

DNN Accelerator Architectures

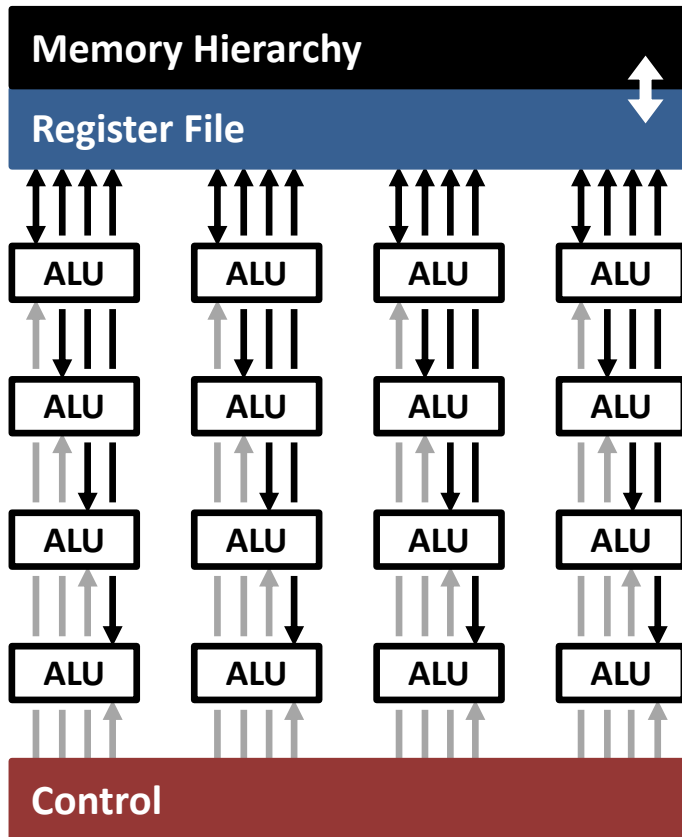
ISCA Tutorial (2019)

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

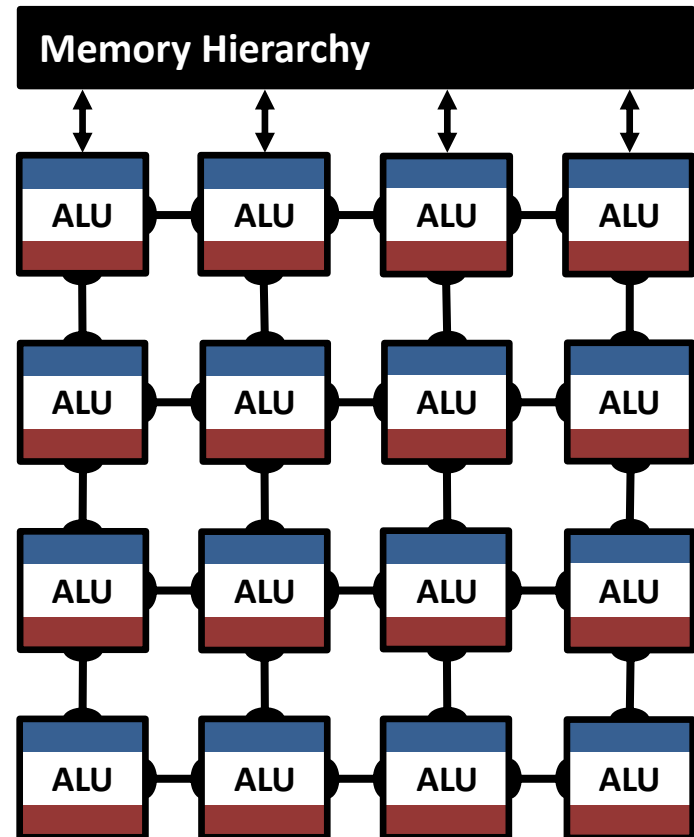
Joel Emer, Vivienne Sze, Yu-Hsin Chen

Highly-Parallel Compute Paradigms

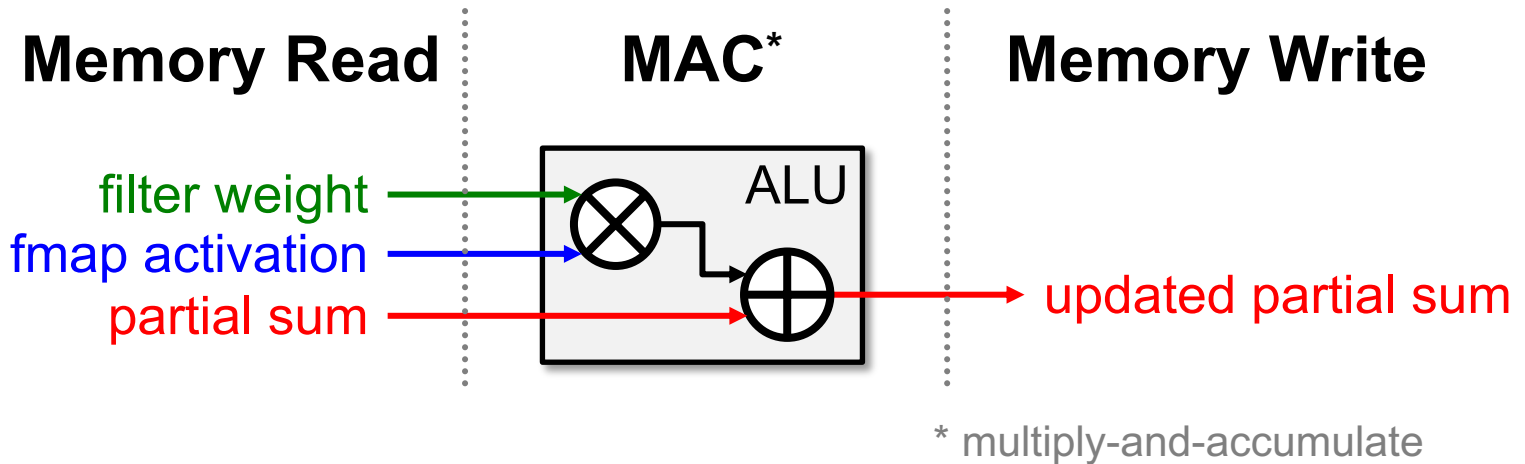
Temporal Architecture
(SIMD/SIMT)



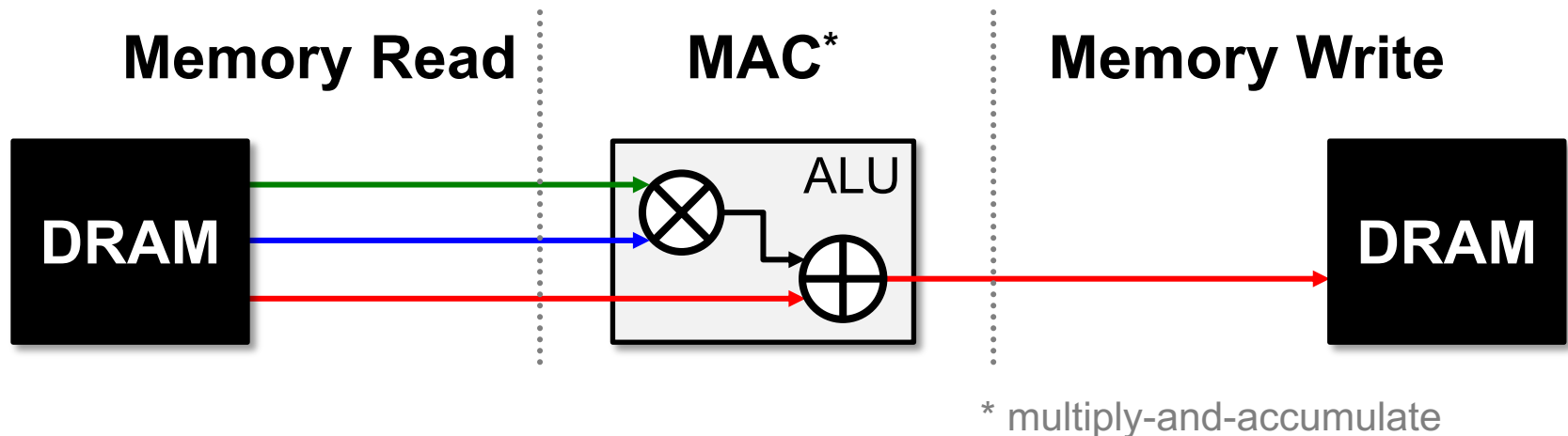
Spatial Architecture
(Dataflow Processing)



Memory Access is the Bottleneck



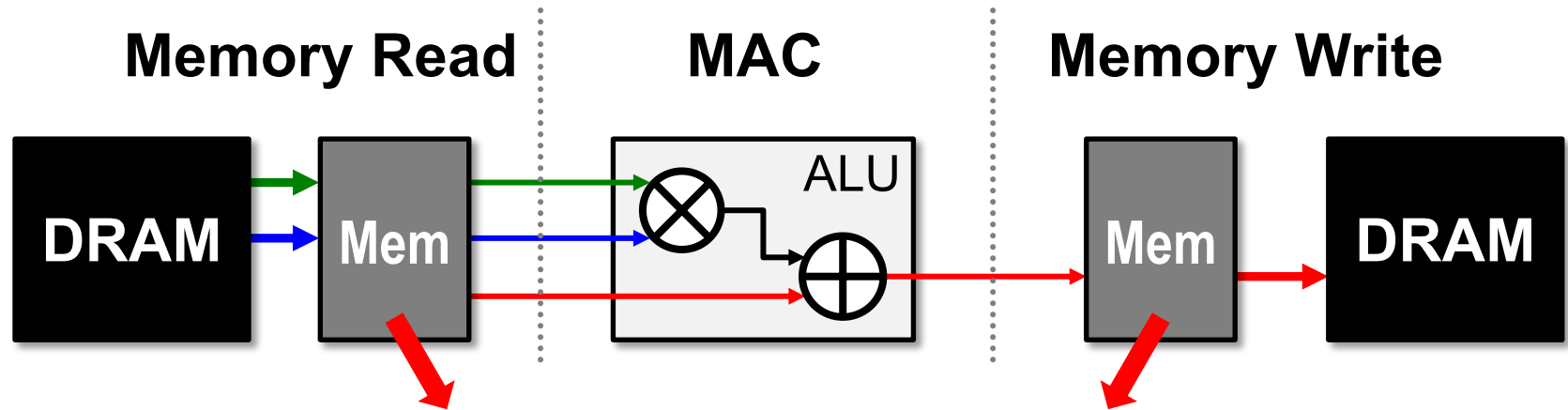
Memory Access is the Bottleneck



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



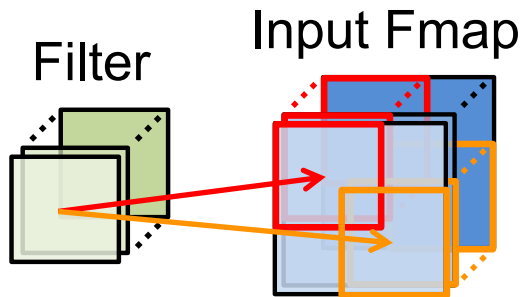
Extra levels of local memory hierarchy

Smaller, but Faster and more Energy-Efficient

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

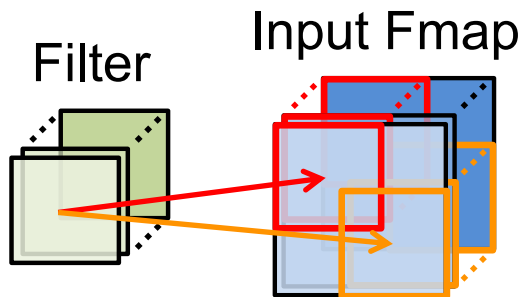


Reuse: **Activations**
Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

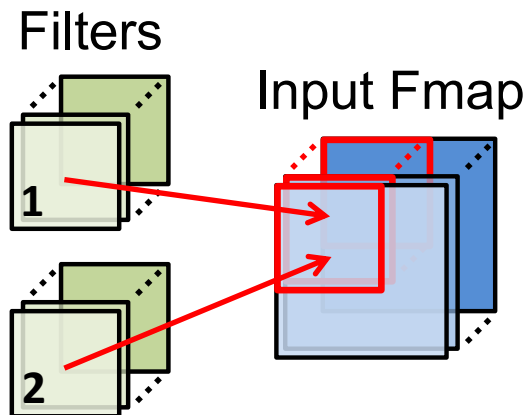
CONV layers only
(sliding window)



Reuse: **Activations**
Filter weights

Fmap Reuse

CONV and FC layers

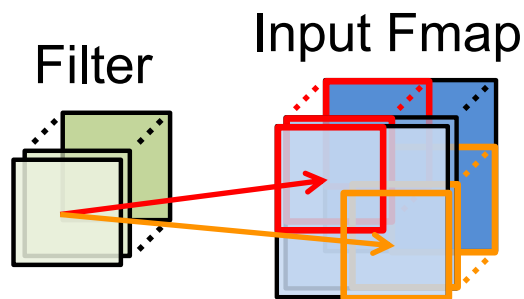


Reuse: **Activations**

Types of Data Reuse in DNN

Convolutional Reuse

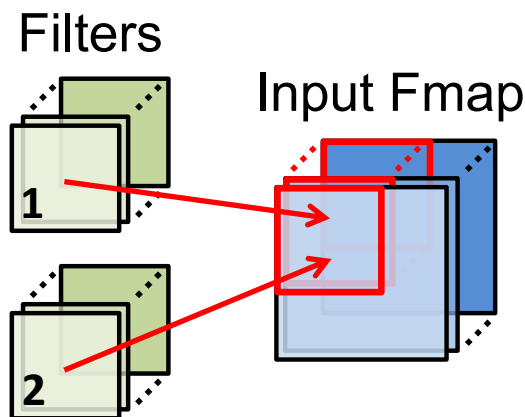
CONV layers only
(sliding window)



Reuse: **Activations**
Filter weights

Fmap Reuse

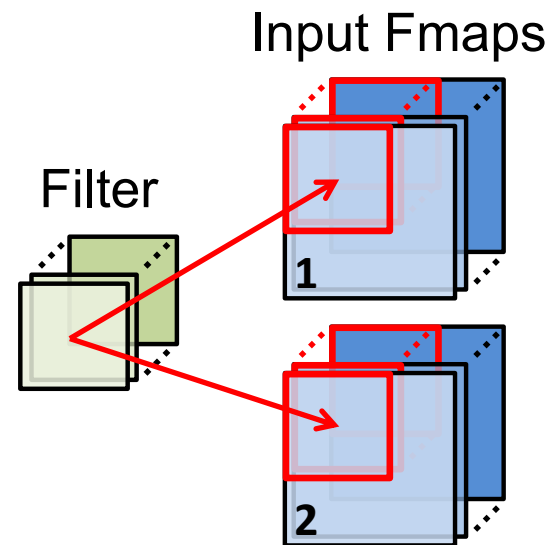
CONV and FC layers



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)

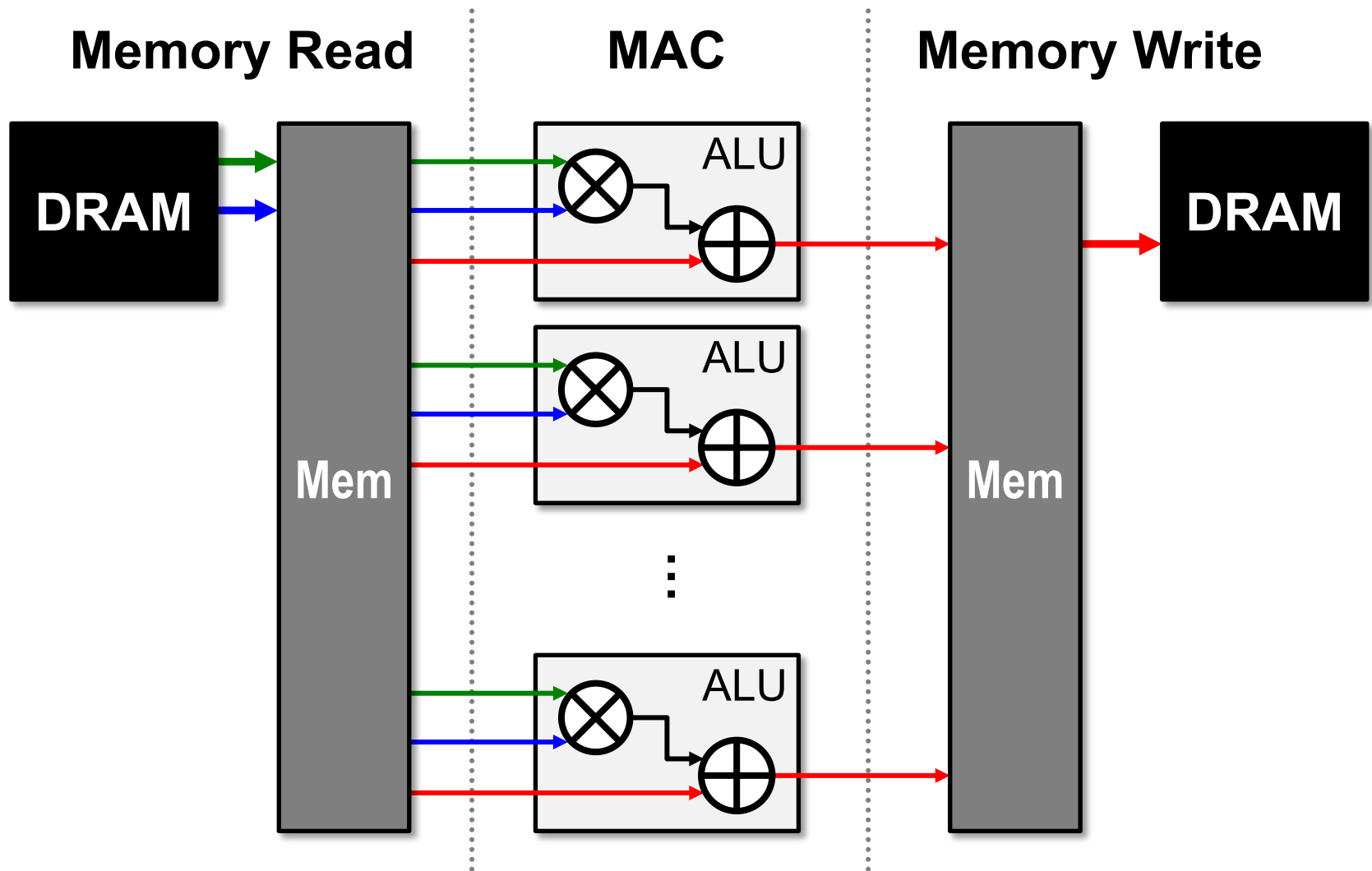
If all data reuse is exploited, DRAM accesses in AlexNet can be reduced from **2896M** to **61M** (best case)

Reuse: Activations
Filter weights

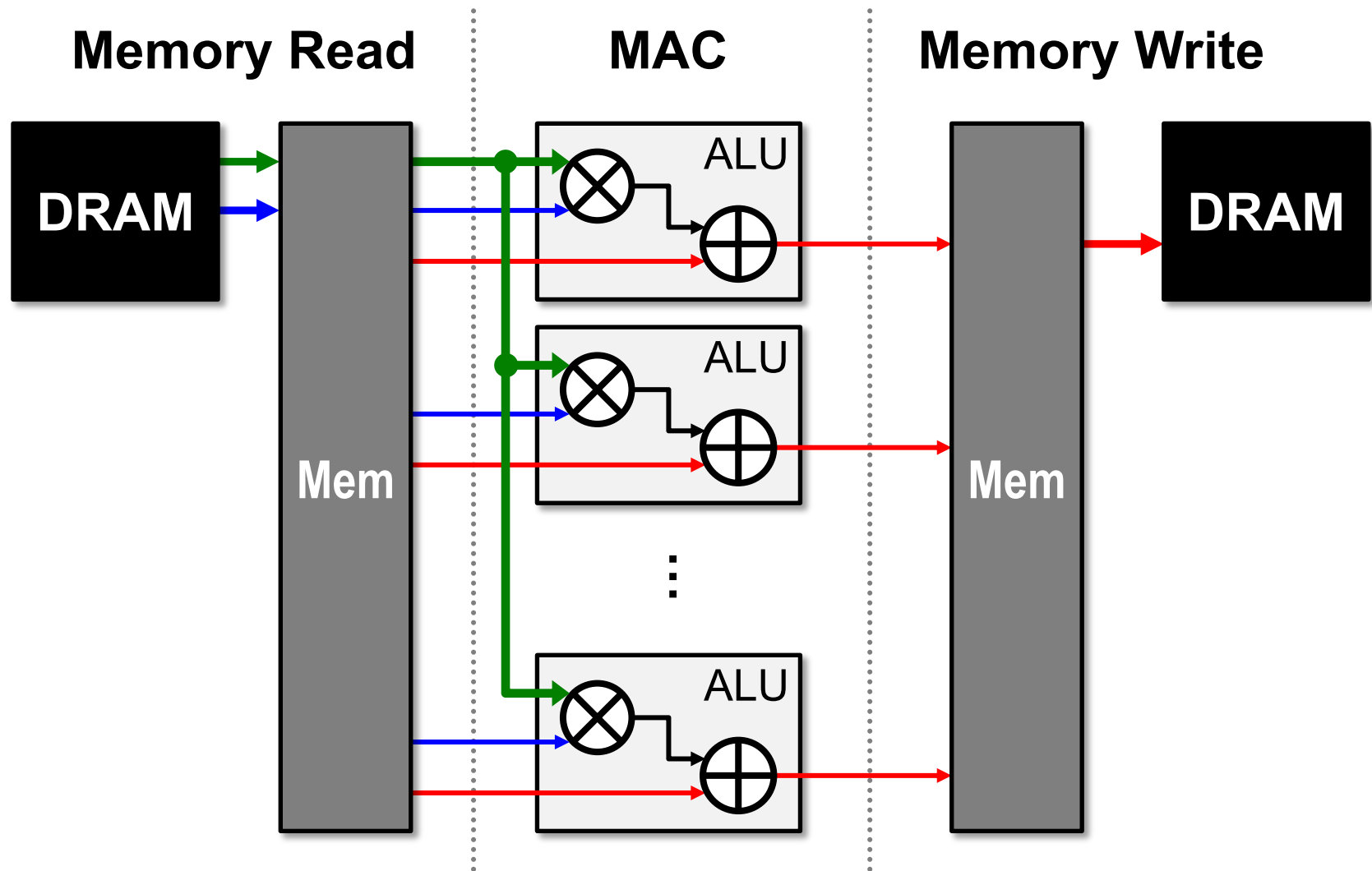
Reuse: Activations

Reuse: Filter weights

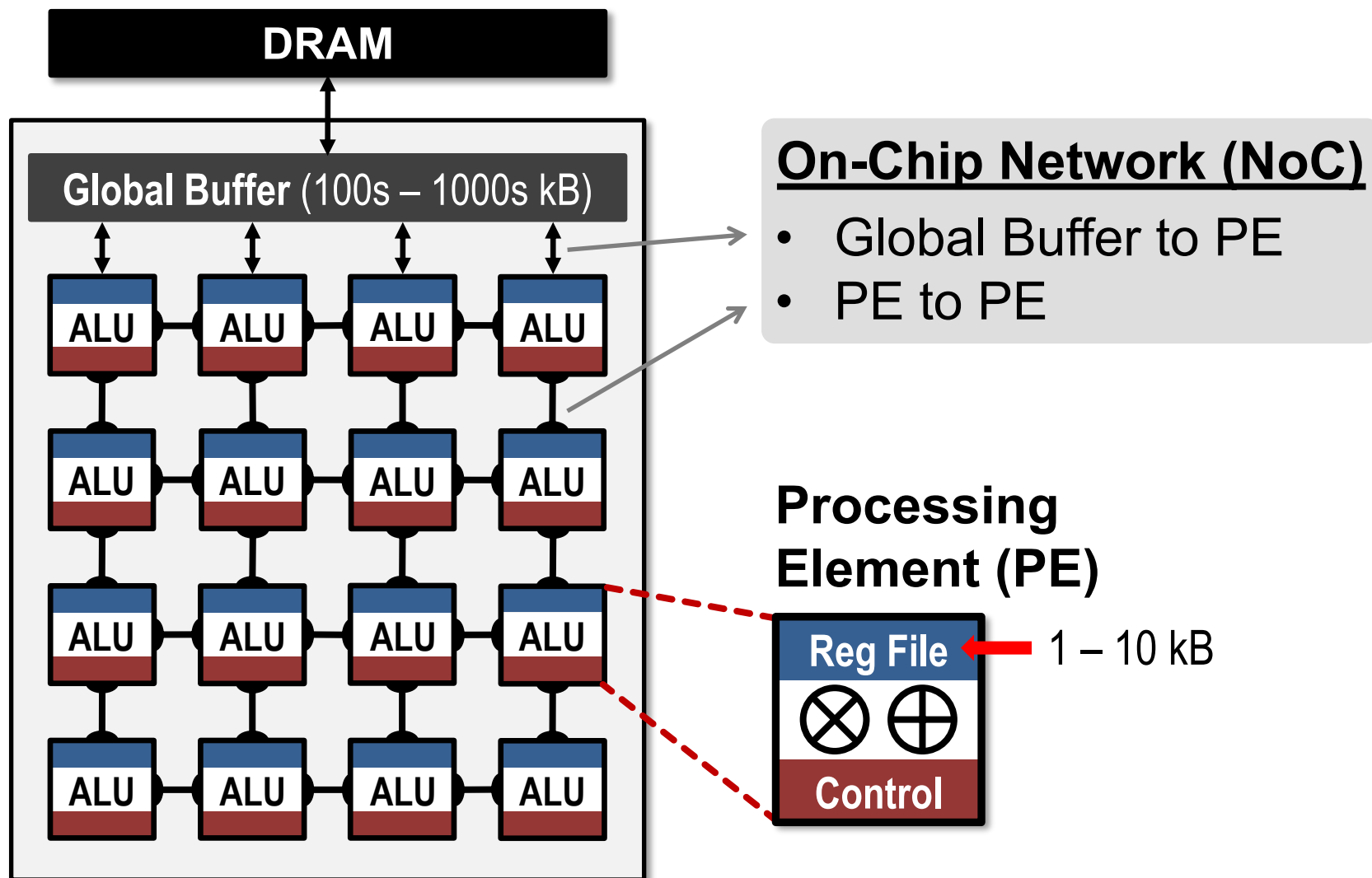
Leverage Parallelism for Higher Performance



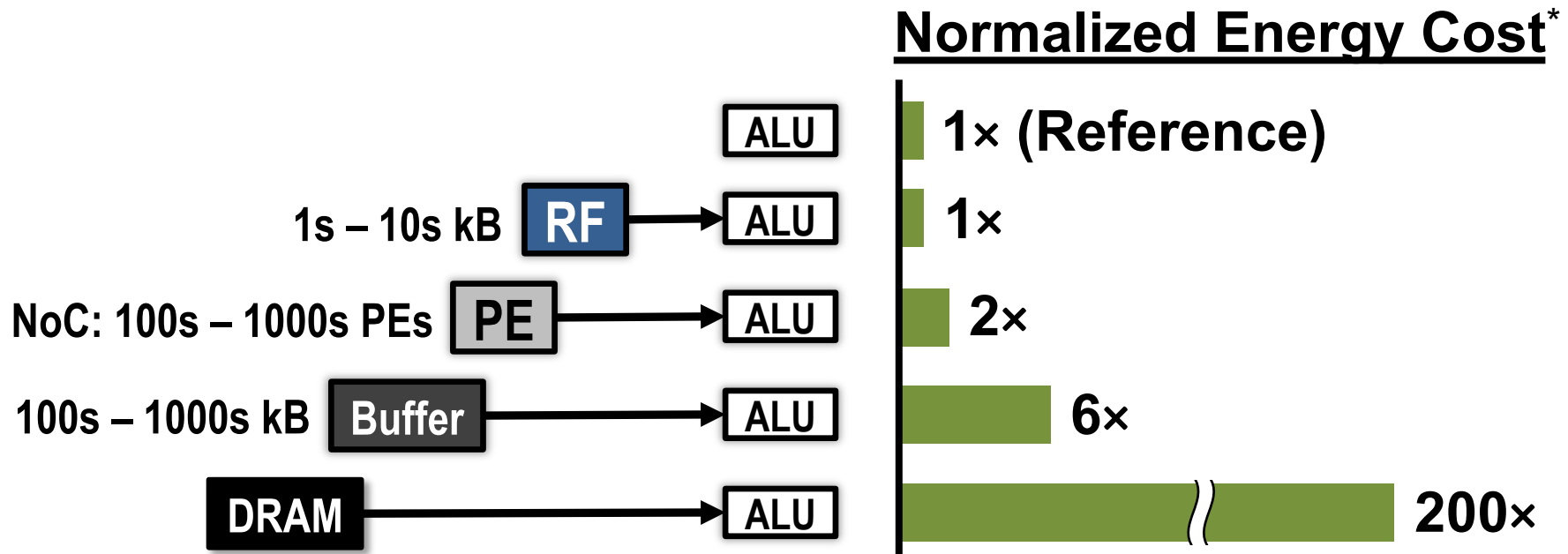
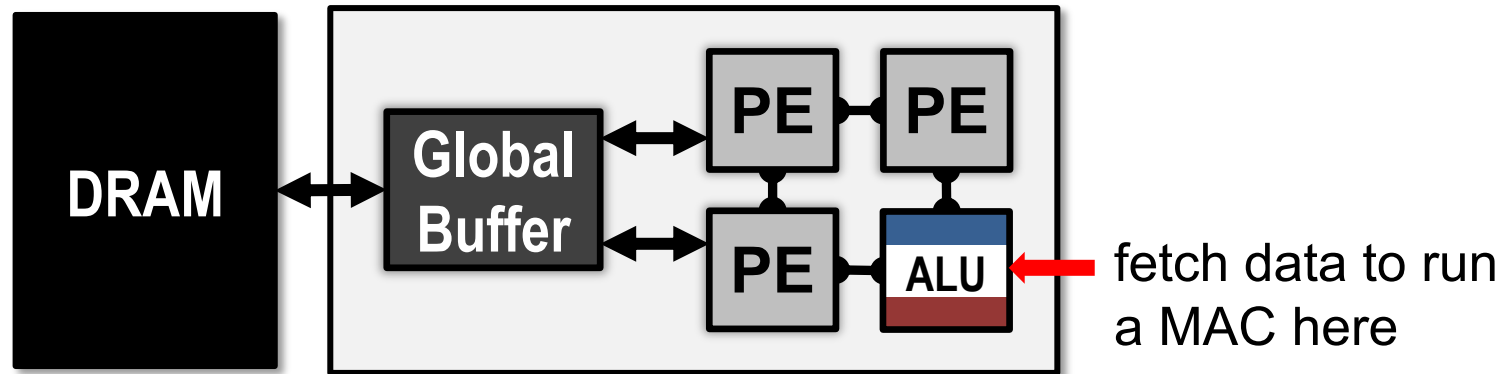
Leverage Parallelism for *Spatial* Data Reuse



Spatial Architecture for DNN

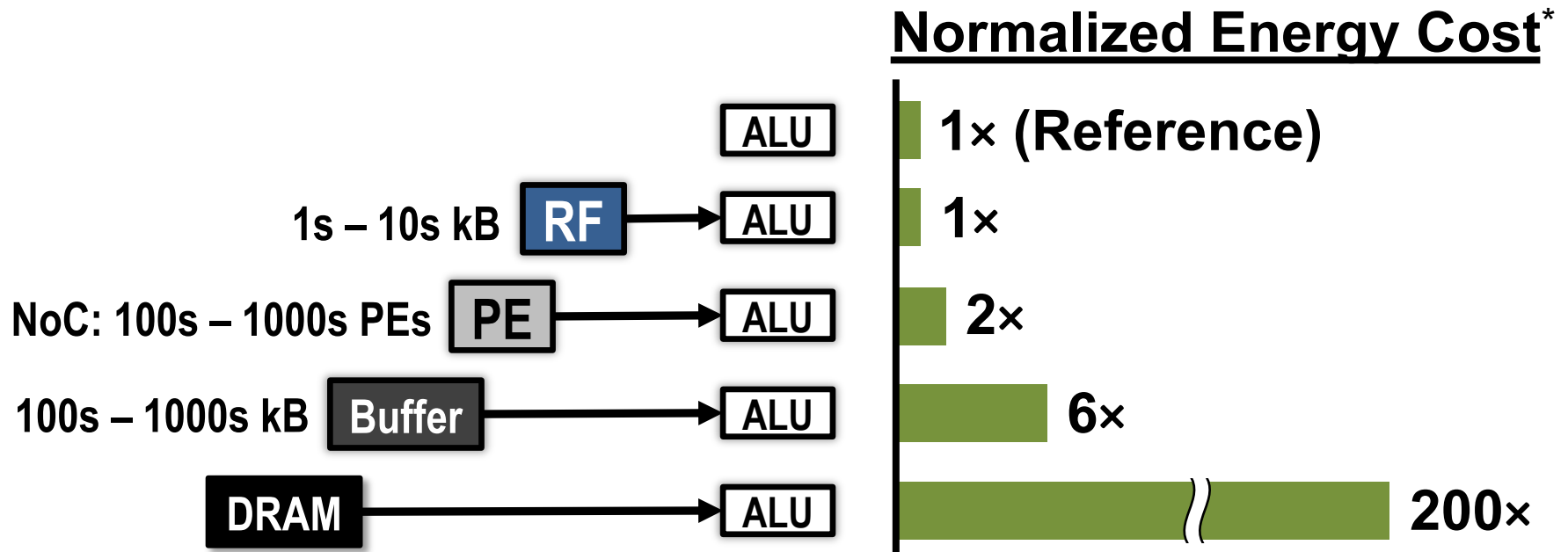


Multi-Level Low-Cost Data Access



Multi-Level Low-Cost Data Access

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

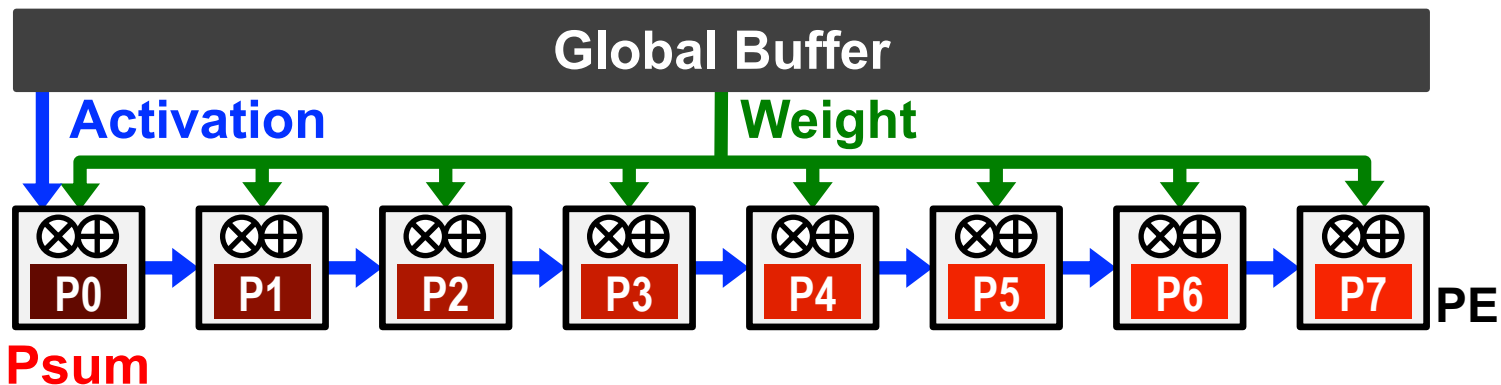


* measured from a commercial 65nm process

Dataflow Taxonomy

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

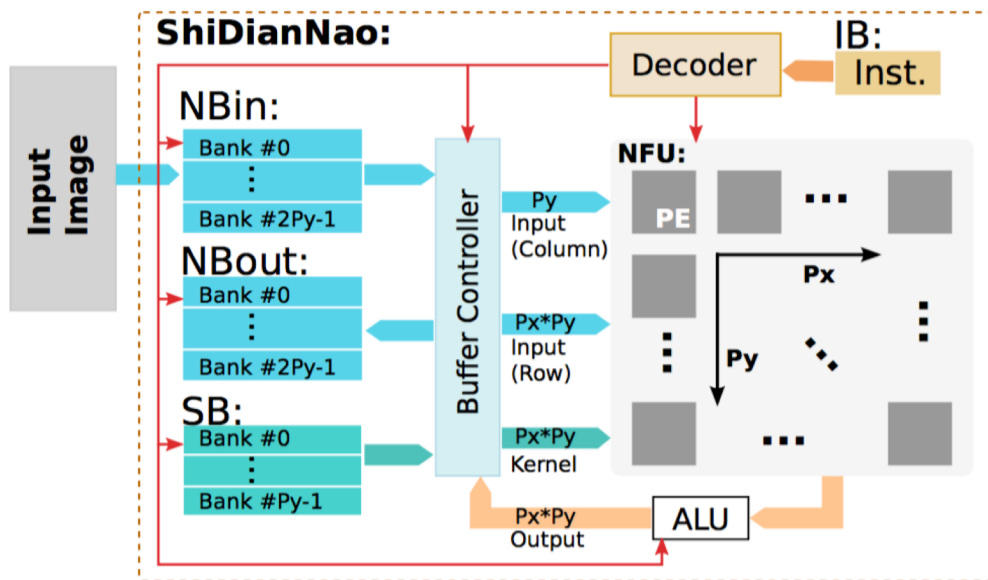
Output Stationary (OS)



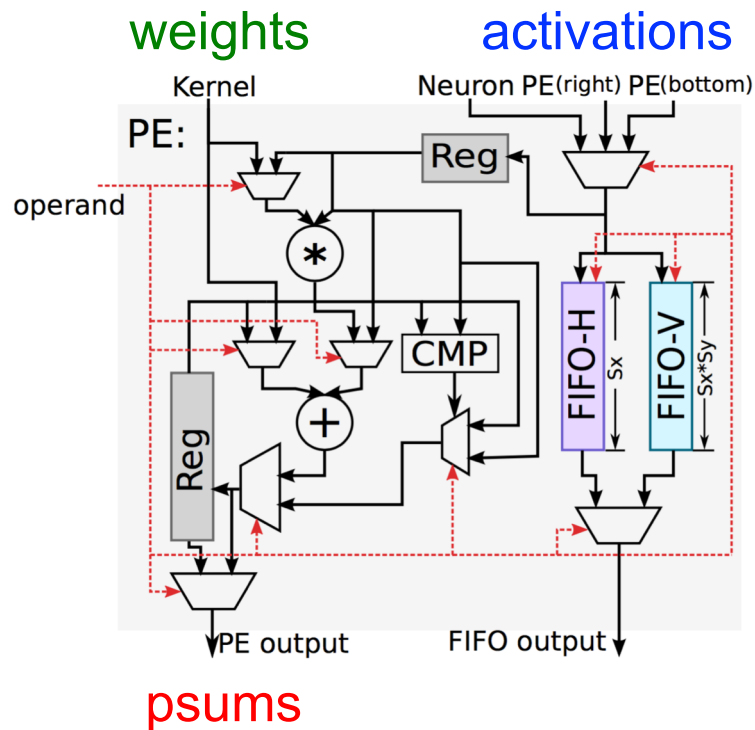
- Minimize **partial sum** R/W energy consumption
 - maximize local accumulation
- Broadcast/Multicast **filter weights** and reuse **activations** spatially across the PE array

OS Example: ShiDianNao

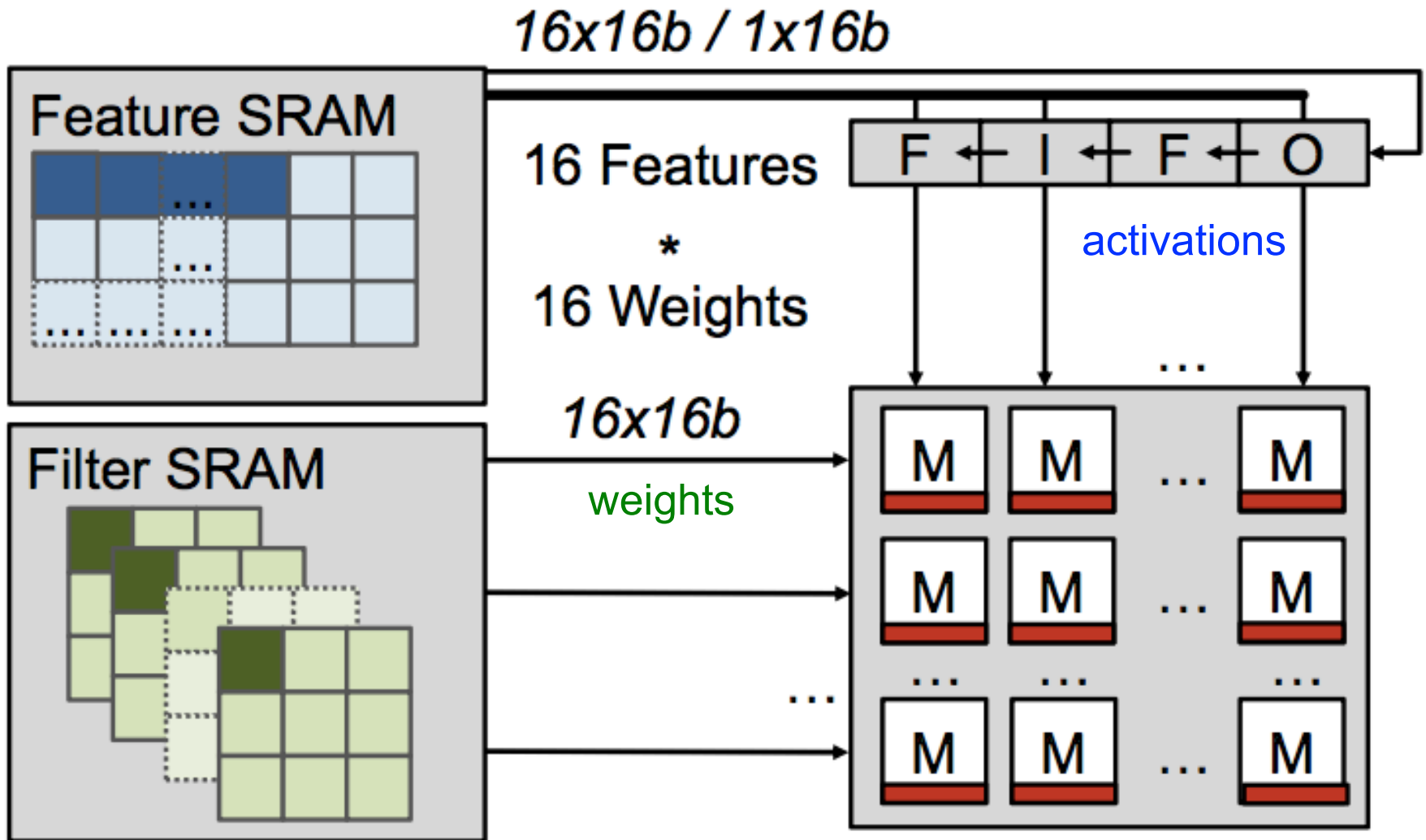
Top-Level Architecture



PE Architecture

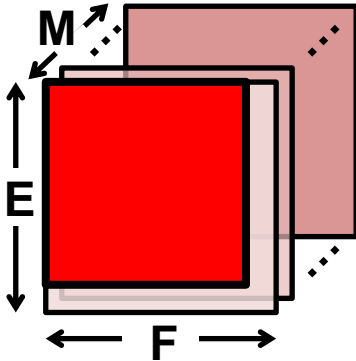
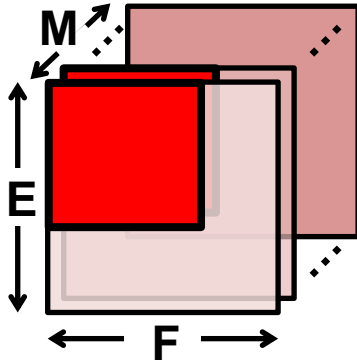
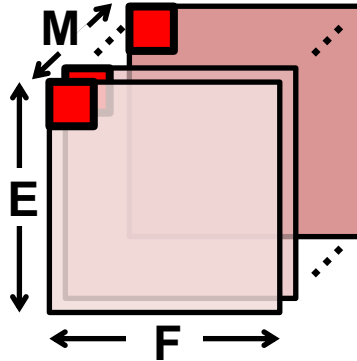


OS Example: ENVISION

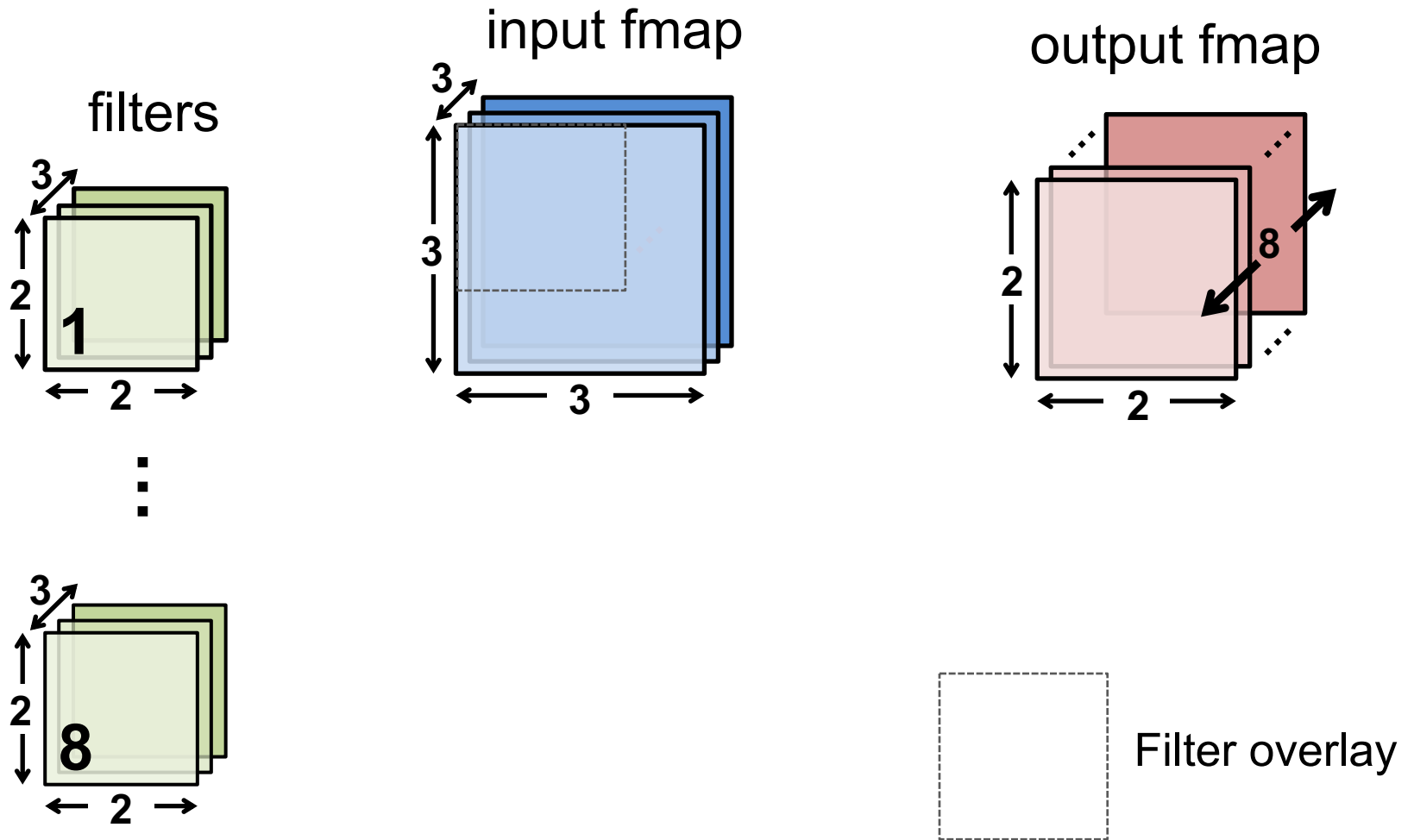


[Moons et al., VLSI 2016, ISSCC 2017]

Variants of Output Stationary

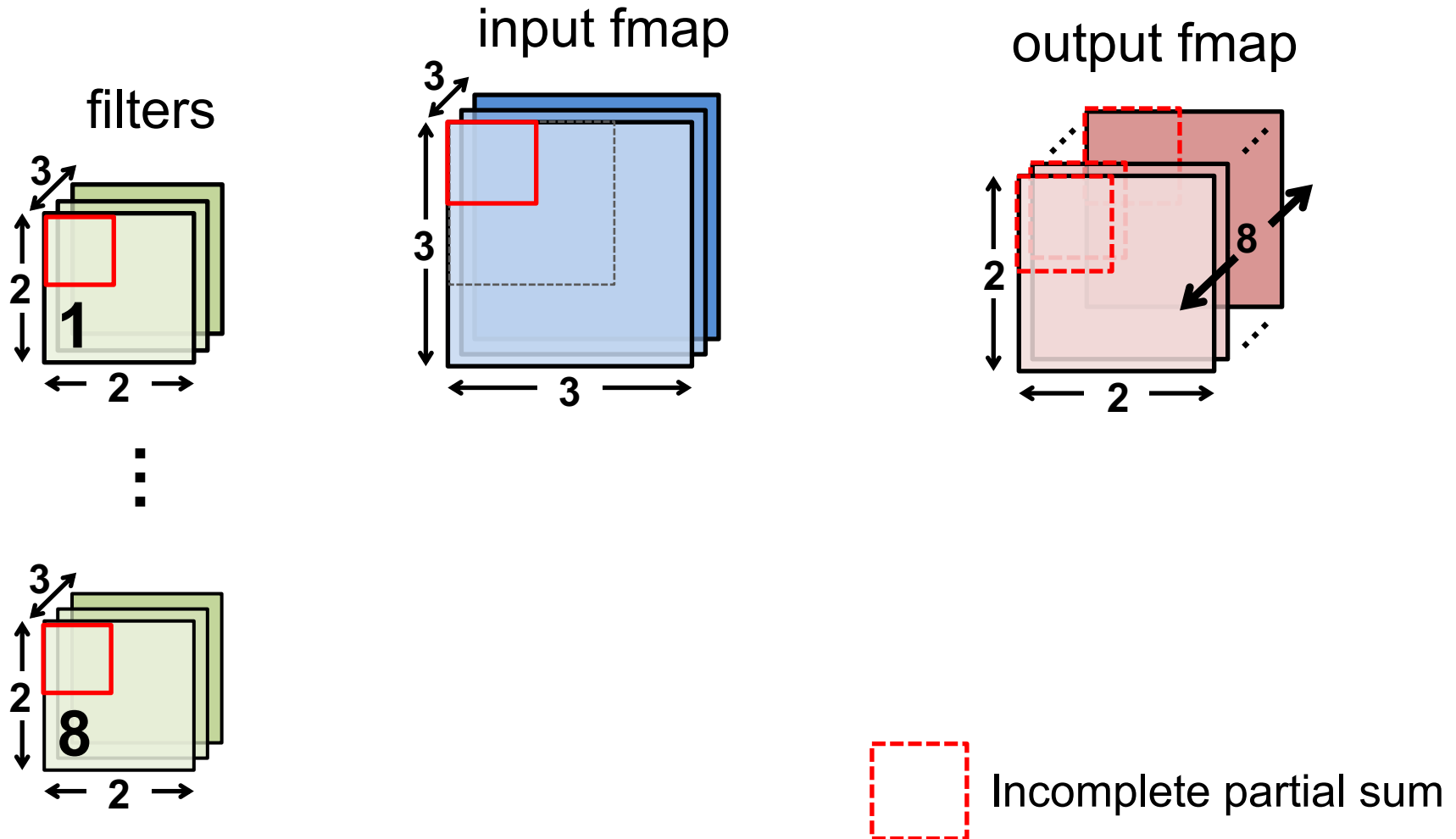
	OS_A	OS_B	OS_C
Parallel Output Region			
# Output Channels	Single	Multiple	Multiple
# Output Activations	Multiple	Multiple	Single
Notes	Targeting CONV layers		Targeting FC layers

OS Example



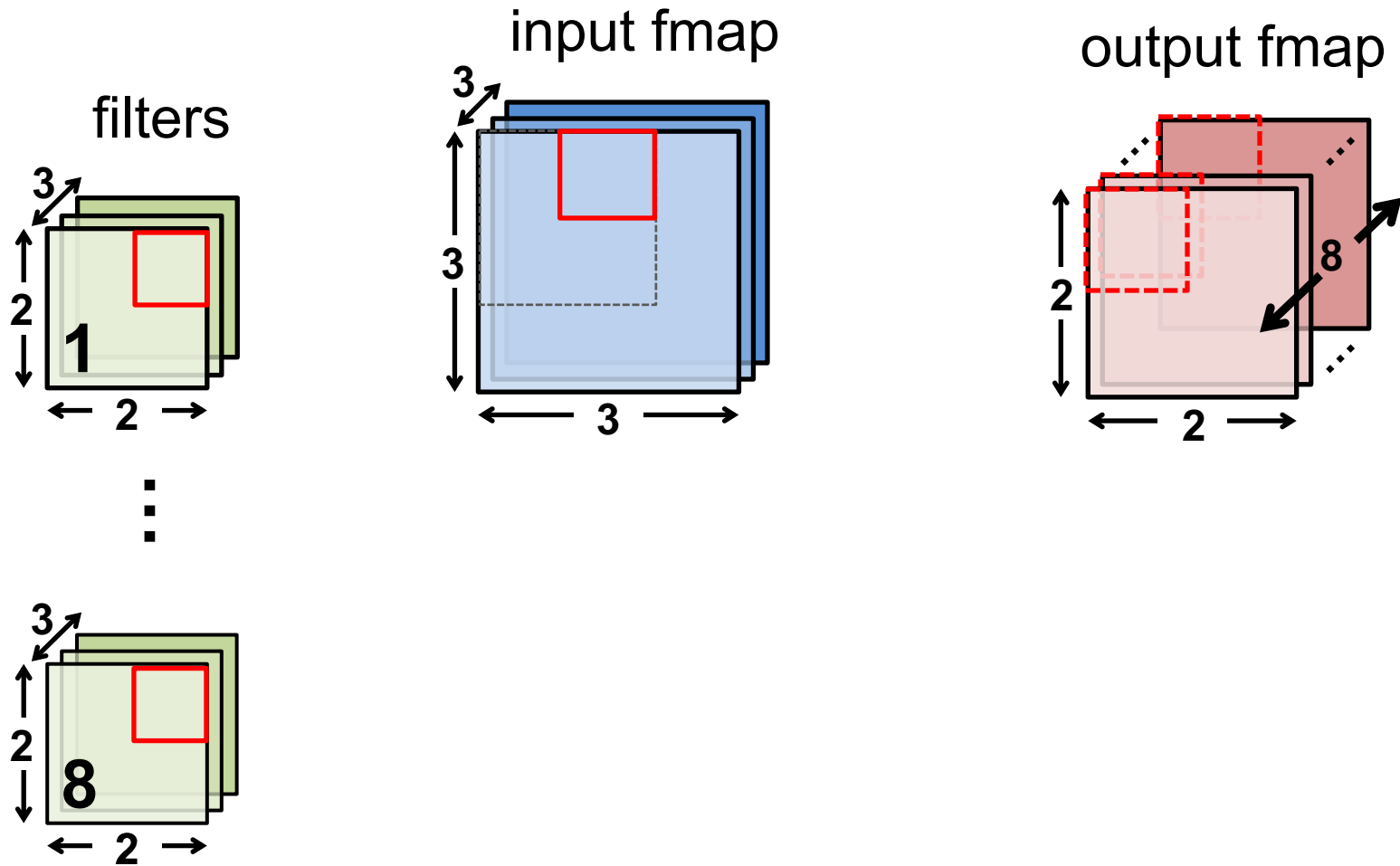
OS Example

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



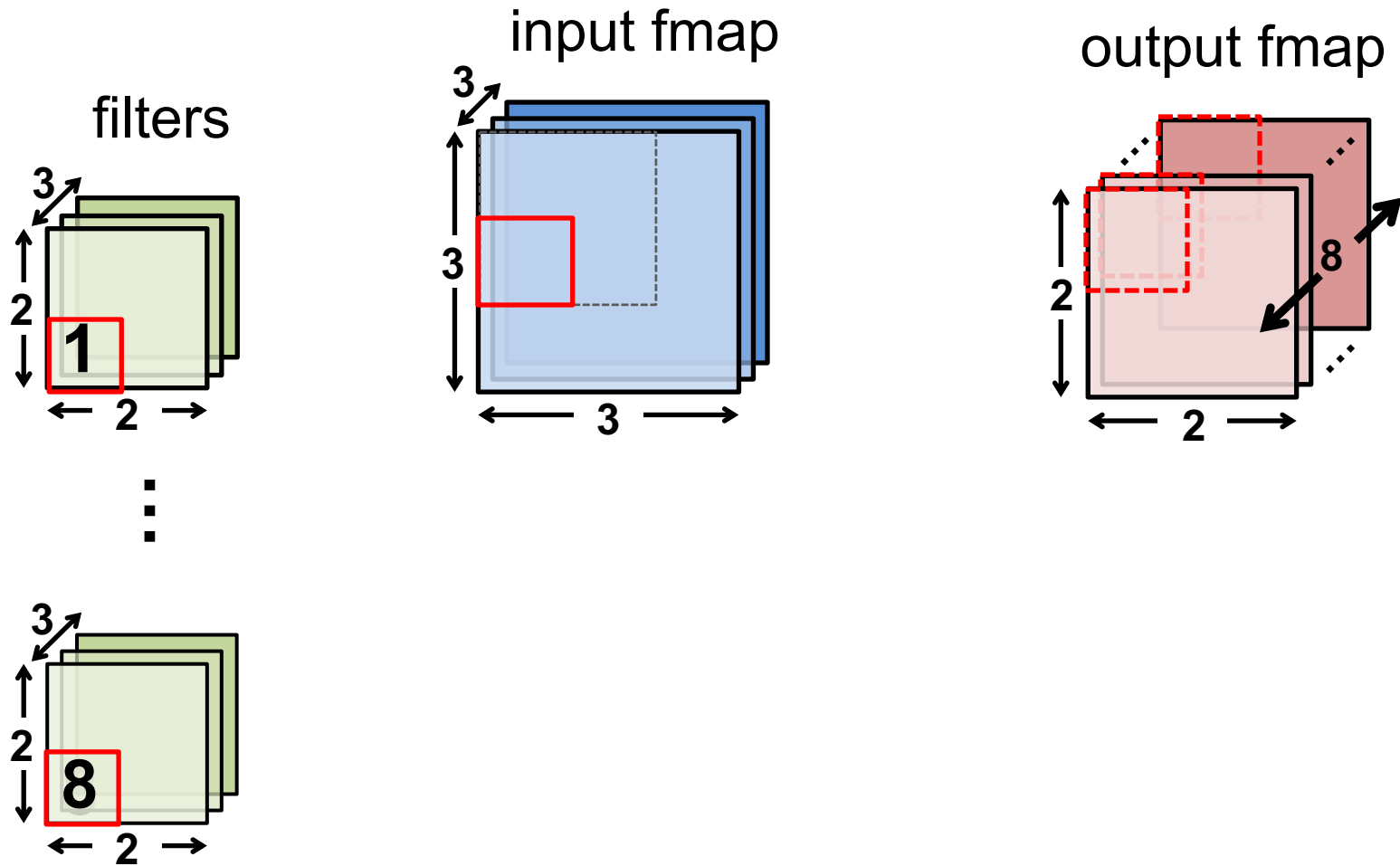
OS Example

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



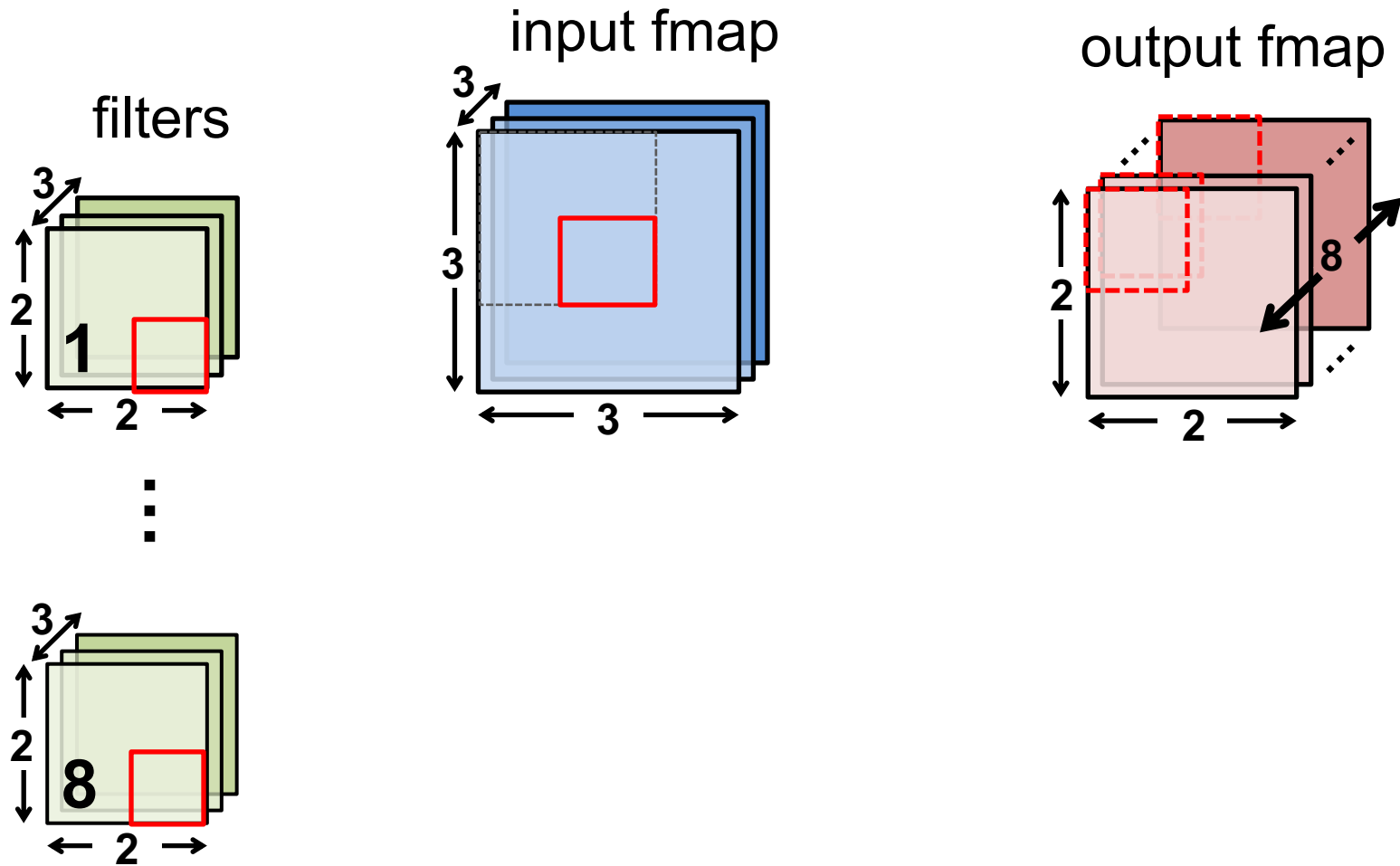
OS Example

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



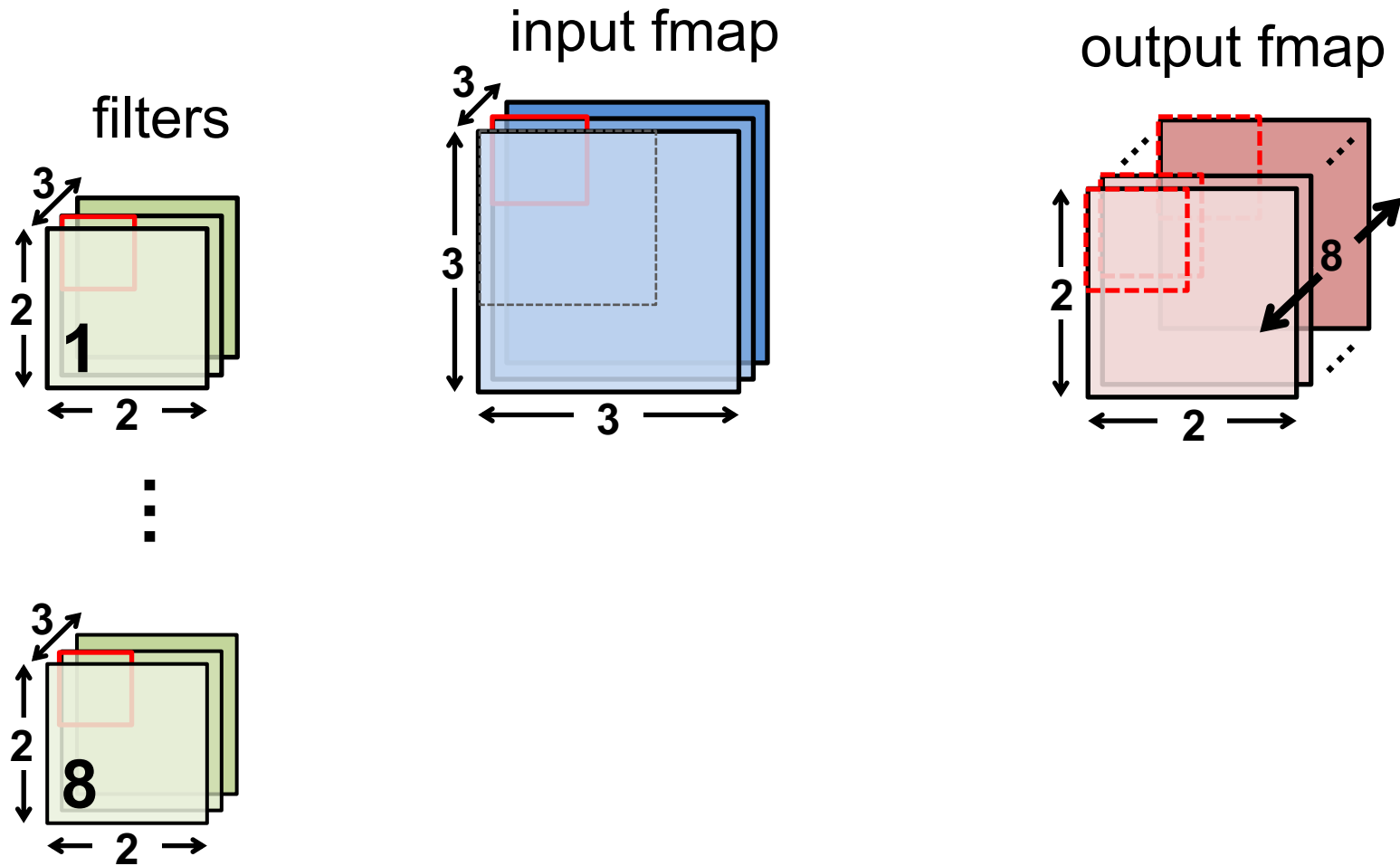
OS Example

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



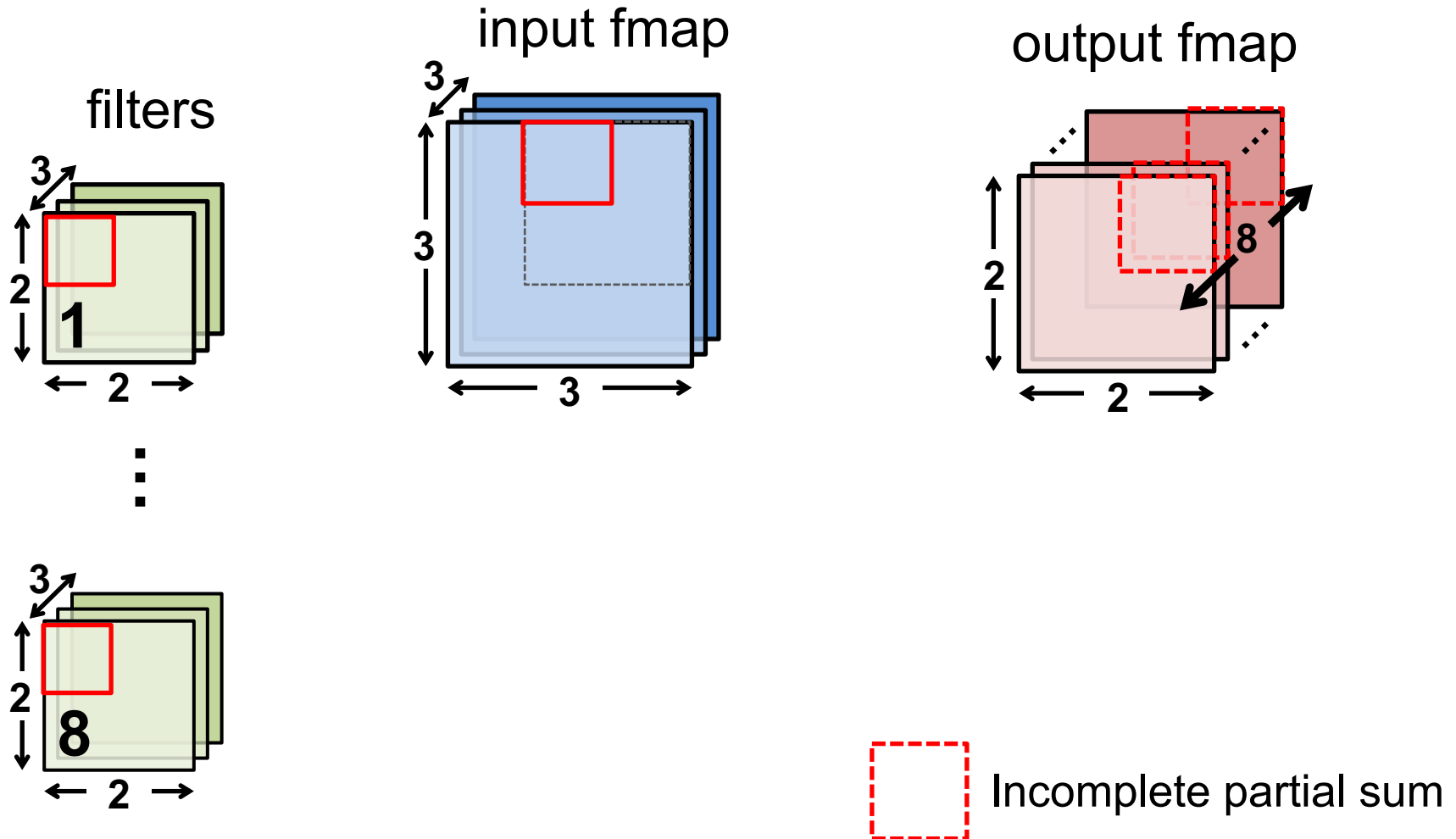
OS Example

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

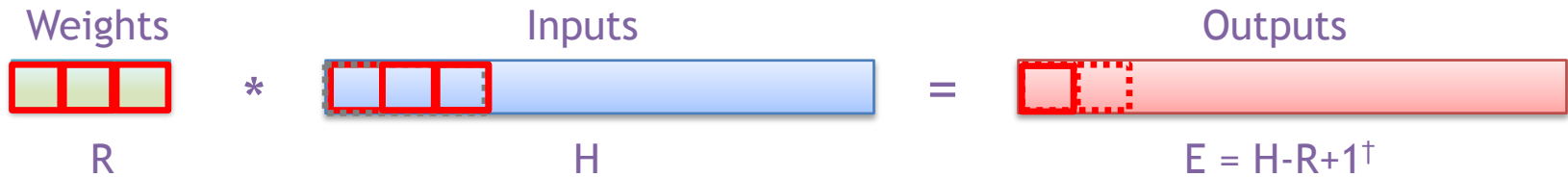


OS Example

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



1-D Convolution – Output Stationary



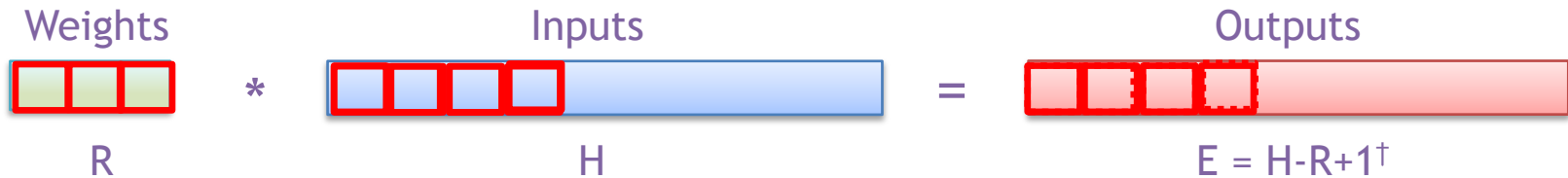
```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations
```

```
for (e = 0; e < E; e++)
  for (r = 0; r < R; r++)
    O[e] += I[e+r] * W[r];
```

No constraints
on loop
permutations!

[†] Assuming: ‘valid’ style convolution

1-D Convolution

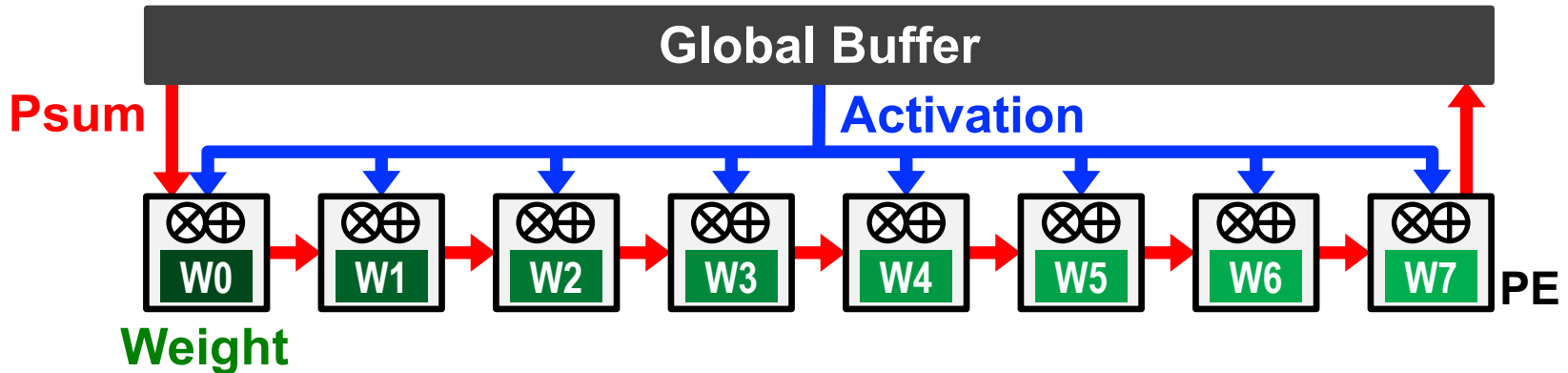


```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations

for (r = 0; r < R; r++)
    for (e = 0; e < E; e++)
        O[e] += I[e+r] * W[r];
```

[†] Assuming: 'valid' style convolution

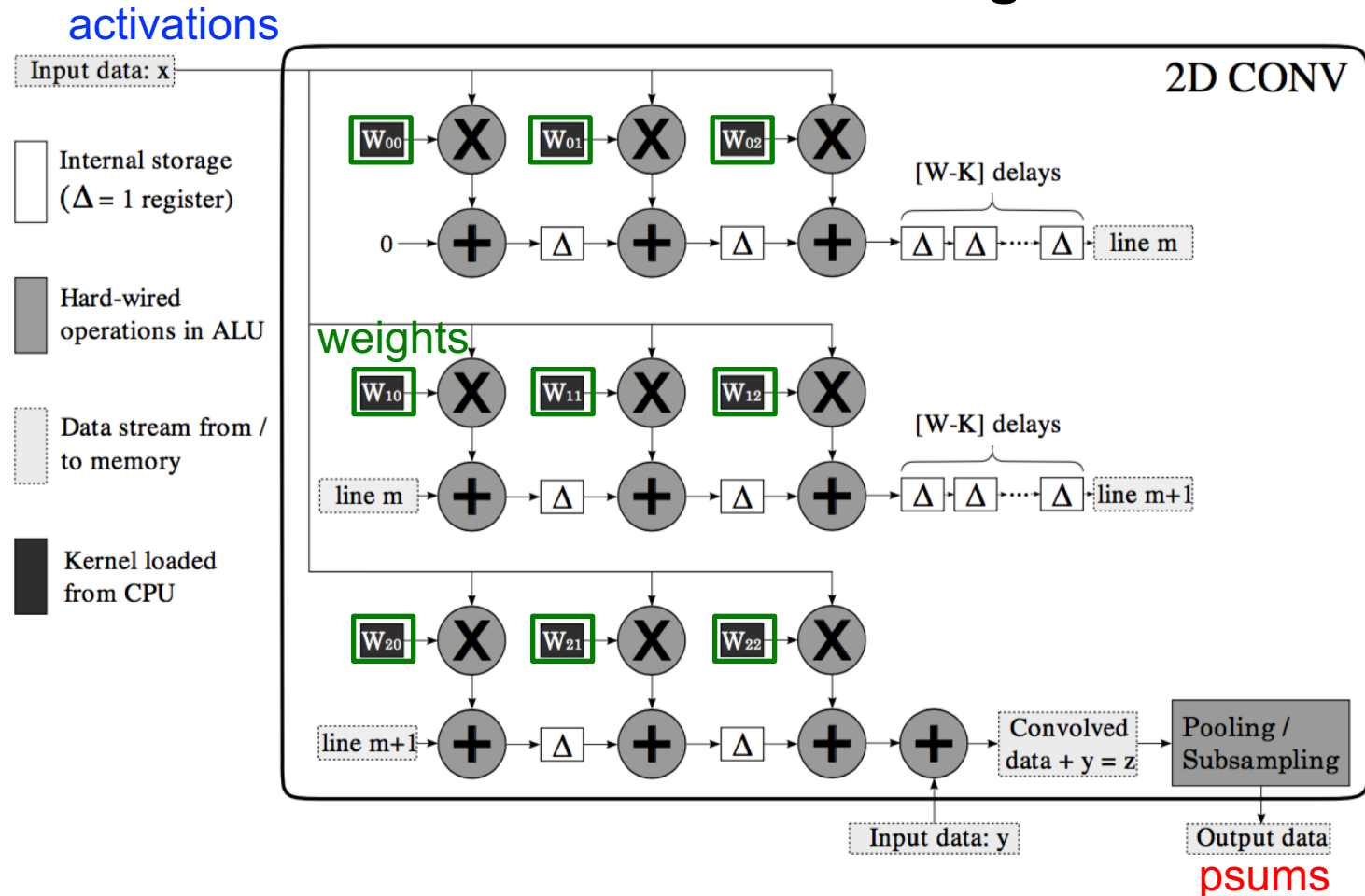
Weight Stationary (WS)



- **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



WS Example: NVDLA (simplified)

Released Sept 29, 2017

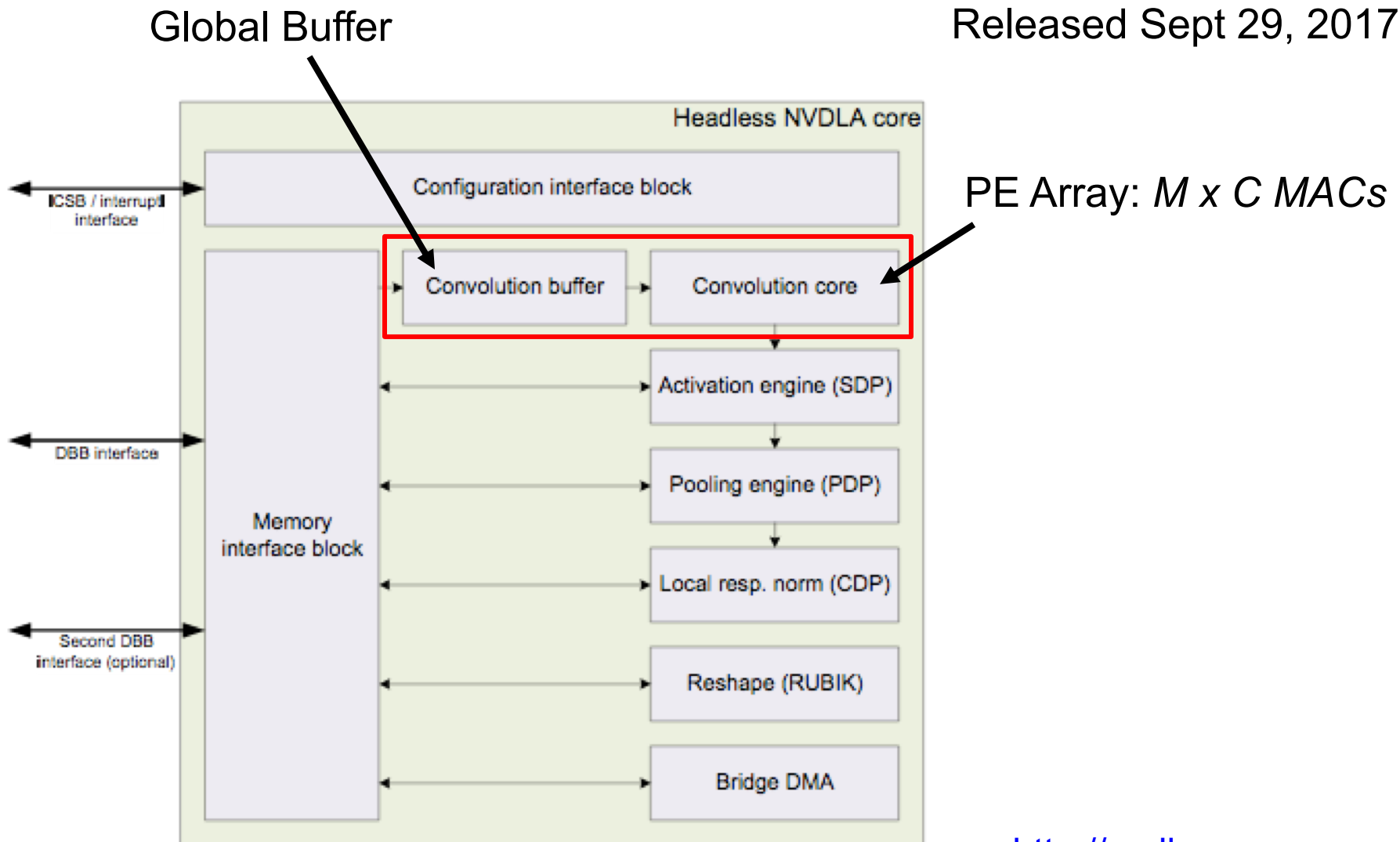
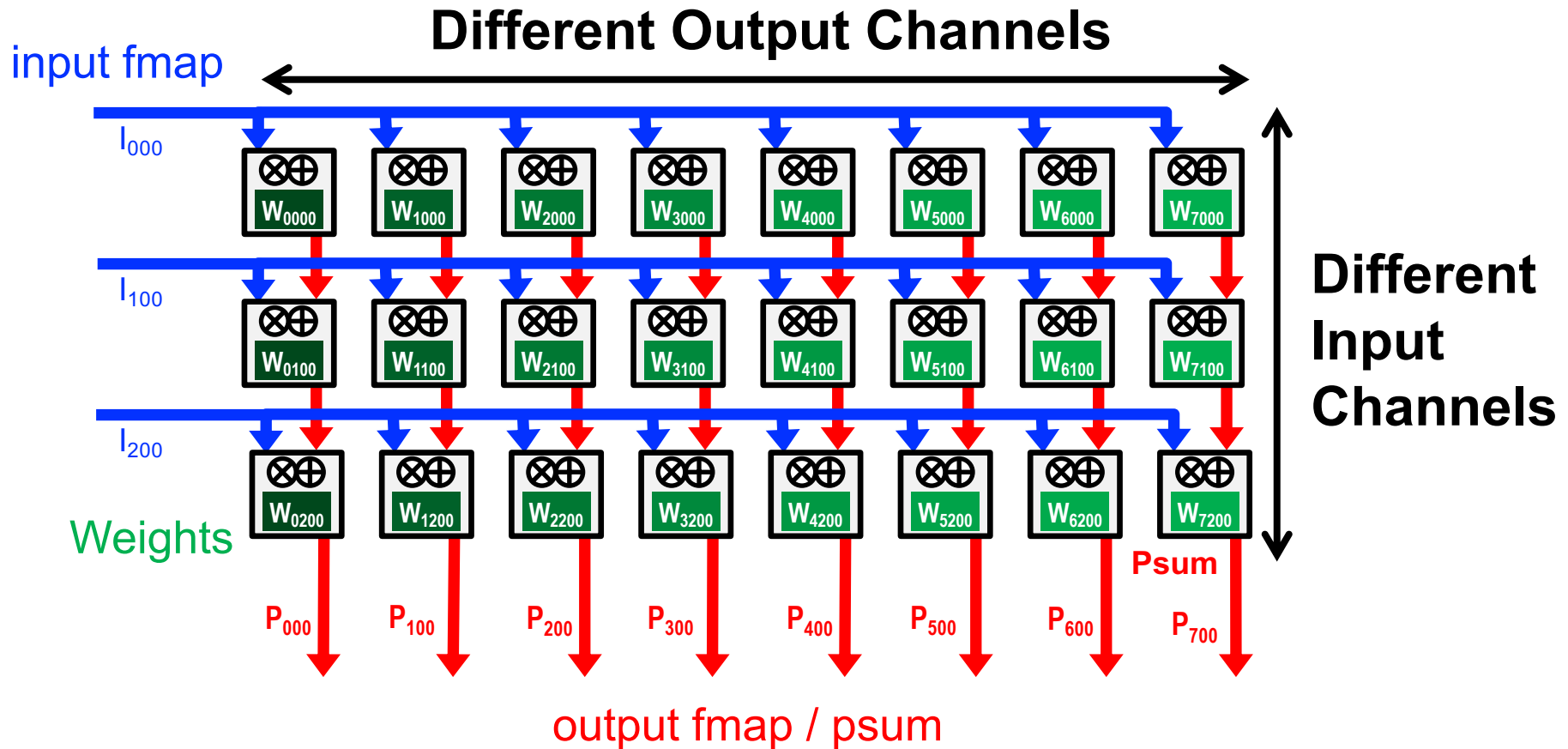


Image Source: Nvidia

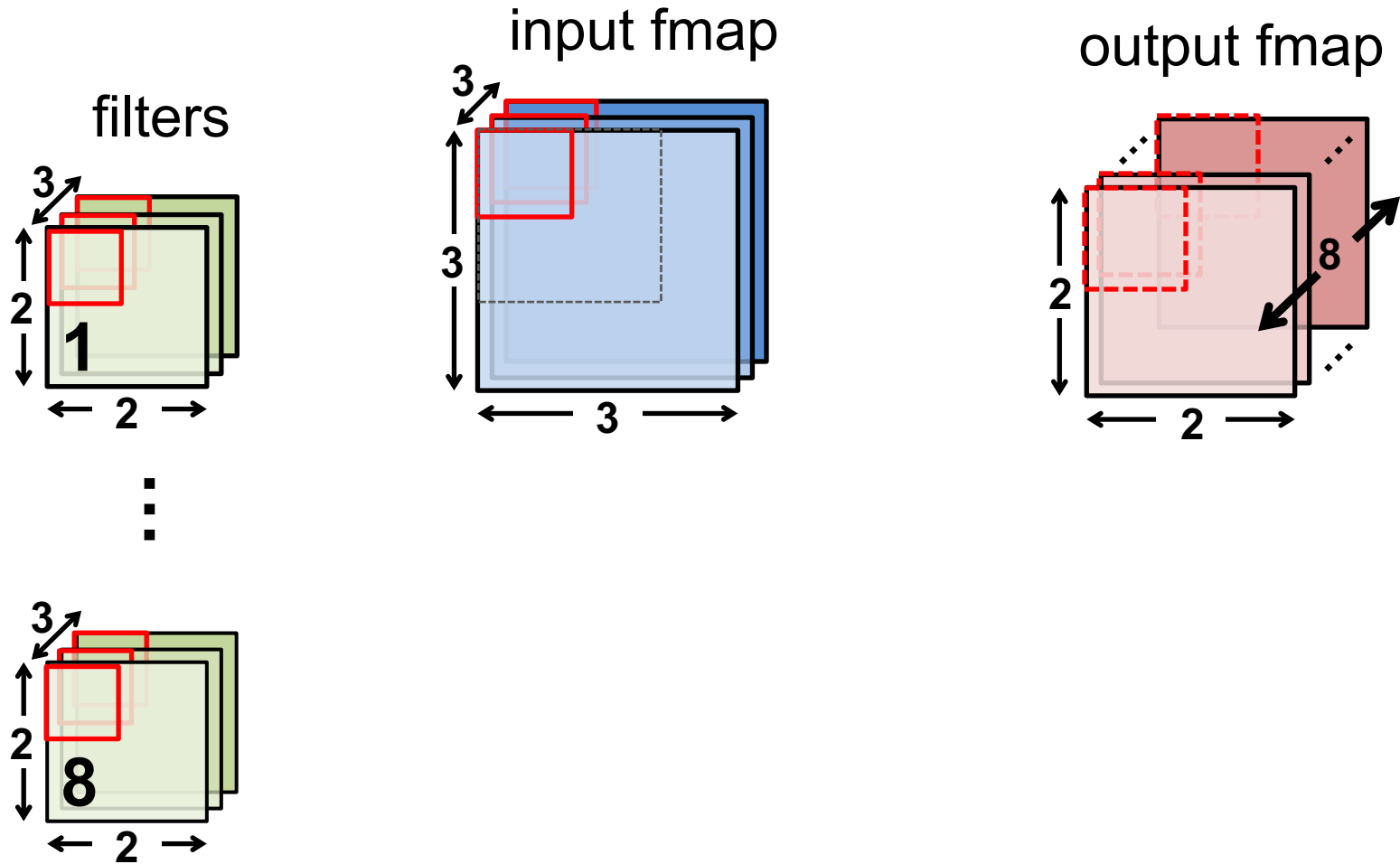
<http://nvdla.org>

WS Example: NVDLA (simplified)



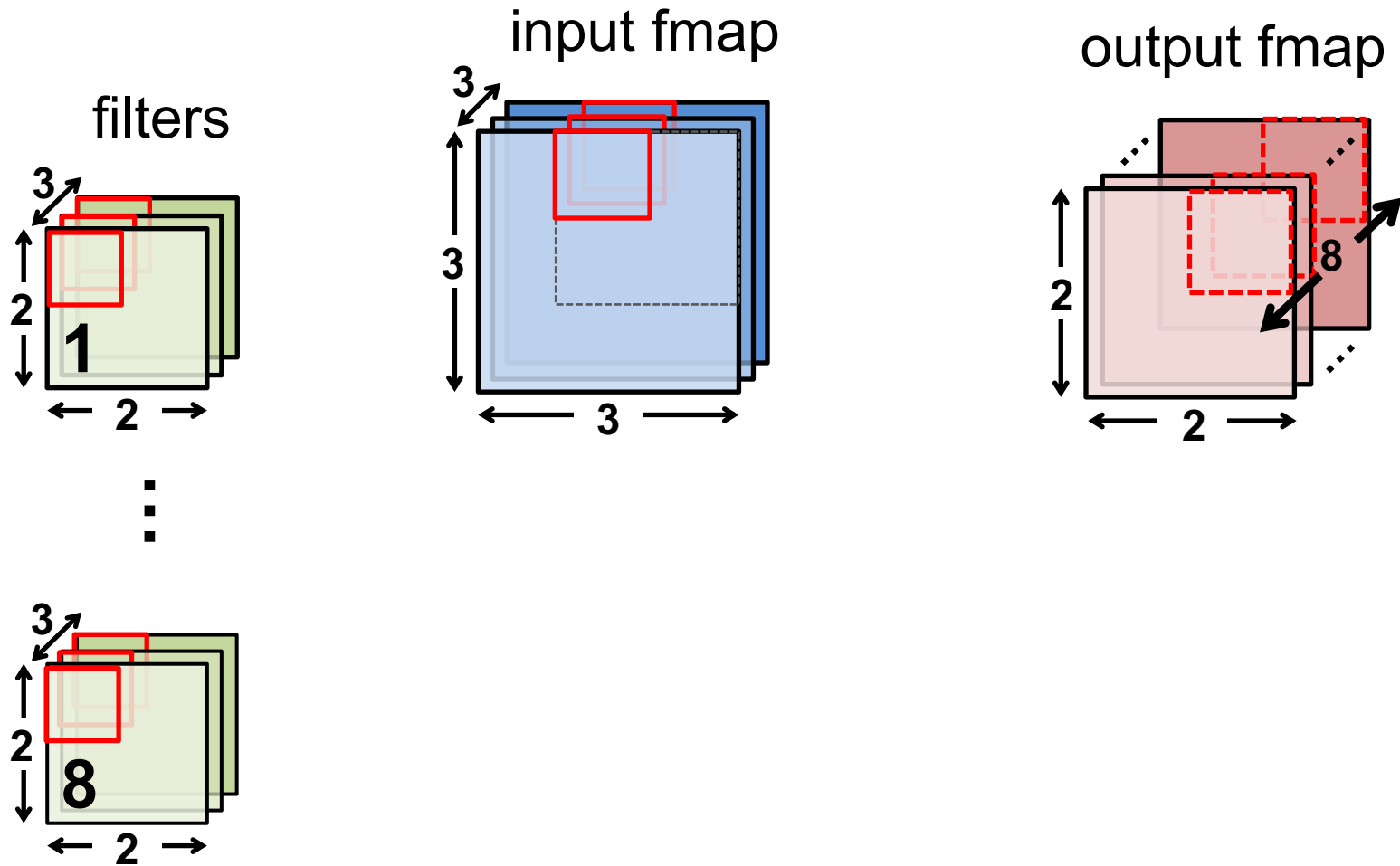
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



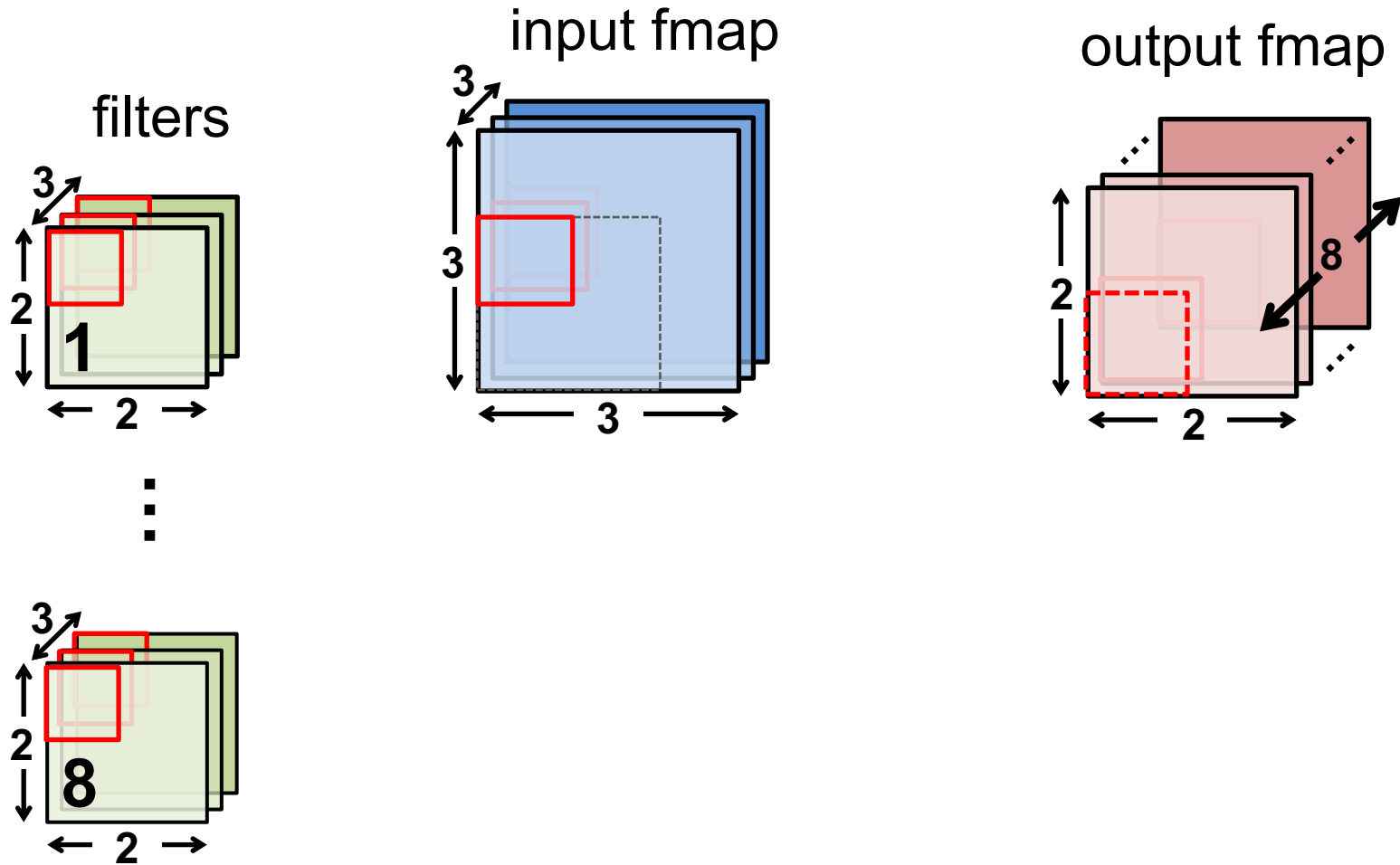
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



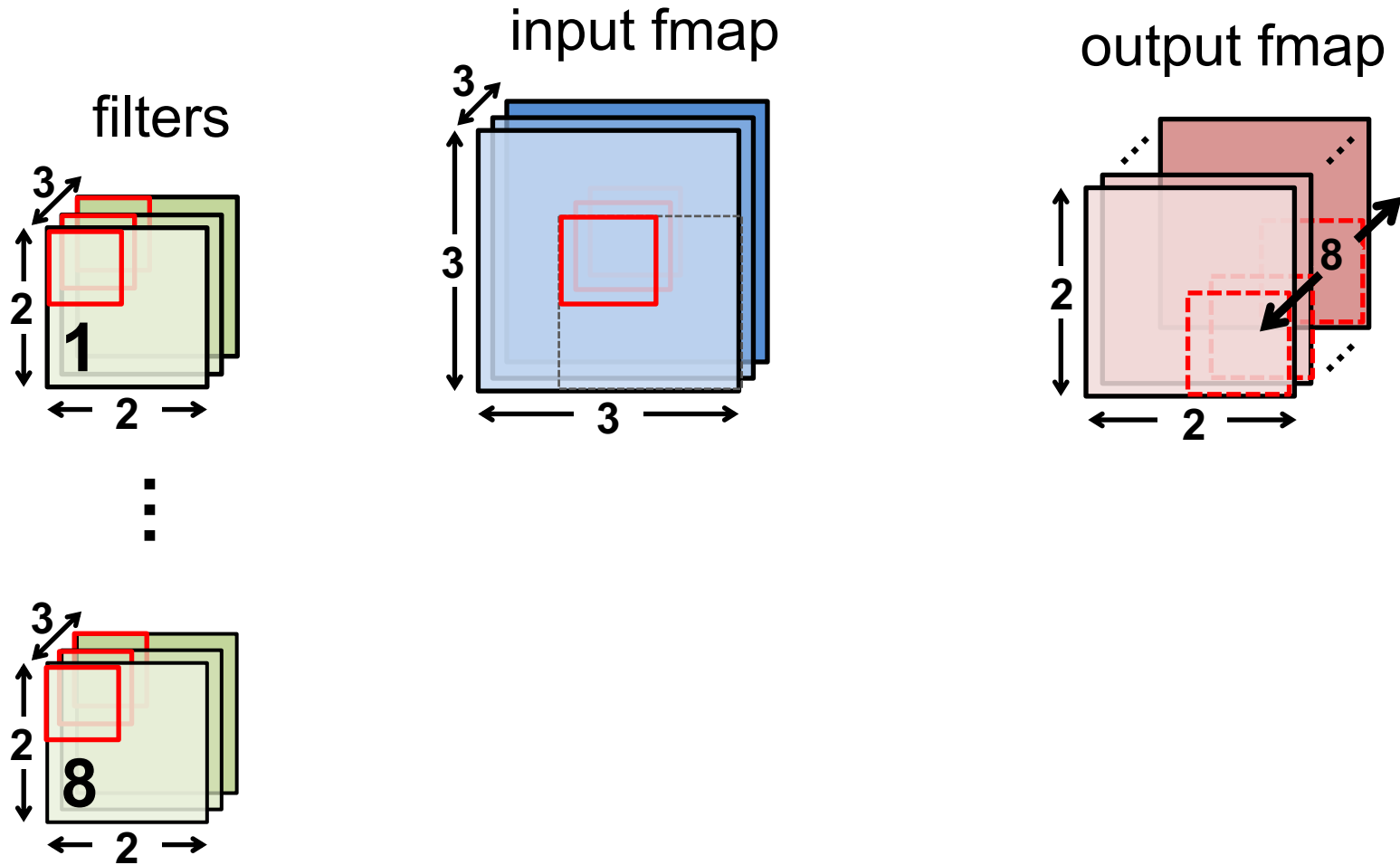
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



WS Example: NVDLA (simplified)

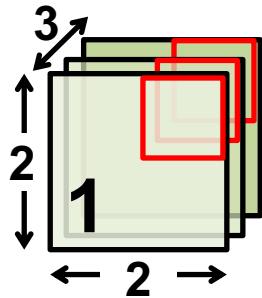
Cycle through input and output fmap (hold weights)



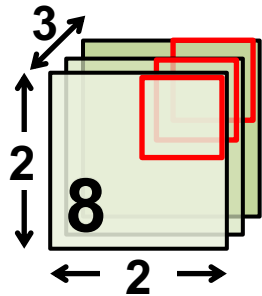
WS Example: NVDLA (simplified)

Load new weights

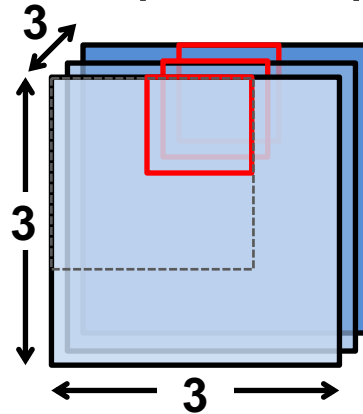
filters



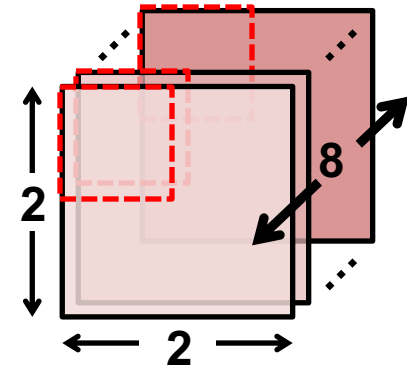
⋮



input fmap

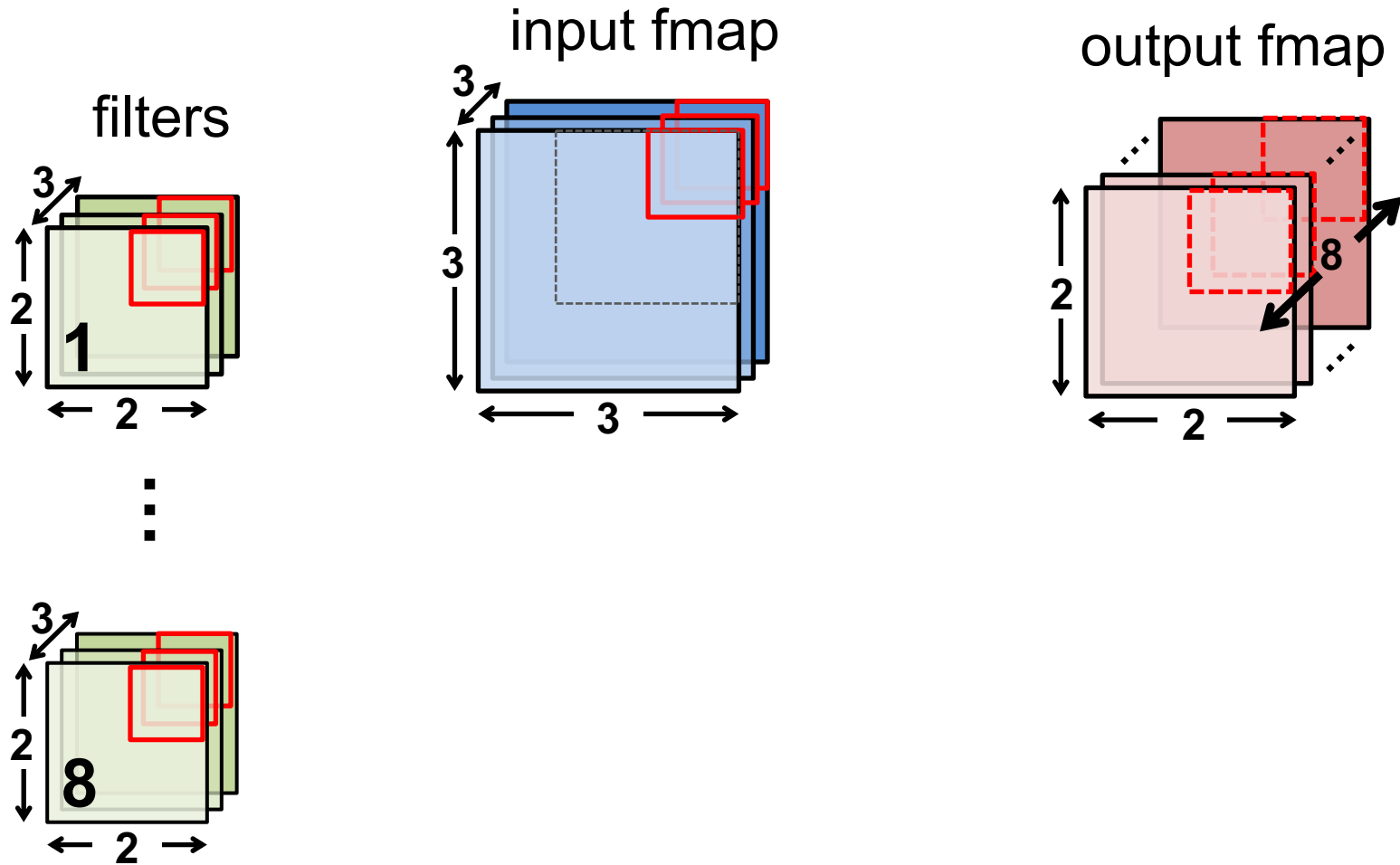


output fmap



WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



Taxonomy: More Examples

- **Output Stationary (OS)**

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

[Gupta, *ICML* 2015] [Moons, *VLSI* 2016] [Thinker, *VLSI* 2017]

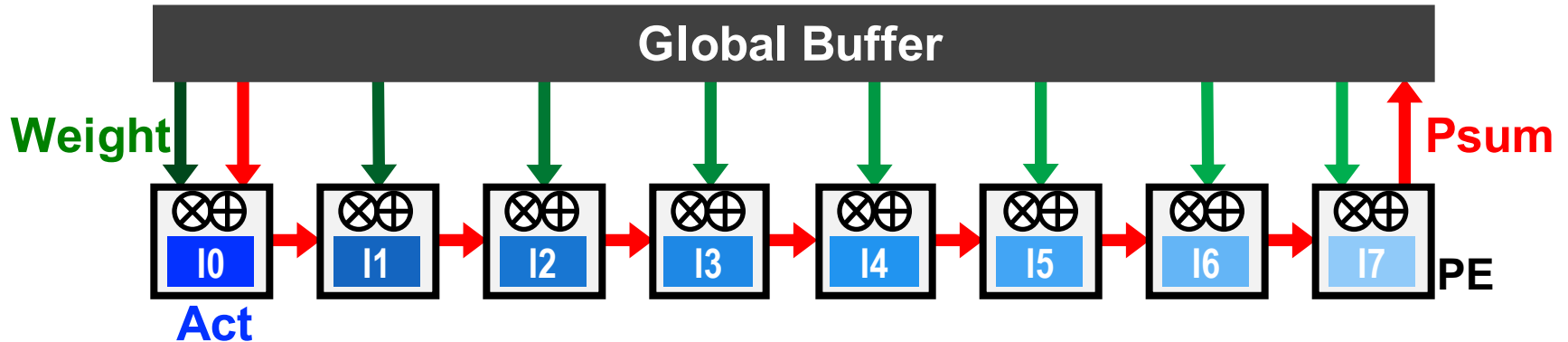
- **Weight Stationary (WS)**

[Chakradhar, *ISCA* 2010] [nn-X (NeuFlow), *CVPRW* 2014]

[Park, *ISSCC* 2015] [ISAAC, *ISCA* 2016] [PRIME, *ISCA* 2016]

[TPU, *ISCA* 2017]

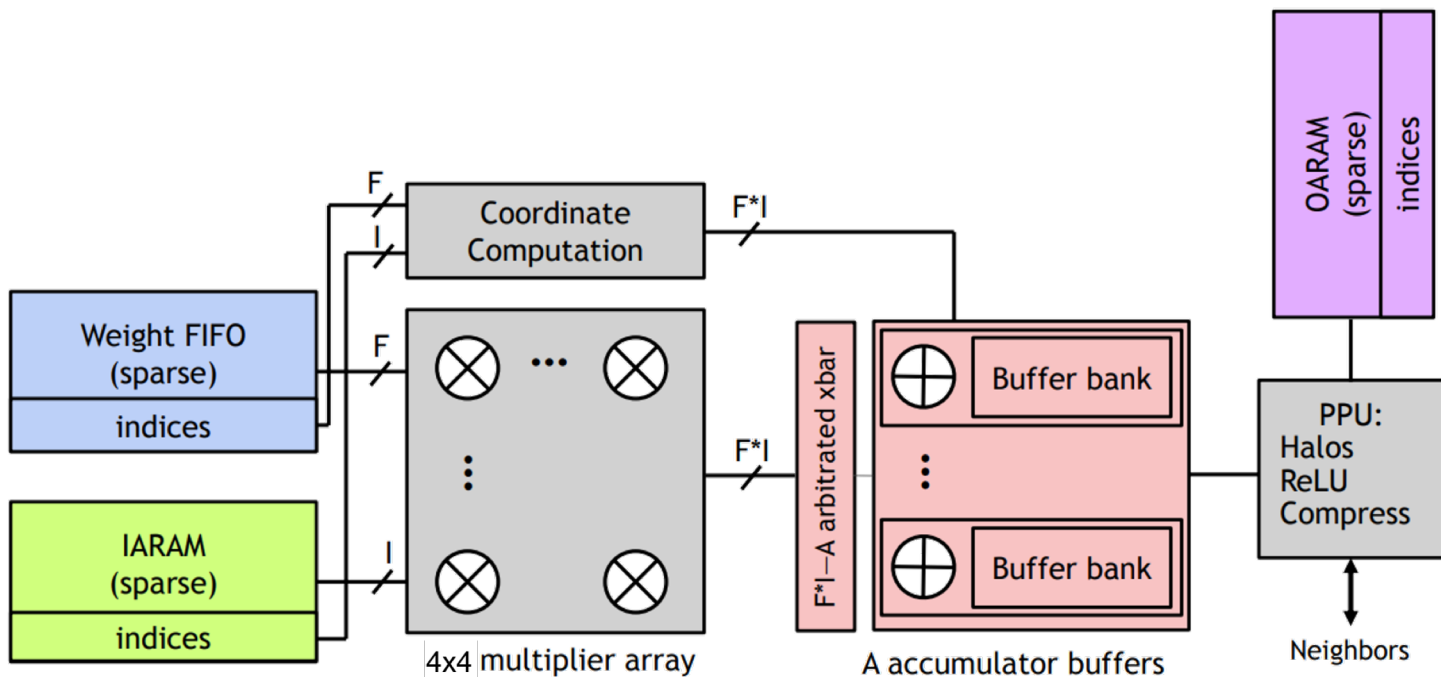
Input Stationary (IS)



- **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



1-D Convolution – Weight Stationary



```
int I[H];    // Input activations
int W[R];    // Filter weights
int O[E];    // Output activations

for (e = 0; e < E; e++)
    for (r = 0; r < R; r++)
        O[e] += I[e+r] * W[r];
```

How can we implement input stationary with no input index?

[†] Assuming: 'valid' style convolution

1-D Convolution – Input Stationary



```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations

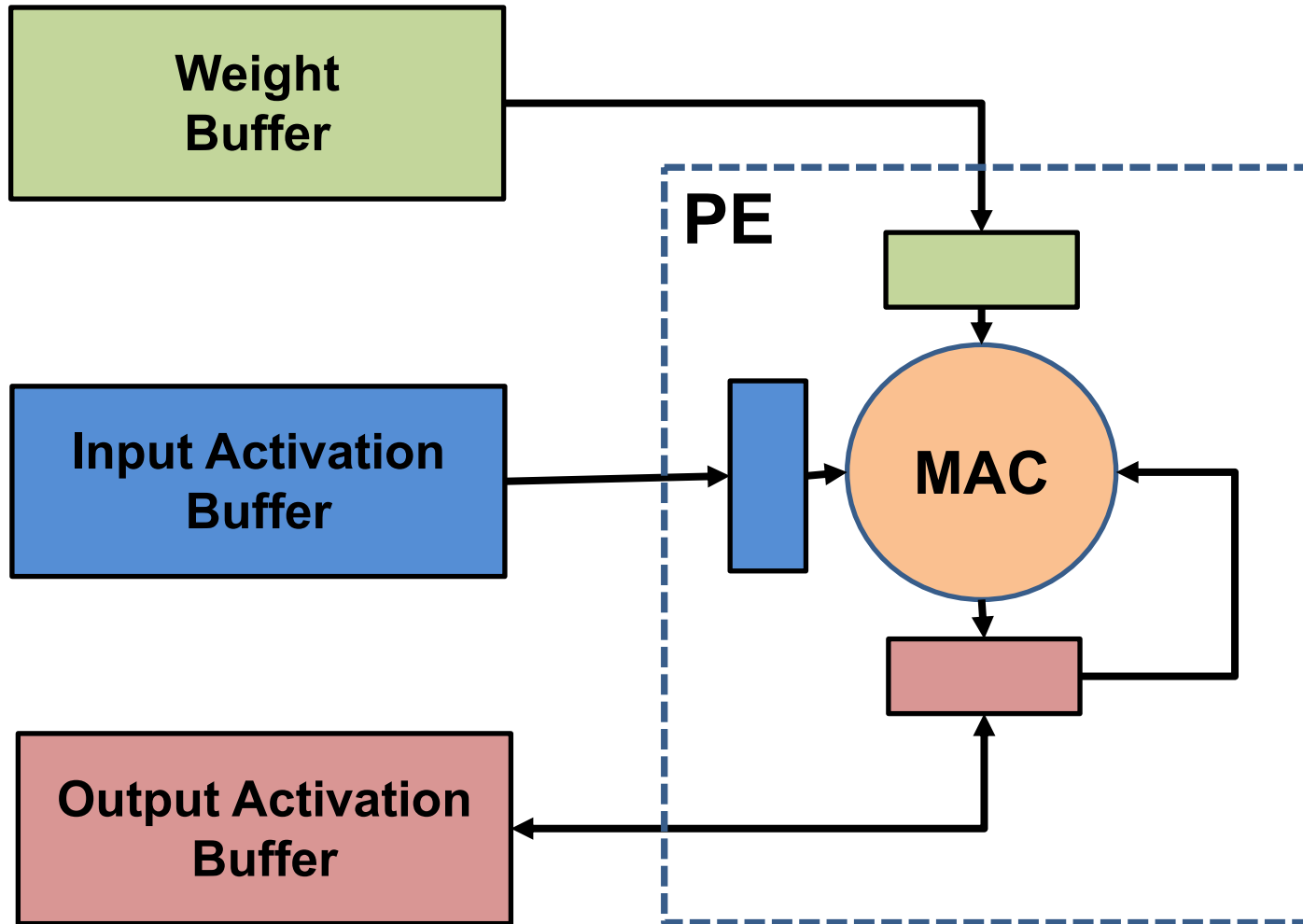
for (h = 0; h < H; h++)
    for (r = 0; r < R; r++)
        O[h-r] += I[h] * W[r];
```

Beware w-r
must be ≥ 0
and $< E$

[†] Assuming: ‘valid’ style convolution

Reference Patterns of Different Dataflows

Single PE Setup



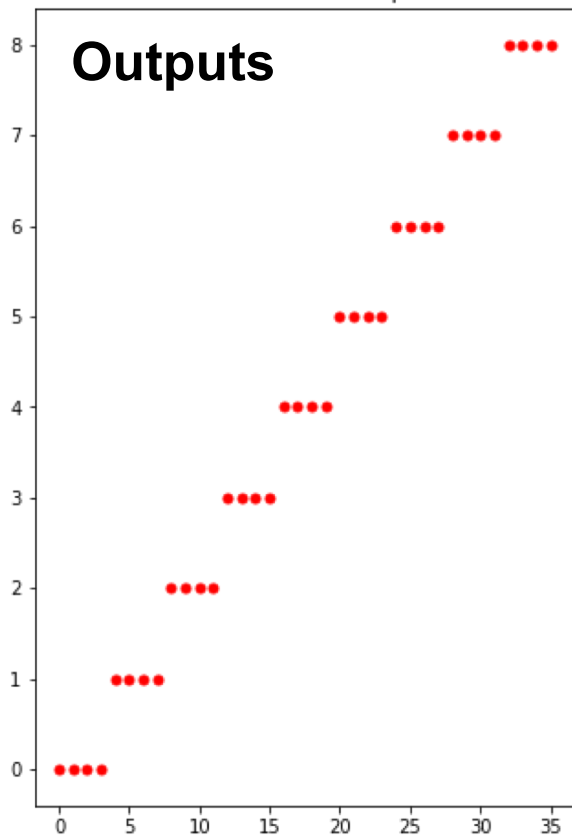
Output Stationary – Reference Pattern

```
for (e = 0; e < E; e++)  
  for (r = 0; r < R; r++)  
    O[e] += I[e+r] * W[r];
```

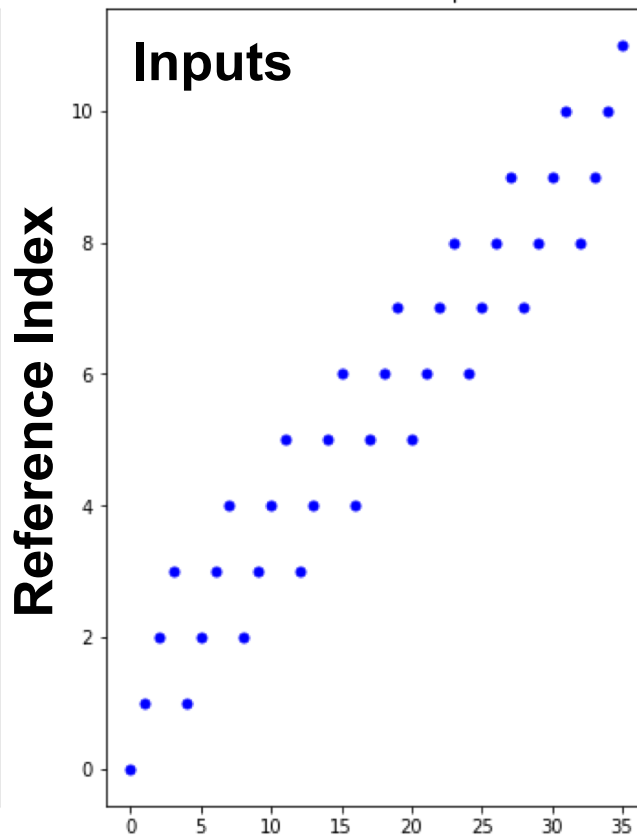
Layer Shape:

- H = 12
- R = 4
- E = 9

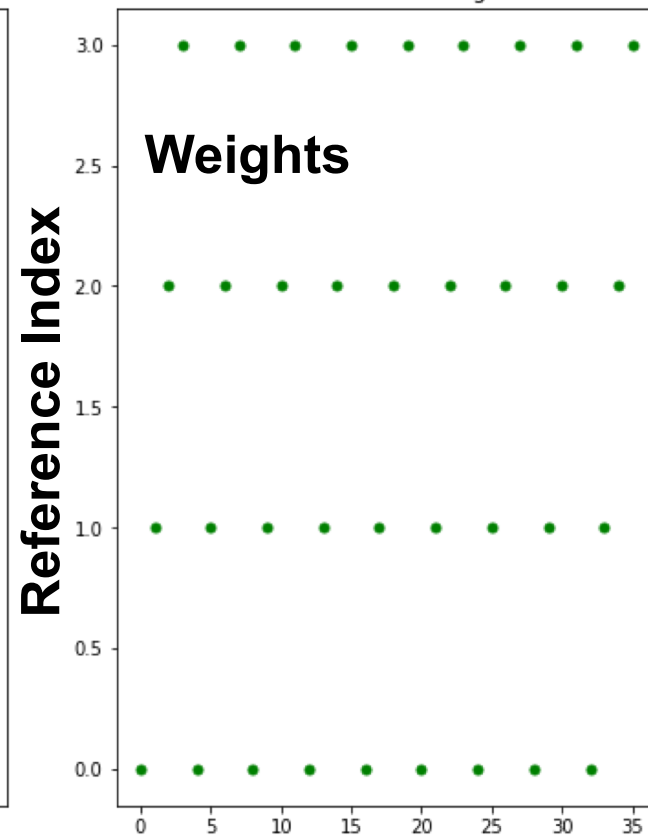
References for Output



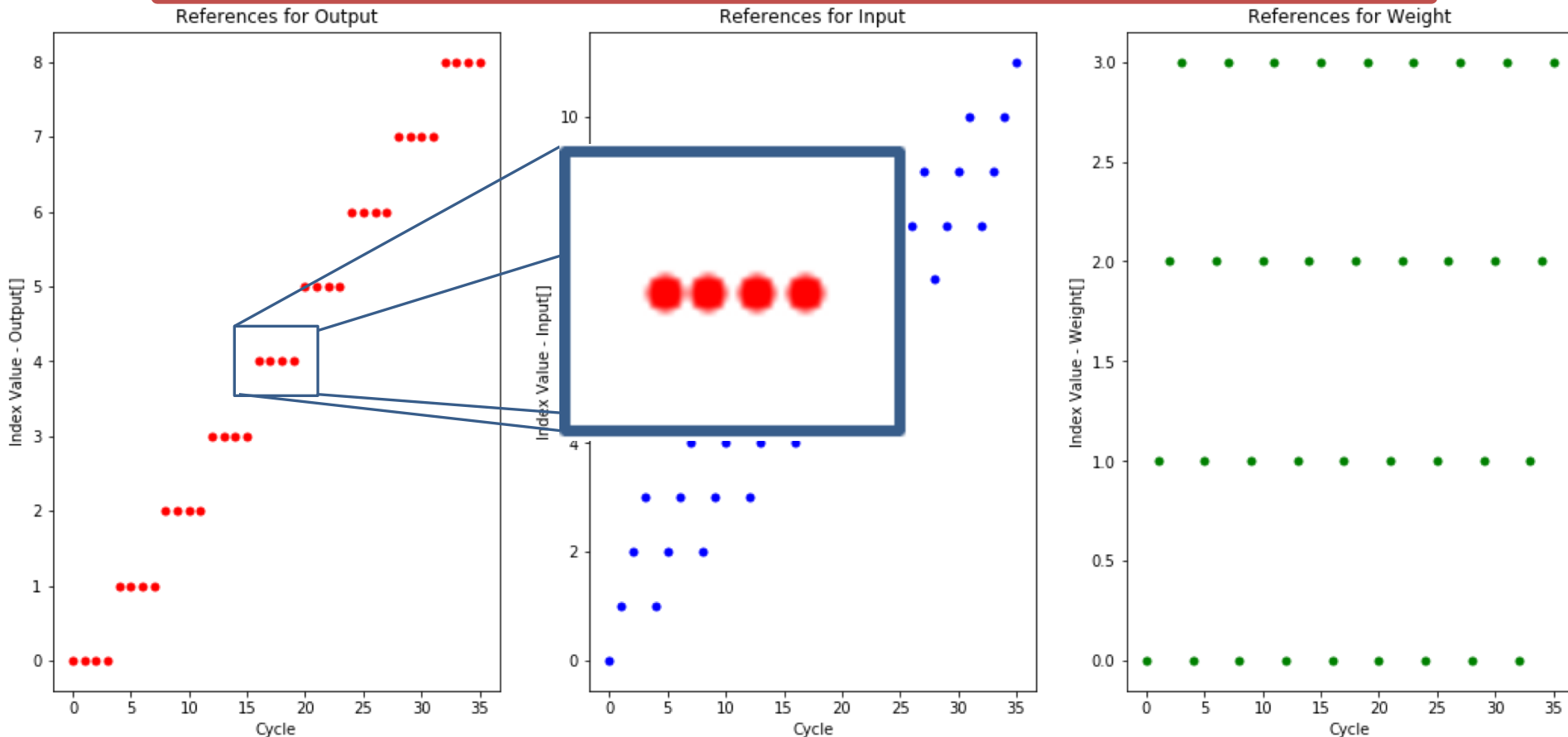
References for Input



References for Weight



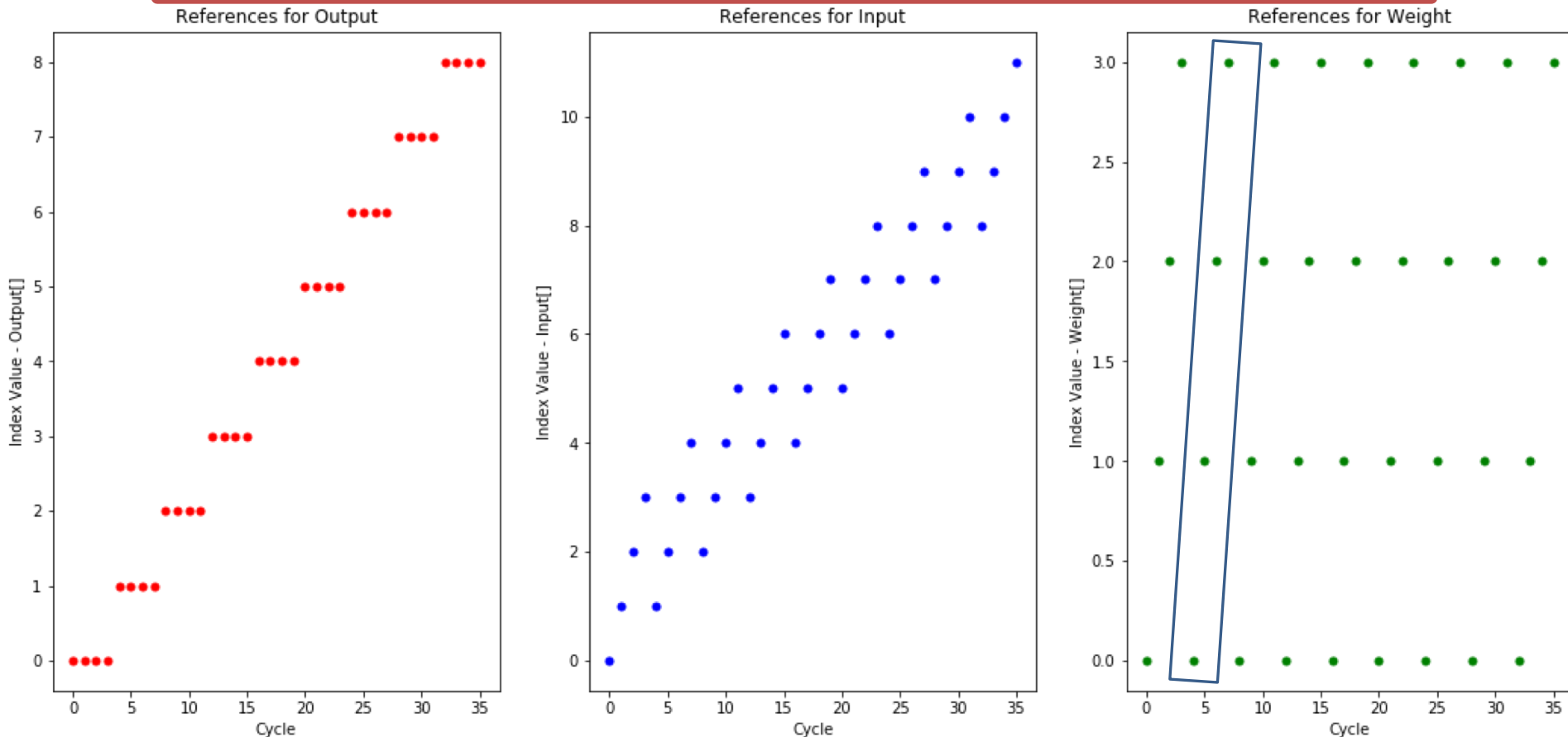
Output Stationary – Reference Pattern



Observations:

- Single **output** is reused many times (R)

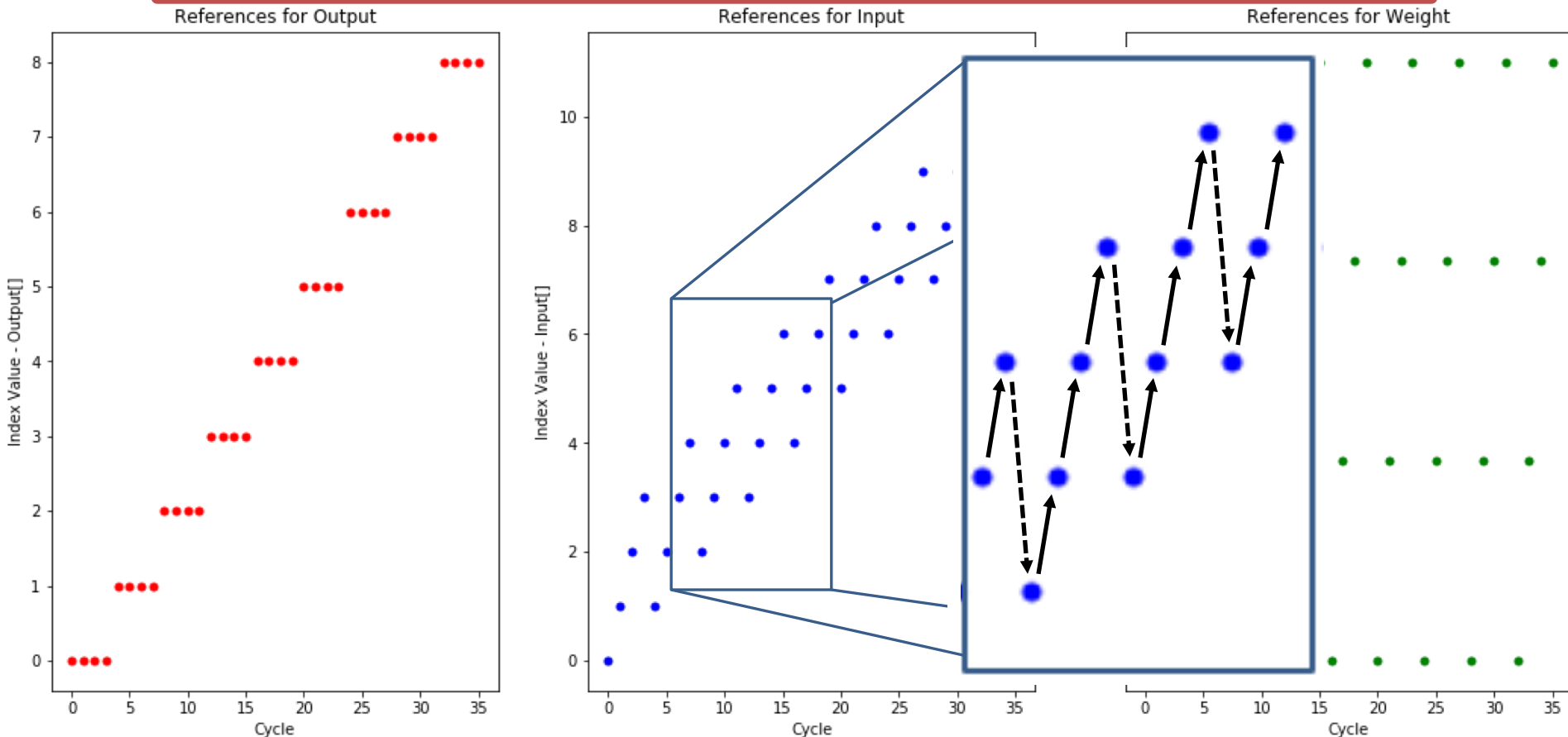
Output Stationary – Reference Pattern



Observations:

- Single **output** is reused many times (R)
- All **weights** reused repeatedly

Output Stationary – Reference Pattern



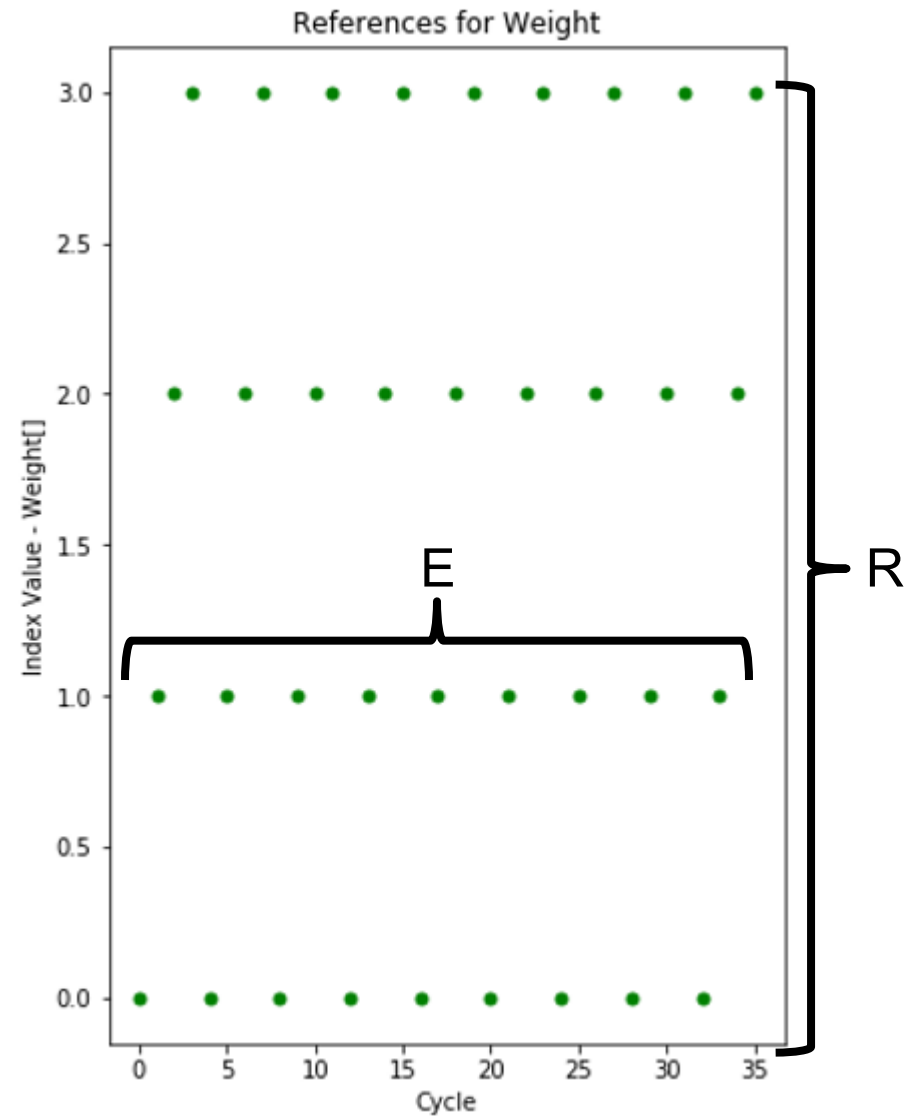
Observations:

- Single **output** is reused many times (R)
- All **weights** reused repeatedly
- Sliding window of **inputs** (size = R)

Buffer Data Accesses - Weights

```
for (e = 0; e < E; e++)  
  for (r = 0; r < R; r++)  
    O[e] += I[e+r] * W[r];
```

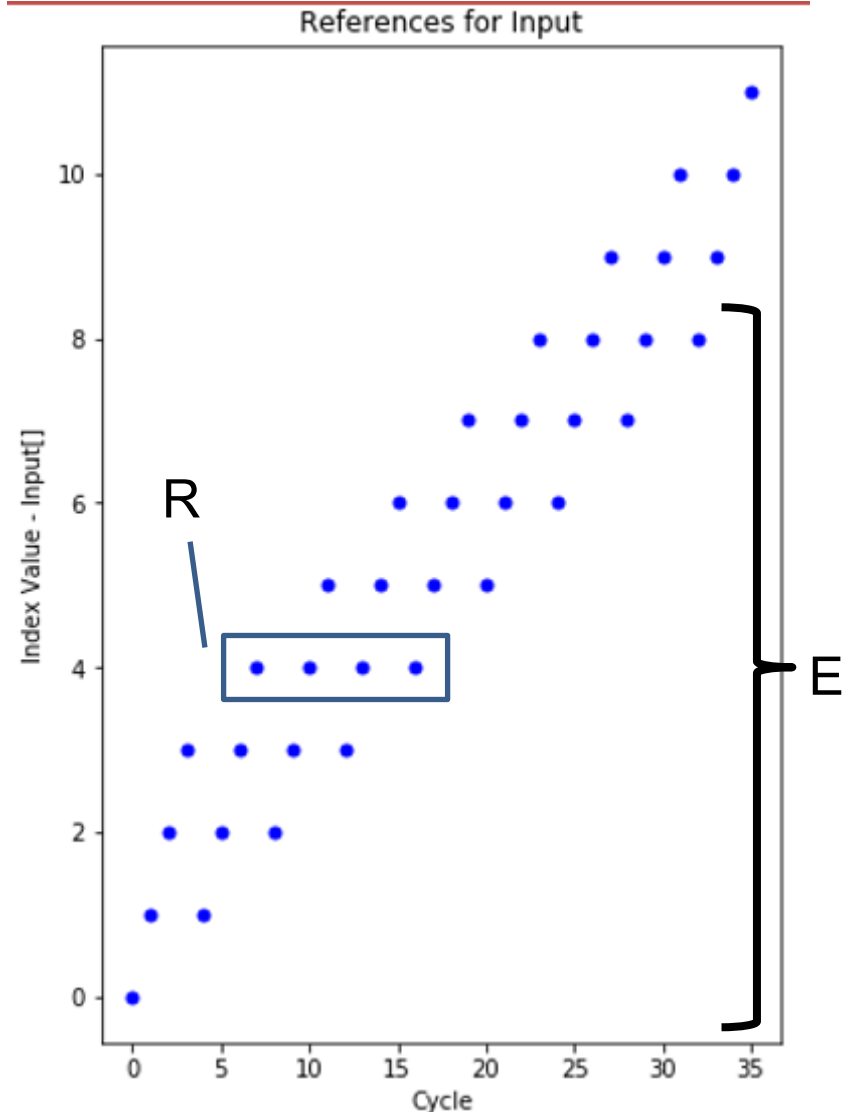
	OS
MACs	$E \cdot R$
Weight Reads	$E \cdot R$
Input Reads	
Output Reads	
Output Writes	



Buffer Data Accesses - Inputs

```
for (e = 0; e < E; e++)  
  for (r = 0; r < R; r++)  
    O[e] += I[e+r] * W[r];
```

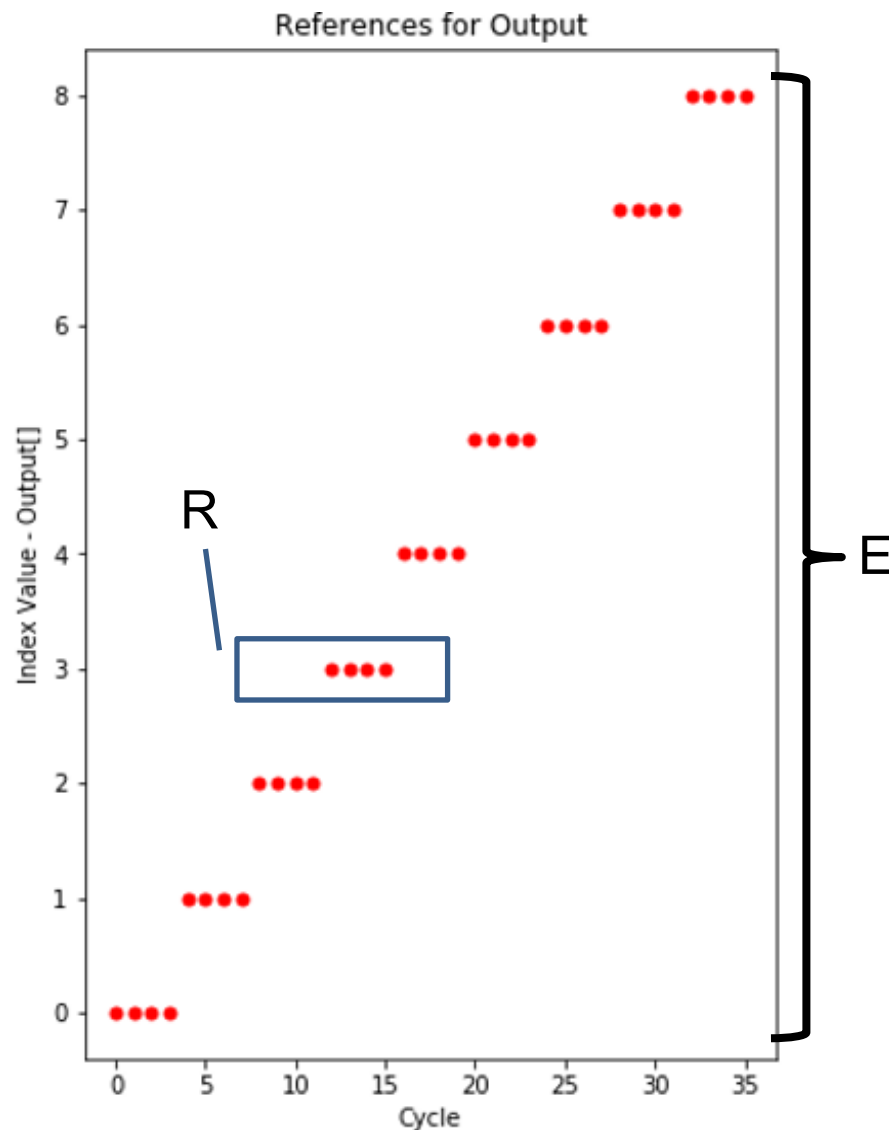
	OS
MACs	$E \cdot R$
Weight Reads	$E \cdot R$
Input Reads	$E \cdot R$
Output Reads	
Output Writes	



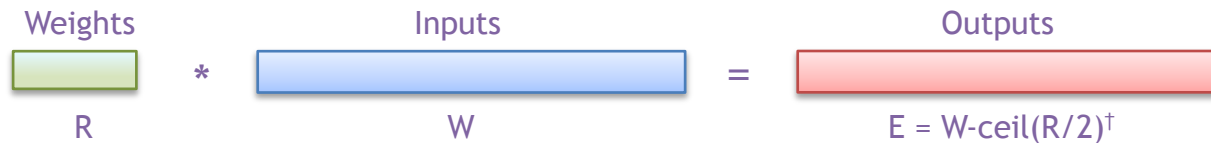
Buffer Data Accesses - Outputs

```
for (e = 0; e < E; e++)  
  for (r = 0; r < R; r++)  
    O[e] += I[e+r] * W[r];
```

	OS
MACs	$E \cdot R$
Weight Reads	$E \cdot R$
Input Reads	$E \cdot R$
Output Reads	0
Output Writes	E



1-D Convolution – Weight Stationary

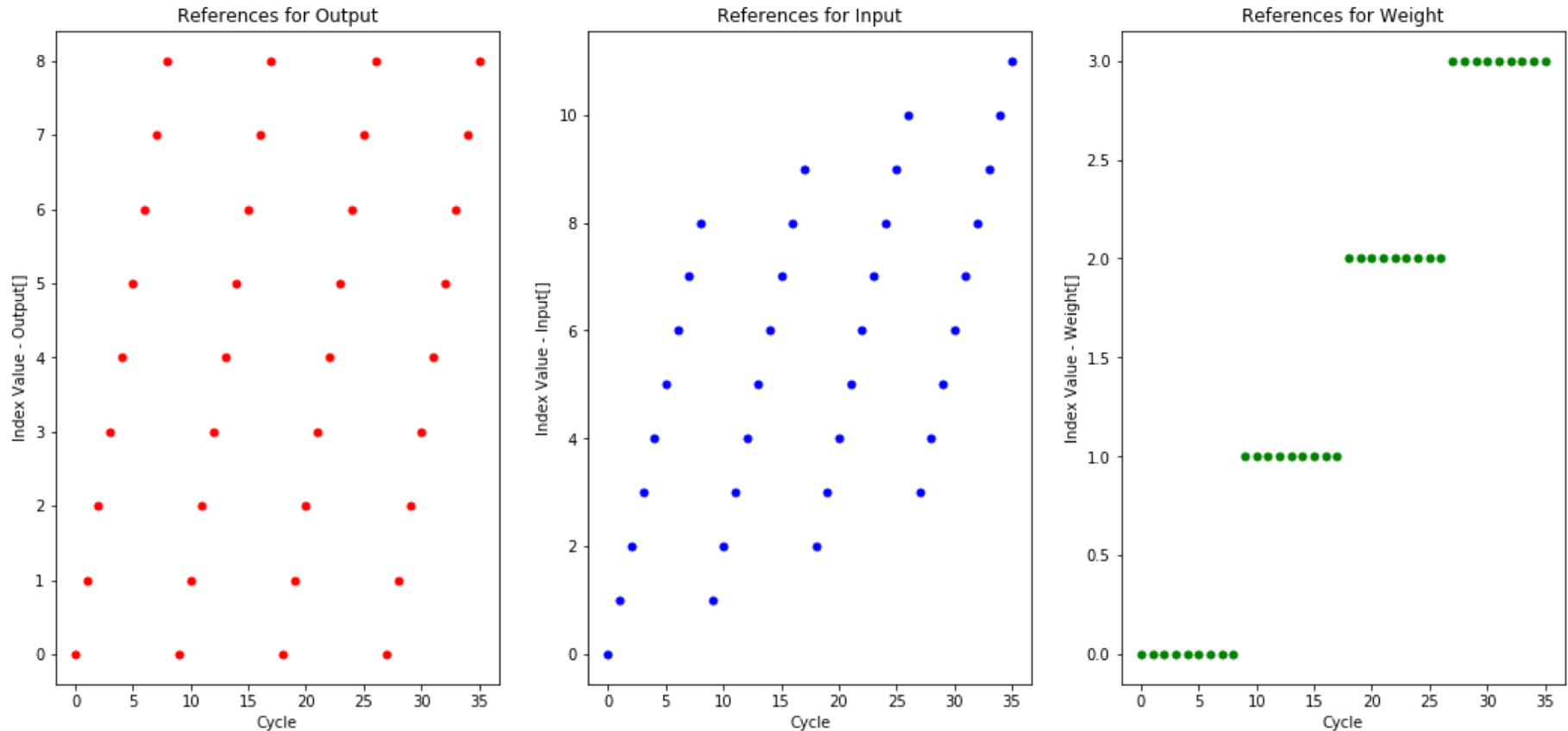


```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations

for (e = 0; e < E; e++)
    for (r = 0; r < R; r++)
        O[e] += I[e+r] * W[r];
```

[†] Assuming: 'valid' style convolution

Weight Stationary – Reference Pattern



Observations:

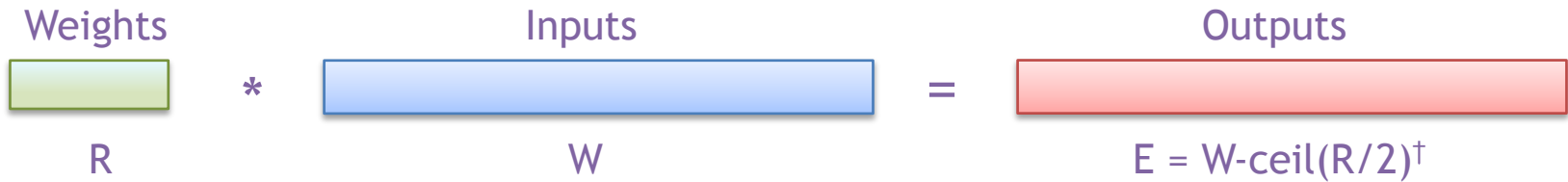
- Single **weight** is reused many times (E)
- Large sliding window of **inputs** (size = E)
- Fixed window of **outputs** (size = E)

L1 Weight Stationary - Costs

```
for (r = 0; r < R; r++)  
  for (e = 0; e < E; e++)  
    O[e] += I[e+r] * W[r];
```

	OS	WS		
MACs	$E \cdot R$	$E \cdot R$		
Weight Reads	$E \cdot R$	R		
Input Reads	$E \cdot R$	$E \cdot R$		
Output Reads	0	$E \cdot R$		
Output Writes	E	$E \cdot R$		

1-D Convolution – Input Stationary



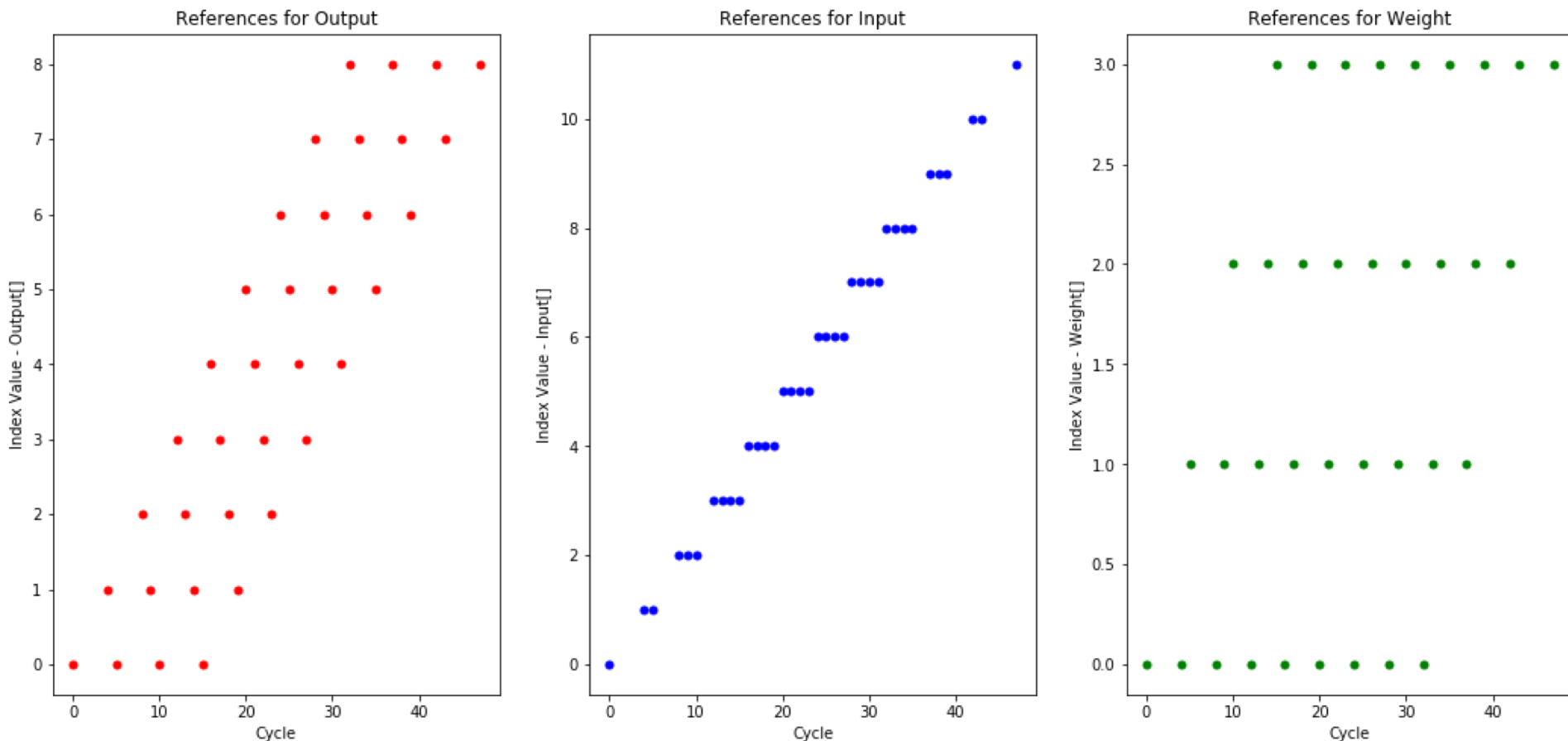
```
int I[H];      // Input activations
int W[R];      // Filter weights
int O[E];      // Output activations

for (h = 0; h < H; h++)
    for (r = 0; r < R; r++)
        O[h-r] += I[h] * W[r];
```

Beware w-r
must be ≥ 0
and $< E$

[†] Assuming: ‘valid’ style convolution

Input Stationary – Reference Pattern



Observations:

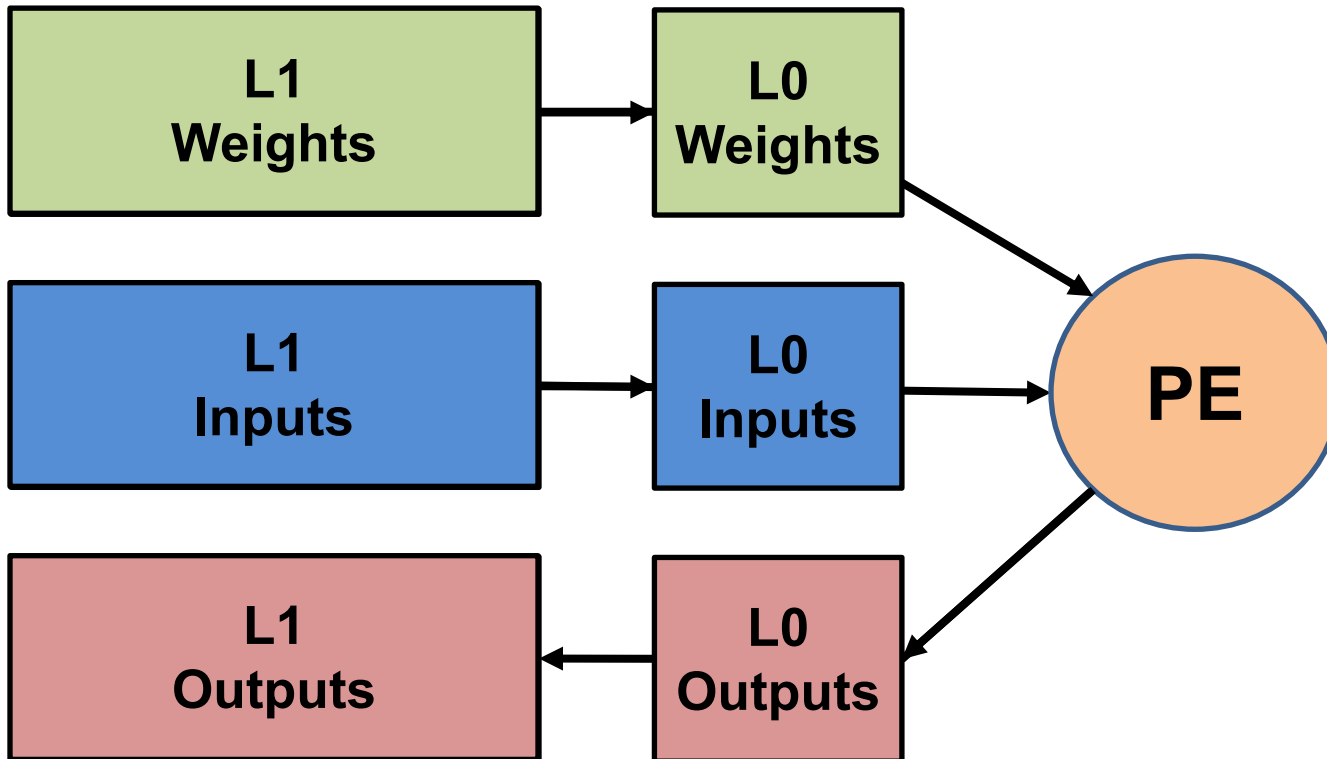
- **Inputs** used repeatedly (R times)
- **Weights** reused in large window (size = R)
- Sliding window of **outputs** (size = R)

Minimum Costs

	OS	WS	IS	Min
MACs	$E \cdot R$	$E \cdot R$	$E \cdot R$	$E \cdot R$
Weight Reads	$E \cdot R$	R	$E \cdot R$	R
Input Reads	$E \cdot R$	$E \cdot R$	E	E
Output Reads	0	$E \cdot R$	$E \cdot R$	0
Output Writes	E	$E \cdot R$	$E \cdot R$	E

Assume: $W \sim E$

Intermediate Buffering



1-D Convolution – Buffered



```
int I[H];    // Input activations
int W[R];    // Filter Weights
int O[E];    // Output activations
```

```
// Level 1
```

```
for (e1 = 0; e1 < E1; e1++)
```

```
    for (r1 = 0; r1 < R1; r1++)
```

```
        // Level 0
```

```
            for (e0 = 0; e0 < E0; e0++)
```

```
                for (r0 = 0; r0 < R0; r0++)
```

```
                    O[e1*E0+e0] += I[e1*E0+e0 + r1*R0+r0]
                                   * W[r1*R0+r0];
```

Note E and R are factored so:
 $E0 * E1 = E$
 $R0 * R1 = R$

[†] Assuming: 'valid' style convolution

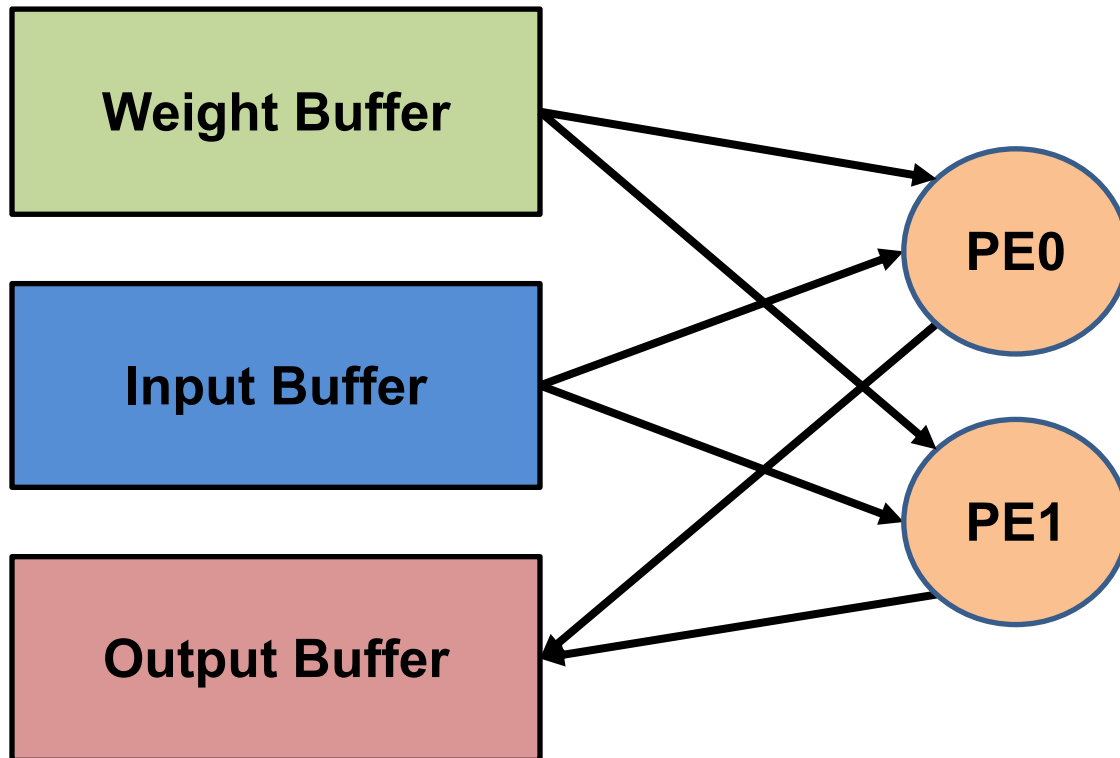
Buffer sizes

```
// Level 1
for (e1 = 0; e1 < E1; e1++)
  for (r1 = 0; r1 < R1; r1++)
    // Level 0
    for (e0 = 0; e0 < E0; e0++)
      for (r0 = 0; r0 < R0; r0++)
        O[e1*E0+e0] += I[e1*E0+e0 + r1*R0+r0] * W[r1*R0+r0];
```

- Level 0 buffer size is volume needed in each Level 1 iteration.
- Level 1 buffer size is volume needed to be preserved and re-delivered in future (usually successive) Level 1 iterations.

A legal assignment of loop limits will fit into the hardware's buffer sizes

Spatial PEs



1-D Convolution – Spatial



```
int I[W];      // Input activations
int W[R];      // Filter Weights
int O[E];      // Output activations
```

```
// Level 1
```

```
for (r1 = 0; r1 < R1; r1++)
    for (e1 = 0; e1 < E1; e1++)
```

```
    // Level 0
```

```
        spatial-for (r0 = 0; r0 < R0; r0++)
```

```
            spatial-for (e0 = 0; e0 < E0; e0++)
```

```
                O[e1*E0+e0] += I[e1*E0+e0 + r1*R0+r0]
                               * W[r1*R0+r0];
```

Note:

- $E0 * E1 = E$
- $R0 * R1 = R$
- $R0 * E0 \leq \text{\#PEs}$

[†] Assuming: 'valid' style convolution

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.