6.5930/1

Hardware Architectures for Deep Learning

Dataflow for DNN Accelerator Architectures (Part 2)

March 13, 2024

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Lecture

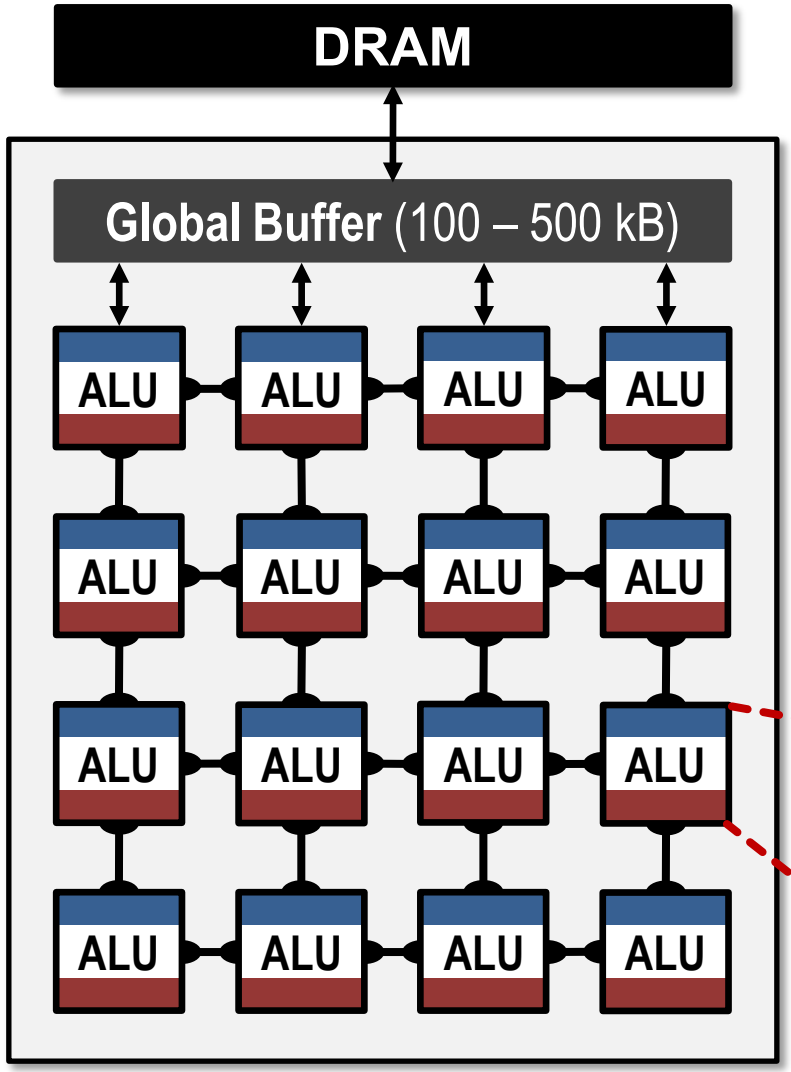
- Taxonomy of dataflows for CNNs
  - Output Stationary (last lecture)
  - Weight Stationary
  - Input Stationary
- Advanced dataflow
  - Row Stationary (Eyeriss)
- Data Orchestration

Background Reading

- **DNN Accelerators**
  - ***Efficient Processing of Deep Neural Networks***
    - Chapter 5

All these books and their online/e-book versions are available through MIT libraries.

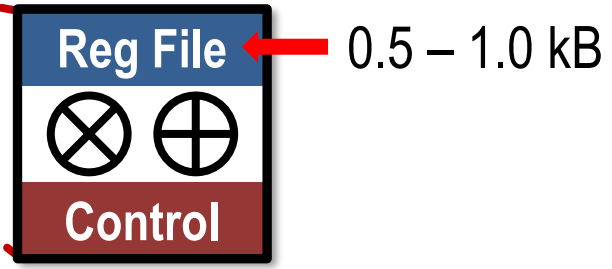
Spatial Architecture for DNN



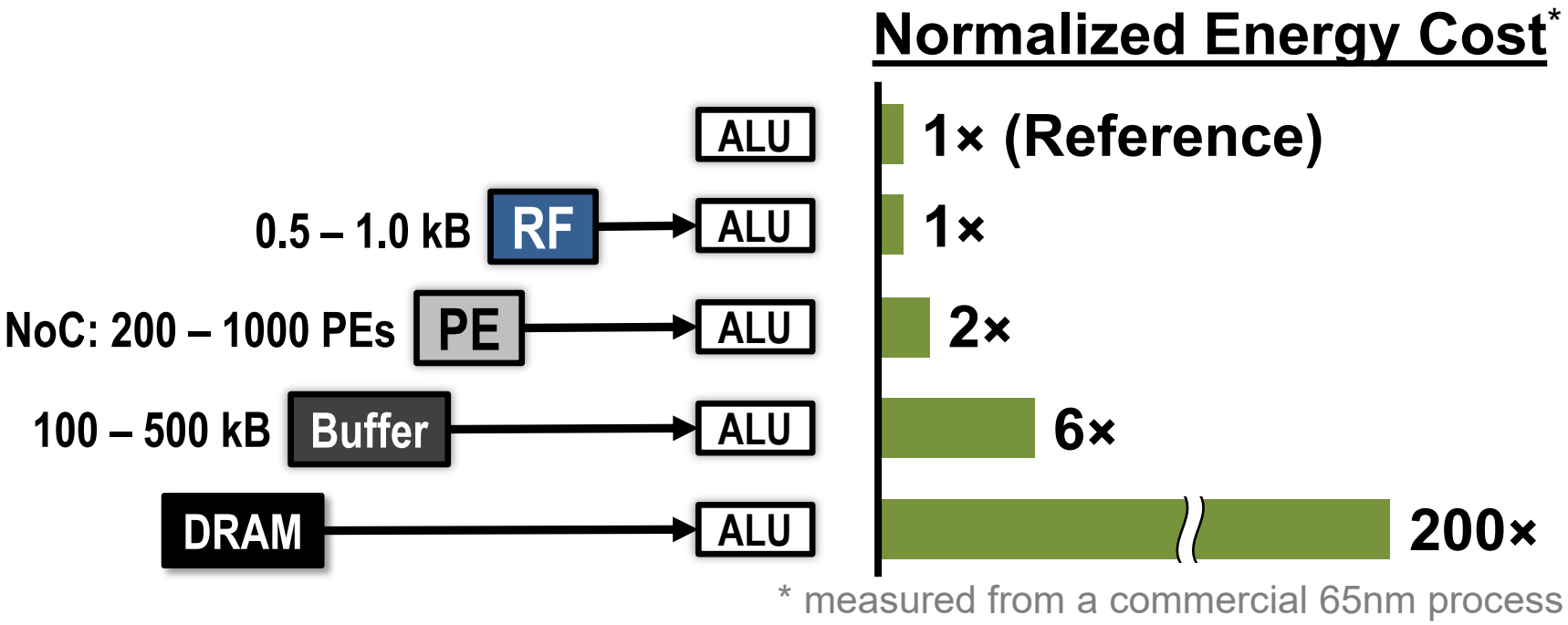
Local Memory Hierarchy

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

Processing Element (PE)



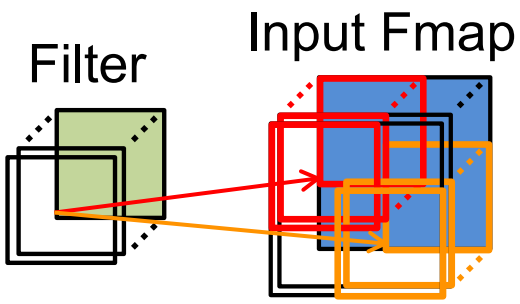
Energy Cost of Data Movement



Types of Data Reuse in DNN

Convolutional Reuse

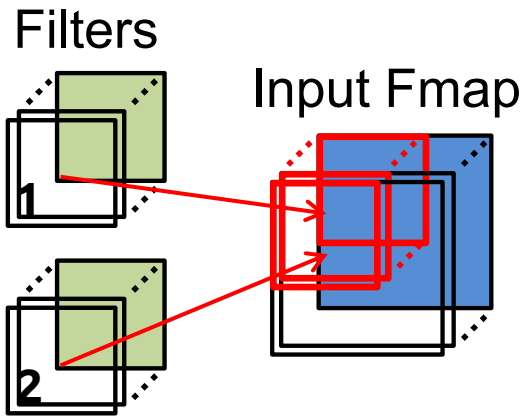
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

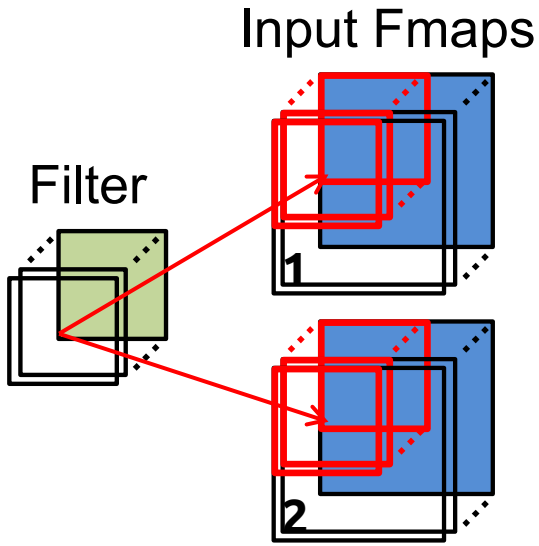
CONV and FC layers



Reuse: Activations

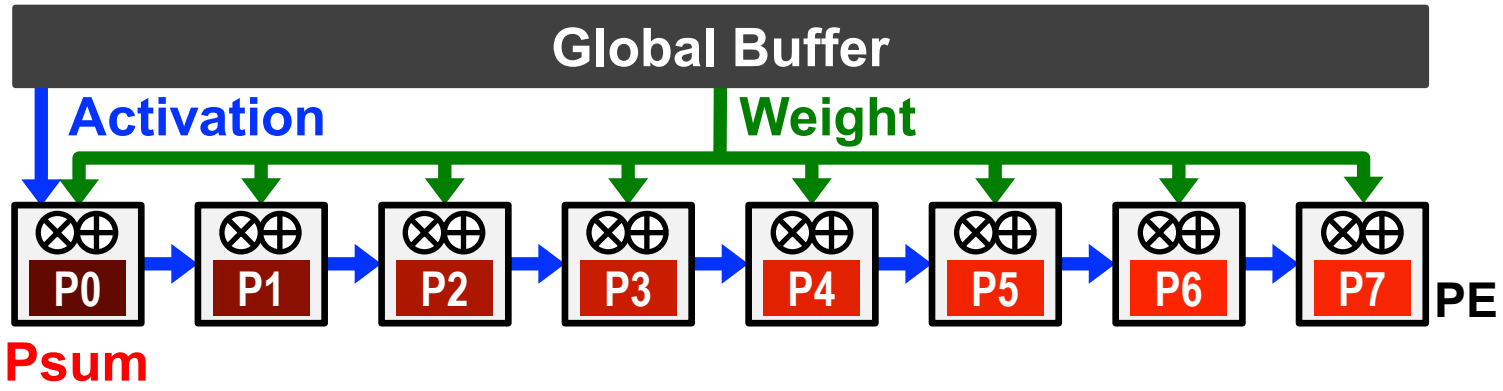
Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

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

Output activations are in the outer loop

1-D Convolution – Output Stationary


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        o[q] += i[q+s]*f[s]
```

No constraints on loop permutations!

† Assuming: 'valid' style convolution

1-D Convolution

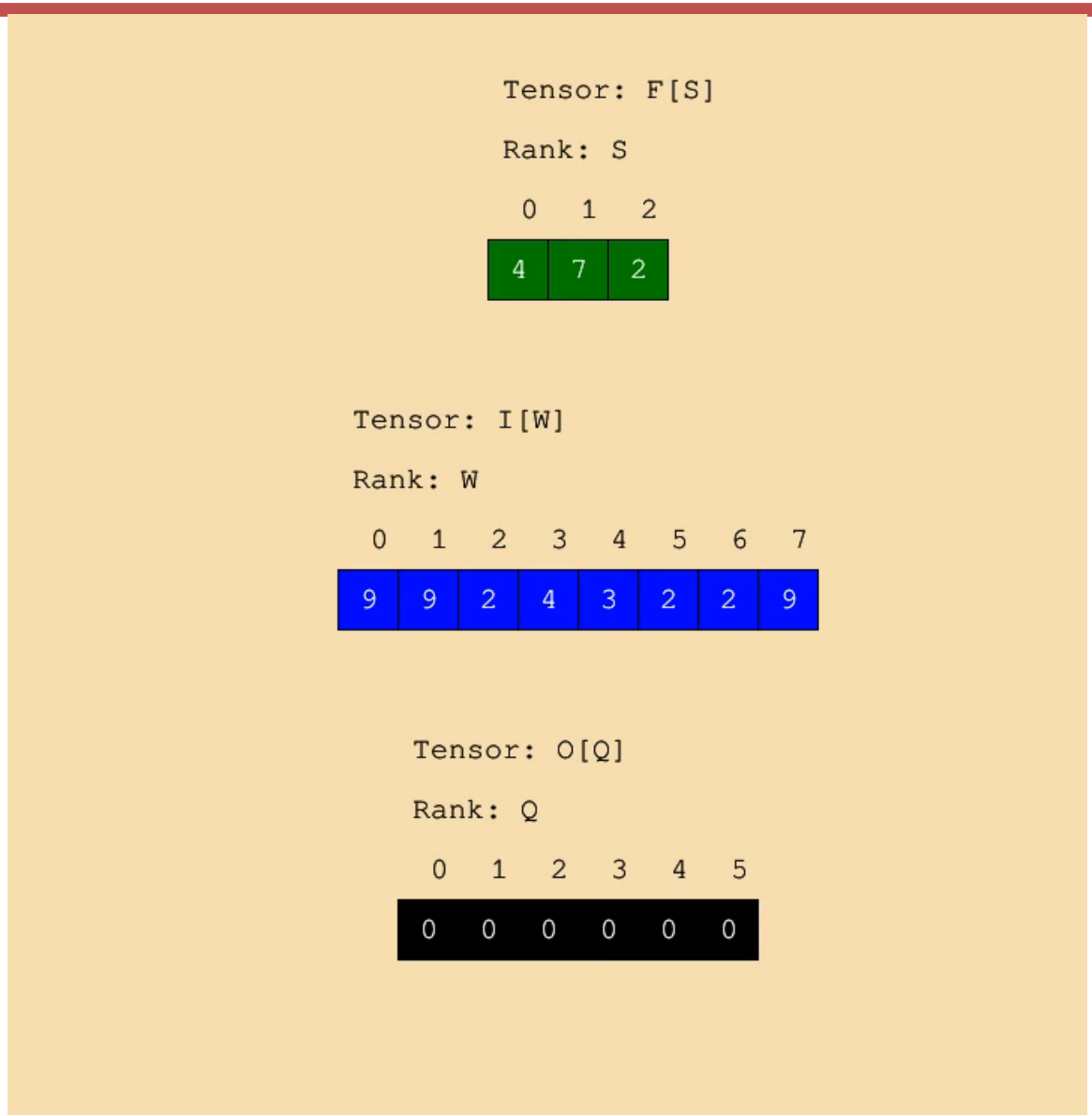

```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for s in [0, S):
    for q in [0, Q):
        o[q] += i[q+s]*f[s]
```

What dataflow is this?

† Assuming: 'valid' style convolution

Weight Stationary

Weight Stationary - Animation



Weight Stationary - Spacetime

Tensor: F[S, T]

Rank: T

Rank: S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Tensor: I[W, T]

Rank: T

Rank: W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
1	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
5	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
6	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
7	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

Tensor: O[Q, T]

Rank: T

Rank: Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	36	36	36	36	36	36	99	99	99	99	99	99	103	103	103	103	103	103
1	0	36	36	36	36	36	36	50	50	50	50	50	50	58	58	58	58	58
2	0	0	8	8	8	8	8	8	36	36	36	36	36	36	42	42	42	42
3	0	0	0	16	16	16	16	16	16	37	37	37	37	37	37	41	41	41
4	0	0	0	0	12	12	12	12	12	12	26	26	26	26	26	26	30	30
5	0	0	0	0	0	8	8	8	8	8	8	22	22	22	22	22	22	40

1-D Convolution Einsum + WS

Serial WS design

Einsum: $O_q = I_{q+s} \times F_s$

Traversal order (fastest to slowest): Q, S

WS design for parallel weights

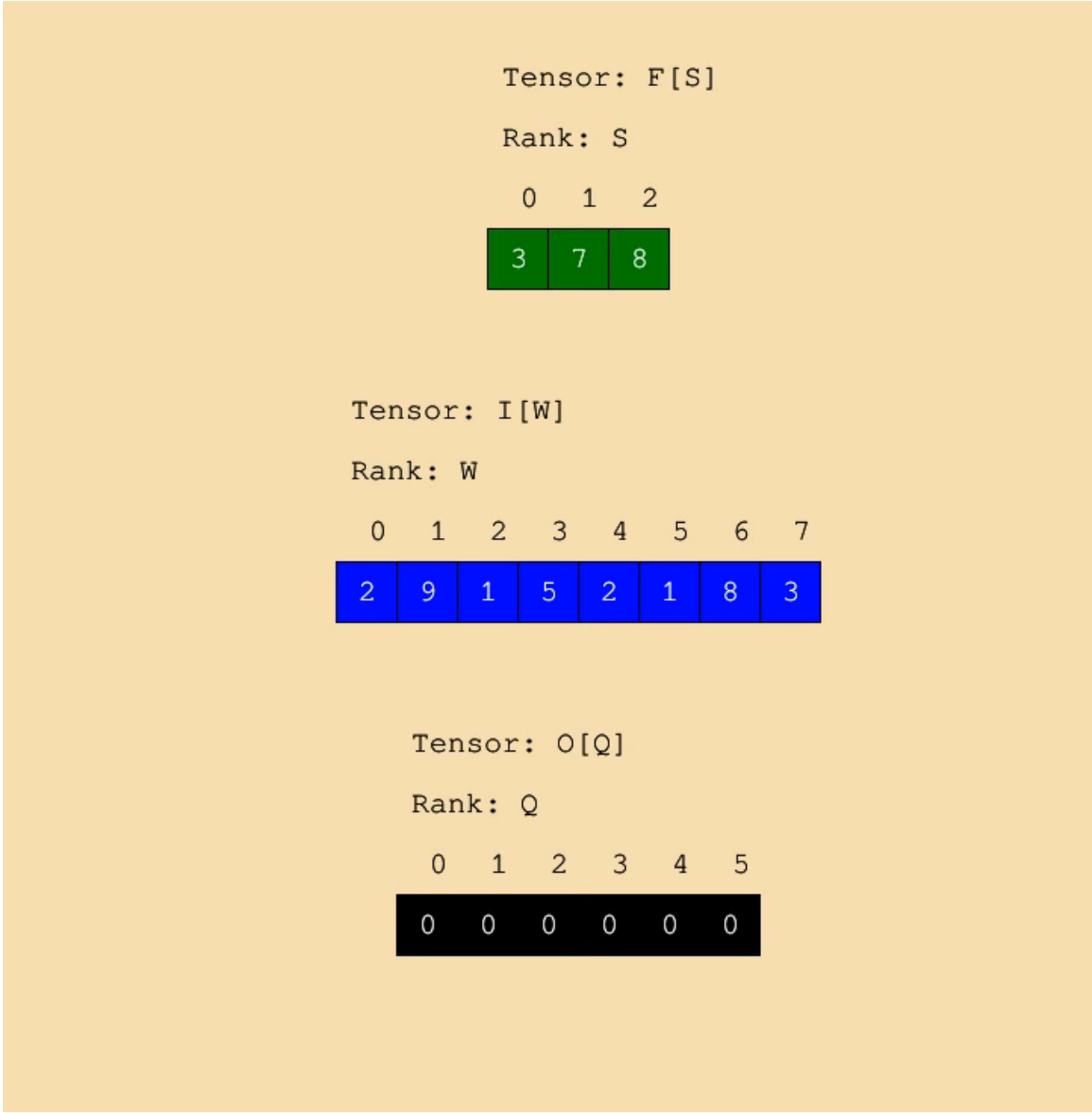
Einsum: $O_q = I_{q+s} \times F_s$

Parallel Ranks: S

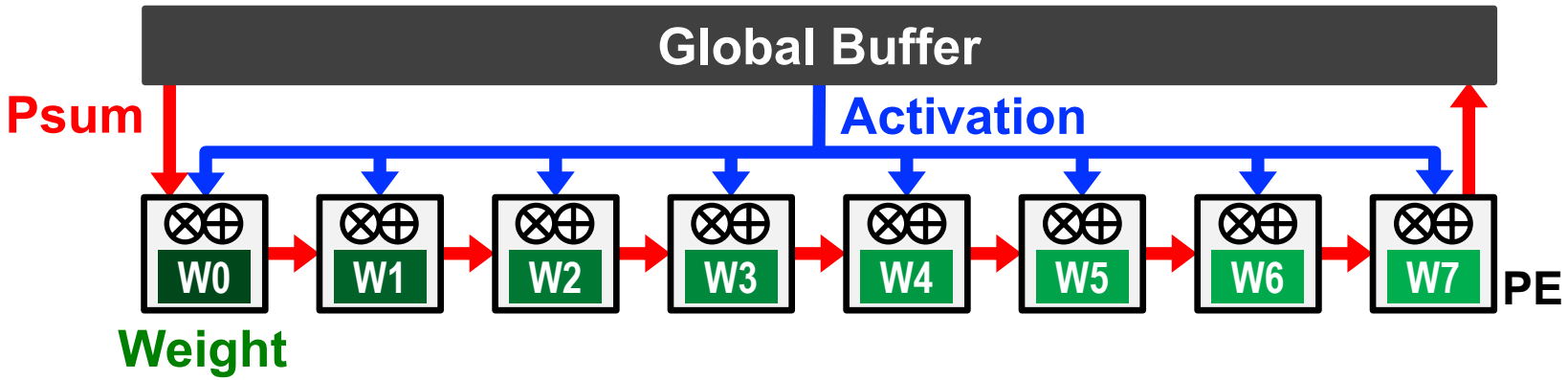
Traversal order (fastest to slowest): Q

Can you write the loop nest? I hope so 😊

Parallel Weight Stationary - Animation



Weight Stationary (WS)

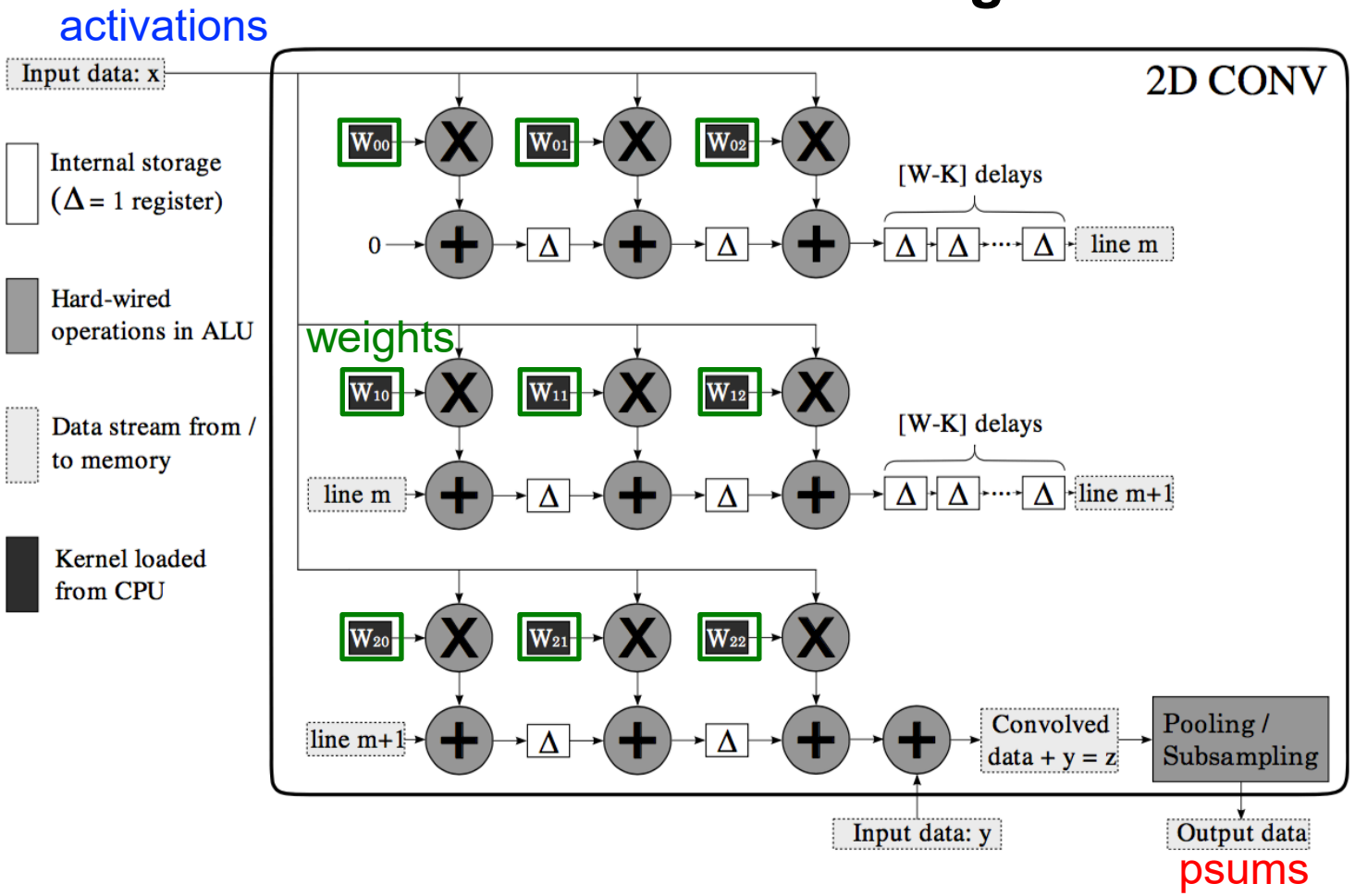


- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights
- **Broadcast activations** and **accumulate psums spatially** across the PE array.

Weights are in the outer loop

WS Example: nn-X (NeuFlow)

A 3×3 2D Convolution Engine

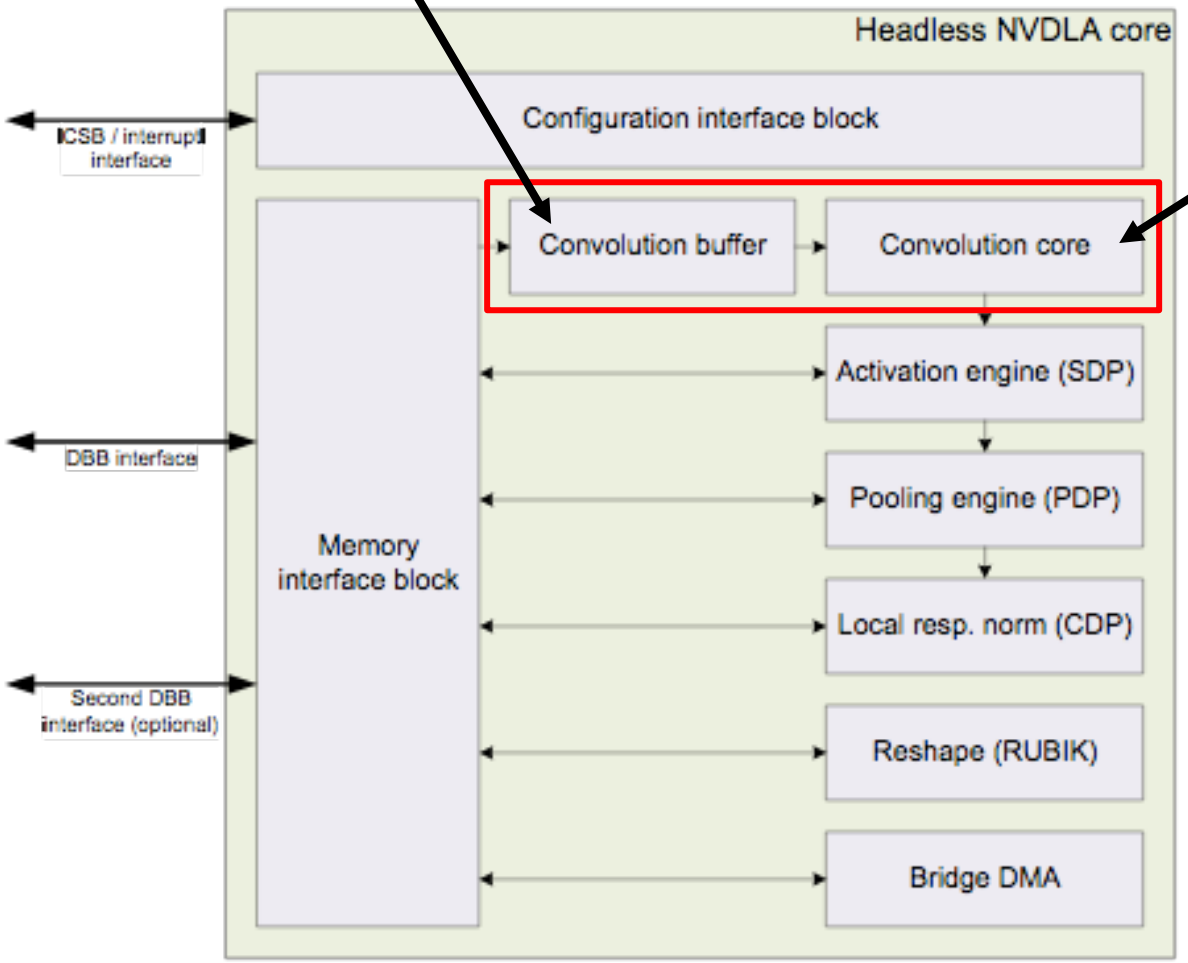


[Farabet et al., ICCV 2009]

WS Example: NVDLA (simplified)

Global Buffer

Released Sept 29, 2017



PE Array

M x C MACs:
The number of input and output channels must match array, otherwise MACs idle
Range:

- C (16 to 128)
- M (4 to 16)

*Note: In this class we use M for number of kernels (filters), while NVDLA uses K

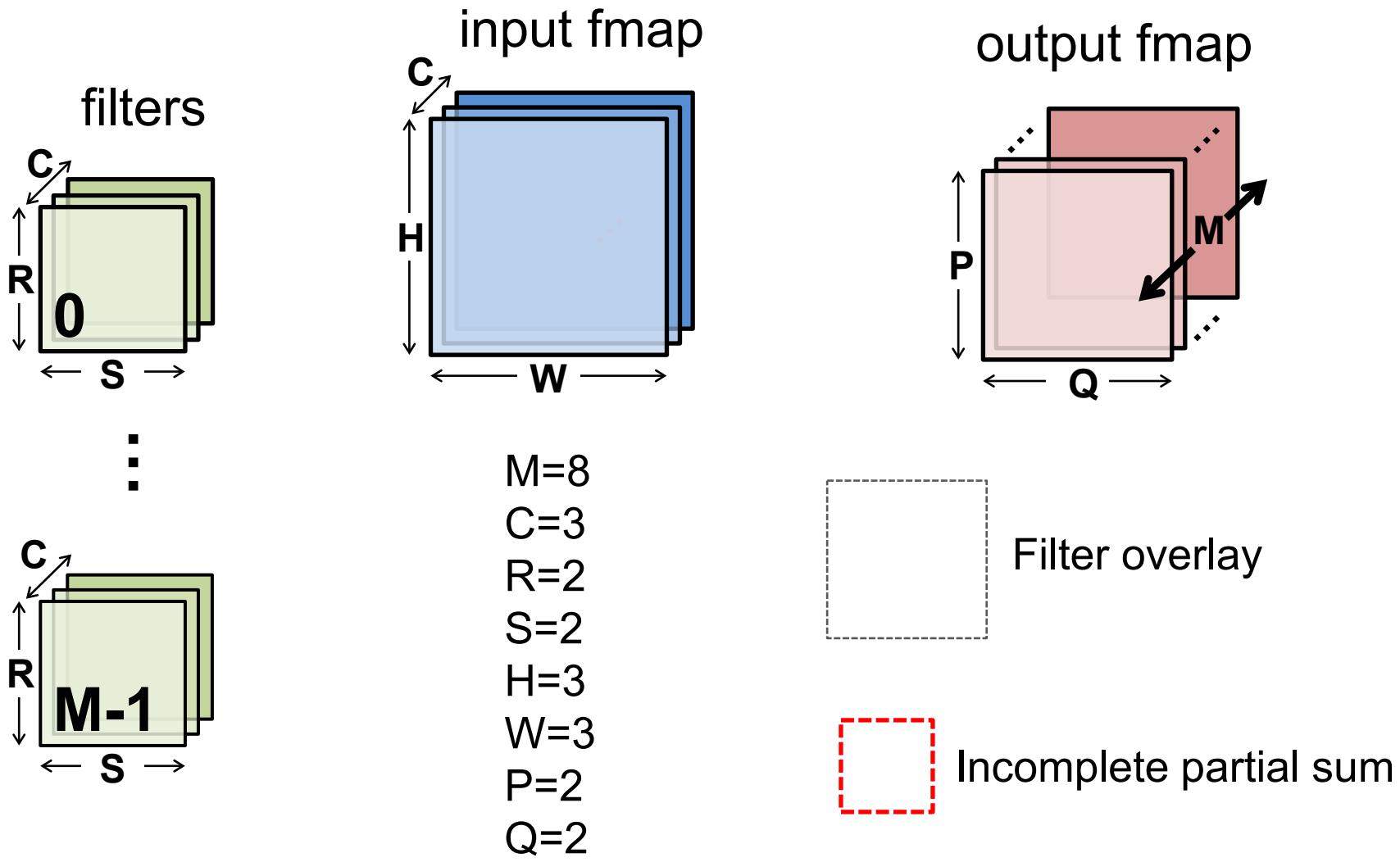
Image Source: Nvidia

http://nvdla.org

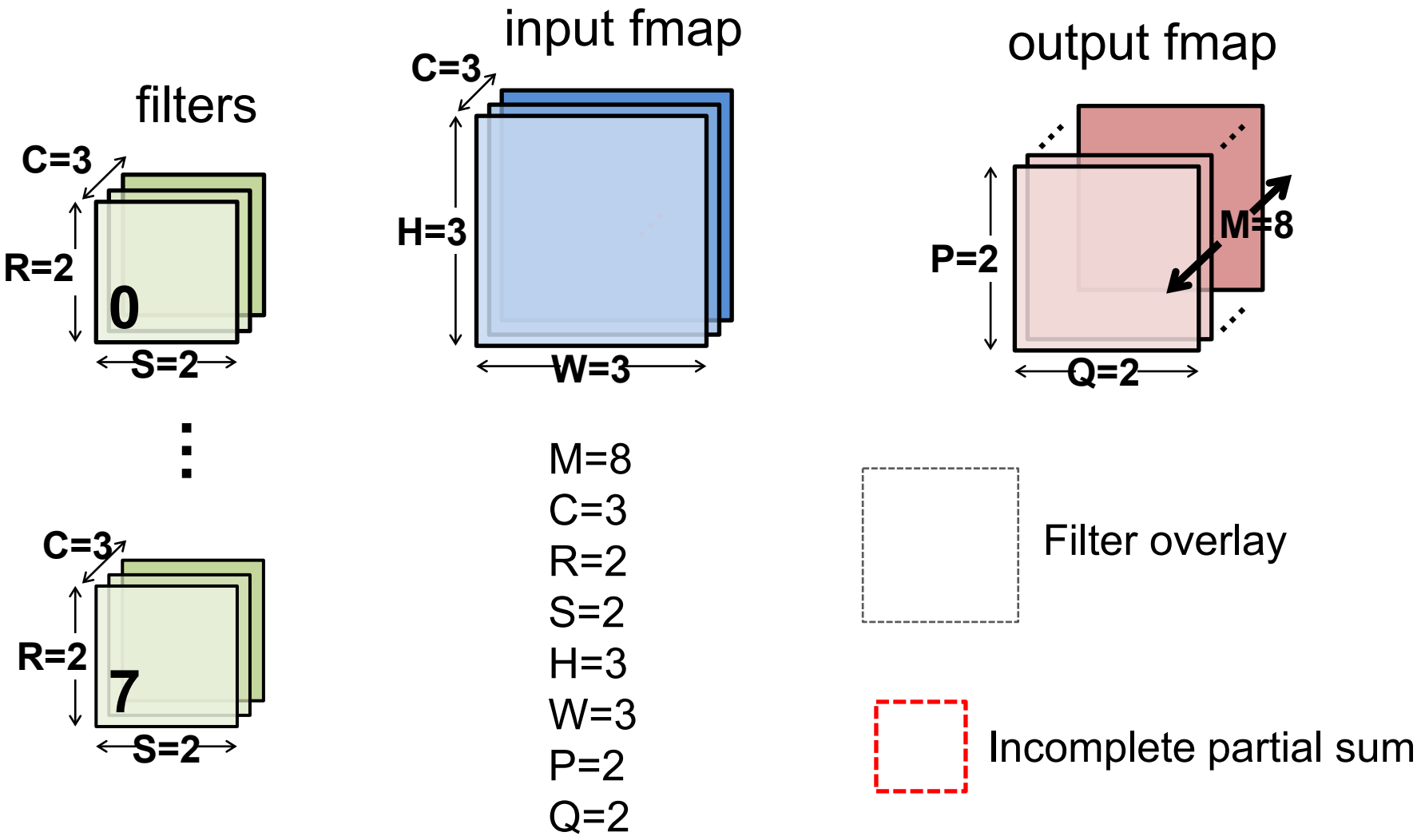
WS Example: Nvidia DLA

- **Convolution Buffer**
  - Stores both weights and activations
  - Ratio of weights and activation data varies across layers
  - Four access ports: R+W feature & R+W weights
    - Convolution read bandwidth determine size of read port (C*datasize)
  - Optional compression support
  - Prefer to either store all weights or all activations

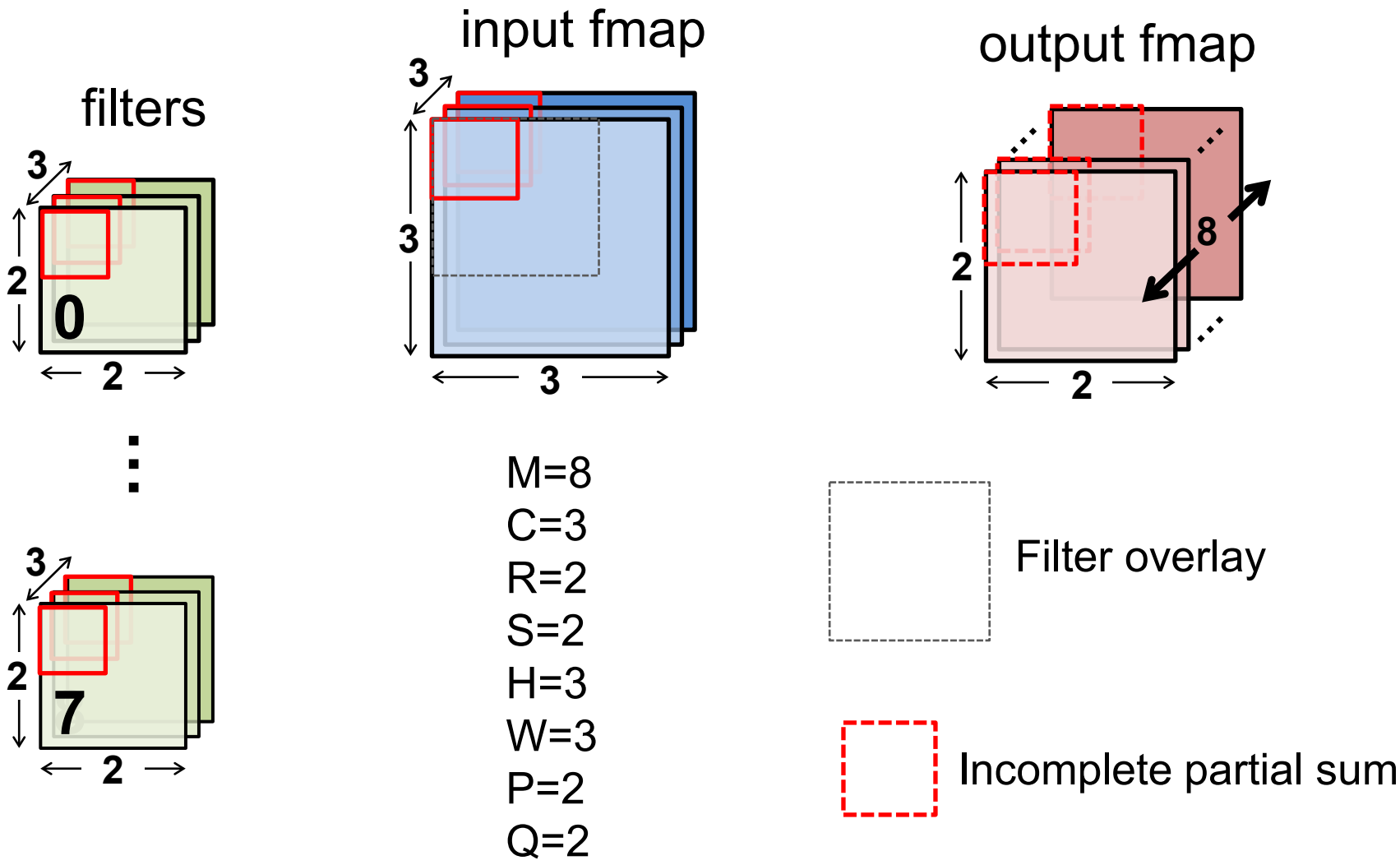
WS Example: NVDLA (simplified)



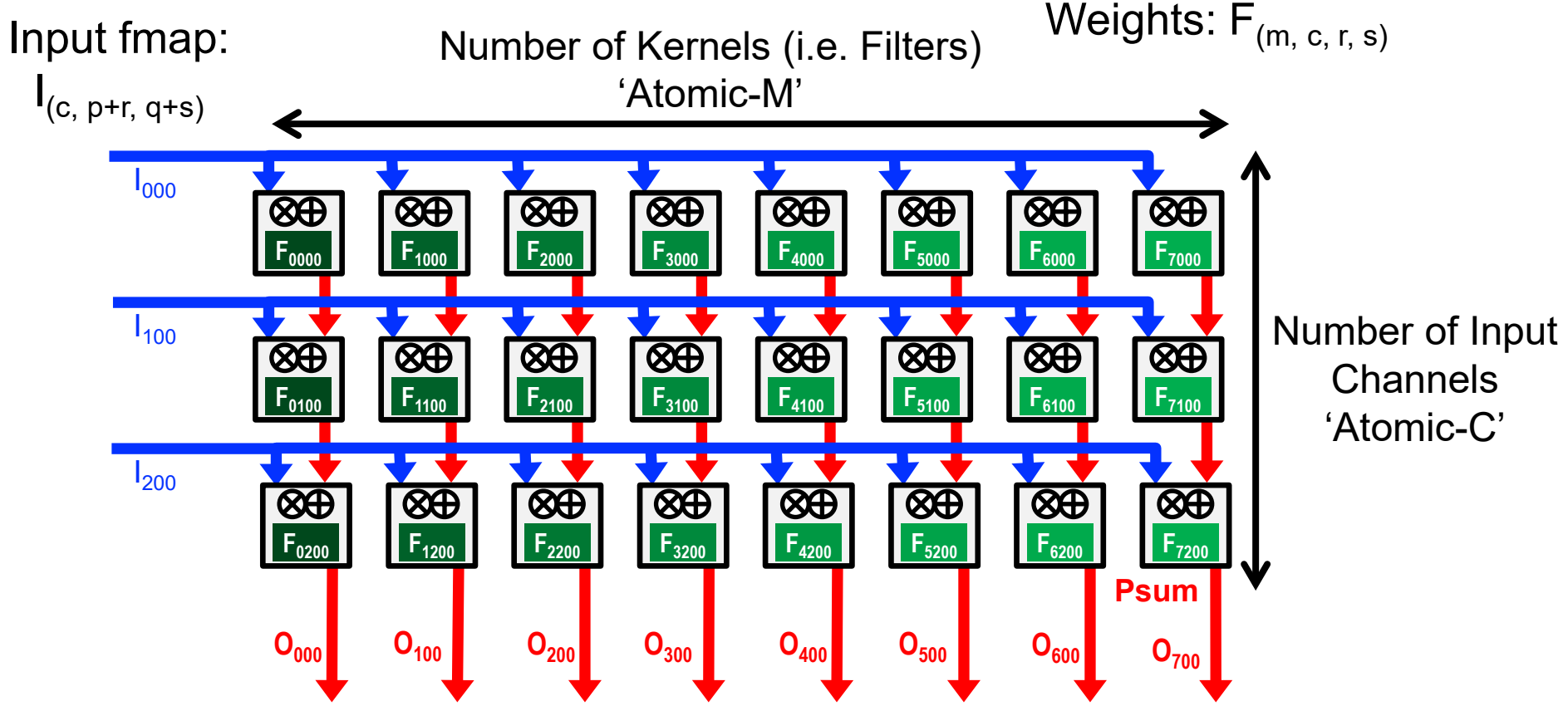
WS Example: NVDLA (simplified)



WS Example: NVDLA (simplified)

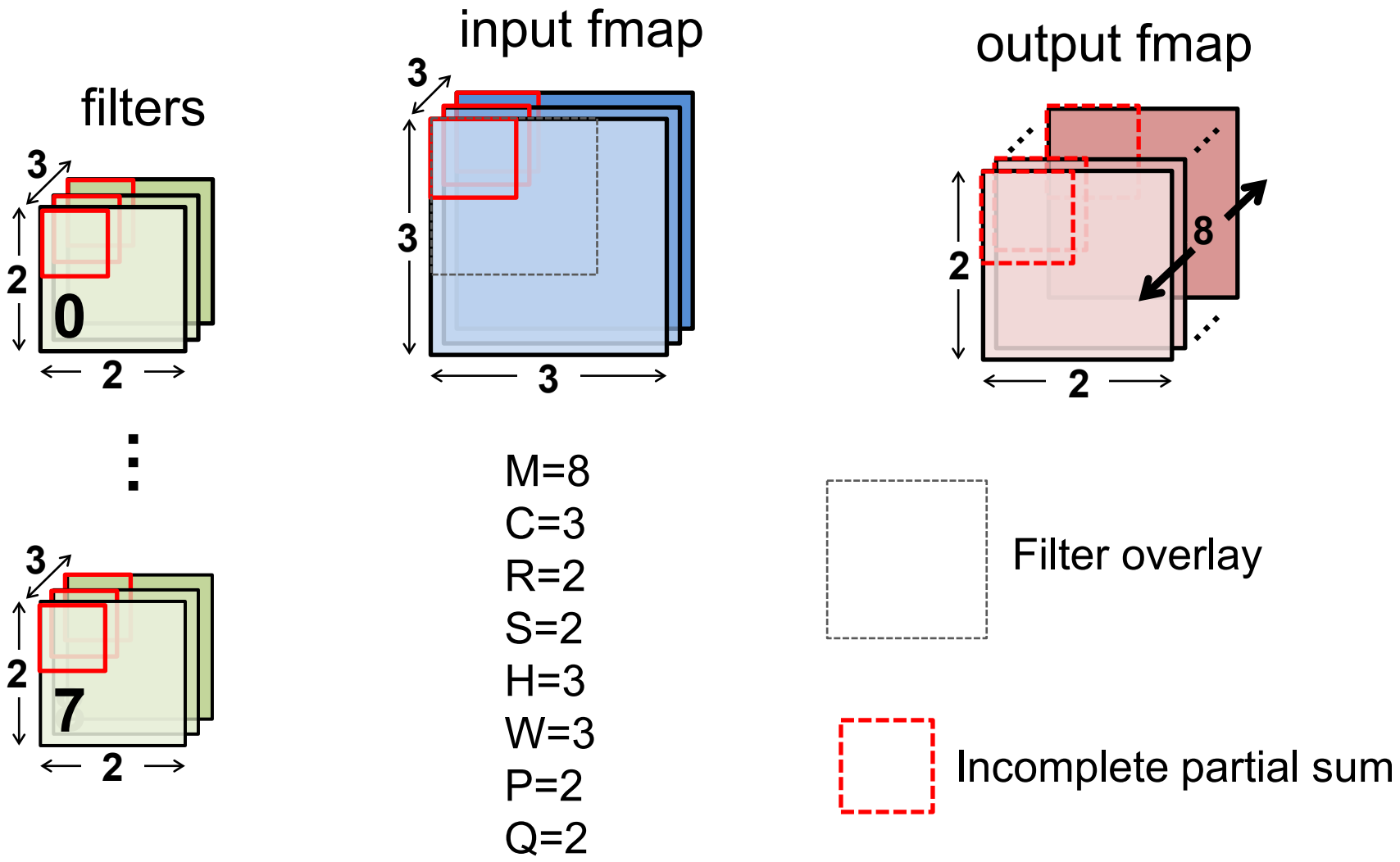


WS Example: NVDLA (simplified)



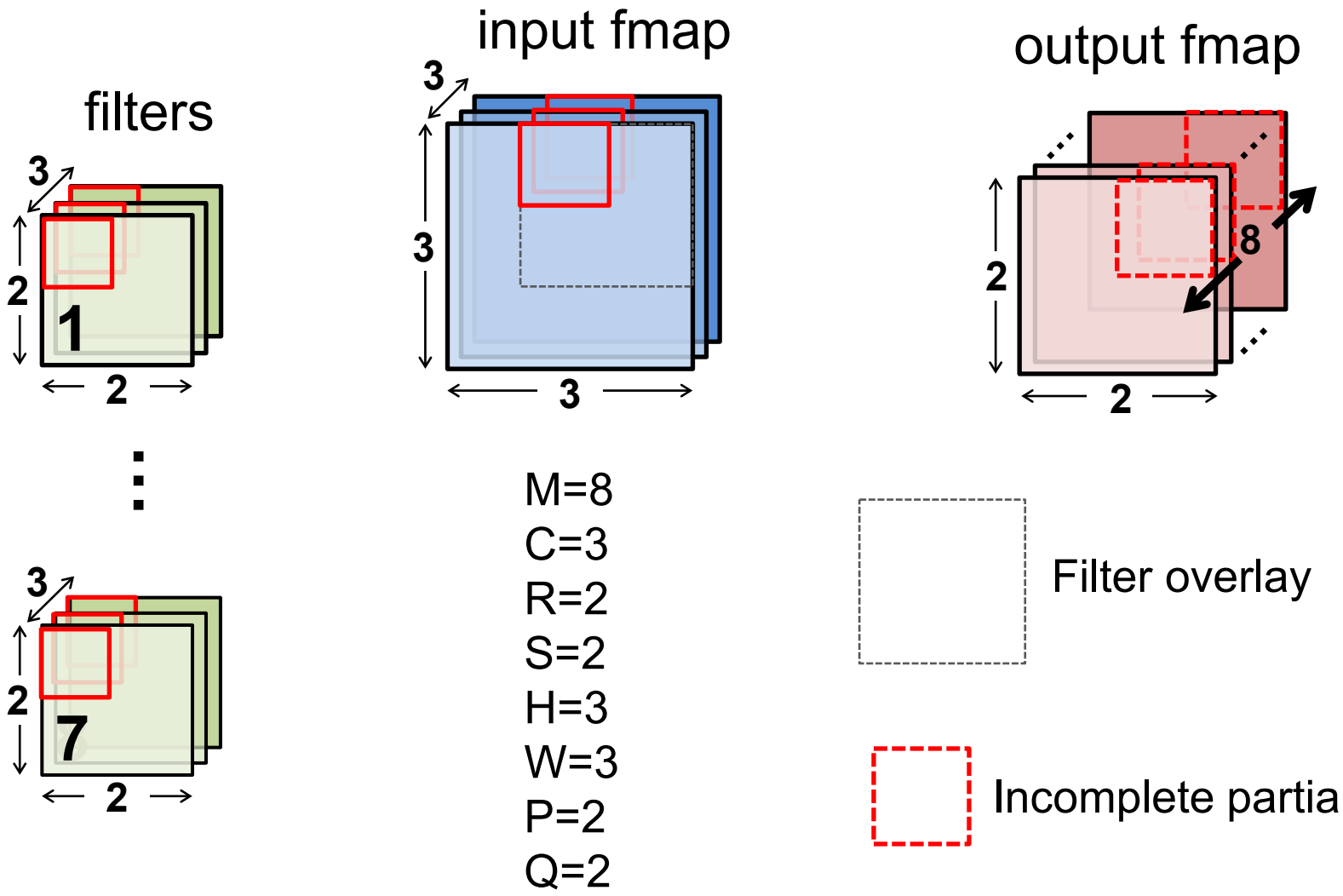
Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)



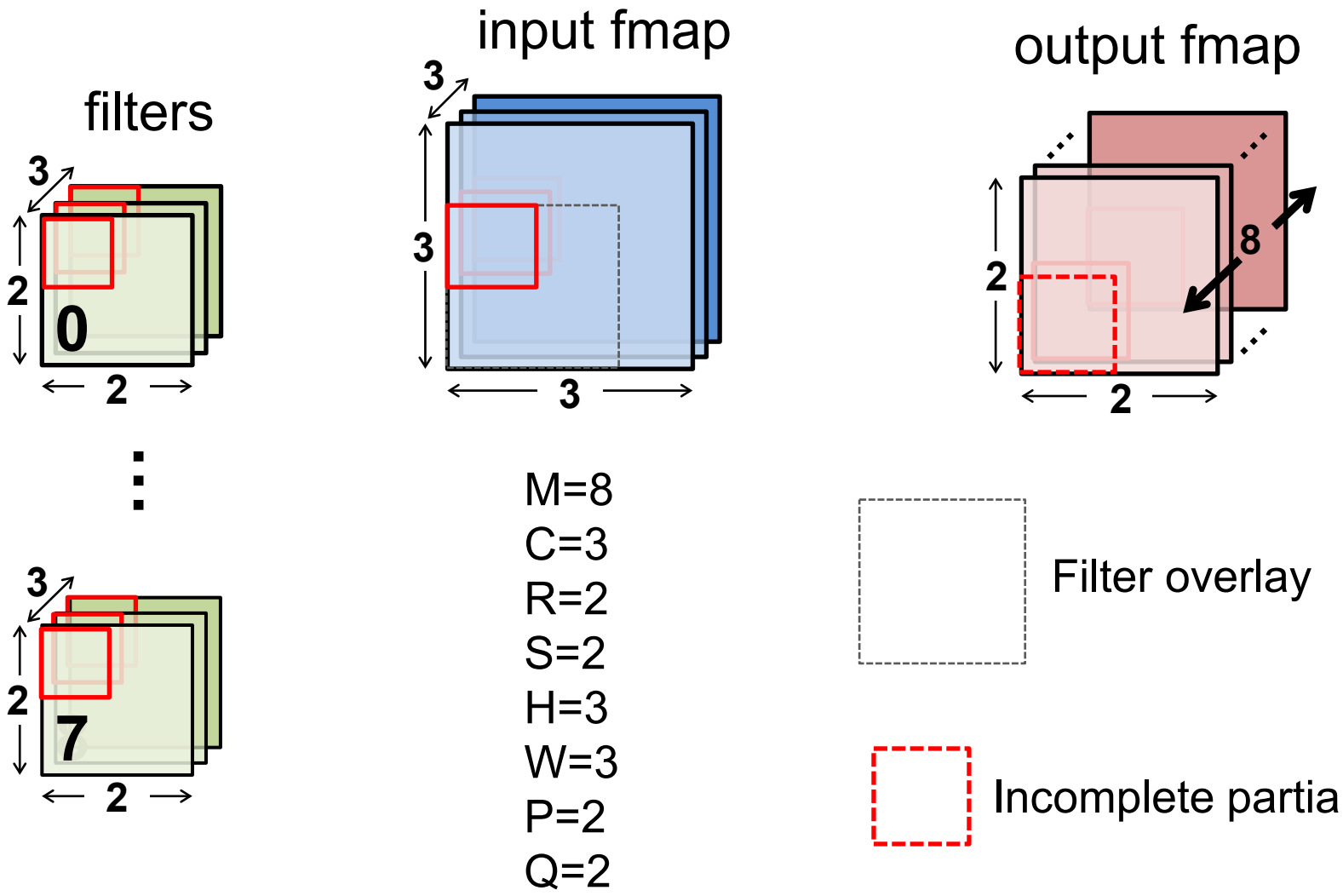
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weight)



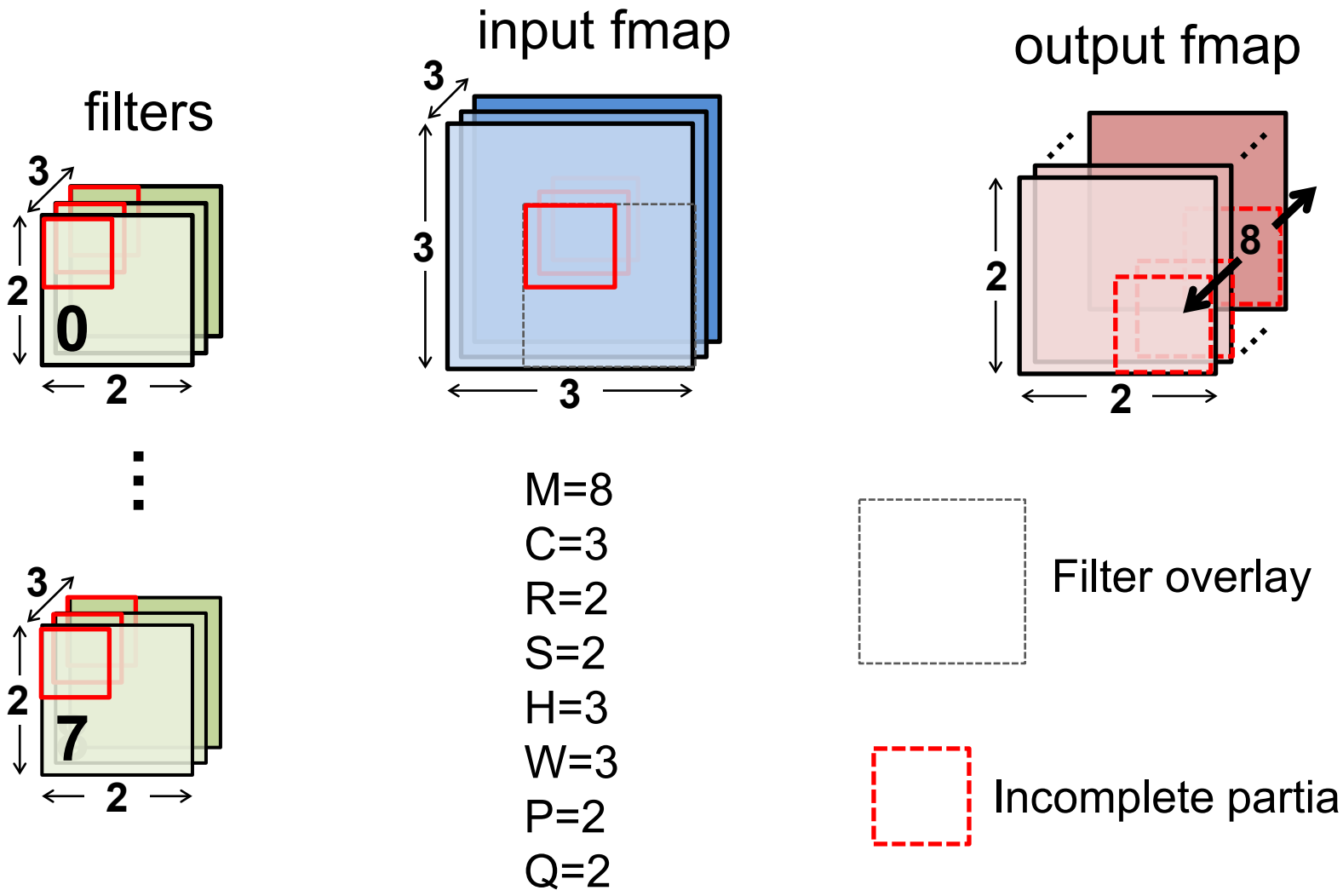
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weight)

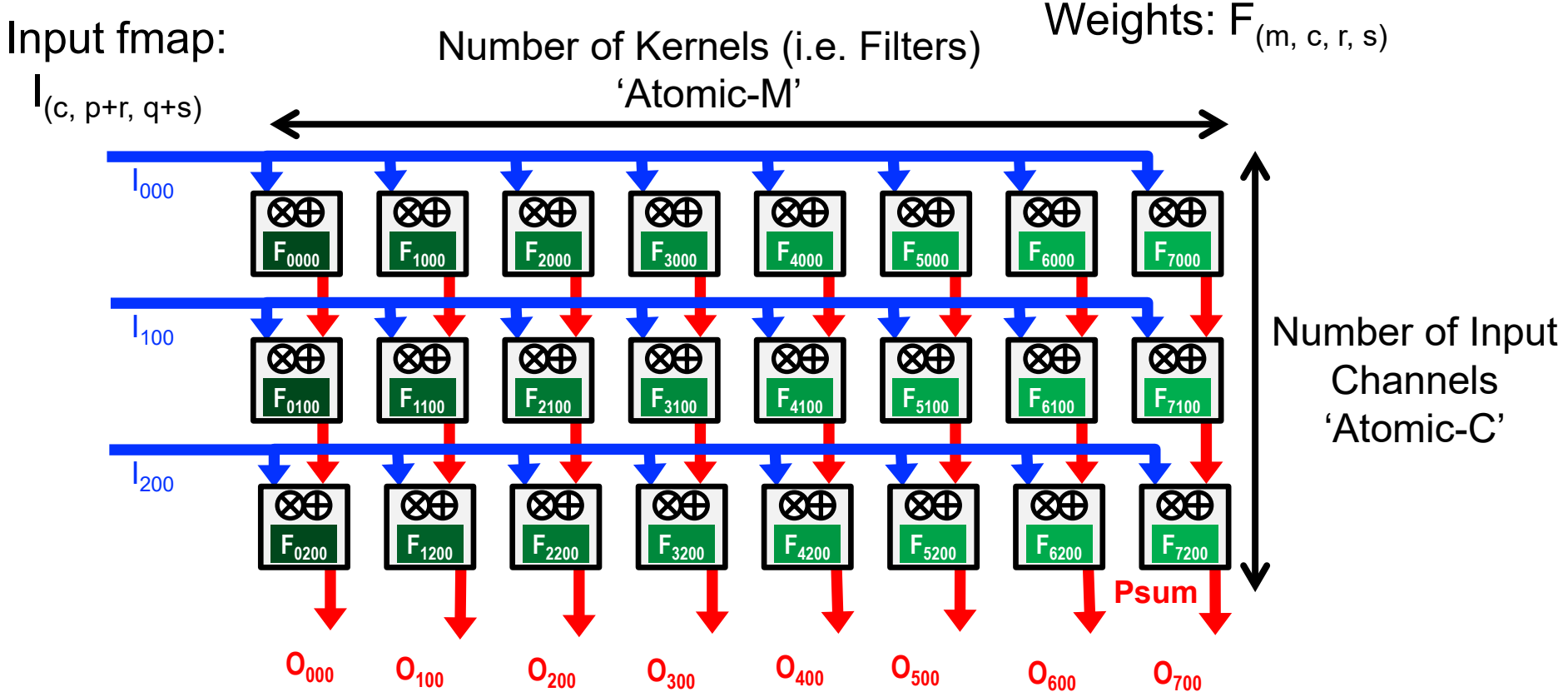


WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)

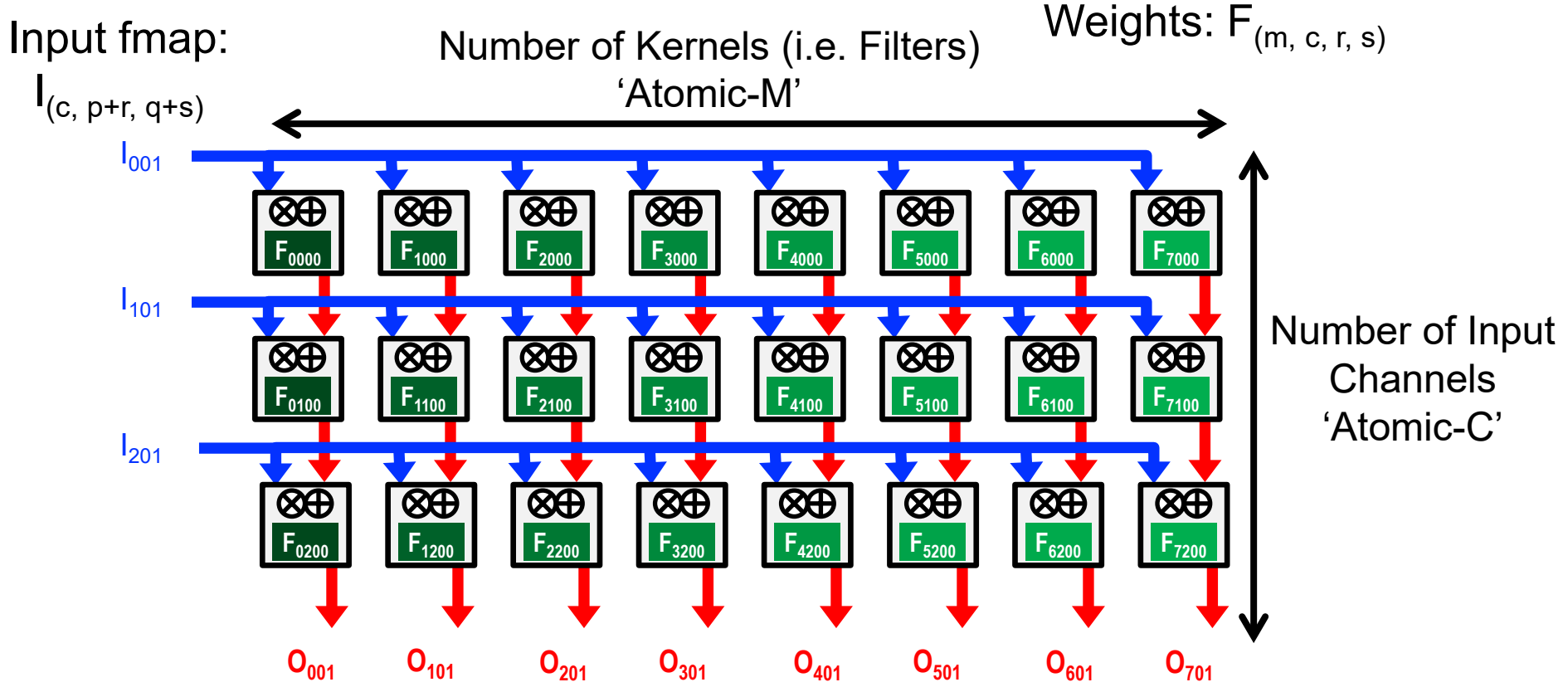


WS Example: NVDLA (simplified)



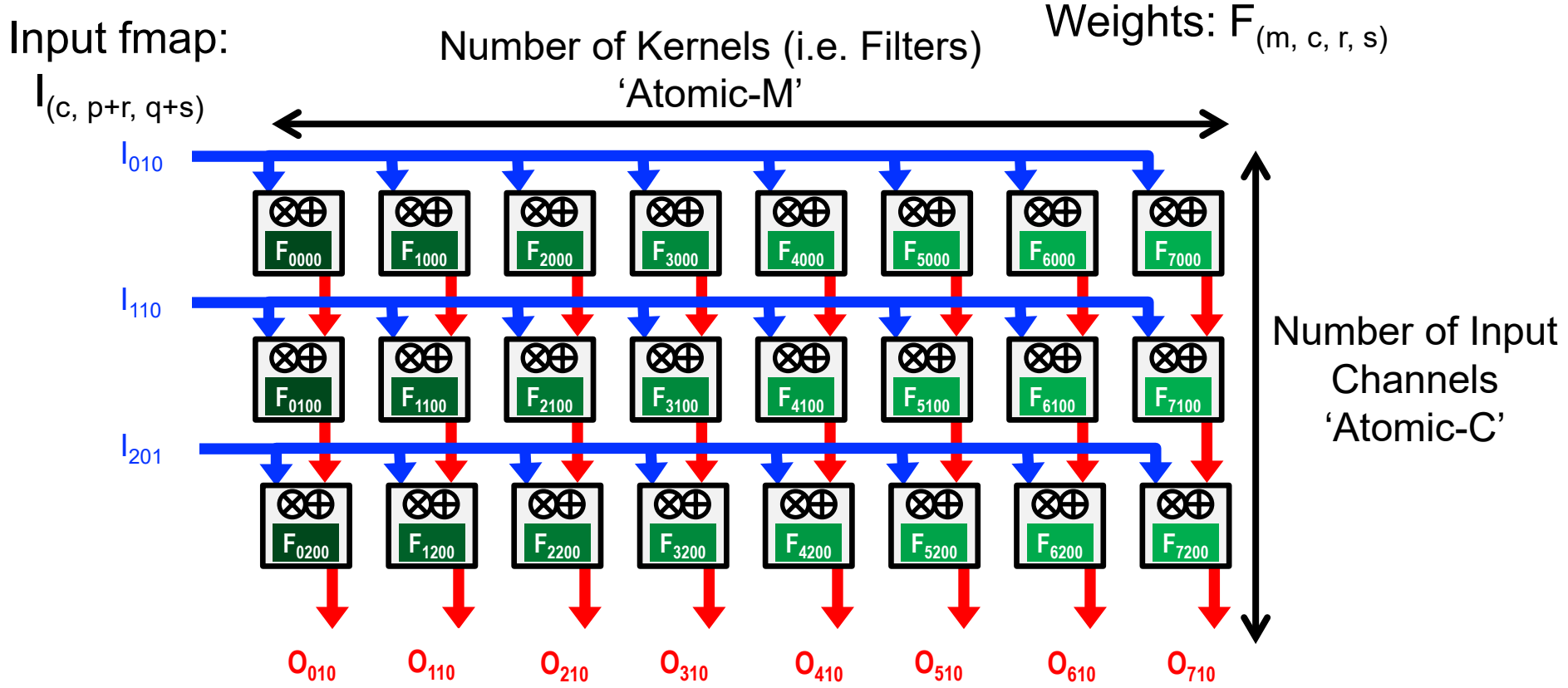
Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)



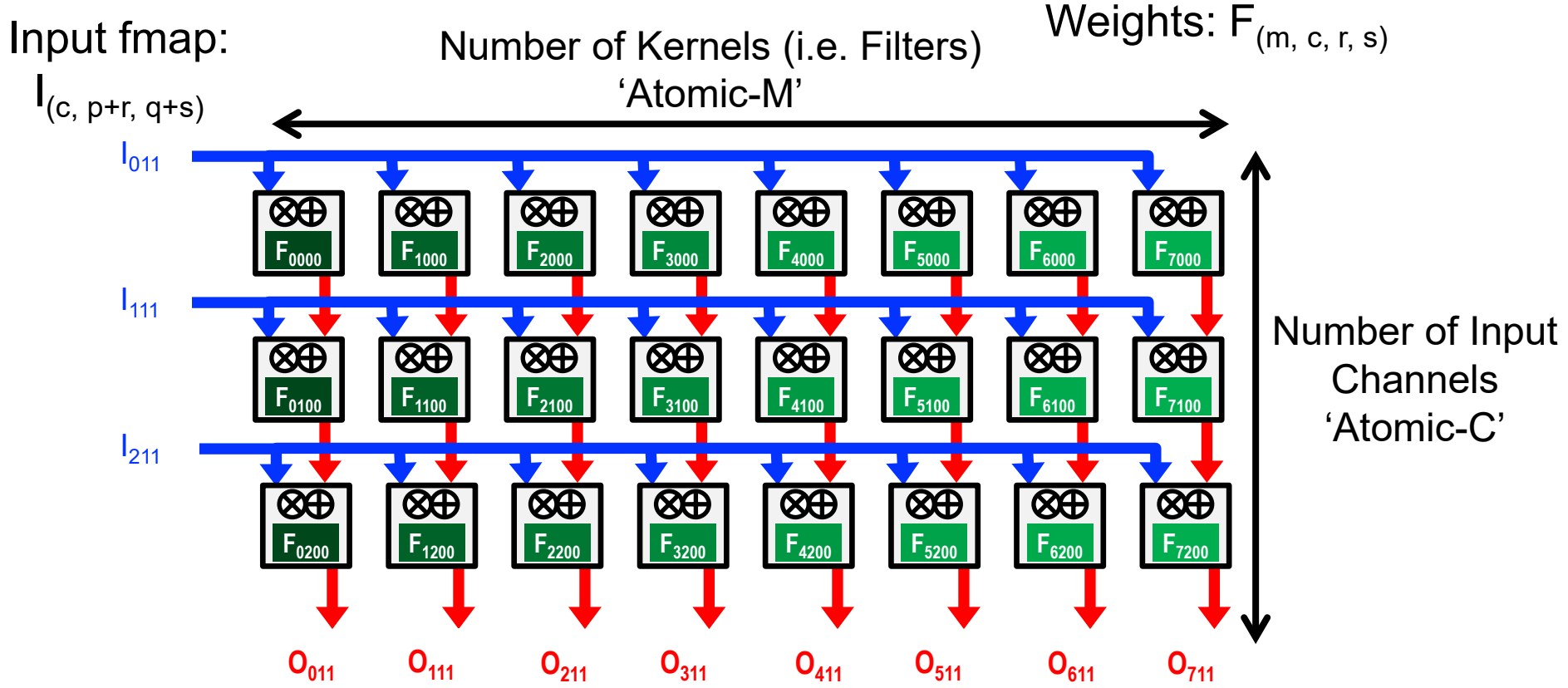
Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)



Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)

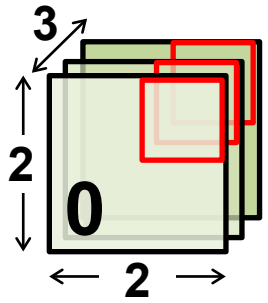


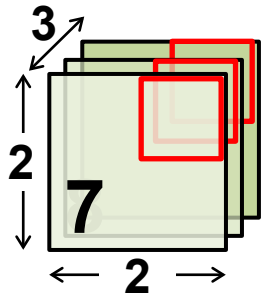
Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)

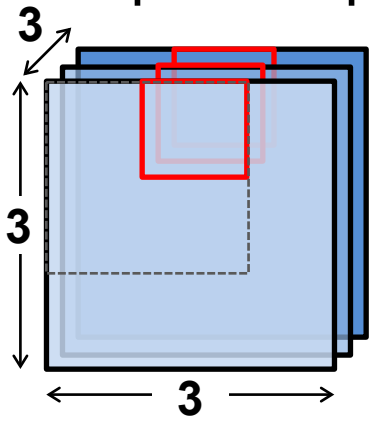
Load new weights

filters

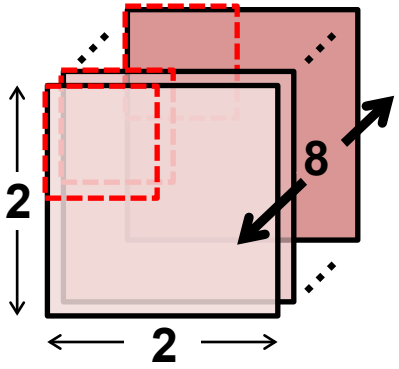




input fmap



output fmap



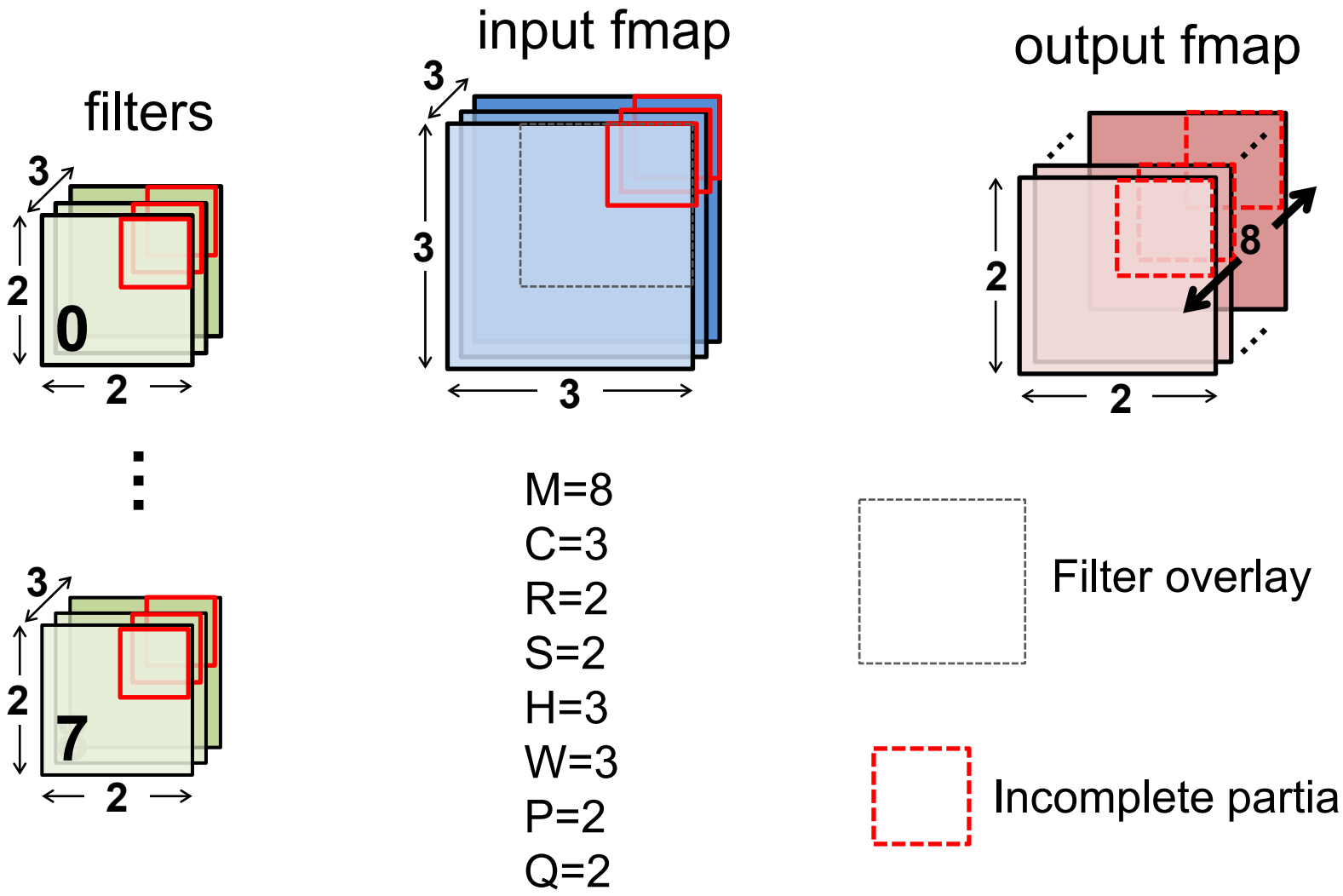
M=8
C=3
R=2
S=2
H=3
W=3
P=2
Q=2

Filter overlay

Incomplete partial sum

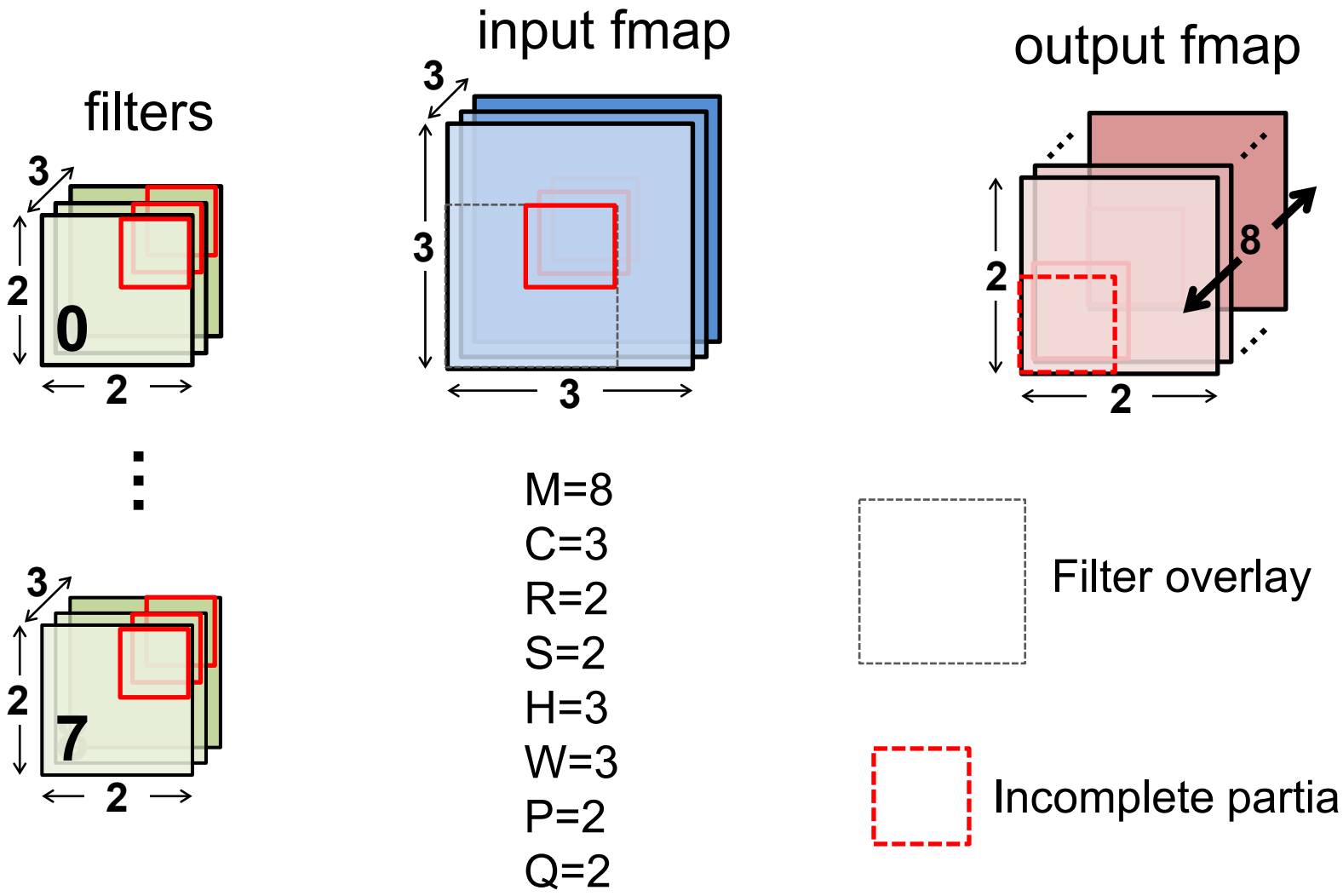
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)



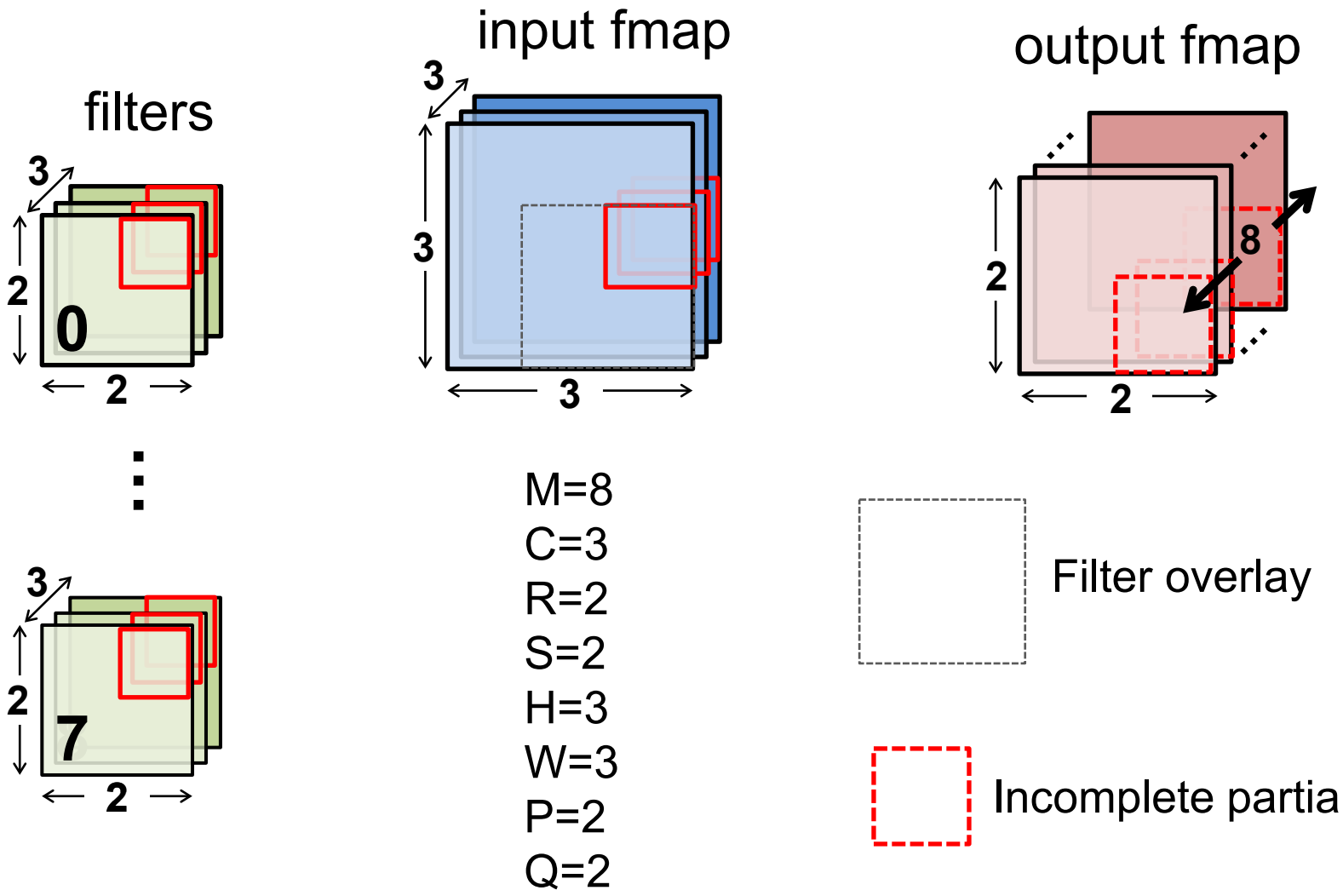
WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)

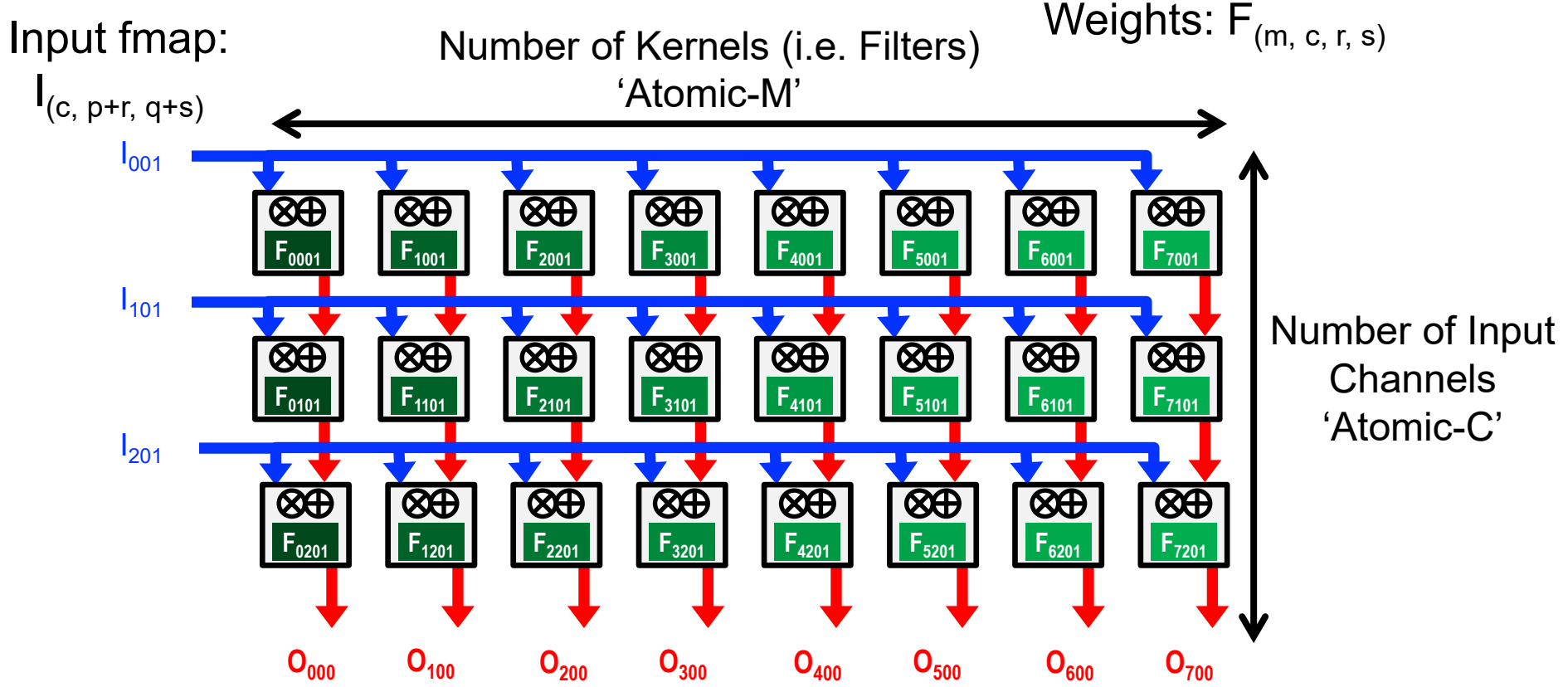


WS Example: NVDLA (simplified)

Cycle through input and output fmap (hold weights)

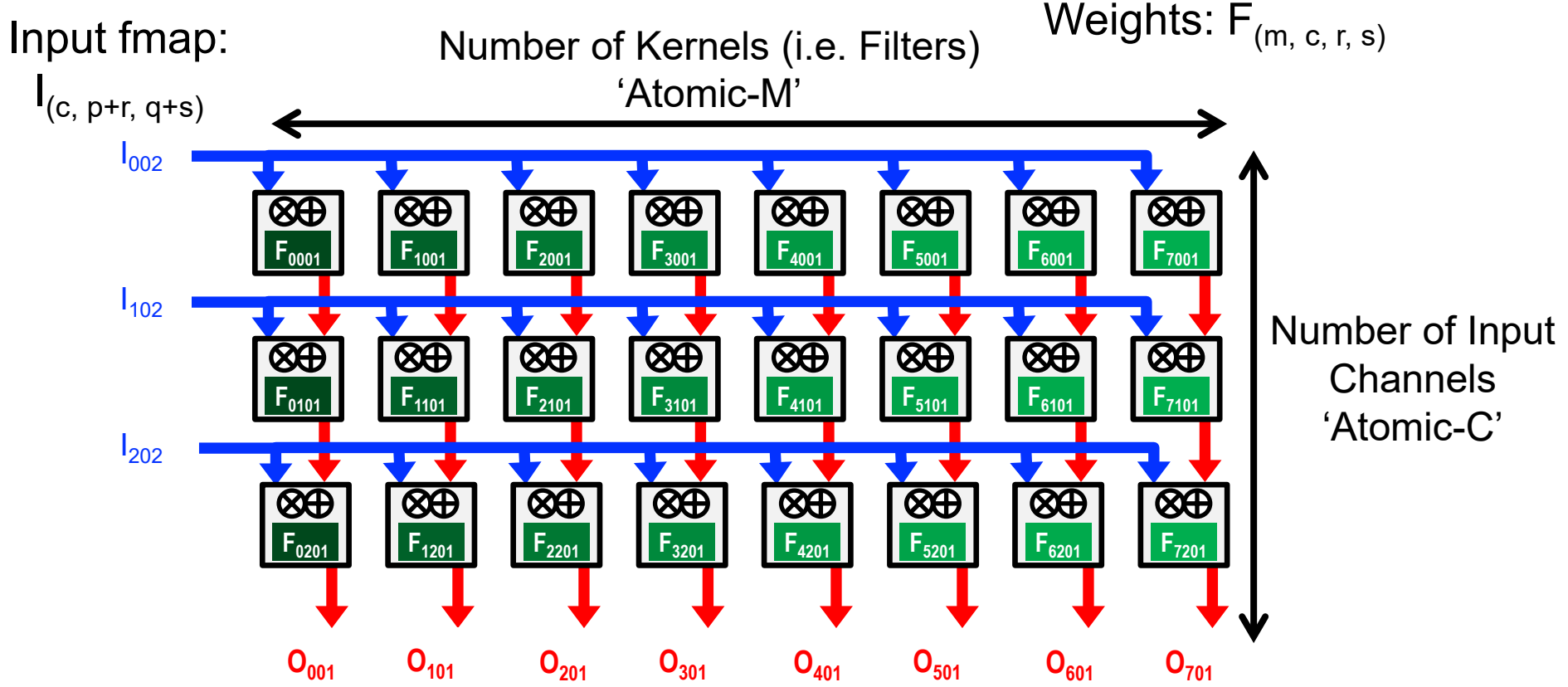


WS Example: NVDLA (simplified)



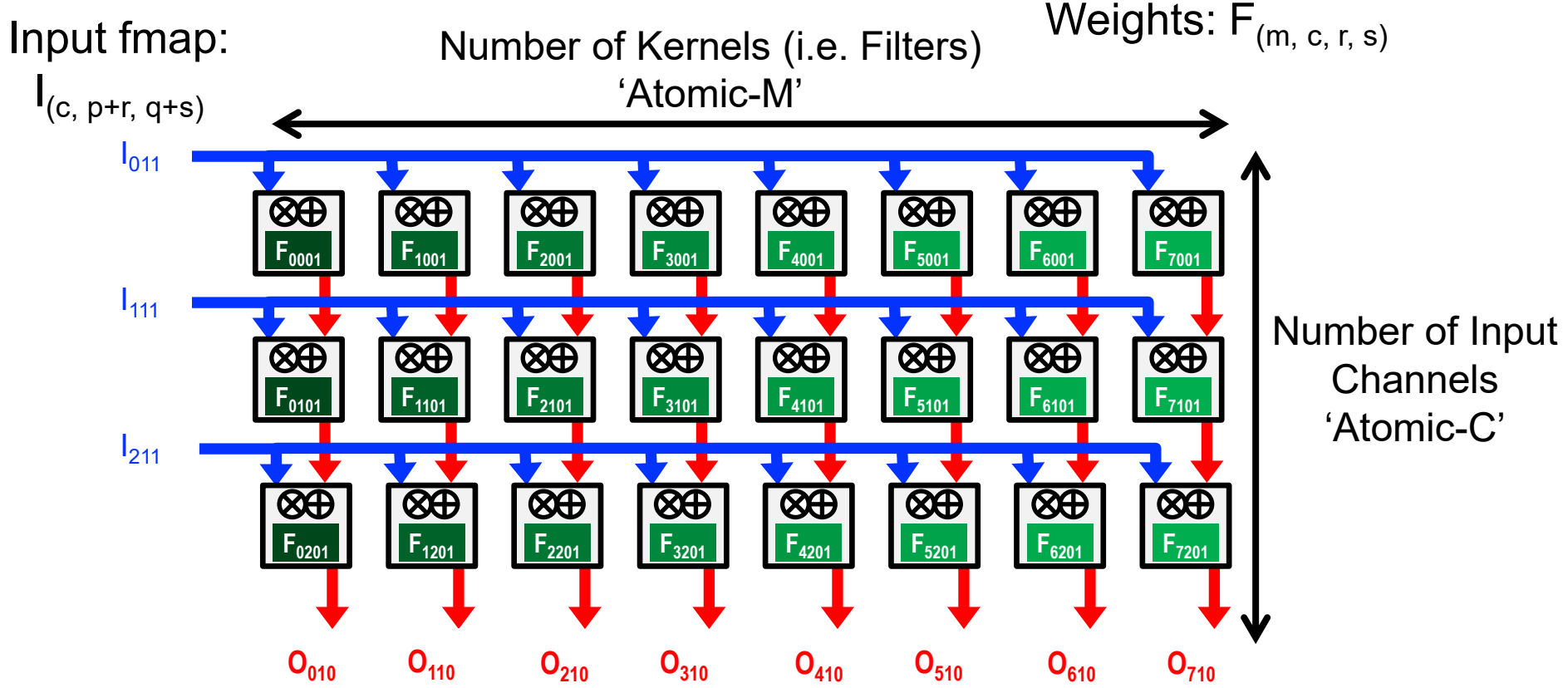
Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)



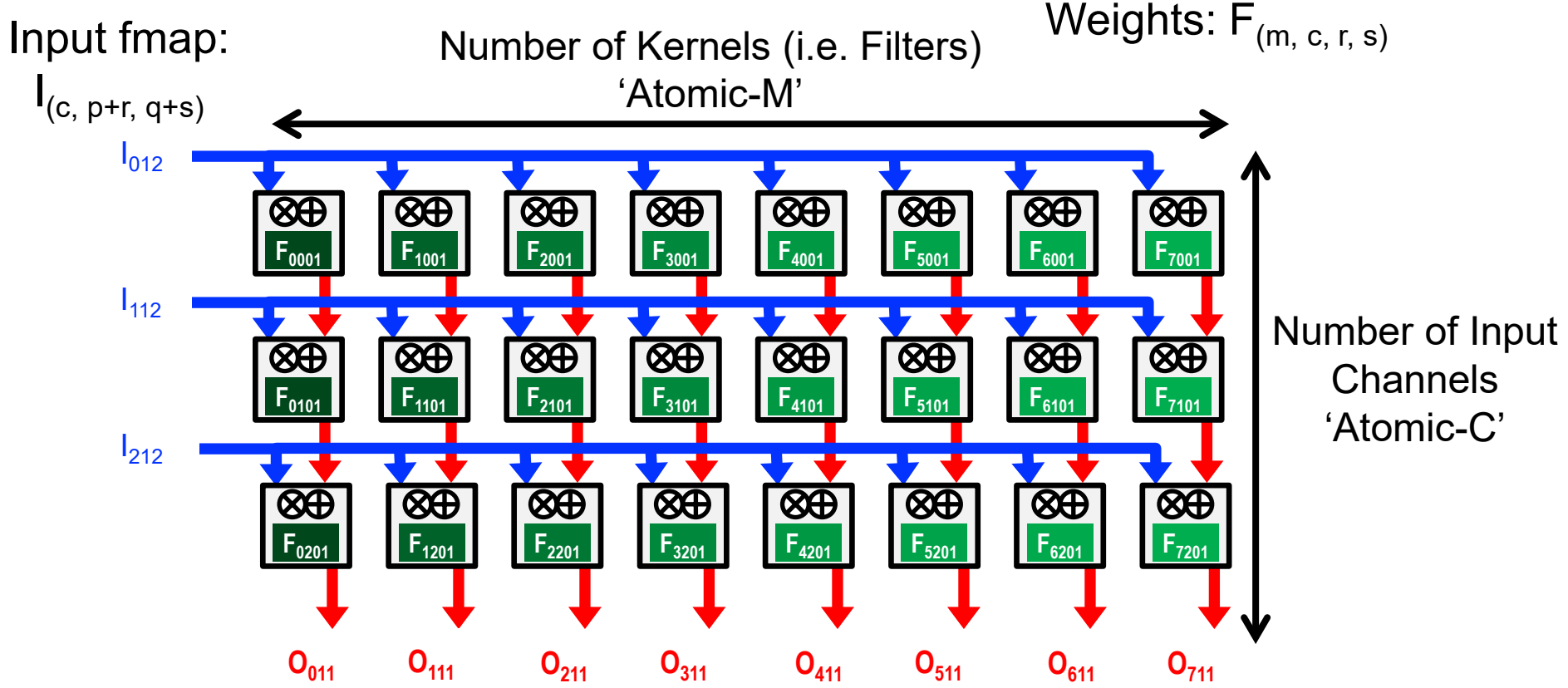
Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)



Output feature map: $O_{(m, p, q)}$

WS Example: NVDLA (simplified)



Output feature map: $O_{(m, p, q)}$

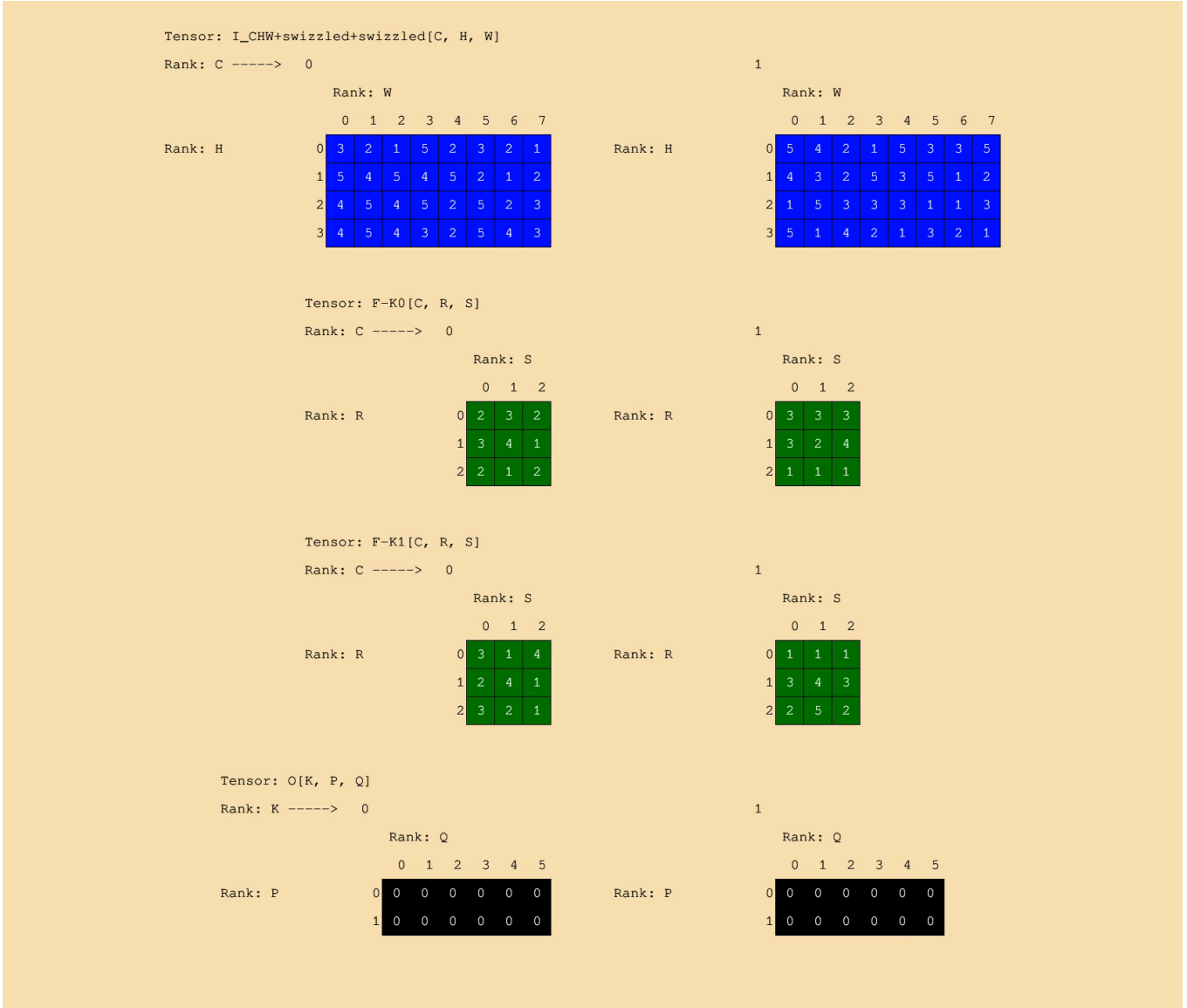
Loop Nest: NVDLA (simplified)

```
M = 8; C = 3;
R = 2; S = 2
P = 2; Q = 2

int i[C,H,W];      # Input activations
int f[M,C,R,S];    # Filter weights
int o[M,P,Q];      # Output activations

for r in [0,R):
  for s in [0,S):
    for p in [0,P):
      for q in [0,Q):
        parallel-for m in [0, M):
          parallel-for c in [0, C):
            o[m,p,q] += i[c,p+r,q+s] * f[m,c,r,s]
```

How can we tell this is weight stationary?

Top loops are r and s

NVDLA (Simplified) - Animation



M = 2
C = 2
R = 3
S = 3
H = 4
W = 8
P = 2
Q = 6

CONV-layer Einsum

$$O_{m,p,q} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

Traversal order (fastest to slowest):

Parallel Ranks:

WS Example: NVDLA (simplified)

Global Buffer

Released Sept 29, 2017

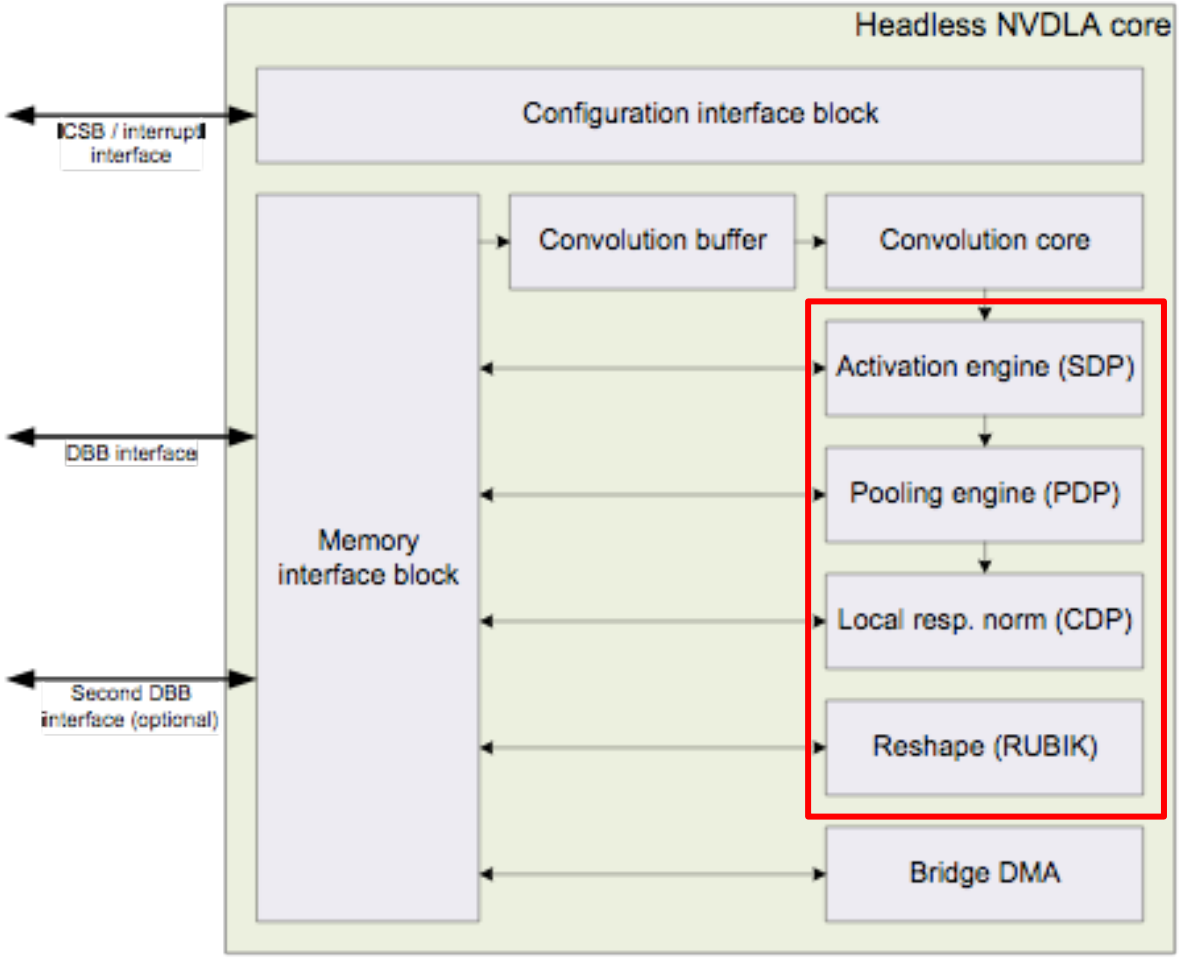


Image Source: Nvidia

http://nvdla.org

WS Example: Nvidia DLA

- Single Data Point Operations
  - Immediately after CNN
  - Linear functions
    - e.g. bias addition, precision scaling (before writing to memory), batch normalization, element-wise operation
    - Scaling: all values, per channel, per pixel
  - Non-linear functions use LUT
    - e.g. ReLU, PReLU, sigmod, tanh
- Planar Data Operations
  - For pooling: Max, min, average
- Multi-Plane Operation
  - Cross-channel processing for LRN

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]

What other dataflow exists?

Input Stationary

Input Stationary

- Hold inputs stationary rather than outputs or weights
- Used for sparse CNNs *[Parashar et al., SCNN, ISCA 2017]*
  - Sparse CNN is where many weights are zeros
  - Advantage is the inputs are larger than weights thus require larger memory

    → reduce reads to larger memory
- Not analyzed for dense

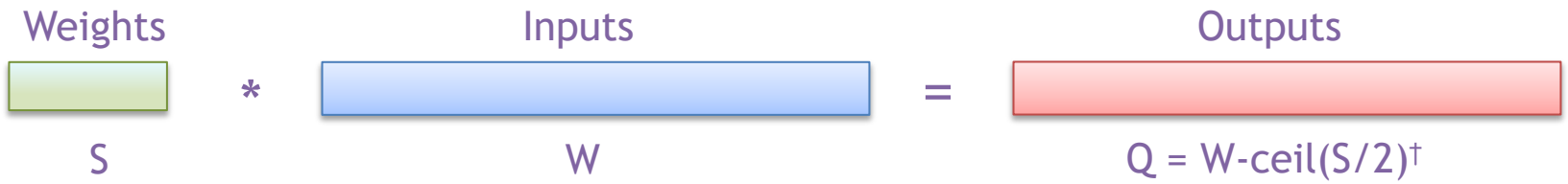
1-D Convolution – Output Stationary


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q + s
        o[q] += i[w]*f[s]
```

How can we implement input stationary with no input index?

† Assuming: 'valid' style convolution

1-D Convolution – Input Stationary




```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for w in [0, W):
      for s in [0, S):
          q = w - s
          o[q] += i[w]*f[s]
```

Beware q must be >= 0 and <Q

† Assuming: 'valid' style convolution

1-D Convolution Einsum + IS

$$O_q = I_{q+s} \times F_s$$

$$O_q^Q = I_{q+s}^W \times F_s^S$$

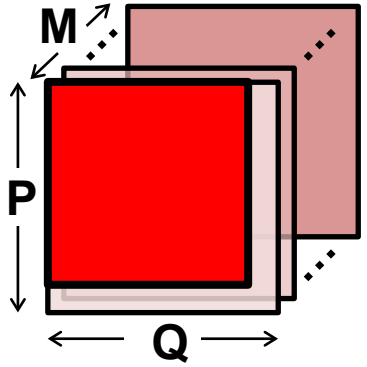
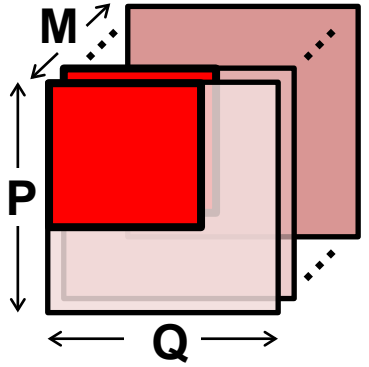
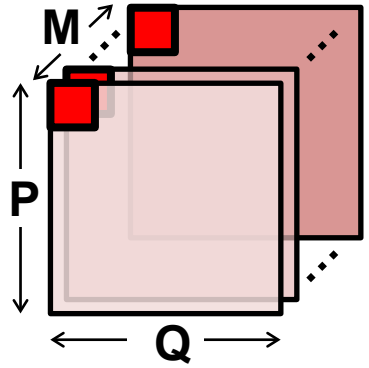
Note: w = q+s, so q = w-s, and therefore

$$O_{w-s}^Q = I_w^W \times F_s^S$$

Traversal order (fastest to slowest): S, W

Output Stationary (revisited)

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 tilings

$$O_{m,p,q} = I_{c,p+r,q+s,} \times F_{m,c,r,s}$$

OS_A

$$O_{m,p1,q1,p0,q0} = I_{c,p1,p1,p0+r,q0+s,} \times F_{m,c,r,s}$$

Parallel: P0, Q0

OS_B

$$O_{m1,m0,p1,q1,p0,q0} = I_{c,p1,p1,p0+r,q0+s,} \times F_{m1,m0,c,r,s}$$

Parallel: M0, P0, Q0

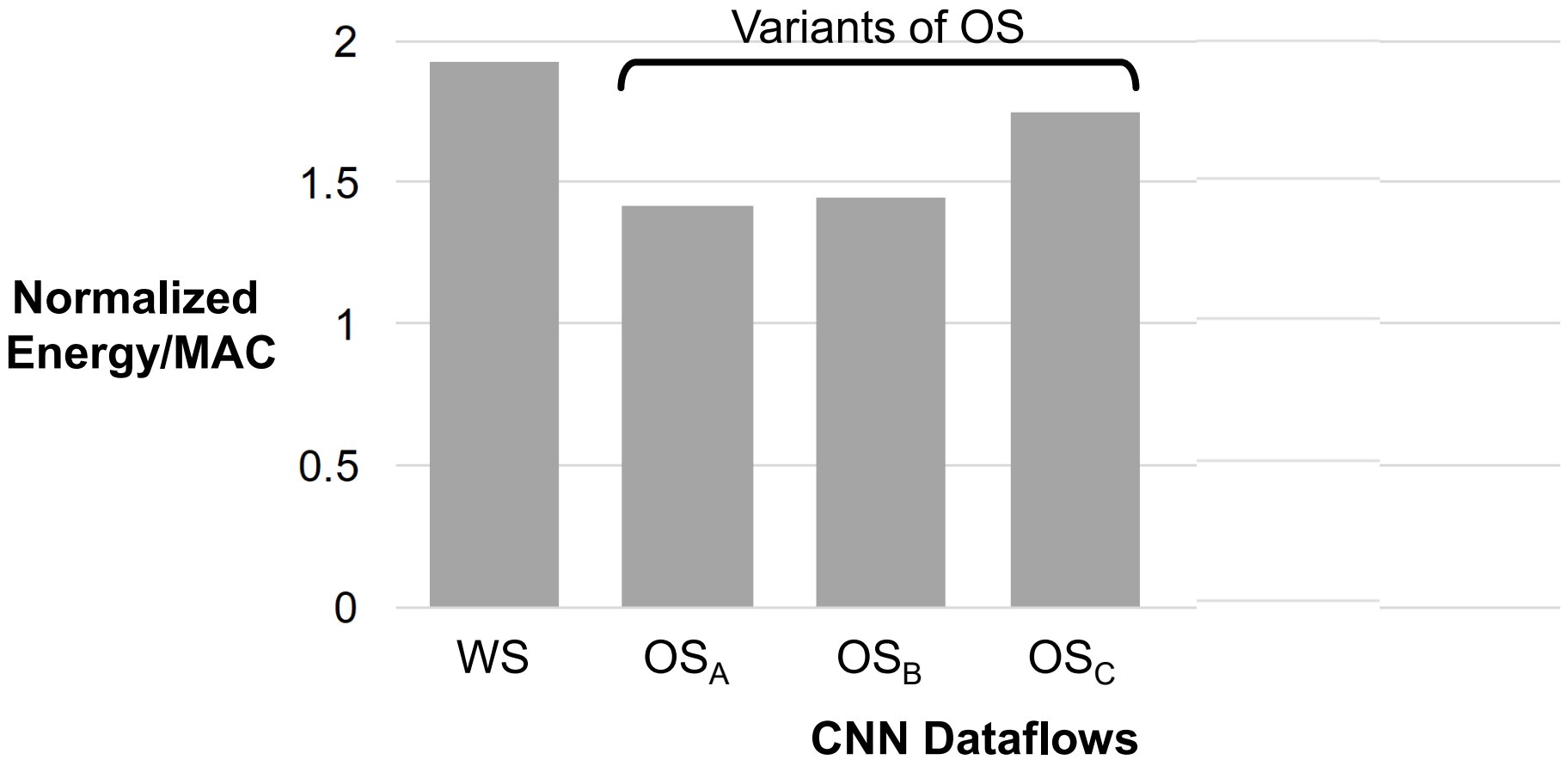
OS_C

$$O_{m1,m0,p,q} = I_{c,p+r,q+s,} \times F_{m1,m0,c,r,s}$$

Parallel: M0

Energy Efficiency Comparison

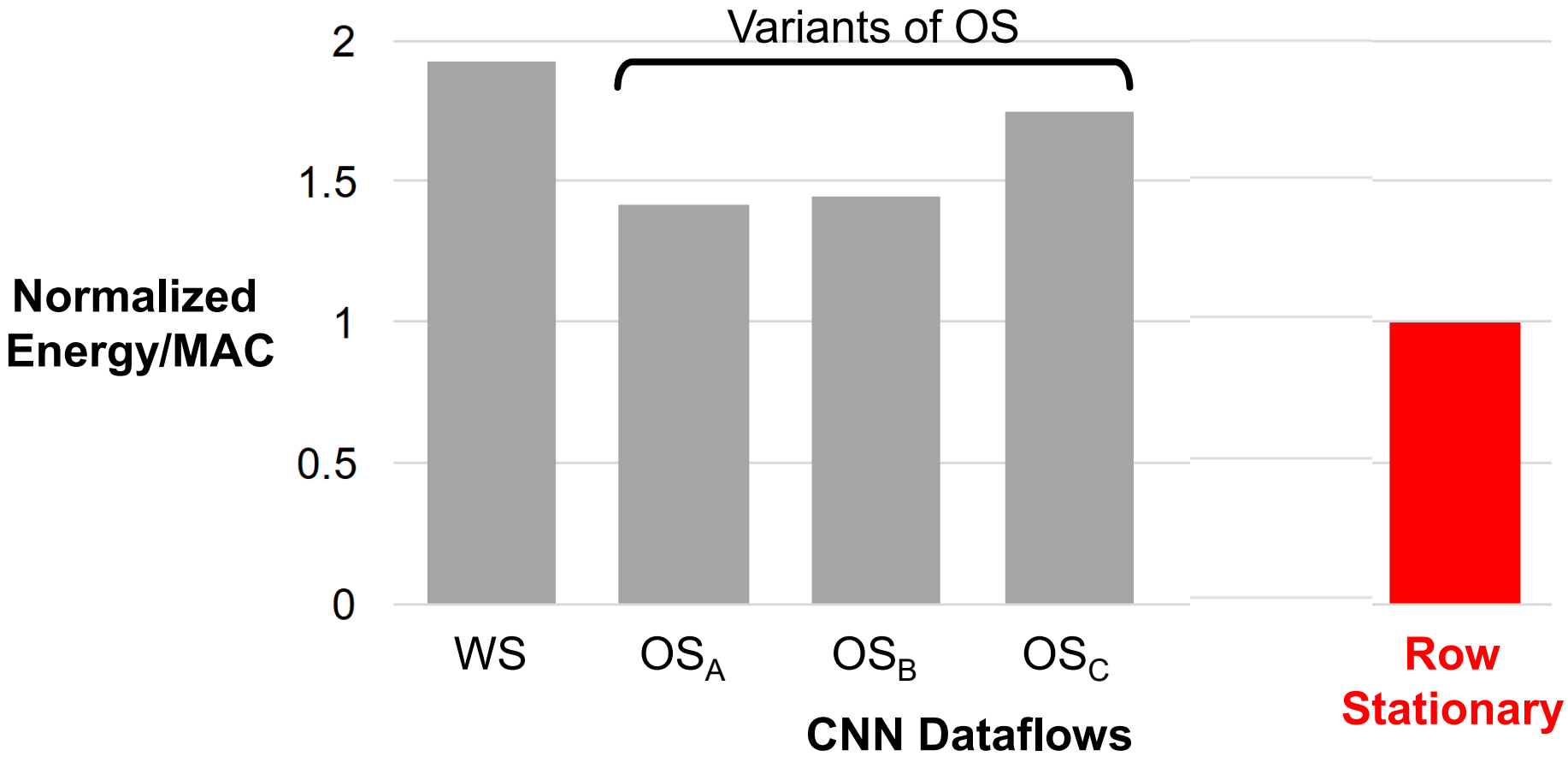
- Same total area
- AlexNet CONV layers
- 256 PEs
- Batch size = 16



[Chen et al., ISCA 2016]

Energy Efficiency Comparison

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



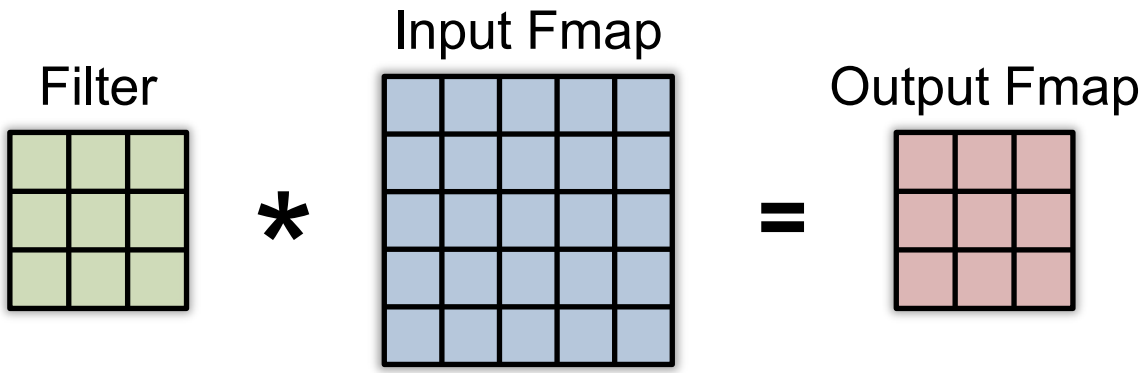
[Chen et al., ISCA 2016]

Energy-Efficient Dataflow: Row Stationary (RS)

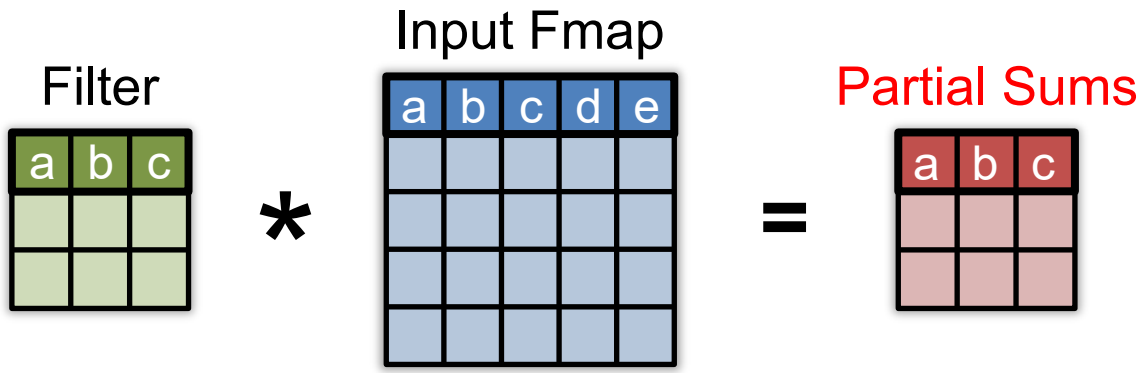
- **Maximize** reuse and accumulation at **RF**
- Optimize for **overall** energy efficiency instead for *only* a certain data type

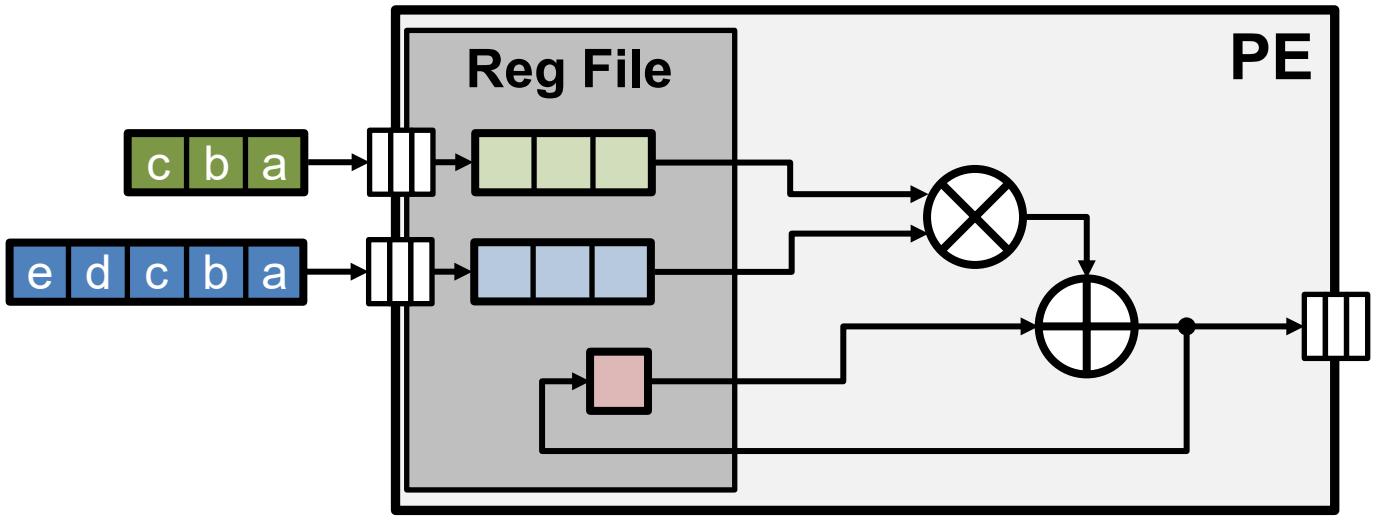
[Chen et al., ISCA 2016]

Row Stationary: Energy-efficient Dataflow

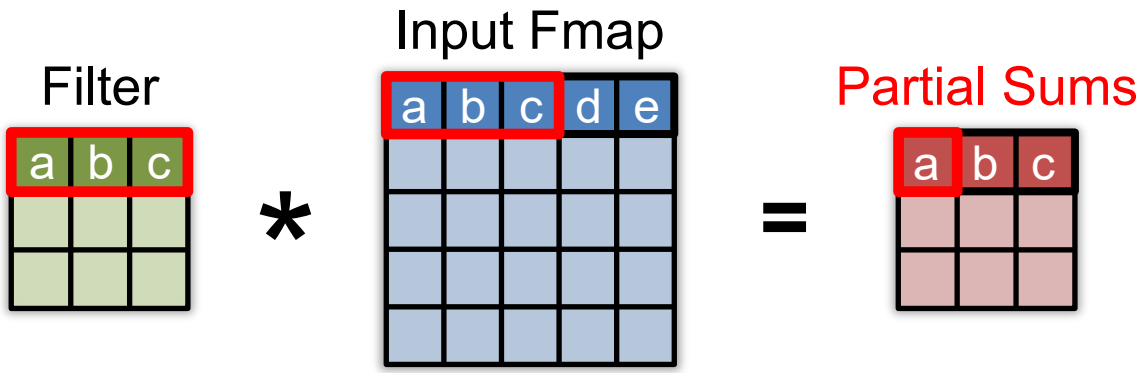


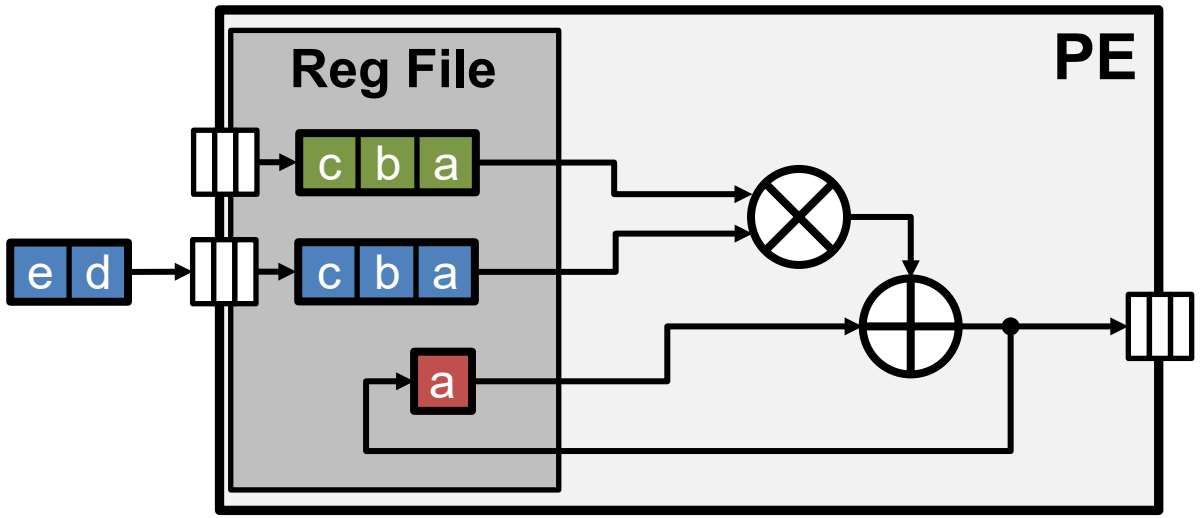
1D Row Convolution in PE



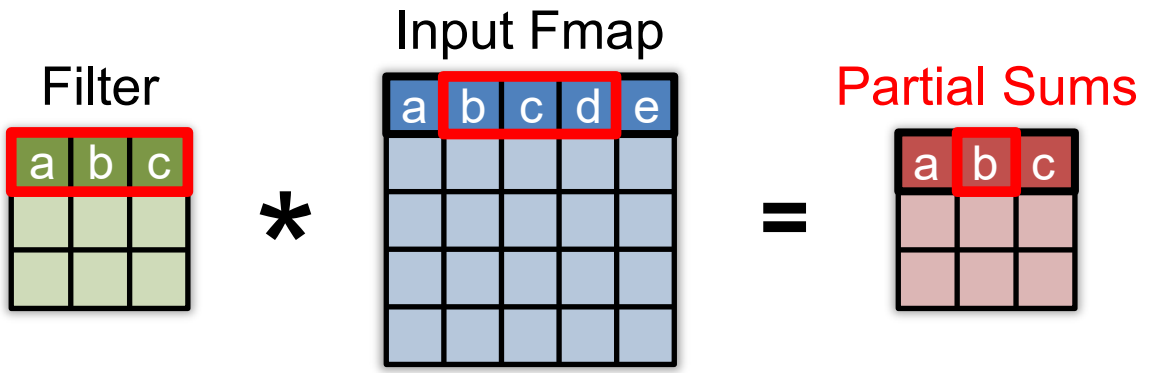


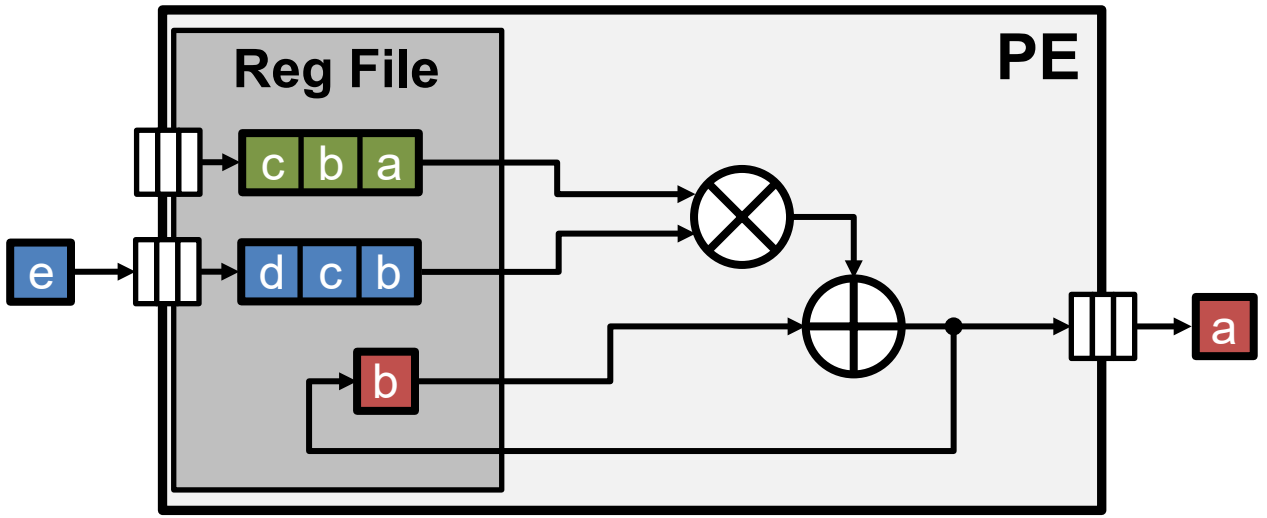
1D Row Convolution in PE



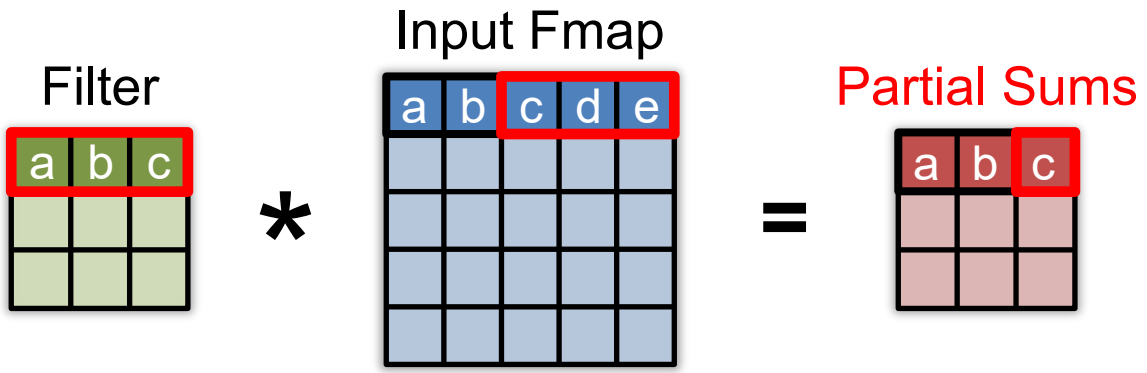


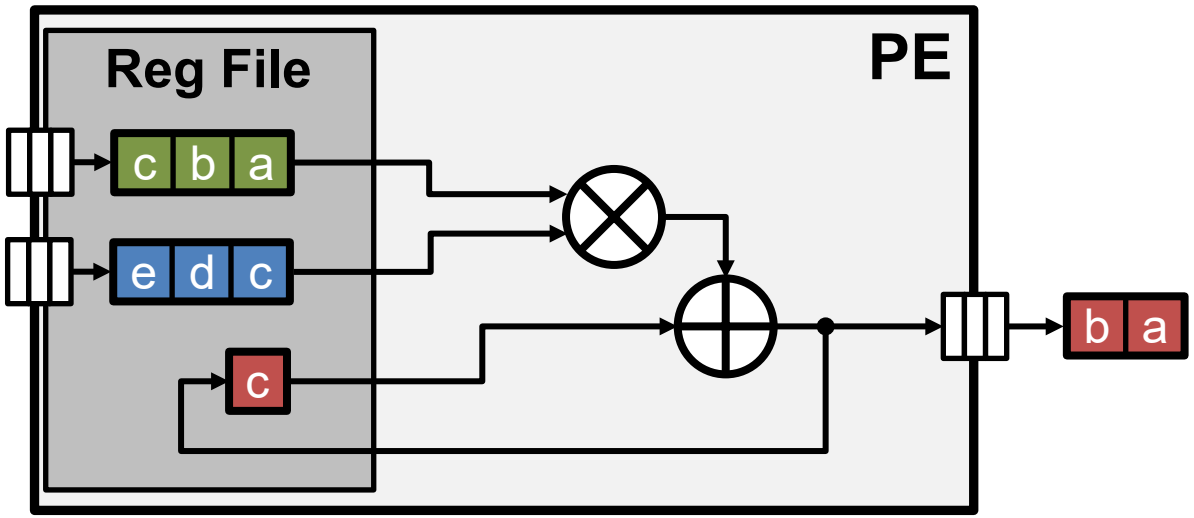
1D Row Convolution in PE





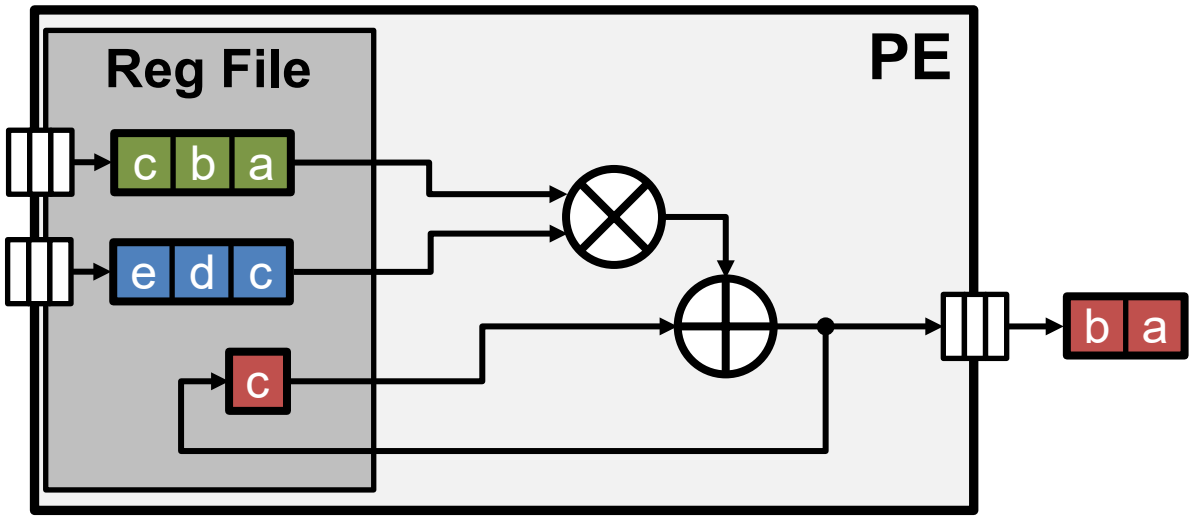
1D Row Convolution in PE



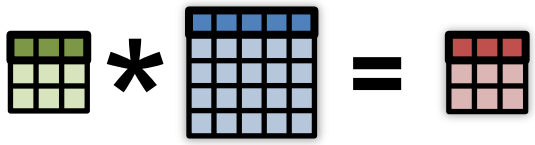


1D Row Convolution in PE

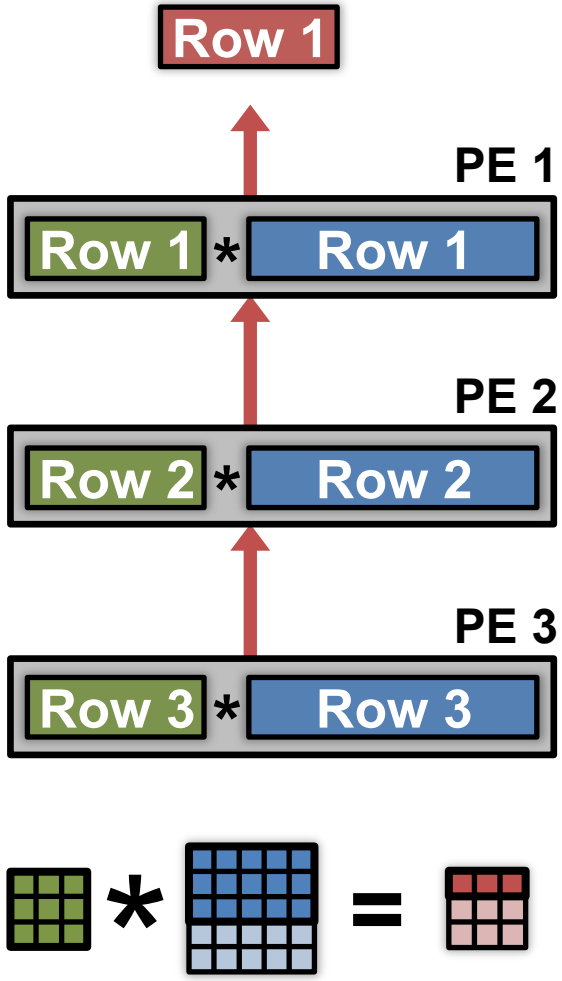
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF



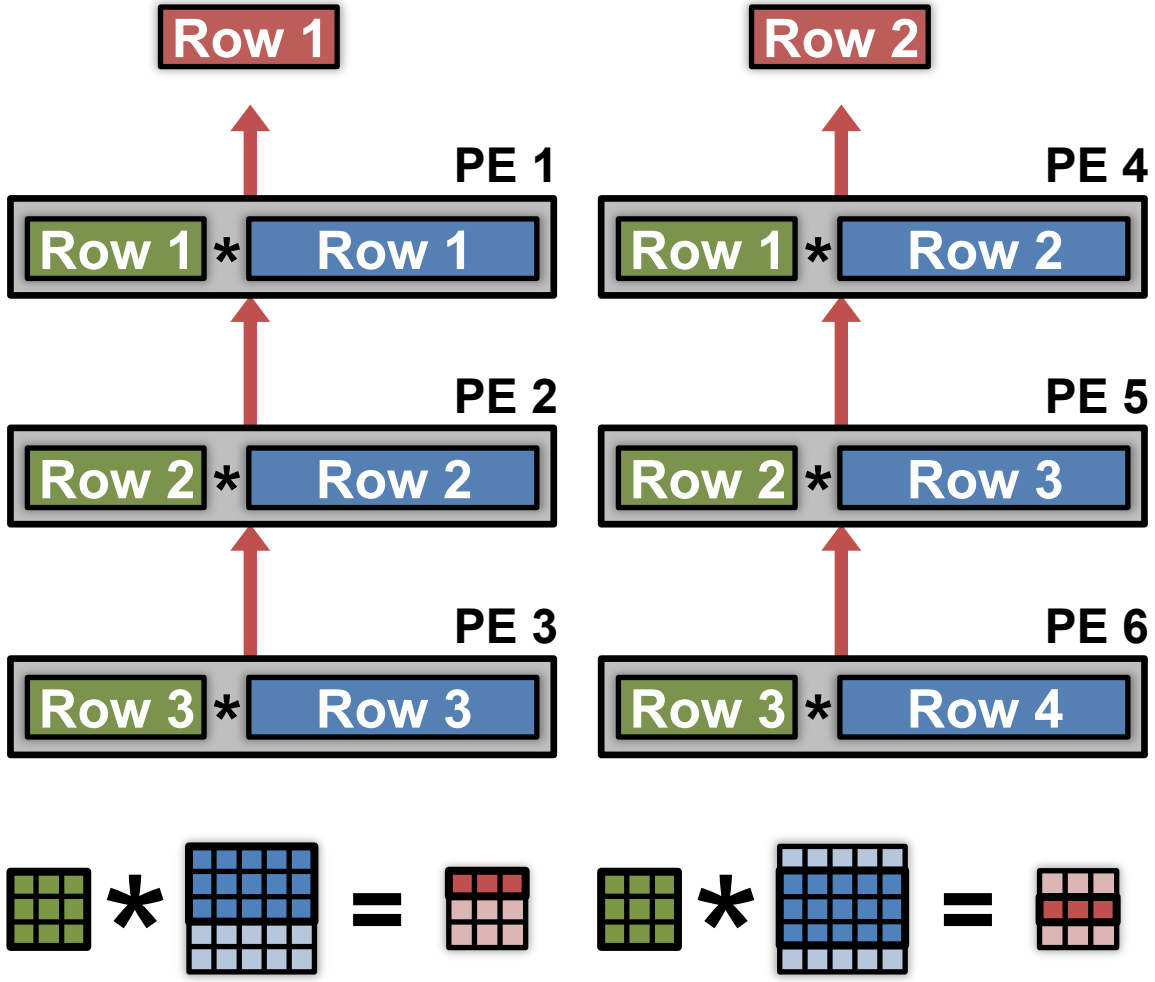
2D Convolution in PE Array



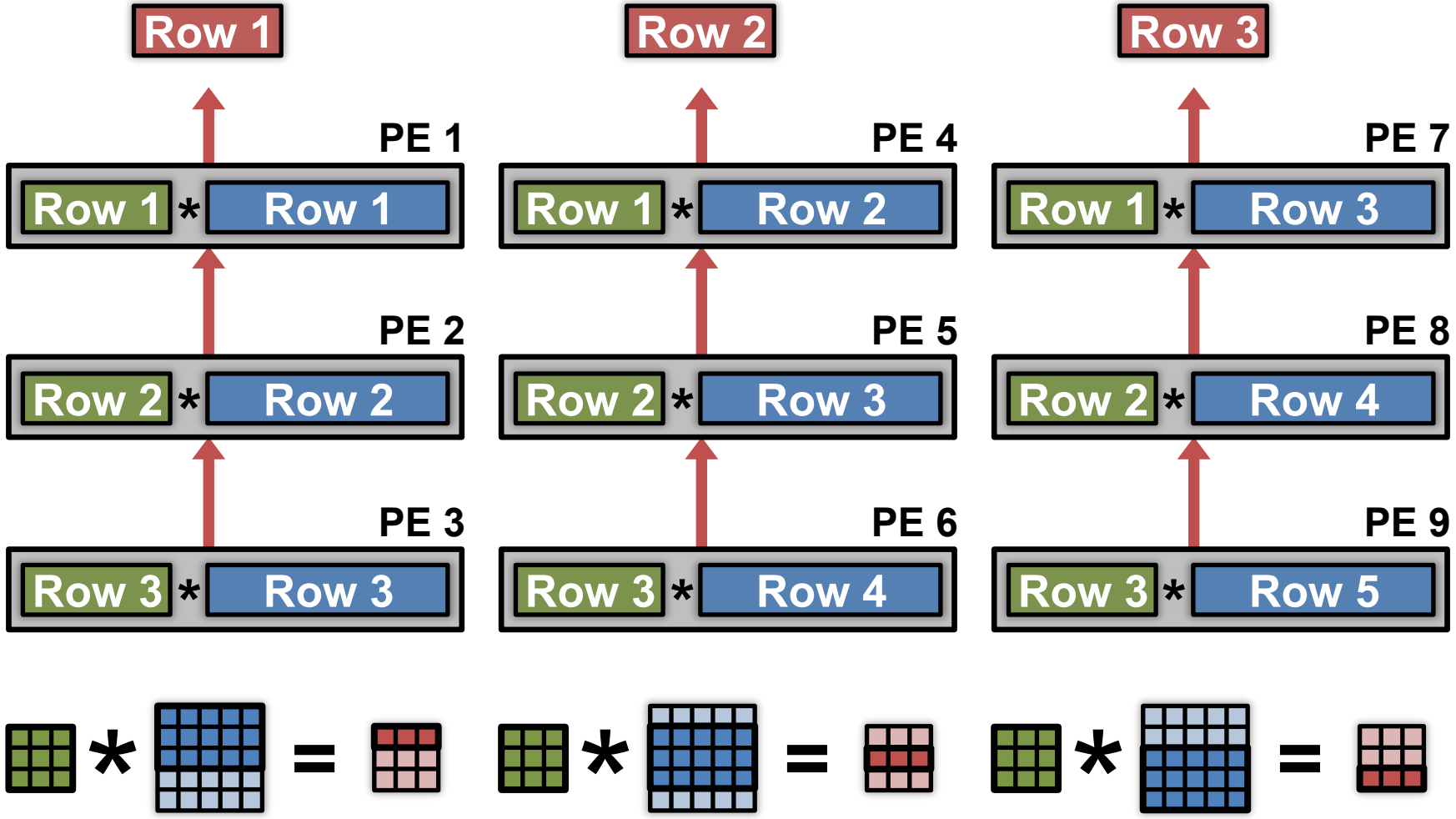
2D Convolution in PE Array



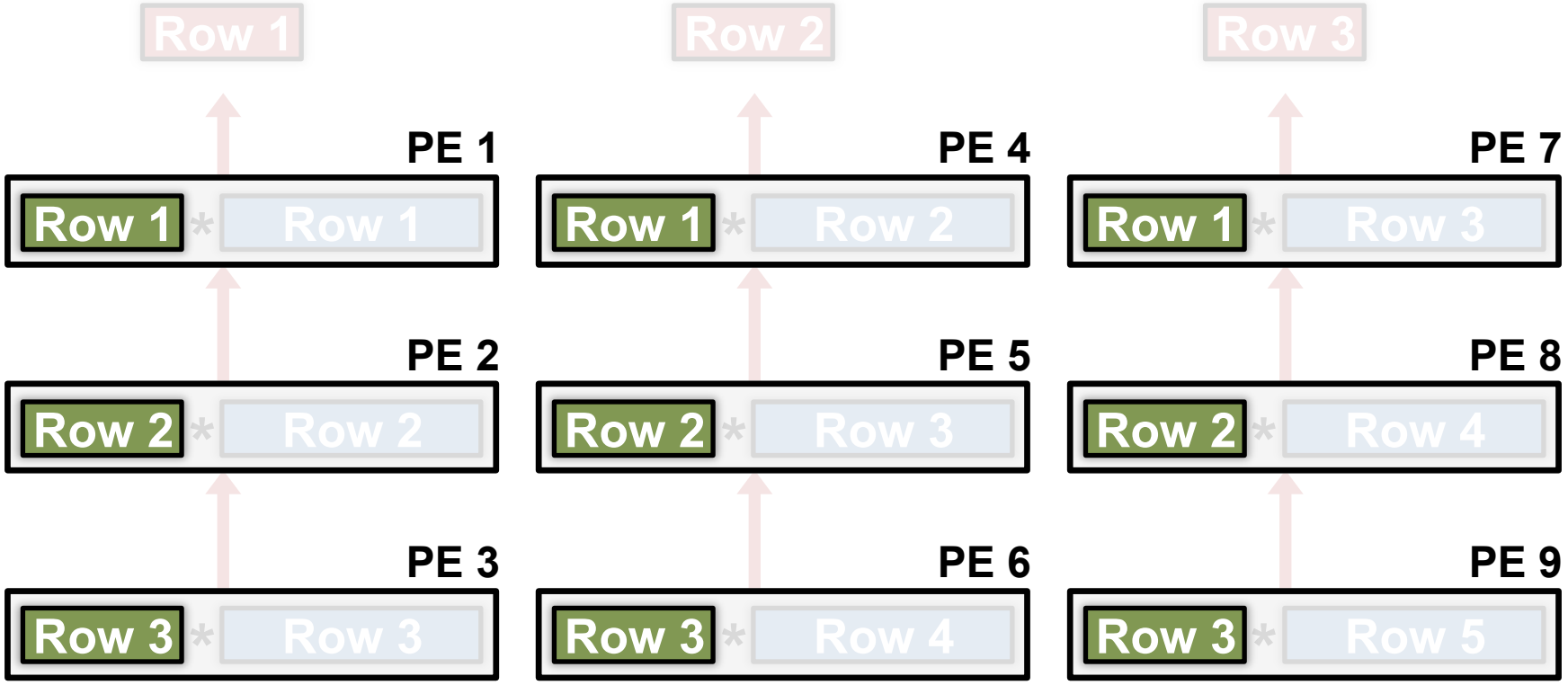
2D Convolution in PE Array



2D Convolution in PE Array

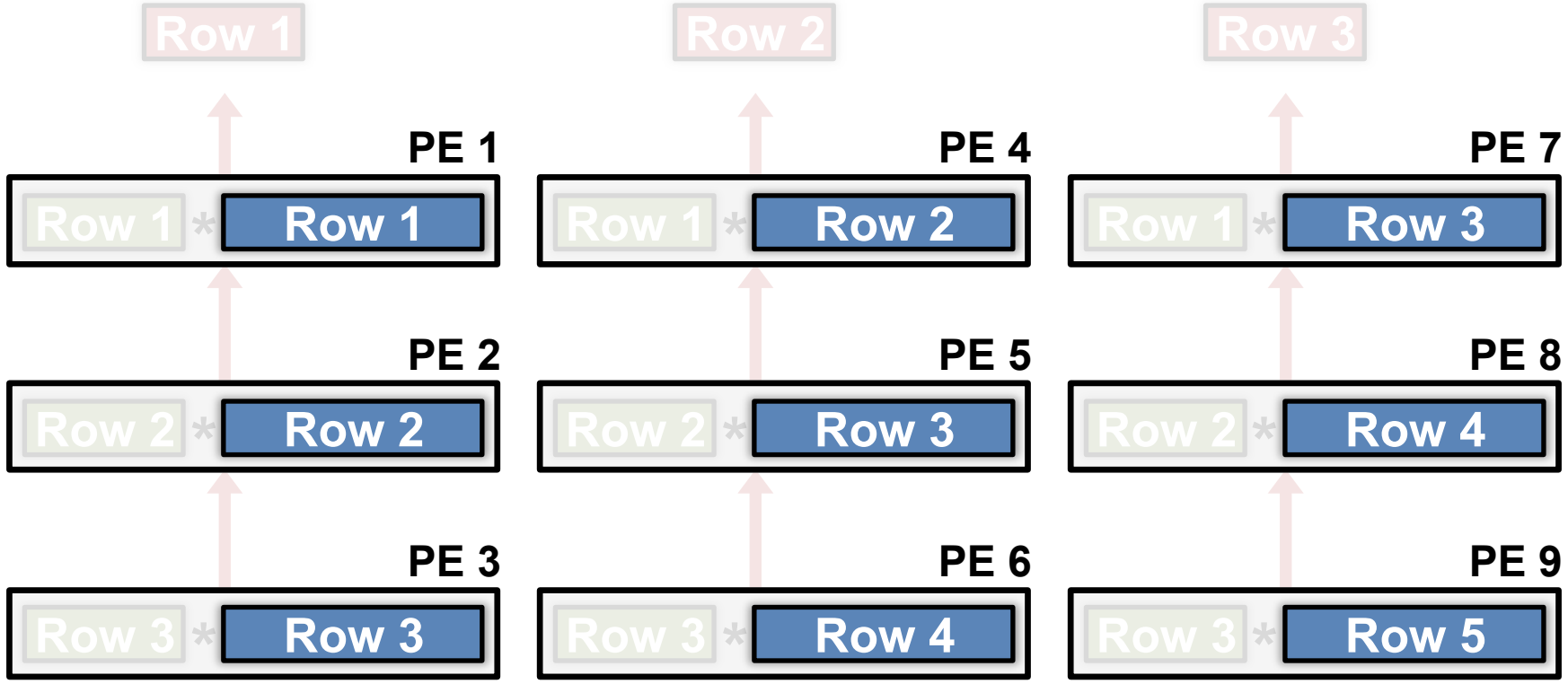


Convolutional Reuse Maximized



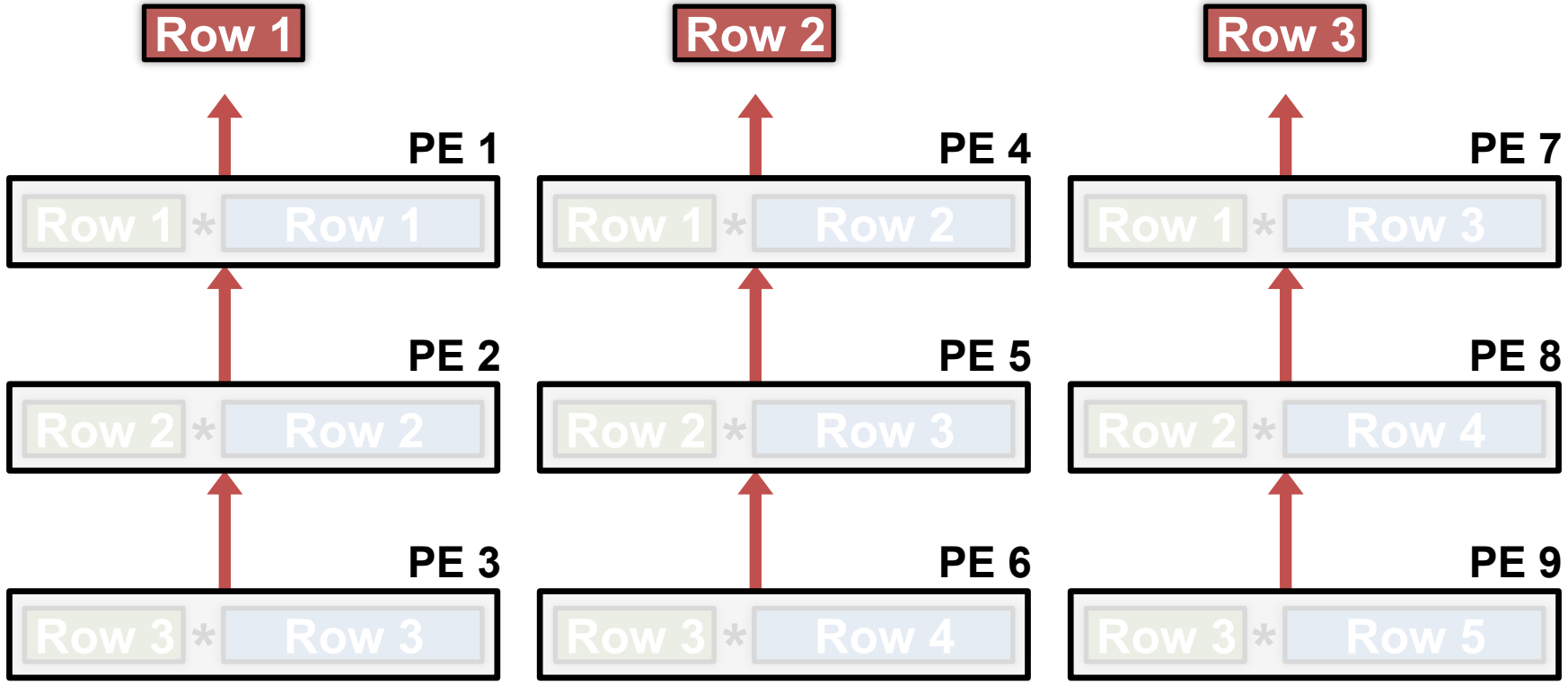
Filter rows are reused across PEs **horizontally**

Convolutional Reuse Maximized



Fmap rows are reused across PEs **diagonally**

Maximize 2D Accumulation in PE Array



Partial sums accumulate across PEs **vertically**

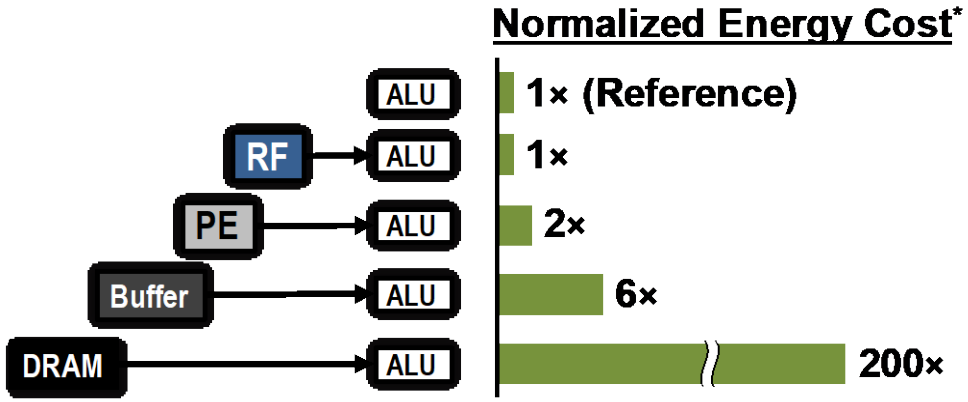
Dataflow Simulation Results

Evaluate Reuse in Different Dataflows

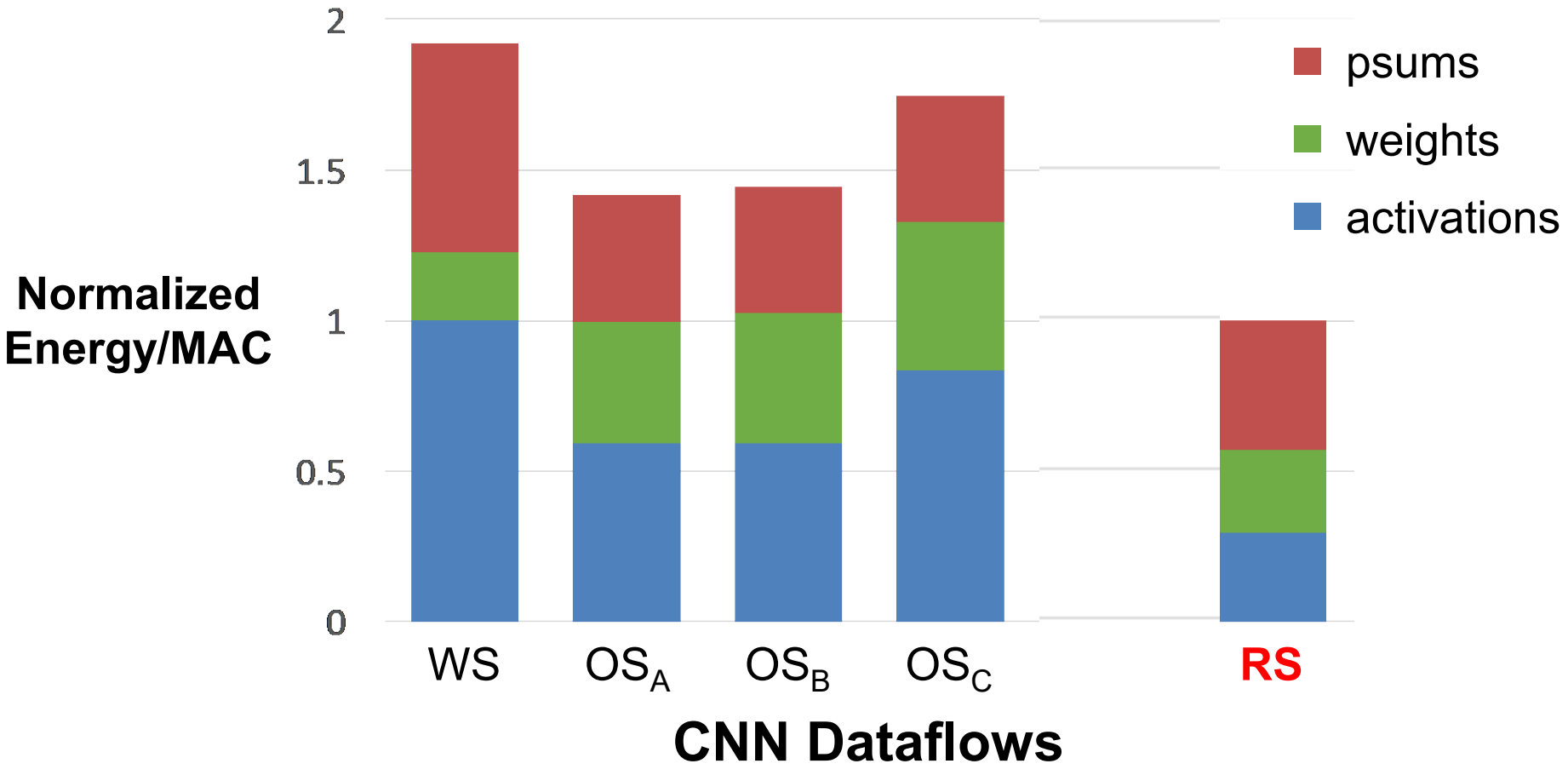
- **Weight Stationary**
  - Minimize movement of filter weights
- **Output Stationary**
  - Minimize movement of partial sums
- **No Local Reuse**
  - No PE local storage. Maximize global buffer size.
- **Row Stationary**

Evaluation Setup

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



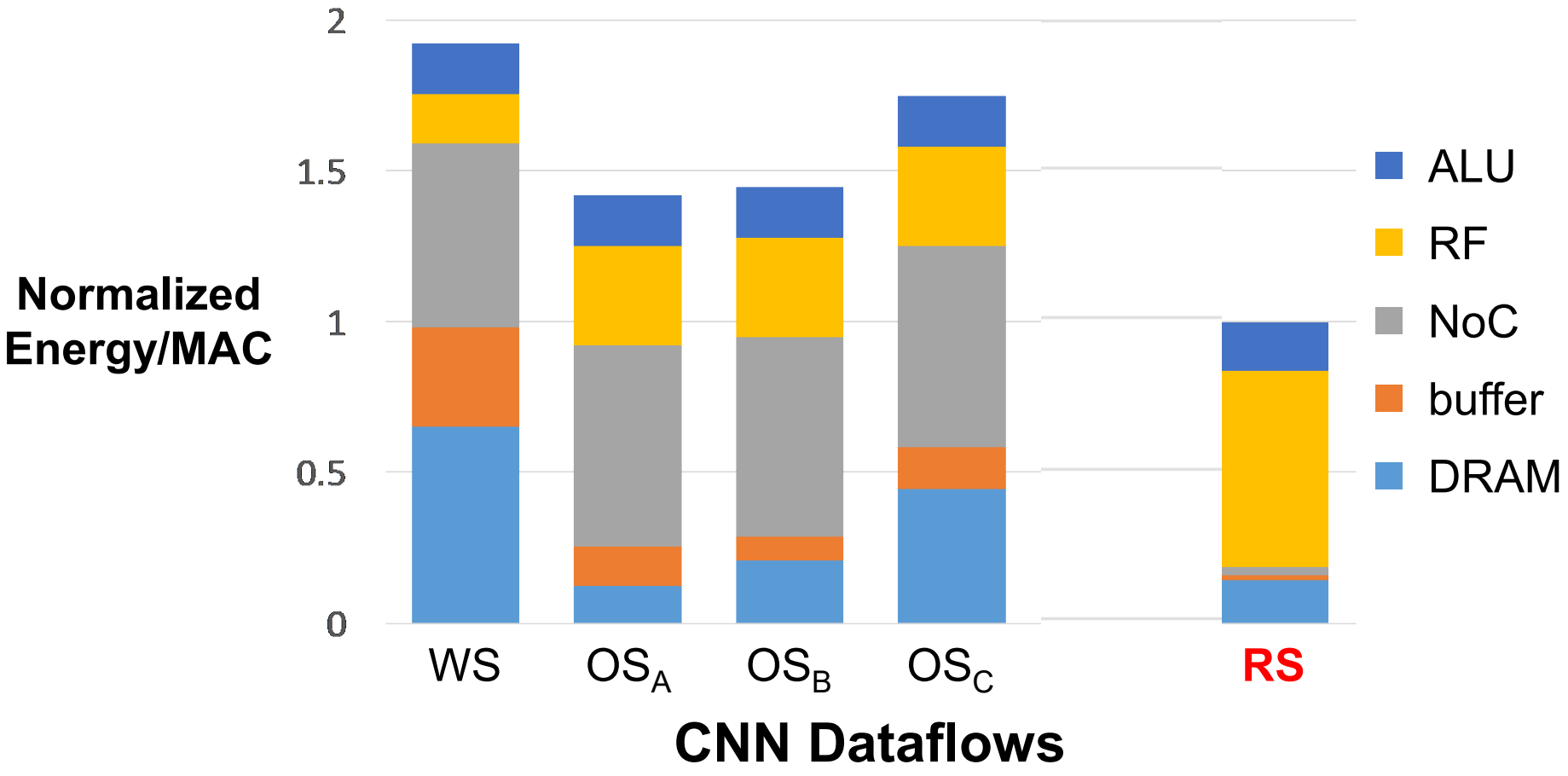
Dataflow Comparison: CONV Layers



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

[Chen et al., ISCA 2016]

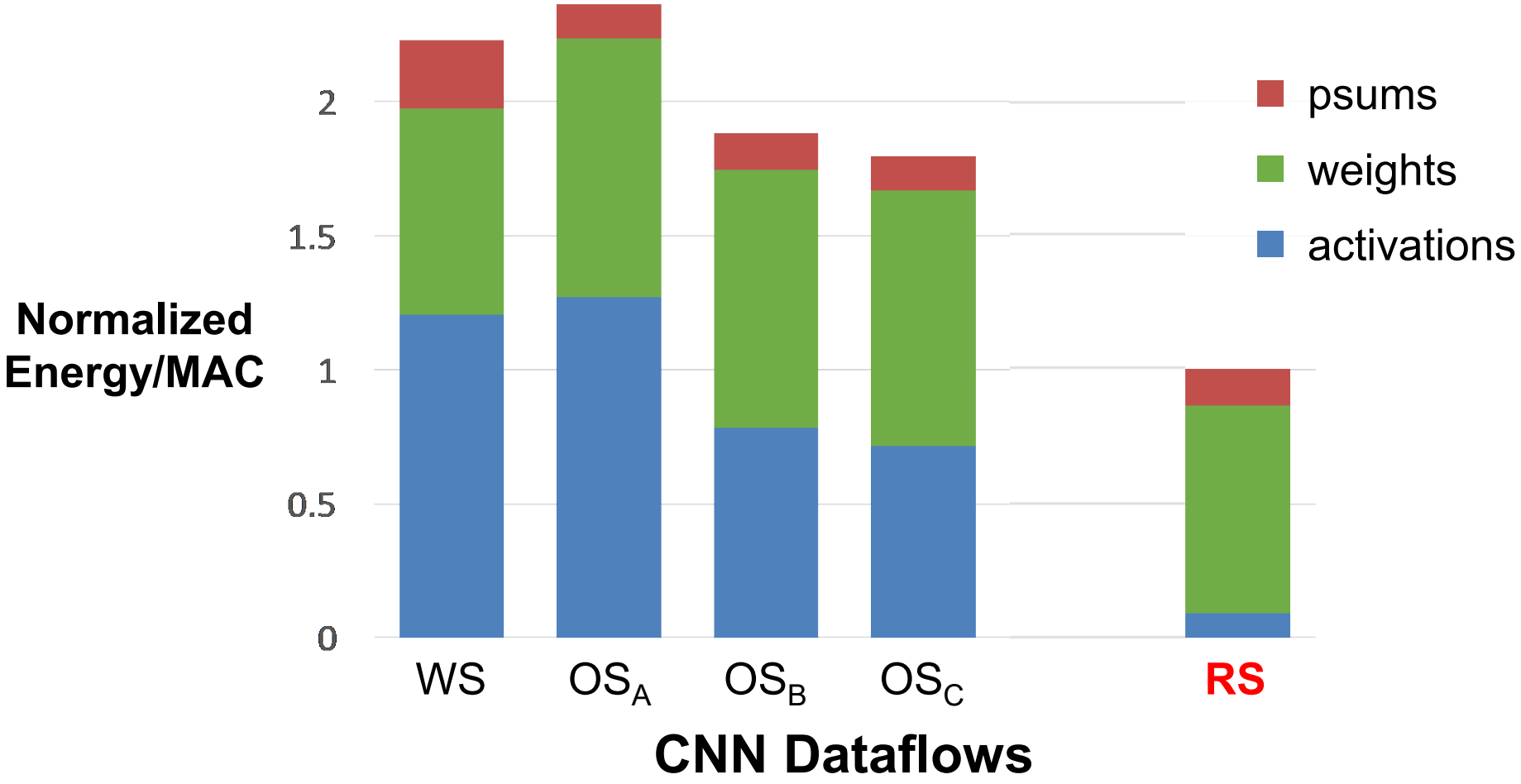
Dataflow Comparison: CONV Layers



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

[Chen et al., ISCA 2016]

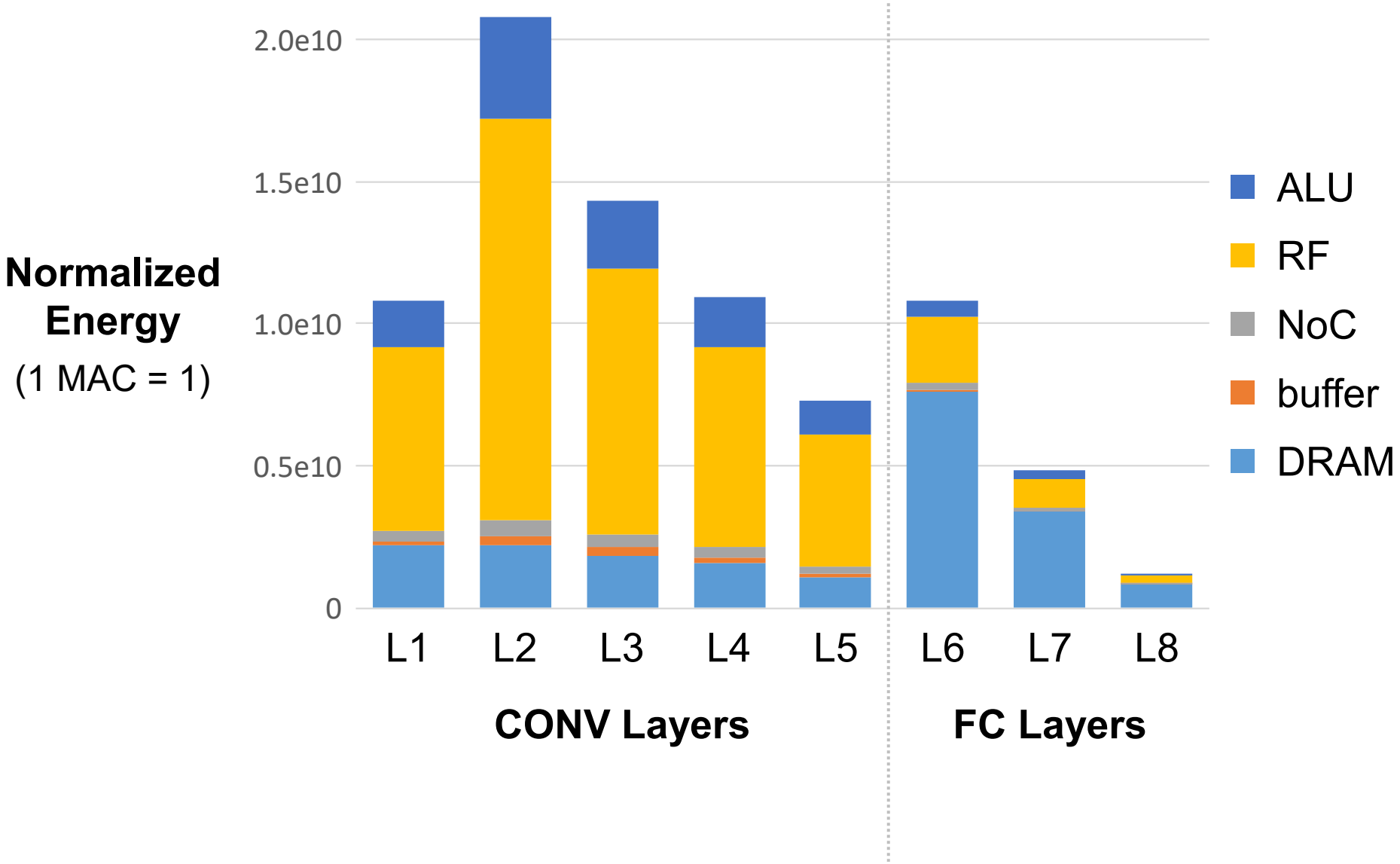
Dataflow Comparison: FC Layers



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

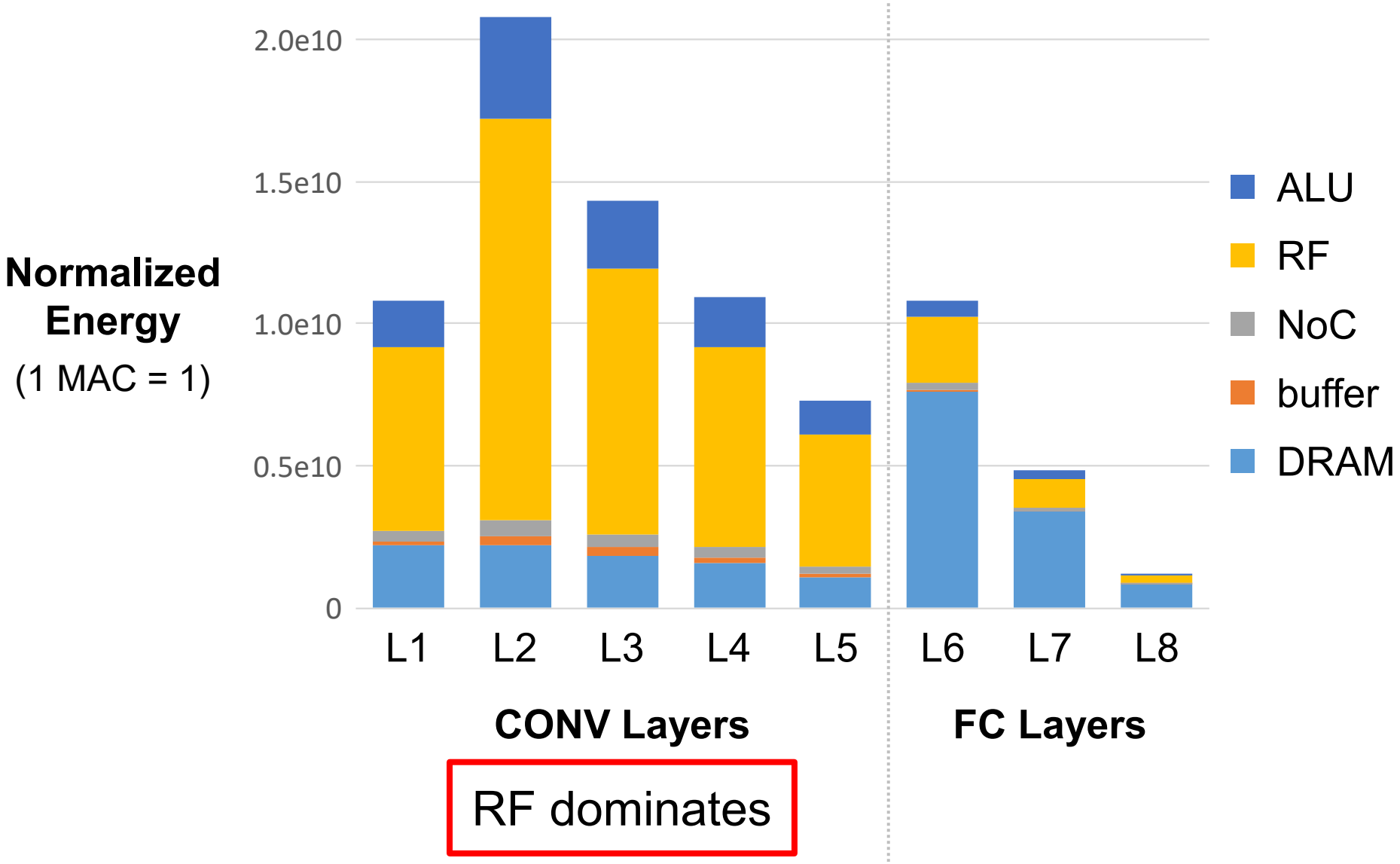
[Chen et al., ISCA 2016]

Row Stationary: Layer Breakdown



[Chen et al., ISCA 2016]

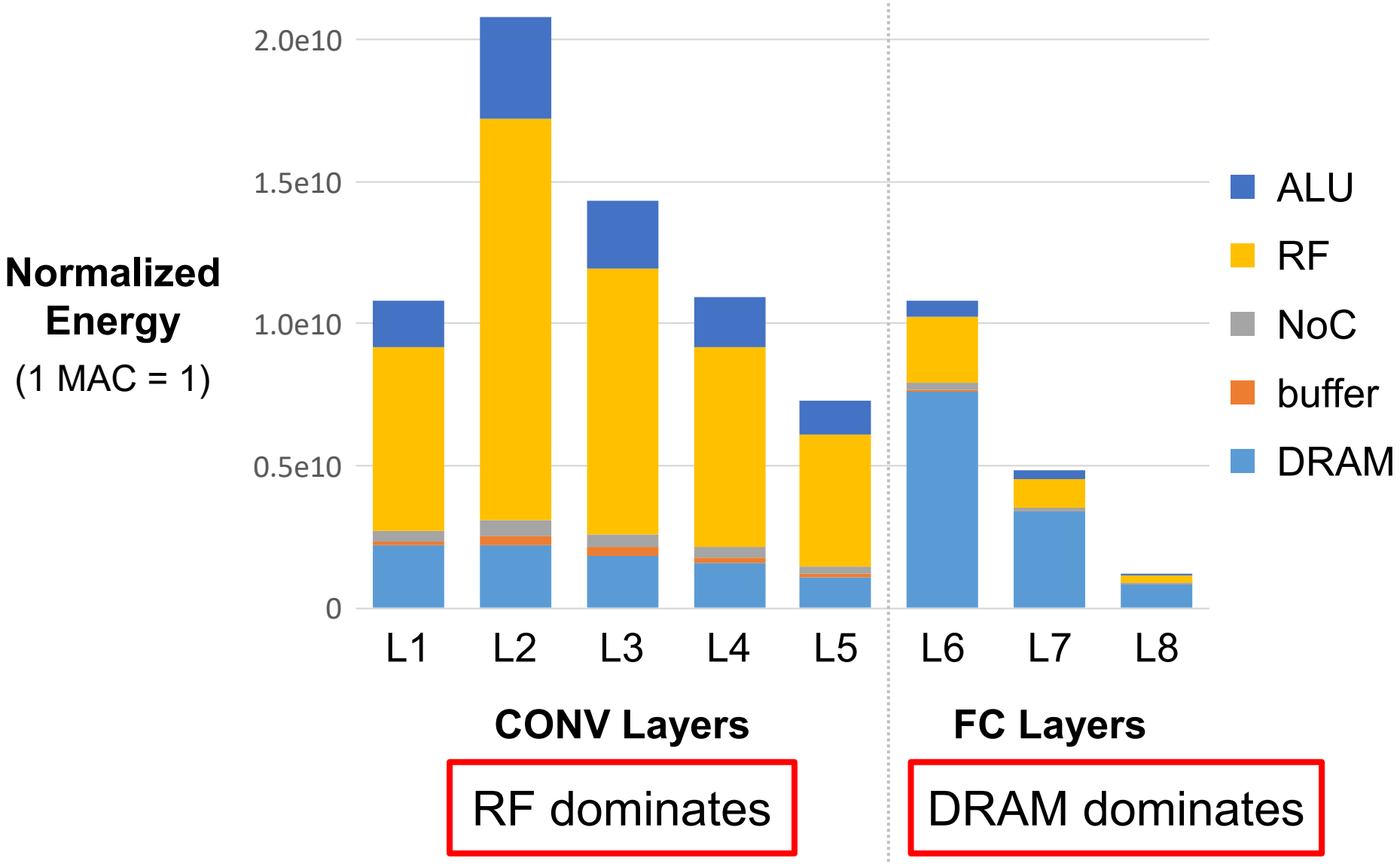
Row Stationary: Layer Breakdown



RF dominates

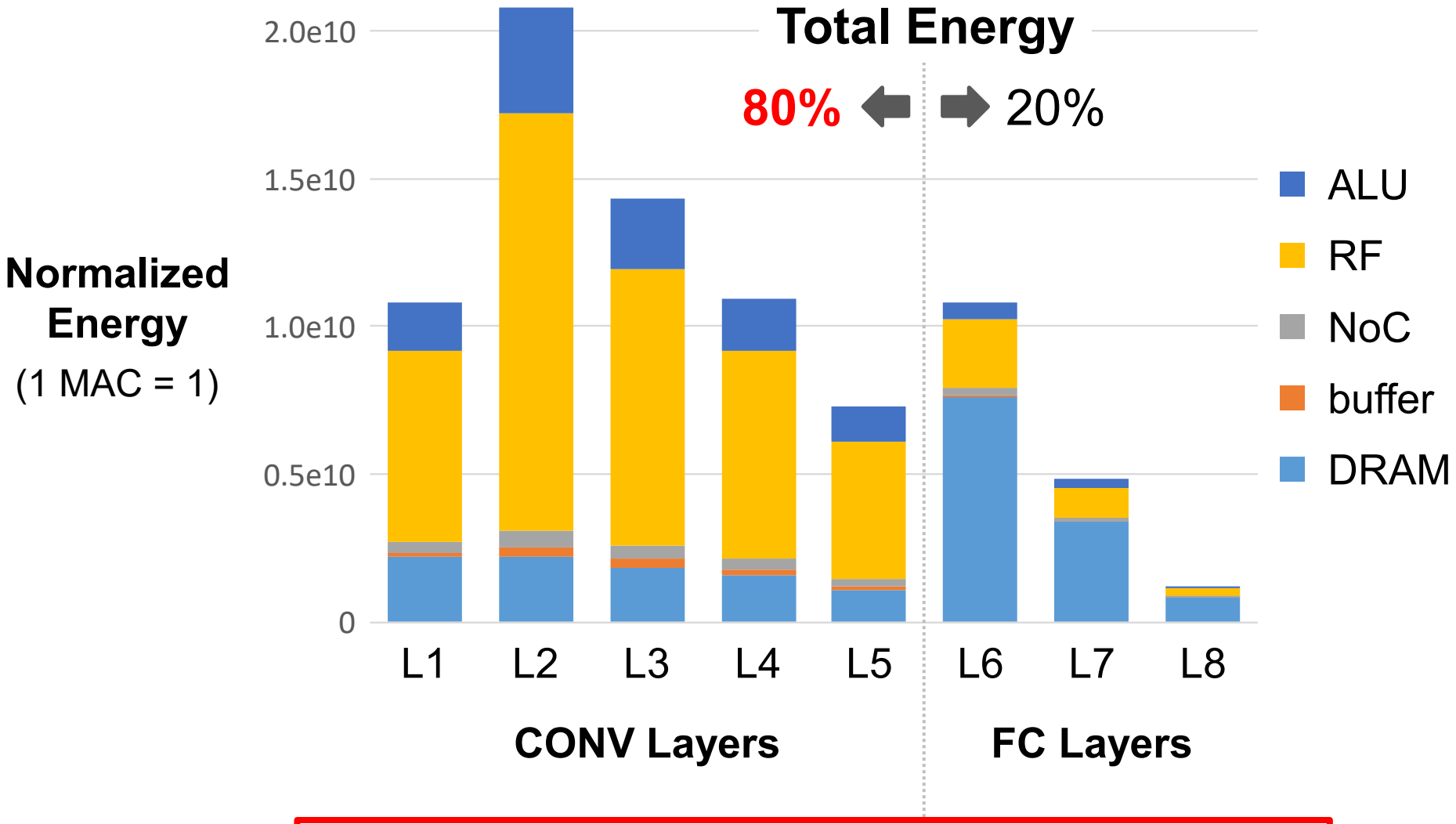
[Chen et al., ISCA 2016]

Row Stationary: Layer Breakdown



[Chen et al., ISCA 2016]

Row Stationary: Layer Breakdown



CONV layers dominate energy consumption!

Next:

Mapping