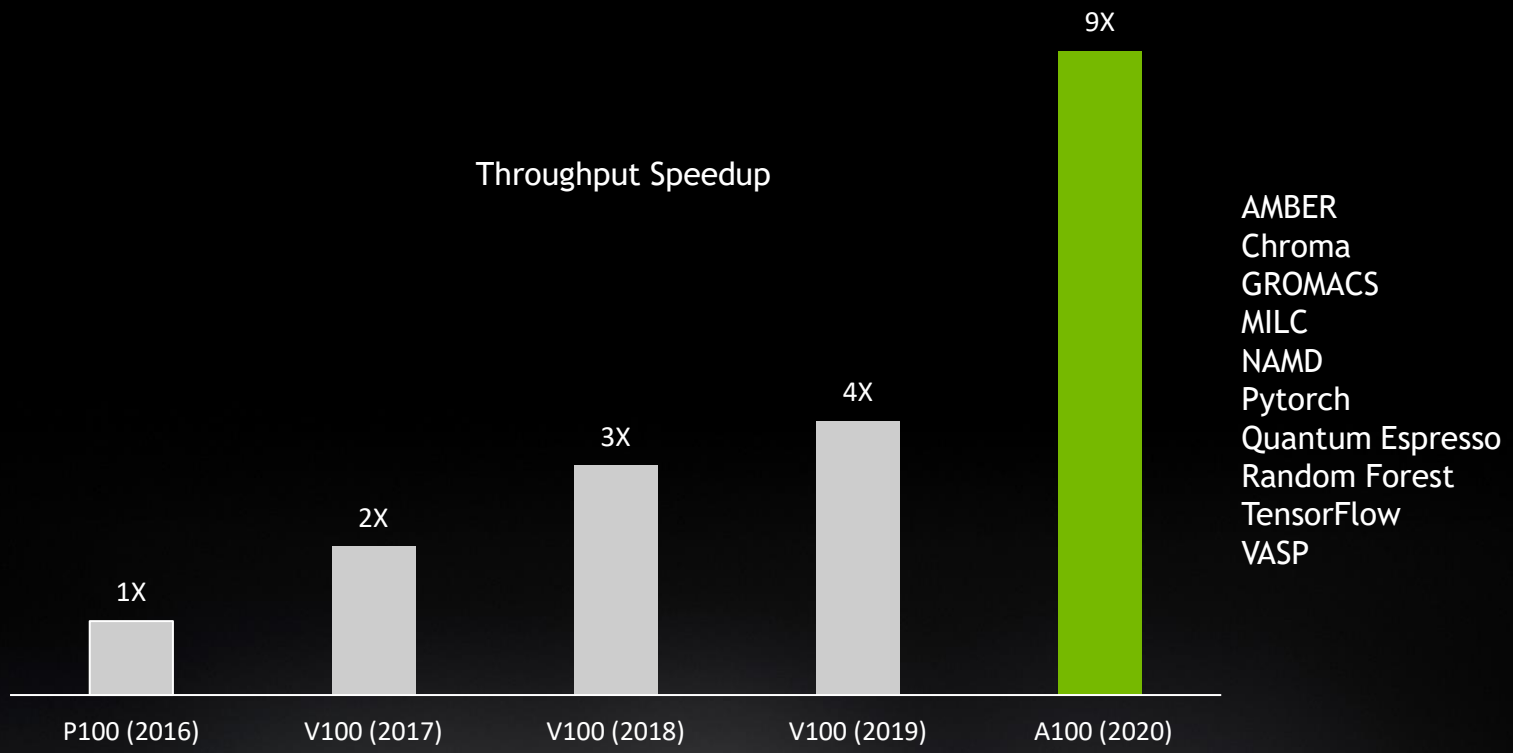
7X HIGHER INFERENCE THROUGHPUT WITH MIG



BERT Large Inference | T4: TRT 7.1, Precision = INT8, Batch Size =256, V100: TRT 7.1, Precision = FP16, Batch Size =256 | A100 with 7 MIG instances of 1g.5gb : Pre-production TRT, Batch Size =94, Precision = INT8 with Sparsity

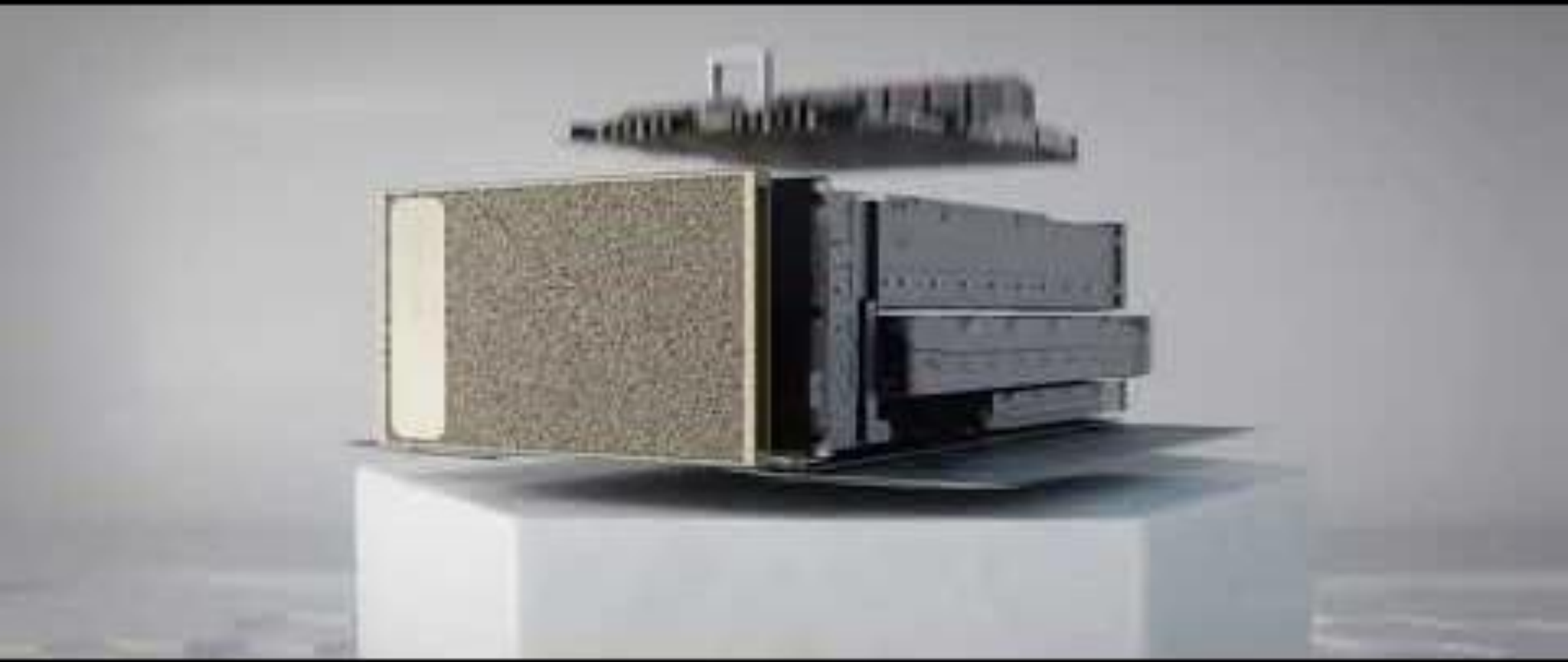
9X MORE PERFORMANCE IN 4 YEARS

Beyond Moore's Law With Full Stack Innovation



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

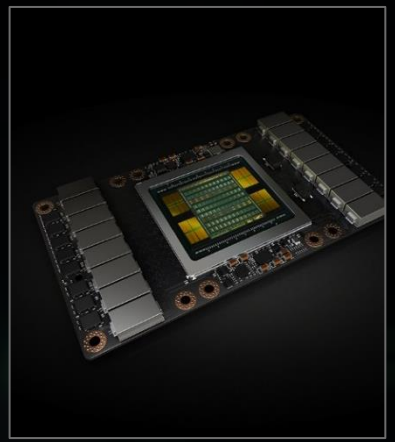
1. New Tensor Core
2. Strong Scaling
3. Elastic GPU
4. **Productivity**

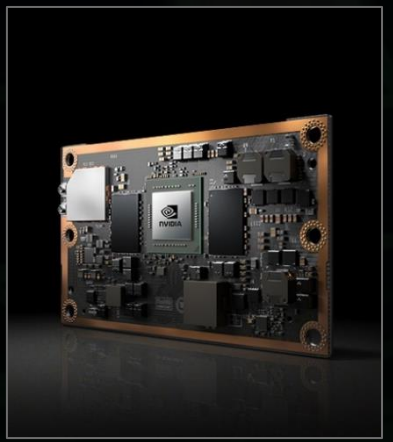


https://www.youtube.com/watch?v=TJcKYUTaBtg&t=3s

NVIDIA

One Platform. All challenges.



ONE ARCHITECTURE — CUDA

KEY ANNOUNCEMENT ASSETS

JHH Keynote On Demand

Tuesday, May 19th

Inside the NVIDIA Ampere Architecture
09:00 AM - 10:00 AM

CUDA New Features And Beyond
10:15 AM - 11:15 AM

CUDA on NVIDIA Ampere GPU Architecture: Taking Your Algorithms to the Next Level of Performance
11:30 AM - 12:30 PM

Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing
1:30 PM - 2:30 PM

Wednesday, May 20th

Introducing NVIDIA DGX A100: The Universal AI System for Enterprise
9:00 AM - 10:00 AM

Accelerating Deep Learning Inference With Sparse Tensor Cores of Ampere GPU Architecture
1:30 PM - 2:30 PM

Mixed-Precision Training of Neural Networks
2:45 PM - 3:45 PM

Thursday, May 21st

Tensor Core Performance on NVIDIA GPUs: The Ultimate Guide
9:00 AM - 10:00 AM

Optimizing Applications for NVIDIA Ampere GPU Architecture
10:15 AM - 11:15 AM

Developing CUDA kernels to push Tensor Cores to the Absolute Limit on NVIDIA A100
11:30 AM - 12:30 PM

High-Performance Next-Generation Deep-Learning Clusters
1:30 PM - 2:30


NVIDIA®