Accelerating Inference for DNNs

Yu-Hsin Chen

PhD, MIT
NVIDIA Research

CS217
October 18, 2018

Contributors to this Talk

Joel Emer

Senior Distinguished Research Scientist

NVIDIA

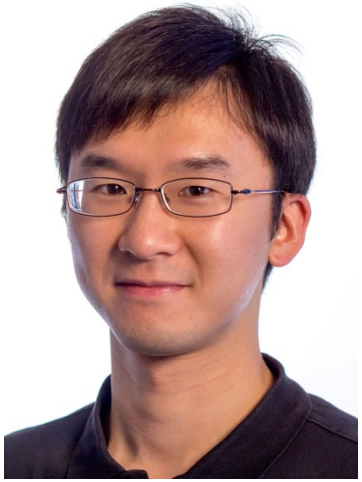
Professor

MIT

Vivienne Sze

Professor

MIT



Yu-Hsin Chen

Research Scientist

NVIDIA

PhD

MIT

Tien-Ju Yang

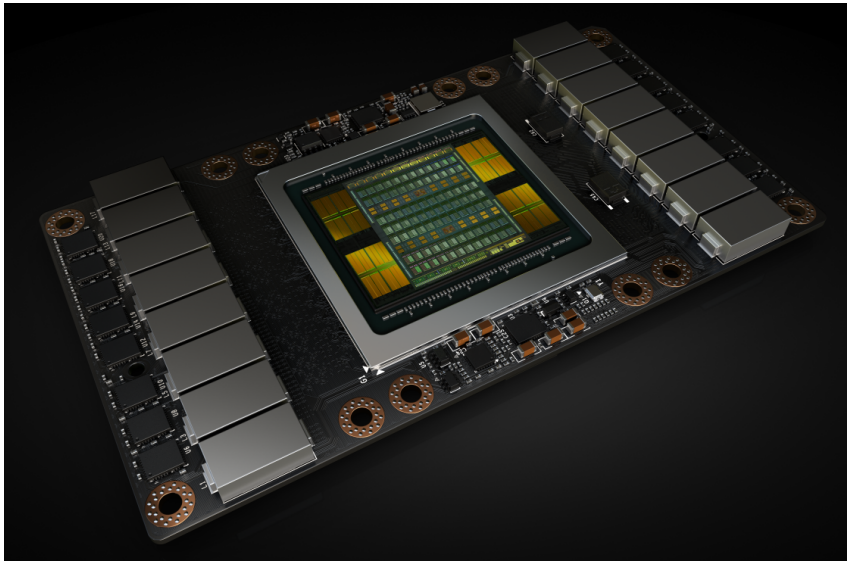
PhD Candidate

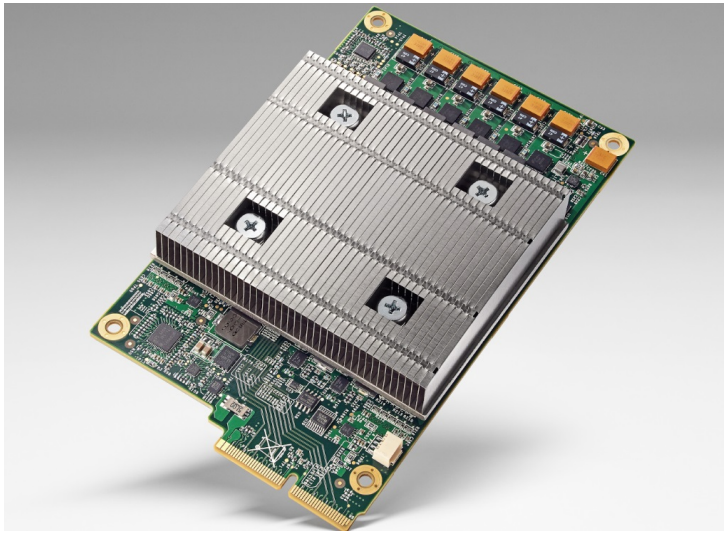
MIT

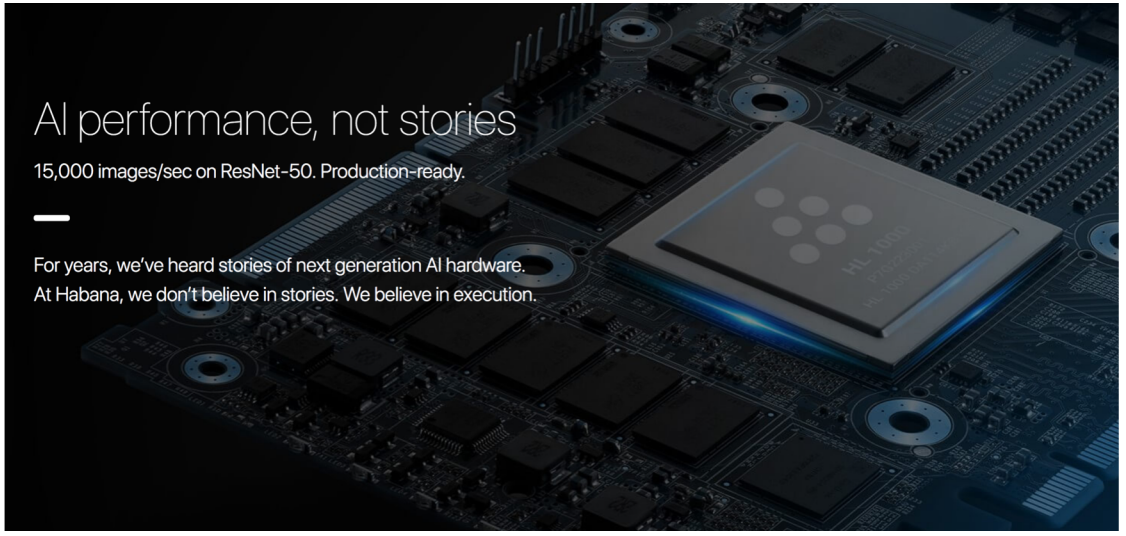
DNNs are Everywhere

[image sources: Amazon, nature, sciencenews.org]

Hardware Designed for DNNs







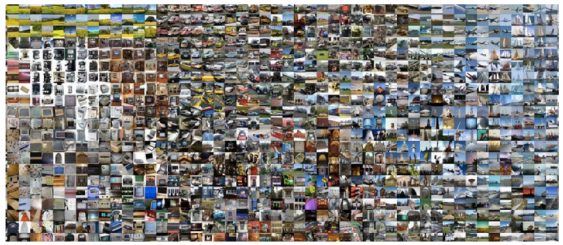
[image sources: NVIDIA, Huawei, Google, Habana Labs]

Key Metrics to Evaluate DNN Hardware

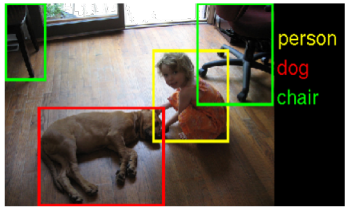
- **Accuracy**
  - Evaluate using the appropriate DNN model and dataset
- **Programmability**
  - Support multiple applications
- **Throughput / Latency**
  - GOPS, frame rate, delay
- **Energy / Power**
  - Energy per operation
  - DRAM Bandwidth
- **Cost**
  - Area (memory size, # of cores)

MNIST

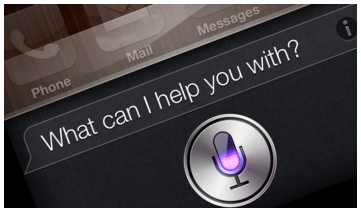
IMAGENET

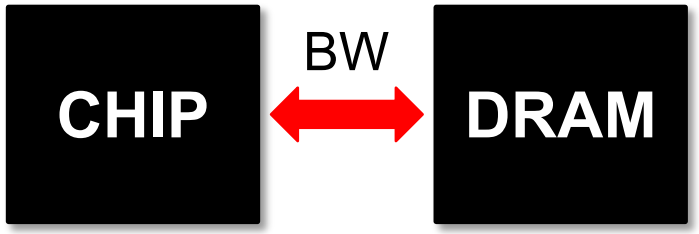


Computer Vision



Speech Recognition





[Sze et al., CICC 2017]

Outlines

- **Architecture Design for DNN Accelerators**
- **DNN Model and Hardware Co-Design**

For more details:

Efficient Processing of Deep Neural Networks: A Tutorial and Survey
V.Sze, Y.-H. Chen, T.-J. Yang, J. Emer
Proceedings of IEEE
https://arxiv.org/abs/1703.09039

Your Takeaways

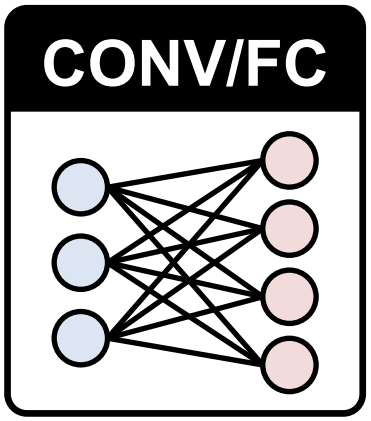
- **Understand the trade-offs between various DNN accelerator architectures**
- **Assess the utility of various optimization approaches**
- **Understand recent implementation trends and opportunities**

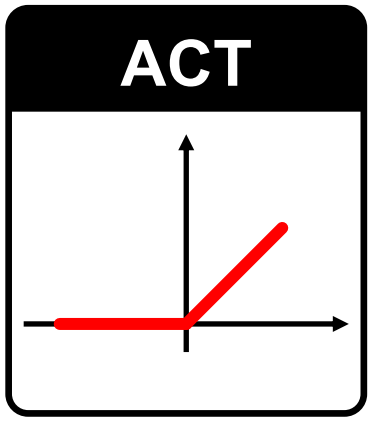
Primer on DNNs

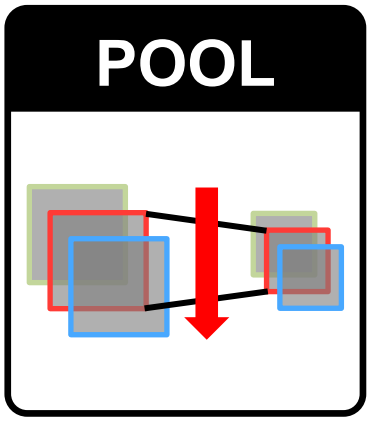
10s – 1000s layers

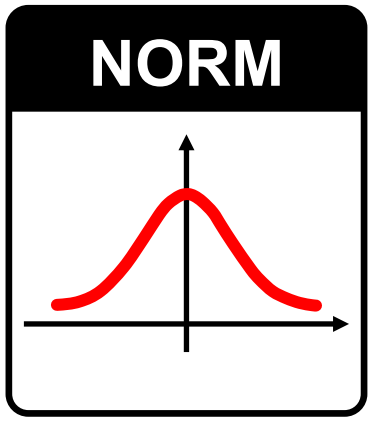
DNN Layer → DNN Layer → … → DNN Layer → "Car"

many layer types to choose from









…

[1] Tesla

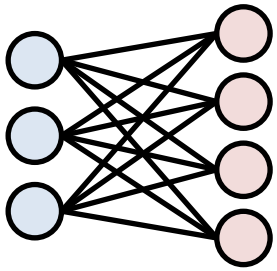
Primer on DNNs

10s – 1000s layers

DNN Layer → DNN Layer → … → DNN Layer → "Car"

many layer types to choose from

CONV/FC



ACT POOL NORM …

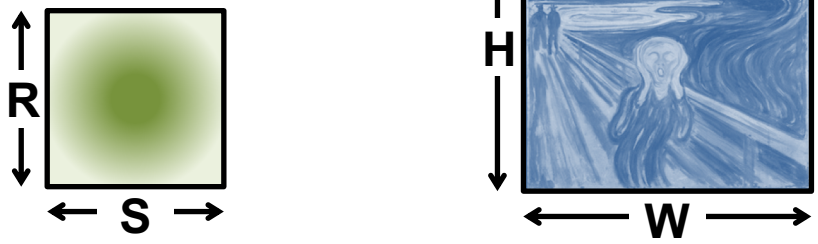
CONV and FC layers account for >99% of the computation

[1] Tesla

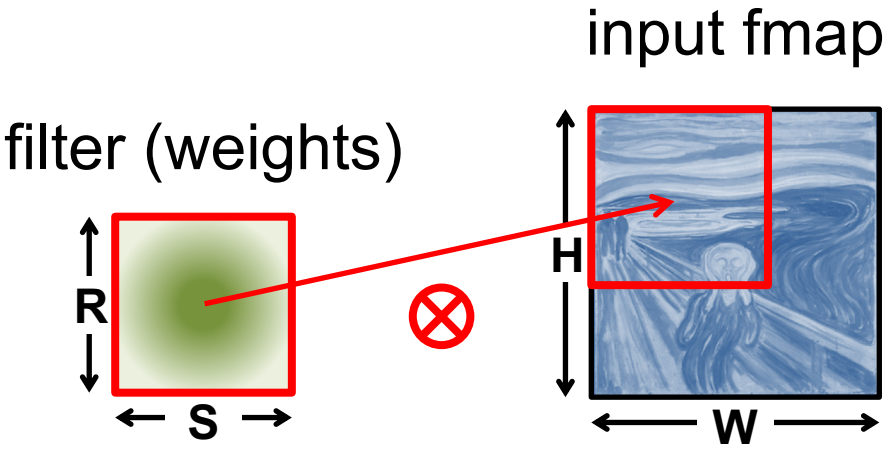
Convolutional (CONV) Layer

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

filter (weights)

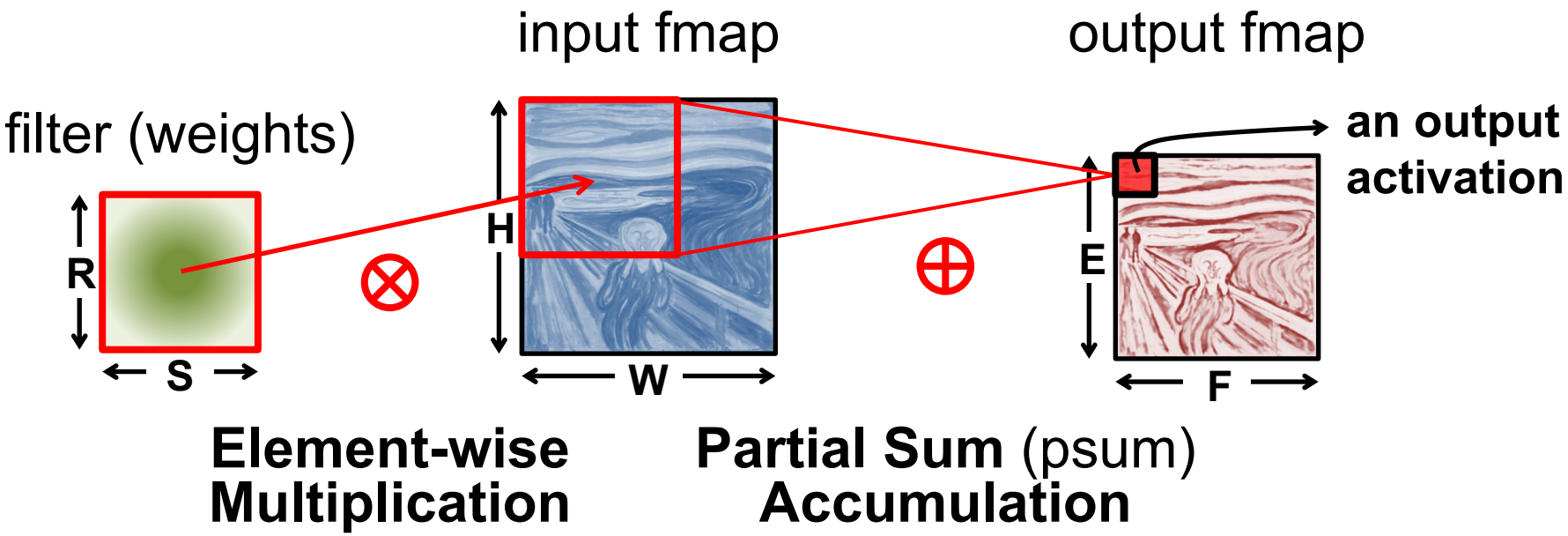


Convolutional (CONV) Layer

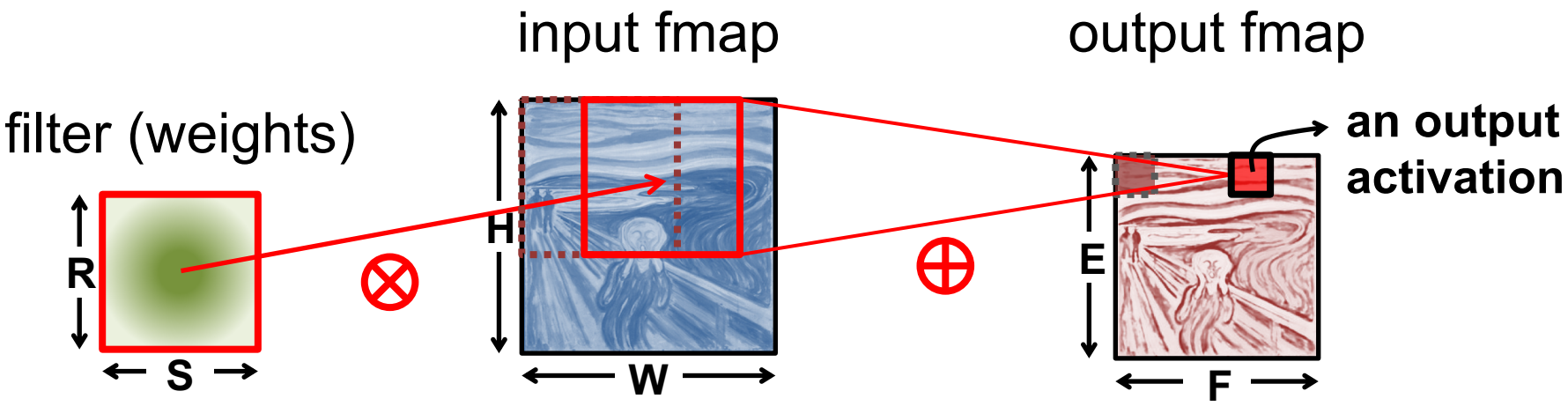


Element-wise Multiplication

Convolutional (CONV) Layer

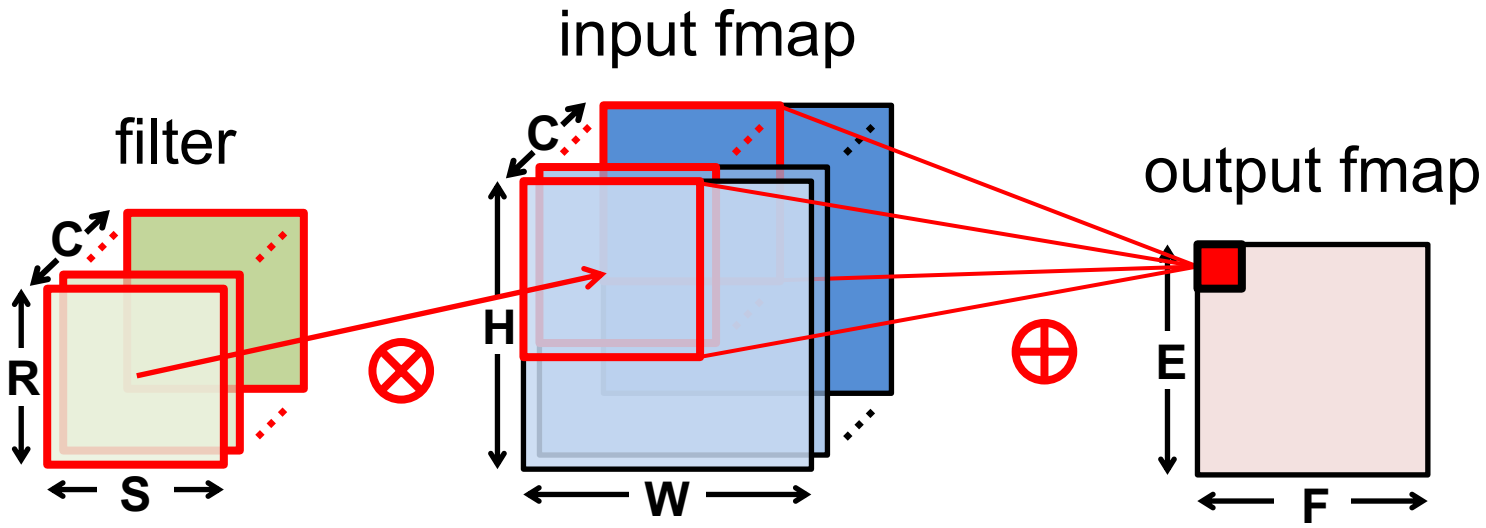


Convolutional (CONV) Layer



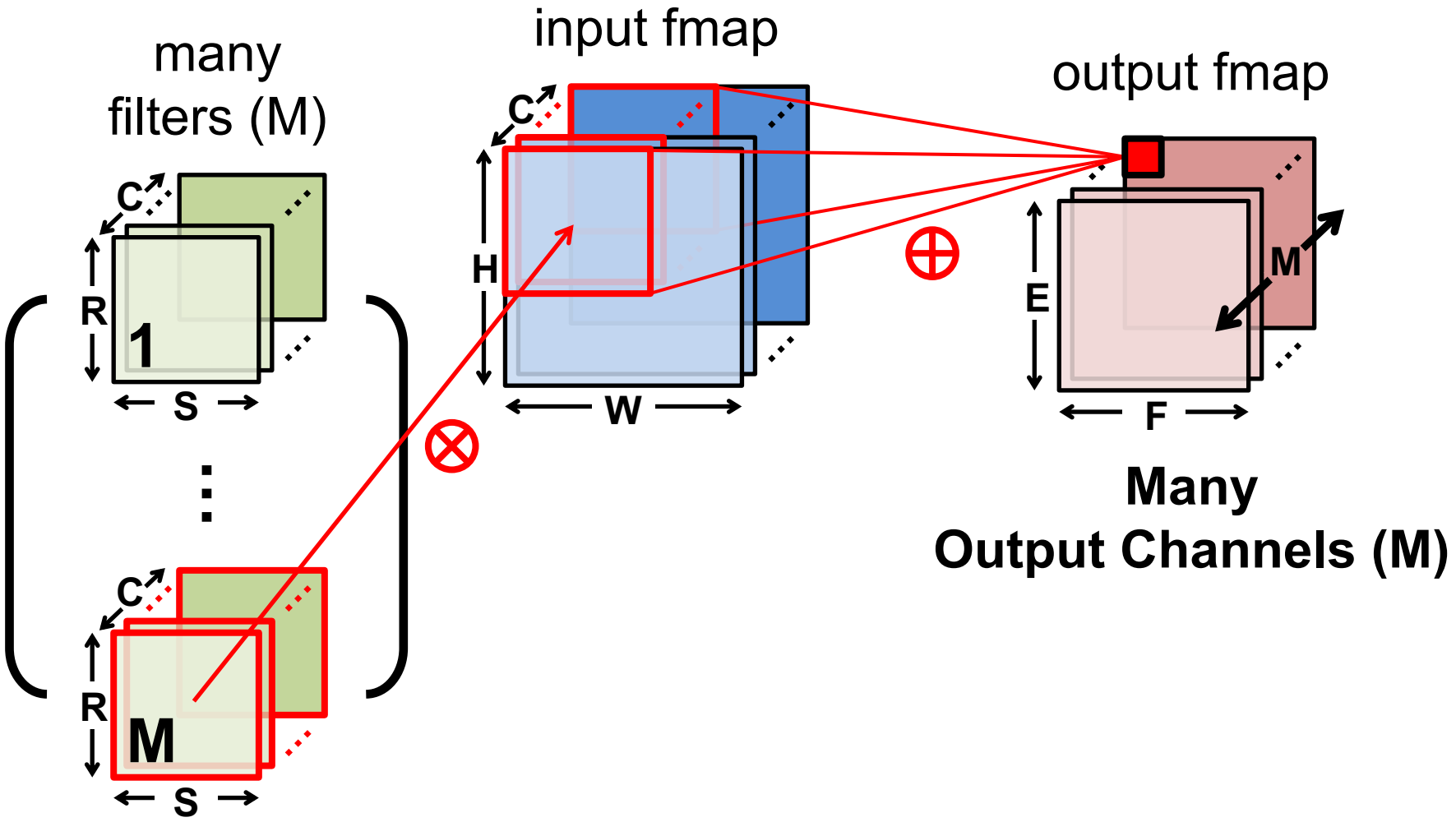
Sliding Window Processing

Convolutional (CONV) Layer

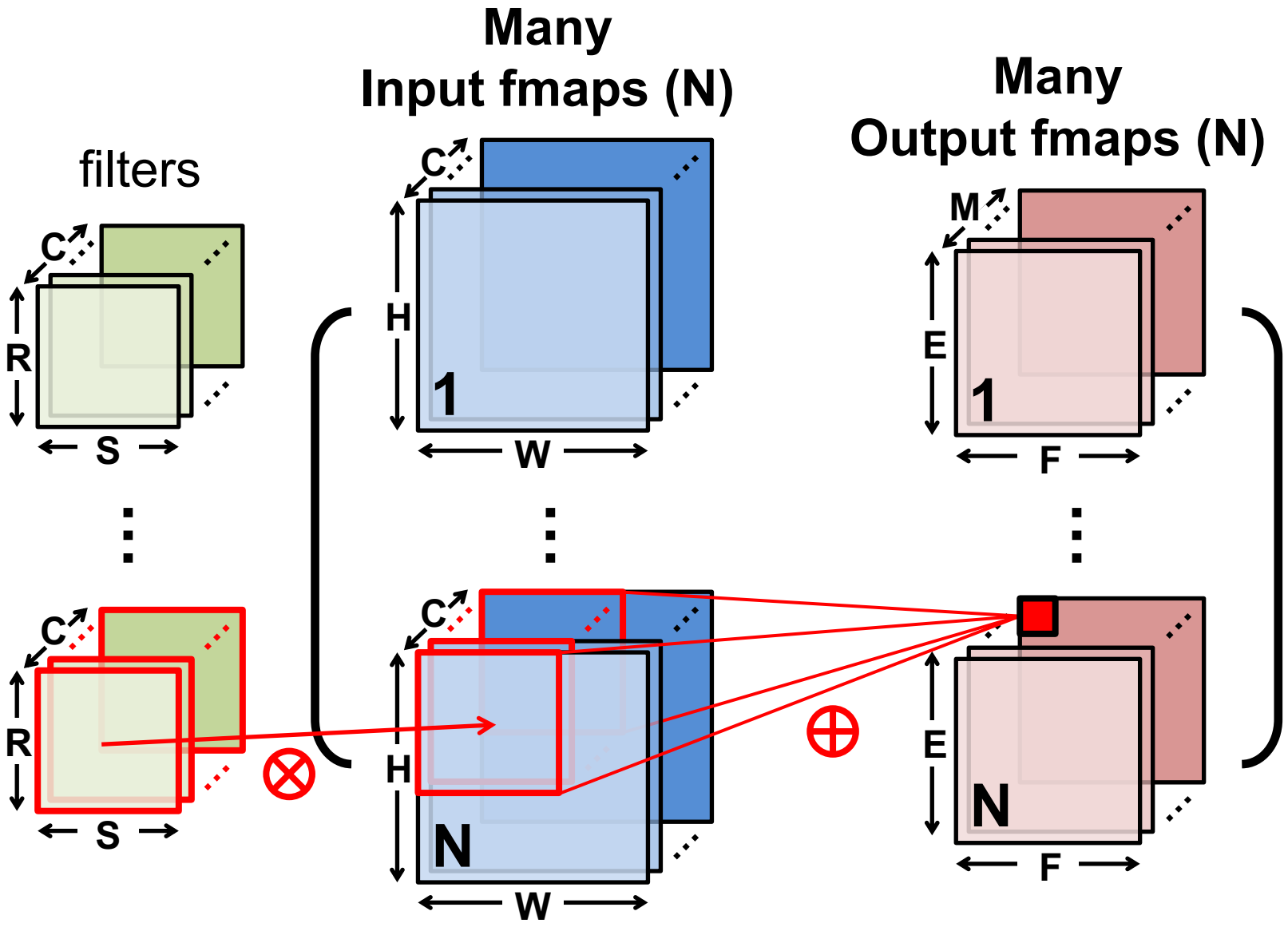


Many Input Channels (C)

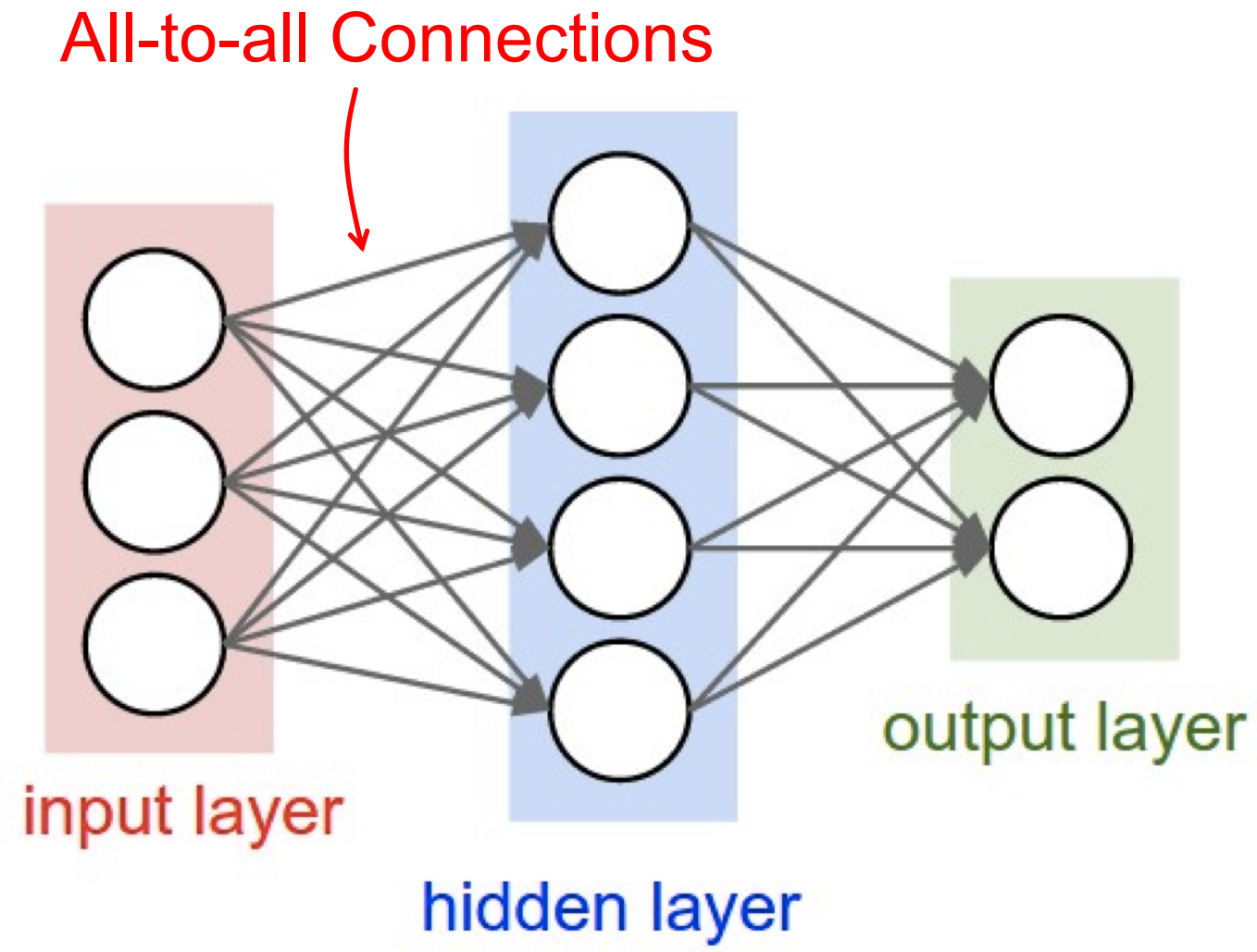
Convolutional (CONV) Layer



Convolutional (CONV) Layer

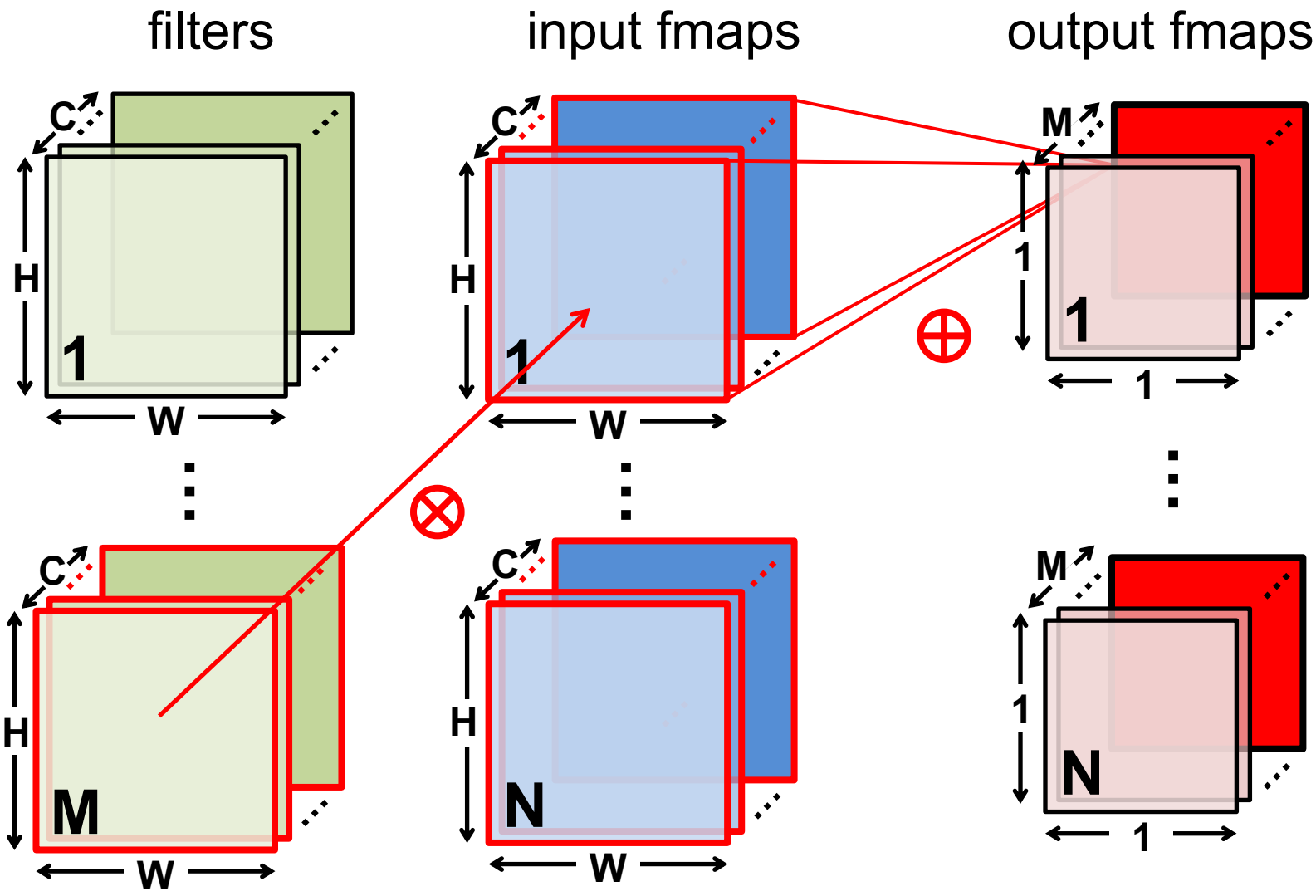


Fully-Connected (FC) Layer



[images source: Stanford cs231n]

FC Layer – from CONV Layer POV

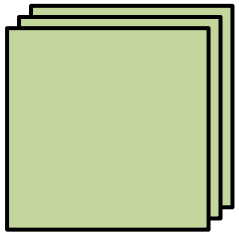


Widely Varying Layer Shapes

AlexNet[1] Layer Shape Configurations

Layer	Filter Size (R)	# Filters (M)	# Channels (C)	Stride
1	11×11	96	3	4
2	5×5	256	48	1
3	3×3	384	256	1
8	1×1	1000	4096	1

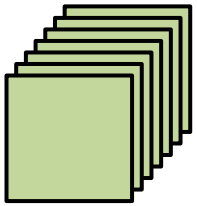
Layer 1



34k weights

105M MACs*

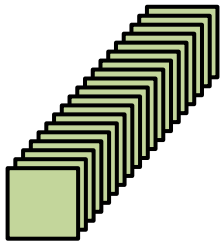
Layer 2



307k weights

224M MACs

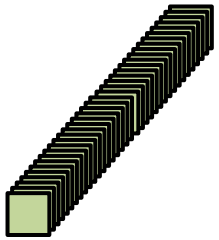
Layer 3



885k weights

150M MACs

Layer 8



4096k weights

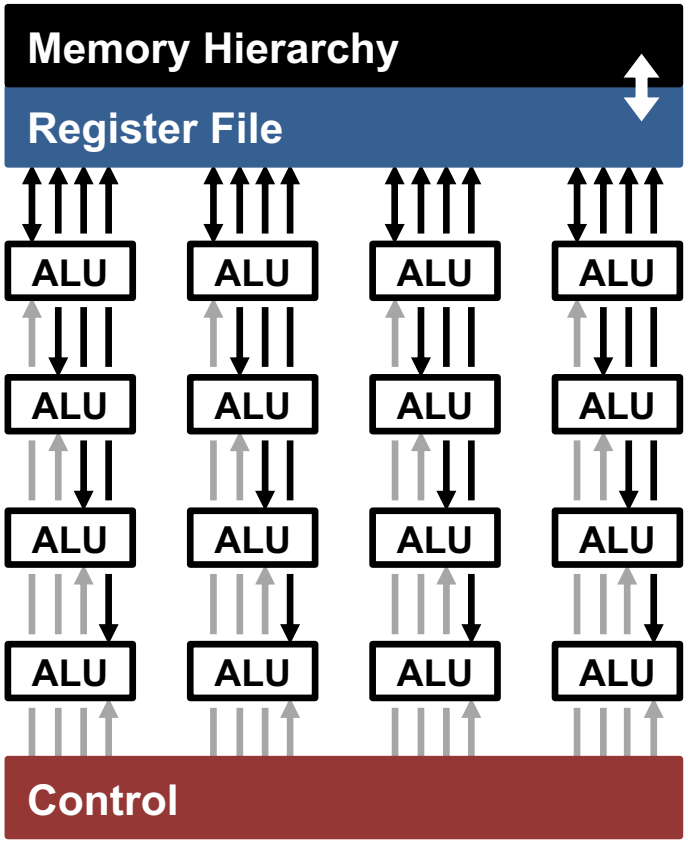
4M MACs

* multiply-accumulate

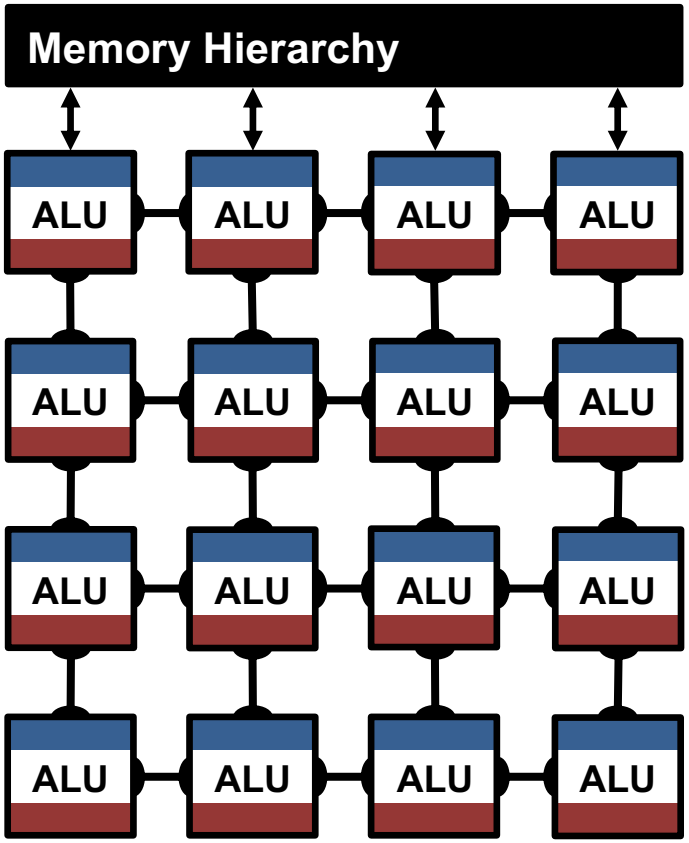
[1] Krizhevsky, *NIPS* 2012

Highly-Parallel Compute Paradigms

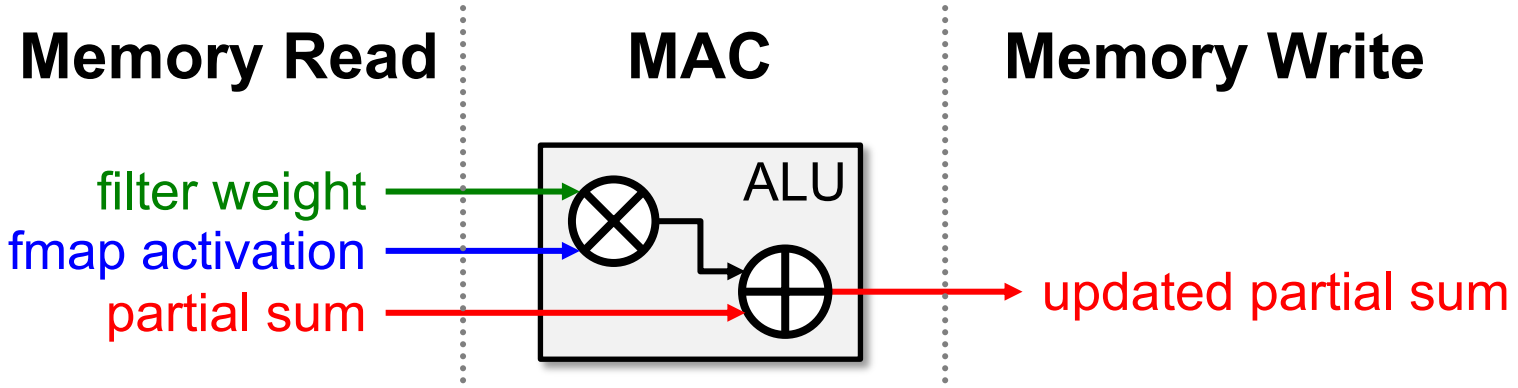
Temporal Architecture (SIMD/SIMT)



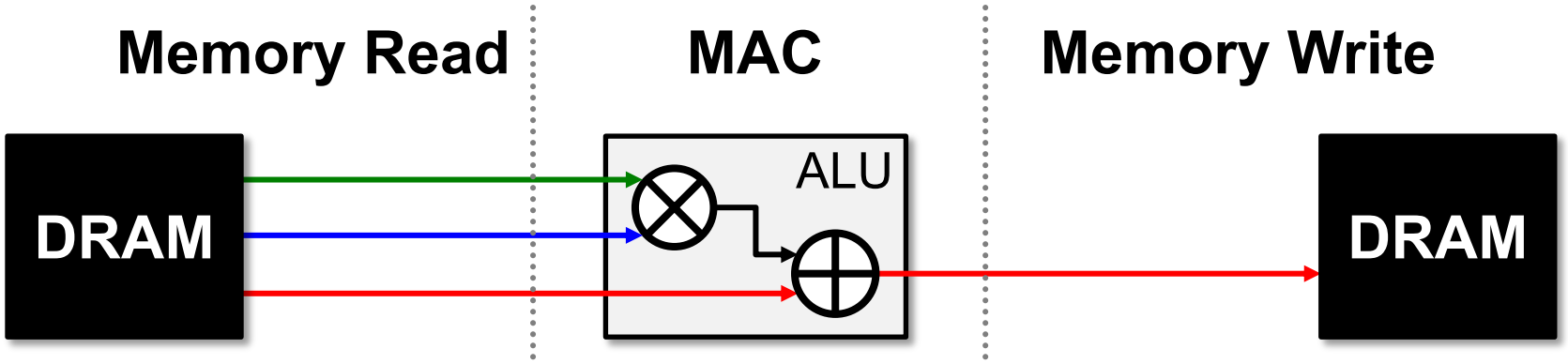
Spatial Architecture (Dataflow Processing)



Memory Access is the Bottleneck



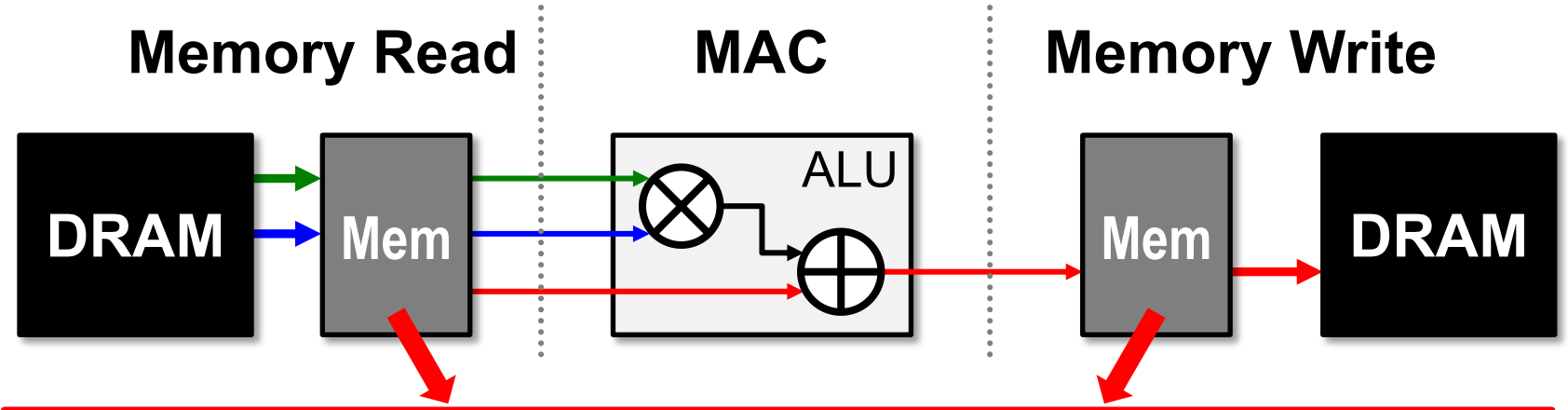
Memory Access is the Bottleneck



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

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

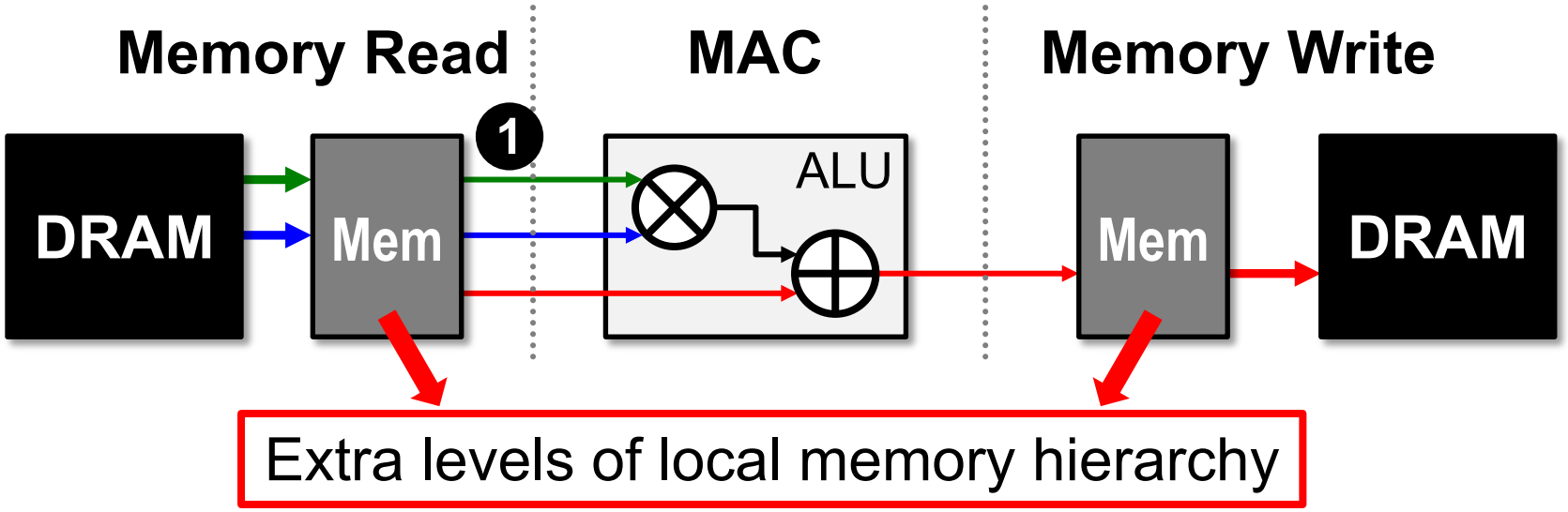
Memory Access is the Bottleneck



Extra levels of local memory hierarchy

Smaller, but **Faster** and **more Energy-Efficient**

Memory Access is the Bottleneck

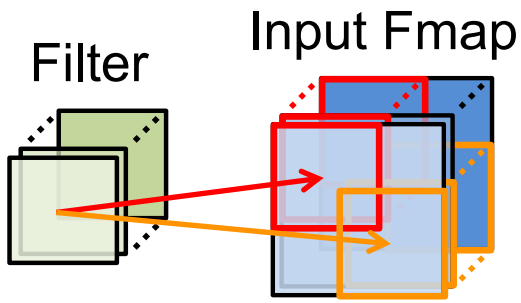


Opportunities: 1 **data reuse**

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

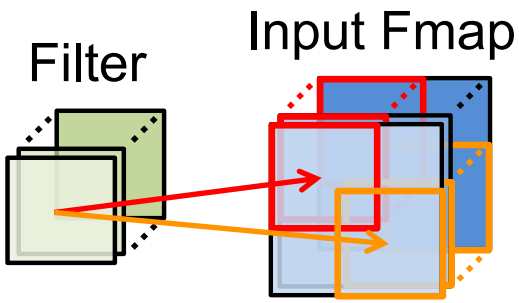


Reuse: Activations
Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

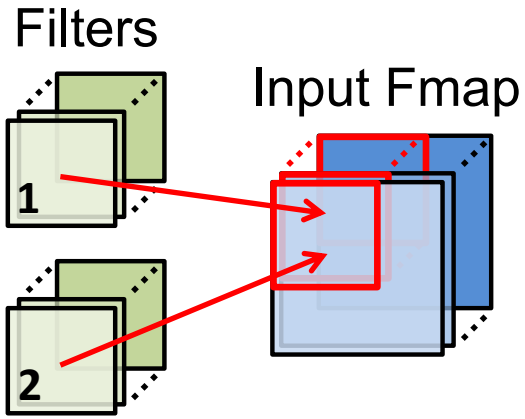
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

CONV and FC layers

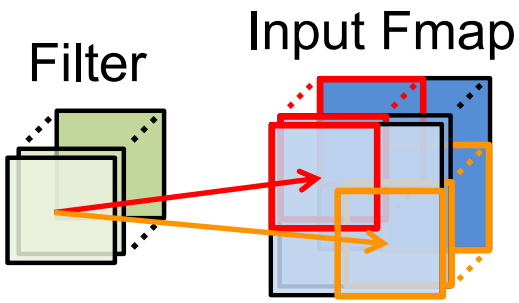


Reuse: Activations

Types of Data Reuse in DNN

Convolutional Reuse

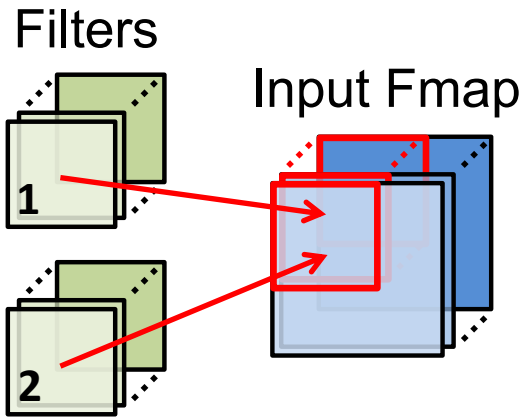
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

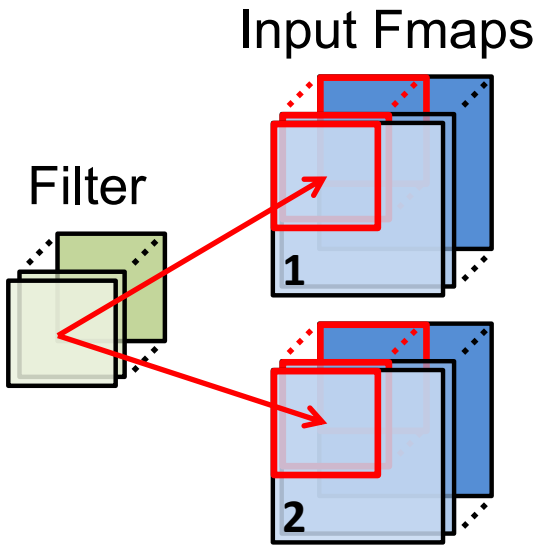
CONV and FC layers



Reuse: Activations

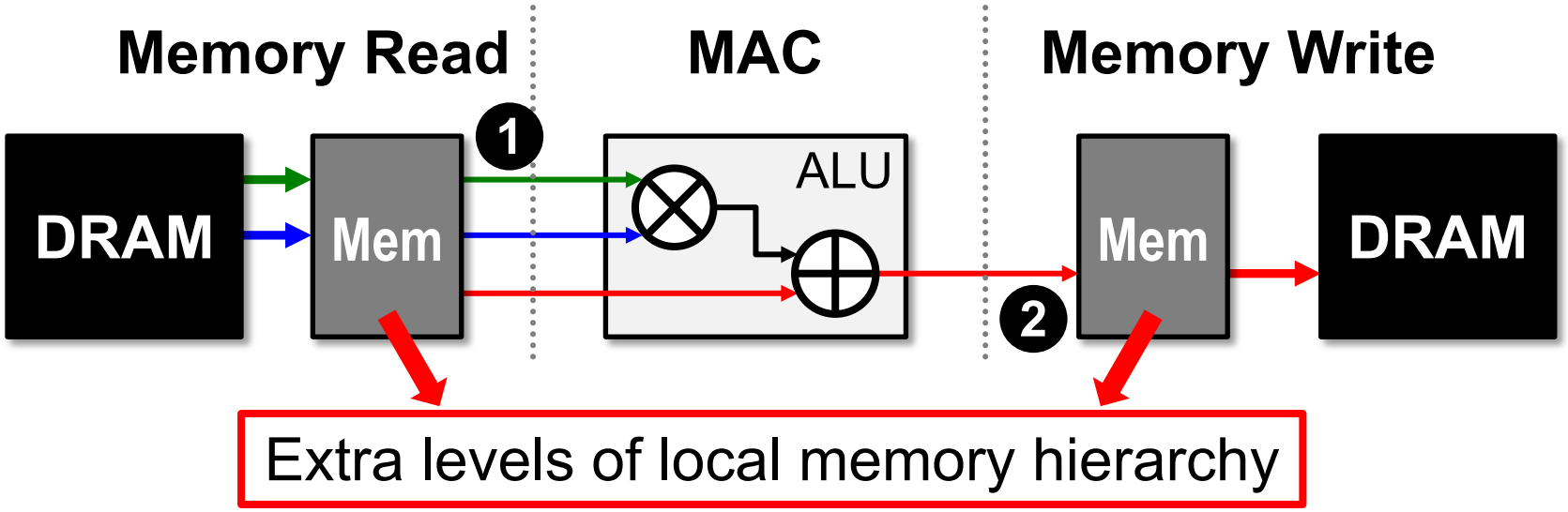
Filter Reuse

CONV and FC layers
(batch size > 1)



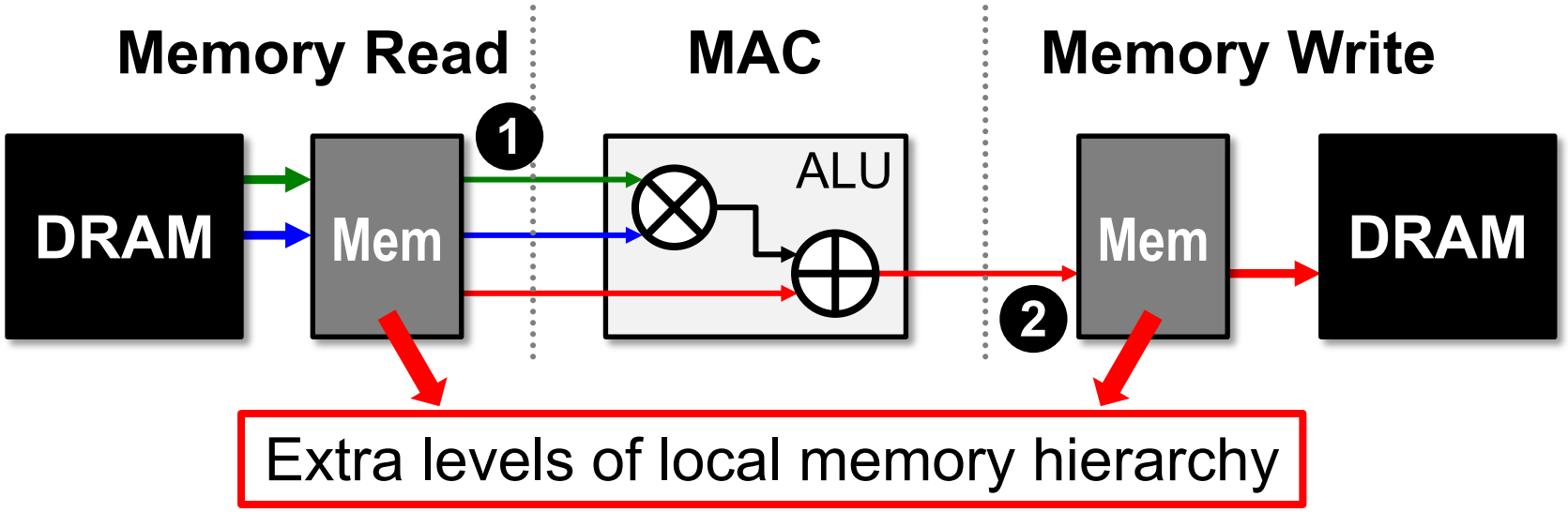
Reuse: Filter weights

Memory Access is the Bottleneck



Opportunities: ❶ **data reuse** ❷ **local accumulation**

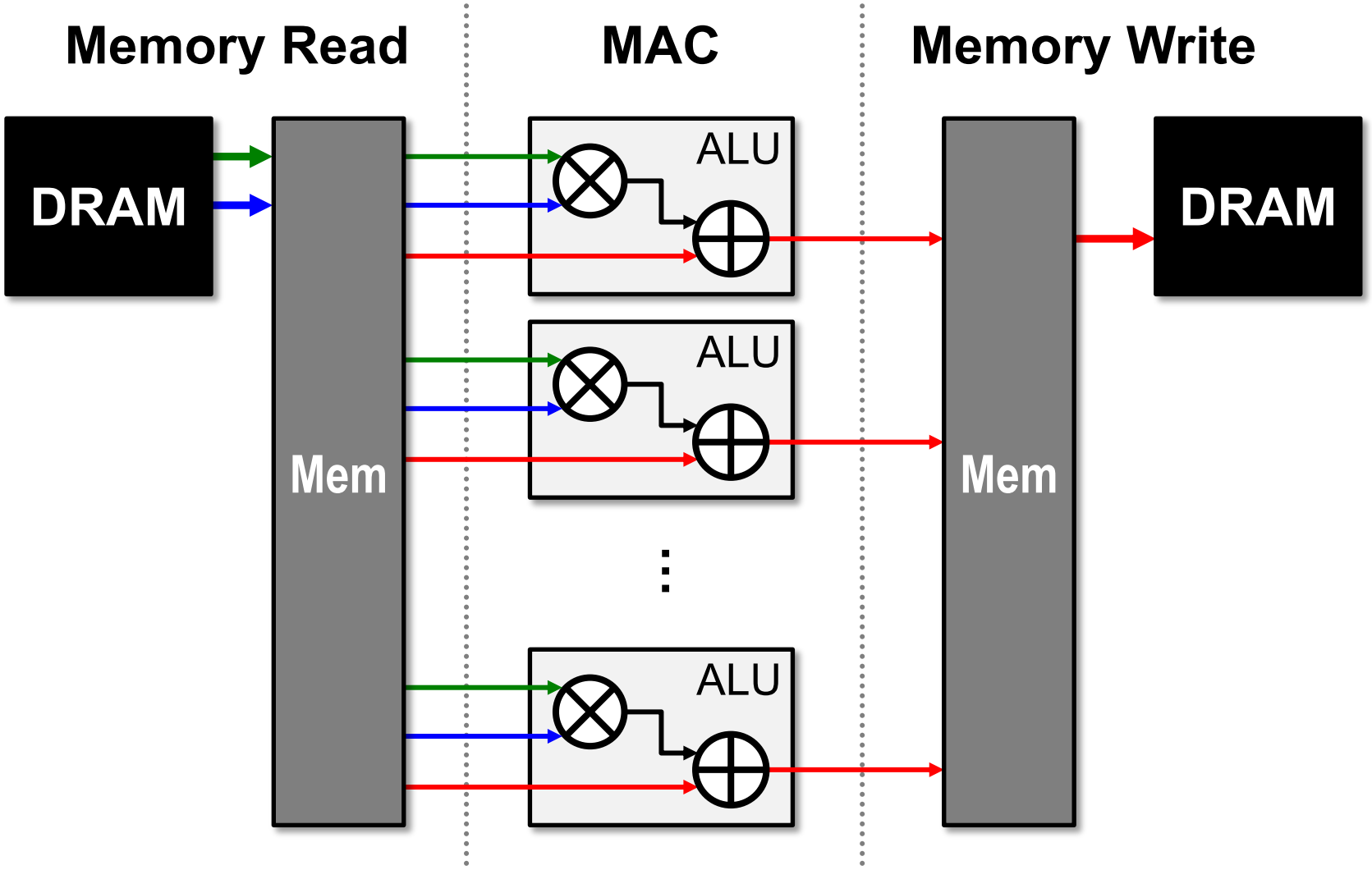
Memory Access is the Bottleneck



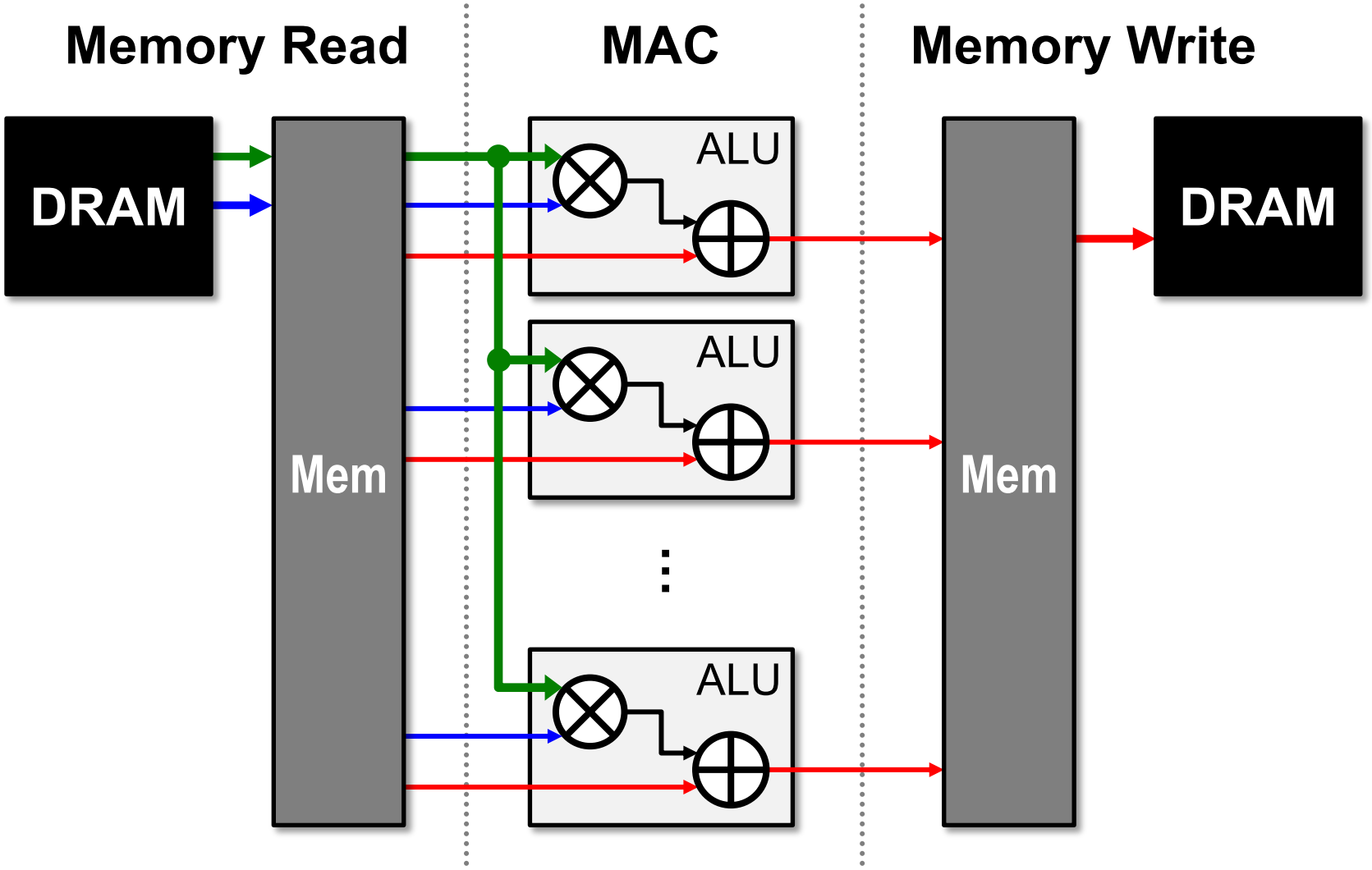
Opportunities: ❶ **data reuse** ❷ **local accumulation**

- Example: DRAM access in AlexNet can be reduced from **2896M** to **61M** (best case)

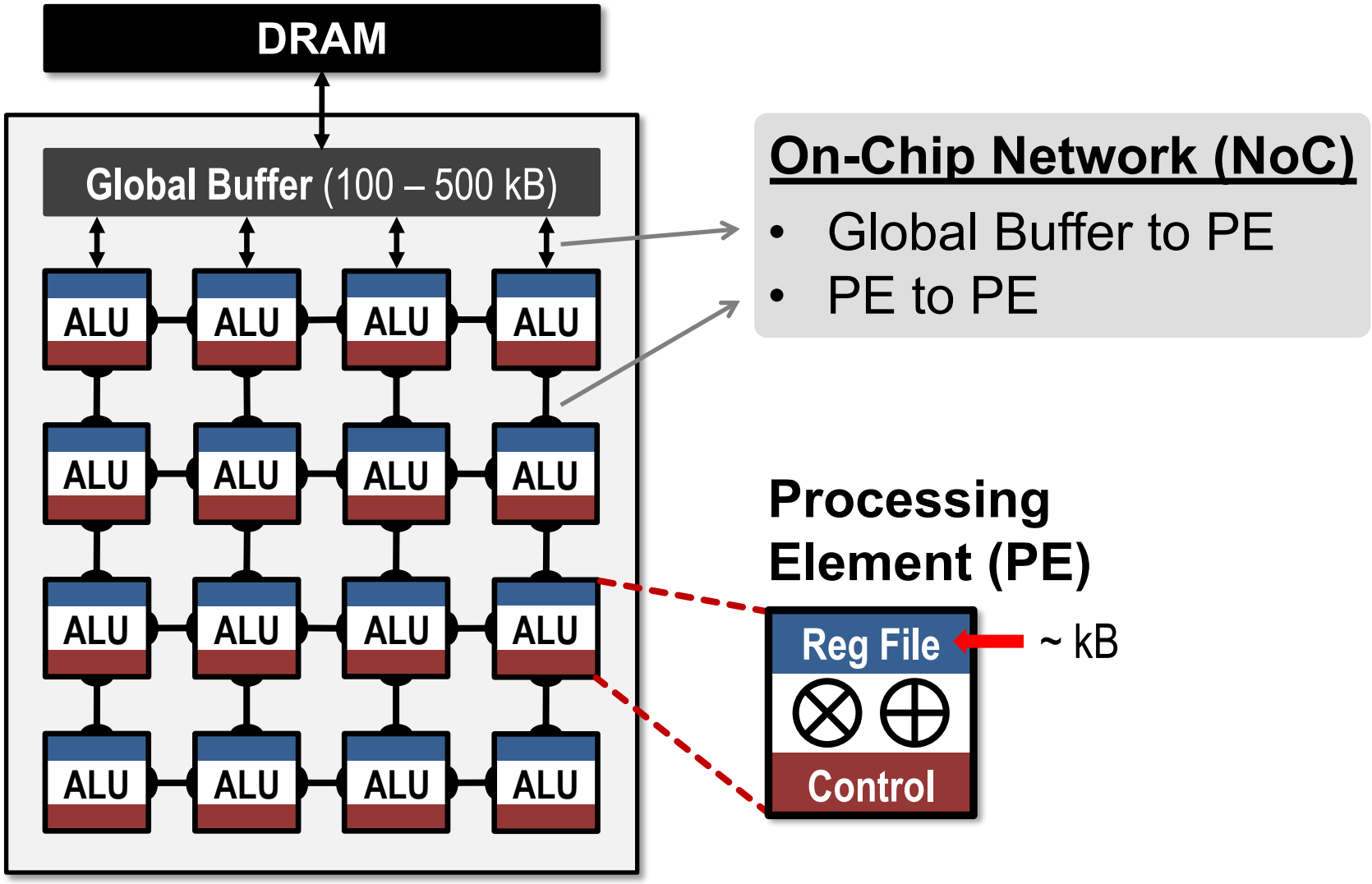
Leverage Parallelism for Higher Performance



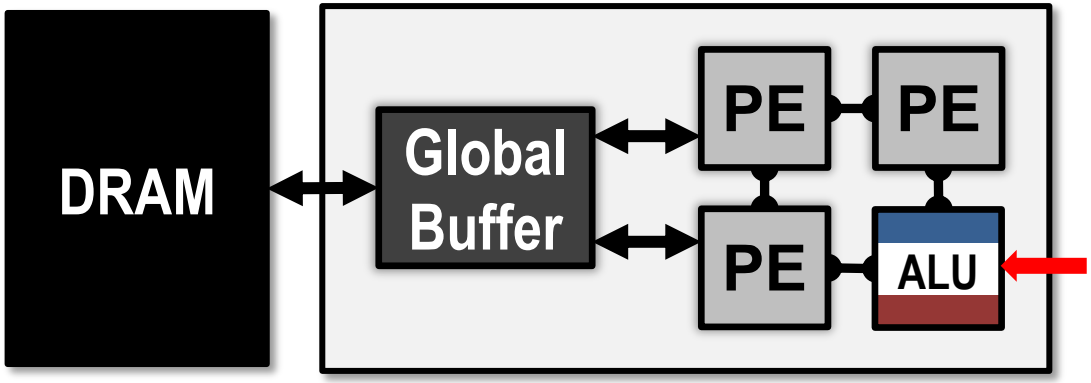
Leverage Parallelism for *Spatial* Data Reuse

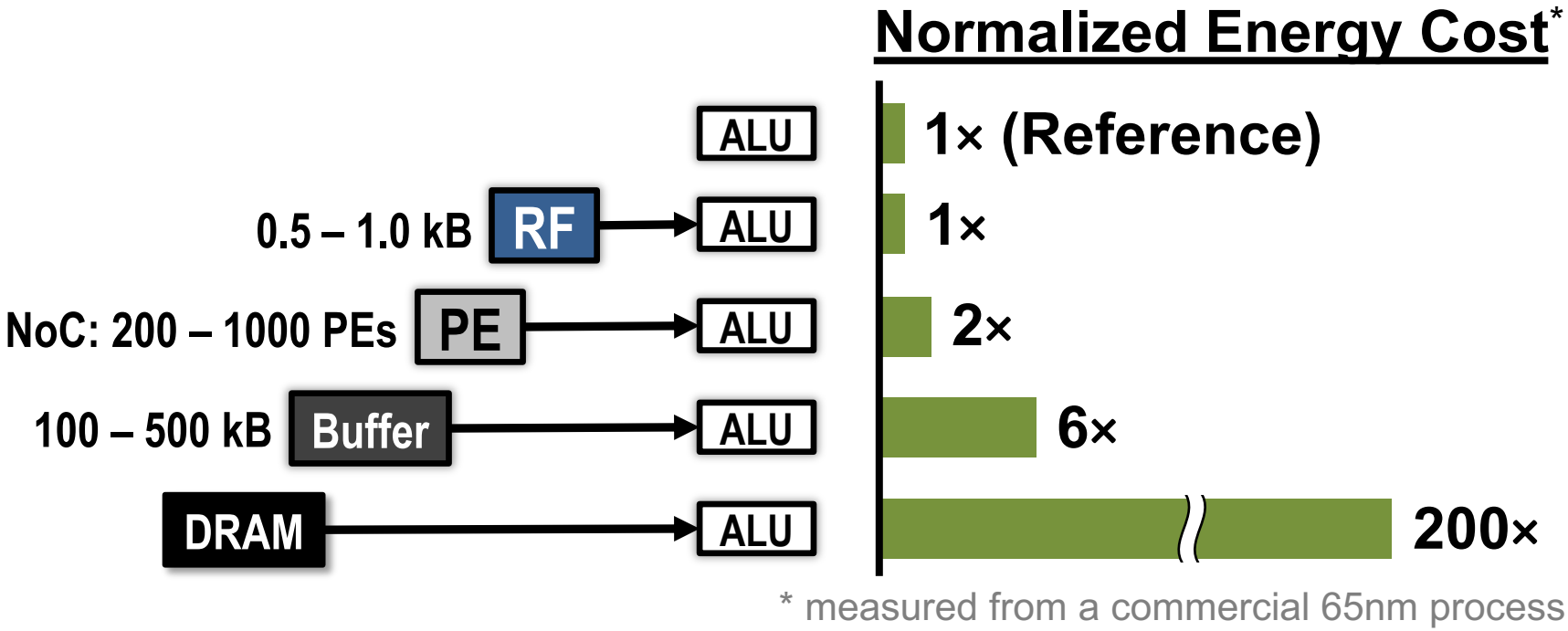


Spatial Architecture

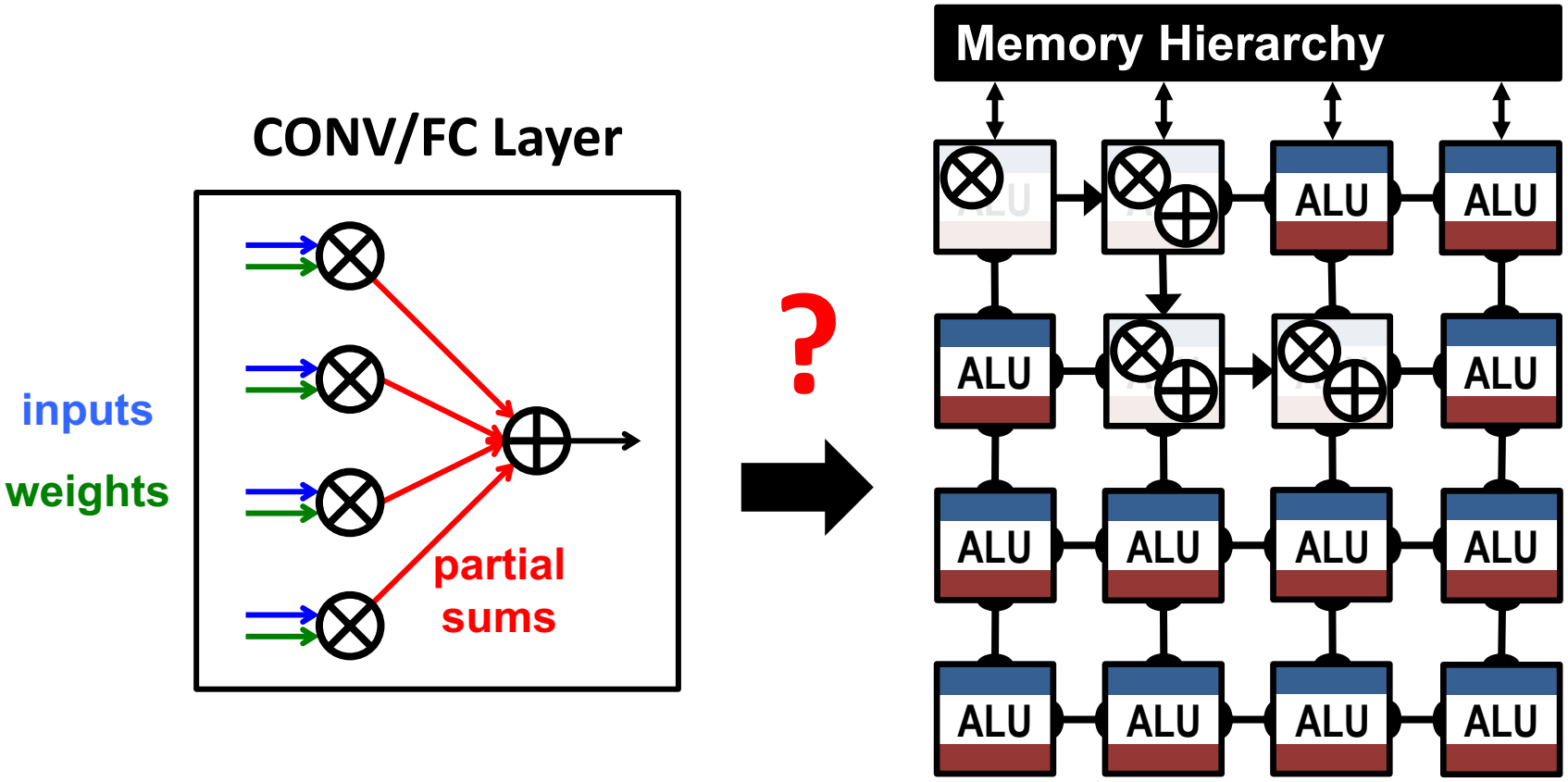


Low-Cost Local Data Access





How to Map the Dataflow?



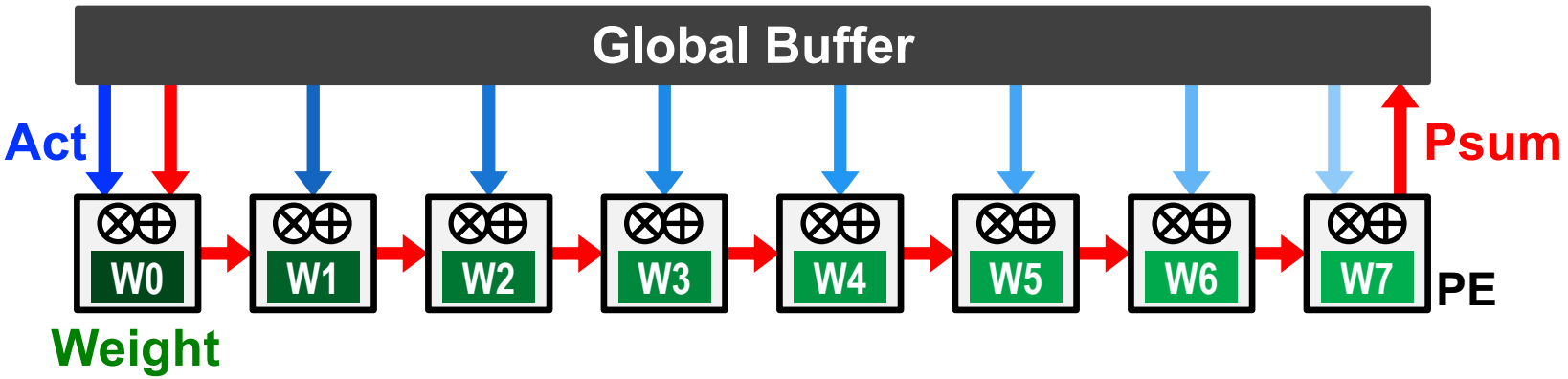
How to maximally exploit **data reuse** with the **low-cost memory hierarchy** and **parallelism?**

Dataflow Taxonomy

- Weight Stationary (WS)
- Output Stationary (OS)
- Input Stationary (IS)
- No Local Reuse (NLR)

[Chen et al., ISCA 2016]

Weight Stationary (WS) Dataflow



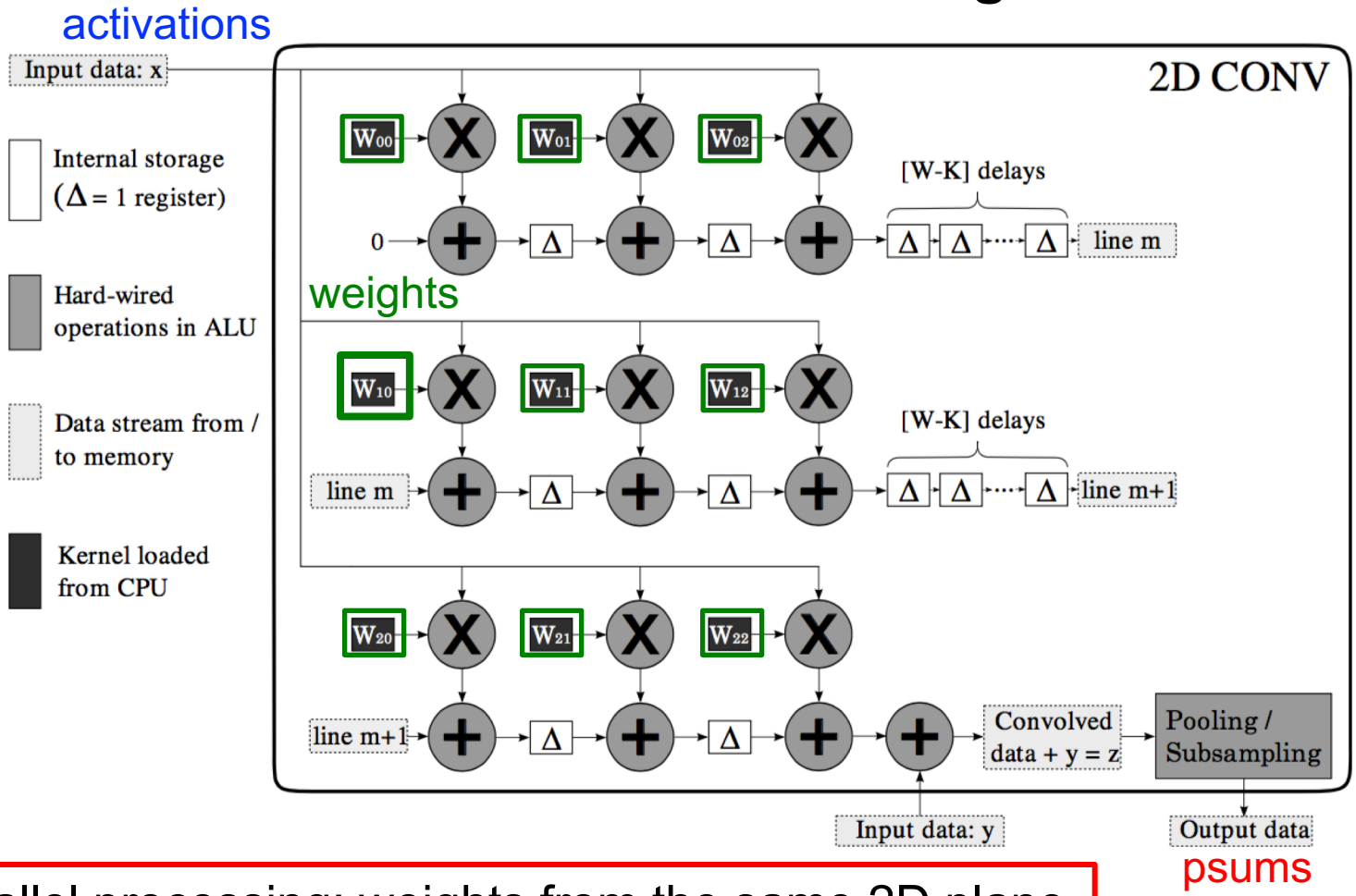
- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights
- **Examples:**

  [**nn-X (NeuFlow)**, *CVPRW* 2014] [**Park**, *ISSCC* 2015]

  [**Origami**, *GLSVLSI* 2015] [**Google TPU**, *ISCA* 2017]

WS Example: nn-X (NeuFlow)

A 3×3 2D Convolution Engine

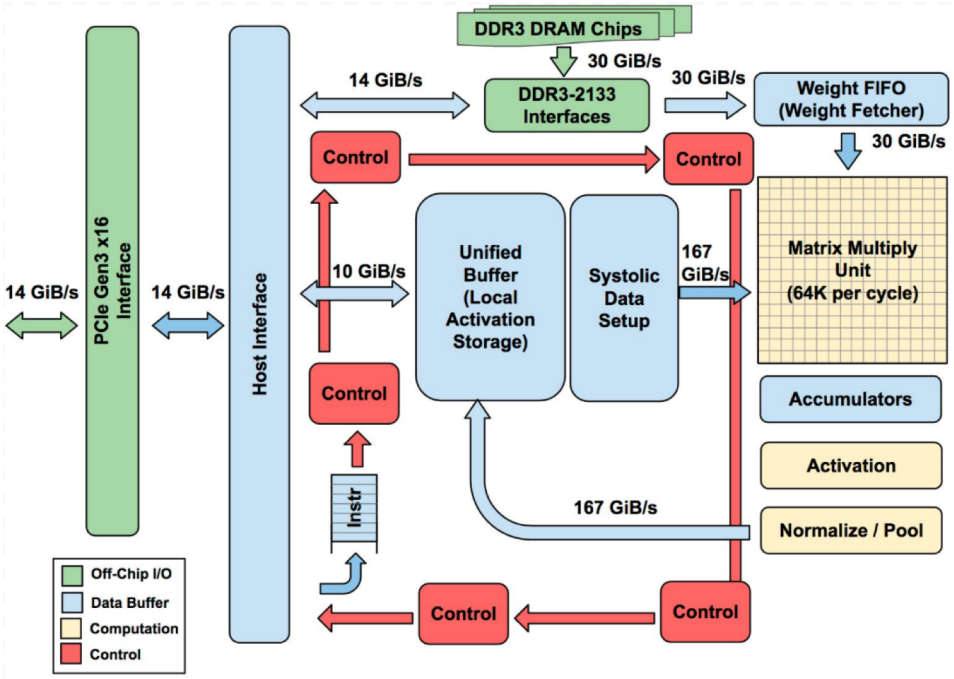


Parallel processing: weights from the same 2D plane

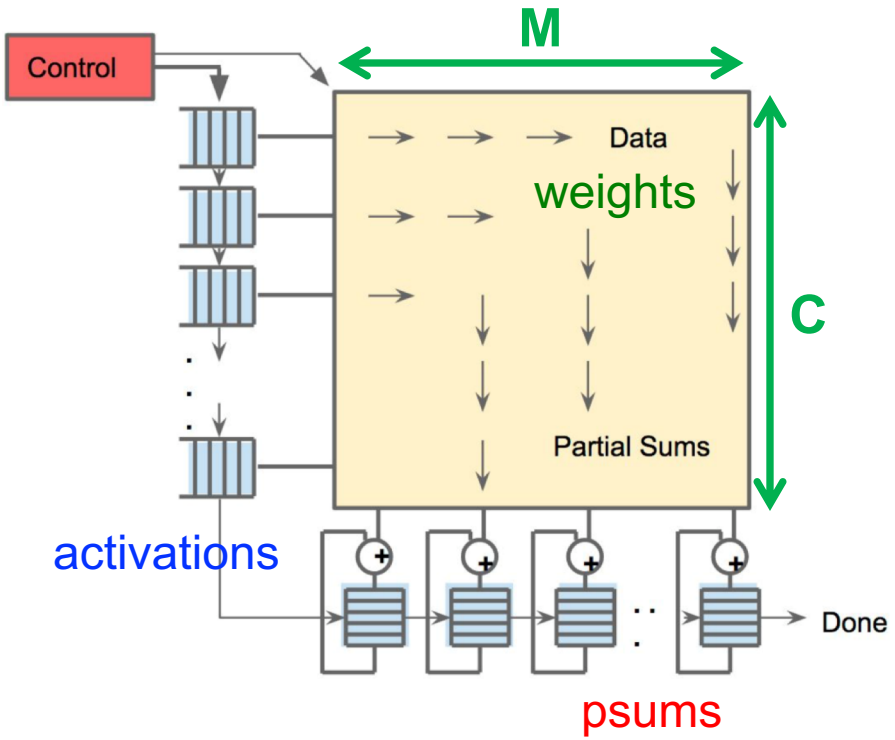
[Farabet et al., ICCV 2009]

WS Example: Google TPU

Top-Level Architecture



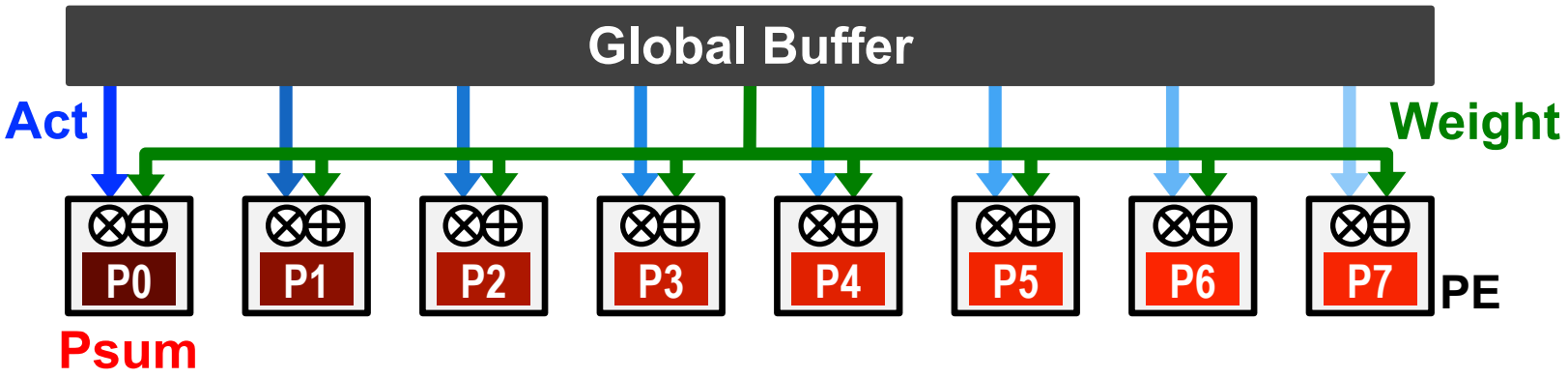
Matrix Multiply Unit



Parallel processing: weights from different 2D planes

[Jouppi et al., ISCA 2017]

Output Stationary (OS) Dataflow



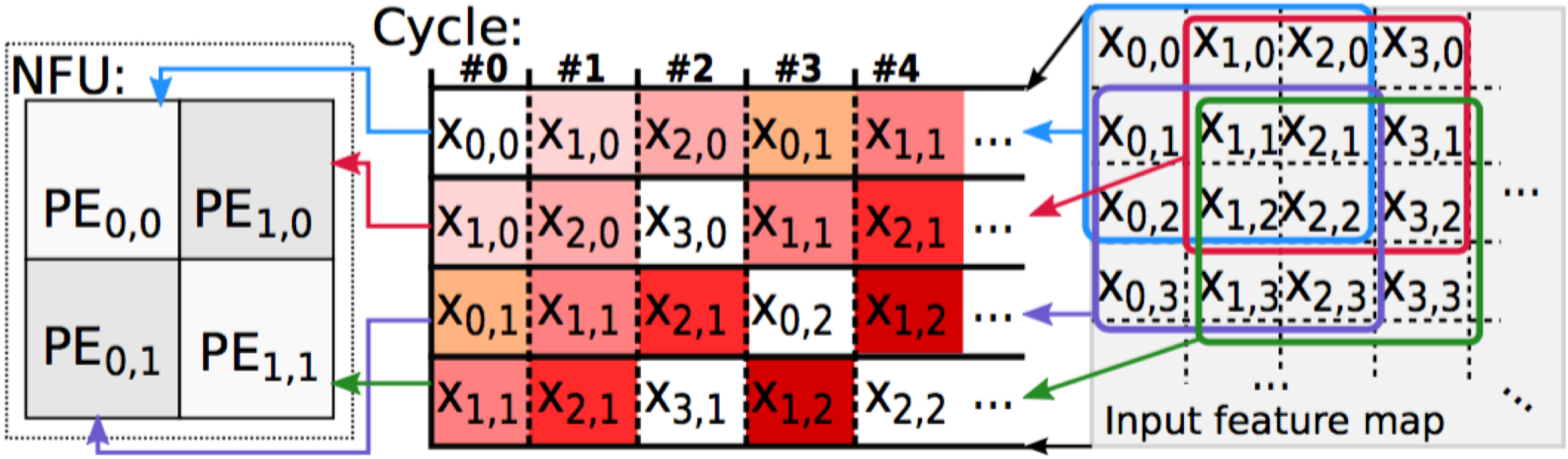
- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Examples:**

[**Gupta**, *ICML* 2015] [**ShiDianNao**, *ISCA* 2015]

[**ENVISION**, *ISSCC* 2017] [**Thinker**, *JSSC* 2017]

OS Example: ShiDianNao

Dataflow in the PE Array

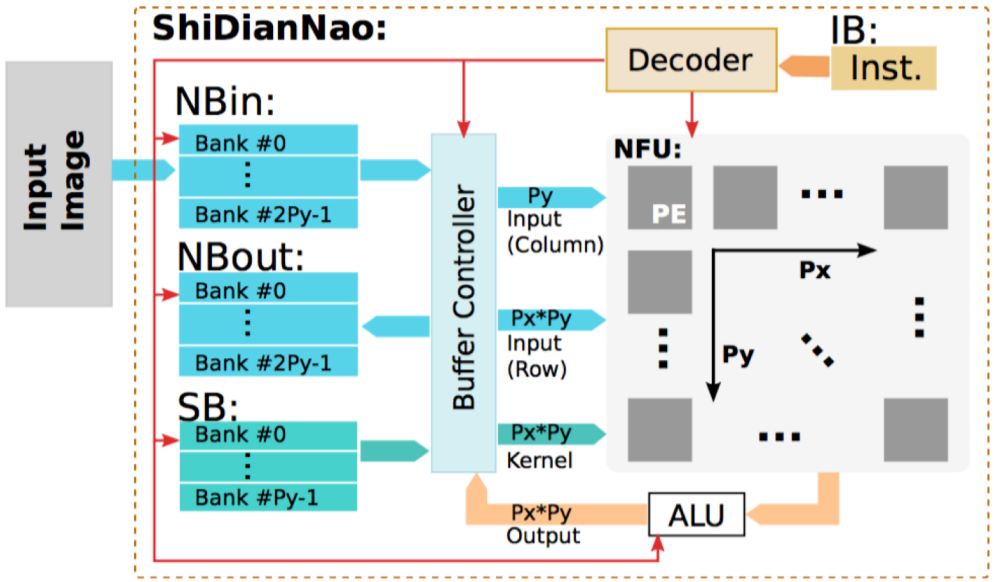


Parallel processing: many output activations from the same 2D plane

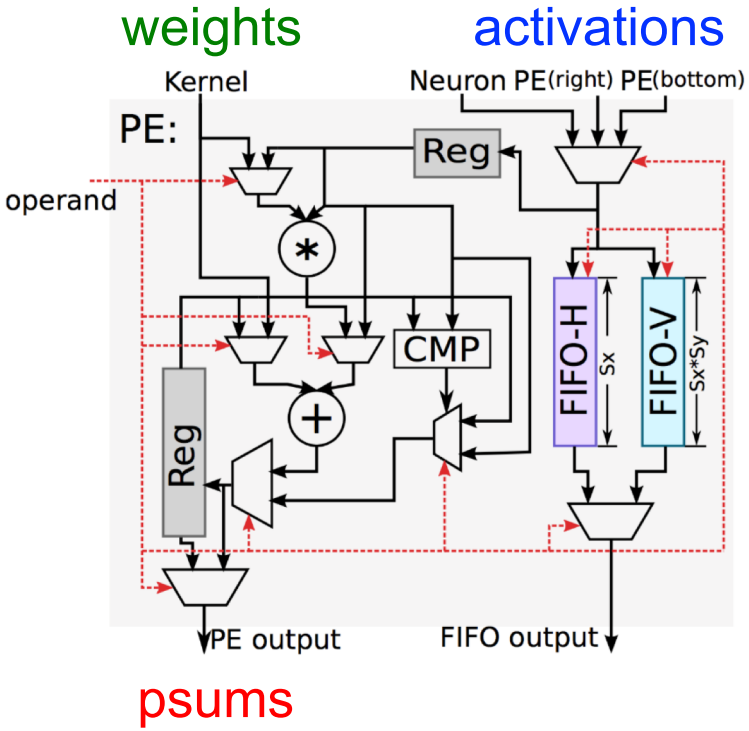
[Du et al., ISCA 2015]

OS Example: ShiDianNao

Top-Level Architecture

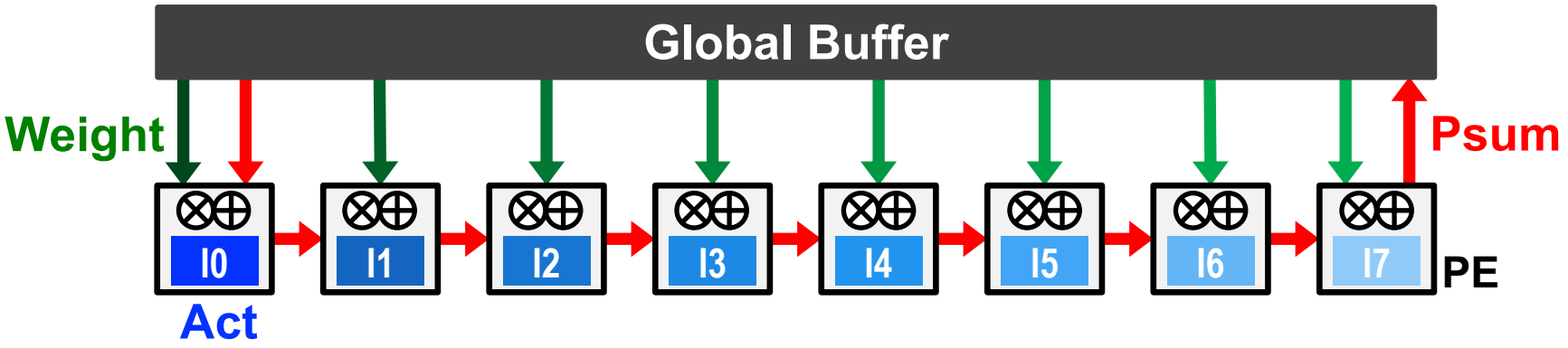


PE Architecture



[Du et al., ISCA 2015]

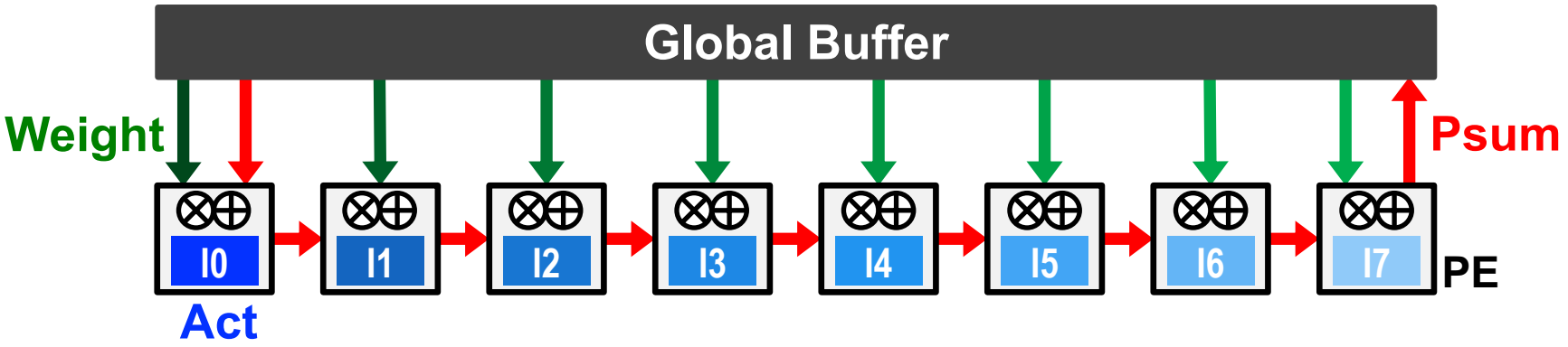
Input Stationary (IS) Dataflow



- **Minimize Input Act.** read energy consumption
  - maximize convolutional and fmap reuse of acts
- **Examples:**

  [**SCNN**, *ISCA* 2017]

Input Stationary (IS) Dataflow



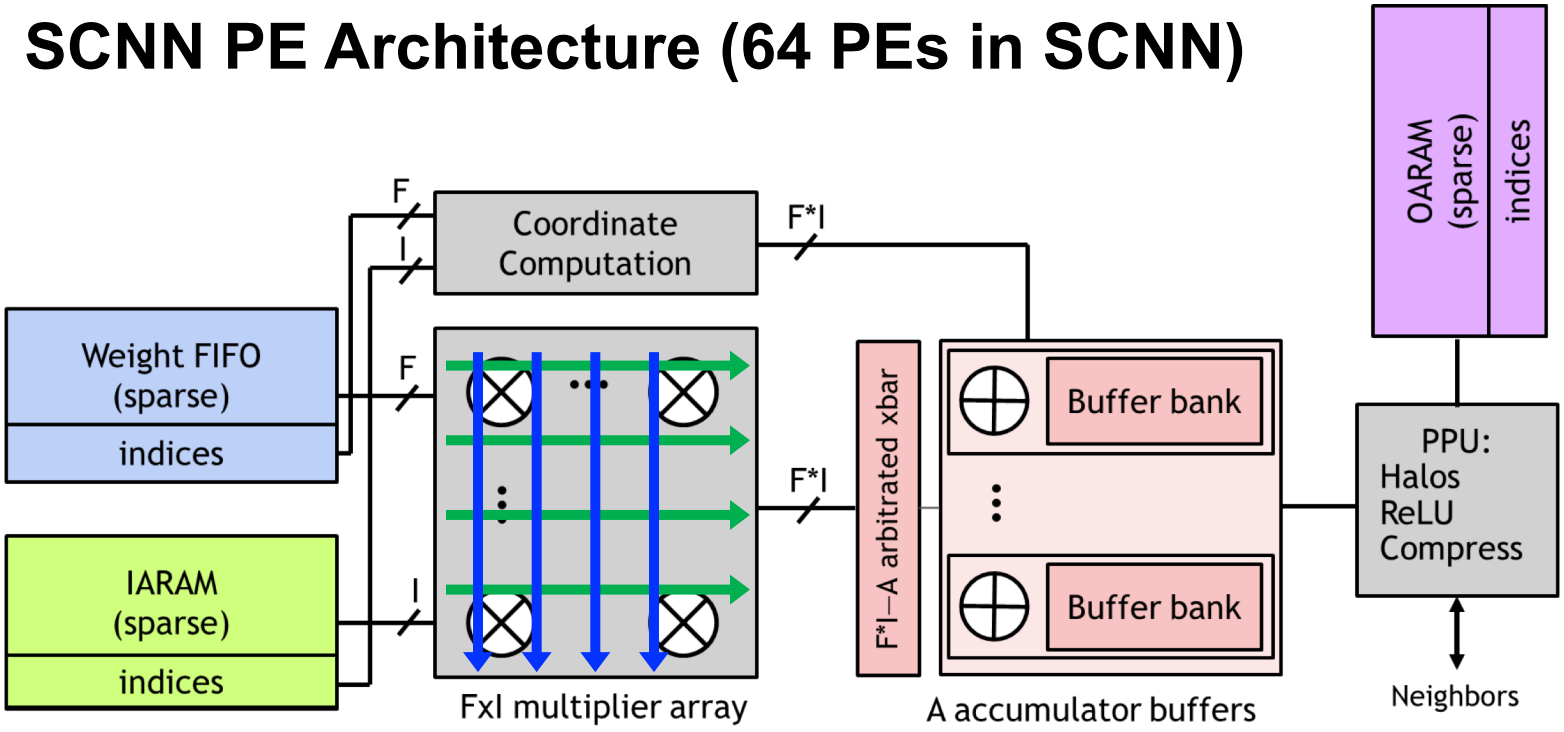
- **Minimize Input Act.** read energy consumption
  - maximize convolutional and fmap reuse of acts
- **Examples:**

  [**SCNN**, *ISCA* 2017]

Why is it not as popular?

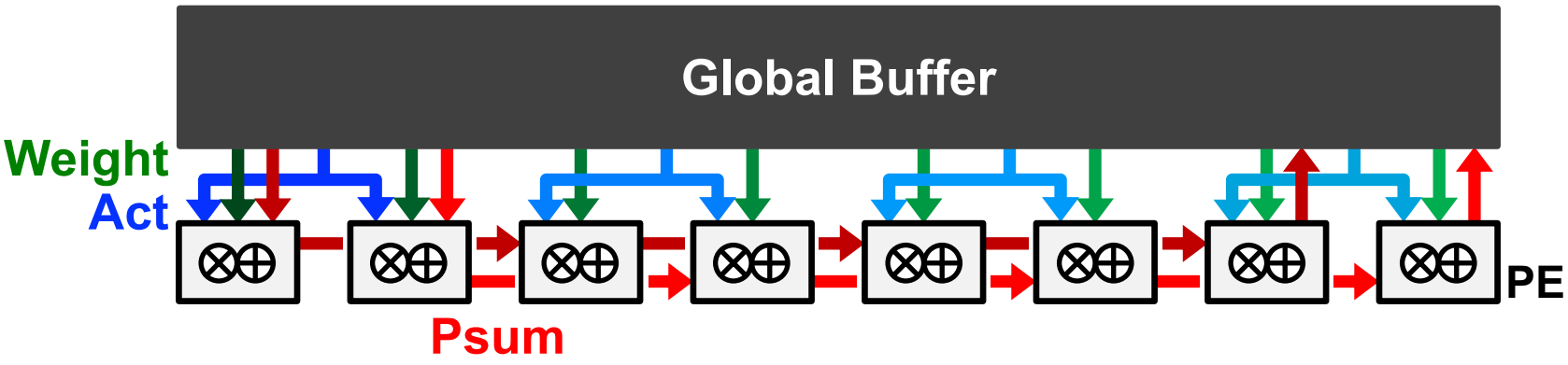
IS Example: SCNN

SCNN PE Architecture (64 PEs in SCNN)



Parallel processing: many input activations from the same 2D plane

No Local Reuse (NLR) Dataflow

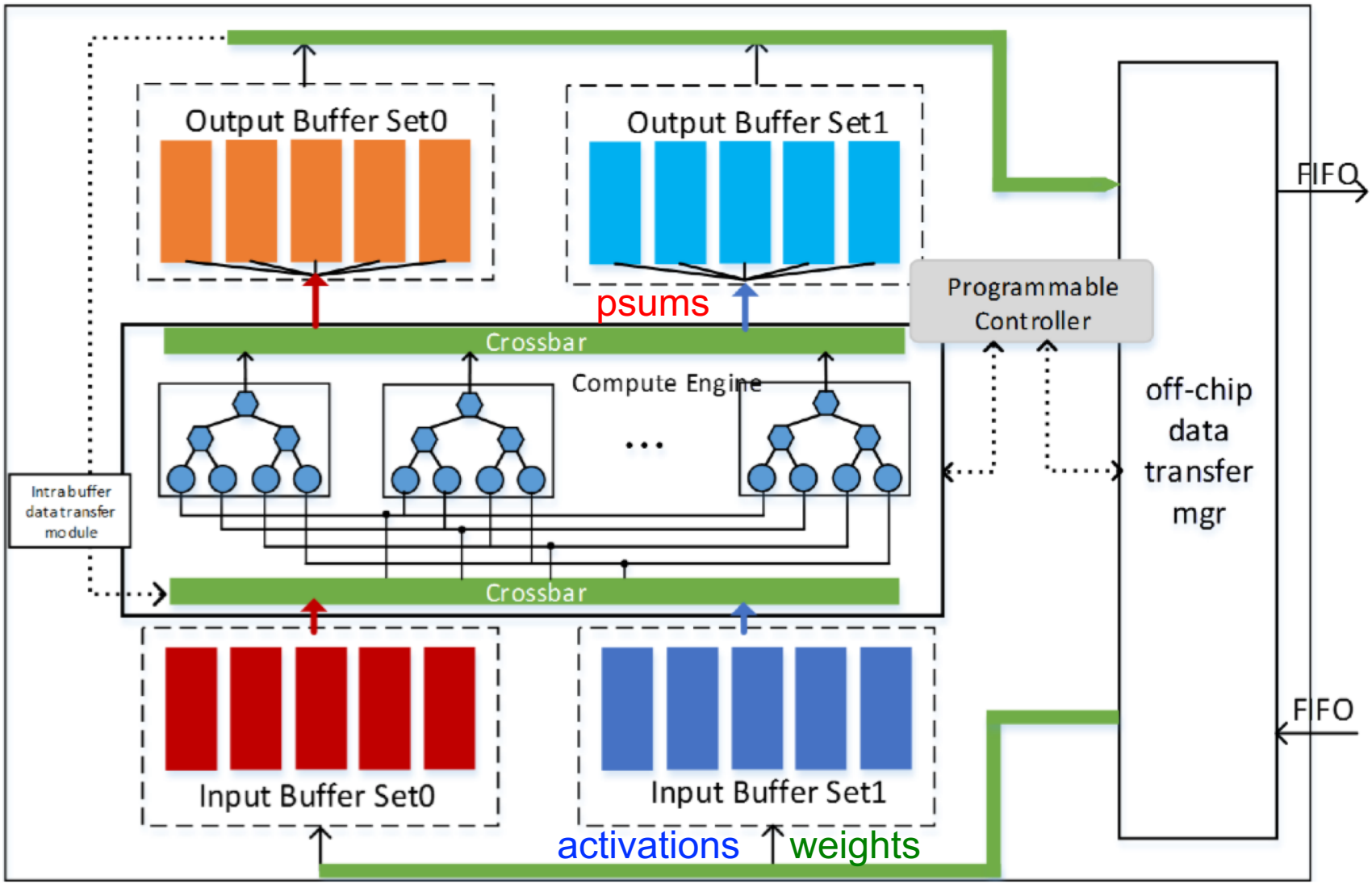


- Use a **large global buffer** as shared storage
  - Reduce **DRAM** access energy consumption
- **Examples:**

  [**DianNao**, *ASPLOS* 2014] [**DaDianNao**, *MICRO* 2014]

  [**Zhang**, *FPGA* 2015]

NLR Example: UCLA



[Zhang et al., FPGA 2015]

Dataflow Taxonomy Summary

- **Weight Stationary**
  - minimize movement of **weights**
- **Output Stationary**
  - minimize movement of partial sums
- **Input Stationary**
  - minimize movement of input activations
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**

Dataflow Taxonomy Summary

- **Weight Stationary**
  - minimize movement of **weights**
- **Output Stationary**
  - minimize movement of partial sums
- **Input Stationary**
  - minimize movement of input activations
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**

Does this define the space of all possible dataflows?

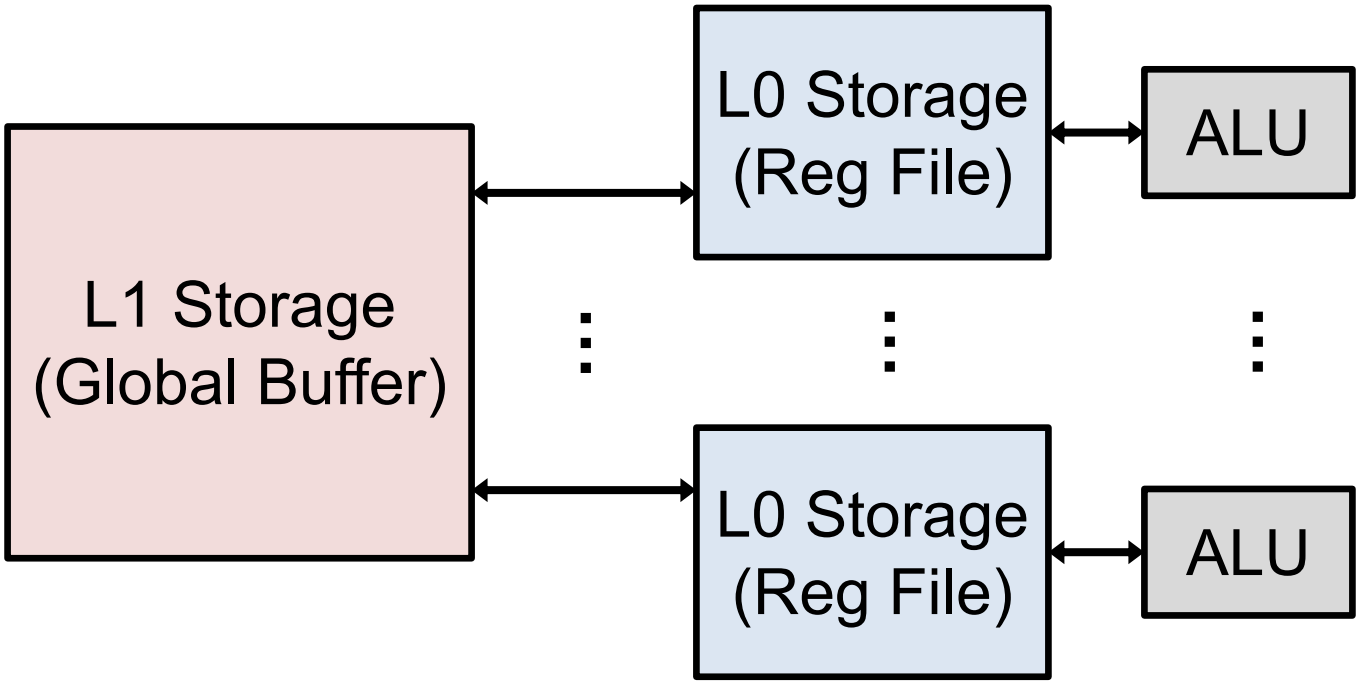
Dataflow Taxonomy Summary

- **Weight Stationary**
  - minimize movement of **weights**
- **Output Stationary**
  - minimize movement of partial sums
- **Input Stationary**
  - minimize movement of input activations
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**

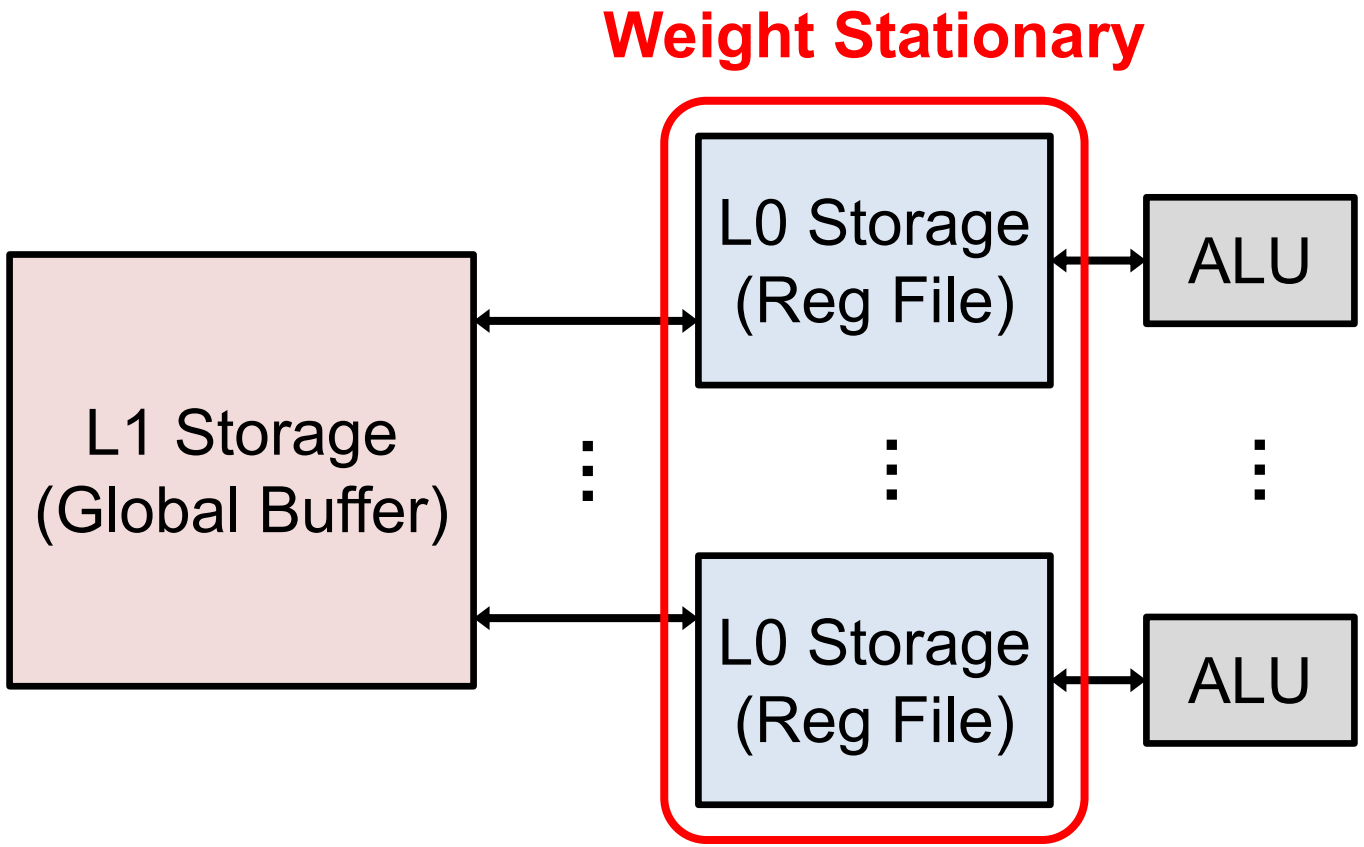
Does this define the space of all possible dataflows? Not at all!

Can we have multiple ***stationariness*** in an architecture?

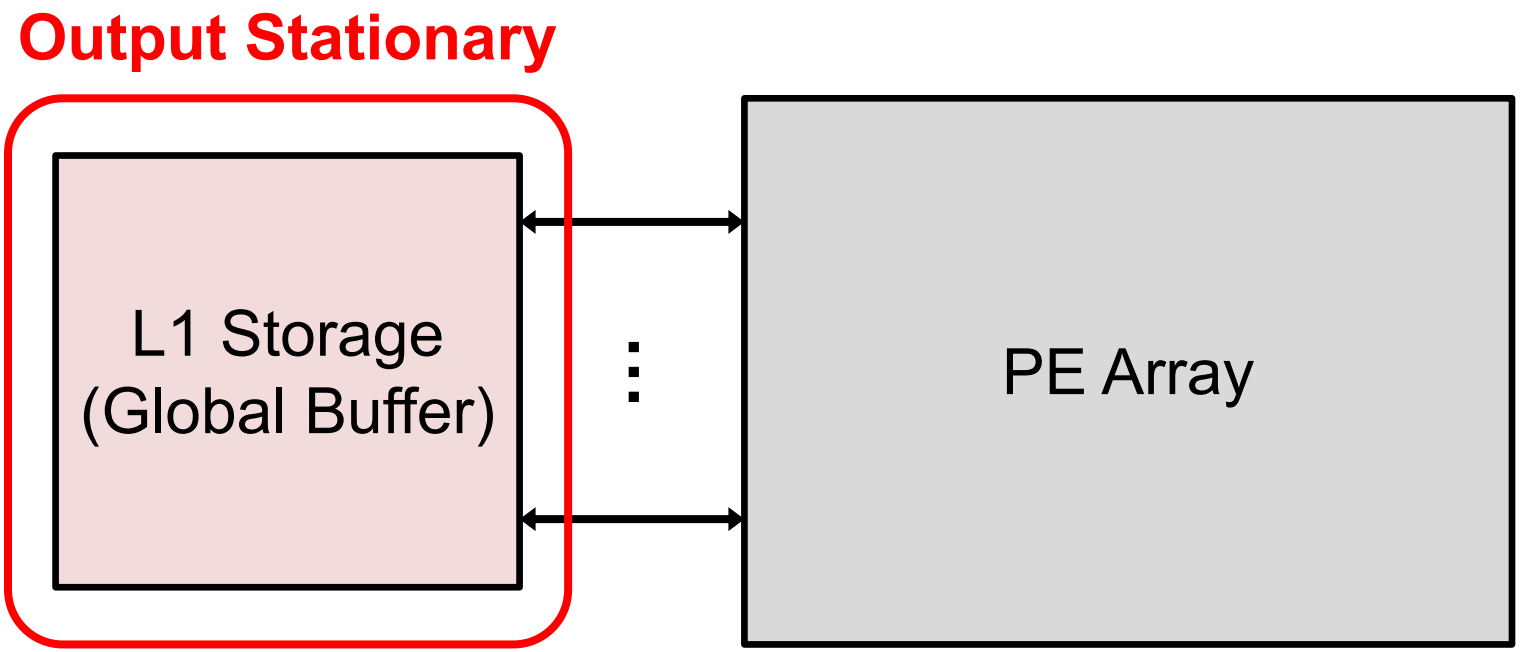
Multi-level Storage Hierarchy vs. Dataflow



Multi-level Storage Hierarchy vs. Dataflow

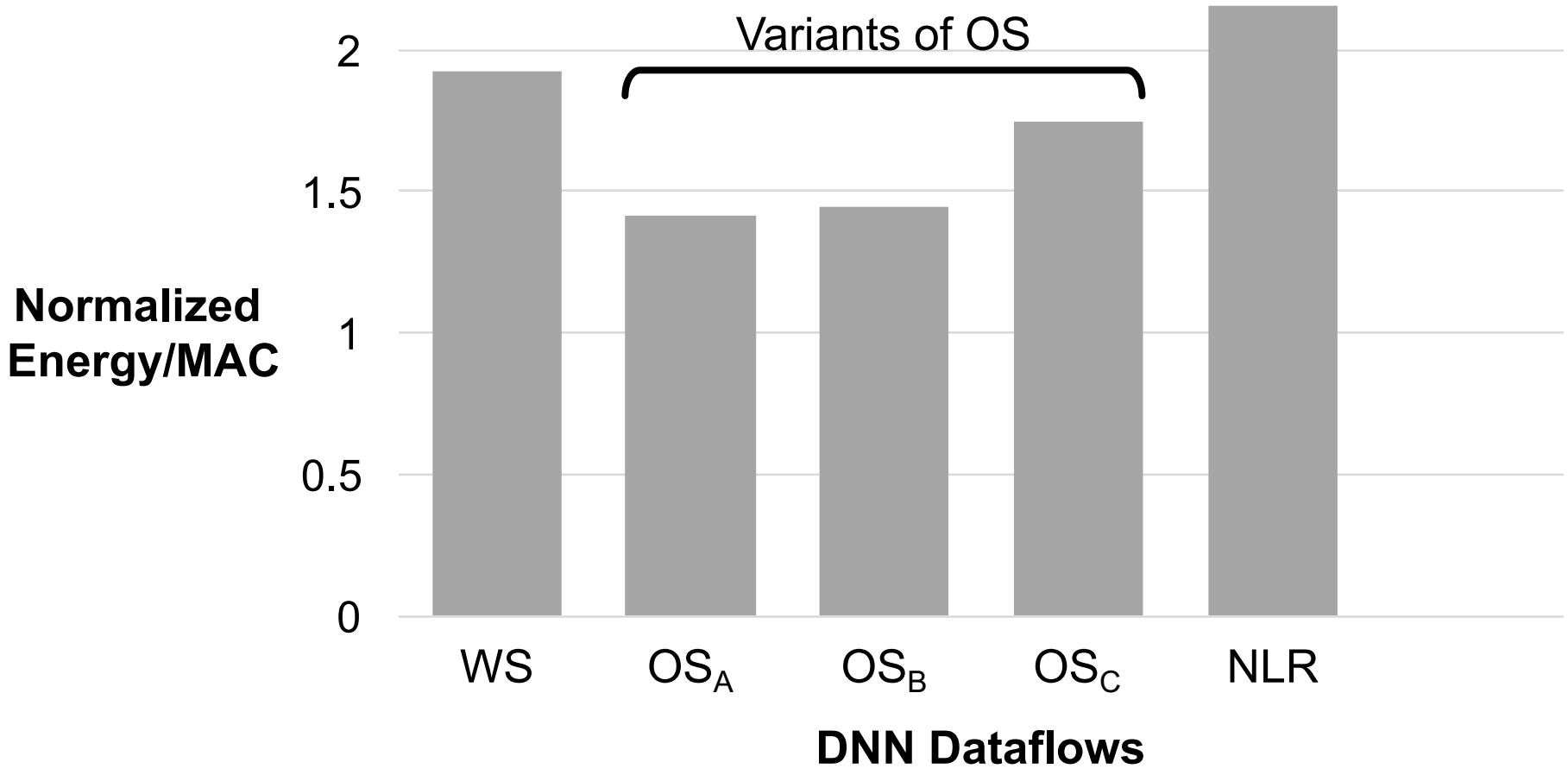


Multi-level Storage Hierarchy vs. Dataflow



Energy Efficiency Comparison

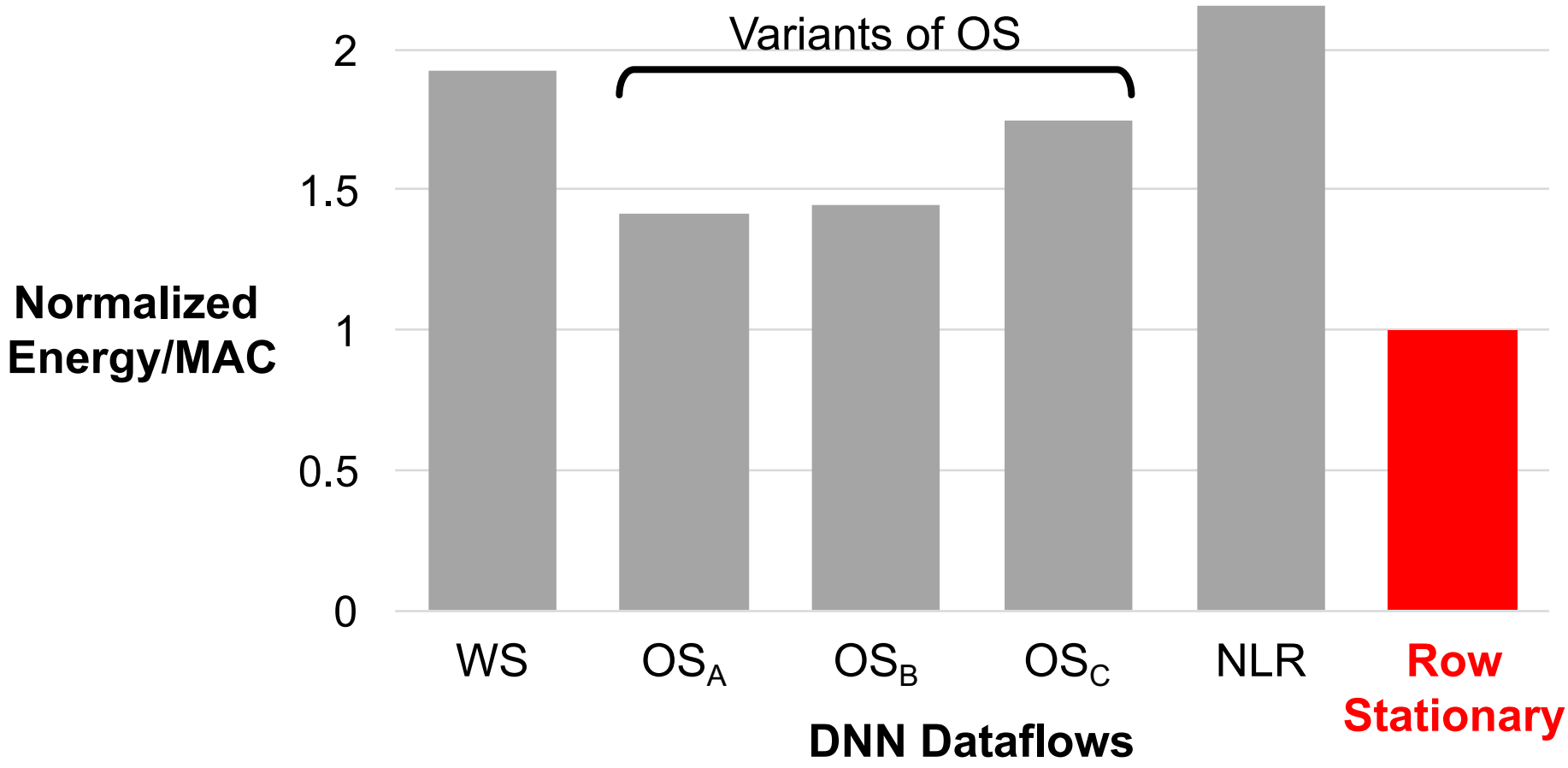
- Same total area
- AlexNet CONV layers
- 256 PEs
- Batch size = 16



[Chen et al., ISCA 2016]

Energy Efficiency Comparison

- Same total area
- AlexNet CONV layers
- 256 PEs
- Batch size = 16



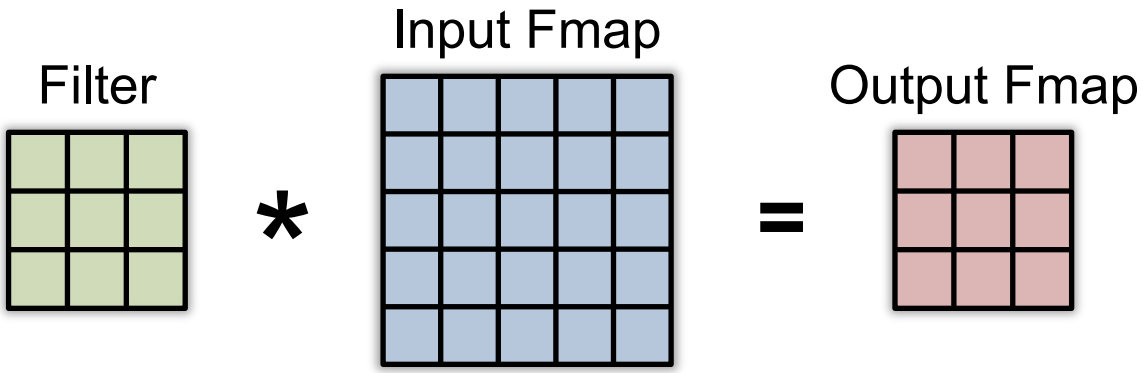
[Chen et al., ISCA 2016]

Energy-Efficient Dataflow: Row Stationary (RS)

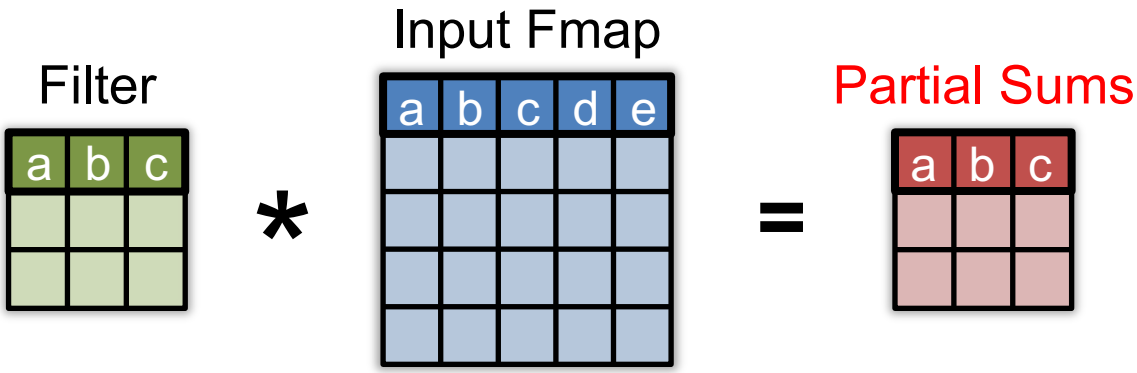
- **Maximize** reuse and accumulation at **RF**
- Optimize for **overall** energy efficiency instead for *only* a certain data type

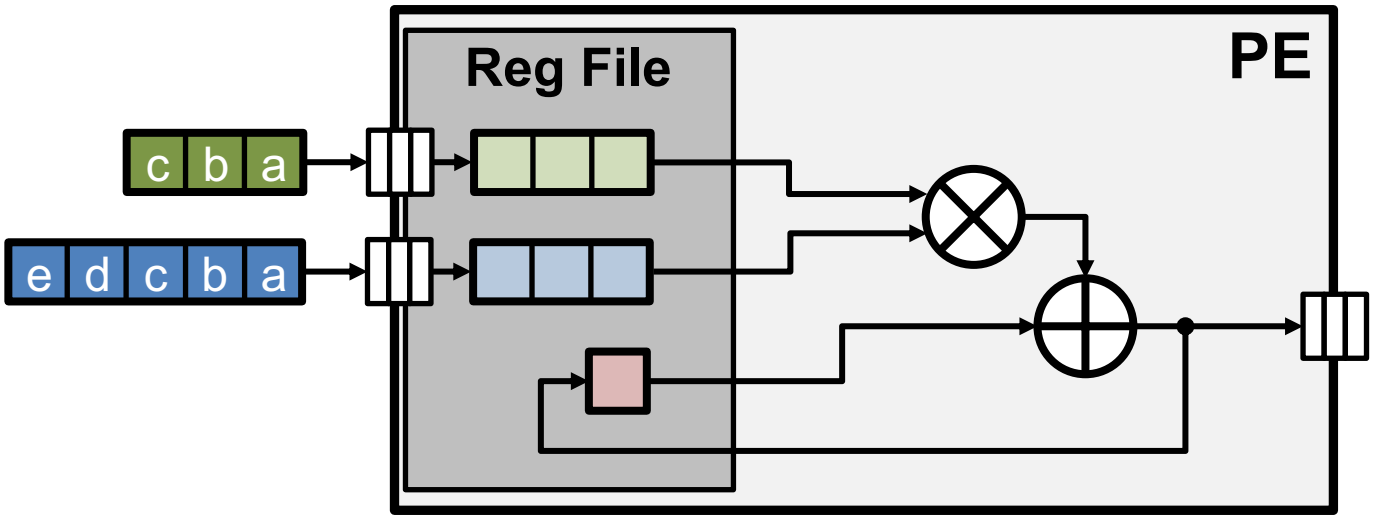
[Chen et al., ISCA 2016]

1D Row Convolution in PE

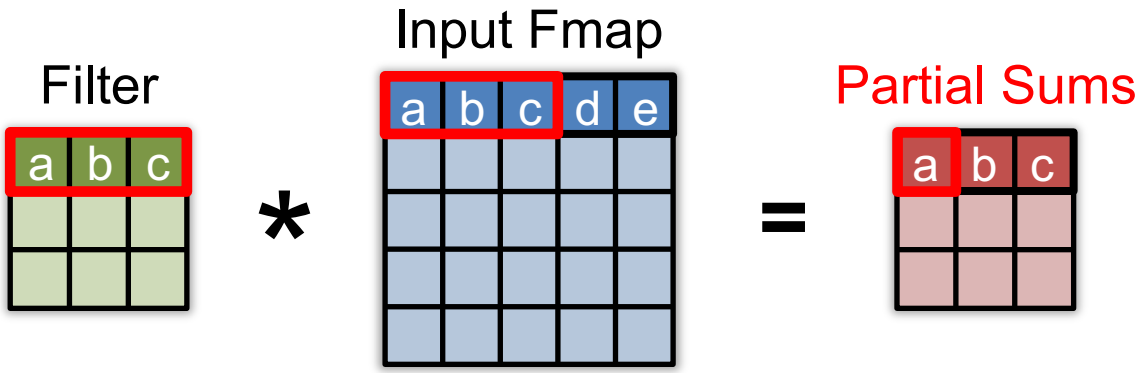


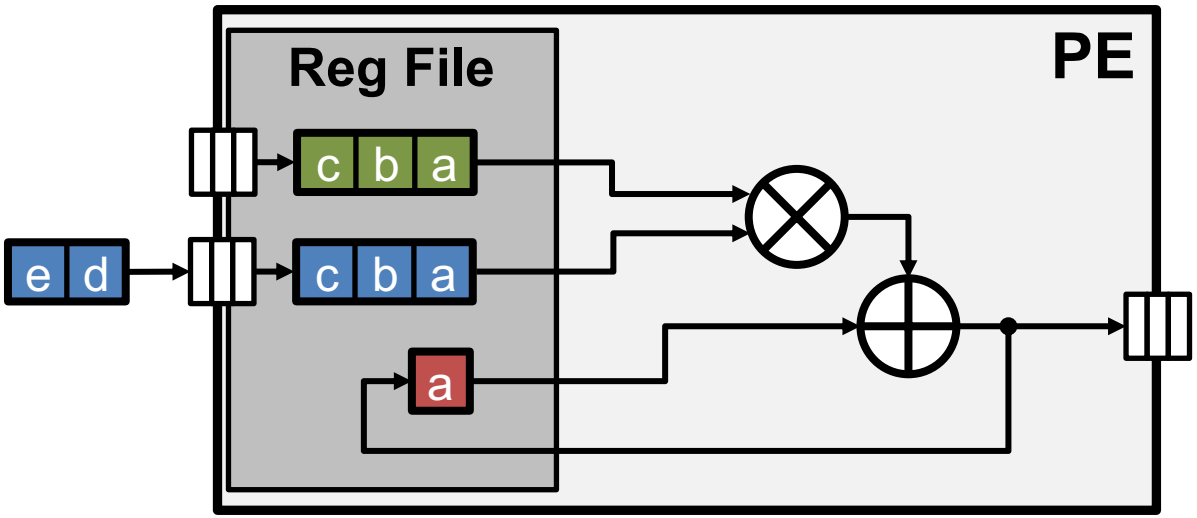
1D Row Convolution in PE



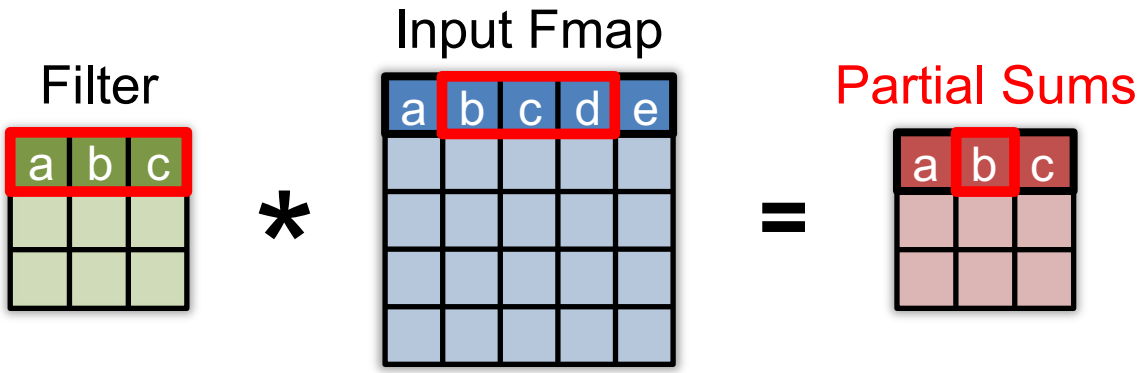


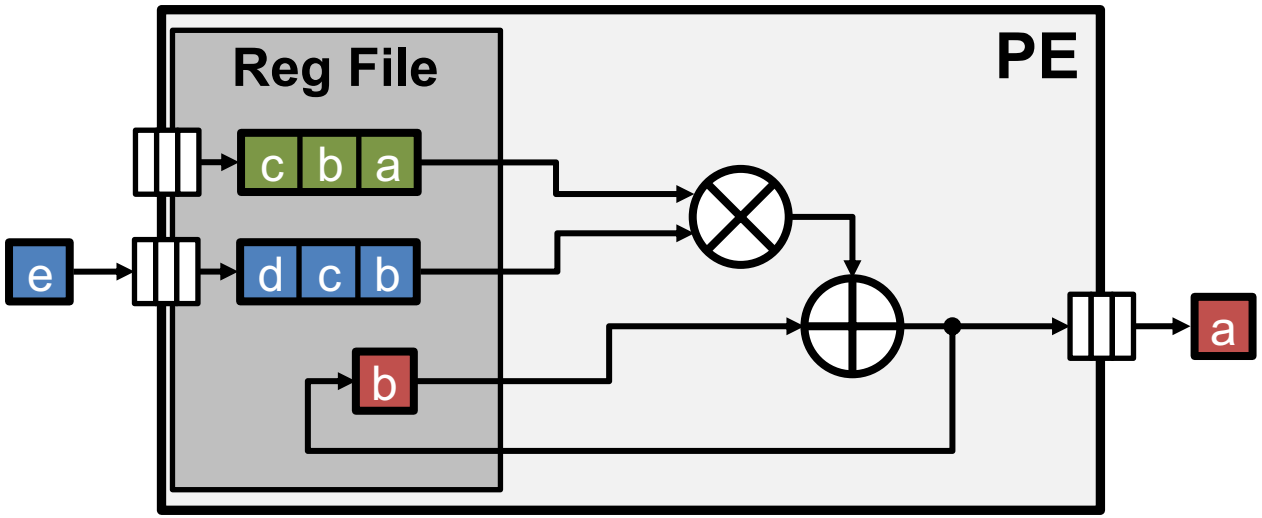
1D Row Convolution in PE



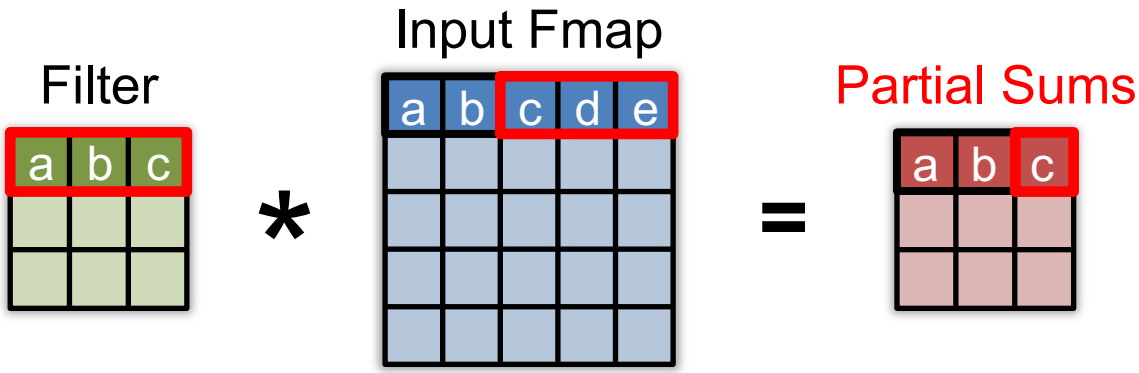


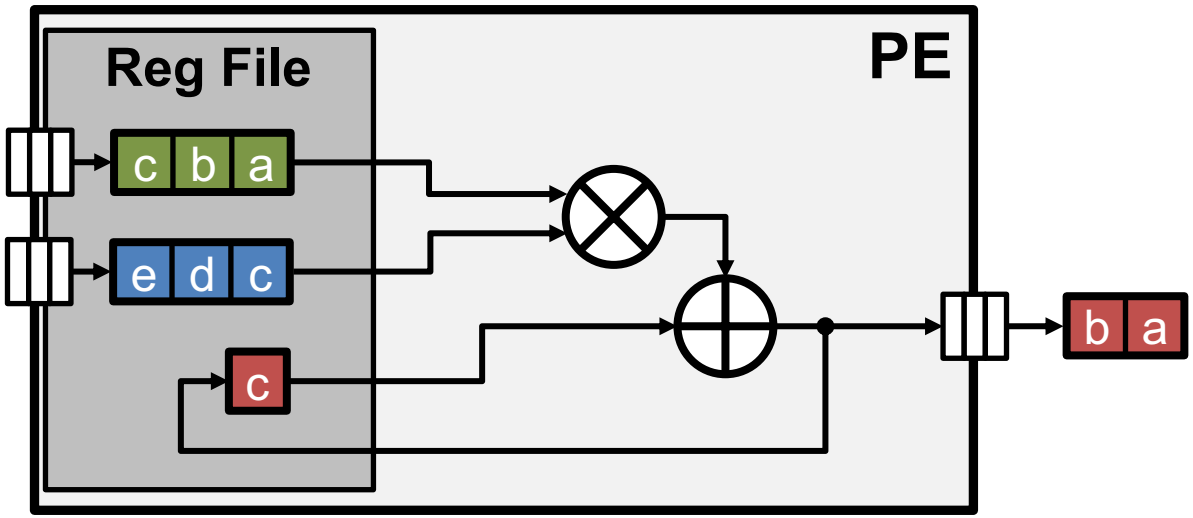
1D Row Convolution in PE





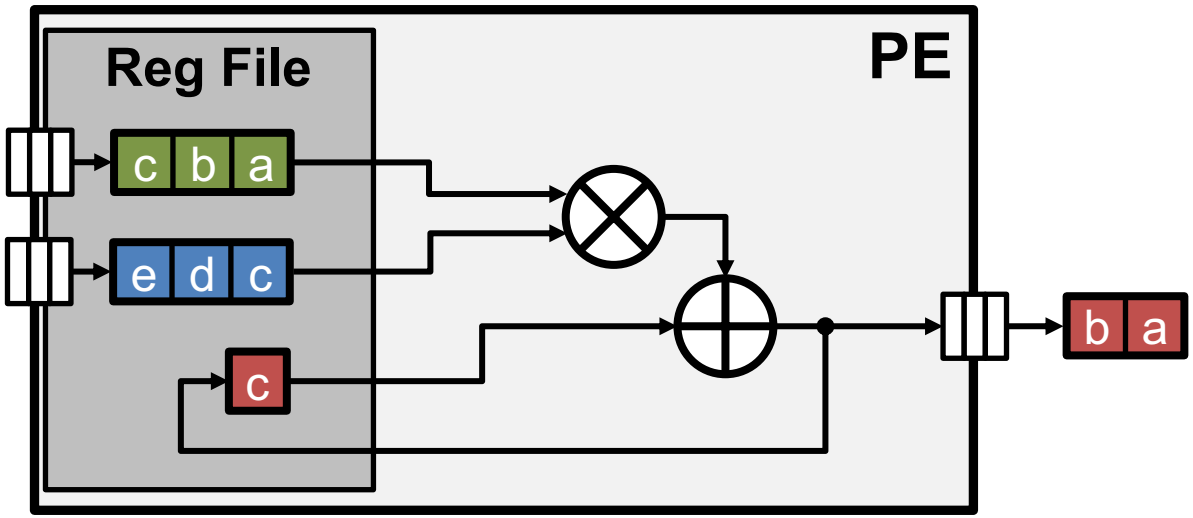
1D Row Convolution in PE



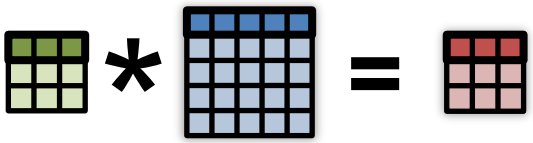


1D Row Convolution in PE

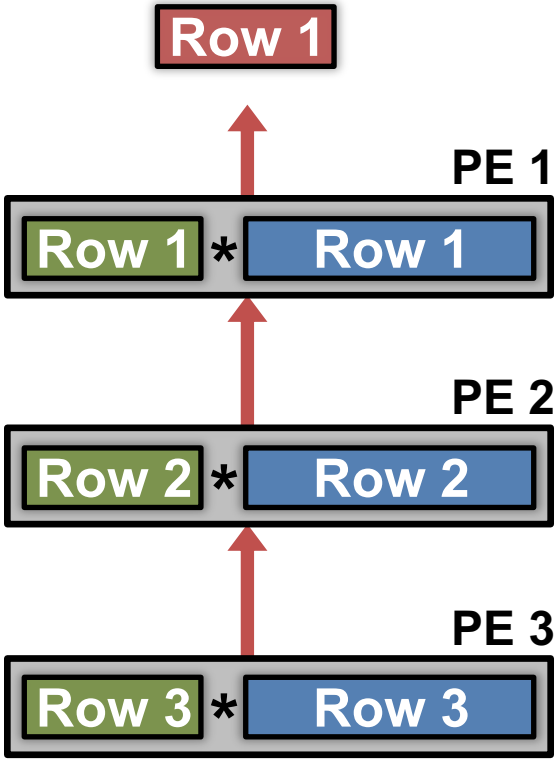
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF

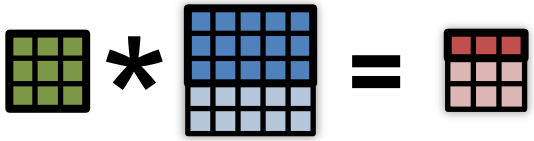


2D Convolution in PE Array

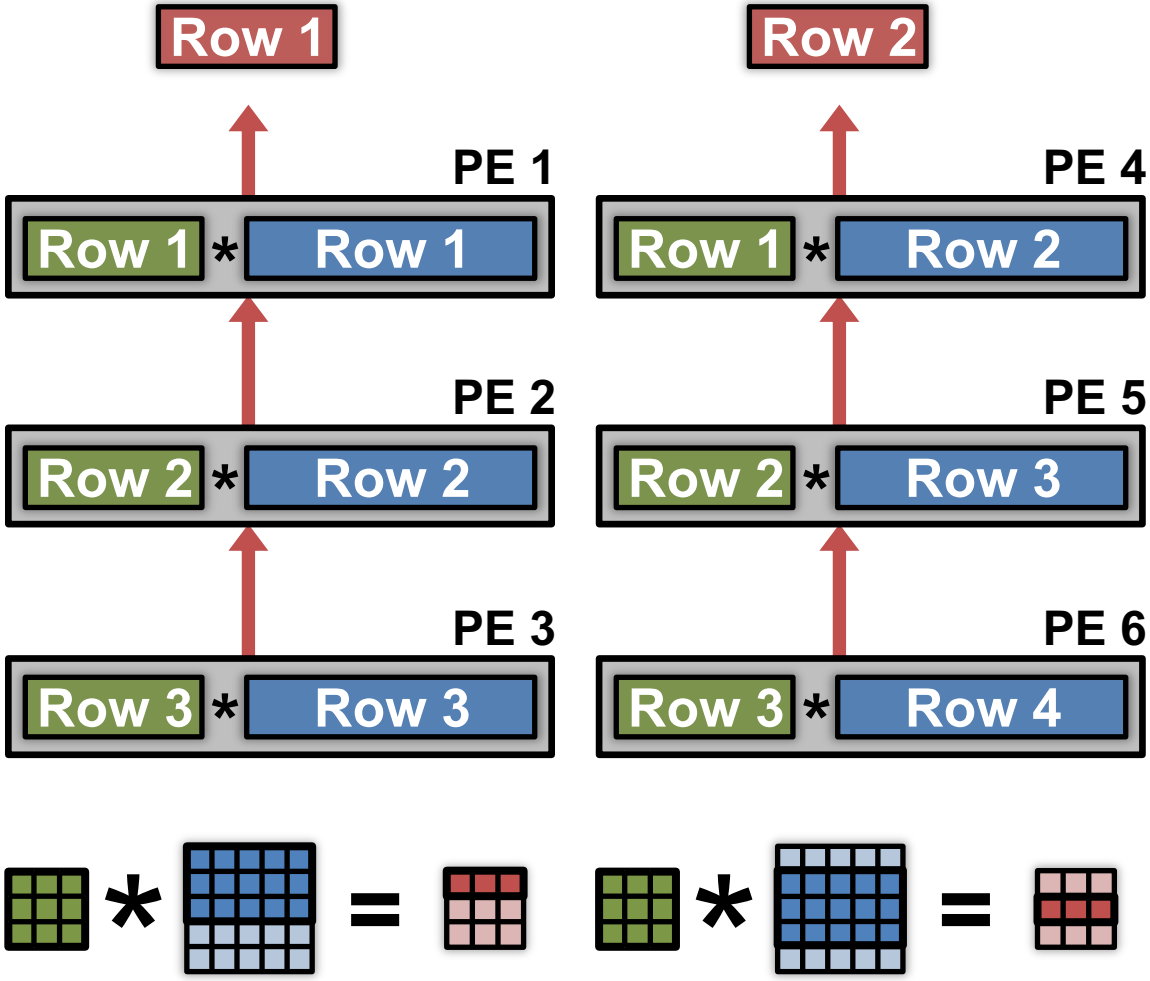


2D Convolution in PE Array

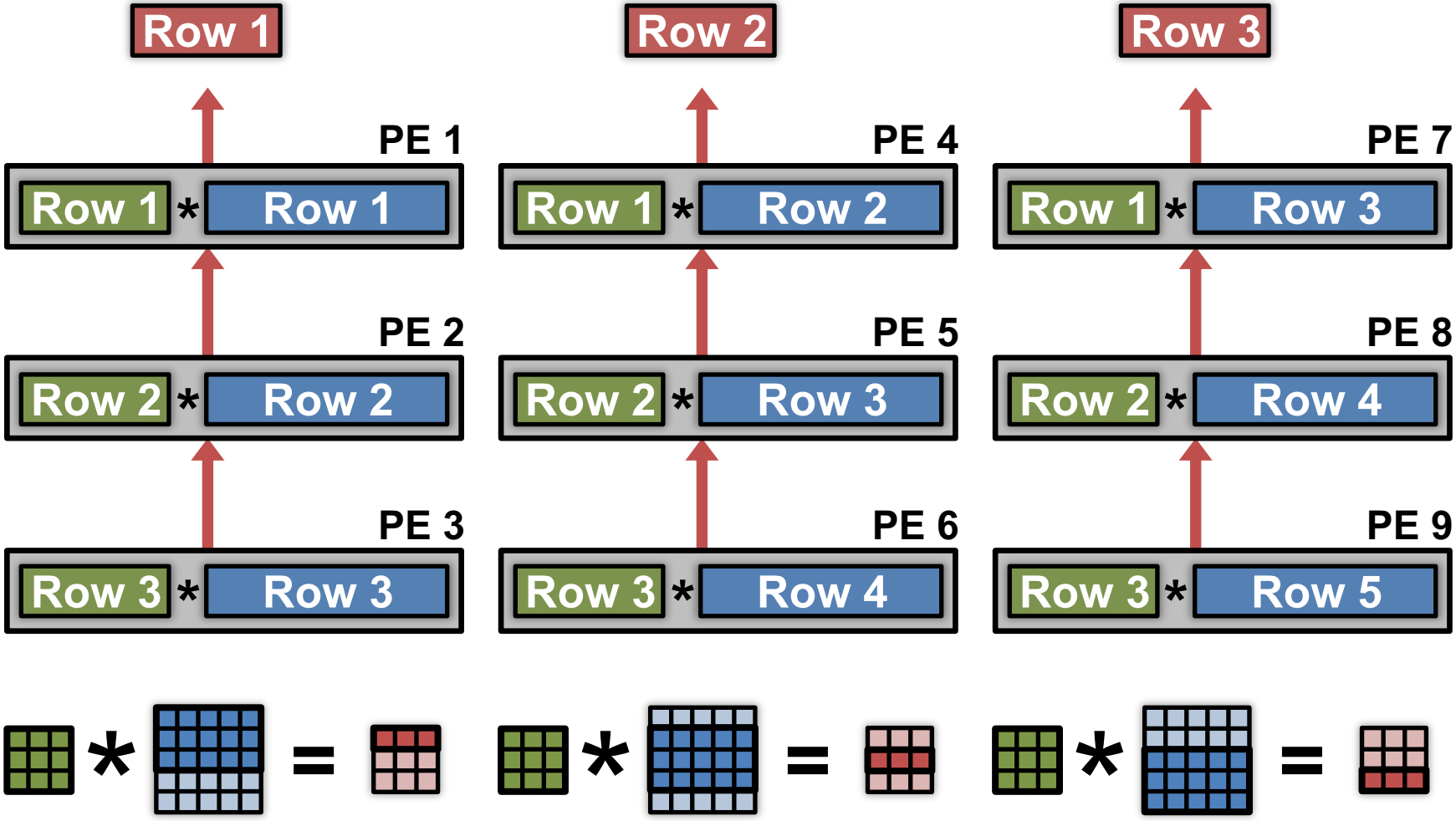




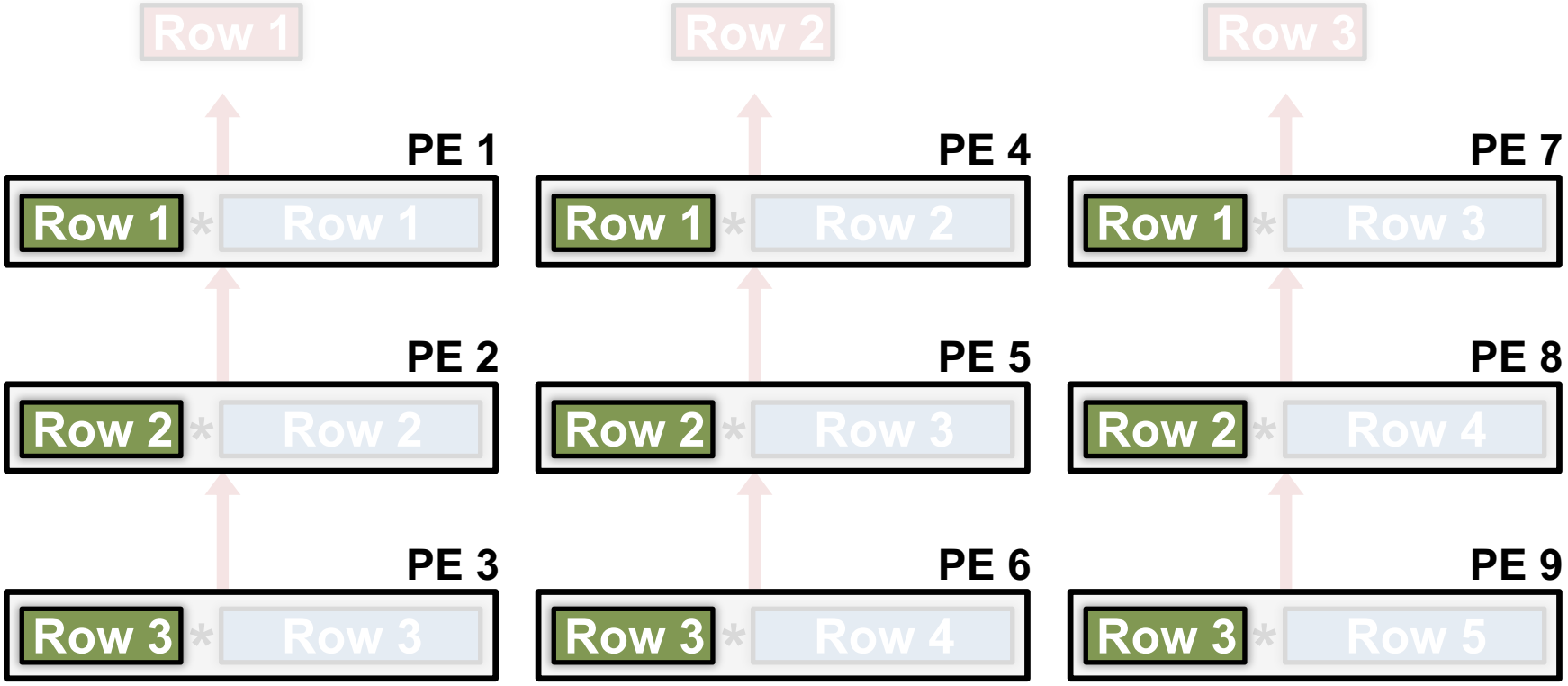
2D Convolution in PE Array



2D Convolution in PE Array

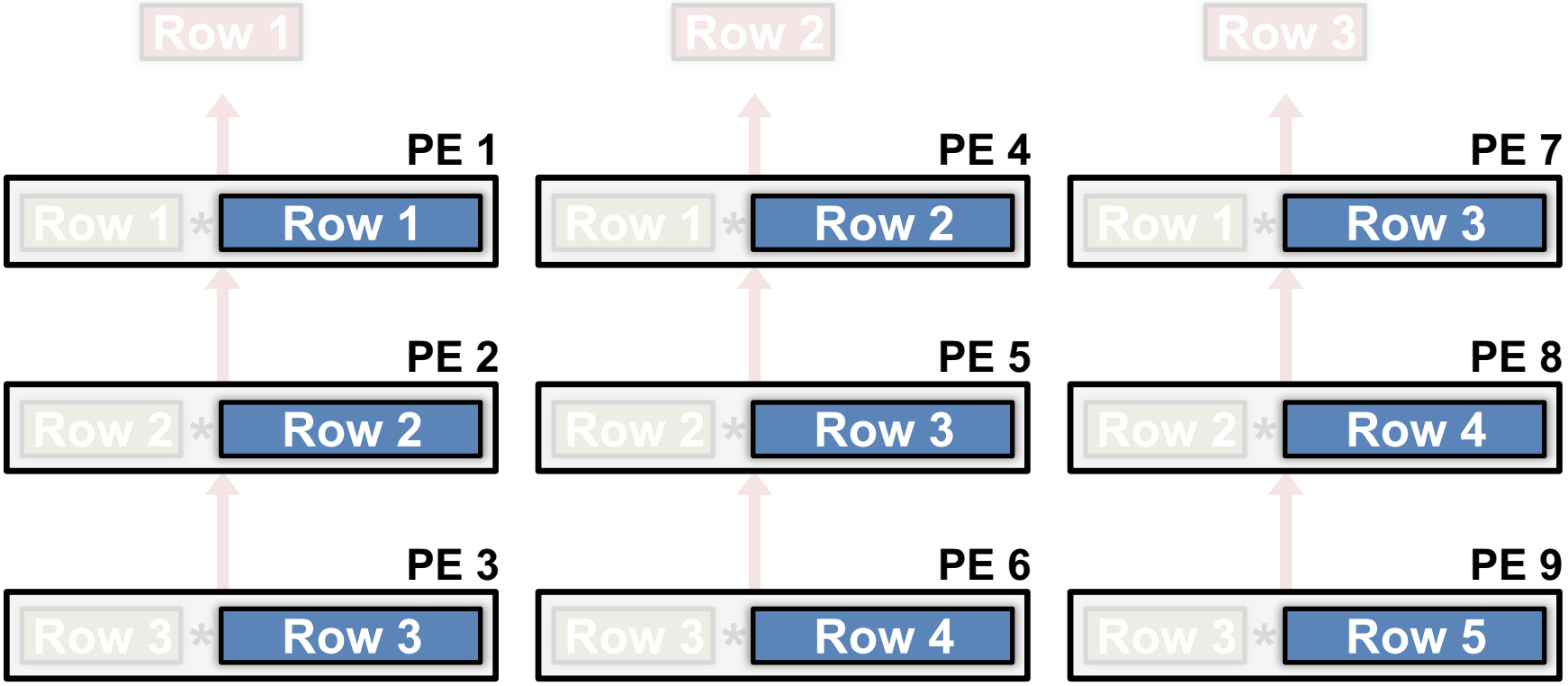


Convolutional Reuse Maximized



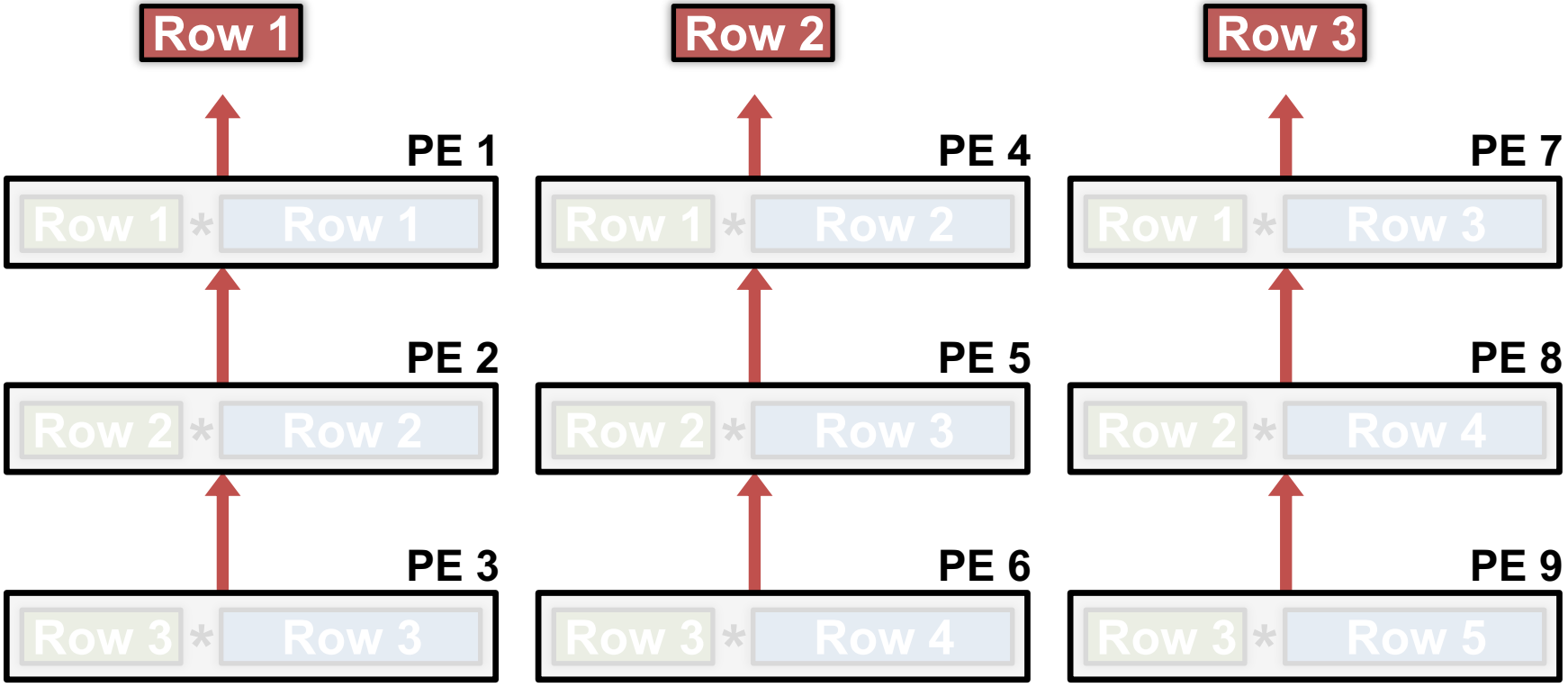
Filter rows are reused across PEs **horizontally**

Convolutional Reuse Maximized



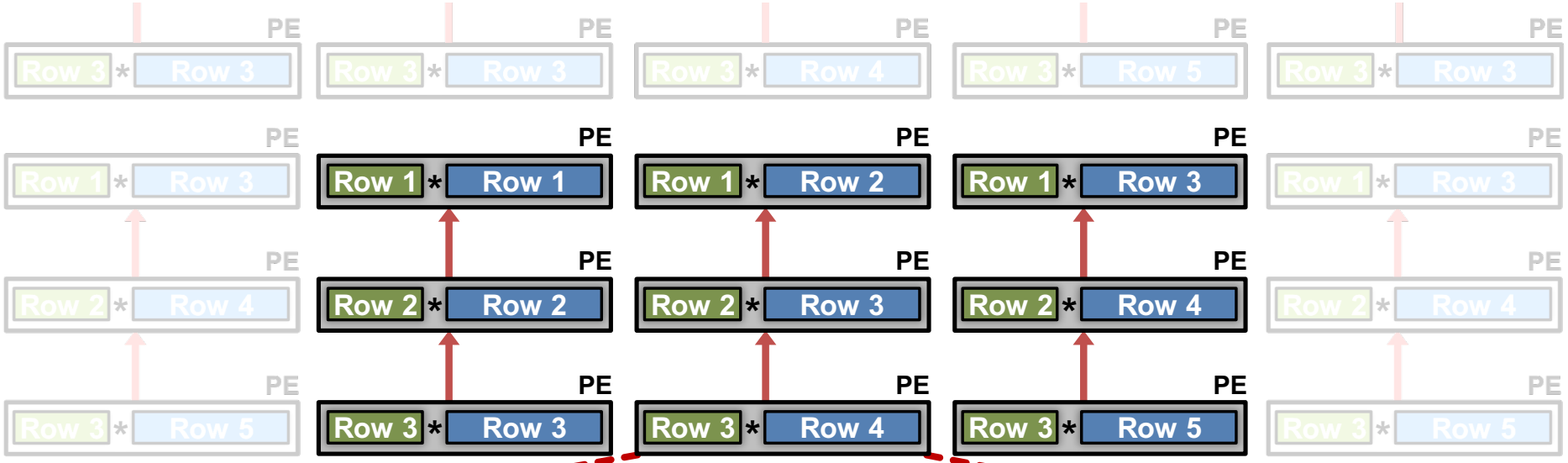
Fmap rows are reused across PEs **diagonally**

Maximize 2D Accumulation in PE Array



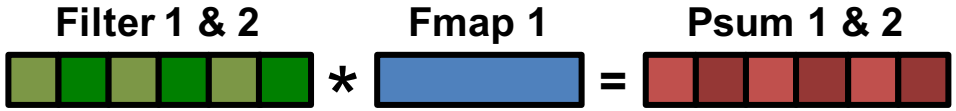
Partial sums accumulate across PEs **vertically**

Flexibility to Map Multiple Dimensions



Multiple **fmaps**:

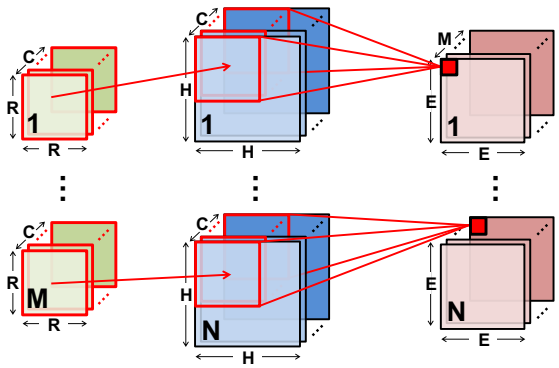
Multiple **filters**:



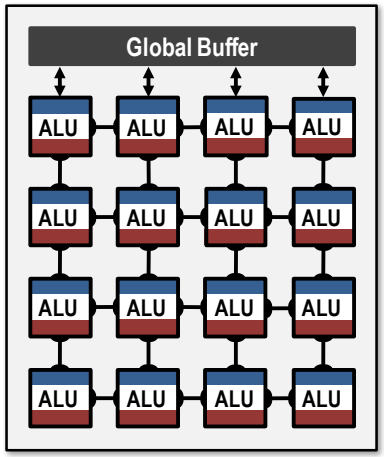
Multiple **channels**:

Finding the Optimal Mapping

DNN Configurations

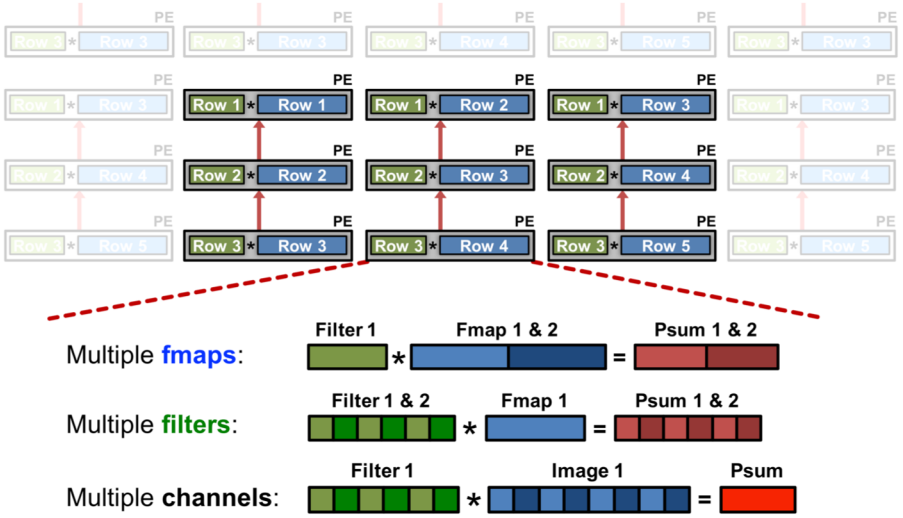


Hardware Resources

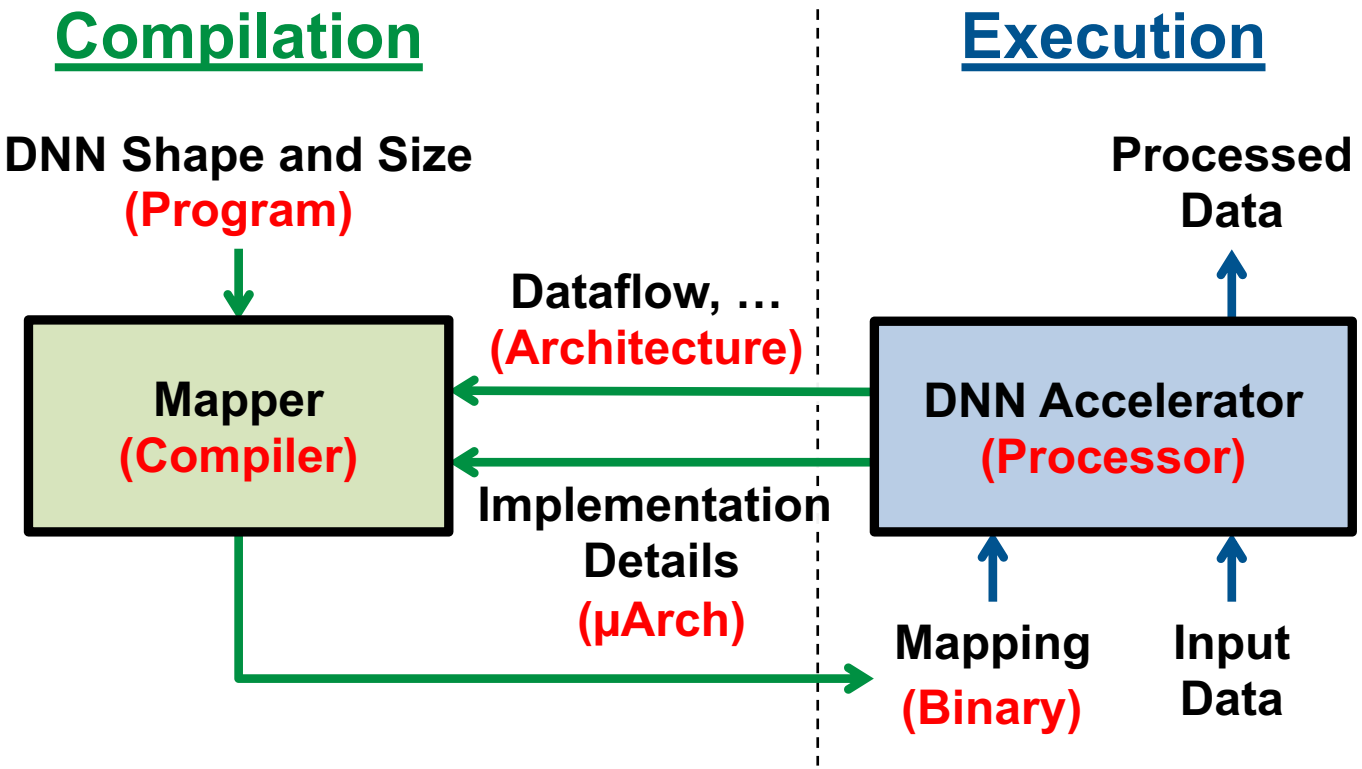


Optimization Compiler

Row Stationary Mapping



Computer Architecture Analogy



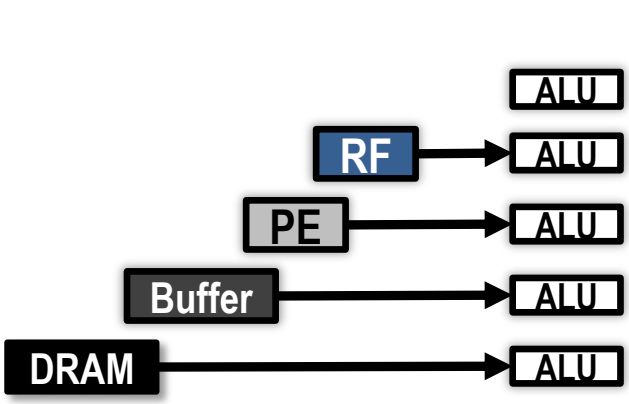
[Chen et al., Micro Top-Picks 2017]

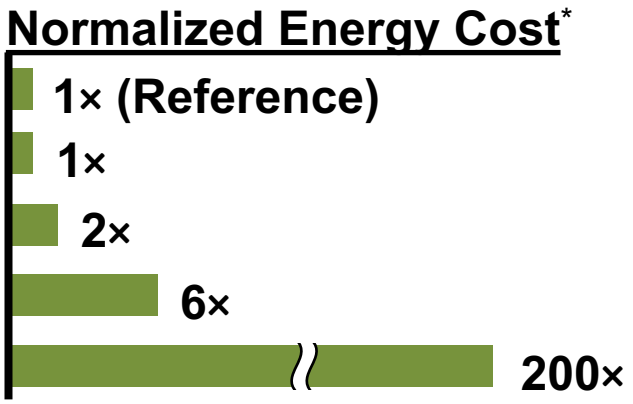
Evaluate Different Dataflows

- **Weight Stationary**
  - minimize movement of weights
- **Output Stationary**
  - minimize movement of partial sums
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**
- **Row Stationary**

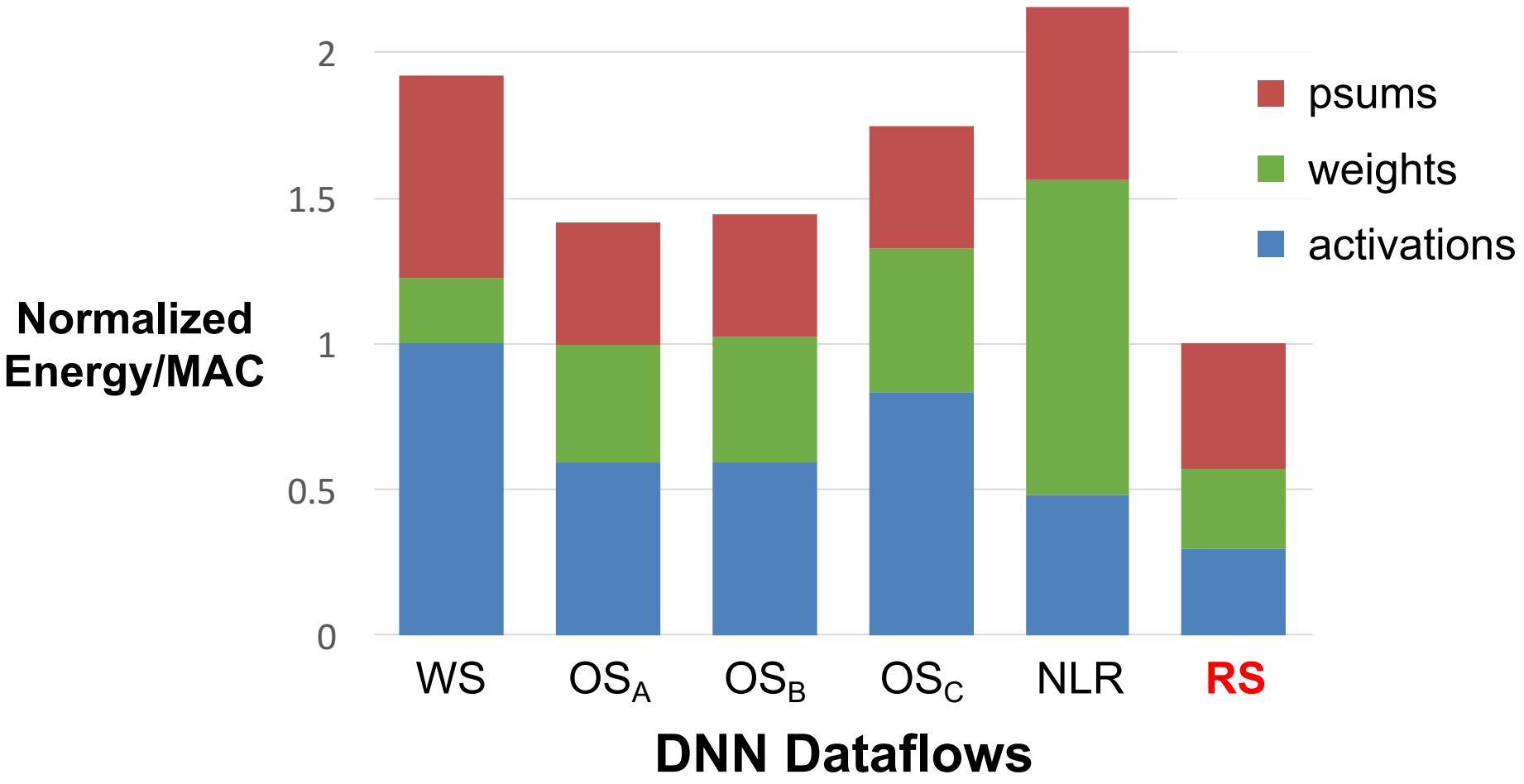
Evaluation Setup

- Same Total Area
- AlexNet
- 256 PEs
- Batch size = 16





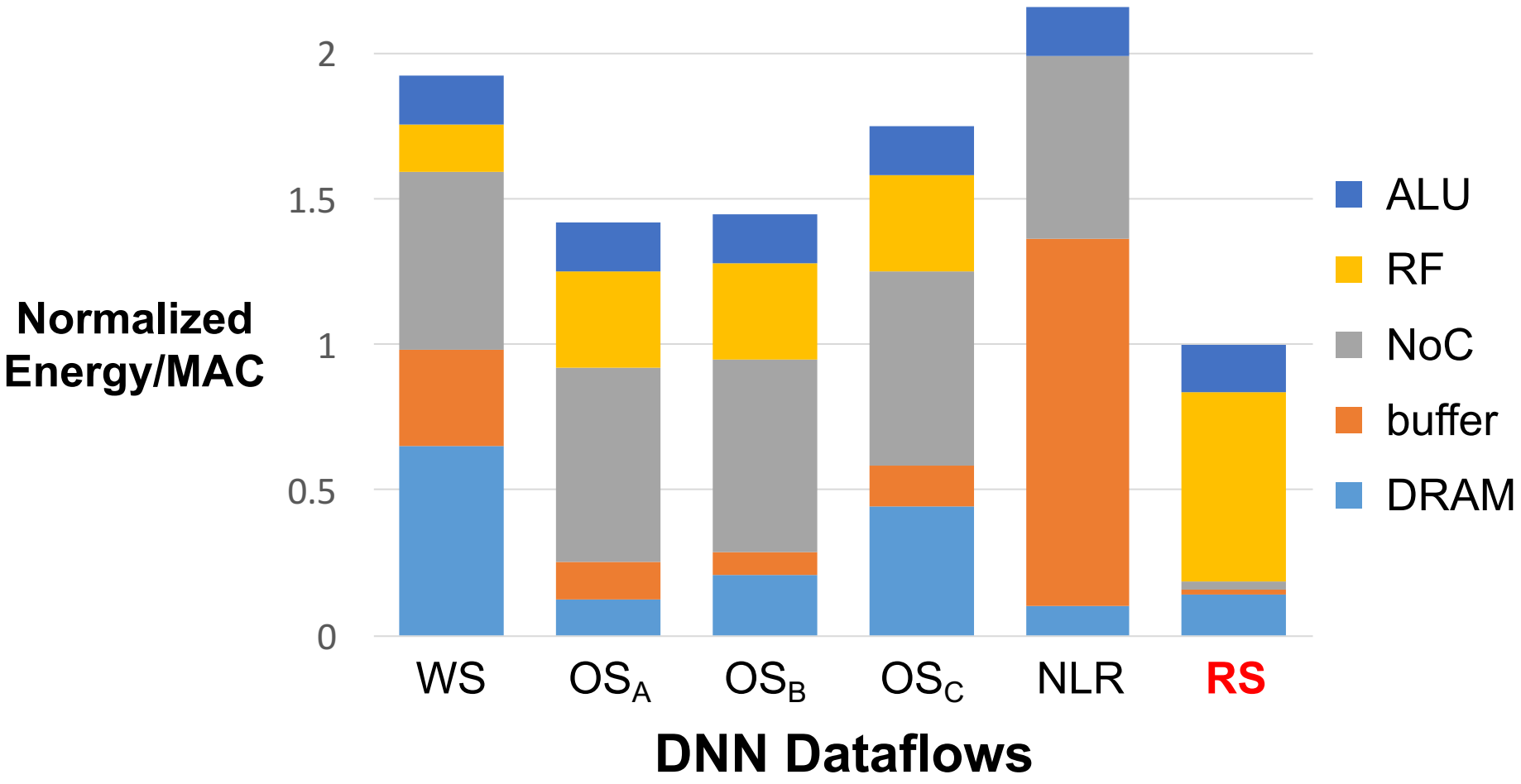
Dataflow Comparison: CONV Layers



RS optimizes for the best **overall** energy efficiency

[Chen et al., ISCA 2016]

Dataflow Comparison: CONV Layers

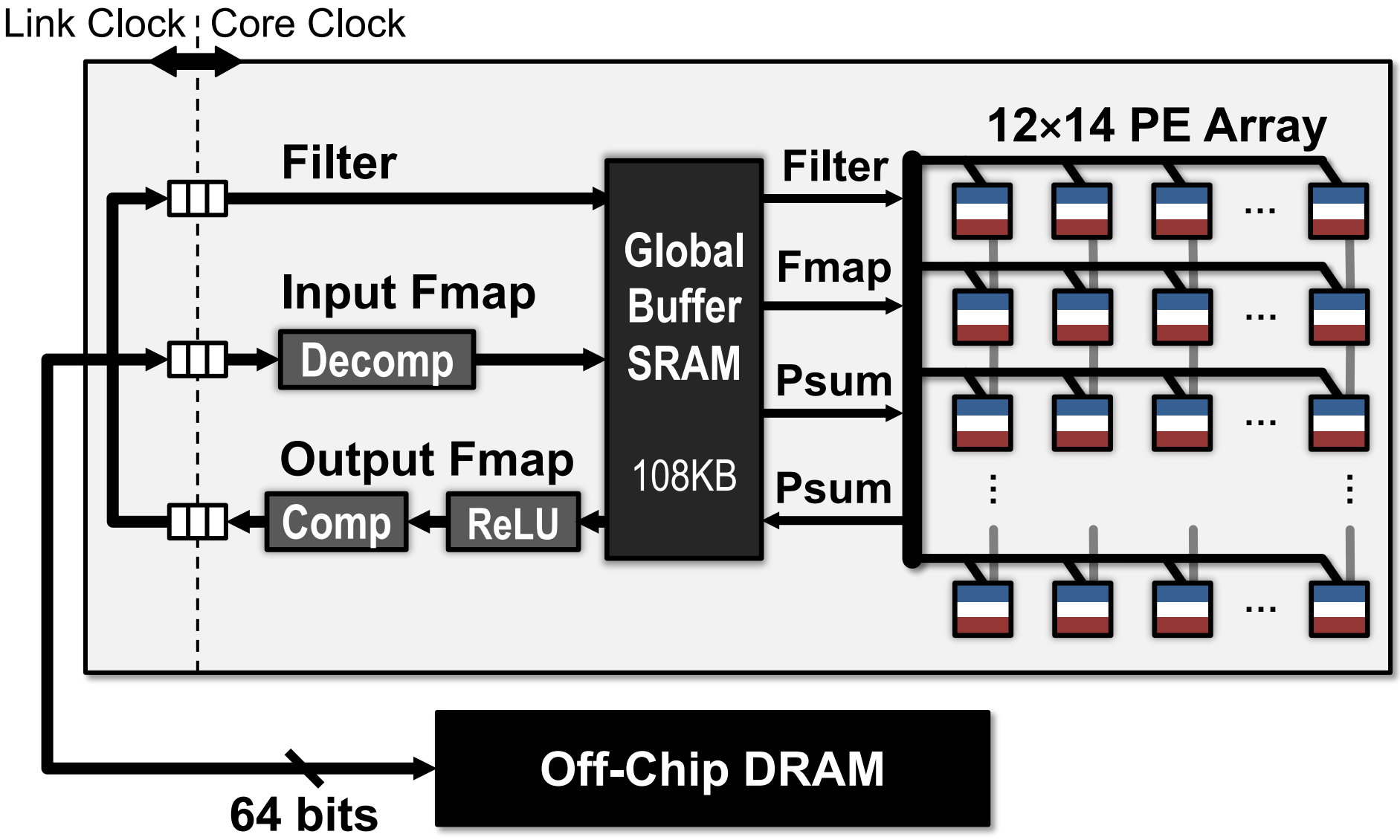


RS uses **1.4× – 2.5× lower** energy than other dataflows

[Chen et al., ISCA 2016]

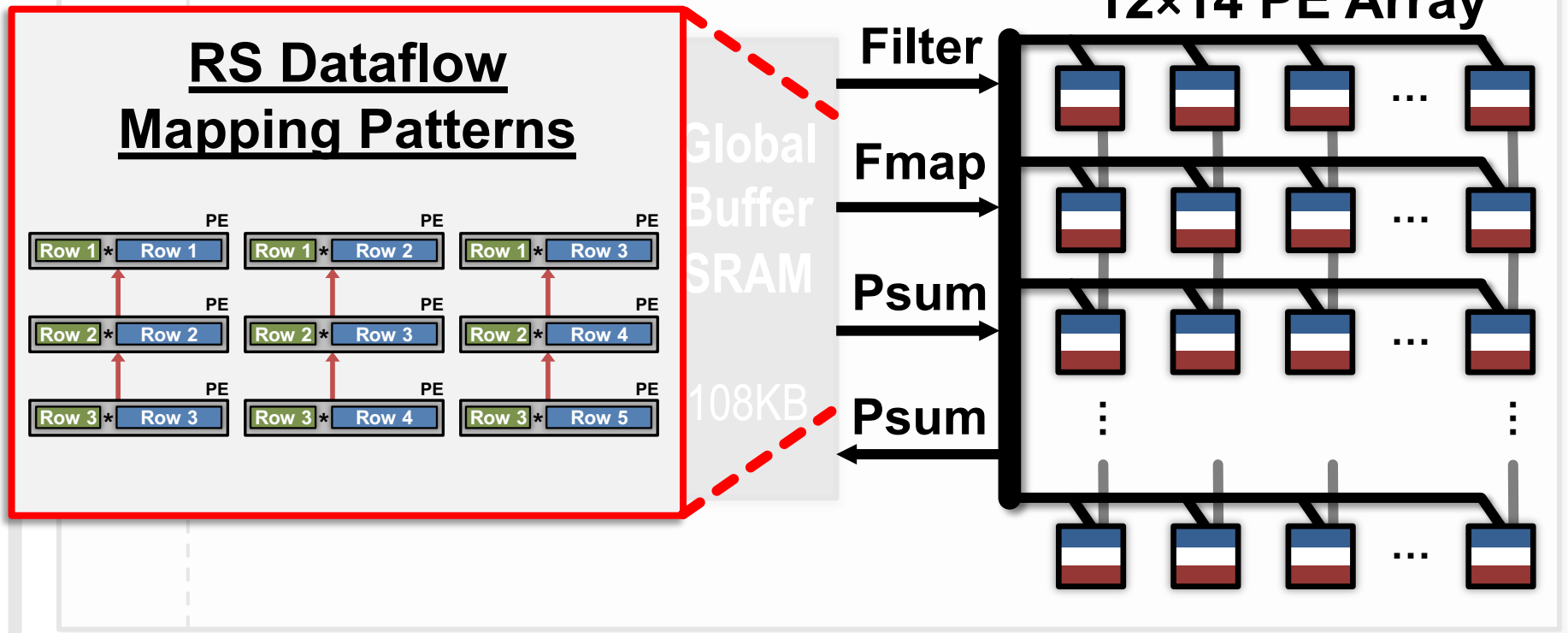
Hardware Architecture for RS Dataflow

Eyeriss: DNN Accelerator for RS Dataflow



[Chen et al., ISSCC 2016]

Flexibility Required for Mapping

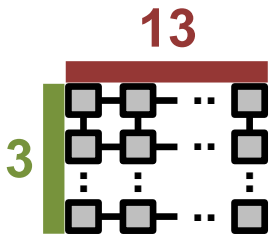


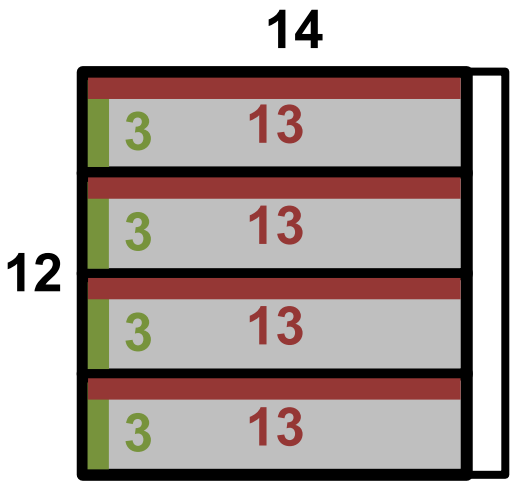
How to map **different shapes** with a **fixed-size PE array**?

Flexible Mapping Strategy

Replication

AlexNet
Layer 3-5



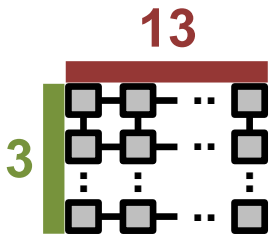


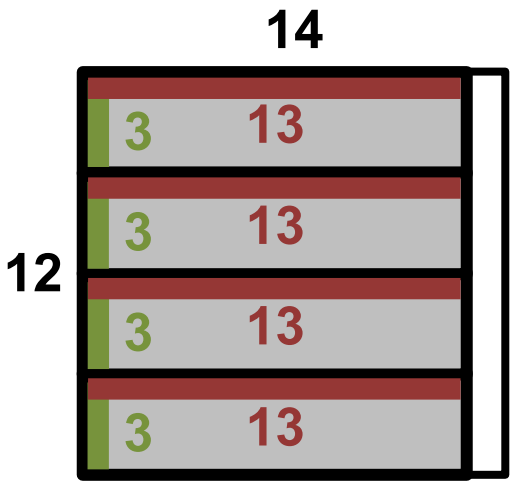
Physical PE Array

Flexible Mapping Strategy

Replication

AlexNet
Layer 3-5

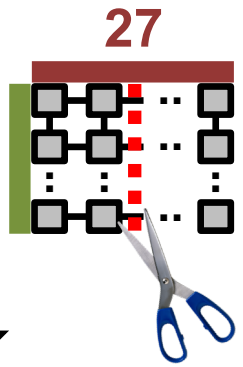


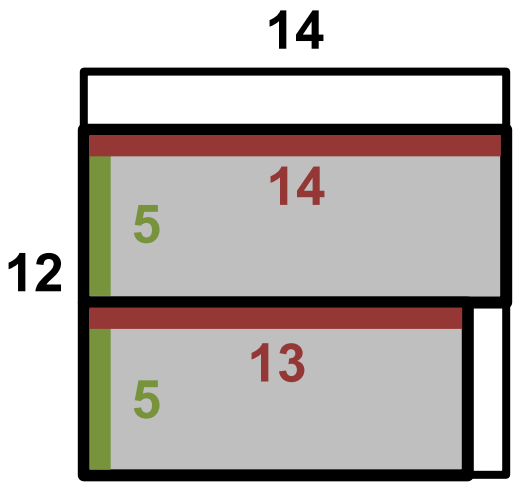


Physical PE Array

Folding

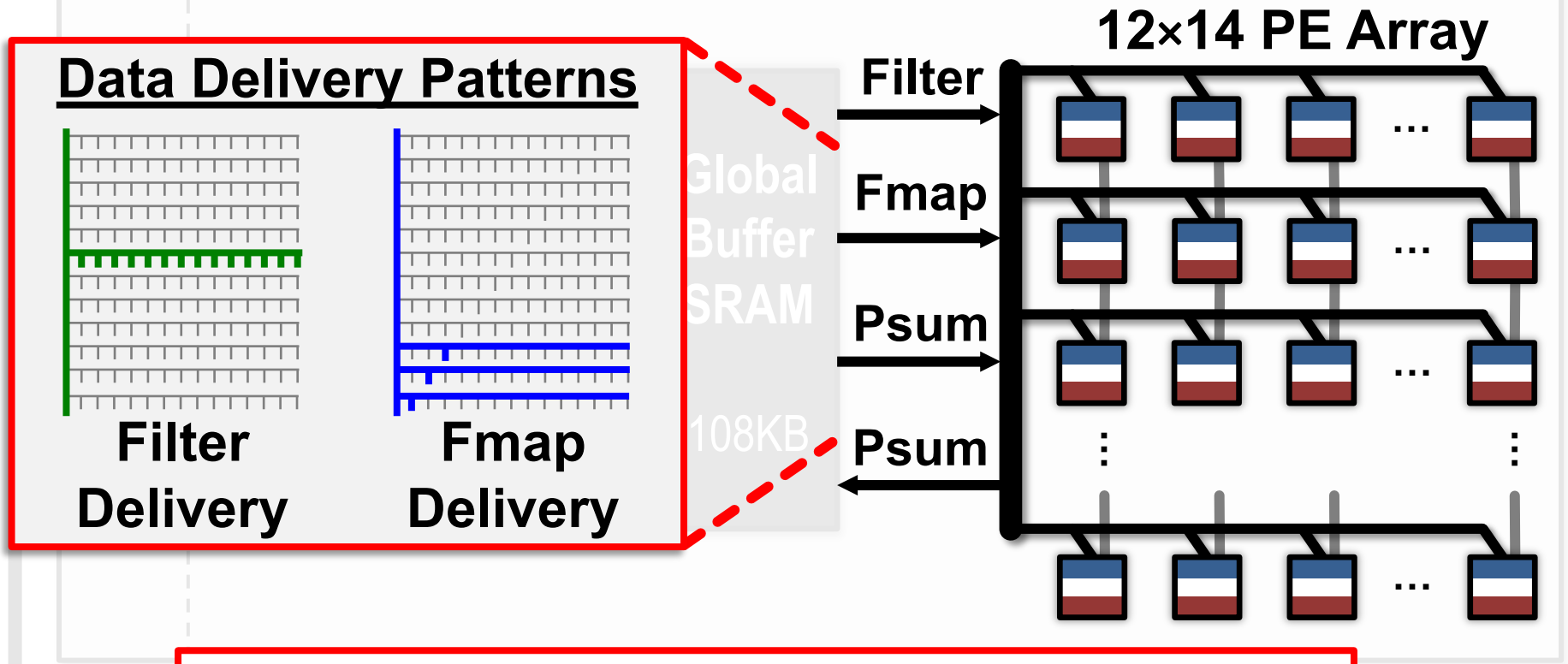
AlexNet
Layer 2





Physical PE Array

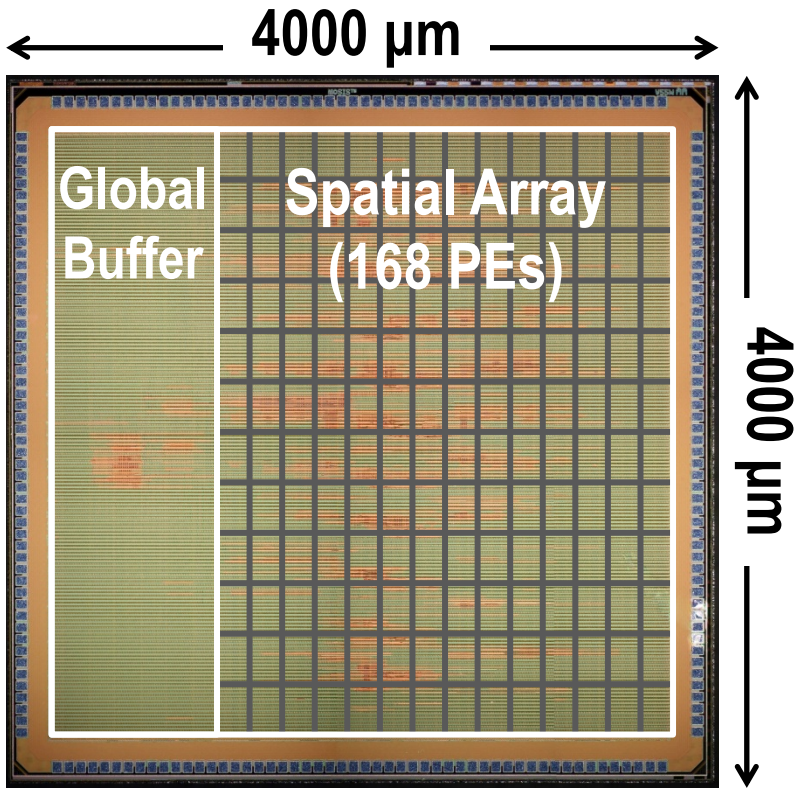
Data Delivery with On-Chip Network



Supports **any delivery pattern** & saves **over 80% energy** compared to broadcast

Chip Spec & Measurement Results

Technology	TSMC 65nm LP 1P9M
On-Chip Buffer	108 KB
# of PEs	168
Scratch Pad / PE	0.5 KB
Core Frequency	100 – 250 MHz
Peak Performance	33.6 – 84.0 GOPS
Word Bit-width	16-bit Fixed-Point
Natively Supported DNN Shapes	Filter Width: 1 – 32 Filter Height: 1 – 12 Num. Filters: 1 – 1024 Num. Channels: 1 – 1024 Horz. Stride: 1–12 Vert. Stride: 1, 2, 4



To support 2.66 GMACs [8 billion 16-bit inputs (**16GB**) and 2.7 billion outputs (**5.4GB**)], only requires **208.5MB** (buffer) and **15.4MB** (DRAM)

[Chen et al., ISSCC 2016]

DNN Model & Hardware Co-Design

Approaches

- **Reduce size of operands for storage/compute**
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

- **Reduce number of operations for storage/compute**
  - Exploit Activation Statistics (Compression)
  - Network Pruning
  - Compact Network Architectures

Approaches

- **Reduce size of operands for storage/compute**
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

- **Reduce number of operations for storage/compute**
  - Exploit Activation Statistics (Compression)
  - Network Pruning
  - Compact Network Architectures

Accuracy needs to be taken into account!

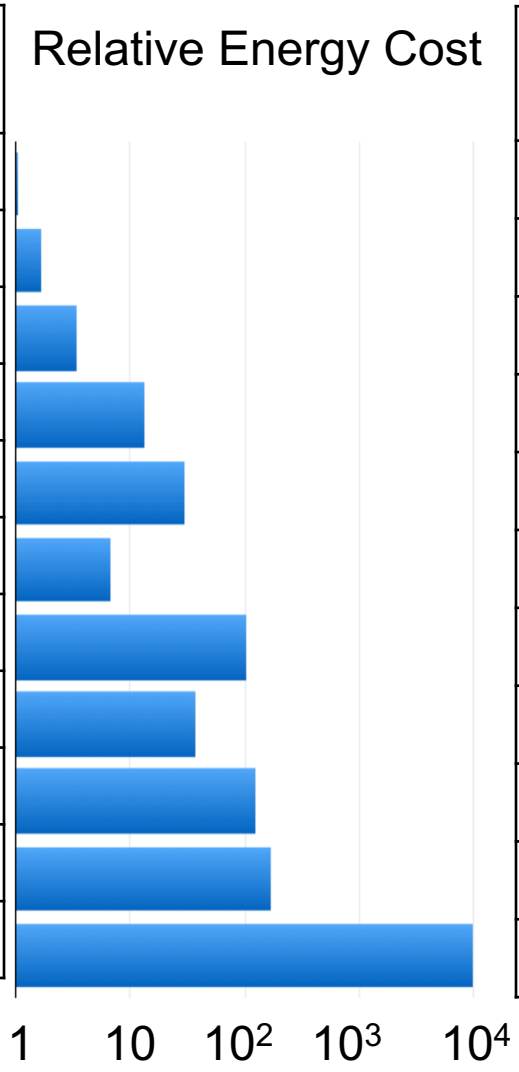
Approaches

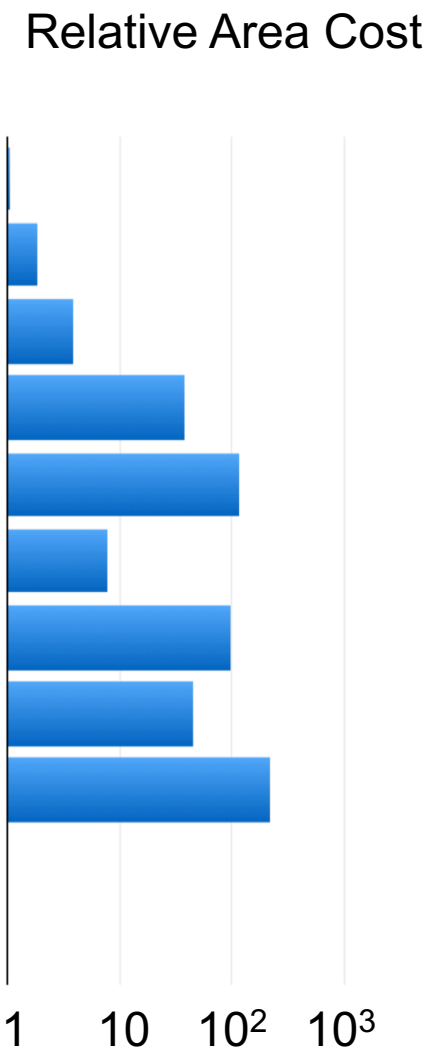
- **Reduce size of operands for storage/compute**
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

- **Reduce number of operations for storage/compute**
  - Exploit Activation Statistics (Compression)
  - Network Pruning
  - Compact Network Architectures

Cost of 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

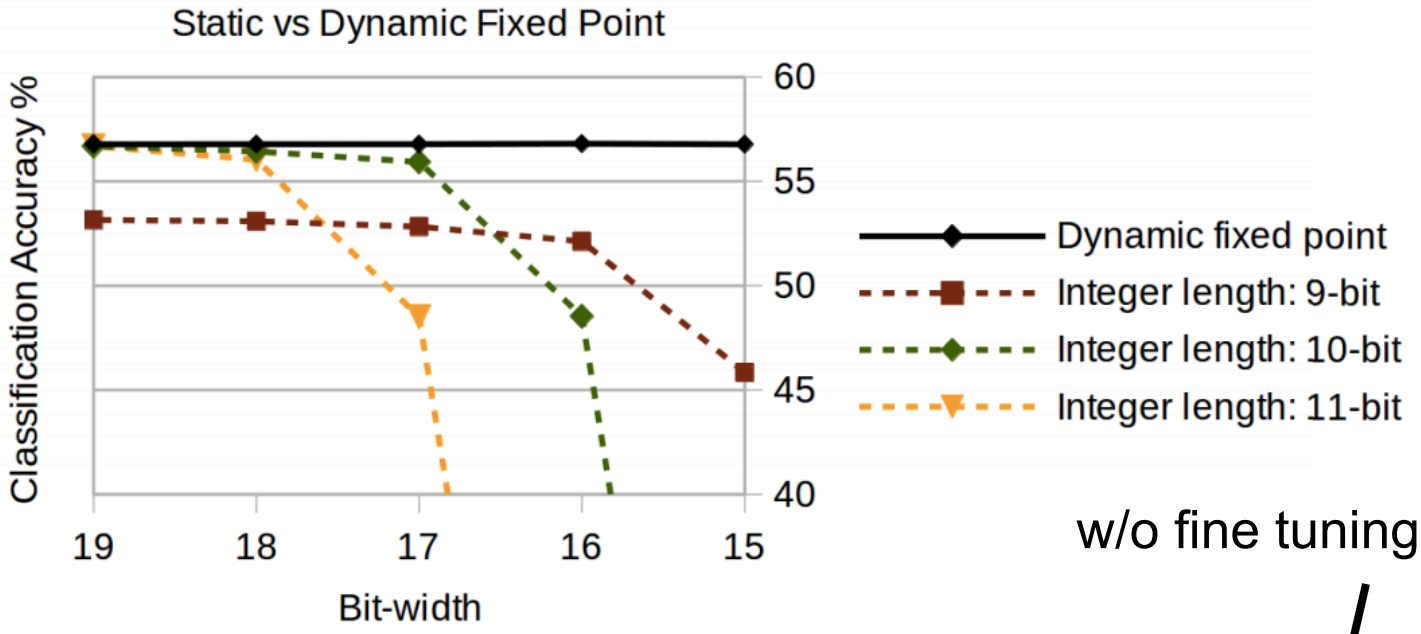




[Horowitz et al., ISSCC 2014]

Impact on Accuracy

Top-1 accuracy on of CaffeNet on ImageNet



w/o fine tuning

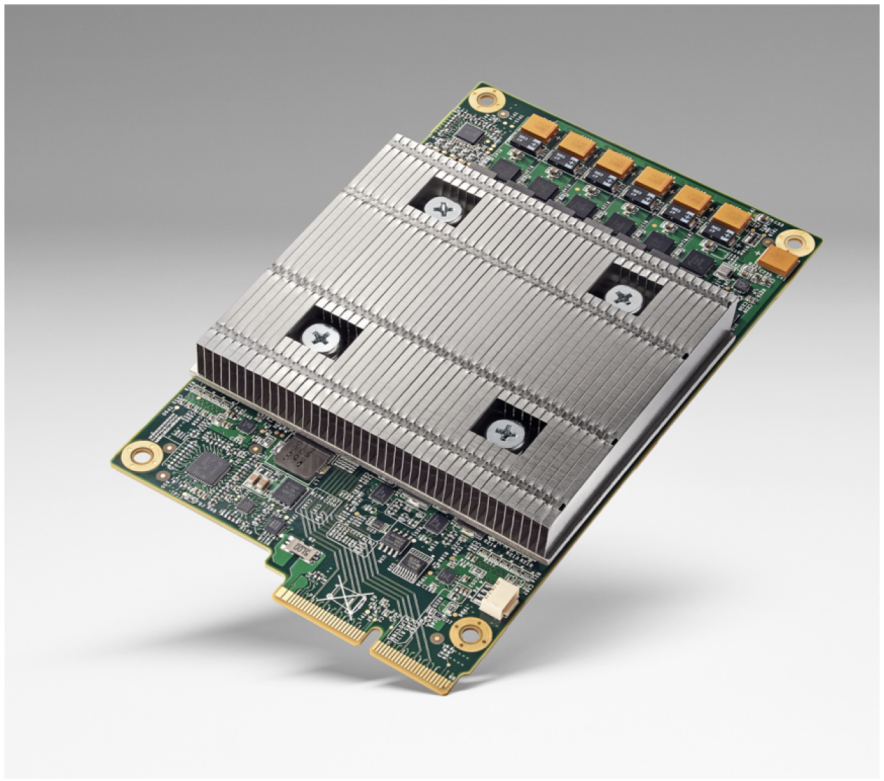
	Layer outputs	CONV parameters	FC parameters	32-bit floating point baseline	Fixed point accuracy
LeNet (Exp 1)	4-bit	4-bit	4-bit	99.1%	99.0% (98.7%)
LeNet (Exp 2)	4-bit	2-bit	2-bit	99.1%	98.8% (98.0%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.7%	81.4% (80.6%)
SqueezeNet top-1	8-bit	8-bit	8-bit	57.7%	57.1% (55.2%)
CaffeNet top-1	8-bit	8-bit	8-bit	56.9%	56.0% (55.8%)
GoogLeNet top-1	8-bit	8-bit	8-bit	68.9%	66.6% (66.1%)

[Gysel et al., Ristretto, ICLR 2016]

Reduced Precision in the Industry

8-bit INT is widely used for inference in commercial products

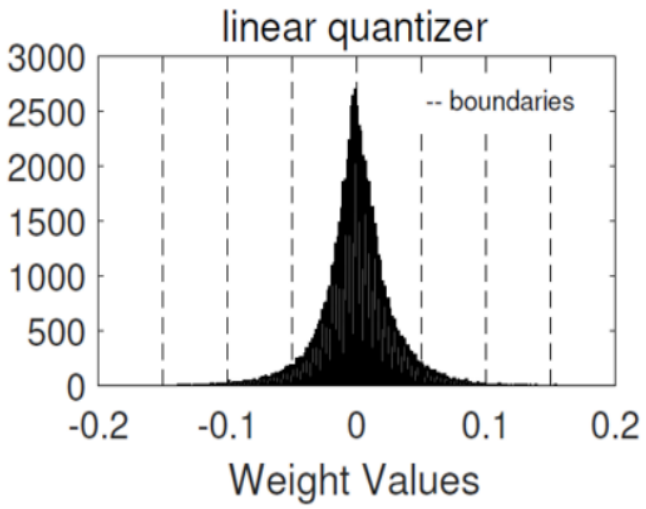
Nvidia Pascal (2016)

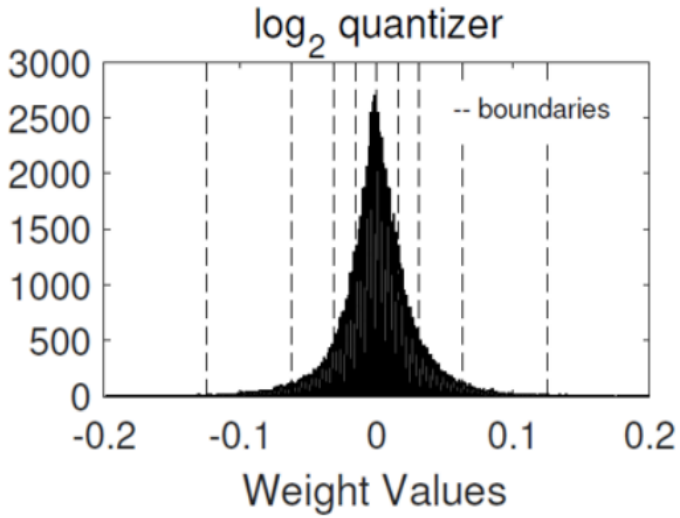


Google's TPU v1 (2016)

Reduced Precision in Research

Network	Precision		Accuracy Loss*
	Weights	Activations	
Binarized Neural Net (BNN)	1	1	29.8%
XNOR-Net	1†	1†	11.0%
HWGQ-Net	1†	2†	5.2%
Binary Weight Net (BWN)	1†	32f	0.8%
Trained Ternary Quant (TTQ)	2†	32f	0.6%
LogNet	4/5	4	3.2%





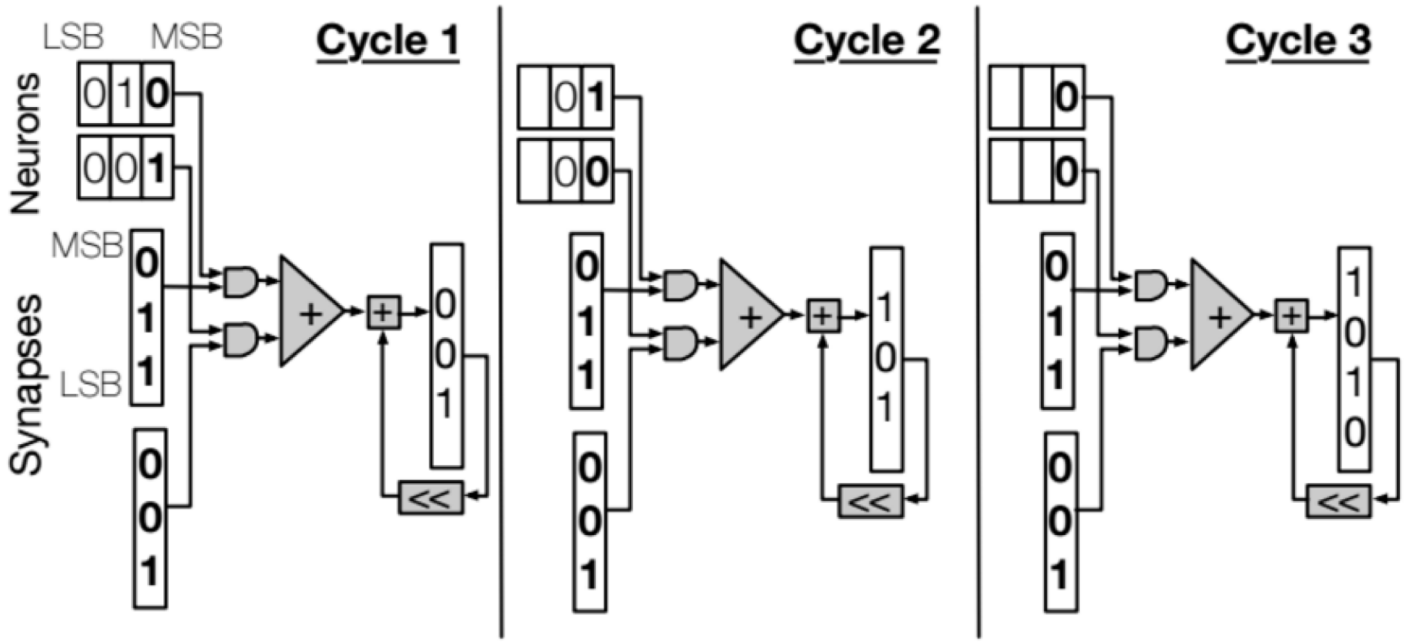
* AlexNet, compared to 32f

† 32f in first and last layers

Bitwidth Scaling for Higher Throughput

Bit-Serial Processing: Reduce Bitwidth → Skip Cycles
2.24x speed-up vs. 16-bit fixed

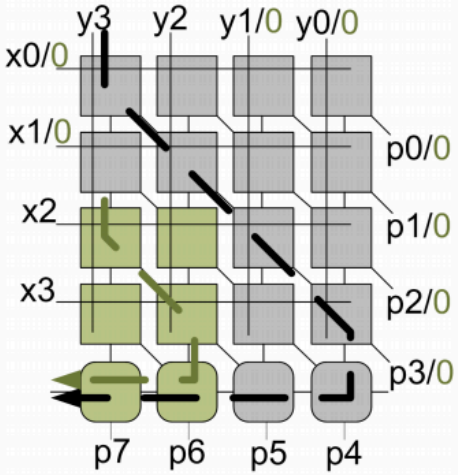
$$\sum_{i=0}^{N_i-1} s_i \times n_i = \sum_{i=0}^{N_i-1} s_i \times \sum_{b=0}^{P-1} n_i^b \times 2^b = \sum_{b=0}^{P-1} 2^b \times \sum_{i=0}^{N_i-1} n_i^b \times S_i$$

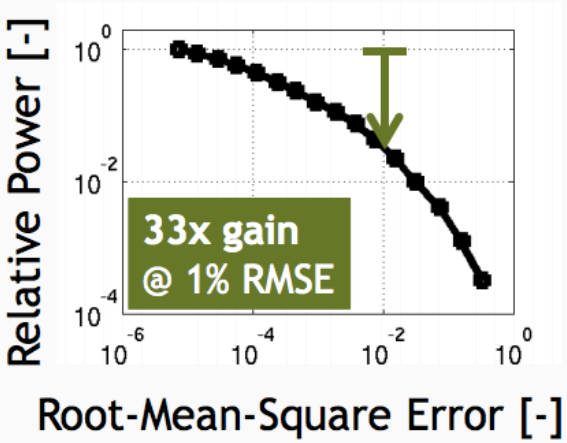


[Judd et al., Stripes, CAL 2016]

Bitwidth Scaling for Lower Power

Reduce Bitwidth
→ Shorter Critical Path
→ Reduce Voltage

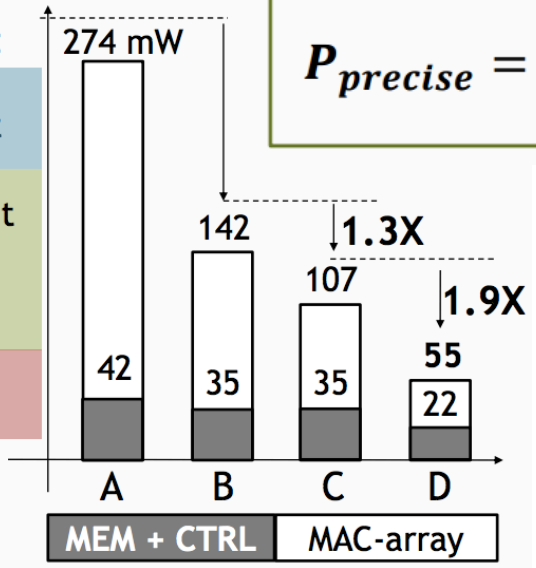




$$P_{precise} = \alpha C f V^2 \quad \Rightarrow \quad P_{imprecise} = \frac{\alpha}{k_1} C f \left(\frac{V}{k_2}\right)^2$$

AlexNet Layer 2 example:

A. 2D-baseline	@ 16 bit
B. Precision-Scaling	@ 7-7 bit
C. Voltage-Scaling	@ 0.9 V
D. Sparse operation guarding	



2.56x power reduction
vs. 16-bit fixed
(AlexNet layer 2)

[Moons et al., VLSI 2016]

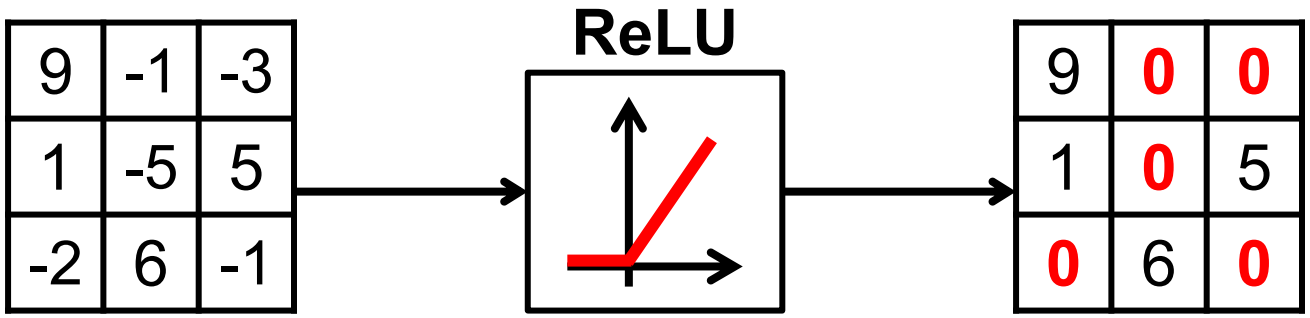
Approaches

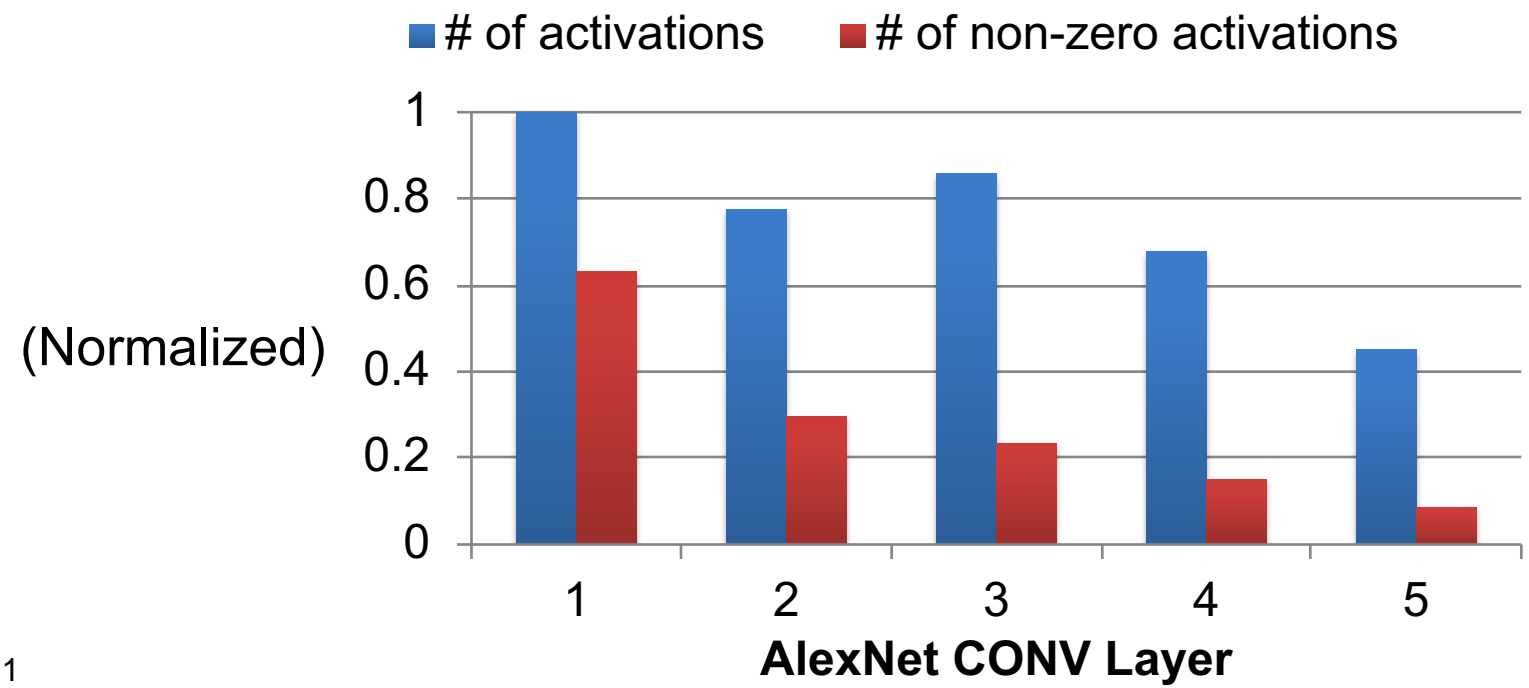
- Reduce size of operands for storage/compute
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

- **Reduce number of operations for storage/compute**
  - Exploit Activation Statistics (Compression)
  - Network Pruning
  - Compact Network Architectures

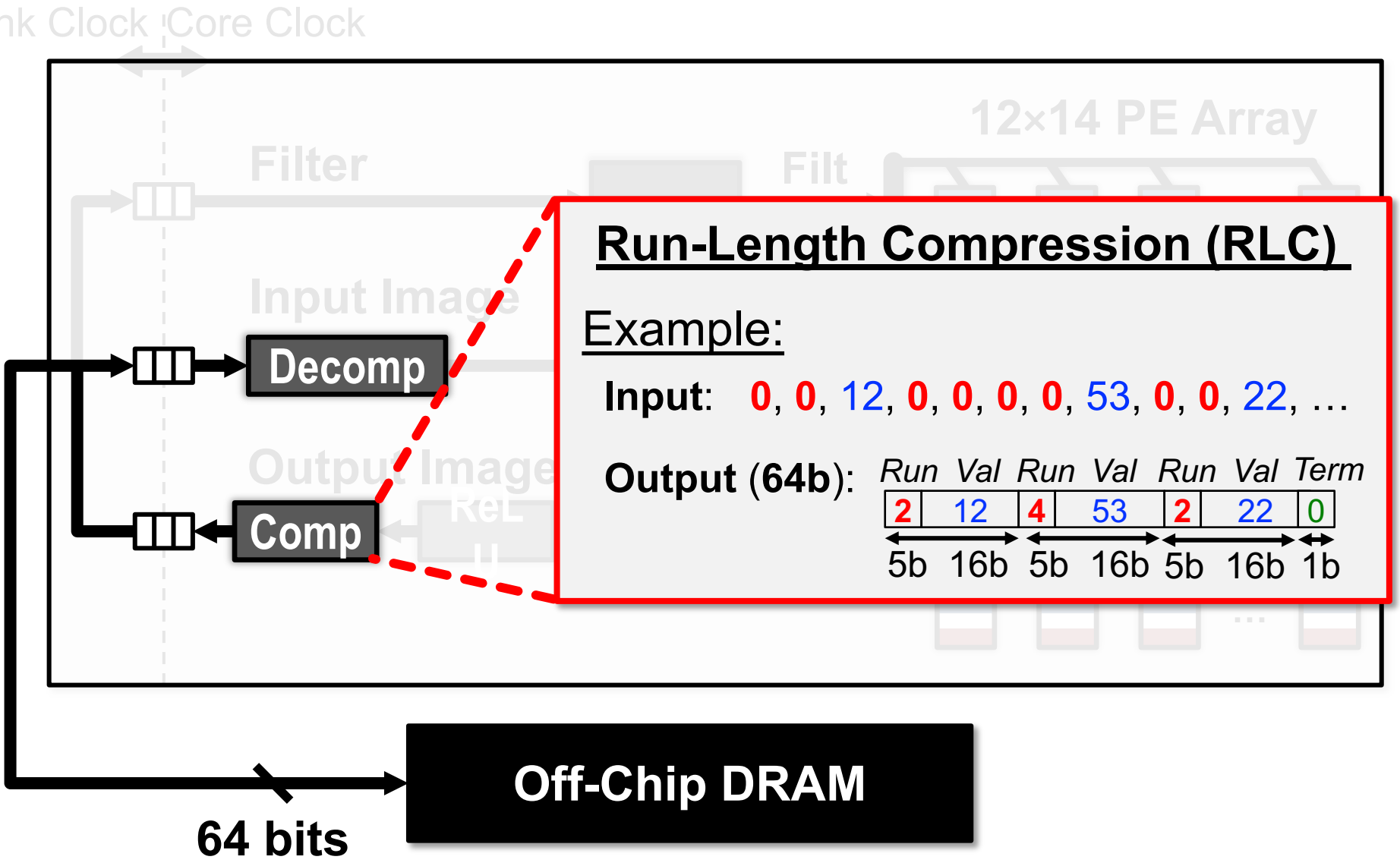
Sparsity in Feature Maps

Many **zeros** in **feature maps** after **ReLU**



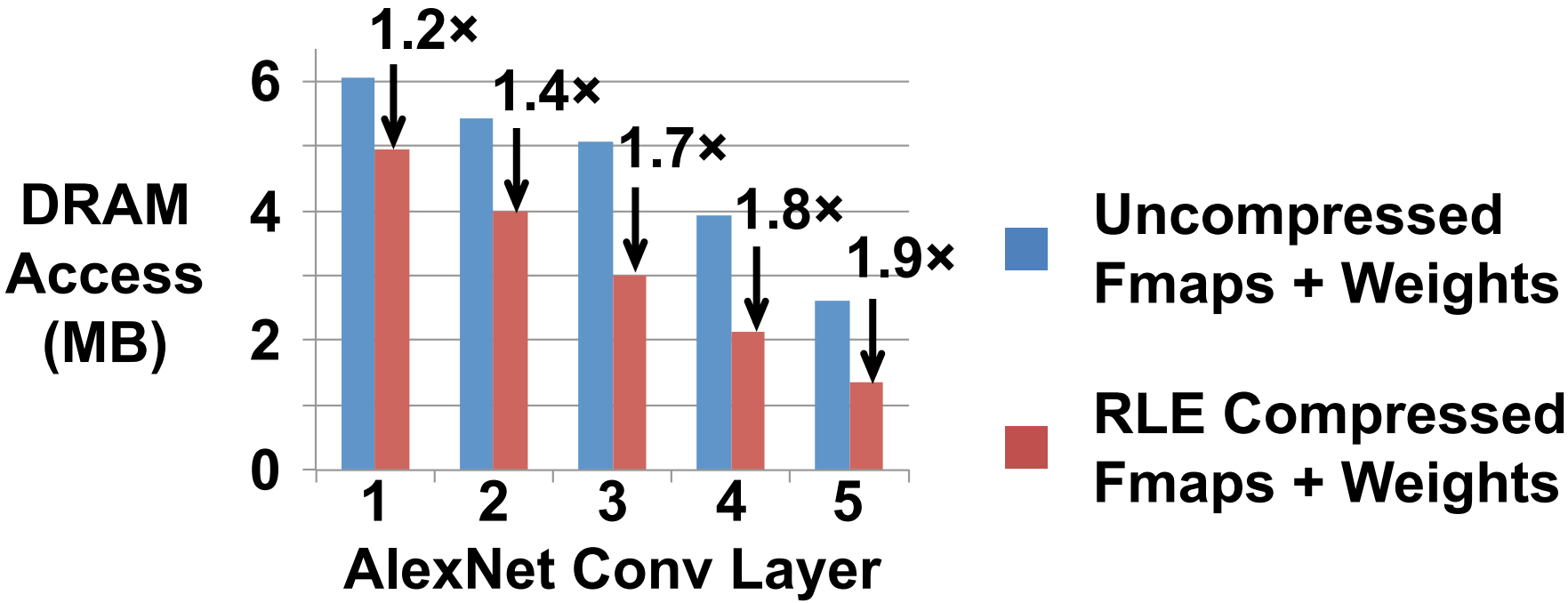


I/O Compression in Eyeriss



[Chen et al., ISSCC 2016]

Compression Reduces DRAM BW

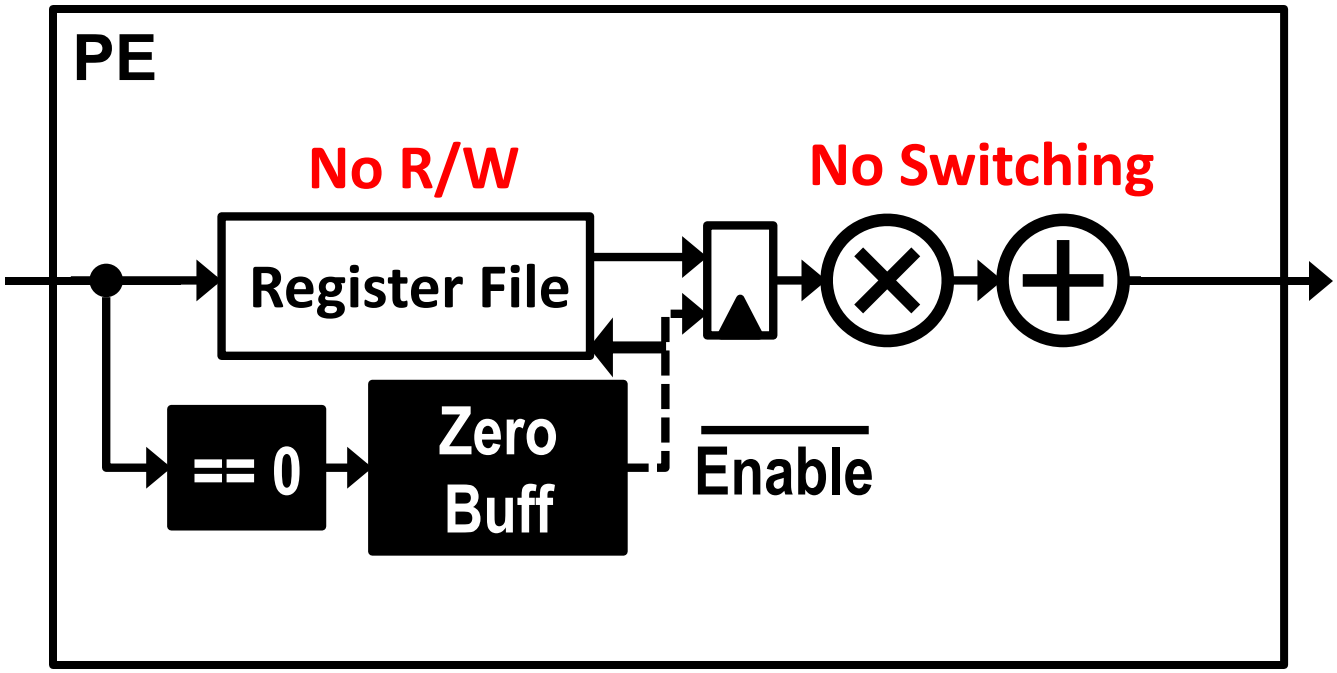


Simple RLC within 5% - 10% of theoretical entropy limit

[Chen et al., ISSCC 2016]

Data Gating / Zero Skipping in Eyeriss

Reduce PE Power by **45%**

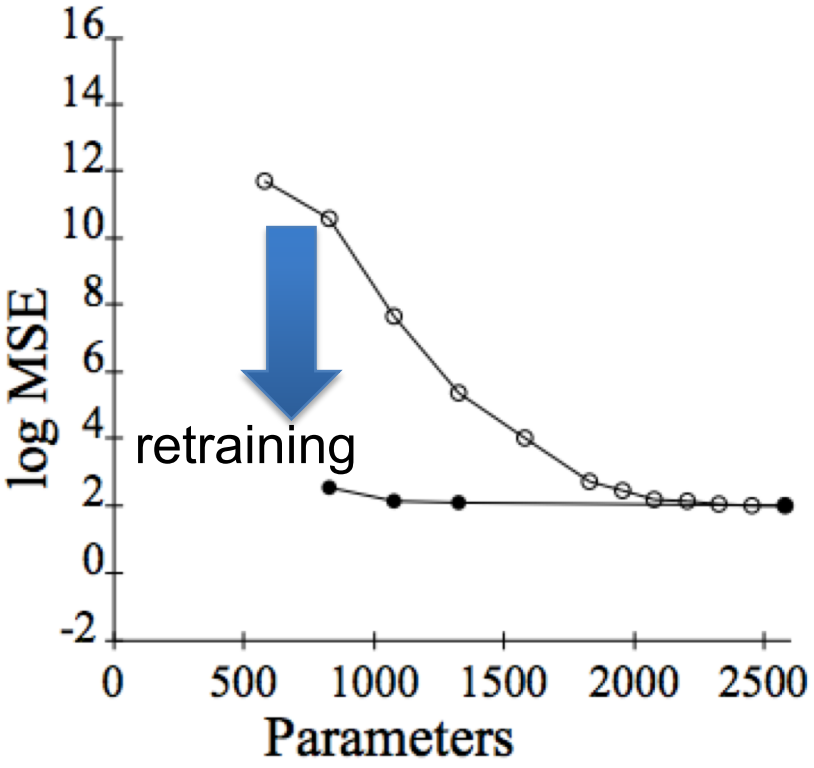


[Chen et al., ISSCC 2016]

Pruning – Make Weights Sparse

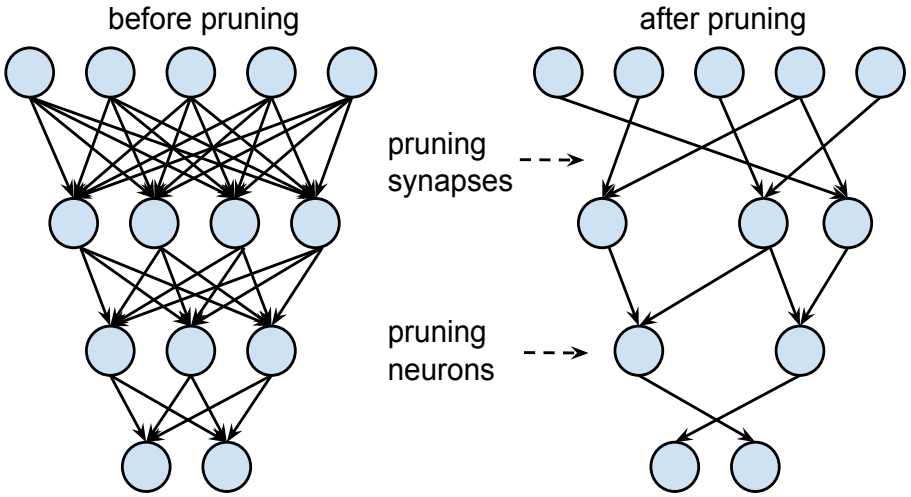
Optimal Brain Damage

[LeCun et al., NIPS 1989]



Magnitude-based Weight Pruning

[Han et al., NIPS 2015]



Example: *AlexNet*

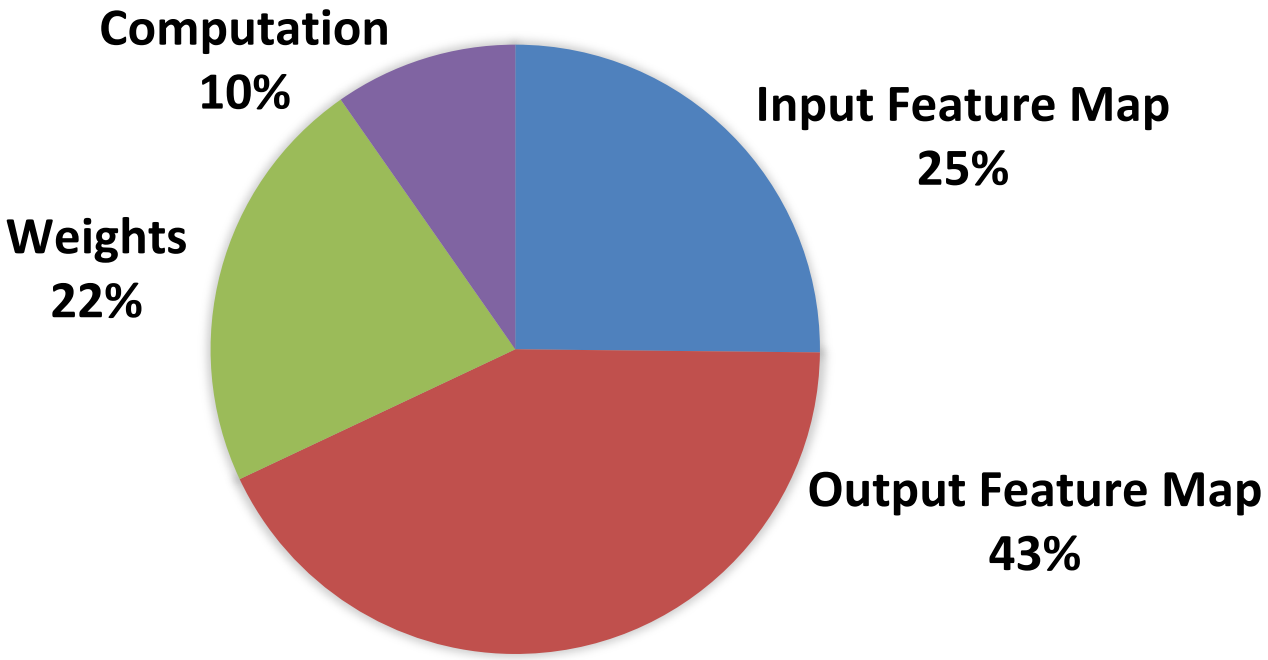
Weight Reduction:
CONV layers 2.7x, ***FC layers 9.9x***

Overall Reduction:
Weights 9x, MACs 3x

Key Observations on Pruning

- Number of weights *alone* is NOT a good metric for energy
- All data types should be considered

GoogLeNet Energy Consumption

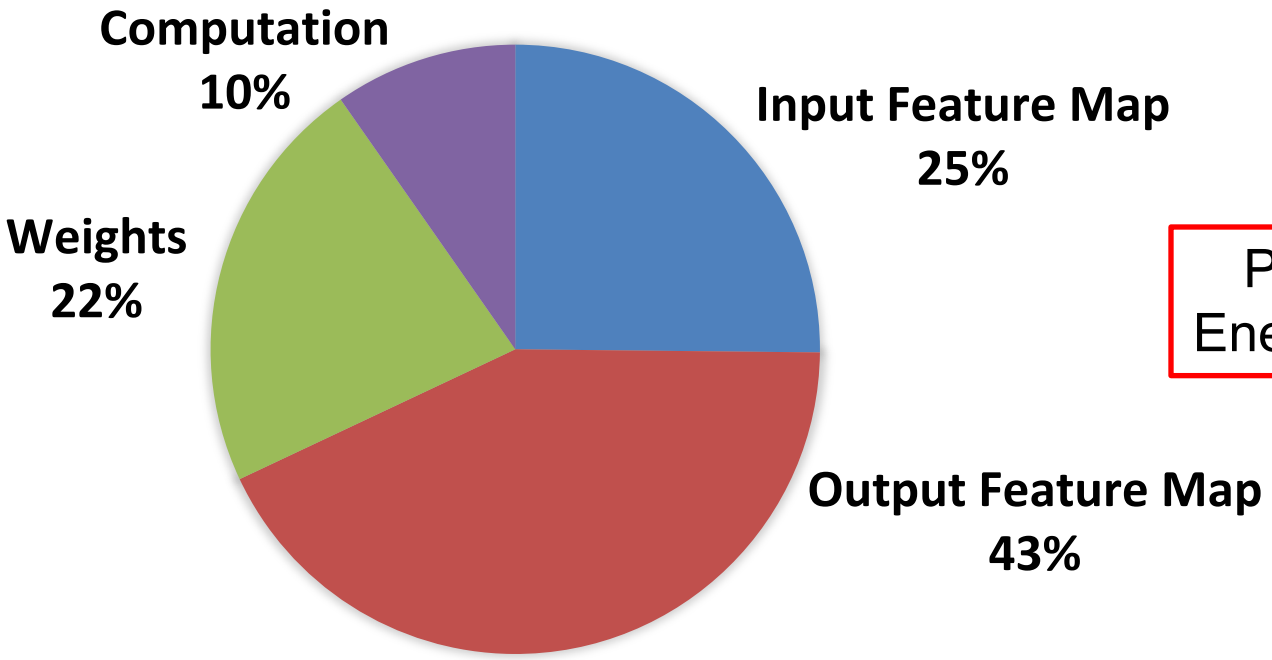


[Yang et al., CVPR 2017]

Key Observations on Pruning

- Number of weights *alone* is NOT a good metric for energy
- All data types should be considered

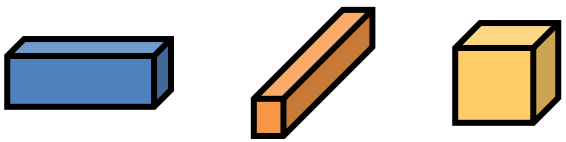
GoogLeNet Energy Consumption



Prune Based on Energy Consumption

[Yang et al., CVPR 2017]

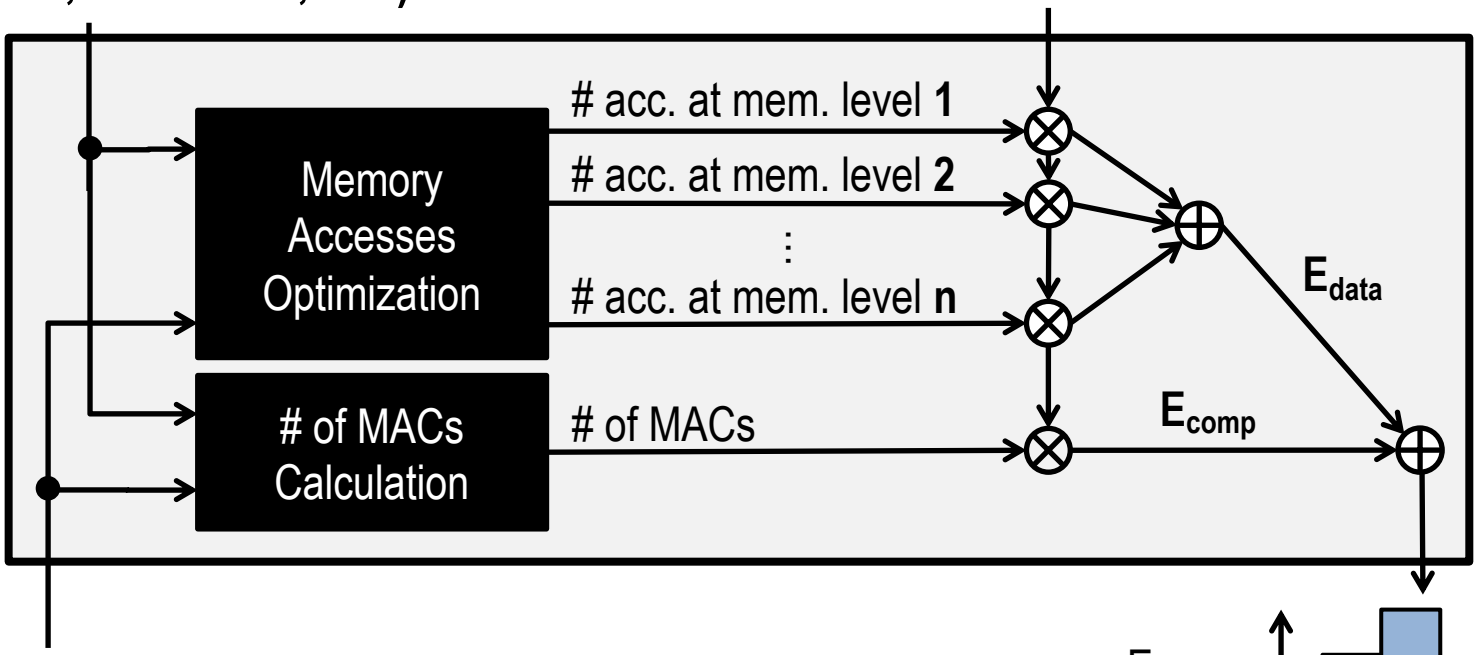
Energy-Evaluation Methodology



Tool available at http://eyeriss.mit.edu

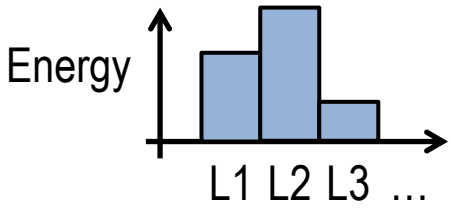
CNN Shape Configuration
(# of channels, # of filters, etc.)

Hardware Energy Costs of each MAC and Memory Access



CNN Weights and Input Data

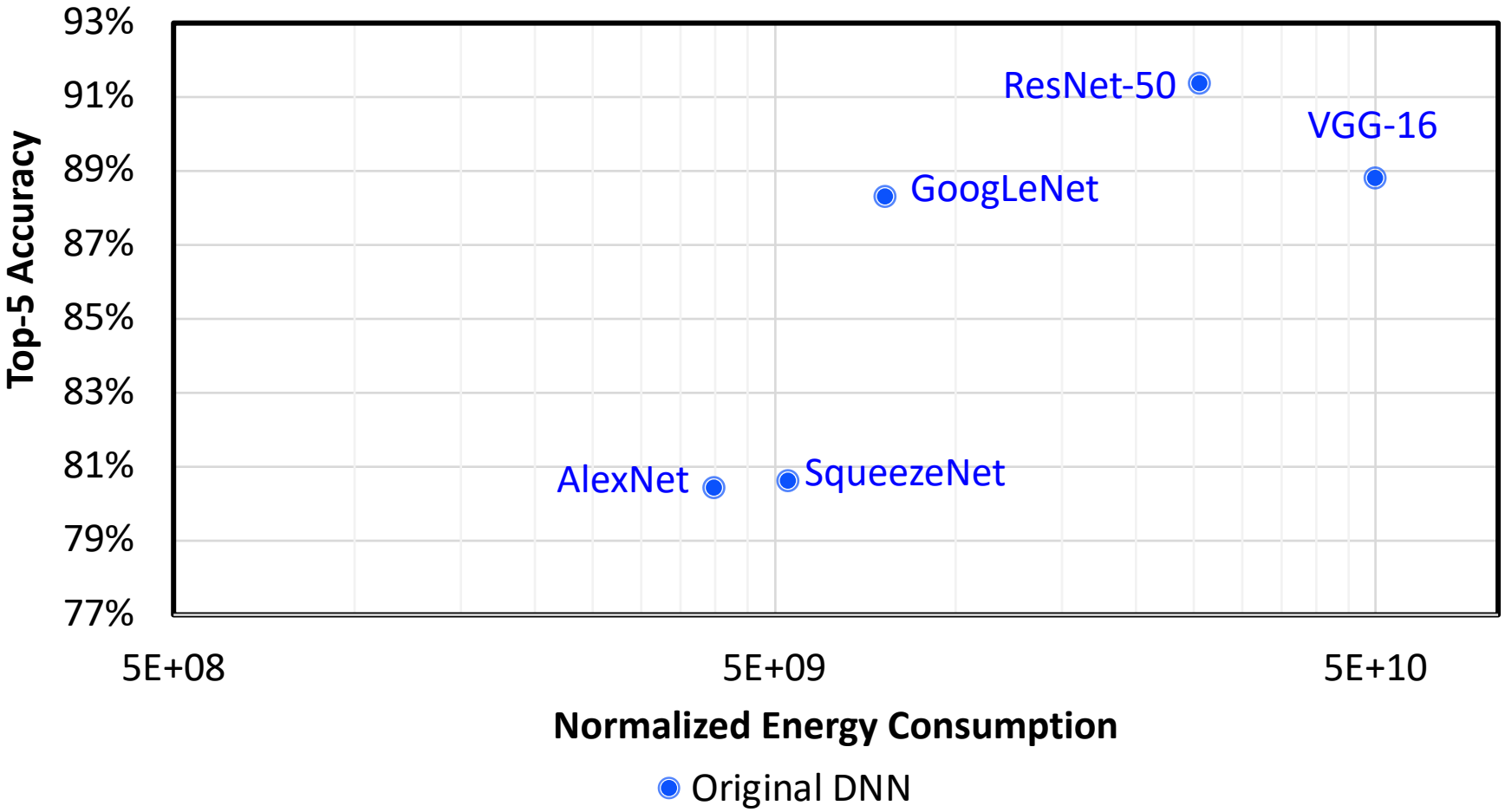
[0.3, 0, -0.4, 0.7, 0, 0, 0.1, …]



CNN Energy Consumption

[Yang et al., CVPR 2017]

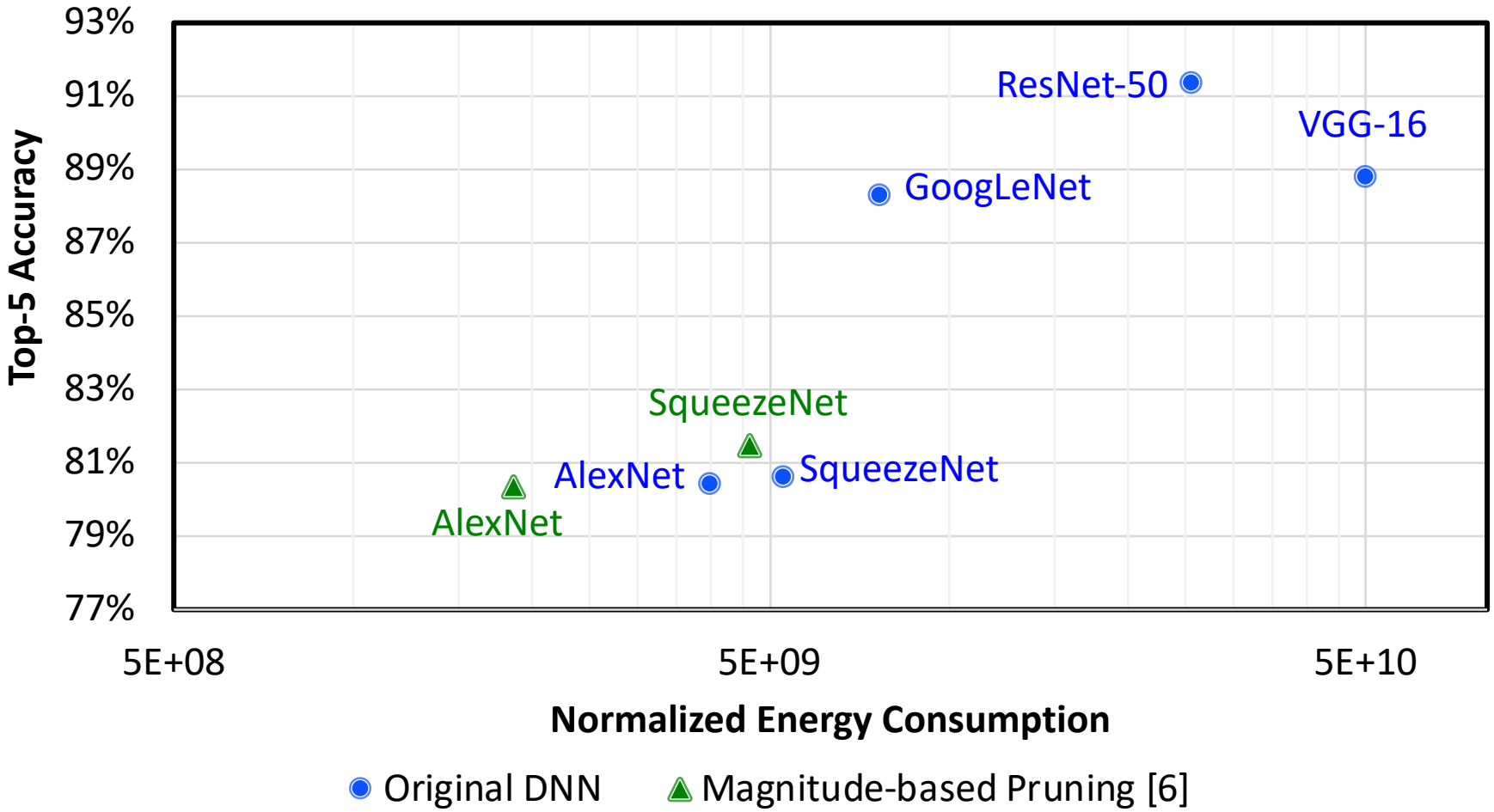
Energy Consumption of Existing DNNs



Deeper CNNs with fewer weights do not necessarily consume less energy than shallower CNNs with more weights

[Yang et al., CVPR 2017]

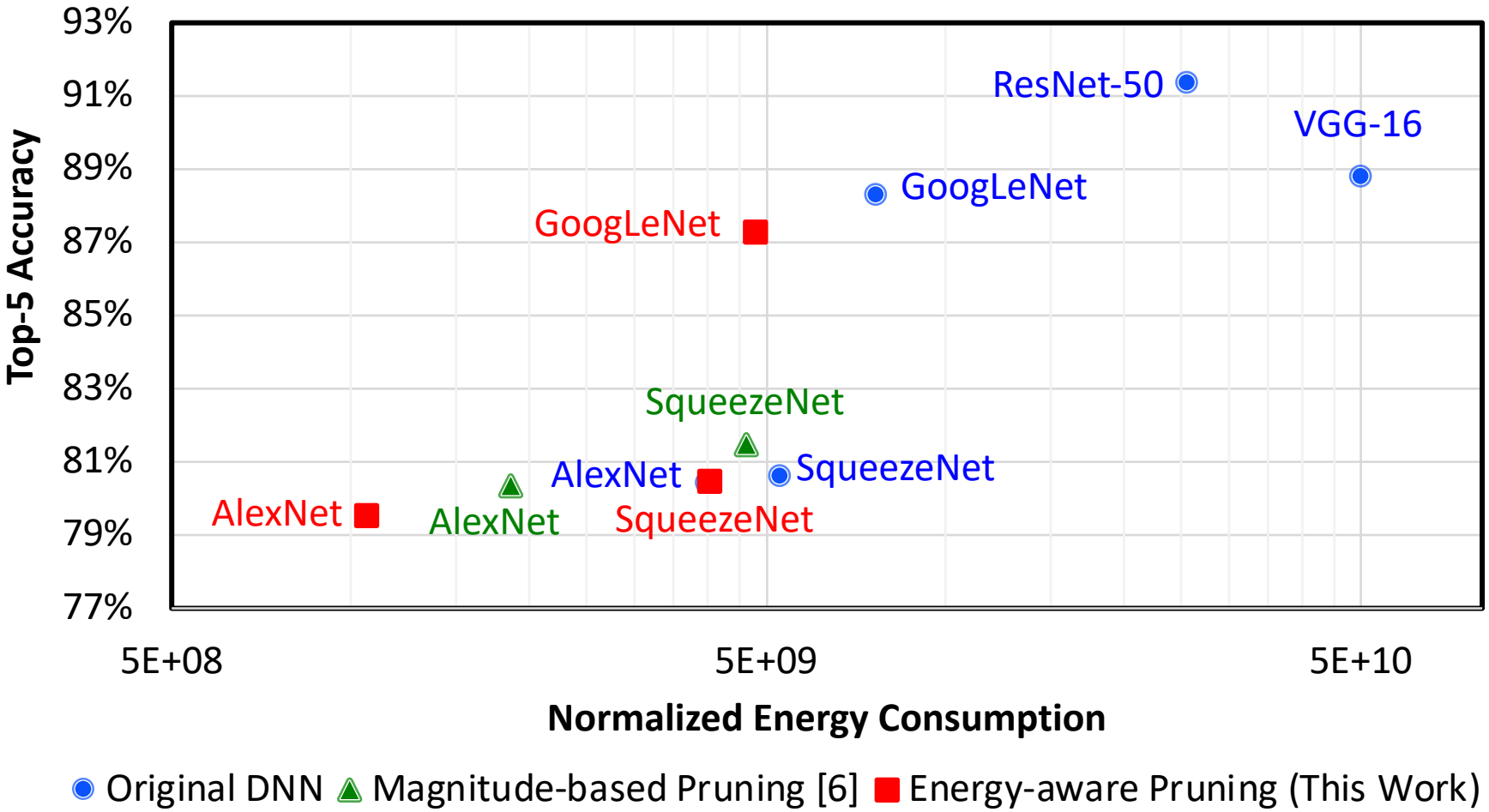
Magnitude-based Weight Pruning



Reduce number of weights by removing small magnitude weights

[Yang et al., CVPR 2017]

Energy-Aware Weight Pruning



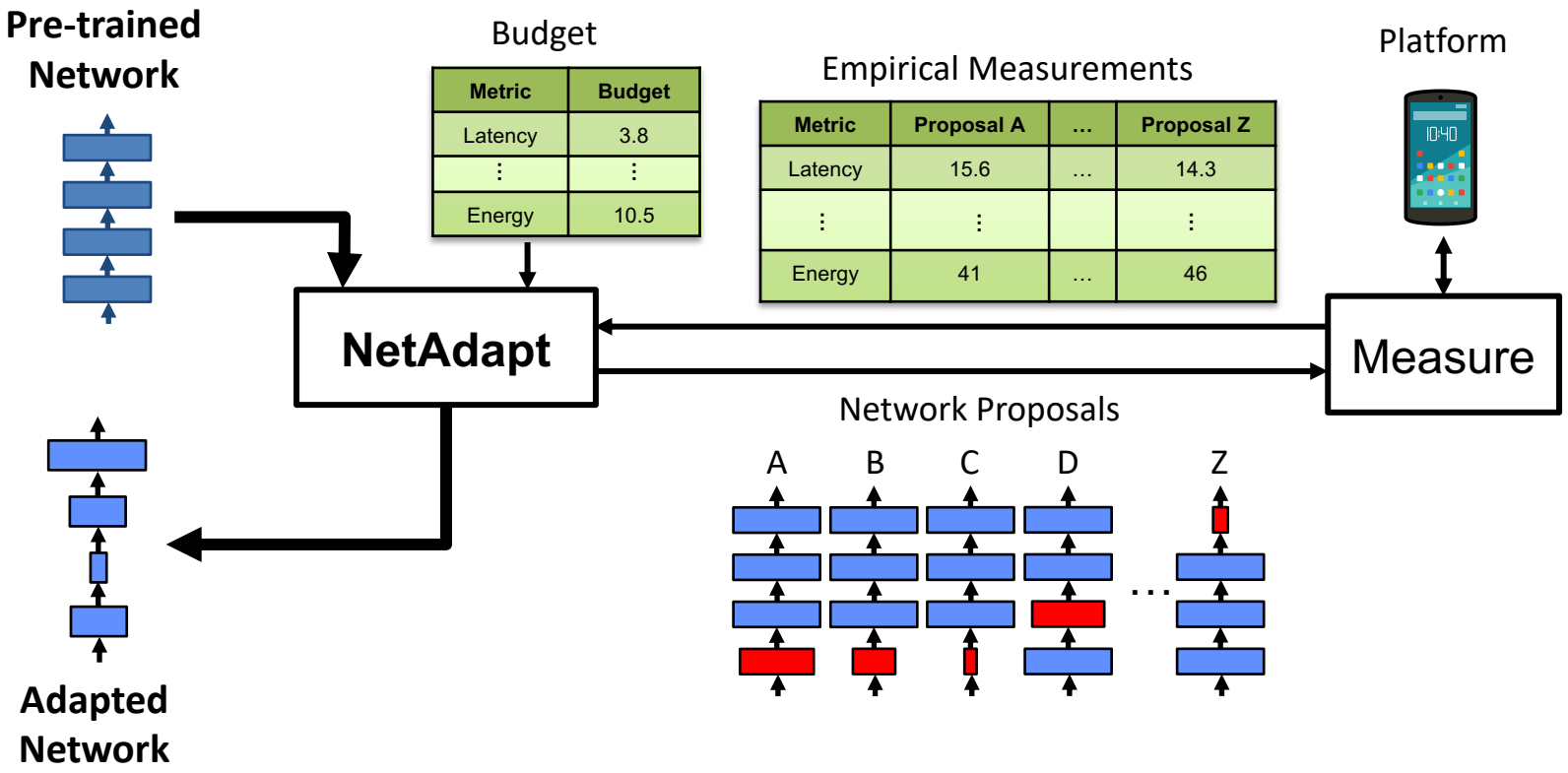
Remove weights from layers in order of highest to lowest energy

3.7x reduction in AlexNet / **1.6x** reduction in GoogLeNet

[Yang et al., CVPR 2017]

NetAdapt – Platform-Aware DNN Adaptation

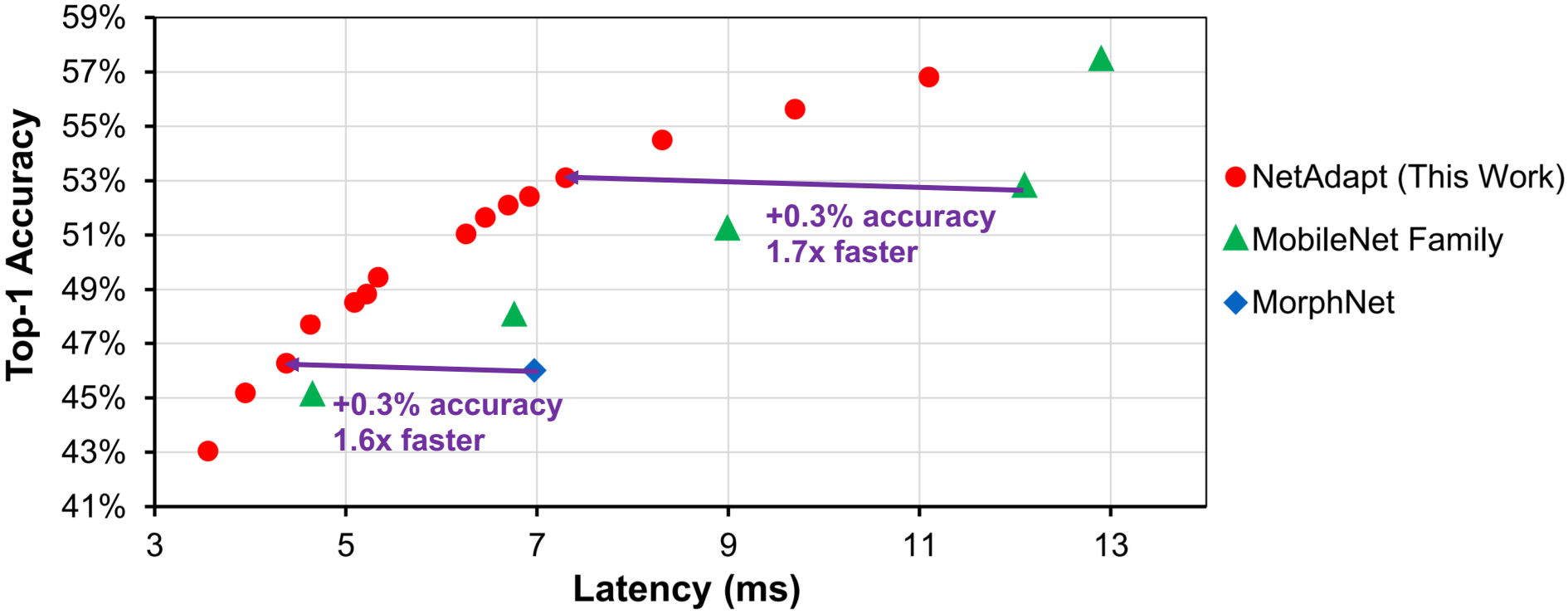
- **Automatically adapt DNN** to a mobile platform to reach a target latency or energy budget
- Use **empirical measurements** to guide optimization (avoid modeling of tool chain or platform architecture)



[Yang et al., arXiv 2018]

Latency vs. Accuracy Trade-off

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

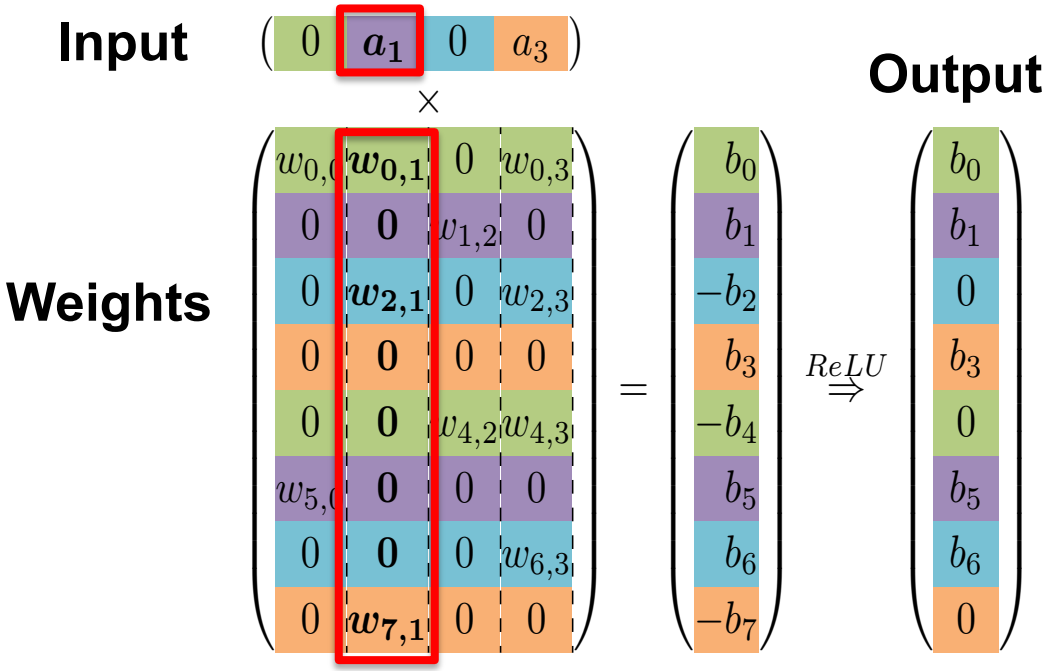


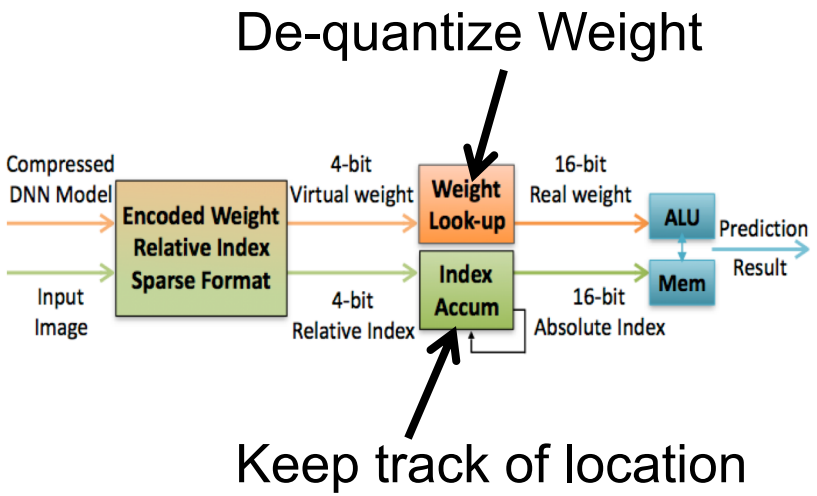
* Tested on the ImageNet dataset and a Google Pixel 1 CPU

[Yang et al., arXiv 2018]

Sparse Architecture Example: EIE

- Supports pruned **FC Layers**
- Store weights in Compressed Sparse Column (CSC) format
- Read corresponding weight column when input is non-zero



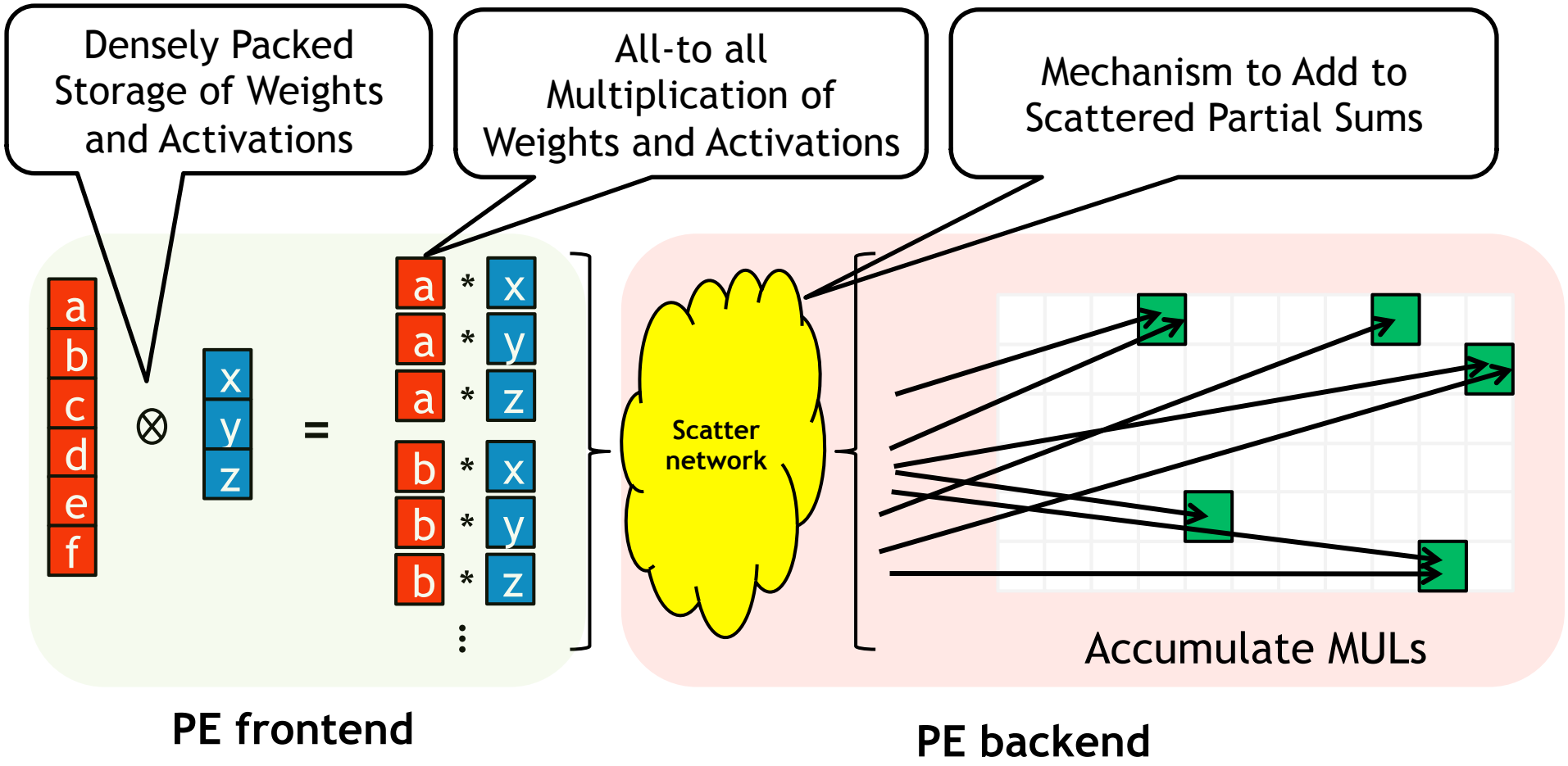


Output Stationary Dataflow

[Han et al., ISCA 2016]

Sparse Architecture Example: SCNN

- Supports pruned **CONV Layers**



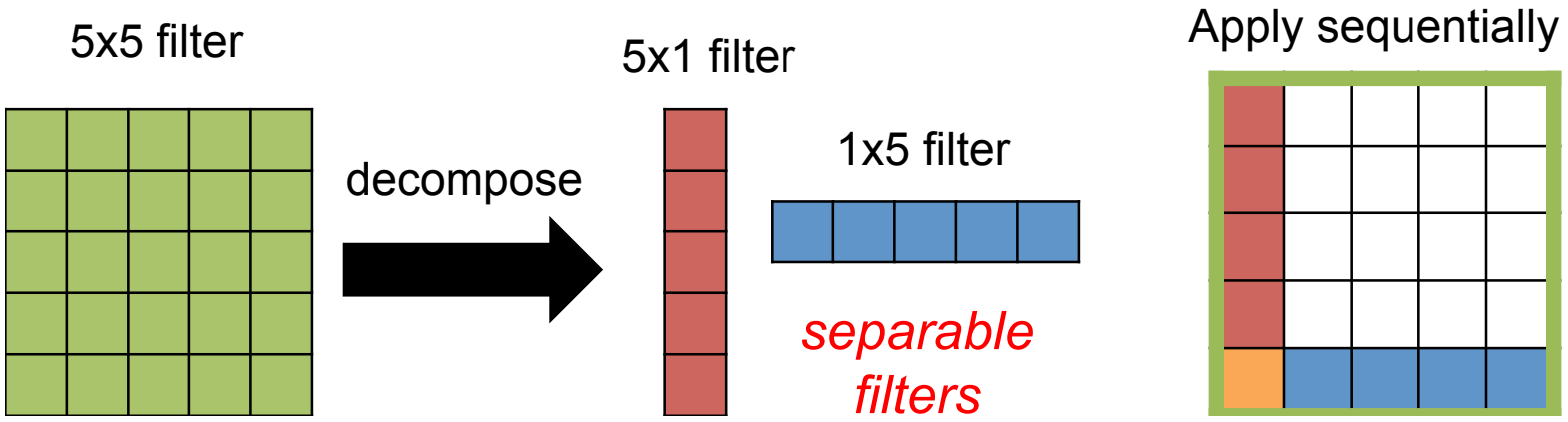
Input Stationary Dataflow

[Parashar et al., ISCA 2017]

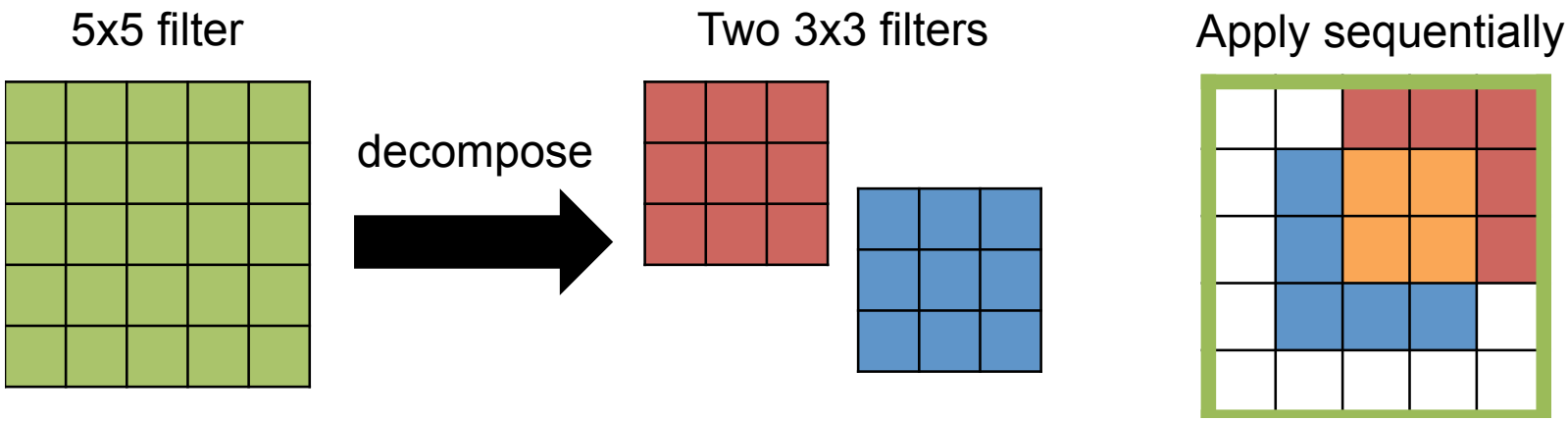
Compact Network Architecture Design

Build Network with series of Small Filters

GoogLeNet / Inception v3

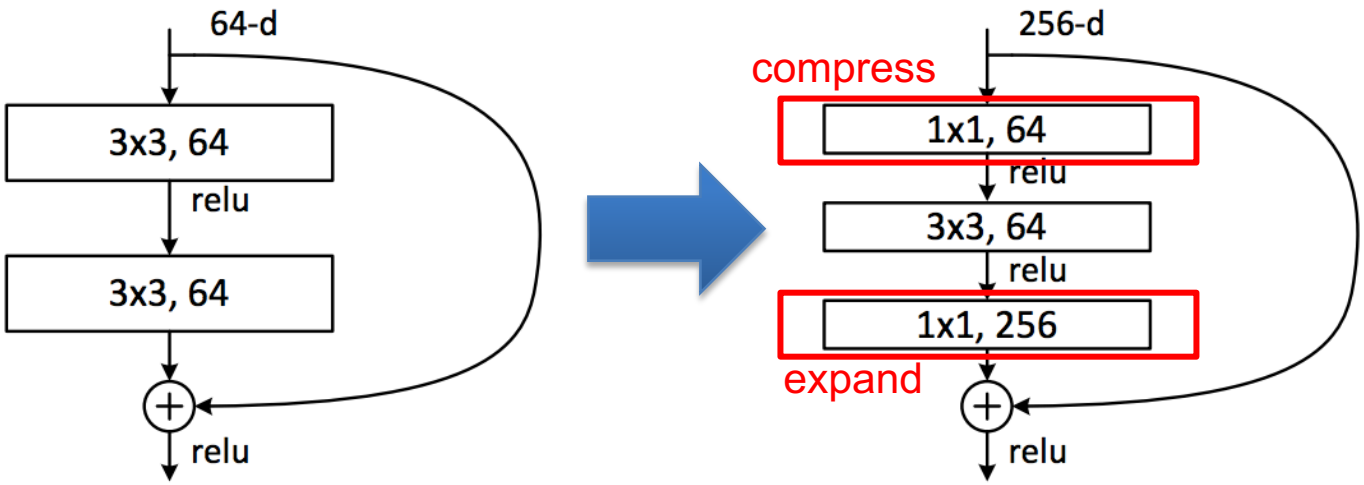


VGG-16

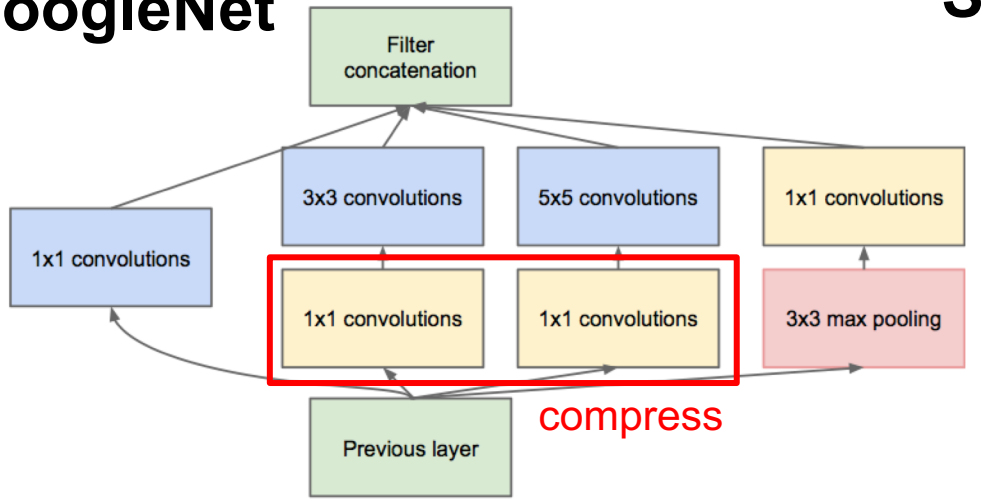


1×1 Bottleneck Layers are Popular

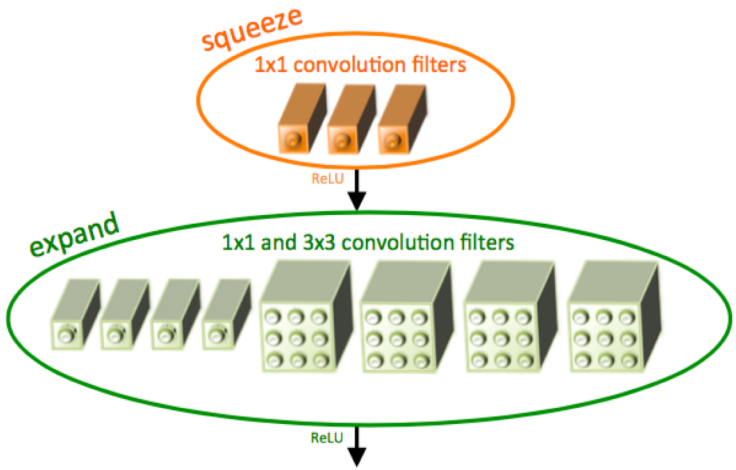
ResNet



GoogleNet



SqueezeNet



Summary

- **Dataflow** is one of the key attributes that define the architecture of DNN accelerators

- Co-design of **network model**, **dataflow**, and **hardware** is critical for the optimization of **performance, energy efficiency** and **flexibility** for DNN accelerators.

- It's important to consider a **comprehensive set of metrics** when evaluating different DNN solutions: **accuracy, flexibility, speed, energy, and cost**

Thank You

Resources:

Efficient Processing of Deep Neural Networks: A Tutorial and Survey
V.Sze, Y.-H. Chen, T.-J. Yang, J. Emer
Proceedings of IEEE
https://arxiv.org/abs/1703.09039

Eyeriss Website
More info on Eyeriss, Energy Modeling, Energy-Aware Pruning, NetAdapt, etc.
http://eyeriss.mit.edu