Designing Hardware Accelerators for Deep Neural Networks

Manish Pandey

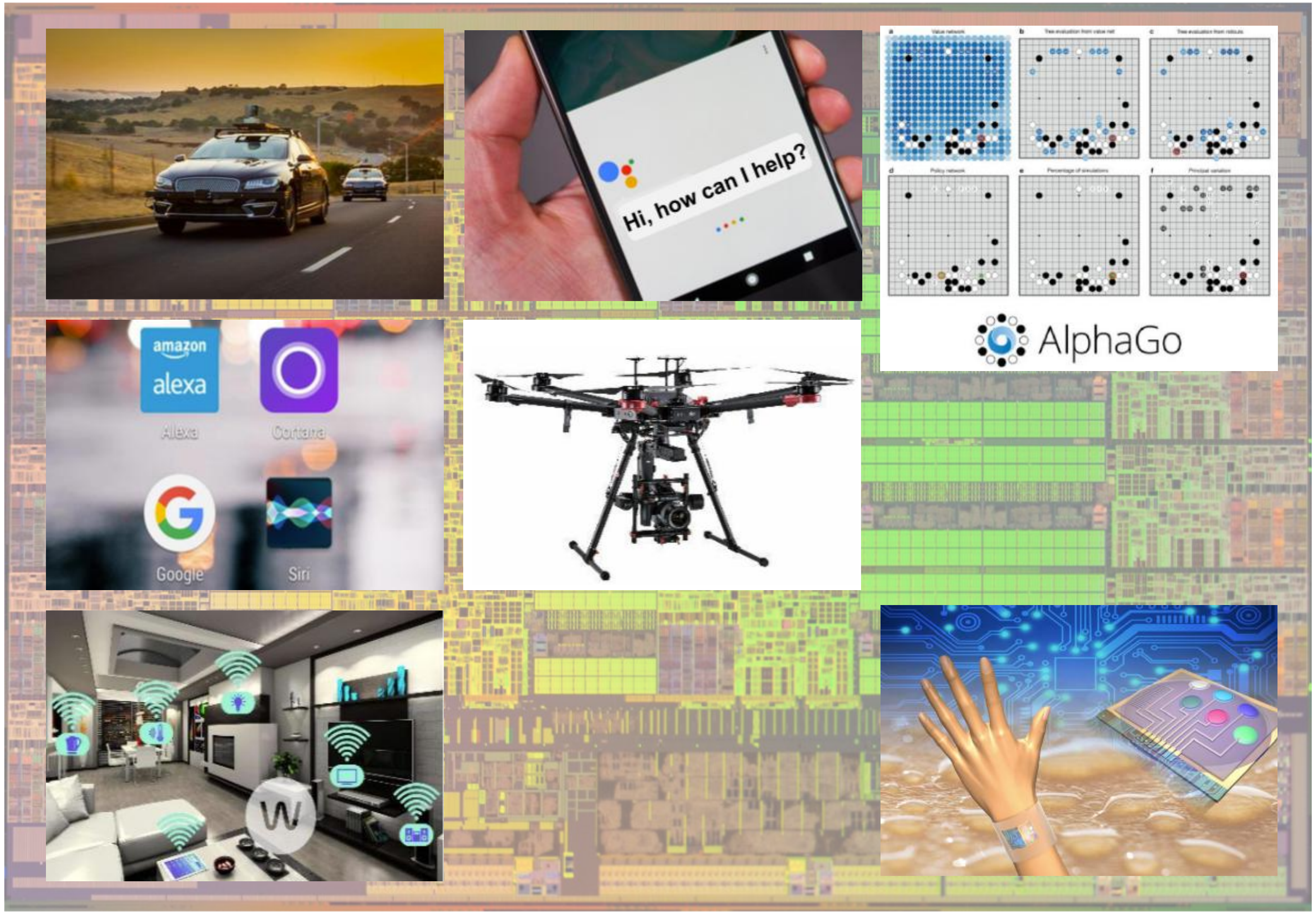
Synopsys, Inc., Mountain View, CA

Jan 4, 2020

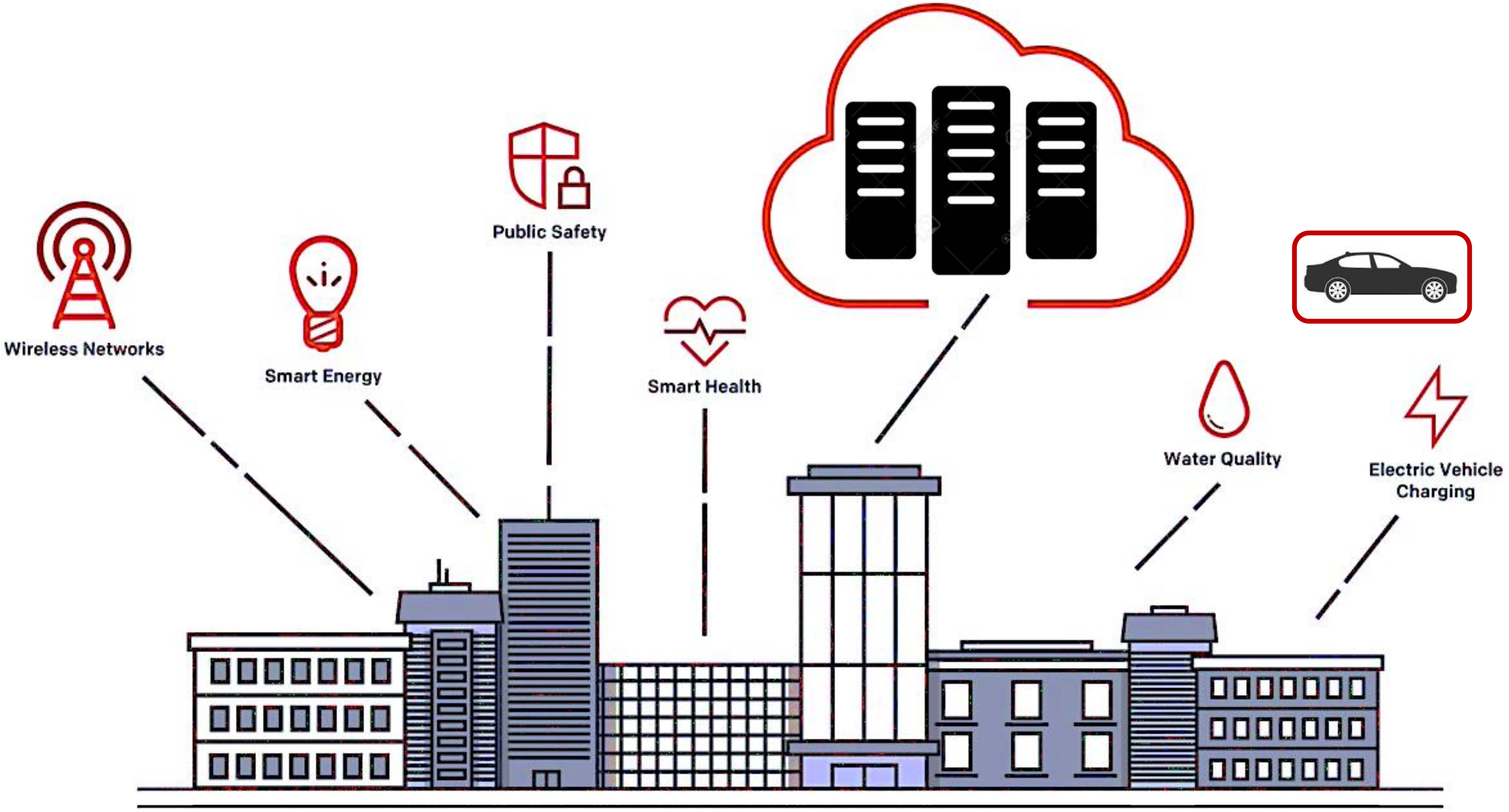
Roadmap

- Background and Introduction
- Co-Design for DNN Model Compression
- Quantization and Number Representations
- Parallel Architectures and Memory Subsystem
- Case Studies

Revolution in Artificial Intelligence

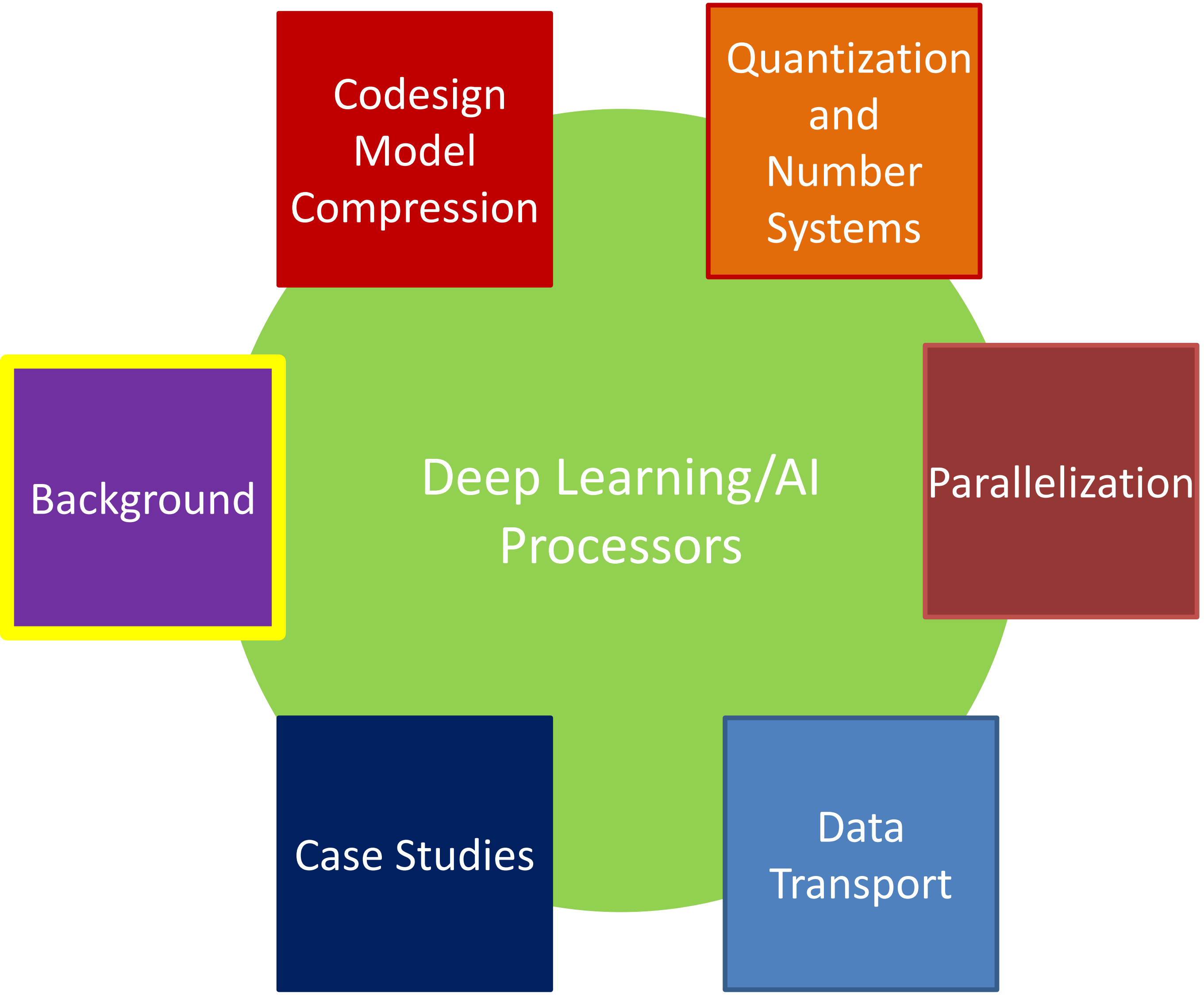


Intelligence Infused from the Edge to the Cloud

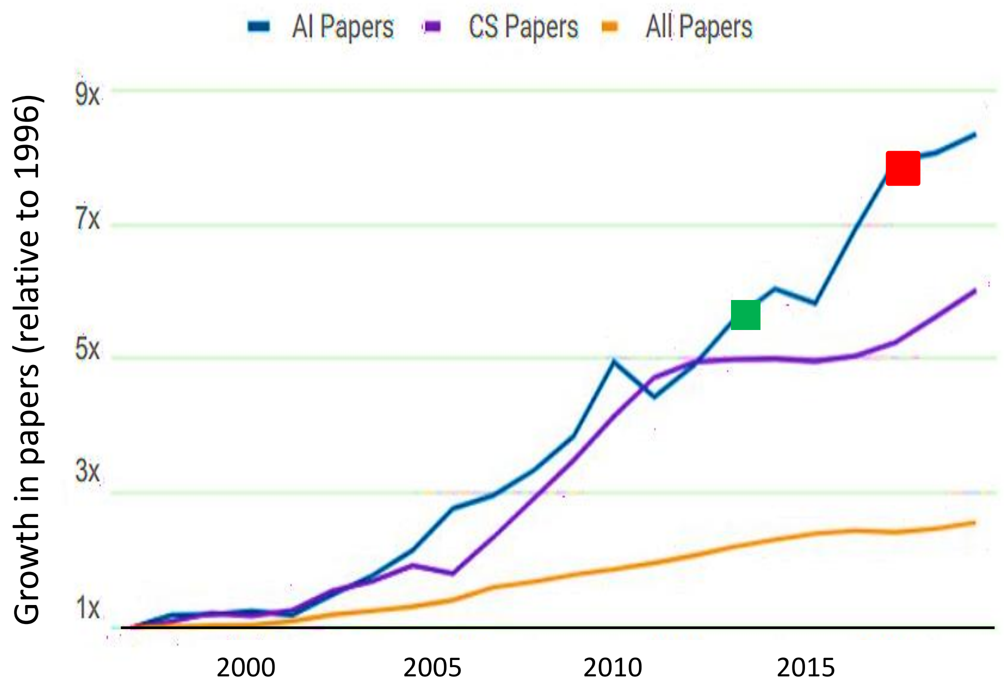


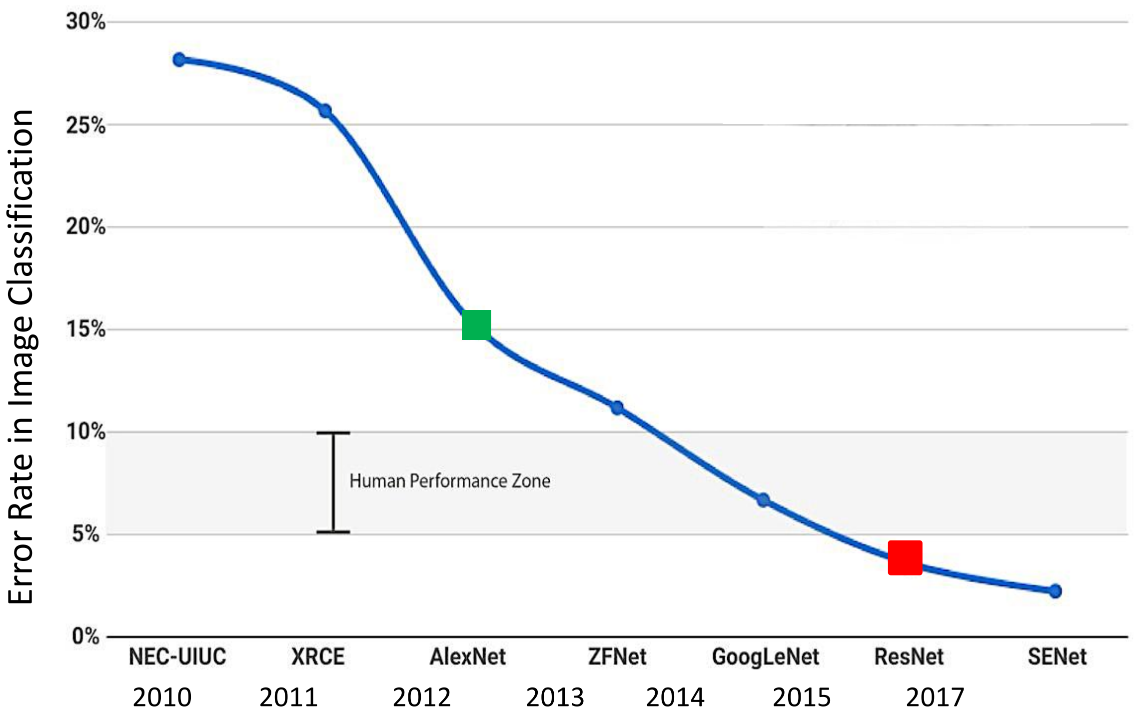
This Tutorial

- DNN Primer and specialization for DNNs
  - The underlying technical basis driving DNN processors
- Specialization
  - Co-Design for DNN Model Compression
  - Eliminate unneeded accuracy
    - IEEE replaced by lower precision FP
    - 16/8 bit integers, low precision and binary representations
  - Parallelism and memory systems
    - Exploiting parallelism of models
    - Overcoming memory systems bottleneck
- Putting it together
  - CPUs, GPUs, ASICs, FPGAs


Codesign Model Compression
Quantization and Number Systems
Background
Deep Learning/AI Processors
Parallelization
Case Studies
Data Transport

Advances in AI/ML





Breakthrough Performance across Domains

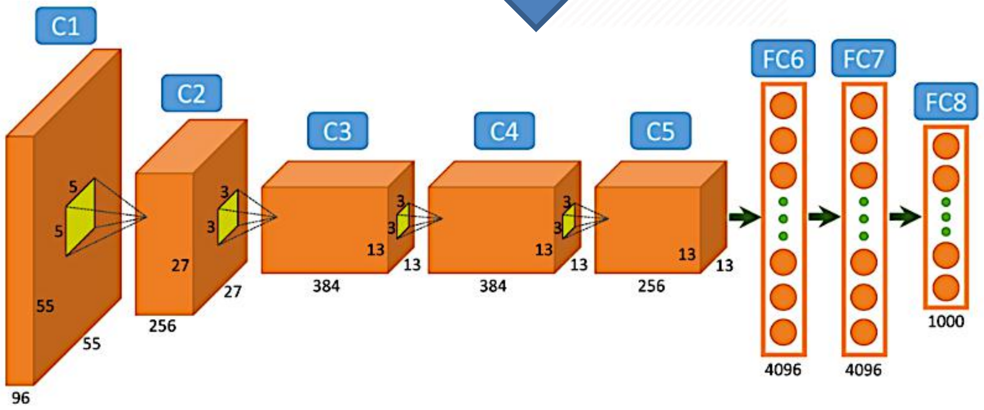
NLP Performance Breakthrough

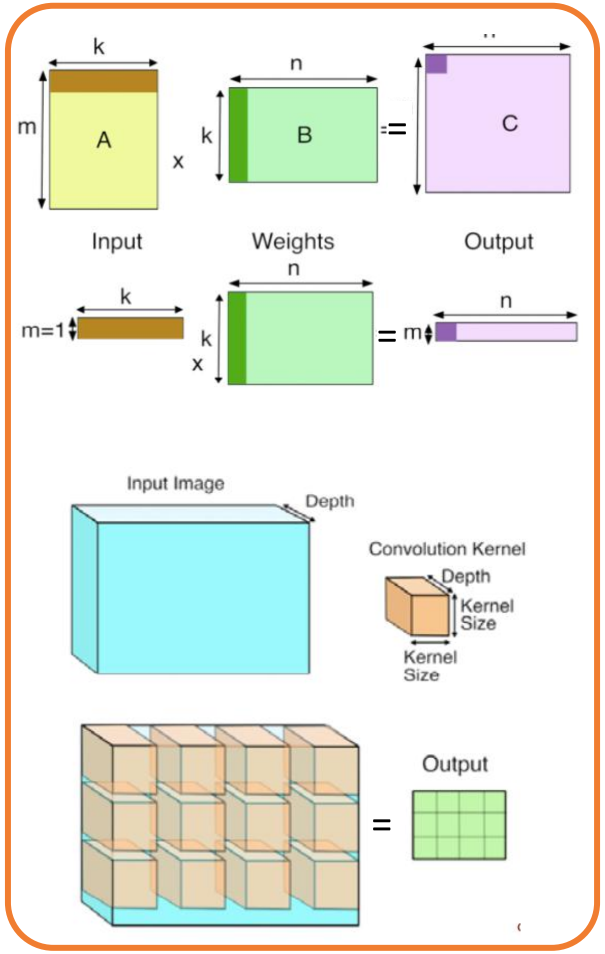
SQuAD2.0
The Stanford Question Answering Dataset

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Mar 20, 2019	BERT + DAE + AoA (ensemble) *Joint Laboratory of HIT and iFLYTEK Research*	**87.147**	**89.474**
2 Mar 15, 2019	BERT + ConvLSTM + MTL + Verifier (ensemble) *Layer 6 AI*	86.730	89.286
3 Mar 05, 2019	BERT + N-Gram Masking + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	86.673	89.147

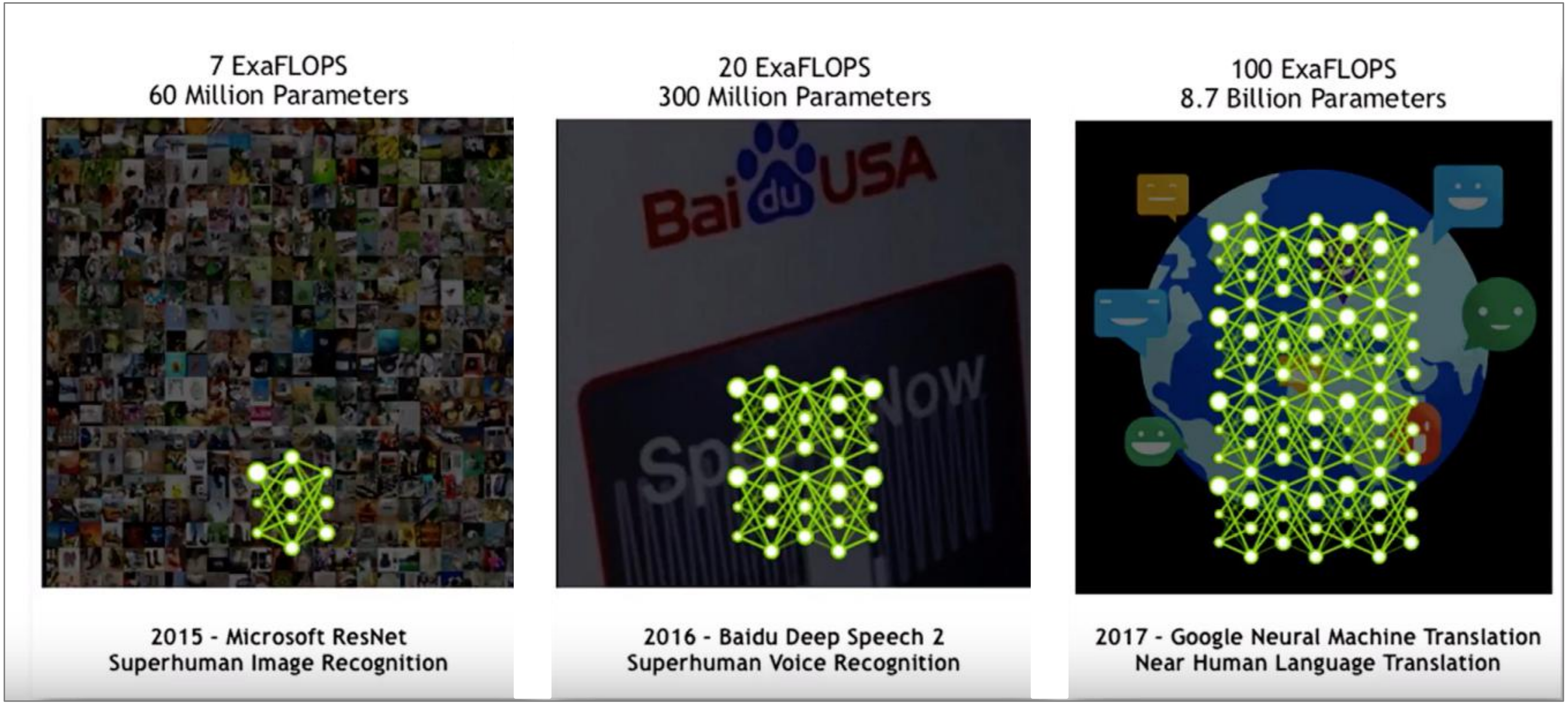
AI driven by advances in hardware

Chiwawa





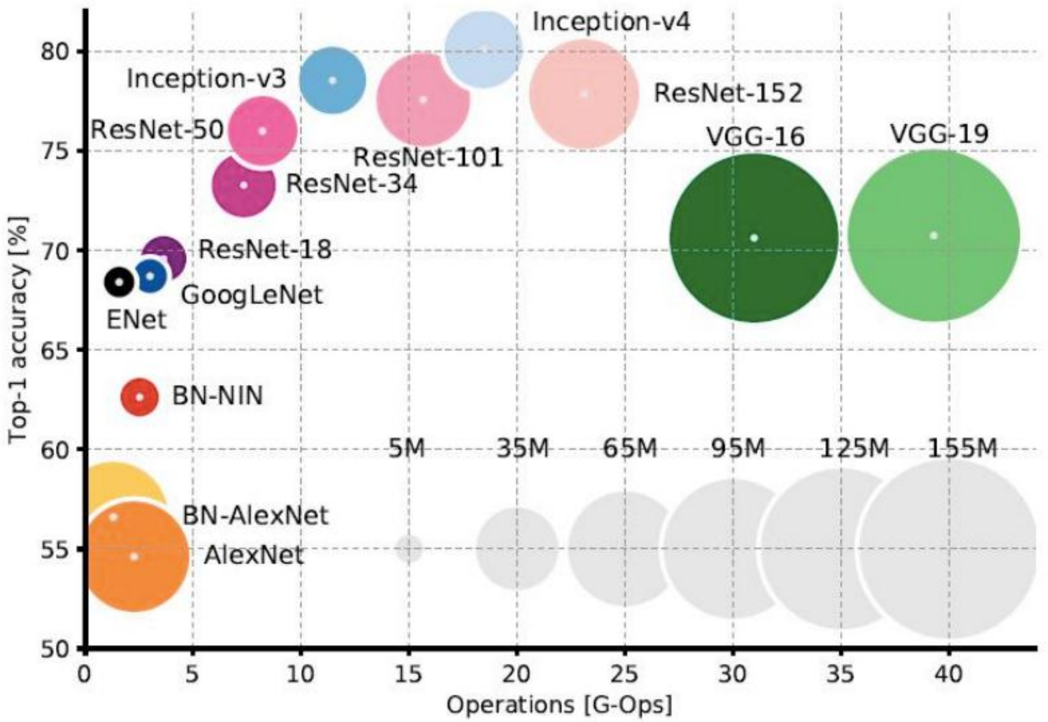
Deep Learning Advances Gated by Hardware

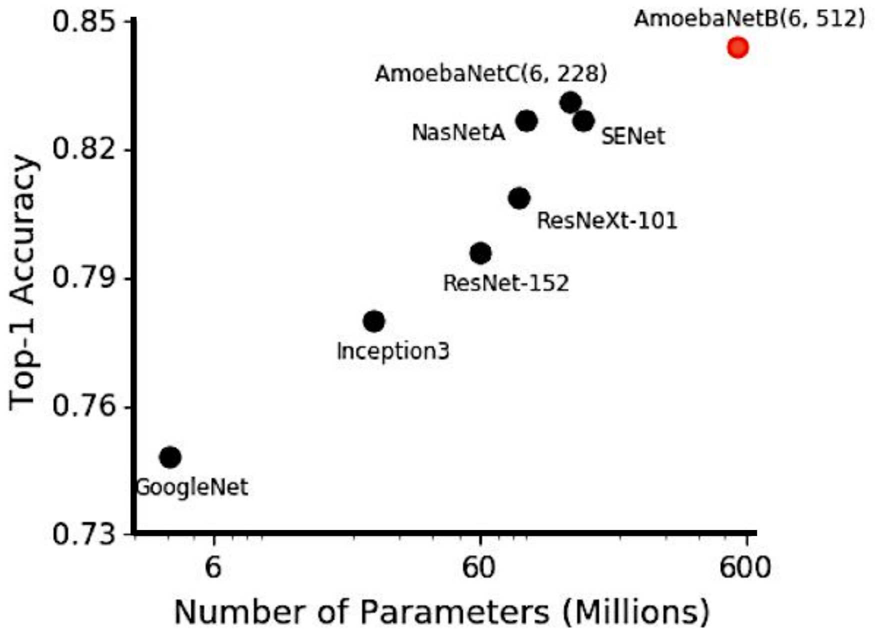


[Bill Dally, SysML 2018]

Deep Learning Advances Gated by Hardware

- Results improve with
  - Larger Models
  - Larger Datasets





NLP Transformers

https://talktotransformer.com/

Custom prompt

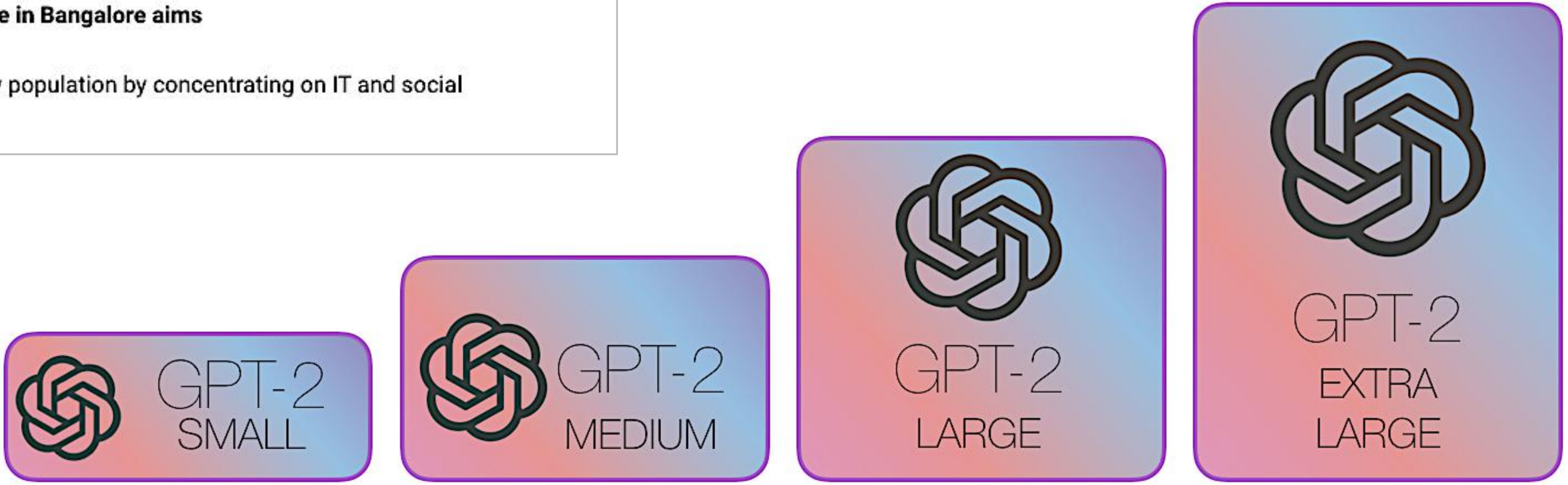
VLSID 2020 Conference in Bangalore aims

GENERATE ANOTHER

Completion

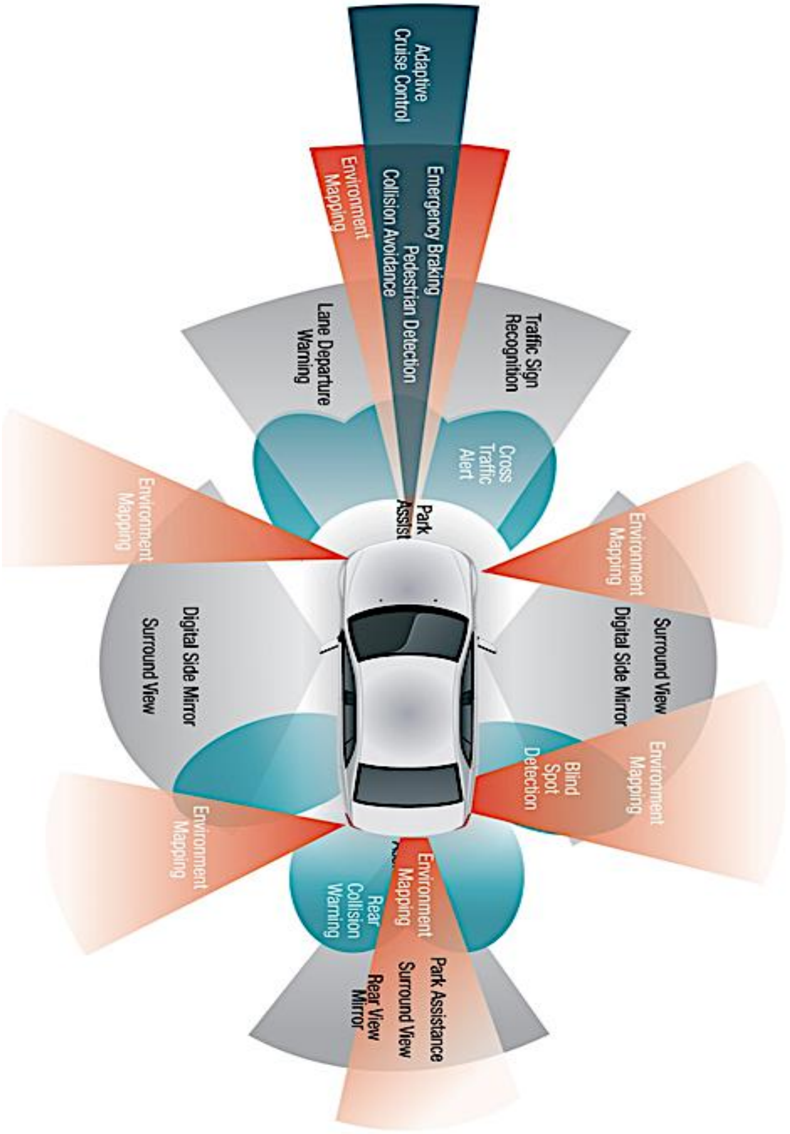
VLSID 2020 Conference in Bangalore aims

to reach out to this new population by concentrating on IT and social challenges associated



117M Parameters | 345M Parameters | 762M Parameters | 1,542M Parameters

How Much Compute?



12 HD Cameras

Inferencing**

- 25 Million Weights
- 300 Gops for HD Image
- 9.4 Tops for 30fps
- 12 Cameras, 3 nets = 338 Tops

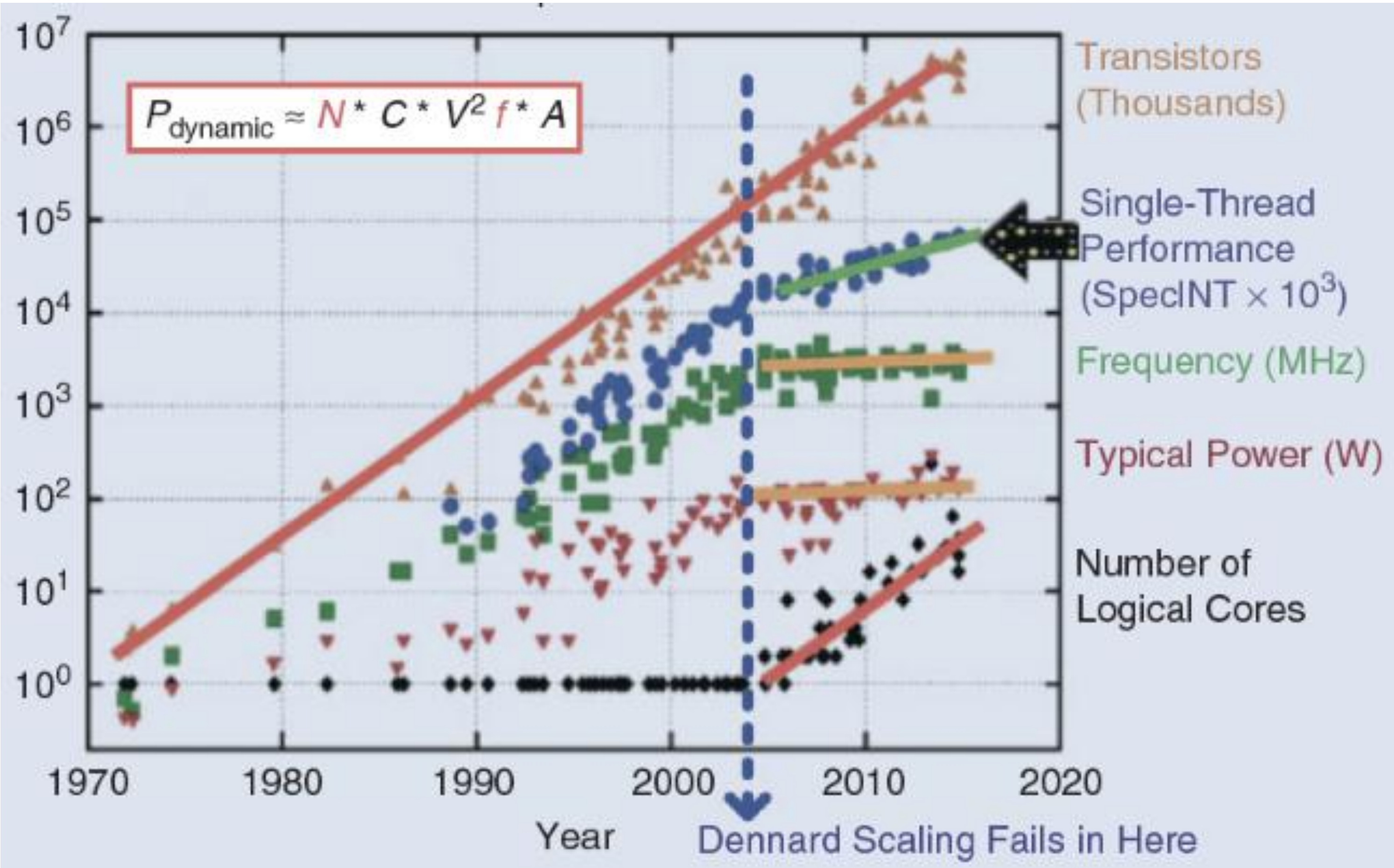
Training

- 30Tops x 10^8 (train set) x $10^{\wedge 2}$ (epochs) = 10^{23} Ops

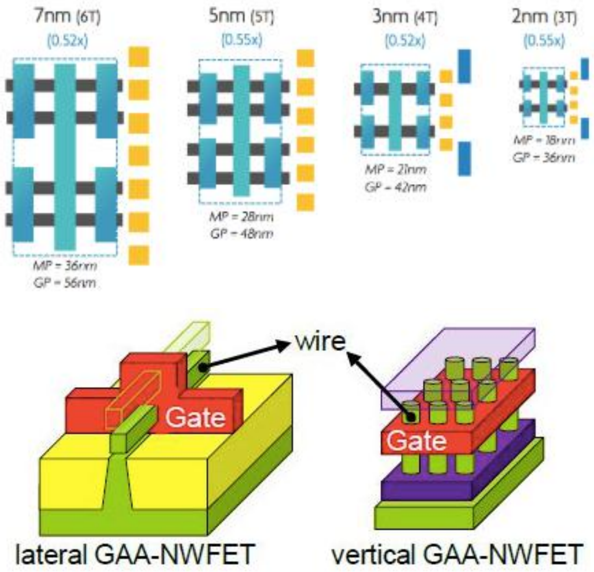
**ResNet-50

[Bill Dally, SysML 2018]

Challenge: End of Line for CMOS Scaling



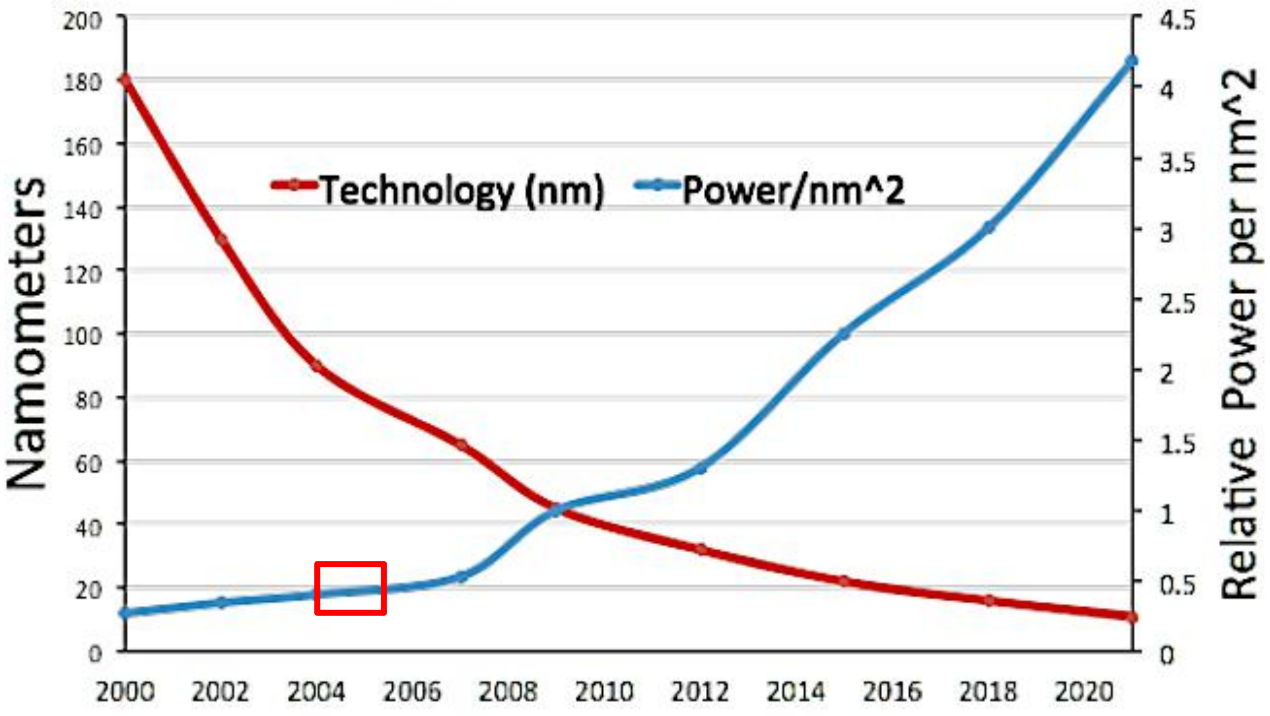
- Device scaling down slowing



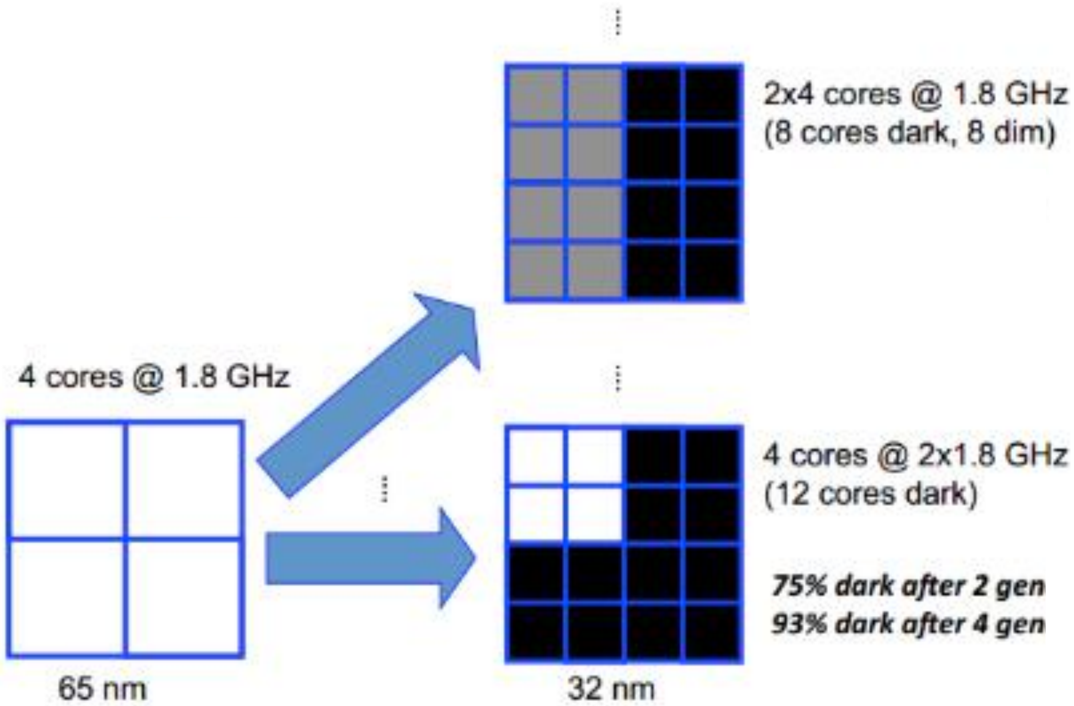
- Power Density stopped scaling in 2005

[Olukotun, Horowitz] [IMEC]

Dennard Scaling to Dark Silicon



Transistor property	Dennard	Post-Dennard
Δ Quantity	S^2	S^2
Δ Frequency	S	S
Δ Capacitance	$1/S$	$1/S$
V_{DD}^2	$1/S^2$	1
$\Rightarrow \Delta$ Power $= \Delta$ $QFCV^2$	1	$\boxed{S^2}$
$\Rightarrow \Delta$ Utilization $= 1/$Power	1	$\boxed{1/S^2}$

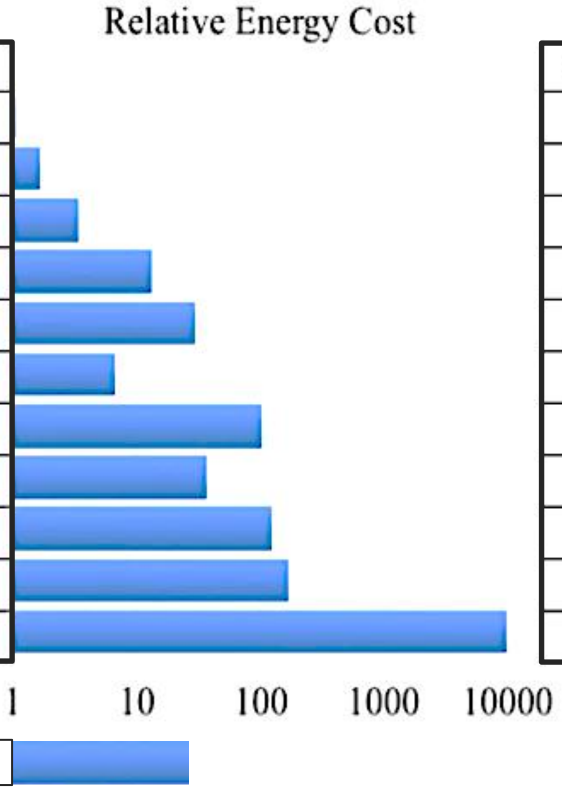


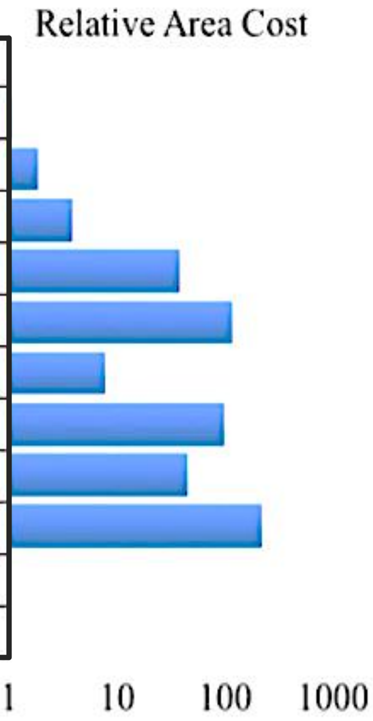
Can we specialize designs for DNNs?

Taylor, "A landscape of the new dark silicon design regime." IEEE Micro 33.5 (2013): 8-19

Energy Cost for Operations

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





Instruction Fetch/D	70

*45nm

[Horowitz ISSCC 2014]

Energy Cost for DNN Ops

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

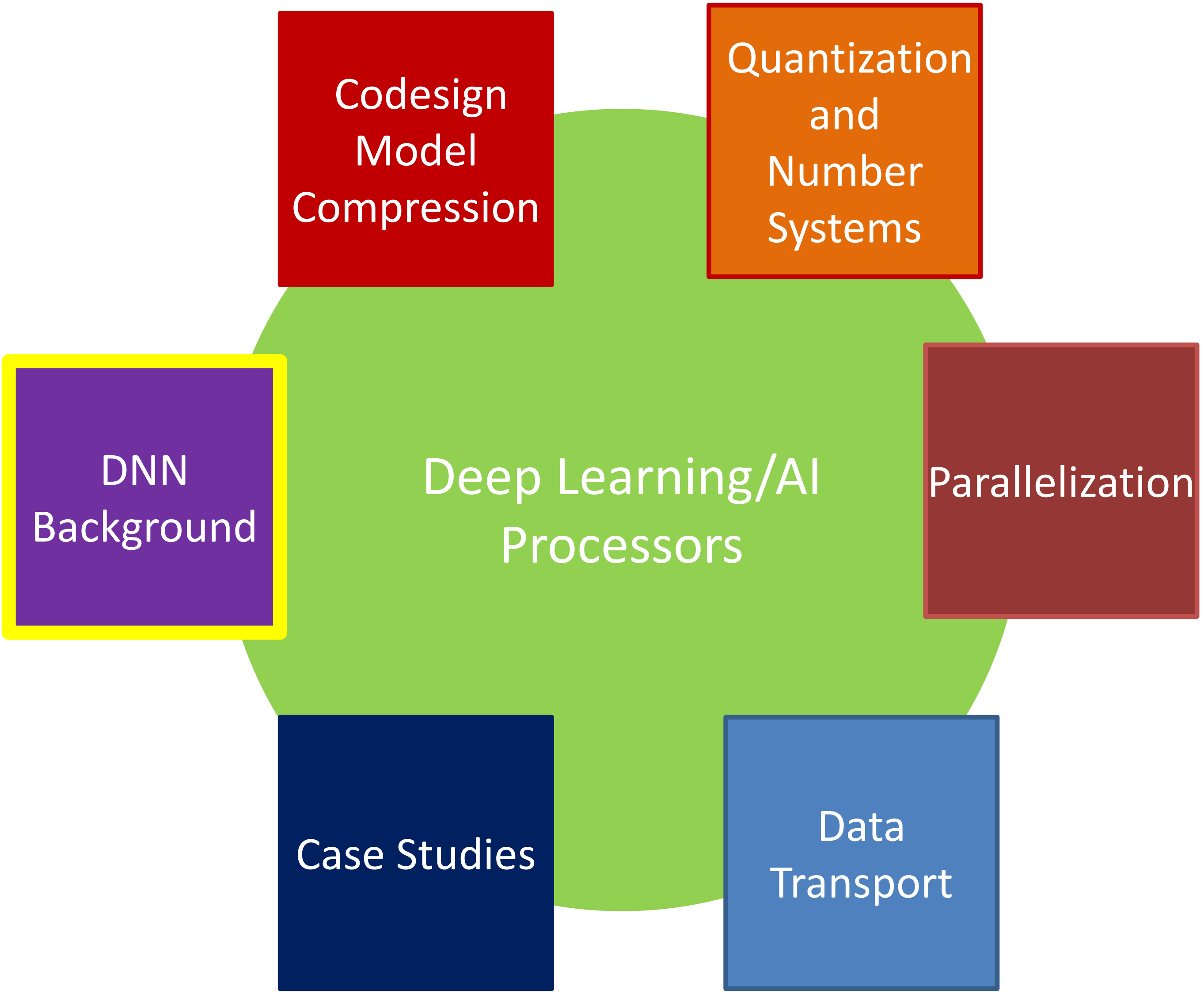
$$Y = AB + C$$

Instruction	Data	Type	Energy	Energy/Op
1 Op/Instr (*,+)	Memory	fp32	89 nJ	693 pJ
128 Ops/Instr (AX+B)	Memory	fp32	72 nJ	562 pJ
128 Ops/Instr (AX+B)	Cache	fp32	0.87 nJ	6.82 pJ
128 Ops/Instr (AX+B)	Cache	fp16	0.47 nJ	3.68 pJ

*45nm

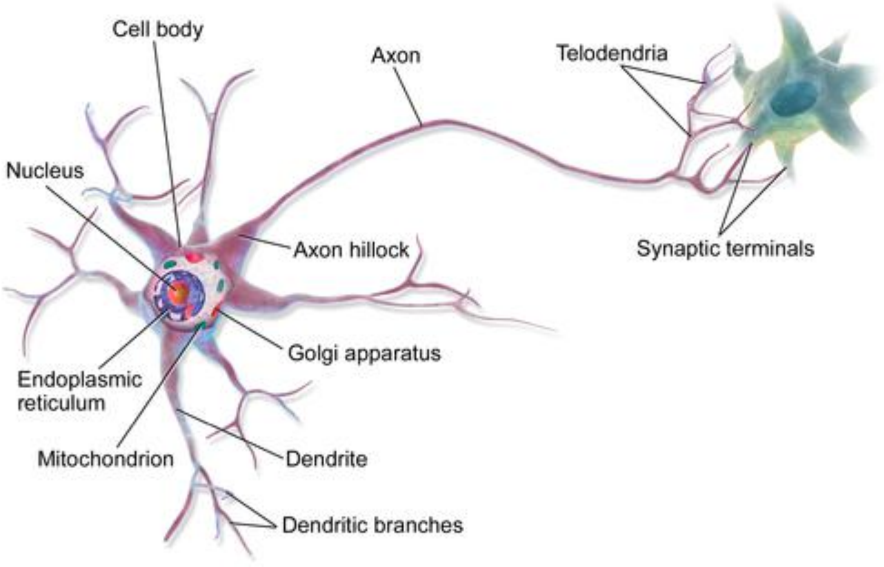
Build a processor tuned to application - specialize for DNNs

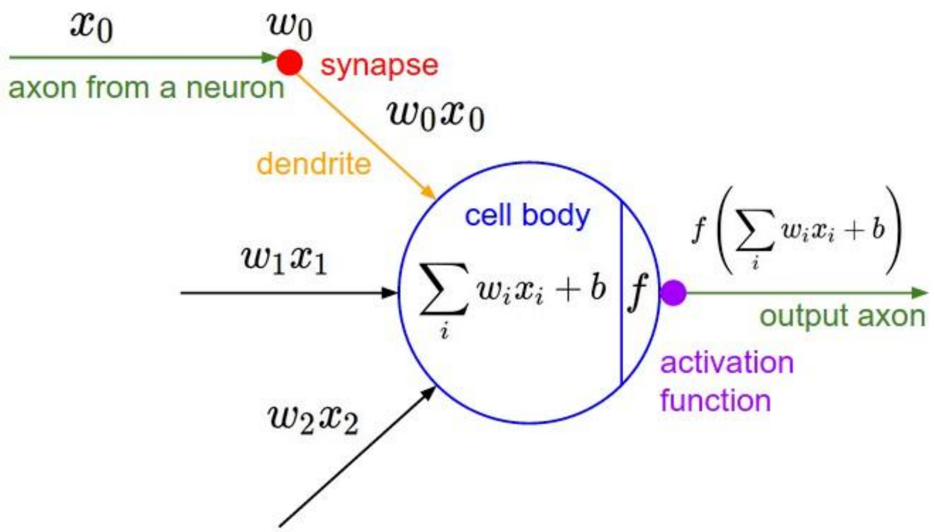
- More effective parallelism for AI Operations (not ILP)
  - SIMD vs. MIMD
  - VLIW vs. Speculative, out-of-order
- Eliminate unneeded accuracy
  - IEEE replaced by lower precision FP
  - 32-64 bit bit integers to 8-16 bit integers
- Model Compression
- Efficient memory systems


Codesign
Model
Compression
Quantization
and
Number
Systems
DNN
Background
Deep Learning/AI
Processors
Parallelization
Case Studies
Data
Transport

DNN Primer

Artificial Neurons Tries to Imitate Real Neurons

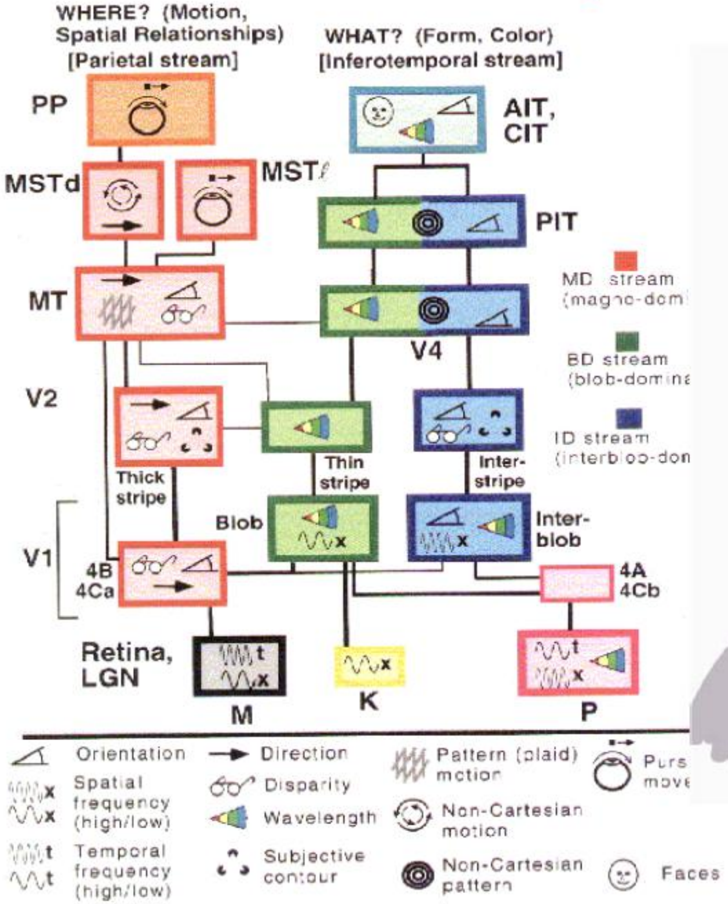


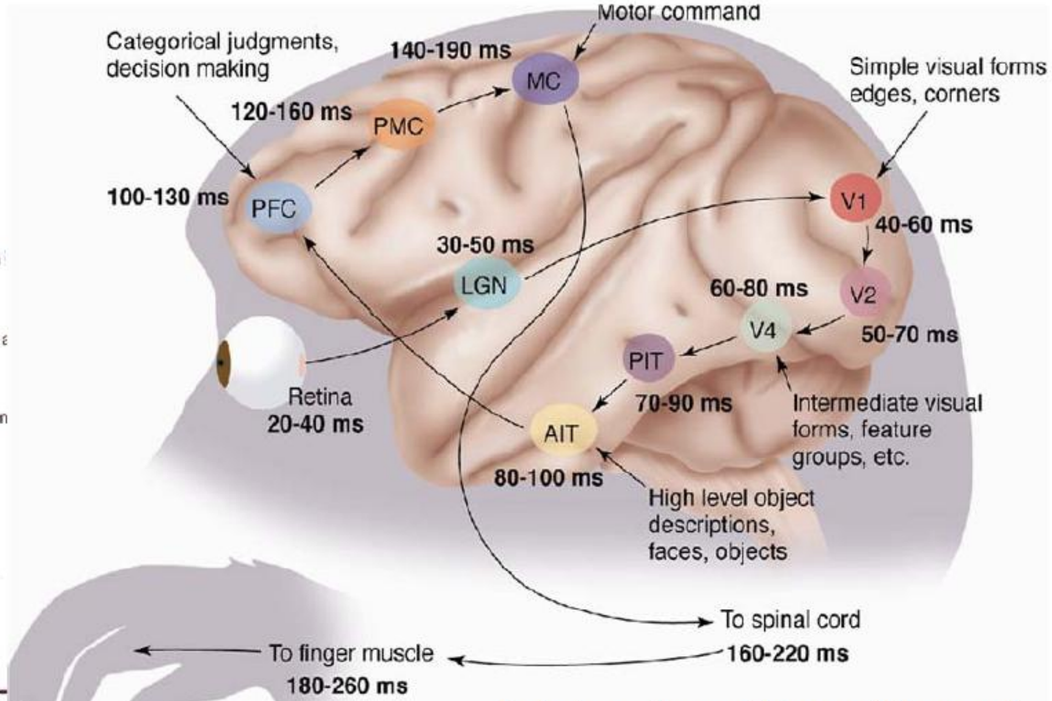


[https://en.wikipedia.org/wiki/Neuron]

Neural Networks Attempts to Imitate Real World Neural Networks

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



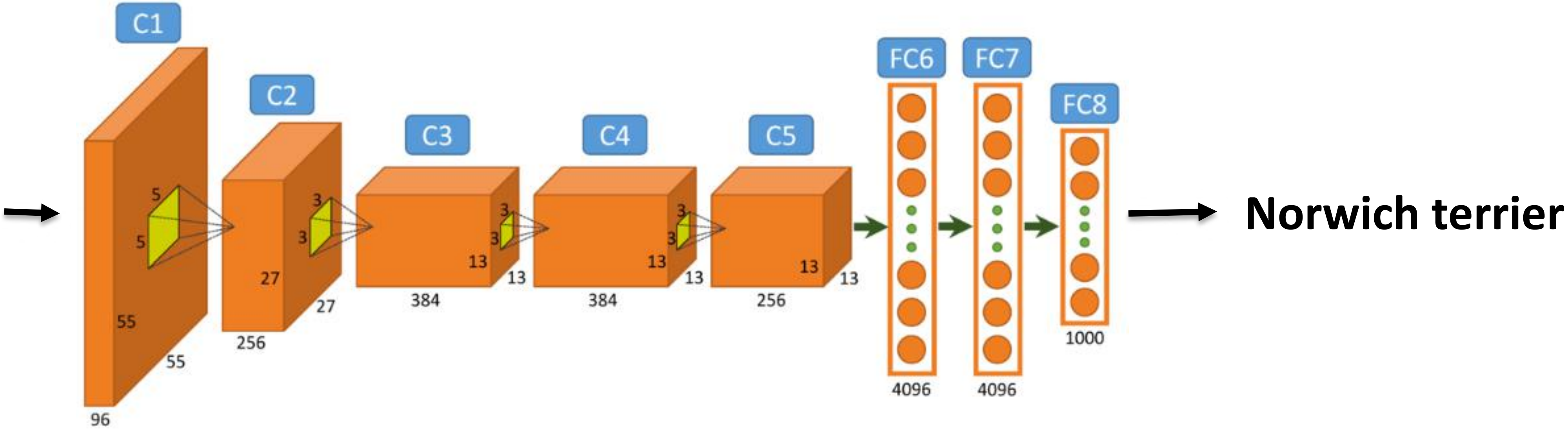


[picture from Simon Thorpe]

[Gallant & Van Essen]

Deep Learning Neural Network

3x227x227



Deep Learning Operations

Data Layers
Image Data
Database
HDF5 Input
HDF5 Output
Input
Window Data
Memory Data
Dummy Data

Vision Layers
Convolution
SeparableConvolution
OcvaveConvolution
Pooling
- MaxPooling
- AvgPooling
- RandomPooling

GlobalAveragePooling
Crop
Deconvolution Layer
UpSampling

Recurrent Layers
Recurrent
RNN
LSTM
GRU

Common Layers
Dense
Dropout
Embed

Utility Layers
Flatten
Reshape
Batch Reindex
Split
Concat
Slicing
Eltwise
Filter/Mask
Parameter
Reduction
Silence
ArgMax
Softmax

Normalization Layers
Batch Normalization

Activation Layers
ReLU/Rectified-Linear and Leaky-ReLU
ELU
Sigmoid
Tanh
Absolute Value
Hard sigmoid

Deep Neural Network

- A deep neural network (DNN) consists of many **layers** of neurons (perceptrons)
- Each connection indicates a weight w
- Feeding neurons into each other and non-linearities allows a DNN to learn **complex decision boundaries**

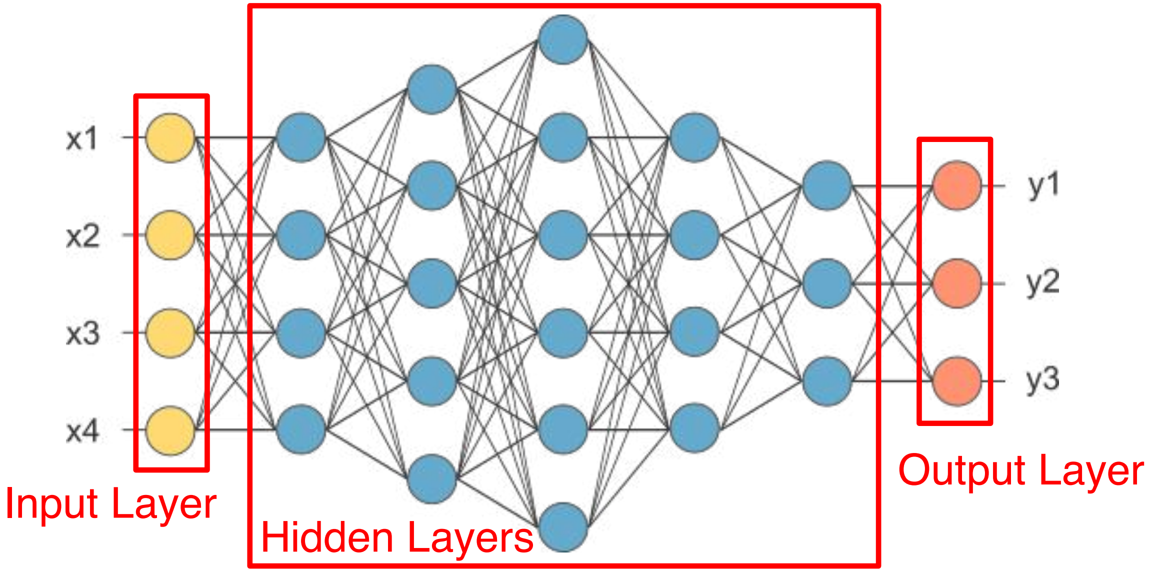
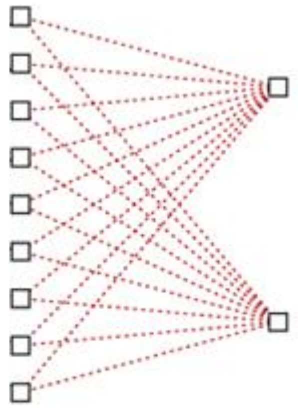
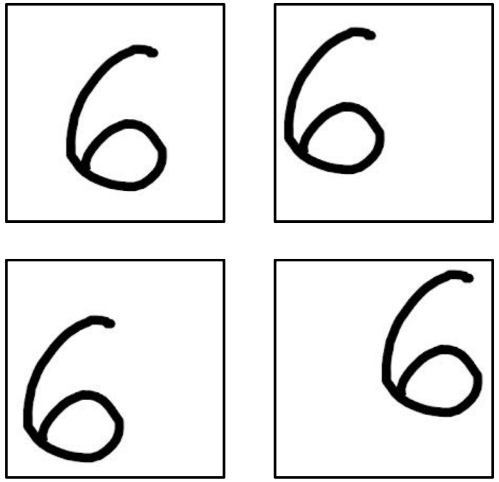


Image credit: http://www.opennn.net/

Neural Networks for Images

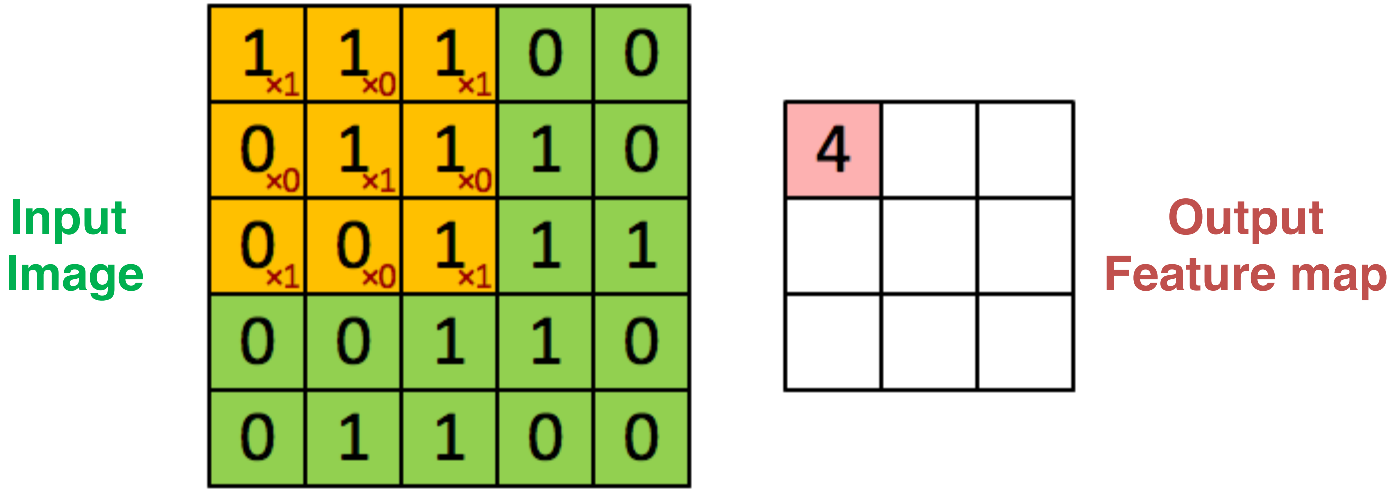


- So far, we've see networks built from **fully-connected layers**
- These networks don't work well for images. Why?



- Images are typically **shift-invariant** (i.e. a 6 is a 6 even when shifted)
- But a fully-connected neuron probably won't work when the input is shifted

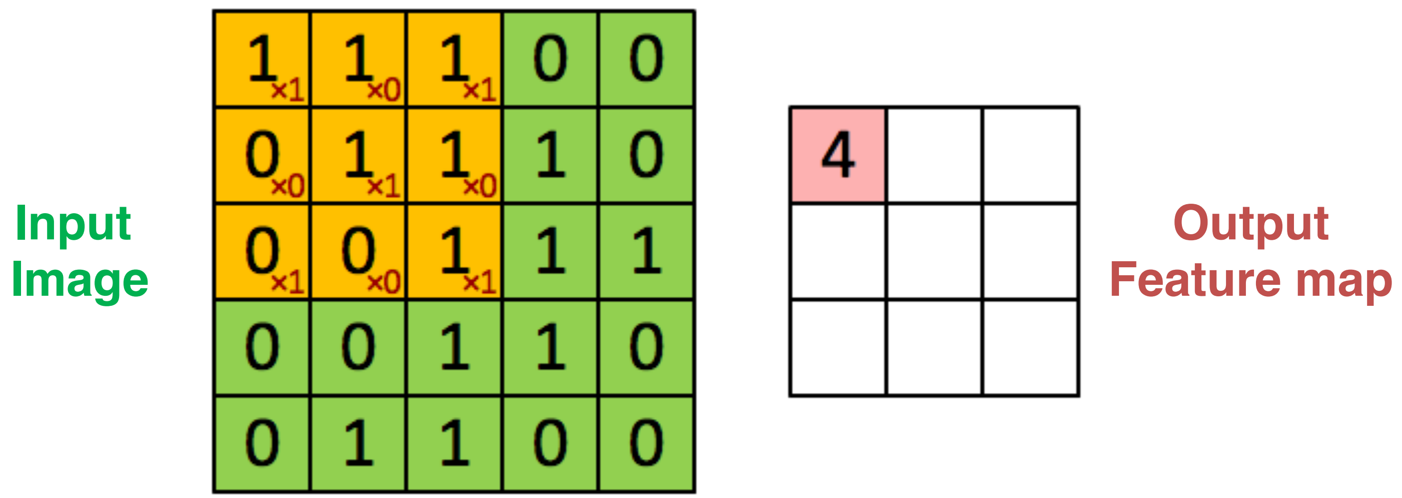
The Convolutional Filter



- Each neuron learns a **weight filter** and **convolves** the filter over the image
- Each neuron outputs a 2D **feature map**
  (basically an image of features)

Image credit: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

The Convolutional Filter

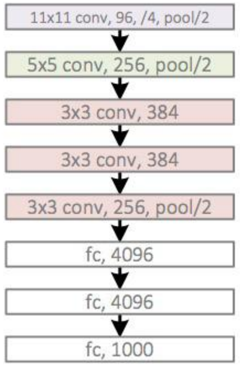


- Each point in the feature map encodes both a decision and its spatial location
- **Detects the pattern anywhere in the image!**

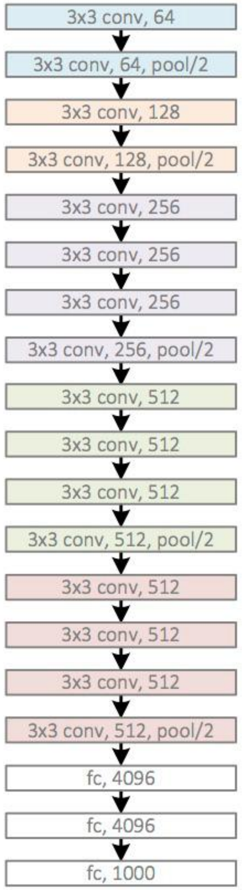
Image credit: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Revolution of Depth

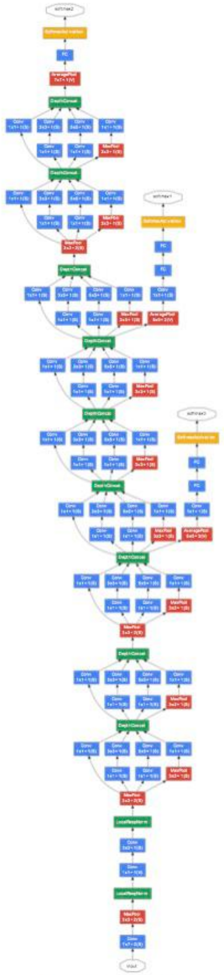
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

RNNs – Because We Need to Remember the History…

Limitations of CNN based networks

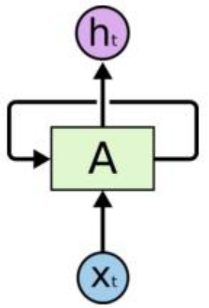
- We only look at the present to make a decision
- Examples have fixed length

Examples where history is important

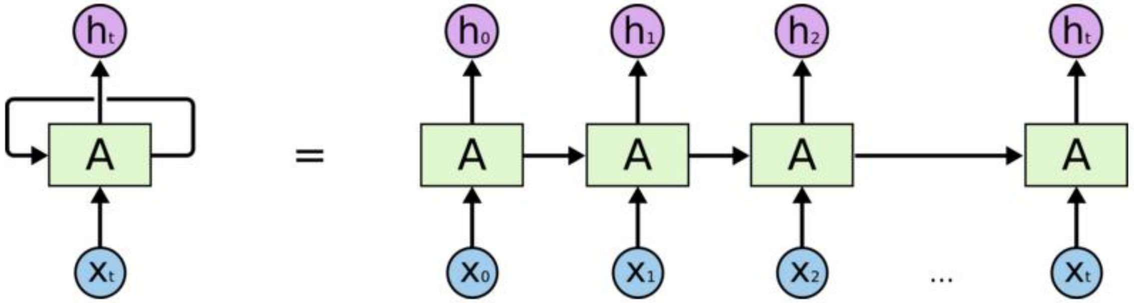
- Video processing
- Audio
- Text analysis

Recurrent Neural Networks

- RNNs remembers the past by feeding back memory to next iteration
- Implementation is really an unrolling of the network



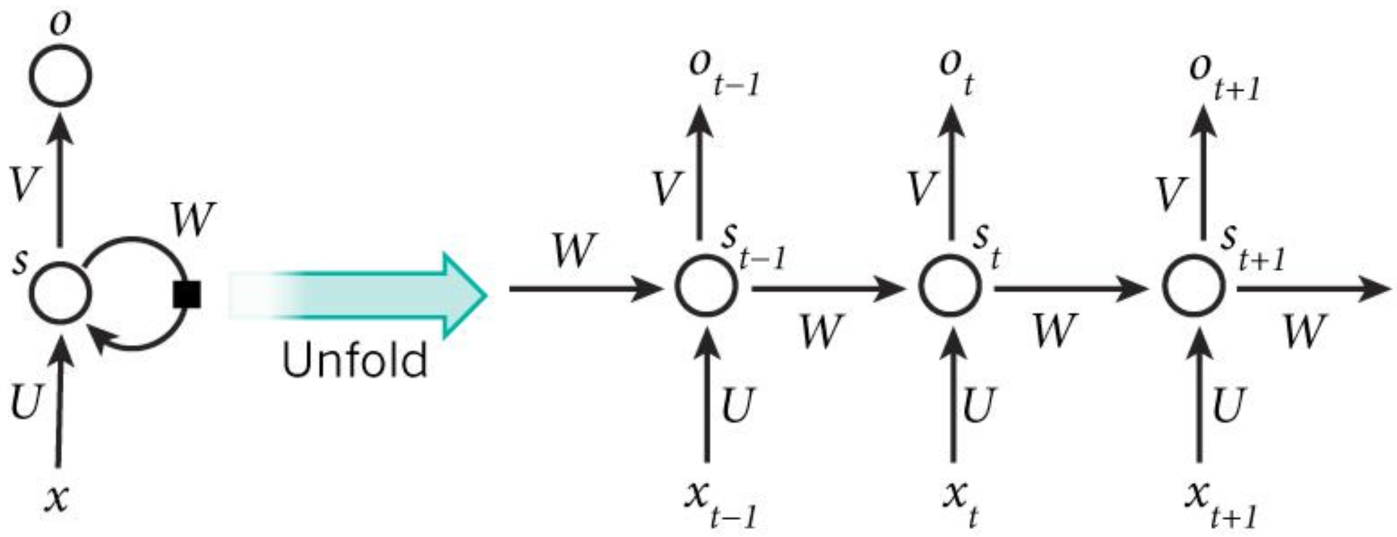
Recurrent Neural Networks have loops.



An unrolled recurrent neural network.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

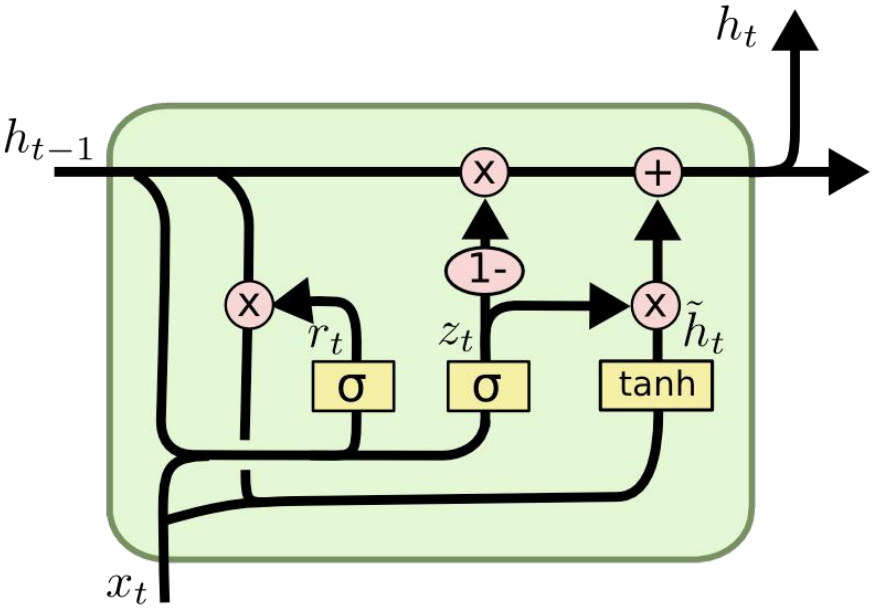
A Closer Look at RNNs



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = \mathrm{softmax}(Vs_t)$$

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

LSTM (Long Short Term Memory)

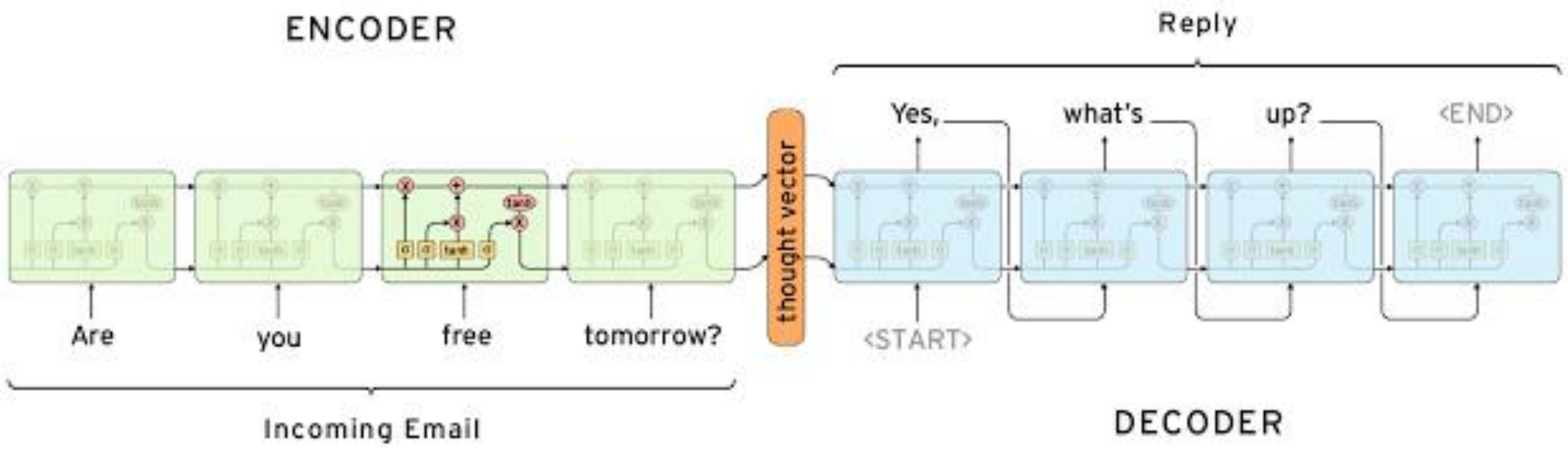


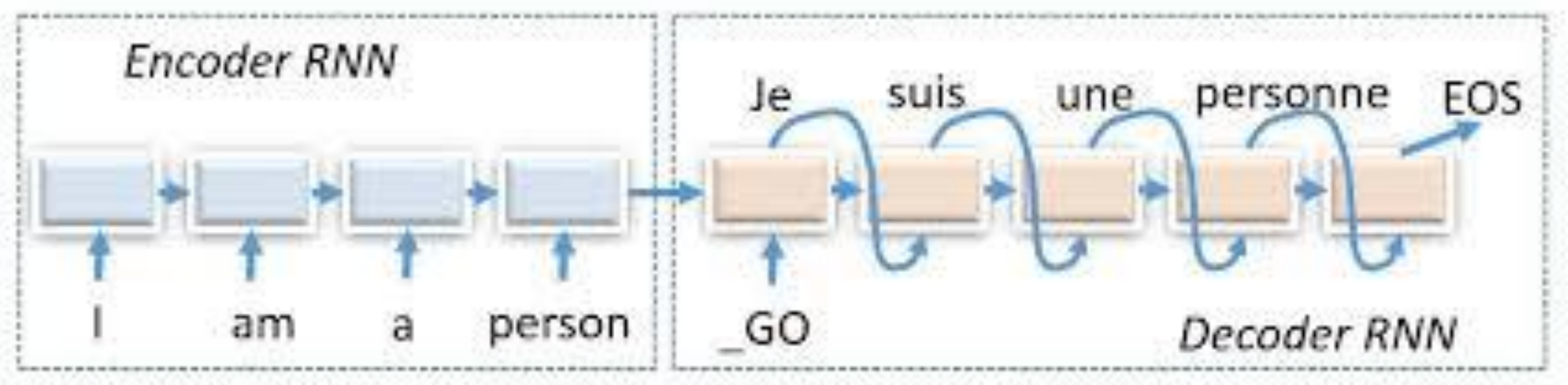
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

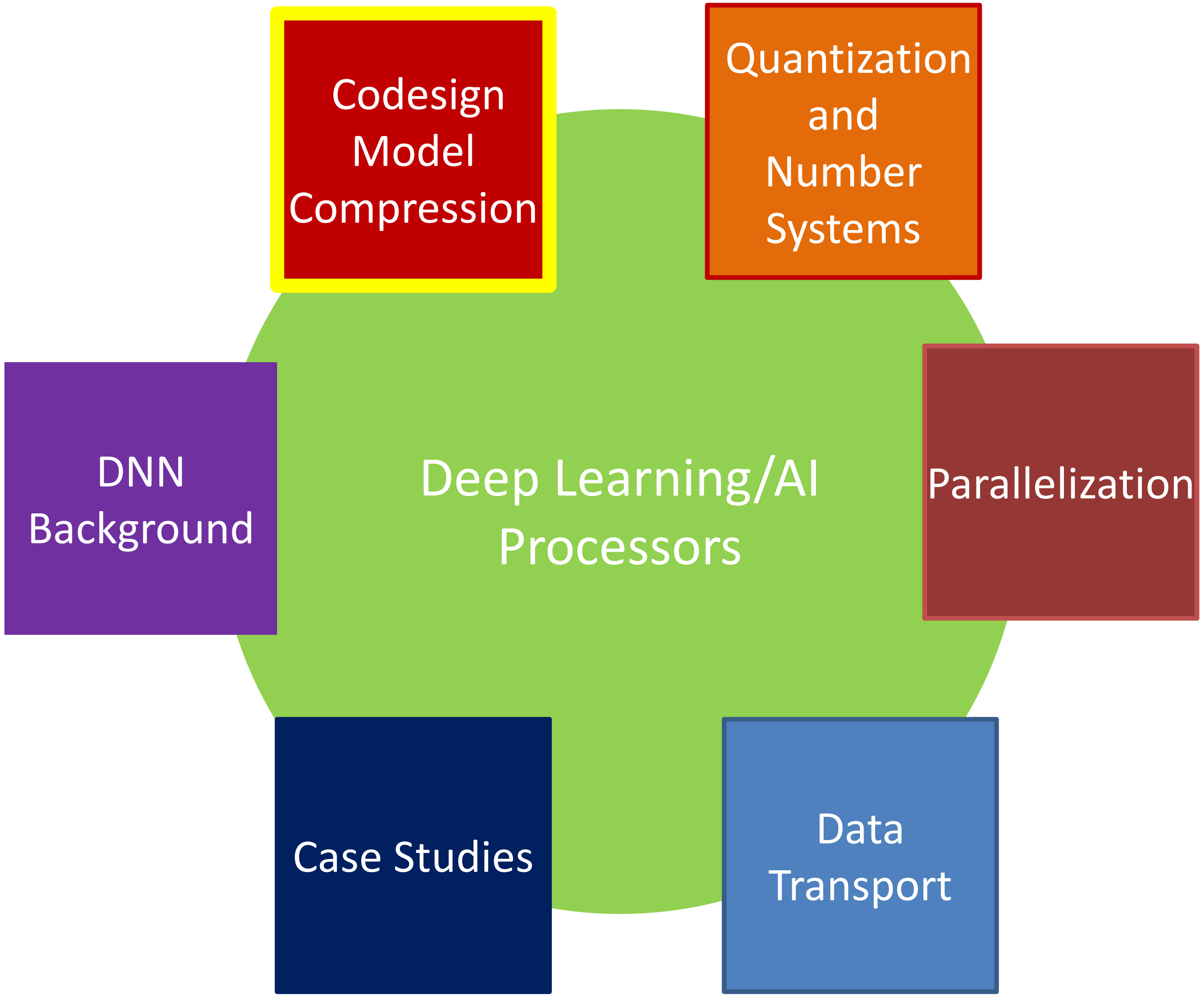
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

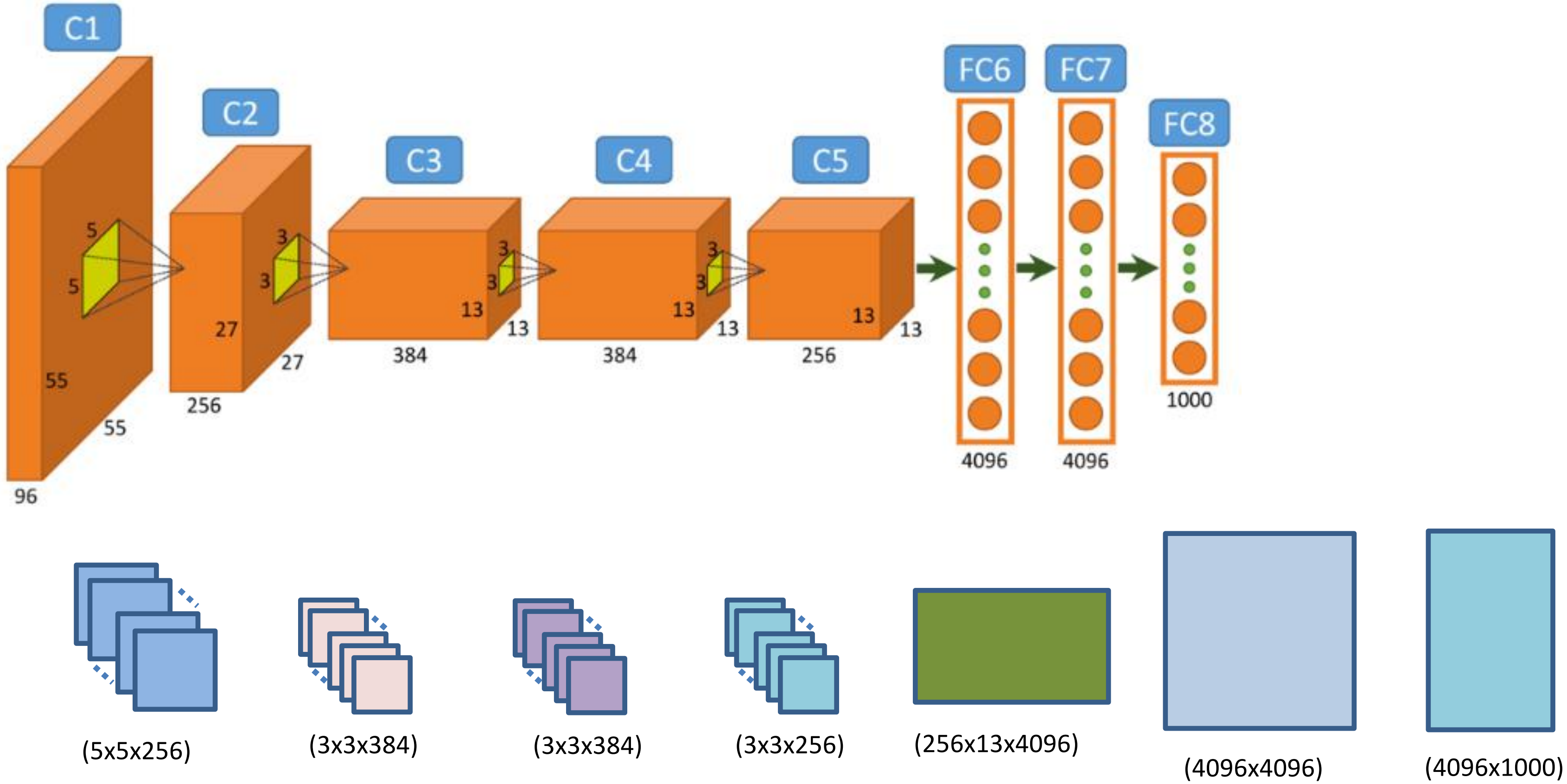

ENCODER
Reply
Yes,
what's
up?
<END>
thought vector
Are
you
free
tomorrow?
<START>
Incoming Email
DECODER


Encoder RNN
Je
suis
une
personne
EOS
I
am
a
person
_GO
Decoder RNN


Codesign Model Compression
Quantization and Number Systems
DNN Background
Deep Learning/AI Processors
Parallelization
Case Studies
Data Transport

Co-Design for DNN Model Compression

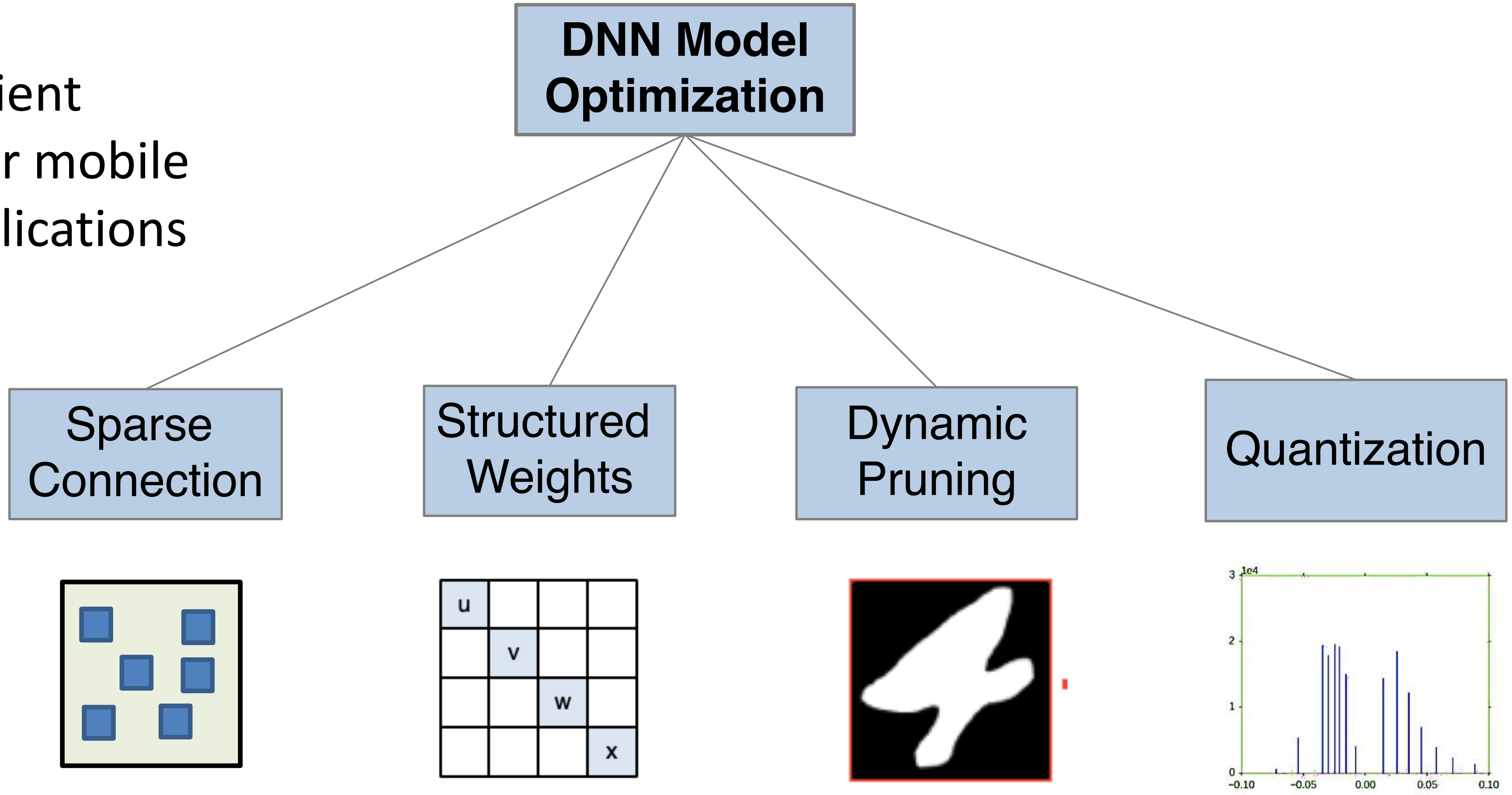
Why Model Optimization?



100M+ parameters
⇒ 100M 32-bit accesses
⇒ 64 mJ just to load the weights!

Model Optimization

Enable efficient inference for mobile or edge applications



Spectrum of DNN Compression Techniques

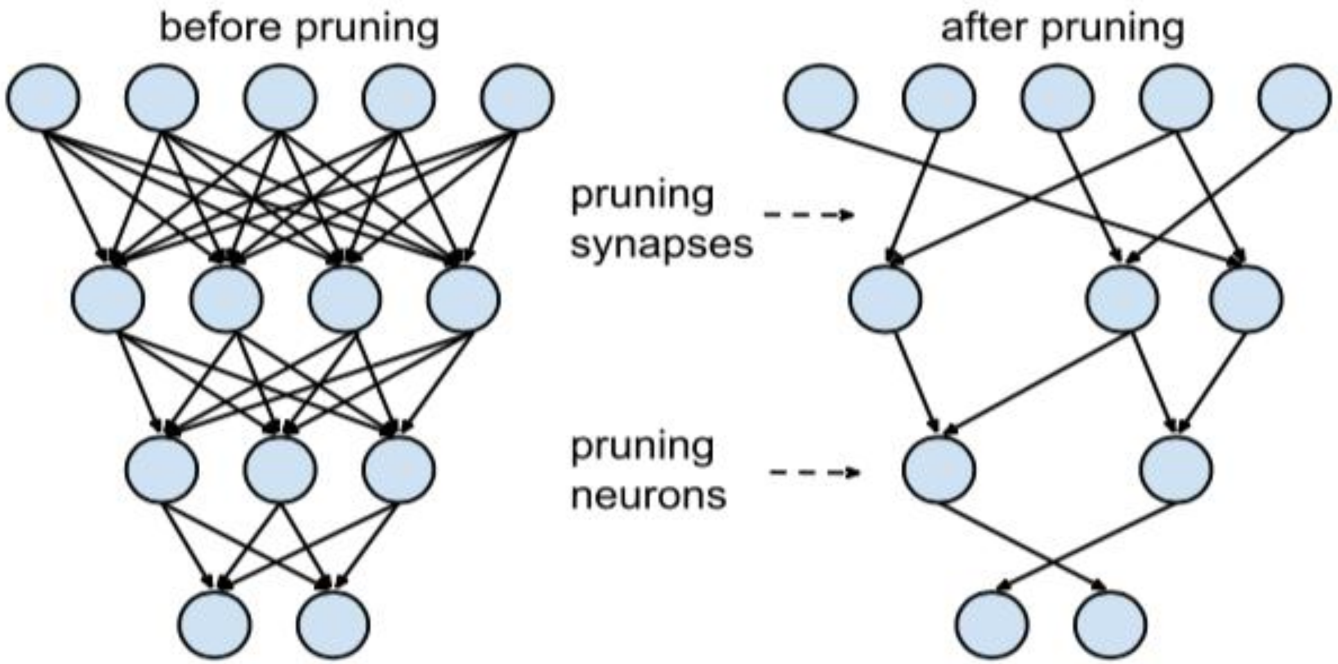
- **DNN Model Optimization**
  - Quantization
    - 16-bit Quantized CNN [Gupta JMLR'15]
    - Binarized CNN [Courbariaux NIPS'16] …
    - Low-Precision SGD [De Sa ISCA'17]
    - BNN-FPGA [Zhao FPGA'17] …
  - Sparse Connection
    - **Sparse Pruning** [Han NIPS'15, arXiv'16]
    - EIE, ESE [Han ISCA'16, FPGA'17]
    - Cambricon-X [Zhang ISCA'16]
    - SCNN [Parashar ISCA'17] …
    - **Group Convolution**: Xception [Canziani, arXiv'16], **MobileNets** [Howard arXiv'17] **ShuffleNet** [Zhang arXiv'17] …
  - Structured Weights
    - Circulant Projections [Cheng ICCV'15]
    - Structured Transforms [Sindhwani NIPS'16]
    - CirCNN [Ding MICRO'17], C-LSTM [Wang FPGA'18]
    - UCConv/HadaNet [Zhao CVPR'19] …
  - Dynamic Pruning
    - Runtime Neural Pruning [Lin NIPS'17]
    - Feature Boosting & Suppression [Gao ICLR'19] …
    - Channel Gating [Hua arXiv'18]

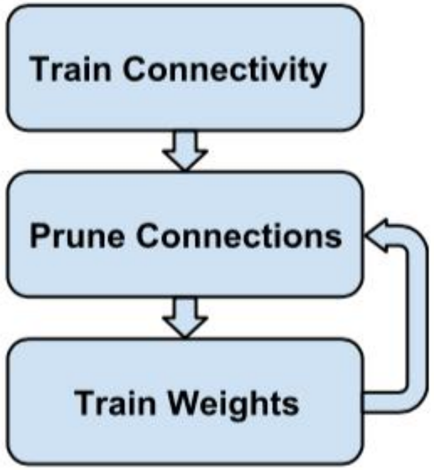
Goal: Enabling efficient inference in resource-limited devices in mobile or edge settings

Co-design of algorithms & hardware yields highest efficiency

Pruning – How many values?

90% values can be thrown away





Dense layers dominate the model size in classical DNNs

[Lecun et al. NIPS'89]
[Han et al. NIPS'15]

Drawbacks of Sparse Pruning

Dense Matrix

	3.2	1.3	
	-1.7		2.1
-2.1	-1.1	2.9	
	-1.0	-3.1	

Compressed Sparse Row (CSR)

3.2	1.3	-1.7	2.1	-2.1	-1.1	2.9	-1.0	-3.1

0	2	4	7	9

1	2	1	3	0	1	2	1	2

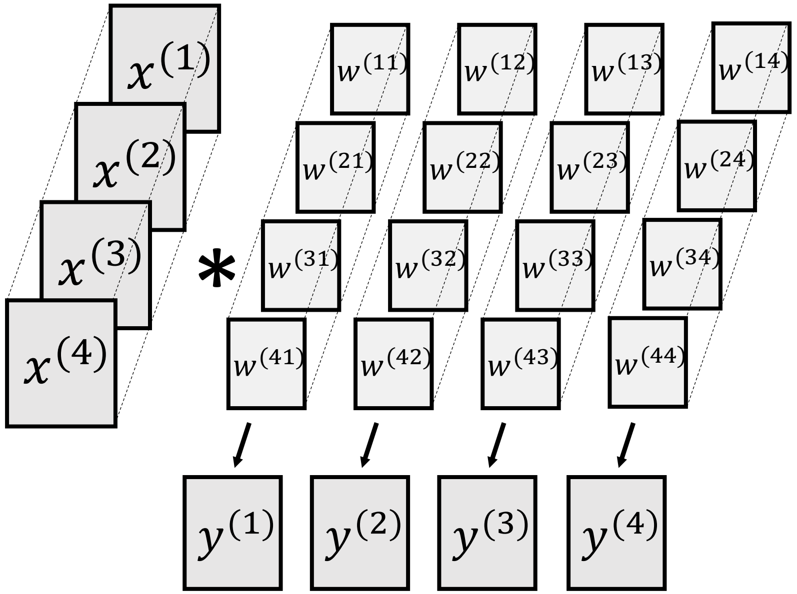
- Requires sparse matrix representation (e.g. CSR, CSC) to track locations of non-zero values
  - – Extra storage overhead
  - – Memory indirection
  - – Irregular parallelism

[Zhang – DAC 2019]

Regular Convolution (Conv) Revisited

$$y^{(j)} = \sum_{i=1}^{M} x^{(i)} * W^{(ij)}$$

- Assume M input features $x^{(i)}$ and N output features $y^{(j)}$
  - M input channels, N output channels
- $W = \{W^{(ij)}\}$ is the weight tensor of M x N filters



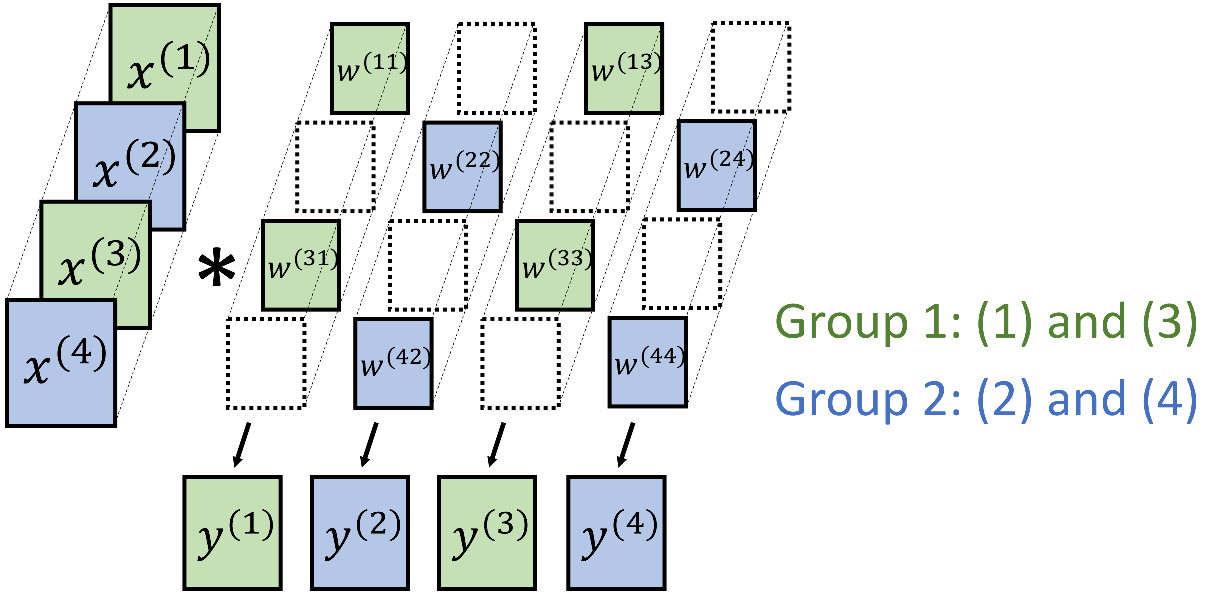
Example: a regular conv with $M = 4$, $N = 4$

[Zhang – DAC 2019]

Group Convolution (Group Conv)

$$y^{(g,j)} = \sum_{i=1}^{M/G} x^{(g,i)} * W^{(g,ij)}$$

- Assume the number of groups is G, each group has M/G input features $x^{(g,i)}$ and N/G output features $y^{(g,j)}$
- $W = \{W^{(g,ij)}\}$ is the weight tensor of $(M \text{ x } N)/G$ filters

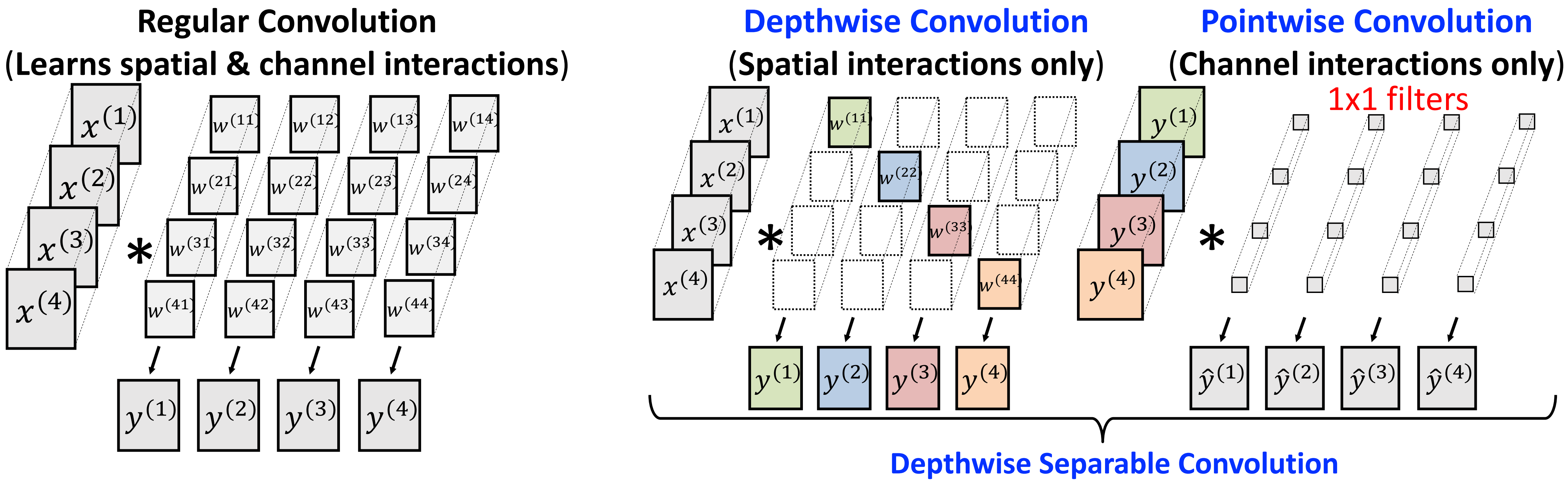


Example: a group conv with $M = 4,\ N = 4,\ G = 2$

Group conv reduces compute cost and weight size by a factor of G;
it preserves regularity/structure in both compute and memory access

[Zhang – DAC 2019]

MobileNets [1] : Depthwise Separable Convolution

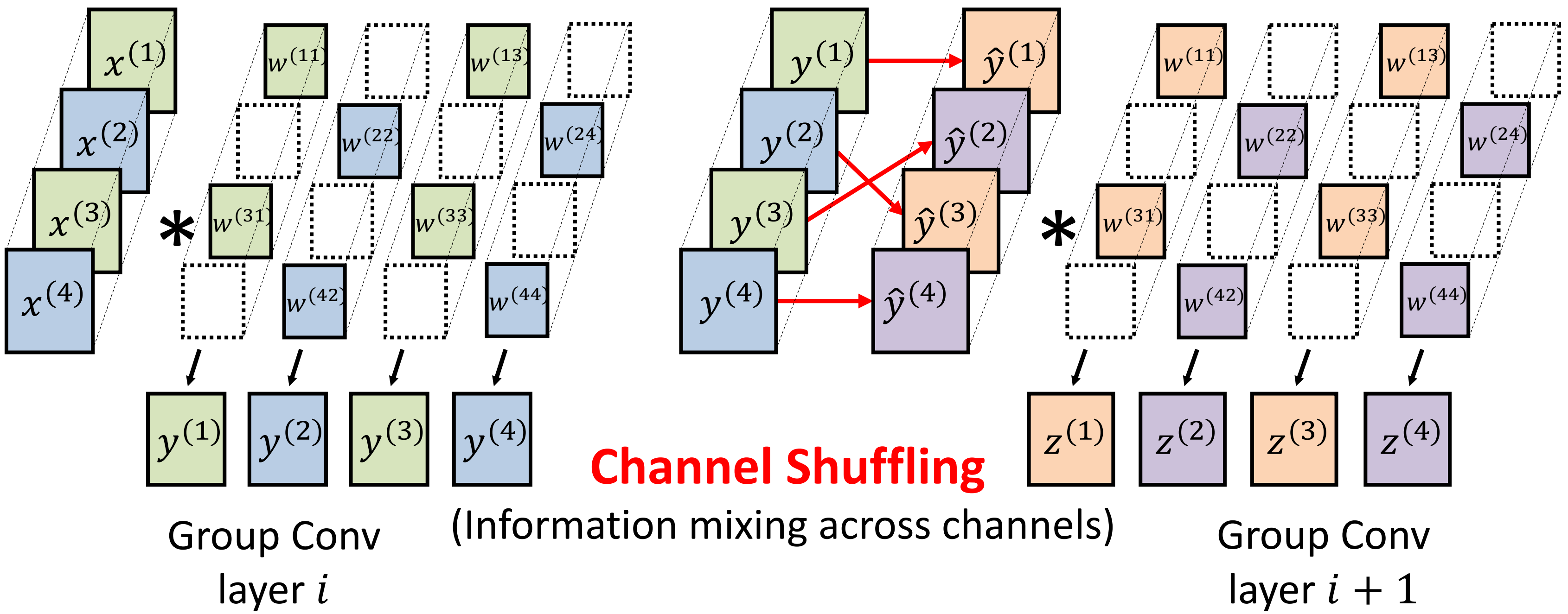


- Depthwise conv is an extreme case of group conv
  - Number of groups = input channels (one filter per channel)
- Pointwise conv enforces information mixing across channels

[1] Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv 2017.

[Zhang – DAC 2019]

ShuffleNet [1] : Group Conv + Channel Shuffling



- ShuffleNet compares favorably against MobileNets in accuracy and compute cost (FLOPs)

[1] Zhang et al., Shufflenet: An extremely efficient convolutional neural network for mobile devices. CVPR'2018.

DNN Compression with Structured Weight Matrices

- **Circulant matrix**: each row is the previous row shifted right

$$
\begin{bmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{bmatrix}
=
\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}^*
\times
\begin{bmatrix} u & & & \\ & v & & \\ & & w & \\ & & & x \end{bmatrix}
\times
\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}
$$

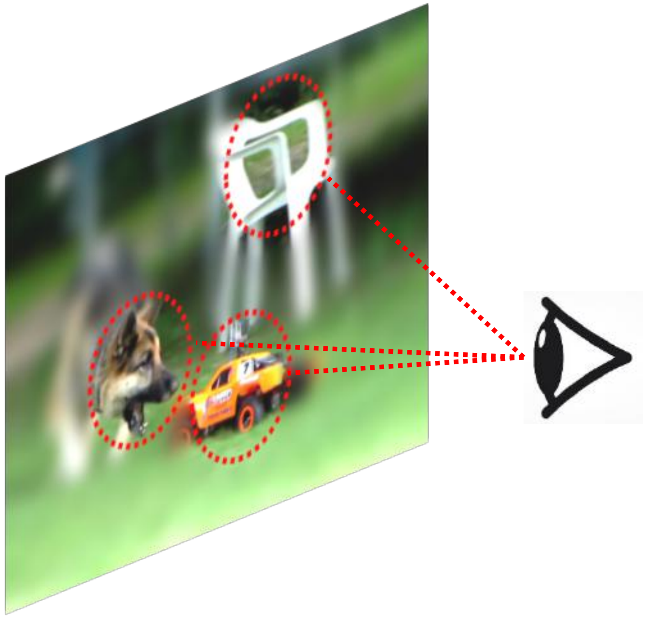
$$C = F^* D \, F$$

- Circulant matrices are diagonalized by Fourier transform
  - They are sparse in the Fourier domain

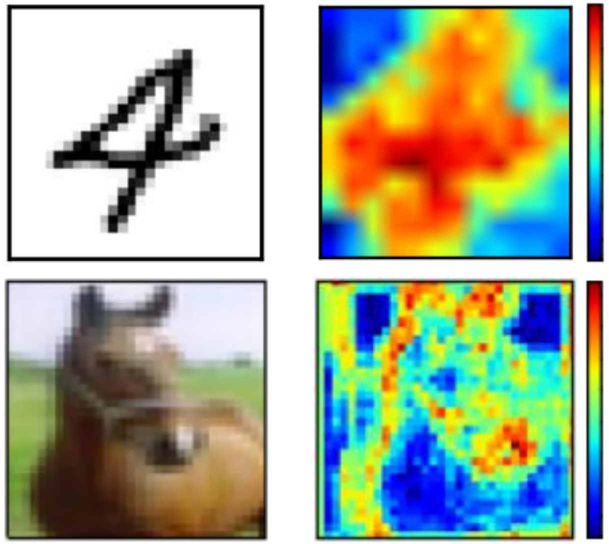
[Zhang – DAC 2019]

Compressing DNNs using Saliency?

Human visual recognition focuses on salient regions



Basic Idea: dynamically reduce compute effort at unimportant regions of an image

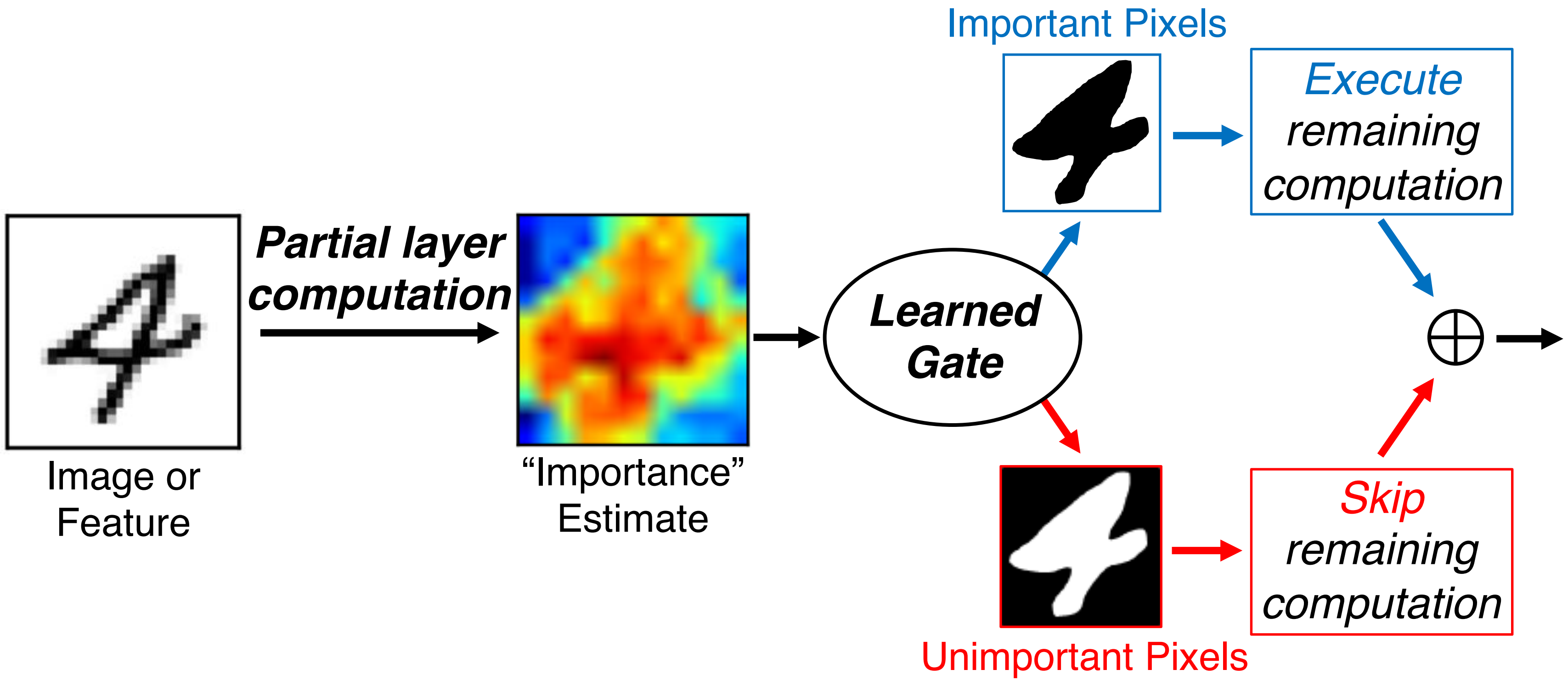


[Zhang – DAC 2019]

Channel Gating [1] : High-Level Idea

For each layer:

1. Estimate “importance” from partial layer computation
2. Gate remaining computation at “unimportant” pixels

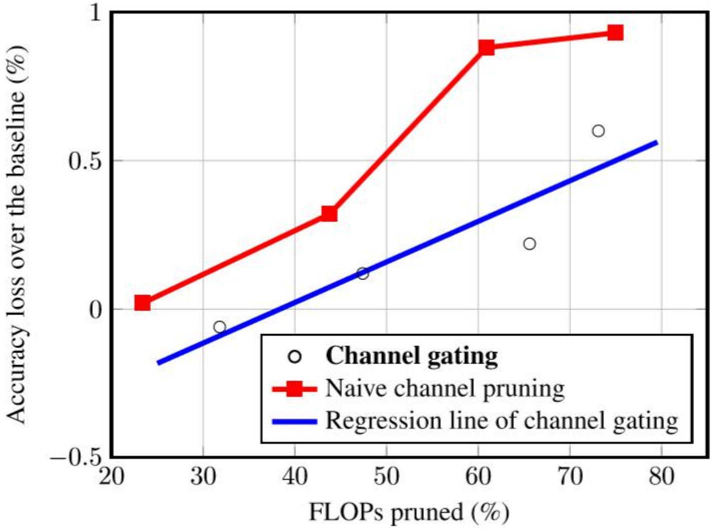


[1] W. Hua, C. De Sa, Z. Zhang, and G. E. Suh, **Channel Gating Neural Networks**, arXiv 2018.

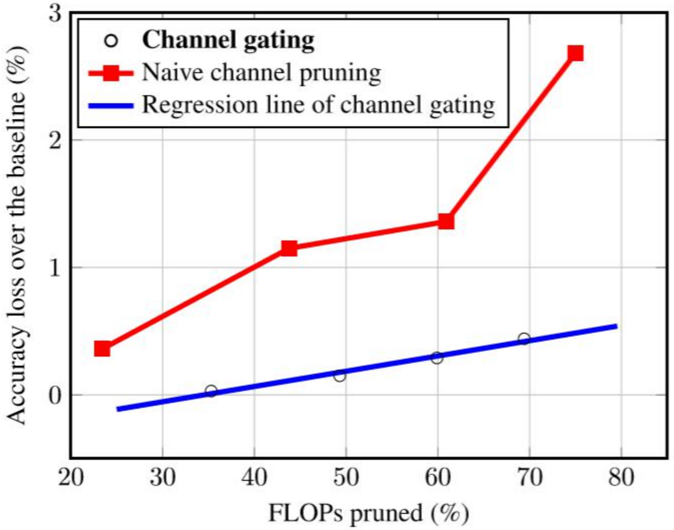
[Zhang – DAC 2019]

Channel Gating vs. Static Pruning

ResNet-18



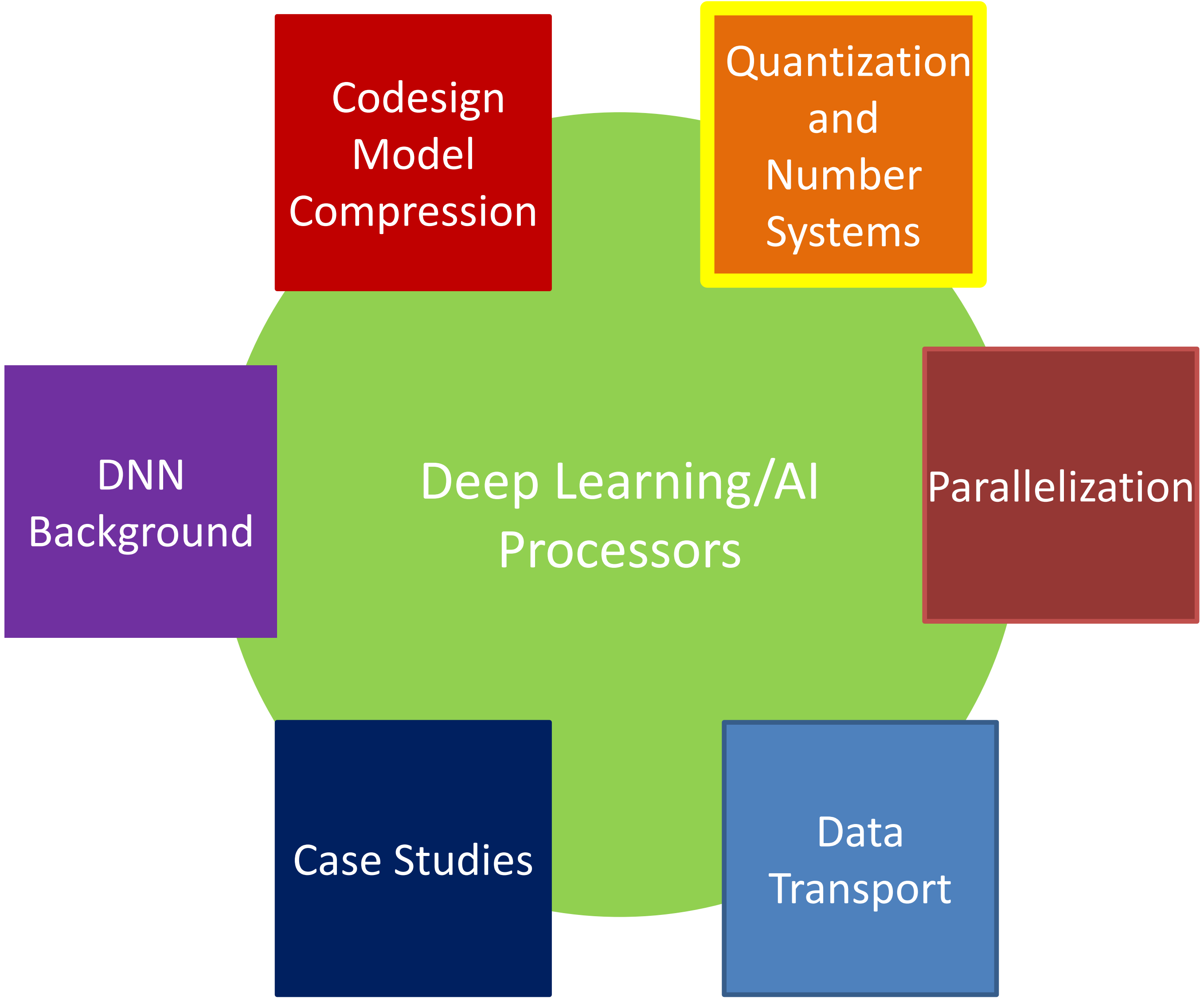
(a) CIFAR-10



(b) CIFAR-100

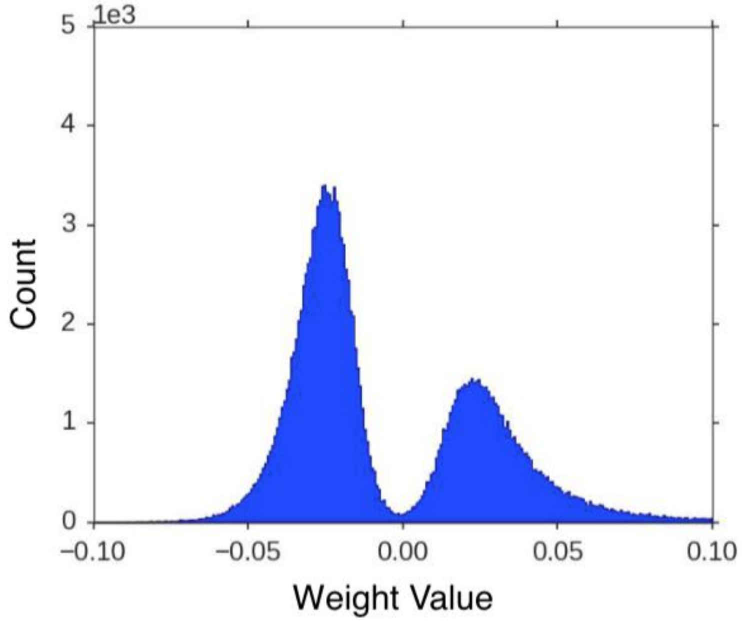
- Channel gating outperforms static pruning on both CIFAR-10 and CIFAR-100 datasets
  - Here the naïve static approach removes the same fraction of channels for all layers (except the first and last one)

[Zhang – DAC 2019]


Codesign Model Compression
Quantization and Number Systems
DNN Background
Deep Learning/AI Processors
Parallelization
Case Studies
Data Transport

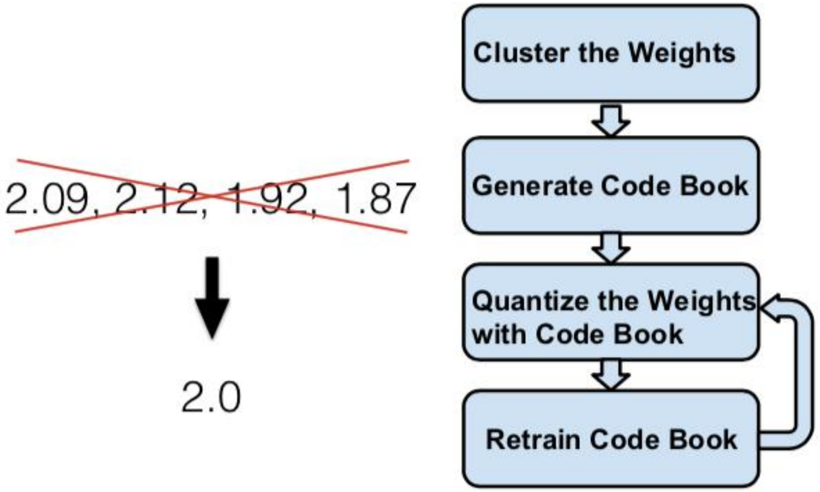
Quantization

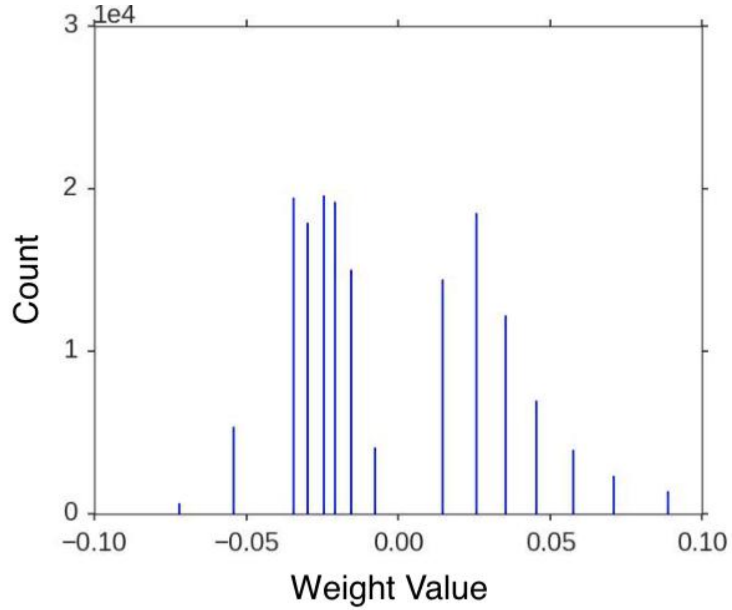
Quantization – How many distinct values?



16 Values => 4 bits representation

Instead of 16 fp32 numbers, store16 4-bit indices

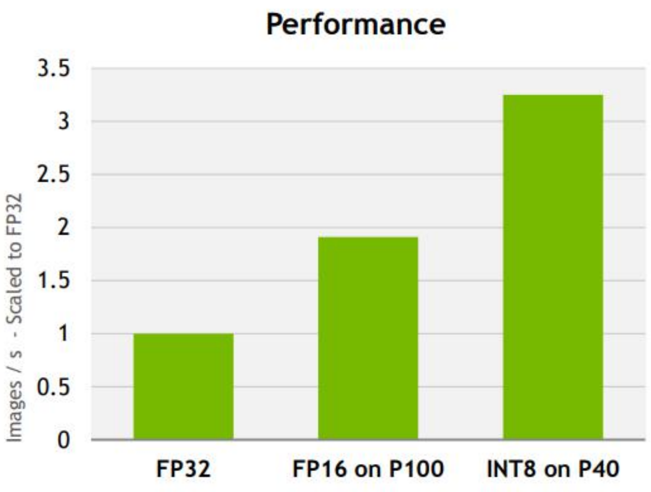




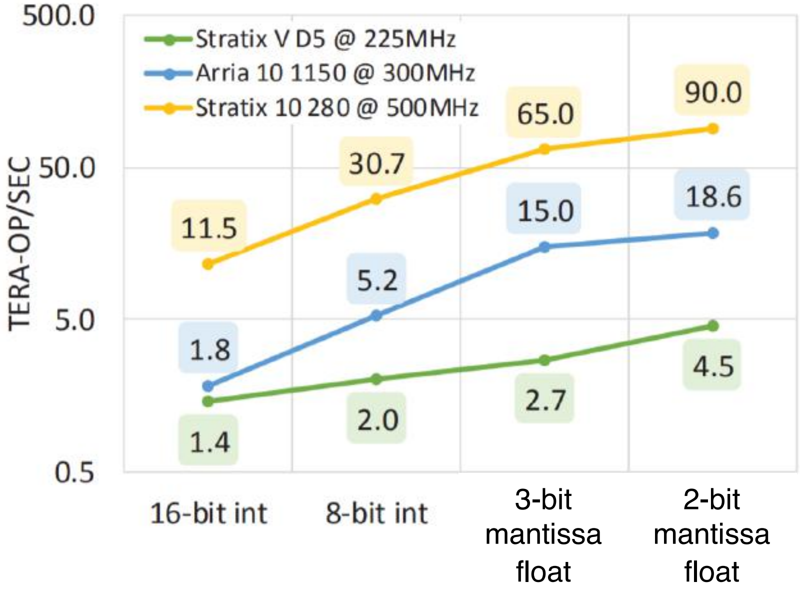
[Song et al, 2015]

Quantization is Key to Efficient Inference

ResNet50 on GPU



FPGA Performance



Reduced Precision → narrower arithmetic operators
→ less area and energy per op
→ fewer bits of storage per data

[1] https://developer.nvidia.com/tensorrt

[2] E. Chung, J. Fowers et al. **Serving DNNs in Real Time at Datacenter Scale with Project Brainwave**. *IEEE Micro*, April 2018.

Training Quantized Networks

- STE – Straight-Through Estimator
  - During forward pass, it is common to use quantized network
  - During backward pass, approximate the quantizer as a linear function
  - Usually present good results when compared with other approaches

```
def quantizer(x):
  return x + K.stop_gradient(do_something_with_x(x) - x)
```

https://arxiv.org/abs/1609.07061

Number Representations

Several Flavors of Data Types Available

- +1, -1 or 1,0 (xor, binary)
- +1, 0, -1 (ternary)
- Fixed point (ac_fixed)
- Power-of-2 (exponents)
- Shared exponent per layer or matrix
- Floating point numbers (bfloat16, fp8, fp16, fp32, fp64)
- POSITs

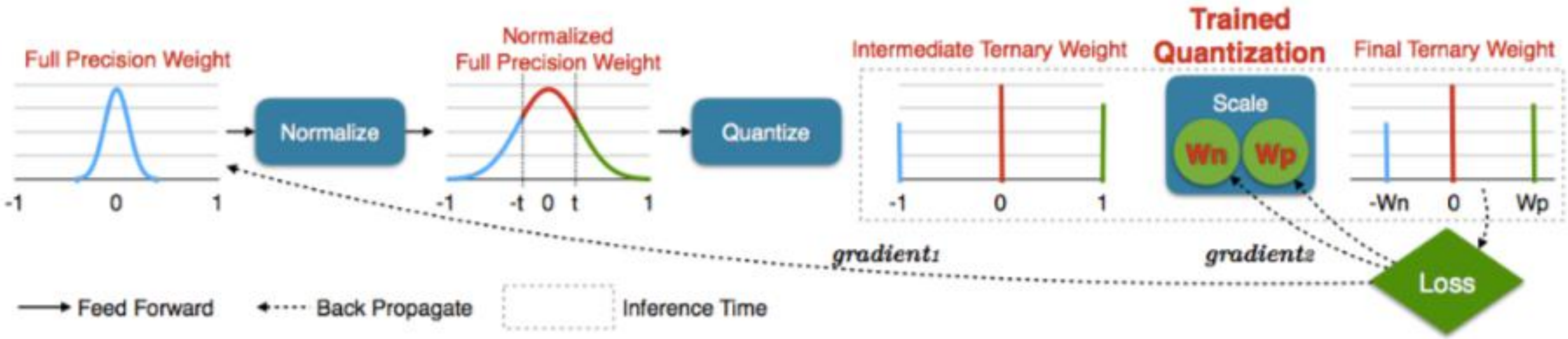
Stochastic Binary

$$x^b = \mathrm{sign}(x - z) = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

- Acts as a strong regularizer, making network more immune to noise
- When used in weights, remember that training responds to expected value of the gradient (= $2\sigma(x) - 1$, but we will need to clip weights between -1 and +1)
- Hard to implement in hardware as an activation layer

Ternary (+a, -b, 0)

- Q: What can we do with single bit or ternary (+a, -b, 0) representations
- A: Surprisingly, a lot. Just retrain/increase the number of activation layers

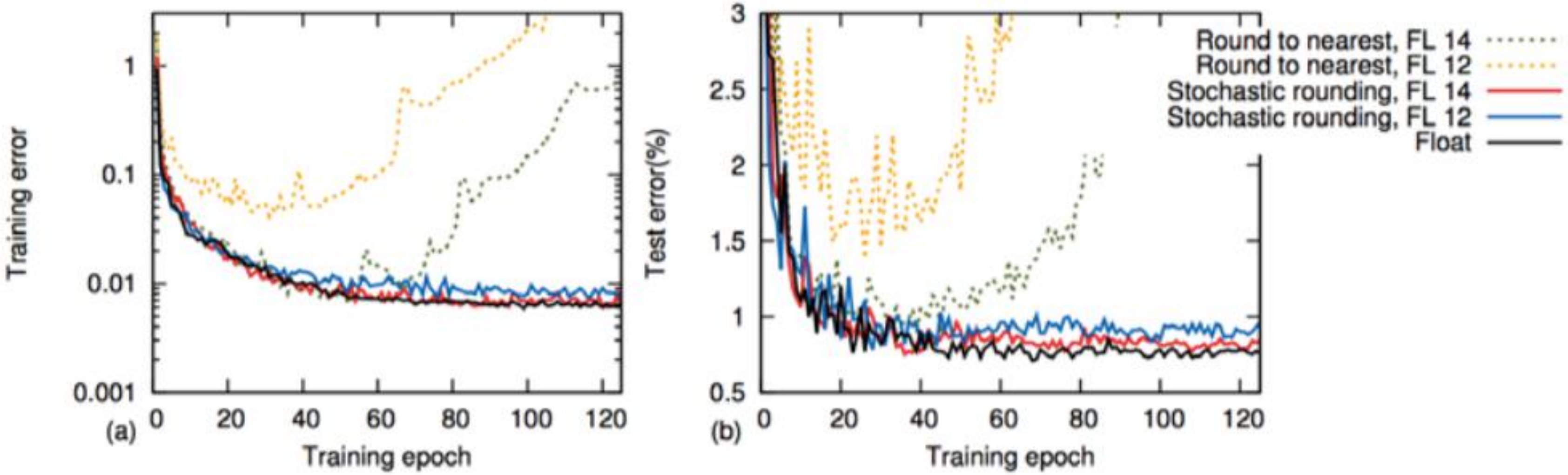


Model	Full resolution	Ternary Network	Improvement
ResNet-20	8.23	**8.87**	**-0.64**
ResNet-32	7.67	**7.63**	**0.04**
ResNet-44	7.18	**7.02**	**0.16**
ResNet-56	6.80	**6.44**	**0.36**

[Chenzhuo et al,, arxiv 1612.01064]

Removing Bias During Quantization

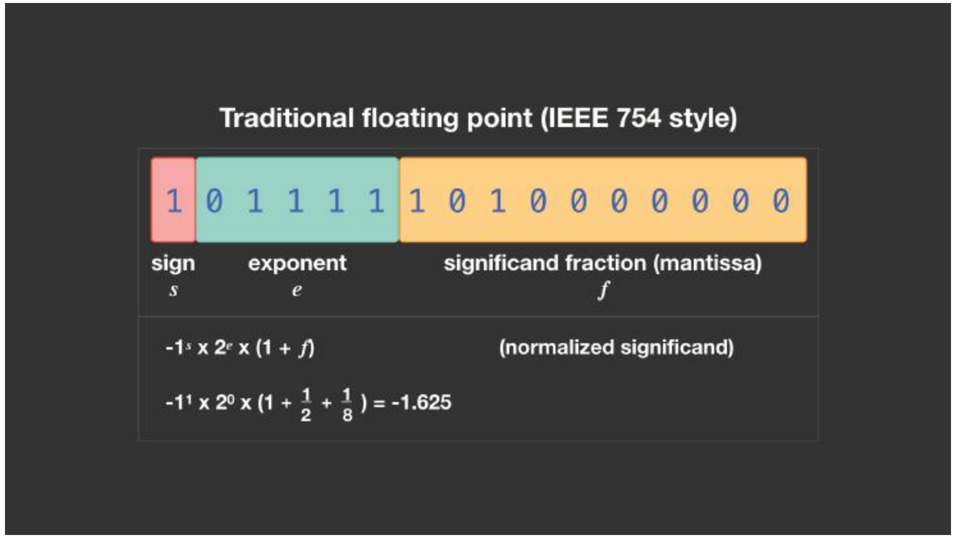
Stochastic Rounding: 2.2 rounded to 2 with probability 0.8
rounded to 3 with probability 0.2

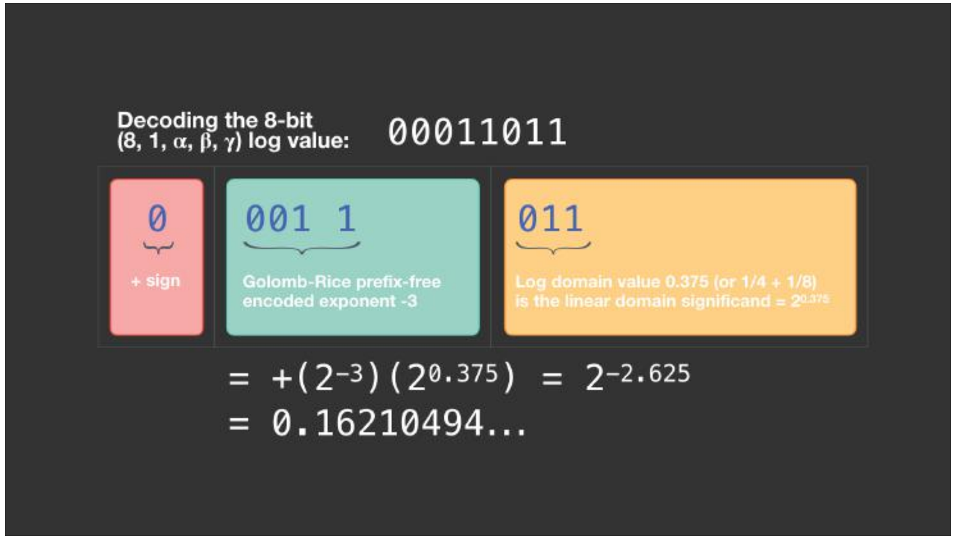


16-bits accuracy matches 32-bits!

Gupta, Suyog, et al. "Deep learning with limited numerical precision." CoRR, abs/1502.02551 392 (2015).

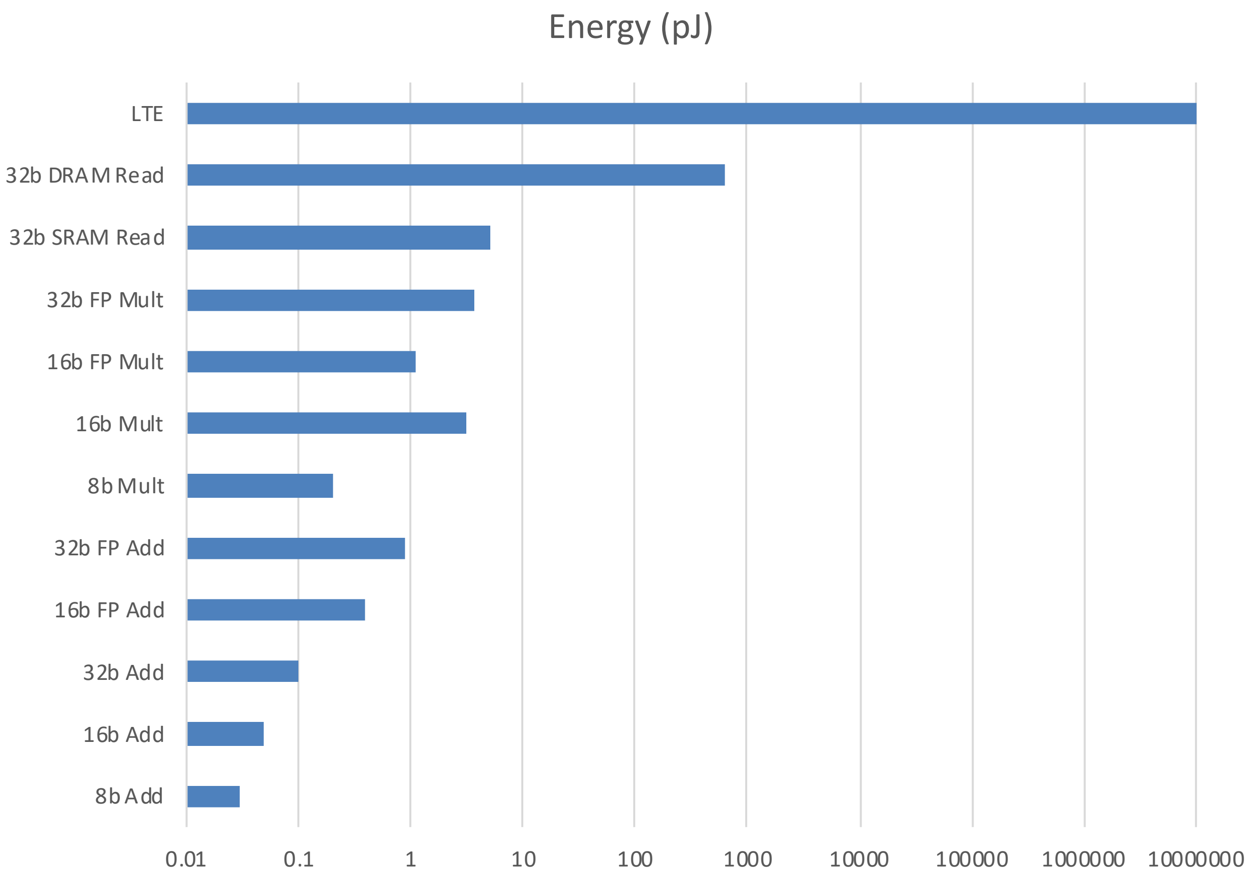
POSIT Numbers

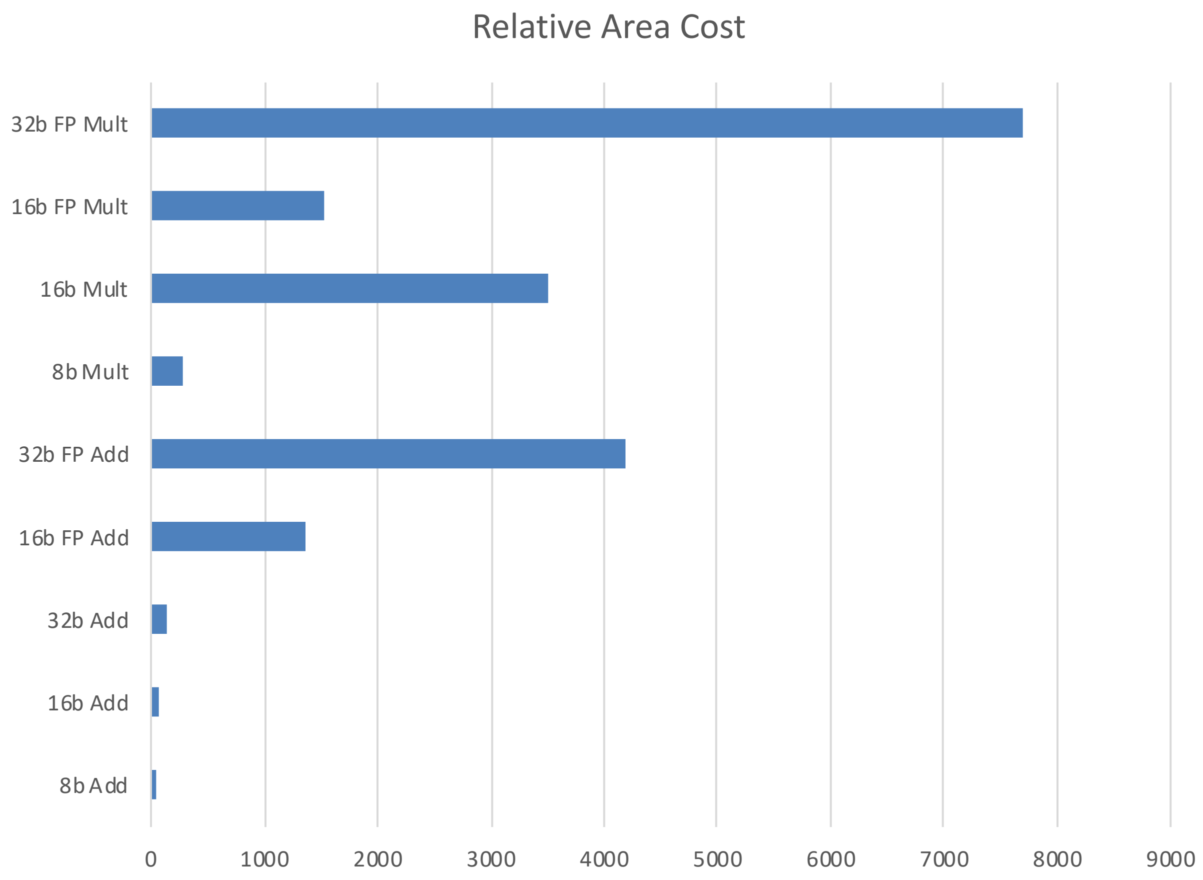




- Uses more mantissa bits towards 0, less bits in later representations
- Reduce number of bits required to store numbers, reducing memory traffic
- Still represented internally as a FP number (part of the exponent is variable
- Posit<8,1> encodes sigmoid in storage

https://code.fb.com/ai-research/floating-point-math/



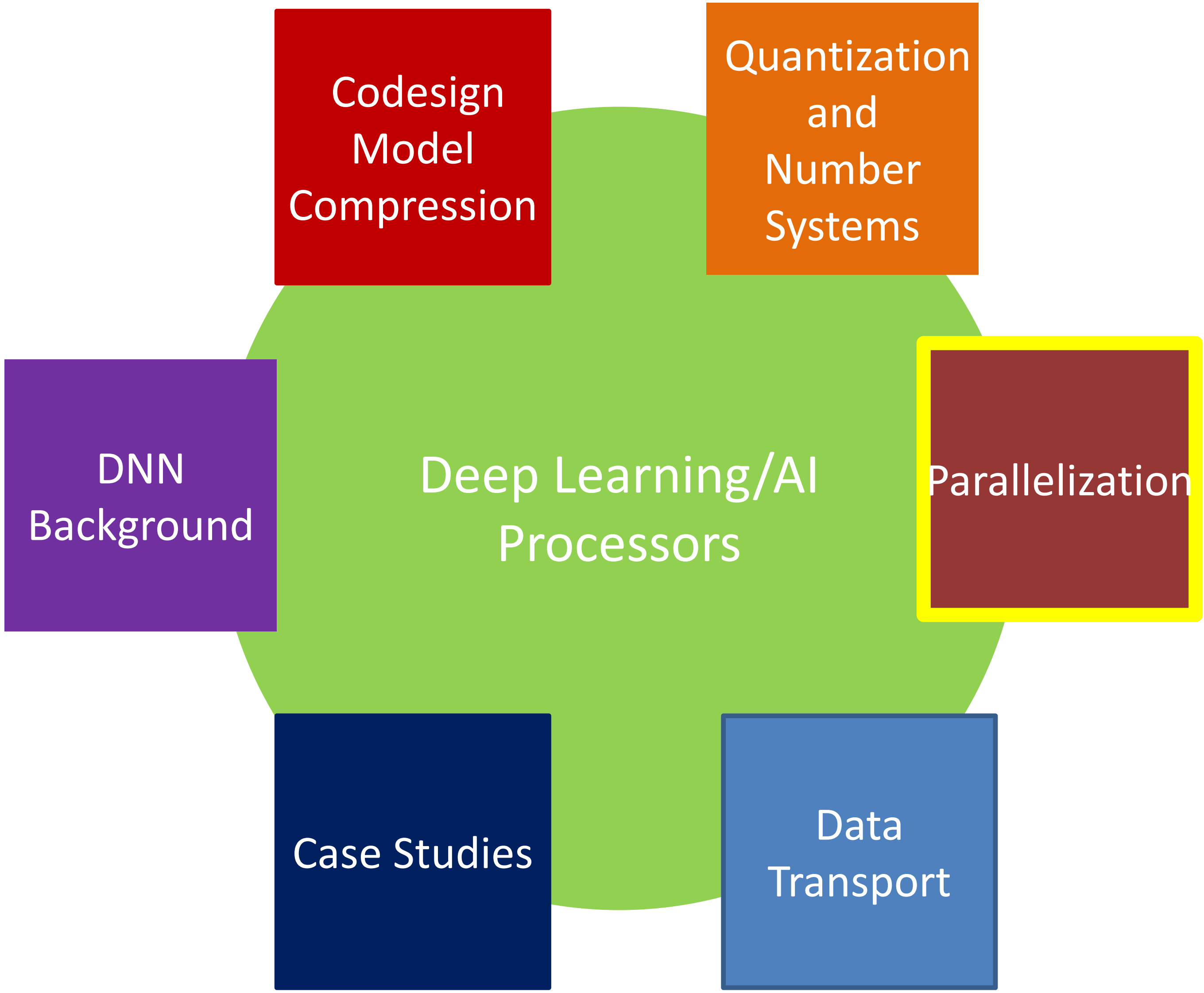


www.youtube.com/watch?v=eZdOkDtYMoo&feature=youtu.be&t=497

Jonh Brunhaver, Billl Dally, Mark Horowitz

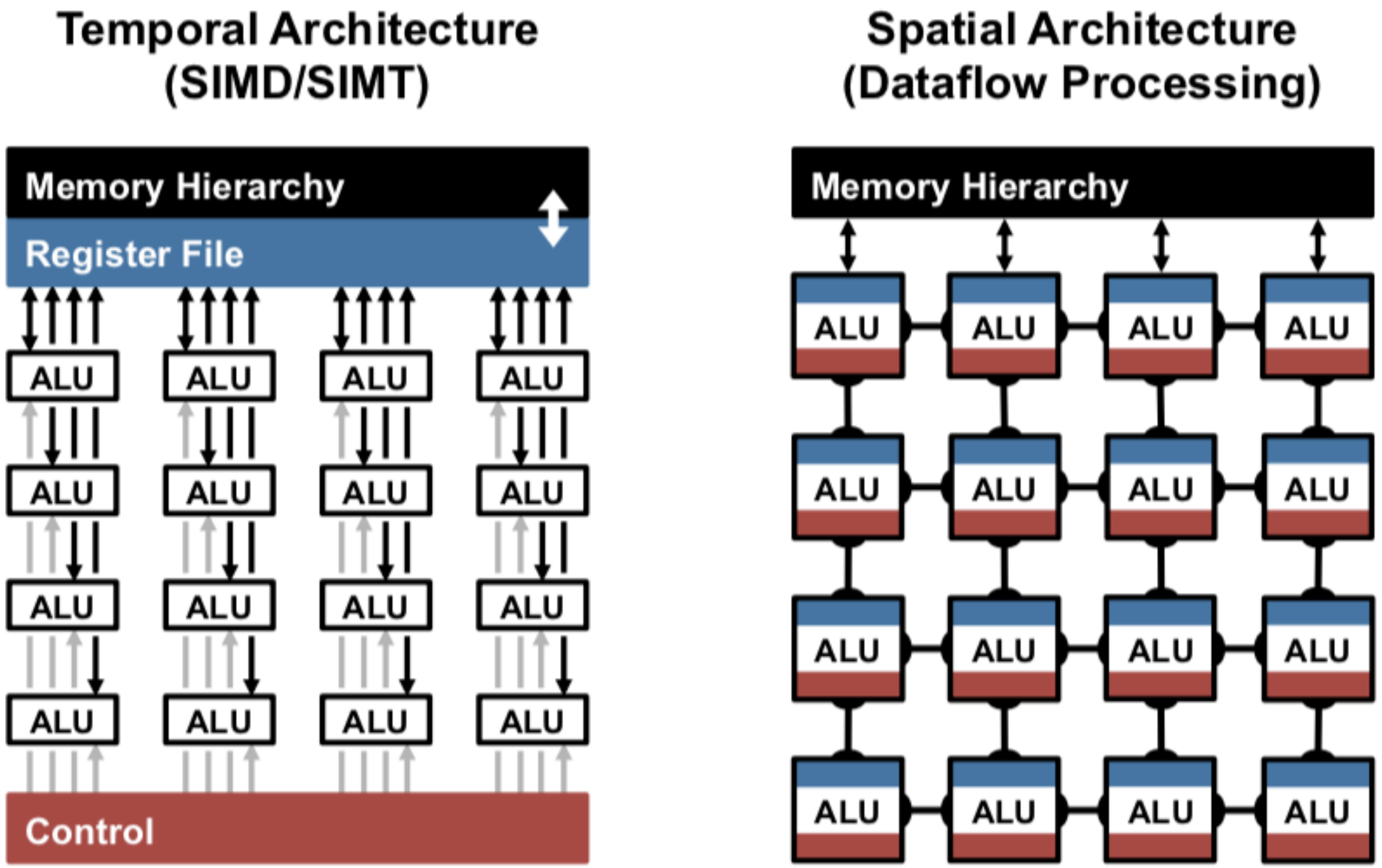
Energy, Area, Delay Trade-offs

- From energy point of view, reducing traffic to memory or to wireless is important
- From area point of view, reducing operations and muxes is important
- From delay/throughput point of view, non-sharing operations is important


Codesign Model Compression
Quantization and Number Systems
DNN Background
Deep Learning/AI Processors
Parallelization
Case Studies
Data Transport

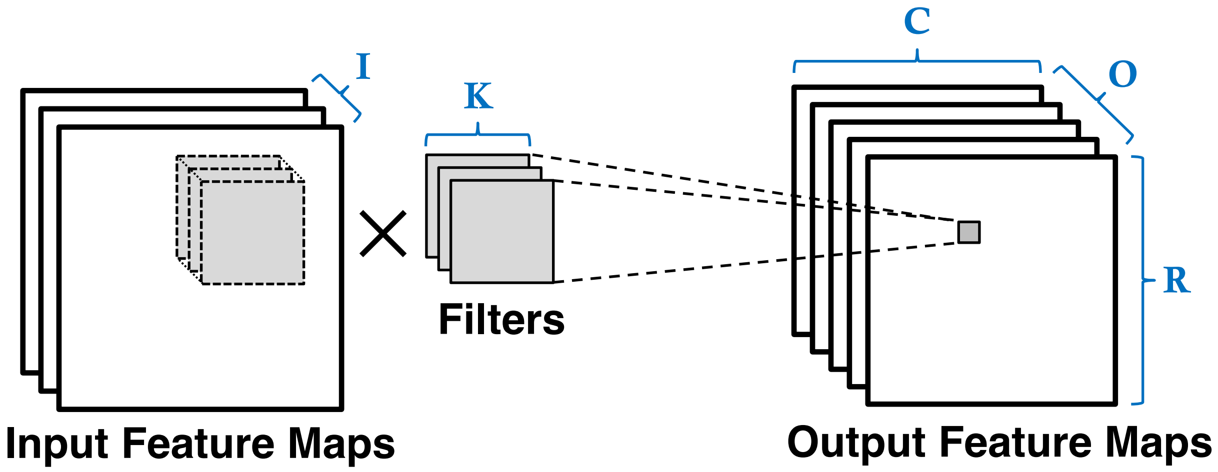
Parallel Architectures

Temporal and Spatial Parallelization



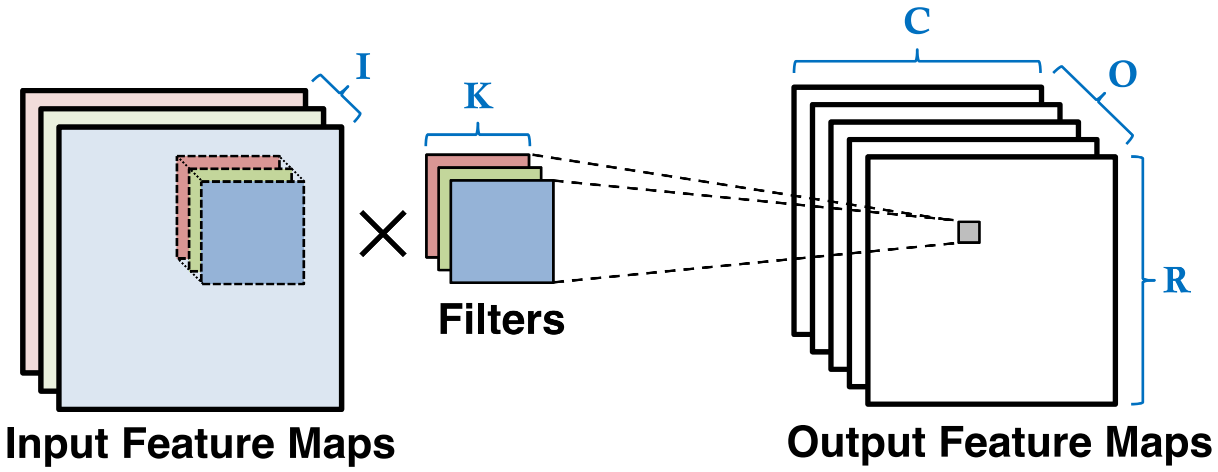
[Sze et al, arxiv 1703.09039]

Convolutional Layer



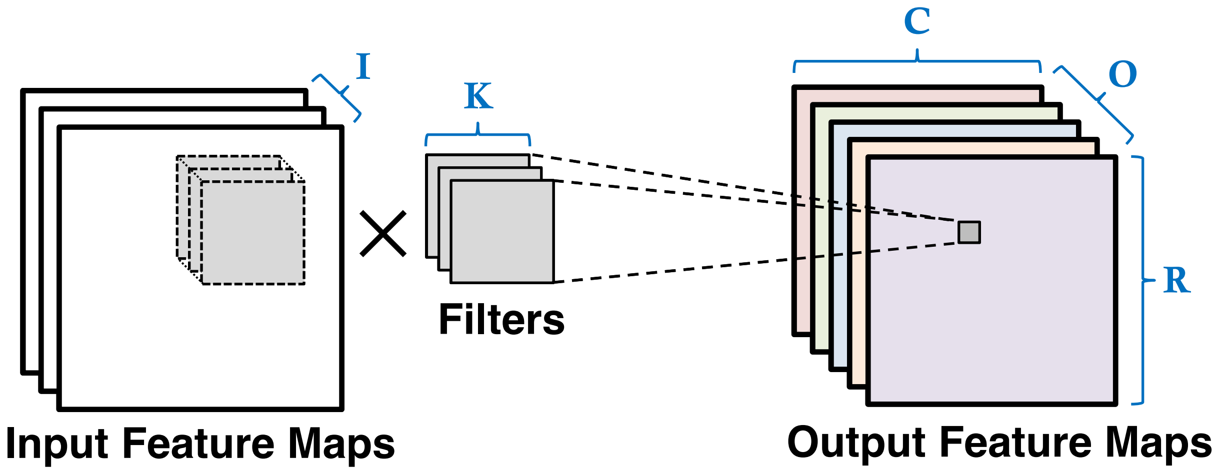
- An output pixel is connected to its neighboring region on each input feature map
- All pixels on an output feature map use the same filter weights

Parallelism in the Convolutional Layer



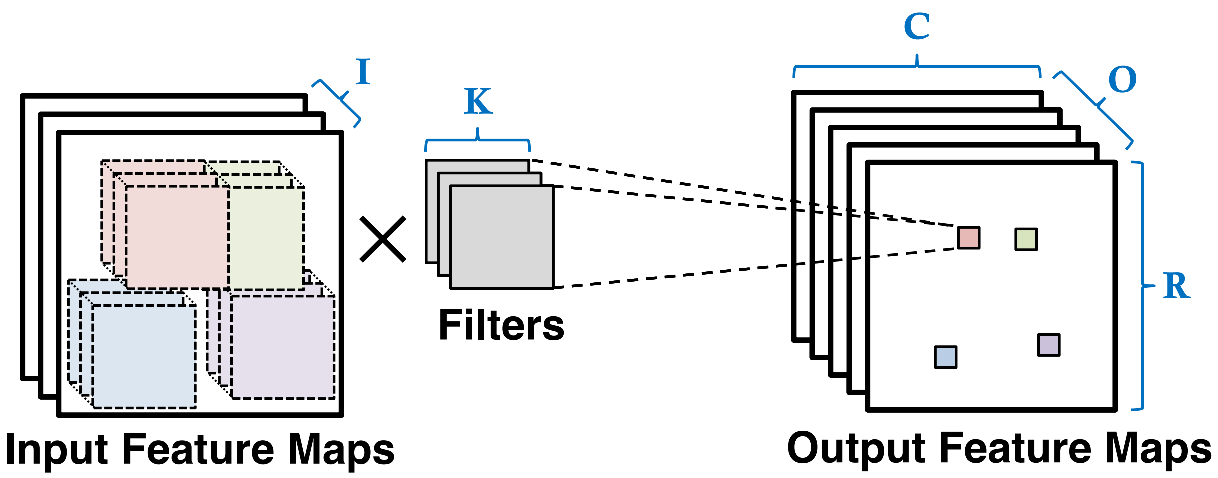
- Four main sources of parallelism
  1. **Across input feature maps**

Parallelism in the Convolutional Layer



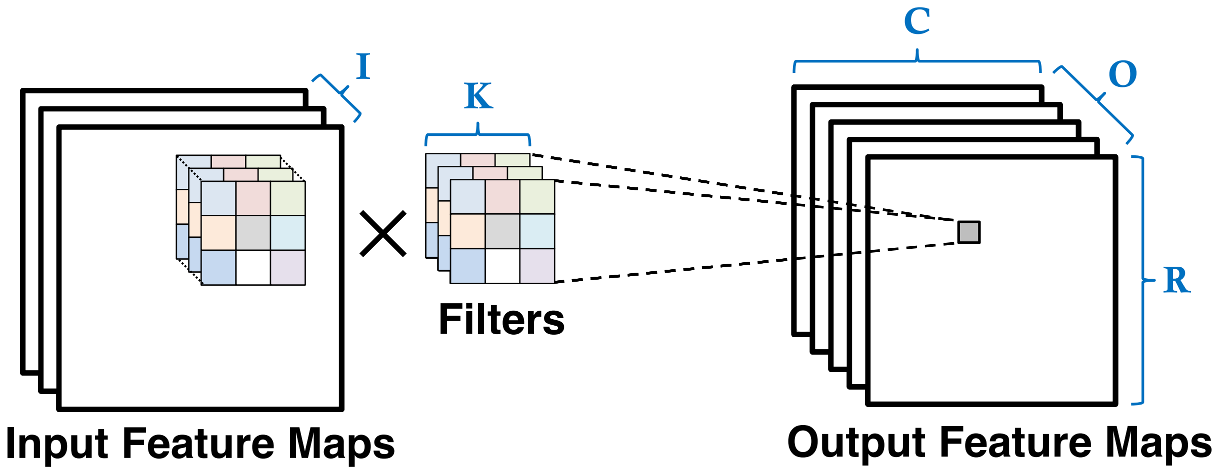
- Four main sources of parallelism
    1. Across input feature maps
    2. **Across output feature maps**

Parallelism in the Convolutional Layer



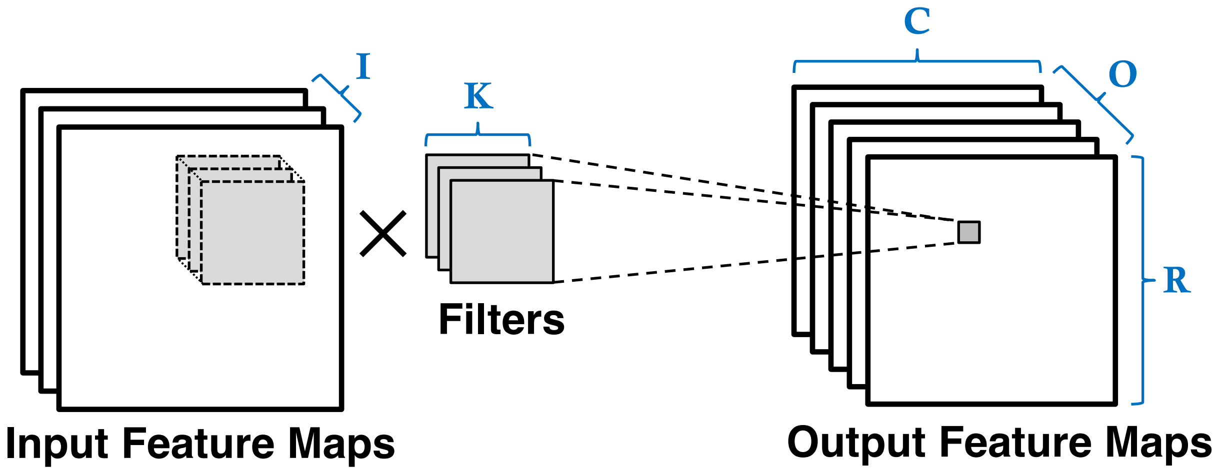
- Four main sources of parallelism
  1. Across input feature maps
  2. Across output feature maps
  3. **Across different output pixels (i.e. filter positions)**

Parallelism in the Convolutional Layer



- Four main sources of parallelism
  1. Across input feature maps
  2. Across output feature maps
  3. Across different output pixels (i.e. filter positions)
  4. **Across filter pixels**

Parallelism in the Code




```
for (row=0; row<R; row++) {
    for (col=0; col<C; col++) {
        for (to=0; to<O; to++) {
            for (ti=0; ti<I; ti++) {
                for (ki=0; ki<K; ki++) {
                    for (kj=0; kj<K; kj++) {
                        output_fm[to][row][col] +=
                        weights[to][ti][ki][kj]*input_fm[ti][S*row+ki][S*col+kj];
}}}}}}
```

Loop over output pixels

Loop over output feature maps

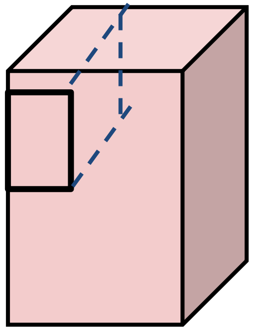
Loop over input feature maps

Loop over filter pixels

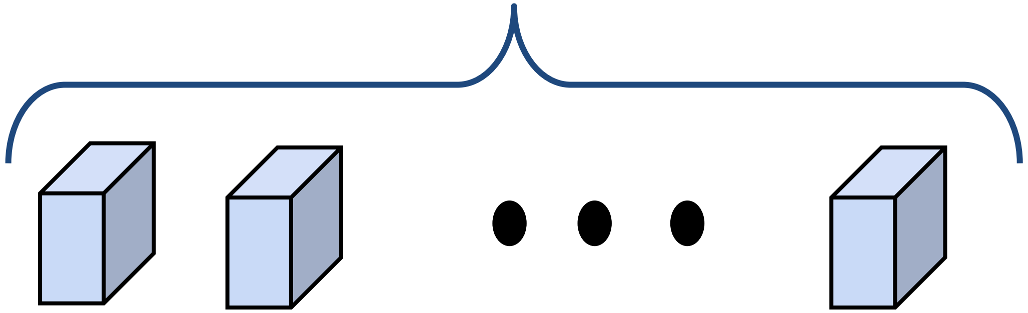
S is the stride

Convolution Implementation

Feature map: H x W x C

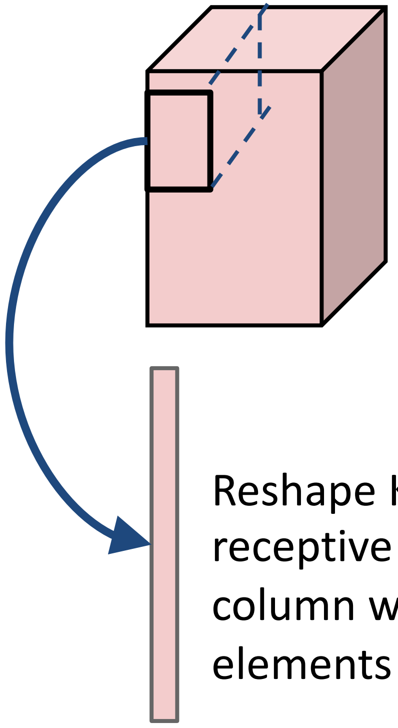


Conv weights: D filters, each K x K x C



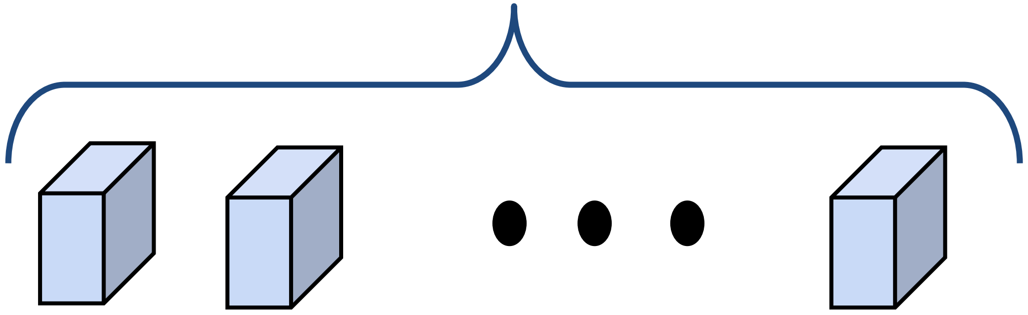
Convolution Implementation

Feature map: H x W x C



Reshape K x K x C receptive field to column with K^2C elements

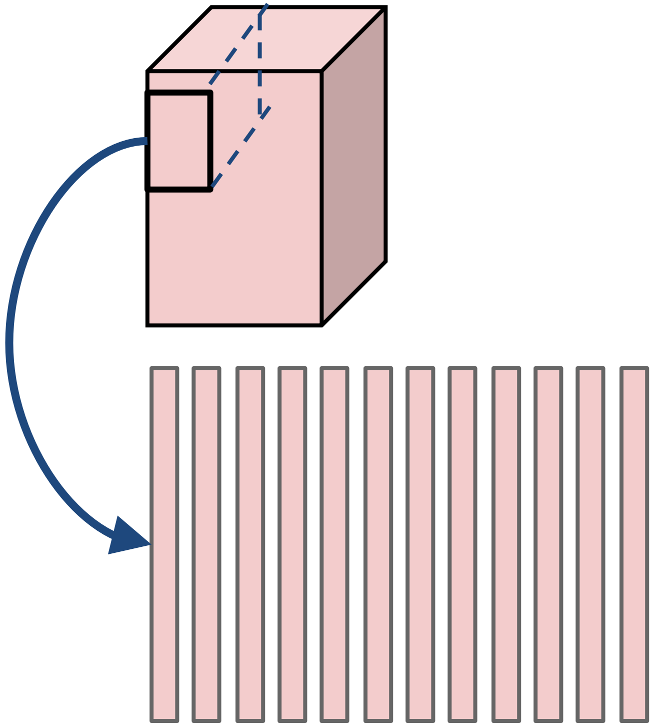
Conv weights: D filters, each K x K x C

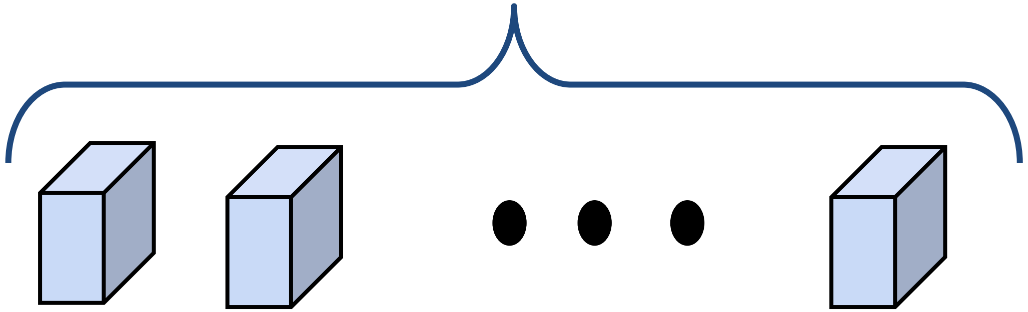


Convolution Implementation

Feature map: H x W x C

Conv weights: D filters, each K x K x C



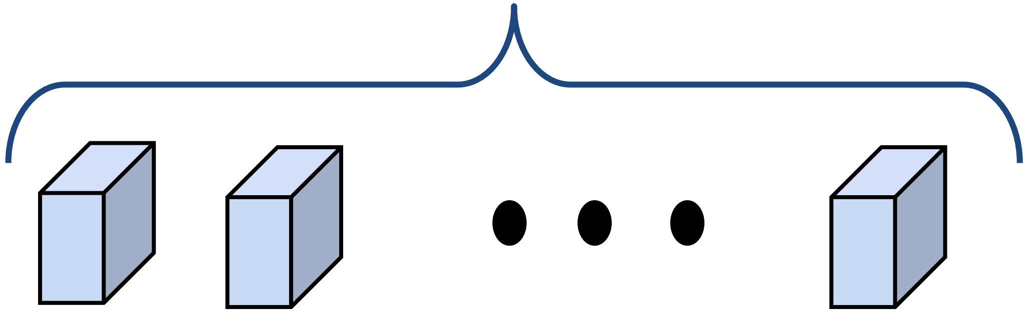


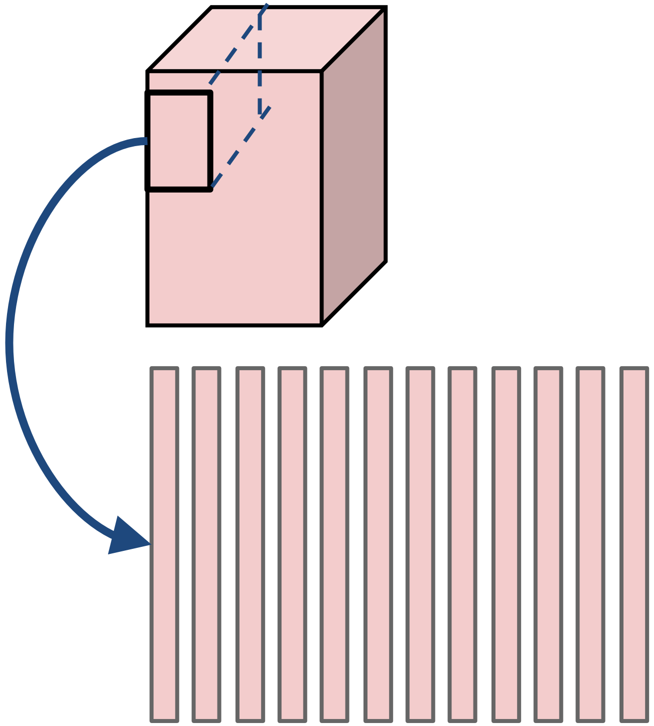
Repeat for all columns to get $(K^2C) \times N$ matrix
(N receptive field locations – i.e., total N values per output channel)

Convolution Implementation

Feature map: H x W x C

Conv weights: D filters, each K x K x C





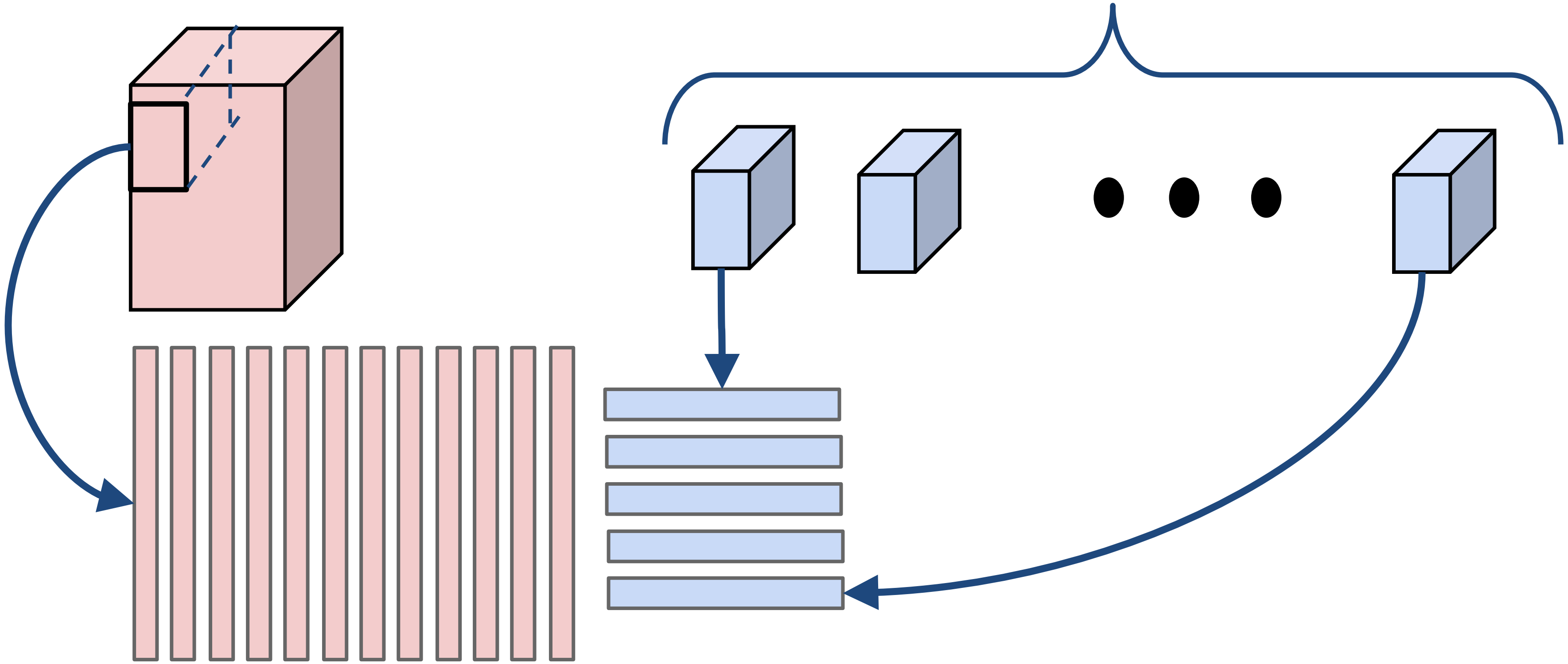
Elements appearing in multiple receptive fields are duplicated; this uses a lot of memory

Repeat for all columns to get (K^2C) x N matrix
(N receptive field locations)

Convolution Implementation

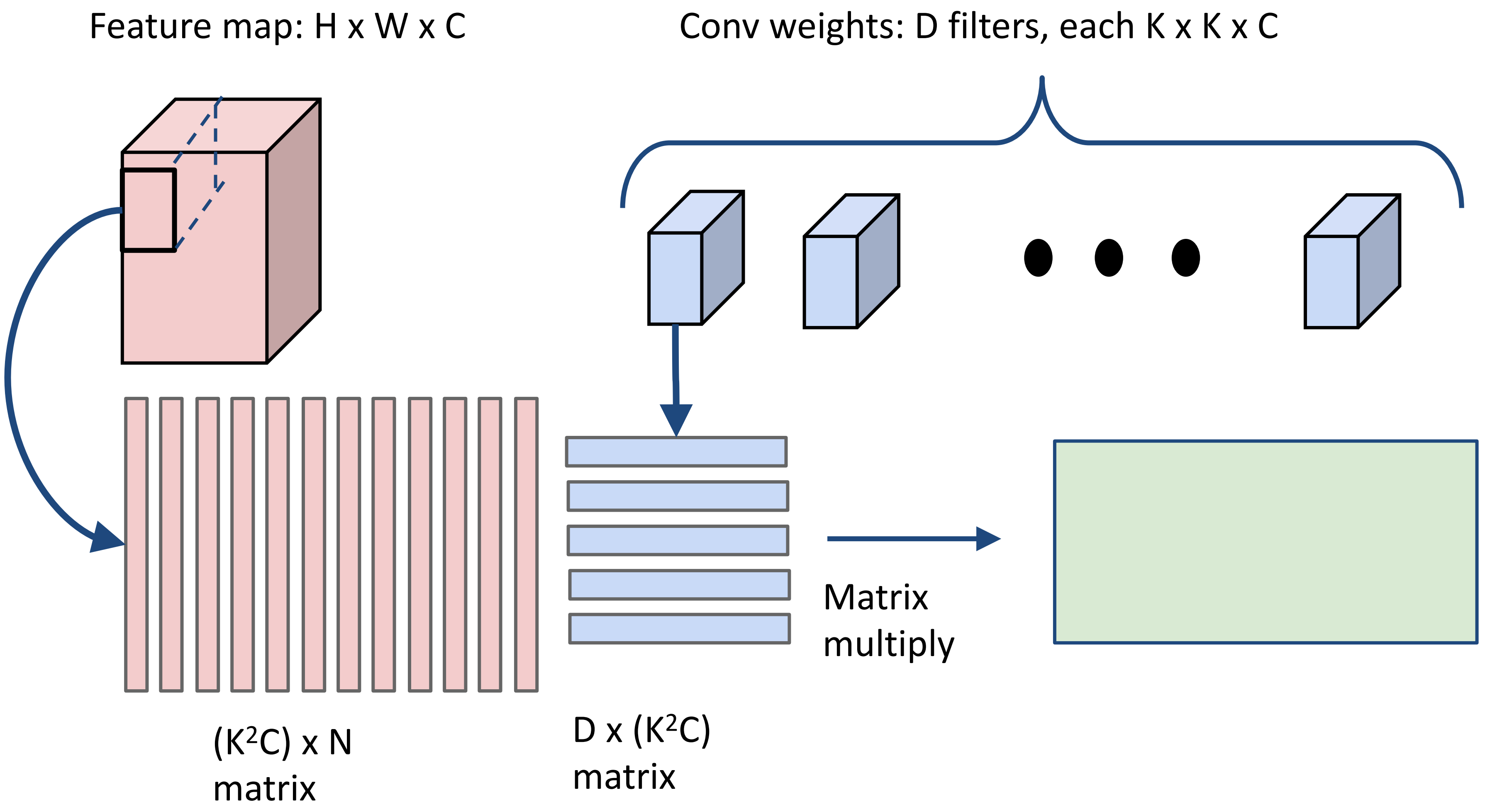
Feature map: H x W x C

Conv weights: D filters, each K x K x C



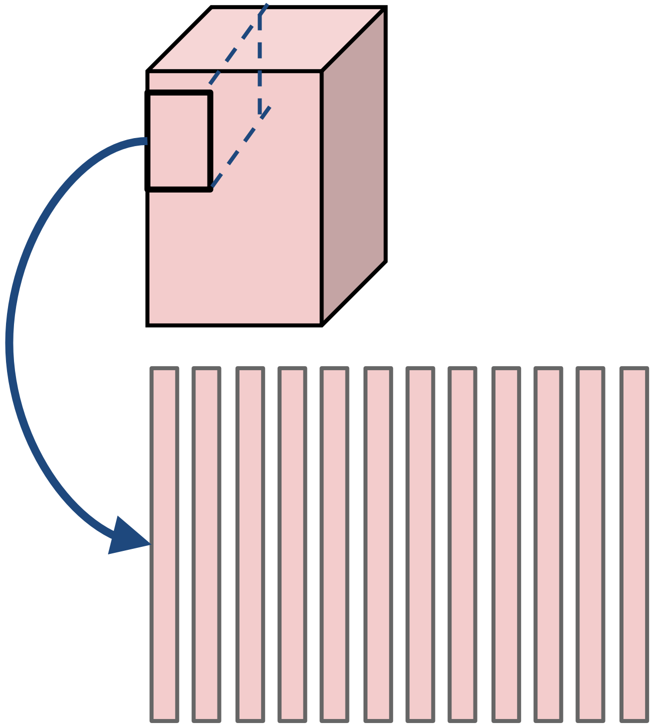
Reshape each filter to K^2C row,
making D x (K^2C) matrix

Convolution Implementation



Convolution Implementation

Feature map: H x W x C

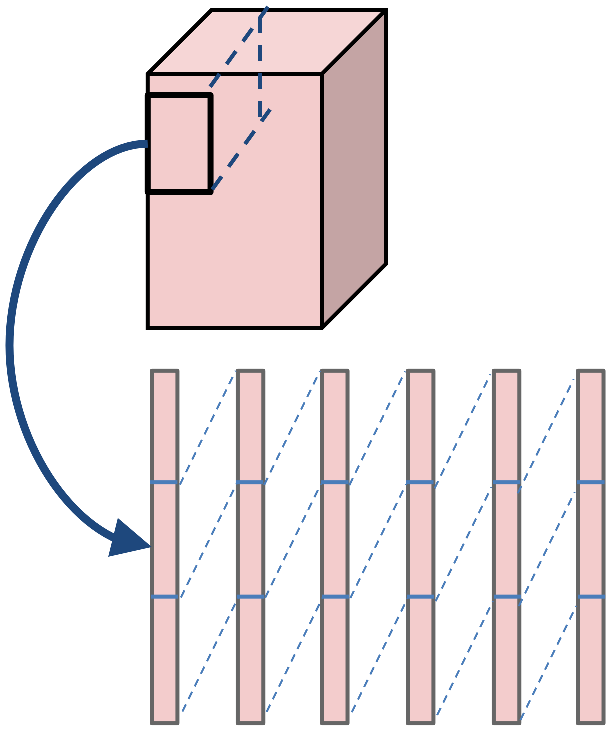


(K^2C) x N matrix

Bottleneck!

Convolution Implementation

Feature map: H x W x C



(K^2C) x N matrix

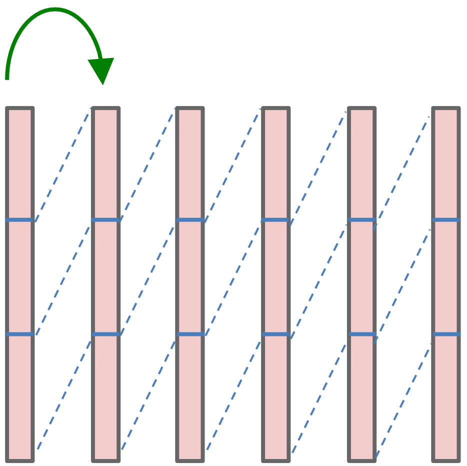
Bottleneck!

Note that columns overlap

If we group the elements by H or W, we get K blocks per column, a "sliding" pattern is evident

Convolution Implementation

When we call matrix multiply, we specify a "stride" between columns.
Normally this is the actual column height



Note that columns overlap

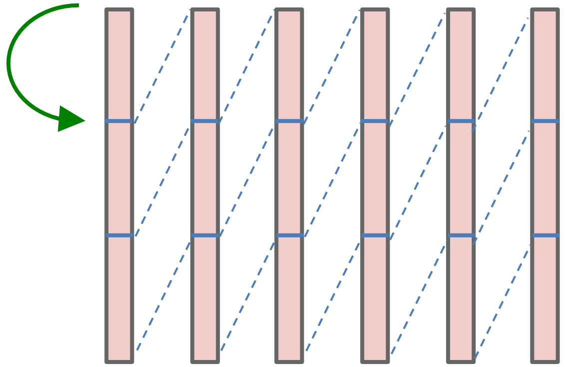
If we group the elements by H or W, we get K blocks per column, a "sliding" pattern is evident

(K^2C) x N matrix

Bottleneck!

Convolution Implementation

But if we instead call MM with a “column” separation which is the height of one of the K blocks, we can perform the same operation with less memory

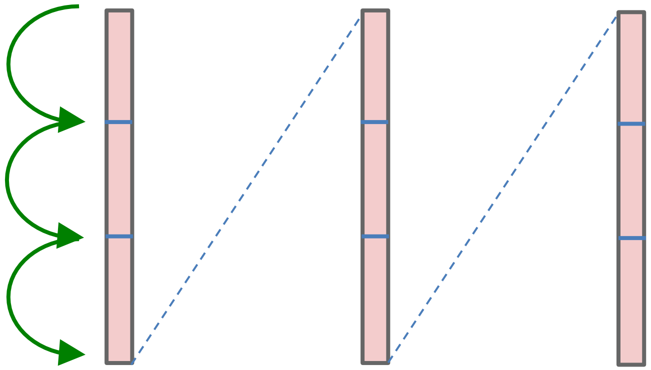


(K^2C) x N matrix

Bottleneck!

Convolution Implementation

But if we instead call MM with a “column” separation which is the height of one of the K blocks, we can perform the same operation with less memory (reduce column count by K)

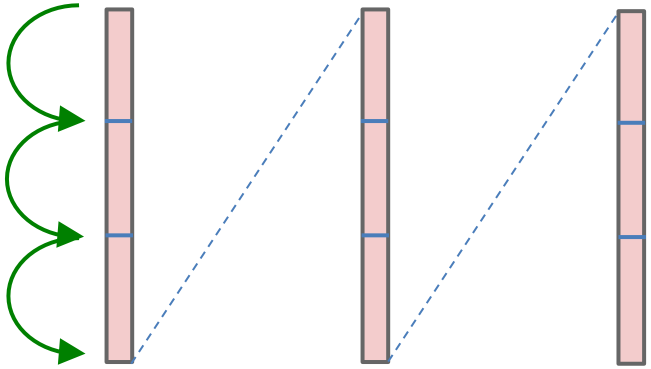


(KC) x N matrix

Better!

Convolution Implementation

You can also perform the convolution "in place" – without an additional memory buffer, using K matrix multiplies



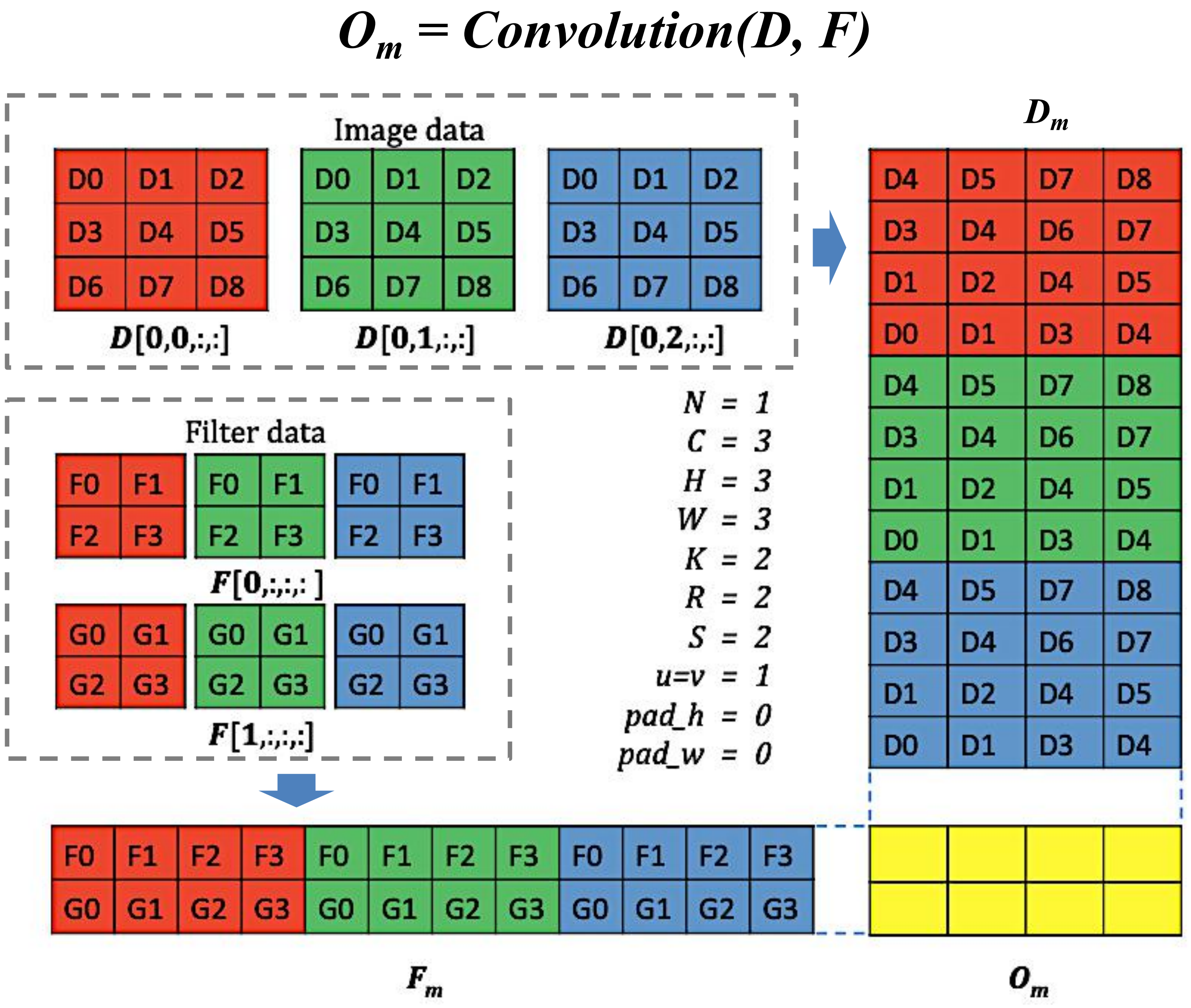
C x N matrix

Best!

[cs231n Li, Johnson]

Convolutions to Matrix Multiplication

- Convolutions lowered onto matrix multiplication.
- Fixed sized submatrices of inputs D_m and F_m are moved on-chip
- Lazily materialize D_m
- Compute on tiles of D_m and F_m while fetching the next tiles off-chip memory into on-chip caches and other memories.
- Compute pipelining hides data transfer latency.



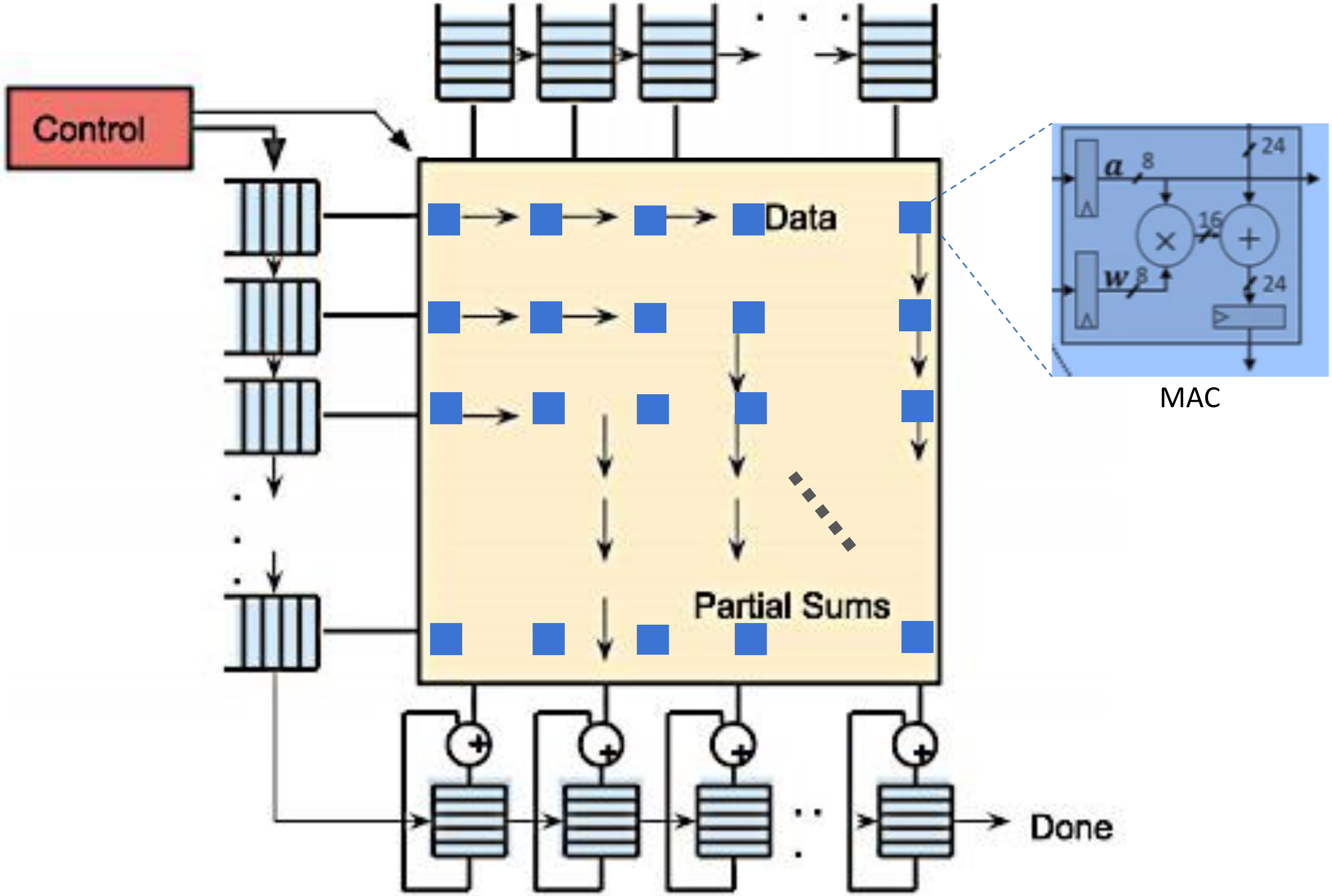
[Chetlur et al. cuDNN arxiv 1410.0759]

2-D Grid of Processing Elements

Systolic Arrays

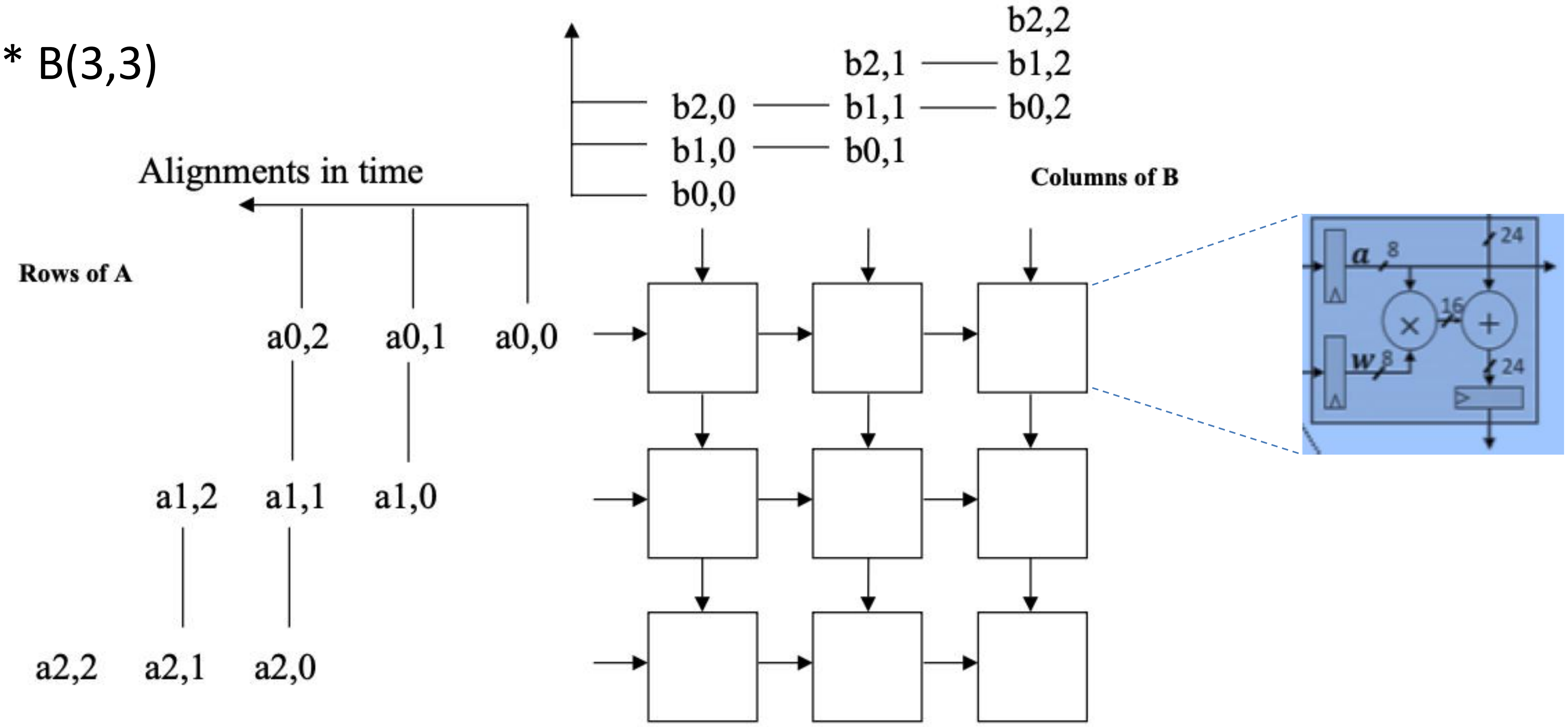
- Balance compute and I/O
- Local Communication
- Concurrency

- M1*M2 nxn mult in 2n cycles

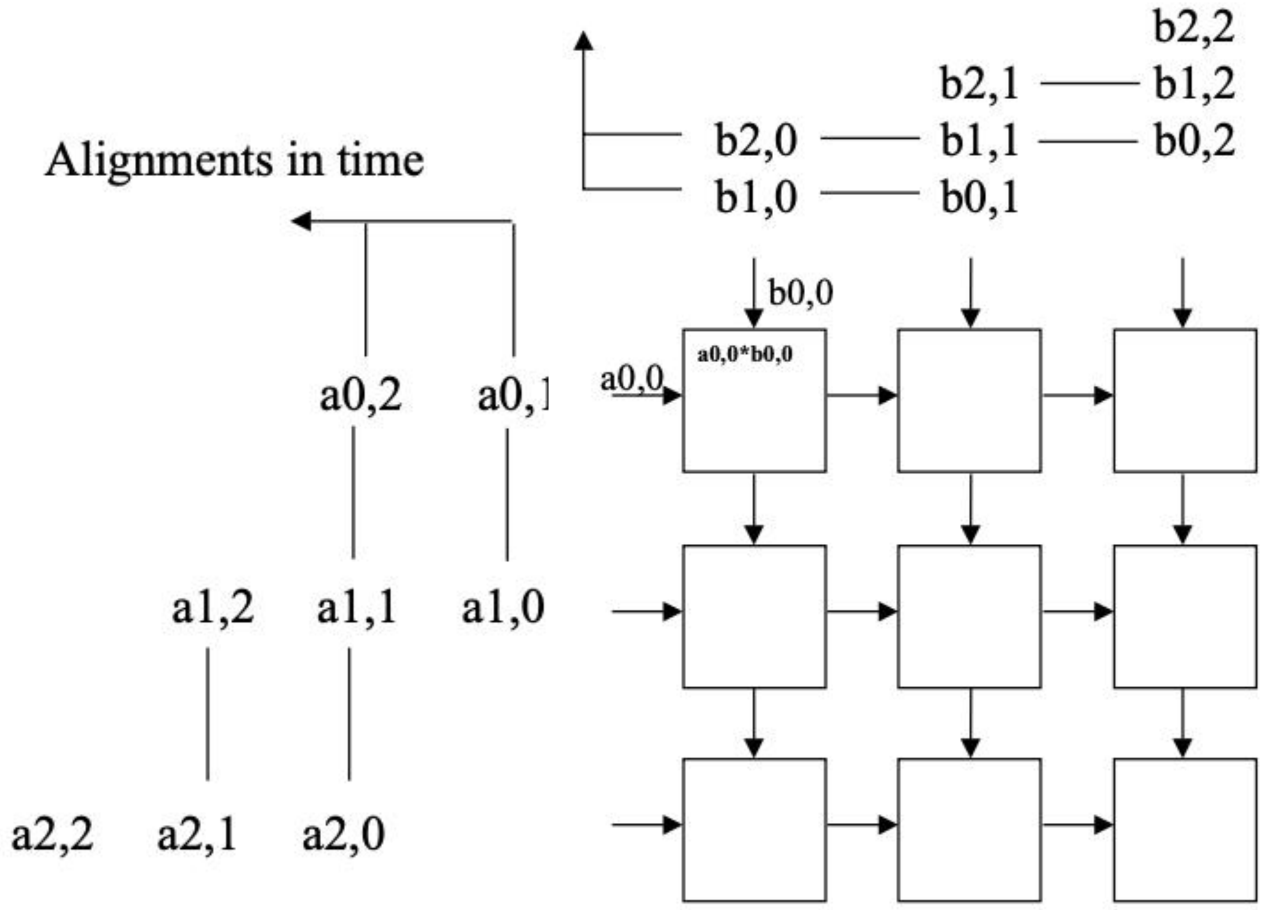


3x3 Systolic Matrix Multiplication

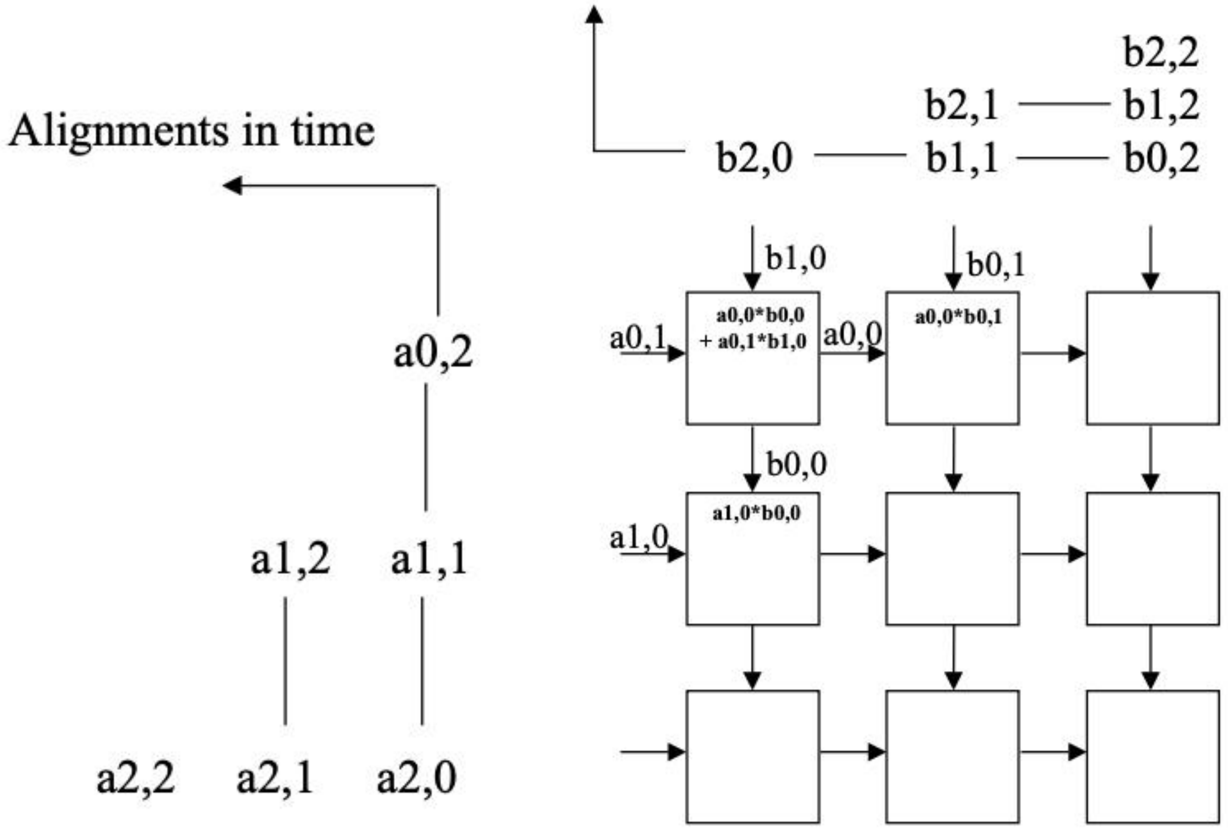
Y = A (3,3) * B(3,3)



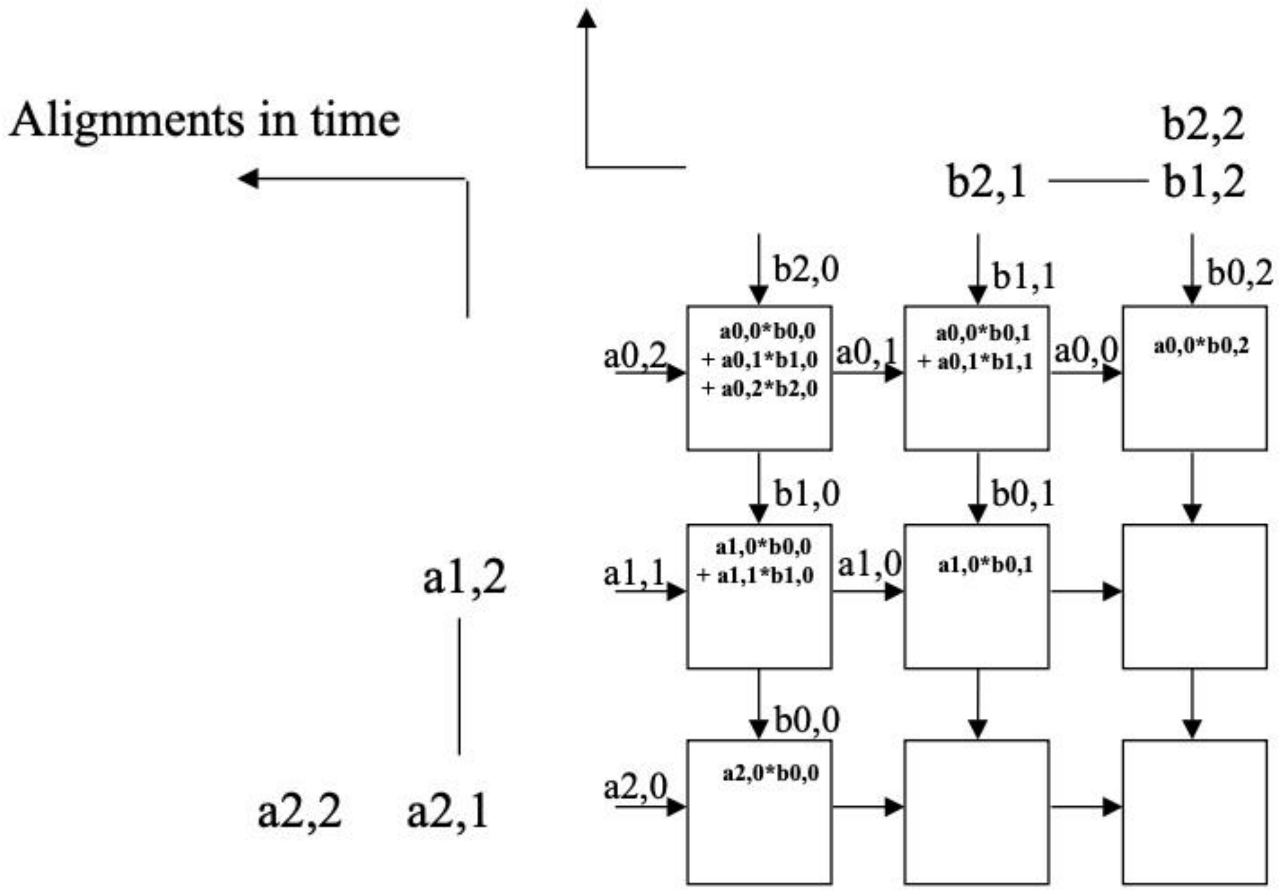
3x3 Systolic Matrix Multiplication



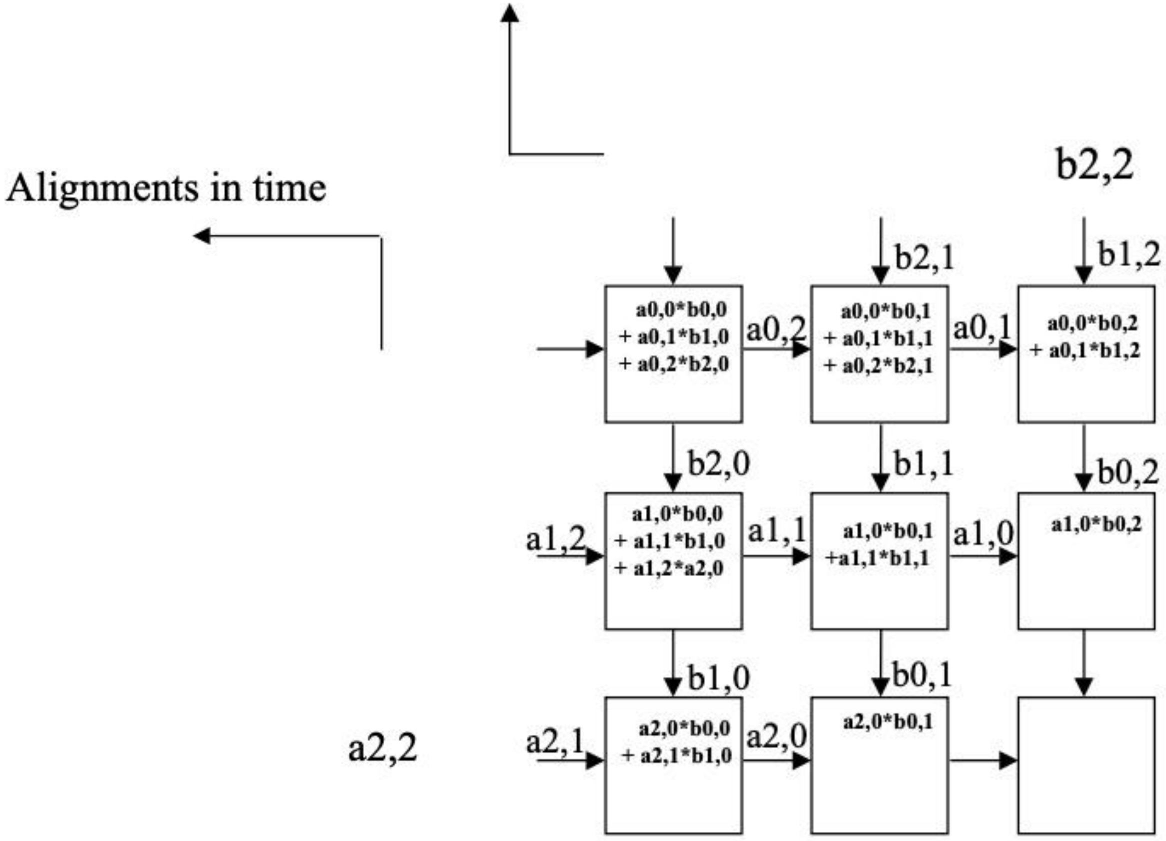
3x3 Systolic Matrix Multiplication



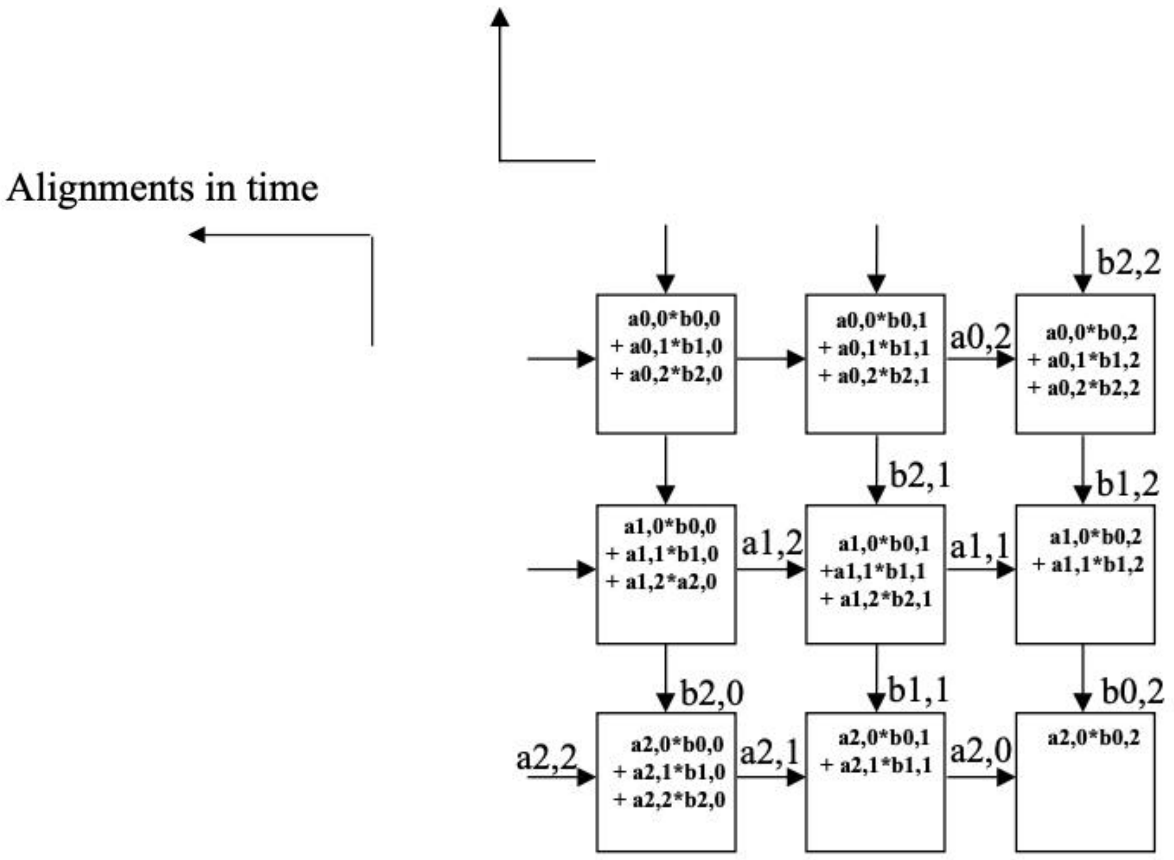
3x3 Systolic Matrix Multiplication



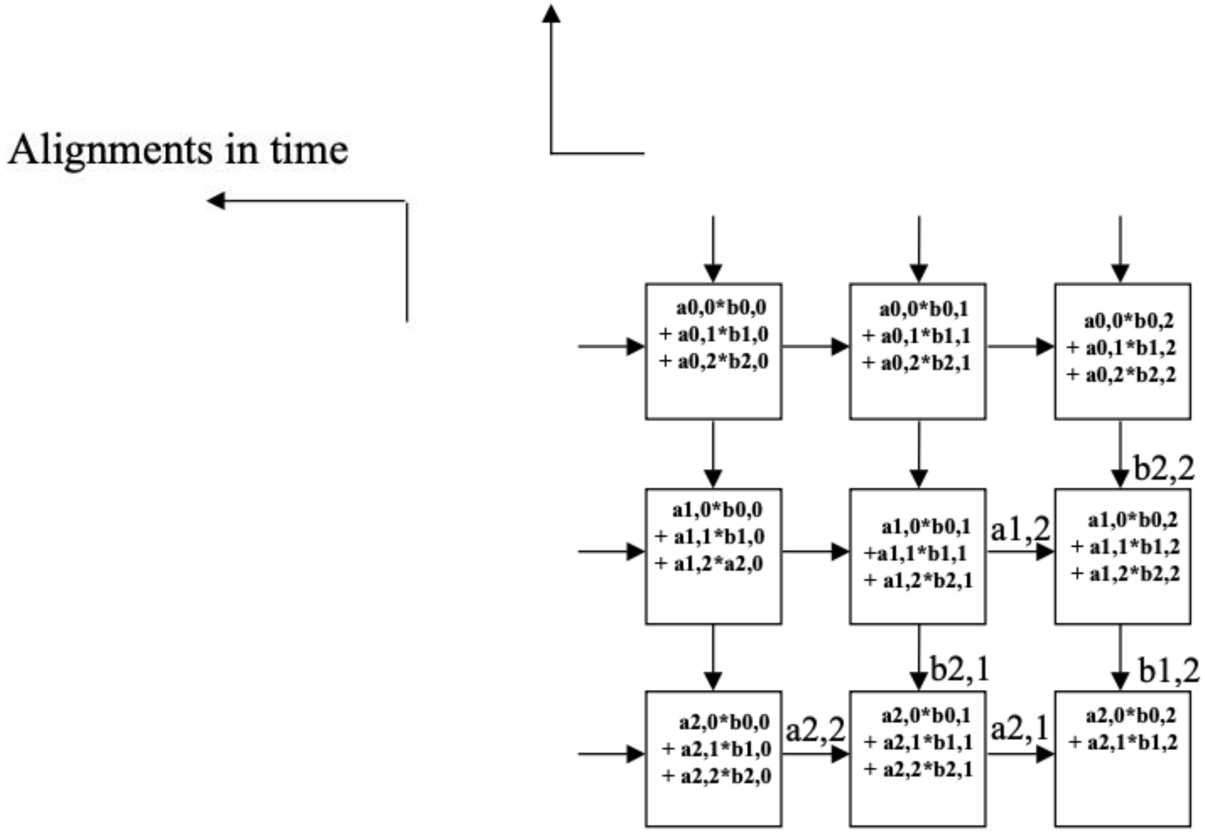
3x3 Systolic Matrix Multiplication



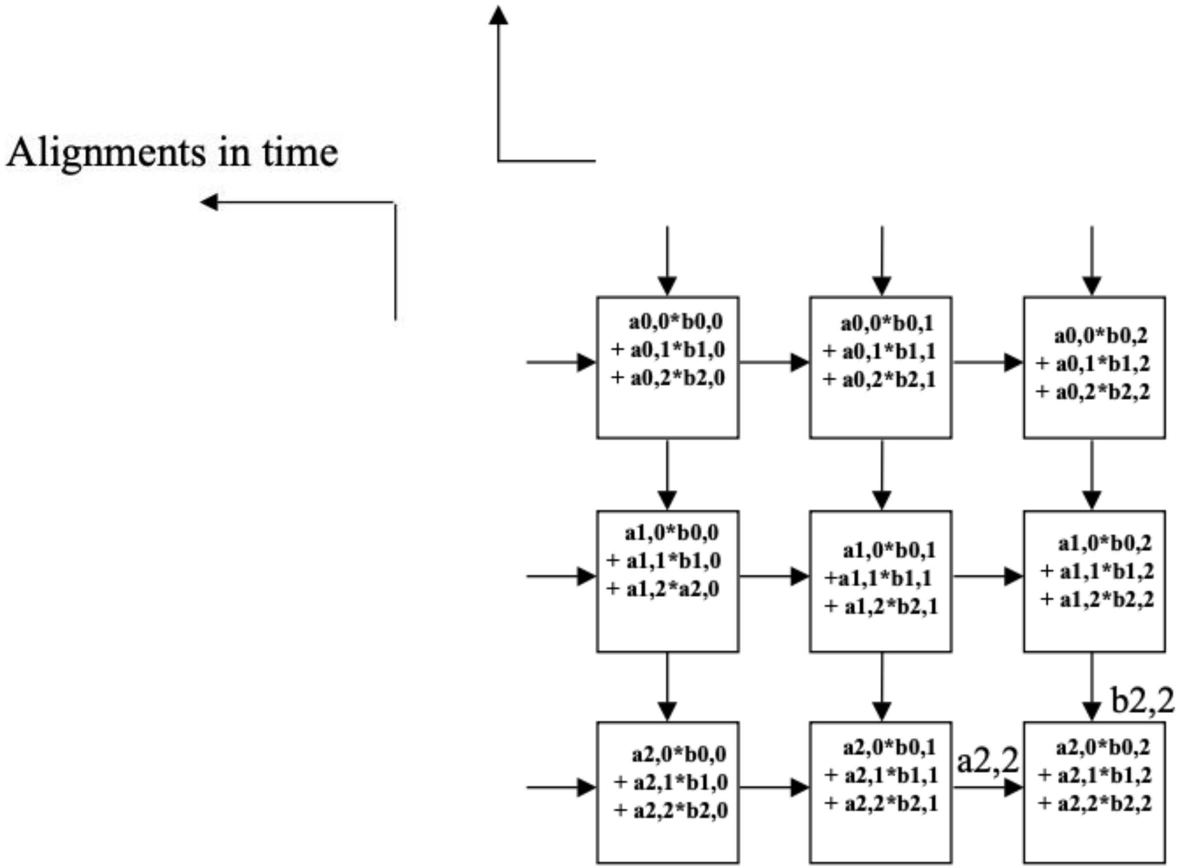
3x3 Systolic Matrix Multiplication



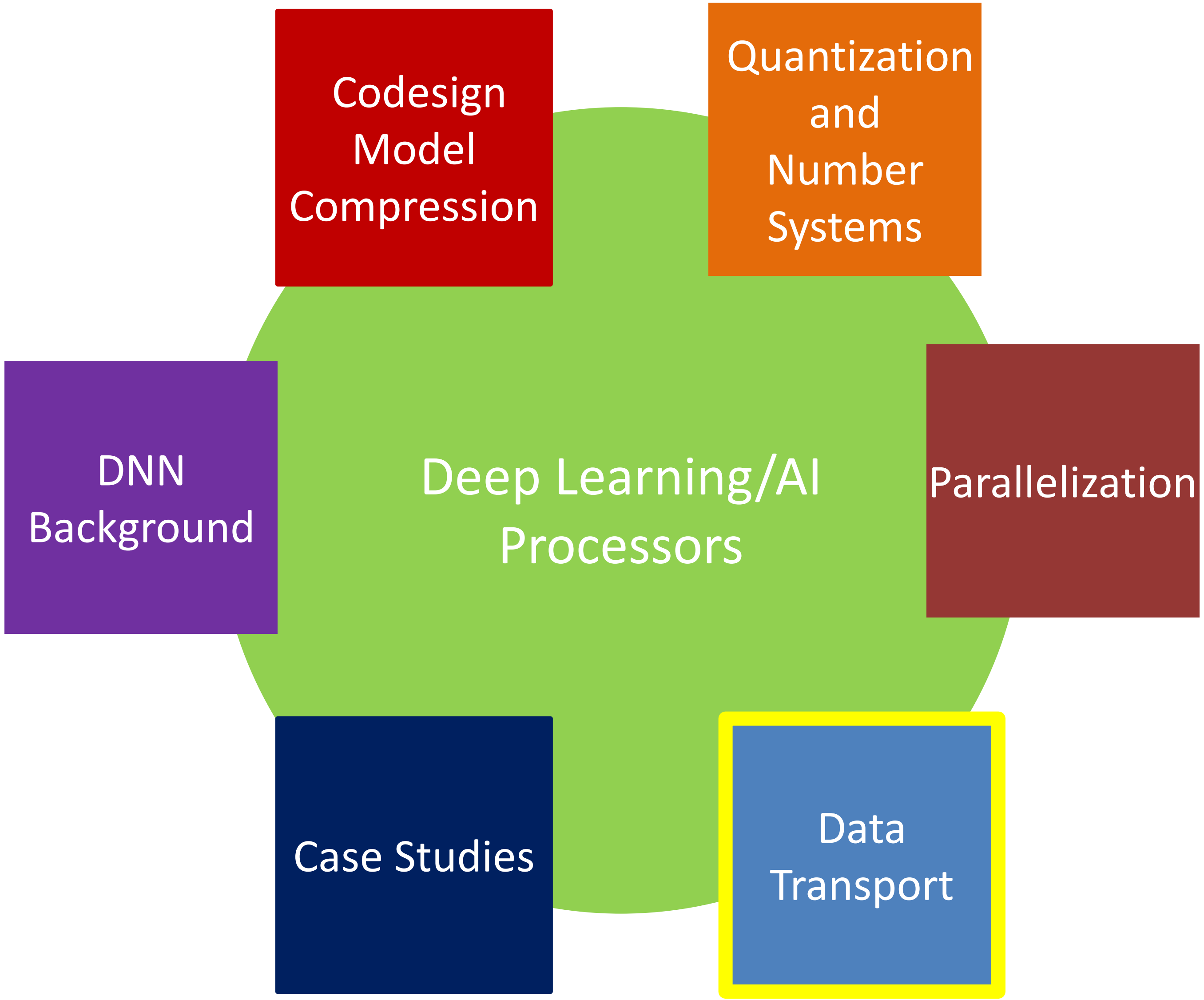
3x3 Systolic Matrix Multiplication



3x3 Systolic Matrix Multiplication

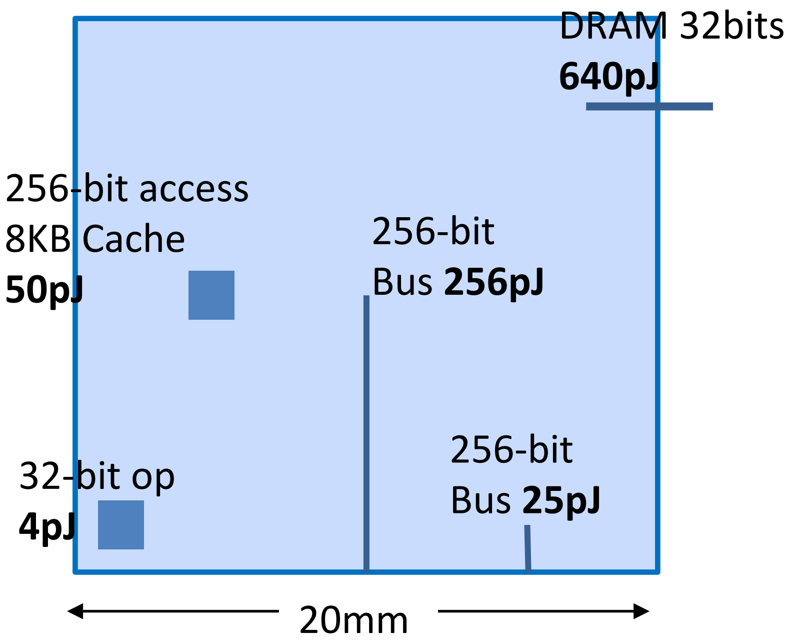


- Y = A * B
- 2n+1 Steps

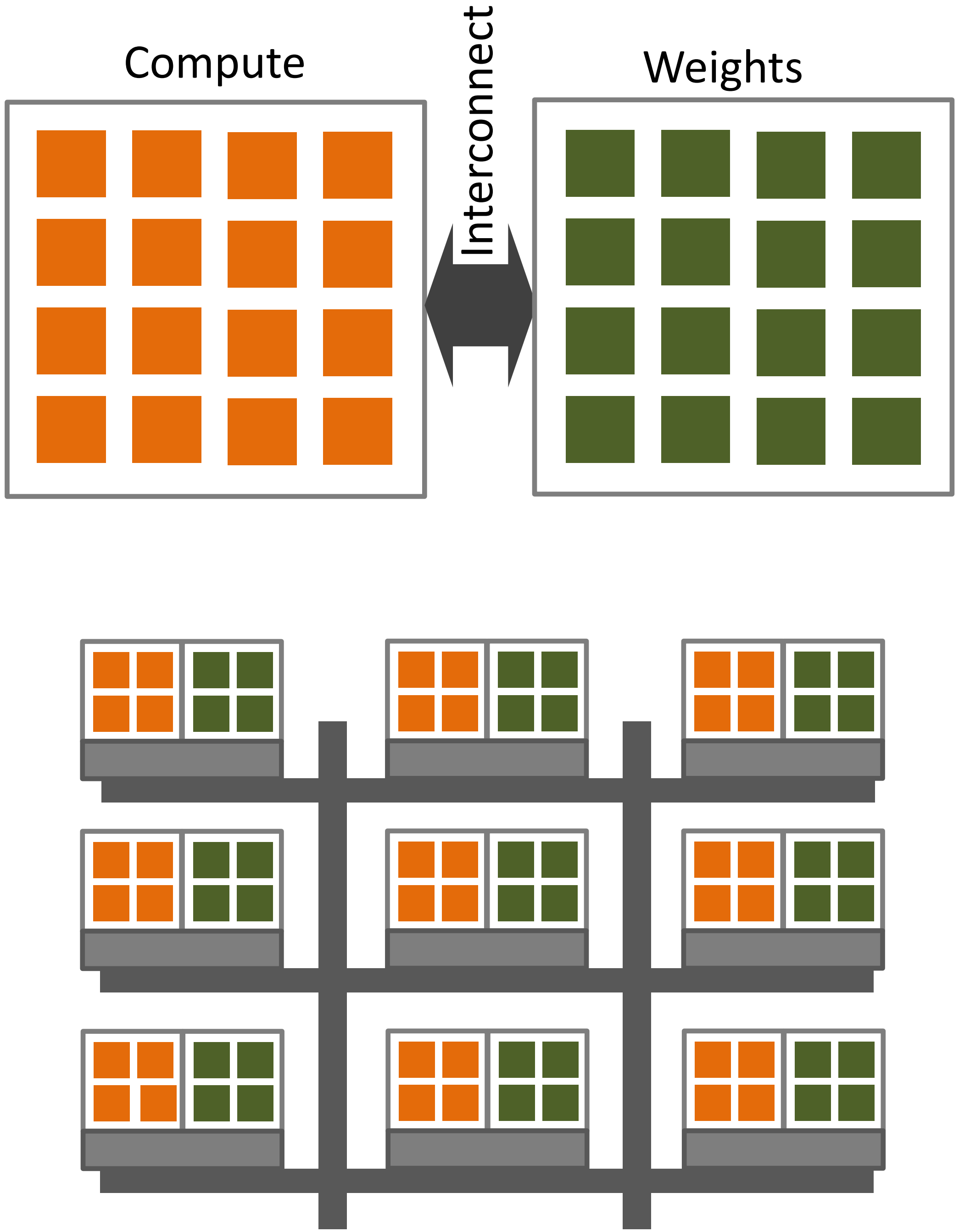

Codesign Model Compression
Quantization and Number Systems
DNN Background
Deep Learning/AI Processors
Parallelization
Case Studies
Data Transport

Memory Subsystems

Memory Locality

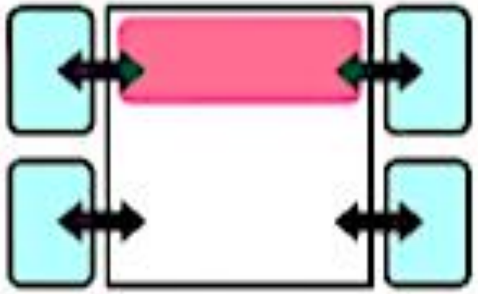


Bring Compute Closer to Memory



Memory and inter-chip communication advances

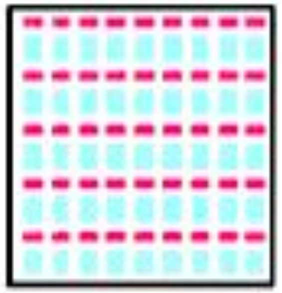
HBM2 on interposer



64pJ/B
16GB
900GB/s @ 60W

SRAM on chip

10pJ/B
256MB
6TB/s @ 60W

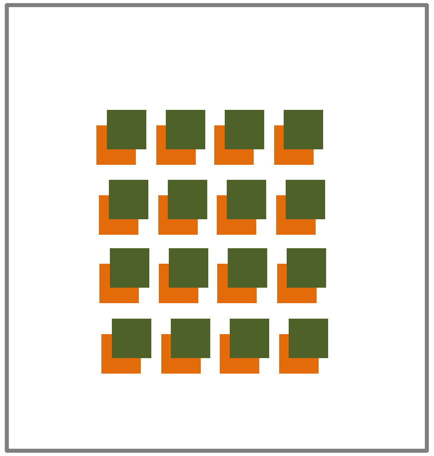


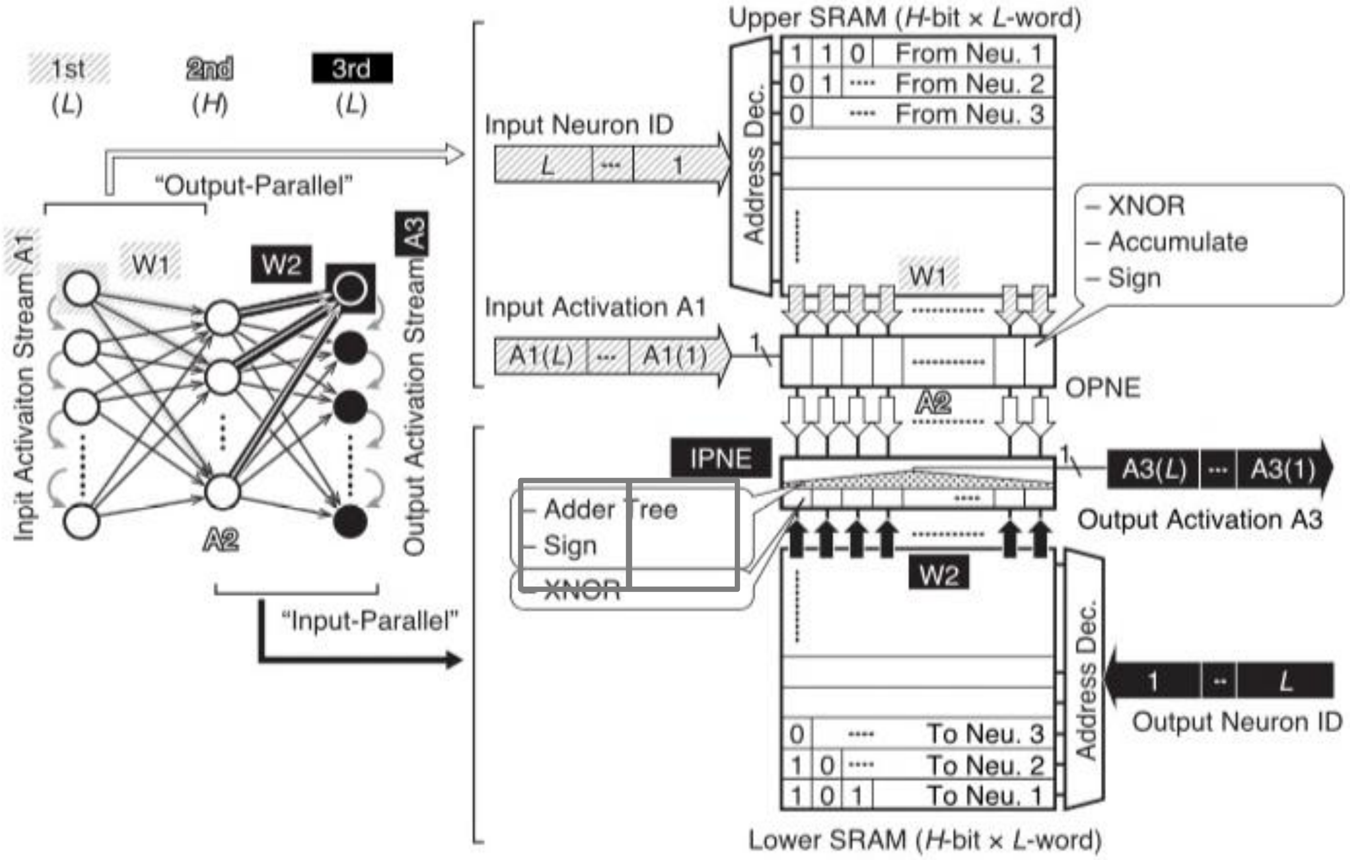
1pJ/B
1000 x 256KB
60TB/s @ 60W

Memory power density
Is ~25% of logic power density

[Graphcore 2018]

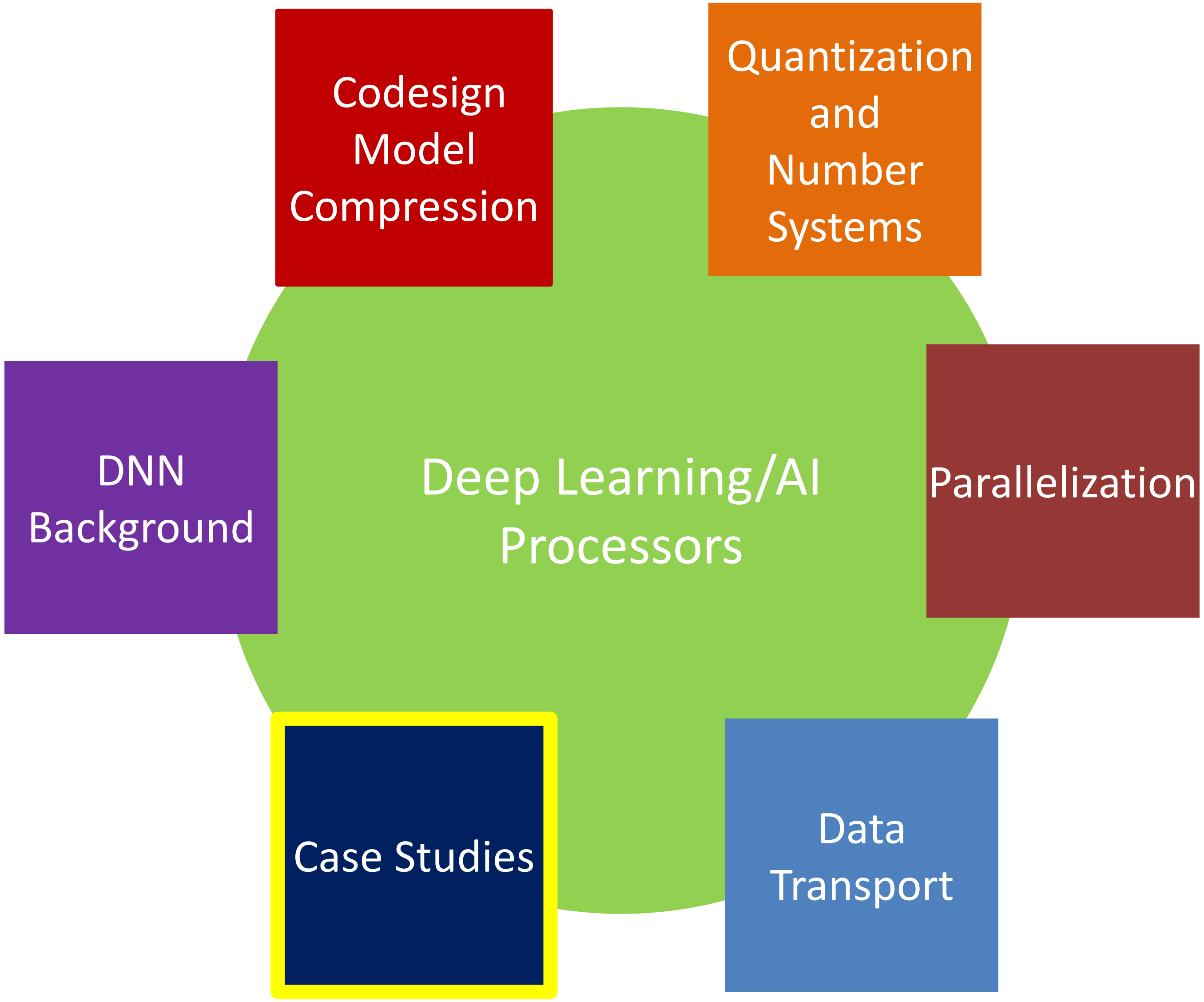
In-Memory Compute





In-memory compute with low-bits/value, low cost ops

[Ando et al., JSSC 04/18]



Codesign Model Compression
Quantization and Number Systems
DNN Background
Deep Learning/AI Processors
Parallelization
Case Studies
Data Transport

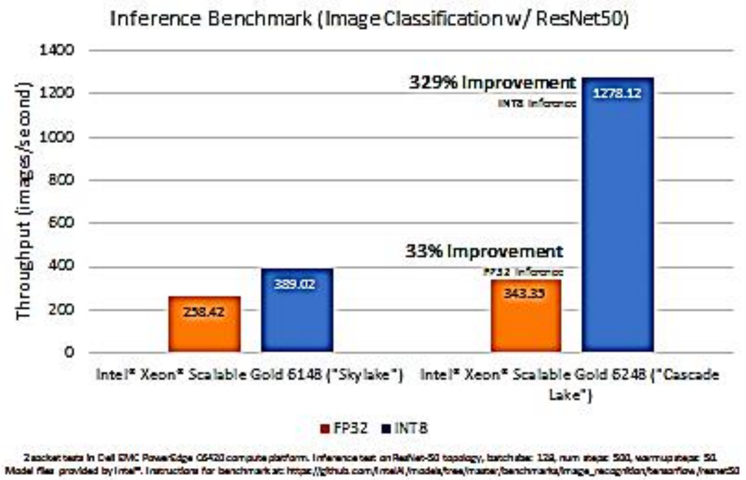
Putting it all together

CPU Based Acceleration

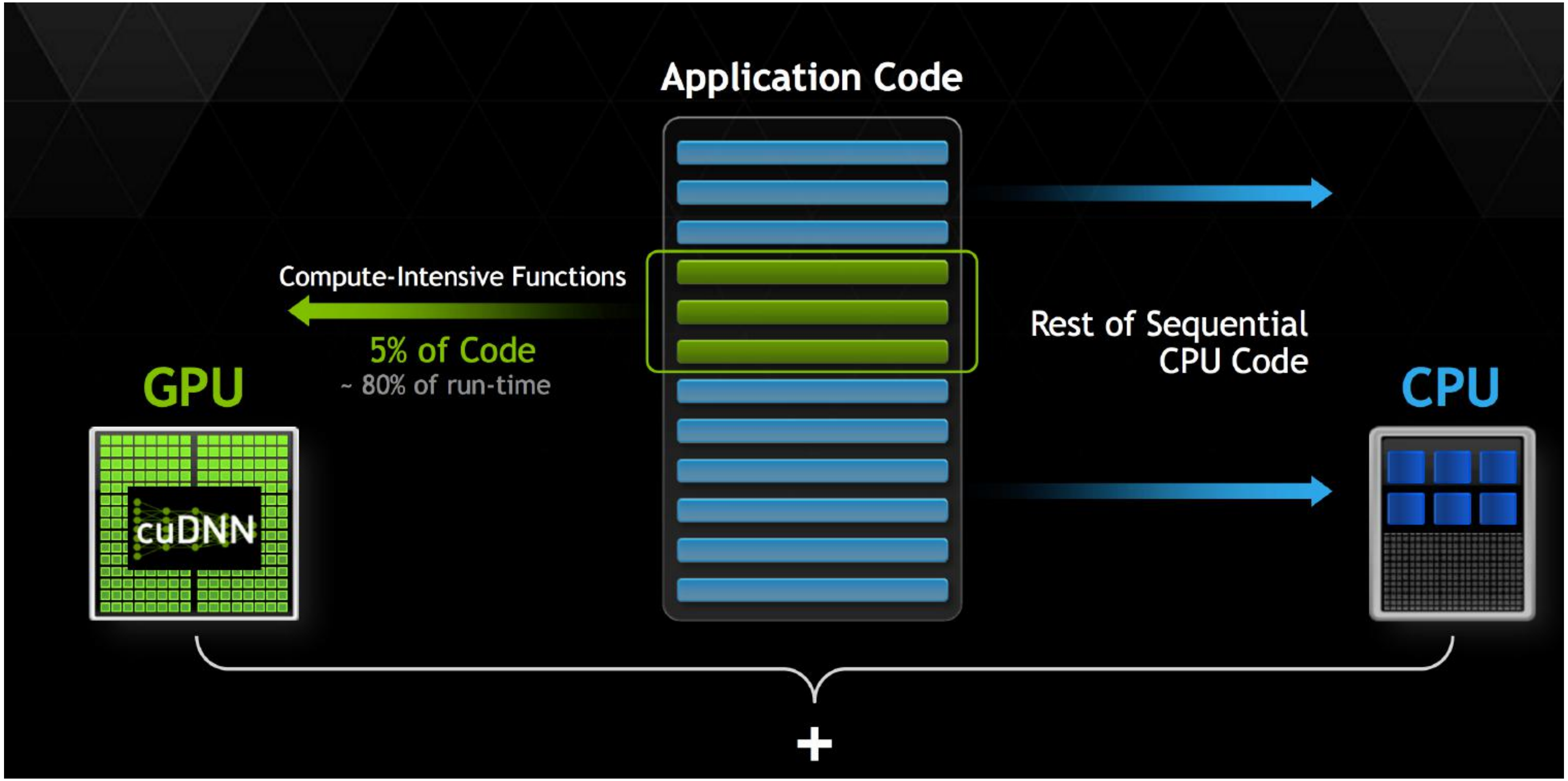
- All ML and DL algorithms start and end in machine subsystem
- Multi-core/multi-thread systems are common nowadays
- Processor have high speed bandwidth to memory subsystem
- Hard to beat Numpy math library
- But you end up using all machine resources for a single user

Intel® CPU Outperforms NVIDIA* GPU on ResNet-50 Deep Learning Inference

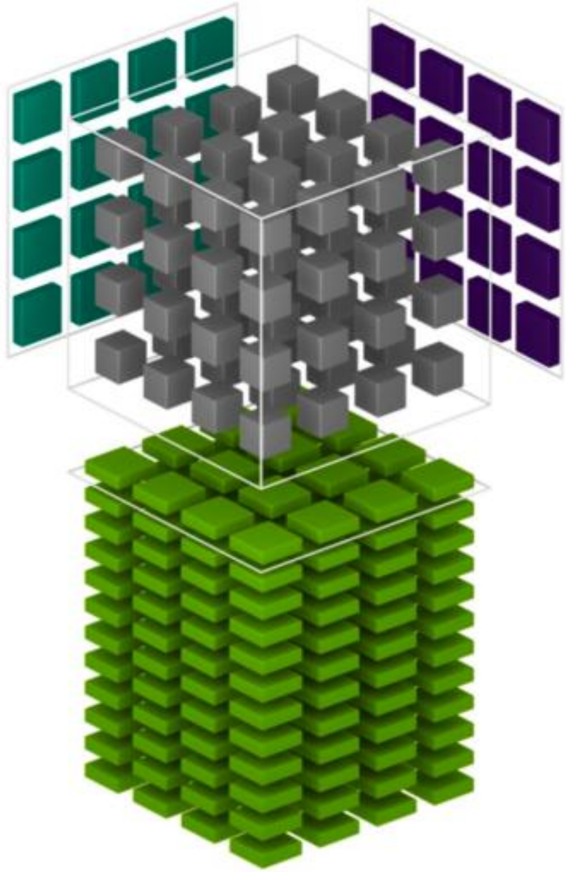
By Haihao Shen, Feng Tian, Xu Deng, Cong Xu, Andres Rodriguez, Indu K., Wei Li, published on May 13, 2019

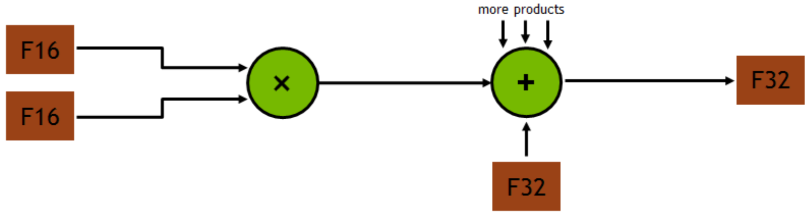


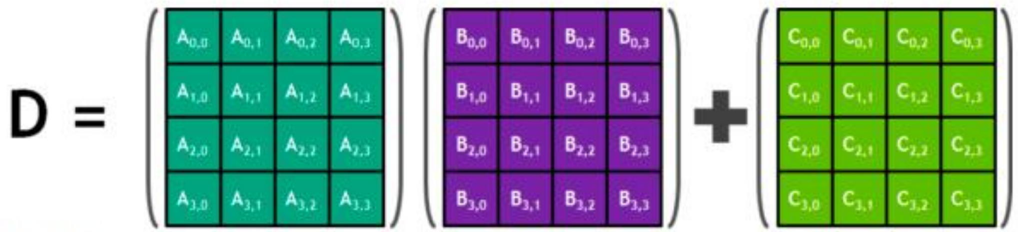
GPUs Based Acceleration



Tensor Core

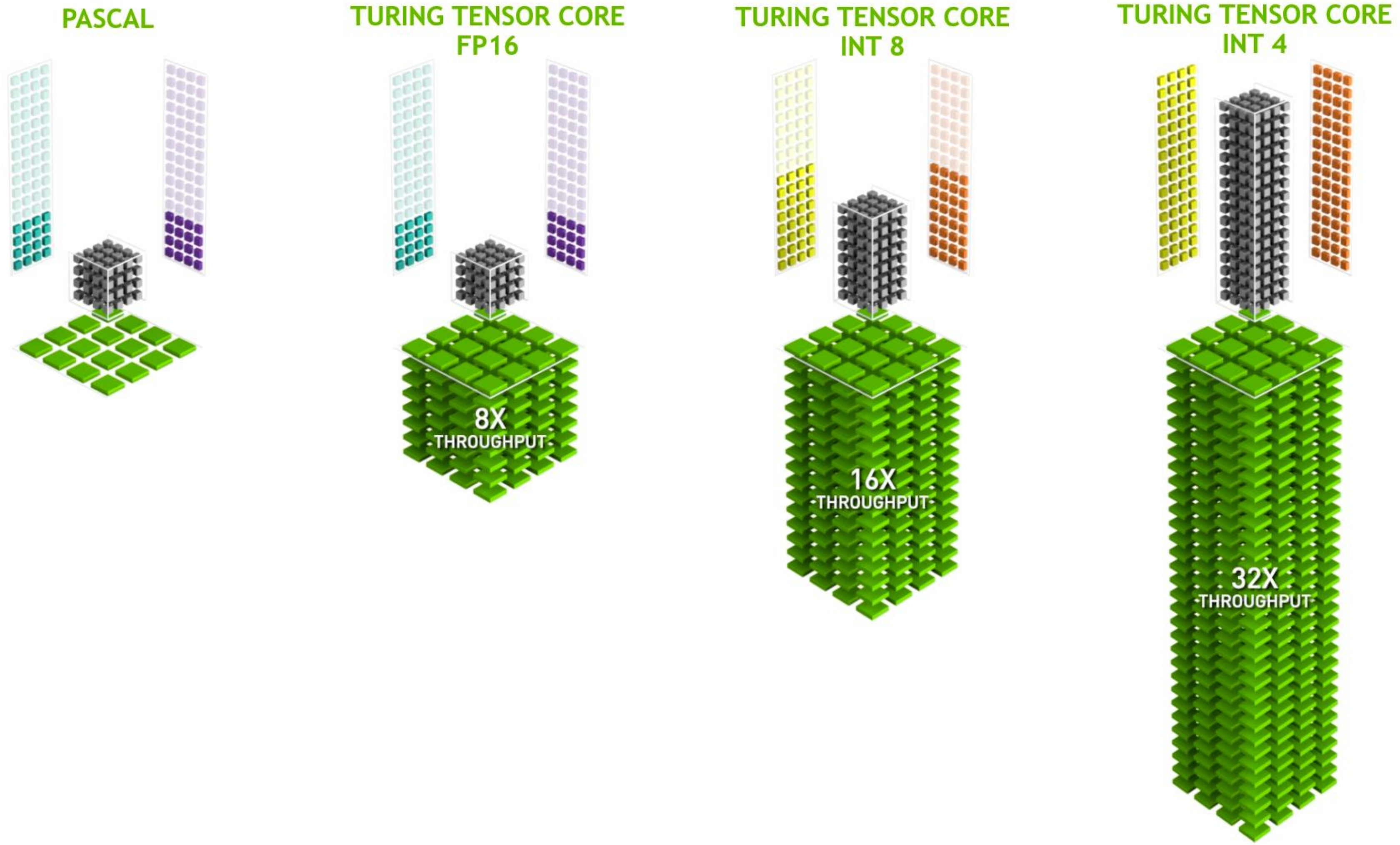




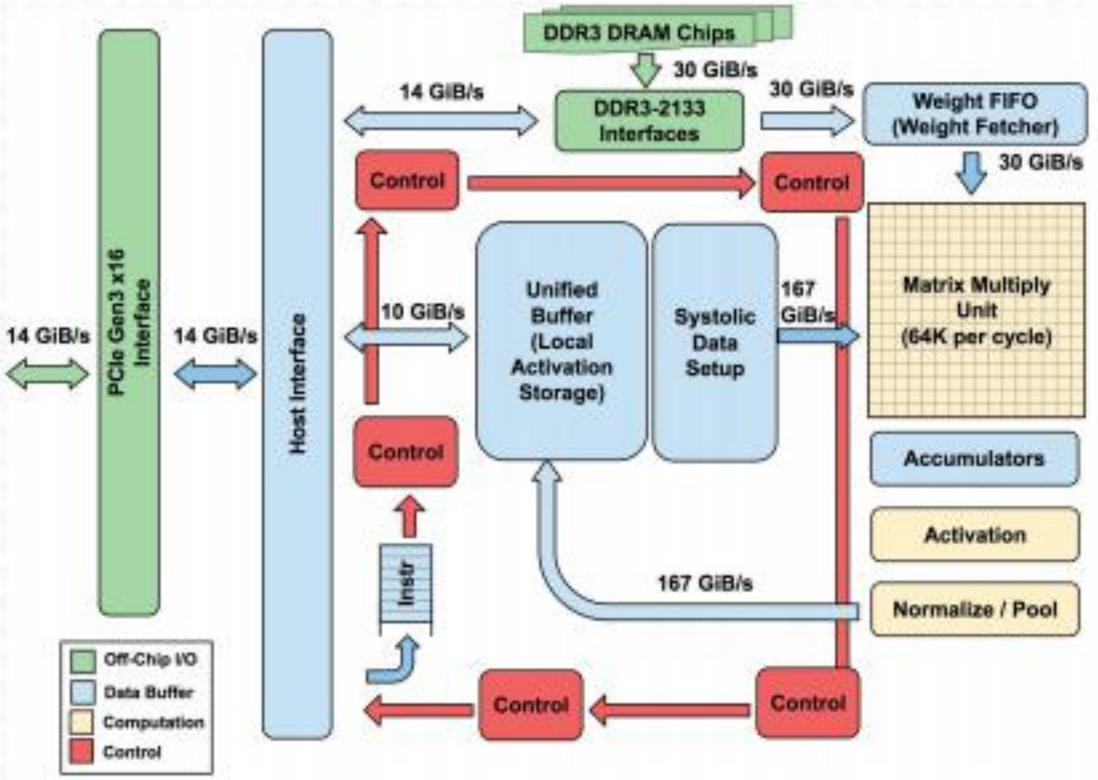


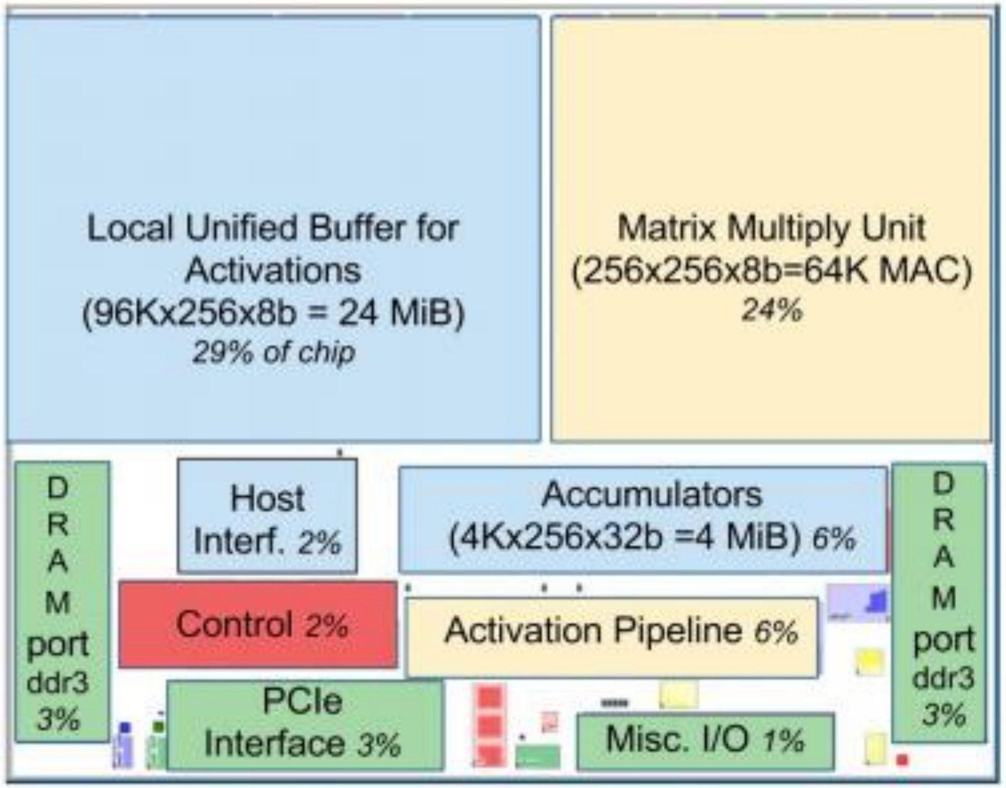
https://devblogs.nvidia.com/tensor-core-ai-performance-milestones/

Data Types in Tensor Core

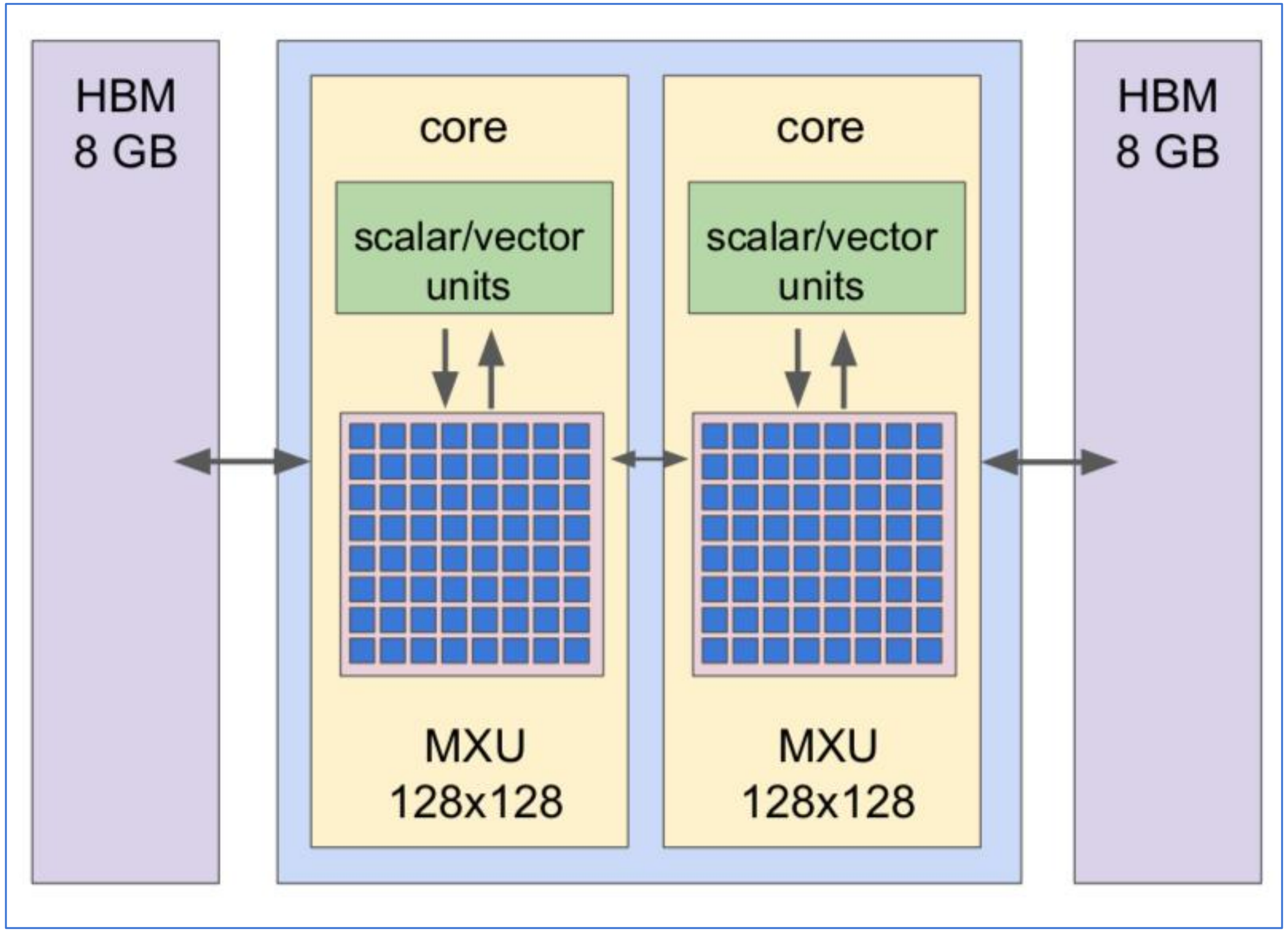


TPUv1


DDR3 DRAM Chips
30 GiB/s
14 GiB/s
DDR3-2133 Interfaces
30 GiB/s
Weight FIFO (Weight Fetcher)
30 GiB/s
Control
Control
14 GiB/s
PCIe Gen3 x16 Interface
14 GiB/s
Host Interface
10 GiB/s
Unified Buffer (Local Activation Storage)
Systolic Data Setup
167 GiB/s
Matrix Multiply Unit (64K per cycle)
Accumulators
Activation
Normalize / Pool
Control
Instr
167 GiB/s
Control
Control
Off-Chip I/O
Data Buffer
Computation
Control


Local Unified Buffer for Activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix Multiply Unit (256x256x8b=64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b =4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation Pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

TPUv2

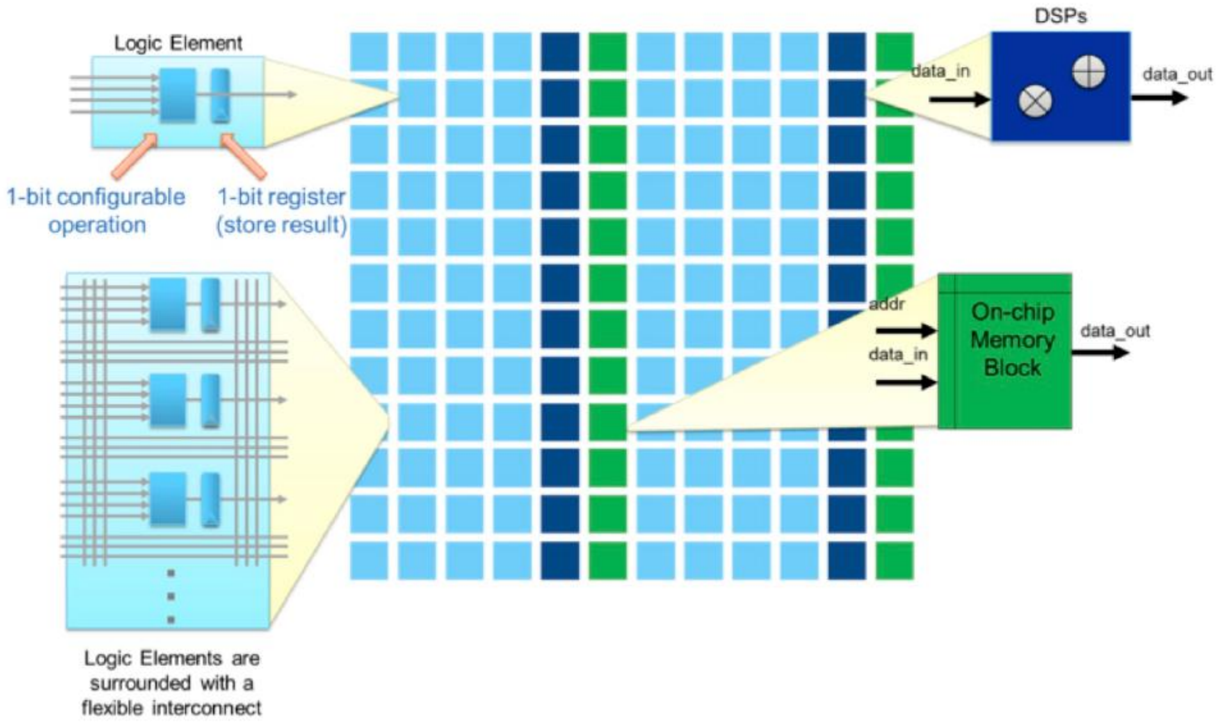


- 8GB + 8GB HBM
- 600 GB/s Mem BW
- 32b float
- MXU fp32 accumulation
- Reduced precision mult.

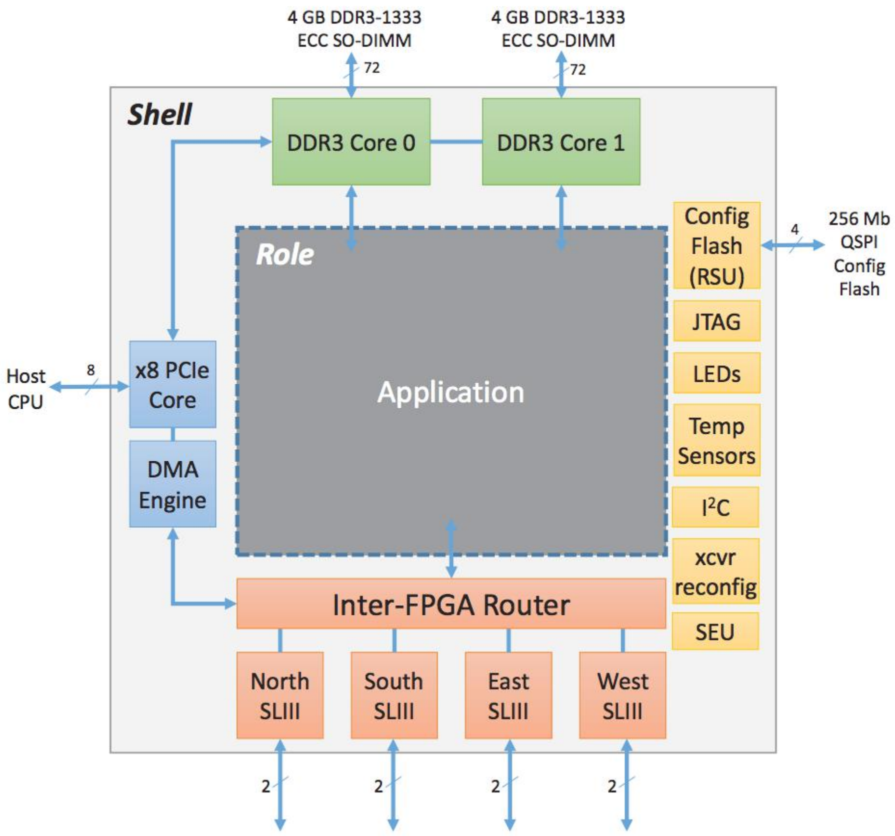
[Dean NIPS'17]

FPGA Based Acceleration

- RTL/HLS/OpenCL-based flow gaining acceptance and high-speed development
- Uses DSP-based blocks commonly used in FPGAs
- Being fully programmable means users can recompile new binaries to address new architecture challenges
- If using very low precision quantization, they can efficiently use resources and have very good performance



FPGA Based Acceleration



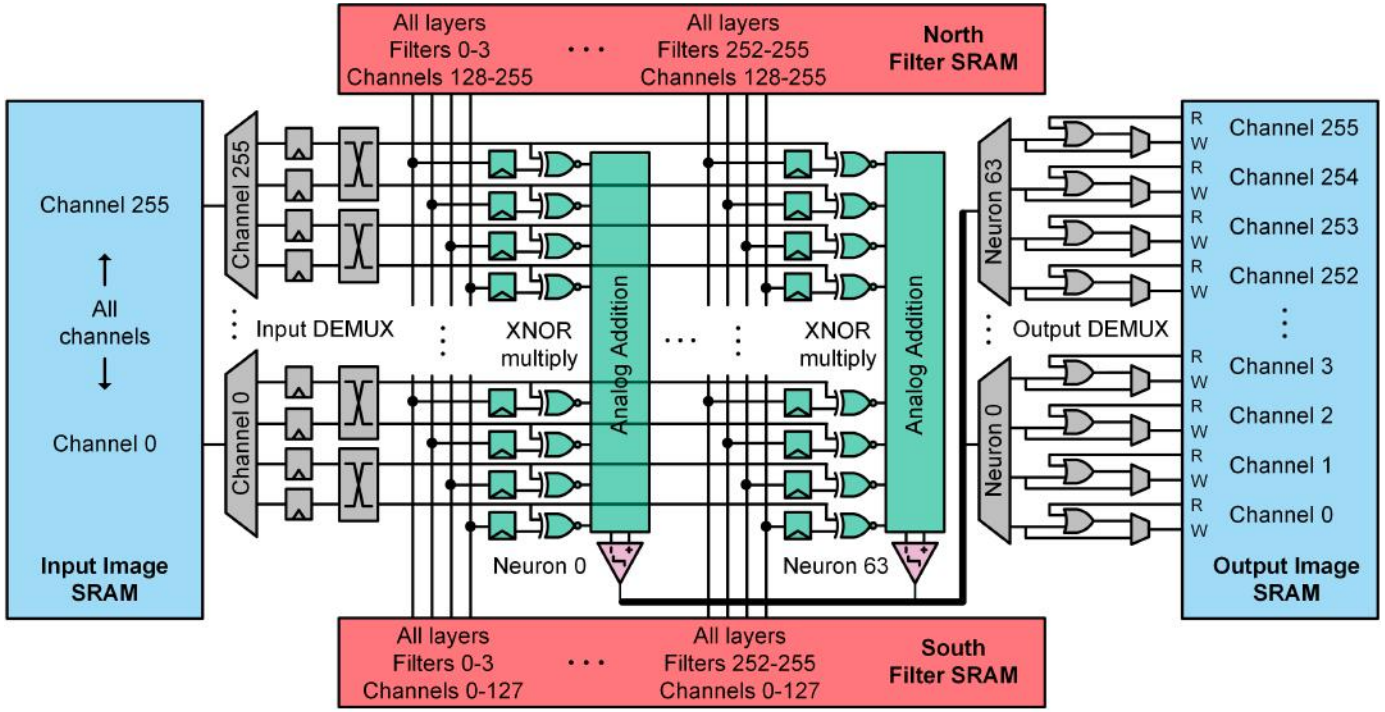
- Can adapt to multiple requirements, whether logic or computational

- Shell provides common infra-structure (including communication) to applications

- User has to worry only about application implementation

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Catapult_ISCA_2014.pdf

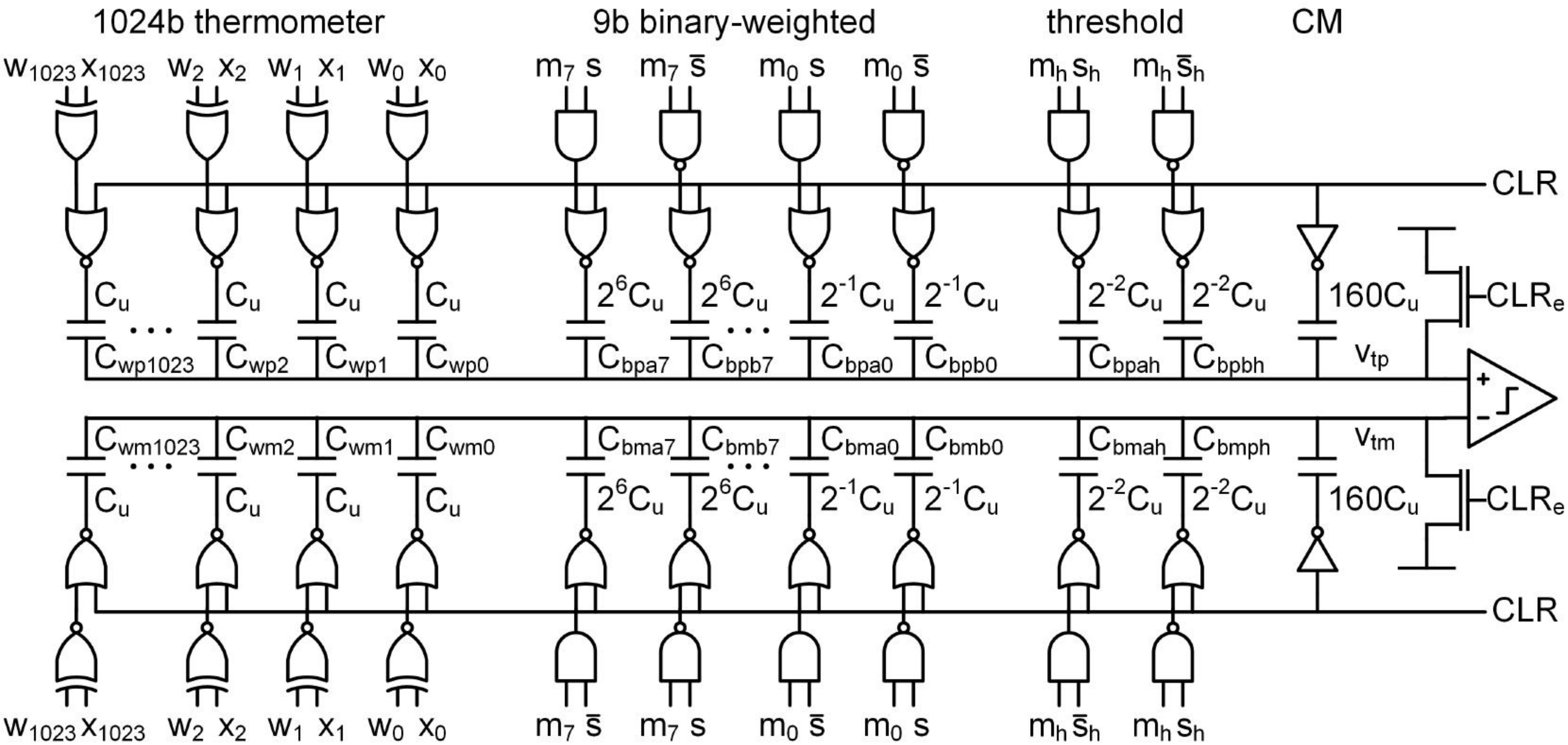
Mixed Signal Design



- +1/-1 XNOR network
- Use binary to thermometer to normalize the first layer
- Store and retrieve intermediate values in SRAM
- Perform multiplication using XNOR and analog addition

https://murmann-group.stanford.edu/mediawiki/index.php/Danny_Bankman

Mixed Signal Design



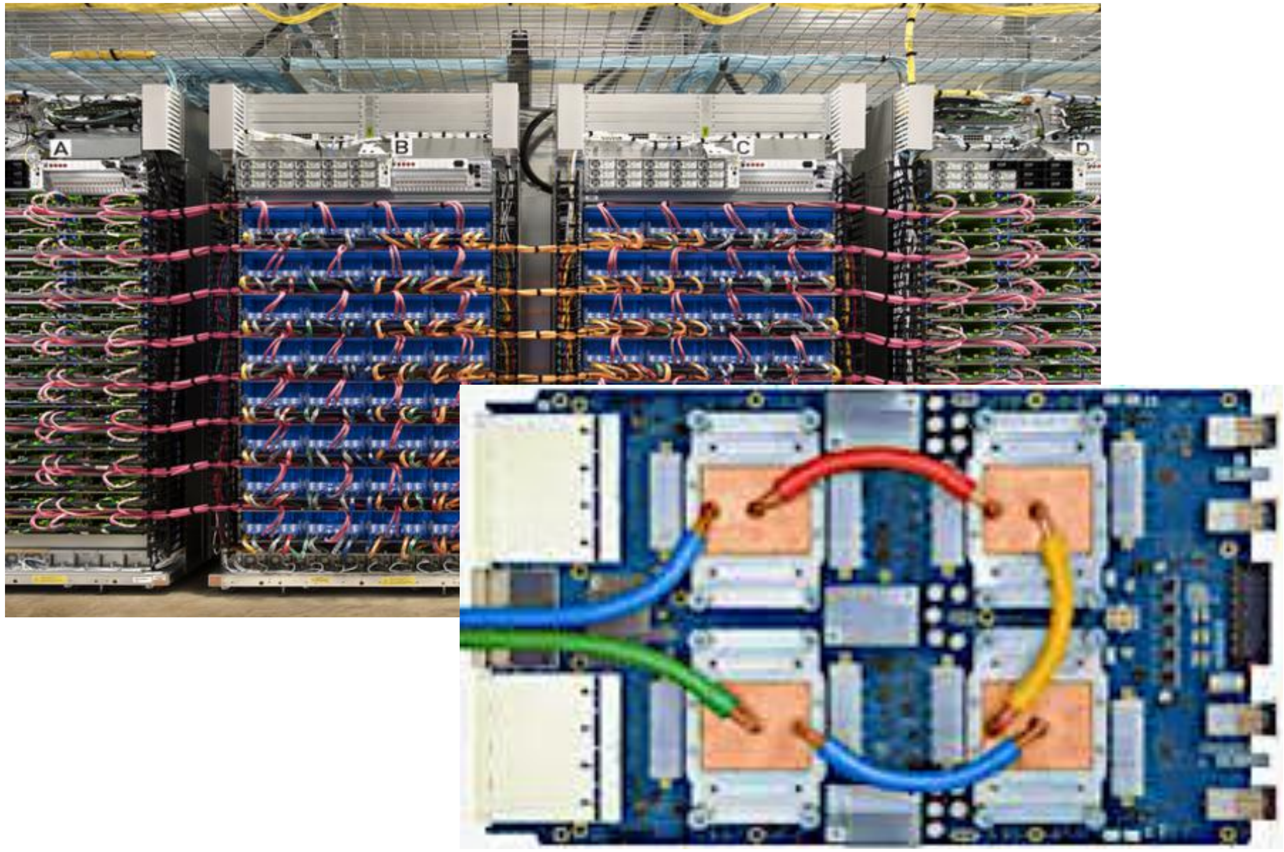
Summing Up

AI Application-System Co-design

John Hennessy [Turing Award, 2017]:

"A new approach to computer architecture is needed. We need Renaissance computer architecture. Instead of people who understand a sliver of the vertical stack, we need [..] people who understand applications, compilers, architecture [..]"

A Gap in Capabilities



vs

- 1-10 Exaflops
- 4PB memory
- 20 Watts

Compute-Energy Gap of 10^6

Thank You!