Energy-Efficient Hardware for Embedded Vision and Deep Convolutional Neural Networks

Vivienne Sze

Massachusetts Institute of Technology

In collaboration with Yu-Hsin Chen, Joel Emer, Tushar Krishna, Amr Suleiman, Zhengdong Zhang

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

Video is the Biggest Big Data

Over 70% of today's Internet traffic is video
Over 300 hours of video uploaded to YouTube **every minute**
Over 500 million hours of video surveillance collected **every day**

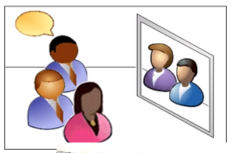
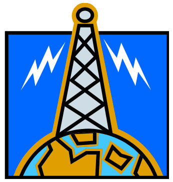
Energy limited due to battery capacity

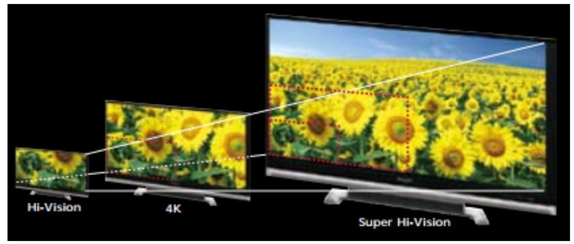
Power limited due to heat dissipation

Need energy-efficient pixel processing!

Energy-Efficient Multimedia Systems Group

Next-Generation Video Coding (Compress Pixels)

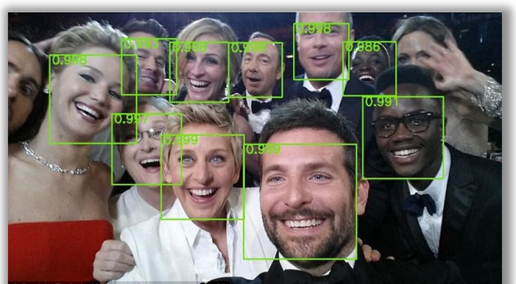





Ultra-HD


Goal: Increase coding efficiency, speed and energy-efficiency

Energy-Efficient Computer Vision & Deep Learning (Understand Pixels)



Recognition

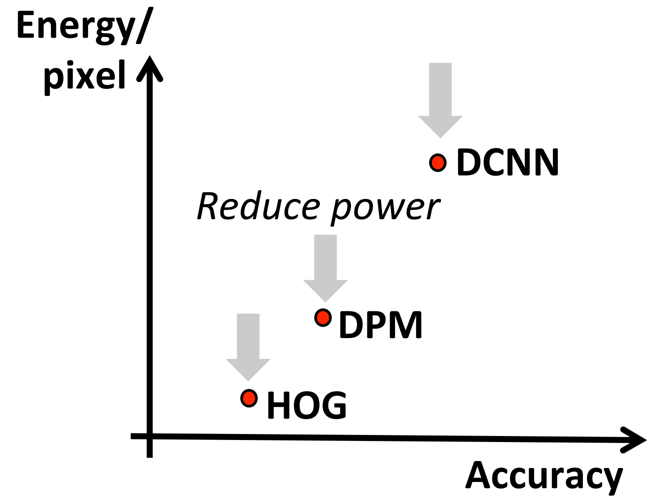

Self-Driving Cars


AI

Goal: Make computer vision as ubiquitous as video coding

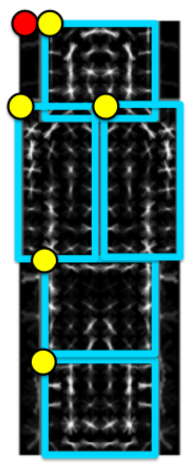
Features for Object Detection/Classification

- **Hand-crafted features**
  - Histogram of Oriented Gradients (HOG)
  - Deformable Parts Model (DPM)
- **Trained features (using machine learning)**
  - Deep Convolutional Neural Nets (DCNN)



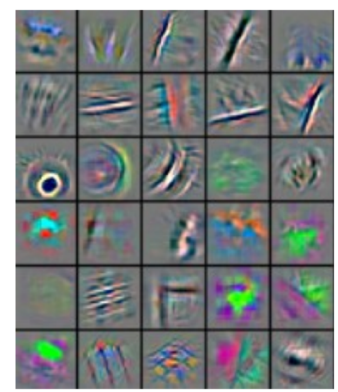
HOG
Rigid Template
based on edges

[Dalal, CVPR 2005]
Cited by 14500



DPM
Flexible Template
based on edges

[Felzenszwalb, PAMI 2010]
Cited by 4063



DCNN
High level
Abstraction

[Krizhevsky, NIPS 2012]
Cited by 4843

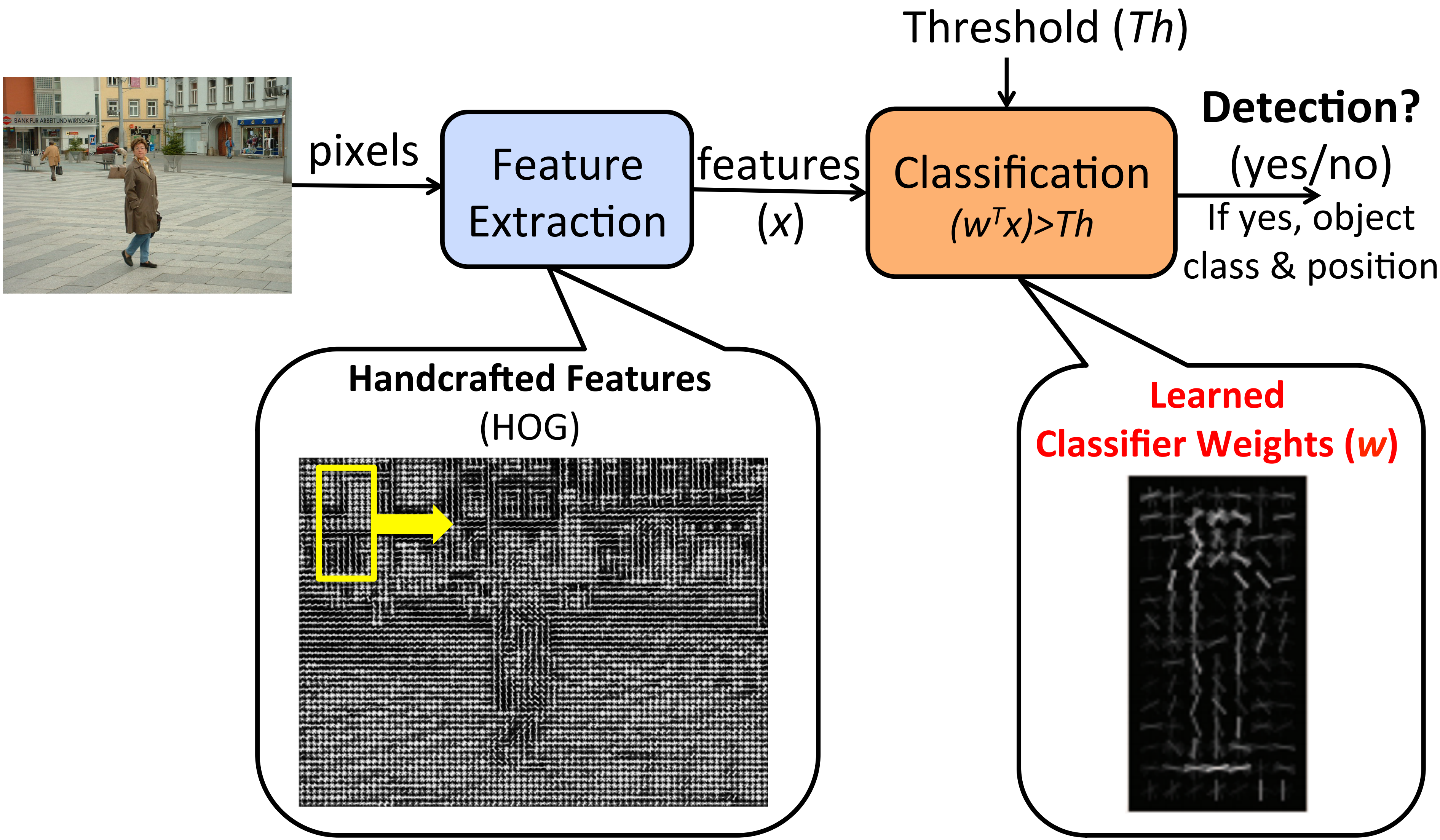
Energy-Efficient Approaches

- **Joint algorithm and hardware design**
  – Use algorithm to make data sparse; hardware to exploit it
- **Minimize data movement**
  – Maximize data reuse and leverage compression
- **Balance flexibility and energy-efficiency**
  – Configurable hardware for different applications

HOG+SVM Accelerator

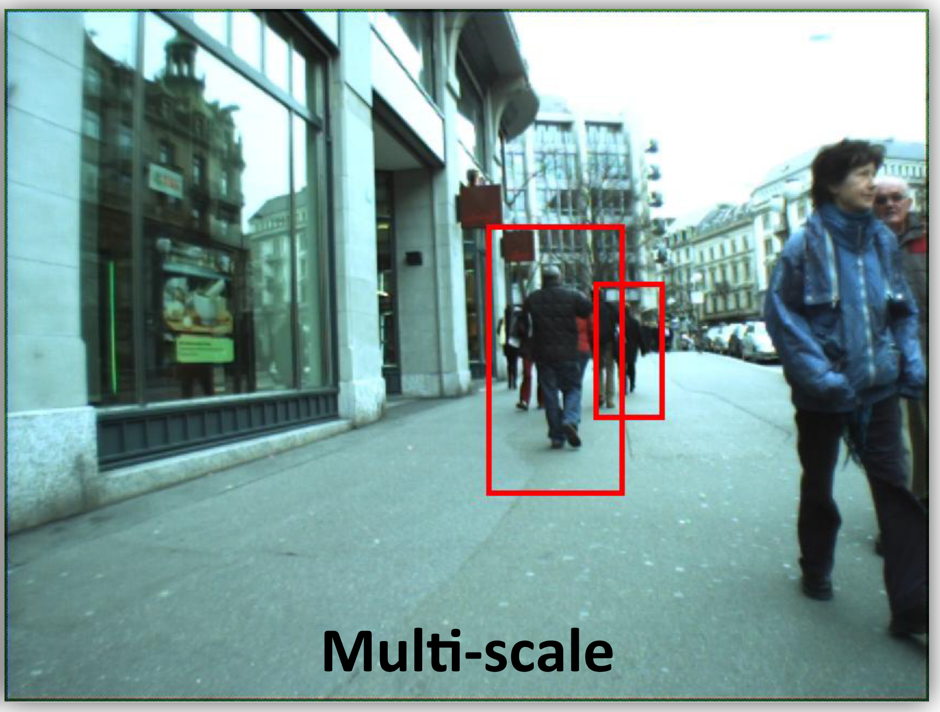
Amr Suleiman, Vivienne Sze, Journal of Signal Processing Systems 2015 [paper]

Object Detection Pipeline



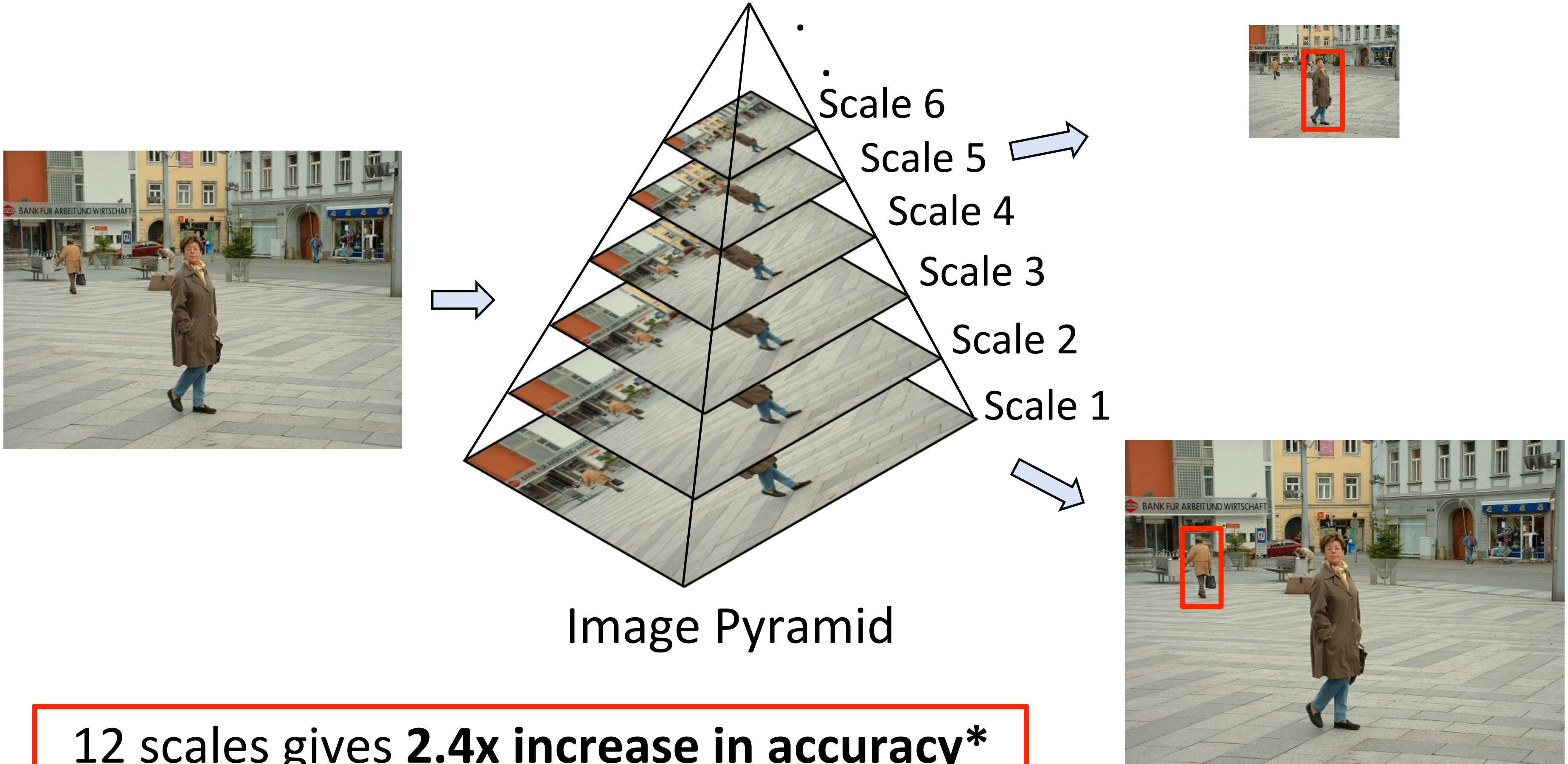
[Dalal et al., CVPR 2005]

Multi-Scale Object Detection



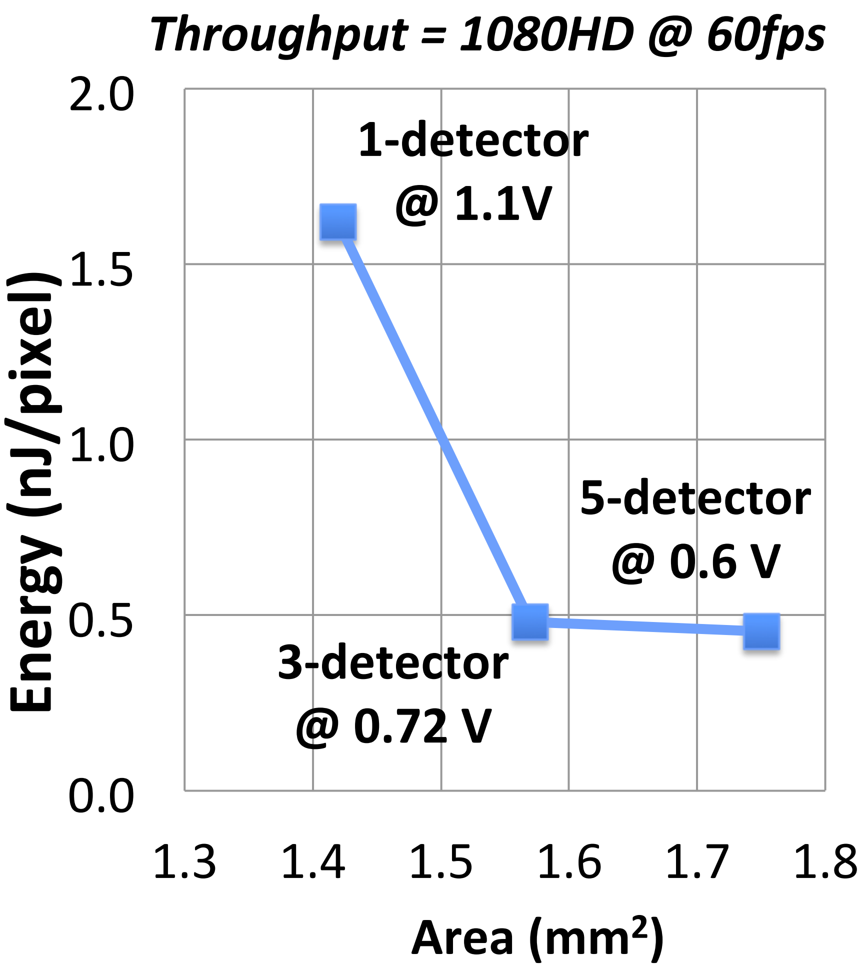
Detecting Objects with Different Sizes

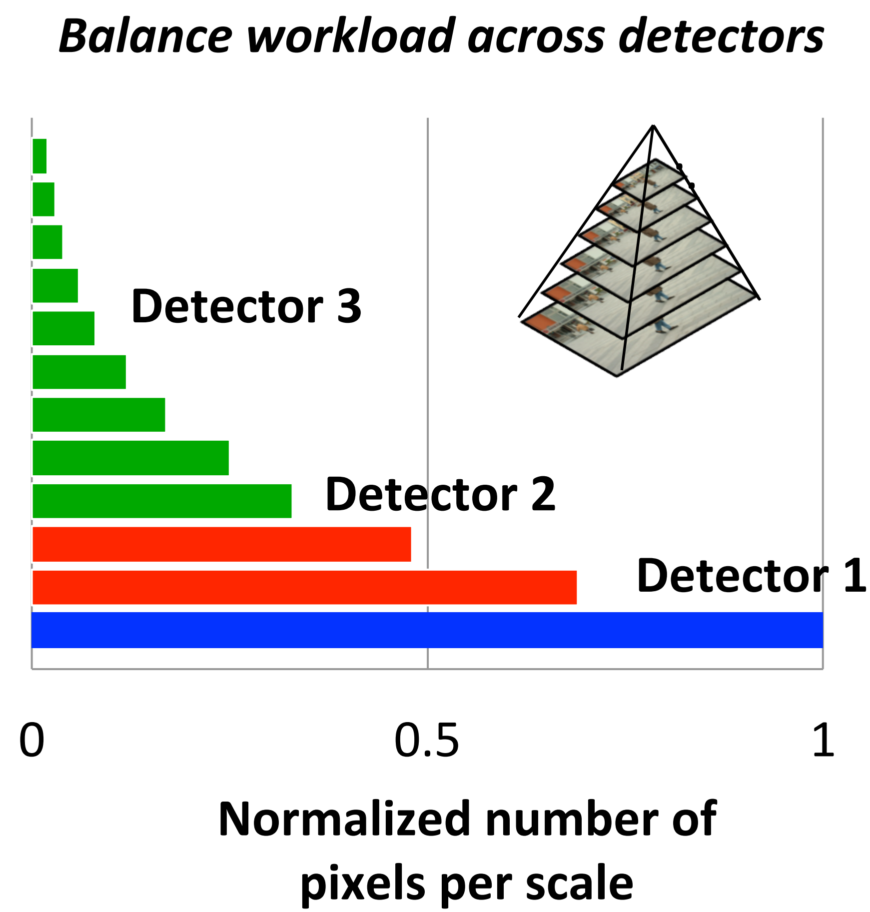
- Process different resolutions of the same frame.



12 scales gives **2.4x increase in accuracy*** at the cost of **3.2x increase in processing**

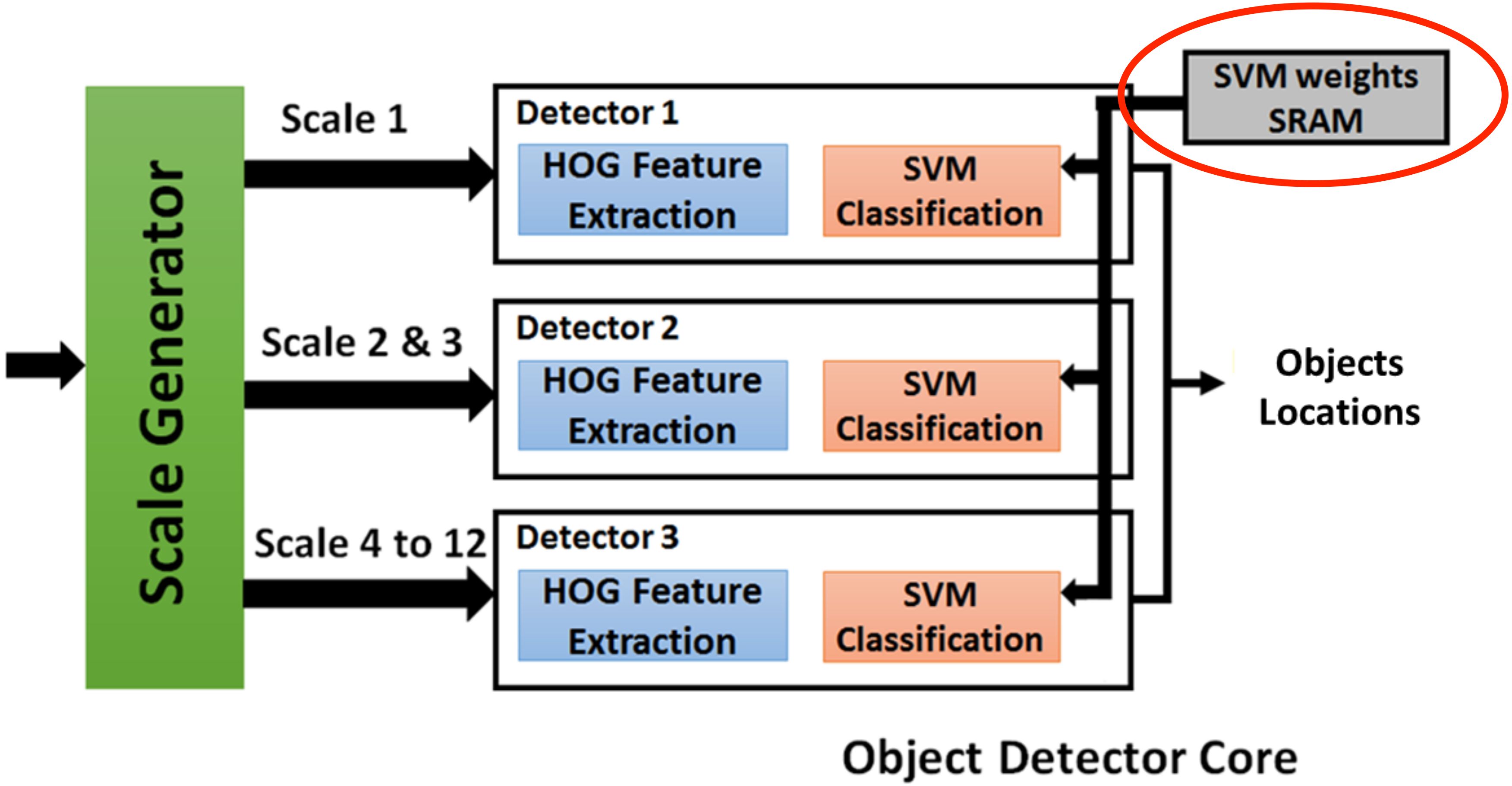
Parallel Detectors and Voltage Scaling





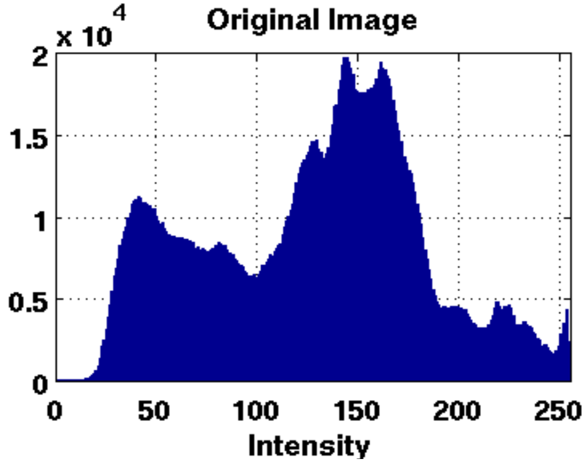
Use **three** parallel detectors at 0.72V for a **3.4x energy reduction**

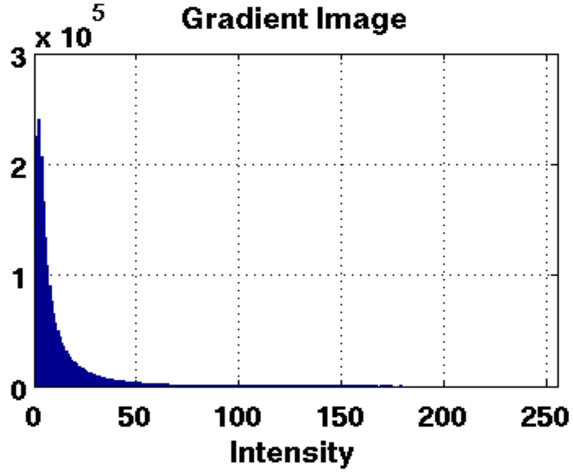
Share Reads Across Parallel Detectors



Synchronize detectors to share SVM weight memory
(**20%** reduction in power)

Image Pre-Processing





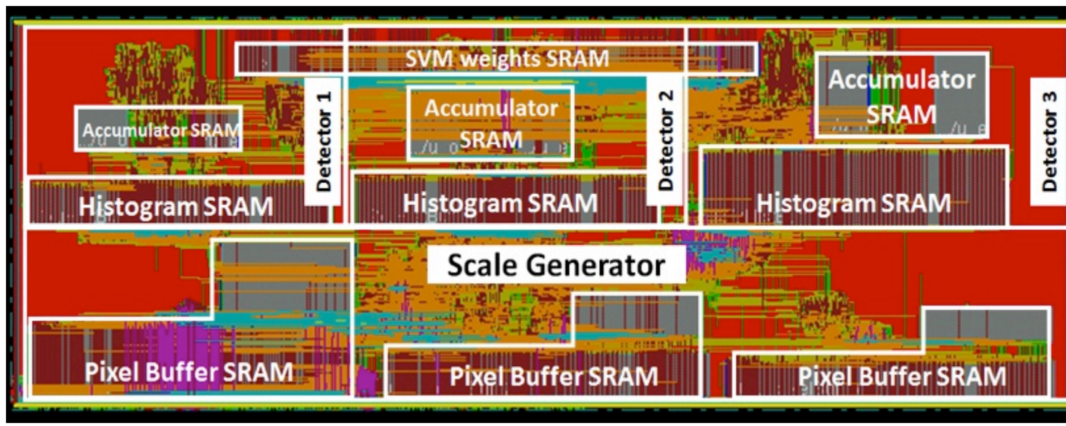
- Gradient pre-processing reduces cost of image scale generation
  - Reduce memory size by 2.7x
  - Reduce power consumption by 43%
  - Reduce detection accuracy by 2%

Real-Time HOG Detector Summary

- An energy-efficient object detector is implemented delivering **real-time processing of 1920x1080 at 60 fps**
- Multi-scale support for **2.4x higher detection accuracy**
- Parallel detectors, voltage scaling and image pre-processing for **4.5x energy reduction**

Area	2.8 mm^2
Max Frequency	270 MHz
Scales/frame	12
Gate count	490 kgates
On-chip SRAM	0.538 Mbit

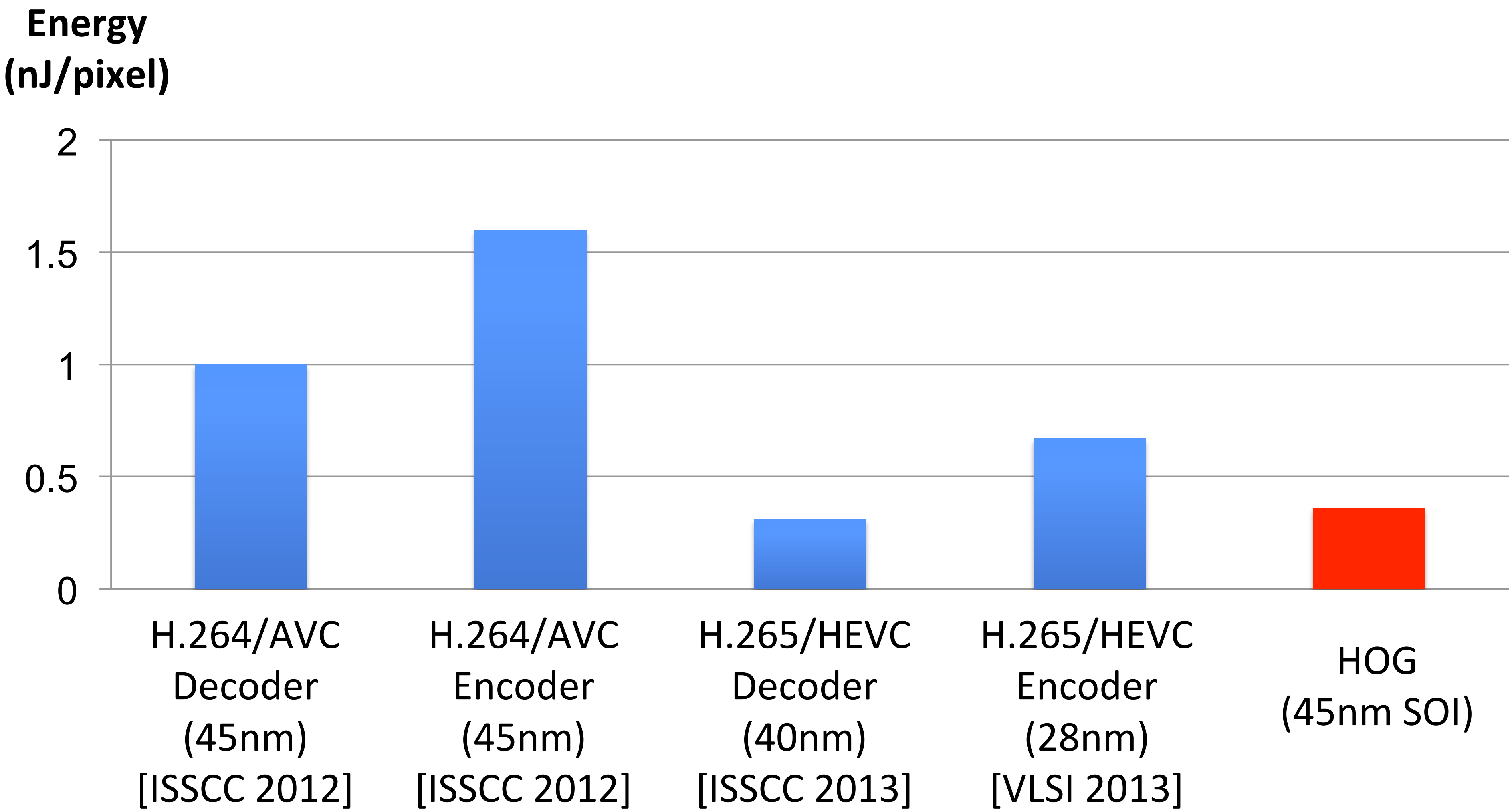
Post-layout simulations



45nm SOI process

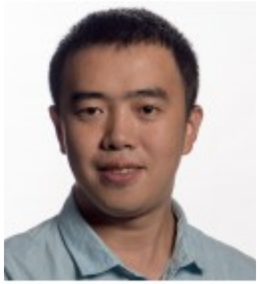
Real-time multi-scale object detection at 45mW **(0.36 nJ/pixel)**

Comparison with Video Coding



Deformable Parts Model Hardware Accelerator

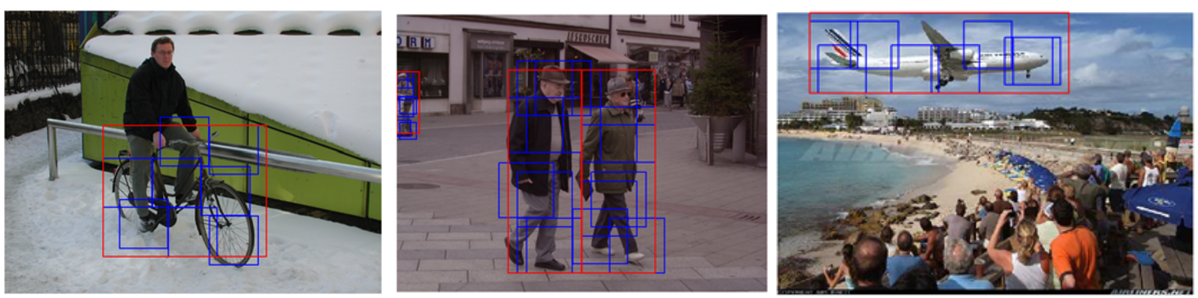
Amr Suleiman, Zhendong Zhang, Vivienne Sze, VLSI 2016 [paper]

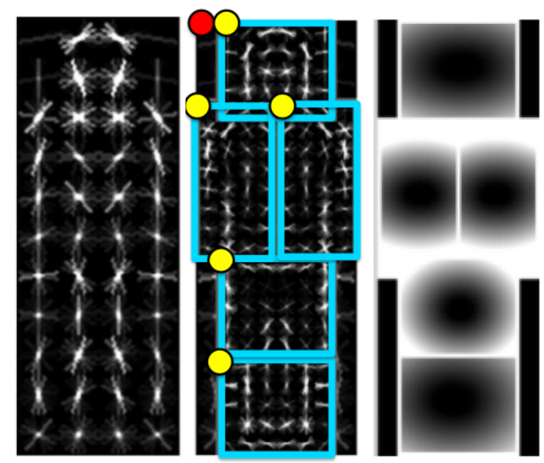


Deformable Parts Models (DPM)

- Define HOG templates for an object (root) and its parts (at 2x root resolution) with relative locations (anchors)
- Allow anchors to move with deformation penalty

$$DPMScore = RootScore + \overbrace{\sum_{i=1}^{8} \max_{dx,dy}\left(PartScore_i(dx,dy) - DeformCost_i(dx,dy)\right)}^{\textbf{Impact of parts and deformation}}$$





~2x higher accuracy than rigid template (HOG)
High classification cost!

[Felzenszwalb et al., PAMI 2010]

Object Detection Pipeline

pixels

Feature Extraction

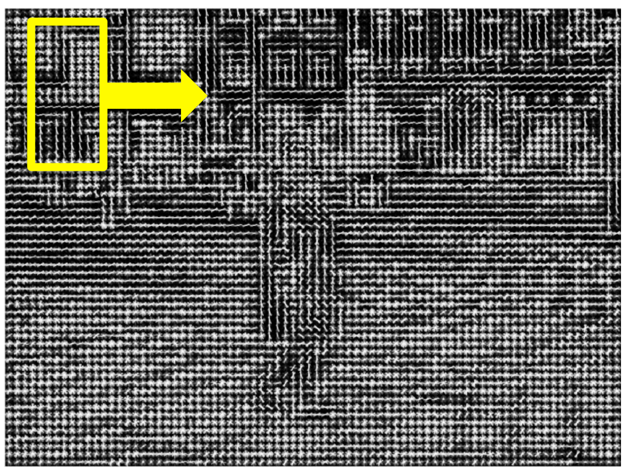
features (x)

Threshold (Th)

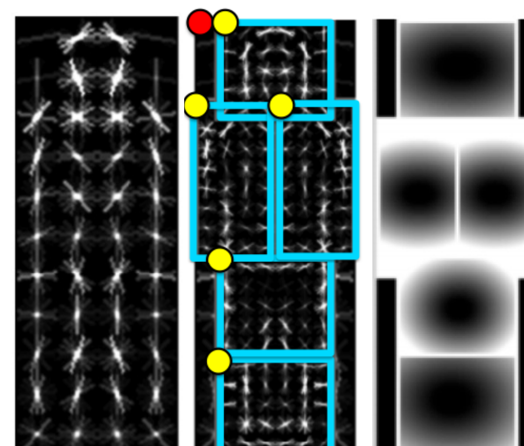
Classification

Detection?
(yes/no)
If yes, object class & position

Handcrafted Features
(HOG)



Learned Classifier Weights (w)



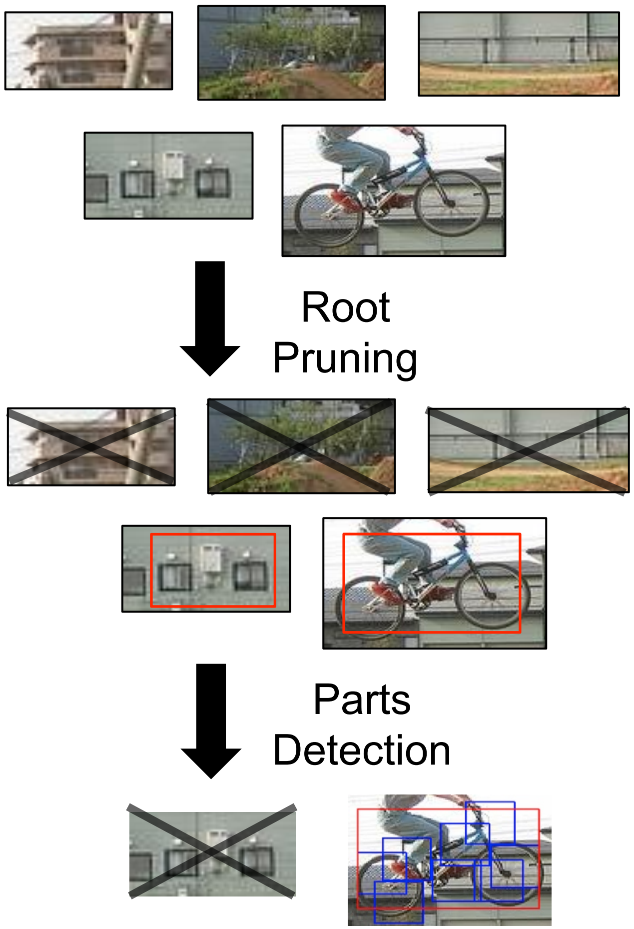
Root *Parts* *Deformation*

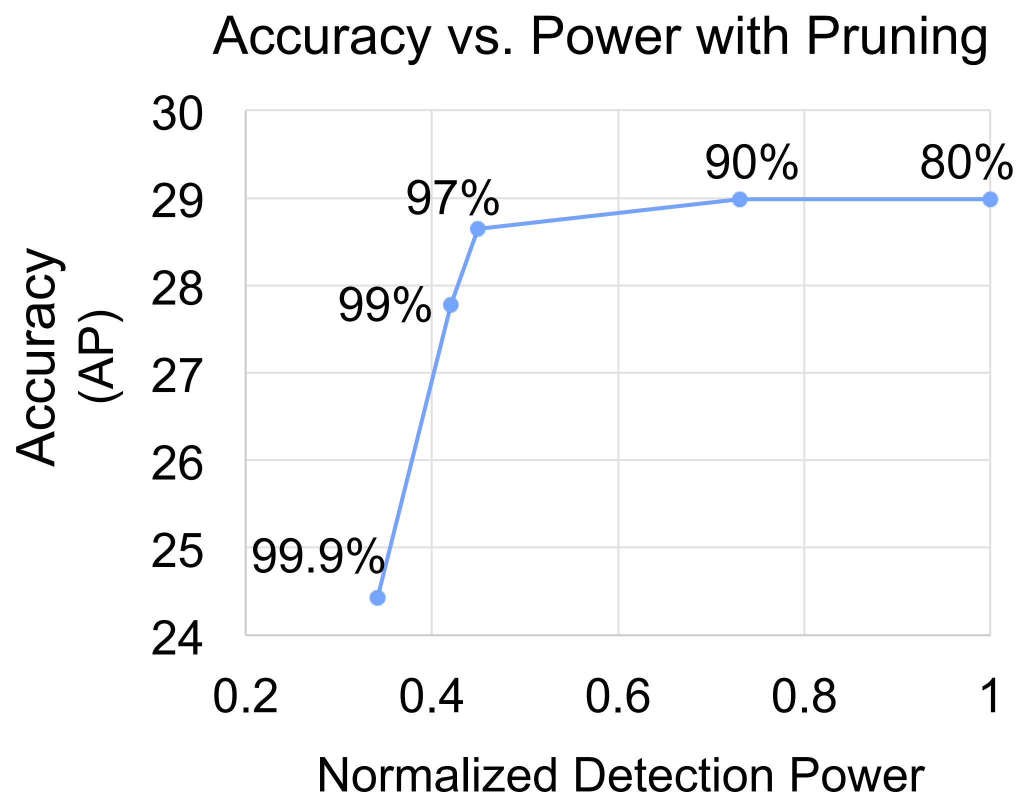
Flexible vs. Rigid Template Complexity

- **DPM classification with 8 parts requires >10x more operations than root only classification**
  - Due to parts template, parts resolution, deformation computation
- **Approaches to reducing complexity**
  - **Root Pruning:** Reduce number of part classifications based on root
  - **Basis Projection:** Reduce amount of computation per classification

Low Power Parts Classification

Prune >80% roots to reduce parts classification

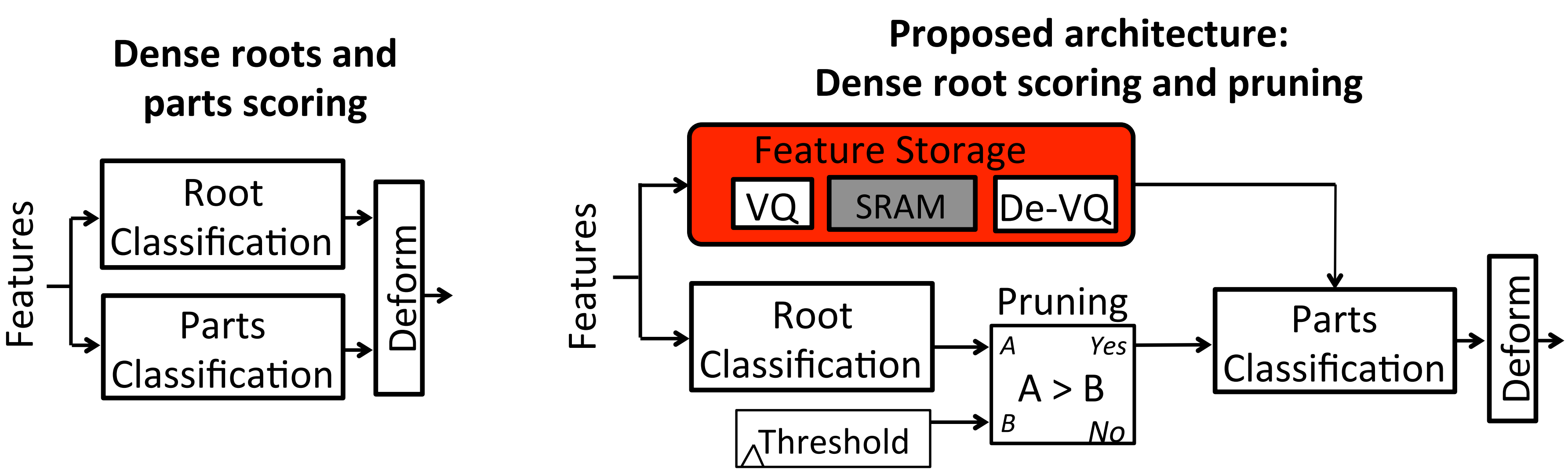




Measured on VOC 2007 Dataset

Low Power Parts Classification

- Store features for reuse by parts to avoid re-computation
- Use Vector Quantization to reduce feature storage cost
  - **16x reduction in memory size** [520kB vs. 32kB]
  - **7.6x reduction in area** [520kB vs. VQ + 32kB + De-VQ]



Low Power Roots and Parts Classification

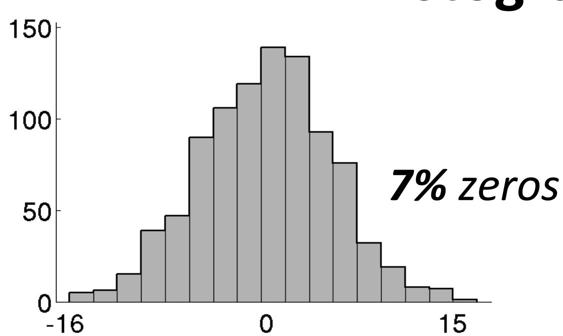
Reduce the number of multiplications by projecting onto a basis that increases sparsity (>1.8x power reduction)

Basis Projection Equation

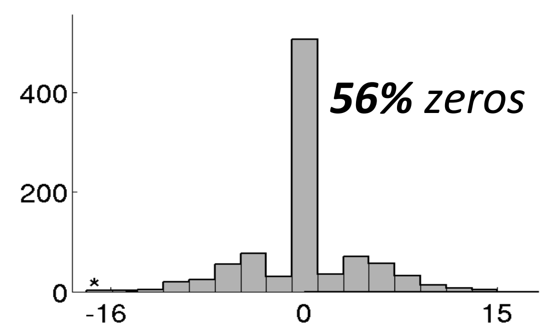
$$\left\langle H, W \right\rangle = \left\langle H, \sum_d S_d \alpha_d \right\rangle = \sum_d \left\langle H, S_d \right\rangle \alpha_d = \sum_d P_d \alpha_d$$

Features Weights Basis Projected Features Projected Weights

Histogram of Weights



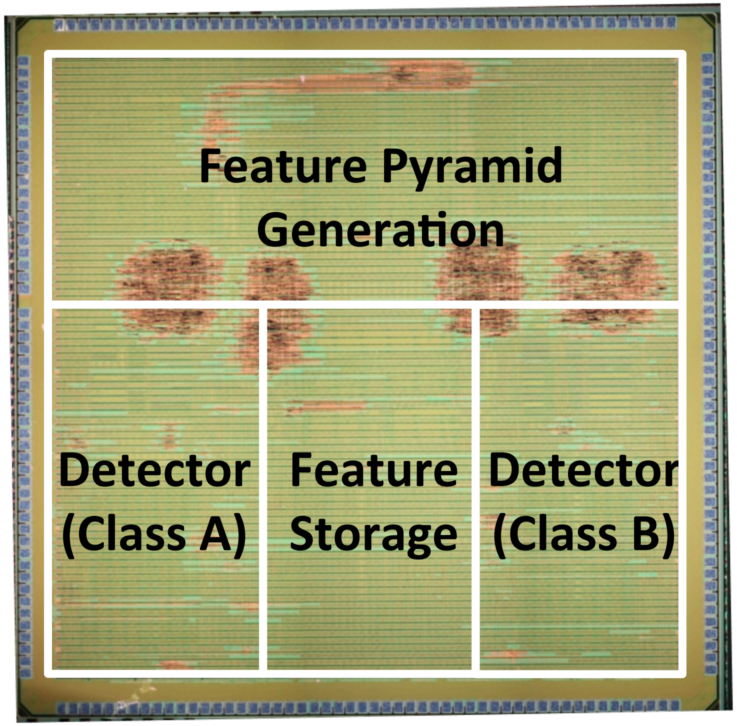
Weights (W)

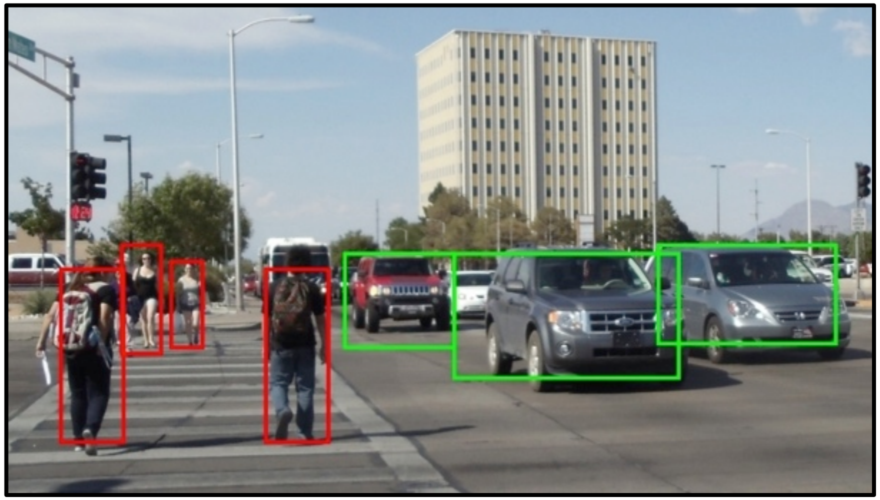


Projected Weights (α_d)

DPM Test Chip

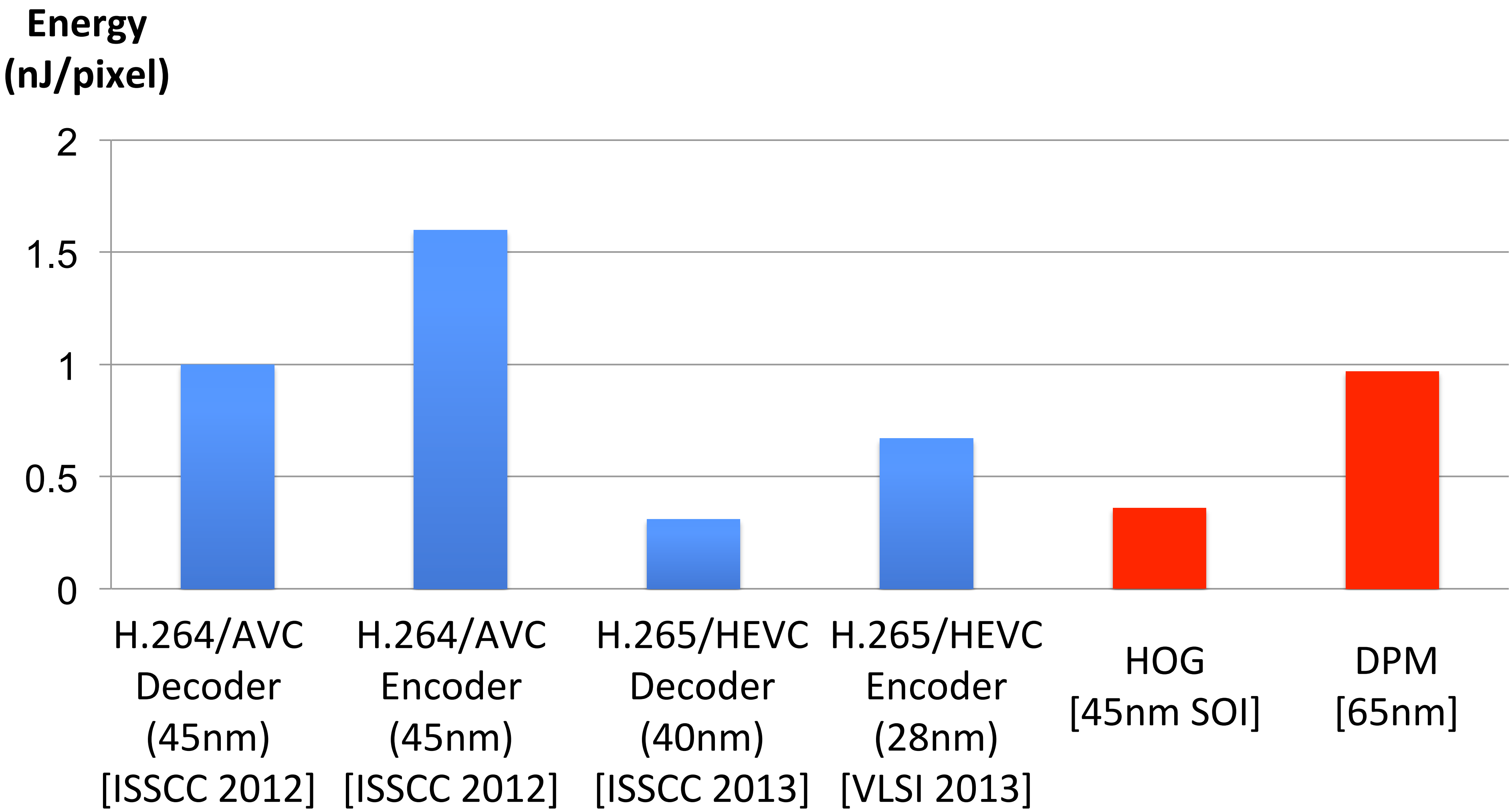
Technology	65nm LP CMOS
Core size	3.5mm x 3.5mm
Logic gates	3283 kgates
SRAM	280 KB
Resolution	1920x1080
Supply	0.77 – 1.11 V
Frequency	62.5 – 125 MHz
Frame rate	30 – 60 fps
Power	58.6 – 216.5 mW
Energy	0.94 – 1.74 nJ/pixel





Overall Tradeoff
5x power reduction,
3.6x memory reduction,
4.8% accuracy reduction

Comparison with Video Coding

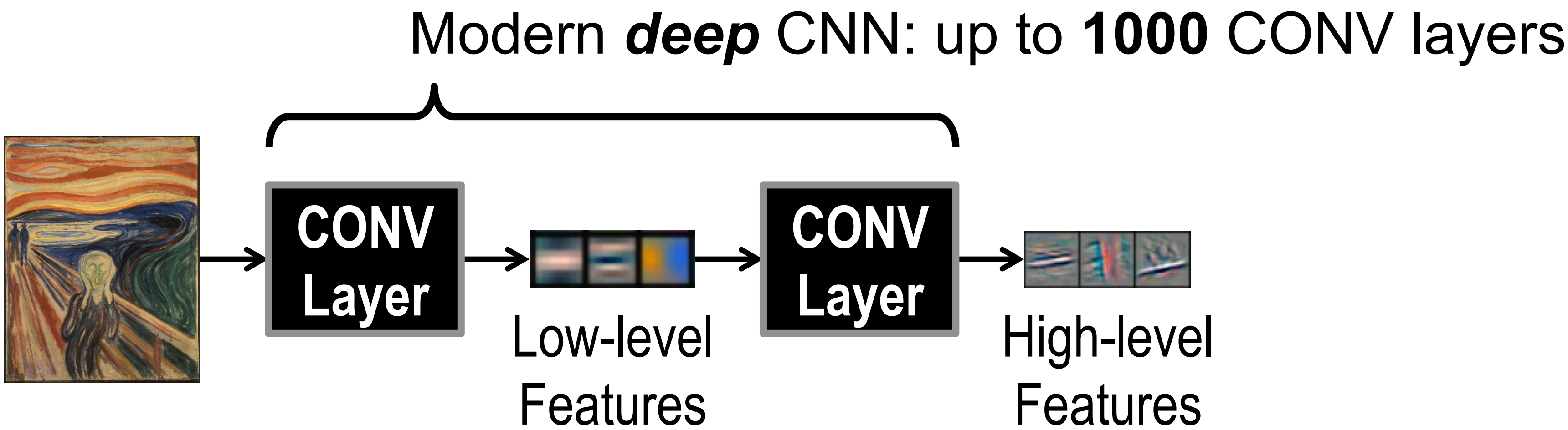


Eyeriss: Energy-Efficient Hardware for DCNNs

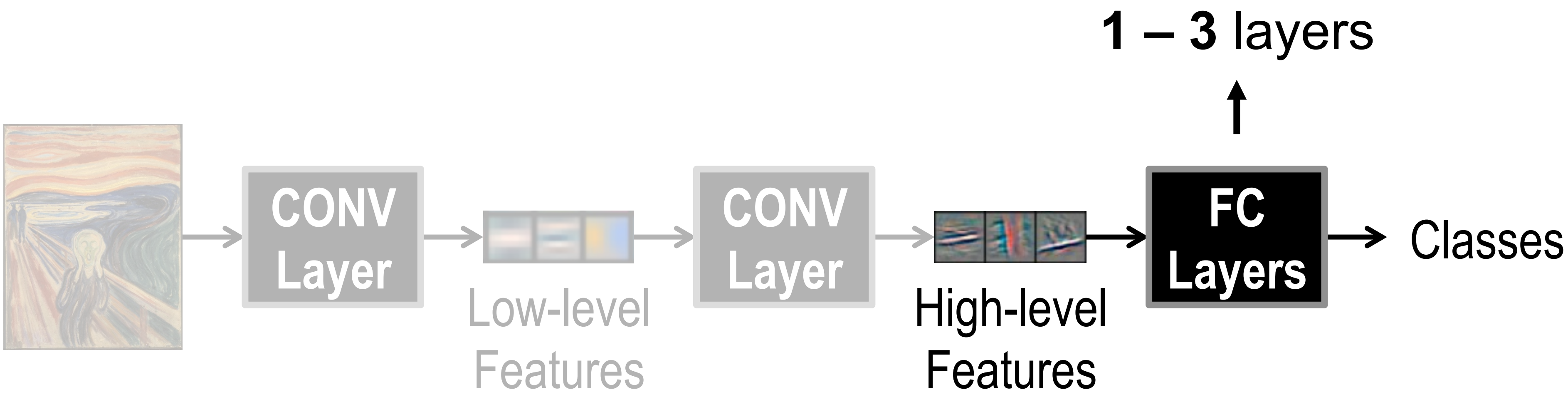
Yu-Hsin Chen, Tushar Krishna, Joel Emer, Vivienne Sze, ISSCC 2016 [paper] / ISCA 2016 [paper]



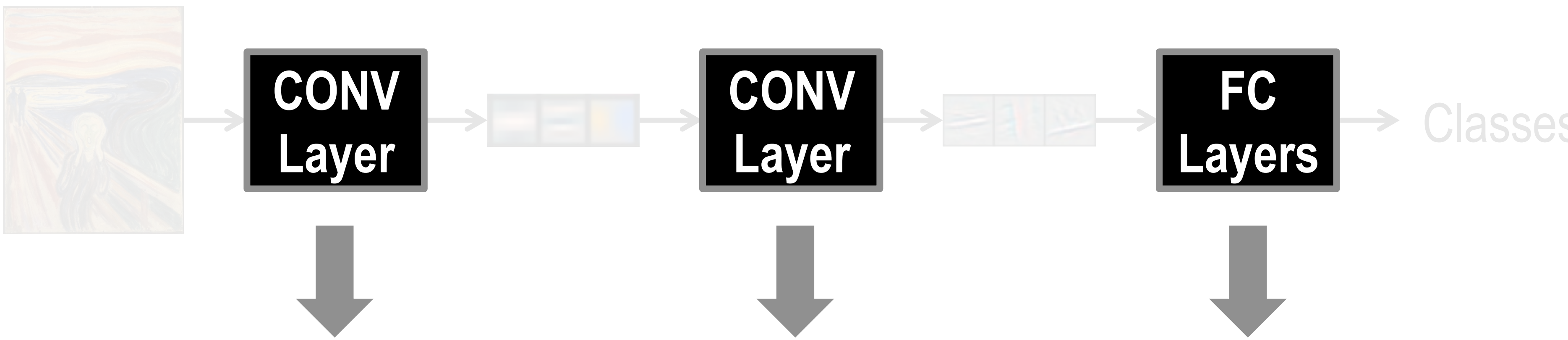

Deep Convolutional Neural Networks

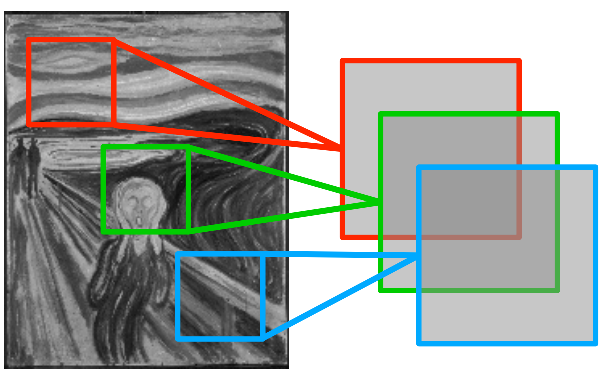


Deep Convolutional Neural Networks



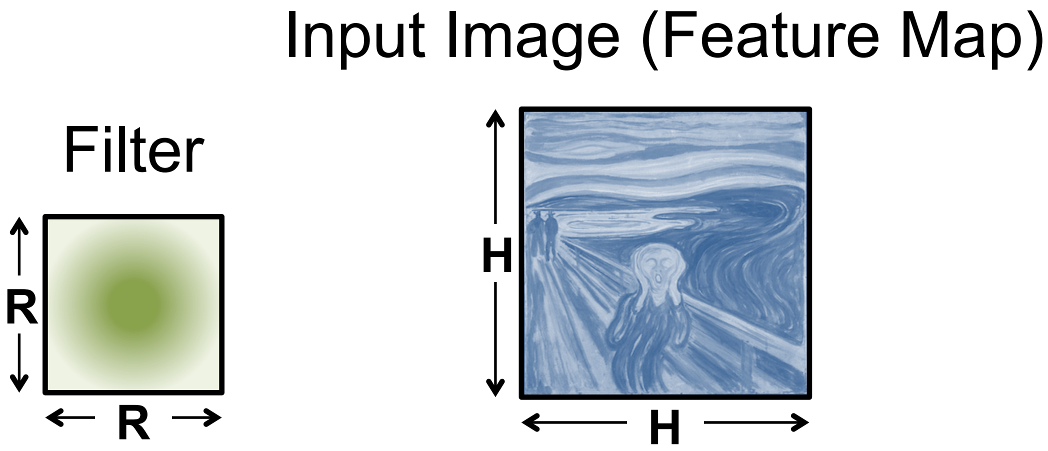
Deep Convolutional Neural Networks



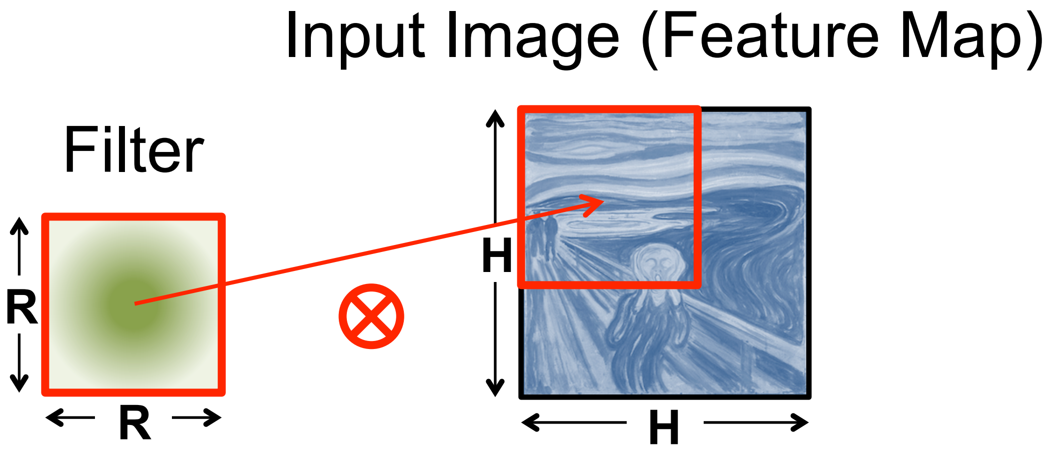


Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

High-Dimensional CNN Convolution

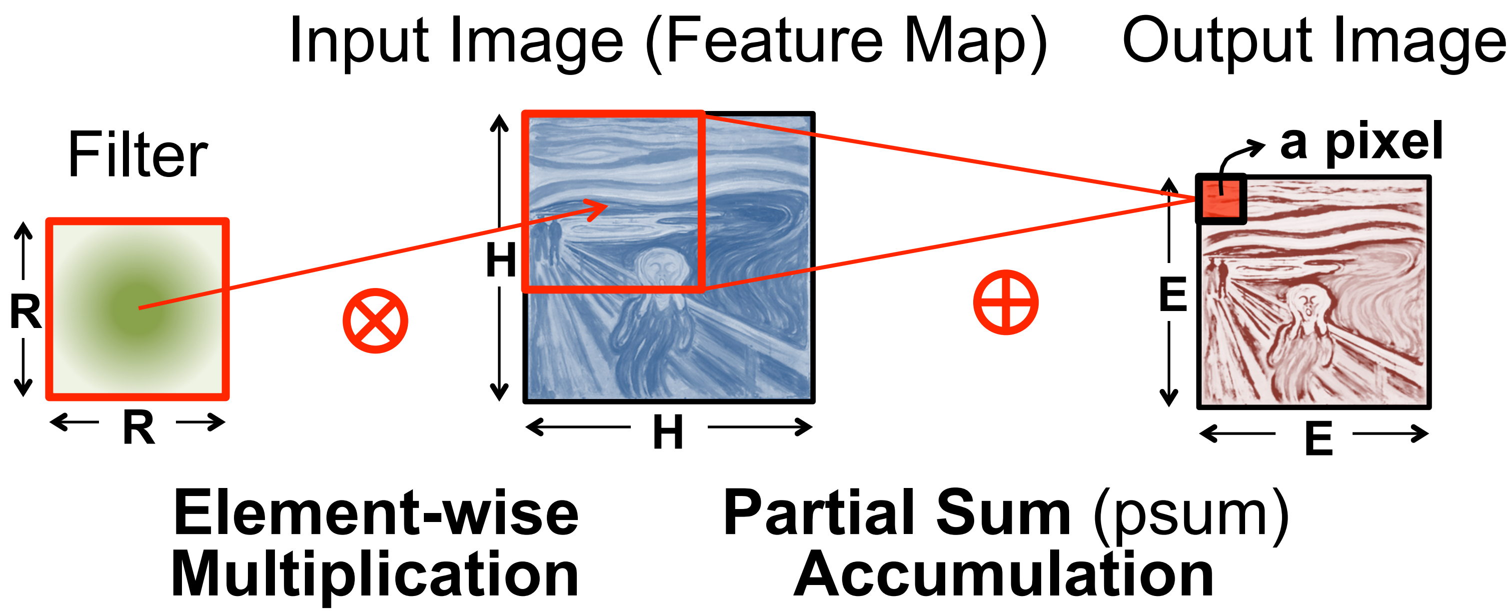


High-Dimensional CNN Convolution

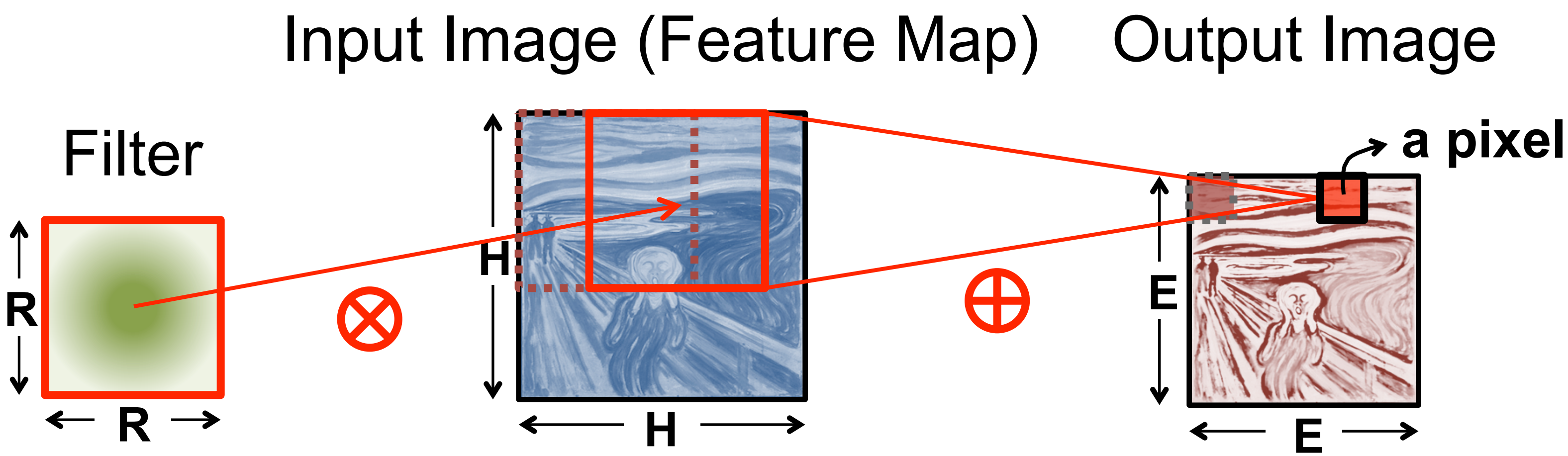


Element-wise Multiplication

High-Dimensional CNN Convolution

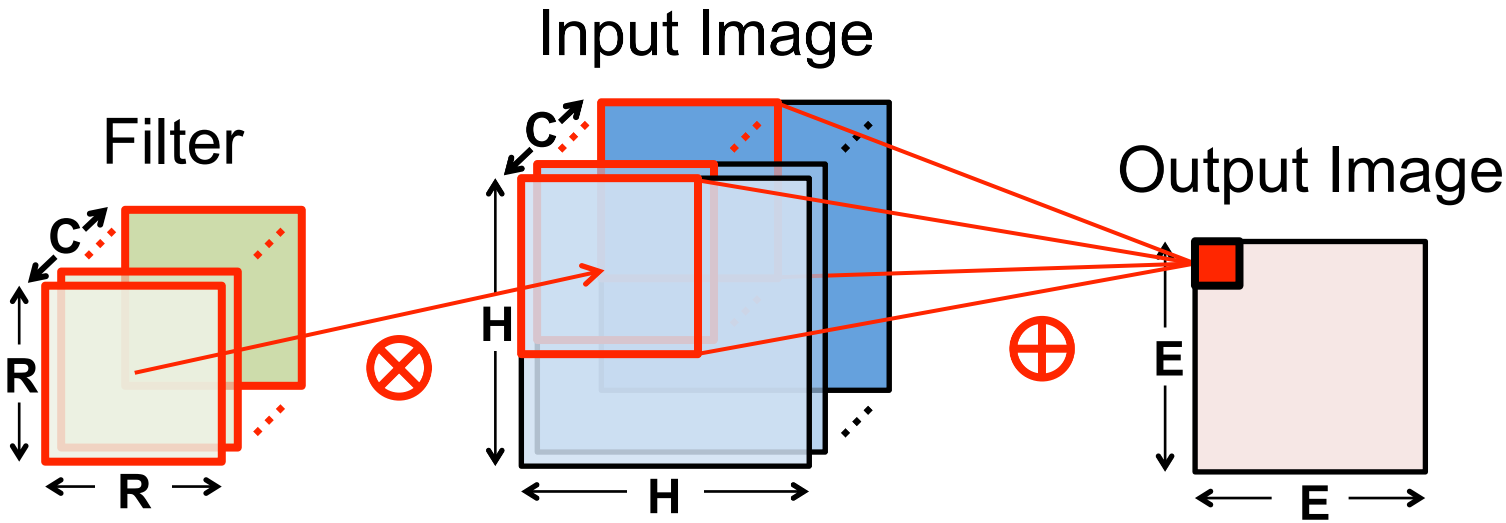


High-Dimensional CNN Convolution



Sliding Window Processing

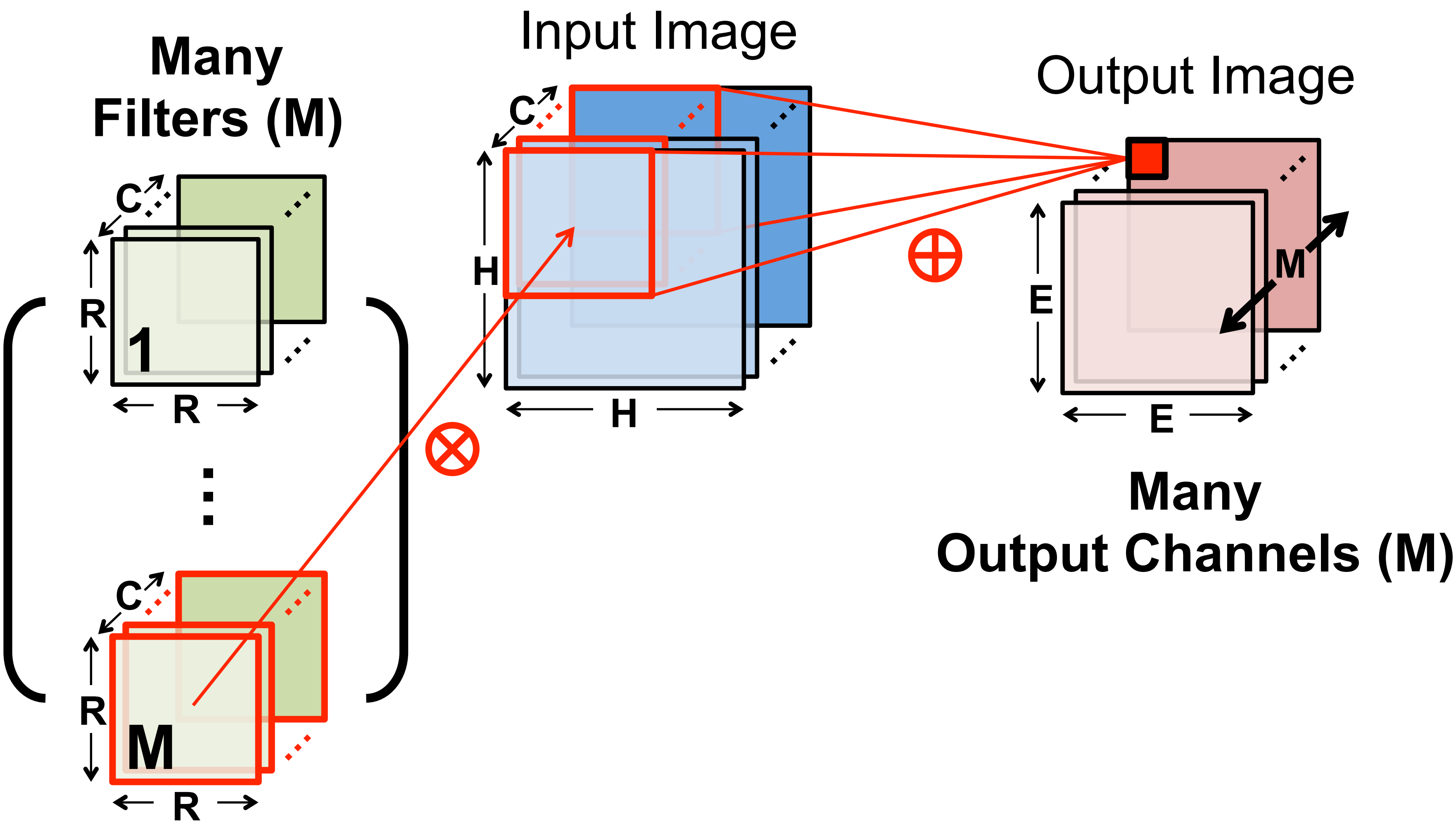
High-Dimensional CNN Convolution



Many Input Channels (C)

AlexNet: 3 – 192 Channels (C)

High-Dimensional CNN Convolution



AlexNet: 96 – 384 Filters (M)

High-Dimensional CNN Convolution

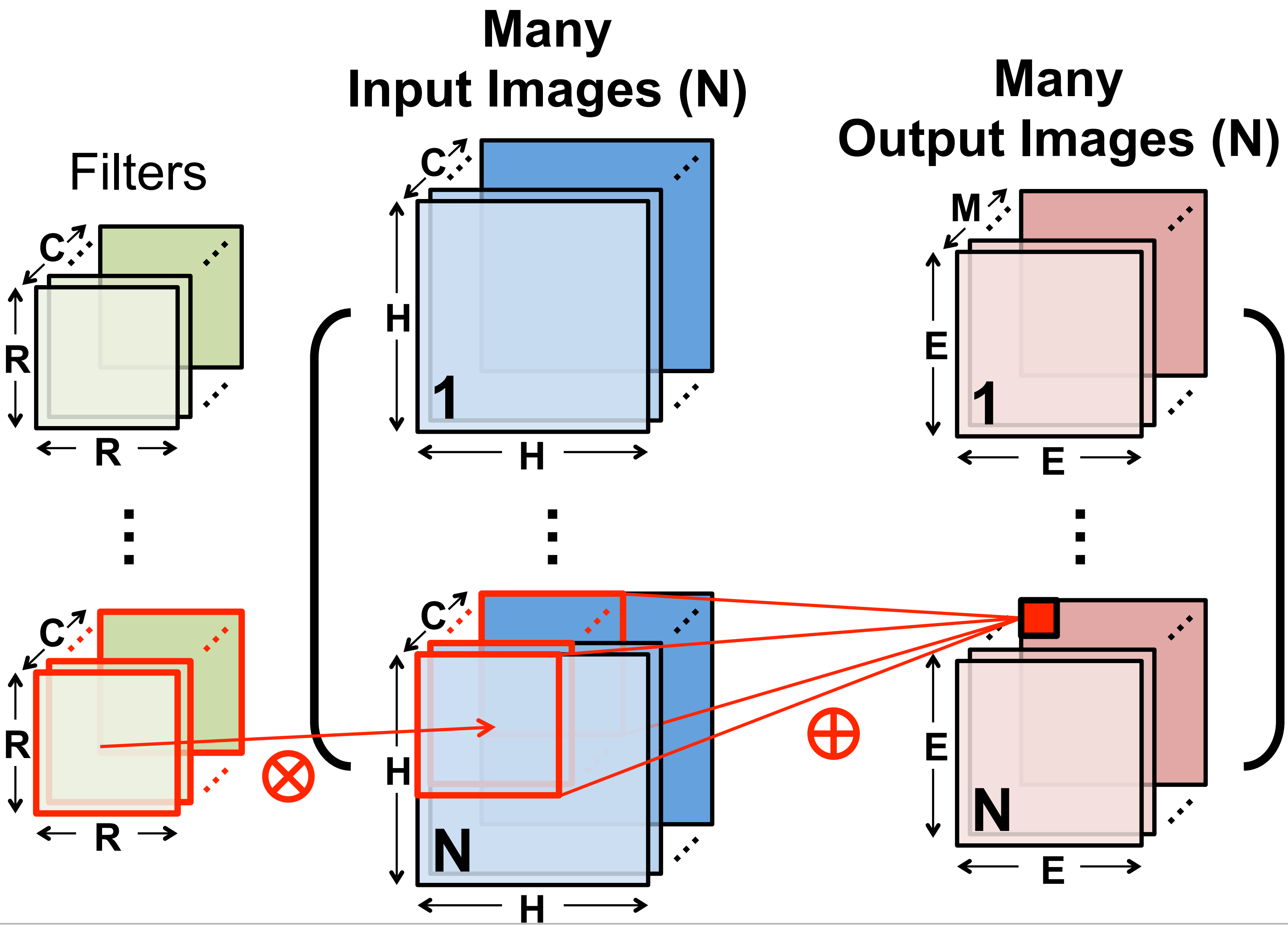


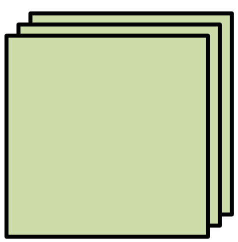
Image batch size: 1 – 256 (N)

Large Sizes with Varying Shapes

AlexNet[1] Convolutional Layer Configurations

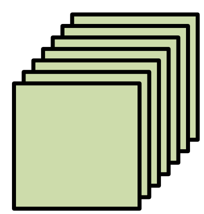
Layer	Filter Size (R)	# Filters (M)	# Channels (C)	Stride
1	11x11	96	3	4
2	5x5	256	48	1
3	3x3	384	256	1
4	3x3	384	192	1
5	3x3	256	192	1

Layer 1



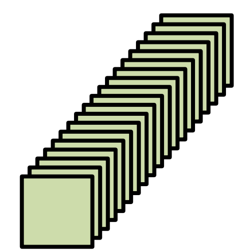
34k Params
105M MACs

Layer 2



307k Params
224M MACs

Layer 3



885k Params
150M MACs

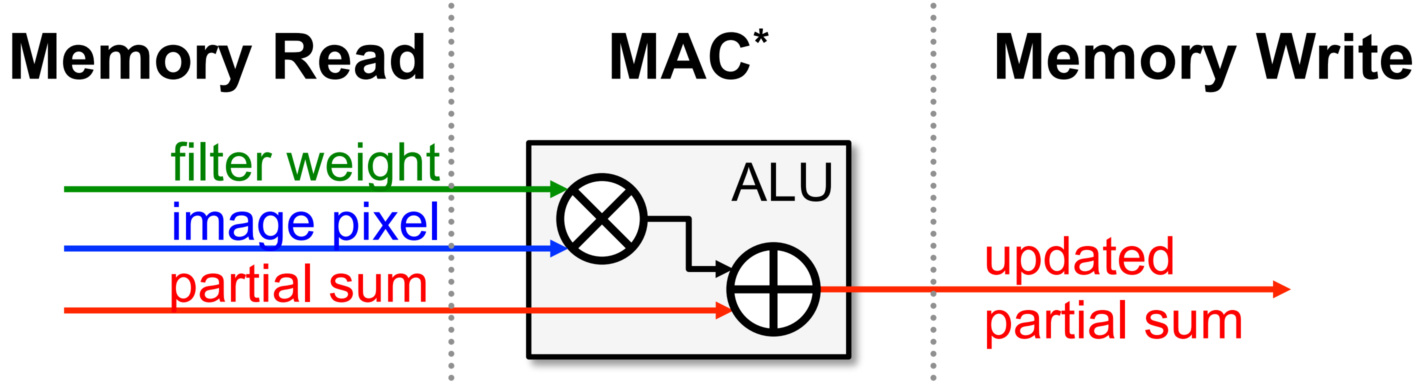
1. [Krizhevsky, NIPS 2012]

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

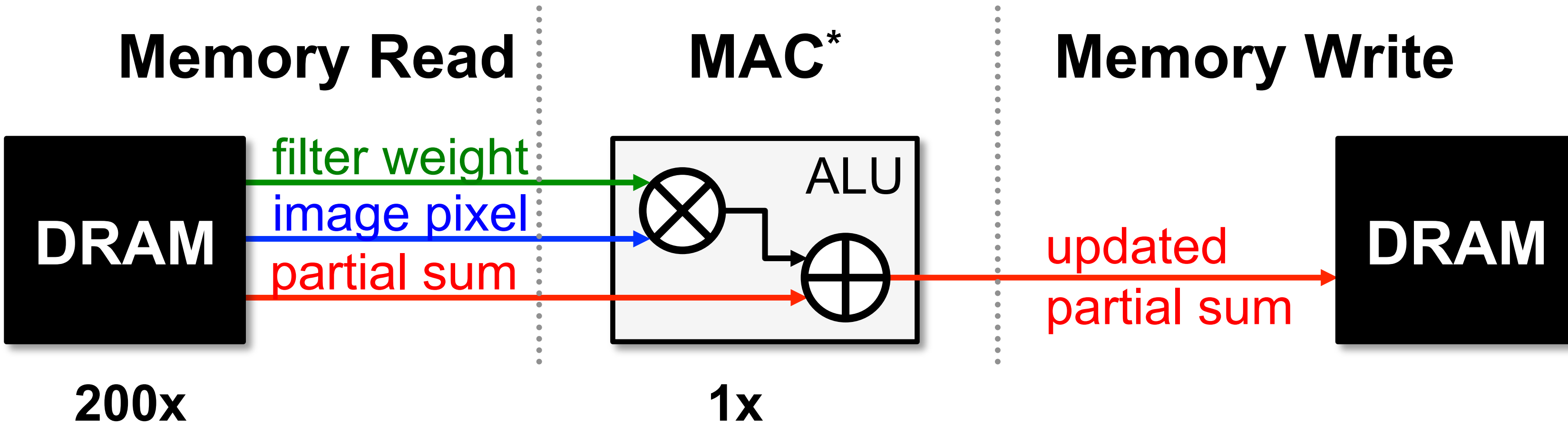


* multiply-and-accumulate

Properties We Can Leverage

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

- Memory Access is the Bottleneck

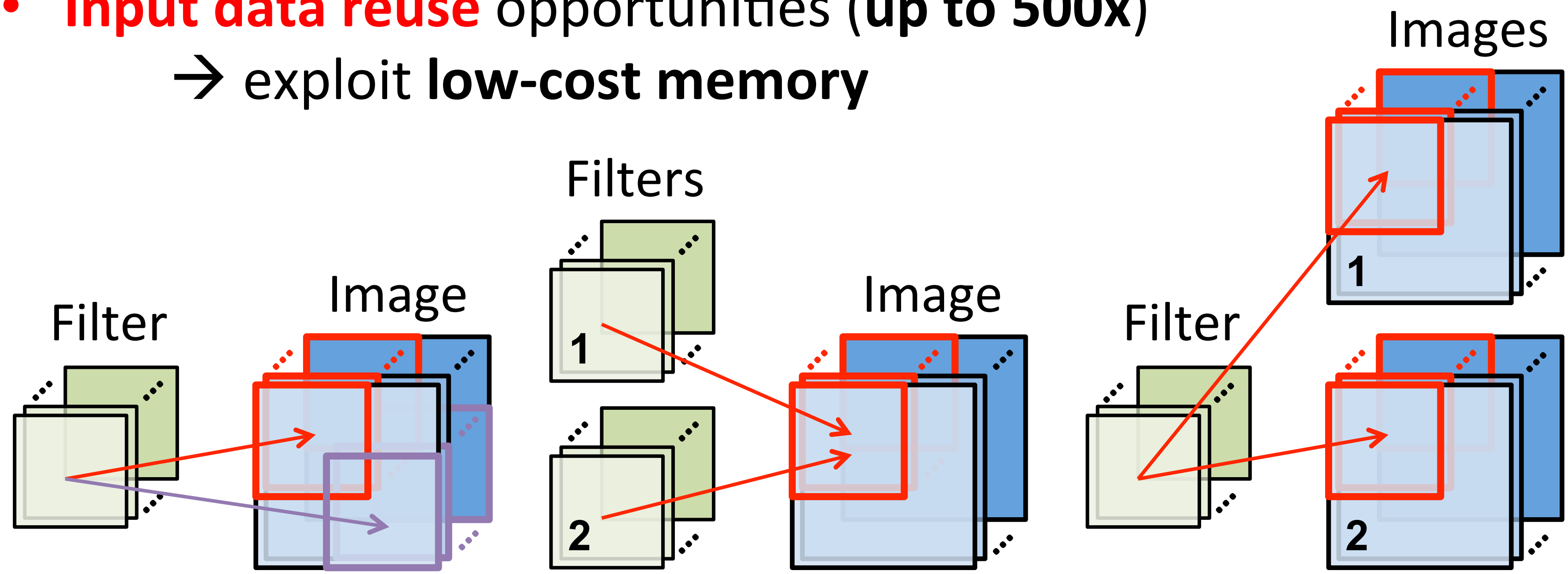


Worst Case: all memory R/W are **DRAM** accesses

- 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**



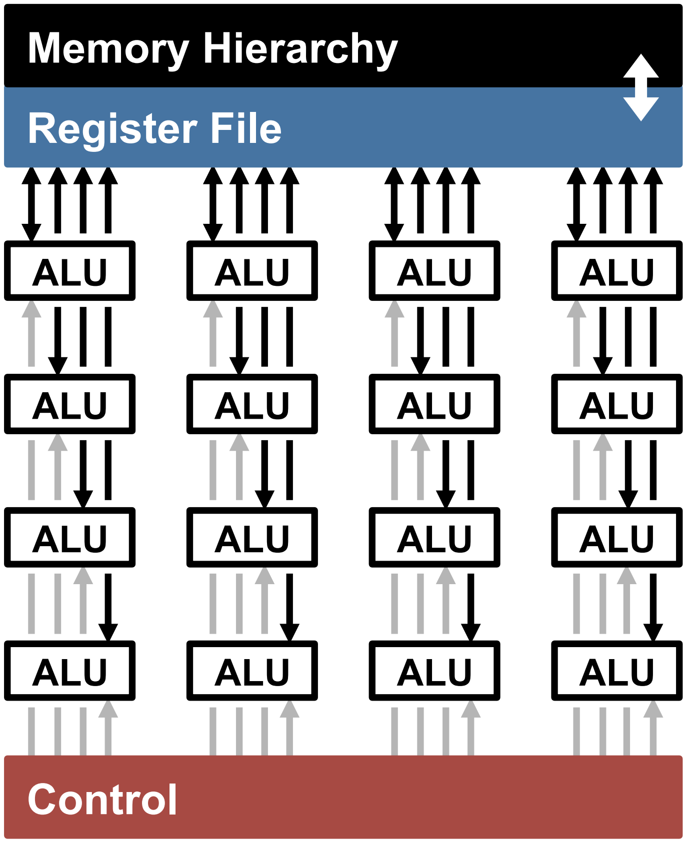
Convolutional Reuse (pixels, weights)

Image Reuse (pixels)

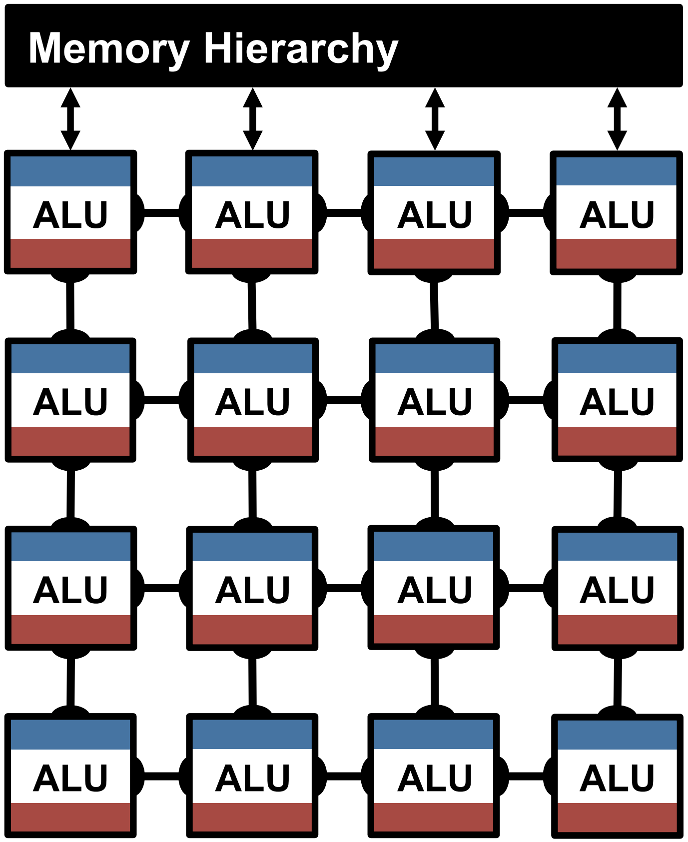
Filter Reuse (weights)

Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)



Spatial Architecture (Dataflow Processing)



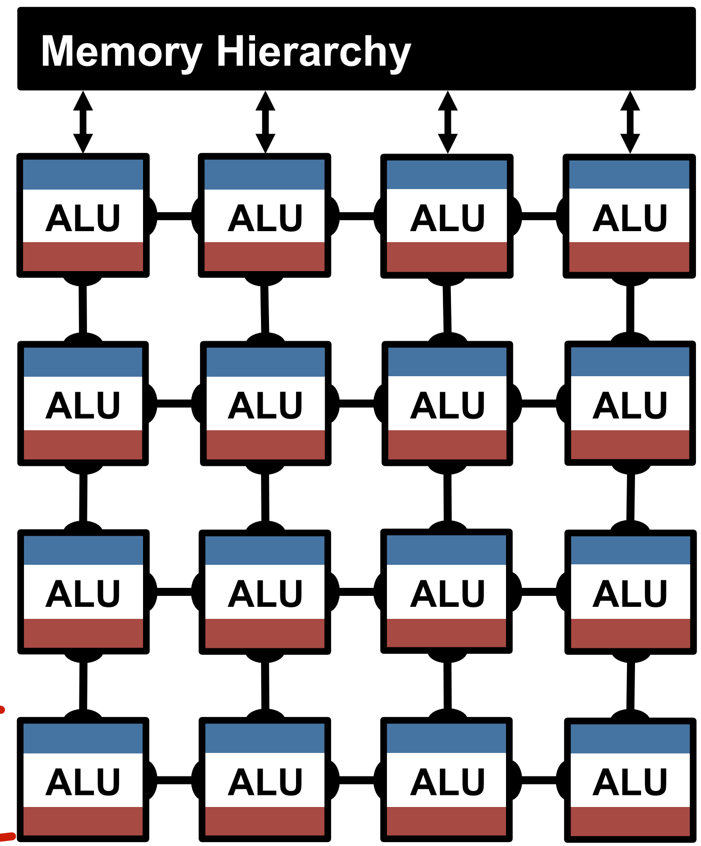
Advantages of Spatial Architecture

Temporal Architecture
(SIMD/SIMT)

Efficient Data Reuse
Distributed local storage (RF)

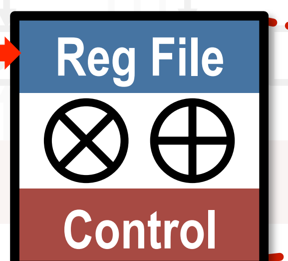
Inter-PE Communication
Sharing among regions of PEs

Spatial Architecture (Dataflow Processing)



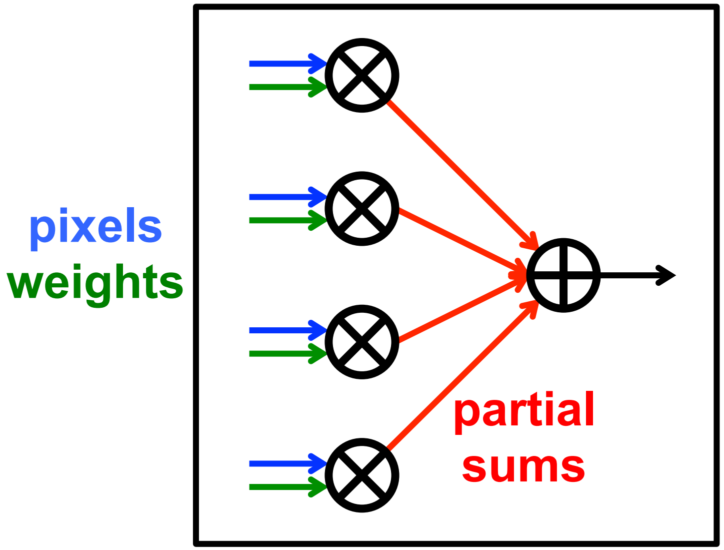
Processing Element (PE)

0.5 – 1.0 kB

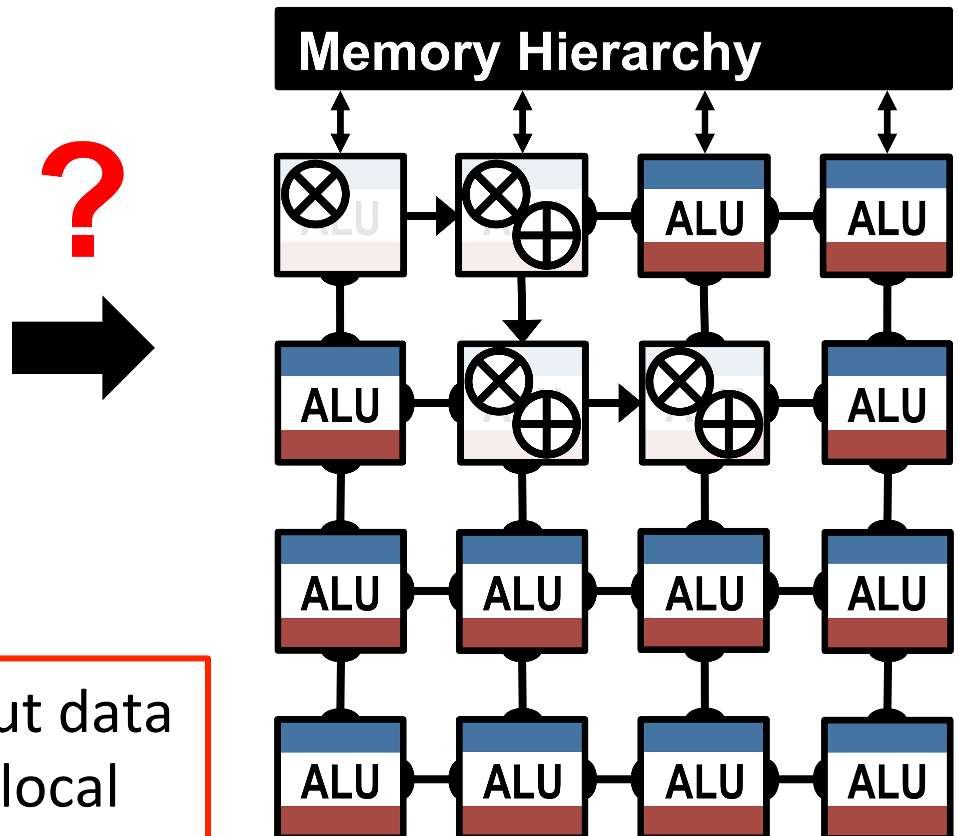


How to Map the Dataflow?

CNN Convolution



Spatial Architecture (Dataflow Processing)



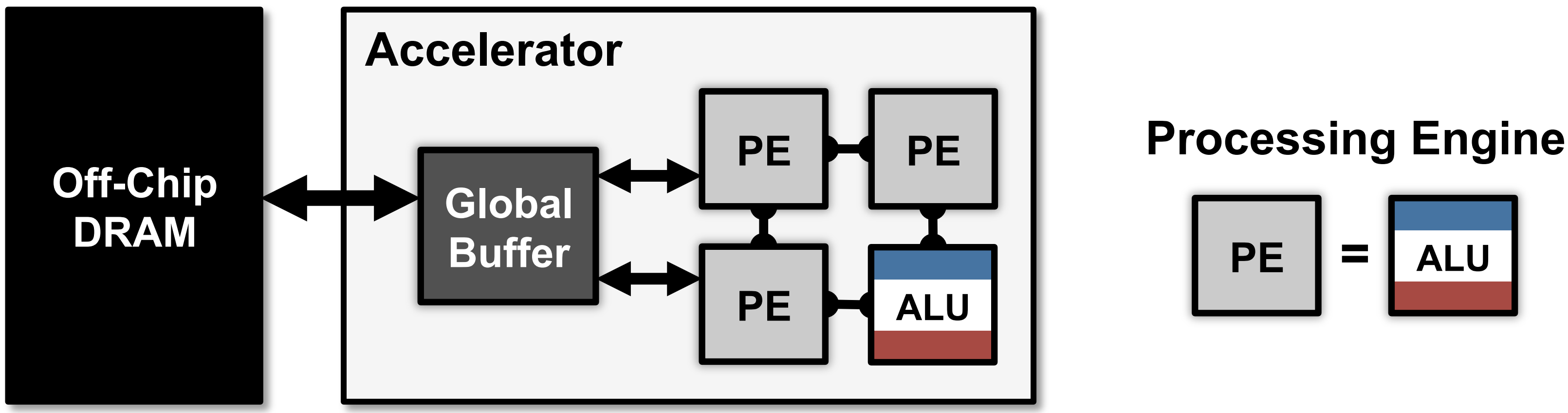
Goal: Increase reuse of input data (**weights** and **pixels**) and local **partial sums** accumulation

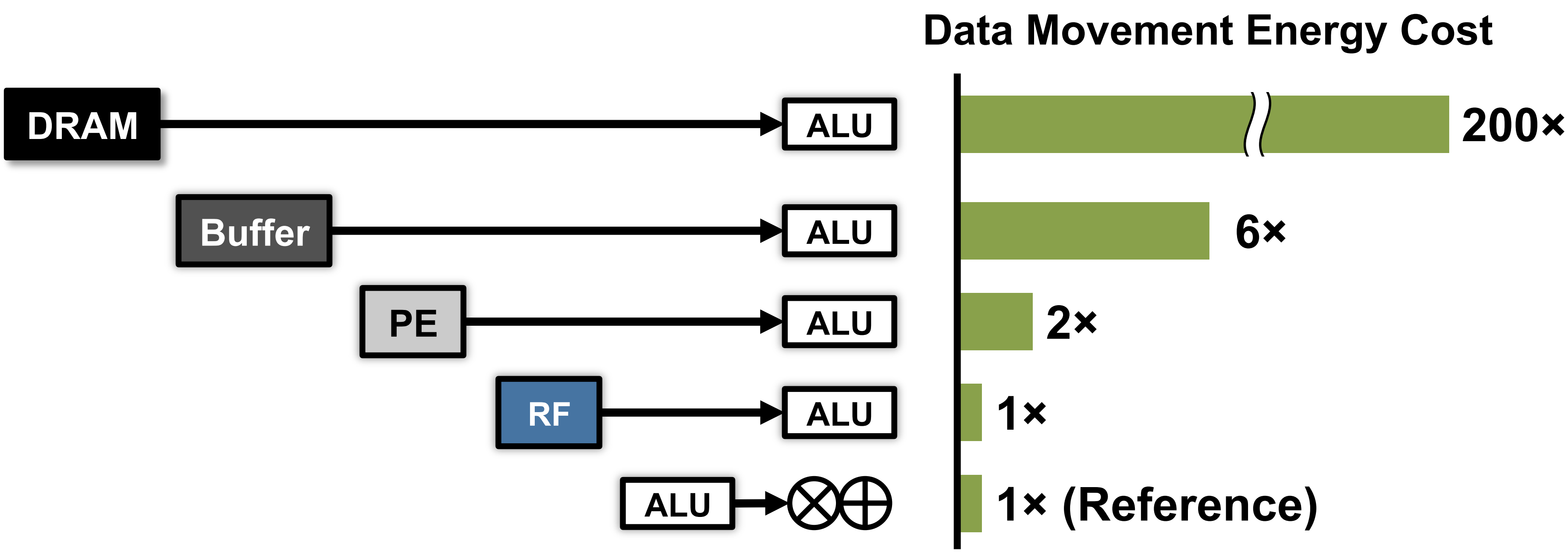
Energy-Efficient Dataflow

Yu-Hsin Chen, Joel Emer, Vivienne Sze, ISCA 2016 [paper]

Maximize data reuse and accumulation at RF

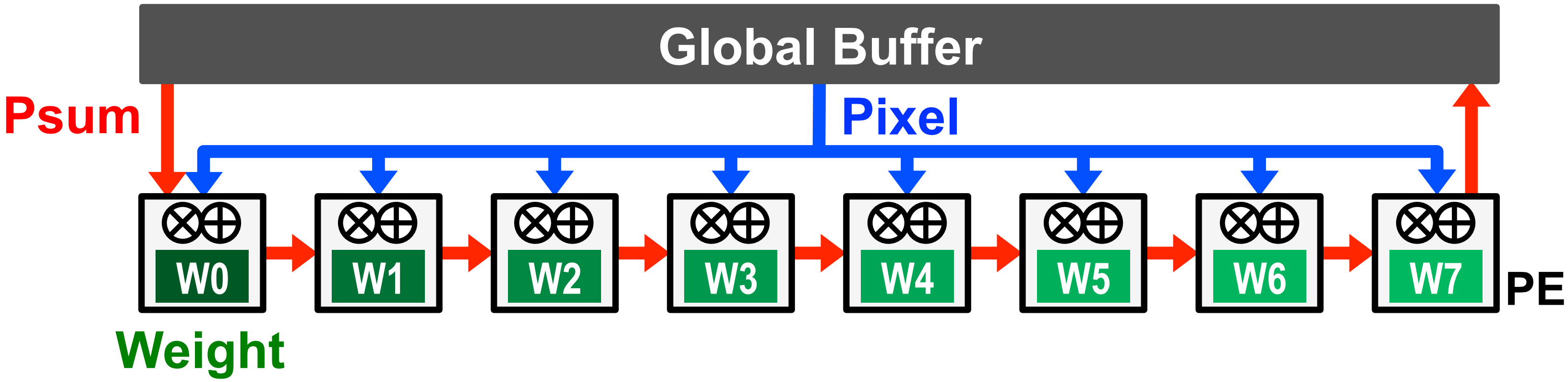
Data Movement is Expensive





Maximize data reuse at lower levels of hierarchy

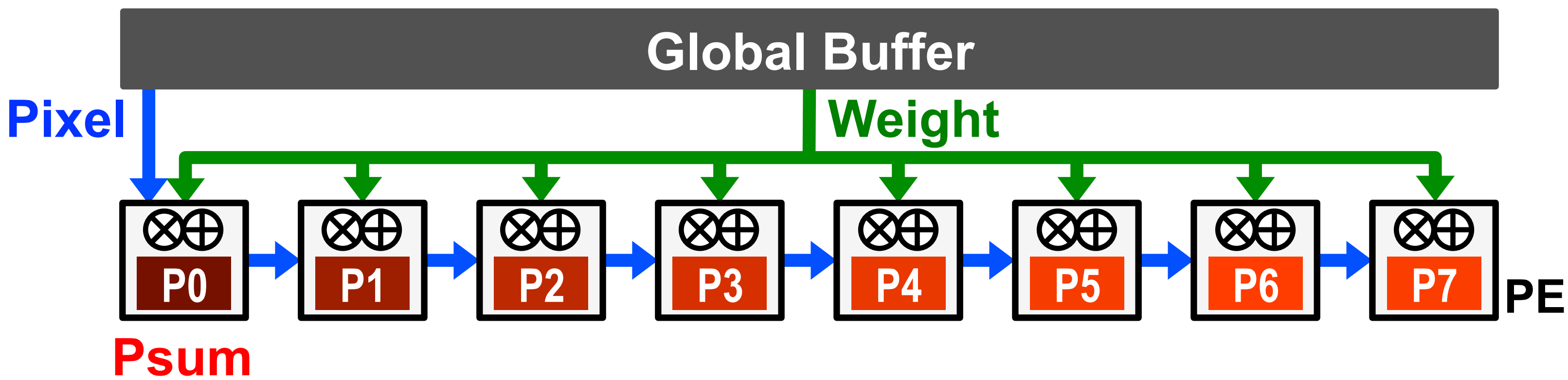
Weight Stationary (WS)



- **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]

Output Stationary (OS)

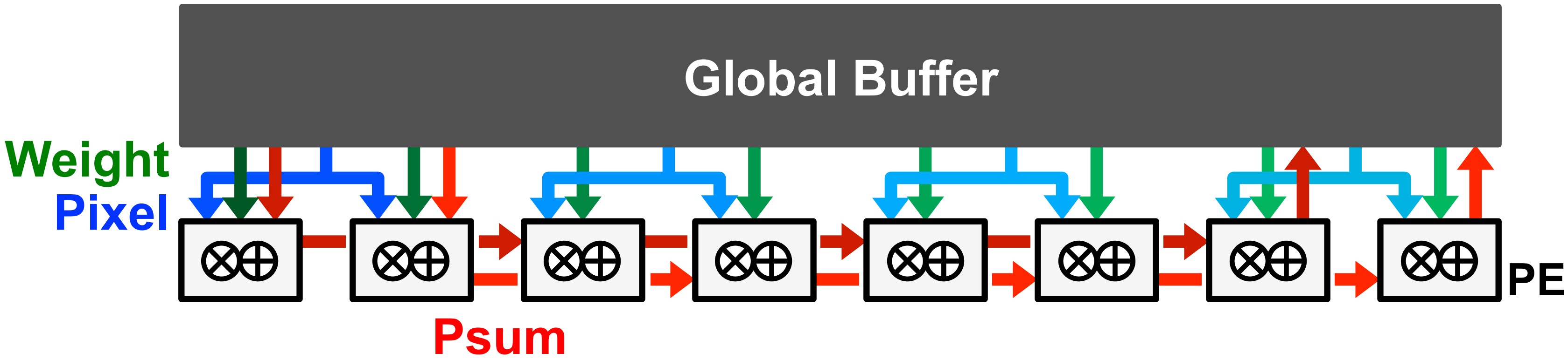


- **Minimize** **partial sum** R/W energy consumption
  - maximize local accumulation
- **Examples:**

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

  [**Peemen**, *ICCD* 2013]

No Local Reuse (NLR)

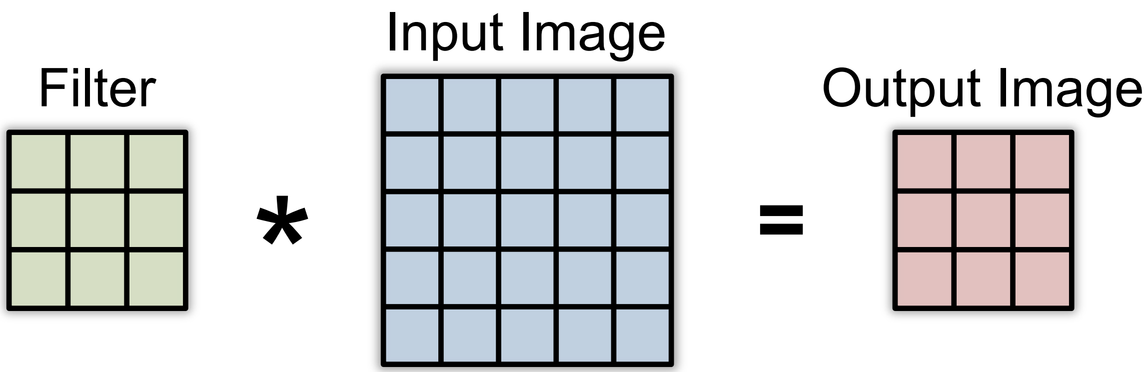


- Use a **large global buffer** as shared storage
  - Reduce **DRAM** access energy consumption
- **Examples:**

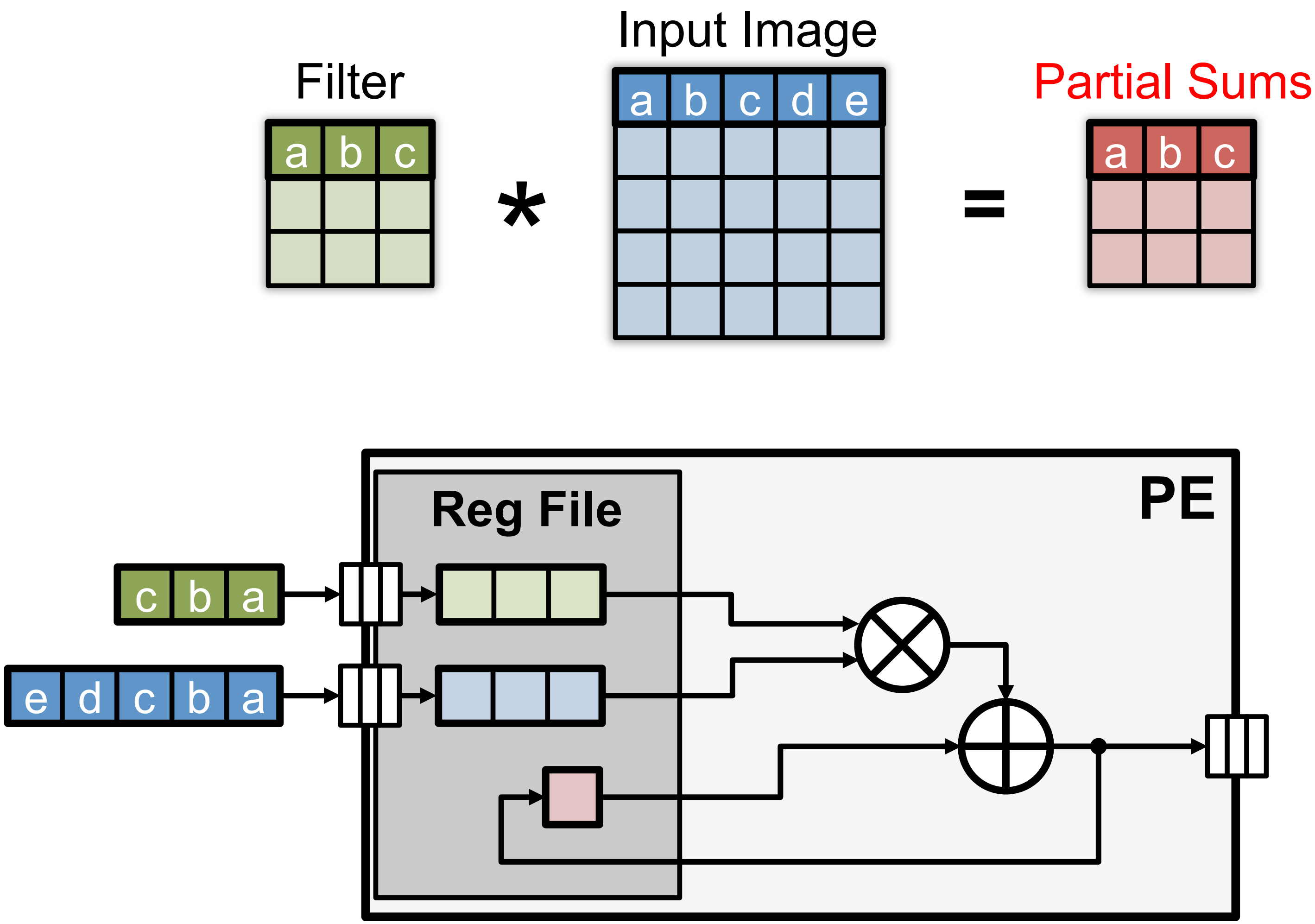
  [**DianNao**, *ASPLOS* 2014] [**DaDianNao**, *MICRO* 2014]

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

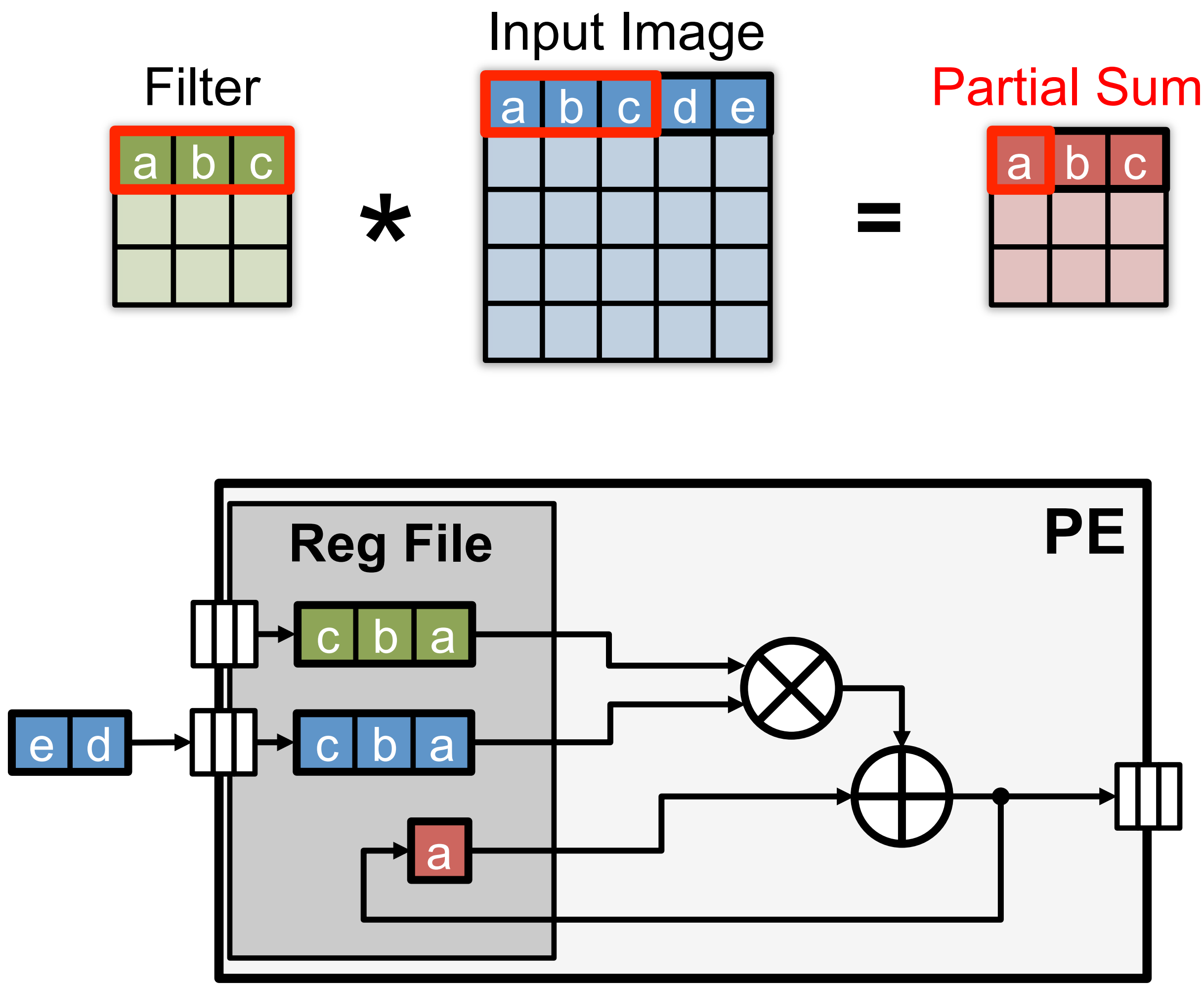
Row Stationary: Energy-efficient Dataflow



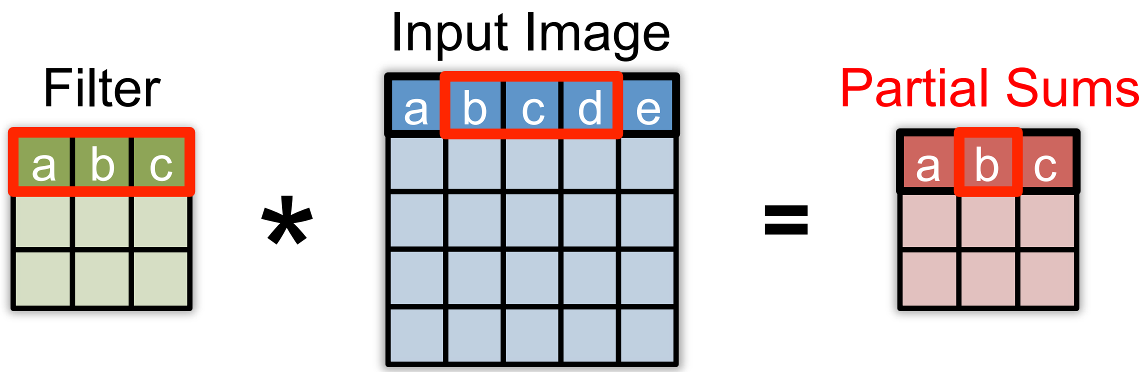
1D Row Convolution in PE

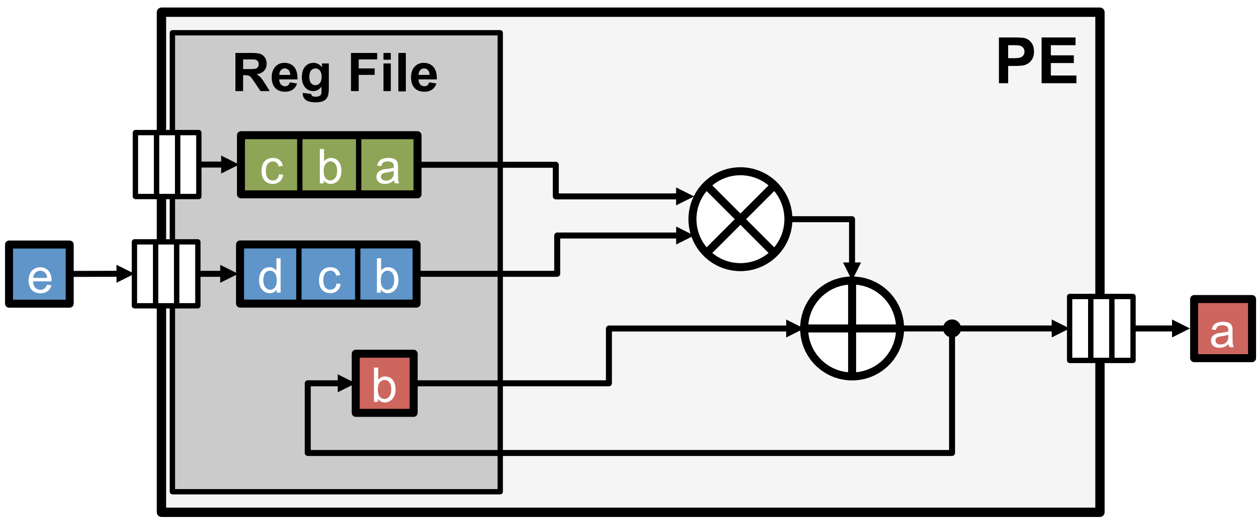


1D Row Convolution in PE

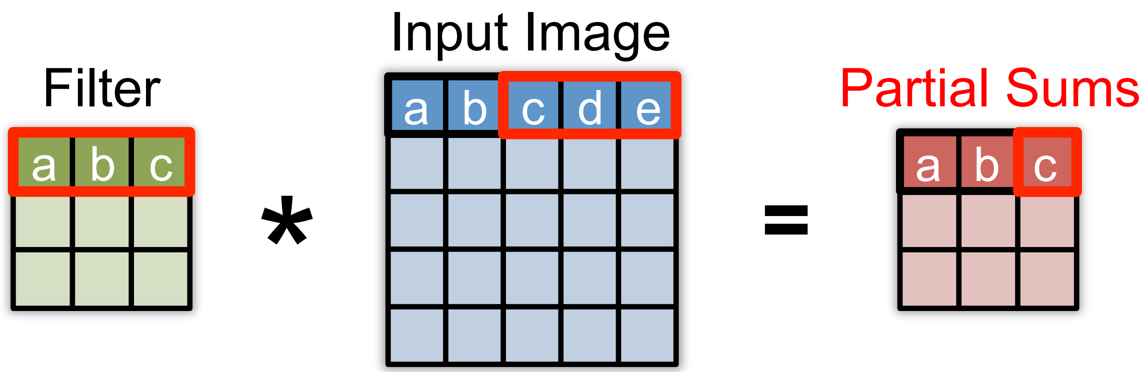


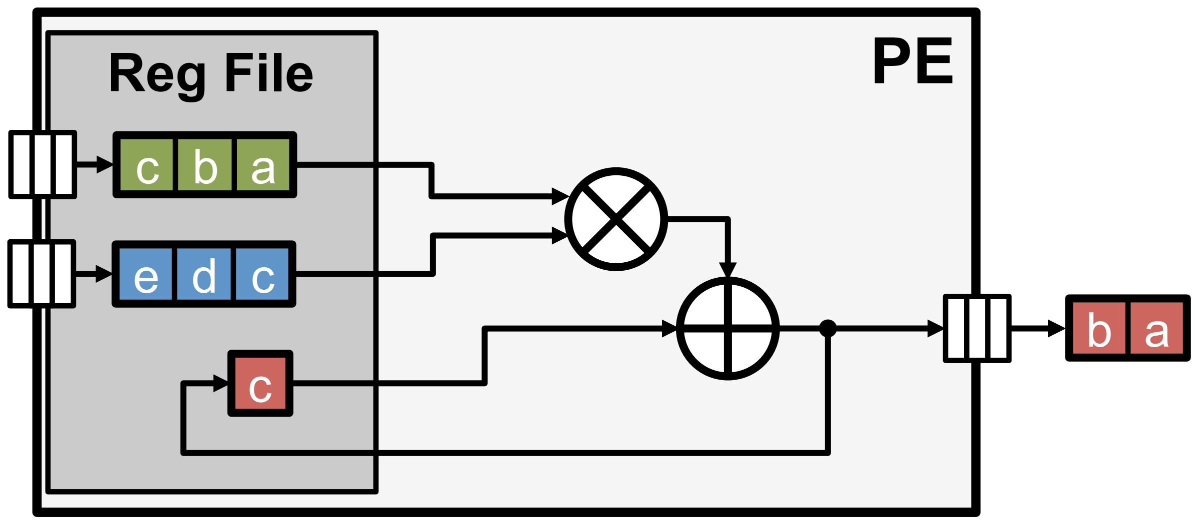
1D Row Convolution in PE





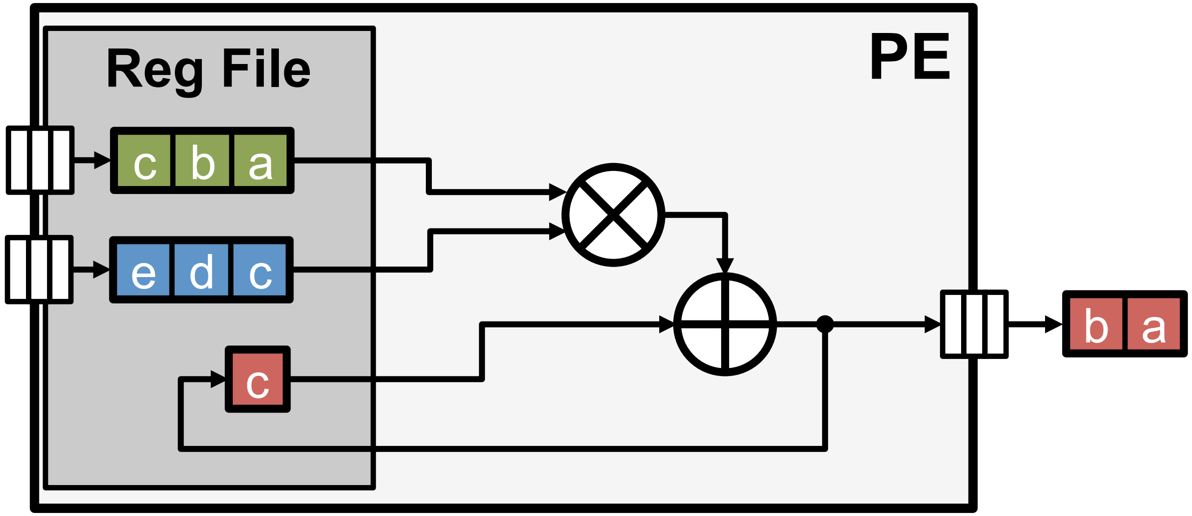
1D Row Convolution in PE



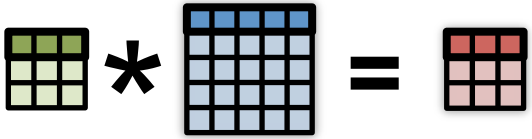


1D Row Convolution in PE

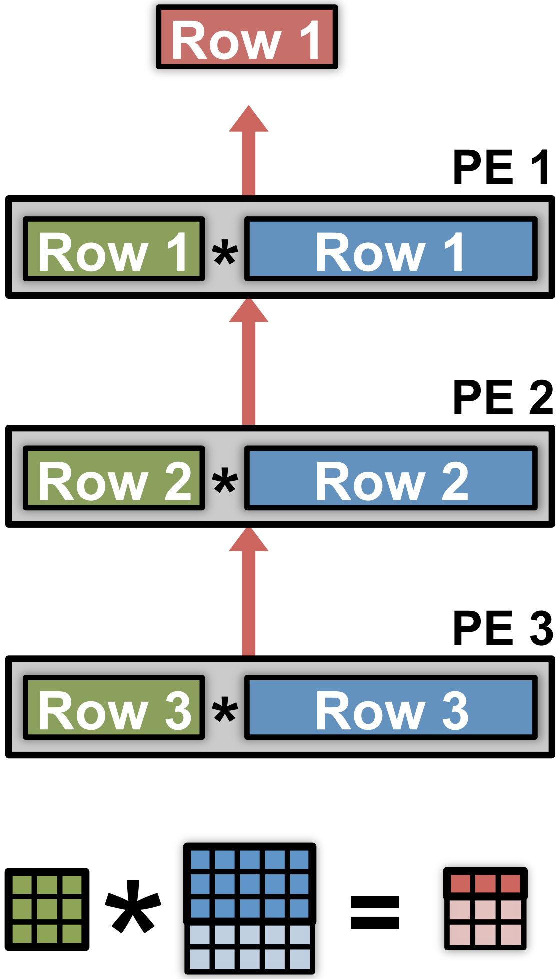
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **image** sliding window in RF
- Maximize row **psum** **accumulation** in RF



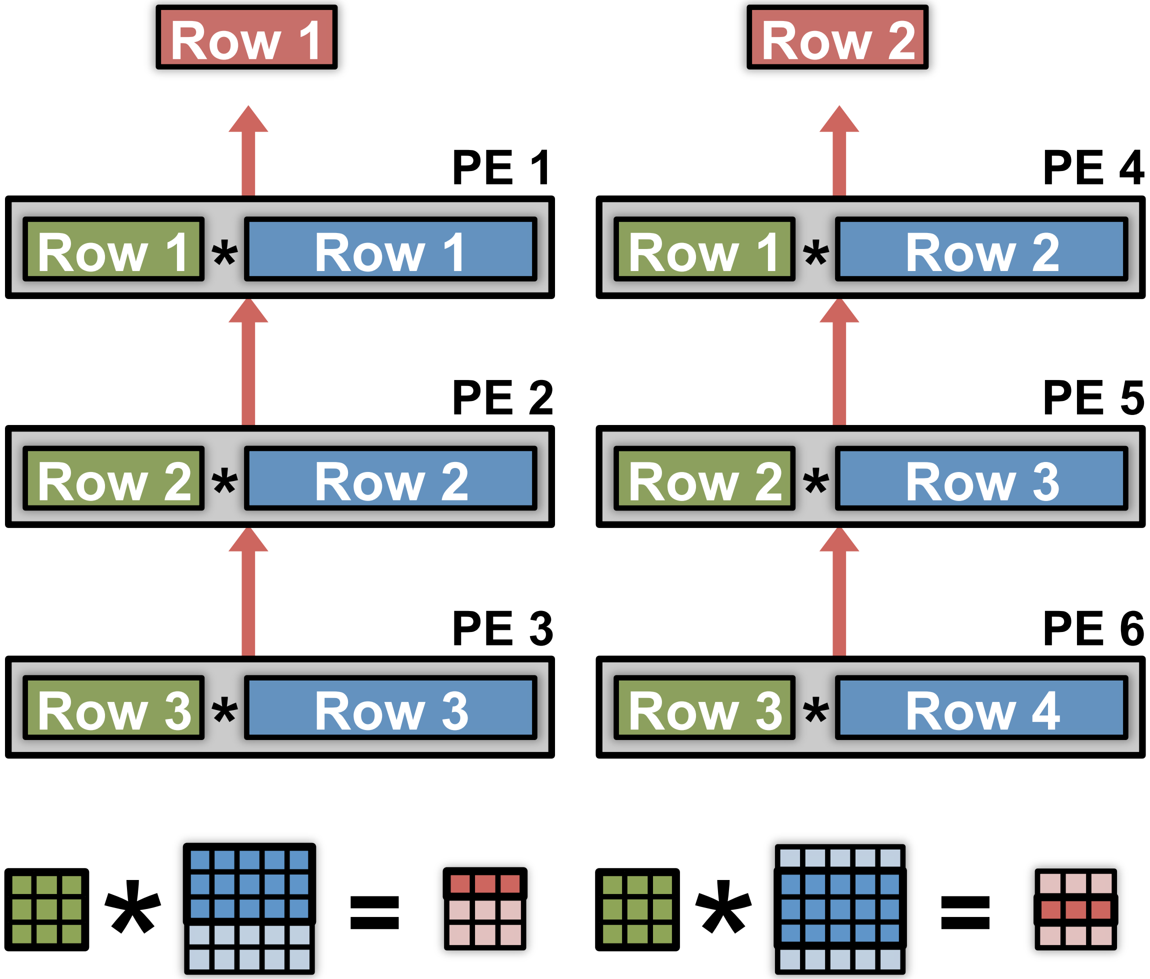
2D Convolution in PE Array



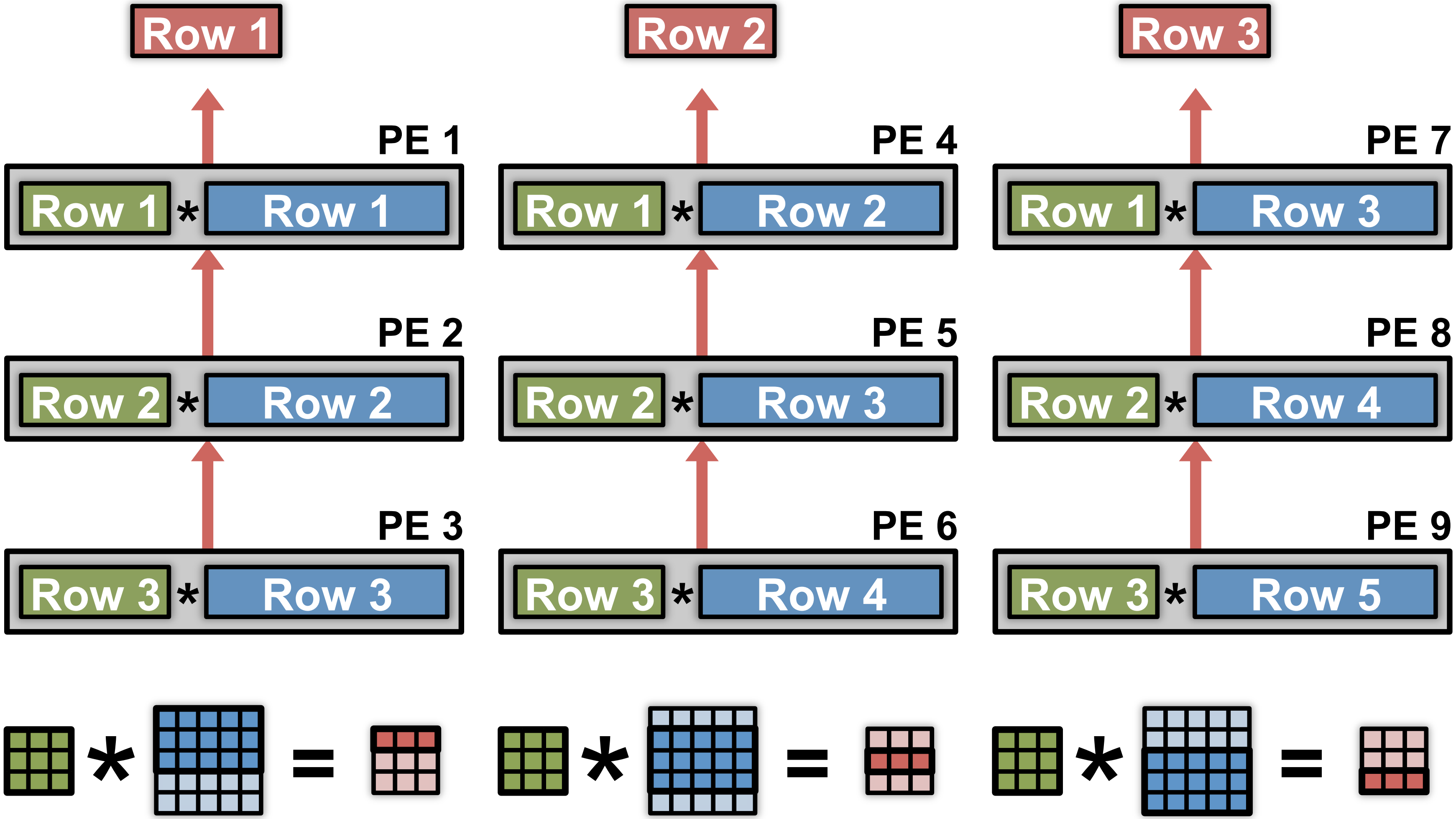
2D Convolution in PE Array



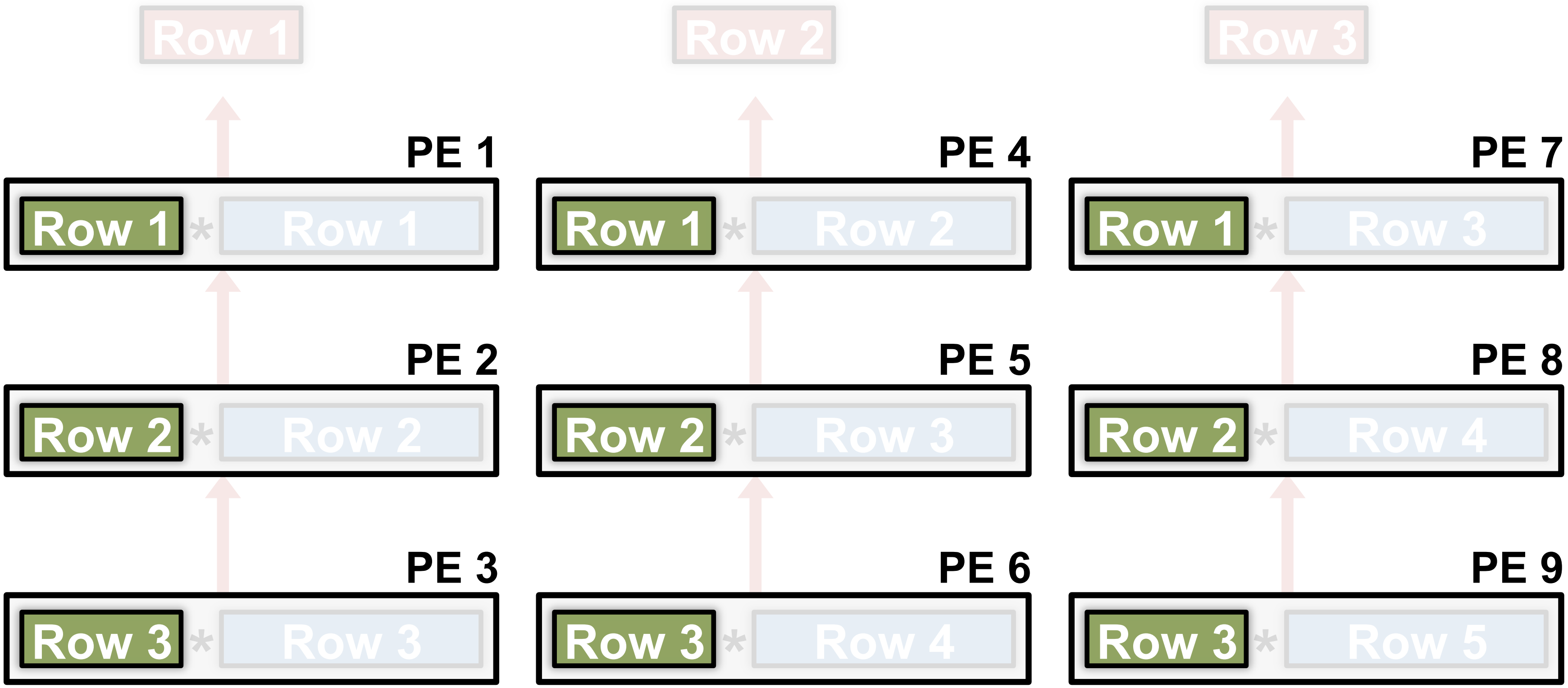
2D Convolution in PE Array



2D Convolution in PE Array



Convolutional Reuse Maximized



Filter rows are reused across PEs **horizontally**

Convolutional Reuse Maximized

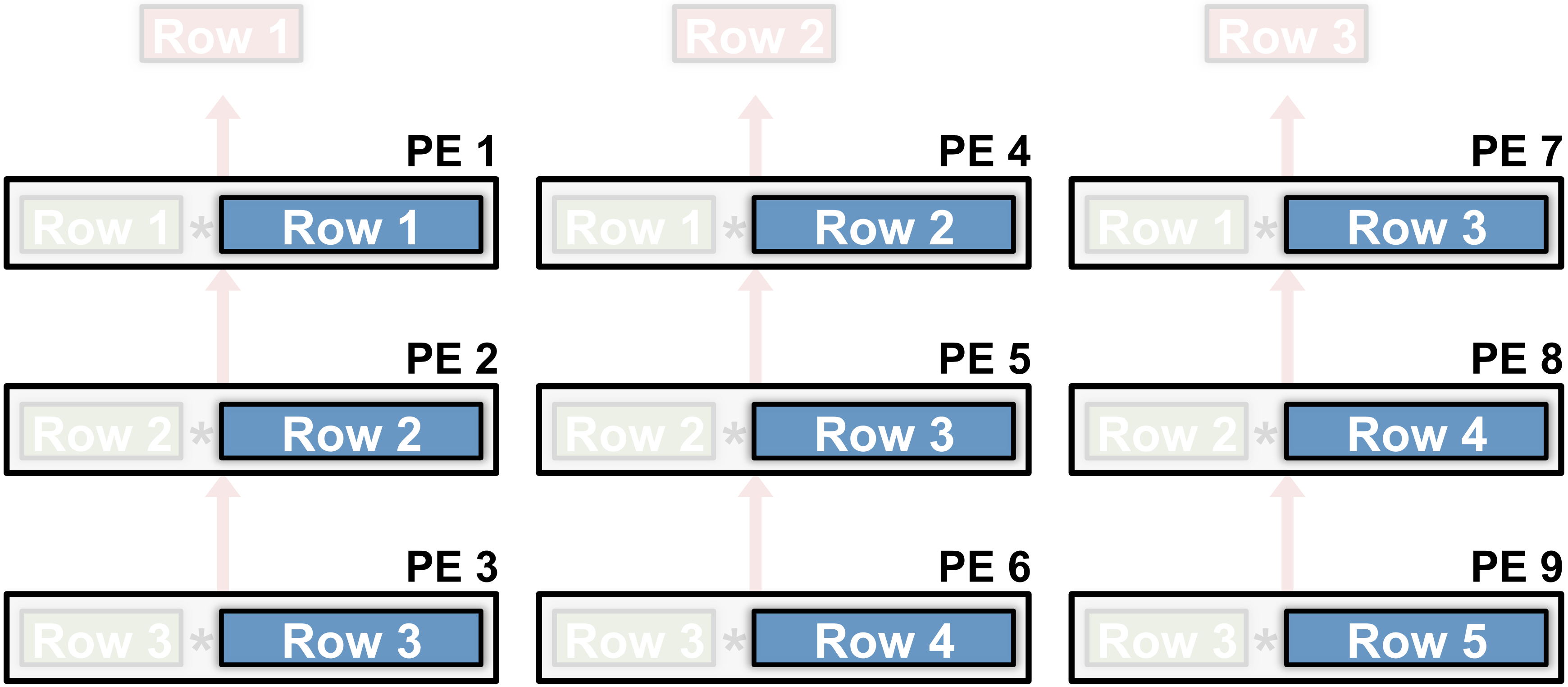
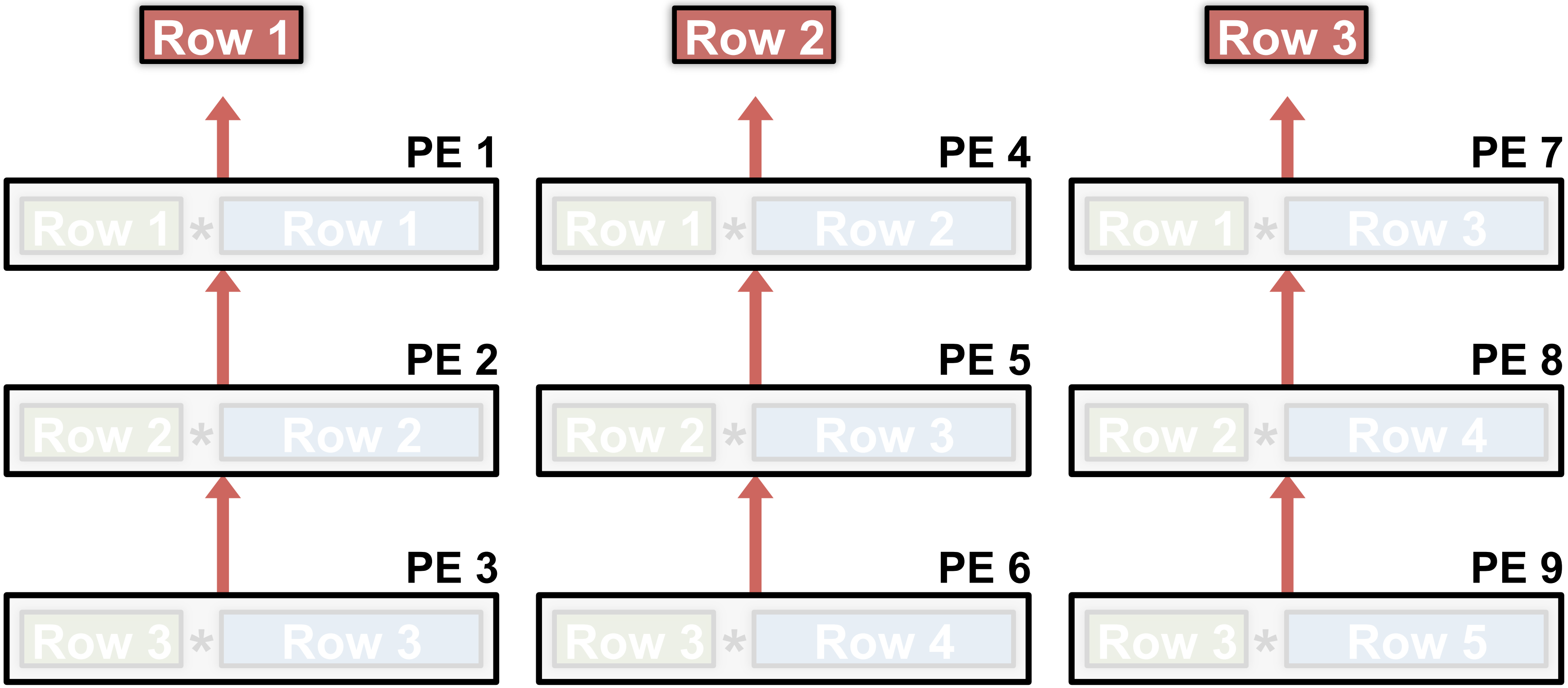


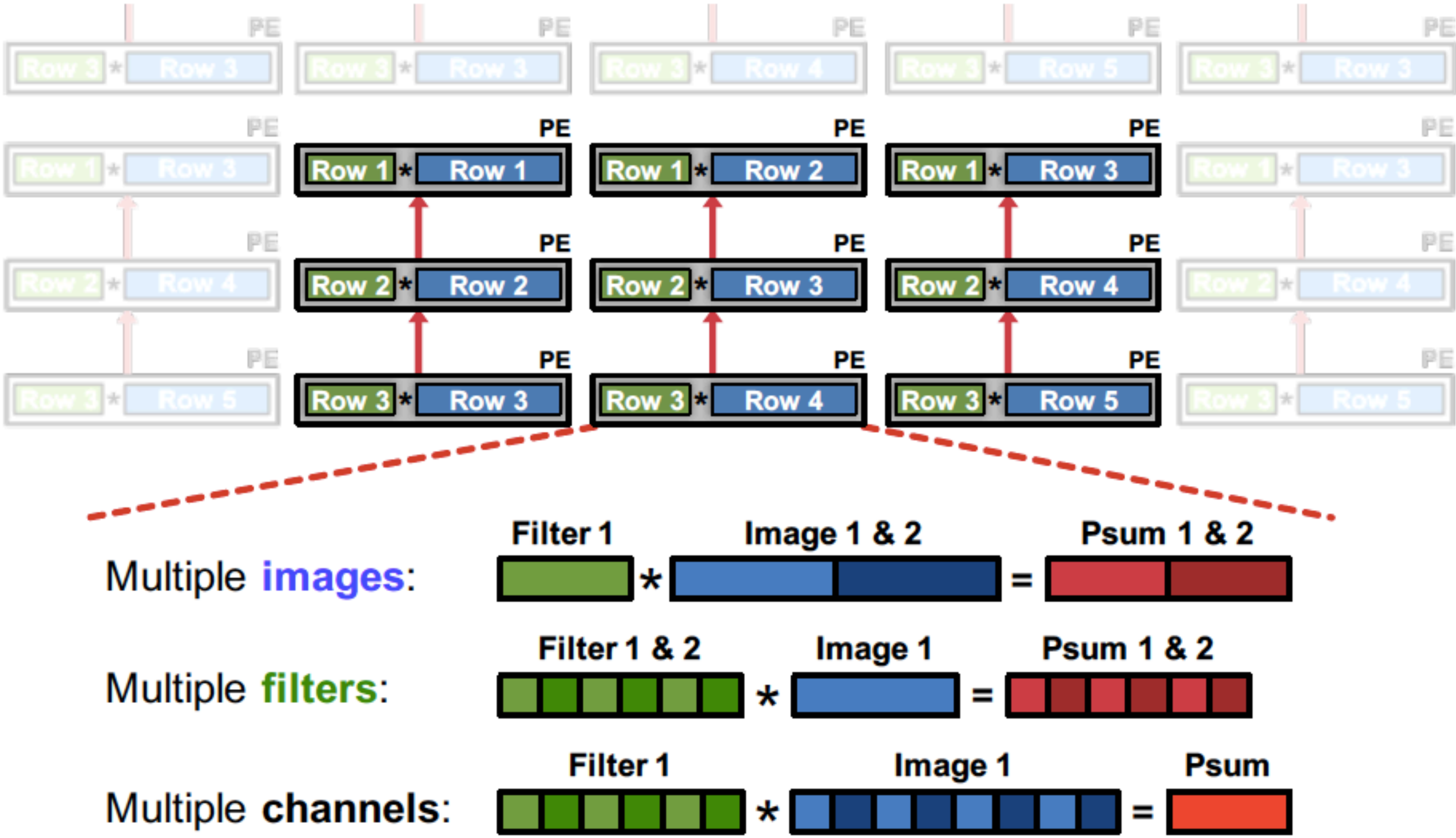
Image rows are reused across PEs **diagonally**

Maximize 2D Accumulation in PE Array



Partial sums accumulate across PEs **vertically**

CNN Convolution – The Full Picture



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

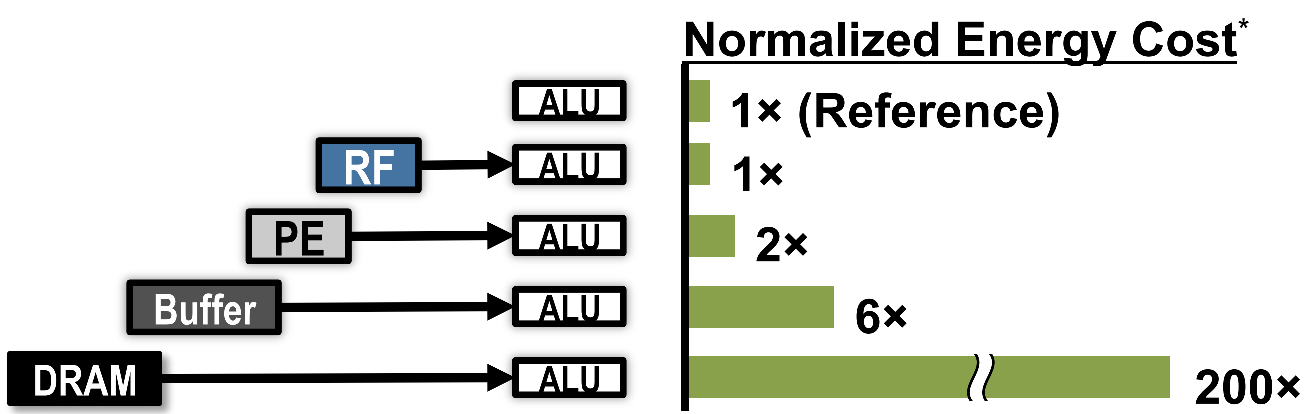
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - Don't use any local PE storage. Maximize global buffer size.
- **Row Stationary**

Evaluate Reuse in Different Dataflows

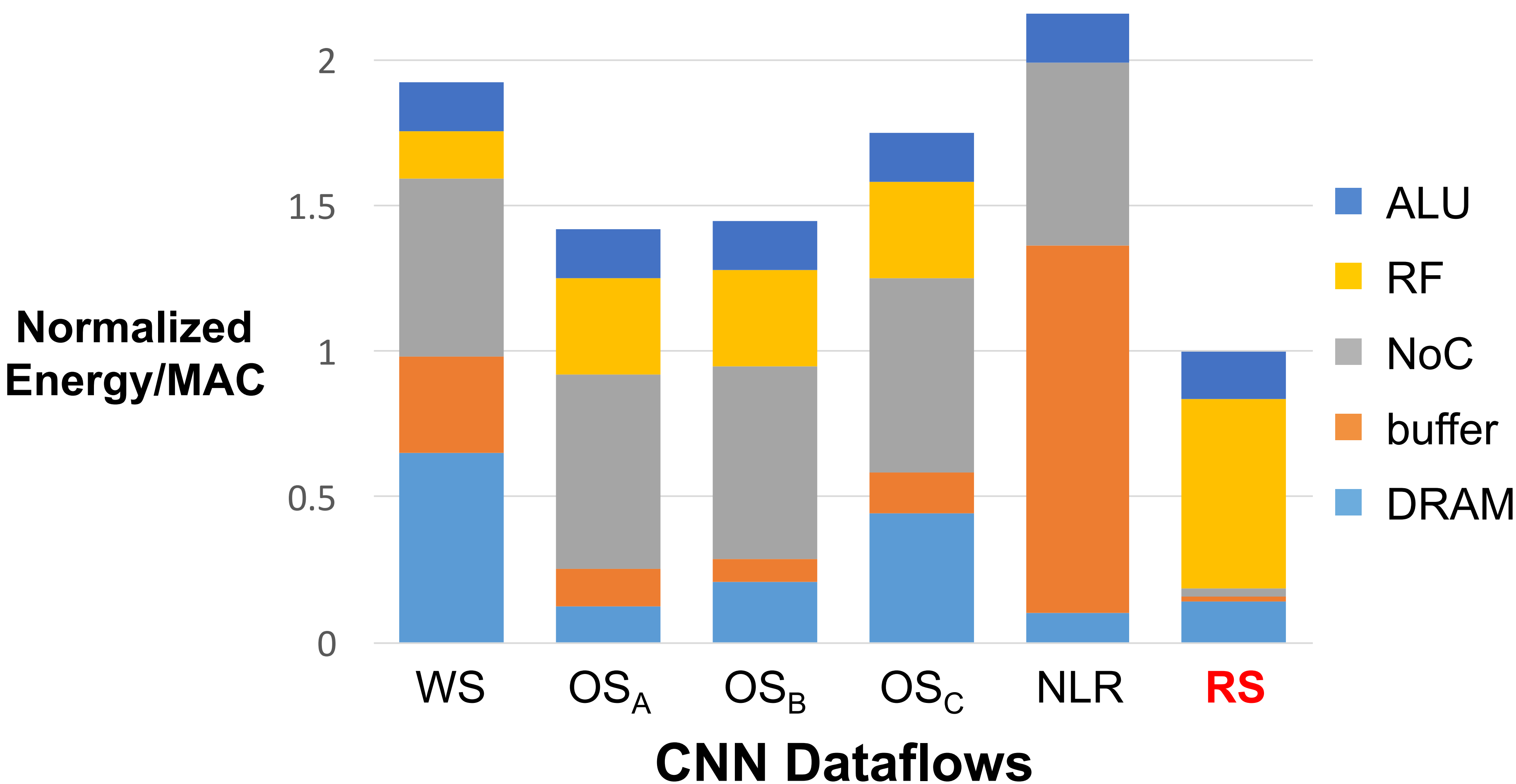
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - Don't use any local PE storage. Maximize global buffer size.
- **Row Stationary**

Evaluation Setup

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

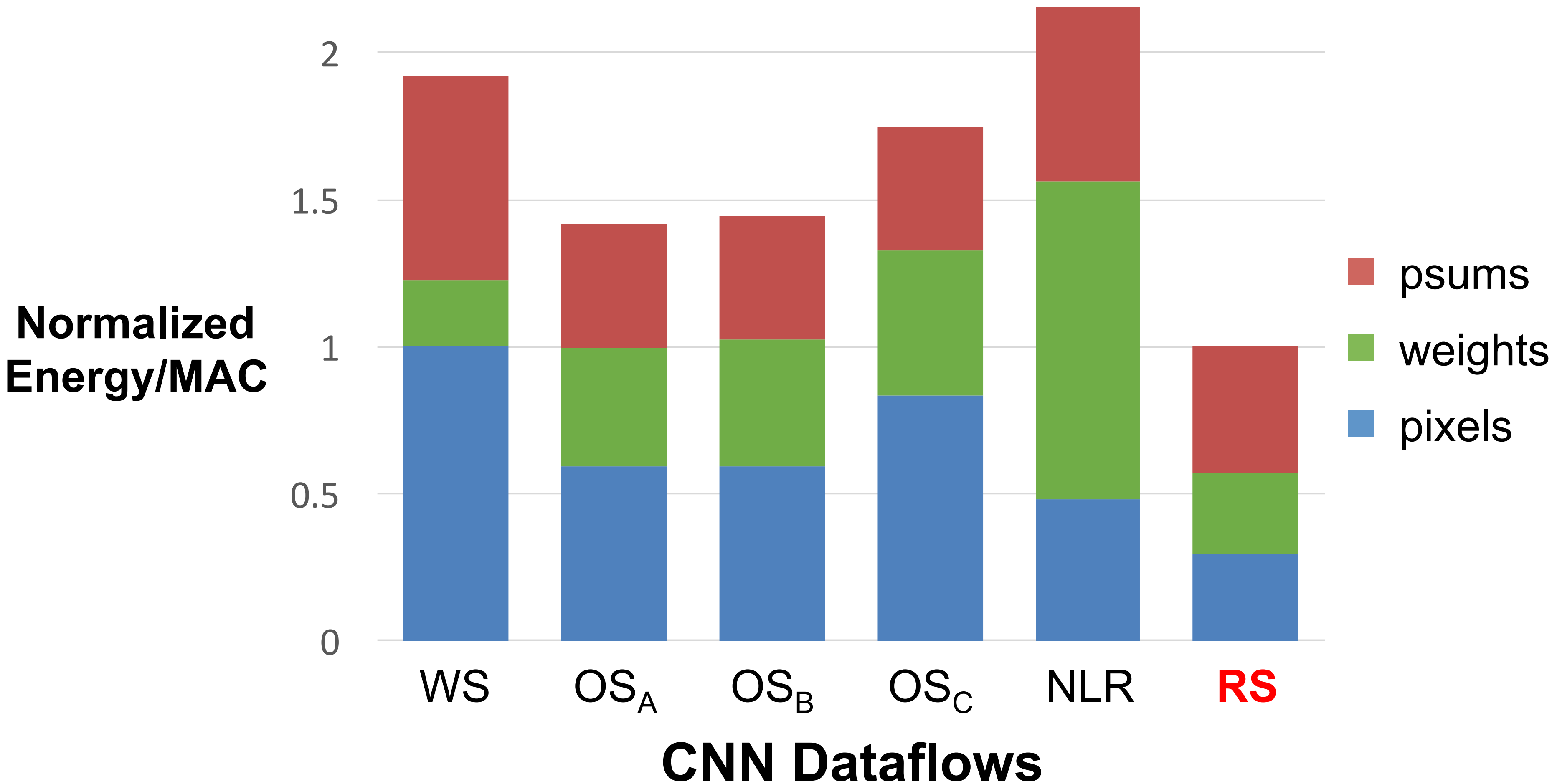


Dataflow Comparison: CONV Layers



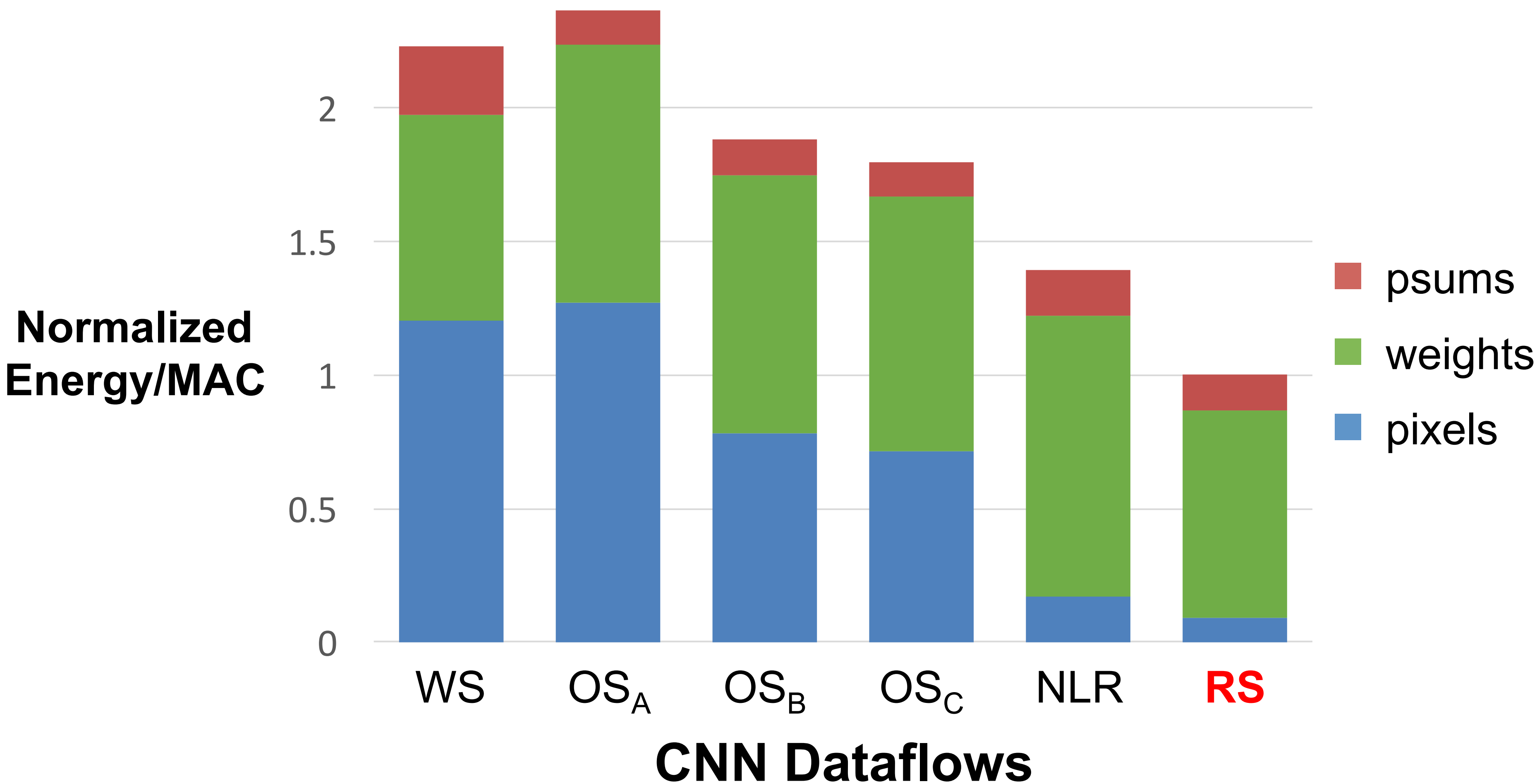
RS uses **1.4× – 2.5× lower** energy than other dataflows

Dataflow Comparison: CONV Layers



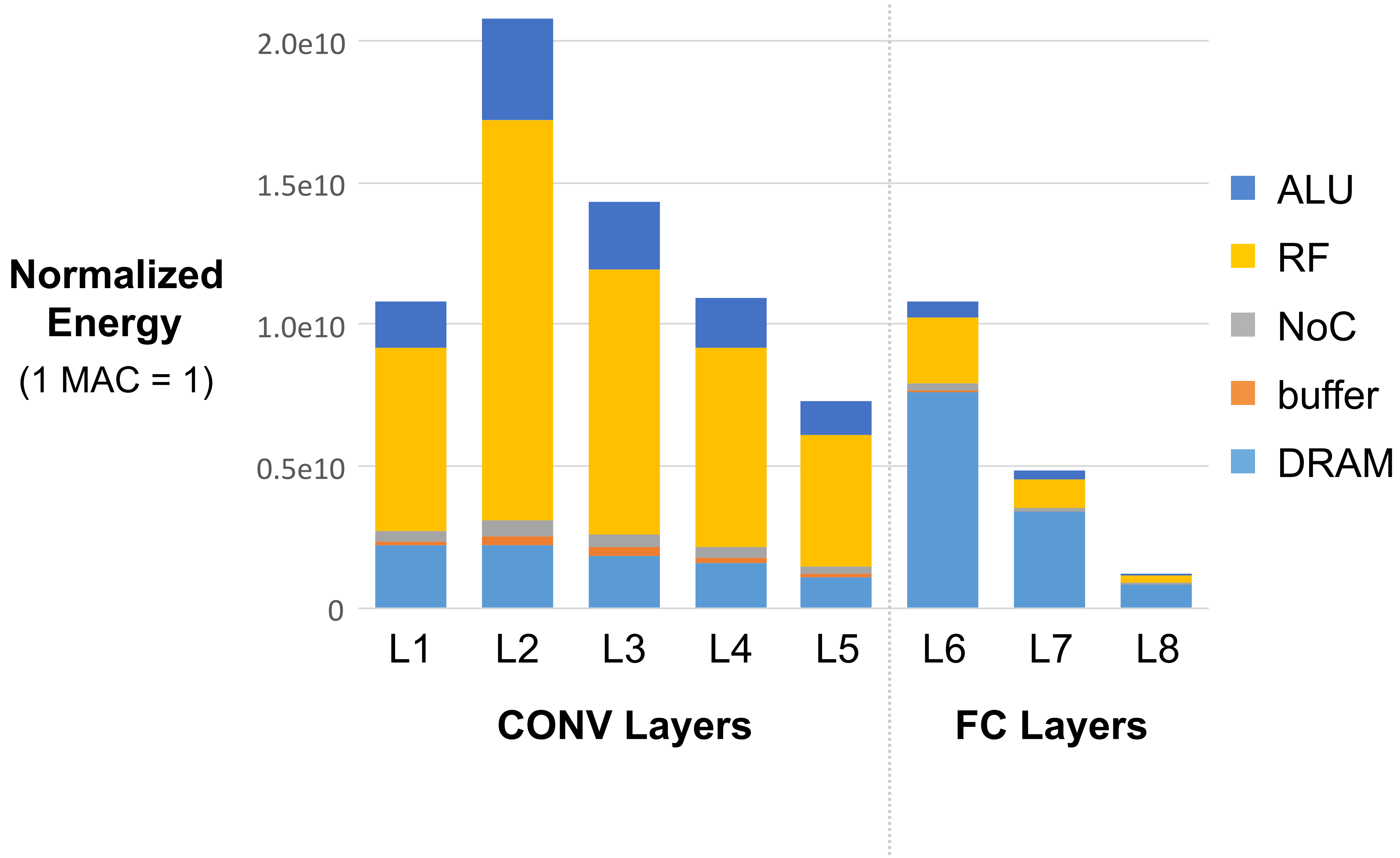
RS optimizes for the best **overall** energy efficiency

Dataflow Comparison: FC Layers

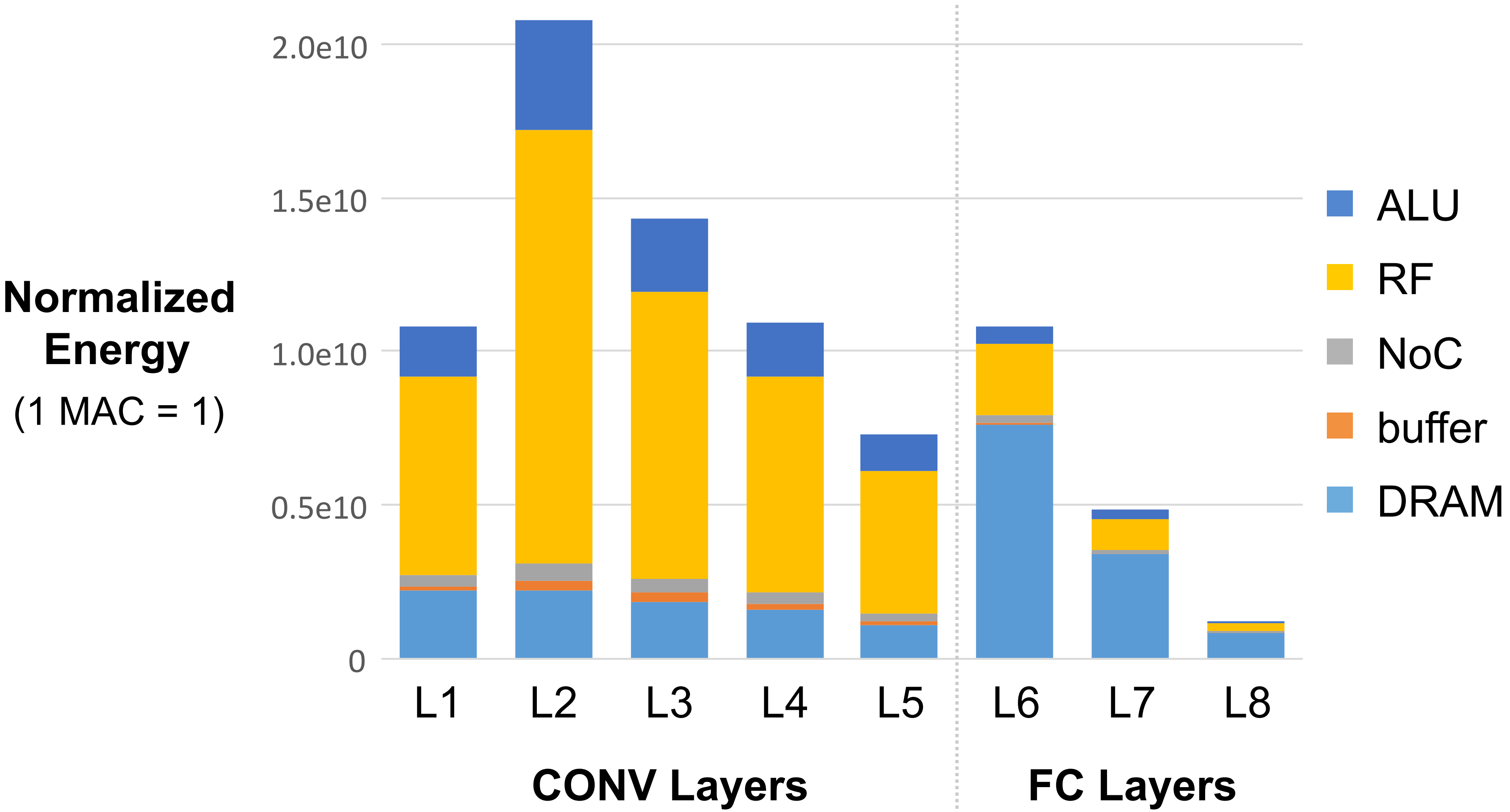


RS uses at least **1.3× lower** energy than other dataflows

Row Stationary: Layer Breakdown

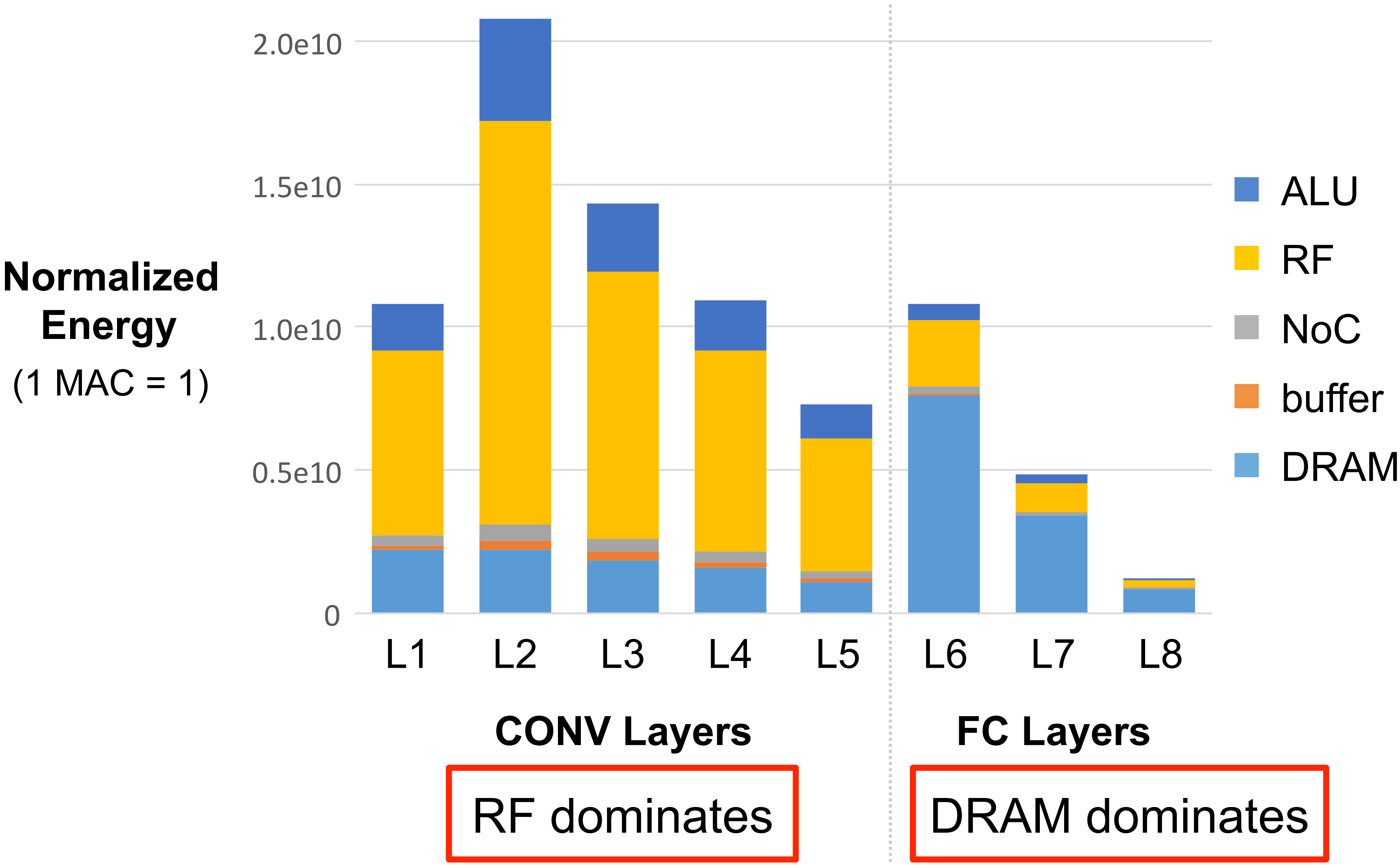


Row Stationary: Layer Breakdown

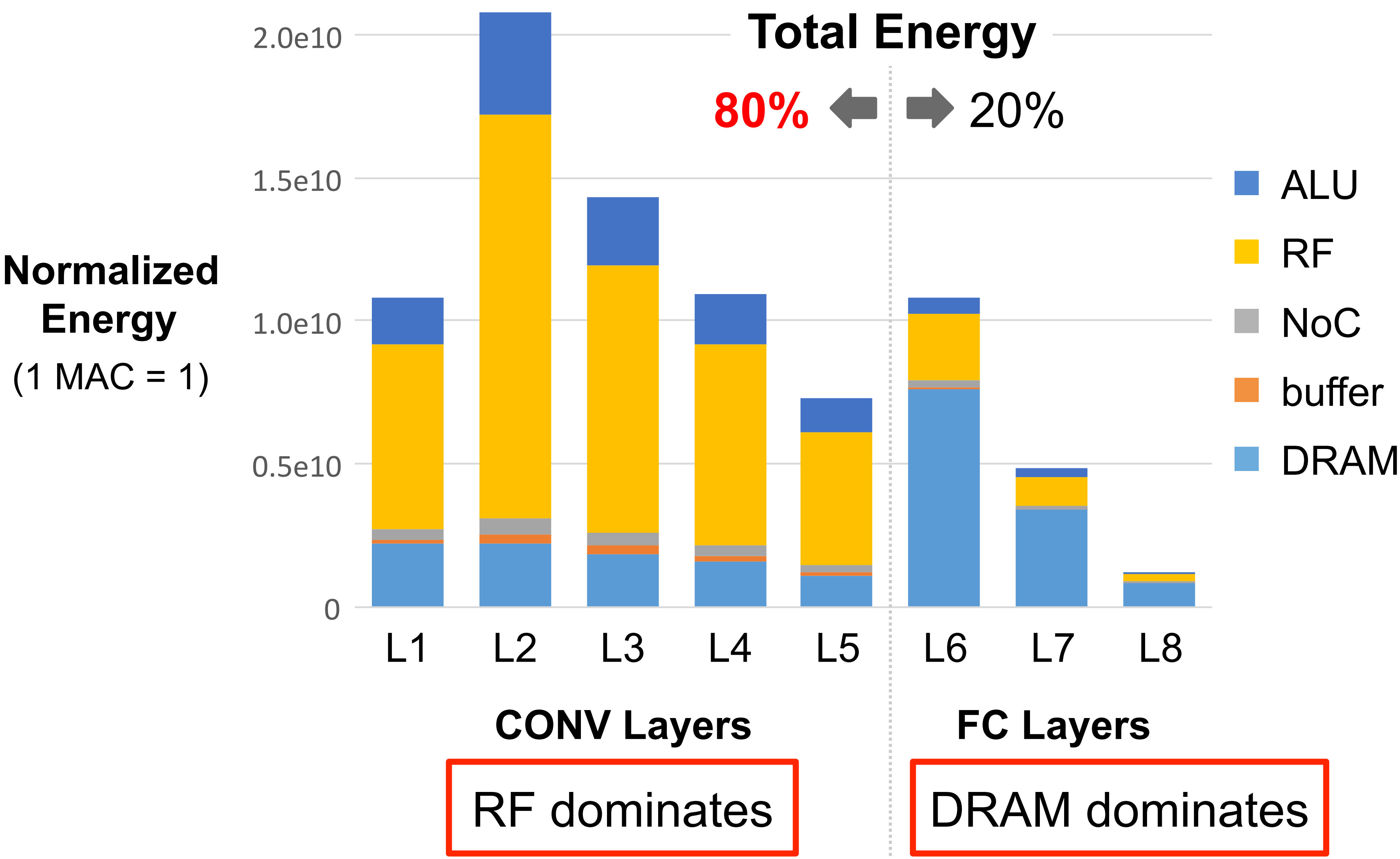


RF dominates

Row Stationary: Layer Breakdown



Row Stationary: Layer Breakdown

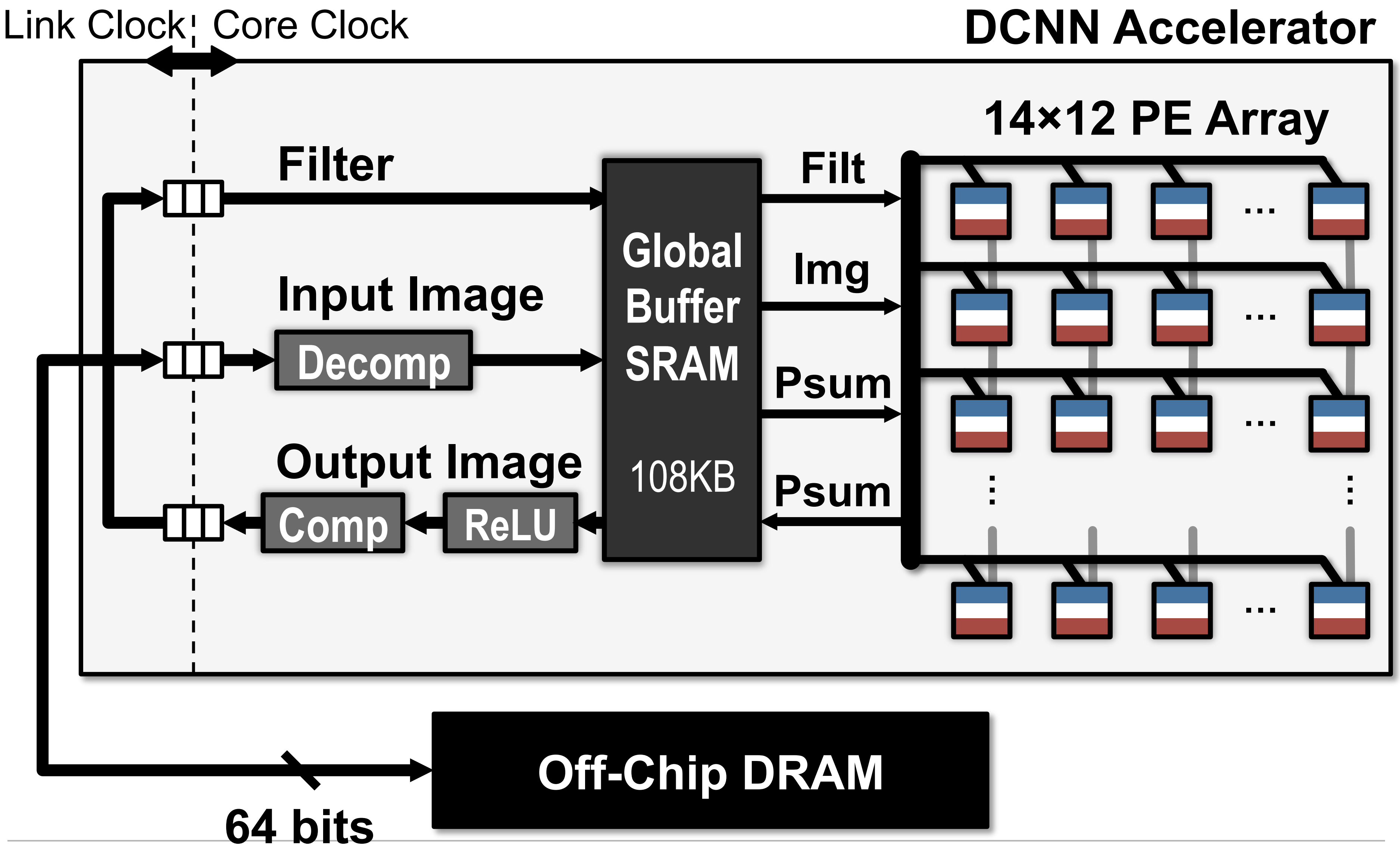


Energy-Efficient Accelerator

Yu-Hsin Chen, Tushar Krishna, Joel Emer, Vivienne Sze, ISSCC 2016 [paper]

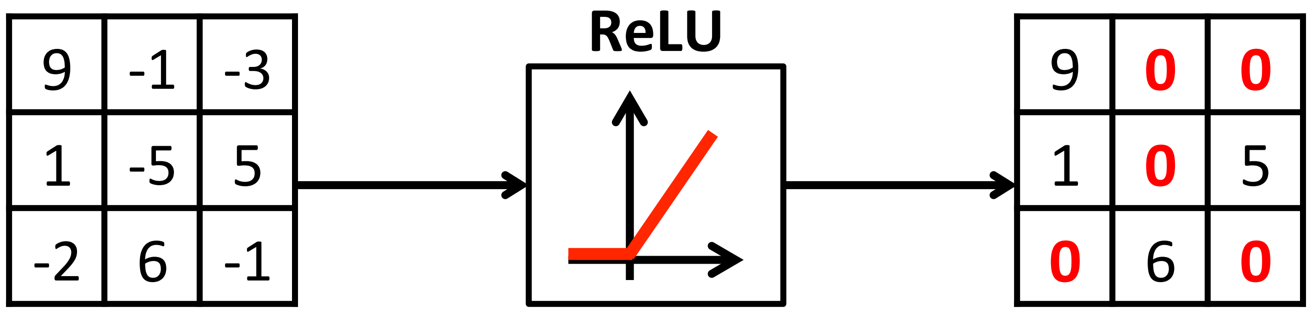
Exploit data statistics

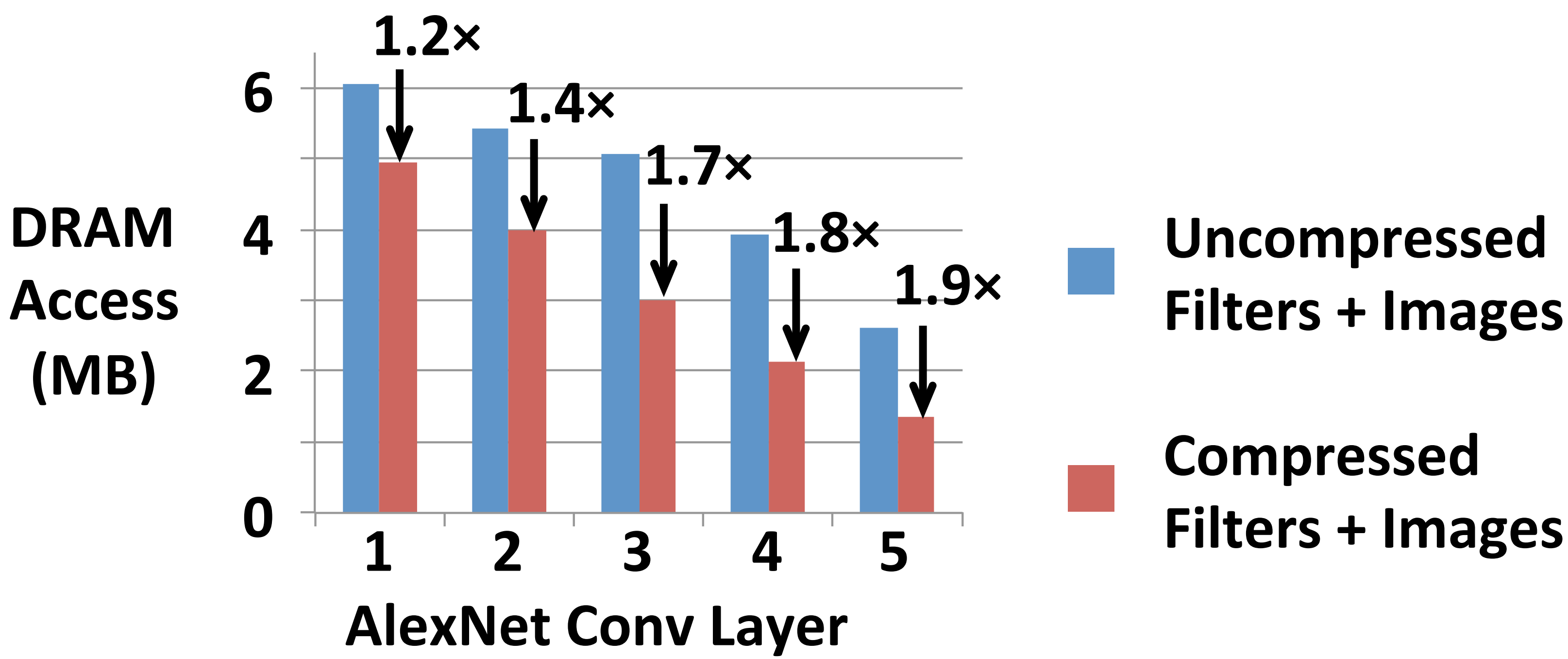
Eyeriss Deep CNN Accelerator



Data Compression Saves DRAM BW

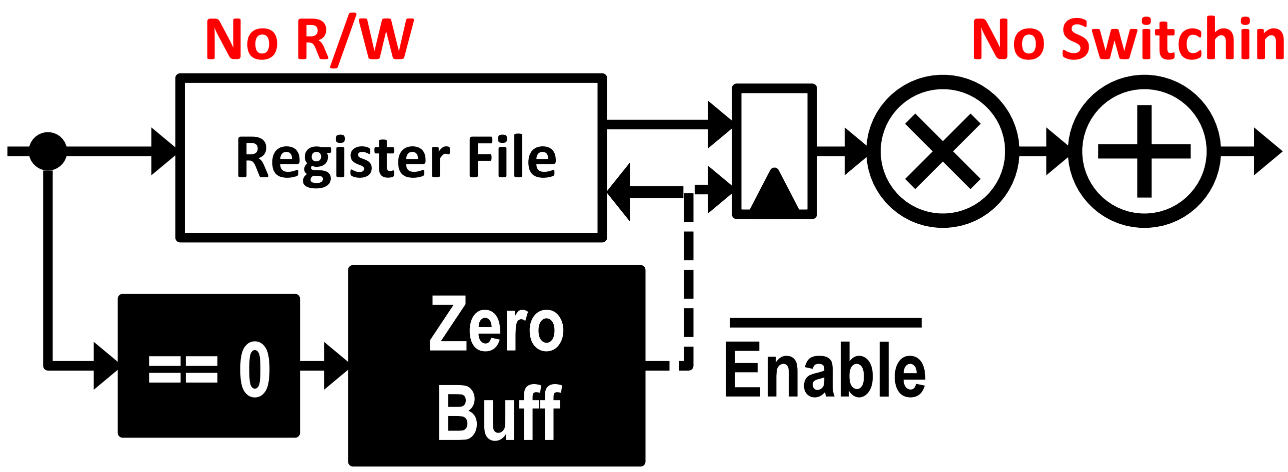
Apply Non-Linearity (**ReLU**) on Filtered Image Data





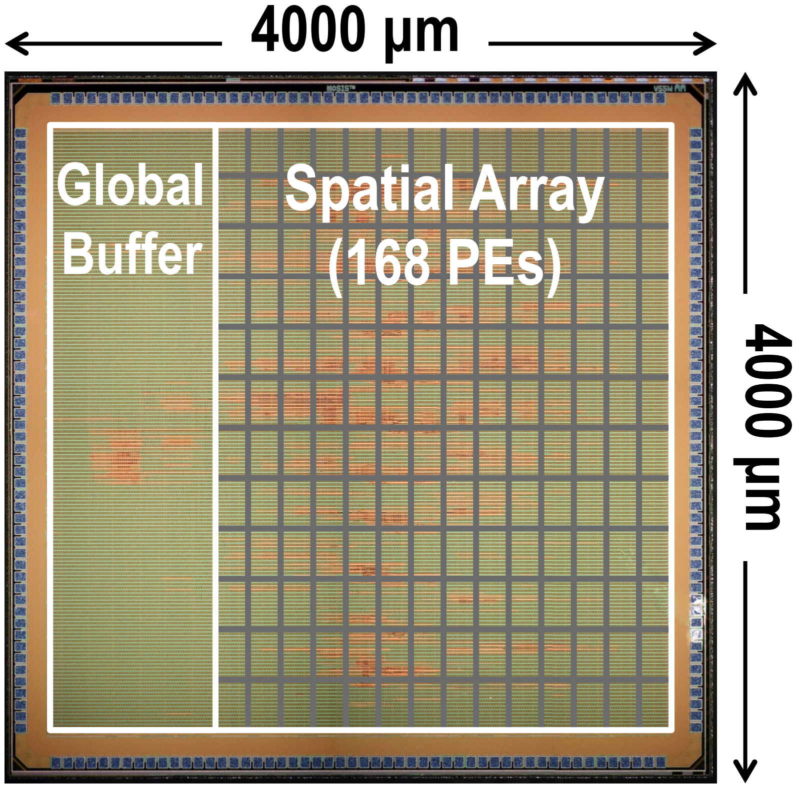
Zero Data Processing Gating

- Skip PE local **memory access**
- Skip MAC **computation**
- Save PE processing power by 45%



Chip Spec & Measurement Results[1]

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 Filter Height: 1 – 12 Num. Filters: 1 – 1024 Num. Channels: 1 – 1024 Horz. Stride: 1–12 Vert. Stride: 1, 2, 4



1. Yu-Hsin Chen, Tushar Krishna, Joel Emer and Vivienne Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *ISSCC 2016*

Benchmark – AlexNet Performance

Image Batch Size of **4** (i.e. 4 frames of 227x227)

Core Frequency = 200MHz / Link Frequency = 60 MHz

Layer	Power (mW)	Latency (ms)	# of MAC (MOPs)	Active # of PEs (%)	Buffer Data Access (MB)	DRAM Data Access (MB)
1	332	20.9	422	154 (92%)	18.5	5.0
2	288	41.9	896	135 (80%)	77.6	4.0
3	266	23.6	598	156 (93%)	50.2	3.0
4	235	18.4	449	156 (93%)	37.4	2.1
5	236	10.5	299	156 (93%)	24.9	1.3
Total	**278**	**115.3**	**2663**	**148 (88%)**	**208.5**	**15.4**

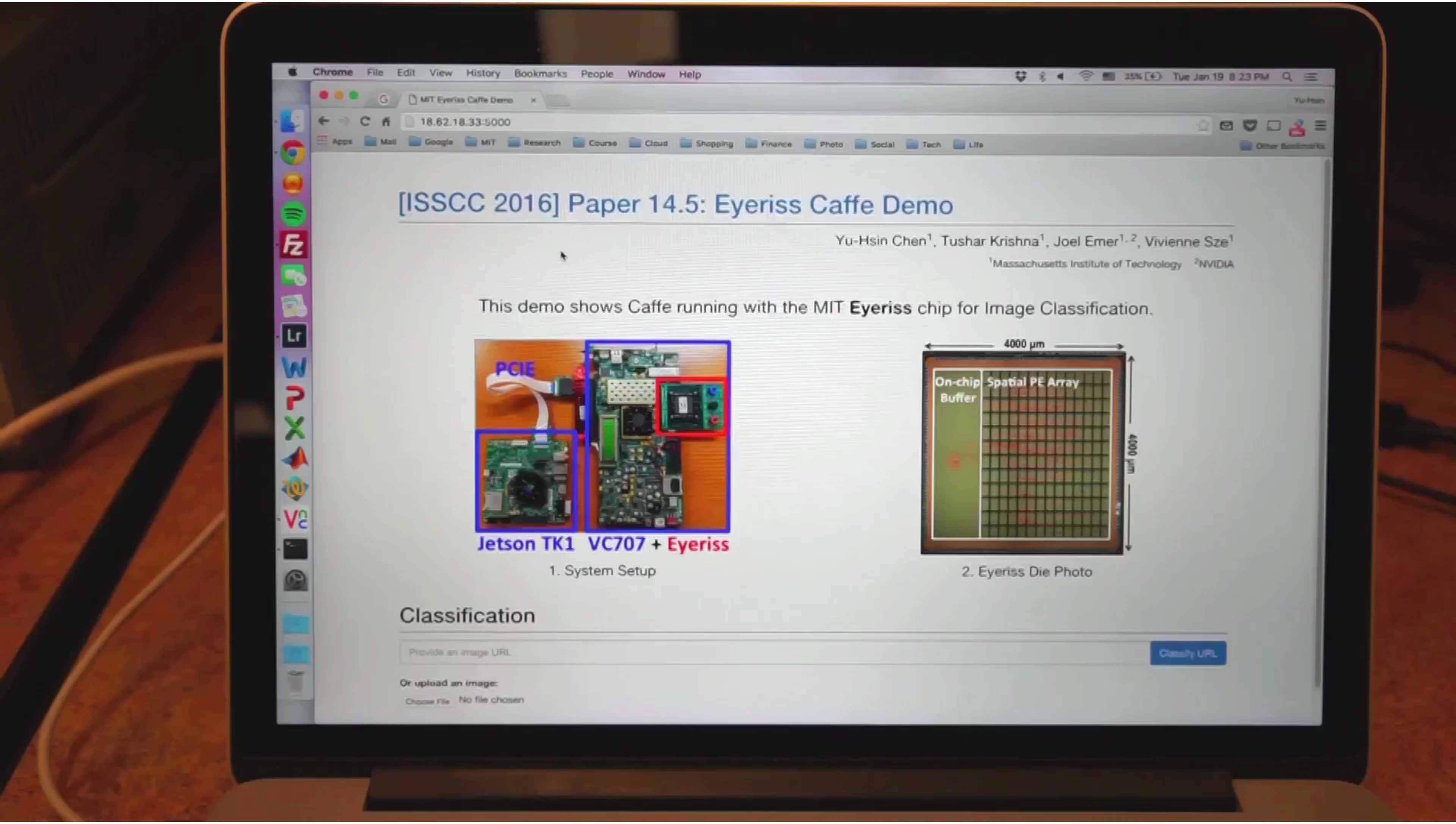
To support 2.66 GMACs [8 billion 16-bit inputs (**16GB**) and 2.7 billion outputs (**5.4GB**)], only requires **208.5MB** (buffer) and **15.4MB** (DRAM)

Comparison with GPU

	This Work	NVIDIA TK1 (Jetson Kit)
Technology	65nm	28nm
Clock Rate	200MHz	852MHz
# Multipliers	168	192
On-Chip Storage	Buffer: 108KB Spad: 75.3KB	Shared Mem: 64KB Reg File: 256KB
Word Bit-Width	16b Fixed	32b Float
Throughput[1]	34.7 fps	68 fps
Measured Power	278 mW	Idle/Active[2]: 3.7W/10.2W
DRAM Bandwidth	127 MB/s	1120 MB/s [3]

1. AlexNet Convolutional Layers Only
2. Board Power
3. Modeled from [Tan, SC11]

Demo of Image Classification on Eyeriss



https://vimeo.com/154012013

Integrated with BVLC Caffe DL Framework

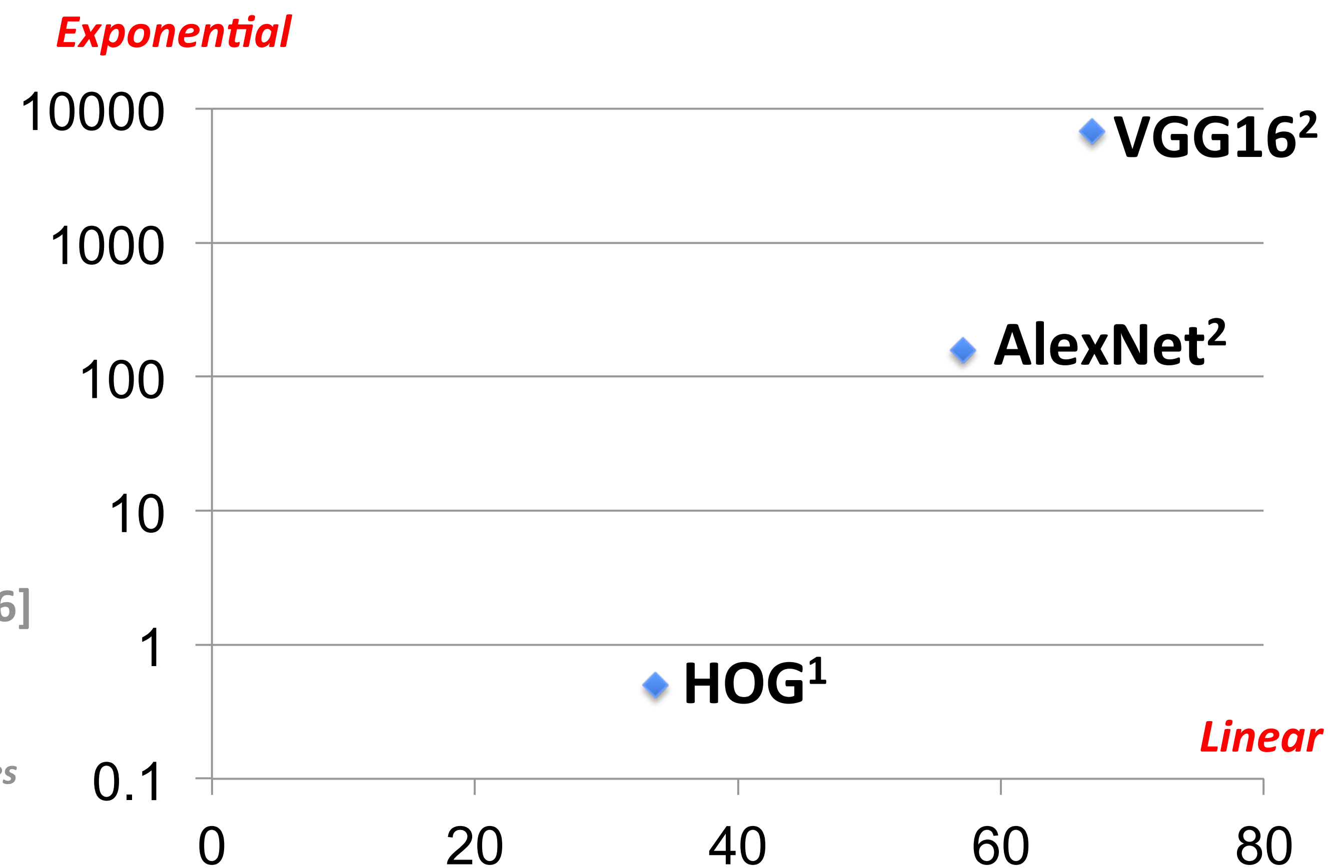
Summary of Eyeriss Deep CNN

- **Eyeriss:** a **reconfigurable** accelerator for state-of-the-art deep CNNs **at below 300mW**
- Energy-efficient **dataflow to reduce data movement**
- **Exploit data statistics** for high energy efficiency
- **Integrated** with the **Caffe DL framework** and demonstrated an image classification system

Learn more about **Eyeriss** at

http://eyeriss.mit.edu

Features: Energy vs. Accuracy



Energy/ Pixel (nJ)

*Measured in 65nm**

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

** Only feature extraction. Does not include ensemble, classification, etc.*

Accuracy (Average Precision)

Measured in on VOC 2007 Dataset

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

Summary

- **Energy-Efficient Approaches**
  - Exploit sparsity with joint algorithm and hardware design
  - Minimize data movement
  - Balance flexibility and energy-efficiency
- With energy-efficient approaches, **hand-crafted feature based object detection** can have **similar energy-efficiency as video coding**
- **Linear increase in accuracy** requires **exponential increase in energy**

Acknowledgements: This work is funded by the DARPA YFA grant, TSMC University Shuttle Program, MIT Center for Integrated Circuits & Systems, and gifts from Intel and Texas Instruments.