DNN Accelerator Architectures

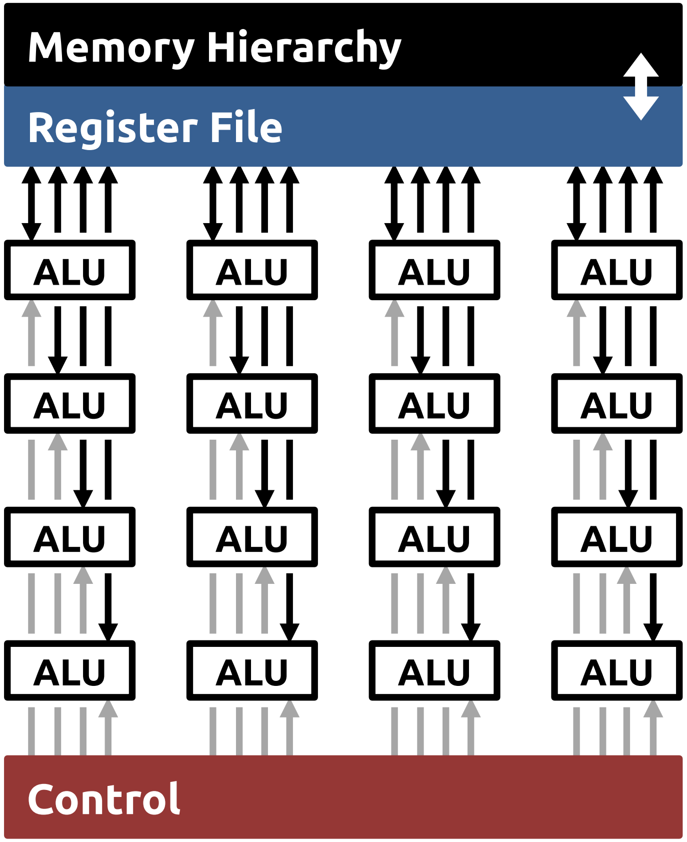
MICRO Tutorial (2016)

Website: http://eyeriss.mit.edu/tutorial.html

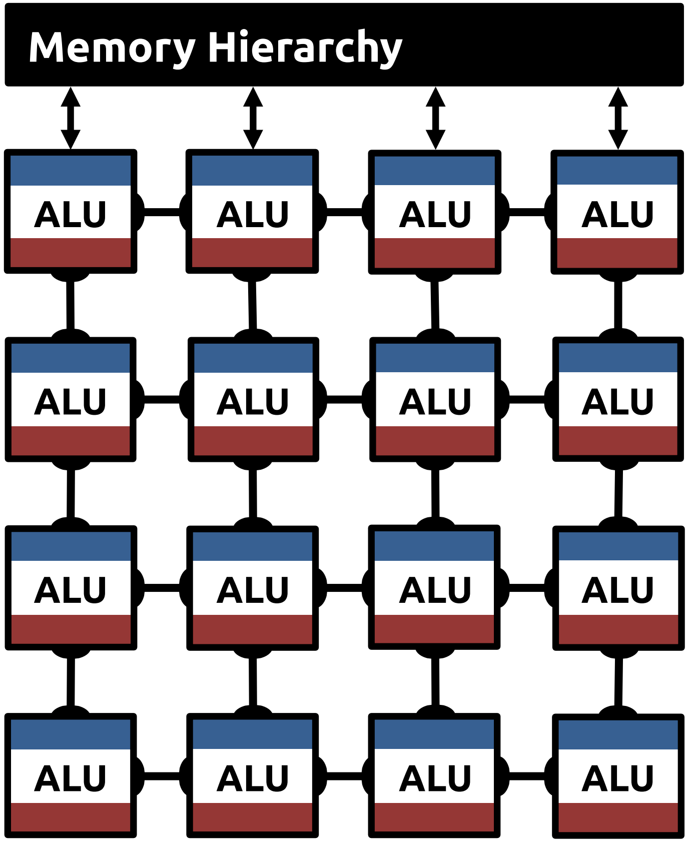
Joel Emer, Vivienne Sze, Yu-Hsin Chen

Highly-Parallel Compute Paradigms

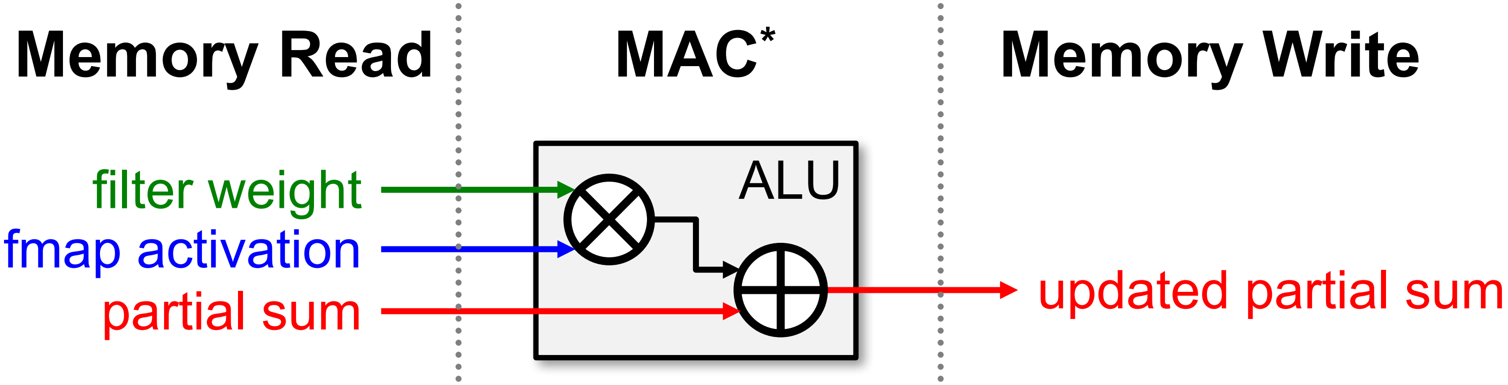
Temporal Architecture (SIMD/SIMT)



Spatial Architecture (Dataflow Processing)

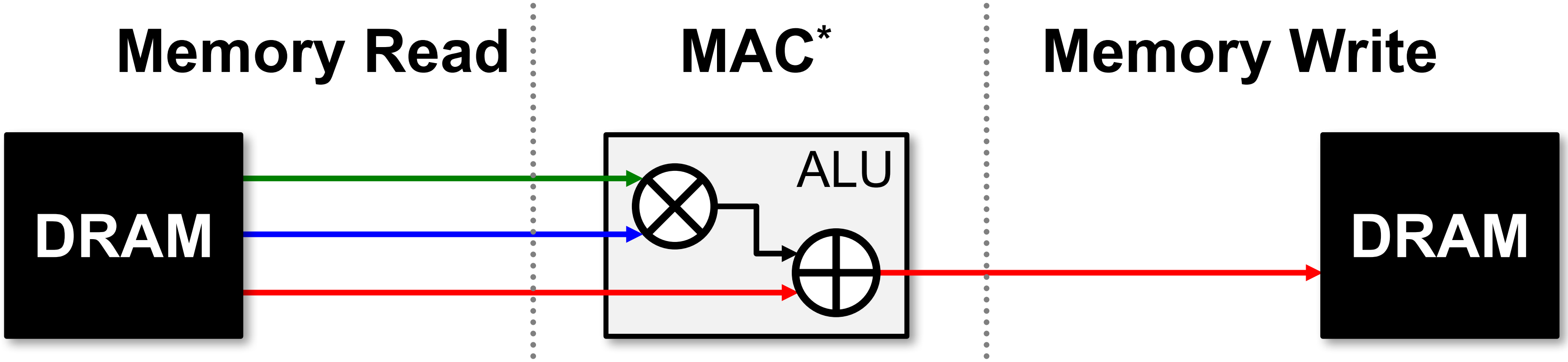


Memory Access is the Bottleneck



* multiply-and-accumulate

Memory Access is the Bottleneck

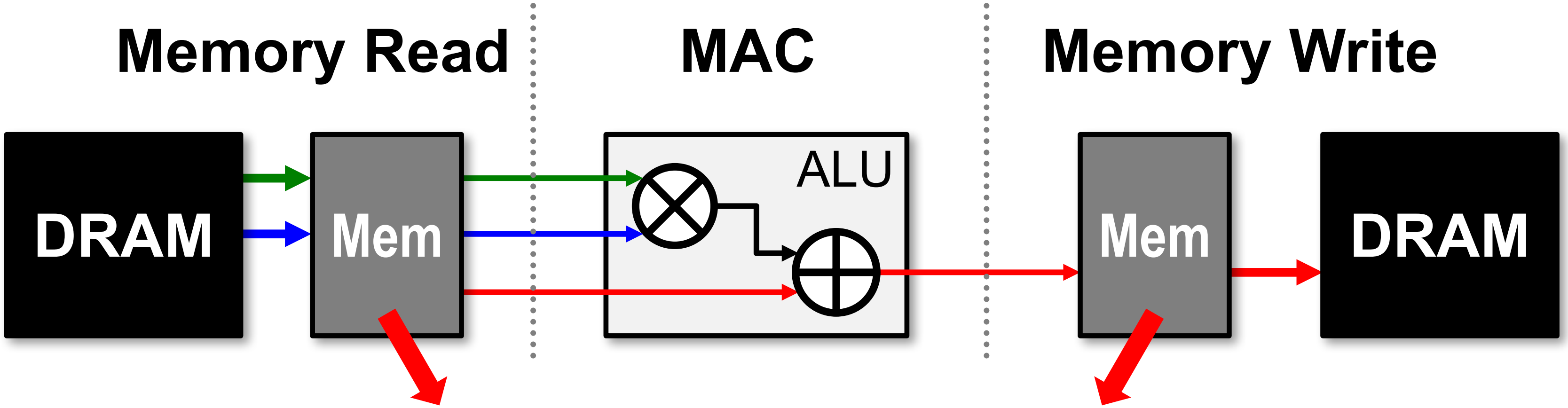


* multiply-and-accumulate

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

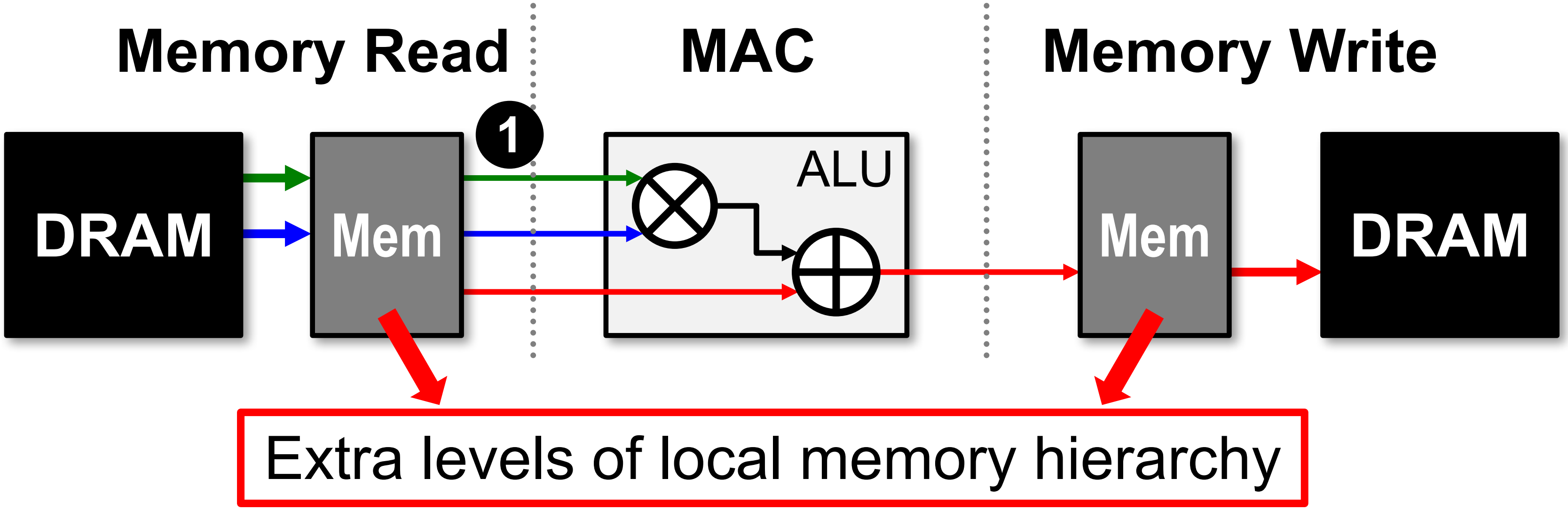
- Example: AlexNet [NIPS 2012] has **724M** MACs
  → **2896M** DRAM accesses required

Memory Access is the Bottleneck



Extra levels of local memory hierarchy

Memory Access is the Bottleneck

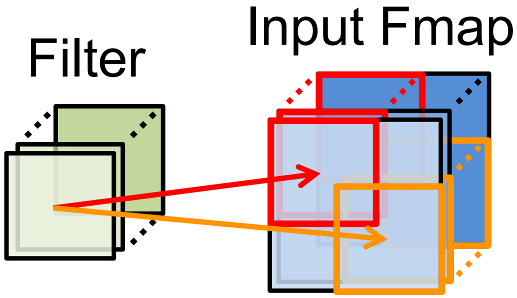


Opportunities: ❶ **data reuse**

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

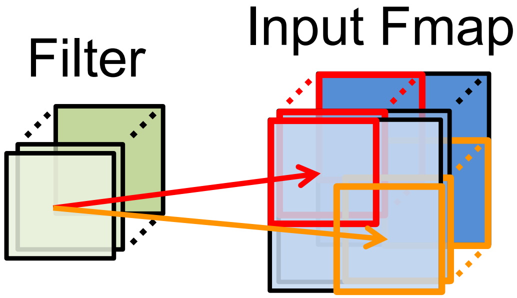


Reuse: Activations
Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

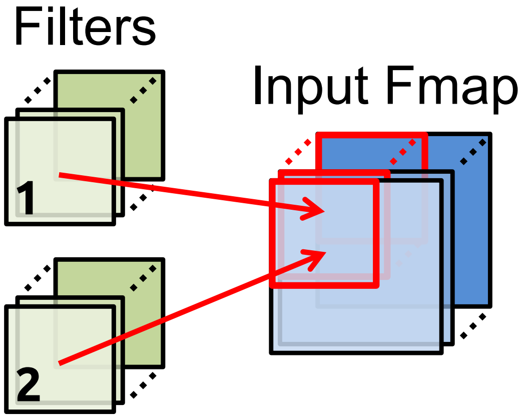
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

CONV and FC layers

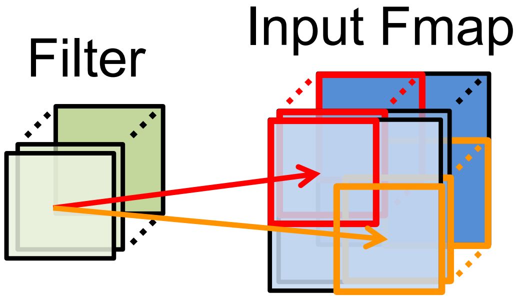


Reuse: Activations

Types of Data Reuse in DNN

Convolutional Reuse

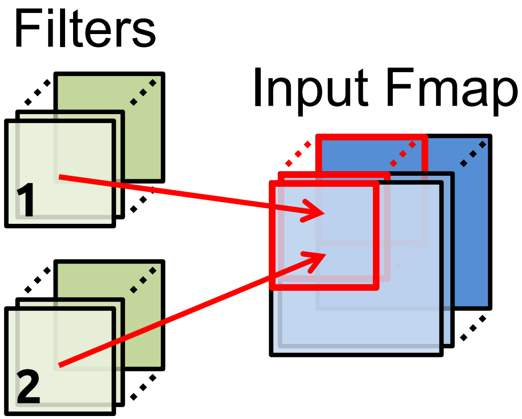
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

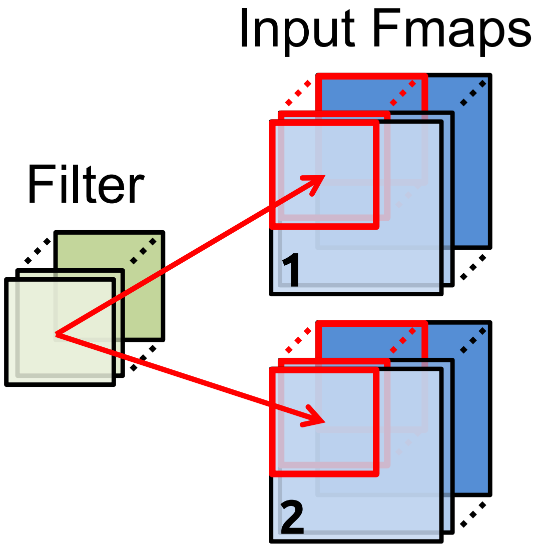
CONV and FC layers



Reuse: Activations

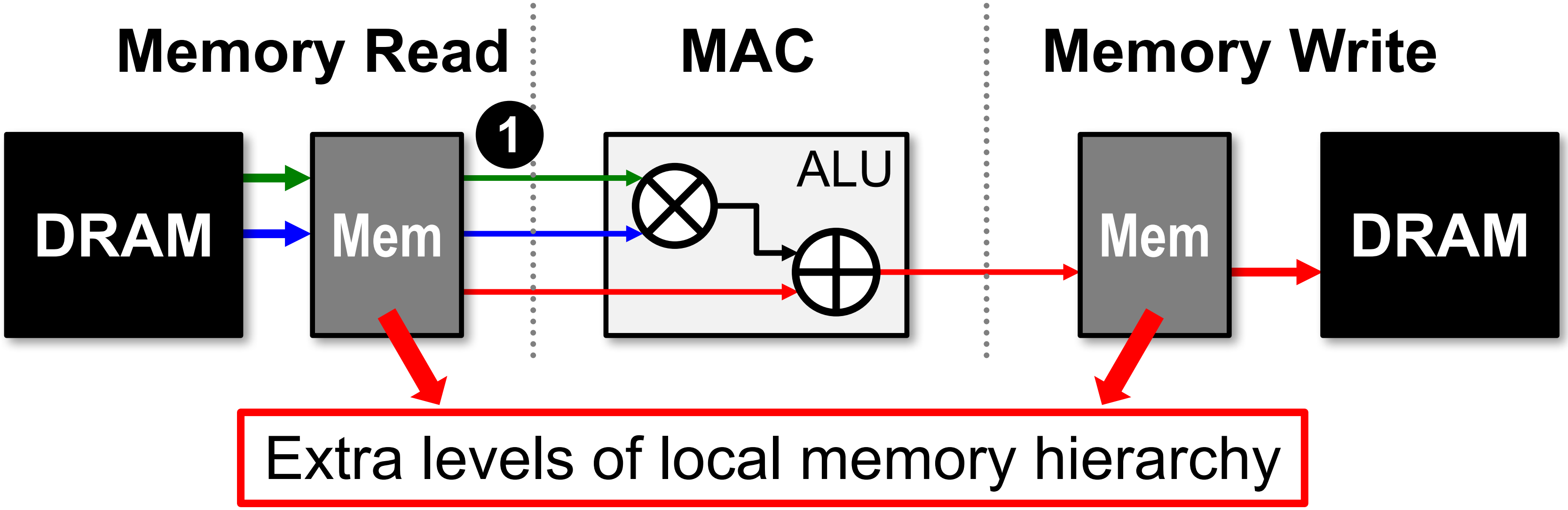
Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Memory Access is the Bottleneck

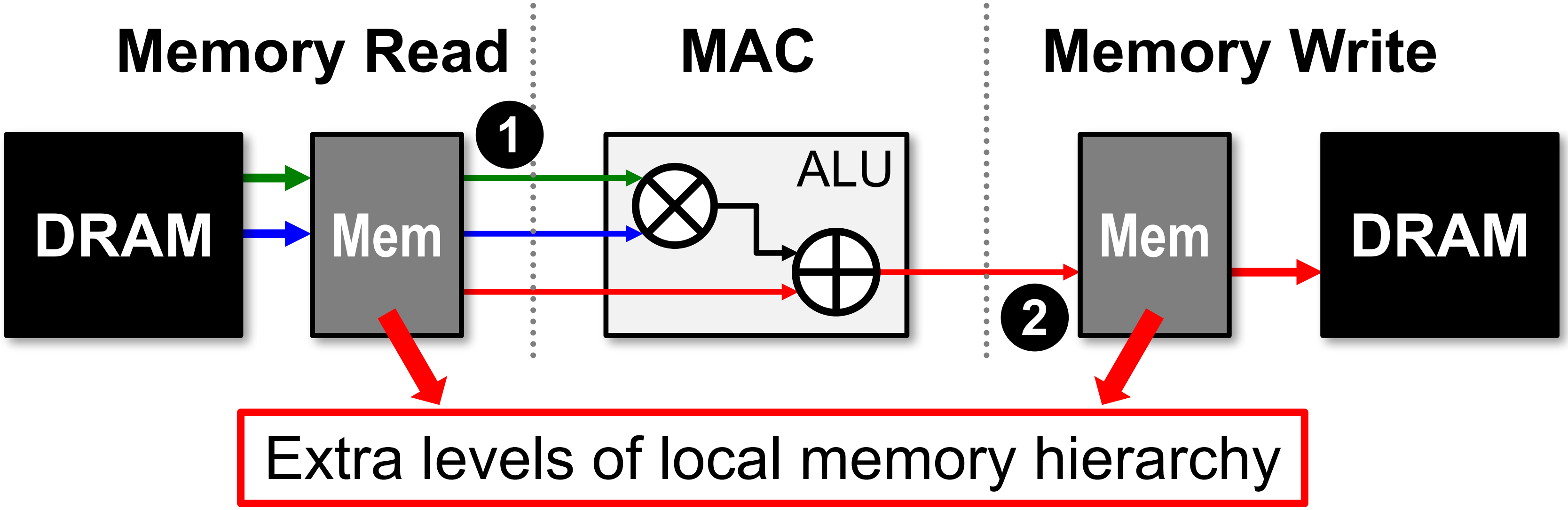


Opportunities: ❶ **data reuse**

❶ Can reduce DRAM reads of filter/fmap by up to **500×**[**]

[**] AlexNet CONV layers

Memory Access is the Bottleneck

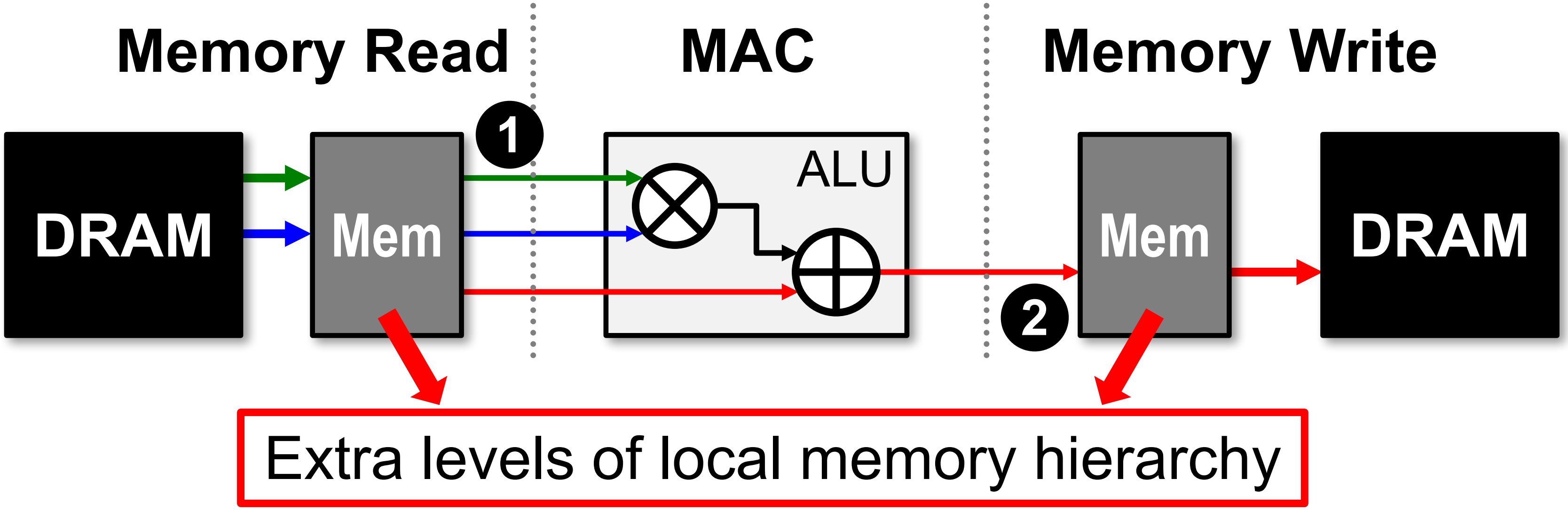


Opportunities: 1 **data reuse** 2 **local accumulation**

1 Can reduce DRAM reads of filter/fmap by up to **500×**

2 Partial sum accumulation does **NOT** have to access DRAM

Memory Access is the Bottleneck



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

❶ Can reduce DRAM reads of filter/fmap by up to **500×**

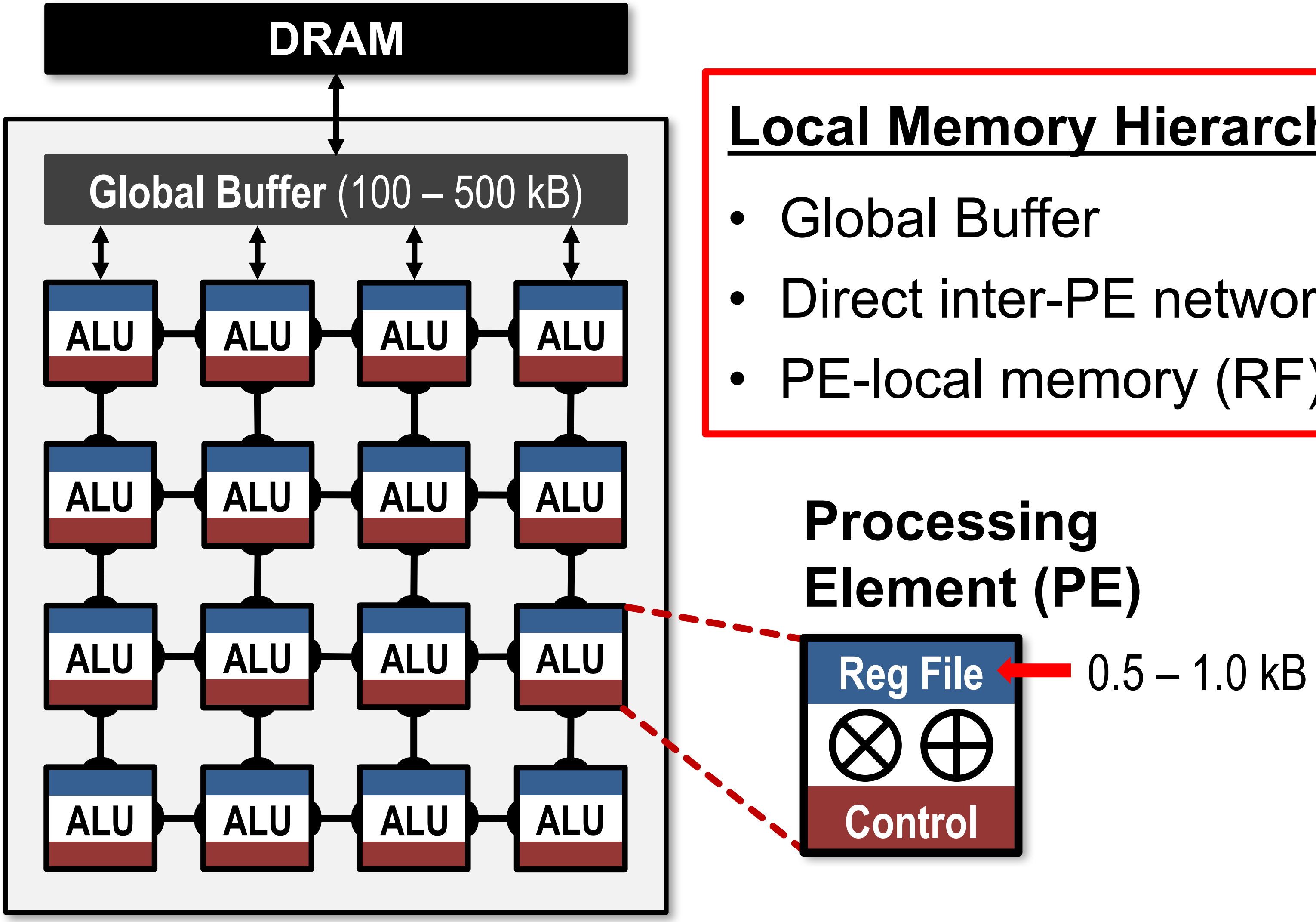
❷ Partial sum accumulation does **NOT** have to access DRAM

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

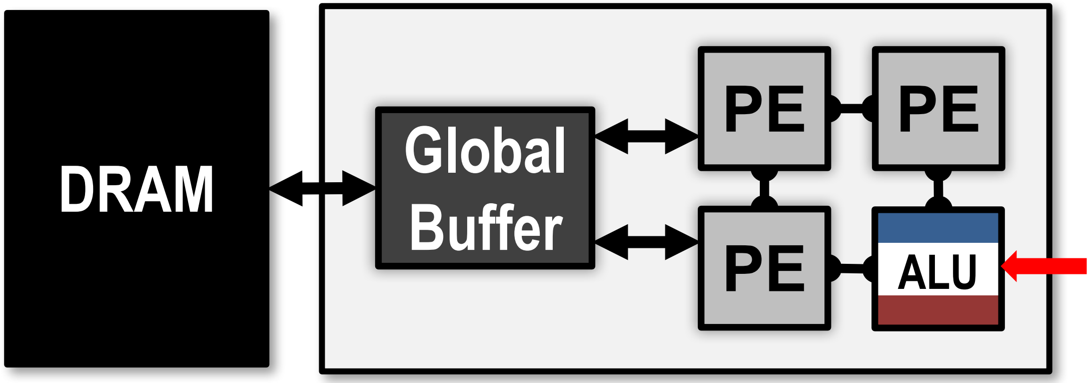
Spatial Architecture for CNN

Local Memory Hierarchy

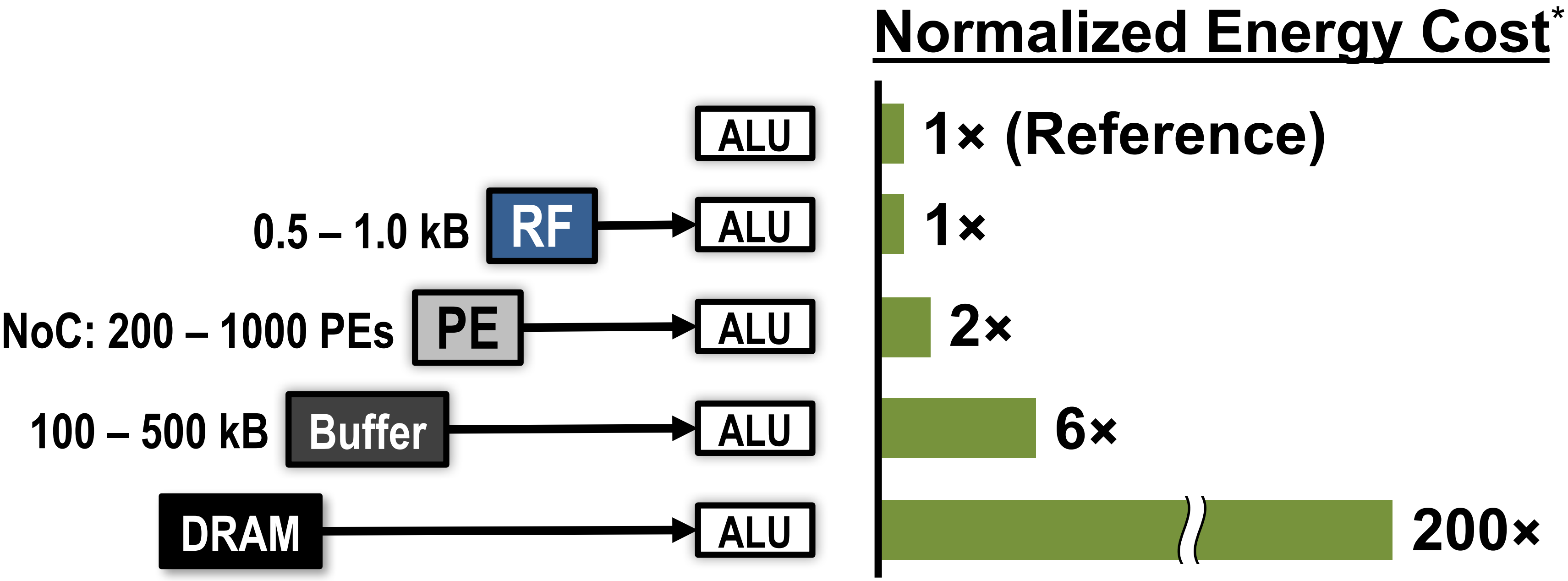
- Global Buffer
- Direct inter-PE network
- PE-local memory (RF)



Low-Cost Local Data Access



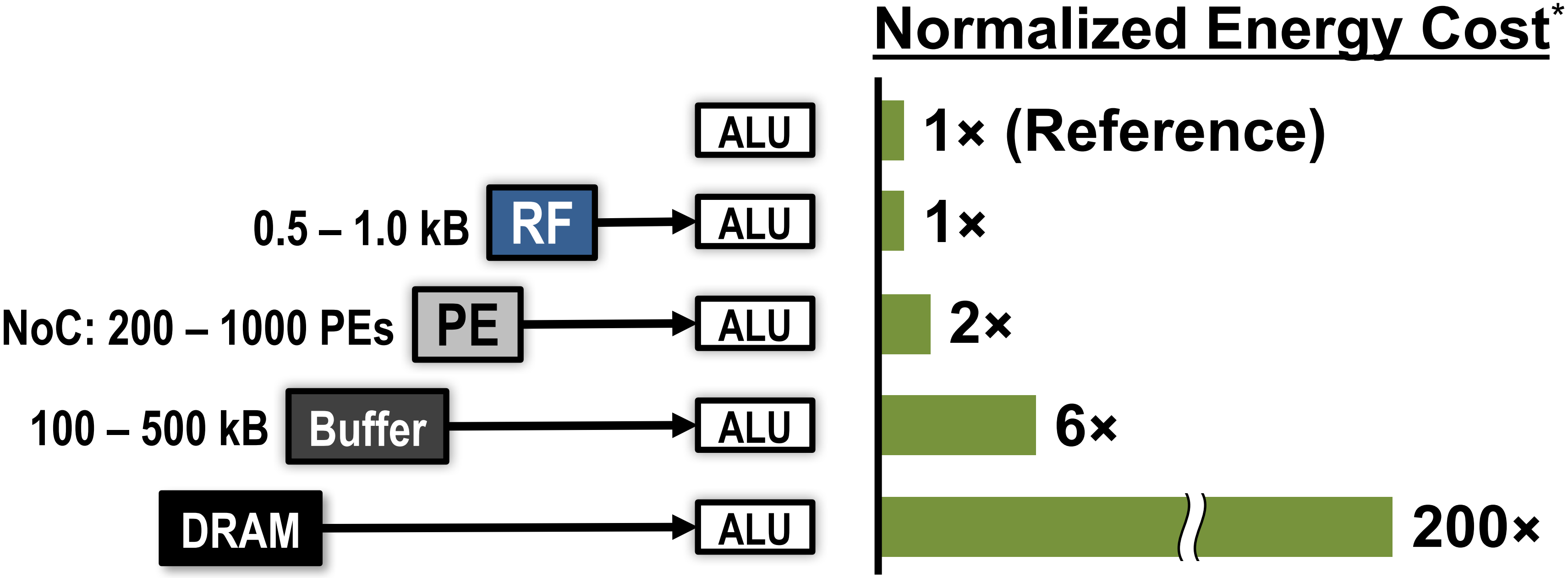
fetch data to run a MAC here



* measured from a commercial 65nm process

Low-Cost Local Data Access

How to exploit ❶ **data reuse** and ❷ **local accumulation** with *limited* low-cost local storage?

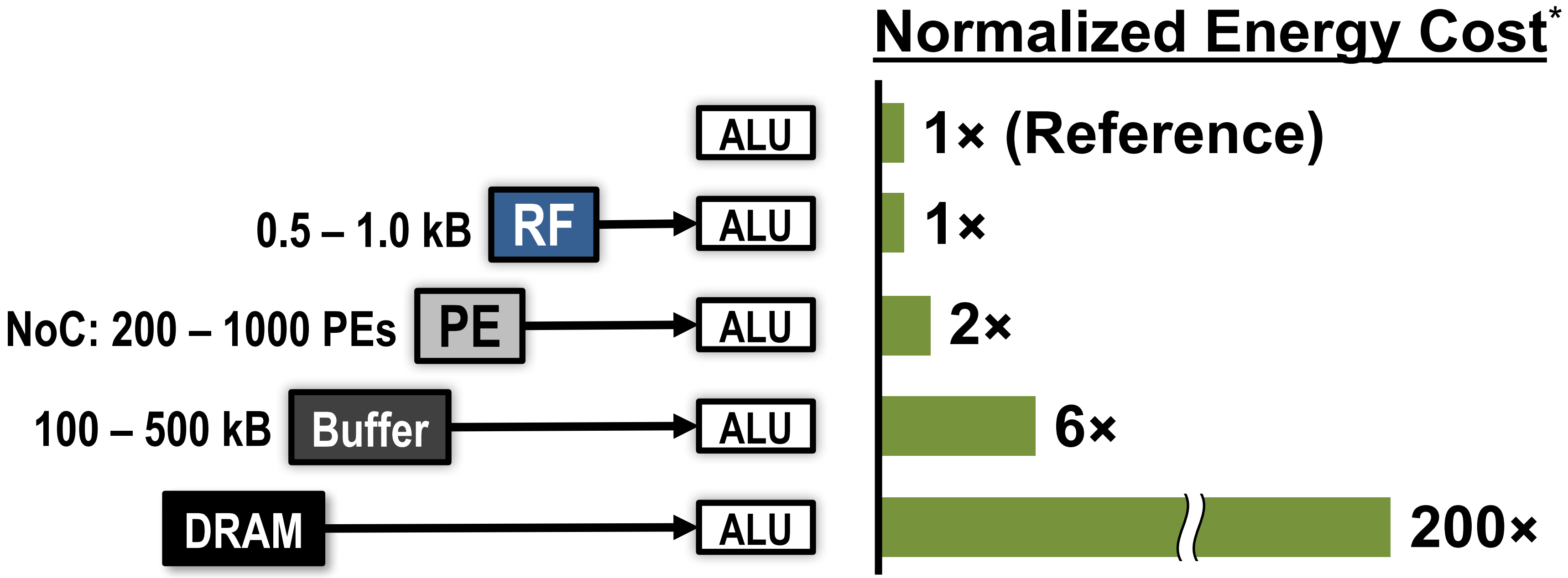


* measured from a commercial 65nm process

Low-Cost Local Data Access

How to exploit ❶ **data reuse** and ❷ **local accumulation** with *limited* low-cost local storage?

specialized **processing dataflow** required!



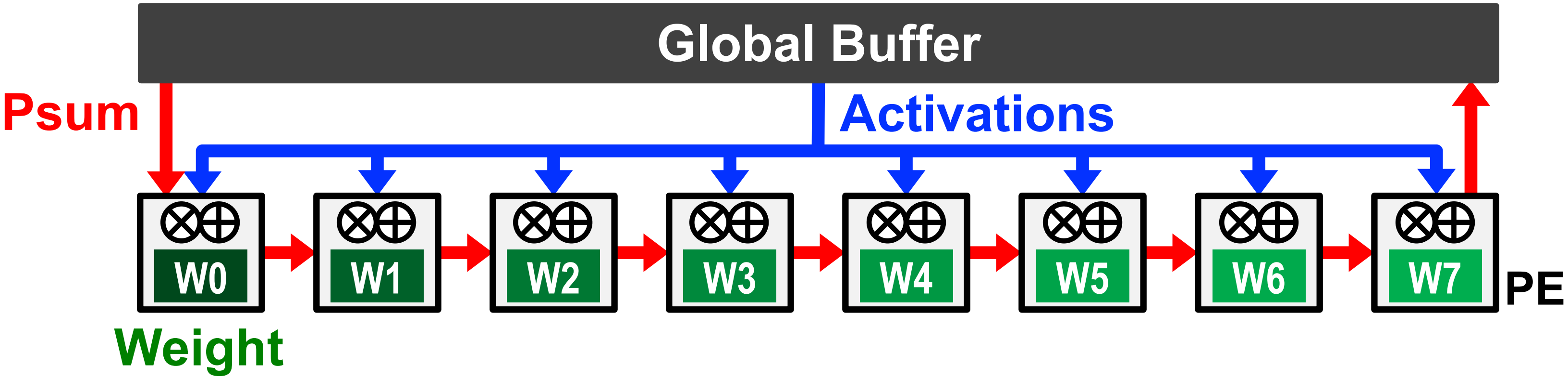
* measured from a commercial 65nm process

Dataflow Taxonomy

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

[Chen et al., ISCA 2016]

Weight Stationary (WS)

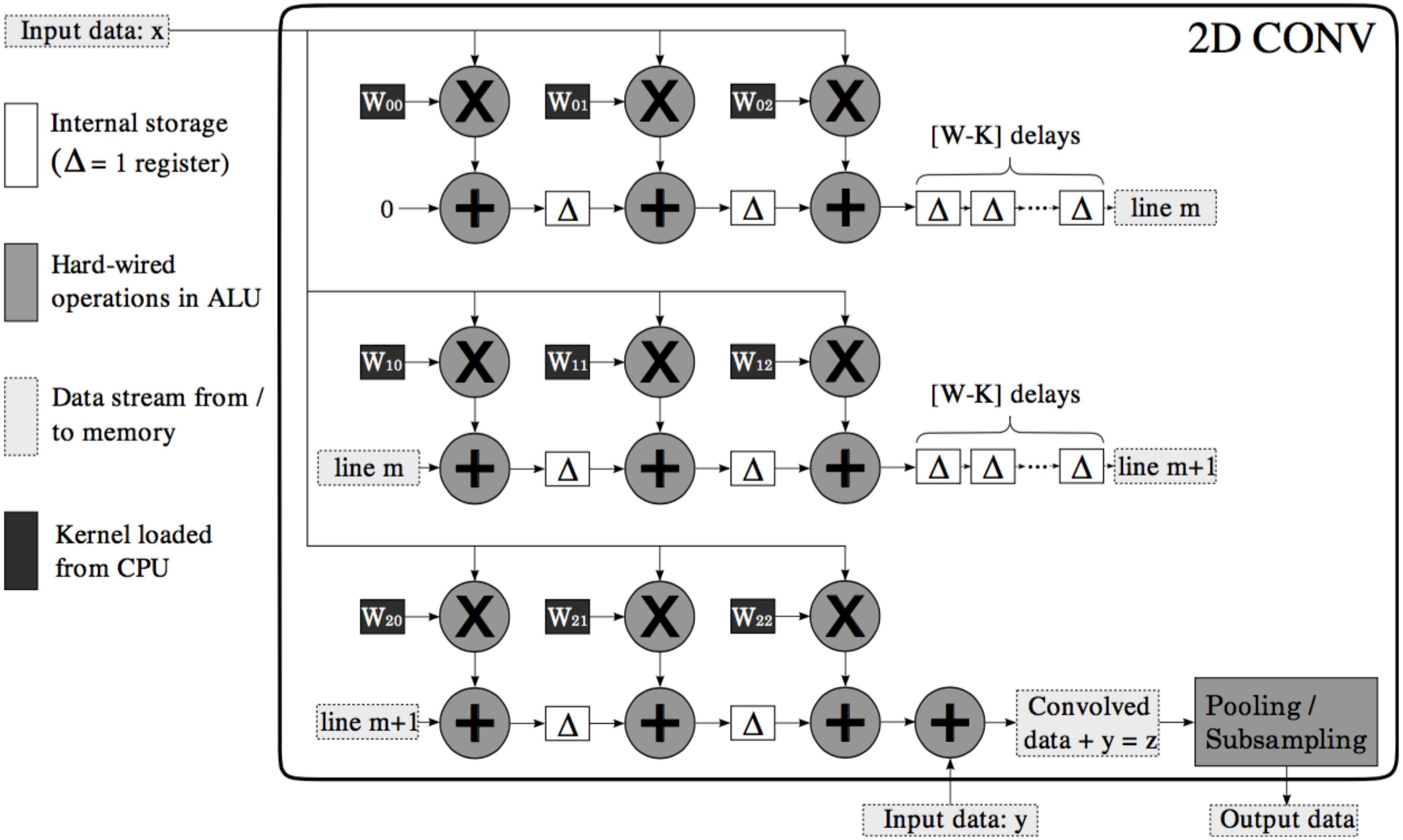


- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights

- **Broadcast activations** and **accumulate psums spatially** across the PE array.

WS Example: nn-X (NeuFlow)

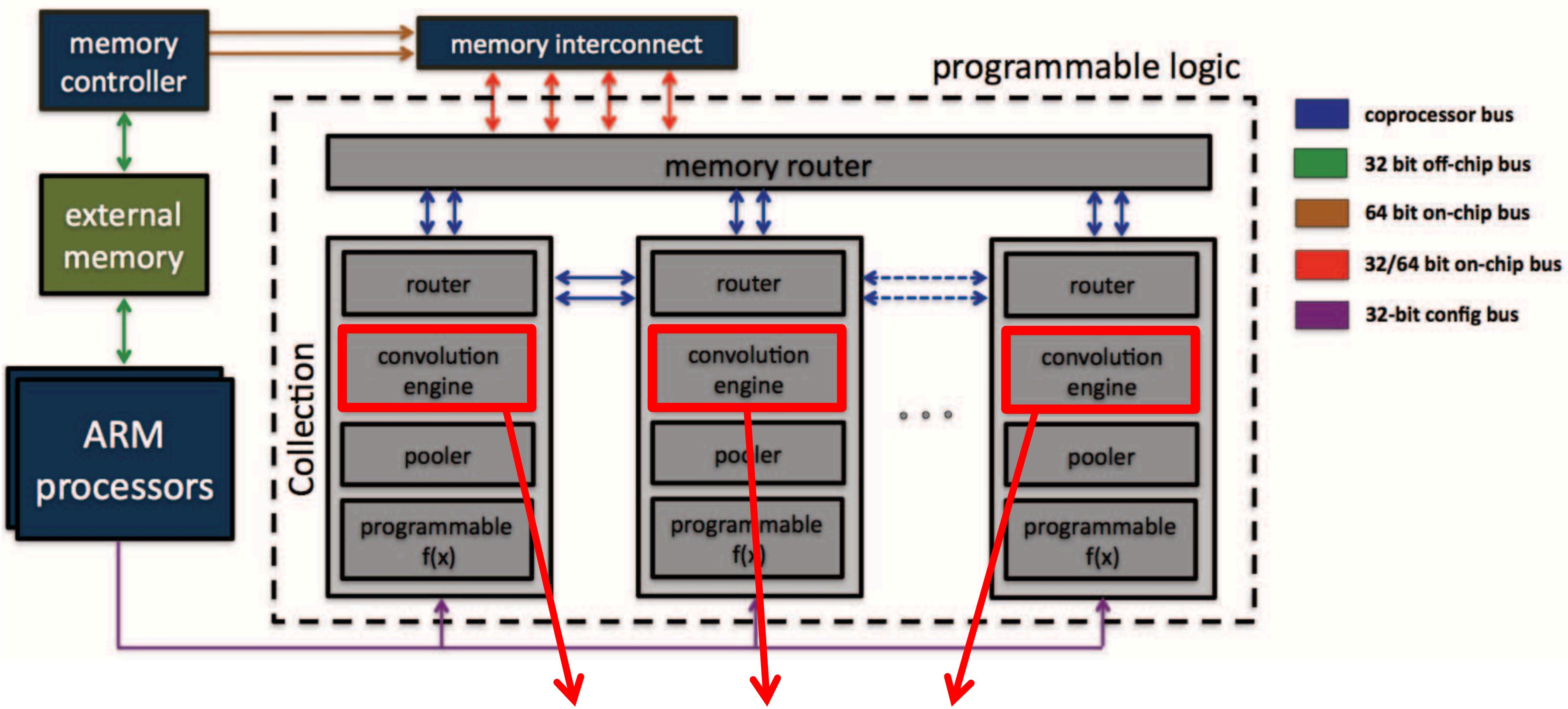
A 3×3 2D Convolution Engine



[Farabet et al., ICCV 2009]

WS Example: nn-X (NeuFlow)

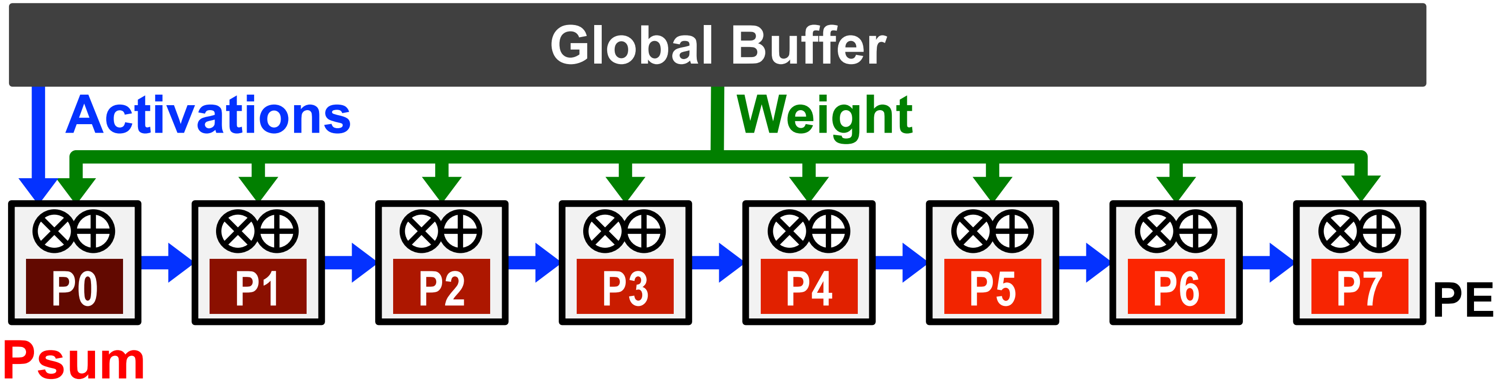
Top-Level Architecture



10×10 2D Convolution Engines

[Gokhale et al., CVPRW 2014]

Output Stationary (OS)

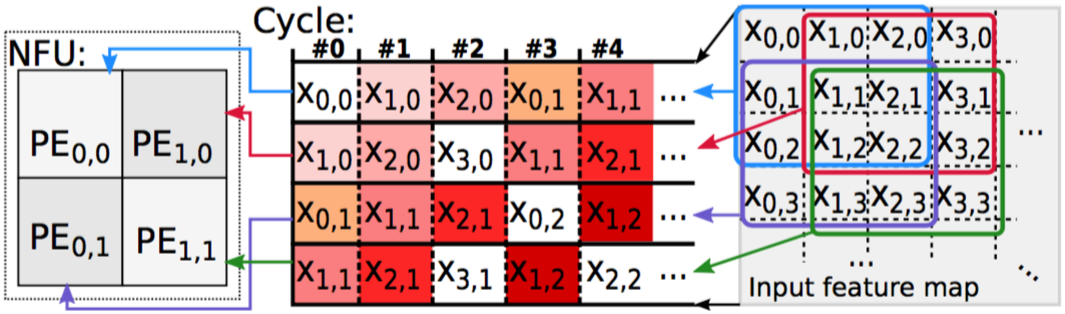


- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation

- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array

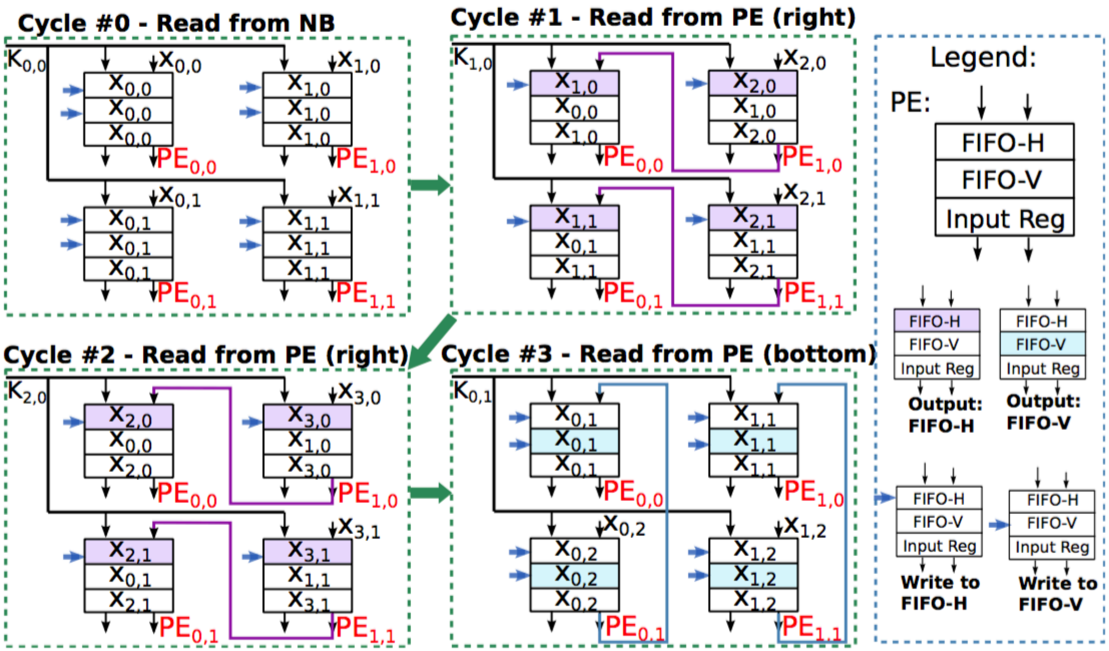
OS Example: ShiDianNao

Input Fmap Dataflow in the PE Array



Cycle: #0	#1	#2	#3	#4	
$x_{0,0}$	$x_{1,0}$	$x_{2,0}$	$x_{0,1}$	$x_{1,1}$	…
$x_{1,0}$	$x_{2,0}$	$x_{3,0}$	$x_{1,1}$	$x_{2,1}$	…
$x_{0,1}$	$x_{1,1}$	$x_{2,1}$	$x_{0,2}$	$x_{1,2}$	…
$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	$x_{1,2}$	$x_{2,2}$	…

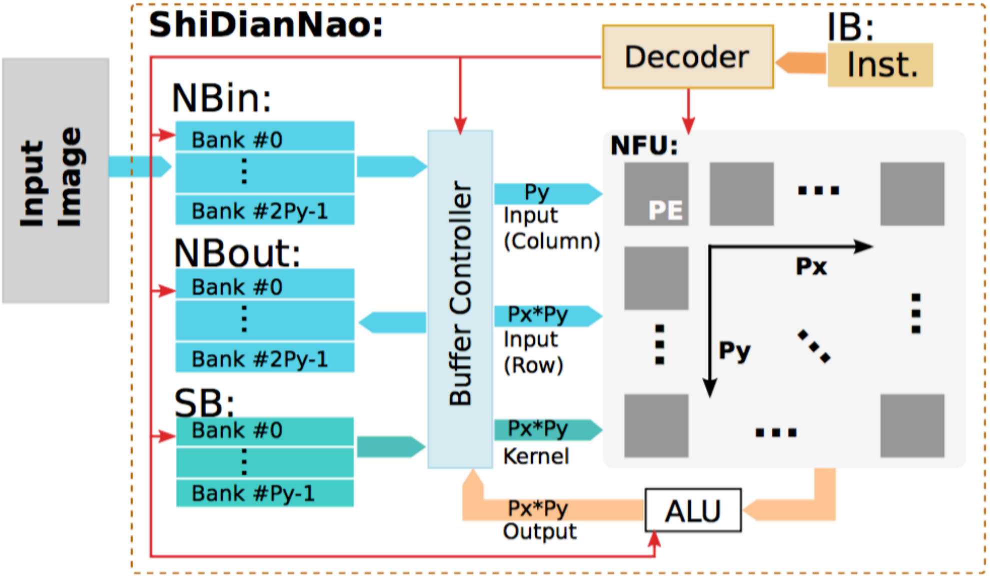
$x_{0,0}$	$x_{1,0}$	$x_{2,0}$	$x_{3,0}$	
$x_{0,1}$	$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	…
$x_{0,2}$	$x_{1,2}$	$x_{2,2}$	$x_{3,2}$	
$x_{0,3}$	$x_{1,3}$	$x_{2,3}$	$x_{3,3}$	
	…			⋱



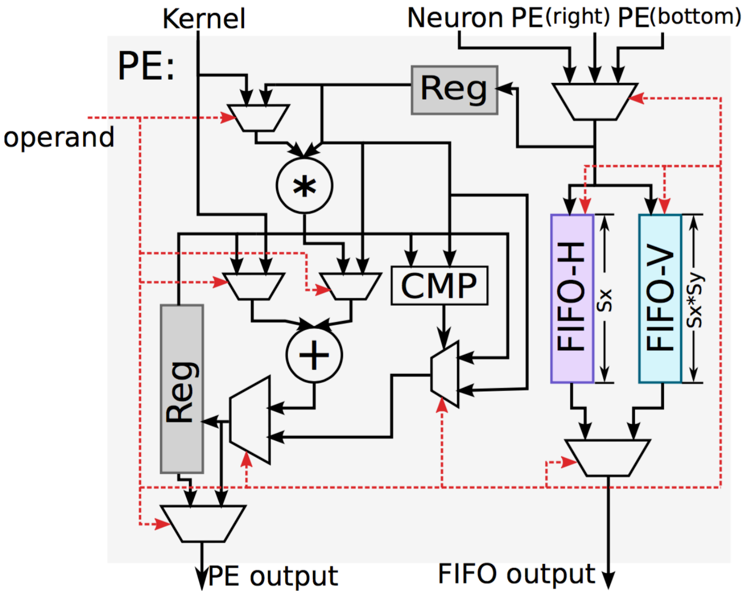
[Du et al., ISCA 2015]

OS Example: ShiDianNao

Top-Level Architecture

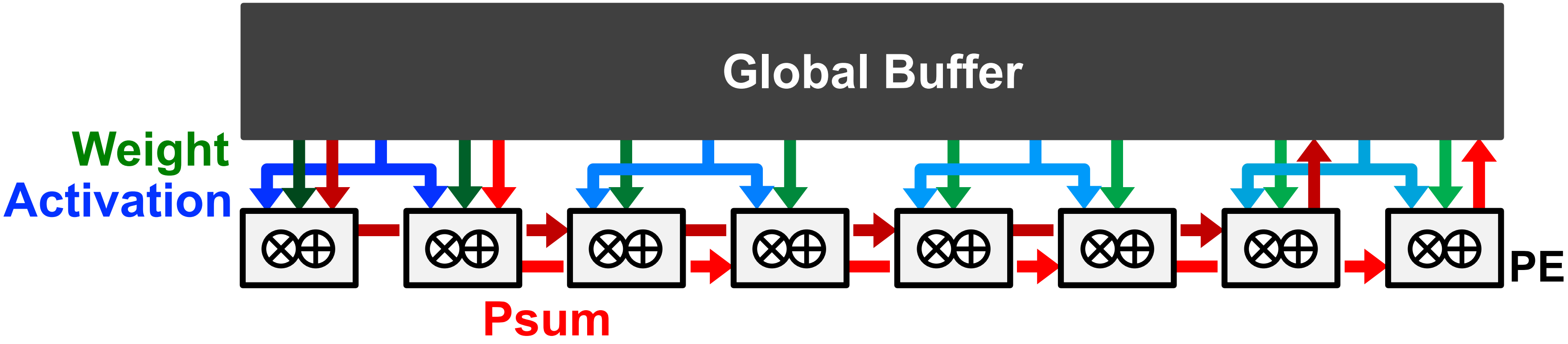


PE Architecture



[Du et al., ISCA 2015]

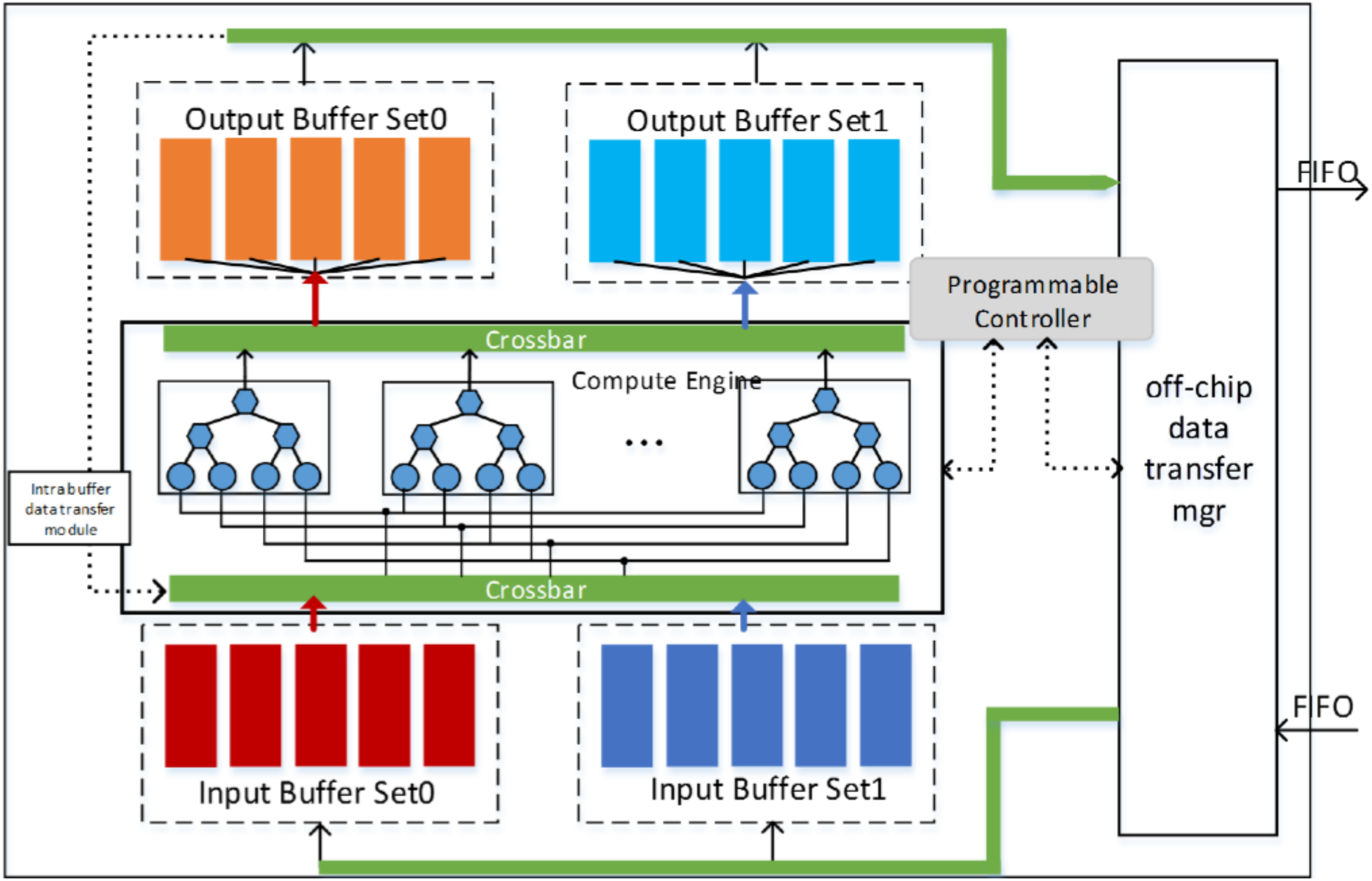
No Local Reuse (NLR)



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

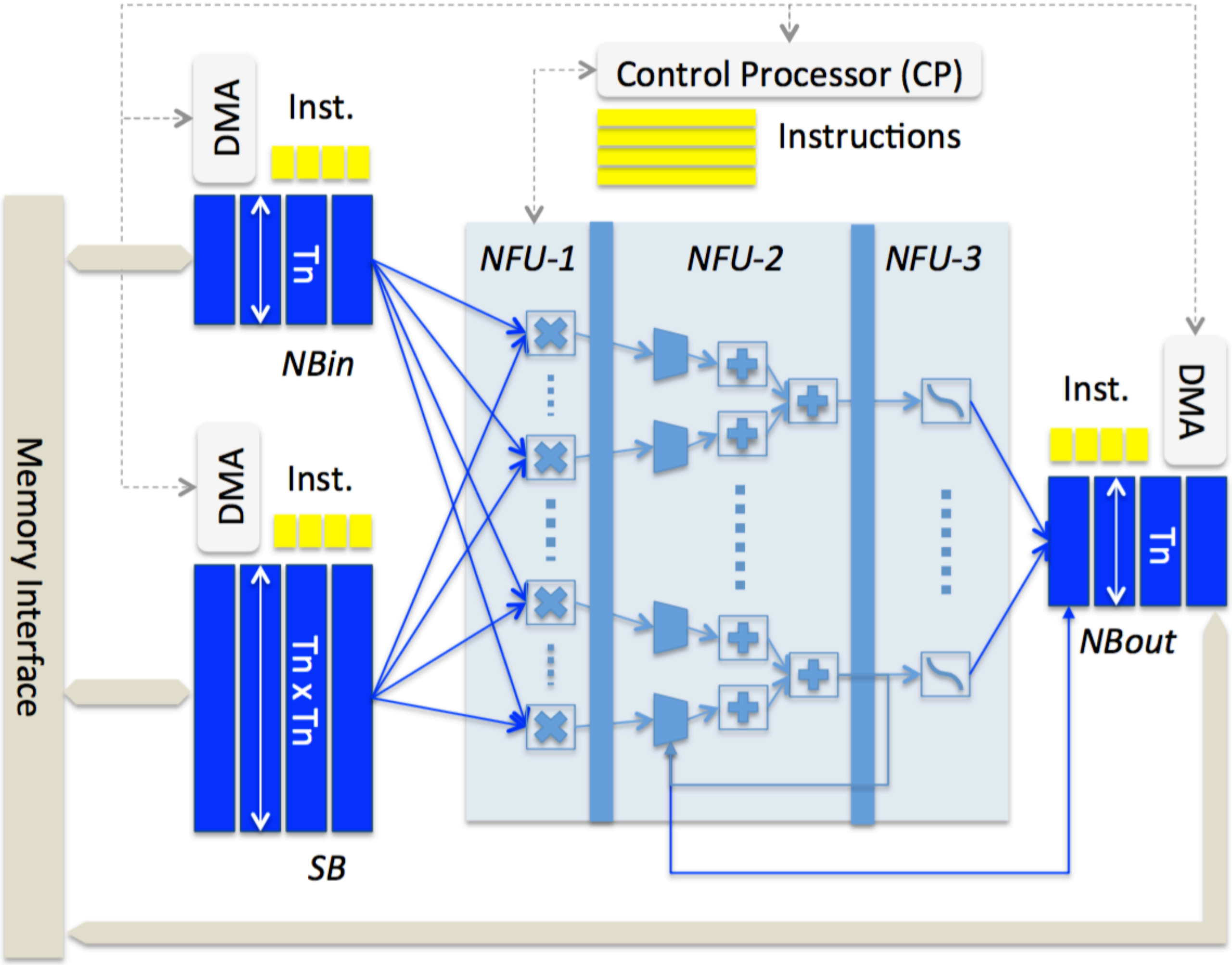
- **Multicast activations, single-cast weights, and accumulate psums spatially** across the PE array

NLR Example: UCLA



[Zhang et al., FPGA 2015]

NLR Example: DianNao



[Chen et al., ASPLOS 2014]

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)**

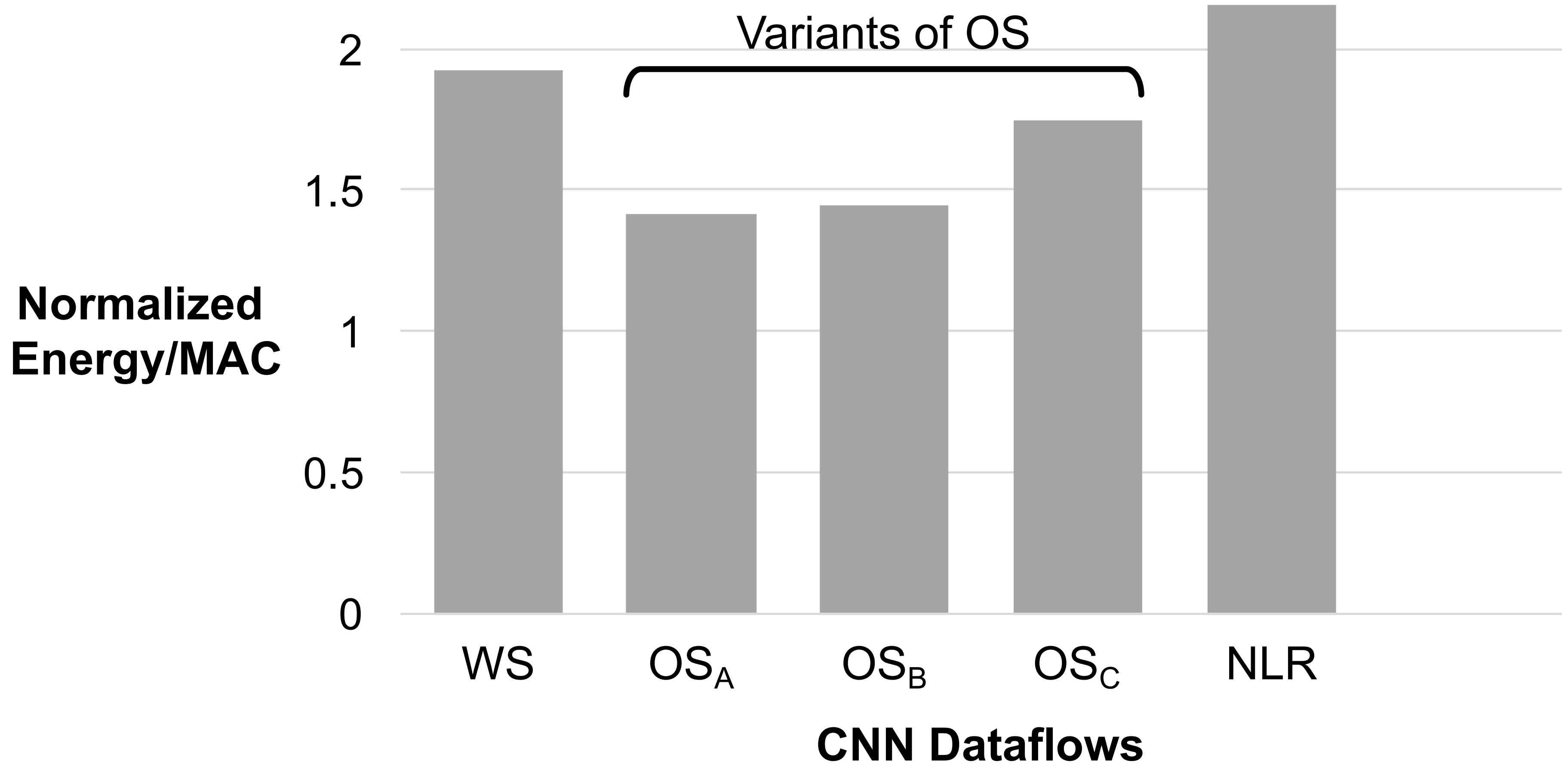
  [**Peemen**, *ICCD* 2013] [**ShiDianNao**, *ISCA* 2015]
  [**Gupta**, *ICML* 2015] [**Moons**, *VLSI* 2016]

- **No Local Reuse (NLR)**

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

Energy Efficiency Comparison

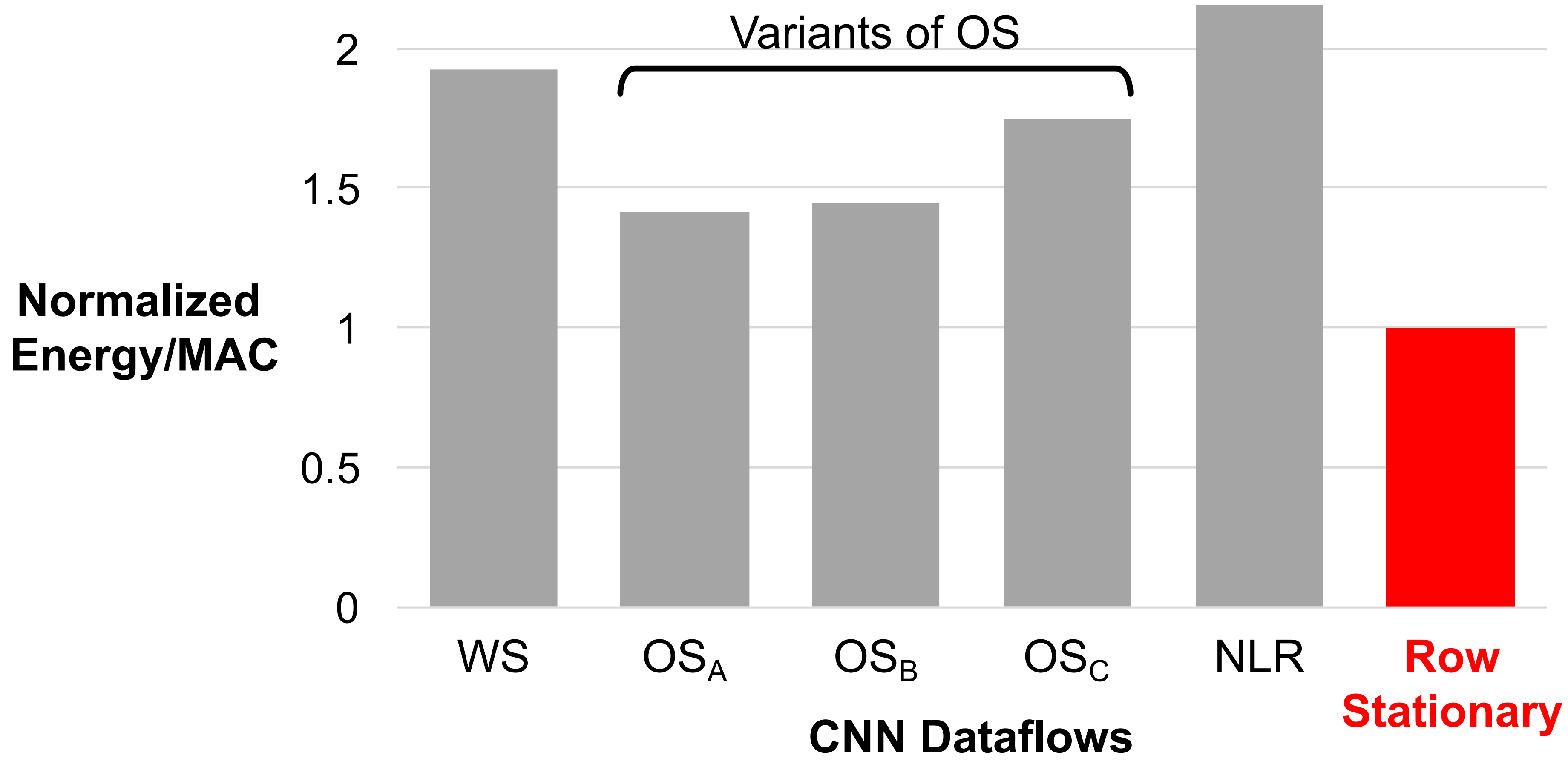
- Same total area
- AlexNet CONV layers
- 256 PEs
- Batch size = 16



[Chen et al., ISCA 2016]

Energy Efficiency Comparison

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



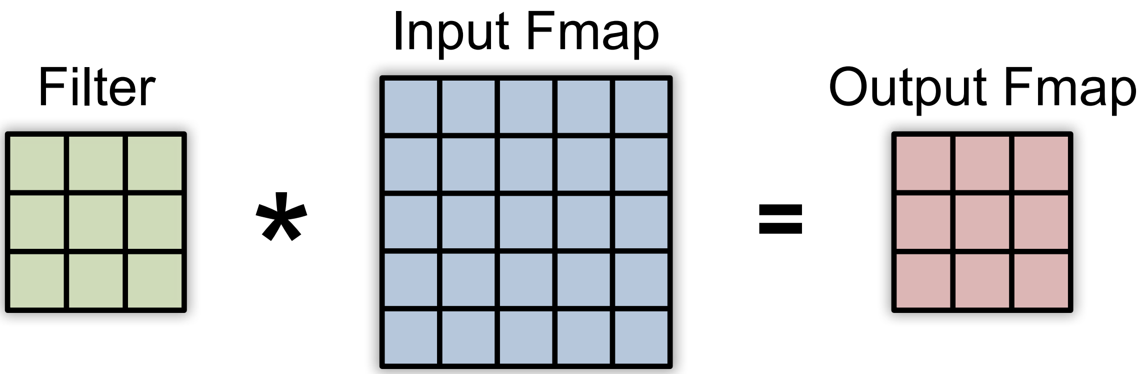
[Chen et al., ISCA 2016]

Energy-Efficient Dataflow:
Row Stationary (RS)

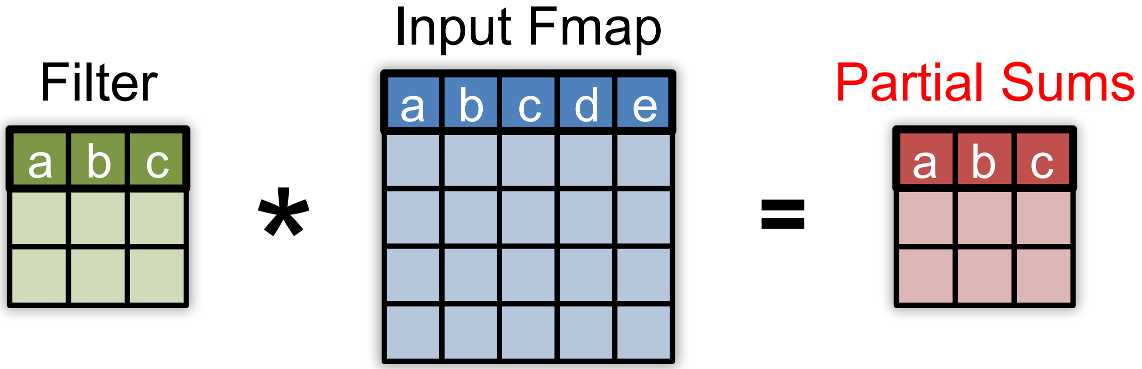
- **Maximize** reuse and accumulation at **RF**
- Optimize for **overall** energy efficiency instead for *only* a certain data type

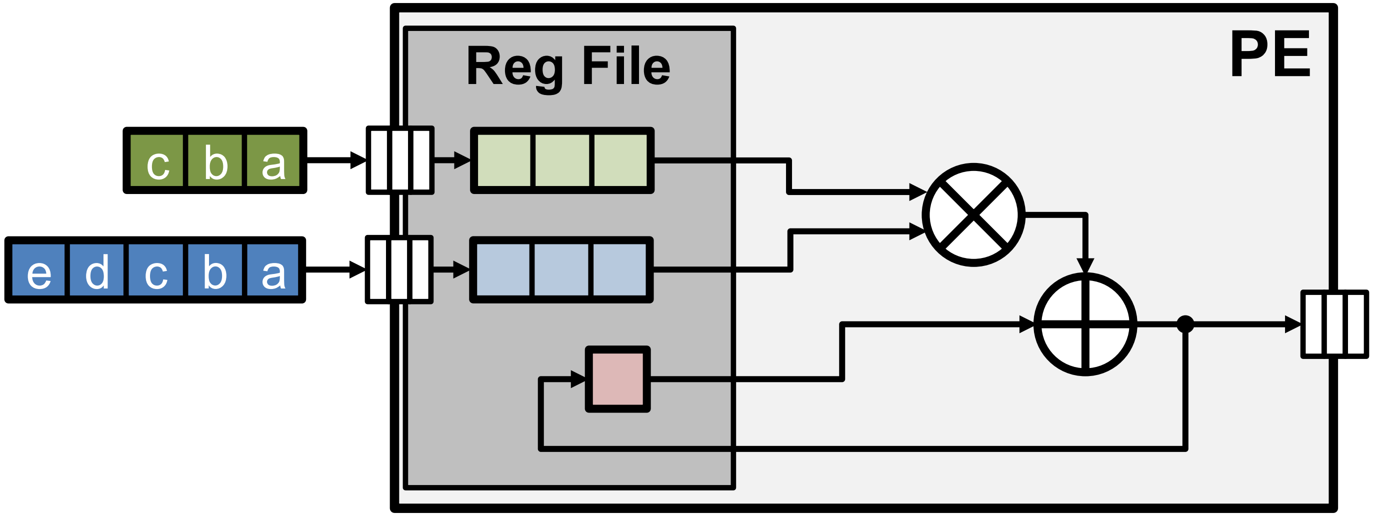
[Chen et al., ISCA 2016]

Row Stationary: Energy-efficient Dataflow

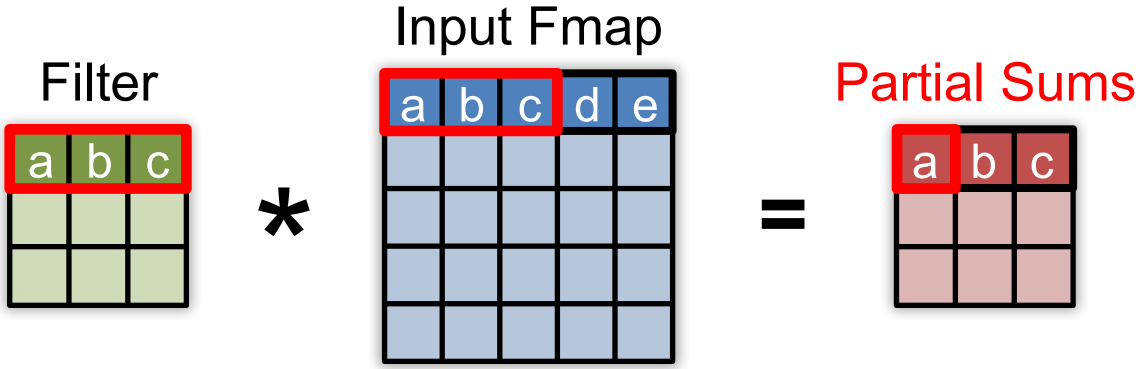


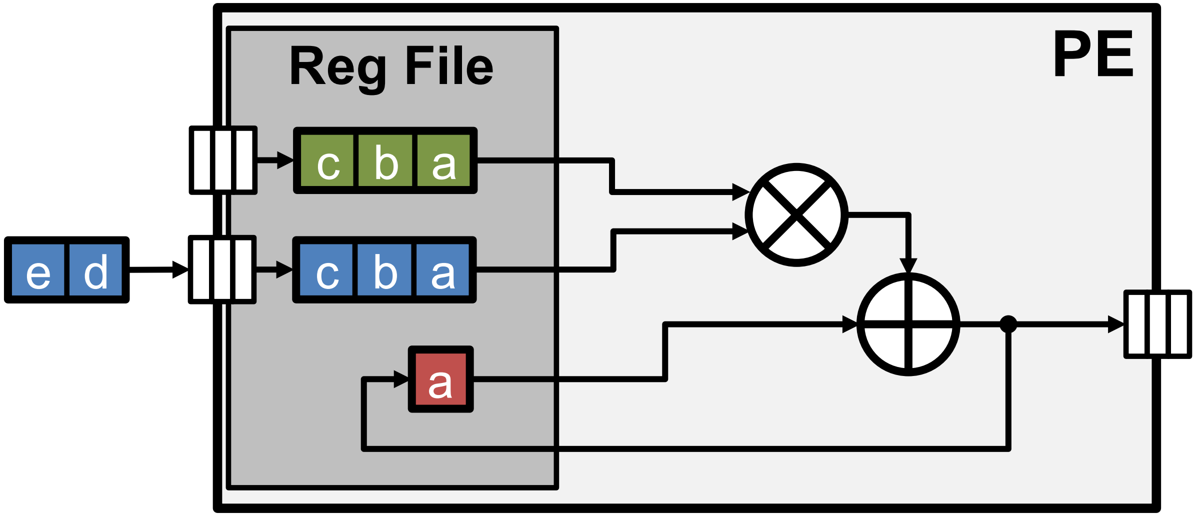
1D Row Convolution in PE



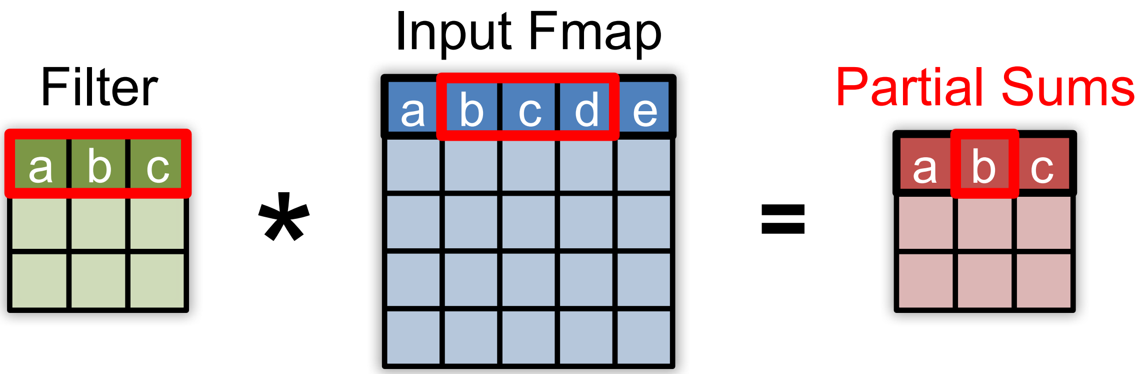


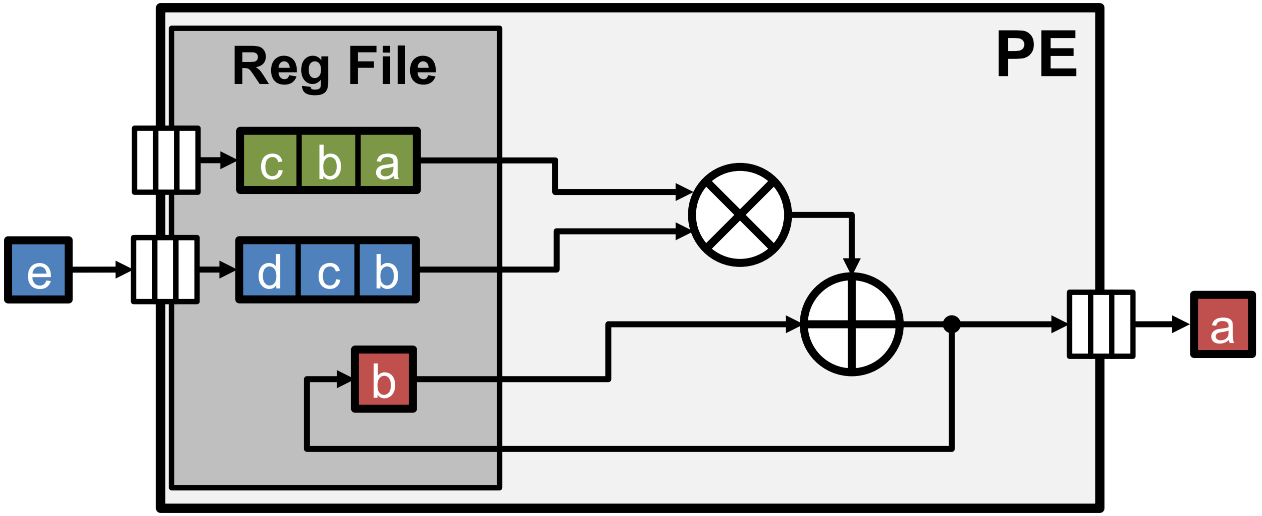
1D Row Convolution in PE



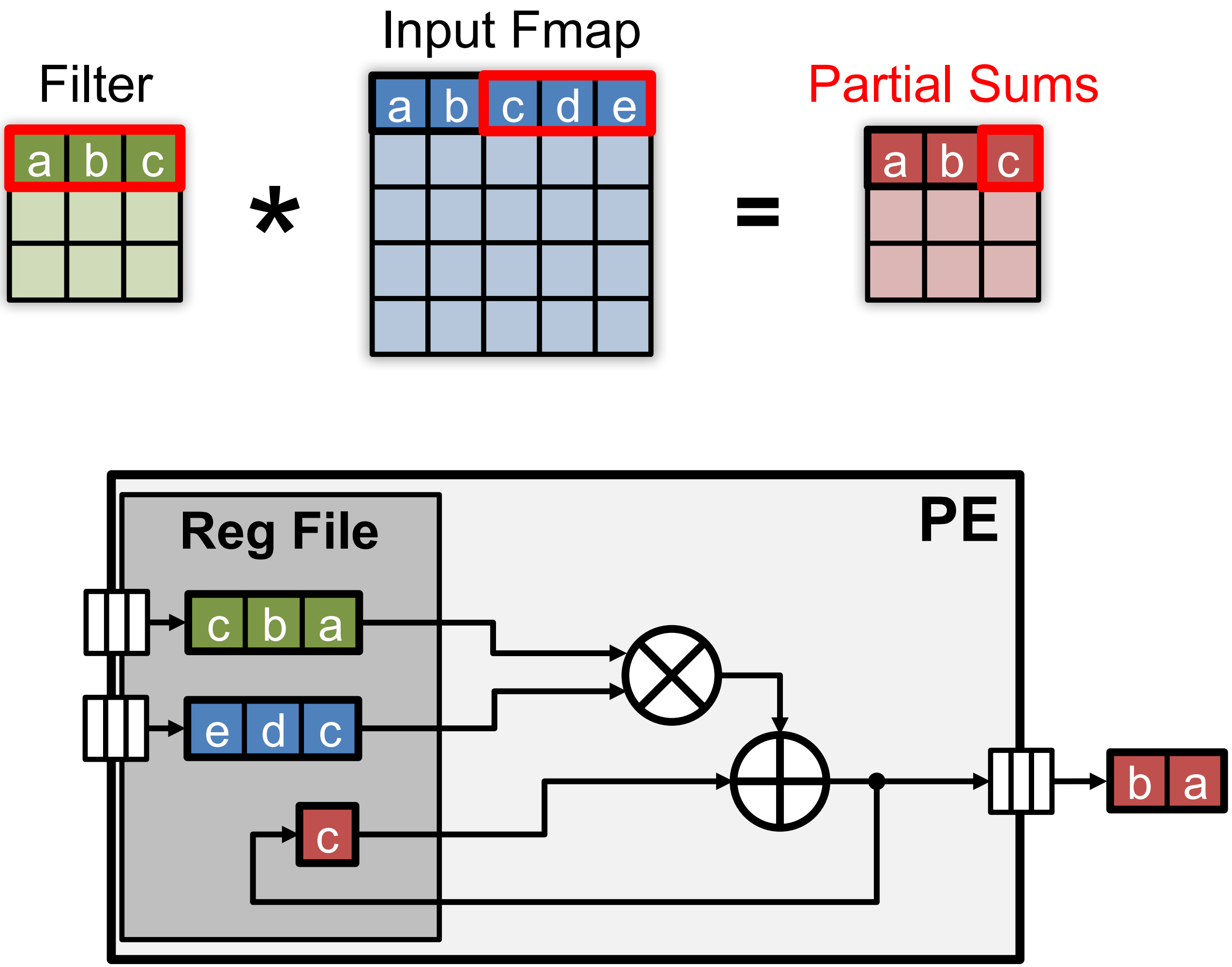


1D Row Convolution in PE



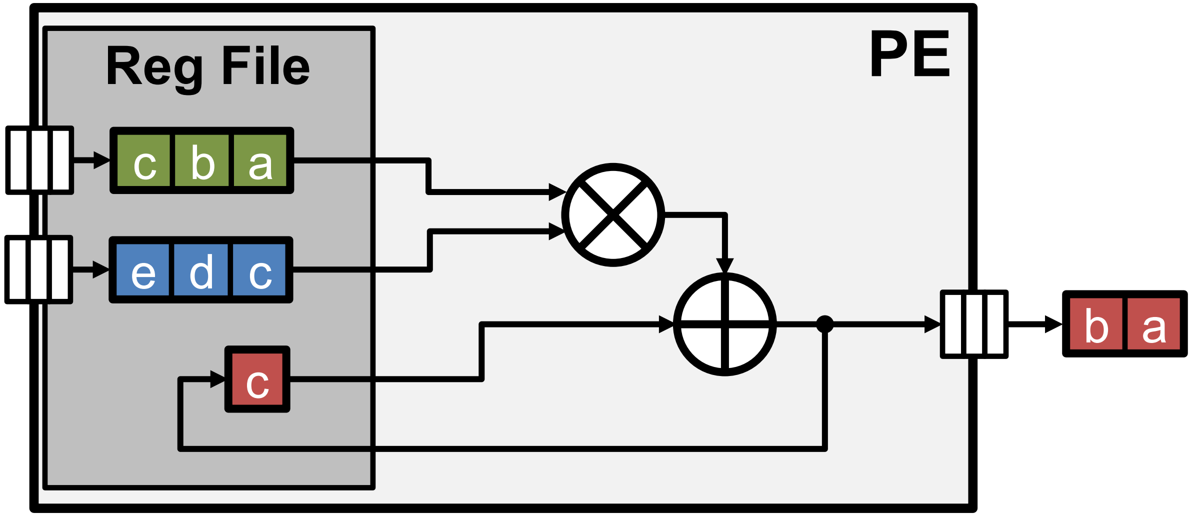


1D Row Convolution in PE

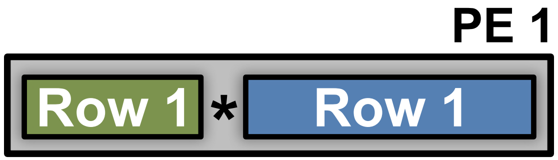


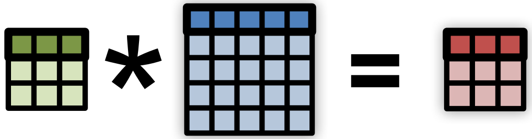
1D Row Convolution in PE

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

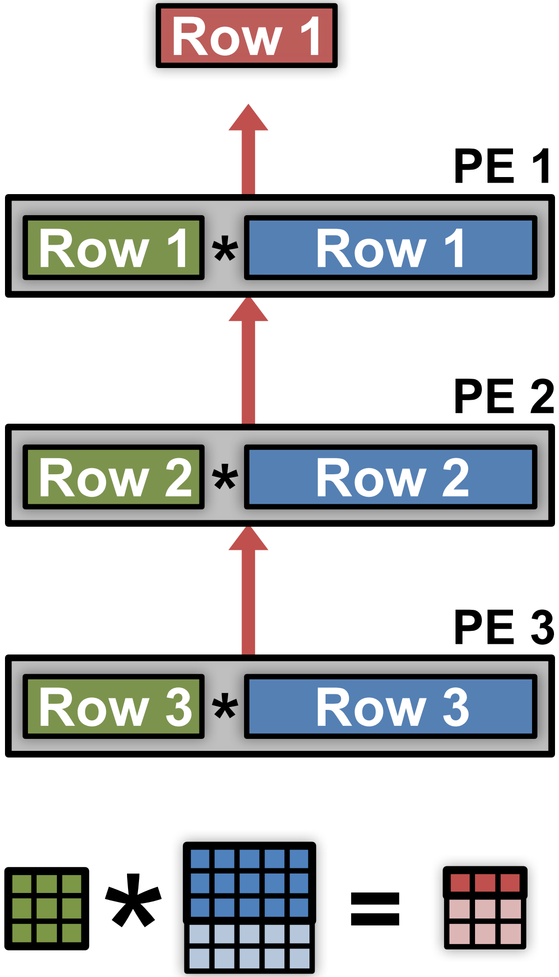


2D Convolution in PE Array

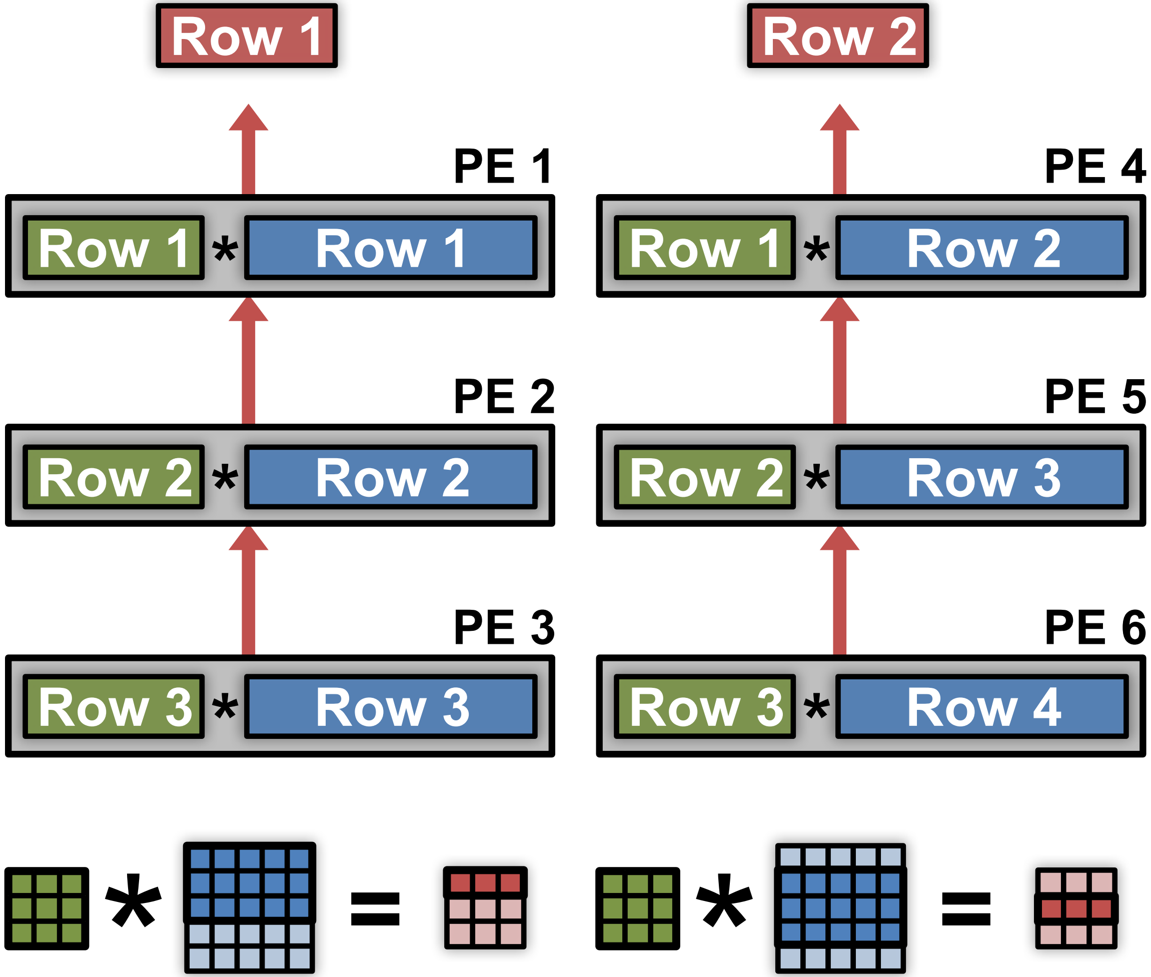




2D Convolution in PE Array



2D Convolution in PE Array



Row 1

Row 2

PE 1

Row 1 * Row 1

PE 4

Row 1 * Row 2

PE 2

Row 2 * Row 2

PE 5

Row 2 * Row 3

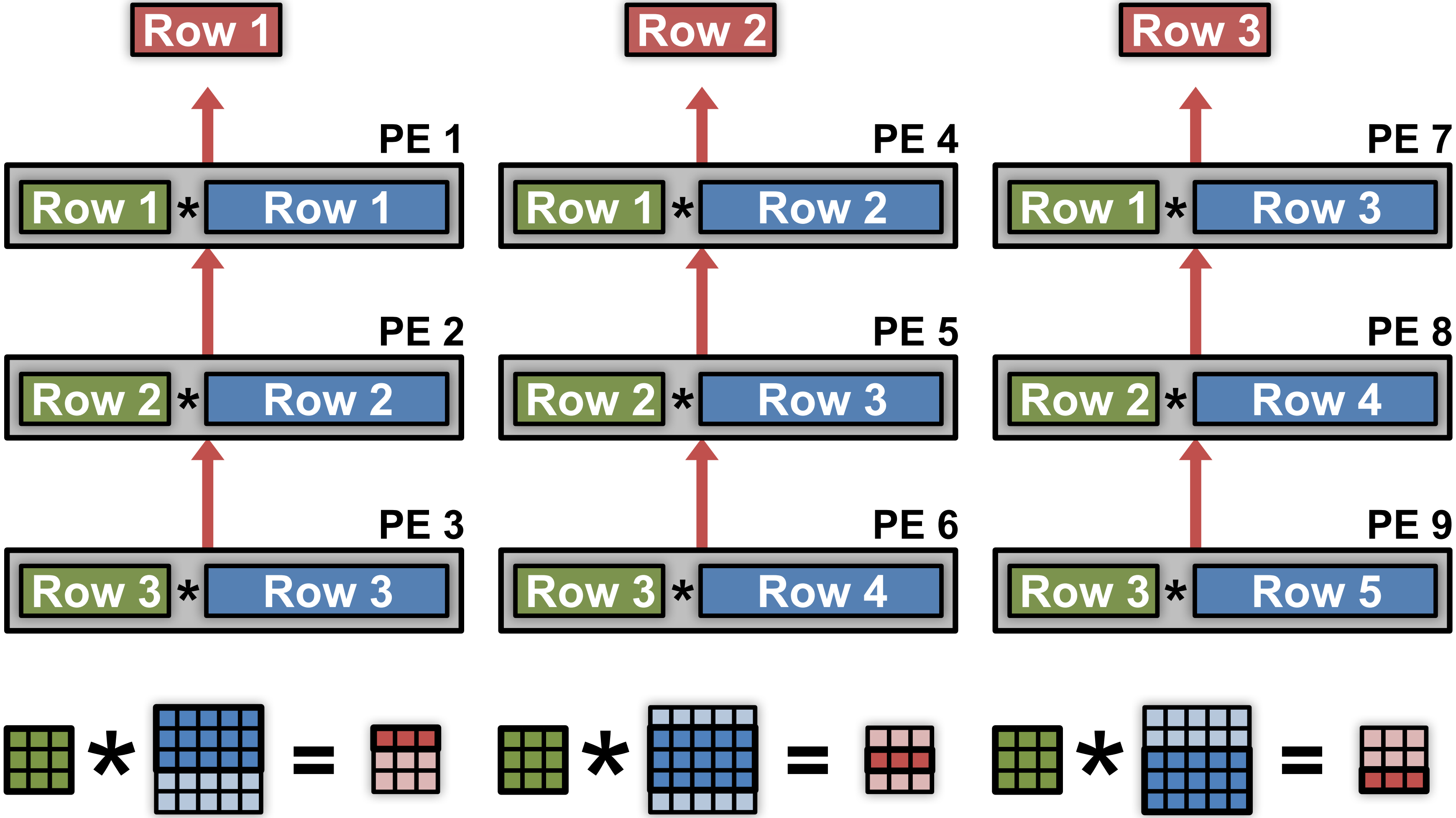
PE 3

Row 3 * Row 3

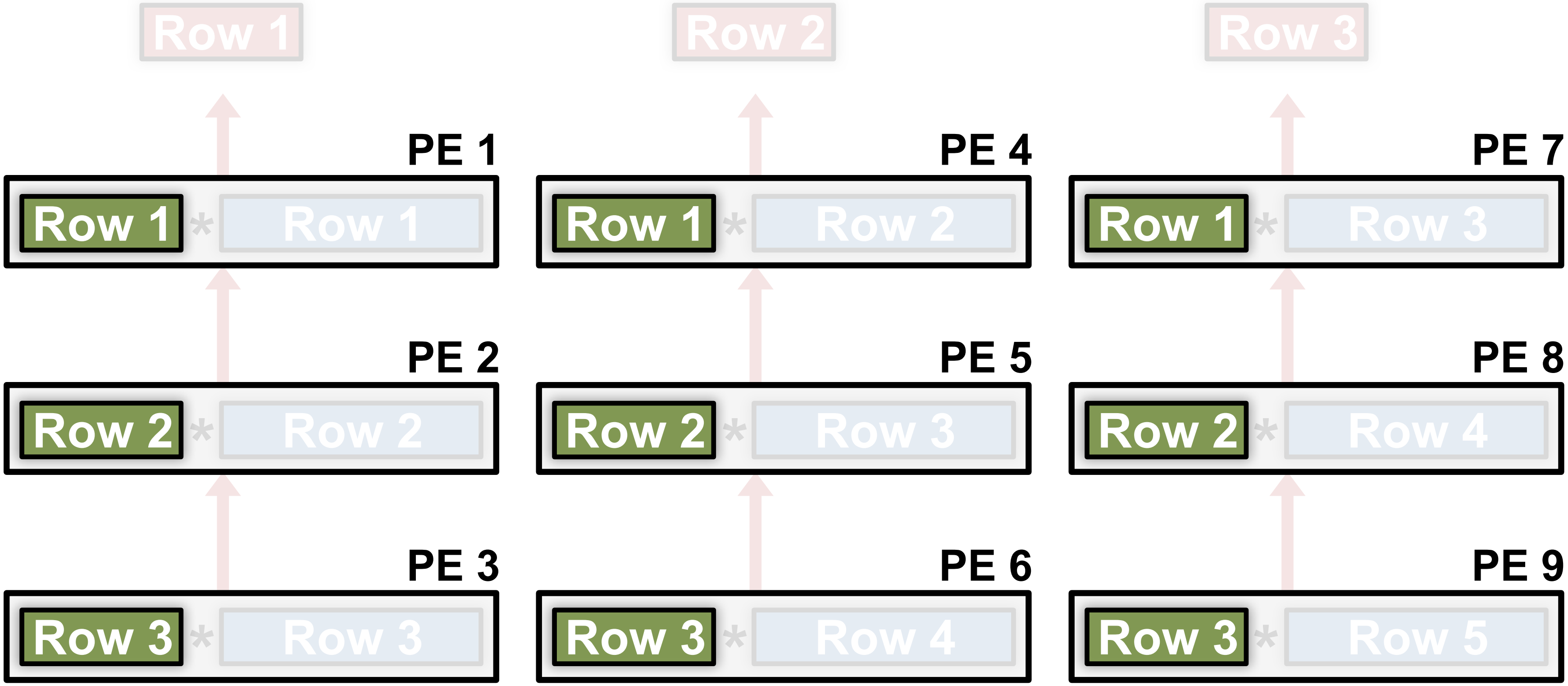
PE 6

Row 3 * Row 4

2D Convolution in PE Array

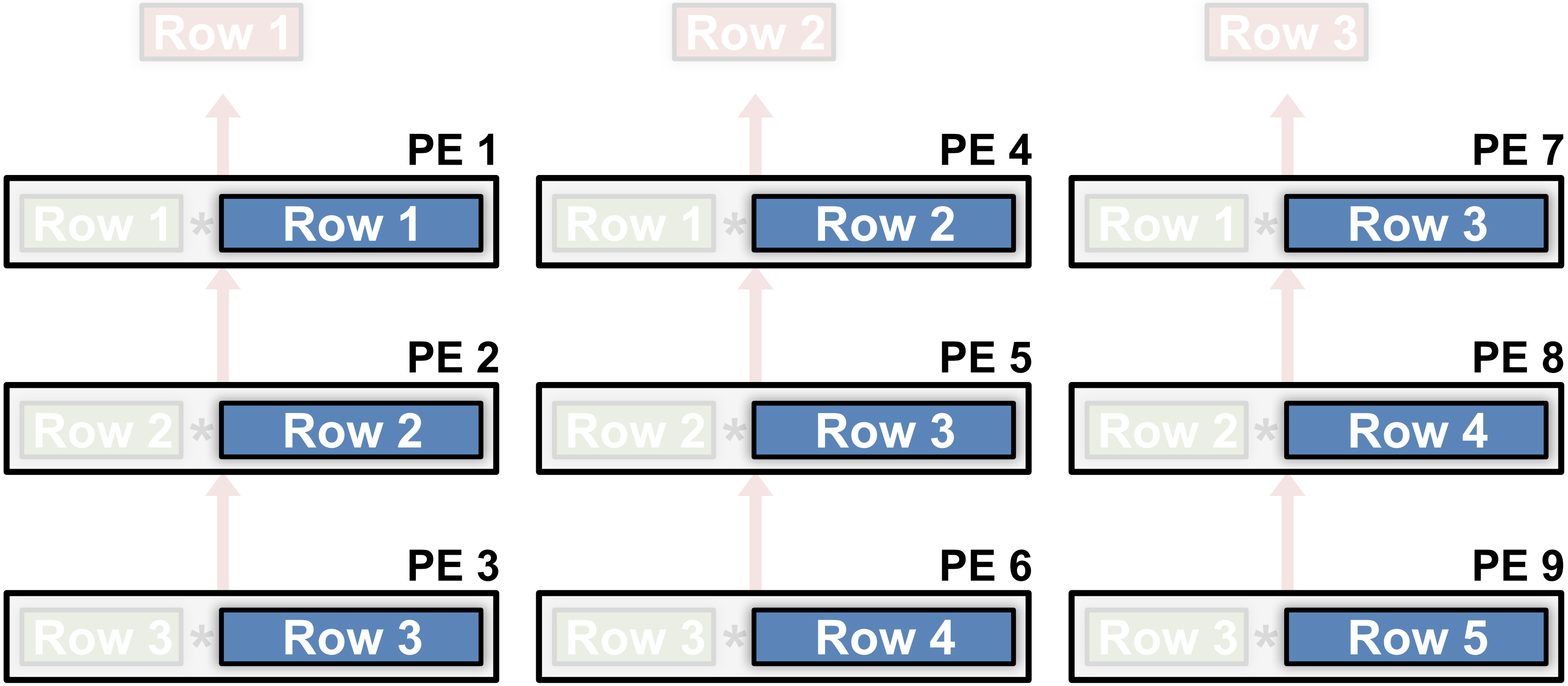


Convolutional Reuse Maximized



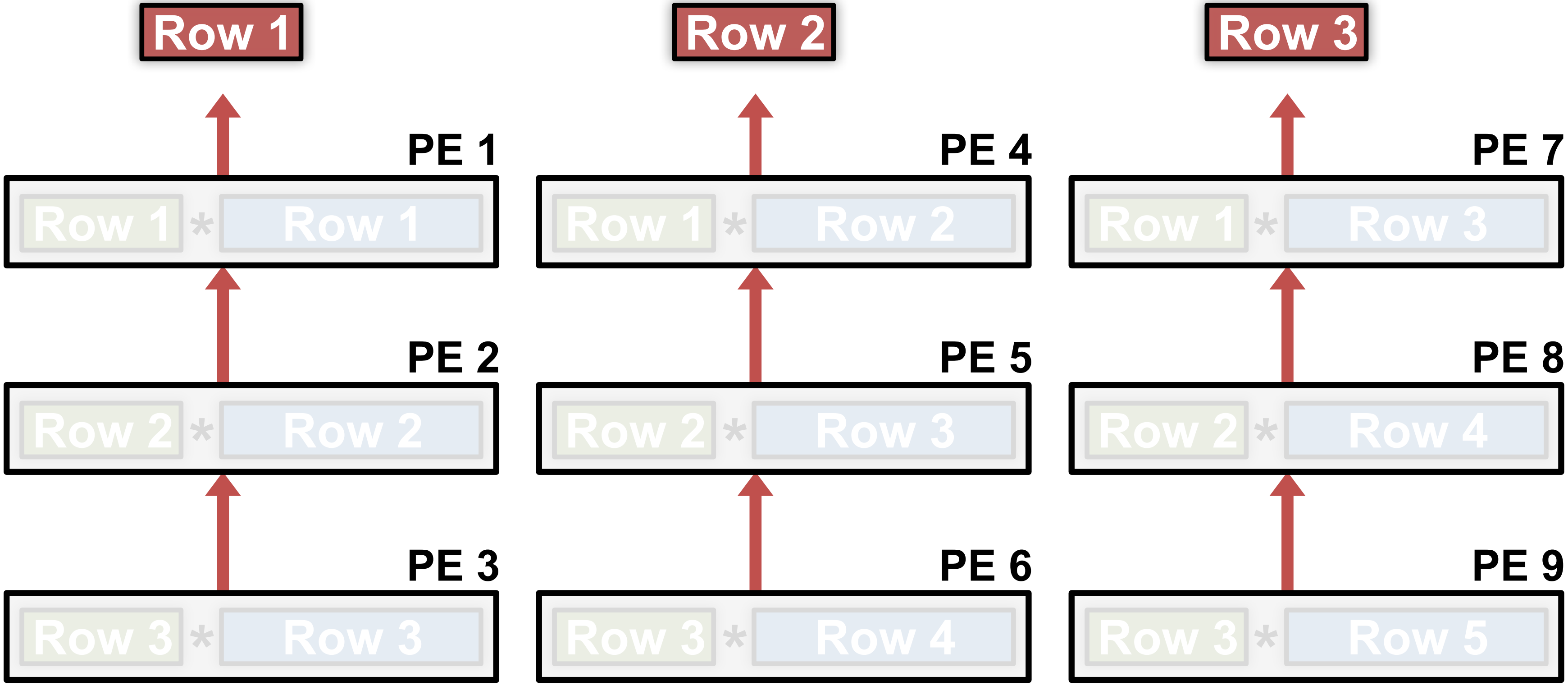
Filter rows are reused across PEs **horizontally**

Convolutional Reuse Maximized



Fmap rows are reused across PEs **diagonally**

Maximize 2D Accumulation in PE Array



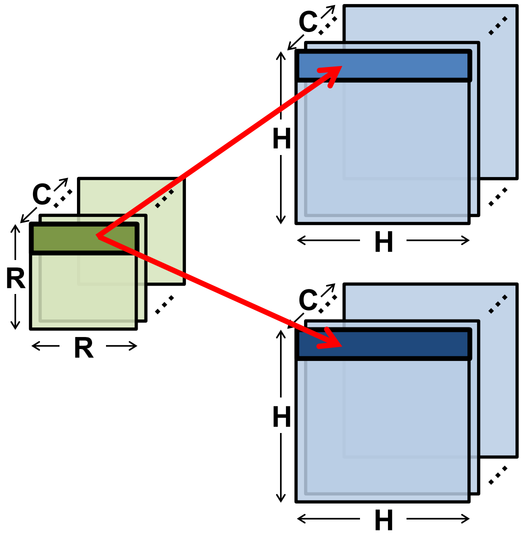
Partial sums accumulate across PEs **vertically**

Dimensions Beyond 2D Convolution

1 **Multiple Fmaps** 2 **Multiple Filters** 3 **Multiple Channels**

Filter Reuse in PE

1 **Multiple Fmaps** 2 Multiple Filters 3 Multiple Channels

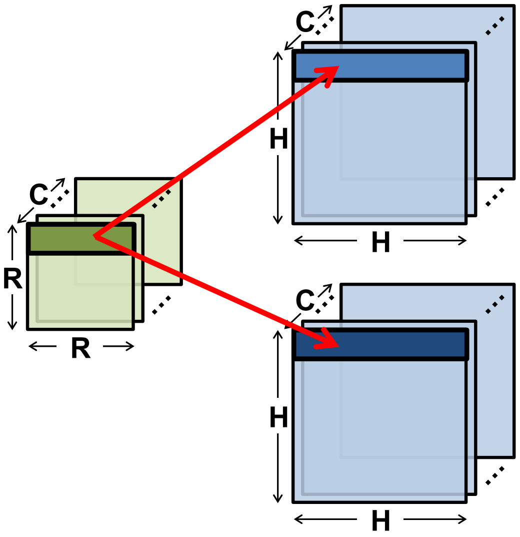


	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

	Filter 1		Fmap 2		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

Filter Reuse in PE

1 Multiple Fmaps   2 Multiple Filters   3 Multiple Channels

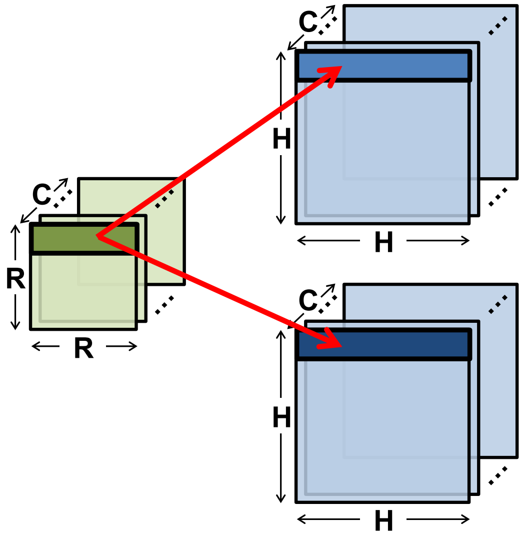


	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 1		Fmap 2		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

share the same filter row

Filter Reuse in PE

1 **Multiple Fmaps** 2 Multiple Filters 3 Multiple Channels



	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

	Filter 1		Fmap 2		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

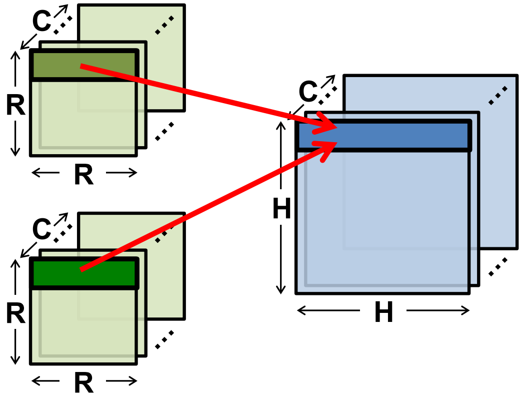
share the same filter row

Processing in PE: **concatenate fmap rows**

	Filter 1		Fmap 1 & 2		Psum 1 & 2
Channel 1	Row 1	*	Row 1 Row 1	=	Row 1 Row 1

Fmap Reuse in PE

1 Multiple Fmaps **2 Multiple Filters** 3 Multiple Channels

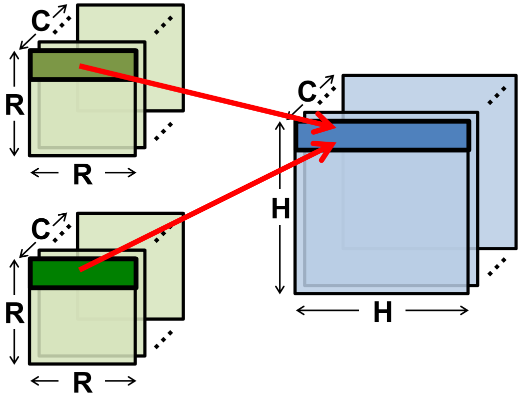


	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

	Filter 2		Fmap 1		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

Fmap Reuse in PE

1 Multiple Fmaps **2 Multiple Filters** 3 Multiple Channels



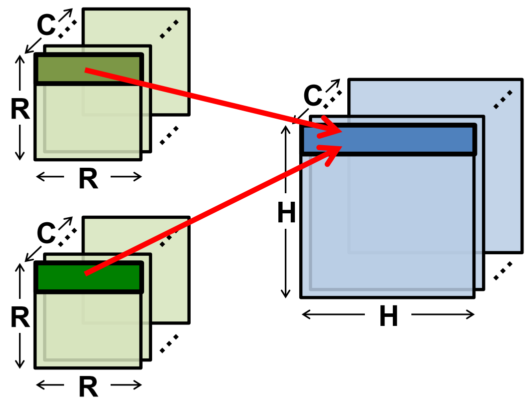
	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

	Filter 2		Fmap 1		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

share the same fmap row

Fmap Reuse in PE

1 Multiple Fmaps 2 **Multiple Filters** 3 Multiple Channels

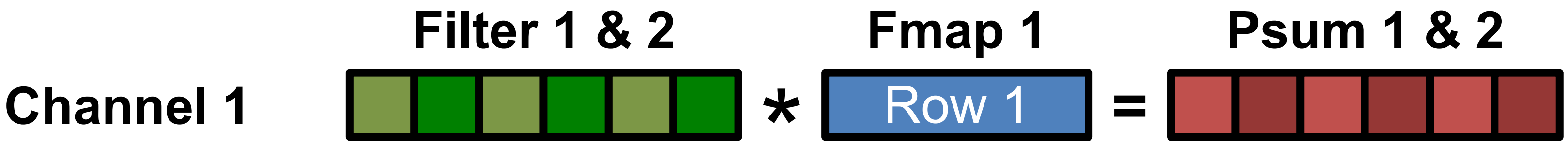


	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

	Filter 2		Fmap 1		Psum 2
Channel 1	Row 1	*	Row 1	=	Row 1

share the same fmap row

Processing in PE: **interleave filter rows**


Filter 1 & 2

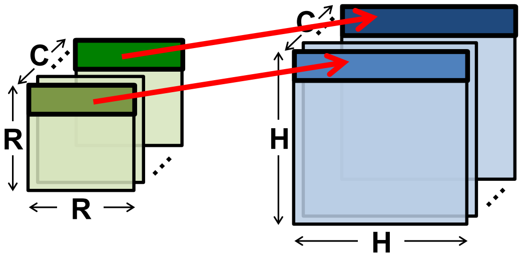
Fmap 1

Psum 1 & 2

Channel 1 🟩🟩🟩🟩🟩🟩 * Row 1 = 🟥🟥🟥🟥🟥🟥

Channel Accumulation in PE

1 Multiple Fmaps   2 Multiple Filters   3 **Multiple Channels**

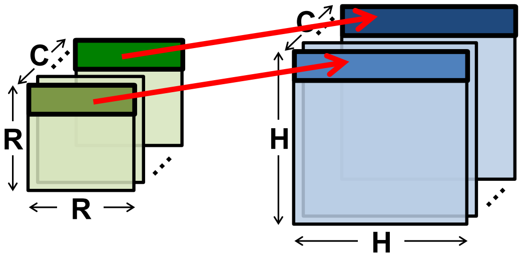


	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

	Filter 1		Fmap 1		Psum 1
Channel 2	Row 1	*	Row 1	=	Row 1

Channel Accumulation in PE

1 Multiple Fmaps 2 Multiple Filters 3 **Multiple Channels**



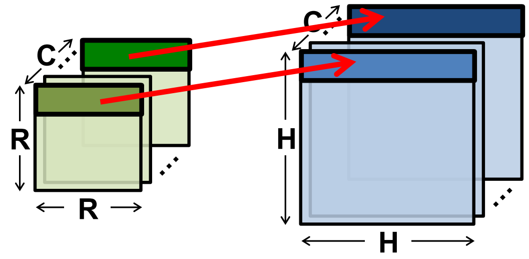
	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1
	Filter 1		Fmap 1		Psum 1
Channel 2	Row 1	*	Row 1	=	Row 1

accumulate psums

Row 1 + Row 1 = Row 1

Channel Accumulation in PE

1 Multiple Fmaps 2 Multiple Filters 3 **Multiple Channels**



Filter 1 Fmap 1 Psum 1

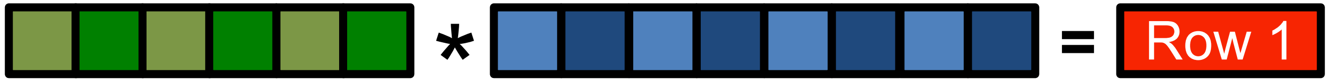
Channel 1 Row 1 * Row 1 = Row 1

Filter 1 Fmap 1 Psum 1

Channel 2 Row 1 * Row 1 = Row 1

accumulate psums

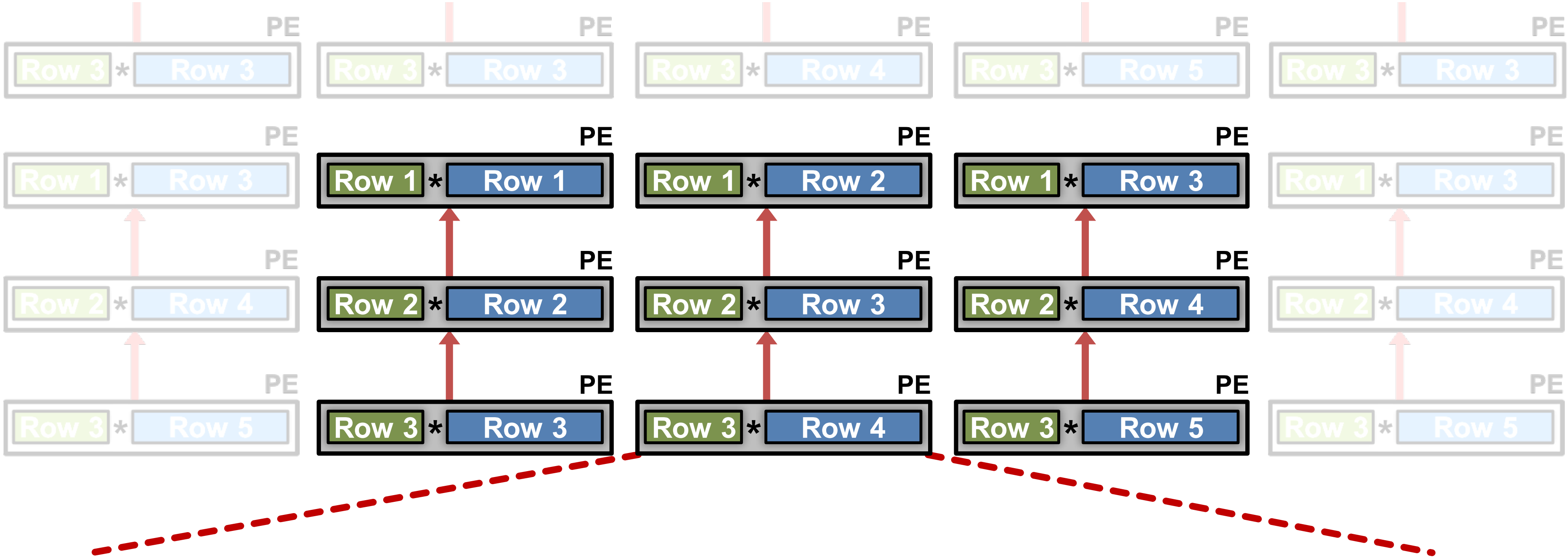
Processing in PE: **interleave channels**



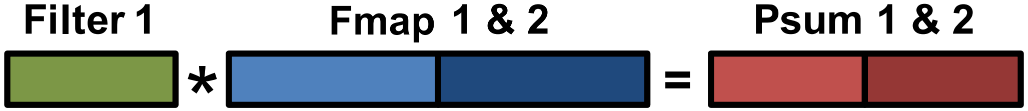
Filter 1 Fmap 1 Psum

Channel 1 & 2 * = Row 1

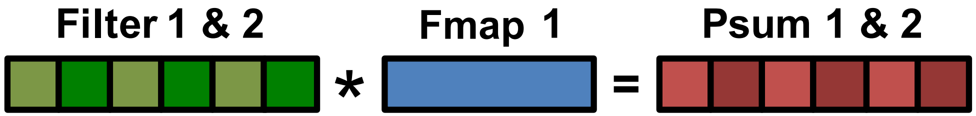
DNN Processing – The Full Picture



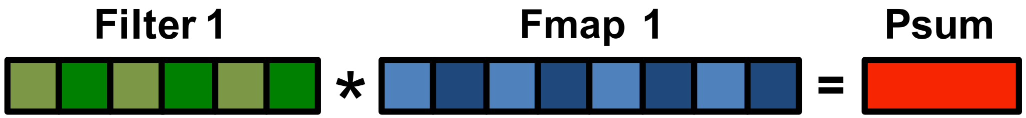
Multiple **fmaps**:



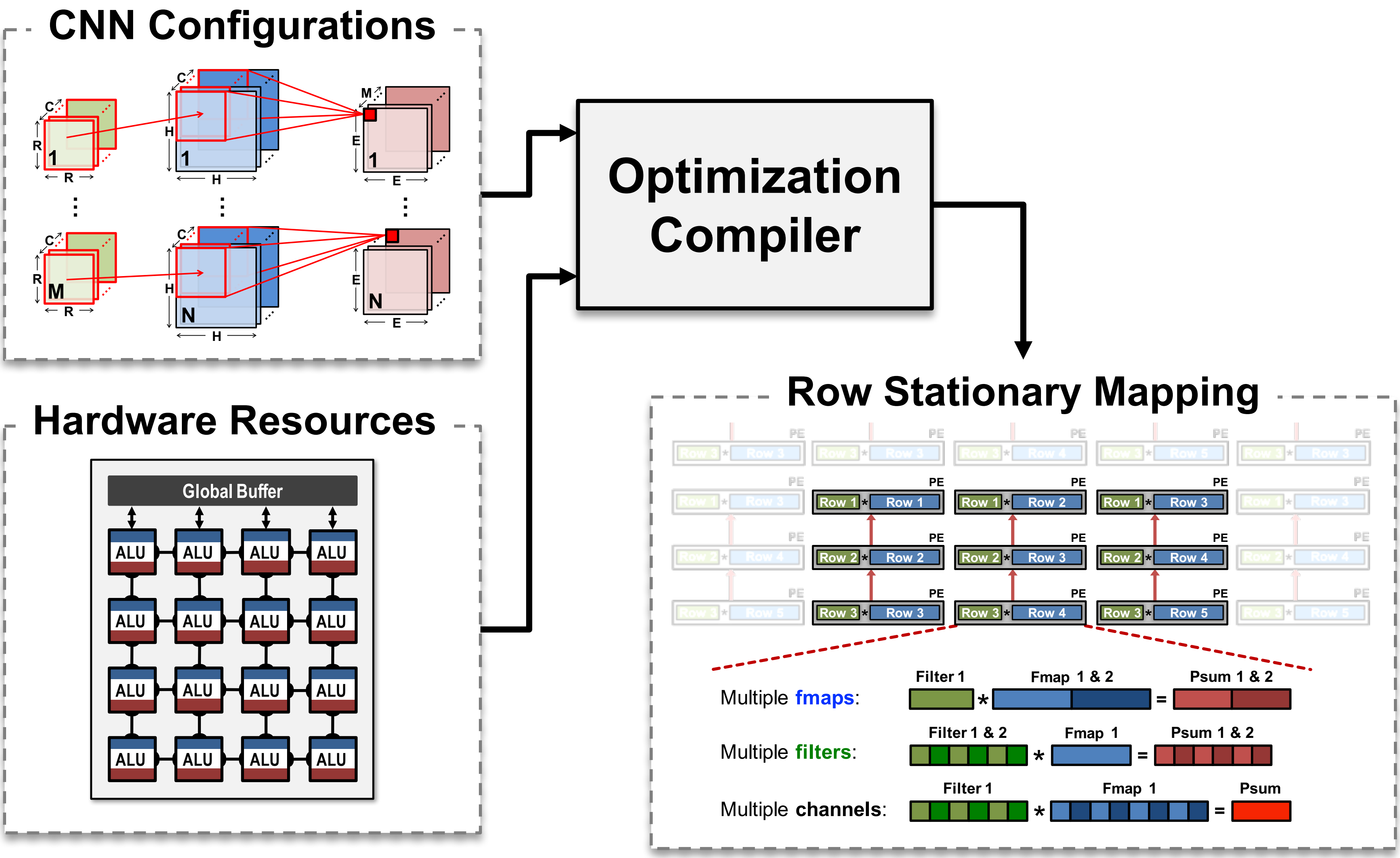
Multiple **filters**:



Multiple **channels**:



Optimal Mapping in Row Stationary



CNN Configurations

Hardware Resources

Optimization Compiler

Row Stationary Mapping

Multiple **fmaps**: Filter 1 * Fmap 1 & 2 = Psum 1 & 2

Multiple **filters**: Filter 1 & 2 * Fmap 1 = Psum 1 & 2

Multiple **channels**: Filter 1 * Fmap 1 = Psum

[Chen et al., ISCA 2016]

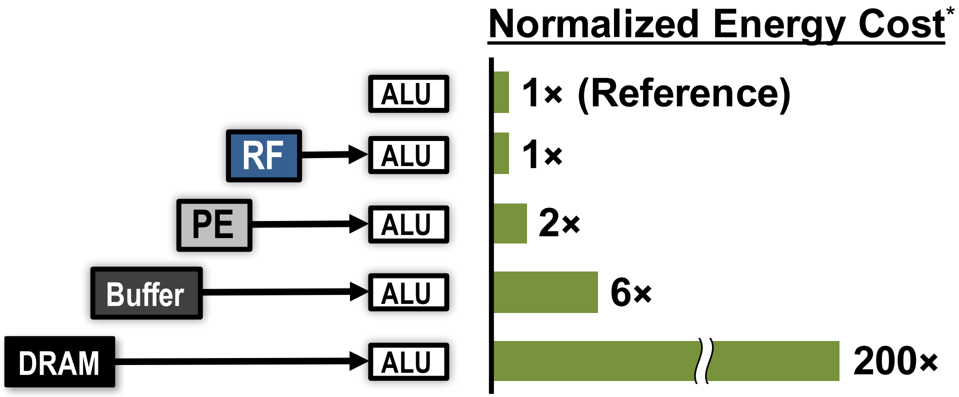
Dataflow Simulation Results

Evaluate Reuse in Different Dataflows

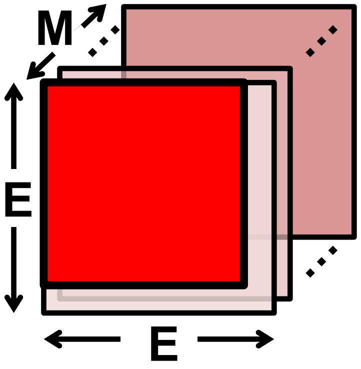
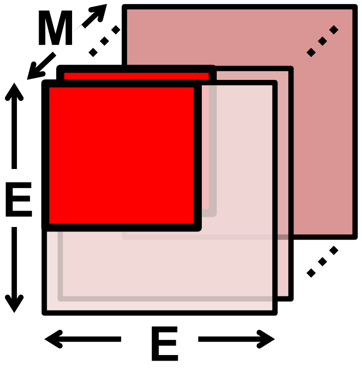
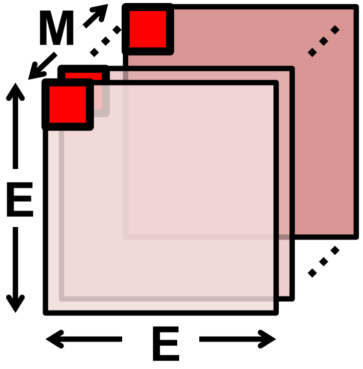
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - No PE local storage. Maximize global buffer size.
- **Row Stationary**

Evaluation Setup

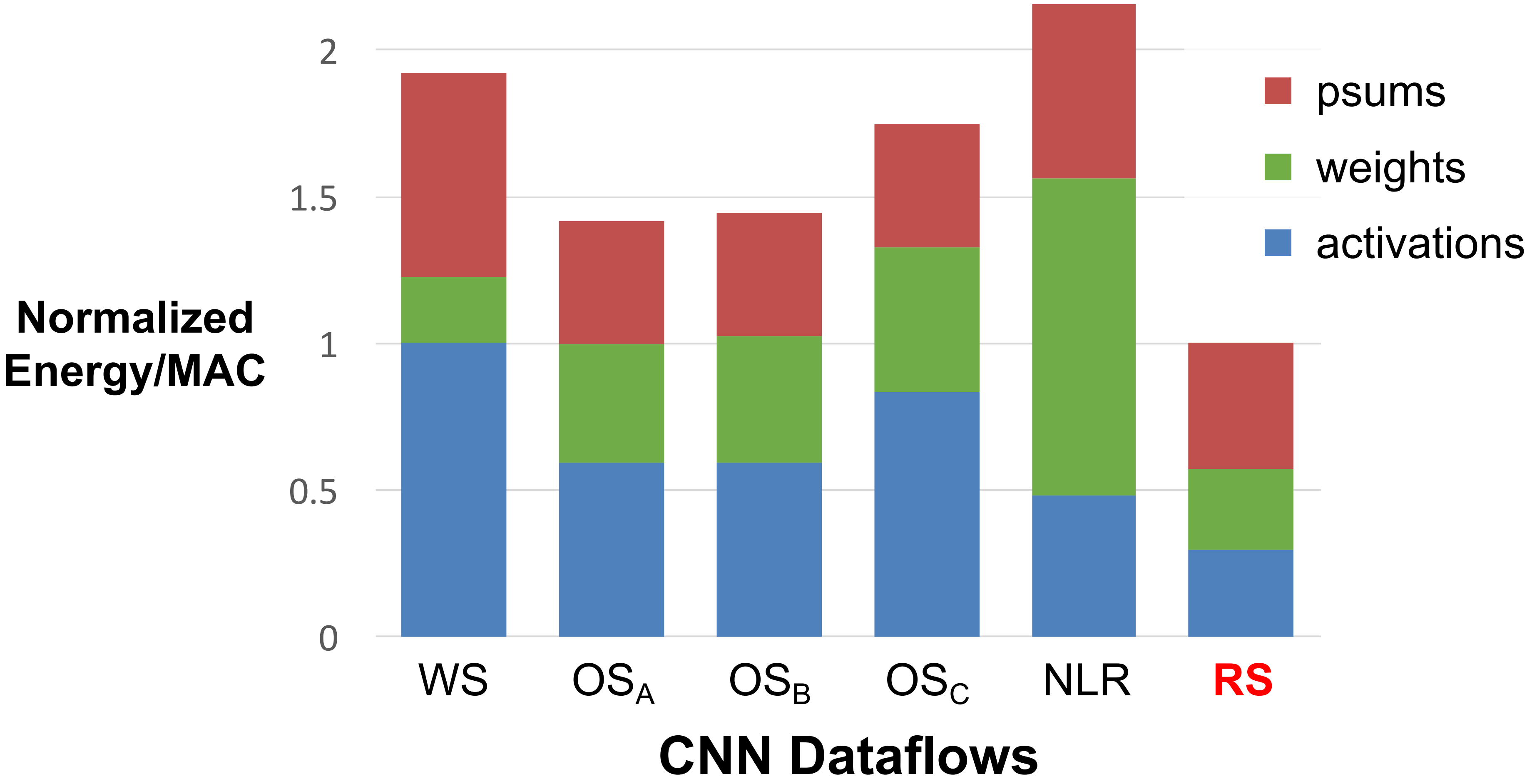
- same total area
- 256 PEs
- AlexNet
- batch size = 16



Variants of Output Stationary

	OS_A	OS_B	OS_C
Parallel Output Region			
# Output Channels	Single	Multiple	Multiple
# Output Activations	Multiple	Multiple	Single
Notes	Targeting **CONV** layers		Targeting **FC** layers

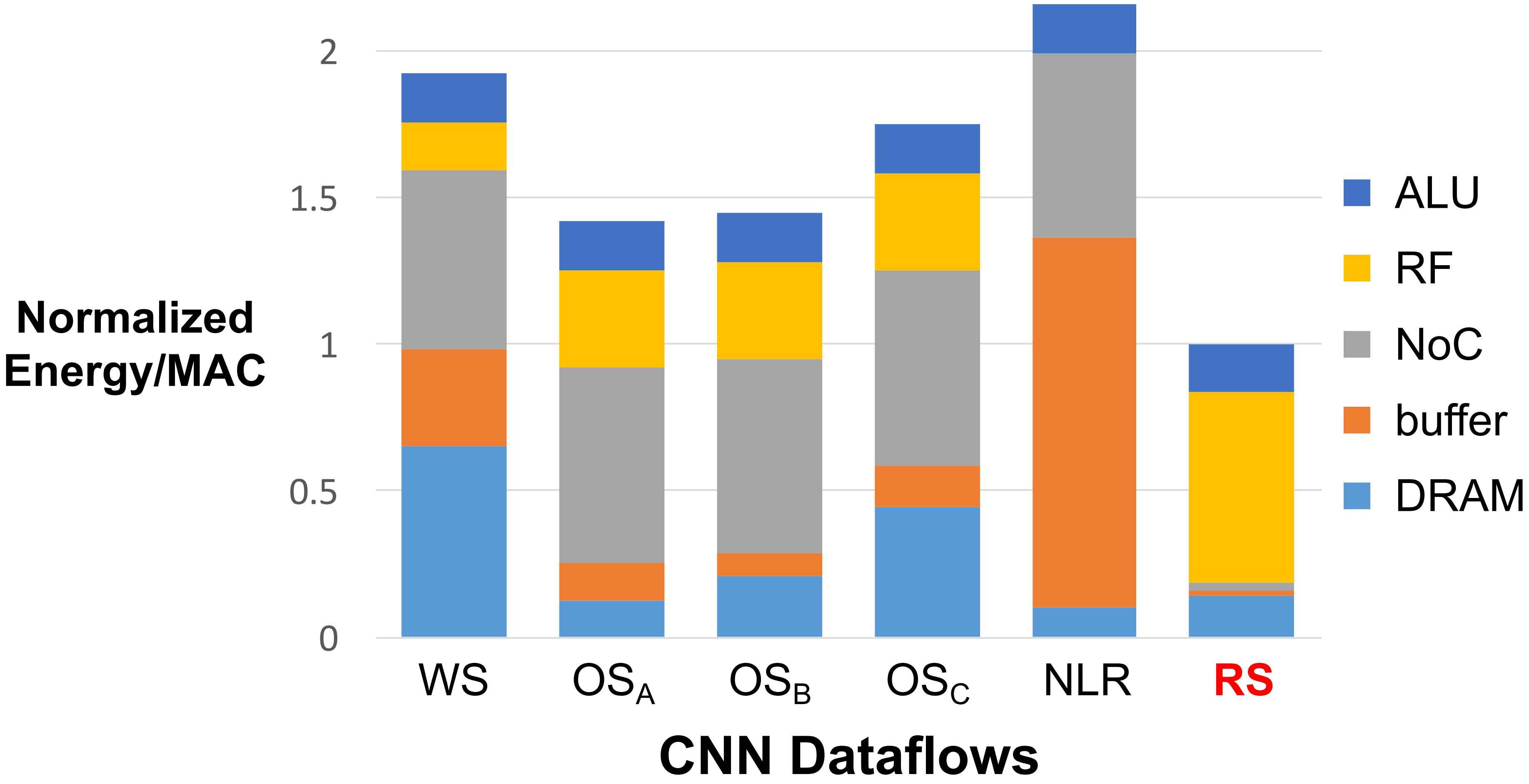
Dataflow Comparison: CONV Layers



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

[Chen et al., ISCA 2016]

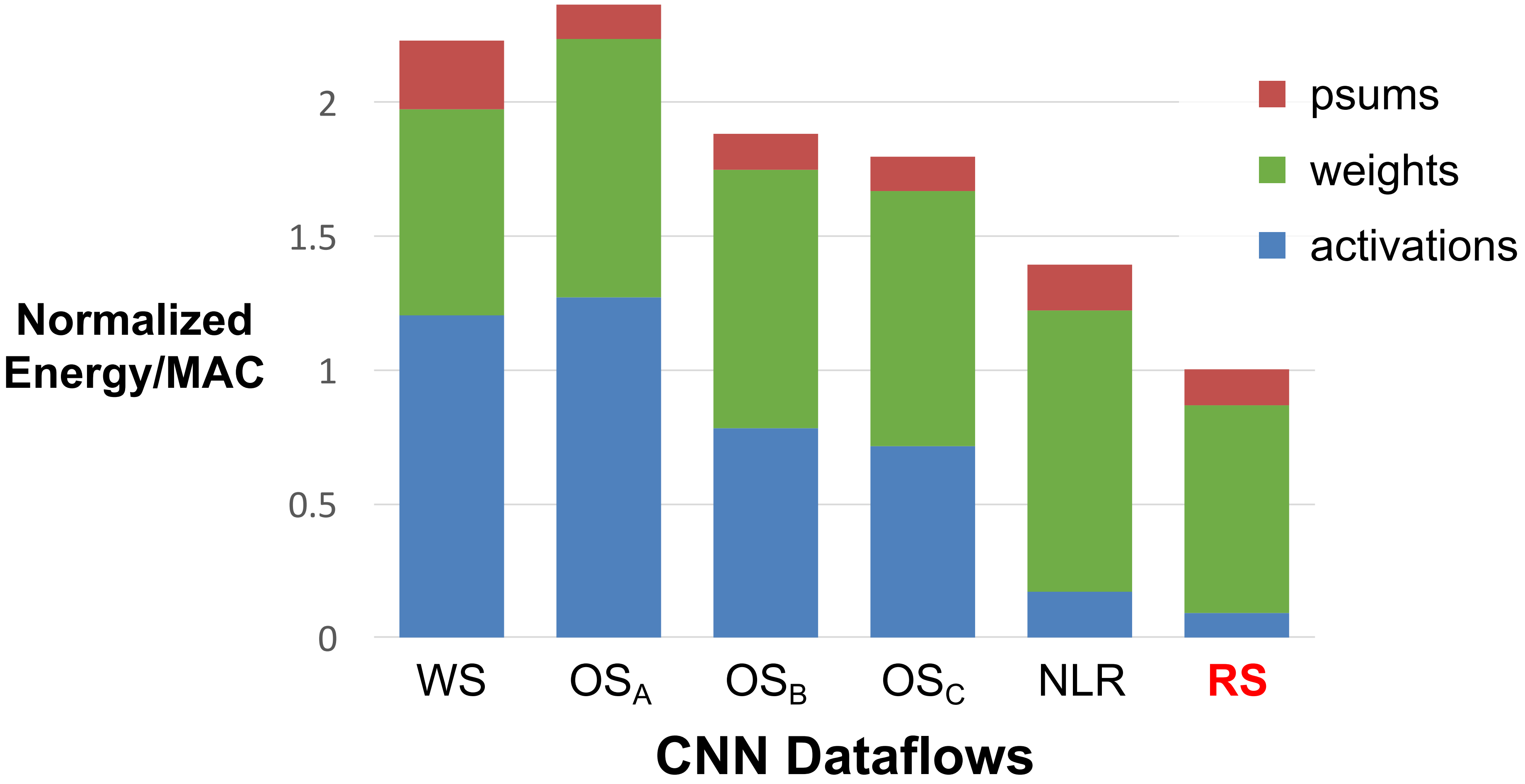
Dataflow Comparison: CONV Layers



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

[Chen et al., ISCA 2016]

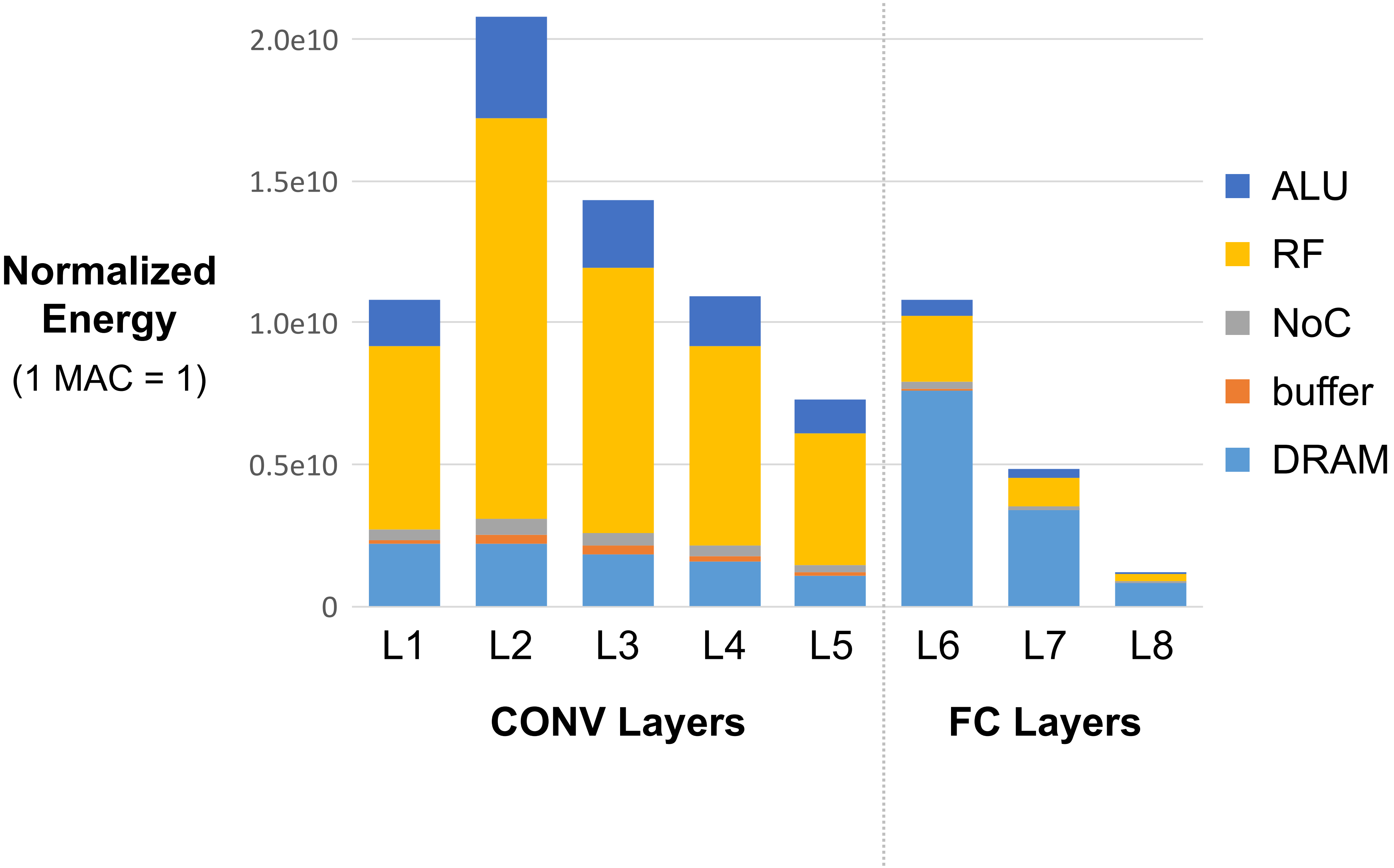
Dataflow Comparison: FC Layers



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

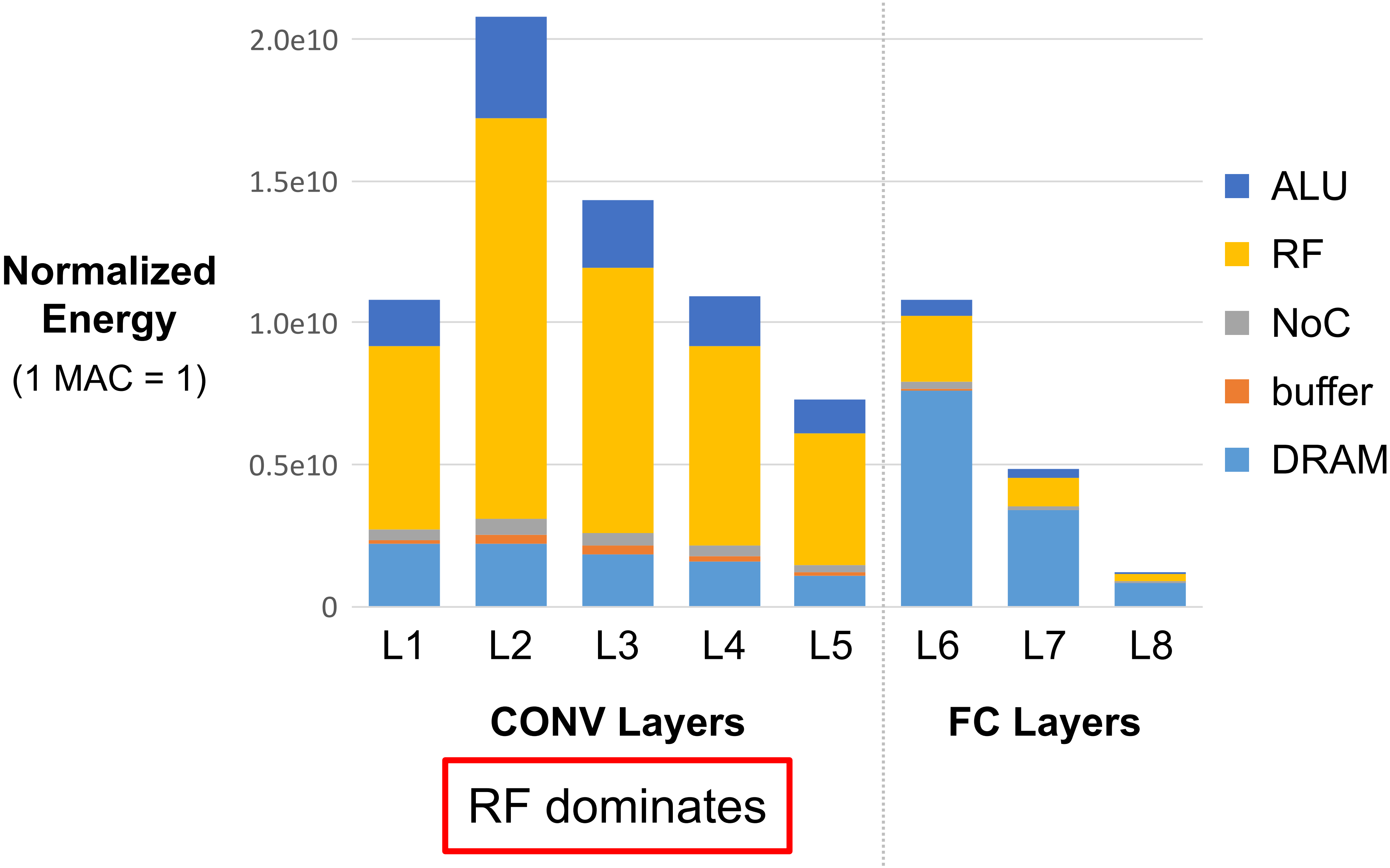
[Chen et al., ISCA 2016]

Row Stationary: Layer Breakdown



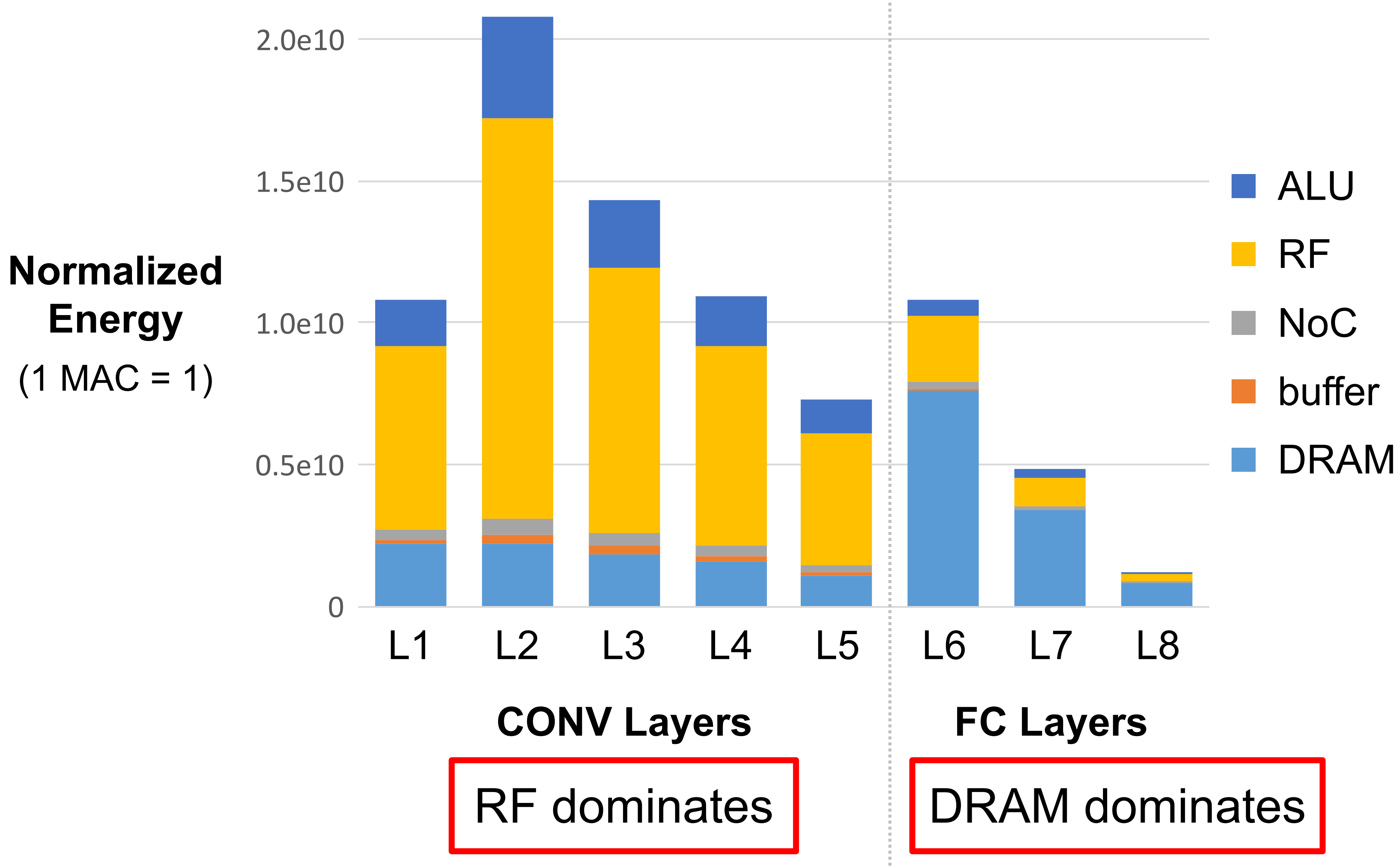
[Chen et al., ISCA 2016]

Row Stationary: Layer Breakdown



RF dominates

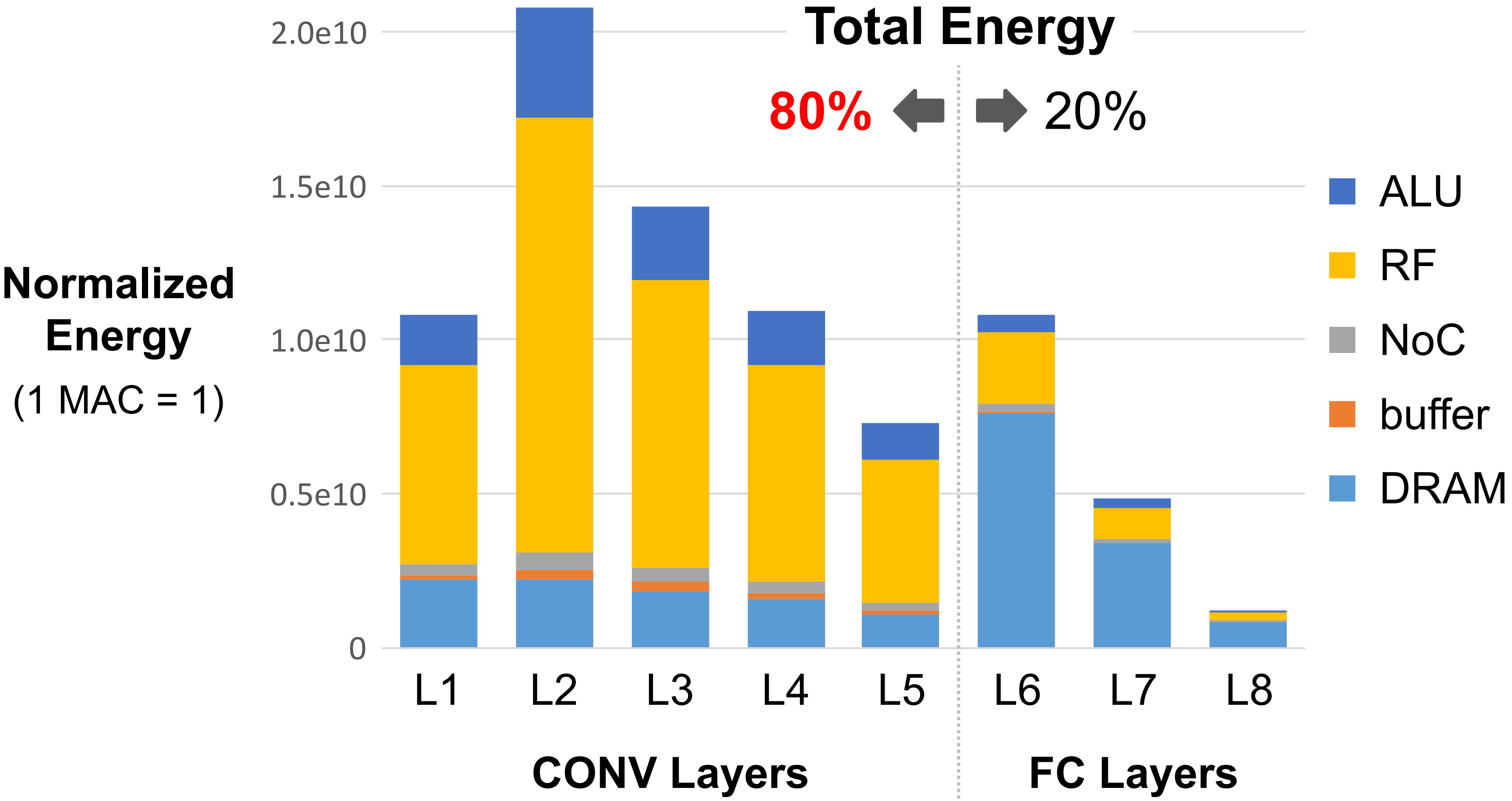
Row Stationary: Layer Breakdown



RF dominates

DRAM dominates

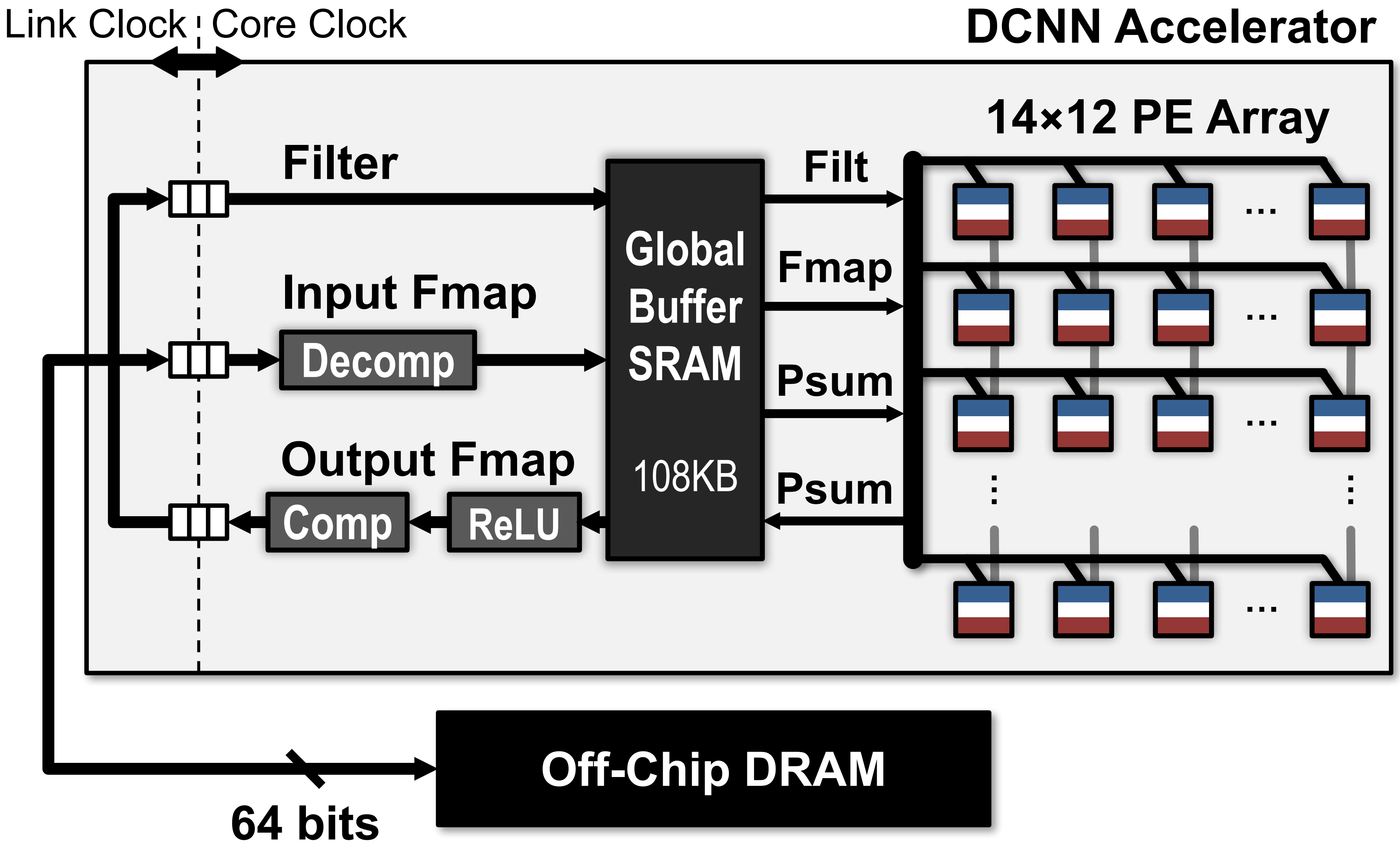
Row Stationary: Layer Breakdown



CONV layers dominate energy consumption!

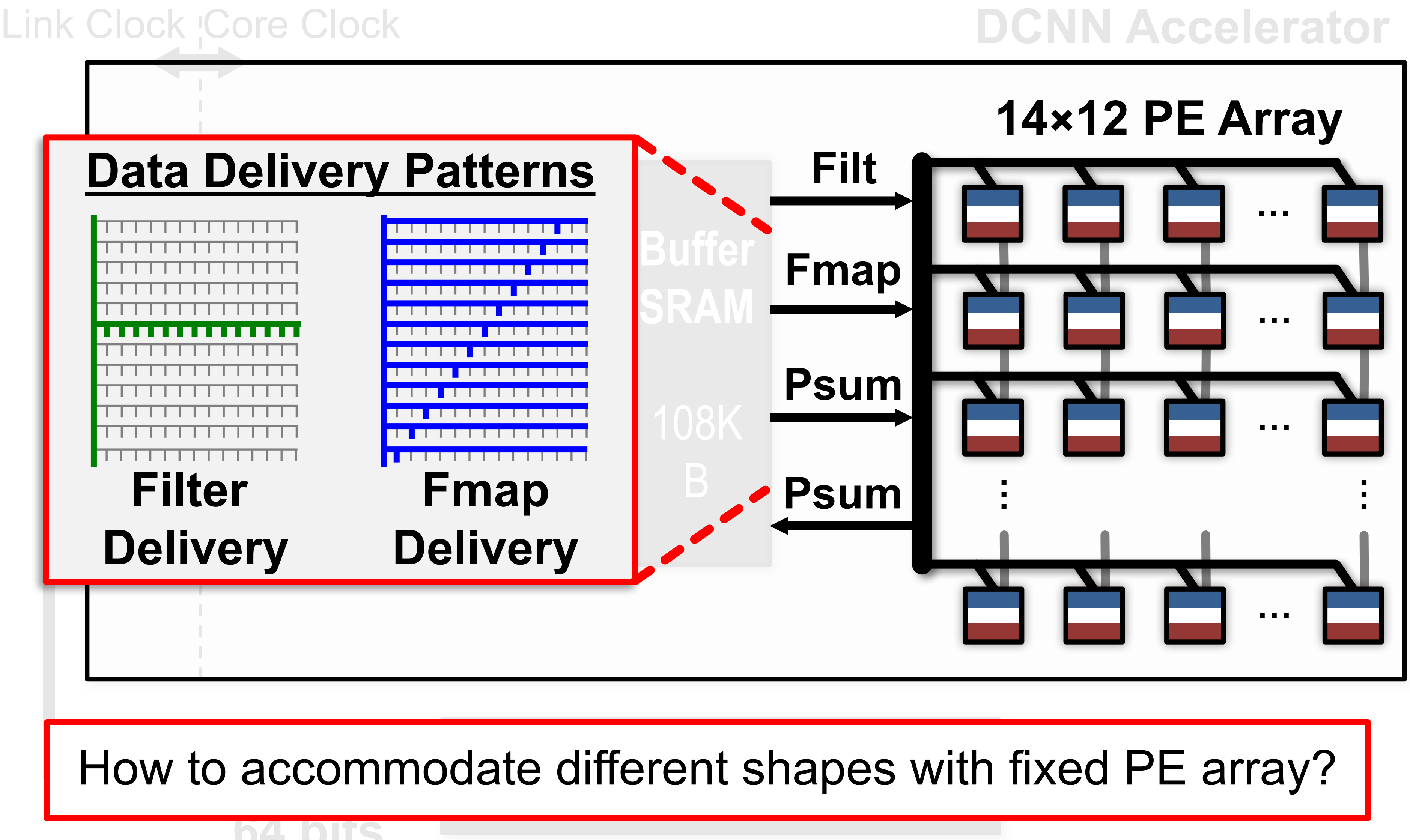
Hardware Architecture for RS Dataflow

Eyeriss Deep CNN Accelerator



[Chen et al., ISSCC 2016]

Data Delivery with On-Chip Network

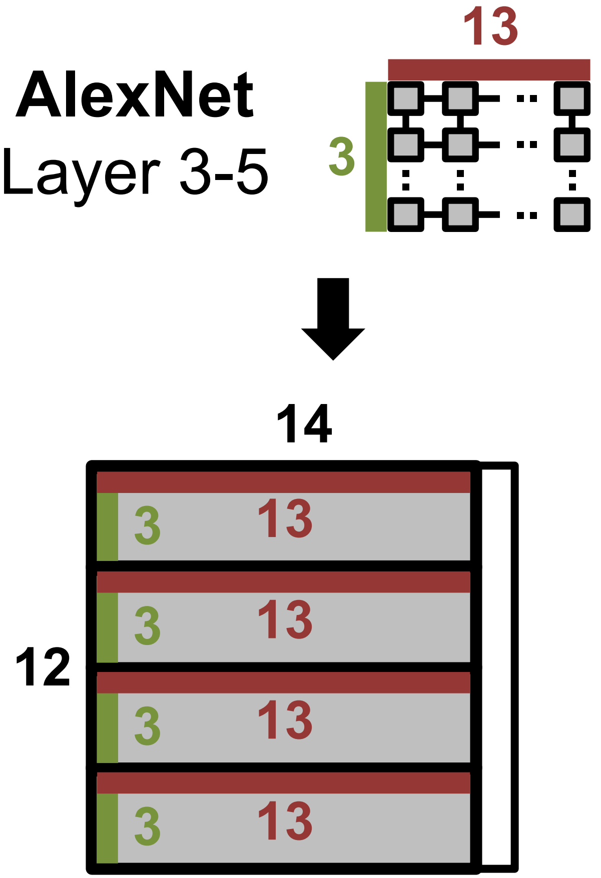


How to accommodate different shapes with fixed PE array?

Logical to Physical Mappings

Replication

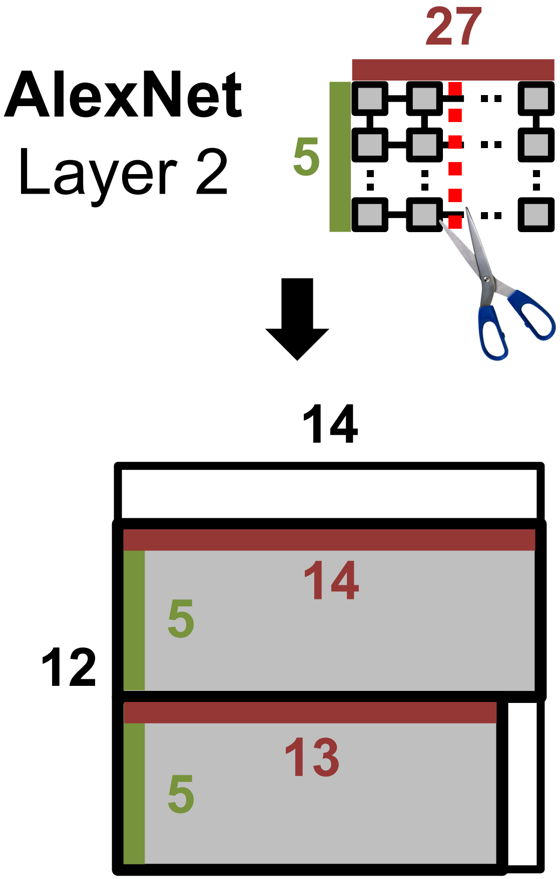
AlexNet
Layer 3-5



Physical PE Array

Folding

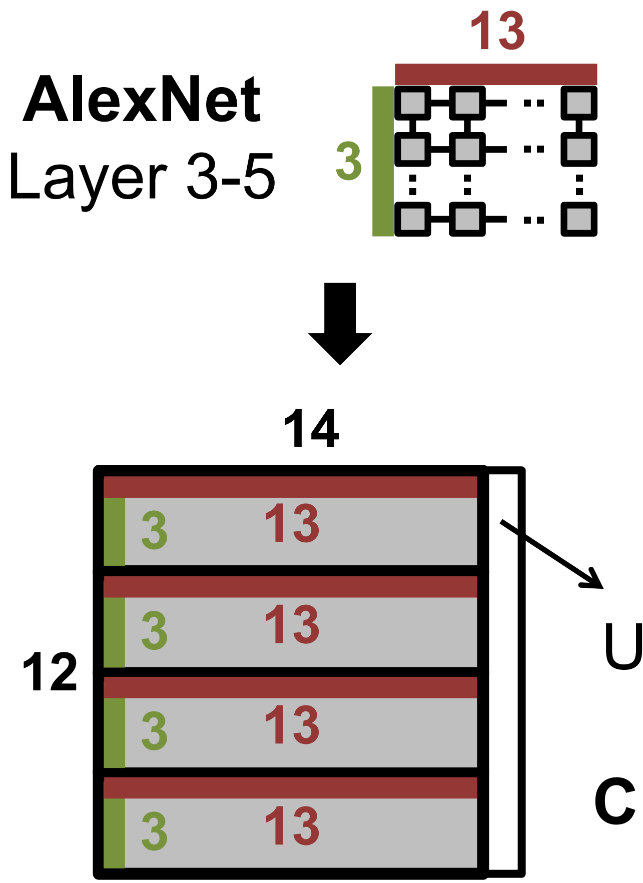
AlexNet
Layer 2



Physical PE Array

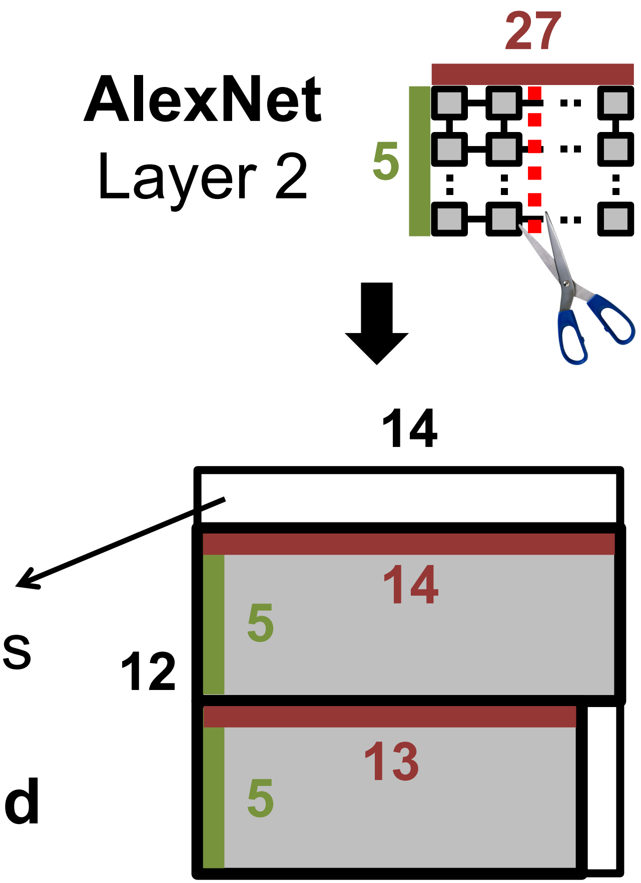
Logical to Physical Mappings

Replication



Physical PE Array

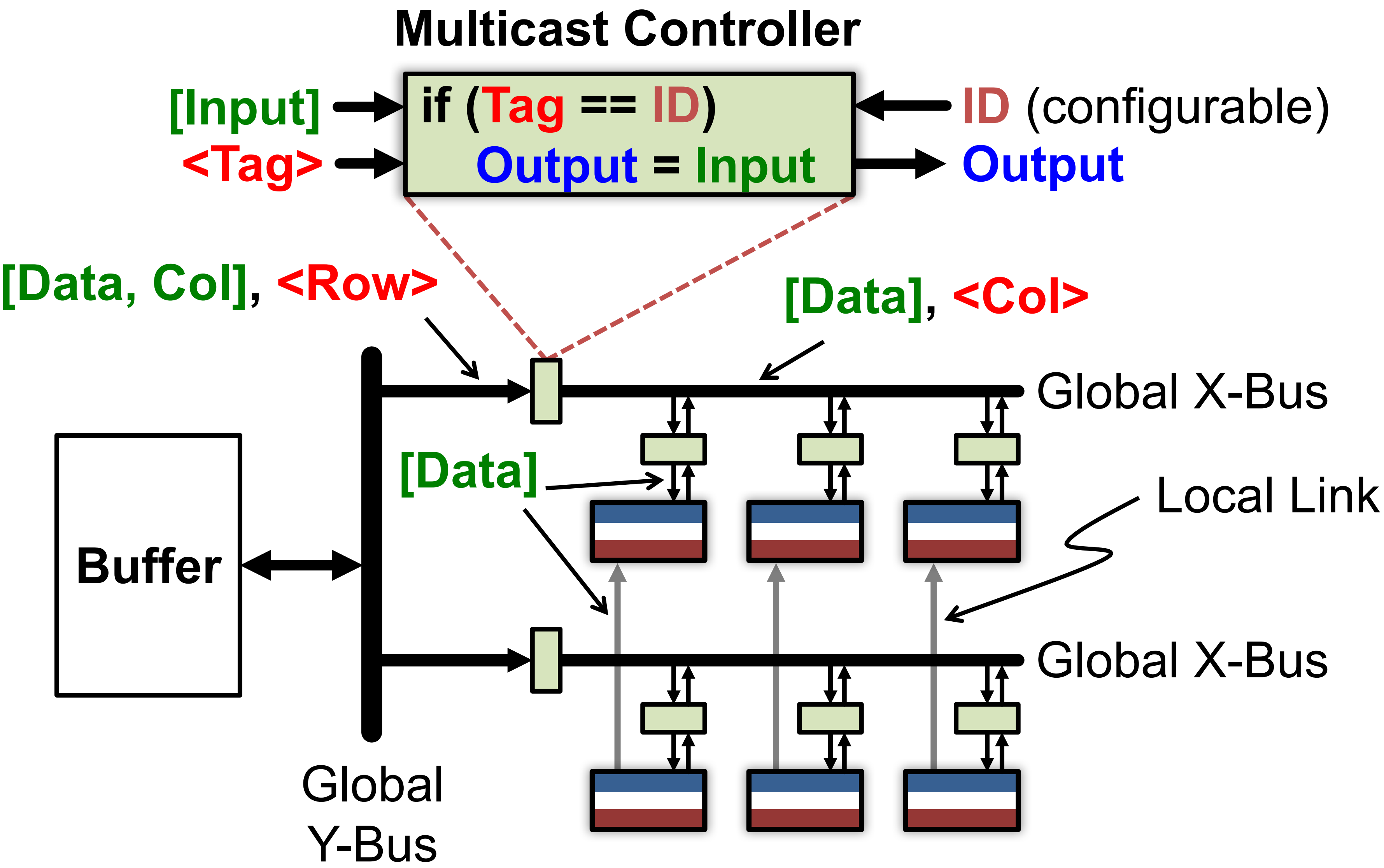
Folding



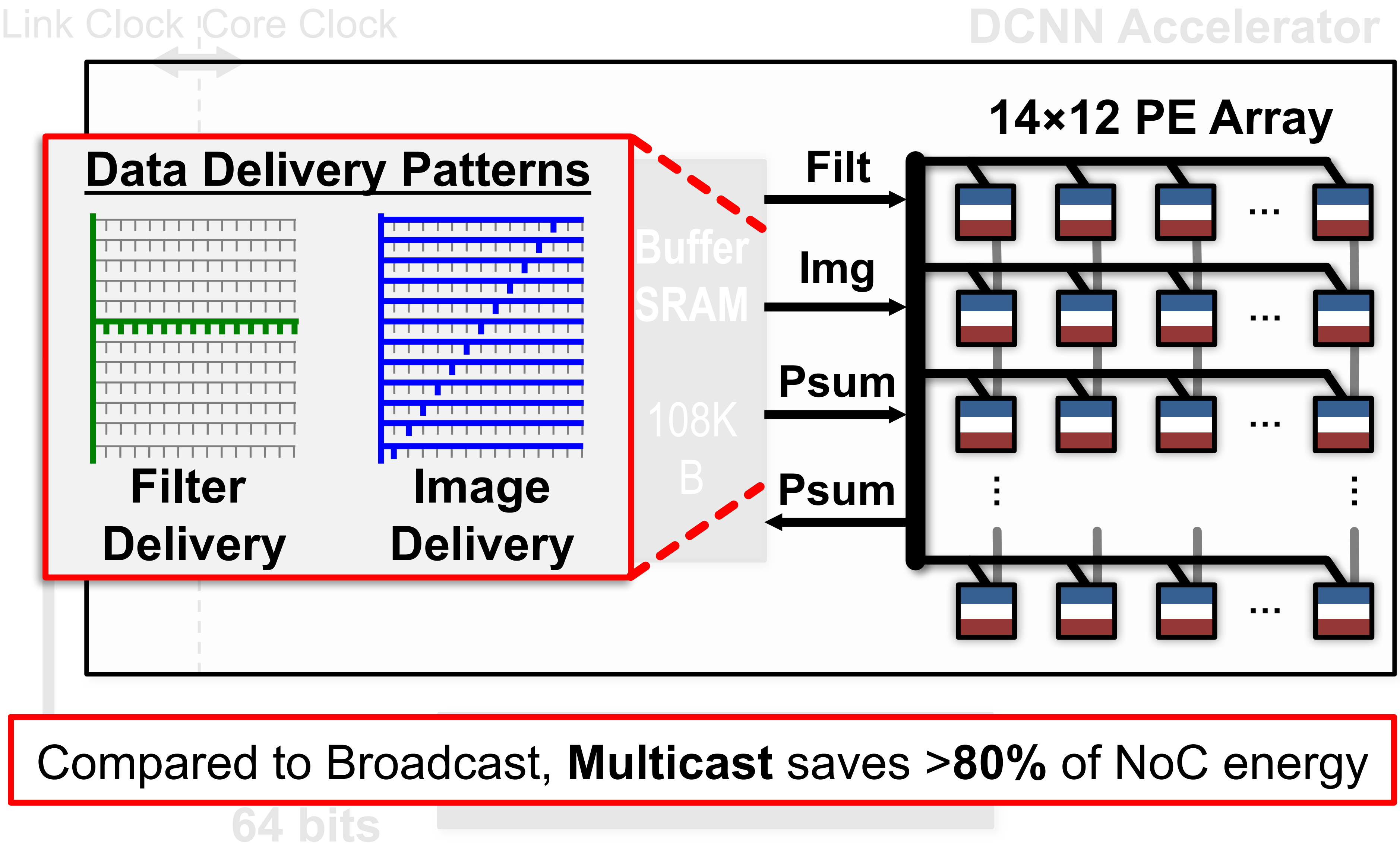
Physical PE Array

Unused PEs are **Clock Gated**

Multicast Network Design



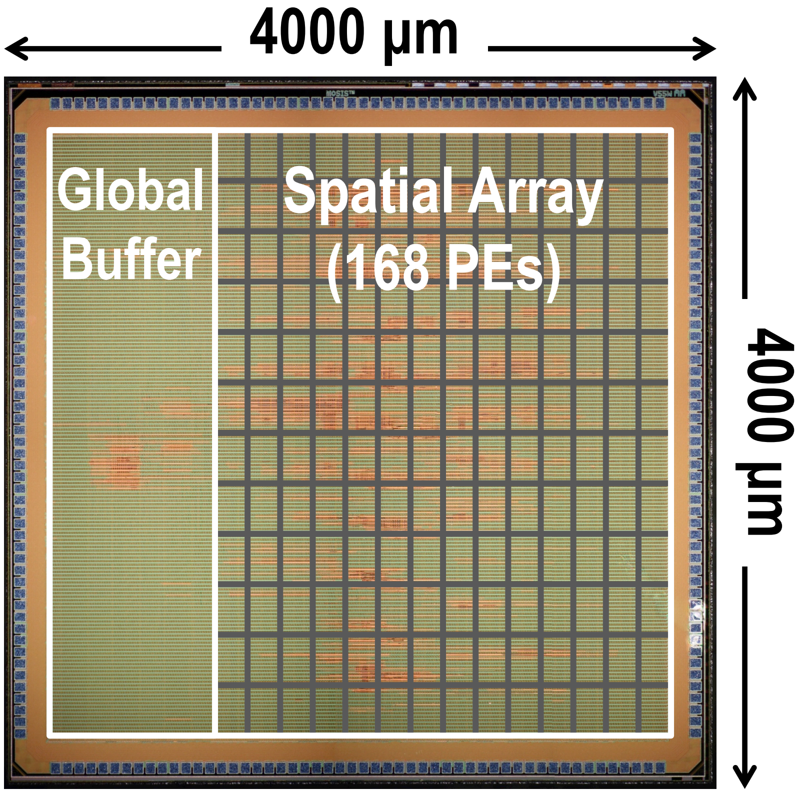
Data Delivery with On-Chip Network



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

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



[Chen et al., 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)

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**

51682 operand* access/input image pixel

→ **506** access/pixel from buffer **+ 37** access/pixel from DRAM

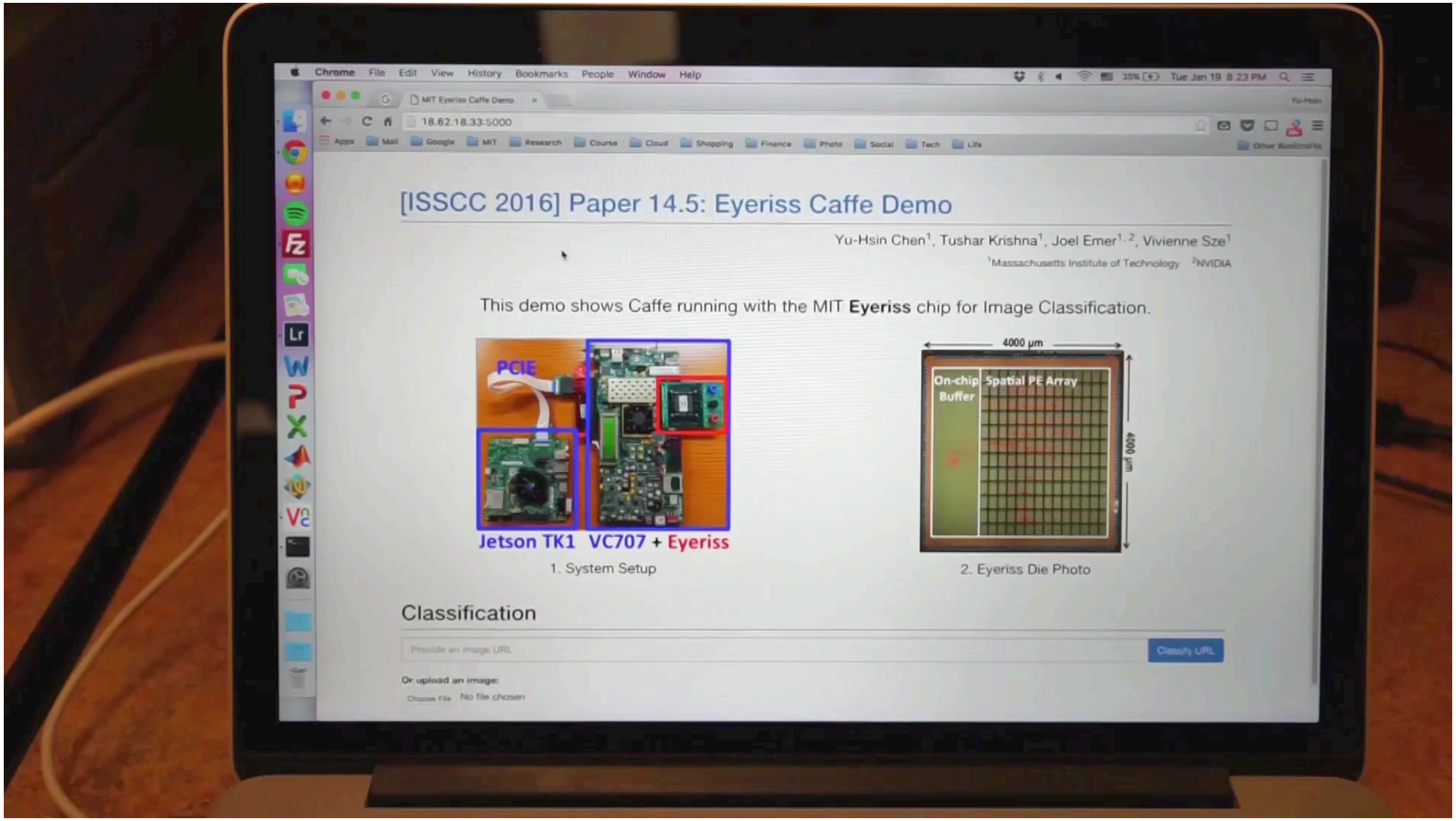
*operand = weight, activation, psum

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 CONV Layers
2. Board Power
3. Modeled from [Tan, SC 2011]

From Architecture to System



[ISSCC 2016] Paper 14.5: Eyeriss Caffe Demo

Yu-Hsin Chen[1], Tushar Krishna[1], Joel Emer[1,2], Vivienne Sze[1]
[1]Massachusetts Institute of Technology [2]NVIDIA

This demo shows Caffe running with the MIT **Eyeriss** chip for Image Classification.

1. System Setup

2. Eyeriss Die Photo

Classification

Provide an image URL

Or upload an image:

No file chosen

https://vimeo.com/154012013

Summary of DNN Dataflows

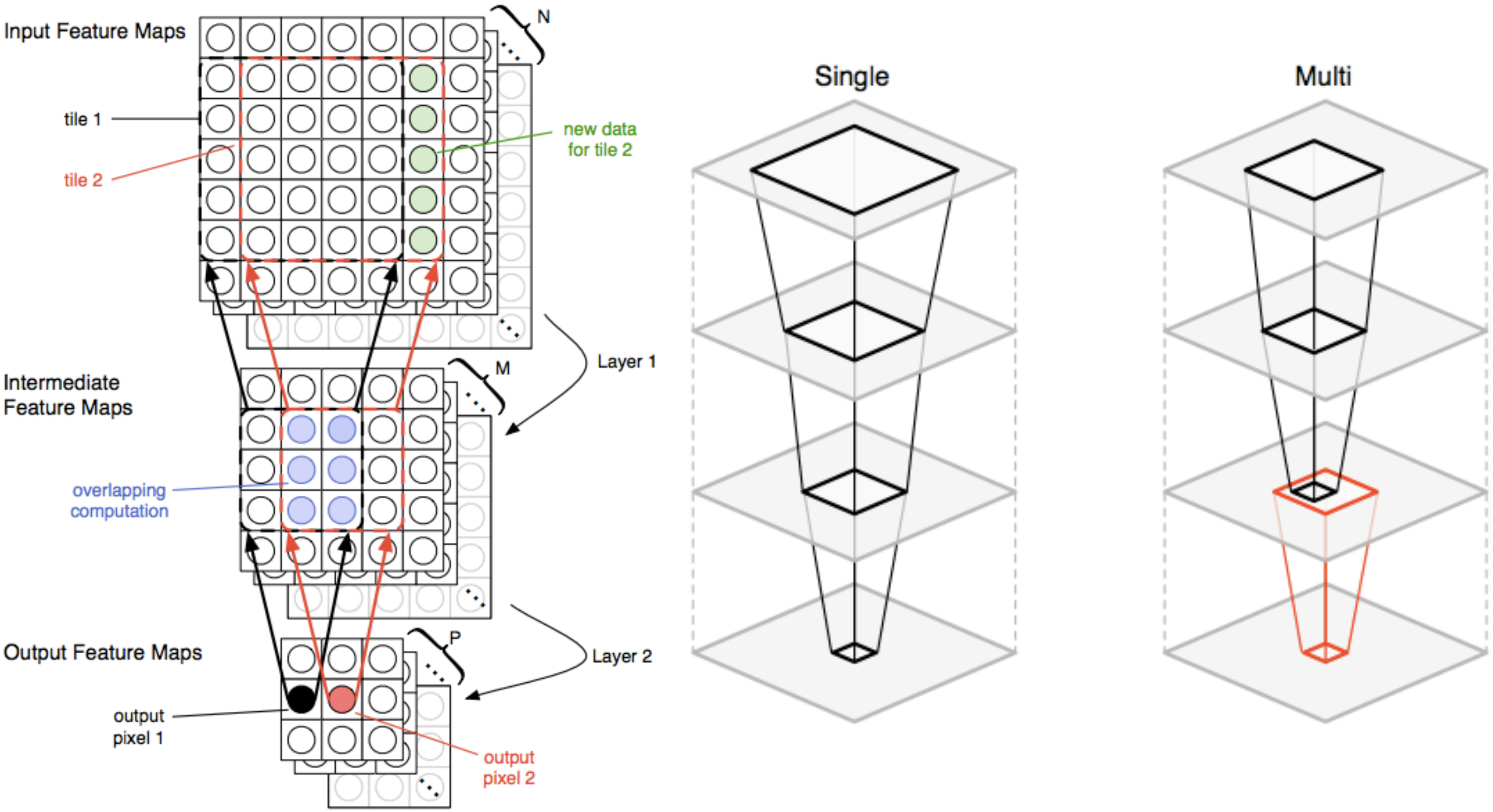
- **Weight Stationary**
  - Minimize movement of filter weights
  - Popular with processing-in-memory architectures
- **Output Stationary**
  - Minimize movement of partial sums
  - Different variants optimized for CONV or FC layers
- **No Local Reuse**
  - No PE local storage → maximize global buffer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

MICRO 2016 Papers in the Taxonomy

- **Stripes:** bit-serial computation in a **NLR**-like engine (based on DaDianNao)
- **NEUTRAMS**: a toolset for accelerators running the **WS** dataflow (synaptic weight memory array)
- **Fused-layer**: exploit inter-layer data reuse in a **NLR** engine (based on [Zhang, *FPGA* 2015])

Fused Layer

- **Dataflow across multiple layers**



[Alwani et al., MICRO 2016]