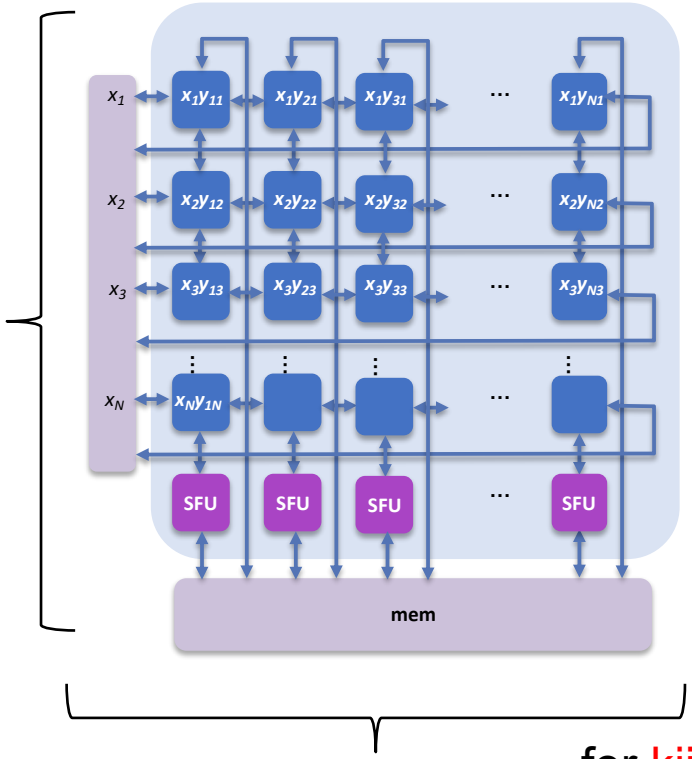
ECE284 Fall 21 W4S2

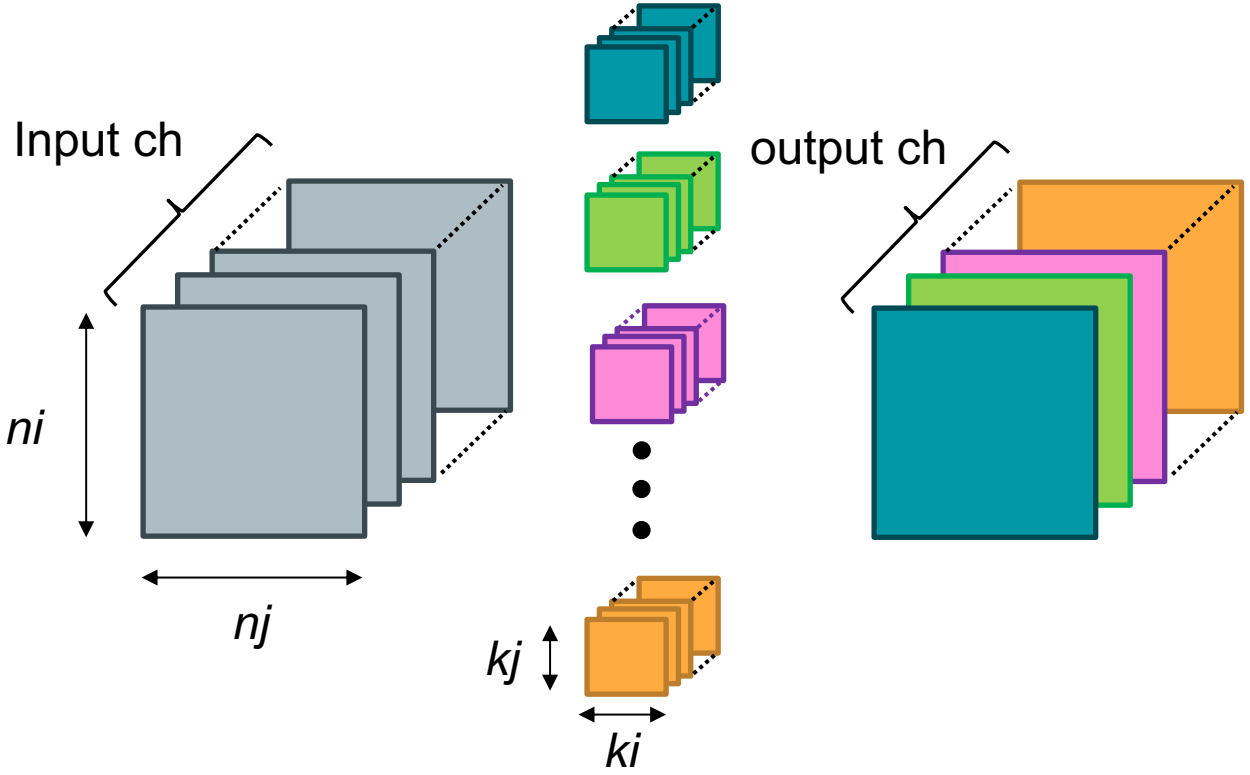
Low-power VLSI Implementation for Machine Learning

Prof. Mingu Kang

UCSD Computer Engineering

Tiling of the Workload (e.g., in/output channel # = 256)





Weight stationary

```
for kij = 1:9 (time)
  for out_tile = 0:3 (time)
    for in_tile = 0:3 (time)
      for out_ch = 64*out_tile:64*(out_tile+1) -1 (col #)
        for in_ch = 64*in_tile:64*(in_tile+1) -1 (row #)
          for nij = 0:255 (time for horizontal input)
            psum(in_tile, out_tile, out_ch, kij, nij)  += w(in_tile, out_tile, kij)
                                                        * x(in_tile, in_ch, nij)
```

[Example1] VGGNet_Hardware_Mapping

- VGGNet 3rd layer
- Weight stationary with an array size = 64 X 64
- Input and output channels are both 64
- Goals is to format input, psum, and output in the following form:
- Input (a_pad) format: [ic index (row #), nij (time step)]
- psum (psum) format: [oc index (col #), nij (time step), kij]
- Output (out) format: [oc index, o_nij]

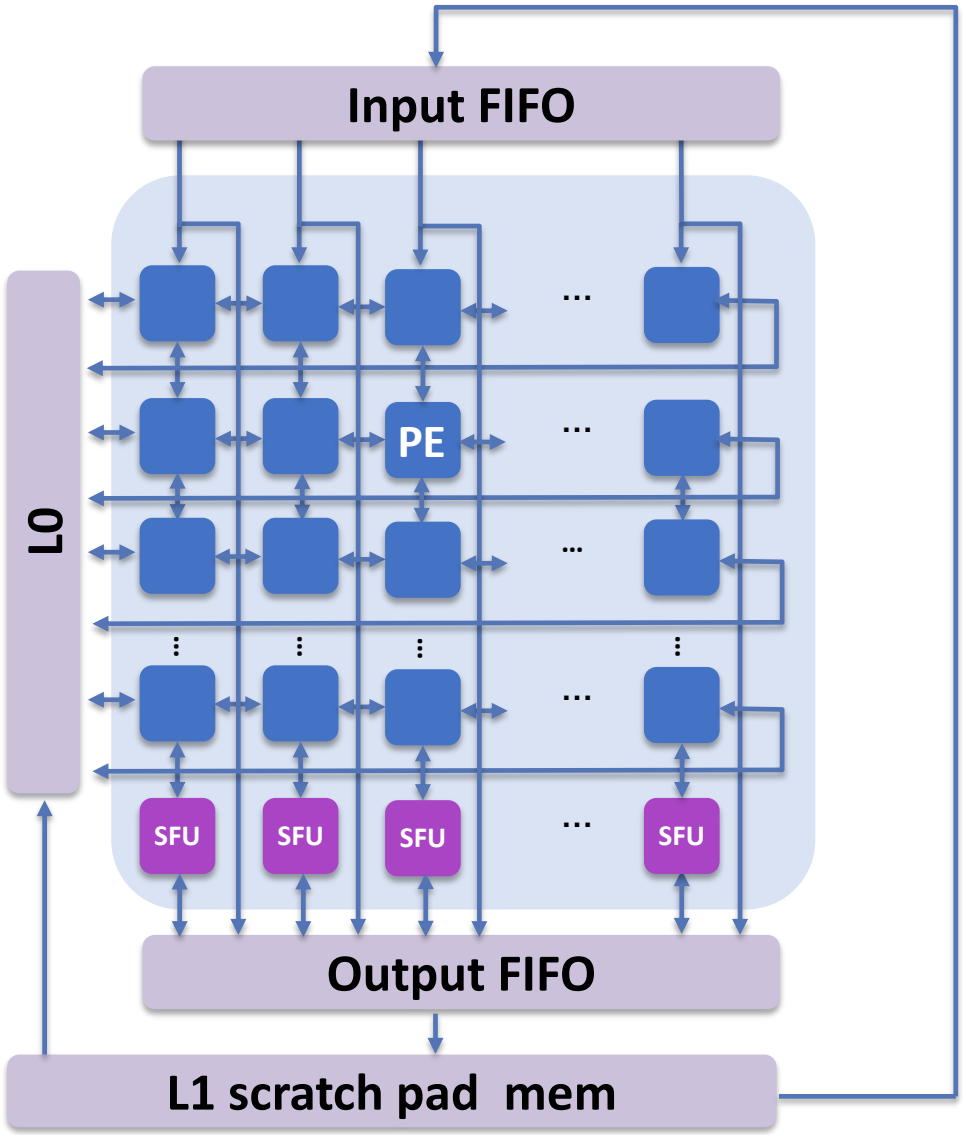
Abbreviation - ic: input channel, oc: output channel, o_nij: nij index for output

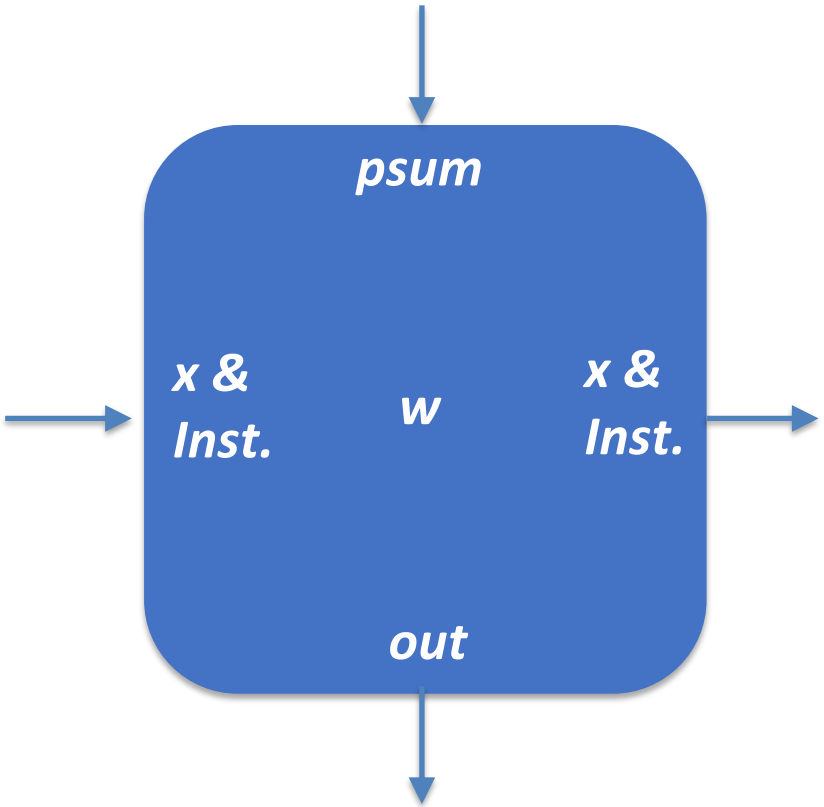
[HW_prob1] VGGNet_Hardware_Mapping_Tiling

- VGGNet 3rd layer
- Weight stationary with an array size = 16 X 16
- Input and output channels are both 64
- Needs to be tiled into 16 pieces
- Goals is to format input, psum, and output in the following form:
- Input (a_tile) format: [ic_tile, oc_tile, ic index (row #), nij (time step)]
- psum (psum) format: [ic_tile, oc_tile, oc index (col #), nij (time step), kij]
- Output (out) format: [oc index, o_nij]

Abbreviation - ic: input channel, oc: output channel, o_nij: nij index for output

Complete Architecture and Instruction Flow

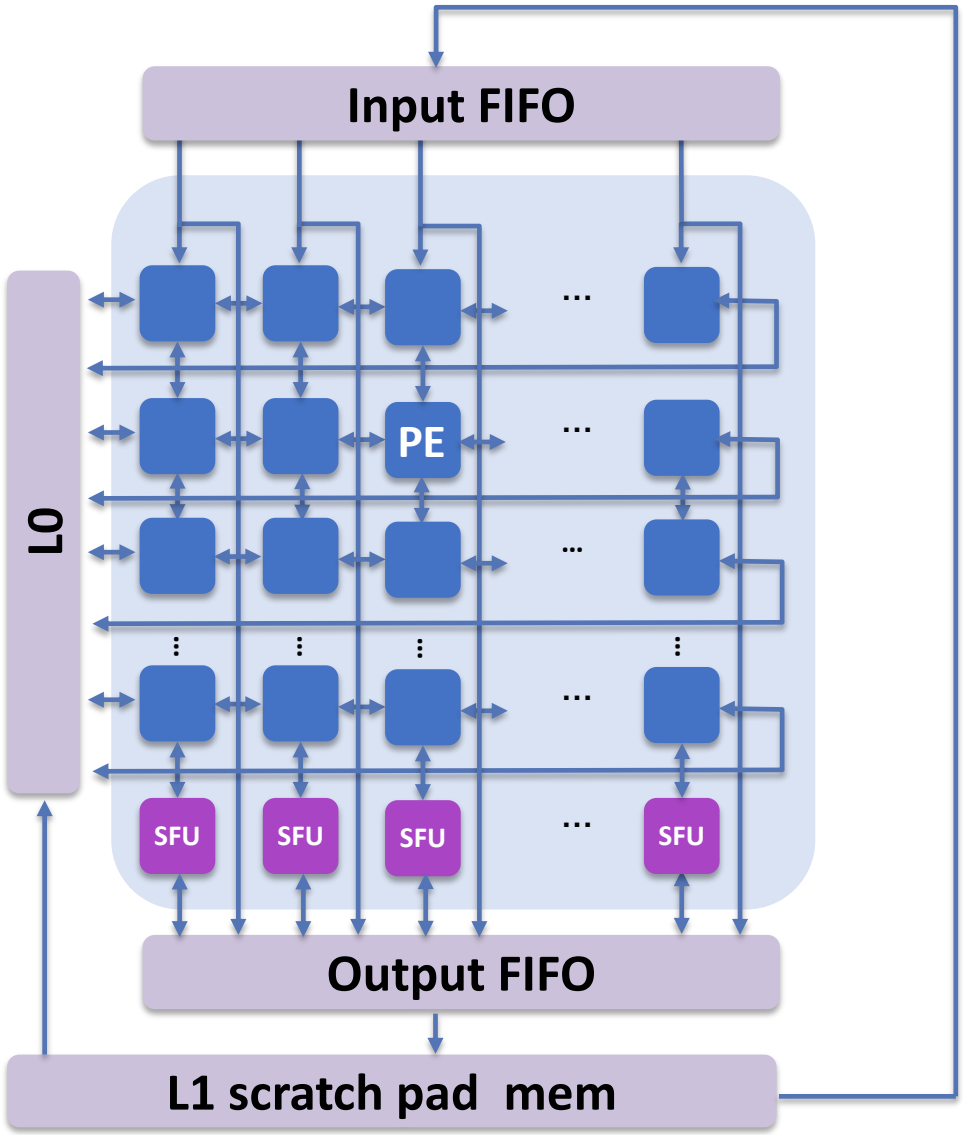




Data and Instruction flow

J.Oh, S. Lee, M. Kang, at al., “A 3.0 TFLOPS 0.62V Scalable Processor Core for High Compute Utilization AI Training and Inference”, VLSI symp 2020

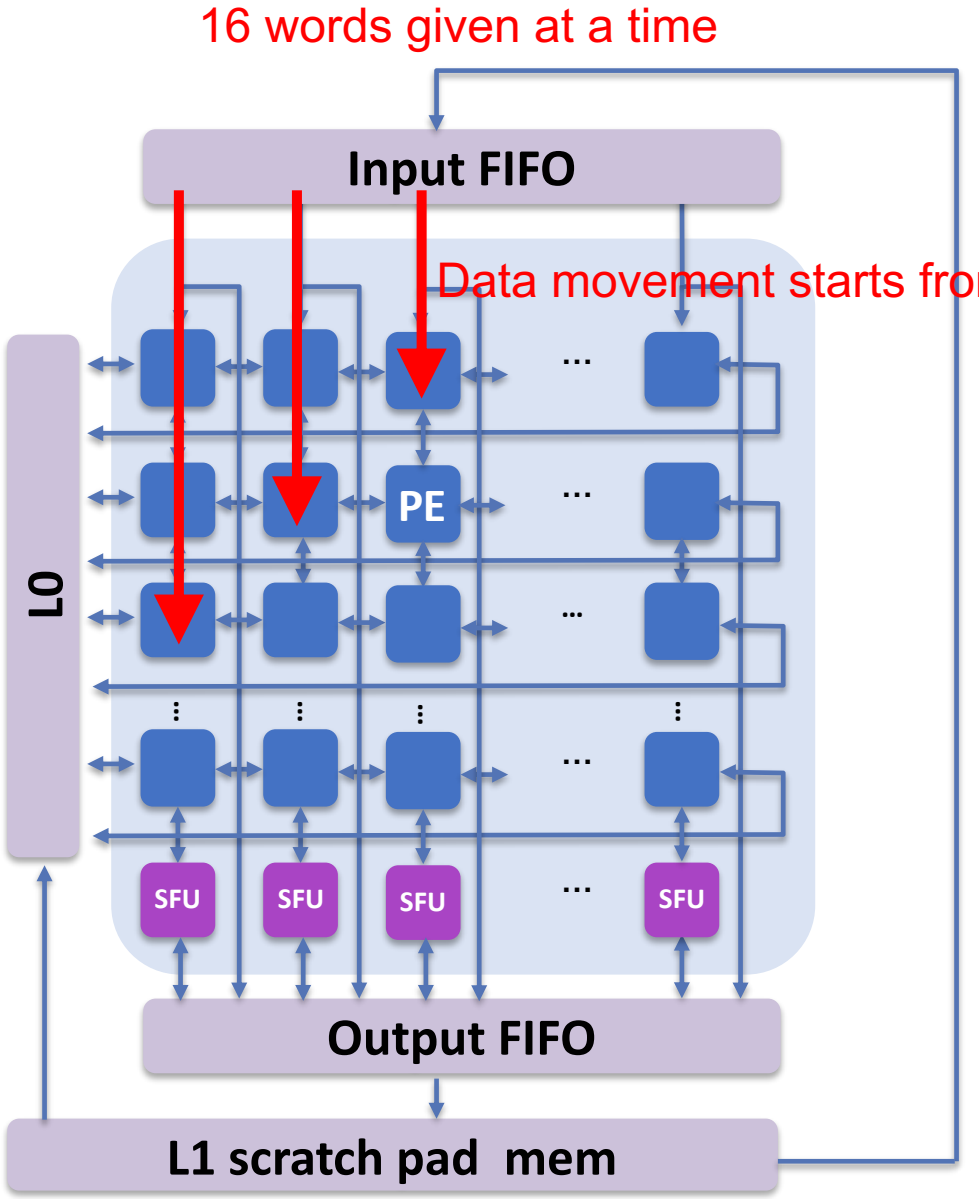
Processing Stage (Convolution)



Processing stage

1. L1 scratch pad loading (e.g., from DRAM)
2. Kernel data loading to PE register (via IFIFO)
3. L0 data loading
   (simultaneously with Execution)
4. Execution with PEs (psum computation)
5. psum movement to L1 scratch pad (via OFIFO)
   (simultaneously with Execution)
6. Accumulation in SFU
   (psum brought back from L1 scratch pad)
   Then, output store in L1 scratch pad

Kernel Data Loading (with 16 X 16 array)



Note: all data packets (word) received at a time from L1 scratch pad, but outputs at different timing.
-> IFIFO is needed

1st cyc $w_{ic0,oc0}, w_{ic0,oc1}, w_{ic0,oc2}, \dots w_{ic0,oc15}$

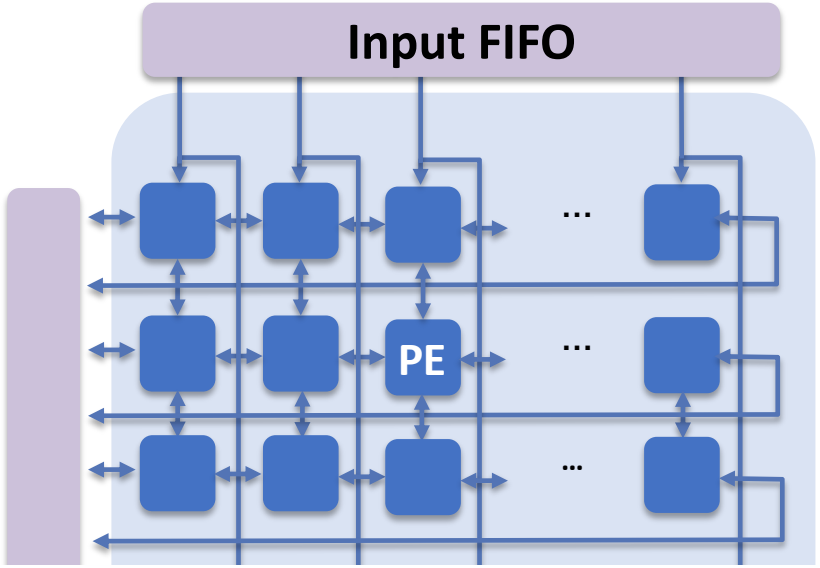
2nd cyc $w_{ic1,oc0}, w_{ic1,oc1}, w_{ic1,oc2}, \dots w_{ic1,oc15}$

…

14th cyc $w_{ic14,oc0}, w_{ic14,oc1}, w_{ic14,oc2}, \dots w_{ic14,oc15}$

15th cyc $w_{ic15,oc0}, w_{ic15,oc1}, w_{ic15,oc2}, \dots w_{ic15,oc15}$

Kernel Data Loading (with 16 X 16 array)



cyc0

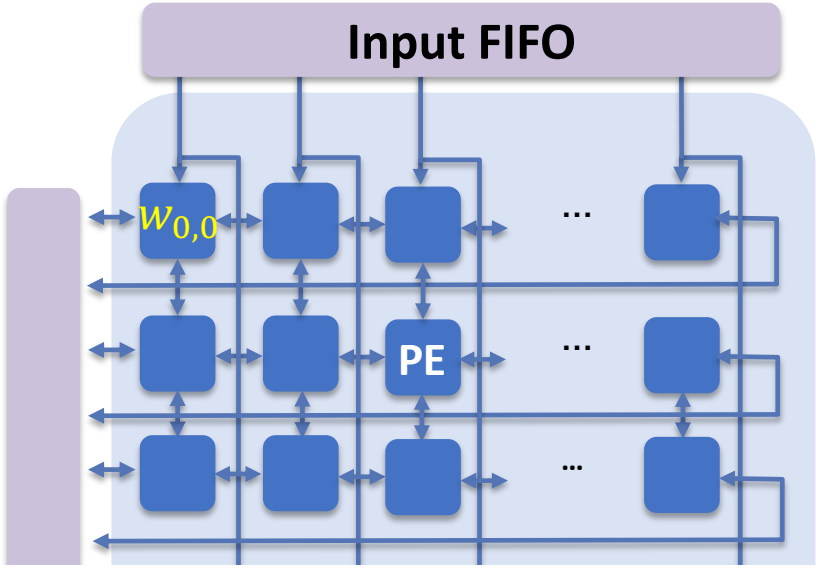
IFIFO Contents

cyc0

$w_{ic0,oc0}$	$w_{ic0,oc1}$	$w_{ic0,oc2}$	…	$w_{ic0,oc15}$

Read

Kernel Data Loading (with 16 X 16 array)



cyc1

IFIFO Contents

cyc0

$w_{ic0,oc0}$	$w_{ic0,oc1}$	$w_{ic0,oc2}$		$w_{ic0,oc15}$

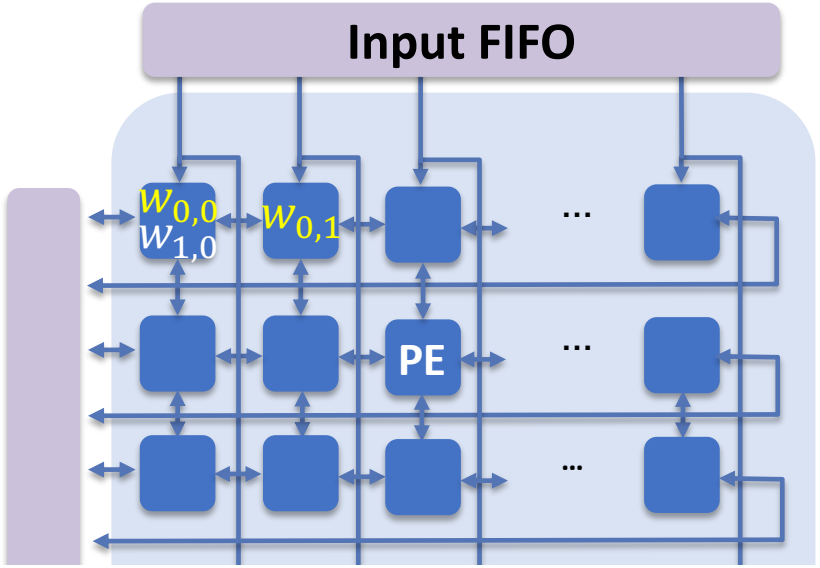
cyc1

$w_{ic1,oc0}$	$w_{ic1,oc1}$	$w_{ic1,oc2}$		$w_{ic1,oc15}$
	$w_{ic0,oc1}$	$w_{ic0,oc2}$		$w_{ic0,oc15}$

Read ⇨ Read

Yellow text: captured in the PE's local register

Kernel Data Loading (with 16 X 16 array)



cyc2

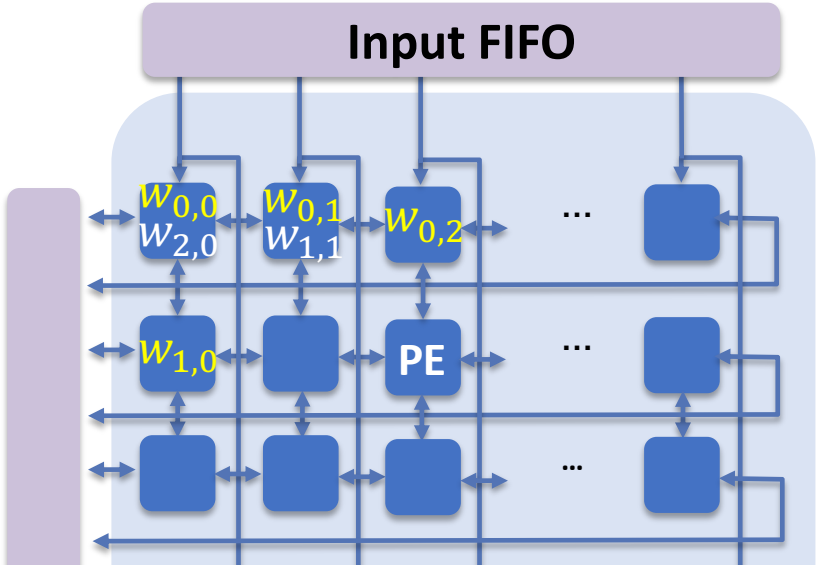
IFIFO Contents

cyc0	$w_{ic0,oc0}$	$w_{ic0,oc1}$	$w_{ic0,oc2}$		$w_{ic0,oc15}$
cyc1	$w_{ic1,oc0}$	$w_{ic1,oc1}$	$w_{ic1,oc2}$		$w_{ic1,oc15}$
		$w_{ic0,oc1}$	$w_{ic0,oc2}$		$w_{ic0,oc15}$
cyc2	$w_{ic2,oc0}$	$w_{ic2,oc1}$	$w_{ic2,oc2}$		$w_{ic2,oc15}$
		$w_{ic1,oc1}$	$w_{ic1,oc2}$		$w_{ic1,oc15}$
			$w_{ic0,oc2}$		$w_{ic0,oc15}$

Read → Read → Read

Yellow text: captured in the PE's local register

Kernel Data Loading (with 16 X 16 array)



cyc3

Yellow text: captured in the PE's local register

IFIFO Contents

cyc0

$w_{ic0,oc0}$	$w_{ic0,oc1}$	$w_{ic0,oc2}$		$w_{ic0,oc15}$

cyc1

$w_{ic1,oc0}$	$w_{ic1,oc1}$	$w_{ic1,oc2}$		$w_{ic1,oc15}$
	$w_{ic0,oc1}$	$w_{ic0,oc2}$		$w_{ic0,oc15}$

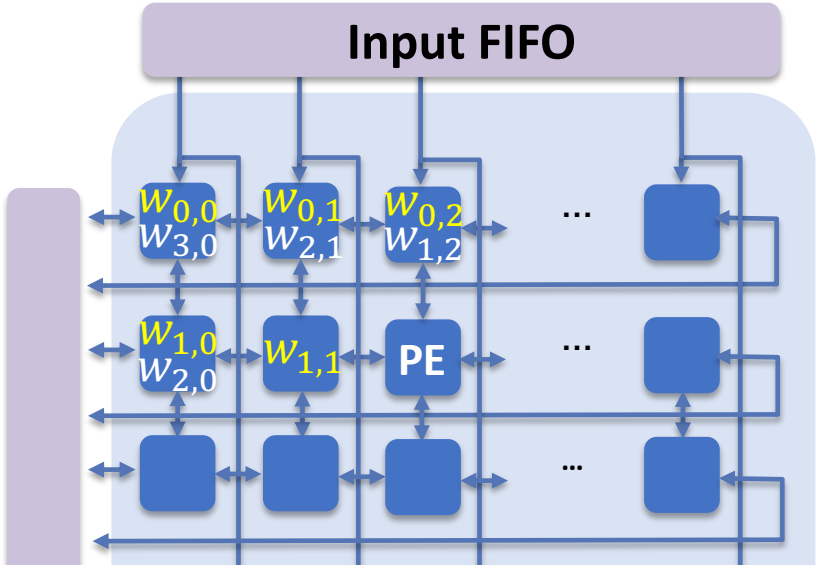
cyc2

$w_{ic2,oc0}$	$w_{ic2,oc1}$	$w_{ic2,oc2}$		$w_{ic2,oc15}$
	$w_{ic1,oc1}$	$w_{ic1,oc2}$		$w_{ic1,oc15}$
		$w_{ic0,oc2}$		$w_{ic0,oc15}$

cyc3

$w_{ic3,oc0}$	$w_{ic3,oc1}$	$w_{ic3,oc2}$		$w_{ic3,oc15}$
	$w_{ic2,oc1}$	$w_{ic2,oc2}$		$w_{ic2,oc15}$
		$w_{ic1,oc2}$		$w_{ic1,oc15}$
				$w_{ic0,oc15}$

Kernel Data Loading (with 16 X 16 array)



cyc4

Yellow text: captured in the PE's local register

IFIFO Contents

cyc2

$w_{ic2,oc0}$	$w_{ic2,oc1}$	$w_{ic2,oc2}$		$w_{ic2,oc15}$
	$w_{ic1,oc1}$	$w_{ic1,oc2}$		$w_{ic1,oc15}$
		$w_{ic0,oc2}$		$w_{ic0,oc15}$

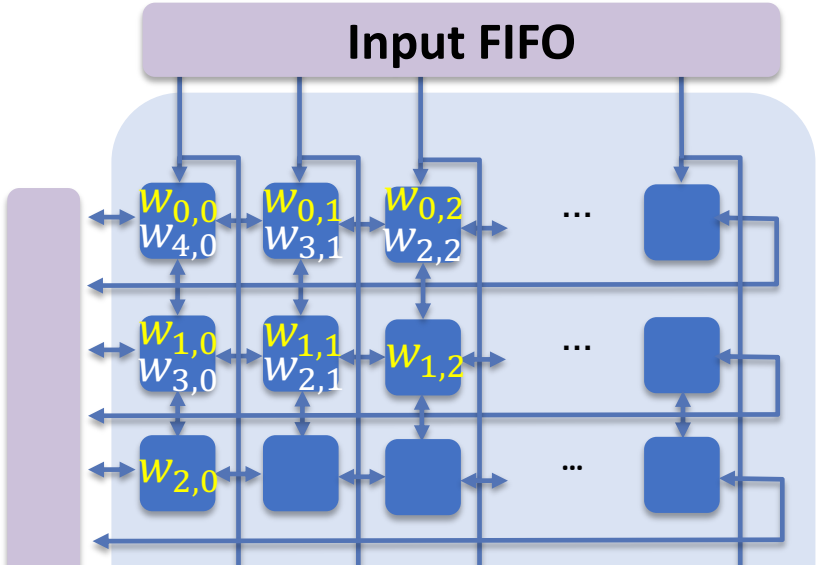
cyc3

$w_{ic3,oc0}$	$w_{ic3,oc1}$	$w_{ic3,oc2}$		$w_{ic3,oc15}$
	$w_{ic2,oc1}$	$w_{ic2,oc2}$		$w_{ic2,oc15}$
		$w_{ic1,oc2}$		$w_{ic1,oc15}$
				$w_{ic0,oc15}$

cyc4

$w_{ic4,oc0}$	$w_{ic4,oc1}$	$w_{ic4,oc2}$		$w_{ic4,oc15}$
	$w_{ic3,oc1}$	$w_{ic3,oc2}$		$w_{ic3,oc15}$
		$w_{ic2,oc2}$		$w_{ic2,oc15}$
				$w_{ic1,oc15}$
				$w_{ic0,oc15}$

Kernel Data Loading (with 16 X 16 array)



cyc5

Yellow text: captured in the PE's local register

IFIFO Contents

cyc3

$w_{ic3,oc0}$	$w_{ic3,oc1}$	$w_{ic3,oc2}$		$w_{ic3,oc15}$
	$w_{ic2,oc1}$	$w_{ic2,oc2}$		$w_{ic2,oc15}$
		$w_{ic1,oc2}$		$w_{ic1,oc15}$
				$w_{ic0,oc15}$

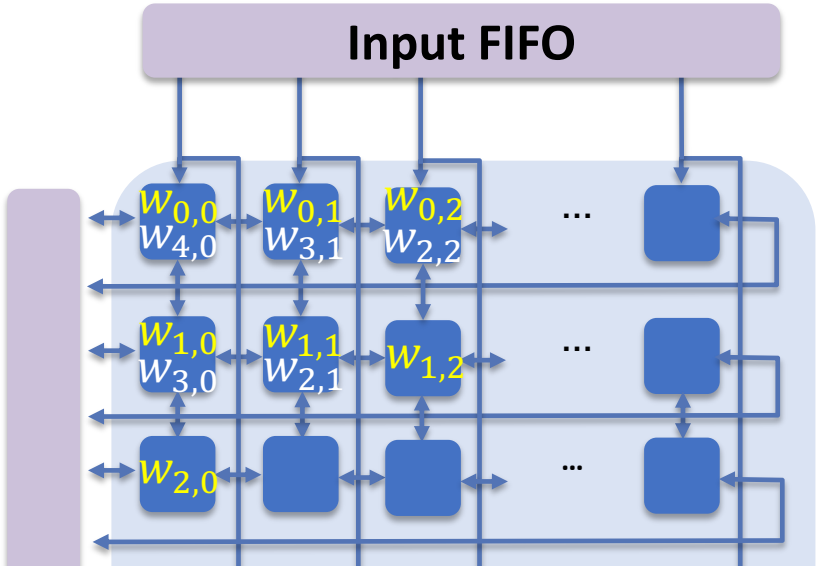
cyc4

$w_{ic4,oc0}$	$w_{ic4,oc1}$	$w_{ic4,oc2}$		$w_{ic4,oc15}$
	$w_{ic3,oc1}$	$w_{ic3,oc2}$		$w_{ic3,oc15}$
		$w_{ic2,oc2}$		$w_{ic2,oc15}$
				$w_{ic1,oc15}$
				$w_{ic0,oc15}$

cyc5

$w_{ic5,oc0}$	$w_{ic5,oc1}$	$w_{ic5,oc2}$		$w_{ic5,oc15}$
	$w_{ic4,oc1}$	$w_{ic4,oc2}$		$w_{ic4,oc15}$
		$w_{ic3,oc2}$		$w_{ic3,oc15}$
				$w_{ic2,oc15}$
				$w_{ic1,oc15}$
				$w_{ic0,oc15}$

Kernel Data Loading (with 16 X 16 array)



cyc5

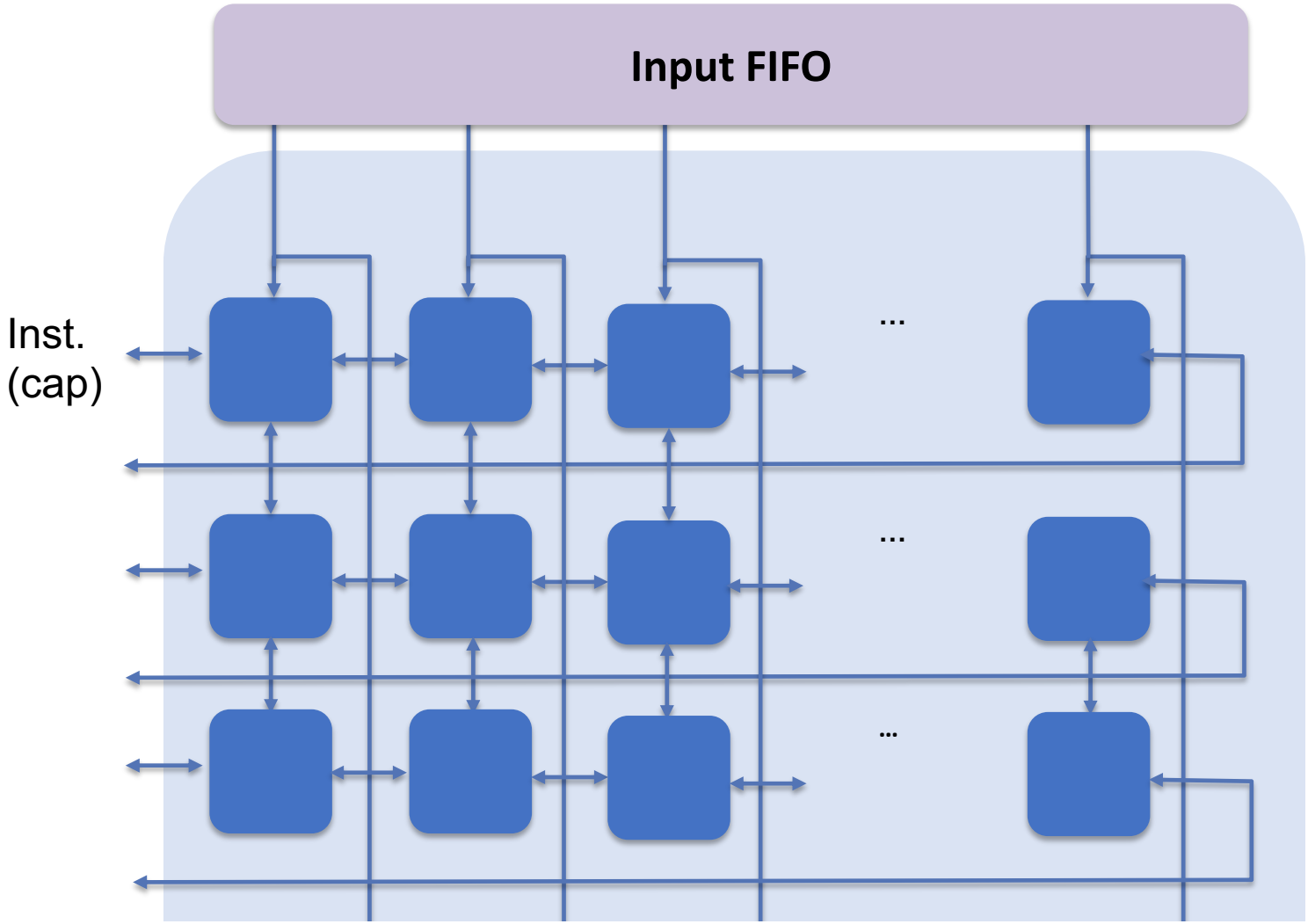
IFIFO Contents

cyc5

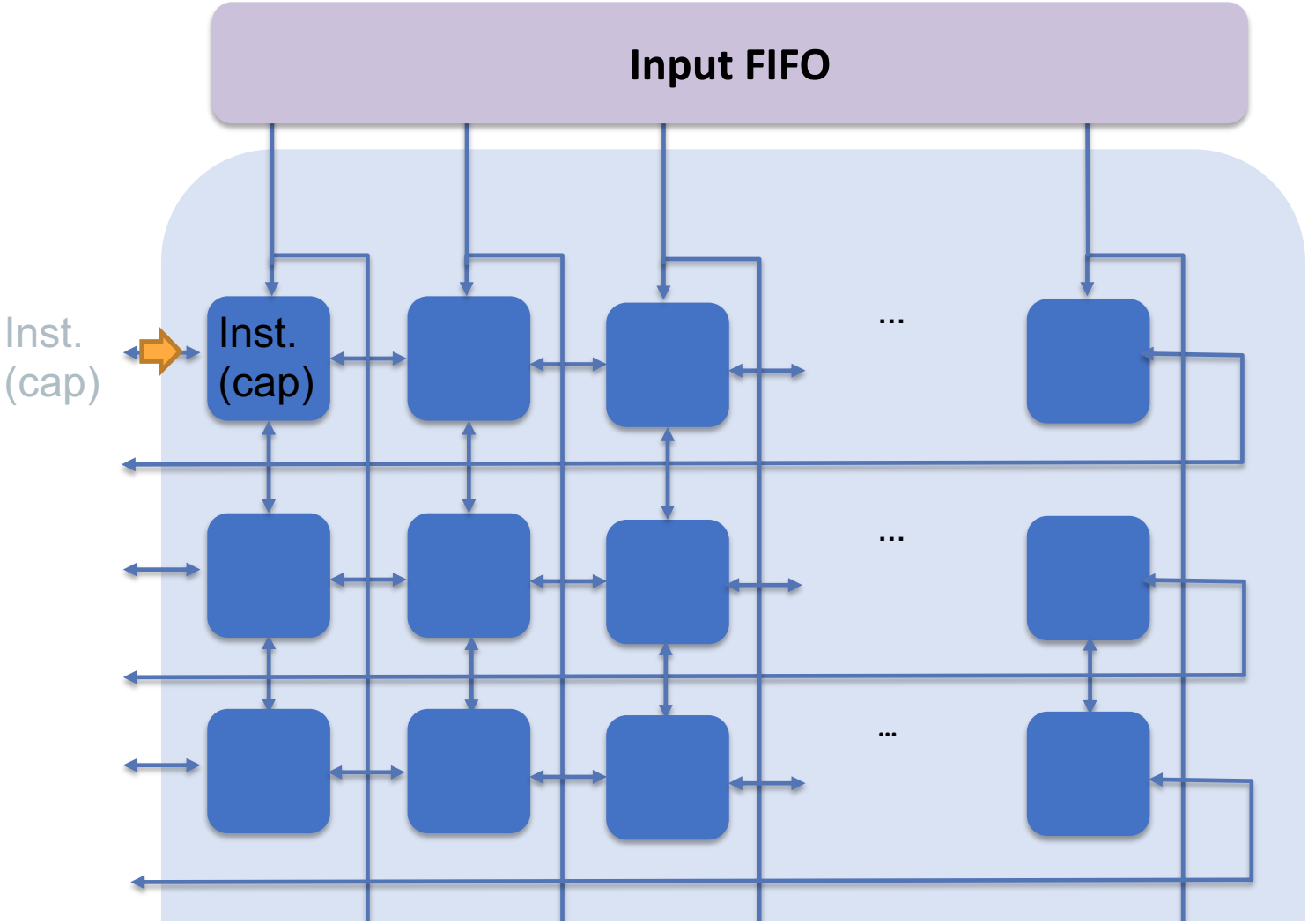
$w_{ic5,oc0}$	$w_{ic5,oc1}$	$w_{ic5,oc2}$		$w_{ic5,oc15}$
	$w_{ic4,oc1}$	$w_{ic4,oc2}$		$w_{ic4,oc15}$
		$w_{ic3,oc2}$		$w_{ic3,oc15}$
				$w_{ic2,oc15}$
				$w_{ic1,oc15}$
				$w_{ic0,oc15}$

Observation
- IFIFO: requires up to (minimum) 16 depth at the last column
- PE: captures the value at the right timing by the instruction flow
- Roughly 16 (last column start) + 16 (last element start) +16 (last element delivery)
  = 48 cycles required to fill the entire array

Instruction Flow

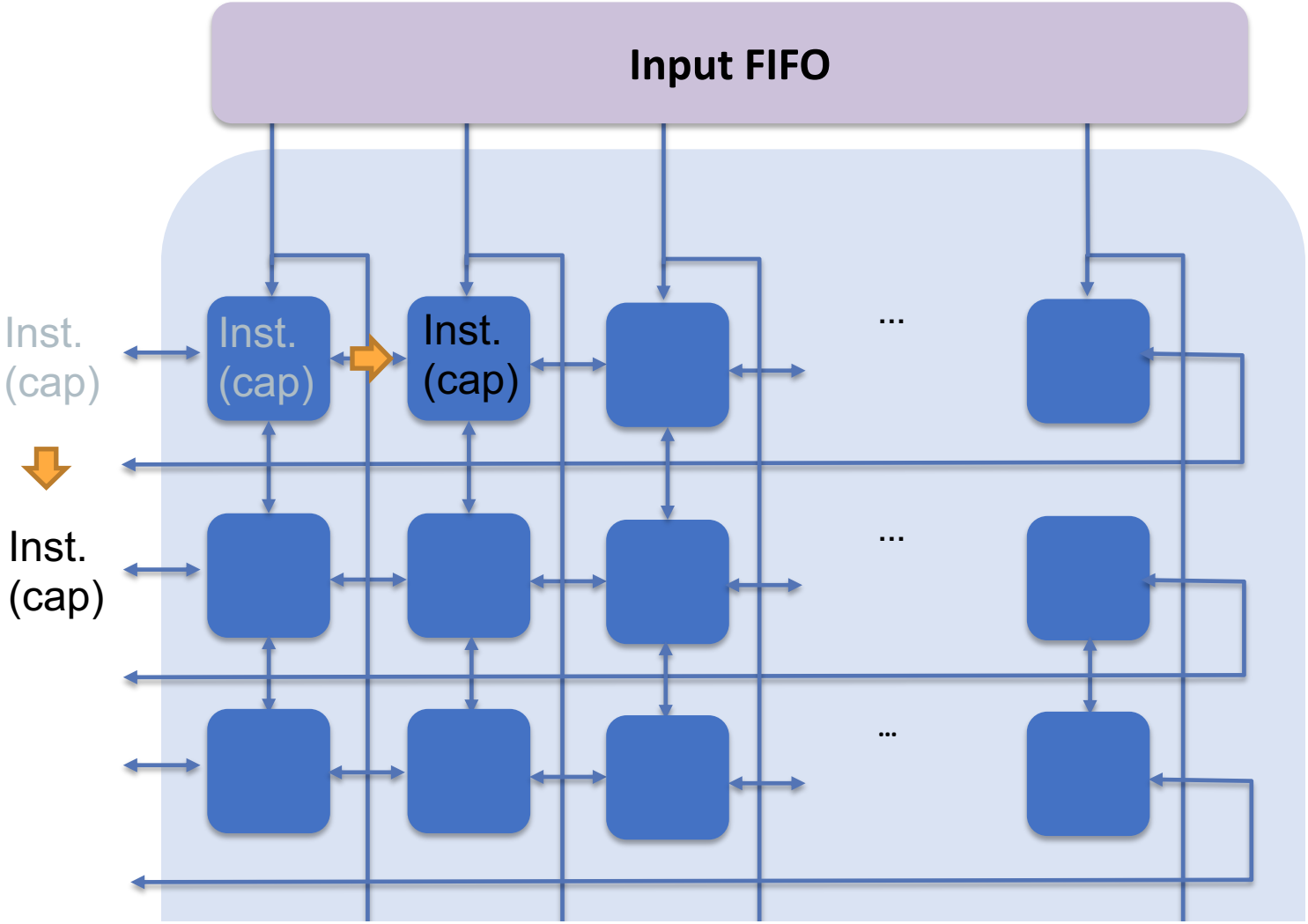


Instruction Flow



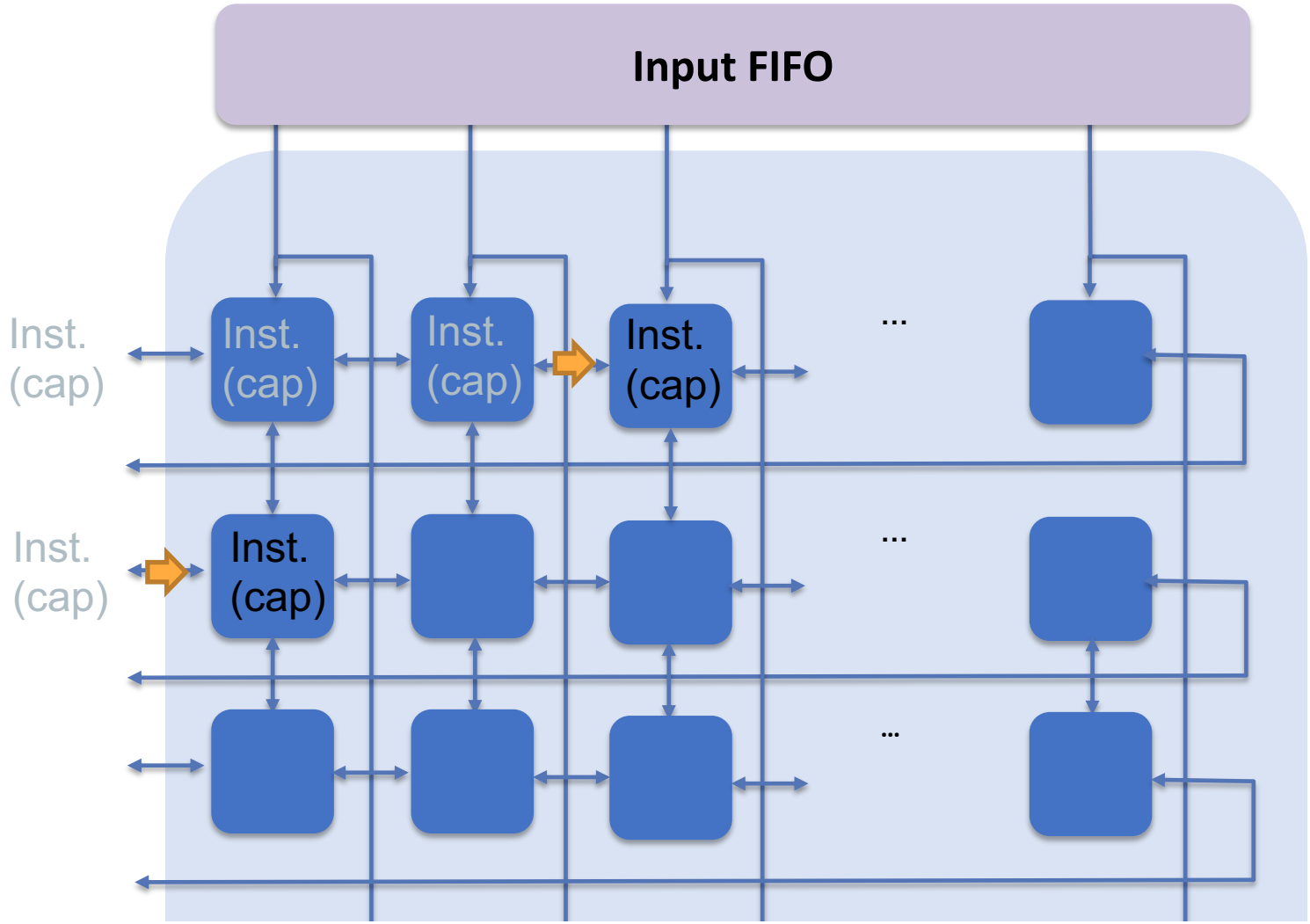
cyc 1

Instruction Flow



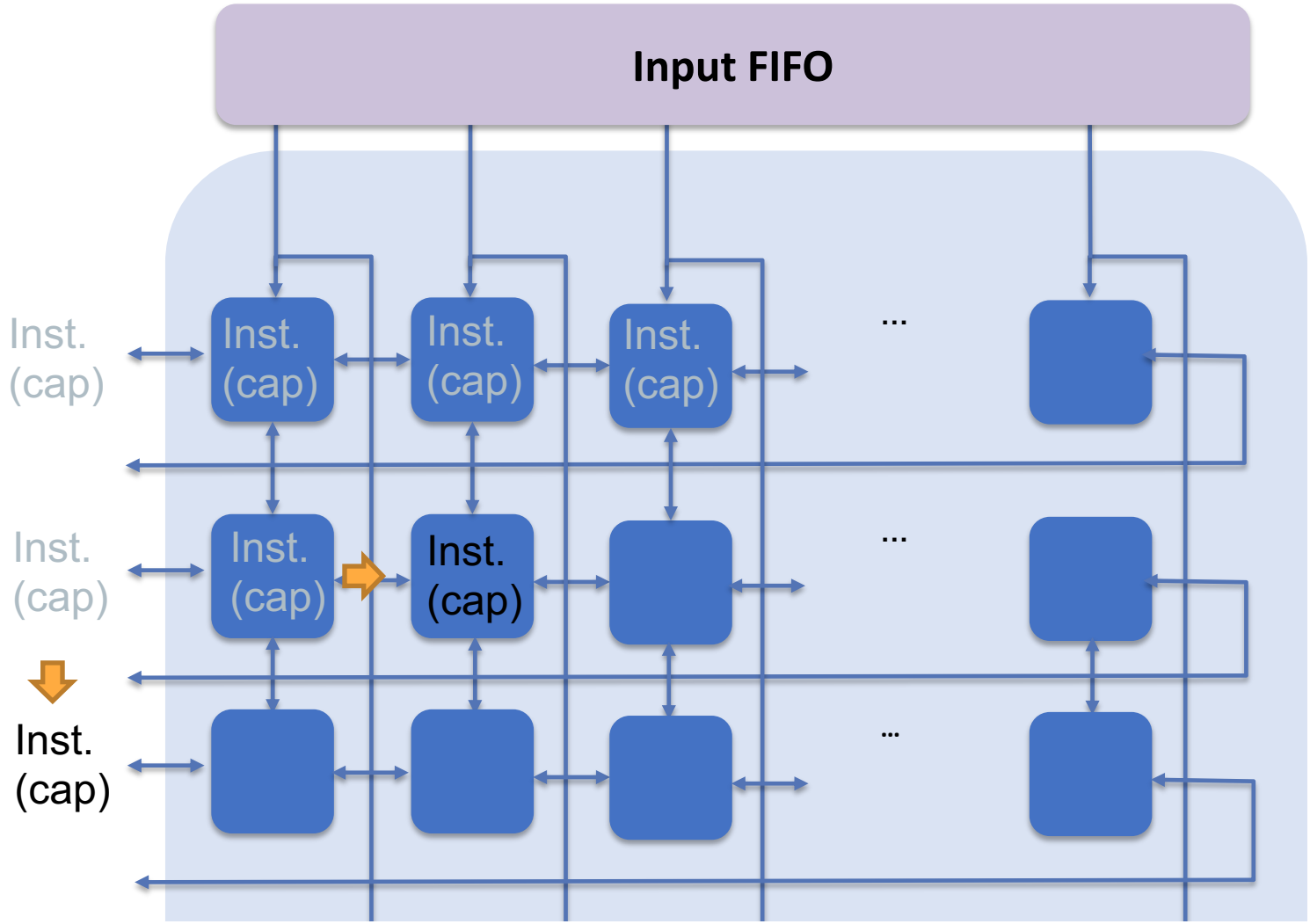
cyc 2

Instruction Flow



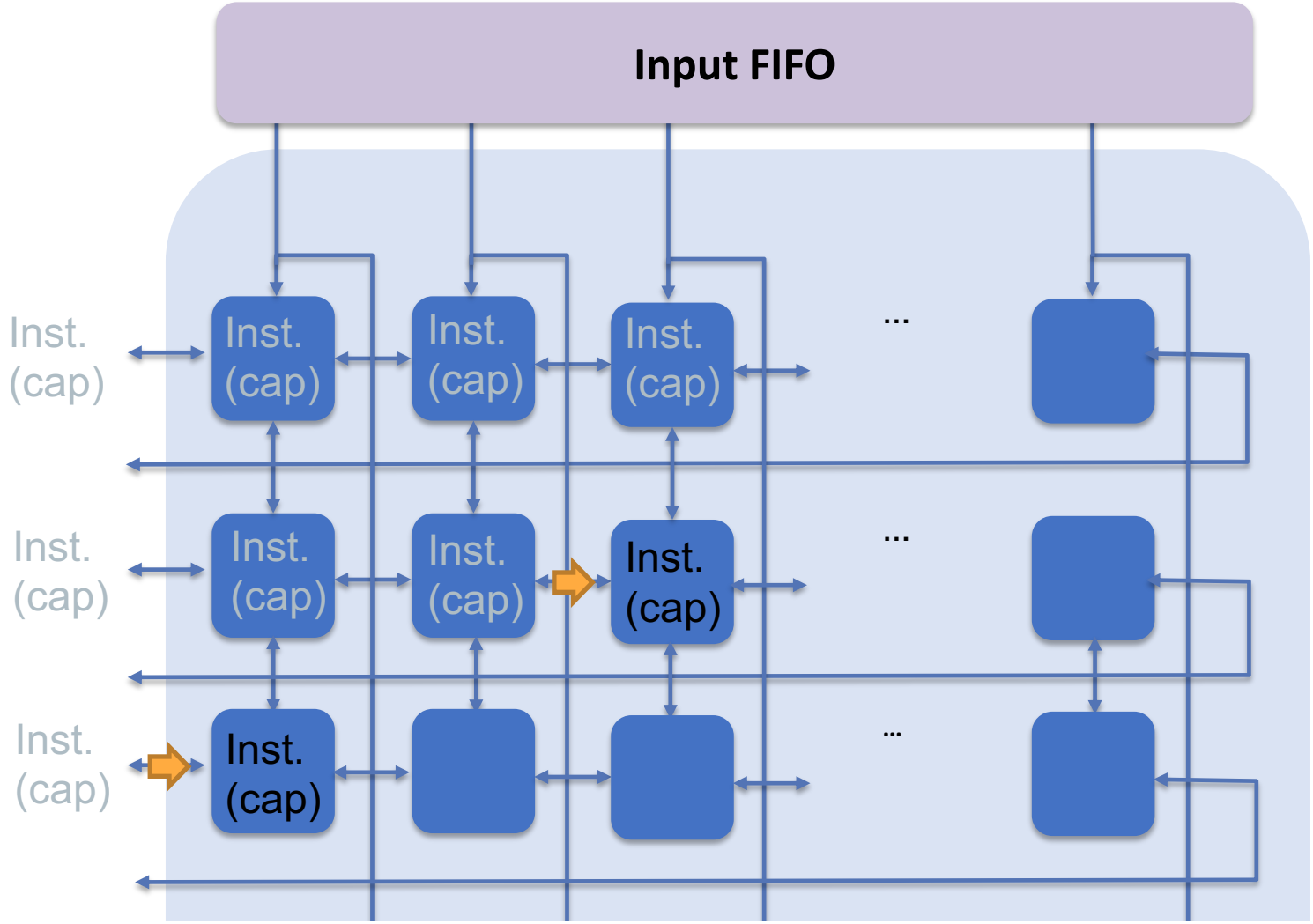
cyc 3

Instruction Flow



cyc 4

Instruction Flow



cyc 5

L0 Loading and Execution

- L0 is also a FIFO similar to Input FIFO (last row has max depth)
- Data is received at a time, but first row outputs first
- L0 loading and execution can be processed simultaneously
- Execution instruction also flows similar to kernel loading

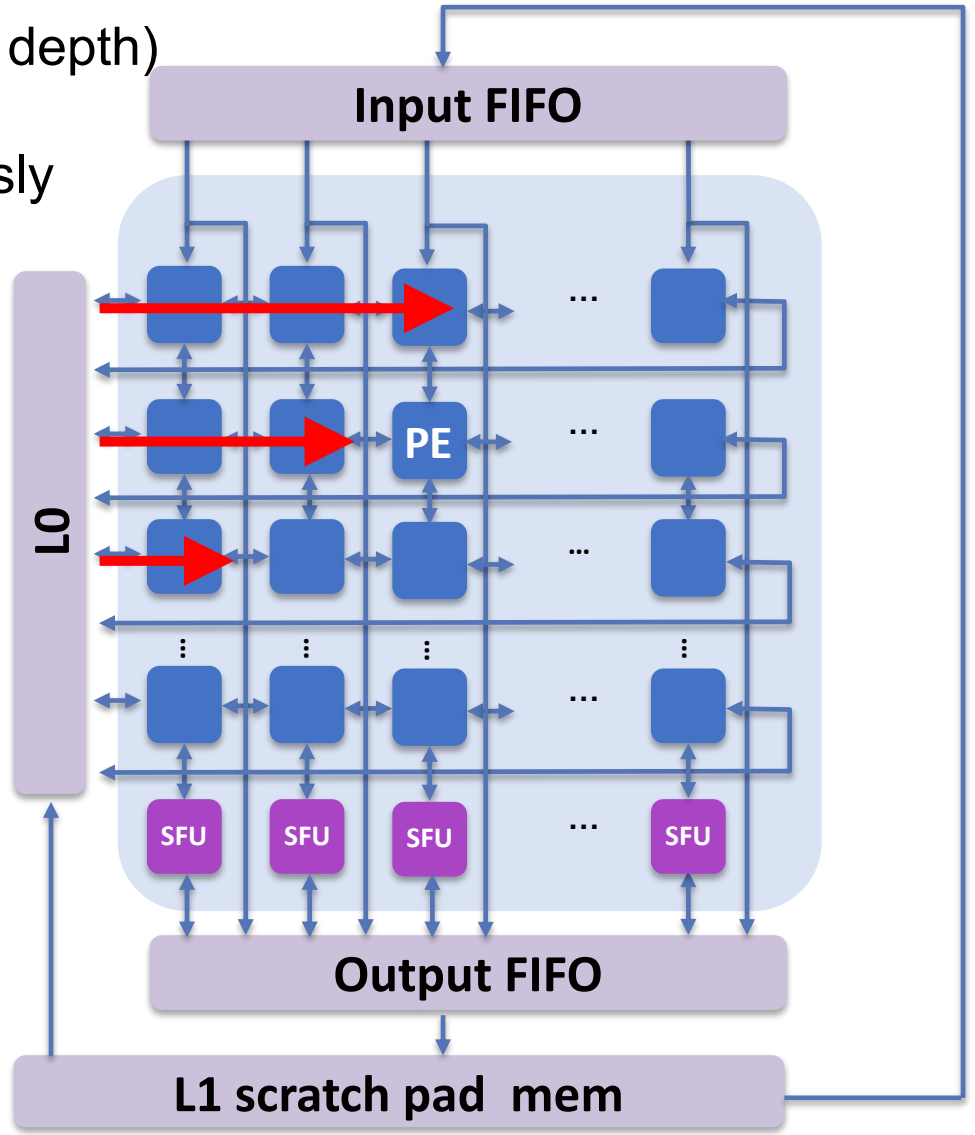
1st cyc $x_{ic0,nij0}, x_{ic1,nij0}, x_{ic2,nij0}, \dots x_{ic15,nij0}$

2nd cyc $x_{ic0,nij1}, x_{ic1,nij1}, x_{ic2,nij1}, \dots x_{ic15,nij1}$

…

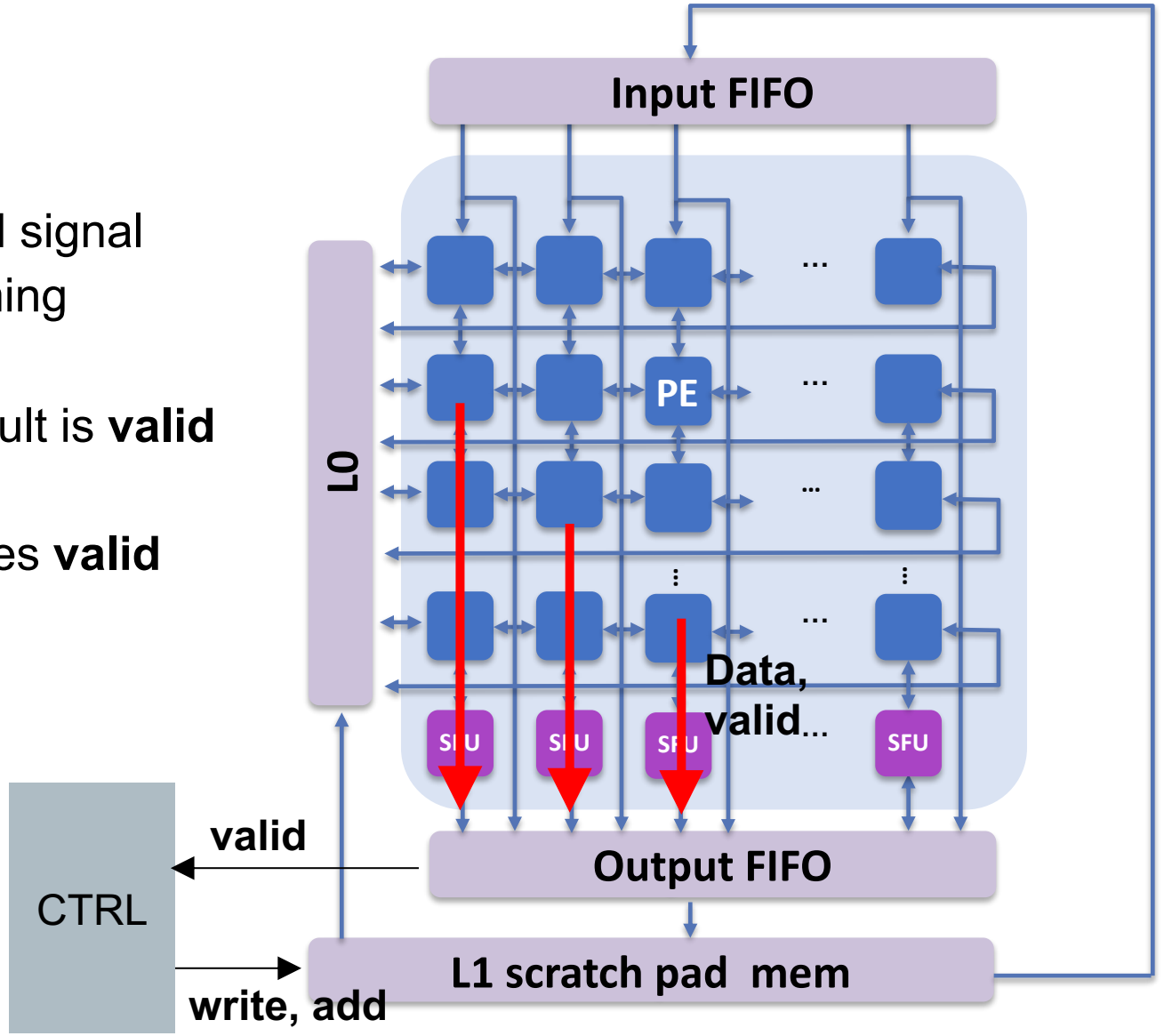
14th cyc $x_{ic0,nij14}, x_{ic1,nij14}, x_{ic2,nij14}, \dots x_{ic15,nij14}$

15th cyc $x_{ic0,nij15}, x_{ic1,nij15}, x_{ic2,nij15}, \dots x_{ic15,nij15}$



Output FIFO to L1 Scratch pad

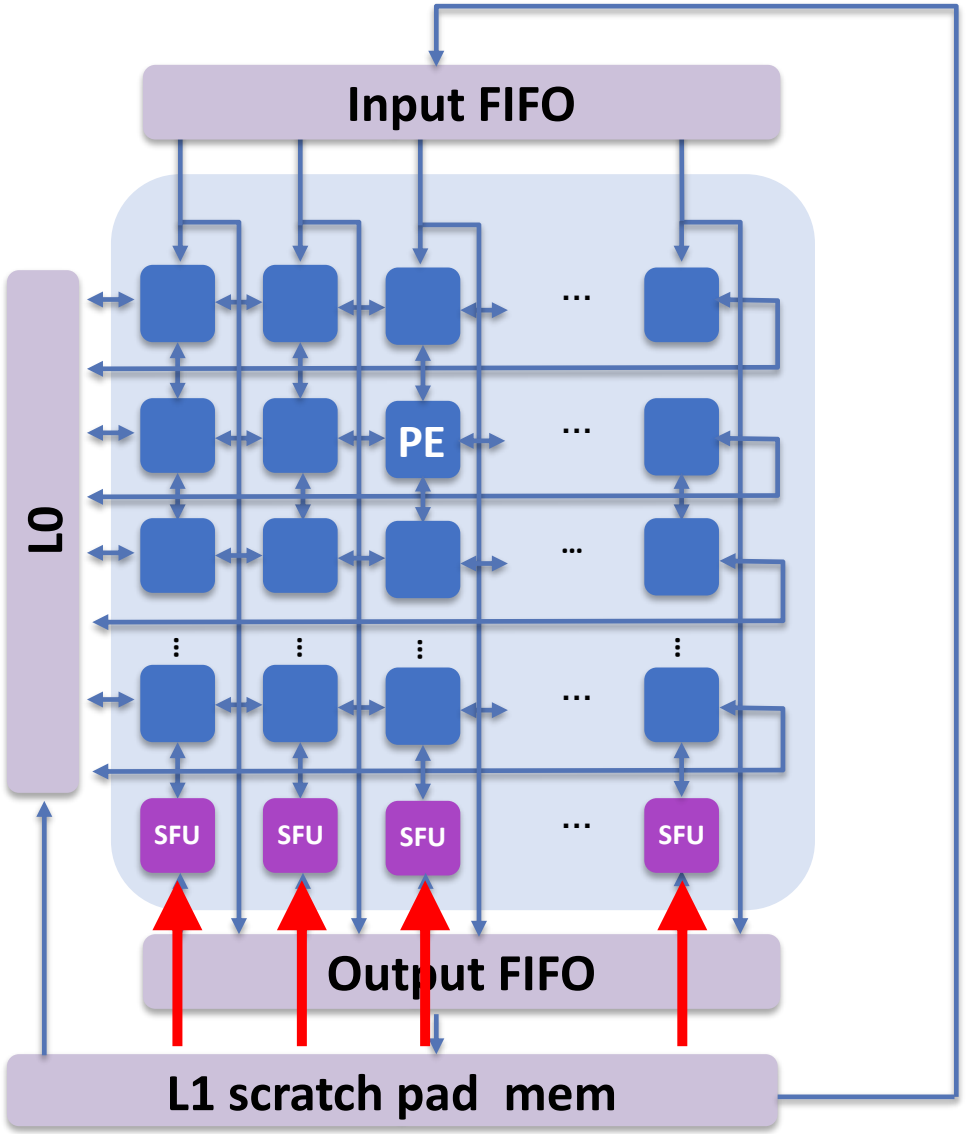
- Data from each column sends the data and valid signal
- Output fifo (OFIFO) receives data at different timing from each column
- But, outputs at a time once the last column's result is **valid**
- Thus, OFIFO's first column requires more depth
- Once OFIFO has a complete row of data, it issues **valid** signal to CTRL, and write into L1 scratch pad



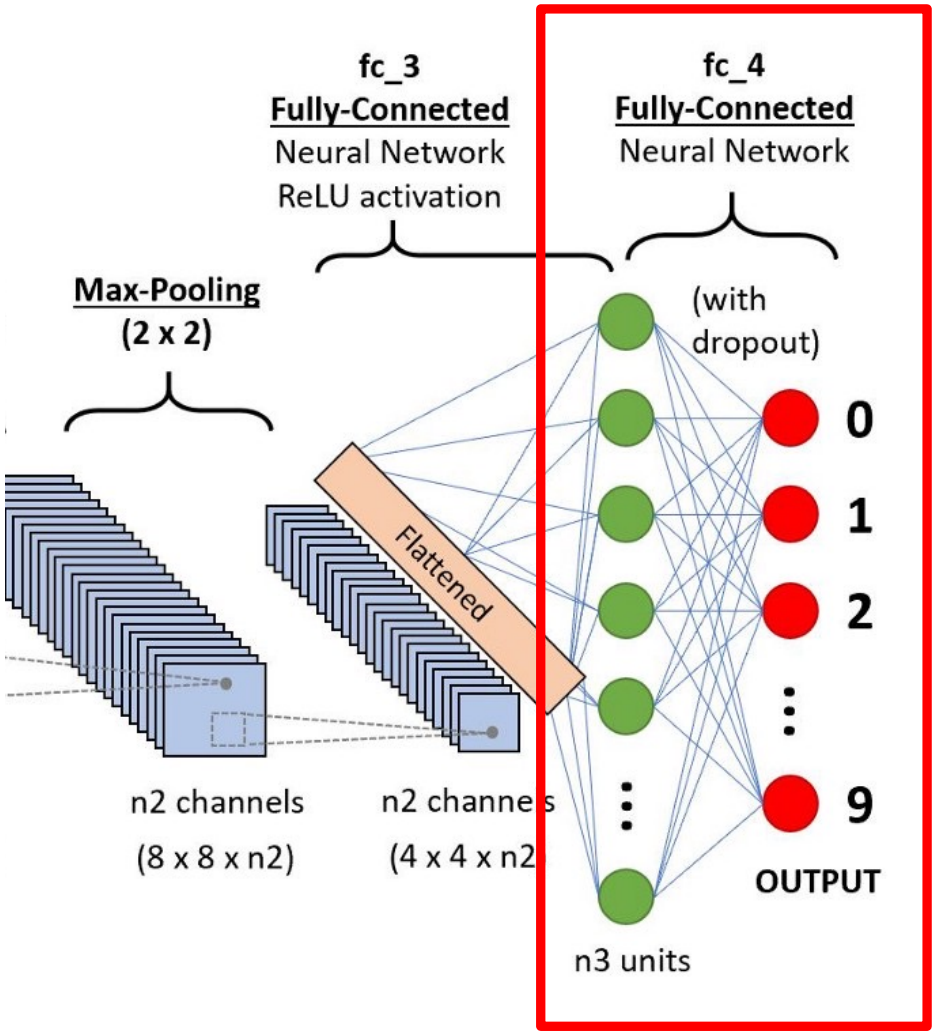
Accumulation (or Activation, e.g., ReLU)

- 1D special function unit (SFU) receives the data from L1 mem.
- Receives 9 consecutive vectors (for 3 X 3 kernels) and accumulate in SFP.
- Then, send back to L1 scratch pad mem.

- Similarly, activation functions (e.g., ReLU) can be processed right after the accumulation if needed



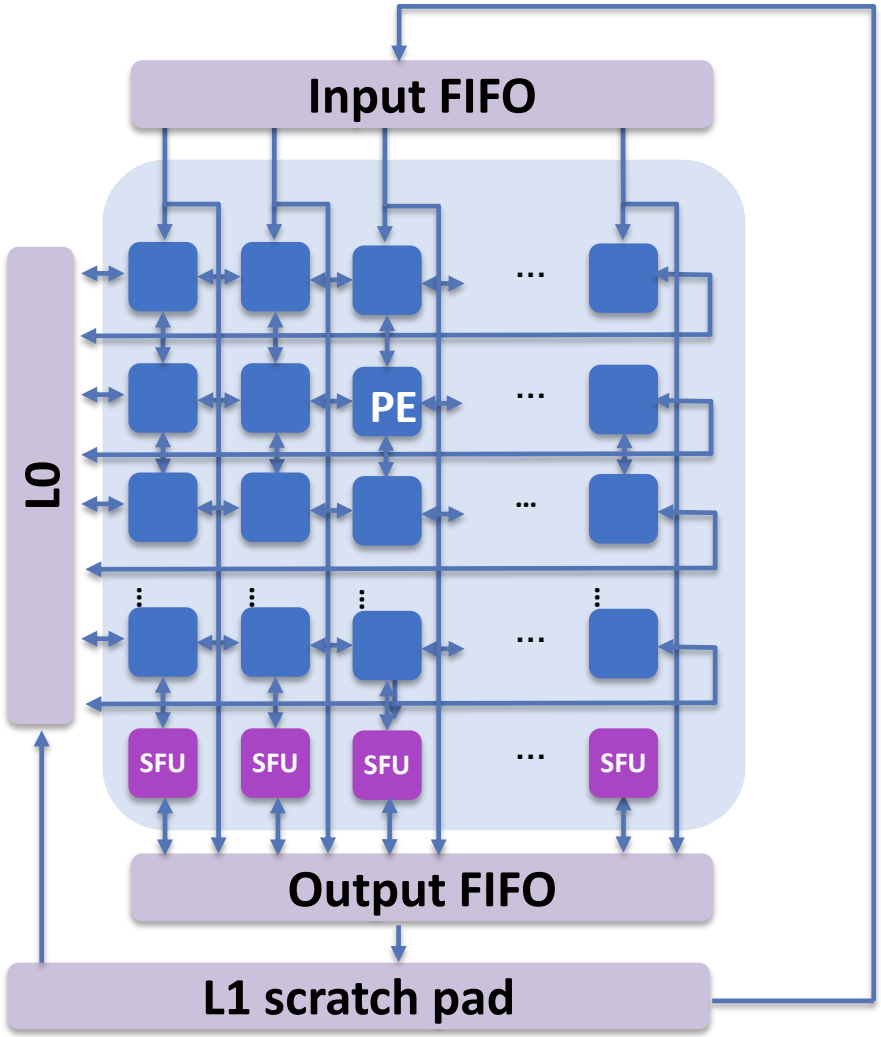
Fully-Connected Layer Computation

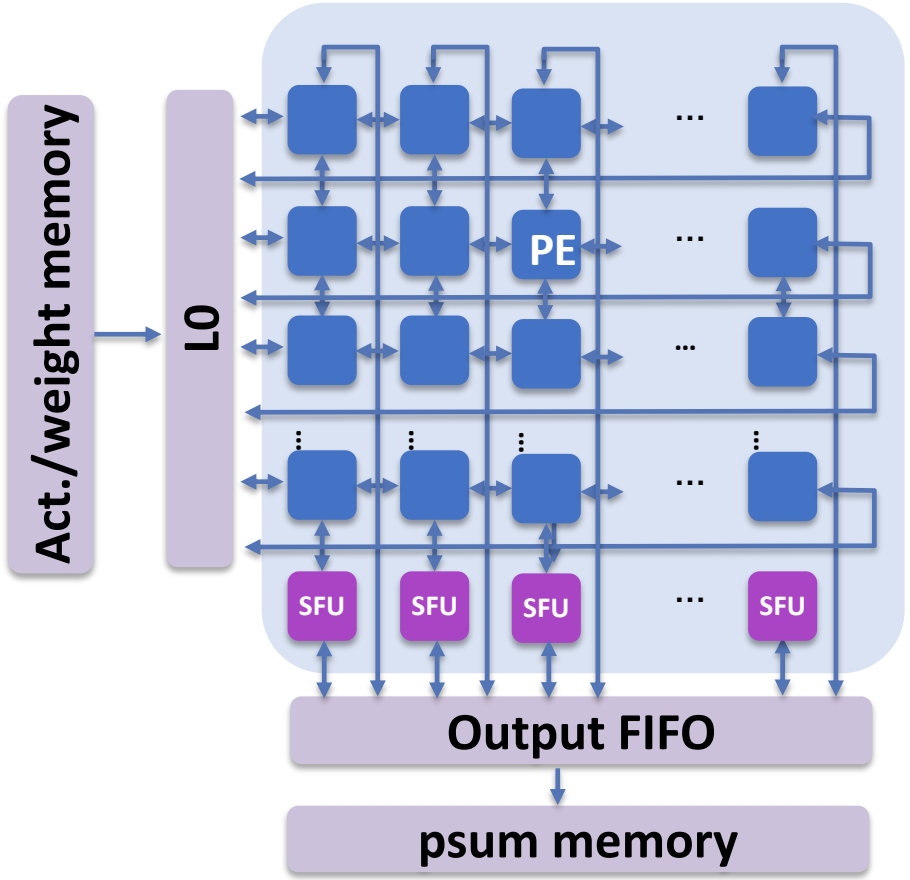


$$[x_{ic0} \quad x_{ic1} \quad \cdots \quad x_{ic255}] * \begin{bmatrix} w_{ic0,oc0} & w_{0,1} & \cdots & w_{0,9} \\ w_{ic1,oc0} & w_{1,1} & \cdots & w_{1,9} \\ \cdots & \cdots & \cdots & \cdots \\ w_{ic255,oc0} & w_{255,1} & \cdots & w_{255,9} \end{bmatrix}$$

- Vector × matrix computation (instead of mat × mat)
- Weight is arranged in the array as above matrix
- No notion of nij, thus no data reuse
- Can be reused when batch-size > 1
- The kernel weight is used just once and discarded

Activation and PSUM Memory





L1 Scratch Pad Memory Capacity

- Assume that a layer should be computed without DRAM communication in the middle
- Weight / Activation: 4-bit, psum / output: 16-bit
- e.g., VGGNet 4-th layer
  nij = 32 X 32 = 1024, ic = 64, oc = 64, kij = 9

- Padding not considered for simplicity
- Weight: 64 (ic) * 64 (oc) * 9 (kij) * 4bits = 18 KB
- Activation: 64 (ic) * 1024 (nij) * 4bits = 32 KB
- psum: 64 (oc) * 1024 * 9 (kij) * 16bits = 1152 KB => psum: 64 (oc) * 1024 * 1 * 16bits = 128 KB
- fout: 64 (oc) * 1024 (nij) * 16bits = 128 KB

Execute & acc parallel processing case

- Roughly, < 1.2MB required assuming some data can be overwritten (e.g., psum, activation)