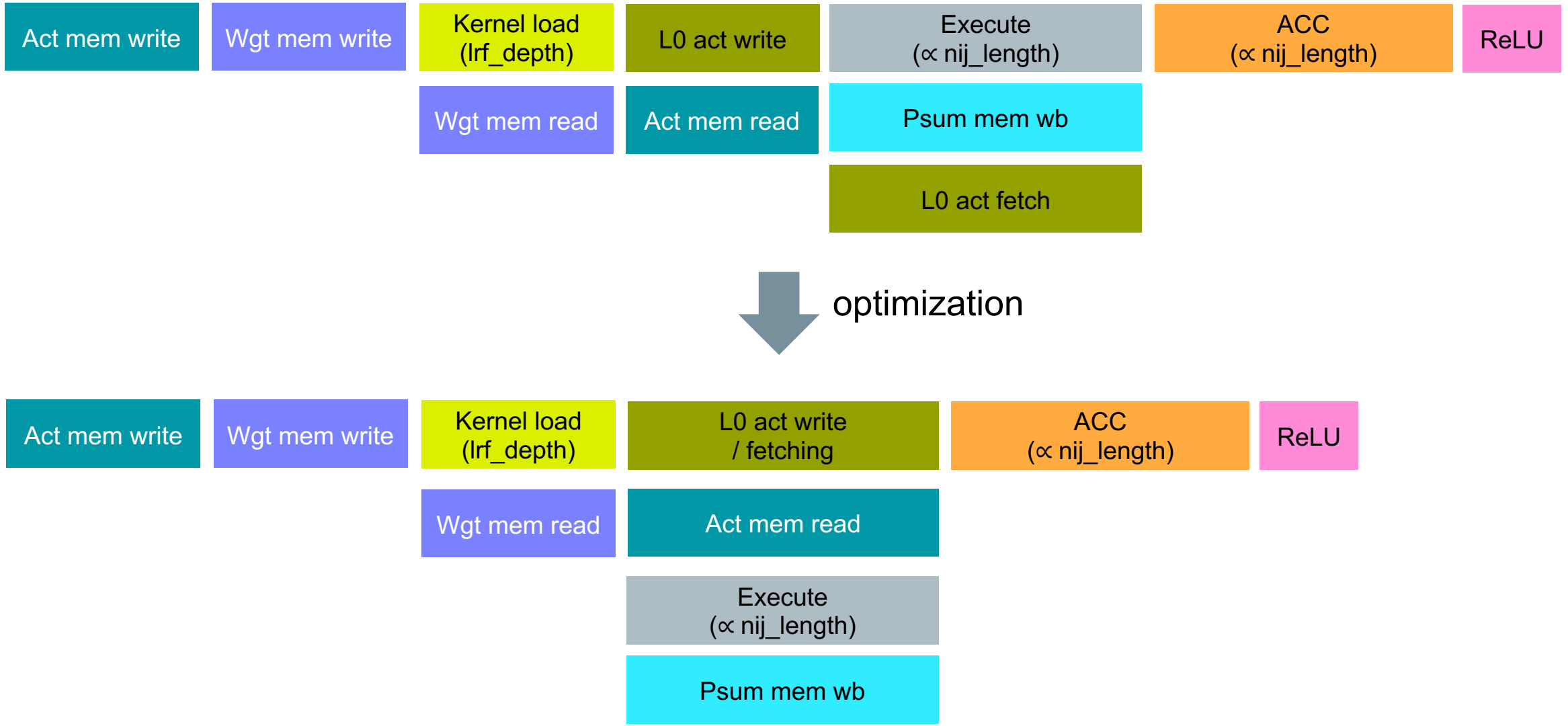
ECE284 Fall 21 W5S1

Low-power VLSI Implementation for Machine Learning

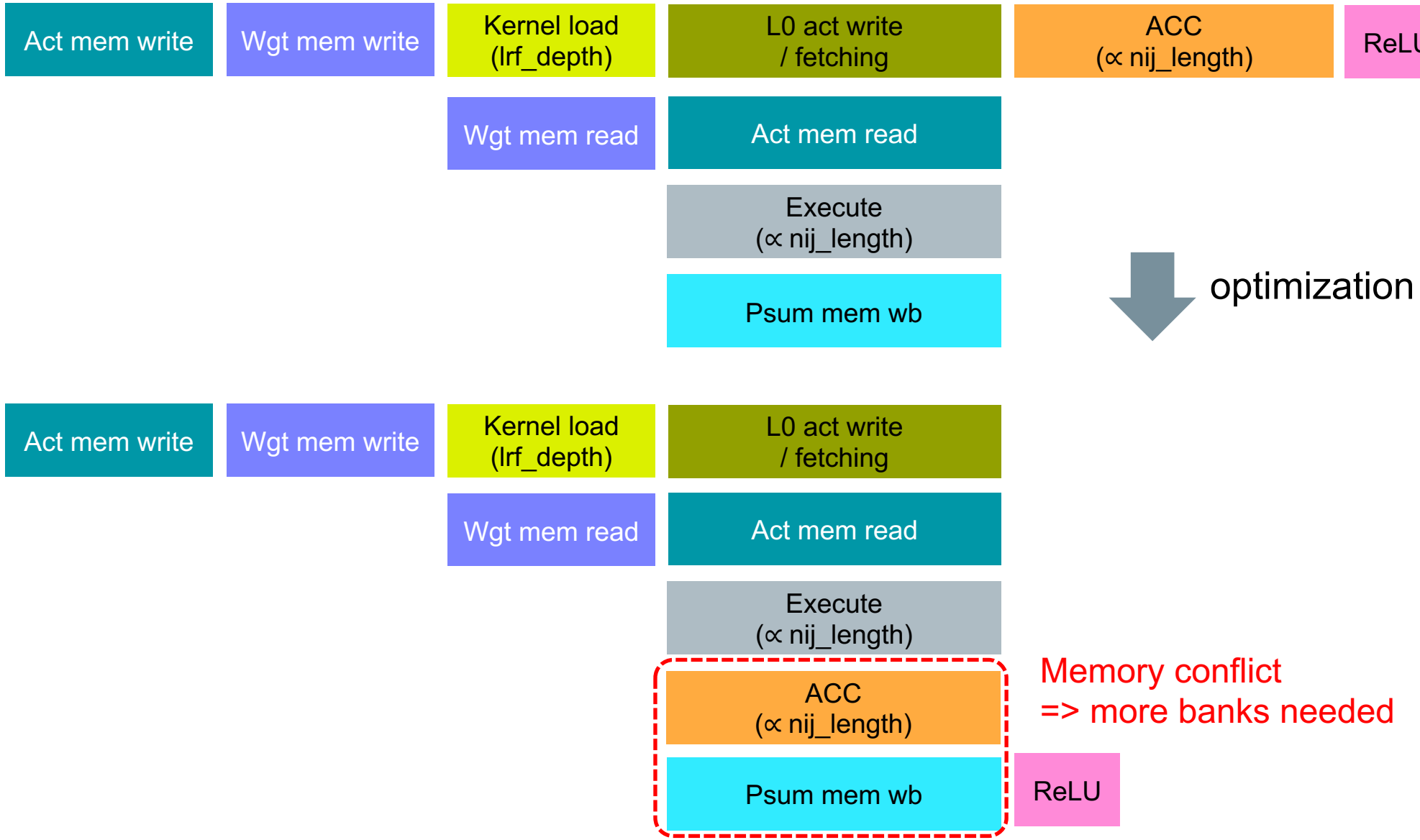
Prof. Mingu Kang

UCSD Computer Engineering

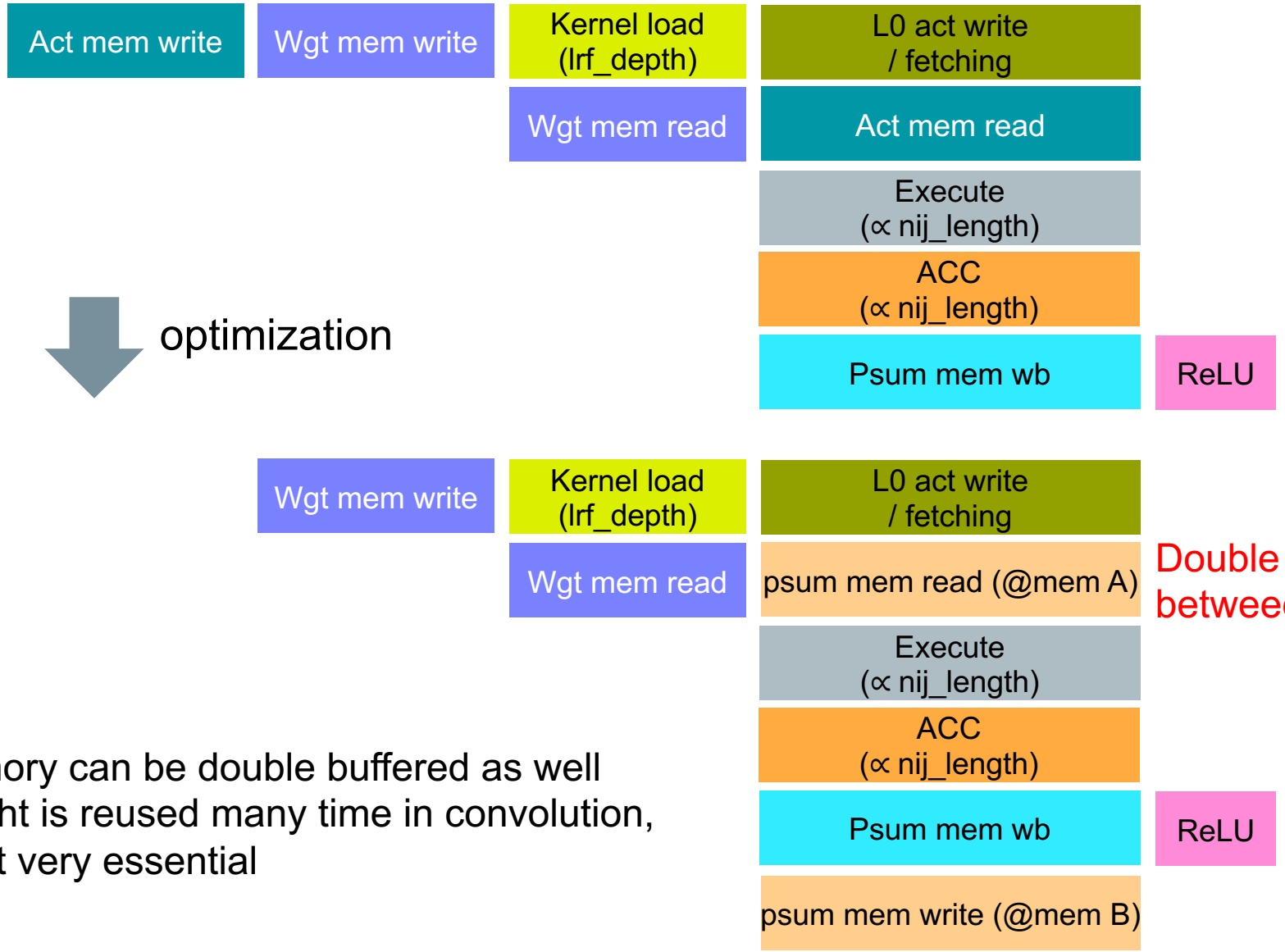
2D Systolic Array Processing Cycles (Convolution)



2D Systolic Array Processing Cycles (Convolution)



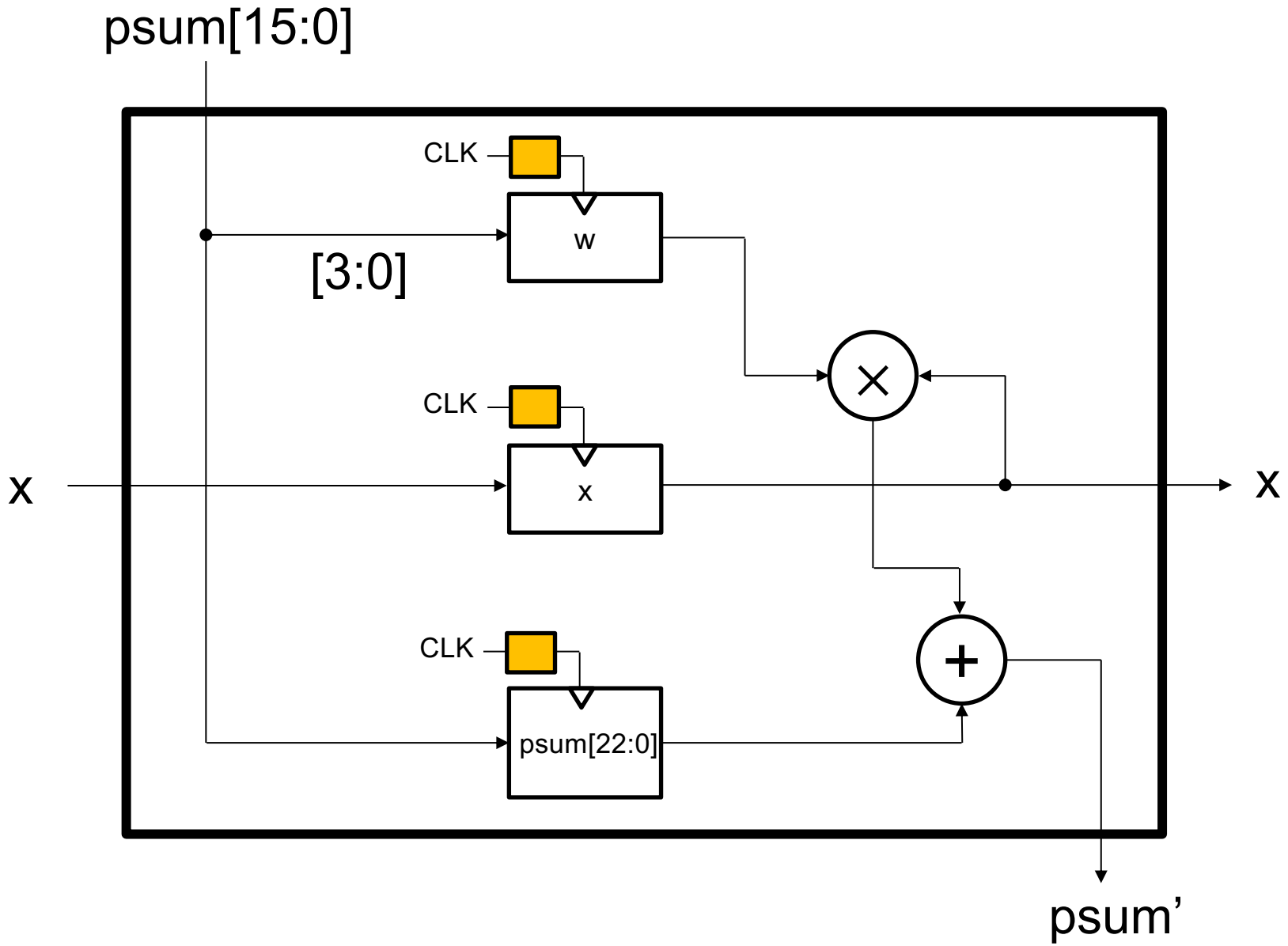
2D Systolic Array Processing Cycles (Convolution)



Double buffering betweeen mem A and B

- Wgt memory can be double buffered as well
- But, weight is reused many time in convolution, so it is not very essential

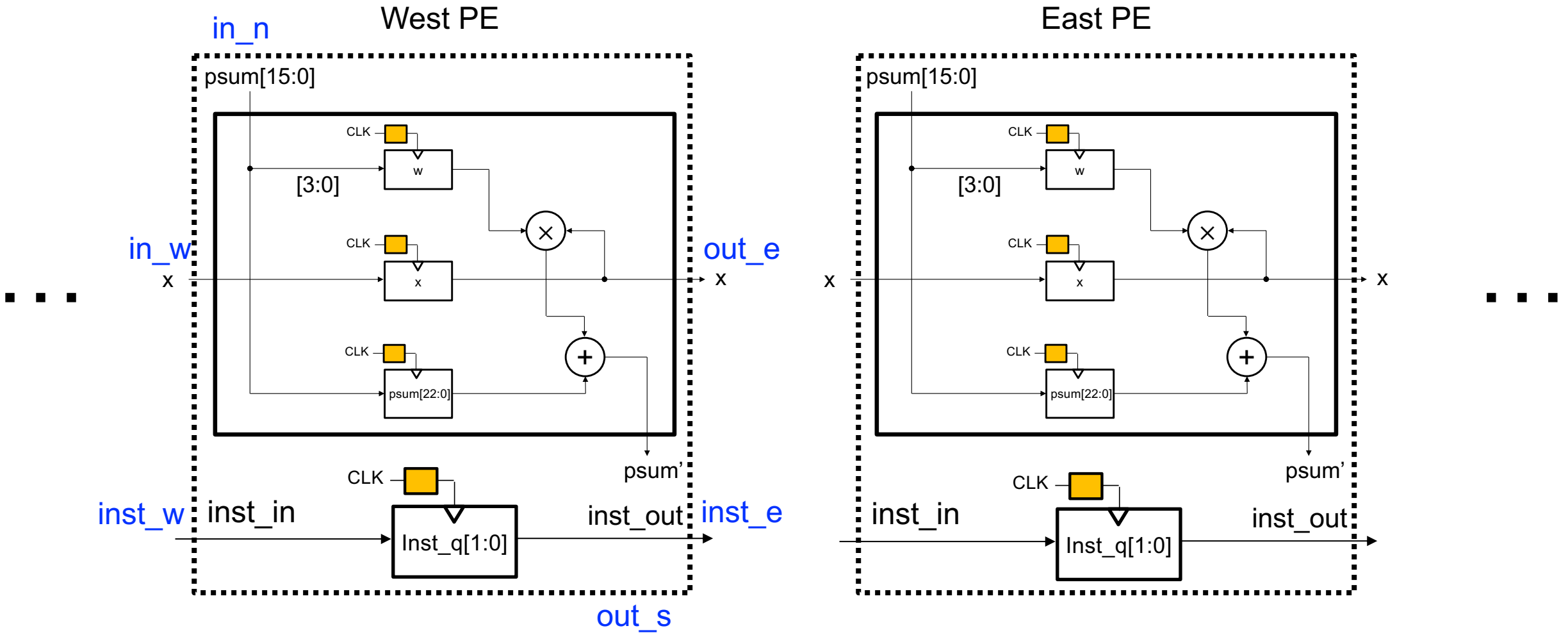
[HW_prob1] PE Tile Design



- x port and rail are reused for kernel loading

- yellow box means conditional logic

[HW_prob1] PE Tile Design (add instruction flow)



e.g., inst[0]: kernel loading, inst[1]: execution

[HW_prob1] mac_tile Design (no need to simulate)

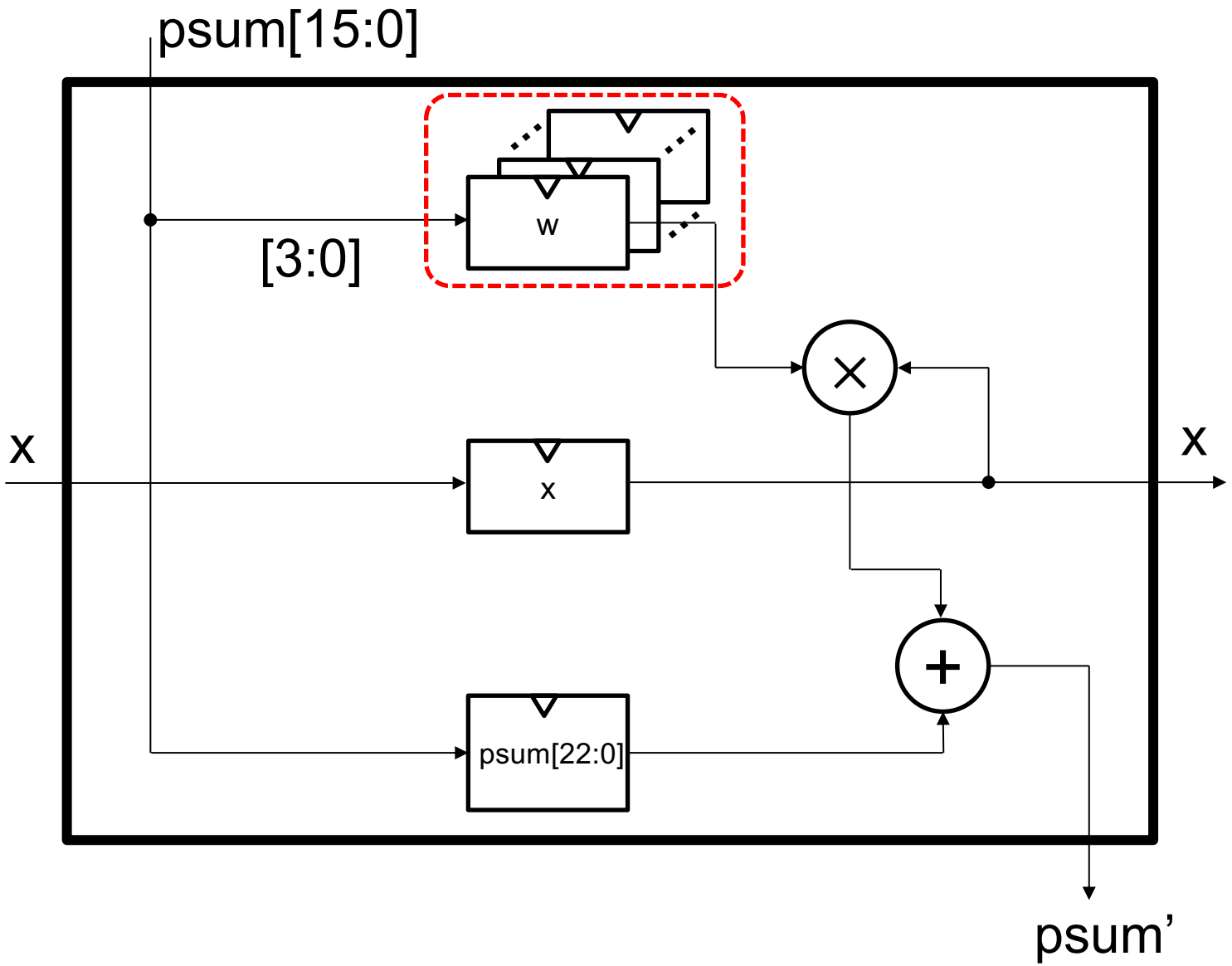
- Download from github: hw/w5/hw1_2
- Equip ports: in_w, out_e, in_n, out_s, inst_w (input), inst_e (output), clk, reset (synchronous)
- Add latches: inst_q[1:0], a_q (activation), b_q (weight), c_q (psum), load_ready_q
- a_q is connected to out_e, and inst_e is connected to inst_q
- When reset ==1, inst_q[1:0] becomes all 0, and load_ready_q becomes 1
- Accept your inst_w[1] (execution) always into inst_q[1] latch.
- When either inst_w[0] or inst_w[1], accept the new in_w into a_q latch.
- When inst_w[0] (kernel load) == 1 and also load_ready_q ==1, accept the new weight in the b_q latch via in_w port. At the same time, load_ready_q becomes 0.
- When load_ready_q ==0, latch inst_w[0] into inst_q[0], which is connected to inst_e

[HW_prob2] mac_row & mac_array (no need to simulate)

- Use for loop to build mac_row based on mac_tile from prob1
- PE row should provide a “valid[col-1:0]” vector to let ofifo know
- valid for the column is inst_e[1] for the column
- Look at “temp” wire use-case

- Similarly, use for loop to build mac_array based on mac_row
- PE row should provide a “valid[col-1:0]” vector to let ofifo know
- Here, only the last row’s valid signals are used.
- Inst_w[1:0] is also propagated from row0 to row7

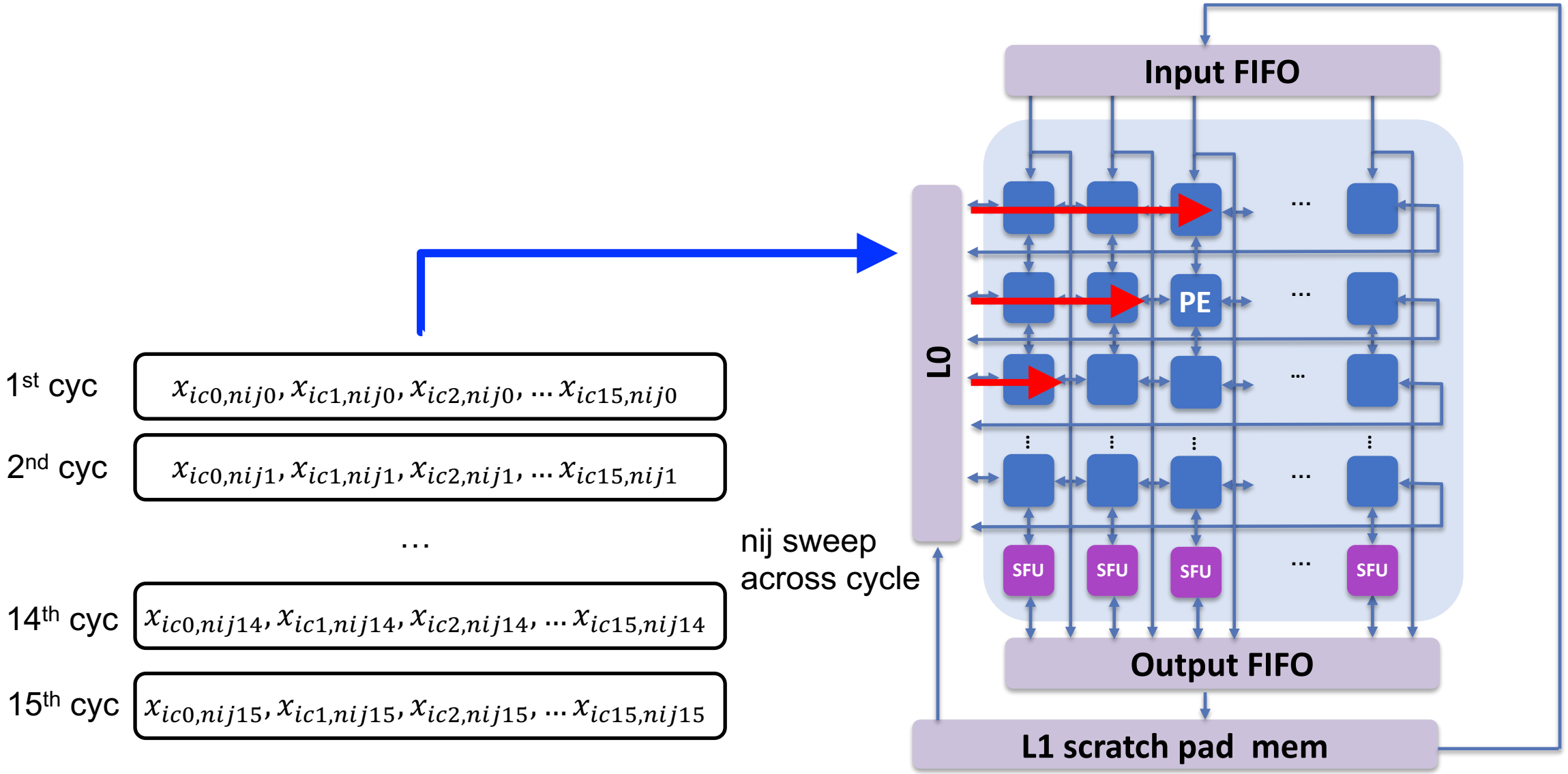
Multiple Registers in PE with PE-internal Accumulation



- Stores multiple weights

- Use by taking turns for PE-internal accumulation

Data Flow with Single Register in PE



Data Flow with 2 Registers with Internal Accumulation

- Two registers stores
  w(ic=0, oc=0, kij=0), w(ic=16, oc=0, kij=0)
- More input channels can be computed within single PE
- Even cycle uses first weight and odd cycle uses the second
- Accumulate internally once and then send psum to south
  i.e., send psum to south every other cycle

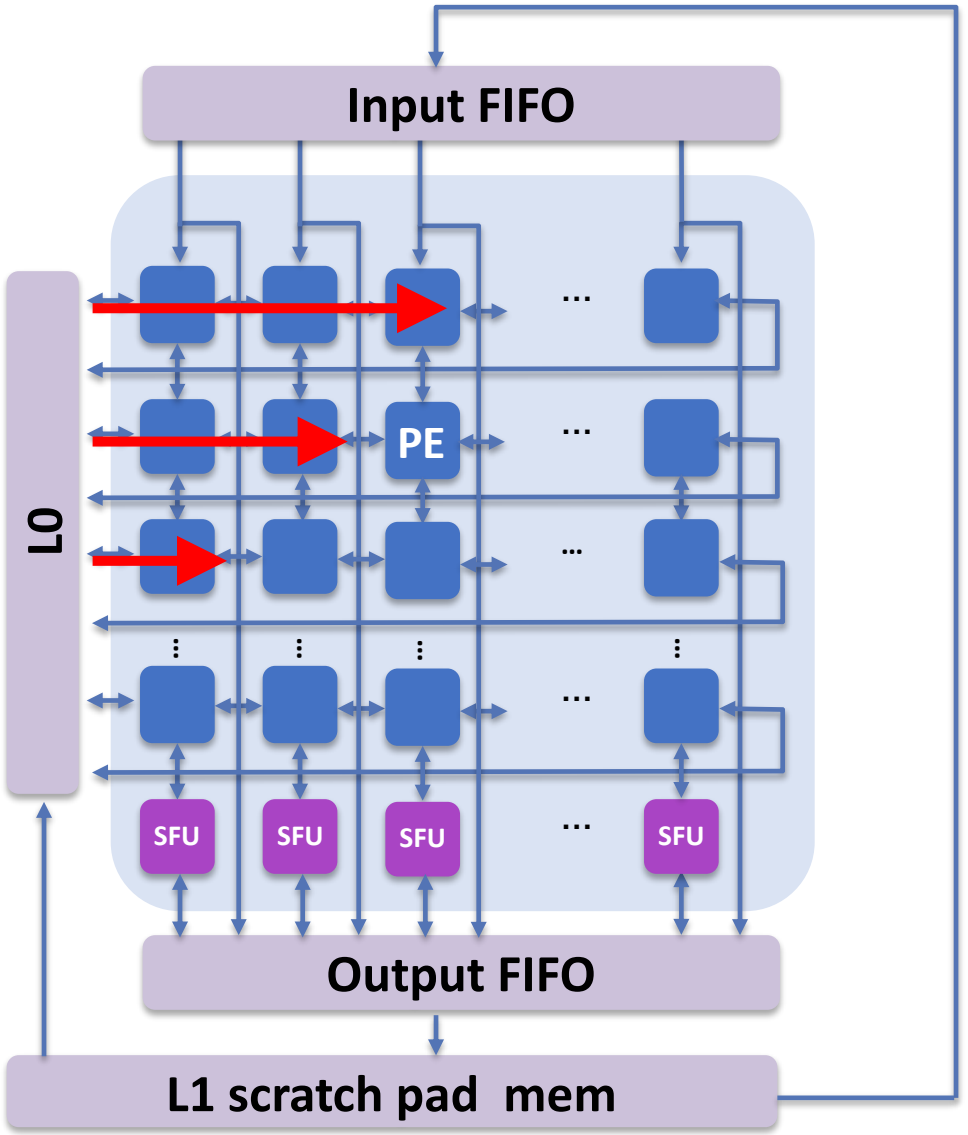
1st cyc $x_{ic0,nij0}, x_{ic1,nij0}, x_{ic2,nij0}, \cdots x_{ic15,nij0}$

2nd cyc $x_{ic16,nij0}, x_{ic17,nij0}, x_{ic18,nij0}, \cdots x_{ic31,nij0}$

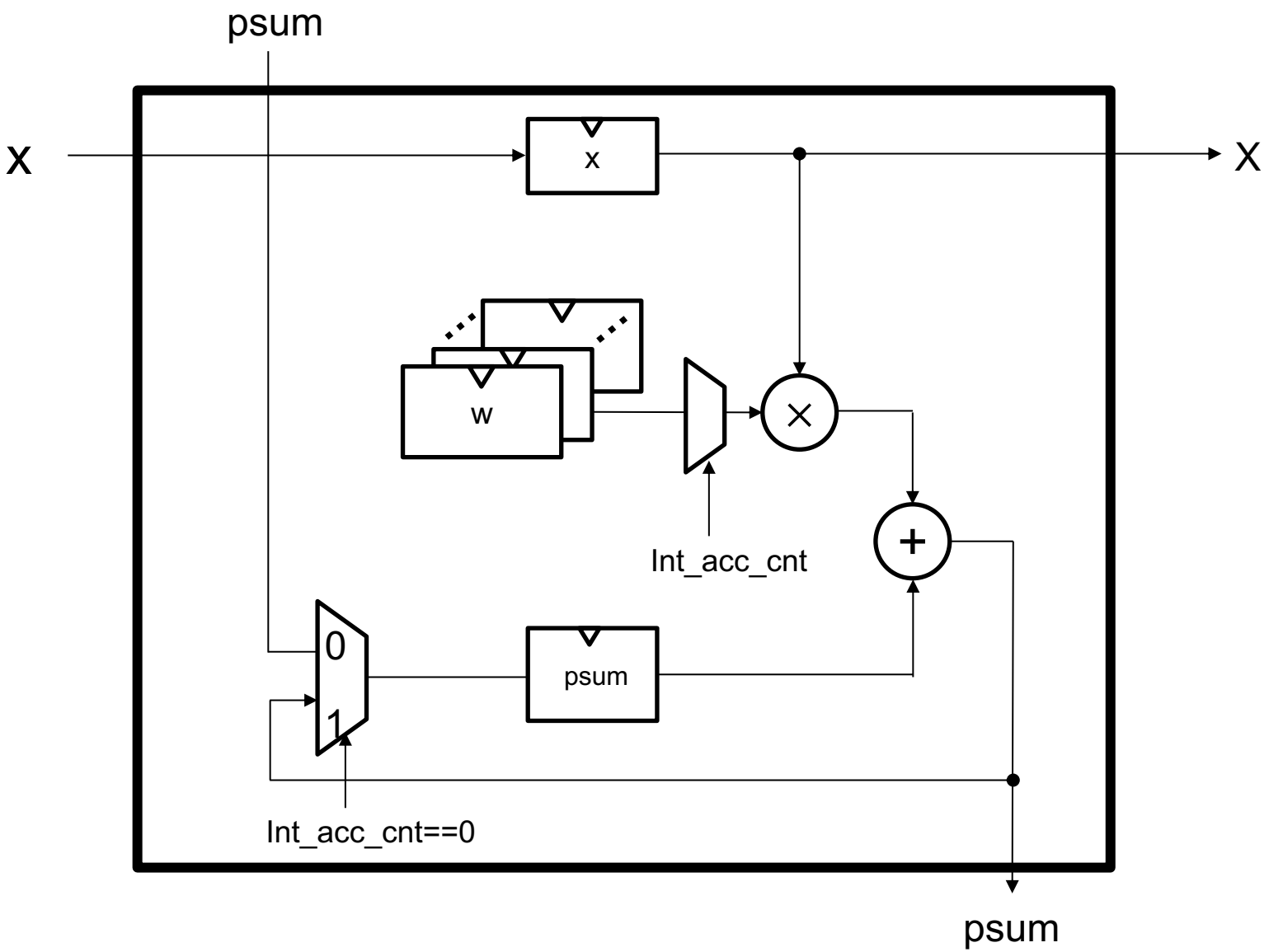
3rd cyc $x_{ic0,nij1}, x_{ic1,nij1}, x_{ic2,nij1}, \cdots x_{ic15,nij1}$

4th cyc $x_{ic16,nij1}, x_{ic17,nij1}, x_{ic18,nij1}, \cdots x_{ic31,nij1}$

…



Internal Accumulation with Multiple Registers



- Assume we have N reg.
- psum sent to south when int_acc_cnt = N

- Benefit:
  - behave as if we have N*16 rows of array
  - Accumulation in SFP requires psum mem read and write, but PE internal acc does not require mem access

[HW_prob3] L0 (= IFIFO) Design

- Fill out place holders in l0.v

- o_ready signal outputs to inform that there is at least a room to receive a new vector
- o_full signal is enabled if at least a column is full
- Create a two different versions:
  1. version1: read all row at a time.
     a. Then, verify your read values are correct
     b. Verify your full and empty signals are working fine
  2. version2: read 1 row at a time
     a. when rd = 1, row0 enabled, then row1, then row2, ….
     b. when rd = 0, row0 disabled, then row1, then row2, …

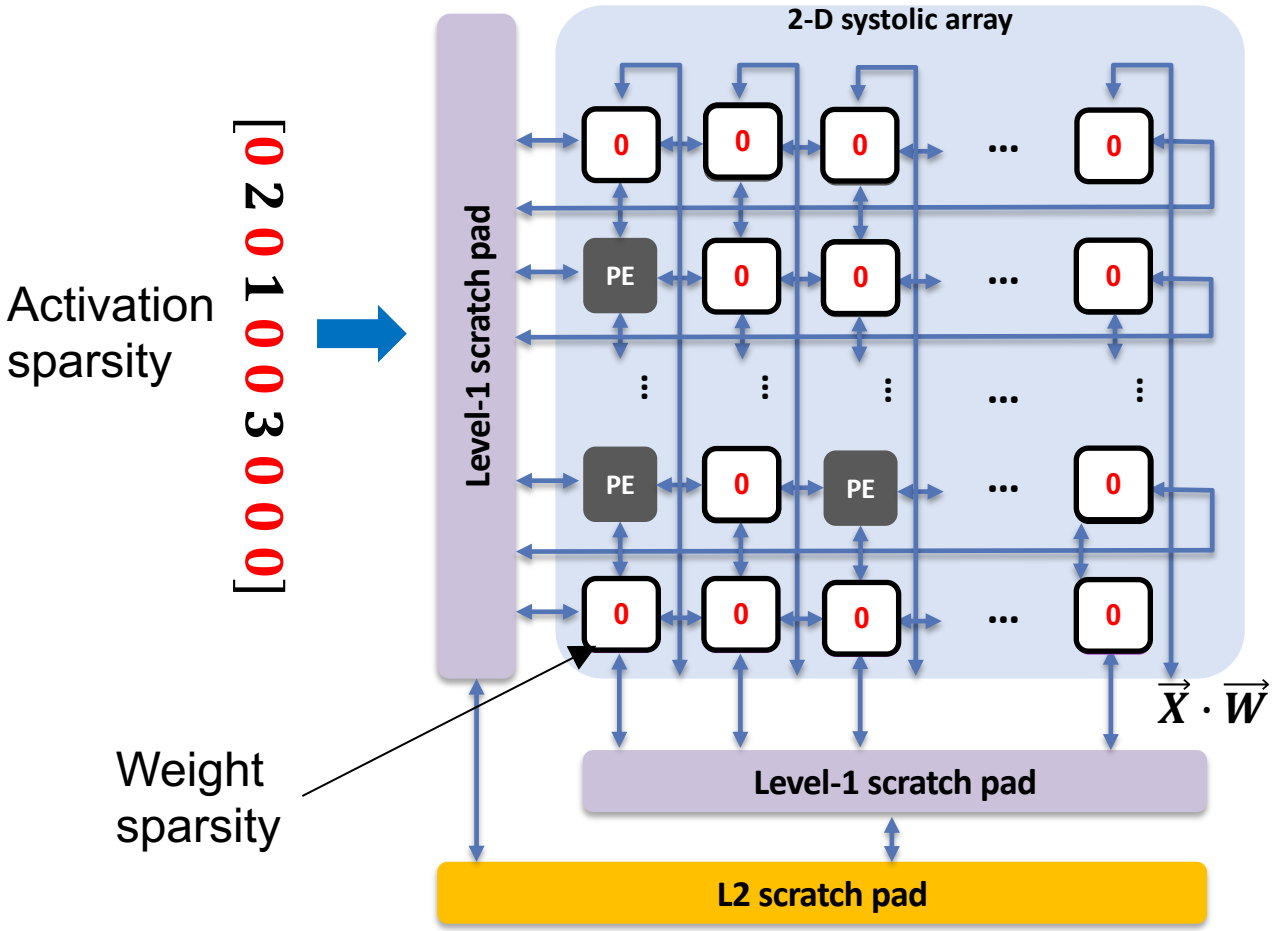
- Reference vcds are included for both versions

[HW_prob4] OFIFO Design (no need to simulate)

- OFIFO is the same as L0 in general, but requires o_valid signal
- Receives the data at each column at different timing

  e.g., wr should be vector (1bit per col), not single value
- o_valid signal outputs to inform that there is at least one full vector ready
- Read out all columns at a time
- Do not need to simulate with test bench

Sparsity

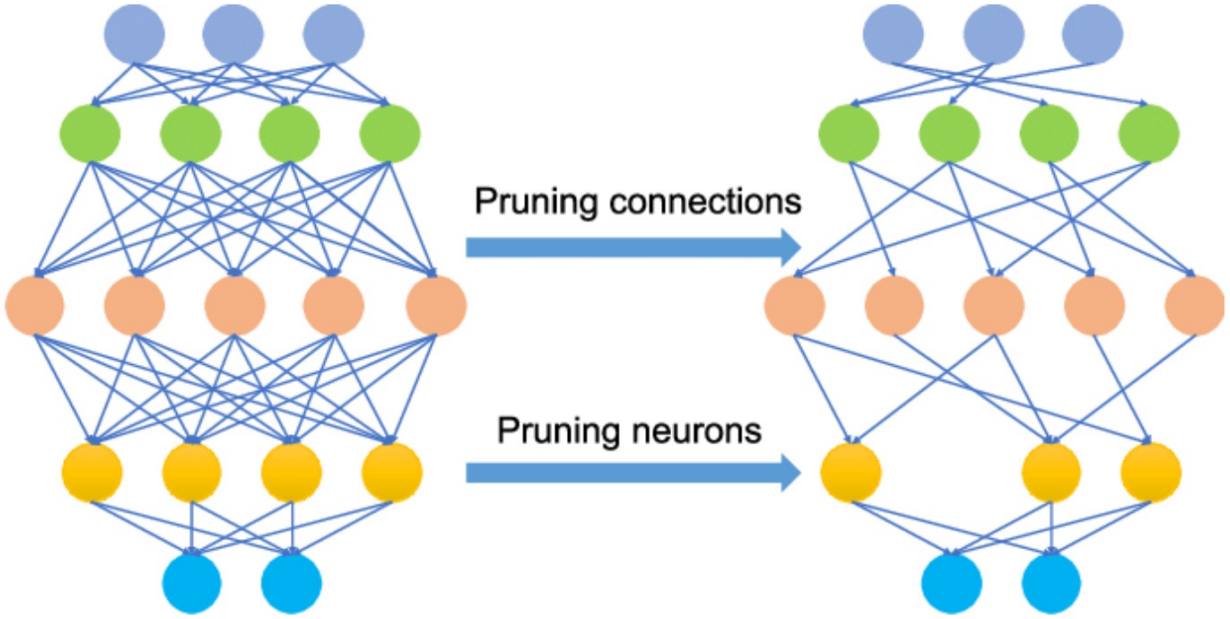


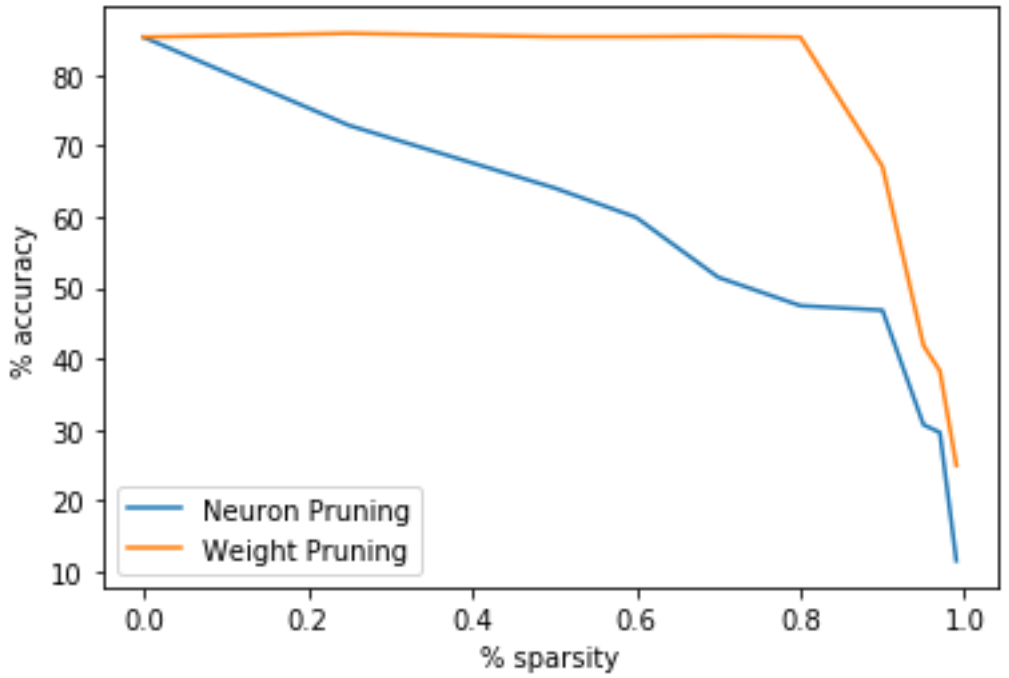
- Sparsity: many zeros in the activation and weight matrices

- High sparsity increase due to:
  1. ReLU
  2. Quantization
  3. Pruning

- Opportunity for energy savings

Pruning

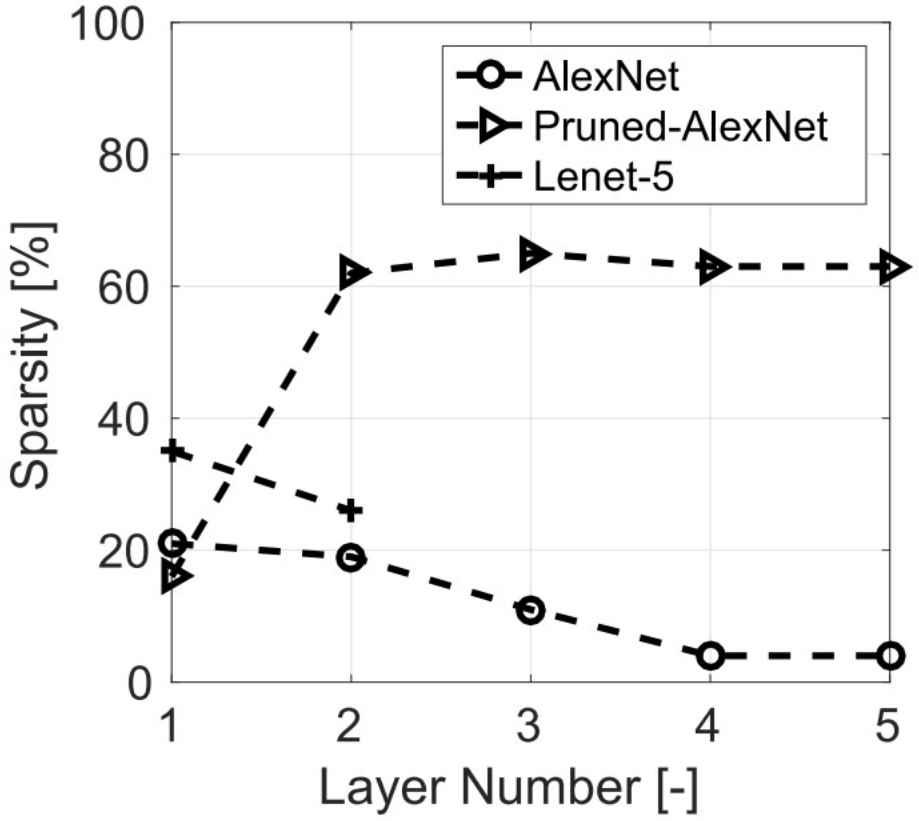




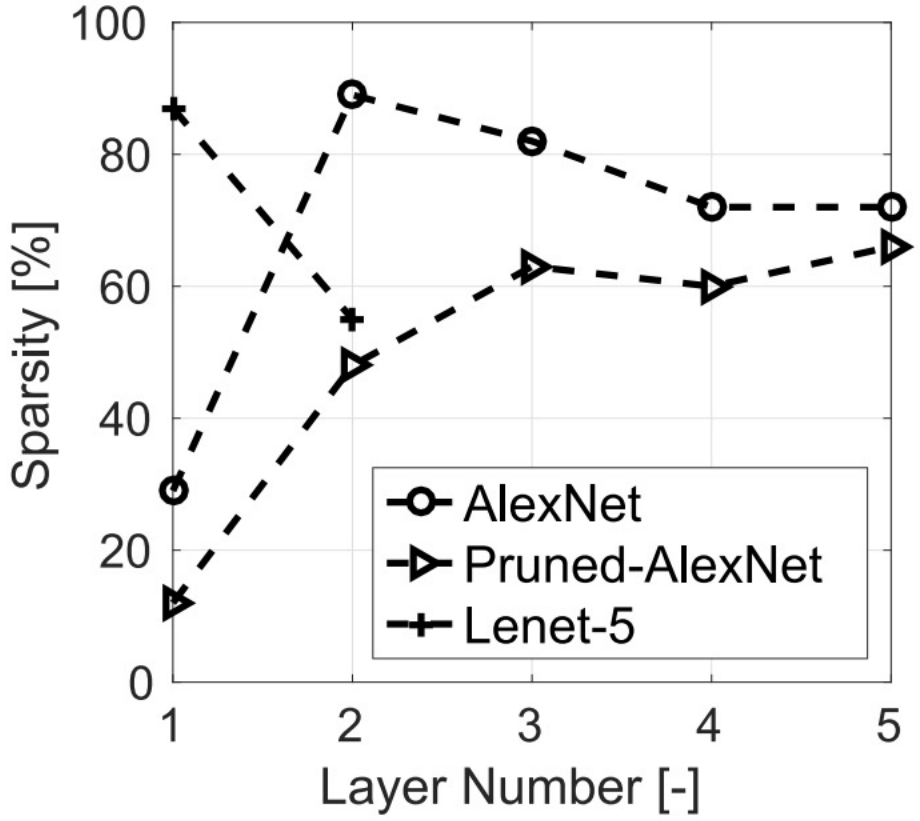
- Weight pruning:
  set to zero the smallest k% of the weights based magnitude (e.g., L1 or L2 norm).
- Activation pruning:
  remove the k% of columns of a weight matrix according to their magnitude.

https://medium.com/@yasersakkaf123/pruning-neural-networks-using-pytorch-3bf03d16a76e

Sparsity Example before / after Pruning



Weight sparsity



Activation sparsity

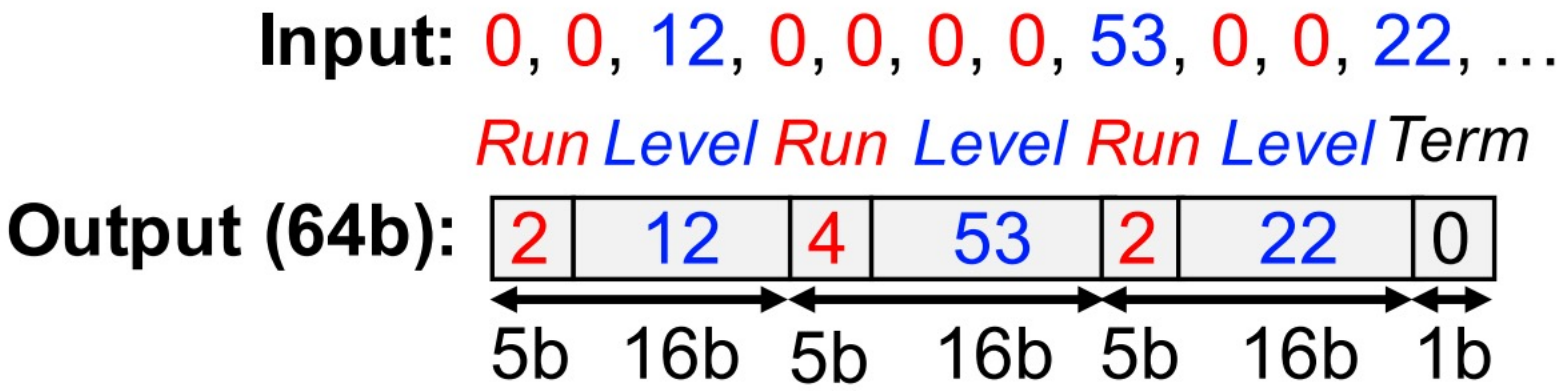
B. Moons, ”An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS” JSSC17

Unstructured vs. Structured Pruning

- Unstructured pruning
  - Any random location in the weight matrix is pruned
  - Hardware unfriendly (because the computation at any random location cannot be skipped in the 2D systolic array)
  - Better pruning rate achieved

- Structured pruning
  - Specific row or column in the weight matrix is pruned
  - Thus, input or output channel is regularly pruned out
  - More hardware-friendly (we can skip the row or column in the 2D systolic array)

Run-length compression (RLC)



- Consecutive zeros with a maximum length of 31 represented with 5-b
- Non-zero value is represented with 16-b
- Every three pairs of run and level are packed into a 64-b word (for regularity)
- The last bit indicating if the word is the last one in the code.
- If the sequence of 0 is too long, there is a filler 0 (which is regarded as non zero)

YH. Chen, ”Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks”, JSSC17