Network on Chip (NoC)-based Deep Neural Network Design Framework

Kun-Chih (Jimmy) Chen 陳坤志

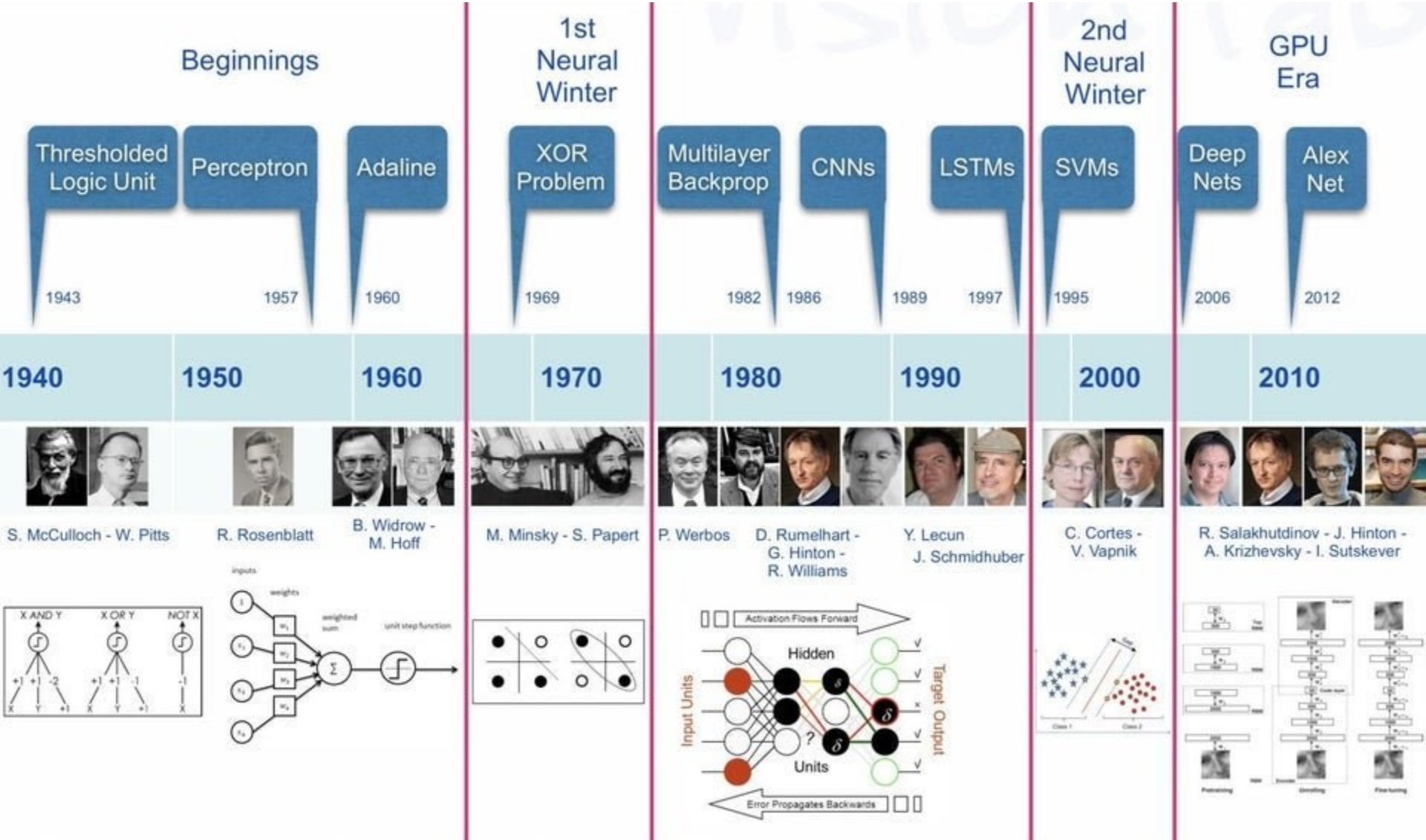
kcchen@nycu.edu.tw

Institute of Electronics,
National Yang Ming Chiao Tung University

Outline

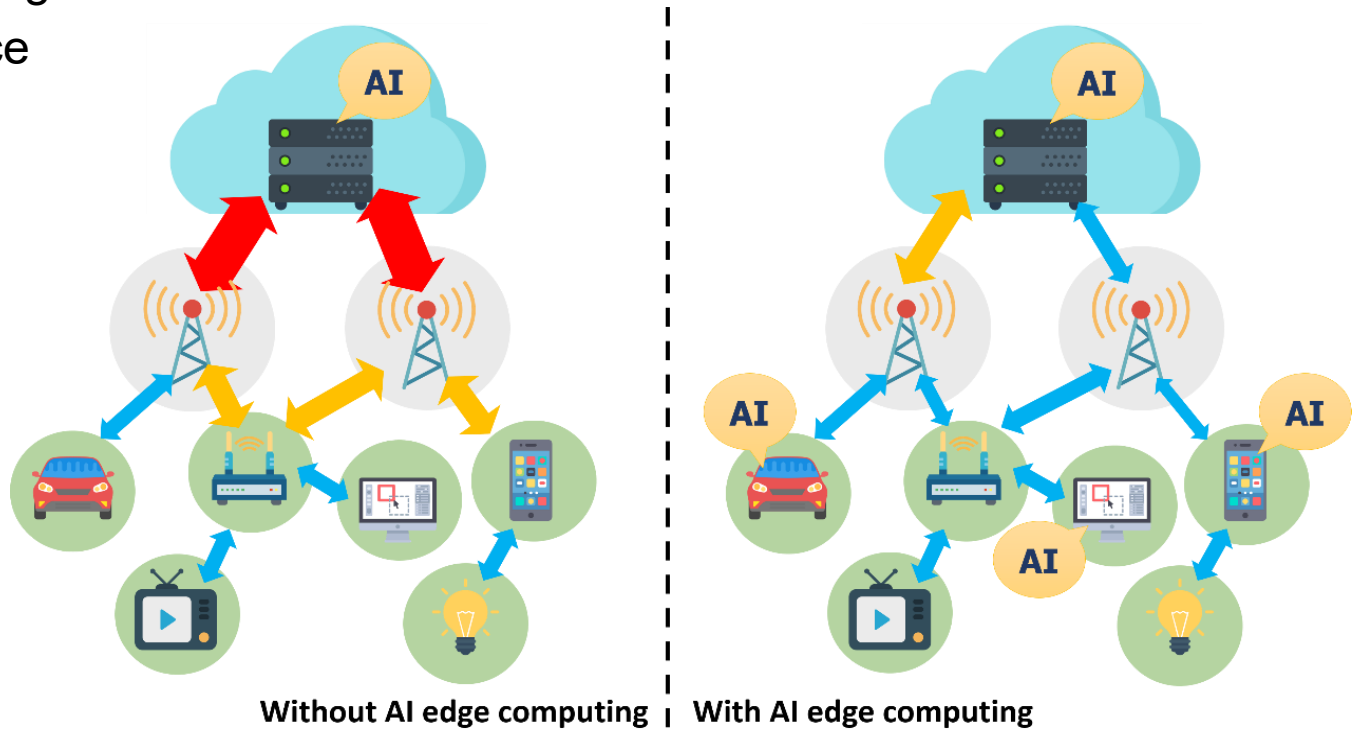
- Fundamental of DNN accelerator design: opportunities and challenges
- Fundamental of Network-on-Chip (NoC) interconnection
- NoC-based DNN design: algorithms and architectures

Timeline of AI History



From Cloud Server to Edge : On-device DNN

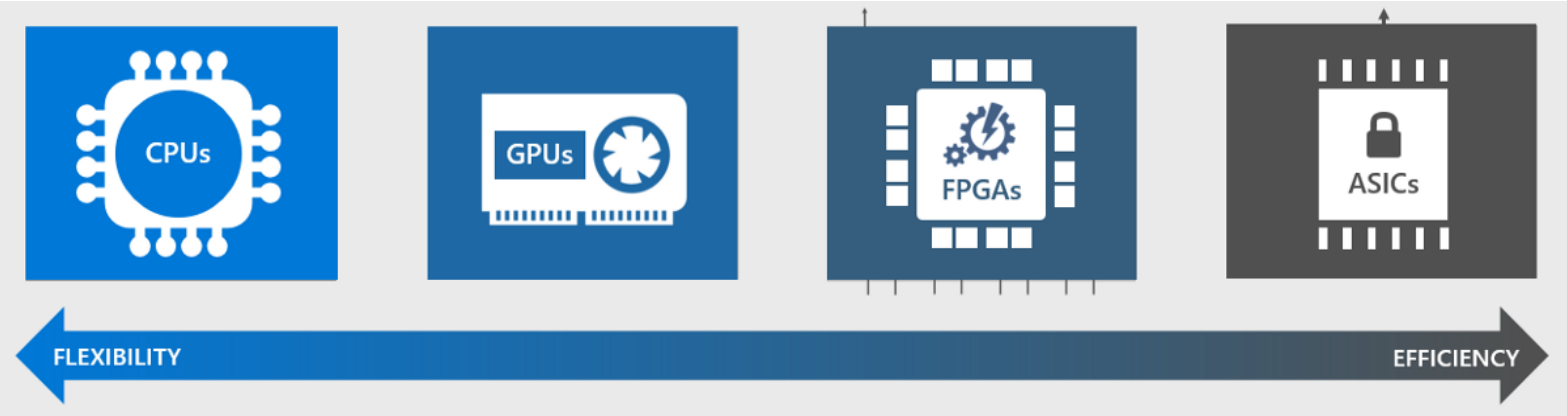
- The AI service move toward the local devices, such as IoTs [1]
  - Bandwidth
  - Privacy
  - Power Saving
  - Performance



[1] W. Shi *et al.*, “Edge computing: Vision and challenges”, *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Jun. 2016.

Architecture Selection of On-device DNN

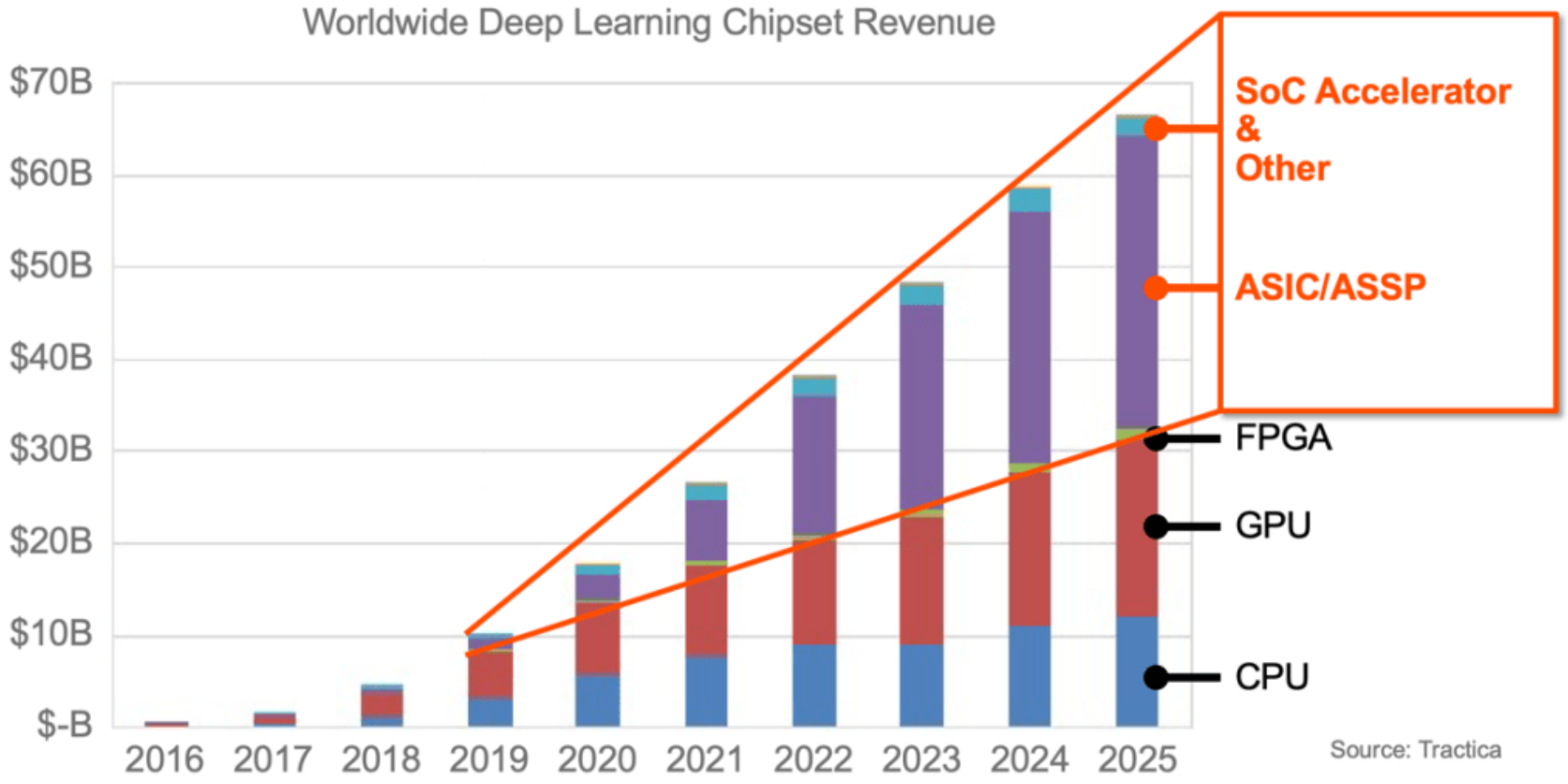
- Hardware efficiency and flexibility
  - GPUs : A massively parallel processing.
  - FPGAs : Reconfigurable interconnections of basic components.
  - ASICs : Customization and high performance.



➢ ***The Spectrum of four processor architecture.***

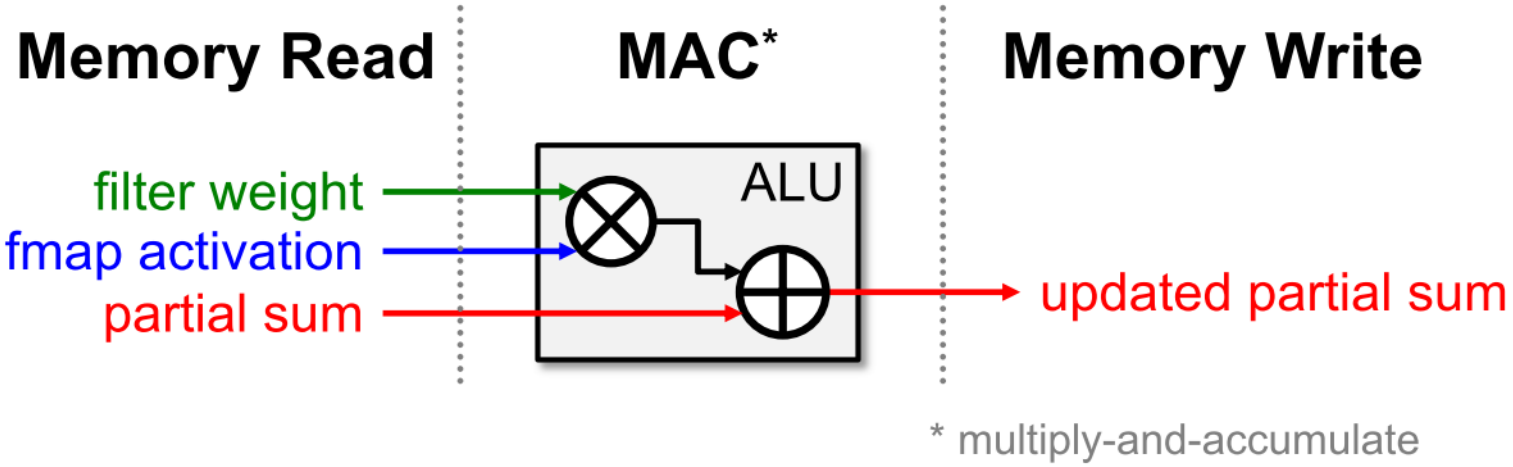
On-device DNN is the Future!!

- The AI service has moved toward the local devices, such as IoTs, vehicles *etc.*, which tends to be personal usage habits and experience.

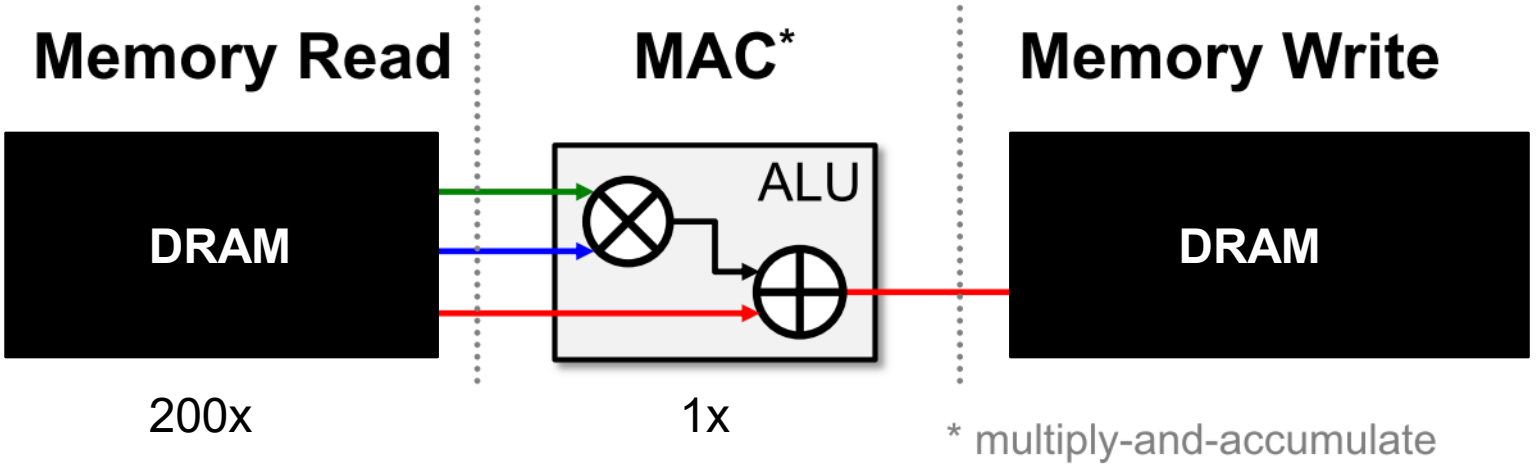


Fundamental of DNN accelerator design

Memory Access is the Bottleneck

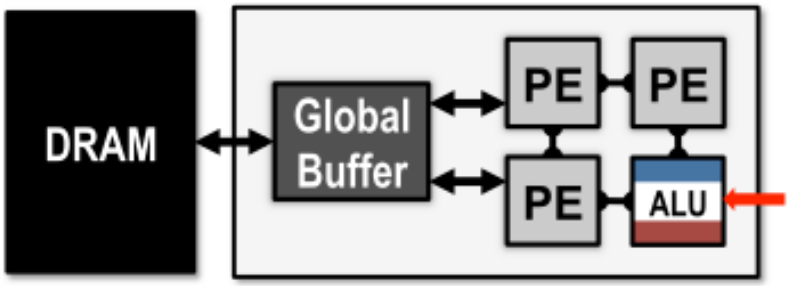


Memory Access is the Bottleneck



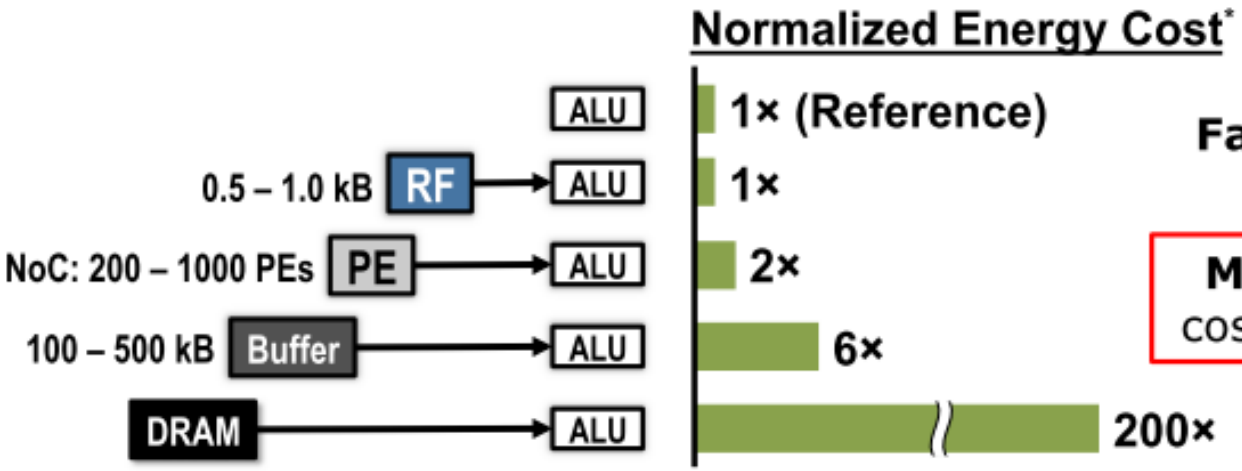
- Worst Case: all memory R/W are **DRAM accesses**
  - Example: AlexNet [NIPS 2012] has **724M** MACs
    → **2,896M** DRAM accesses required

Data Movement is Expensive



Specialized hardware with small (< 1kB) low cost memory near compute

fetch data to run a MAC here

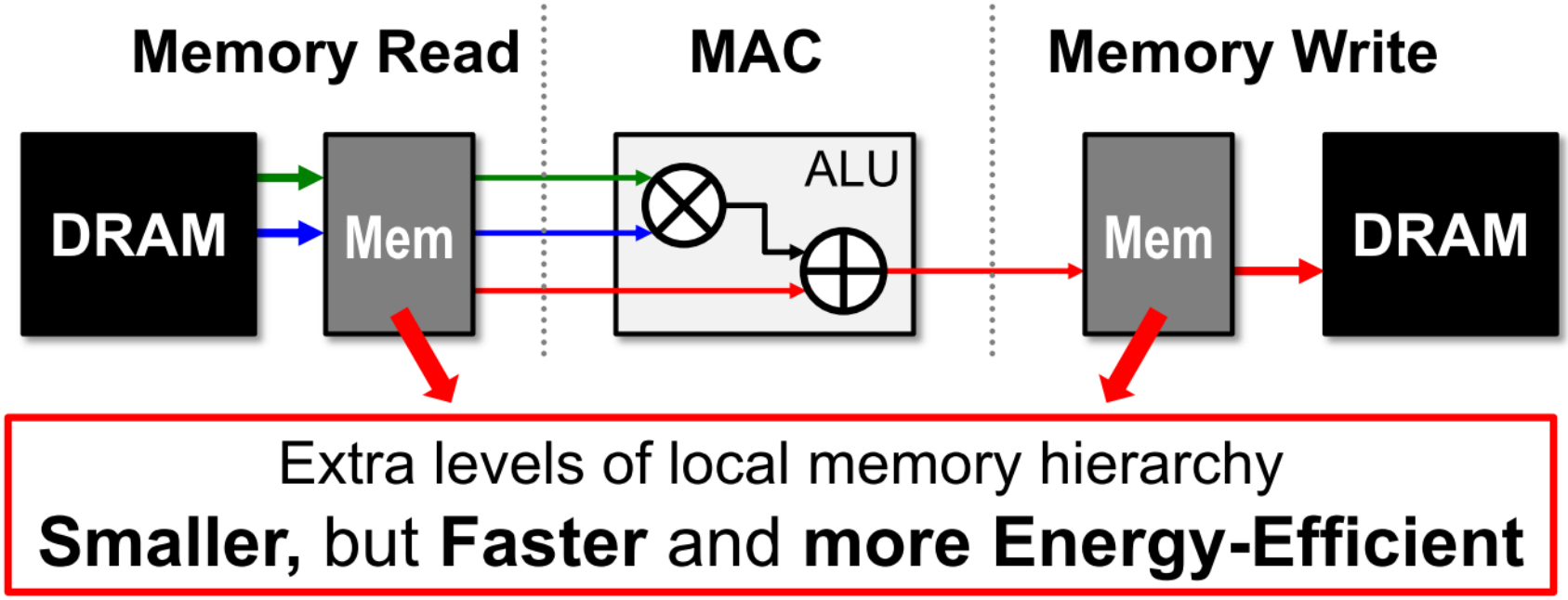


Farther and **larger** memories consume more power

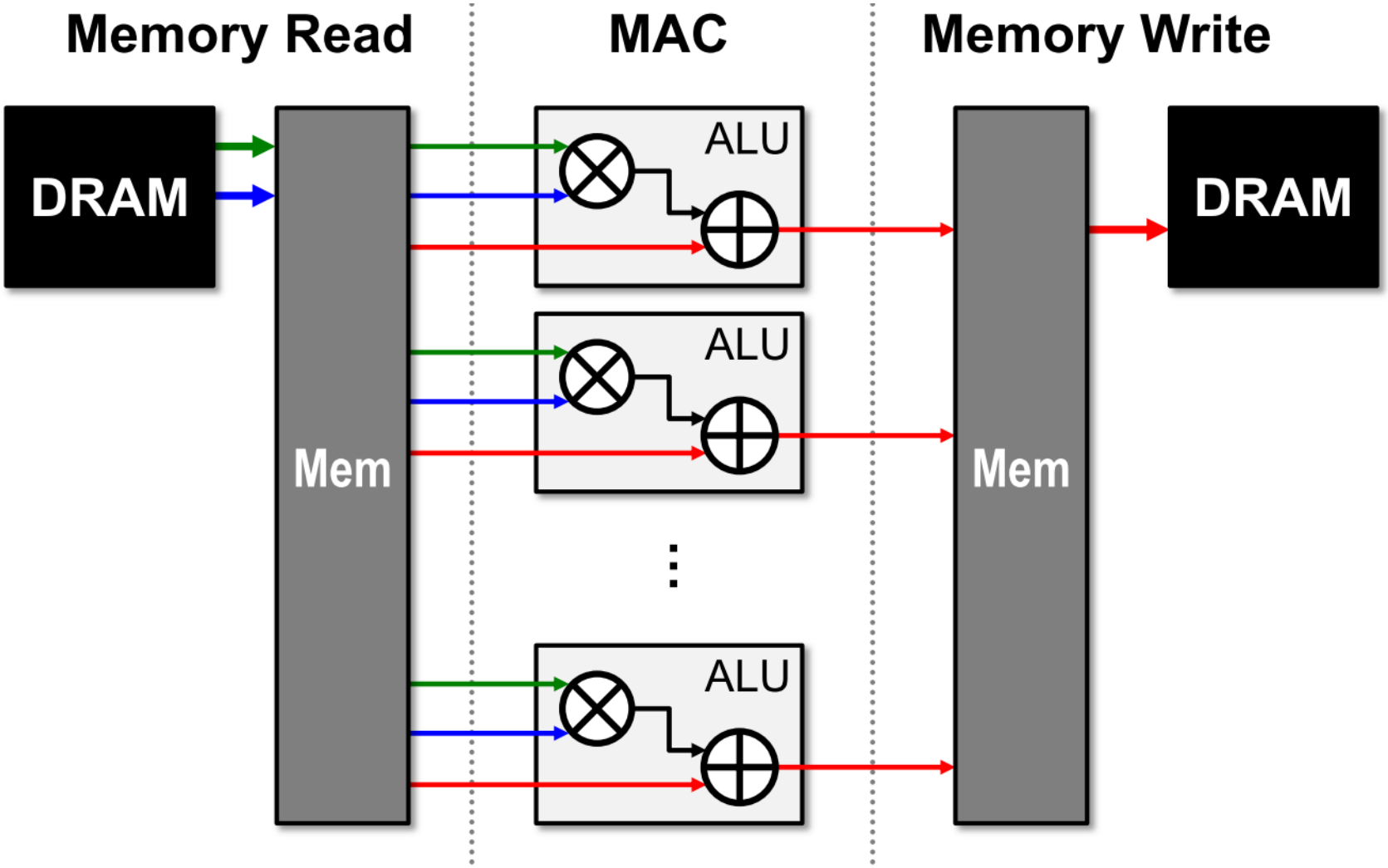
Maximize data reuse at low cost levels of memory hierarchy

* measured from a commercial 65nm process

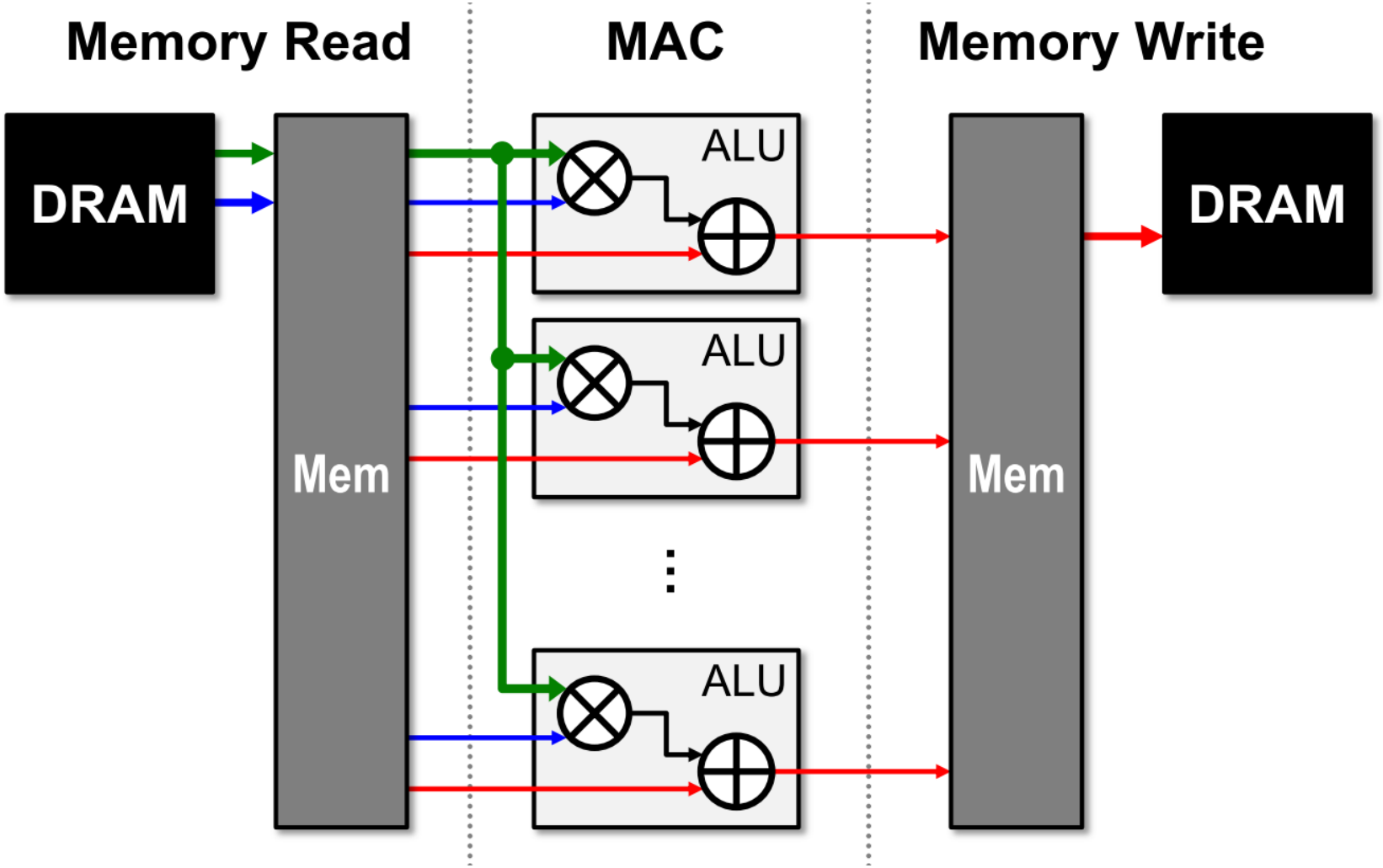
Leverage Local Memory for Data Reuse



Leverage Parallelism for Higher Performance



Leverage Parallelism for Spatial Data Reuse



Design Objectives Reduce Data Movement

- Reduce Data Movement
  - Reducing the number of times values are moved from source with high energy cost
    - DRAM, large on-chip buffer
  - Reducing the cost of moving each value
    - Reducing the data bit-width

- Increase PE utilization
  - Allocating work to as many PEs as possible
    - So as they can work in parallel
  - Minimizing the PEs idle cycles
    - Assuring enough bandwidth to deliver data to PEs

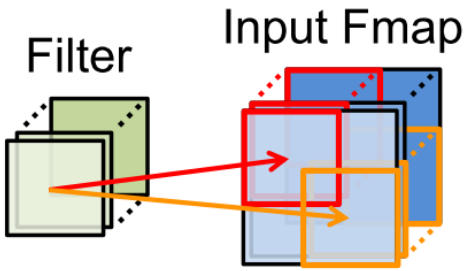
Data Movement Issue

- Exploiting ***data reuse***
  - The same piece of data is often reused for multiple MACs
- Three forms of data reuse
  - Input feature map (Fmap) reuse
  - Filter reuse
  - Convolutional reuse

Forms of Data Reuse in DNN

Convolutional Reuse

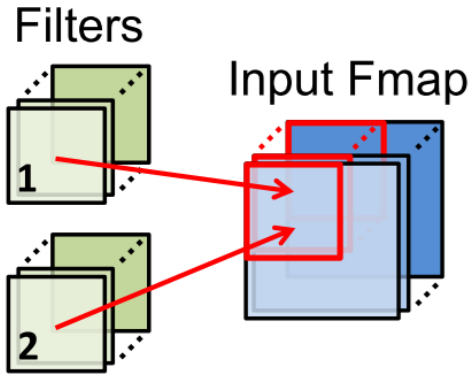
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

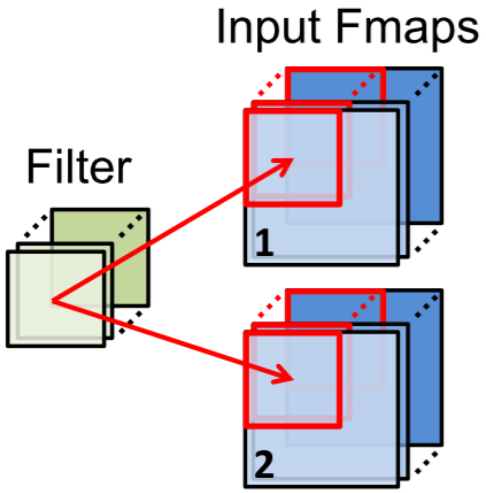
CONV and FC layers



Reuse: Activations

Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

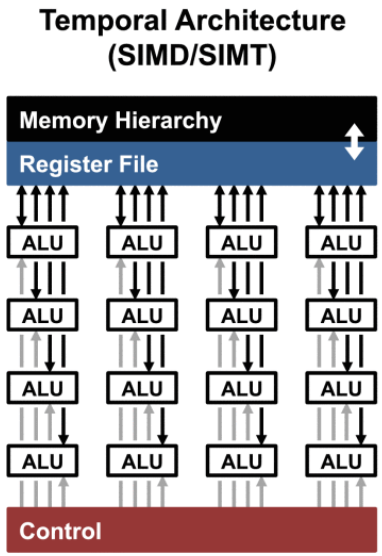
If all data reuse is exploited, DRAM accesses in AlexNet can be reduced from 2,896M to 61M (best case)

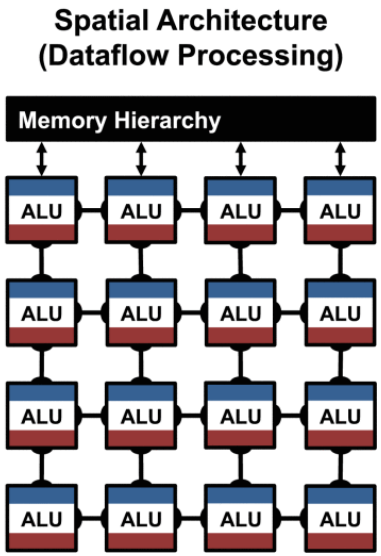
Data Reuse Types

- Data read once from a large expensive memory

- **Temporal Reuse**
  - Store data to a small cheap memory and reuse data several times

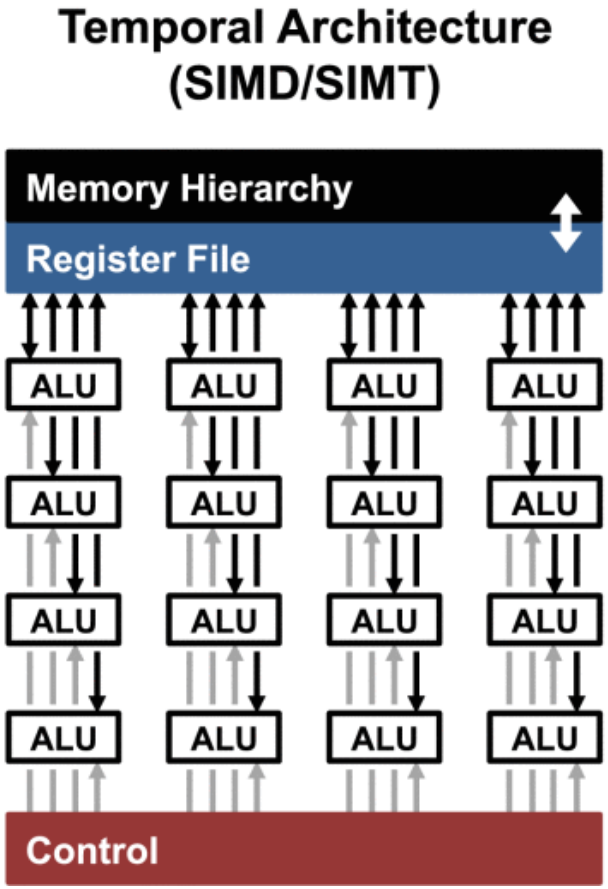
- **Spatial Reuse**
  - Send the same data to multiple PEs and reuse data at different PEs





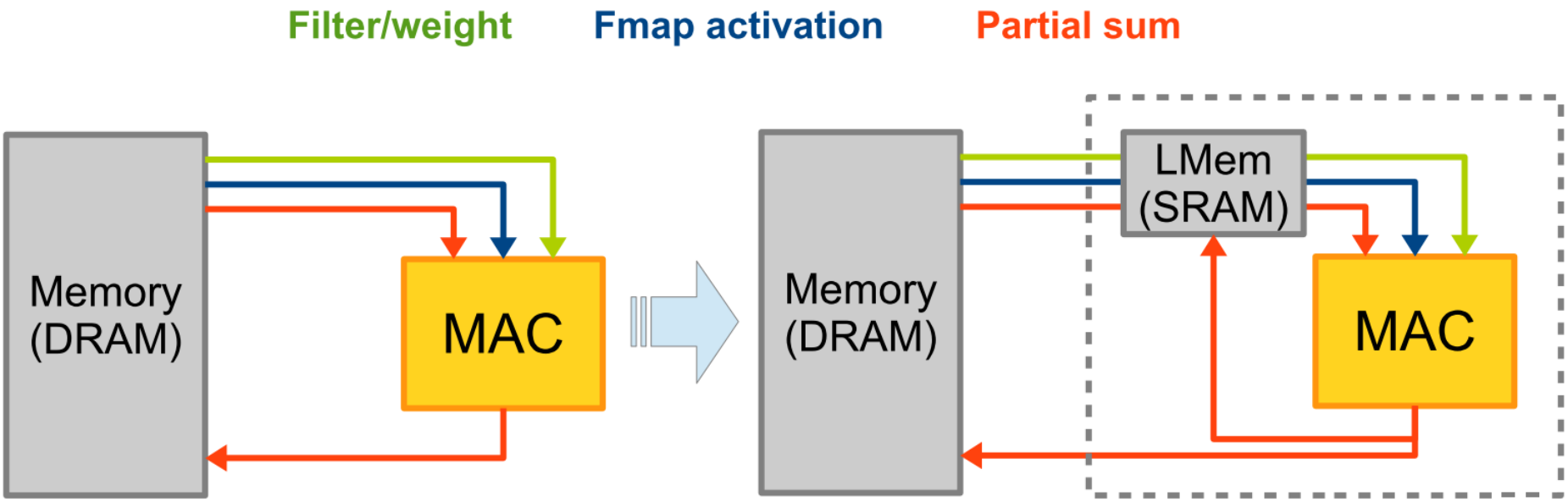
Temporal Architectures

- To improve parallelism
  - SIMD(Single Instruction Multiple Data)
  - SIMT(Single Instruction Multiple Thread)
- Such temporal architecture use a centralized control for a large number of ALUs.
- ✗ These ALUs can only fetch data from the memory hierarchy and cannot communicate directly with each other.



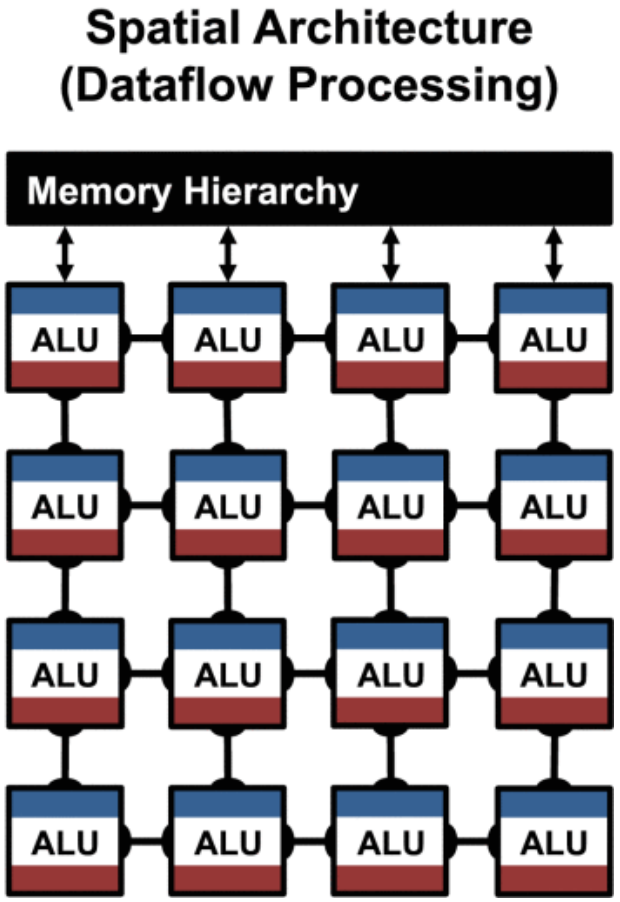
Temporal Reuse

- The same data is used more than once by the same PE
- Exploited by adding intermediate memory level



Spatial Architectures

- ALUs form a processing chain so that they can pass data from one to another directly.
- Each ALU can have its own control logic and local memory
  - Register file(RF)
  - Processing element(PE)
    - Each PE is a simple multiply-accumulator
- Extremely large number of PEs
- Very high peak throughput!

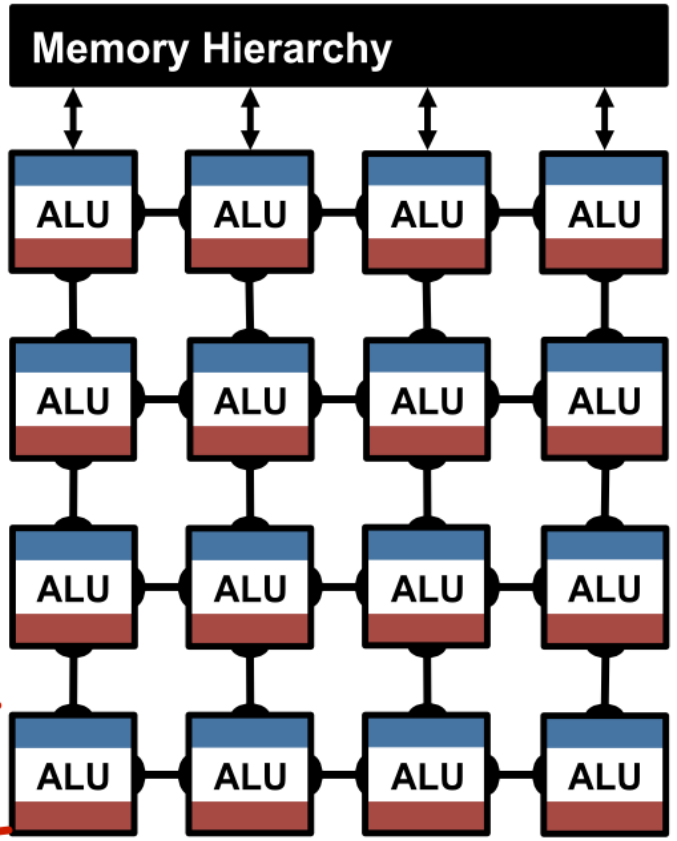


Advantages of Spatial Architecture

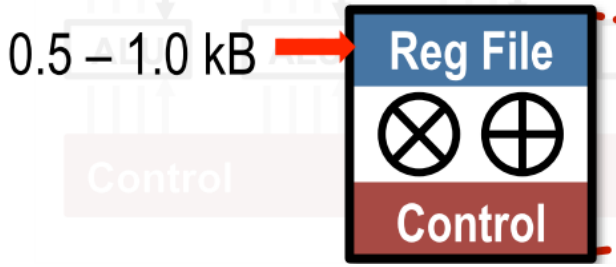
Efficient Data Reuse
Distributed local storage (RF)

Inter-PE Communication
Sharing among regions of PEs

Spatial Architecture (Dataflow Processing)



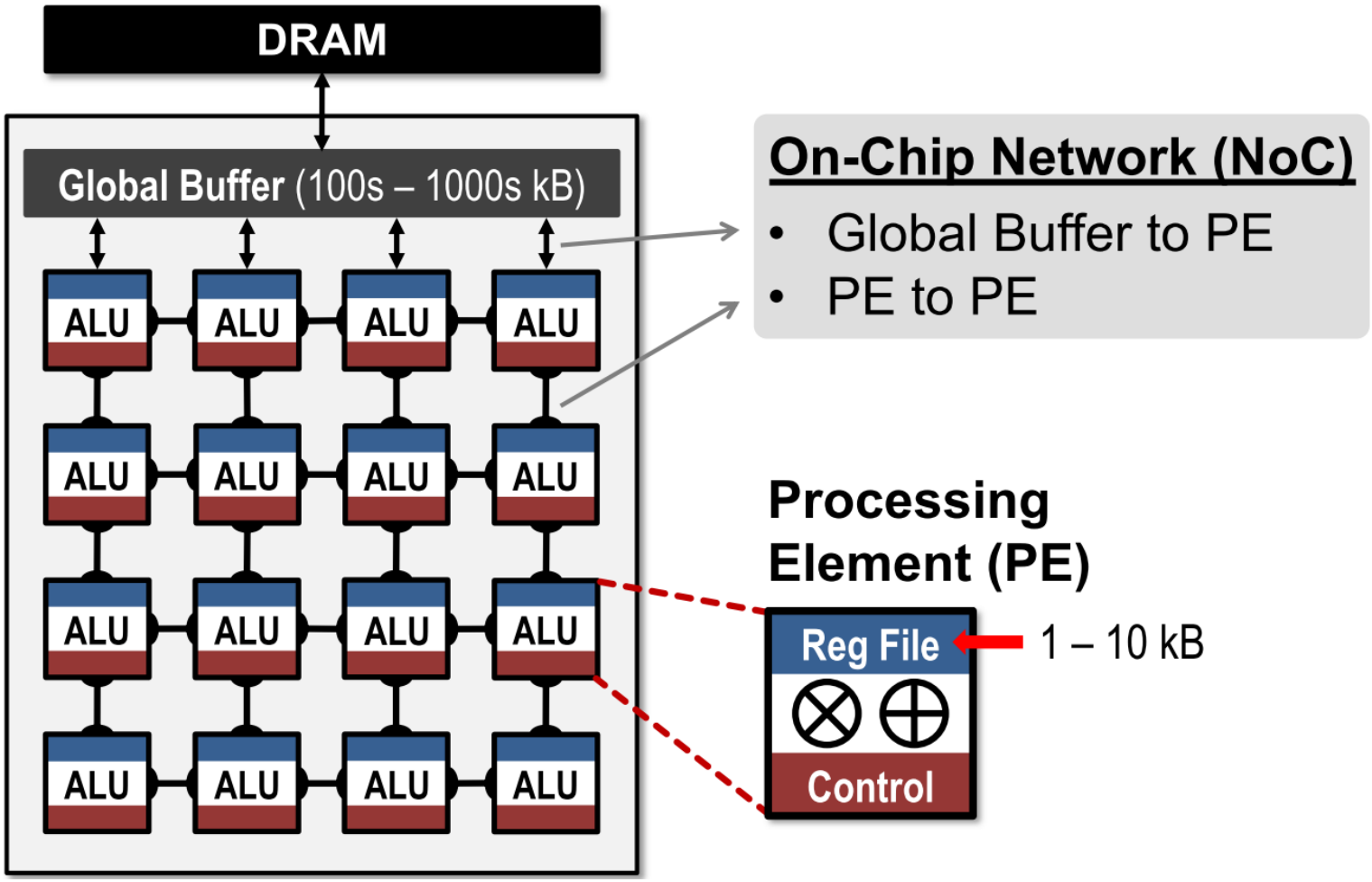
Processing Element (PE)



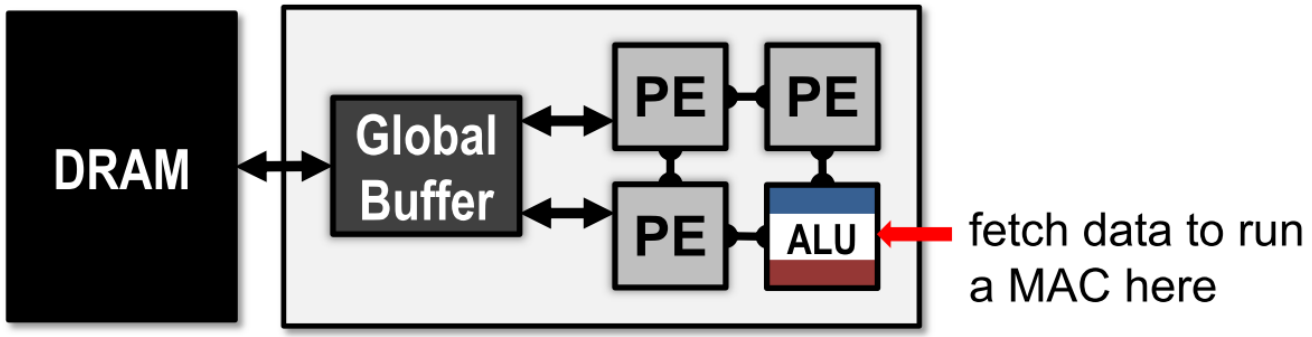
Short Summary to Spatial Reuse

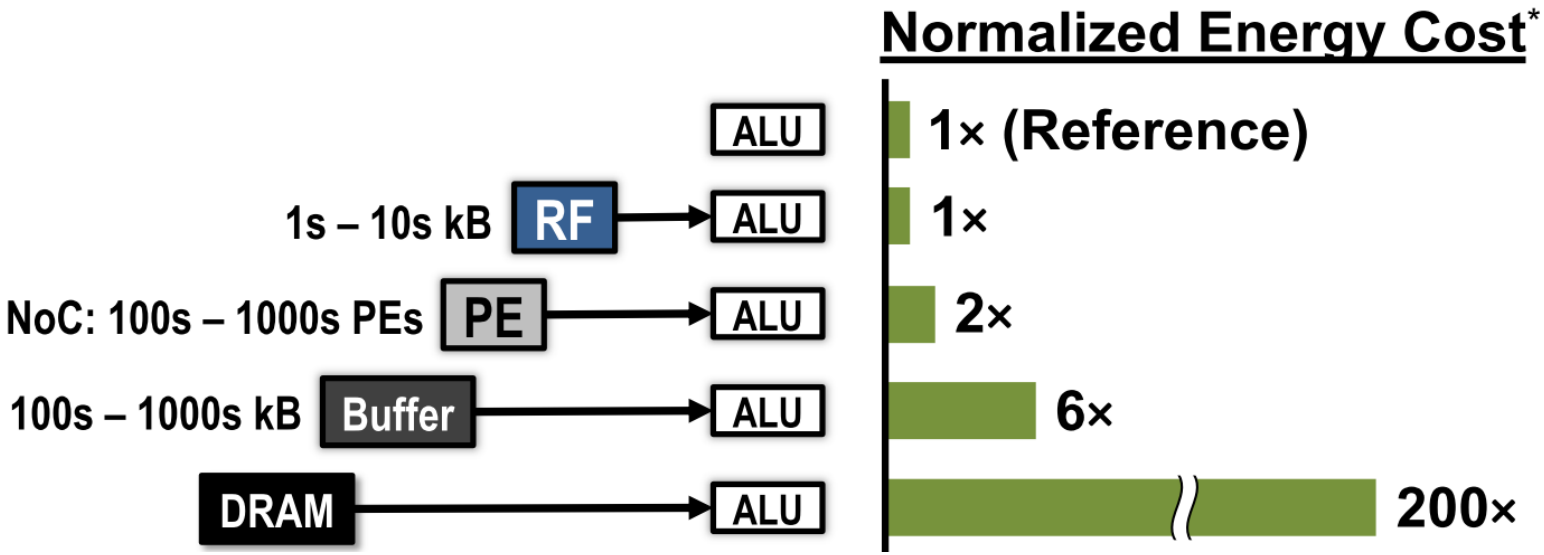
- The same data value is used by more PEs
- Exploitation
  - Data is read once from memory and multicast to all the PEs
- Advantages
  - Reduce the number of memory access
  - Reduce the required bandwidth from the memory

Spatial Architecture for DNN



Multi-Level Low-Cost Data Access

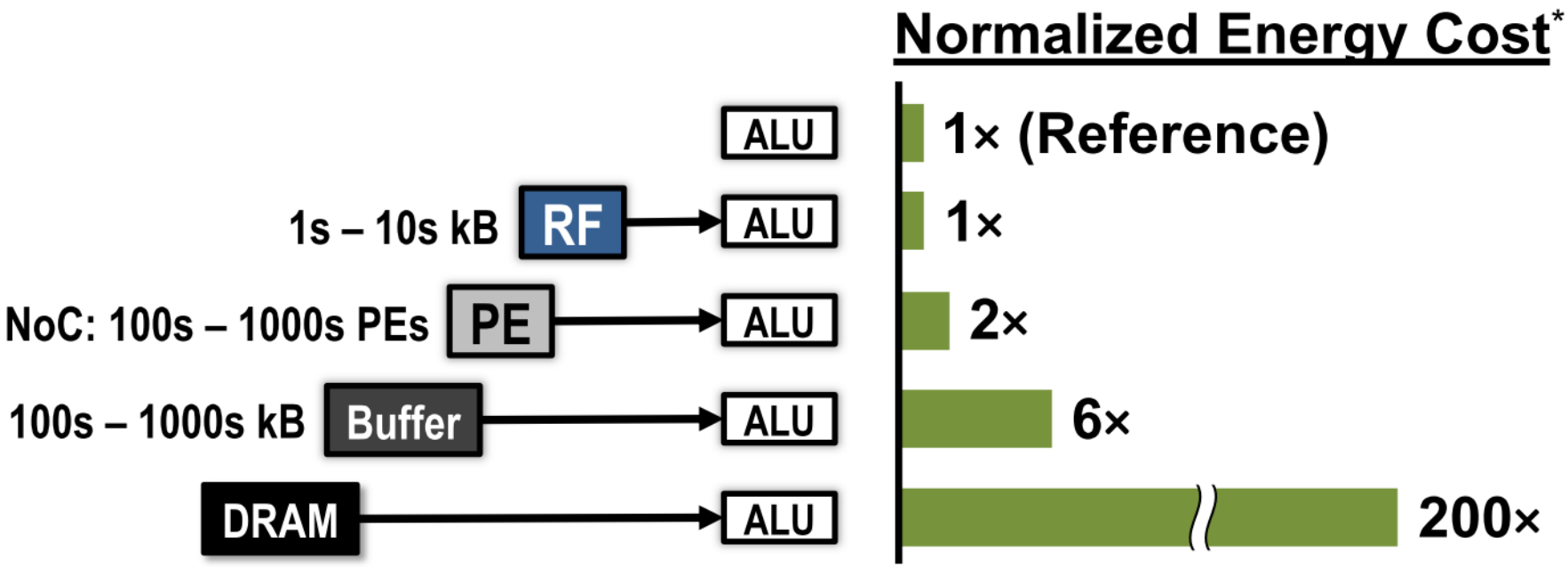




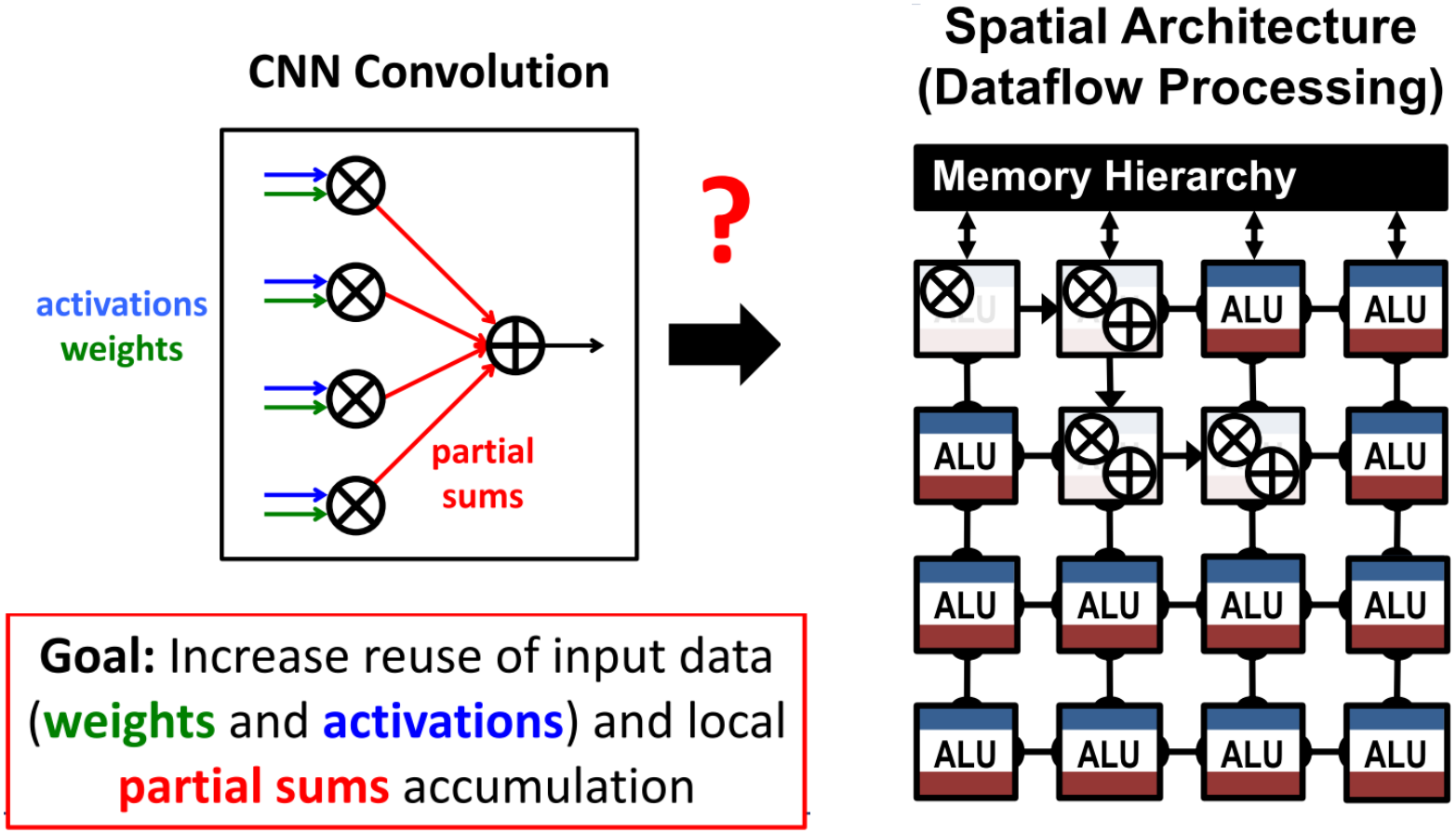
*measured from a commercial 65nm processor

Multi-Level Low-Cost Data Access

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



How to Map the Dataflow?

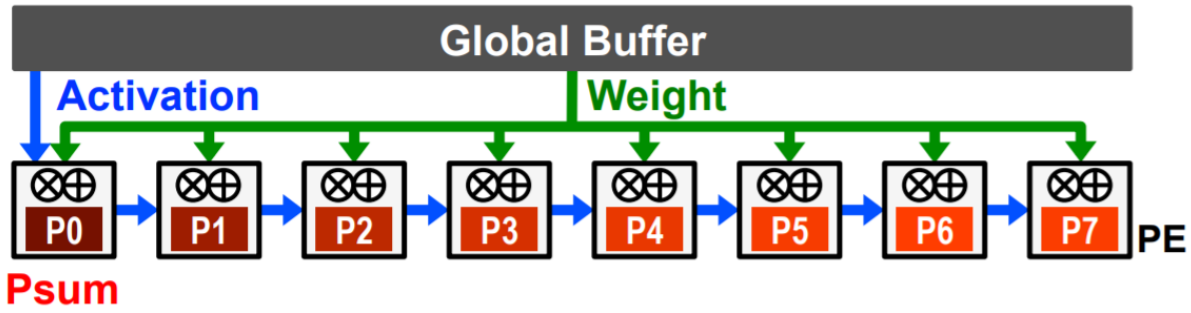


Goal: Increase reuse of input data (**weights** and **activations**) and local **partial sums** accumulation

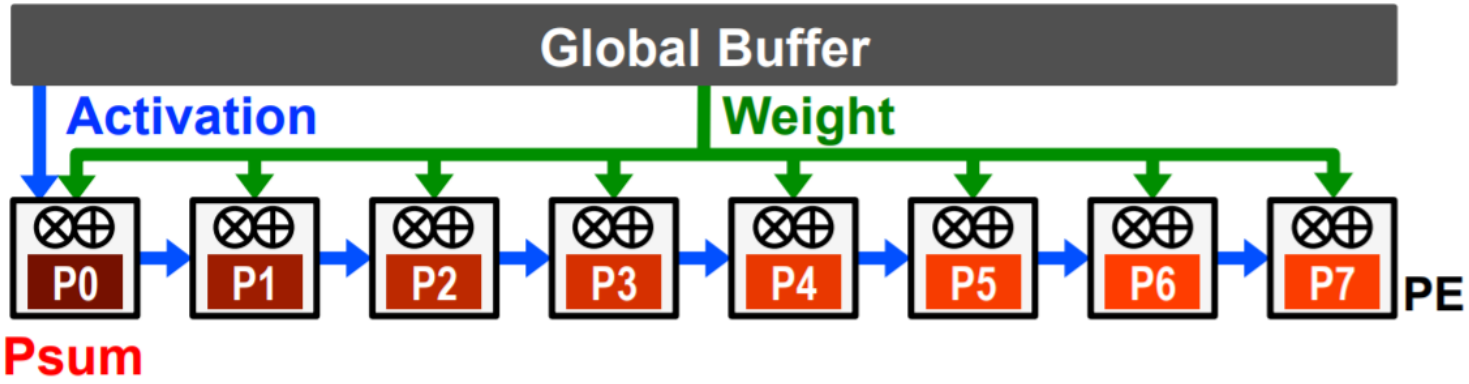
Data Flow Techniques

Output Stationary (1/2)

- Keep partial sums cached on PEs – Work on subset of output at a time
  - Effective for FC layers, where each output depend on many input/weights
  - Also for convolution layers when it has too many layers
- Each weight is broadcast to all PEs, and input related to neighboring PEs
  - Intuition: Each PE is working on an adjacent position in an output sub-space



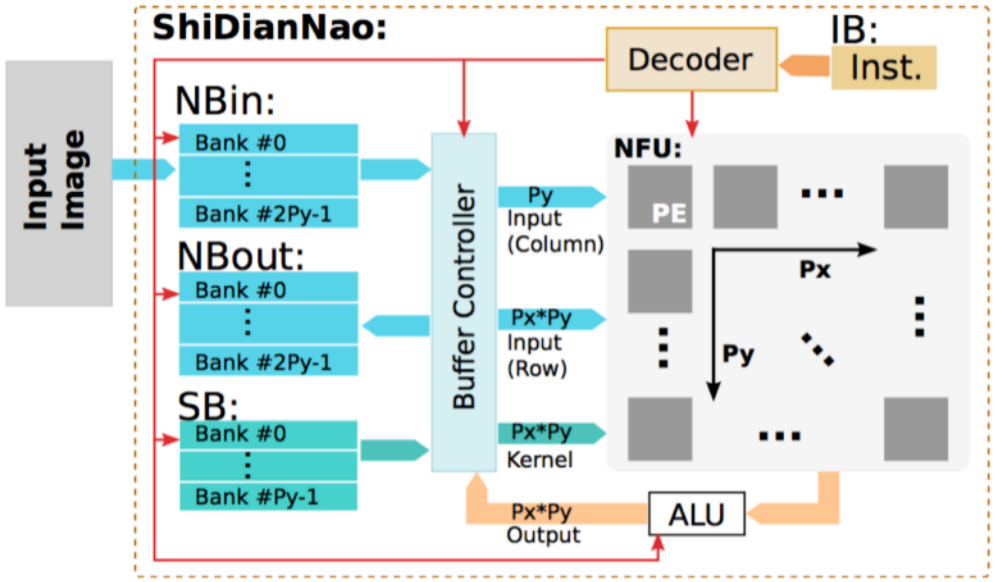
Output Stationary (2/2)



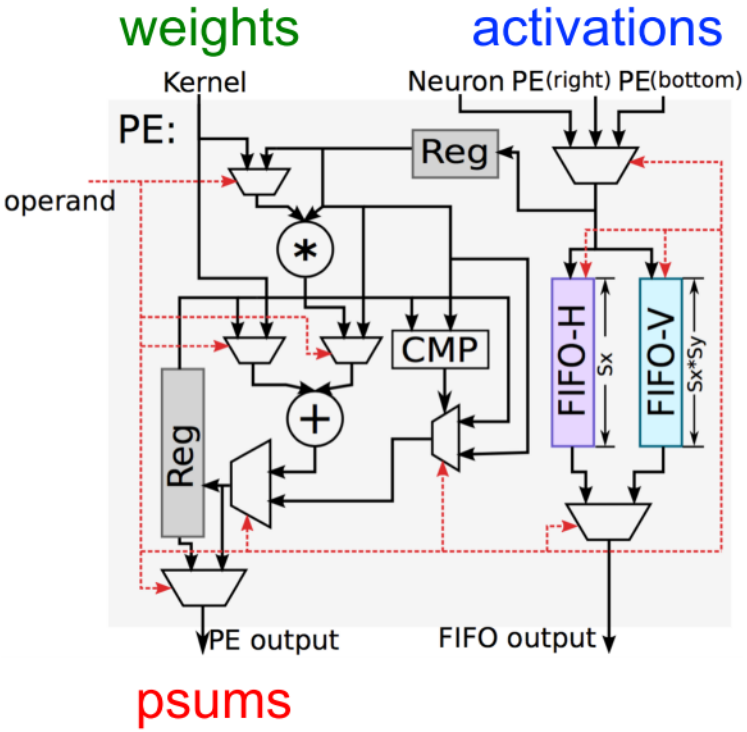
- **Minimize partial sum** R/W energy consumption
  - maximize local accumulation
- **Broadcast/Multicast filter weights** and **reuse activations spatially** across the PE array

OS Example: ShiDianNao

Top-Level Architecture

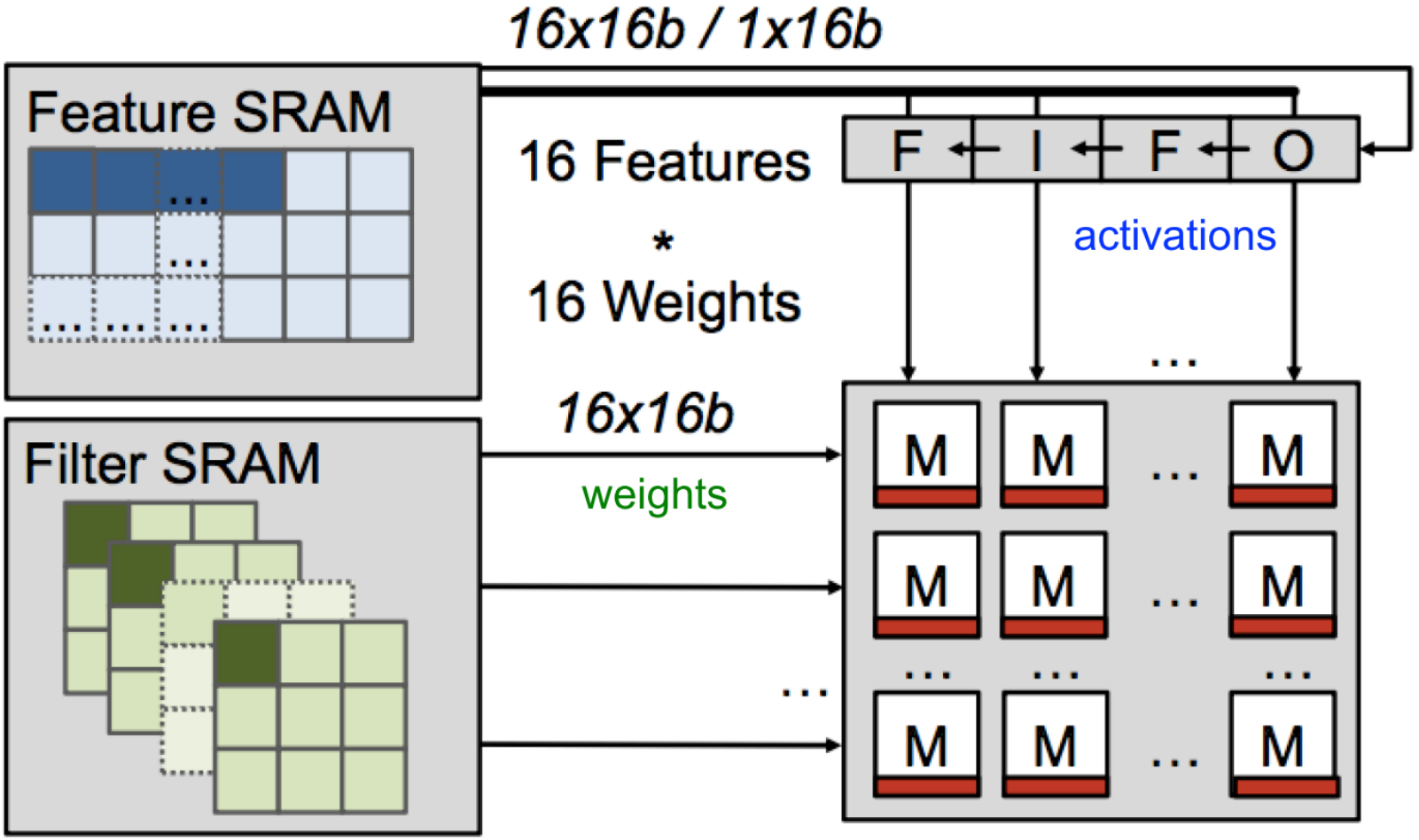


PE Architecture



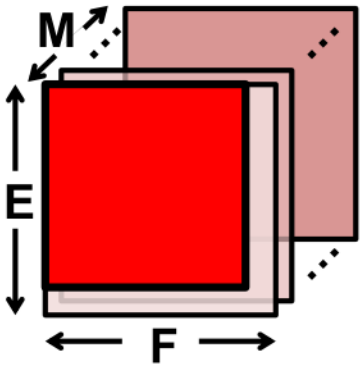
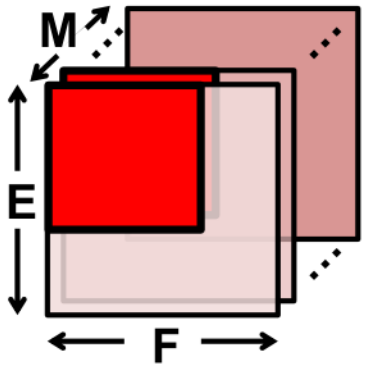
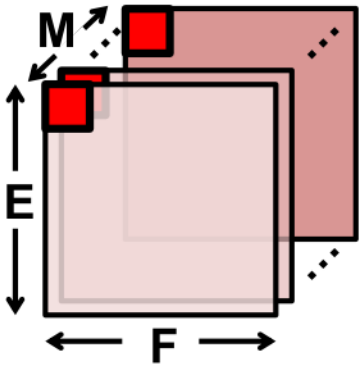
Du et al., ISCA 2015

OS Example: ENVISION

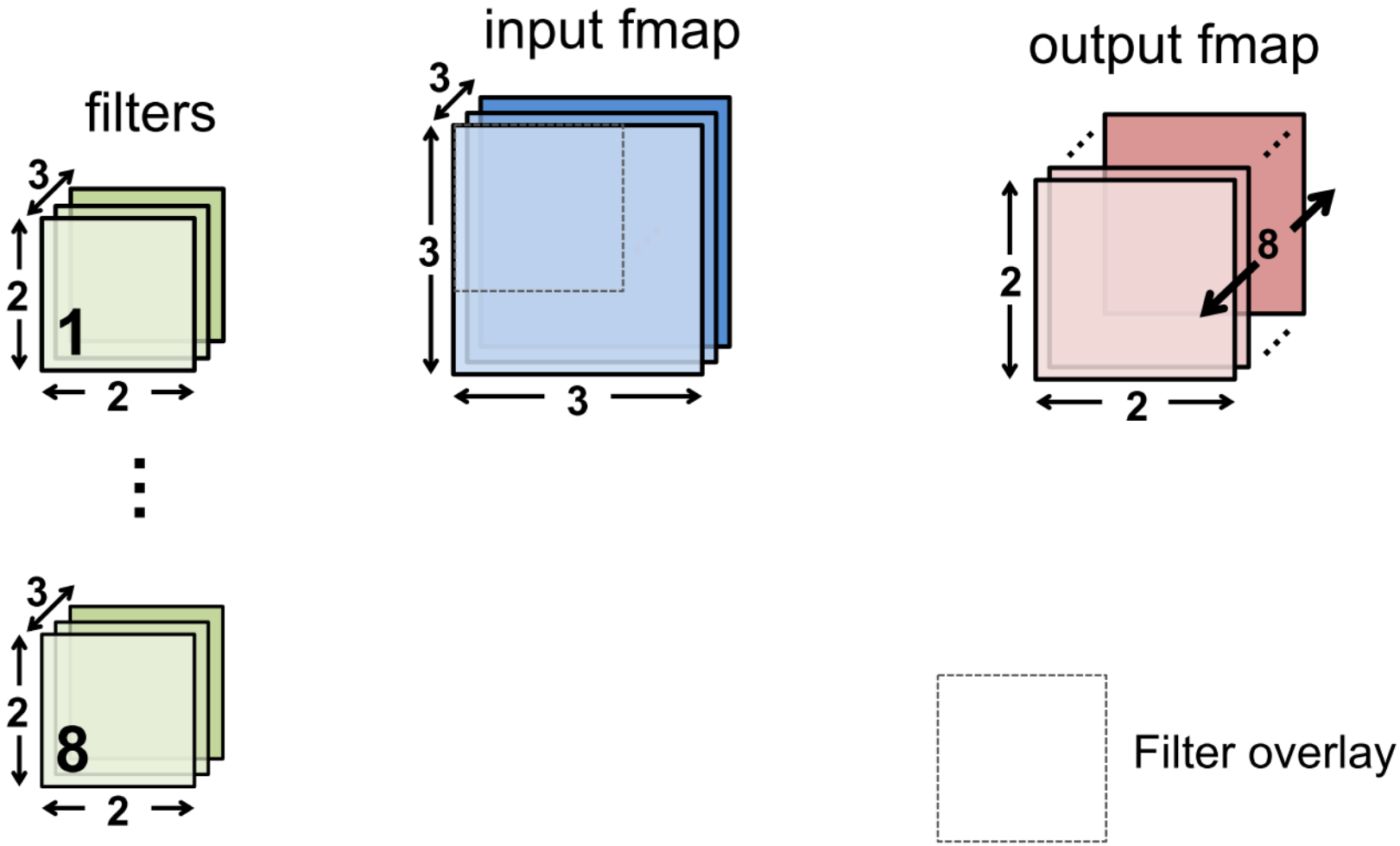


[Moons et al., VLSI 2016, ISSCC 2017]

Variants of Output Stationary

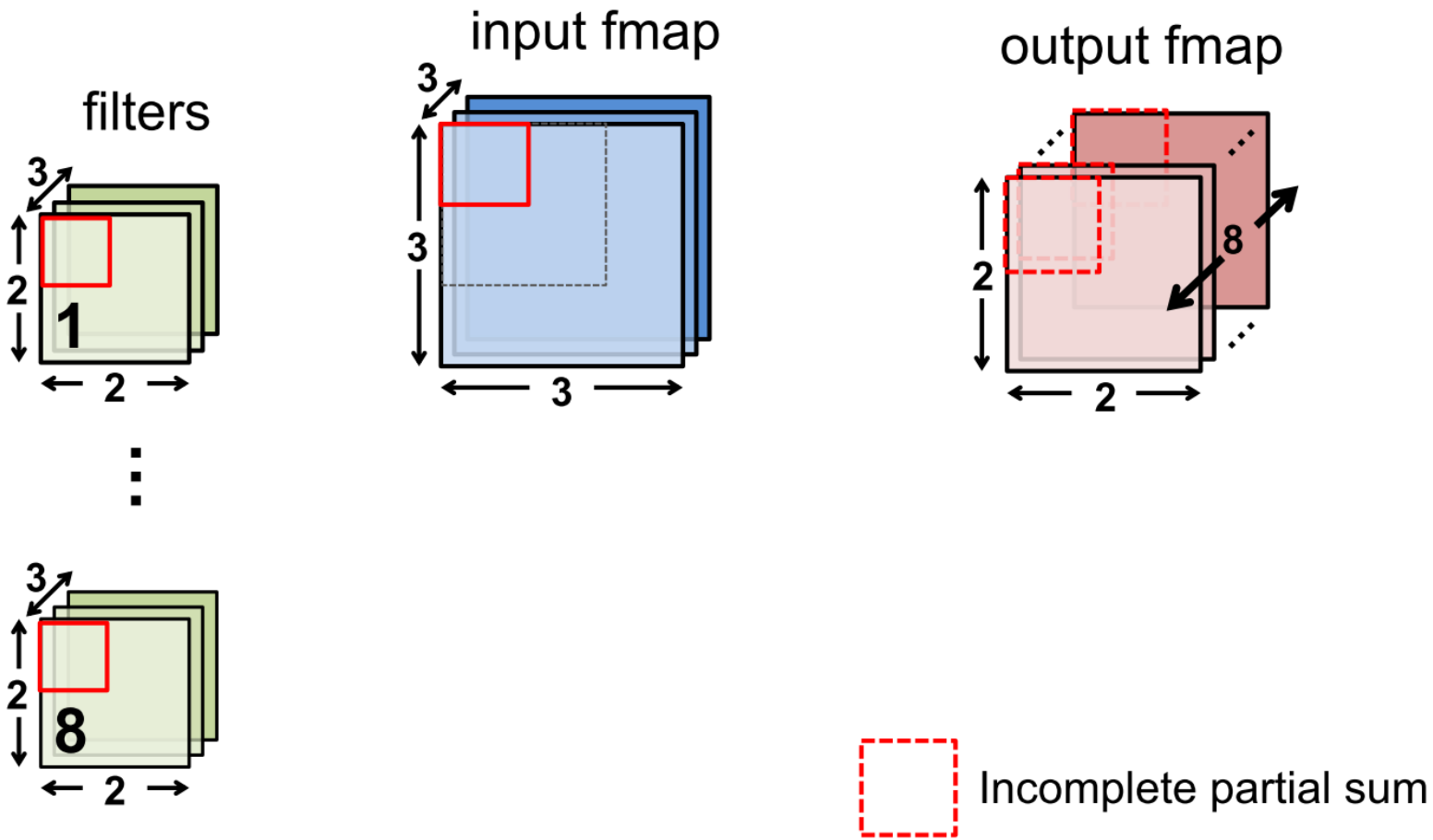
	OS_A	OS_B	OS_C
Parallel Output Region			
# Output Channels	Single	Multiple	Multiple
# Output Activations	Multiple	Multiple	Single
Notes	Targeting **CONV** layers		Targeting **FC** layers

OS Example


filters
3
2
1
2
3
8
2
input fmap
3
3
3
output fmap
8
2
2
Filter overlay

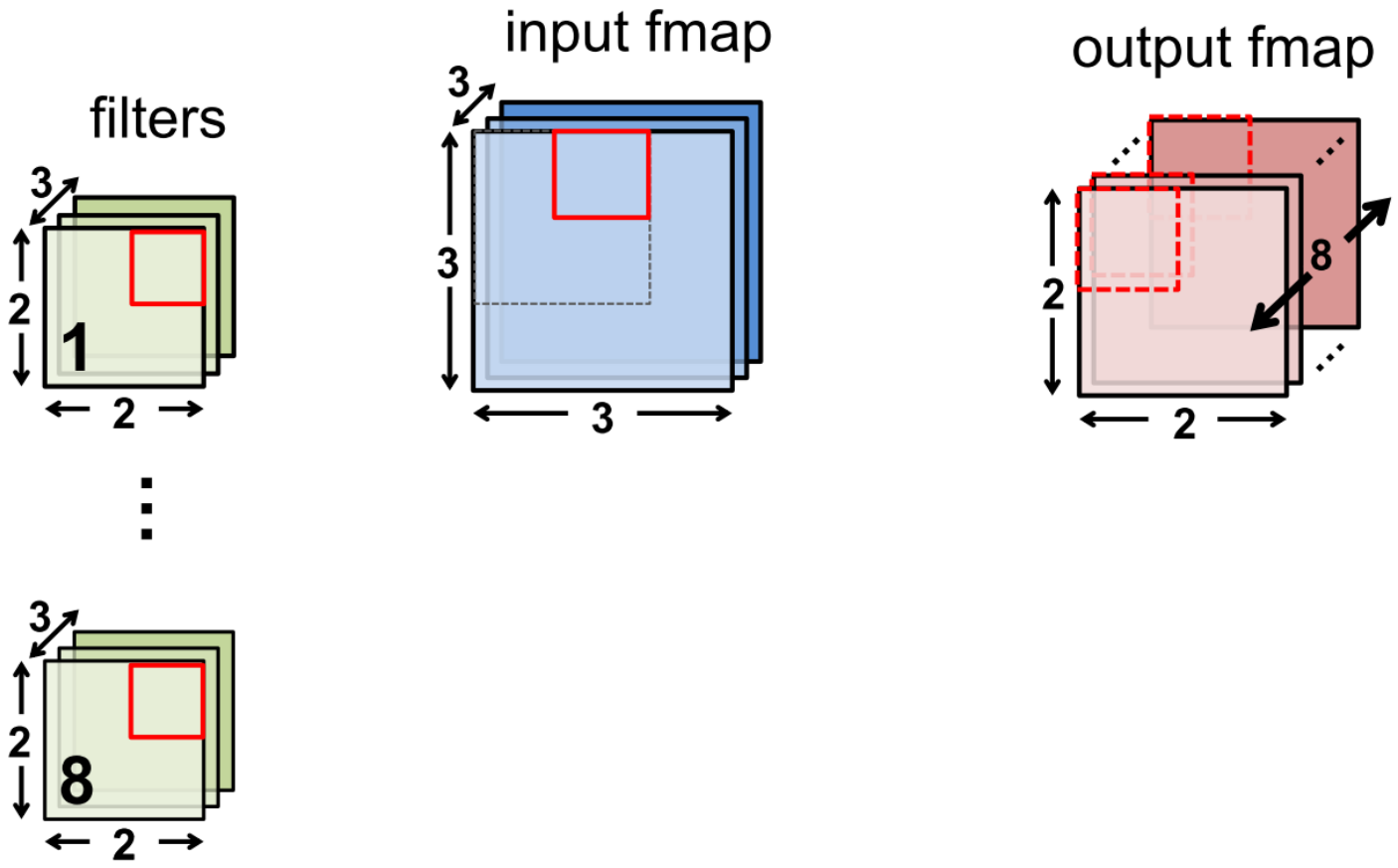
OS Example

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



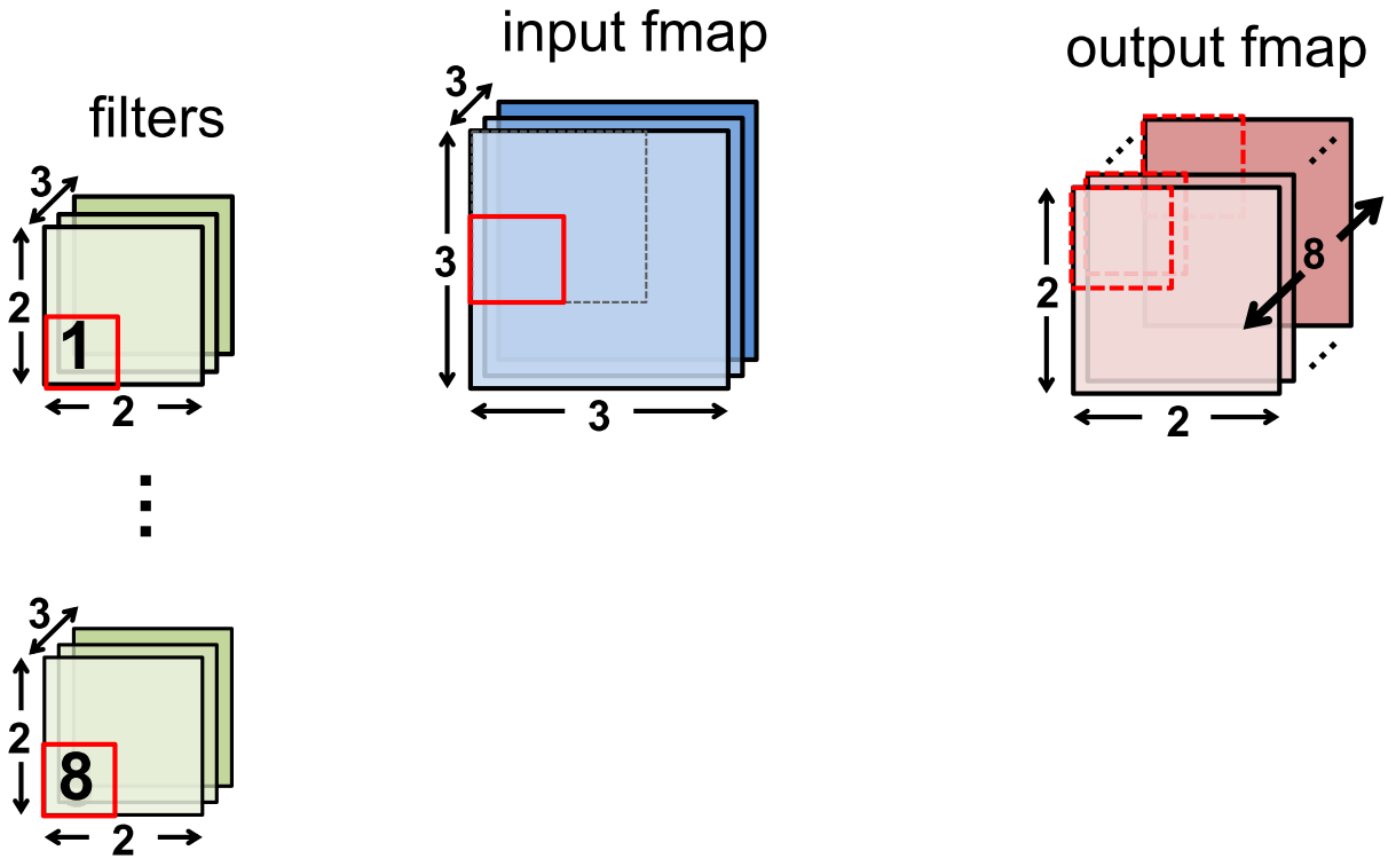
OS Example

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



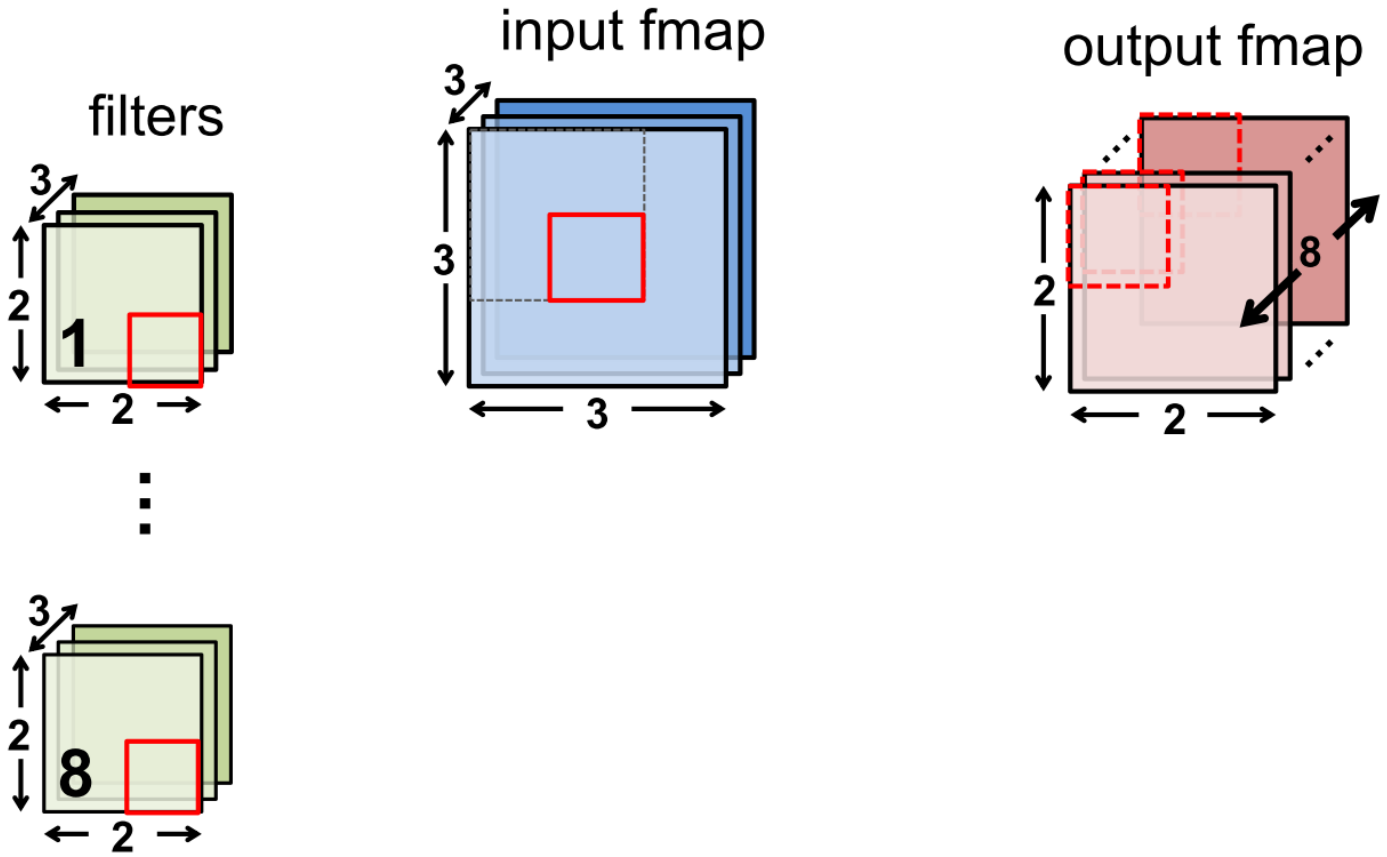
OS Example

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



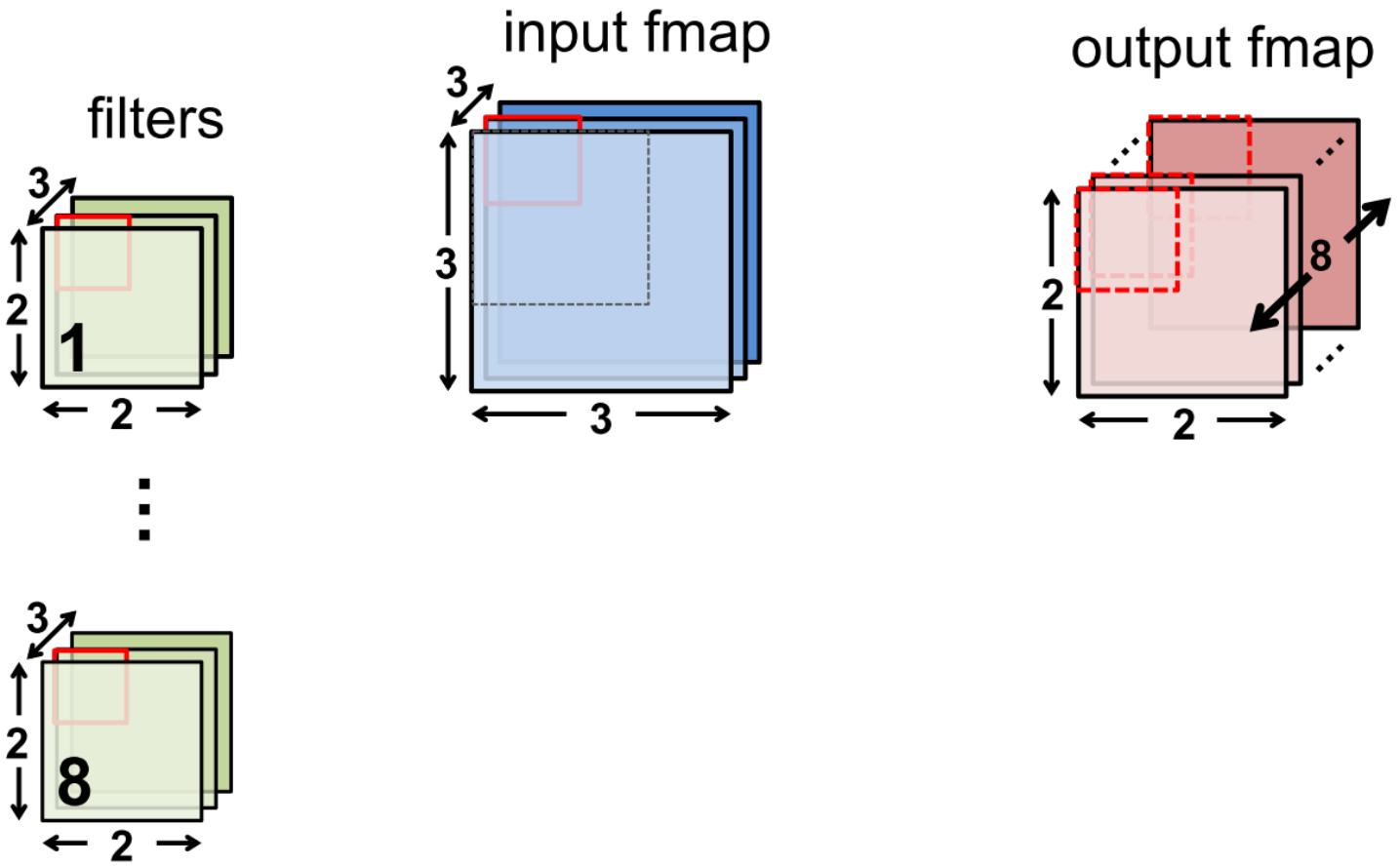
OS Example

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



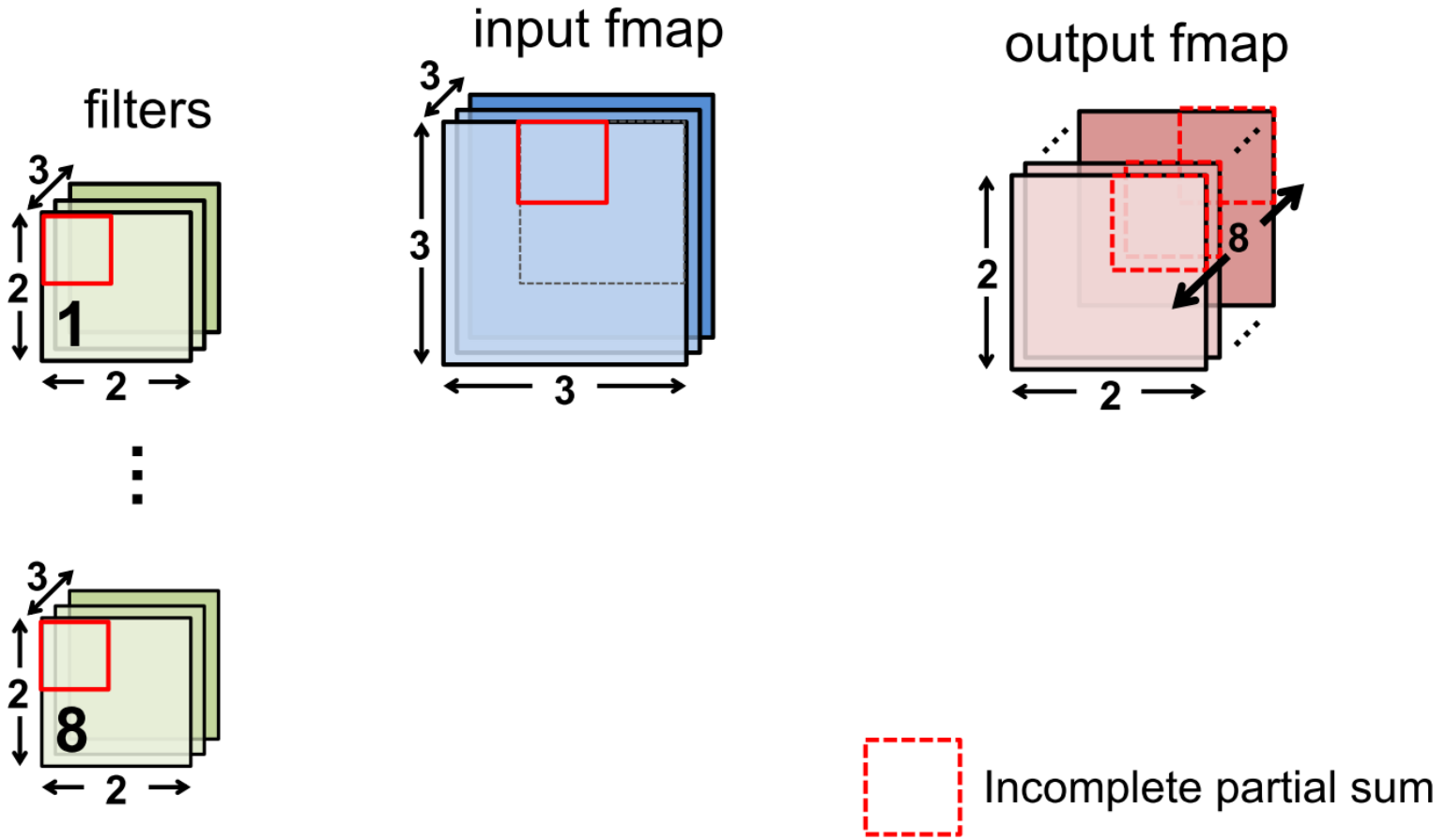
OS Example

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

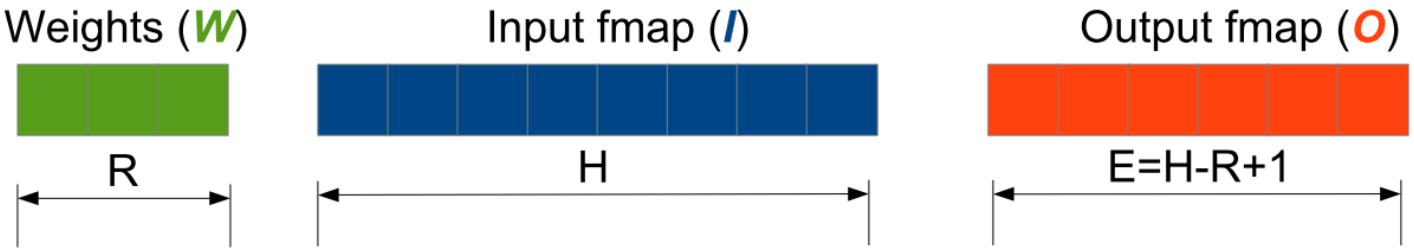


OS Example

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

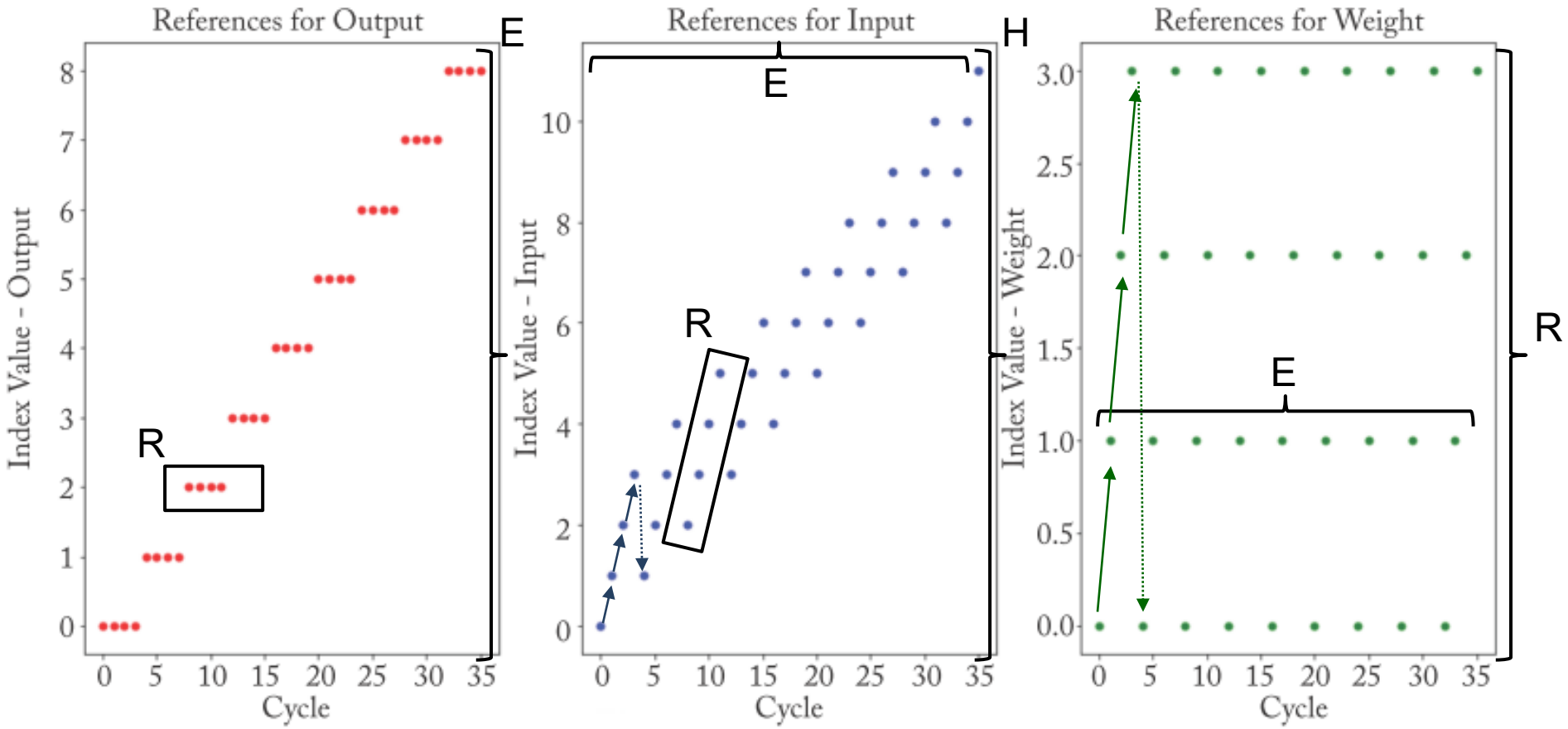


1-D Convolution – Output Stationary




```
int W[R];
int I[H];
int O[E];

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

Space-time Diagram of OS

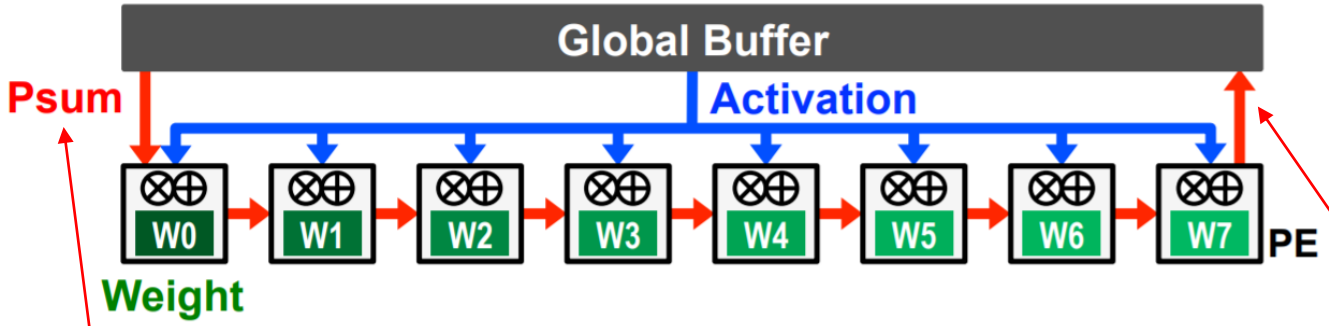


MACs	Weight Reads	Input Reads	Output Reads	Output Write
E x R	E x R	E x R	0	E

Weight Stationary (1/2)

- Keep weights cached in PE register files
  - Effective for convolution especially if all weights can fit in-PEs
- Each activation is broadcast to all PEs, and computed partial sum is forwarded to other PEs to complete computation
  - Intuition: Each PE is working on an adjacent position of an input row

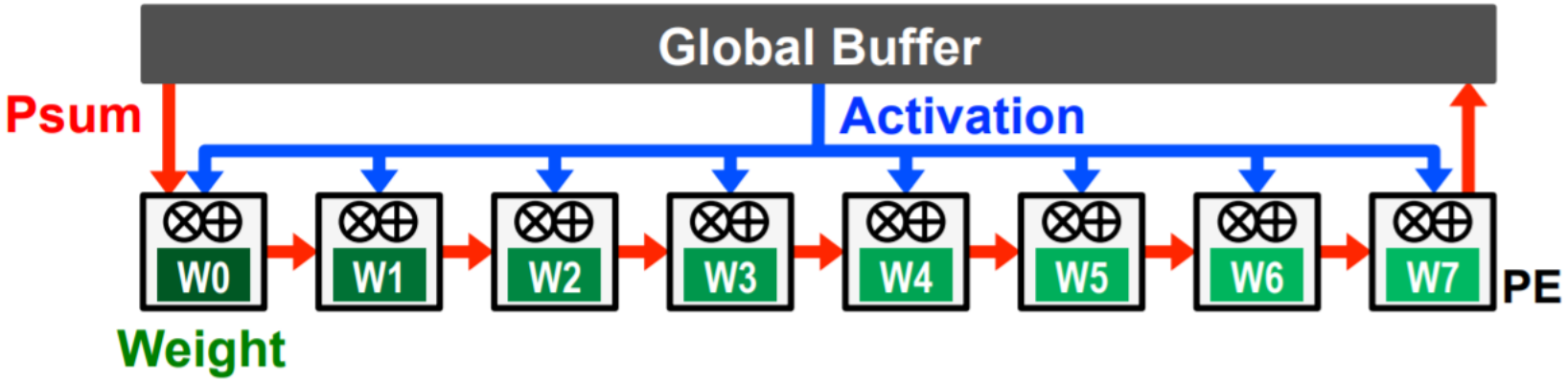
Weight stationary convolution for a row in the convolution



Partial sum of a previous activation row

Partial sum for stored for next activation row, or final sum

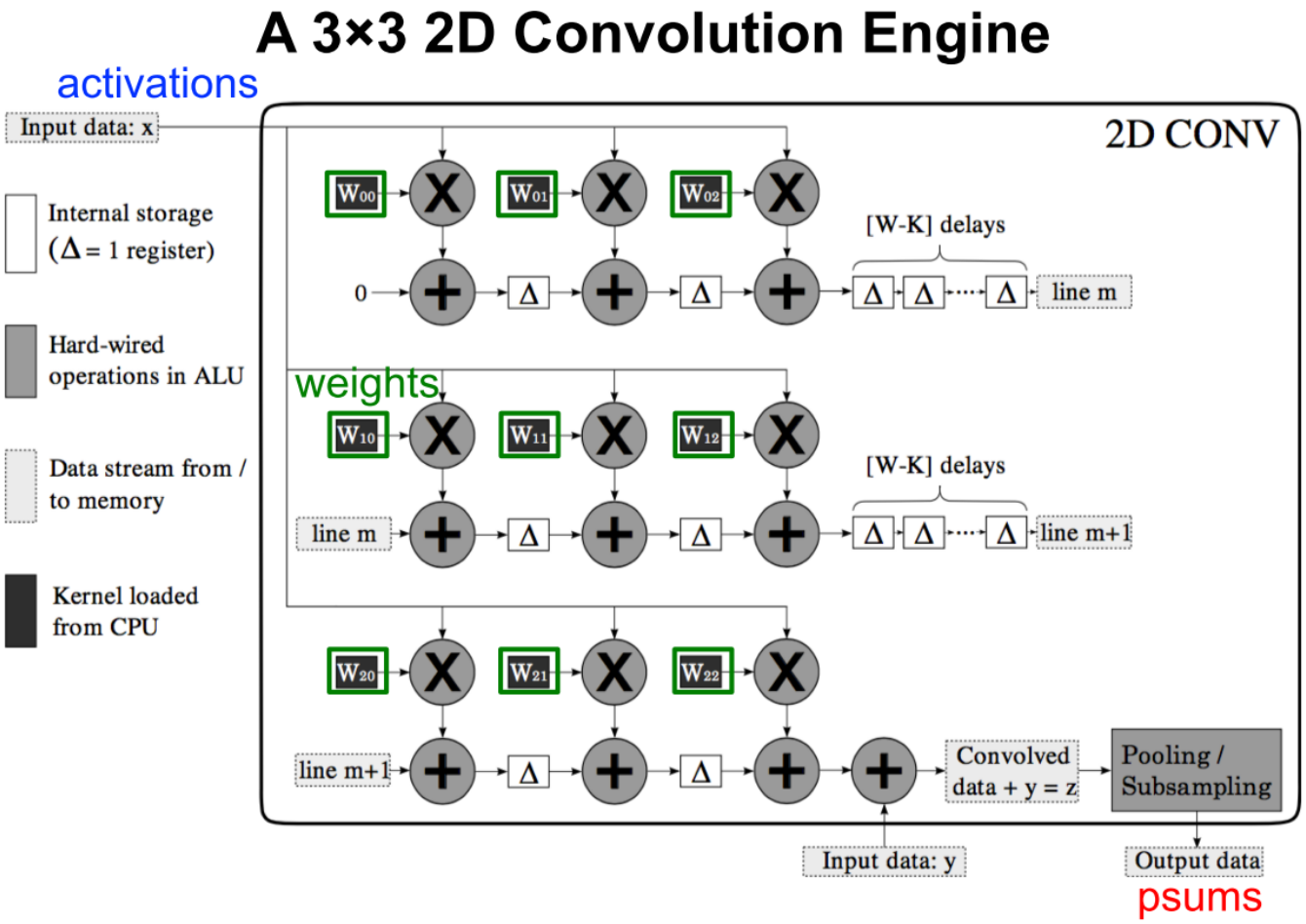
Weight Stationary (2/2)



- **Minimize weight** read energy consumption
  - maximize convolutional and filter reuse of weights

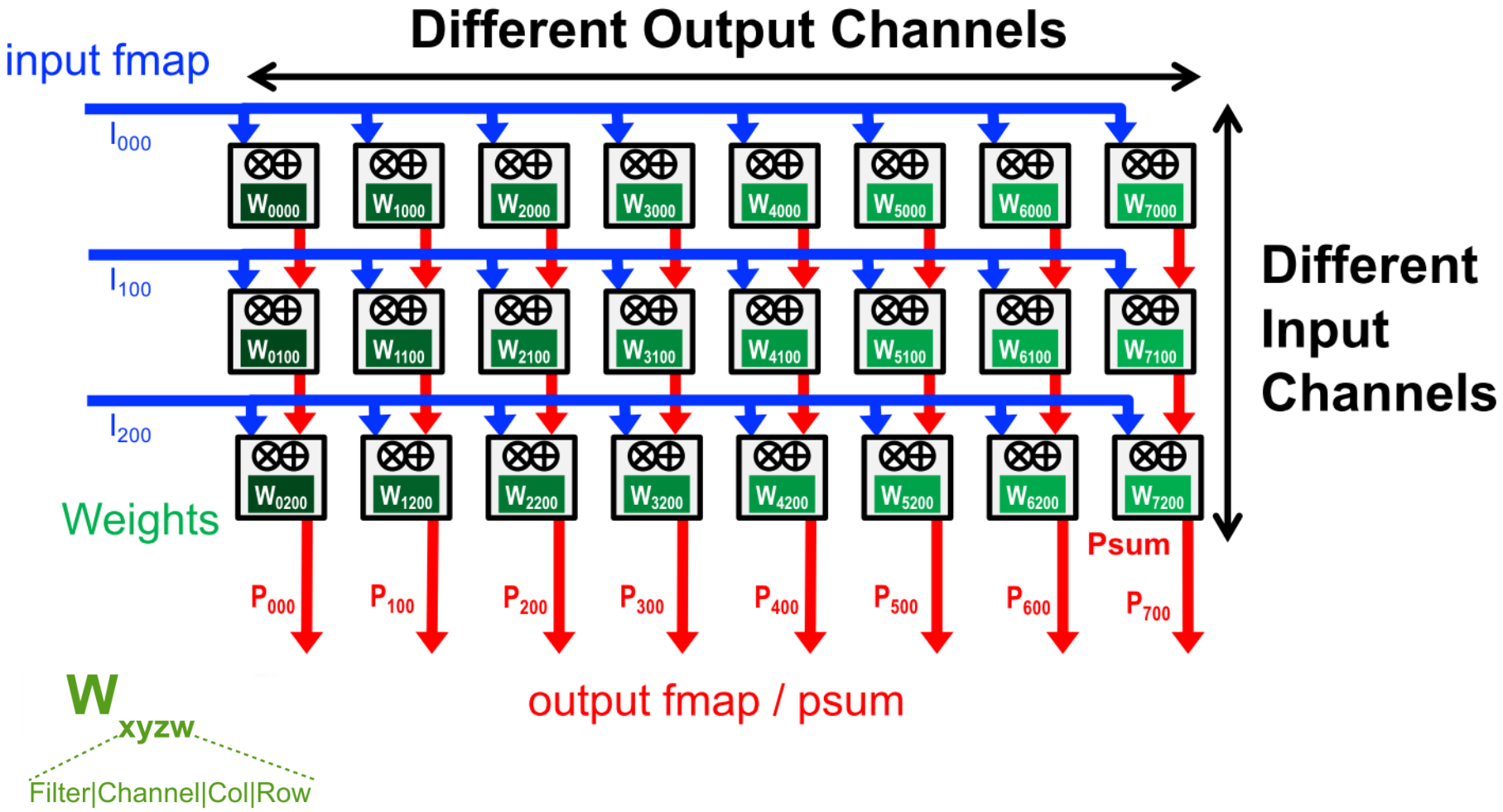
- **Broadcast activations** and **accumulate partial sums spatially** across the PE array

WS Example: nn-X (NeuFlow)



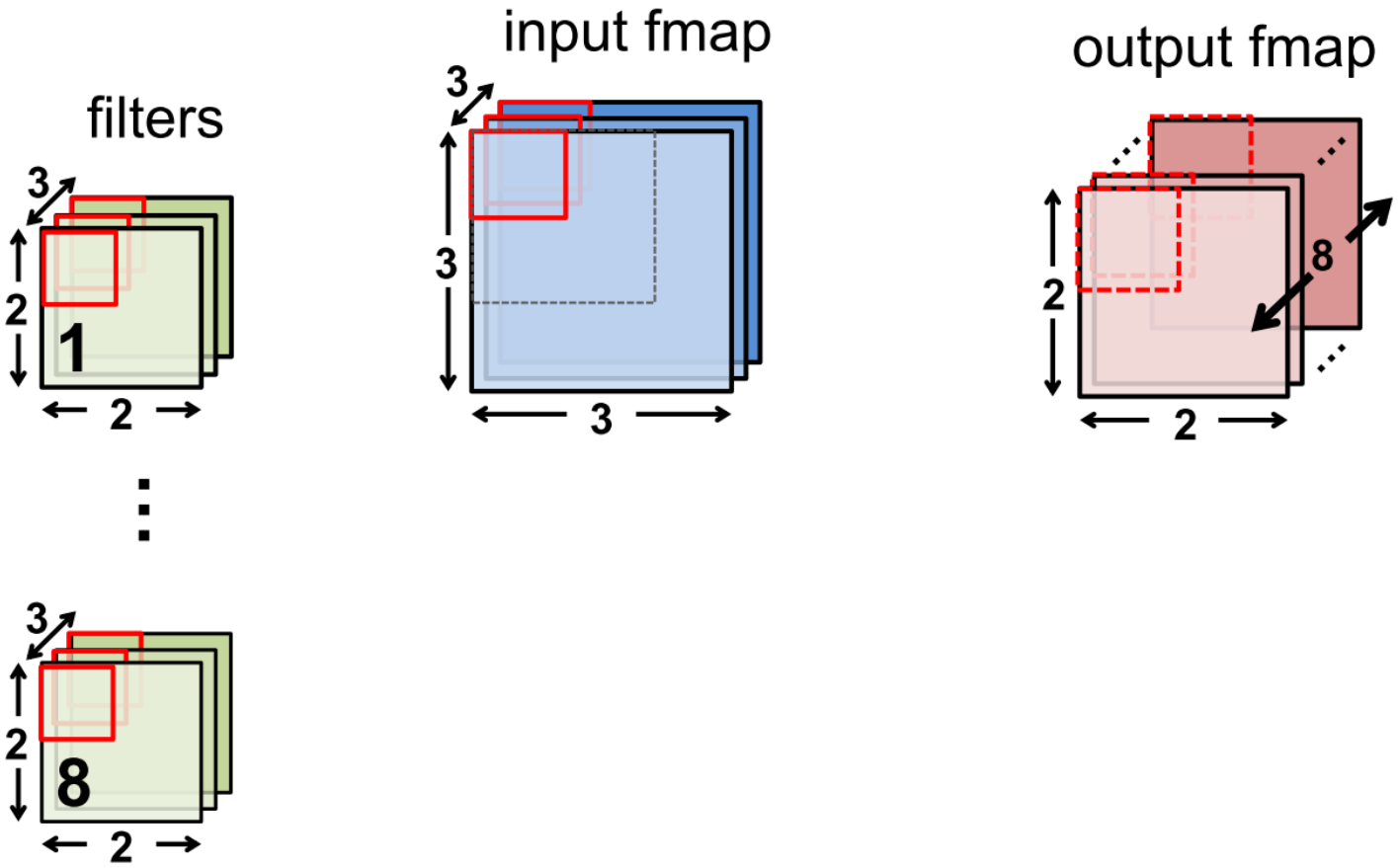
[Farabet et al., ICCV 2009]

WS Example: NVDLA (simplified)



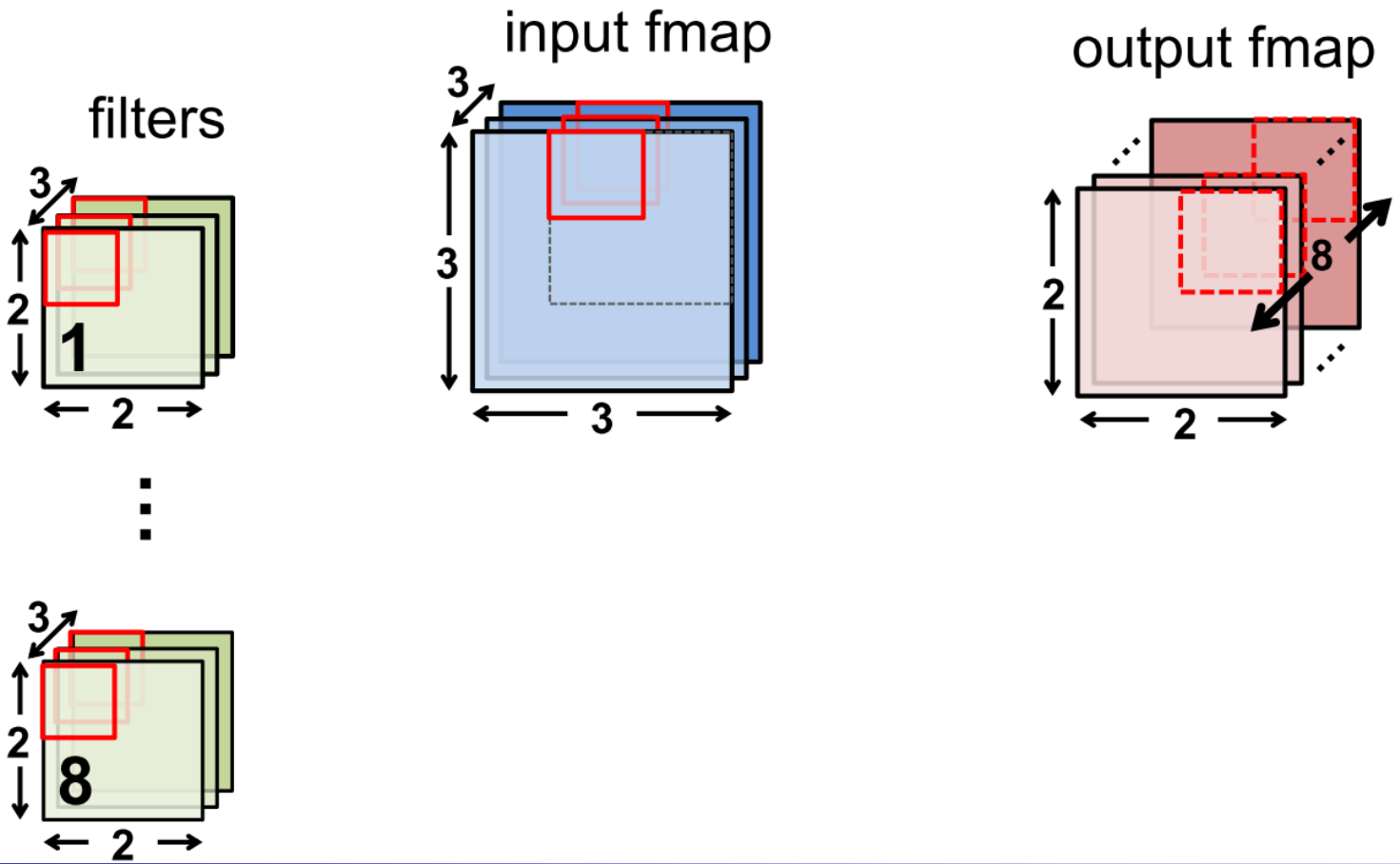
WS Example

- Cycle through input and output fmap (hold weights)



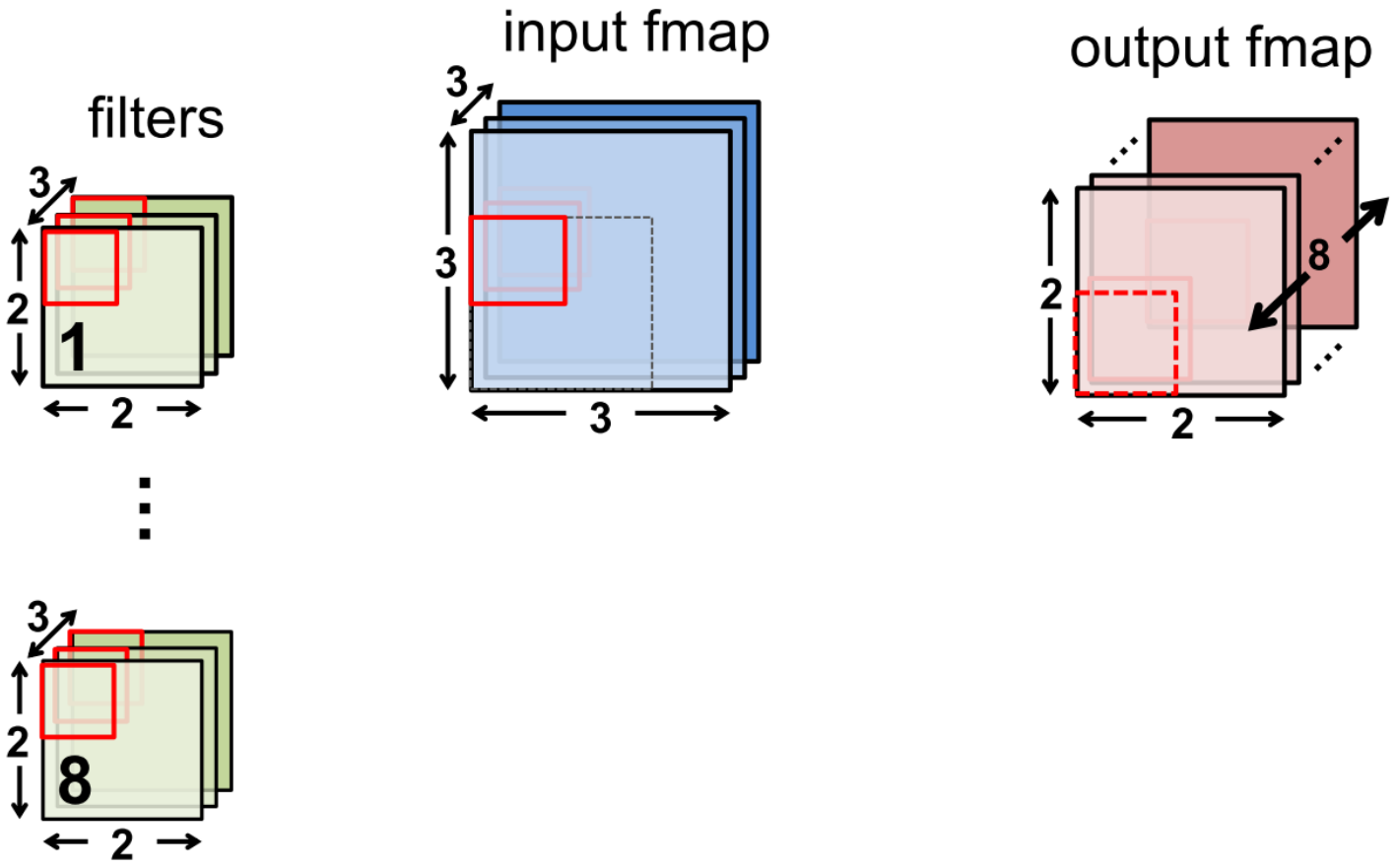
WS Example

- Cycle through input and output fmap (hold weights)



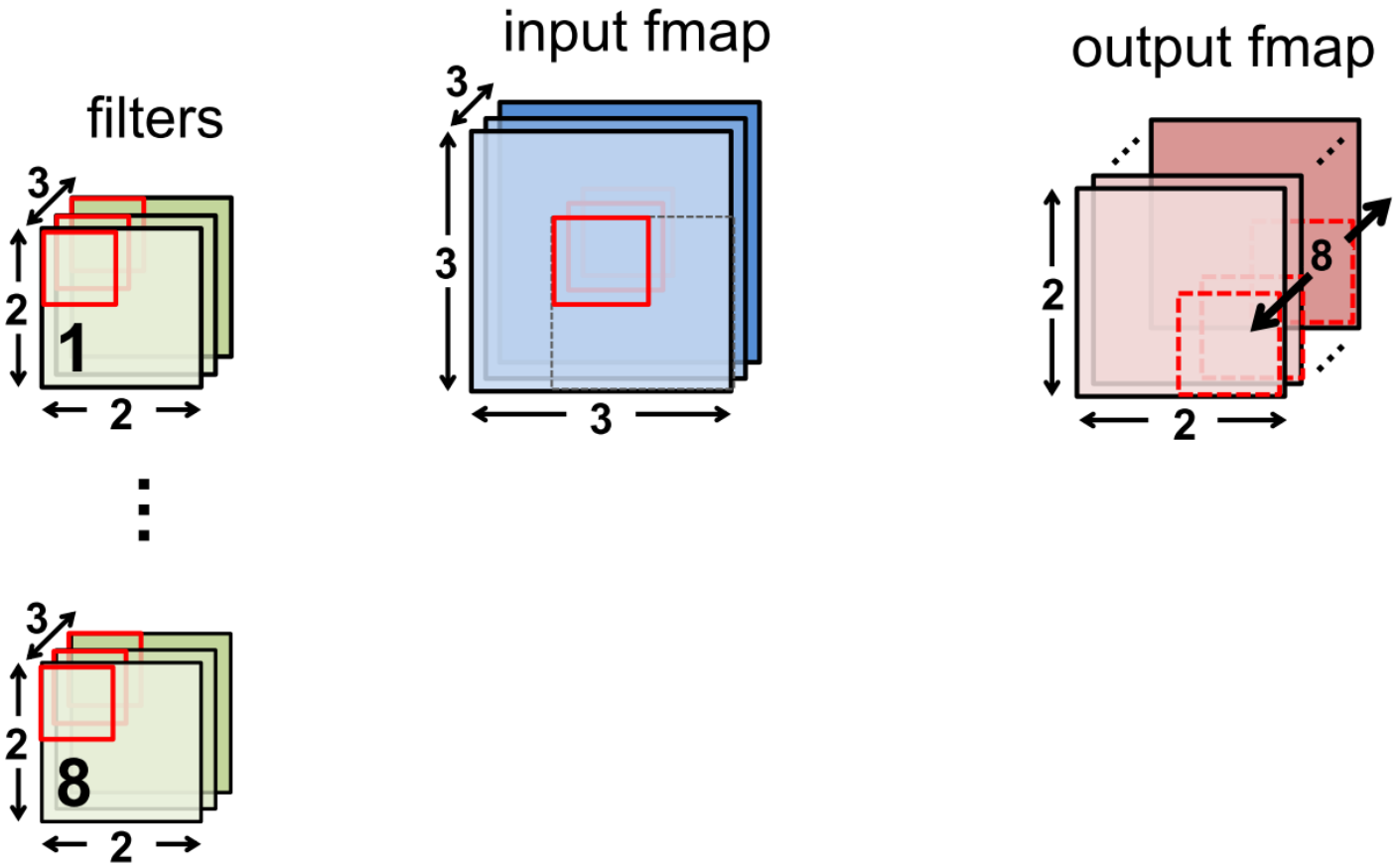
WS Example

- Cycle through input and output fmap (hold weights)



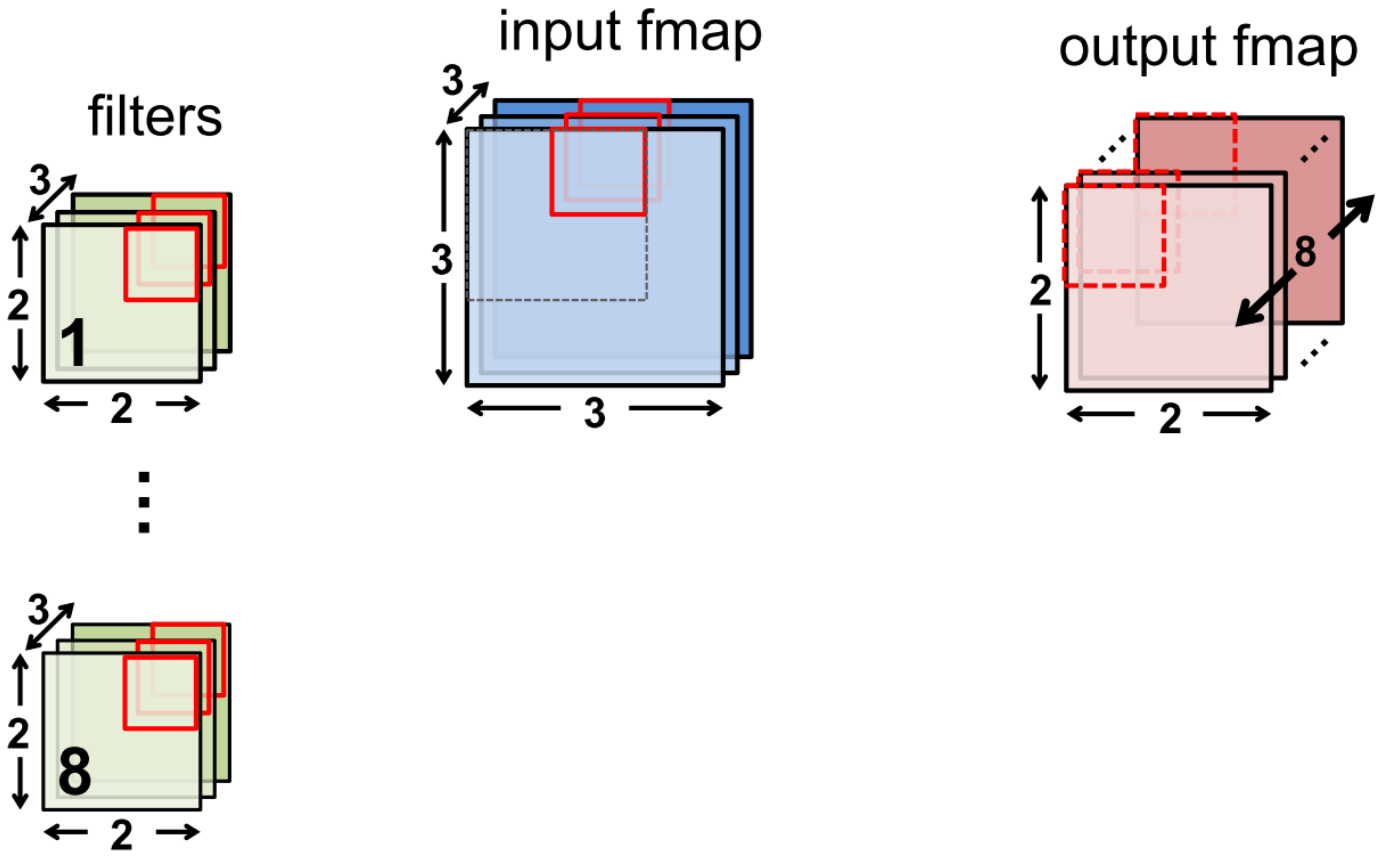
WS Example

- Cycle through input and output fmap (hold weights)



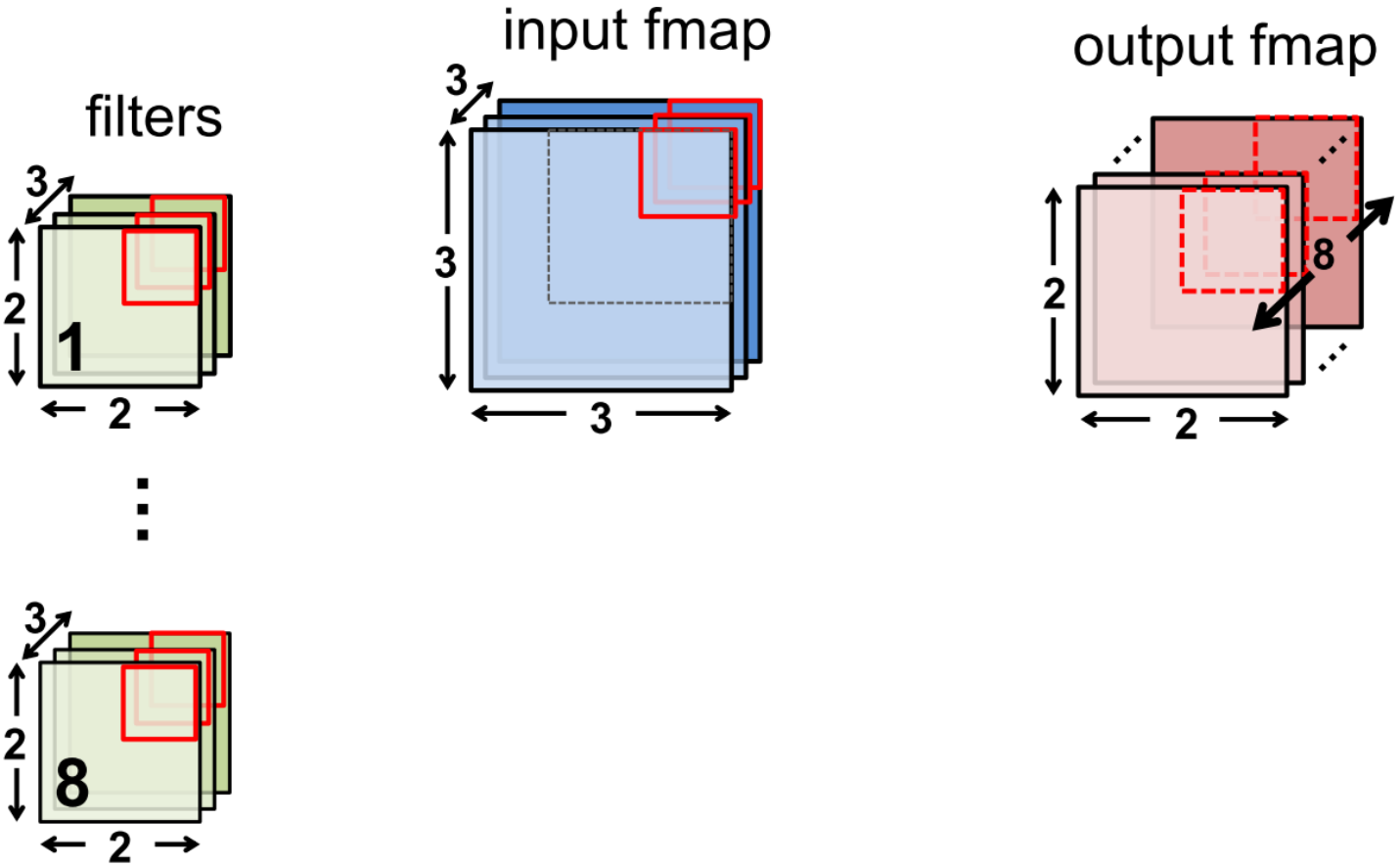
WS Example

- Load new weights

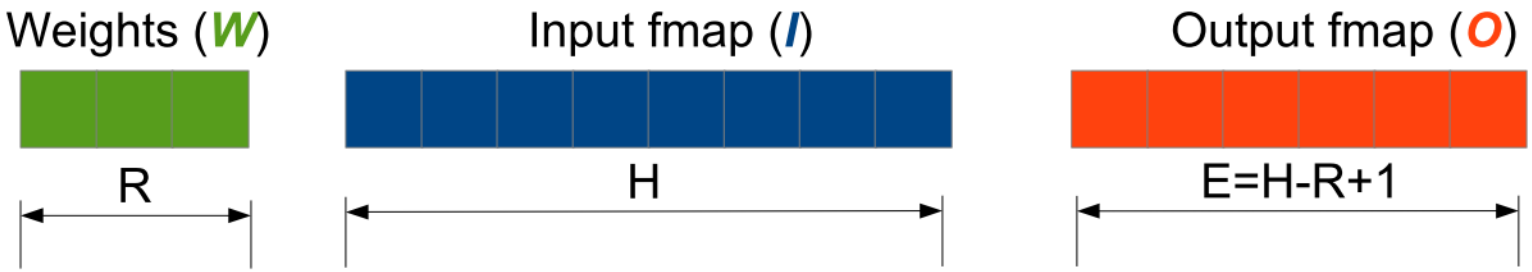


WS Example

- Cycle through input and output fmap (hold weights)

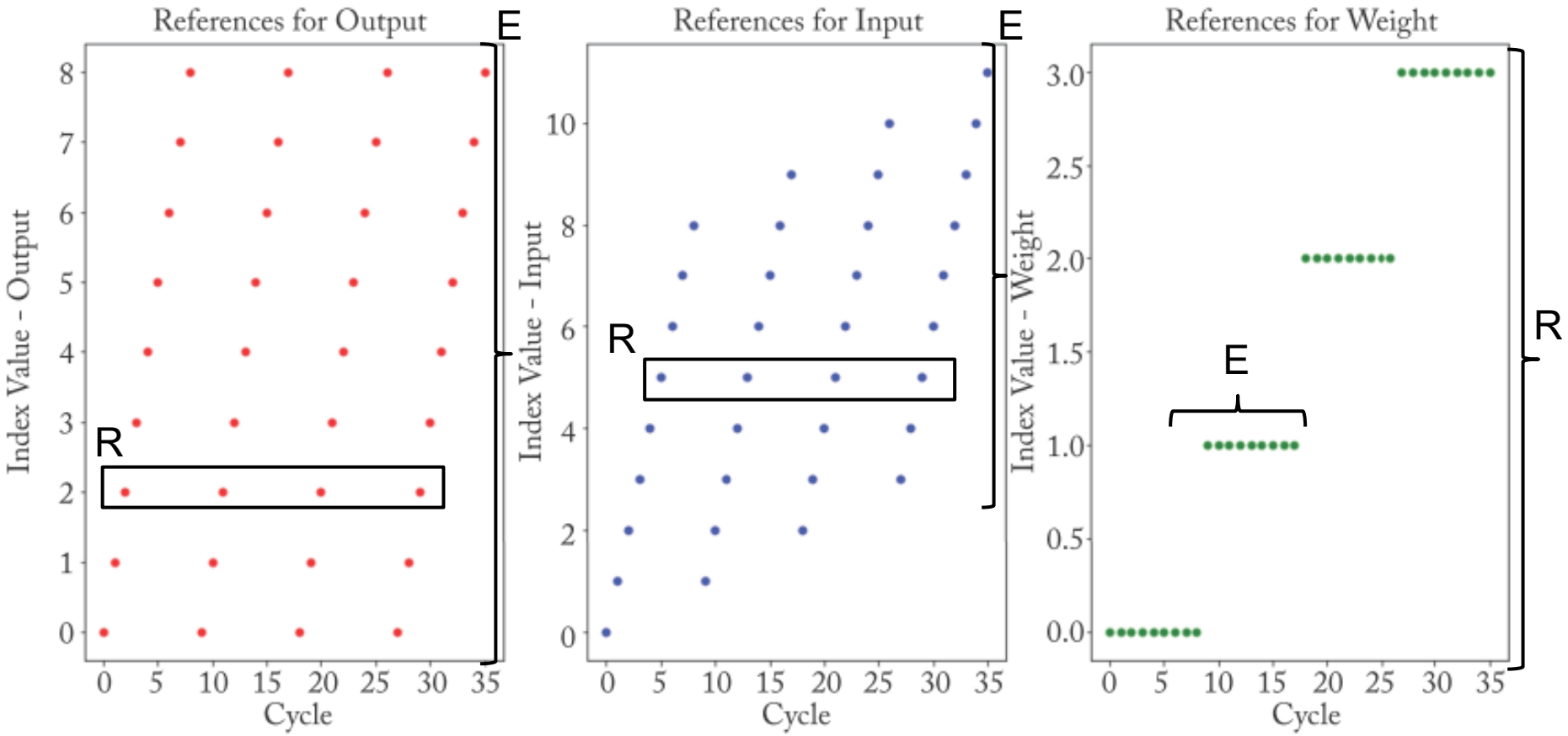


1-D Convolution – Weight Stationary



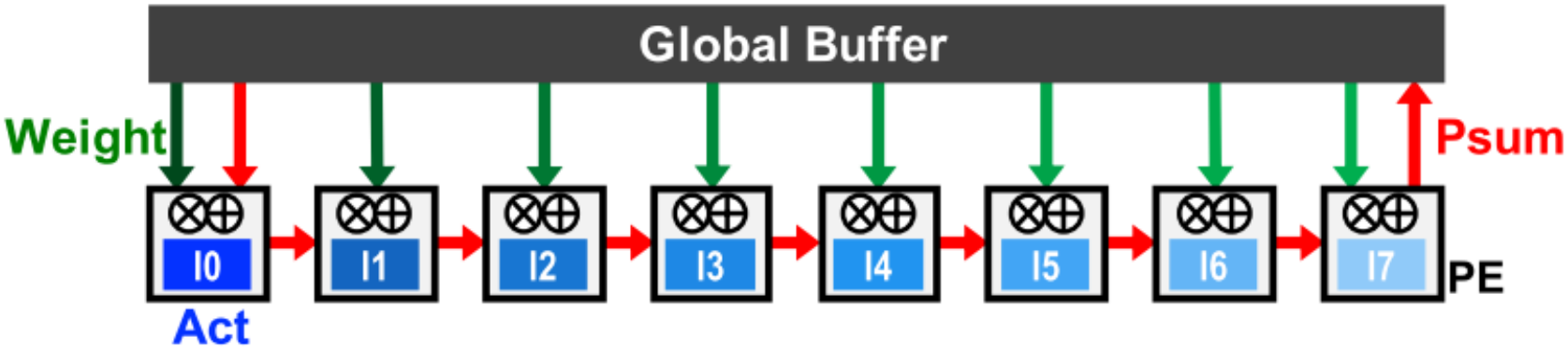

```
int W[R];
int I[H];
int O[E];

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

Space-time Diagram of WS



MACs	Weight Reads	Input Reads	Output Reads	Output Write
E x R	R	E x R	E x R	E x R

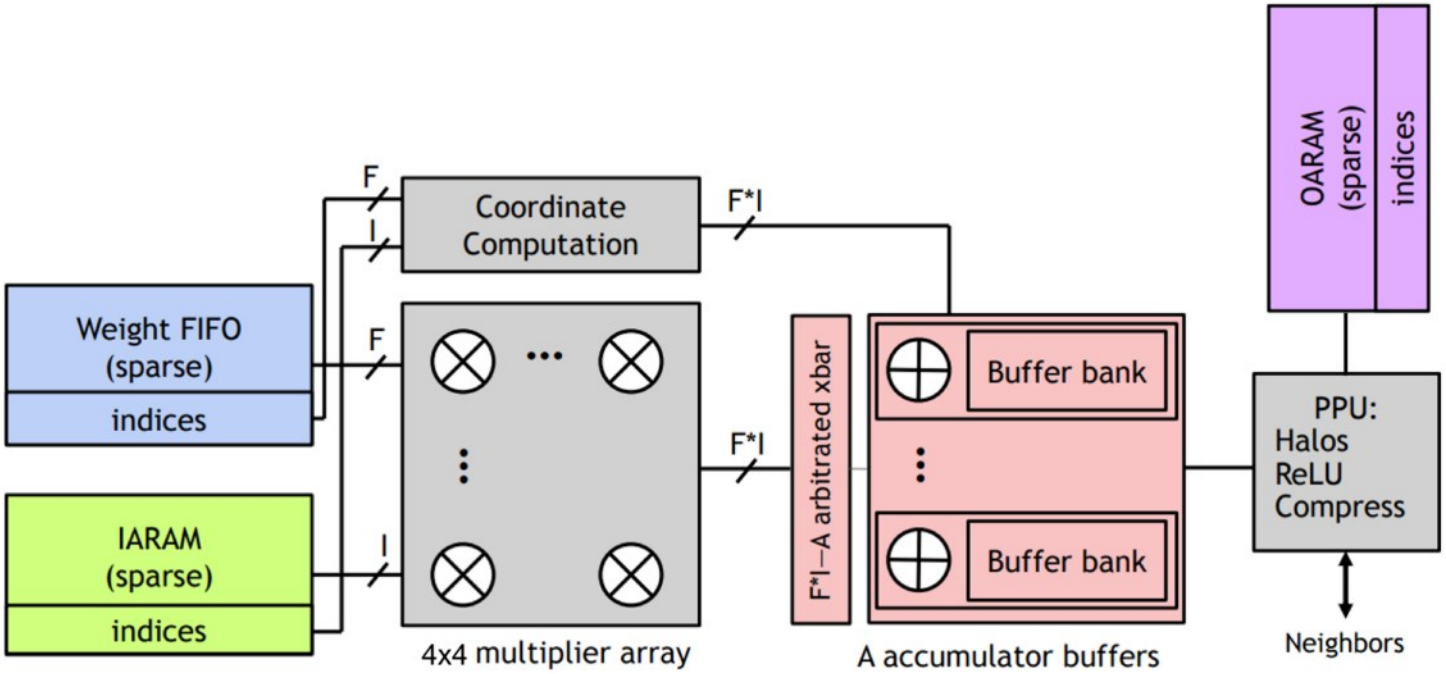
Input Stationary (IS)



- **Minimize activation** read energy consumption
  - maximize convolutional and fmap reuse of activations
- **Unicast weights** and **accumulate partial sums spatially** across the PE array

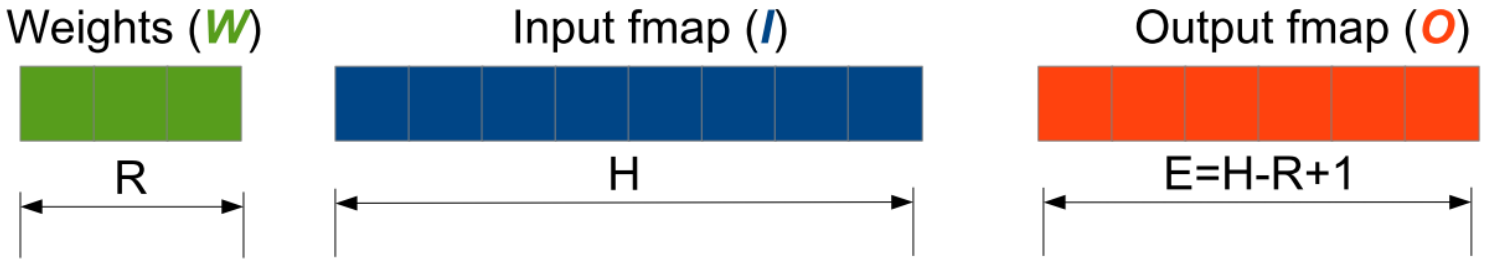
IS Example: SCNN

- Used for sparse CNNs
  - Sparse CNN is where many weights are zeros
  - Activations also have sparsity from ReLU



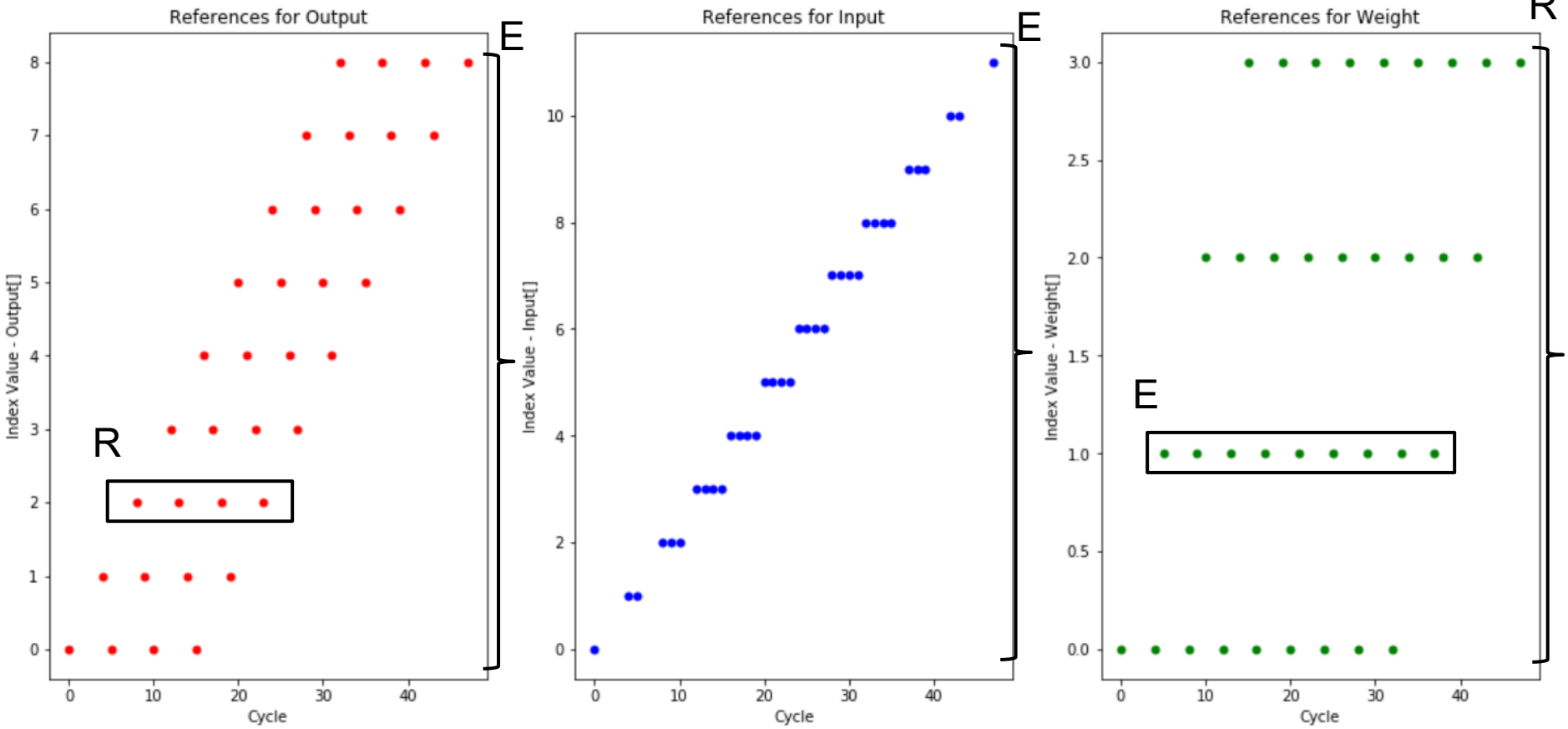
[Parashar et al., ISCA 2017]

1-D Convolution – Input Stationary




```
int W[R];
int I[H];
int O[E];

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

Space-time Diagram of IS



MACs	Weight Reads	Input Reads	Output Reads	Output Write
E x R	E x R	E	E x R	E x R

Summary – Minimum Costs

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

Assume: W~=E

Summary of DNN Dataflows

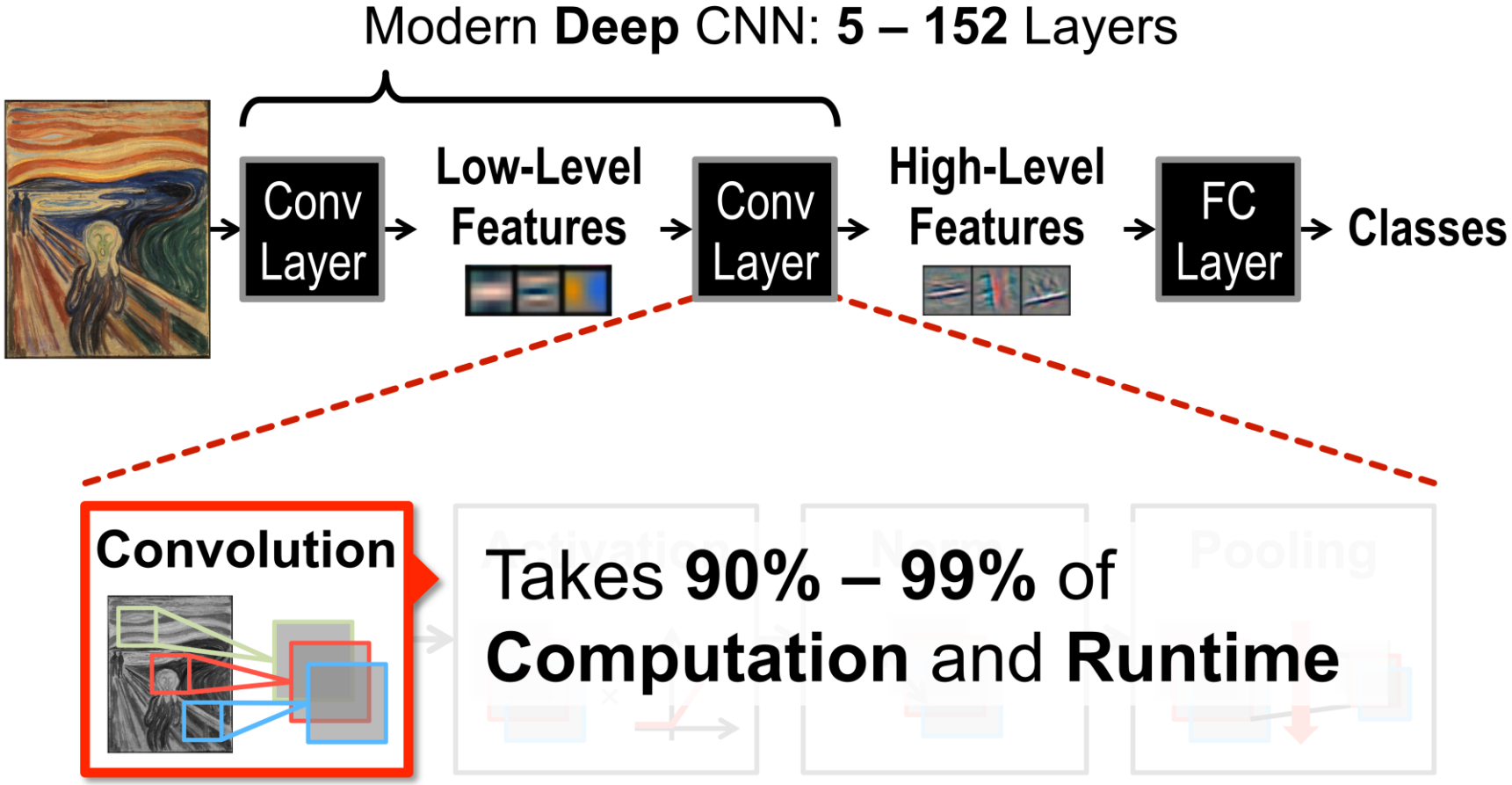
- Minimizing **data movement** is the key to achieving high **energy efficiency** for DNN accelerators

- Dataflow taxonomy:
  - **Output Stationary:** minimize movement of psums
  - **Weight Stationary:** minimize movement of weights
  - **Input Stationary:** minimize movement of inputs

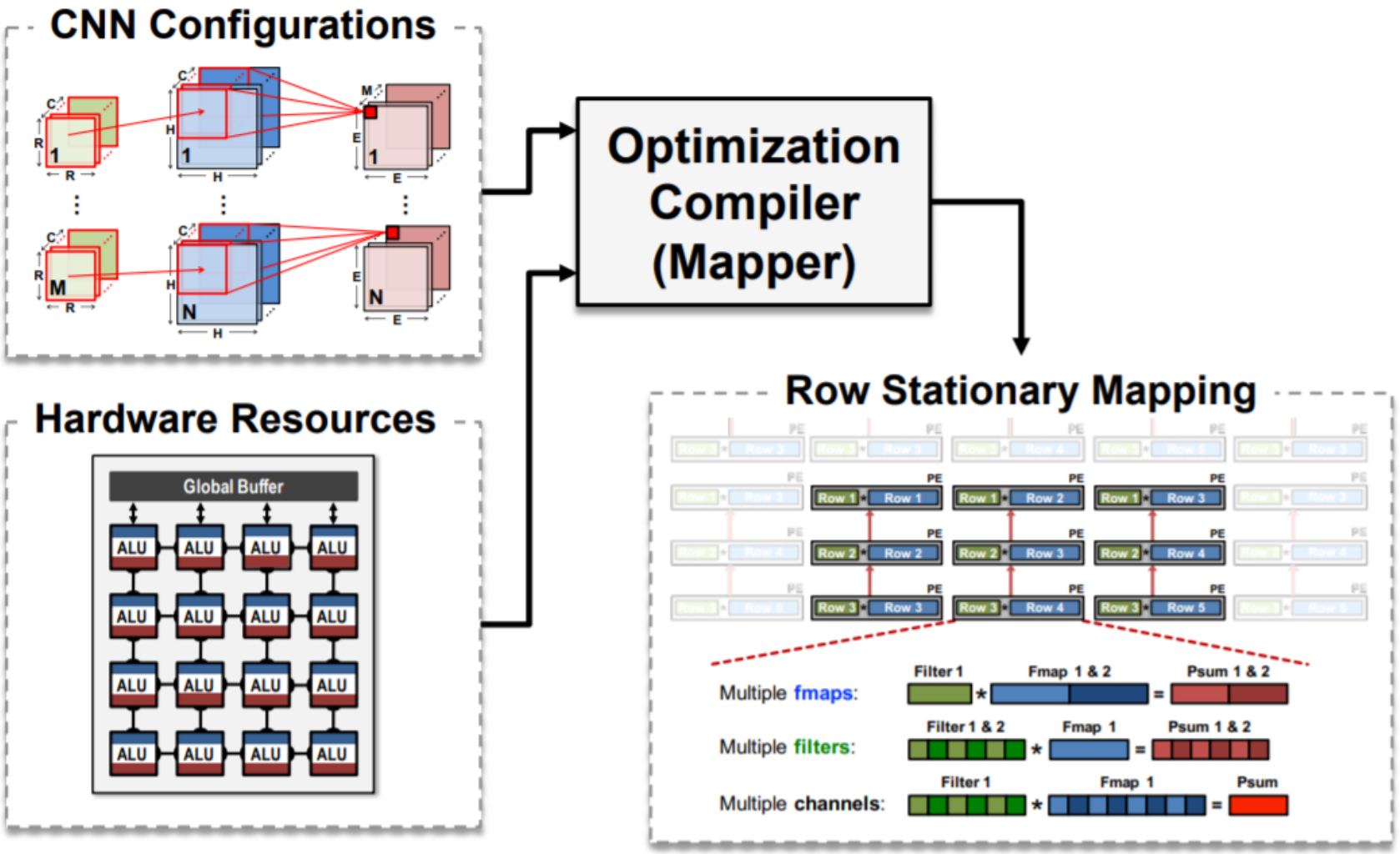
- **Loop nest** provides a compact way to describe various properties of a dataflow, e.g., data tiling in multi-level storage and spatial processing.

DNN Architecture Design with Row Stationary

Efficient convolution is important

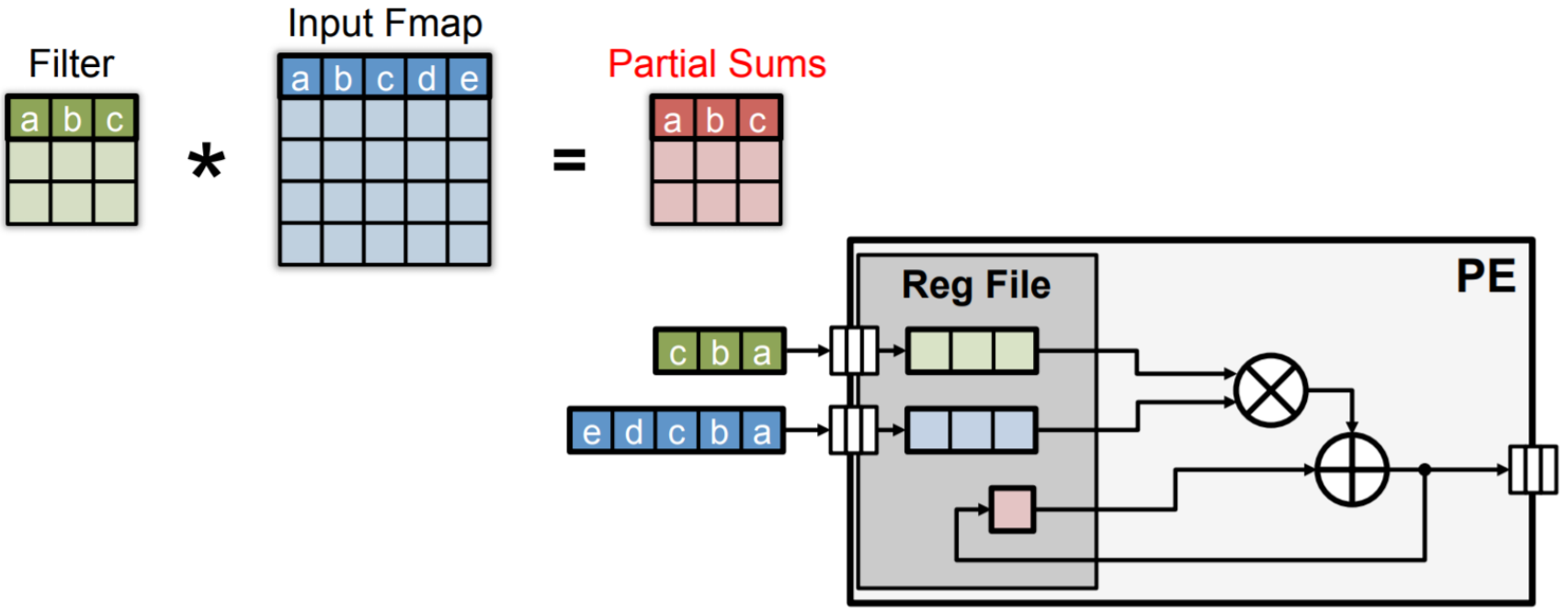


Static Resource Mapping

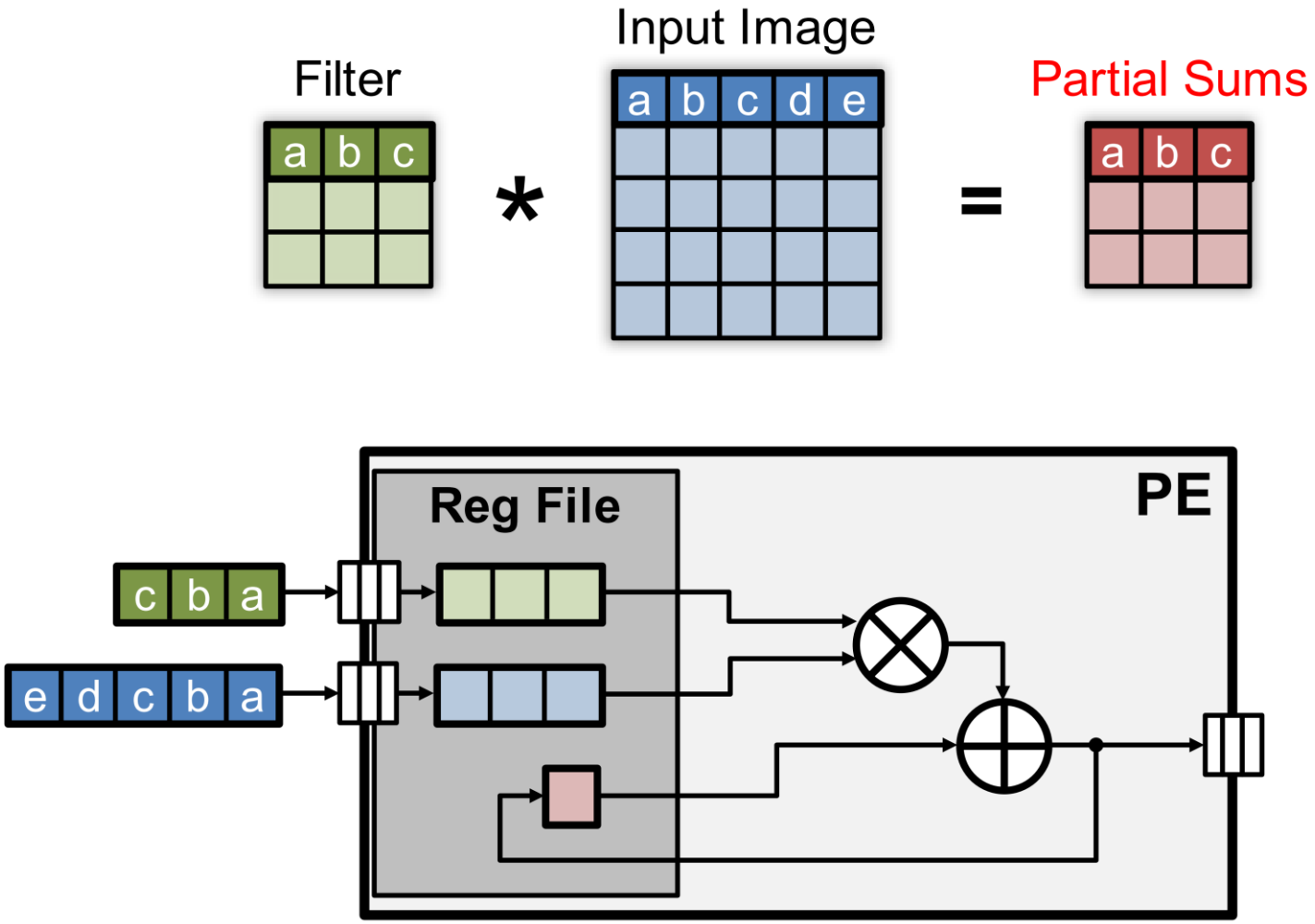


Row Stationary

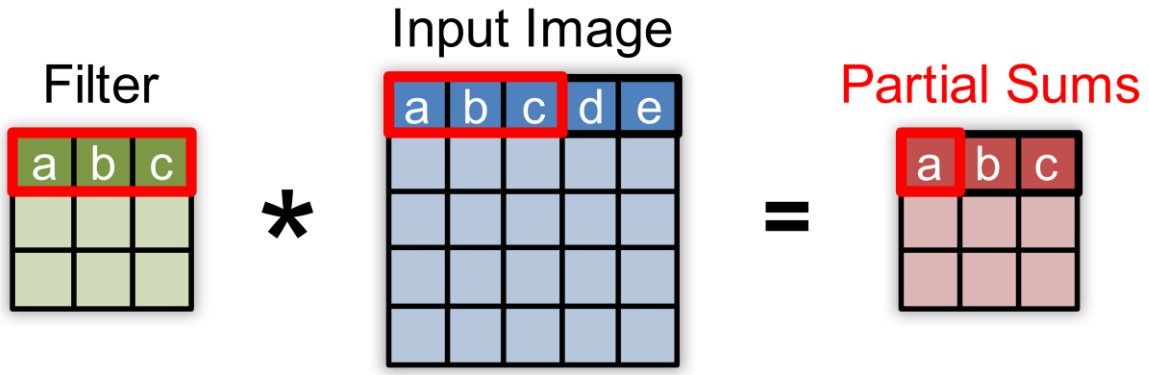
- Keep as much related to the same filter row cached… Across PEs
  - Filter weights, input, output…
- Not much reuse in a PE
  - Weight stationary if filter row fits in register file

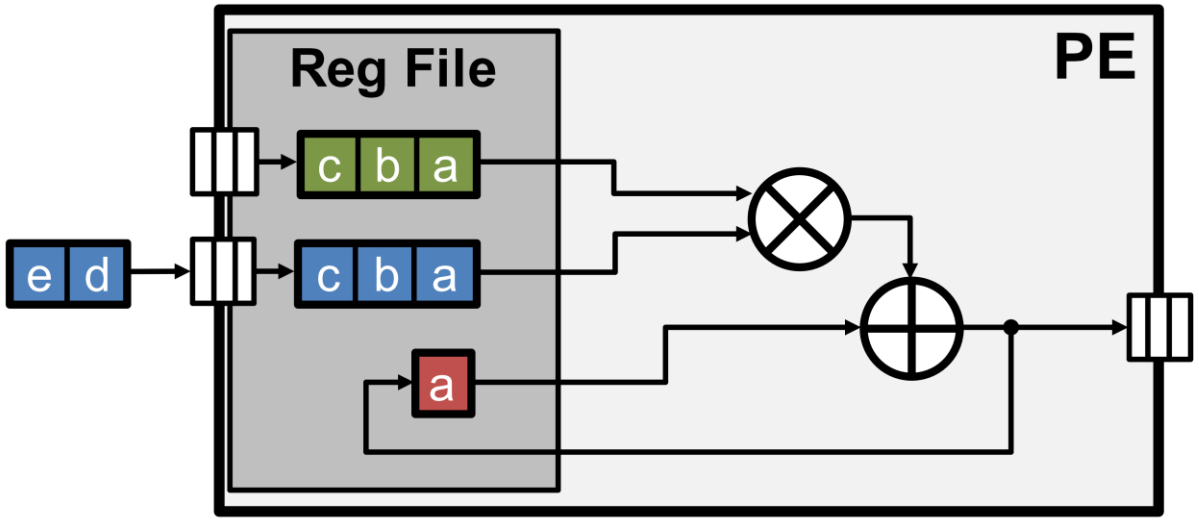


1D Row Convolution in PE (1/5)

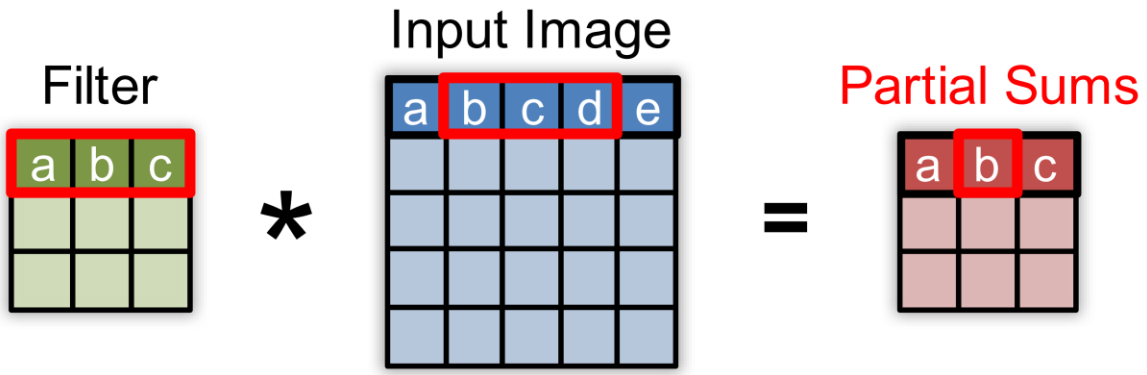


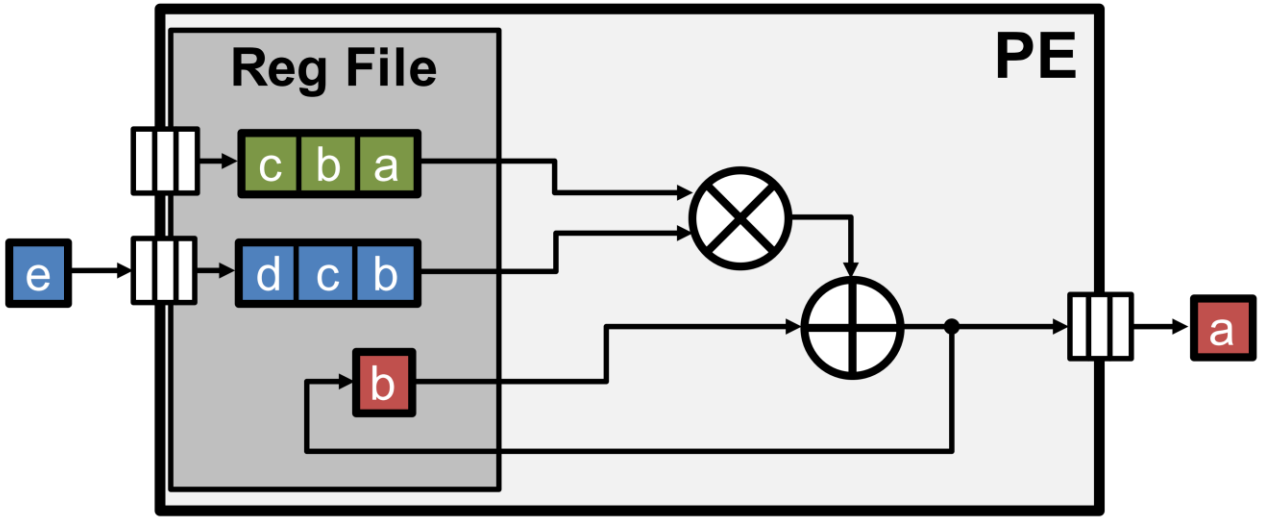
1D Row Convolution in PE (2/5)



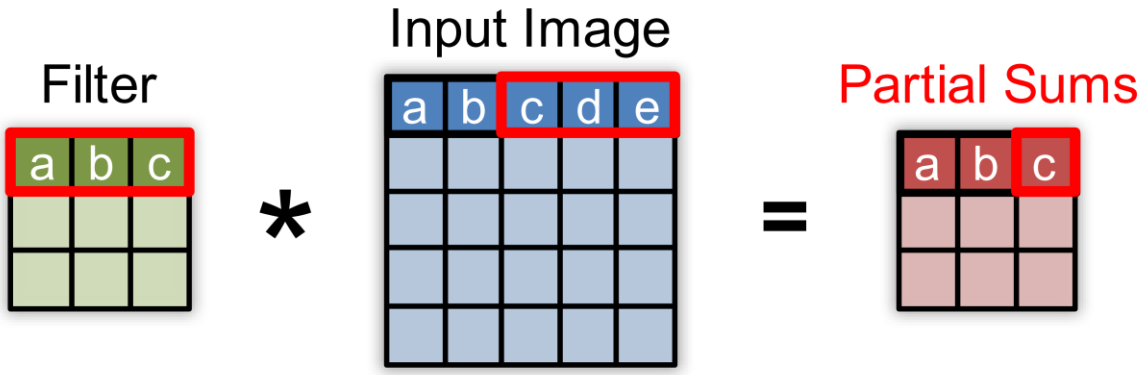


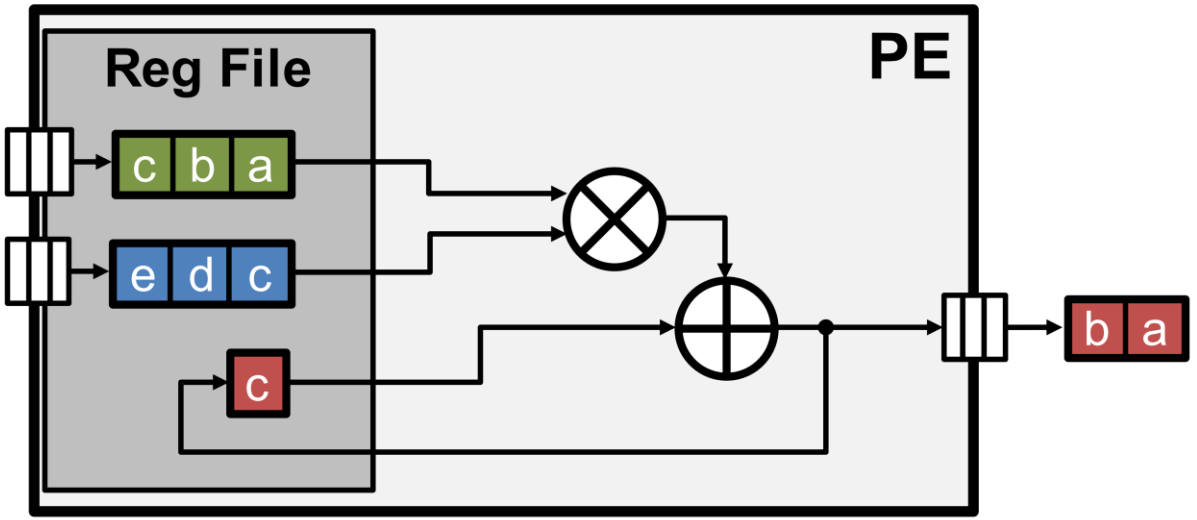
1D Row Convolution in PE (3/5)





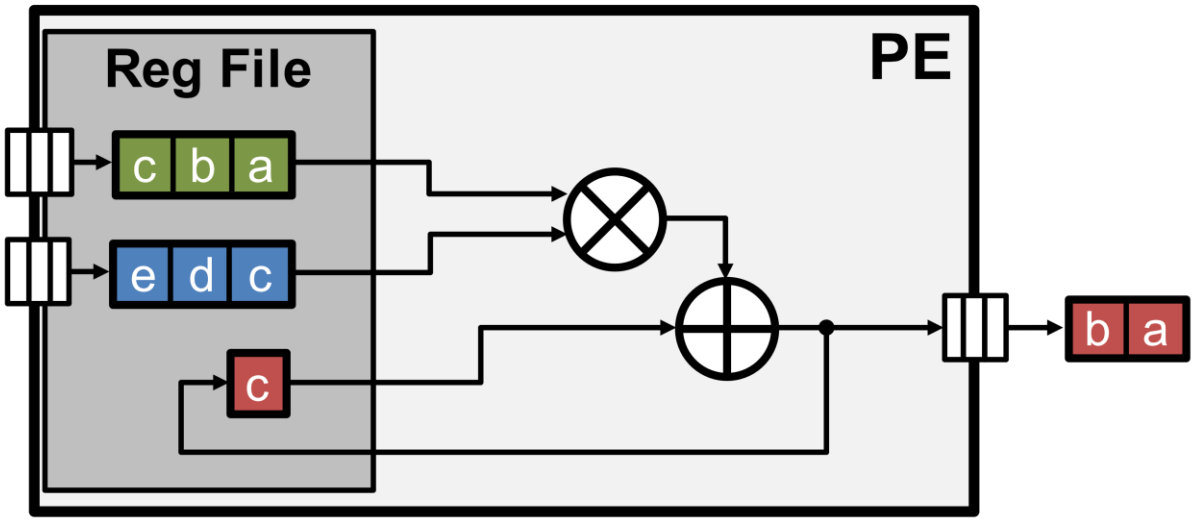
1D Row Convolution in PE (4/5)



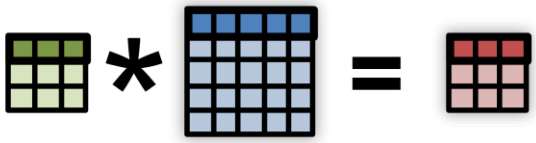


1D Row Convolution in PE (5/5)

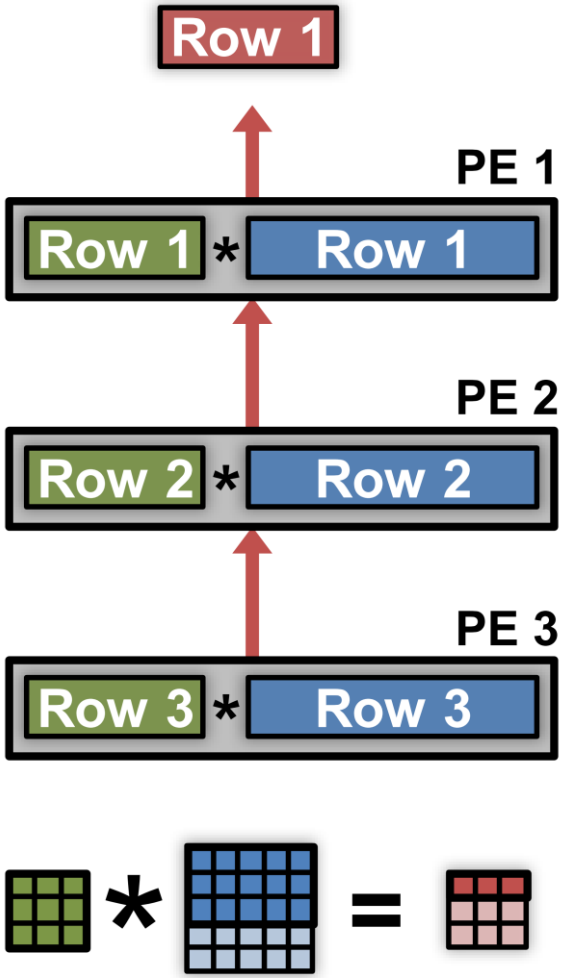
- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **image** sliding window in RF
- Maximize row **psum accumulation** in RF



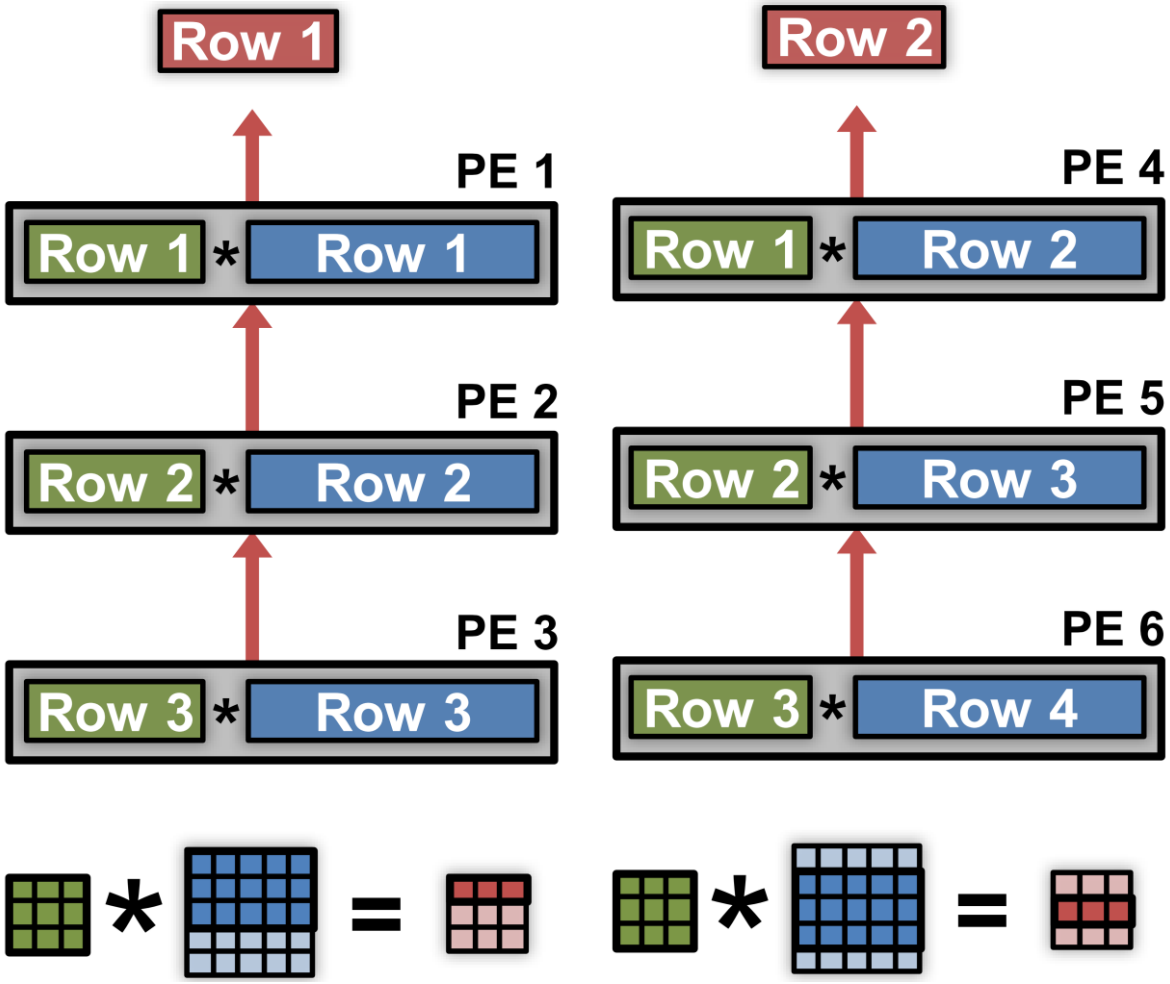
2D Row Convolution in PE (1/4)



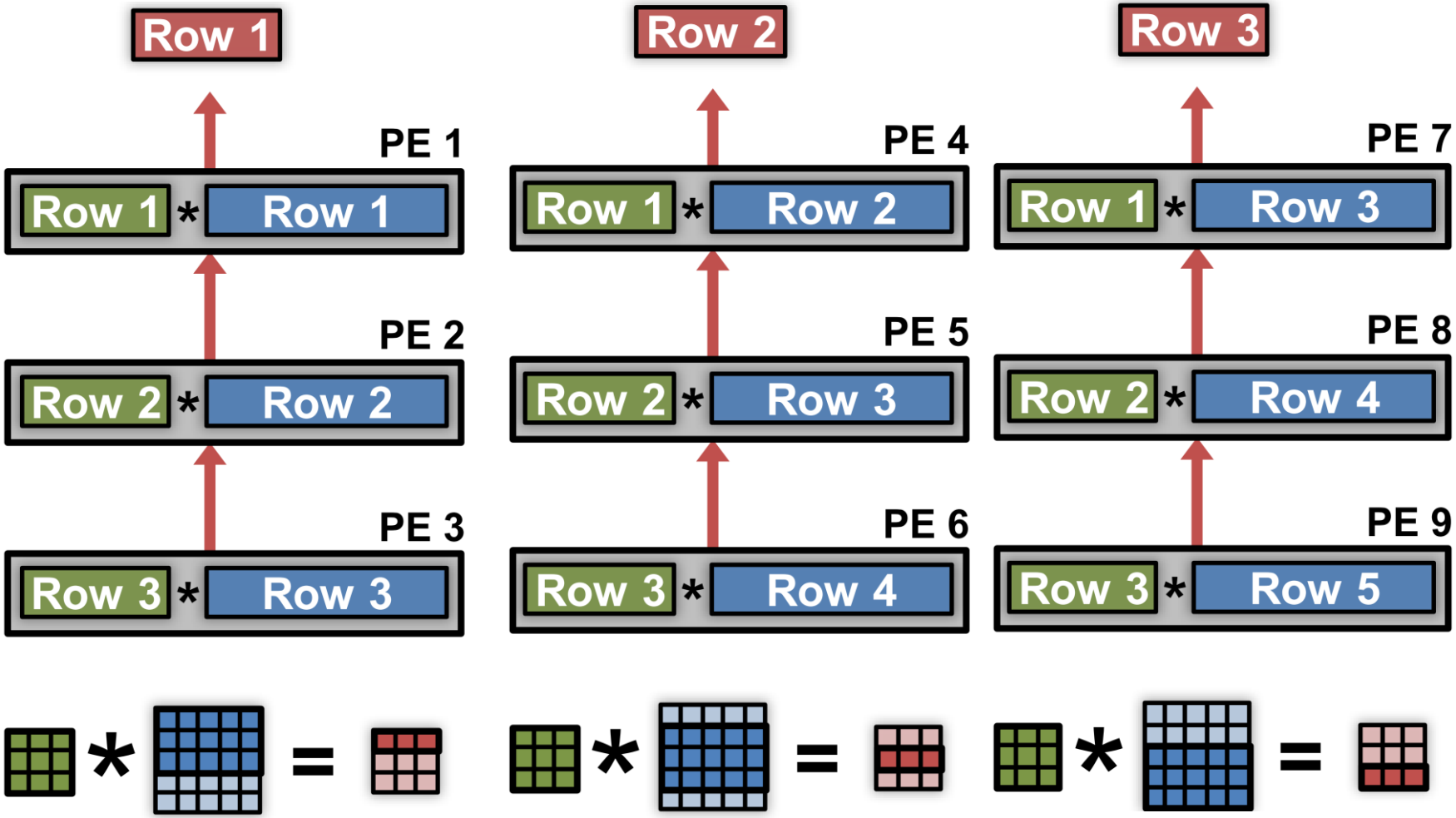
2D Row Convolution in PE (2/4)



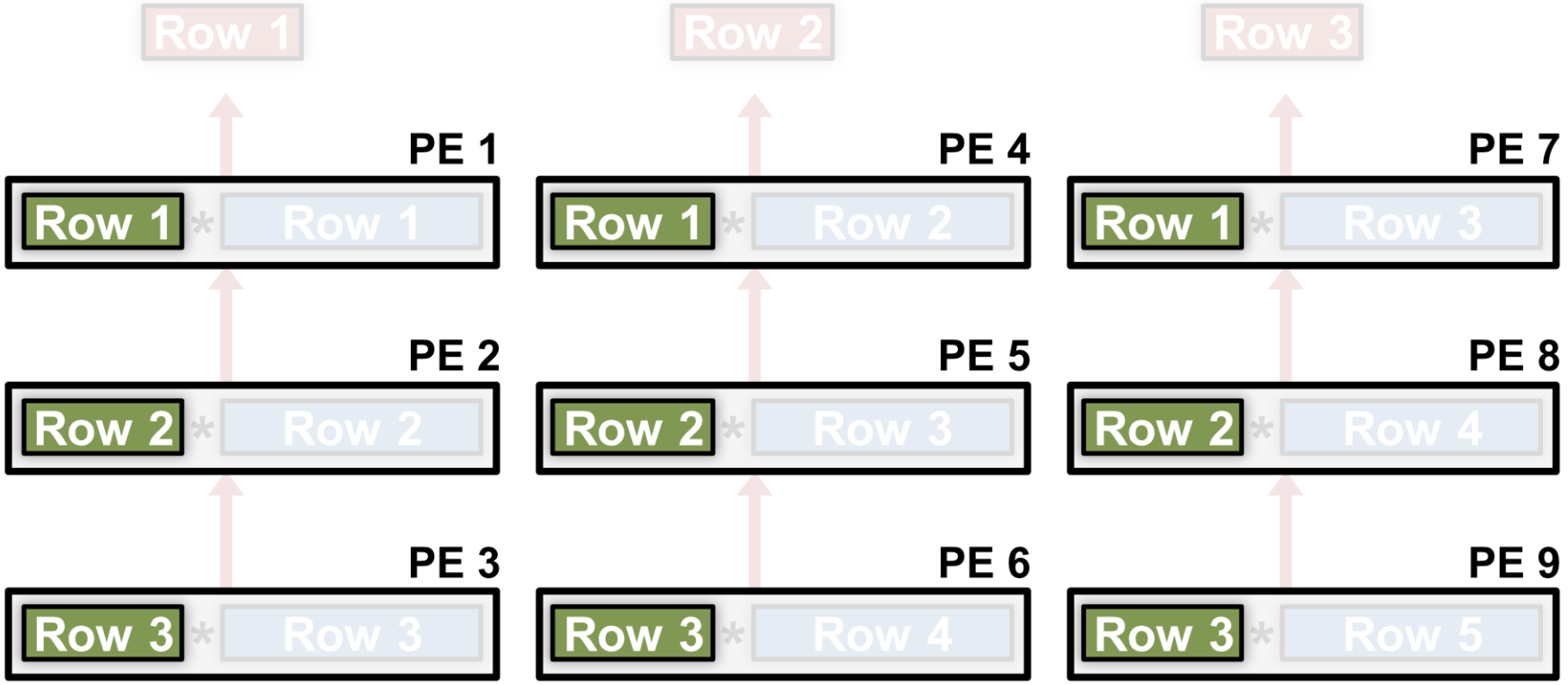
2D Row Convolution in PE (3/4)



2D Row Convolution in PE (4/4)



Convolutional Reuse Maximized



Filter rows are reused across PEs **horizontally**

Convolutional Reuse Maximized

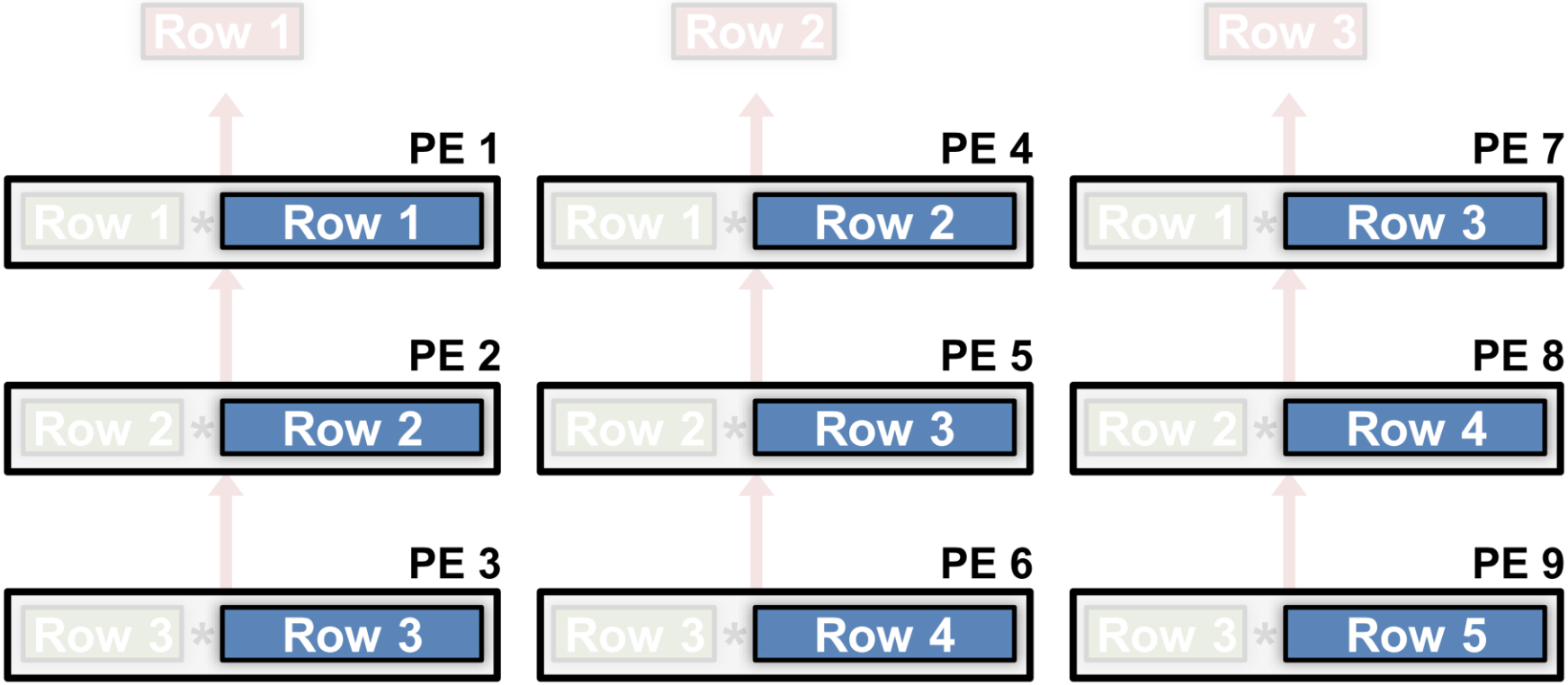
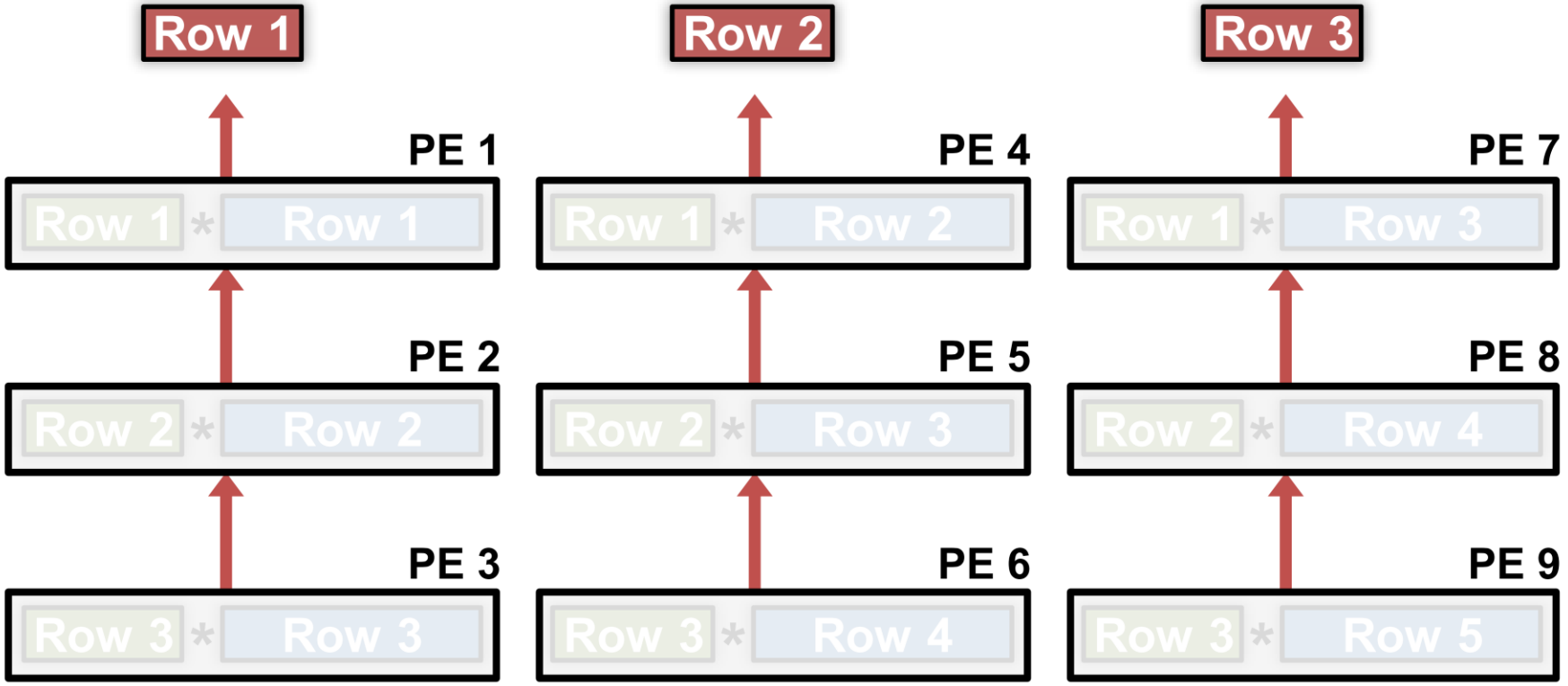


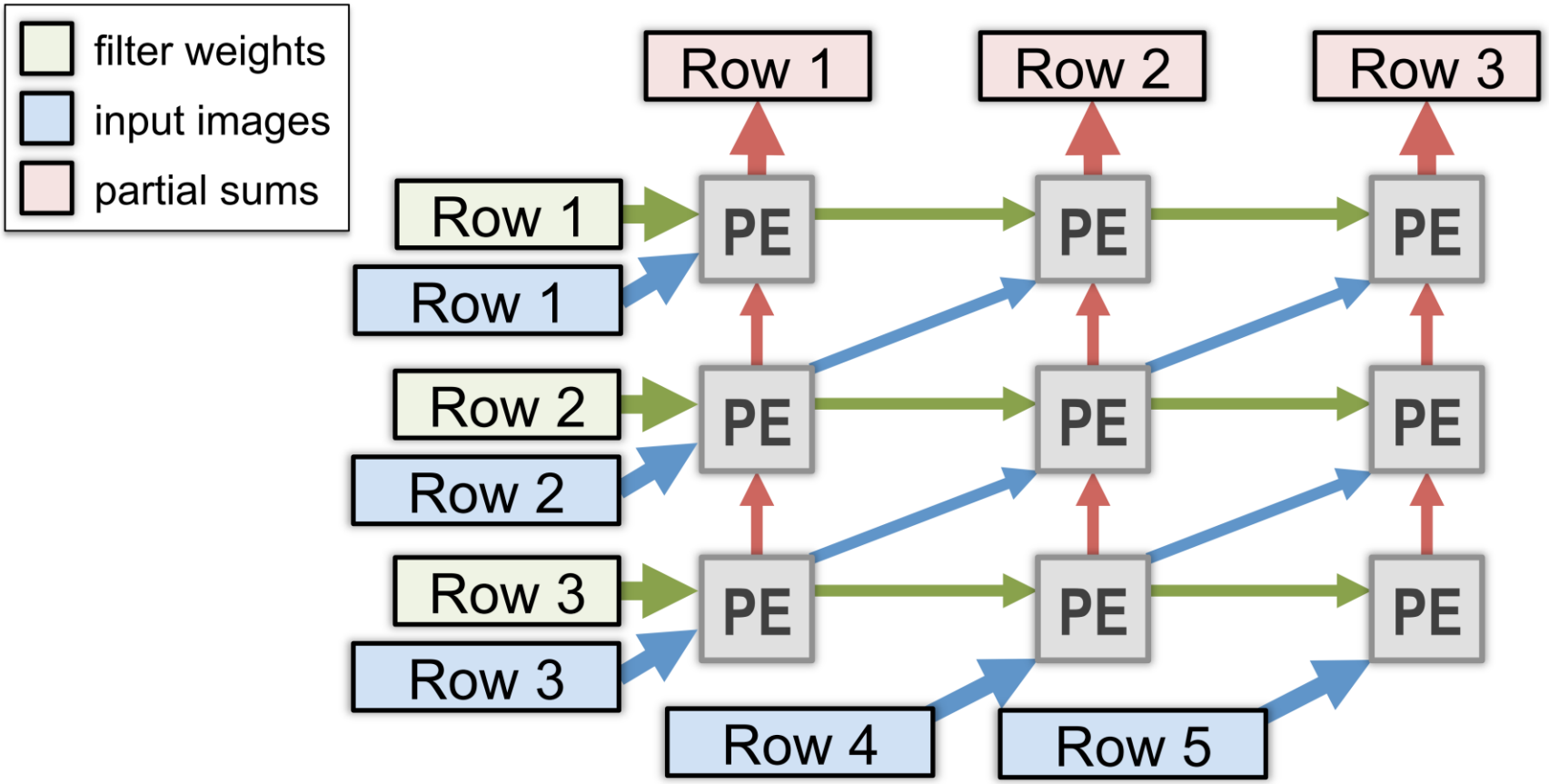
Image rows are reused across PEs **diagonally**

Maximize 2D Accumulation in PE Array



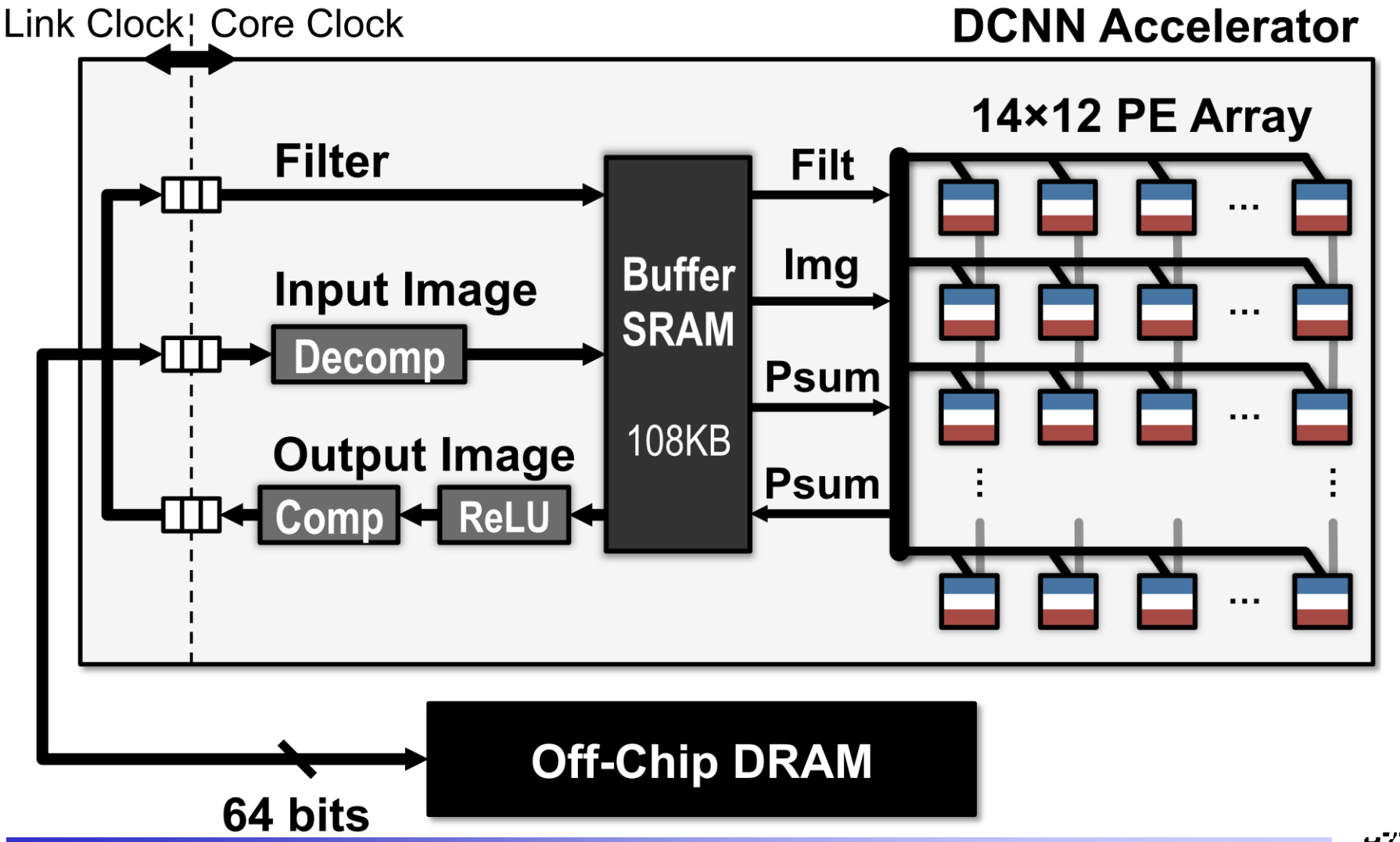
Partial sums accumulate across PEs **vertically**

Convolutional Reuse within PE Array



Mapping rows from **multiple channels** and/or **multiple filter/images** to each PE results in even more **reuse**

The DCNN Accelerator Architecture

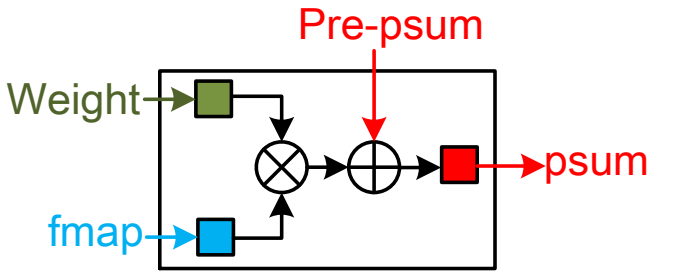


WS vs. OS vs. RS

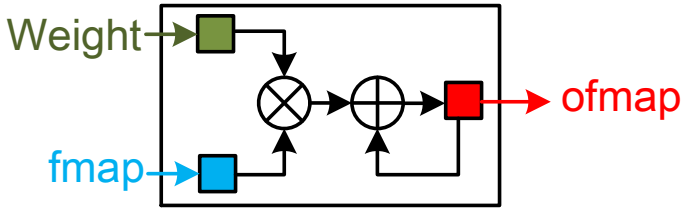
	WS	OS	RS
Timing	4 ns		
Area	9,488,890 um^2	11,359,573 um^2	**9,425,788 um^2**
Total cycles	4,512,512	4,600,005	**3,747,732**
Energy cost	3.383768 mJ	**1.805017 mJ**	2.05629 mJ

- Model: VGG 19
- Fixed
  - Image size
  - FPS
  - Clock rate
  - # PEs

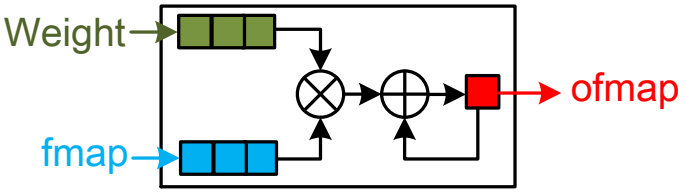
WS



OS

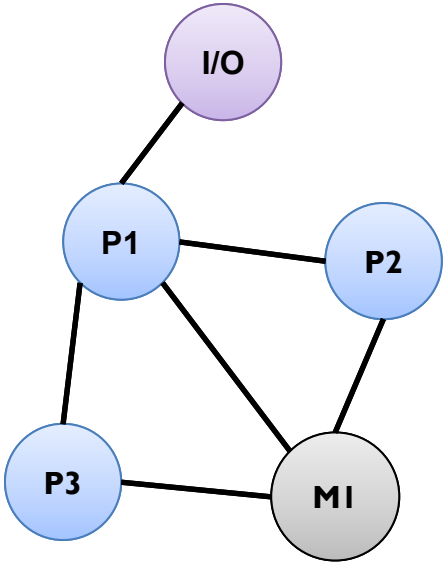


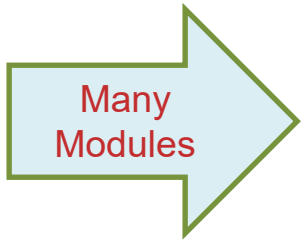
RS

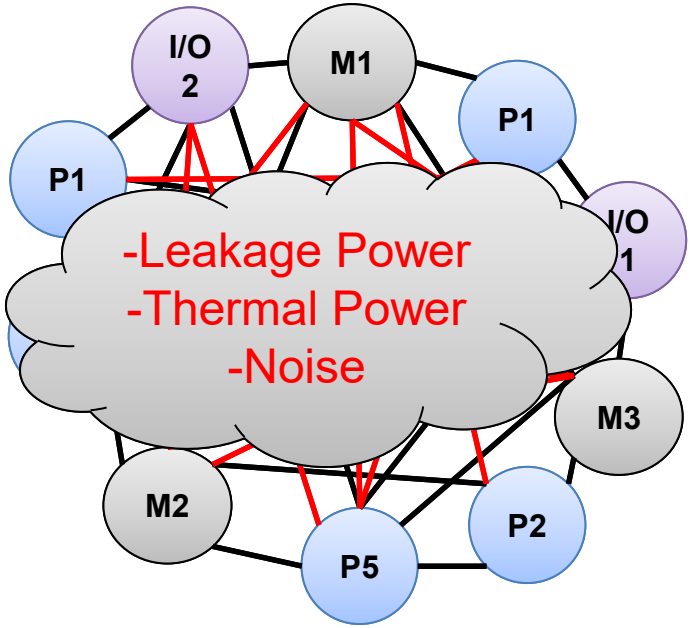


Fundamental of Network-on-Chip (NoC) interconnection

On-chip Interconnection Types

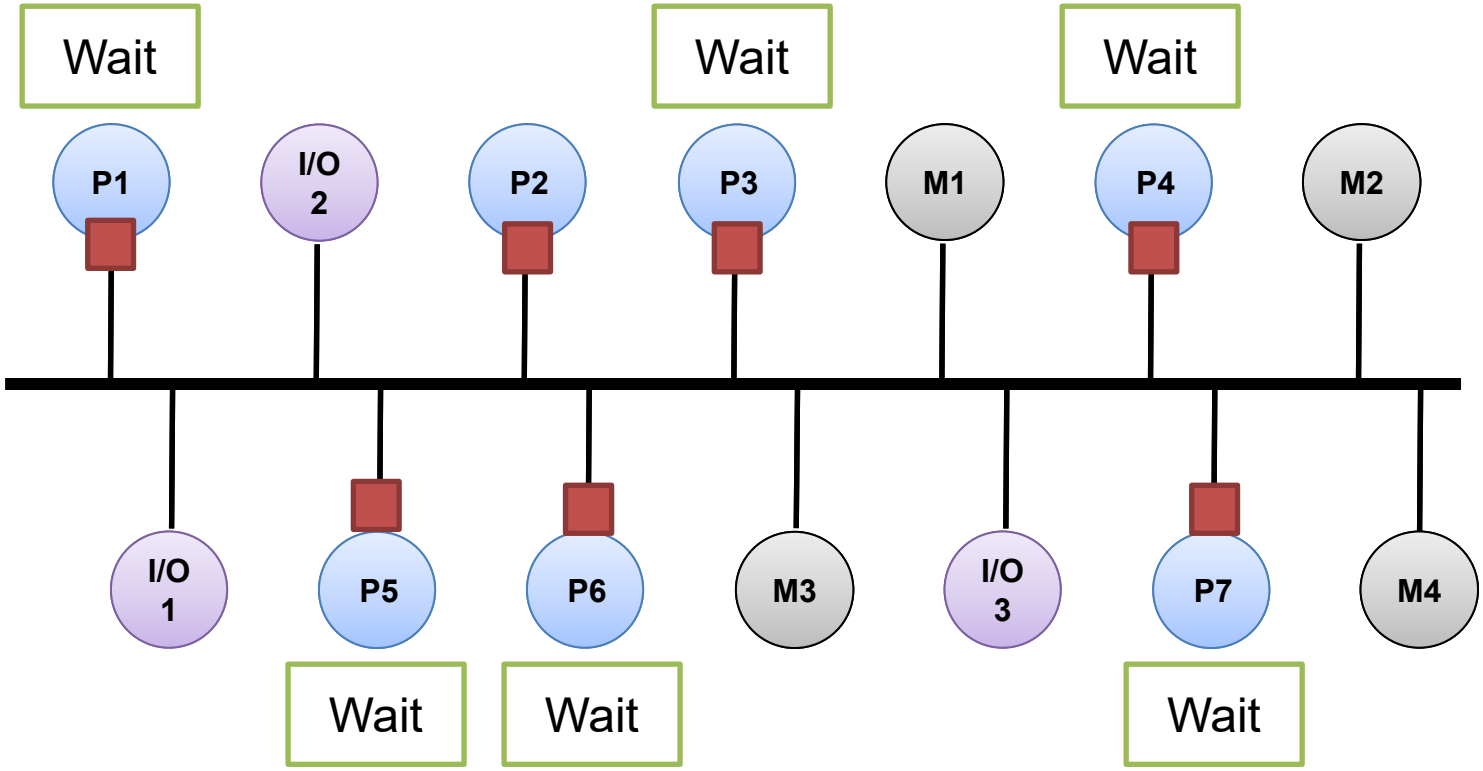






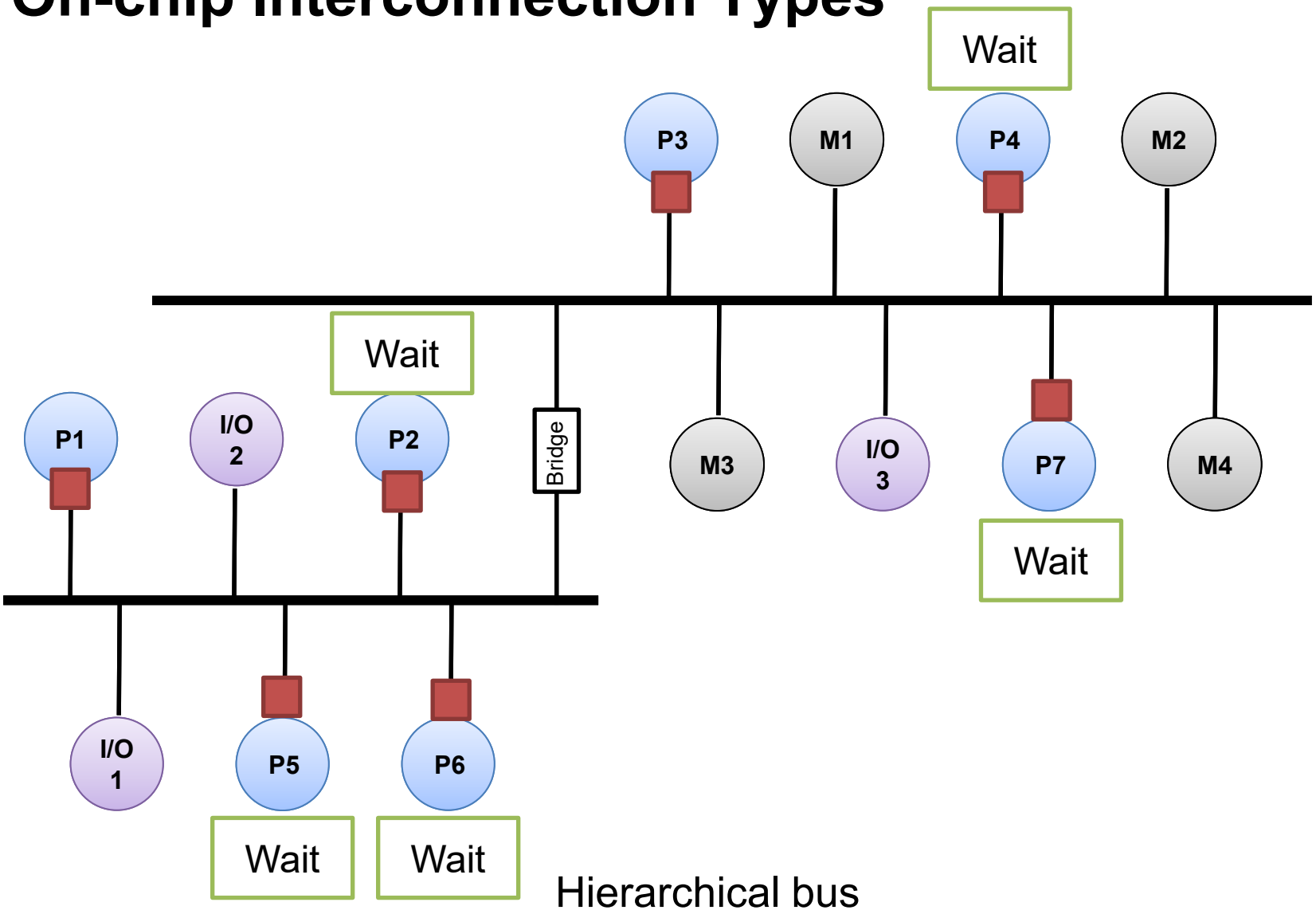
Point-to-Point

On-chip Interconnection Types



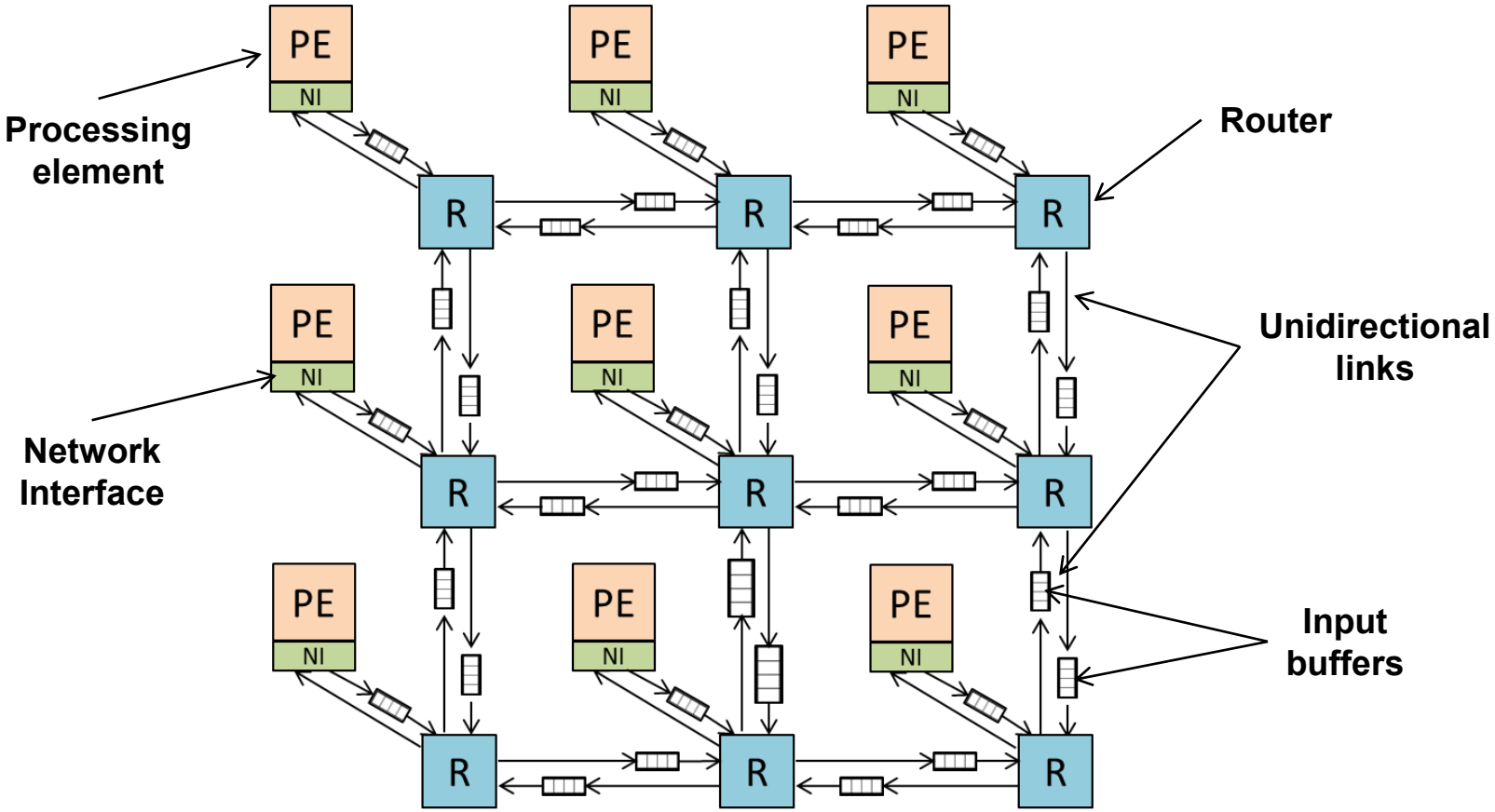
Shared bus

On-chip Interconnection Types



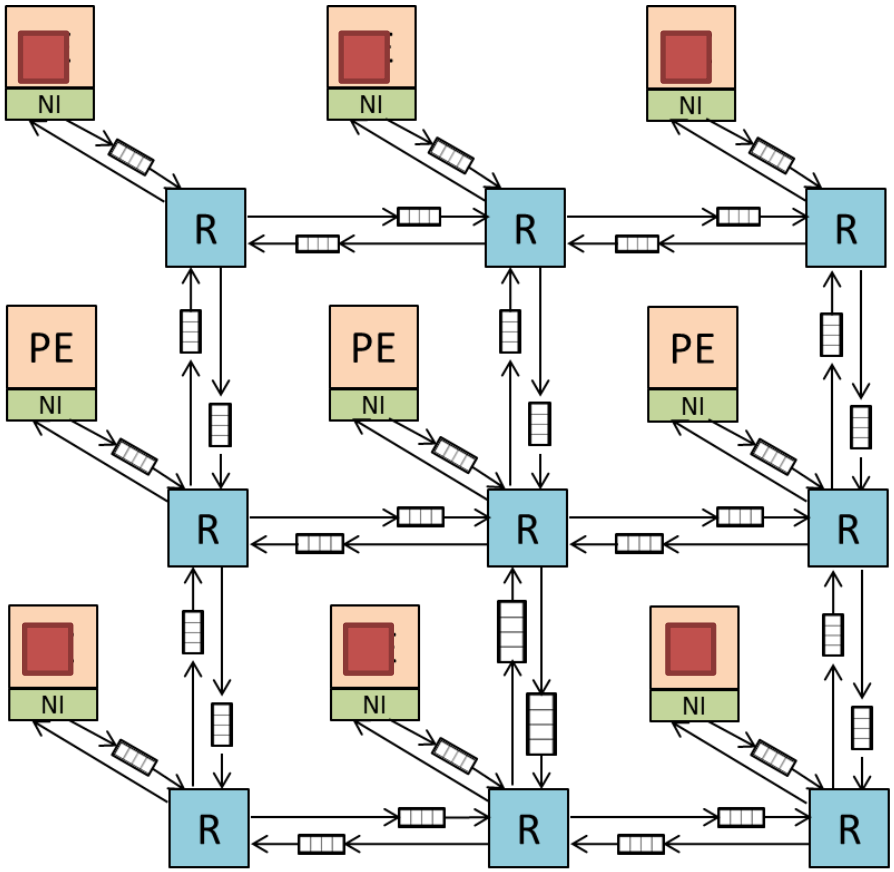
Hierarchical bus

On-chip Interconnection Types



Network-on-Chip -> our main topic in this lecture.

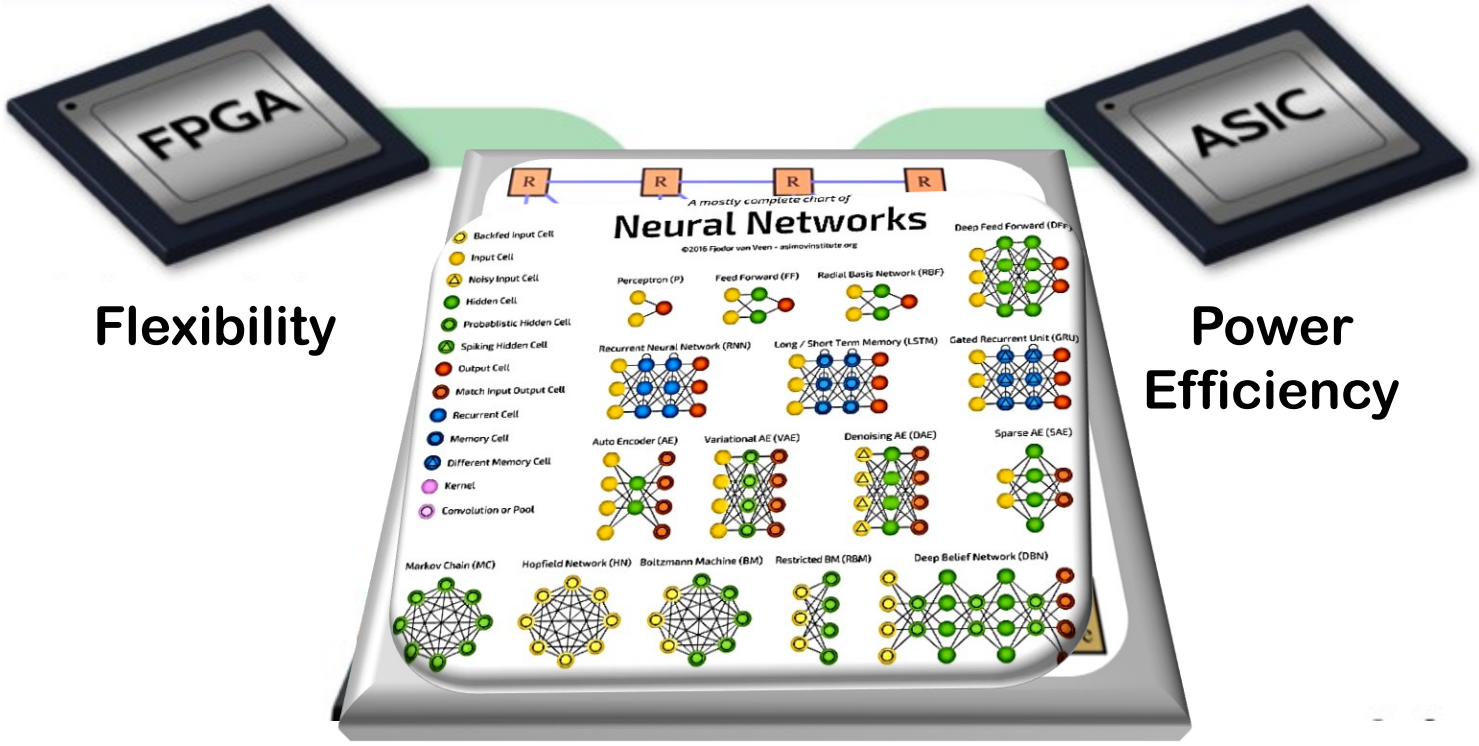
On-chip Interconnection Types



Network-on-Chip -> our main topic in this lecture

NoC is a good solution in AI and multi-core market!

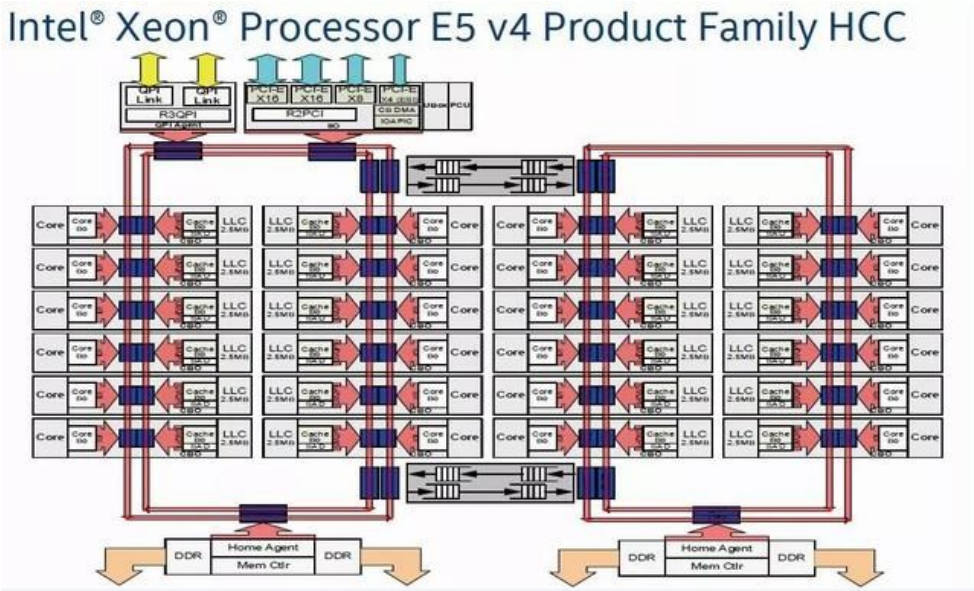
- NoC-based accelerators :
  - High flexibility from NoC technology
  - High power efficiency from ASIC PE

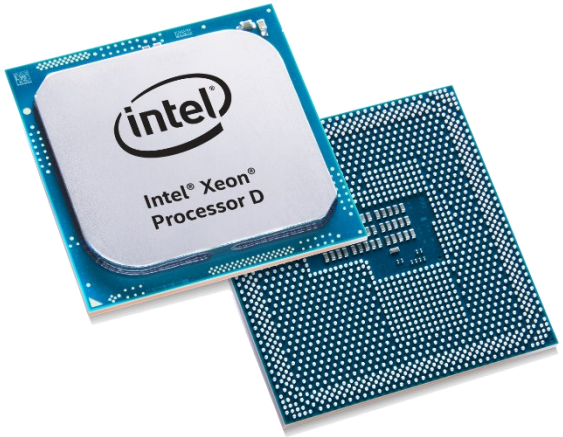


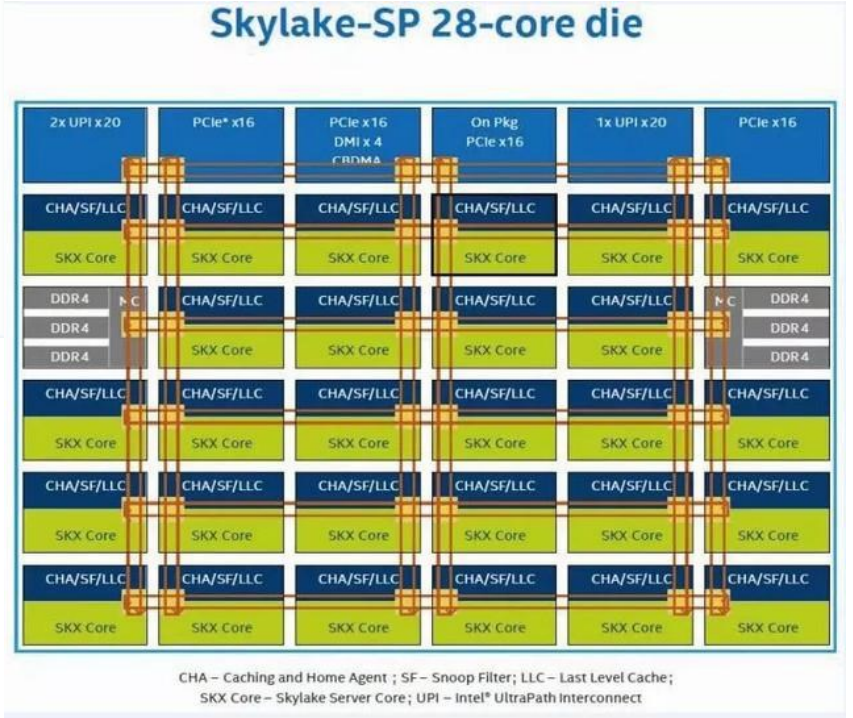
Commercial Product 1: AMD-Xilinx Versal Series FPGA, 2018

- Adaptive Compute Acceleration Platform (ACAP)
  - New name of “programmable logic”
  - Connect each SoC component via NoC interconnection
  - Target the 5G and AI application

Commercial Product 2: Intel Skylake Architecture, 2018

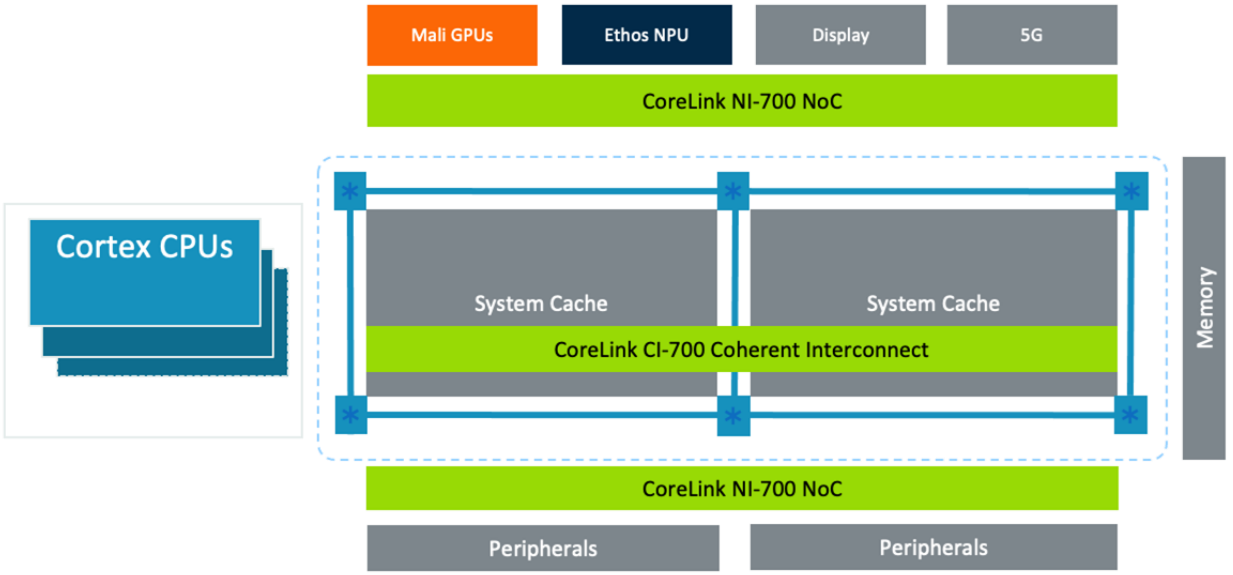






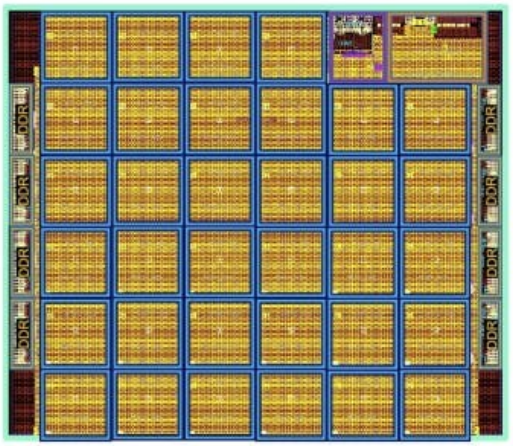
Commercial Product 3: ARM CoreLink NI-700, 2021

- ARM CoreLink NI-700,
  - A new flexible packetized network-on-chip interconnect for both high-bandwidth accelerators
  - Scalable and highly configurable network-on-chip (NoC)
  - all the AMBA transactions are converted to a packetized format
    - Reduce the wire count by 30% on average

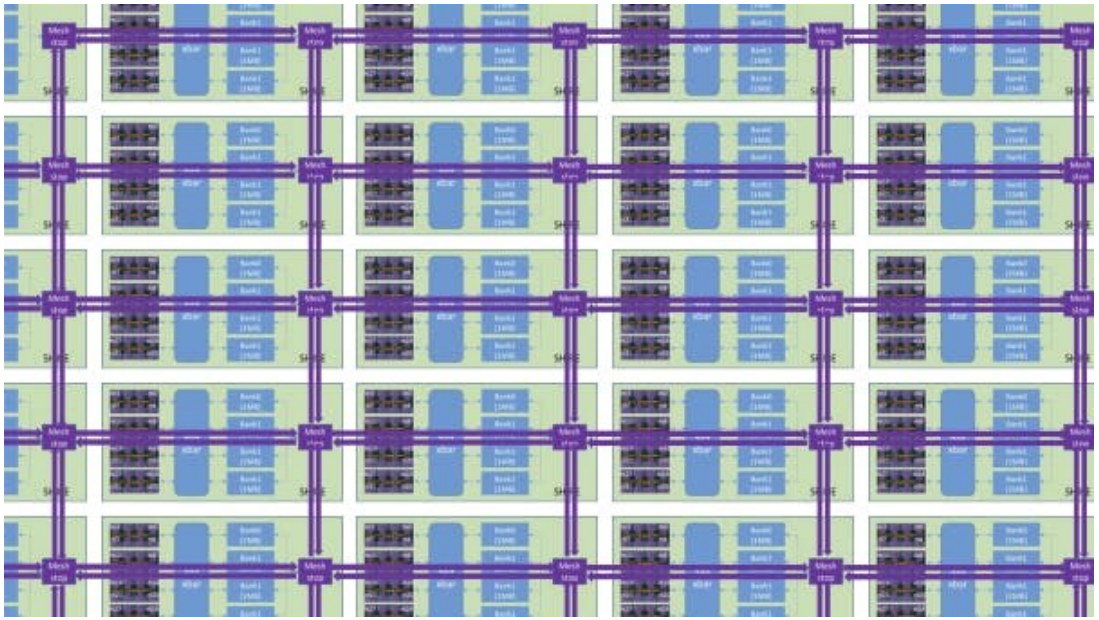


Commercial Product 4: Esperanto ET-SoC-1, 2021

- Esperanto ET-SoC-1
  - 1093 RISC-V AI Accelerator
  - The first AI processor in Esperanto Inc.

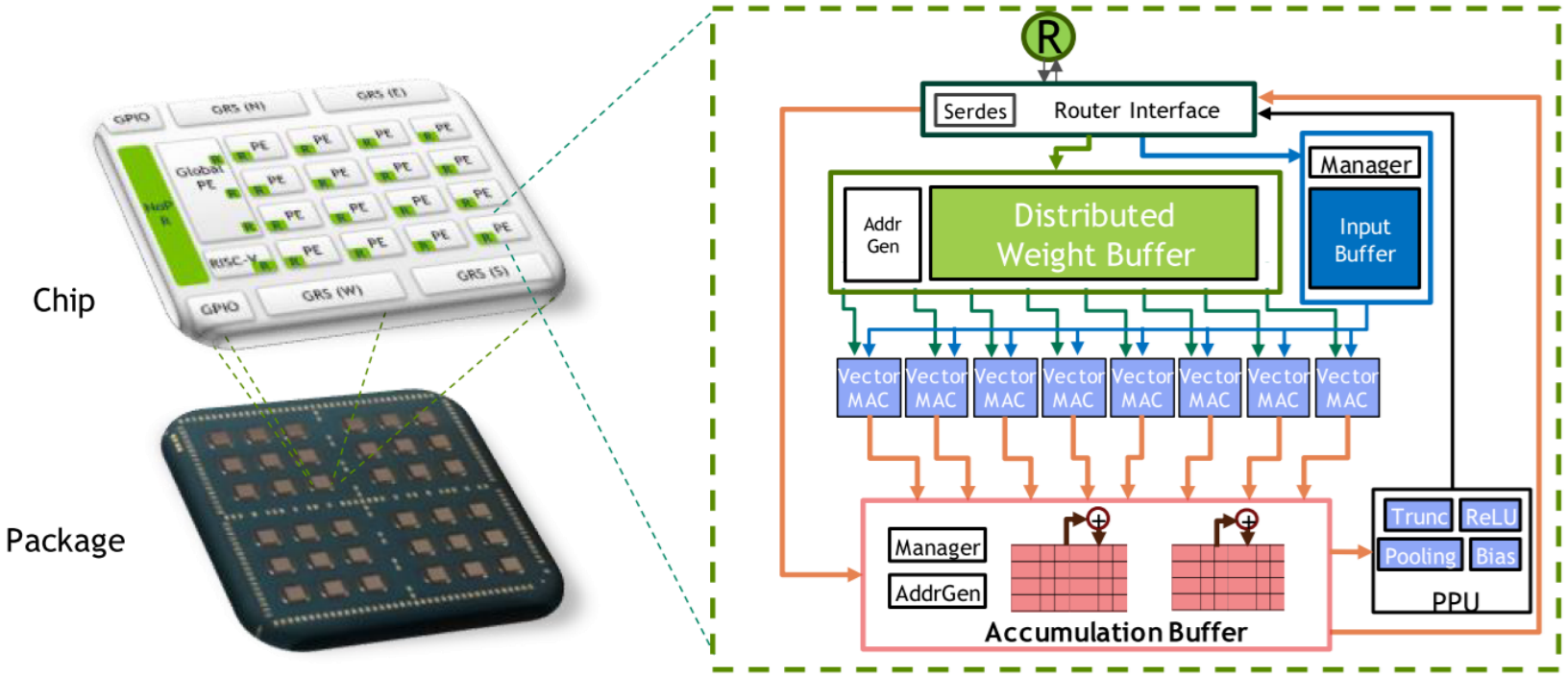


- TSMC 7nm
- 4 ET-Maxion (superscalar out-of-order core)
- 1089 ET-Minion (in-order multithreaded core)
- 32GB DRAM
- 137GB/sec memory bandwidth
- 256-bit wide interface



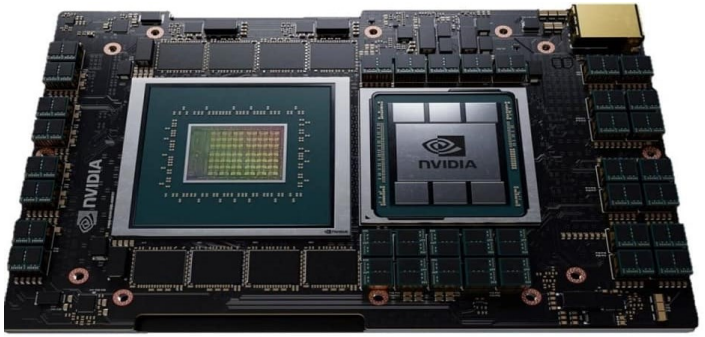
Commercial Product 5: NVIDIA Muti-Chip-Module (MCM), 2019

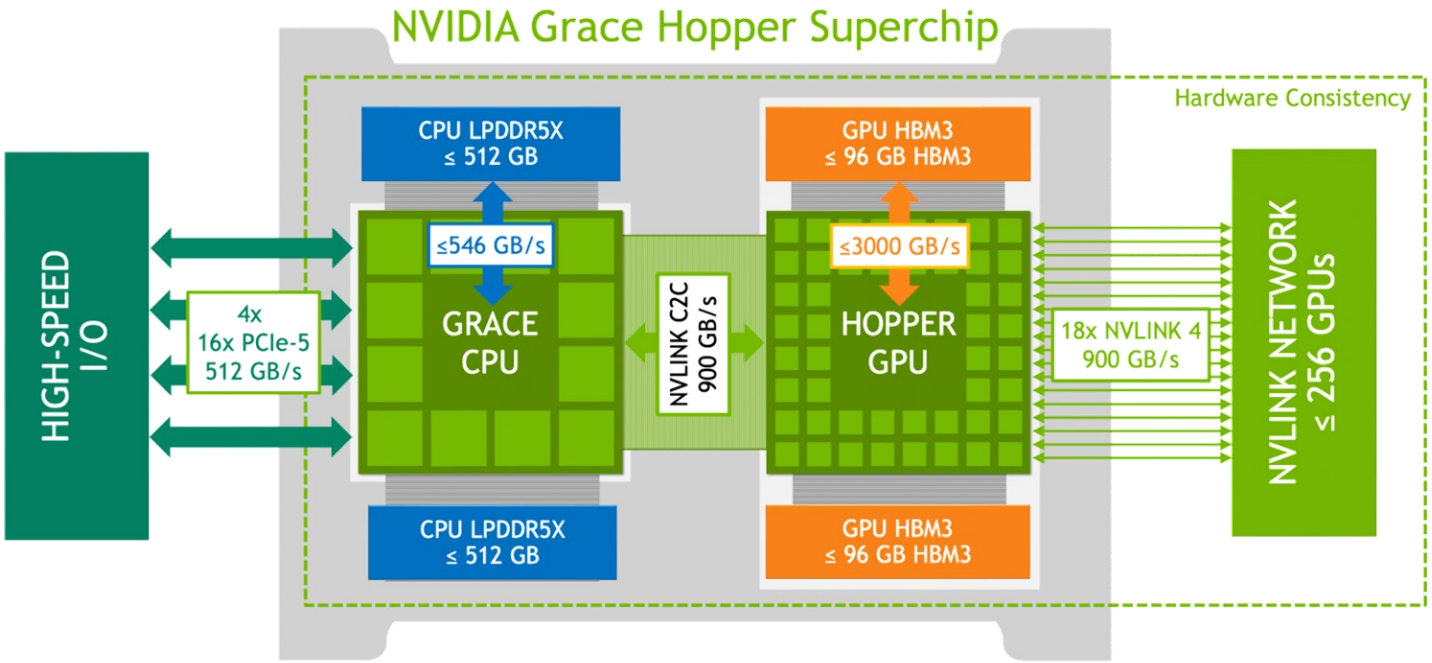
- Multi-Chip-Module (MCM)
  - Hierarchical NoC interconnection
    - 4x5 mesh topology connects 16 PEs, 1 global PE, and 1 RISC-V
    - 6x6 mesh topology connects 36 chips in package



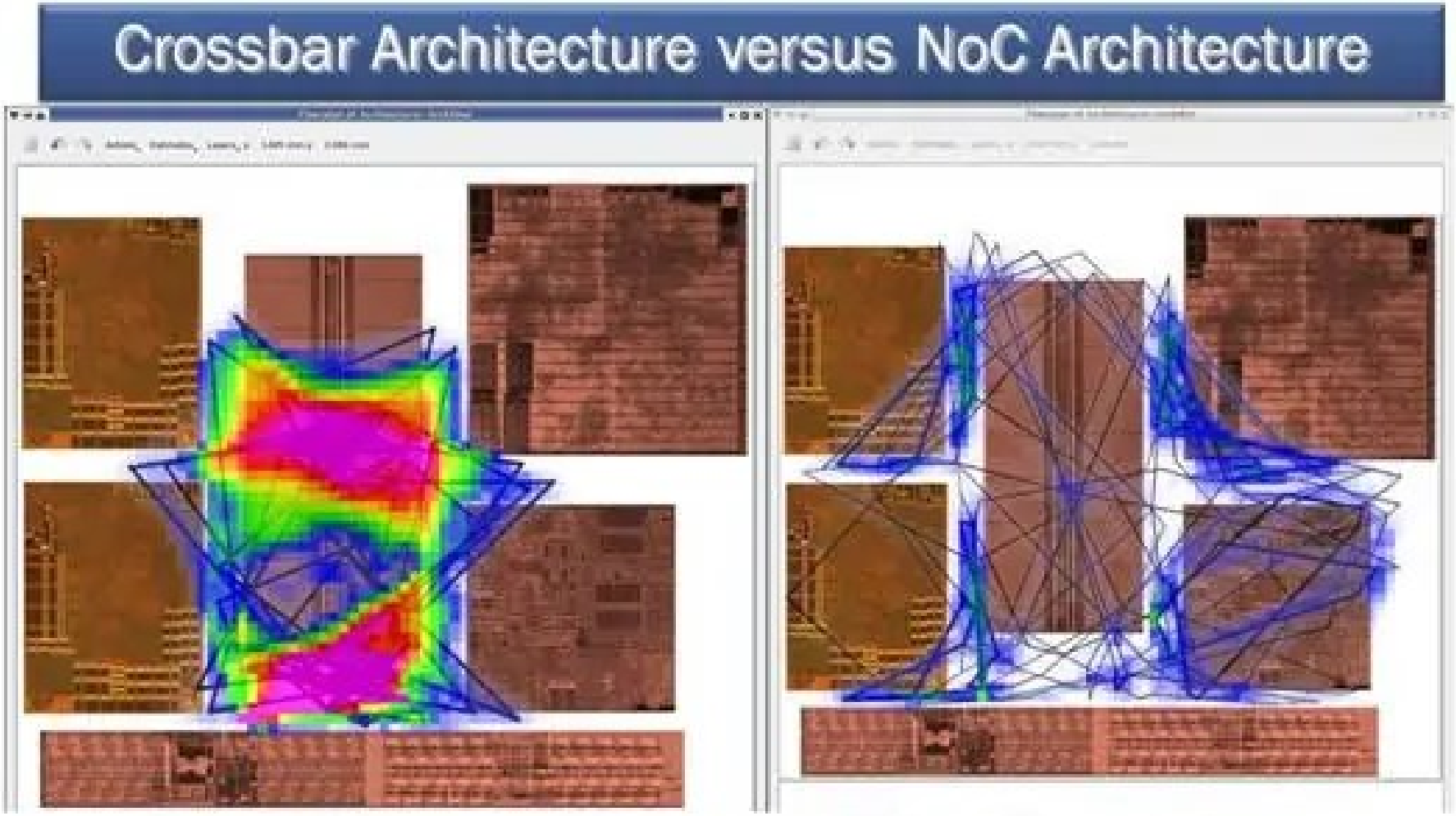
Commercial Product 6: NVIDIA Grace-Hopper, 2022

- NVIDIA Grace CPU
  - 72 ARM Neoverse V2 core
- Chiplet with NVLink4.0 protocol





Commercial Product 6: Arteris FlexNoC, 2013



Keep promoting this new design methods: Special Issue organization

IEEE JOURNAL ON
EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS

Call for Papers

Communication-aware Designs and Methodologies for Reliable and Adaptable On-Chip AI SubSystems and Accelerators

Guest editors

Kun-Chih (Jimmy) Chen, National Sun Yat-sen University, Taiwan (kcchen@mail.cse.nsysu.edu.tw)
Masoumeh (Azin) Ebrahimi, KTH Royal Institute of Technology, Sweden (mebr@kth.se)
Maurizio Palesi, University of Catania, Italy (maurizio.palesi@dieei.unict.it)
Tim Kogel, Synopsys, Germany (tim.kogel@synopsys.com)

An Overview of Efficient Interconnection Networks for Deep Neural Network Accelerators

Seyed Morteza Nabavinejad, Mohammad Baharloo, Kun-Chih Chen, *Member, IEEE*, Maurizio Palesi, *Senior Member, IEEE*, Tim Kogel, and Masoumeh Ebrahimi, *Senior Member, IEEE*

Keep promoting this new design methods: Conference organization

NoCArc, General Chair

General Chair

Kun-Chih (Jimmy) Chen
National Sun Yat-sen University, Taiwan

TPC Chairs

Sergi Abadal
Universitat Politècnica de Catalunya, Spain

Salvatore Monteleone
University of Catania, Italy

Publicity Chair

William Fornaciari
Politecnico di Milano, Italy

General Information

With the advancement in both computing architectures and process technology, many-core architectures can have thousands of cores into a single chip. This integration opens up a plethora of challenges, e.g., in terms of specialization and energy-focused implementations, and supports the spread of various applications and computational paradigms, ranging from multiprocessing to reconfigurable computing, from quantum computing to the emerging area of neuromorphic computing. Such wild increase in the number of processing elements (PE) per chip, together with the growing architectural and workload heterogeneity, calls for efficient, versatile, scalable and reliable communication infrastructures. The Network-on-Chip (NoC) design paradigm, based on a modular packet-switched mechanism, can address many of the on-chip communication issues such as performance limitations of long interconnects, the integration of a large number of PEs on a chip, or heterogeneous workloads. Novel techniques and architectures are needed to efficiently design and optimize the NoC and evaluate it at the network or system level.

The goal of NoCArc is to provide a forum for researchers to present and discuss innovative ideas and solutions related to the design and implementation of multi-core systems on chip. The workshop will focus on issues related to design, analysis, testing, and application of on-chip networks.

13th IEEE/ACM International Symposium on Networks-on-Chip (NOCS 2019)

October 17 – 18. 2019 | New York. USA | co-located with ESWEEK 2019

NOCS, Special Session

Special Sessions

Special Session 1: Interconnection Networks for Deep Neural Networks

Date: Thursday October 17, 2019

Time: 14:00-15:15

Organizers: Kun-Chih (Jimmy) Chen (National Sun Yat-sen University), Masoumeh (Azin) Ebrahimi (KTH Royal Institute of Technology)

Keep promoting this new design methods: Invited Talks

NUS | Computing
National University of Singapore

Speech at NUS, 2019
hosted by Prof. Li Shuan Peh

EVENTS CALENDAR

CS SEMINAR

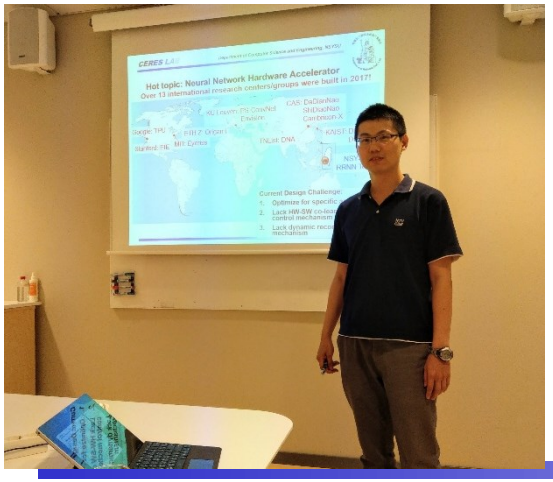
Intelligent Network-on-Chip Algorithm Designs and Hardware Applications

Speaker **Assistant Professor Kun-Chih (Jimmy) Chen, Department of Computer Science and Engineering, National Sun Yat-Sen University**

Chaired by **Dr PEH Li Shiuan, Provost's Chair Professor, School of Computing**

peh@comp.nus.edu.sg

03 Sep 2019 Tuesday, 02:00 PM to 03:30 PM



Welcome!

Speech at KTH, 2018
hosted by Prof. Masoumeh Ebrahimi

The agenda is:

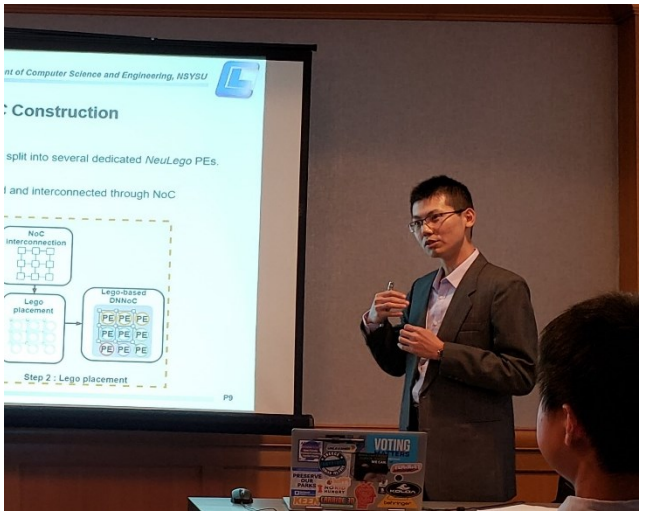
13:00 to 14:00 Jimmy's presentation on “Intelligent NoC design and hardware applications”

14:00 to 14:10 a short coffee break

14:10 to 14:45 Ahmed's presentation on “NOC and Synchronicity enables design of Billion Gate ASICs with the effort of programming”

Keep promoting this new design methods: Invited Talks

Speech at NoCArc, 2022
hosted by Prof. Cristinel Ababei



Speech at EIU, 2022
hosted by Prof. Farhadur Reza

Department of Mathematics and Computer Science

Thursday, October 6, 2022, 4:00 pm

COLLOQUIUM TALK

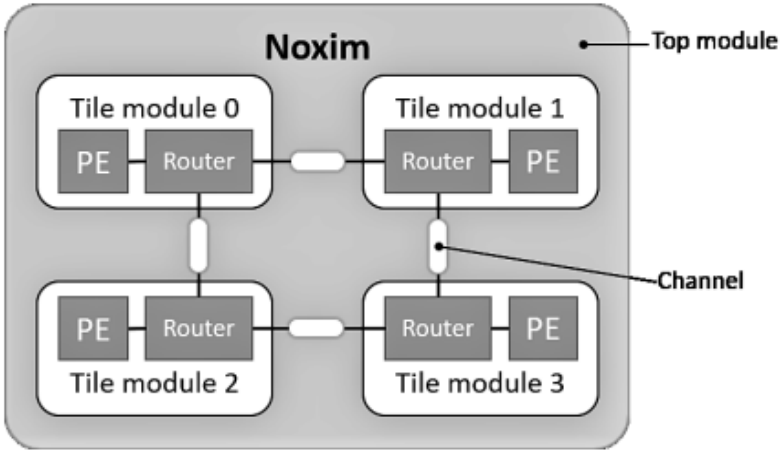
Speaker: Kun-Chih (Jimmy) Chen (National Sun Yat-sen University)

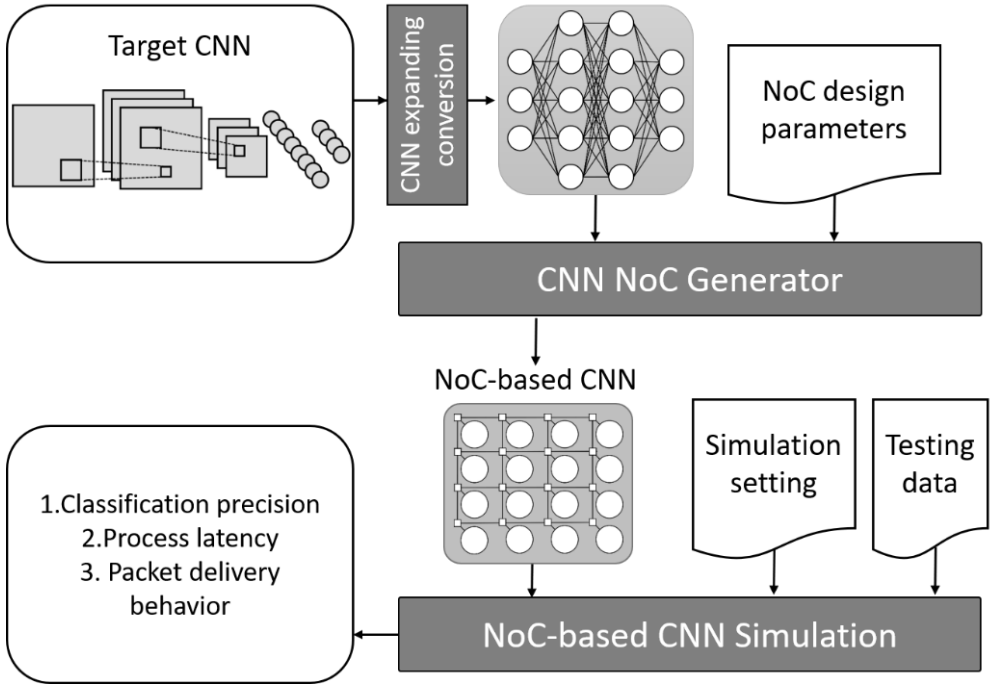
Old Main 2231

Deep Neural Network on Chip (DNNoC) Design Paradigm for Future AI Accelerator Realization

Cycle-Accurate DNNoC Simulator 1: DNNoxim

- Extend the Noxim [5] to make it be able to simulate the complete NoC-based CNN computing behavior.
  - Target CNN will be load in to NoC after CNN pre-process.
  - Map the neuron computing to each PE based on the PE's capability
  - Evaluate the designed DNNoC

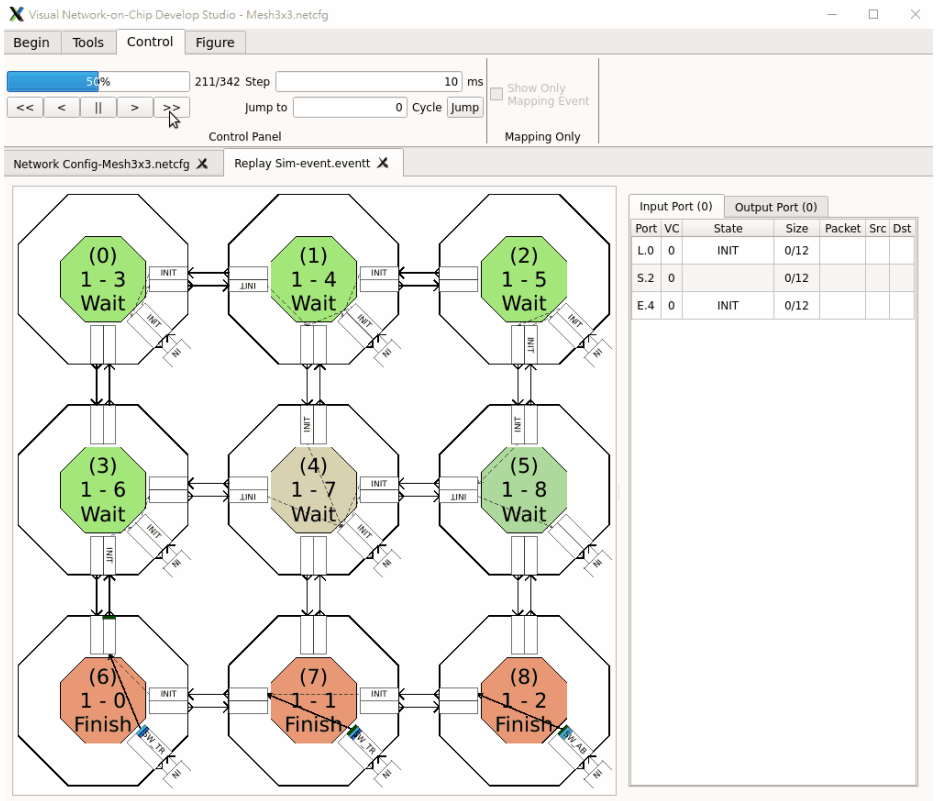




[5] V. Cataniz et al., "Cycle-Accurate Network on Chip Simulation with Noxim," ACM Trans. Modeling and Computer Simulation, vol. 27, no. 1, article 4, Aug. 2016.

Cycle-Accurate DNNoC Simulator 2: ESY-DNNsim

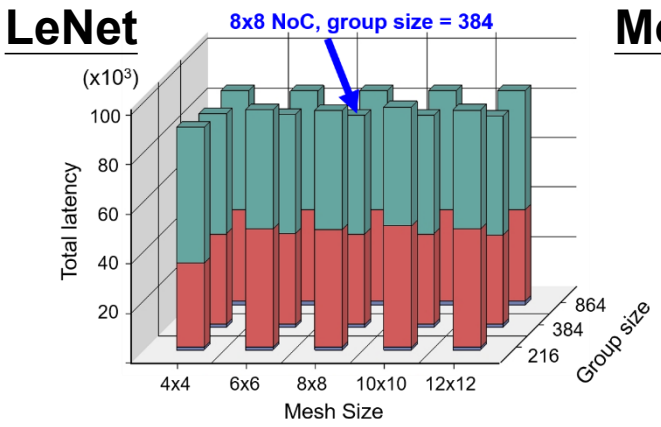
- ESYSim is a visualize cycle-accurate NoC simulator [6].
- We modify ESYSim to make it support neural computing functions.
  - Realizes high-level simulation.
  - Provides early architectural exploration.
- Visual interface
  - Conducive to observation
    - Control bar
    - Traffic situation
    - Router and packet processing information

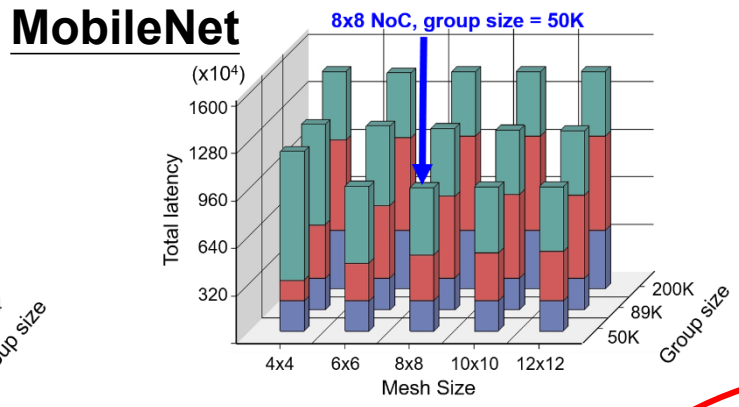


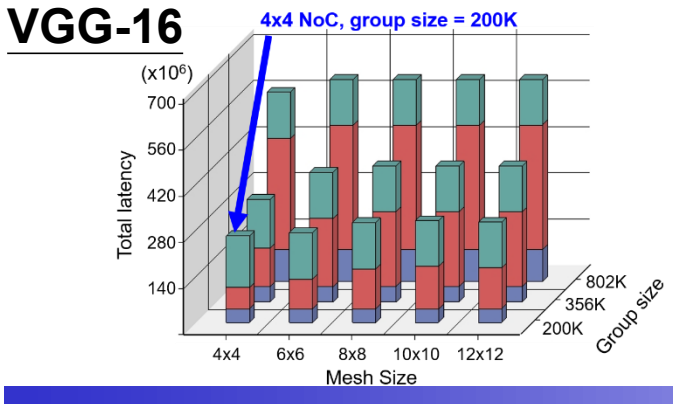
[6] WANG, Junshi et al., “Visualnoc: A visualization and evaluation environment for simulation and mapping,” Proceedings of the Third ACM International Workshop on Many-core Embedded Systems, pp. 18-25, 2016.

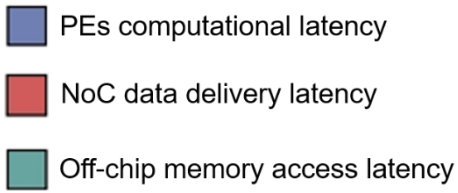
Leverage the DNNoC Evaluation!! (design trade-off)

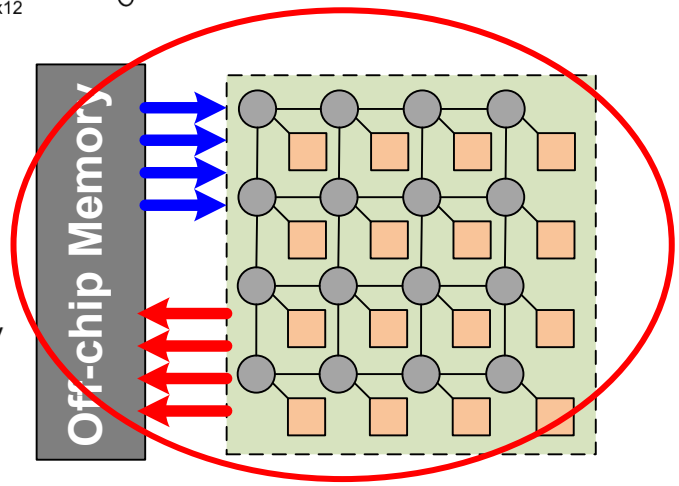
- The designers can find the proper design parameters to achieve better performance





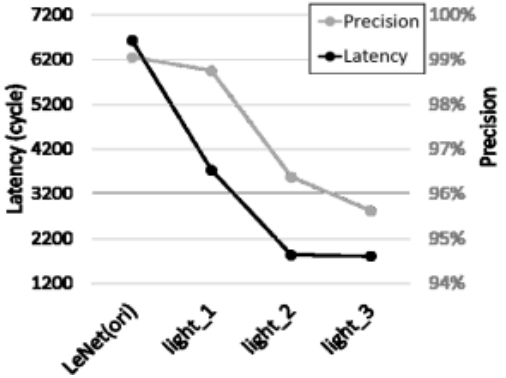


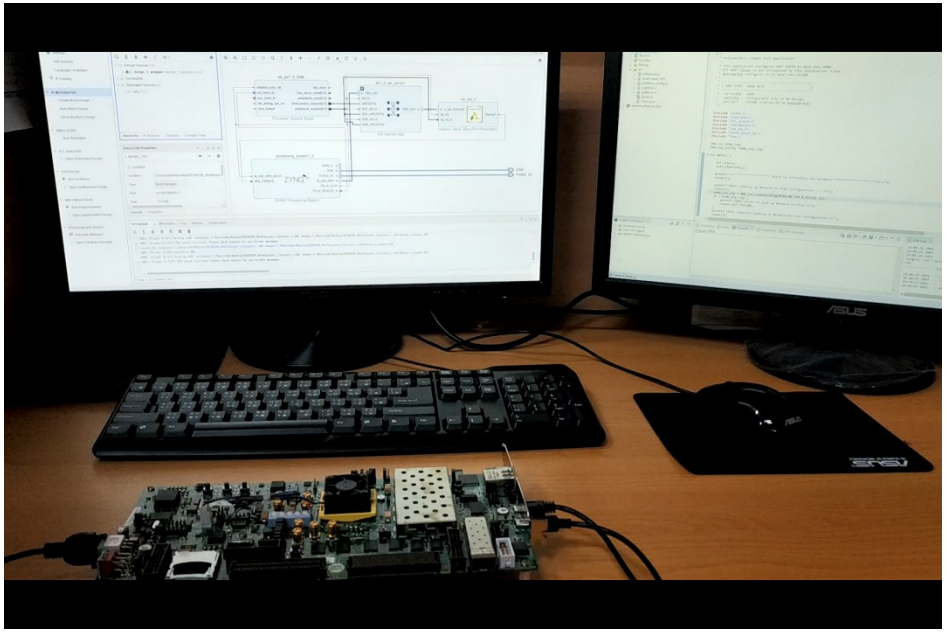




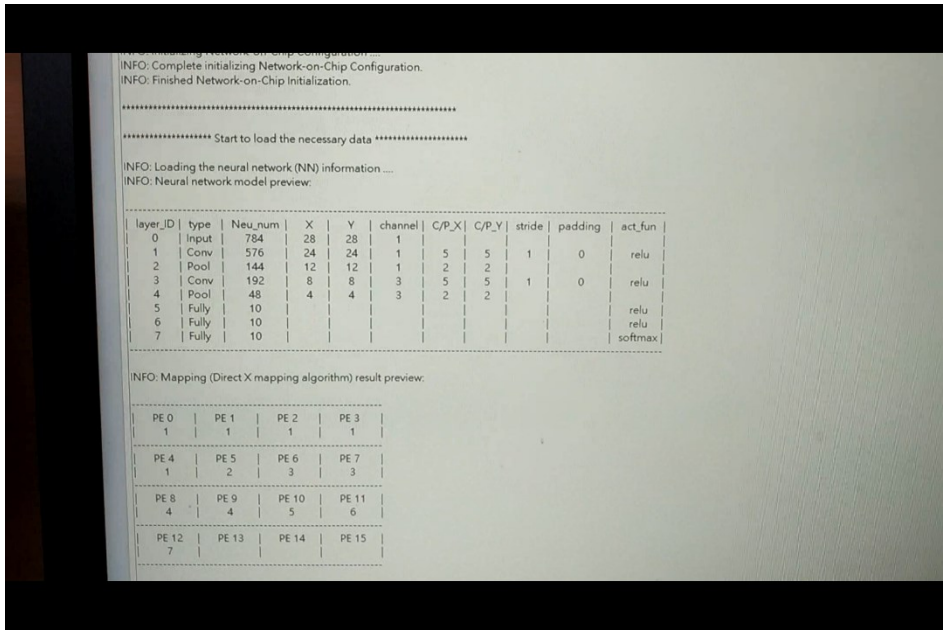
FPGA Verification

- Verify a LetNet model on ZC706 FPGA
  - MINIST dataset
  - 4x4 mesh-based NoC





Development environment



Verification results

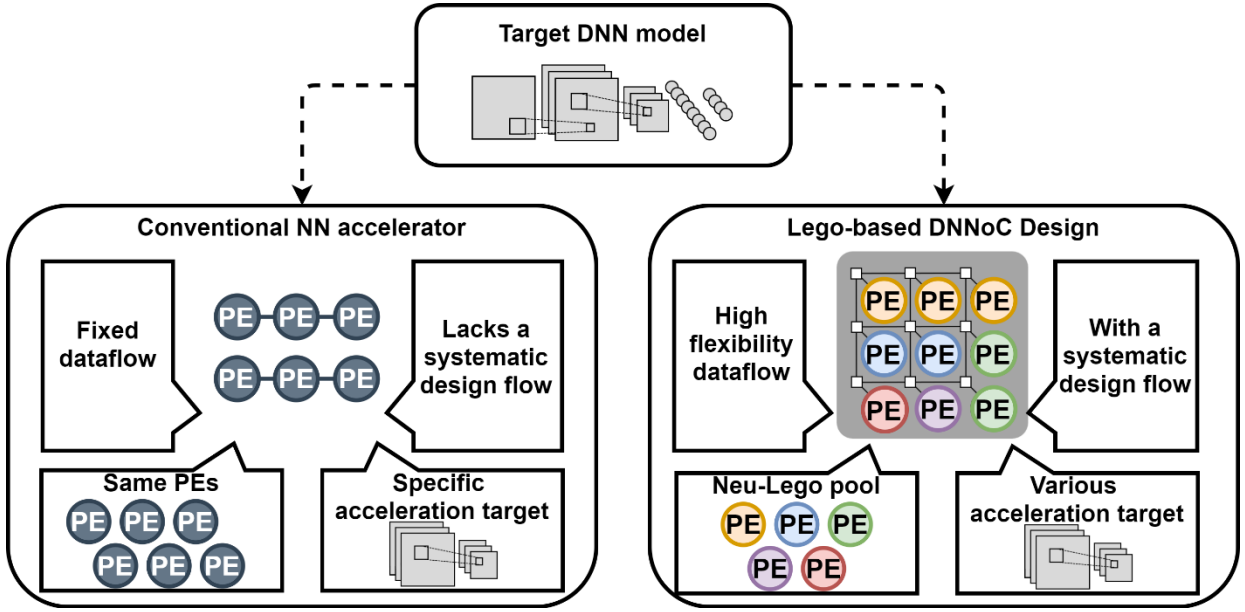
NoC-based DNN design: algorithms and architectures

Heterogeneous Lego-based DNNoC design

Kun-Chih (Jimmy) Chen, Cheng-Kang Tsai, Ya-Sheng Liao, Han-Bo Xu, and Masoumeh Ebrahimi, “A Lego-based Neural Network Design Methodology with Flexible NoC,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 11, no. 4, pp. 711-724, Dec. 2021.

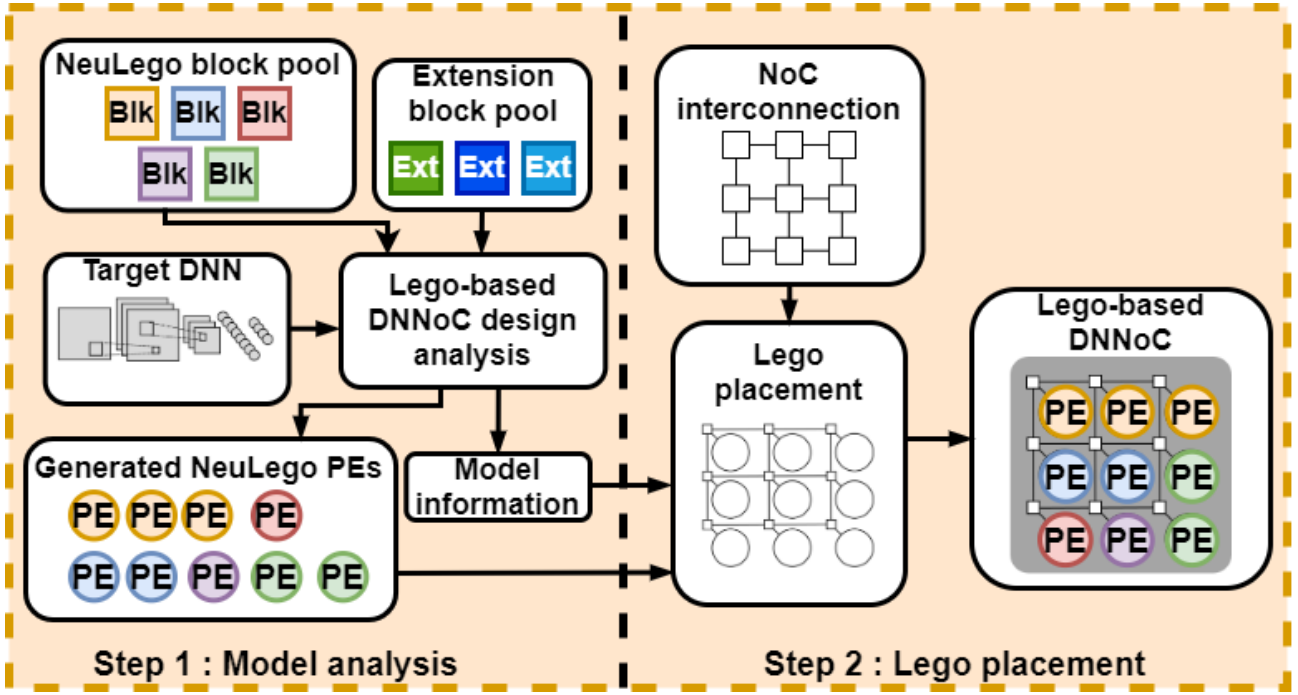
A Lego-based Deep Neural Network Design Methodology with NoC Interconnection

- A systematic design flow in Lego-based *DNNoC* design methodology for higher design flexibility.
- Lego placement and dynamic mapping algorithm for higher PE reusability and hardware efficiency.
- Traffic load reduction method for the data transmission performance



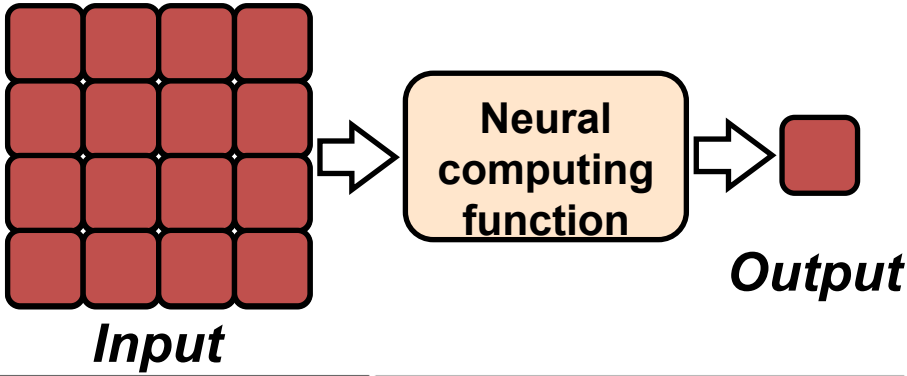
Lego-based *DNNoC* Construction

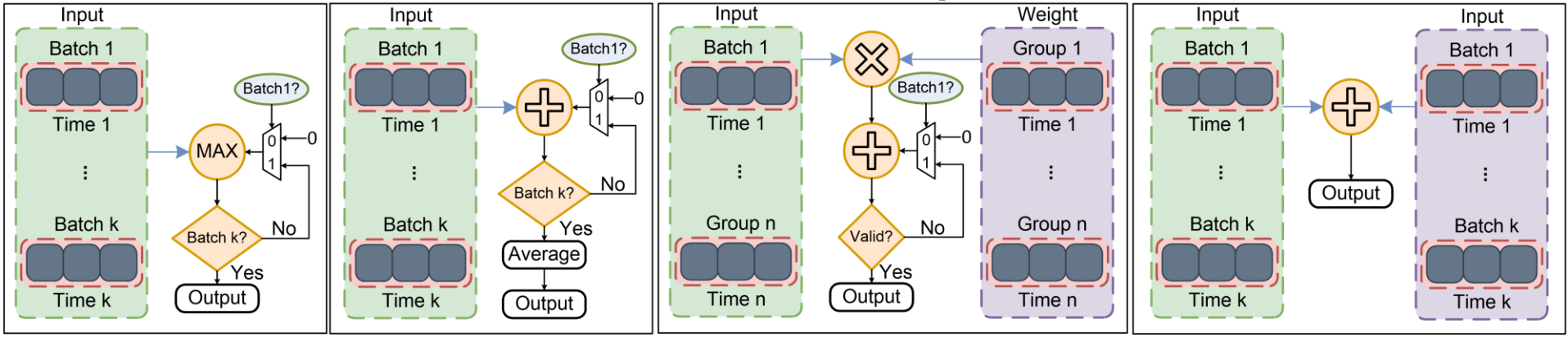
- 1) **Model analysis**
  - The target DNN model is analyzed and split into several dedicated *NeuLego* PEs.
- 2) **Lego placement**
  - The proper *NeuLego* PEs will be placed and interconnected through NoC interconnection.



NeuLego Block Designs

- Uses different *NeuLego* blocks to perform different neural computing functions.
  - The four operations are often used in modern DNN models.
- Batch operation
  - Implement large-scale operations with multiple inferences.
  - Limits the size of the *NeuLego* block.





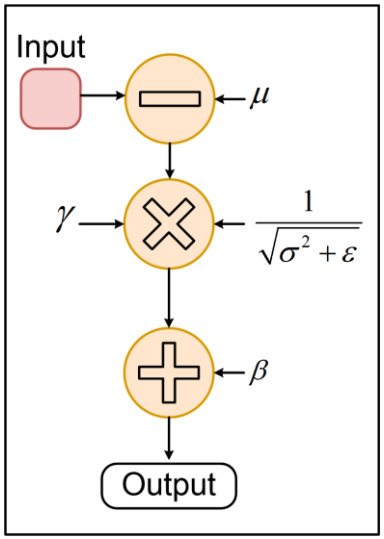
MP ***GAP*** ***MA*** ***ADD***

Extension Blocks

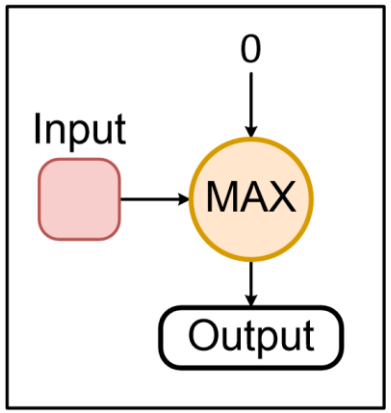
- The extension PE is used to perform the data post-processing for the output of *NeuLego* PEs.
  - Normalization layers or activation function layers are often used in modern DNN models.

$$BN(x) = \frac{\gamma(x-\mu)}{\sqrt{\sigma^2+\varepsilon}} + \beta$$

$$ReLU(x) = \max(0,\ x)$$



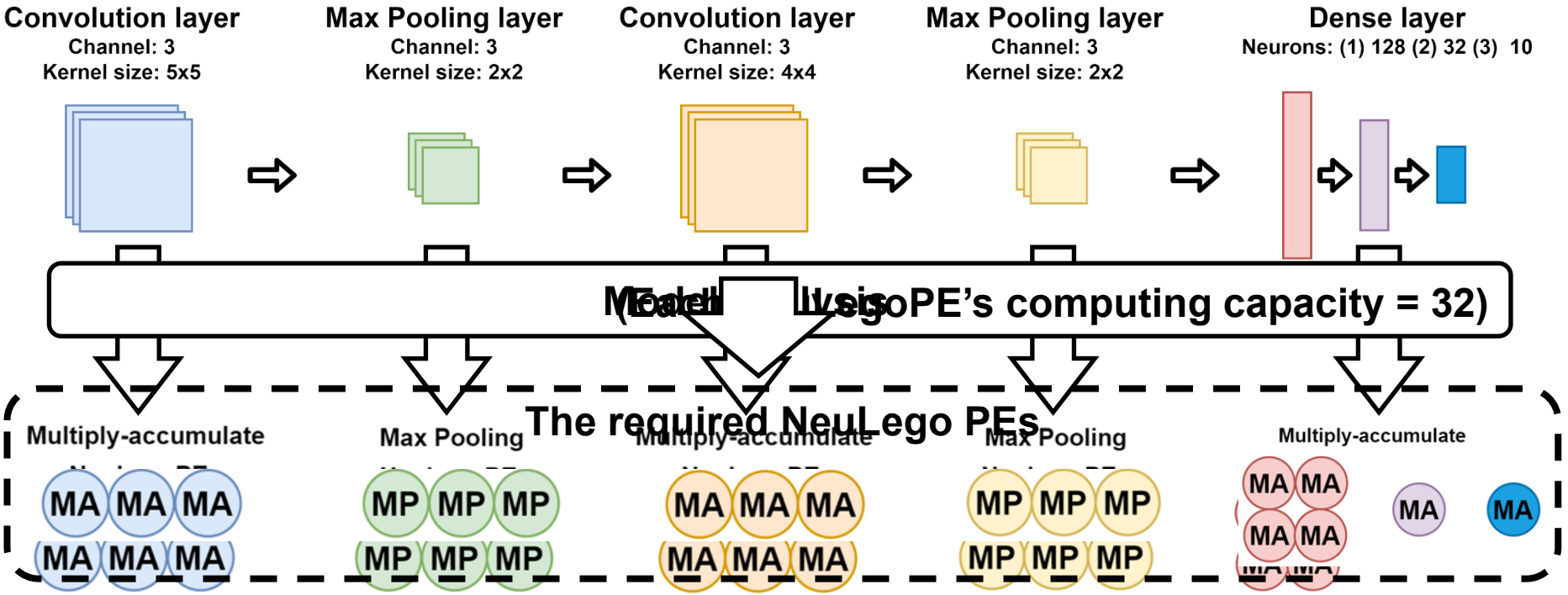
BN



ReLU

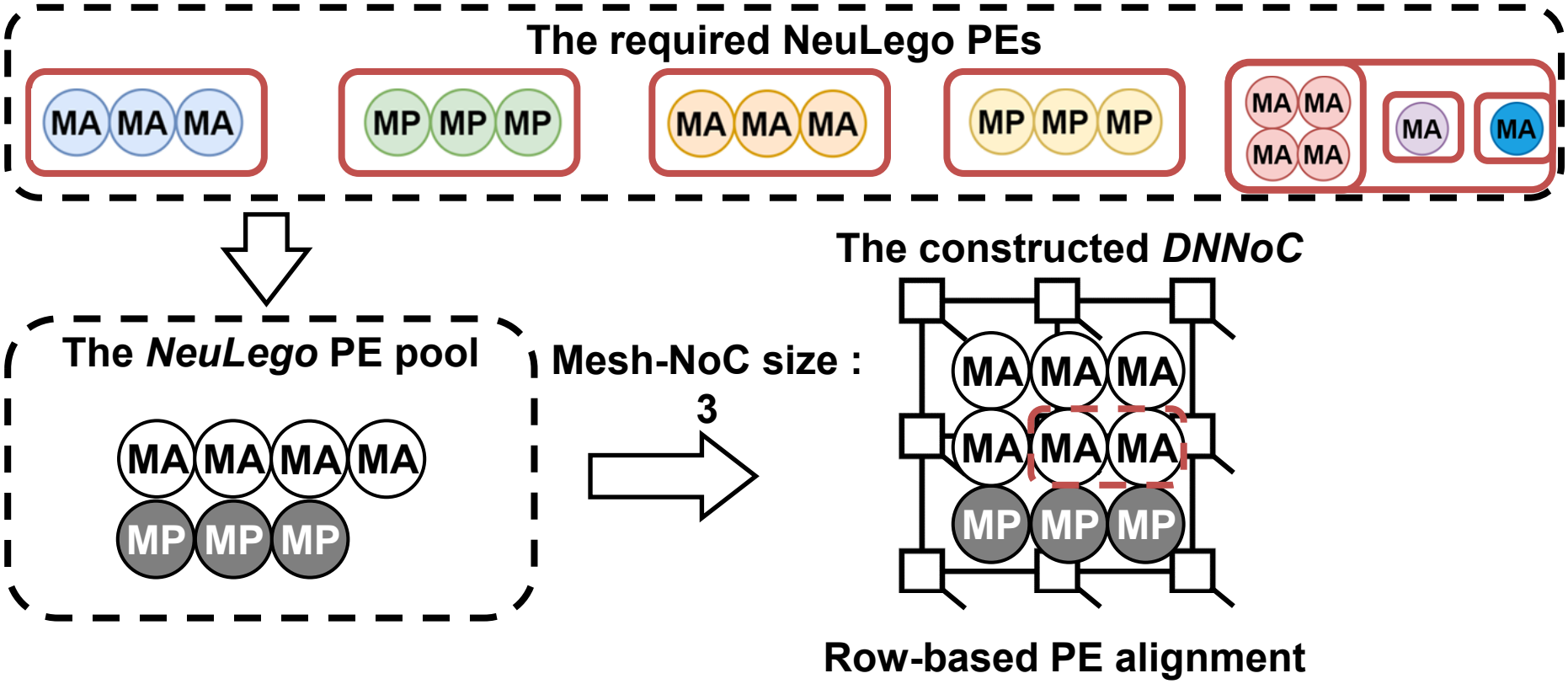
DNNoC Construction (1/2) : Model Analysis

- Obtains the required number of *NeuLego* PEs and model information from the given DNN model.
- Each *NeuLego* PE is used to process data from the same data dimension.
  - Facilitates data exchange.



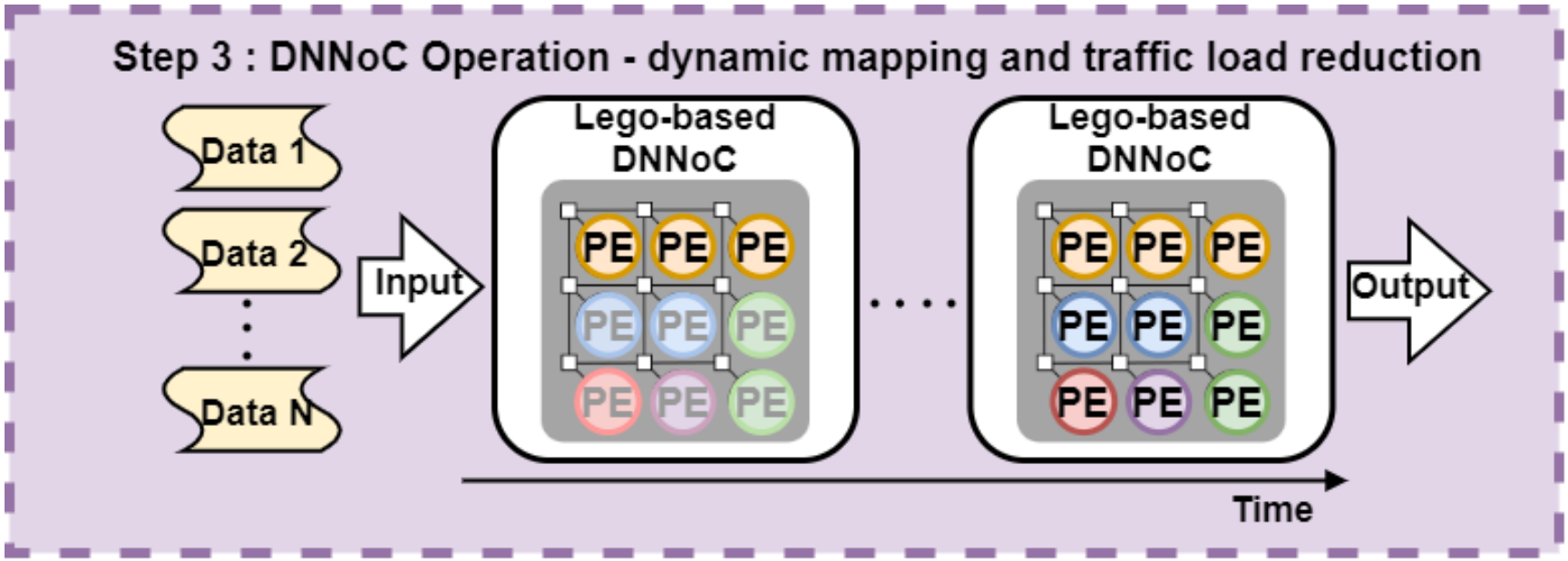
DNNoC Construction (2/2) : *DNNoC* Construction Flow and Lego Placement

- We propose to share the computing resources and find the proper number of *NeuLego* PEs for a given DNN model.
  - Improve hardware efficiency



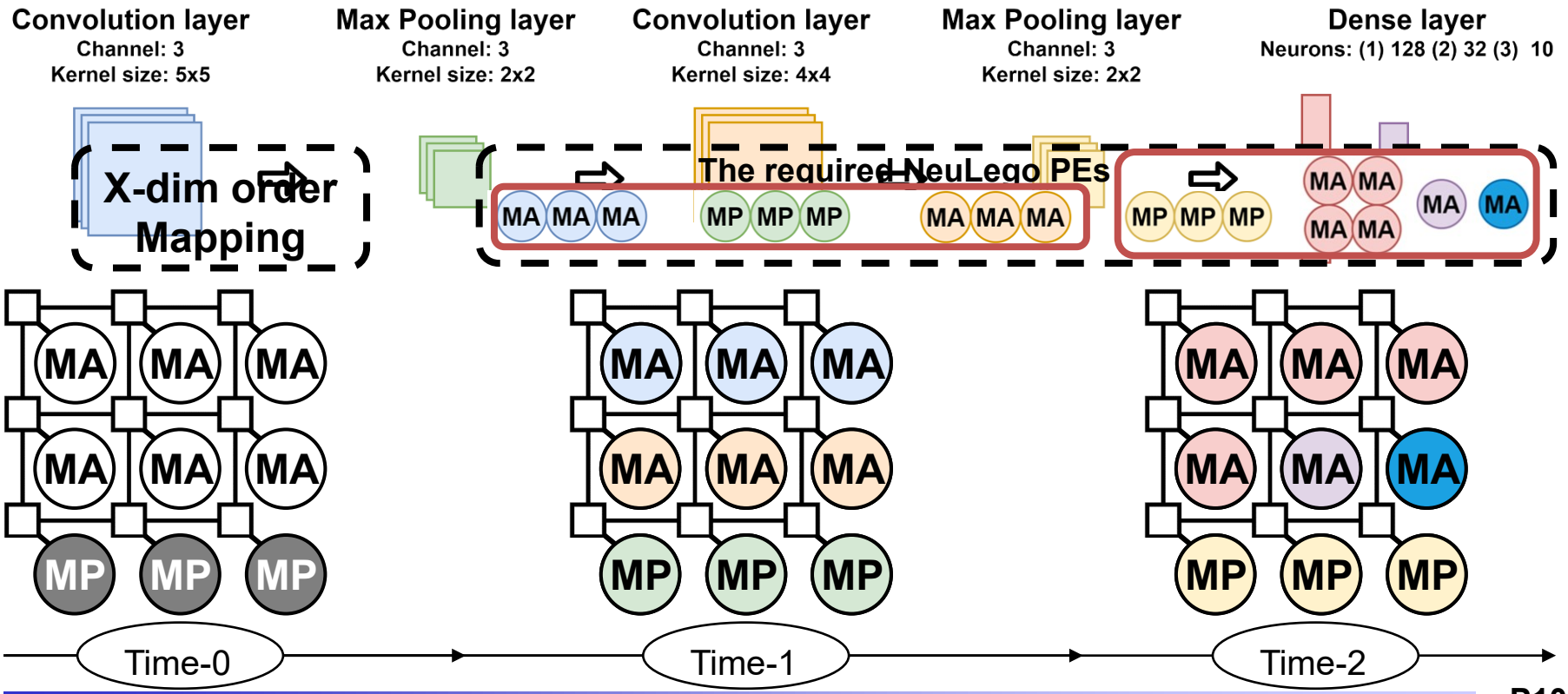
DNNoC Execution

- The involved *DNNoC* platform is used to compute the target DNN model.
  - **Dynamic mapping algorithm**
    - Enables large-scale DNN models to be operated on a resources-limited platform.
- **Traffic load reduction method**
  - Reduce overall traffic and improve system performance.



DNNoC Execution

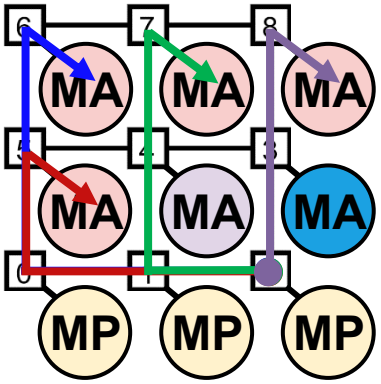
- **Layer-wise dynamic mapping algorithm**
  - The available computing resources of the constructed *DNNoC* would be sufficient to fit the largest layer of the target DNN model.
  - Maps a large-scale DNN model to the source-limited *DNNoC* platform.



Traffic Load Reduction Method (1/2) : Multicasting

- Based on the model analysis method, most packets have the same source.
  - Uses multicasting method to reduce traffic complexity and number of packets.
- Adopts the Hamiltonian routing algorithm [8]
  - The Hamiltonian path makes the traffic flow regular.
  - Low-complex routing algorithm

Total number of packets = 4



Packet Header

Type	Source		Destination	
Head	X	Y	X	Y
00	2	0	Dst-x	Dst-y

Packet Body

Type	Data from NeuLego$_{blk}$								
Body	0	1	2	3	4	5	6	7	8
01	9	9	9	9	6	0	0	0	0

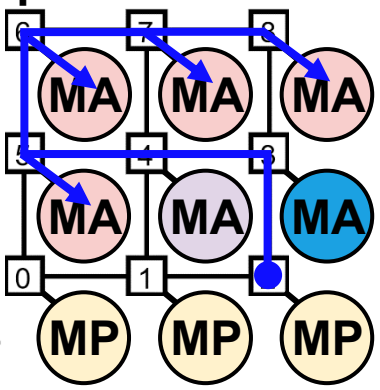
Packet Tail

Type
Tail
10

(Dst-x, Dst-y) = {(0, 1), (0, 2), (1, 2), (2, 2)}

Unicast

Total number of packets = 1



Packet Header

Type	Source		Destination							
Head	X	Y	0	1	2	3	4	5	6	7	8
00	2	0	0	0	0	0	0	1	1	1	1

Packet Body

Type	Data from NeuLego$_{blk}$								
Body	0	1	2	3	4	5	6	7	8
01	9	9	9	9	6	0	0	0	0

Packet Tail

Type
Tail
10

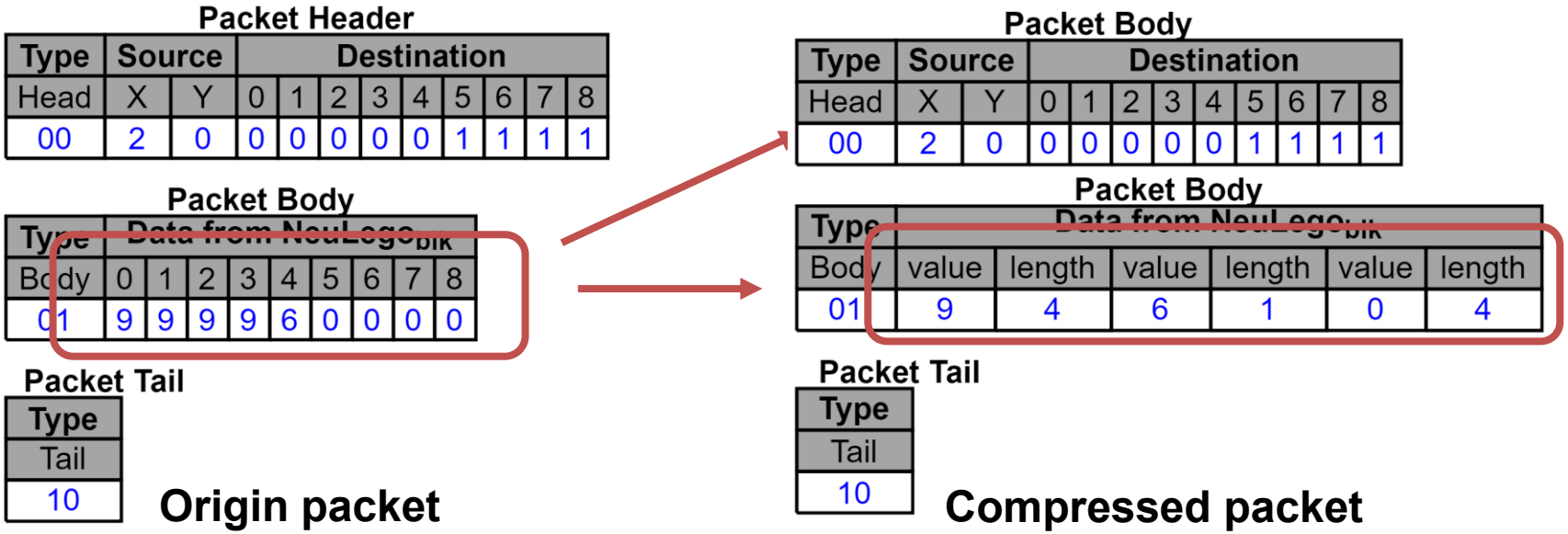
Multicast

Hamiltonian path

[8] X. Lin et al., "Deadlock-free multicast wormhole routing in 2-d mesh multicomputers," IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 8, pp. 793–804, 1994.

Traffic Load Reduction Method (2/2) : Packet Compression

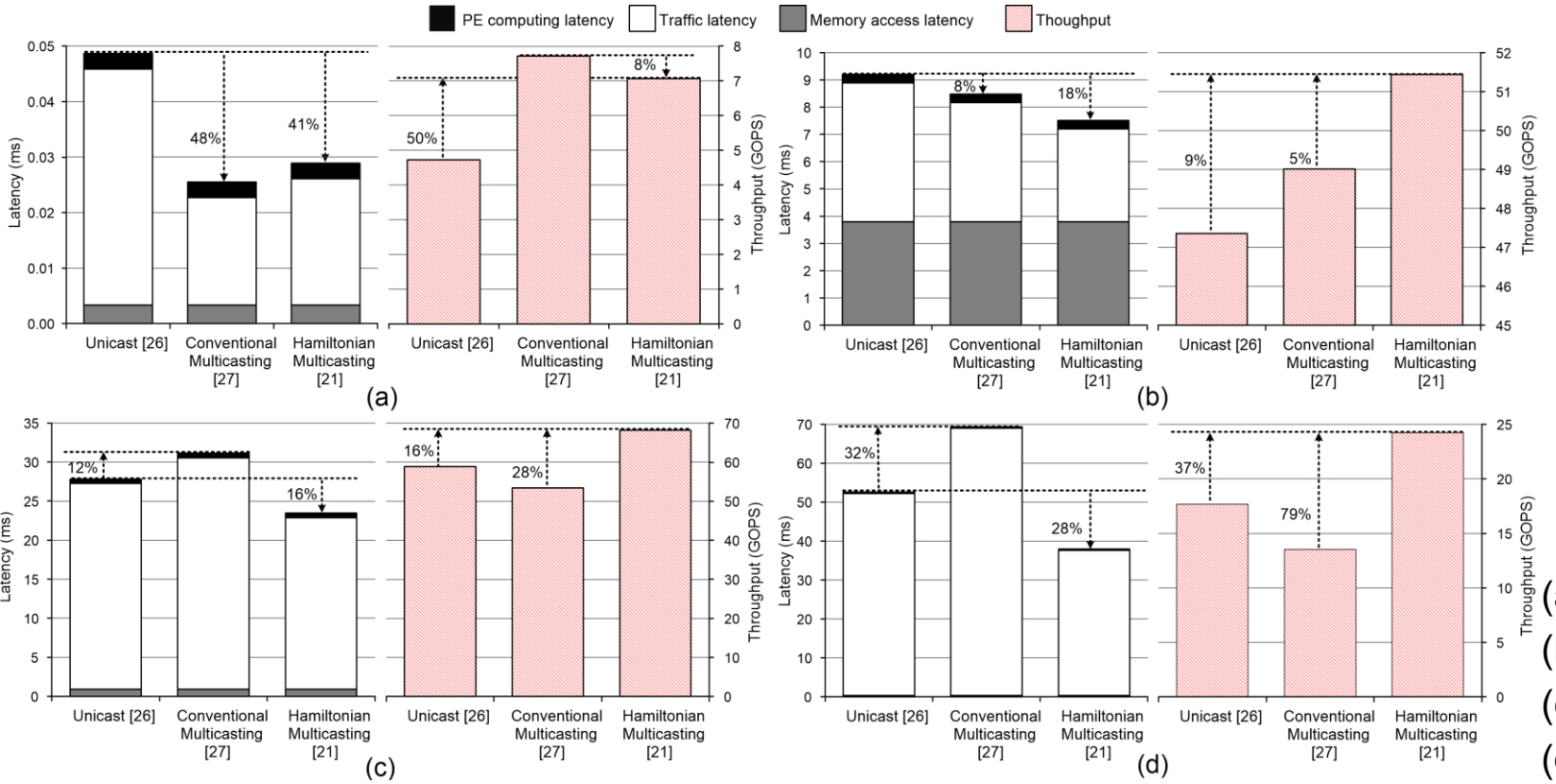
- There are many successive identical data in the computing result of each layer in the DNN model.
  - Because the *ReLU* layer is widely used in modern DNN models.
- Run-length encoding [9]
  - Good compression effect on continuous same values.
  - Helps to reduce packet size.



[9] A. Birajdar, et al., “Image Compression using Run Length Encoding and its Optimisation,” Global Conference for Advancement in Technology (GCAT), pp. 1–6, 2019.

The Performance Comparison by using Hamiltonian Multicasting

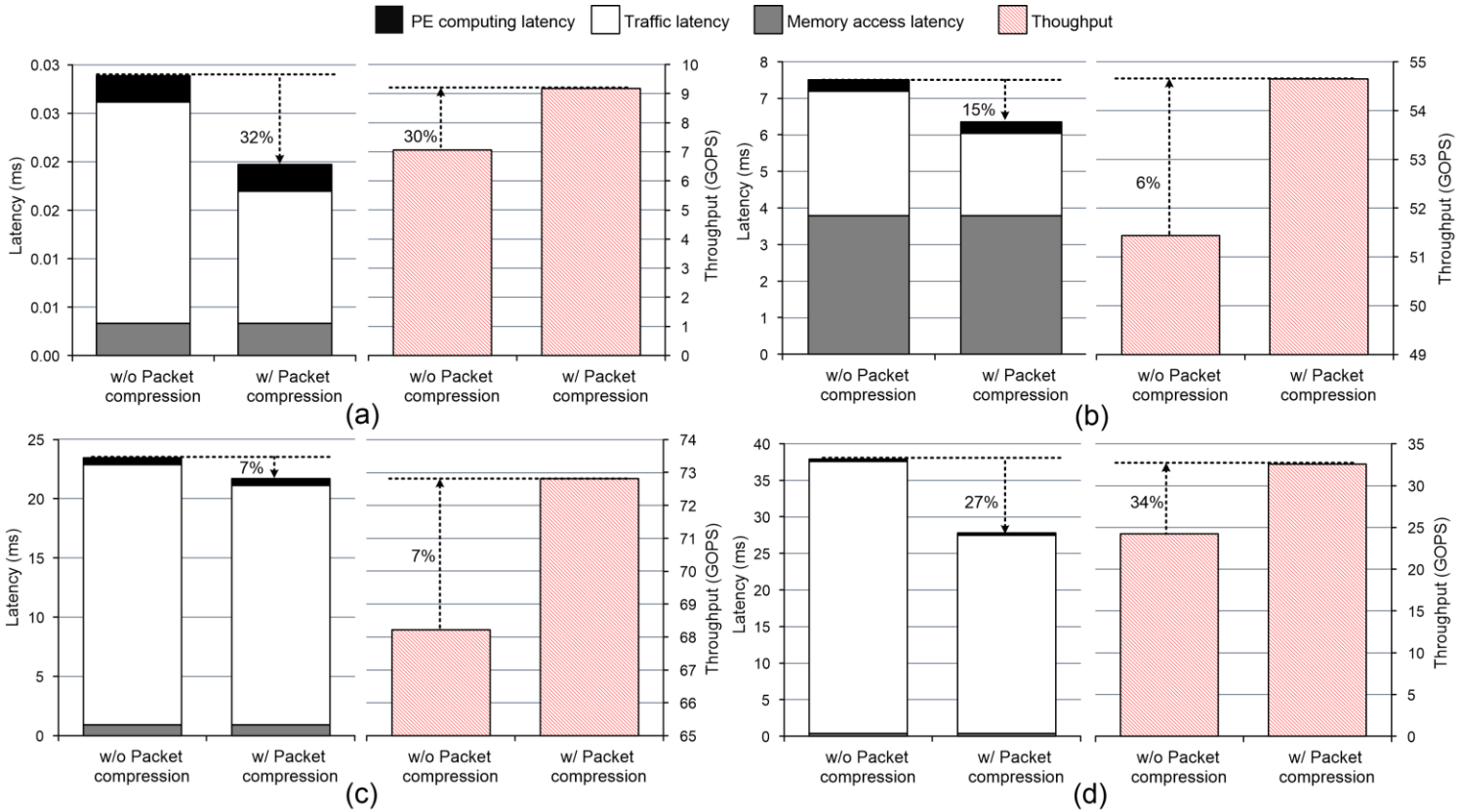
- The multicasting method uses fewer packets to transmit the required data, which mitigates the traffic load on the network.
  - Improves throughput by 9% to 50%.



(a) LeNet
(b) AlexNet
(c) ResNet-18
(d) MobileNet v1

The Performance Comparison by using Packet Compression

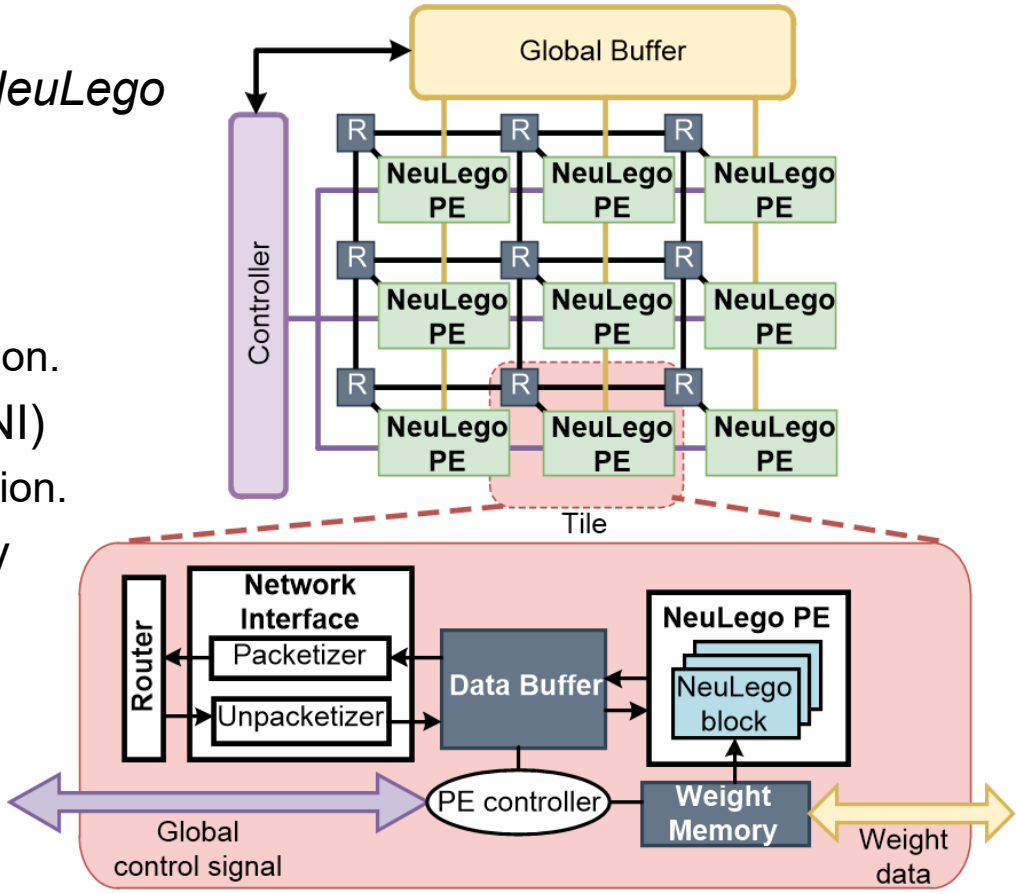
- The packet compression method can compress data and help to reduce packet size.
  - Improves throughput by 6% to 34%.



(a) LeNet
(b) AlexNet
(c) ResNet-18
(d) MobileNet v1

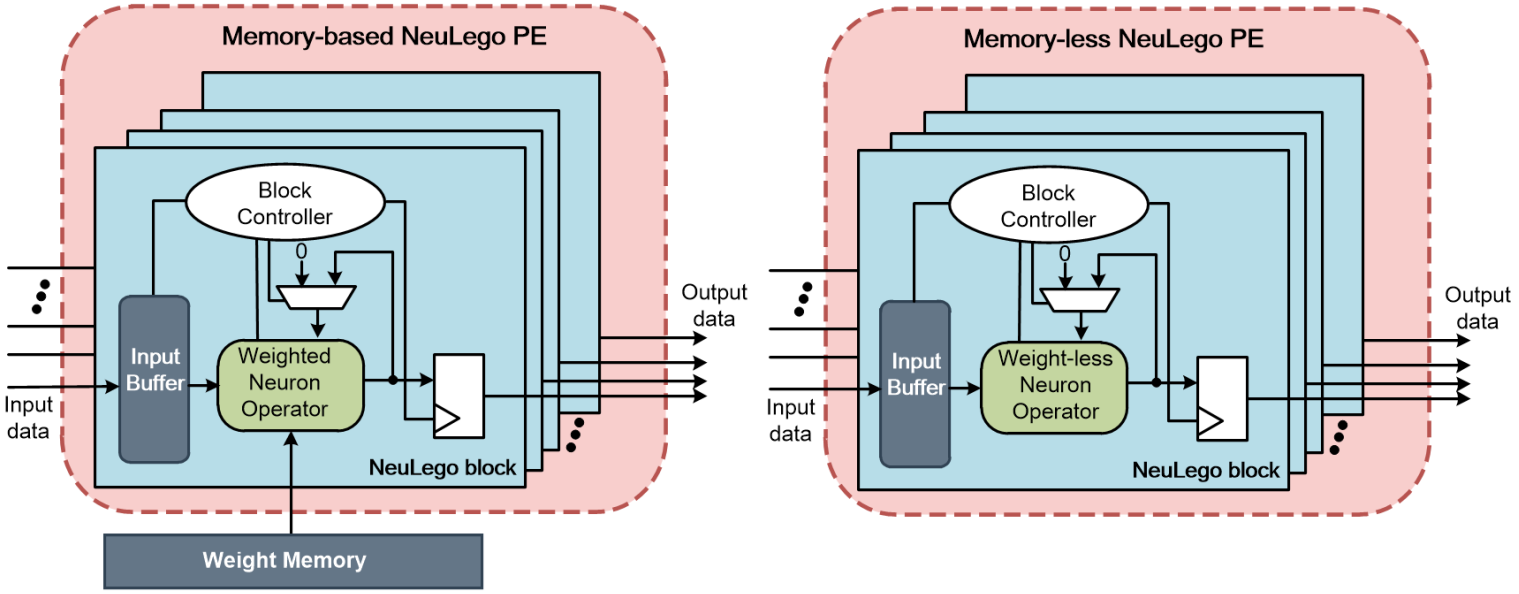
Hardware Architecture for the Proposed Lego-based *DNNoC* Design

- The *DNNoC* contains global buffer, **controller**, and **tiles**.
- Controller
  - Used to send control signal to *NeuLego* PEs and global buffer.
- Tile
  - *NeuLego* PE
    - Provides neural computing function.
  - Router and Network Interface (NI)
    - Provides data transmission function.
  - Data Buffer and Weight Memory
    - Provides data storage function.
  - PE controller



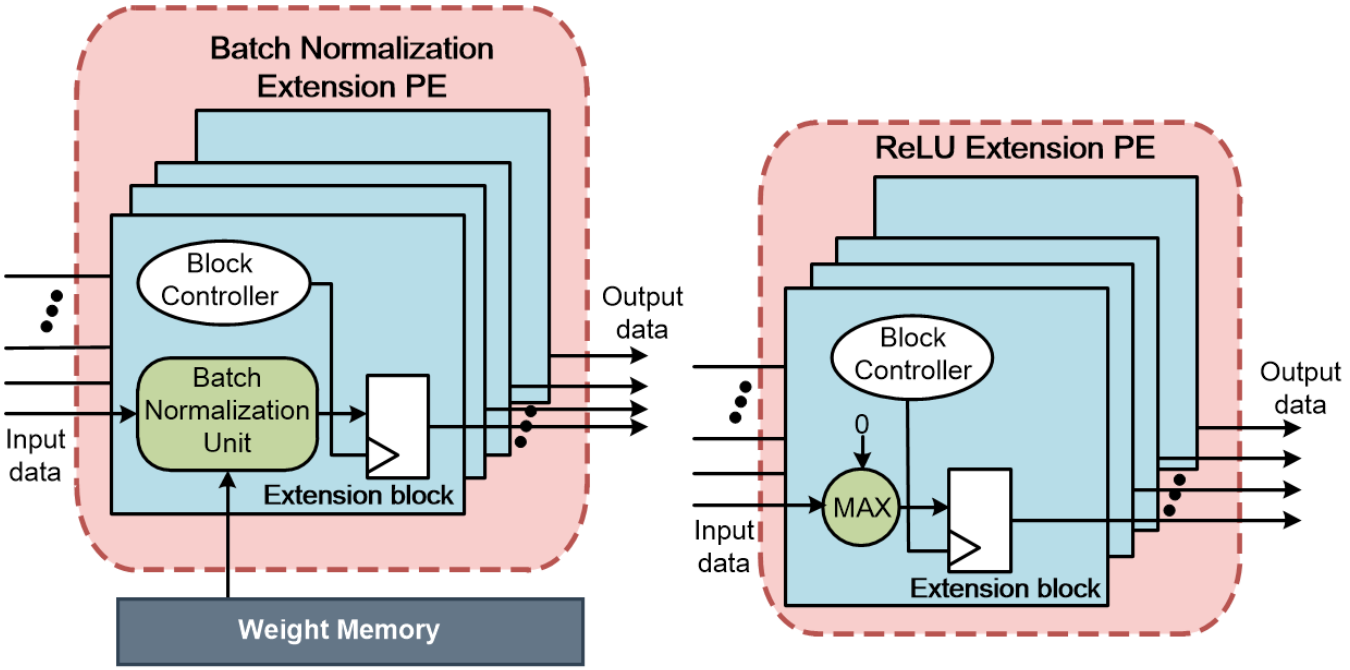
Hardware Design of the *NeuLego* PE

- The NeuLego PE is used to perform the neural computing function.
- The memory-based *NeuLego* PE
  - Connects with the weight memory and provide weight to neuron function.
  - *GAP* or *MA.*
- The memory-less *NeuLego* PE
  - *MP* or *ADD.*



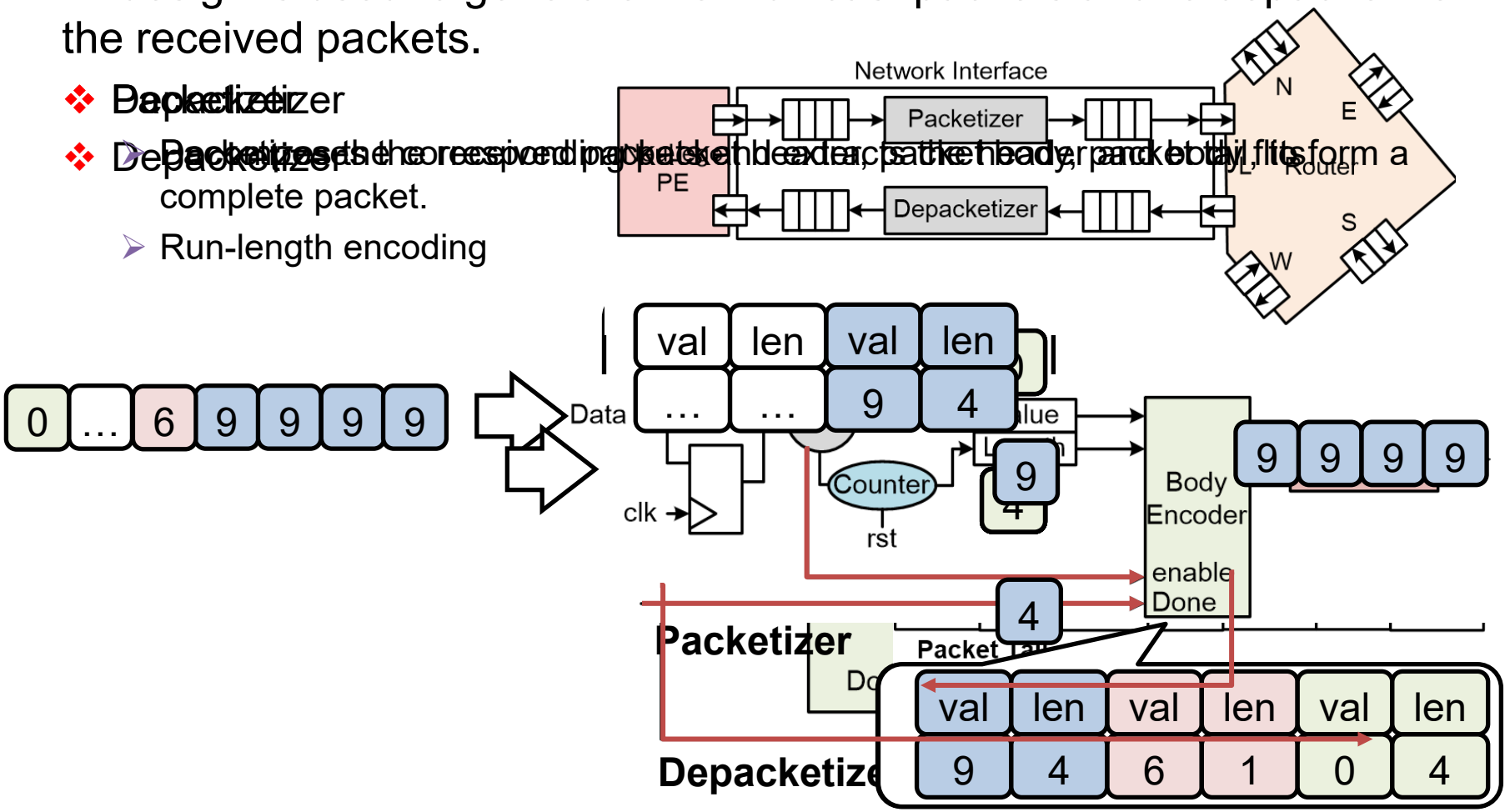
Hardware Design of the Extension PE

- The extension PE is used to perform the data post-processing for the output of *NeuLego* PEs.
  - The *NeuLego* PE and the extension PE are in the same tile.
    - The number of extension blocks should be equal to the number of *Neu-Lego* blocks.
  - *BN* or *ReLU.*



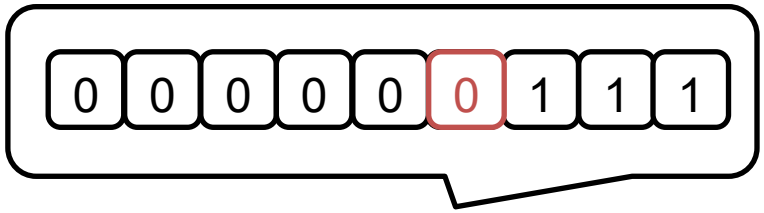
Hardware Design of the NI

- NI design is used to generate the multicast packets and to depacketize the received packets.
  - Packetizer
  - Depacketizer
    - Depacketizer checks the received packet header, strips the header, and builds a complete packet.
      - Run-length encoding



Hardware Design of the Router

- Provides the main traffic function for NoC.
- Routing Computing Unit
  - Supports the Hamiltonian routing algorithm.
  - Supports the multicasting function.



Packet Body

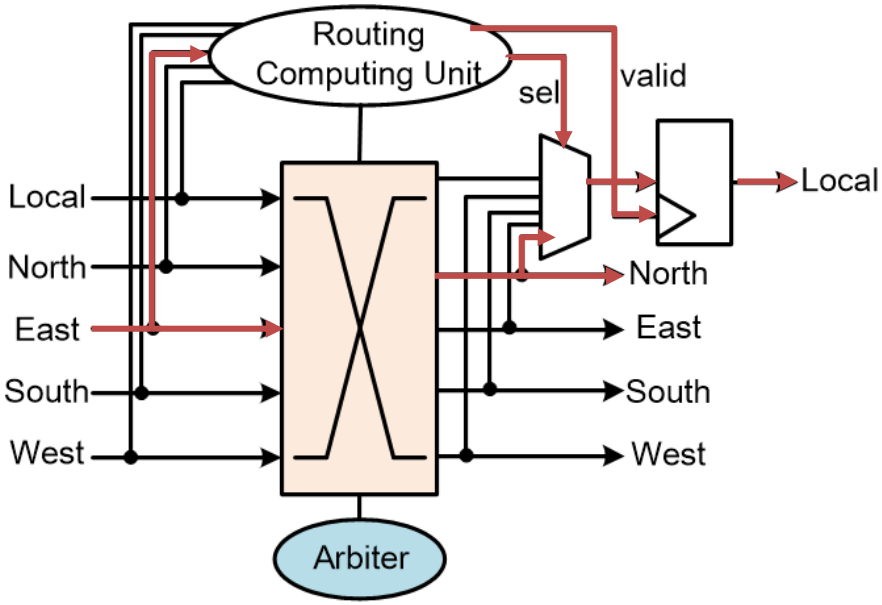
Type	Source		Destination							
Head	X	Y	0	1	2	3	4	5	6	7	8
00	2	0	0	0	0	0	0	1	1	1	1

Packet Body

Type	Data from NeuLego$_{blk}$					
Body	value	length	value	length	value	length
01	9	4	6	1	0	4

Packet Tail

Type
Tail
10



Router architecture

The Performance Comparison

- The performance comparison in the modern DNN models
  - Include more neuron operations
    - Depth-wise convolution, global average pooling, batch normalization, etc.
  - The overall throughput of the conventional design methodologies is low.
    - Because of the low design flexibility.
  - The proposed *DNNoC* design methodology brings the advantage of better throughput than the related works.

	Eyeriss [3]		Eyeriss v2 [5]		UNPU [19]		Proposed *DNNoC*	
Technology	65nm		65nm		65nm		40nm	40nm
Area (Gate count)	1,176k gates		2,695k gates		13,680k gates		3,371k gates	1,012k gates
On-chip SRAM	181.5 KB		246 KB		256 KB		3,760 KB	816 KB
Frequency	200 MHz		200 MHz		200 MHz		200 MHz	
Bit precision	16-bit		16-bit		16-bit		16-bit	
DNN model	**AlexNet**	**MobileNet v1**	**AlexNet**	**MobileNet v1**	**AlexNet**	**MobileNet v1**	**AlexNet**	**MobileNet v1**
Inference/sec (Overall)	0.92	0.69	43.19	1.63	74.35	0.71	53.69	32.79

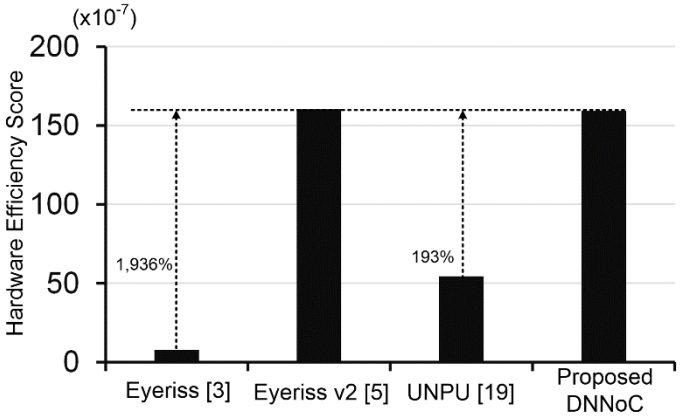
Analysis of Hardware Efficiency

- We use HE_score to analyze the comparison of hardware efficiency between the proposed *DNNoC* and the related works.

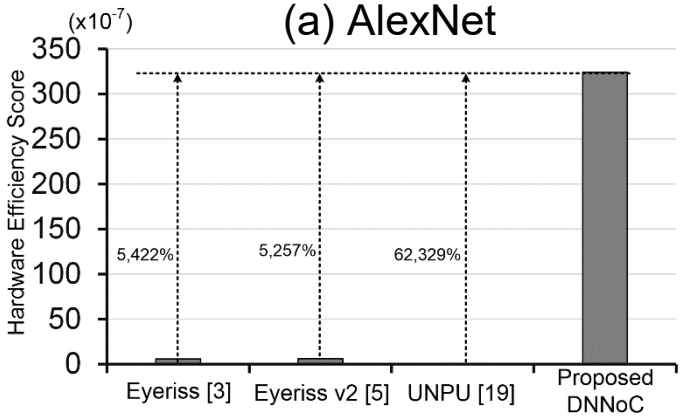
$$HE\_score = \frac{Throughput}{Area}$$

- Improves the hardware efficiency by 193% to 1,936% under the AlexNet model.
  - The hardware efficiency of the *DNNoC* and Eyeriss v2 are almost the same because the *DNNoC* includes a larger global buffer.

- Improves the hardware efficiency by 5,257% to 62,329% under the MobileNet v1 model.



(a) AlexNet



(b) MobileNet v1

NoC-based DNN design: algorithms and architectures

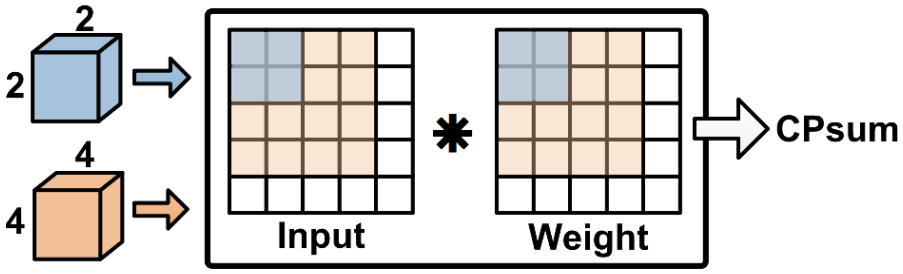
Kernel-size applicable DNNoC design

Kun-Chih (Jimmy) Chen and Yi-Sheng Liao, "A Reconfigurable Deep Neural Network on Chip Design with Flexible Convolutional Operations," *IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc), Oct. 2022*

Design Challenge of DNN Accelerator: Various kernel size

- The convolutional kernel sizes are usually not fixed in the DNN model.
  - Worst-case design consideration
- The register size of processing element (PE) is usually based on the largest kernel size in the target model.
  - Low utilization of PE computational capability.
  - Cannot process the operation.

DNN model	Kernel size/shape
AlexNet	3x3, 5x5, 11x11
GoogLeNet	1x1, 3x3, 5x5, 7x7
DeepSpeech2	21x11, 41x11

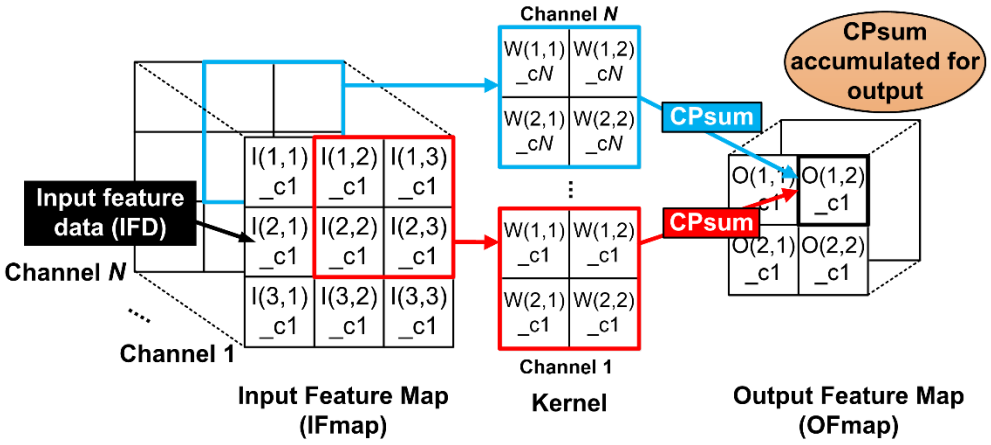


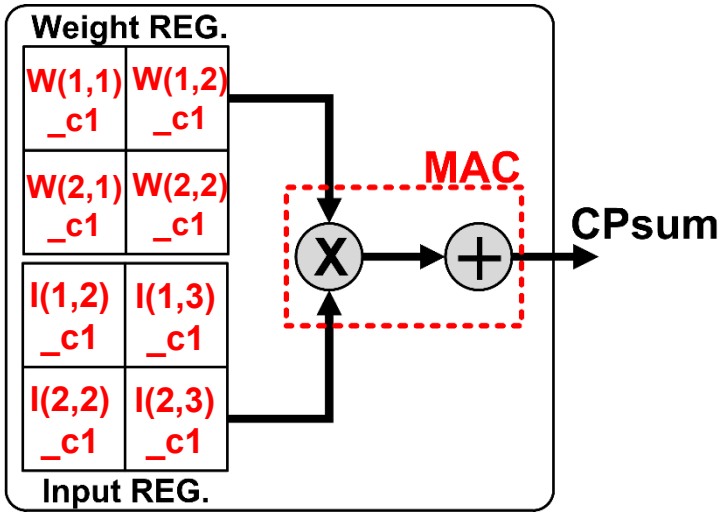
Channel-wise Convolution Operation

- The channel-wise convolution operation.
  - PE is low applicable for arbitrary kernel size.
  - PE generates channel partial sum (*CPsum*).

$$CPsum_{(i,j,c)} = \sum_{m=1}^{h}\sum_{n=1}^{h}\left(I_{(i+m-1,j+n-1,c)} \times W_{(m,n,c)}\right)$$

$$OFmap_{(i,j)} = \sum_{c=1}^{d}\left(CPsum_{(i,j,c)}\right)$$



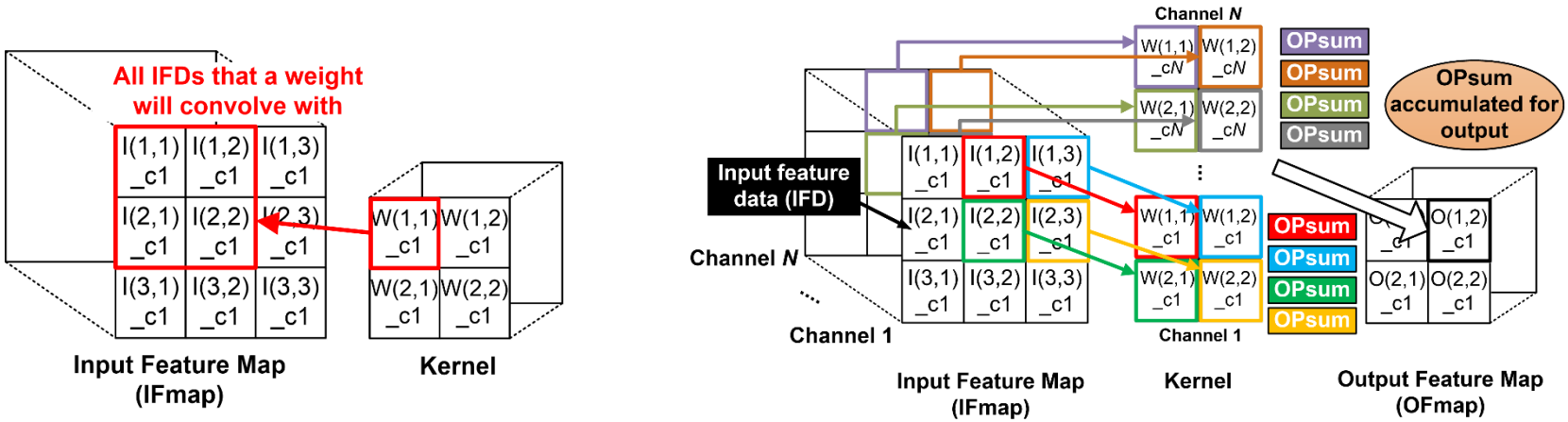


Weight-wise NN Processing Mechanism (1/2)

- We can exploit the shape parameters to infer all Input Feature Map Data (*IFD)* that the weight will convolve with.
- PE will process one weight with corresponding inputs and accumulate operation partial sum (*OPsum*).

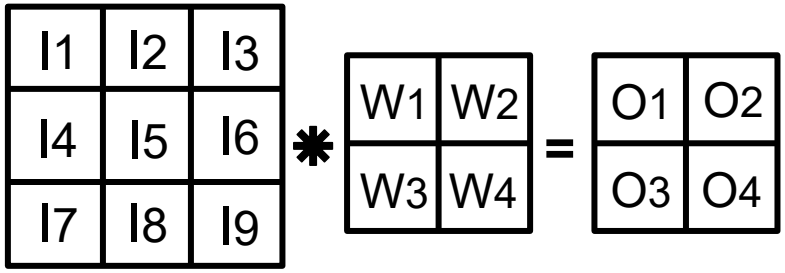
$$OPsum_{(i,j,m,n,c)} = I_{(i+m-1,j+n-1,c)} \times W_{(m,n,c)}$$

$$OFmap_{(i,j)} = \sum_{c=1}^{d} \sum_{m=1}^{h} \sum_{n=1}^{h} \left(OPsum_{(i,j,m,n,c)} \right)$$



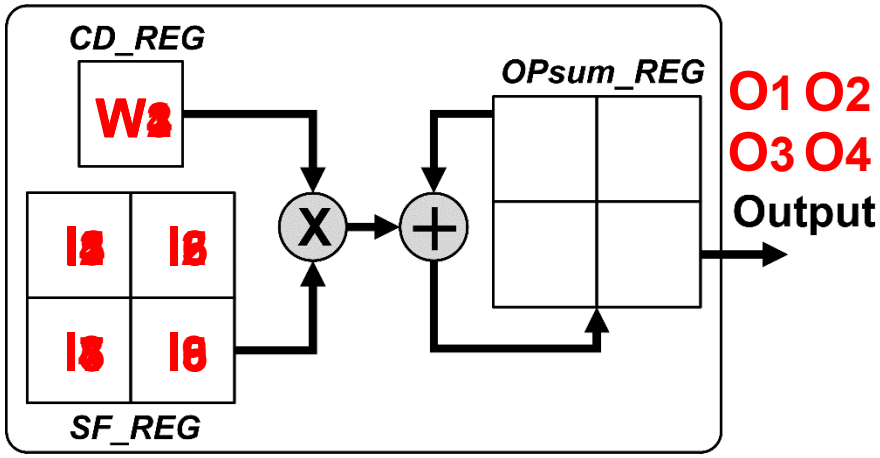
Weight-wise NN Processing Mechanism (2/2)

- Computing data register (*CD_REG*).
- Scaling factor register (*SF_REG*).
  - SF register size will not be restricted.
- Reduce memory access.



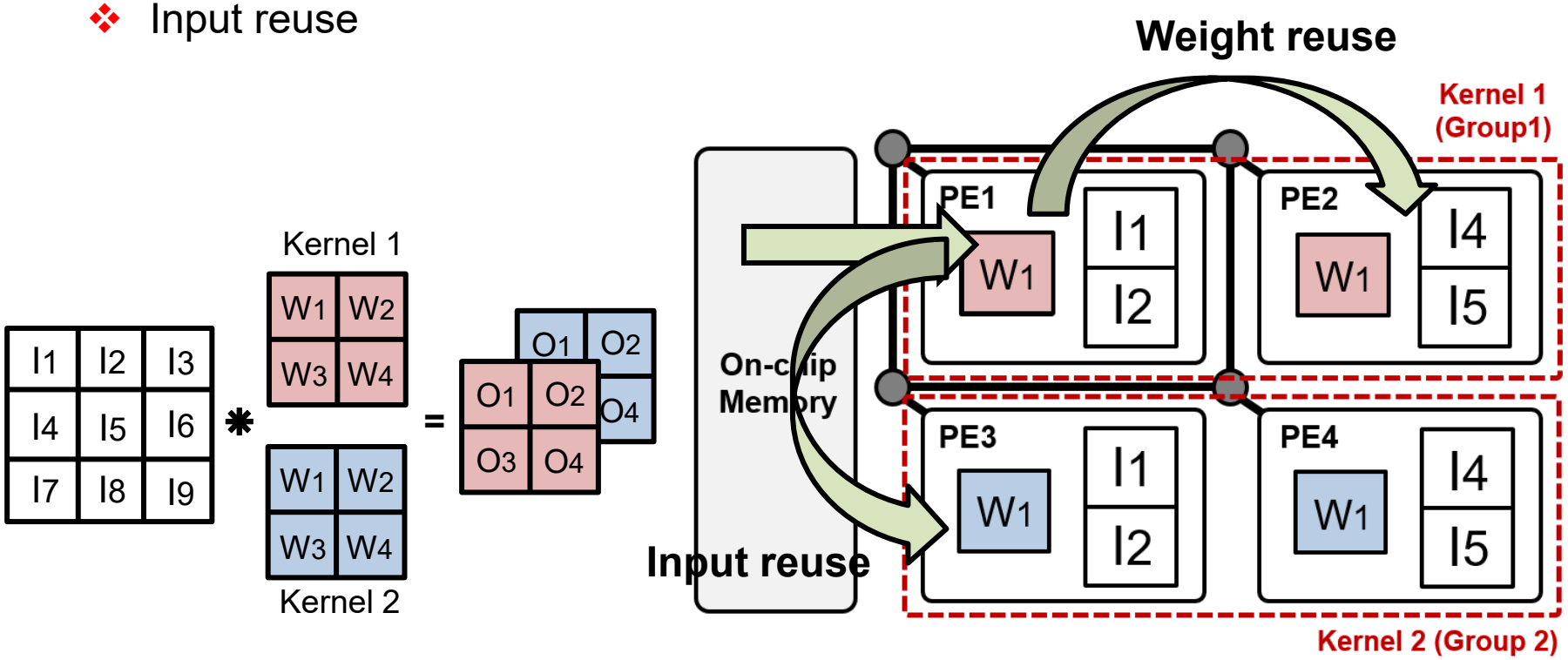
$$O_1 = (I_1 \times W_1) + (I_2 \times W_2) + (I_4 \times W_3) + (I_5 \times W_4)$$
$$O_2 = (I_2 \times W_1) + (I_3 \times W_2) + (I_5 \times W_3) + (I_6 \times W_4)$$
$$O_3 = (I_4 \times W_1) + (I_5 \times W_2) + (I_7 \times W_3) + (I_8 \times W_4)$$
$$O_4 = (I_5 \times W_1) + (I_6 \times W_2) + (I_8 \times W_3) + (I_9 \times W_4)$$

OPsum



Hybrid Data Reuse Method by Using NoC

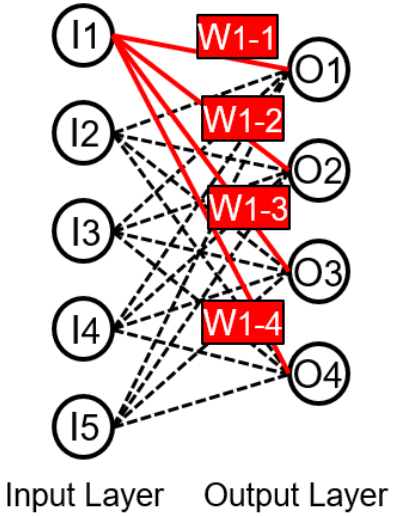
- After accessing the data from on-chip memory once, PE will share duplicated data through packet transmission.
  - Does not need to design complicated dataflow.
  - Weight reuse
  - Input reuse

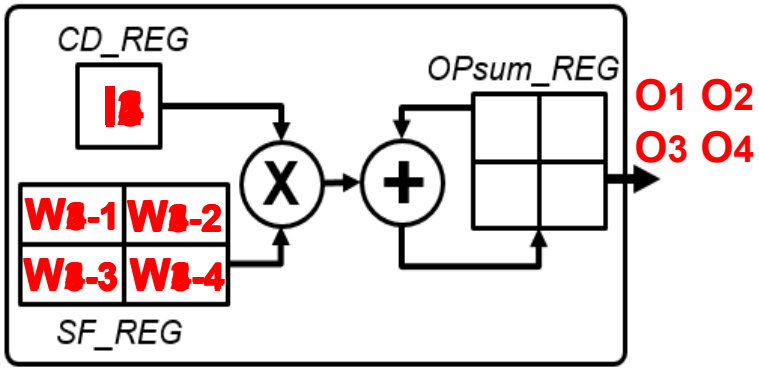


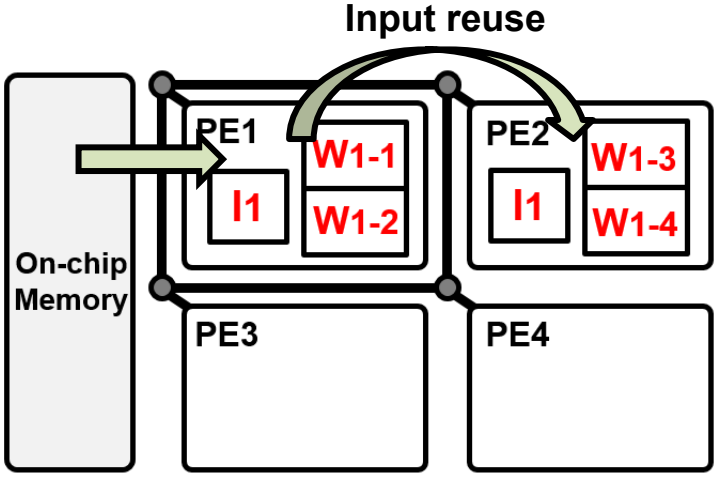
The Processing of Fully-connected Layer

- The proposed processing mechanism can also be applied in the fully-connected layer.
  - Store the input data to the *CD_REG*, and the corresponding weights to the *SF_REG*, respectively.
  - Input reuse

$$O_1 = (I_1 \times W_{1-1}) + (I_2 \times W_{2-1}) + (I_3 \times W_{3-1}) + (I_4 \times W_{4-1}) + (I_5 \times W_{5-1})$$
$$O_2 = (I_1 \times W_{1-2}) + (I_2 \times W_{2-2}) + (I_3 \times W_{3-2}) + (I_4 \times W_{4-2}) + (I_5 \times W_{5-2})$$
$$O_3 = (I_1 \times W_{1-3}) + (I_2 \times W_{2-3}) + (I_3 \times W_{3-3}) + (I_4 \times W_{4-3}) + (I_5 \times W_{5-3})$$
$$O_4 = (I_1 \times W_{1-4}) + (I_2 \times W_{2-4}) + (I_3 \times W_{3-4}) + (I_4 \times W_{4-4}) + (I_5 \times W_{5-4})$$

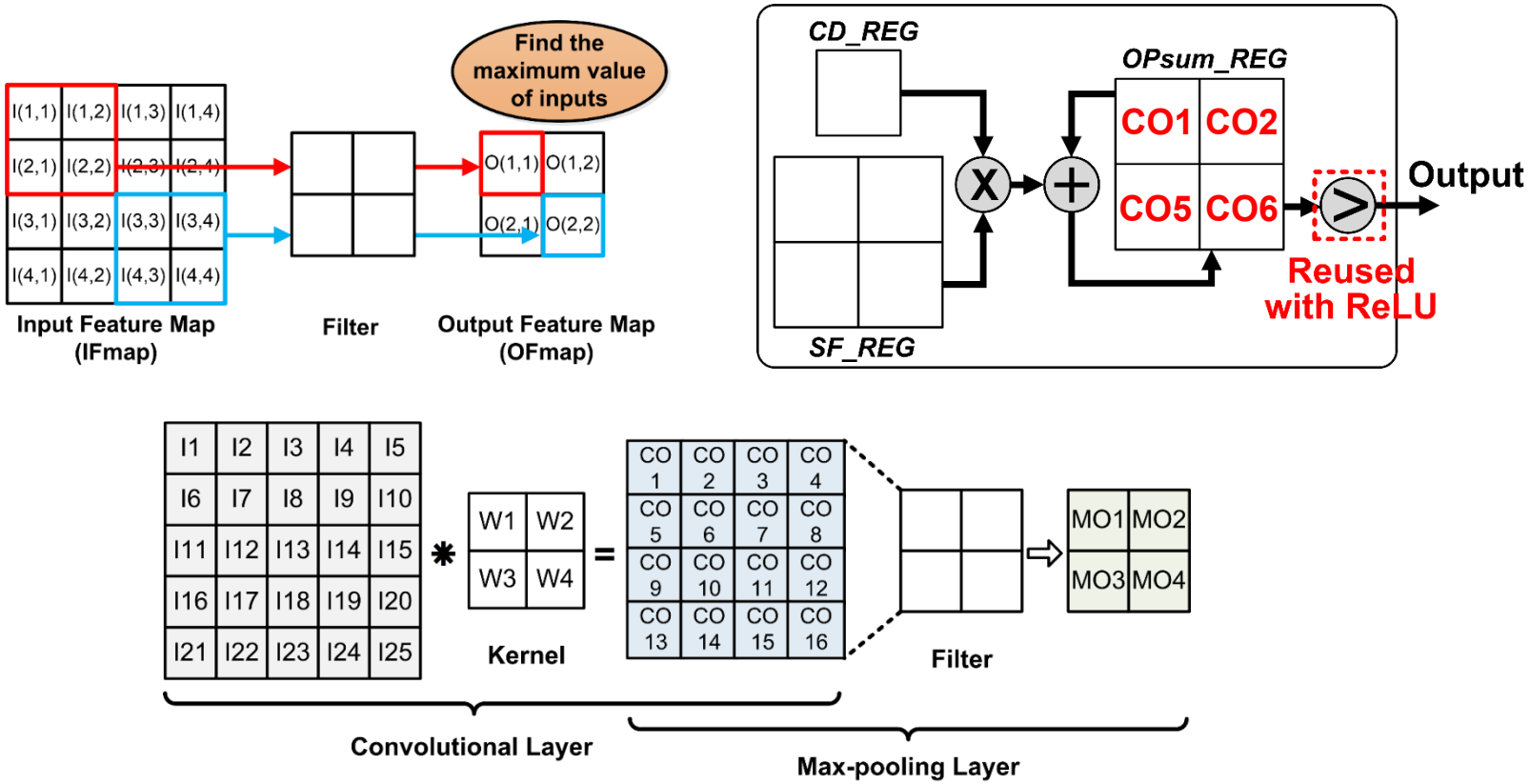






The Applicability of Max-Pooing Layer

- The proposed mechanism can be performed in the max-pooling layer.
- The *SF_REG* size will be designed as a multiple of the filter size.
- The comparator can be reused by ReLU.



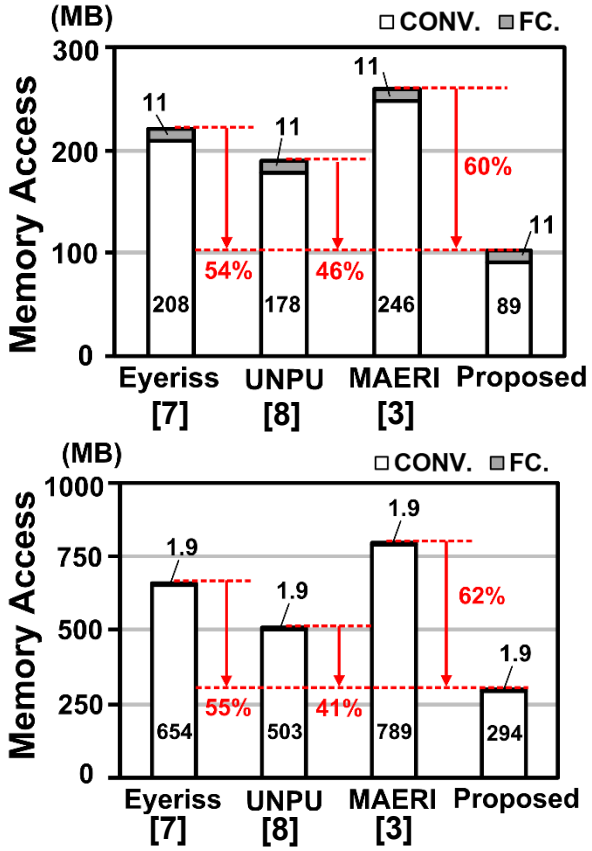
Experiment Setting

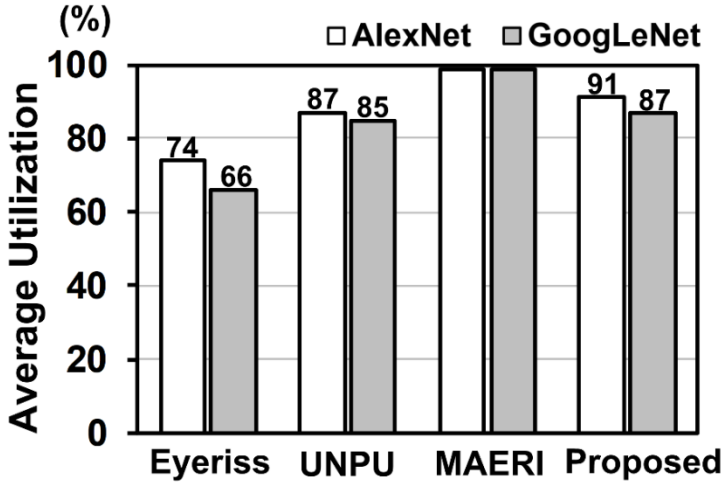
- Adopt a NoC-based DNN simulator (DNNoxim) to simulate
- Compare the utilization of the computing resource in a PE and the memory access with the related works.

	Setting
Mesh NoC size	13 x 13
SF_REG size	81
Routing method	X-Y Routing
Data Precision	16-bit fixed-point number
Target DNN Models	AlexNet GoogLeNet
Comparison	1. On-chip memory access 2. Utilization of computing resource

Experimental Results

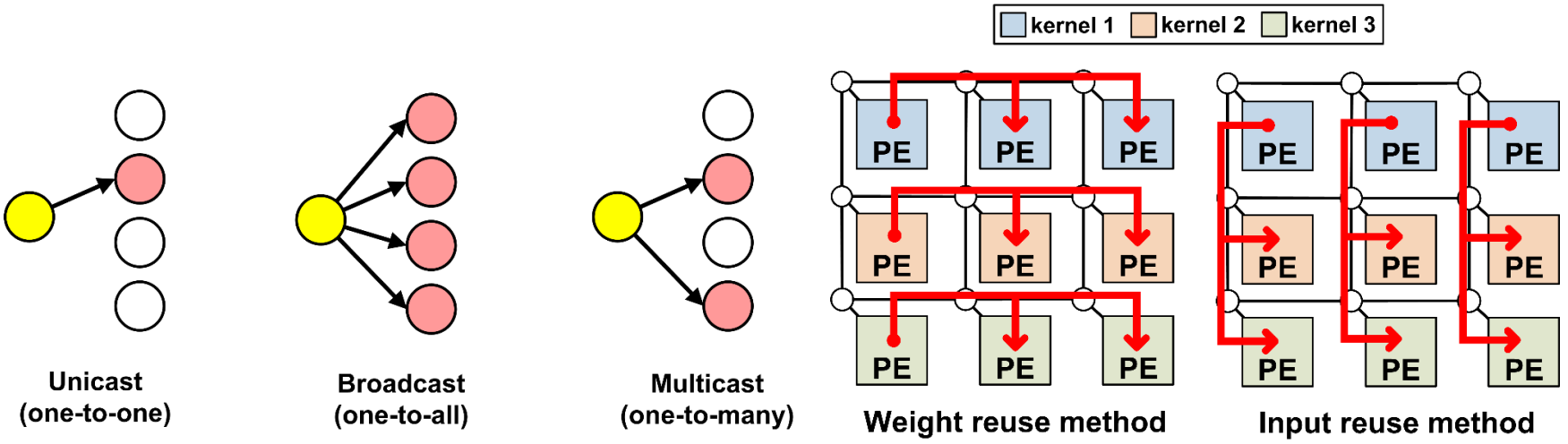
- Compared with the related works, the proposed approach can reduce 41%~62% memory access and improve the average hardware utilization by 5%~18% under the AlexNet and by 2%~21% under the GoogLeNet.





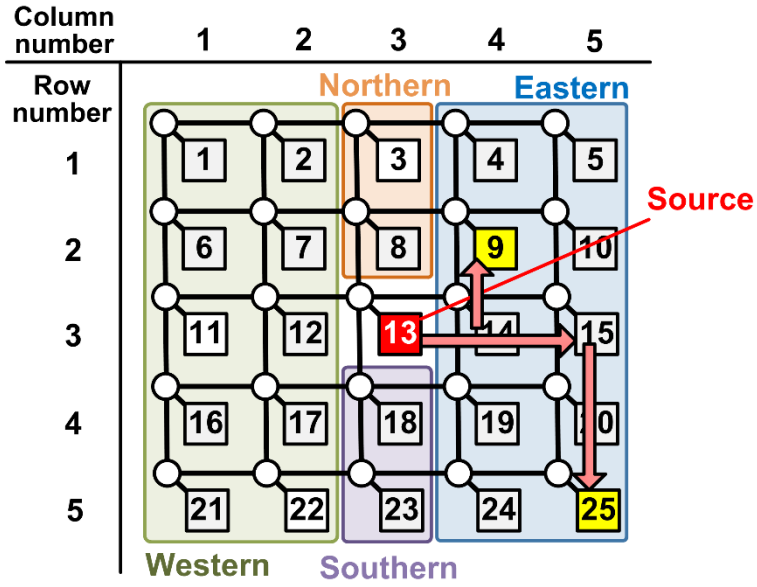
Multicast Transmission

- Although NoC brings the benefits of flexible dataflow, it suffers from the high transmission latency problem.
  - Unicast, broadcast, and multicast
- According to the proposed approaches, the transmission pattern is suitable for the multicast transmission.



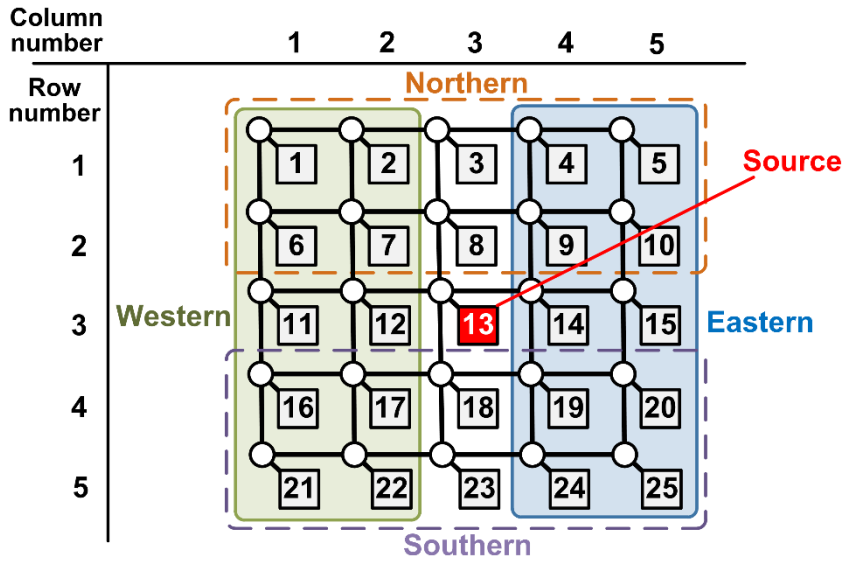
Overview of Proposed Transmission Approaches

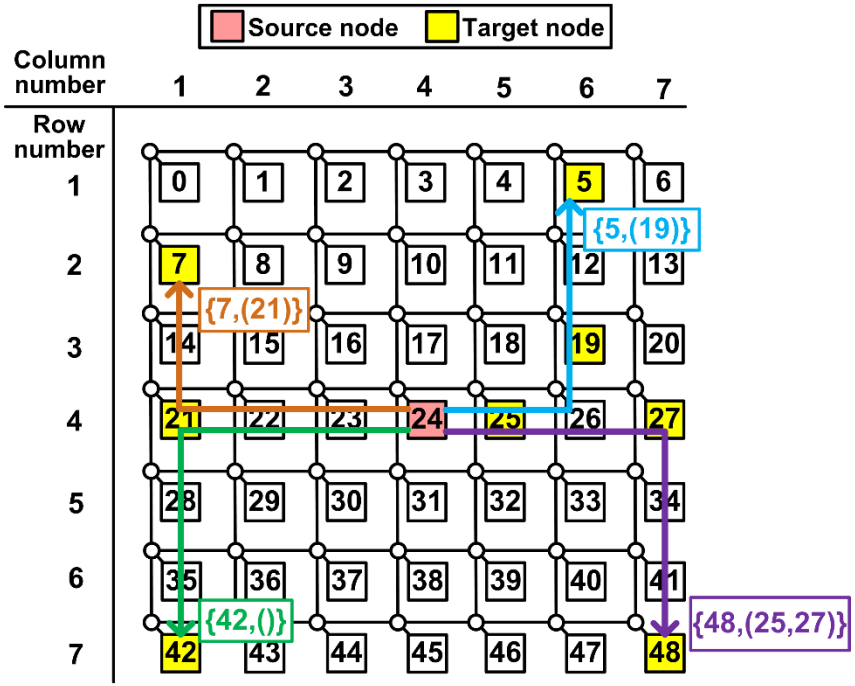
- X-Y routing-based multicast transmission algorithm
  - Based on X-Y routing and MDND [12].
- Low bandwidth-cost field representation method for multicast packet
  - The cost problem caused by traversal destination field representation.
- Input-oriented column mapping algorithm
  - Transmission distance
    - Weight-oriented mapping
    - Input-oriented mapping
  - The number of the generated packets
    - Column mapping
    - Row mapping



X-Y routing-based multicast transmissions

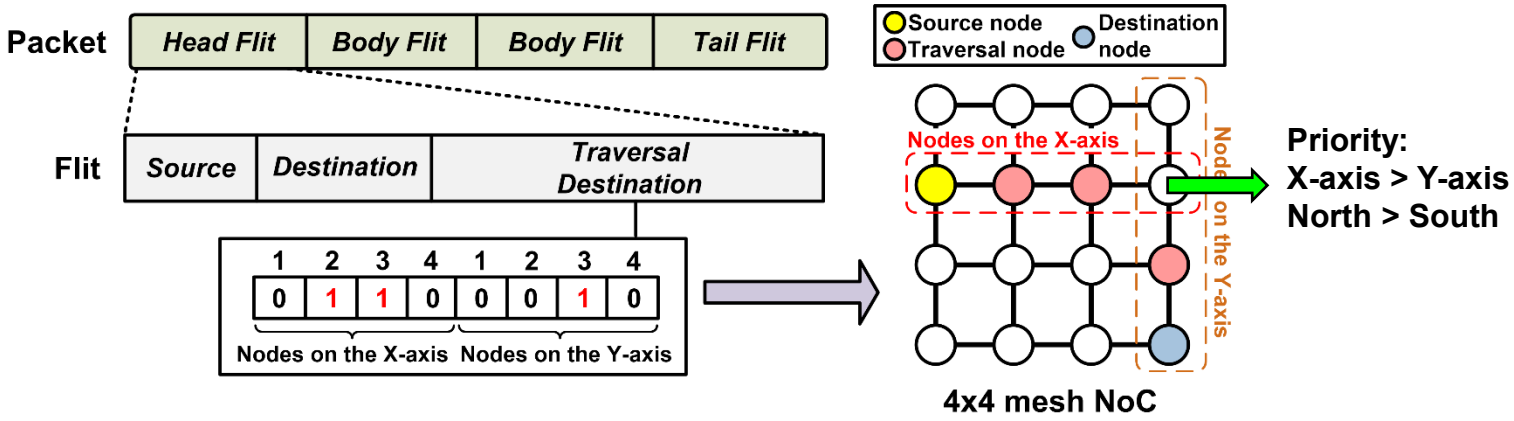
- Redefine the four region of the mesh NoC.
- Process each column (northern and southern, respectively).
- Find farthest target node.
- Mark the rest target nodes on the routing path as *travDst* field.





Low Bandwidth-cost Field Representation Method

- To reduce the number of the bits required by the *travDst* field.
  - Encode the X-axis and Y-axis nodes that will be passed on the routing path.

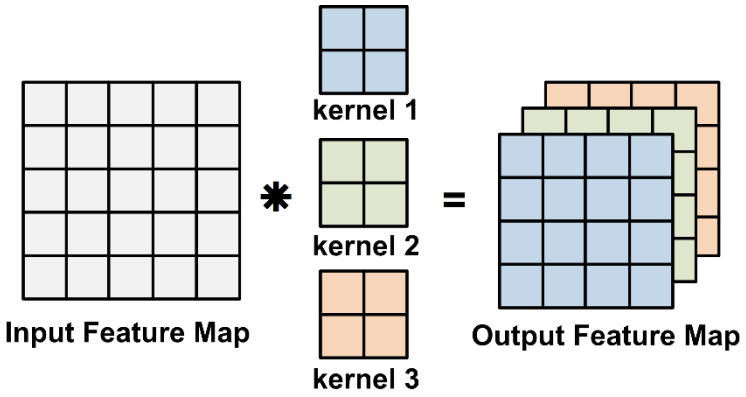


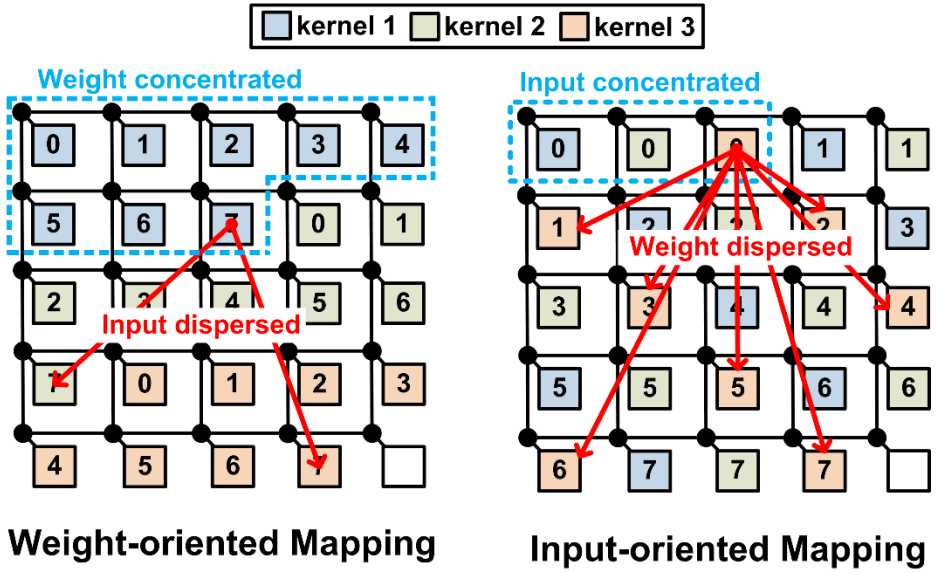
- Compared with the general method, the proposed method only requires *2N* bits while NoC size is *N*×*N*.
  - *N* bits for X-axis nodes.
  - *N* bits for Y-axis nodes.

***N*×*N* Mesh NoC**		
	General	**Proposed**
***Src.* field**	*S* bits	*S* bits
***Dst.* field**	*D* bits	*D* bits
***travDst.* field**	N^2 bits	*2N* bits
Total bits of head flit	***S+D+N²* bits**	***S+D+2N* bits**

Input-oriented Column Mapping Algorithm

- Input-oriented mapping
  - According to the proposed weight-wise NN processing mechanism, the length of input packet is longer than the length of weight packet.
  - Map the operation which require the same input data more closely.
  - Reduce the transmission distance required to send the input multicast packet.





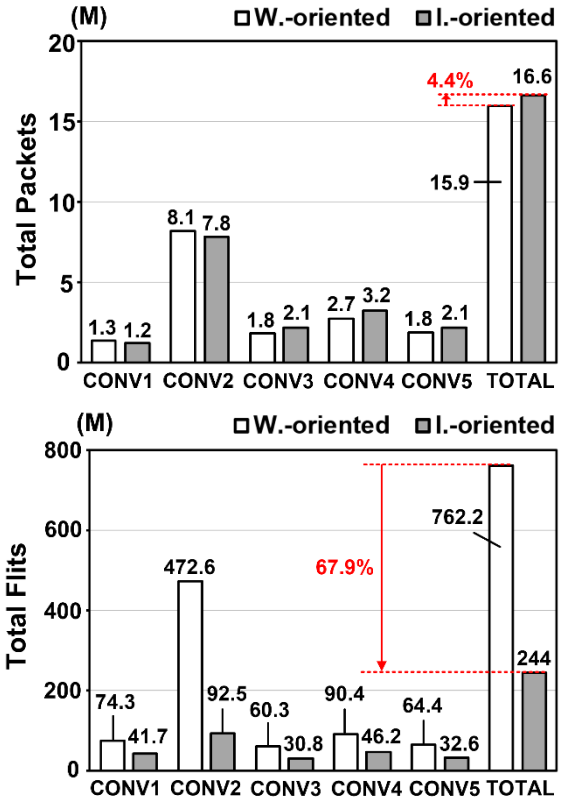
Experimental Setup

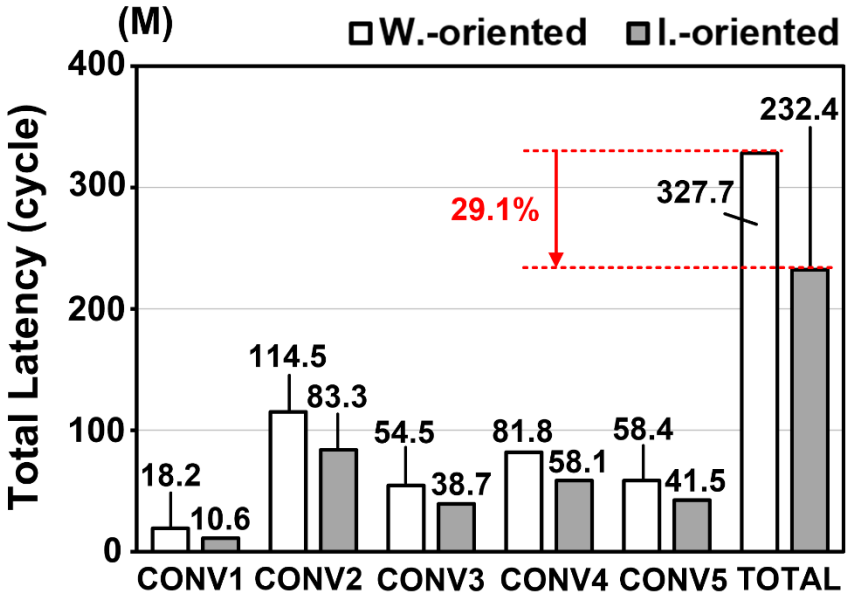
- Evaluation of the proposed multicast transmission.

	Setting	
Mesh NoC size	13 x 13	
***SF_REG* size**	81	
Routing method	X-Y Routing	
Application	AlexNet (CONV. Layers)	
	Experiment 1	**Experiment 2**
Comparison	1. Input-oriented mapping 2. Weight-oriented mapping 3. Row mapping 4. Column mapping	1. w/ multicast transmission 2. w/o multicast transmission
Observation	1. Total number of packets 2. Total number of flits 3. Total latency	

Experimental Results

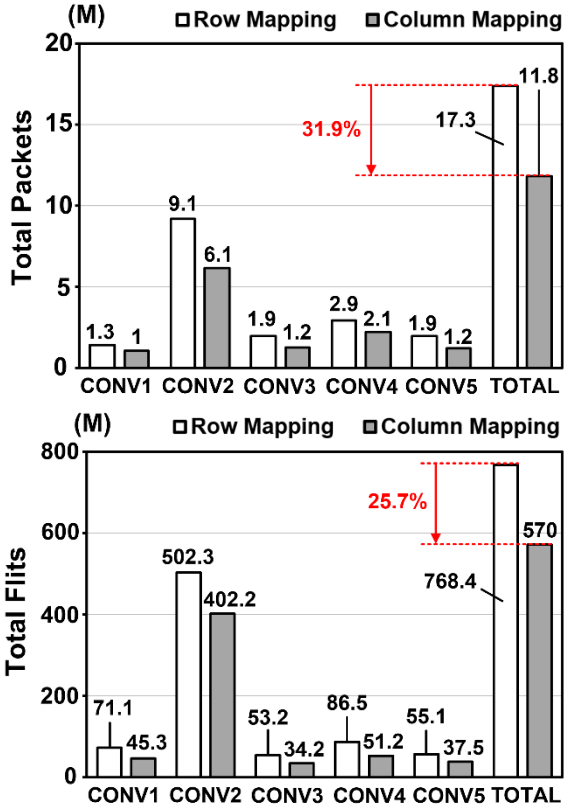
- Analysis between weight-oriented and input-oriented mapping.
  - Input-oriented mapping will increase average 4.4% packets.
  - Input-oriented mapping can reduce average 67.9% flits.
  - Input-oriented mapping can reduce average 29.1% latency.

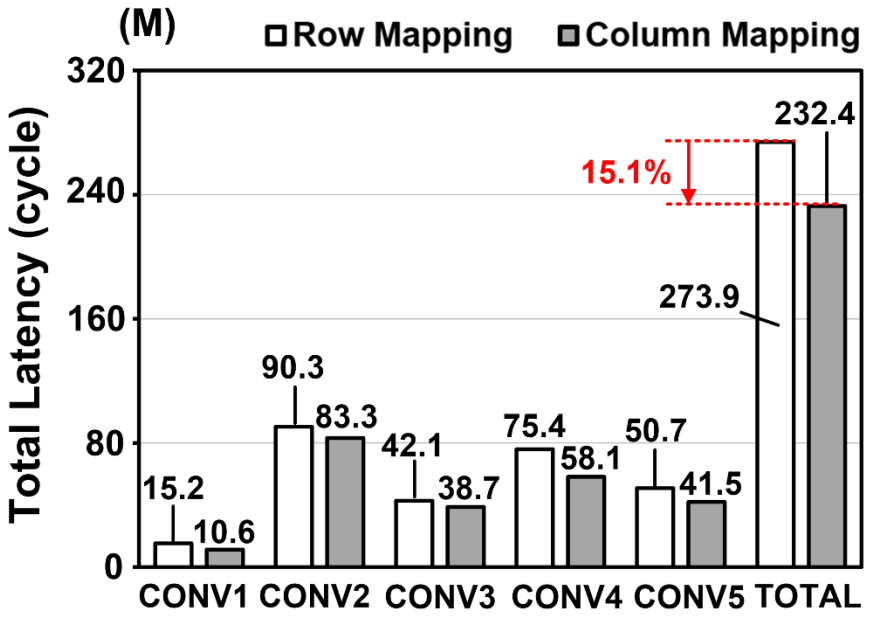




Experimental Results

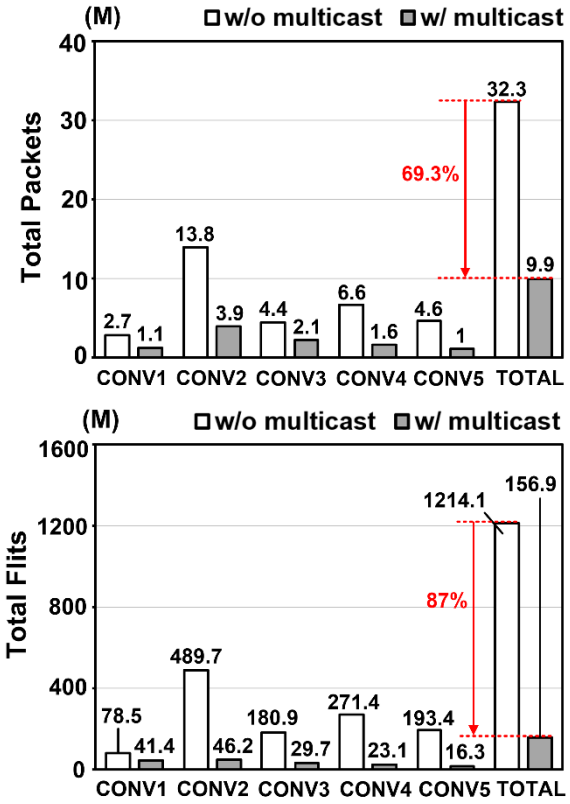
- Analysis between row and column mapping.
  - Column mapping can reduce average 31.9% packets.
  - Column mapping can reduce average 25.7% flits.
  - Column mapping can reduce average 15.1% latency.

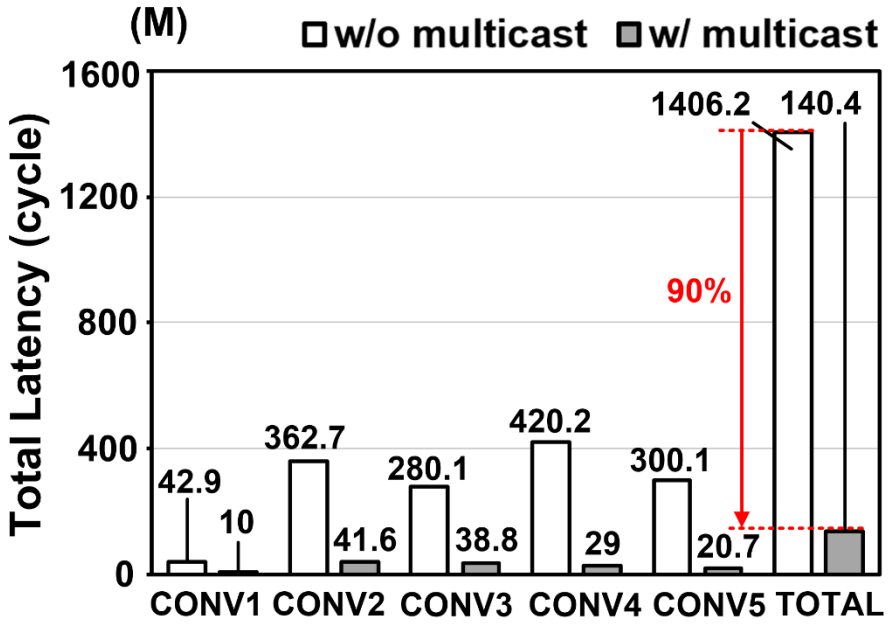




Experimental Results

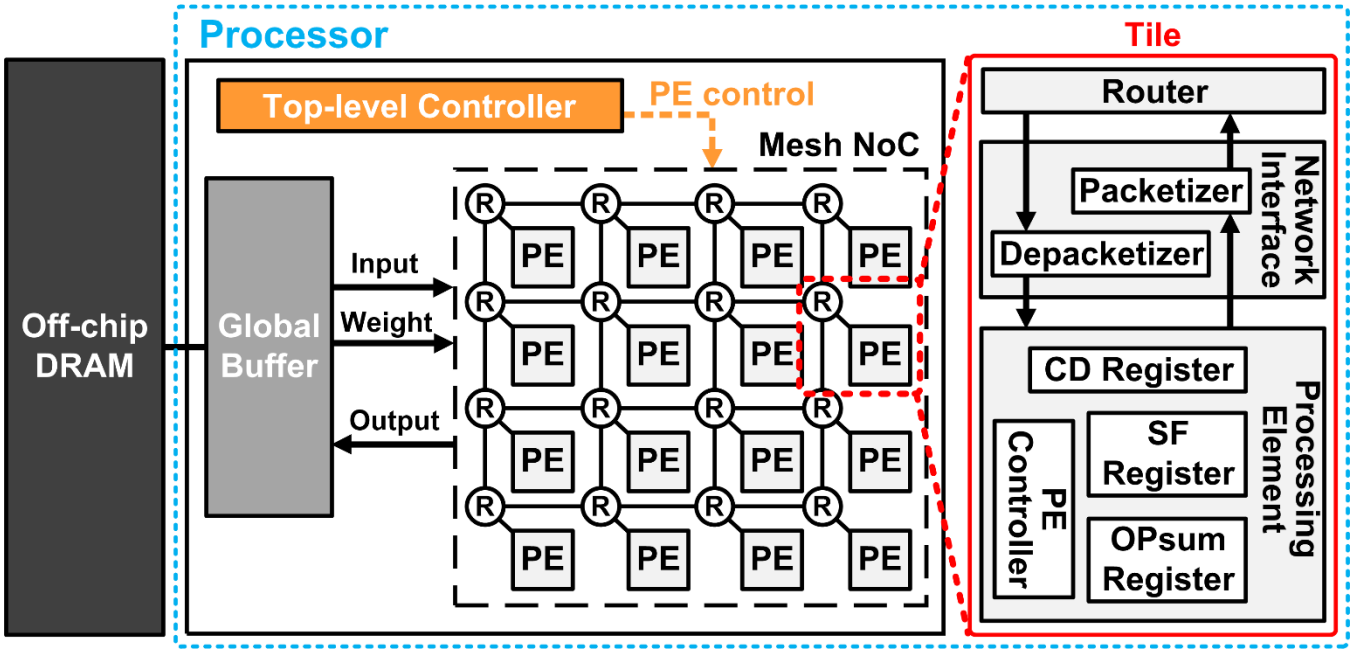
- Analysis between multicast and w/o multicast.
  - Proposed approach can reduce average 69.3% packets.
  - Proposed approach can reduce average 87% flits.
  - Proposed approach can reduce average 90% latency.





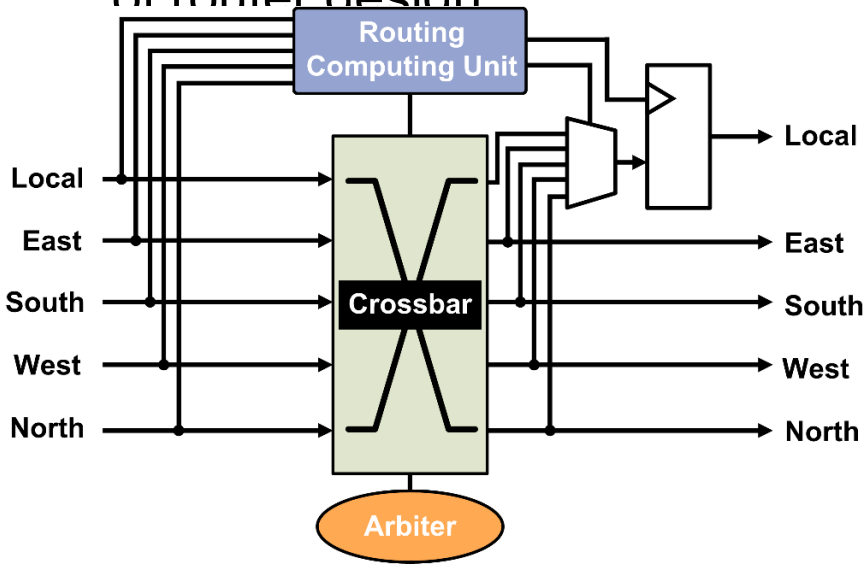
Hardware Architecture Overview

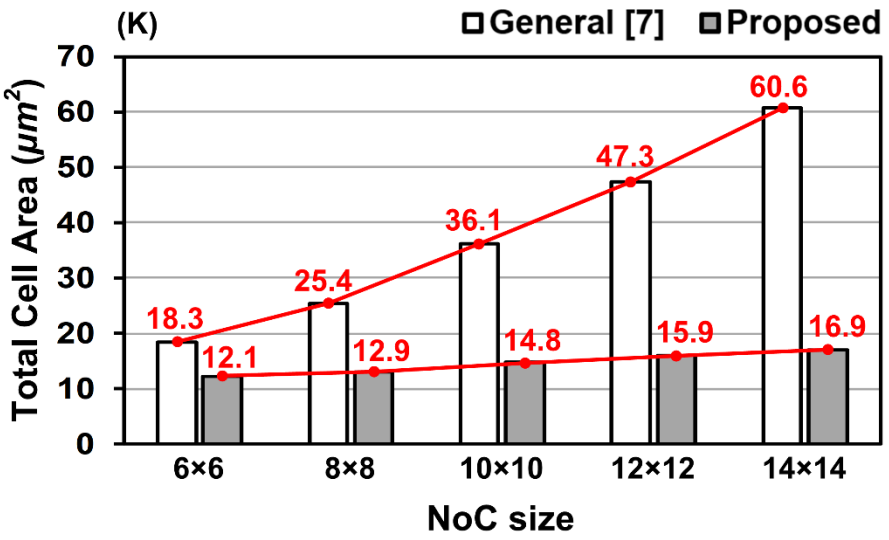
- The control signal sent from the top-level controller.
- The tile includes router, network interface (NI), and processing element (PE).
- Process the DNN model layer by layer.



Router Architecture

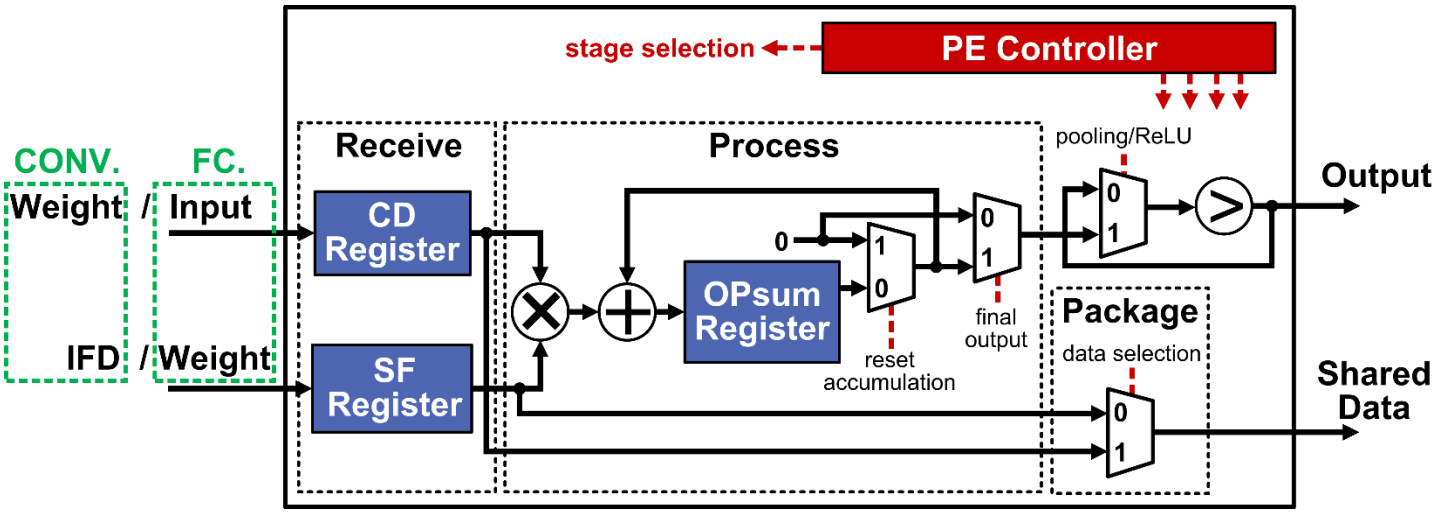
- X-Y routing algorithm.
- Proposed multicast transmission.
- The design is synthesized based on TSMC 40nm process technology.
- Compared with general multicast function (i.e., record everything in the header), the proposed approach can bring smaller area overhead of router design.





Processing Element

- Three operation stages.
  - Receiving stage
  - Processing stage
  - Packaging stage
- The *CD/SF_REG* will store the input/weight data depending on the operational layer.



Implementation Results

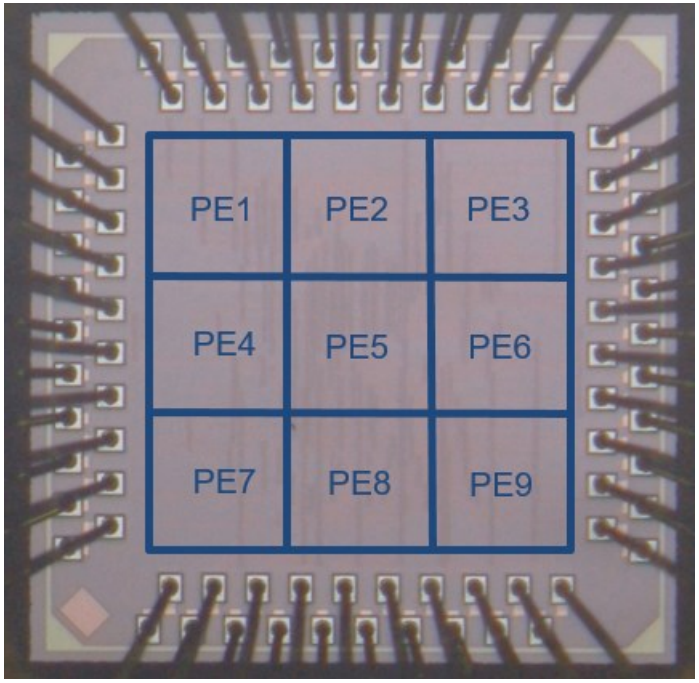
- The design is synthesized based on TSMC 40nm process technology.
- 16-bit fixed-point operation design.
- The proposed approach can save energy consumption by 47% to 61.1% compared with other works.

	Eyeriss	*UNPU*	*MAERI*	This work
Supported CONV. Kernel Shape	H: 1-12 W: 1-32	H: 1-12 W: 1-12	Unlimited	Unlimited
Gate Count	1,176k	13,680k	934k	5,512k
Energy Consumption (uJ)	859 (-54.3%)	741 (-47%)	1,008 (-61.1%)	392

The first NoC-based Reconfigurable DNN Accelerator with AXI communication protocol in Taiwan

- 3 x 3 mesh-based NoC interconnection
- Support arbitrary kernel size and shape to compute the convolution operations
- Adopt AXI4-stream communication protocol

Technology		TSMC 40nm
Area (mm^2)	Chip	1.4 x 1.4
	Core	0.84 x 0.84
	Gate count	6,871k
IO/Core VDD (v)		2.5/0.9
Clock freq (MHz)		105
Power (mW)		10.3672
Throughput (GOPS)		143.5



Conclusion

- Fundamental of DNN accelerator design
  - DNN hardware ensures information privacy, improves performance, and realizes edge AI applications.
  - The data delivery and sharing on chips becomes the performance bottleneck.
- Fundamental of Network-on-Chip (NoC) interconnection
  - NoC-based interconnection provide a flexible and efficient interconnection way to build a DNN accelerator
  - It becomes a popular way in the industry to build a multi-core system
- NoC-based DNN design
  - Lego-based DNNoC design paradigm
  - A flexible NoC-based interconnection leverages a reconfigurable DNN deisgn