Hardware Architectures for Deep Neural Networks

ISCA Tutorial

June 22, 2019

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

Background of Deep Neural Networks

Neural Networks: Weighted Sum

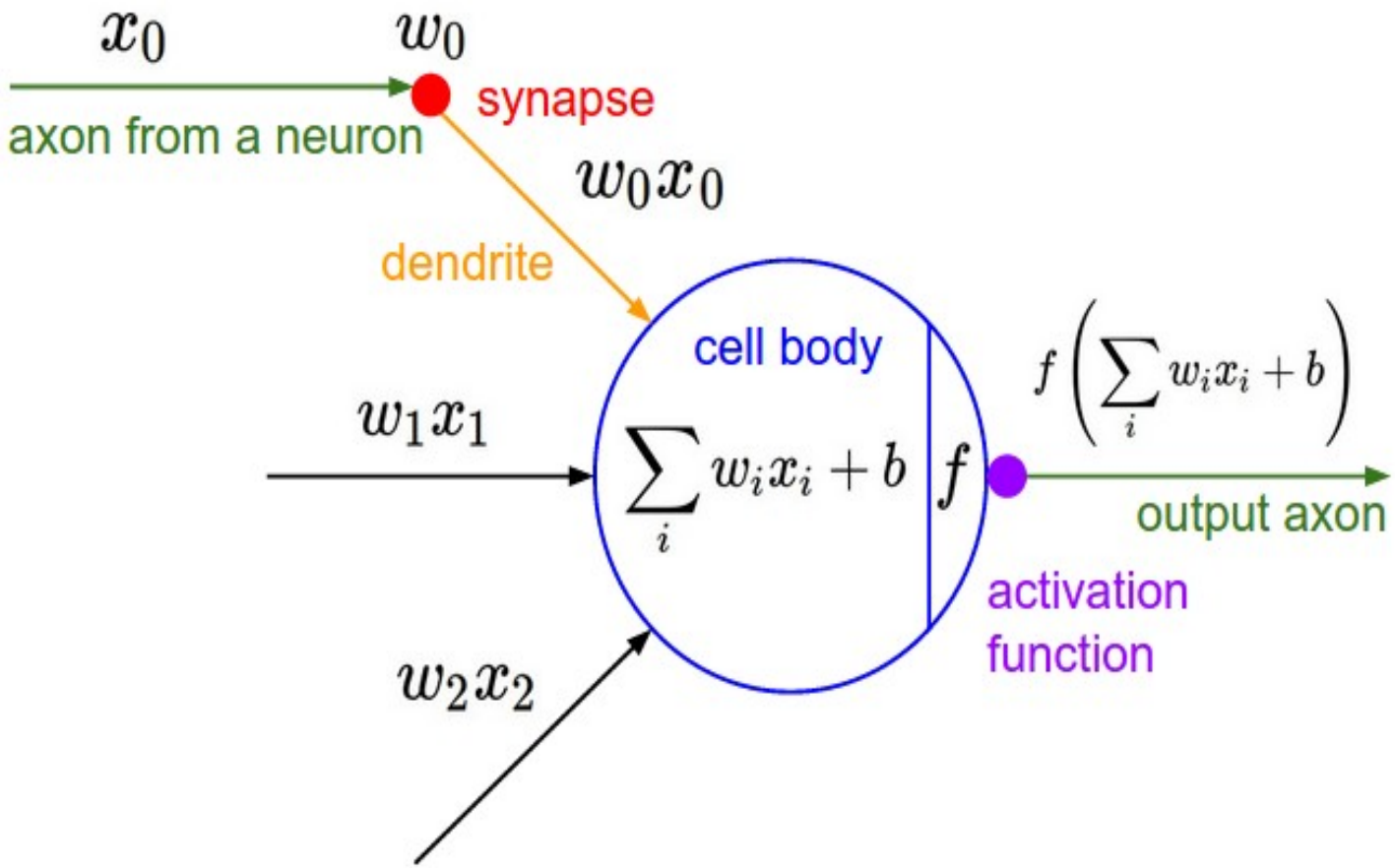
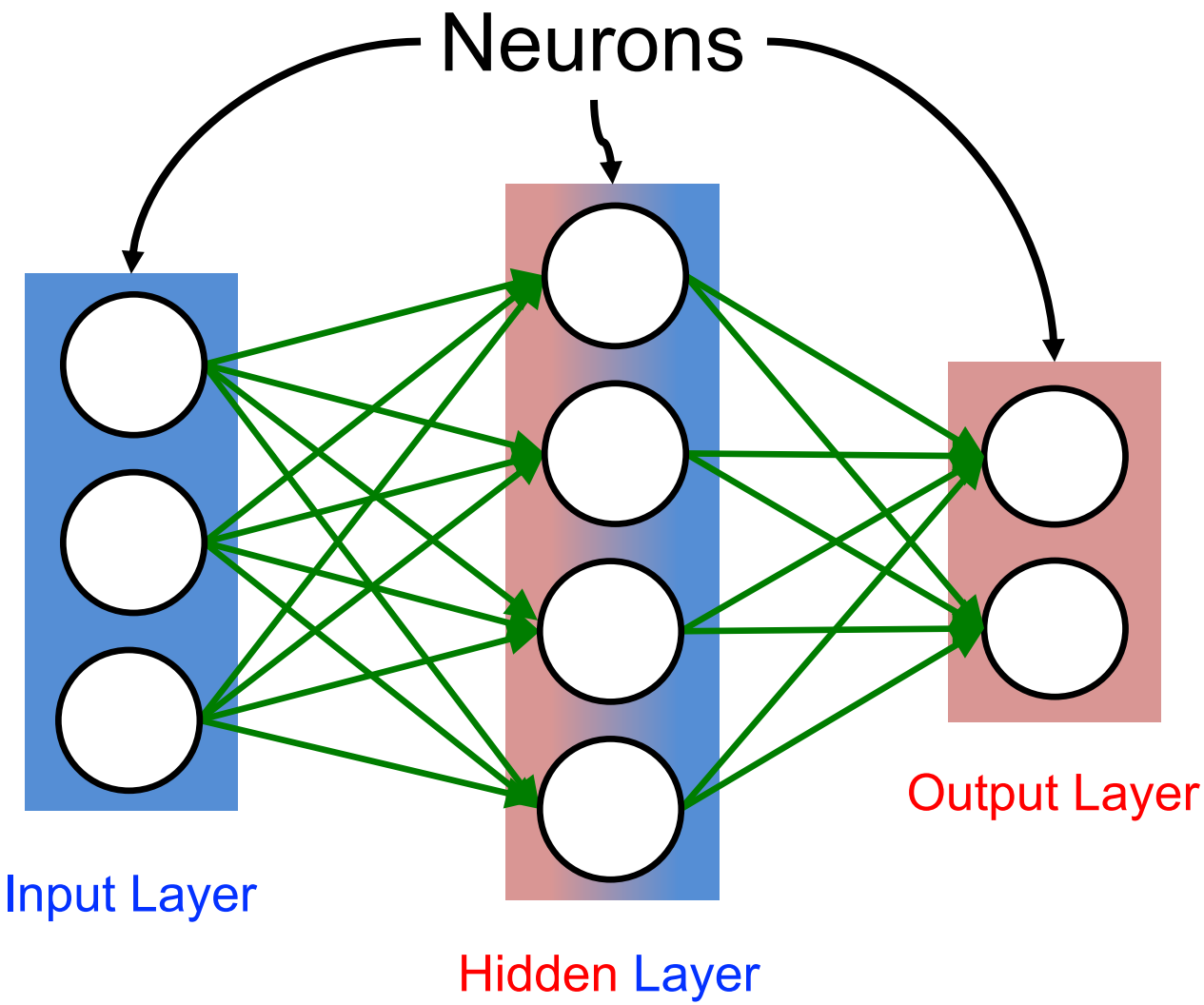
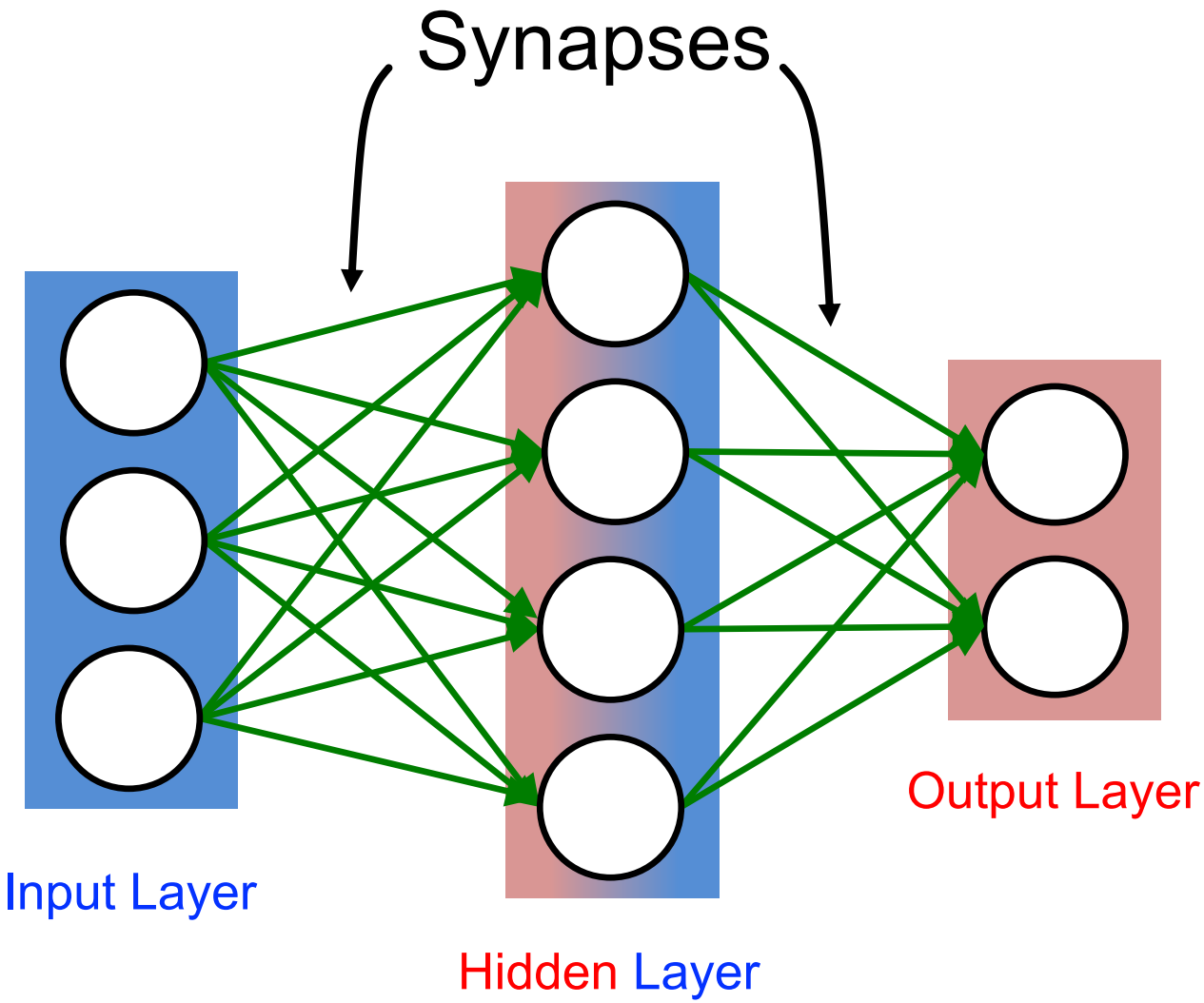


Image Source: Stanford

DNN Terminology 101



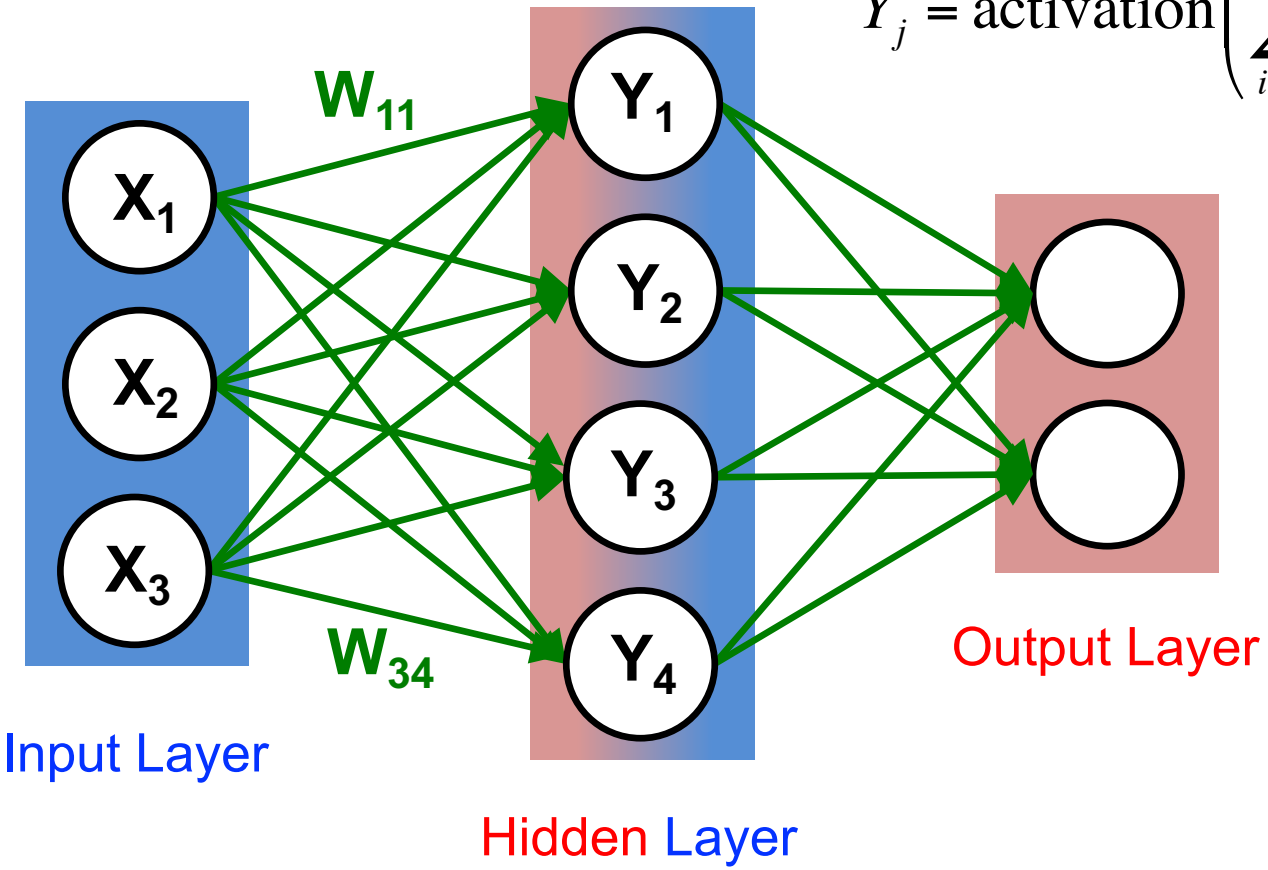
DNN Terminology 101



DNN Terminology 101

Each synapse has a **weight** for neuron **activation**

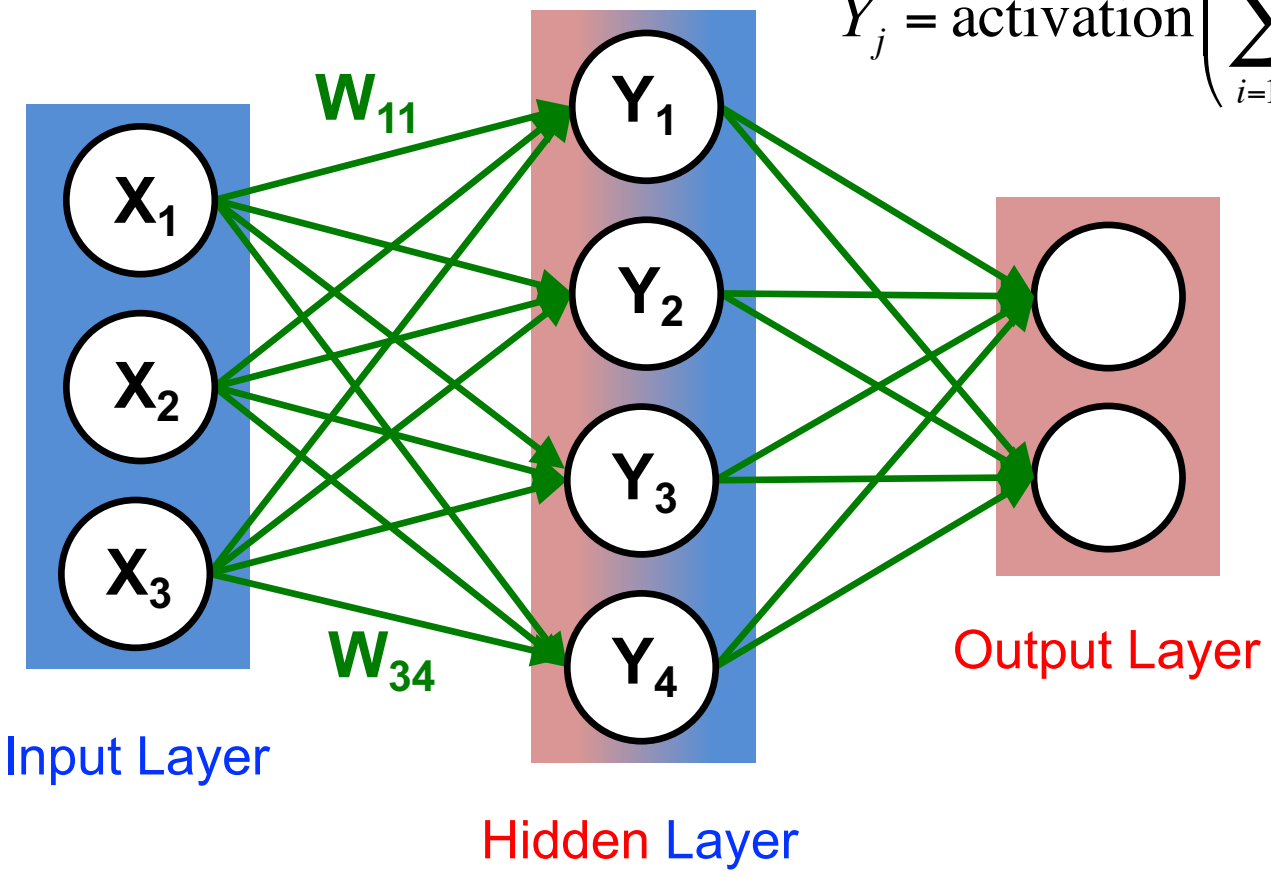
$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



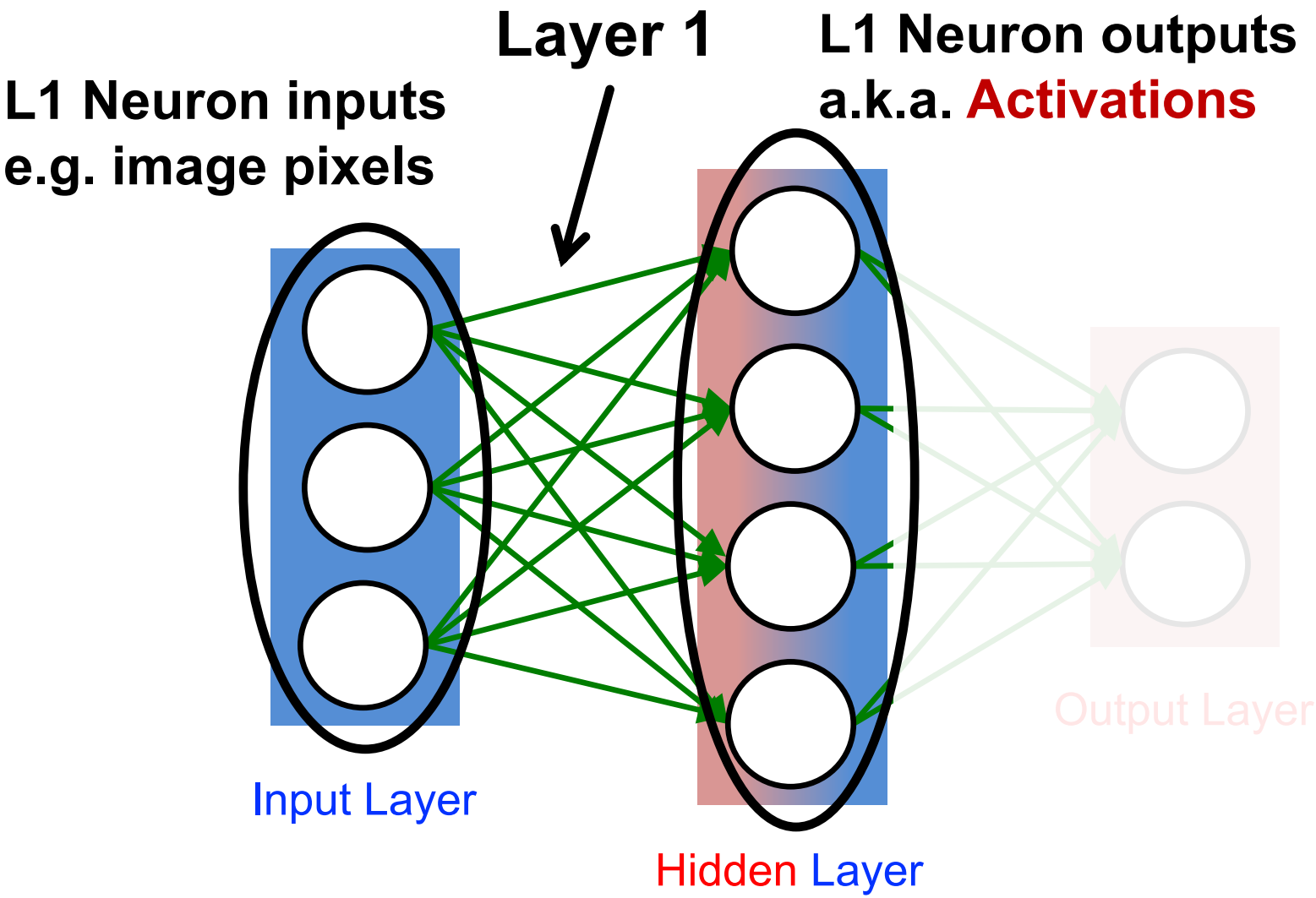
DNN Terminology 101

Weight Sharing: multiple synapses use the **same weight value**

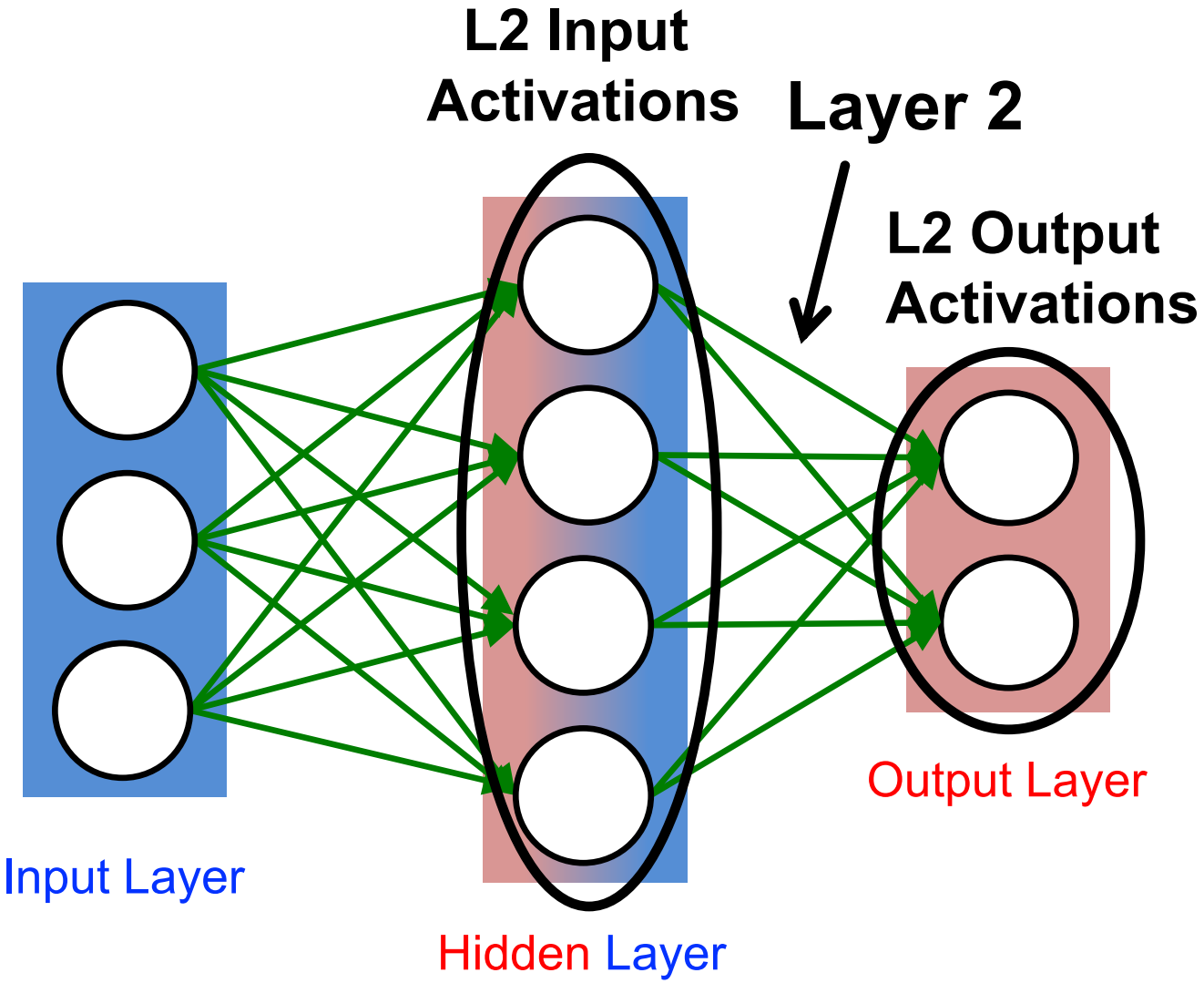
$$Y_j = \text{activation}\left(\sum_{i=1}^{3} W_{ij} \times X_i\right)$$



DNN Terminology 101

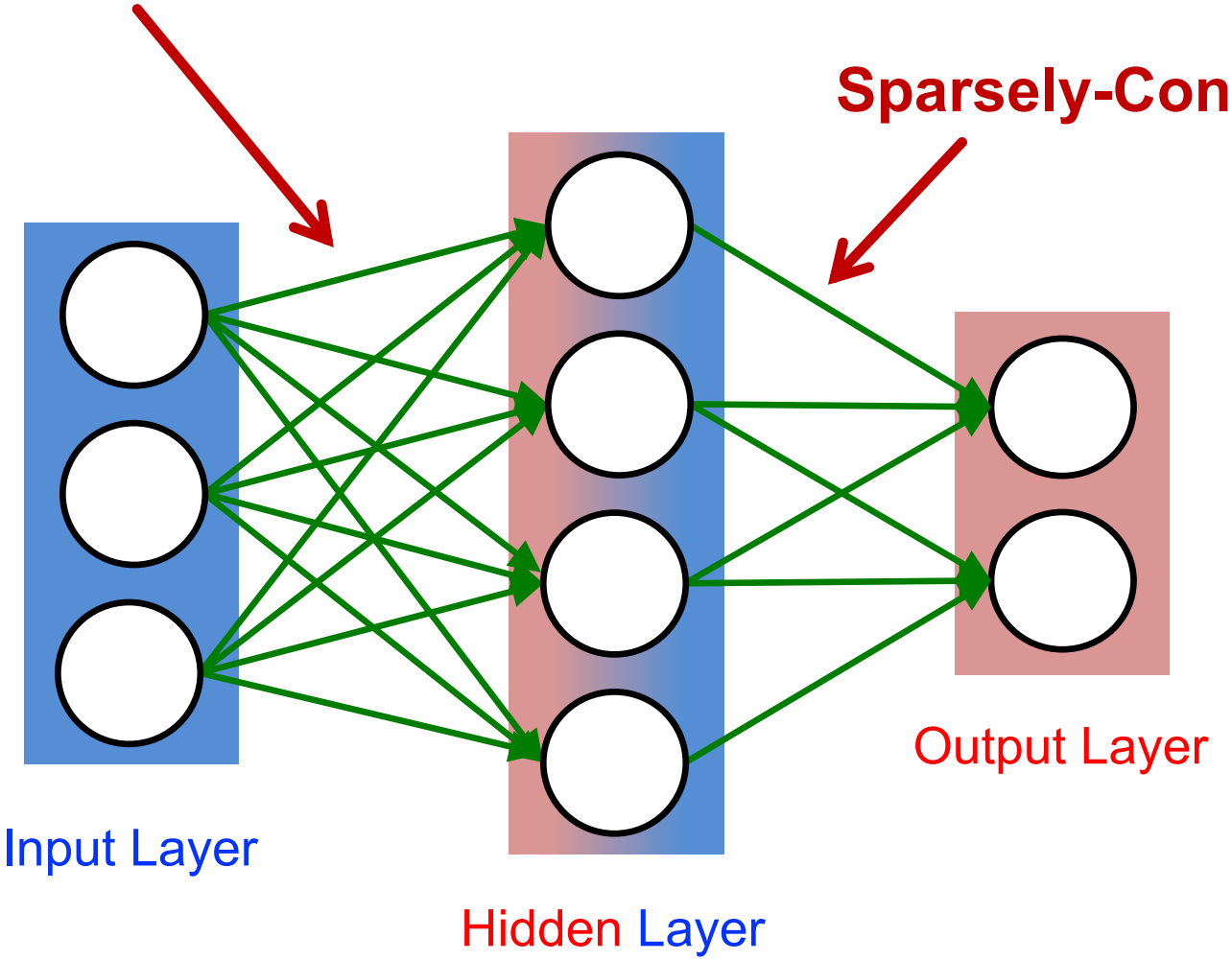


DNN Terminology 101

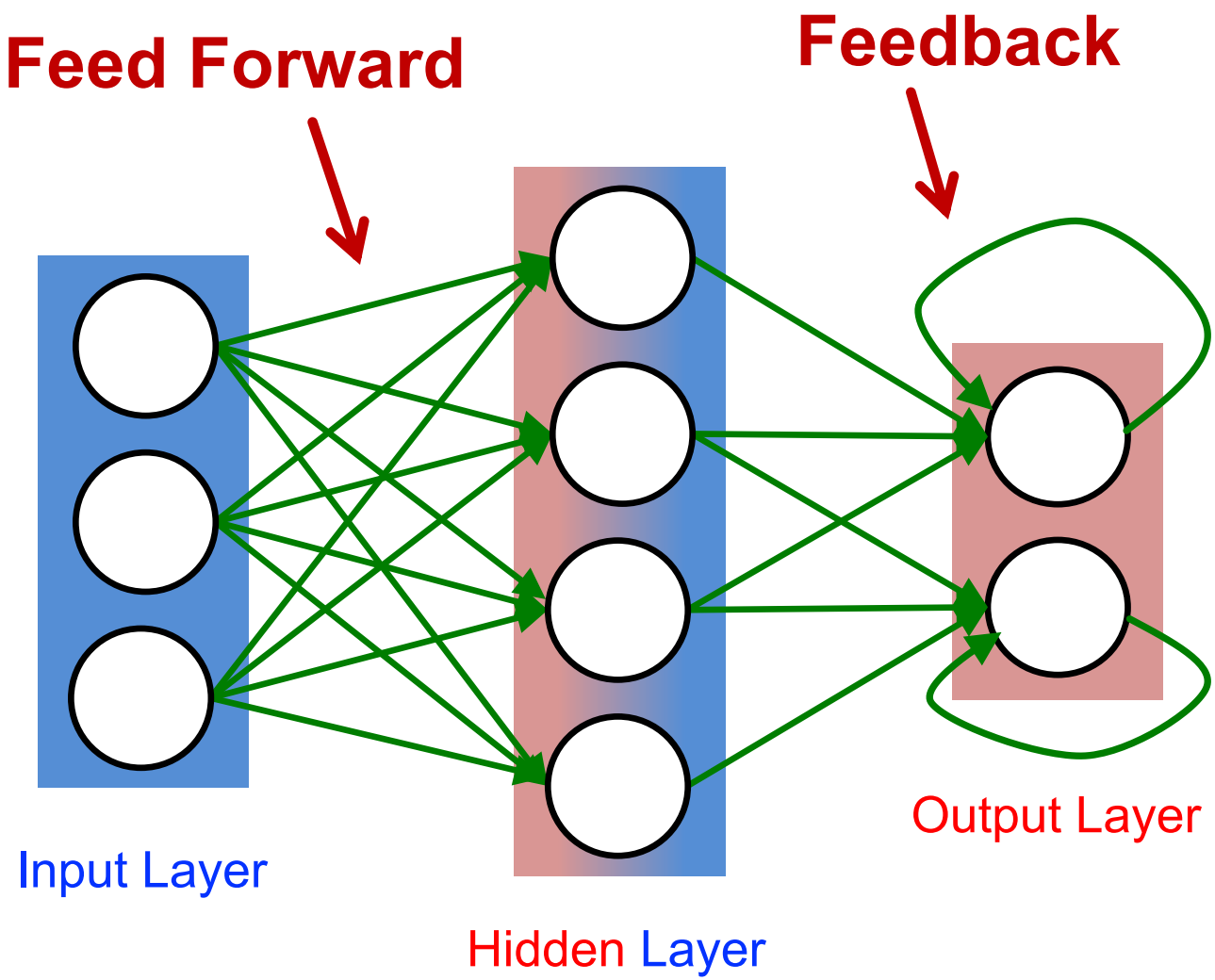


DNN Terminology 101

Fully-Connected: all i/p neurons connected to all o/p neurons



DNN Terminology 101



Popular Types of DNNs

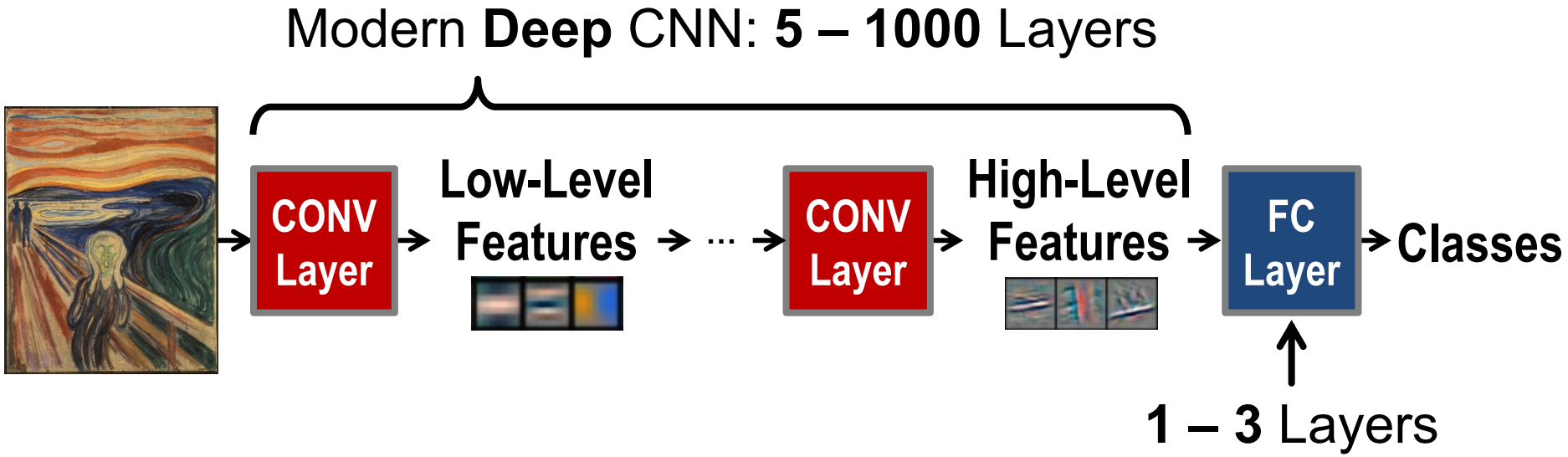
- **Fully-Connected NN**
  - feed forward, a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  - feed forward, sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  - feedback
- **Long Short-Term Memory (LSTM)**
  - feedback + storage

Inference vs. Training

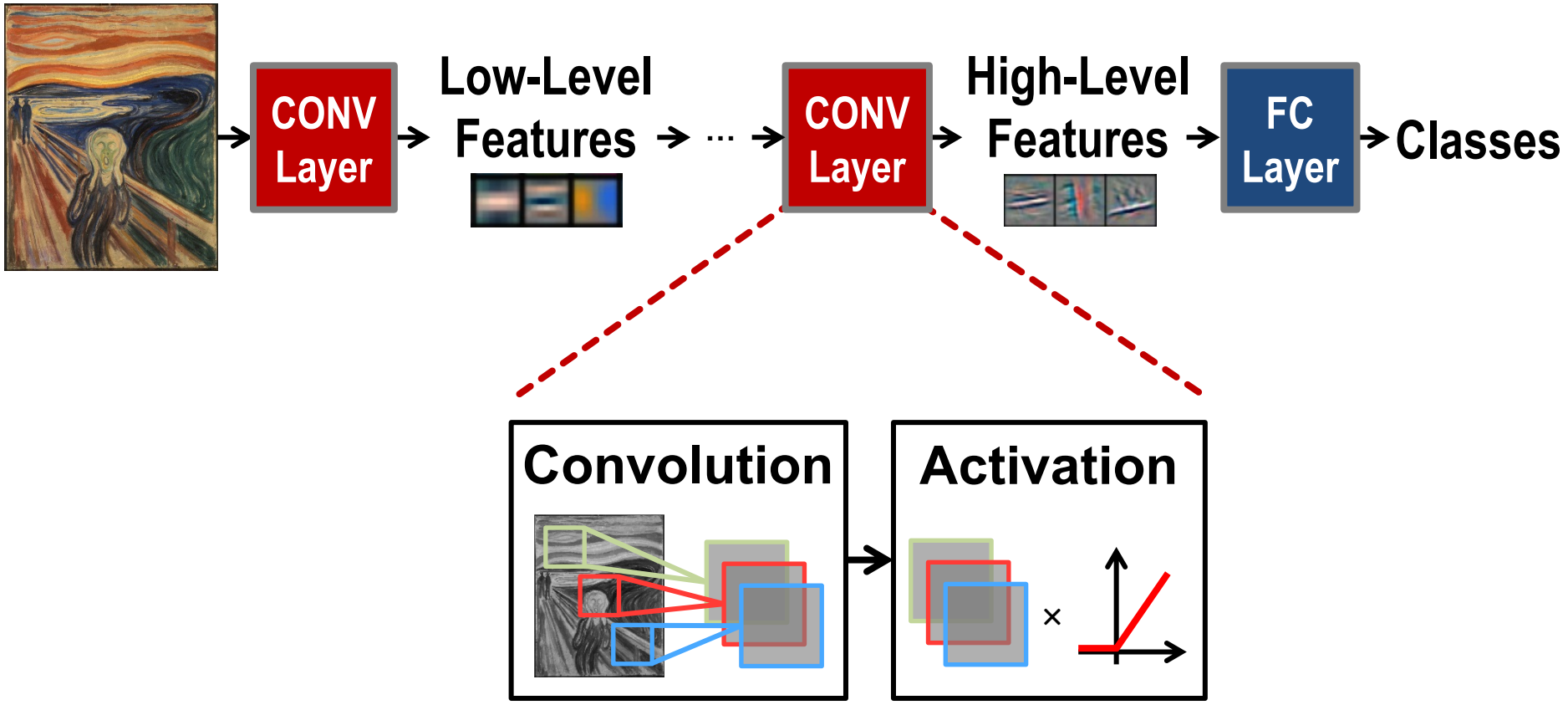
- **Training:** Determine weights
  - **Supervised:**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised / Self-Supervised:**
    - Training set is unlabeled
  - **Semi-supervised:**
    - Training set is partially labeled
  - **Reinforcement:**
    - Output assessed via rewards and punishments

- **Inference:** Apply weights to determine output

Deep Convolutional Neural Networks

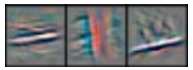


Deep Convolutional Neural Networks



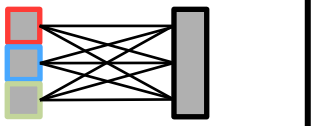
Deep Convolutional Neural Networks

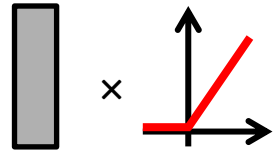
CONV Layer → Low-Level Features → ··· → CONV Layer → High-Level Features → FC Layer → Classes



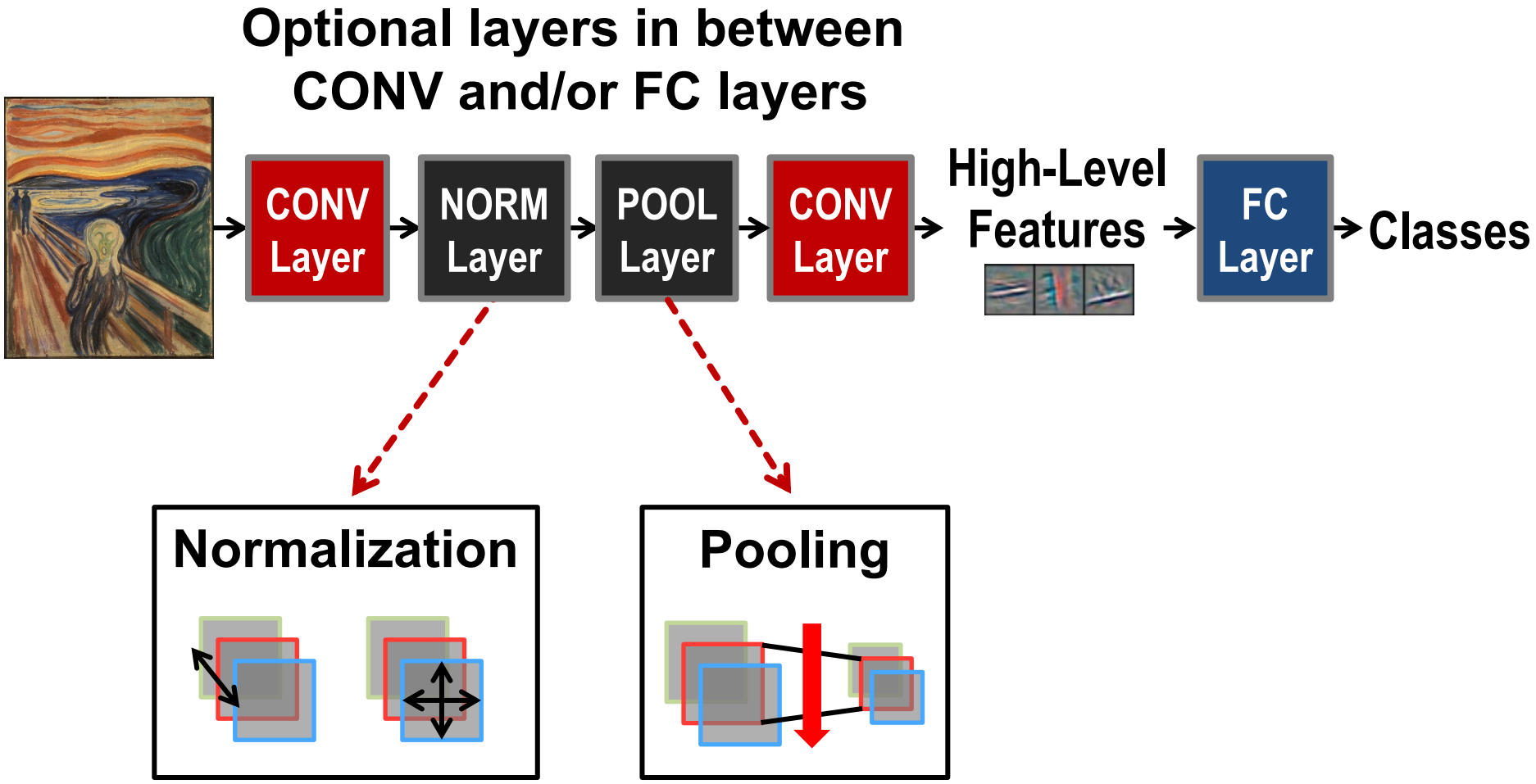
Fully Connected

Activation



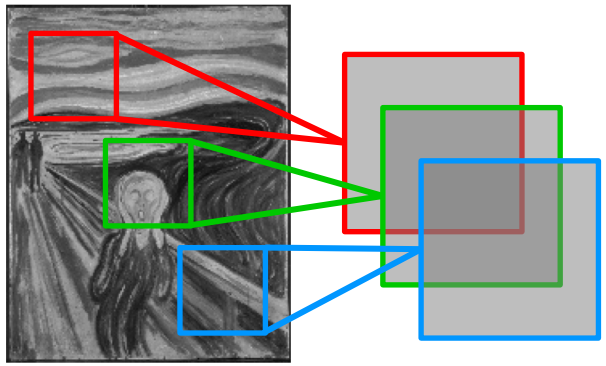


Deep Convolutional Neural Networks



Deep Convolutional Neural Networks

CONV Layer → NORM Layer → POOL Layer → CONV Layer → High-Level Features → FC Layer → Classes

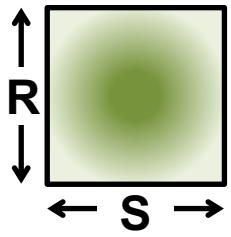


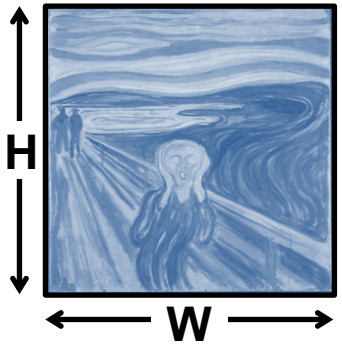
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

Convolution (CONV) Layer

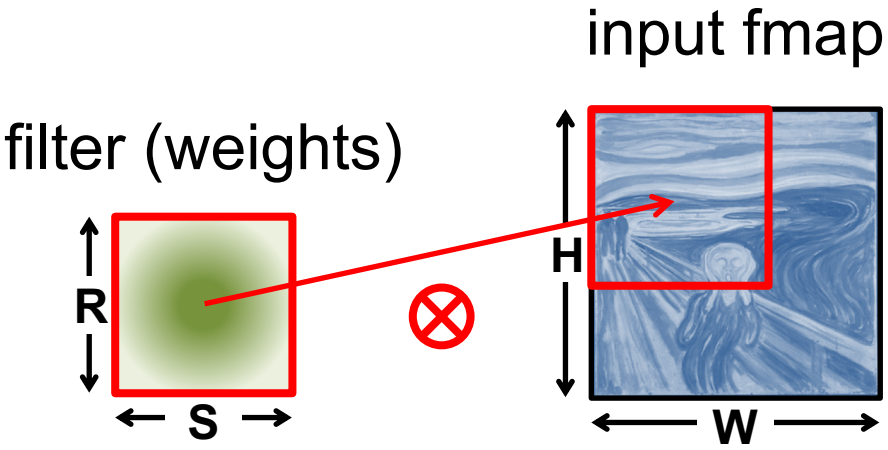
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)



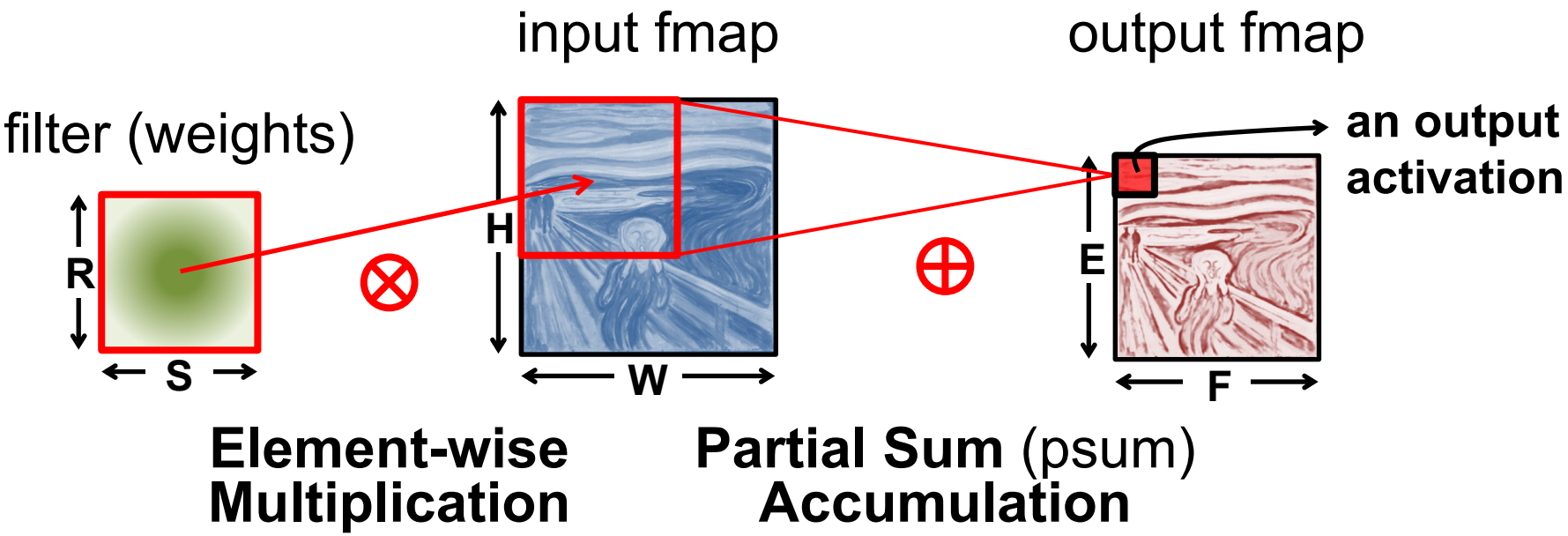


Convolution (CONV) Layer

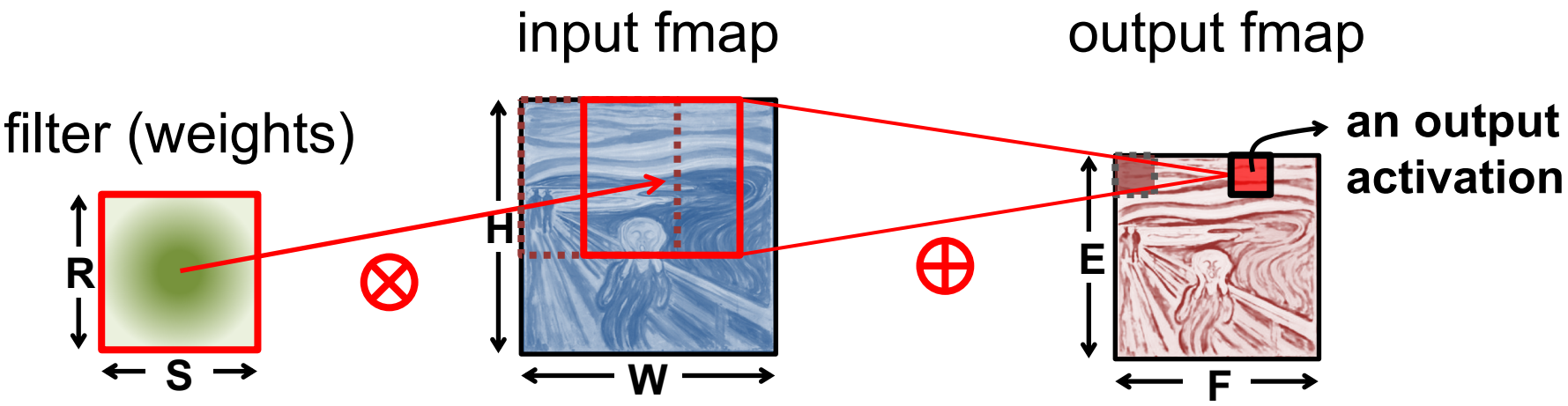


Element-wise Multiplication

Convolution (CONV) Layer

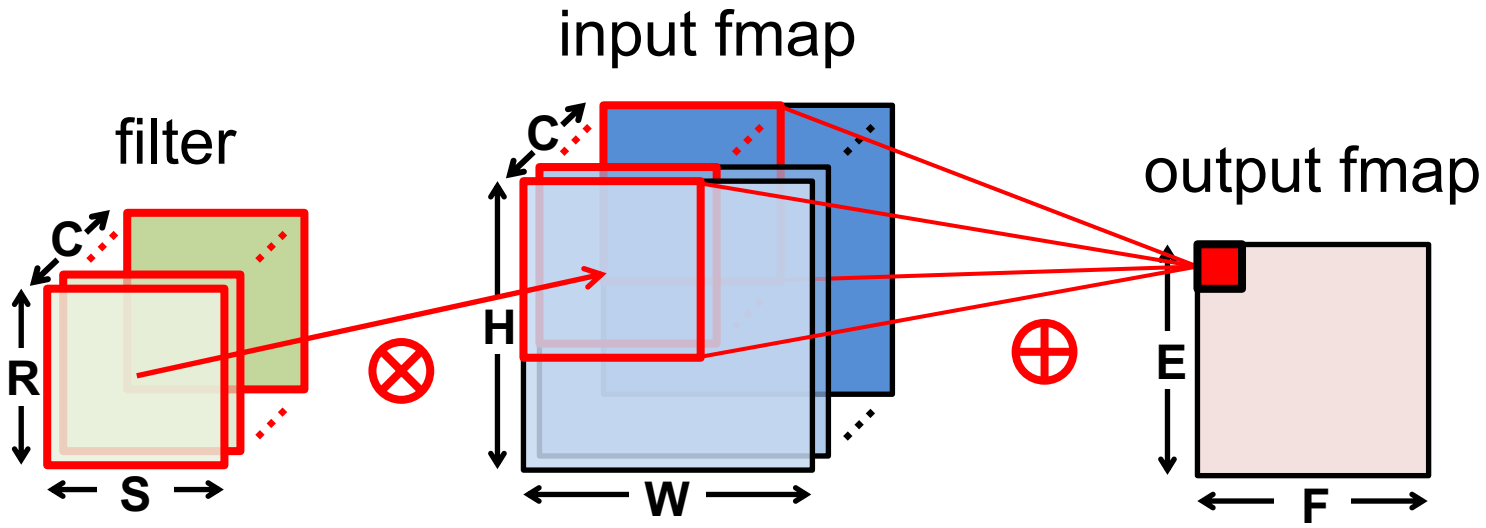


Convolution (CONV) Layer



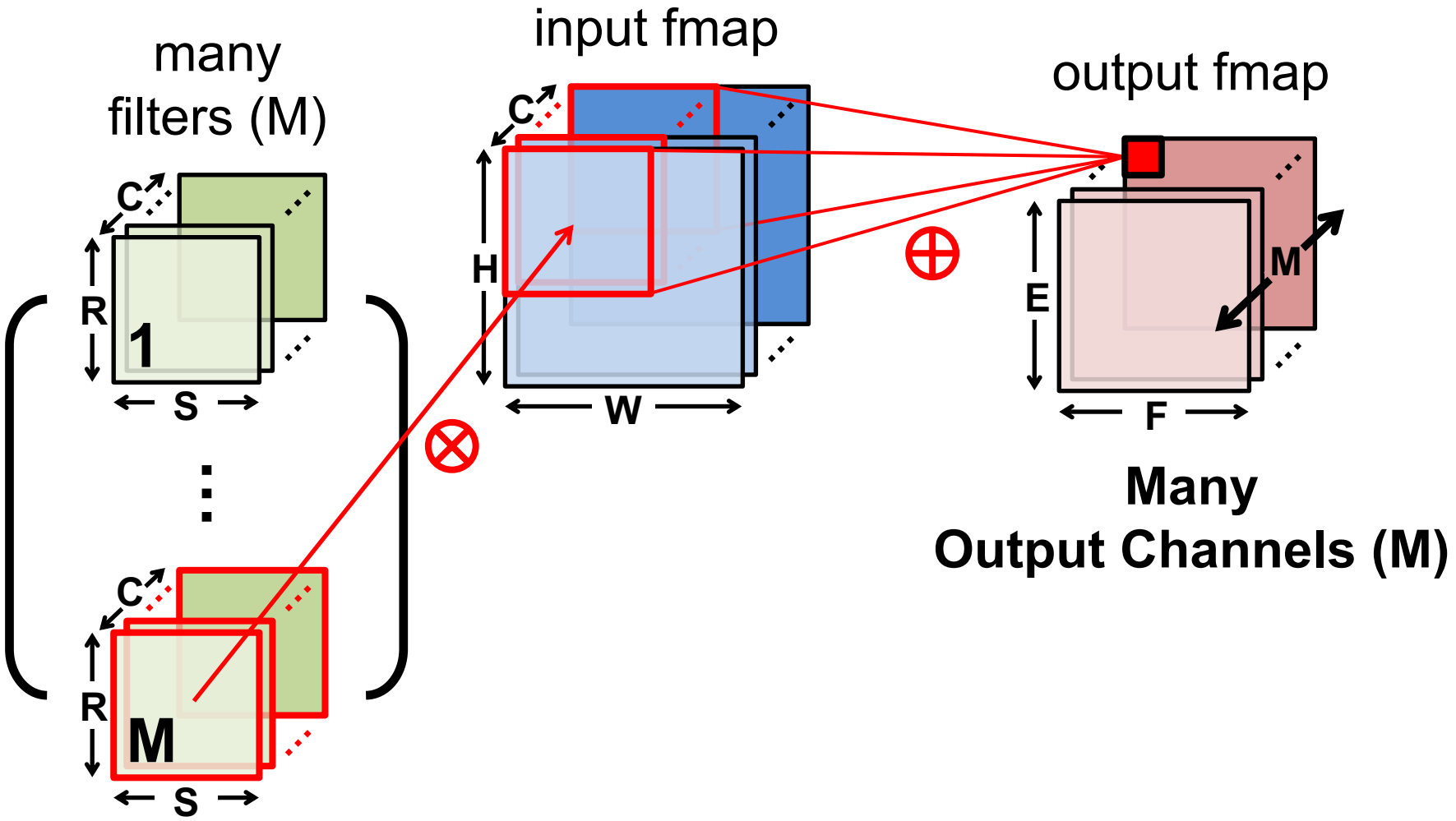
Sliding Window Processing

Convolution (CONV) Layer

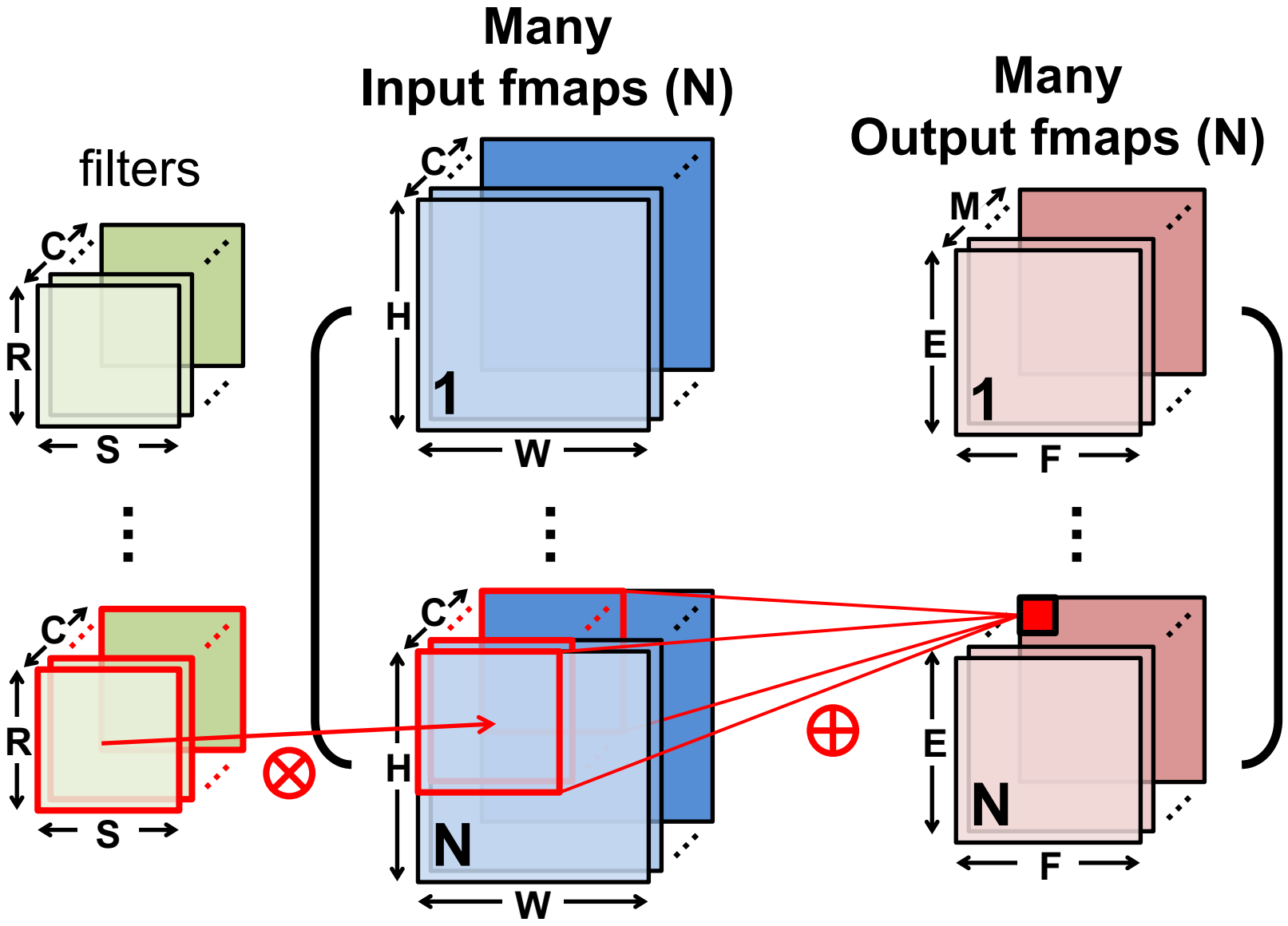


Many Input Channels (C)

Convolution (CONV) Layer



Convolution (CONV) Layer



CNN Decoder Ring

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of 2-D input fmaps /filters (channels)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- R – Height of 2-D filter (weights)
- S – Width of 2-D filter (weights)
- M – Number of 2-D output fmaps (channels)
- E – Height of output fmap (activations)
- F – Width of output fmap (activations)

CONV Layer Tensor Computation

Output fmaps (O)

Biases (B)

Input fmaps (I)

Filter weights (W)

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

$$0 \leq n < N, 0 \leq m < M, 0 \leq y < E, 0 \leq x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

CONV Layer Implementation

Naïve 7-layer for-loop implementation:

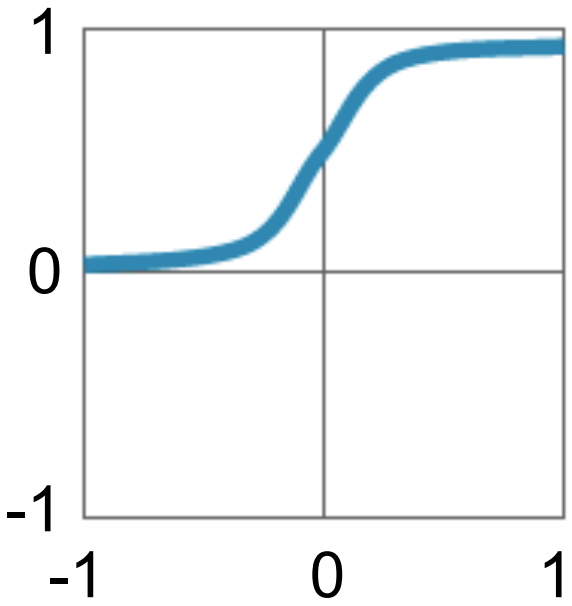
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

for each output fmap value

convolve a window and apply activation

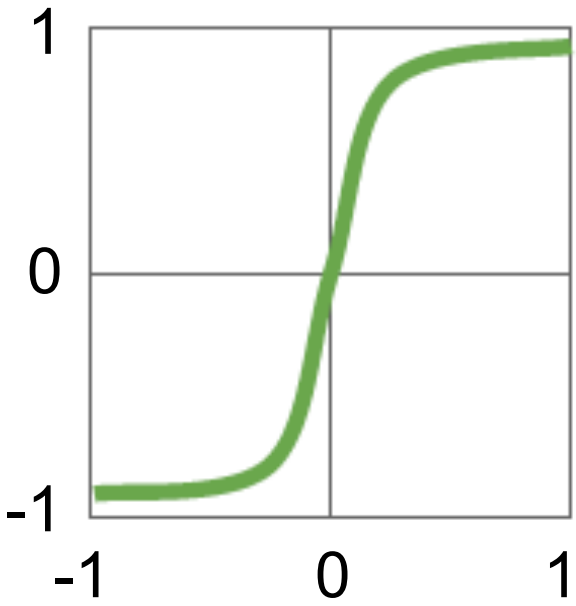
Traditional Activation Functions

Sigmoid



$$y=1/(1+e^{-x})$$

Hyperbolic Tangent

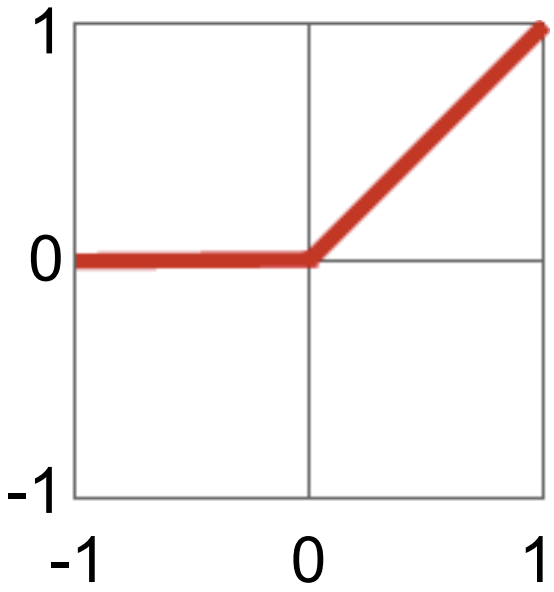


$$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$$

Image Source: Caffe Tutorial

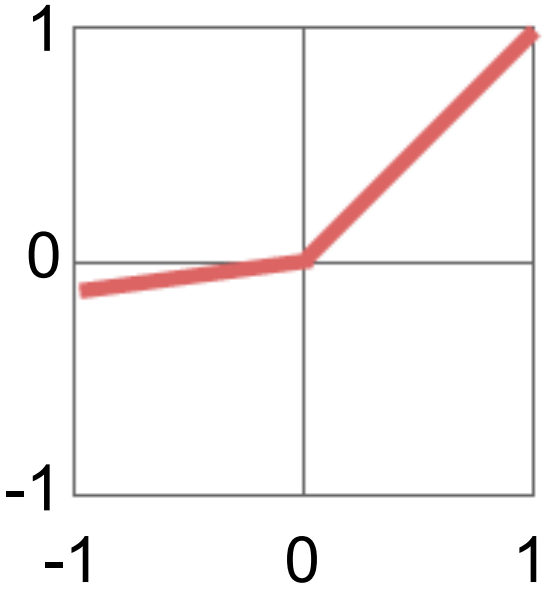
Modern Activation Functions

Rectified Linear Unit (ReLU)



$y = \max(0, x)$

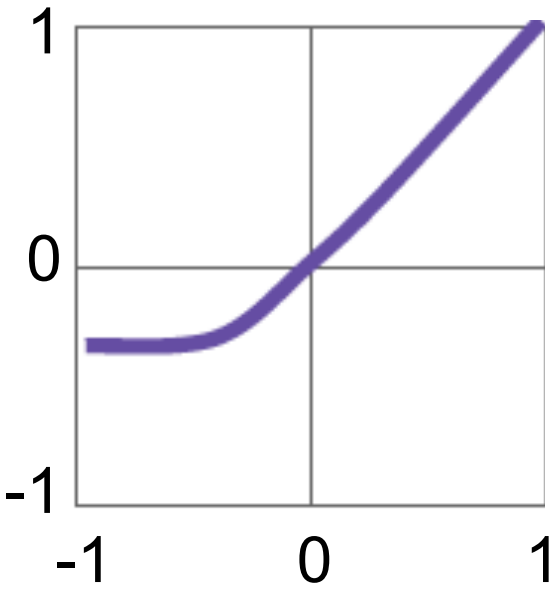
Leaky ReLU



$y = \max(\alpha x, x)$

α = small const. (e.g. 0.1)

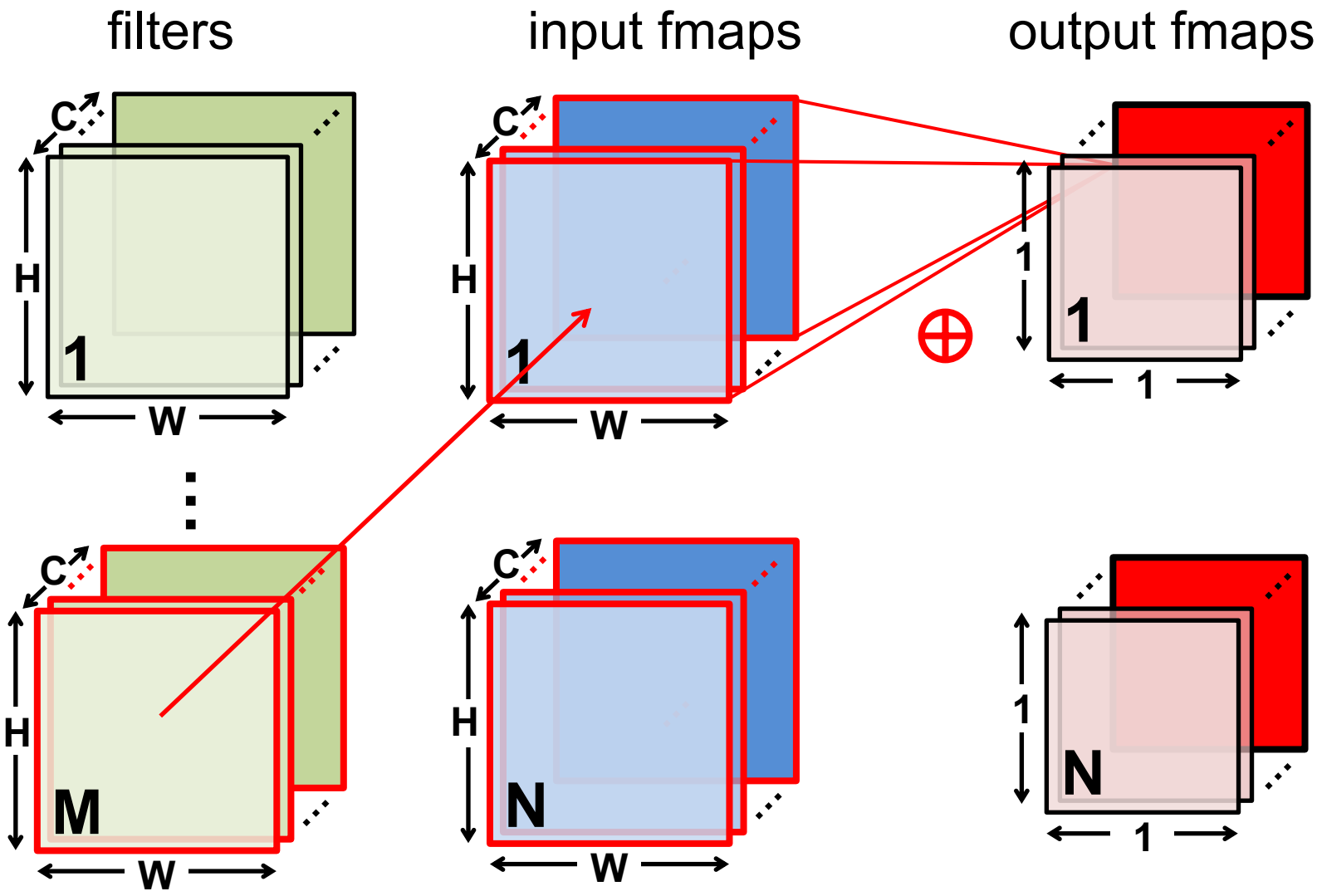
Exponential LU



$$y = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

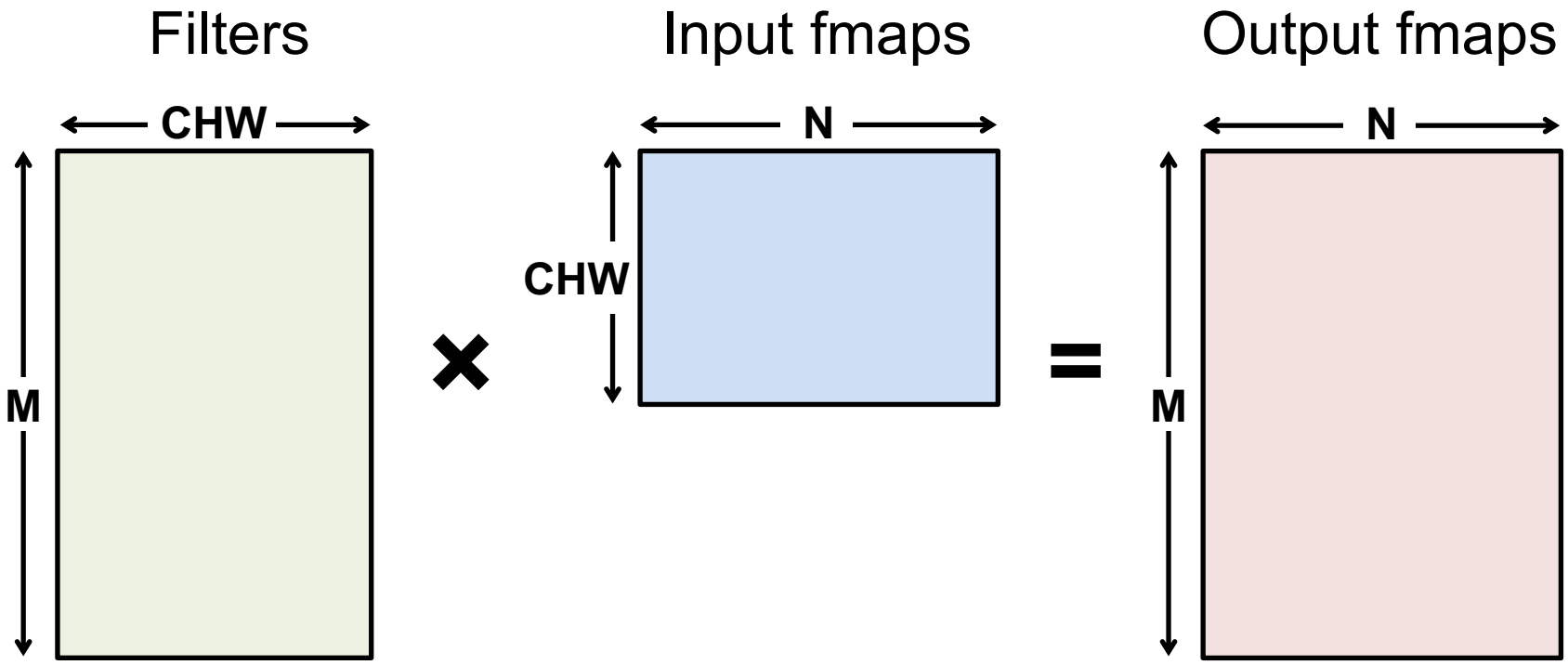
Image Source: Caffe Tutorial

FC Layer – from CONV Layer POV



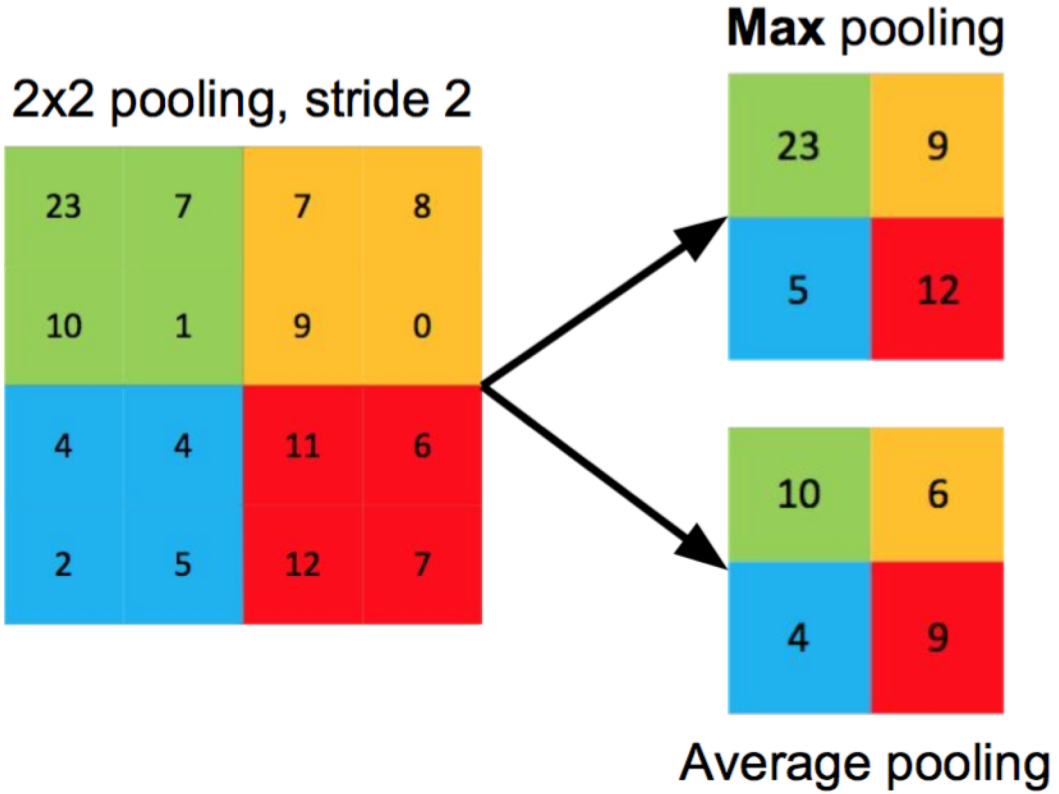
Fully-Connected (FC) Layer

- Height and width of output fmaps are 1 (E = F = 1)
- Filters as large as input fmaps (R = H, S = W)
- Implementation: **Matrix Multiplication**



Pooling (POOL) Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping → depending on stride



Increases translation-invariance and noise-resilience

Image Source: Caffe Tutorial

POOL Layer Implementation

Naïve 6-layer for-loop max-pooling implementation:

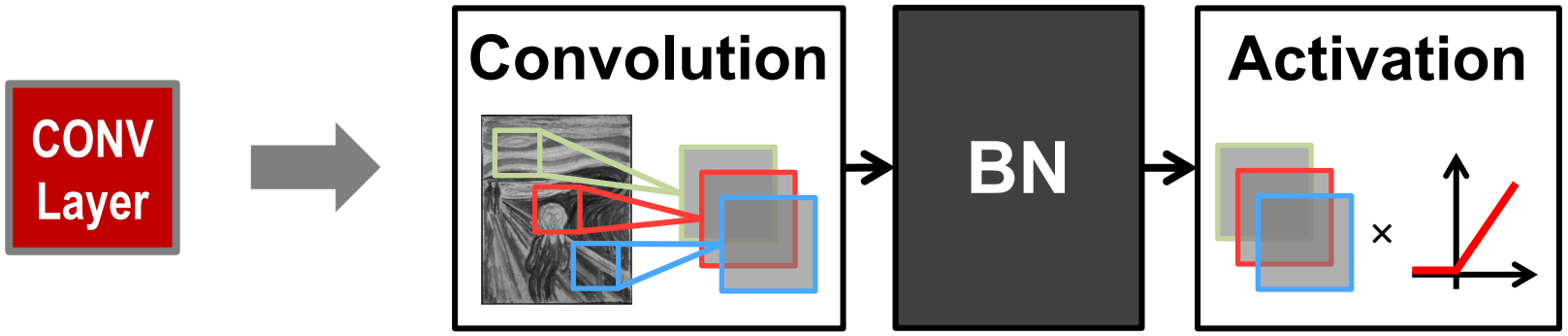
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {

                max = -Inf;
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        if (I[n][m][Ux+i][Uy+j] > max) {
                            max = I[n][m][Ux+i][Uy+j];
                        }
                    }
                }

                O[n][m][x][y] = max;
            }
        }
    }
}
```

for each pooled value

find the max with in a window

Normalization (NORM) Layer

- **Batch Normalization (BN)**
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
  - put **in between** **CONV/FC** and **Activation function**

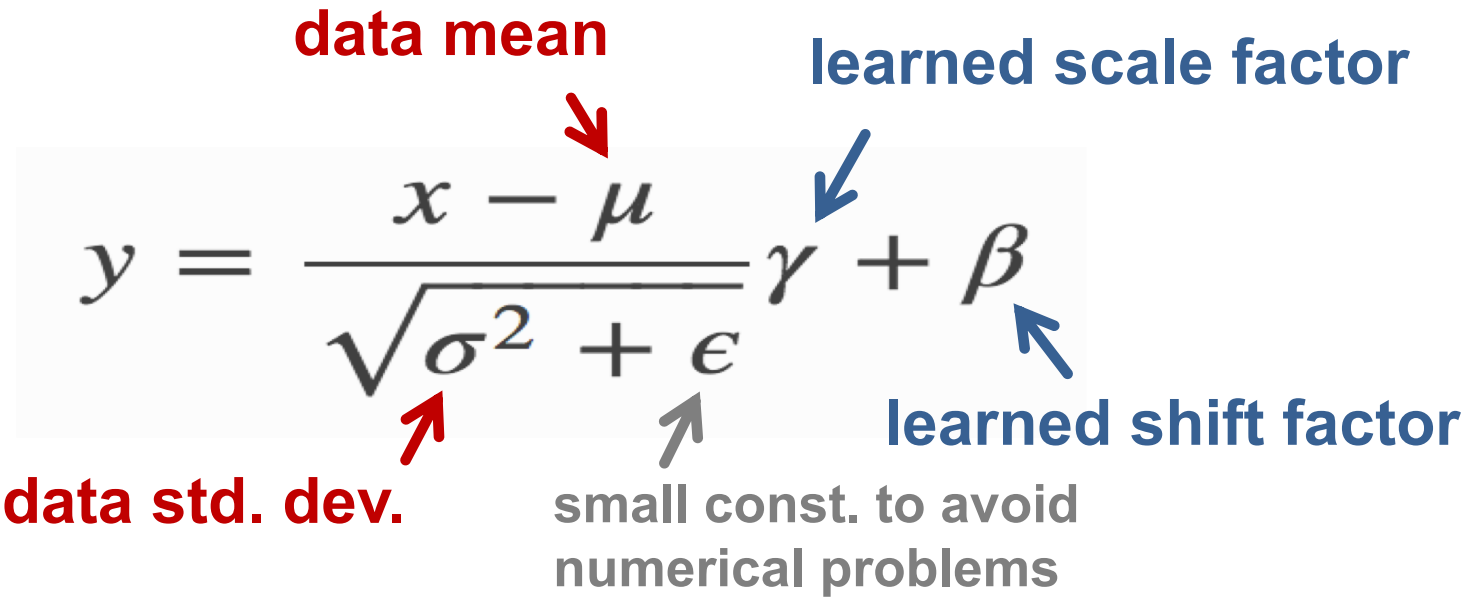


Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]

BN Layer Implementation

- The normalized value is further scaled and shifted, the parameters of which are learned from training



DNN Accelerator Architectures

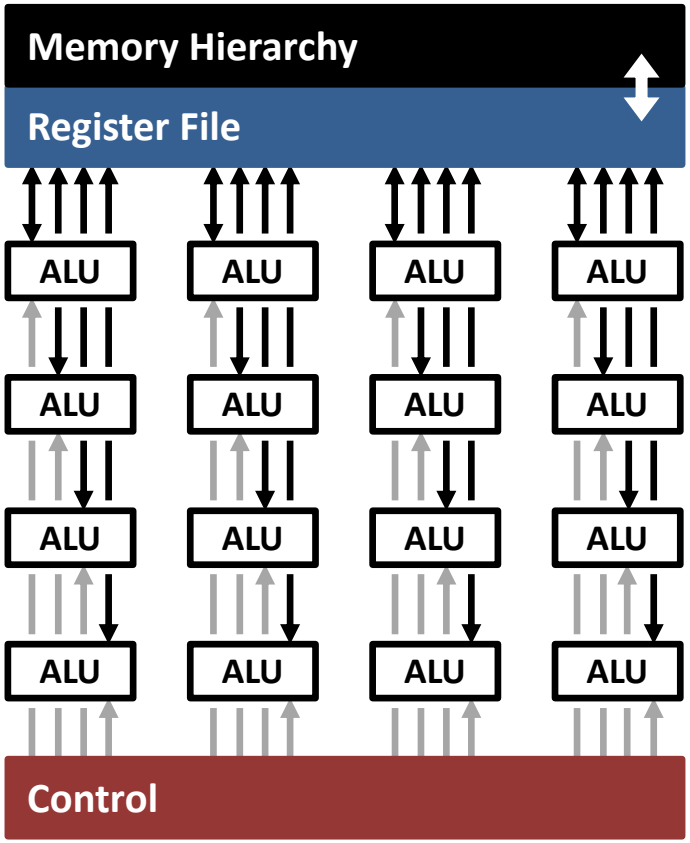
ISCA Tutorial (2019)

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

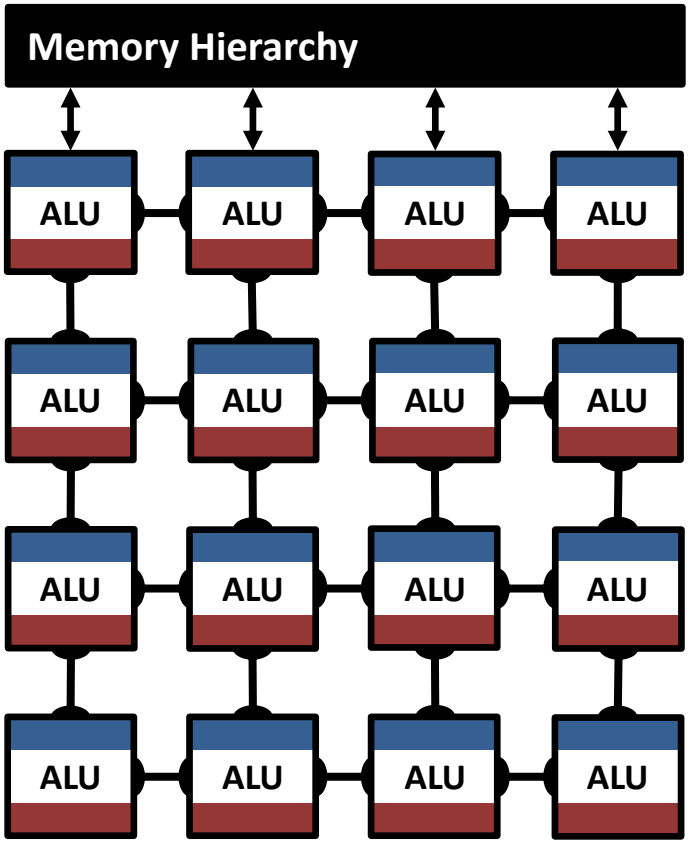
Joel Emer, Vivienne Sze, Yu-Hsin Chen

Highly-Parallel Compute Paradigms

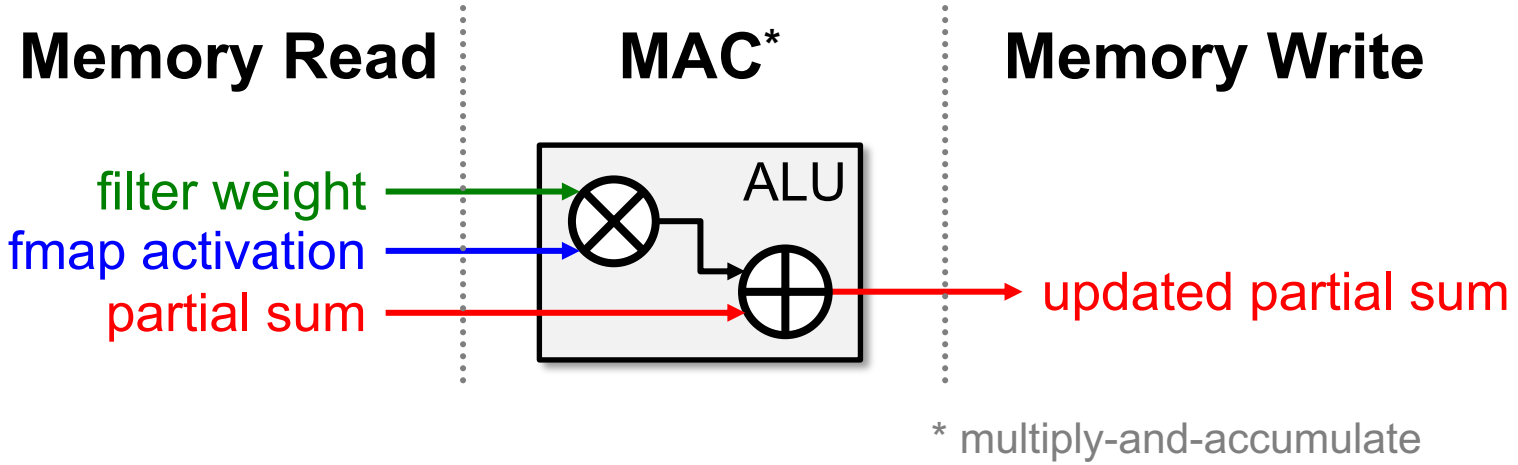
Temporal Architecture (SIMD/SIMT)



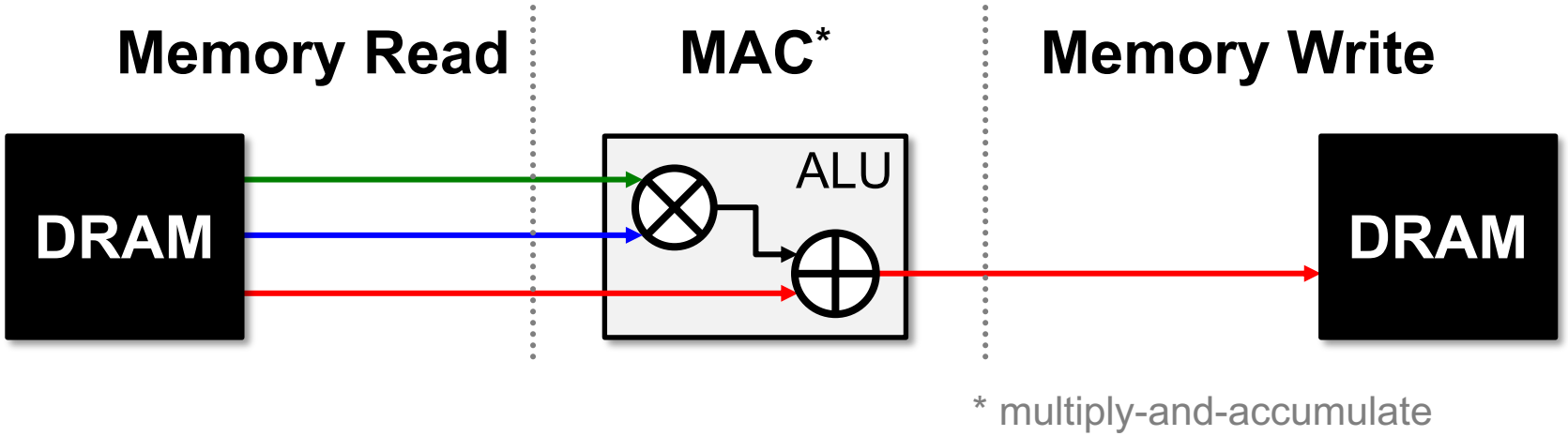
Spatial Architecture (Dataflow Processing)



Memory Access is the Bottleneck



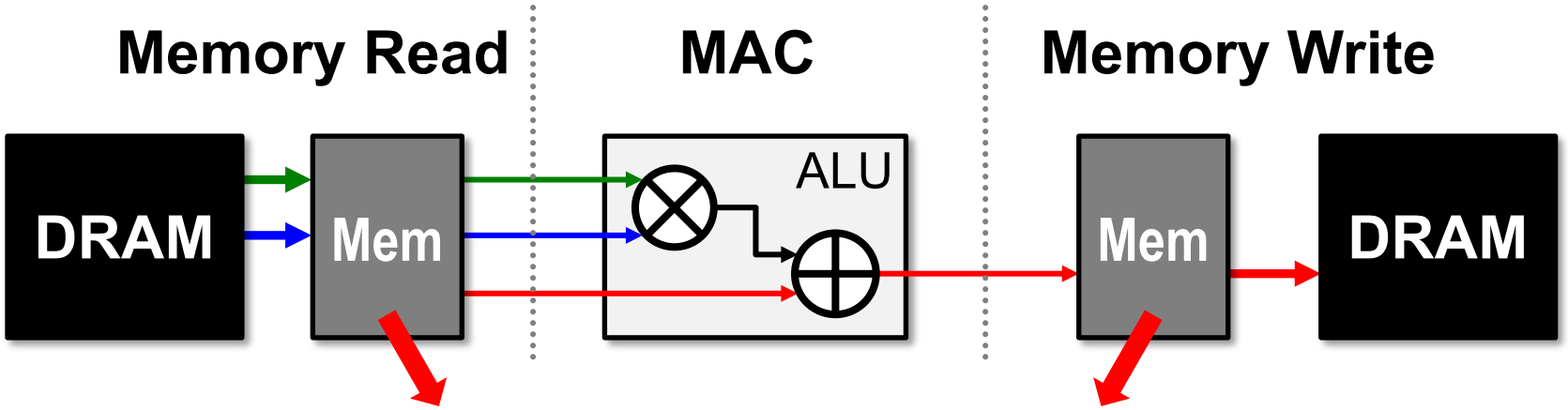
Memory Access is the Bottleneck



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

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

Leverage Local Memory for Data Reuse



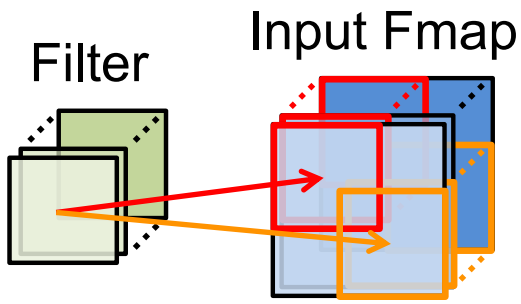
Extra levels of local memory hierarchy

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

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

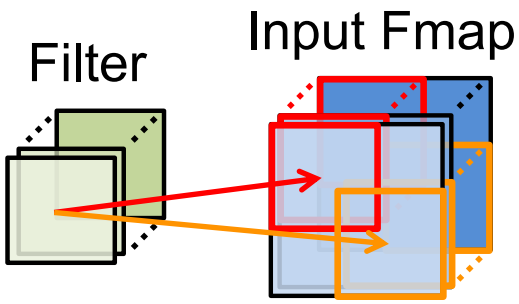


Reuse: Activations
Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

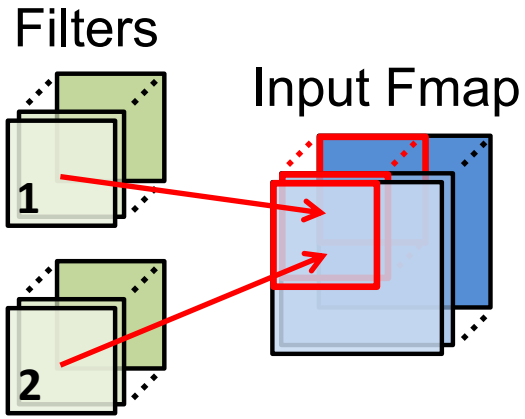
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

CONV and FC layers

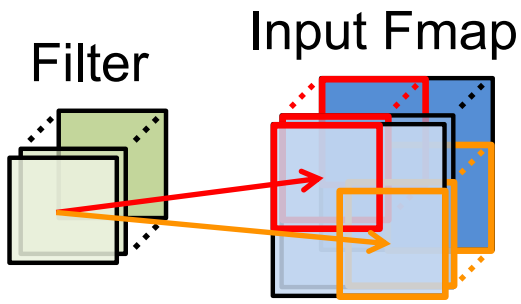


Reuse: Activations

Types of Data Reuse in DNN

Convolutional Reuse

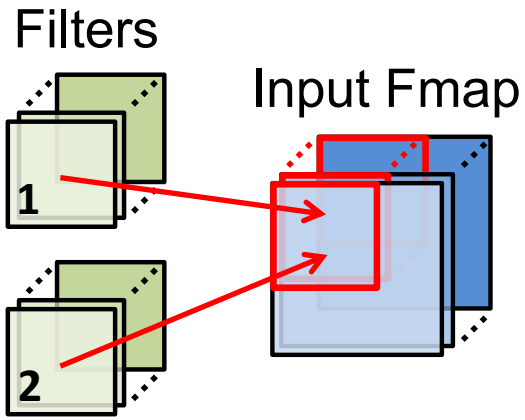
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

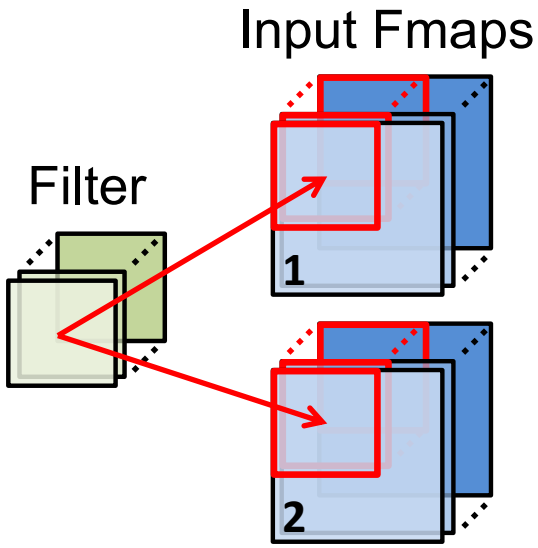
CONV and FC layers



Reuse: Activations

Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

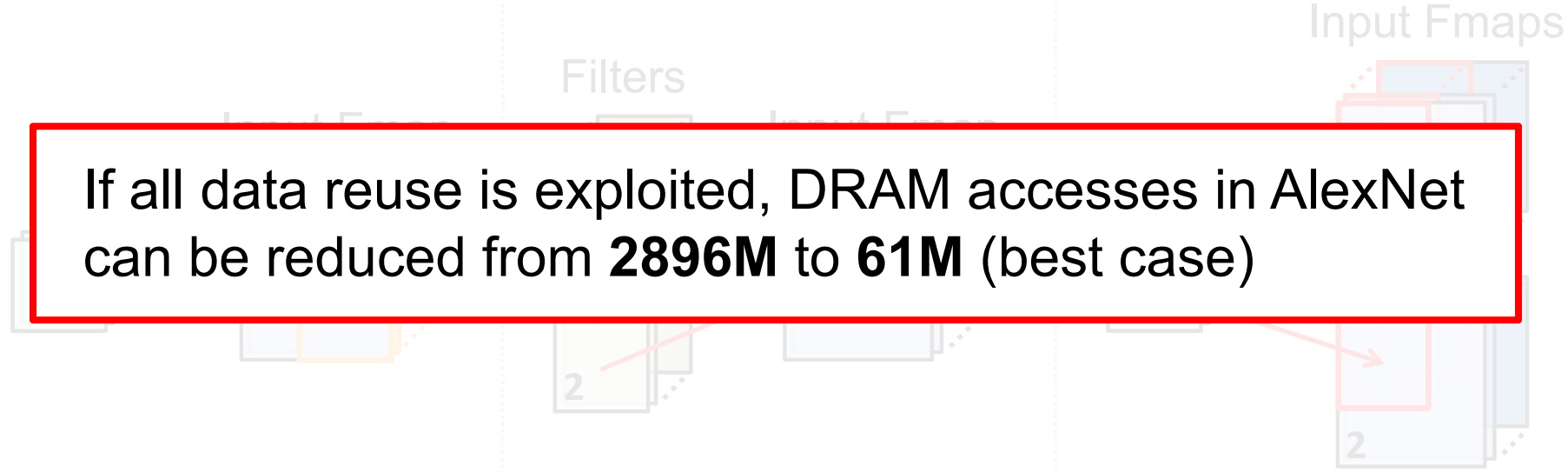
CONV layers only (sliding window)

Fmap Reuse

CONV and FC layers

Filter Reuse

CONV and FC layers (batch size > 1)



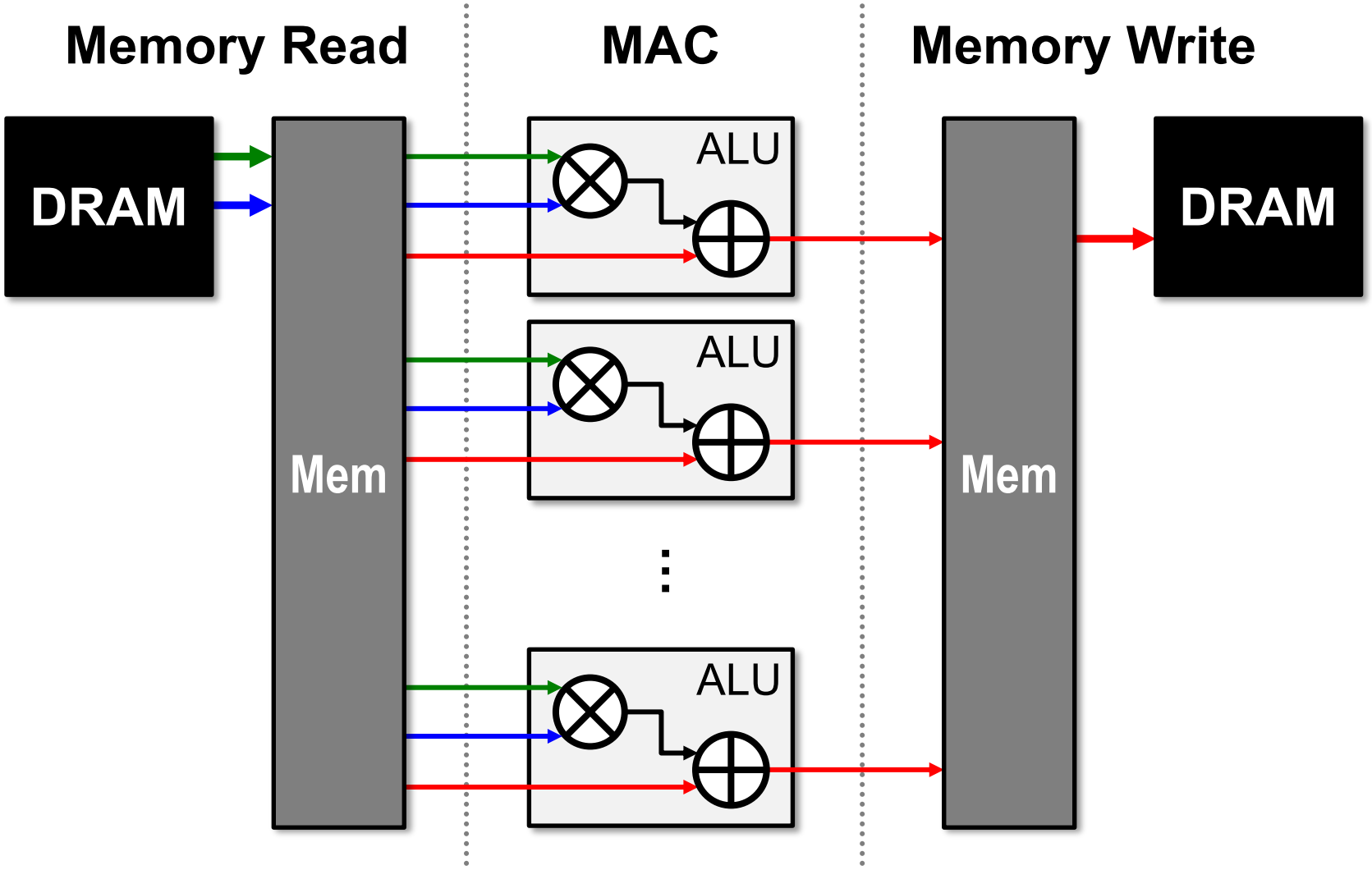
If all data reuse is exploited, DRAM accesses in AlexNet can be reduced from **2896M** to **61M** (best case)

Reuse: Activations, Filter weights

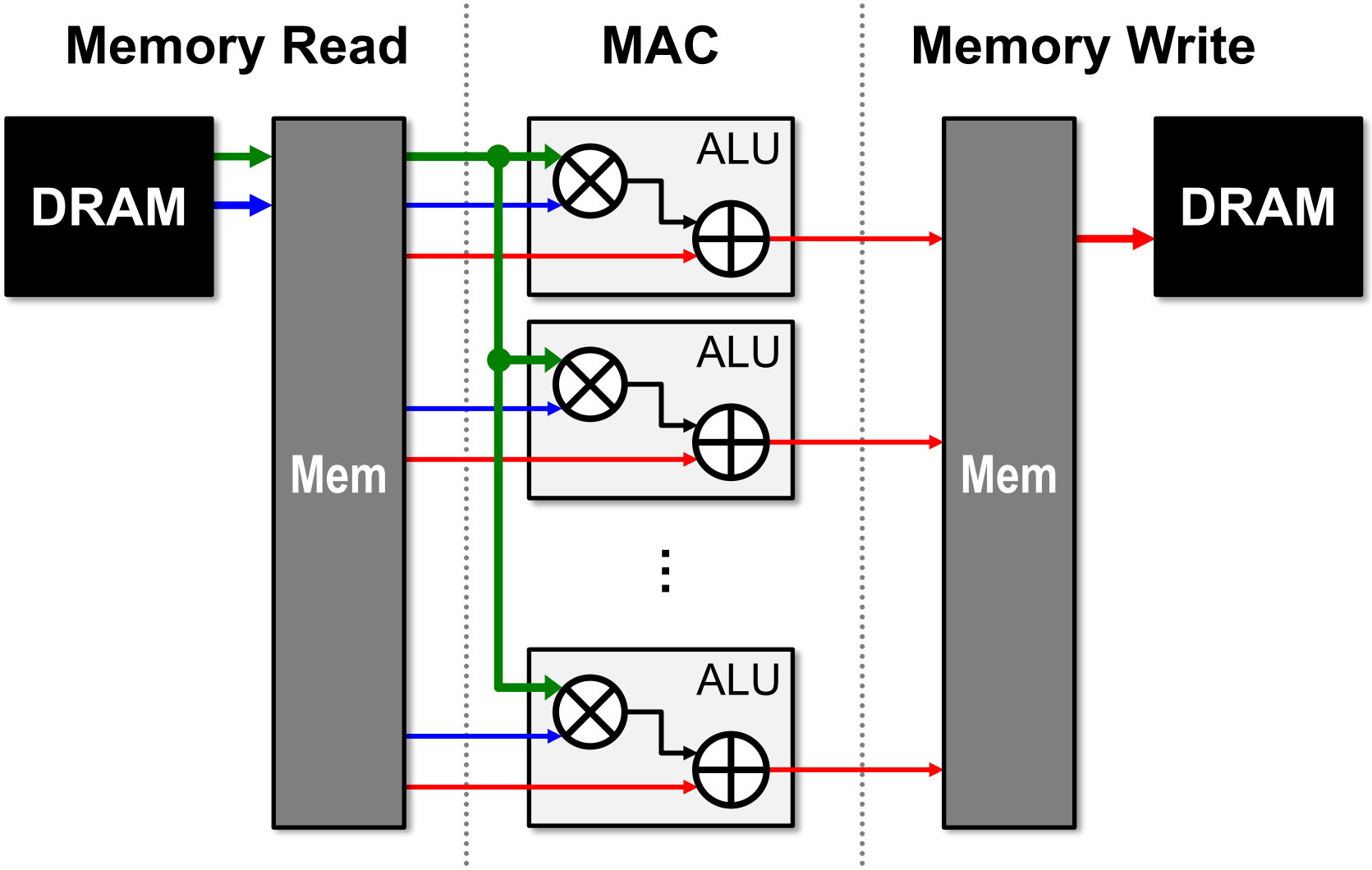
Reuse: Activations

Reuse: Filter weights

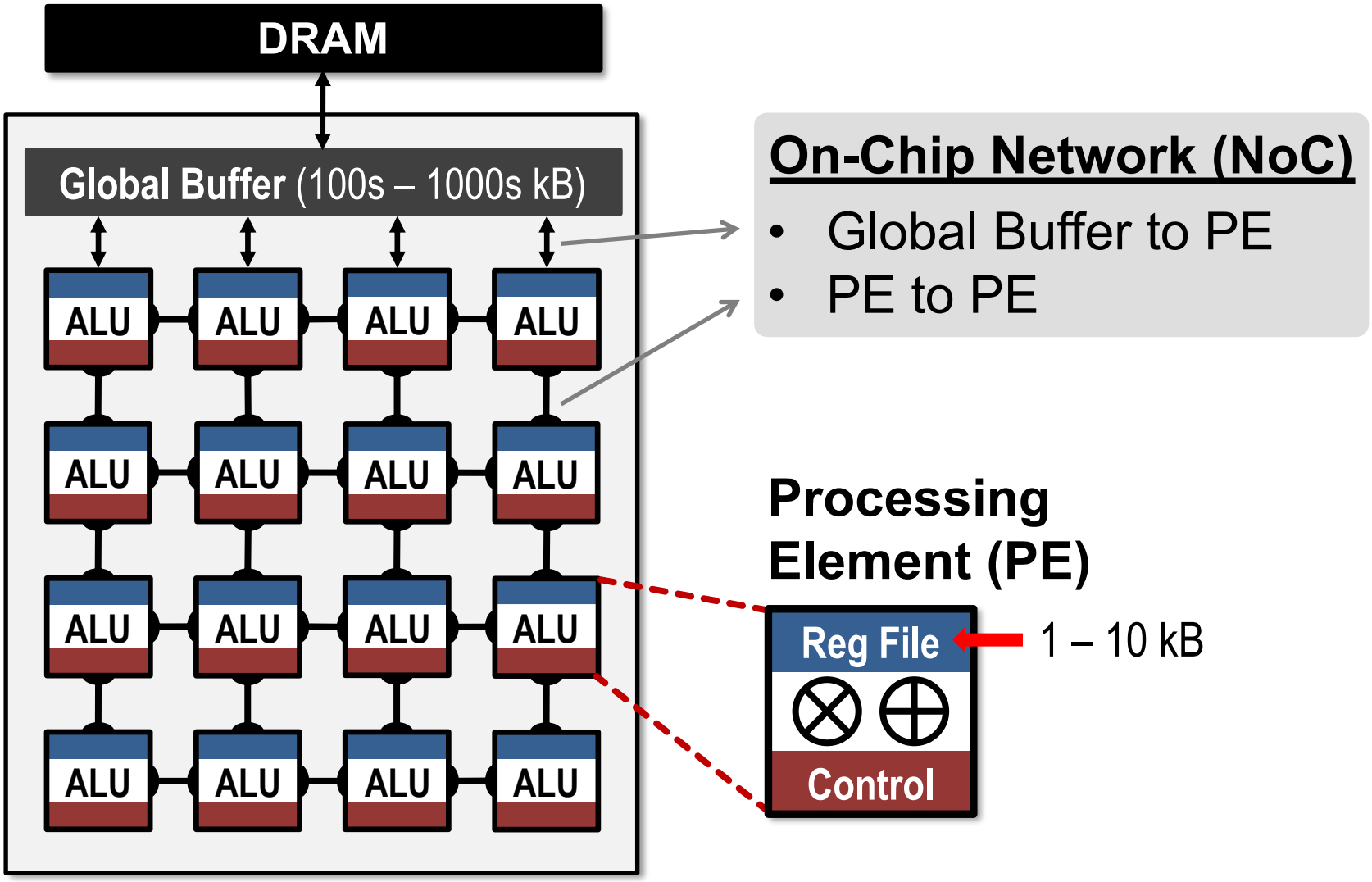
Leverage Parallelism for Higher Performance



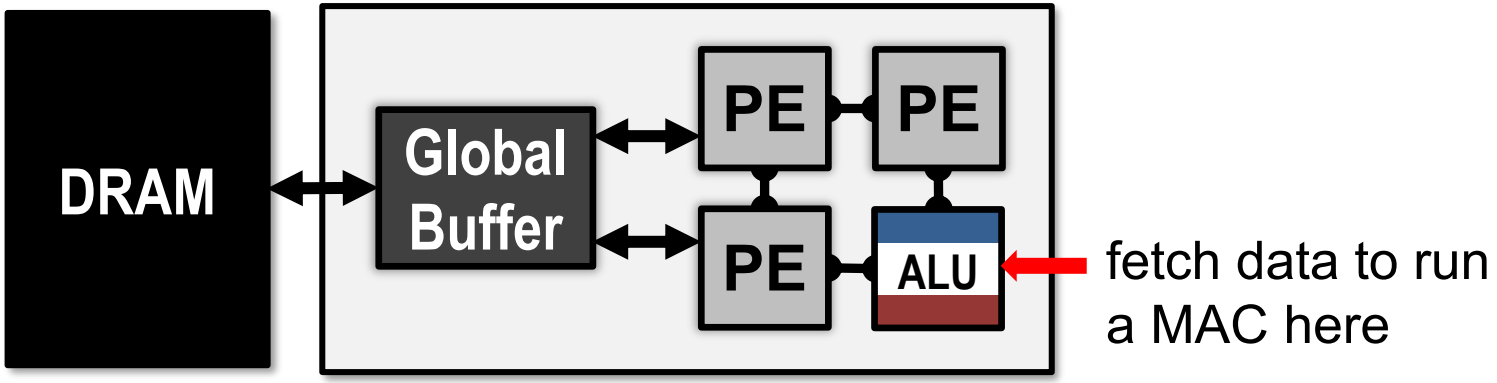
Leverage Parallelism for *Spatial* Data Reuse

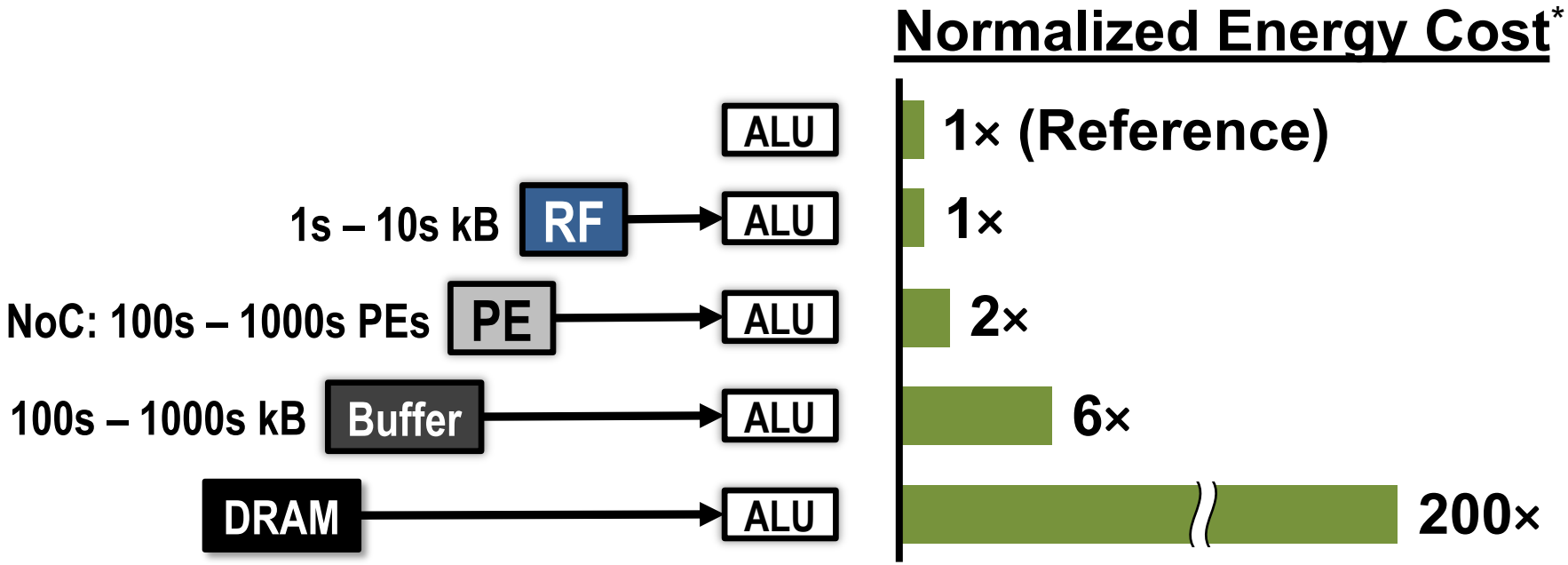


Spatial Architecture for DNN



Multi-Level Low-Cost Data Access

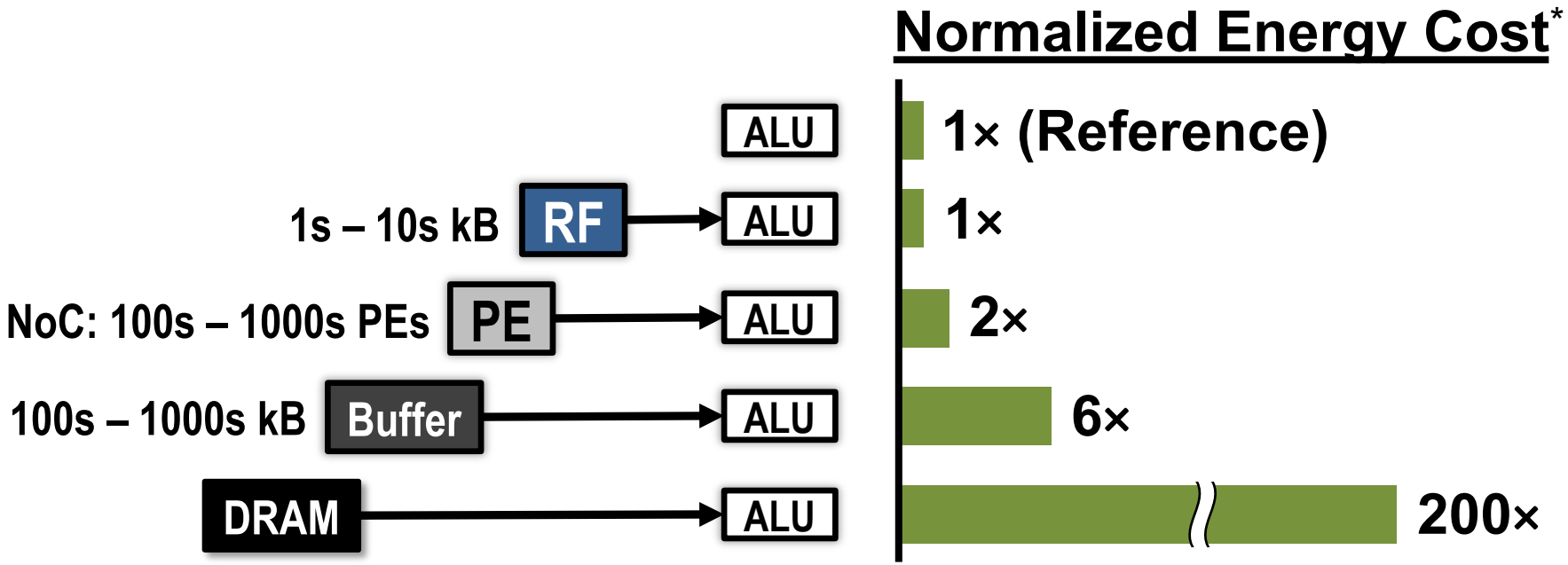




* measured from a commercial 65nm process

Multi-Level Low-Cost Data Access

A **Dataflow** is required to maximally exploit **data reuse** with the **low-cost memory hierarchy** and **parallelism**



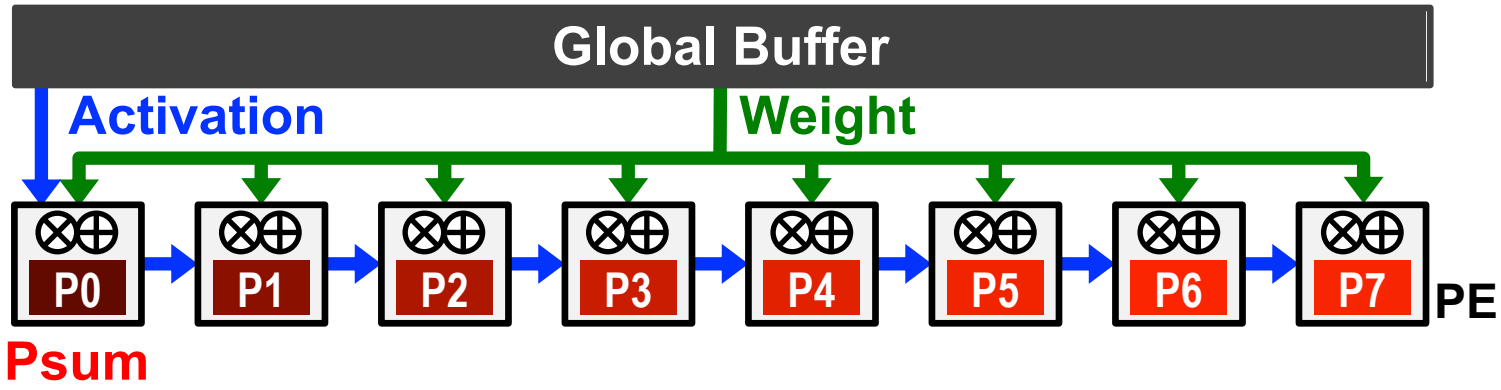
* measured from a commercial 65nm process

Dataflow Taxonomy

- Output Stationary (OS)
- Weight Stationary (WS)
- Input Stationary (IS)

[Chen et al., ISCA 2016]

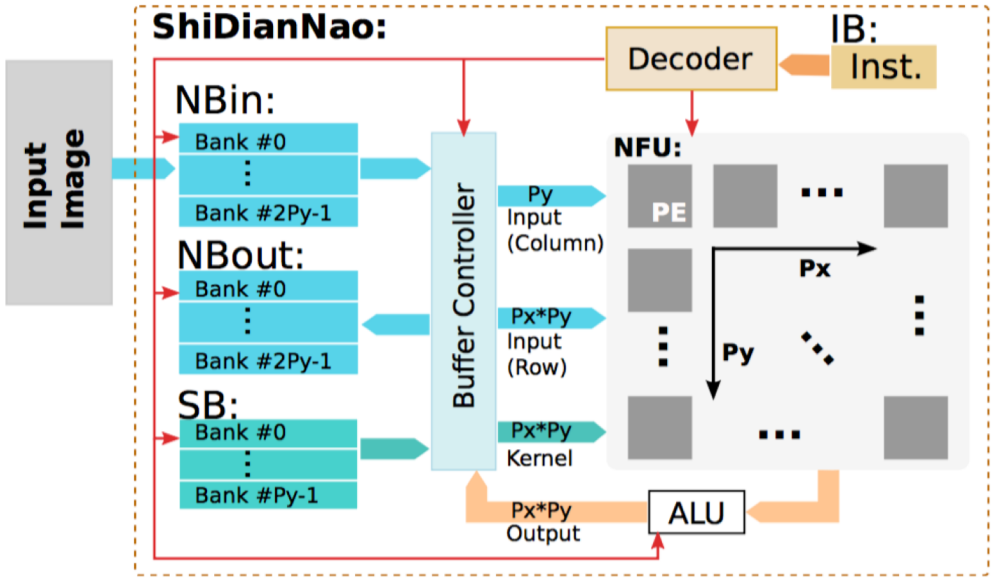
Output Stationary (OS)



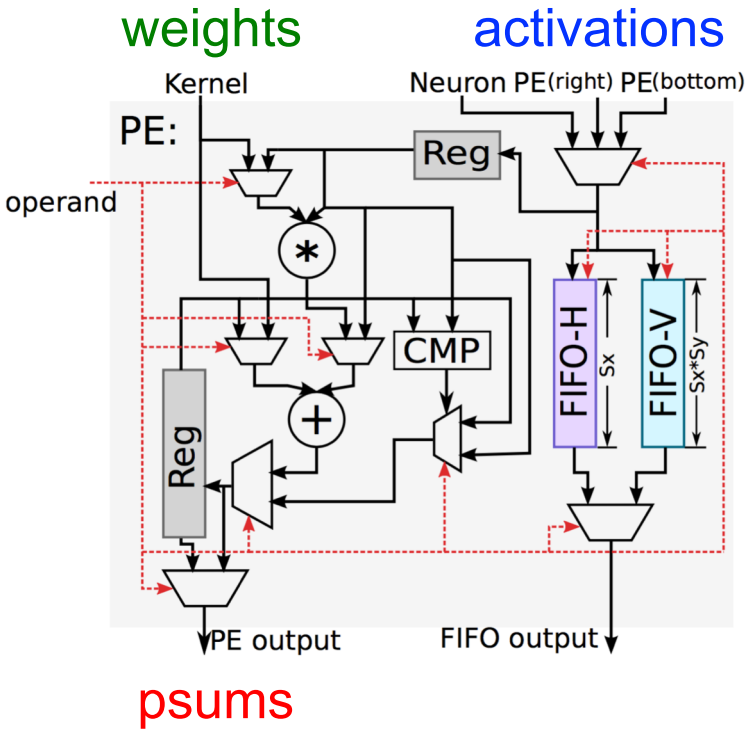
- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array

OS Example: ShiDianNao

Top-Level Architecture

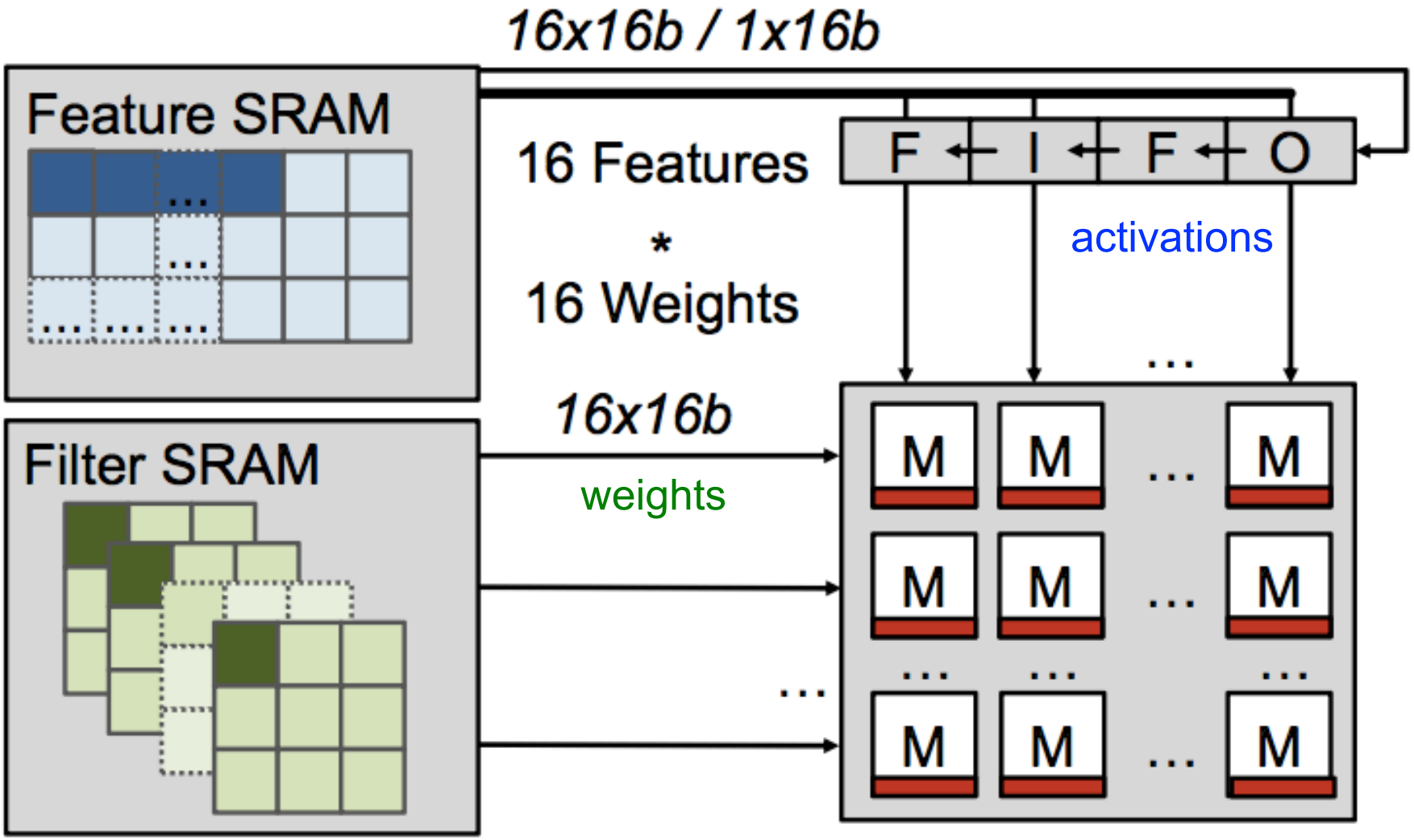


PE Architecture



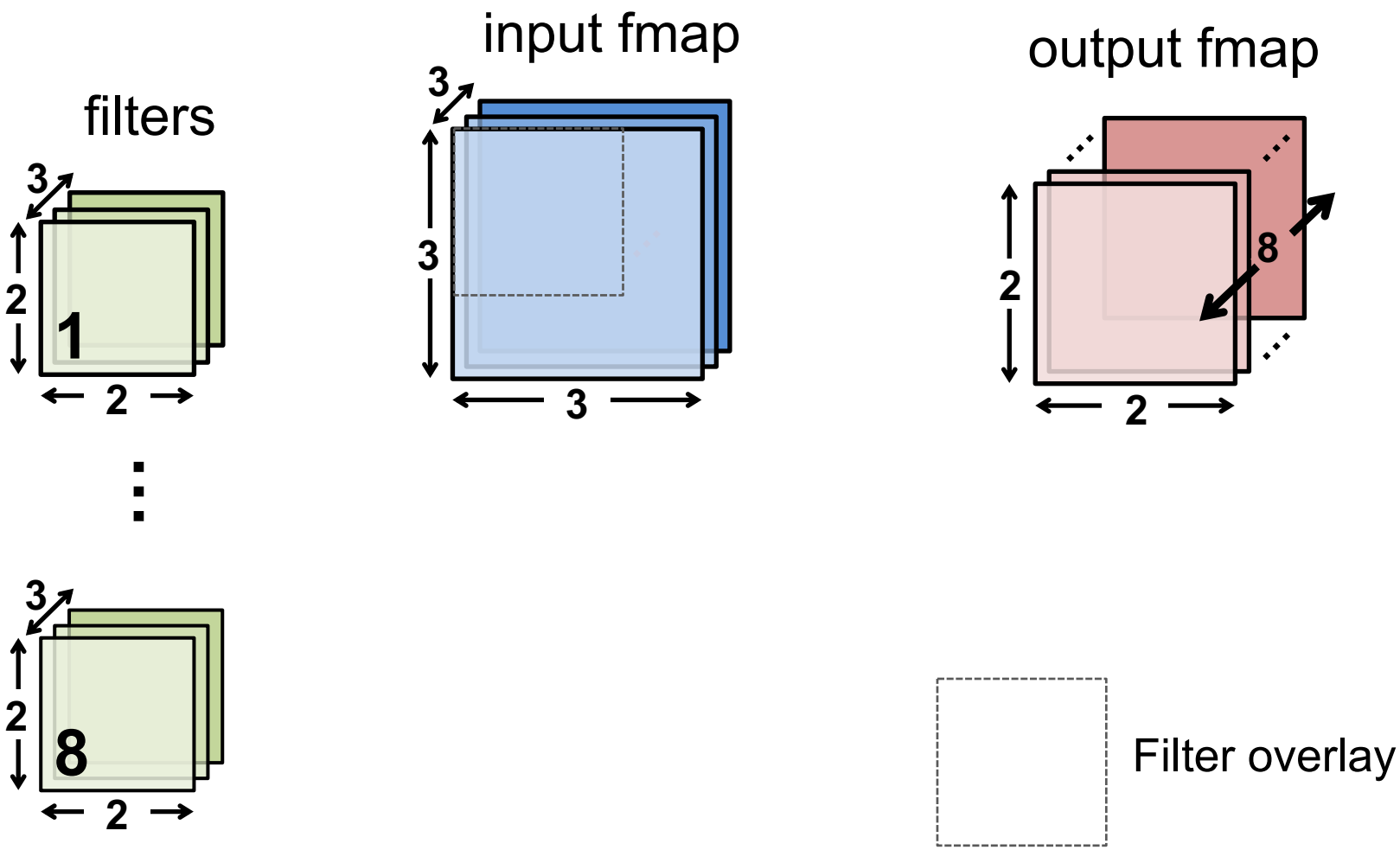
[Du et al., ISCA 2015]

OS Example: ENVISION



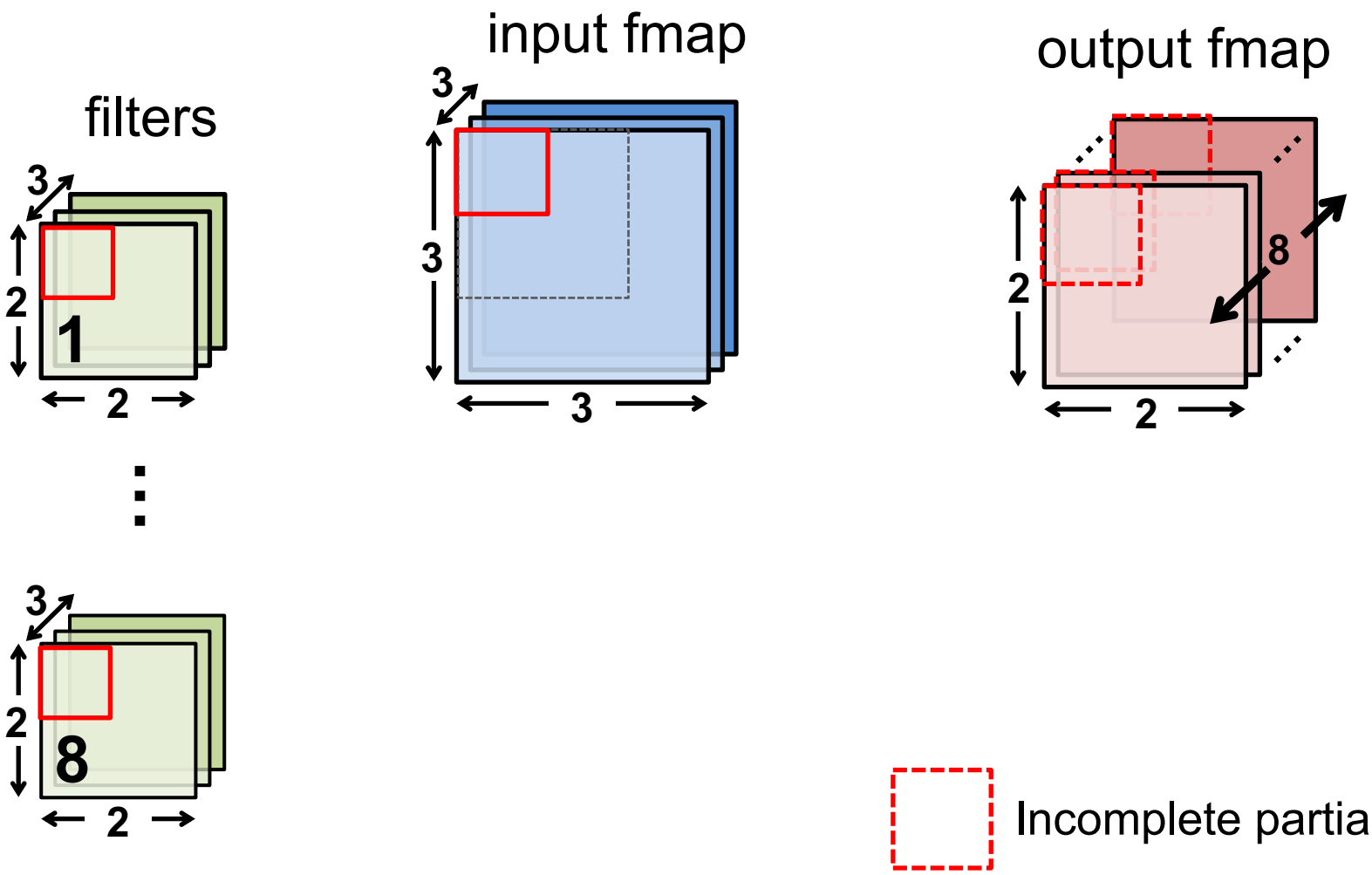
[Moons et al., VLSI 2016, ISSCC 2017]

OS Example


filters
3
2
1
2
8
input fmap
3
3
3
output fmap
2
8
2
Filter overlay

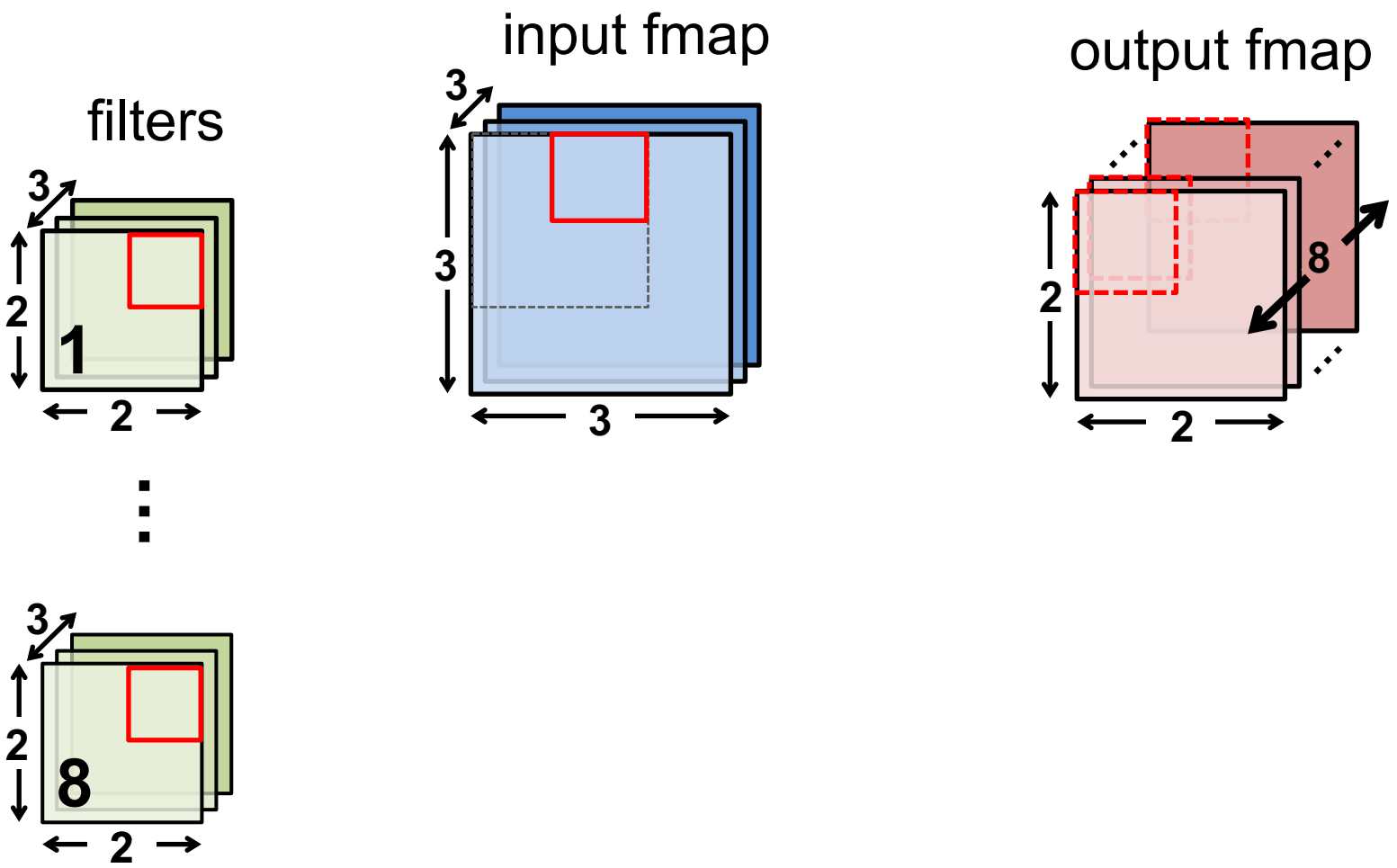
OS Example

Cycle through input fmap and weights (hold psum of output fmap)



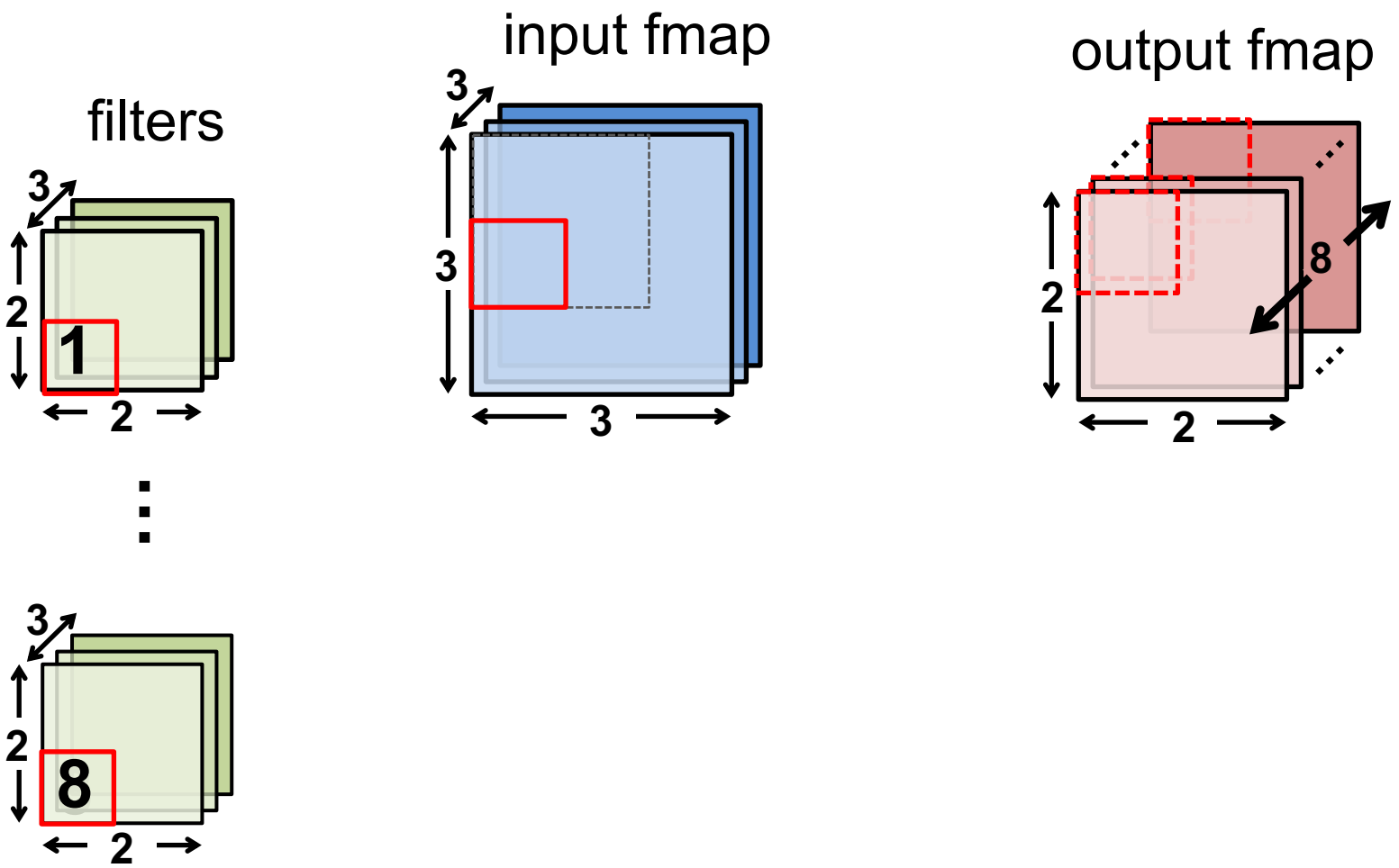
OS Example

Cycle through input fmap and weights (hold psum of output fmap)



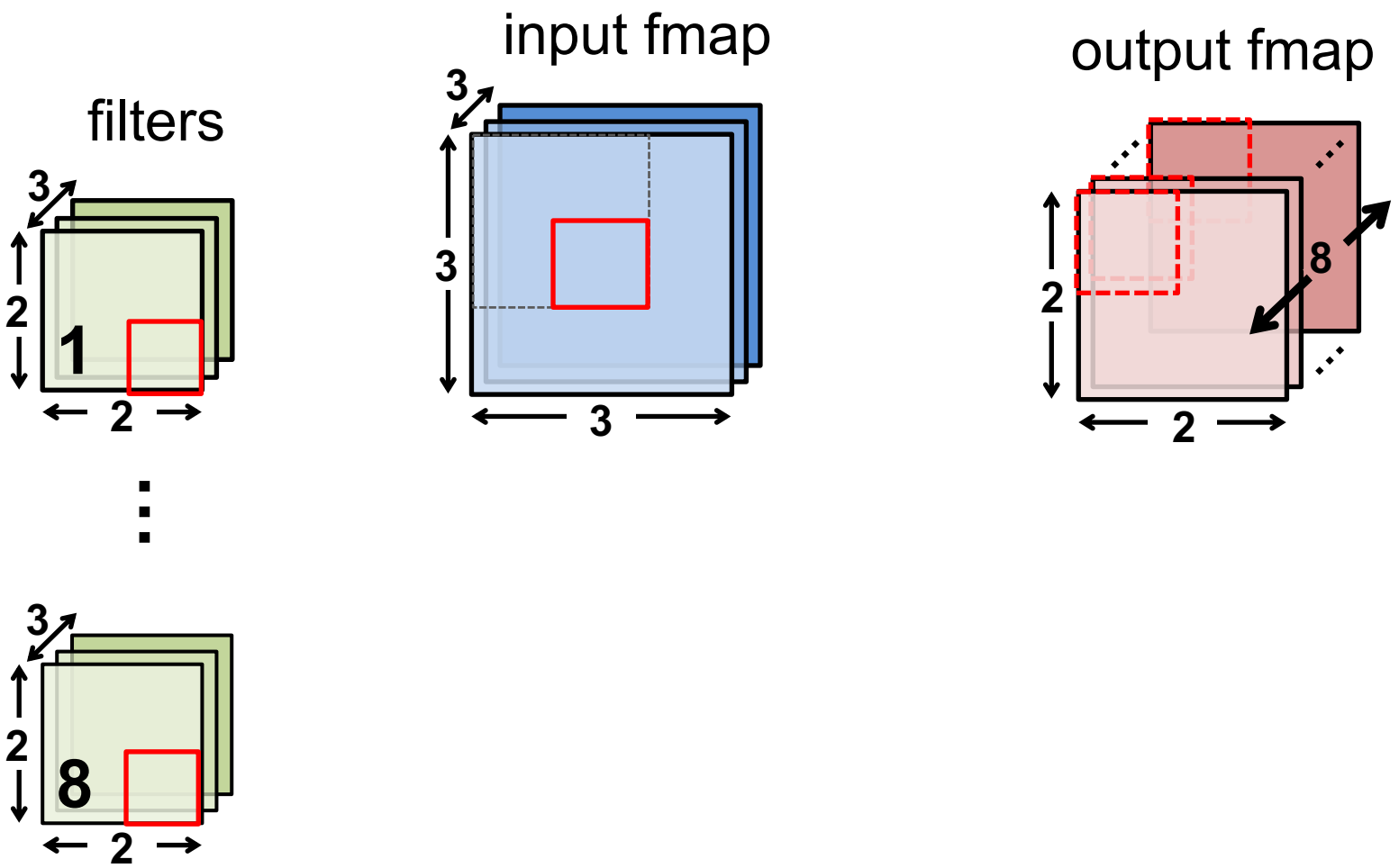
OS Example

Cycle through input fmap and weights (hold psum of output fmap)



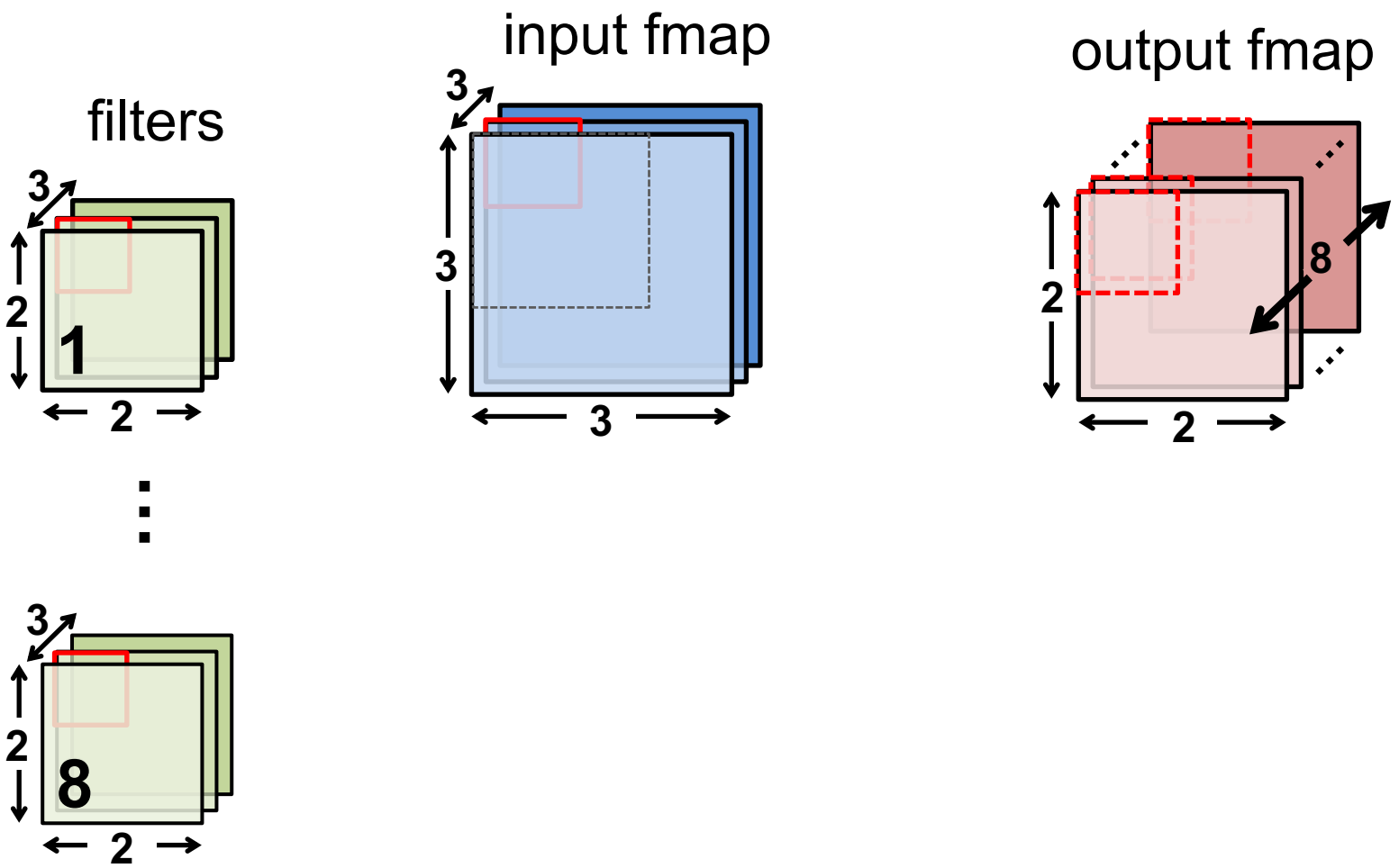
OS Example

Cycle through input fmap and weights (hold psum of output fmap)



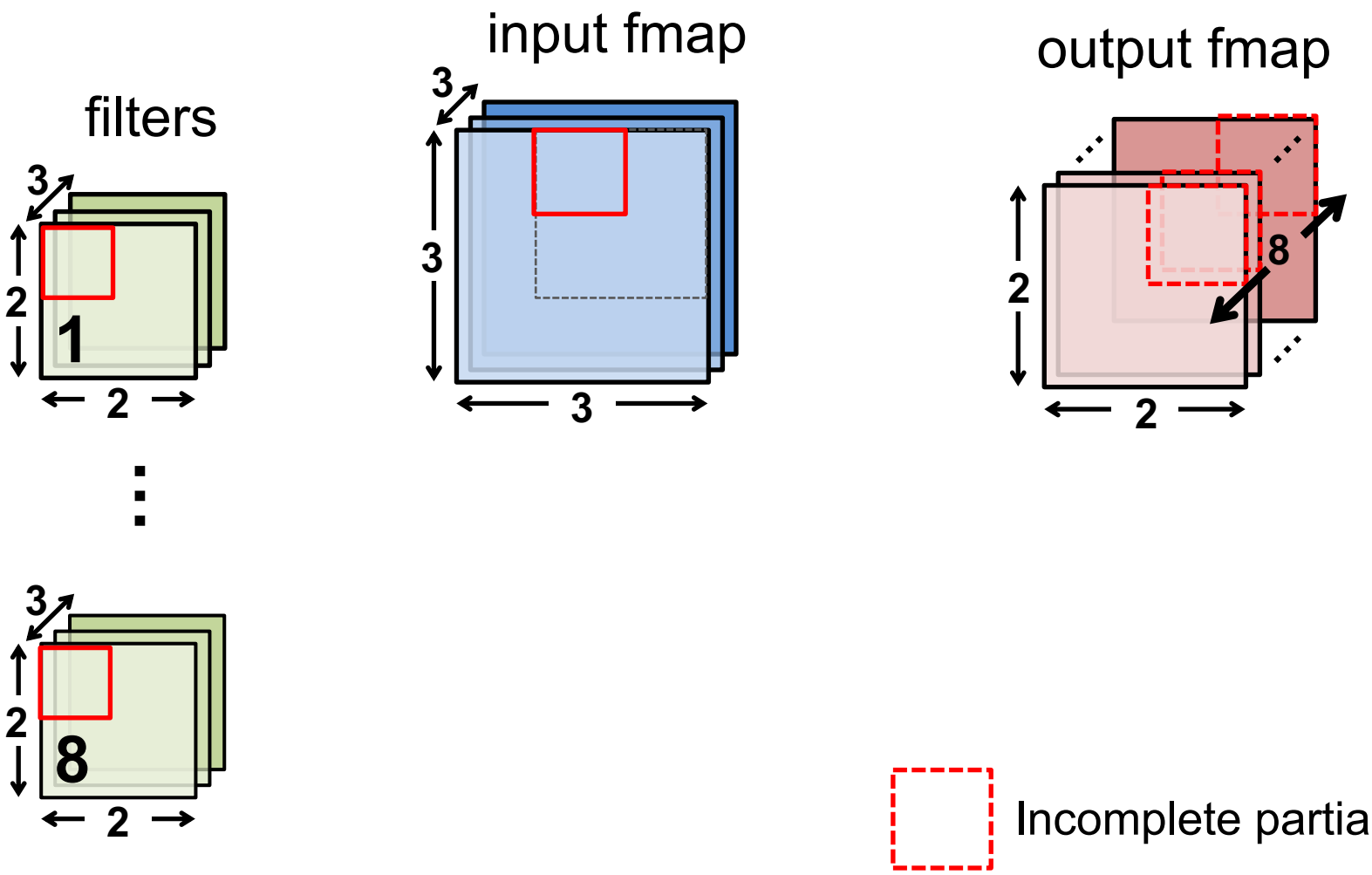
OS Example

Cycle through input fmap and weights (hold psum of output fmap)



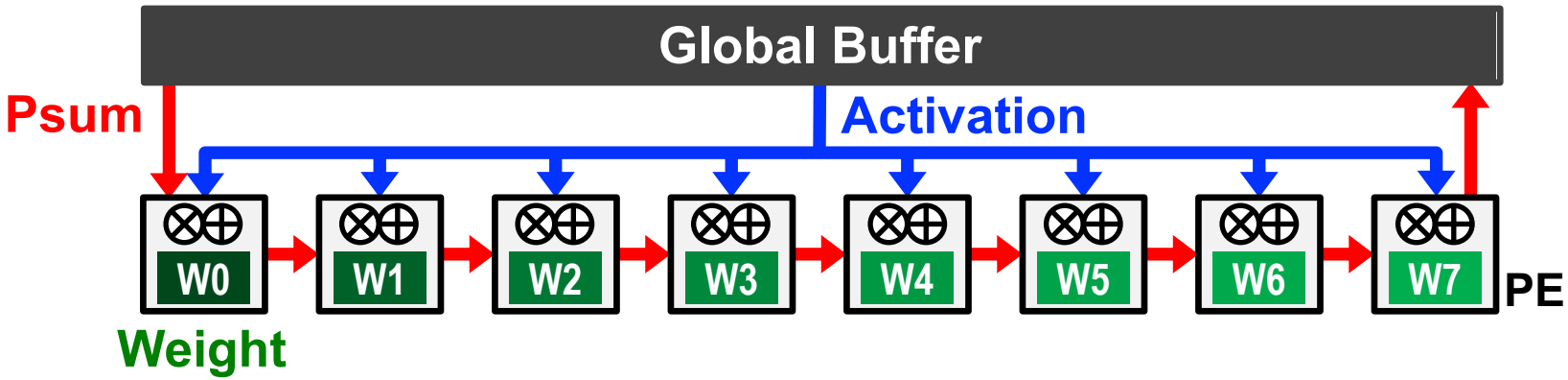
OS Example

Cycle through input fmap and weights (hold psum of output fmap)



Incomplete partial sum

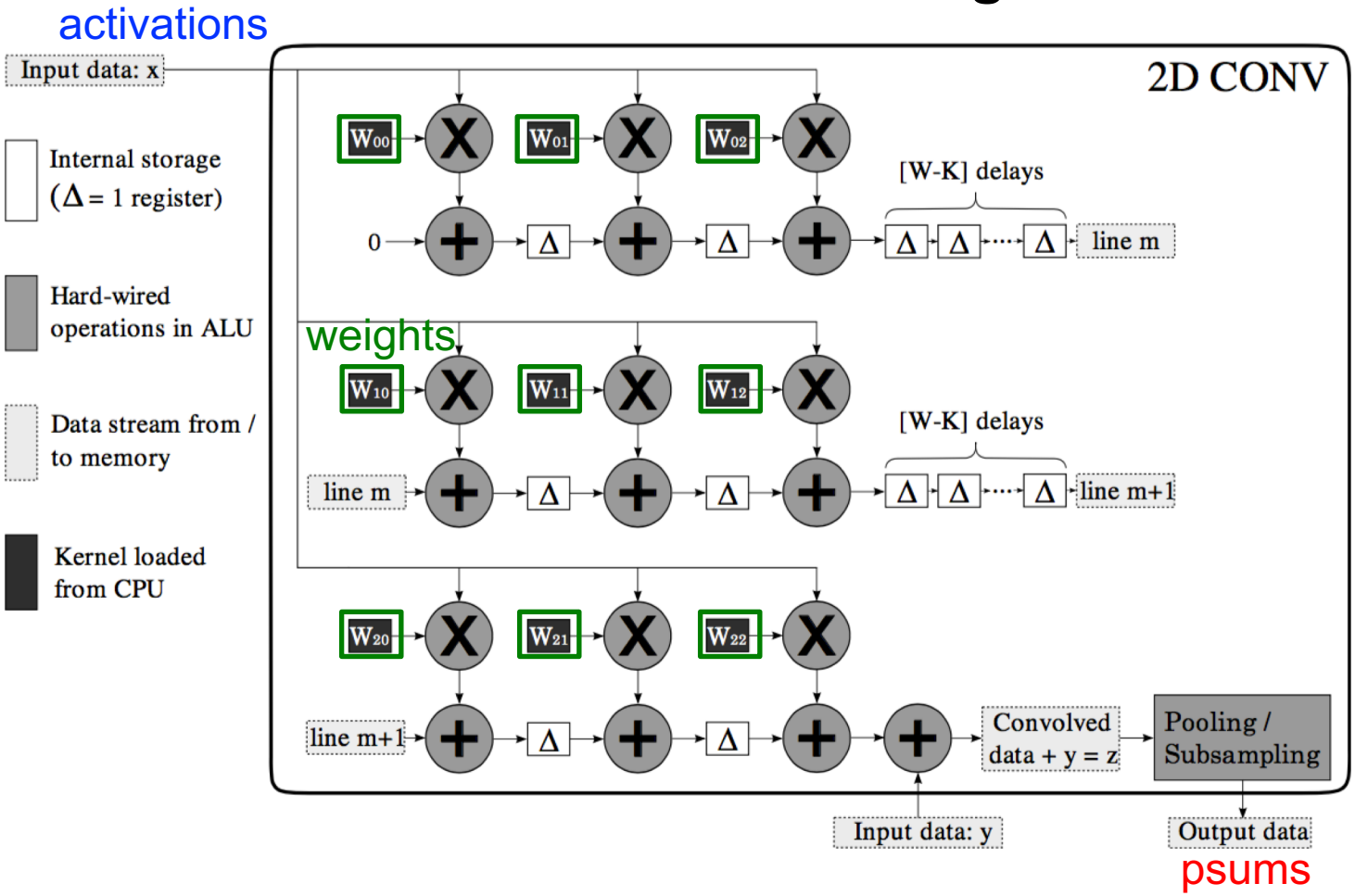
Weight Stationary (WS)



- **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

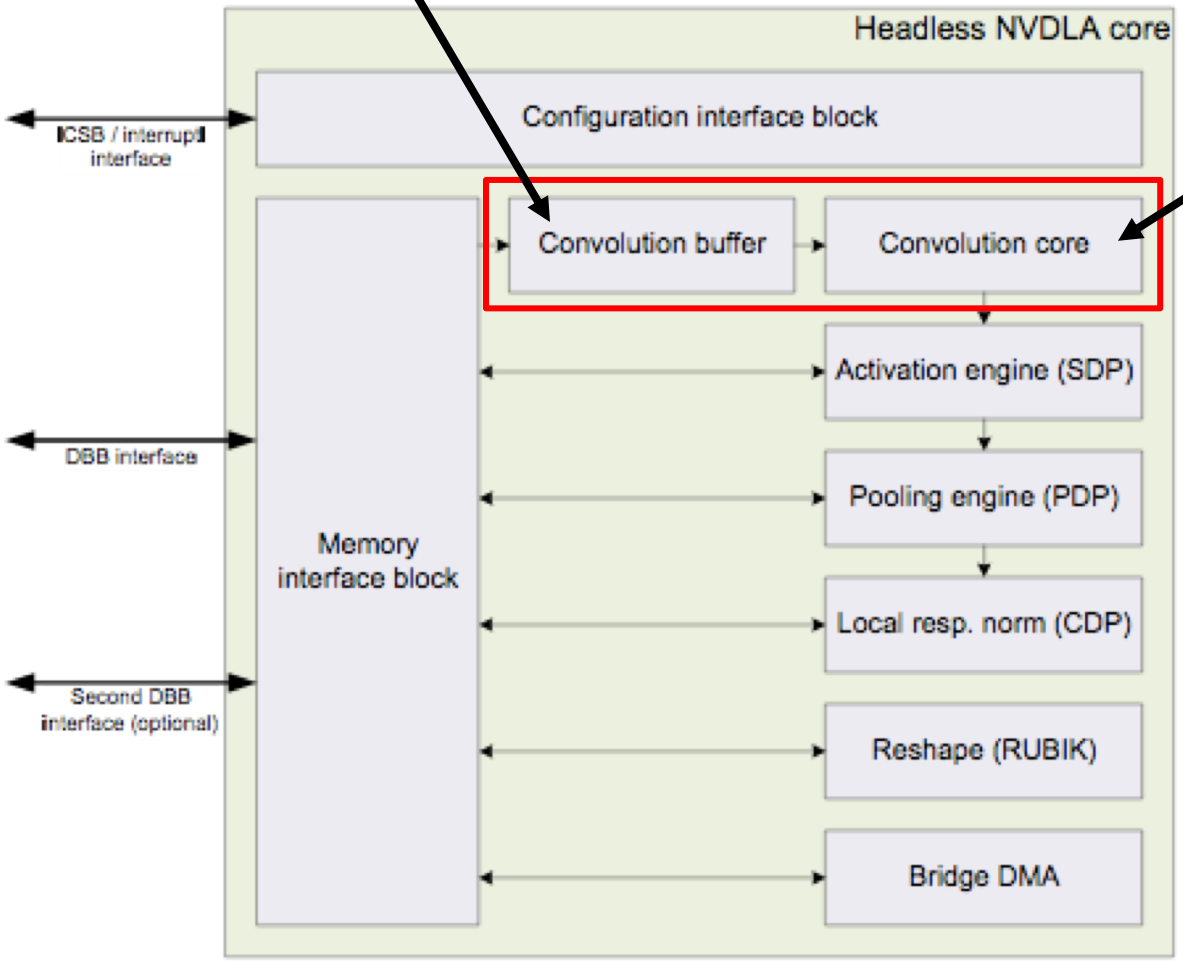


[Farabet et al., ICCV 2009]

WS Example: NVDLA (simplified)

Global Buffer

Released Sept 29, 2017

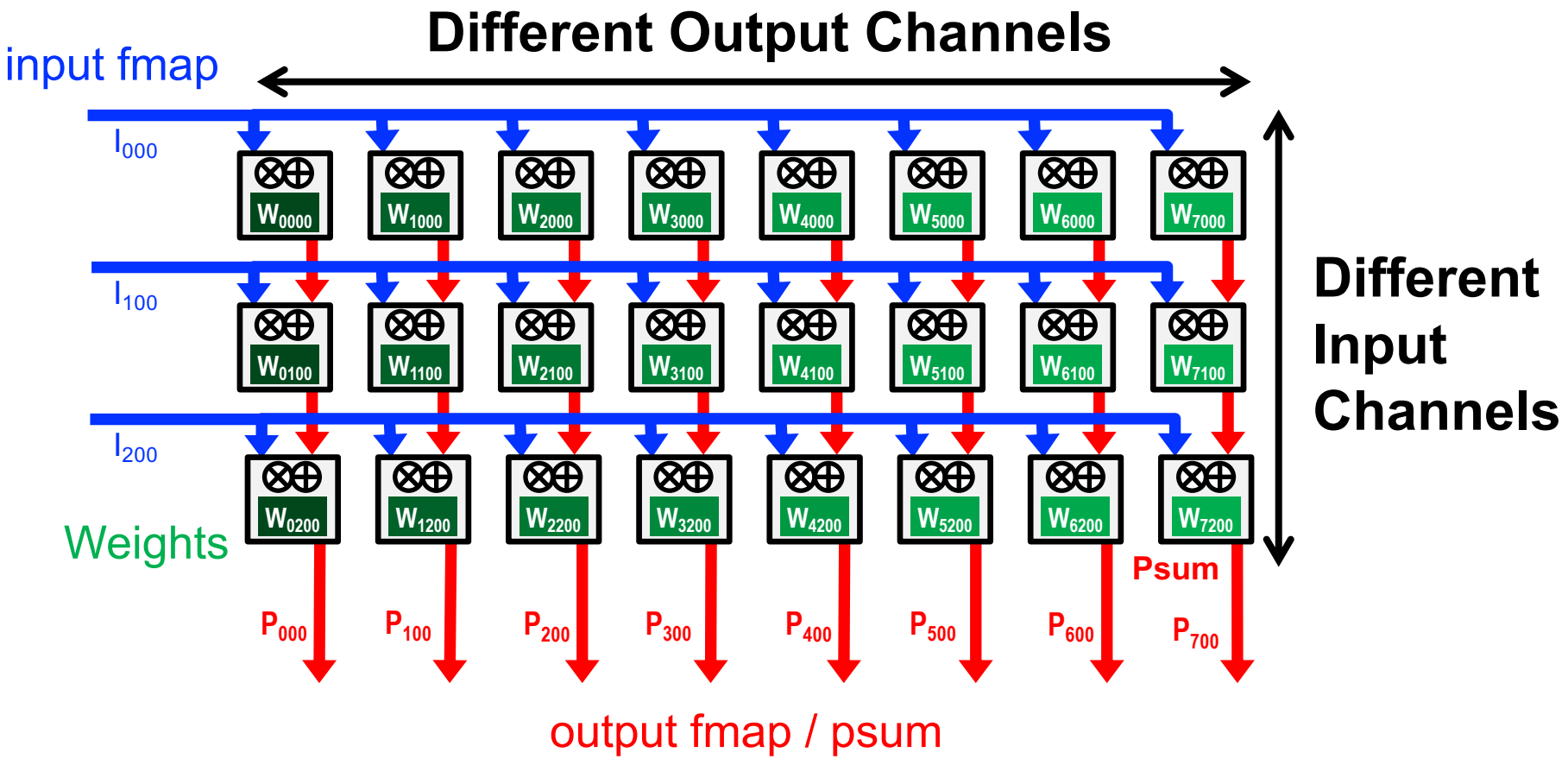


PE Array: *M x C MACs*

Image Source: Nvidia

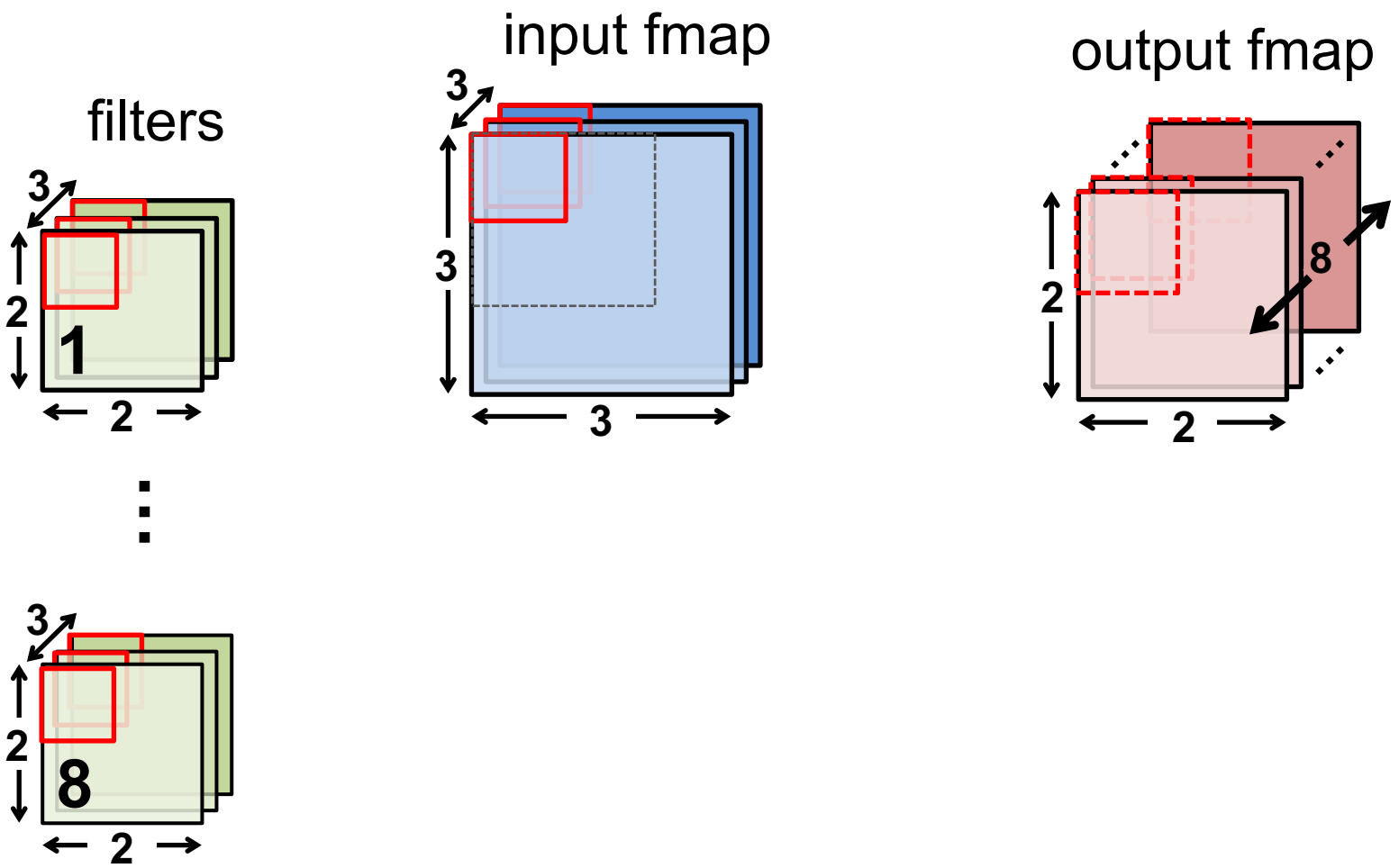
http://nvdla.org

WS Example: NVDLA (simplified)



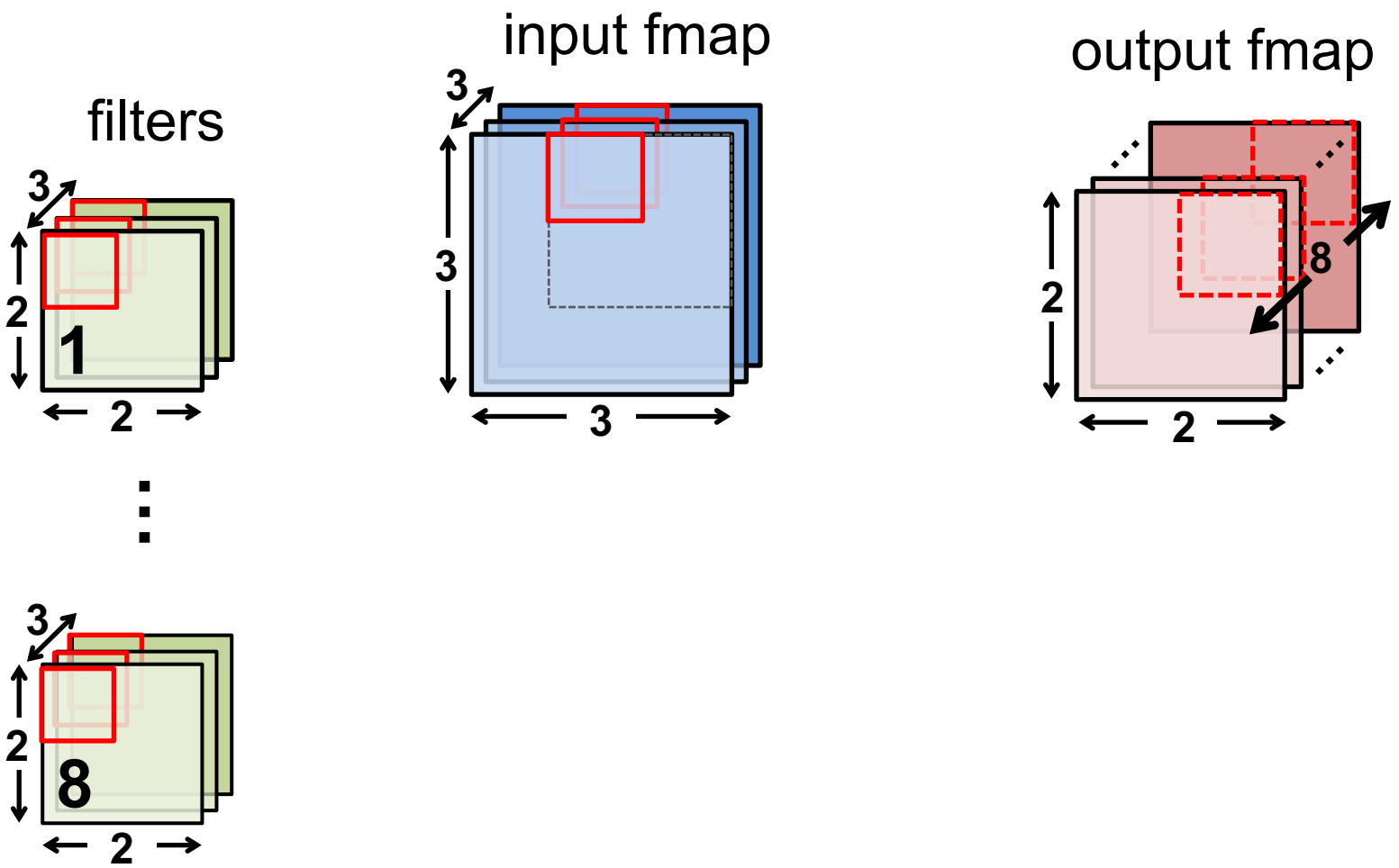
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



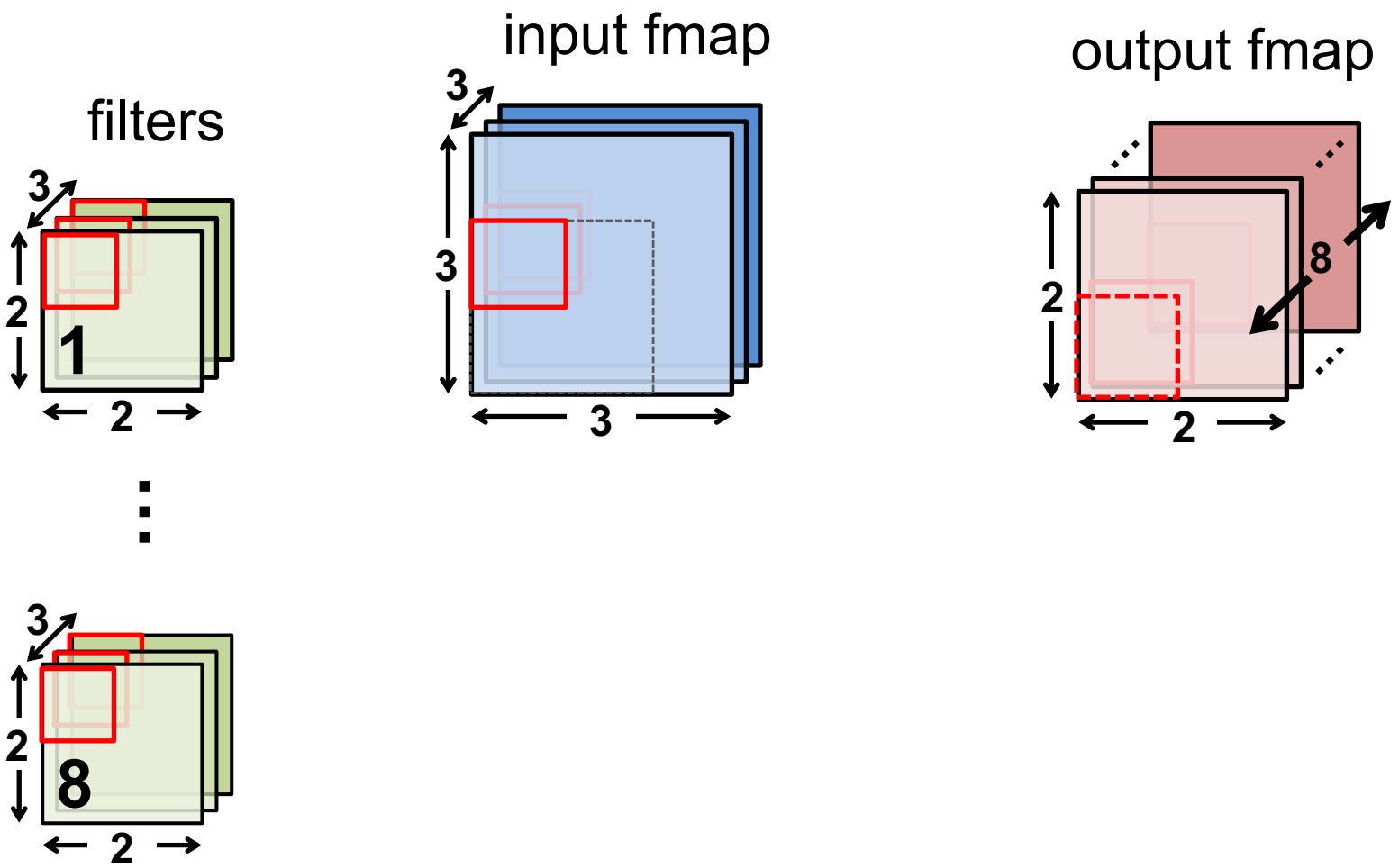
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



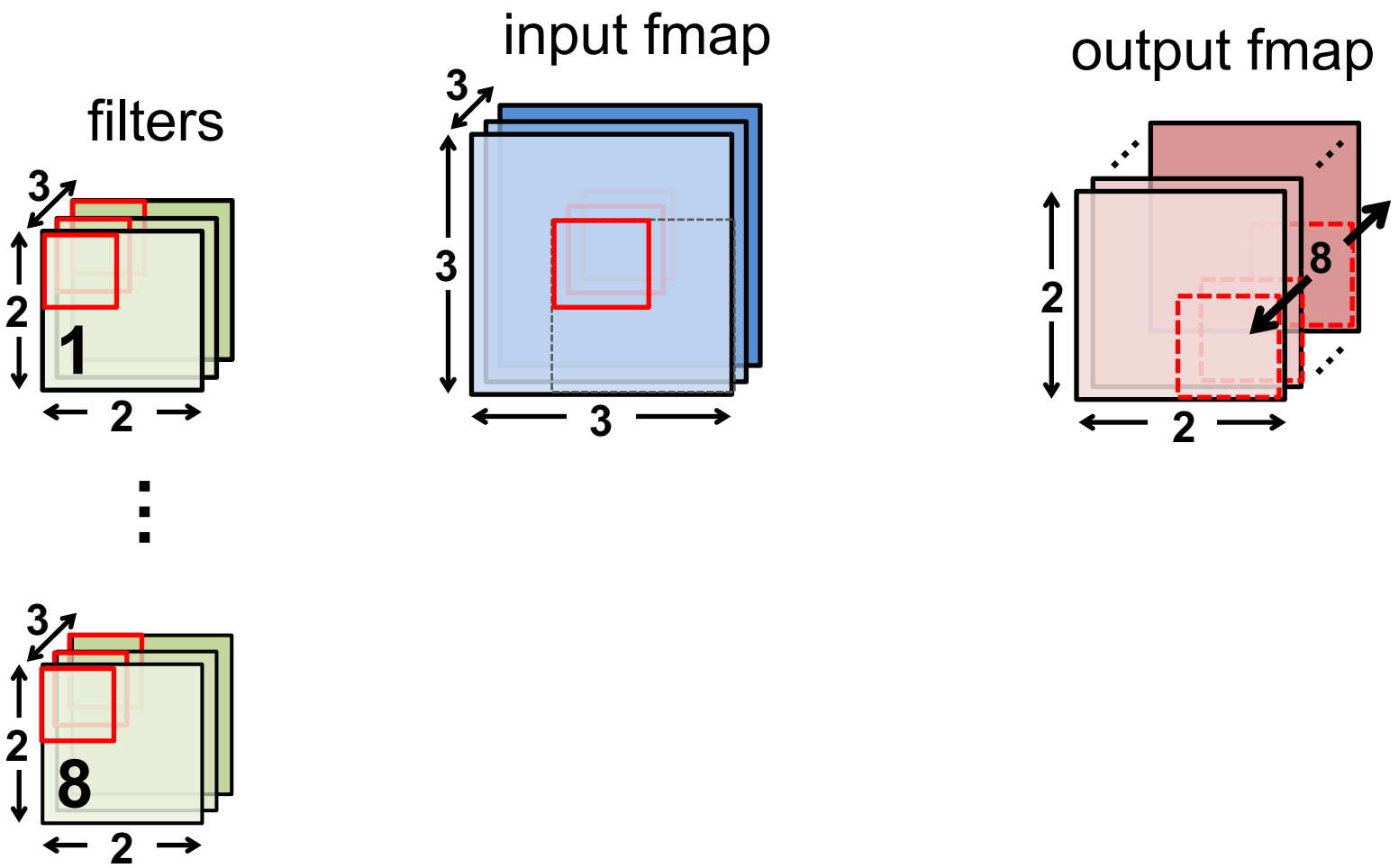
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



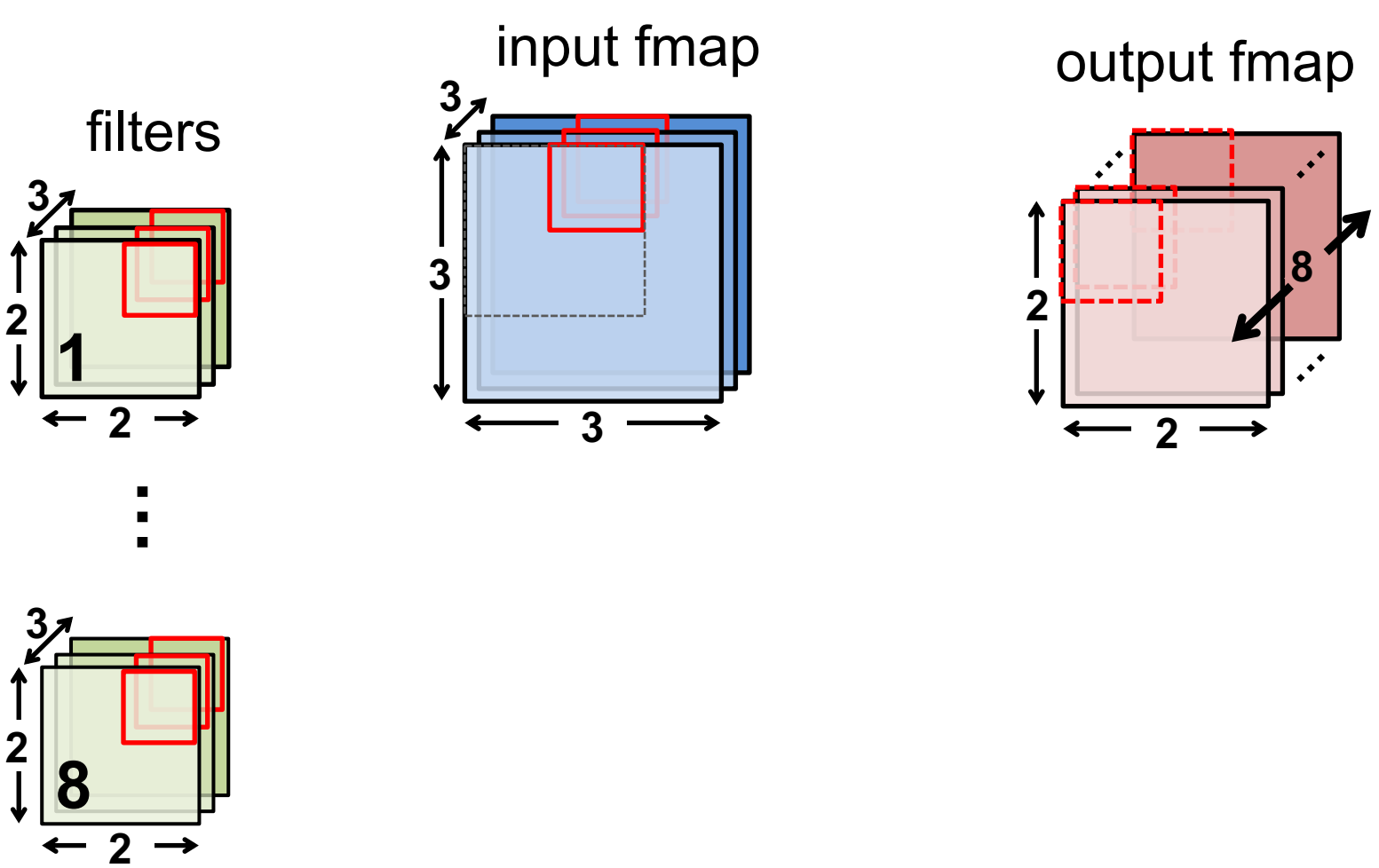
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



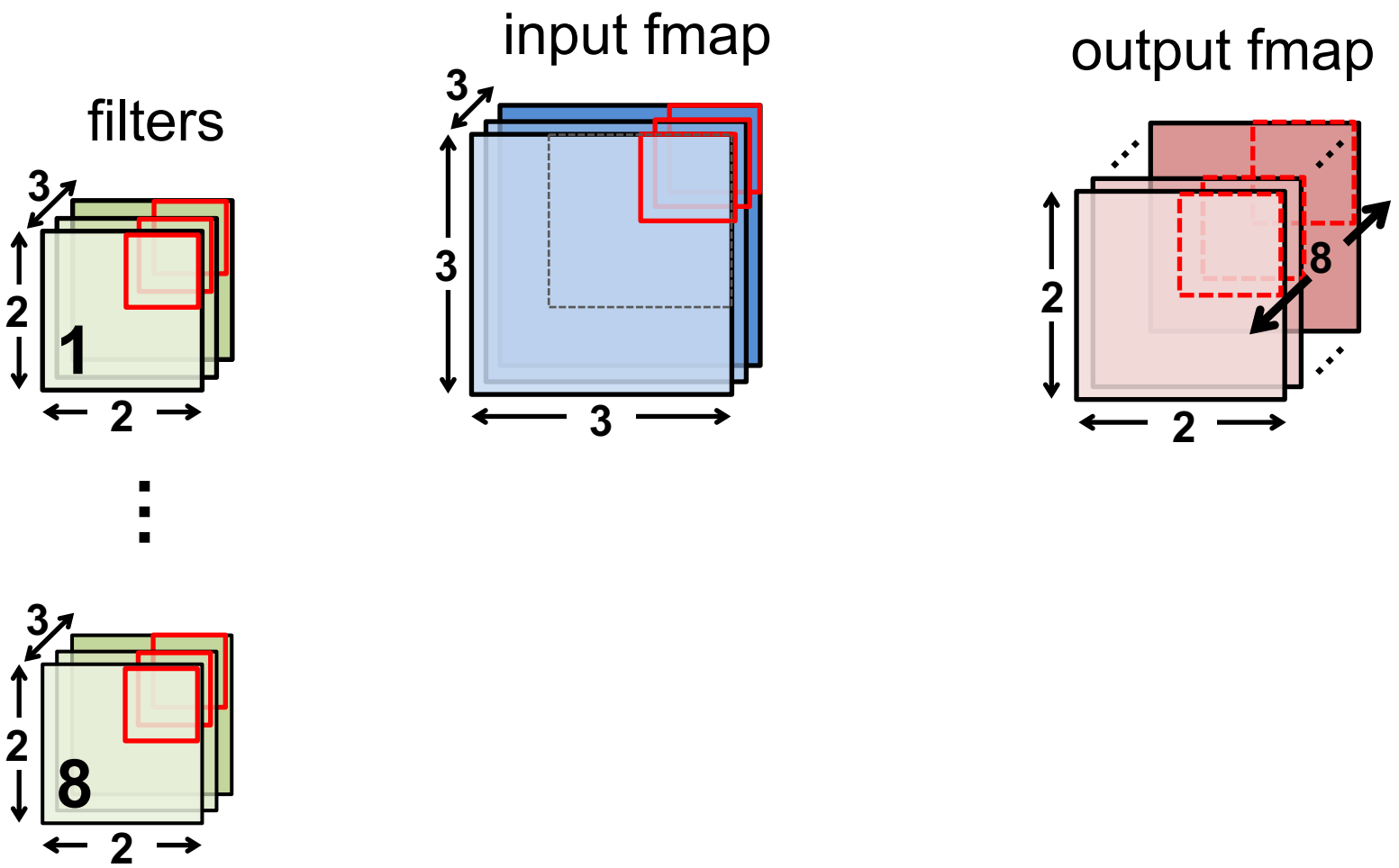
WS Example: NVDLA (simplified)

Load new weights

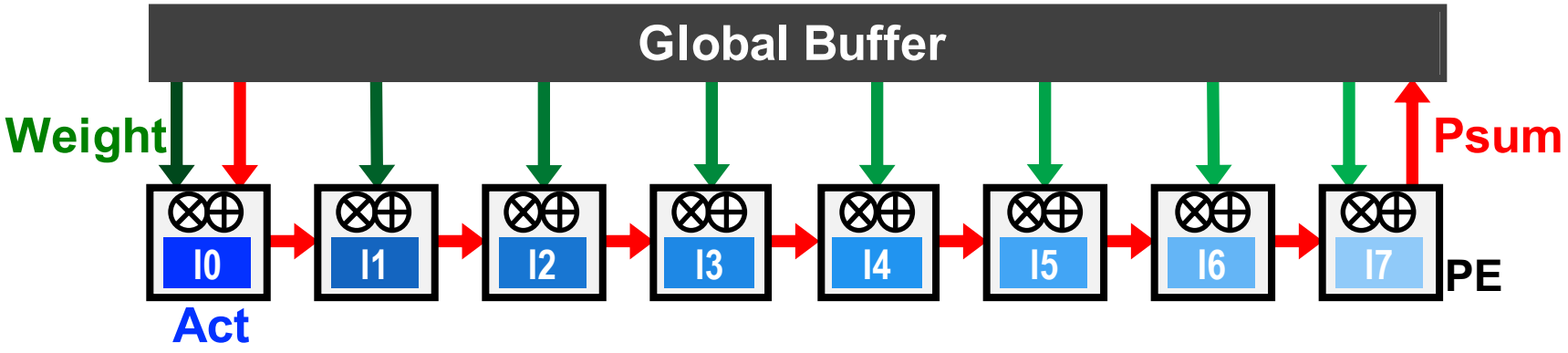


WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



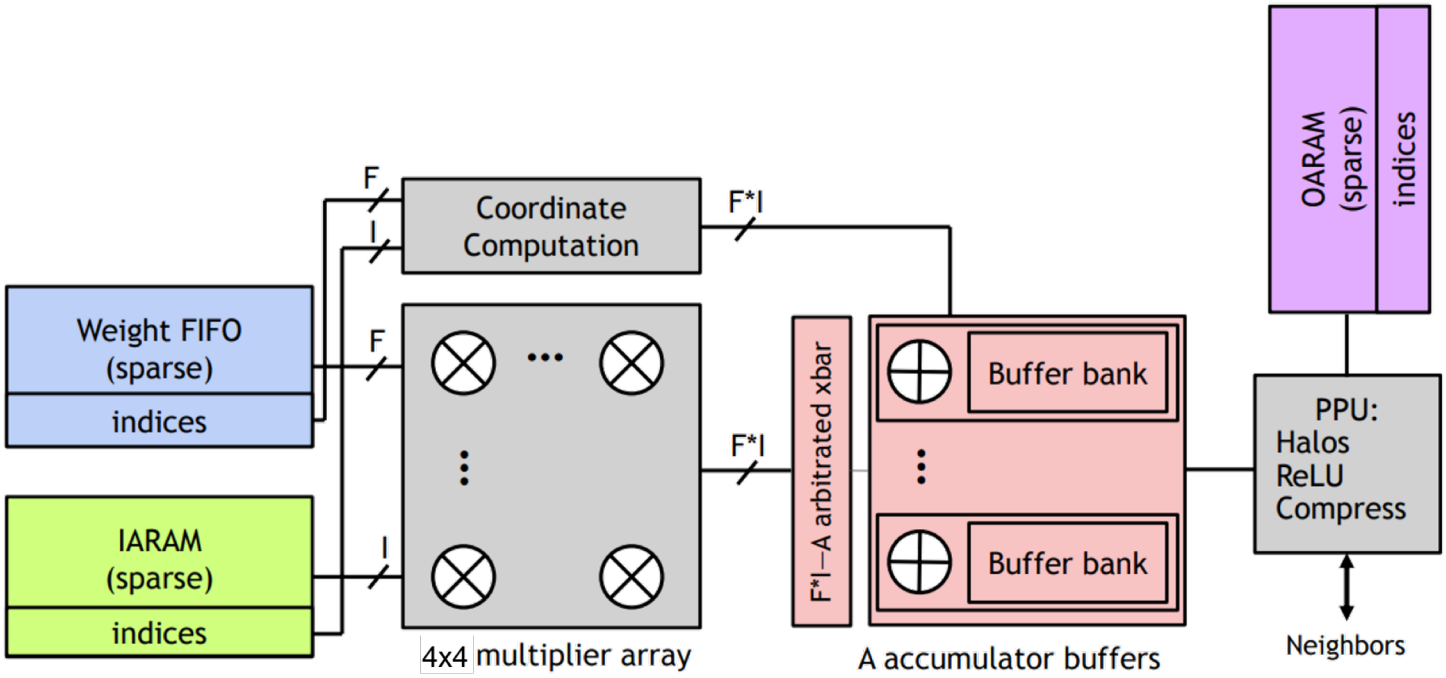
Input Stationary (IS)



- **Minimize activation** read energy consumption
  - maximize convolutional and fmap reuse of activations
- **Unicast weights** and **accumulate psums spatially** across the PE array.

IS Example: SCNN

- Used for sparse CNNs
  - Sparse CNN is where many weights are zeros
  - Activations also have sparsity from ReLU



[Parashar et al., ISCA 2017]

Summary of DNN Dataflows

- Minimizing **data movement** is the key to achieving high **energy efficiency** for DNN accelerators
- Dataflow taxonomy:
  - **Output Stationary**: minimize movement of psums
  - **Weight Stationary**: minimize movement of weights
  - **Input Stationary**: minimize movement of inputs
- **Loop nest** provides a compact way to describe various properties of a dataflow, e.g., data tiling in multi-level storage and spatial processing.

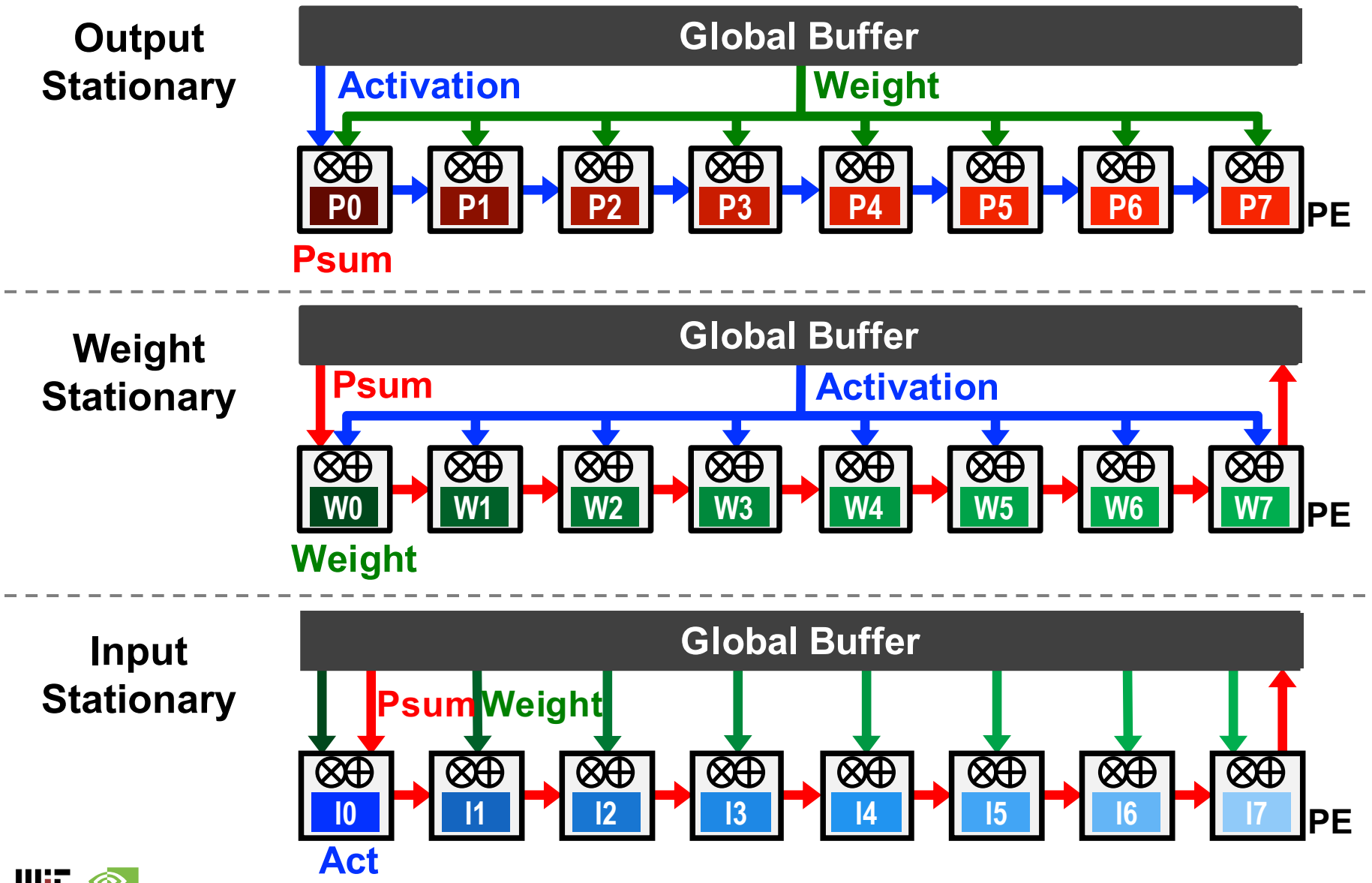
DNN Accelerator Architectures (cont.)

ISCA Tutorial (2019)

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

Joel Emer, Vivienne Sze, Yu-Hsin Chen

Recap on Dataflow Taxonomy

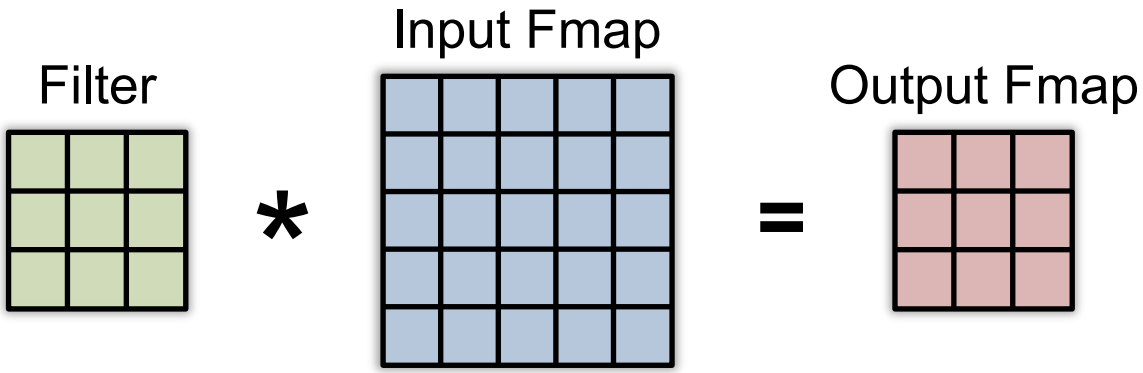


Energy-Efficient Dataflow: Row Stationary (RS)

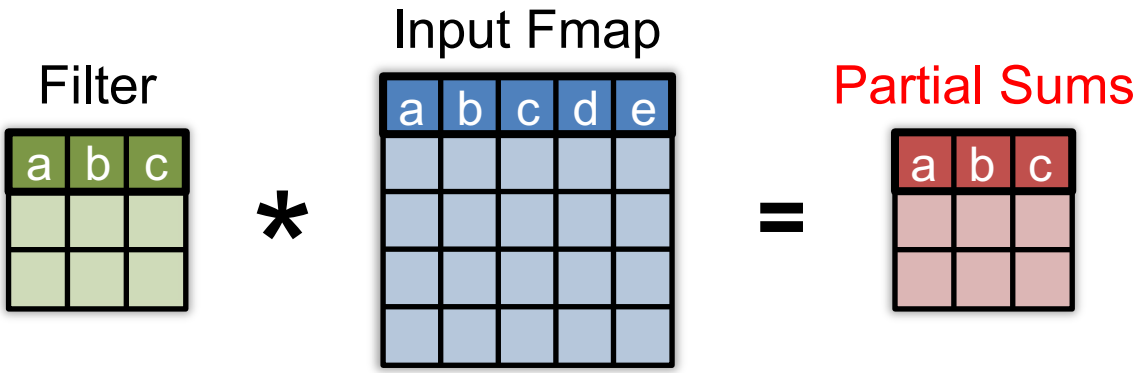
- **Maximize** data reuse at **RF**
- Optimize for **overall** energy efficiency instead for *only* a certain data type

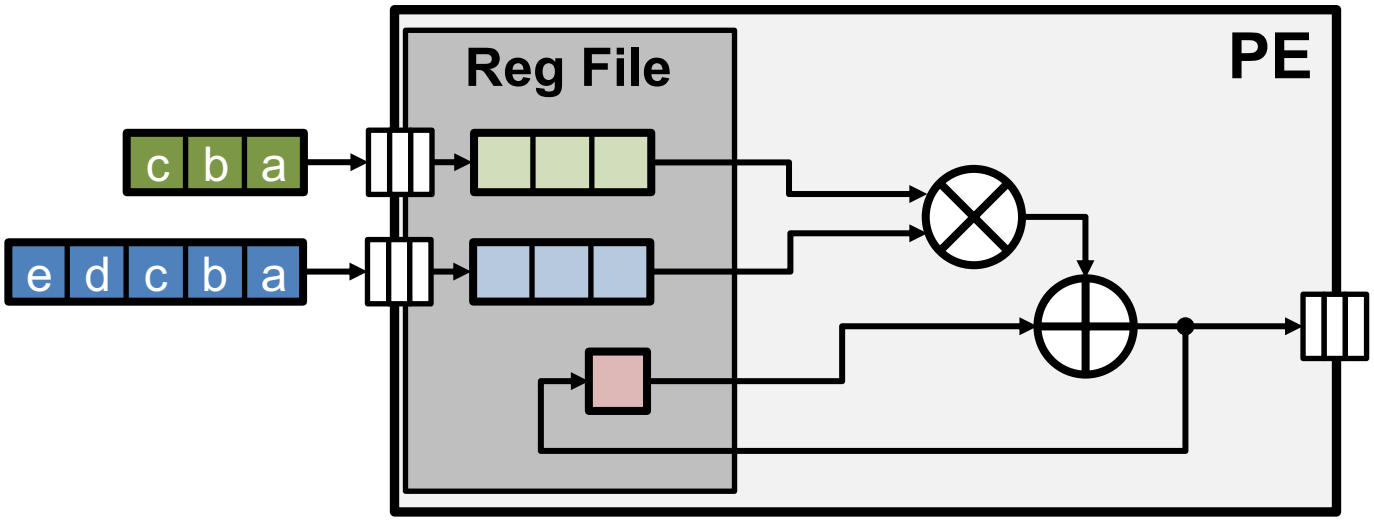
[Chen et al., ISCA 2016]

Row Stationary: Energy-efficient Dataflow

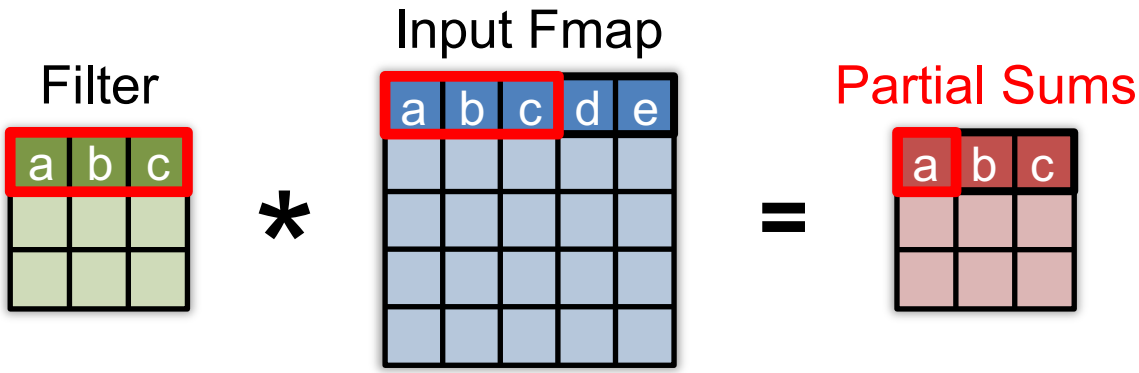


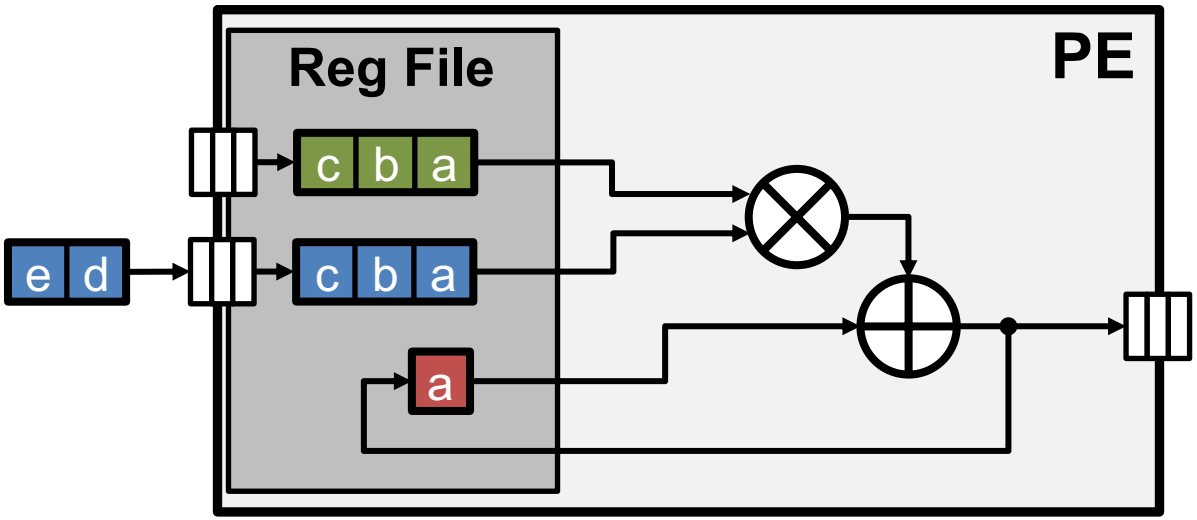
1D Row Convolution in PE



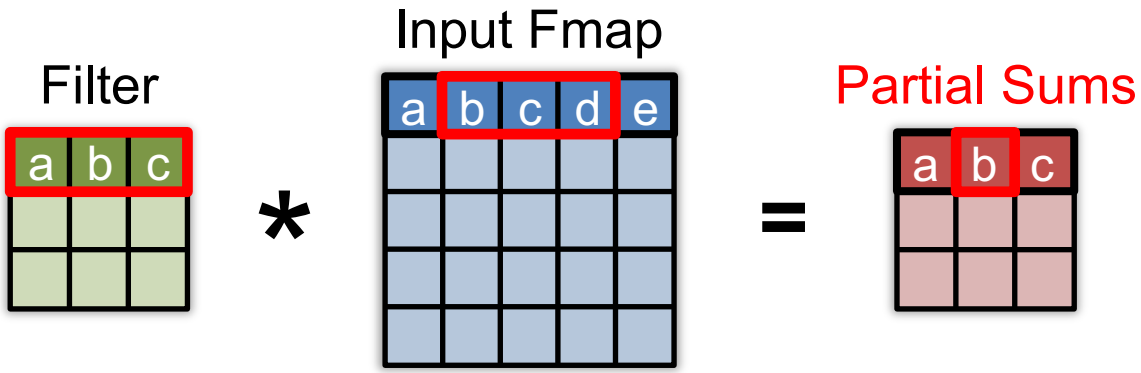


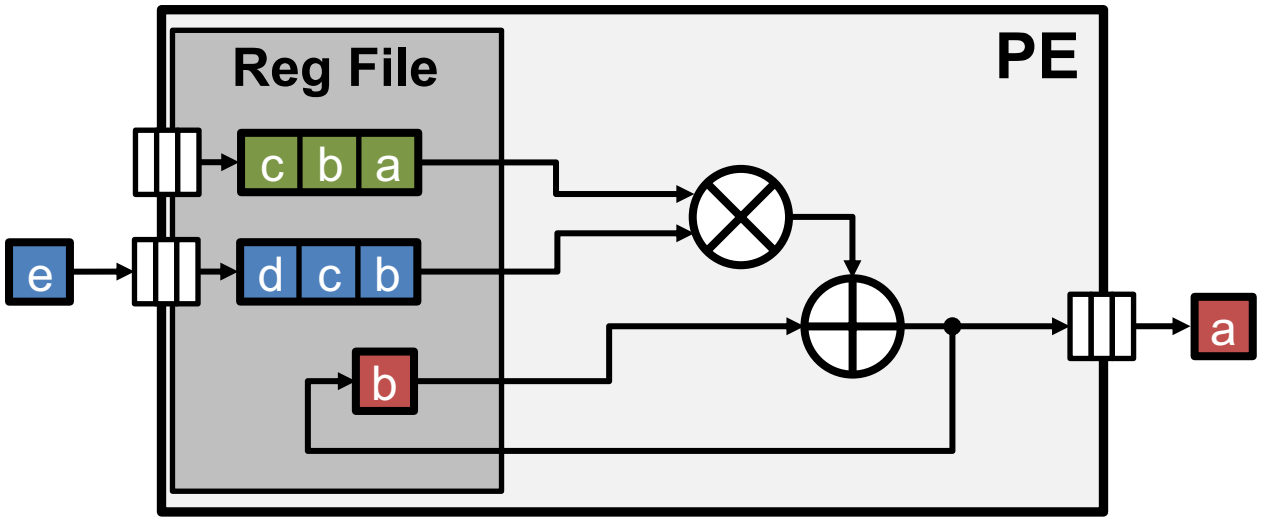
1D Row Convolution in PE



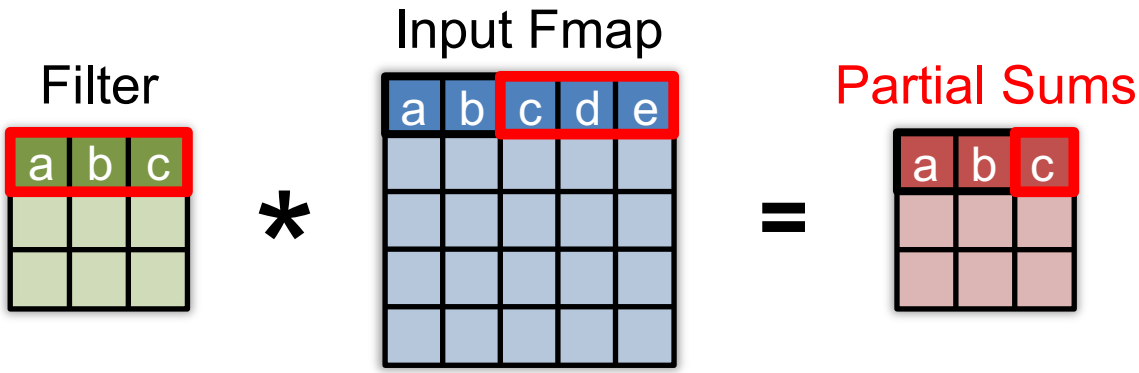


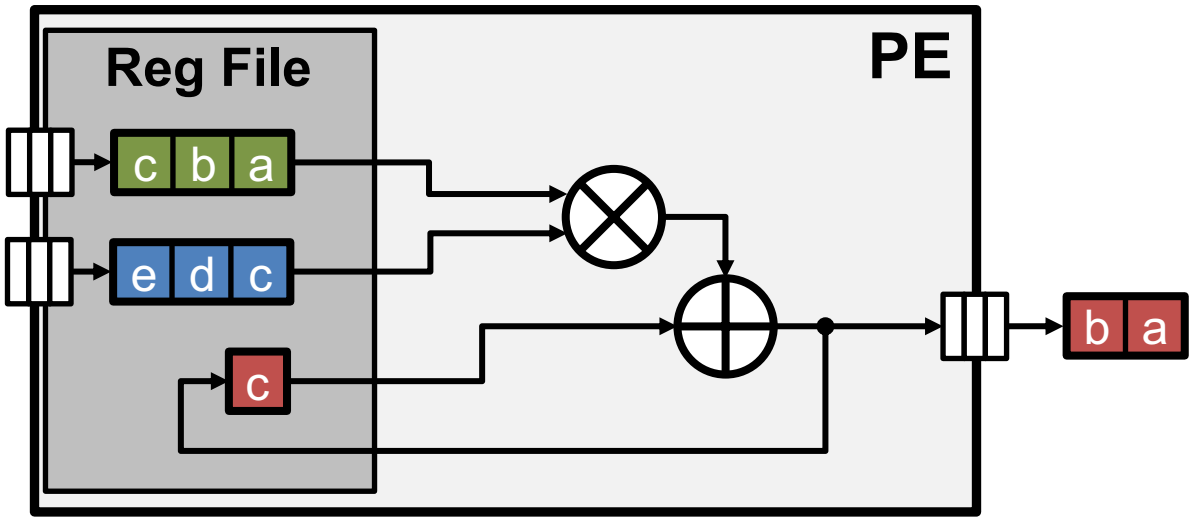
1D Row Convolution in PE





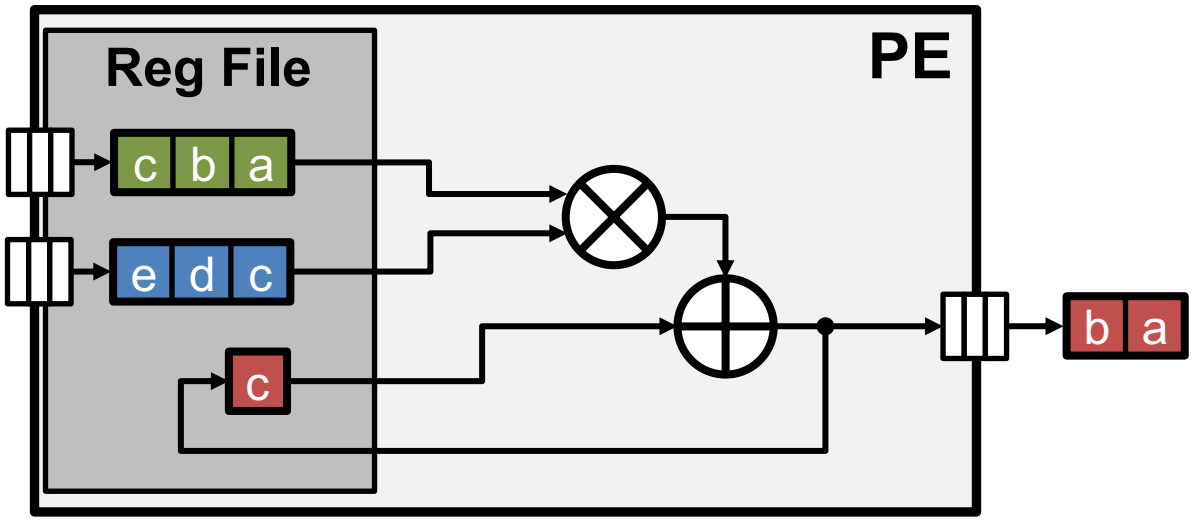
1D Row Convolution in PE



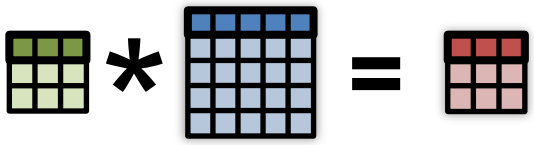


1D Row Convolution in PE

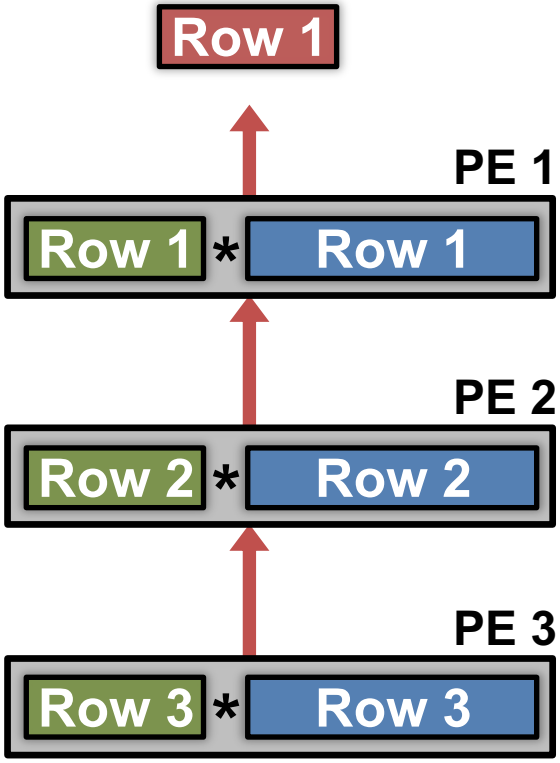
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF

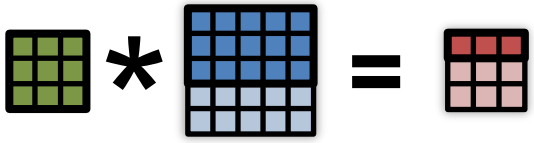


2D Convolution in PE Array

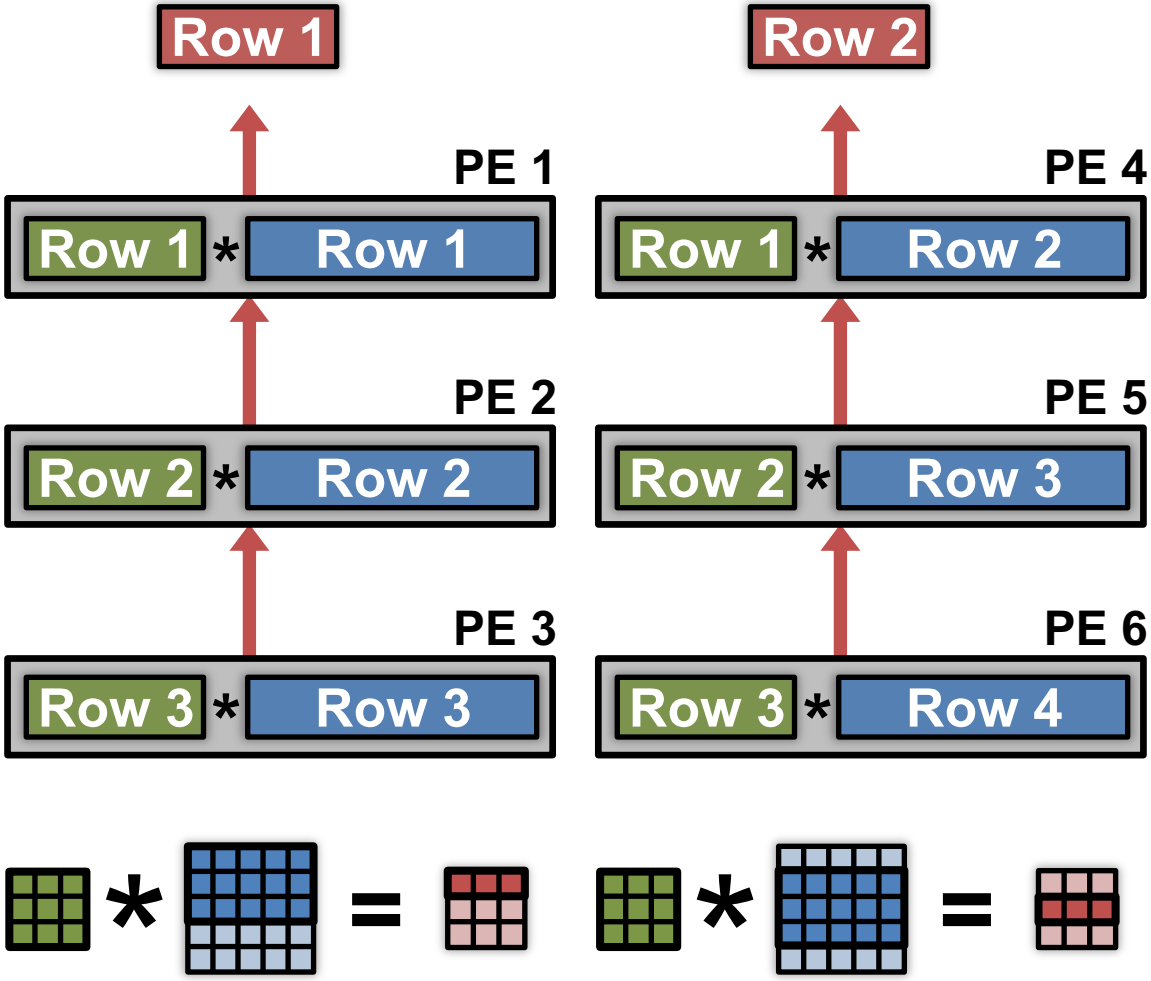


2D Convolution in PE Array

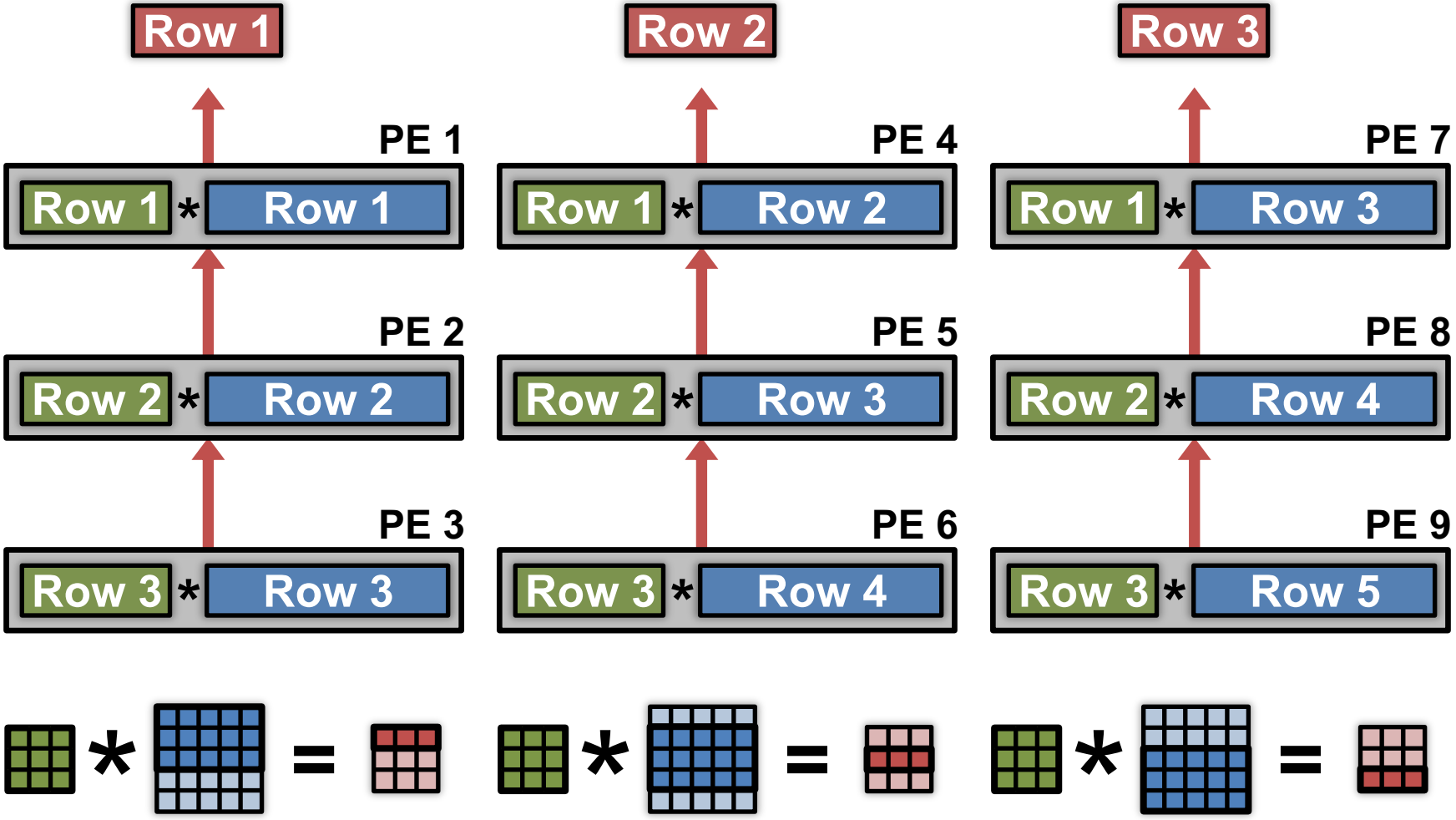




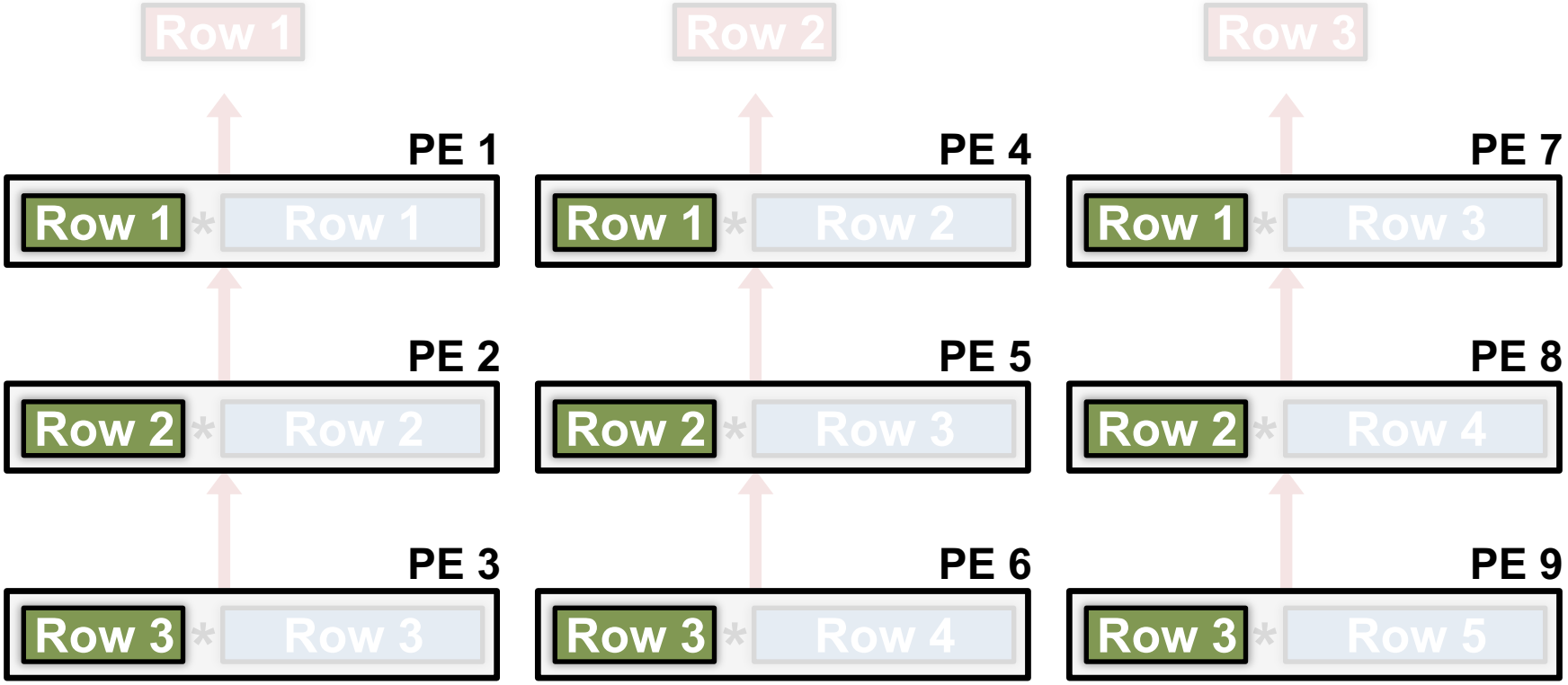
2D Convolution in PE Array



2D Convolution in PE Array

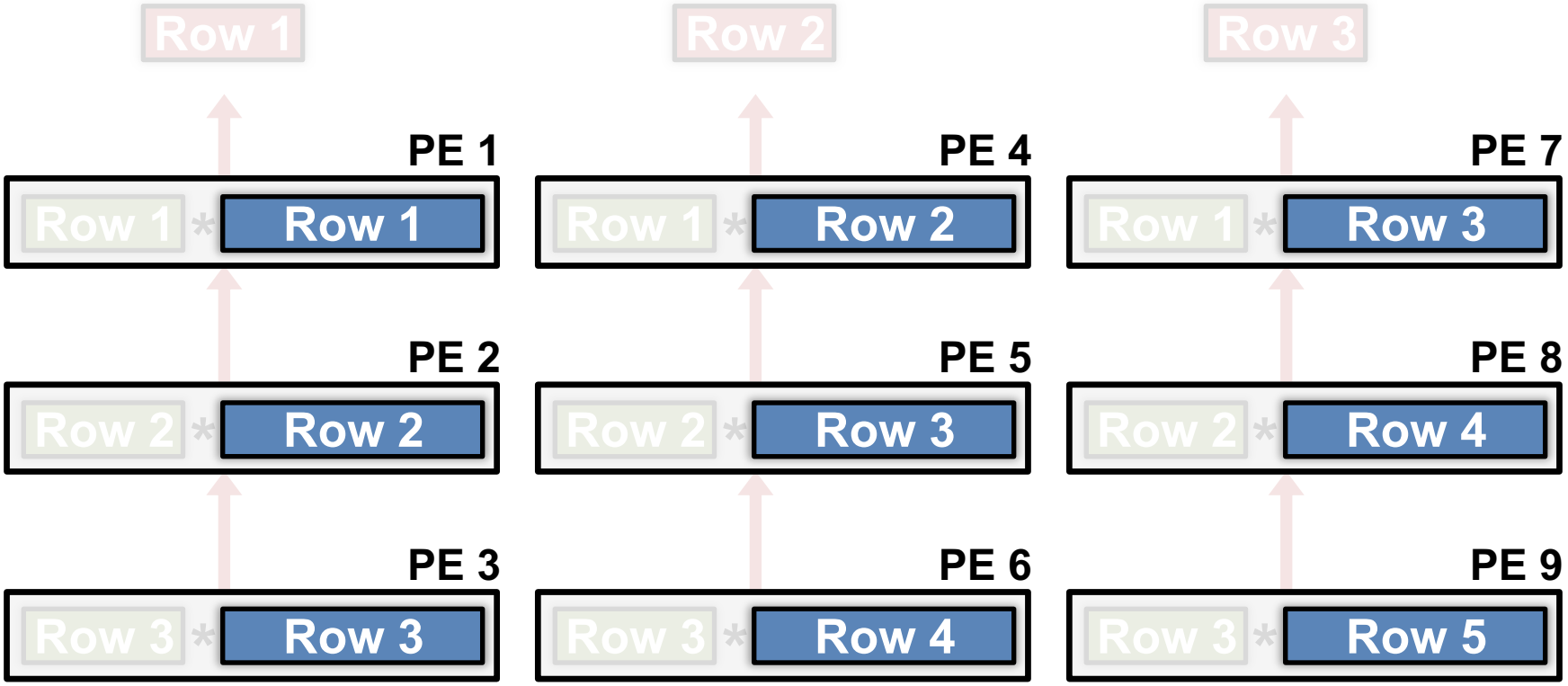


Convolutional Reuse Maximized



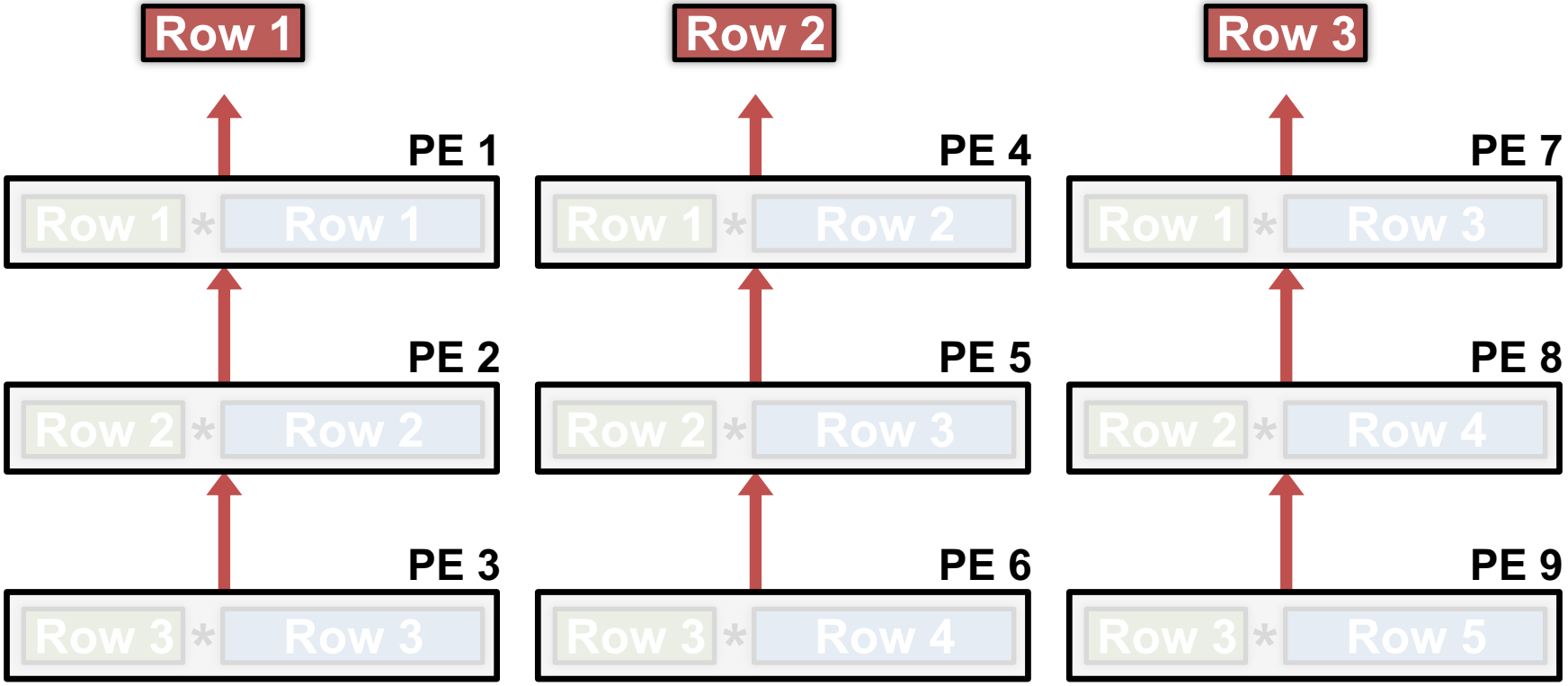
Filter rows are reused across PEs **horizontally**

Convolutional Reuse Maximized



Fmap rows are reused across PEs **diagonally**

Maximize 2D Accumulation in PE Array



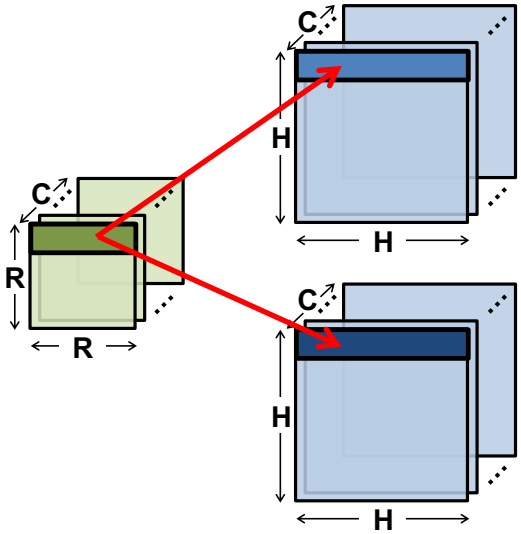
Partial sums accumulate across PEs **vertically**

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

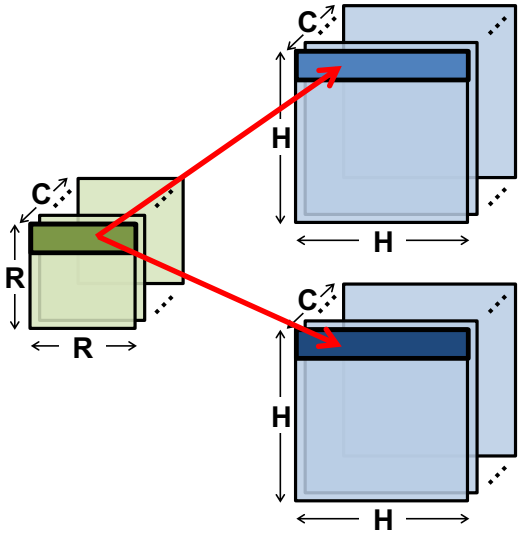


	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

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

Filter Reuse in PE

1 Multiple Fmaps



	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

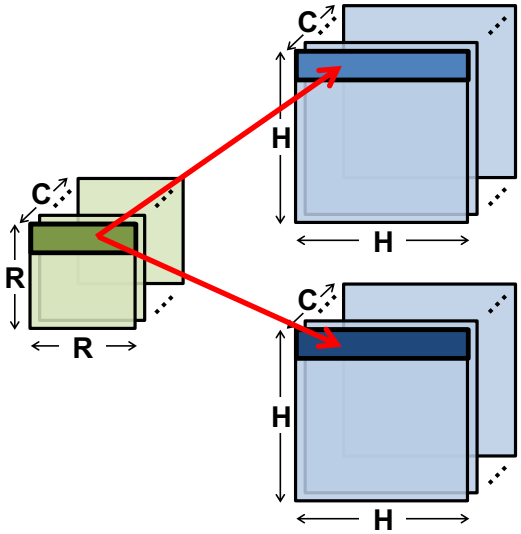
share the same filter row

2 Multiple Filters

3 Multiple Channels

Filter Reuse in PE

1 Multiple Fmaps



	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

2 Multiple Filters

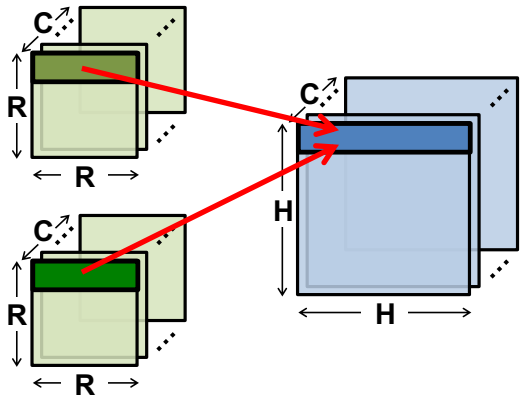
3 Multiple Channels

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

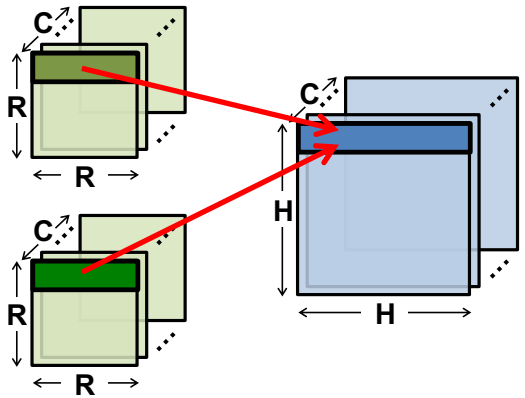


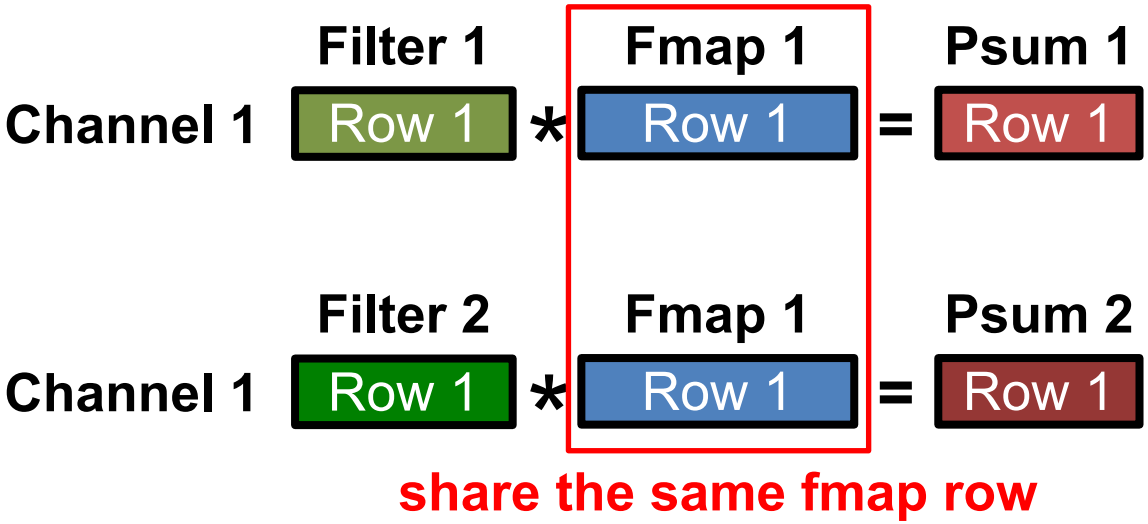
	Filter 1		Fmap 1		Psum 1
Channel 1	Row 1	*	Row 1	=	Row 1

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

Fmap Reuse in PE

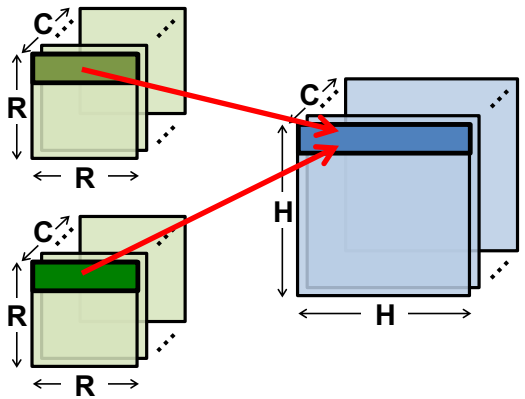
1 Multiple Fmaps **2 Multiple Filters** 3 Multiple Channels





Fmap Reuse in PE

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



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

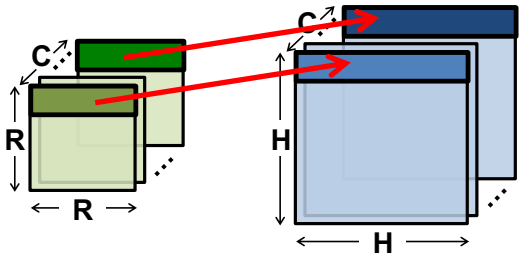
share the same fmap row

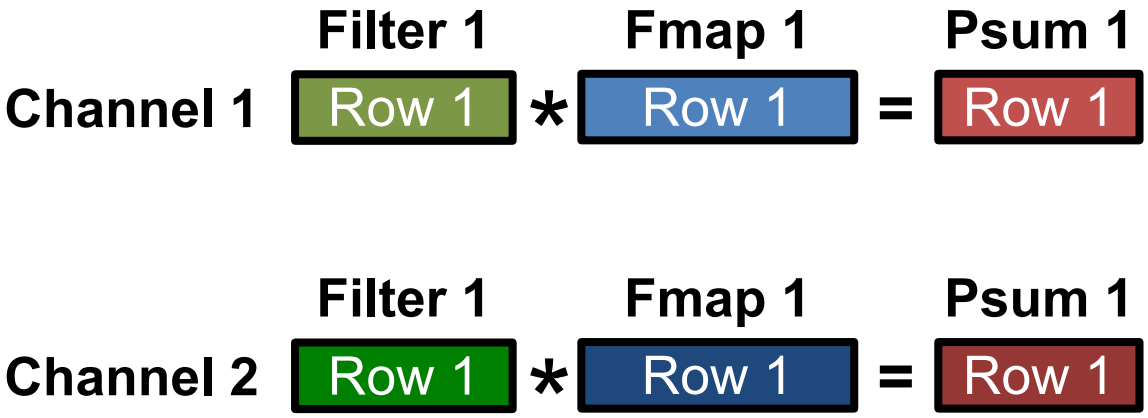
Processing in PE: **interleave filter rows**

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

Channel Accumulation in PE

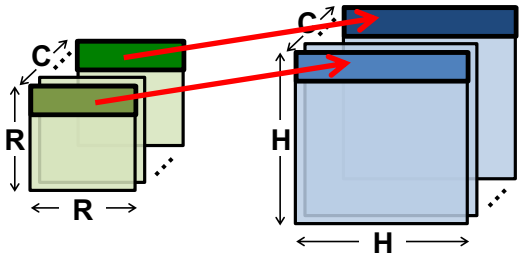
1 Multiple Fmaps 2 Multiple Filters **3 Multiple Channels**

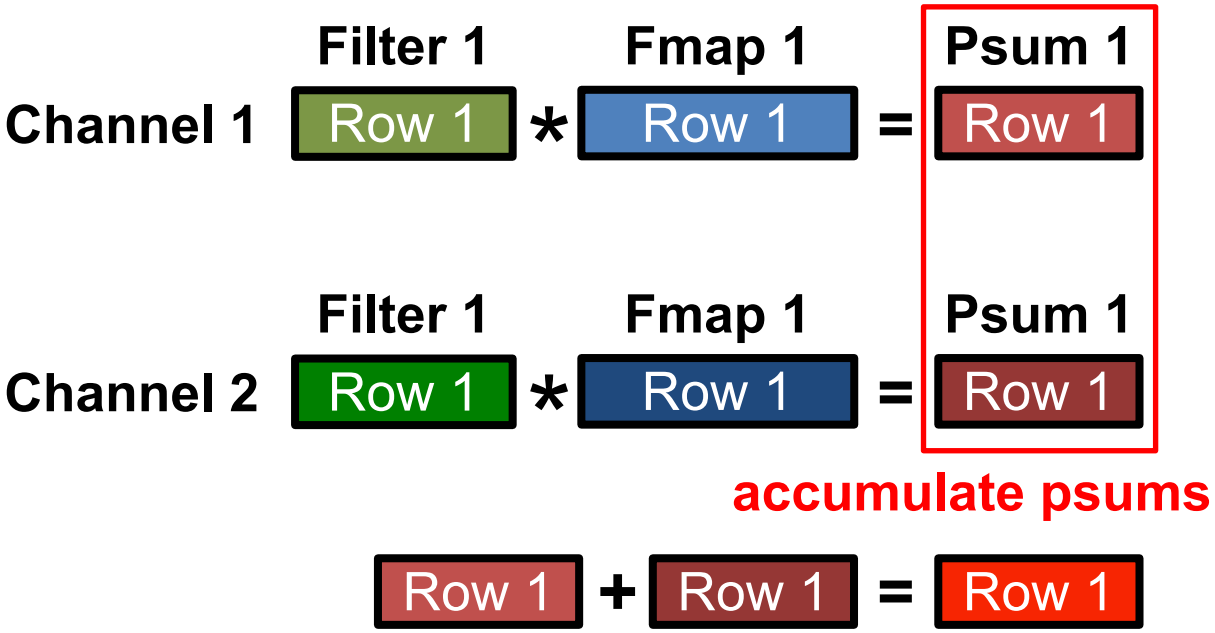




Channel Accumulation in PE

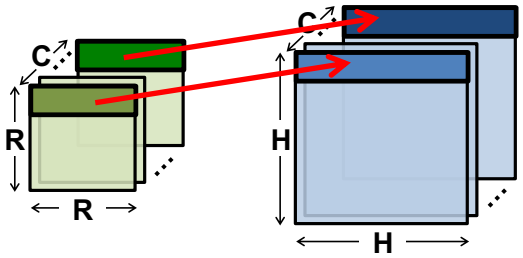
1 Multiple Fmaps 2 Multiple Filters **3 Multiple Channels**

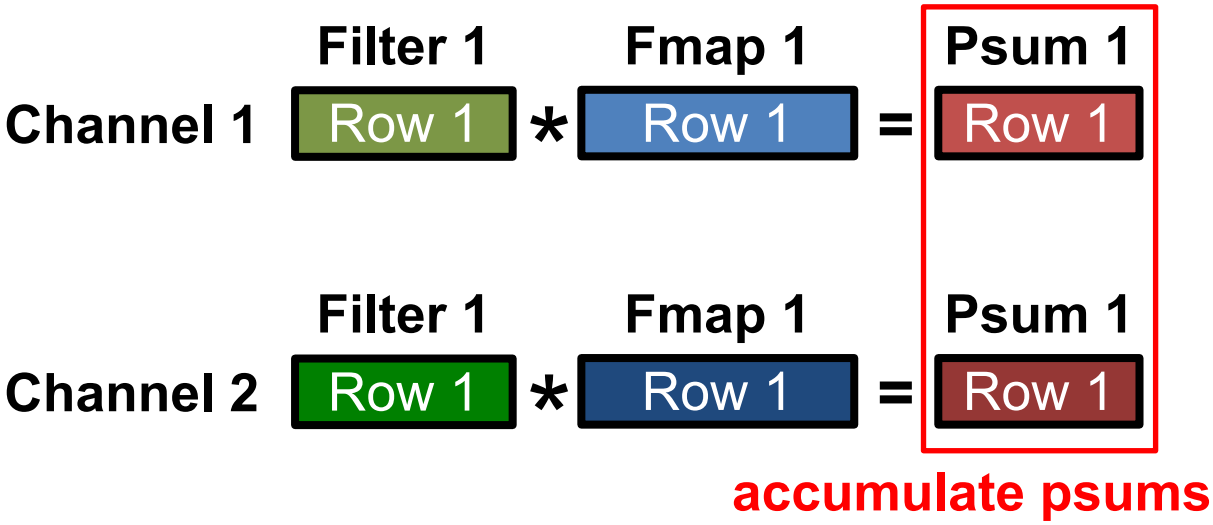




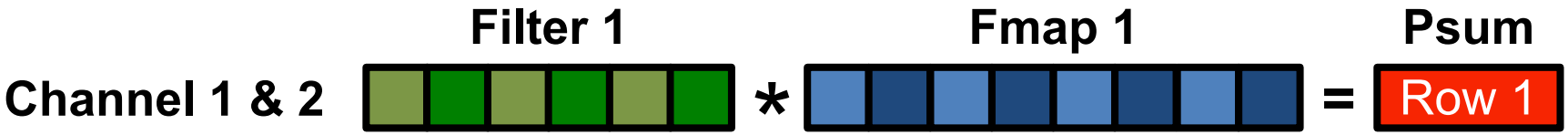
Channel Accumulation in PE

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

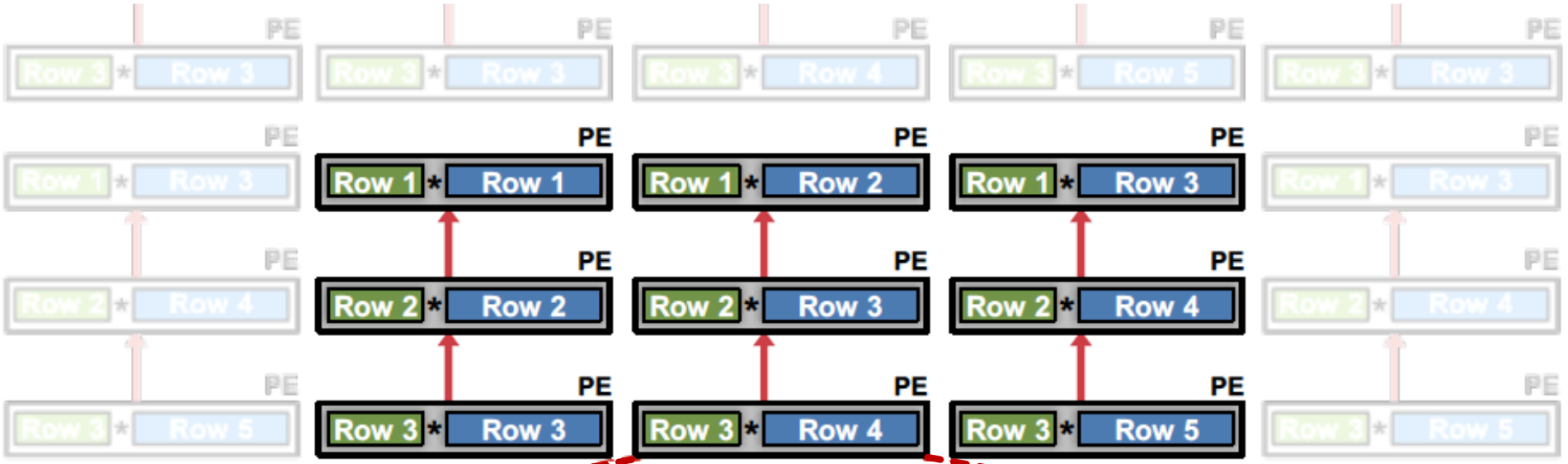




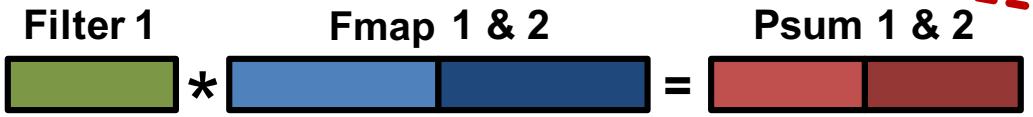
Processing in PE: **interleave channels**



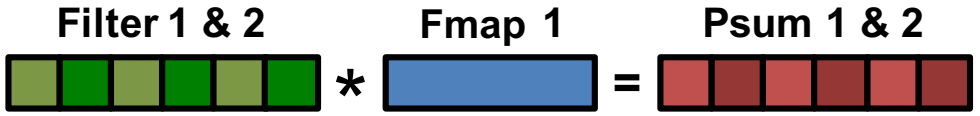
DNN Processing – The Full Picture



Multiple **fmaps**:



Multiple **filters**:

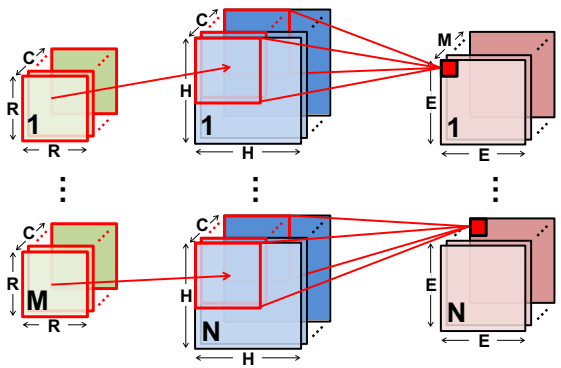


Multiple **channels**:

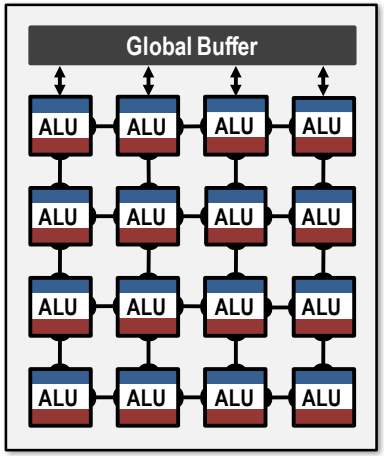
Map rows from **multiple fmaps, filters** and **channels** to same PE to exploit other forms of reuse and local accumulation

Optimal Mapping in Row Stationary

DNN Configurations

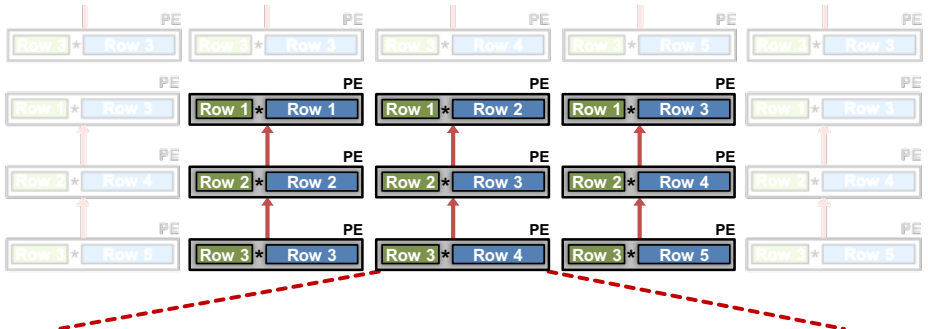


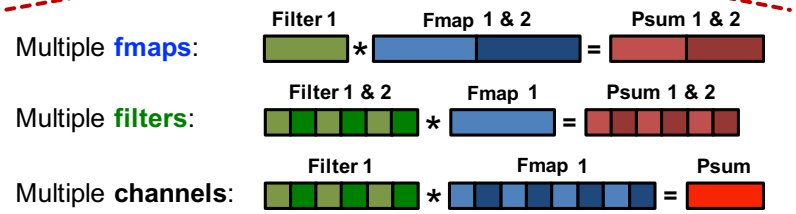
Hardware Resources



Optimization Compiler (Mapper)

Row Stationary Mapping

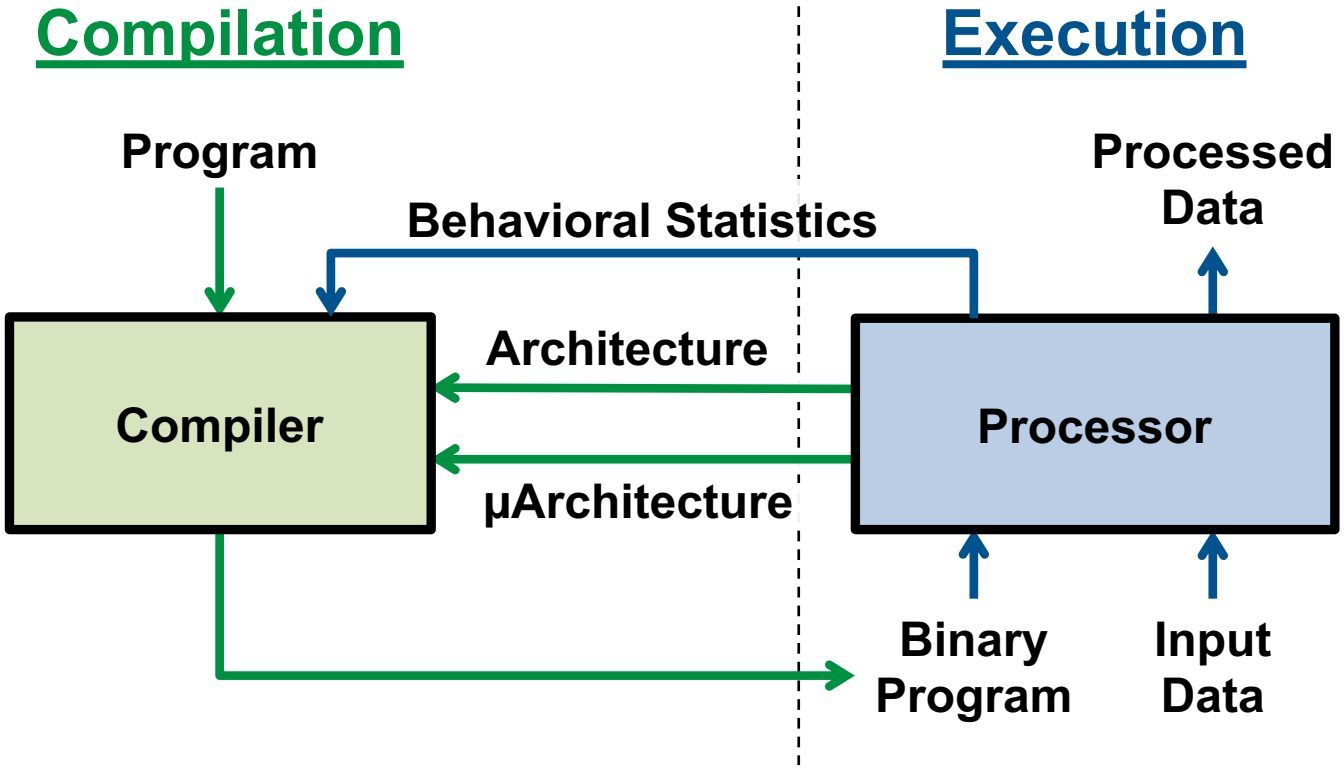




[Chen et al., ISCA 2016]

Computer Architecture Analogy

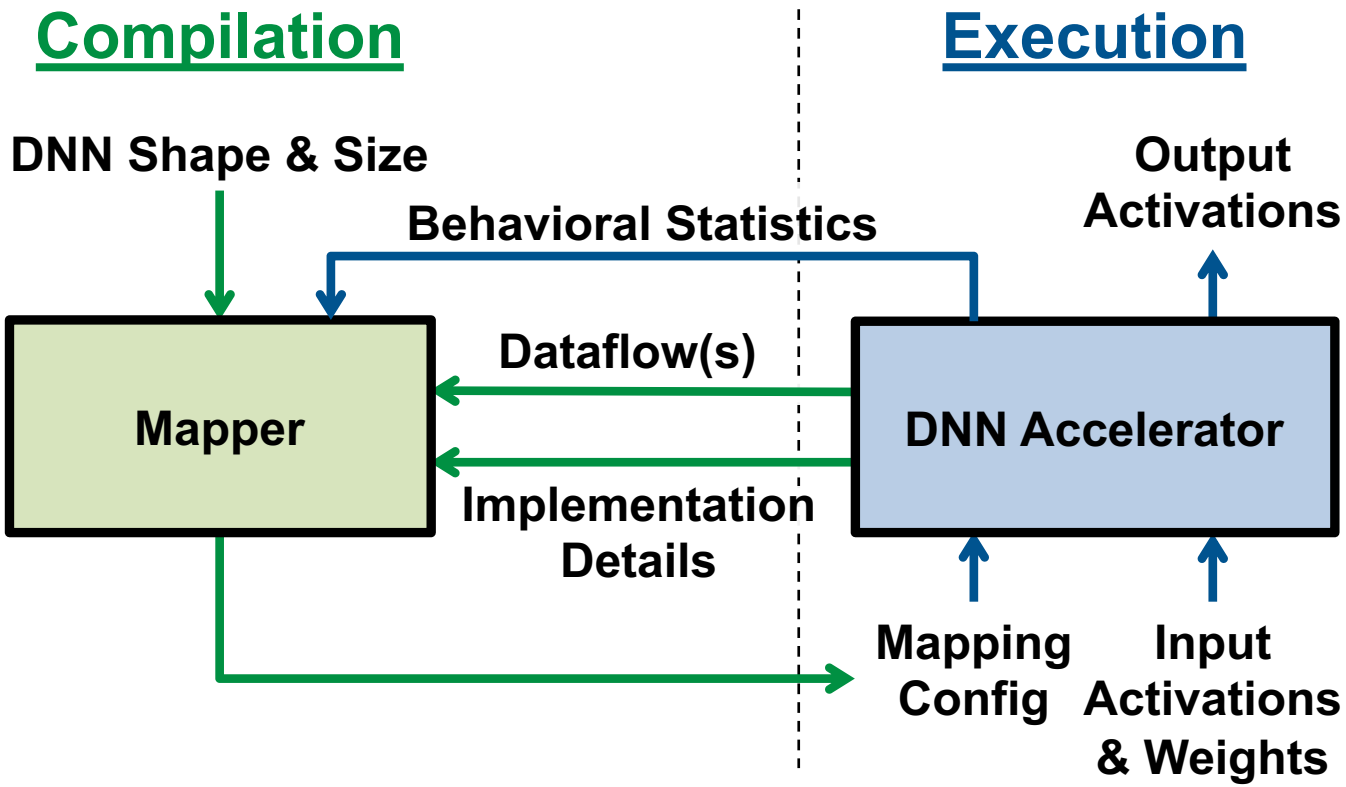
CPU Compute Model



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

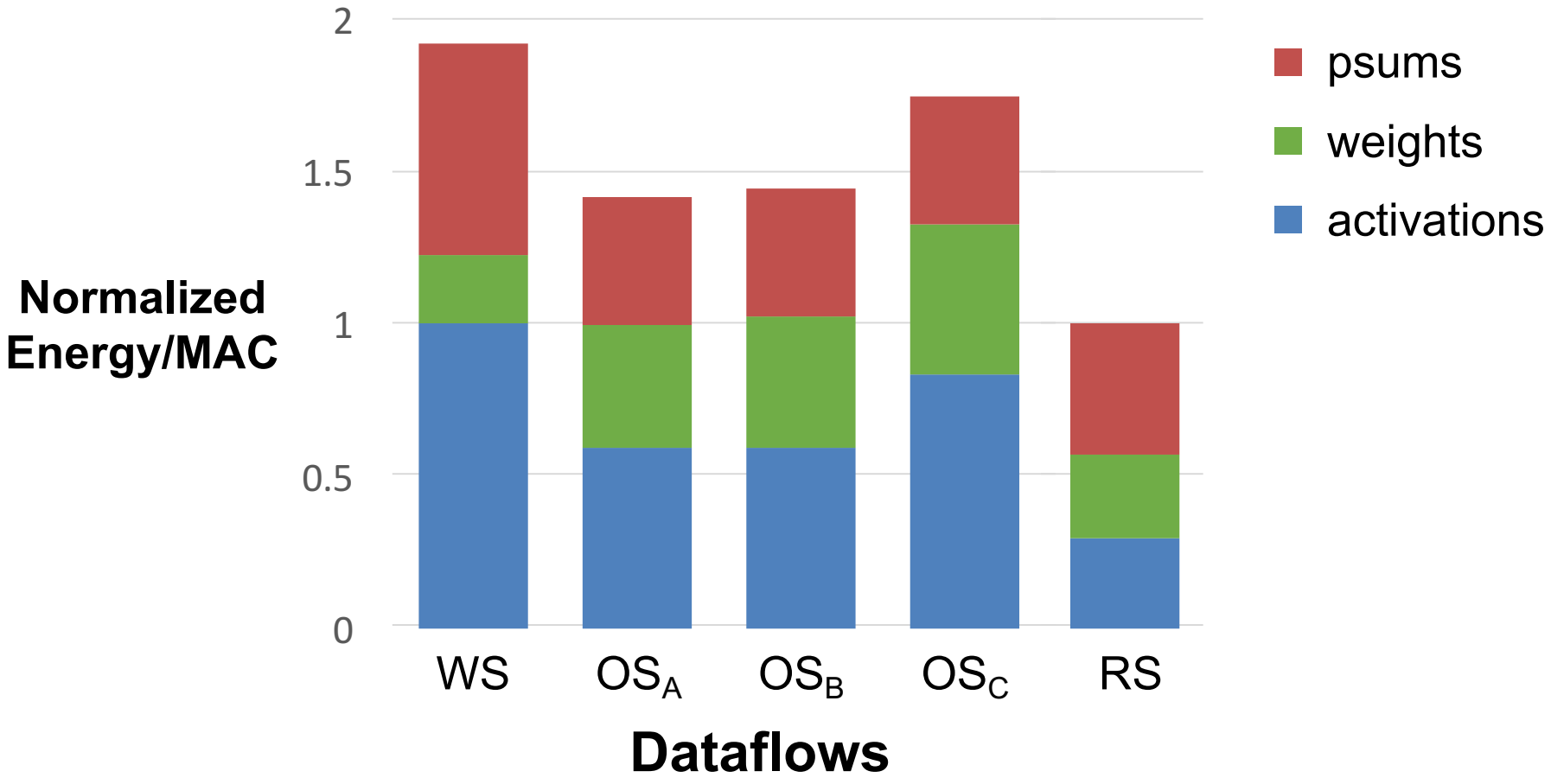
Computer Architecture Analogy

DNN Compute Model



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

Dataflow Comparison



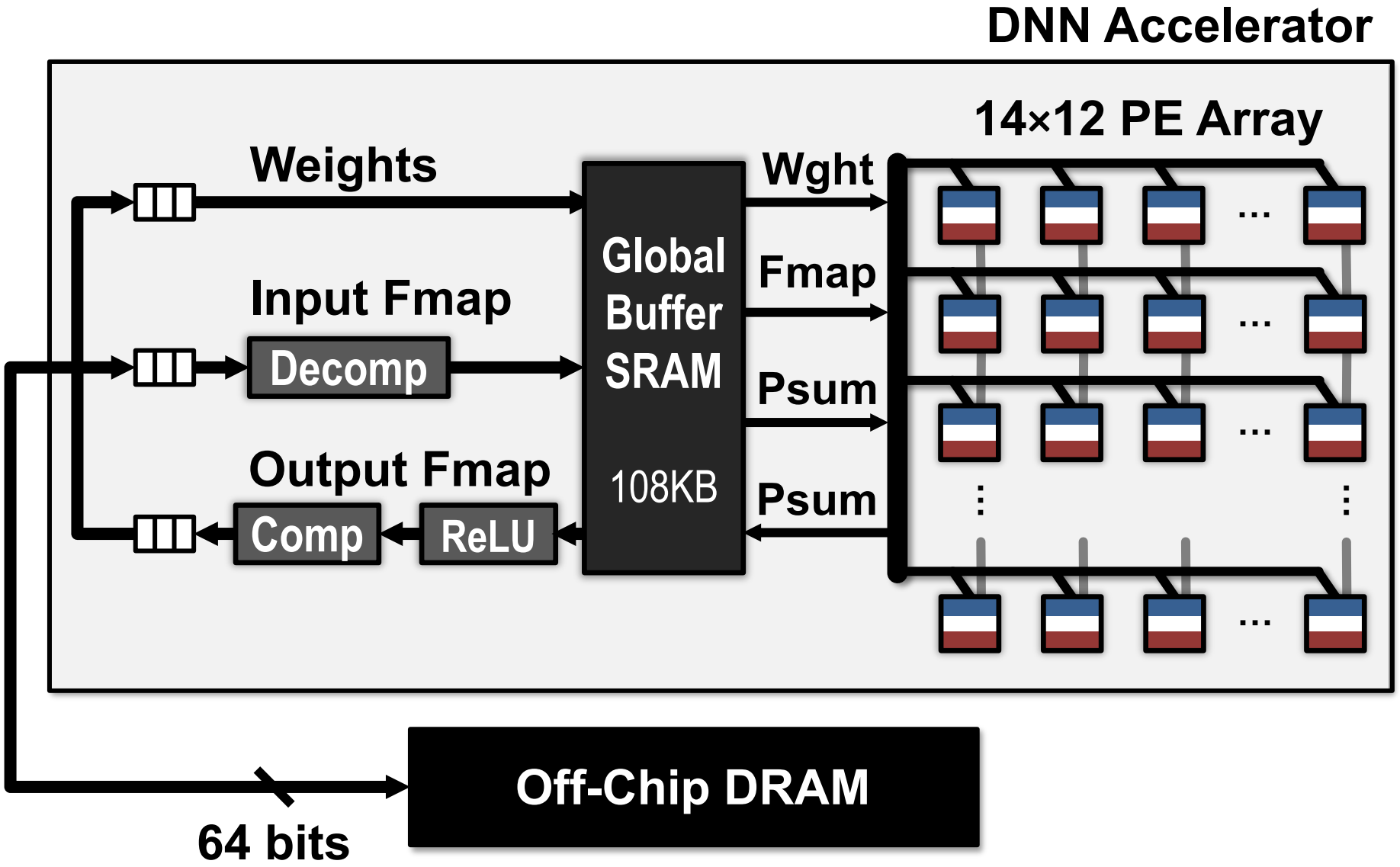
RS optimizes for the best **overall** energy efficiency

[Chen et al., ISCA 2016]

Hardware Architecture for RS Dataflow

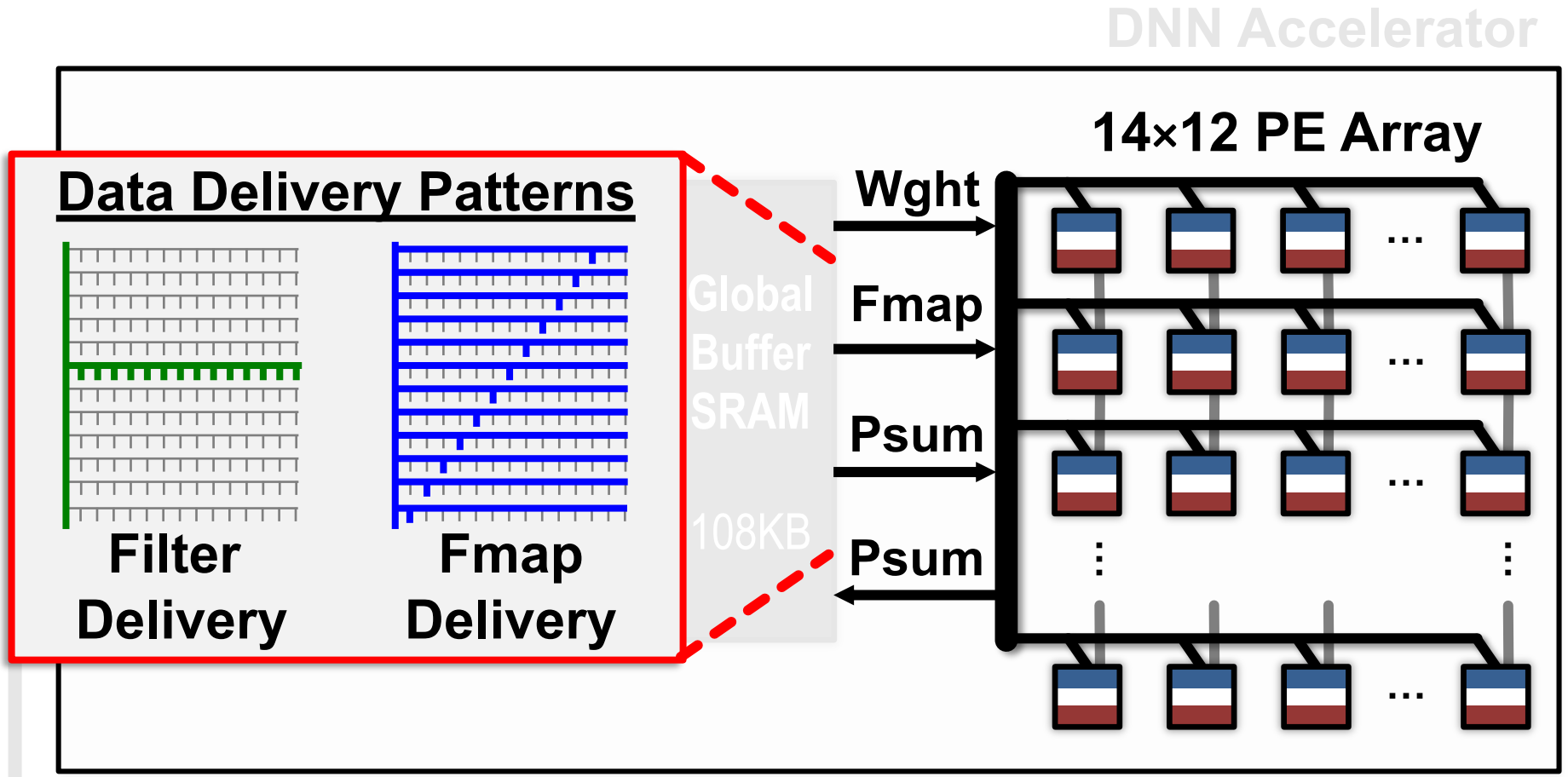
[Chen et al., ISSCC 2016]

Eyeriss DNN Accelerator



[Chen et al., ISSCC 2016]

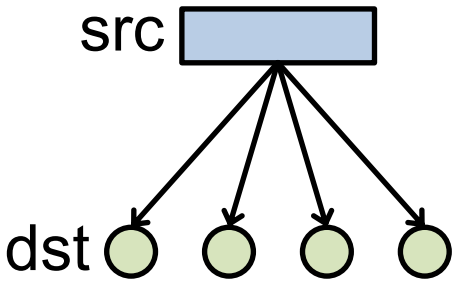
Data Delivery with On-Chip Network



Multicast NoC to support any delivery patterns

On-Chip Network (NoC) is the Bottleneck

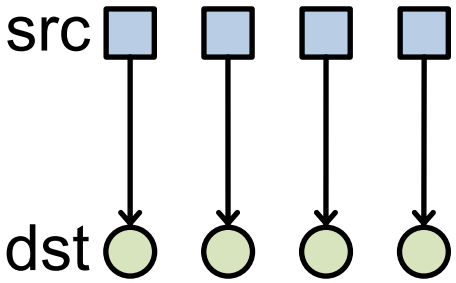
Broadcast Network
(Eyeriss v1 NoC)



High Reuse

Low Bandwidth

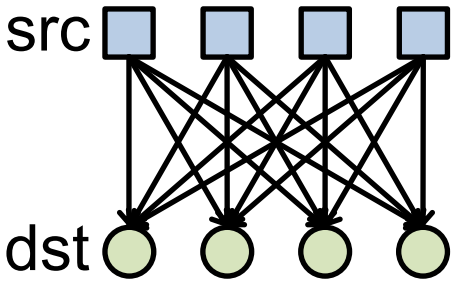
Unicast Networks



Low Reuse

High Bandwidth

All-to-All Networks

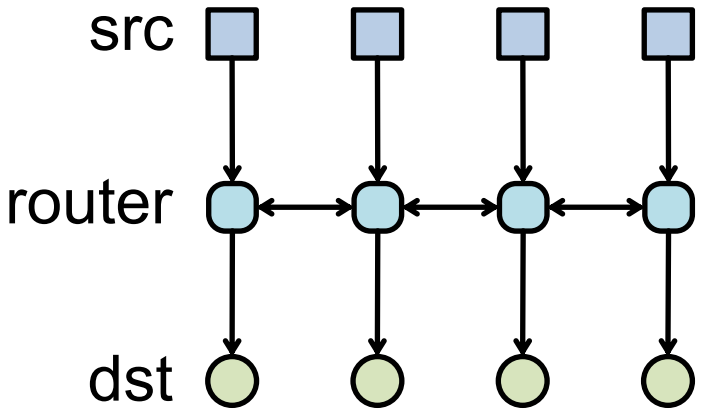


High Reuse

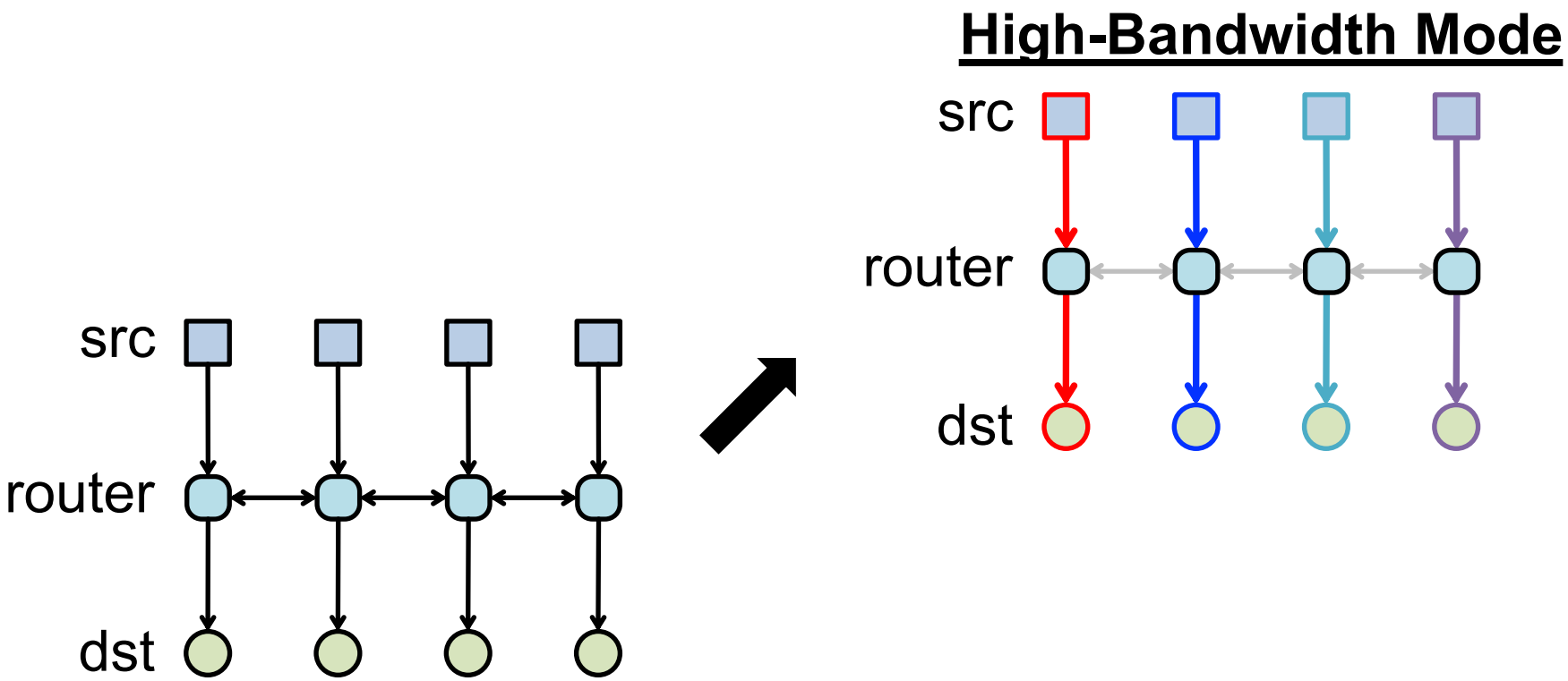
High Bandwidth

Hard to Scale

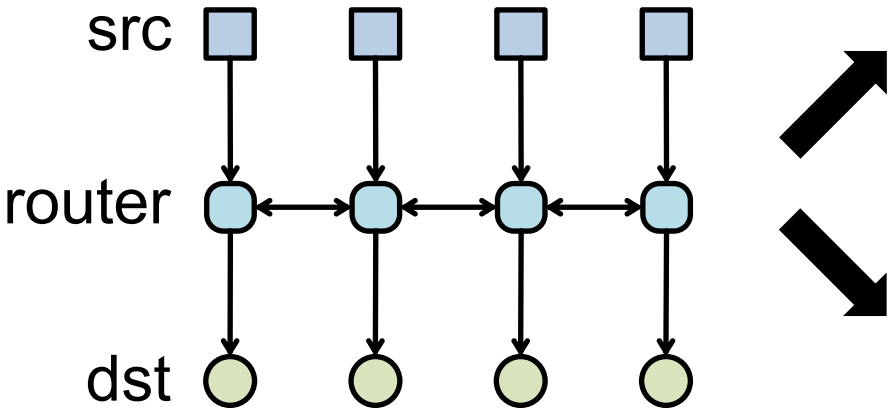
Mesh Network – Best of Both Worlds



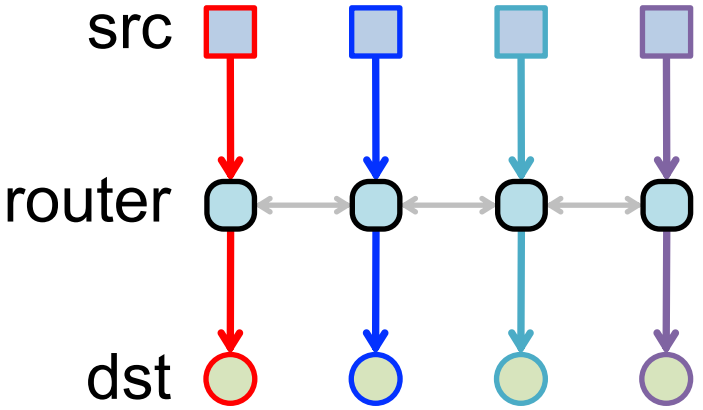
Mesh Network – Best of Both Worlds



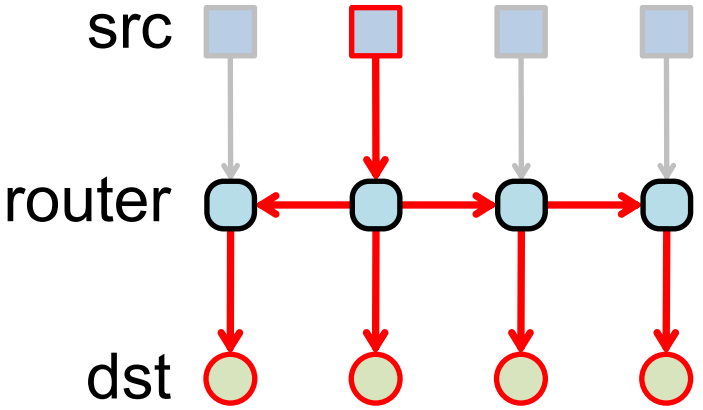
Mesh Network – Best of Both Worlds



High-Bandwidth Mode

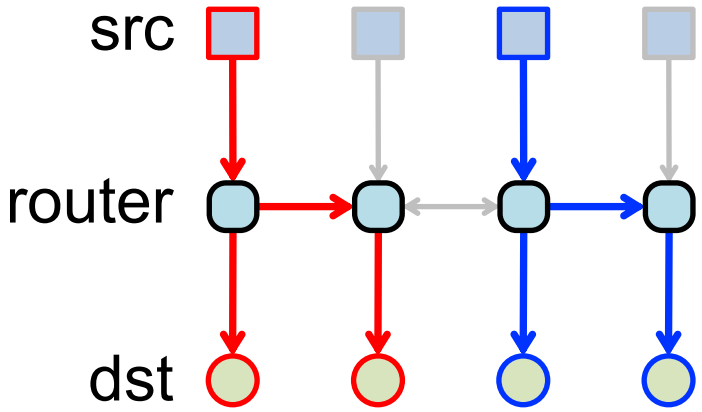


High-Reuse Mode



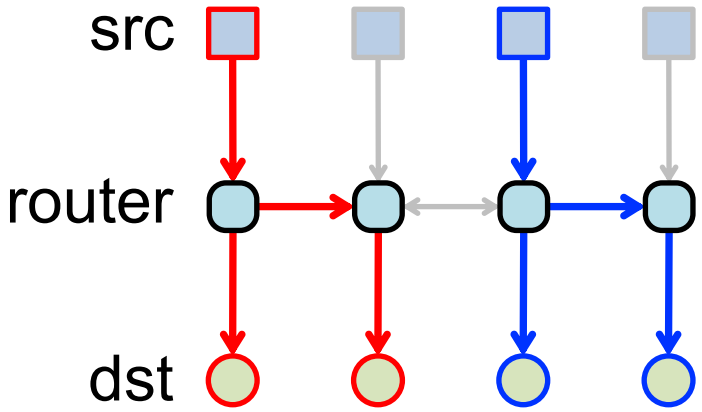
Mesh Network – More Complicated Cases

Grouped-Multicast Mode

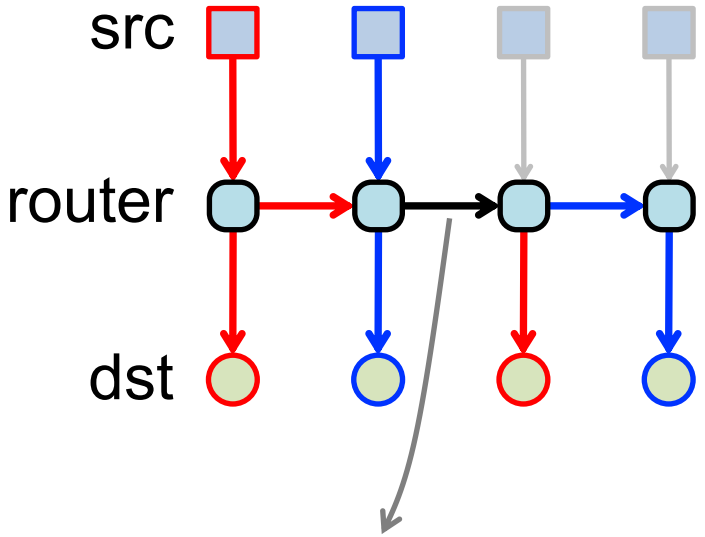


Mesh Network – More Complicated Cases

Grouped-Multicast Mode

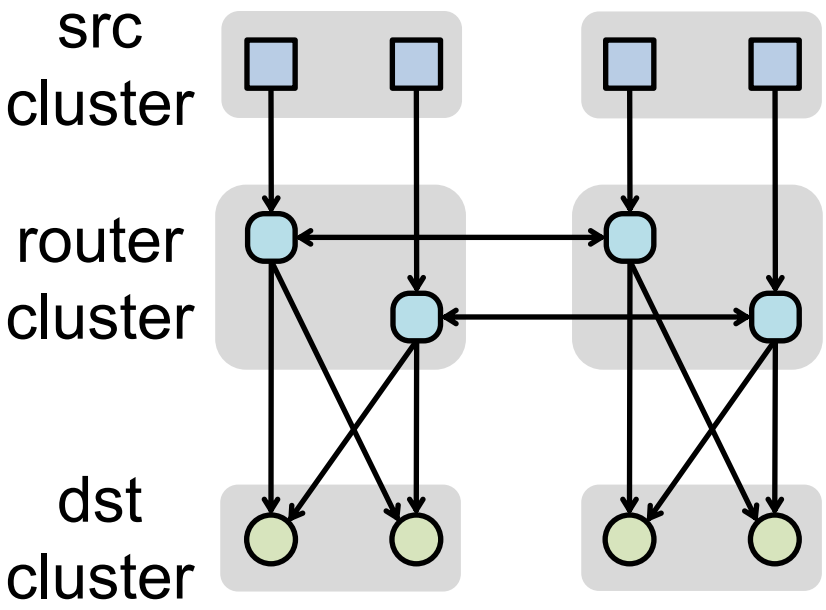


Interleaved-Multicast Mode



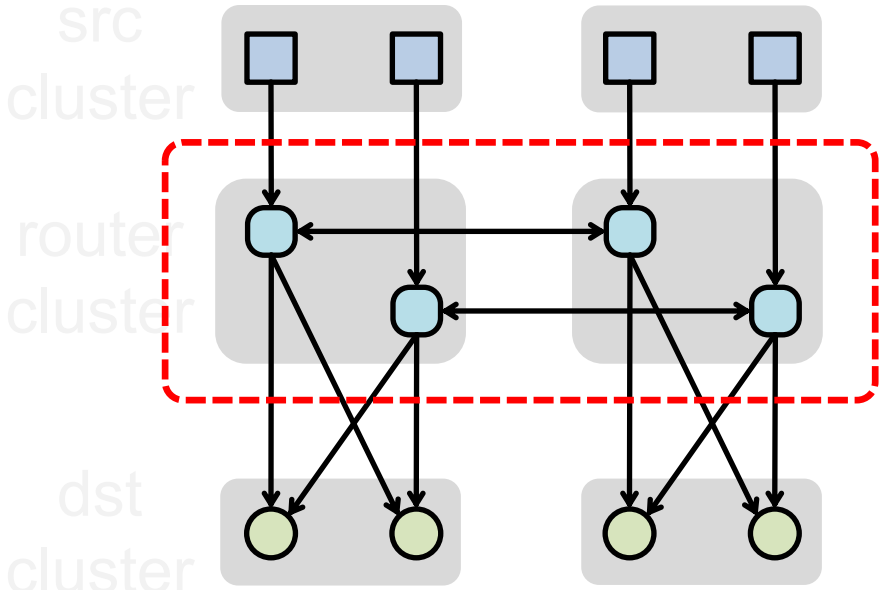
Bandwidth-limited route
(flow control required)

Design of Hierarchical Mesh Network

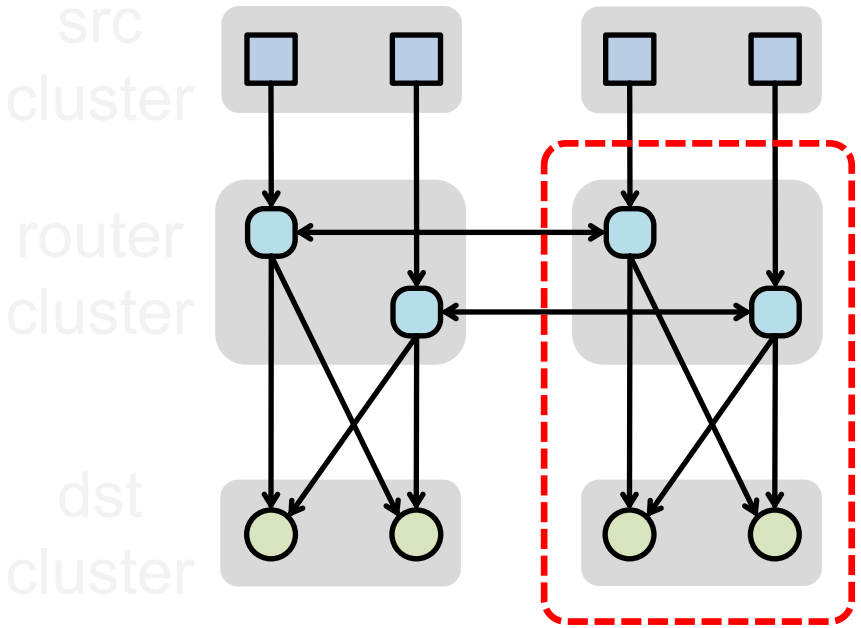


Design of Hierarchical Mesh Network

Mesh Network for **inter-cluster** connections



Design of Hierarchical Mesh Network

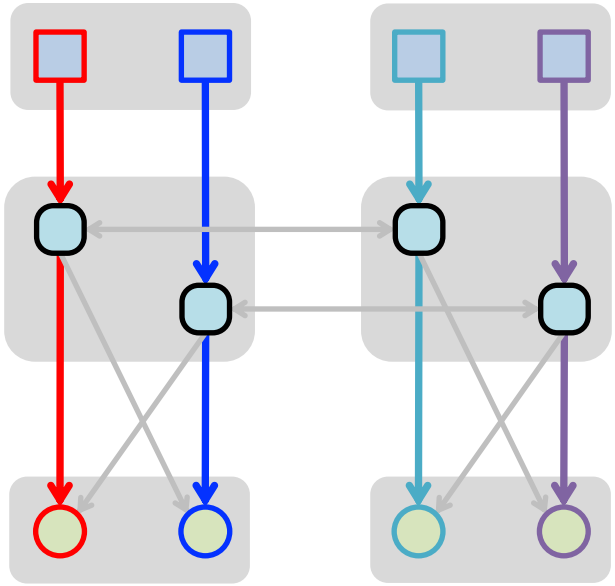


All-to-All Network for **intra-cluster** connections

Complexity is contained within a cluster

Design of Hierarchical Mesh Network

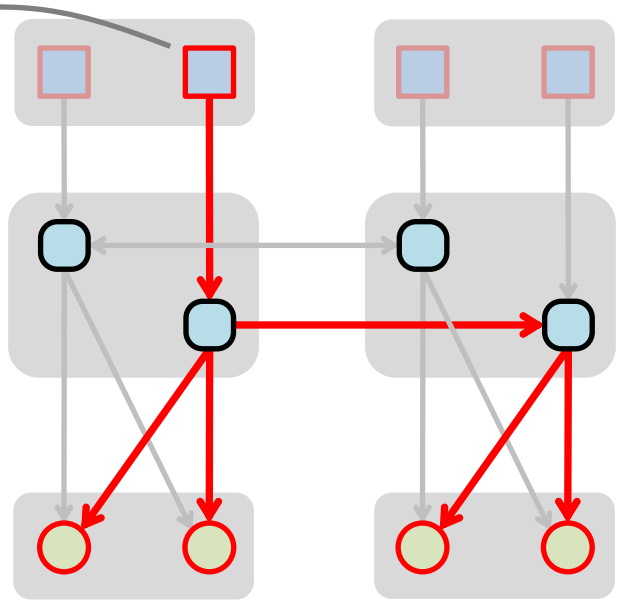
High-Bandwidth Mode



Design of Hierarchical Mesh Network

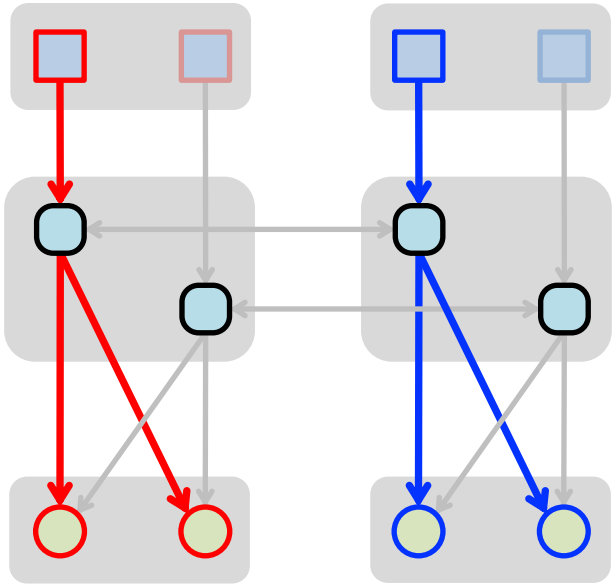
High-Reuse Mode

from any one src



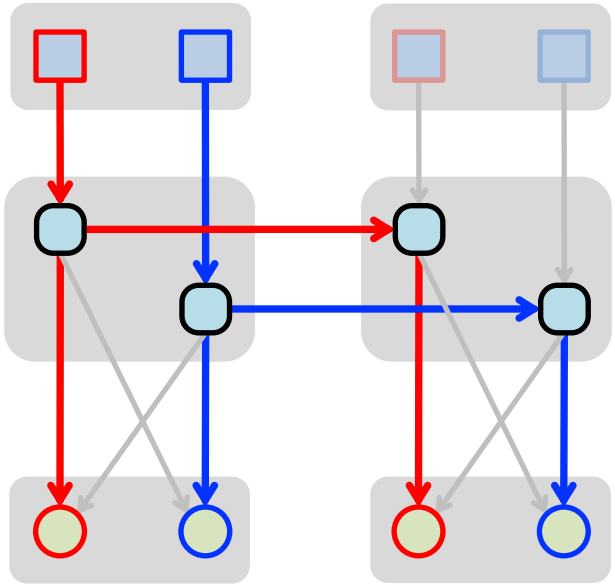
Design of Hierarchical Mesh Network

Grouped-Multicast Mode



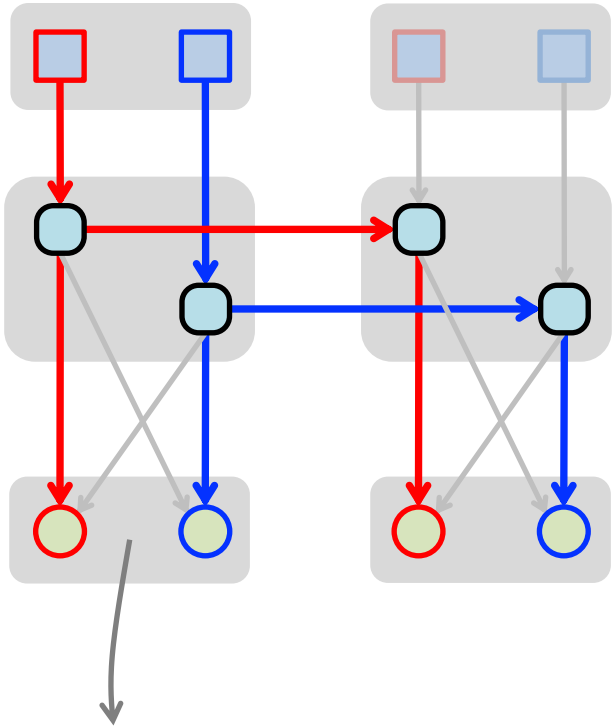
Design of Hierarchical Mesh Network

Interleaved-Multicast Mode



Design of Hierarchical Mesh Network

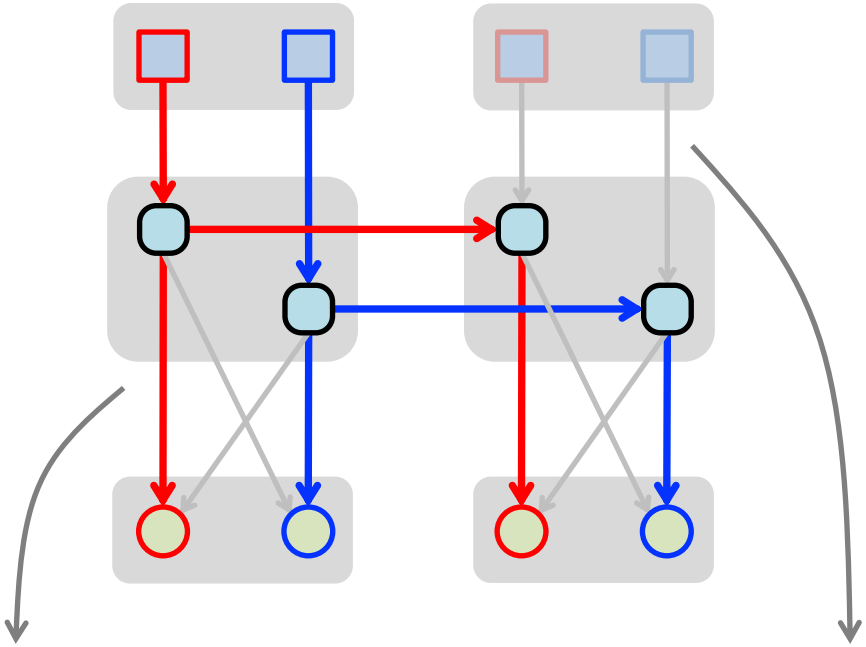
Interleaved-Multicast Mode



Can interleave more by scaling up the cluster size

Design of Hierarchical Mesh Network

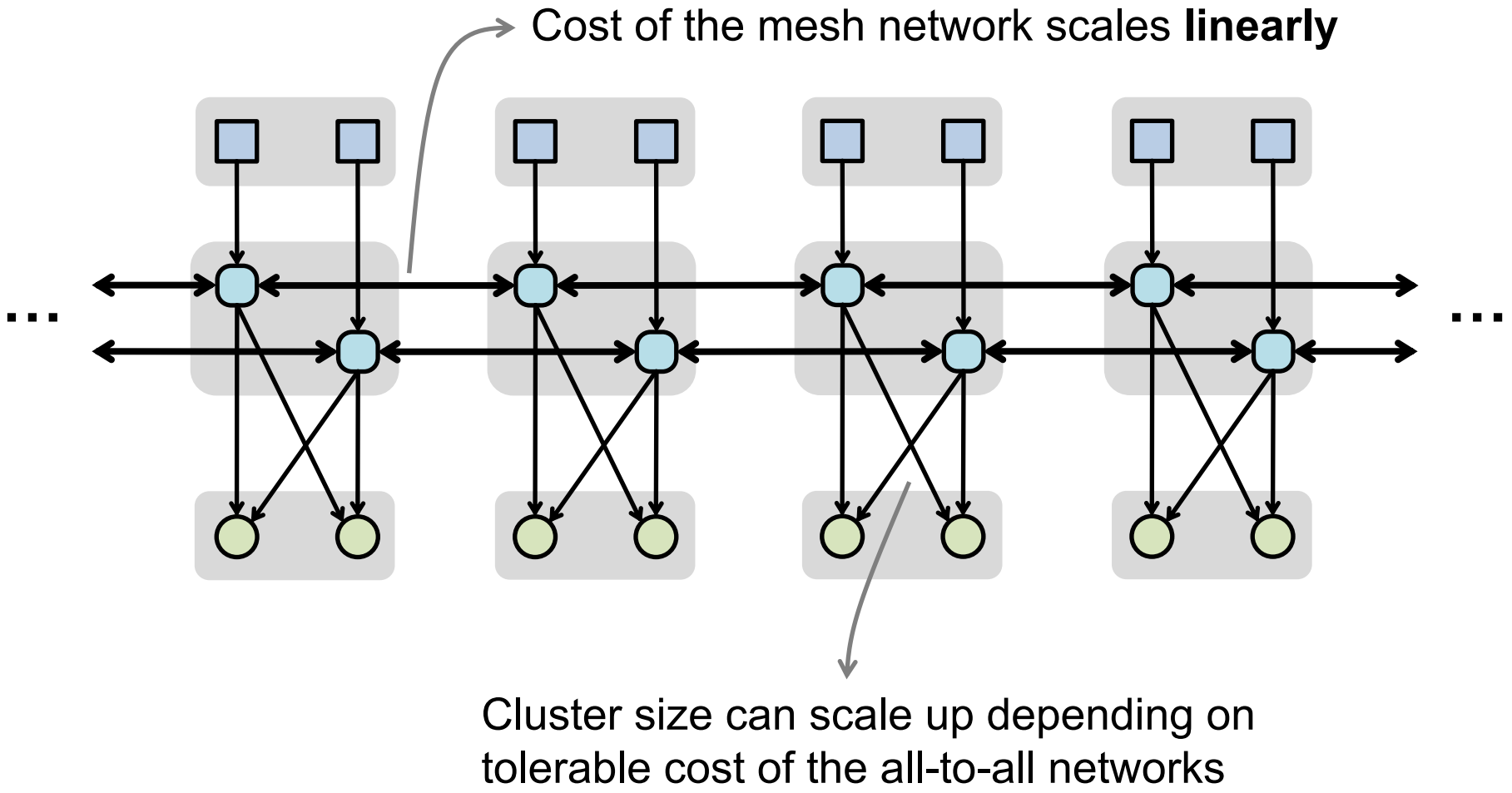
Interleaved-Multicast Mode



All routes are determined **at configuration time**

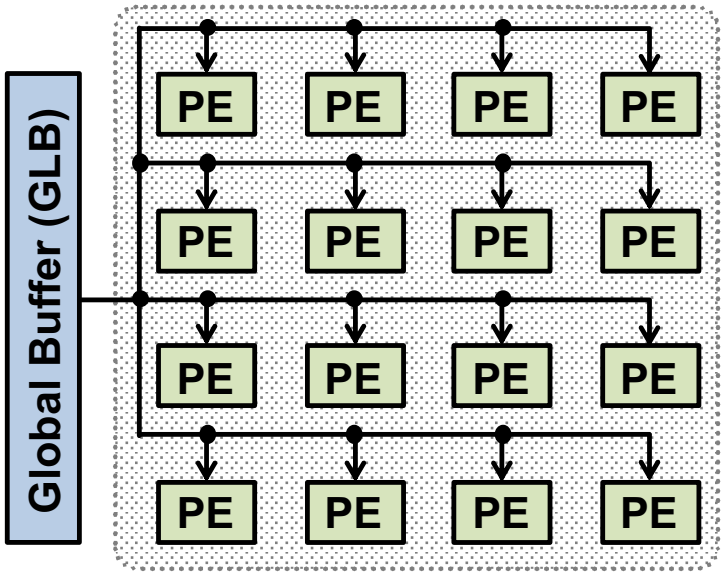
→ **Routers** are **circuit-switched** (only **MUXes**)

Scaling the Hierarchical Mesh Network

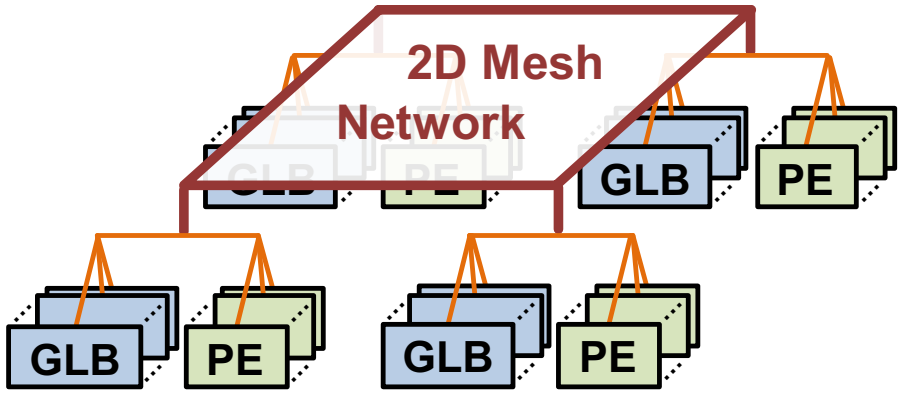


Eyeriss with Hierarchical Mesh Network

Eyeriss v1

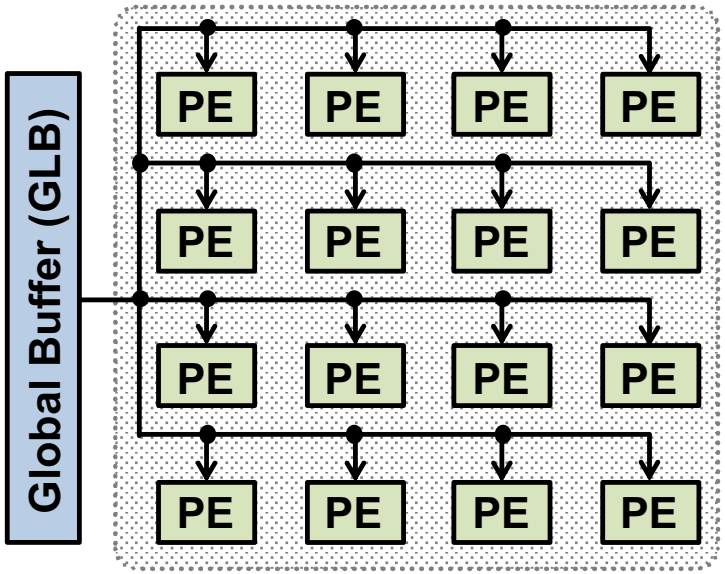


Eyeriss v2

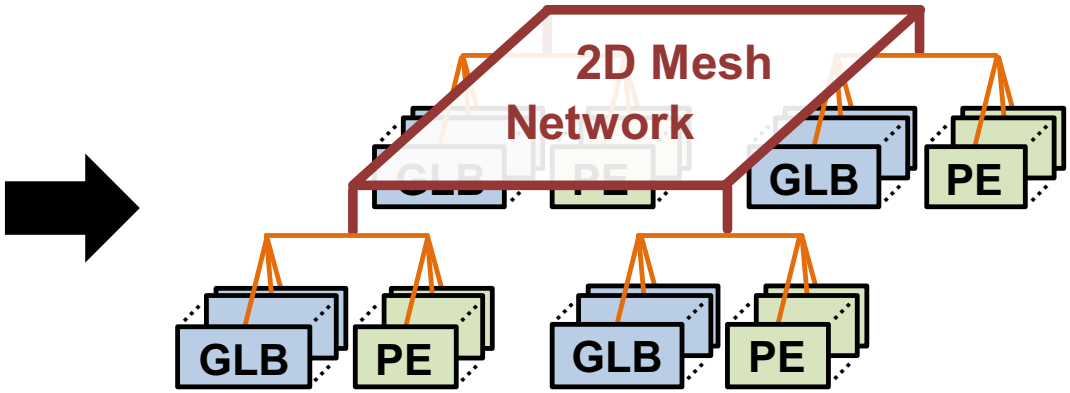


Eyeriss with Hierarchical Mesh Network

Eyeriss v1



Eyeriss v2



	Speedup	Energy Efficiency
AlexNet	6.9×	2.6×
MobileNet	5.6×	1.8×

* not including the benefits from the sparsity features in v2

Summary

- **Data reuse** is the key to achieving **high energy efficiency**.
- High PE utilization with **adaptive on-chip networks** is the key to achieving **high performance**
- Co-design of **dataflow** and **hardware** is critical for the optimization of **performance**, **energy efficiency** and **flexibility** for DNN accelerators.