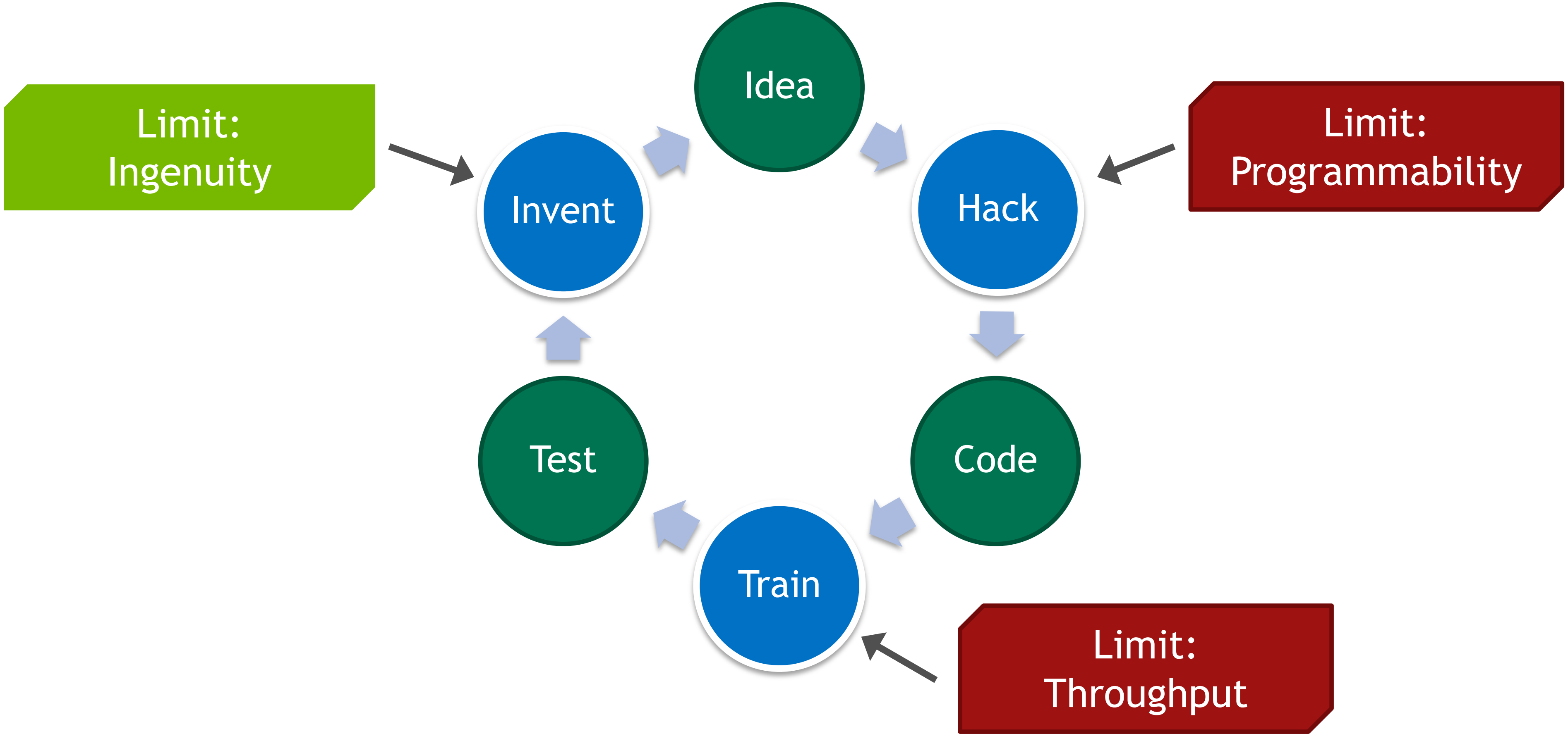
ACCELERATED COMPUTING FOR AI

Bryan Catanzaro, 7 December 2018

ACCELERATED COMPUTING: REDUCE LATENCY OF IDEA GENERATION

Research as a sequential, cyclic process



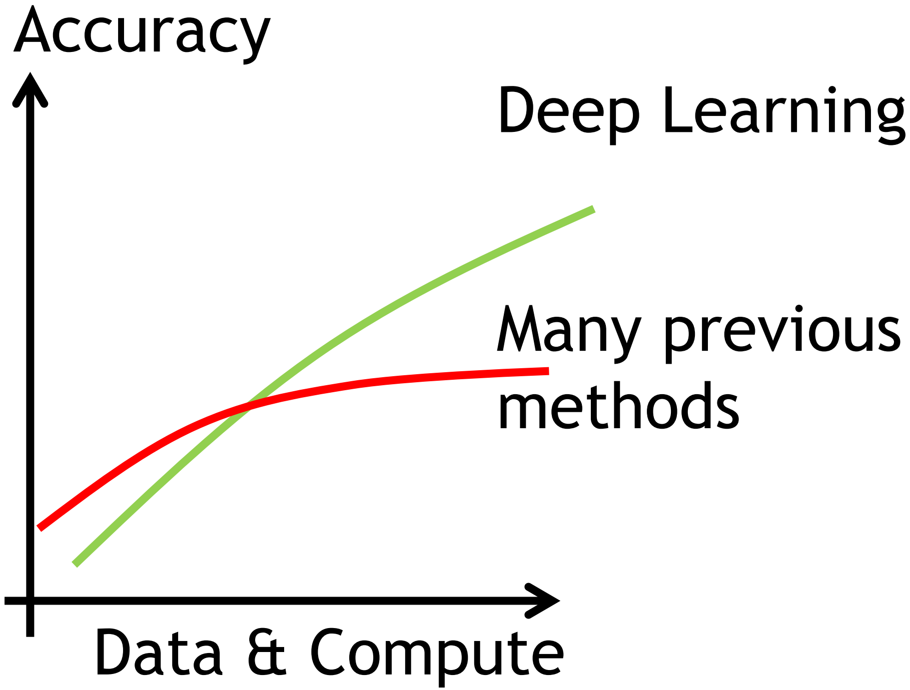
WHY IS DEEP LEARNING SUCCESSFUL

Big data sets

New algorithms

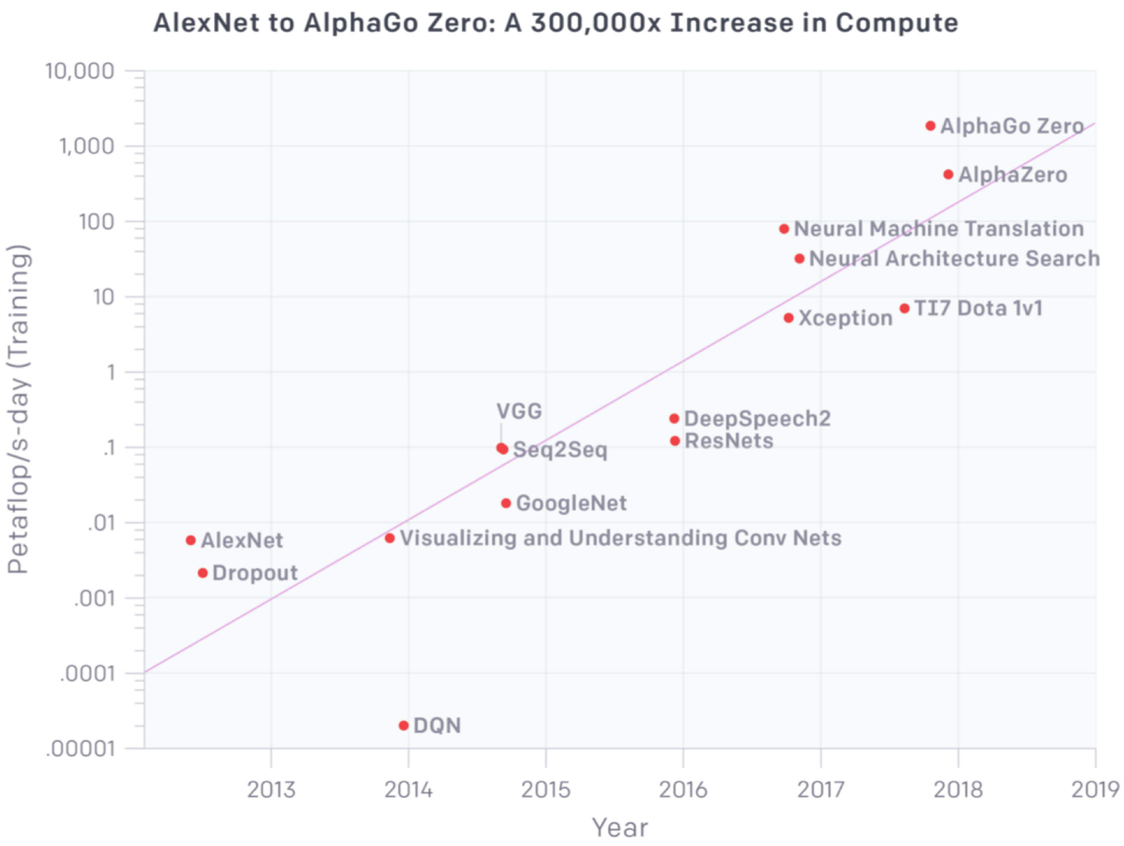
Computing hardware

Focus of this talk



MORE COMPUTE: MORE AI

https://blog.openai.com/ai-and-compute/



DEEP NEURAL NETWORKS 101

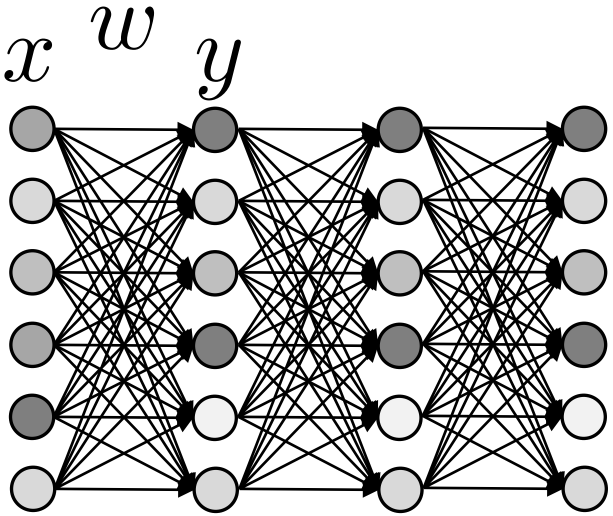
Simple, powerful function approximators

$$y_j = f\left(\sum_i w_{ij} x_i\right)$$

One layer: nonlinearity ∘ linear combination

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

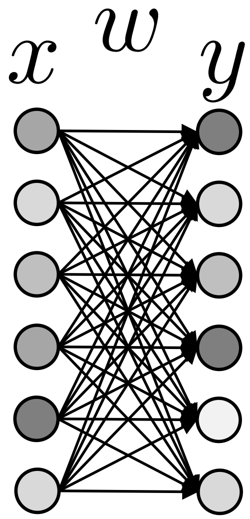
nonlinearity



Deep Neural Network

TRAINING NEURAL NETWORKS

$$y_j = f\left(\sum_i w_{ij} x_i\right)$$



Computation dominated by dot products

Multiple inputs, multiple outputs, batch means it is compute bound

Train one model: 20+ Exaflops

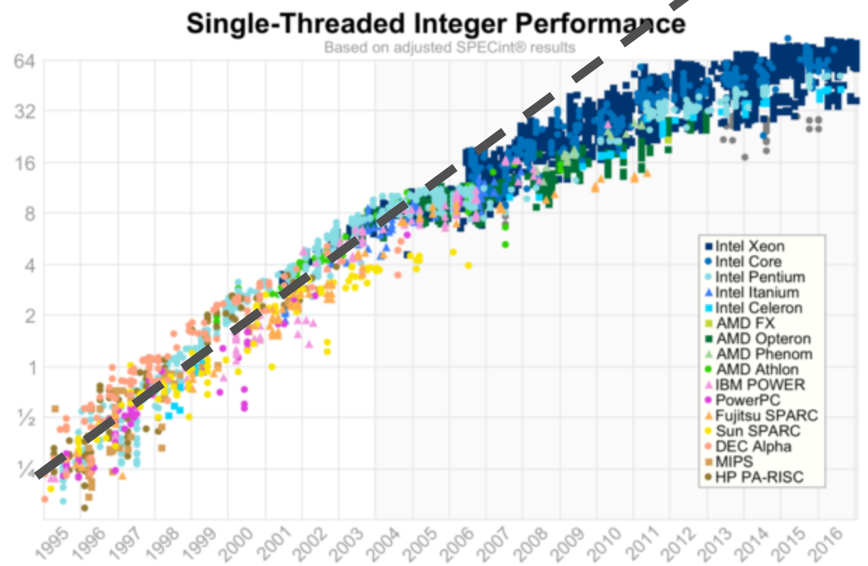
LAWS OF PHYSICS

Successful AI uses Accelerated Computing

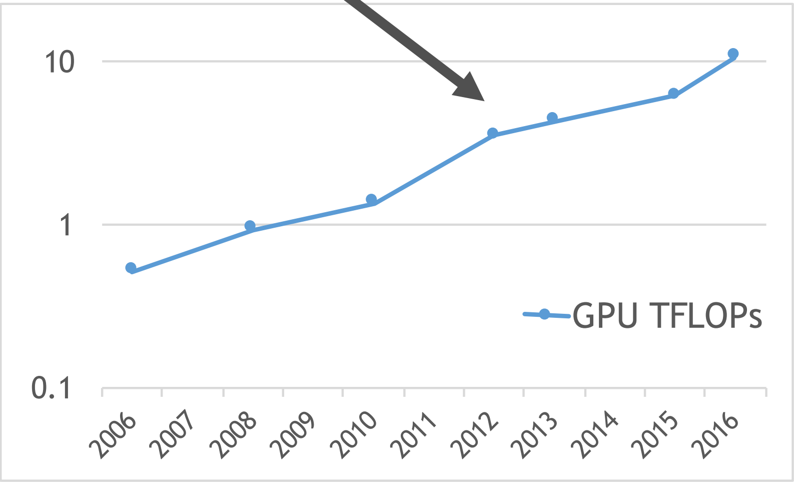
Volta

20X gap
and growing…

20X in 10 years



General Purpose Performance



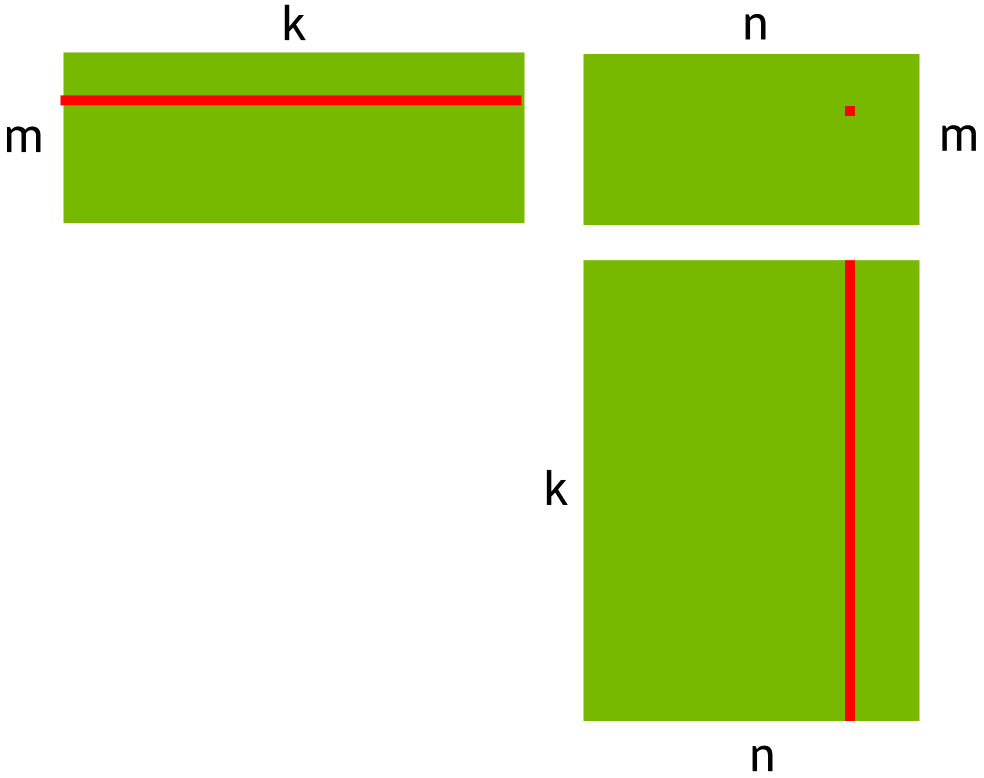
Accelerated Performance

@ctnzr

 NVIDIA.

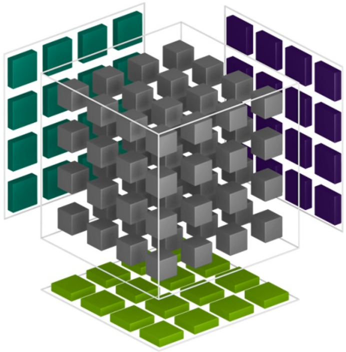
MATRIX MULTIPLICATION

Thor's hammer



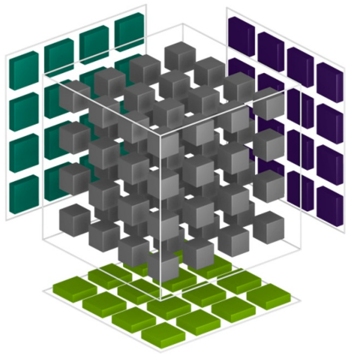
$O(n^2)$ communication

$O(n^3)$ computation



TENSOR CORE

Mixed Precision Matrix Math
4x4 matrices



$$
D = \left(\begin{matrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{matrix}\right) \left(\begin{matrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{matrix}\right) + \left(\begin{matrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{matrix}\right)
$$

FP16 or FP32 FP16 FP16 FP16 or FP32

$$
D = AB + C
$$

CHUNKY INSTRUCTIONS AMORTIZE OVERHEAD

Taking advantage of that $O(n^3)$ goodness

	Operation	Energy**	Overhead*
1 FMA →	HFMA	1.5pJ	2000%
4 FMA →	HDP4A	6.0pJ	500%
128 FMA →	HMMA	110pJ	27%

Tensor cores yield efficiency benefits, but are still programmable

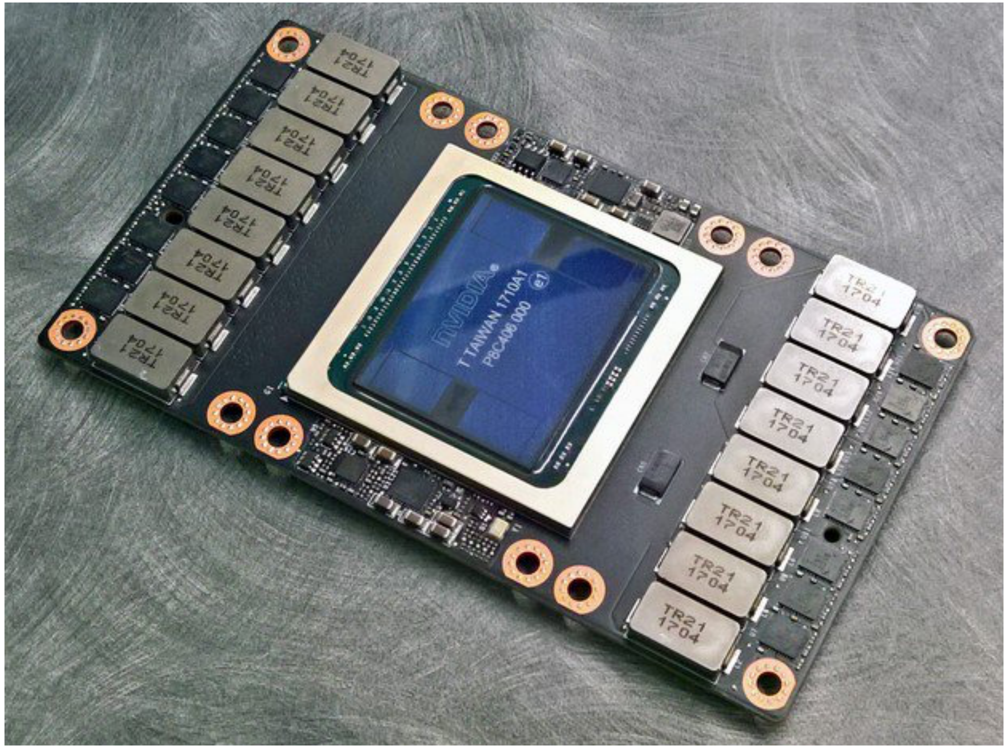
*Overhead is instruction fetch, decode, and operand fetch – 30pJ
**Energy numbers from 45nm process

TESLA V100

21B transistors
815 mm^2

80 SM*
5120 CUDA Cores
640 Tensor Cores

32 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

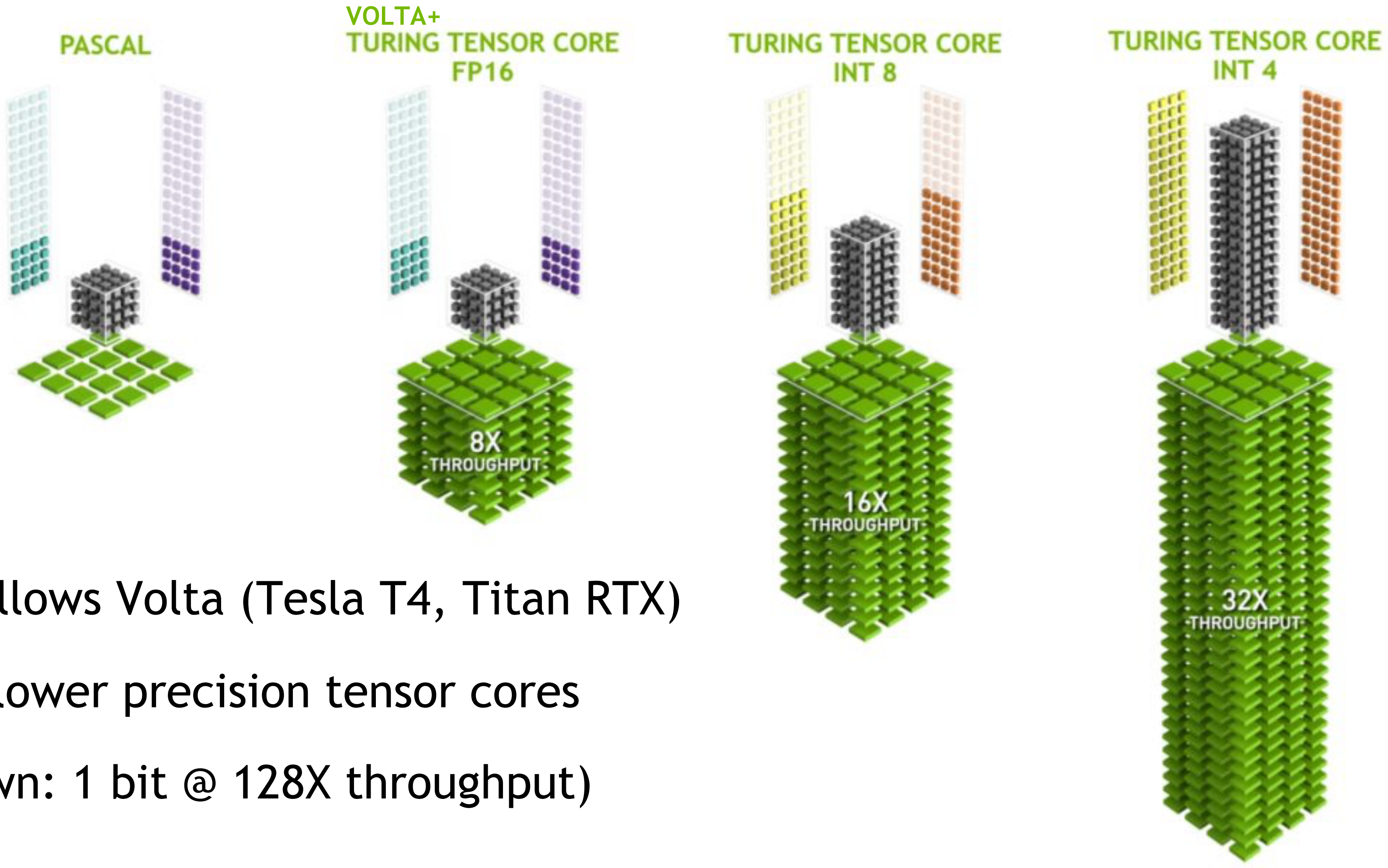


*full GV100 chip contains 84 SMs

GPU PERFORMANCE COMPARISON

	P100	V100	Ratio	T4
Training acceleration	10 TFLOPS	120 TFLOPS	**12x**	65 TFLOPS
Inference acceleration	20 TFLOPS	120 TFLOPS	**6x**	130 TOPS
FP64/FP32	5/10 TFLOPS	7.5/15 TFLOPS	**1.5x**	0.25/8 TFLOPS
Memory Bandwidth	720 GB/s	900 GB/s	**1.2x**	320 GB/s
NVLink Bandwidth	160 GB/s	300 GB/s	**1.9x**	--
L2 Cache	4 MB	6 MB	**1.5x**	4 MB
L1 Caches	1.3 MB	10 MB	**7.7x**	6 MB
Power	250 W	300 W	**1.2x**	70 W

PRECISION



32 bit accumulation

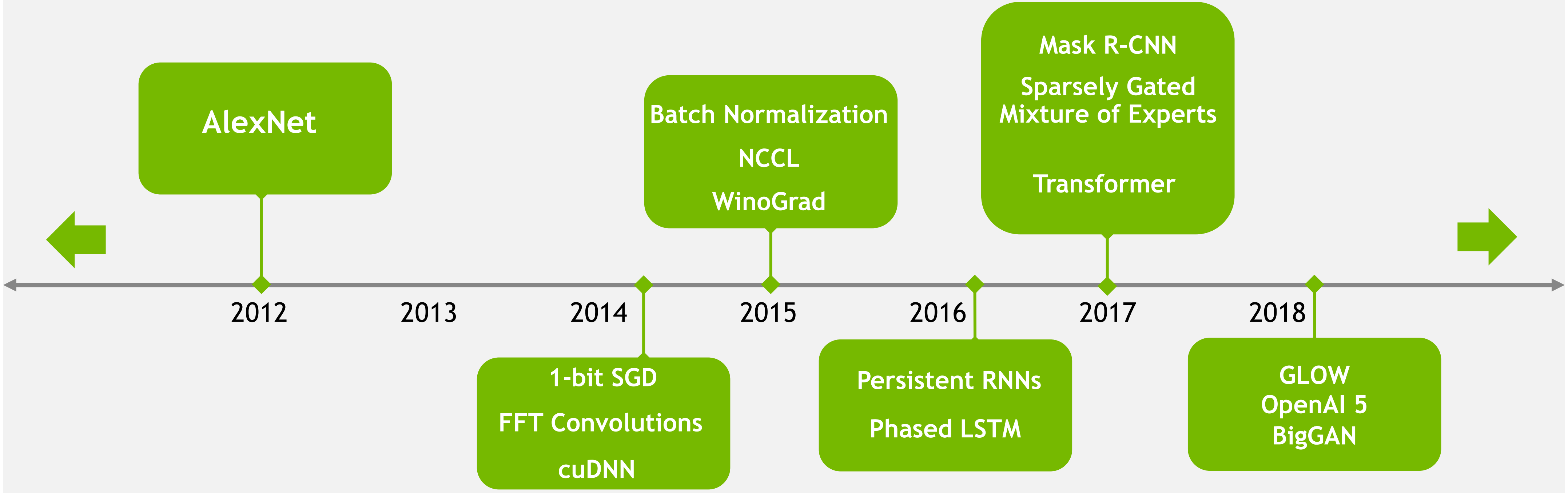
Turing follows Volta (Tesla T4, Titan RTX)

Includes lower precision tensor cores

(Not shown: 1 bit @ 128X throughput)

COMPUTATIONAL EVOLUTION

Deep learning changes every day: In tension with Specialization



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

PROGRAMMABILITY

Where the research happens

Computation dominated by linear operations

But the research happens elsewhere:

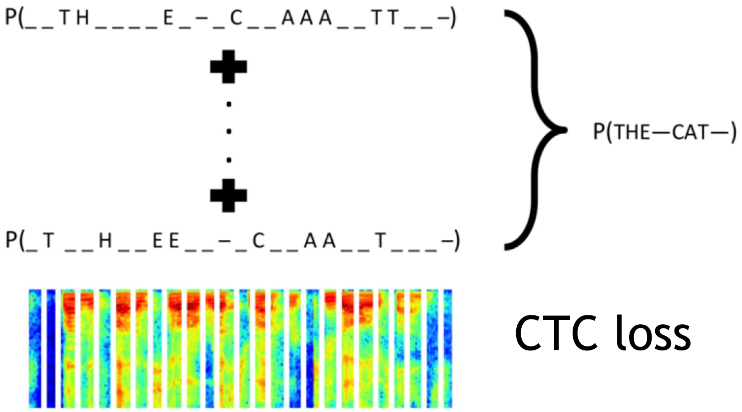
New loss functions

New non-linearities

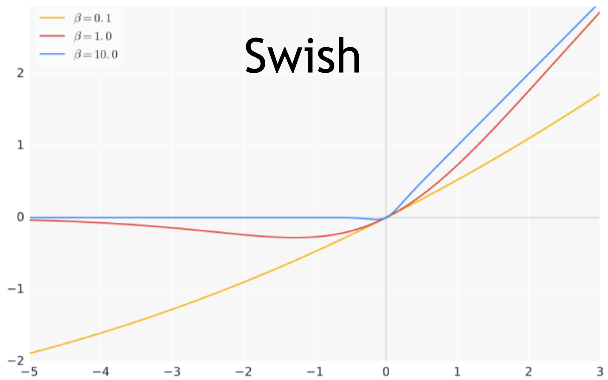
New normalizations

New inputs & outputs

CUDA is fast and flexible parallel C++



CTC loss

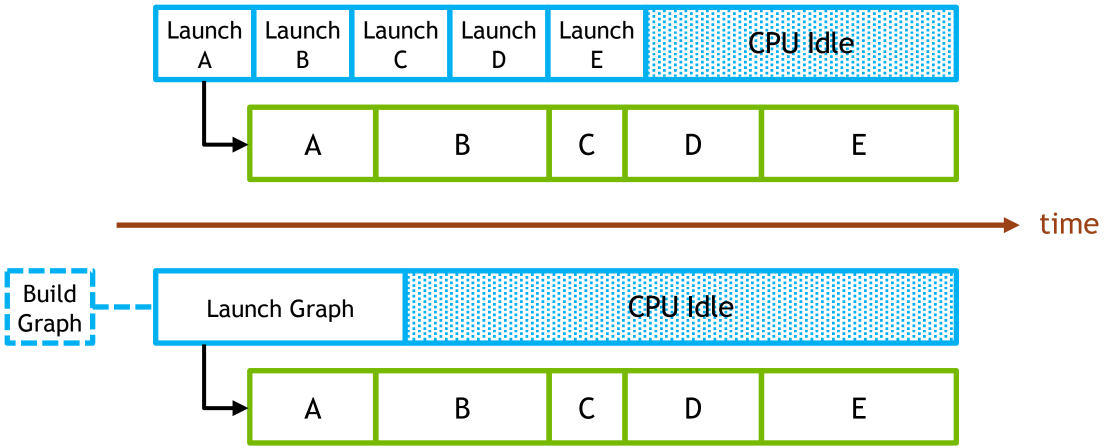


REFINING CUDA: CUDA GRAPHS

Latency & Overhead Reductions

Launch latencies:

- CUDA 10.0 takes at least 2.2us CPU time to launch **each** CUDA kernel on Linux
- Pre-defined graph allows launch of **any number** of kernels in **one single operation**



Useful for small models

Works with JIT graph compilers

CUDA LIBRARIES

Optimized Kernels

CUBLAS: Linear algebra

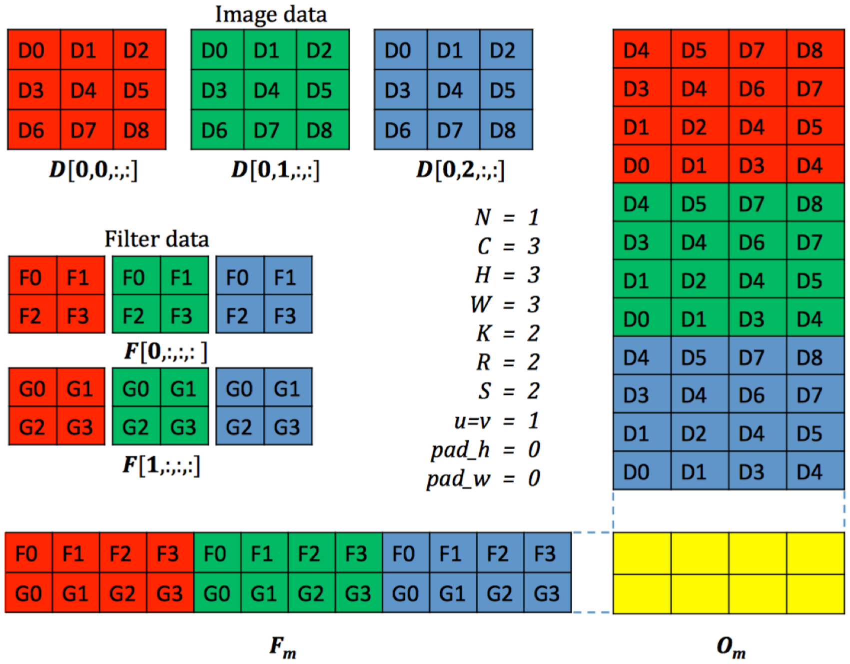
Many flavors of GEMM

CUDNN: Neural network kernels

Convolutions (direct, Winograd, FFT)

Can achieve > Speed of Light!

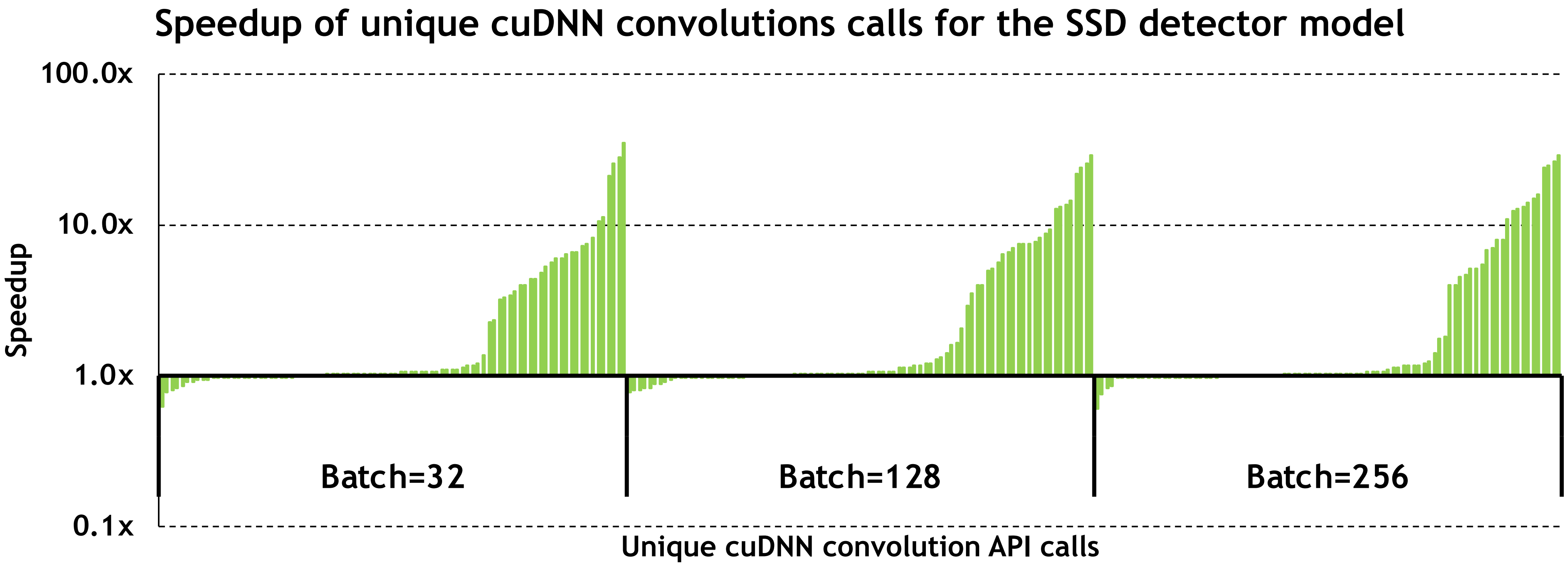
Recurrent Neural Networks



Lowering Convolutions to GEMM

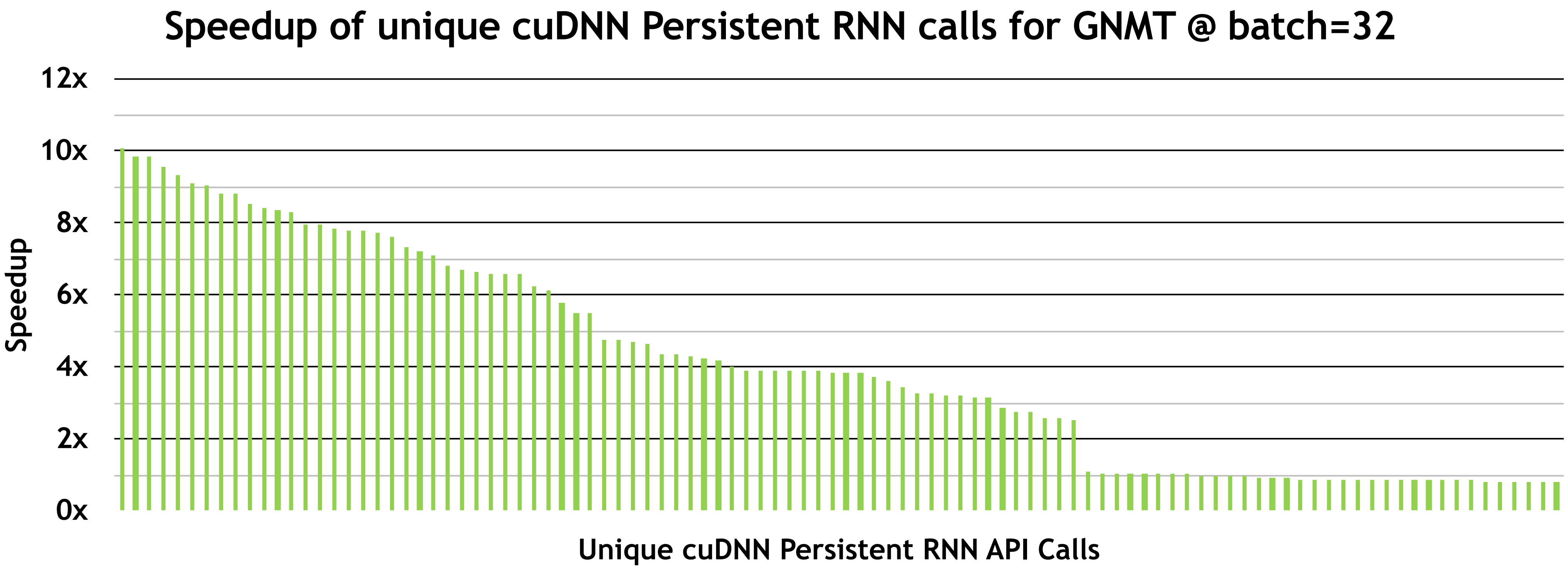
IMPROVED HEURISTICS FOR CONVOLUTIONS

cuDNN 7.4.1 (Nov 2018) vs. cuDNN 7.0.5 (Dec 2017)



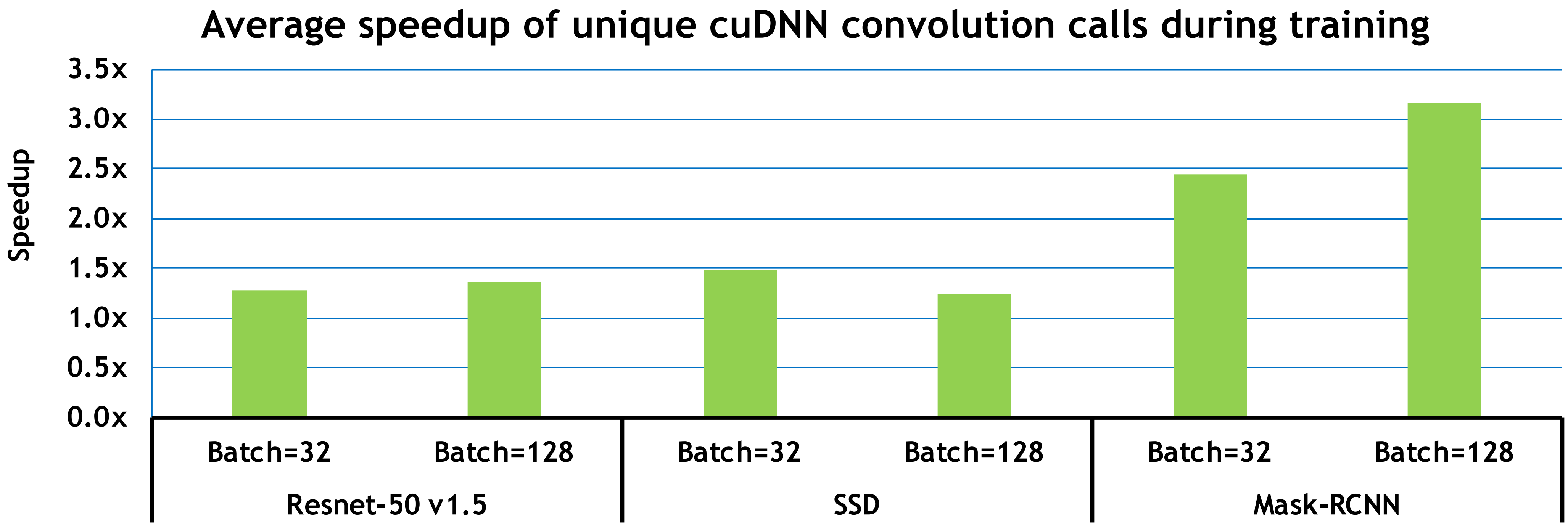
PERSISTENT RNN SPEEDUP ON V100

cuDNN 7.4.1 (Nov 2018) vs. cuDNN 7.0.5 (Dec 2017)



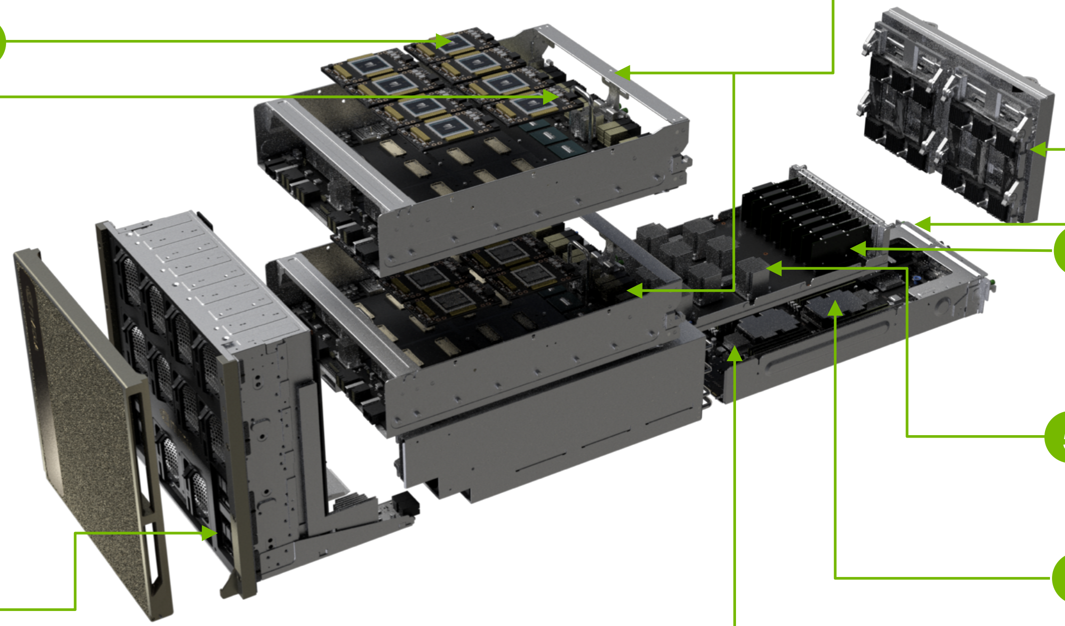
TENSORCORES WITH FP32 MODELS

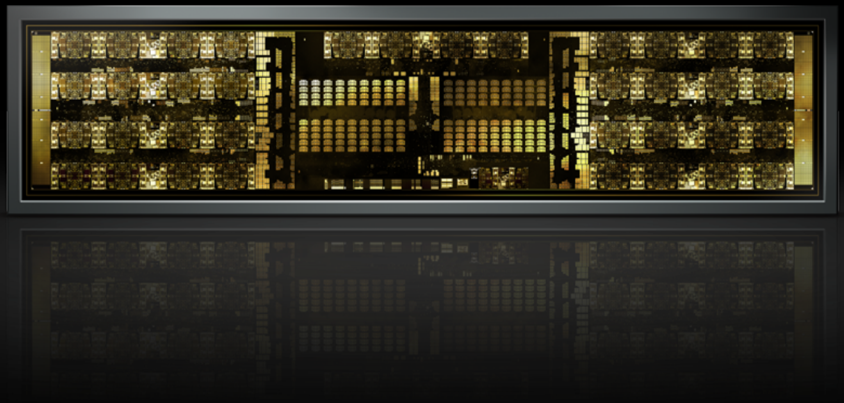
cuDNN 7.4.1 (Nov 2018) vs. cuDNN 7.0.5 (Dec 2017)



- Enabled as an experimental feature in the TensorFlow NGC Container via an environment variable (same for cuBLAS)
- Should use in conjunction with Loss Scaling

NVIDIA DGX-2


1 NVIDIA Tesla V100 32GB
2 Two GPU Boards
8 V100 32GB GPUs per board
6 NVSwitches per board
512GB Total HBM2 Memory
interconnected by
Plane Card
3 Twelve NVSwitches
2.4 TB/sec bi-section
bandwidth
4 Eight EDR Infiniband/100 GigE
1600 Gb/sec Total
Bi-directional Bandwidth
5 PCIe Switch Complex
6 Two Intel Xeon Platinum CPUs
7 1.5 TB System Memory
8 30 TB NVME SSDs
Internal Storage
9 Dual 10/25 GigE

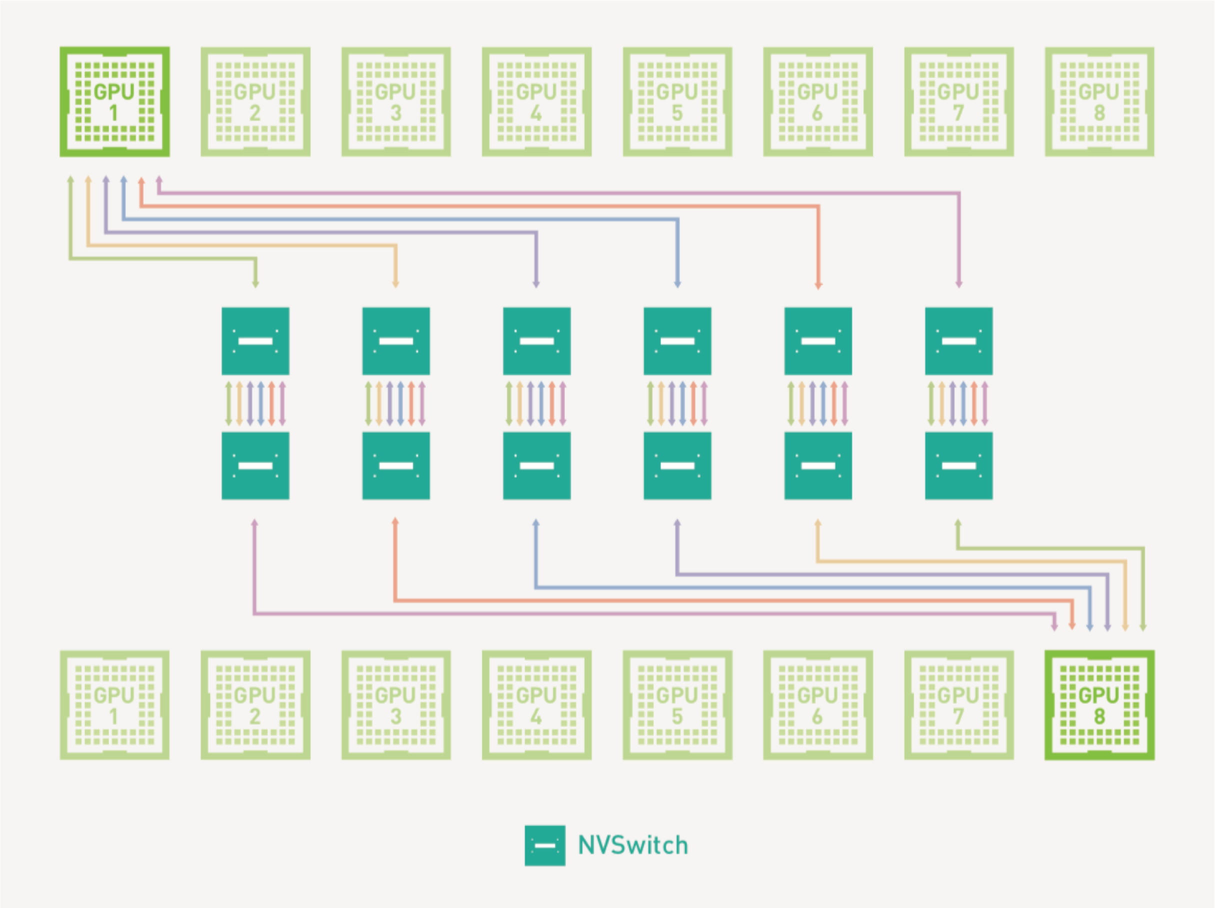


NVSWITCH: NETWORK FABRIC FOR AI

- 2.4 TB/s bisection bandwidth
- Equivalent to a PCIe bus with 1,200 lanes

- Inspired by leading edge research that demands unrestricted model parallelism
    - Each GPU can make random reads, writes and atomics to each other GPU's memory
- 18 NVLink ports per switch

DGX-2: ALL-TO-ALL CONNECTIVITY



Each switch connects to 8 GPUs

Each GPU connects to 6 switches

Each switch connects to the other half of the system with 8 links

2 links on each switch reserved

FRAMEWORKS

Several AI frameworks

- Let researchers prototype rapidly
- Different perspectives on APIs
- All are GPU accelerated

AUTOMATIC MIXED PRECISION

Four Lines of Code => 2.3x Training Speedup in PyTorch (RN-50)

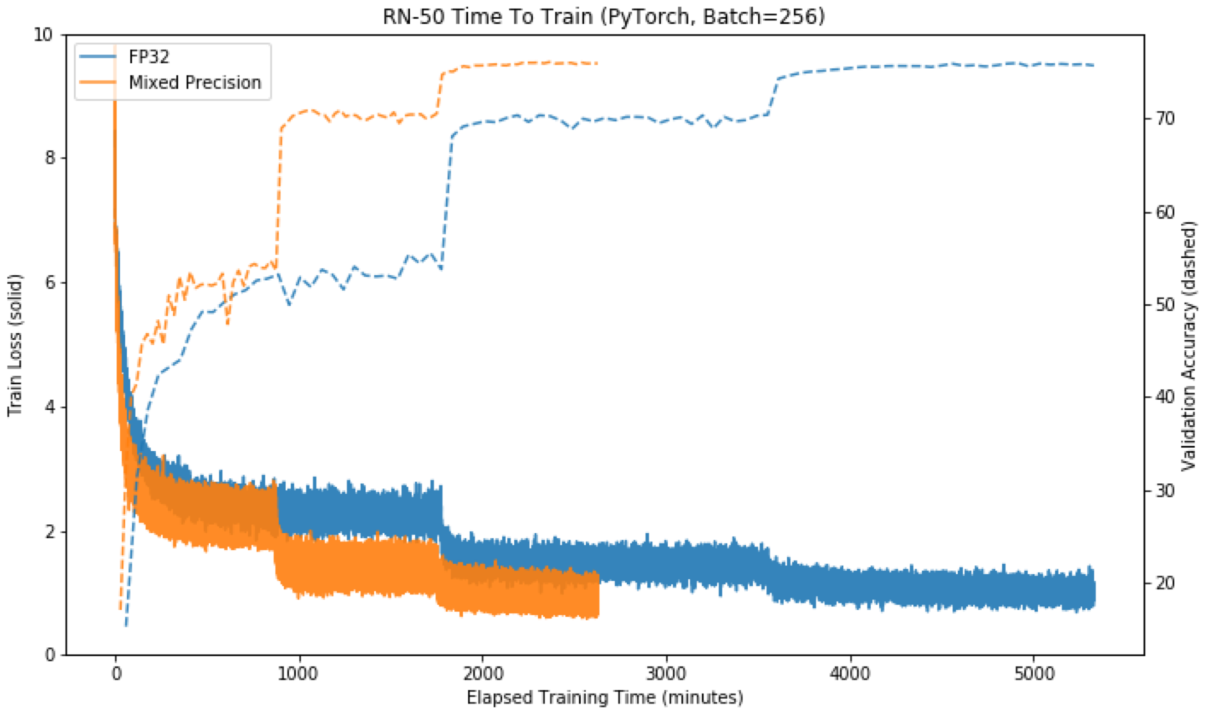
- Mixed precision training uses half-precision floating point (FP16) to accelerate training
- You can start using mixed precision today with four lines of code
- This example uses AMP: Automatic Mixed Precision, a PyTorch library
  - No hyperparameters changed

```
+ amp_handle = amp.init()
# ... Define model and optimizer
for x, y in dataset:
  prediction = model(x)
  loss = criterion(prediction, y)
- loss.backward()
+ with amp_handle.scale_loss(loss,
+         optimizer) as scaled_loss:
+   scaled_loss.backward()
optimizer.step()
```

AUTOMATIC MIXED PRECISION

Four Lines of Code => 2.3x Training Speedup (RN-50)

- Real-world single-GPU runs using default PyTorch ImageNet example
  - NVIDIA PyTorch 18.08-py3 container
  - AMP for mixed precision
- Minibatch=256
- Single GPU RN-50 speedup for FP32 -> M.P. (with 2x batch size):
  - MxNet: 2.9x
  - TensorFlow: 2.2x
  - TensorFlow + XLA: ~3x
  - PyTorch: 2.3x
- Work ongoing to bring to 3x everywhere



DATA LOADERS

Fast training means greater demands on the rest of the system

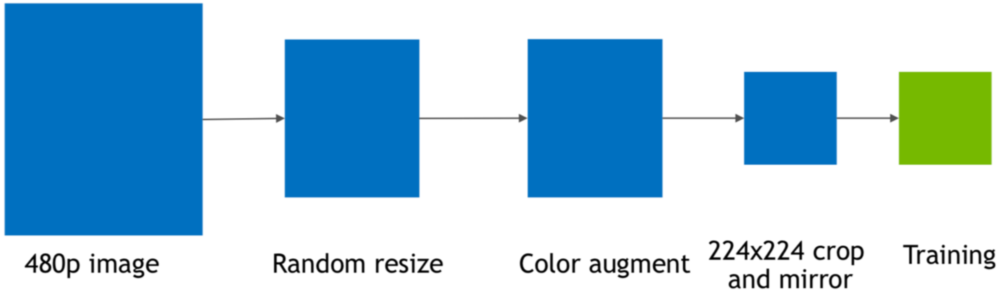
Data transfer from storage (network)

CPU bottlenecks happen fast

GPU accelerated, user defined data loaders

Move decompression & augmentation to GPU

Both for still images and videos



Move all this to the GPU with
DALI: https://github.com/NVIDIA/DALI

Research video data loader using HW decoding:
NVVL: https://github.com/NVIDIA/NVVL

SIMULATION

Many important AI tasks involve agents interacting with the real world

For this, you need simulators

Physics

Appearance

Simulation has a big role to play in AI progress

RL needs good simulators – NVIDIA PhysX is now open source:

https://github.com/NVIDIAGameWorks/PhysX-3.4

MAKE INGENUITY THE LIMITING FACTOR

Accelerated Computing for AI

High computational intensity +

Programmability & flexibility fundamental for AI systems

Need a systems approach

Chips are not enough

And lots of software to make it all useful

Bryan Catanzaro
@ctnzr

NVIDIA