6.5930/1

Hardware Architectures for Deep Learning

Accelerator Architecture (continued)

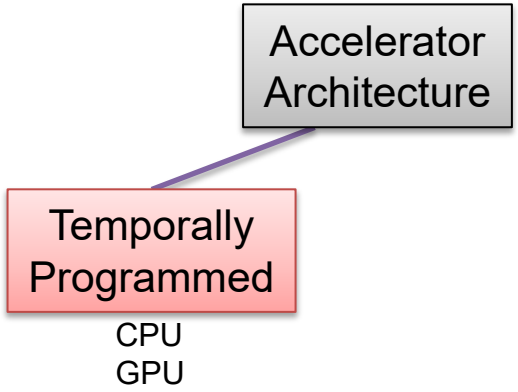
March 11, 2024

Joel Emer and Vivienne Sze

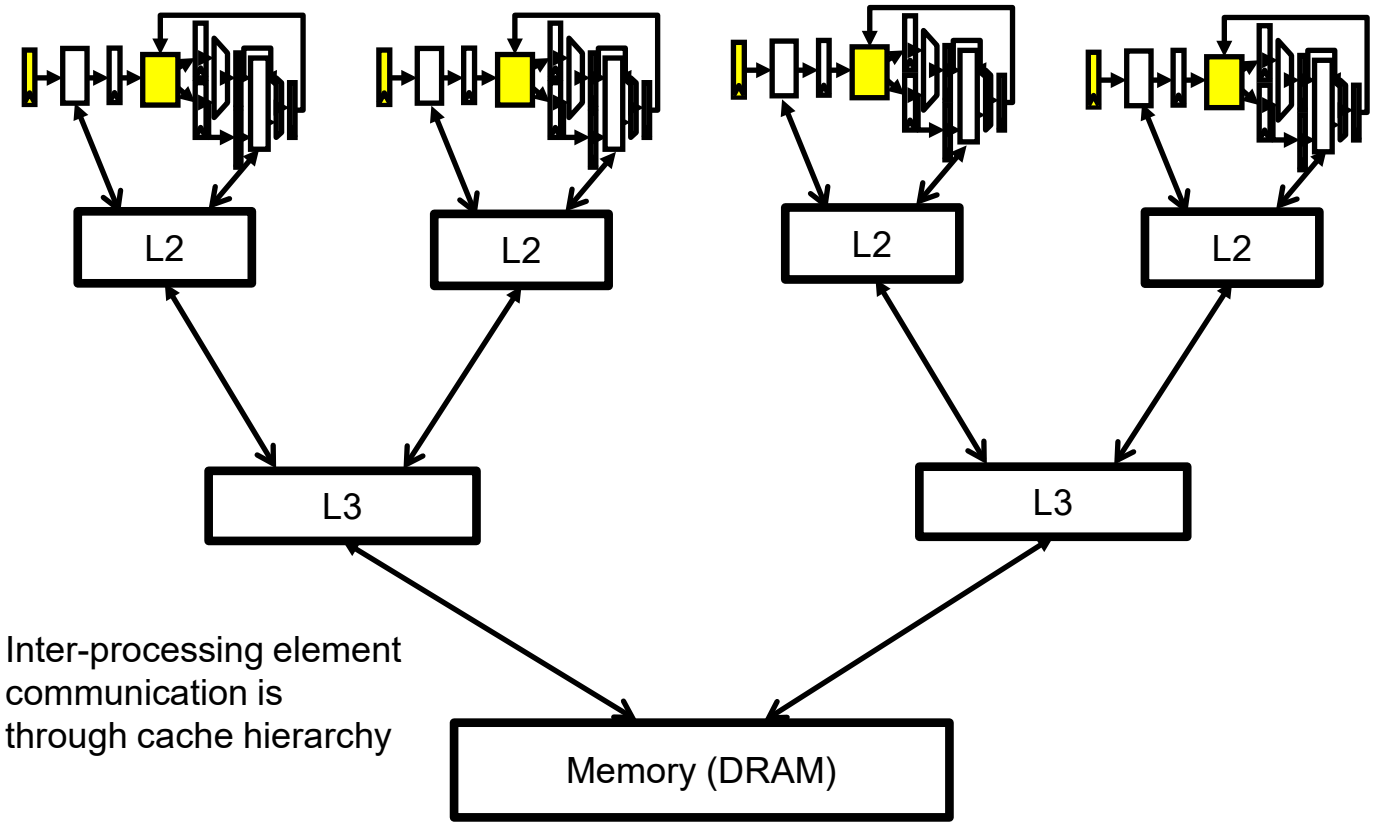
Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Operation Sequencing

Accelerator Taxonomy



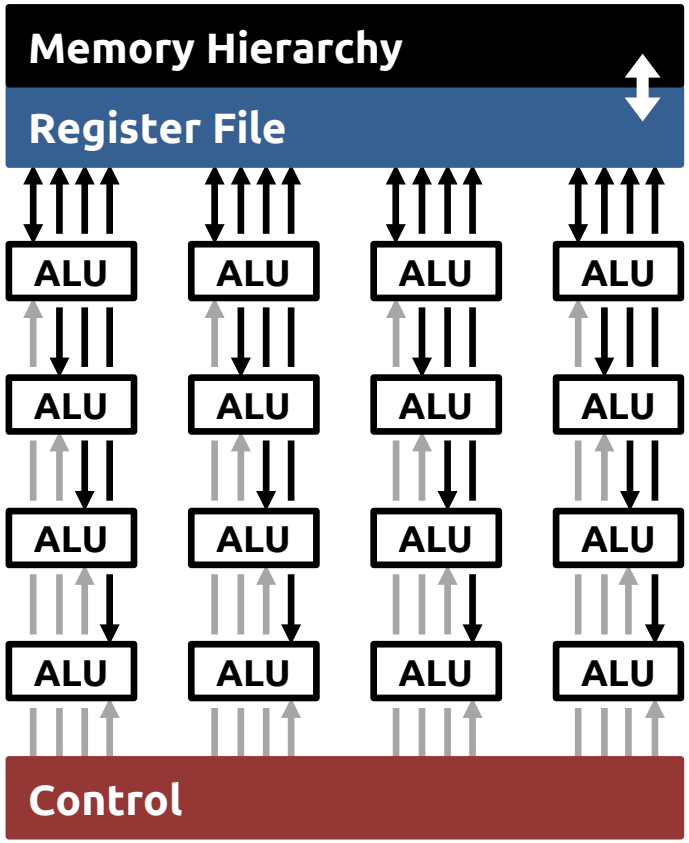
Multiprocessor



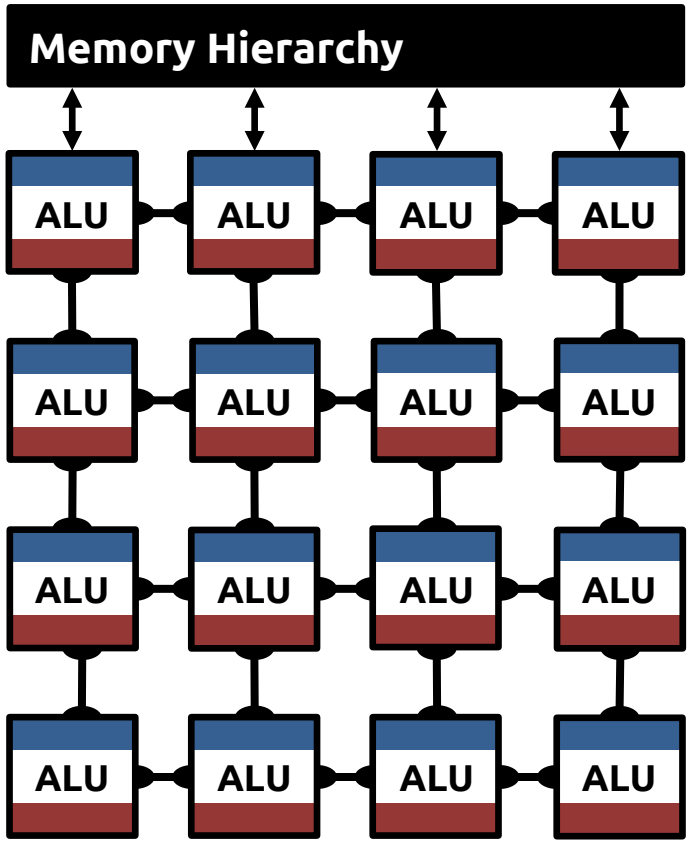
Inter-processing element communication is through cache hierarchy

Highly-Parallel Compute Paradigms

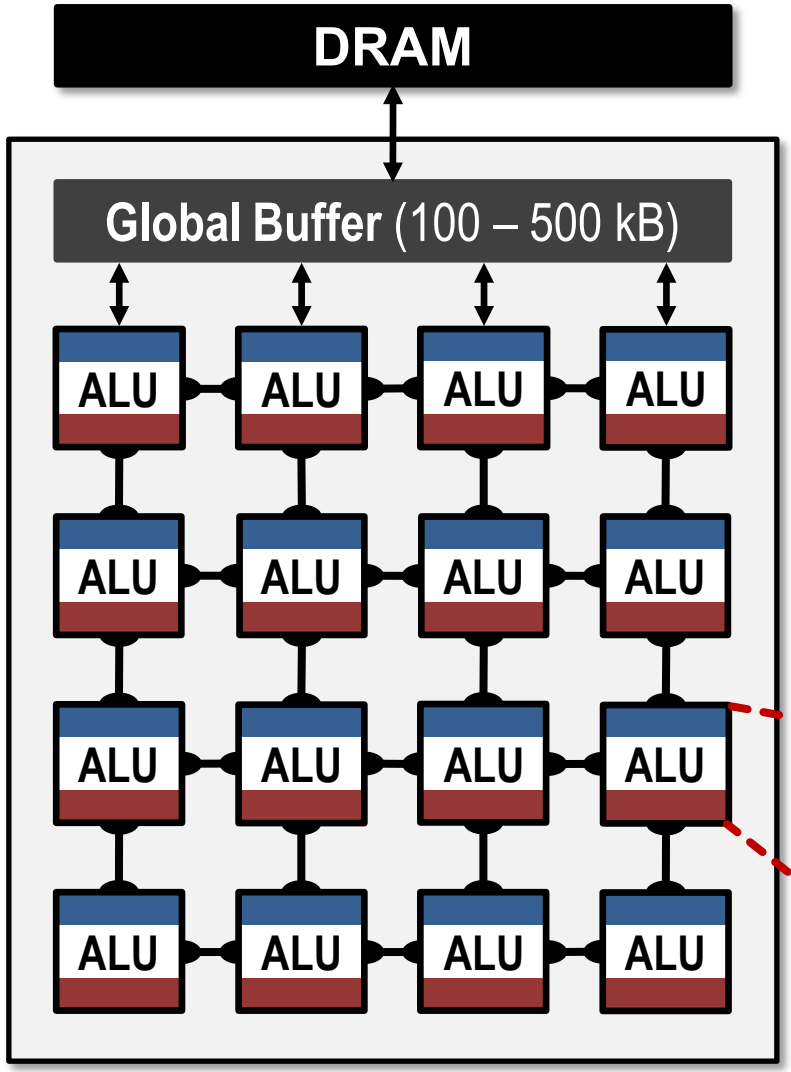
Temporal Architecture (SIMD/SIMT)



Spatial Architecture (Dataflow Processing)



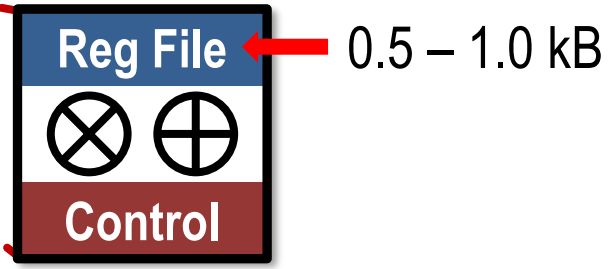
Spatial Architecture for DNN



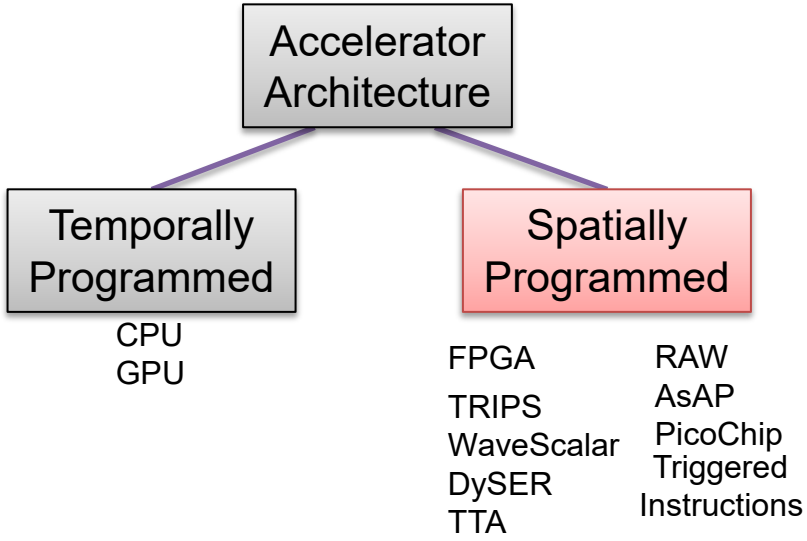
Local Memory Hierarchy

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

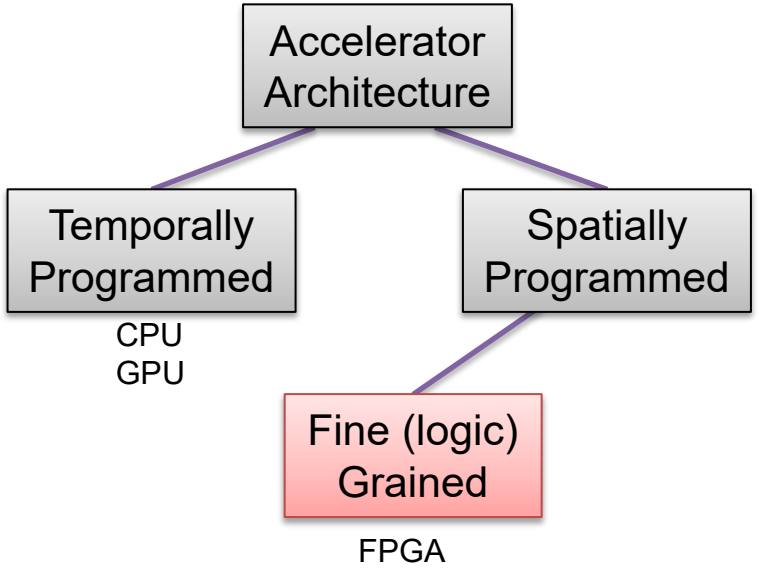
Processing Element (PE)



Accelerator Taxonomy

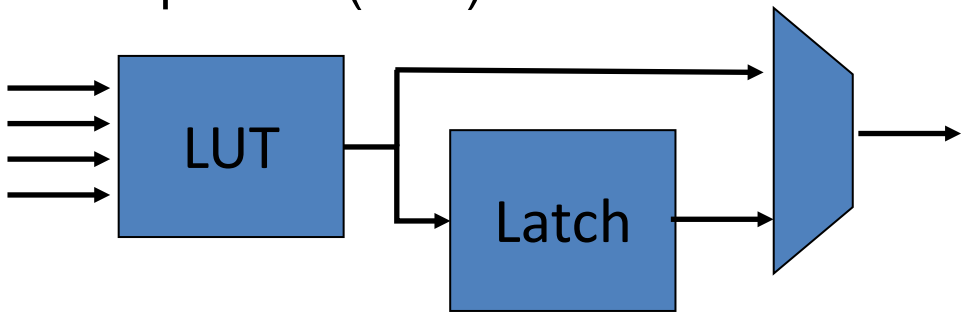


Accelerator Taxonomy



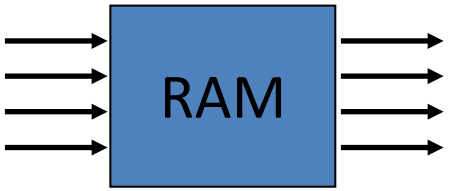
Field Programmable Gate Arrays

Look Up Table (LUT)

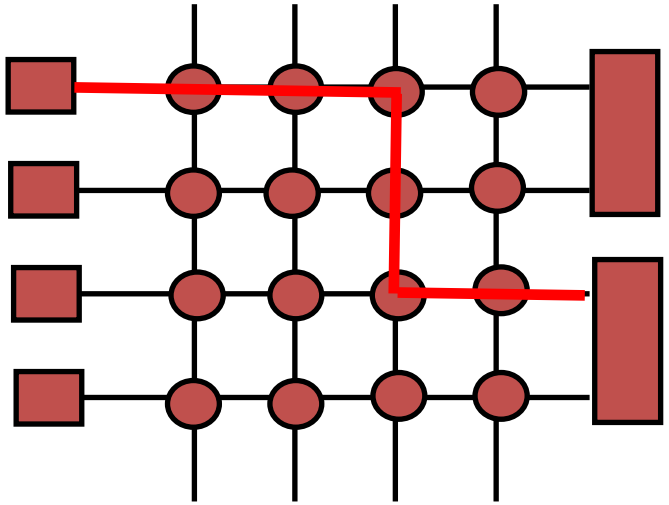


And	
00	0
01	0
10	0
11	1

Or	
00	0
01	0
10	1
11	1

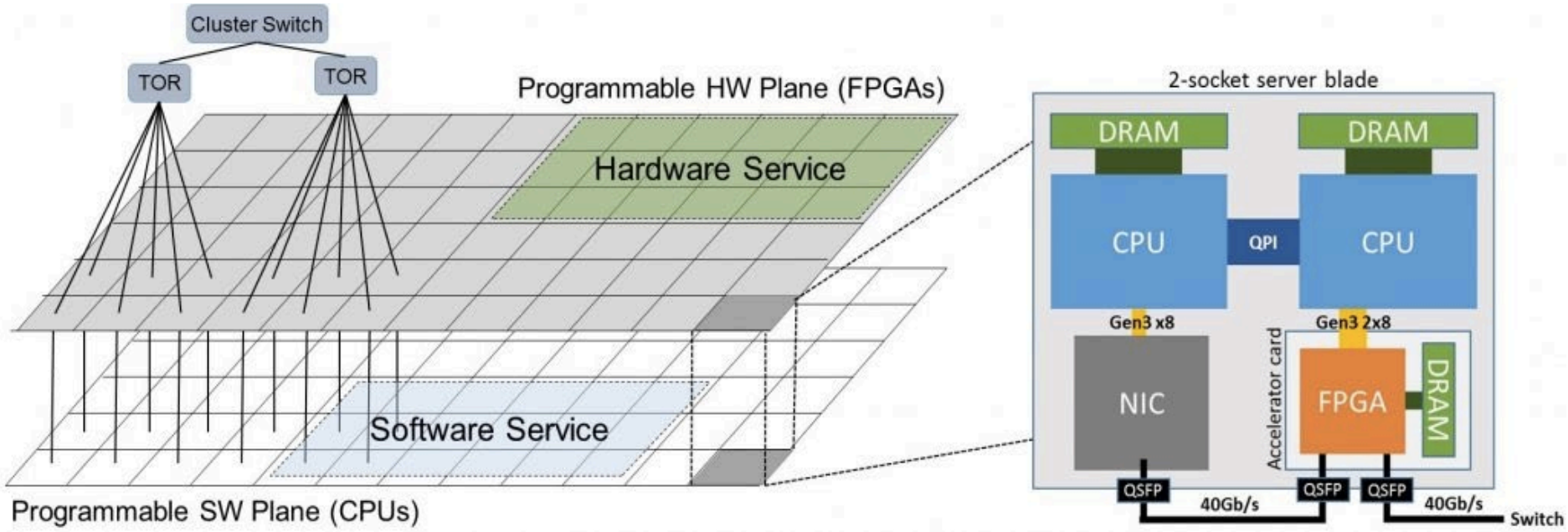


.....



Microsoft Project Catapult

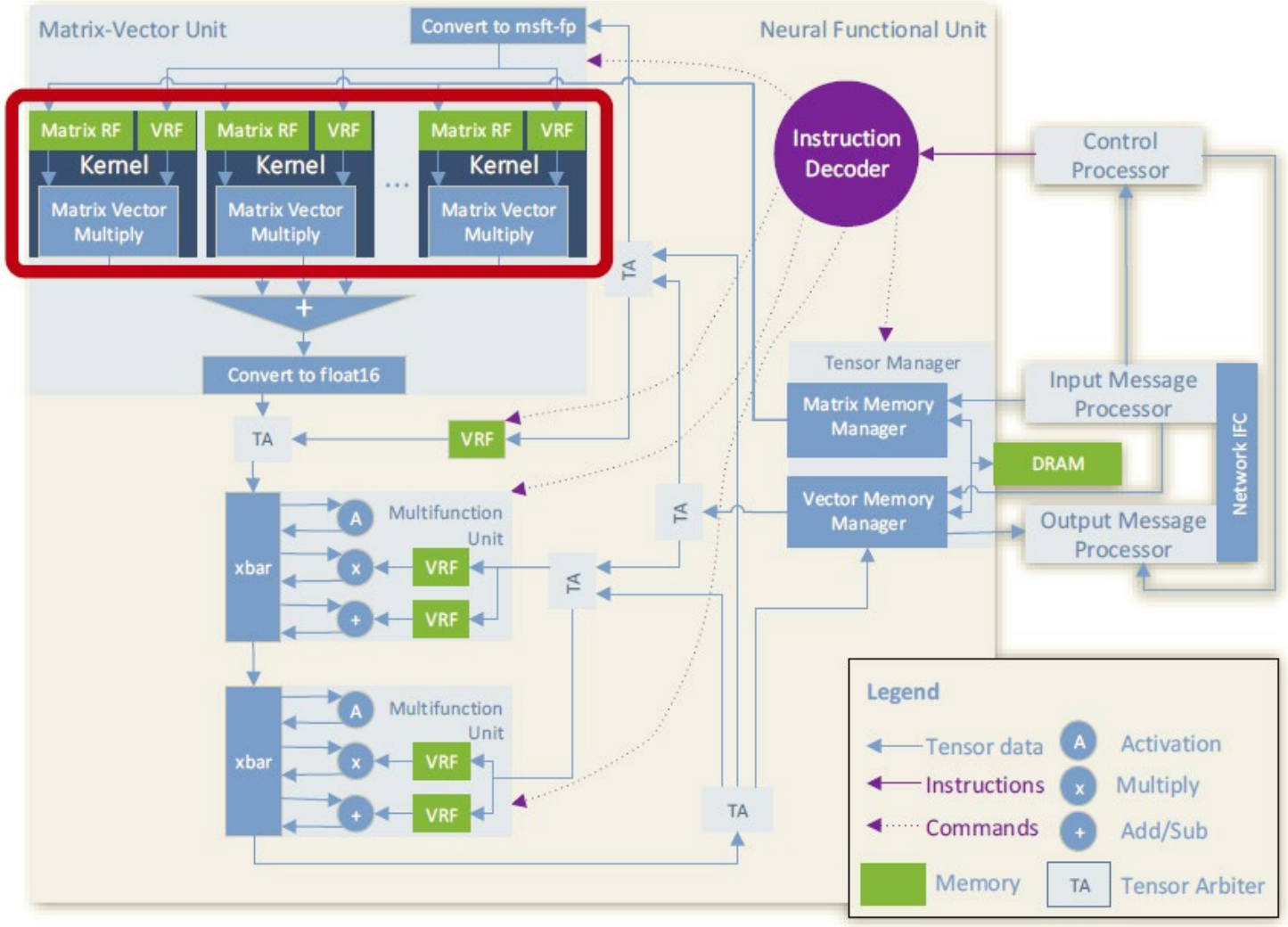
Configurable Cloud (MICRO 2016) for Azure



Accelerate and reduce latency for

- Bing search
- Software defined network
- Encryption and Decryption

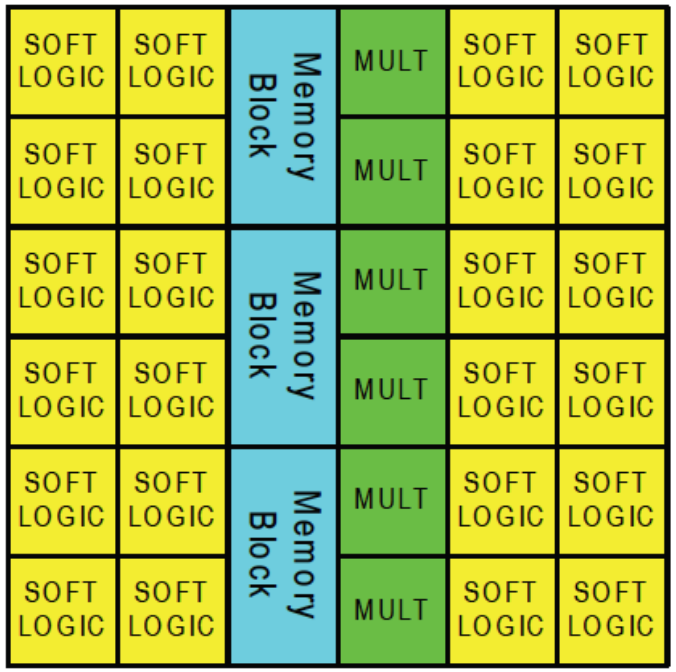
Microsoft Brainwave Neural Processor



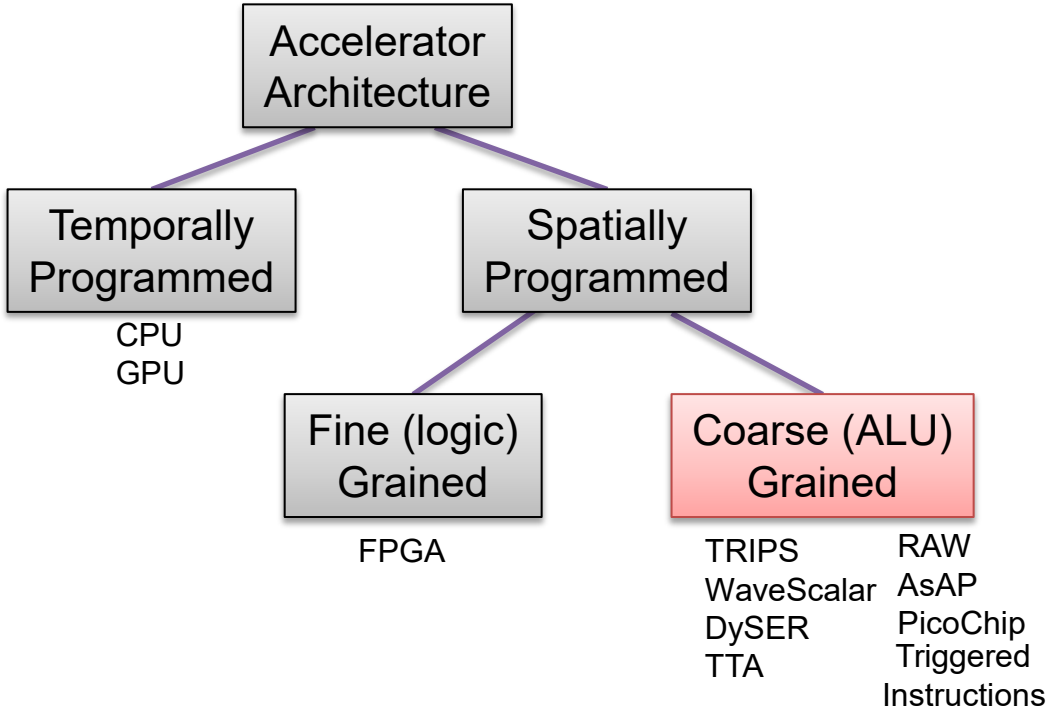
Source: Microsoft

Heterogeneous Blocks

- Add *specific purpose logic* on FPGA
  - Efficient if used (better area, speed, power), wasted if not
- Soft fabric
  - LUT, flops, addition, subtraction, carry logic
  - Convert LUT to memories or shift registers
- Memory block (BRAM)
  - Configure word and address size (aspect ratio)
  - Combine memory blocks to large blocks
  - Significant part for FPGA area
  - Dual port memories (FIFO)
- Multipliers /MACs → DSP
- CPUs and processing elements

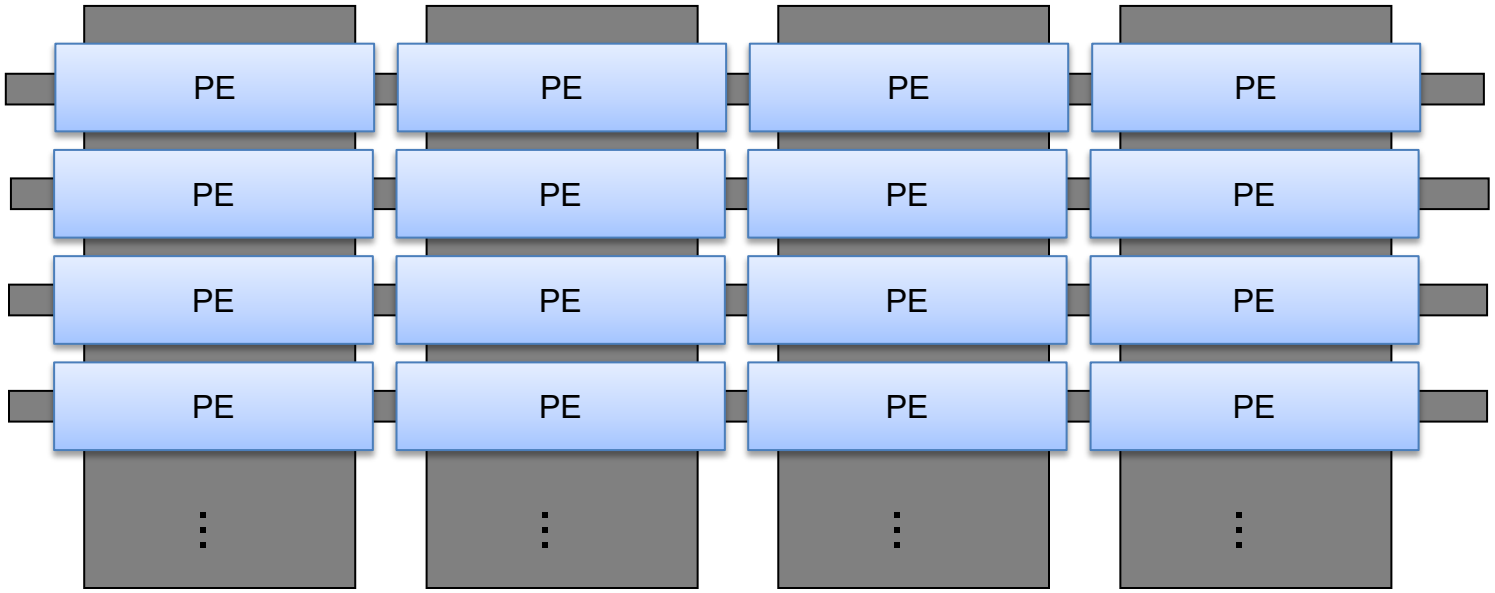


Accelerator Taxonomy



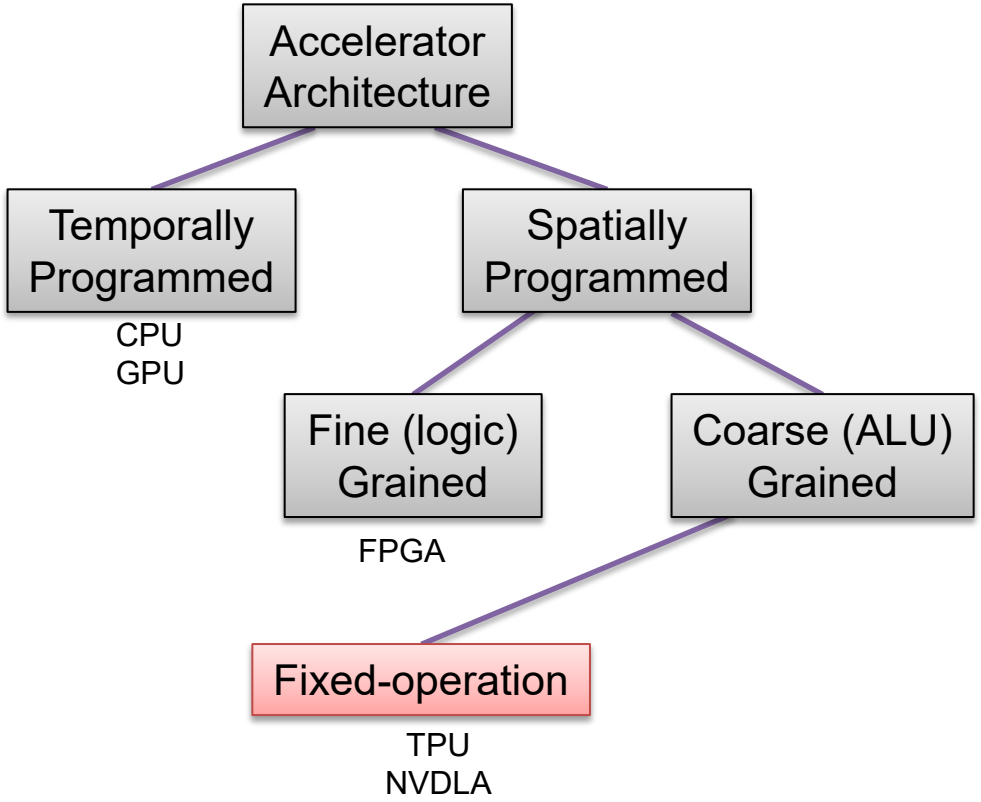
Programmable Accelerators

Processing Element



Many Programmable Accelerators look like an array of PEs, but have dramatically different architectures, programming models and capabilities

Accelerator Taxonomy

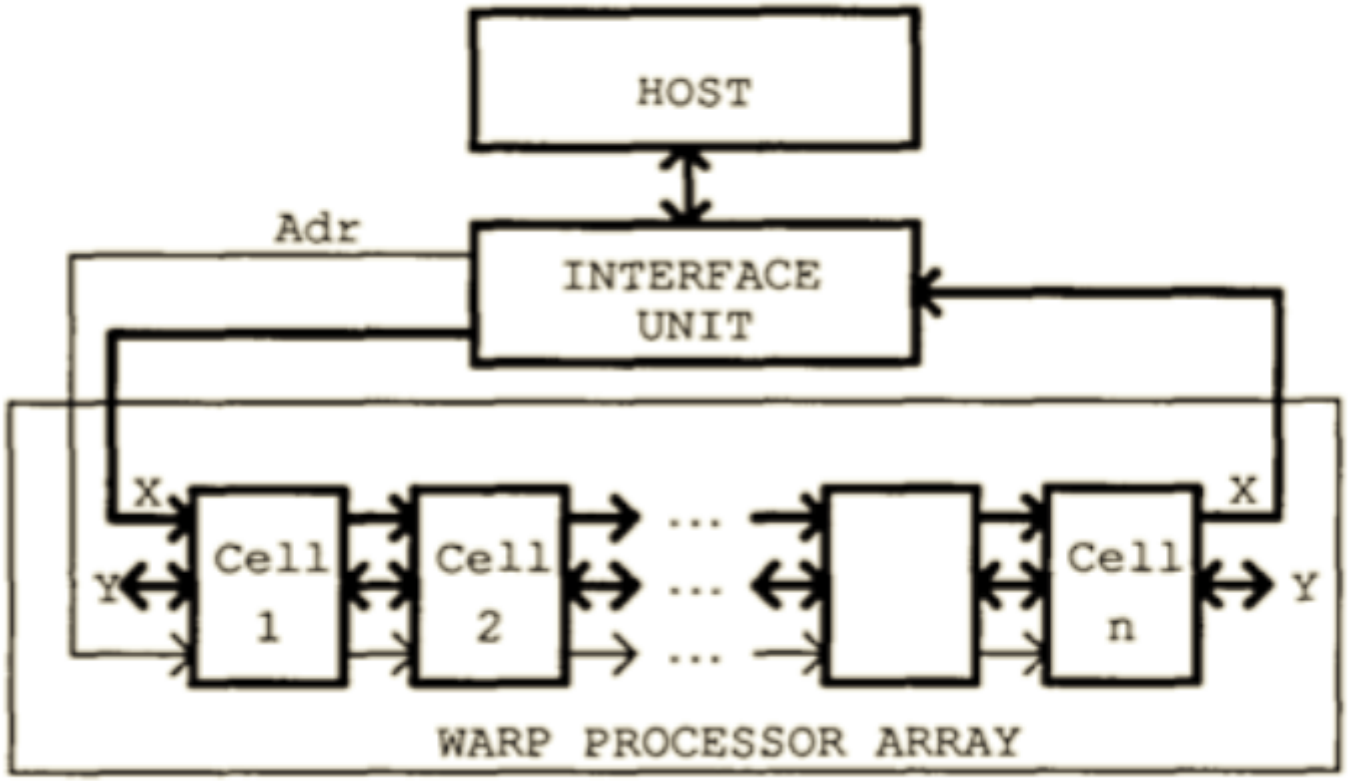


Fixed Operation PEs

- Each PE hard-wired to one operation
- Purely pipelined operation
  - no backpressure in pipeline

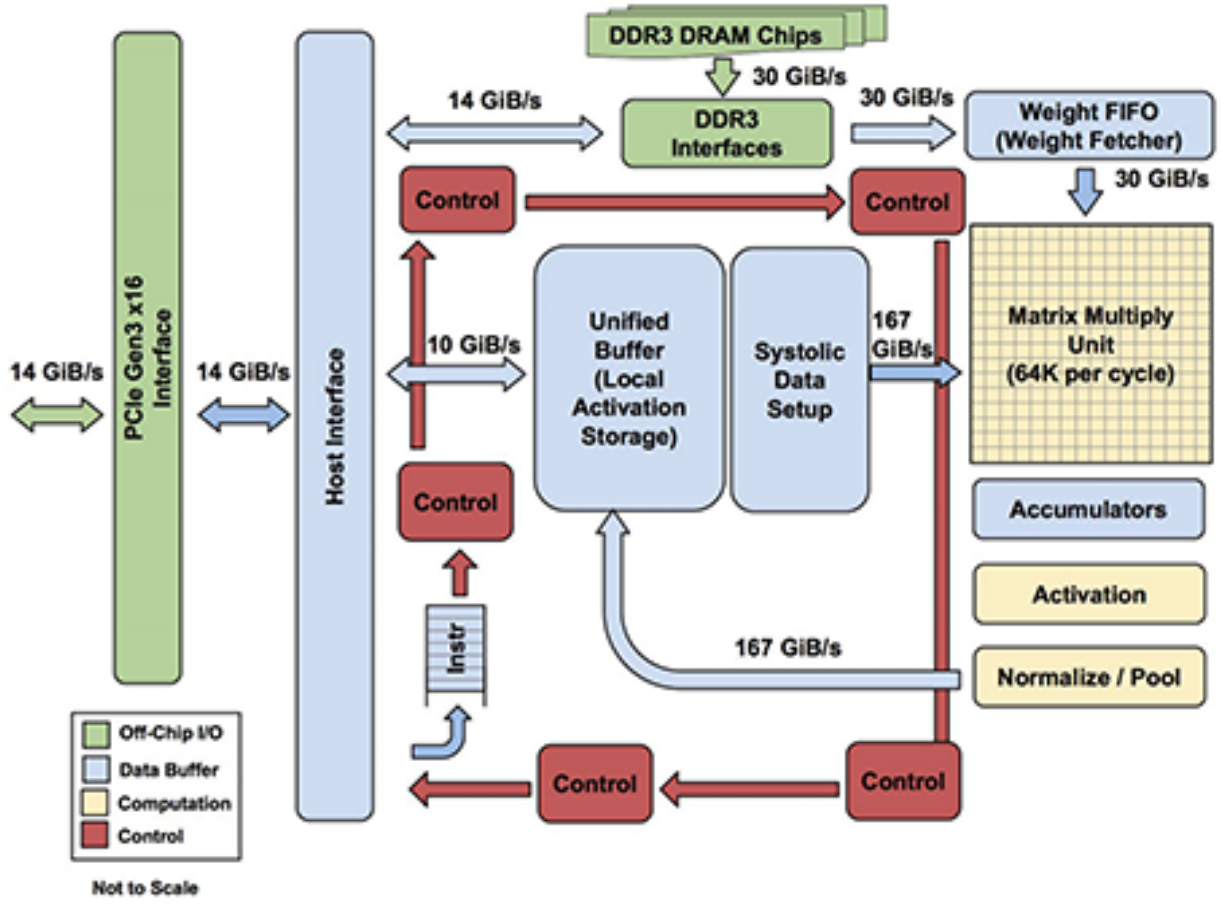
- Attributes
  - High-concurrency
  - Regular design, but
  - Regular parallelism only!
  - Allows for systolic communication

Configurable Systolic Array - WARP



Source: WARP Architecture and Implementation, ISCA 1986

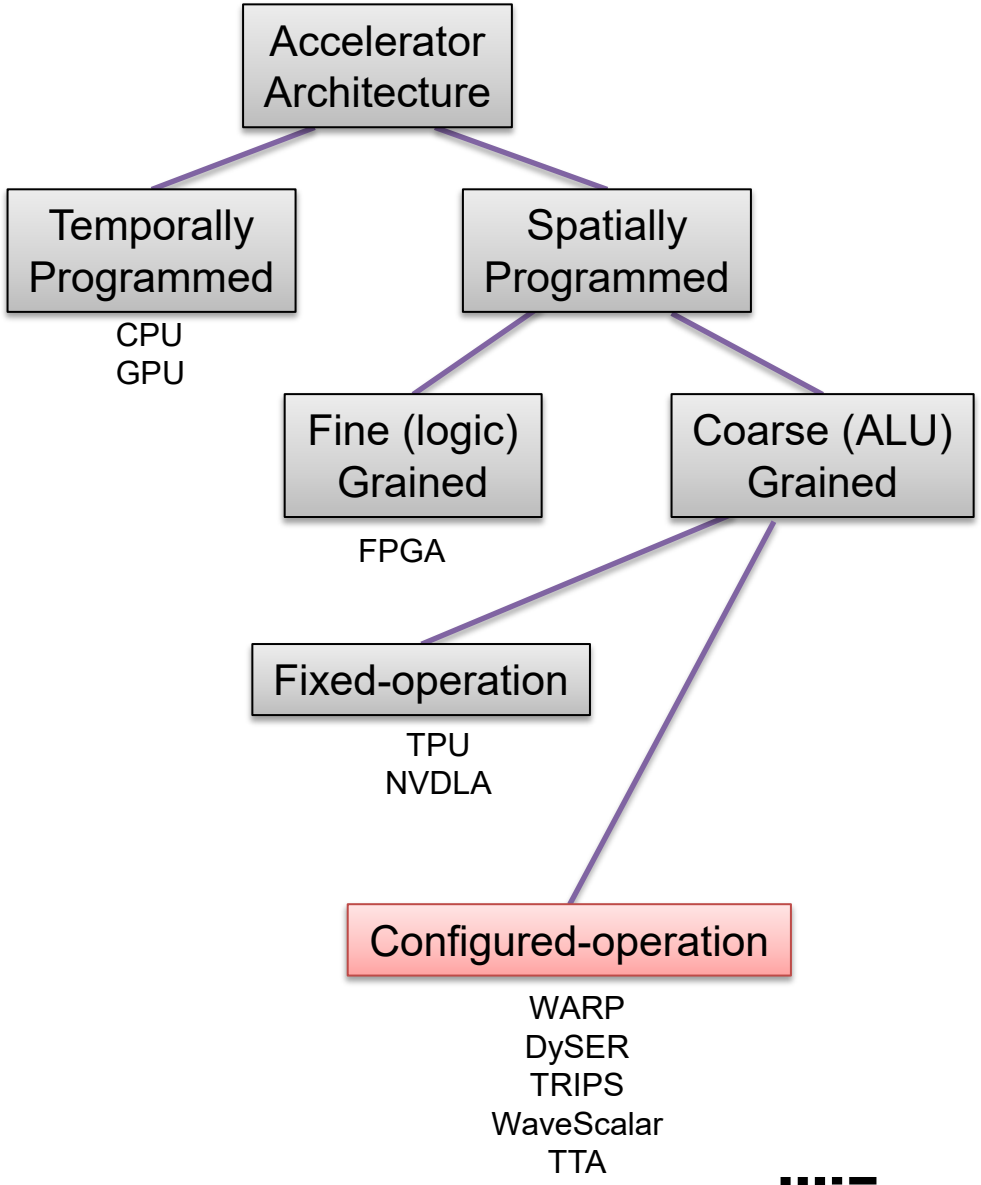
Fixed Operation - Google TPU



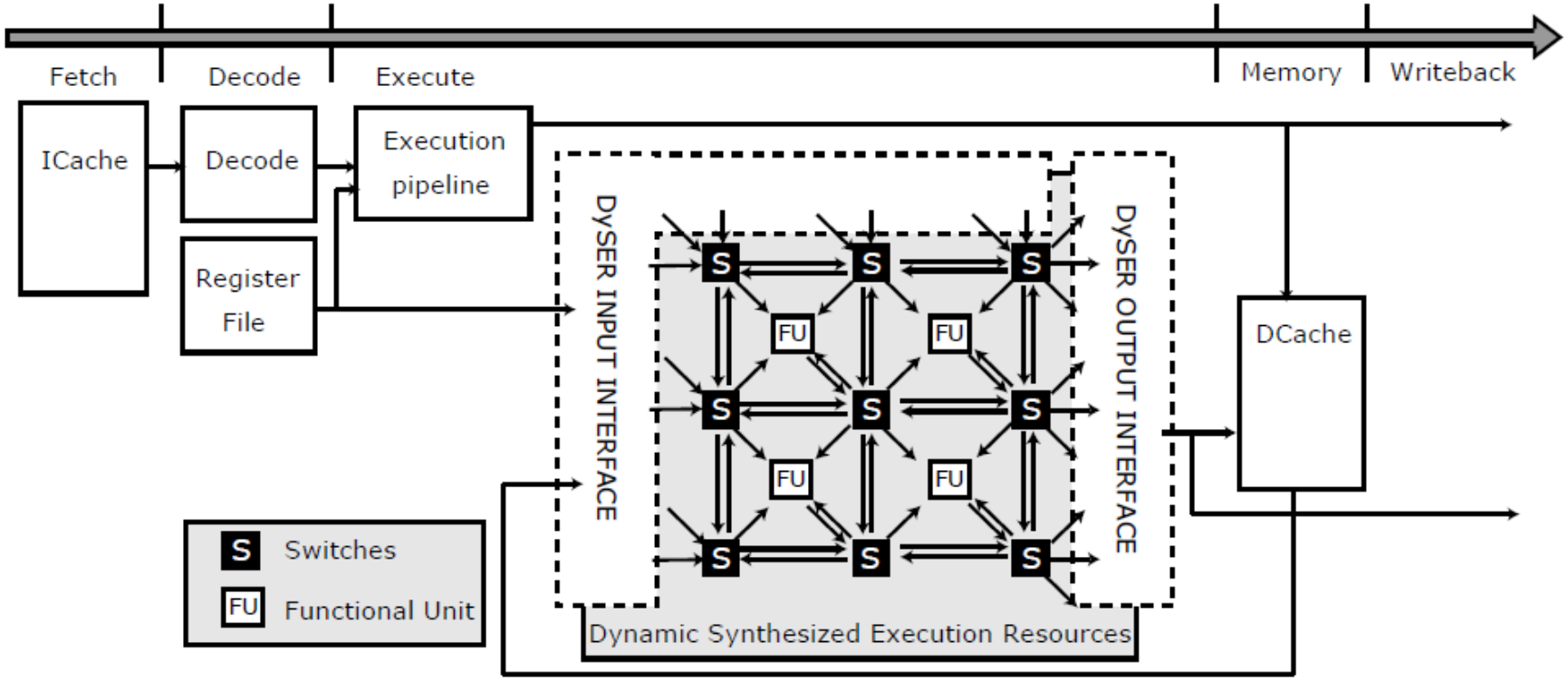
Systolic array does 8-bit 256x256 matrix-multiply accumulate

Source: Google

Accelerator Taxonomy

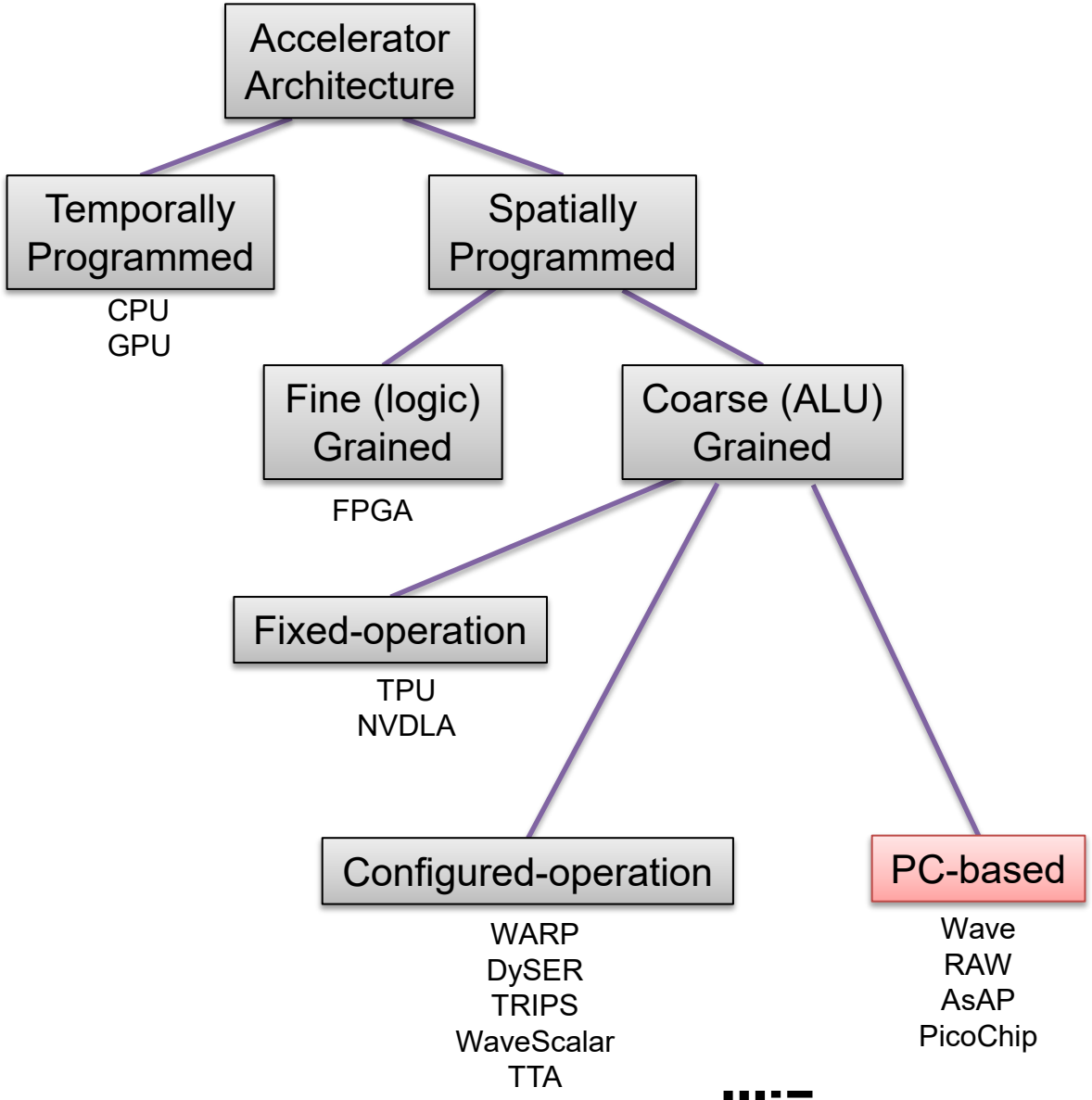


Single Configured Operation - Dyser

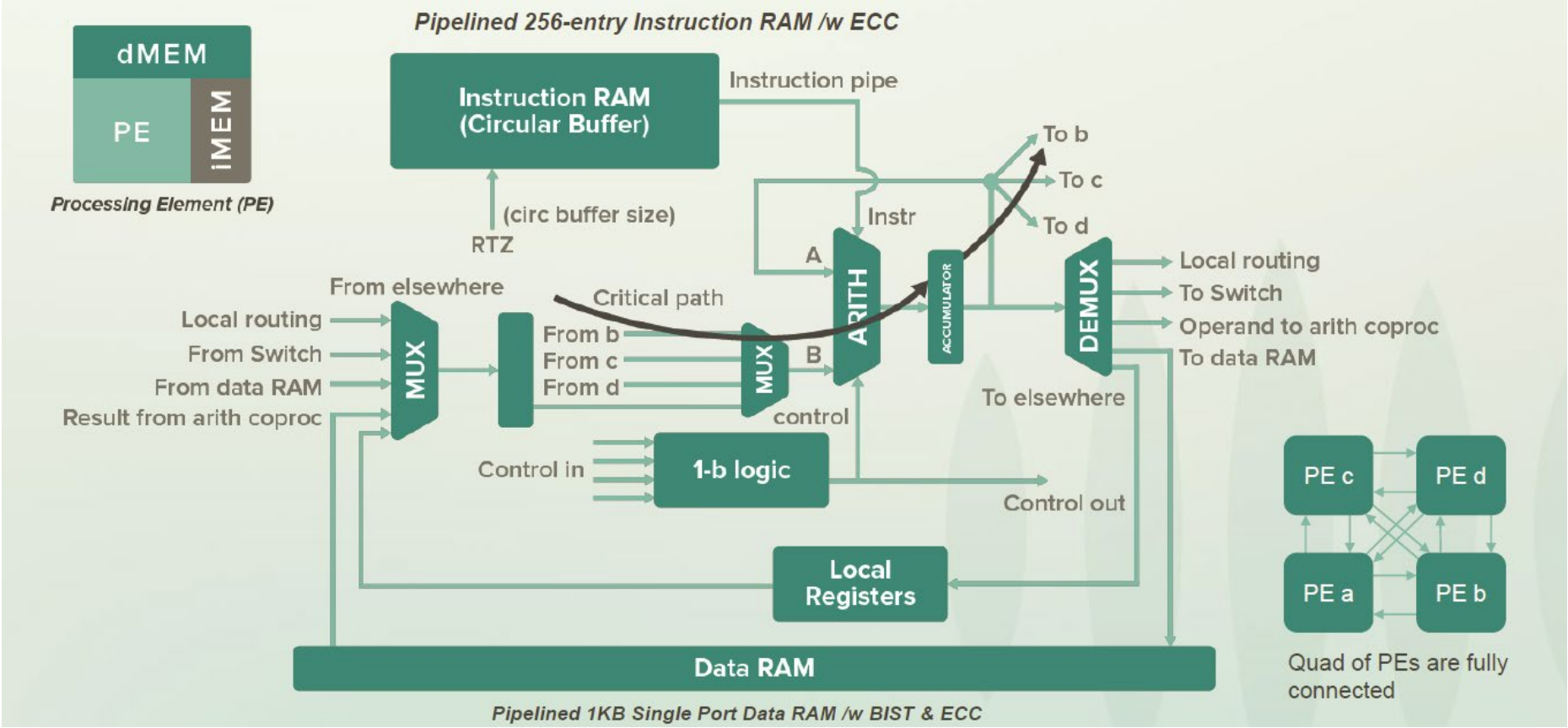


Source: Dynamically Specialized Datapaths for Energy Efficient Computing. HPCA11

Accelerator Taxonomy

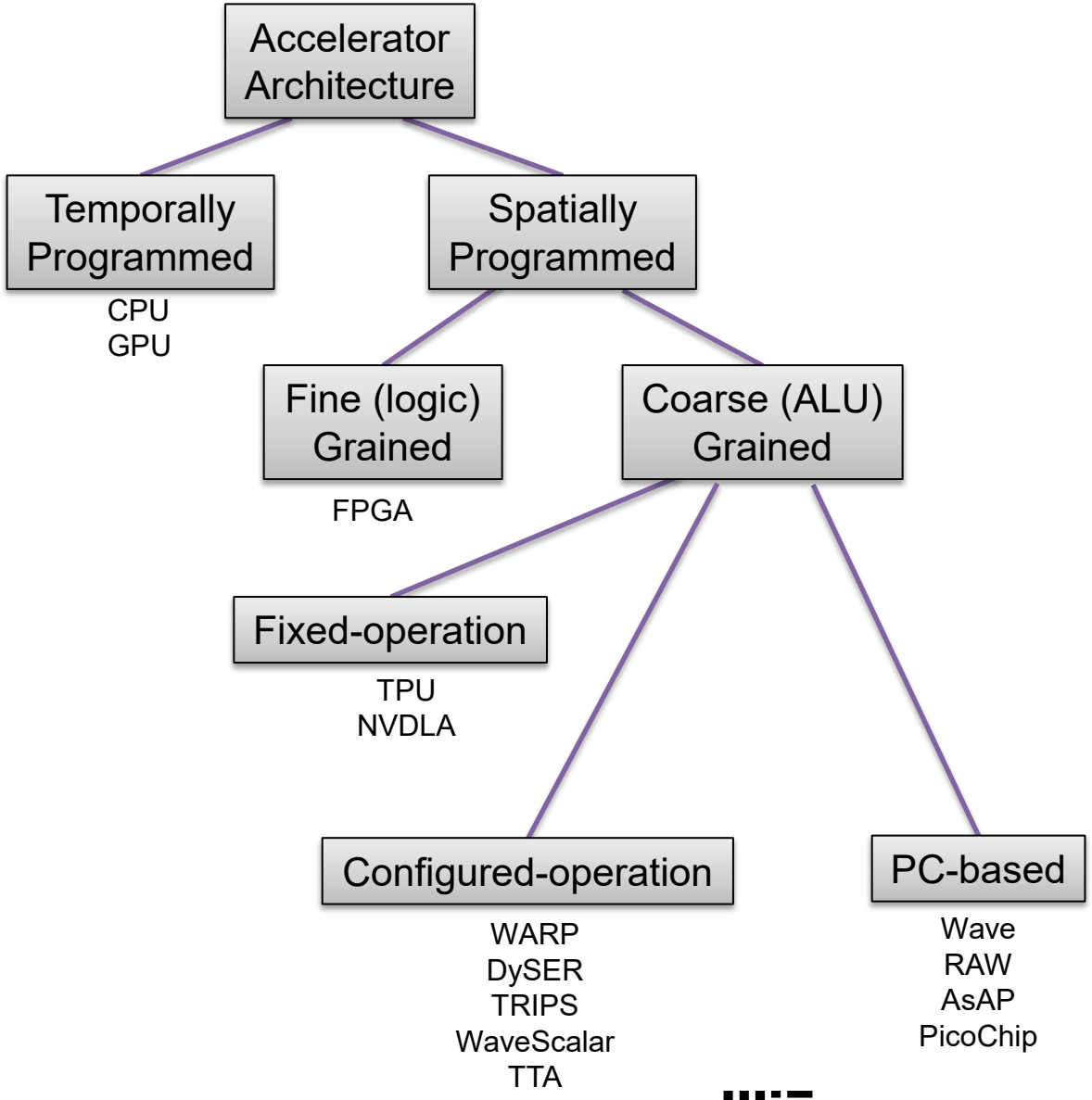


PC-based Control – Wave Computing

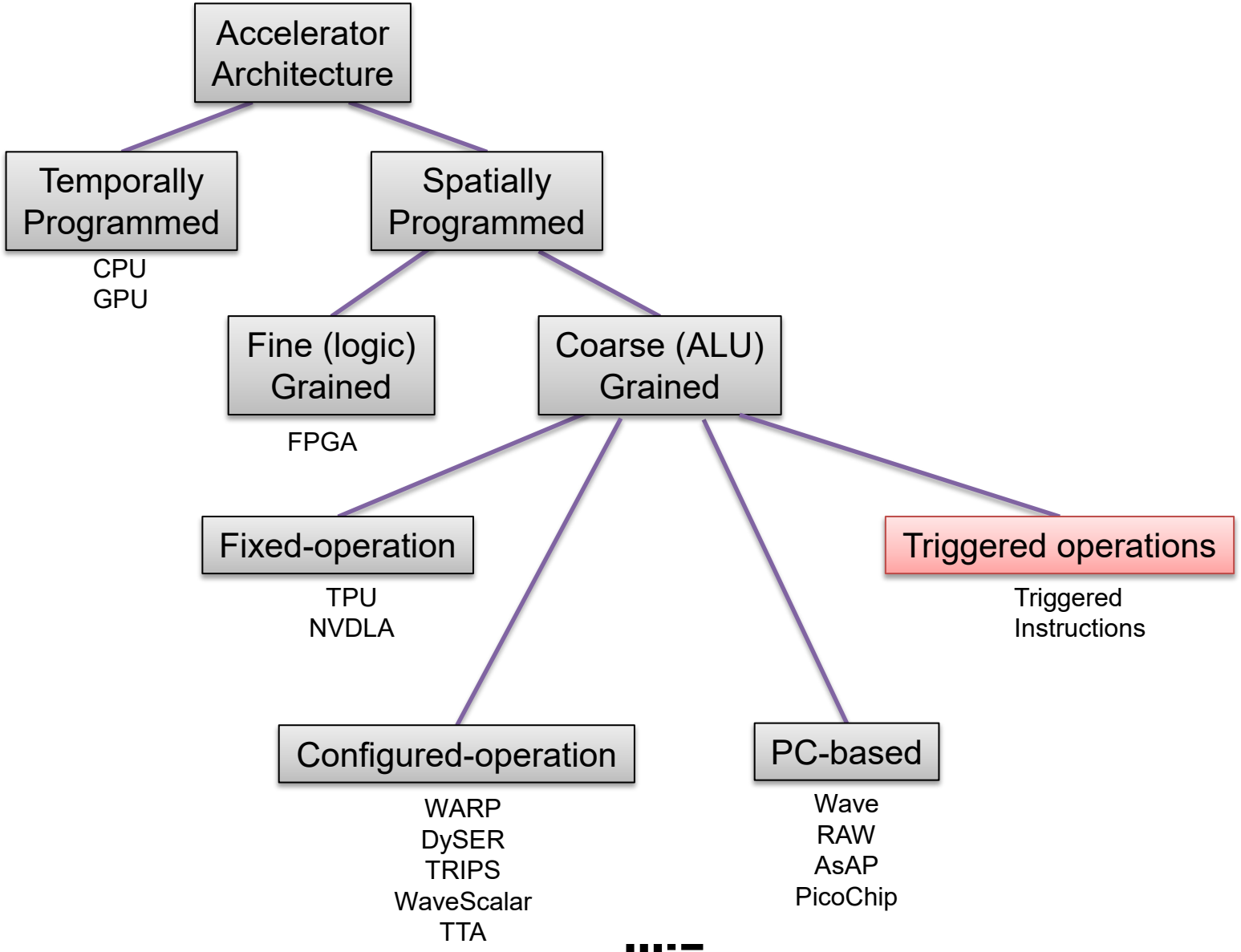


Source: Wave Computing, Hot Chips '17

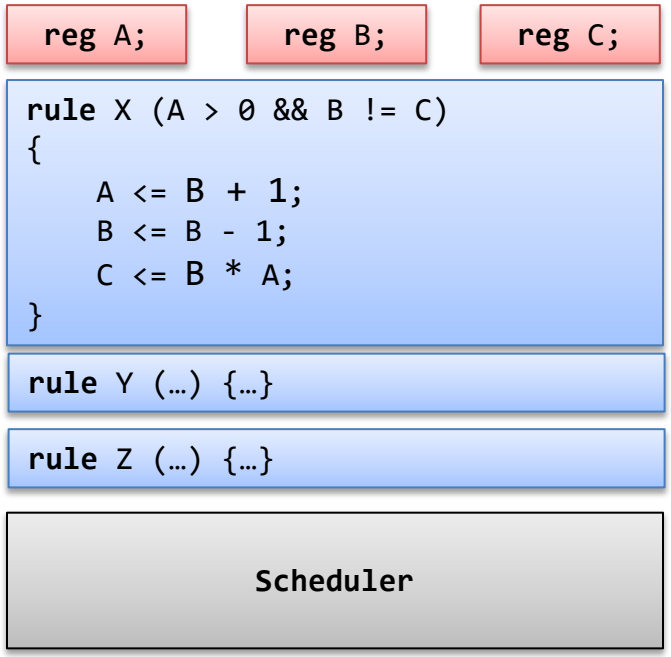
Accelerator Taxonomy



Accelerator Taxonomy



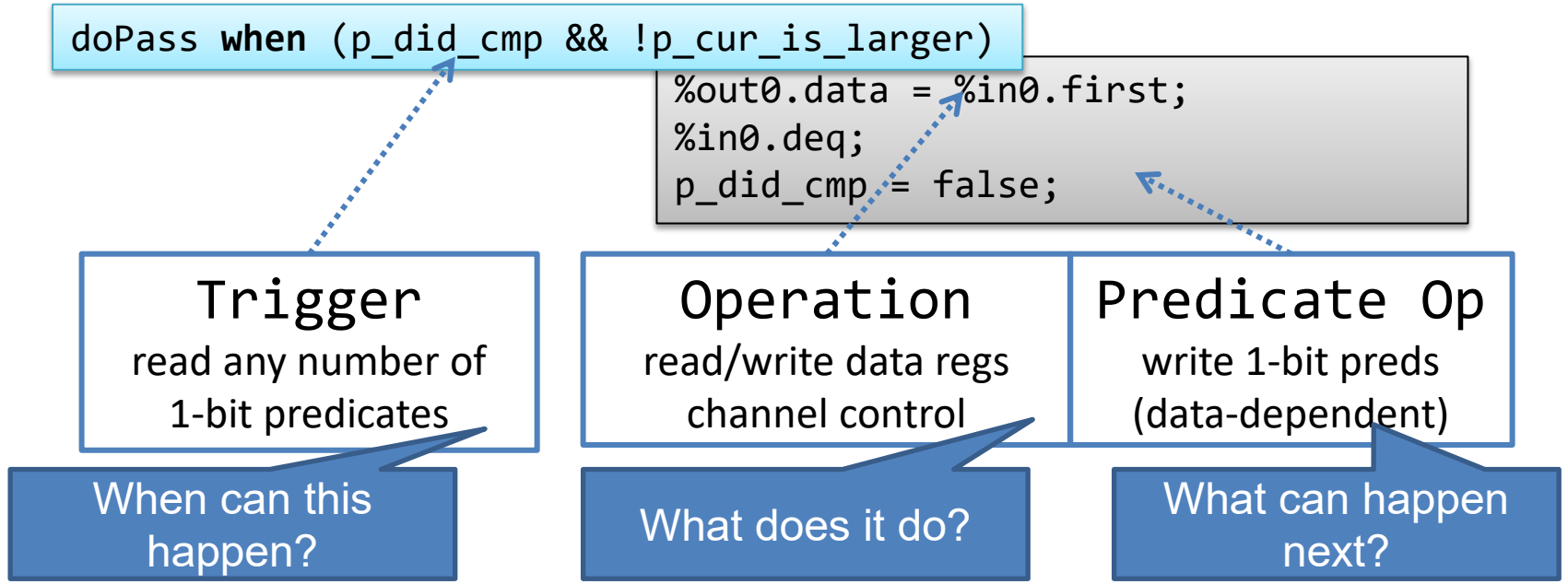
Guarded Actions



- Program consists of **rules** that may perform computations and read/write state
- Each rule specifies conditions (**guard**) under which it is allowed to fire
- Separates description and execution of data (rule body) from control (guards)
- A **scheduler** is generated (or provided by hardware) that evaluates the guards and schedules rule execution
- Sources of Parallelism
  - Intra-Rule parallelism
  - Inter-Rule parallelism
  - Scheduler overlap with Rule execution
  - Parallel access to state

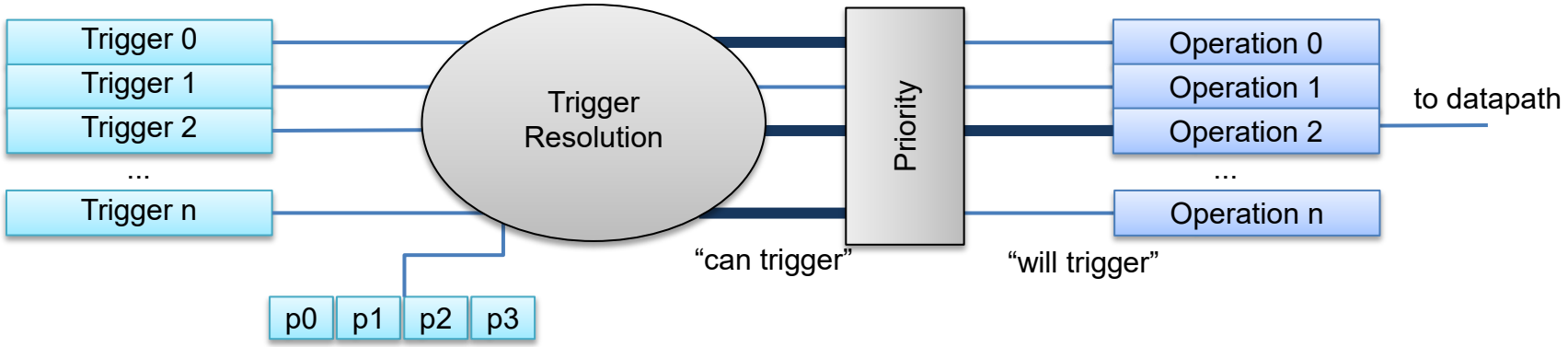
Triggered Instructions (TI)

- Restrict guarded actions down to efficient ISA core:



No program counter or branch instructions

Triggered Instruction Scheduler



- Use combinational logic to evaluate triggers in parallel
- Decide winners if more than one instruction is ready
  - Based on architectural fairness policy
  - Could pick multiple non-conflicting instructions to issue (superscalar)
- Note: no wires toggle unless status changes

6.5930/1

Hardware Architectures for Deep Learning

Dataflow for DNN Accelerator Architectures (Part 1)

March 11, 2024

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Lecture

- Impact of data movement and memory hierarchy on energy consumption
- Taxonomy of dataflows for CNNs
  - Output Stationary
  - Weight Stationary
  - Input Stationary

Background Reading

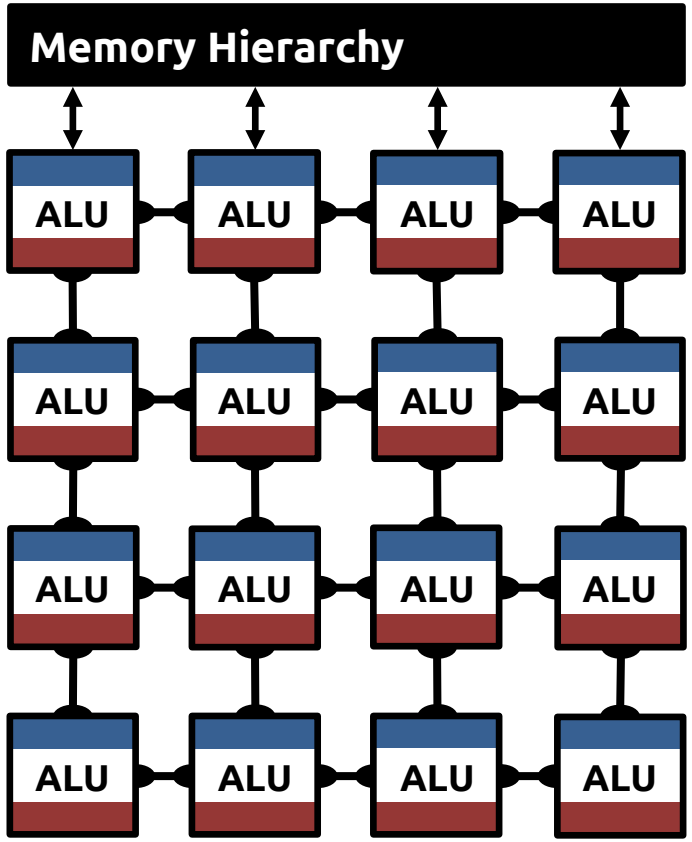
- **DNN Accelerators**
  - ***Efficient Processing of Deep Neural Networks***
    - Chapter 5 – thru 5.7.1
    - Chapter 5 – 5.8

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

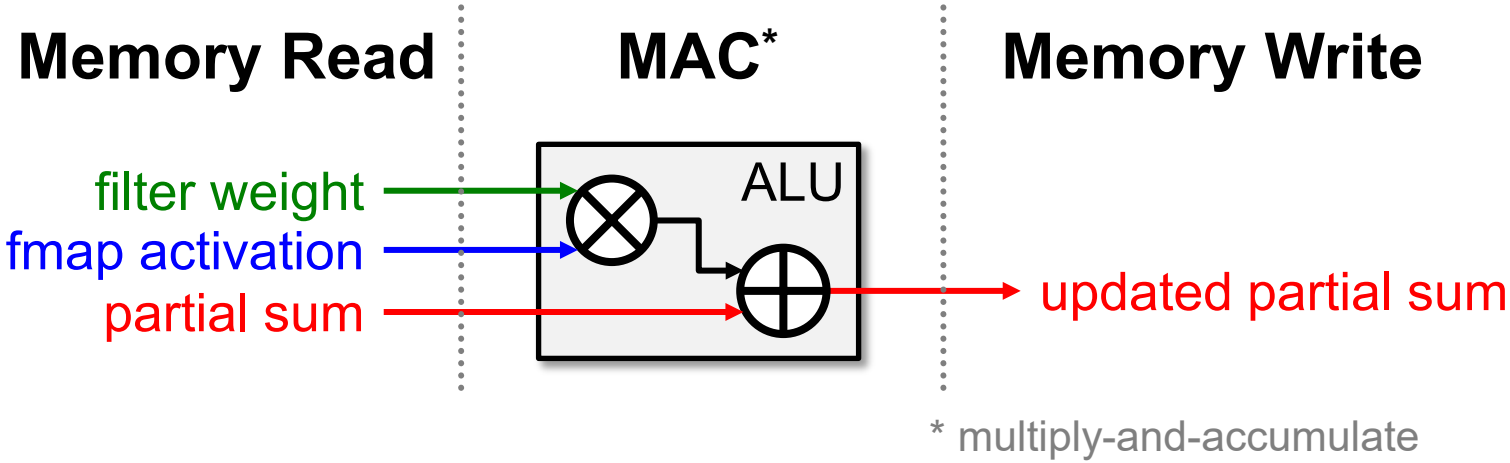
Dataflow and Memory Hierarchy

Spatial Compute Paradigm

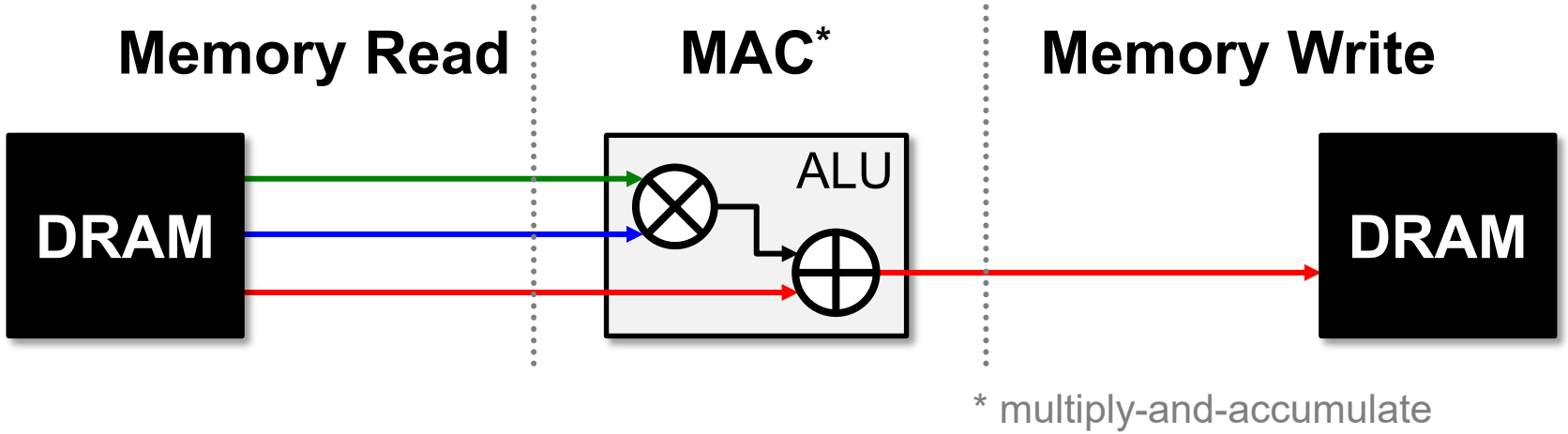
Spatial Architecture
(Dataflow Processing)



Memory Access is the Bottleneck



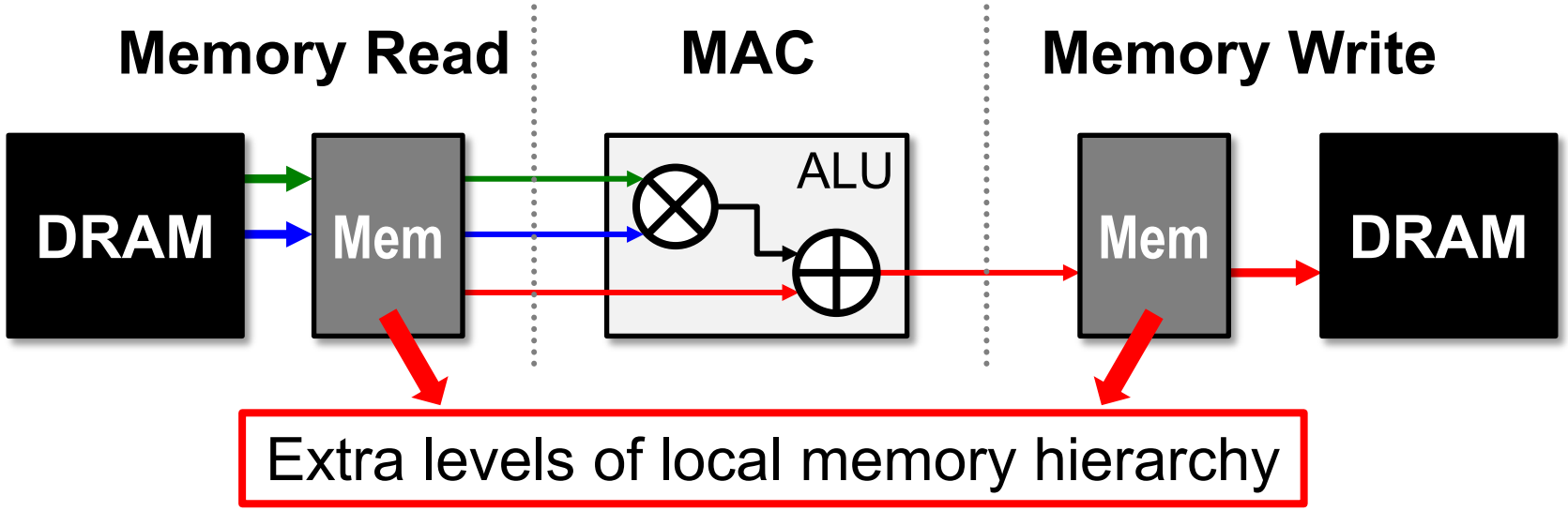
Memory Access is the Bottleneck



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

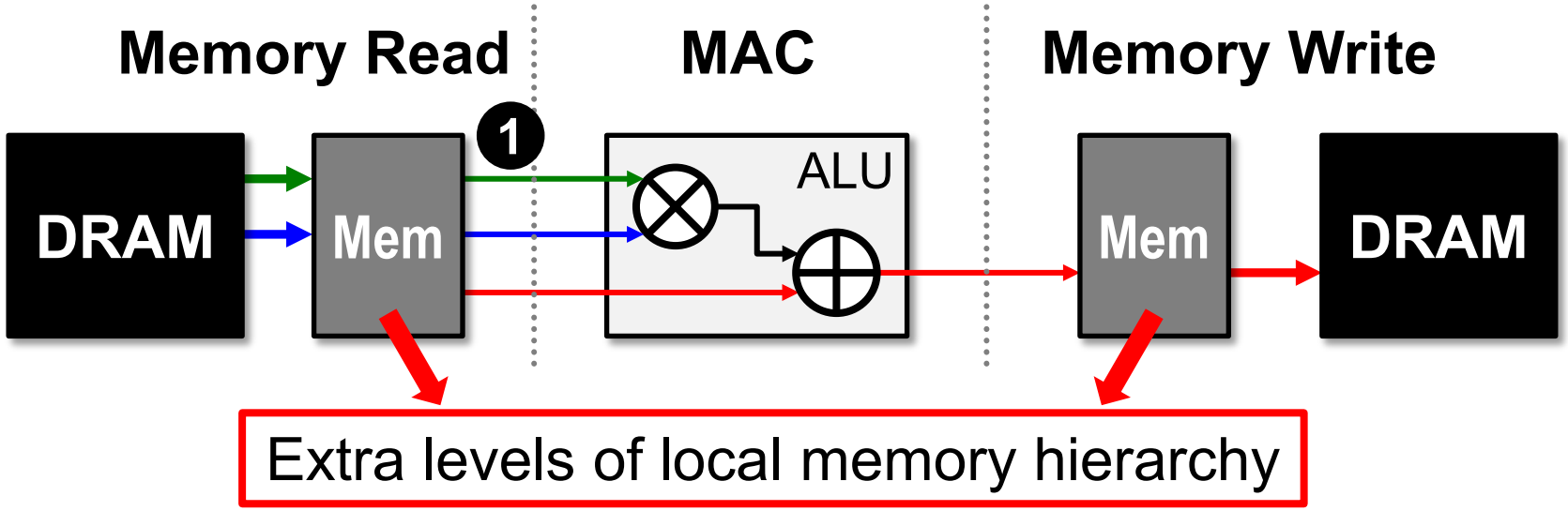
- Example: AlexNet [NeurIPS 2012] has **724M** MACs
  → **2896M** DRAM accesses required

Memory Access is the Bottleneck



Under what circumstances will these extra levels help?

Memory Access is the Bottleneck

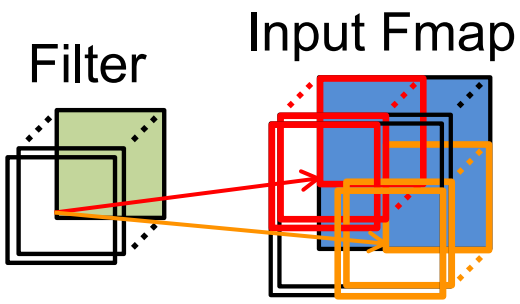


Opportunities: 1 **data reuse**

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

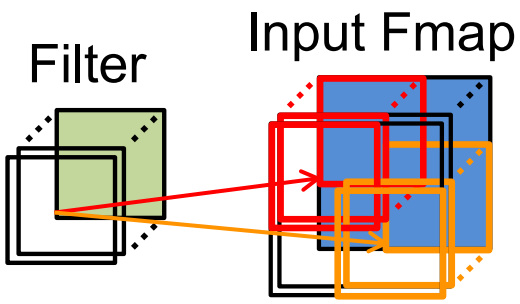


Reuse: Activations
Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

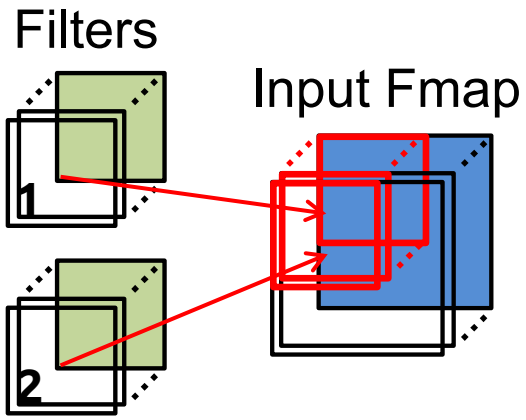
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

CONV and FC layers

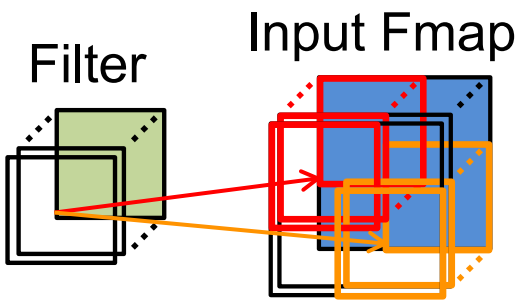


Reuse: Activations

Types of Data Reuse in DNN

Convolutional Reuse

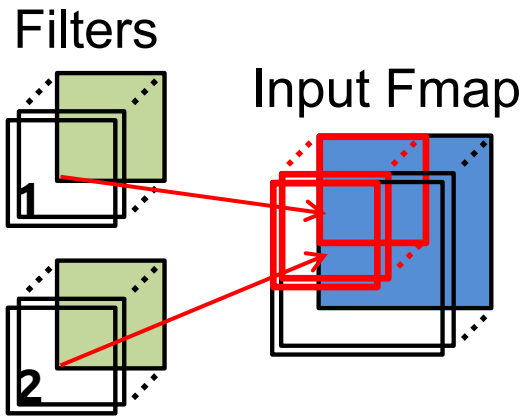
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

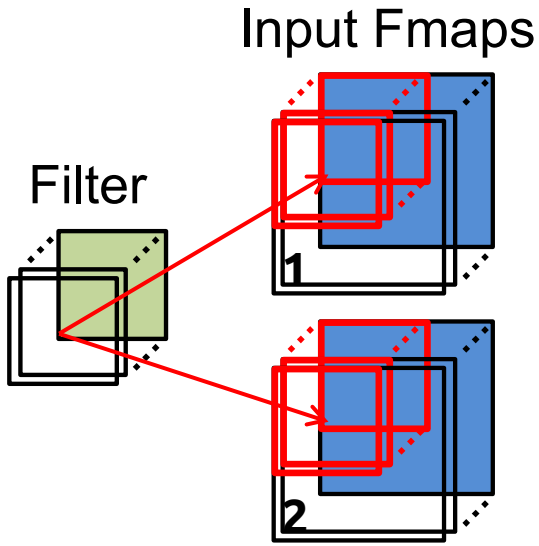
CONV and FC layers



Reuse: Activations

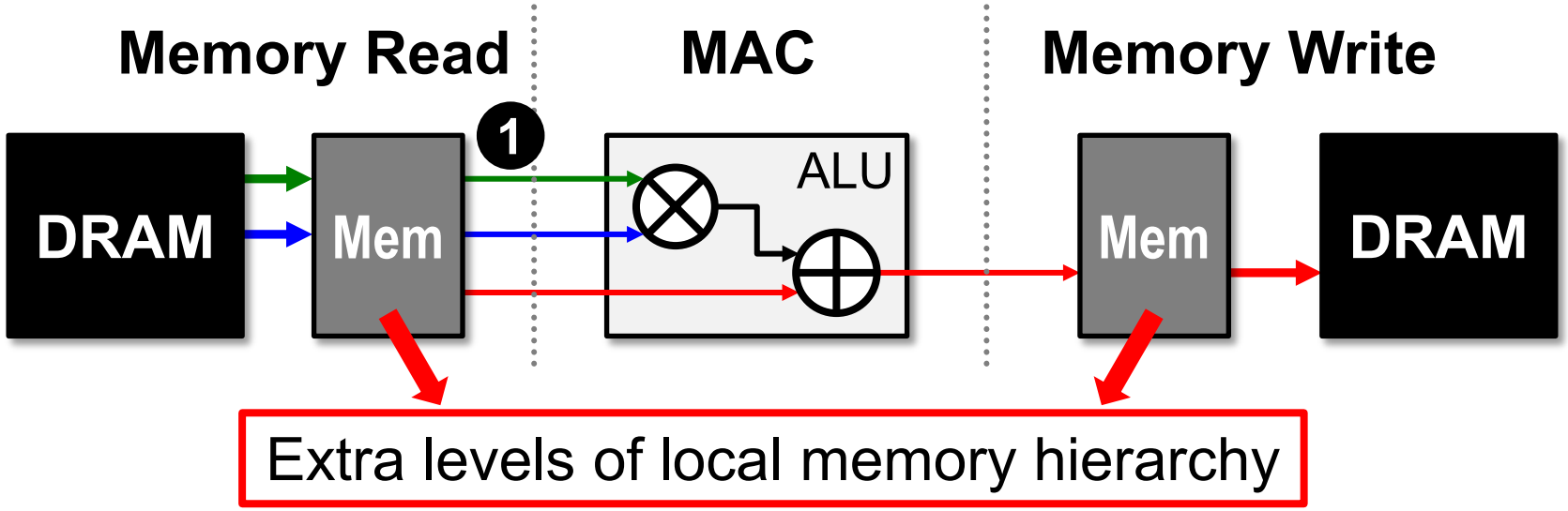
Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Memory Access is the Bottleneck

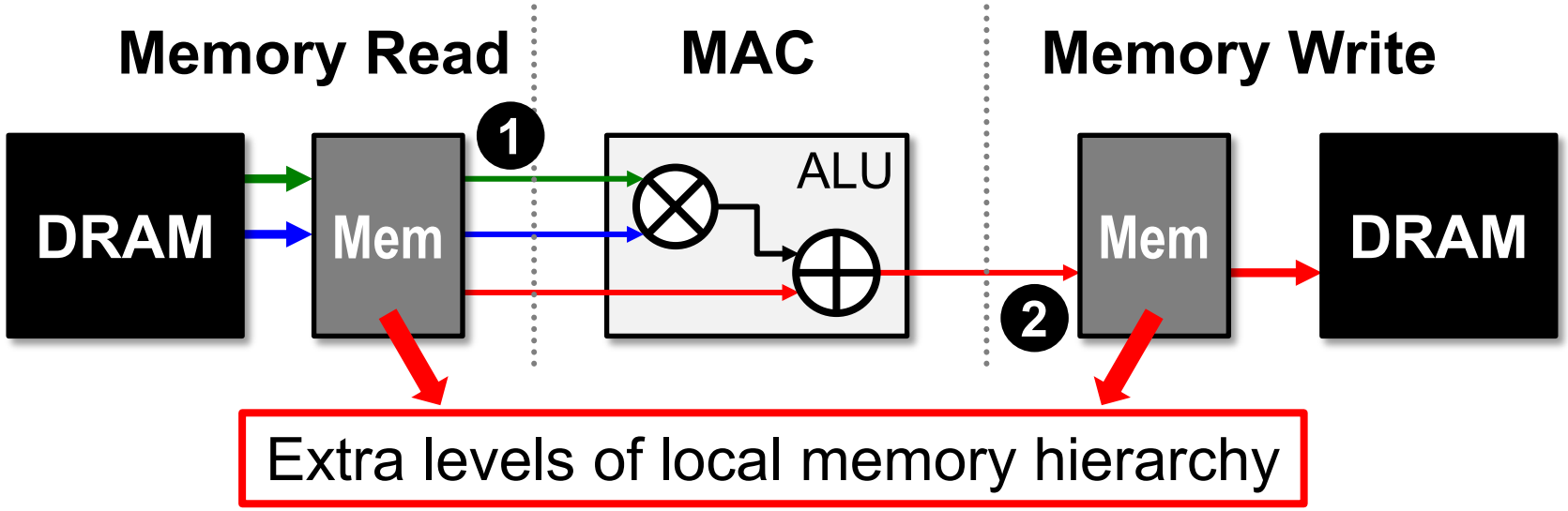


Opportunities: 1 **data reuse**

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

** AlexNet CONV layers

Memory Access is the Bottleneck

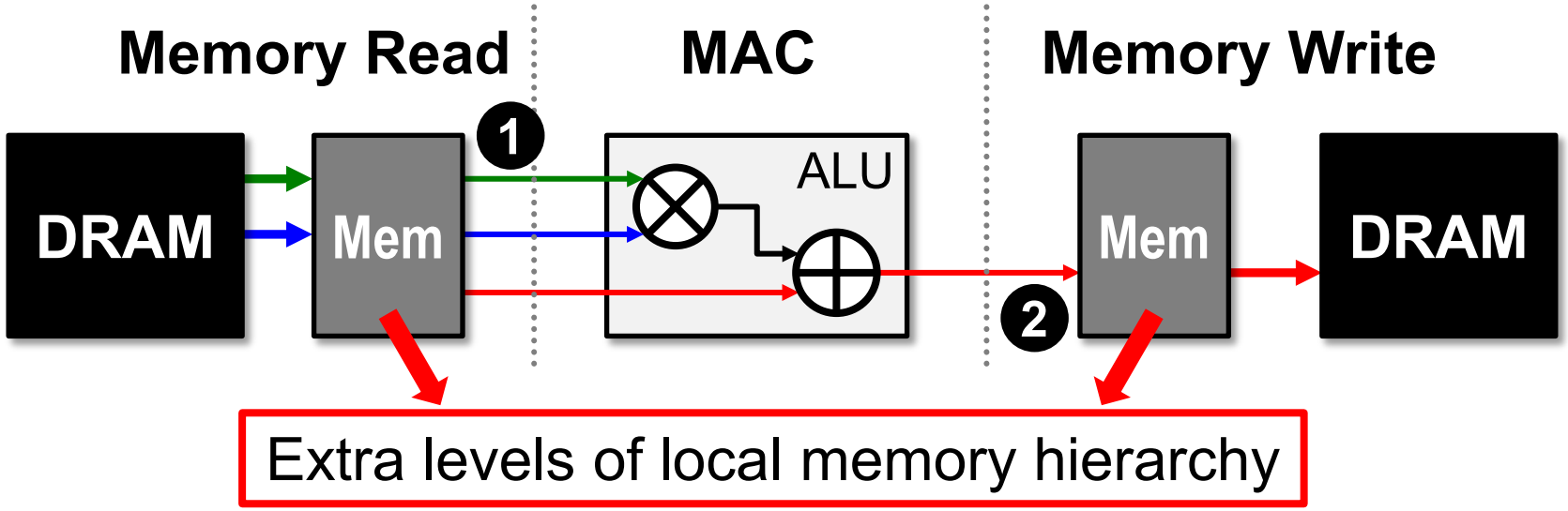


Opportunities: ❶ **data reuse** ❷ **local accumulation**

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

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

Memory Access is the Bottleneck



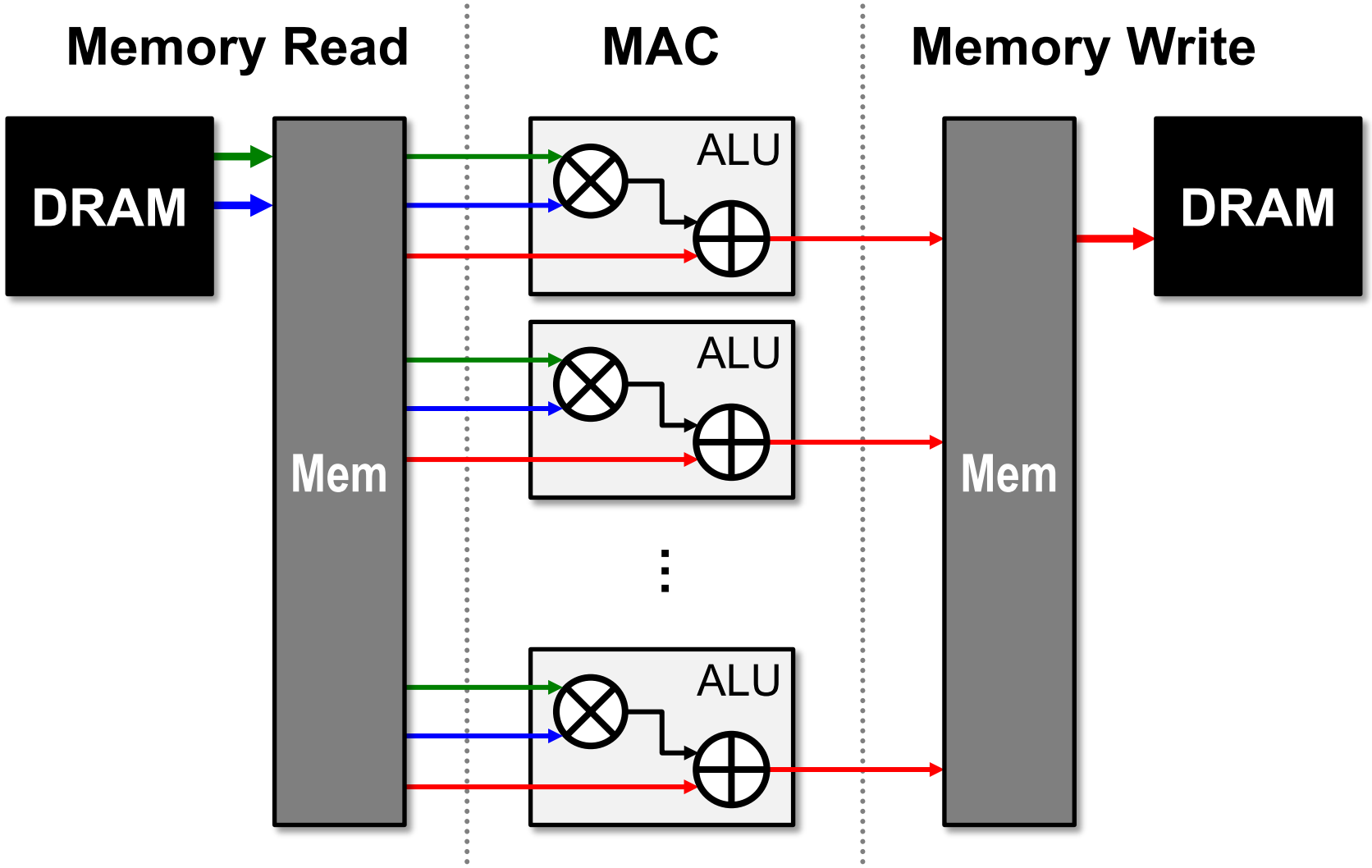
Opportunities: ❶ **data reuse** ❷ **local accumulation**

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

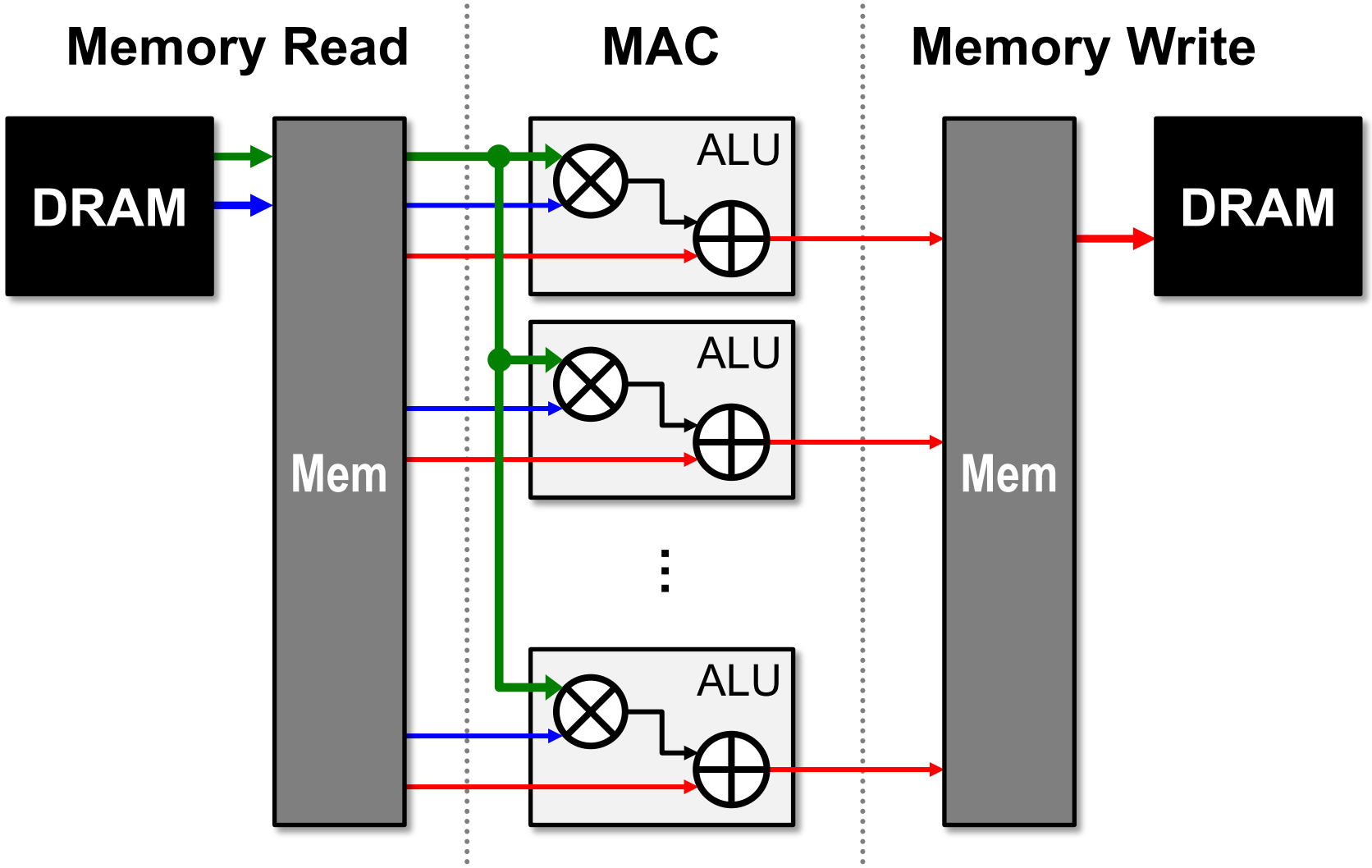
❷ Partial sum accumulation does **NOT** have to access DRAM

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

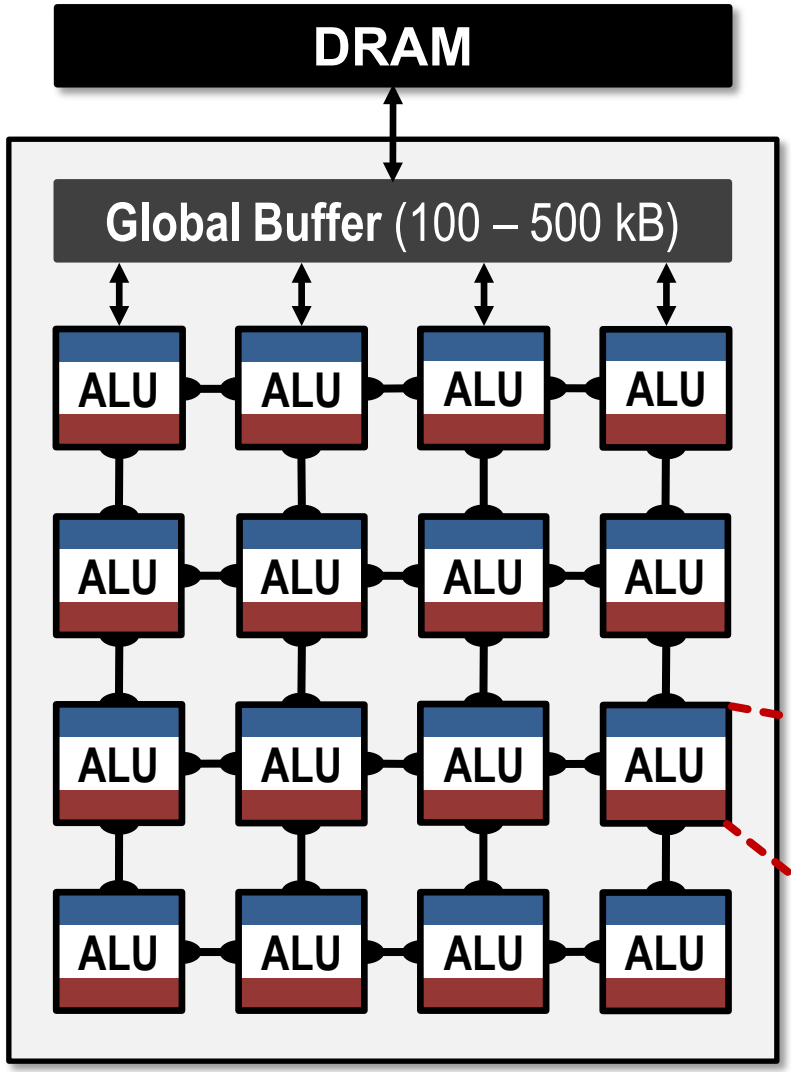
Leverage Parallelism for Higher Performance



Leverage Parallelism for *Spatial* Data Reuse



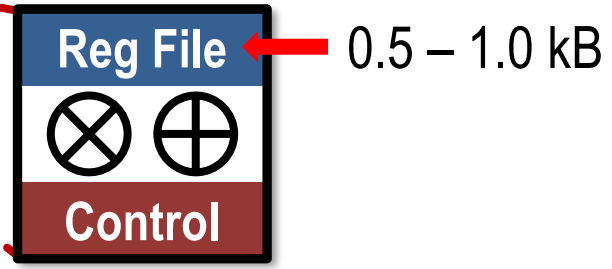
Spatial Architecture for DNN



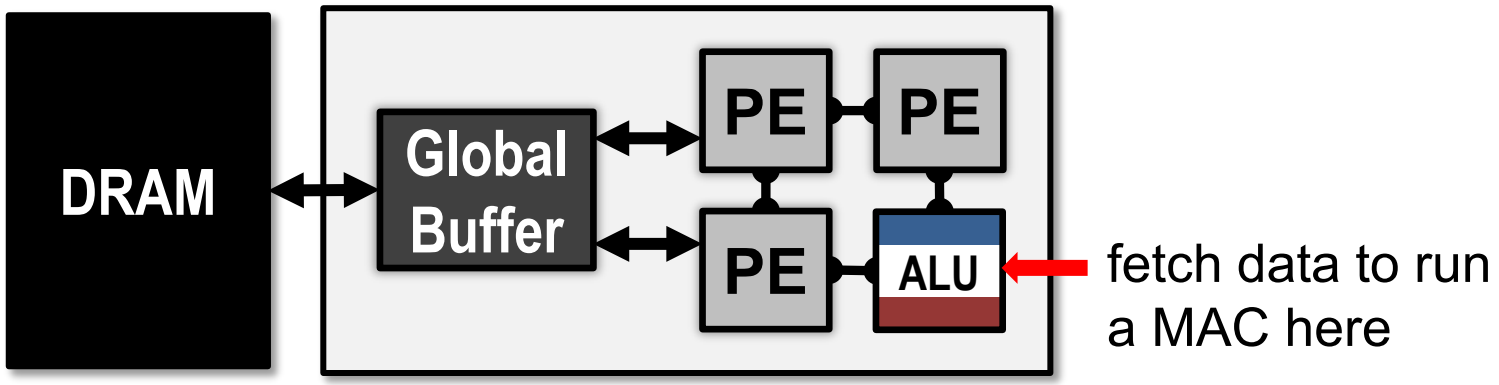
Local Memory Hierarchy

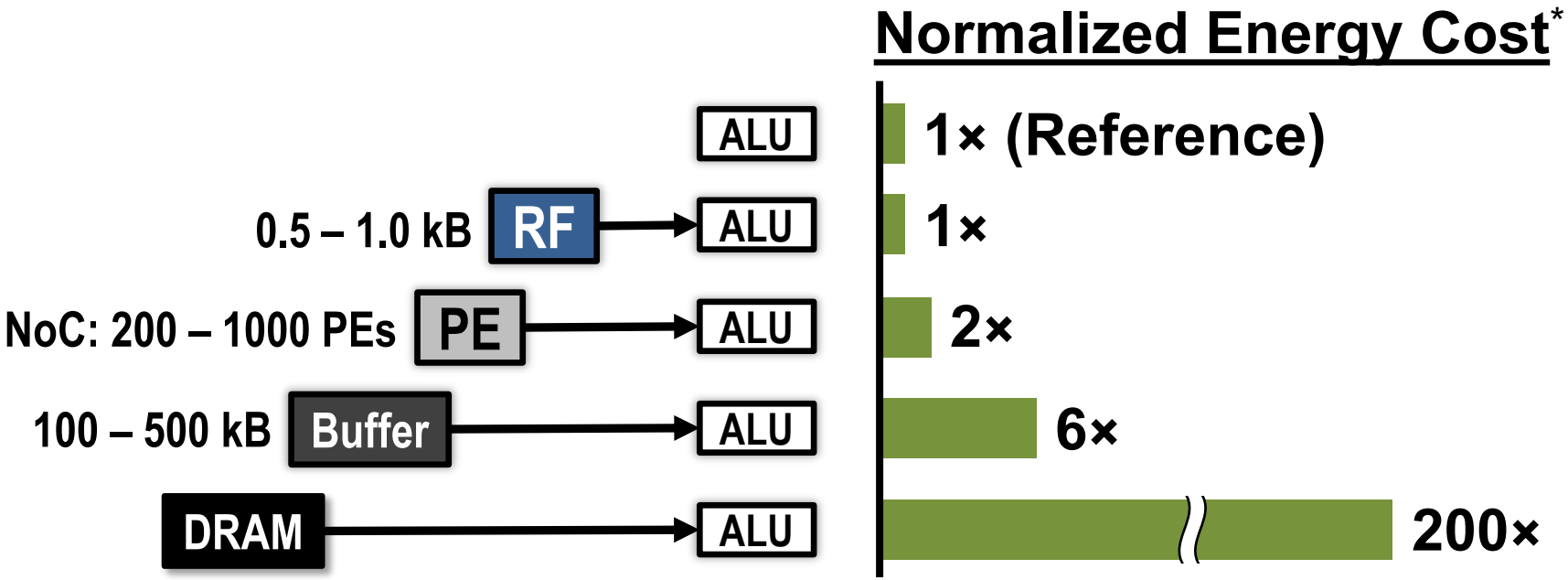
- Global Buffer
- Direct inter-PE network
- PE-local memory (RF)

Processing Element (PE)



Low-Cost Local Data Access

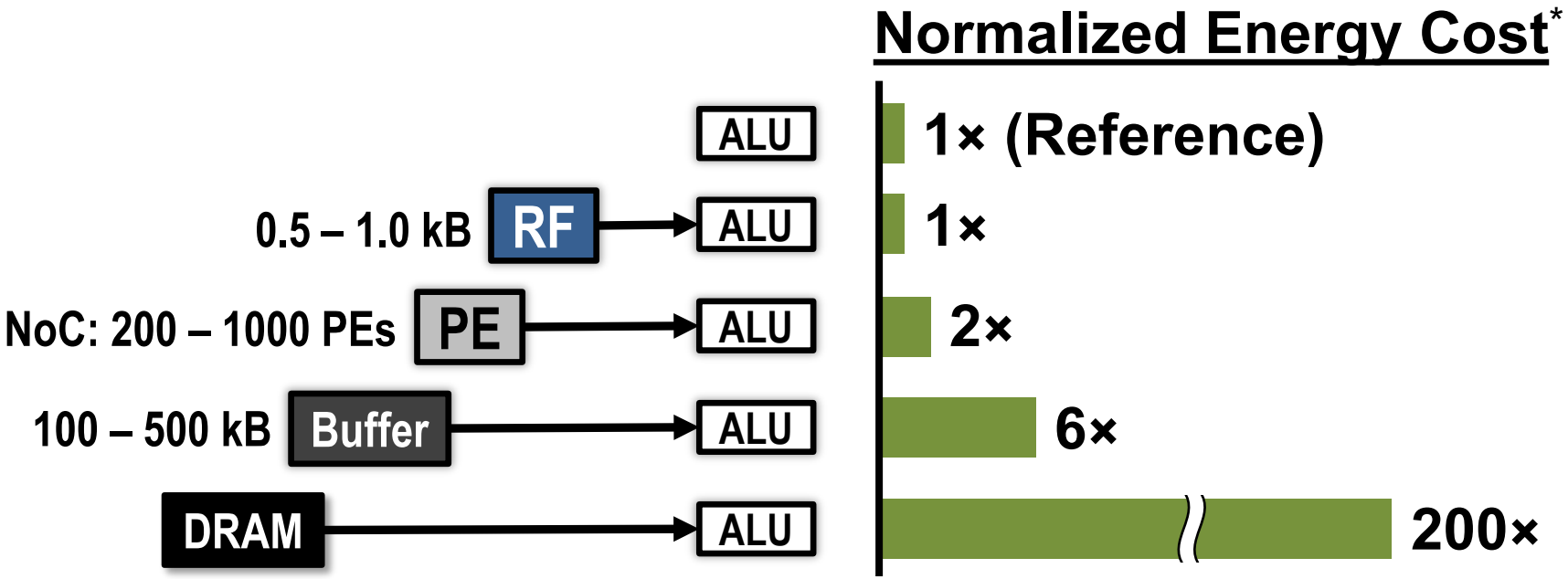




* measured from a commercial 65nm process

Low-Cost Local Data Access

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

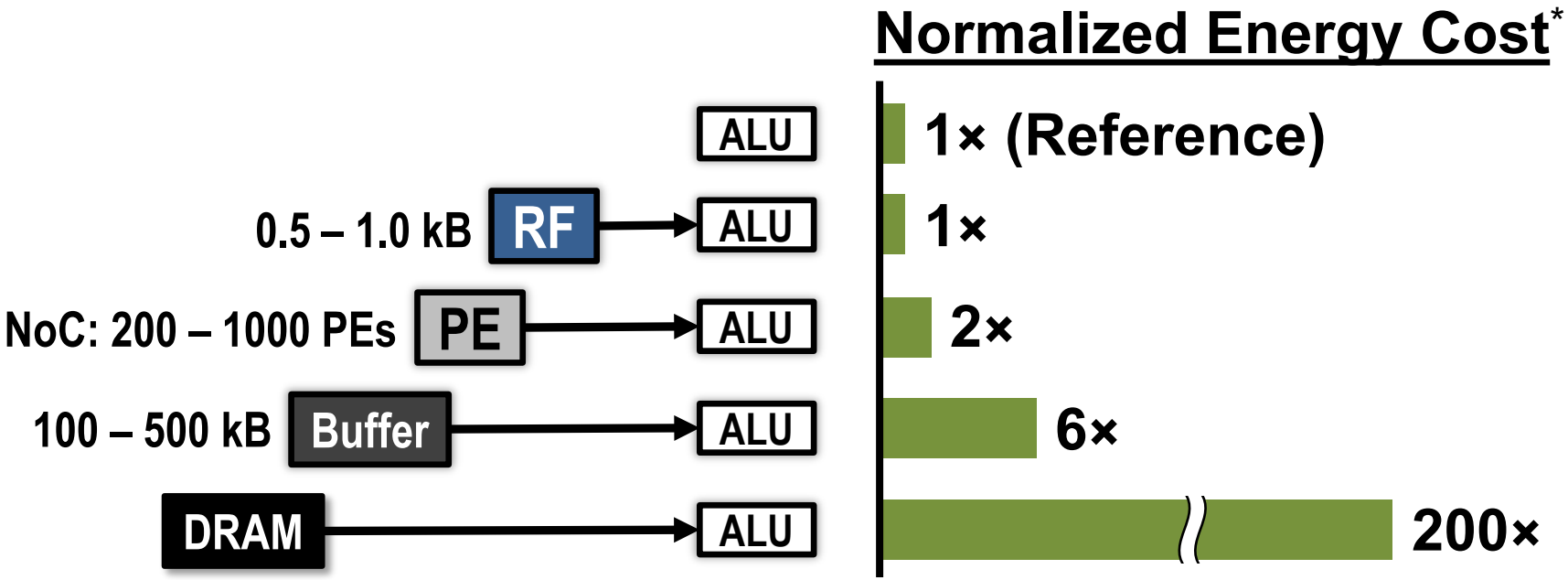


* measured from a commercial 65nm process

Low-Cost Local Data Access

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

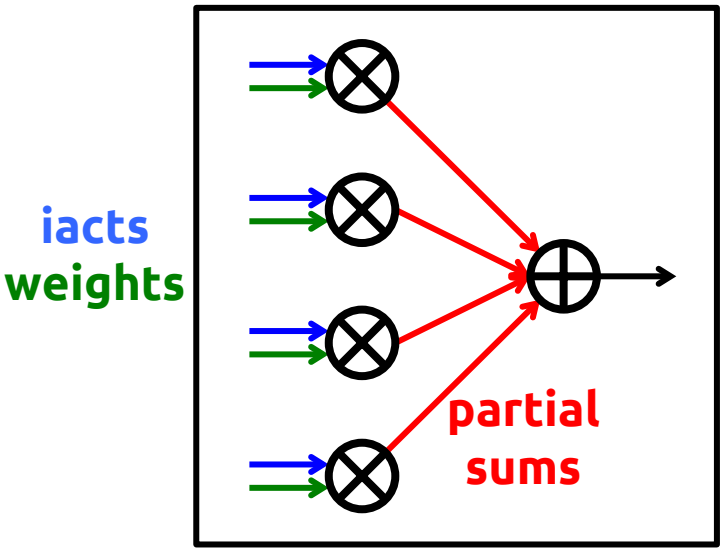
specialized **processing dataflow** required!



* measured from a commercial 65nm process

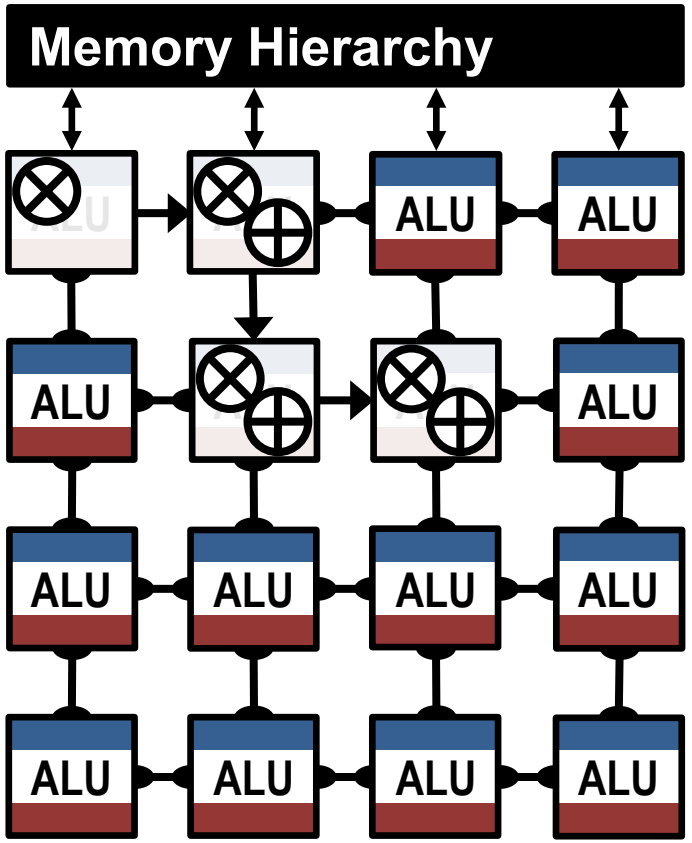
How to Map the Dataflow?

CNN Convolution



?

Spatial Architecture (Dataflow Processing)



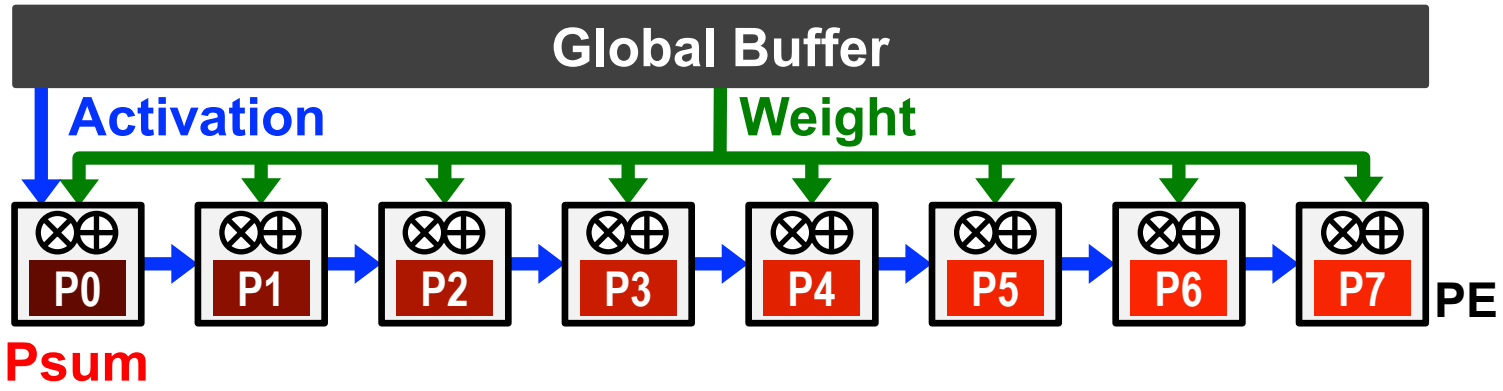
Goal: Increase reuse of input data (**input activations** and **weights**) and local **partial sums** accumulation

Dataflow Taxonomy

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

[Chen et al., ISCA 2016]

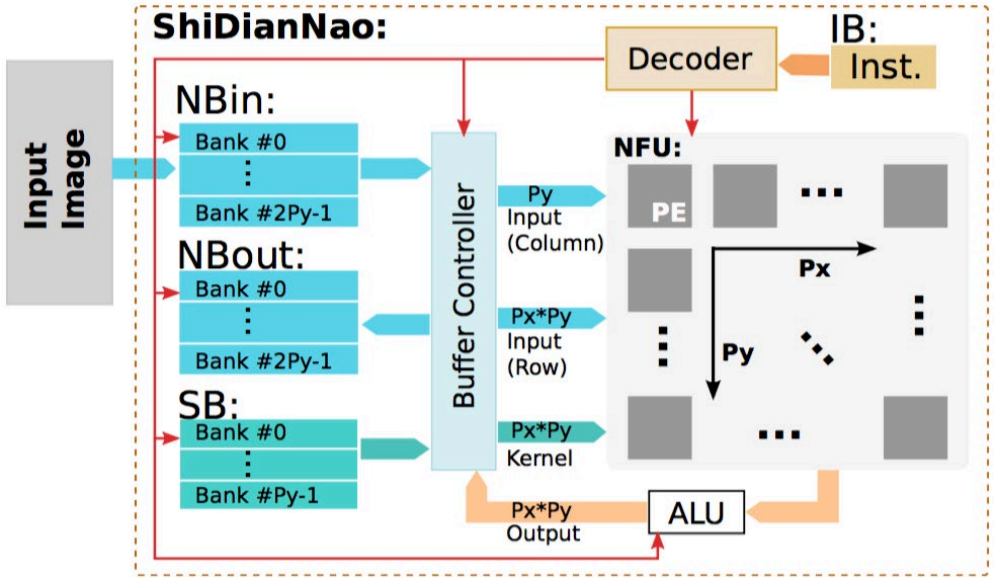
Output Stationary (OS)



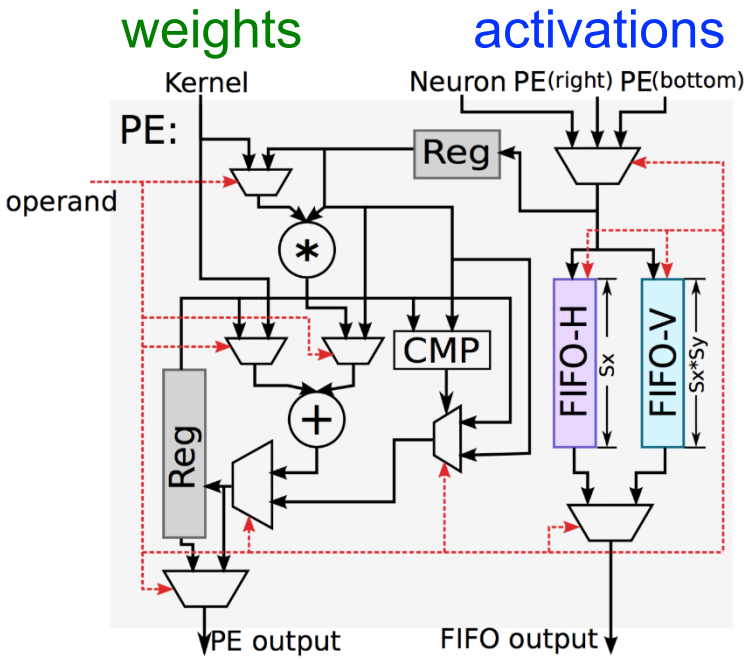
- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array

OS Example: ShiDianNao

Top-Level Architecture



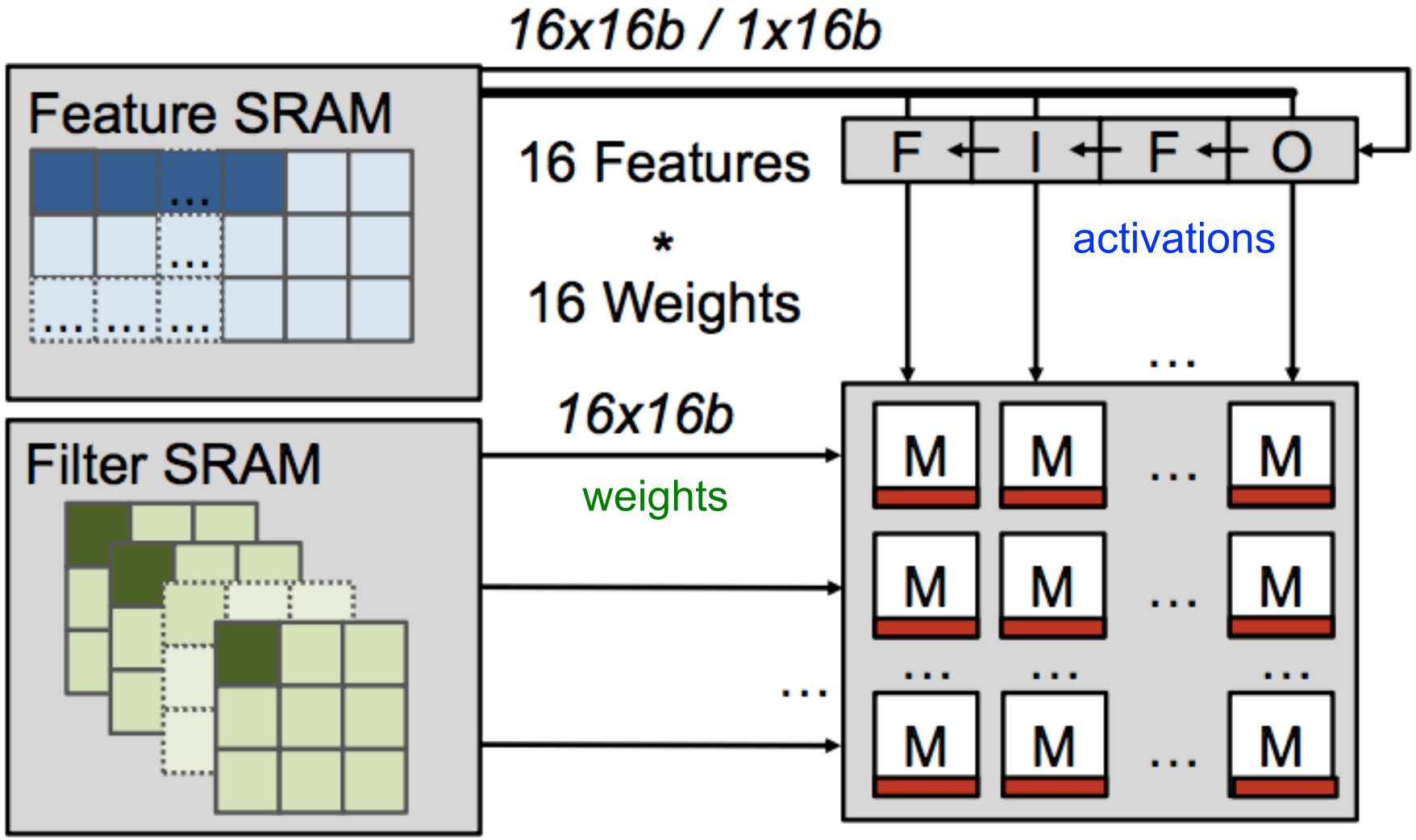
PE Architecture



- Inputs streamed through array
- Weights broadcast
- Partial sums accumulated in PE and streamed out

[Du et al., ISCA 2015]

OS Example: KU Leuven



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

1-D Convolution Einsum

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

Operational definition of Einsum says traverse all valid values of “q” and “s”… but in what order….

Traversal order (fastest to slowest): S, Q

Which “for” loop is outermost?

1-D Convolution

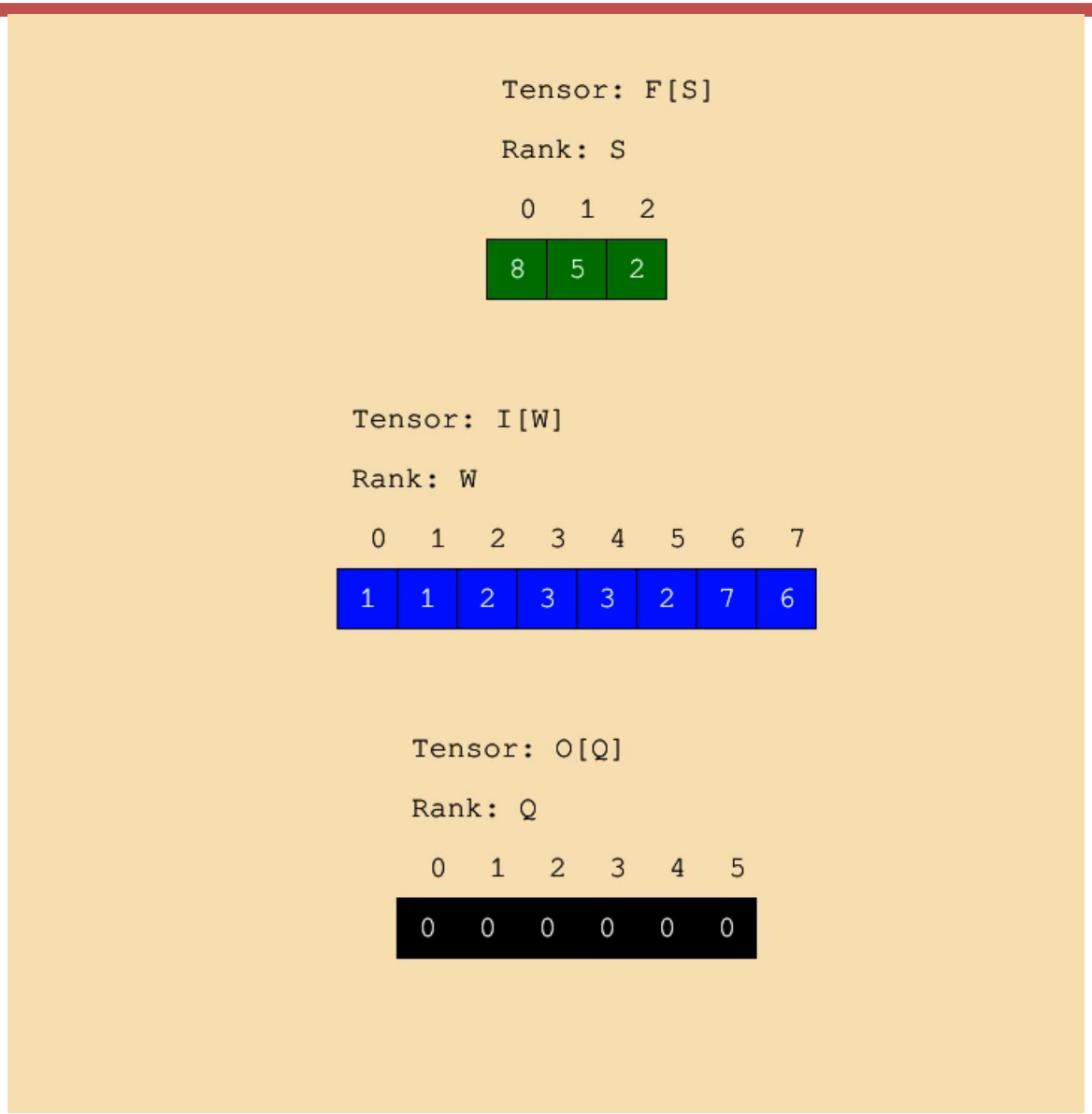

```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

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

What dataflow is this?

Is it easy to tell dataflow from the loop nest?

† Assuming: 'valid' style convolution

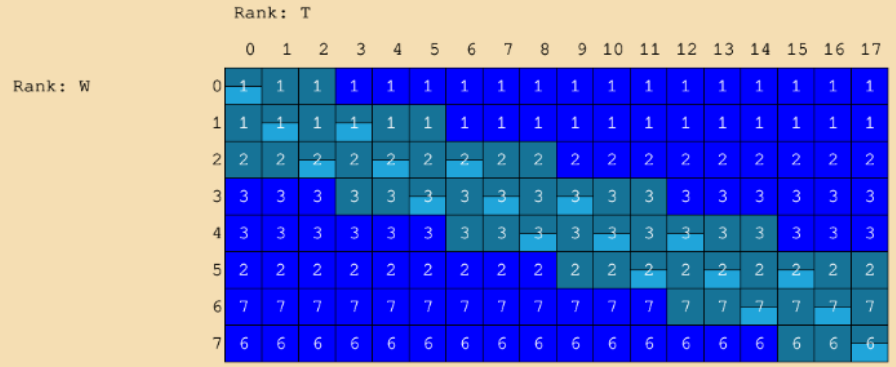
Output Stationary - Movie



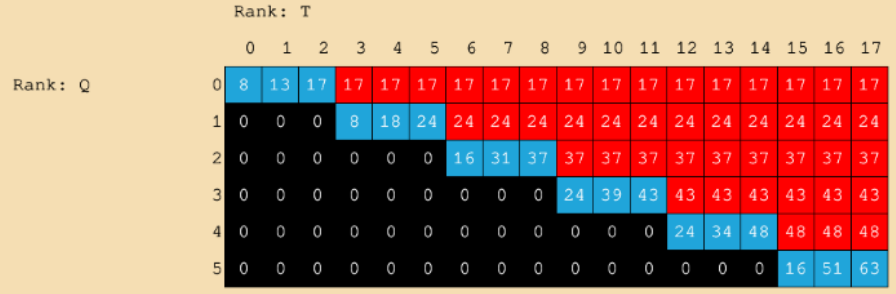
Output Stationary – Spacetime View

Tensor: F[S, T]

Tensor: I[W, T]



Tensor: O[Q, T]



CONV-layer Einsum

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

Traversal order (fastest to slowest): S, R, Q, P

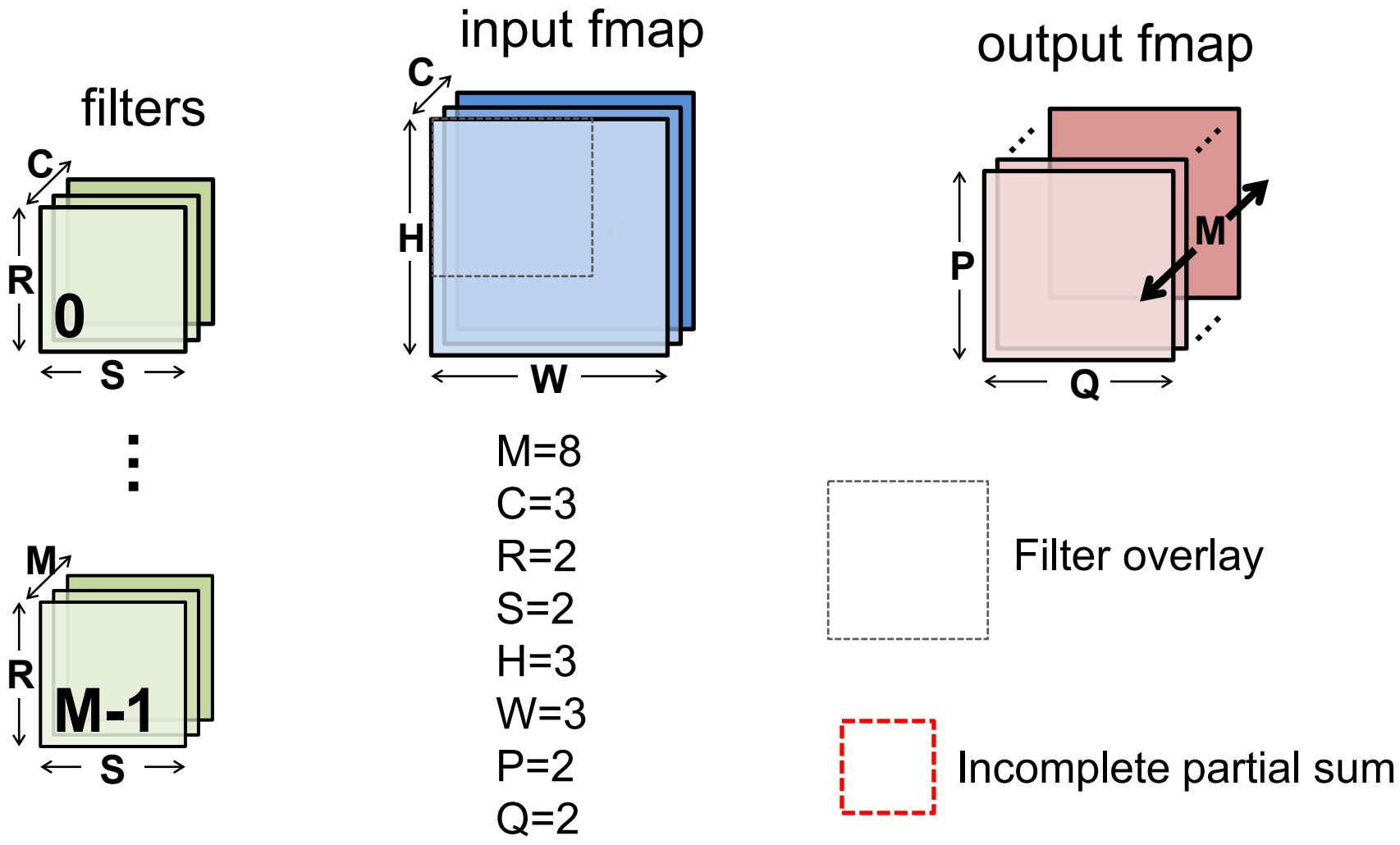
Parallel Ranks: C, M

Can you write the loop nest? I hope so 😊

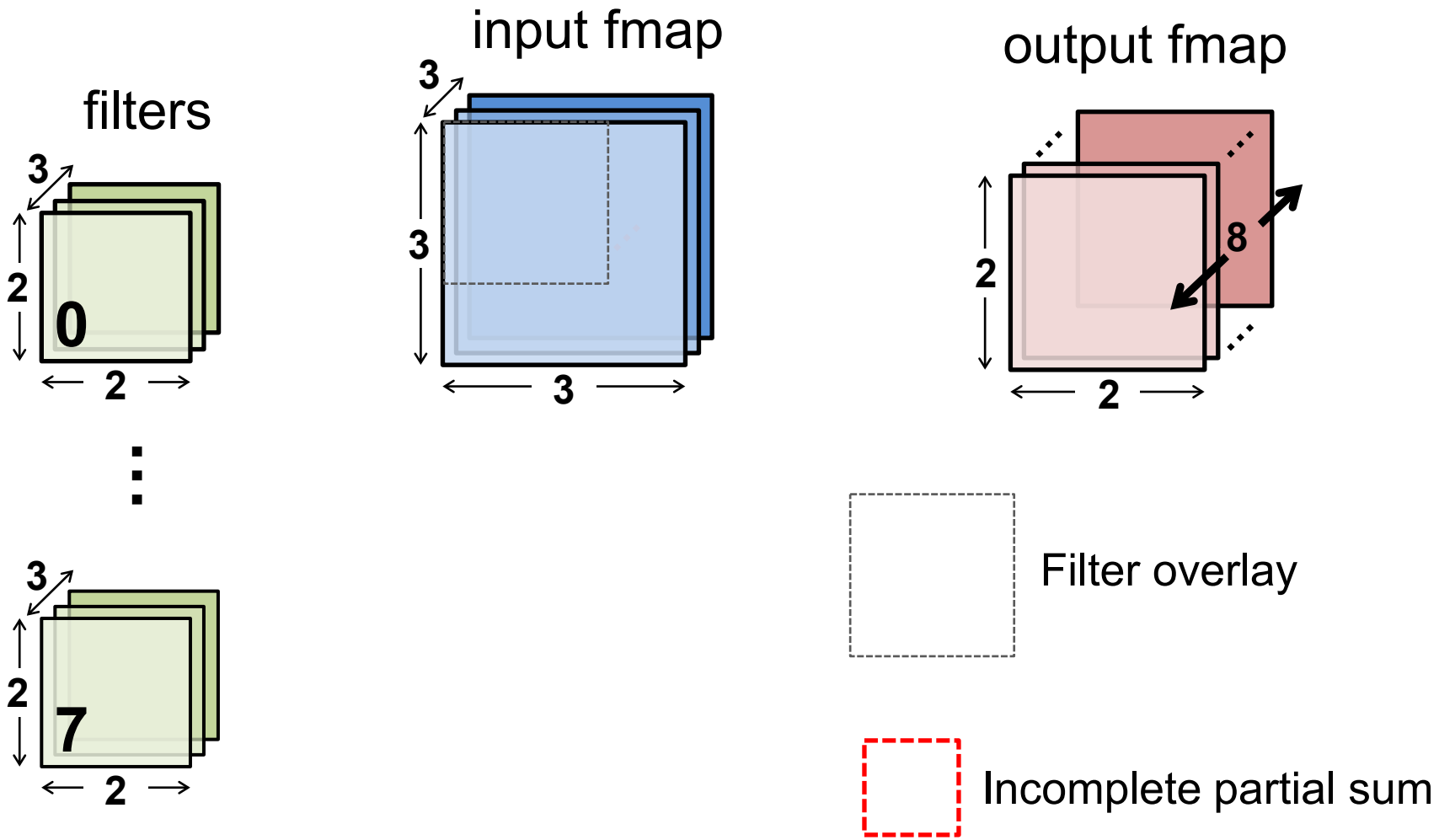
CONV Layer OS Loop Nest

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

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

CONV Layer OS Dataflow

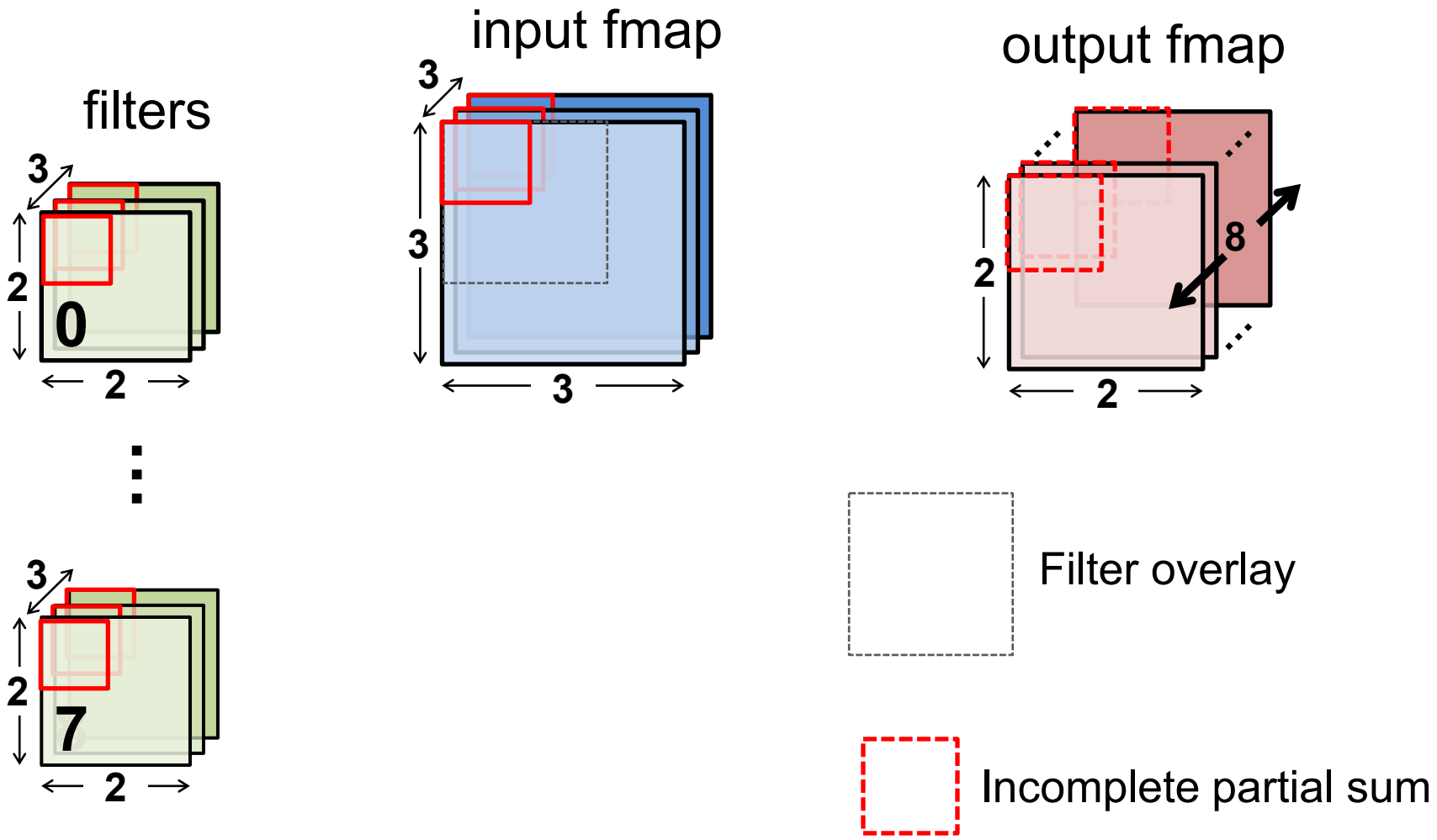


CONV Layer OS Dataflow



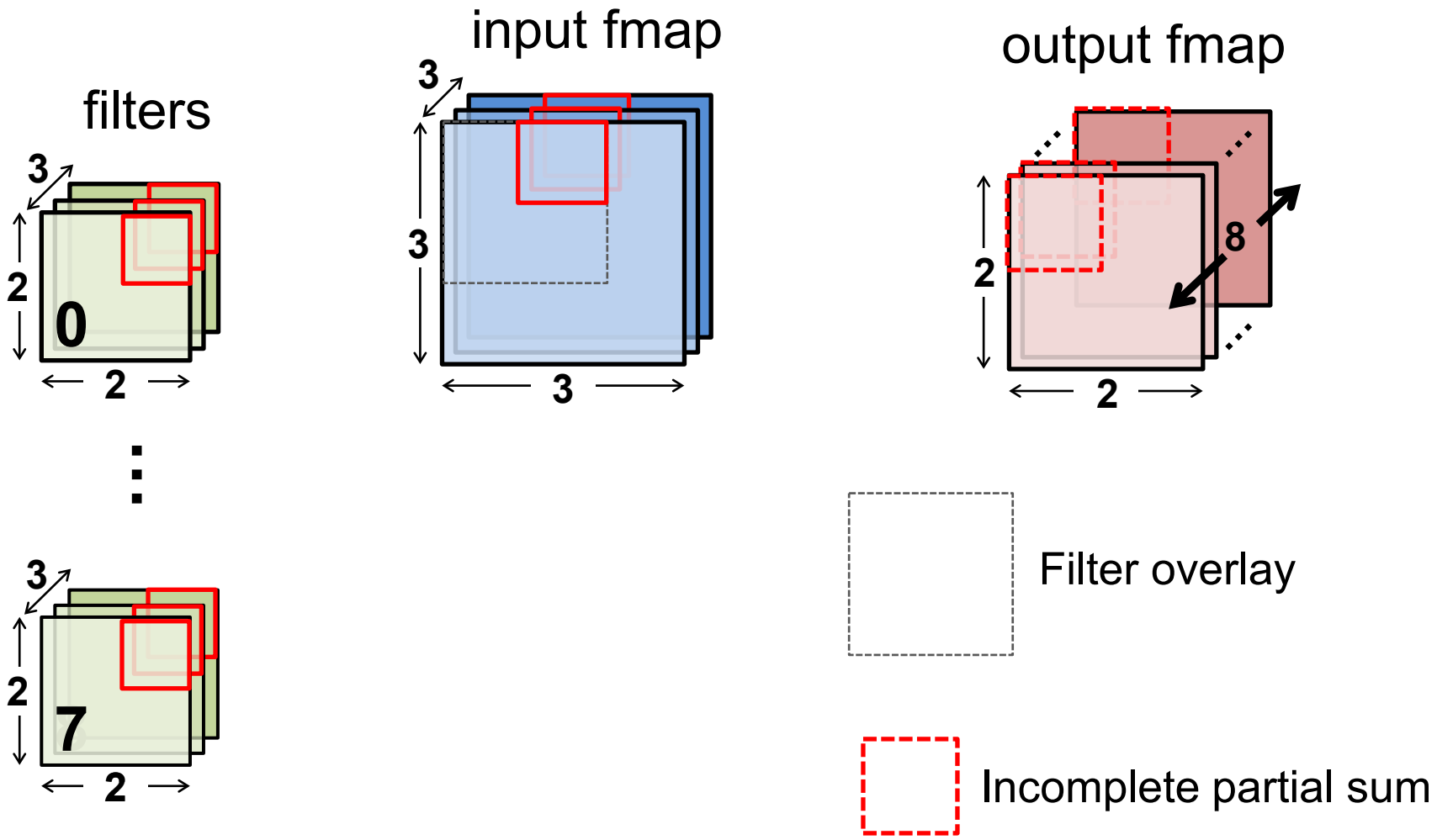
CONV Layer OS Dataflow

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



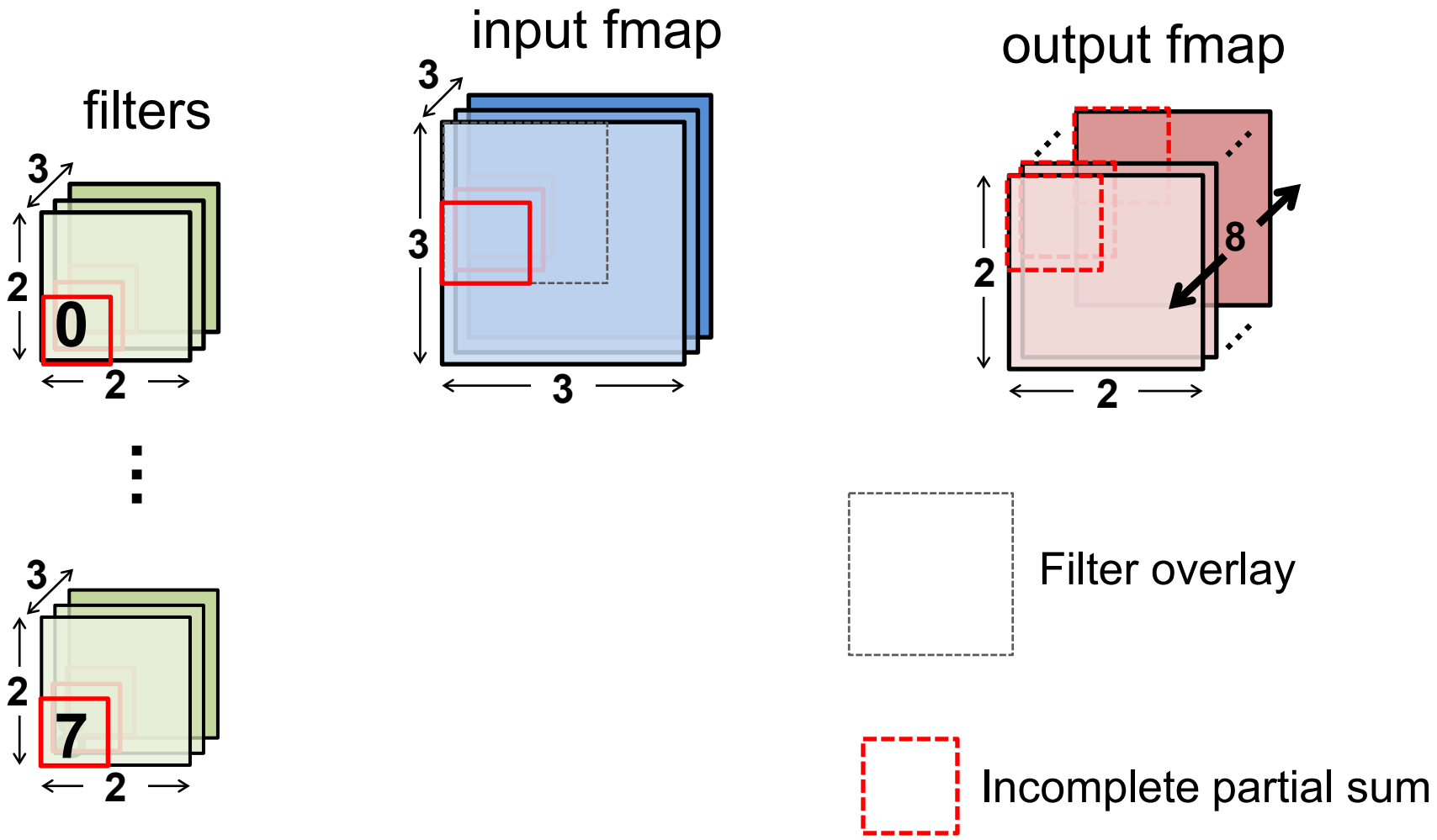
CONV Layer OS Dataflow

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



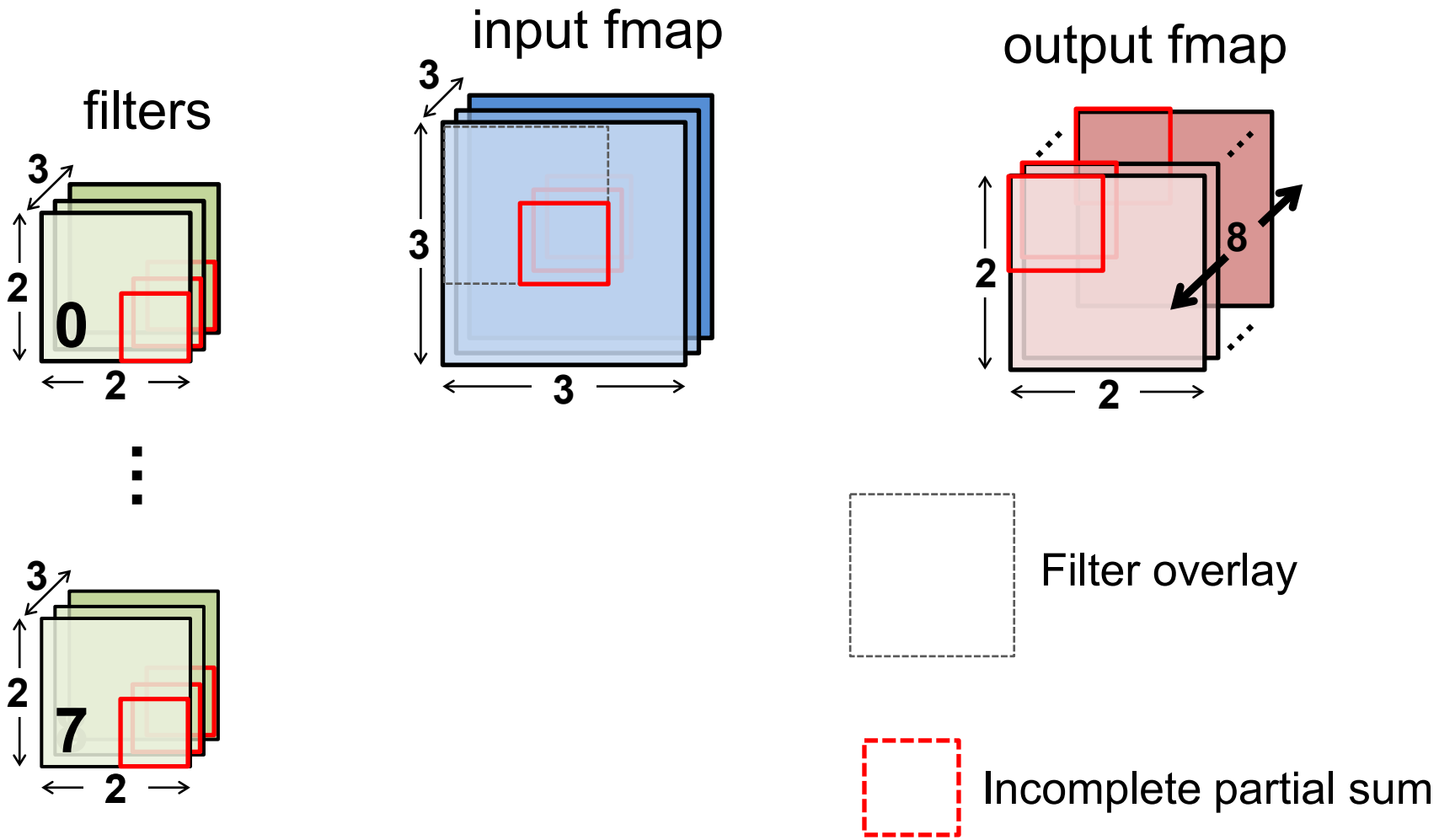
CONV Layer OS Dataflow

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



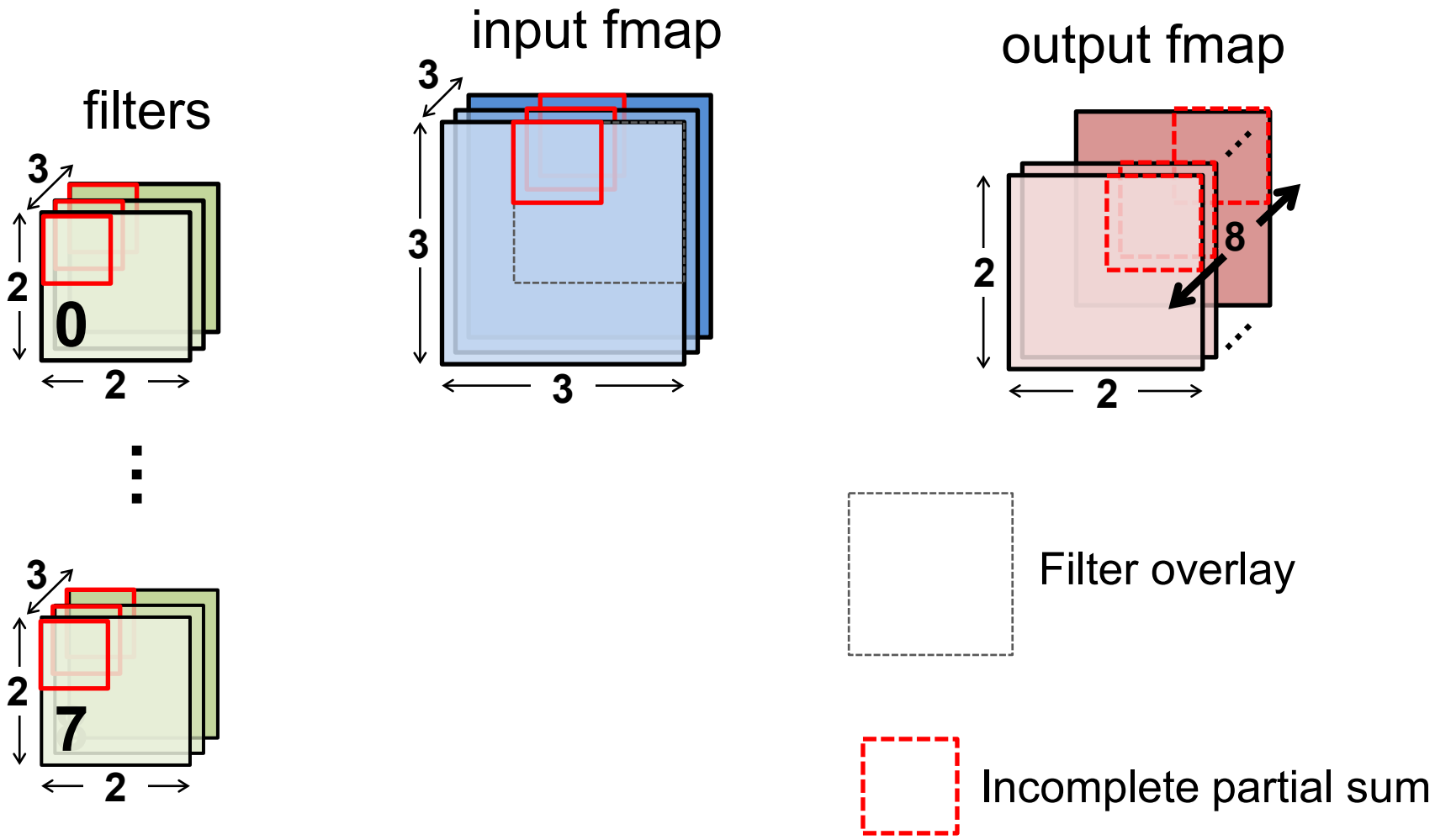
CONV Layer OS Dataflow

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



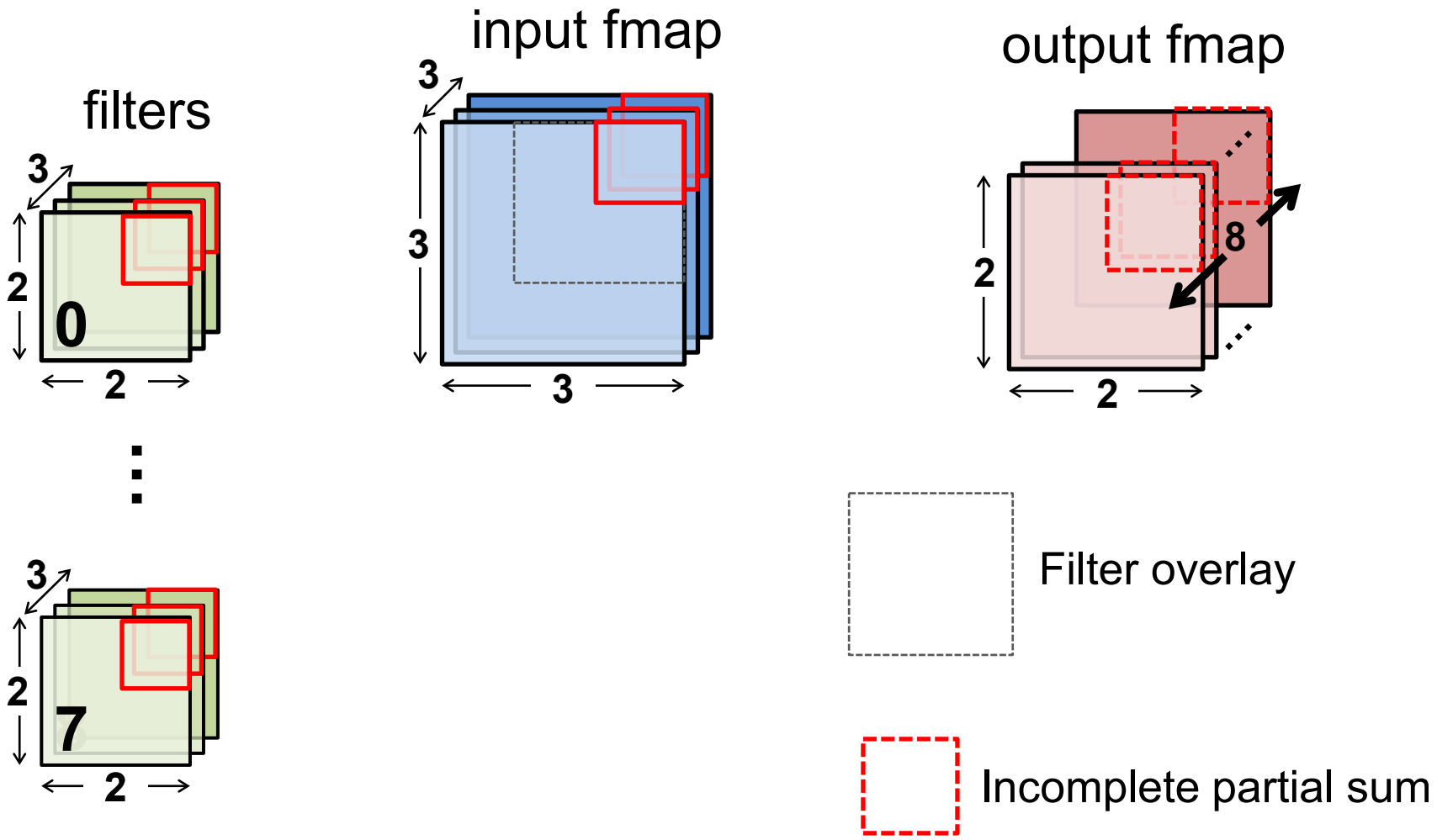
CONV Layer OS Dataflow

Start processing next output feature activations



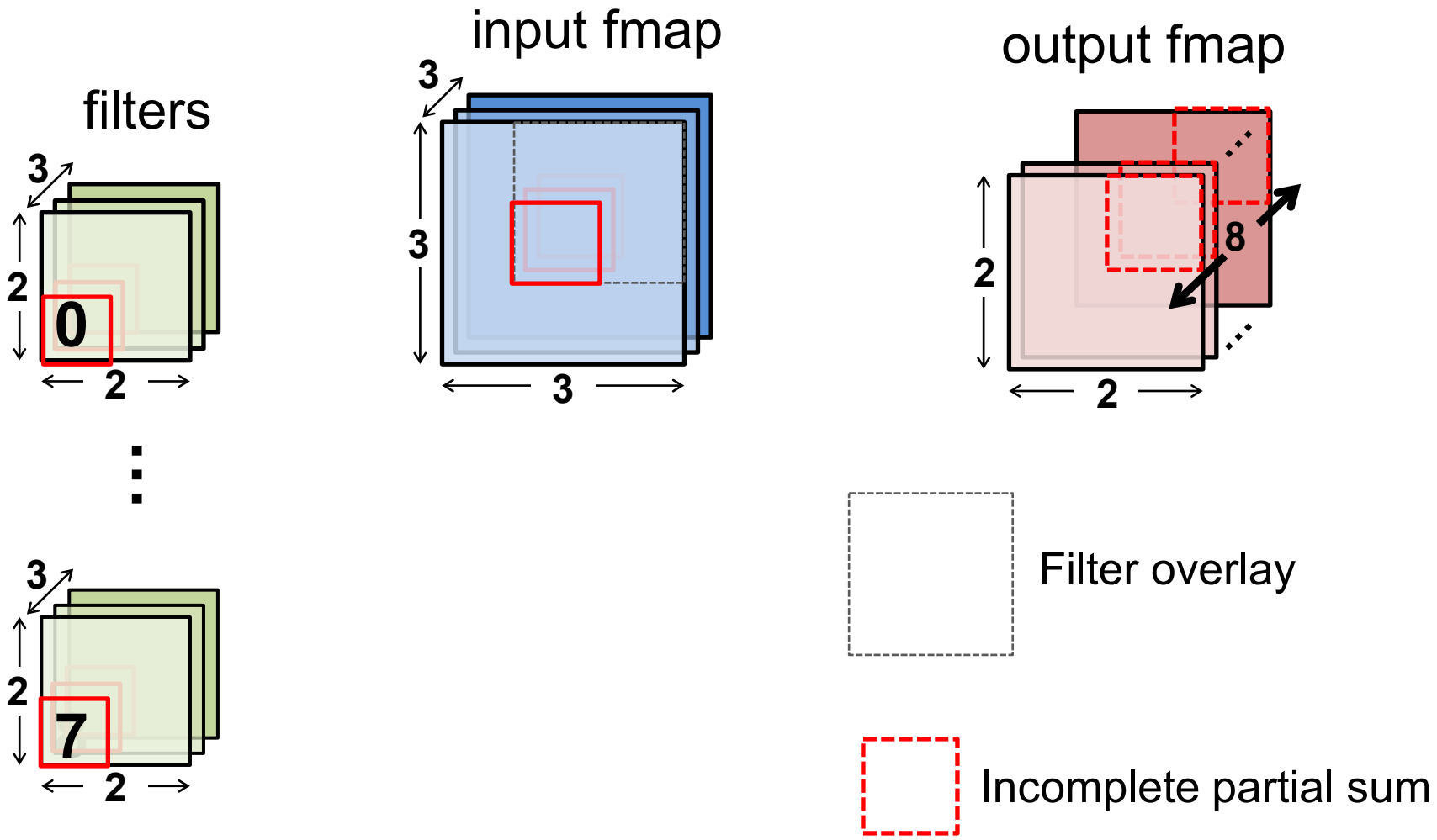
CONV Layer OS Dataflow

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



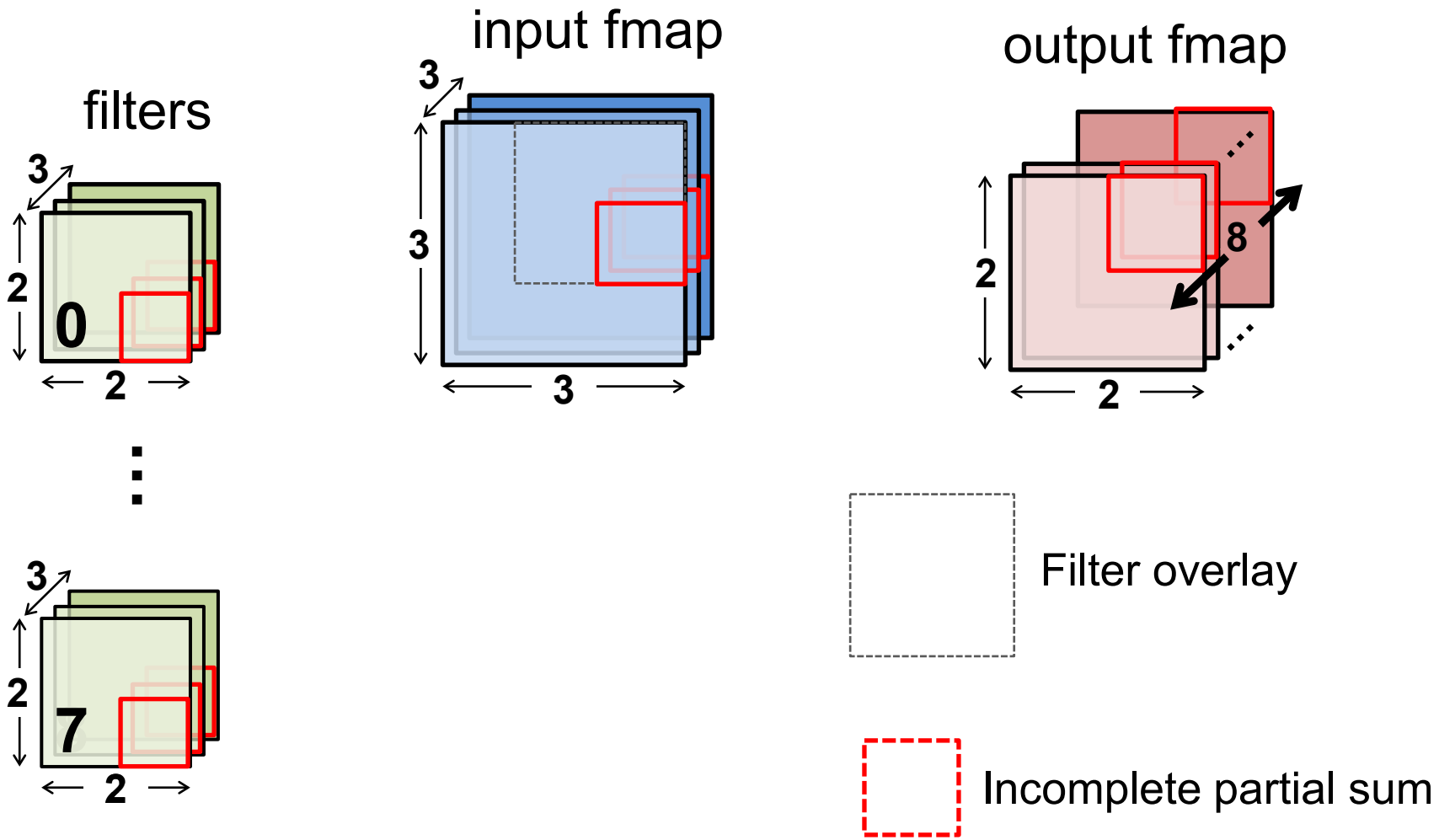
CONV Layer OS Dataflow

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



CONV Layer OS Dataflow

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



Next:

More dataflows