Efficient Processing of Deep Neural Networks: from Algorithms to Hardware Architectures

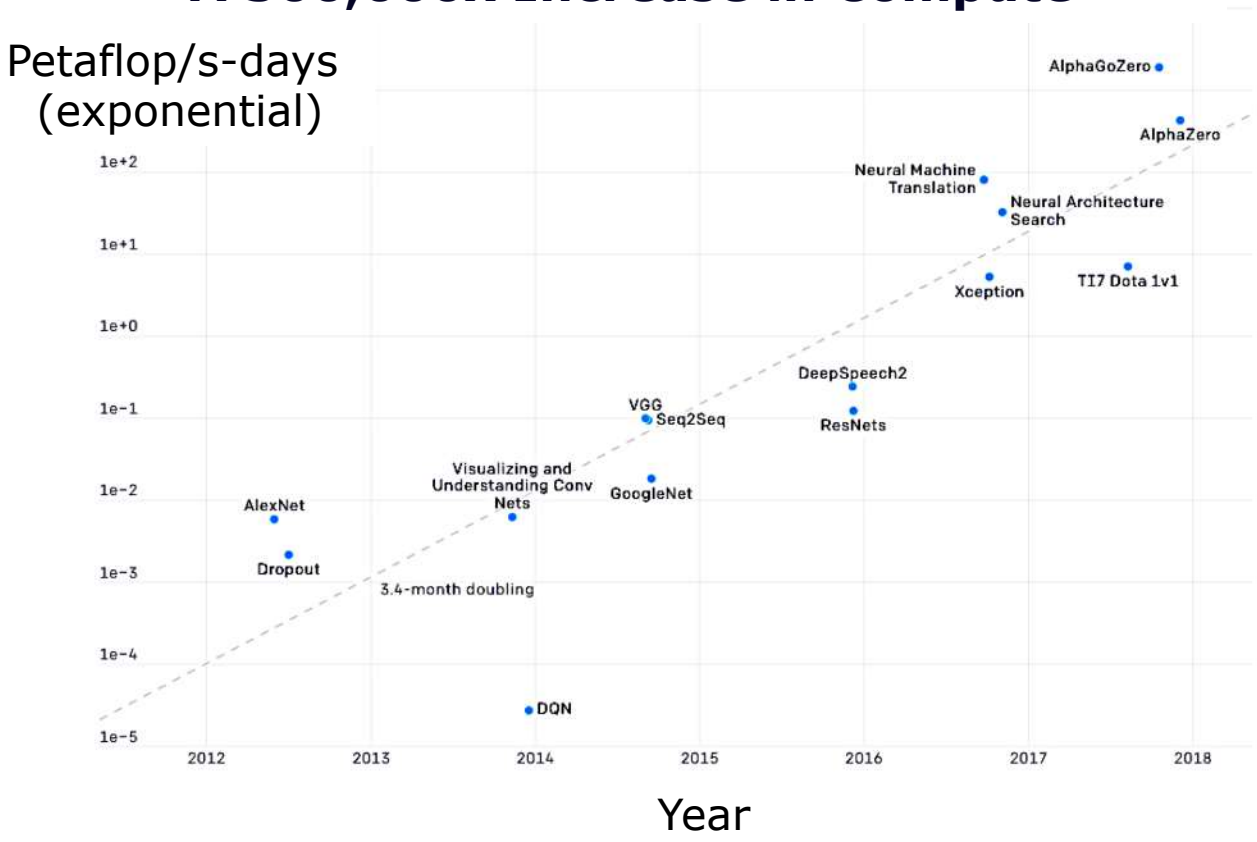
Vivienne Sze
Massachusetts Institute of Technology

In collaboration with
Tien-Ju Yang, Yu-Hsin Chen, Joel Emer

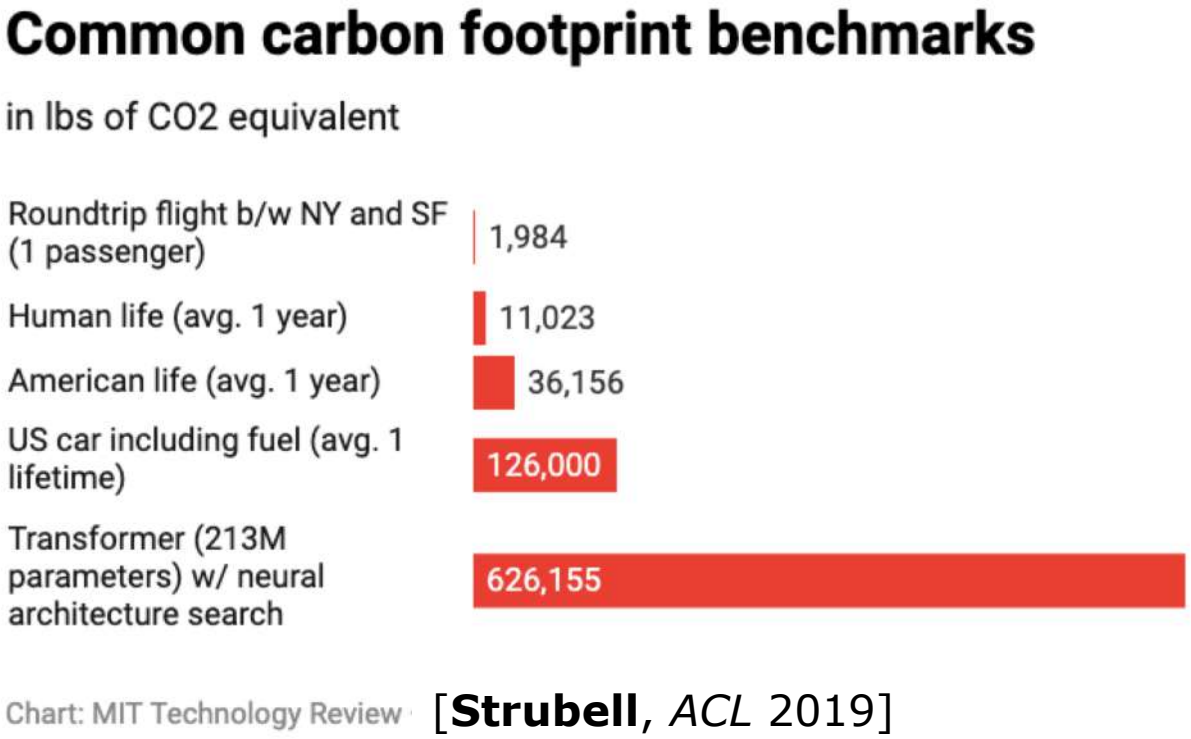
Slides available at
https://tinyurl.com/SzeNeurIPS2019

Compute Demands for Deep Neural Networks

AlexNet to AlphaGo Zero:
A 300,000x Increase in Compute



Source: Open AI (https://openai.com/blog/ai-and-compute/)



[**Strubell**, *ACL* 2019]

Processing at the “Edge” instead of the “Cloud”

Communication

Privacy

Latency

Deep Neural Networks for Self-Driving Cars

JACK STEWART TRANSPORTATION 02.06.18 08:00 AM

SELF-DRIVING CARS USE CRAZY AMOUNTS OF POWER, AND IT'S BECOMING A PROBLEM

Shelley, a self-driving Audi TT developed by Stanford University, uses the brains in the trunk to speed around a racetrack autonomously.

NIKKI KAHN/THE WASHINGTON POST/GETTY IMAGES

WIRED

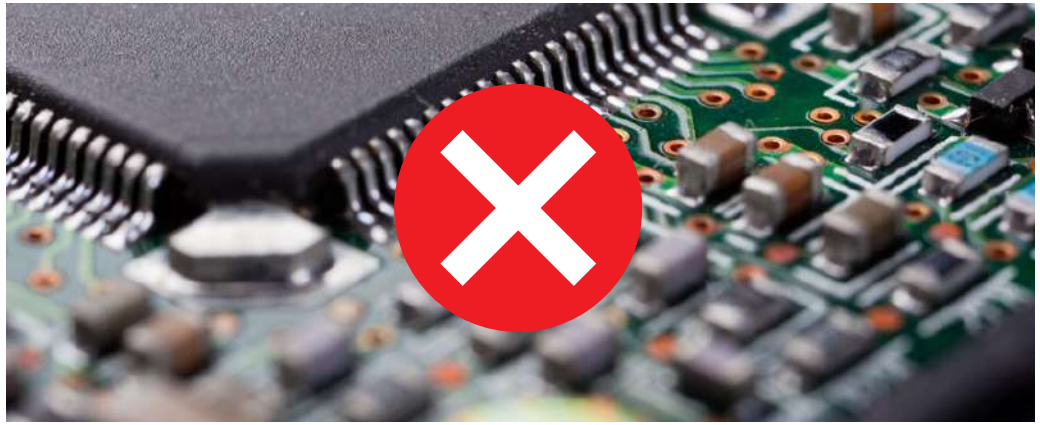
(Feb 2018)

Cameras and radar generate ~6 gigabytes of data every 30 seconds.

Prototypes use around 2,500 Watts. Generates wasted heat and some prototypes need water-cooling!

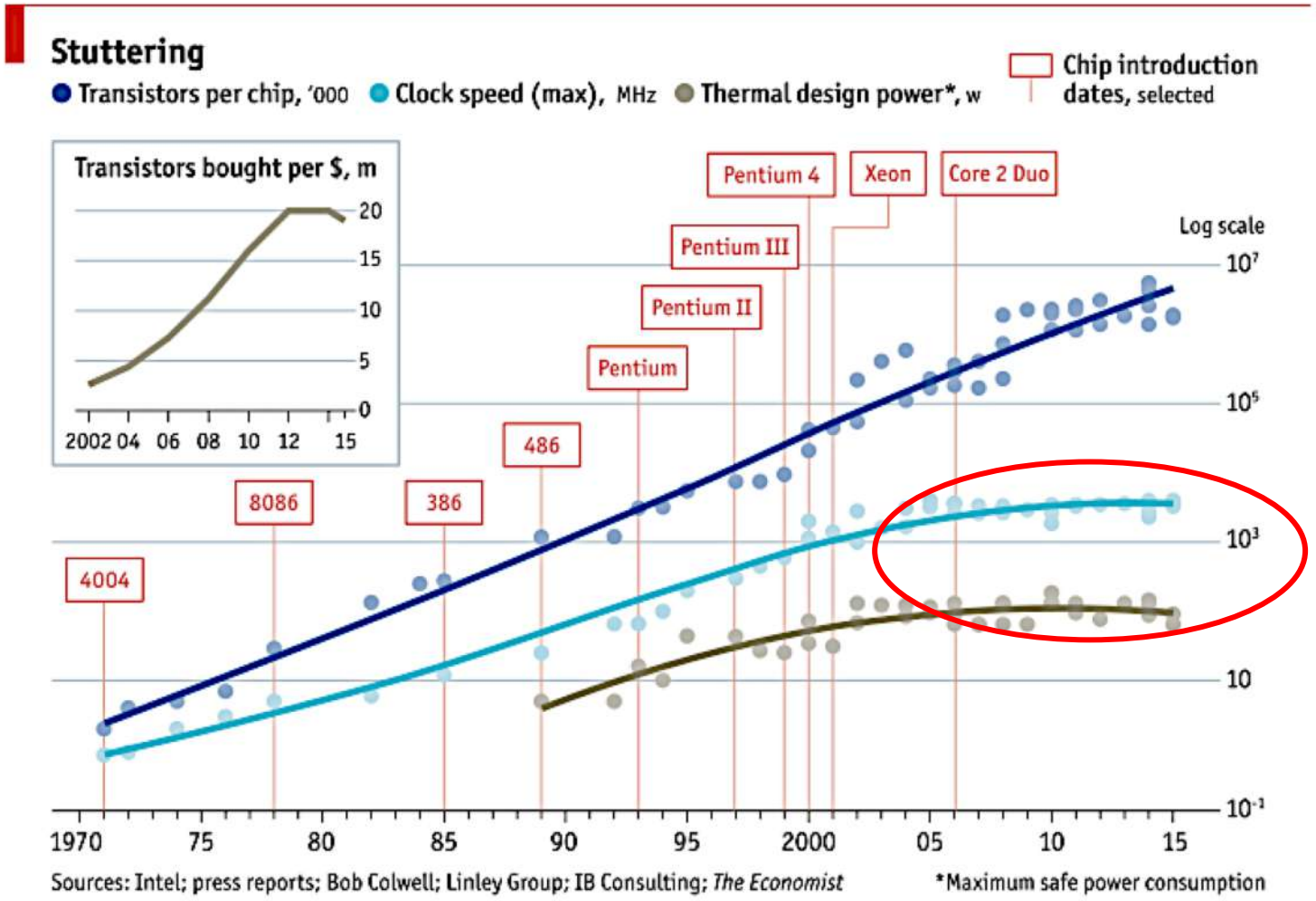
Existing Processors Consume Too Much Power

< 1 Watt



> 10 Watts

Transistors Are Not Getting More Efficient



Slowdown of Moore's Law and Dennard Scaling

General purpose microprocessors not getting faster or more efficient

Slowdown

Need **specialized / domain-specific hardware** for significant improvements in speed and energy efficiency

Goals of this Tutorial

- Many approaches for efficient processing of DNNs. **Too many to cover!**

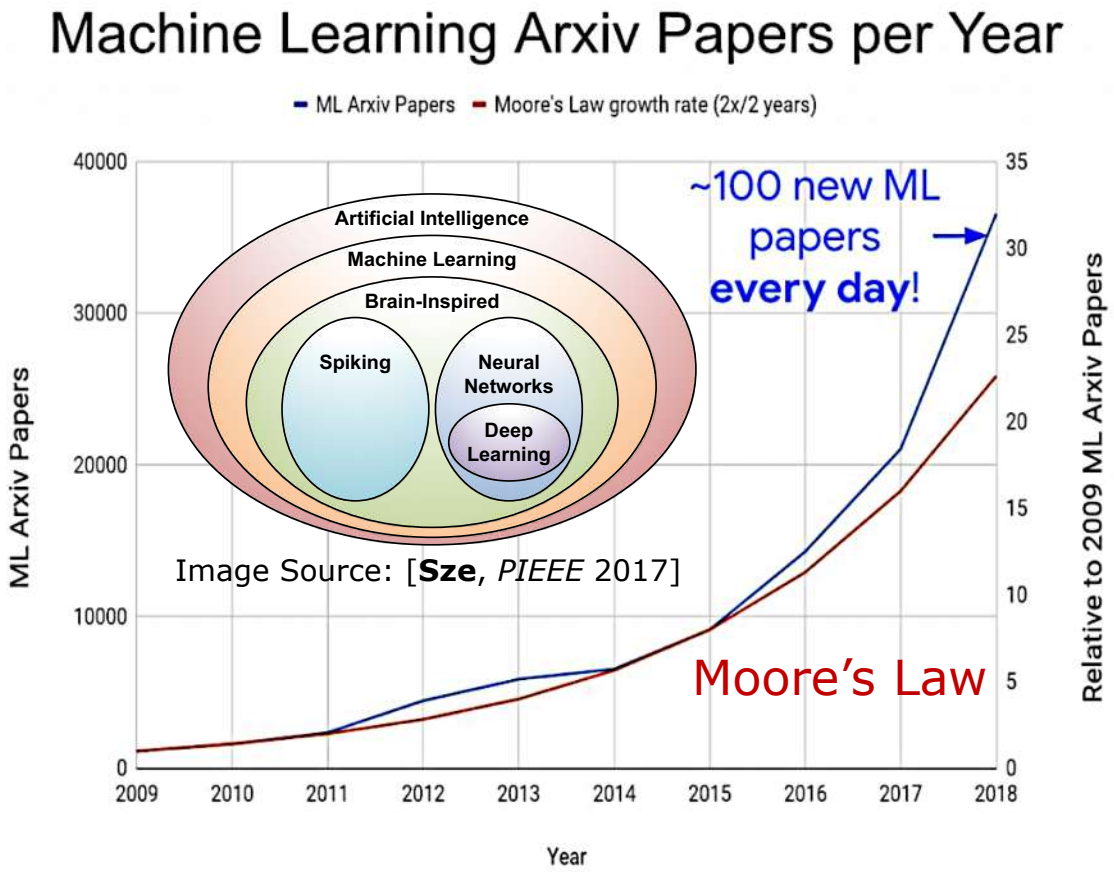


Image Source: [**Sze**, *PIEEE* 2017]

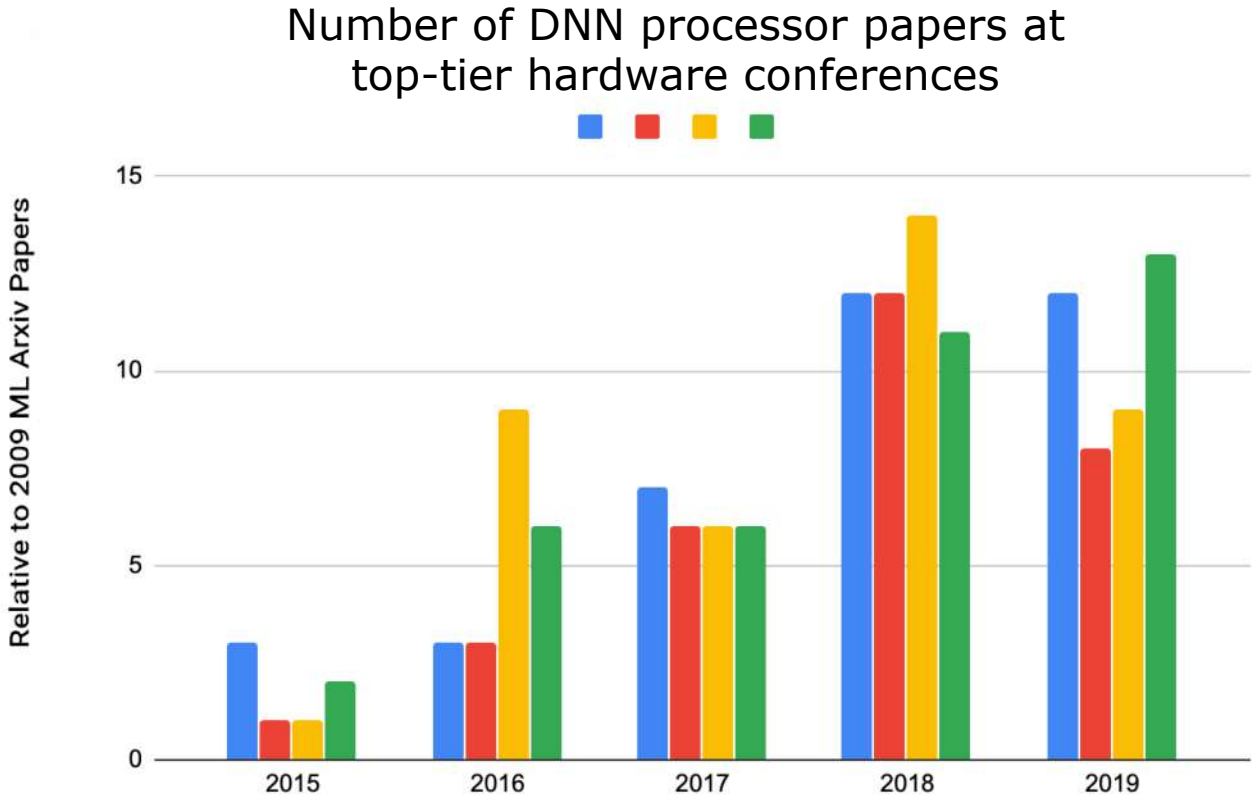


Image Source: [**Dean**, *ISSCC* 2020]

Goals of this Tutorial

- Many approaches for efficient processing of DNNs. **Too many to cover!**

Big Bets On A.I. Open a New Frontier for Chips Start-Ups, Too. *(January 14, 2018)*

"Today, **at least 45 start-ups are working on chips** that can power tasks like speech and self-driving cars, and at least five of them have raised more than $100 million from investors. **Venture capitalists invested more than $1.5 billion in chip start-ups last year**, nearly doubling the investments made two years ago, according to the research firm CB Insights."

Goals of this Tutorial

- Many approaches for efficient processing of DNNs. **Too many to cover!**

- We will focus on how to **evaluate** approaches for efficient processing of DNNs
  - Approaches include the design of DNN hardware processors and DNN models
  - What are the key questions to ask?

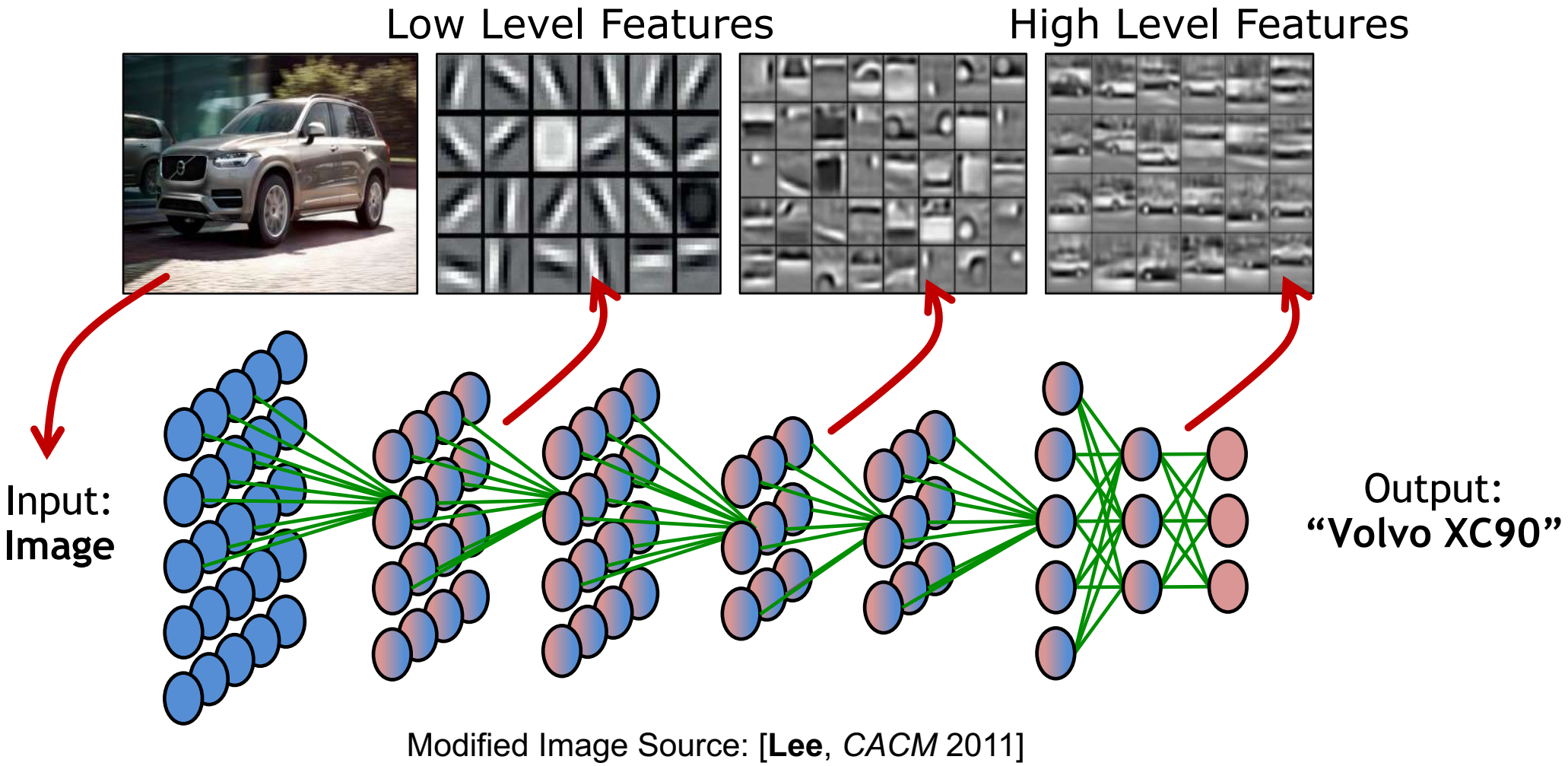
- Specifically, we will discuss
  - What are the **key metrics** that should be measured and compared?
  - What are the **challenges** towards achieving these metrics?
  - What are the **design considerations** and tradeoffs?

- We will focus on inference, but many concepts covered also apply to training

Tutorial Overview

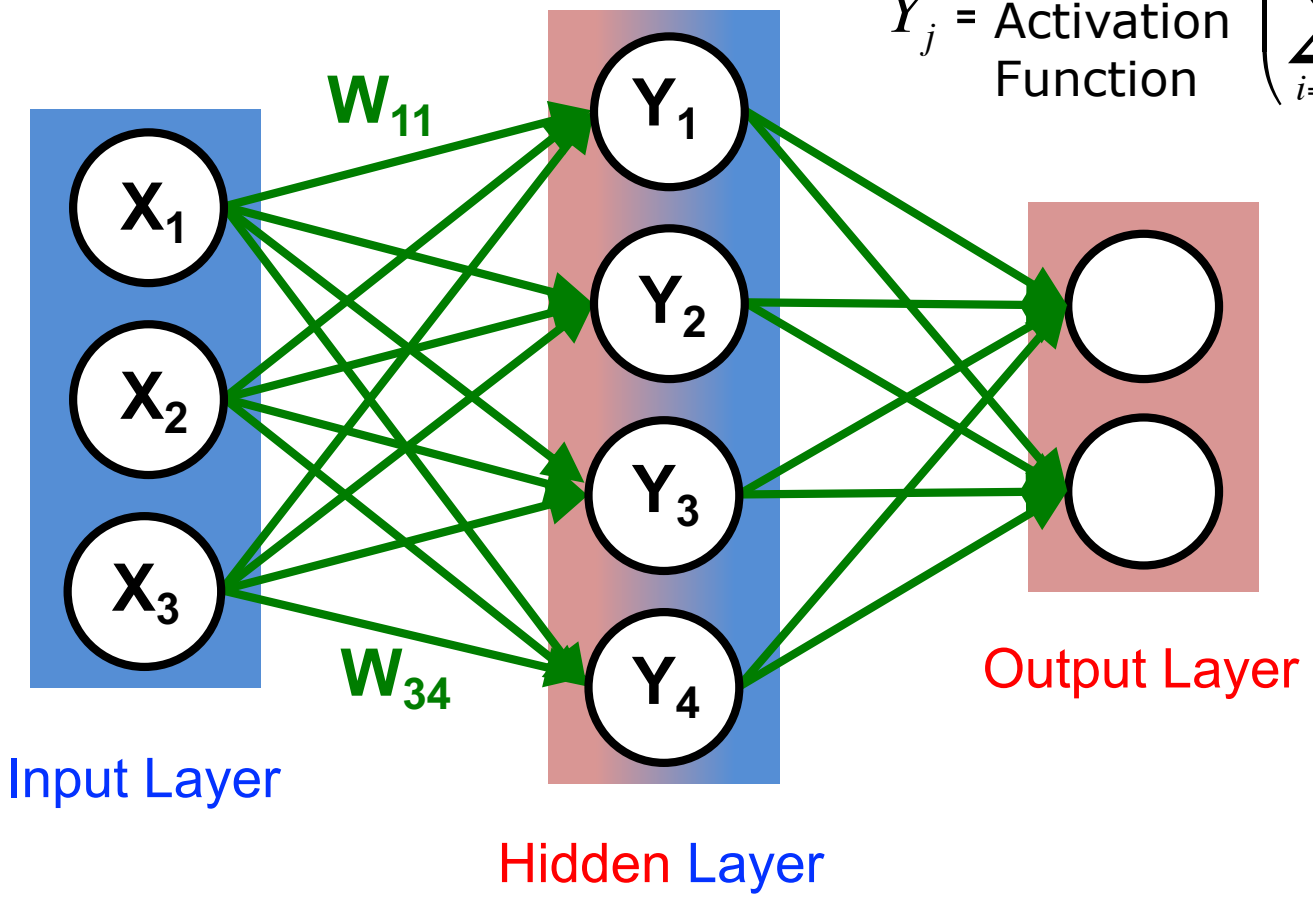
- Deep Neural Networks Overview (Terminology)
- Key Metrics and Design Objectives
- Design Considerations
  - CPU and GPU Platforms
  - Specialized / Domain-Specific Hardware (ASICs)
  - **Break Q&A**
  - Algorithm (DNN Model) and Hardware Co-Design
  - Other Platforms
- Tools for Systematic Evaluation of DNN Processors

What are Deep Neural Networks?



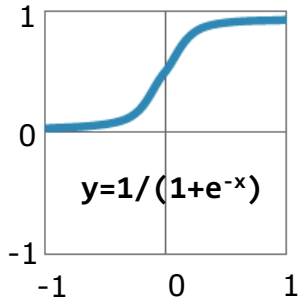
Modified Image Source: [**Lee**, *CACM* 2011]

Weighted Sums



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

Sigmoid



Rectified Linear Unit (ReLU)

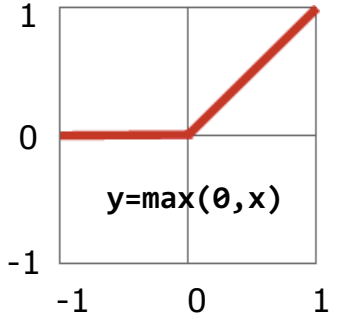
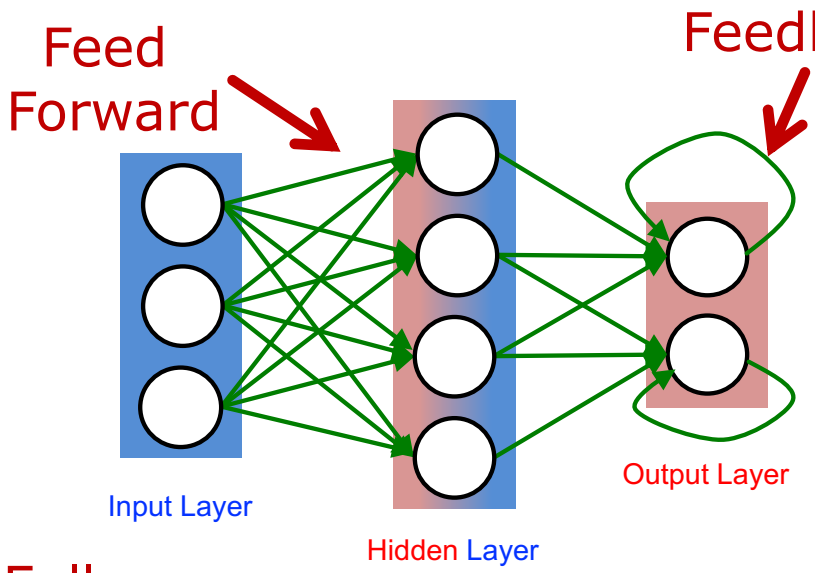


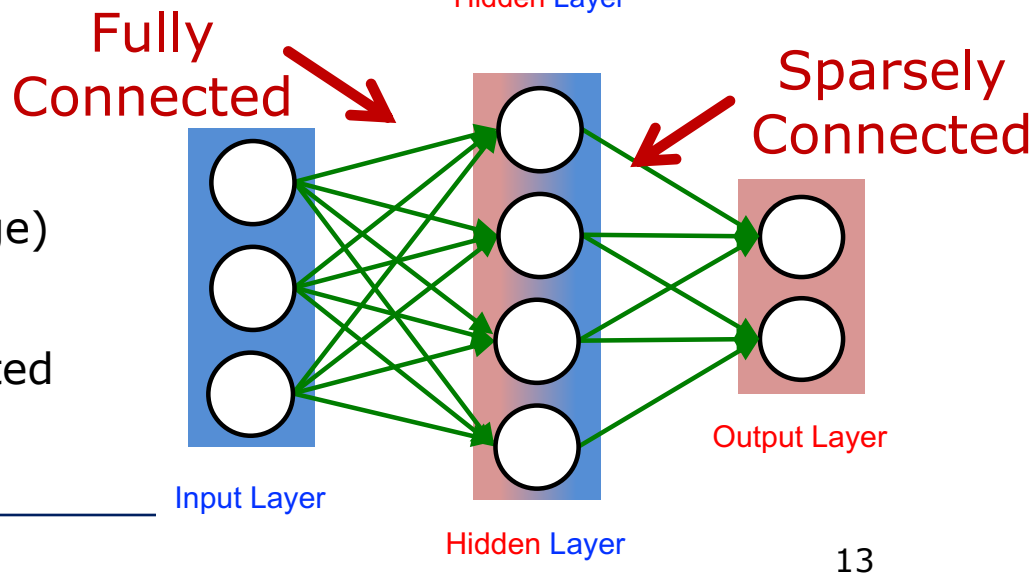
Image source: Caffe tutorial

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]

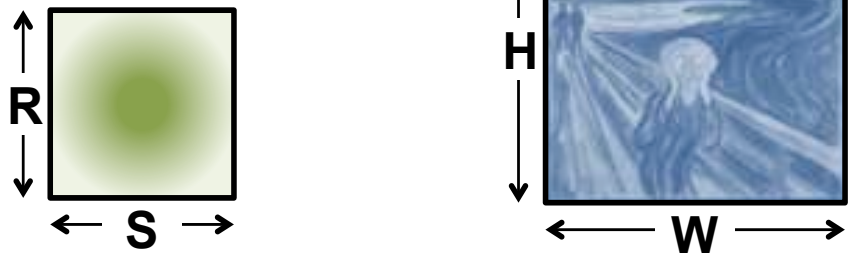




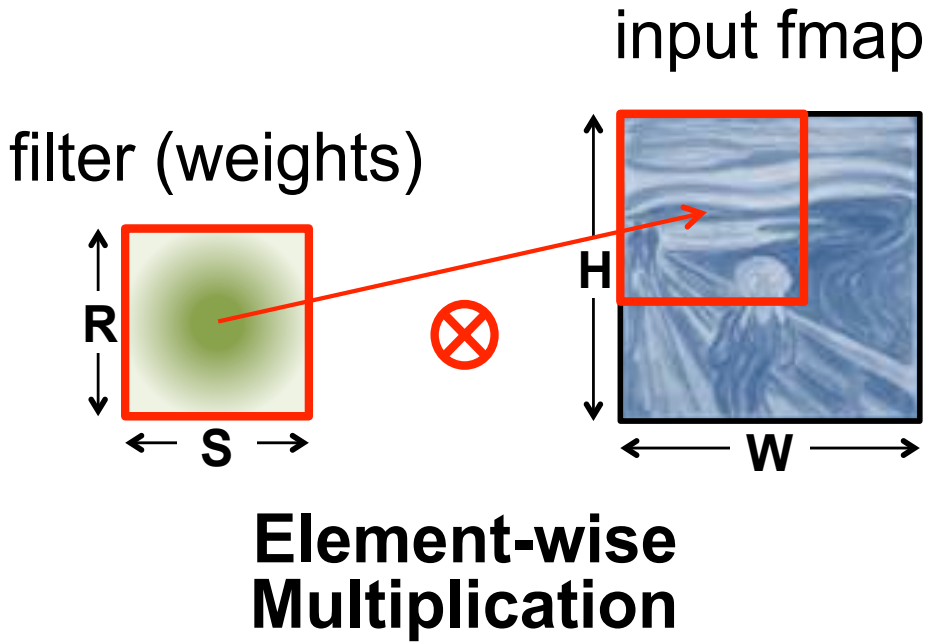
High-Dimensional Convolution in CNN

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

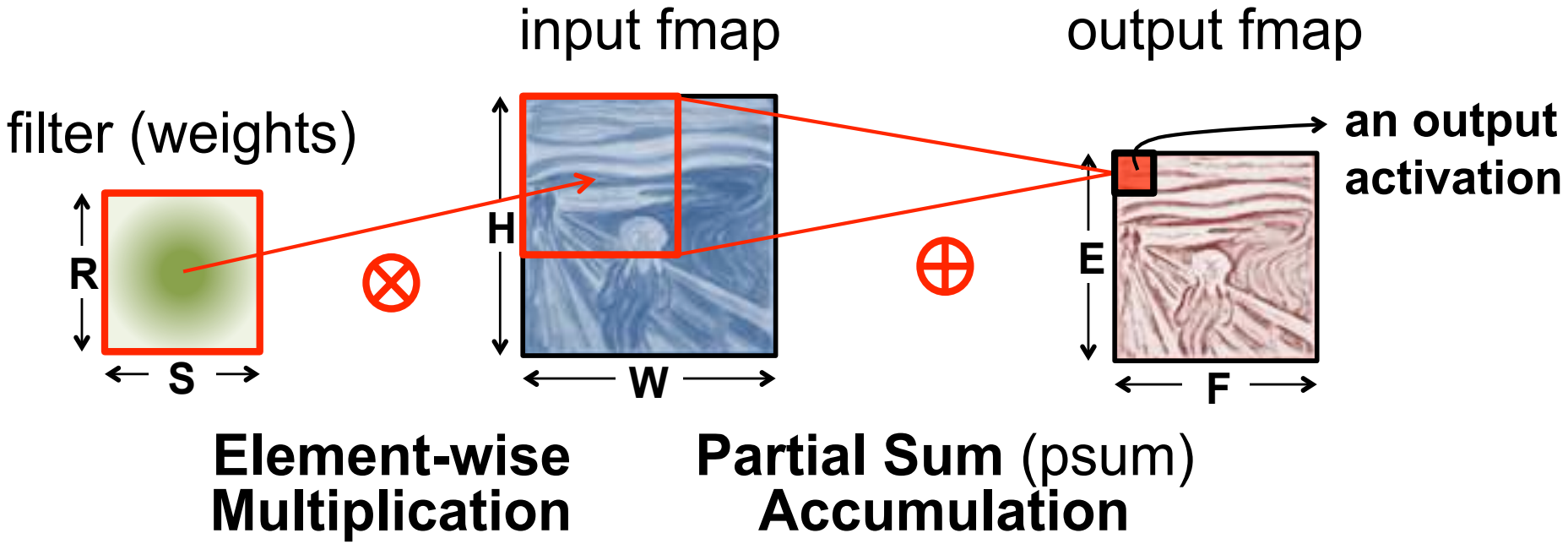
filter (weights)



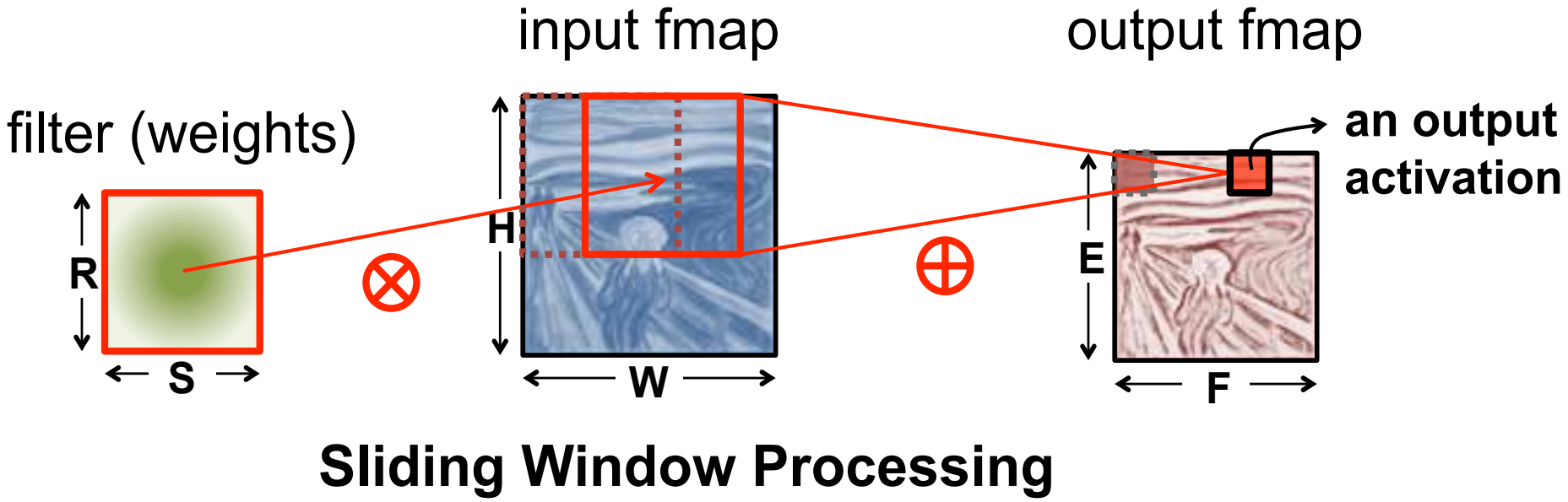
High-Dimensional Convolution in CNN



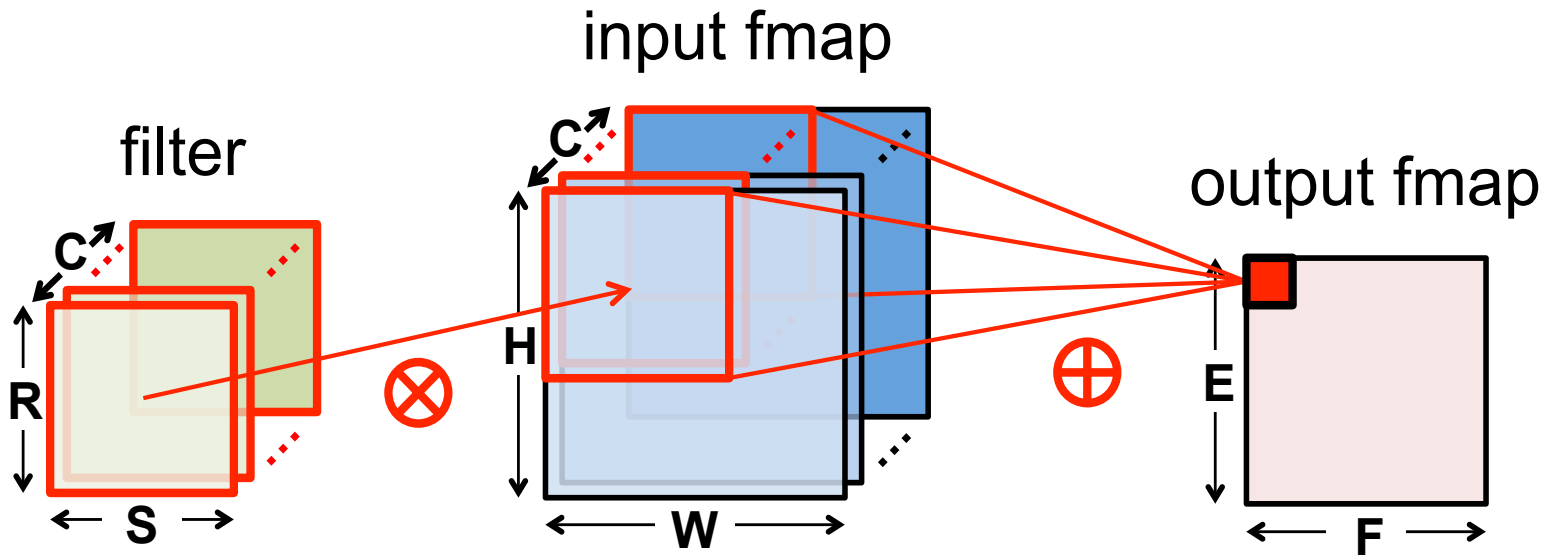
High-Dimensional Convolution in CNN



High-Dimensional Convolution in CNN



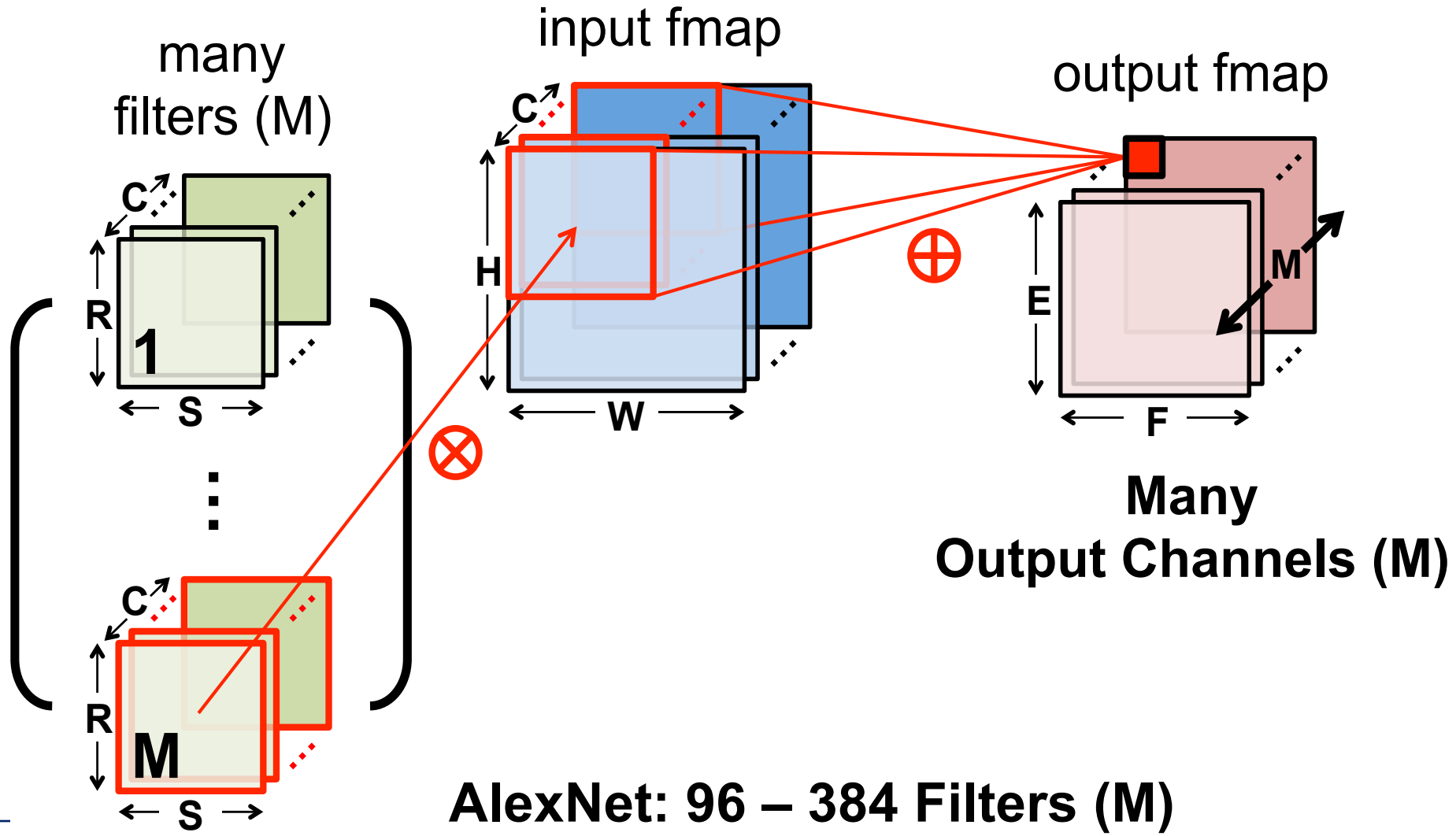
High-Dimensional Convolution in CNN



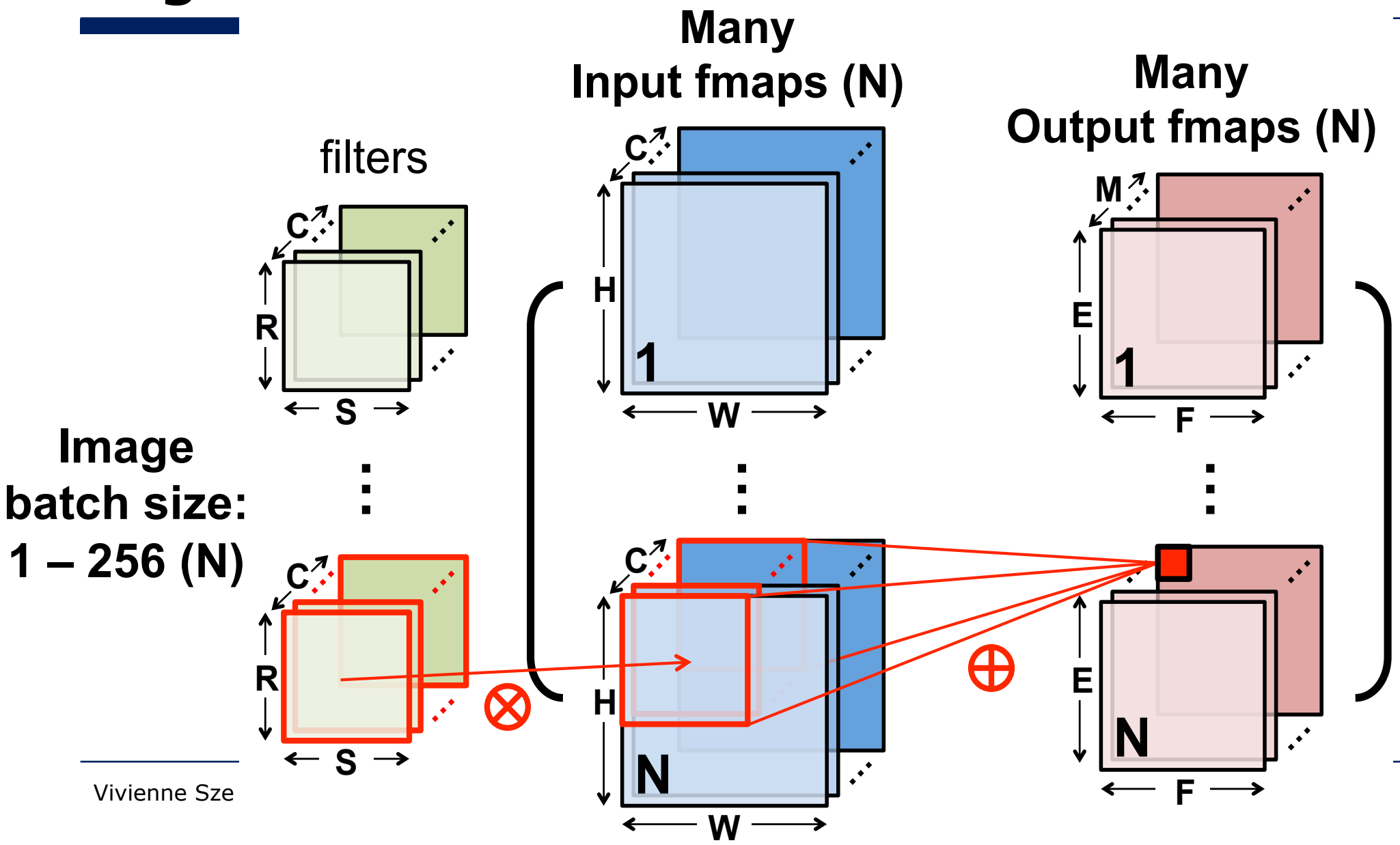
Many Input Channels (C)

AlexNet: 3 – 192 Channels (C)

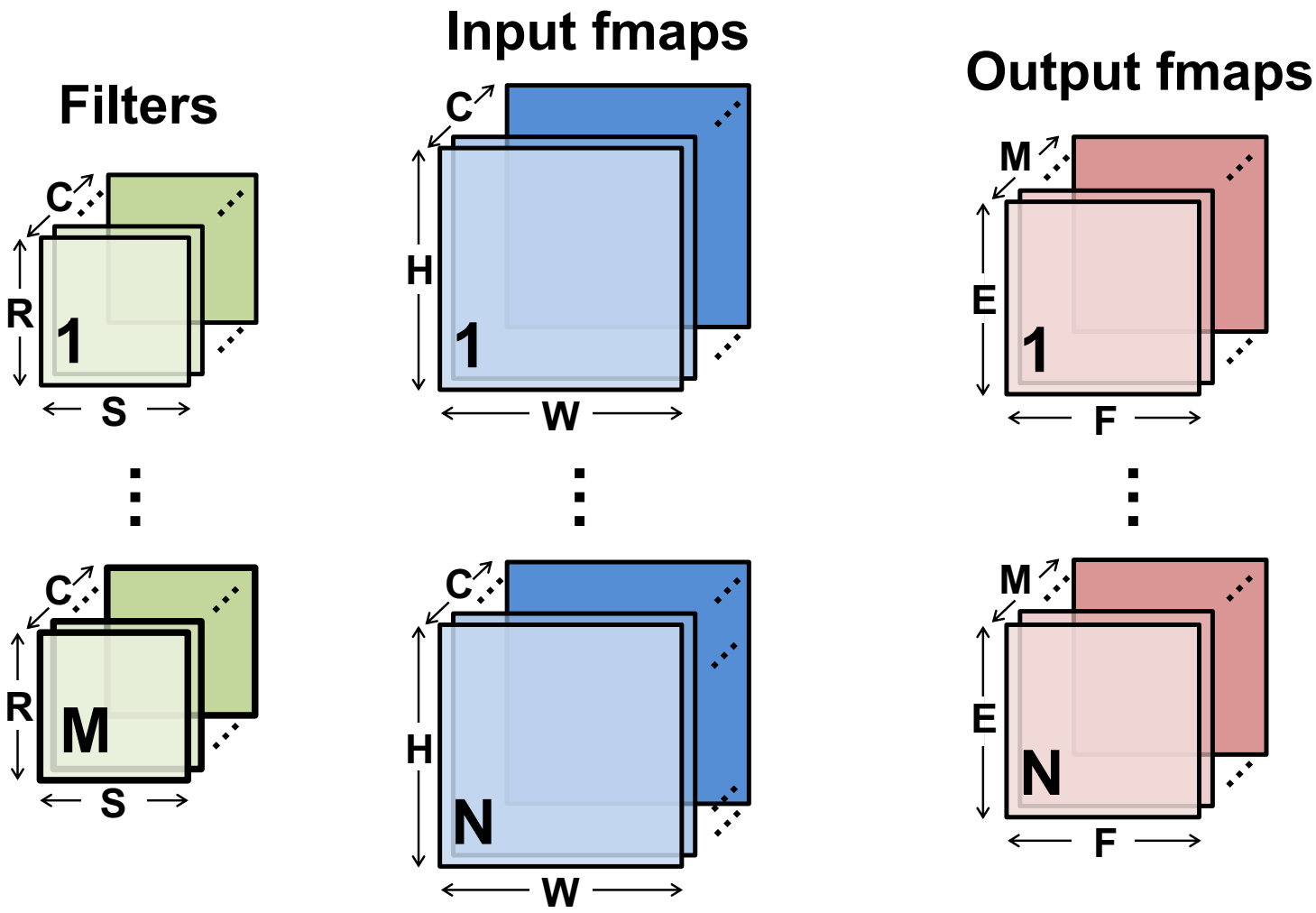
High-Dimensional Convolution in CNN



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)

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**

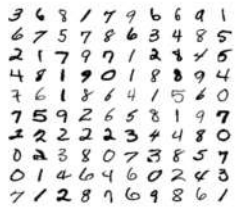
* Does not include multi-crop and ensemble
** Increase in FC layers due to squeeze-and-excitation layers (much smaller than FC layers for classification)

Key Metrics and Design Objectives

Key Metrics: Much more than OPS/W!

- **Accuracy**
  - Quality of result
- **Throughput**
  - Analytics on high volume data
  - Real-time performance (e.g., video at 30 fps)
- **Latency**
  - For interactive applications (e.g., autonomous navigation)
- **Energy and Power**
  - Embedded devices have limited battery capacity
  - Data centers have a power ceiling due to cooling cost
- **Hardware Cost**
  - $$$
- **Flexibility**
  - Range of DNN models and tasks
- **Scalability**
  - Scaling of performance with amount of resources

MNIST



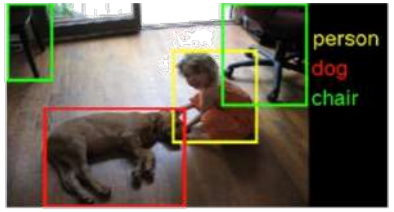
CIFAR-10

ImageNet

Embedded Device

Data Center

Computer Vision



Speech Recognition

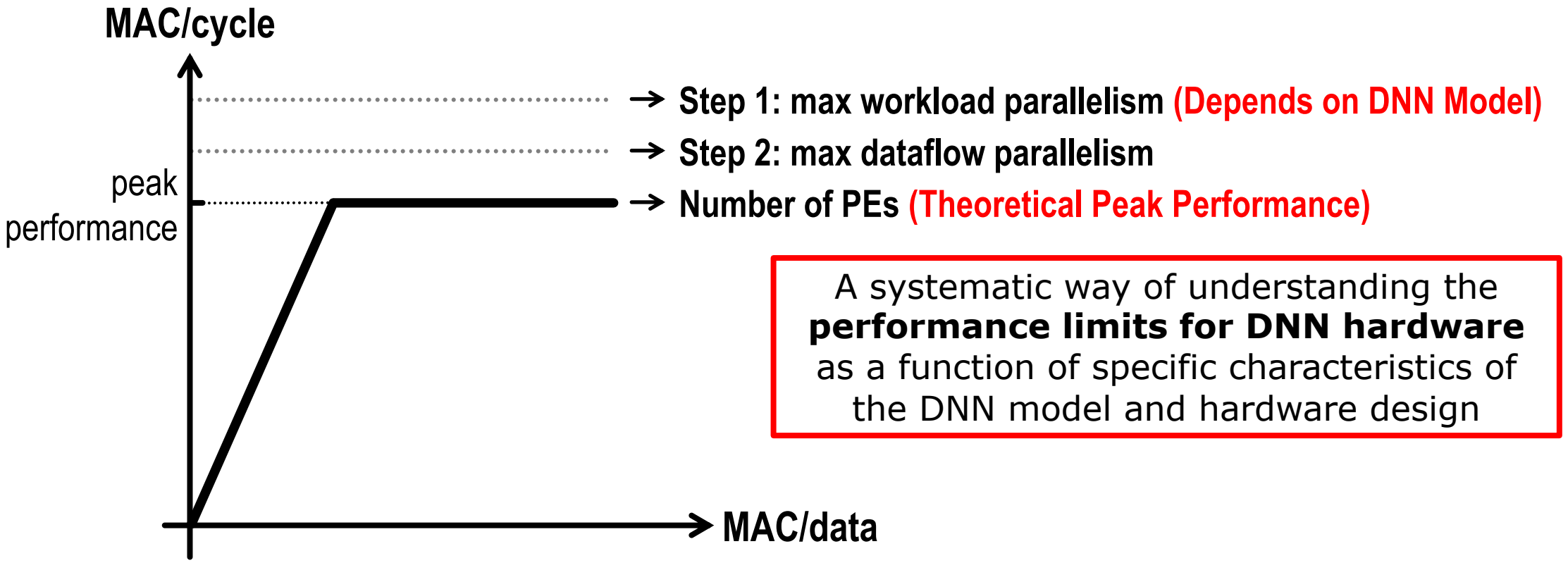
[**Sze**, *CICC* 2017]

Key Design Objectives of DNN Processor

- **Increase Throughput and Reduce Latency**
  - Reduce time per MAC
    - Reduce critical path → increase clock frequency
    - Reduce instruction overhead
  - Avoid unnecessary MACs (save cycles)
  - Increase number of processing elements (PE) → more MACs in parallel
    - Increase area density of PE or area cost of system
  - Increase PE utilization* → keep PEs busy
    - Distribute workload to as many PEs as possible
    - Balance the workload across PEs
    - Sufficient memory bandwidth to deliver workload to PEs (reduce idle cycles)
- Low latency has an additional constraint of **small batch size**

*(100% = peak performance)

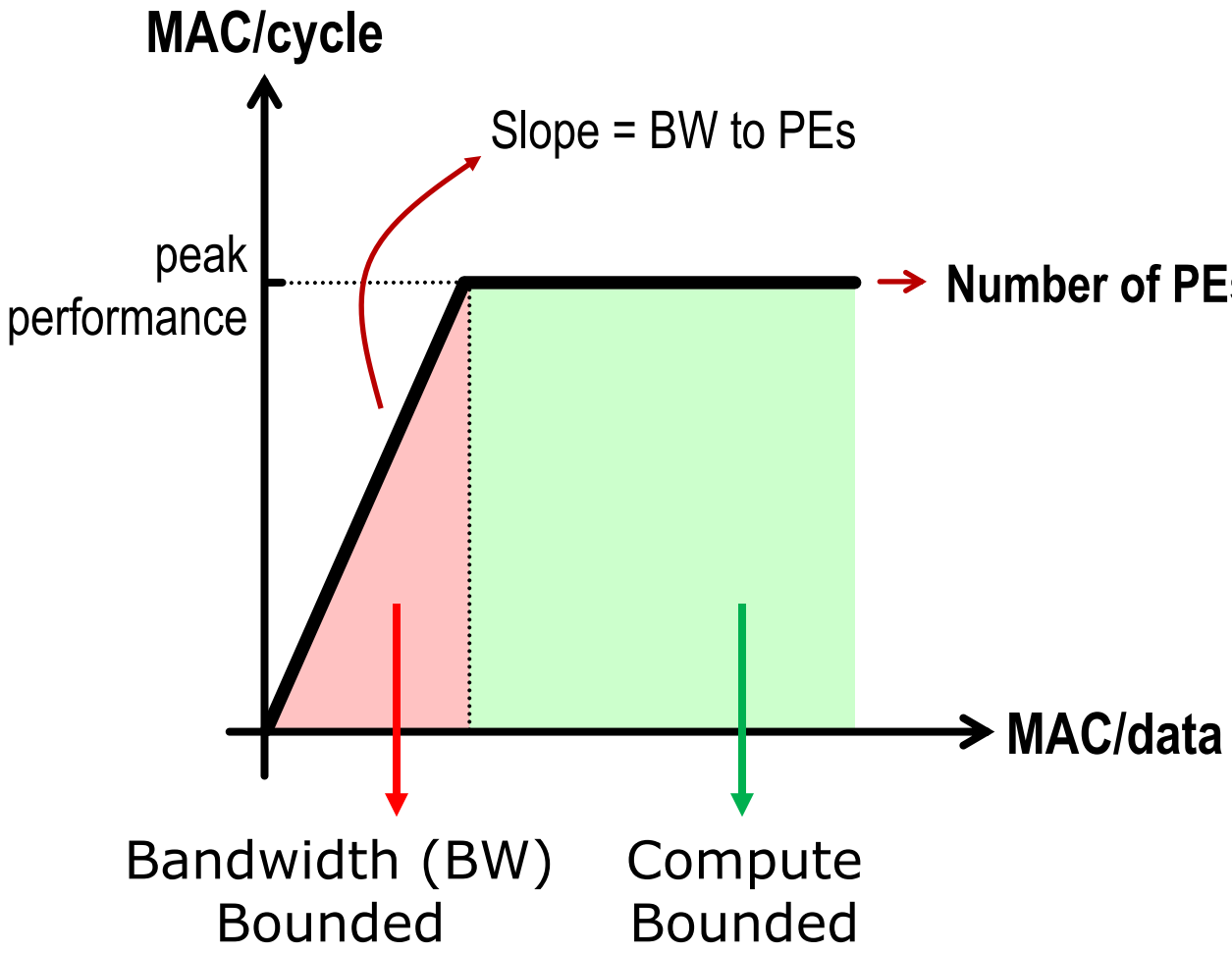
Eyexam: Performance Evaluation Framework



A systematic way of understanding the **performance limits for DNN hardware** as a function of specific characteristics of the DNN model and hardware design

[**Chen**, *arXiv* 2019: https://arxiv.org/abs/1807.07928]

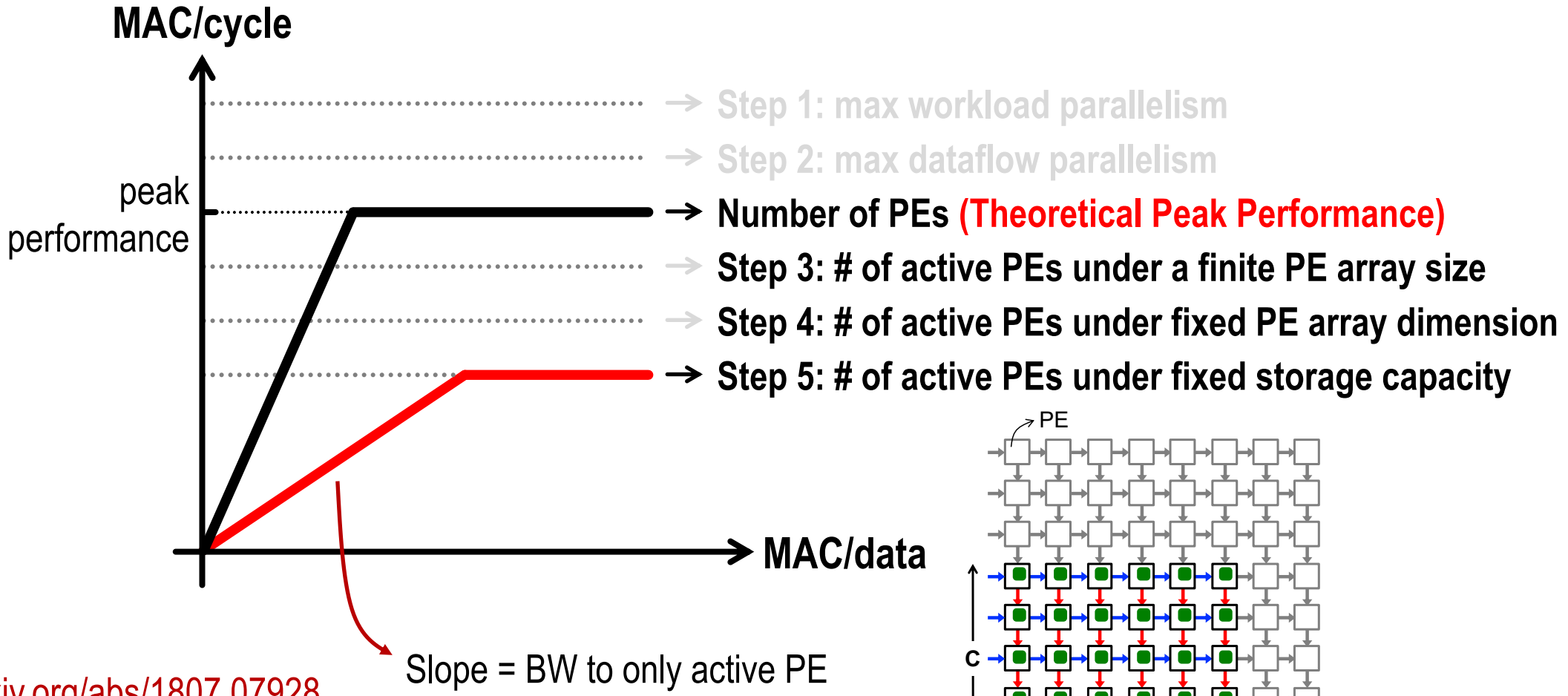
Eyexam: Performance Evaluation Framework



Based on Roofline Model

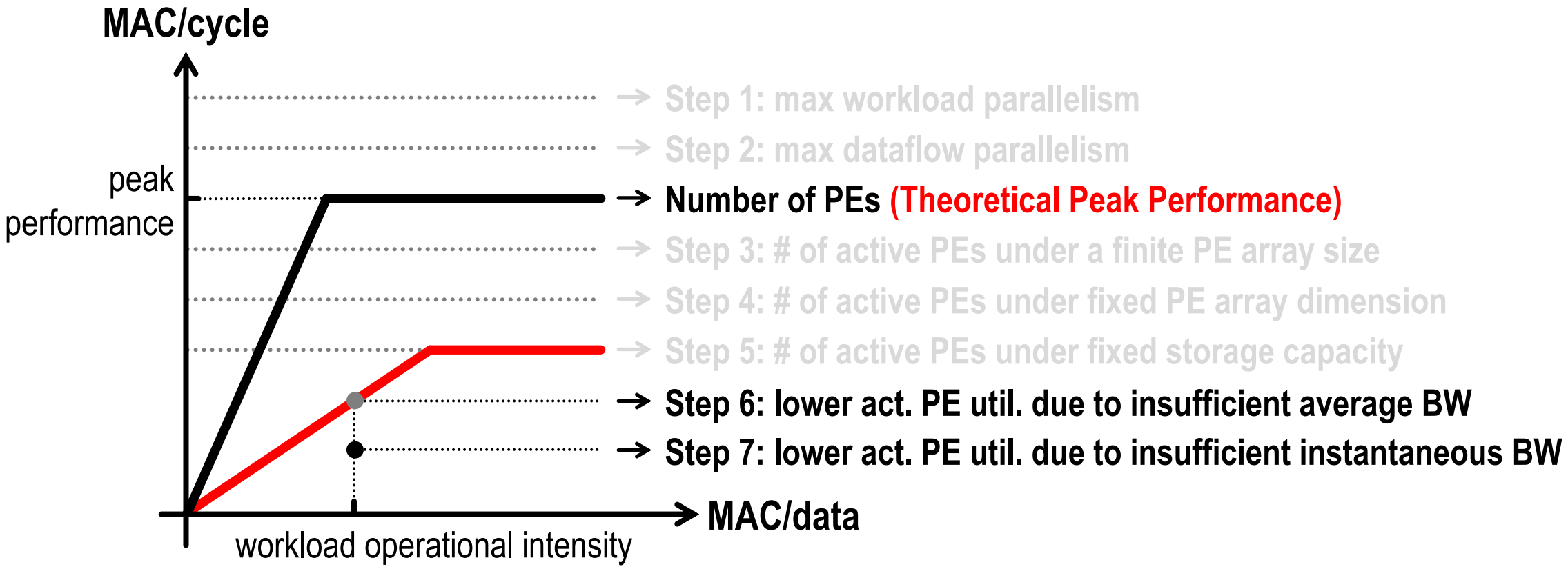
[**Williams**, *CACM* 2009]

Eyexam: Performance Evaluation Framework



https://arxiv.org/abs/1807.07928

Eyexam: Performance Evaluation Framework



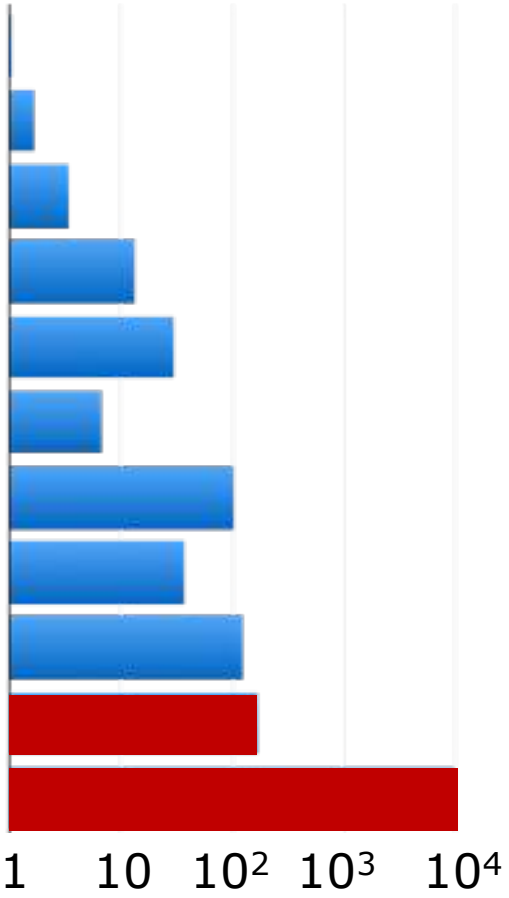
https://arxiv.org/abs/1807.07928

Key Design Objectives of DNN Processor

- **Reduce Energy and Power Consumption**
  - Reduce data movement as it dominates energy consumption
    - Exploit data reuse
  - Reduce energy per MAC
    - Reduce switching activity and/or capacitance
    - Reduce instruction overhead
  - Avoid unnecessary MACs
- Power consumption is limited by heat dissipation, which limits the **maximum # of MACs in parallel** (i.e., throughput)

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

[**Horowitz**, *ISSCC* 2014]

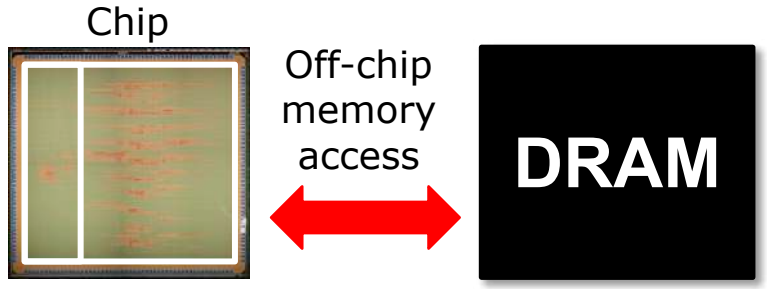


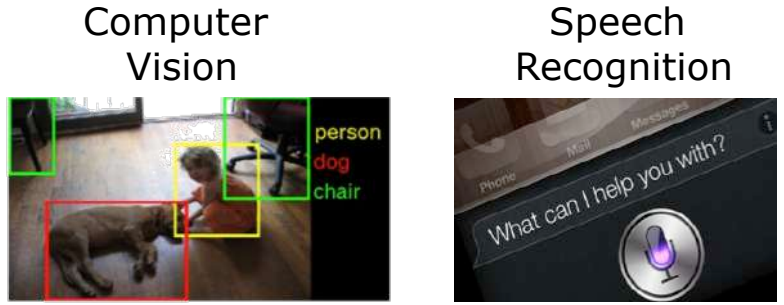
Key Design Objectives of DNN Processor

- **Flexibility**
  - Reduce overhead of supporting flexibility
  - Maintain efficiency across wide range of DNN models
    - Different layer shapes impact the amount of
      - Required storage and compute
      - Available data reuse that can be exploited
    - Different precision across layers & data types (weight, activation, partial sum)
    - Different degrees of sparsity (number of zeros in weights or activations)
    - Types of DNN layers and computation beyond MACs (e.g., activation functions)
- **Scalability**
  - Increase how performance (i.e., throughput, latency, energy, power) scales with increase in amount of resources (e.g., number of PEs, amount of memory, etc.)

Specifications to Evaluate Metrics

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
  - Difficult tasks typically require more complex DNN models
- **Throughput**
  - Number of PEs with utilization (not just peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Off-chip memory access (e.g., DRAM)
- **Hardware Cost**
  - On-chip storage, # of PEs, chip area + process technology
- **Flexibility**
  - Report performance across a wide range of DNN models
  - Define range of DNN models that are efficiently supported





[**Sze**, *CICC* 2017]

Comprehensive Coverage for Evaluation

- All metrics should be reported for fair evaluation of design tradeoffs

- Examples of what can happen if a certain metric is omitted:
  - **Without the accuracy** given for a specific dataset and task, one could run a simple DNN and claim low power, high throughput, and low cost – however, the processor might not be usable for a meaningful task
  - **Without reporting the off-chip memory access**, one could build a processor with *only* MACs and claim low cost, high throughput, high accuracy, and low chip power – however, when evaluating system power, the off-chip memory access would be substantial

- Are results measured or simulated? On what test data?

Example Evaluation Process

The evaluation process for whether a DNN processor is a viable solution for a given application might go as follows:

1. **Accuracy** determines if it can perform the given task
2. **Latency and throughput** determine if it can run fast enough and in real-time
3. **Energy and power consumption** will primarily dictate the form factor of the device where the processing can operate
4. **Cost**, which is primarily dictated by the chip area, determines how much one would pay for this solution
5. **Flexibility** determines the range of tasks it can support

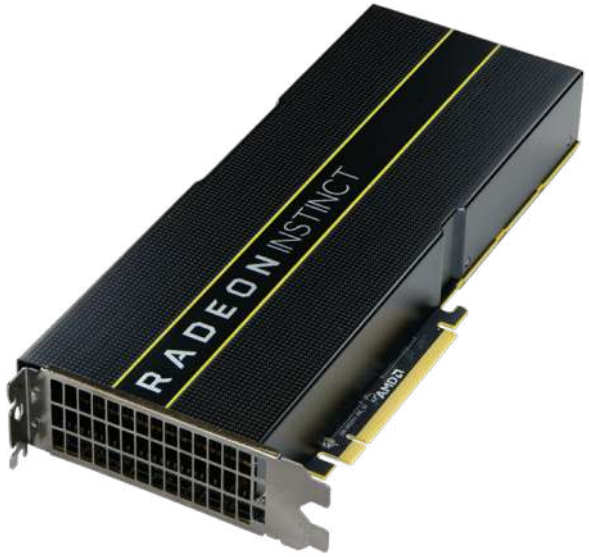
CPU & GPU Platforms

CPUs and GPUs Targeting DNNs

Intel Xeon (Cascade Lake)

Nvidia Tesla (Volta)

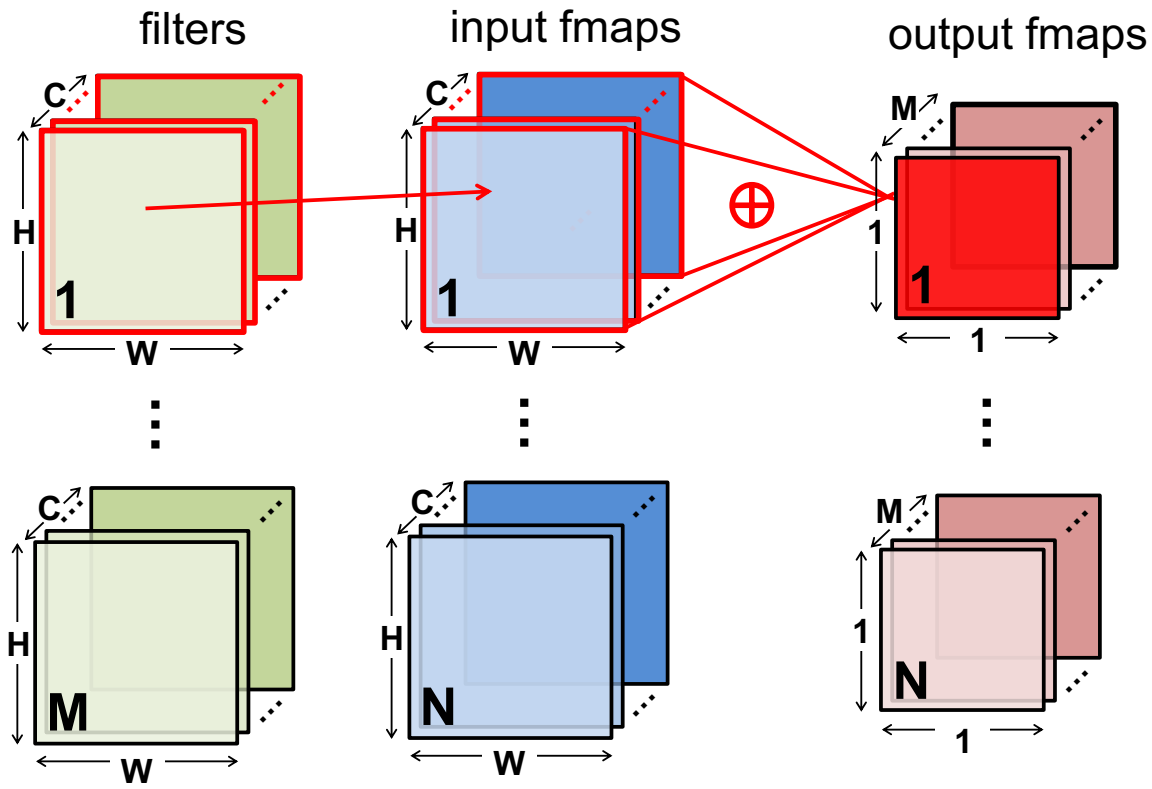
AMD Radeon (Instinct)

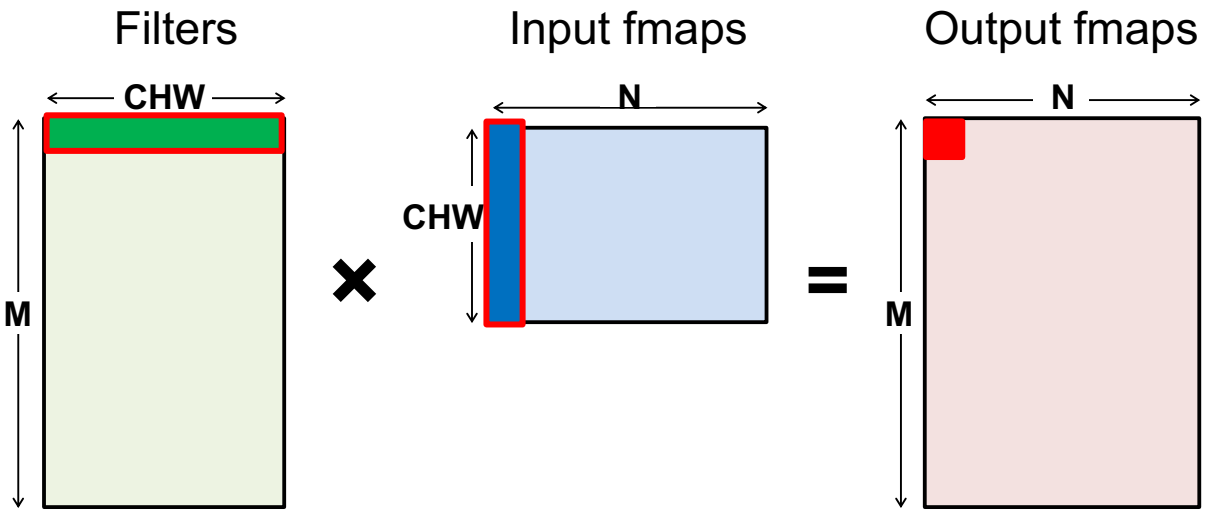


Use **matrix multiplication libraries** on CPUs and GPUs

Map DNN to a Matrix Multiplication

Fully connected layer can be directly represented as matrix multiplication



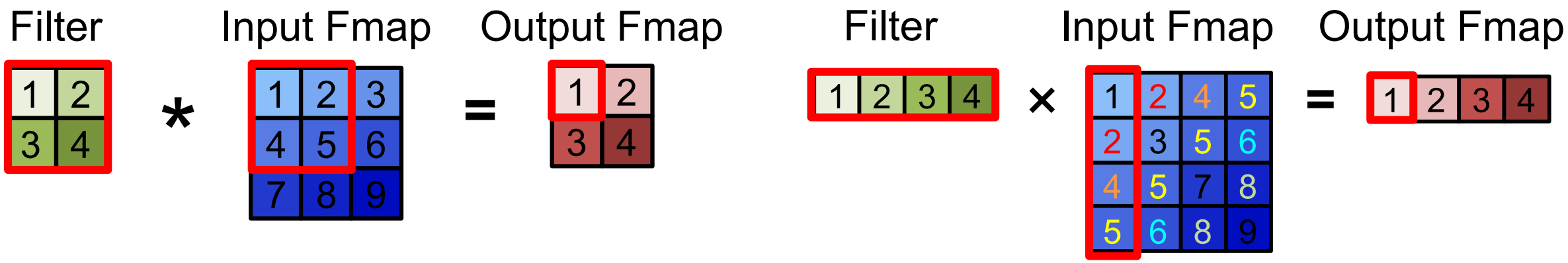


In fully connected layer, filter size (R, S) same as input size (H, W)

Note: Matrix multiplication also heavily used by recurrent and attention layers

Map DNN to a Matrix Multiplication

Convolutional layer can be converted to Toeplitz Matrix



Convolution

Matrix Multiply (by Toeplitz Matrix)

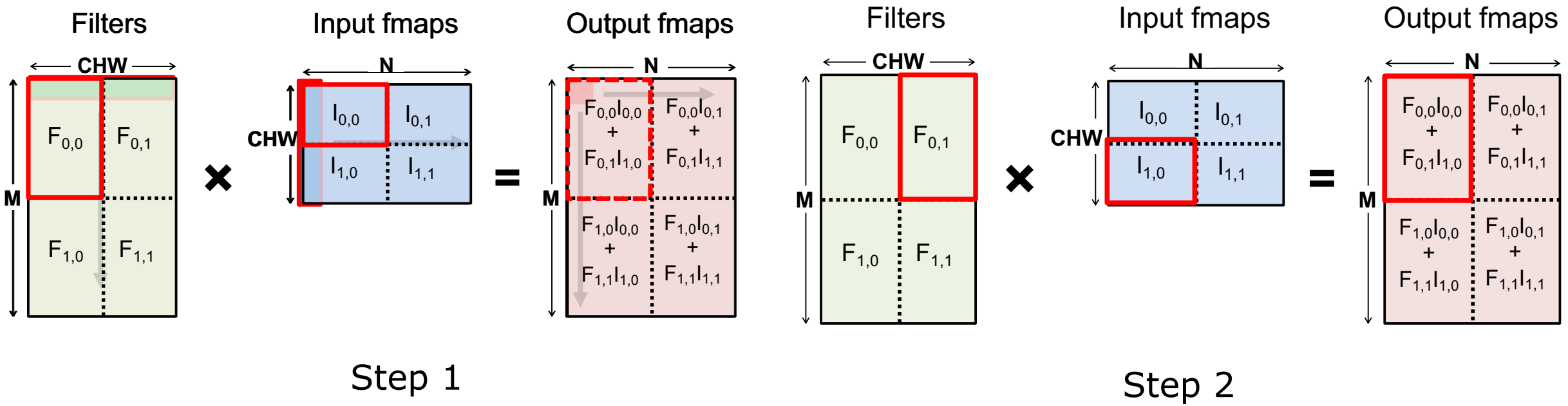
Data is repeated

CPU, GPU Libraries for Matrix Multiplication

- Implementation: Matrix Multiplication (GEMM)
  - CPU: OpenBLAS, Intel MKL, etc
  - GPU: cuBLAS, cuDNN, etc
- Library will note shape of the matrix multiply and select implementation optimized for that shape
- Optimization usually involves proper tiling to memory hierarchy

Tiling Matrix Multiplication

Matrix multiplication **tiled** to fit in cache (i.e., on-chip memory) and computation ordered to maximize reuse of data in cache



Analogy: Gauss's Multiplication Algorithm

$$(a+bi)(c+di) = (ac-bd) + (bc+ad)i.$$

4 multiplications + 3 additions

$k_1 = c \cdot (a + b)$

$k_2 = a \cdot (d - c)$

$k_3 = b \cdot (c + d)$

Real part $= k_1 - k_3$

Imaginary part $= k_1 + k_2$.

3 multiplications + 5 additions

Reduce number of multiplications to **increase** throughput

Reduce Operations in Matrix Multiplication

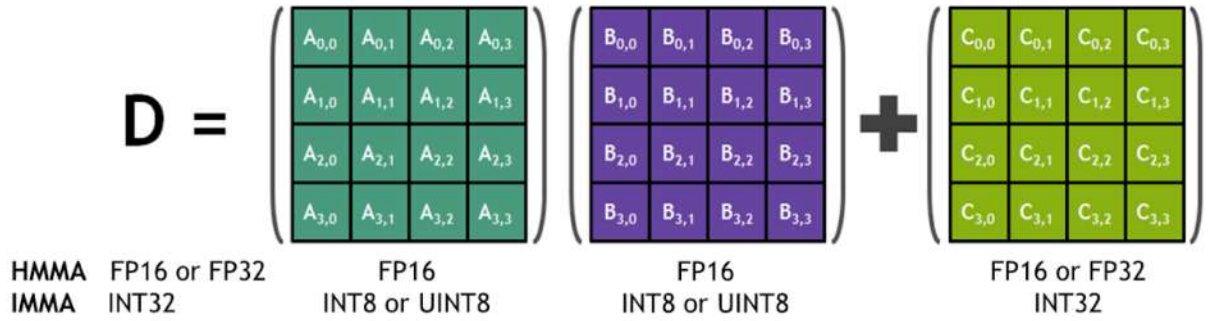
- **Fast Fourier Transform** [**Mathieu**, *ICLR* 2014]
  - Pro: Direct convolution $O(N_o^2 N_f^2)$ to $O(N_o^2 \log_2 N_o)$
  - Con: Increase storage requirements
- **Strassen** [**Cong**, *ICANN* 2014]
  - Pro: $O(N^3)$ to $(N^{2.807})$
  - Con: Numerical stability
- **Winograd** [**Lavin**, *CVPR* 2016]
  - Pro: 2.25x speed up for 3x3 filter
  - Con: Specialized processing depending on filter size

Compiler selects transform based on filter size

Reduce Instruction Overhead

- **Perform more MACs per instruction**
  - **CPU:** SIMD / Vector Instructions
    - e.g., Specialized Vector Neural Network Instructions (VNNI) fuse separate multiply and add instructions into single MAC instruction and avoid storing intermediate values in memory
  - **GPU:** SIMT / Tensor Instructions
    - e.g., New opcode Matrix Multiply Accumulate (HMMA) performs 64 MACs with Tensor Core
- **Perform more MACs per cycle without increasing memory bandwidth by adding support for reduced precision**
  - e.g., If access 512 bits per cycle, can perform **64** 8-bit MACs vs. **16** 32-bit MACs

Tensor Core
Image Source: Nvidia



Design Considerations for CPU and GPU

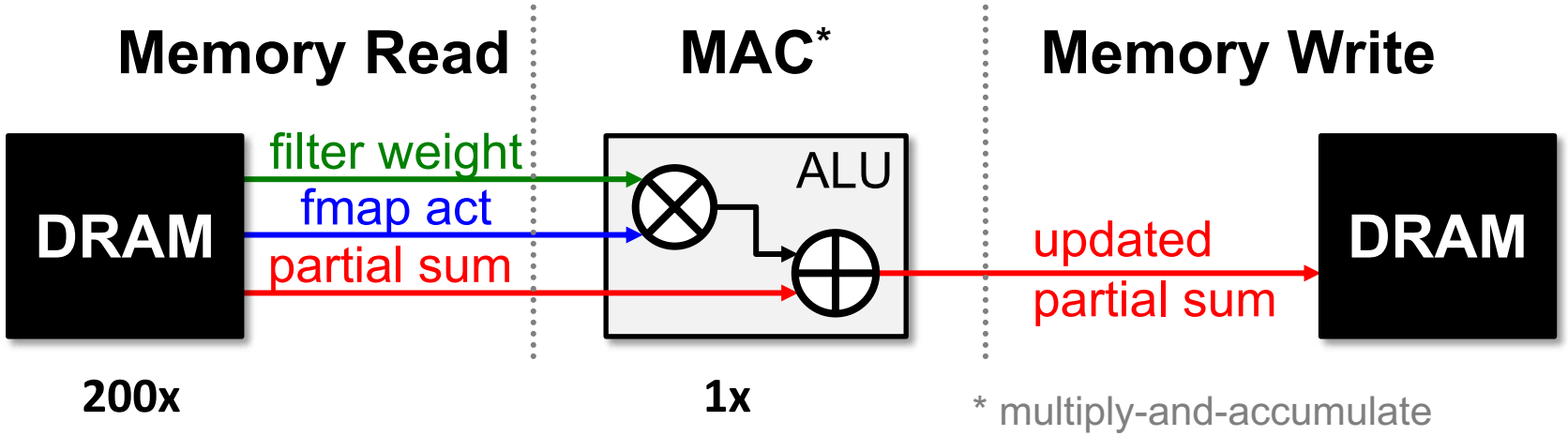
- **Software (compiler)**
  - **Reduce unnecessary MACs**: Apply transforms
  - **Increase PE utilization**: Schedule loop order and tile data to increase data reuse in memory hierarchy
- **Hardware**
  - **Reduce time per MAC**
    - Increase speed of PEs
    - Increase MACs per instruction using large aggregate instructions (e.g., SIMD, tensor core) → requires additional hardware
  - **Increase number of parallel MACs**
    - Increase number of PEs on chip → area cost
    - Support reduced precision in PEs
  - **Increase PE utilization**
    - Increase on-chip storage → area cost
    - External memory BW → system cost

Specialized / Domain-Specific Hardware

Properties We Can Leverage

- Operations exhibit **high parallelism**
  → **high throughput** possible

- Memory Access is the Bottleneck



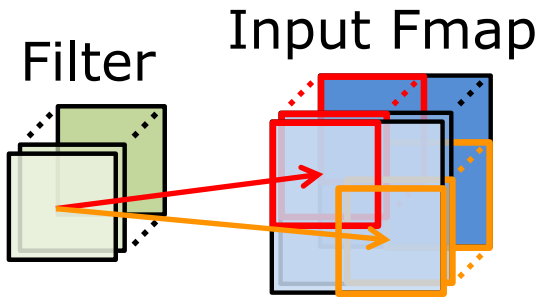
Worst Case: all memory R/W are **DRAM** accesses

Example: AlexNet has **724M** MACs → **2896M** DRAM accesses required

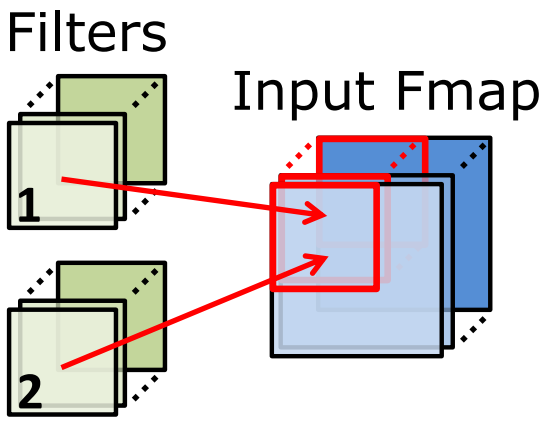
Properties We Can Leverage

- Operations exhibit **high parallelism**
  → **high throughput** possible

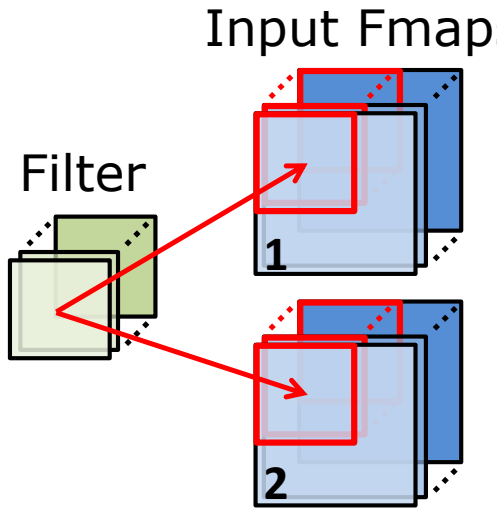
- **Input data reuse** opportunities (e.g., up to 500x for AlexNet)
  → exploit **low-cost memory**



Convolutional Reuse
(Activations, Weights)
CONV layers only
(sliding window)



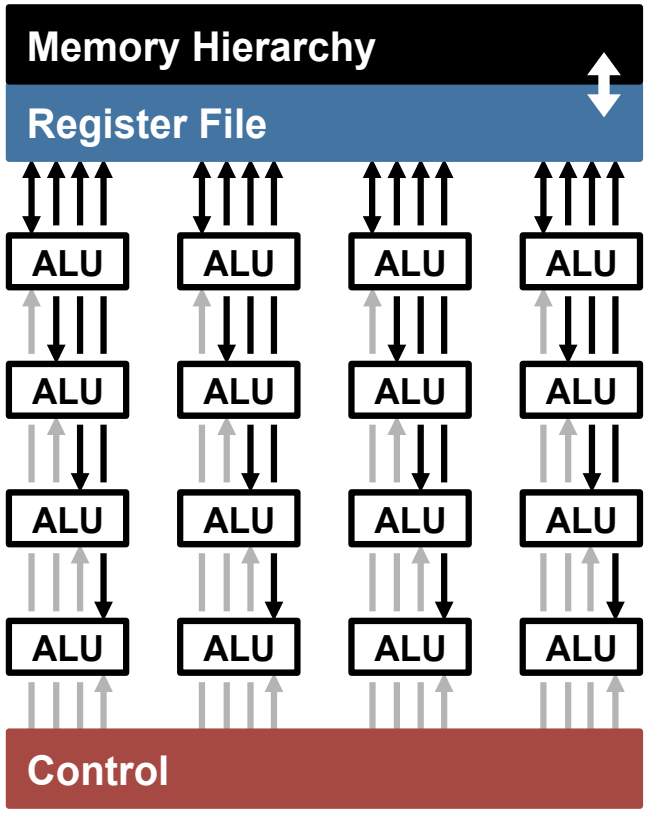
Fmap Reuse
(Activations)
CONV and FC layers



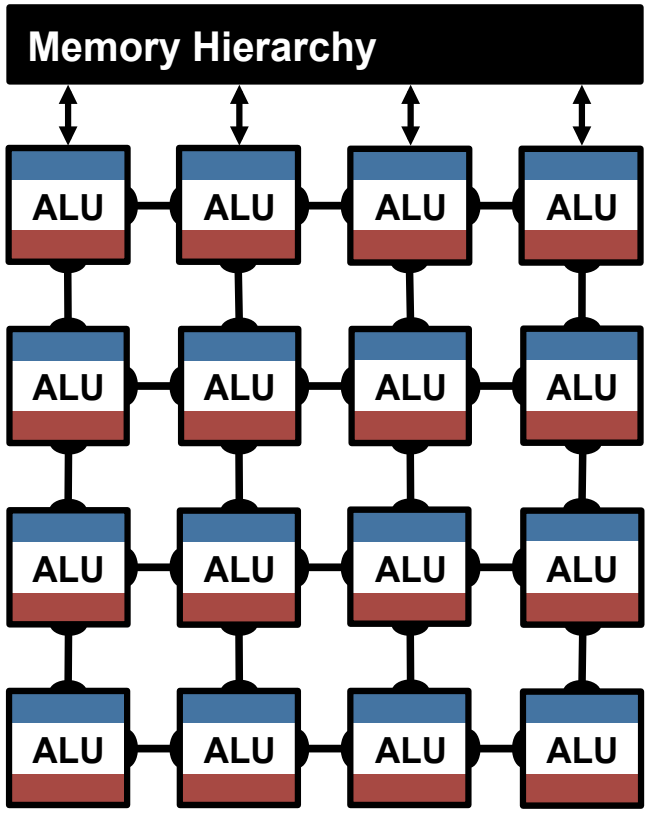
Filter Reuse
(Weights)
CONV and FC layers
(batch size > 1)

Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)



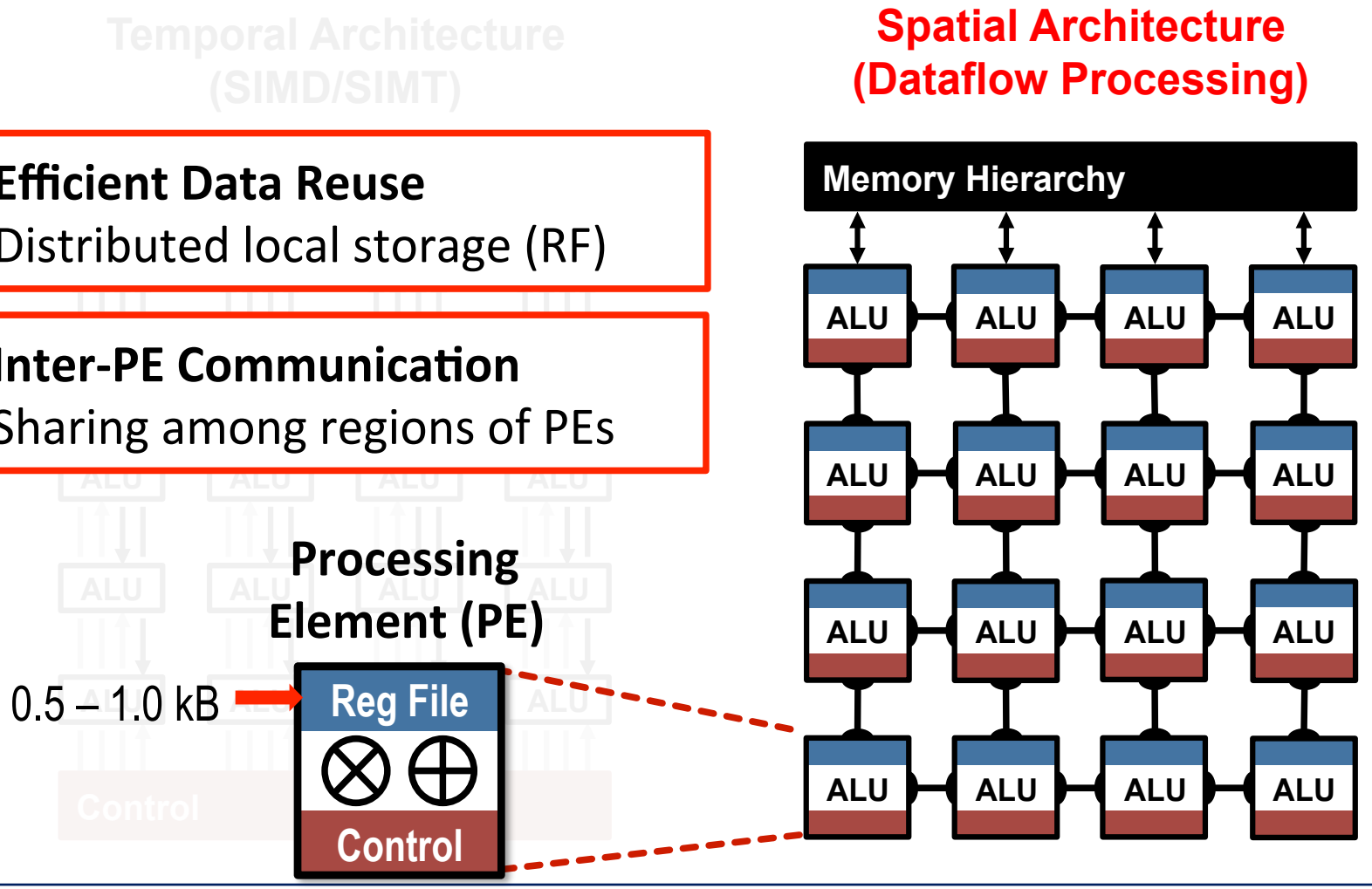
Spatial Architecture (Dataflow Processing)



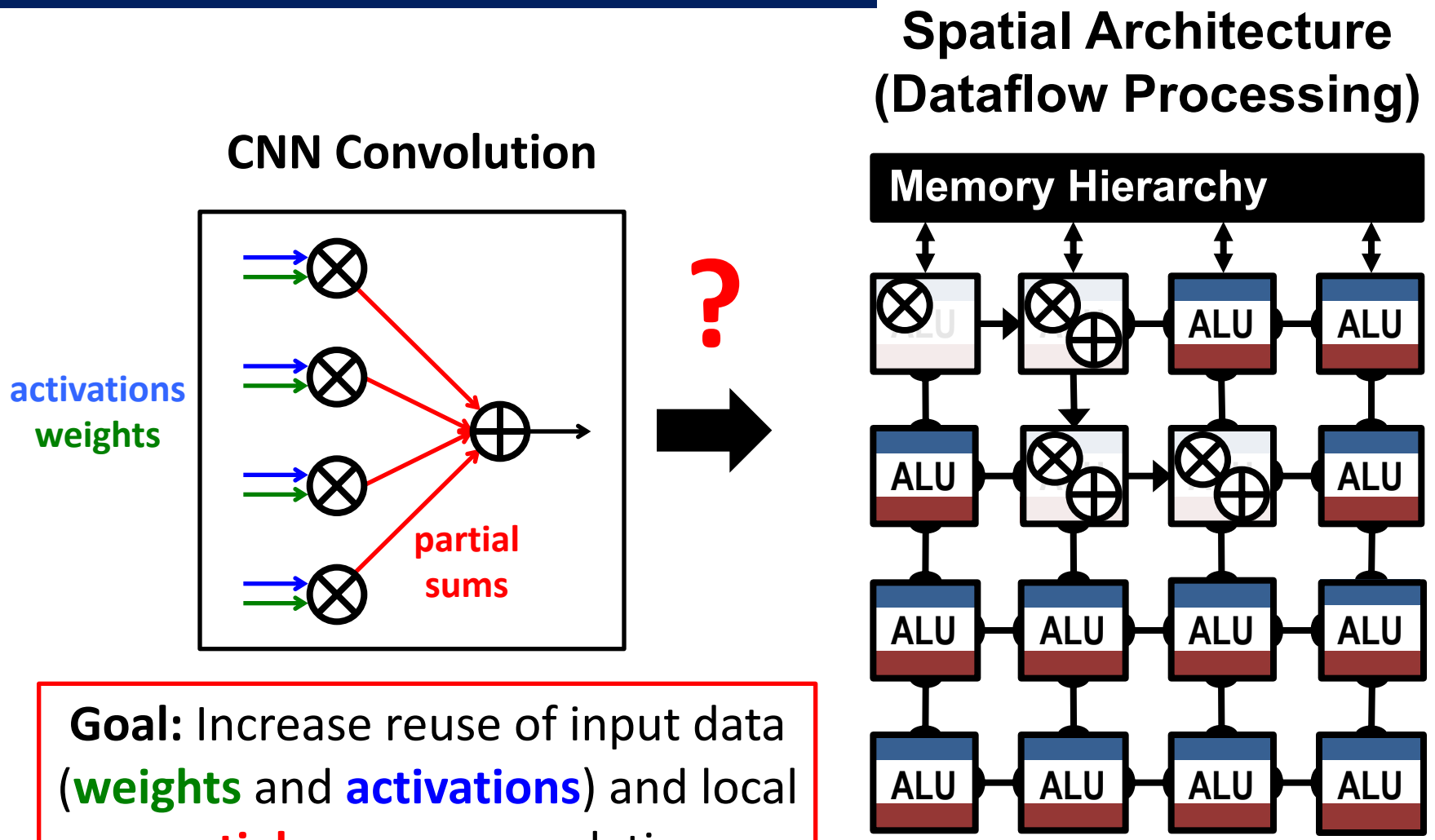
Advantages of Spatial Architecture

Efficient Data Reuse
Distributed local storage (RF)

Inter-PE Communication
Sharing among regions of PEs



How to Map the Dataflow?

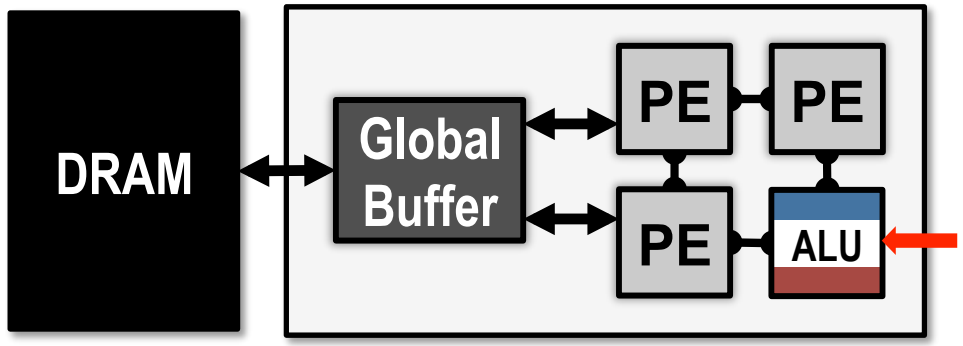


Goal: Increase reuse of input data (**weights** and **activations**) and local **partial sums** accumulation

Efficient Dataflows

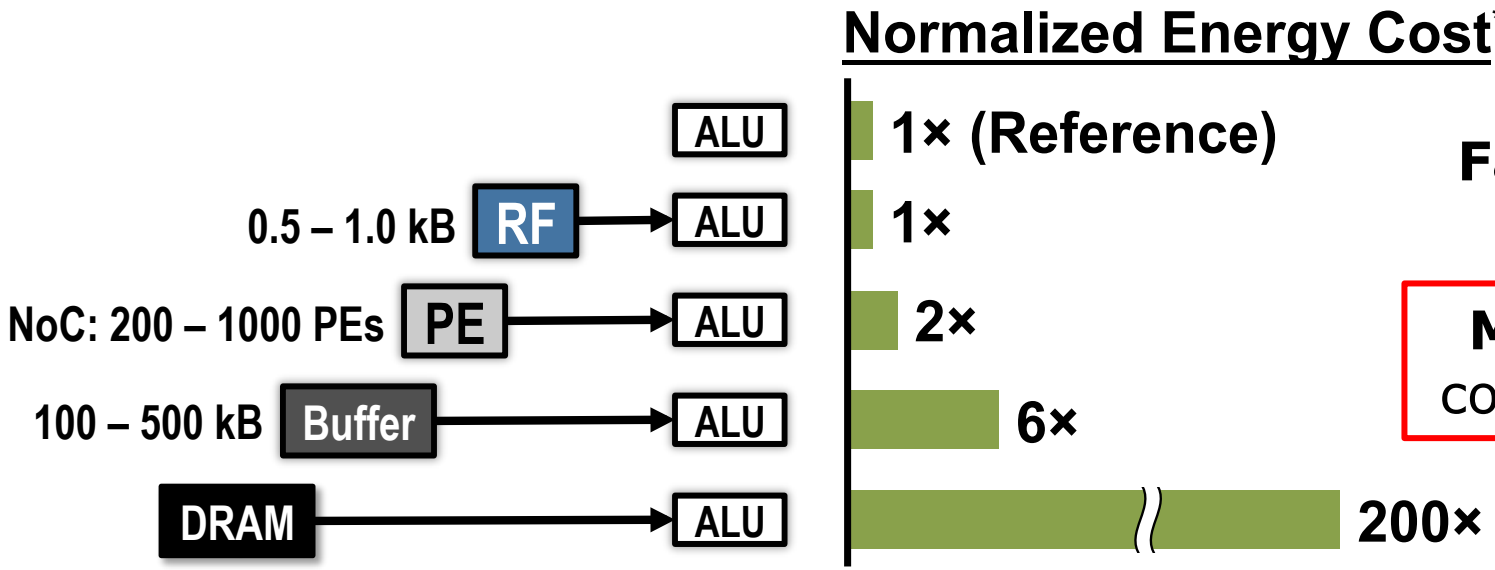
Y.-H. Chen, J. Emer, V. Sze,
"**Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks**,"
International Symposium on Computer Architecture (ISCA), June 2016.

Data Movement is Expensive



Specialized hardware with small (< 1kB) low cost memory near compute

fetch data to run a MAC here



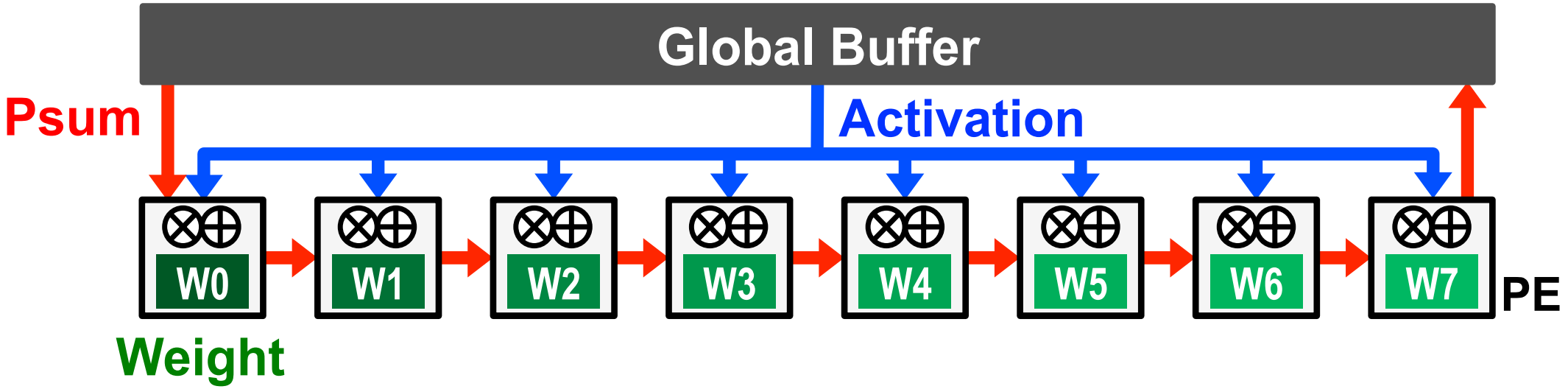
Farther and **larger** memories consume more power

Maximize data reuse at low cost levels of memory hierarchy

* measured from a commercial 65nm process

Weight Stationary (WS)

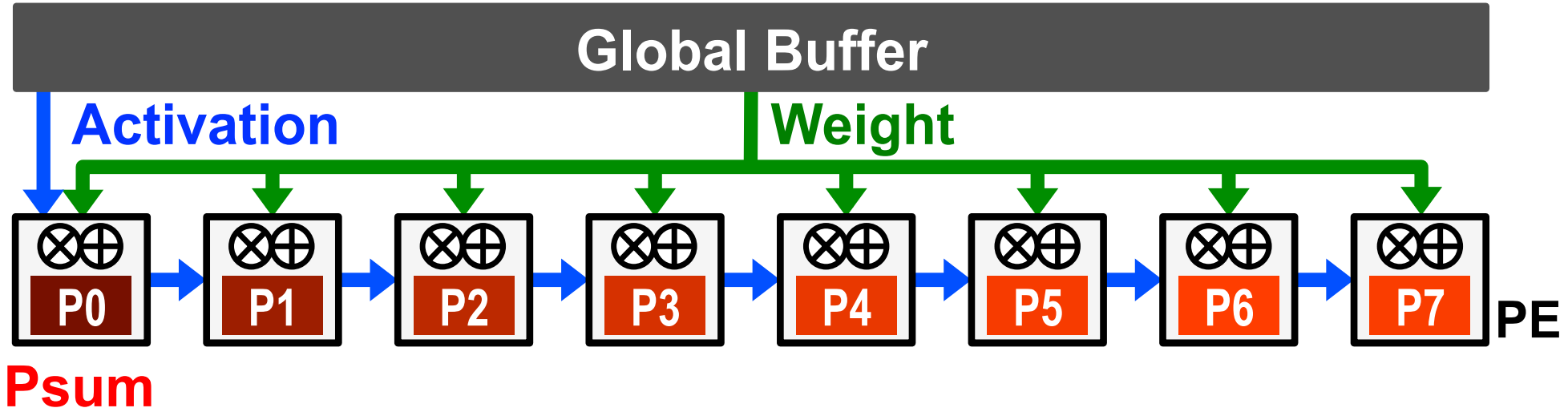
[**Chen**, *ISCA* 2016]



- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights
- **Broadcast activations** and **accumulate partial sums spatially** across the PE array
- Examples: **TPU** [**Jouppi**, *ISCA* 2017], **NVDLA**

Output Stationary (OS)

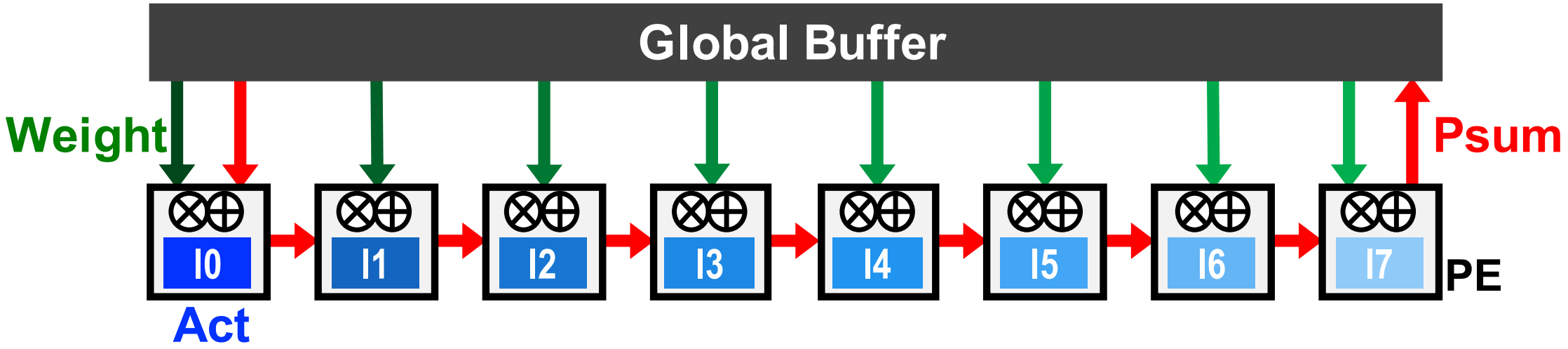
[**Chen**, *ISCA* 2016]



- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array
- Examples: [**Moons**, *VLSI* 2016], [**Thinker**, *VLSI* 2017]

Input Stationary (IS)

[**Chen**, *ISCA* 2016]

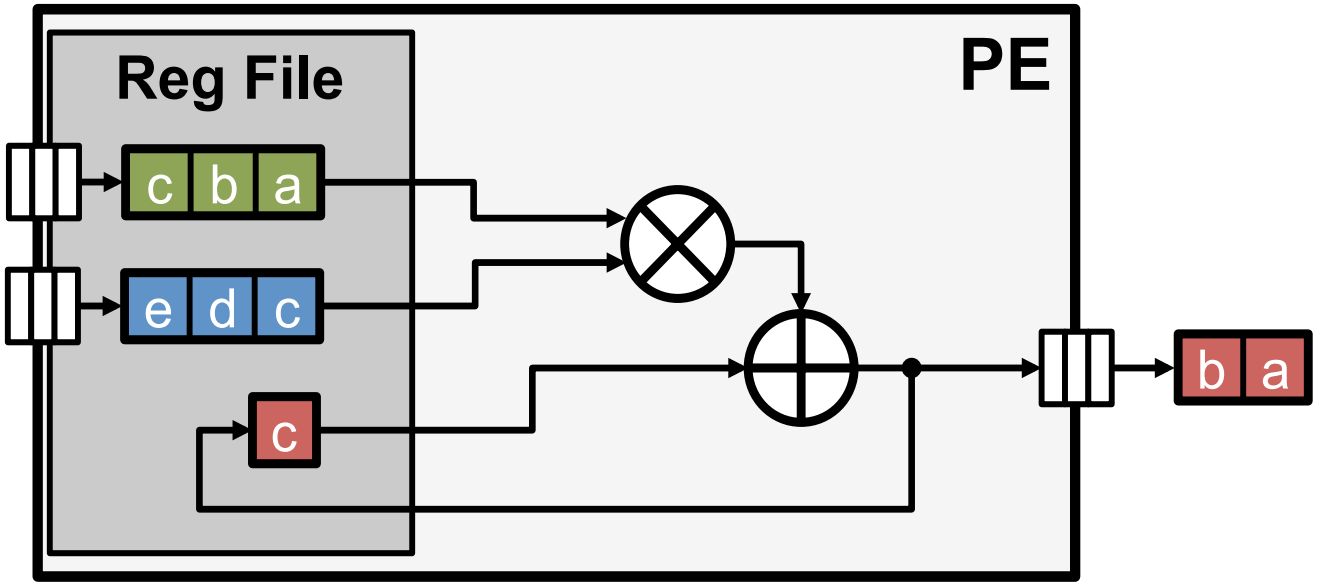


- **Minimize activation** read energy consumption
  - maximize convolutional and fmap reuse of activations
- **Unicast weights** and **accumulate partial sums spatially** across the PE array
- Example: [**SCNN**, *ISCA* 2017]

Row Stationary Dataflow

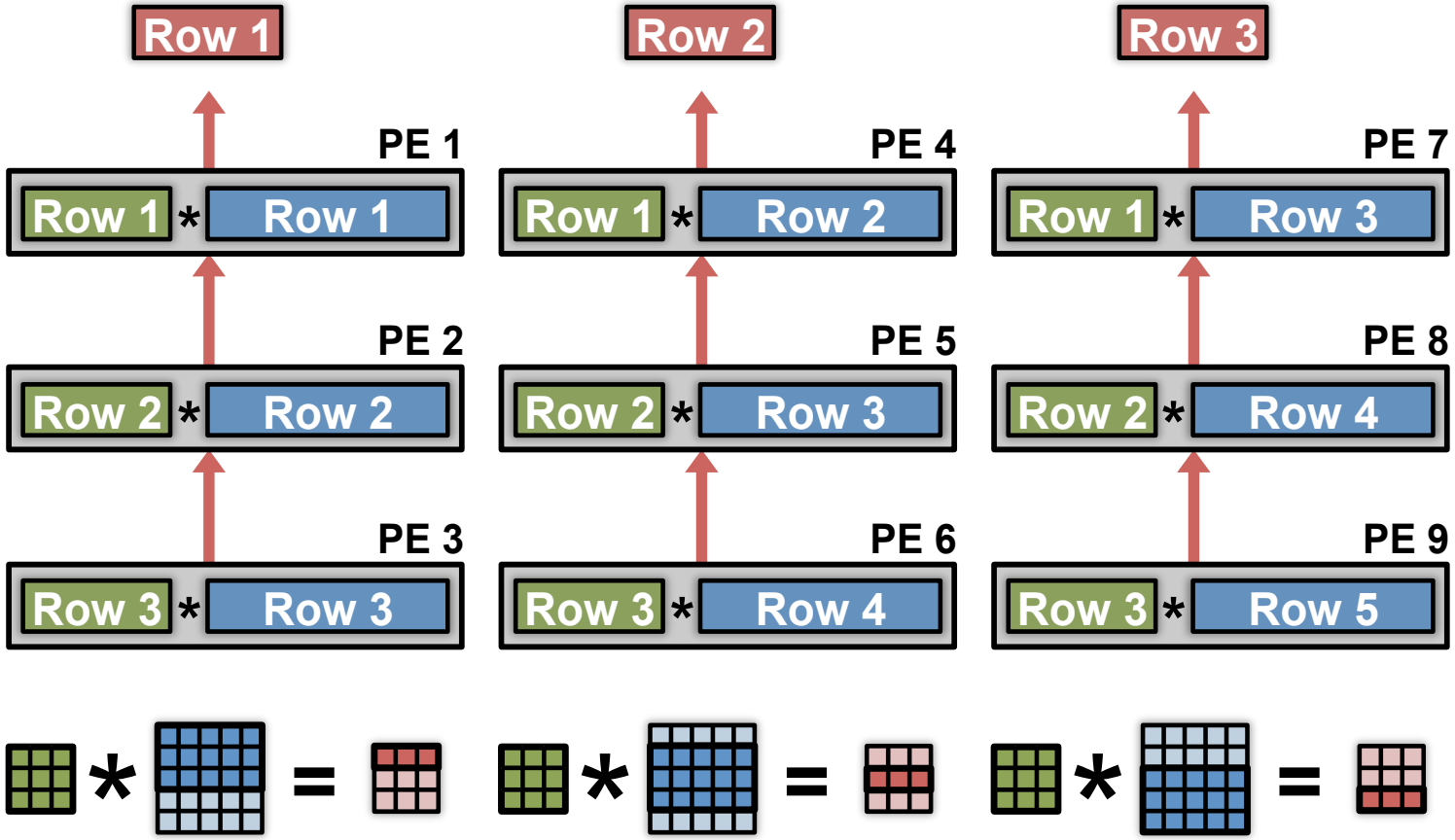
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF

1D convolution within PE



[**Chen**, *ISCA* 2016]

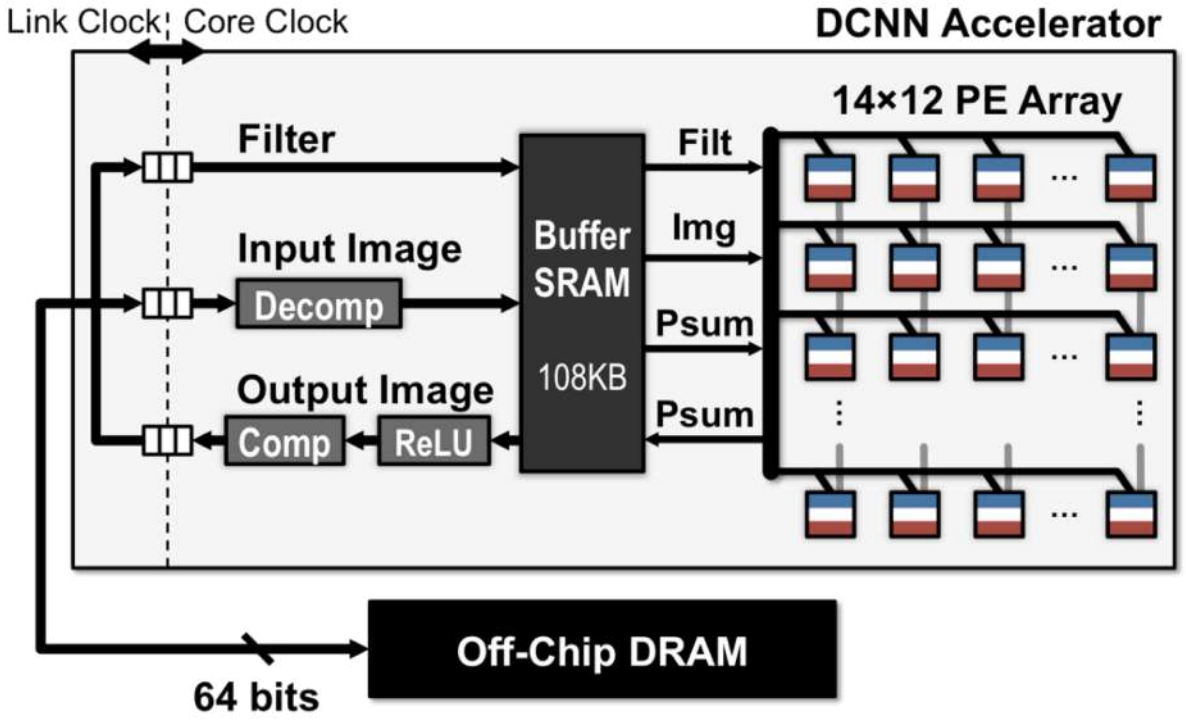
Row Stationary Dataflow

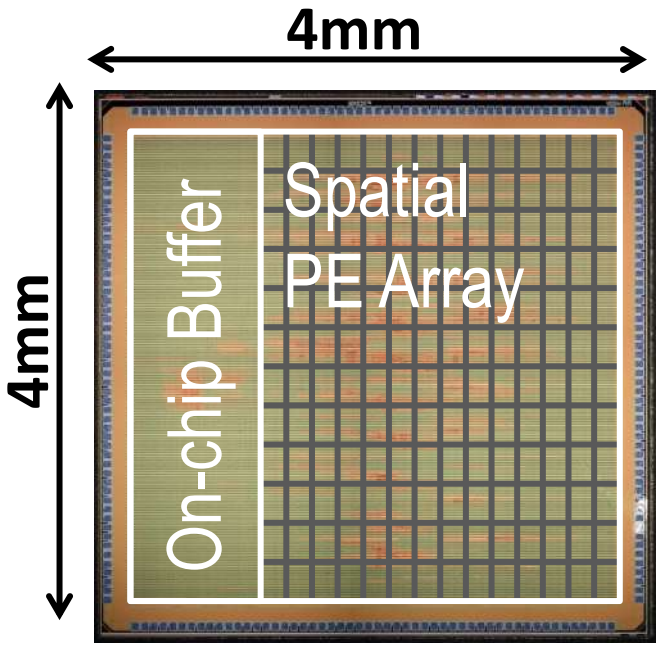


Optimize for **overall energy efficiency** instead for only a certain data type

[**Chen**, *ISCA* 2016]

Eyeriss: Deep Neural Network Accelerator





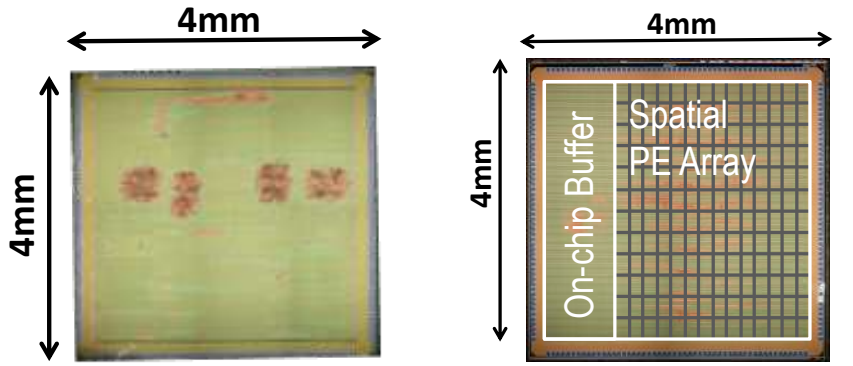
[**Chen**, *ISSCC* 2016]

Exploits data reuse for **100x** reduction in memory accesses from global buffer and **1400x** reduction in memory accesses from off-chip DRAM

Overall **>10x energy reduction** compared to a mobile GPU (Nvidia TK1)

Features: Energy vs. Accuracy

Measured in 65nm*

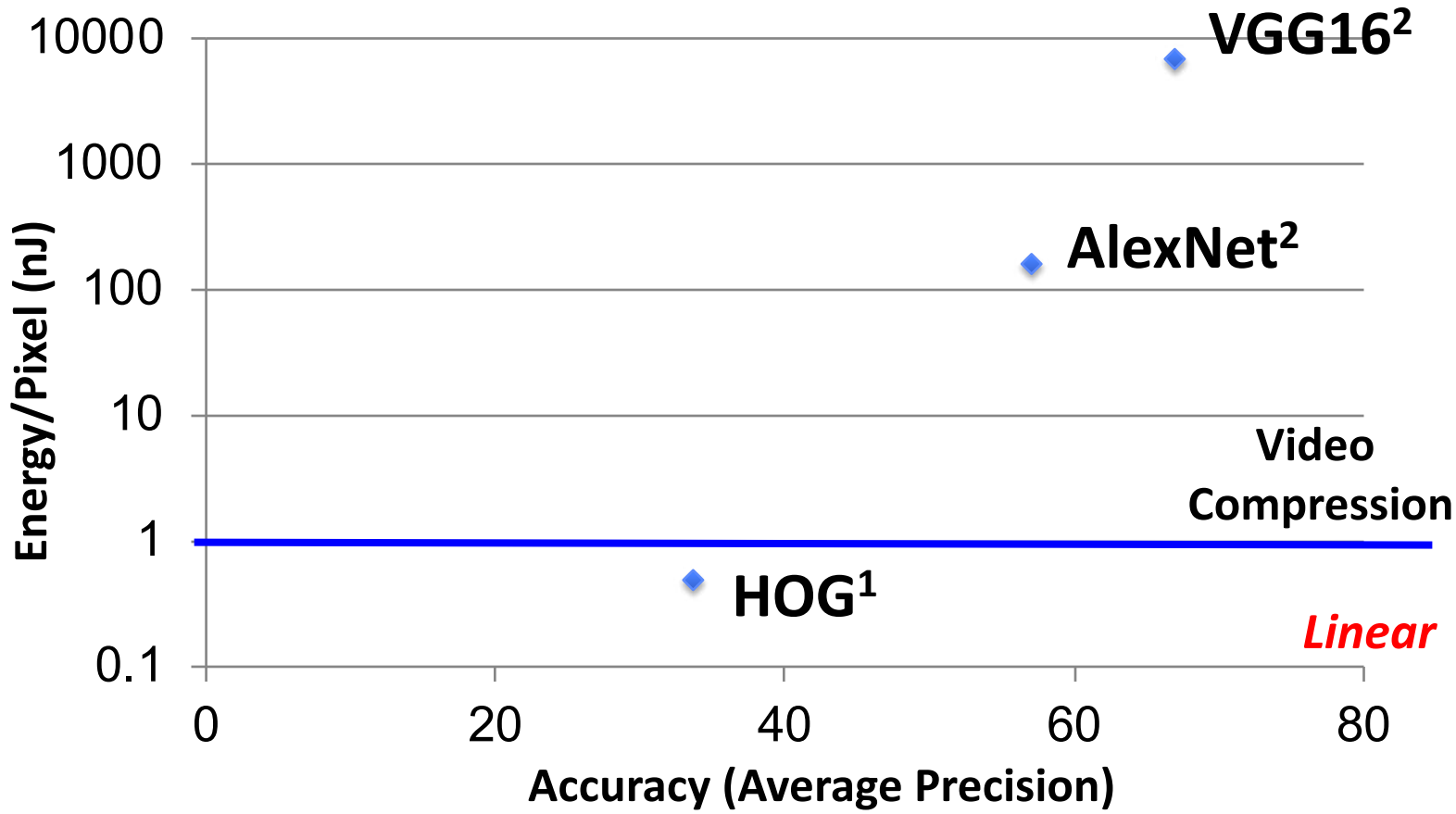


1 [**Suleiman**, *VLSI* 2016] 2 [**Chen**, *ISSCC* 2016]

** Only feature extraction. Does not include data, classification energy, augmentation and ensemble, etc.*

[**Suleiman**, *ISCAS* 2017]

Exponential



Measured in on VOC 2007 Dataset

1. DPM v5 [Girshick, 2012]
2. Fast R-CNN [Girshick, CVPR 2015]

Break for Q&A

Algorithm (DNN Model) & Hardware Co-Design

Algorithm & Hardware Co-Design

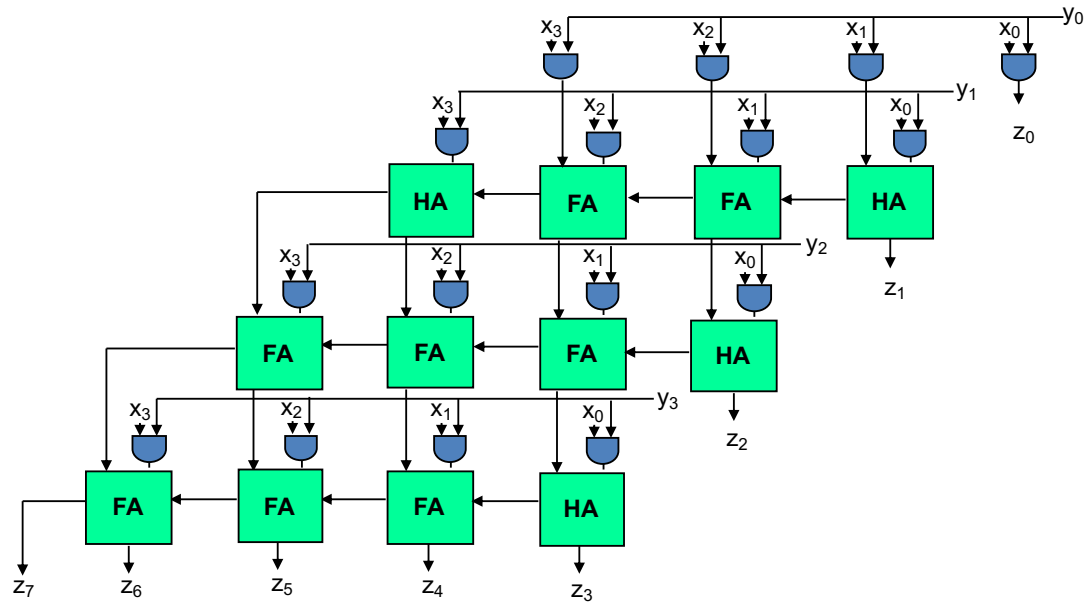
- Co-design algorithm + hardware → better than what each could achieve alone
- Co-design approaches can be loosely grouped into two categories:
  - Reduce ***size*** of operands for storage/compute (**Reduced Precision**)
  - Reduce ***number*** of operations for storage/compute (**Sparsity** and **Efficient Network Architecture**)
- **Hardware support required** to increase savings in latency and energy
  - Ensure that overhead of hardware support does not exceed benefits
- Unlike previously discussed approaches, these approaches can **affect accuracy**!
  - Evaluate tradeoff between accuracy and other metrics

Reduced Precision

Why Reduce Precision (i.e., Reduce Bit Width)?

- **Reduce data movement** and storage cost for inputs and outputs of MAC
  - Smaller memory → lower energy
- **Reduce cost of MAC**
  - Cost of multiply increases with bit width (n) → energy and area by $O(n^2)$; delay by $O(n)$

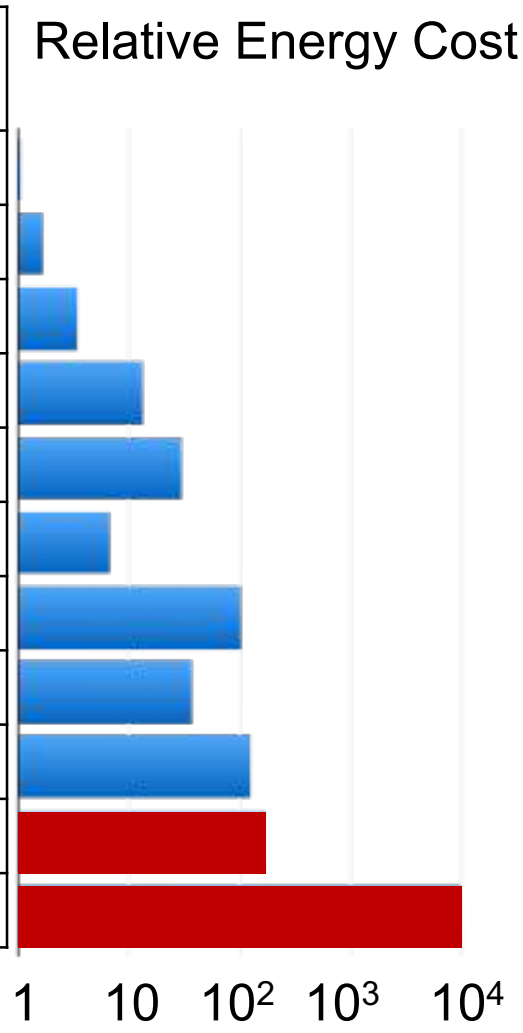
$$
\begin{array}{cccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 \quad \text{Multiplicand} \\
 & & & \times & y_3 & y_2 & y_1 & y_0 \quad \text{Multiplier} \\
\hline
 & & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & \\
 & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & & \text{Partial Product} \\
 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & & \\
+ & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & \\
\hline
z_7 & z_6 & z_5 & z_4 & z_3 & z_2 & z_1 & z_0 \quad \text{Result}
\end{array}
$$

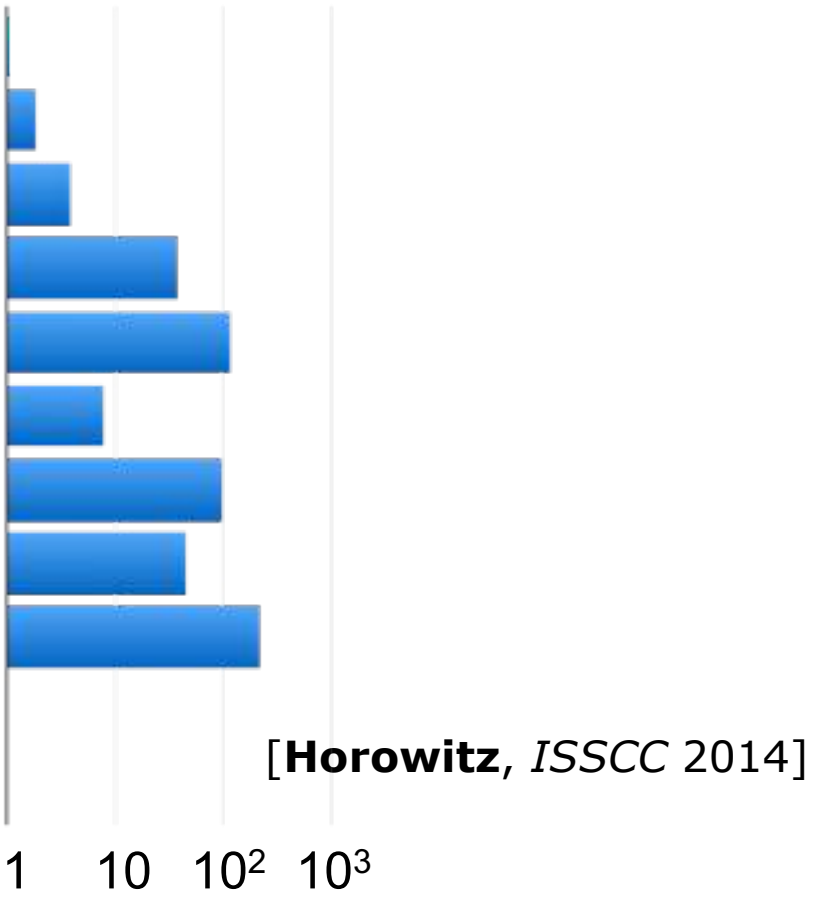


Note: Bit width for multiplication and accumulation in a MAC are different

Impact of Reduced Precision on Energy & Area

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





[**Horowitz**, *ISSCC* 2014]

What Determines Bit Width?

- Number of unique values
  - e.g., **M-bits** to represent 2^{M} values
- Dynamic range of values
  - e.g., **E-bits** to scale values by $2^{(E-127)}$
- Signed or unsigned values
  - e.g., signed requires one extra bit (**S**)
- **Total bits = S+E+M**

- **Floating point (FP)** allows range to change for each value (E-bits)
- **Fixed point (Int)** has fixed range

- Default CPU/GPU is 32-bit float (**FP32**)

Common Numerical Representations

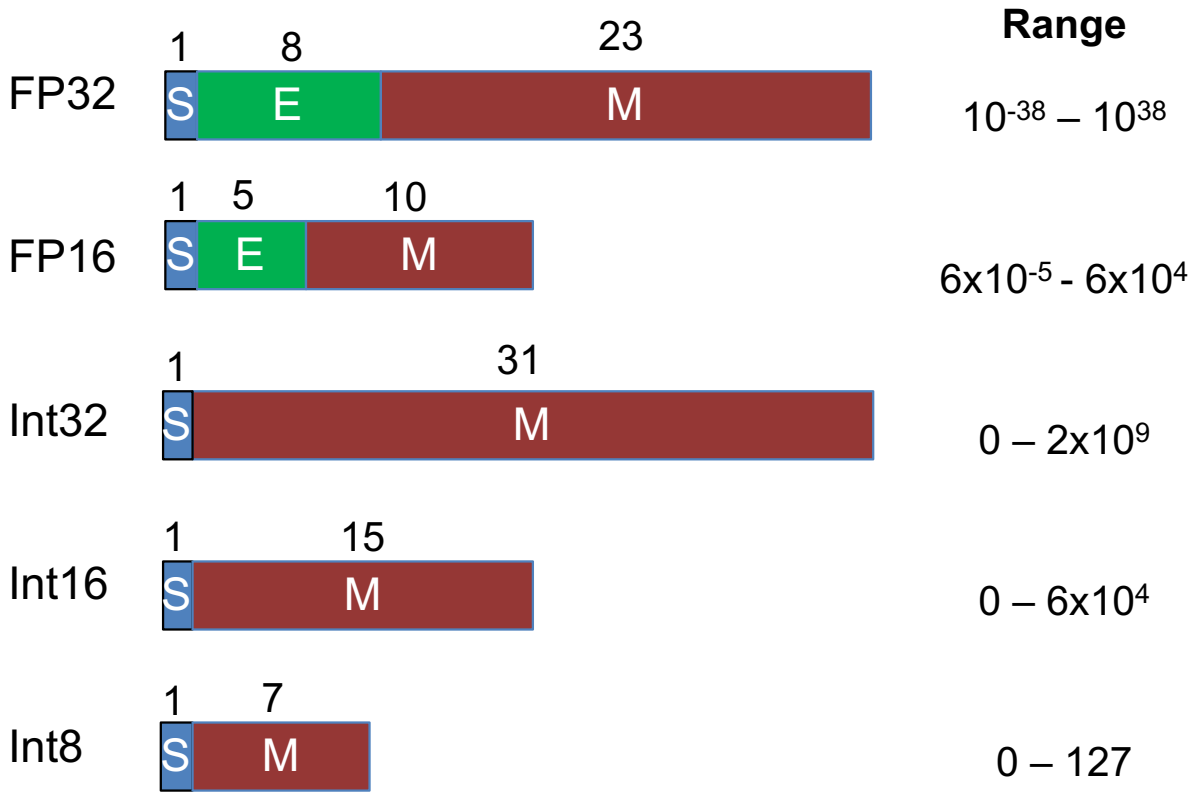


Image Source: B. Dally

What Determines Bit Width?

- For accuracy, require sufficient precision to represent different data types
  - **For inference:** weights, activations, and partial sums
  - **For training:** weights, activations, partial sums, ***gradients***, and ***weight update***
  - Required precision can vary across data types
    - Referred to as **mixed precision**

What Determines Bit Width?

- **Reduce number of unique values** (**M-bits**, a.k.a. mantissa)
  - **Default:** Uniform quantization (values are equally spaced out)
  - Non-uniform quantization (spacing can be computed, e.g., logarithmic, or with look-up-table)
  - Fewer unique values can make transforms and compression more effective

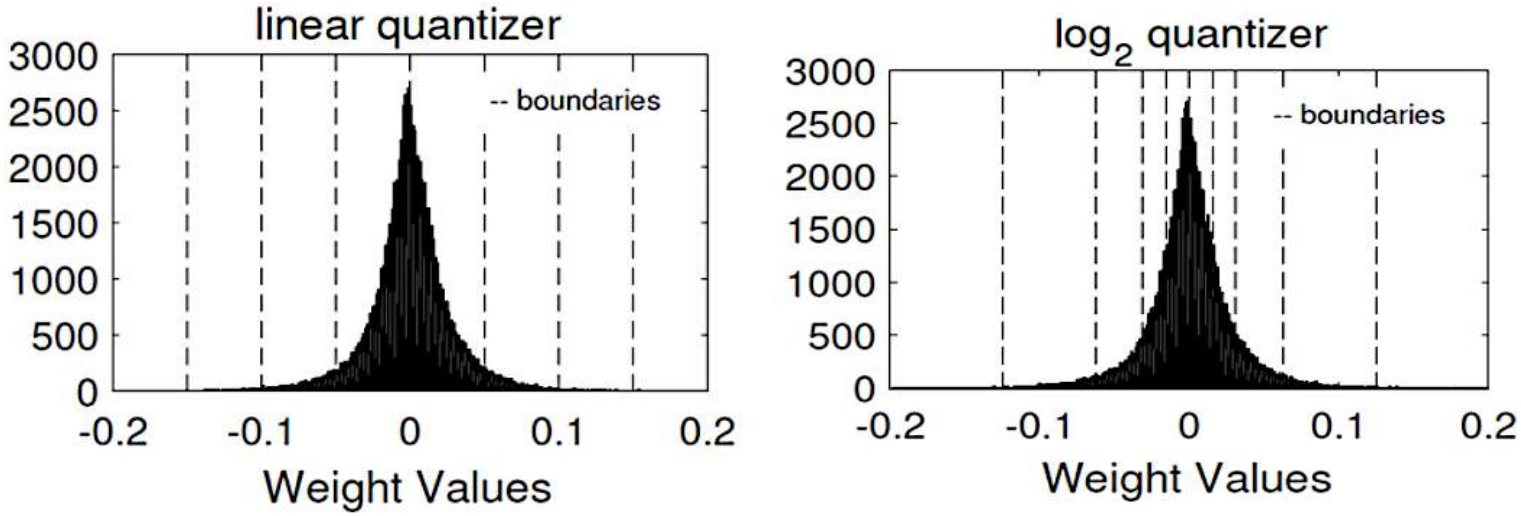


Image Source:[**Lee**, *ICASSP* 2017]

What Determines Bit Width?

- **Reduce number of unique values** (**M-bits**, a.k.a. mantissa)
  - **Default:** Uniform quantization (values are equally spaced out)
  - Non-uniform quantization (spacing can be computed, e.g., logarithmic, or with look-up-table)
  - Fewer unique values can make transforms and compression more effective
- **Reduce dynamic range** (**E-bits**, a.k.a., exponent)
  - If possible, fix range (i.e., used fixed point, E=0)
  - Share range across group of values (e.g., weights for a layer or channel)
- Tradeoff between number of bits allocated to **M-bits** and **E-bits**

fp16 (S=1, E=5, M=10) 

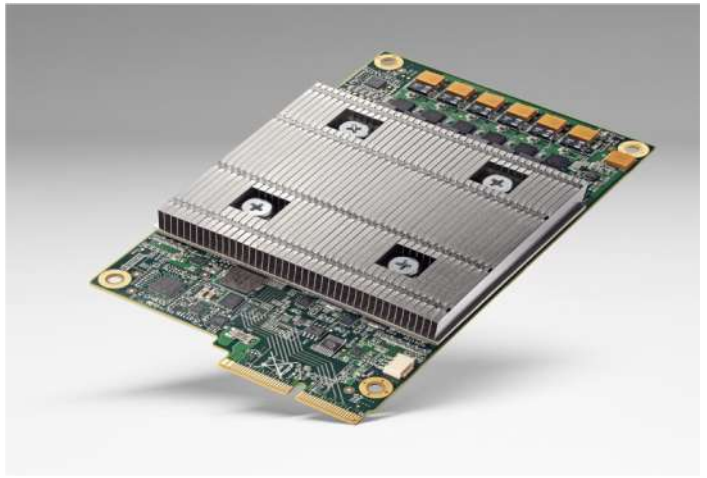
range: ~$5.9e^{-8}$ to ~$6.5e^{4}$

bfloat16 (S=1, E=8, M=7) 

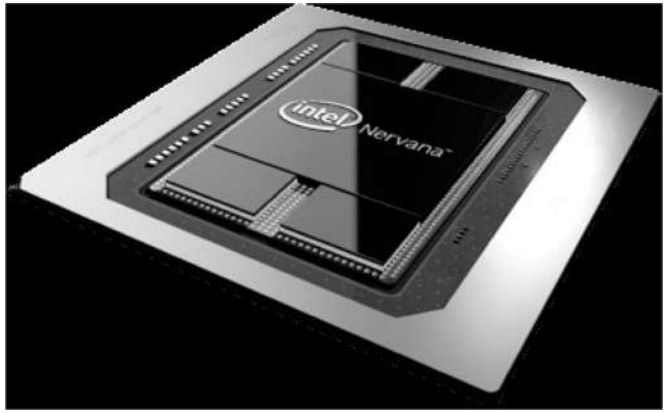
range: ~$1e^{-38}$ to ~$3e^{38}$

Commercial Products Support Reduced Precision

Nvidia's Pascal (2016)



Google's TPU (2016)
TPU v2 & v3 (2019)



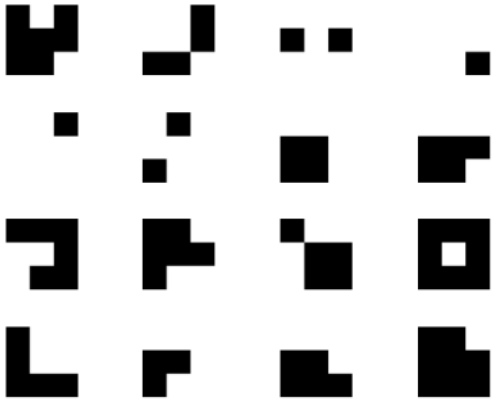
Intel's NNP-L (2019)

8-bit fixed for Inference & **16-bit float** for Training

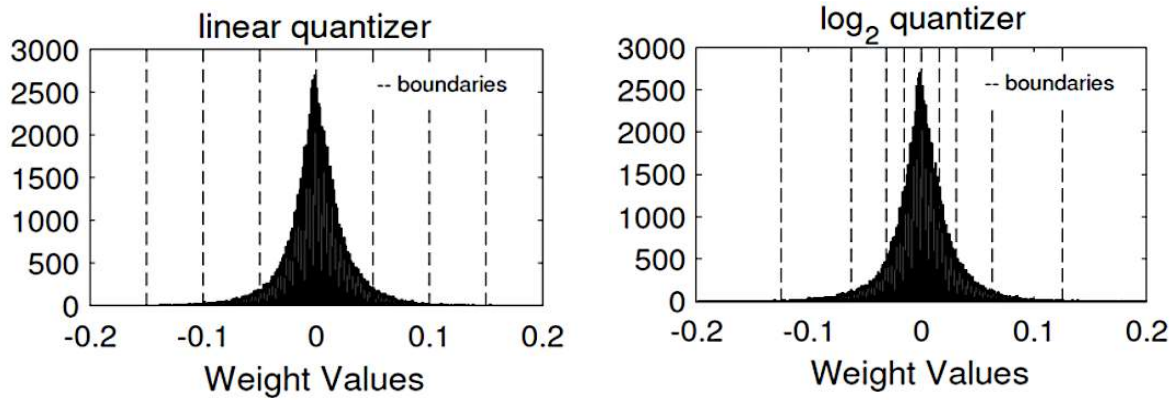
Reduced Precision in Research

- Reduce number of bits
  - Binary Nets [**Courbariaux**, *NeurIPS* 2015]
- Reduce number of unique weights and/or activations
  - Ternary Weight Nets [**Li**, *NeurIPS Workshop* 2016]
  - XNOR-Net [**Rategari**, *ECCV* 2016]
- Non-Linear Quantization
  - LogNet [**Lee**, *ICASSP* 2017]
- Training
  - 8-bit with stochastic rounding [**Wang**, *NeurIPS* 2018]

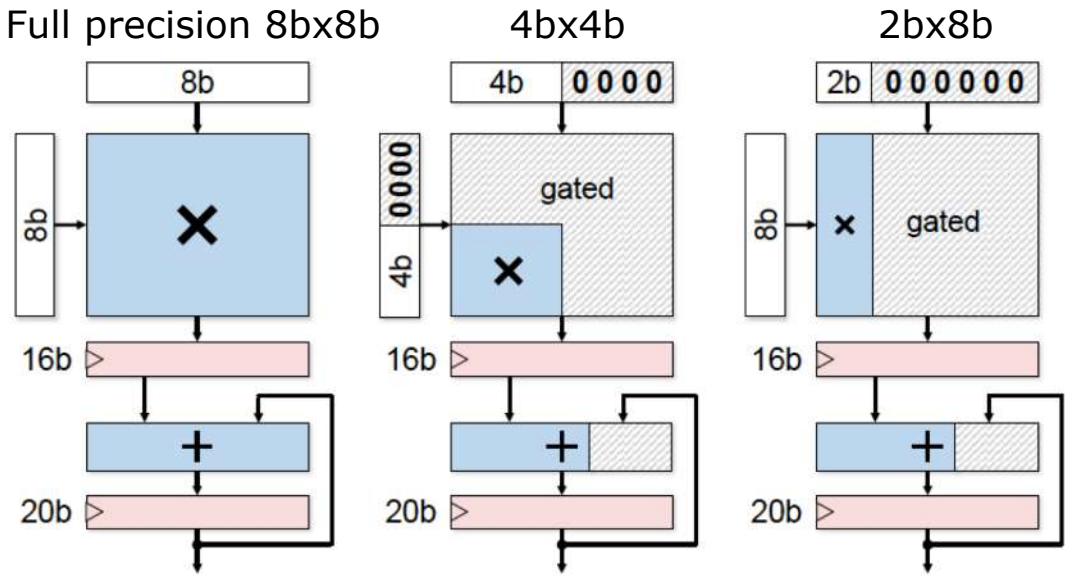
Binary Filters



Log Domain Quantization

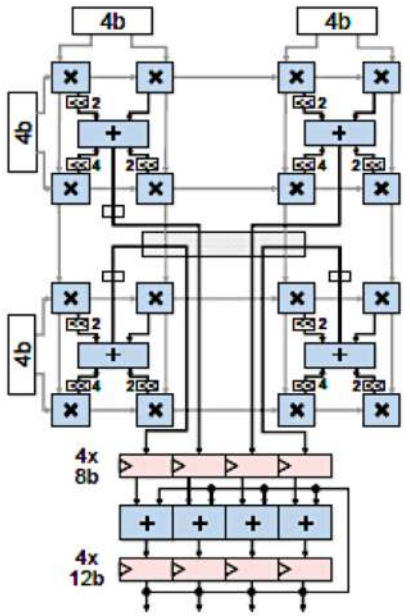


Precision Scalable MACs for ***Varying*** Precision

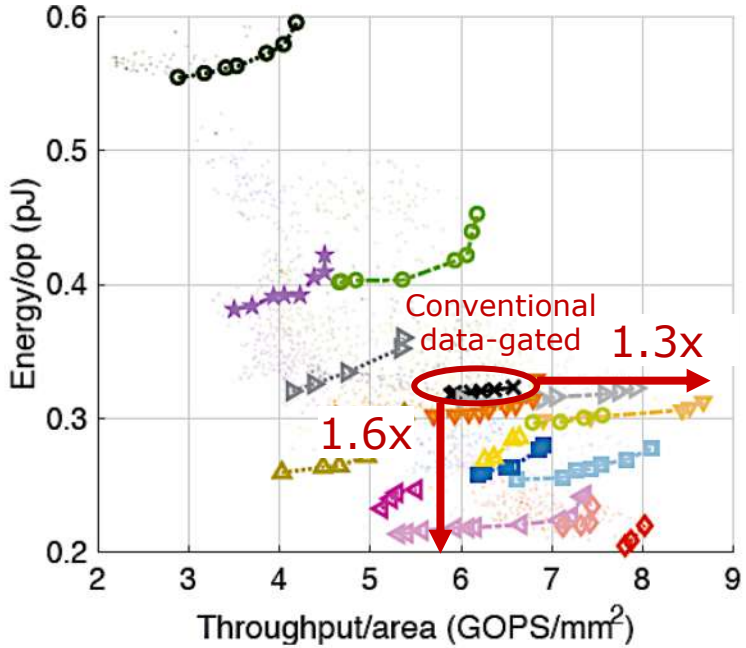


Conventional data-gated MAC
Gate unused logic (e.g., full adders) to reduce energy consumption

[**Camus**, *JETCAS* 2019]



Evaluation of **19 precision scalable MAC designs**
5% of values 8bx8b
95% of values at 2bx2b and 4bx4b



Many approaches add logic to increase utilization for higher throughput/area; however, **overhead can reduce benefits**

Design Considerations for Reduced Precision

- **Impact on accuracy**
  - Must consider difficulty of dataset, task, and DNN model
    - e.g., Easy to reduce precision for an easy task (e.g., digit classification); does method work for a more difficult task?
- **Does hardware cost exceed benefits?**
  - Need extra hardware to support variable precision
    - e.g., Additional shift-and-add logic and registers for variable precision
  - Granularity impacts hardware overhead as well as accuracy
    - e.g., More overhead to support (1b, 2b, 3b … 16b) than (2b, 4b, 8b, 16b)
- **Evaluation**
  - Use 8-bit for inference and 16-bit float for training for baseline
  - 32-bit float is a weak baseline

Sparsity

Why Increase Sparsity?

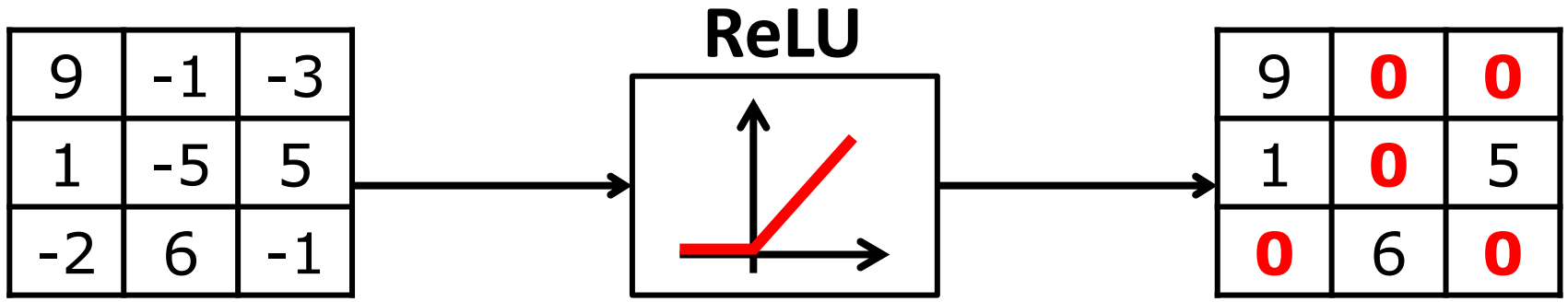
- **Reduce number of MACs**
  - Anything multiplied by zero is zero → avoid performing unnecessary MACs
  - Reduce energy consumption and latency

- **Reduce data movement**
  - If one of the inputs to MAC is zero, can avoid reading the other input
  - Compress data by only sending non-zero values

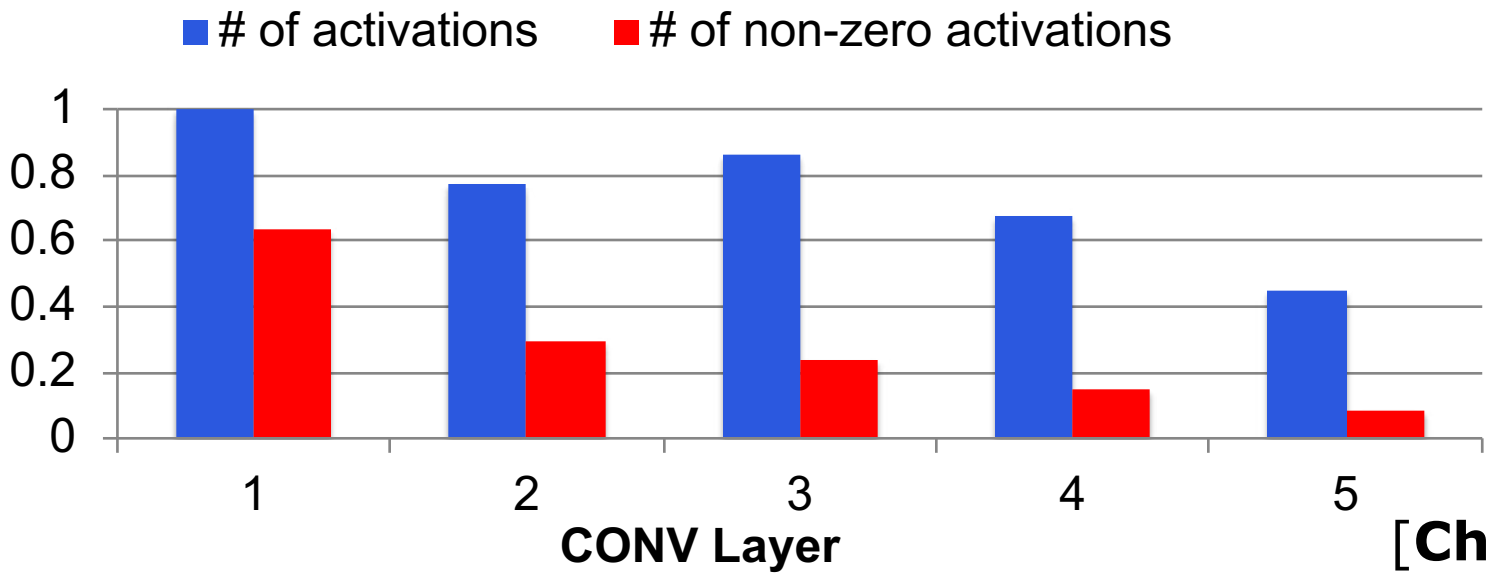
- CPU/GPU libraries typically only support really high sparsity (> 99%) due to the overhead
  - Sparsity for DNNs typically much lower → need specialized hardware

Sparsity in Activation Data

Many zeros in output fmaps after ReLU



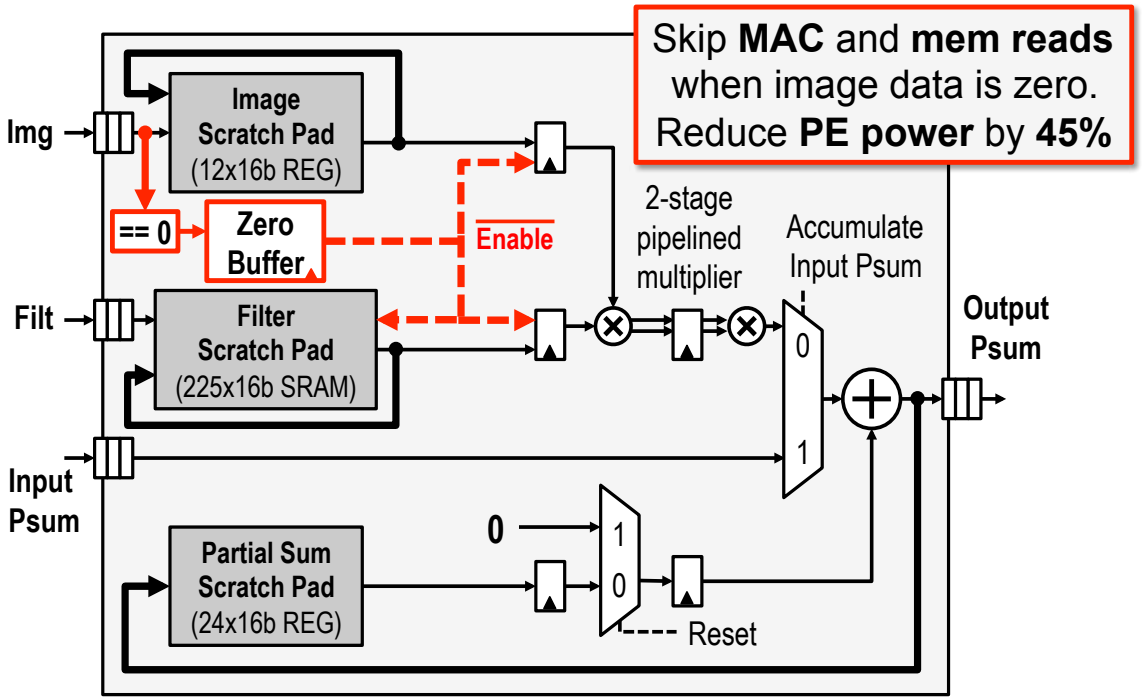
(Normalized)



[**Chen**, *ISSCC* 2016]

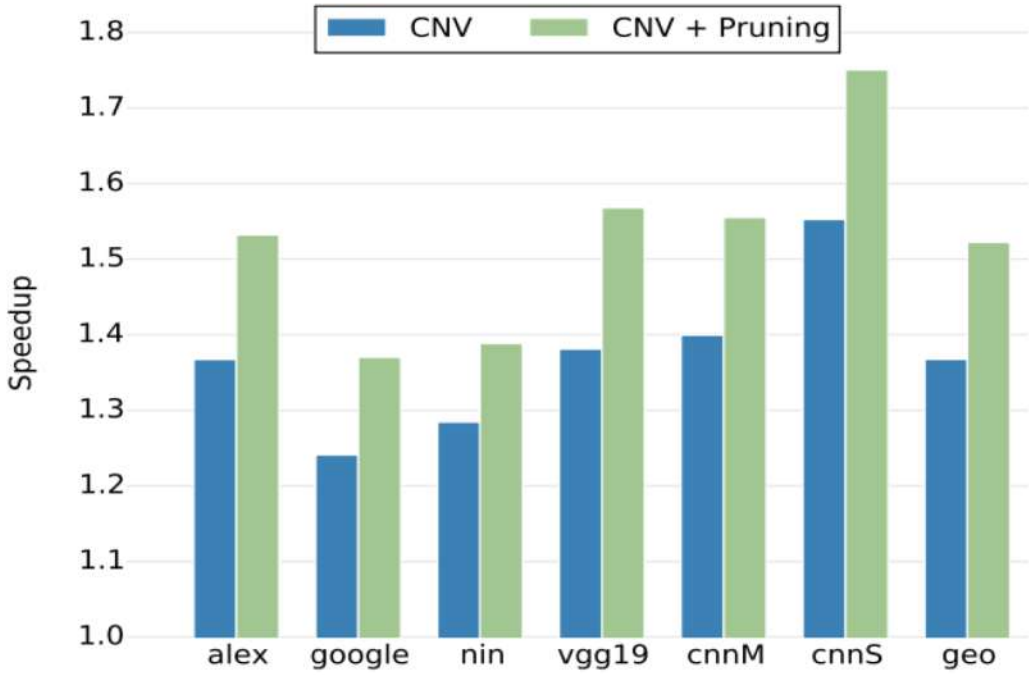
Data Gating / Zero Skipping

Gate operations
(reduce power consumption)



Eyeriss [**Chen**, *ISSCC* 2016]

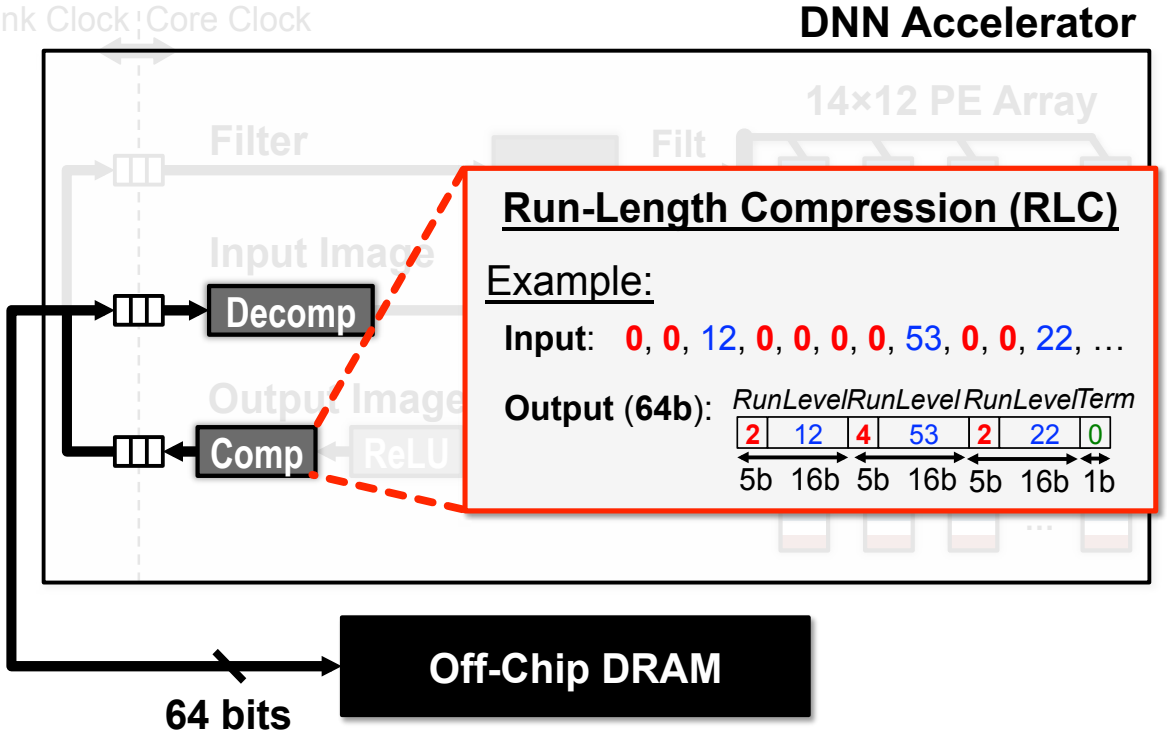
Skip operations
(increase throughput)

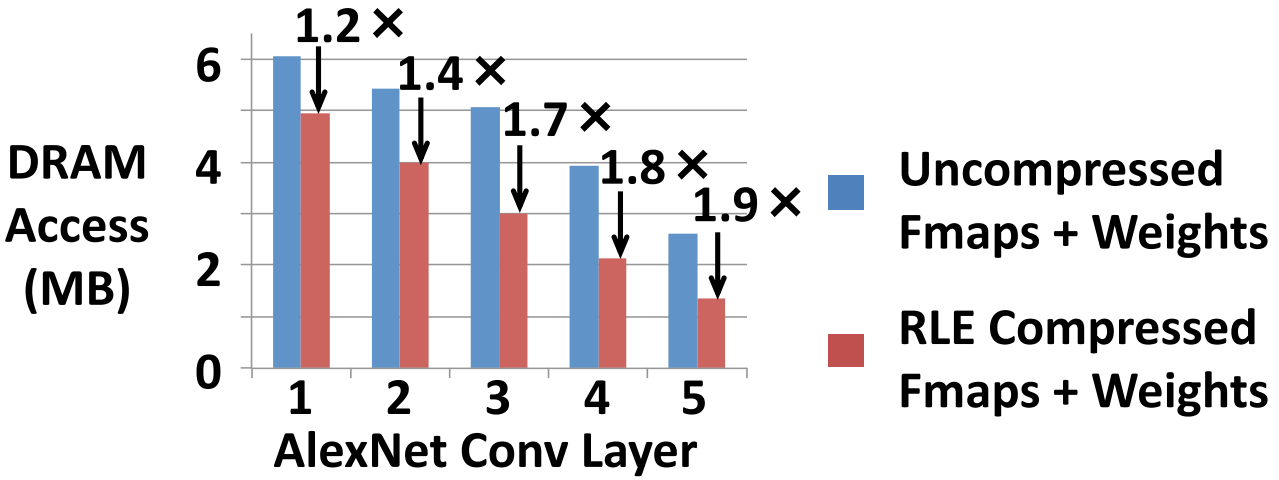


Cnvlutin [**Albericio**, *ISCA* 2016]

Apply Compression to Reduce Data Movement

Example: Eyeriss compresses activations to reduce DRAM BW



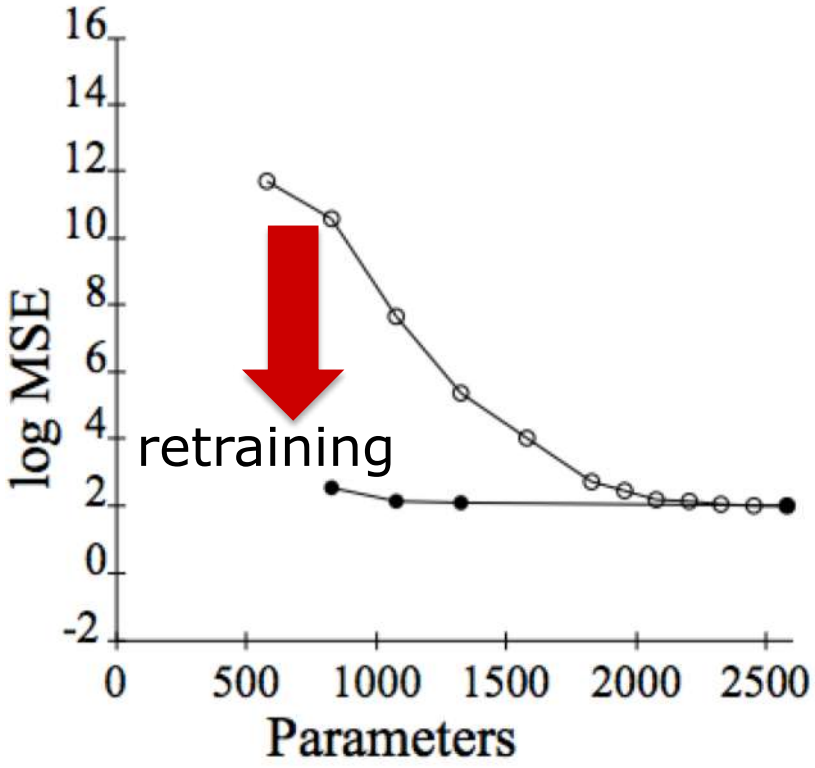


Simple RLC within 5% - 10% of theoretical entropy limit

[**Chen**, *ISSCC* 2016]

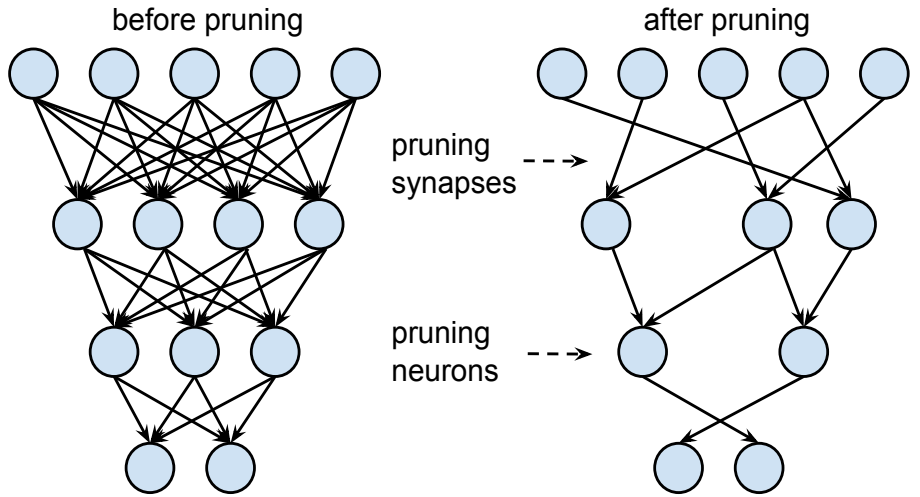
Pruning – Make Weights Sparse

Optimal Brain Damage



[**Lecun**, *NeurIPS* 1990]

Prune DNN based on ***magnitude*** of weights



Example: *AlexNet*
Weight Reduction: *CONV layers 2.7x,* ***FC layers 9.9x***
Overall Reduction: *Weights 9x, MACs 3x*

[**Han**, *NeurIPS* 2015]

Unstructured or Structured Sparsity

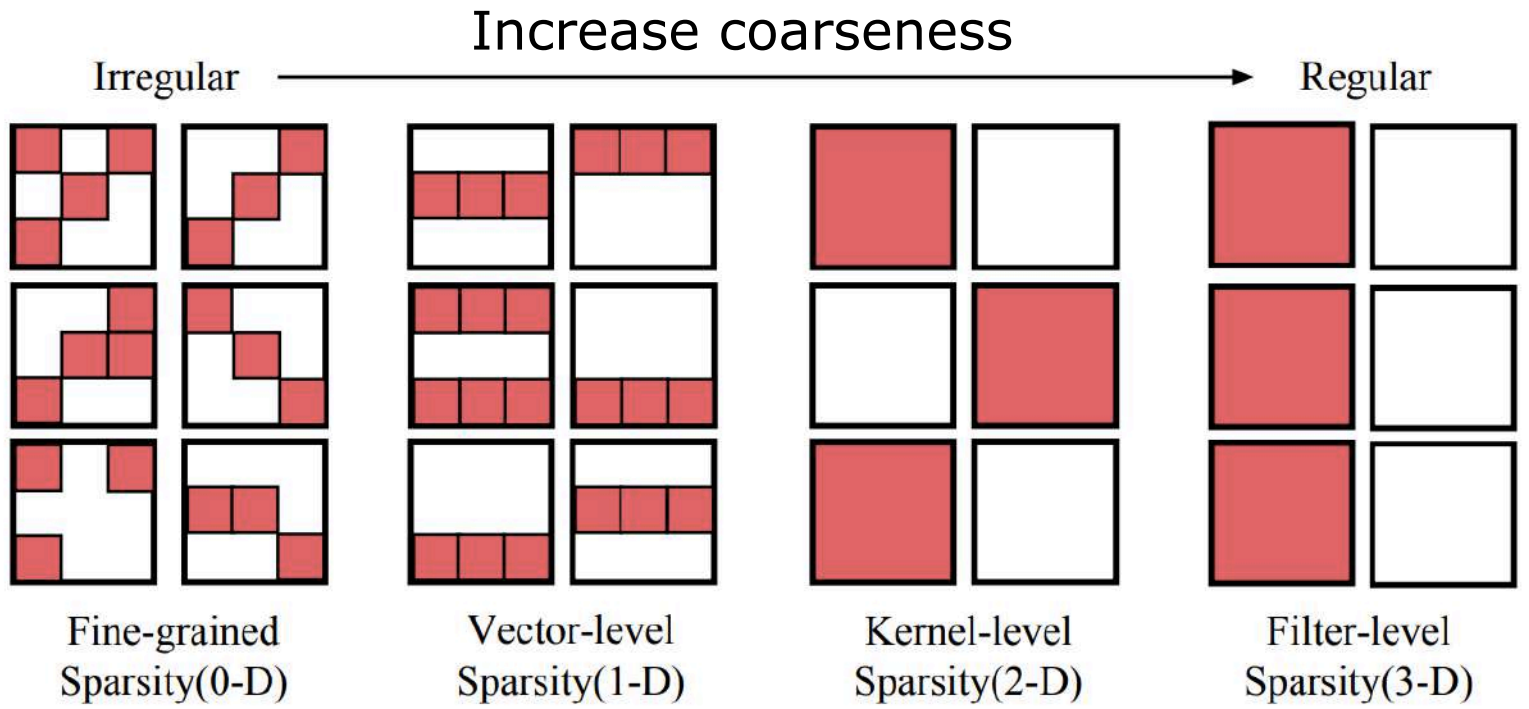


Image Source: [**Mao**, *CVPR Workshop* 2017]

Benefits:

Increase coarseness → more structure in sparsity (easier for hardware)

Less signaling overhead for location of zeros → better compression

Design Considerations for Sparsity

- **Impact on accuracy**
  - Must consider difficulty of dataset, task, and DNN model
    - e.g., AlexNet and VGG known to be over parameterized and thus easy to prune weights; does method work on efficient DNN models?

- **Does hardware cost exceed benefits?**
  - Need extra hardware to identify sparsity
    - e.g., Additional logic to identify non-zeros and store non-zero locations
  - Accounting for sparsity in both weights *and* activations is challenging
    - Need to compute *intersection* of two data streams rather than find next non-zero in one
  - Granularity impacts hardware overhead as well as accuracy
    - e.g., Fine-grained or coarse-grained (structured) sparsity
  - Compressed data will be variable length
    - Reduced flexibility in access order → random access will have significant overhead

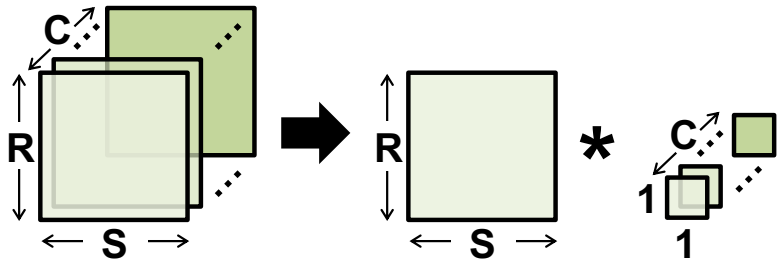
Efficient Network Architectures

Efficient DNN Models

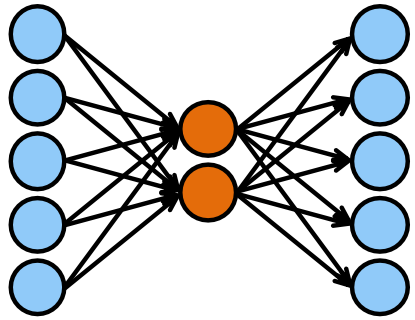
- Design efficient DNN Models
  - Tends to increase variation of layer shapes (e.g., R, S, C) that need to be supported
  - Can be handcrafted or learned using Network/Neural Architecture Search (NAS)

Filter Decomposition

Decompose large filters into smaller filters



1x1 convolutions



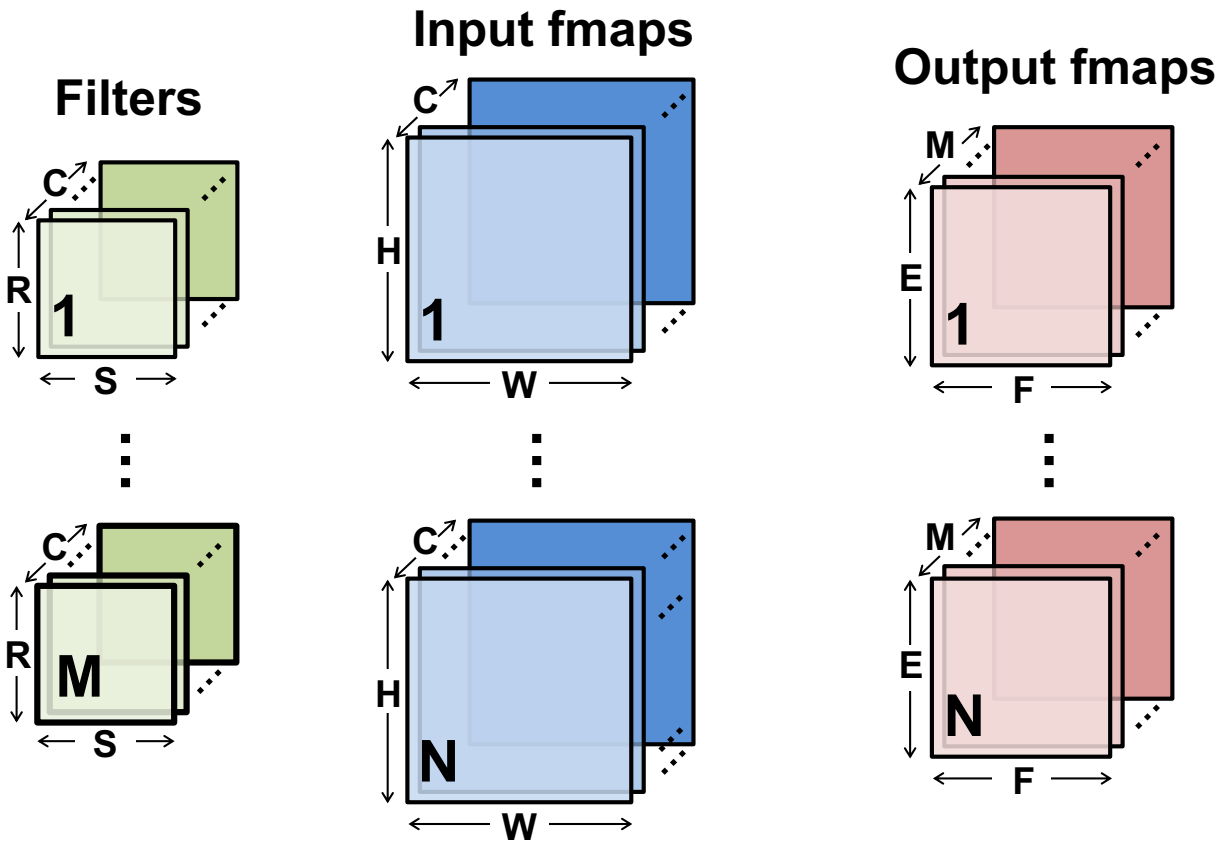
Reduce number of channels before large filter convolution

	Year	Accuracy*	# Layers	# Weights	# MACs
AlexNet	2012	80.4%	8	**61M**	**724M**
MobileNet	2017	**89.5%**	28	**4M**	**569M**

* ImageNet Classification Top-5

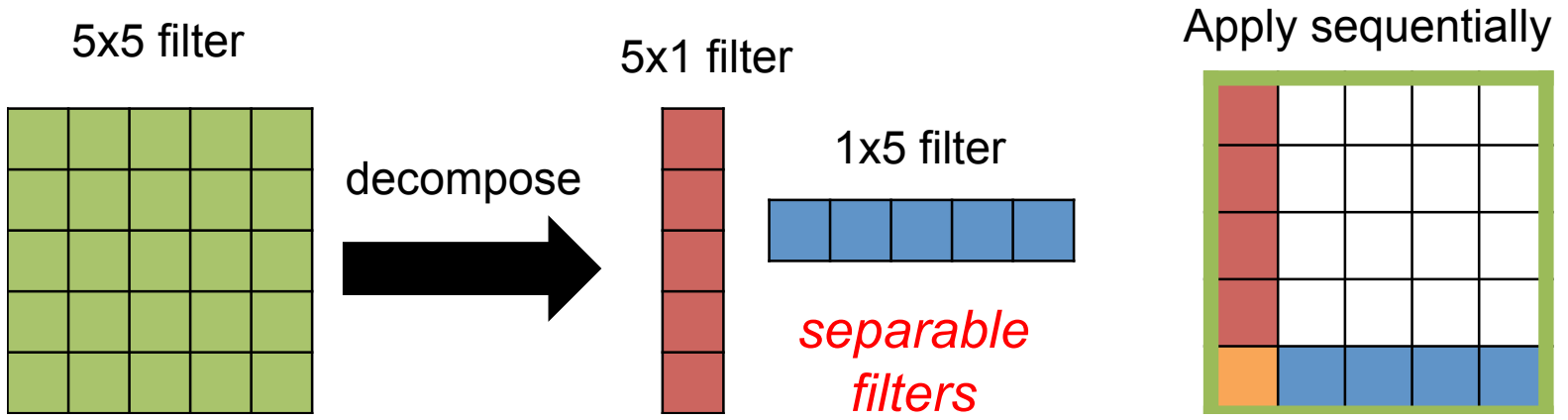
Manual Network Design

- **Reduce Spatial Size (R, S)**
  - stacked filters
- **Reduce Channels (C)**
  - 1x1 convolution, group of filters
- **Reduce Filters (M)**
  - feature map reuse across layers
- **Layer Pooling**
  - global pooling before FC layer

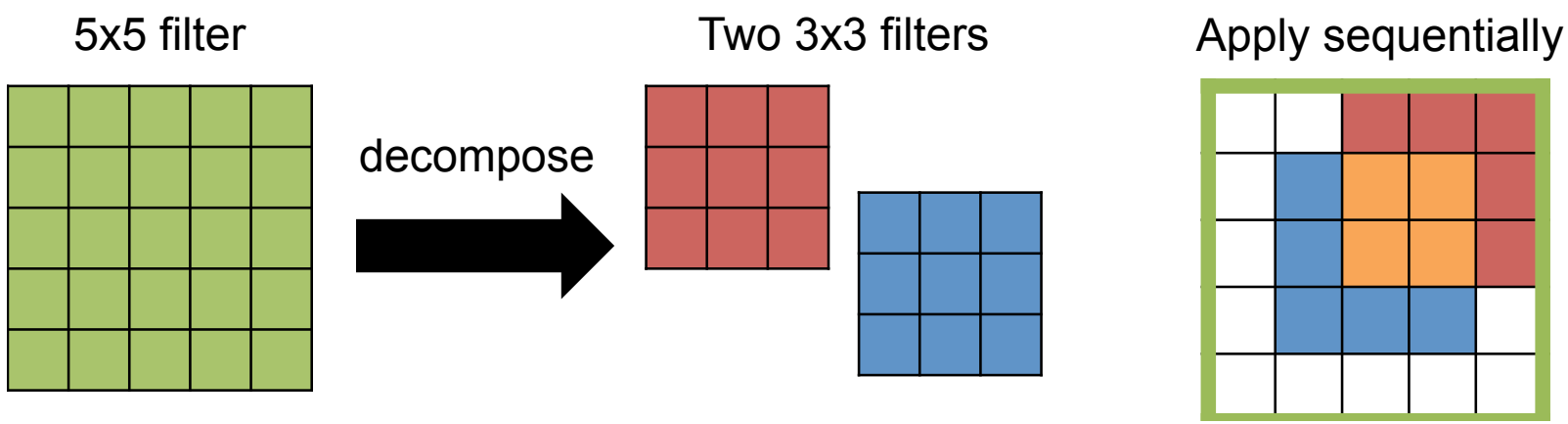


Reduce Spatial Size (R, S): Stacked Filter

GoogleNet/ Inception v3



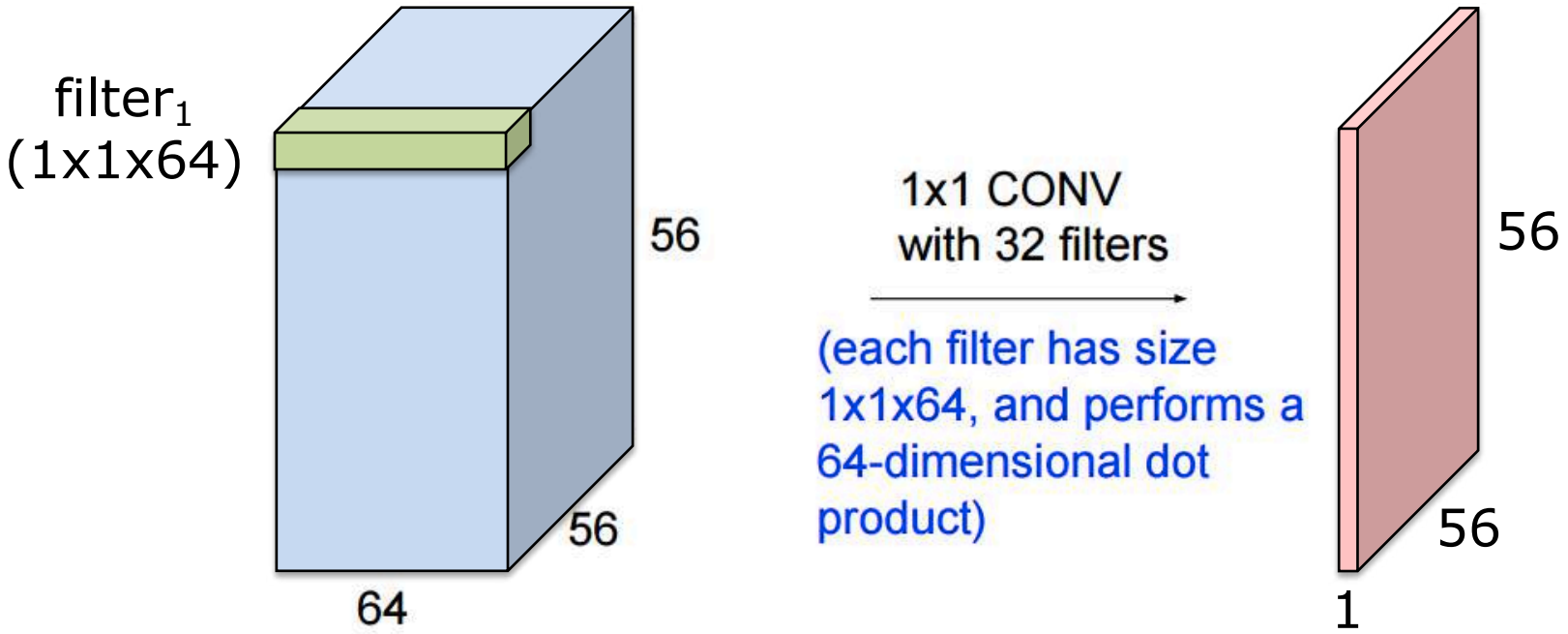
VGG



Replace a large filter with a series of smaller filters

Reduce Channels (C): 1x1 Convolution

Use **1x1 filter** to summarize cross-channel information

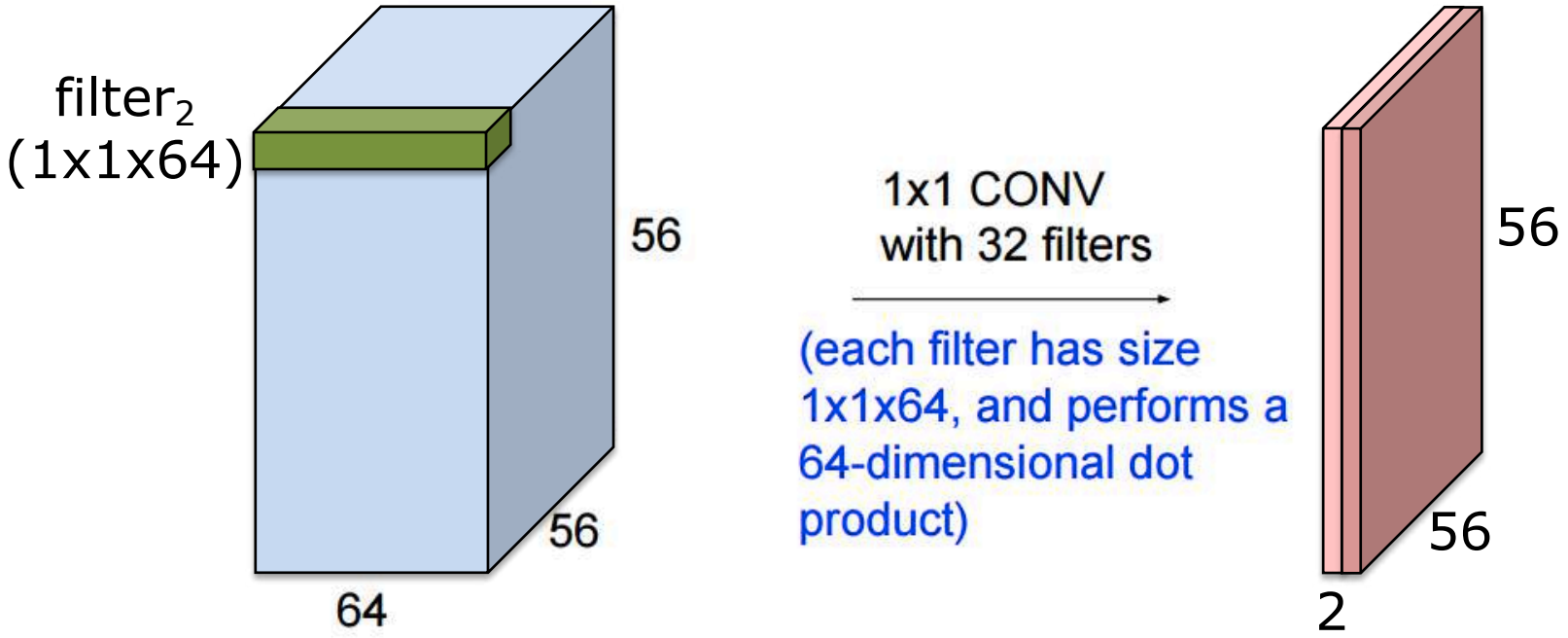


Modified image from source:
Stanford cs231n

[**Lin**, *ICLR* 2014]

Reduce Channels (C): 1x1 Convolution

Use **1x1 filter** to summarize cross-channel information

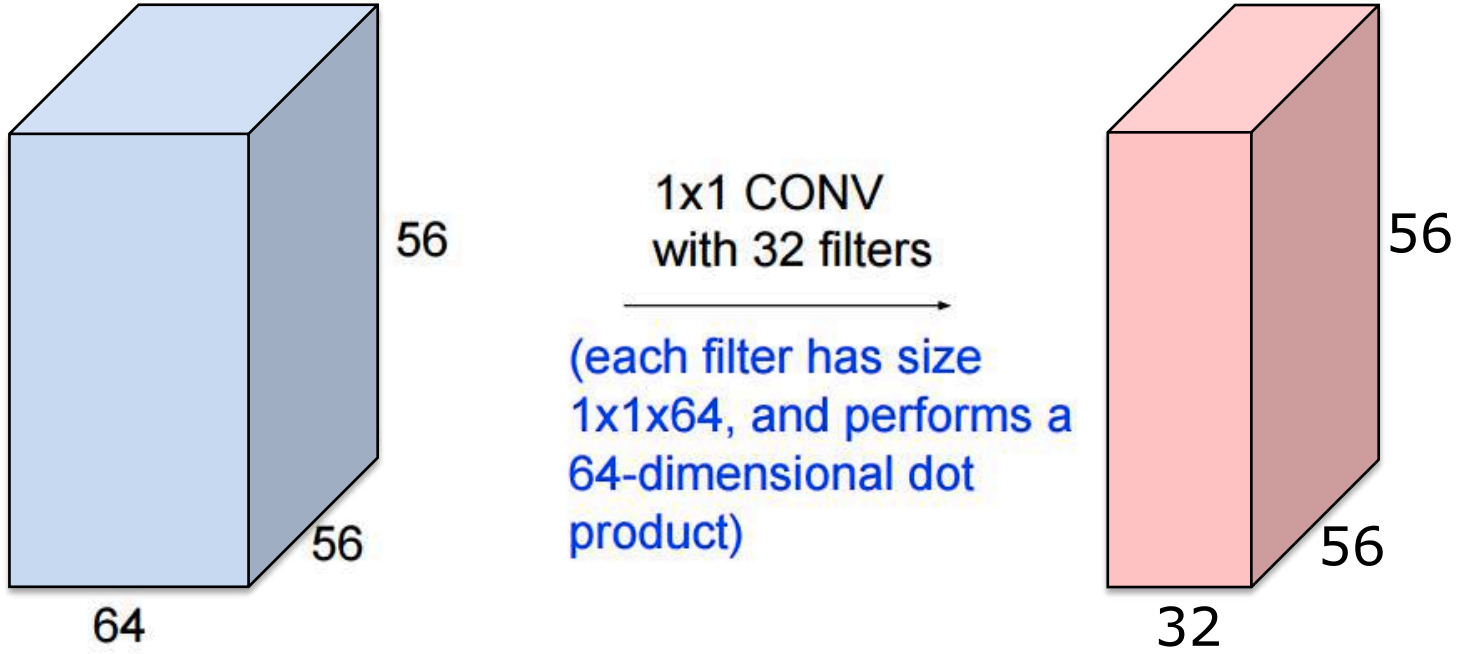


Modified image from source: Stanford cs231n

[**Lin**, *ICLR* 2014]

Reduce Channels (C): 1x1 Convolution

Use **1x1 filter** to summarize cross-channel information



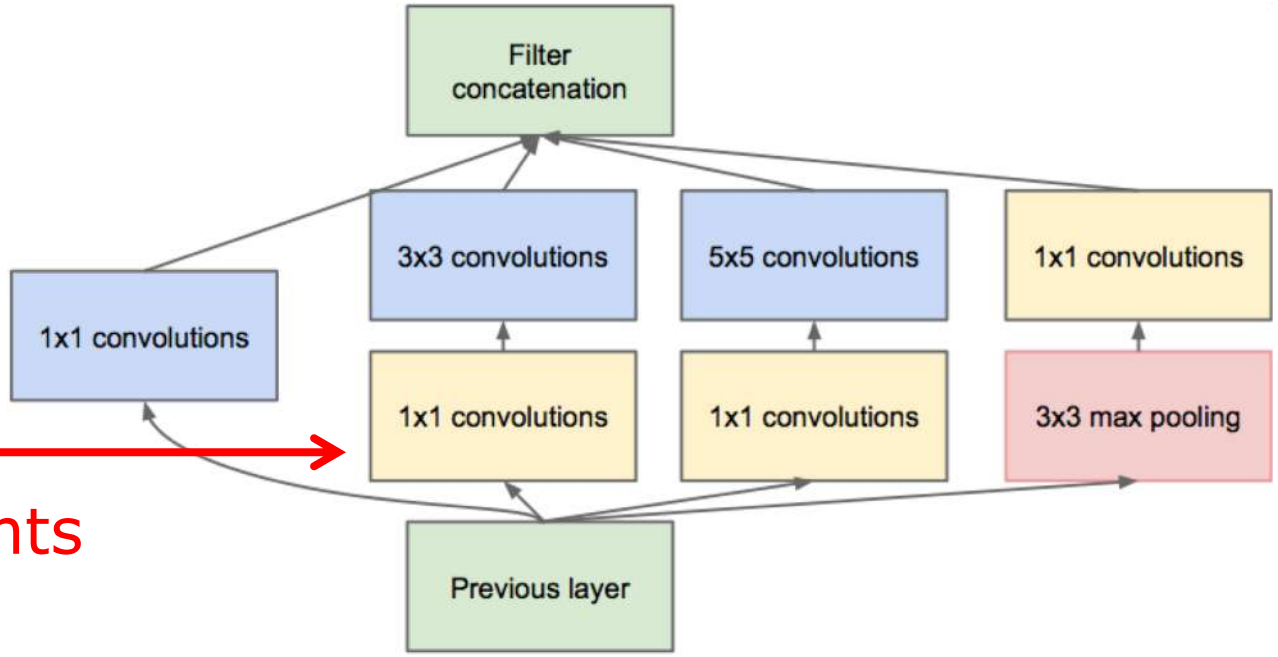
Modified image from source:
Stanford cs231n

[**Lin**, *ICLR* 2014]

GoogLeNet:1x1 Convolution

Apply 1x1 convolution before 'large' convolution filters.
Reduce weights such that **entire CNN can be trained on one GPU.**
Number of multiplications reduced from 854M → 358M

Inception Module



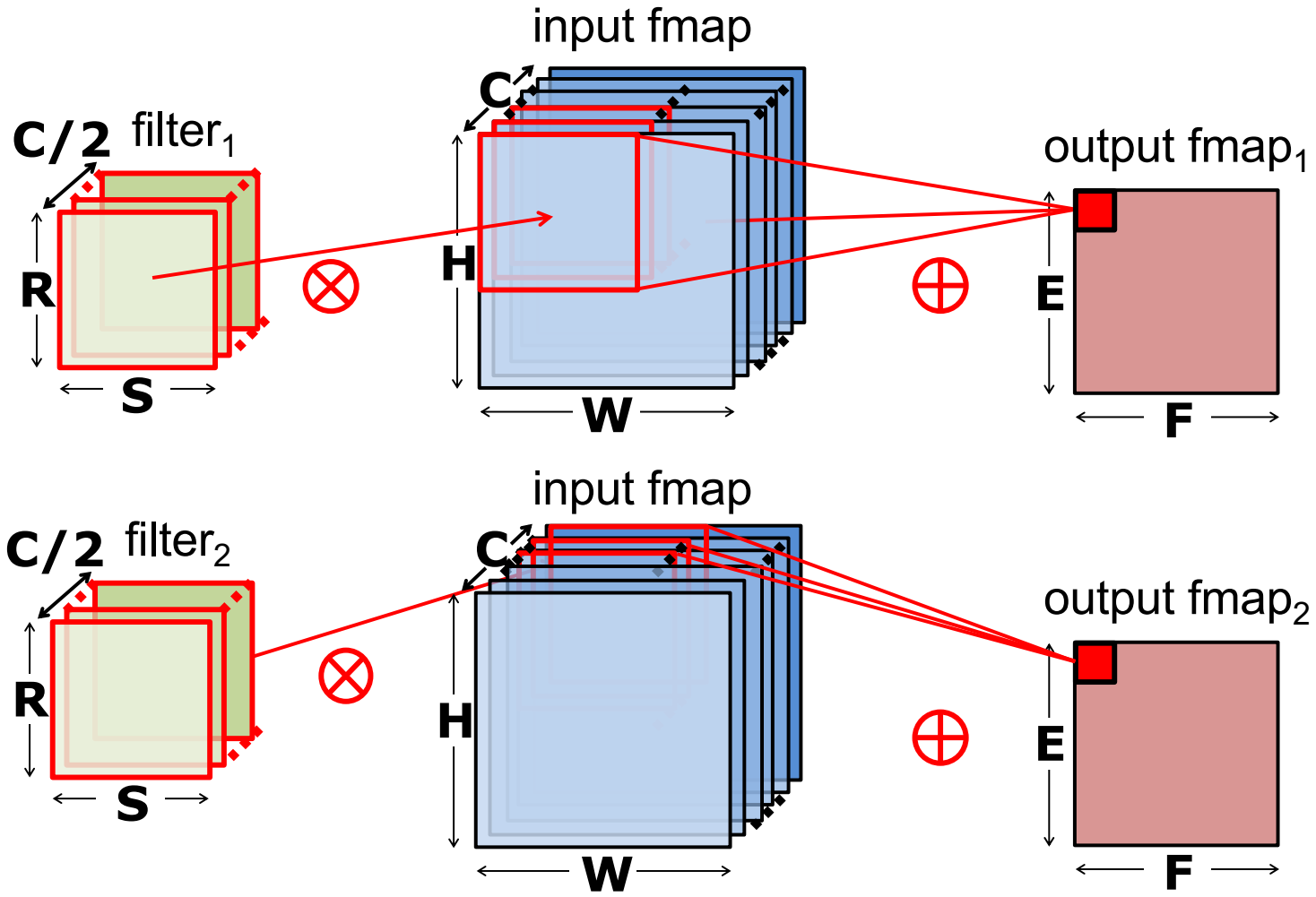
1x1 convolutions to reduce number of weights and multiplications

[**Szegedy**, CVPR 2015]

Reduce Channels (C): Group of Filters

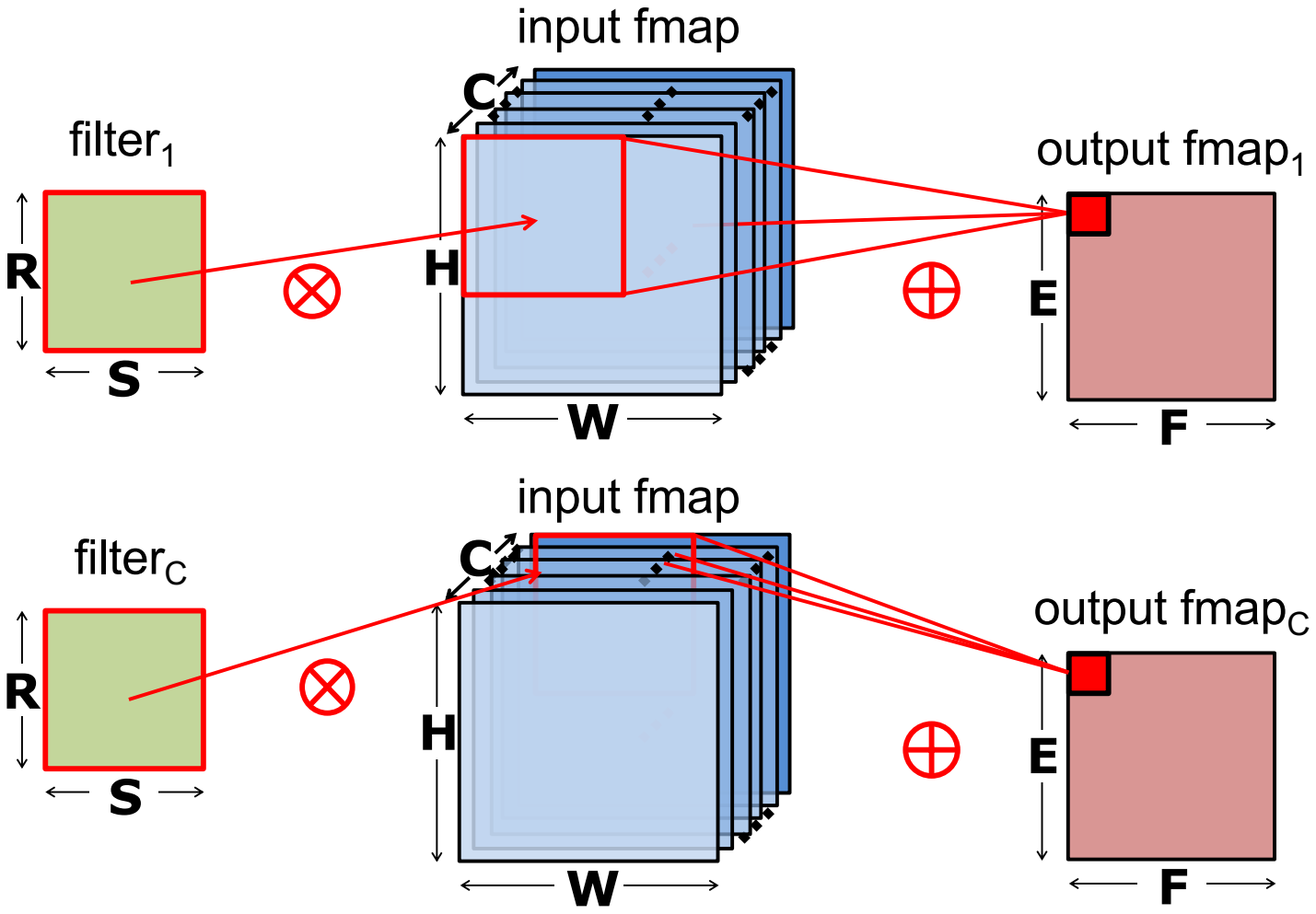
Split filters and channels of feature map into different groups

e.g., For two groups, each filter requires **2x fewer weights and multiplications**



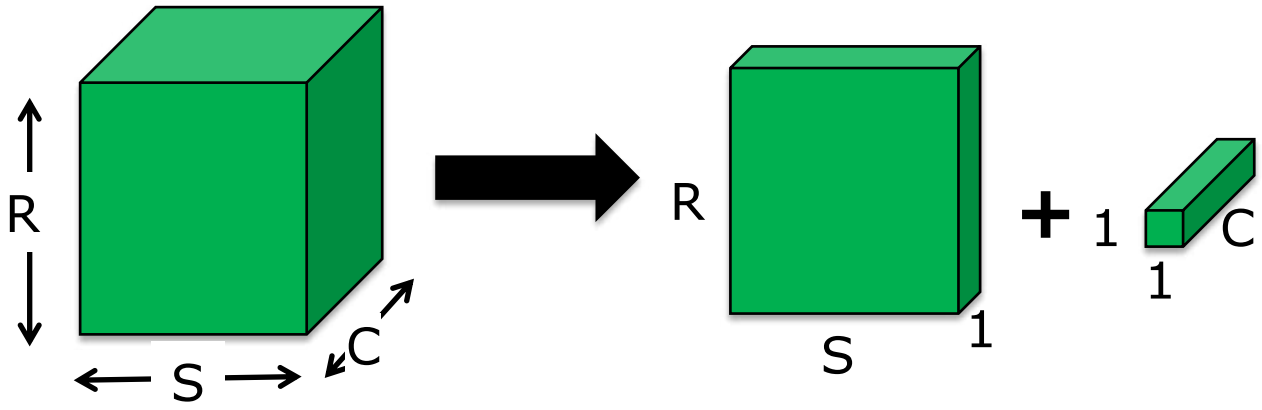
Reduce Channels (C): Group of Filters

The extreme case is **depthwise convolution** – each group contains only one channel

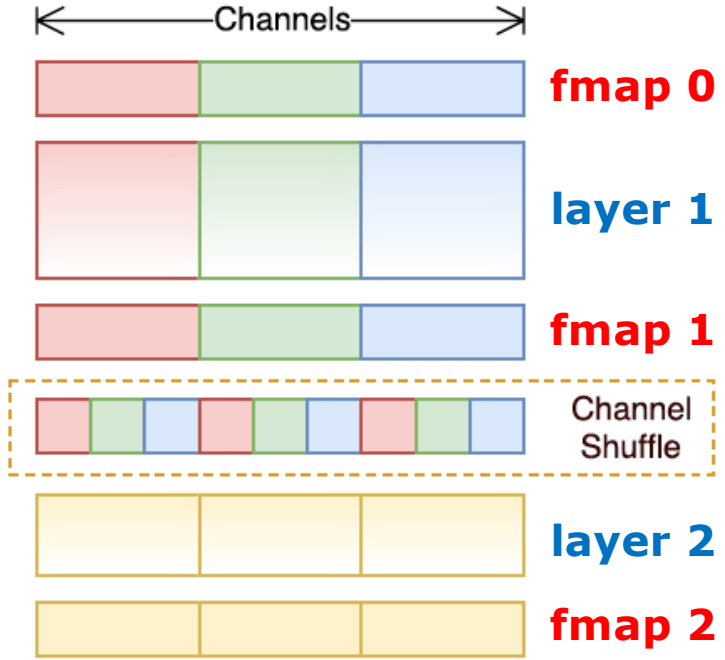


Reduce Channels (C): Group of Filters

Two ways of mixing information across groups

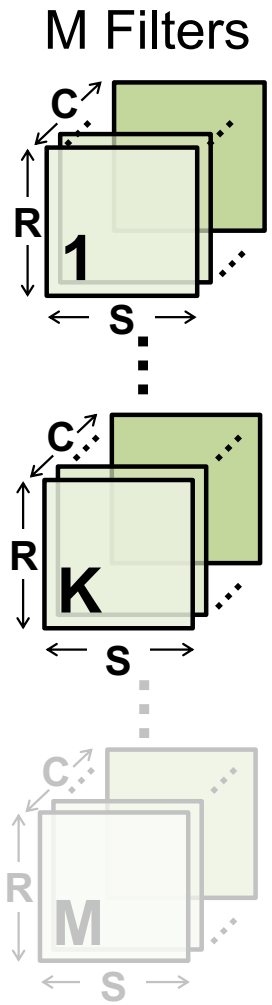


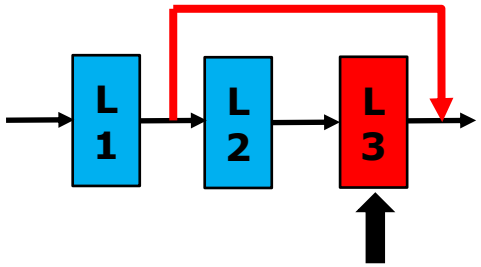
Pointwise (1x1) Convolution
(Mix in one step)
MobileNet [**Howard**, *arXiv* 2017]



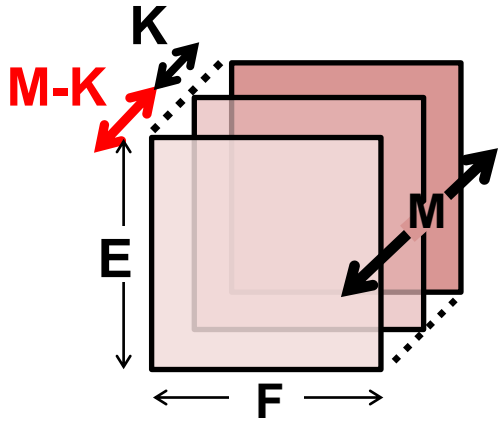
Shuffle Operation
(Mix in multiple steps)
ShuffleNet [**Zhang**, *CVPR* 2018]

Reduce Filters (M): Feature Map Reuse





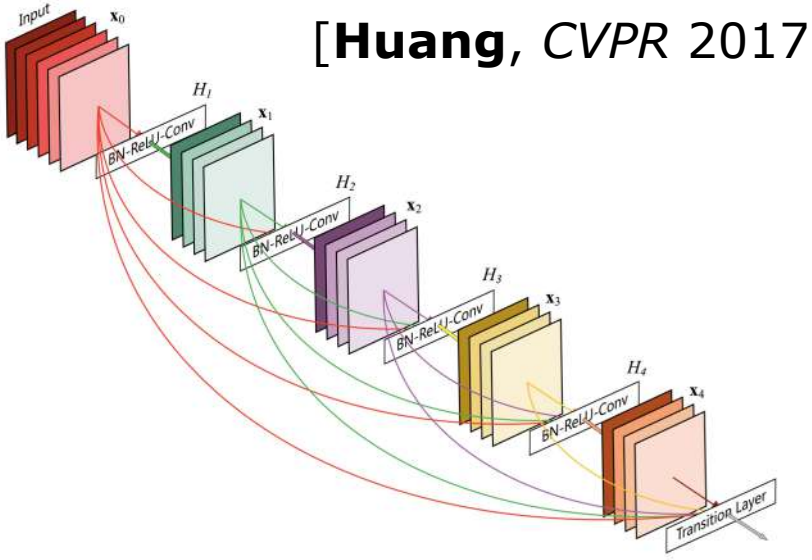
Output fmap with M channels



Reuse **(M-K)** channels in feature maps from previously processed layers

DenseNet reuses feature map from multiple layers

[**Huang**, *CVPR* 2017]

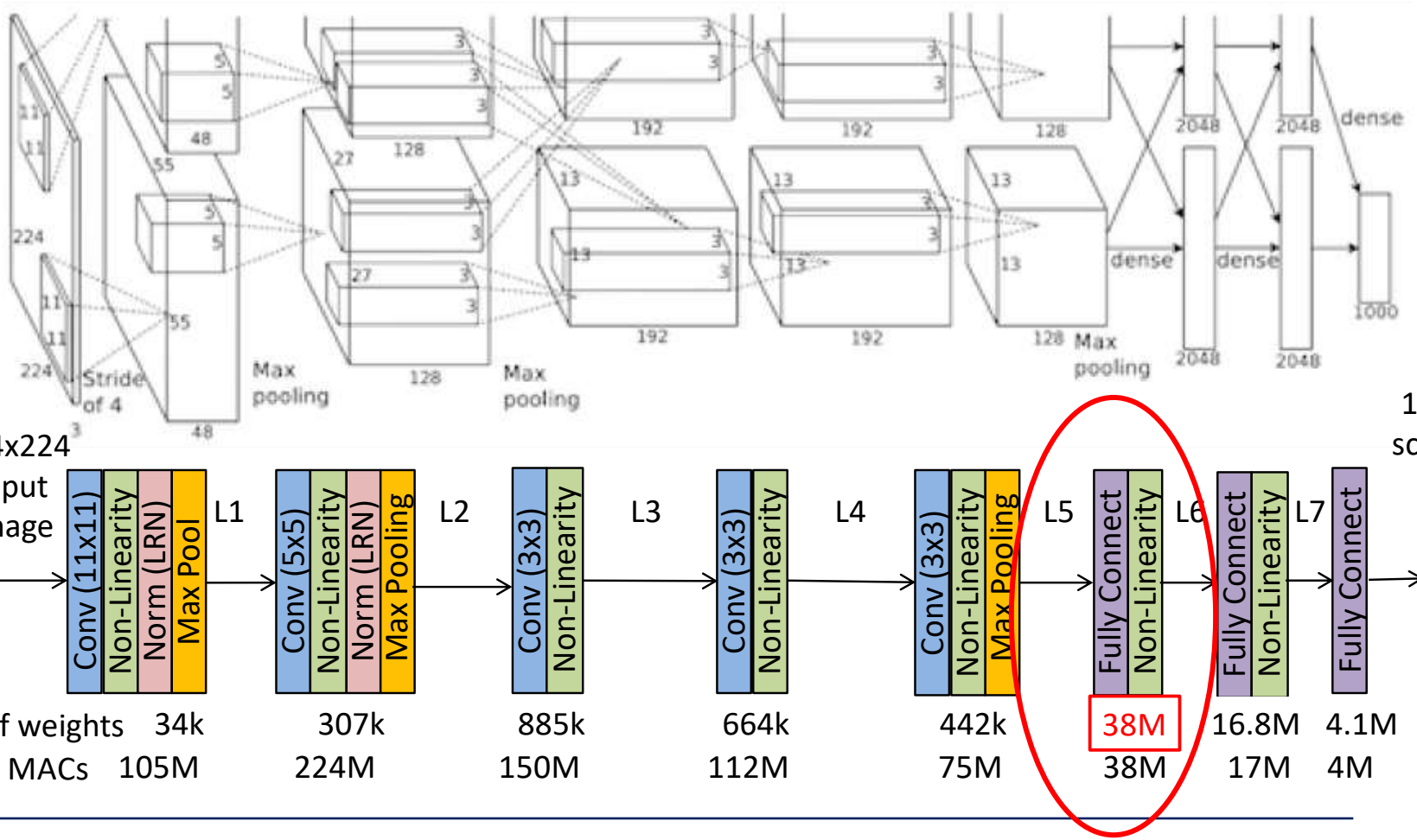


Layer Pooling: Simplify FC Layers

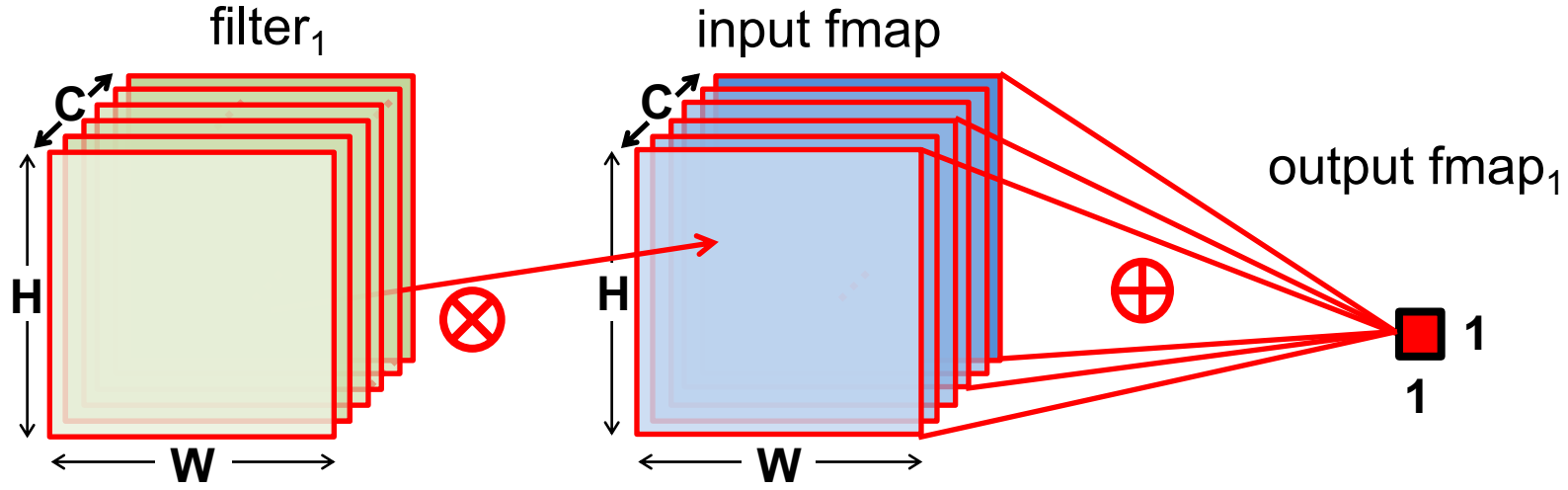
[**Krizhevsky**, NeurIPS 2012]

First FC layer accounts for a significant portion of weights

38M of 61M for AlexNet



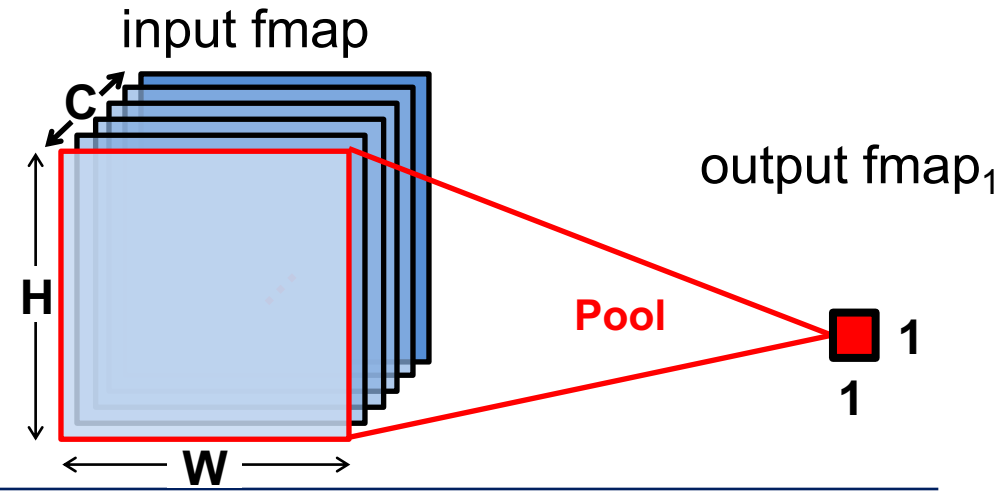
Layer Pooling: Simplify FC Layers



Global Pooling reduces the size of the input to the first FC layer, which reduces its size

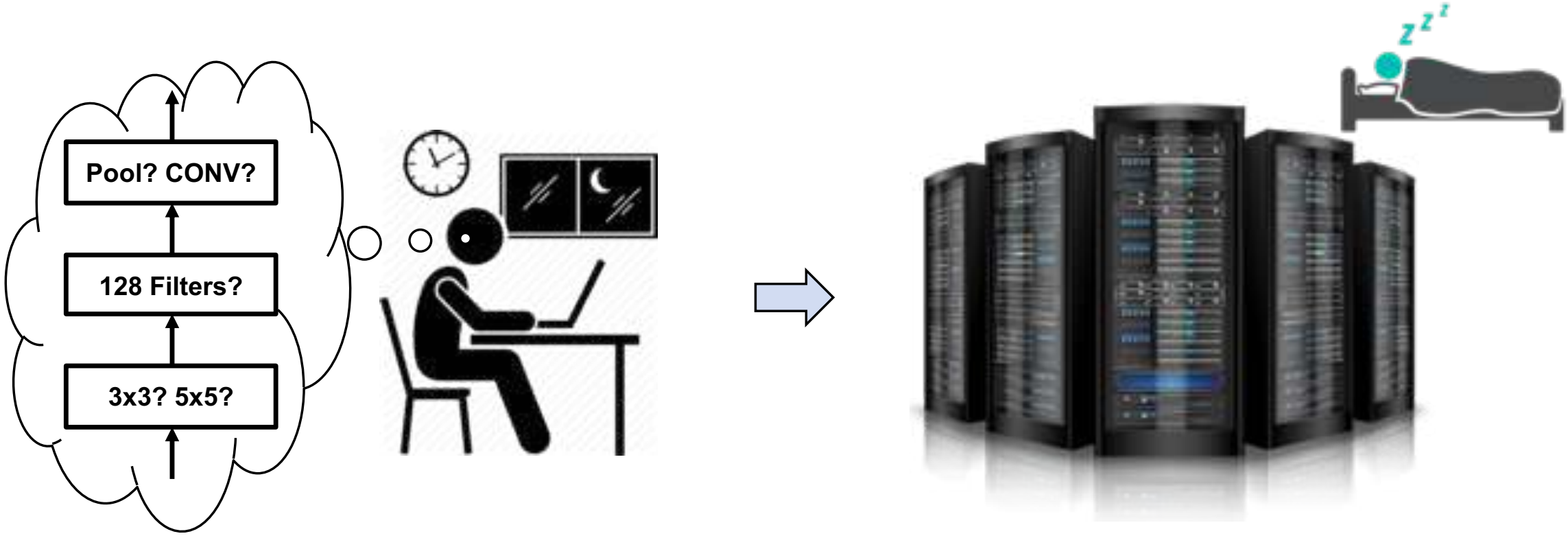
(e.g., HxWxCx1000 → 1x1xCx1000)

[**Lin**, *ICLR* 2014]



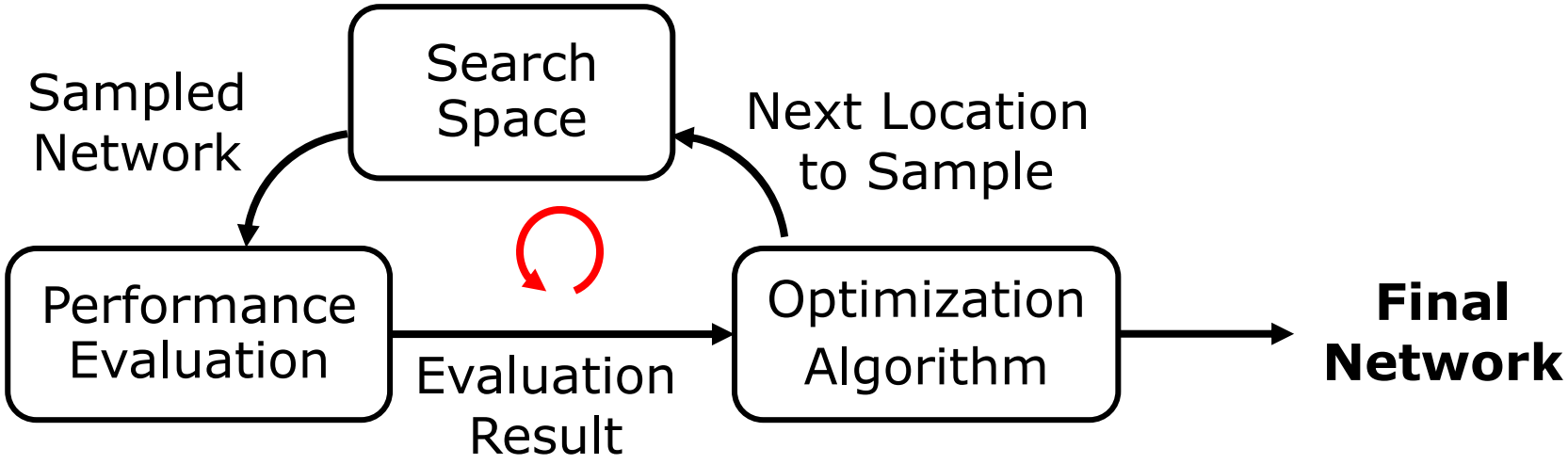
Network/Neural Architecture Search (NAS)

Rather than handcrafting the architecture, automatically search for it



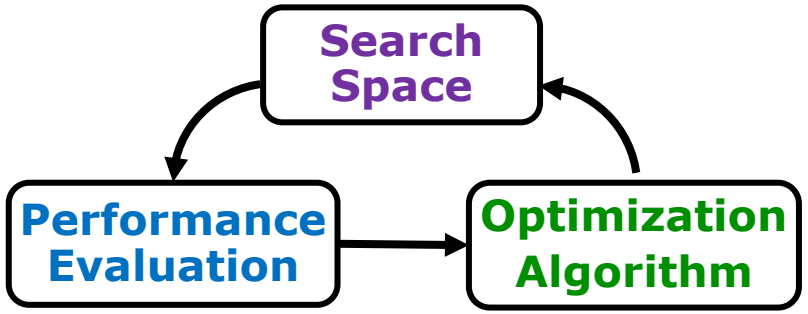
Network/Neural Architecture Search (NAS)

- **Three main components:**
  - Search Space (what is the set of all samples)
  - Optimization Algorithm (where to sample)
  - Performance Evaluation (how to evaluate samples)



Key Metrics: Achievable DNN **accuracy** and required **search time**

Evaluate NAS Search Time



$$time_{nas} = num_{samples} \times time_{per\_sample}$$

$$time_{nas} \propto (num_{nas\_tuning} \times num_{nas\_steps} \times size_{search\_space}) \times (time_{train} + time_{eval})$$

(2) Improve the optimization algorithm

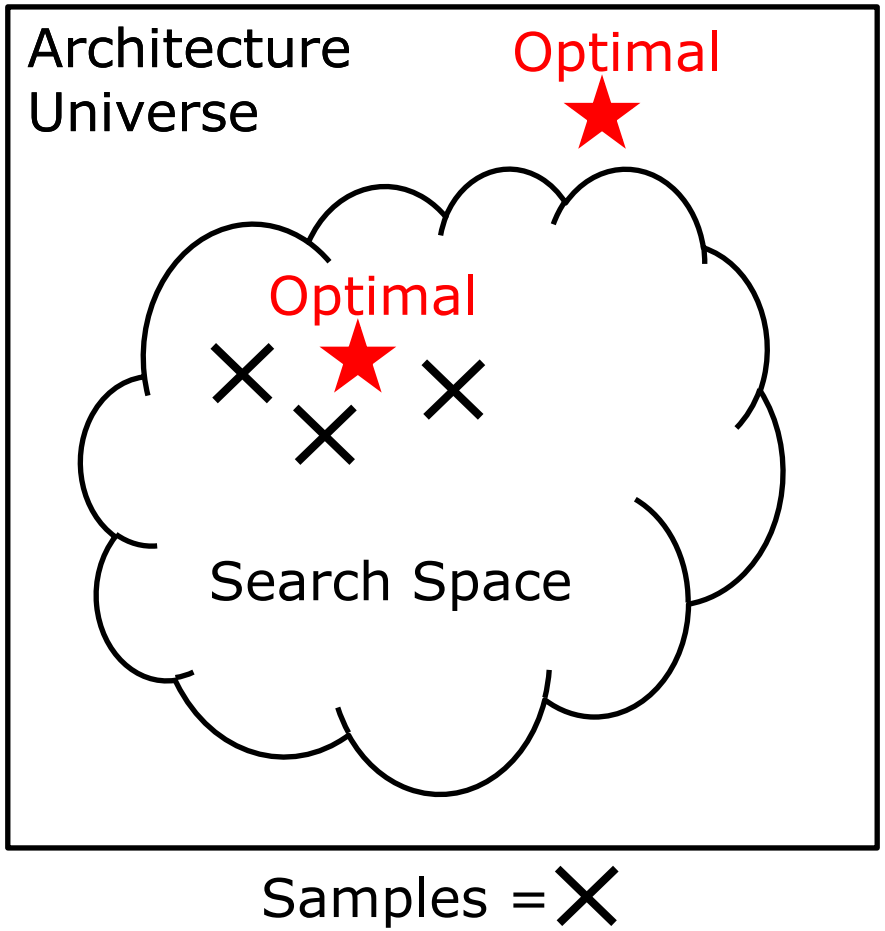
(1) Shrink the search space

(3) Simplify the performance evaluation

Goal: Improve the efficiency of NAS in the three main components

(1) Shrink the Search Space

- Trade the breadth of architectures for search speed
- May limit the performance that can be achieved
- Use domain knowledge from manual network design to help guide the reduction of the search space



(1) Shrink the Search Space

- Search space = **layer operations** + connections between layers

Common layer operations

- Identity
- 1x3 then 3x1 convolution
- 1x7 then 7x1 convolution
- 3x3 dilated convolution
- 1x1 convolution
- 3x3 convolution
- 3x3 separable convolution
- 5x5 separable convolution
- 3x3 average pooling
- 3x3 max pooling
- 5x5 max pooling
- 7x7 max pooling

[**Zoph**, *CVPR* 2018]

(1) Shrink the Search Space

- Search space = layer operations + **connections between layers**

Smaller Search Space

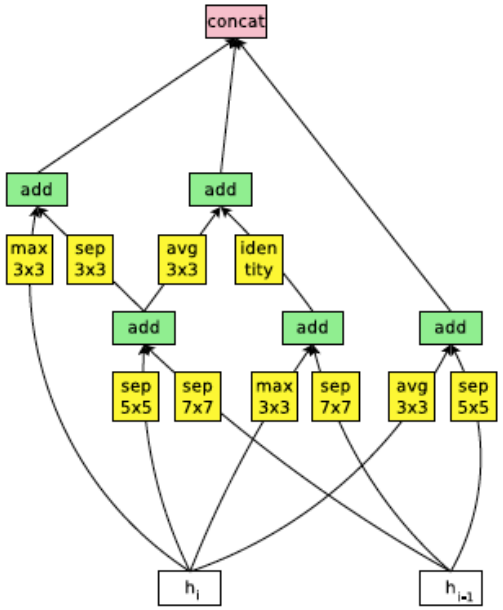
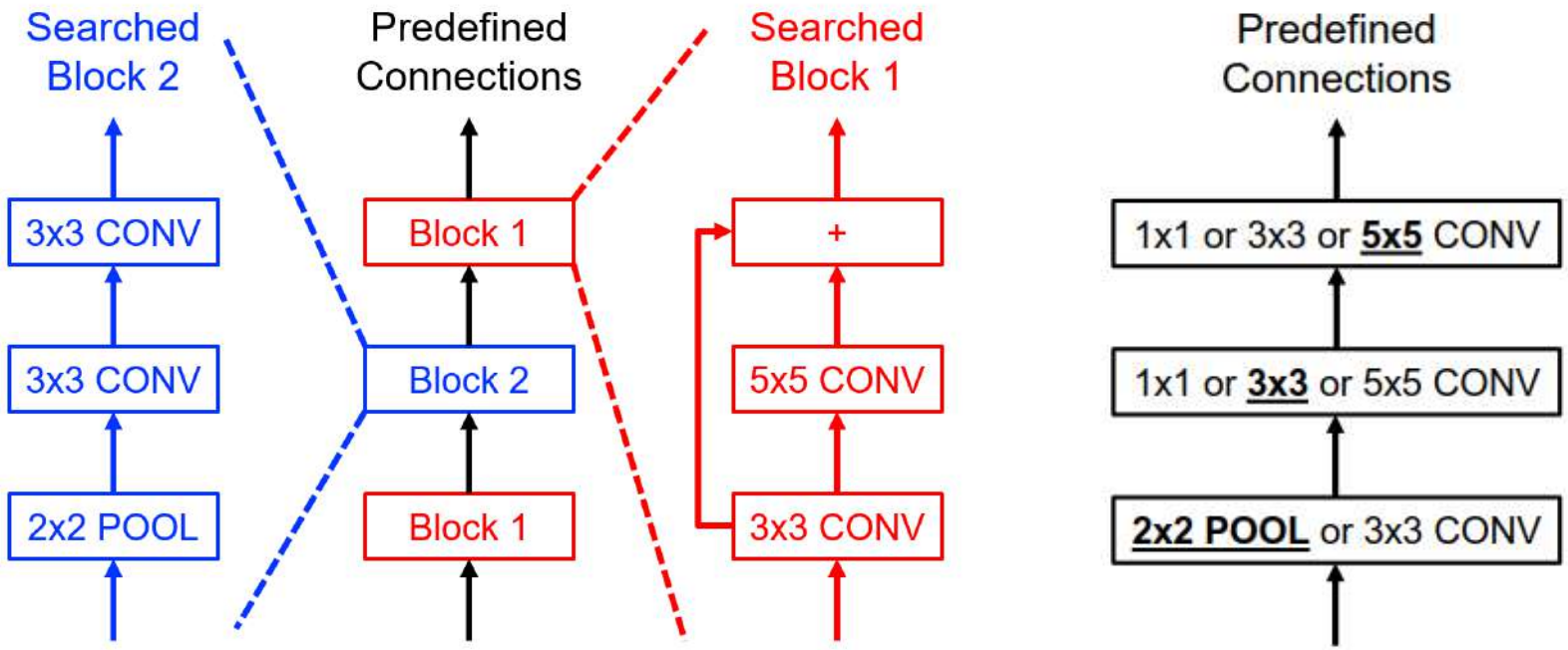


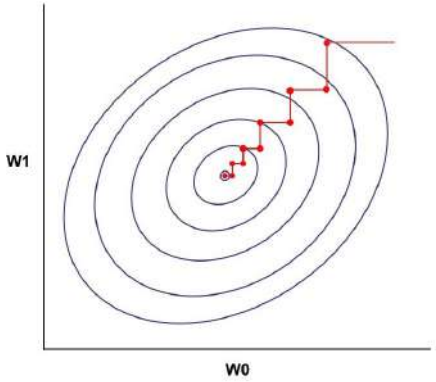
Image Source: [**Zoph**, CVPR 2018]



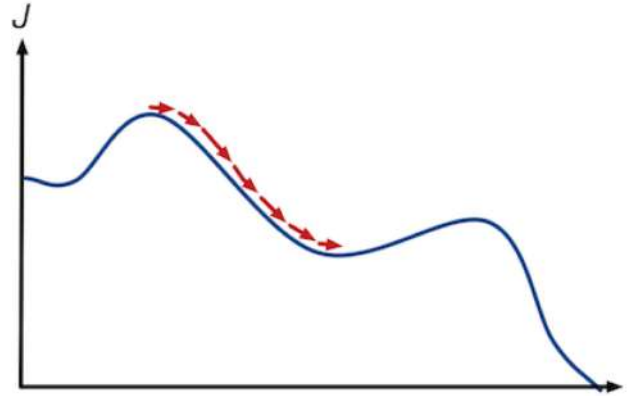
(2) Improve Optimization Algorithm

Random

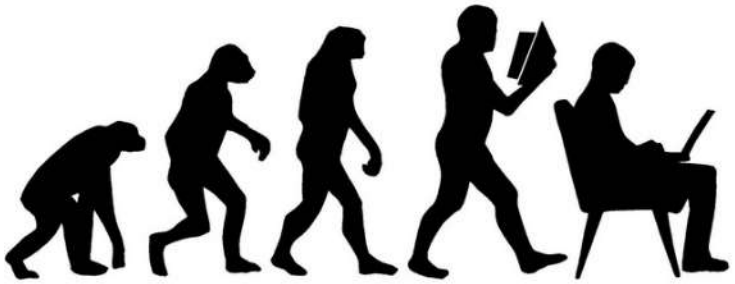
Coordinate Descent



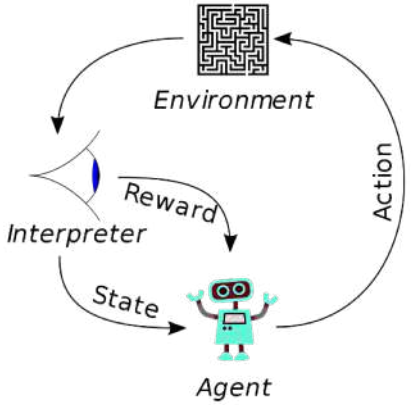
Gradient Descent



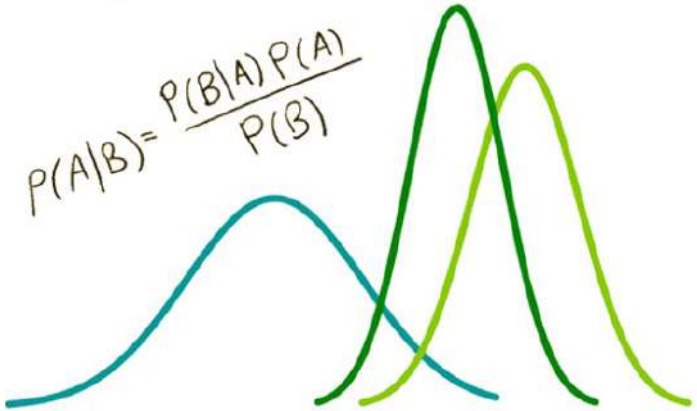
Evolutionary



Reinforcement Learning



Bayesian

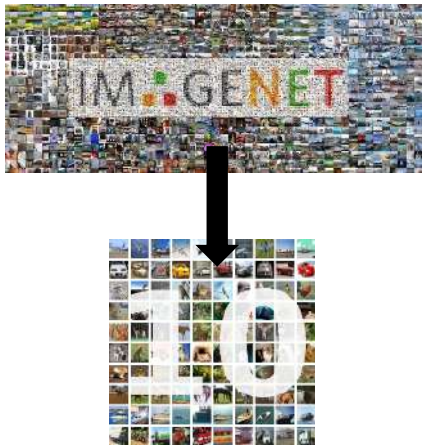


(3) Simplify the Performance Evaluation

- NAS needs only the **rank** of the performance values
- **Method 1:** approximate accuracy

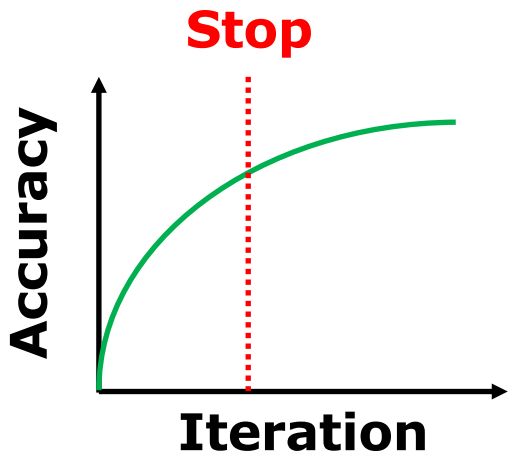
Proxy Task

E.g., Smaller resolution, simpler tasks



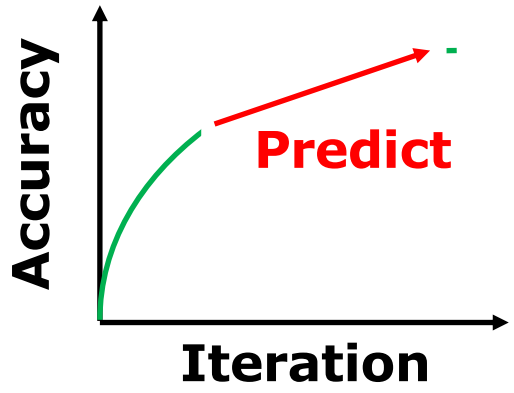
Early Termination

Stop training earlier



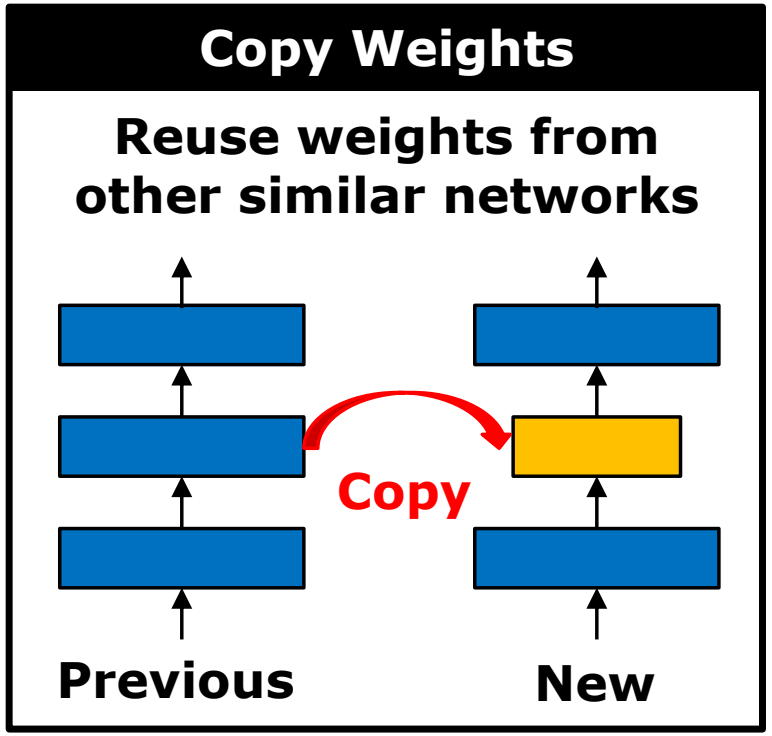
Accuracy Prediction

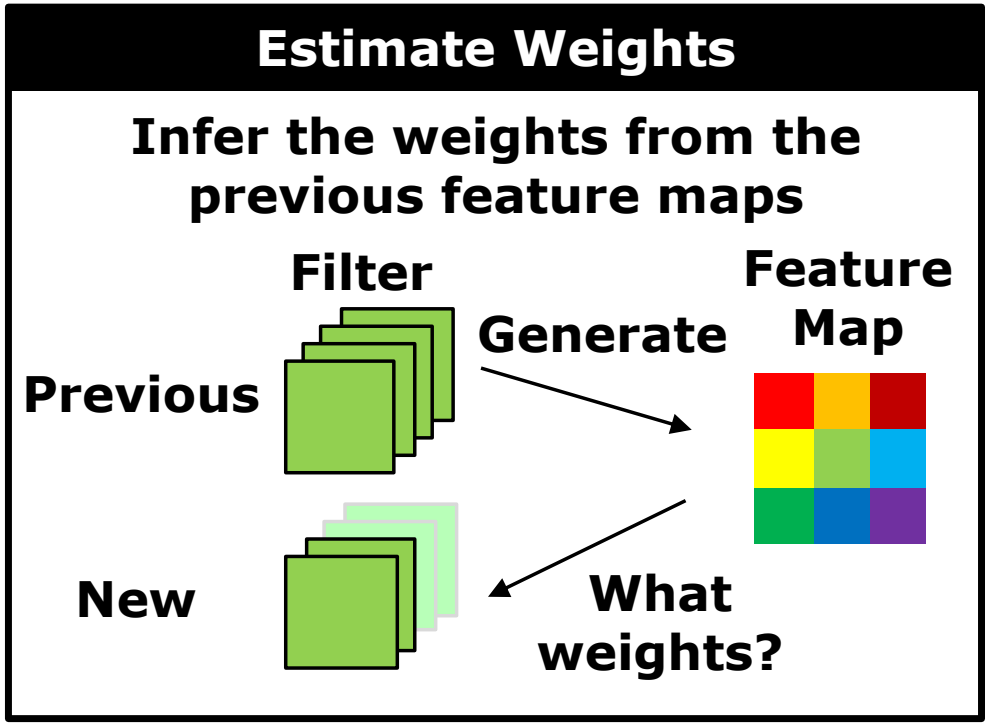
Extrapolate accuracy



(3) Simplify the Performance Evaluation

- NAS needs only the **rank** of the performance values
- **Method 2:** approximate weights



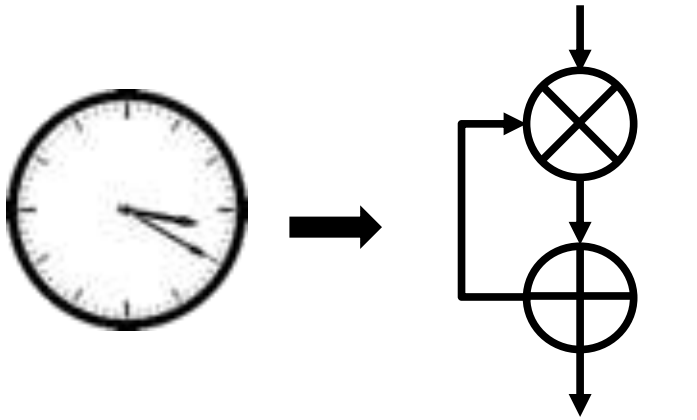


(3) Simplify the Performance Evaluation

- NAS needs only the **rank** of the performance values
- **Method 3:** approximate metrics (e.g., latency, energy)

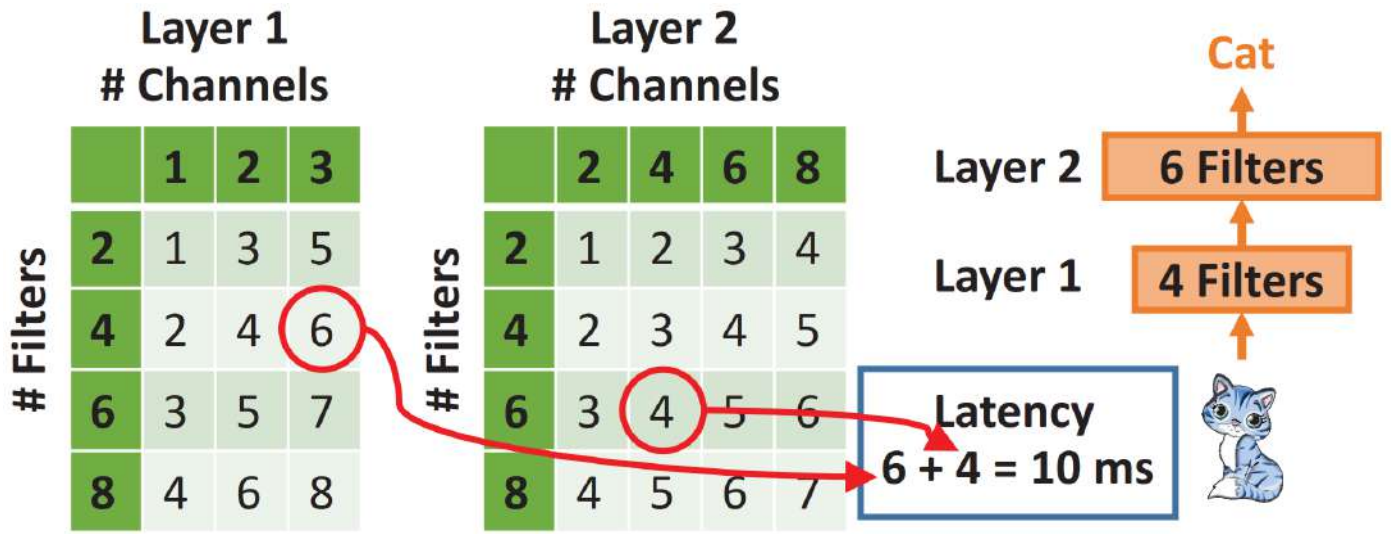
Proxy Metric

Use an easy-to-compute metric to approximate target



Look-Up Table

Use table lookup



Design Considerations for NAS

- **The components may not be chosen individually**
  - Some optimization algorithms limit the search space
  - Type of performance metric may limit the selection of the optimization algorithms

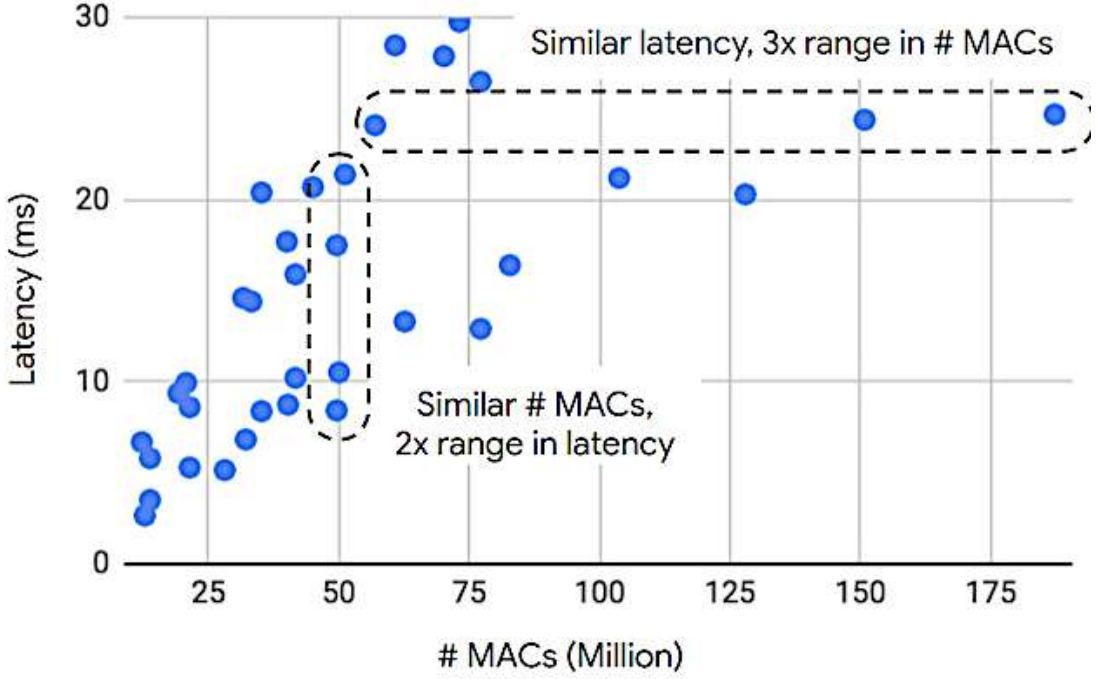
- **Commonly overlooked properties**
  - The complexity of implementation
  - The ease of tuning hyperparameters of the optimization
  - The probability of convergence to a good architecture

Hardware In the Loop

How to Evaluate Complexity of DNN Model?

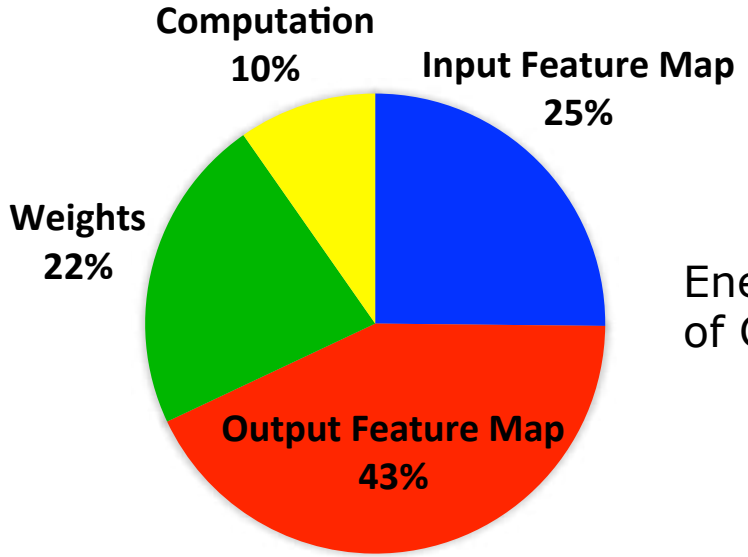
Number of MACs and weights are not good proxies for latency and energy

# of operations (MACs) does not approximate latency well



Source: Google (https://ai.googleblog.com/2018/04/introducing-cvpr-2018-on-device-visual.html)

# of weights ***alone*** is not a good metric for energy (**All data types** should be considered)



Energy breakdown of GoogLeNet

https://energyestimation.mit.edu/

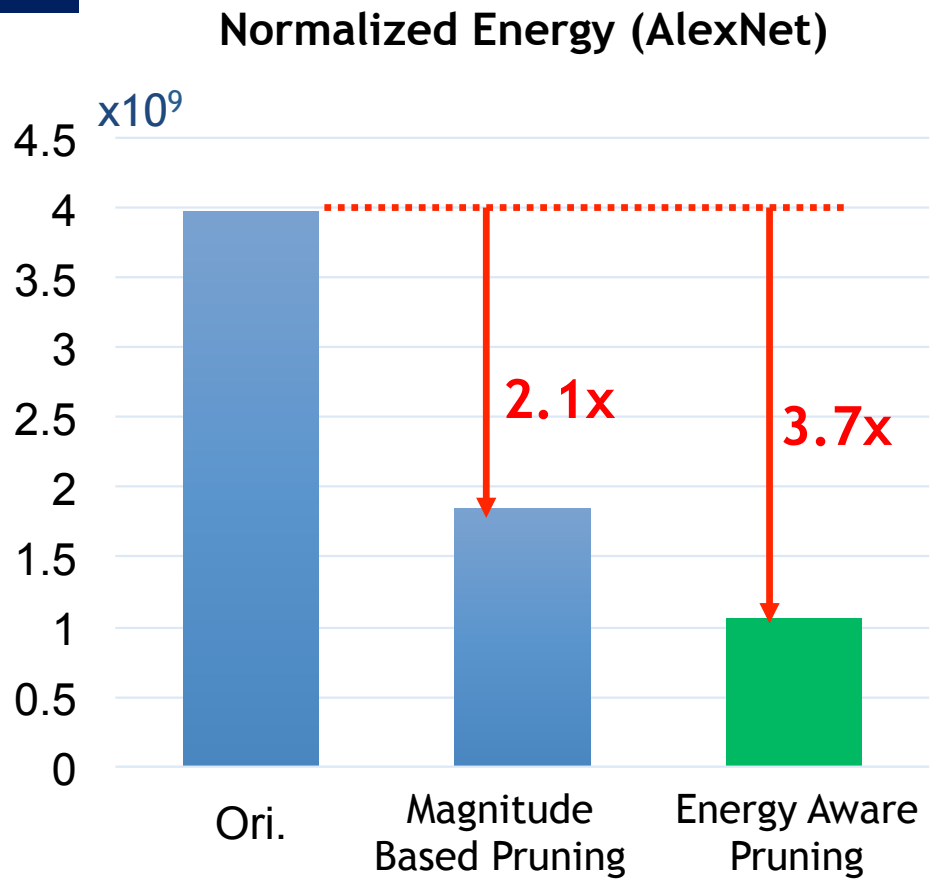
[**Yang**, *CVPR* 2017]

Energy-Aware Pruning

Directly target energy and incorporate it into the optimization of DNNs to provide greater energy savings

- Sort layers based on energy and prune layers that consume the most energy first
- Energy-aware pruning reduces AlexNet energy by **3.7x** and outperforms the previous work that uses magnitude-based pruning by **1.7x**

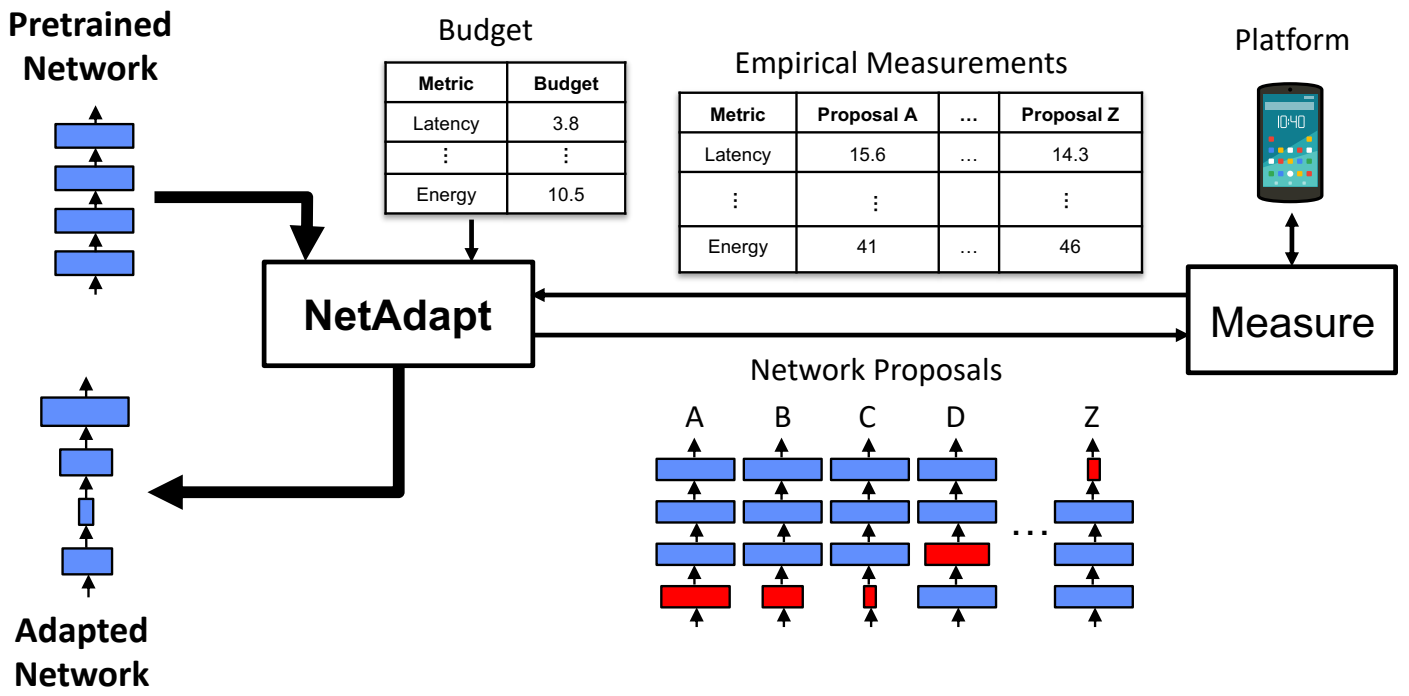
[**Yang**, *CVPR* 2017]



Pruned models available at
http://eyeriss.mit.edu/energy.html

NetAdapt: Platform-Aware DNN Adaptation

- **Automatically adapt DNN** to a mobile platform to reach a target latency or energy budget
- Use **empirical measurements** to guide optimization (avoid modeling of tool chain or platform architecture)
- Requires **very few hyperparameters** to tune



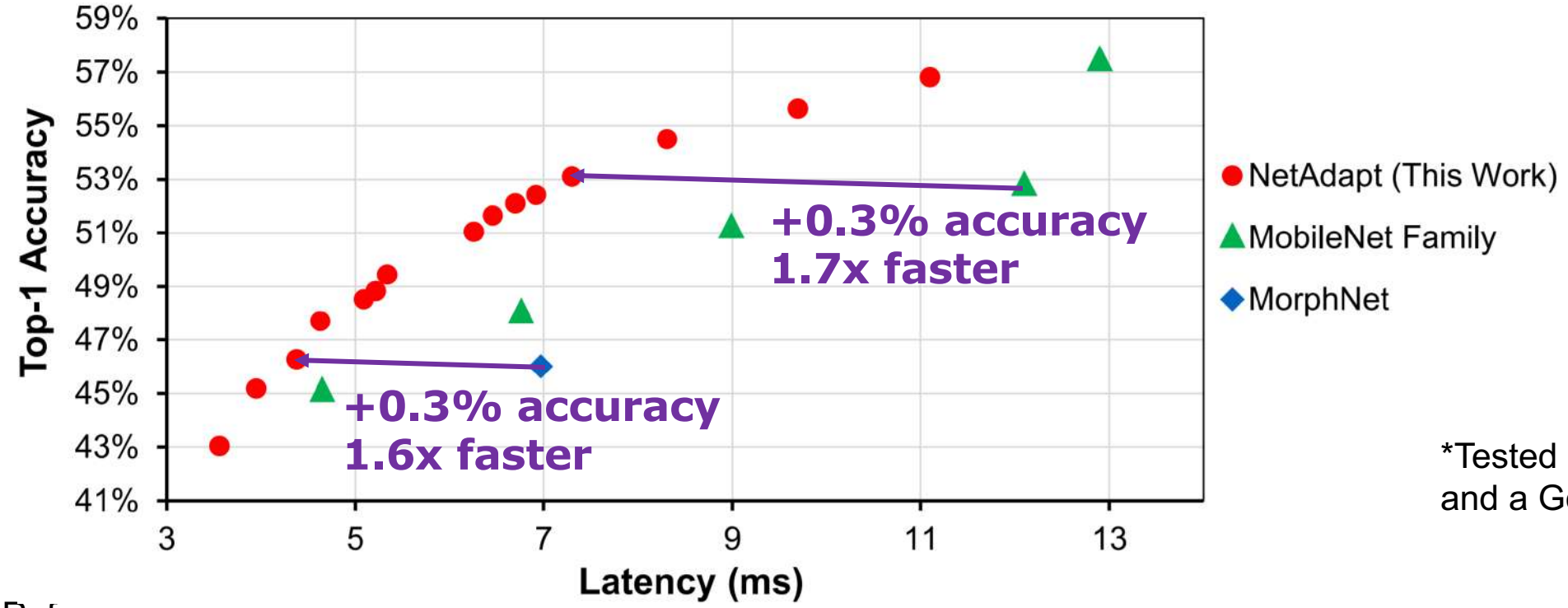
Code available at http://netadapt.mit.edu

[**Yang**, *ECCV* 2018]

In collaboration with Google's Mobile Vision Team

Improved Latency vs. Accuracy Tradeoff

- NetAdapt boosts the measured inference speed of MobileNet by up to 1.7x with higher accuracy



*Tested on the ImageNet dataset and a Google Pixel 1 CPU

Reference:
MobileNet: Howard et al, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv 2017
MorphNet: Gordon et al., "Morphnet: Fast & simple resource-constrained structure learning of deep networks," CVPR 2018

Design Considerations for Co-Design

- **Impact on accuracy**
  - Consider quality of baseline (initial) DNN model, difficulty of task and dataset
  - Sweep curve of accuracy versus latency/energy to see the full tradeoff

- **Does hardware cost exceed benefits?**
  - Need extra hardware to support variable precision and shapes or to identify sparsity
  - Granularity impacts hardware overhead as well as accuracy

- **Evaluation**
  - Avoid only evaluating impact based on number of weights or MACs as they may not be sufficient for evaluating energy consumption and latency

Design Considerations for Co-Design

- **Time required to perform co-design**
  - e.g., Difficulty of tuning affected by
    - Number of hyperparameters
    - Uncertainty in relationship between hyperparameters and impact on performance

- **Other aspects that affect accuracy, latency or energy**
  - Type of data augmentation and preprocessing
  - Optimization algorithm, hyperparameters, learning rate schedule, batch size
  - Training and finetuning time
  - Deep learning libraries and quality of the code

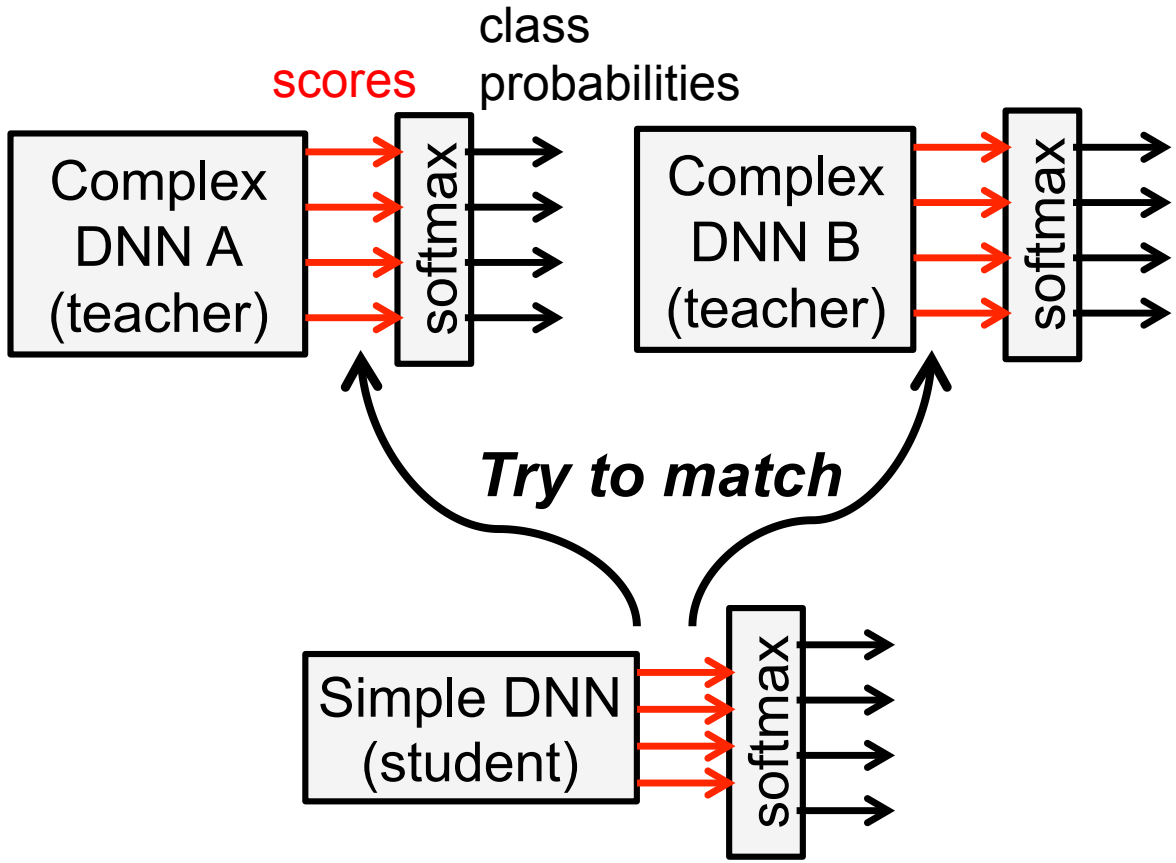
- **How does the approach perform on different platforms?**
  - Is the approach a general method, or applicable on specific hardware?

Training Approaches for Co-Design

1. Train from scratch

2. Use pretrained large DNN model
   a) **Initialize weights** for an efficient DNN model
   b) **Knowledge distillation** to an efficient DNN model
      - Need to keep pretrained model

- No guarantees which approach is better (open area of research)

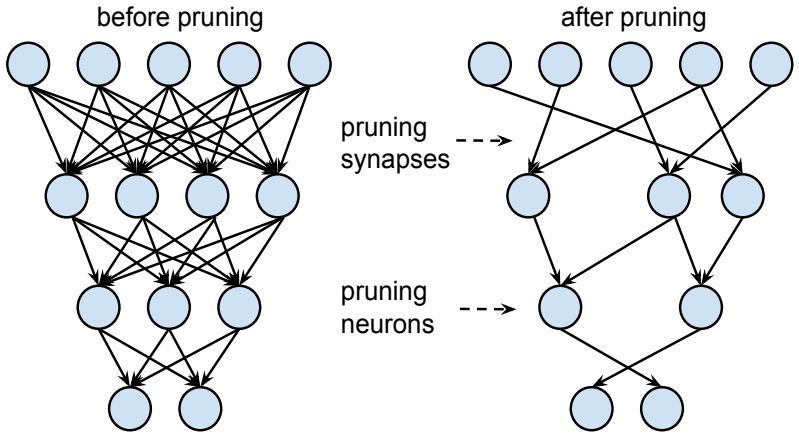


[**Bucilu**, *KDD* 2006],[**Hinton**, arXiv 2015]

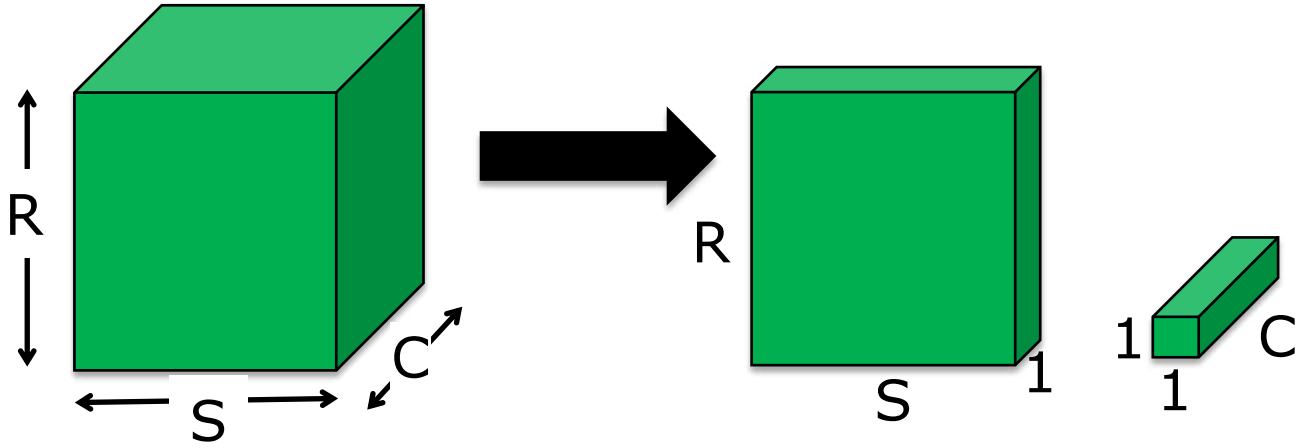
Flexibility & Scalability

Many Efficient DNN Design Approaches

Network Pruning



Efficient Network Architectures



Reduce Precision

32-bit float

8-bit fixed

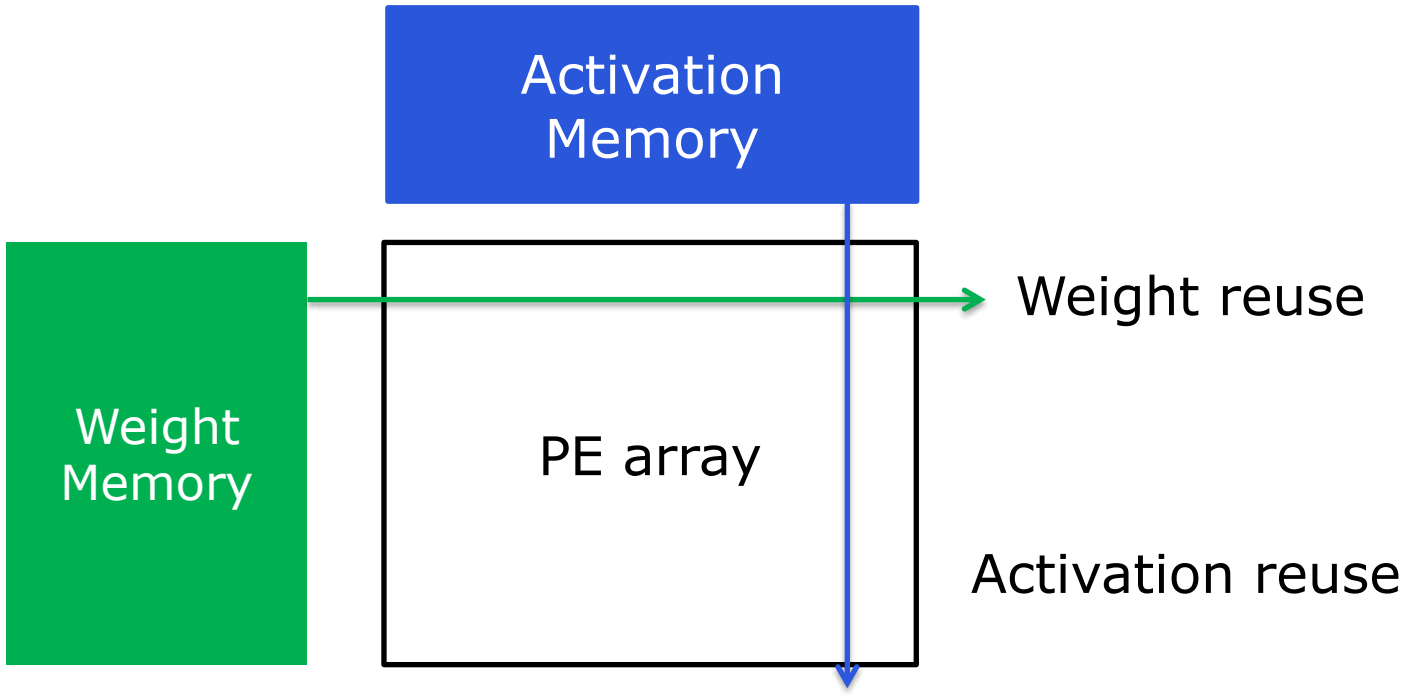
Binary

No guarantee that DNN algorithm designer will use a given approach. **Need flexible DNN processor!**

[**Chen**, *SysML* 2018]

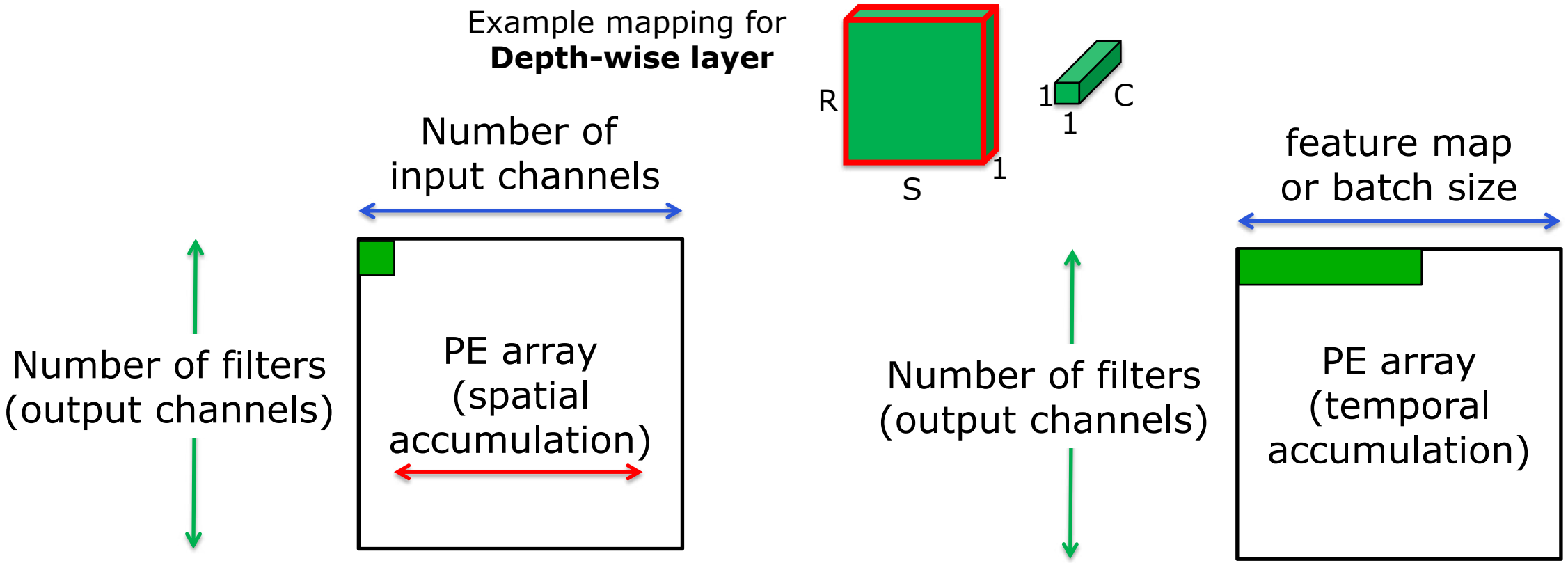
Limitations of Existing DNN Processors

- Specialized DNN processors often rely on certain properties of the DNN model in order to achieve high energy-efficiency
- Example: Reduce memory access by amortizing across PE array



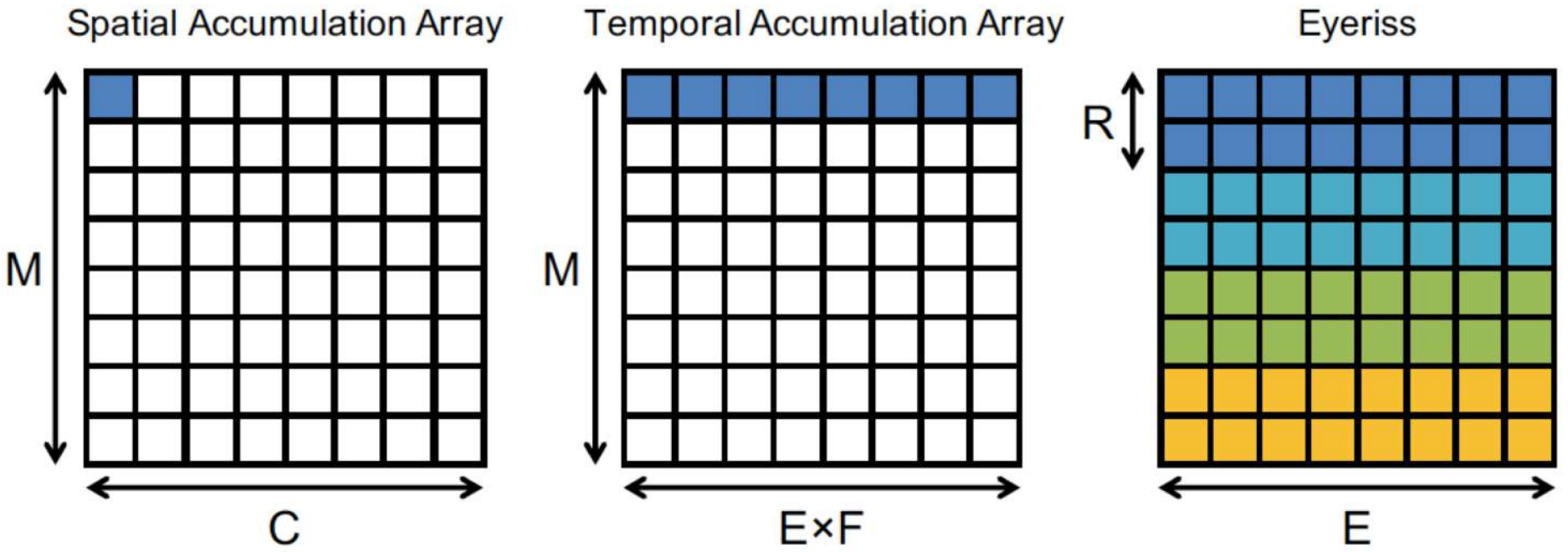
Limitations of Existing DNN Processors

- Reuse depends on # of channels, feature map/batch size
  - Not efficient across all DNN models (e.g., efficient network architectures)



Need Flexible Dataflow

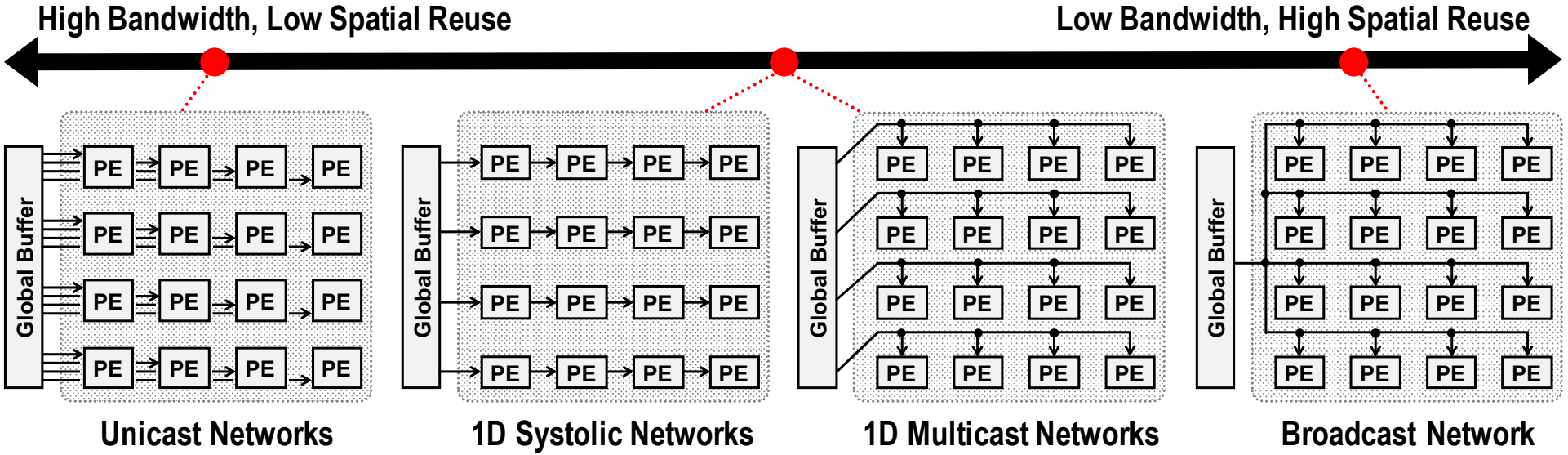
Use flexible dataflow (Row Stationary) to exploit reuse in any dimension of DNN to increase energy efficiency and array utilization



Example: Depth-wise layer

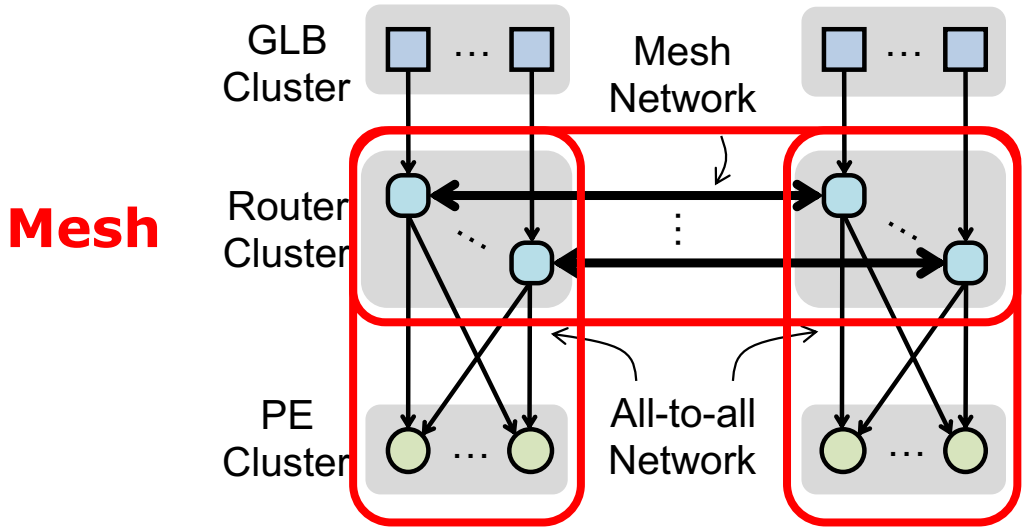
Need Flexible On-Chip Network for Varying Reuse

- When reuse available, need multicast to exploit spatial data reuse for energy efficiency and high array utilization
- When reuse not available, need unicast for high BW for weights for FC and weights & activations for high PE utilization
- An all-to-all on-chip network satisfies above but too expensive and not scalable



[**Chen**, *JETCAS* 2019]

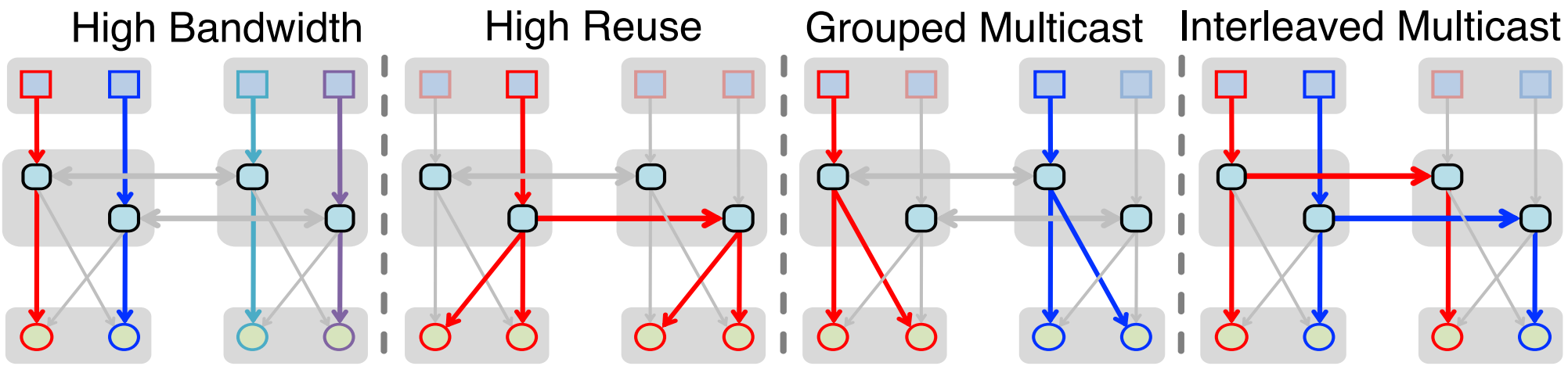
Hierarchical Mesh



Mesh

All-to-All

[**Chen**, *JETCAS* 2019]

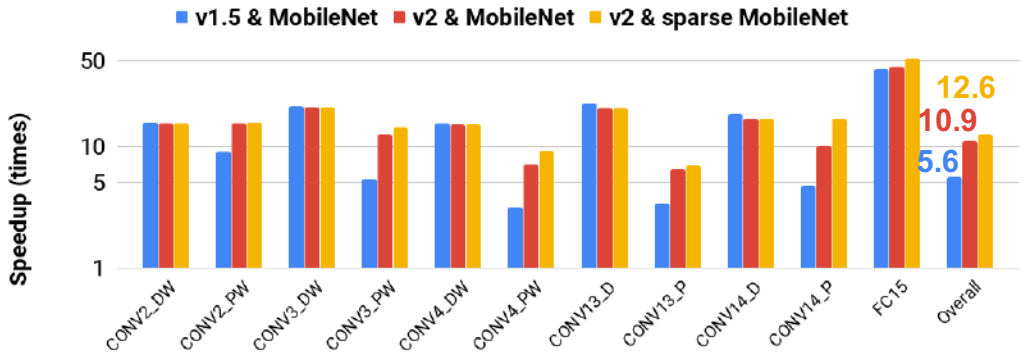


Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

- Wide range of filter shapes
  - Large and Compact
- Different Layers
  - CONV, FC, depth wise, etc.
- Wide range of sparsity
  - Dense and Sparse
- Scalable architecture

Over an order of magnitude faster and more energy efficient than Eyeriss v1



Speed up over Eyeriss v1 scales with number of PEs

# of PEs	256	1024	16384
AlexNet	17.9x	71.5x	1086.7x
GoogLeNet	10.4x	37.8x	448.8x
MobileNet	15.7x	57.9x	873.0x

[**Chen**, *JETCAS* 2019]

Design Considerations for Flexibility and Scalability

- Many of the existing DNN processors rely on certain properties of the DNN model
  - **Properties cannot be guaranteed** as the wide range techniques used for efficient DNN model design has resulted in a more diverse set of DNNs
  - DNN processors should be sufficiently flexible to efficiently support a wide range of techniques

- Evaluate DNN processors on a comprehensive set of benchmarks
  - **MLPerf benchmark** is a start, but may need more (e.g., reduced precision, sparsity, efficient network architectures)

- Evaluate improvement in performance as resources scales up!
  - Multiple chips modules [**Zimmer**, *VLSI* 2019] and Wafer Scale [**Lie**, *HotChips* 2019]

Design Considerations for ASIC

- **Increase PE utilization**
  - Flexible mapping and on-chip network for different DNN models → requires additional hardware
- **Reduce data movement**
  - Custom memory hierarchy and dataflows that exploit data reuse
  - Apply compression to exploit redundancy in data → requires additional hardware
- **Reduce time and energy per MAC**
  - Reduce precision → if precision varies, requires additional hardware; impact on accuracy
- **Reduce unnecessary MACs**
  - Exploit sparsity → requires additional hardware; impact on accuracy
  - Exploit redundant operations → requires additional hardware

Other Platforms

Processing In Memory / In Memory Compute

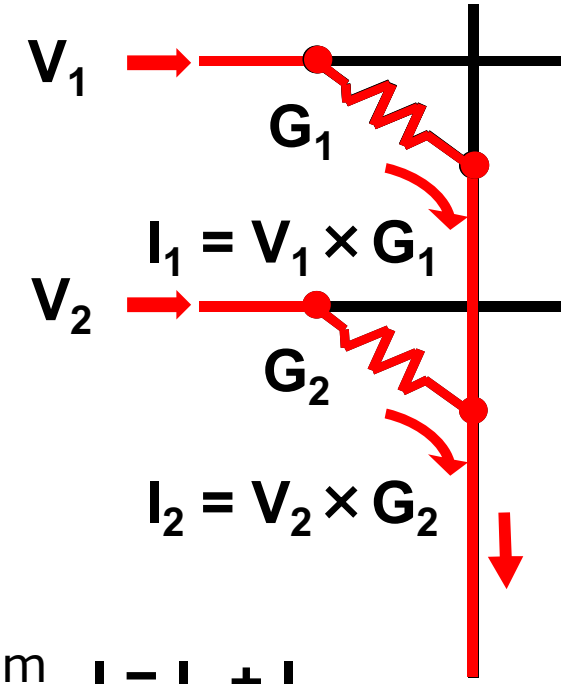
- Reduce data movement by moving compute into memory

- Analog compute
  - **Increased sensitivity** to circuit non-idealities: non-linearities, process, voltage, and temperature variations

eNVM:[**Yu**, *PIEEE* 2018], SRAM:[**Verma**, *SSCS* 2019]

More details in tutorial @ ISSCC 2020

Activation is input voltage (V_i)
Weight is resistor conductance (G_i)



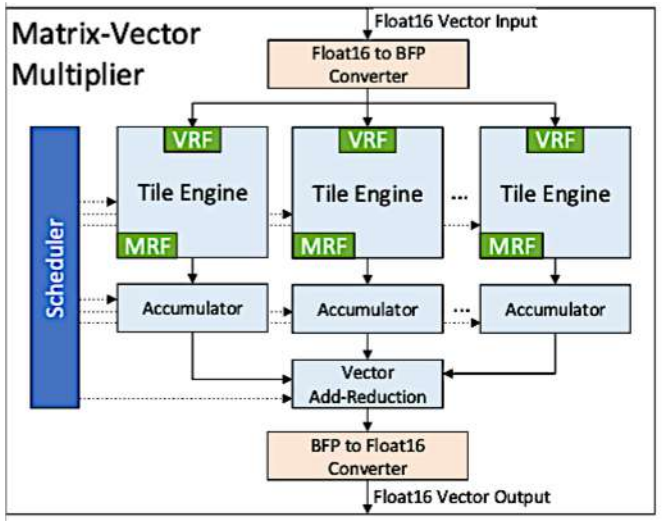
Partial sum is output current

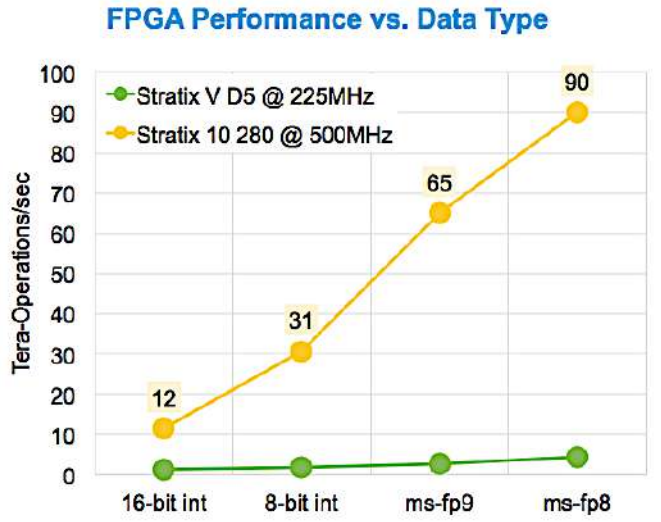
$I = I_1 + I_2$
$= V_1 \times G_1 + V_2 \times G_2$

Image Source: [**Shafiee**, *ISCA* 2016]

Field Programmable Gate Array (FPGA)

- Often implemented as **matrix-vector multiply**
  - e.g., Microsoft Brainwave NPU [**Fowers**, *ISCA* 2018]
- A popular approach uses **weight stationary dataflow** and stores all weights on FPGA for low latency (batch size of 1)
- Reduced precision to fit more weights and MACs on FPGA

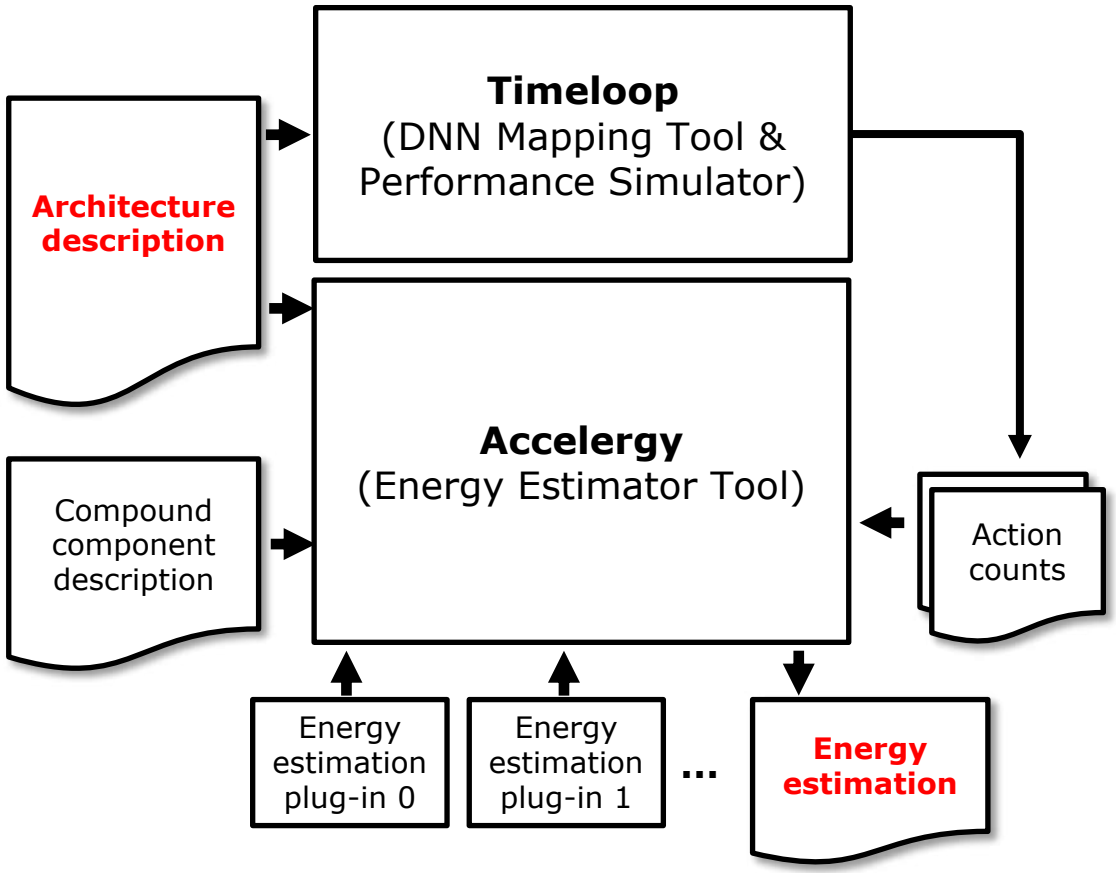




More details in tutorial @ ISSCC 2020

DNN Processor Evaluation Tools

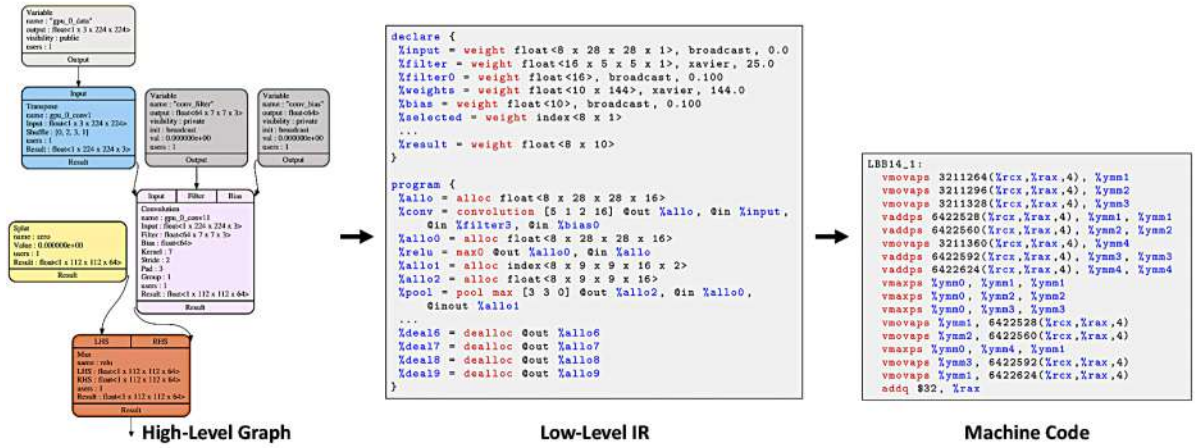
- Require systematic way to
  - Evaluate and compare wide range of DNN processor designs
  - Rapidly explore design space
- **Accelergy** [**Wu**, *ICCAD* 2019]
  - Early stage energy estimation tool at the architecture level
    - Estimate energy consumption based on architecture level components (e.g., # of PEs, memory size, on-chip network)
  - Evaluate architecture level energy impact of emerging devices
    - Plug-ins for different technologies
- **Timeloop** [**Parashar**, *ISPASS* 2019]
  - DNN mapping tool
  - Performance Simulator → Action counts



Open-source code available at:
http://accelergy.mit.edu

DNN Compilers for Diverse DNN Platforms

Compilers generate optimized code for various DNN platforms (backends) from high-level frameworks



https://tvm.apache.org/

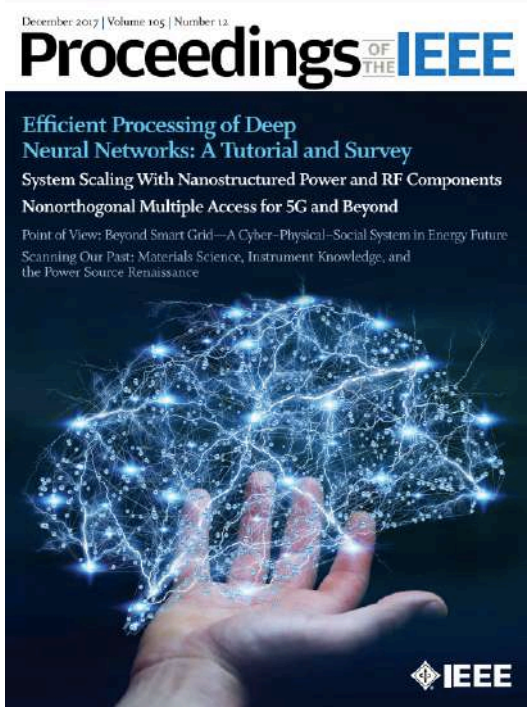
https://github.com/pytorch/glow

Summary

- **DNNs are a critical component in the AI revolution**, delivering record breaking accuracy on many important AI tasks for a wide range of applications; however, it comes at the cost of **high computational complexity**
- Efficient processing of DNNs is an important area of research with many promising **opportunities for innovation at various levels** of hardware design, including algorithm co-design
- When considering different DNN solutions it is important to **evaluate with the appropriate workload** in term of both input and model, and recognize that they are evolving rapidly
- It is important to consider **a comprehensive set of metrics when evaluating different DNN solutions**: accuracy, throughput, latency, power, energy, flexibility, scalability and cost

Additional Resources

V. Sze, Y.-H. Chen, T-J. Yang, J. Emer,
"***Efficient Processing of Deep Neural Networks: A Tutorial and Survey***," Proceedings of the IEEE, Dec. 2017



Book Coming Soon!

MIT Professional Education Course on
"Designing Efficient Deep Learning Systems"
http://professional-education.mit.edu/deeplearning

DNN tutorial website
http://eyeriss.mit.edu/tutorial.html

More info about our research on efficient computing for DNNs, robotics, and health care
http://sze.mit.edu

For updates

EEMS Mailing List

Follow @eems_mit

References (1 of 4)

Deep Learning Overview

- Transformer: Vaswani et al, "Attention is all you need," NeurIPS 2017
- LeNet: LeCun, Yann, et al. "Gradient-based learning applied to document recognition," Proc. IEEE 1998
- AlexNet: Krizhevsky et al. "Imagenet classification with deep convolutional neural networks," NeurIPS 2012
- VGGNet: Simonyan et al.. "Very deep convolutional networks for large-scale image recognition," ICLR 2015
- GoogleNet: Szegedy et al. "Going deeper with convolutions," CVPR 2015
- ResNet: He et al. "Deep residual learning for image recognition," CVPR 2016
- EfficientNet: Tan et al., "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019

Key Metrics and Design Objectives

- Sze et al., "Hardware for machine learning: Challenges and opportunities," CICC 2017, https://www.youtube.com/watch?v=8Qa0E1jdkrE&feature=youtu.be
- Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," Proc. IEEE 2017
- Williams et al., "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," CACM 2009
- Chen et al., Eyexam, https://arxiv.org/abs/1807.07928

CPU and GPU Platforms

- Mathieu et al., "Fast training of convolutional networks through FFTs," ICLR 2014
- Cong et al., "Minimizing computation in convolutional neural networks," ICANN 2014
- Lavin et al., "Fast algorithms for convolutional neural networks," CVPR 2016

References (2 of 4)

Specialized / Domain-Specific Hardware (ASICs): Efficient Dataflows

- Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," ISCA 2016
- Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," ISCA 2017
- NVDLA, http://nvdla.org
- Moons et al., "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," VLSI 2017
- Parashar et al, "Scnn: An accelerator for compressed-sparse convolutional neural networks," ISCA 2017
- Chen et al., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," ISSCC 2016, http://eyeriss.mit.edu
- Suleiman et al., "Towards Closing the Energy Gap Between HOG and CNN Features for Embedded Vision," ISCAS 2017
- Suleiman et al., "A 58.6mW Real-time Programmable Object Detection with Multi-Scale Multi-Object Support Using Deformable Parts Models on 1920×1080 Video at 30fps," VLSI 2016

Co-Design: Reduced Precision

- Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations," NeurIPS 2015
- Li et al., "Ternary weight networks," NeurIPS Workshop 2016
- Rategari et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," ECCV 2016
- Lee et al., "LogNet: Energy-efficient neural networks using logarithmic computation," ICASSP 2017
- Camus et al., "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," JETCAS 2019

References (3 of 4)

Co-Design: Sparsity

- Lecun et al., Optimal Brain Damage," NeurIPS 1990
- Han et al., "Learning both weights and connections for efficient neural networks," NeurIPS 2015
- Albericio et al., "Cnvlutin: Ineffectual-neuron-free deep neural network computing", ISCA 2016

Co-Design: Efficient Network Architectures

- Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv 2017
- Zhang et al., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," CVPR 2018
- Lin et al., "Network in network" ICLR 2014
- Huang et al., "Densely Connected Convolutional Networks," CVPR 2017
- Yu et al., "Deep layer aggregation," CVPR 2018
- Zoph at al., "Learning Transferable Architectures for Scalable Image Recognition," CVPR 2018

Co-Design: Hardware in the Loop

- Yang et al., "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," CVPR 2017, http://eyeriss.mit.edu/energy.html
- Yang et al., "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," ECCV 2018, http://netadapt.mit.edu

References (4 of 4)

Specialized Hardware (ASICs): Flexibility and Scalability

- Chen et al., "Understanding the Limitations of Existing Energy-Efficient Design Approaches for Deep Neural Networks," SysML 2018, https://www.youtube.com/watch?v=XCdy5egmvaU
- Chen et al., "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," JETCAS 2019
- Zimmer et al., "A 0.11pJ/Op, 0.32-128 TOPS, Scalable Multi-Chip-Module-based Deep Neural Network Accelerator with Ground-Reference Signaling in 16nm," VLSI 2019
- Lie (Cerebras), "Wafer Scale Deep Learning," Hot Chips 2019

Other Platforms: Processing In Memory / In-Memory Computing and FPGAs

- Sze et al., "How to Understand and Evaluate Deep Learning Processors," ISSCC Tutorial 2020, http://isscc.org/
- Verma et al., "In-Memory Computing: Advances and prospects," ISSCC Tutorial 2018 / SSCS Magazine 2019
- Yu, "Neuro-Inspired Computing with Emerging Nonvolatile Memorys," Proc. IEEE 2018
- Yang et al., "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," IEDM 2019
- Fowers et al., "A configurable cloud-scale DNN processor for real-time AI," ISCA 2018

DNN Processor Evaluation Tools

- Wu et al., "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," ICCAD 2019, http://accelergy.mit.edu
- Parashar et al., "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," ISPASS 2019

NetAdapt: Problem Formulation

$$\max_{Net} Acc(Net) \ subject\ to\ Res_j(Net) \leq Bud_j, j = 1, \cdots, m$$

Break into a set of simpler problems and solve iteratively

$$\max_{Net_i} Acc(Net_i) \ subject\ to\ Res_j(Net_i) \leq Res_j(Net_{i-1}) - \Delta R_{i,j}, j = 1, \cdots, m$$

*Acc: accuracy function, Res: resource evaluation function,
ΔR: resource reduction, Bud: given budget

- **Advantages**
  - Supports multiple resource budgets at the same time
  - Guarantees that the budgets will be satisfied because the resource consumption decreases monotonically
  - Generates a family of networks (from each iteration) with different resource versus accuracy trade-offs

NetAdapt: Simplified Example of One Iteration