Accelerator Architectures for Machine Learning (AAML)

Lecture 5: Systolic Accelerator

Tsung Tai Yeh
Department of Computer Science
National Yang-Ming Chiao Tung University

Acknowledgements and Disclaimer

- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020
  CS224W: Machine Learning with Graphs, Stanford University, 2021

Outline

- Systolic Array Architecture
  - Google Tensor Processing Unit (TPU)
- Dataflow
  - Weight-stationary
  - Output-stationary
  - Input-stationary

Systolic DNN Accelerator

A Golden Age in Microprocessor Design

- A great leap in microprocessor speed ~10^6 X faster over 40 years
- Architectural innovations
  - Width: 8->16->32->64 bits (~8X)
  - Instruction level parallelism (ILP)
  - Multicore: 1 processor to 16 cores
  - Clock rate: 3 – 4000 MHz (~1000 X through technology & architecture)
- IC technology makes it possible
  - **Moore's Law**: growth in transistor count (2X every 1.5 years)
  - **Dennard Scaling**: power/transistor shrinks at the same rate as transistors are added

John Hennessy, "The Future of Microprocessors", 2017

Current Situation

- **Technology**
  - End of Dennard scaling: power becomes the key constraint
  - Slowdown of Moore's Law: transistor cost
- **Architectural Designs**
  - Inefficiency to exploit instruction level parallelism in the uniprocessor era, 2004
  - Amdahl's Law and its implications end

John Hennessy, "The Future of Microprocessors", 2017

What's Left ?

- Transistors not getting much better
- Power budget not getting much higher
- One inefficient processor/chip to N efficient processors/chip
- Only path left is **Domain Specific Architectures**
  - Just do a few tasks, but extremely well

John Hennessy, "The Future of Microprocessors", 2017

Lessons from DSA

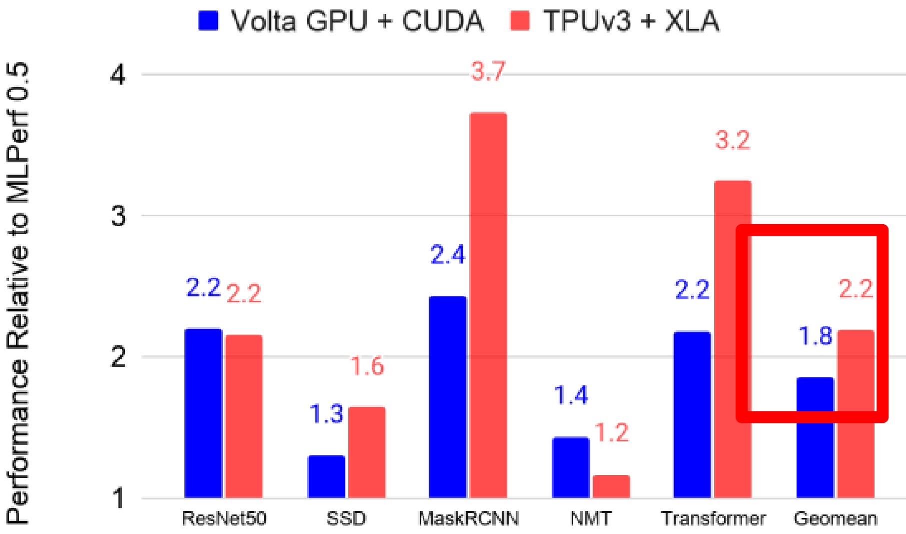
- **Logic, wires, SRAM & DRAM improve unequally**
  - **SRAM access improved only 1.3X – 2.4 X** → SRAM density is scaling slowly
  - **DRAM access improved 6.3X**
    - **Packaging innovations**
    - **High Bandwidth Memory (HBM)**
    - HBM is more energy-efficient than GDDR6 or DDR DRAM
  - **Logic improves much faster than wires and SRAM**

Operation		*Picojoules per Operation*		
		45 nm	*7 nm*	*45 / 7*
+	Int 8	0.03	0.007	4.3
	Int 32	0.1	0.03	3.3
	BFloat 16	--	0.11	--
	IEEE FP 16	0.4	0.16	2.5
	IEEE FP 32	0.9	0.38	2.4
×	Int 8	0.2	0.07	2.9
	Int 32	3.1	1.48	2.1
	BFloat 16	--	0.21	--
	IEEE FP 16	1.1	0.34	3.2
	IEEE FP 32	3.7	1.31	2.8
SRAM	8 KB SRAM	10	7.5	1.3
	32 KB SRAM	20	8.5	2.4
	1 MB SRAM[1]	100	14	7.1
GeoMean[1]		--	--	2.6
DRAM		Circa 45 nm	Circa 7 nm	
	DDR3/4	1300[2]	1300[2]	1.0
	HBM2	--	250-450[2]	--
	GDDR6	--	350-480[2]	--

Jouppi et al. ISCA, 2021

Lessons from DSA

- **Leverage prior compiler optimization**
  - Many DSAs rely on VLIW including TPUs
  - XLA (Accelerated Linear Algebra) compiler
  - XLA raises the TPU by 2.2 X compared to the same compiler 20 months ago
  - C compilers improve general purpose code 1 – 2% annually
  - Good compilers are critical to a DSA's success



Jouppi et al. ISCA, 2021

Lessons from DSA

- **Some inference applications need floating point arithmetic**
  - **Quantized arithmetic** grants area and power savings
  - But may reduce quality, delayed deployment and some apps don't work well when quantized
- **Production inference needs multi-tenancy**
  - **Sharing can lower costs and reduce latency** if applications use many models
  - Multi-tenancy suggests **fast DRAM for DSAs**, since all weights can't fit in SRAM

Lessons from DSA

- **DNN workloads evolve with DNN breakthroughs**
  - MLP drops (65% to 25%)
  - BERT appeared in 2018, yet its's already 28% of the workload
  - A transformer encode + LSTM decoder (RNN0) + a wave RNN (RNN1) is 29%
  - The importance of **programmability and flexibility for inference DSAs** to track DNN progress

Name	*Avg. Size (MB)*	*Max Size (MB)*	*Multi-tenancy?*	*Avg. Number of Programs (StdDev), Range*	*% Use 2016/ 2020*
MLP0	580	2500	Yes	27 (±17), 1-93	61%-25%
MLP1	90	N.A.	Yes	5 (±0.3), 1-5	
CNN0	60	454	No	1	5%-18%
CNN1	120	680	Yes	6 (±10), 1-34	
RNN0	1300	1300	Yes	13 (±3), 1-29	0%-29%
RNN1	120	400	No	1	
BERT0	3000	3000	Yes	9 (±2), 1-14	0%-28%
BERT1	90	N.A.	Yes	5 (±0.3), 1-5	

Jouppi et al. ISCA, 2021

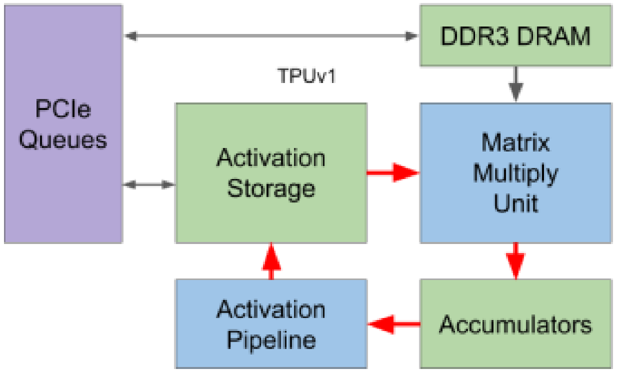
Lessons from DSA

- **DNNs grow ~1.5X per year in memory and compute**
  - DNNs grow as fast as Moore's Law
  - This rate suggests architects should provide headroom so DSAs can remain useful over their full lifetime

Model	*Annual Memory Increase*	*Annual FLOPS Increase*
CNN1	0.97	1.46
MLP1	1.26	1.26
CNN0	1.63	1.63
MLP0	2.16	2.16

Jouppi et al. ISCA, 2021

Tensor Processing Unit (TPU)

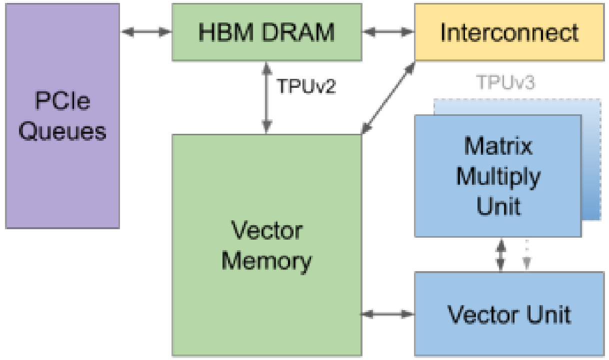


- **TPU v1**
  - Google's first DNN DSA
  - Handle **inference (serving)**
  - The **systolic array MXU** has **64K 8-bit integer Multiply Accumulate (MAC) units**
  - The CPU exchanges over PCIe
    - Model inputs and outputs
    - instructions
  - Perf/Watt compared to GPUs and CPUs
    - 30 – 80 X higher

Feature	*TPUv1*
Peak TFLOPS / Chip	92 (8b int)
First deployed (GA date)	Q2 2015
DNN Target	Inference only
Network links x Gbits/s / Chip	--
Max chips / supercomputer	--
Chip Clock Rate (MHz)	700
Idle Power (Watts) Chip	28
TDP (Watts) Chip / System	75 / 220
Die Size (mm^2)	< 330
Transistors (B)	3
Chip Technology	28 nm
Memory size (on-/off-chip)	28MB / 8GB
Memory GB/s / Chip	34
MXU Size / Core	1 256x256
Cores / Chip	1
Chips / CPUHost	4

Jouppi et al. ISCA, 2021

Tensor Processing Unit (TPU)



- **TPU v2**
  - Addresses **training**
  - Merge activation storage and the accumulators into **a single vector memory**
  - A more programmable **vector unit**
  - Support **Bfloat16** with 16 K MAC units (1/4 of the TPUv1's size)
  - The MXU was attached to the vector unit as **a matrix co-processor**
  - High **HBM DRAM** bandwidth keeps TPUv2 core well utilized
  - TPUv2 fetches its own **322-bit VLIW instructions from a local memory** rather than the host memory

Jouppi et al. ISCA, 2021

Tensor Processing Unit (TPU)

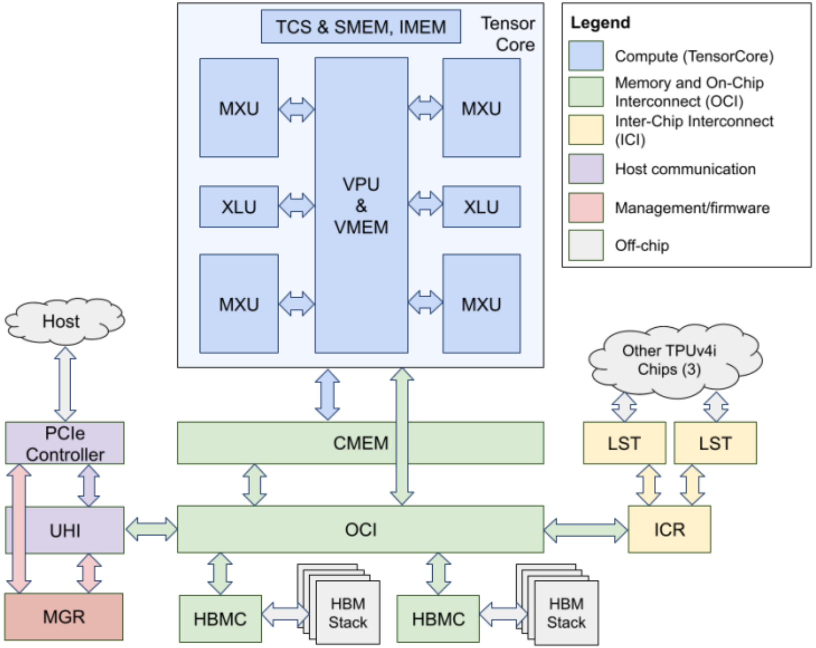
- **TPUv2**
  - Add a **chip-to-chip interconnect fabric (ICI)** enable up to 256 chips
  - **Two TensorCores per chip**
  - Prevent the excessive latency
    - Two small cores per chip vs.
    - A single large full-chip core
  - **TPUv3**
    - Has 2X the number of MXUs and HBM capacity
    - 1024 chips

Feature	*TPUv1*	*TPUv2*
Peak TFLOPS / Chip	92 (8b int)	46 (bf16)
First deployed (GA date)	Q2 2015	Q3 2017
DNN Target	Inference only	Training & Inf.
Network links x Gbits/s / Chip	--	4 x 496
Max chips / supercomputer	--	256
Chip Clock Rate (MHz)	700	700
Idle Power (Watts) Chip	28	53
TDP (Watts) Chip / System	75 / 220	280 / 460
Die Size (mm^2)	< 330	< 625
Transistors (B)	3	9
Chip Technology	28 nm	16 nm
Memory size (on-/off-chip)	28MB / 8GB	32MB / 16GB
Memory GB/s / Chip	34	700
MXU Size / Core	1 256x256	1 128x128
Cores / Chip	1	2
Chips / CPUHost	4	4

Jouppi et al. ISCA, 2021

Tensor Processing Unit (TPU)

- **TPUv4i** (i means inference)
  - Add **128 MB common memory**
    - A large data structure don't fit in vector memory
  - **Tensor DMA engine**
    - Fully decode and execute TensorCore DMA instructions
    - Enable **512B-granular 4D tensor** memory transfers between any pair of architectural memories
    - **Unified DMA engine** across local, remote and host transfer



Jouppi et al. ISCA, 2021

Tensor Processing Unit (TPU)

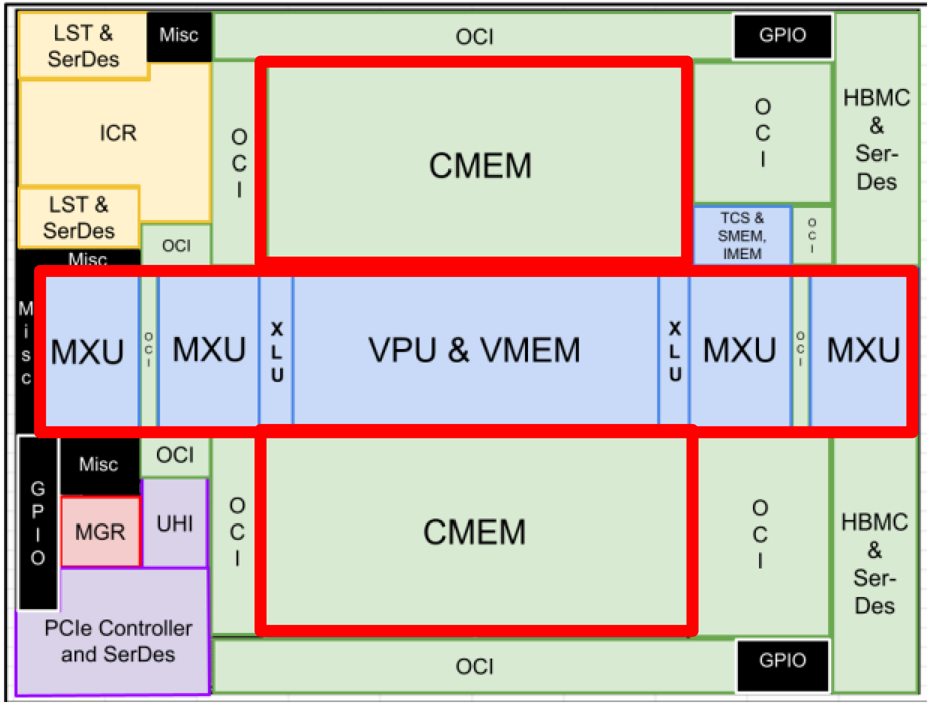
- **TPUv4i**
  - **Custom on-chip interconnect (OCI)**
    - The increase of memory bandwidth and the number of components
    - A point-to-point approach becomes too expensive -> significant routing resources/die area
    - A shared OCI connects all components on the die
  - **Wider data path**
    - 512B native access size instead of 64B cache lines
    - HBM bandwidth per core is 1.3X increased over TPUv3
    - NUMA memory system – use (spatial locality and bisection bandwidth)
    - Physically partitioned into four 128B-wide groups to optimize HBM accesses

Tensor Processing Unit (TPU)

- **TPUv4i**
  - **Arithmetic unit**
    - The **VLIW instruction needs extra fields** to handle the four MXUs and CMEM scratchpad memory -> 25% wider than TPUv3
    - Sums groups of four multiplication results together
    - Adds them to previous partial sum with a series of 32 two-input adders
    - **A four-input floating point adder**
    - Cuts the critical path through the systolic array
    - The four-input adder saves 40% area and 25% power to a series 128 two-input adders

Tensor Processing Unit (TPU)

- **TPUv4i**
  - The die is < 400 mm^2
  - **CMEM is 28% of the area**
  - OCI blocks are filled the space in the abutted floorplan
  - The die dimensions and overall layout are dominated by the TensorCore, CMEM, and SerDes



Jouppi et al. ISCA, 2021

Tensor Processing Unit (TPU)

Jouppi et al. ISCA, 2021

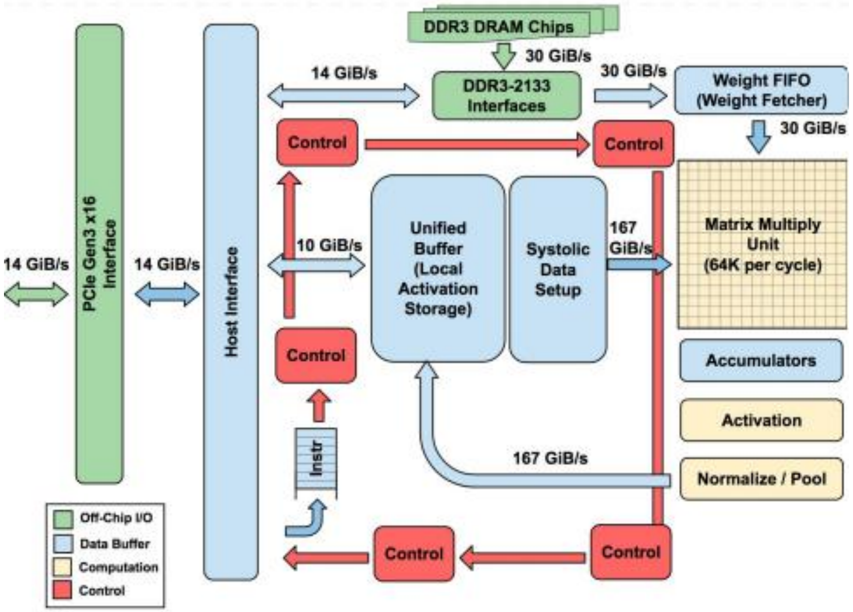
Feature	*TPUv1*	*TPUv2*	*TPUv3*	*TPUv4i*	*NVIDIA T4*
Peak TFLOPS / Chip	92 (8b int)	46 (bf16)	123 (bf16)	138 (bf16/8b int)	65 (ieee fp16)/130 (8b int)
First deployed (GA date)	Q2 2015	Q3 2017	Q4 2018	Q1 2020	Q4 2018
DNN Target	Inference only	Training & Inf	Training & Inf	Inference only	Inference only
Network links x Gbits/s / Chip	--	4 x 496	4 x 656	2 x 400	--
Max chips / supercomputer	--	256	1024	--	--
Chip Clock Rate (MHz)	700	700	940	1050	585 / (Turbo 1590)
Idle Power (Watts) Chip	28	53	84	55	36
TDP (Watts) Chip / System	75 / 220	280 / 460	450 / 660	175 / 275	70 / 175
Die Size (mm^2)	< 330	< 625	< 700	< 400	545
Transistors (B)	3	9	10	16	14
Chip Technology	28 nm	16 nm	16 nm	7 nm	12 nm
Memory size (on-/off-chip)	28MB / 8GB	32MB / 16GB	32MB / 32GB	144MB / 8GB	18MB / 16GB
Memory GB/s / Chip	34	700	900	614	320 (if ECC is disabled)
MXU Size / Core	1 256x256	1 128x128	2 128x128	4 128x128	8 8x8
Cores / Chip	1	2	2	1	40
Chips / CPUHost	4	4	4	8	8

TPU Instruction Set Architectures

- TPU instruction follows the **CISC** fashion
- Average clock cycles per instructions > 10
- **No** program counter and branch instruction
- In-order issue
- SW controls buffer, pipeline synchronization
- A dozen instructions overall, five key ones
  - Read_Host_Memory
  - Read_Weights
  - MatrixMultiply/Convole
  - Activate
  - Write_Host_Memory

TPU Microarchitecture

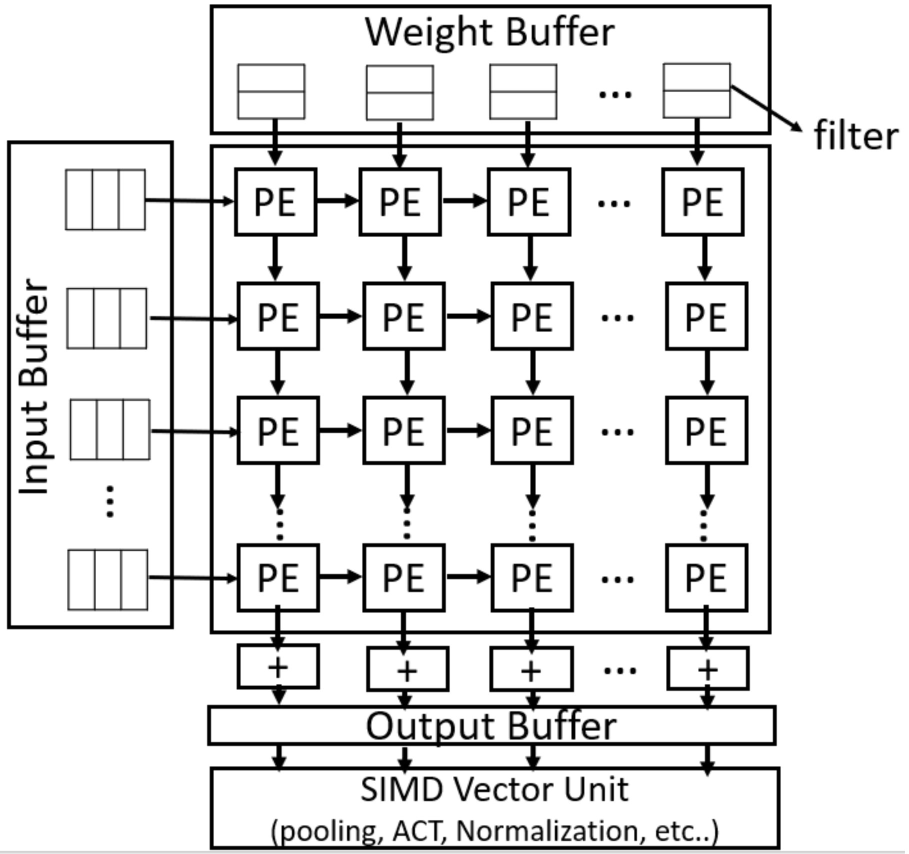
- **4-stage overlapped execution**, 1 instruction type/ stage
- Execute other instructions while MM is busy
- Read_Weight doesn't wait for weights fetched from DRAM
- The MM unit uses **not-ready signal** to indicate data aren't available in unified and Weight FIFO buffer



Jouppi et. al, ISCA 2017

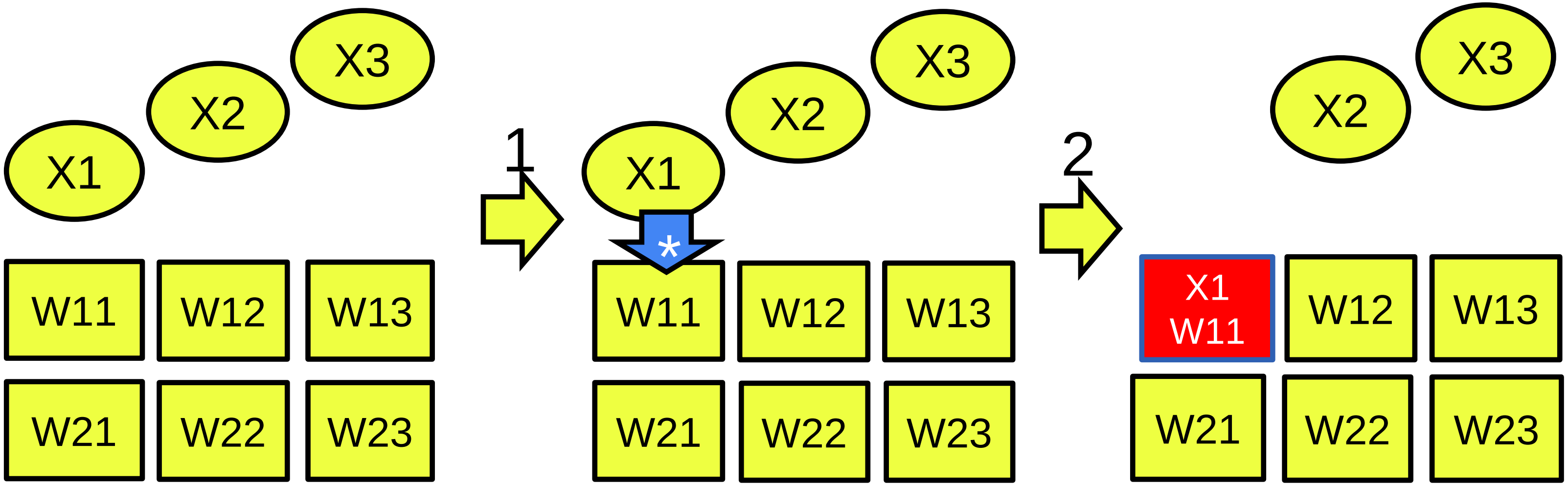
TPU Micro-architecture

- Each PE performs Multiply-and Accumulate (MAC) operation
- The unified memory buffer is decomposed into input, weight, and output buffer
- Each weight buffer stores weights of a filter
- At each cycle, inputs are pushed in the PE horizontally
- Partial sums flow vertically



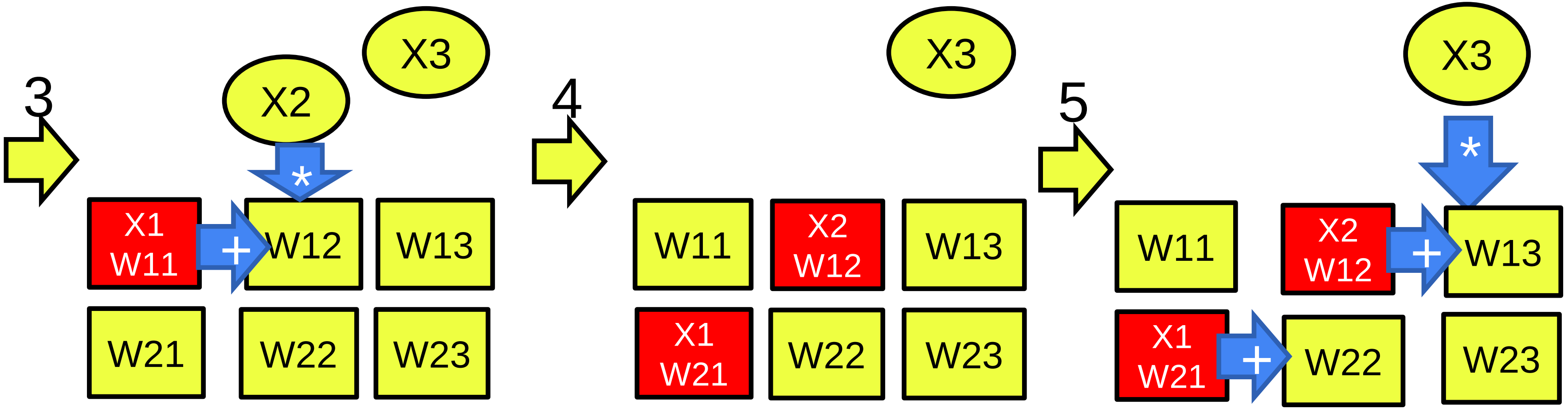
Systolic Execution in TPU

- Reading a large SRAM is much more expansive than arithmetic
- Using systolic execution to reduce R/W of the unified buffer

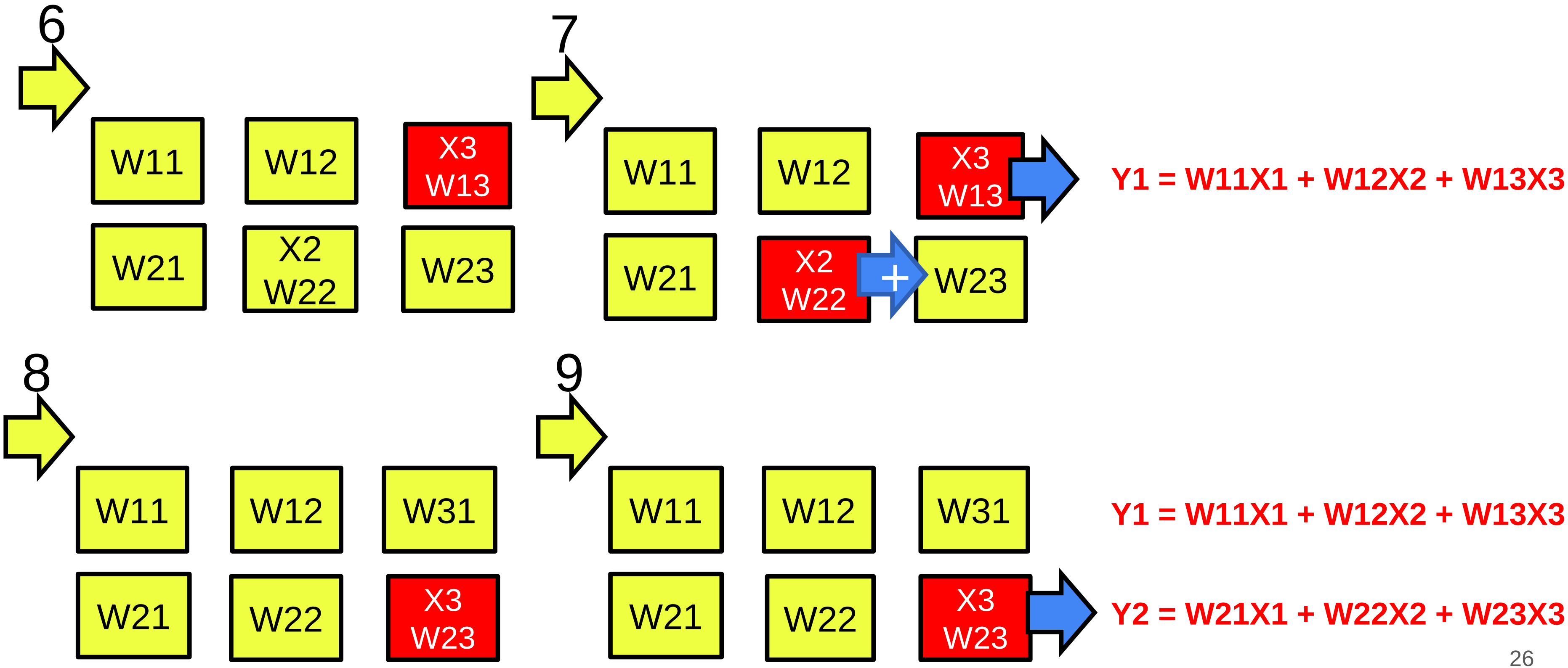


Systolic Execution in TPU

- Reuse input values
- Relies on data from different directions arriving at each array at regular interval to do the calculation



Systolic Execution in TPU



TPU Case Study

- How to map input feature map and filter (weight) to TPU ?
- Suppose the size of the input feature map is 4 x 4, and the size of filter is 2 x 2.

m x m

a_0	a_1	a_2	a_3
a_4	a_5	a_6	a_7
a_8	a_9	a_{10}	a_{11}
a_{12}	a_{13}	a_{14}	a_{15}

Input

*

k x k

W_0	W_1
W_2	W_3

Filter

=

(m-k+1)(m-k+1)

C_0	C_1	C_2
C_3	C_4	C_5
C_6	C_7	C_8

Output

TPU Case Study

- How to map input feature map and filter to TPU ?
- How many cycles takes to complete the CONV of one feature map with 2 x 2 filter, # of filter = 1 ?
    - $(m - K + 1)^2 + K^2 - 1 + (\# \text{ of filter} - 1)$

m x m

a_0	a_1	a_2	a_3
a_4	a_5	a_6	a_7
a_8	a_9	a_{10}	a_{11}
a_{12}	a_{13}	a_{14}	a_{15}

Input

cycles

			a10	a9	a8	a6	a5	a4	a2	a1	a0	W0
		a11	a10	a9	a7	a6	a5	a3	a2	a1	0	W1
0	a14	a13	a12	a10	a9	a8	a6	a5	a4	0	0	W2
a15	a14	a13	a11	a10	a9	a7	a6	a5	0	0	0	W3

Weight buffer

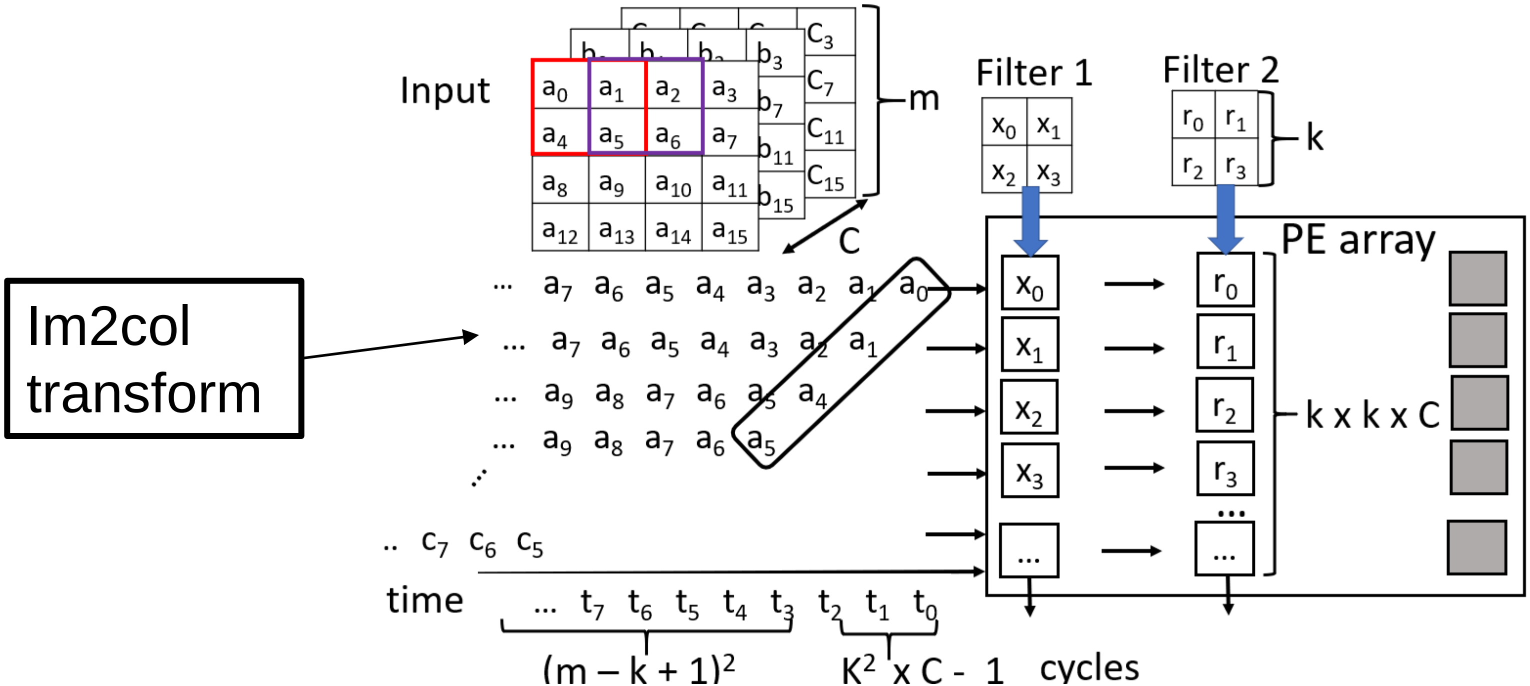
$(m - K + 1)^2$ cycles

$K^2 - 1$ cycles

Im2Col

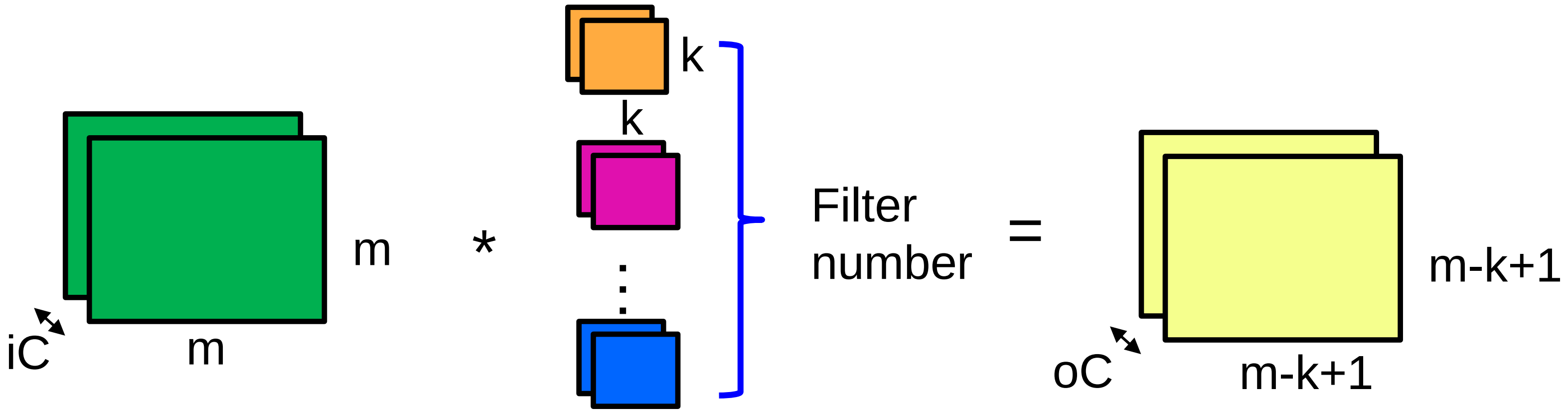
TPU Case Study

- The CONV weight stationary data flow



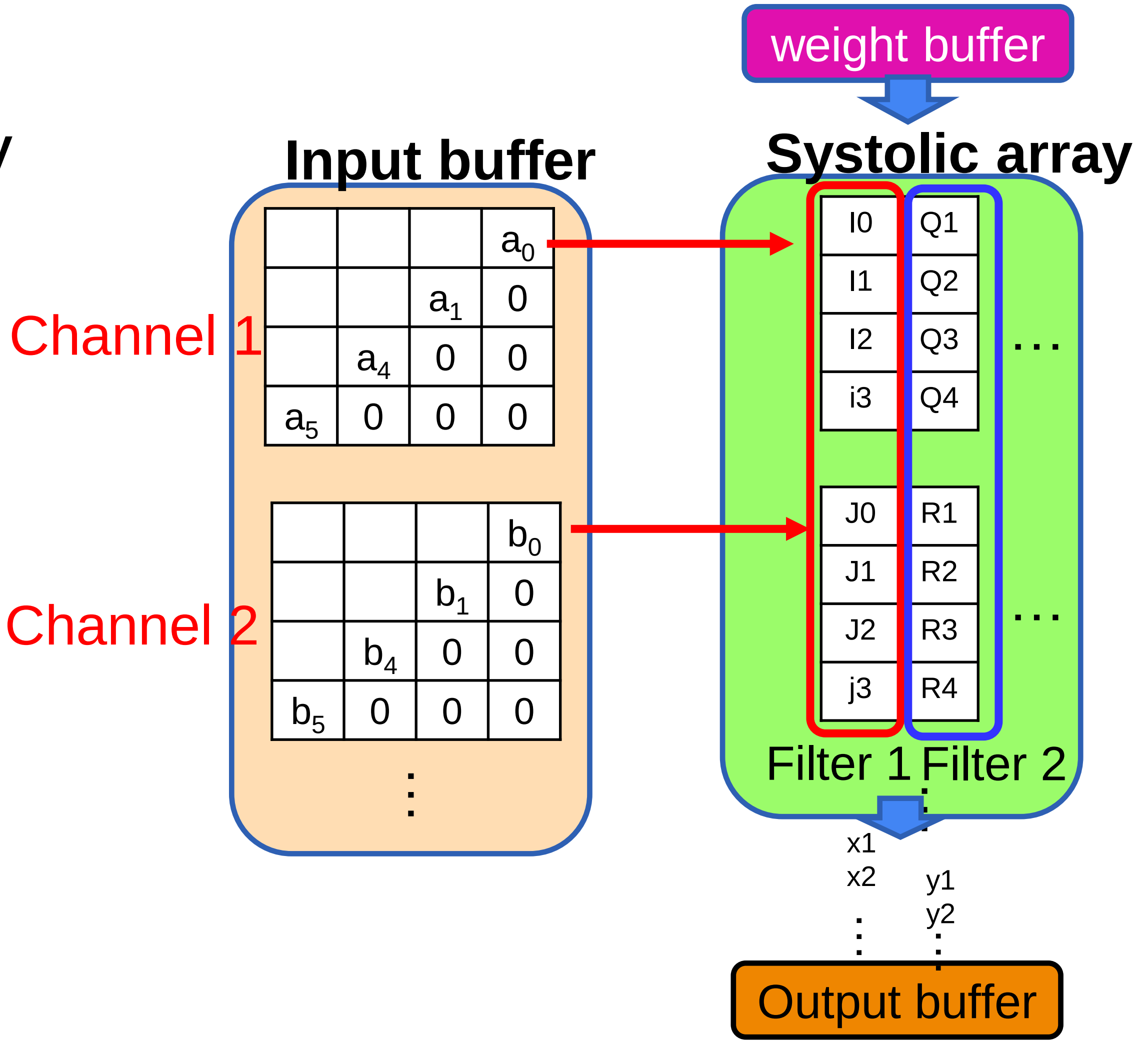
TPU Case Study

- In real-world model, a DNN model often has multiple channels and filters
- How many ops take to complete a CONV in the systolic array ?
  - (m – k + 1) x (m – k + 1) x (k x k x iC x oC)



TPU Case Study

- How to map CONV to the systolic array ?
- Systolic array contains multiple PEs
- Each filter element is placed on the local buffer of each PE



TPU Case Study

- How many cycles takes to complete a CONV ?
  - Systolic array size: 128 x 128
  - Kernel size: 2 x 2
  - Input channel: 256
  - Input size: 10 x 10
  - The number of filter: 16

1. 128 x 128 systolic array can execute floor(128/(2 x 2)) = 32 channels
2. The systolic array needs to take ceil(256/32) = 8 times
3. Each input takes $(10 – 2 + 1)^2 + (16 - 1) = 96$ cycles
4. Total = $96 \times 8 + (2^2 \times 32 - 1) =$ 895 cycles

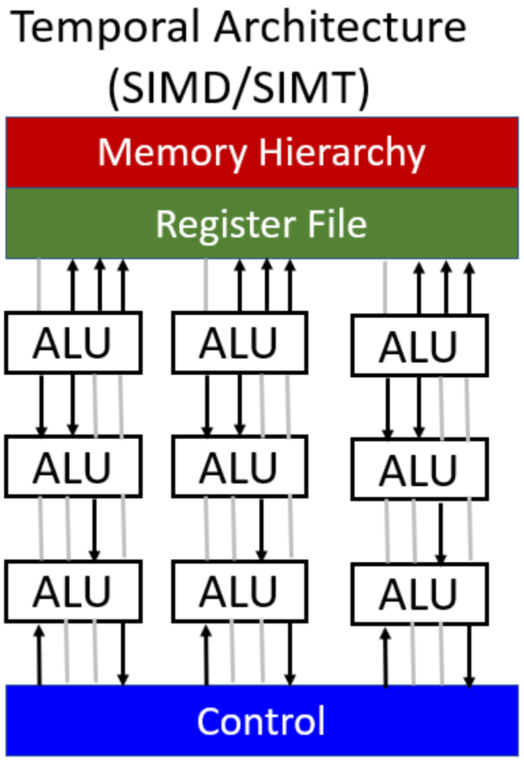
Takeaway Questions

- How does TPU reduce the energy consumption ?
  - (A) Employ the weight stationery data flow
  - (B) Increase the clock frequency of PEs
  - (C) Increase the number of PEs
- Given a DNN layer with 2 x 2 filter with a single channel, how many cycles will take before activate the first row of the systolic array?
  - (A) 3
  - (B) 4
  - (C) 5

Dataflow DNN Accelerator

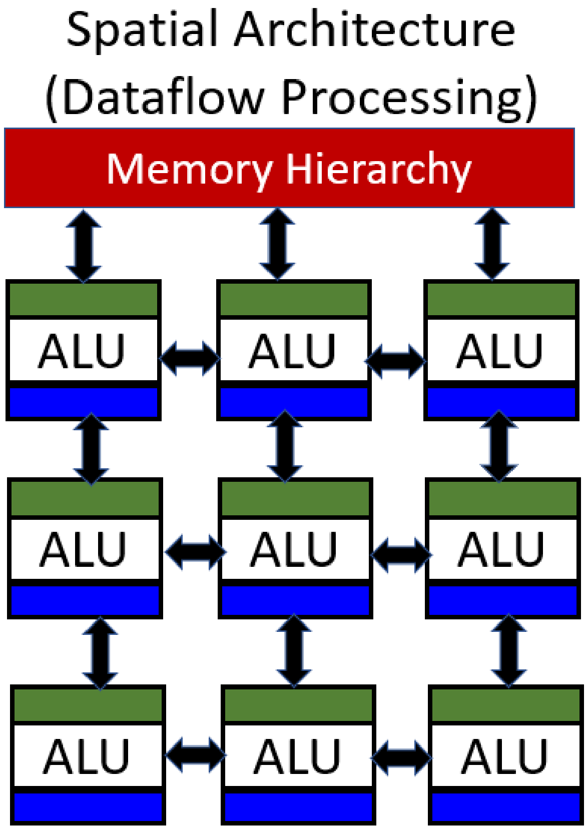
Design Aspects of Temporal Accelerator (TA)

- Centralized control for ALUs
- ALUs can only fetch data from the memory hierarchy
- ALUs “cannot” communicate directly with each other
- Why TA becomes popular? Parallelism
- Design aspects for DNN workloads
  - **Reduce # of multiplication** -> increase throughput
  - **Ordered computation (tiling)** -> improve memory subsystem



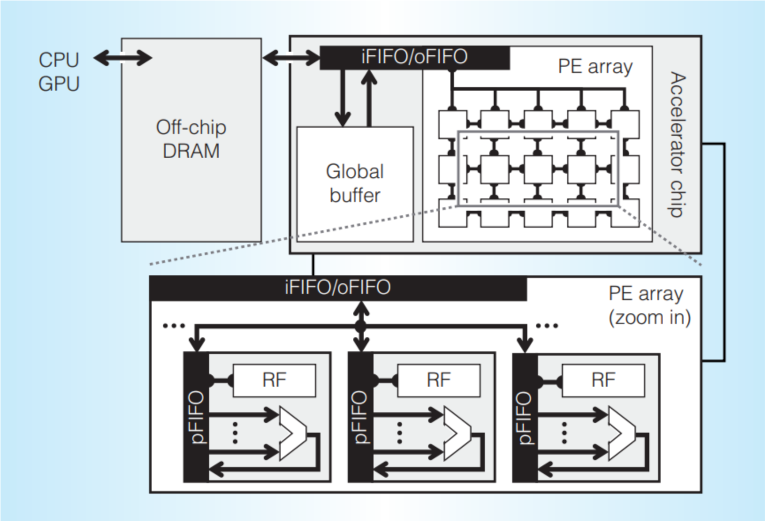
Design Aspects of Spatial Accelerator (SA)

- **ALUs**
  - Can pass data from one to another directly
  - Can have its own control logics and local memory (registers)
- **Dataflow processing**
  - Programmable -> dynamic vs static graphs
  - Dynamic Mapping -> increase data reuse -> energy-efficiency
- **Why SA are popular on DNN workloads?**
  - Consume lower power & high throughput
  - Why? Data reuse -> reduce data movement



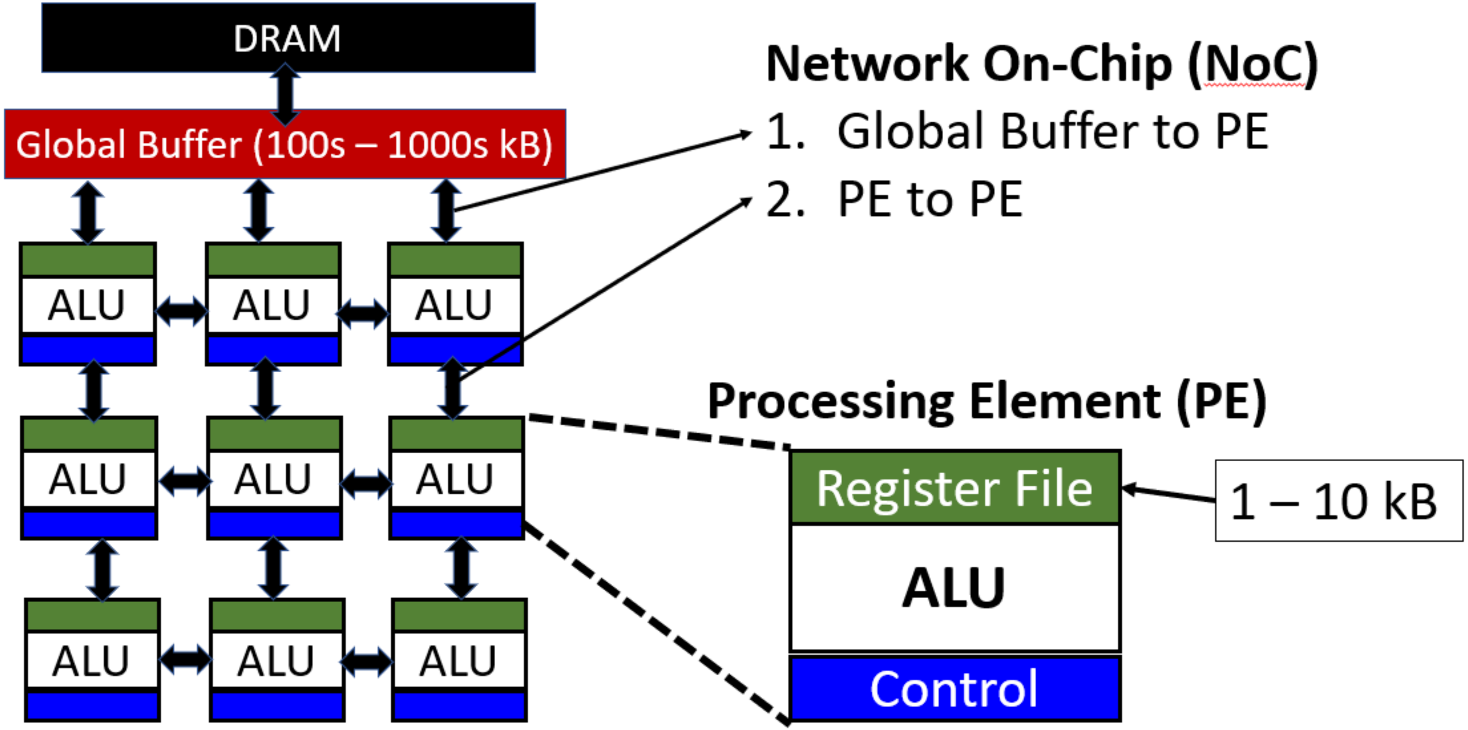
Spatial Array Architecture

- **Spatial array architecture comprises**
  - An array of processing elements (PE)
  - Off-chip DRAM
  - Global buffer
  - Network-on-chip (NOC)
  - Register file (RF) in the PE
- **Input and output FIFO (i/oFIFO)**
  - Use to communicate DRAM, global buffer, and PE
- **PE FIFO (pFIFO)**
  - Control the traffic going in and out of ALU



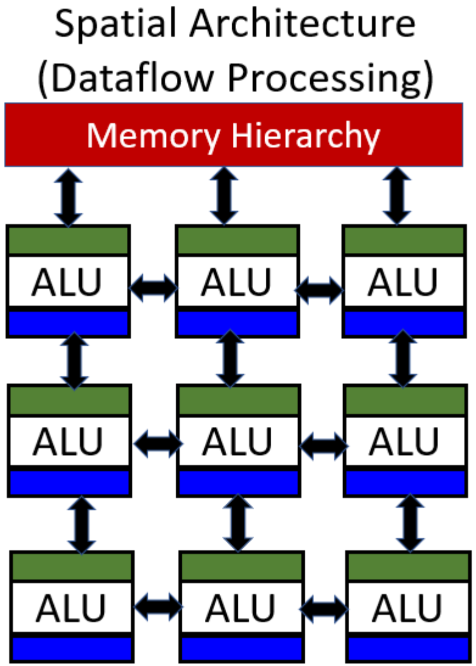
Chen et al, IEEE MICRO, 2018

Spatial Architecture for DNN

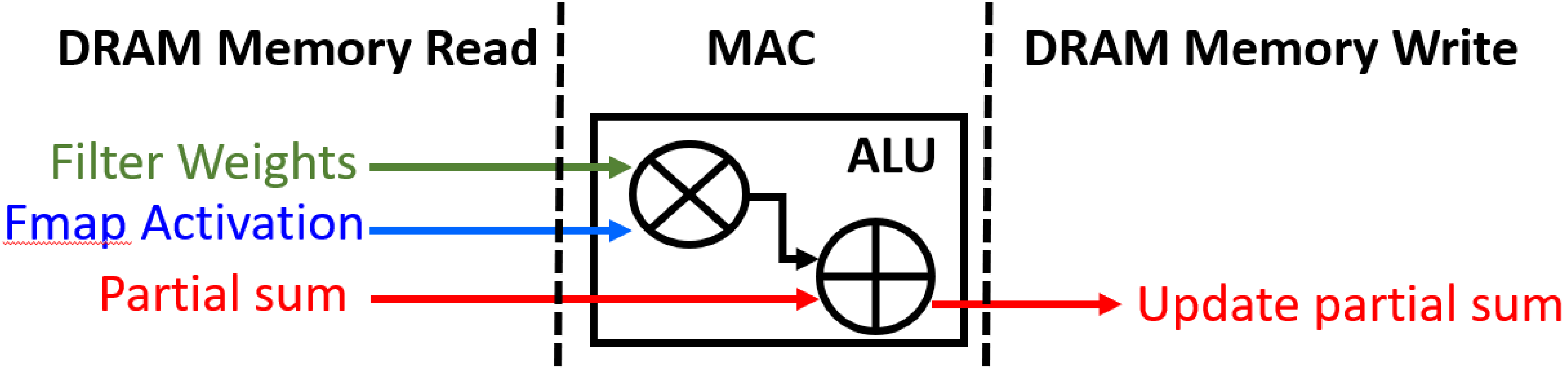


Challenges of Spatial Accelerators

- Memory access is the bottleneck
  - AlexNet has 2896M DRAM accesses required
  - How to decrease expensive DRAM accesses ?
  - Intelligent distributed data allocation
- Varying parameters in DNN models
  - Each layer has different computation volume
  - Different operations in DNN layers and models

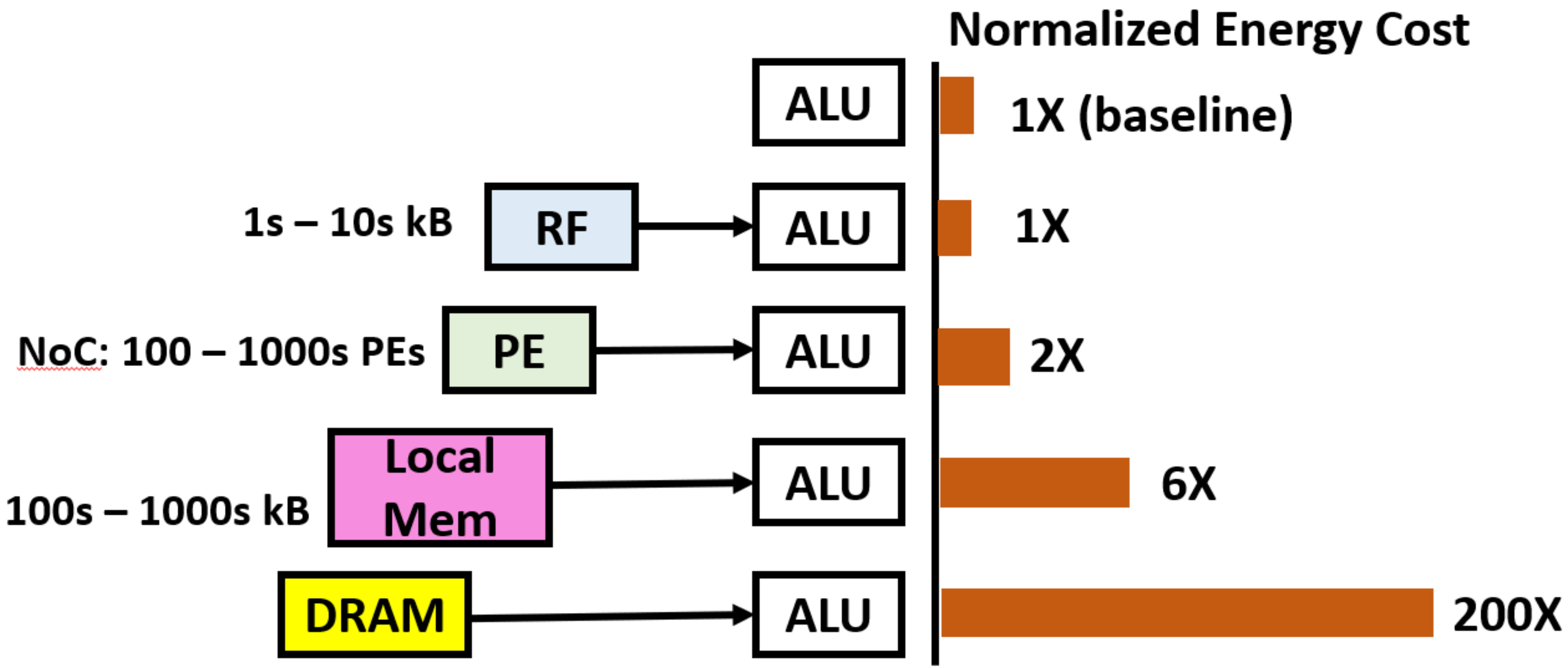


Improve Spatial Accelerator Energy-Efficiency ?



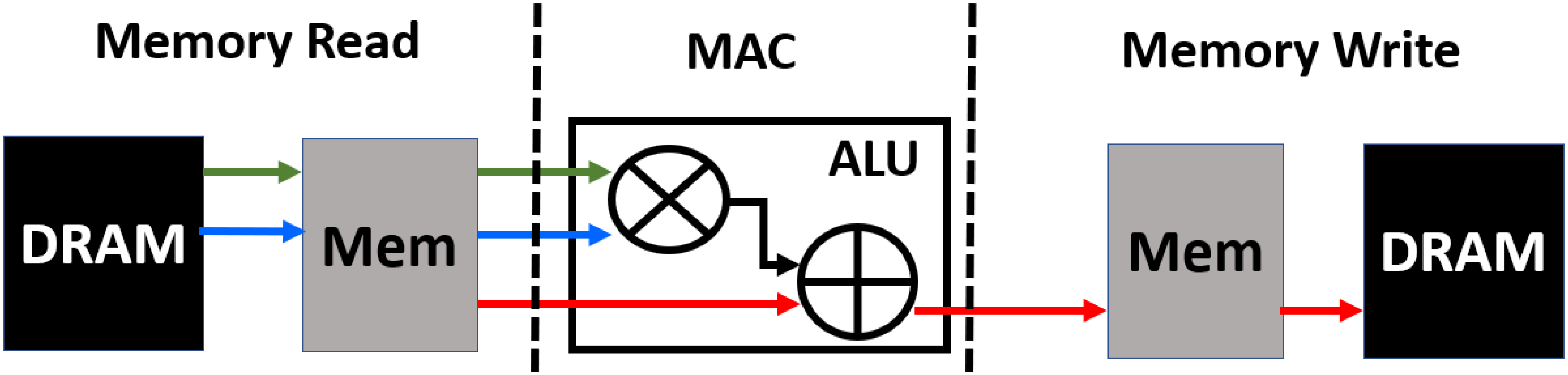
Worst Case: All memory R/W accesses from DRAM

Energy Cost of Memory Access



From a commercial 65 nm process

Data Reuse on Local Memory



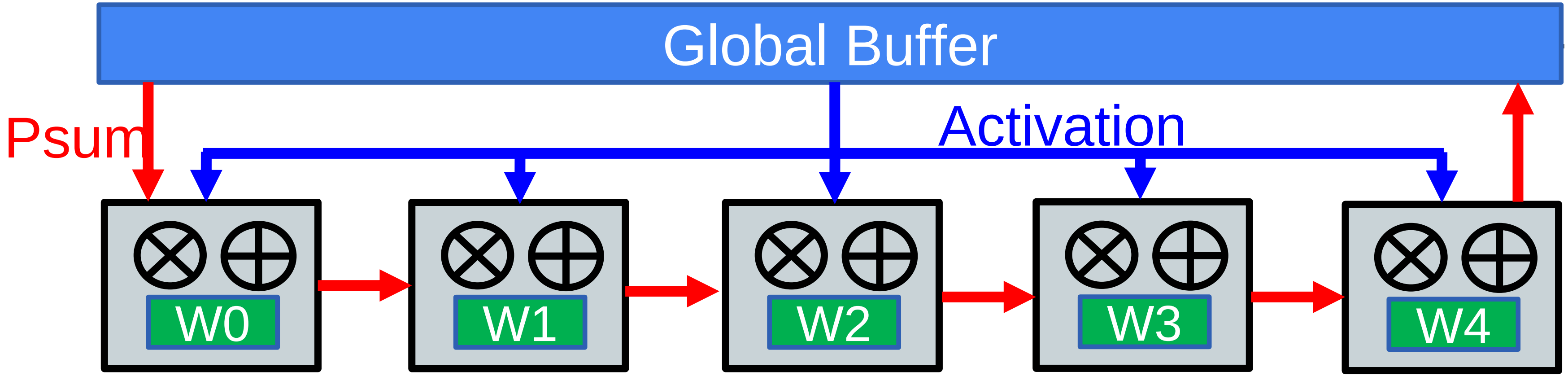
How to leverage local memory to reduce the times of remote DRAM access on DNN workloads ?
Optimal case: reduce **2896 M** to **61 M** DRAM accesses on AlexNet

Dataflow Taxonomy

- Output Stationary (OS)
- Weight Stationary (WS)
- Input Stationary (IS)
- Dataflow
  - Indicates the matrix which is “pinned” to a given PE
  - The **ordering** of the operations
  - Data prioritization across the memory hierarchy and compute data paths

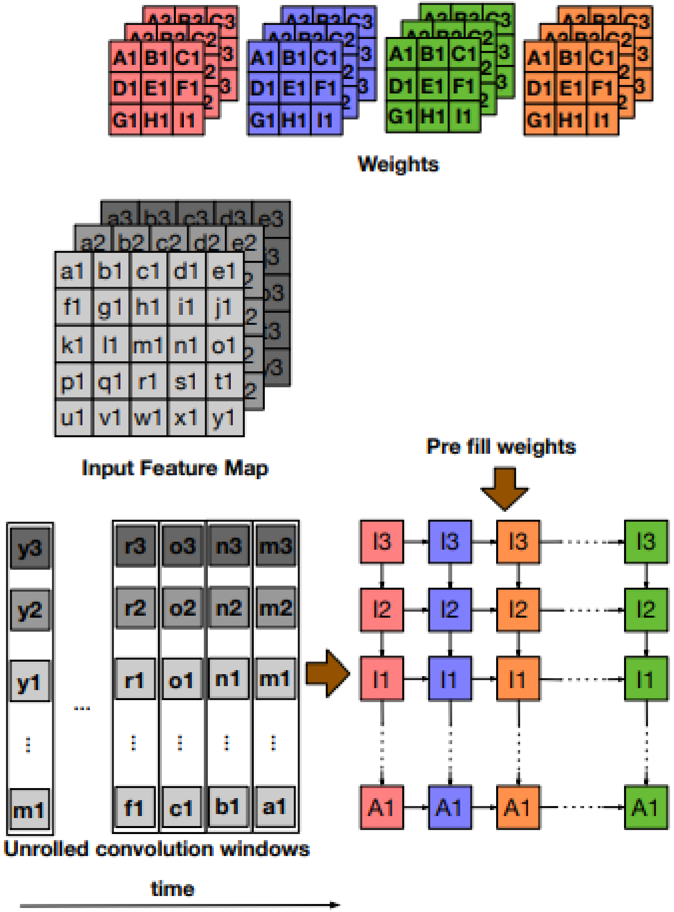
Weight Stationary (WS)

- Minimize weight read energy consumption
- Broadcast activations and accumulate psums spatially across PEs
- Each weight stays stationary in RF of each PE



Weight Stationary (WS)

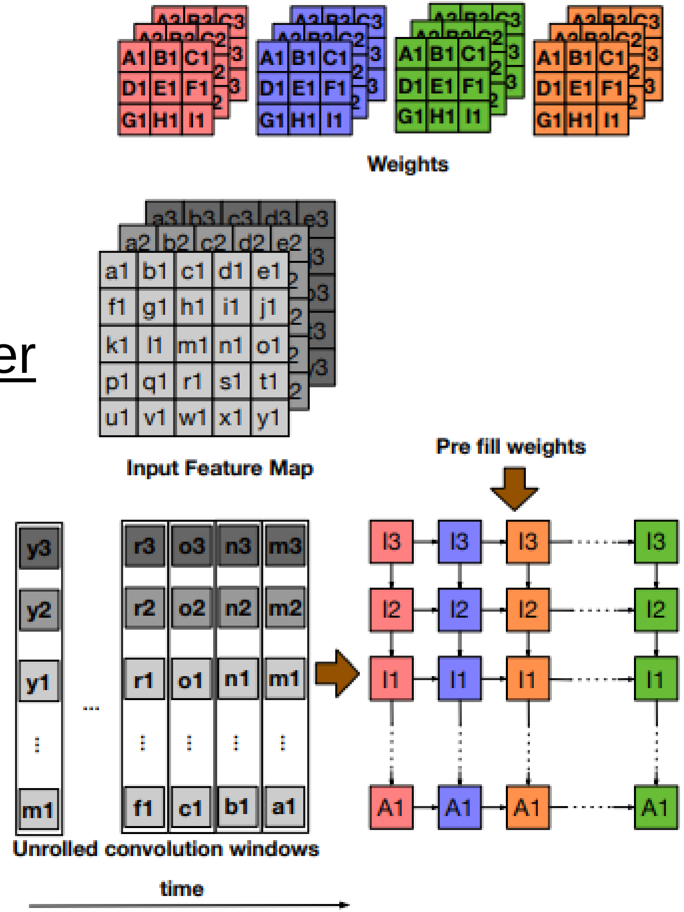
- Each element of the **weight matrix** is uniquely mapped to a given MAC unit
- Every cycle the **input elements** are multiplied with the currently mapped weights
- **Partial sums** are stored within the array
- **Reduction** takes place by communicating the partial sums across the MAC units
  - Take multiple cycles



https://arxiv.org/pdf/1811.02883

Weight Stationary (WS)

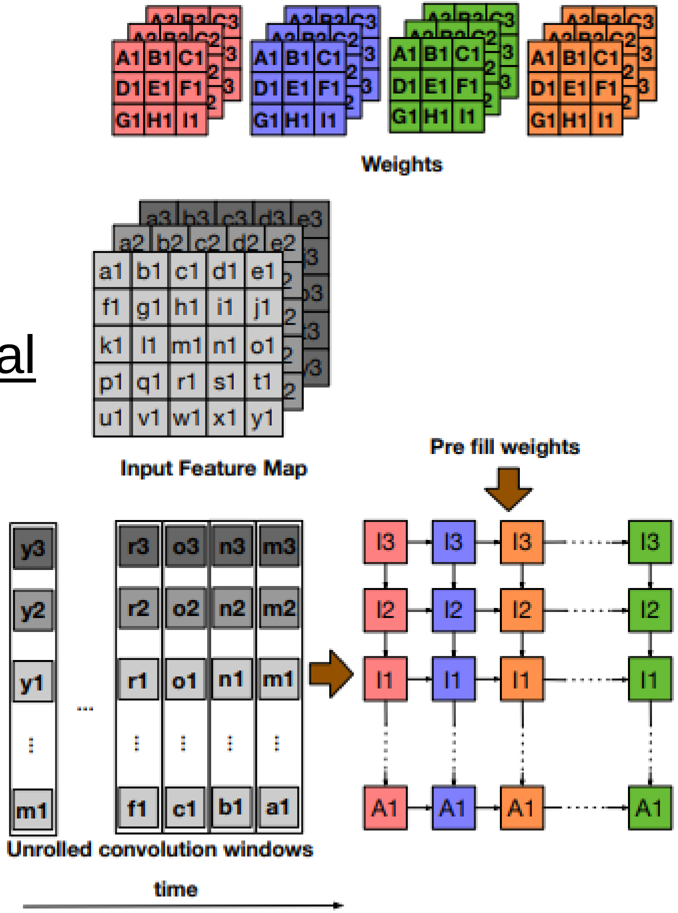
- First step in WS data mapping
  - <u>Each column</u> is assigned to <u>a given filter</u>
  - The elements of the assigned <u>filter matrix are fed in from the top edge</u>
  - After the filter elements are placed, the <u>pixels of inputs are then fed in from the left edge</u>
  - Partial sums for a given output is generated every cycle



https://arxiv.org/pdf/1811.02883

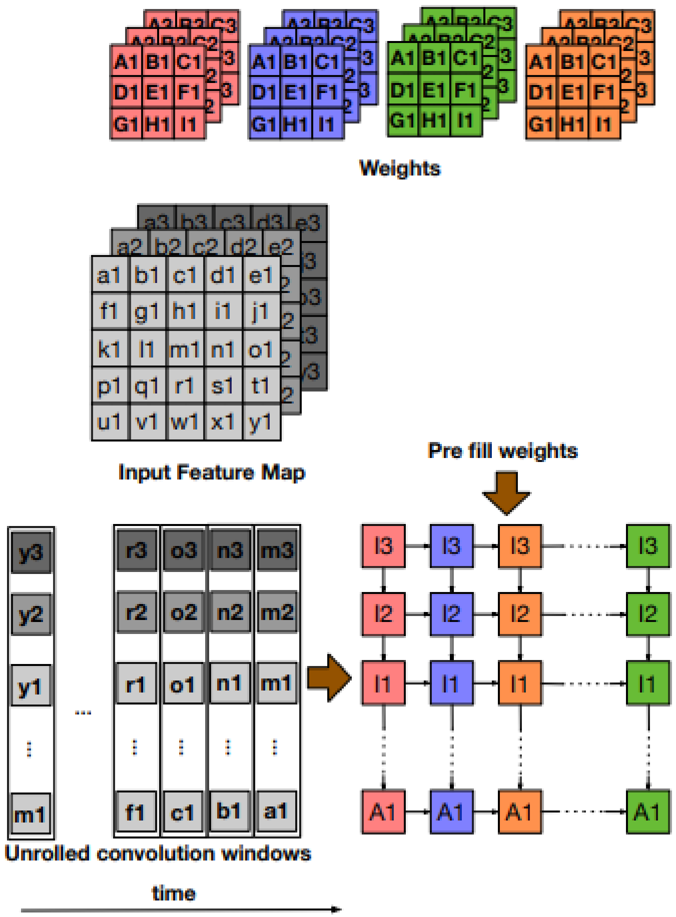
Weight Stationary (WS)

- Second step in WS data mapping
  - For a given output, corresponding <u>partial sums are distributed over a column</u>
  - Partial sums are <u>reduced over the given column in next n cycles</u>
  - n is the number of partial sums generated for a given pixel
  - Once the mapped weight are done, the mapping is replaced with new set of weights



Weight Stationary (WS)

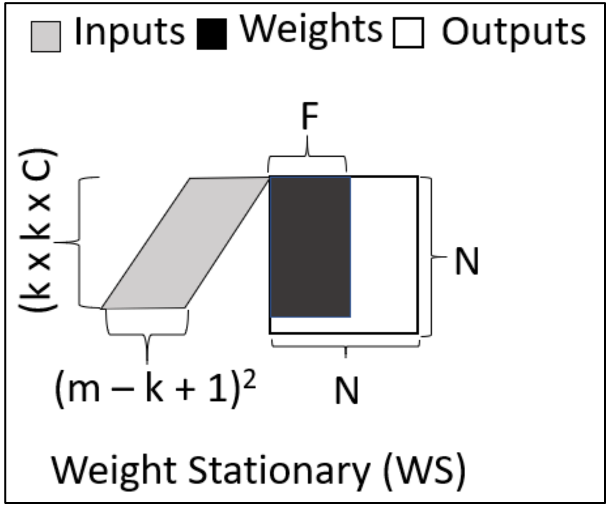
- Shortcoming of the WS
  - Partial sums corresponding multiple outputs are required to be kept in the array until the are reduced
  - Leads to increase in implementation cost (why?)



https://arxiv.org/pdf/1811.02883

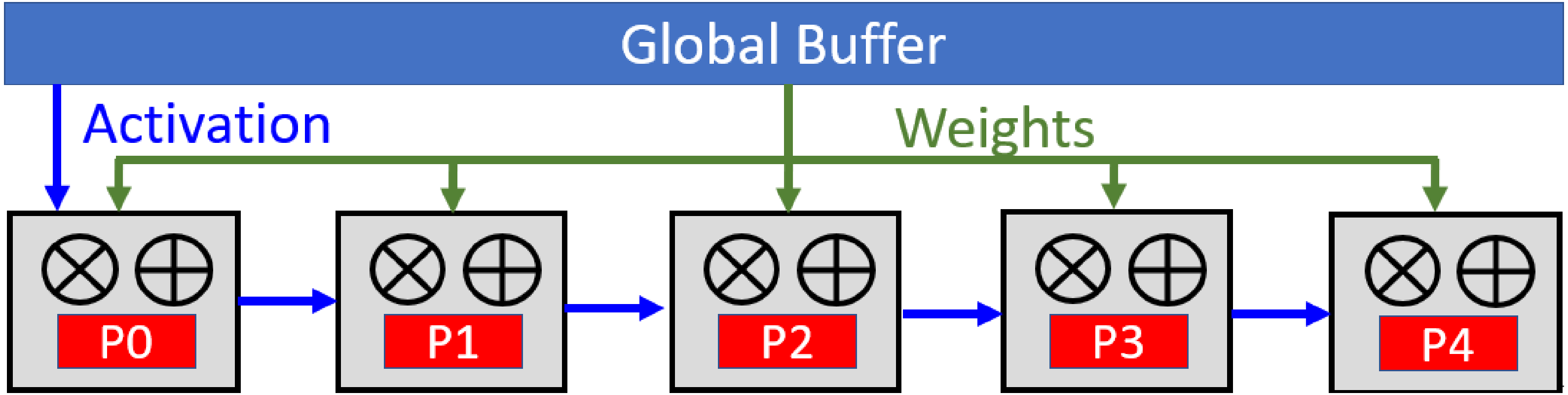
Latency Analysis of Weight Stationary

- **The weight stationary in the systolic array**
  - Inputs take $(m - k + 1)^2 + (k \times k \times C - 1)$ cycles to flow in the spatial array horizontally
  - Inputs also need to take F cycles to pass through each filter
  - Pre-load weights take $(k \times k \times C)$ cycles
  - Total cycles
    - $(m - k + 1)^2 + (k \times k \times C - 1) + (k \times k \times C) + F$



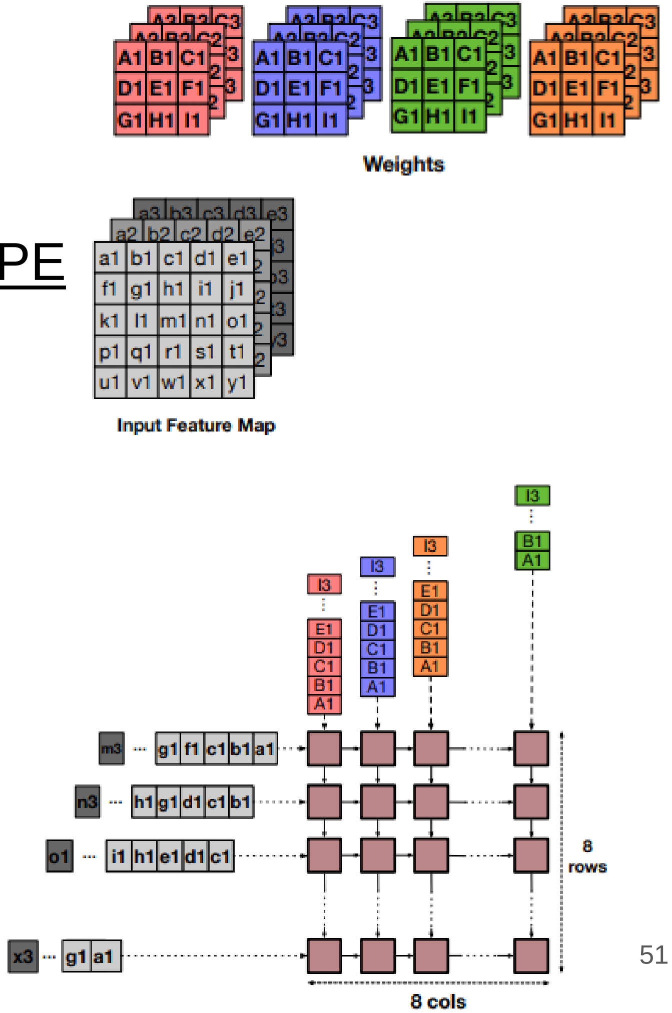
Output Stationary (OS)

- Minimize partial sum R/W energy consumption
- Keep the accumulation of psums stationary in the RF
- Stream input activations across PE array
- Broadcast the weights to all PE array from the global buffer



Output Stationary (OS)

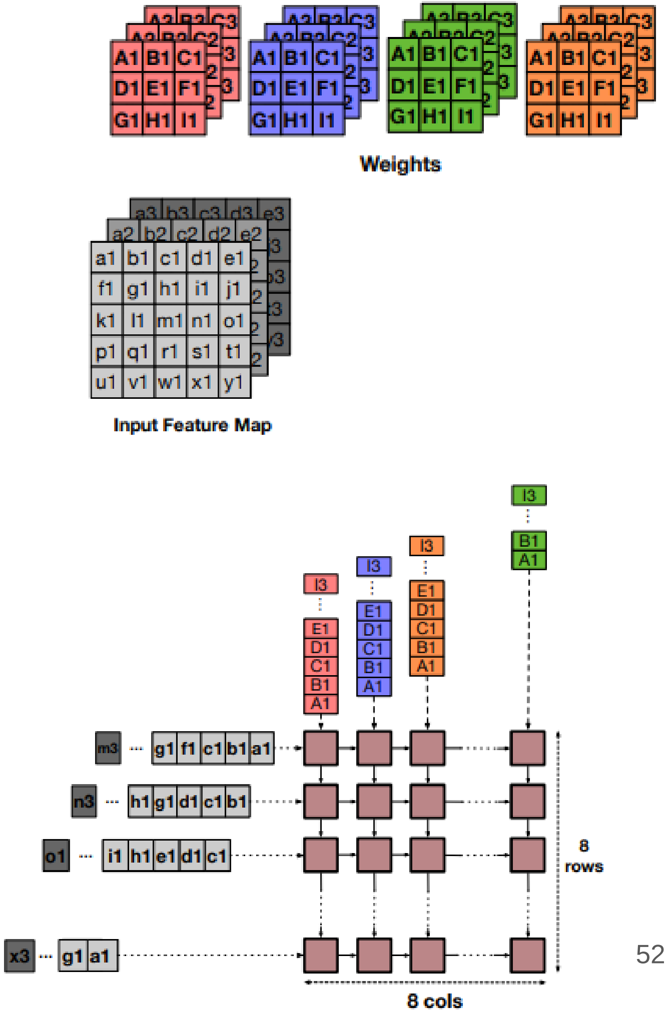
- Each pixel of output is assigned to a given PE
- All compute necessary for generating the given output is done on the PE
- The input and weight are streamed in every cycle
- Reduction operation is done in place, no further communication is needed
- Once one output pixel is generated by a given PE, the result is transferred to the memory, and the PE is assigned another pixel to compute



https://arxiv.org/pdf/1811.02883

Output Stationary (OS)

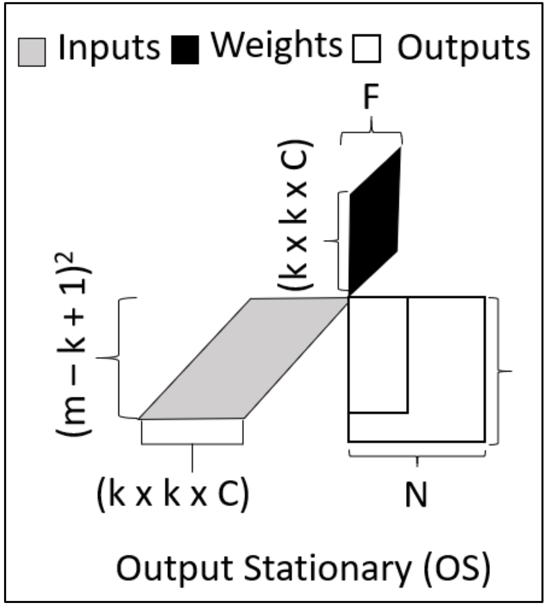
- In a given column PEs in each row
  - Generating adjacent output in a single channel
  - Each column generates pixels corresponding to different output channels
- Shortcoming of the OS
  - The data transferred overhead of generated outputs



https://arxiv.org/pdf/1811.02883

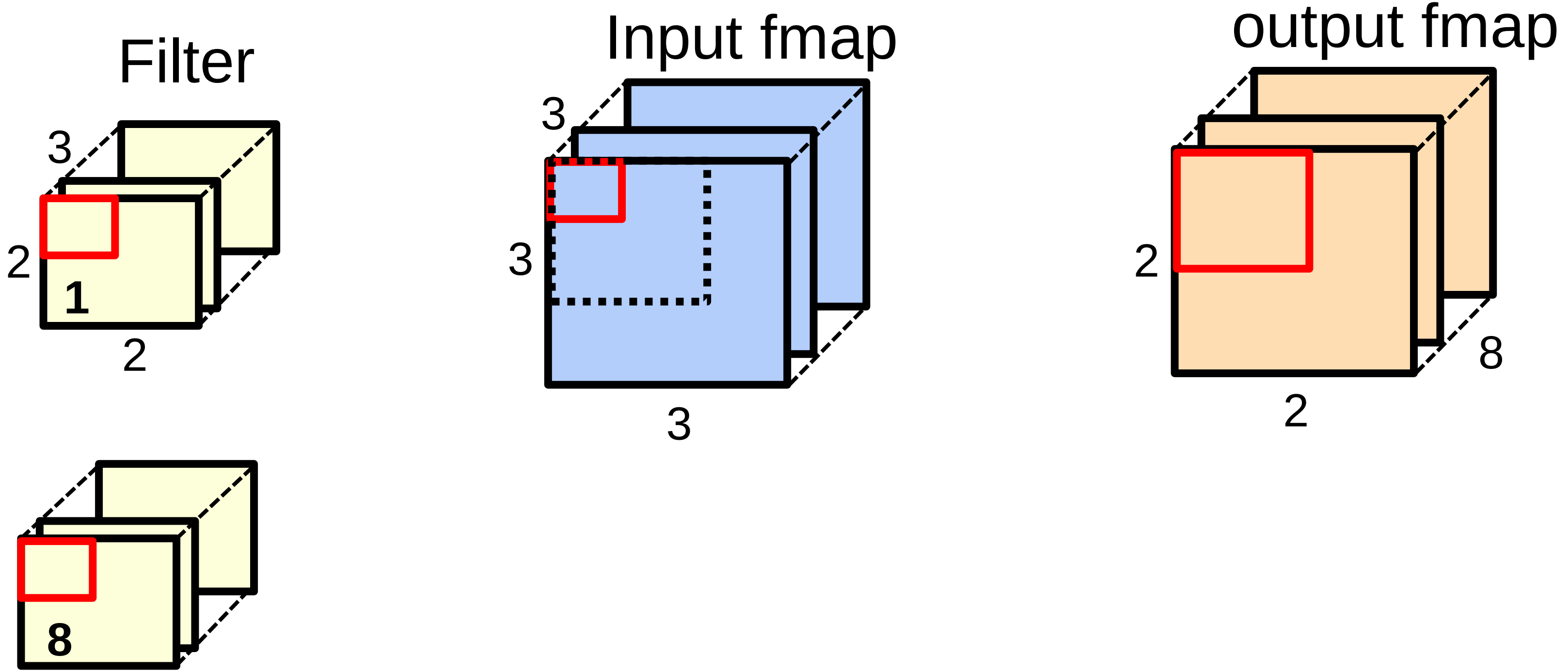
Latency Analysis of Output Stationary

- **The output stationary in the systolic array**
  - Inputs and weights are pushed in the systolic array and takes $(k \times k \times C - 1) + (m - k + 1)^2$
  - Taking F cycles to pass through outputs
  - Outputs are accumulated in-place
  - Total cycles
    - $(k \times k \times C - 1) + (m - k + 1)^2 + F$



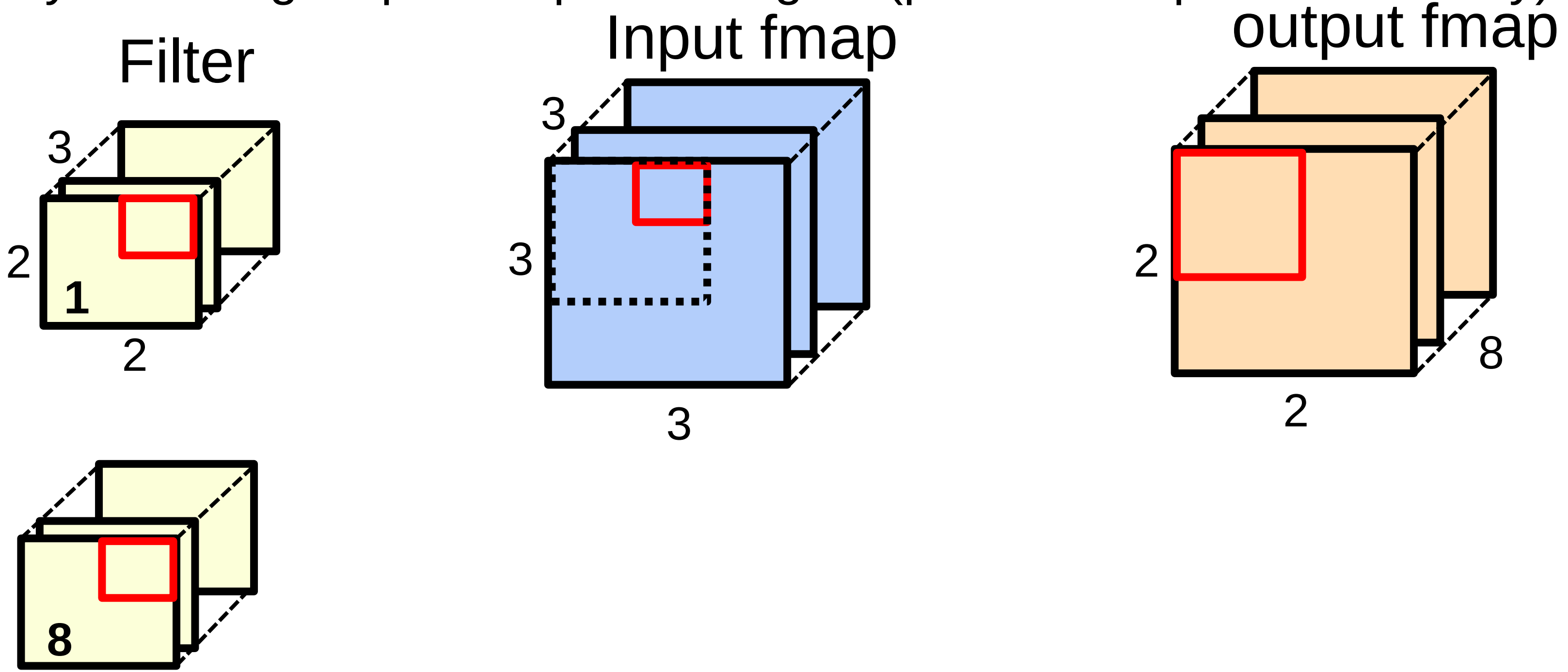
OS Dataflow Example

- Cycle through input fmap and weights (psum of output is stationary)



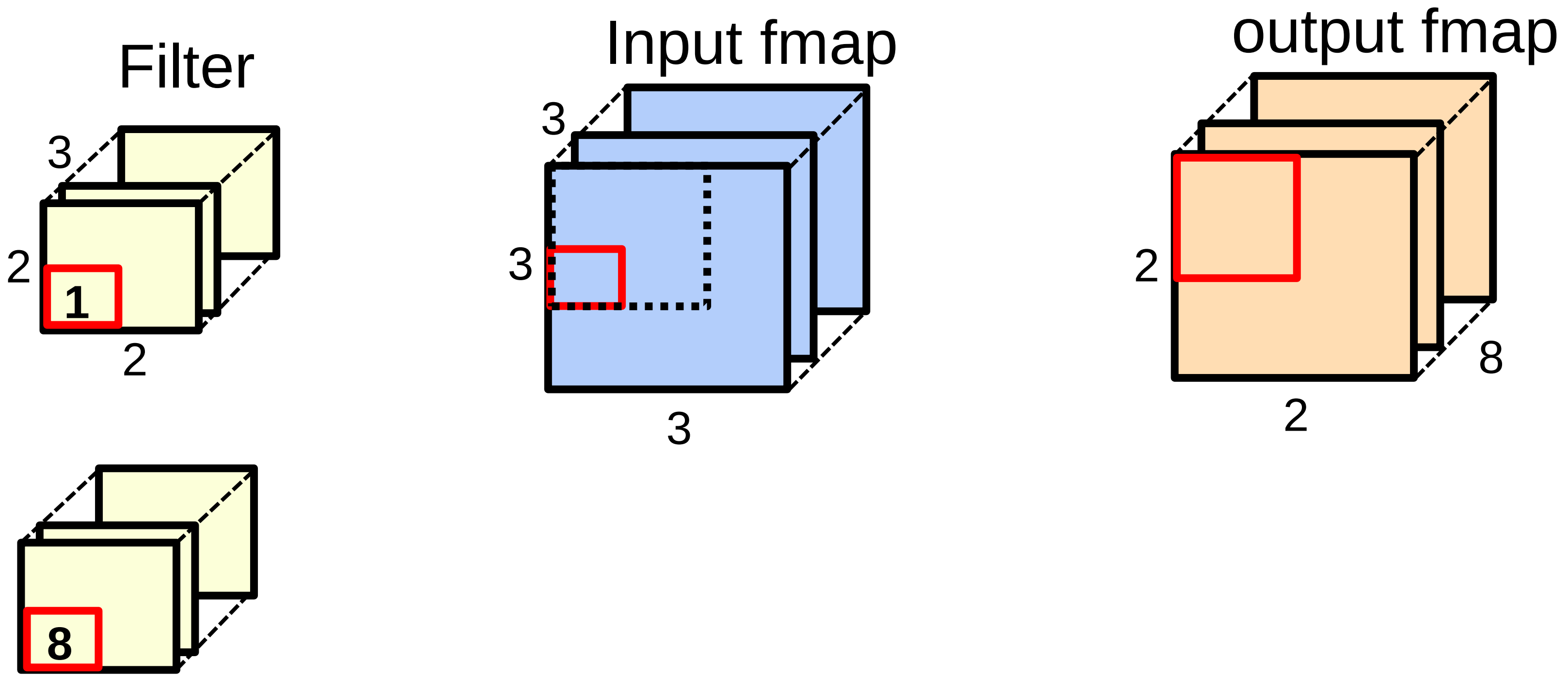
OS Dataflow Example

- Cycle through input fmap and weights (psum of output is stationary)



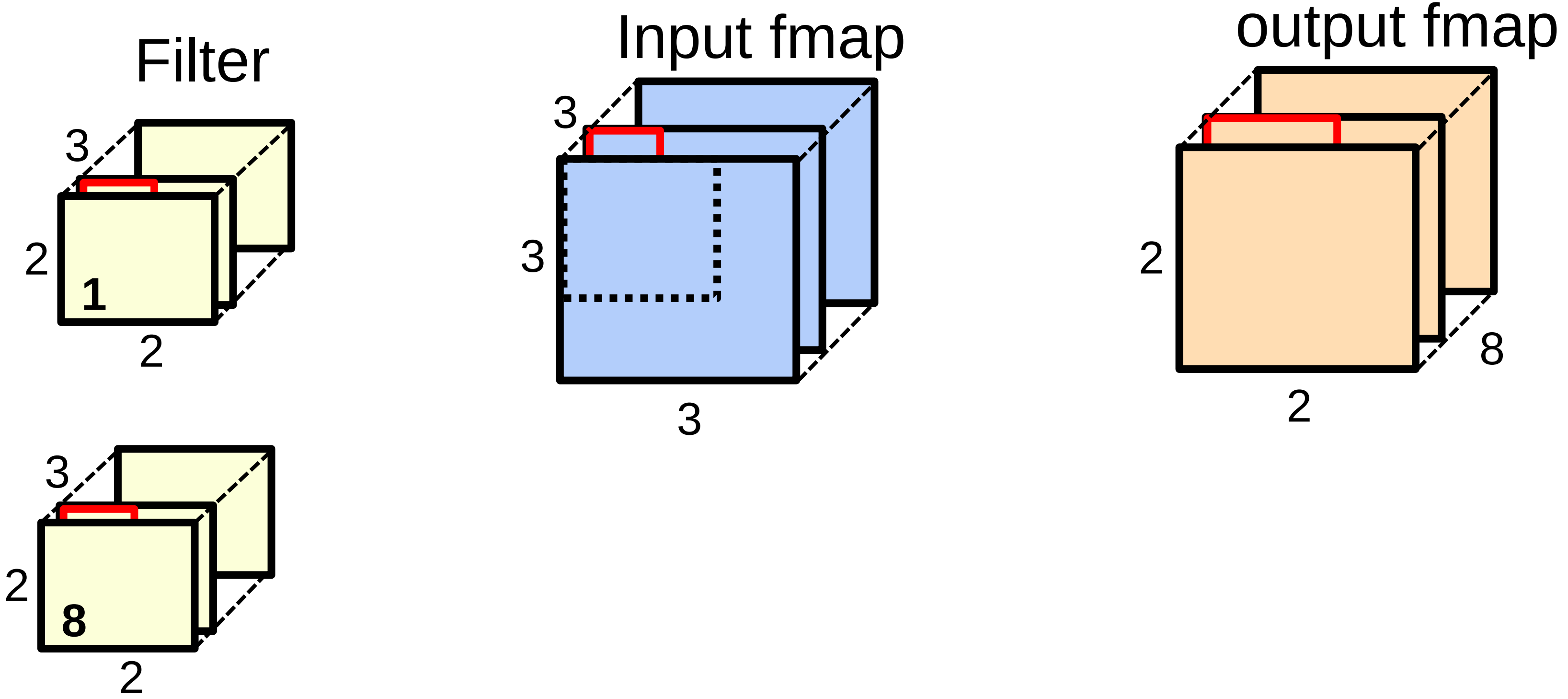
OS Dataflow Example

- Cycle through input fmap and weights (psum of output is stationary)



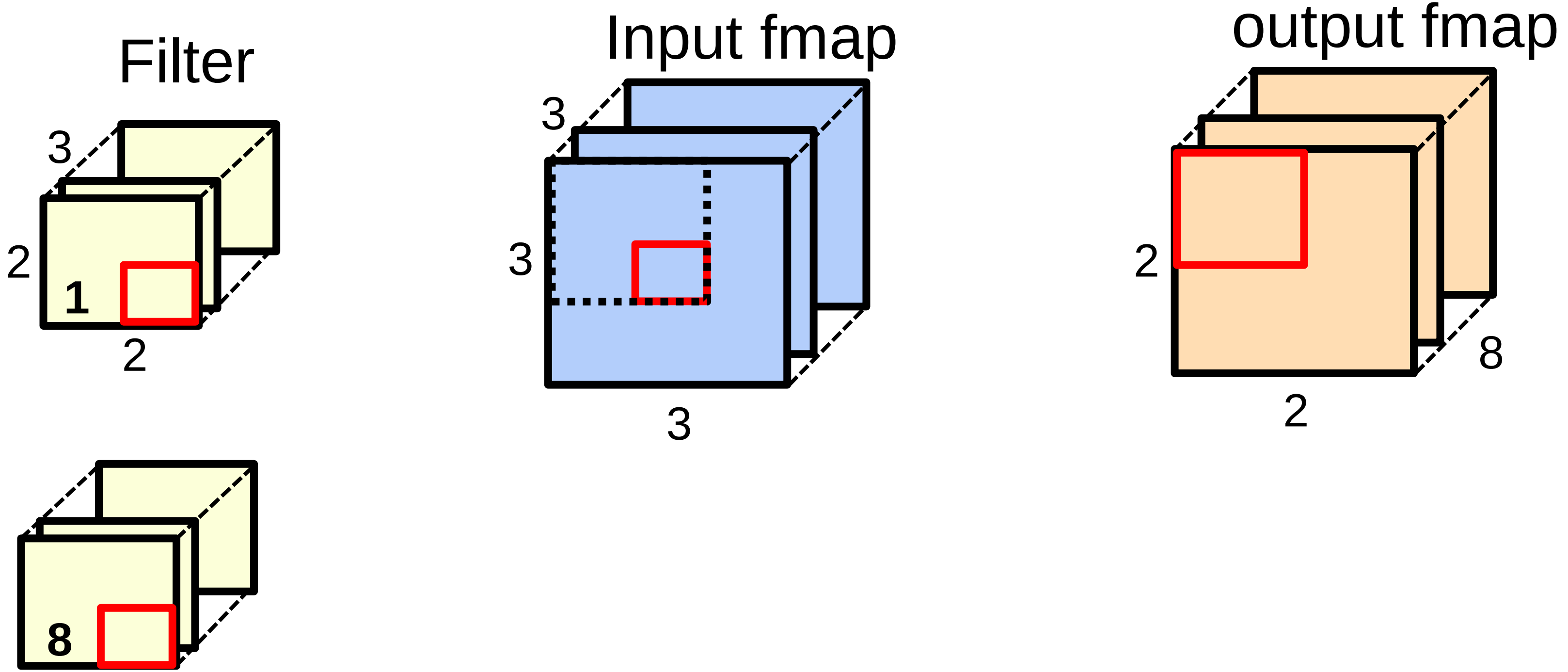
OS Dataflow Example

- Cycle through input fmap and weights (psum of output is stationary)



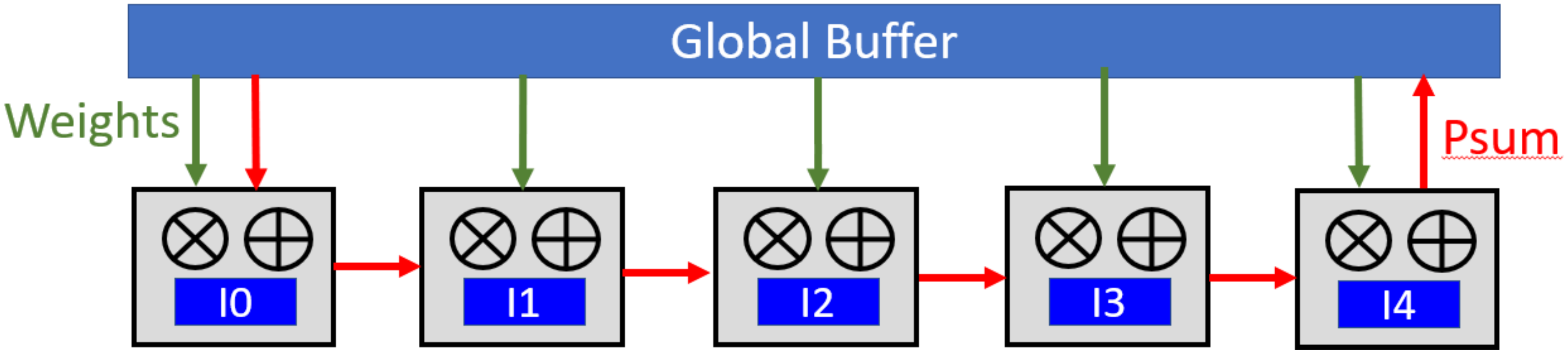
OS Dataflow Example

- Cycle through input fmap and weights (psum of output is stationary)



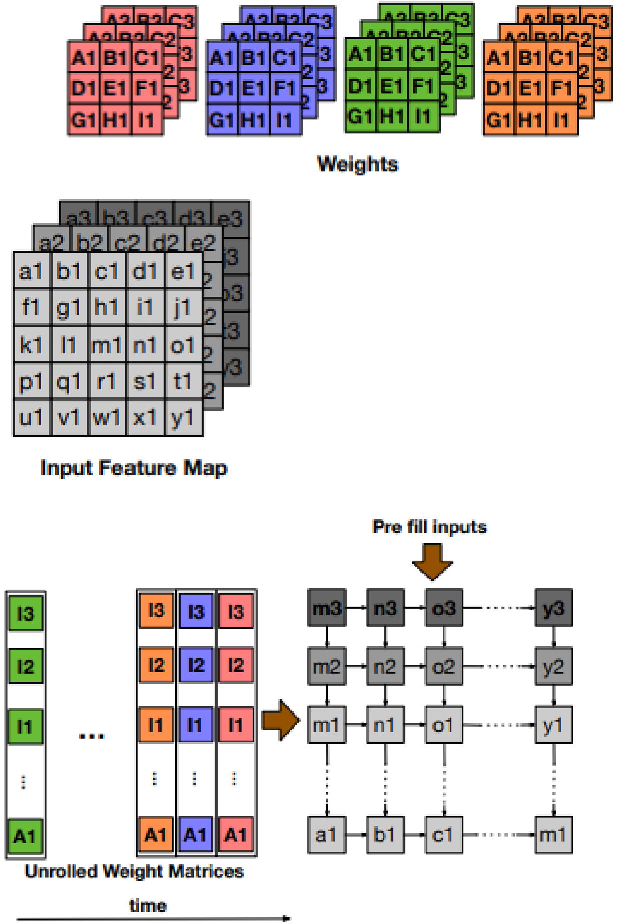
Input Stationary (IS)

- Minimize the energy consumption of reading input activations
- Unique filter weights are uni-cast into PEs at each cycle
- Psums are spatially accumulated across PEs



Input Stationary (IS)

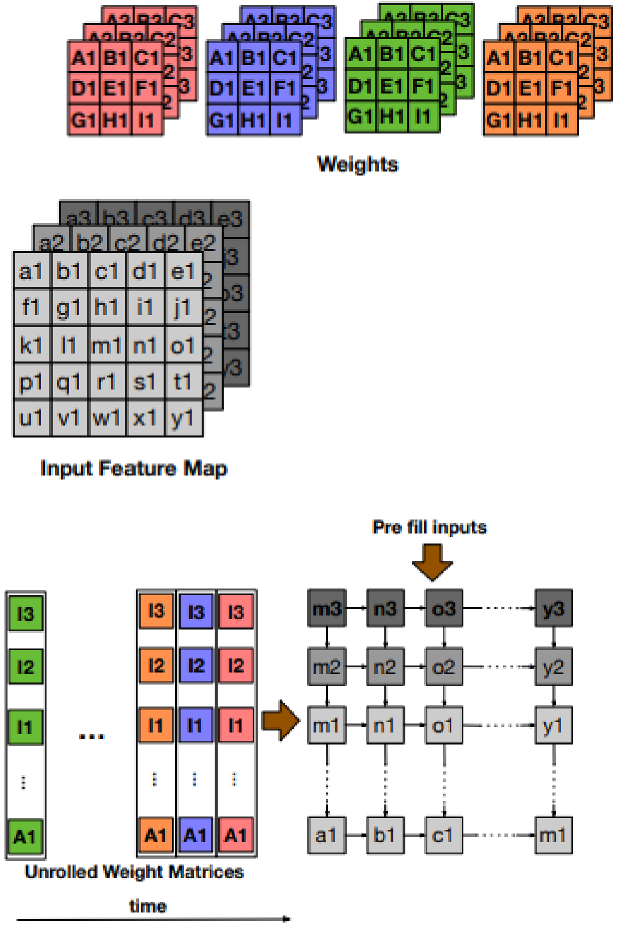
- Input feature map (IFMAP) are "pinned" with the PEs
- The elements of the weight matrices are streamed in
- Each column is assigned to a convolution window
- The convolution window is a set of all the pixels in the IFMAP which are required to generated a single OFMAP



https://arxiv.org/pdf/1811.02883

Input Stationary (IS)

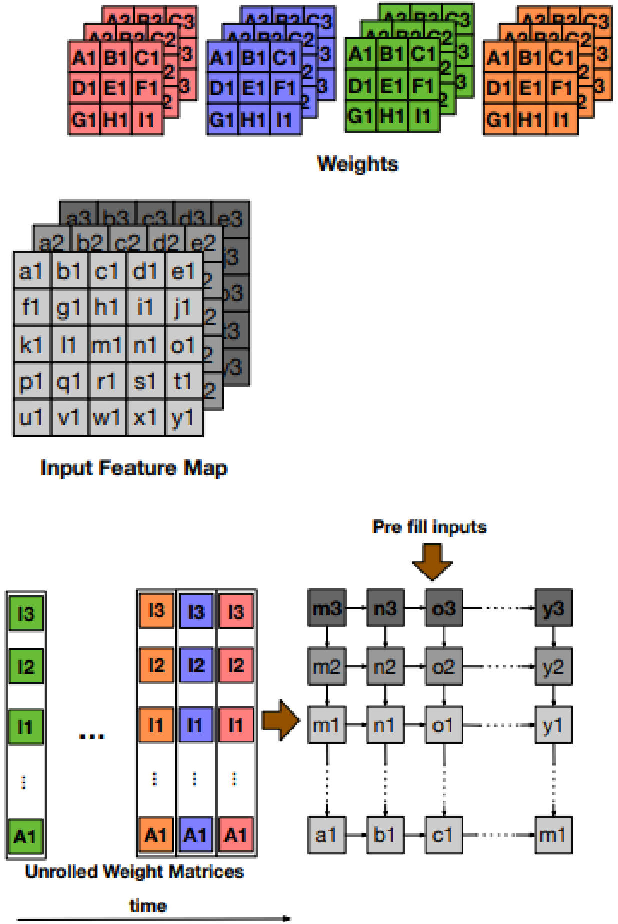
- Once the inputs are fed in, the elements of the weight matrices are streamed in from the left edge
- The reduction is performed over a given column
- The convolution windows are kept around until all the computations requiring these elements are done



https://arxiv.org/pdf/1811.02883

Input Stationary (IS)

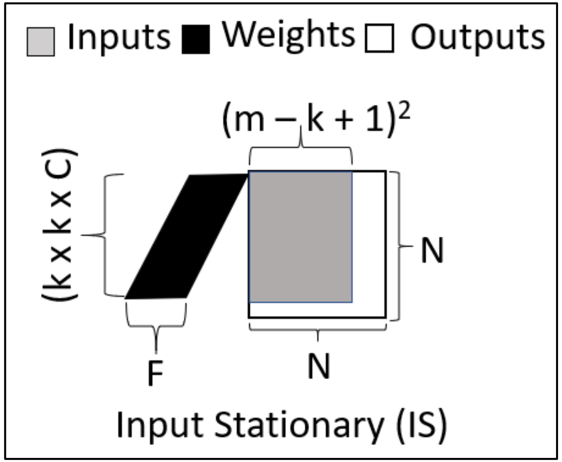
- Benefits
  - Lower SRAM bank requirements as compared to OS
- Shortcoming
  - The cost and runtime compared to WS varies by workload



https://arxiv.org/pdf/1811.02883

Latency Analysis of Input Stationary

- **The input stationary in the systolic array**
  - Weights stream into the systolic array horizontally and takes (k x k x C - 1) + F cycles
  - Weights also take $(m – k + 1)^2$ cycles to pass through entire inputs
  - Pre-load inputs takes (k x k x C) cycles
  - Total cycles
    - $(k \times k \times C) + (k \times k \times C - 1) + F + (m – k + 1)^2$



Parameters of CNN Network

Parameters	
m	The width and height of input feature map
K	The width and height of filter
F	The number of filters
C	The number of channels
N	The width and height of spatial array

Dataflow Cost Analysis

R: size of filter weight
E: size of output activations

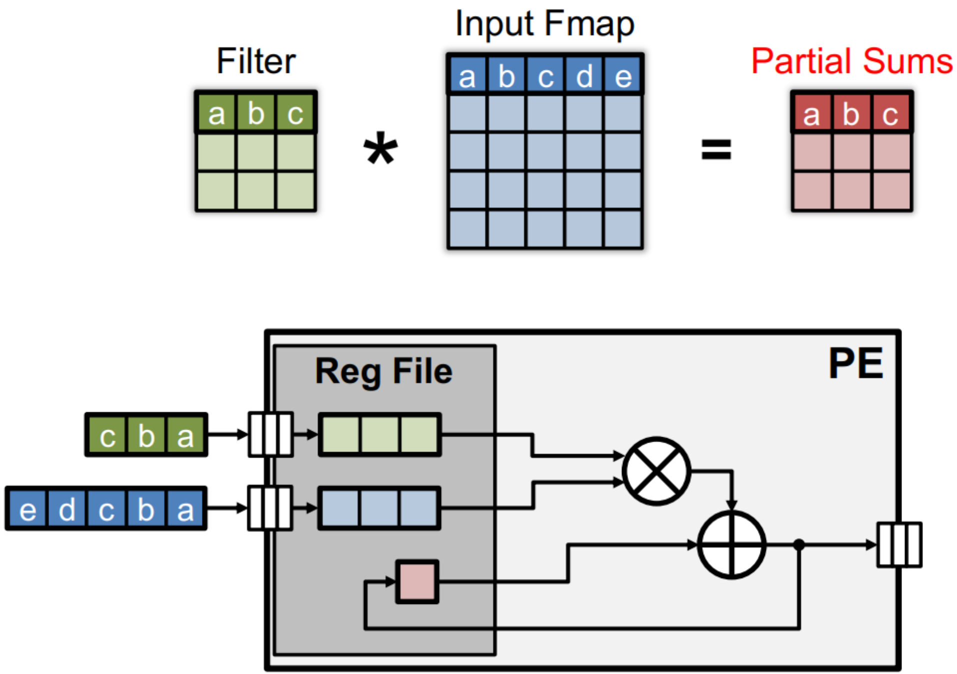
- OS minimizes output reads (0)
- WS saves # of weight reads (E)
- IS saves # of input reads (E)

These dataflows only reduce a specific reads. Could we do better ?

	OS	WS	IS
MACs	E*R	E*R	E*R
Weight Reads	E*R	R	E*R
Input Reads	E*R	E*R	E
Output Reads	0	E*R	E*R
Output Writes	E	E*R	E*R

Row Stationary (RS)

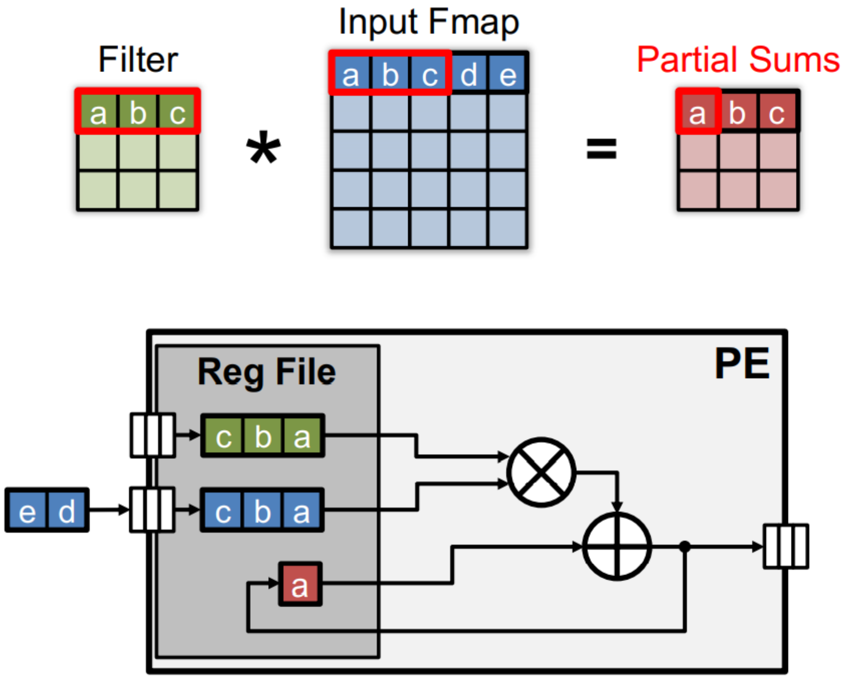
- Minimize data reuse at
- Optimize for overall data type energy efficiency



Chen et al., ISCA 2017

How does RS work ?

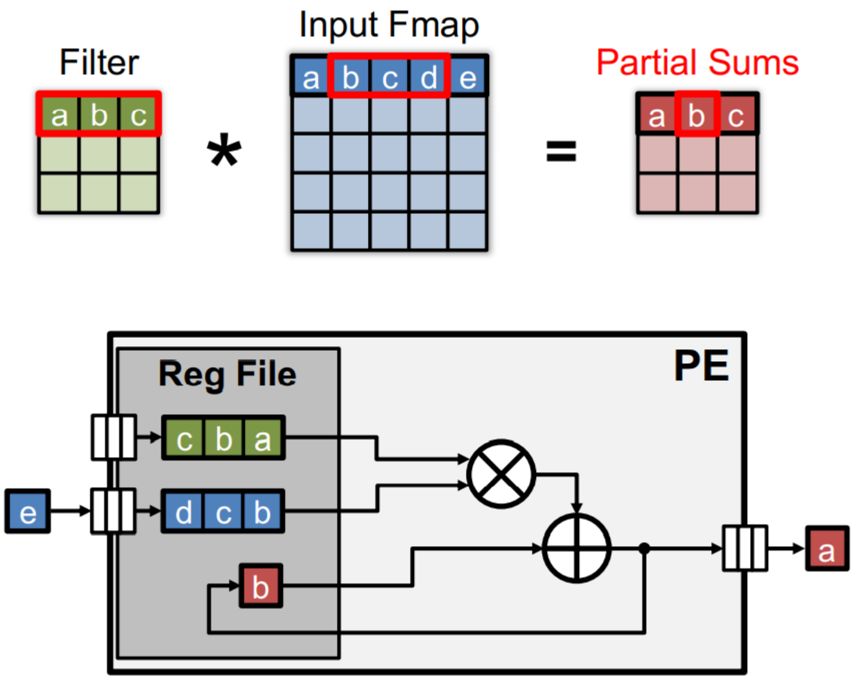
- Keep the row of filter weights stationary in RF of a PE
- PE does MACs for each sliding window of ifmap at a time
- Use only one memory space to accumulate Psums
- Overlap ifmap between different sliding windows -> reuse ifmap



Chen et al., ISCA 2017

How does RS work ?

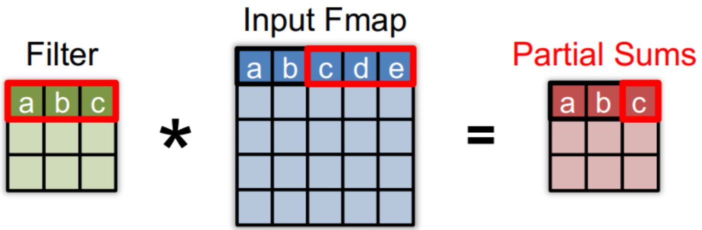
- Ifmap sliding window right shifts
- Pop the value “a” out of RF
- Accumulate Psum “b”

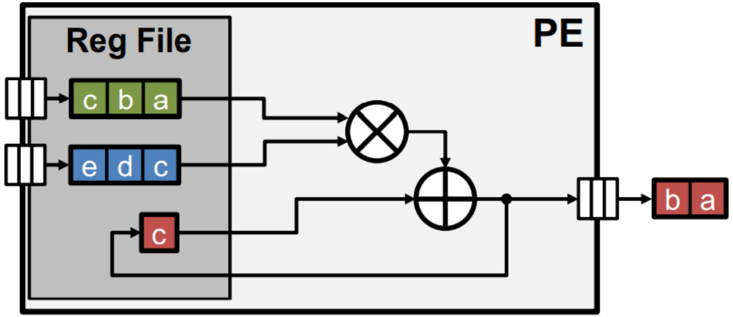


Chen et al., ISCA 2017

How does RS work ?

- Ifmap sliding window continues to right shift
- Pop out the value “b” in RF
- Accumulate psum “c”

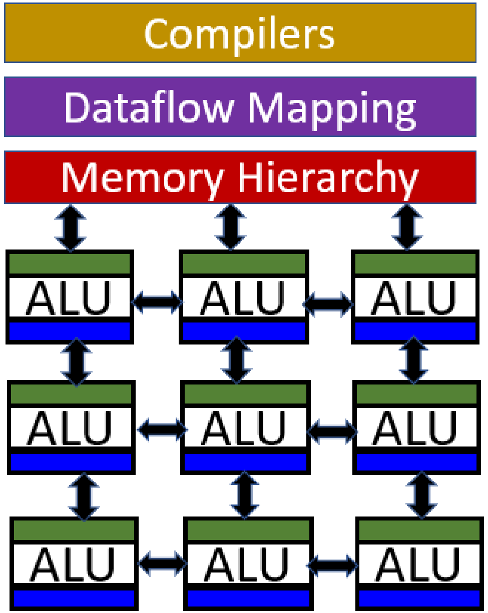




Chen et al., ISCA 2017

What do we learn from DNN Dataflow ?

- DNN layer shape and hardware resources provided determine the energy efficiency of dataflow mapping
- How can the fixed-size PE array accommodate different layer shapes?
- Known DNN layer shapes offline, could compiler/runtime system guide the mapping ?



Takeaway Questions

- What are the purposes of dataflow used by DNN applications?
  - (A) Reduce the data movement across off-chip memory
  - (B) Improve the clock frequency of PE
  - (C) Decrease the energy consumption of spatial array accelerator
- What kind of dataflow implemented by the PE on the right-hand side?
  - (A) WS
  - (B) IS
  - (C) OS