# Efficient Edge Solutions for Deep Learning Applications


## Vivienne Sze

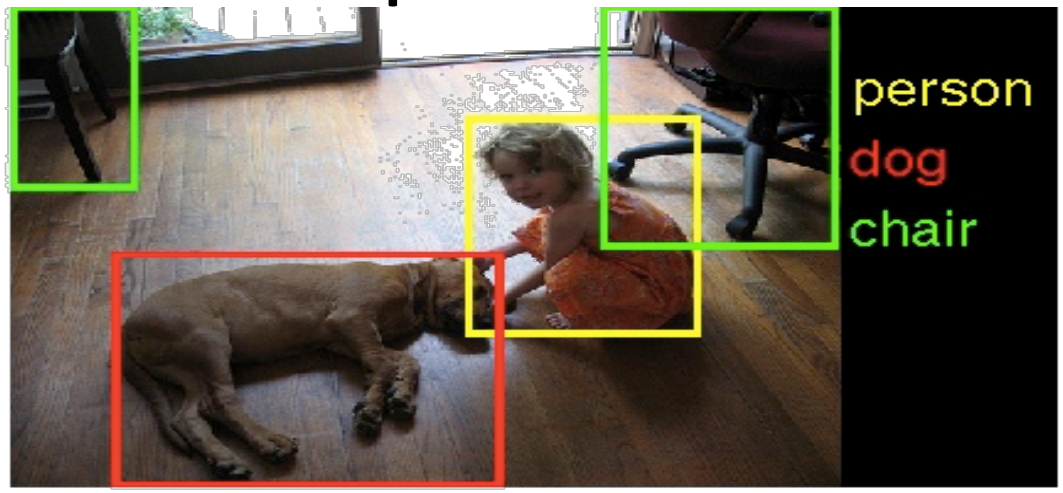
**Massachusetts Institute of Technology**

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

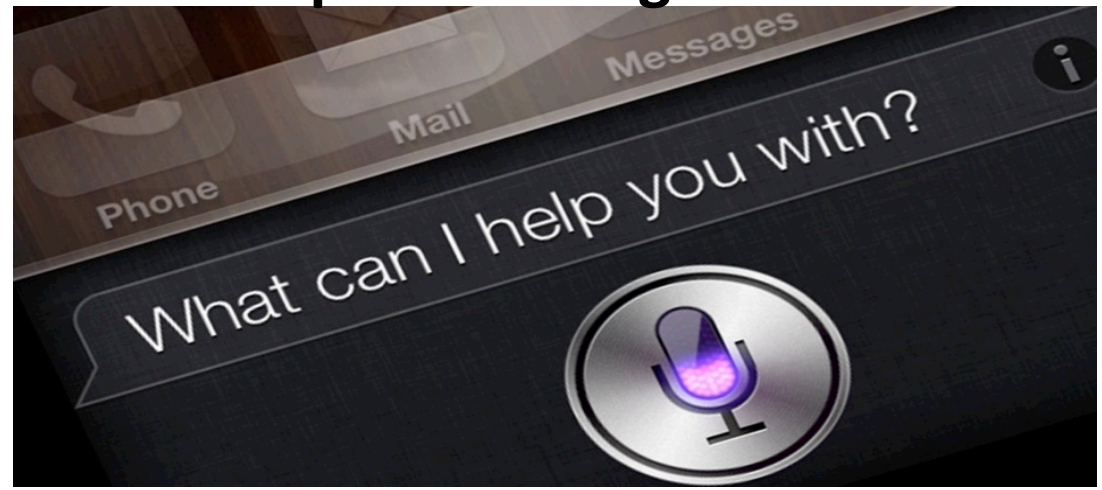
Contact Info
email: sze@mit.edu
website: www.rle.mit.edu/eems

# Example Applications of Deep Learning

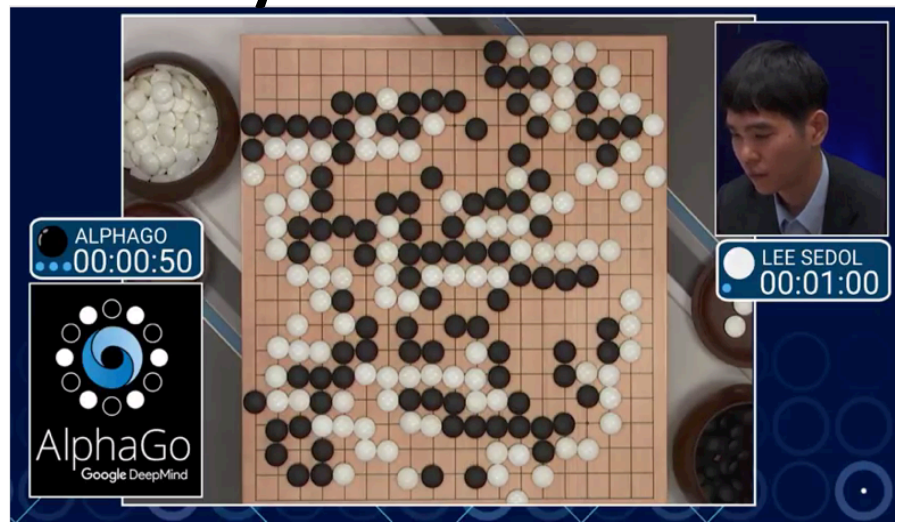
## Computer Vision



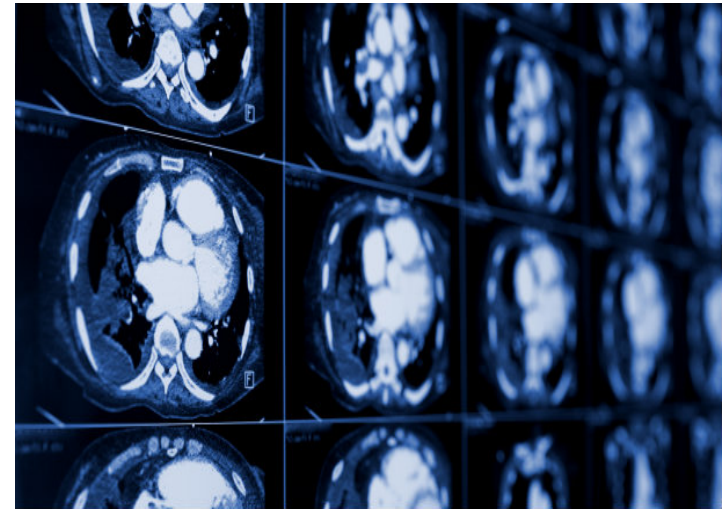

## Speech Recognition




## Game Play

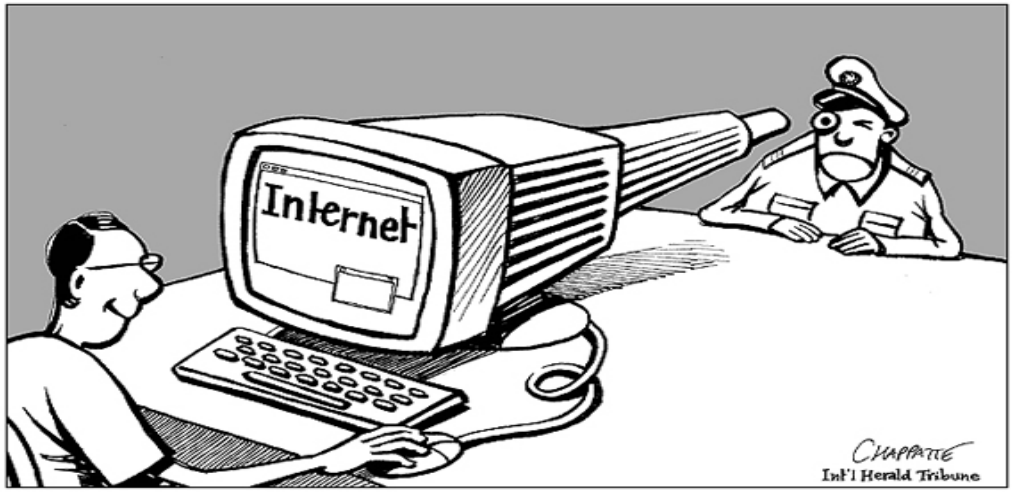




## Medical



# Processing at "Edge" instead of the "Cloud"

## Privacy




## Communication


Image source: www.theregister.co.uk

## Latency

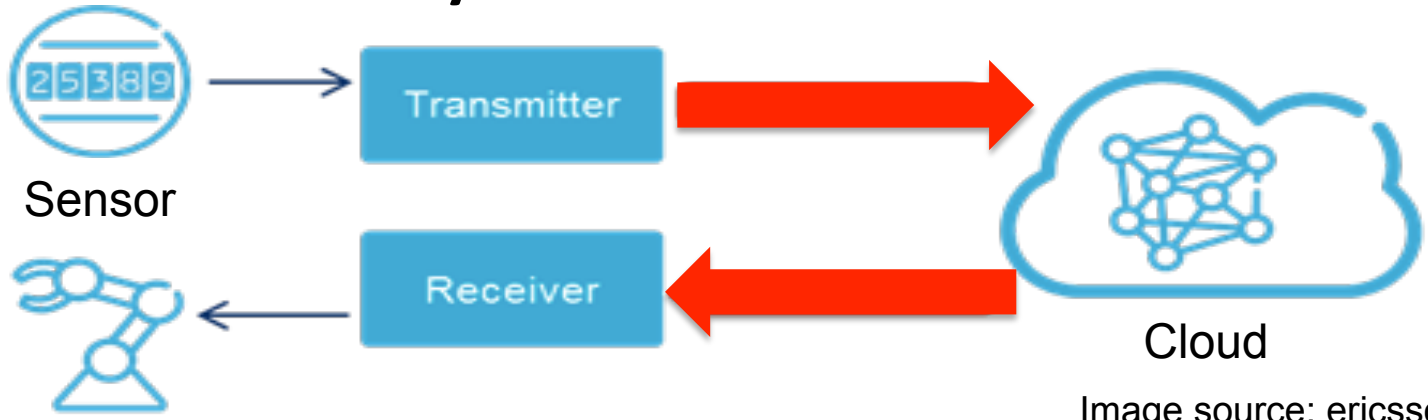
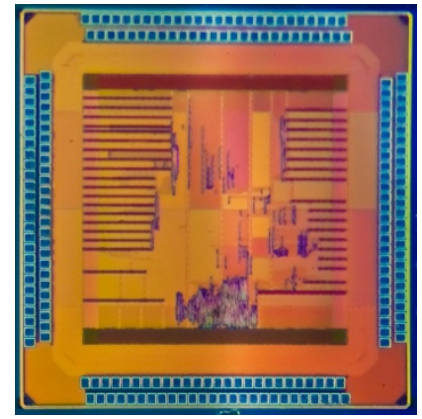


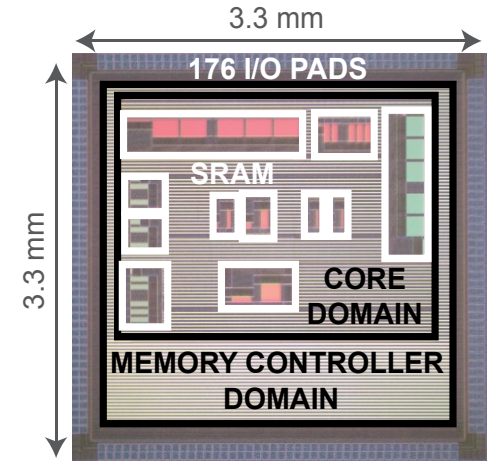

Image source: ericsson.com

# Typical Constraints on Edge Processing

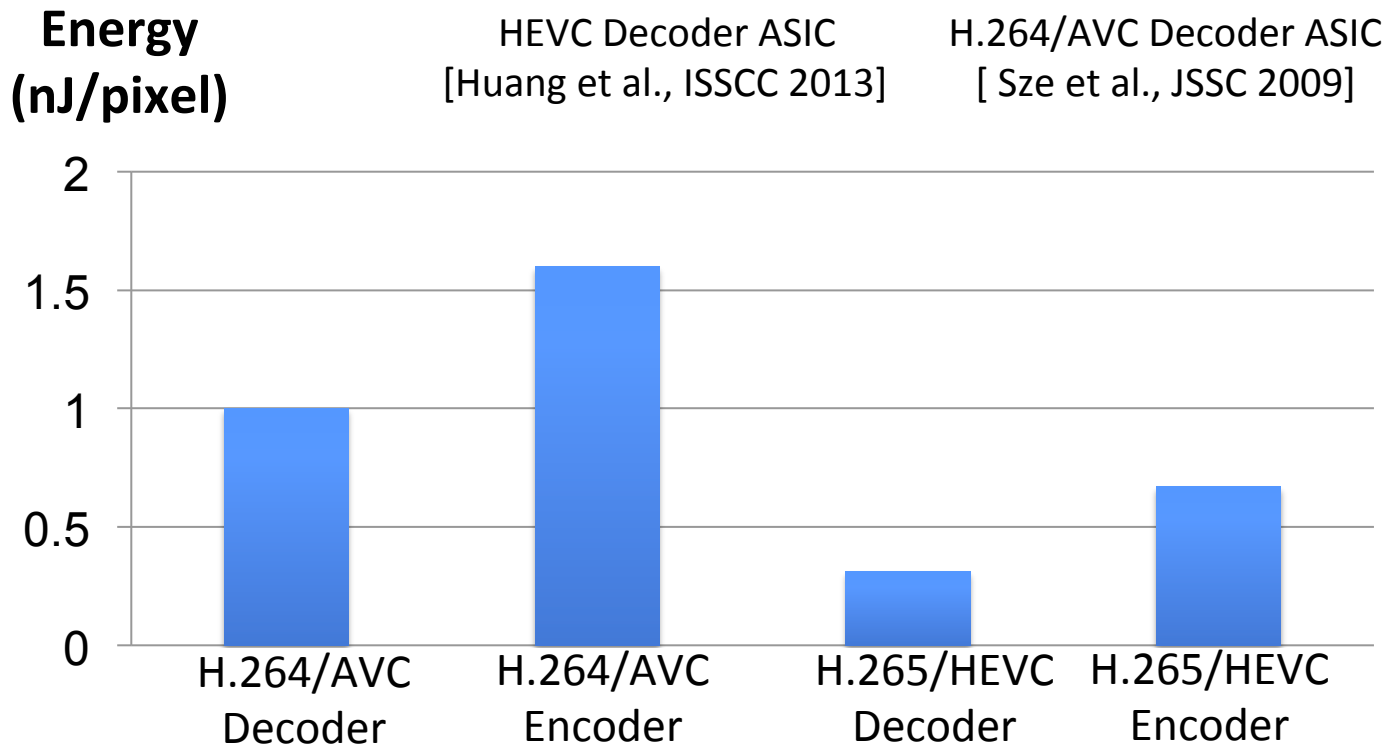
- Use **Video Compression** as a baseline
- **Area cost**
  - Memory Size 100-500kB
- **Power budget**
  - < 1W for smartphones
- **Throughput**
  - Real-time 30 fps
- **Energy**
  - ~1nJ/pixel



HEVC Decoder ASIC
[Huang et al., ISSCC 2013]

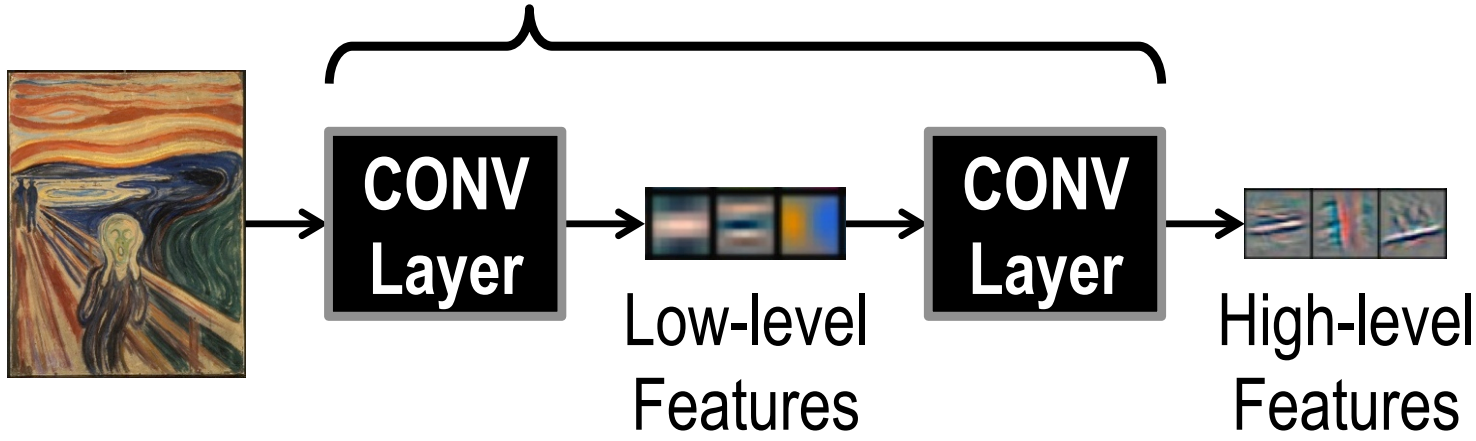



H.264/AVC Decoder ASIC
[ Sze et al., JSSC 2009]

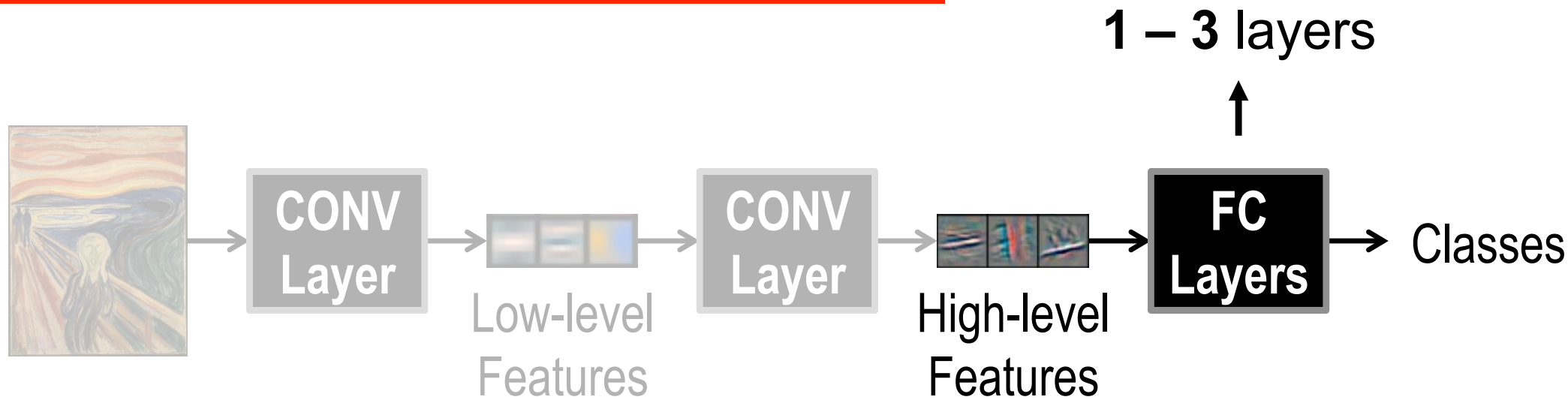


# Deep Convolutional Neural Networks

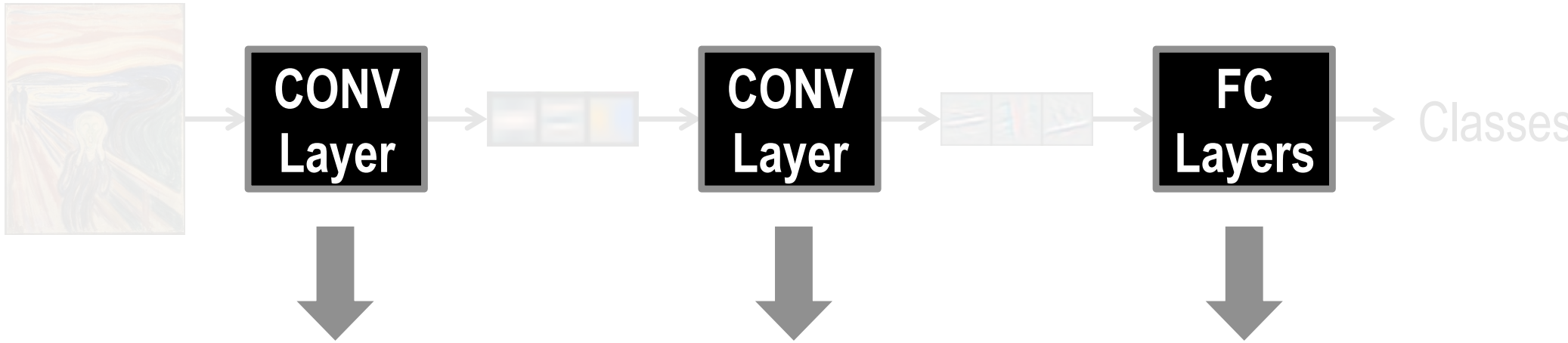
Modern ***deep*** CNN: up to **1000** CONV layers

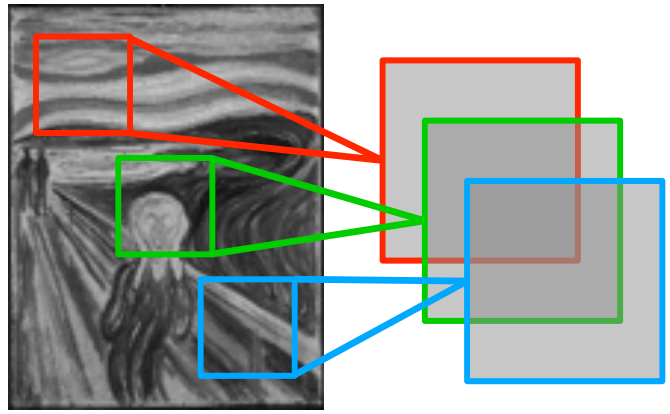


# Deep Convolutional Neural Networks



# Deep Convolutional Neural Networks



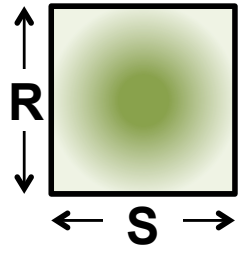



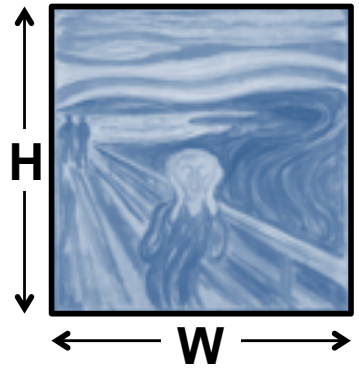
**Convolutions** account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

# High-Dimensional CNN Convolution

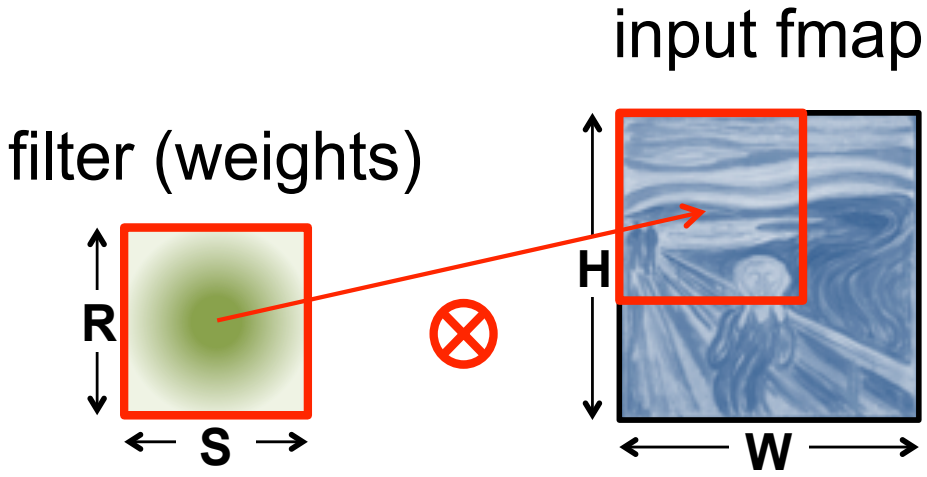
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)



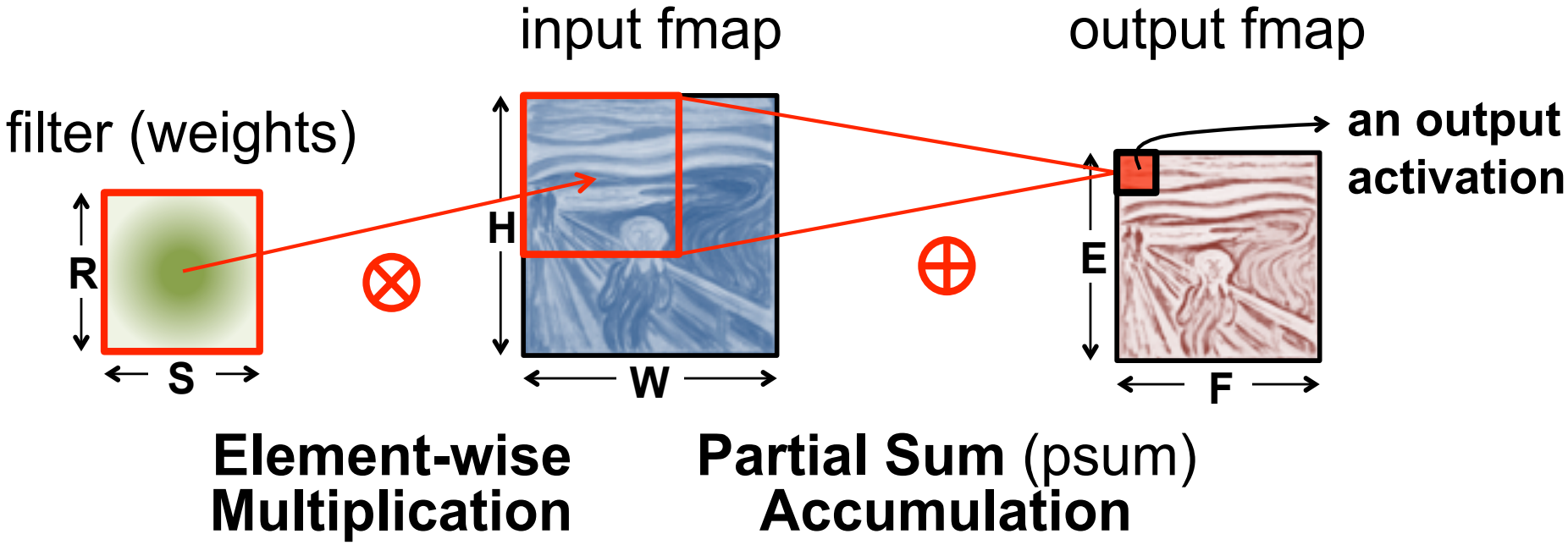



# High-Dimensional CNN Convolution

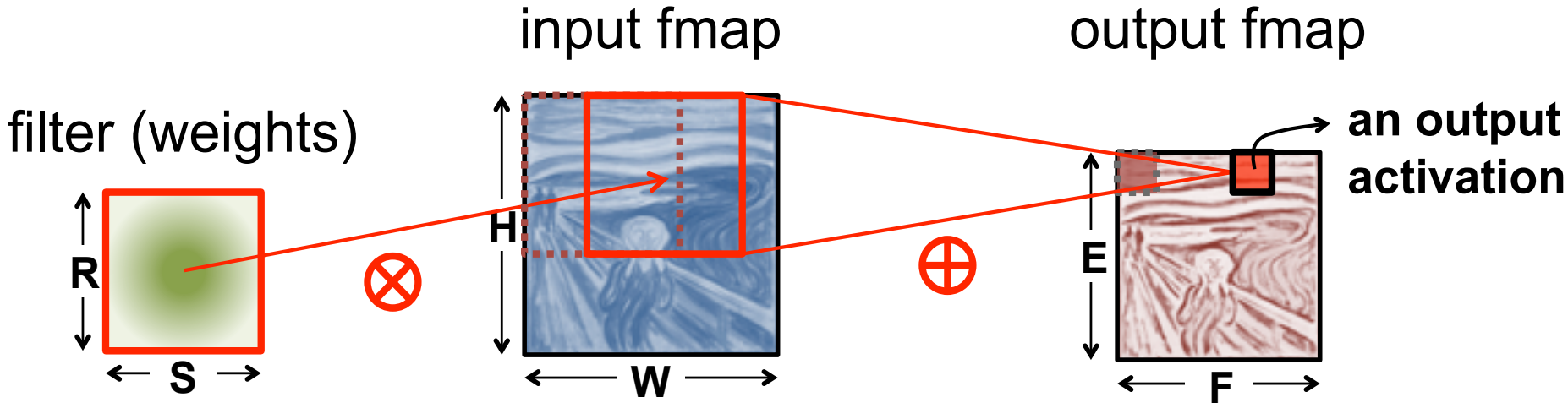



**Element-wise Multiplication**

# High-Dimensional CNN Convolution

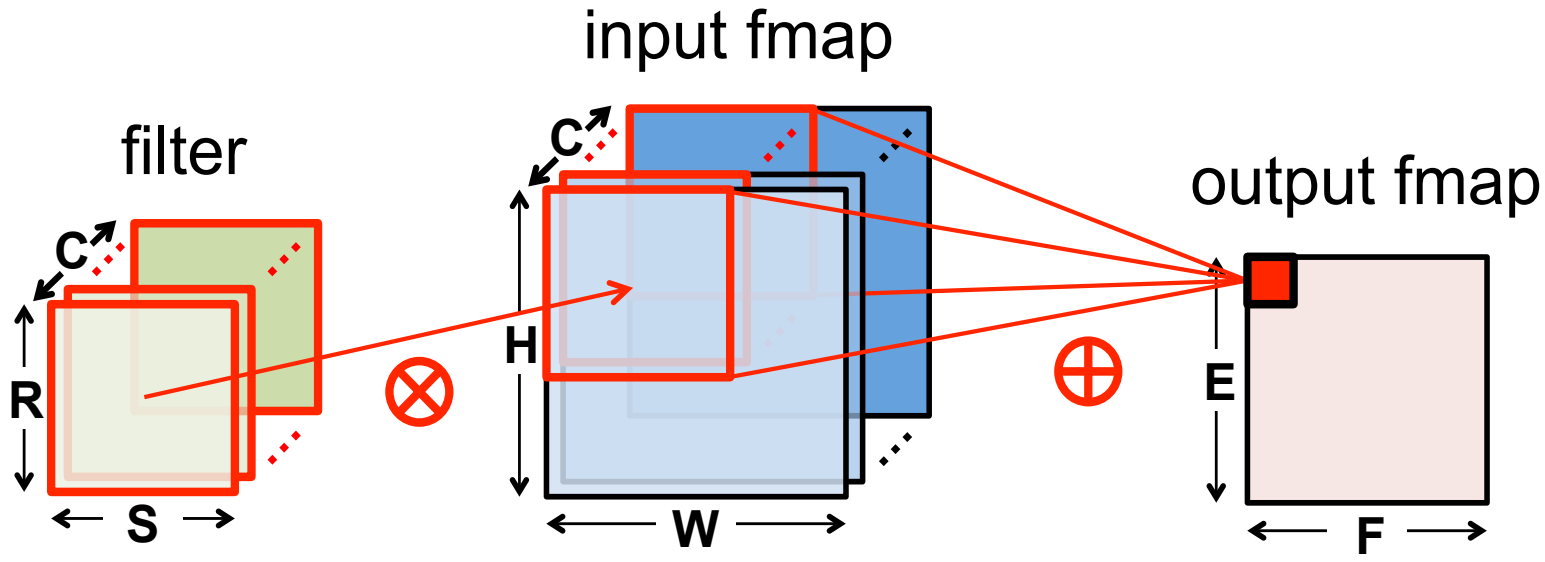


# High-Dimensional CNN Convolution




**Sliding Window Processing**

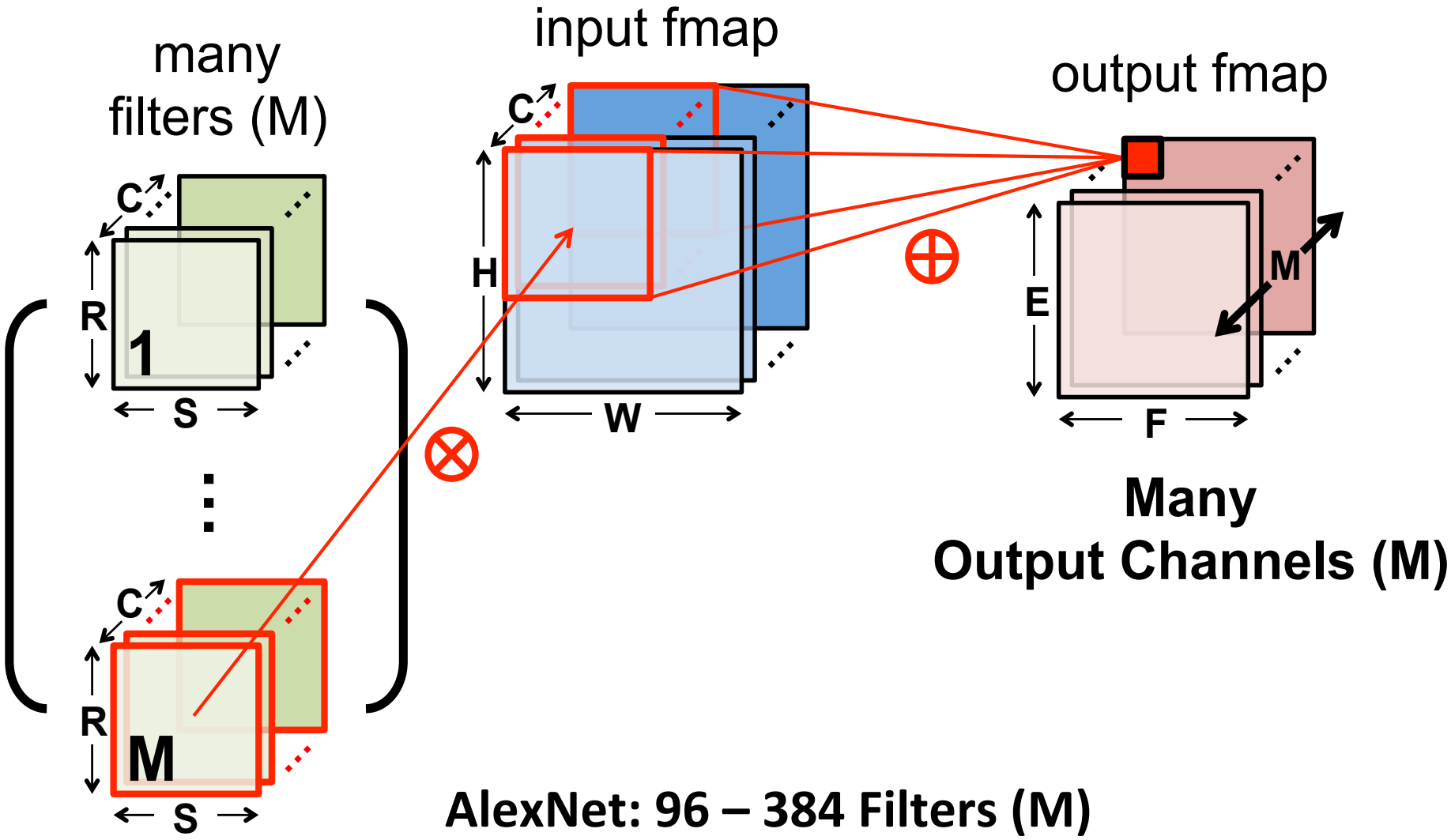
# High-Dimensional CNN Convolution




**Many Input Channels (C)**

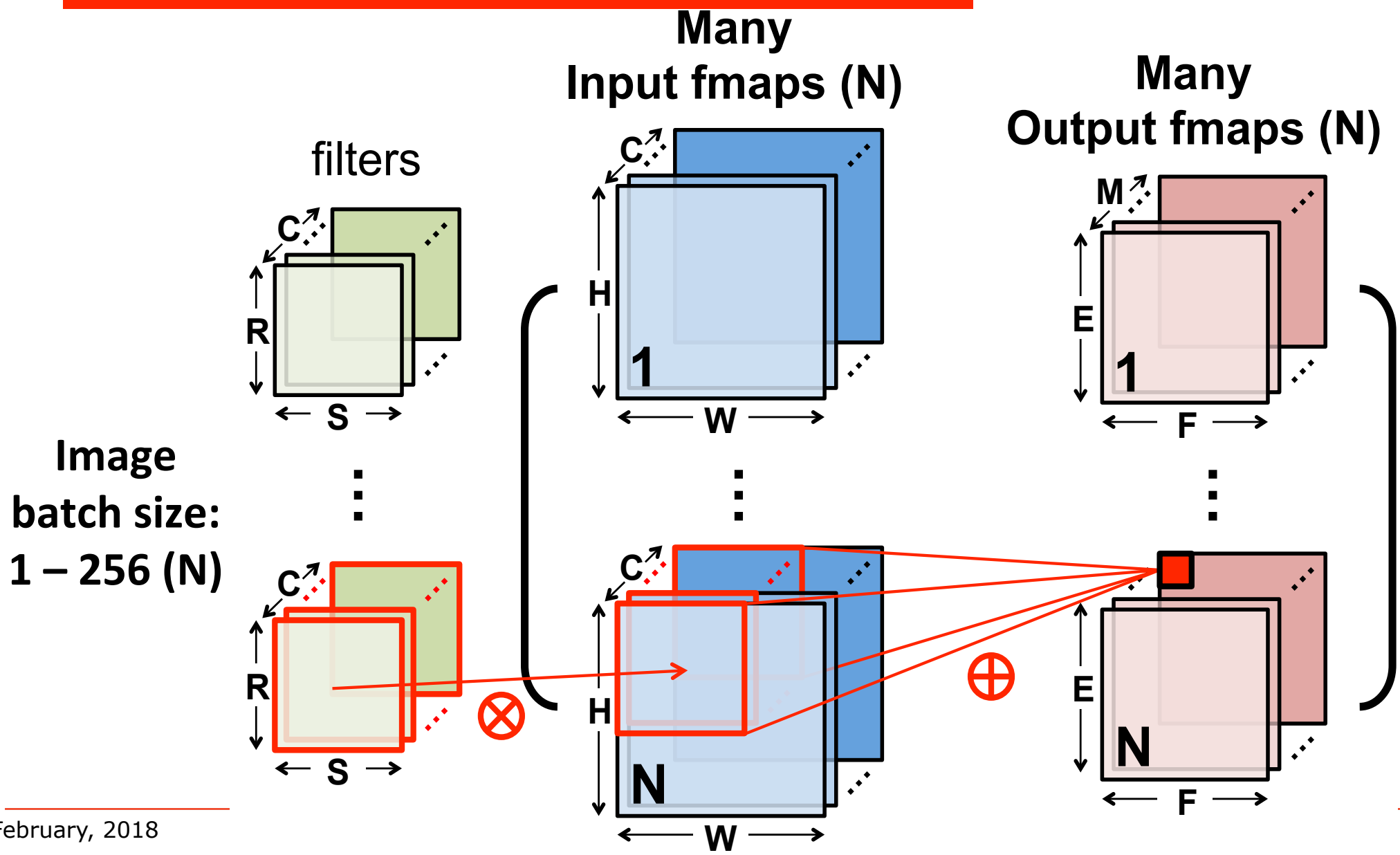
**AlexNet: 3 – 192 Channels (C)**

# High-Dimensional CNN Convolution




**AlexNet: 96 – 384 Filters (M)**

# High-Dimensional CNN Convolution



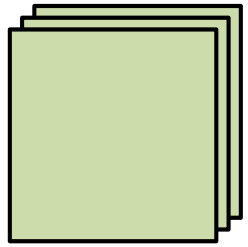
# Large Sizes with Varying Shapes

## AlexNet Convolutional Layer Configurations

| Layer | Filter Size (R) | # Filters (M) | # Channels (C) | Stride |
|---|---|---|---|---|
| **1** | 11x11 | 96 | 3 | 4 |
| **2** | 5x5 | 256 | 48 | 1 |
| **3** | 3x3 | 384 | 256 | 1 |
| **4** | 3x3 | 384 | 192 | 1 |
| **5** | 3x3 | 256 | 192 | 1 |

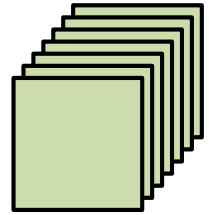
[Krizhevsky, NIPS 2012]

**Layer 1**



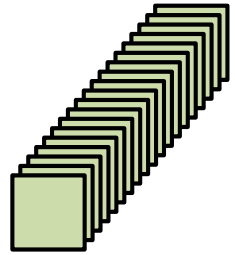
**34k Params**
**105M MACs**

**Layer 2**



**307k Params**
**224M MACs**

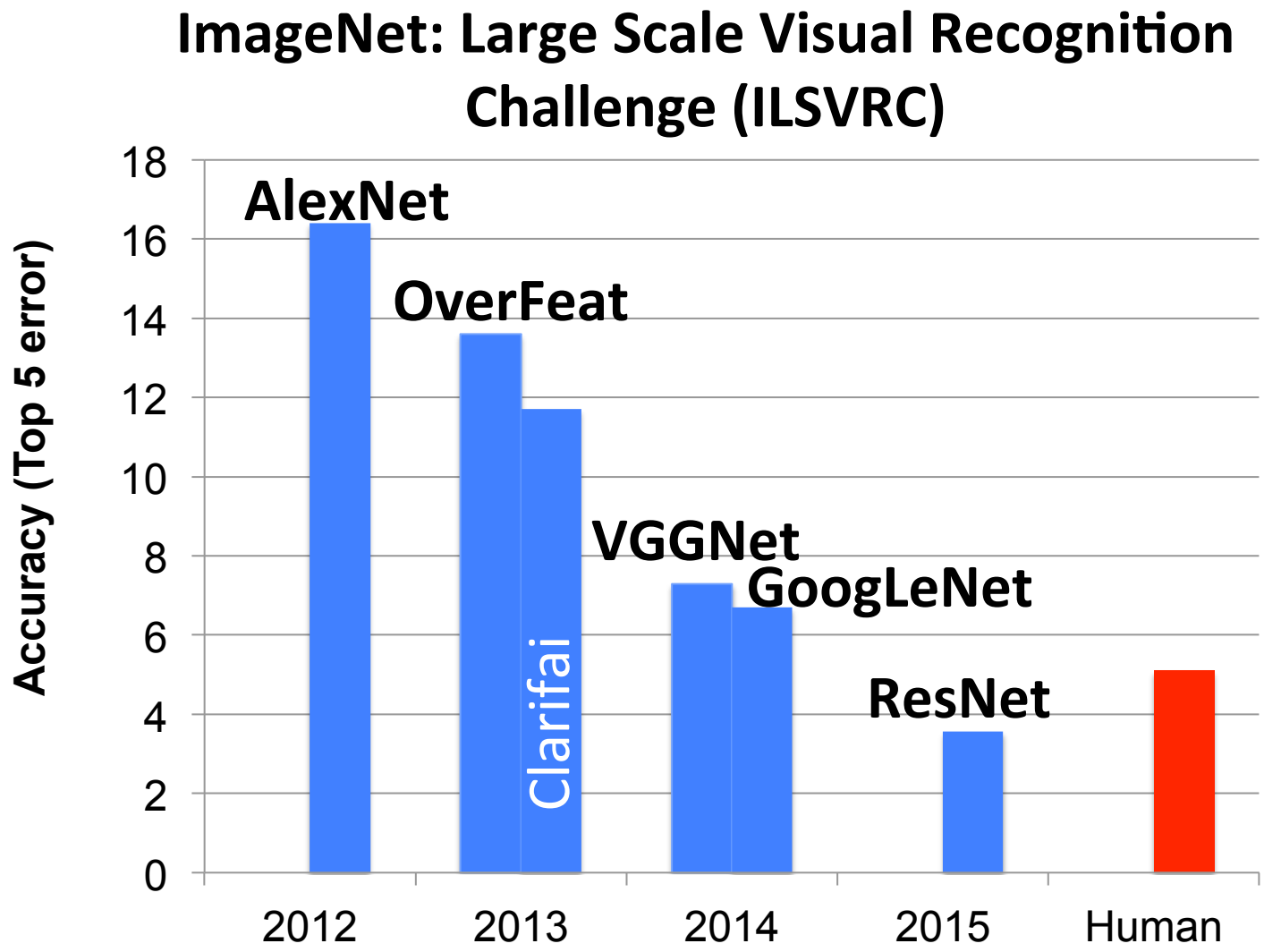
**Layer 3**



**885k Params**
**150M MACs**

# Popular DNNs

- LeNet (1998)
- AlexNet (2012)
- OverFeat (2013)
- VGGNet (2014)
- GoogleNet (2014)
- ResNet (2015)




[O. Russakovsky et al., IJCV 2015]

# Summary of Popular DNNs

| Metrics | LeNet-5 | AlexNet | VGG-16 | GoogLeNet (v1) | ResNet-50 |
|---|---|---|---|---|---|
| Top-5 error | n/a | 16.4 | 7.4 | 6.7 | 5.3 |
| Input Size | 28x28 | 227x227 | 224x224 | 224x224 | 224x224 |
| **# of CONV Layers** | **2** | **5** | **16** | **21 (depth)** | **49** |
| # of Weights | 2.6k | 2.3M | 14.7M | 6.0M | 23.5M |
| # of MACs | 283k | 666M | 15.3G | 1.43G | 3.86G |
| **# of FC layers** | **2** | **3** | **3** | **1** | **1** |
| # of Weights | 58k | 58.6M | 124M | 1M | 2M |
| # of MACs | 58k | 58.6M | 124M | 1M | 2M |
| **Total Weights** | **60k** | **61M** | **138M** | **7M** | **25.5M** |
| **Total MACs** | **341k** | **724M** | **15.5G** | **1.43G** | **3.9G** |

CONV Layers increasingly important!

# Complexity versus Difficulty of Task

- **Evaluate hardware using the appropriate DNN model and dataset**
  - Difficult tasks typically require larger models
  - Different datasets for different tasks

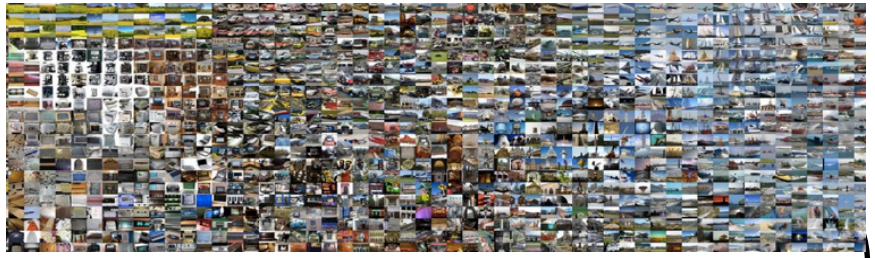
**MNIST**

**ImageNet**

# Training vs. Inference

**Training**
**(determine weights)**

**Large Datasets**



**Weights**

**Inference**
**(use weights)**

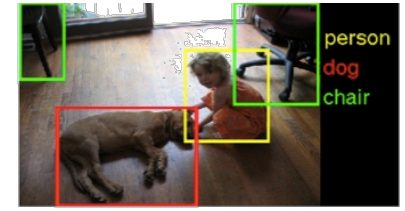
# Challenges

# Key Metrics

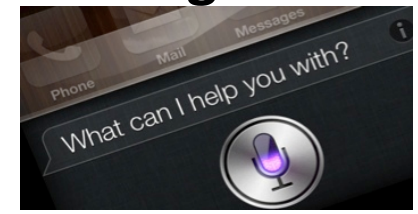
- **Accuracy**
  - Well defined dataset, DNN Model and task
- **Programmability**
  - Support various DNN Models with different filter weights
- **Energy/Power:**
  - Energy per operation and DRAM Bandwidth
- **Throughput/Latency**
  - GOPS, frame rate, delay, batch size
- **Cost**
  - Area (memory and logic size)

**ImageNet**

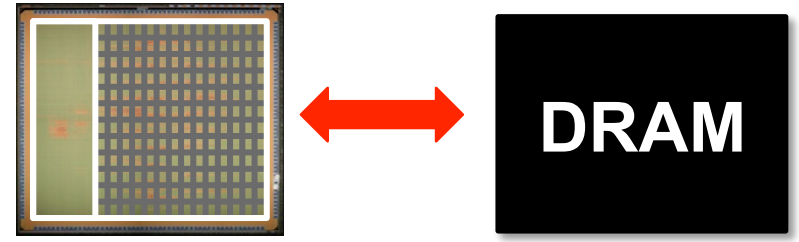
**Computer Vision**




**Speech Recognition**




**Chip**




[Sze et al., CICC 2017]

# Opportunities in Architecture

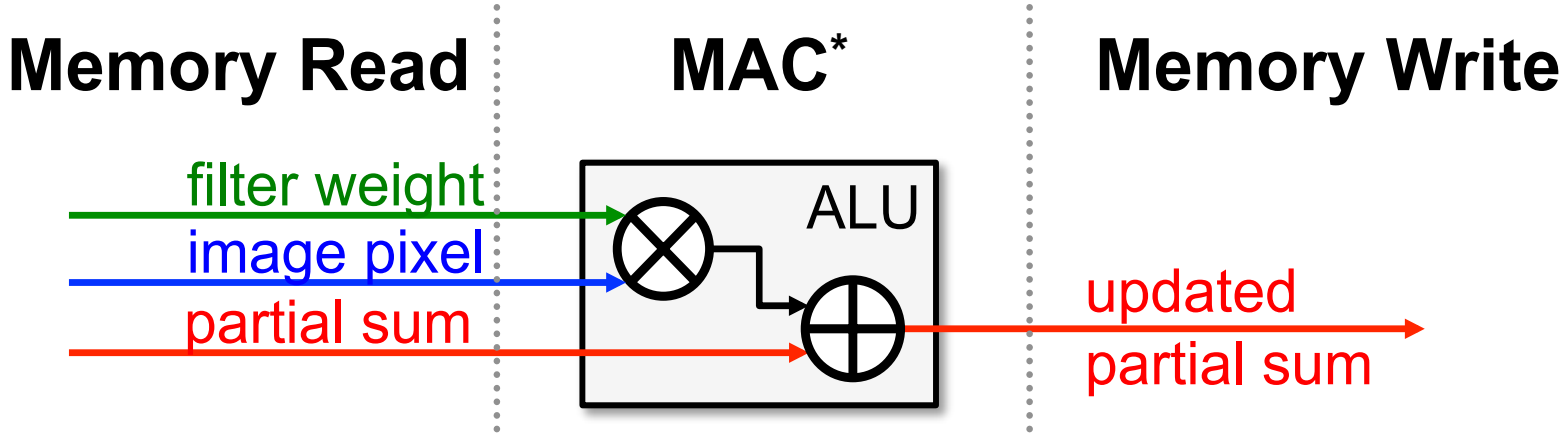
# Properties We Can Leverage

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

# Properties We Can Leverage

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

- Memory Access is the Bottleneck

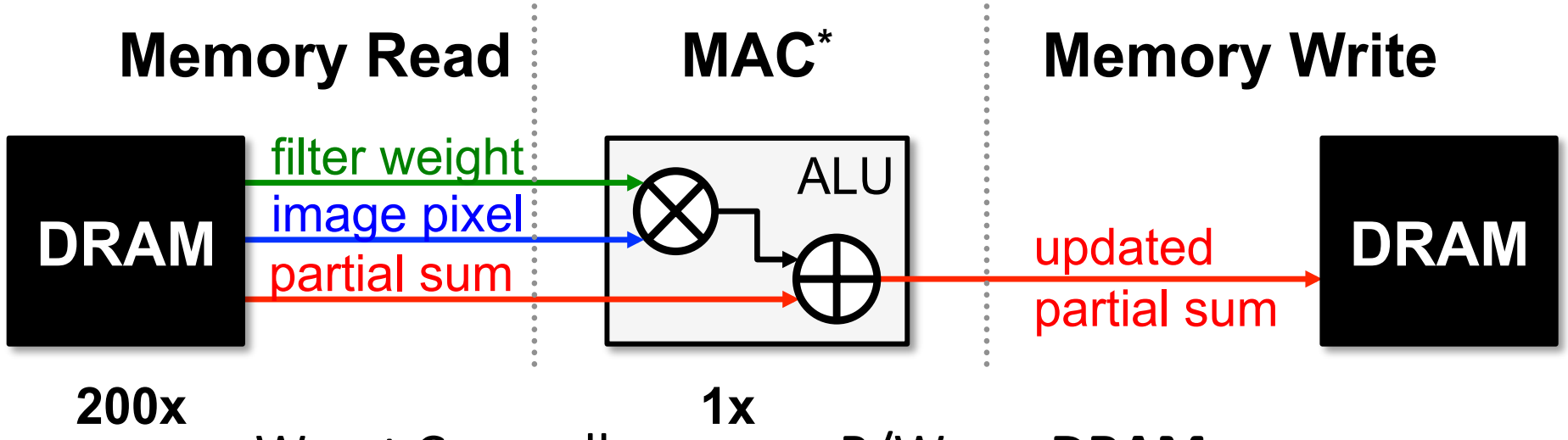



* multiply-and-accumulate

# Properties We Can Leverage

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

- Memory Access is the Bottleneck



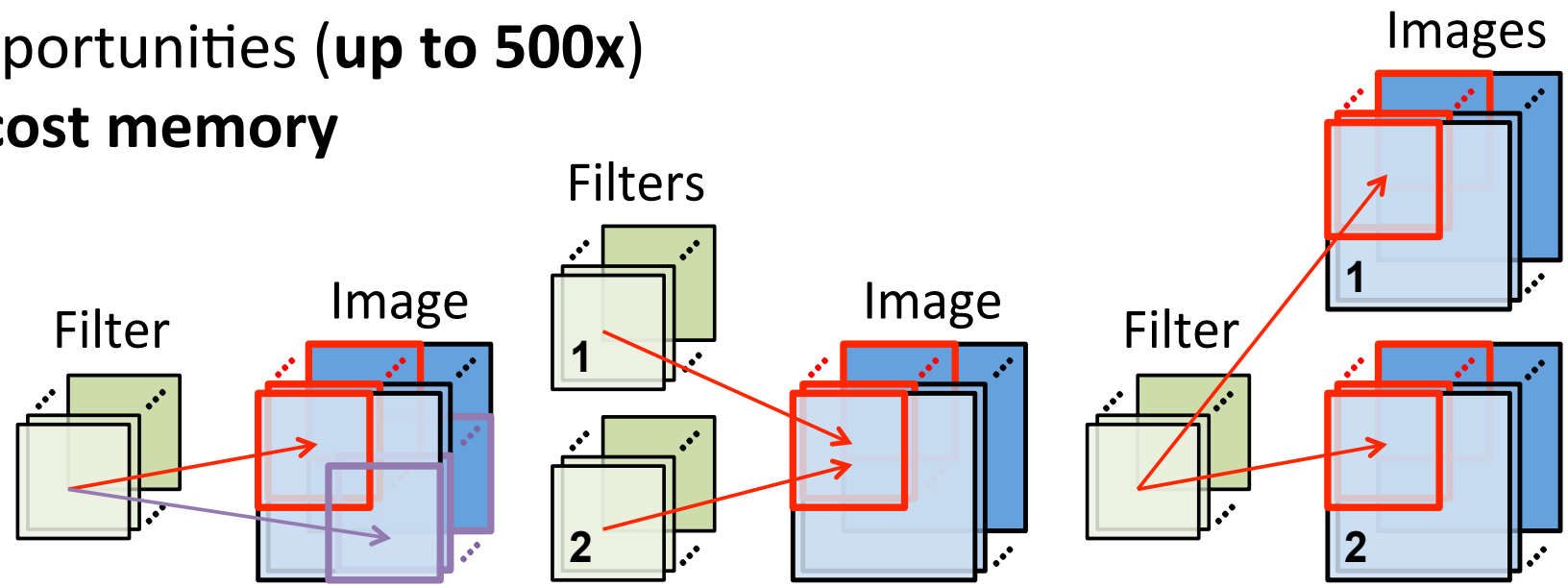

Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NIPS 2012] has **724M** MACs
  → **2896M** DRAM accesses required

# Properties We Can Leverage

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

- **Input data reuse** opportunities (**up to 500x**)
  → exploit **low-cost memory**



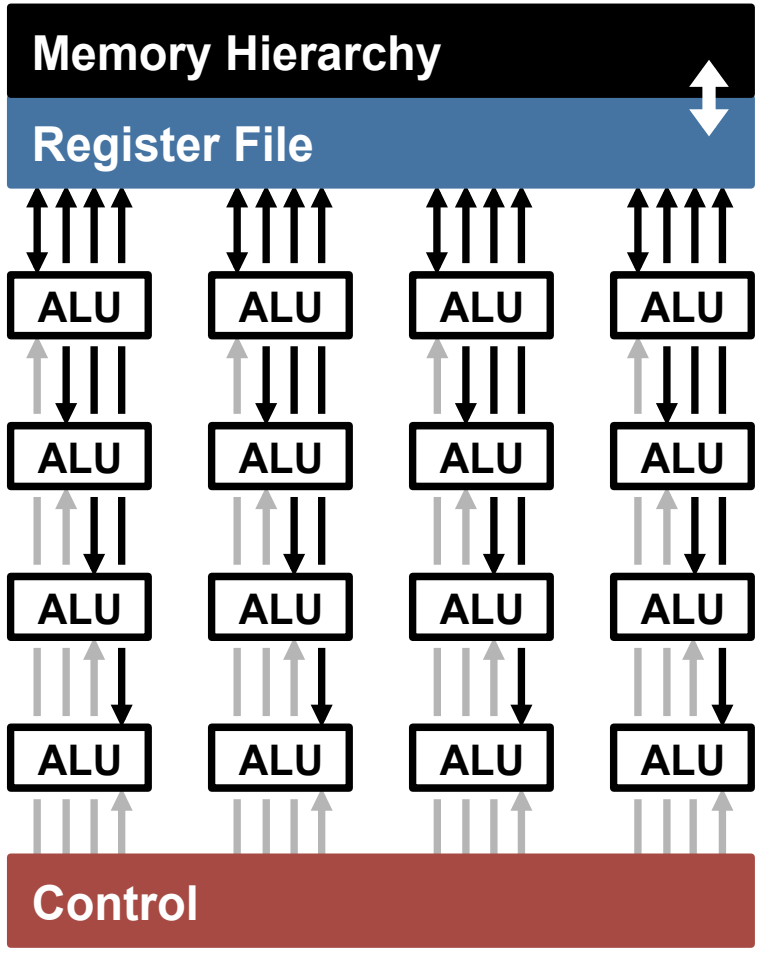

**Convolutional Reuse** (pixels, weights)

**Image Reuse** (pixels)

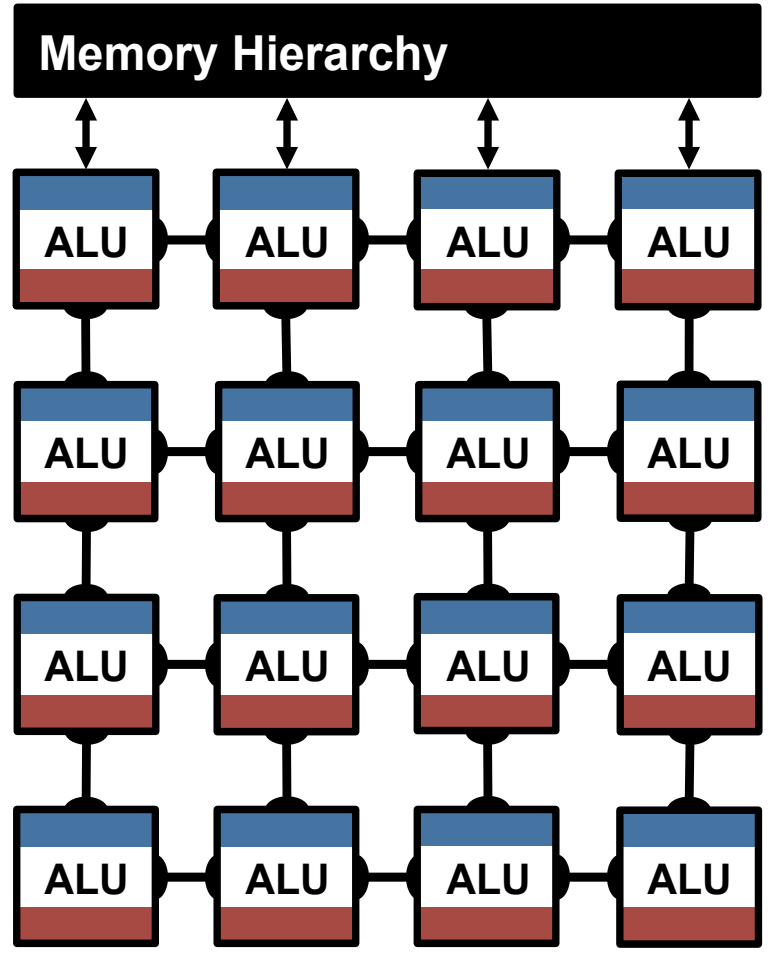
**Filter Reuse** (weights)

# Highly-Parallel Compute Paradigms

**Temporal Architecture (SIMD/SIMT)**




**Spatial Architecture (Dataflow Processing)**



# Advantages of Spatial Architecture

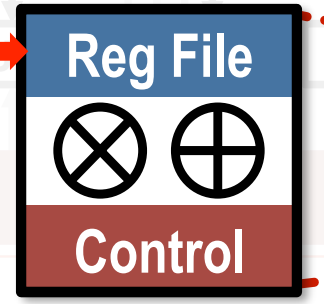
Temporal Architecture
(SIMD/SIMT)

**Efficient Data Reuse**
Distributed local storage (RF)

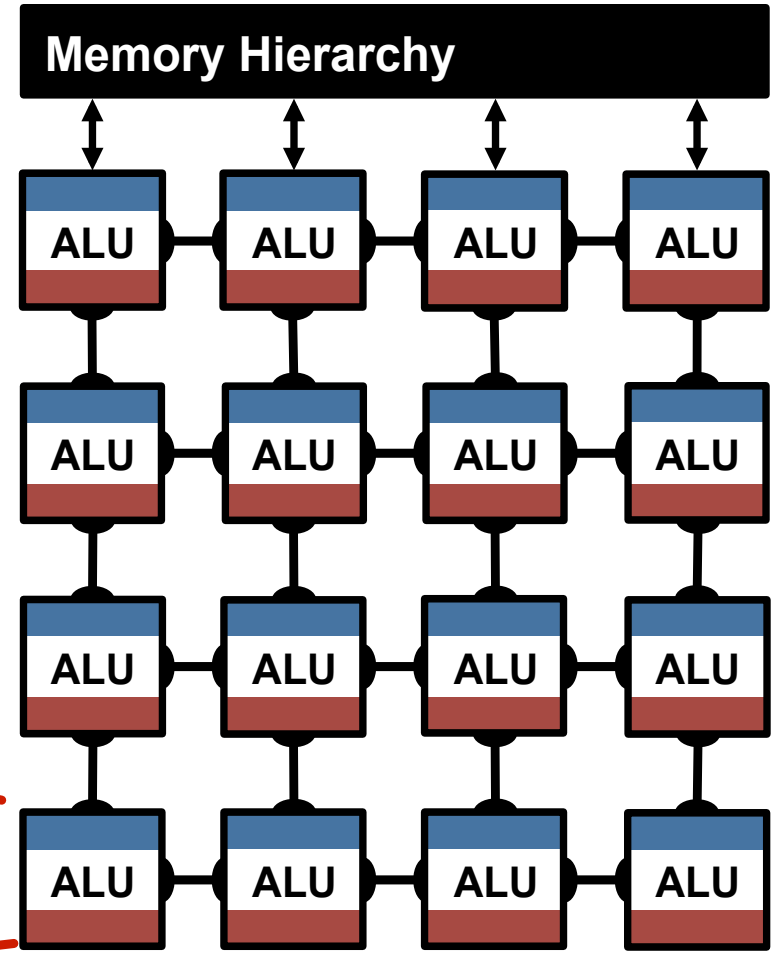
**Inter-PE Communication**
Sharing among regions of PEs

**Processing Element (PE)**

0.5 – 1.0 kB

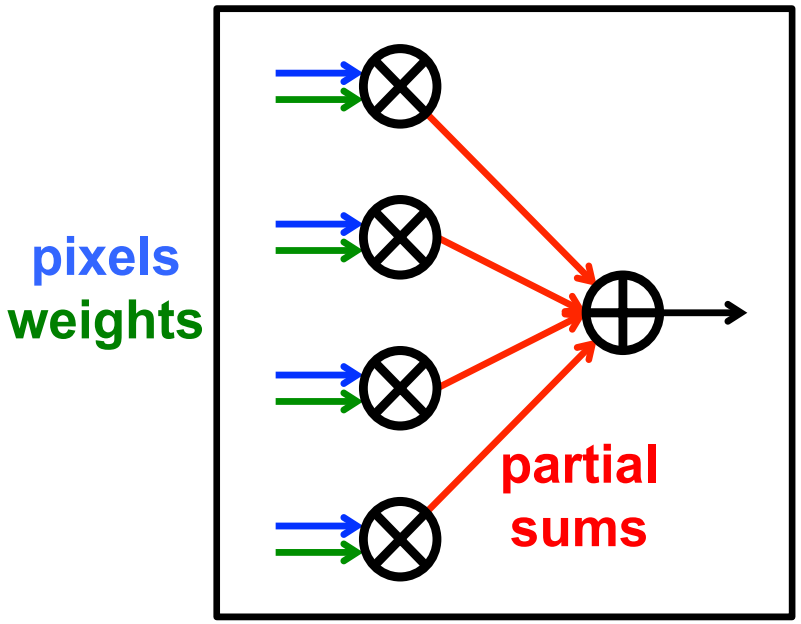



**Spatial Architecture (Dataflow Processing)**



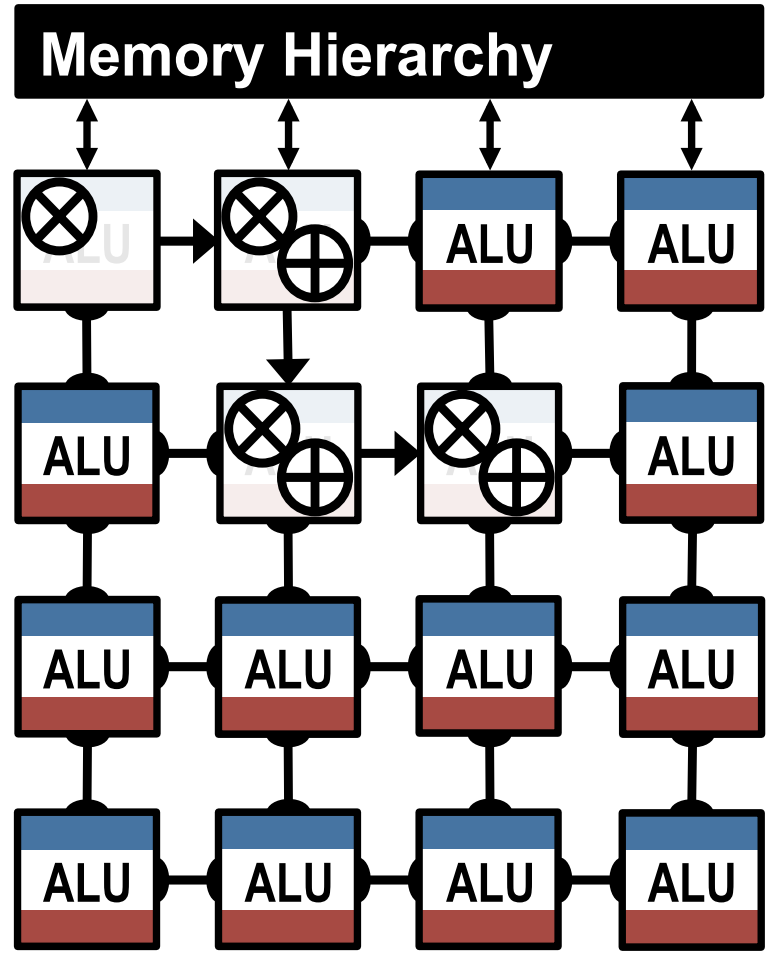
# How to Map the Dataflow?

CNN Convolution




?

Spatial Architecture
(Dataflow Processing)

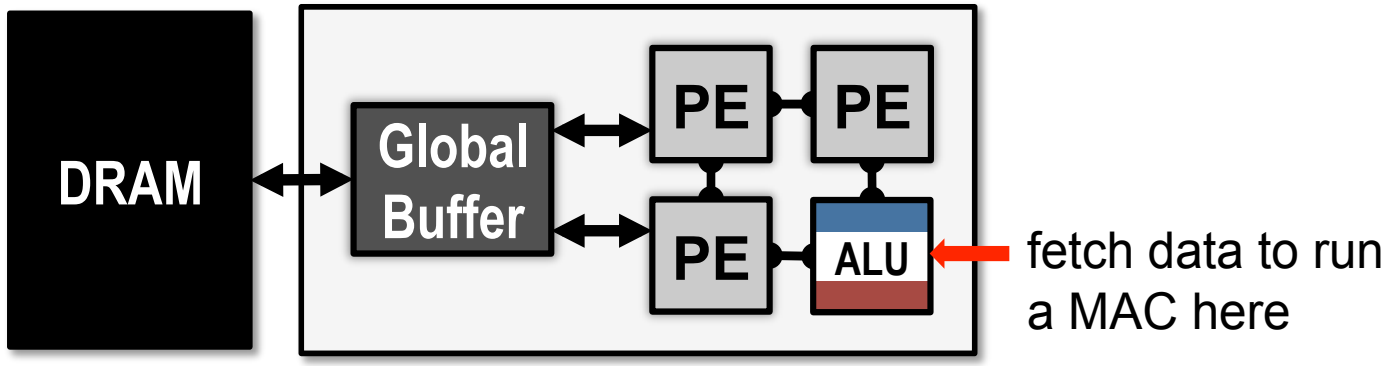


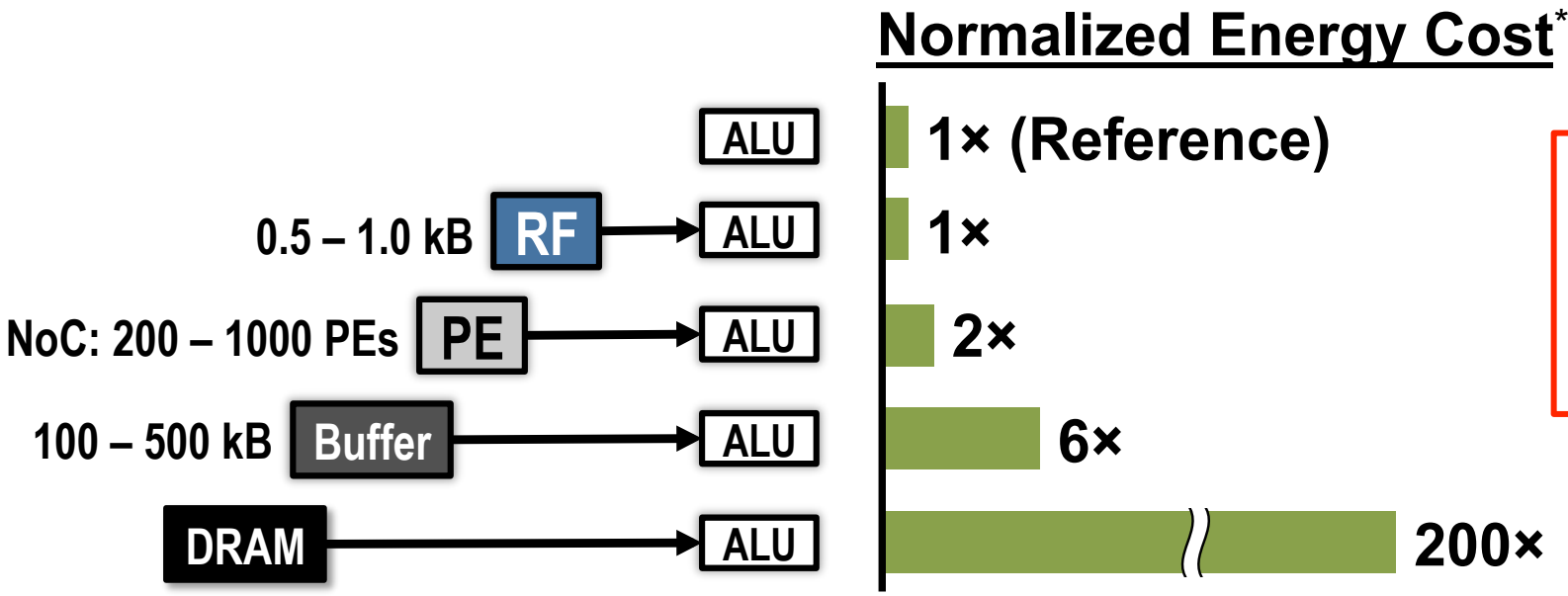

**Goal:** Increase reuse of input data (**weights** and **pixels**) and local **partial sums** accumulation

# Energy-Efficient Dataflow

[Chen et al., ISCA 2016]

# Data Movement is Expensive

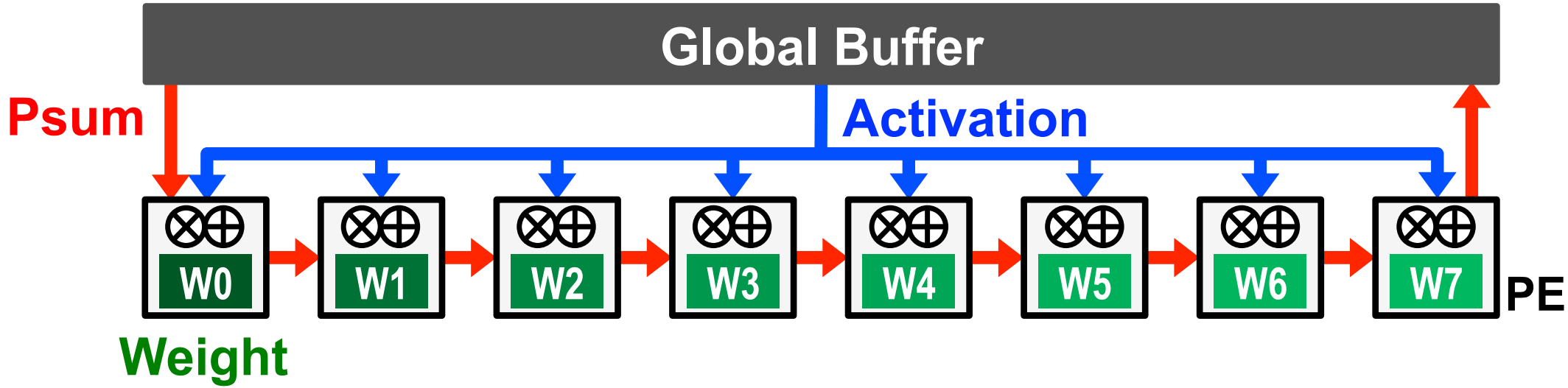



**Maximize data reuse** at low cost levels of hierarchy

* measured from a commercial 65nm process

# Weight Stationary (WS)



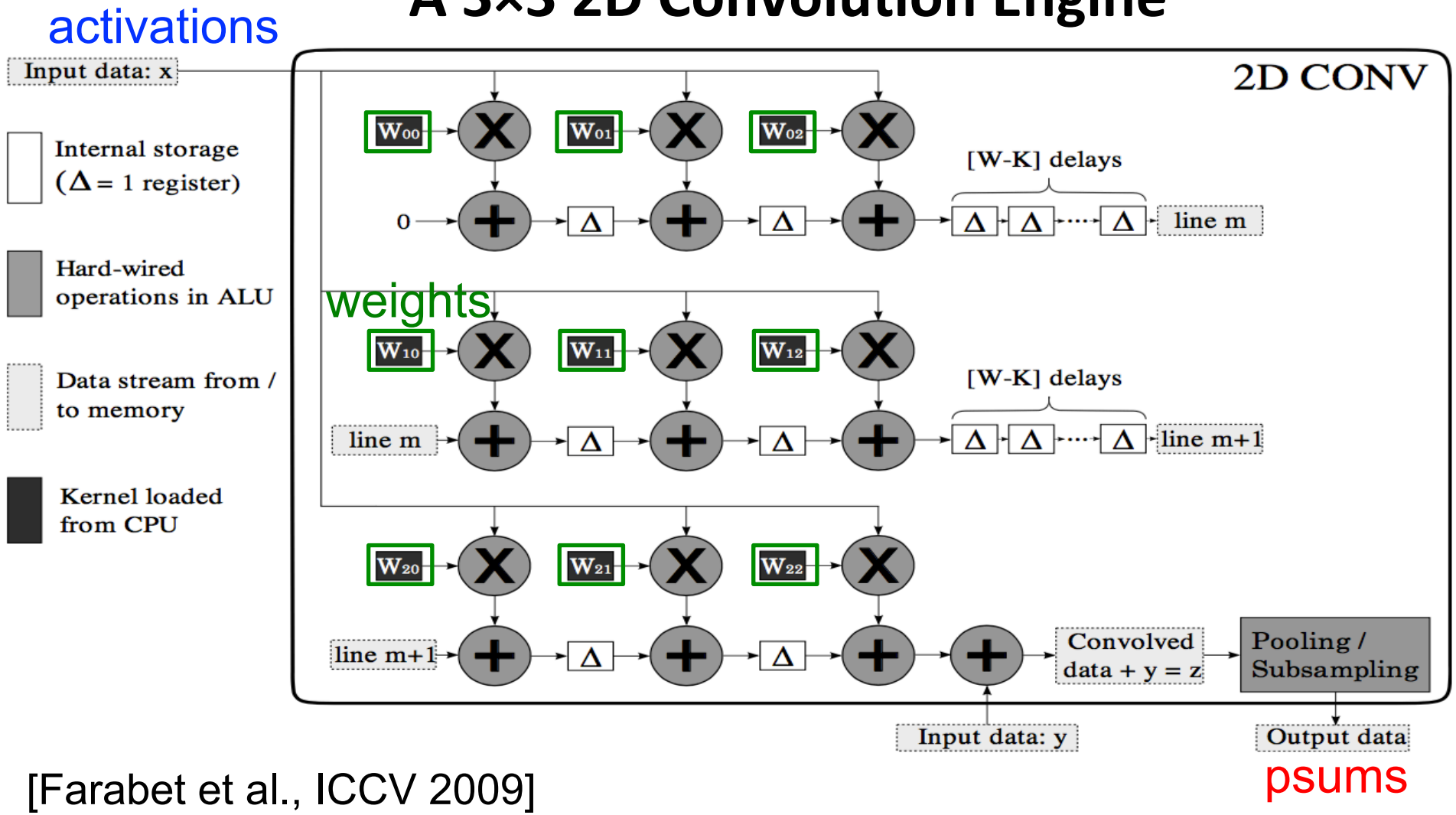

- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights

- **Examples:**

  [**Chakradhar**, *ISCA* 2010] [**nn-X (NeuFlow)**, *CVPRW* 2014]
  [**Park**, *ISSCC* 2015] [**Origami**, *GLSVLSI* 2015]

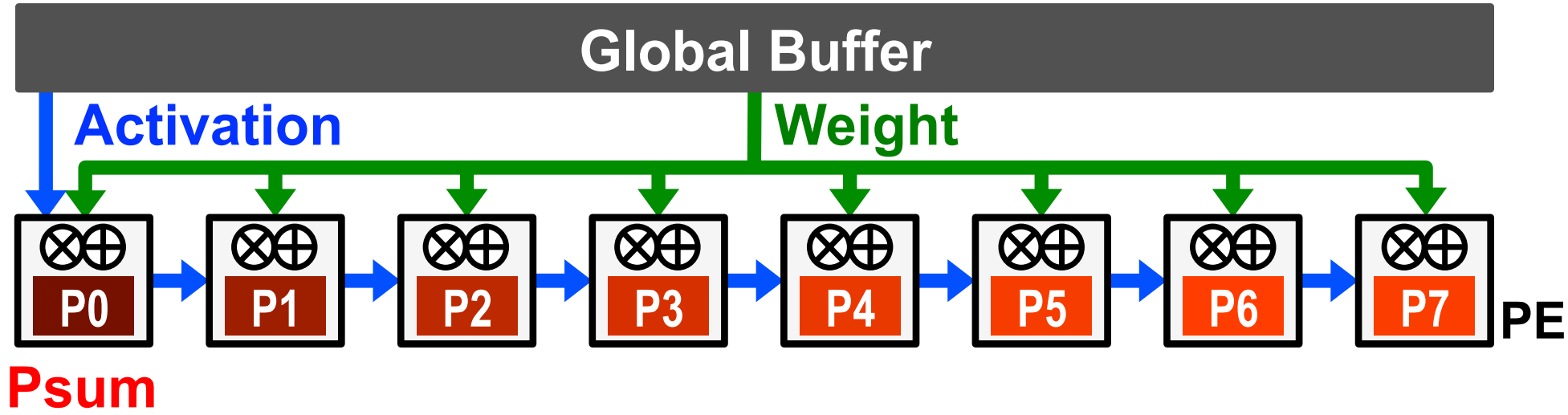
# WS Example: nn-X (NeuFlow)

## A 3×3 2D Convolution Engine




[Farabet et al., ICCV 2009]

# Output Stationary (OS)



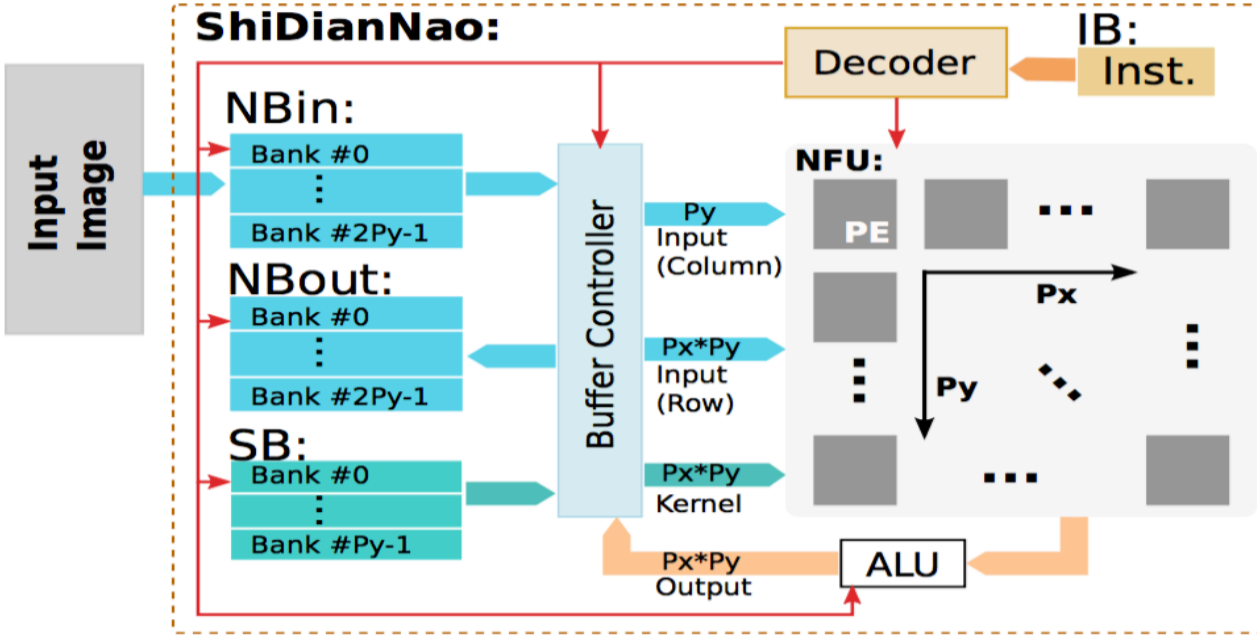

- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation

- **Examples:**

  [**Gupta**, *ICML* 2015]
  [**Peemen**, *ICCD* 2013]
  [**ShiDianNao**, *ISCA* 2015]

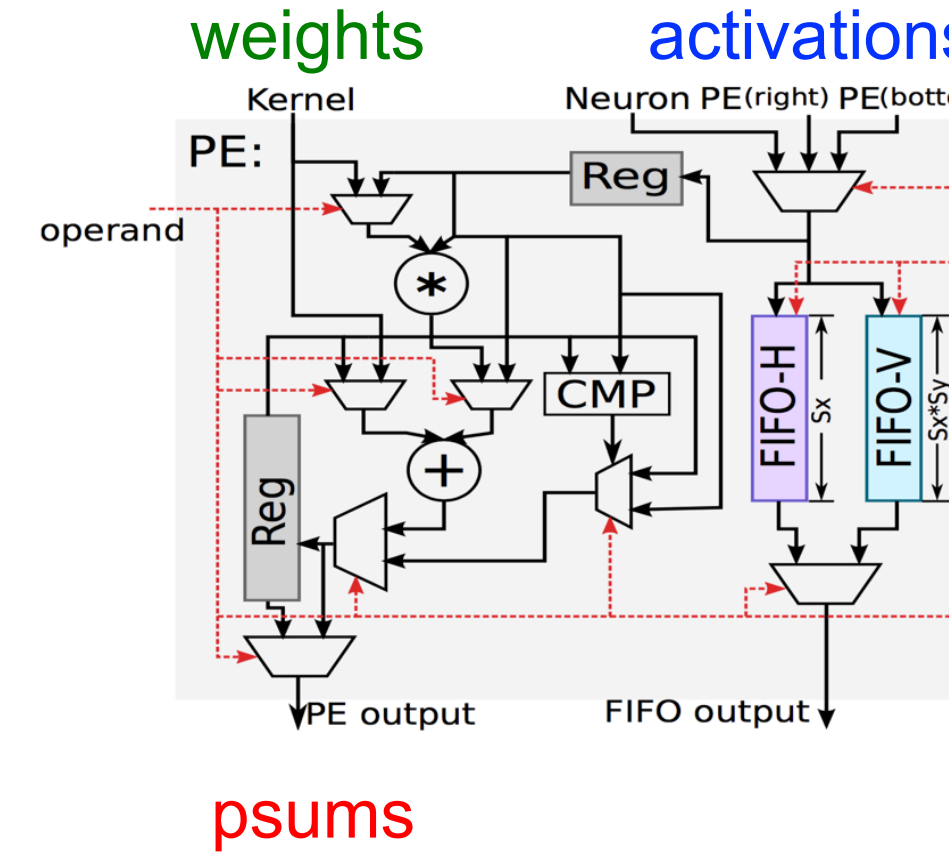
# OS Example: ShiDianNao

## Top-Level Architecture

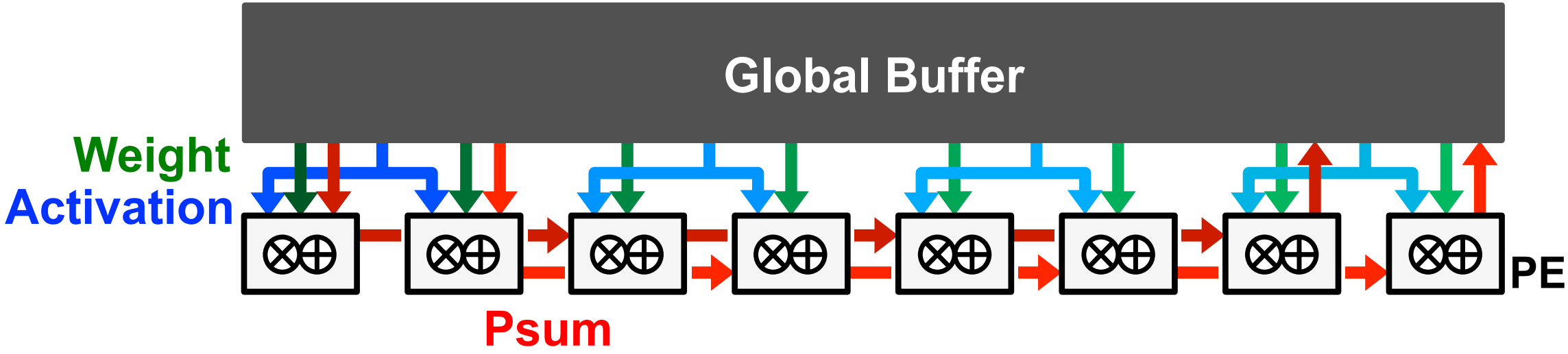



[Du et al., ISCA 2015]

## PE Architecture



# No Local Reuse (NLR)

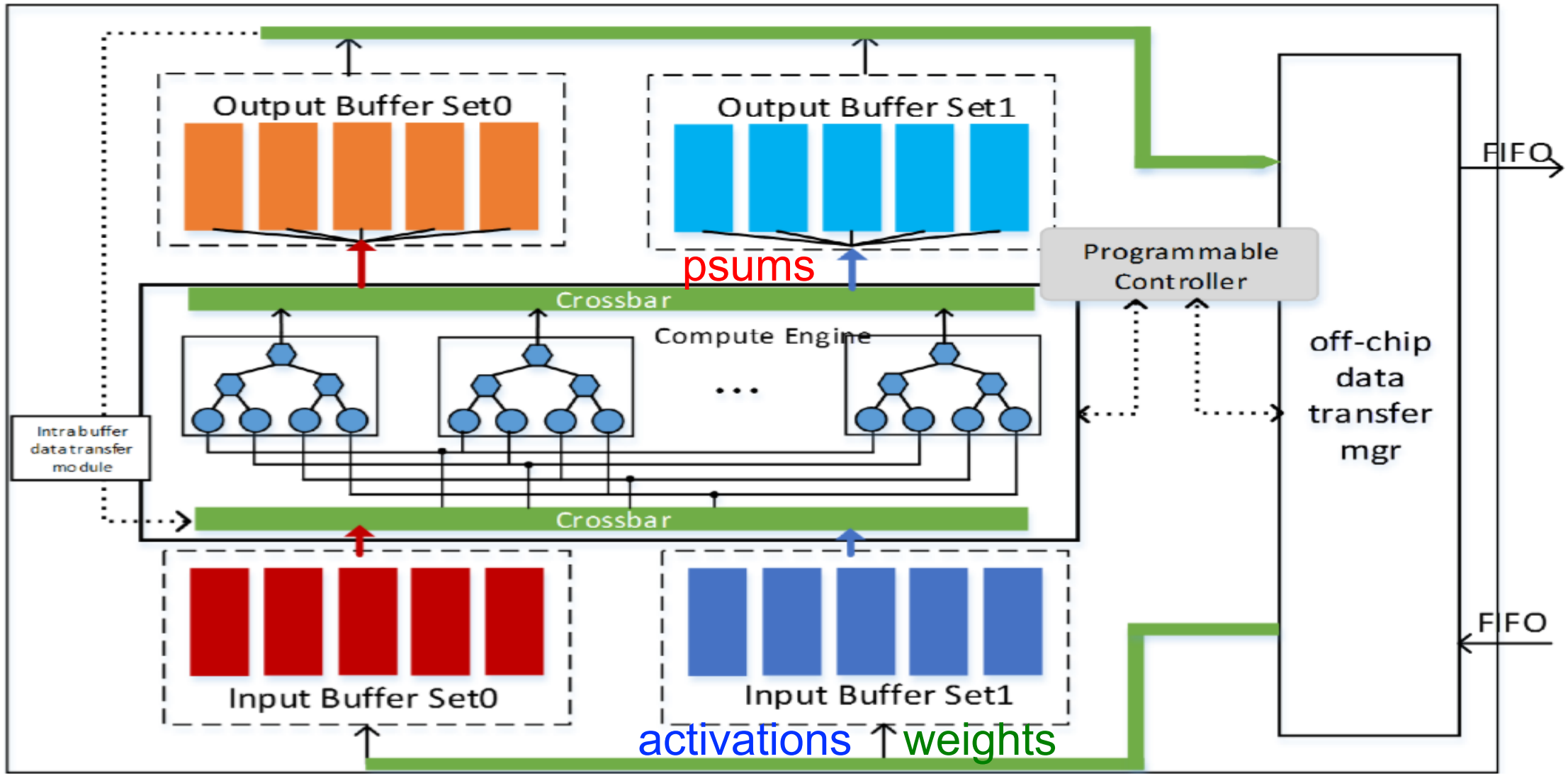



- Use a **large global buffer** as shared storage
  - Reduce **DRAM** access energy consumption

- **Examples:**

  [**DianNao**, *ASPLOS* 2014] [**DaDianNao**, *MICRO* 2014]
  [**Zhang**, *FPGA* 2015]

# NLR Example: UCLA




[Zhang et al., FPGA 2015]

# Taxonomy: More Examples

- **Weight Stationary (WS)**

  [**Chakradhar**, *ISCA* 2010] [**nn-X (NeuFlow)**, *CVPRW* 2014]
  [**Park**, *ISSCC* 2015] [**ISAAC**, *ISCA* 2016] [**PRIME**, *ISCA* 2016]

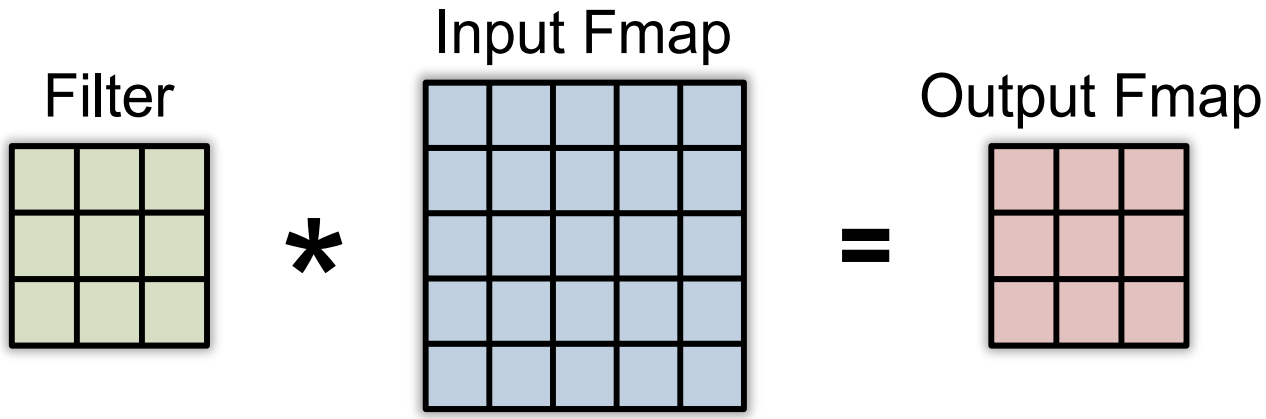
- **Output Stationary (OS)**

  [**Peemen**, *ICCD* 2013] [**ShiDianNao**, *ISCA* 2015]
  [**Gupta**, *ICML* 2015] [**Moons**, *VLSI* 2016] [**Thinker**, *VLSI* 2017]

- **No Local Reuse (NLR)**

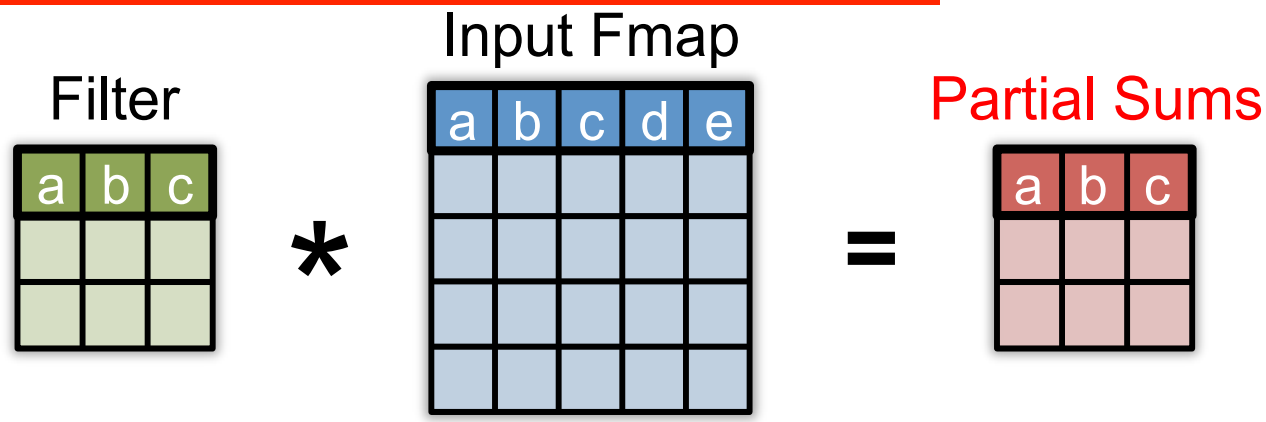
  [**DianNao**, *ASPLOS* 2014] [**DaDianNao**, *MICRO* 2014]
  [**Zhang**, *FPGA* 2015] [**TPU**, *ISCA* 2017]

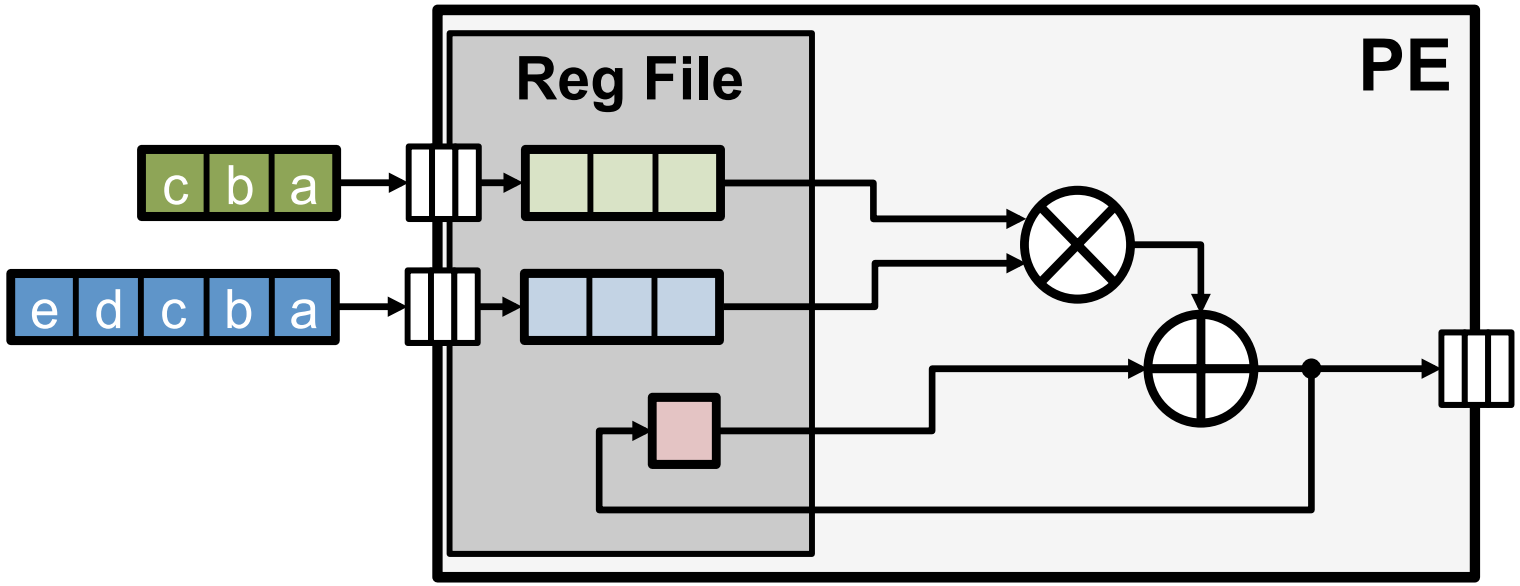
# Row Stationary: Energy-efficient Dataflow




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

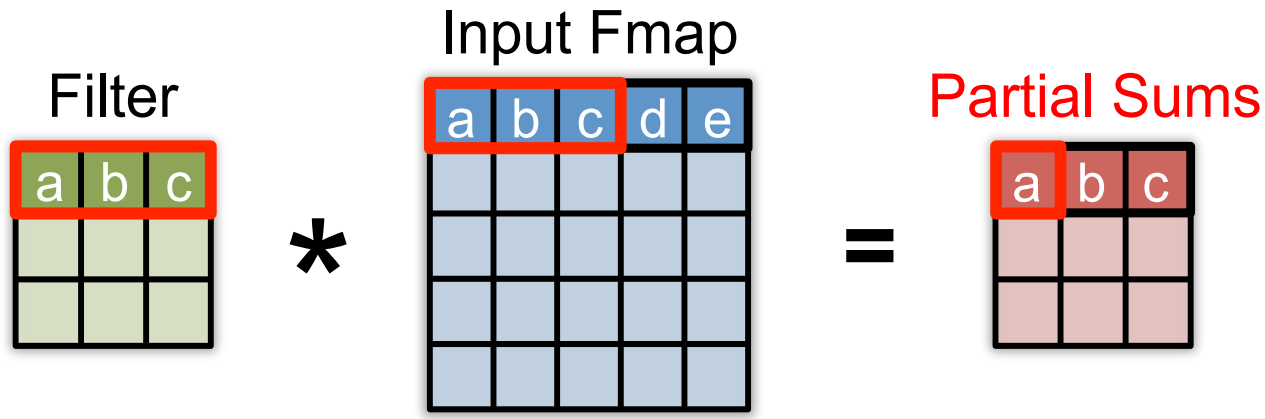
# 1D Row Convolution in PE

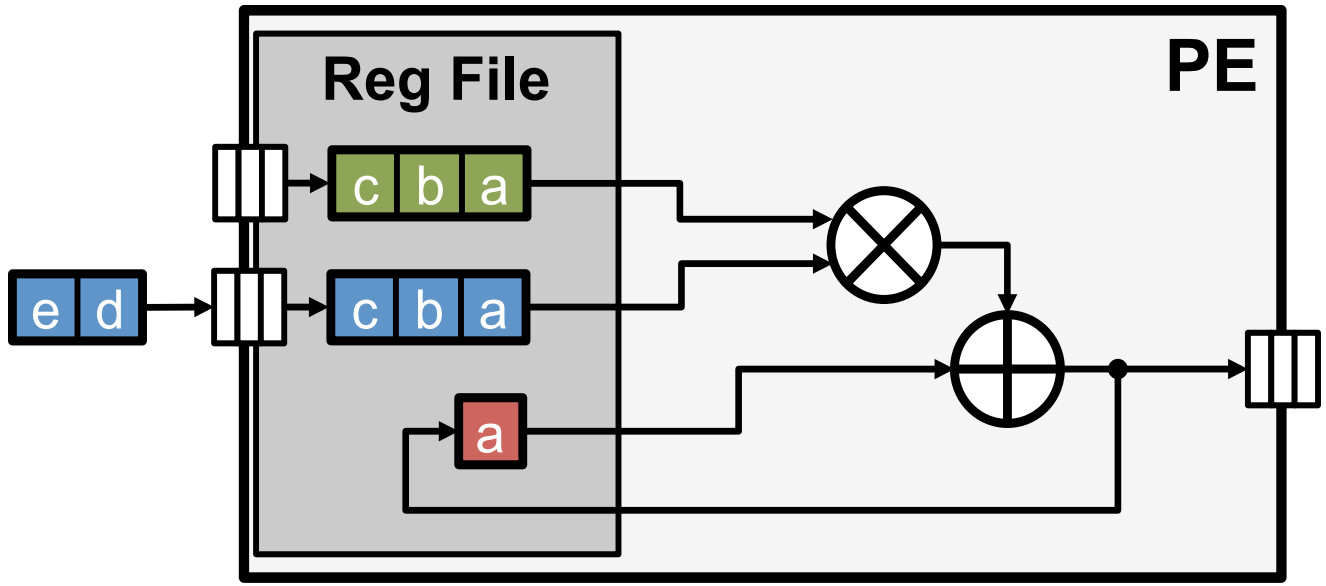



[**Chen**, *ISCA* 2016]

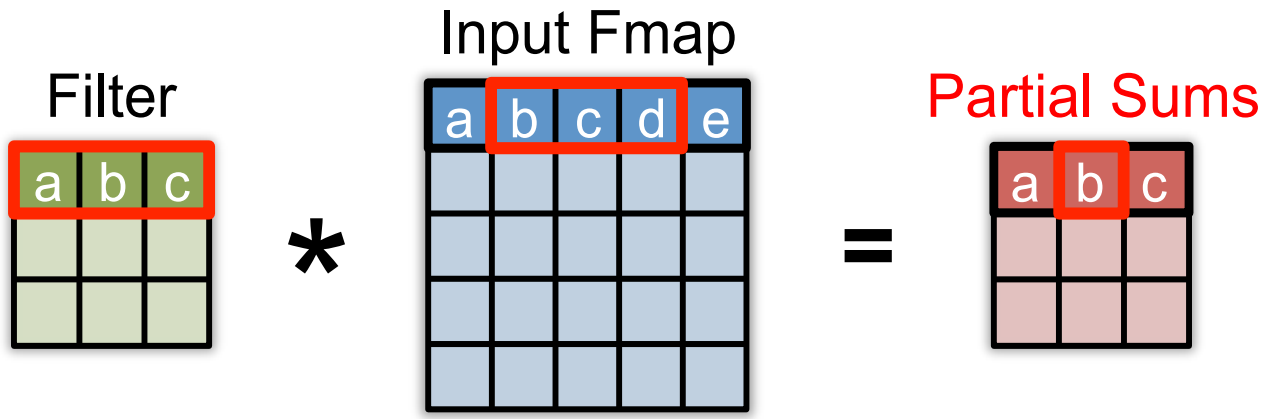
# 1D Row Convolution in PE

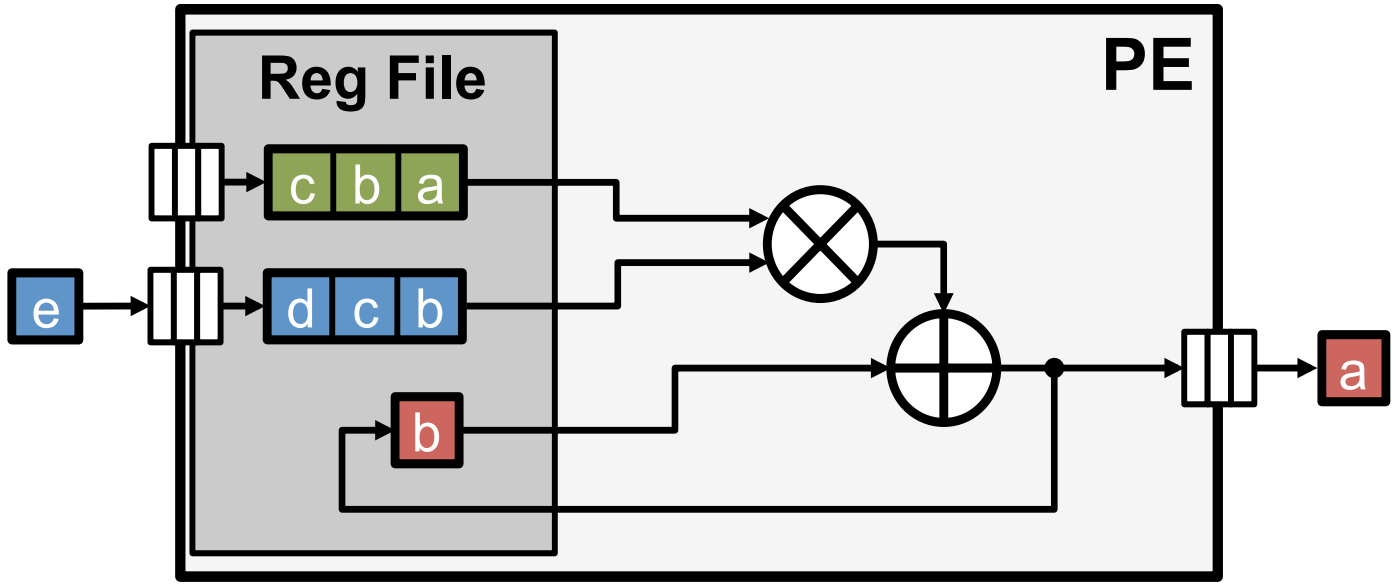



[**Chen**, *ISCA* 2016]

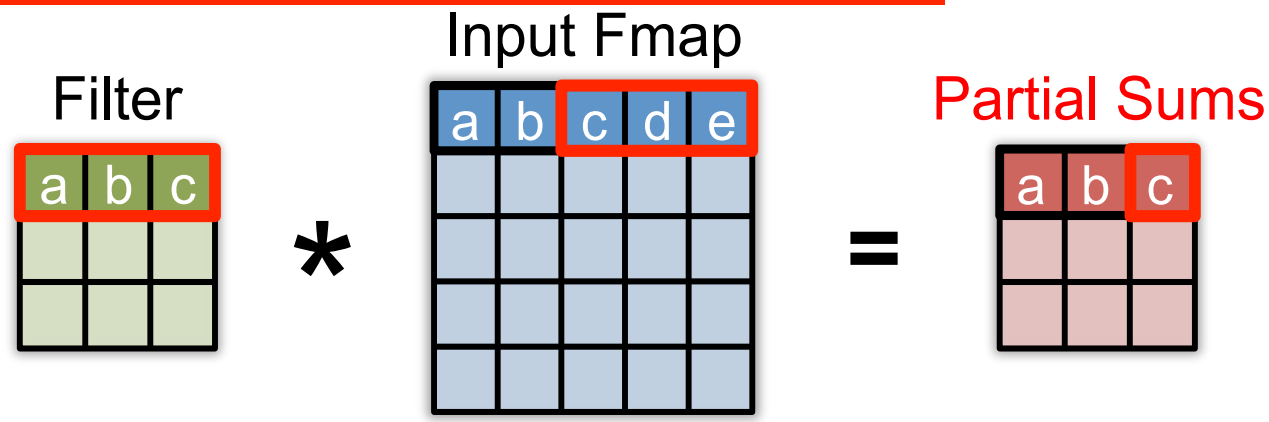
# 1D Row Convolution in PE

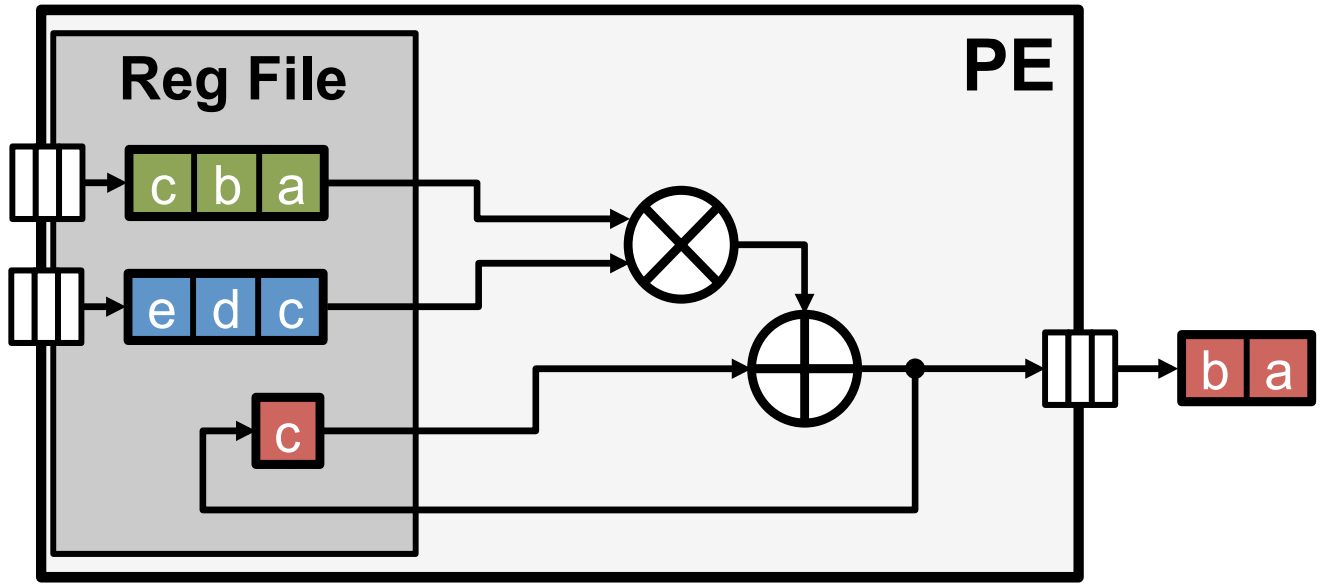



[**Chen**, *ISCA* 2016]

# 1D Row Convolution in PE

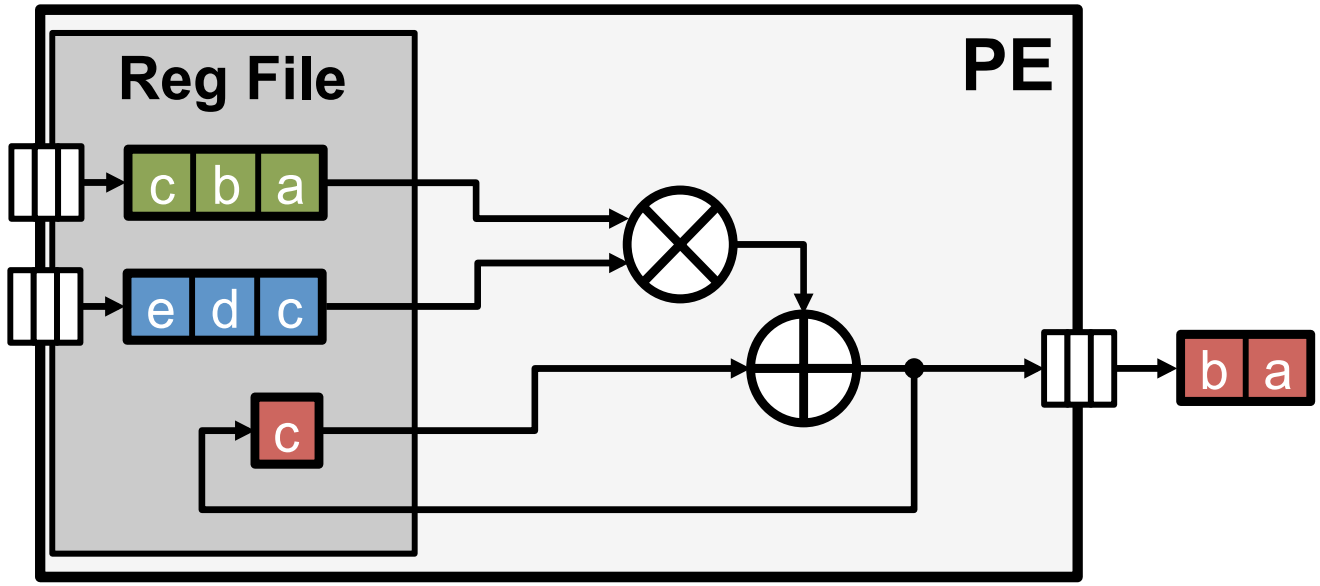



[**Chen**, *ISCA* 2016]

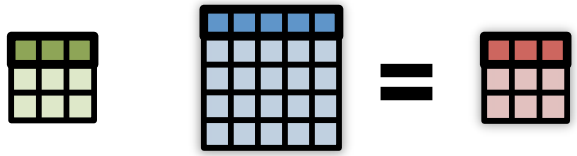
# 1D Row Convolution in PE

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

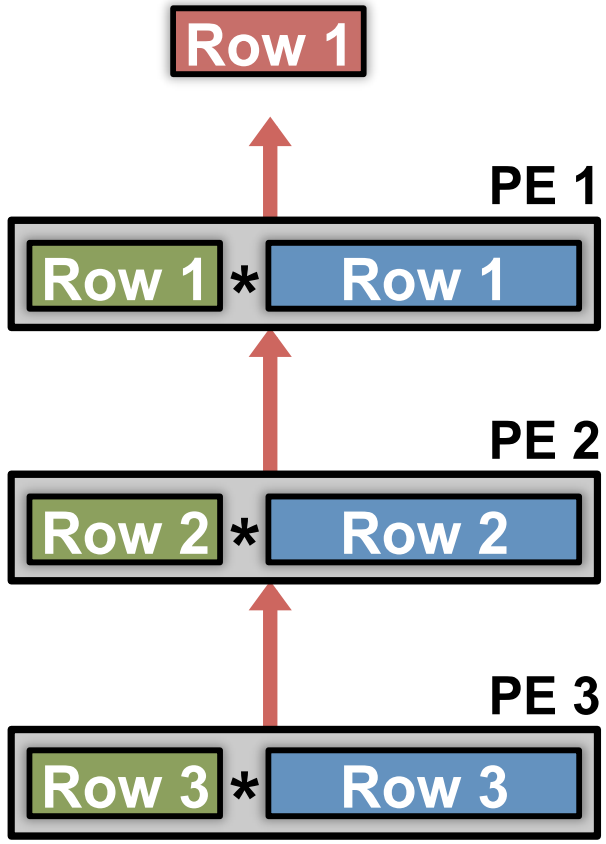


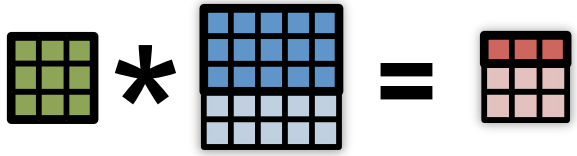

[**Chen**, *ISCA* 2016]

# 2D Convolution in PE Array

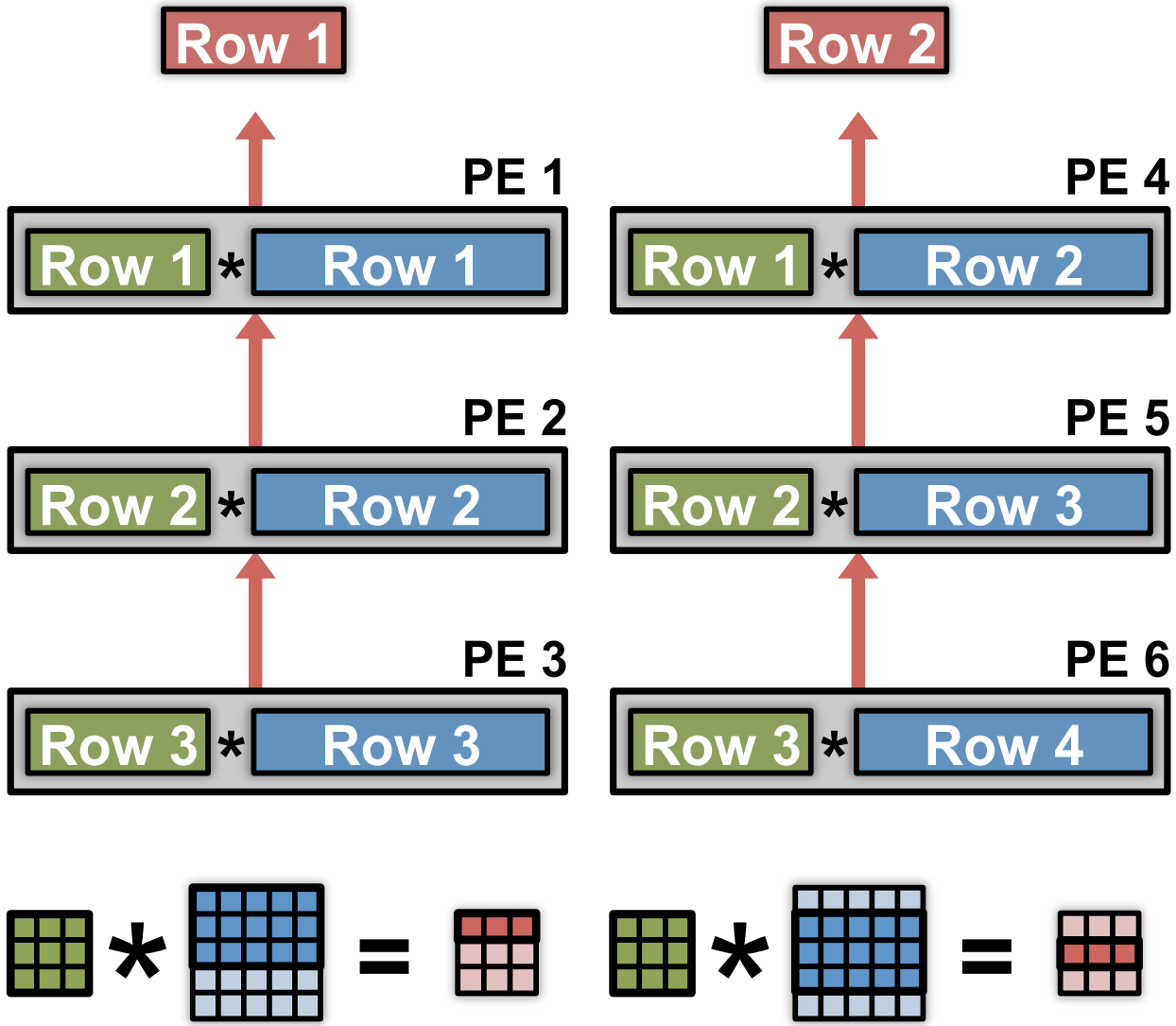



# 2D Convolution in PE Array

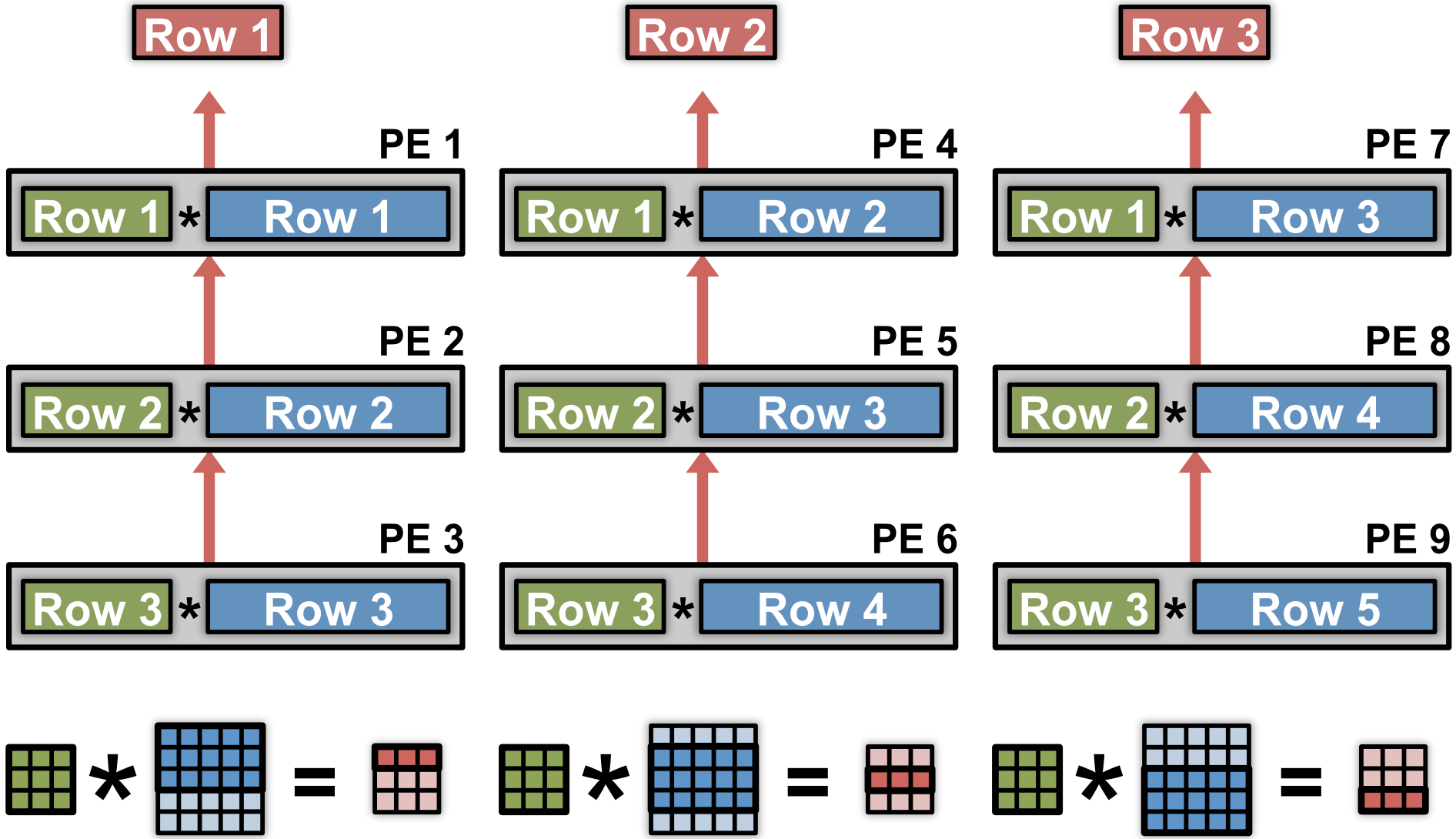


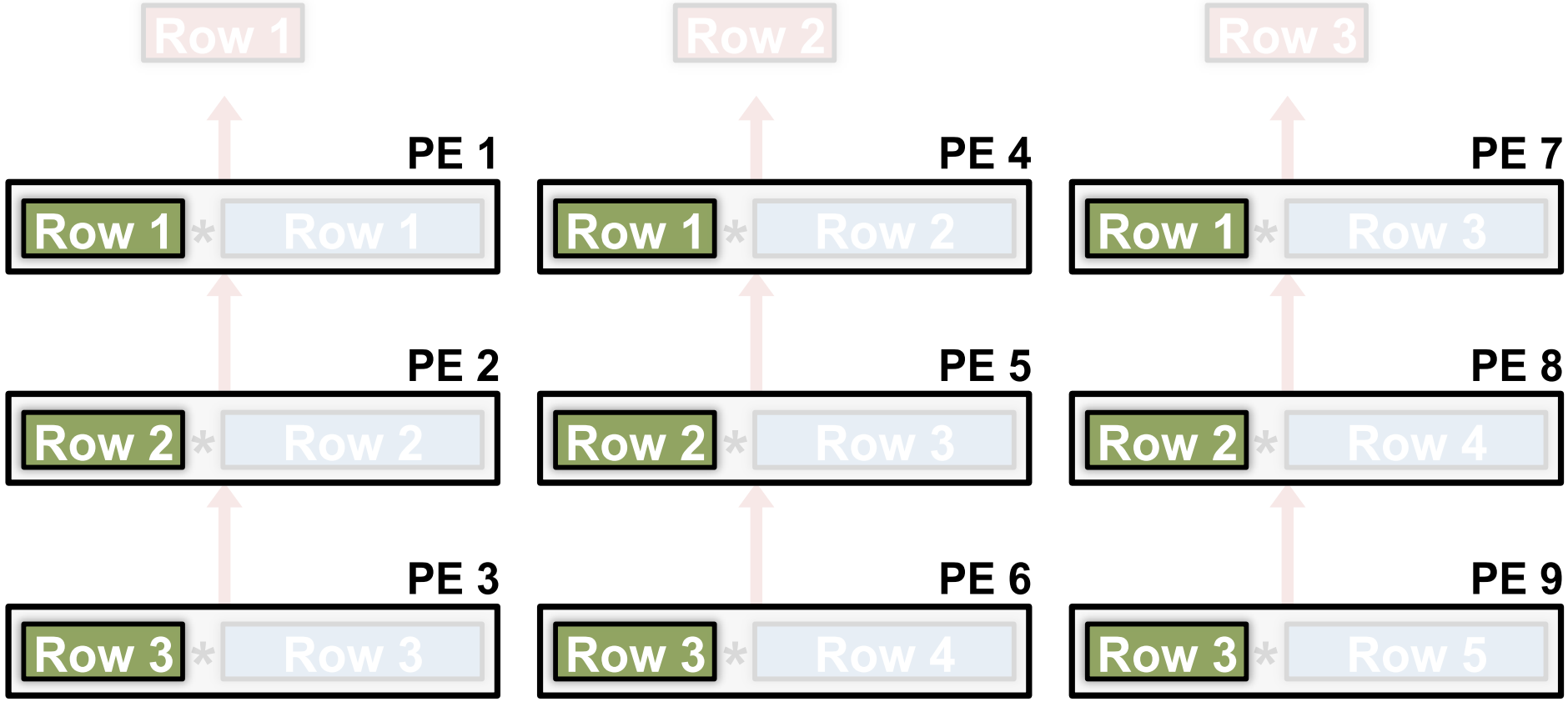
# 2D Convolution in PE Array



# 2D Convolution in PE Array



# Convolutional Reuse Maximized




**Filter rows** are reused across PEs **horizontally**

# Convolutional Reuse Maximized

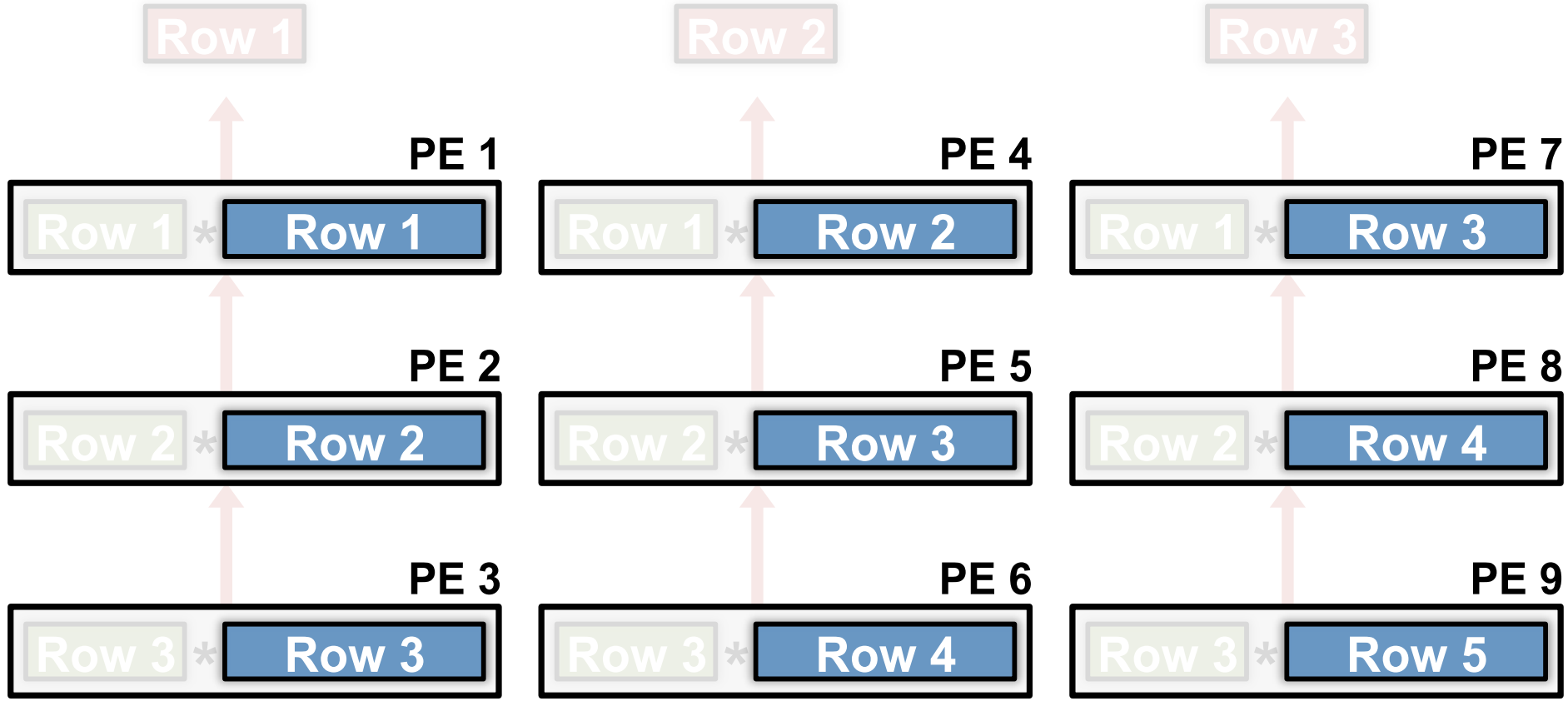
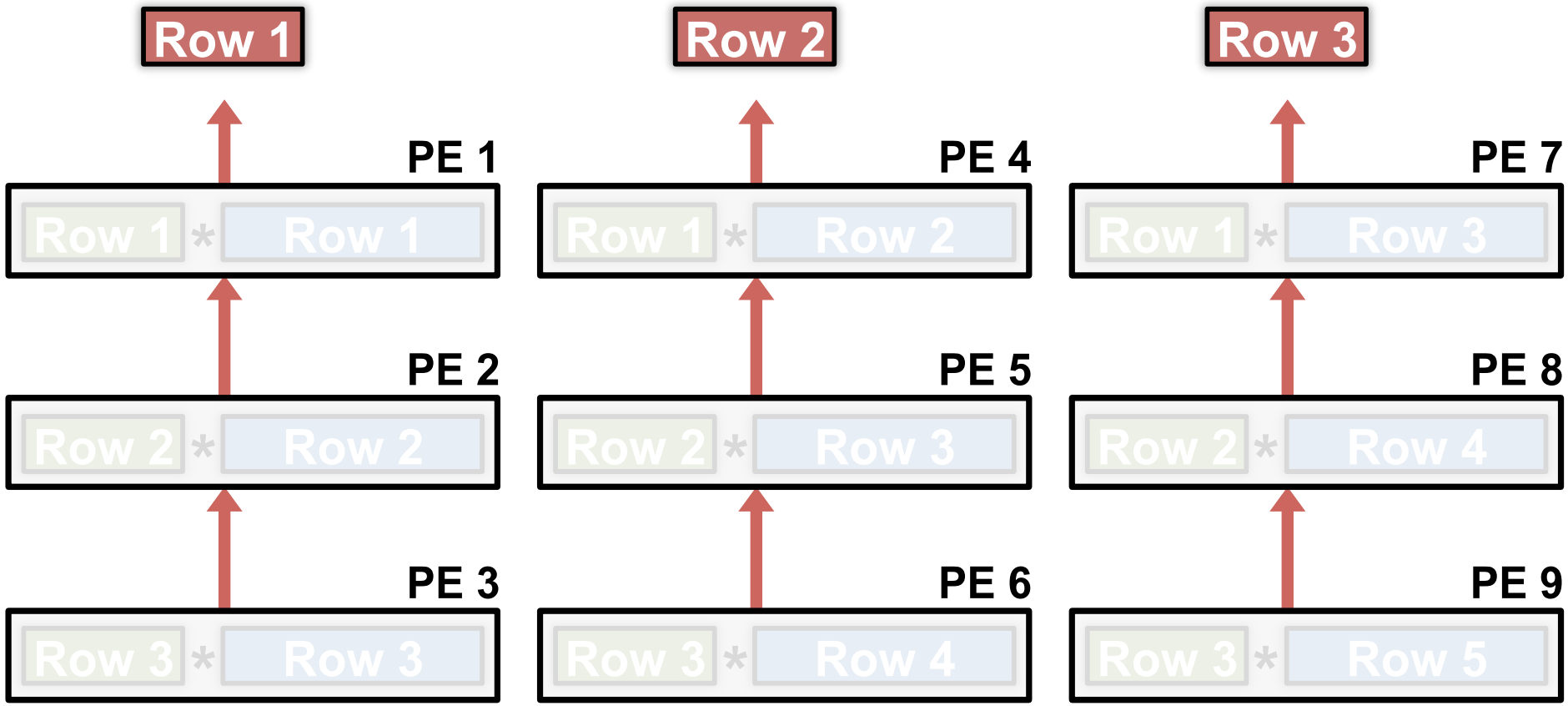


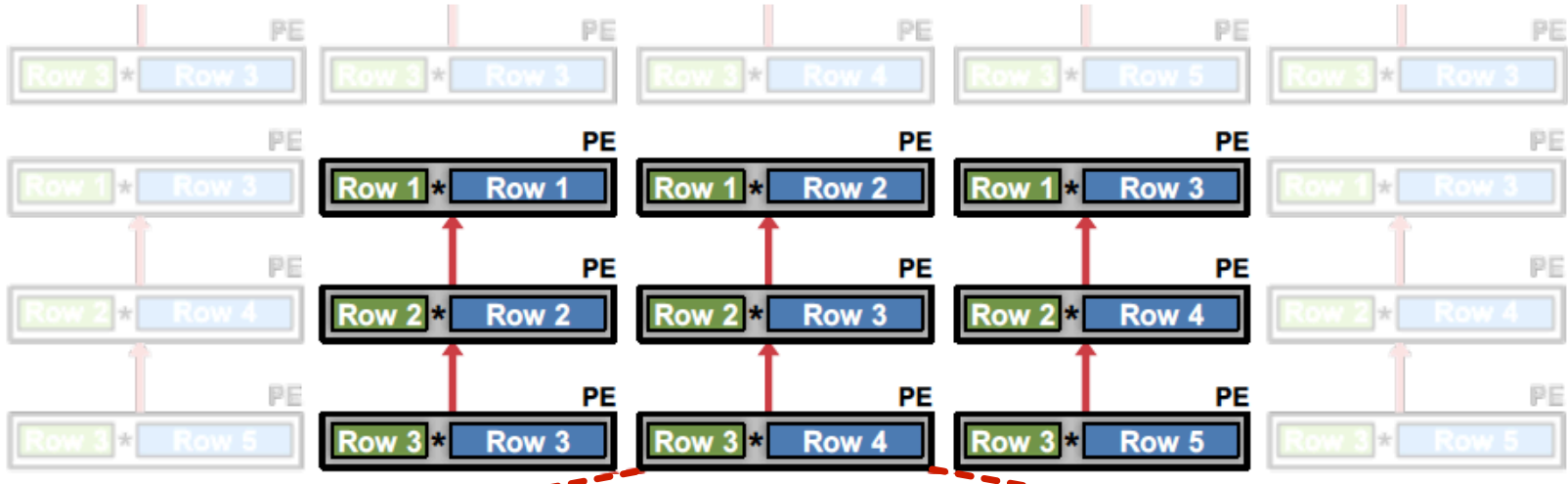

**Image rows** are reused across PEs **diagonally**

# Maximize 2D Accumulation in PE Array



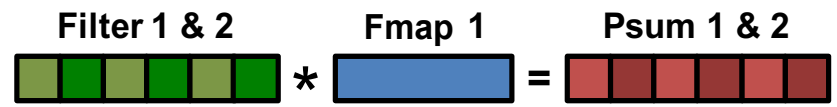

**Partial sums** accumulate across PEs **vertically**

# CNN Convolution – The Full Picture




Multiple **fmaps**:


Multiple **filters**:




Multiple **channels**:


Map rows from **multiple images**, **filters** and **channels** to same PE to exploit other forms of reuse and local accumulation

# Evaluate Reuse in Different Dataflows

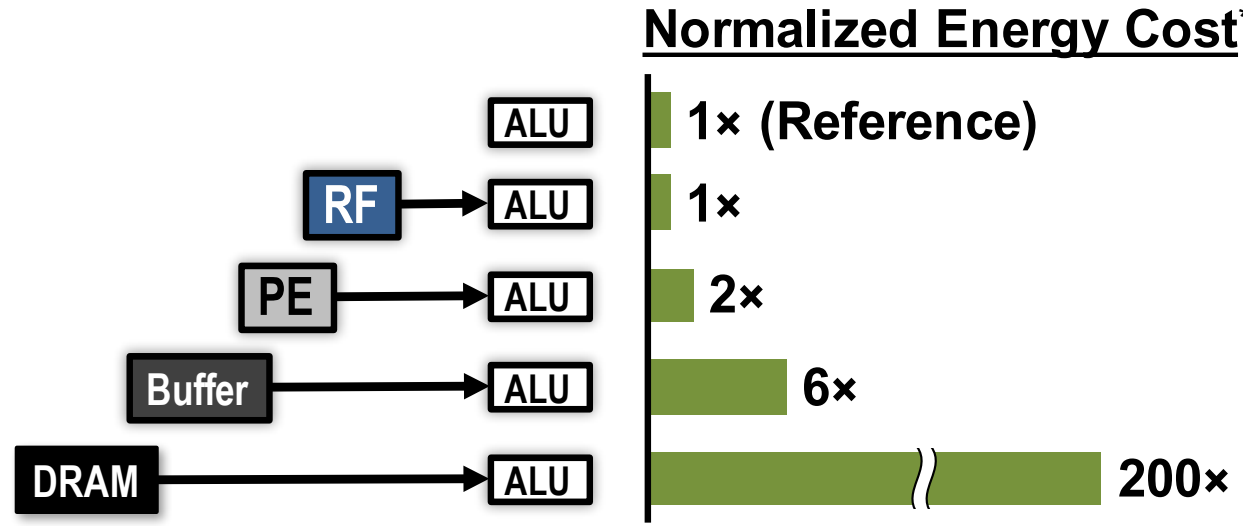
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - Don't use any local PE storage. Maximize global buffer size.
- **Row Stationary**

# Evaluate Reuse in Different Dataflows

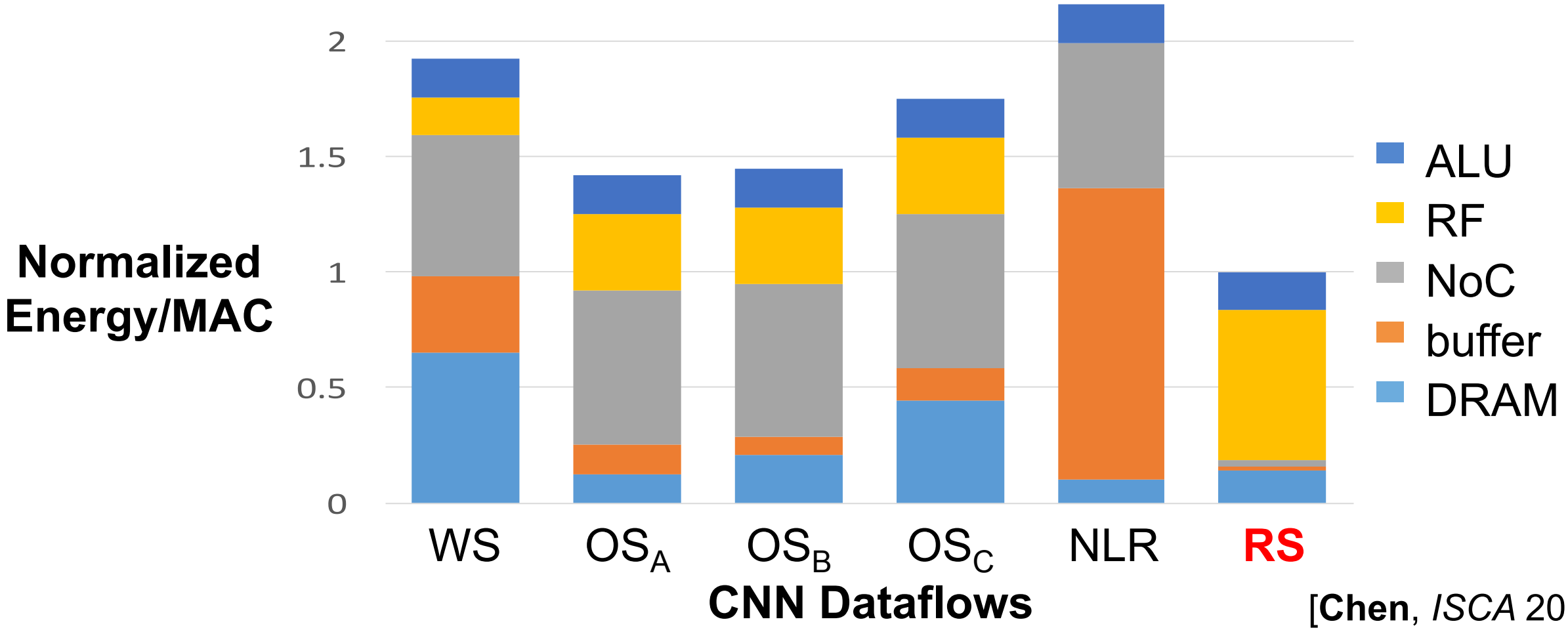
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - Don't use any local PE storage. Maximize global buffer size.
- **Row Stationary**

**Evaluation Setup**

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



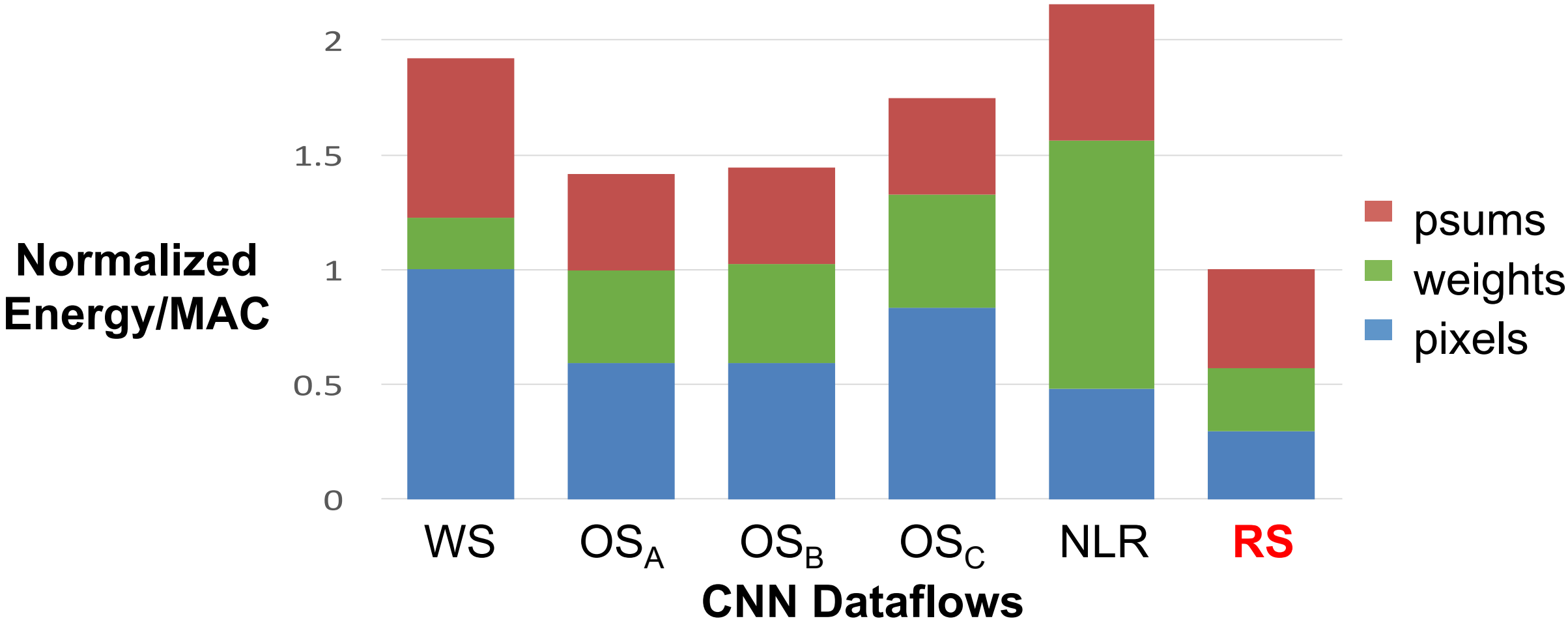
# Dataflow Comparison: CONV Layers




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

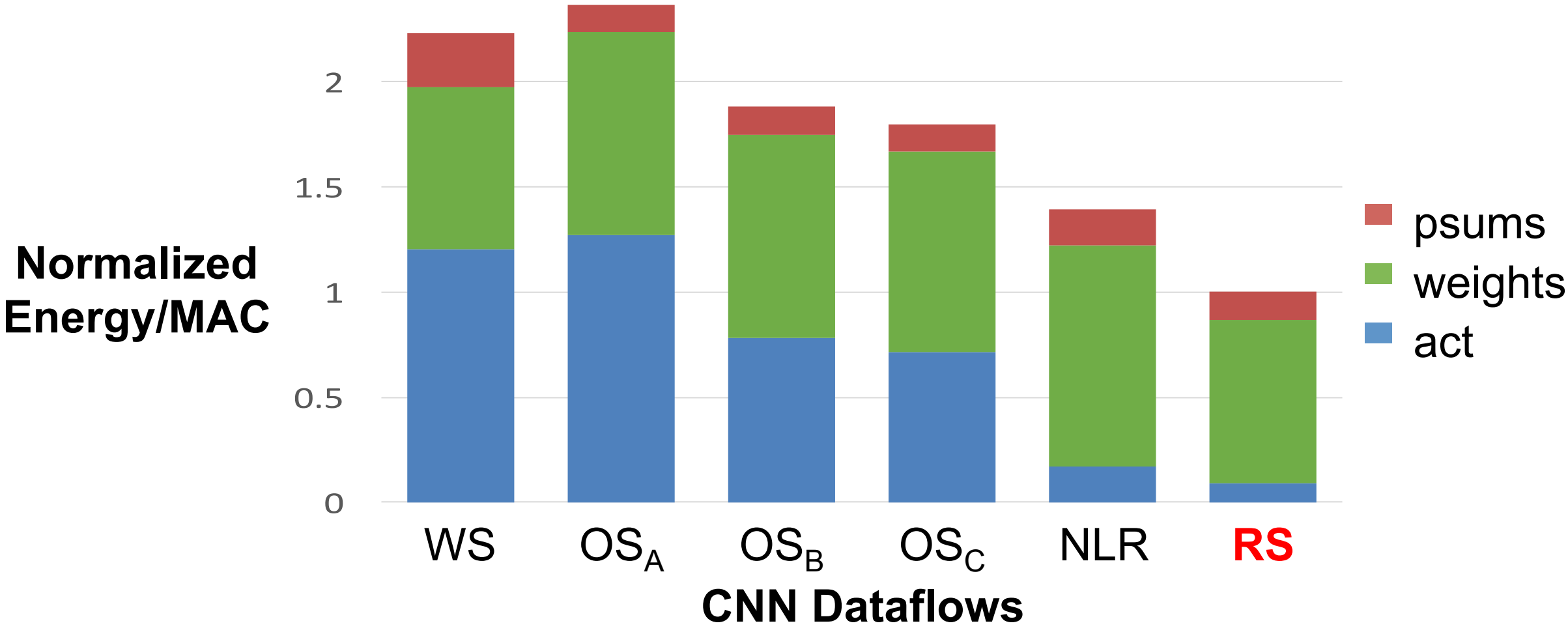
RS uses **1.4× – 2.5× lower** energy than other dataflows

# Dataflow Comparison: CONV Layers



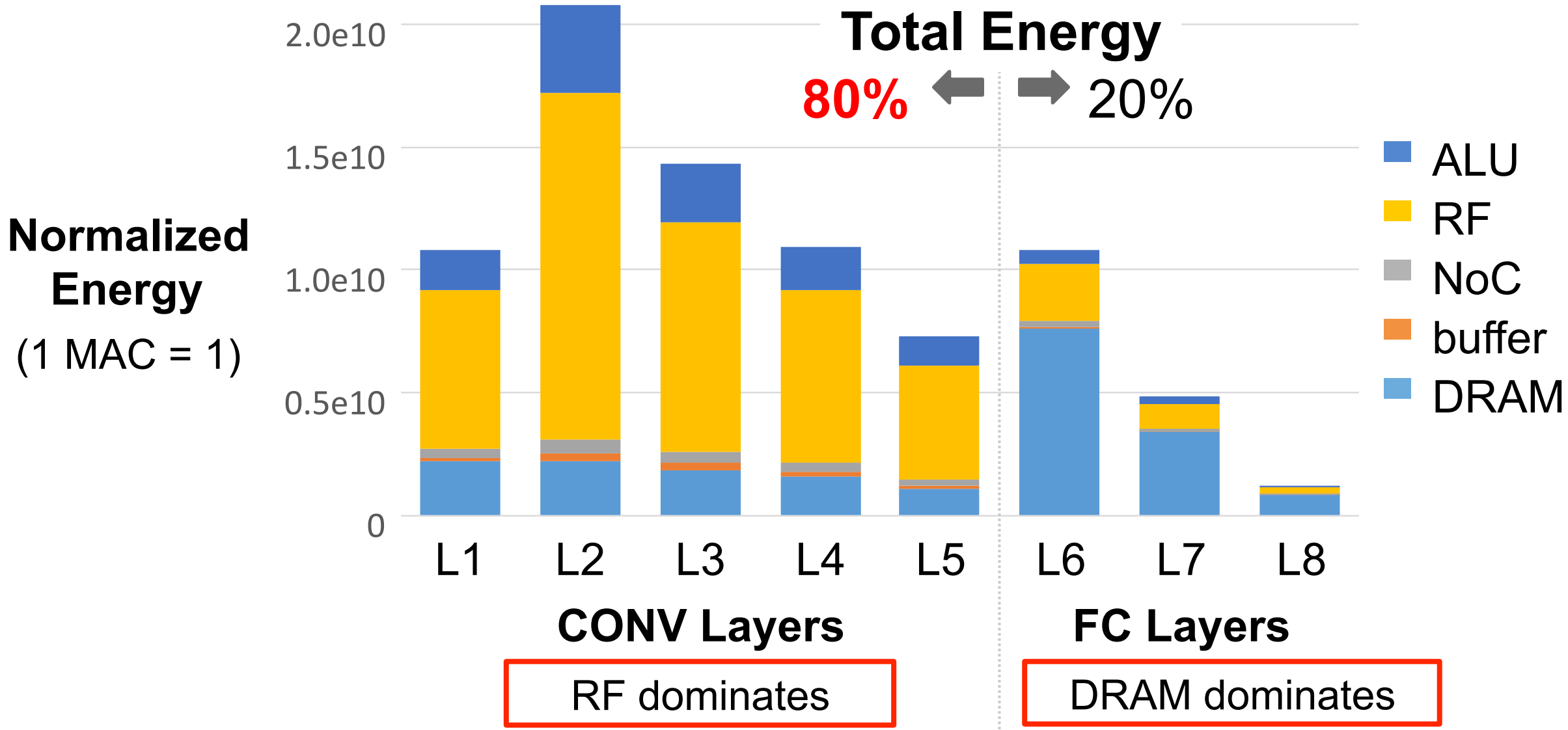

RS optimizes for the best **overall** energy efficiency

# Dataflow Comparison: FC Layers




RS uses at least **1.3× lower** energy than other dataflows

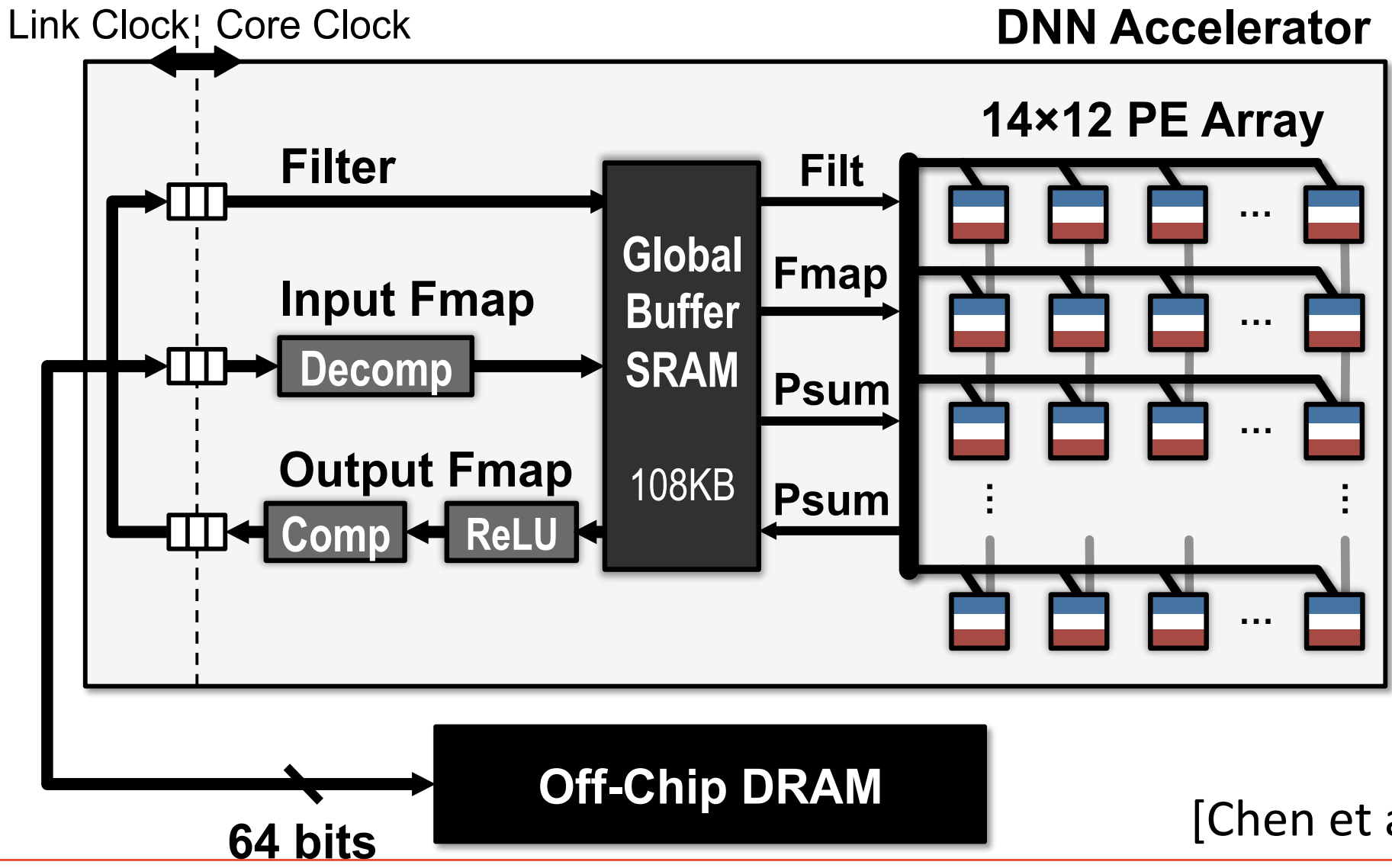
# Row Stationary: Layer Breakdown



# Hardware for Row Stationary Dataflow

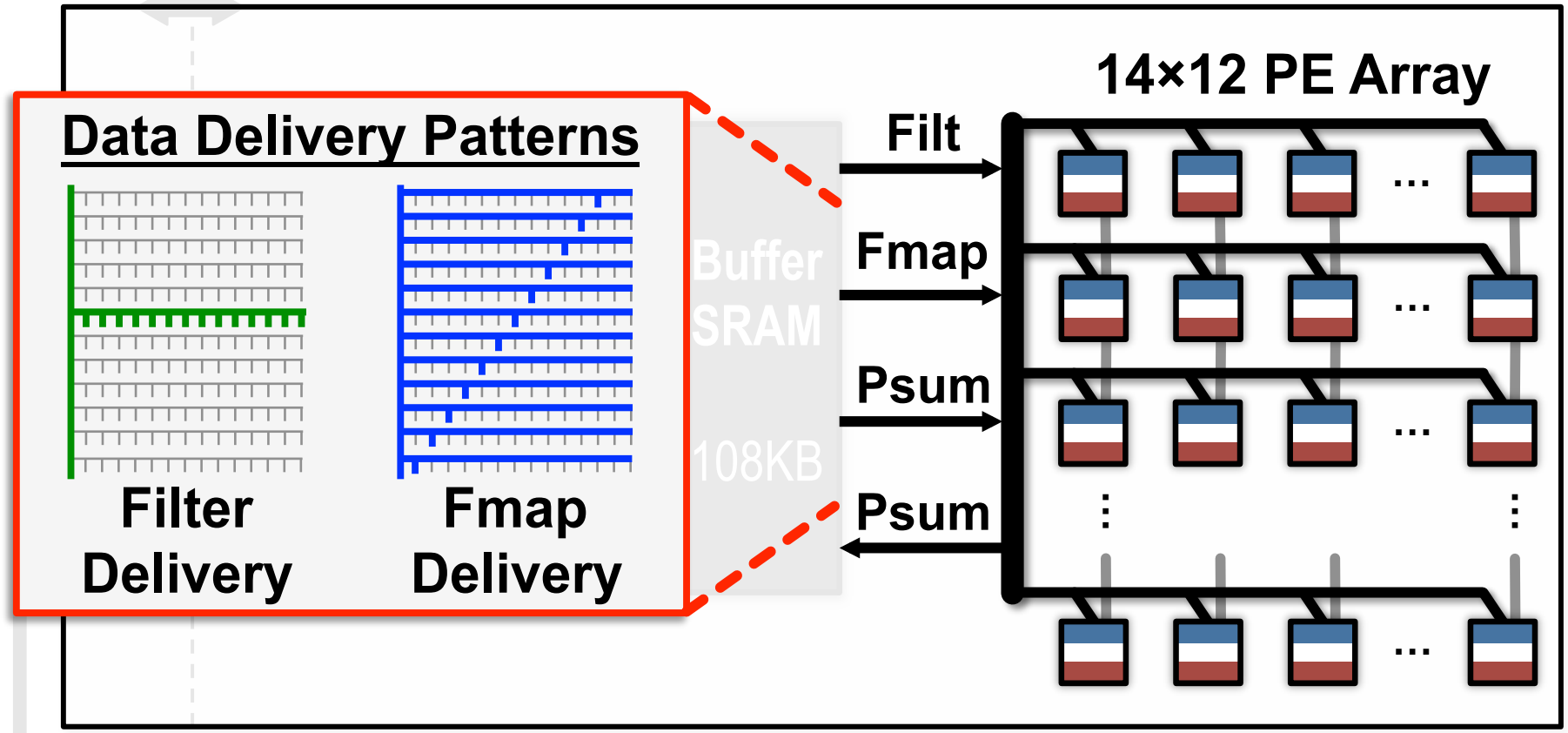
[Chen et al., ISSCC 2016]

# Eyeriss Deep CNN Accelerator




[Chen et al., ISSCC 2016]

# Data Delivery with On-Chip Network

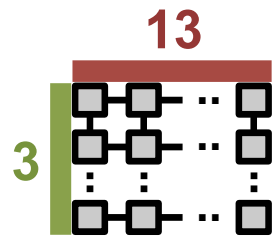


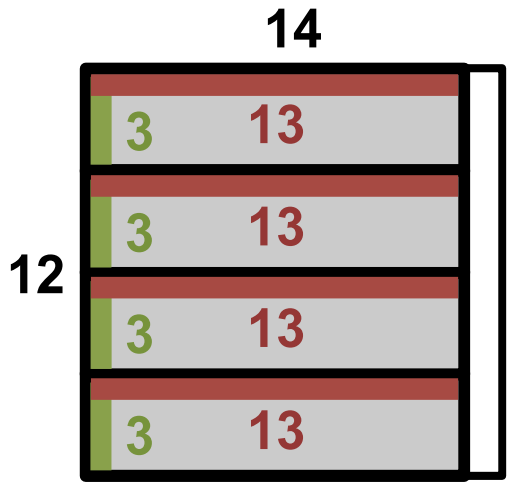

How to accommodate different shapes with fixed PE array?

# Logical to Physical Mappings

## Replication

**AlexNet**
Layer 3-5

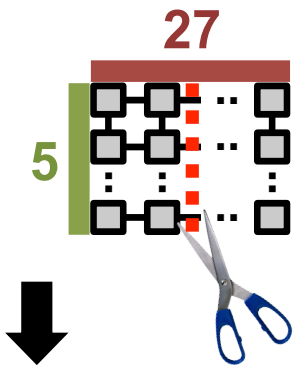


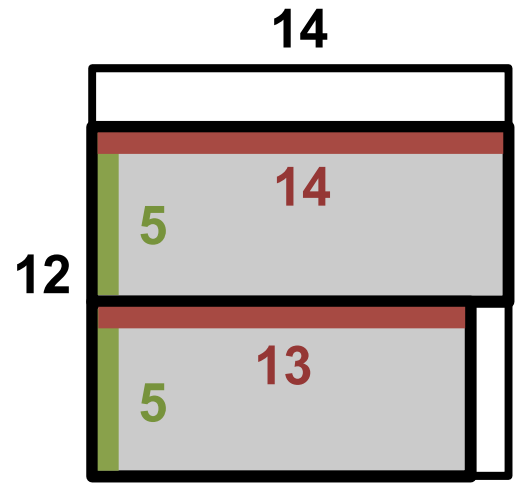




**Physical PE Array**

## Folding

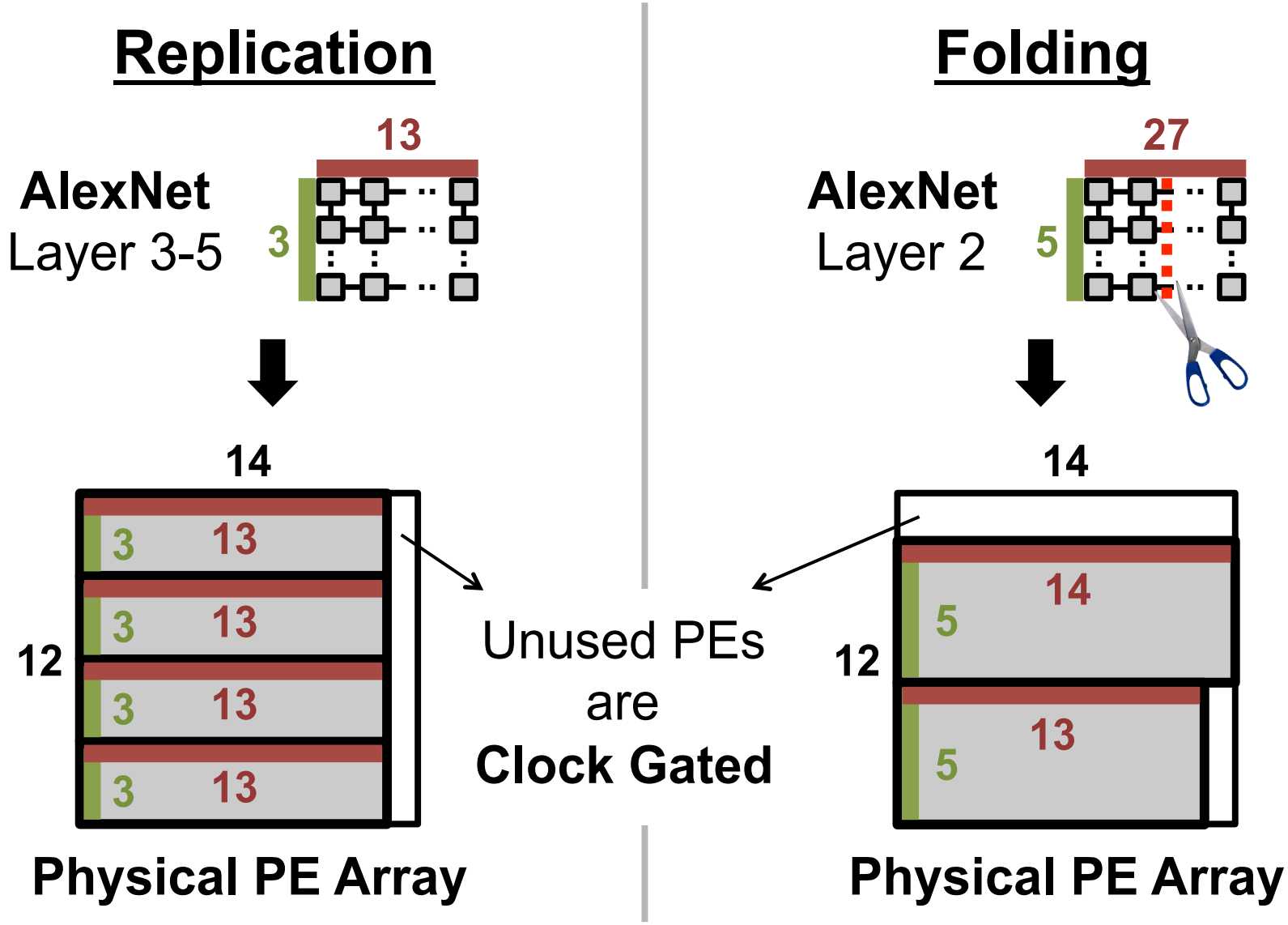
**AlexNet**
Layer 2



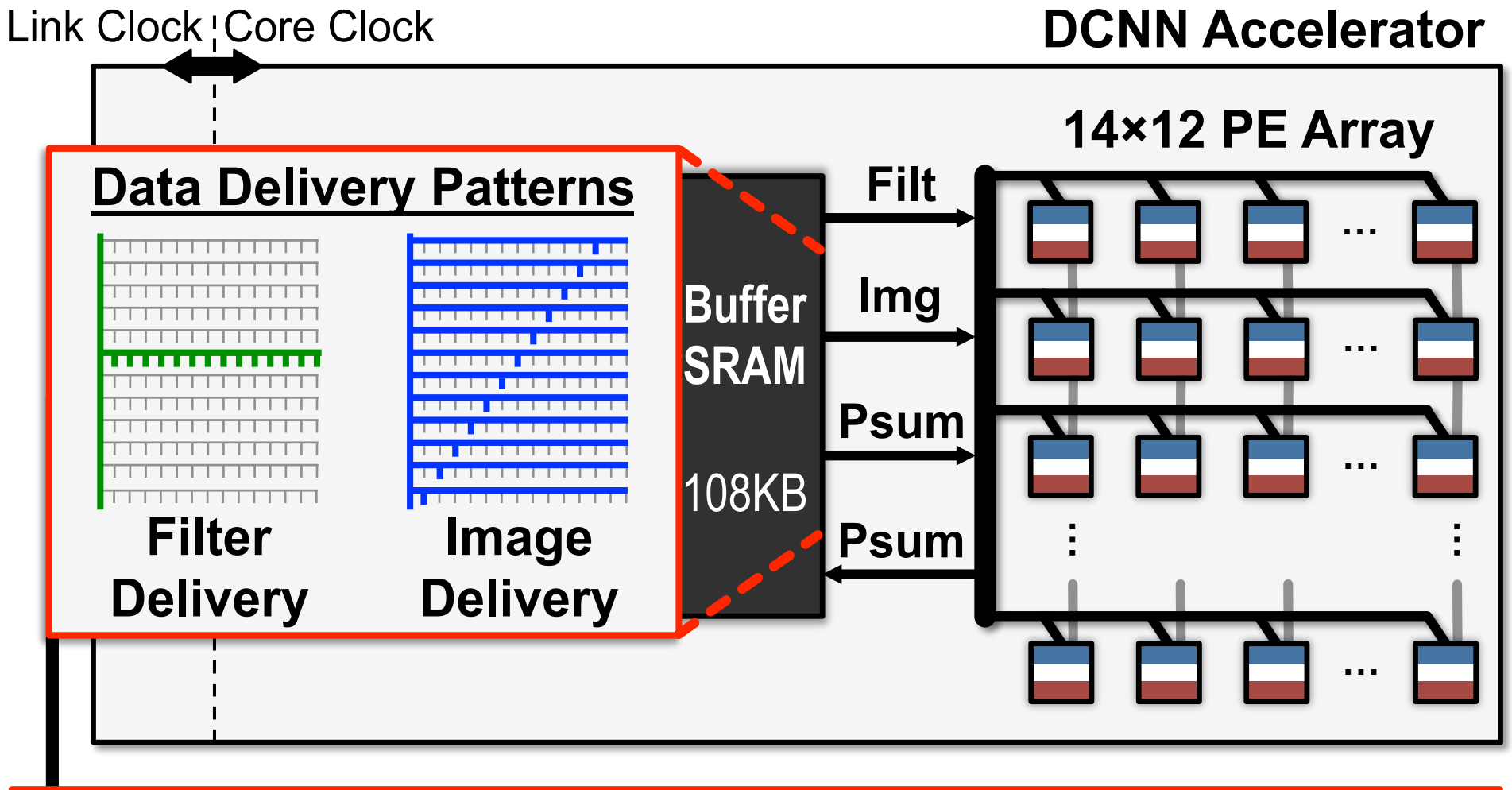




**Physical PE Array**

# Logical to Physical Mappings



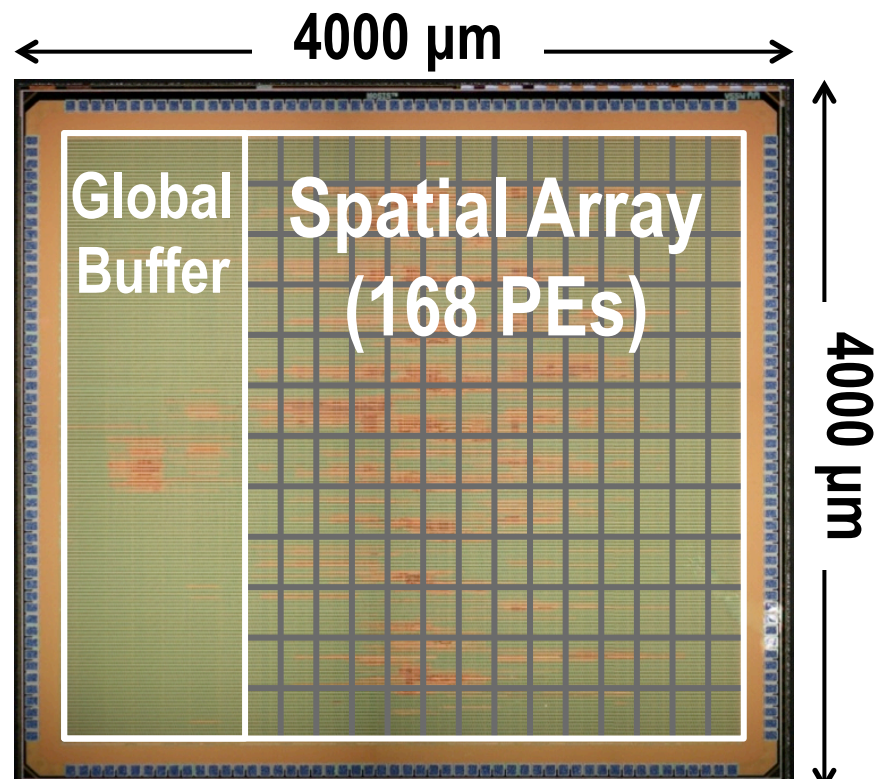
# Data Delivery with On-Chip Network




Compared to Broadcast, **Multicast** saves >**80%** of NoC energy

# Eyeriss 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 CNN Shapes** | Filter Width: 1 – 32<br>Filter Height: 1 – 12<br>Num. Filters: 1 – 1024<br>Num. Channels: 1 – 1024<br>Horz. Stride: 1–12<br>Vert. Stride: 1, 2, 4 |




[Chen et al., ISSCC 2016]

AlexNet: For 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)

# Summary of DNN Dataflows

- **Weight Stationary**
  - Minimize movement of filter weights
  - Popular with processing-in-memory architectures
- **Output Stationary**
  - Minimize movement of partial sums
  - Different variants optimized for CONV or FC layers
- **No Local Reuse**
  - No PE local storage → maximize global buffer size
- **Row Stationary**
  - Adapt to the NN shape and hardware constraints
  - Optimized for overall system energy efficiency

# Understanding the Energy Gap with Hand-Crafted Features

[Suleiman et al., ISCAS 2016]

# Hand-Crafted vs. Learned Features

Image

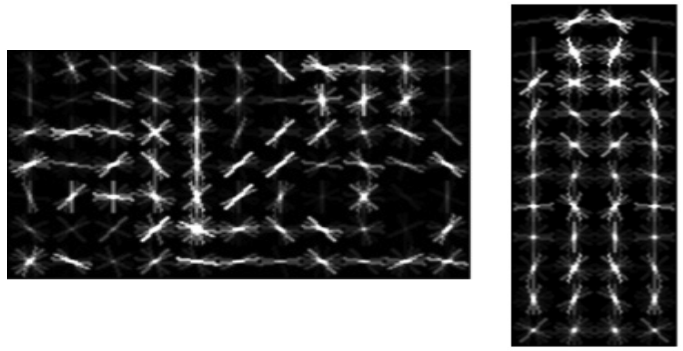
pixels → **Feature Extraction** → Features (**x**) → **Classification ($w^T x$)** → Scores

Trained weights (**w**)

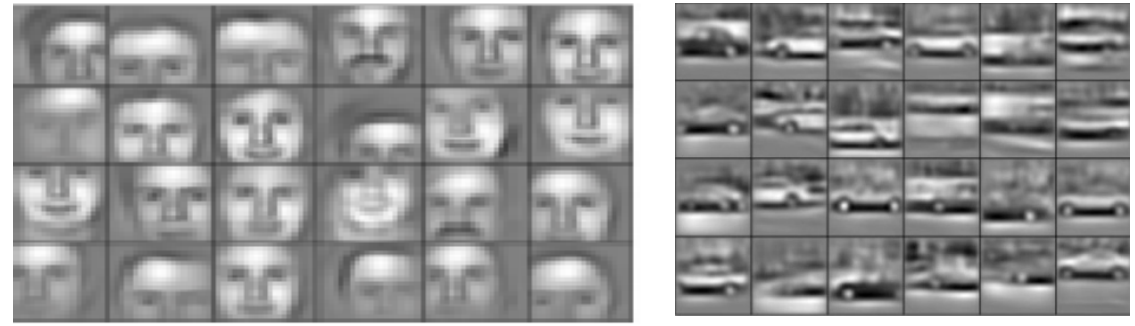
Scores per class

**(select class based on max or threshold)**

**Handcrafted Features (e.g. HOG)**

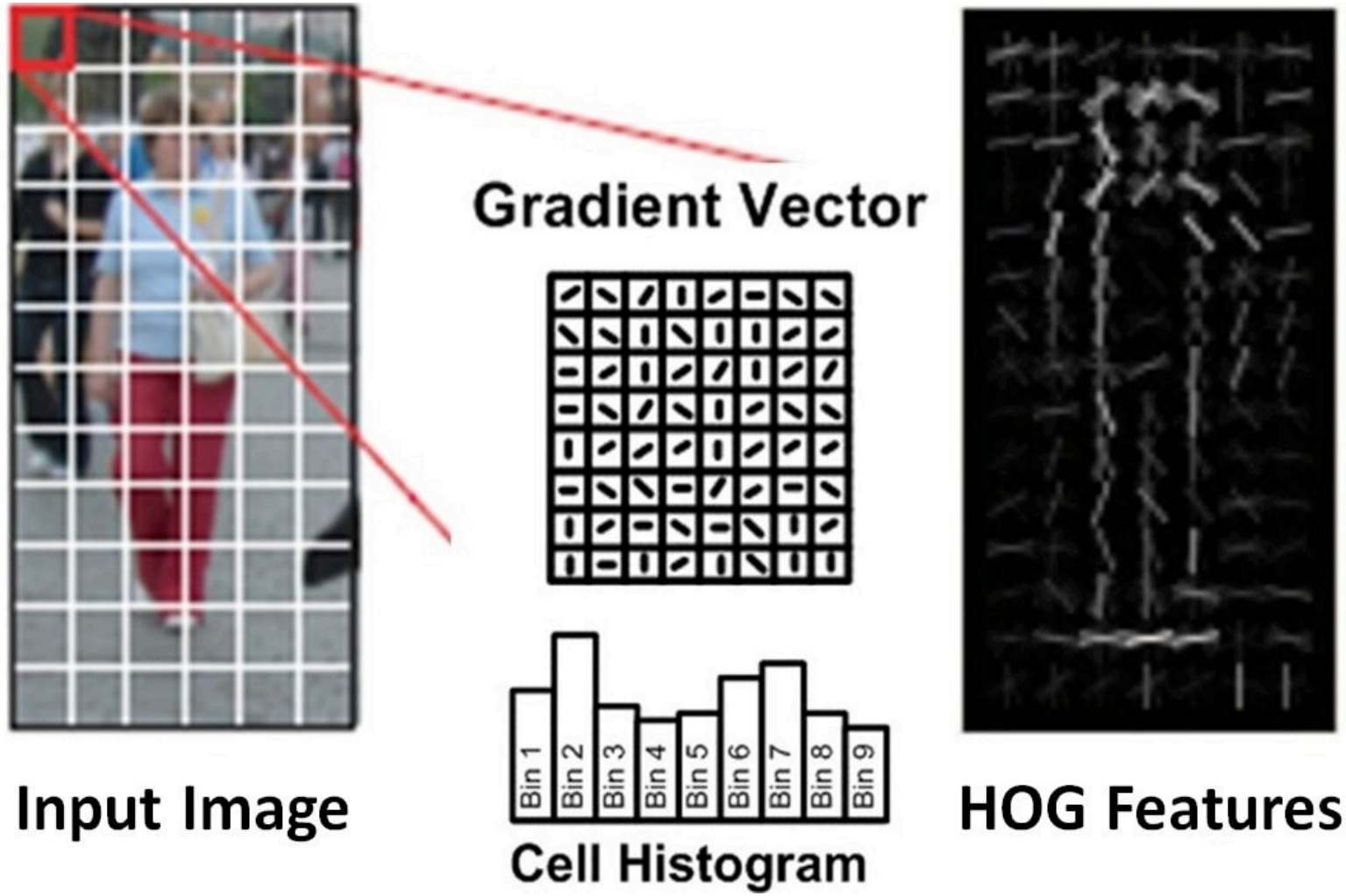


**Learned Features (e.g. CNN)**



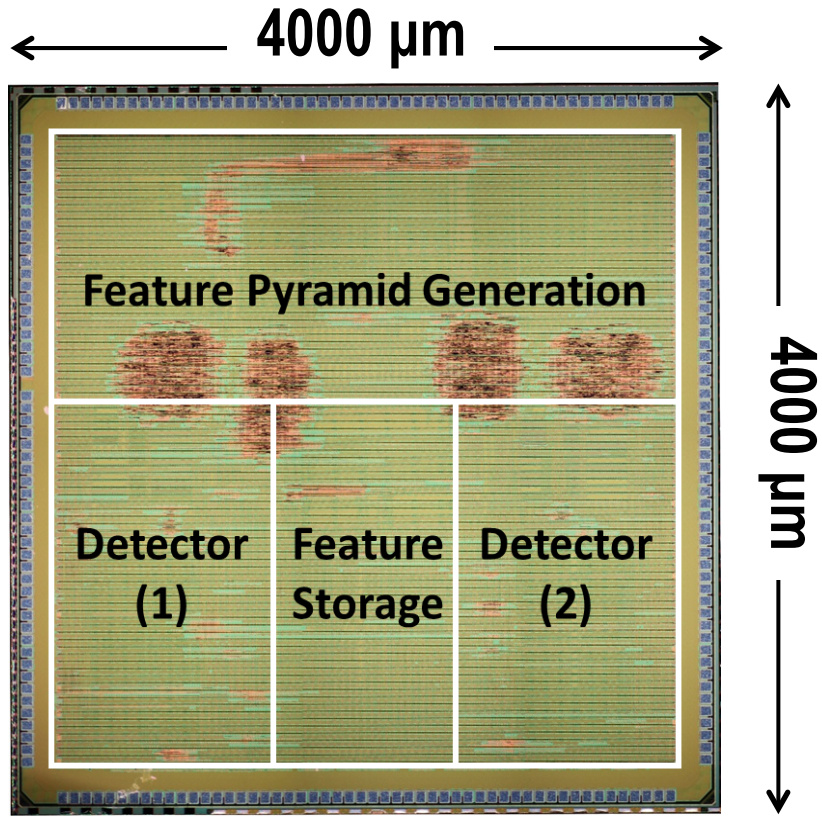
# Hand-Crafted Features (HOG)

## HOG = Histogram of Oriented Gradients

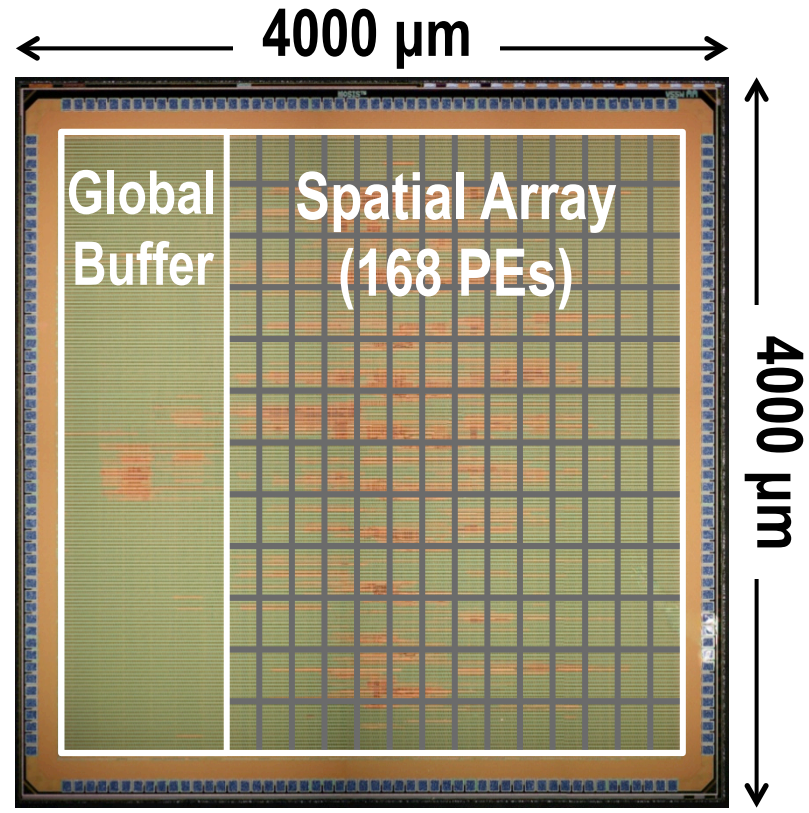


# Compare HOG and CNN

**Compare using measured results from test chips (65 nm)**

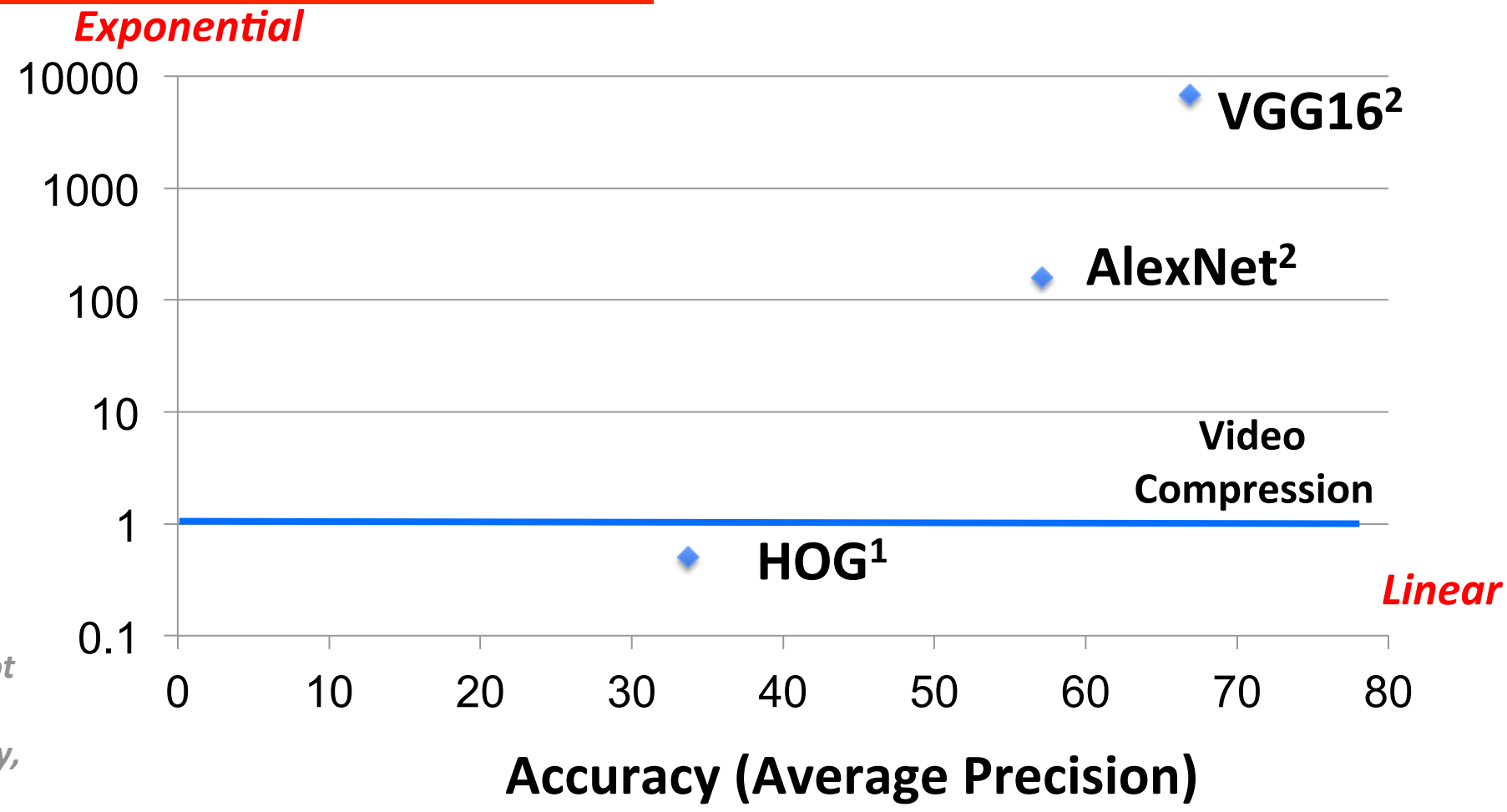



Object Detection using **HOG features** and Deformable Parts Models [Suleiman et al., VLSI 2016]




Eyeriss: **Convolutional Neural Networks** [Chen et al., ISSCC 2016, ISCA 2016]

# Features: Energy vs. Accuracy




**Energy/ Pixel (nJ)**

*Measured in 65nm**

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

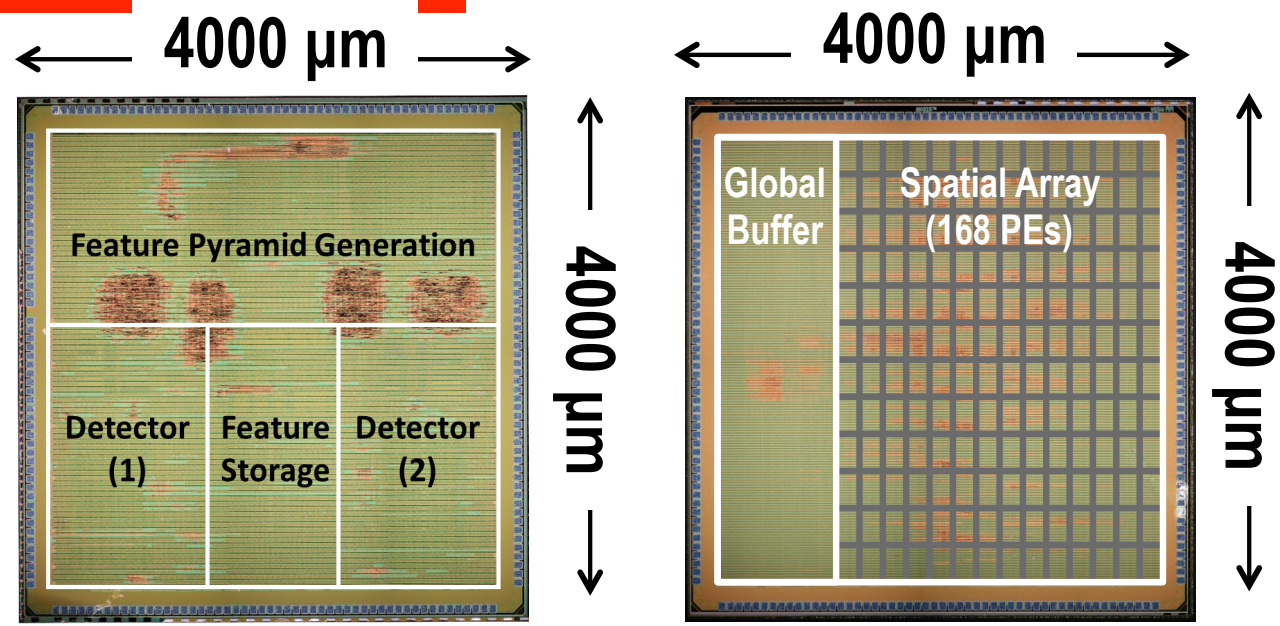
** Only feature extraction. Does not include data, augmentation, ensemble and classification energy, etc.*

*Measured in on VOC 2007 Dataset*

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

[Suleiman et al., ISCAS 2017]

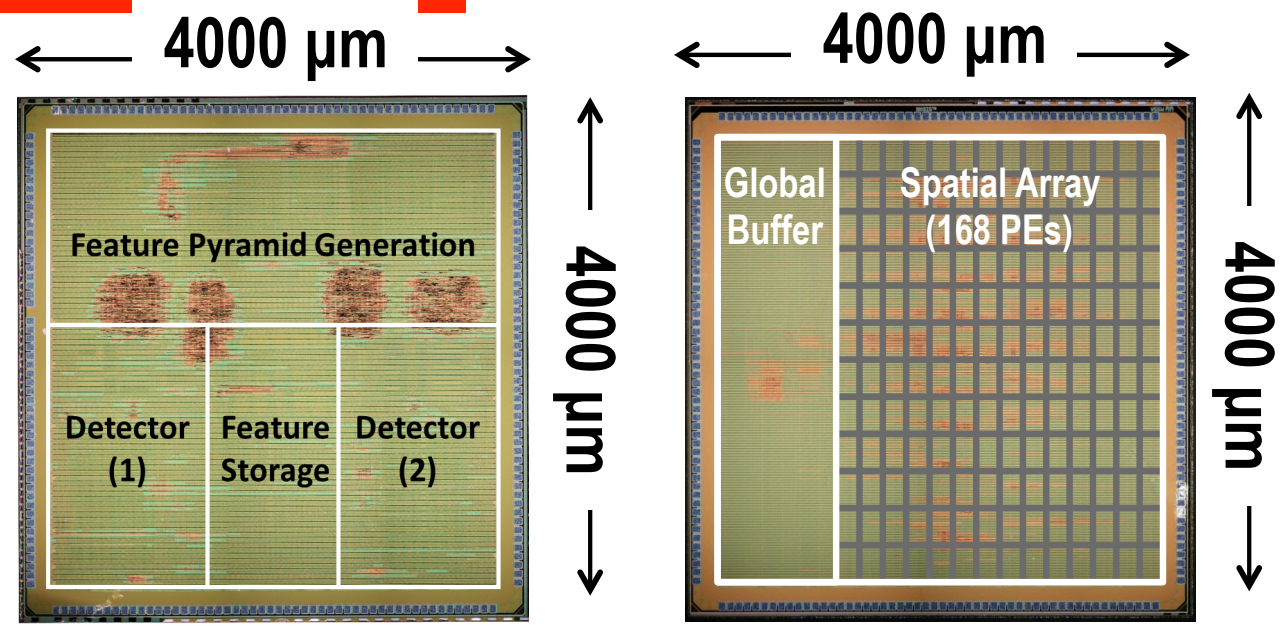
# HOG vs. CNN: Hardware Cost




| | HOG [VLSI 2016] | CNN [ISSCC 2016] |
|---|---|---|
| **Technology** | TSMC LP 65nm | TSMC LP 65m |
| **Gate Count (kgates)** | 893 | 1176 |
| **Memory (kB)** | 159 | 181.5 |

Similar Hardware Cost (comparable with Video Compression)

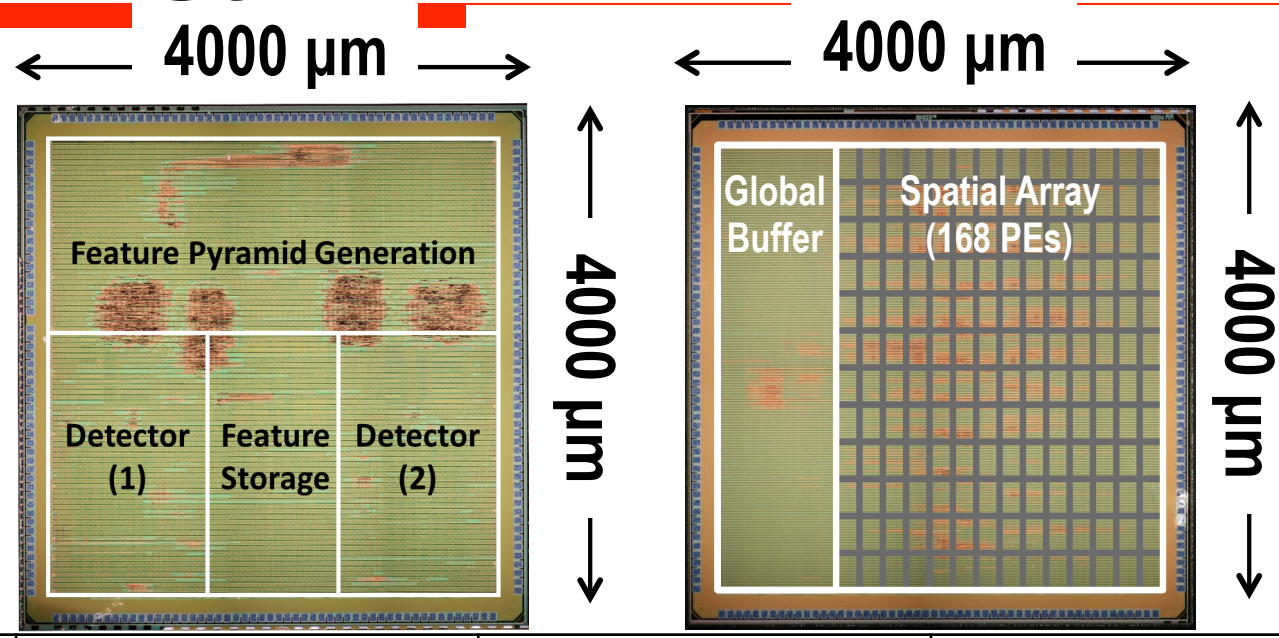
# HOG vs. CNN: Throughput




| | HOG | CNN (AlexNet) | CNN (VGG-16) |
|---|---|---|---|
| **Throughput (Mpixels/s)** | 62.5 | 1.8 | 0.04 |
| **GOP/Mpixel** | 0.7 | 25.8 | 610.3 |
| **Throughput (GOPS)** | 46.0 | 46.2 | 21.4 |

Throughput gap explained by GOP/Mpixel gap

# HOG vs. CNN: Energy and DRAM Access




| | HOG | CNN (AlexNet) | CNN (VGG-16) |
|---|---|---|---|
| Energy (nJ/pixel) | 0.5 | 155.5 | 6742.9 |
| GOP/Mpixel | 0.7 | 25.8 | 610.3 |
| Energy (GOPS/W) | 1570 | 166.2 | 90.7 |
| DRAM (B/pixel) | 1.0 | 74.7 | 2128.6 |

Energy gap larger than GOPS/W gap

# Energy Gap between CNN and HOG

- **CNNs require more operations per pixel**
  - AlexNet vs. HOG = **37x**
  - VGG-16 vs. HOG = **872x**
- **CNN requires a programmable architecture**
  - **Example:** AlexNet CONV layers have **2.3M weights** (assume 8-bits per weight); Area budget of HOG chip is **~1000 kgates, 150kB**
  - **<u>Design A:</u> Hard-wired weights**
    - Only have 10k multipliers with fixed weights **(>100x increase in area)**
  - **<u>Design B</u>: Store all weights on-chip**
    - Only store 150k weights on chip **(>10x increase in storage)**
  - **Support different shapes per layer and different weights**

# Opportunities in Joint Algorithm Hardware Design

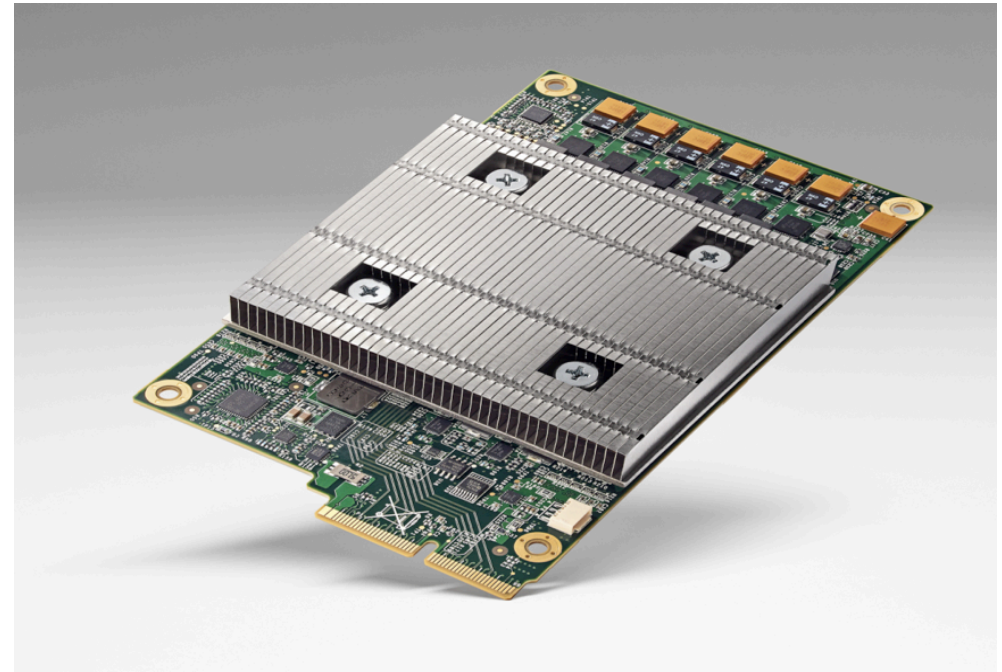
# Approaches

- 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

# Commercial Products using 8-bit Integer


Nvidia's Pascal (2016)

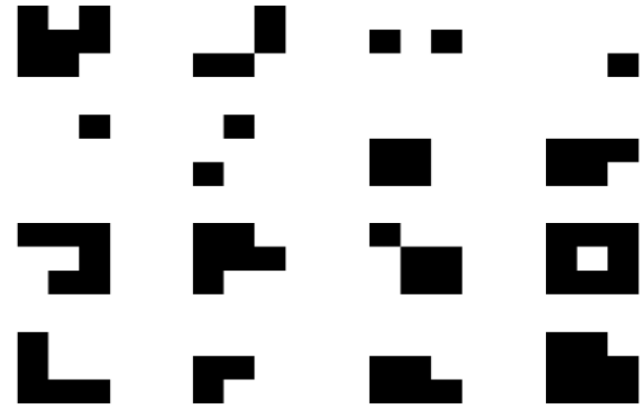


Google's TPU (2016)

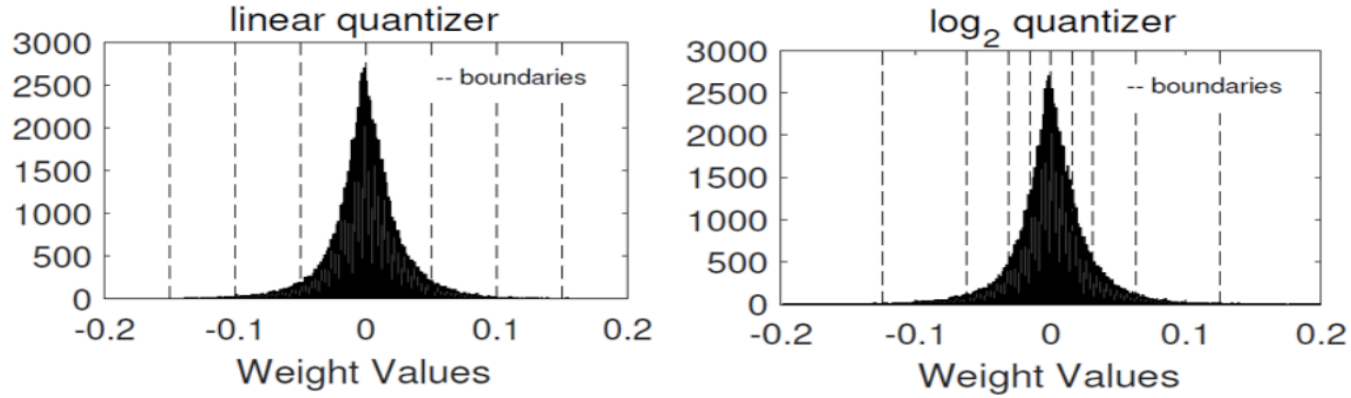
# Reduced Precision in Research

- **Reduce number of bits**
  - Binary Nets [Courbariaux, NIPS 2015]
- **Reduce number of unique weights**
  - Ternary Weight Nets [Li, arXiv 2016]
  - XNOR-Net [Rategari, ECCV 2016]
- **Non-Linear Quantization**
  - LogNet [Lee, ICASSP 2017]

*Binary Filters*



*Log Domain Quantization*



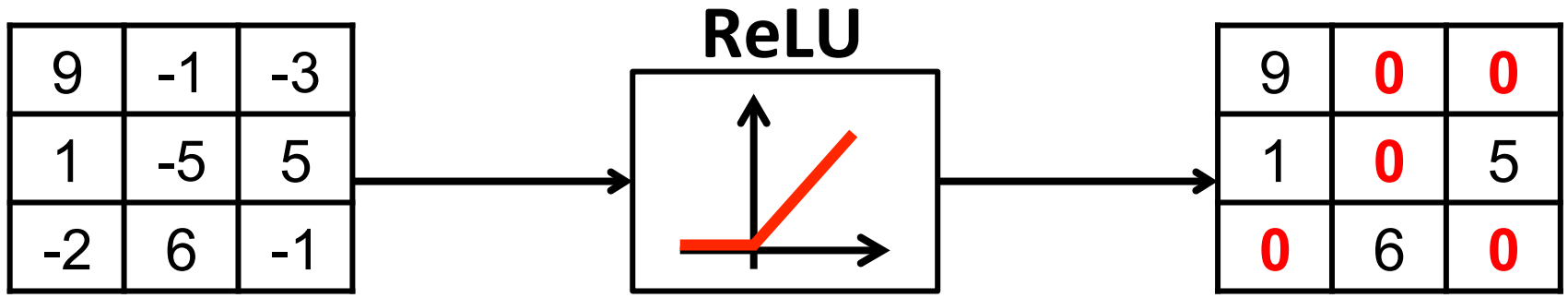
# Approaches

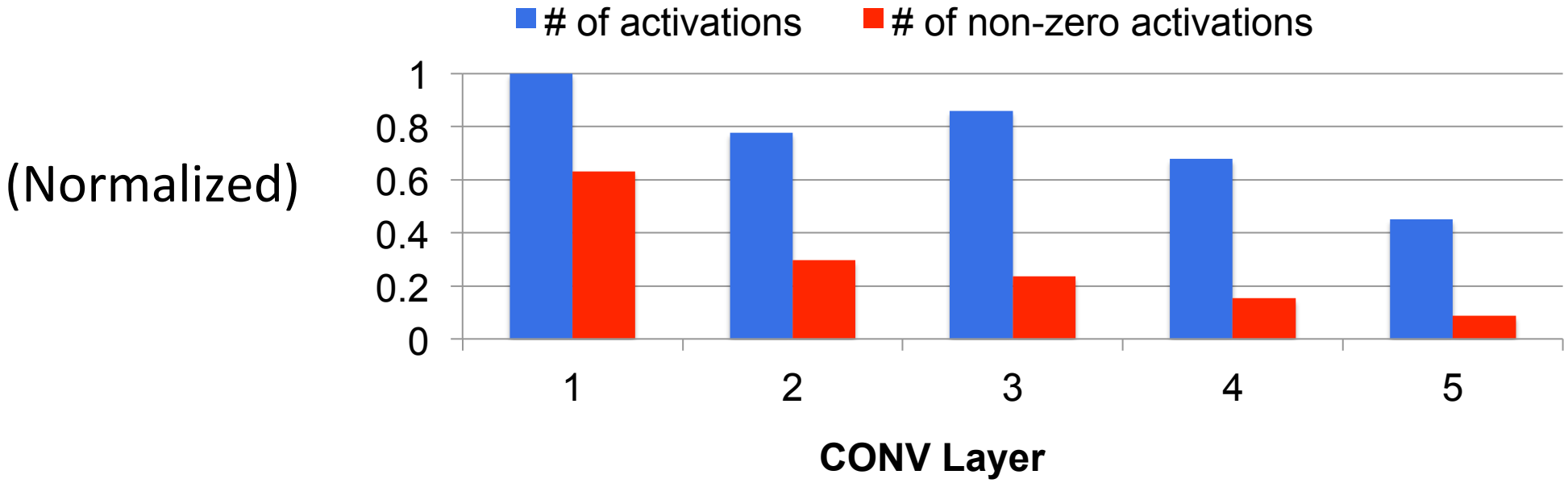
- Reduce size of operands for storage/compute
  - Floating point → Fixed point
  - Bit-width reduction
  - Non-linear quantization

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

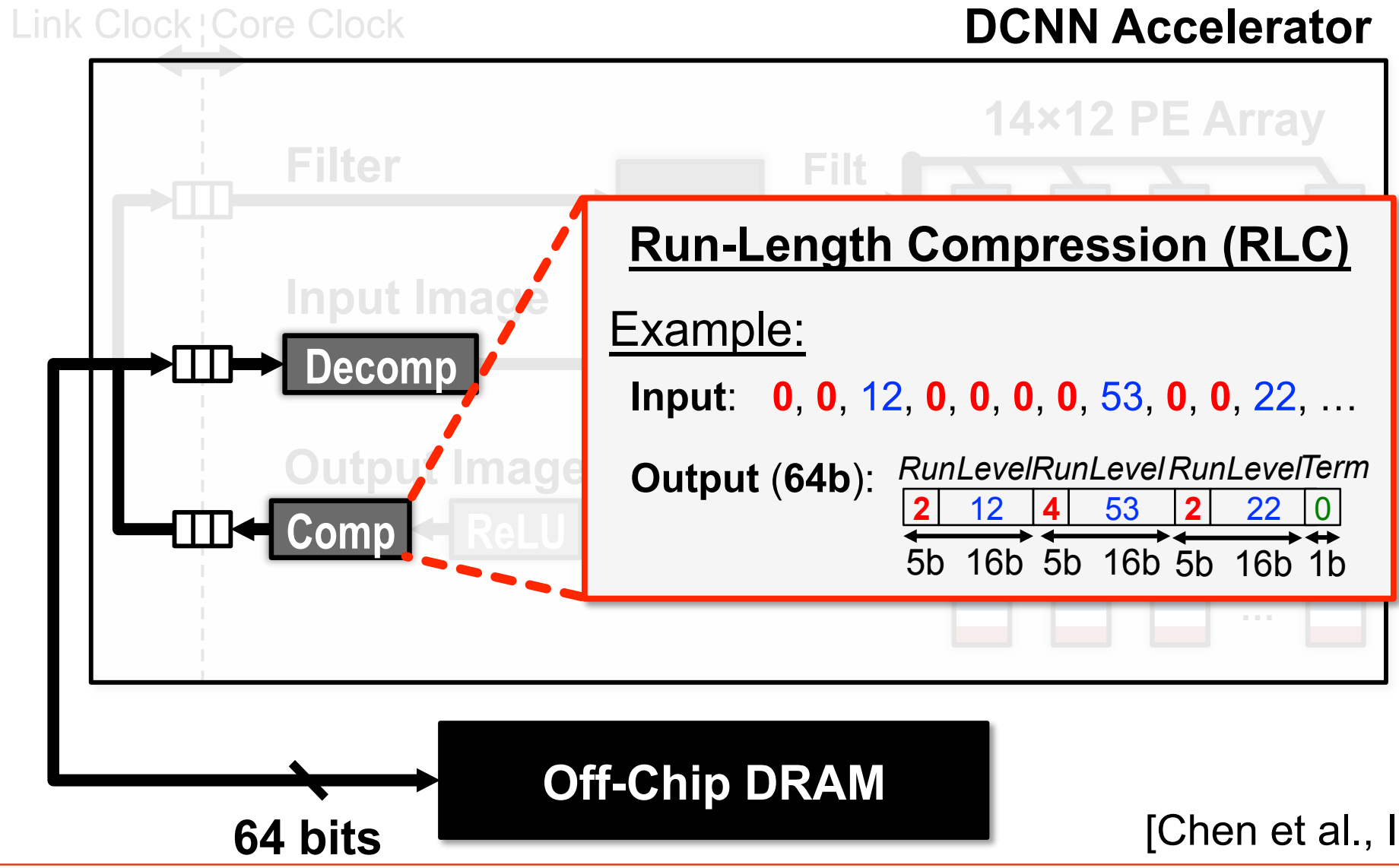
# Sparsity in Data

Many **zeros** in **output fmaps** after **ReLU**



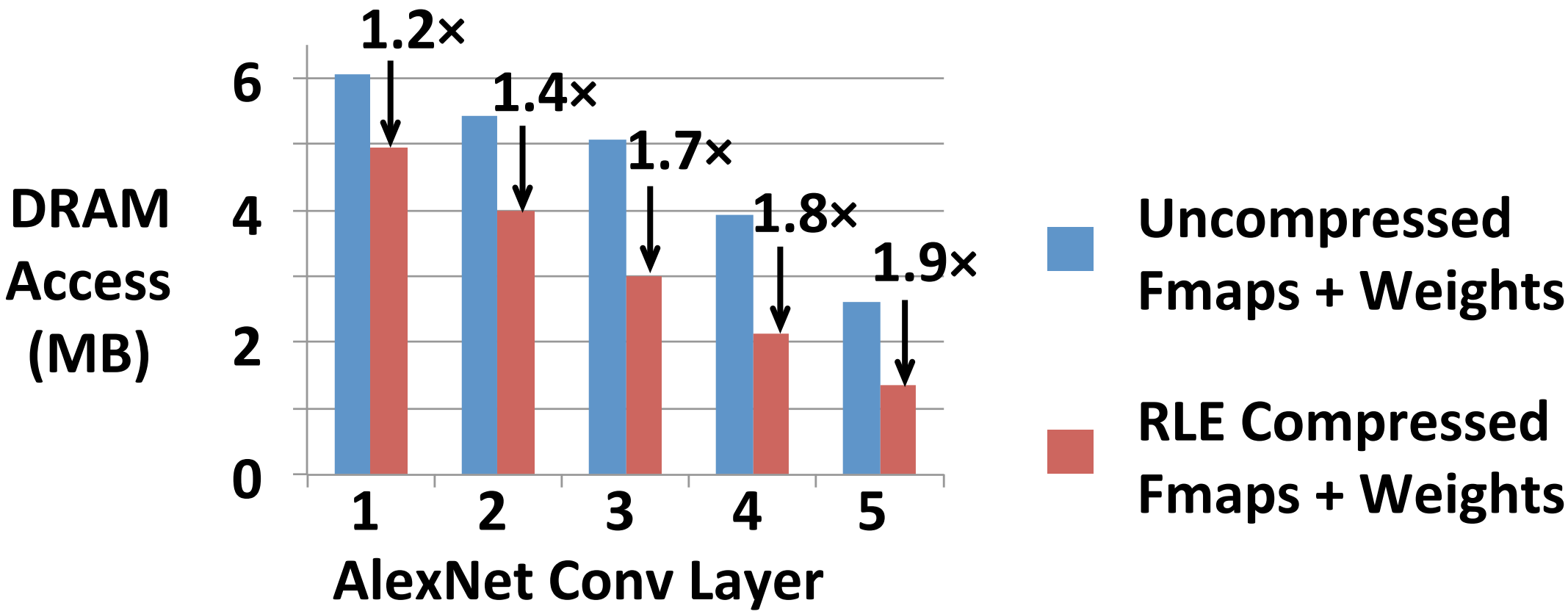



# I/O Compression in Eyeriss




[Chen et al., ISSCC 2016]

# Compression Reduces DRAM BW

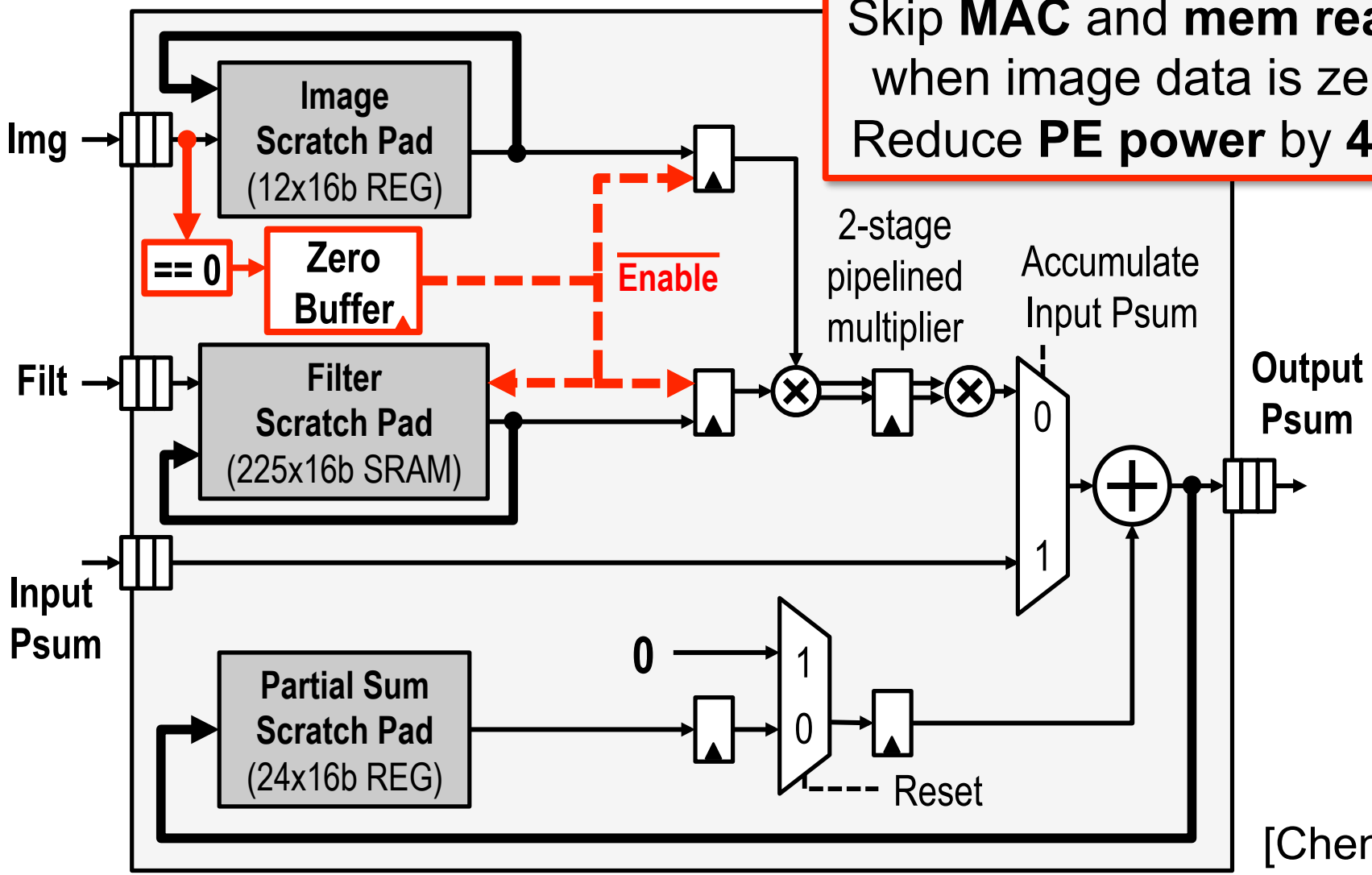



Simple RLC within 5% - 10% of theoretical entropy limit

[Chen et al., ISSCC 2016]

# Data Gating / Zero Skipping in Eyeriss

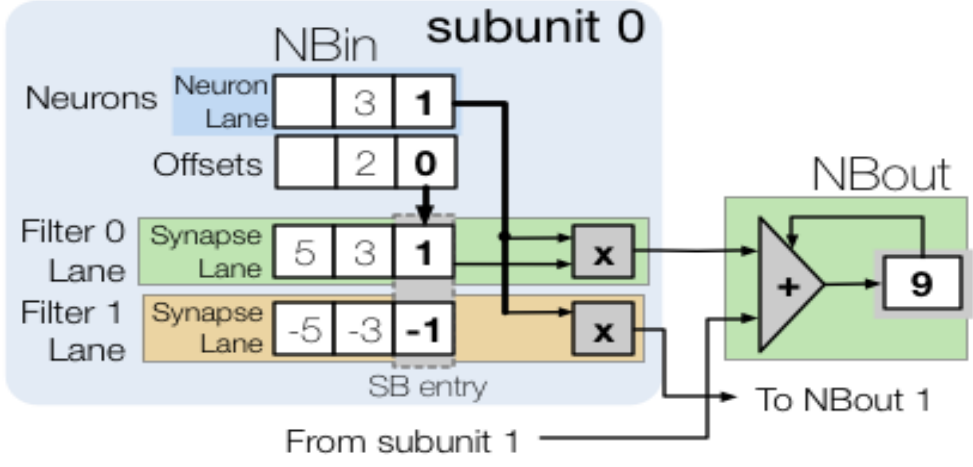
Skip **MAC** and **mem reads** when image data is zero. Reduce **PE power** by **45%**



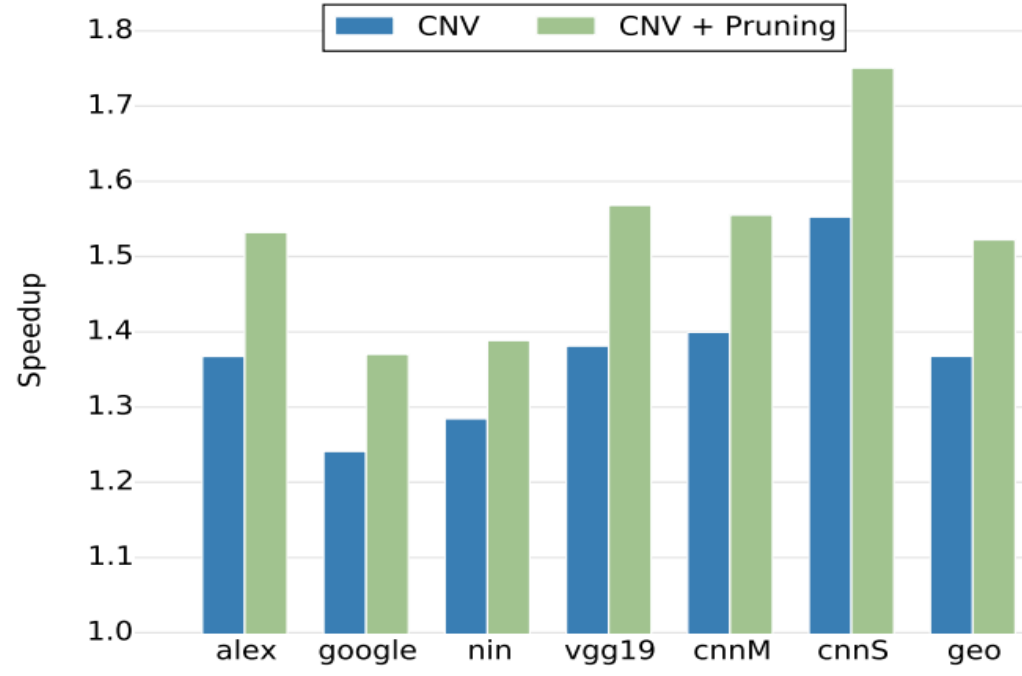

[Chen et al., ISSCC 2016]

# Cnvlutin

- Process Convolution Layers
- Built on top of DaDianNao (4.49% area overhead)
- Speed up of 1.37x (1.52x with activation pruning)



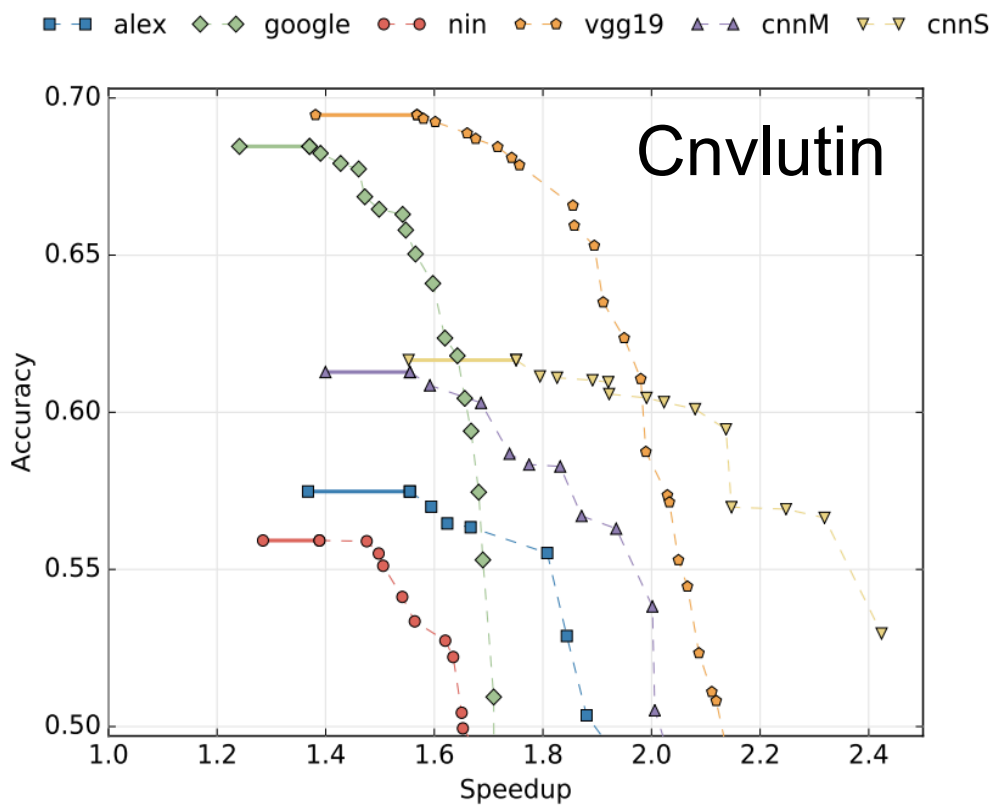

[Albericio et al., ISCA 2016]



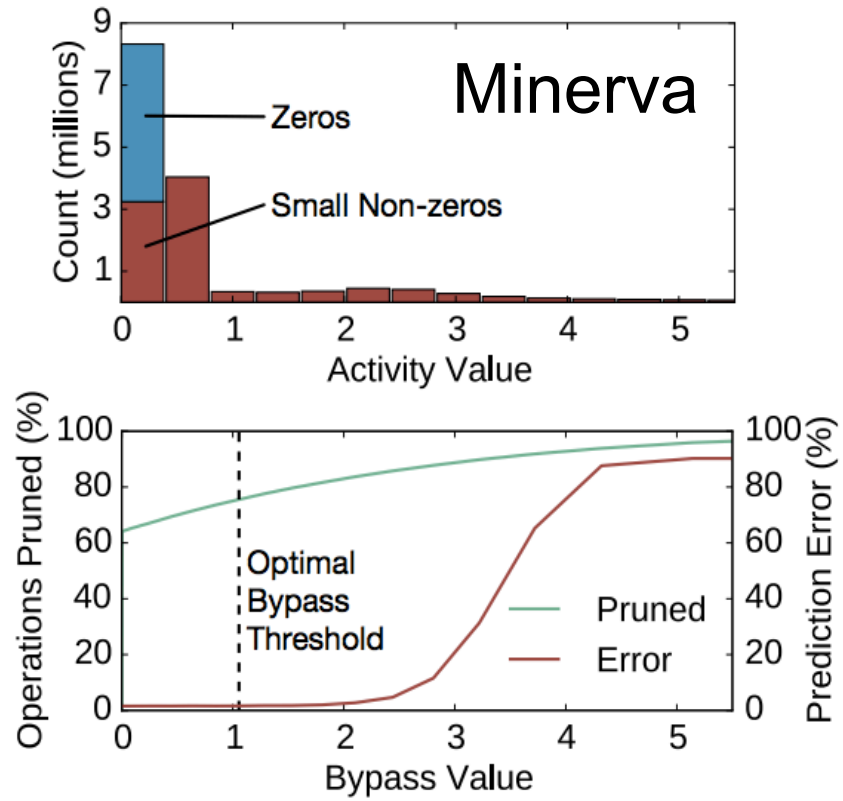
# Pruning Activations

## Remove small activation values

***Speed up 11% (ImageNet)***




***Reduce power 2x (MNIST)***



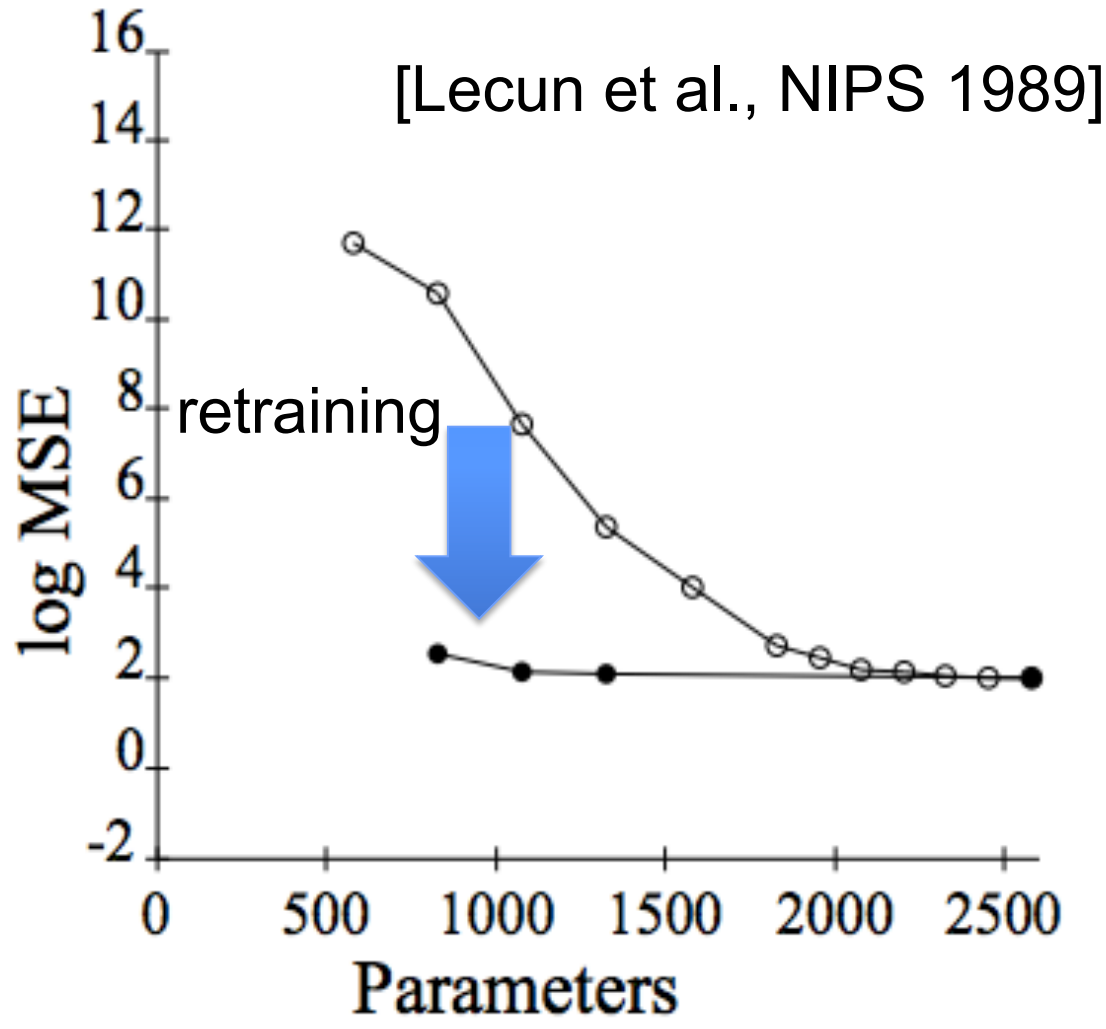

[Albericio et al., ISCA 2016]

[Reagen et al., ISCA 2016]

# Pruning – Make Weights Sparse

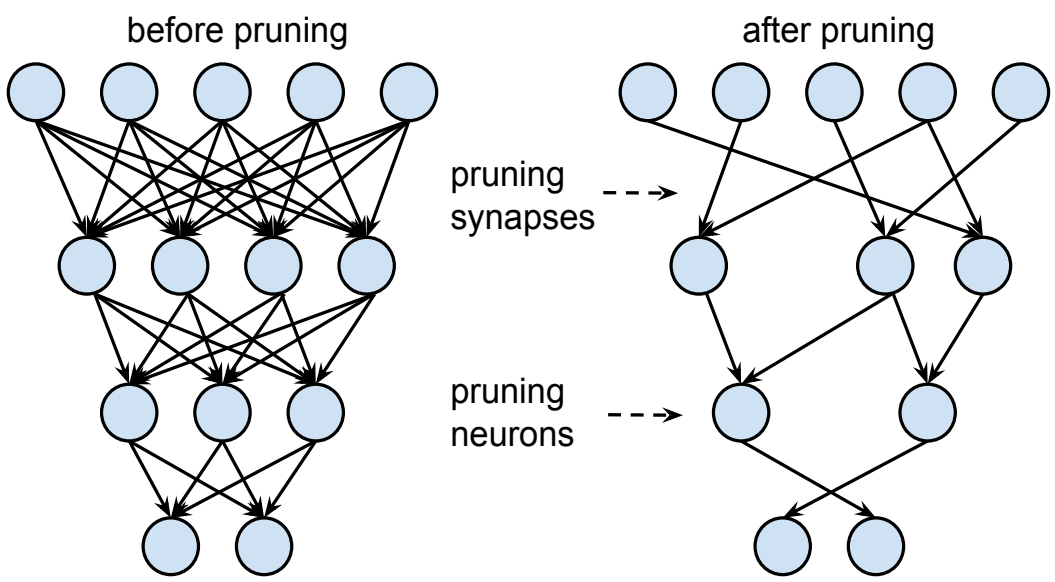
## Optimal Brain Damage

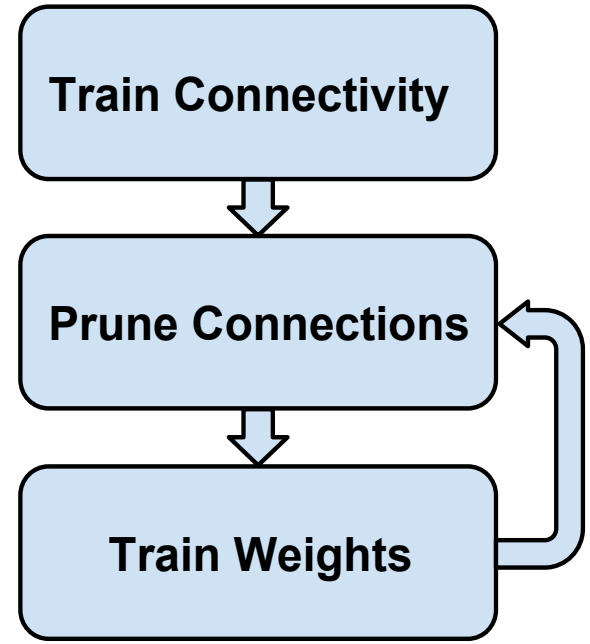
1. Choose a reasonable network architecture
2. Train network until reasonable solution obtained
3. Compute the second derivative for each weight
4. Compute saliencies (i.e. impact on training error) for each weight
5. Sort weights by saliency and delete low-saliency weights
6. Iterate to step 2



# Pruning – Make Weights Sparse

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




***Example:*** *AlexNet*
***Weight Reduction:*** *CONV layers 2.7x, FC layers 9.9x*
*(Most reduction on fully connected layers)*
***Overall:*** *9x weight reduction, 3x MAC reduction*

[Han et al., NIPS 2015]

# Compression of Weights & Activations

- Compress weights and activations between DRAM and accelerator
- Variable Length / Huffman Coding

  Example:

  Value: **16'b0** → Compressed Code: {**1'b0**}

  Value: **16'bx** → Compressed Code: {**1'b1**, **16'bx**}

- Tested on AlexNet → 2× overall BW Reduction

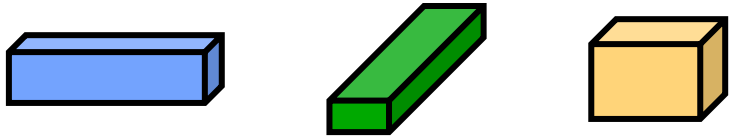
| Layer | Filter / Image bits (0%) | Filter / Image BW Reduc. | IO / HuffIO (MB/frame) | Voltage (V) | MMACs/ Frame | Power (mW) | Real (TOPS/W) |
|---|---|---|---|---|---|---|---|
| General CNN | 16 (0%) / 16 (0%) | 1.0x | | 1.1 | — | **288** | **0.3** |
| AlexNet l1 | 7 (21%) / 4 (29%) | 1.17x / 1.3x | 1 / 0.77 | 0.85 | 105 | 85 | 0.96 |
| AlexNet l2 | 7 (19%) / 7 (89%) | 1.15x / **5.8x** | 3.2 / 1.1 | 0.9 | 224 | 55 | 1.4 |
| AlexNet l3 | 8 (11%) / 9 (82%) | 1.05x / 4.1x | 6.5 / 2.8 | 0.92 | 150 | 77 | 0.7 |
| AlexNet l4 | 9 (04%) / 8 (72%) | 1.00x / 2.9x | 5.4 / 3.2 | 0.92 | 112 | 95 | 0.56 |
| AlexNet l5 | 9 (04%) / 8 (72%) | 1.00x / 2.9x | 3.7 / 2.1 | 0.92 | 75 | 95 | 0.56 |
| Total / avg. | — | — | 19.8 / **10** | — | — | **76** | **0.94** |
| LeNet-5 l1 | 3 (35%) / 1 (87%) | 1.40x / 5.2x | 0.003 / 0.001 | 0.7 | 0.3 | 25 | 1.07 |
| LeNet-5 l2 | 4 (26%) / 6 (55%) | 1.25x / 1.9x | 0.050 / 0.042 | 0.8 | 1.6 | 35 | 1.75 |
| Total / avg. | — | — | 0.053 / **0.043** | — | — | **33** | **1.6** |

[Moons et al., VLSI 2016; Han et al., ICLR 2016]

# Key Metrics for Embedded DNN

- Accuracy → Measured on Dataset
- Storage Footprint → Number of Weights
- Speed → Number of MACs
- Energy → ?

# Energy-Evaluation Methodology



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

**Hardware Energy Costs of each MAC and Memory Access**

Memory Accesses Optimization

# of MACs Calculation

# acc. at mem. level **1**

# acc. at mem. level **2**

⋮

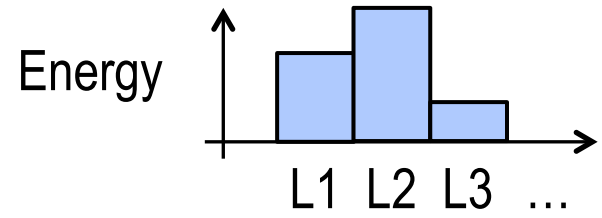
# acc. at mem. level **n**

# of MACs

$E_{data}$

$E_{comp}$

**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 Estimation Tool

Website: https://energyestimation.mit.edu/

## Deep Neural Network Energy Estimation Tool

### Overview

This Deep Neural Network Energy Estimation Tool is used for evaluating and designing energy-efficient deep neural networks that are critical for embedded deep learning processing. Energy estimation was used in the development of the energy-aware pruning method (Yang et al., CVPR 2017), which reduced the energy consumption of AlexNet and GoogLeNet by 3.7x and 1.6x, respectively, with less than 1% top-5 accuracy loss. This website provides a simplified version of the energy estimation tool for shorter runtime (around 10 seconds).

### Input

To support the variety of toolboxes, this tool takes a single network configuration file. The network configuration file is a txt file, where each line denotes the configuration of a CONV/FC layer. The format of each line is:

height nChannels nZeroEntries y bottom right
width nMapsOrFilts bitwidth x top left

2 , 27 , 27 , 96 , 44 , 3.5731e+05 , 16 , 5 , 5 , 48 , 256 , 0 , 16 , 27 , 27 , 256 , 44 , 6.623e+06 , 16 , 1 , 1 , 2 , 2 , 2 , 2

Layer_Index Conf_IfMap Conf_Filt Conf_OfMap Stride Padding

- Layer Index: the index of the layer, from 1 to the number of layers. It should be the same as the line number.
- Conf_IfMap, Conf_Filt, Conf_OfMap: the configuration of the input feature maps, the filters and the output feature maps. The configuration of each of the three data types is in the format of "height width number_of_channels number_of_maps_or_filts number_of_zero_entries bitwidth_in_bits".
- Stride: the stride of this layer. It is in the format of "stride_y stride_x".
- Pad: the amount of input padding. It is in the format of "pad_top pad_bottom pad_left pad_right".

Therefore, there will be 25 entries separated by commas in each line.

### Running the Estimation Model

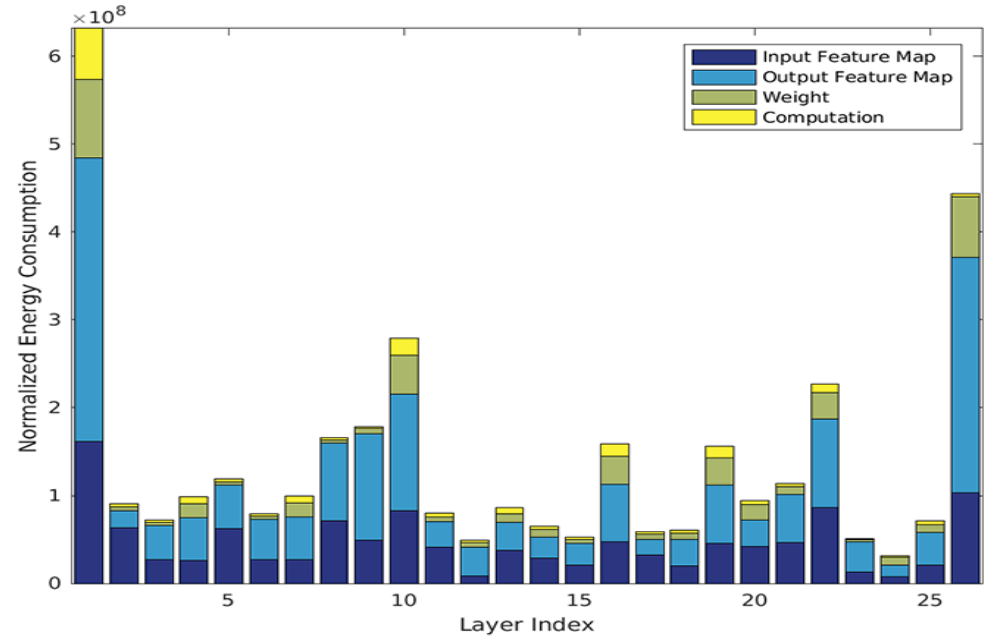
After creating your text file, follow these steps to upload your text file and run the estimation model:

1. Check the "I am not a robot" checkbox and complete the Google reCAPTCHA challenge. Help us prevent spam.
2. Click the "Choose File" button below to choose your text file from your computer.
3. Click the "Run Estimation Model" button below to upload your text file and run the estimation model.

## Input DNN Configuration File

```
Layer_Index,Input_Feature_Map,Output_Feature_Map,Weight,Computation
1,161226686.785535,323273662,88858340.625,58290651
2,63540403.7543396,19104256.6840292,4770357.50868125,3263307.50868125
3,26787638.0555562,39583335.5555542,3272222.77777708,2285942.77777708
4,26018817.2746958,48841502.8019458,15927826.1926396,7847418.06763958
5,62285050.8236438,49433953.294575,4188476.6472875,3227376.6472875
6,27267689.7685187,45381705.7407417,3740581.20370417,2666586.20370417
7,26787131.0480146,48586492.3413917,16216779.2956958,8136371.17069583
```

## Output DNN energy breakdown across layers

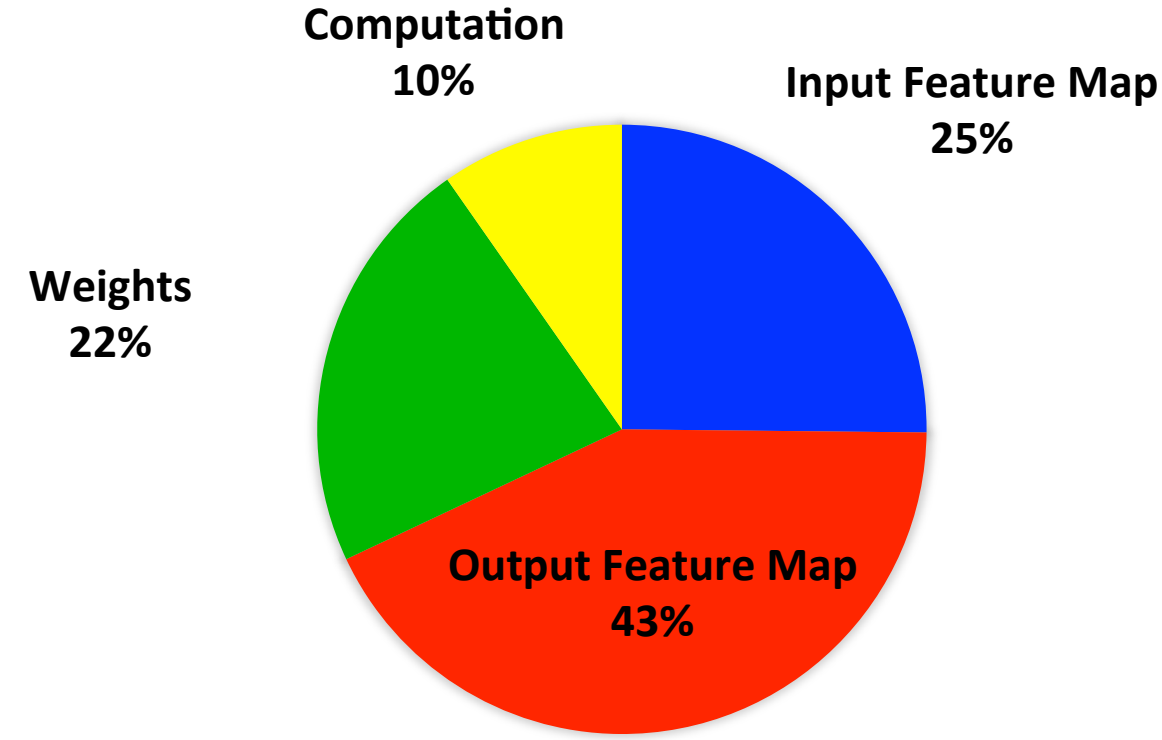



[Yang et al., CVPR 2017]

# Key Observations

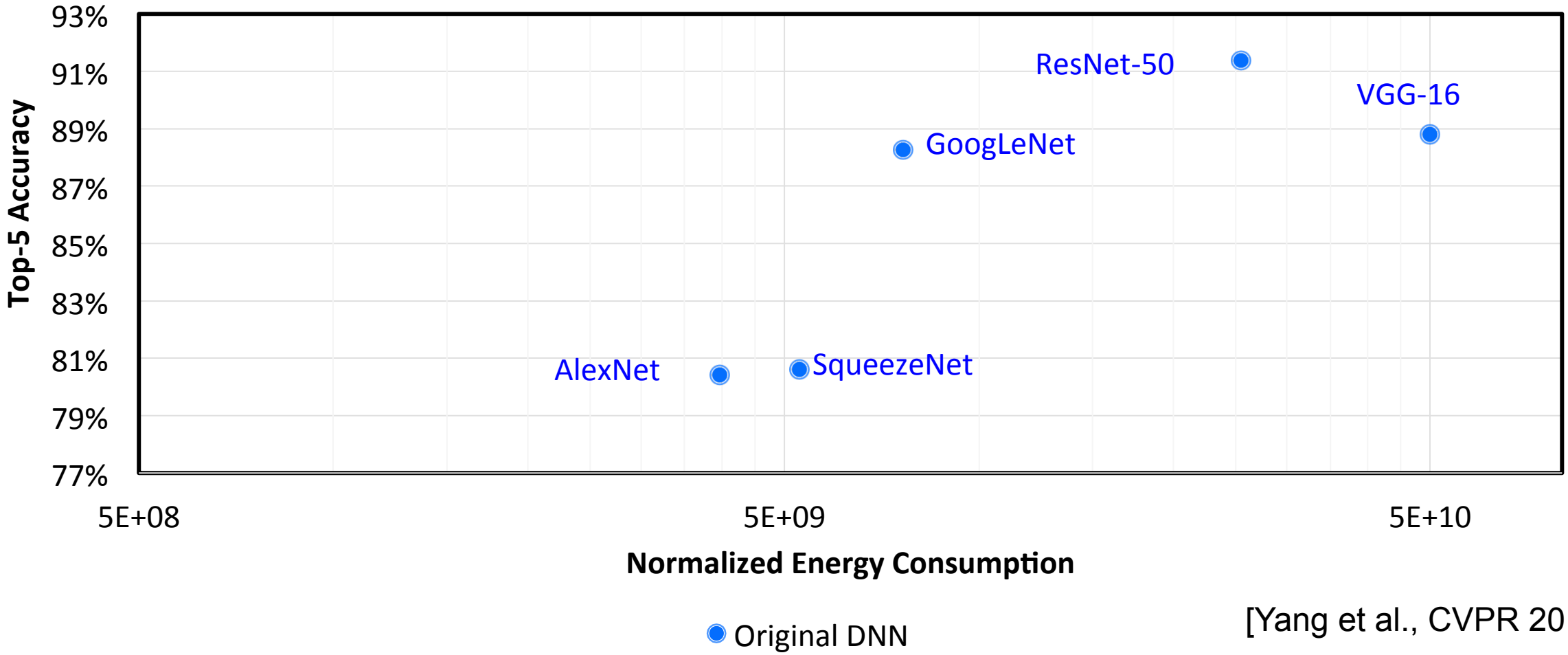
- Number of weights ***alone*** is not a good metric for energy
- **All data types** should be considered

**Energy Consumption of GoogLeNet**




[Yang et al., CVPR 2017]

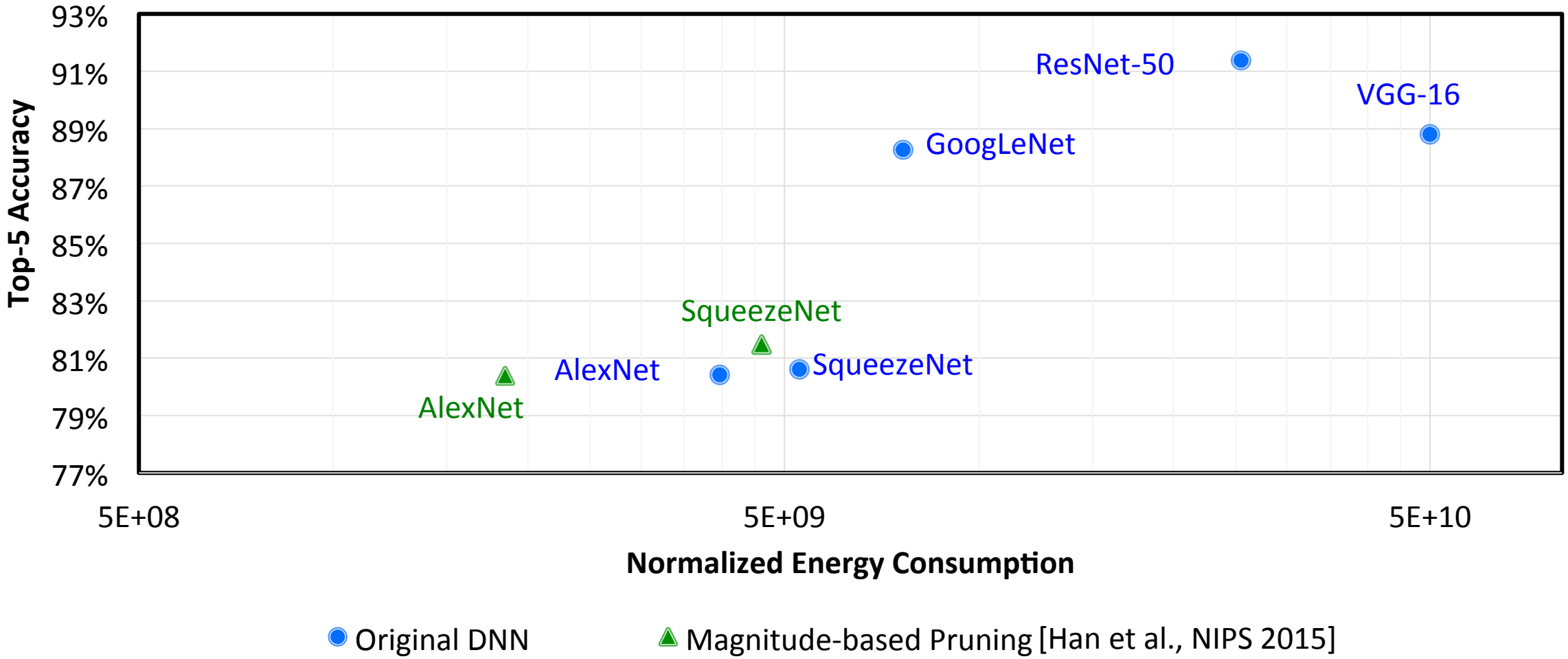
# Energy Consumption of Existing DNNs




[Yang et al., CVPR 2017]

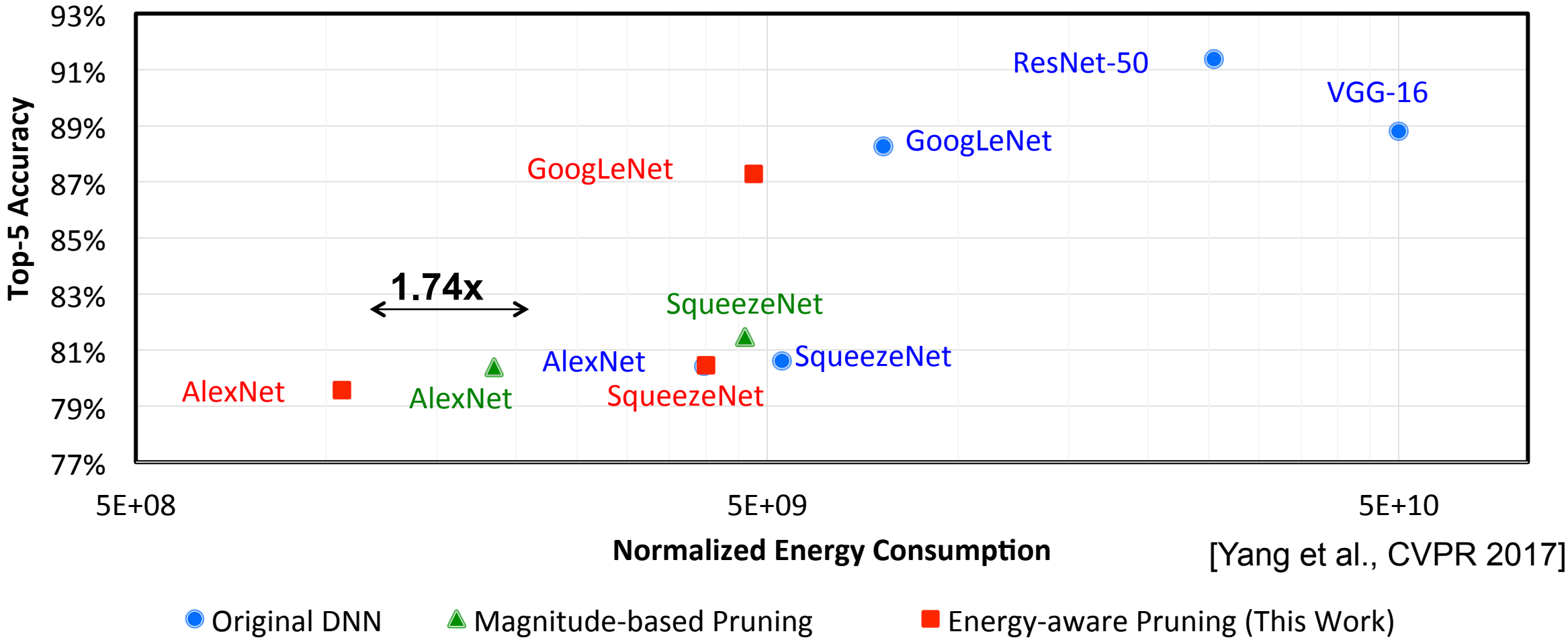
Deeper CNNs with fewer weights do not necessarily consume less energy than shallower CNNs with more weights

# Magnitude-based Weight Pruning




Reduce number of weights by **removing small magnitude weights**

# Energy-Aware Pruning



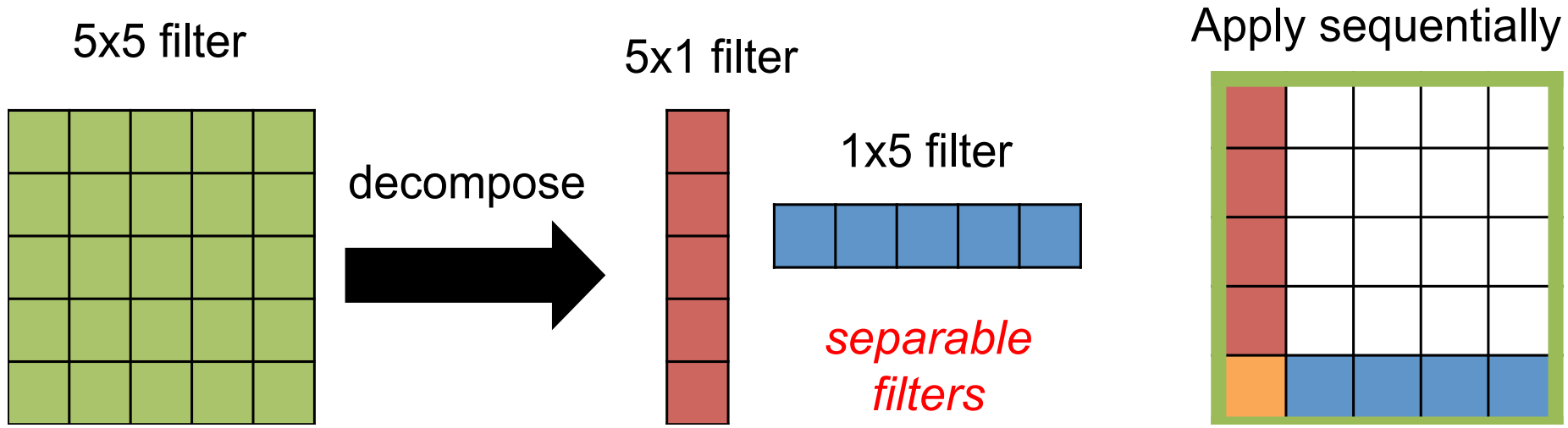

[Yang et al., CVPR 2017]

Remove weights from layers **in order of highest to lowest energy 3.7x reduction in AlexNet / 1.6x reduction in GoogLeNet**

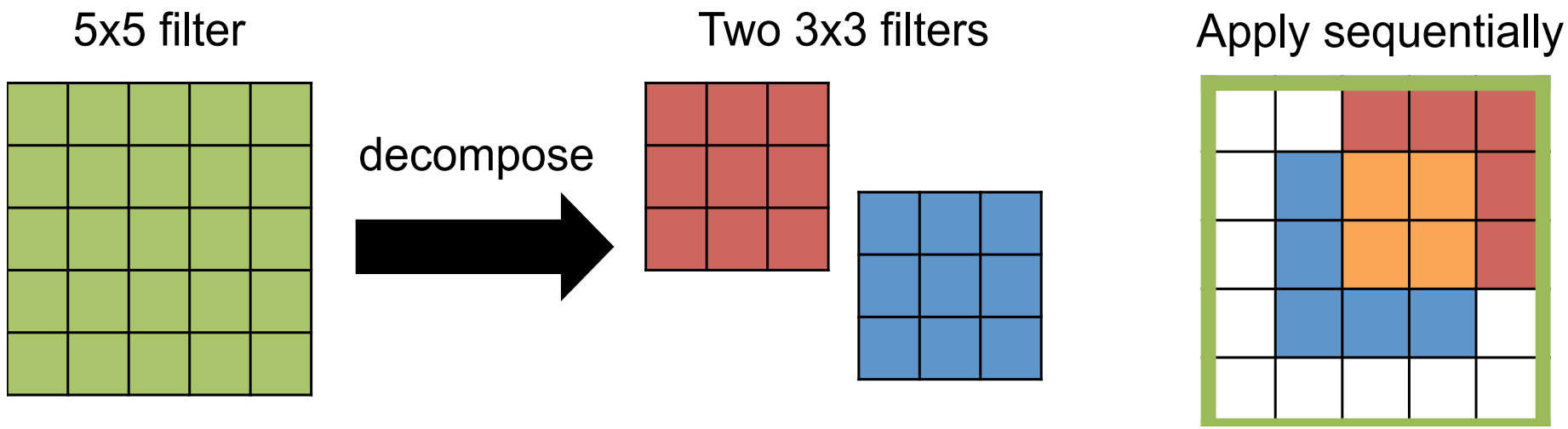
# Network Architecture Design

Build Network with series of Small Filters

**GoogleNet/ Inception v3**



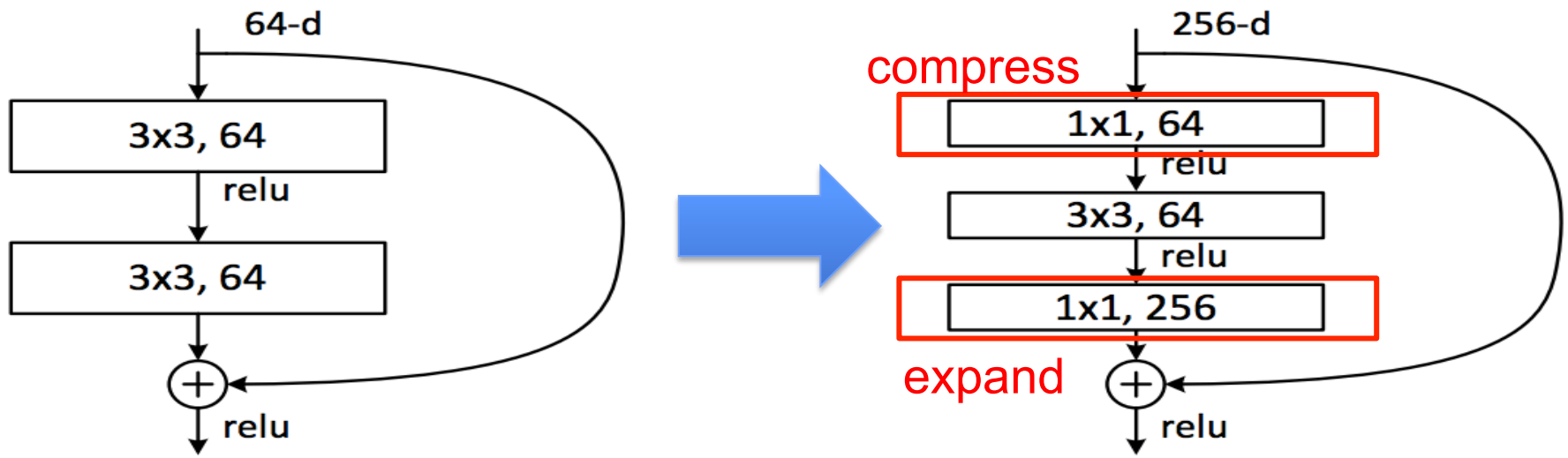

**VGG-16**



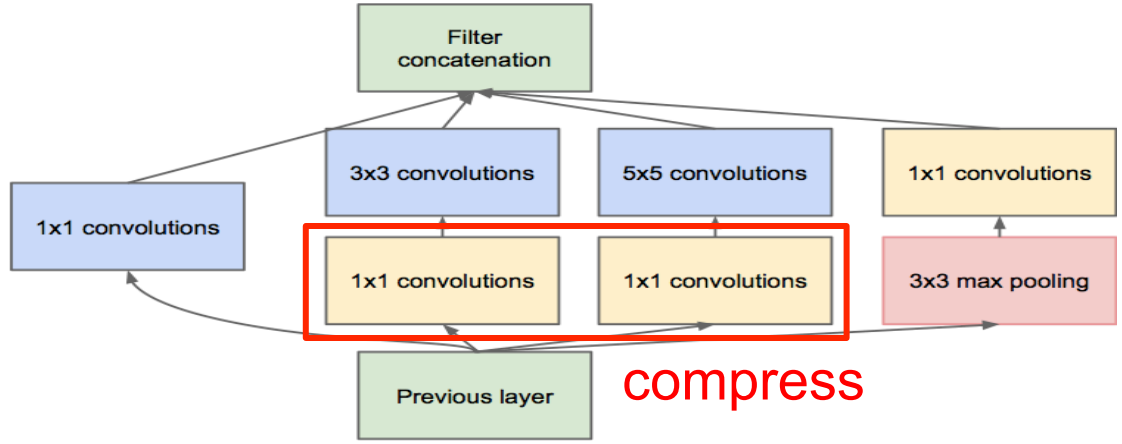
# Bottleneck in Popular DNN models

Reduce size and computation with 1x1 Filter (**bottleneck**)

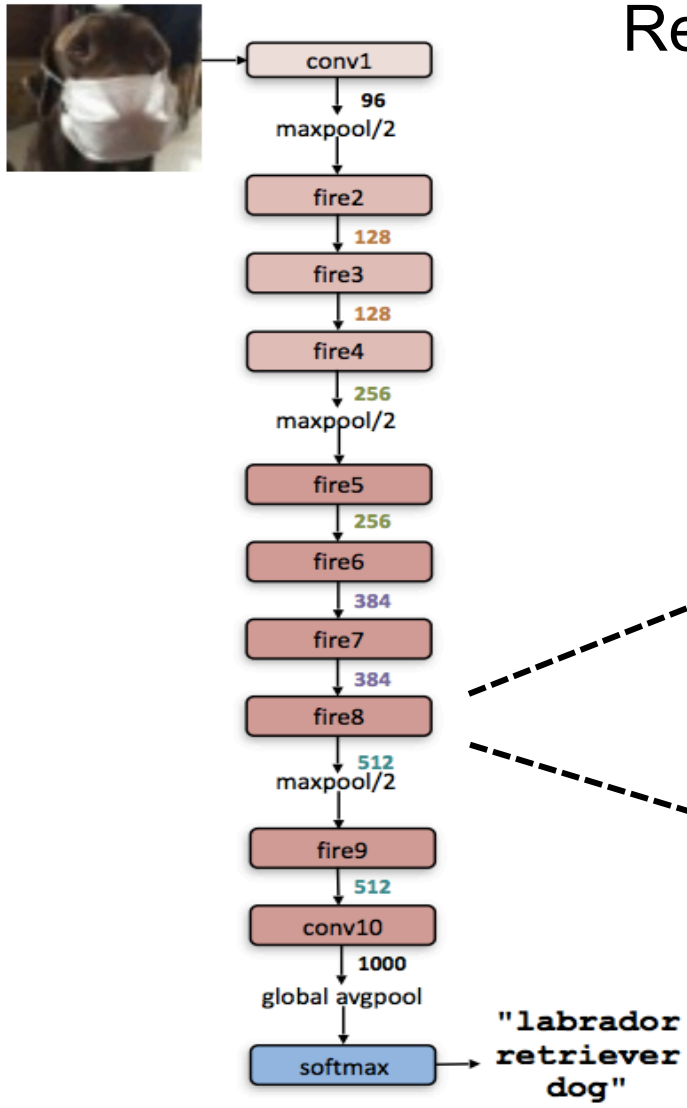
**ResNet**




**GoogleNet**



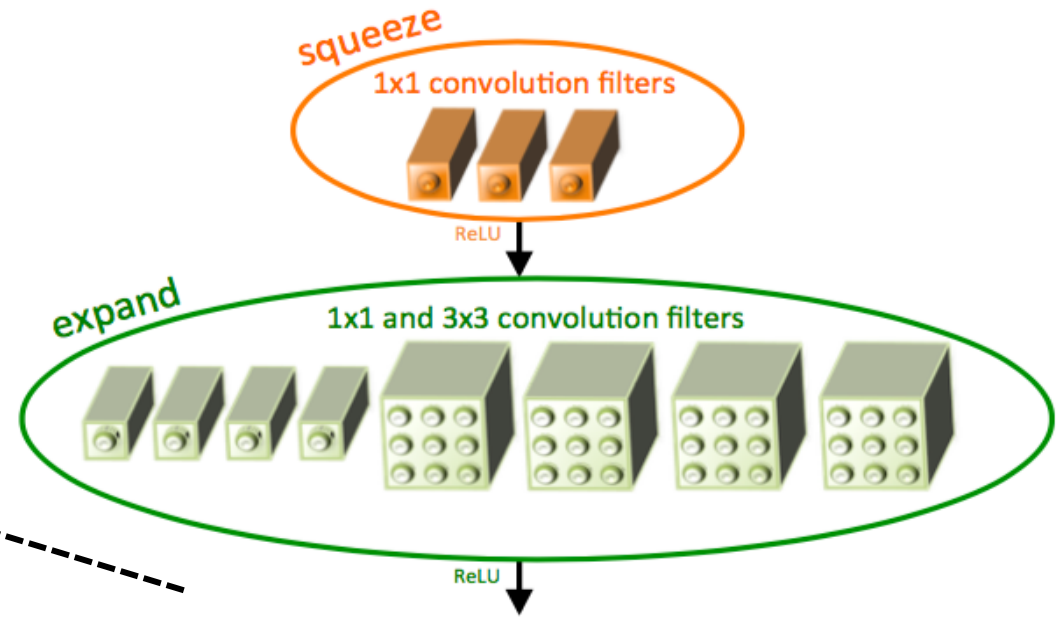
# SqueezeNet




Reduce weights by reducing number of input channels by "squeezing" with 1x1

**50x fewer weights than AlexNet (no accuracy loss)**

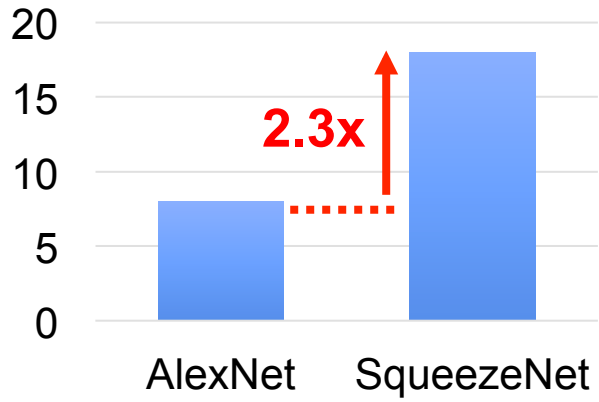
## Fire Module



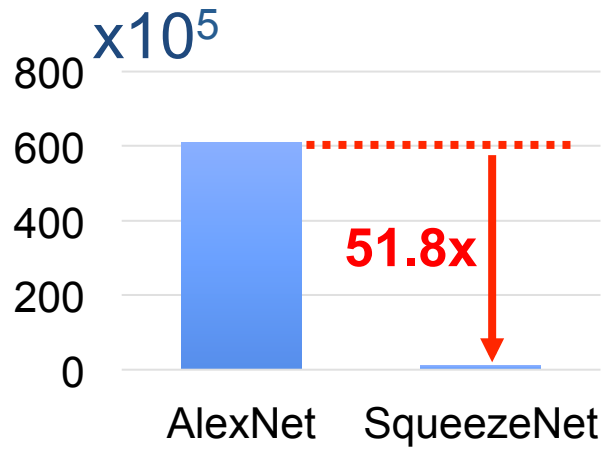

[Iandola et al., ICLR 2017]

# SqueezeNet

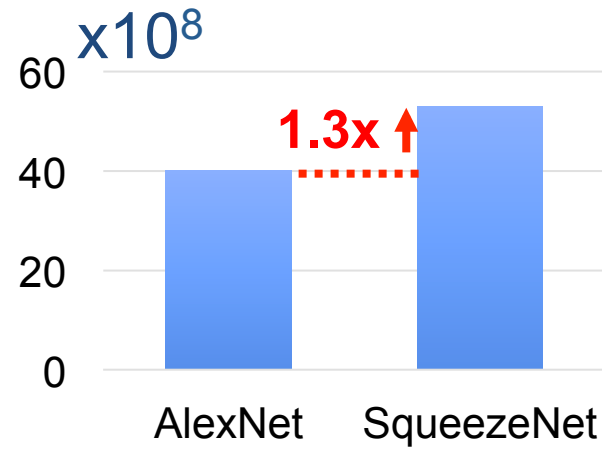
Deeper CNNs with fewer weights do not necessarily consume less energy than shallower CNNs with more weights




**# of Layers**




**# of Weights**




**Normalized Energy**

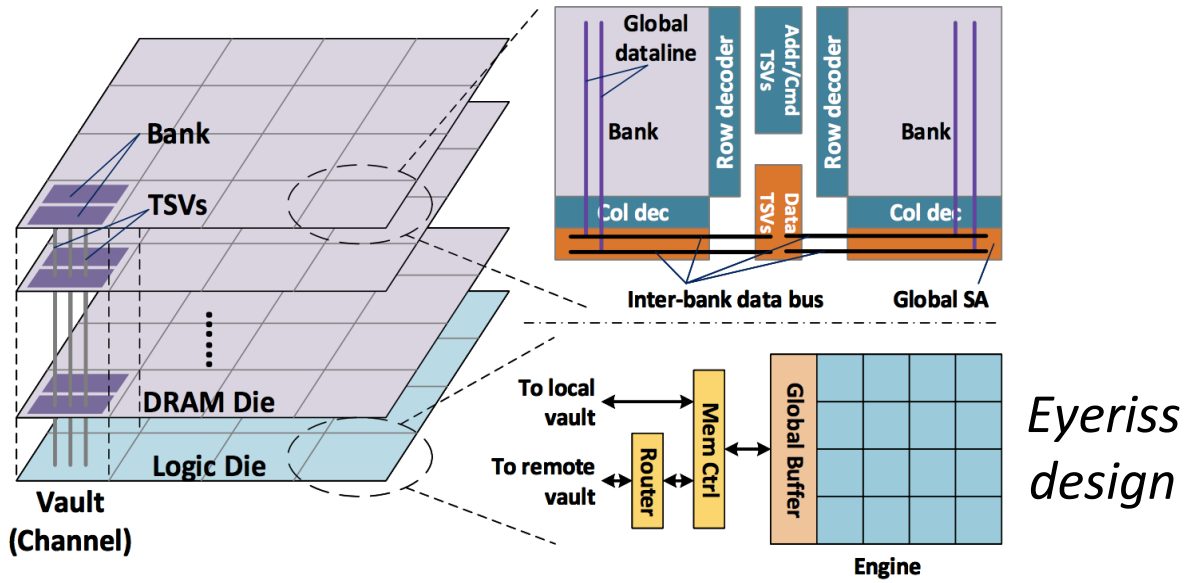
Measured with energy estimation tool: http://energyestimation.mit.edu

# Opportunities in Advanced Technologies

# Advanced Memory Technologies

Many new memories and devices explored to reduce data movement

## Stacked DRAM



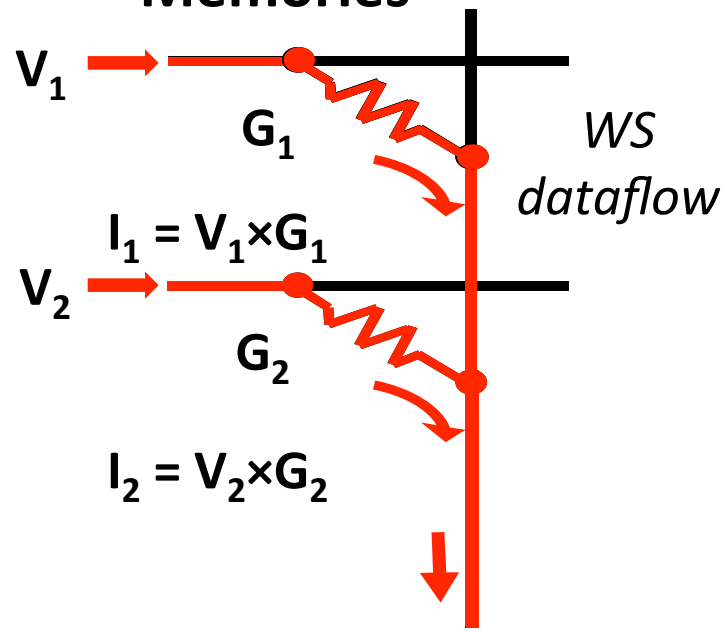

*Eyeriss design*

[Gao et al., Tetris, ASPLOS 2017]
[Kim et al., NeuroCube, ISCA 2016]

## eDRAM

[Chen et al., DaDianNao, MICRO 2014]

## Non-Volatile Resistive Memories




*WS dataflow*

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

[Shafiee et al., ISCA 2016]
[Chi et al., PRIME, ISCA 2016]

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

# References

**Overview Paper**

V. Sze, Y.-H. Chen, T-J. Yang, J. Emer, "*Efficient Processing of Deep Neural Networks: A Tutorial and Survey,*" Proceedings of the IEEE, December 2017
https://arxiv.org/pdf/1703.09039.pdf

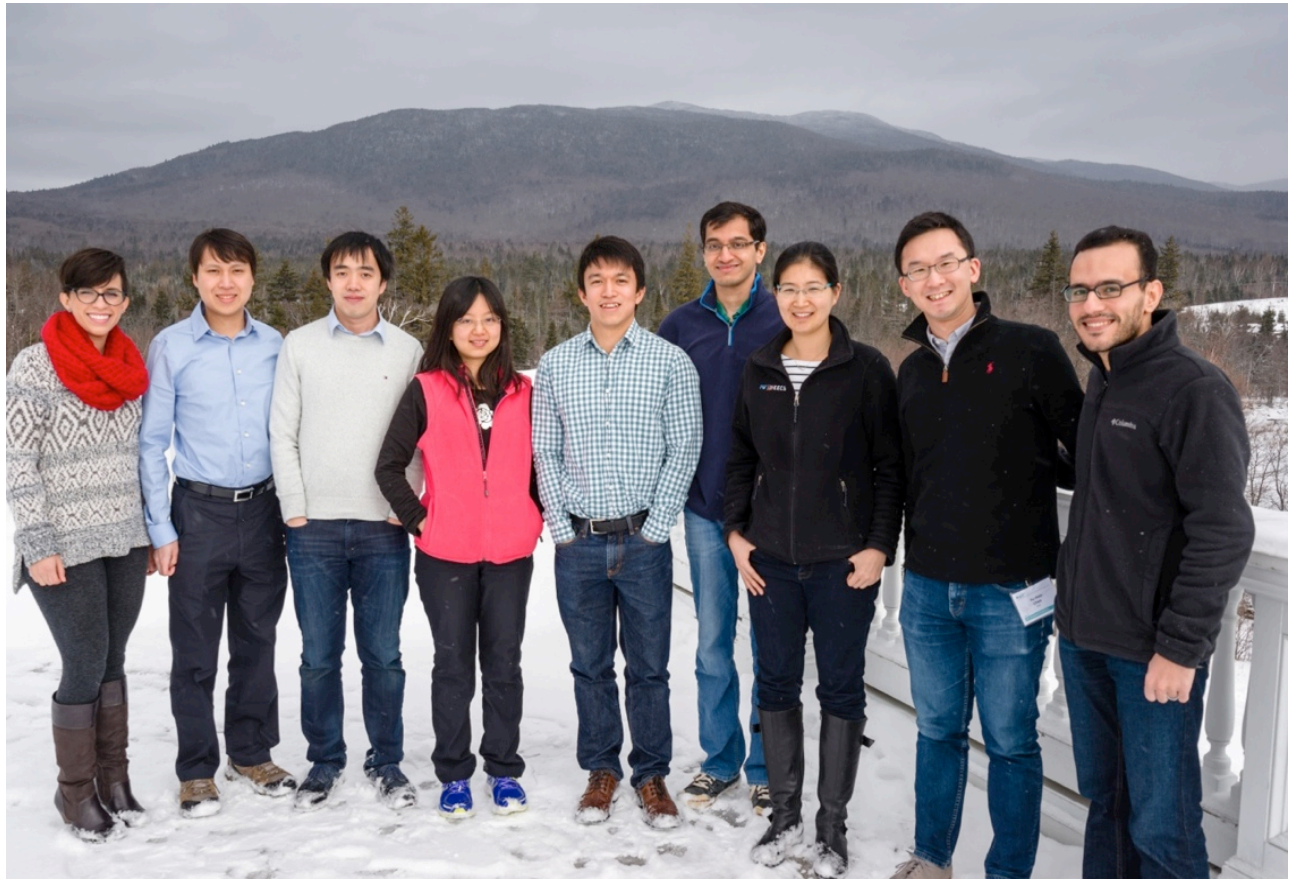
More info about **Eyeriss** and **Tutorial on DNN Architectures** at
http://eyeriss.mit.edu

MIT Professional Education Course on **"Designing Efficient Deep Learning Systems"** *March 28 – 29, 2018 in Mountain View, CA*
http://professional-education.mit.edu/deeplearning

**For updates** Follow @eems_mit

http://mailman.mit.edu/mailman/listinfo/eems-news

# Acknowledgements



Research conducted in the **MIT Energy-Efficient Multimedia Systems Group** would not be possible without the support of the following organizations:

AFOSR AIR FORCE OFFICE OF SCIENTIFIC RESEARCH, NSF, DARPA, SRC, 3M, ANALOG DEVICES, BROADCOM, Google, intel, NVIDIA, QUALCOMM, TEXAS INSTRUMENTS, tsmc