CSCI 381/780
Cloud Computing

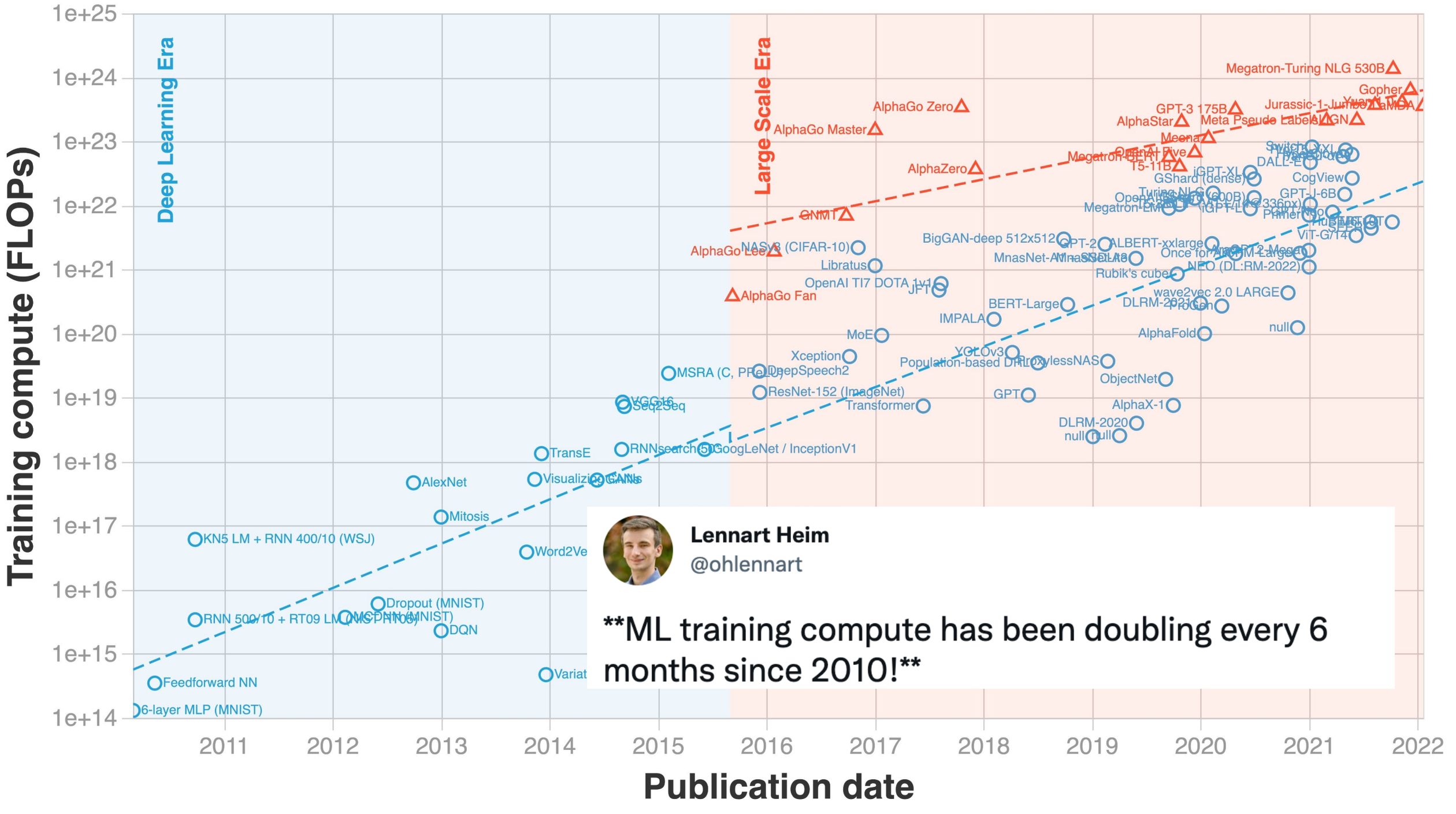
Machine Learning

Jun Li
Queens College

Machine learning has achieved great successes in various applications, such as computer vision, NLP, robotics, etc.

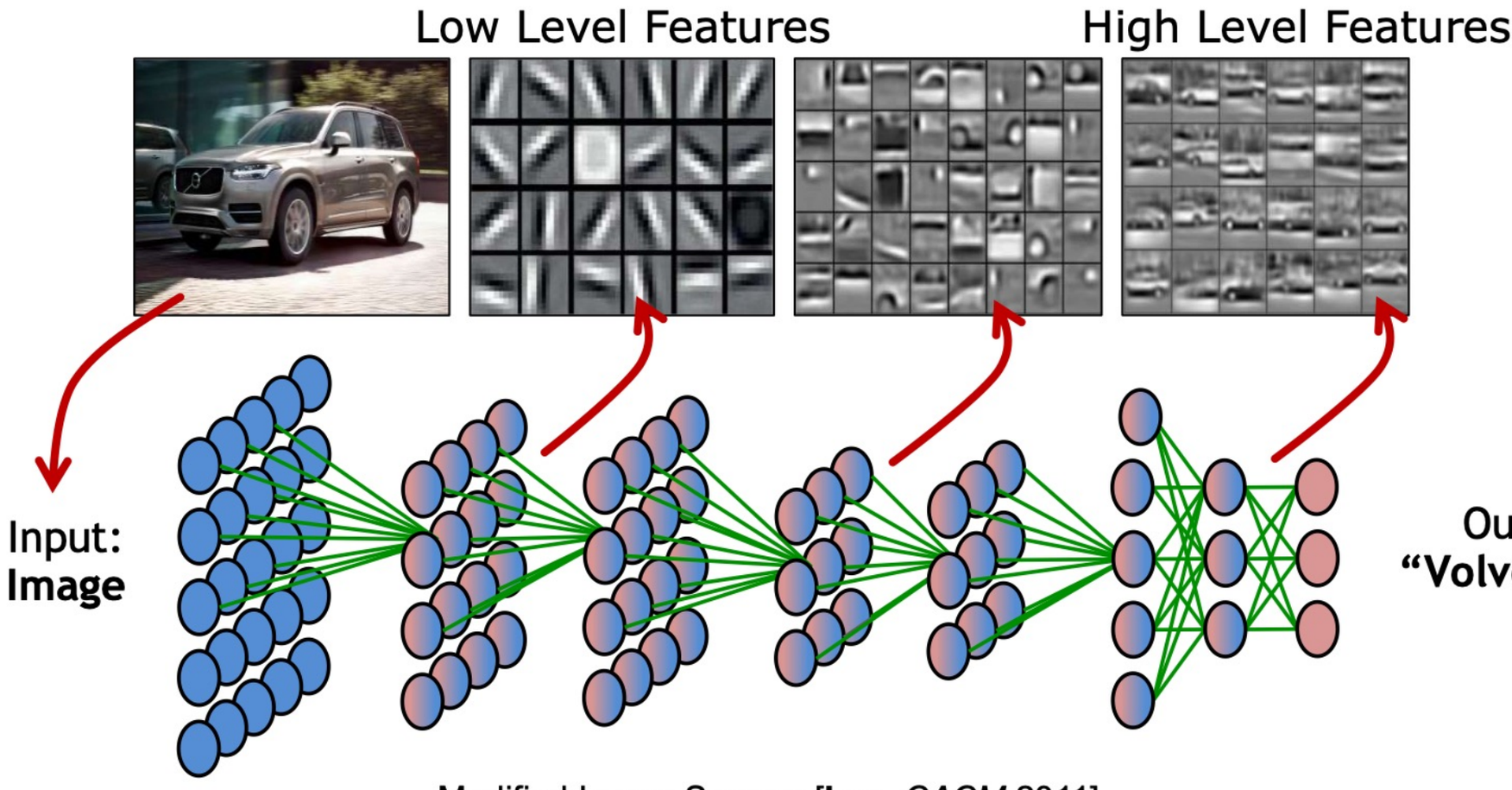
Training compute (FLOPs) of milestone Machine Learning systems over time

n = 99



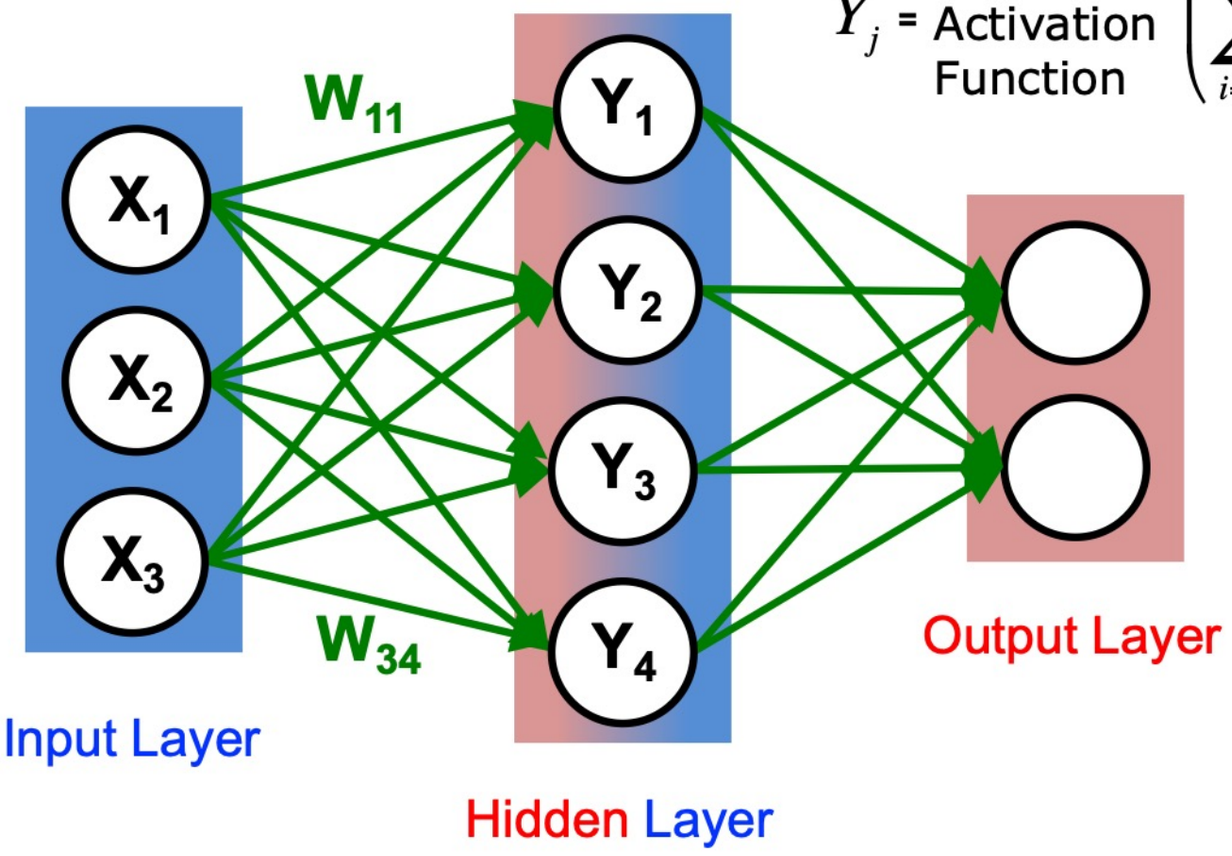
J. Sevilia, et al., “Compute Trends Across Three Eras of Machine Learning”, arXiv:2202.05924, 2022

What are Deep Neural Networks?



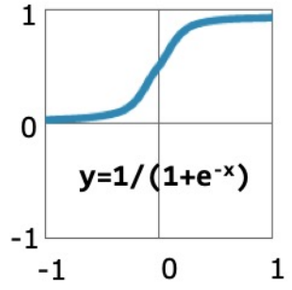
Modified Image Source: [**Lee**, *CACM* 2011]

Weighted Sums



$$Y_j = \text{Nonlinear Activation Function}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$

Sigmoid



Rectified Linear Unit (ReLU)

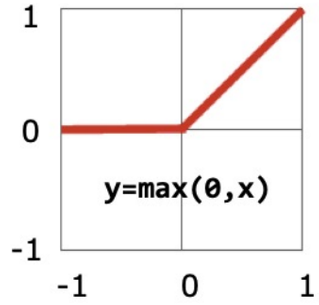
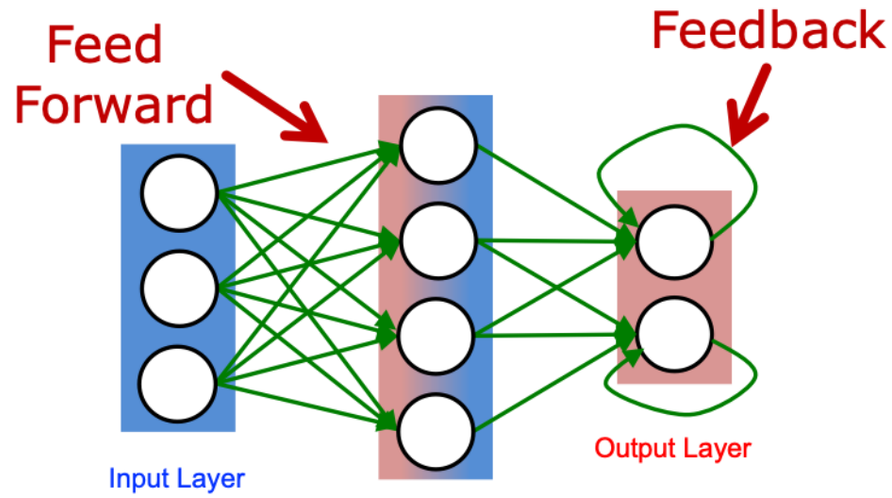


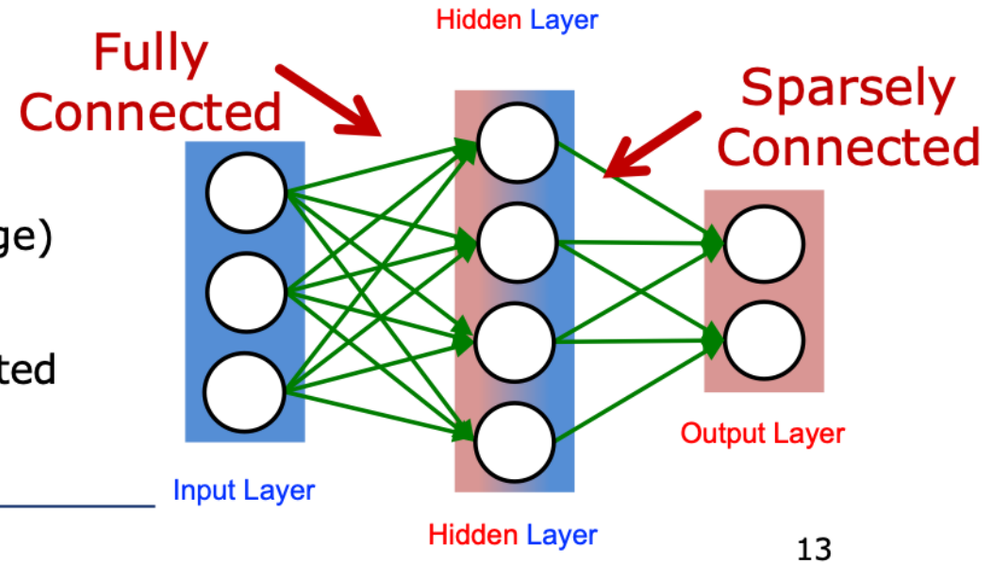
Image source: Caffe tutorial

Key operation is
multiply and accumulate (MAC)
Accounts for > 90% of computation

Popular Types of Layers in DNNs

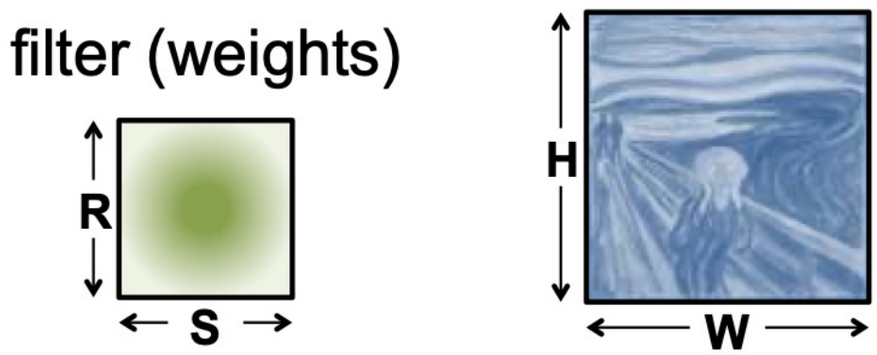
- **Fully Connected Layer**
  - Feed forward, fully connected
  - Multilayer Perceptron (MLP)
- **Convolutional Layer**
  - Feed forward, sparsely-connected w/ weight sharing
  - Convolutional Neural Network (CNN)
  - Typically used for images
- **Recurrent Layer**
  - Feedback
  - Recurrent Neural Network (RNN)
  - Typically used for sequential data (e.g., speech, language)
- **Attention Layer/Mechanism**
  - Attention (matrix multiply) + feed forward, fully connected
  - Transformer [**Vaswani**, *NeurIPS* 2017]



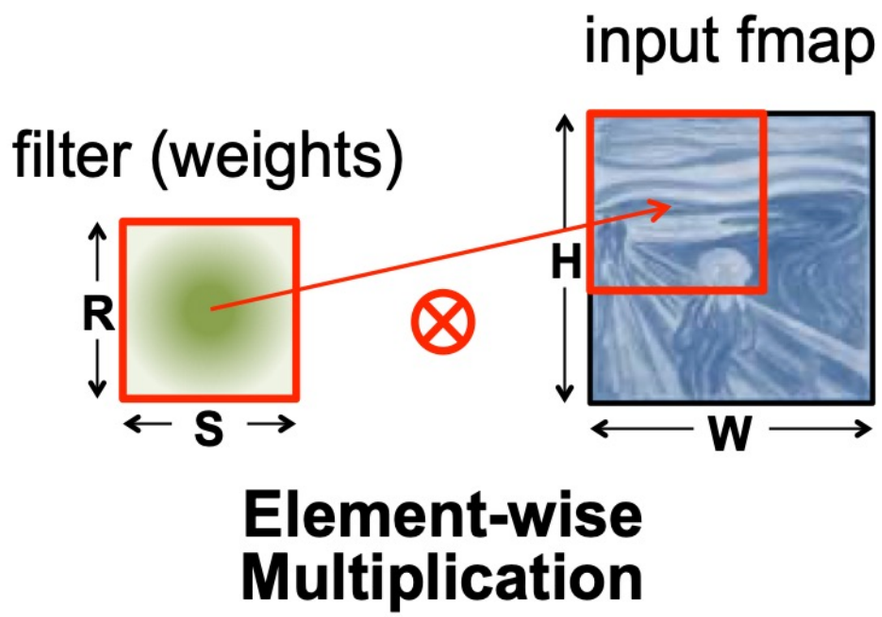


High-Dimensional Convolution in CNN

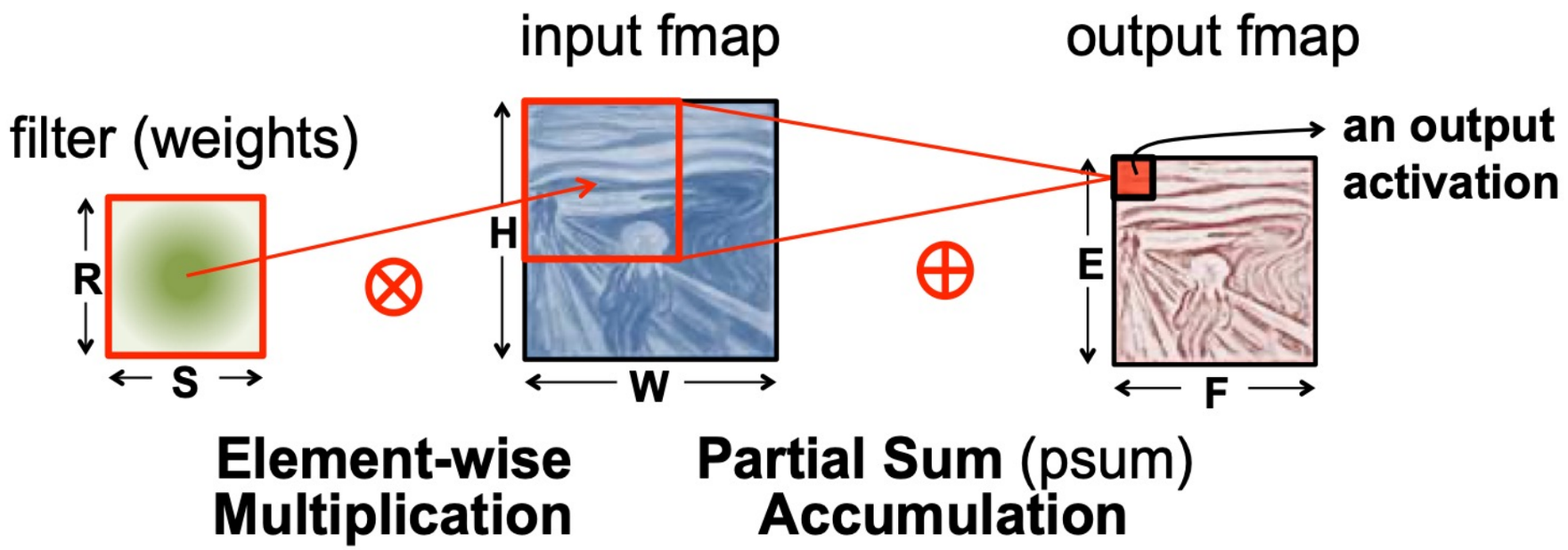
a plane of input activations
a.k.a. **input feature map (fmap)**



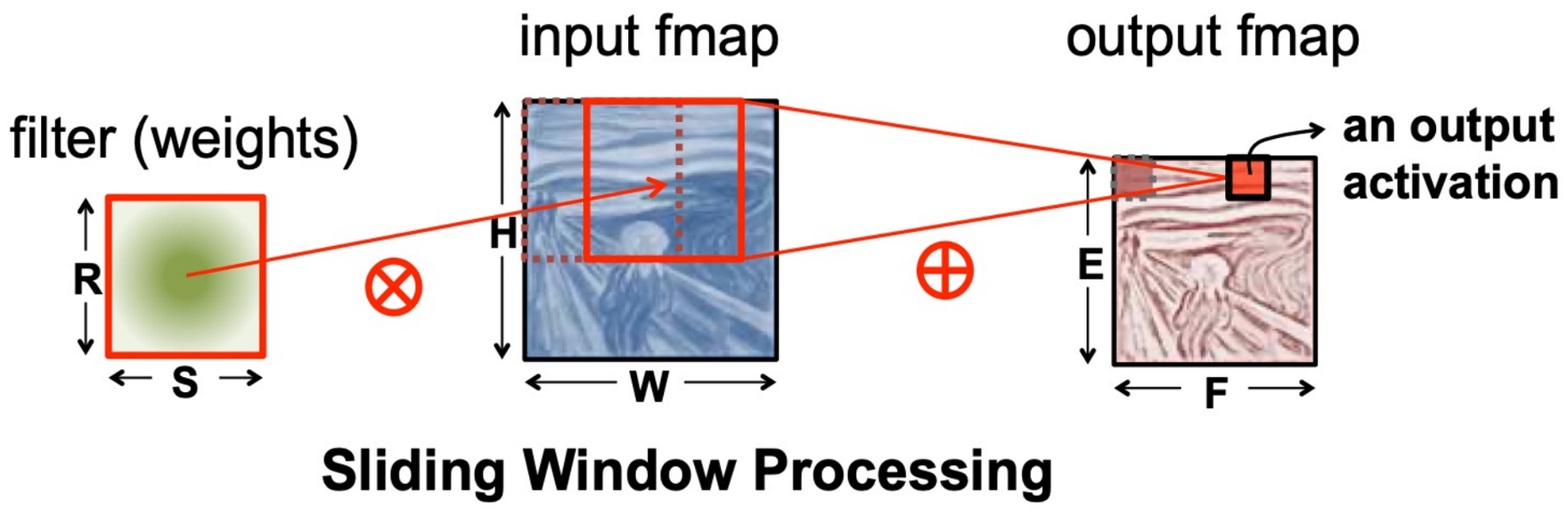
High-Dimensional Convolution in CNN



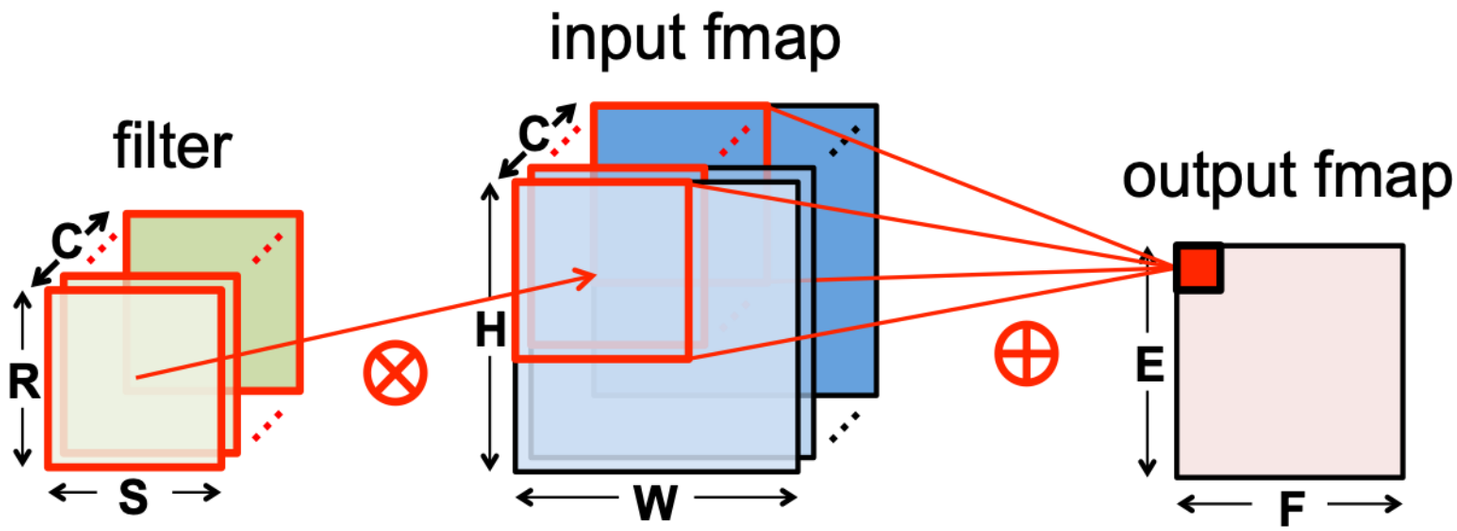
High-Dimensional Convolution in CNN



High-Dimensional Convolution in CNN



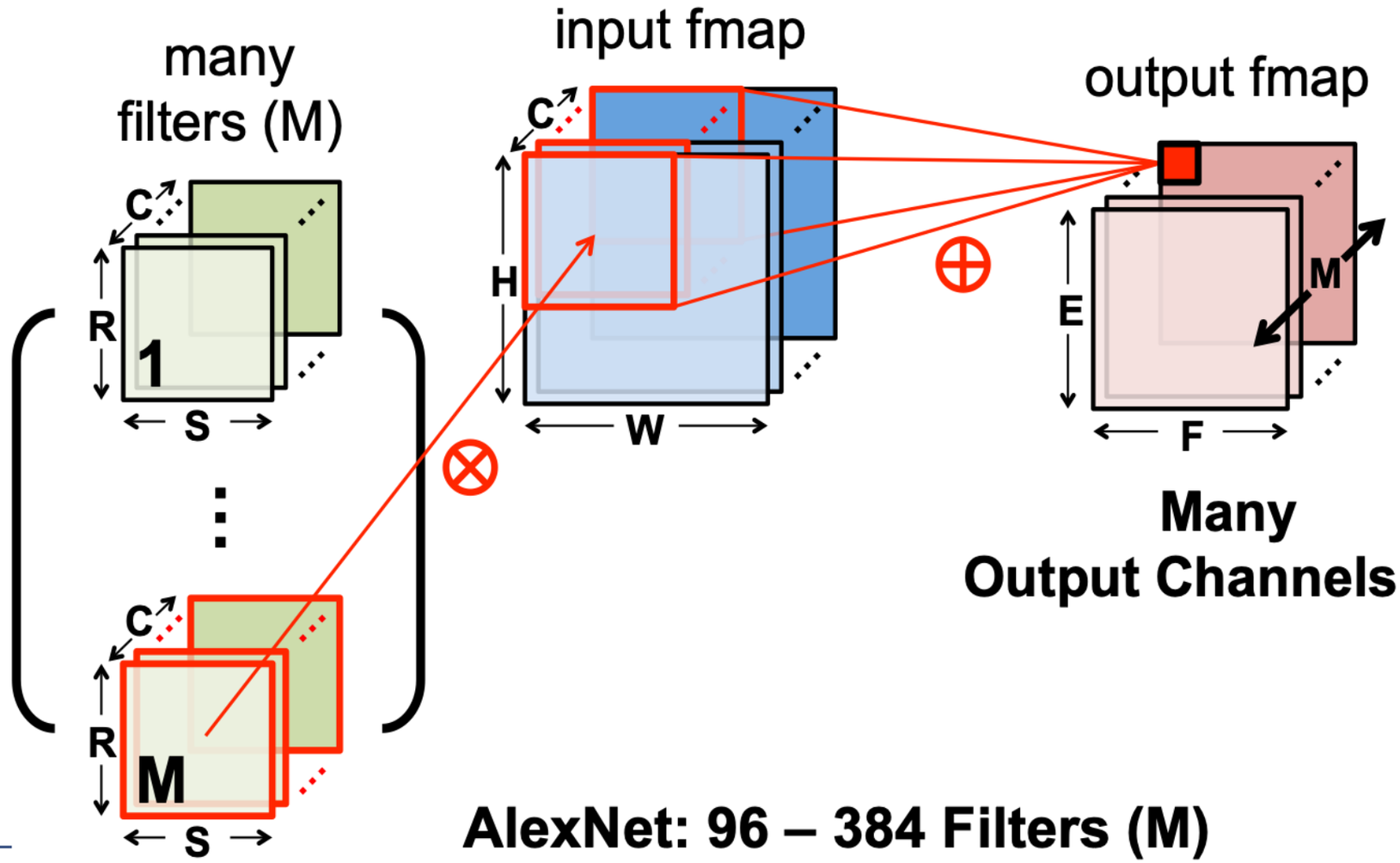
High-Dimensional Convolution in CNN



Many Input Channels (C)

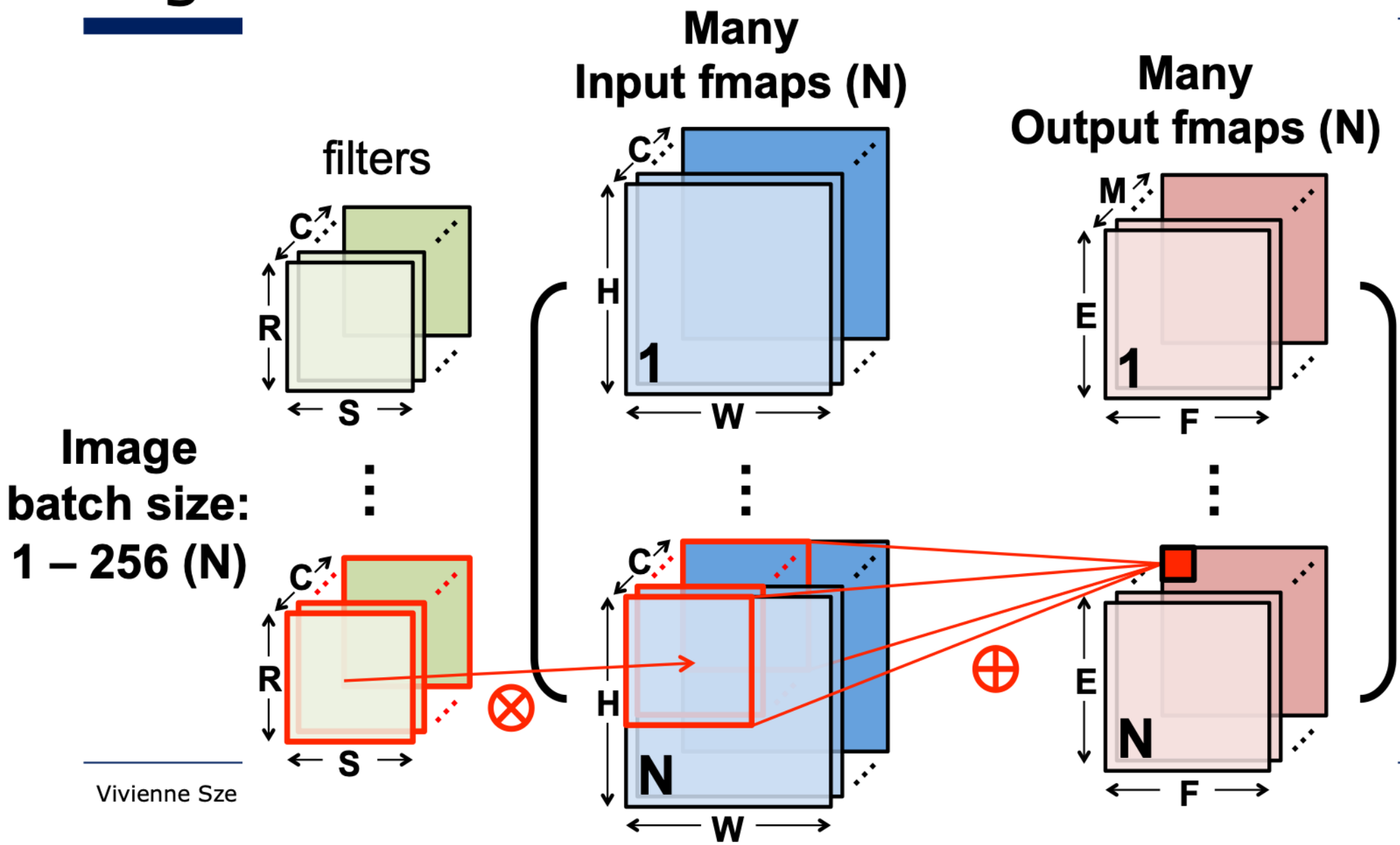
AlexNet: 3 – 192 Channels (C)

High-Dimensional Convolution in CNN

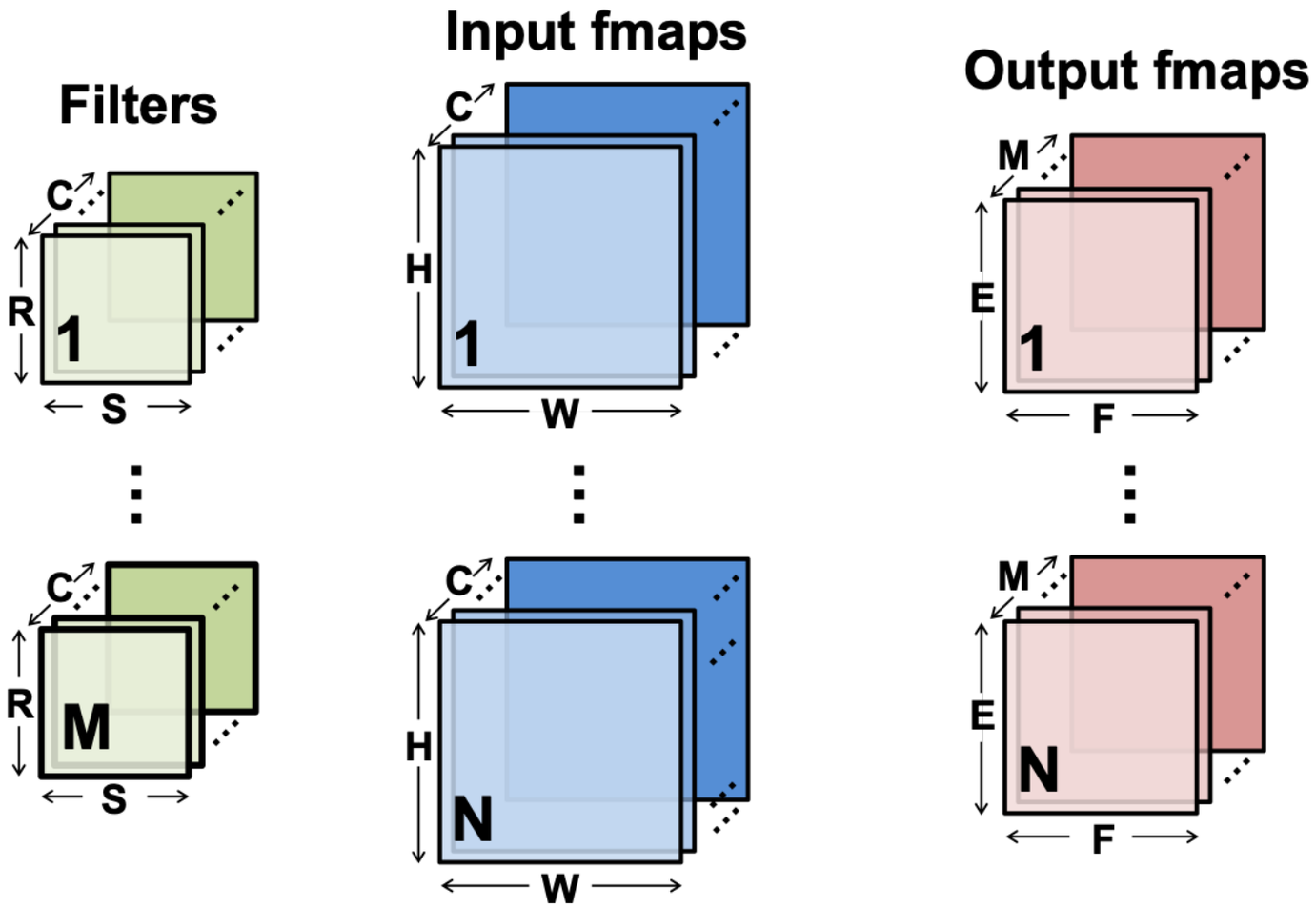


AlexNet: 96 – 384 Filters (M)

High-Dimensional Convolution in CNN



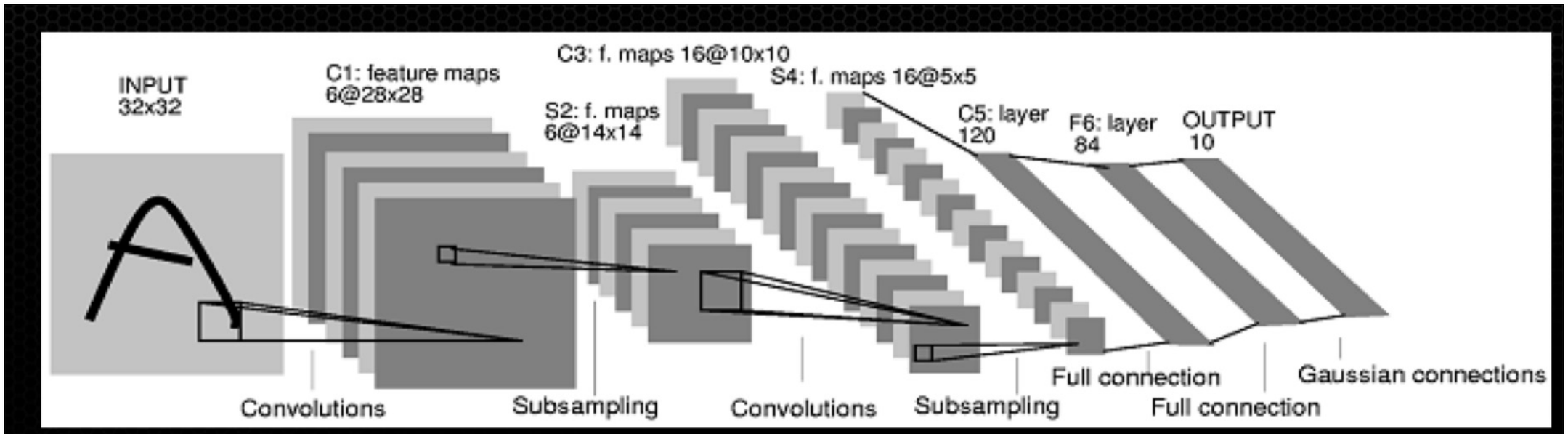
Define Shape for Each Layer



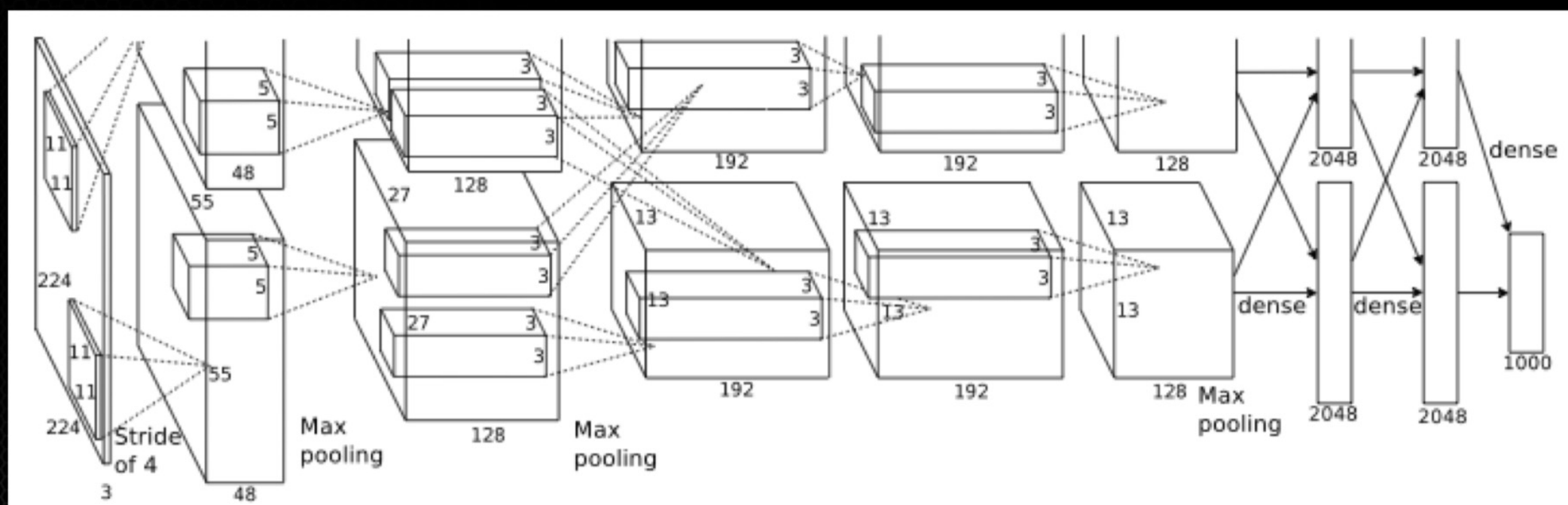
Shape **varies** across layers

H – Height of input fmap (activations)
W – Width of input fmap (activations)
C – Number of 2-D input fmaps /filters (channels)
R – Height of 2-D filter (weights)
S – Width of 2-D filter (weights)
M – Number of 2-D output fmaps (channels)
E – Height of output fmap (activations)
F – Width of output fmap (activations)
N – Number of input fmaps/output fmaps (batch size)

Convolutional Neural Network



Y. LeCun et al. 1989-1998 : Handwritten digit reading



Layers with Varying Shapes

MobileNetV3-Large Convolutional Layer Configurations

Block	Filter Size (RxS)	# Filters (M)	# Channels (C)
1	3x3	16	3
		⋮	
3	1x1	64	16
3	3x3	64	1
3	1x1	24	64
		⋮	
6	1x1	120	40
6	5x5	120	1
6	1x1	40	120
		⋮	

[**Howard**, *ICCV* 2019]

Popular DNN Models

Metrics	LeNet-5	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50	EfficientNet-B4
Top-5 error (ImageNet)	n/a	16.4	7.4	6.7	5.3	3.7*
Input Size	28x28	227x227	224x224	224x224	224x224	380x380
# of CONV Layers	**2**	**5**	**16**	**21 (depth)**	**49**	**96**
# of Weights	2.6k	2.3M	14.7M	6.0M	23.5M	14M
# of MACs	283k	666M	15.3G	1.43G	3.86G	4.4G
# of FC layers	**2**	**3**	**3**	**1**	**1**	**65****
# of Weights	58k	58.6M	124M	1M	2M	4.9M
# of MACs	58k	58.6M	124M	1M	2M	4.9M
Total Weights	**60k**	**61M**	**138M**	**7M**	**25.5M**	**19M**
Total MACs	**341k**	**724M**	**15.5G**	**1.43G**	**3.9G**	**4.4G**
Reference	**Lecun,** *PIEEE* 1998	**Krizhevsky,** *NeurIPS* 2012	**Simonyan,** *ICLR* 2015	**Szegedy,** *CVPR* 2015	**He,** *CVPR* 2016	**Tan,** *ICML* 2019

DNN models getting **larger** and **deeper**

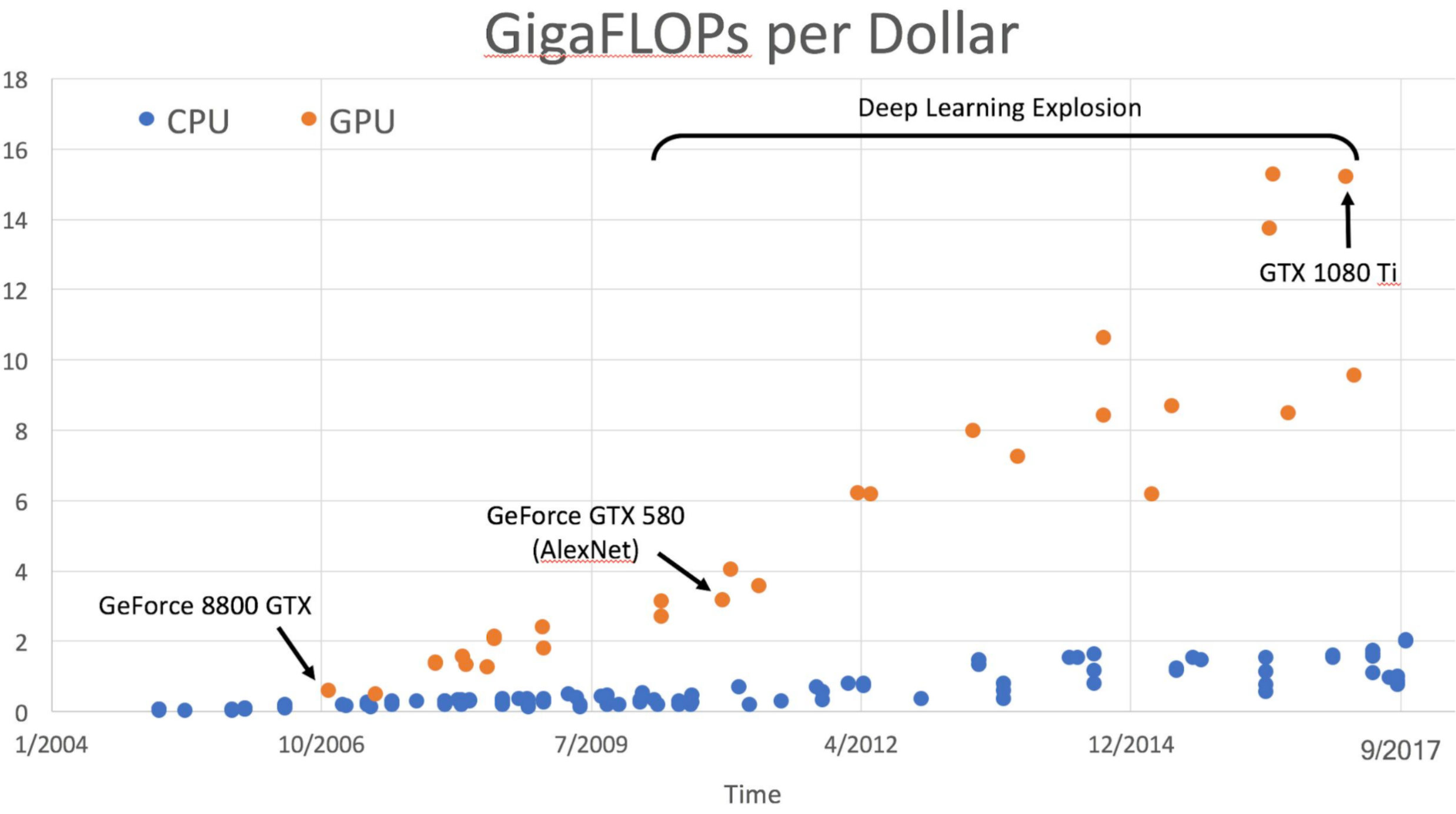
* Does not include multi-crop and ensemble
** Increase in FC layers due to squeeze-and-excitation layers (much smaller than FC layers for classification)

Hardware for ~~Machine~~ Deep Learning

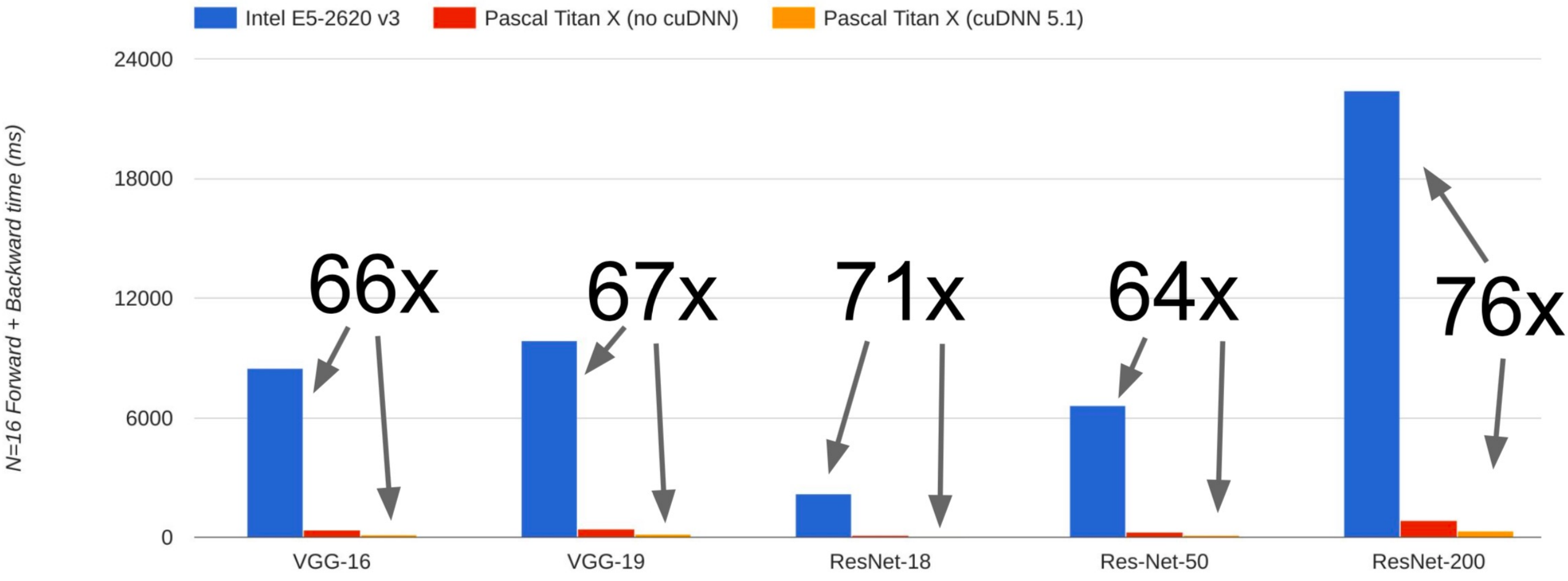
A computer for machine learning

- CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

- GPU: More cores, but each core is much slower and “dumber”; great for parallel tasks


GigaFLOPs per Dollar
CPU
GPU
Deep Learning Explosion
GTX 1080 Ti
GeForce GTX 580
(AlexNet)
GeForce 8800 GTX
18
16
14
12
10
8
6
4
2
0
1/2004
10/2006
7/2009
4/2012
12/2014
9/2017
Time

CPU vs GPU in practice



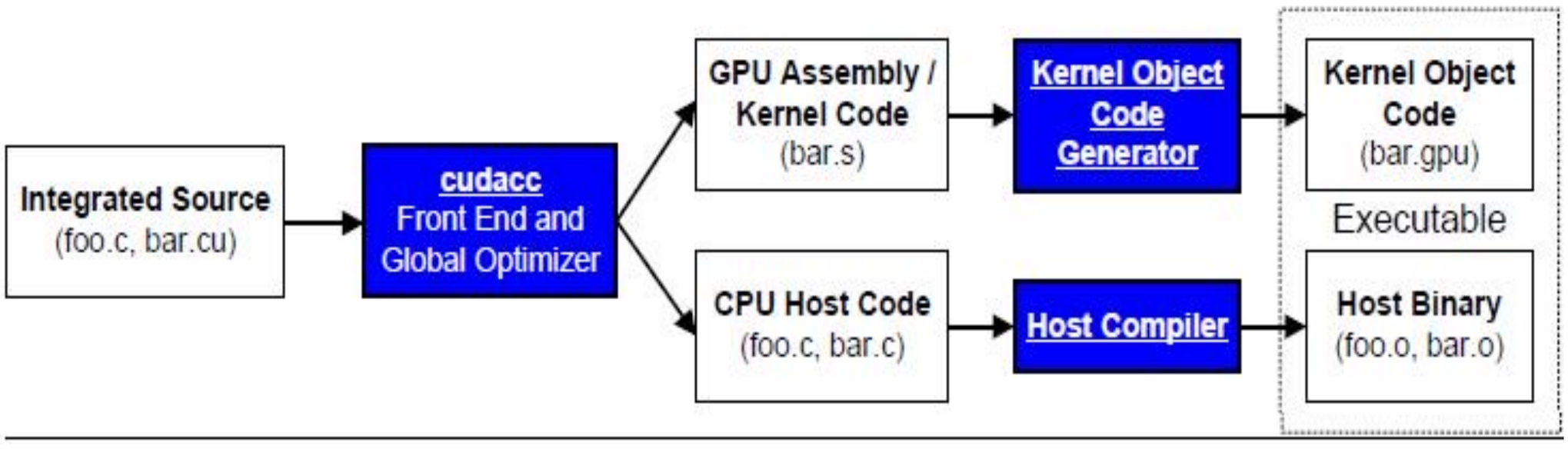
The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for [deep neural networks](#).

CUDA

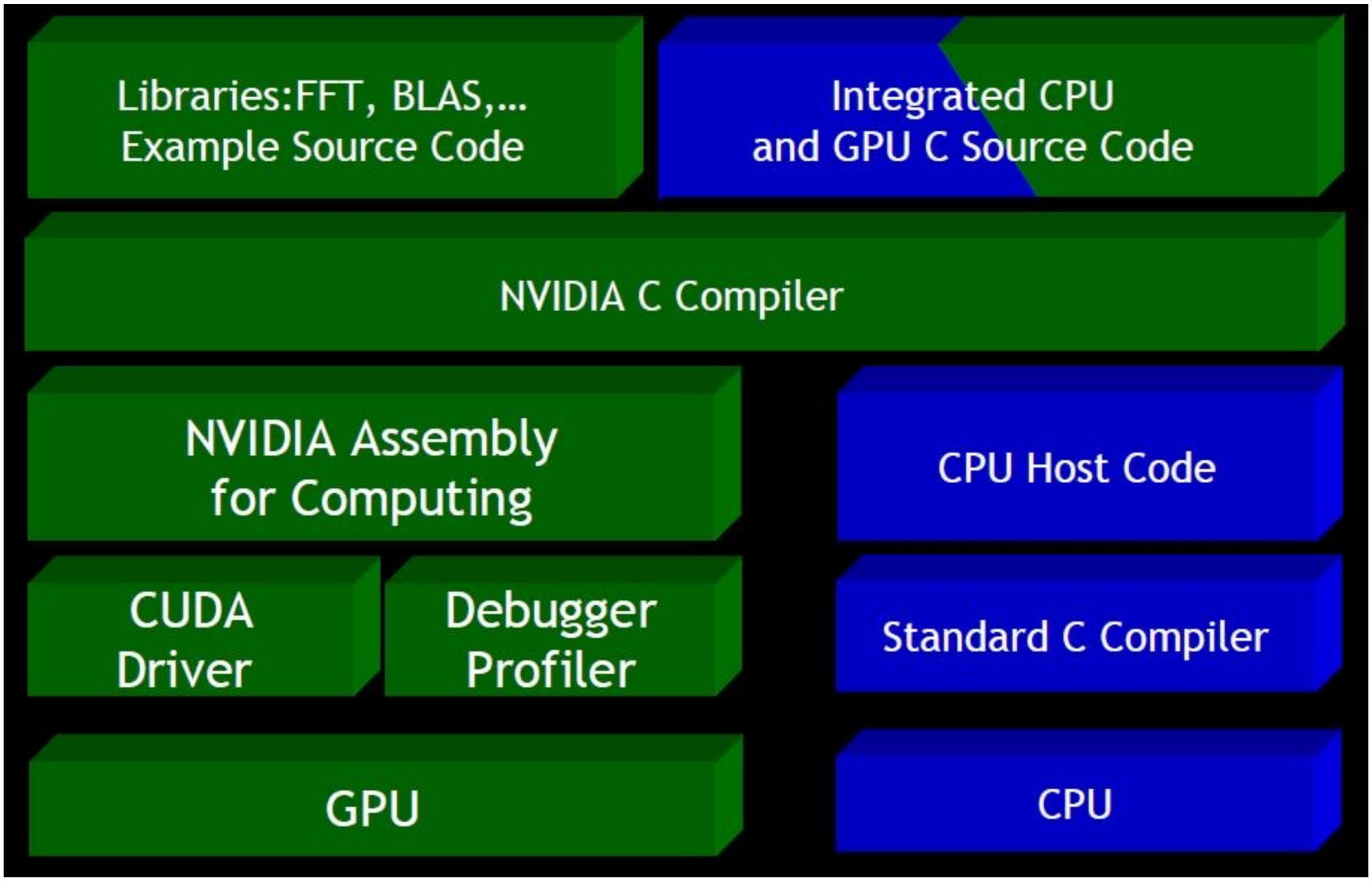
- CUDA is a platform for performing massively parallel computations on graphics accelerators
- CUDA was developed by NVIDIA
- It was first available with their G8X line of graphics cards
- Approximately 1 million CUDA capable GPUs are shipped every week
- CUDA presents a unique opportunity to develop widely-deployed parallel applications

CUDA Compilation

- As a programming model, CUDA is a set of extensions to ANSI C
- CPU code is compiled by the host C compiler and the GPU code (kernel) is compiled by the CUDA compiler. Separate binaries are produced



CUDA Stack


Libraries:FFT, BLAS,...
Example Source Code
Integrated CPU
and GPU C Source Code
NVIDIA C Compiler
NVIDIA Assembly
for Computing
CPU Host Code
CUDA
Driver
Debugger
Profiler
Standard C Compiler
GPU
CPU

Kernel Functions

- A kernel function is the basic unit of work within a CUDA thread
- Kernel functions are CUDA extensions to ANSI C that are compiled by the CUDA compiler and the object code generator

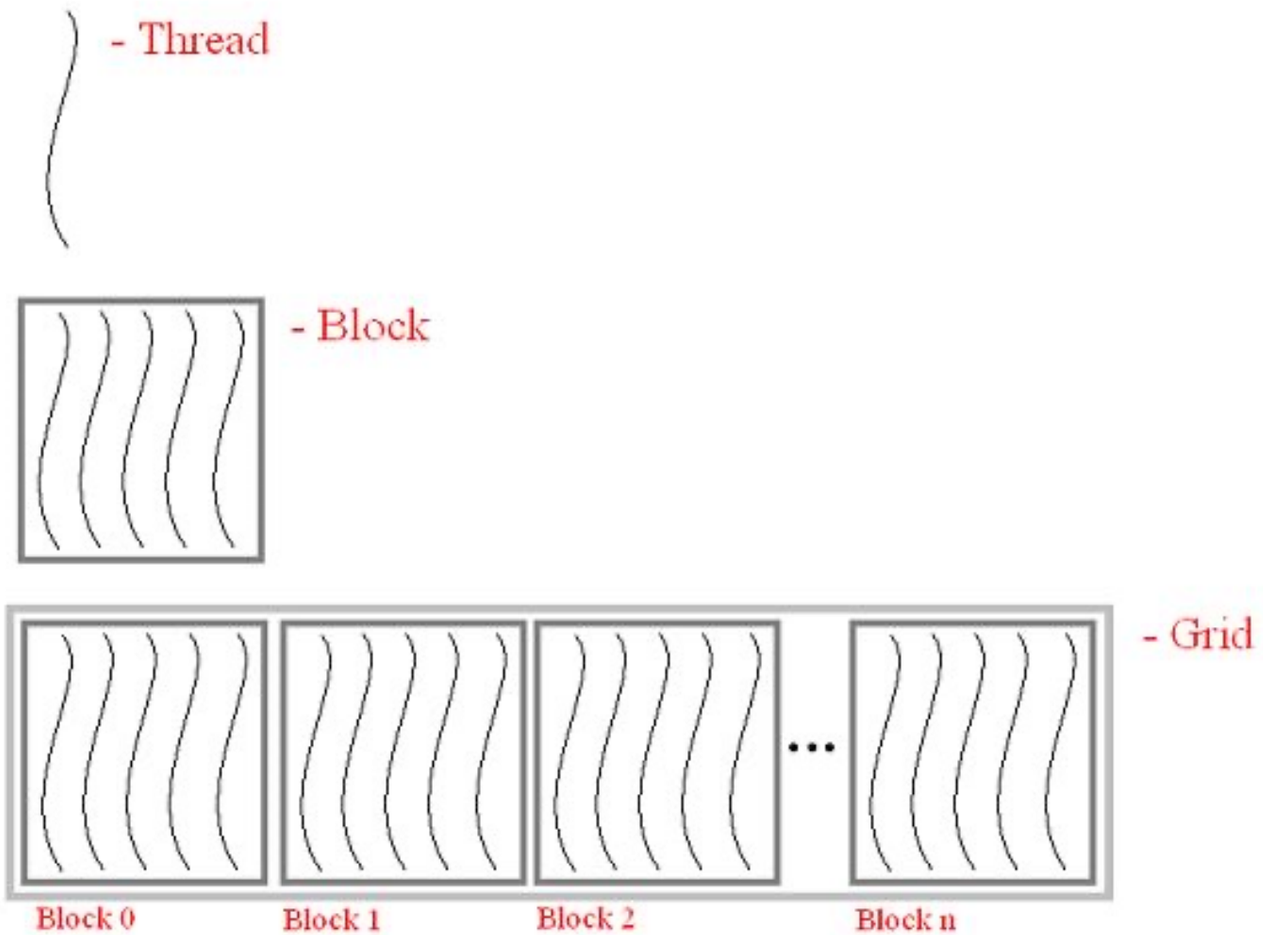
Kernel Limitations

- There must be no recursion; there's no call stack
- There must be no static variable declarations
- Functions must have a non-variable number of arguments

CUDA Warp

- CUDA utilizes SIMT (Single Instruction Multiple Thread)
- Warps are groups of 32 threads. Each warp receives a single instruction and “broadcasts” it to all of its threads.
- CUDA provides “zero-overhead” warp and thread scheduling. Also, the overhead of thread creation is on the order of 1 clock.
- Because a warp receives a single instruction, it will diverge and converge as each thread branches independently

Thread Hierarchy

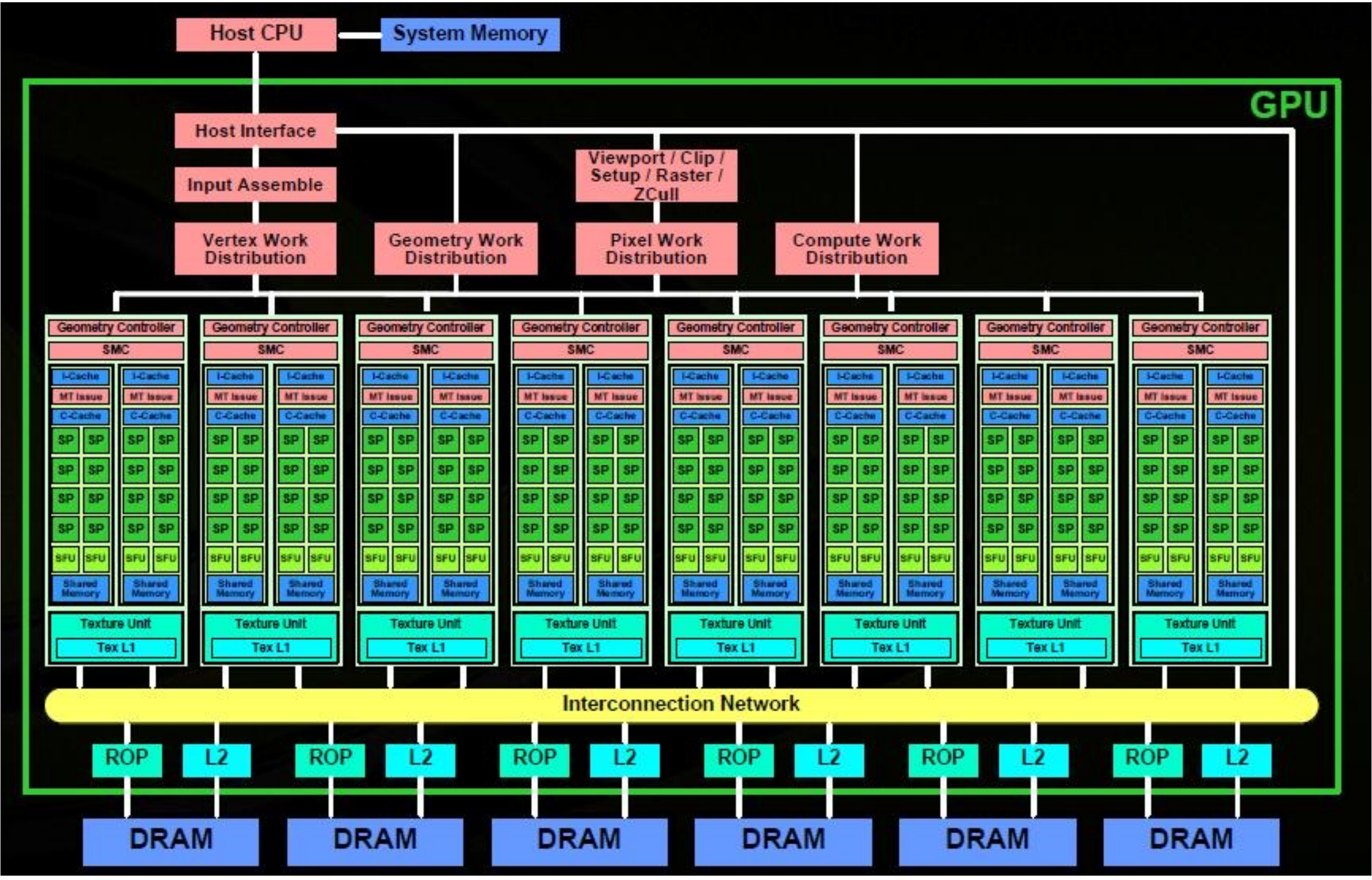


Thread – Distributed by the CUDA runtime **(identified by threadIdx)**

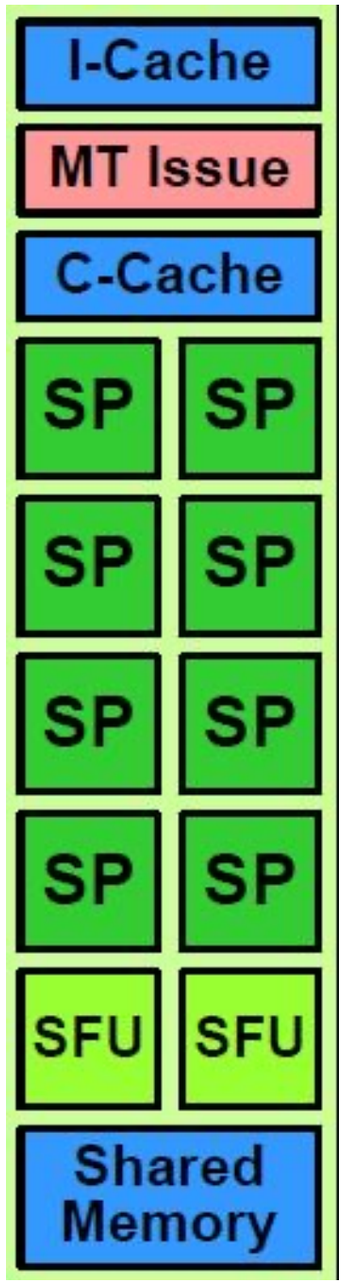
Warp – A scheduling unit of up to 32 threads

Block – A user defined group of 1 to 512 threads. **(identified by blockIdx)**

Grid – A group of one or more blocks. A grid is created for each CUDA kernel function


Host CPU
System Memory
GPU
Host Interface
Input Assemble
Viewport / Clip / Setup / Raster / ZCull
Vertex Work Distribution
Geometry Work Distribution
Pixel Work Distribution
Compute Work Distribution
Geometry Controller
SMC
I-Cache
MT Issue
C-Cache
SP
SFU
Shared Memory
Texture Unit
Tex L1
Interconnection Network
ROP
L2
DRAM

Streaming Multiprocessor (SM)



- Each SM has 8 Scalar Processors (SP)

- IEEE 754 32-bit floating point support (incomplete support)

- Each SP is a 1.35 GHz processor (32 GFLOPS peak)

- Supports 32 and 64 bit integers

- 8,192 dynamically partitioned 32-bit registers

- Supports 768 threads in hardware (24 SIMT warps of 32 threads)

- Thread scheduling done in hardware

- 16KB of low-latency shared memory

- 2 Special Function Units (reciprocal square root, trig functions, etc)

NVIDIA GeForce 8800GTX GPU has 16 SMs

Scalar Processor

- Supports 32-bit IEEE floating point instructions:
  FADD, FMAD, FMIN, FMAX, FSET, F2I, I2F
- Supports 32-bit integer operations
  IADD, IMUL24, IMAD24, IMIN, IMAX, ISET, I2I, SHR, SHL, AND, OR, XOR

- Fully pipelined

Code Example

-SAXPY in C

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

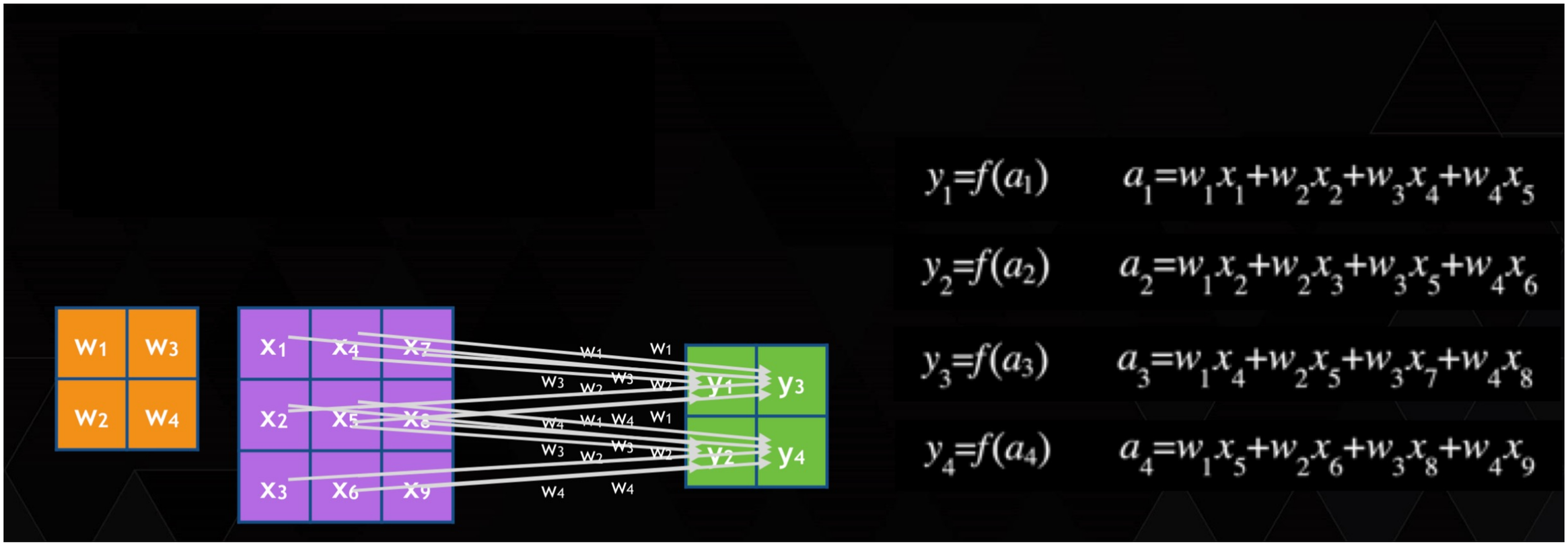
-SAXPY in CUDA

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
```

```
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

calls the kernel, jumps to GPU

Convolution layer

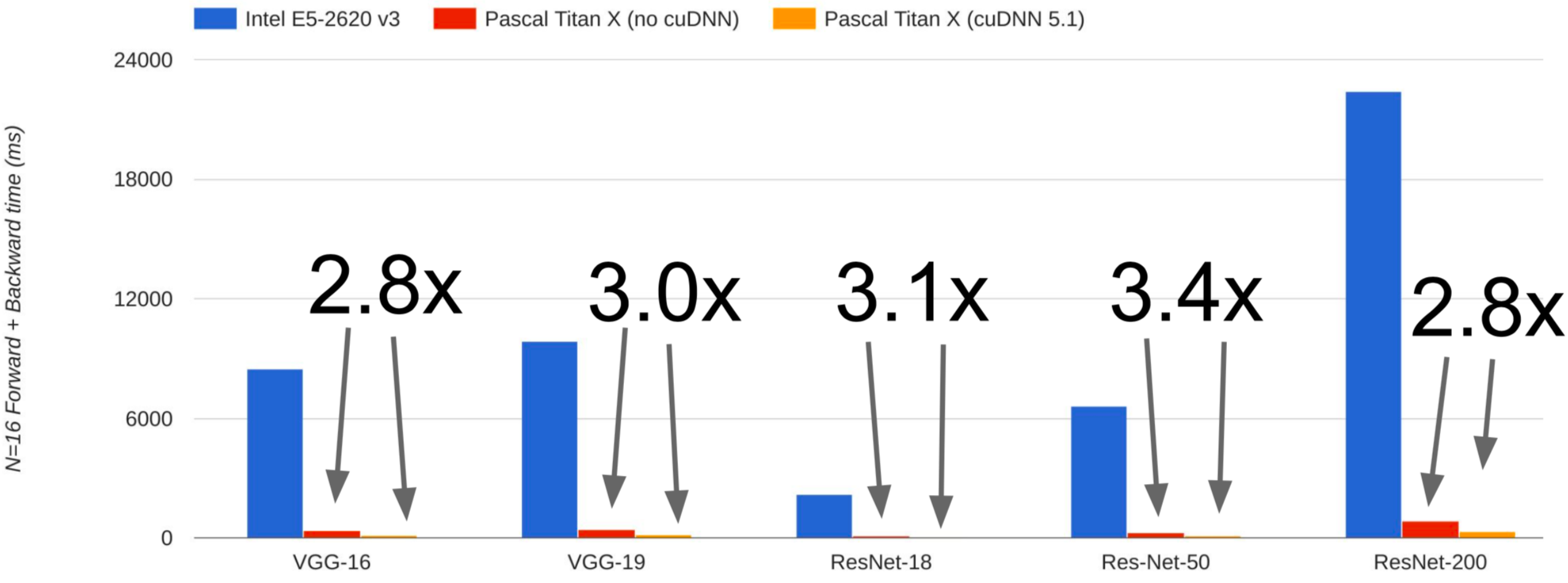


cuDNN

- Low-level Library of GPU-accelerated routines; similar in intent to BLAS
- Out-of-the-box speedup of Neural Networks
- Developed and maintained by NVIDIA
- Optimized for current and future NVIDIA GPU generations

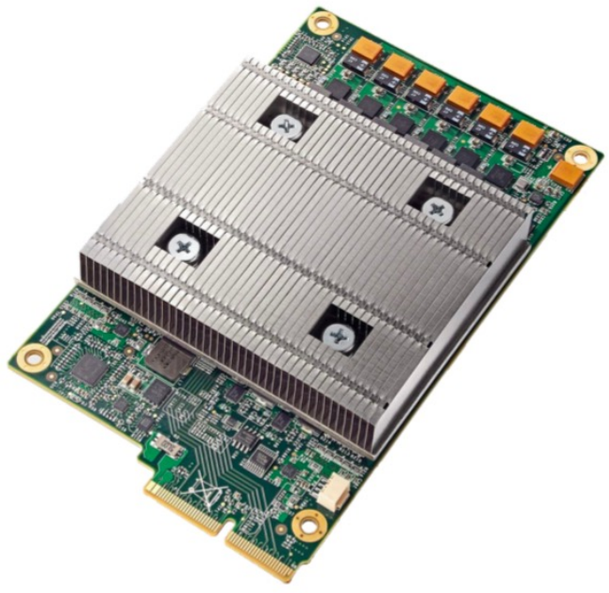
- Flexible API : arbitrary dimension ordering, striding, and sub-regions for 4d tensors
- Less memory, more performance : Efficient forward and backward convolution routines with zero memory overhead
- Easy Integration : black box implementation of convolution and other routines – ReLu, Sigmoid, Tanh, Pooling, Softmax

CPU vs GPU in practice



TPU

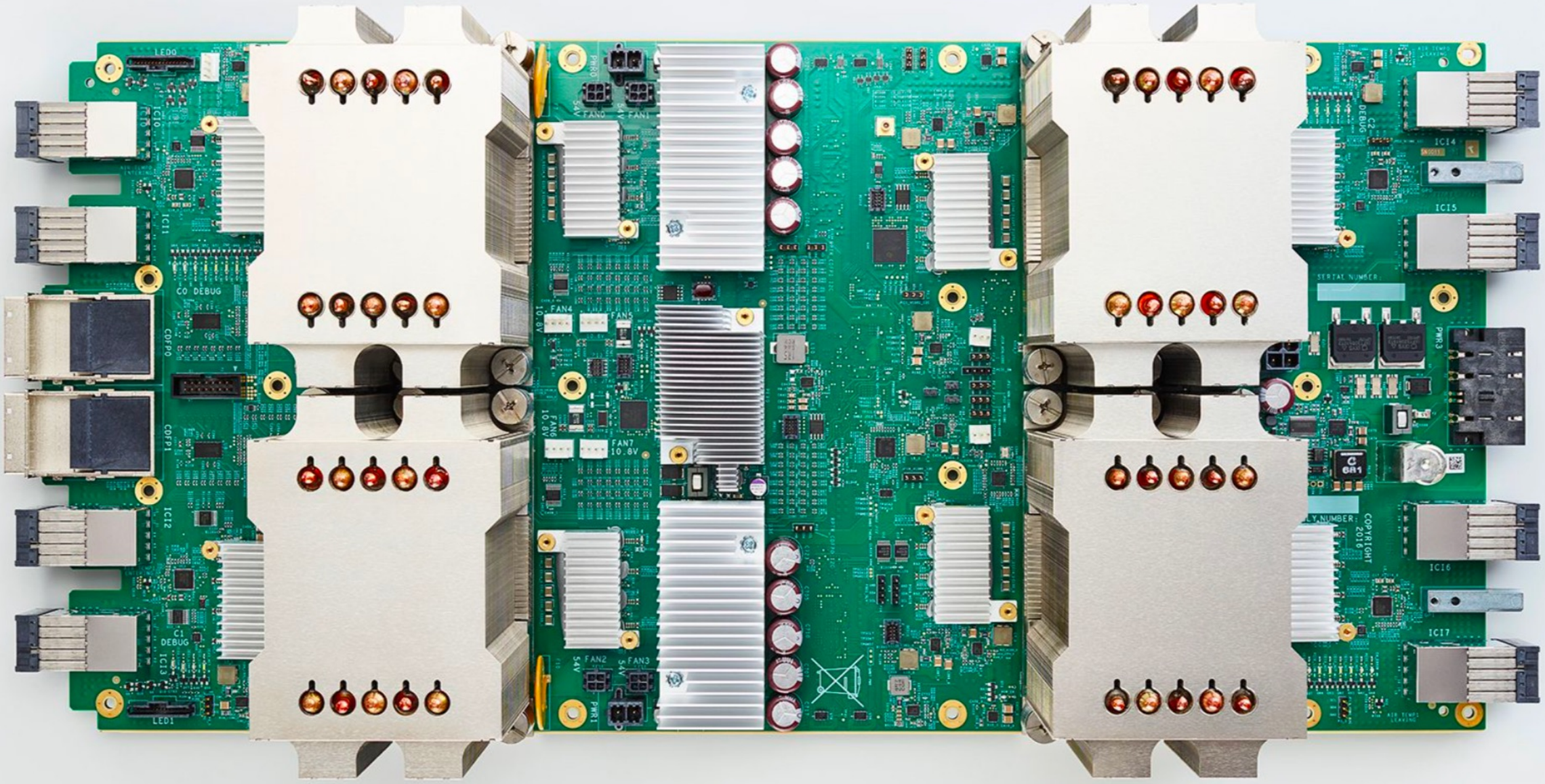
Google-designed chip for neural net **inference**



In production use for ~36 months: used on search queries, for neural machine translation, for speech, for image recognition, for AlphaGo match, …

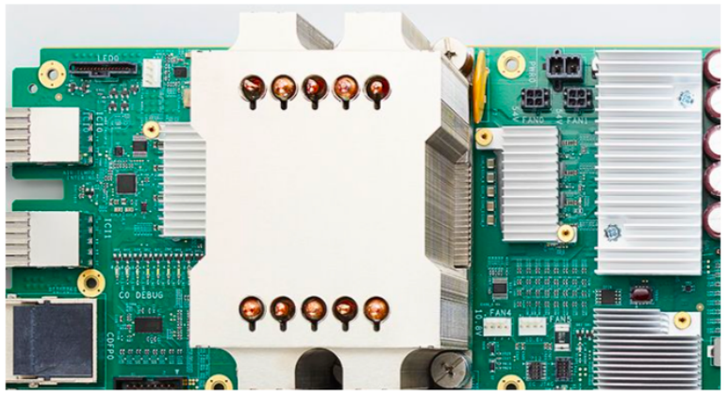
In-Datacenter Performance Analysis of a Tensor Processing Unit, Jouppi, Young, Patil, Patterson et al., ISCA 2017, arxiv.org/abs/1704.04760

TPU v2

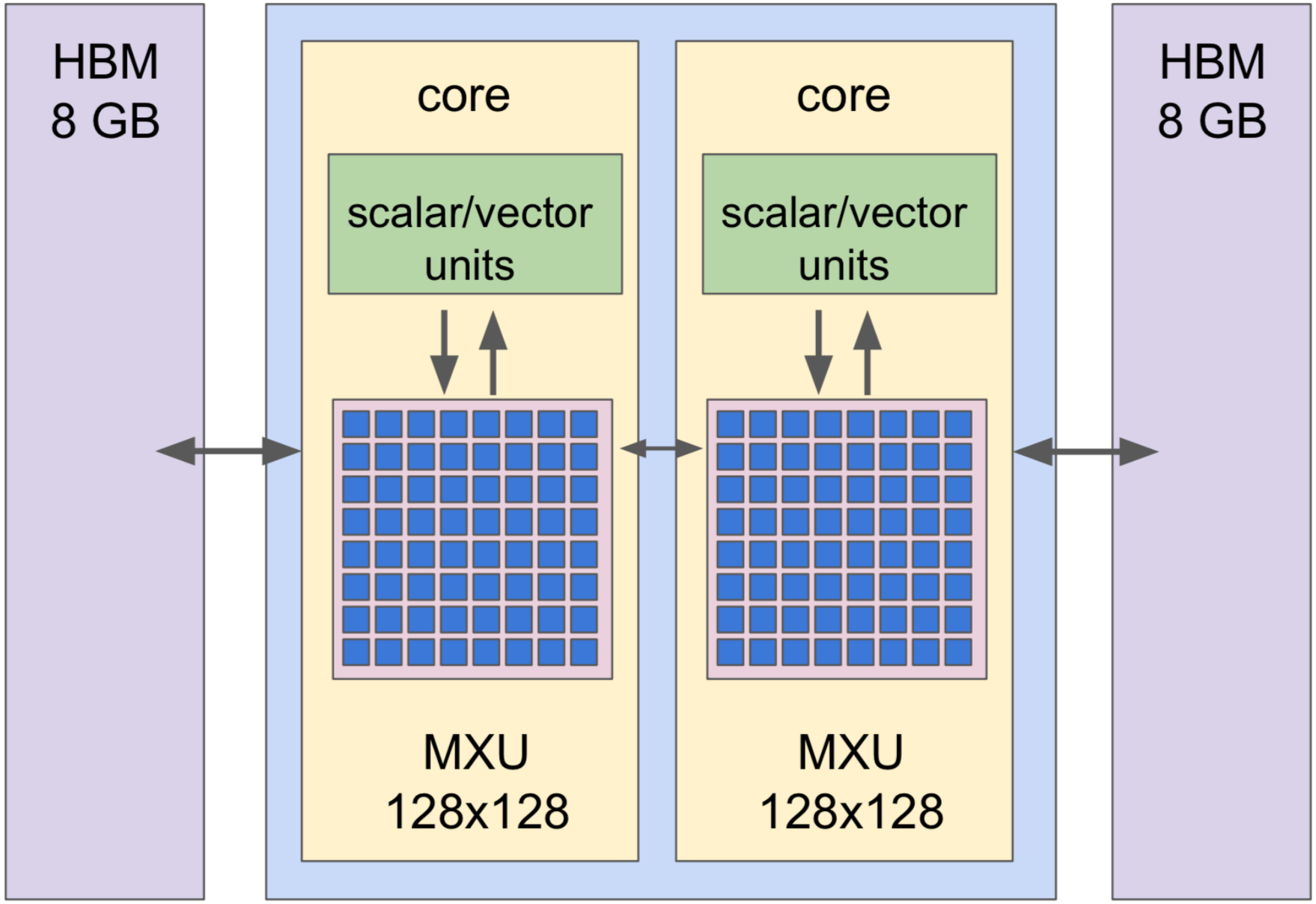


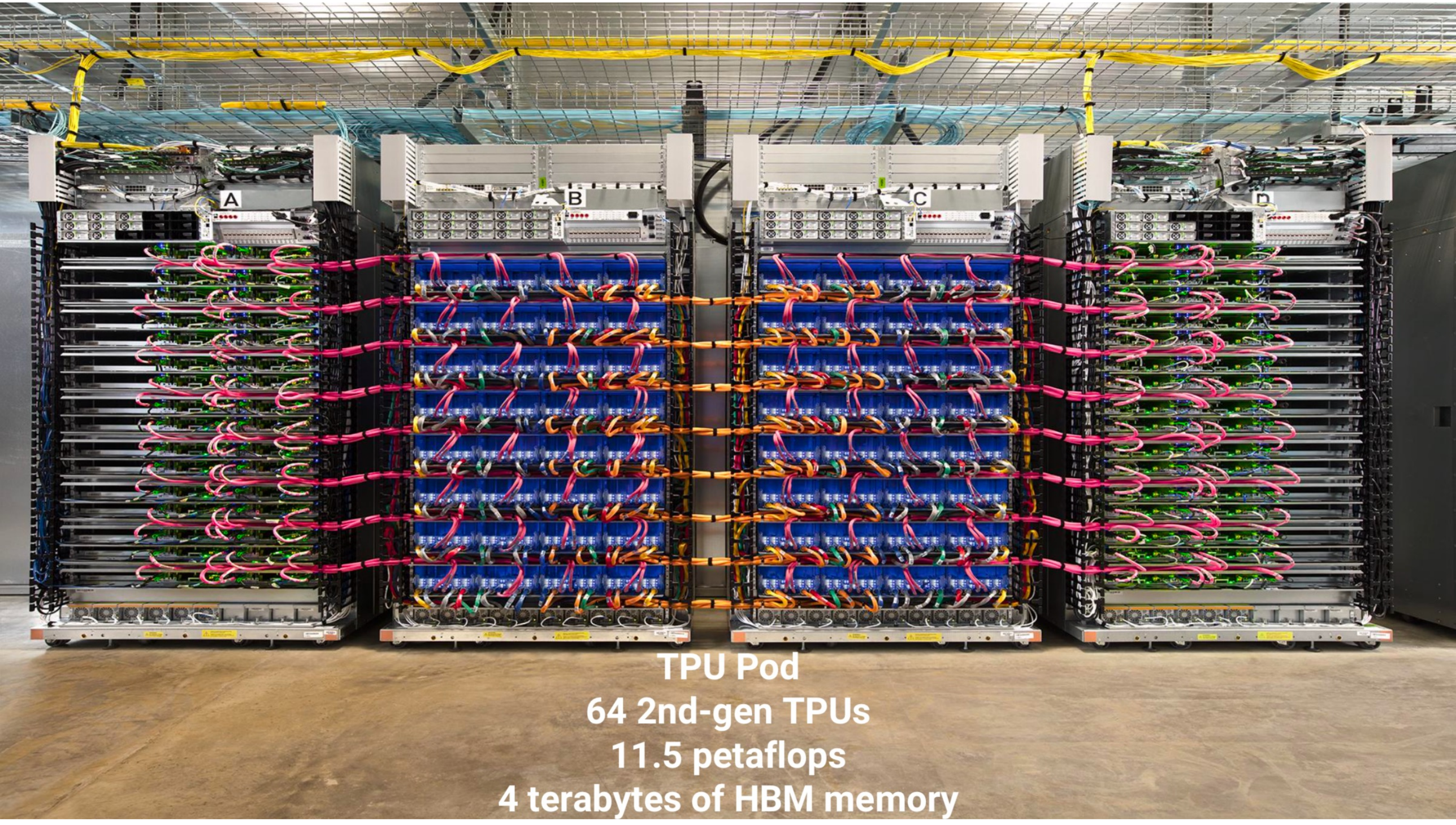
Google-designed device for neural net **training** and **inference**

TPUv2 Chip



- 16 GB of HBM
- 600 GB/s mem BW
- Scalar/vector units: 32b float
- MXU: 32b float accumulation but reduced precision for multipliers
- 45 TFLOPS




A
B
C
D
TPU Pod
64 2nd-gen TPUs
11.5 petaflops
4 terabytes of HBM memory