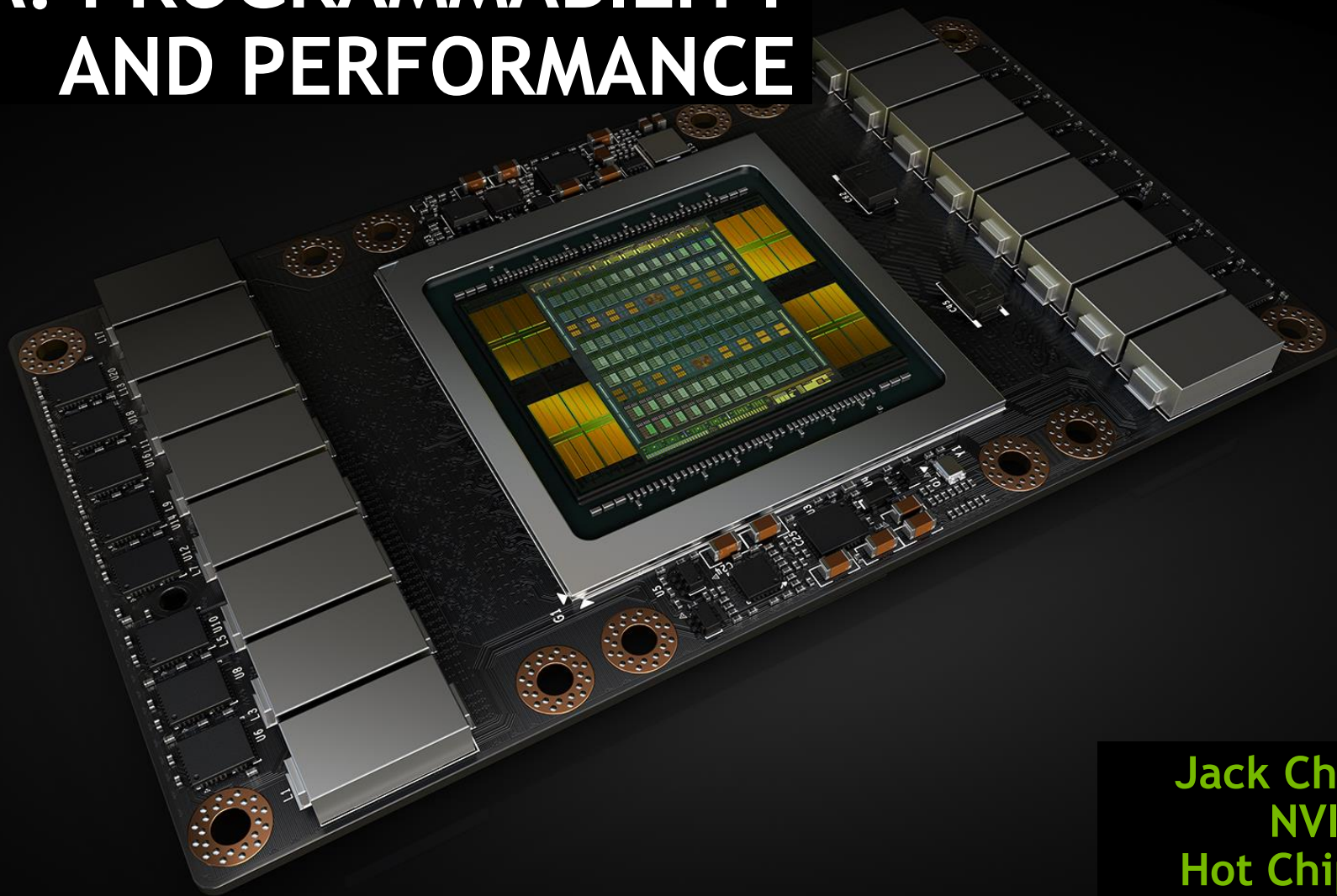


VOLTA: PROGRAMMABILITY AND PERFORMANCE



Jack Choquette
NVIDIA
Hot Chips 2017

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

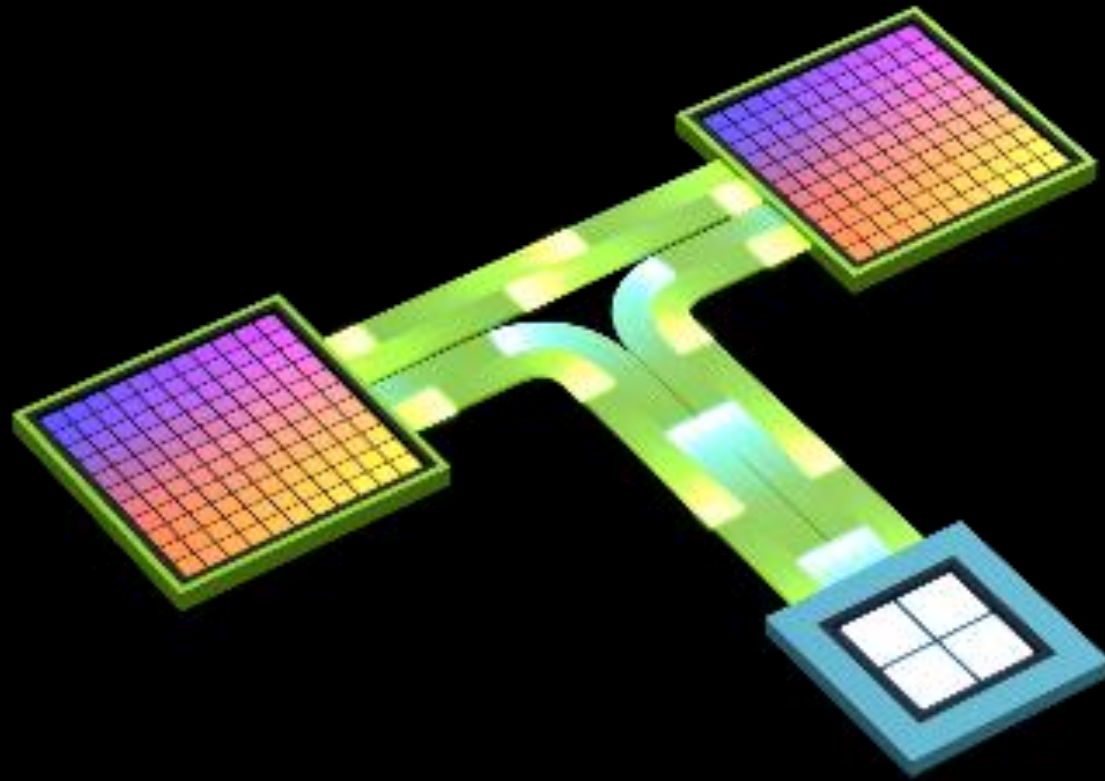


*full GV100 chip contains 84 SMs

GPU PERFORMANCE COMPARISON

	P100	V100	Ratio
DL Training	10 TFLOPS	120 TFLOPS	12x
DL Inferencing	21 TFLOPS	120 TFLOPS	6x
FP64/FP32	5/10 TFLOPS	7.5/15 TFLOPS	1.5x
HBM2 Bandwidth	720 GB/s	900 GB/s	1.2x
STREAM Triad Perf	557 GB/s	855 GB/s	1.5x
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

NVLINK



NVLINK - PROGRAMMABILITY

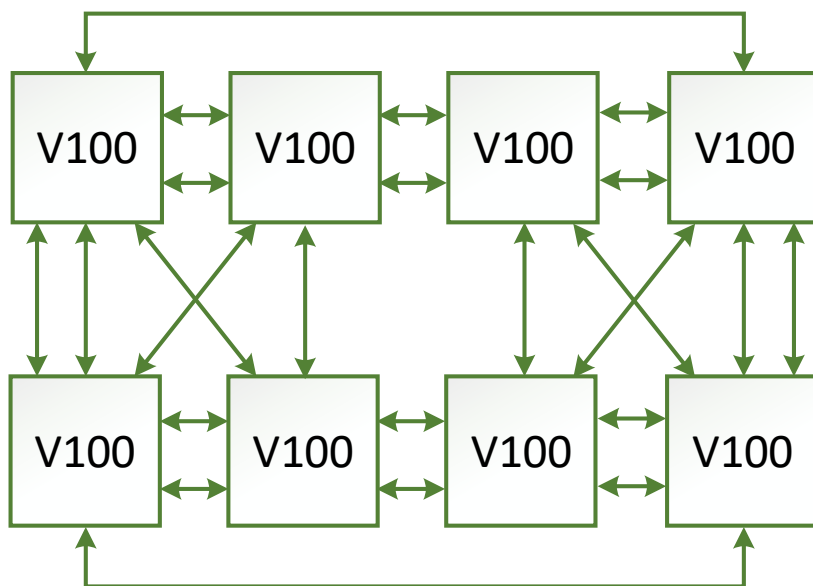
CPU Mastering	Direct Load/Store access for all agents Flat shared address space
GPU & CPU atomics	Atomic completion at destination Read-Modify-Write primitive for unsupported atomics
Address Translation Services	GPU uses CPU page tables directly

NVLINK - PERFORMANCE AND POWER

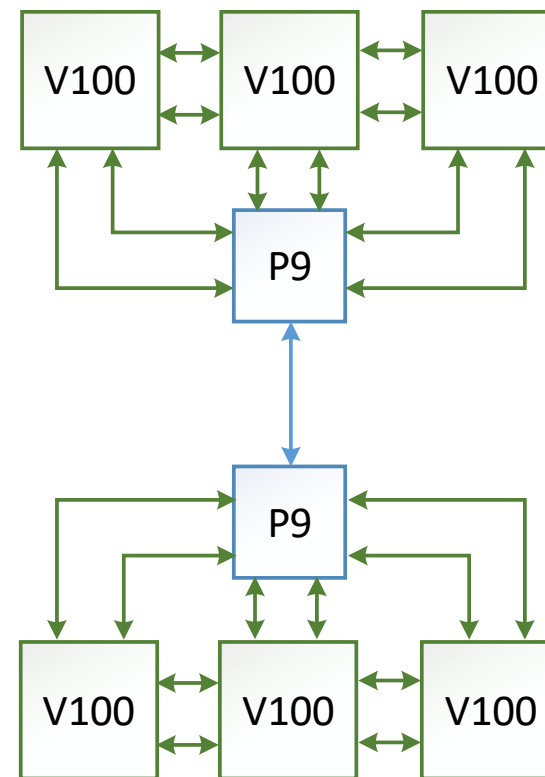
Bandwidth	25Gbps signaling 6 NVLinks for GV100 1.9 x Bandwidth improvement over GP100
Coherence	Latency sensitive CPU caches GMEM Fast access in local cache hierarchy Probe filter in GPU
Power Savings	Reduce number of active lanes for lightly loaded link

NVLINK NODES

DL - HYBRID CUBE MESH - DGX-1 w/ Volta



HPC - P9 CORAL NODE - SUMMIT

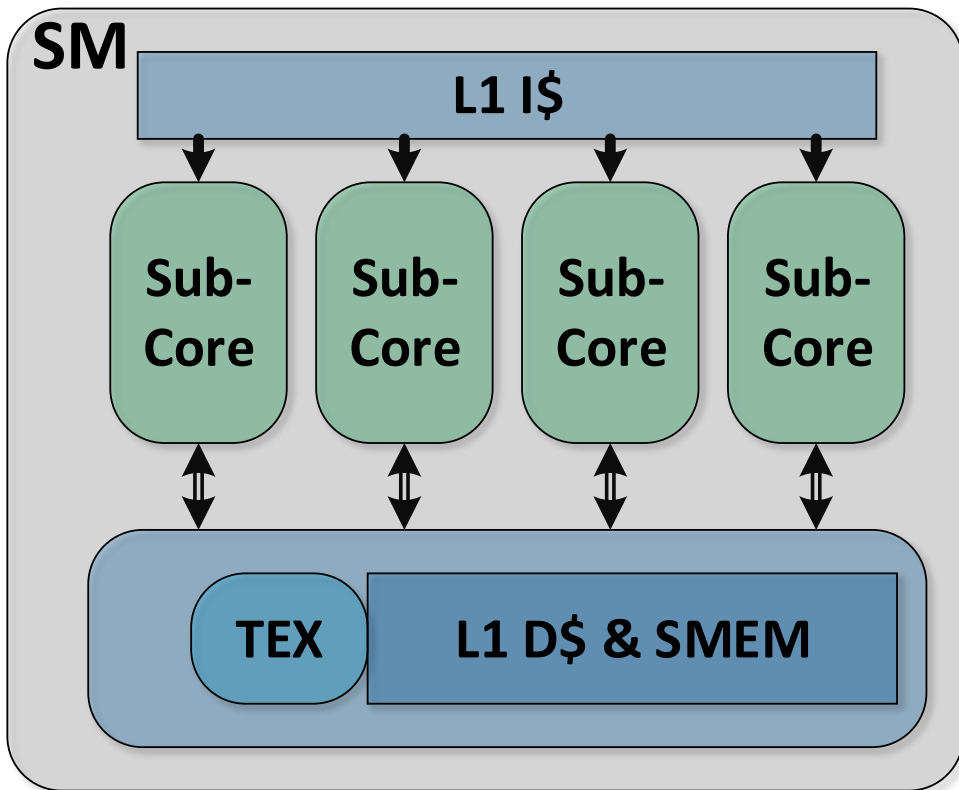


SM CORE



VOLTA GV100 SM

Redesigned for Productivity and Accessible Performance



Twice the schedulers

Simplified Issue Logic

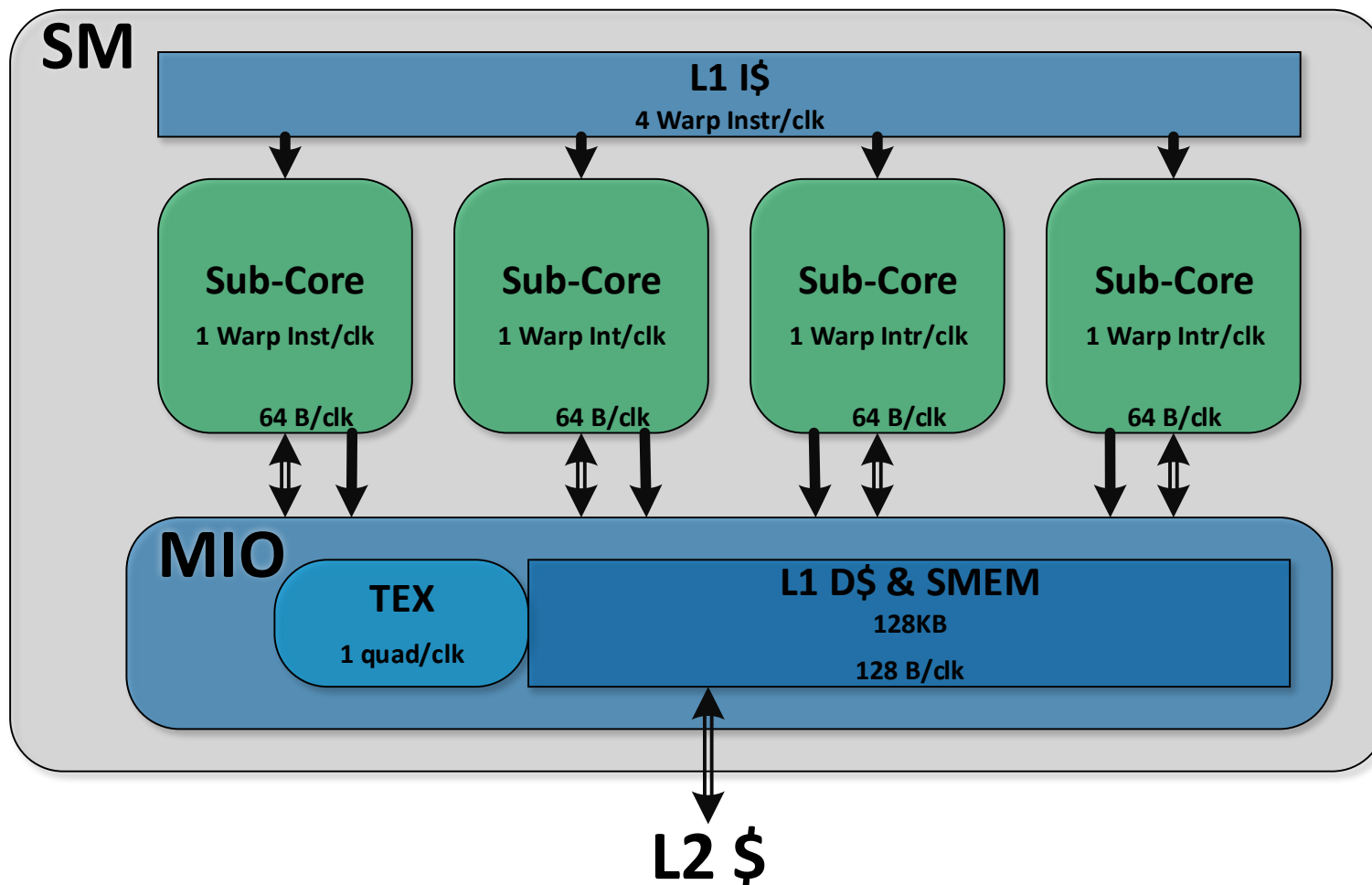
Large, fast L1 cache

Improved SMT model

Tensor acceleration

+50% energy efficiency vs GP100 SM

SM MICROARCHITECTURE

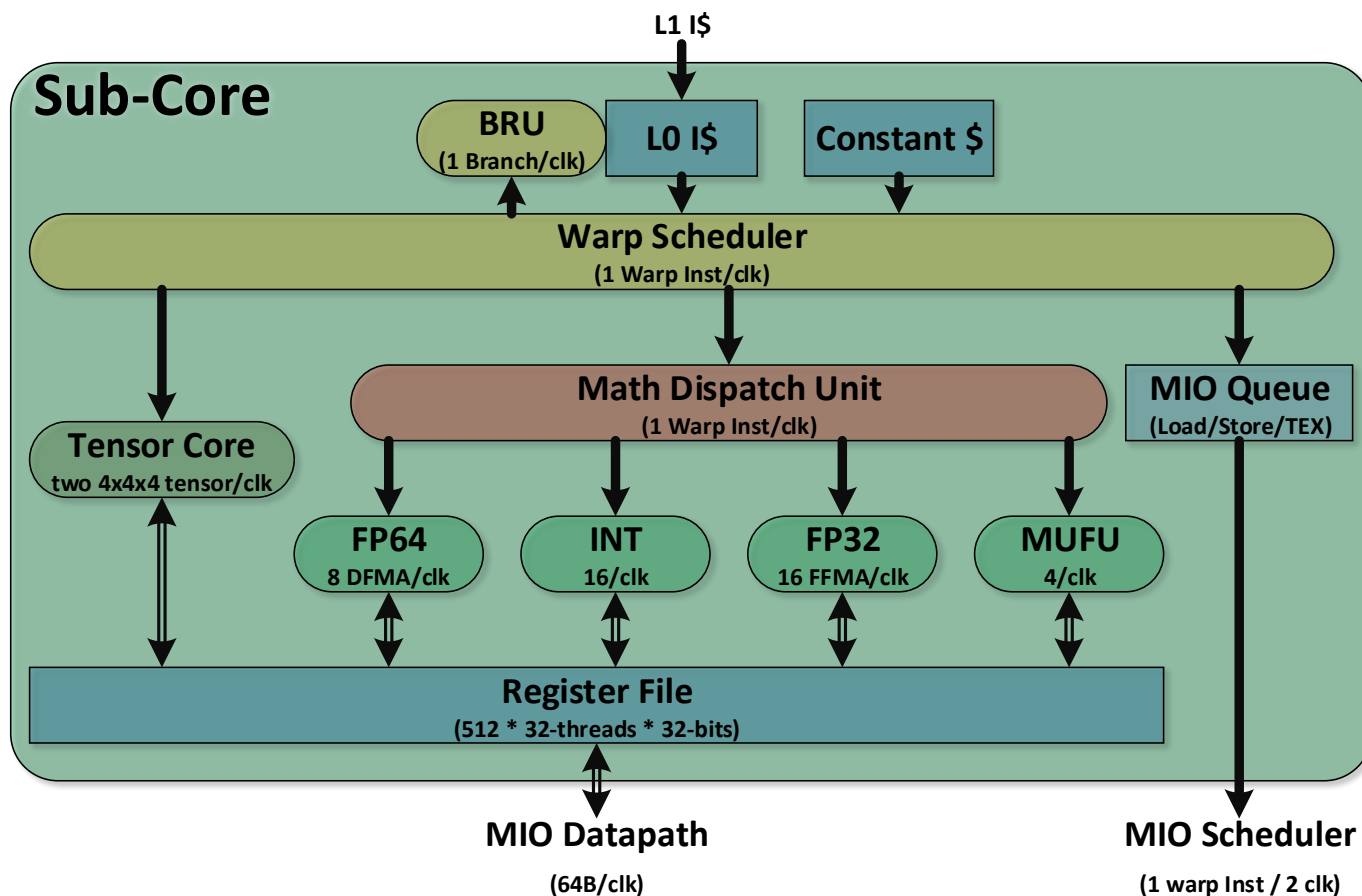


Shared L1 I\$

4 Independently
Scheduled Sub-Cores

Shared MIO
(TEX/L1\$/SMEM)

SUB-CORE



Warp Scheduler

- 1 Warp instr/clock
- L0 I\$, branch unit

Math Dispatch Unit

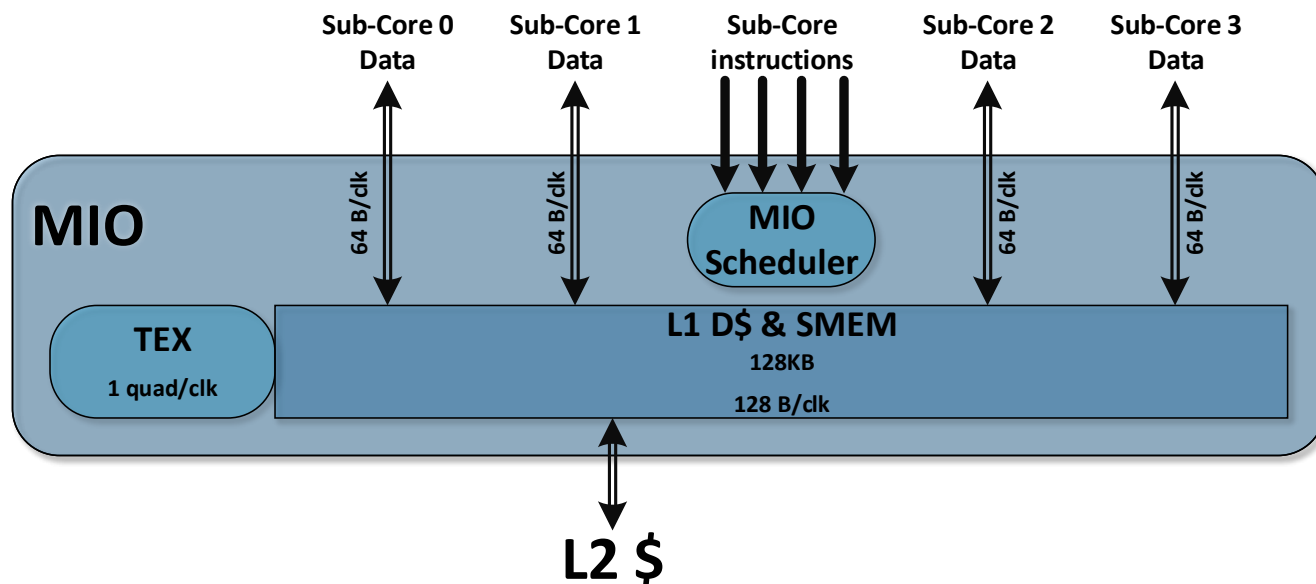
- Keeps 2+ Datapaths Busy

MIO Instruction Queue

- Hold for Later Scheduling

Two 4x4x4 Tensor Cores

L1 AND SHARED MEMORY



Streaming L1\$

- Unlimited cache misses in flight
- Low cache hit latency
- 4x bandwidth vs GP100
- 4x capacity vs GP100

Shared Memory

- Unified Storage with L1\$
- Configurable up to 96KB

NARROWING THE SHARED MEMORY GAP

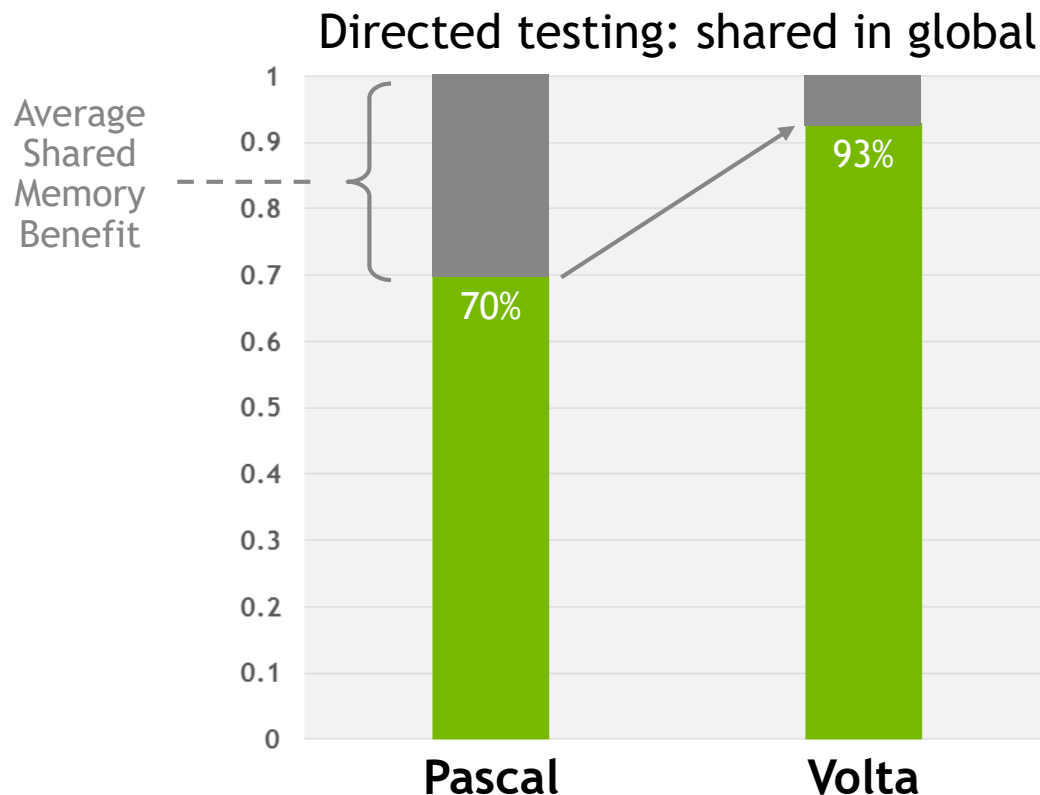
with the GV100 L1 cache

Cache: vs shared

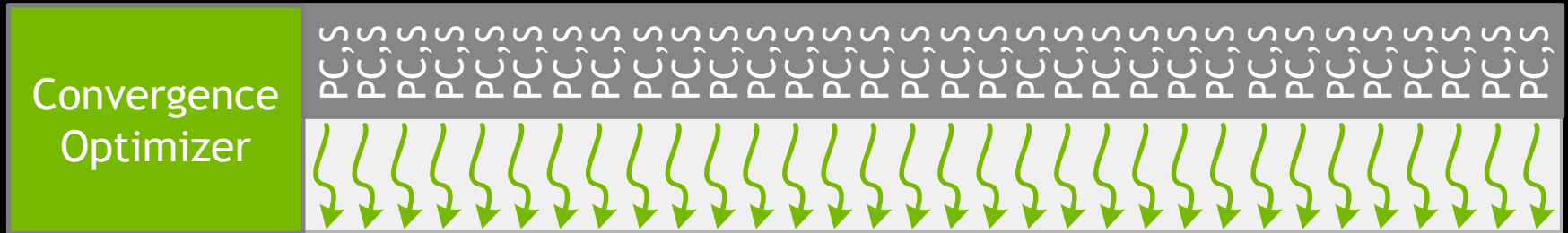
- Easier to use
- 90%+ as good

Shared: vs cache

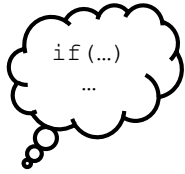
- Faster atomics
- More banks
- More predictable



INDEPENDENT THREAD SCHEDULING



PROGRAMMING MODEL: MEMORY ADDRESSES



Programs written
using annotations



Programs written
as normal

causes

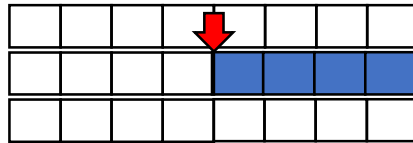
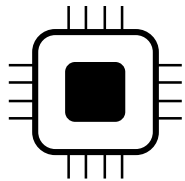
\$ (CC)

Convergence must
be proven



Convergence is
perf optimization

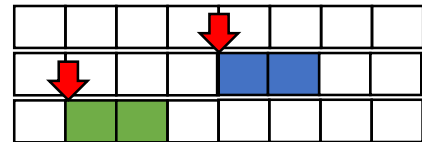
causes



Convergence is
prescribed

(e.g. Vector)

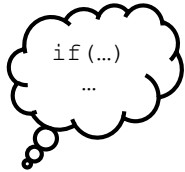
GPU Innovation



Convergence is
an optimization

(e.g. NVIDIA GPUs)

PROGRAMMING MODEL: CONTROL ADDRESSES



Programs written
without synchronization



Programs written
as normal

causes

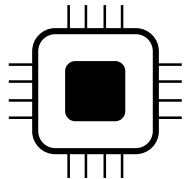
\$ (CC)

Convergence must
be proven



Convergence is
perf optimization

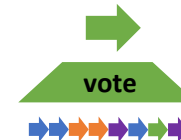
causes



Convergence is
prescribed

(e.g. Scalar + Predicated Vector)
(e.g. pre-Volta GPUs)

Volta Innovation

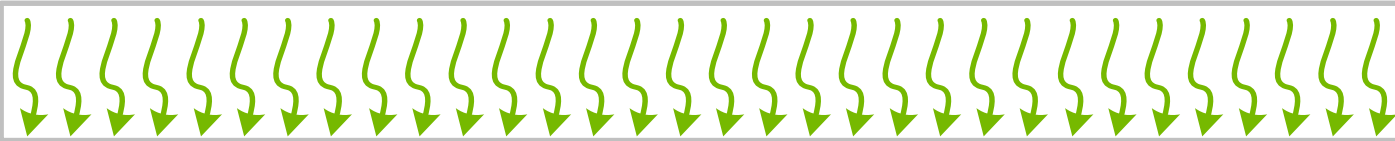


Convergence is
an optimization

**(e.g. Volta's Independent
Thread Scheduling)**

NVIDIA SIMT GPUS: SCALAR THREADS ON SIMD UNITS

C, C++, ..., FORTRAN



→ Array of scalar threads

Scalar Compiler

→ 50+ year history of optimizations

Scalar ISA

→ NVIDIA GPUs have always used

Thread Virtualization Tech

→ 30+ year history of TLP-to-ILP conversion

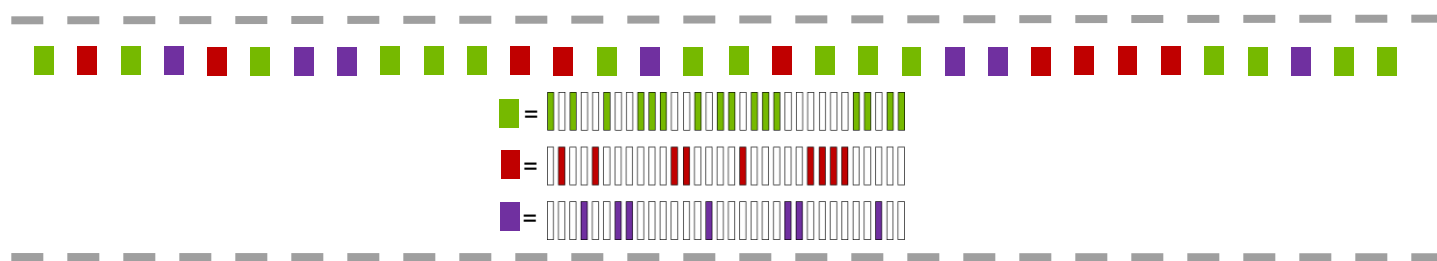
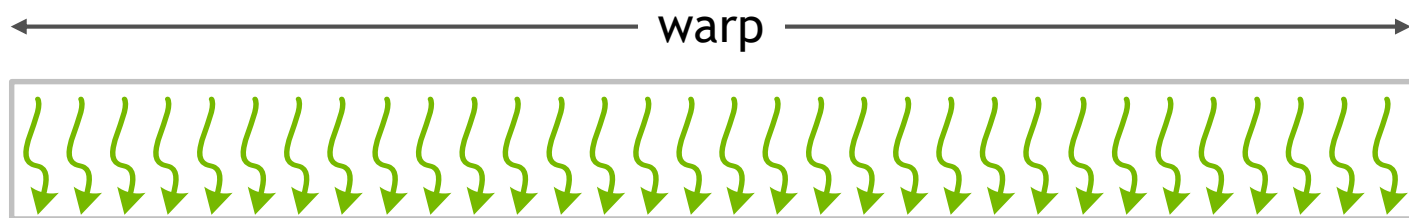
SIMD Units

→ Where the efficiency comes from

*Volta expands
TLP exploitation!*

COOPERATION → SYNCHRONIZATION

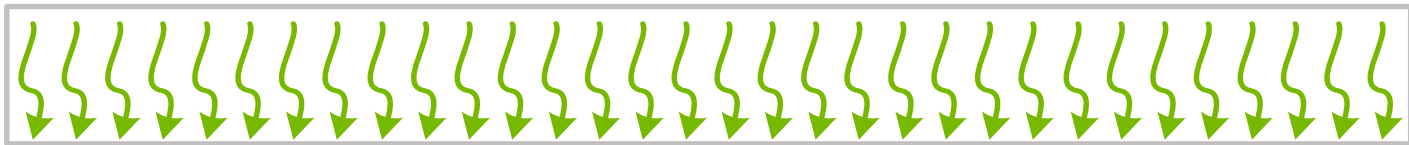
Example: `match.any.sync`



Warp-synchronizing operation

e.g. our *NEW* lane data compare

Result distributed across warp



EFFICIENT GPU THREAD SYNCHRONIZATION

Critical Section Performance

(relative to single thread)

10's

CPU Threads

10K's

GPU Threads

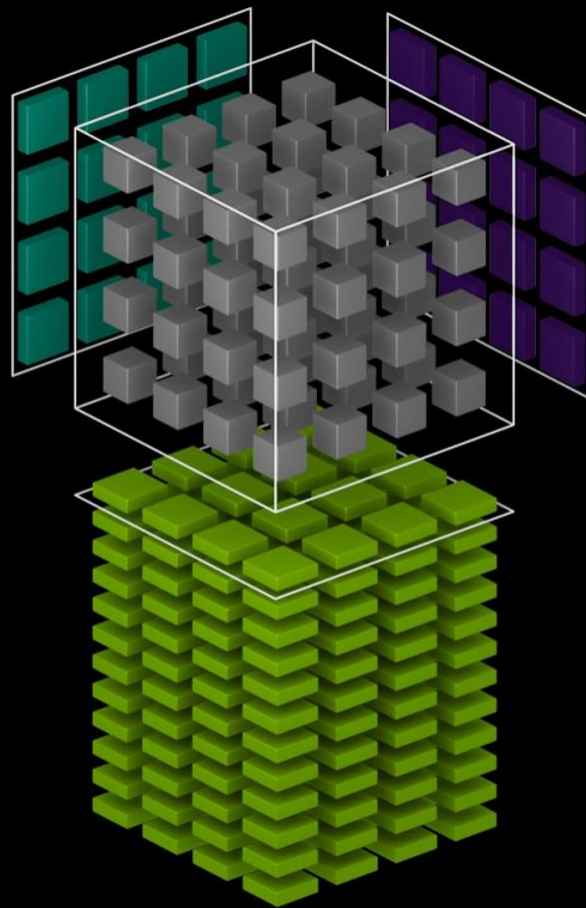
```
struct mutex {  
  
    void lock() {  
        while (1) {  
            bool state = fa  
            if (locked.comp  
                std::memo  
            return;  
            while (locked.load(std::memory_order_relaxed))  
                do_exponential_backoff(); // for brevity  
        }  
    }  
  
    void unlock() {  
        locked.store(false, std::memory_order_release);  
    }  
  
    atomic<bool> locked = ATOMIC_VAR_INIT(false);  
};
```

Note: Unfair spinlock with exponential back-off algorithm,
x86-64 Linux 4.8.0, i7-5820K, CUDA 9.0 pre-release software,
V100 pre-production hardware, Aug 2017

Performance (ratio)

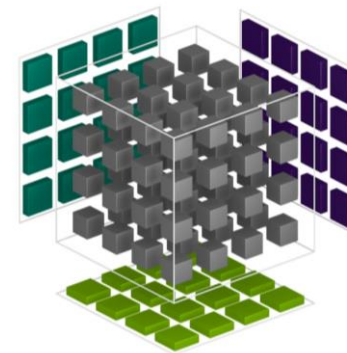
- 50%-100%
- 25%-50%
- 13%-25%
- 6%-13%

TENSOR CORE



TENSOR CORE

Mixed Precision Matrix Math
4x4 matrices



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

FP16 or FP32

FP16

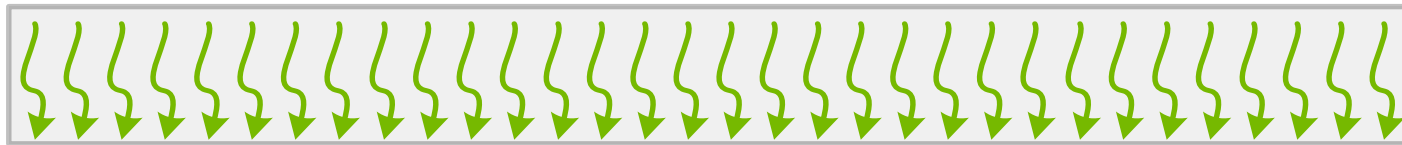
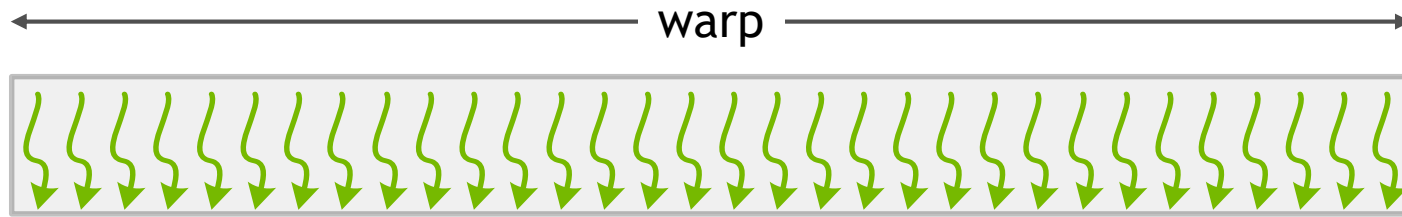
FP16

FP16 or FP32

$$\mathbf{D} = \mathbf{AB} + \mathbf{C}$$

TENSOR SYNCHRONIZATION

Full Warp 16x16 Matrix Math

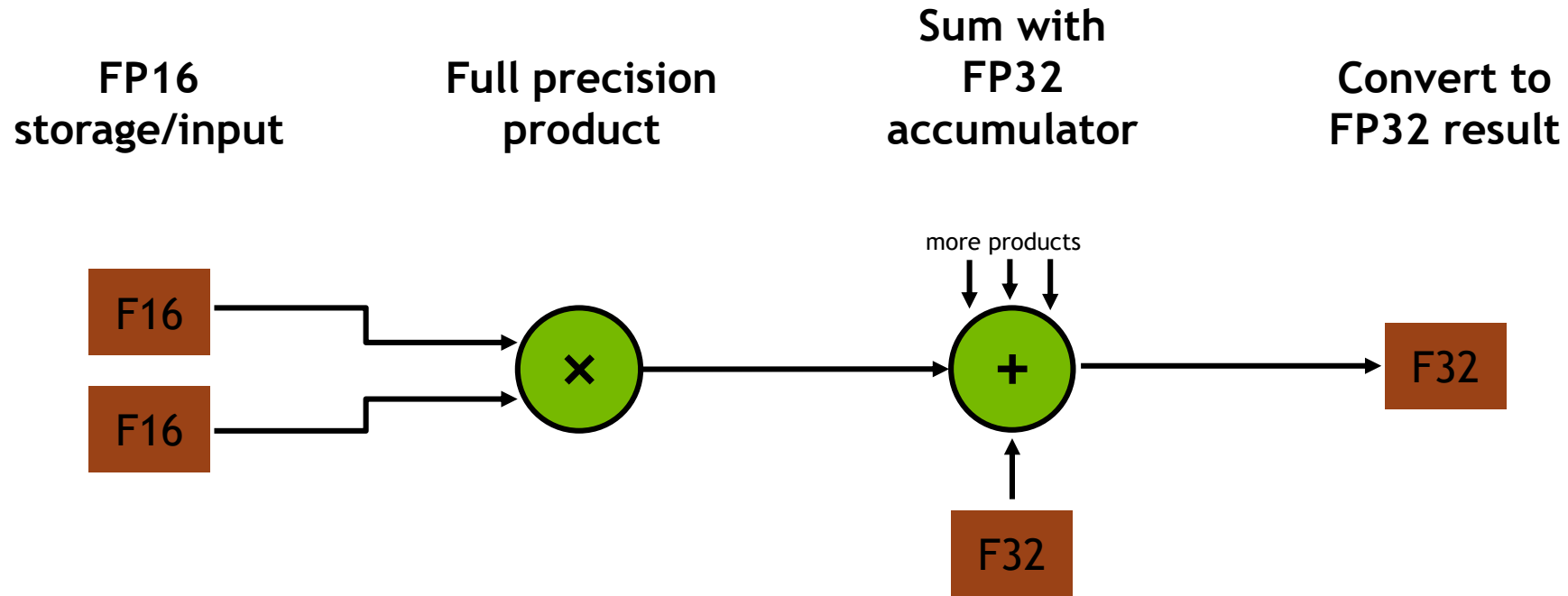


Warp-synchronizing operation

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

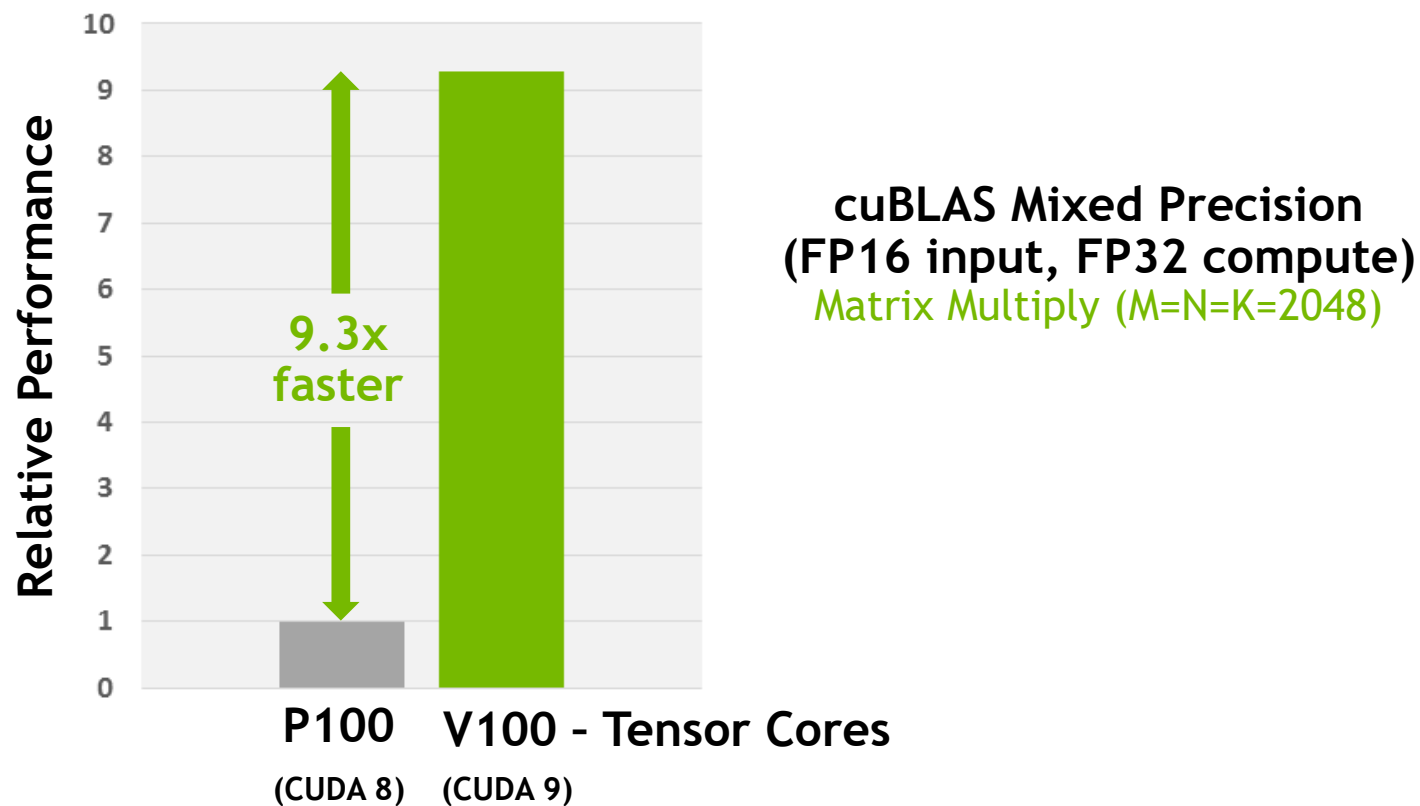
Result distributed across warp

VOLTA TENSOR OPERATION



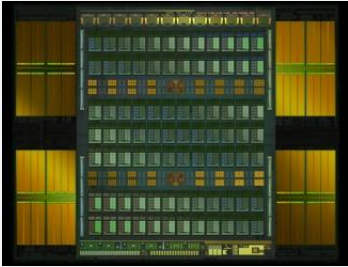
Also supports FP16 accumulator mode for inferencing

A GIANT LEAP FOR DEEP LEARNING



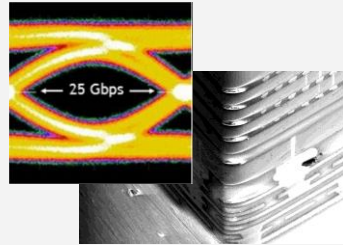
TESLA V100

Volta Architecture



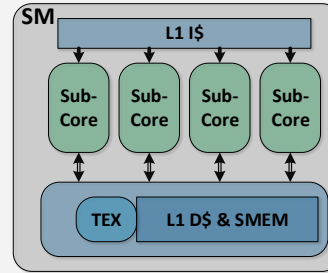
Most Productive GPU

Improved NVLink & HBM2



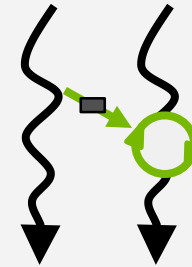
Efficient Bandwidth

New SM Core



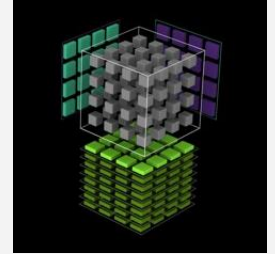
Performance & Programmability

Independent Thread Scheduling



New Algorithms

Tensor Core



120 Programmable
TFLOPS Deep Learning

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

The Fastest and Most Productive GPU for Deep Learning and HPC

QUESTIONS?