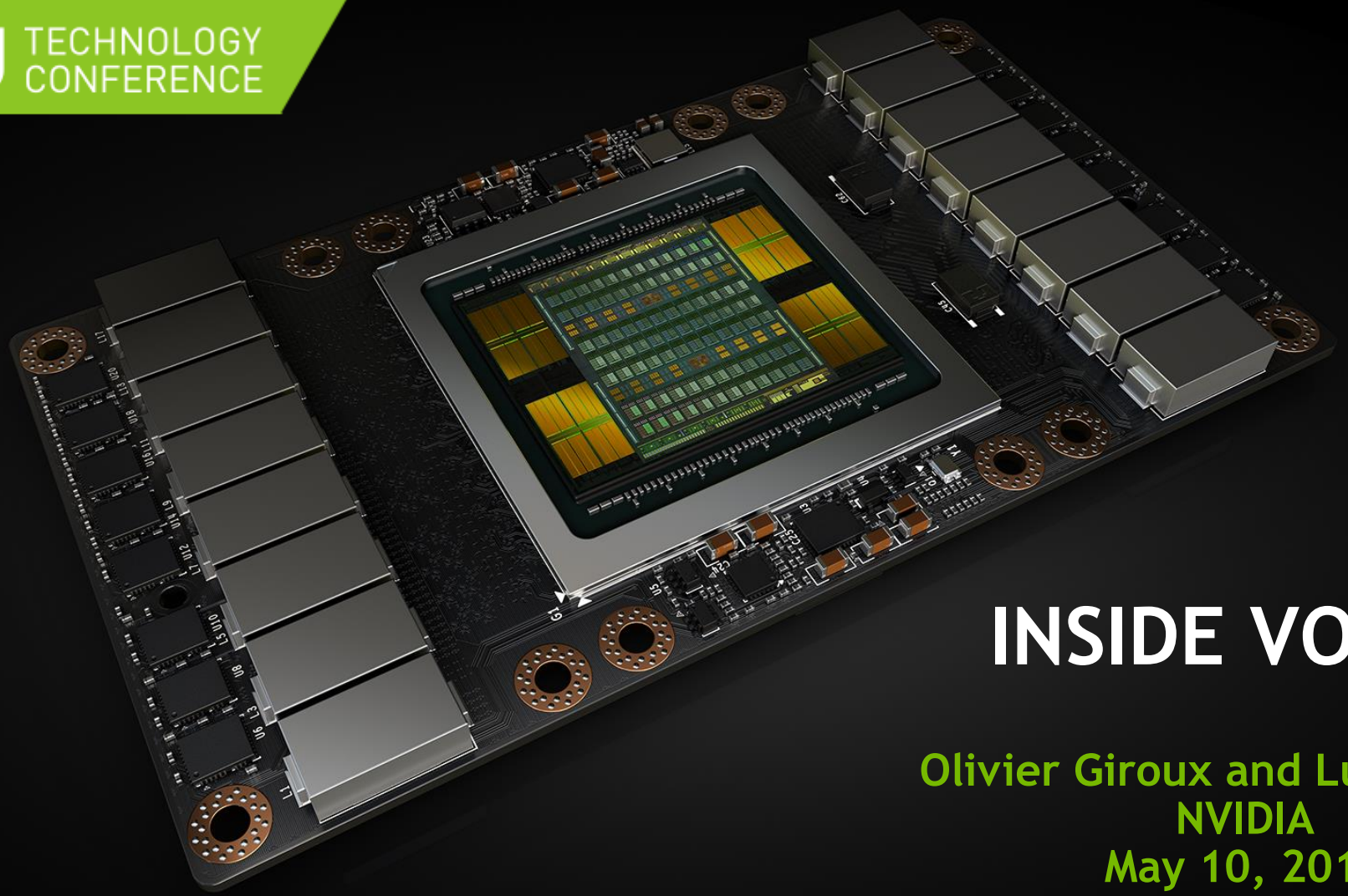
GPU TECHNOLOGY CONFERENCE

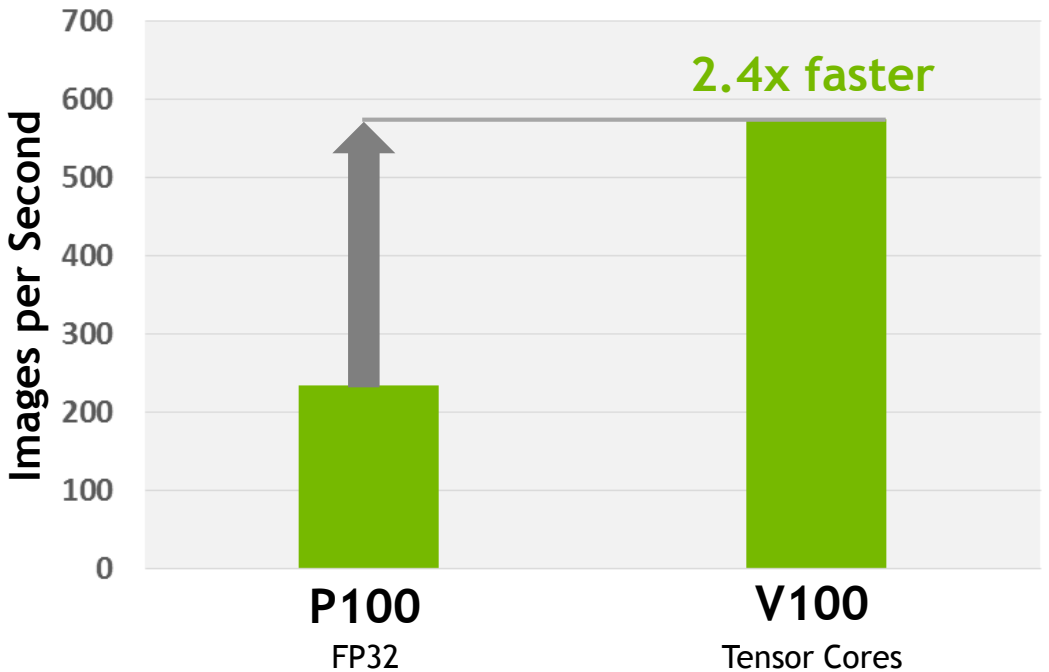


INSIDE VOLTA

Olivier Giroux and Luke Durant
NVIDIA
May 10, 2017

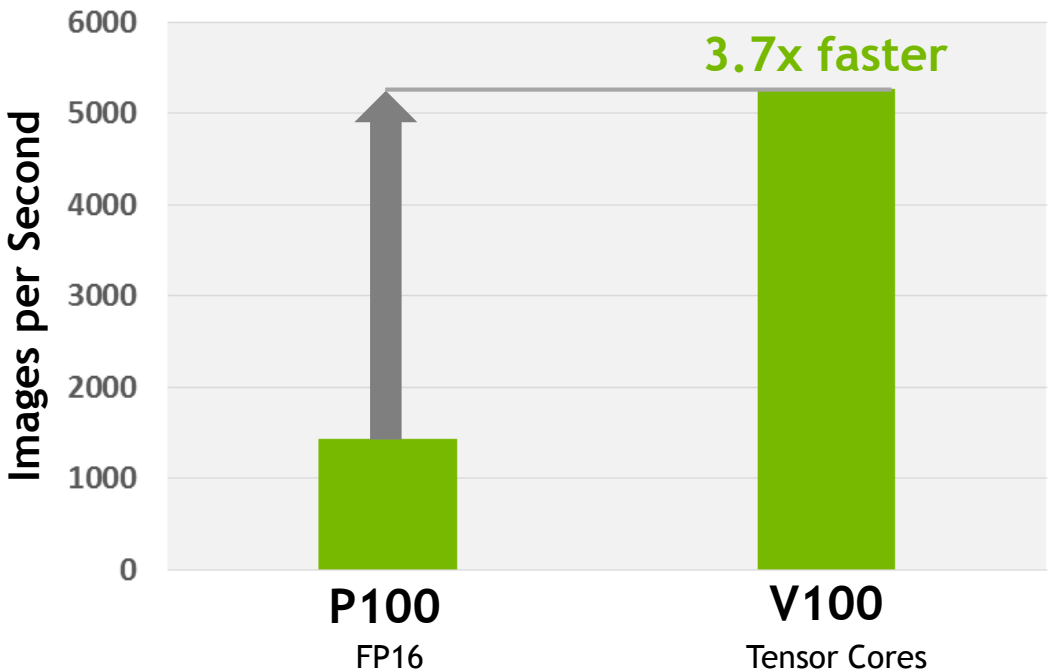
VOLTA: A GIANT LEAP FOR DEEP LEARNING

ResNet-50 Training



ResNet-50 Inference

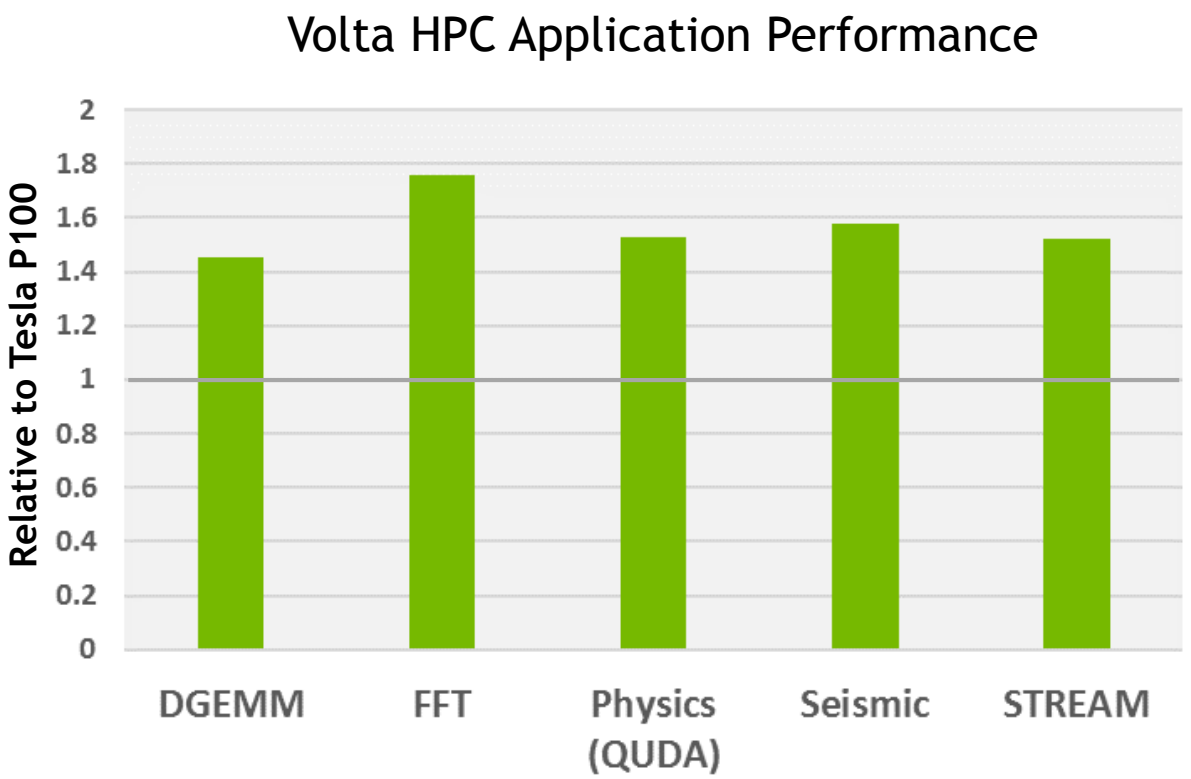
TensorRT - 7ms Latency



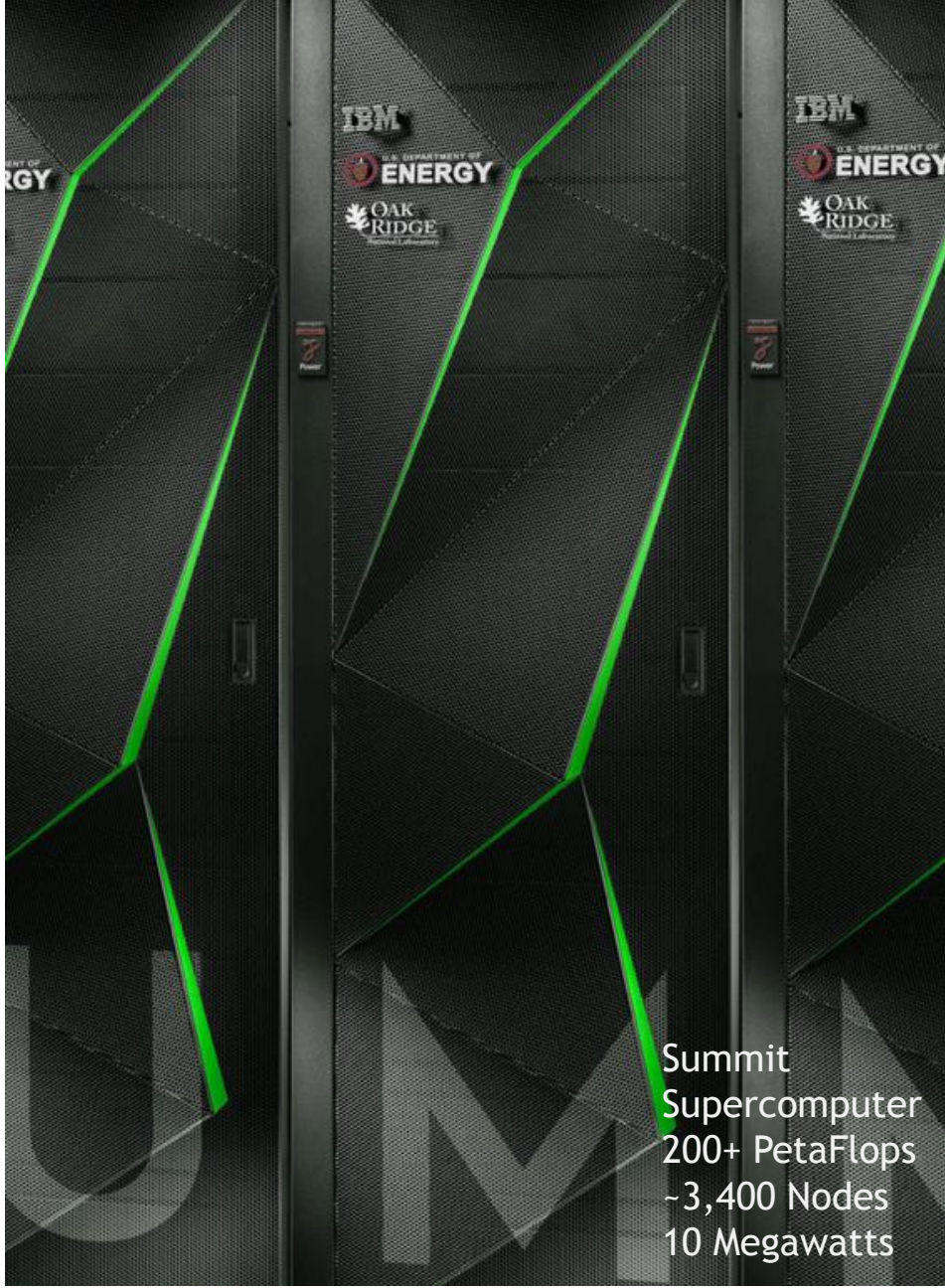
V100 measured on pre-production hardware.

ROAD TO EXASCALE

Volta to Fuel Most Powerful US Supercomputers

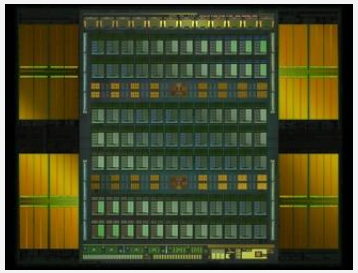


System Config Info: 2X Xeon E5-2690 v4, 2.6GHz, w/ 1X Tesla P100 or V100. V100 measured on pre-production hardware.



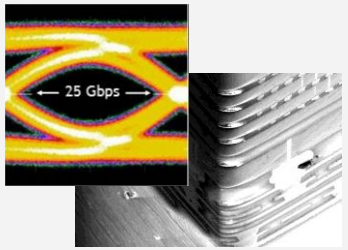
INTRODUCING TESLA V100

Volta Architecture



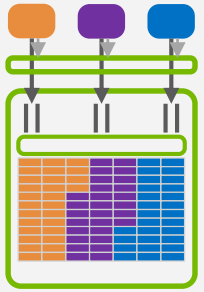
Most Productive GPU

Improved NVLink & HBM2



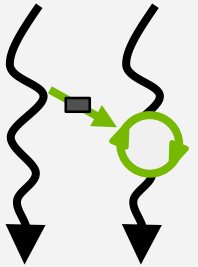
Efficient Bandwidth

Volta MPS



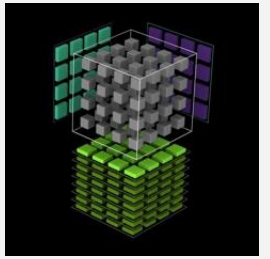
Inference Utilization

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

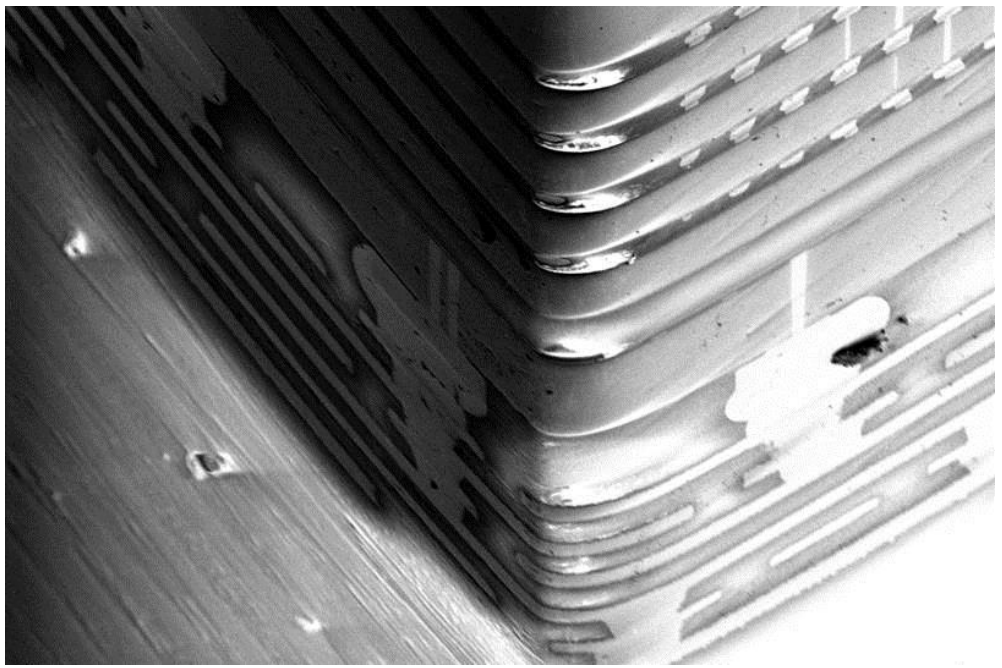
16 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

*full GV100 chip contains 84 SMs

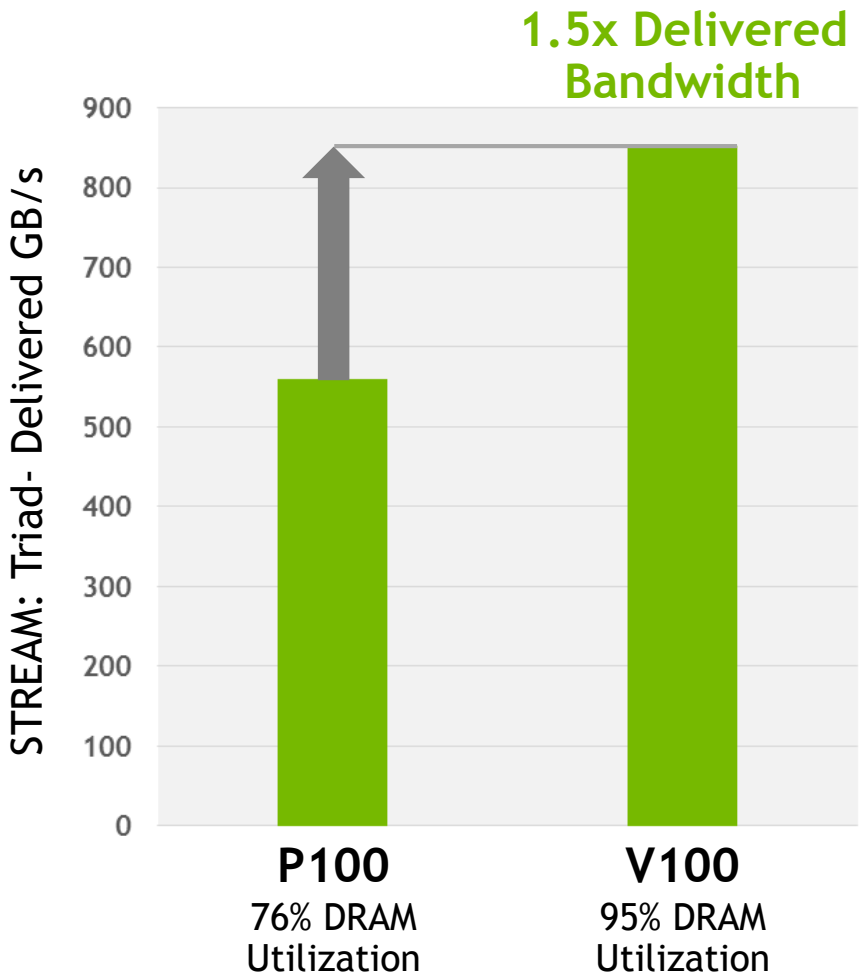
GPU PERFORMANCE COMPARISON

	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

NEW HBM2 MEMORY ARCHITECTURE



HBM2 stack



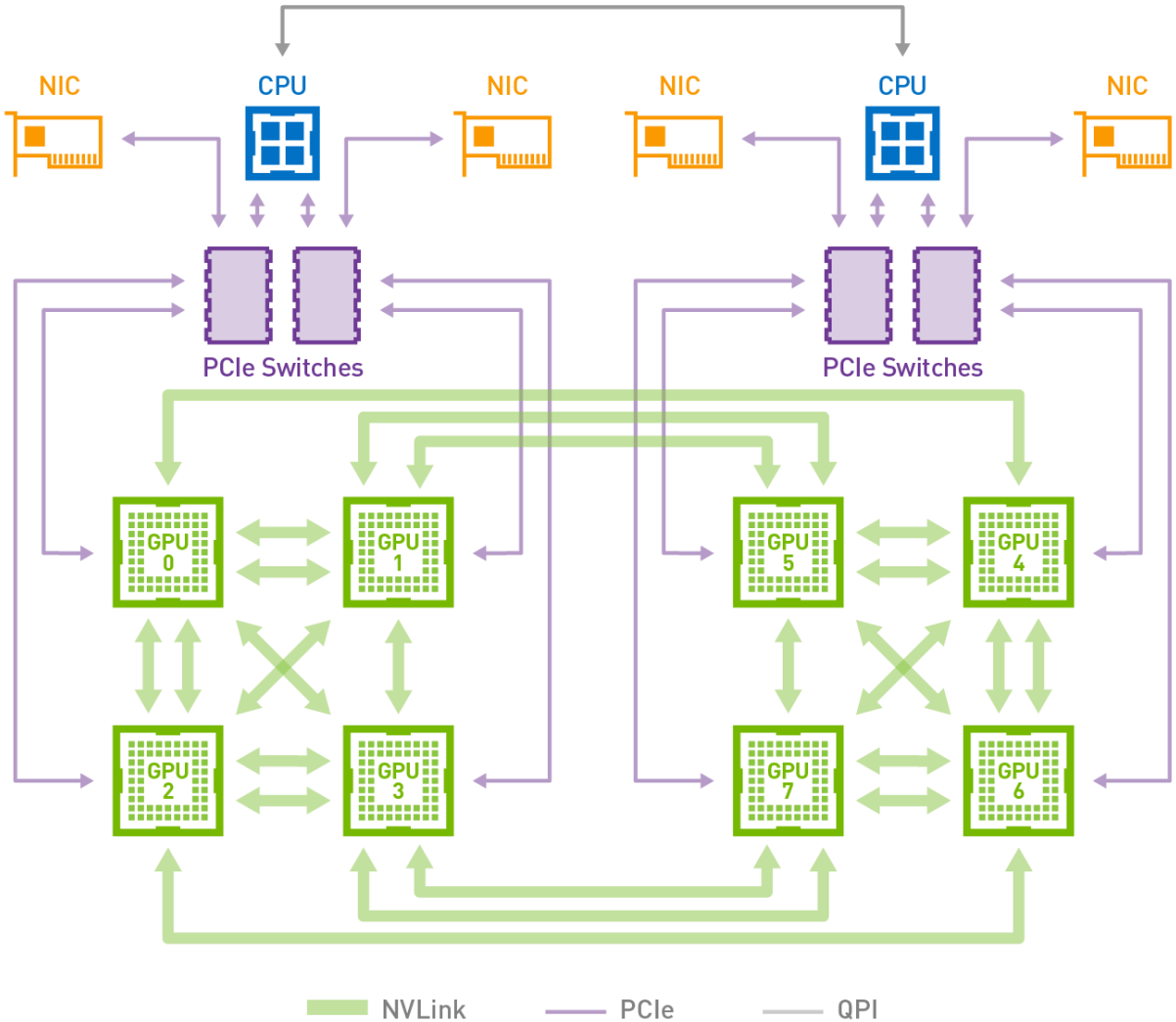
V100 measured on pre-production hardware.

VOLTA NVLINK

300GB/sec

50% more links

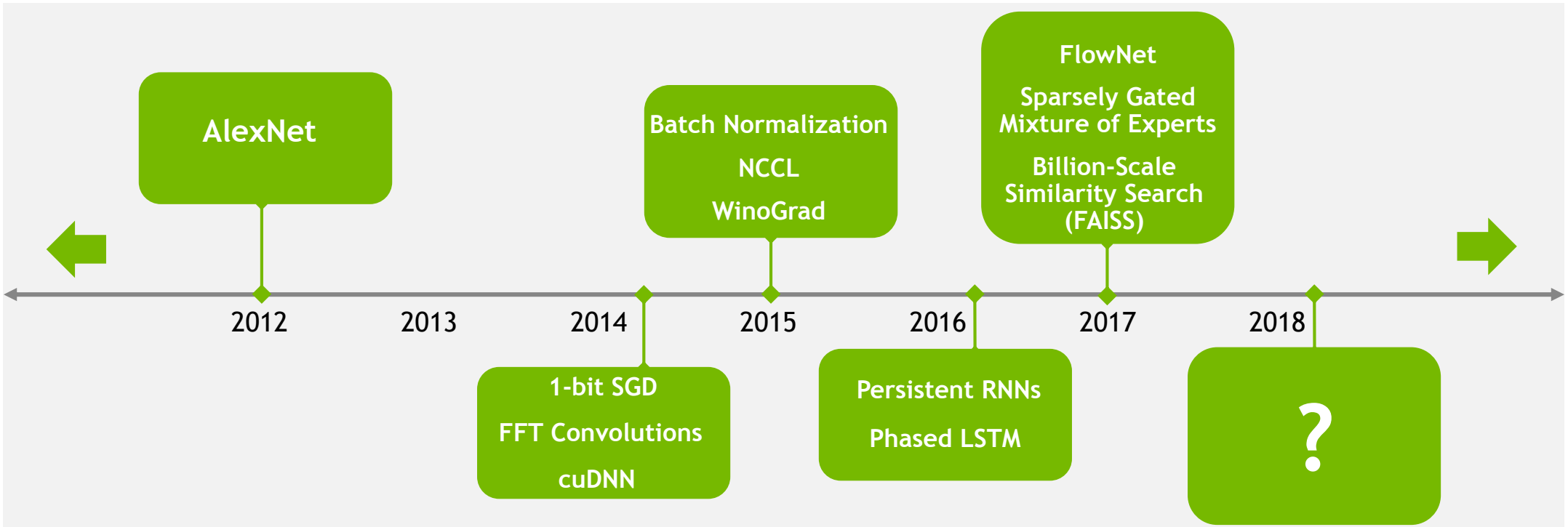
28% faster signaling



PROGRAMMABILITY

PROGRAMMABILITY DRIVES DEEP LEARNING

Deep Learning methods developed using CUDA



New solvers, new layers, new scaling techniques, new applications for old techniques, and much more…

STATE OF UNIFIED MEMORY

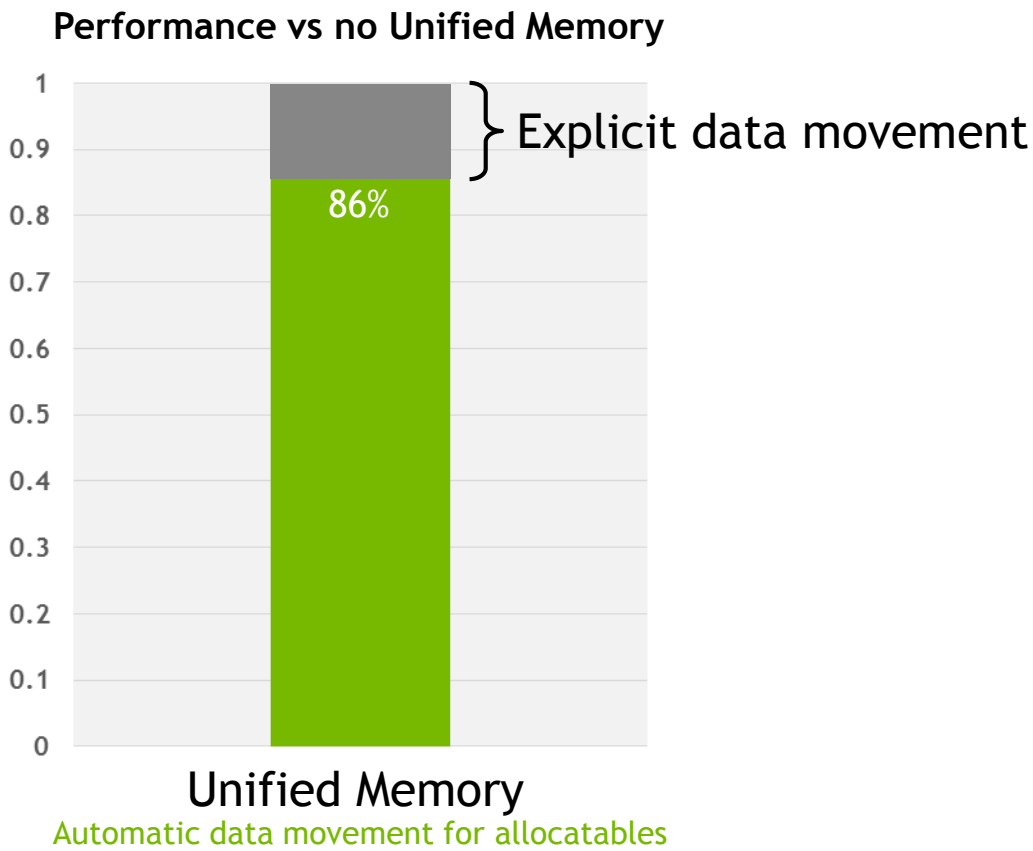
High performance, low effort

PGI OpenACC on Pascal P100

Geometric mean across all 15 SPEC ACCEL™ benchmarks

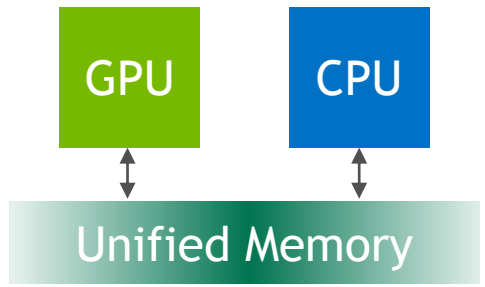
86% PCI-E, 91% NVLink

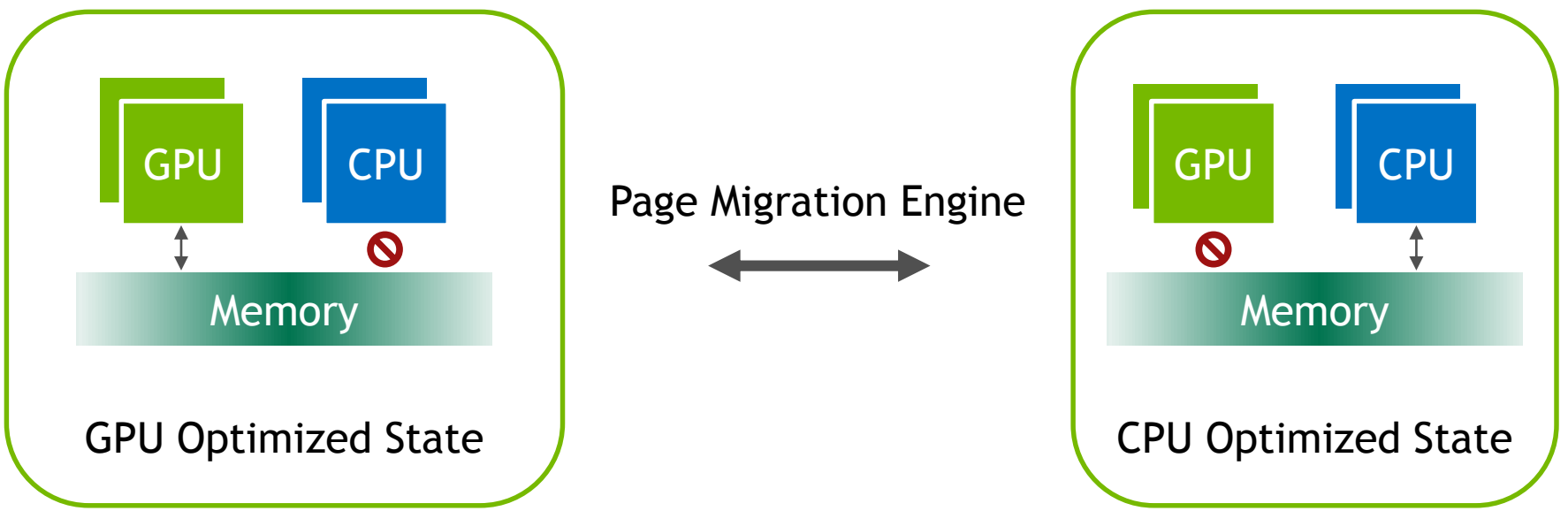
S7285 – Unified Memory on the Latest GPU Architectures



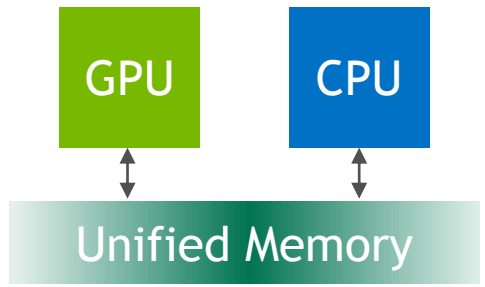
PGI 17.1 Compilers OpenACC SPEC ACCEL™ 1.1 performance measured March, 2017. SPEC® and the benchmark name SPEC ACCEL™ are registered trademarks of the Standard Performance Evaluation Corporation.

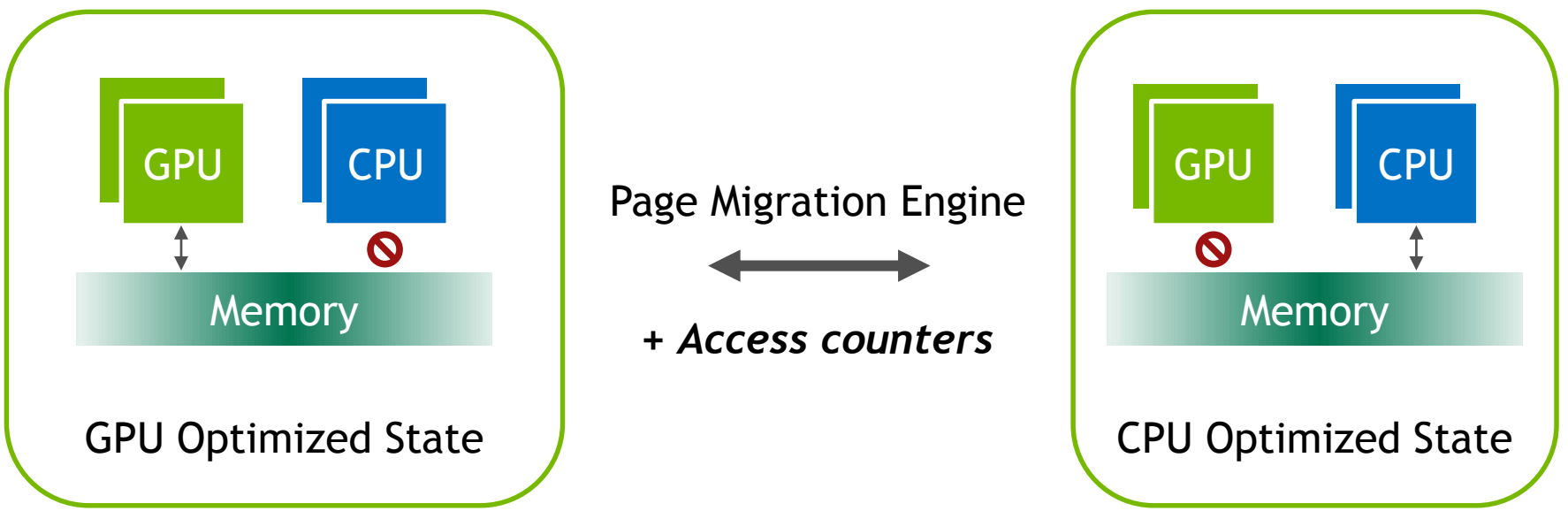
PASCAL UNIFIED MEMORY



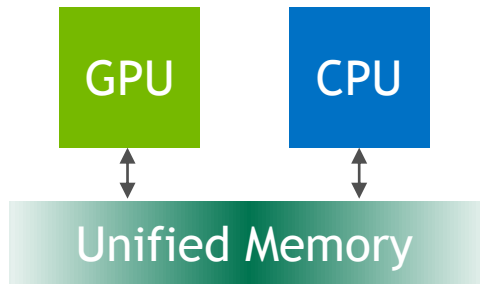


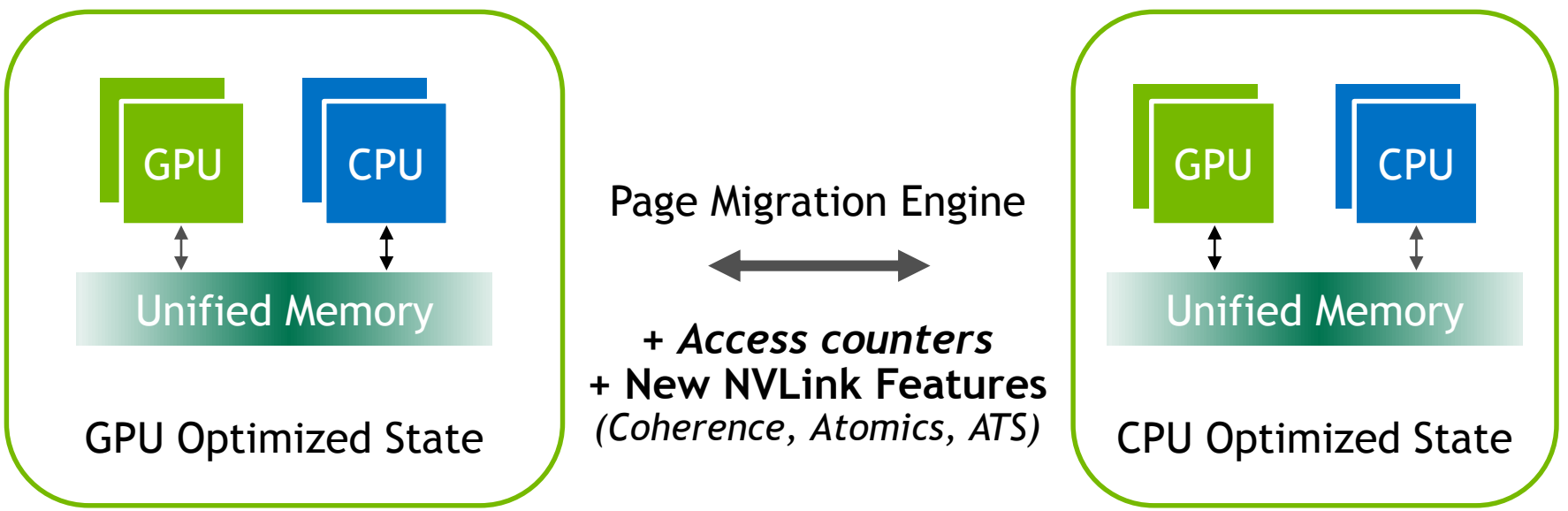
VOLTA + PCIE CPU UNIFIED MEMORY





VOLTA + NVLINK CPU UNIFIED MEMORY





GPU MULTI-PROCESS SCHEDULING

Background

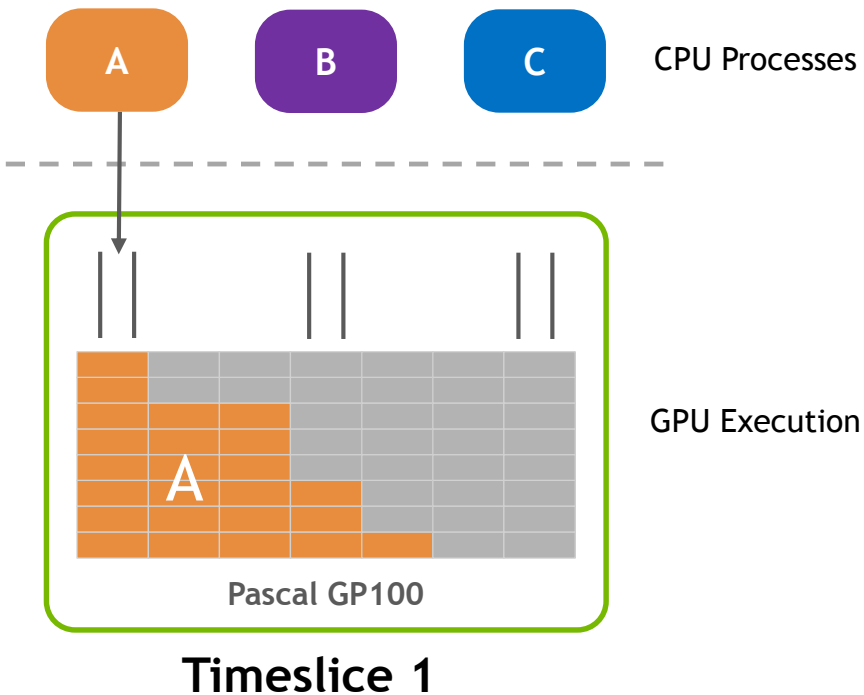
Timeslice Scheduling

Single-process throughput optimized

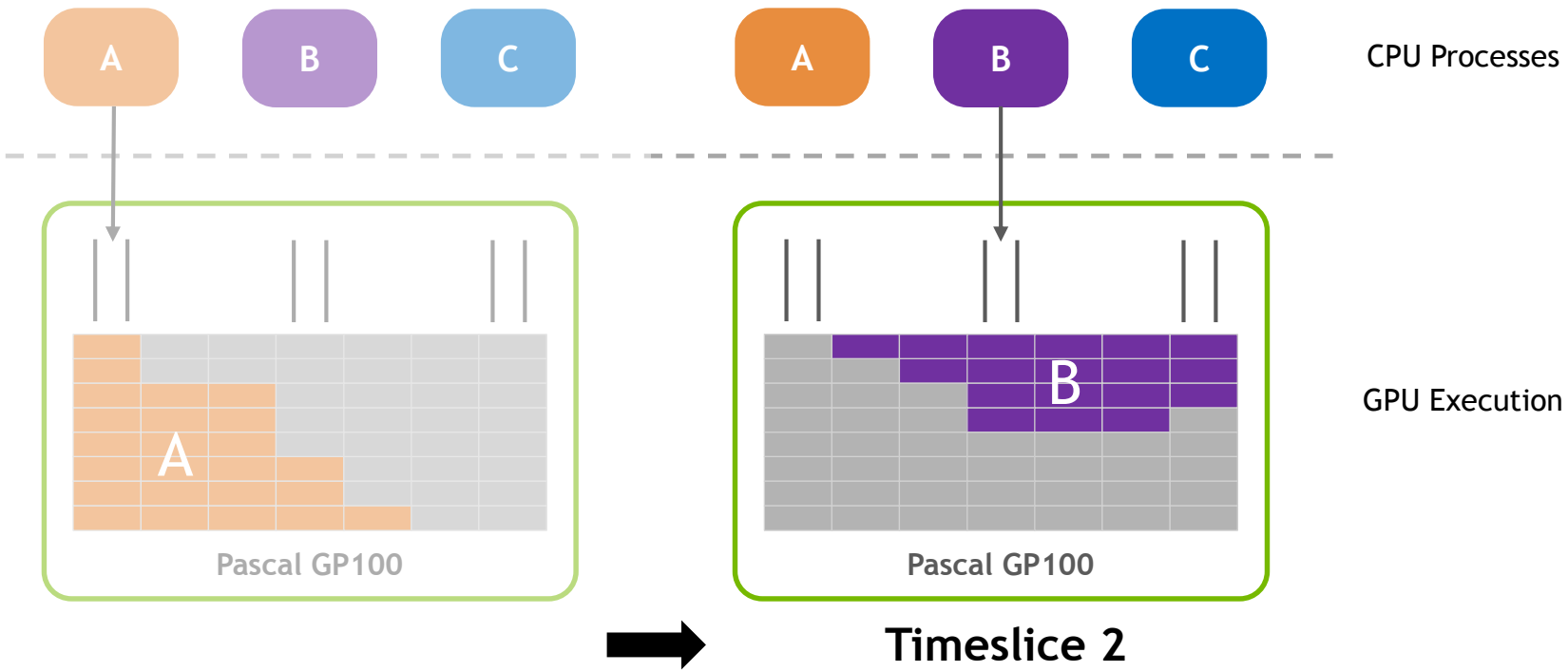
Multi-Process Service

Multi-process throughput optimized

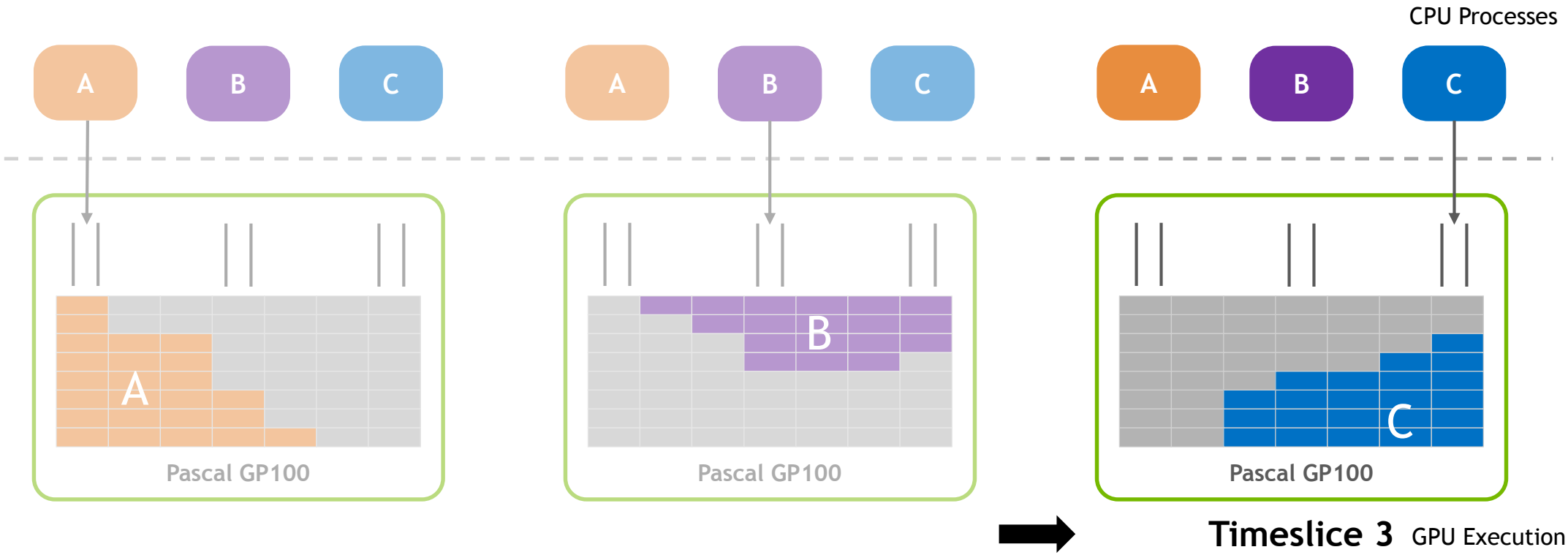
MULTI-PROCESS TIMESLICING



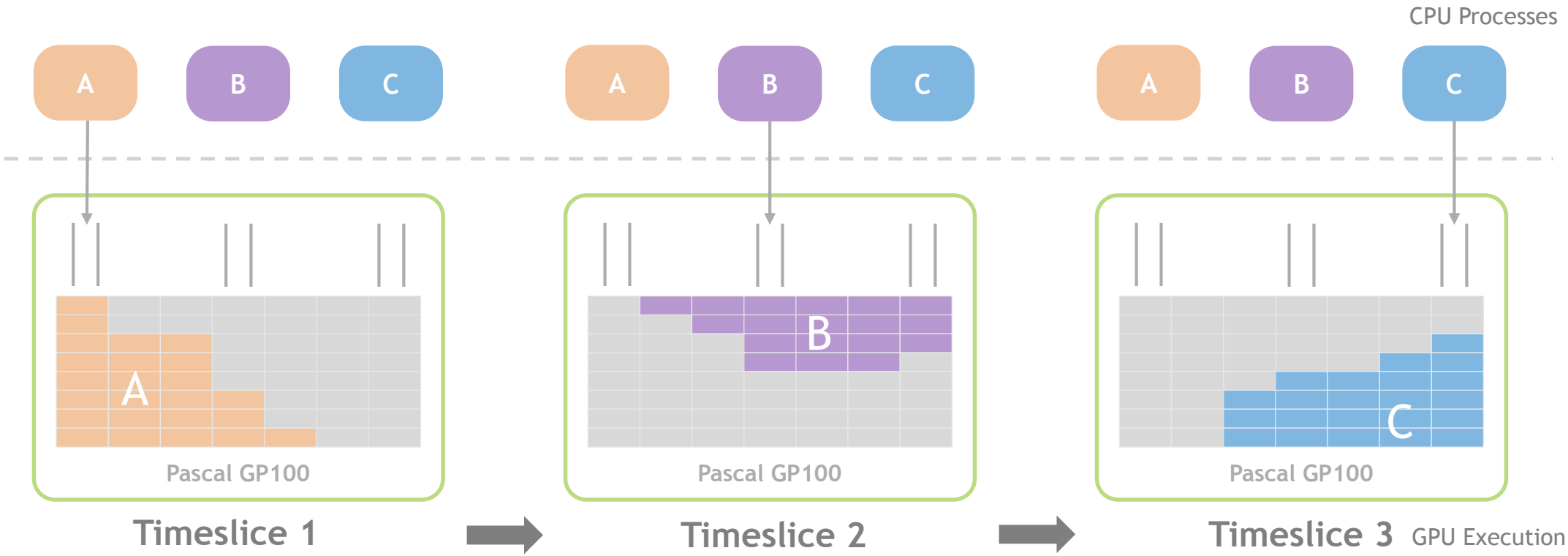
MULTI-PROCESS TIMESLICING



MULTI-PROCESS TIMESLICING



MULTI-PROCESS TIMESLICING

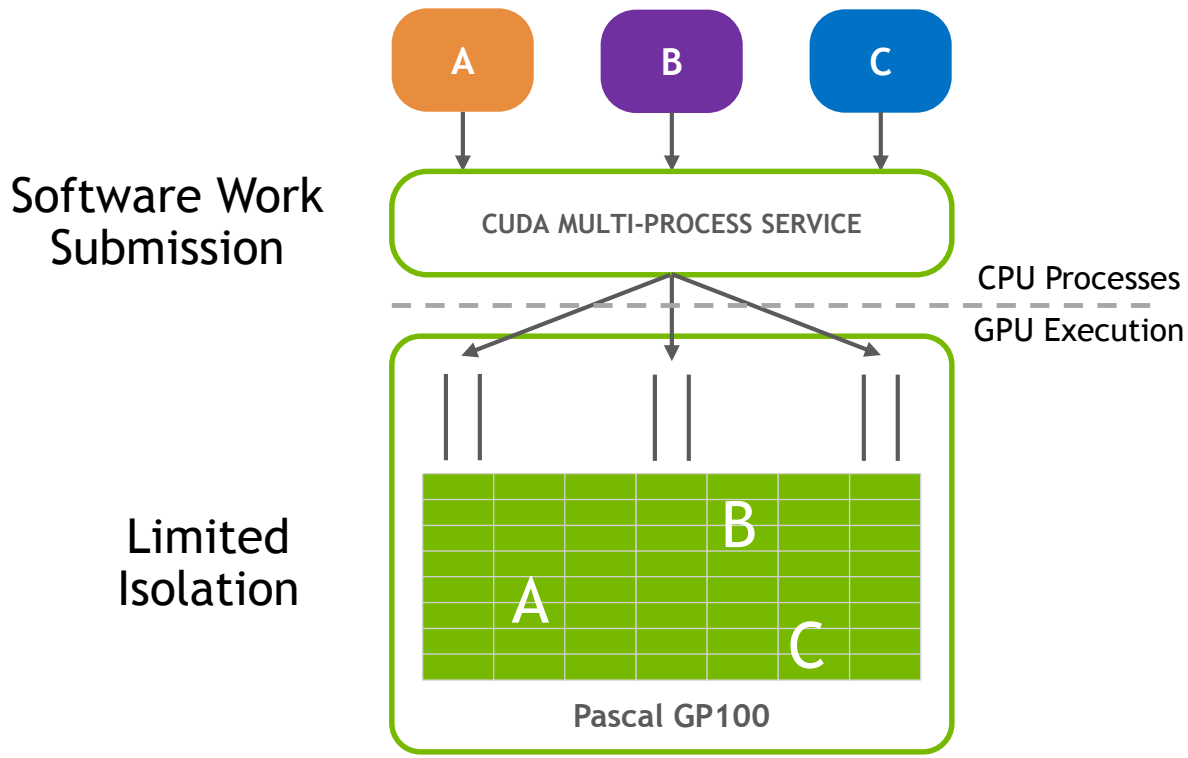


Full process isolation, peak throughput optimized for each process

PASCAL MULTI-PROCESS SERVICE

CUDA Multi-Process Service:

Improve GPU utilization by sharing resources amongst small jobs



Opt-in: Limited isolation, peak throughput optimized **across** processes

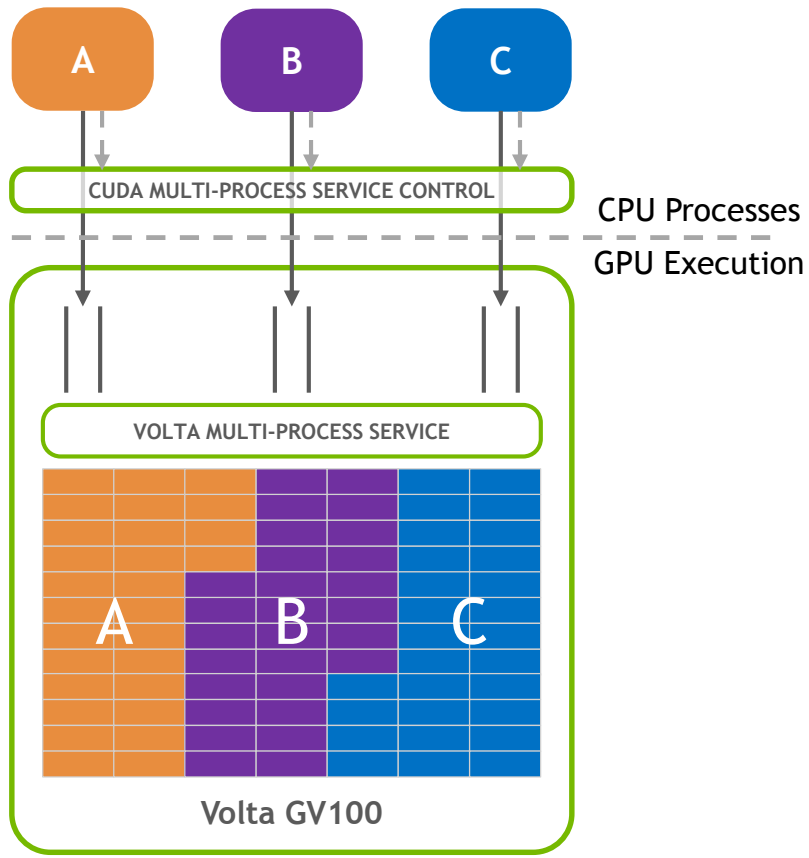
VOLTA MULTI-PROCESS SERVICE

Volta MPS Enhancements:

- Reduced launch latency
- Improved launch throughput
- Improved quality of service with scheduler partitioning
  - More reliable performance
- 3x more clients than Pascal

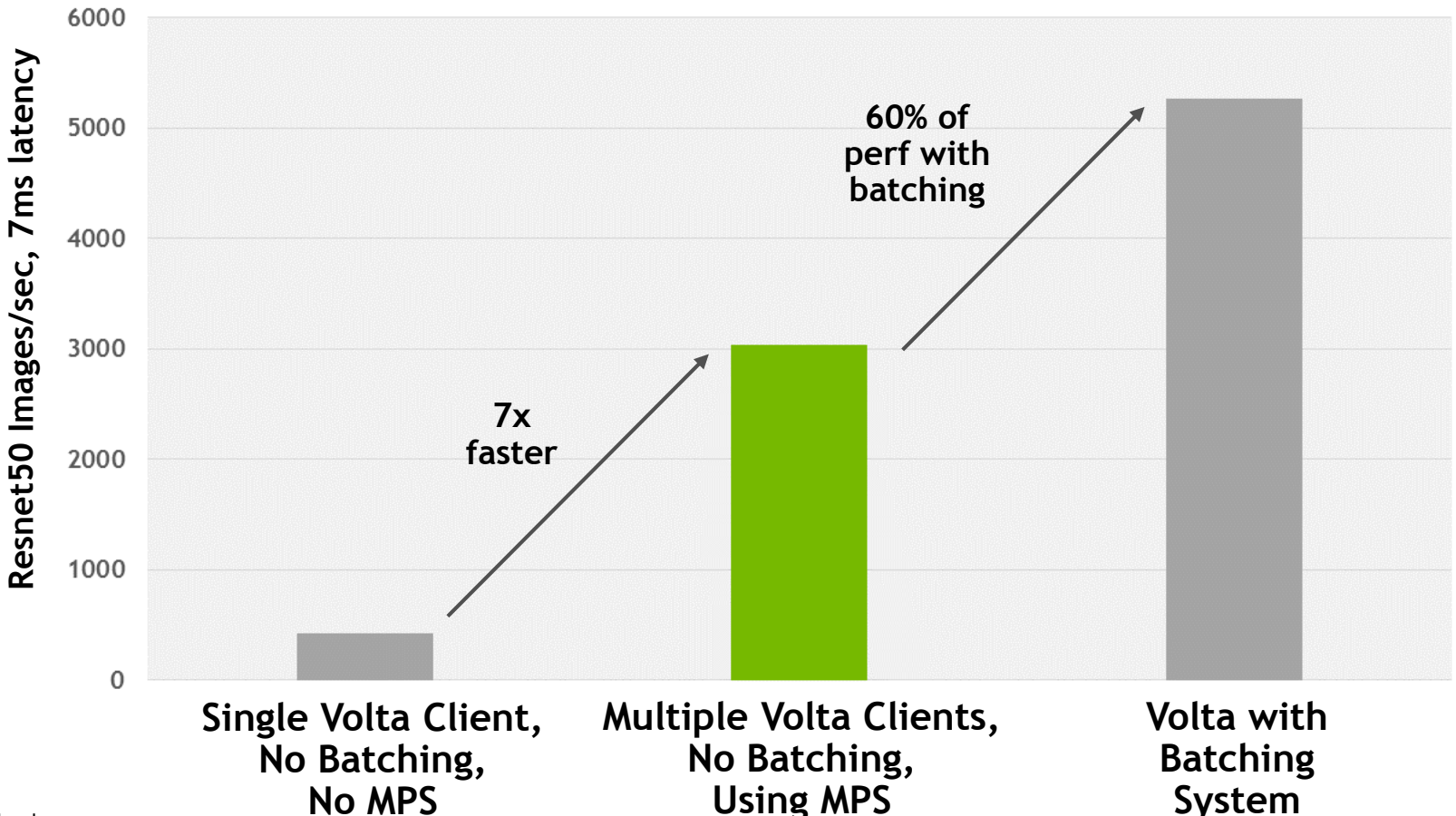
Hardware Accelerated Work Submission

Hardware Isolation



VOLTA MPS FOR INFERENCE

Efficient inference deployment without batching system



V100 measured on pre-production hardware.

NEW SM MICROARCHITECTURE

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

Completely new ISA

Twice the schedulers

Simplified Issue Logic

Large, fast L1 cache

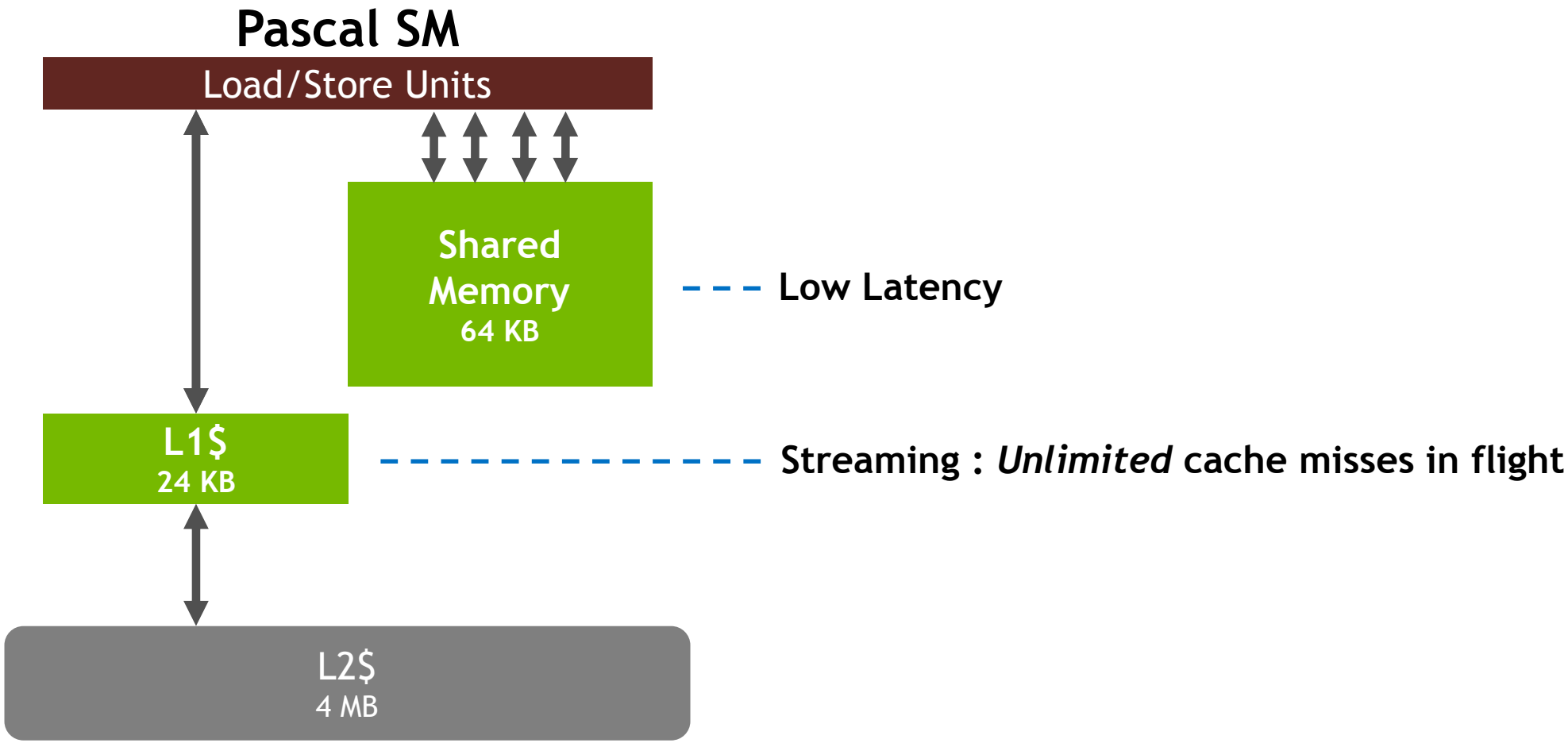
Improved SIMT model

Tensor acceleration

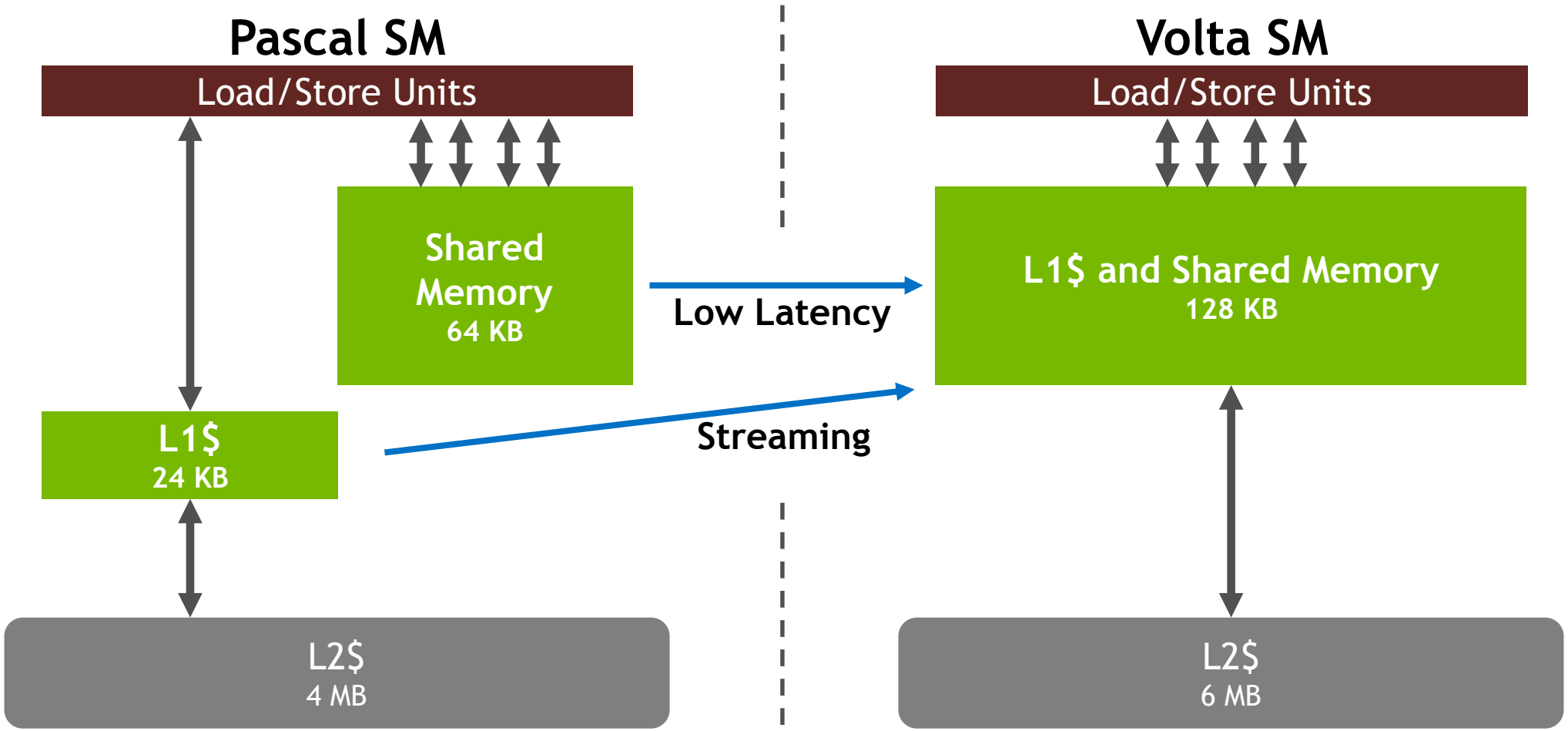
=

The easiest SM to program yet

RECAP: PASCAL L1 AND SHARED MEMORY



UNIFYING KEY TECHNOLOGIES



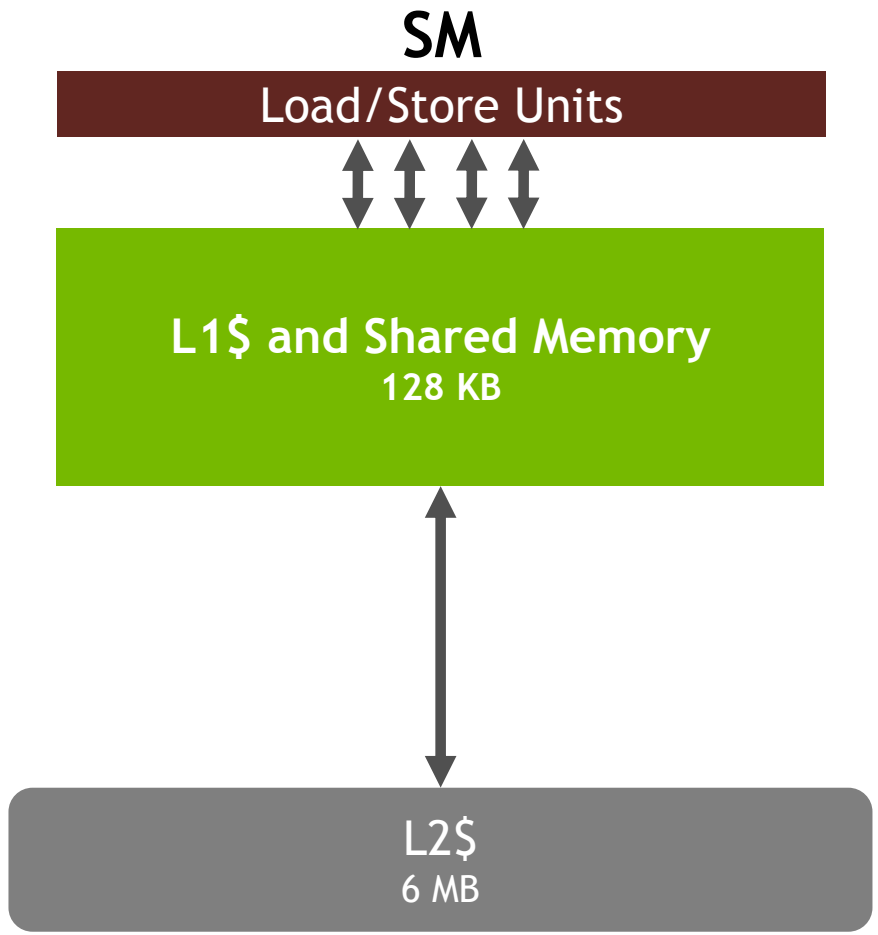
VOLTA L1 AND SHARED MEMORY

Volta Streaming L1$:

Unlimited cache misses in flight
Low cache hit latency
4x more bandwidth
5x more capacity

Volta Shared Memory :

Unified storage with L1
Configurable up to 96KB



NARROWING THE SHARED MEMORY GAP

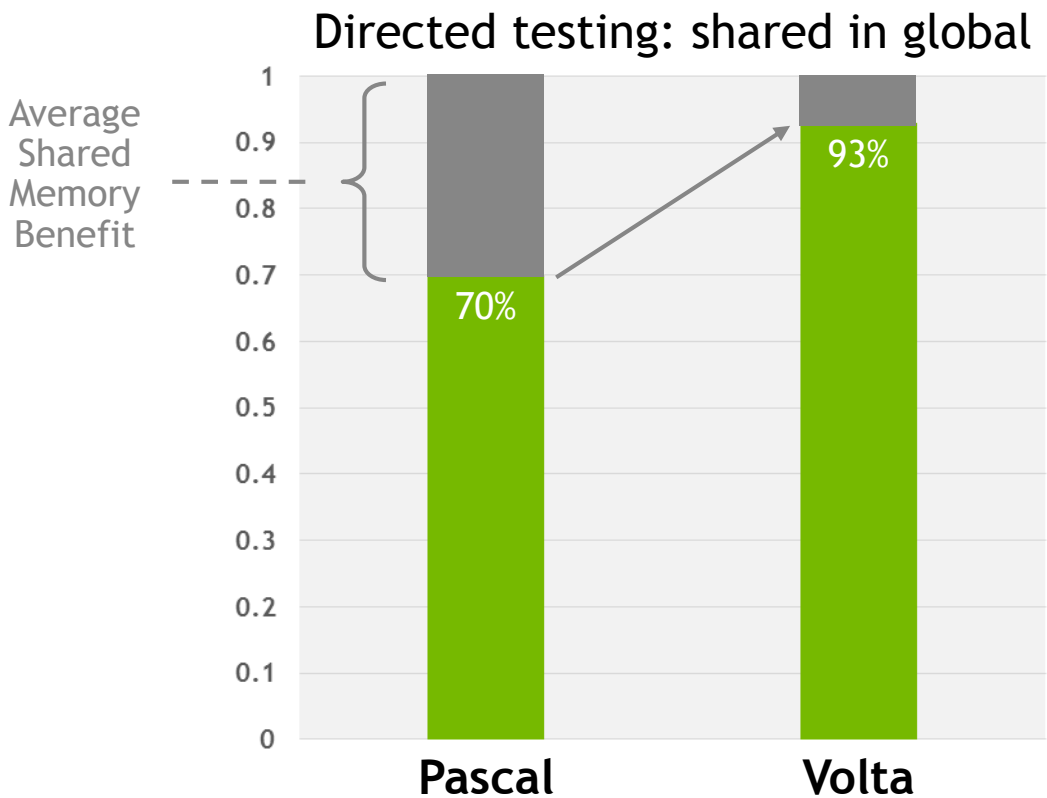
with the GV100 L1 cache

Cache: vs shared

- Easier to use
- 90%+ as good

Shared: vs cache

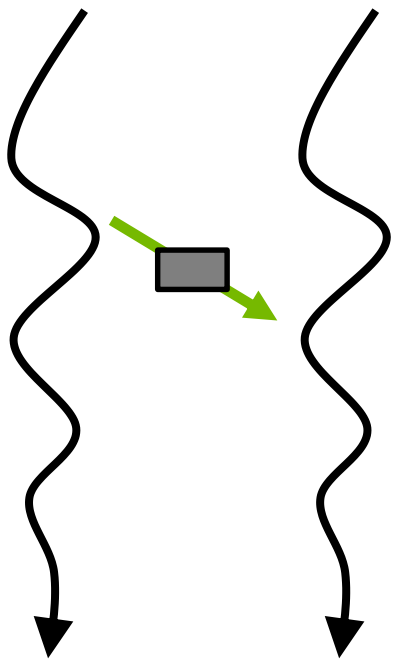
- Faster atomics
- More banks
- More predictable



INDEPENDENT THREAD SCHEDULING

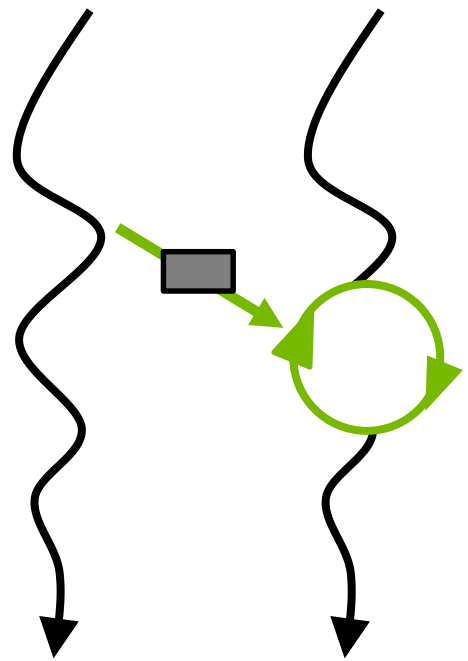
VOLTA: INDEPENDENT THREAD SCHEDULING

Communicating Algorithms



Pascal: Lock-Free Algorithms

Threads cannot wait for messages



Volta: Starvation Free Algorithms

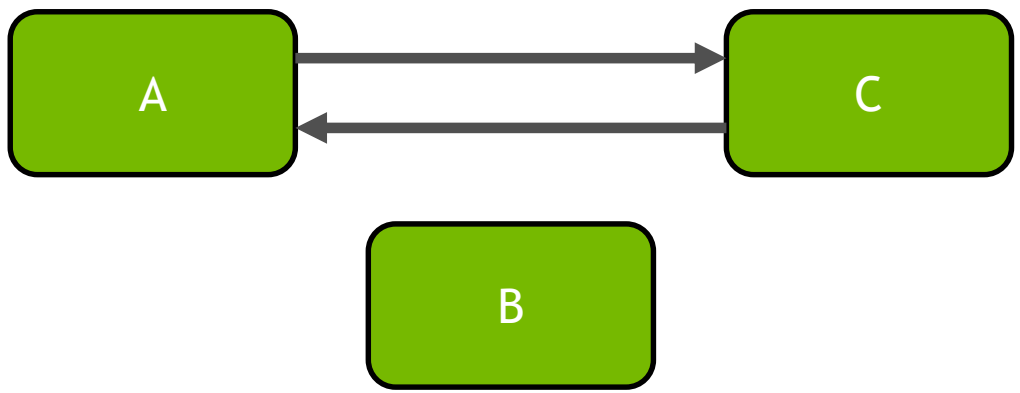
Threads may wait for messages

STARVATION FREE ALGORITHM

Example

```
__device__ void insert_after(Node *a, Node *b)
{
    Node *c;
    lock(a); lock(a->next);
    c = a->next;

    a->next = b;
    b->prev = a;

    b->next = c;
    c->prev = b;

    unlock(c); unlock(a);
}
```

Doubly-Linked List with Fine Grained Lock

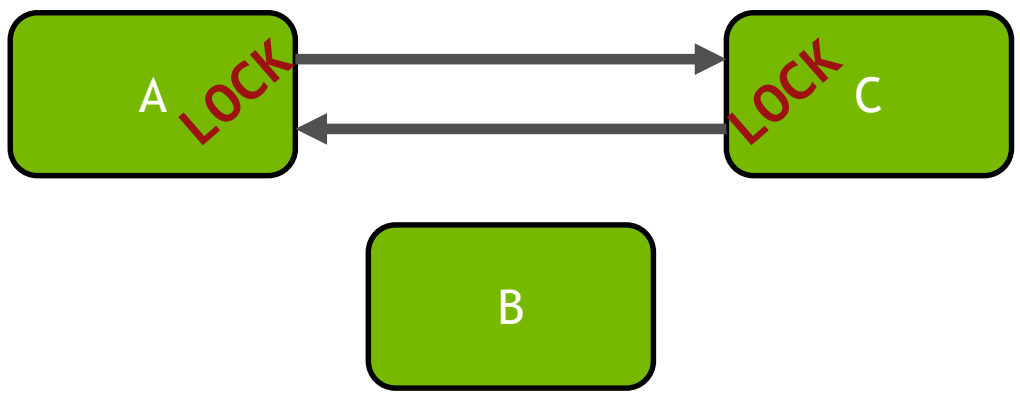


STARVATION FREE ALGORITHM

Example

```
__device__ void insert_after(Node *a, Node *b)
{
    Node *c;
    lock(a); lock(a->next);
    c = a->next;

    a->next = b;
    b->prev = a;

    b->next = c;
    c->prev = b;

    unlock(c); unlock(a);
}
```

Doubly-Linked List with Fine Grained Lock



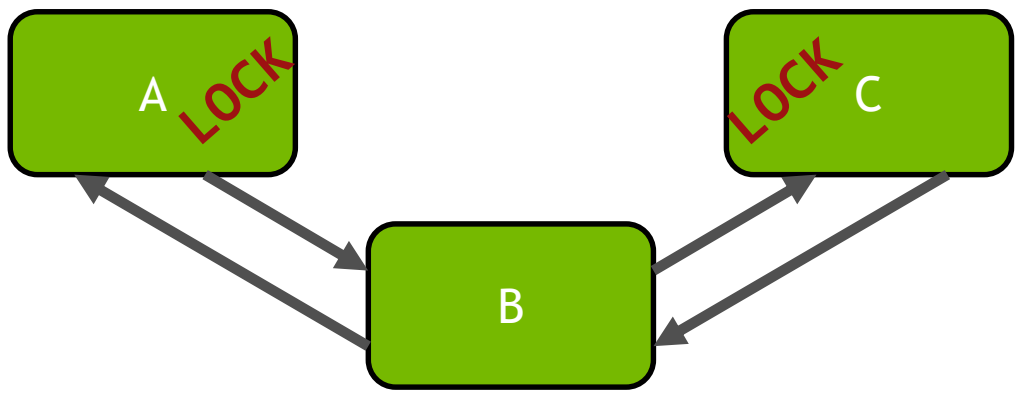
***Not shown: lock() implementation**

STARVATION FREE ALGORITHM

Example

```
__device__ void insert_after(Node *a, Node *b)
{
    Node *c;
    lock(a); lock(a->next);
    c = a->next;

    a->next = b;
    b->prev = a;

    b->next = c;
    c->prev = b;

    unlock(c); unlock(a);
}
```

Doubly-Linked List with Fine Grained Lock

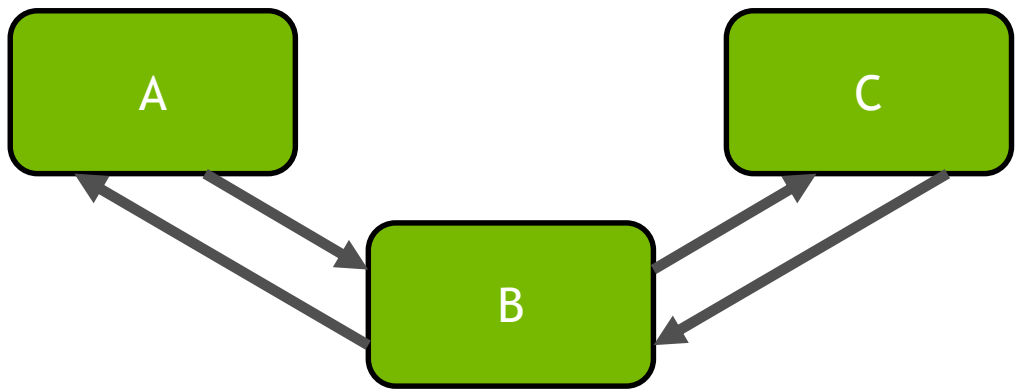


STARVATION FREE ALGORITHM

Example

```
__device__ void insert_after(Node *a, Node *b)
{
    Node *c;
    lock(a); lock(a->next);
    c = a->next;

    a->next = b;
    b->prev = a;

    b->next = c;
    c->prev = b;

    unlock(c); unlock(a);
}
```

Doubly-Linked List with Fine Grained Lock

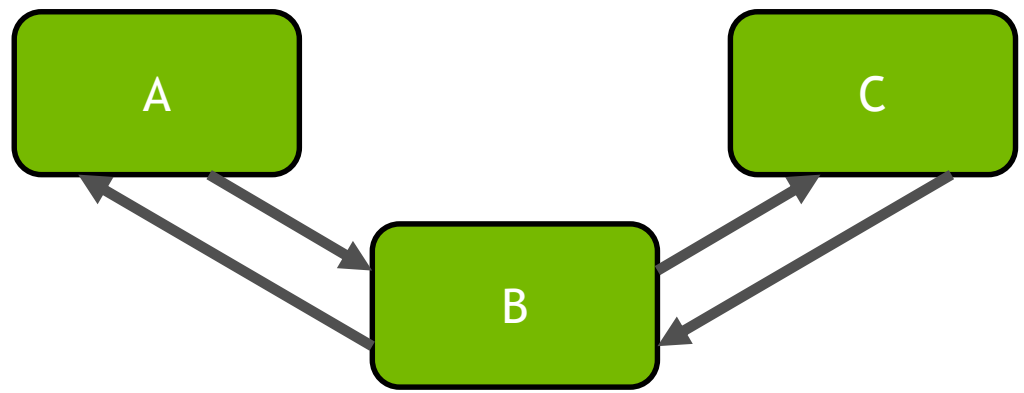


STARVATION FREE ALGORITHM

Example

```
__device__ void insert_after(Node *a, Node *b)
{
    Node *c;
    lock(a); lock(a->next);
    c = a->next;

    a->next = b;
    b->prev = a;

    b->next = c;
    c->prev = b;

    unlock(c); unlock(a);
}
```

Doubly-Linked List with Fine Grained Lock

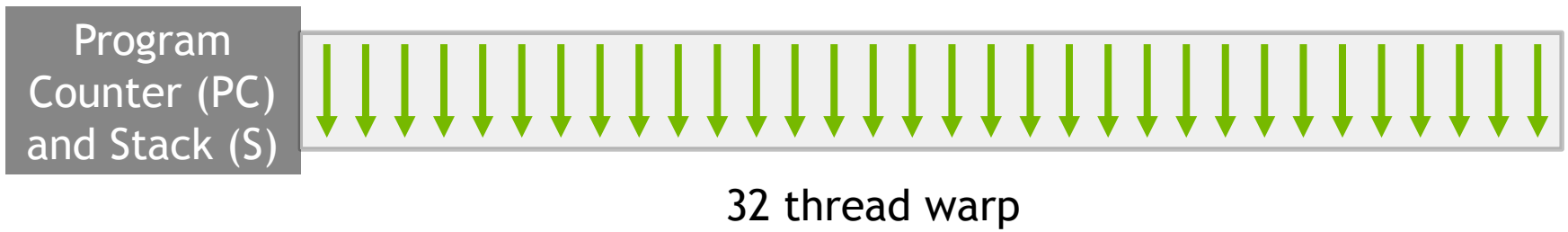


Tip! Volta can run 163,840 threads simultaneously

Minimize Contention!

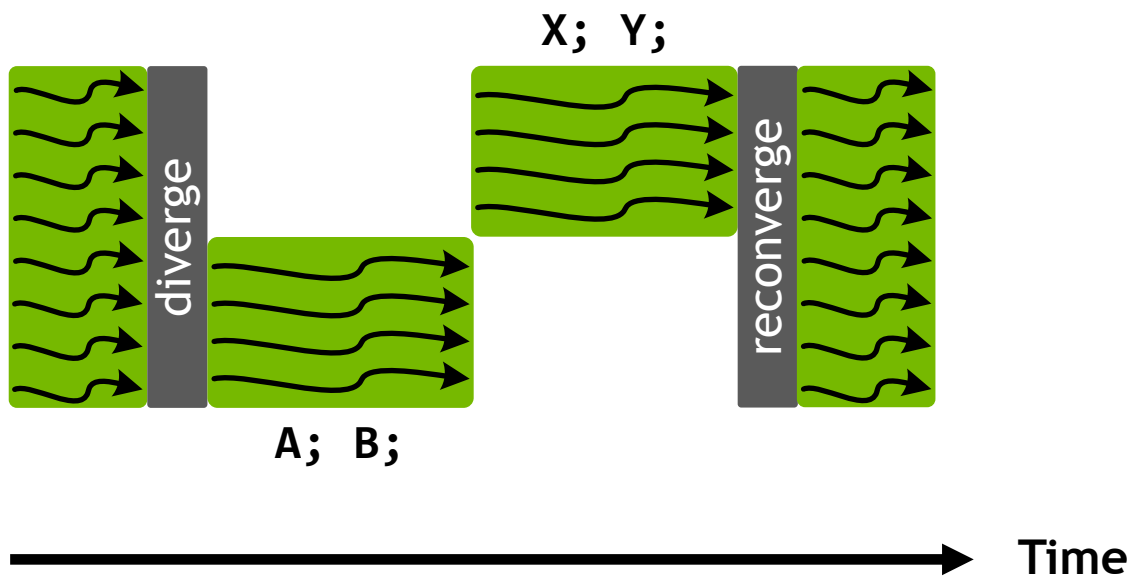
WARP IMPLEMENTATION

Pre-Volta



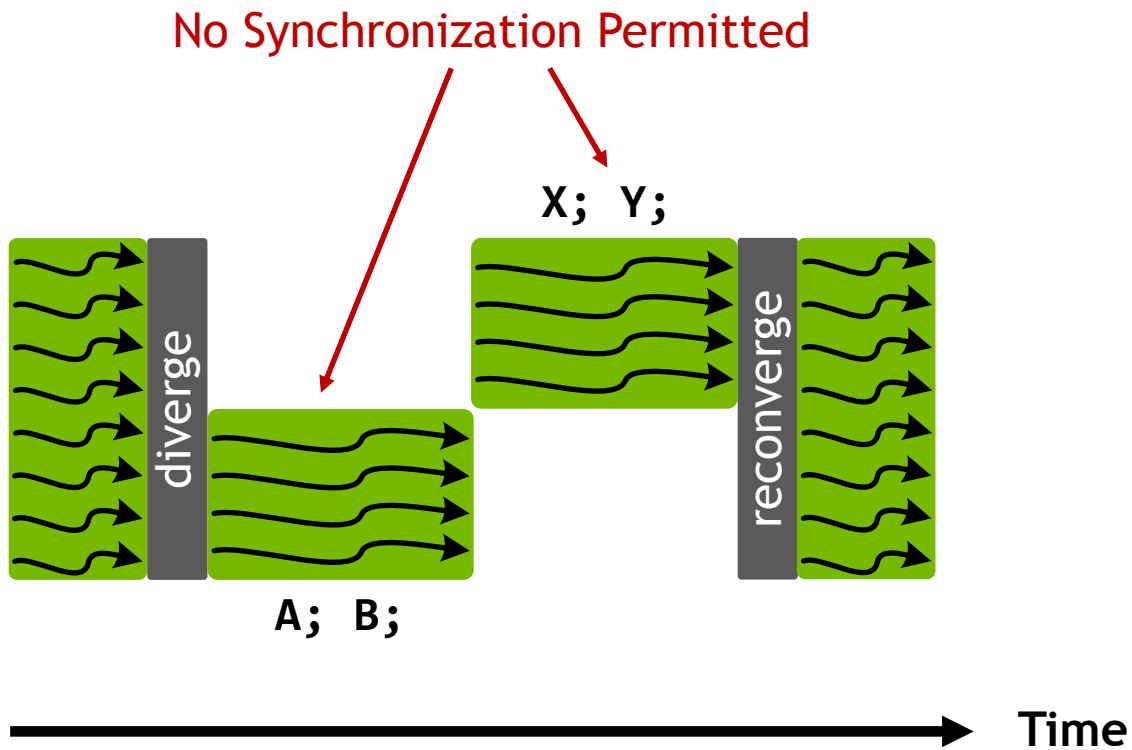
PASCAL WARP EXECUTION MODEL

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

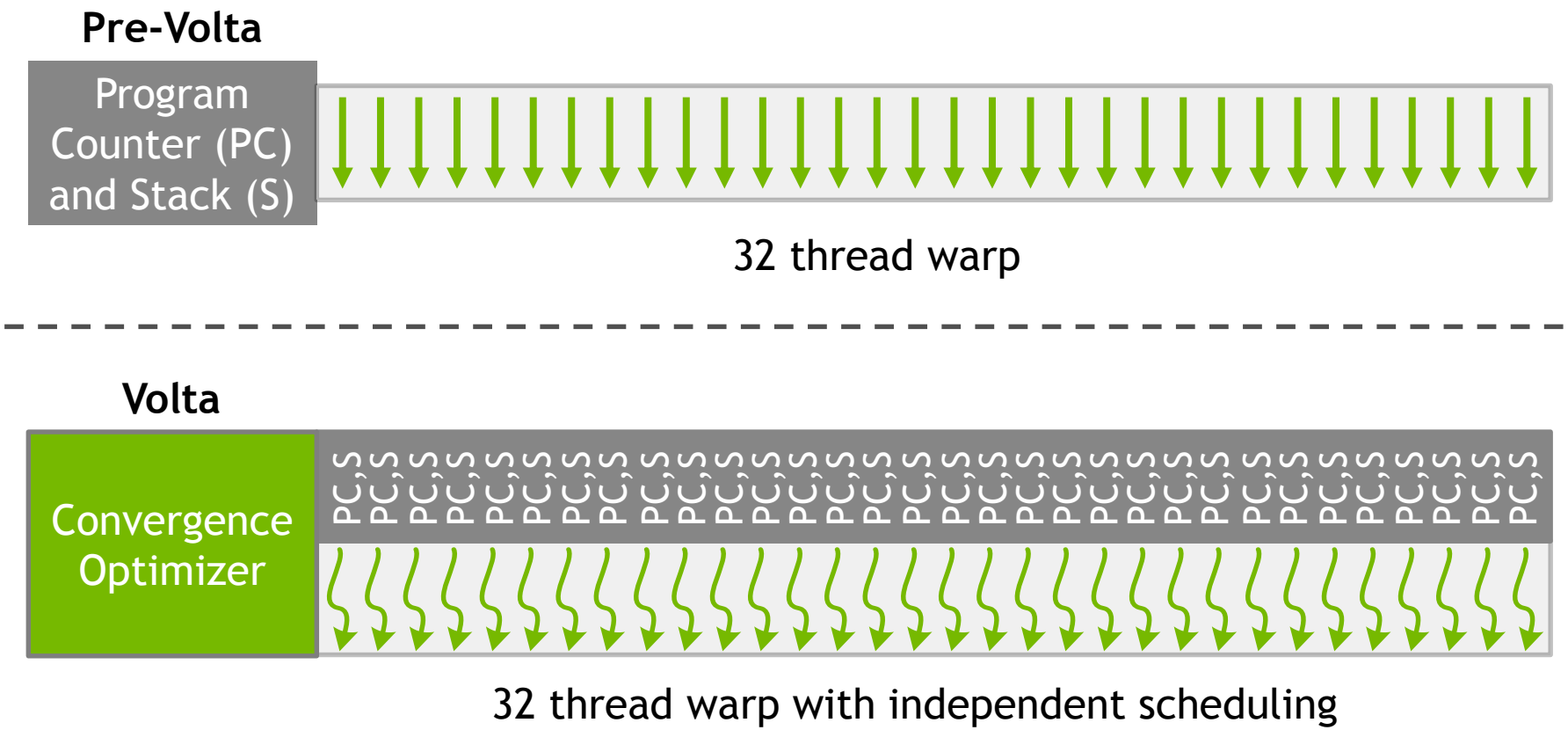


PASCAL WARP EXECUTION MODEL

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



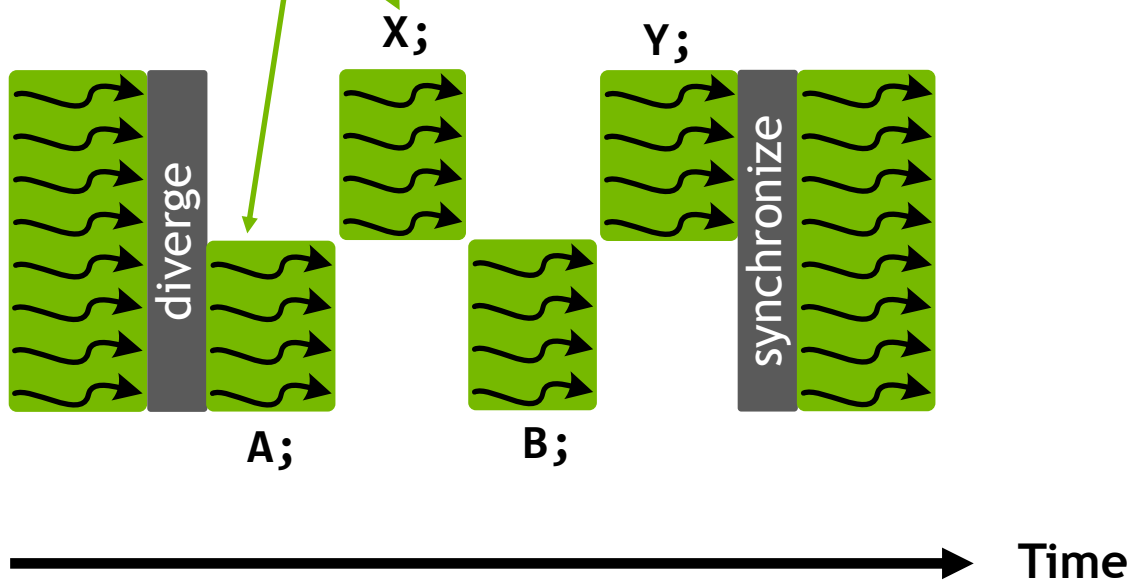
WARP IMPLEMENTATION



VOLTA WARP EXECUTION MODEL

Synchronization may lead to interleaved scheduling!

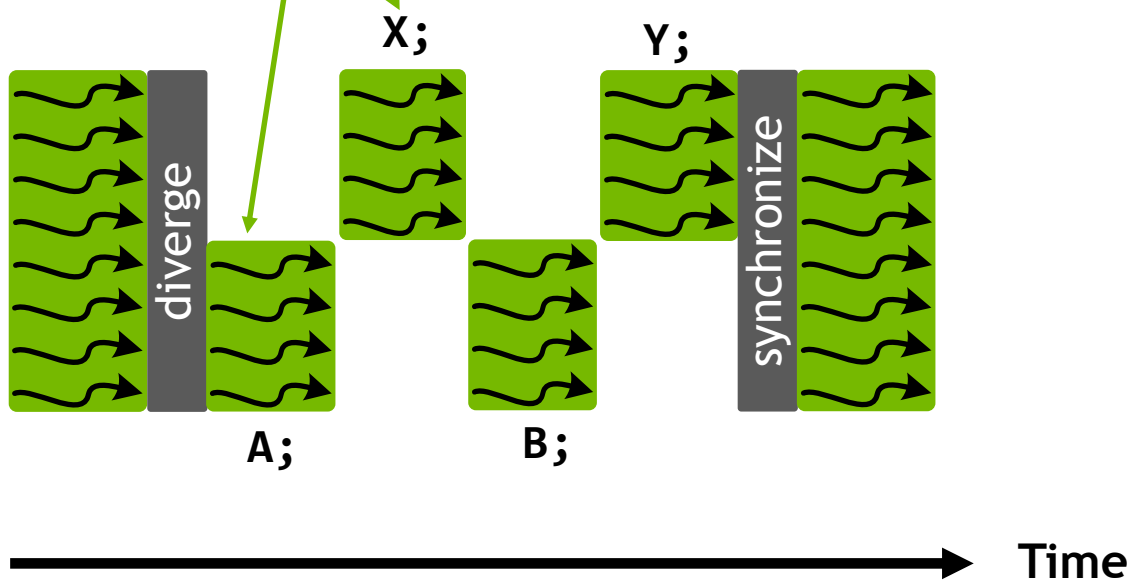
```
if (threadIdx.x < 4) {
    A;
    __syncwarp();
    B;
} else {
    X;
    __syncwarp();
    Y;
}
__syncwarp();
```



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'S EXTENDED SIMT MODEL

The SIMT model:

enable thread-parallel programs to execute with vector efficiency

	CPU	Pascal GPU	Volta GPU
Thread-parallelism	MIMD	SIMT (lock-free)	SIMT
Data-parallelism	SIMD	SIMT	SIMT

COOPERATION ➔ SYNCHRONIZATION

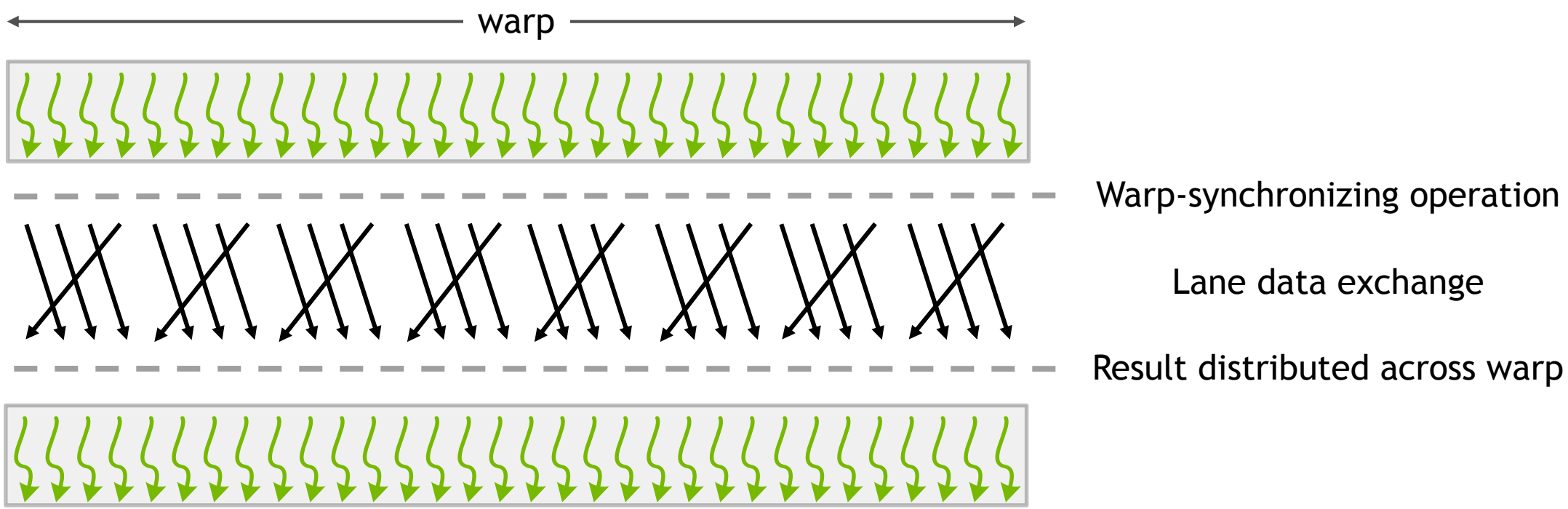
See also:

CUDA 9.0, S7132, Wednesday 230pm

Cooperative Groups, S7622, Wednesday 4pm

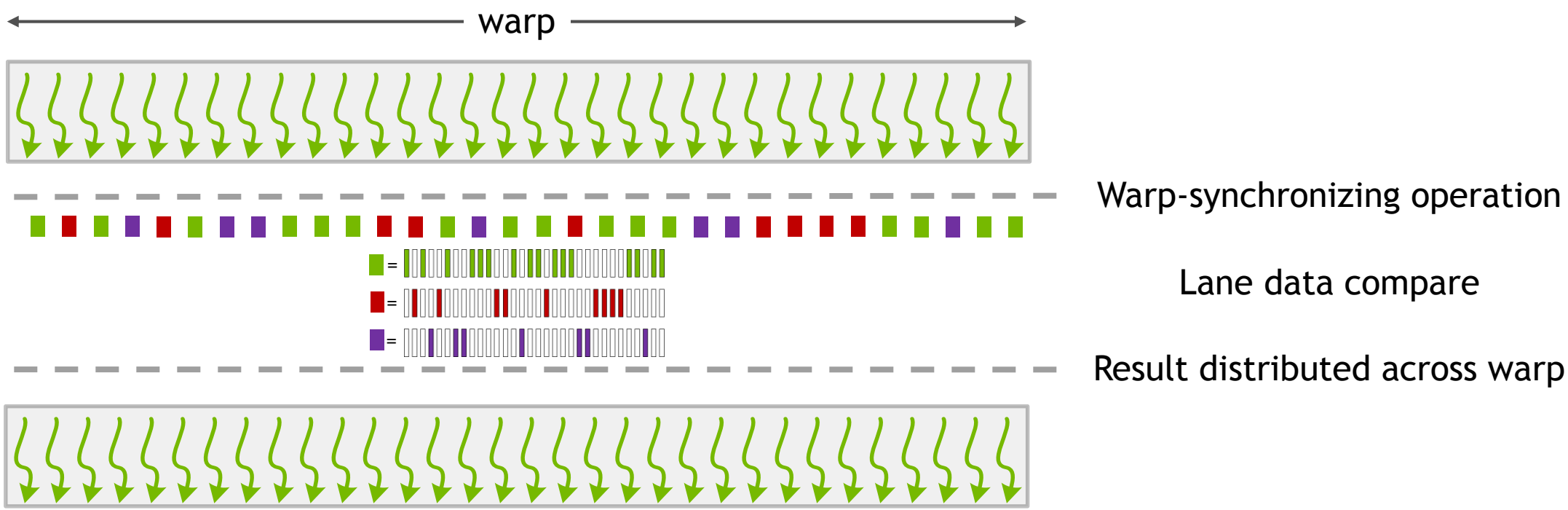
SHUFFLE SYNCHRONIZATION

__shfl_sync - deprecates __shfl

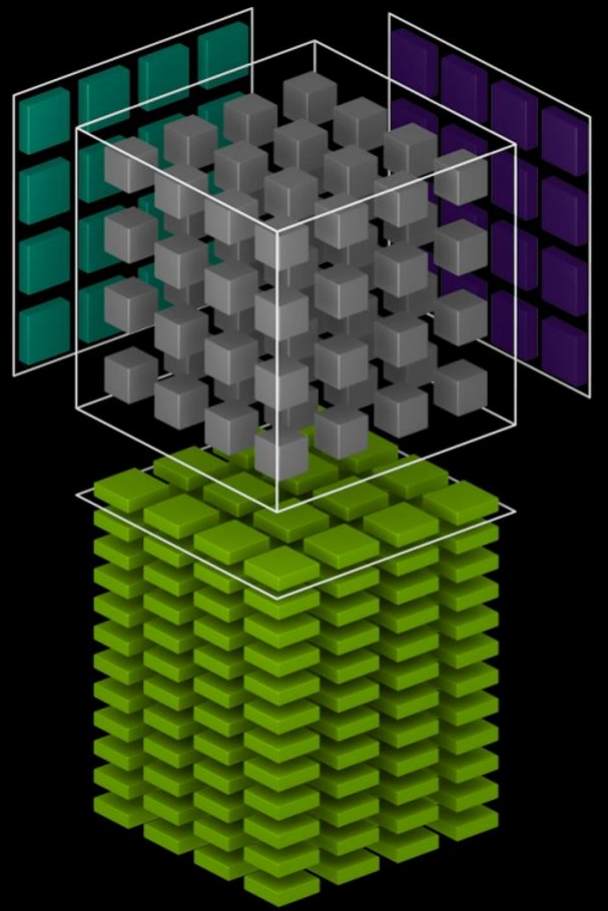


COMPARE SYNCHRONIZATION

__ballot_sync, __[any|all]_sync - deprecate namesakes
__match[any|all]_sync, __activemask - new

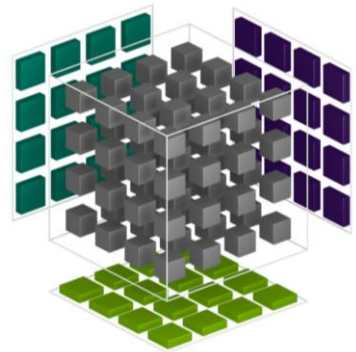


VOLTA TENSOR CORE



TENSOR CORE

Mixed Precision Matrix Math
4x4 matrices



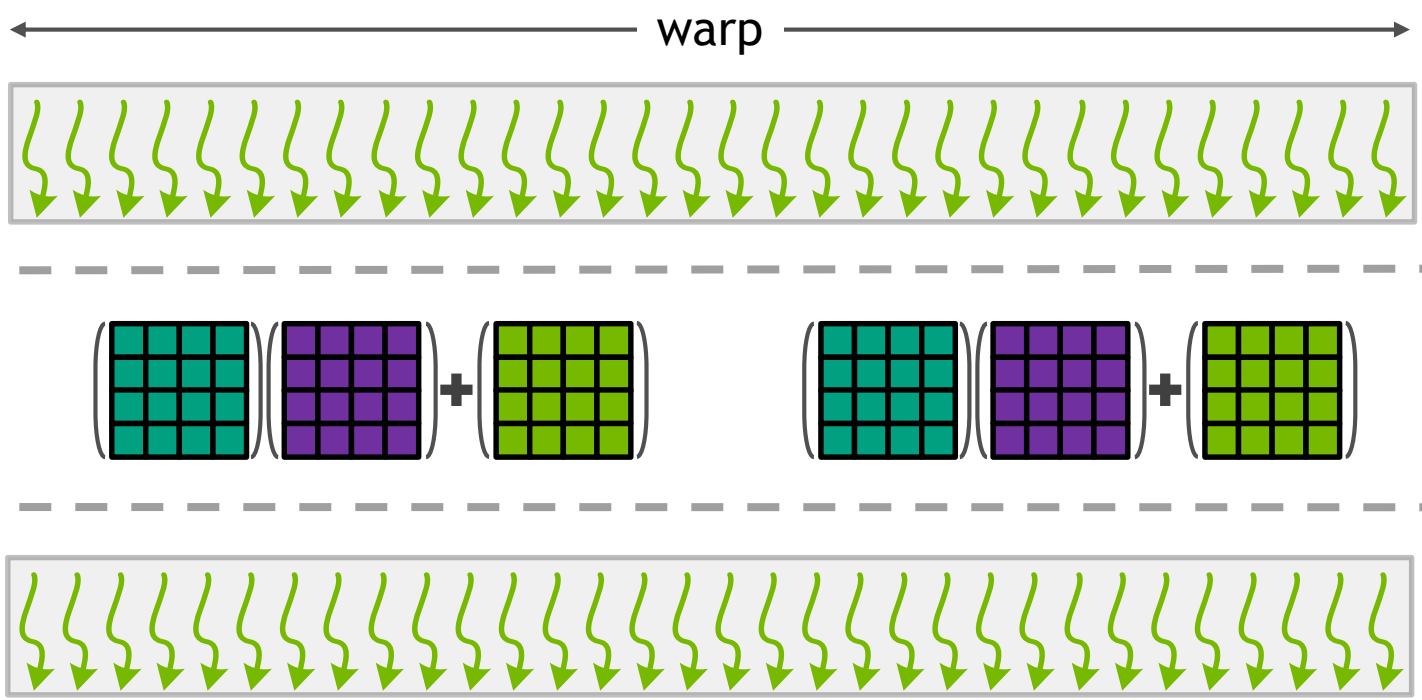
$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0}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

FP16 or FP32 | FP16 | FP16 | FP16 or FP32

$$D = AB + C$$

TENSOR SYNCHRONIZATION

Full Warp 16x16 Matrix Math

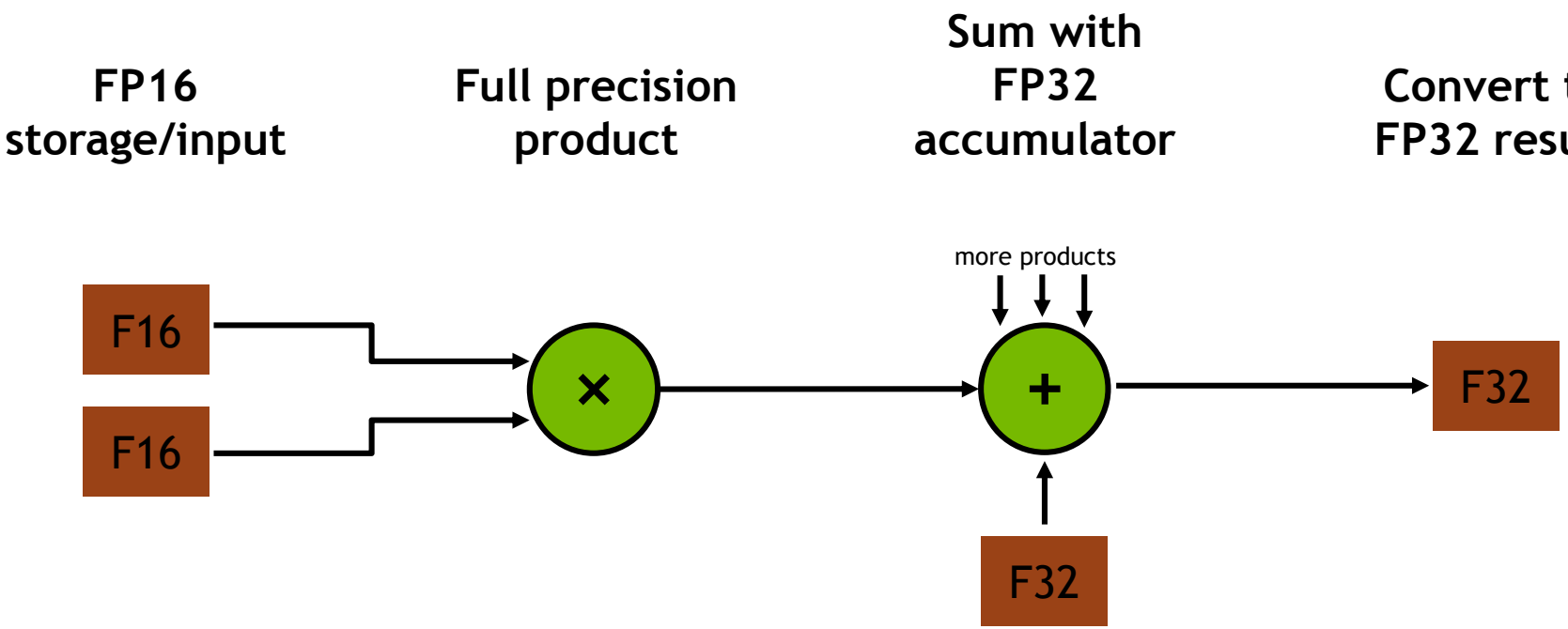


Warp-synchronizing operation

Composed Matrix Multiply and Accumulate for **16x16** matrices

Result distributed across warp

VOLTA TENSOR OPERATION



Also supports FP16 accumulator mode for inferencing

USING TENSOR CORES

NVIDIA cuDNN, cuBLAS, TensorRT

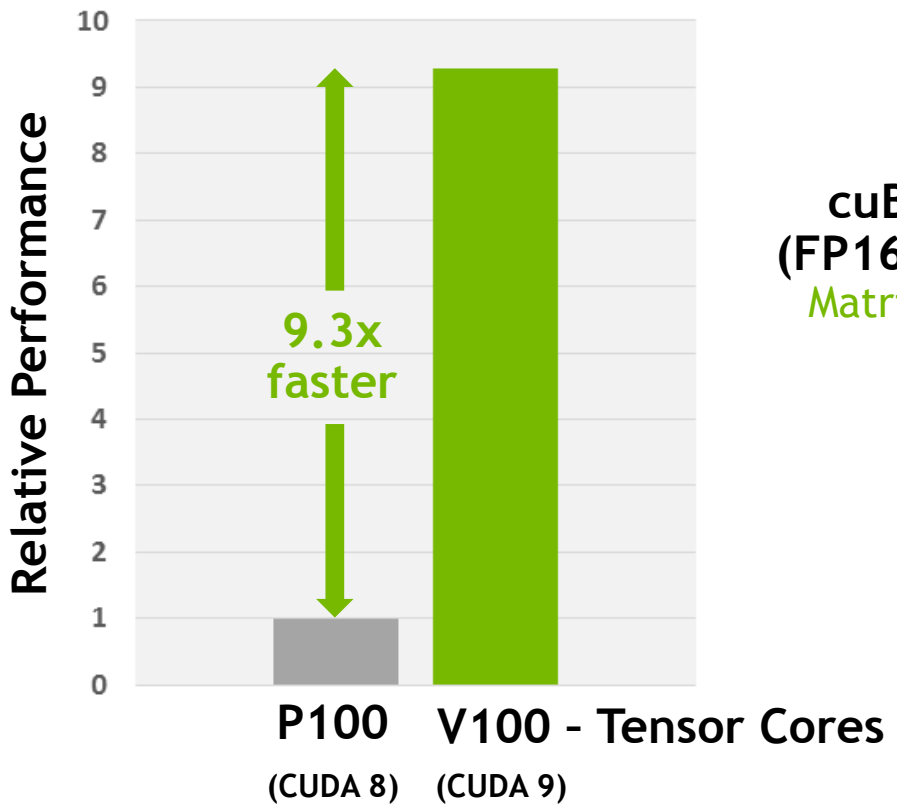
Volta Optimized Frameworks and Libraries

```
__device__ void tensor_op_16_16_16(
    float *d, half *a, half *b, float *c)
{
  wmma::fragment<matrix_a, …> Amat;
  wmma::fragment<matrix_b, …> Bmat;
  wmma::fragment<matrix_c, …> Cmat;

  wmma::load_matrix_sync(Amat, a, 16);
  wmma::load_matrix_sync(Bmat, b, 16);
  wmma::fill_fragment(Cmat, 0.0f);

  wmma::mma_sync(Cmat, Amat, Bmat, Cmat);

  wmma::store_matrix_sync(d, Cmat, 16,
                          wmma::row_major);
}
```

CUDA C++

Warp-Level Matrix Operations

A GIANT LEAP FOR DEEP LEARNING

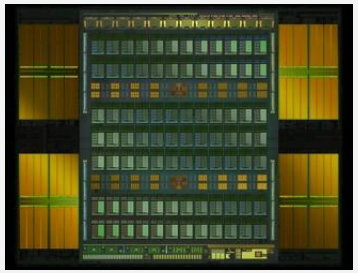


cuBLAS Mixed Precision
(FP16 input, FP32 compute)
Matrix Multiply (M=N=K=2048)

V100 measured on pre-production hardware.

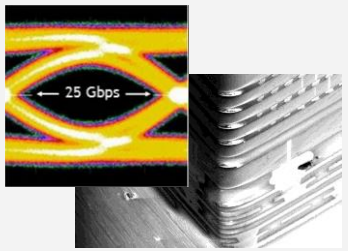
INTRODUCING TESLA V100

Volta Architecture



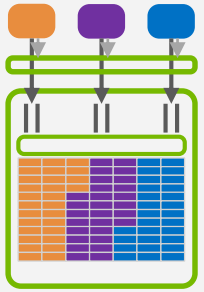
Most Productive GPU

Improved NVLink & HBM2



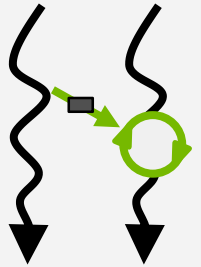
Efficient Bandwidth

Volta MPS



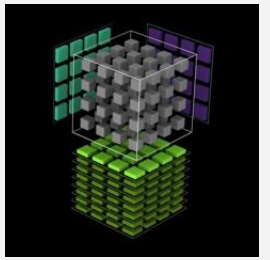
Inference Utilization

Improved SIMT Model



New Algorithms

Tensor Core



120 Programmable TFLOPS Deep Learning

More V100 Features: 2x L2 atomics, int8, new memory model, copy engine page migration, and more …

The Fastest and Most Productive GPU for Deep Learning and HPC