CS5120 VLSI System Design, Spring 2024

Data Reuse and Dataflow

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
NATIONAL TSING HUA UNIVERSITY

資訊工程學系
Computer Science

Lecture 05

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容 (例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

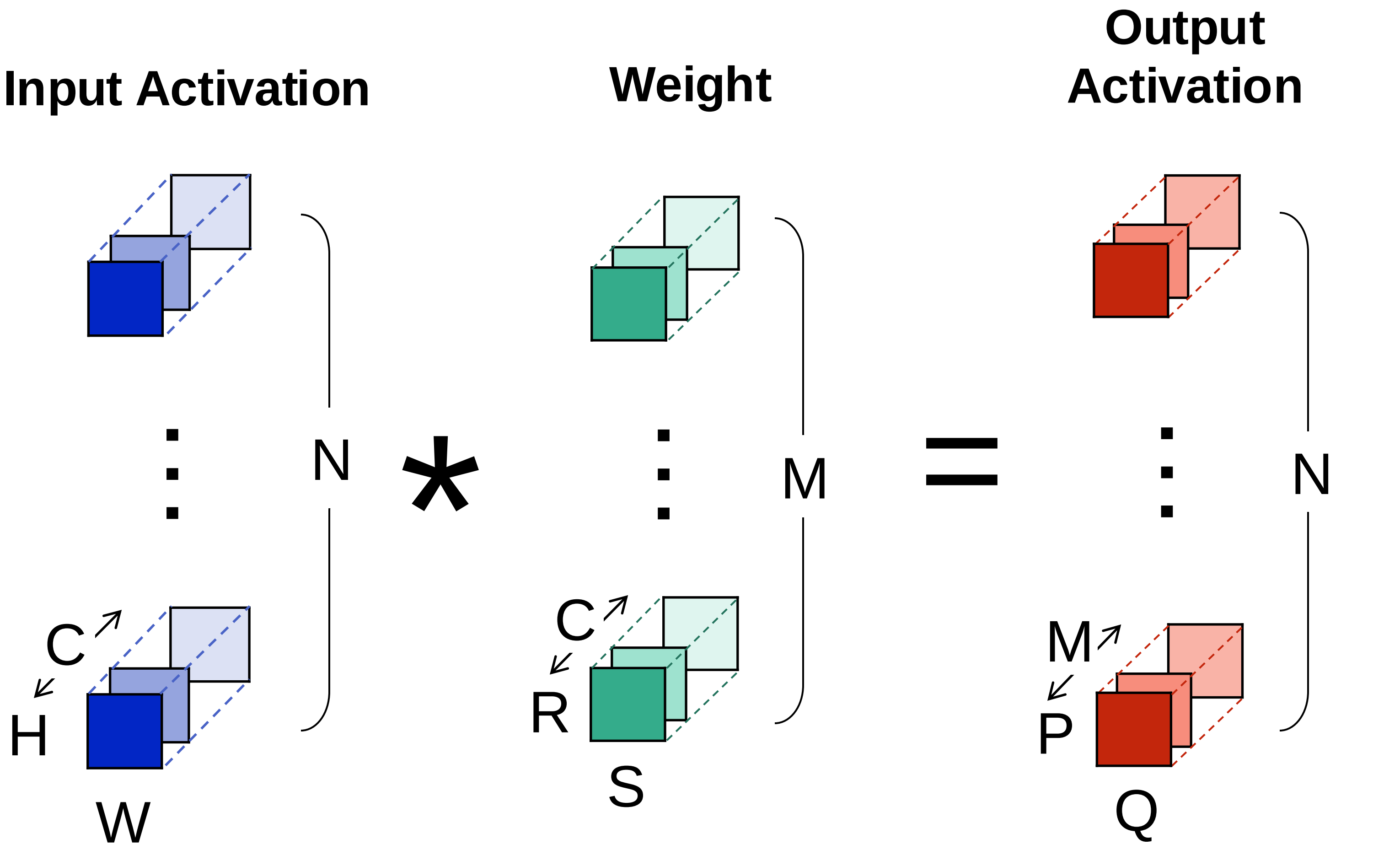
Outline

- Matrix Multiplication in DNN
- Tiling
- Data Locality and Reuse
- Data Reuse in DNN
  - Temporal reuse
  - Spatial reuse
  - Reducing reuse distance
- Dataflow
- Tiled Loop Nest for Dataflow

Reference:
V. Sze, Y.-H. Chen, T-.J. Yang and J. S. Emer,"
Efficient Processing of Deep Neural Networks -- Synthesis Lectures on Computer Architecture, "
Morgan&Calypool Publishers, 2020.

Recap: Matrix Multiplication in DNN

Recap: Fully-Connected Layer



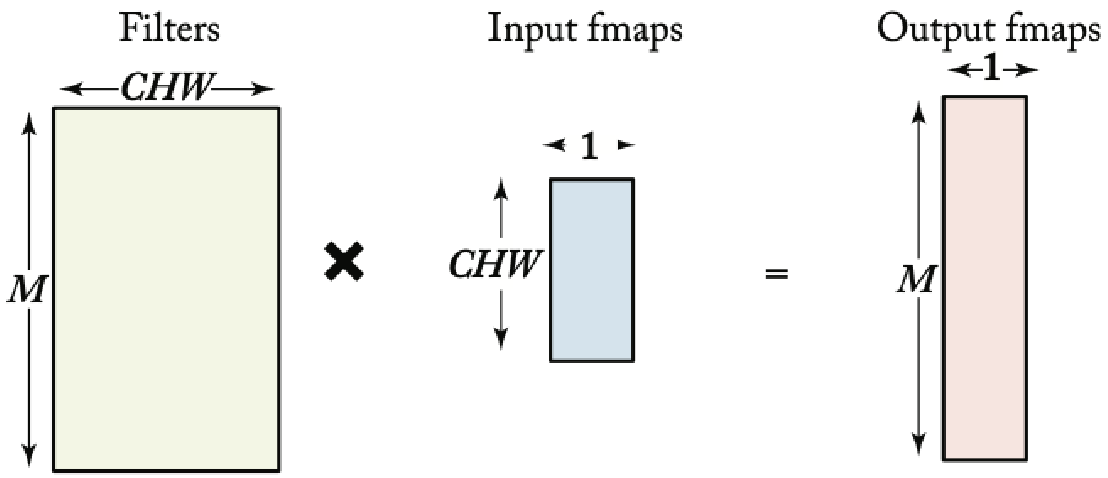
H = 1
W = 1
R = 1
S = 1
P = 1
Q = 1

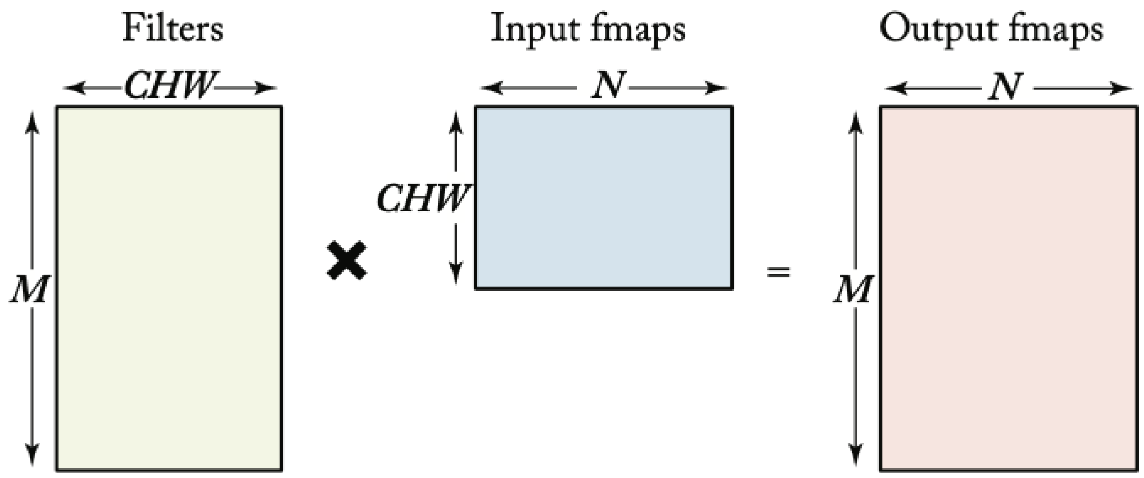
U (Stride)= 1

Pad (Padding) = 0

C: # of Input Channels
M: # of Output Channels
N: Batch size

Alternative: Mapping FC Layer to Matrix Multiplication





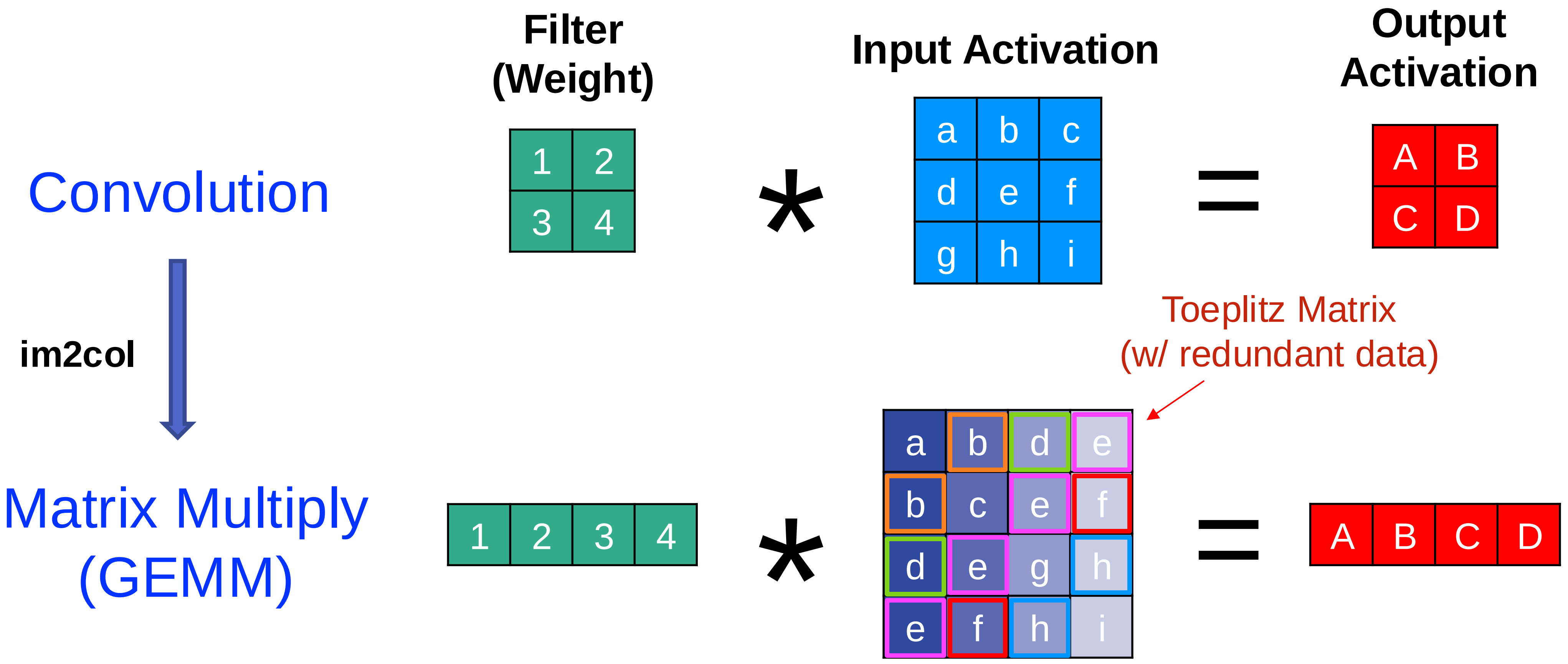
Single Input Activation Map

Batch Size = 1

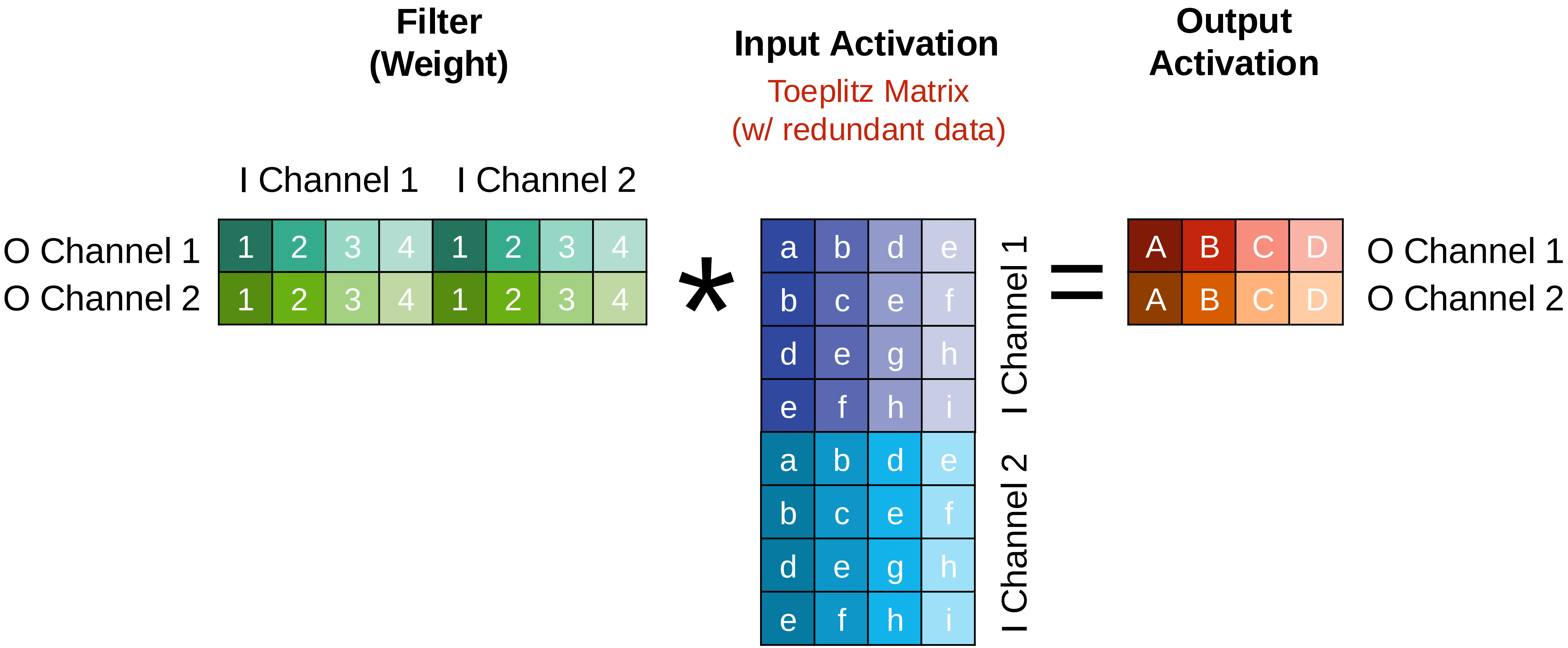
Batch Size = *N*

Recap: Mapping Convolution to Matrix Multiplication (Redundant Input Activations)

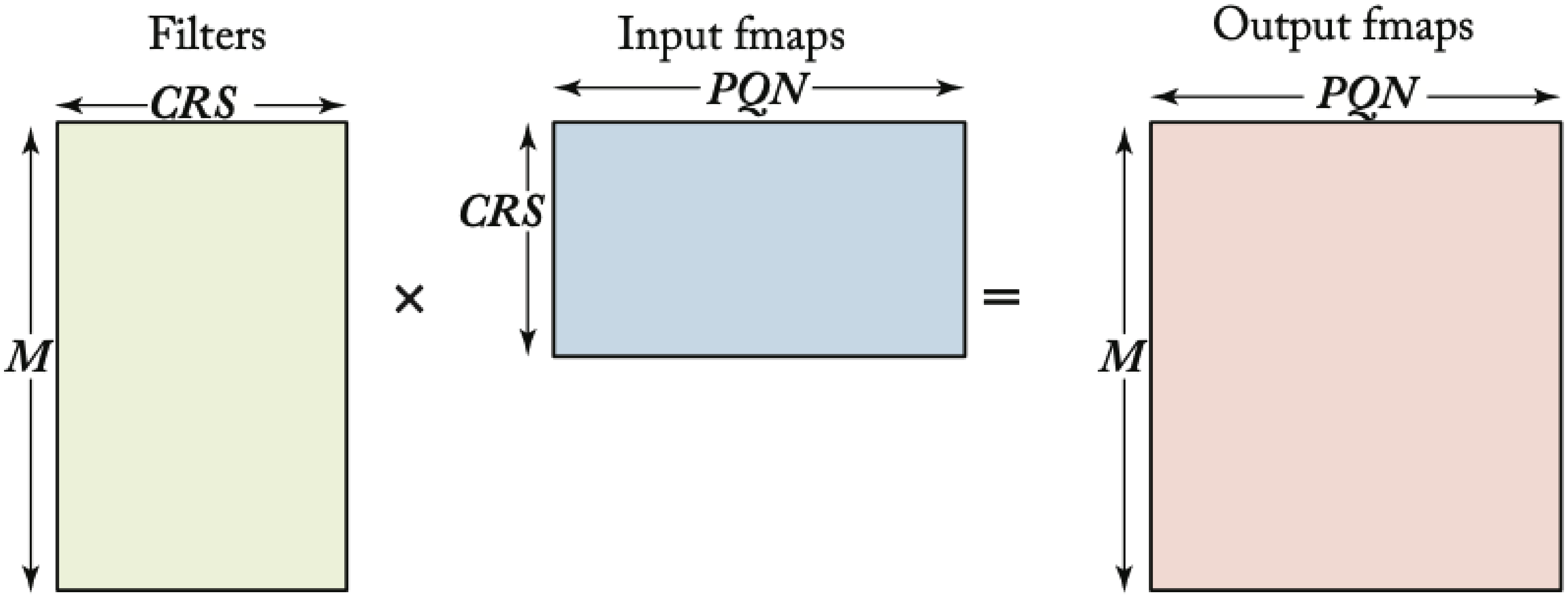
- Converting convolution to GEMM via **im2col**



Mapping Convolution to Matrix Multiplication: Multiple Input/Output Channels

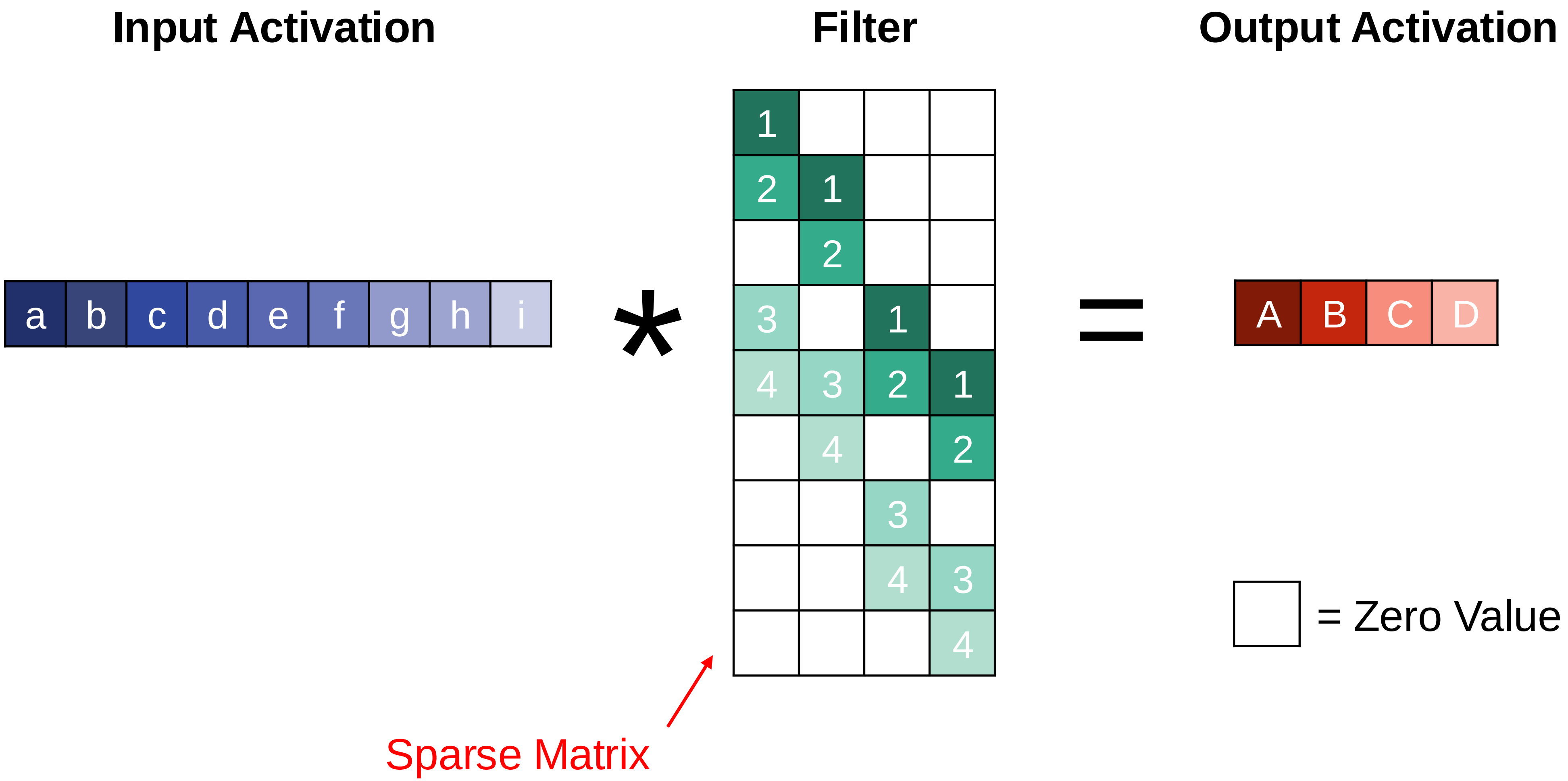


Mapping Convolution Layer to Matrix Multiplication



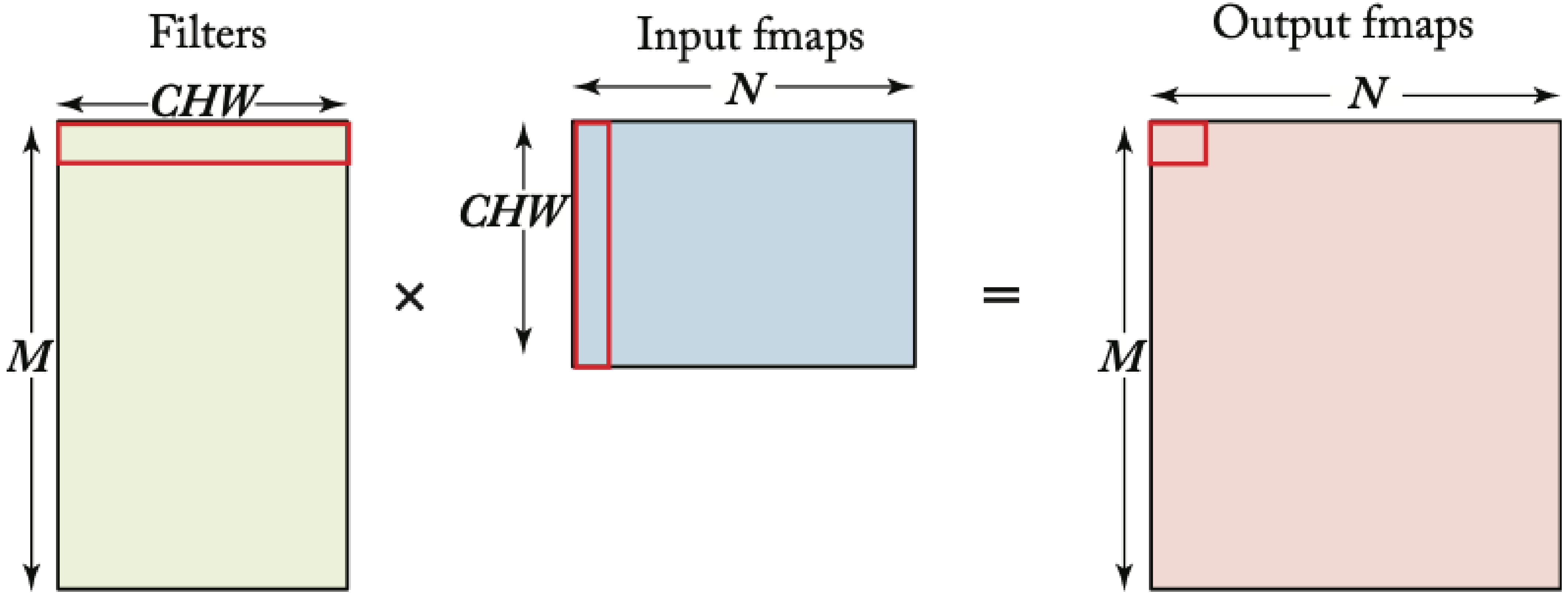
$$P = \frac{(H - R + U)}{U}, Q = \frac{(W - S + U)}{U}$$

Mapping Convolution to Matrix Multiplication (Redundant Weights)

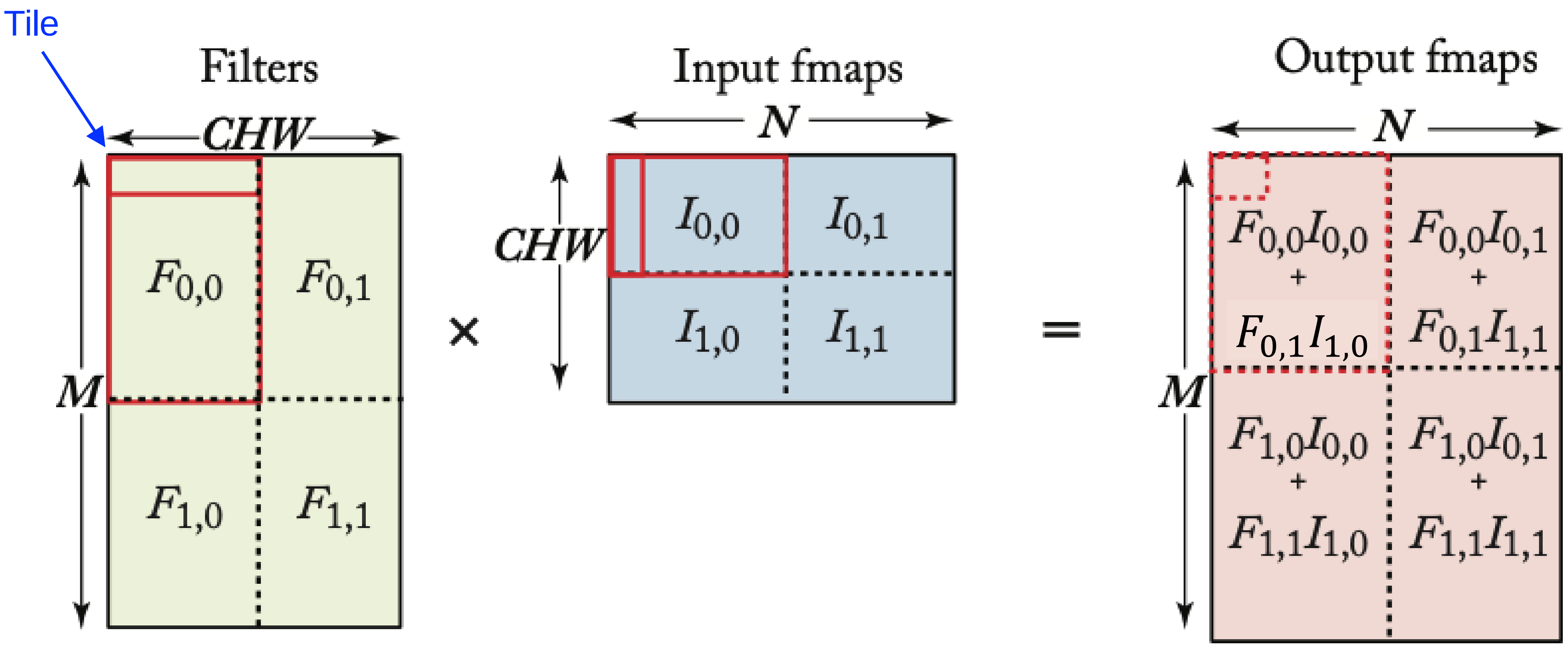


Tiling

Recap:
Naïve Matrix Multiplication for Fully-Connected Layer



Concept of Tiling: Tile-based Computation

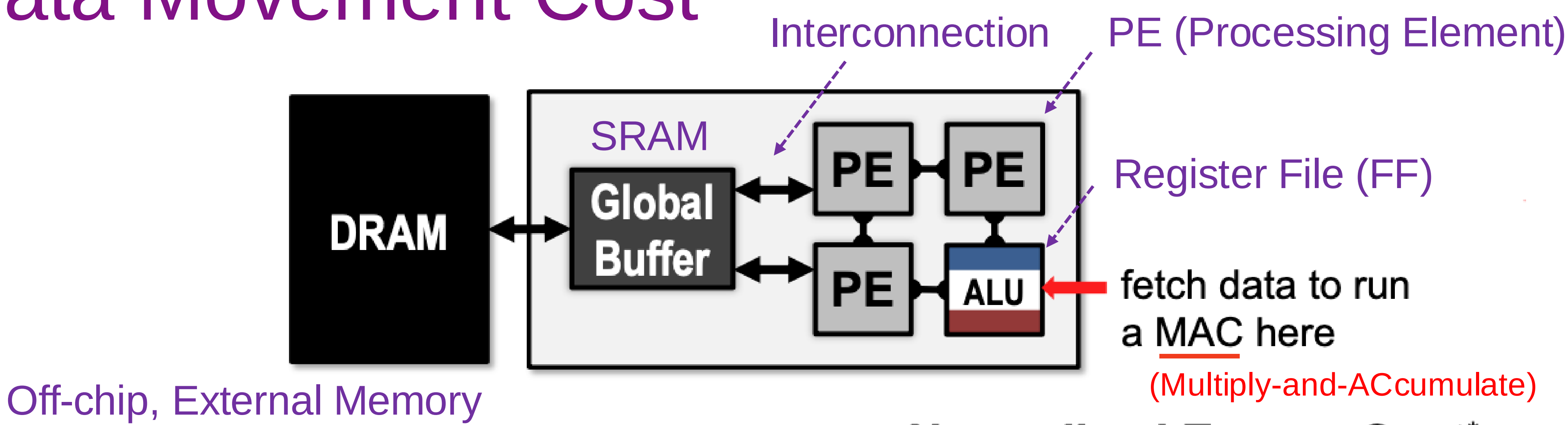


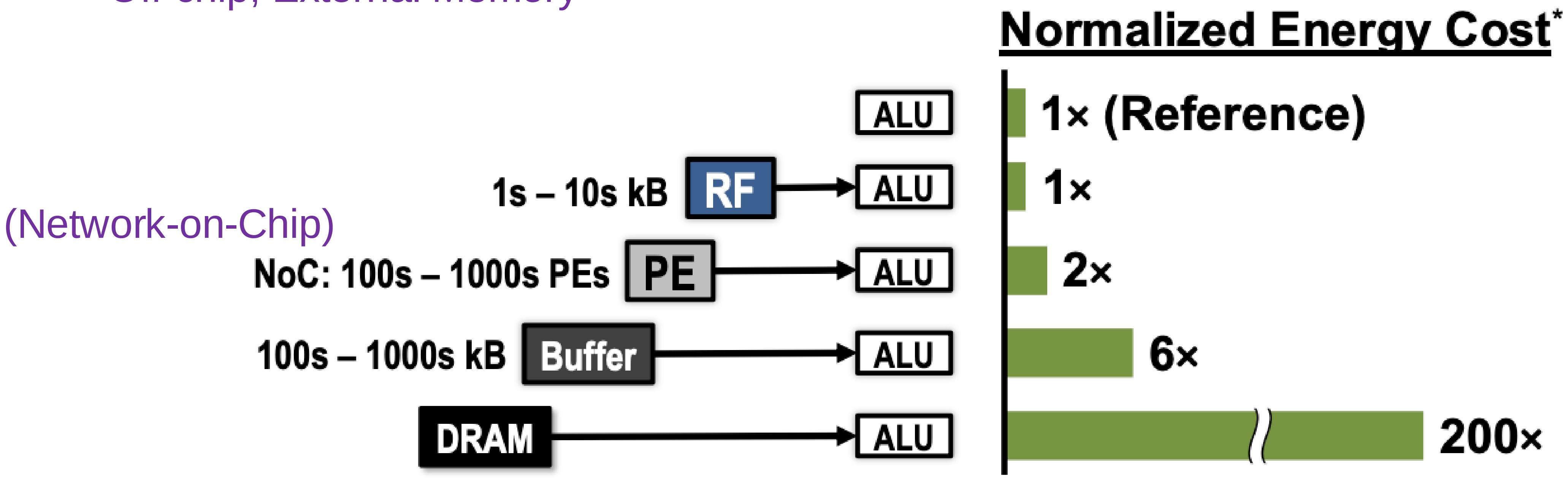
Tiling for Matrix Multiplication in Hardware

- Partition the matrix operation into small submatrices (tiles) to optimize hardware execution
  - Improves data reuse (locality) across different levels of memory hierarchy
  - Reduces memory bandwidth by keeping data closer to compute units
  - Allows for parallel execution on hardware accelerators
  - May generate subsequent partial results (psum) to be added
- Memory management
  - Not primarily managed by hardware caches (i.e., implicitly data orchestration)
  - Instead, explicitly managed local buffers (scratchpads) store and reuse tiles controlled explicitly by programmers or designers
  - Optimized data flow reduces costly global memory access

Data Locality and Reuse

Data Movement Cost

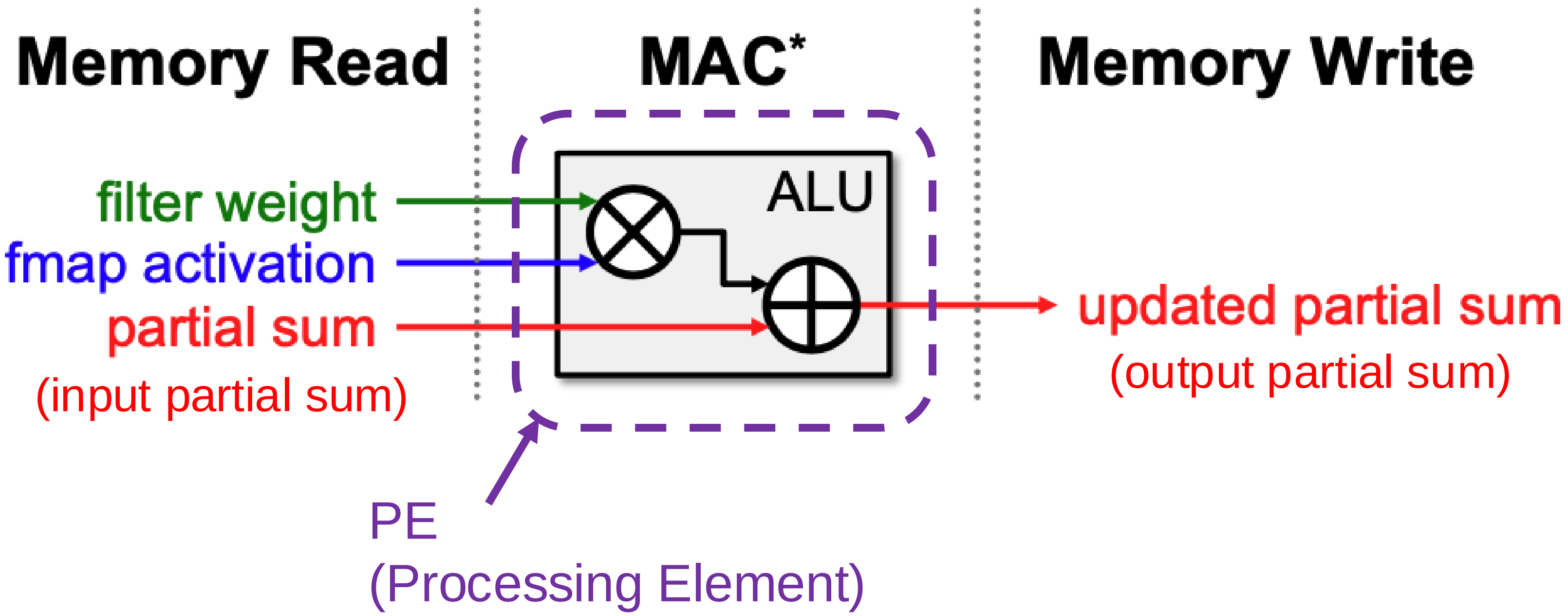




[Source: MIT Eyeriss Tutorial]

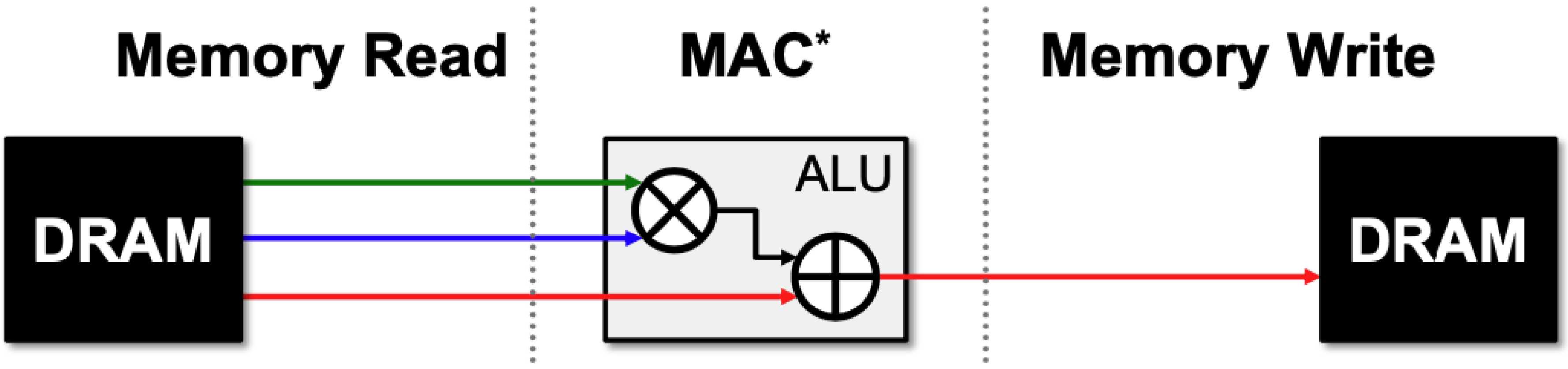
Single MAC Unit for DNN

* MAC: Multiply-and-ACcumulate



[Source: MIT Eyeriss Tutorial]

Memory Access is the Bottleneck

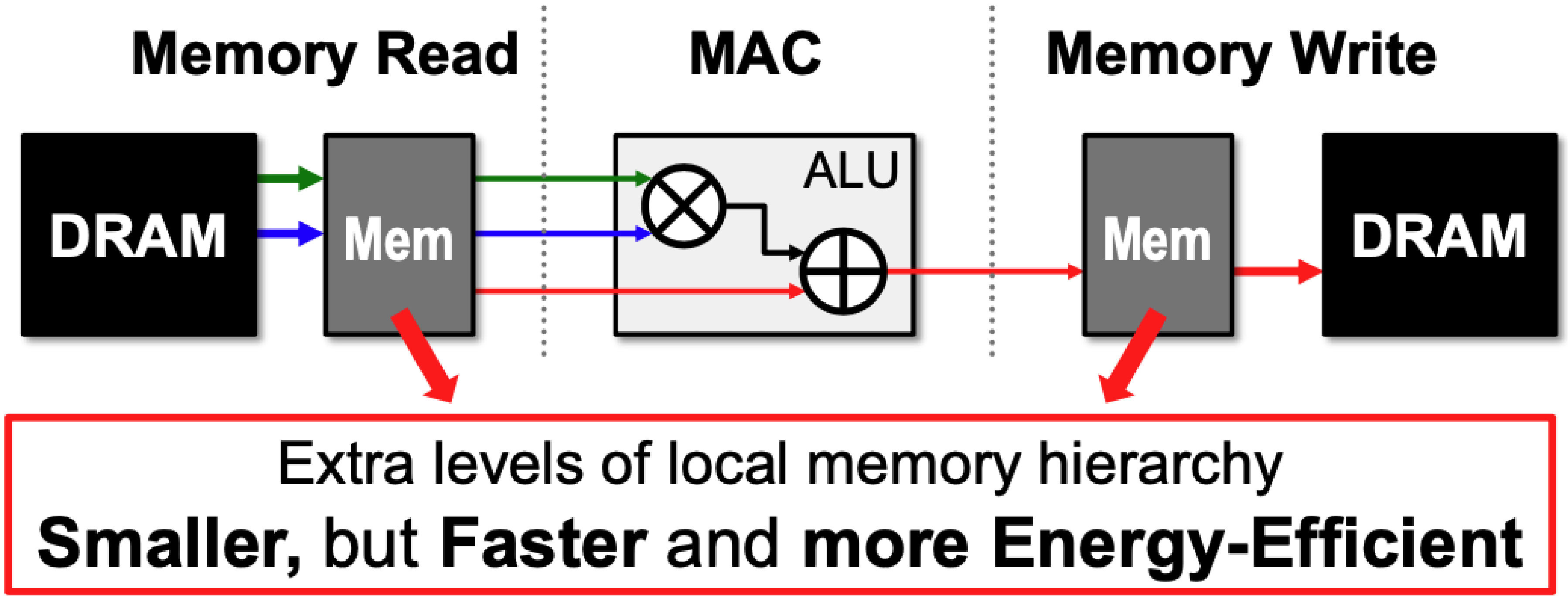


* MAC: Multiply-and-ACcumulate

- Worst case: all memory R/W are DRAM accesses
- E.g., AlexNet has 724M MACs
  ➔ 2896M DRAM accesses required
  (extremely energy inefficient!)

[Source: MIT Eyeriss Tutorial]

Locality: Leverage Local Memory for Data Reuse



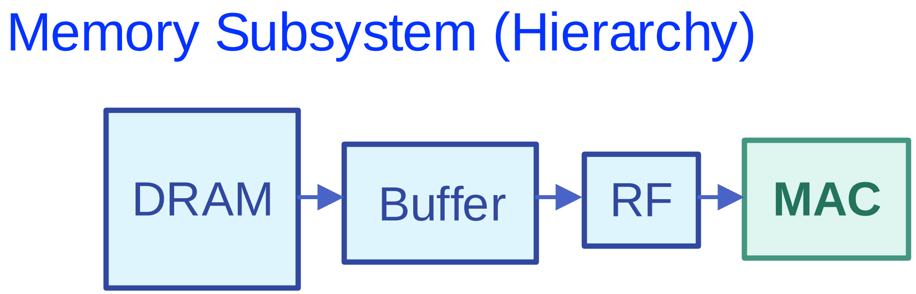
(Usually more expensive with more levels)

- **Temporal** reuse: the same data is used **more than once over time** by the same consumer

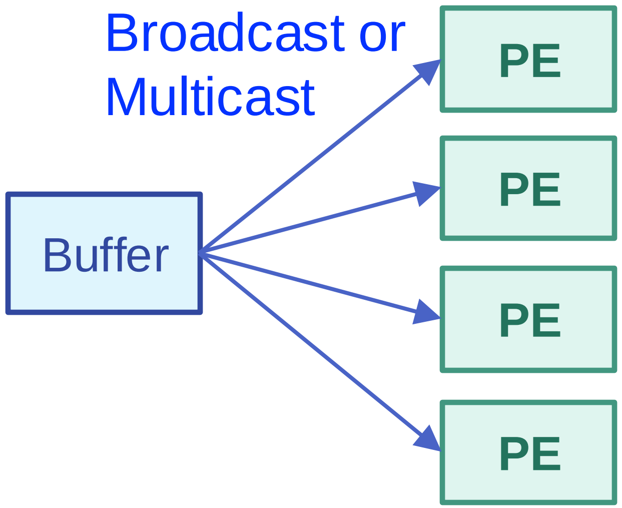
[Source: MIT Eyeriss Tutorial]

Temporal and Spatial Reuse

- **Temporal** reuse
  - The same data is used **more than once over time** by the same consumer

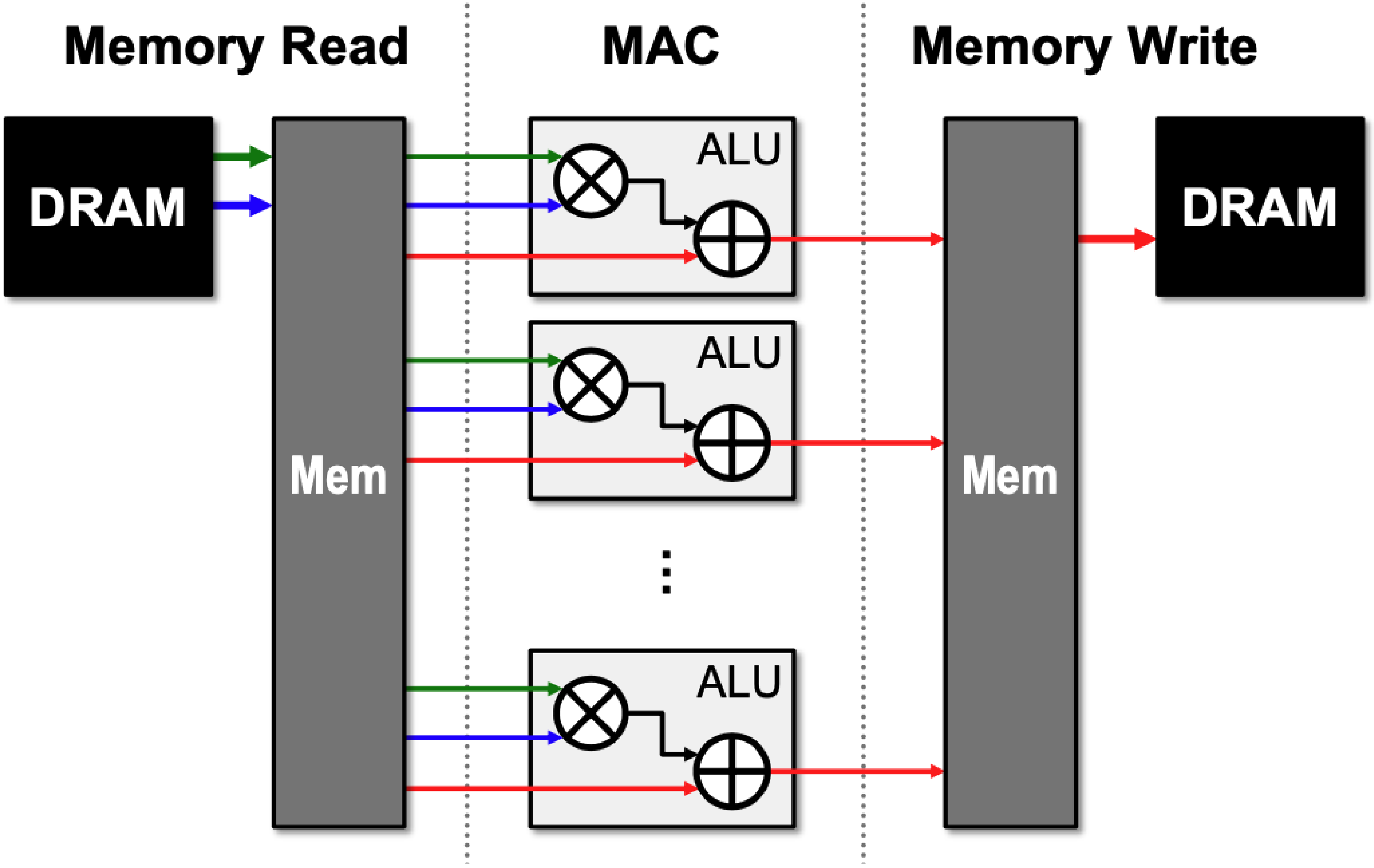


- **Spatial** reuse
  - The same data is used by **more than one consumer** at **different spatial locations** of the hardware



[Source: Prof. Sophia Shao, EE290-2, Berkeley]

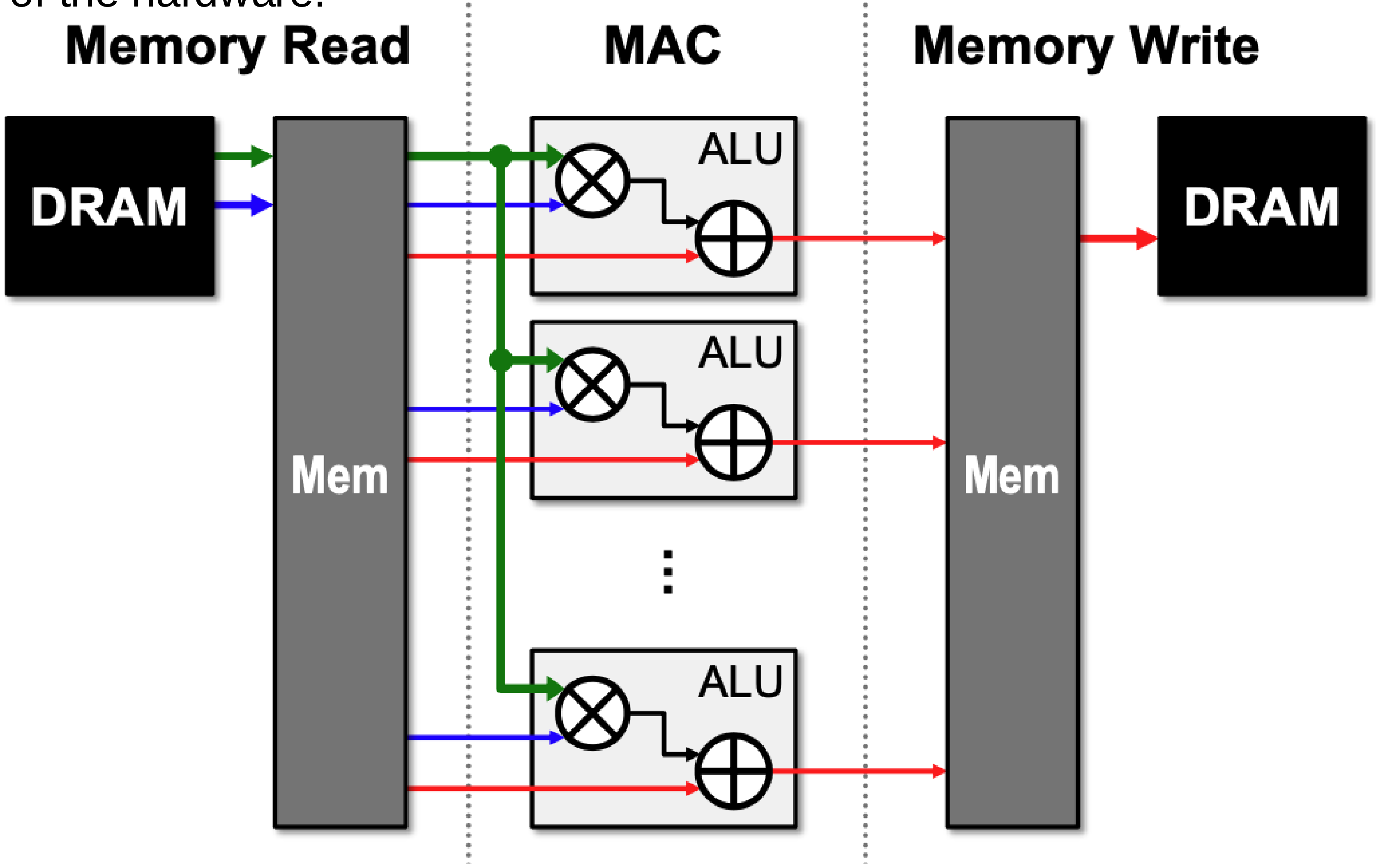
Leverage Parallelism for Higher Performance



[Source: MIT Eyeriss Tutorial]

Leverage Parallelism with Spatial Data Reuse

- **Spatial reuse**: the same data is used by **more than one consumer** at **different spatial locations** of the hardware.

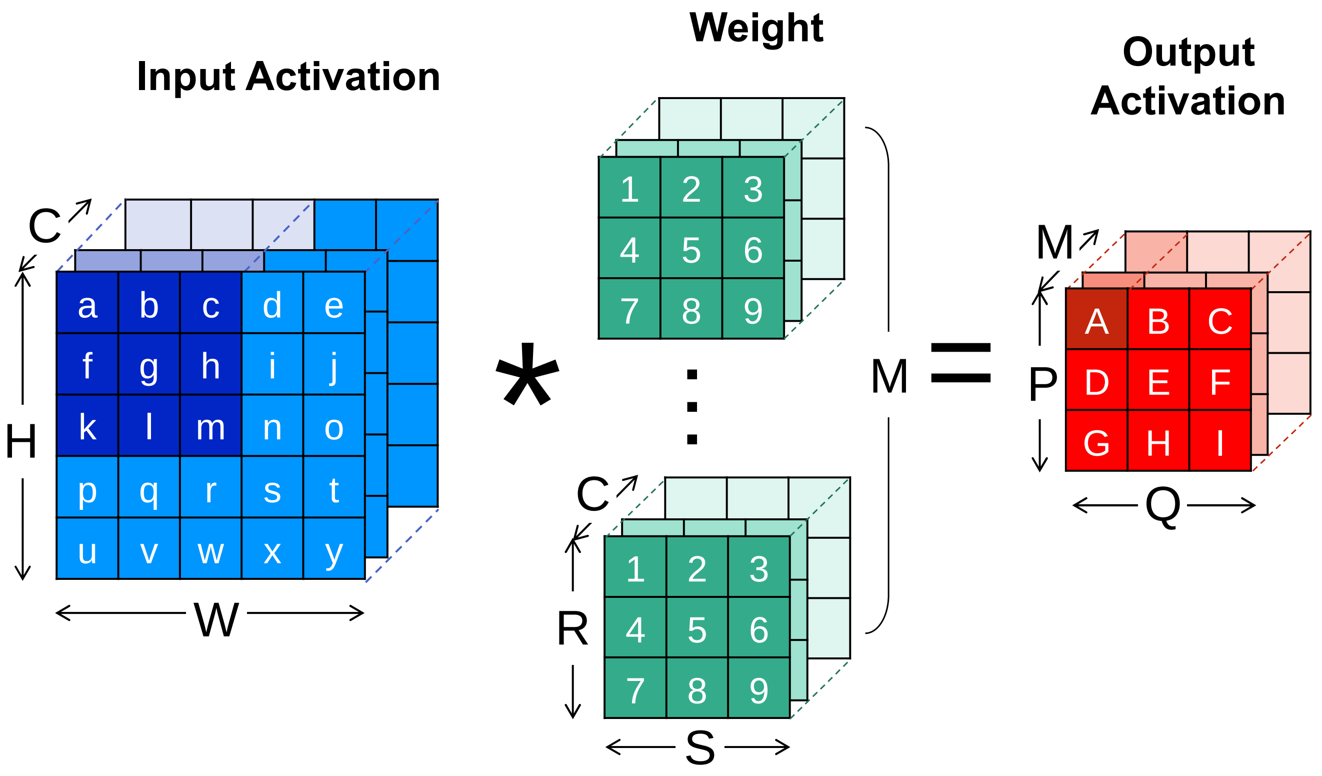


[Source: MIT Eyeriss Tutorial]

Data Reuse in DNN

Data Reuse in DNN

Batch Size = 1 ($N = 1$)



- # MACs:
  - $RSCPQM$
- # Input Activations
  - Size: HWC
  - Max reuse: $\sim RSM$
- # Weights
  - Size: $RSCM$
  - Max reuse: PQ
- # Output Activations
  - Size: PQM
  - Max reuse: RSC

Dataflow and Mapping to Hardware

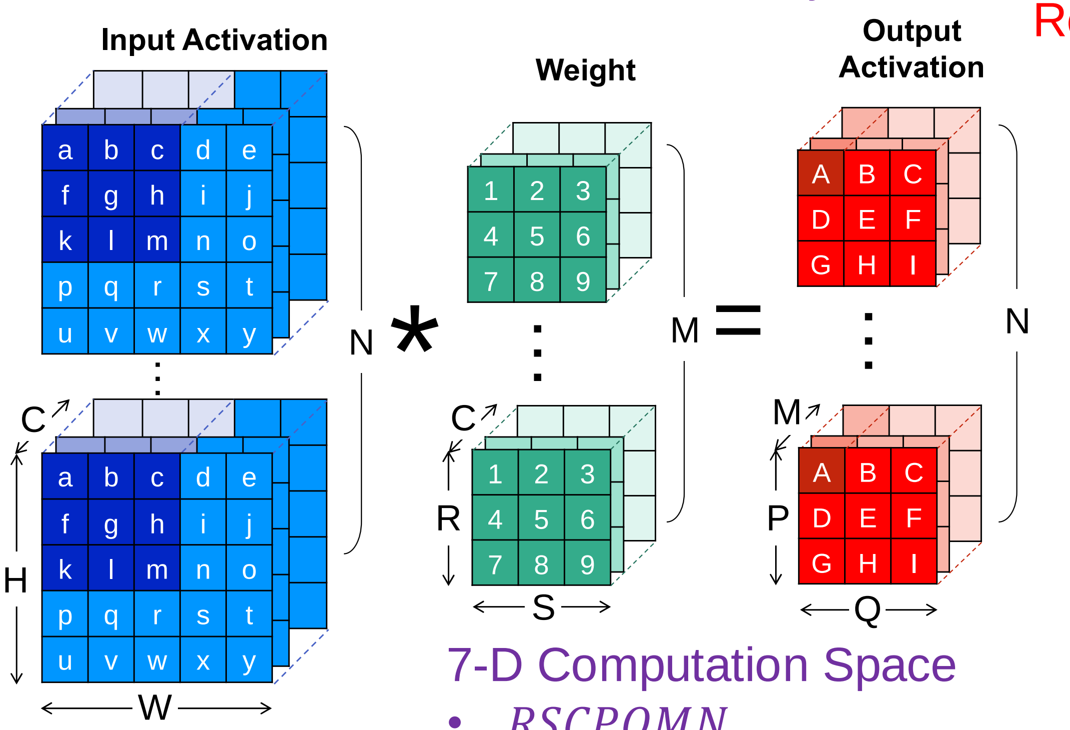
Millions of non-trivial mappings

7-dimensional Network Layer

Algorithmic Reuse

Hardware Reuse

1-d or 2-d (or 3-d or …) Hardware

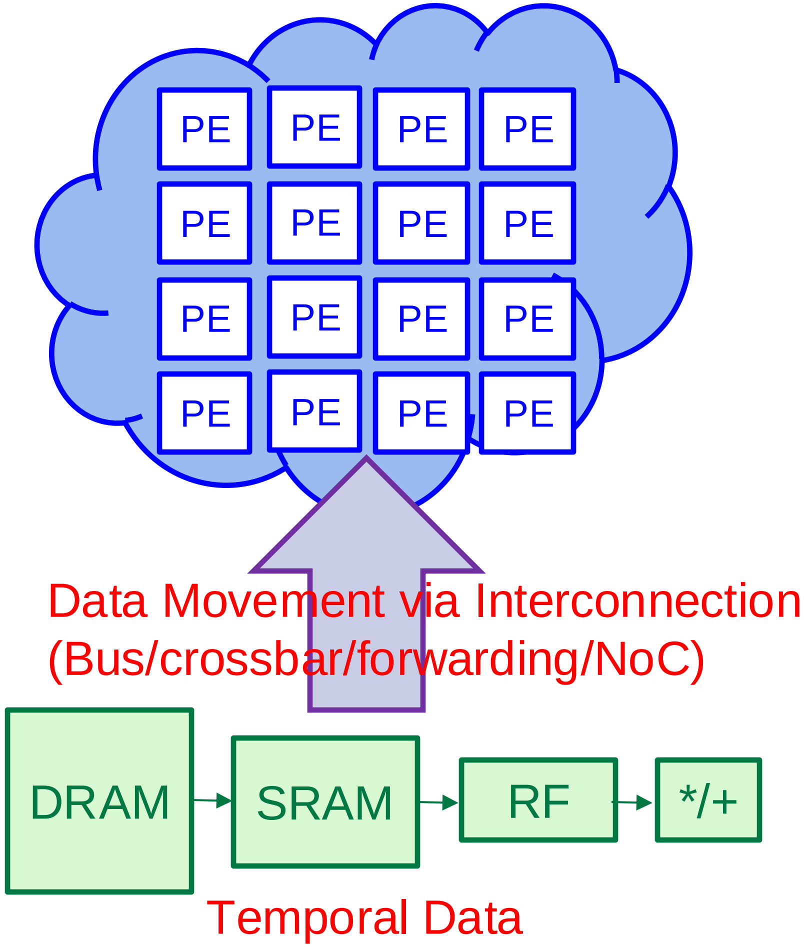


7-D Computation Space

- $RSCPQMN$

4D Operand / Result Space

- Weights: $RSCM$
- Inputs: $HWCN$
- Outputs: $PQMN$



http://synergy.ece.gatech.edu/tools/maestro/

Some Statistics for Your Reference

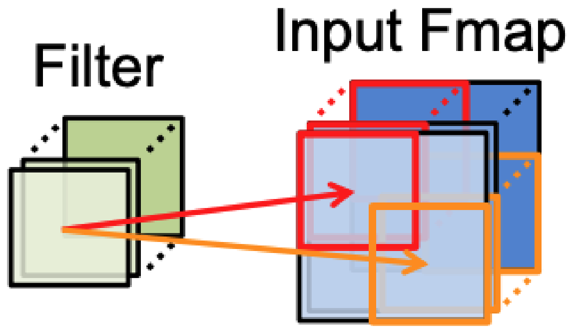
- AlexNet conv2
  - Ifmap: 46K; Filter: 300K; Ofmap: 180K (total: 528K data)
  - MAC: 224M
- MobileNet V1 conv2_1/dw
  - Ifmap: 420K; Filter 9.2K, Ofmap: 400K (total: 829K data)
  - MAC: 116M
- ResNet50 res5a_branch1
  - Ifmap: 200K; Filter 2.1M, Ofmap: 100K (total: 2.4M data)
  - MAC: 103M

Types of Data Reuse in DNN

For ideal data reuse, DRAM accesses in AlexNet can be reduced from 2896M to 61M

Convolutional Reuse

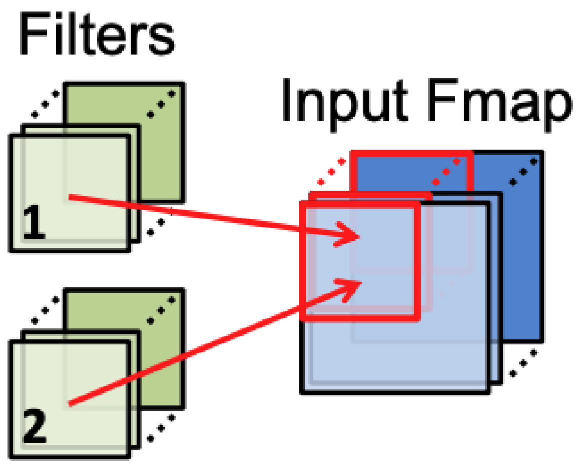
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

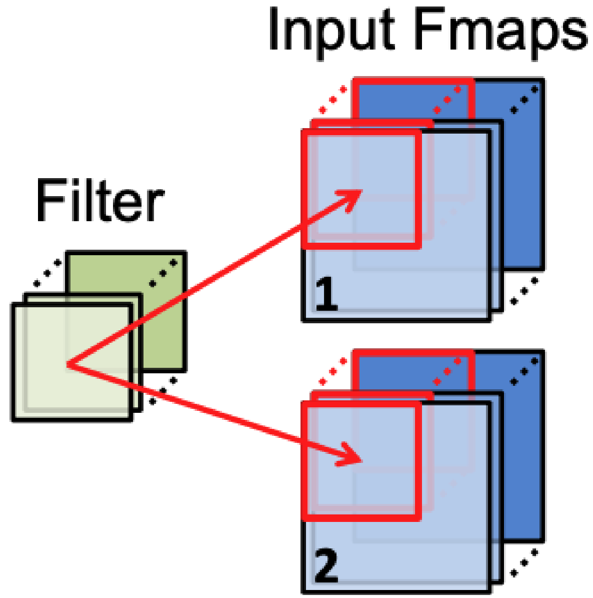
CONV and FC layers



Reuse: Activations

Filter Reuse

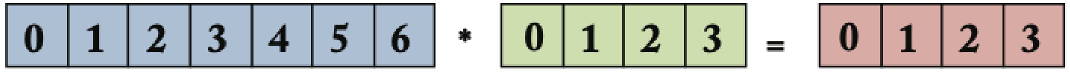
CONV and FC layers
(batch size > 1)



Reuse: Filter weights

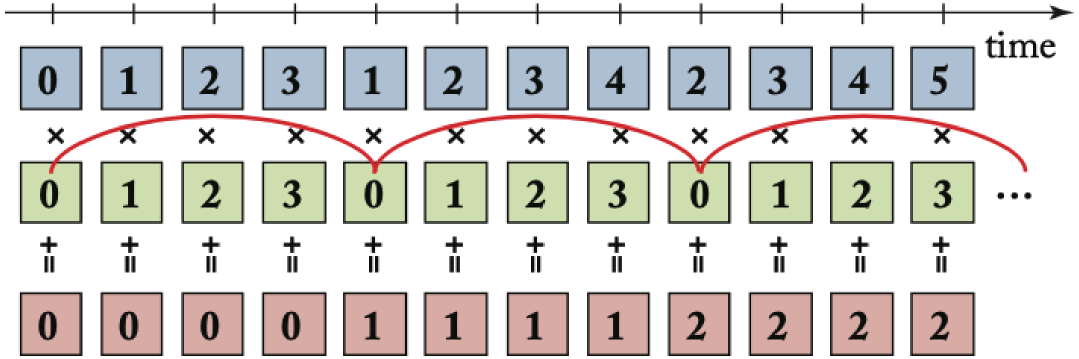
Temporal Reuse

1-D Convolution with Temporal Reuse



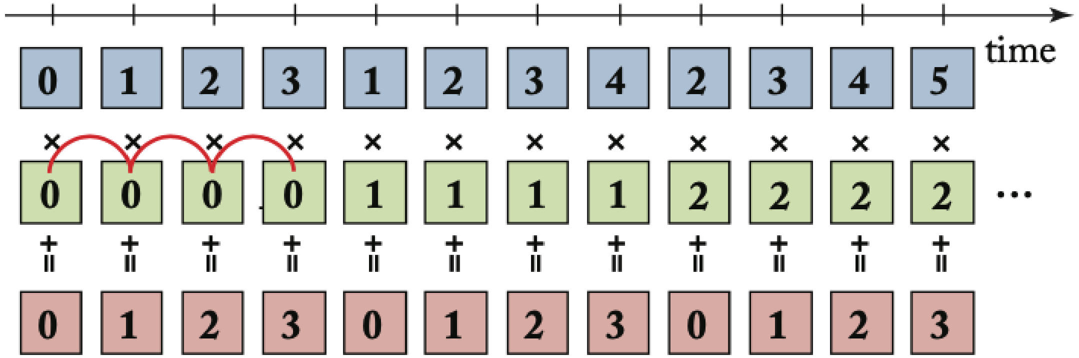
Example 1D convolution

Scheduling 1

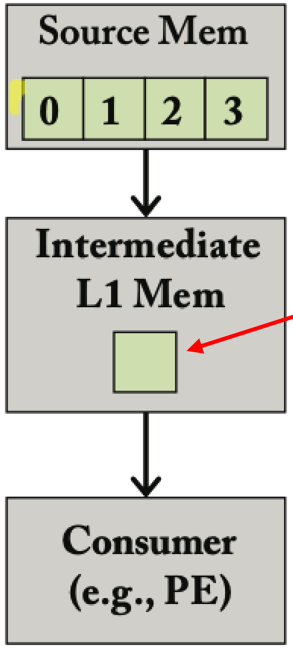


Operation ordering 1: weight reuse distance = 4

Scheduling 2



Operation ordering 2: weight reuse distance = 1



* Only storage for weights is shown

Memory Hierarchy

1 slot of intermediate memory for weights (buffer)

Temporal Reuse to Improve Efficiency

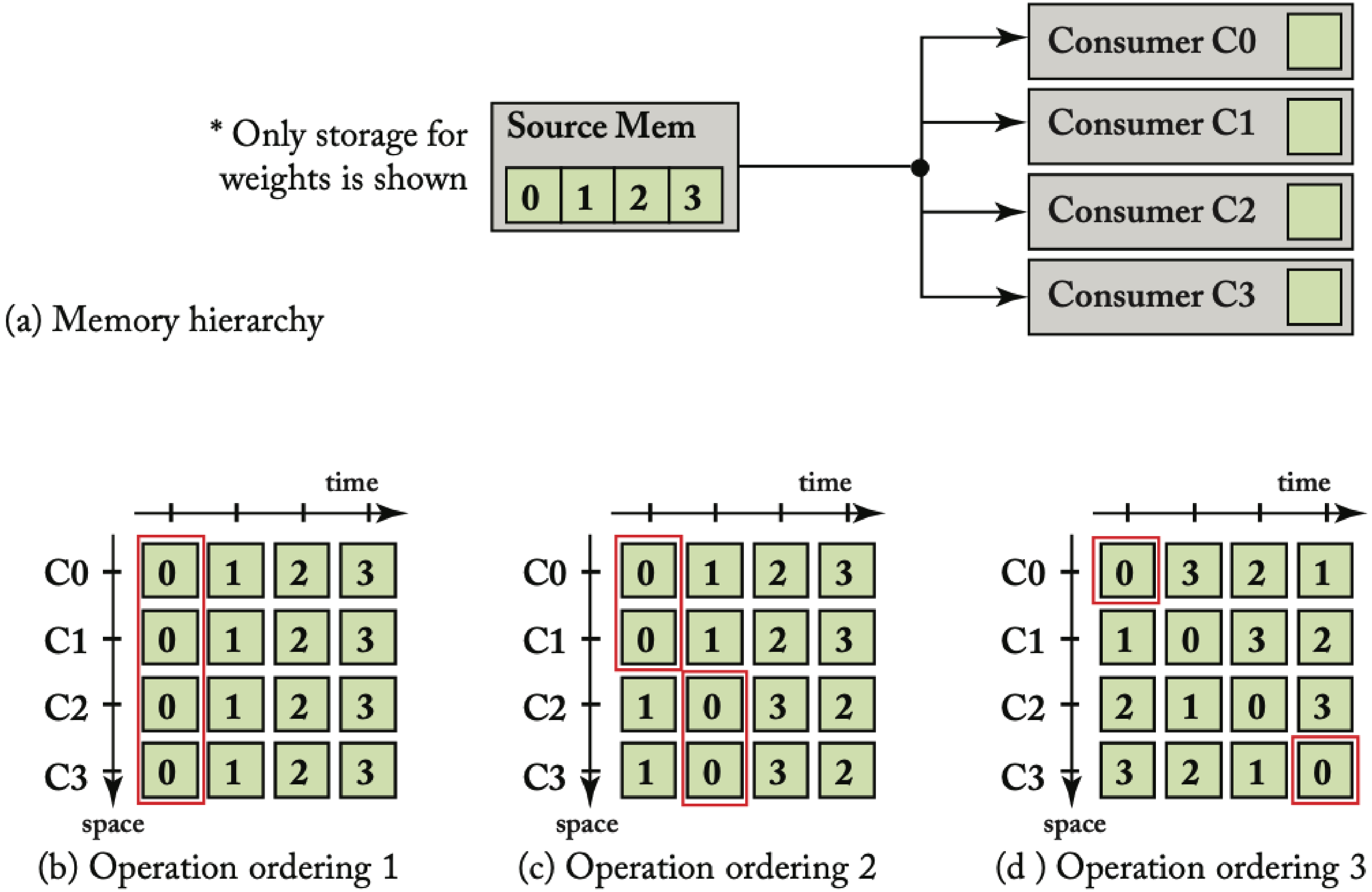
- Temporal reuse occurs when the same data value is used more than once by the same consumer (e.g., a PE)
  - By adding an intermediate memory level to the memory hierarchy of the hardware
- Benefits
  - Less energy by accessing data from smaller memory level
  - May be faster too

Reuse Distance for Temporal Reuse

- Reuse distance of temporal reuse
  - Number of data accesses required by the consumer in between the accesses to the same data value
  - A function of the ordering of operations
- Storage capacity of intermediate memory limits the maximum reuse distance to be exploited
  - However, larger storage capacity implies more energy
- Reducing the reuse distance of one data type (ifmap, weight, or ofmap) often comes at the cost of increasing the reuse distance of other types
- Changing the processing order of compute can alter the reuse distance

Spatial Reuse

1-D Convolution with Spatial Reuse



Spatial Reuse to Improve Efficiency

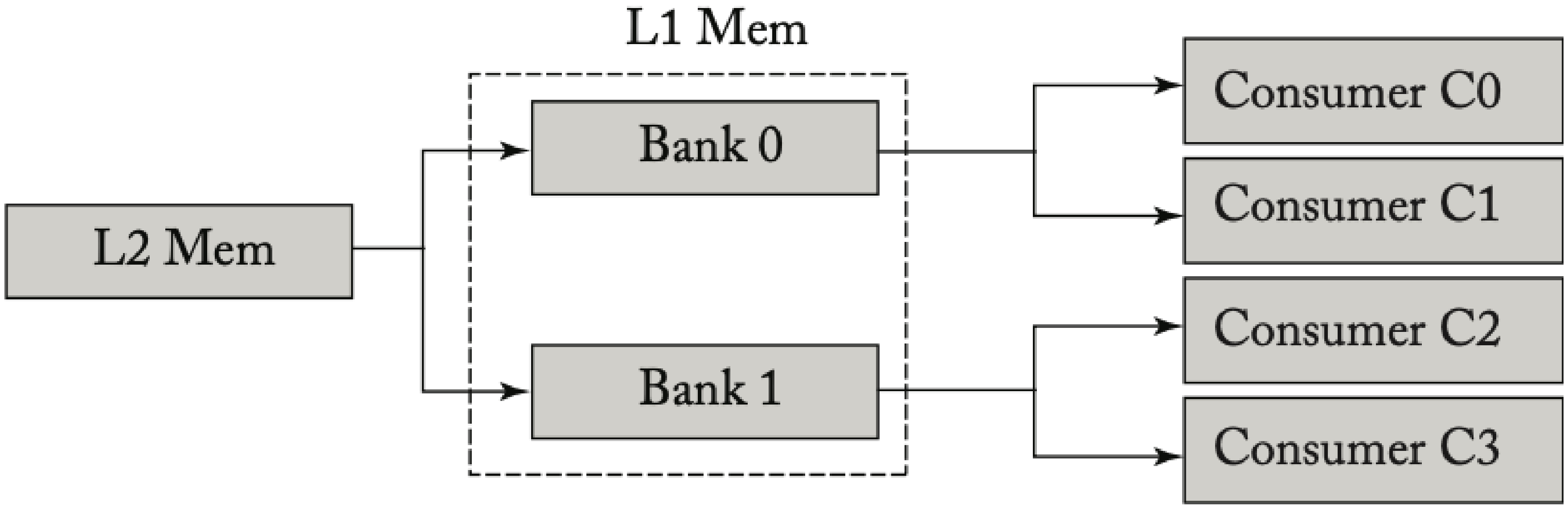
- Spatial reuse occurs when the same data value is used by more than one consumer (e.g., a group of PEs) at different spatial locations of the hardware
  - By reading the data once from the source memory level and multicasting it to all consumers
- Benefits
  - Reducing # accesses to the source memory level
    - Less energy cost
  - Reducing the bandwidth required from the source memory level
    - Helps to keep the PEs busy (higher utilization) ➔ higher performance

Reuse Distance for Spatial Reuse

- Reuse distance of spatial reuse
  - The maximum number of data accesses in between any pair of consumers that access the same data value
  - A function of the ordering of operations
- If group of consumers have no storage capacity
  - Spatial reuse by multicast in the same cycle
  - Data need to be resent when it is needed in a different cycle

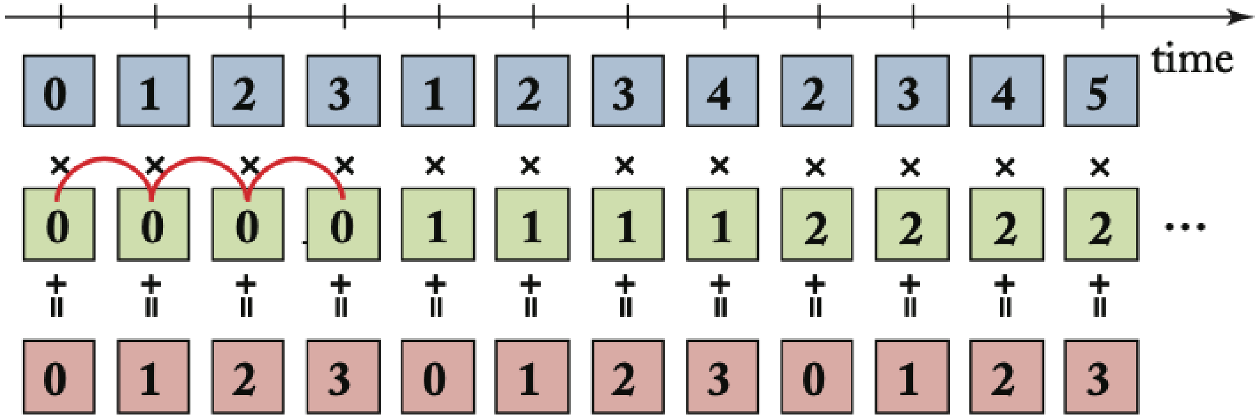
Physical Interconnection and Multi-Bank Memory

- Physical connectivity of the interconnection between L1 and consumers may limit the multicast from any banks in L1 to all consumers
- Data should be multicast from L2 to L1 ➔ data duplication

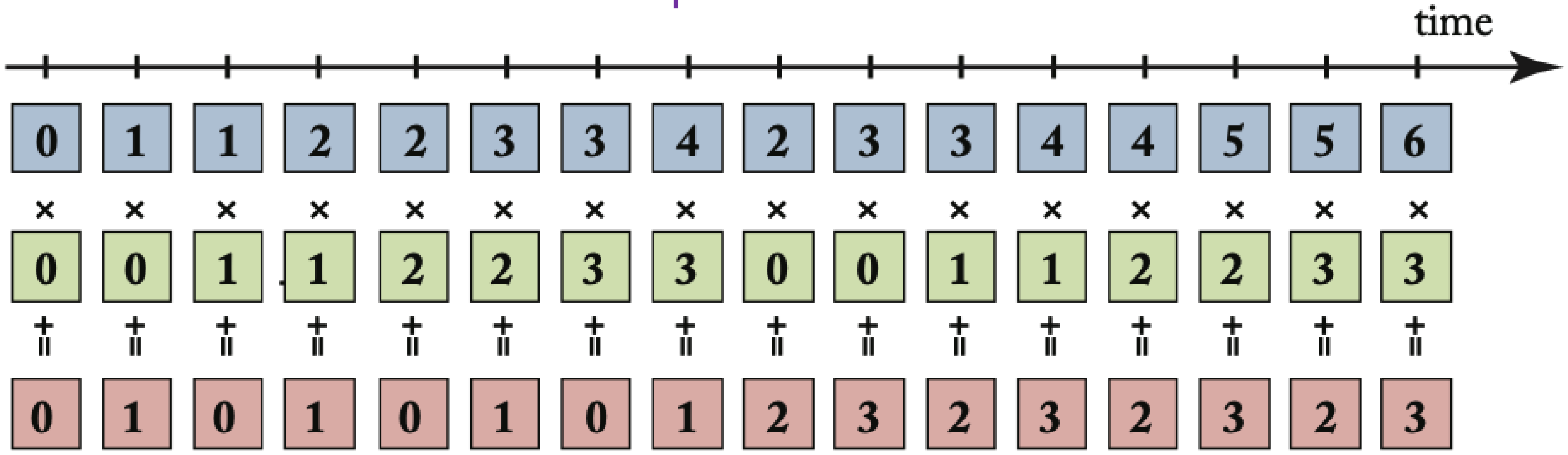


Reducing Reuse Distance

Concept of Temporal Tiling to Reduce Reuse Distance of Partial Sums

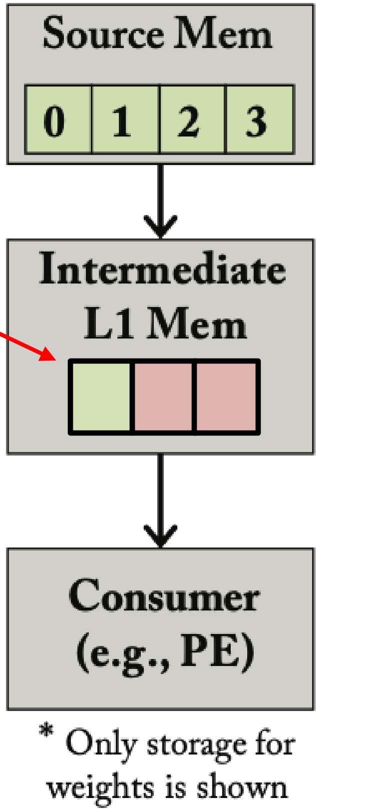


Untiled Temporal Reuse



Tiled Temporal Reuse

3 slots of intermediate memory



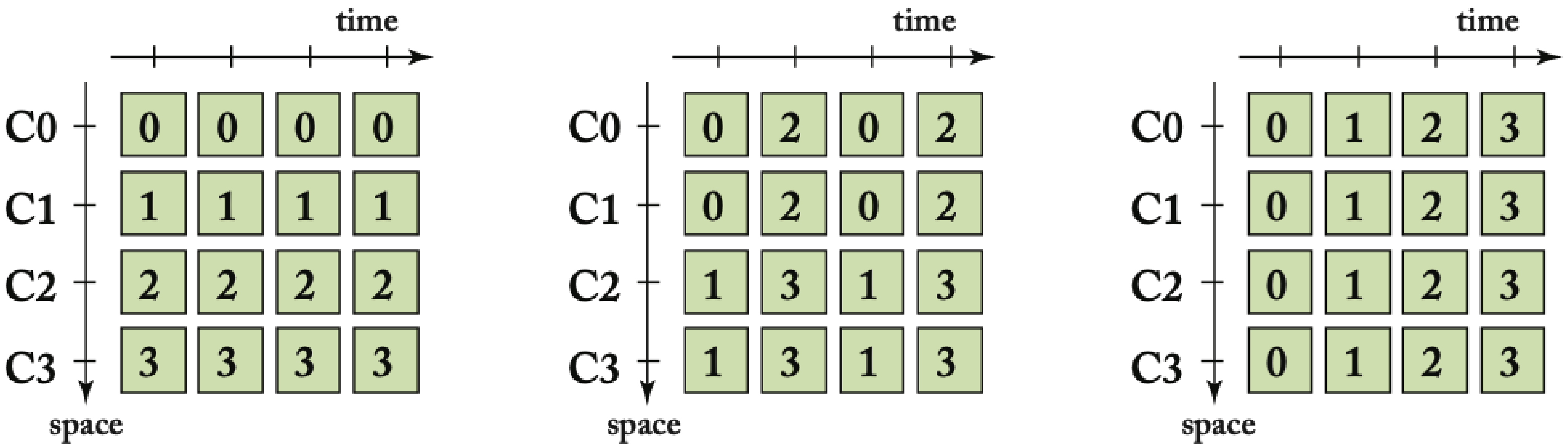
Memory Hierarchy

Tiling

- The goal is to tile (partition) the data so that the reuse distance becomes smaller
  - However, it is not feasible to minimize the reuse distance for all data types simultaneously
- Temporal tiling (tiling for temporal reuse)
  - Reducing the reuse distance of specific data types to make it smaller than the storage capacity of a certain memory level in the memory hierarchy

Spatial Tiling (Tiling for Spatial Reuse)

- Reusing the same data value by as many consumers as possible
- Reducing the reuse distance so that one multicast can serve the maximum possible consumers given a fixed amount of storage capacity at each consumer



(a) No spatial reuse

(b) Medium degree of spatial reuse

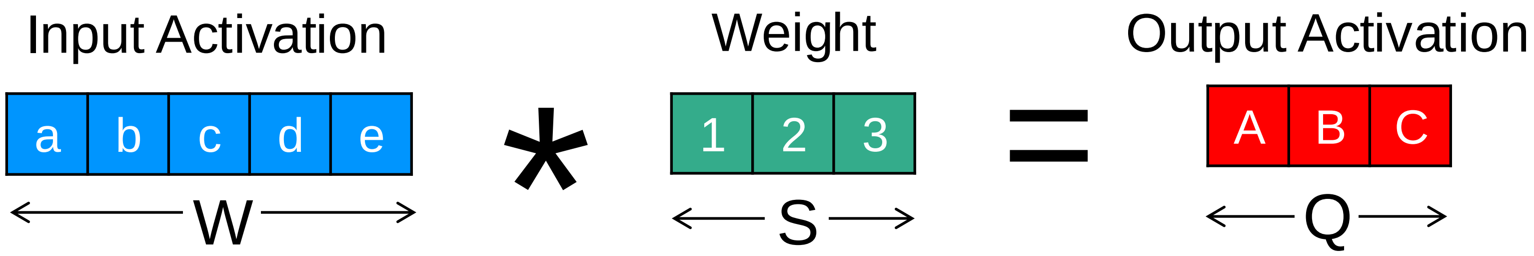
(c) High degree of spatial reuse

Dataflow

Dataflow

- Dataflow refers to how data is processed within the hardware architecture
  - Determines the path that data moves and how it is transformed and manipulated through the system
  - Defines the execution order of the DNN operations in hardware
    - Computation order
    - Data movement order
- Loop nest is a compact way to describe the dataflow supported in hardware
  - `for` (temporal for): describes the temporal execution order
  - `parallel_for` (spatial for): describes the parallel execution

Weight-Stationary (WS) Dataflow and Output-Stationary (OS) Dataflow



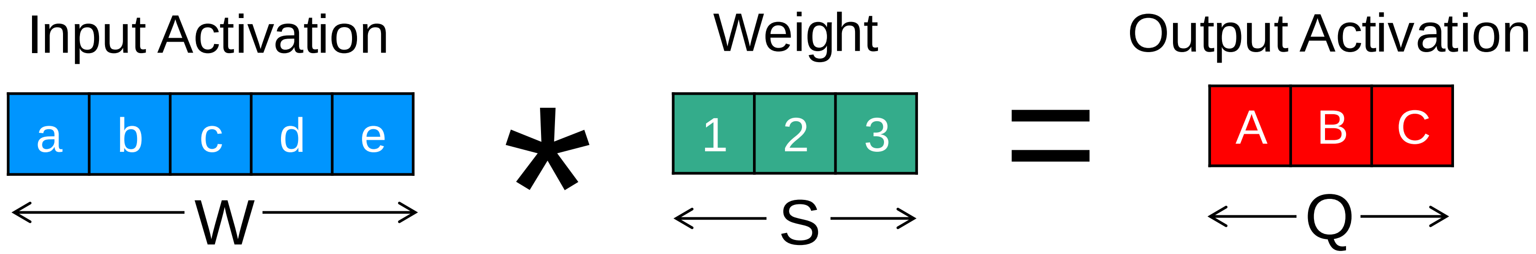

```
for s in range(S):
  for q in range(Q):
    w = q+s;
    OA[q] += IA[w] * W[s];
```

Weight-Stationary (WS) Dataflow

```
for q in range(Q):
  for s in range(S):
    w = q+s;
    OA[q] += IA[w] * W[s];
```

Output-Stationary (OS) Dataflow

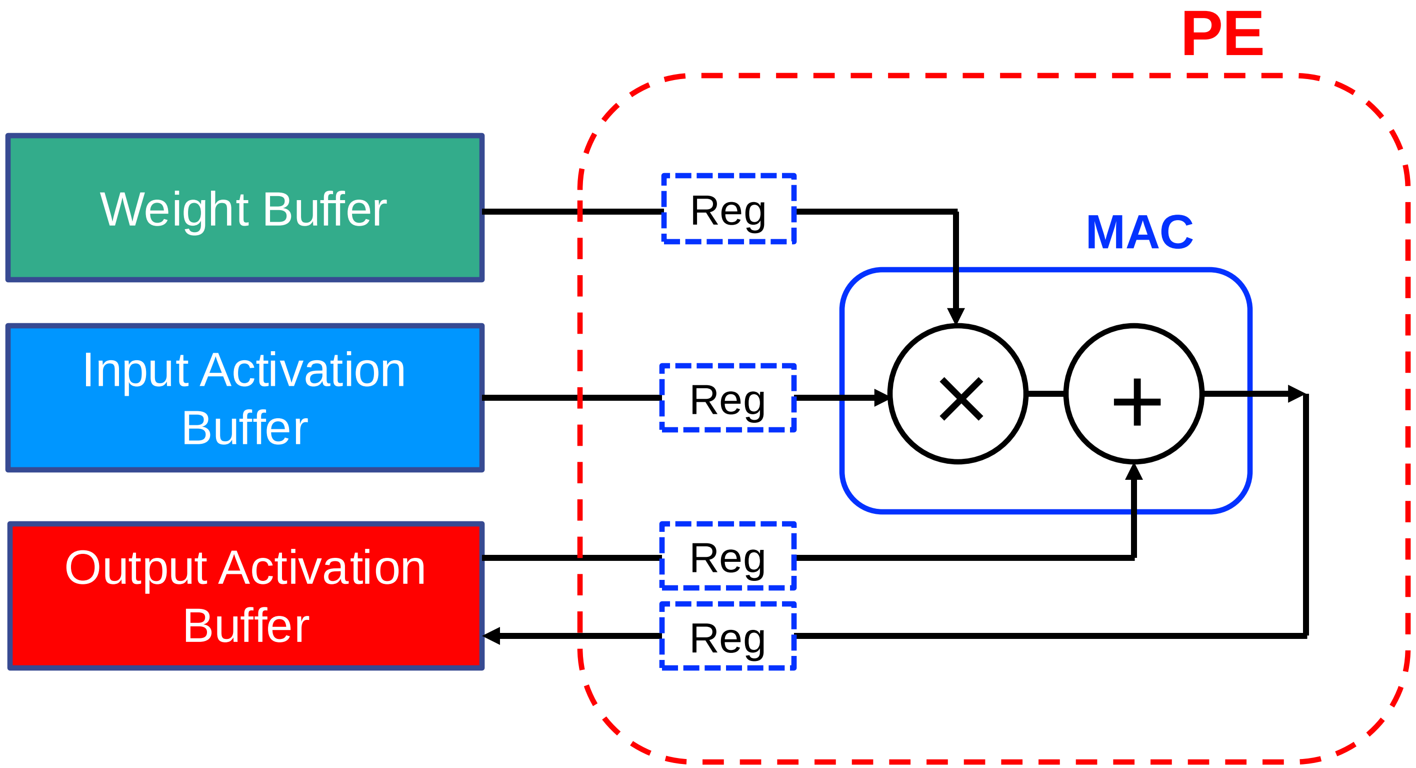
More Dataflow: Input-Stationary (IS)




```
for w in range(W):
  for s in range(S):
    q = w-s;
    OA[q] += IA[w] * W[s];
```

Input-Stationary (IS) Dataflow

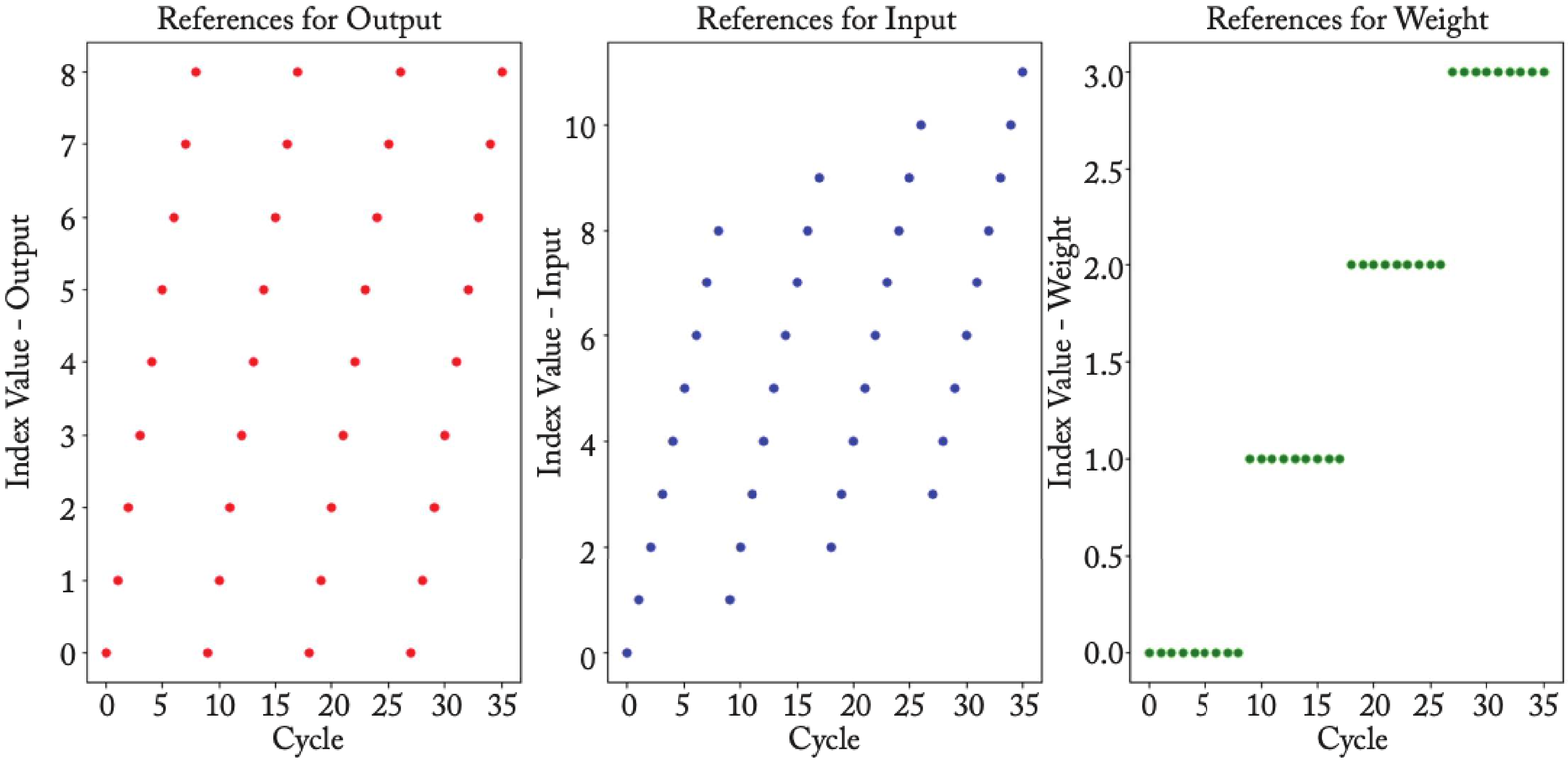
Model of Single Processing Element (PE)



Space-Time Diagram for Weight-Stationary (WS) Dataflow

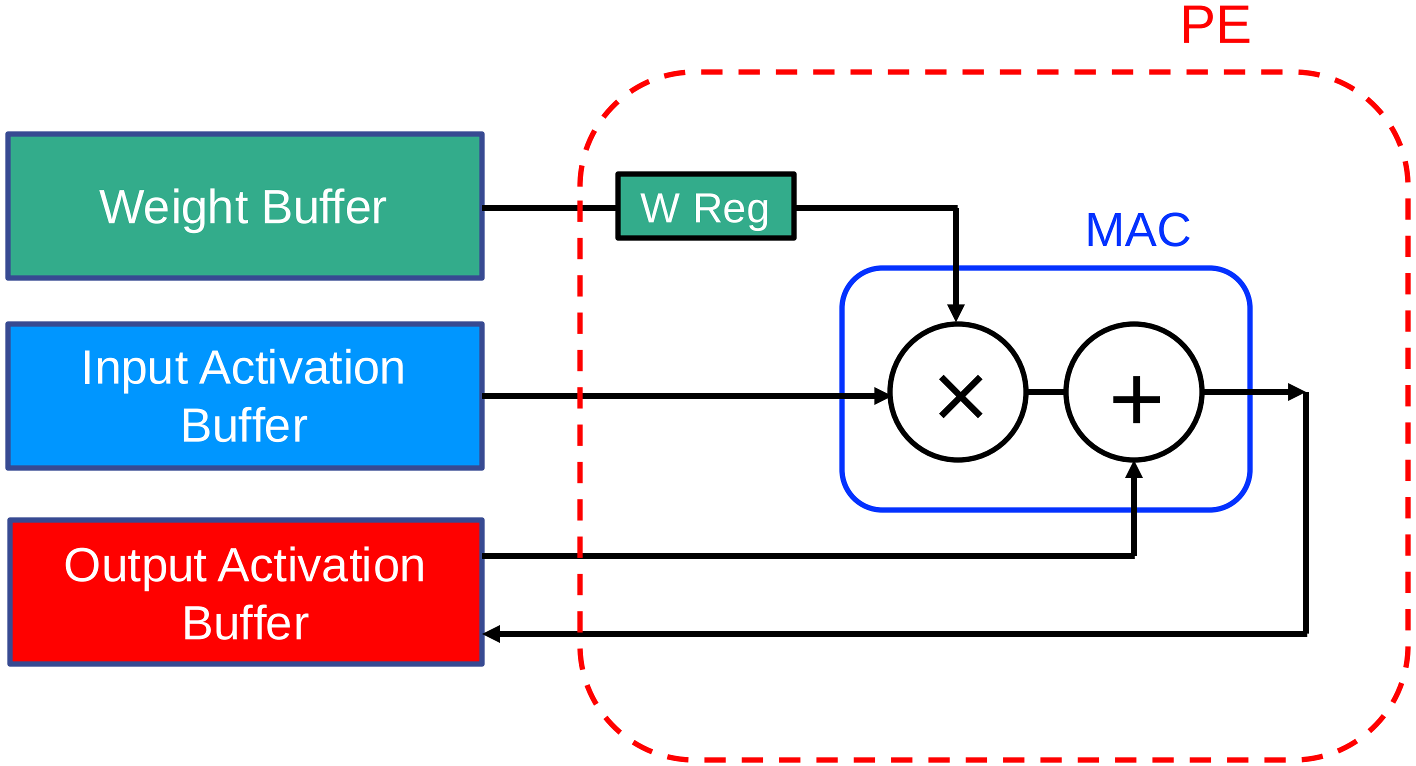
```
for s in range(S):
  for q in range(Q):
    w = q+s
    OA[q] += IA[w] * W[s];
```

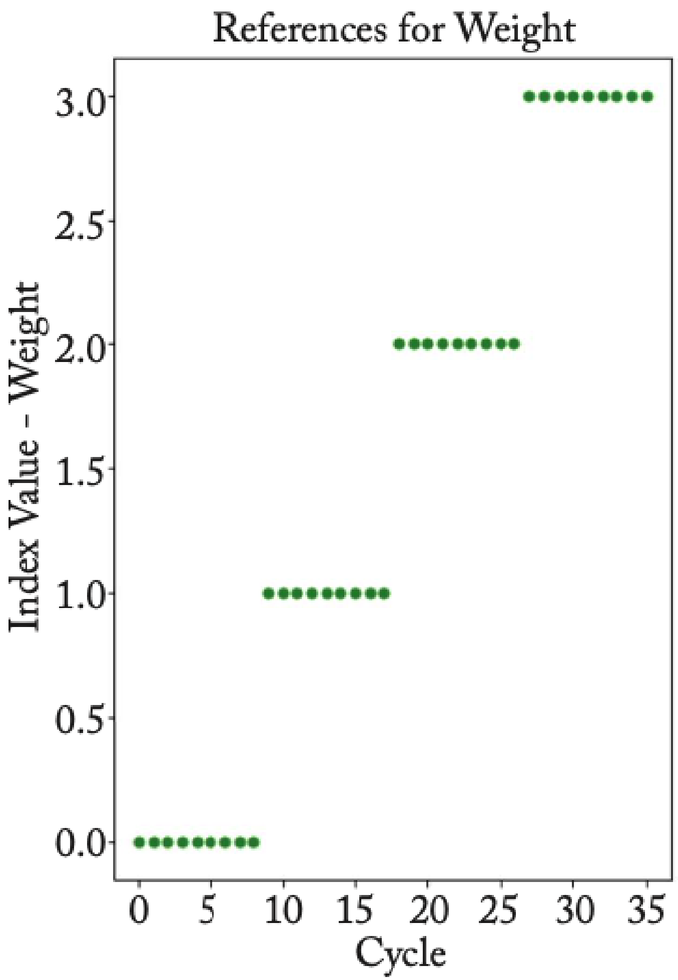
$Q = 9, W = 12, S = 4$



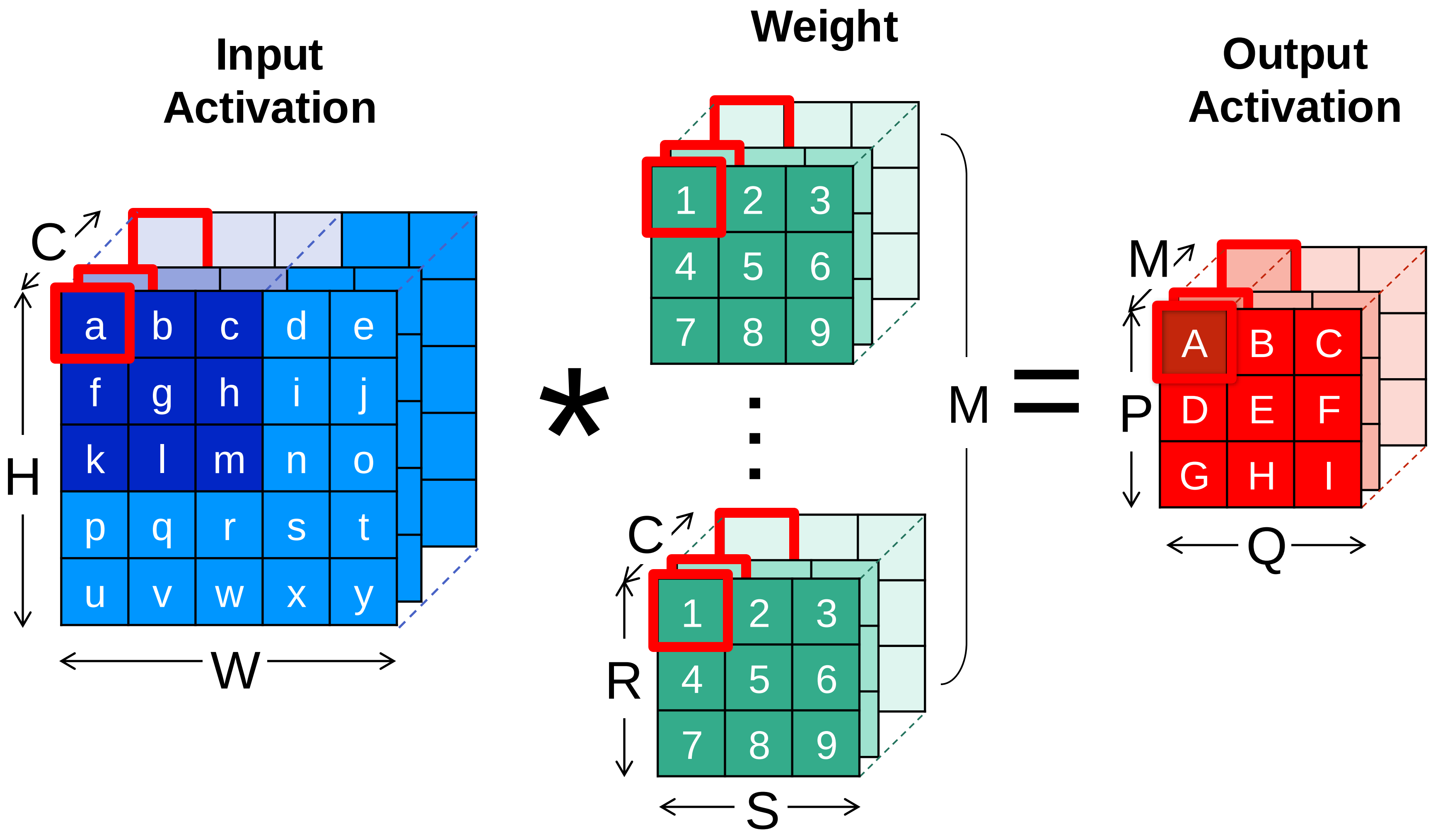
Space-Time Diagram for Weight-Stationary (WS) Dataflow

Observation: Single weight reused *Q* times

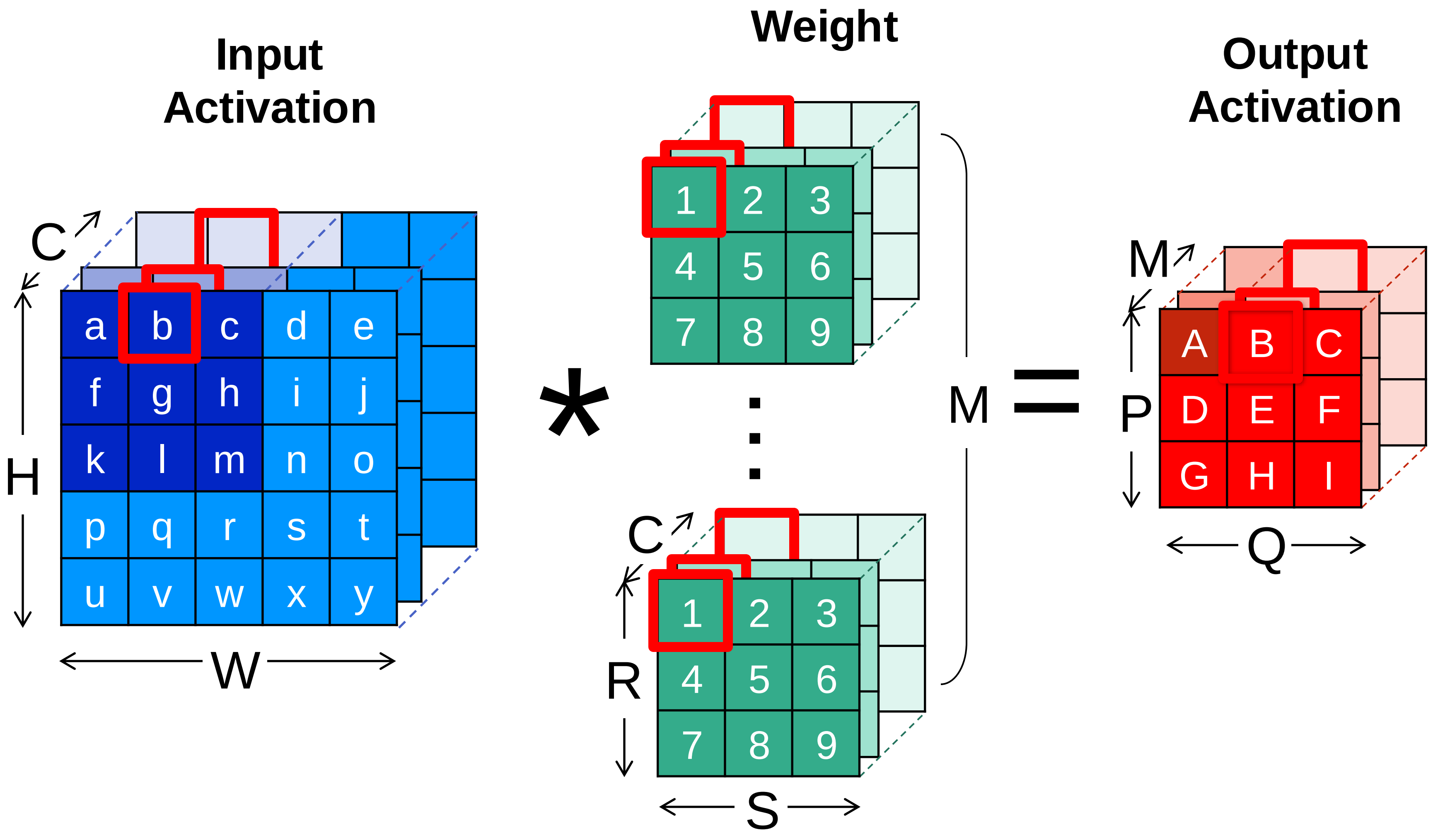




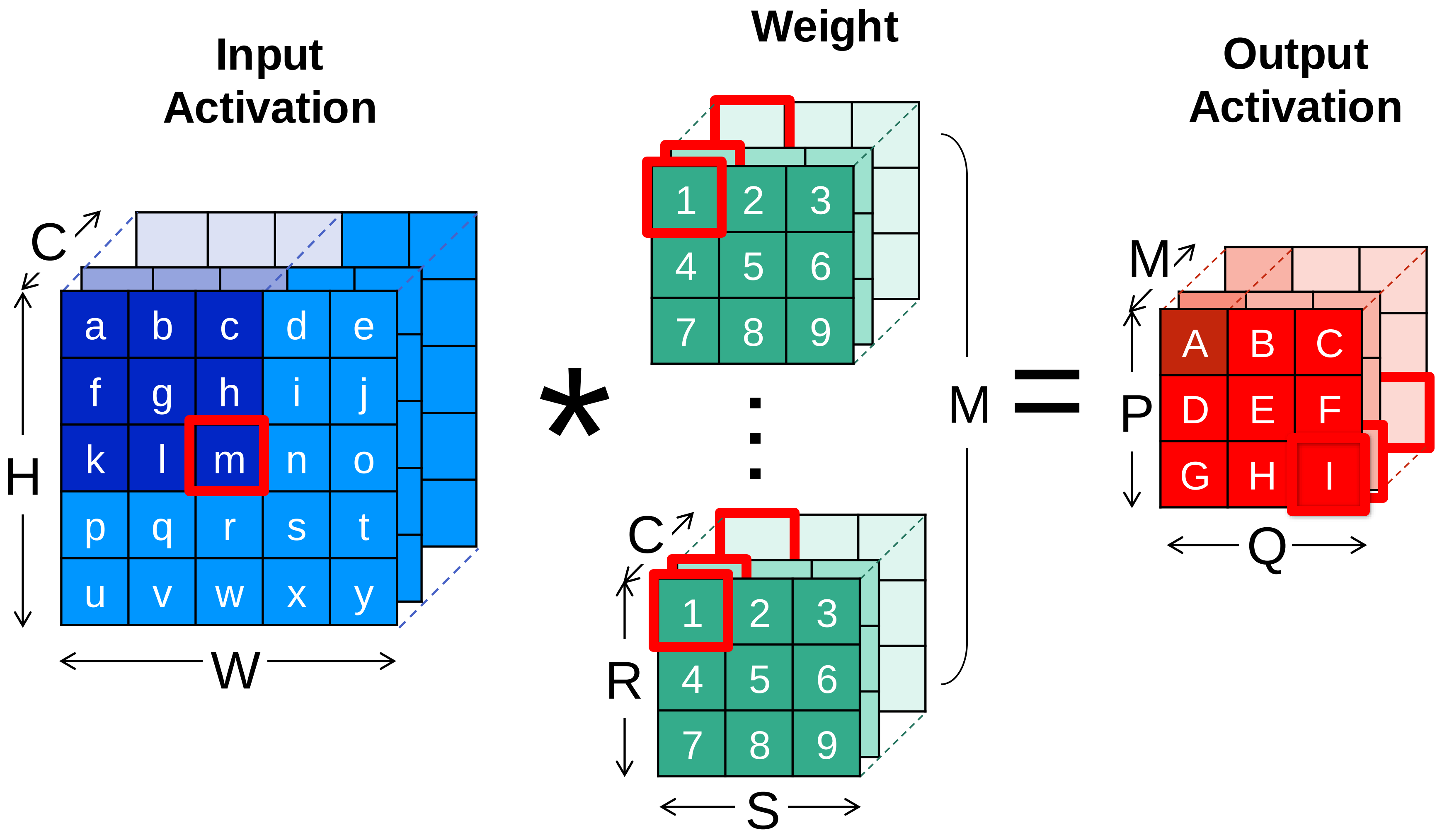
Weight-Stationary Dataflow



Weight-Stationary Dataflow



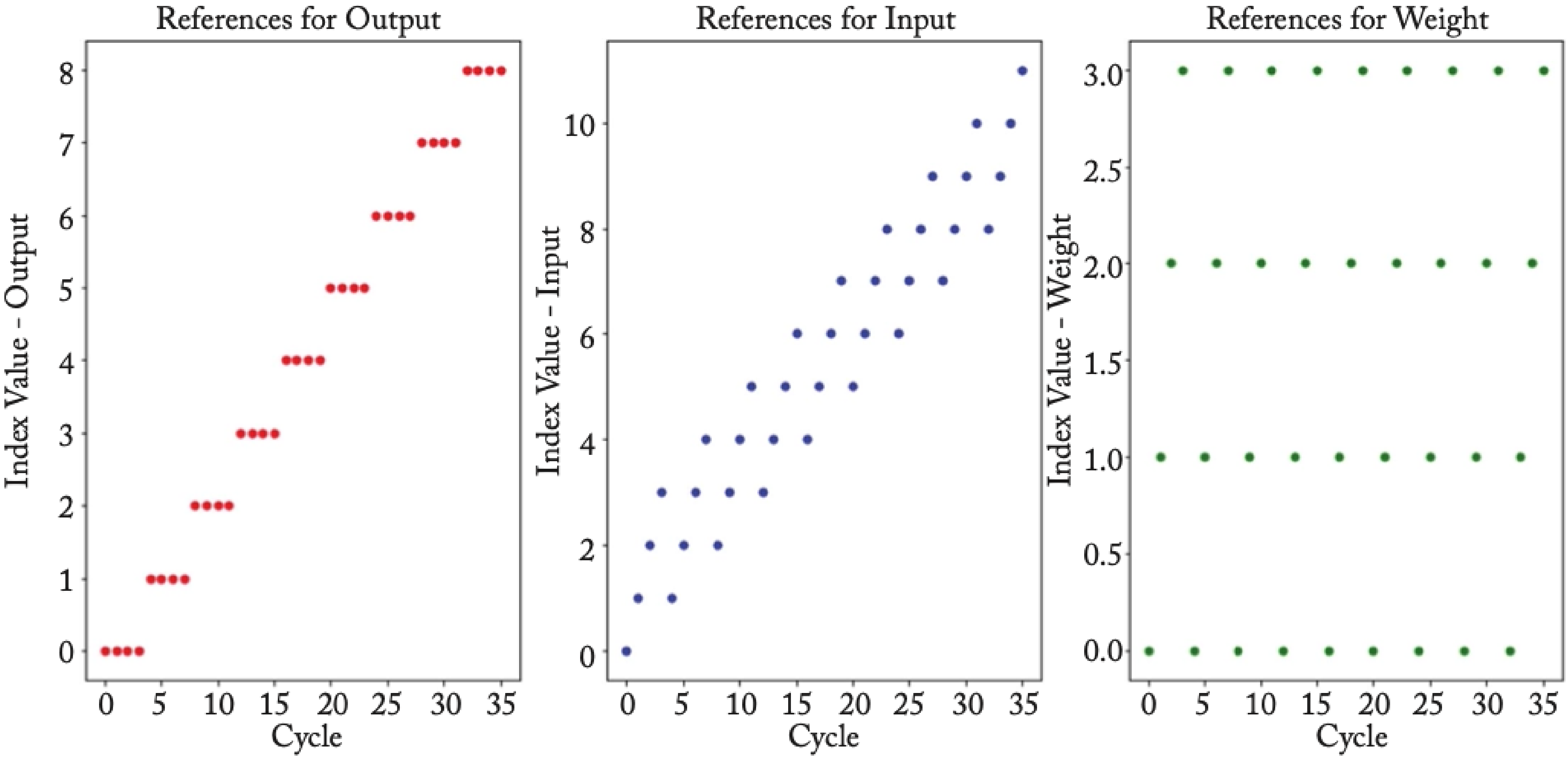
Weight-Stationary Dataflow



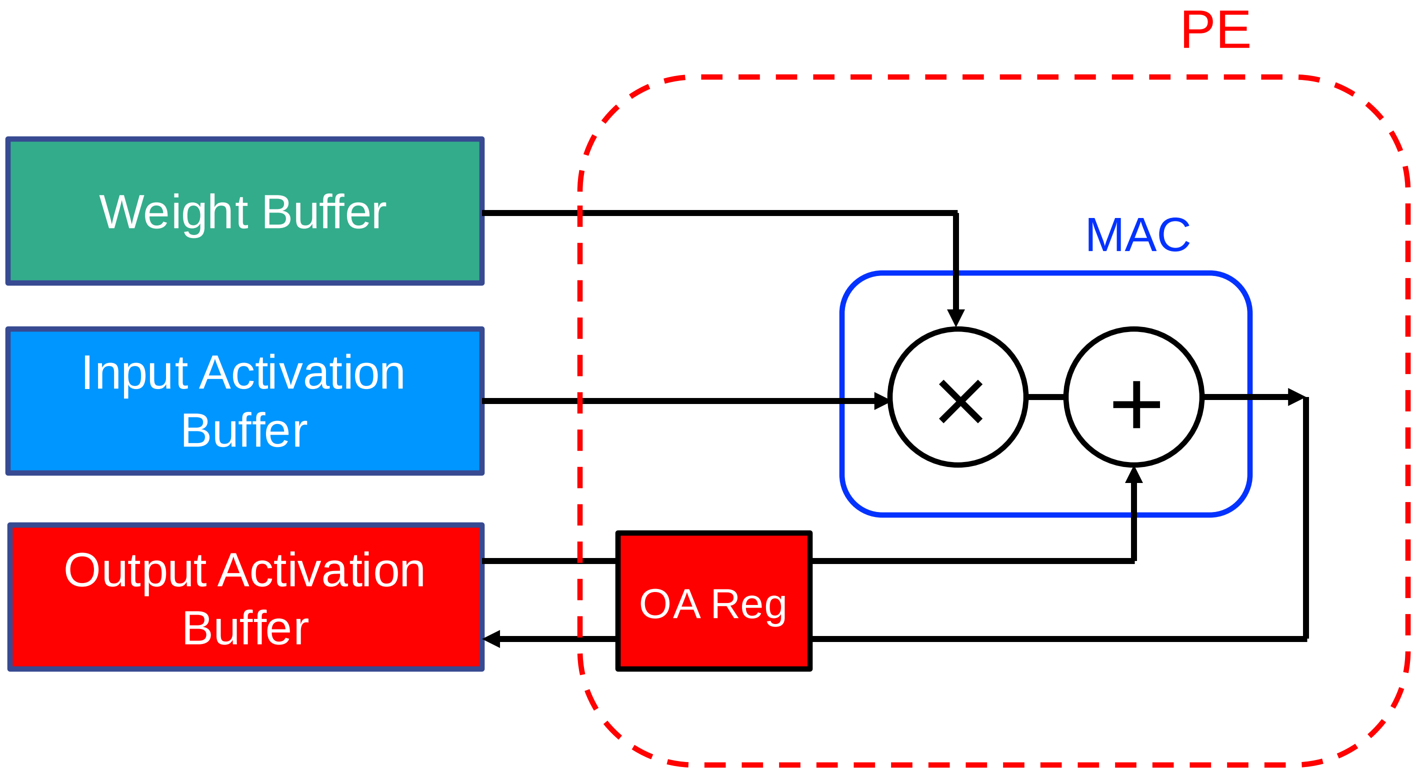
Space-Time Diagram for Output-Stationary (OS) Dataflow

```
for q in range(Q):
  for s in range(S):
    w = q+s
    OA[q] += IA[w] * W[s];
```

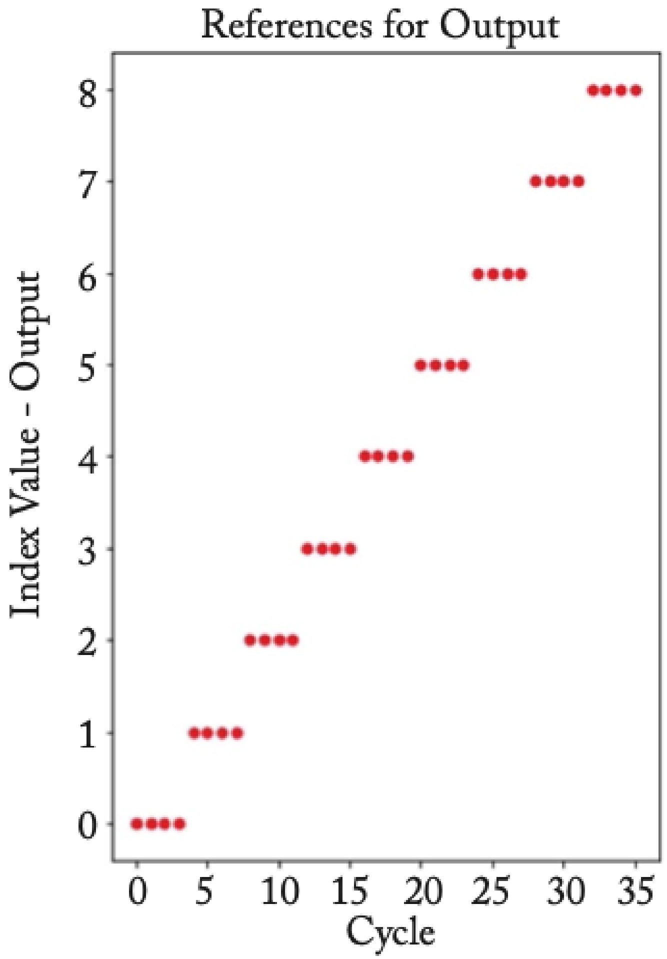
$Q = 9, W = 12, S = 4$



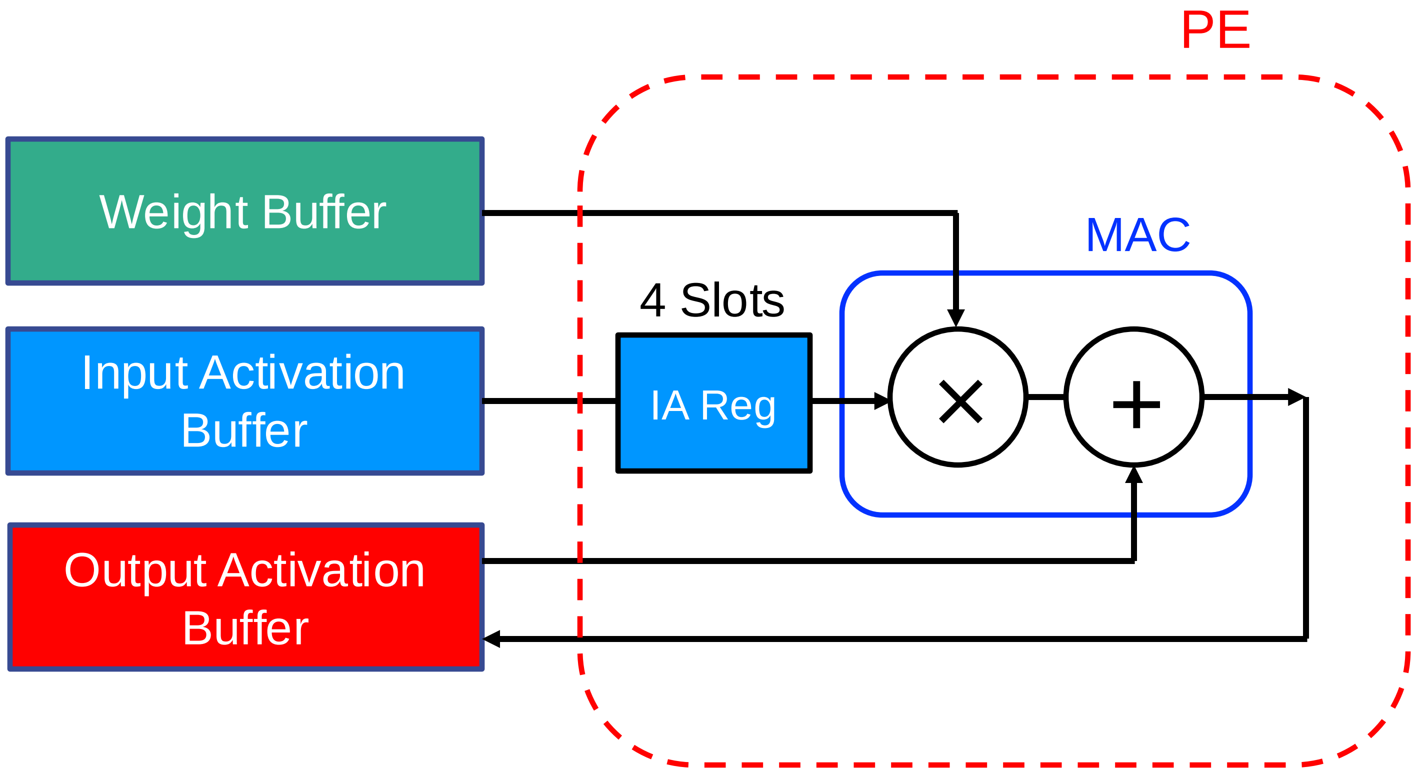
Space-Time Diagram for Output-Stationary (OS) Dataflow

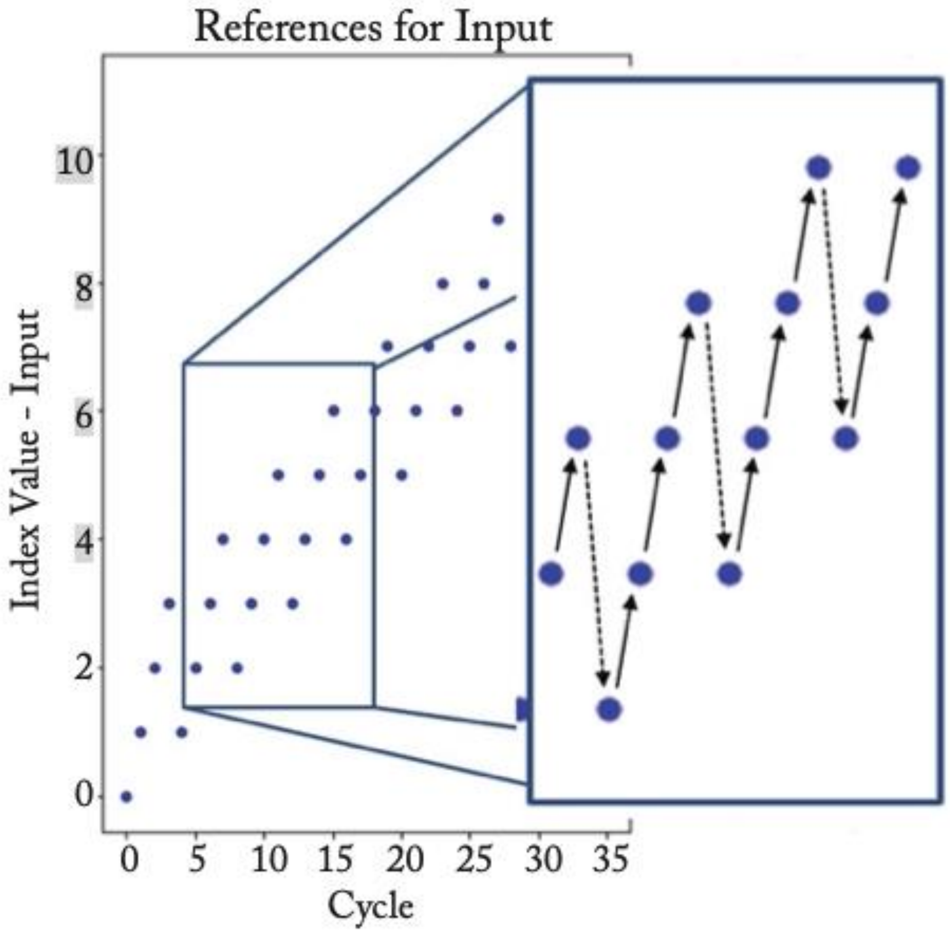


Observation: Single output reused *S* times

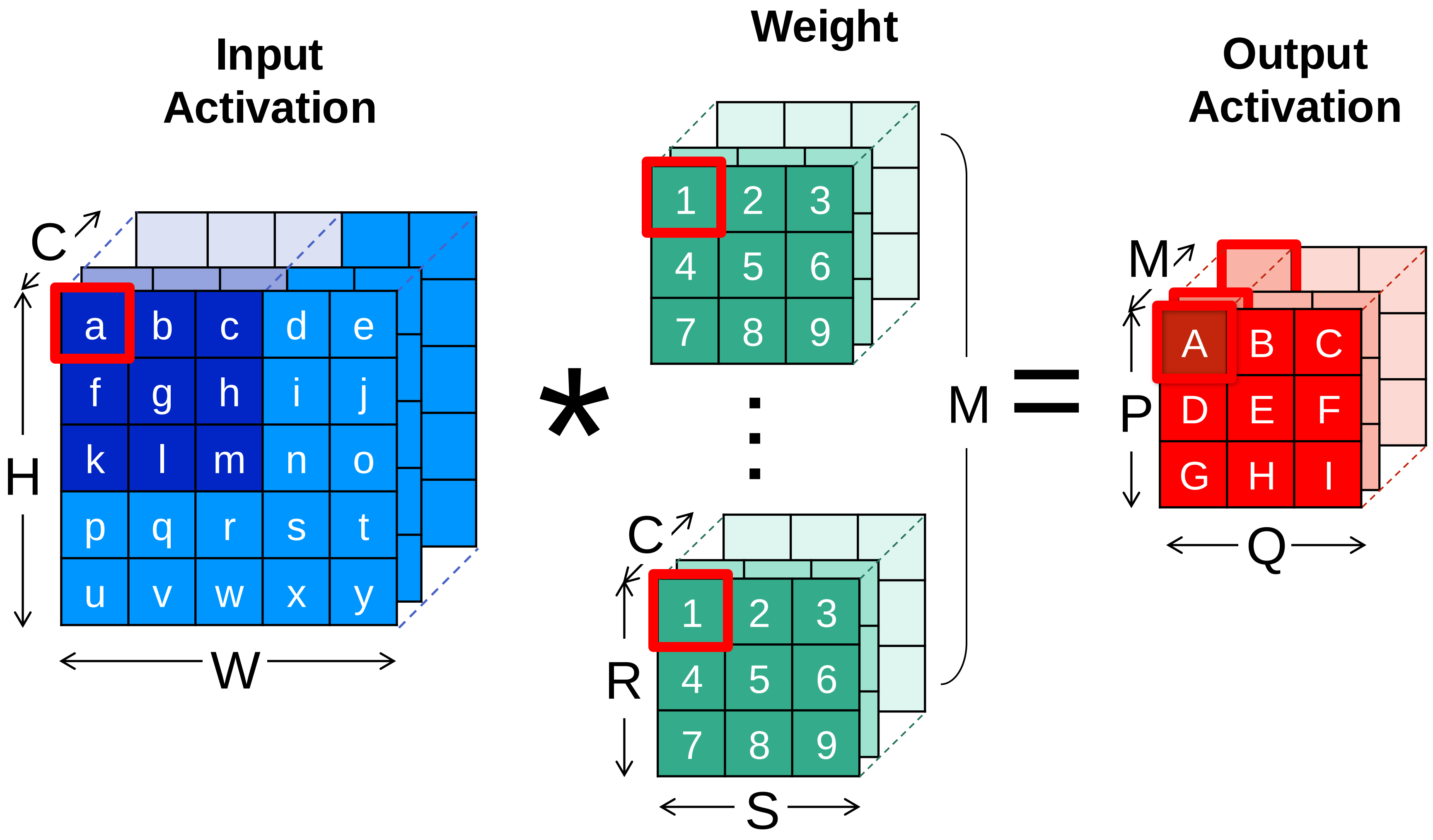


Sliding Window in Output-Stationary

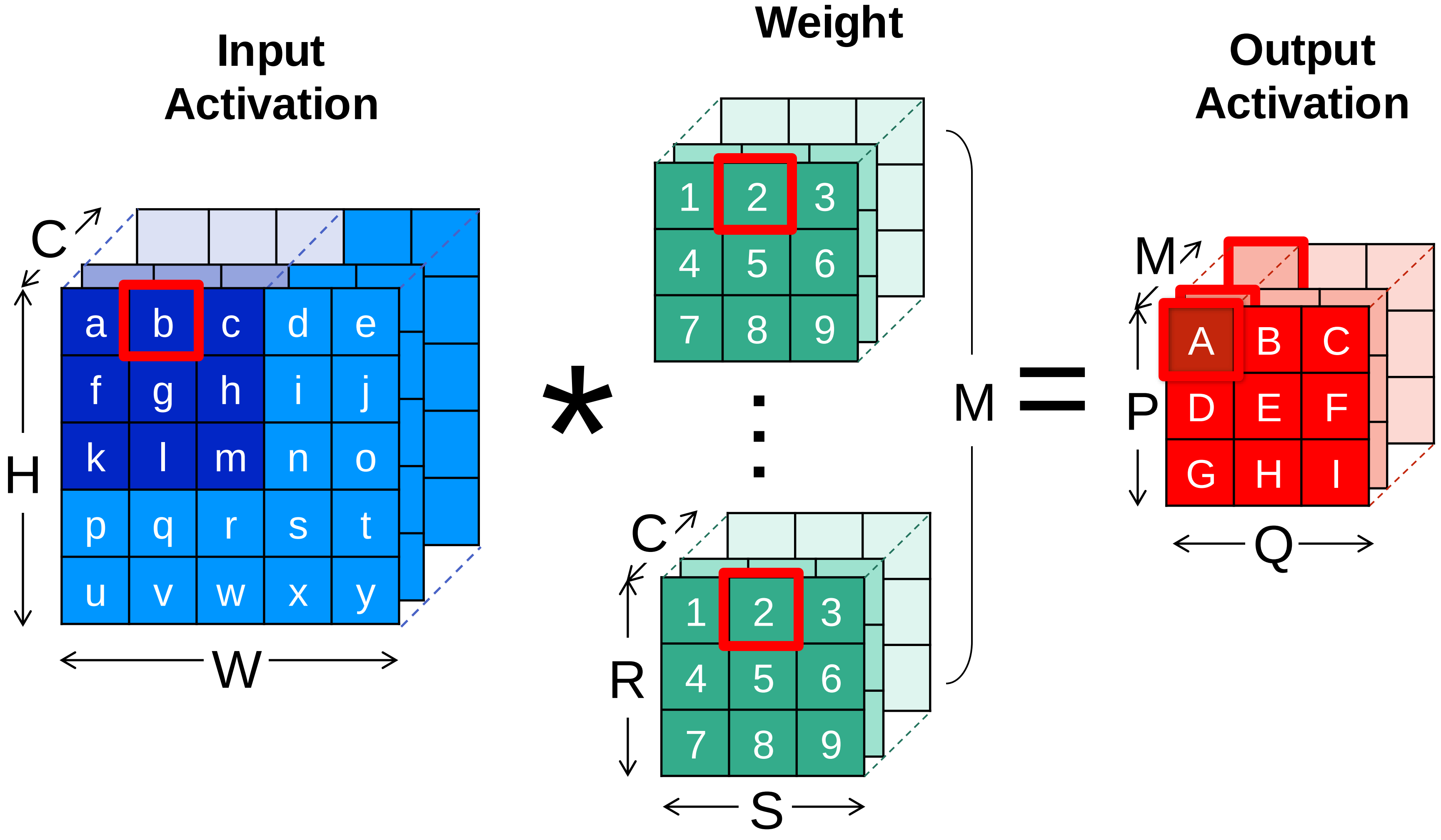




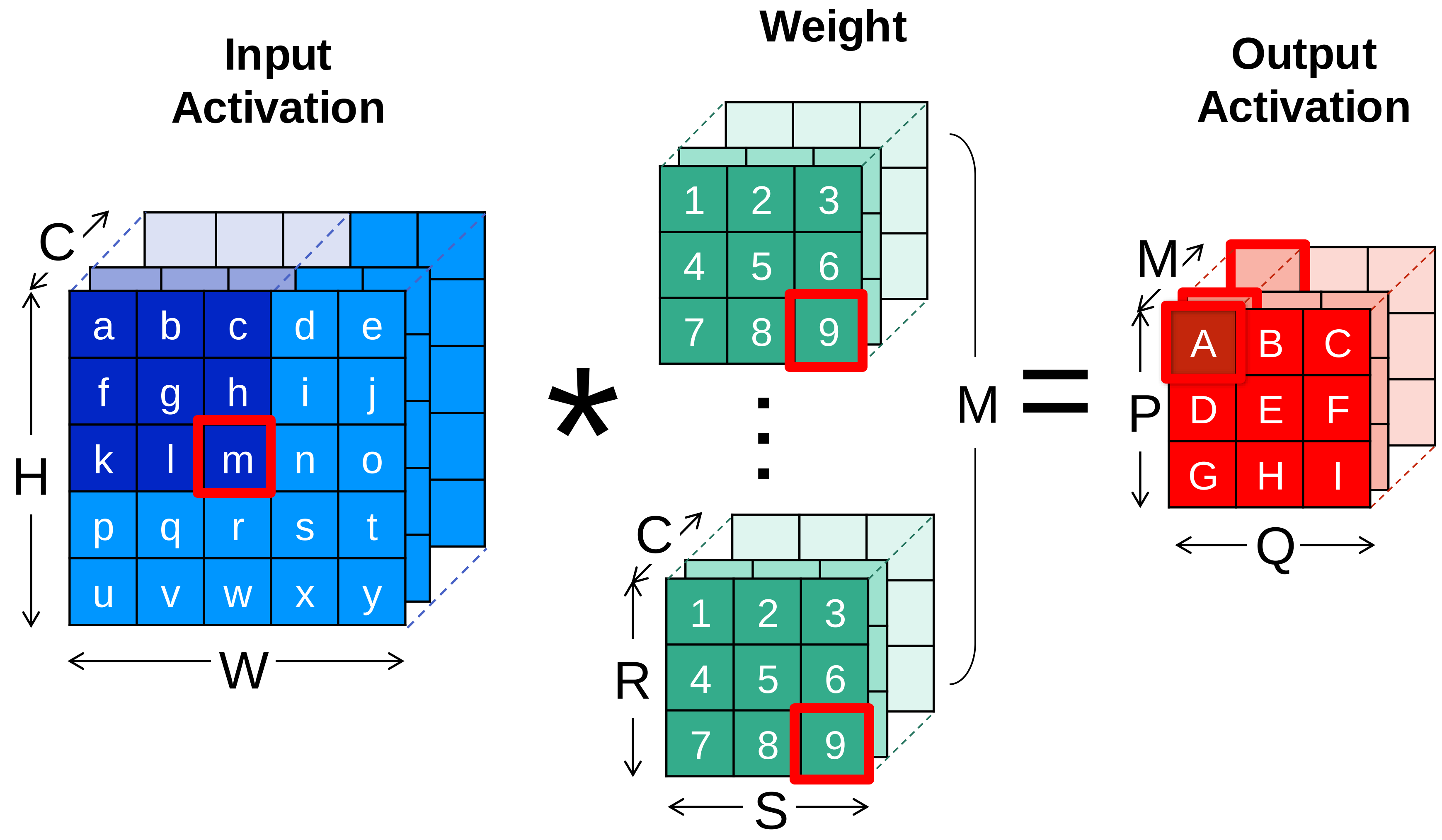
Output-Stationary Dataflow



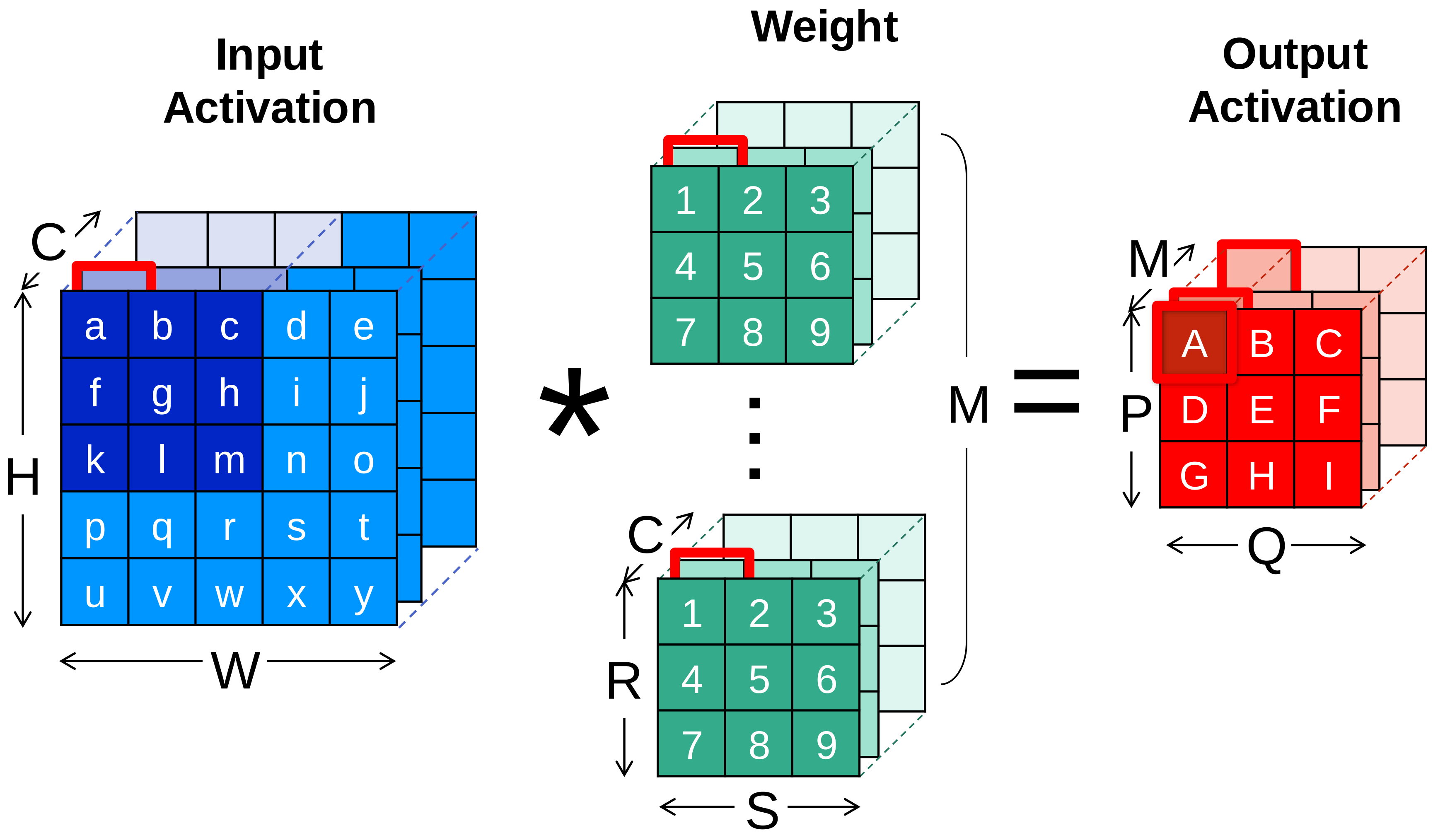
Output-Stationary Dataflow



Output-Stationary Dataflow

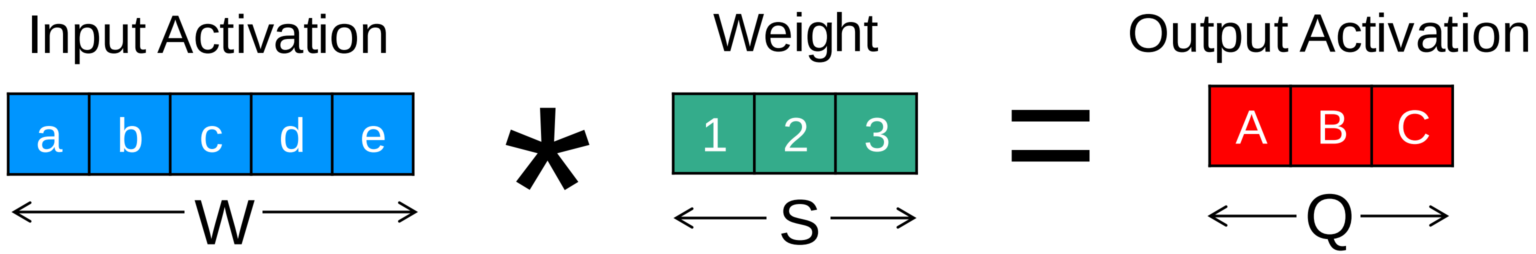


Output-Stationary Dataflow



Tiled Loop Nest for Dataflow

Tiled Loop Nest for Weight-Stationary Dataflow



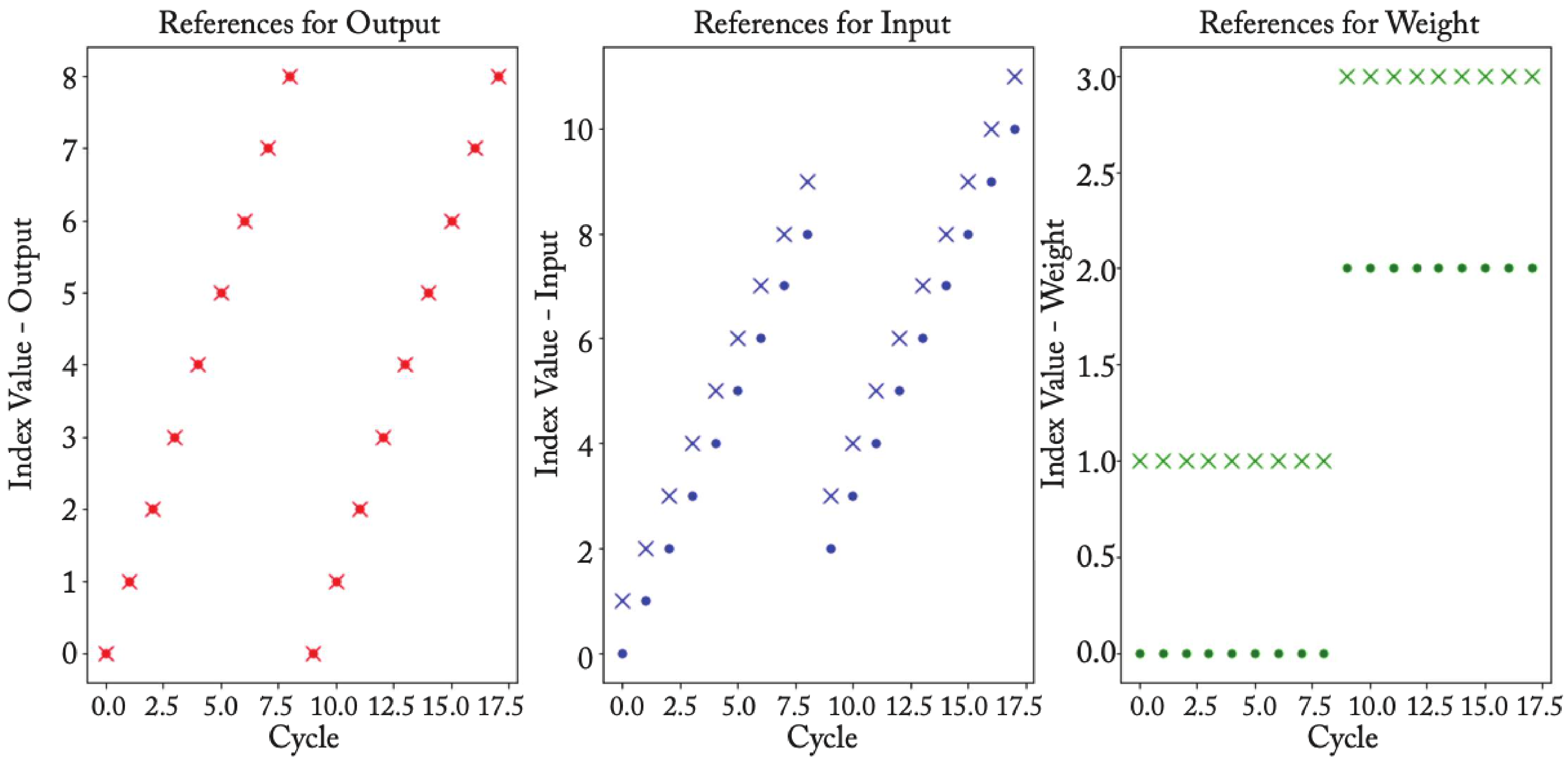
$Q = 9, W = 12, S = 4; S \text{ is tiled into } S1 = 2, S0 = 2$

```
for s1 in range(S1):
  for q in range(Q):
    parallel_for s0 in range(S0):
      s = s1*S0 + s0;
      w = q+s;
      OA[q] += IA[w] * W[s];
```

Space-Time Diagrams of Tiled WS Dataflow

```
for s1 in range(S1):
  for q in range(Q):
    parallel_for s0 in range(S0):
      s = s1*S0 + s0; w = q+s;
      OA[q] += IA[w] * W[s];
```

$Q = 9, W = 12, S = 4; S$ is tiled into $S1 = 2, S0 = 2$



Summary

- Parallelizing the compute may reduce the latency
- Minimizing data movement is the key to high energy efficiency for DNN accelerators
  - Maximizing data reuse at the higher level of memory hierarchy
- 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 properties of a dataflow
  - E.g., **data tiling** in multi-level storage and temporal/spatial processing