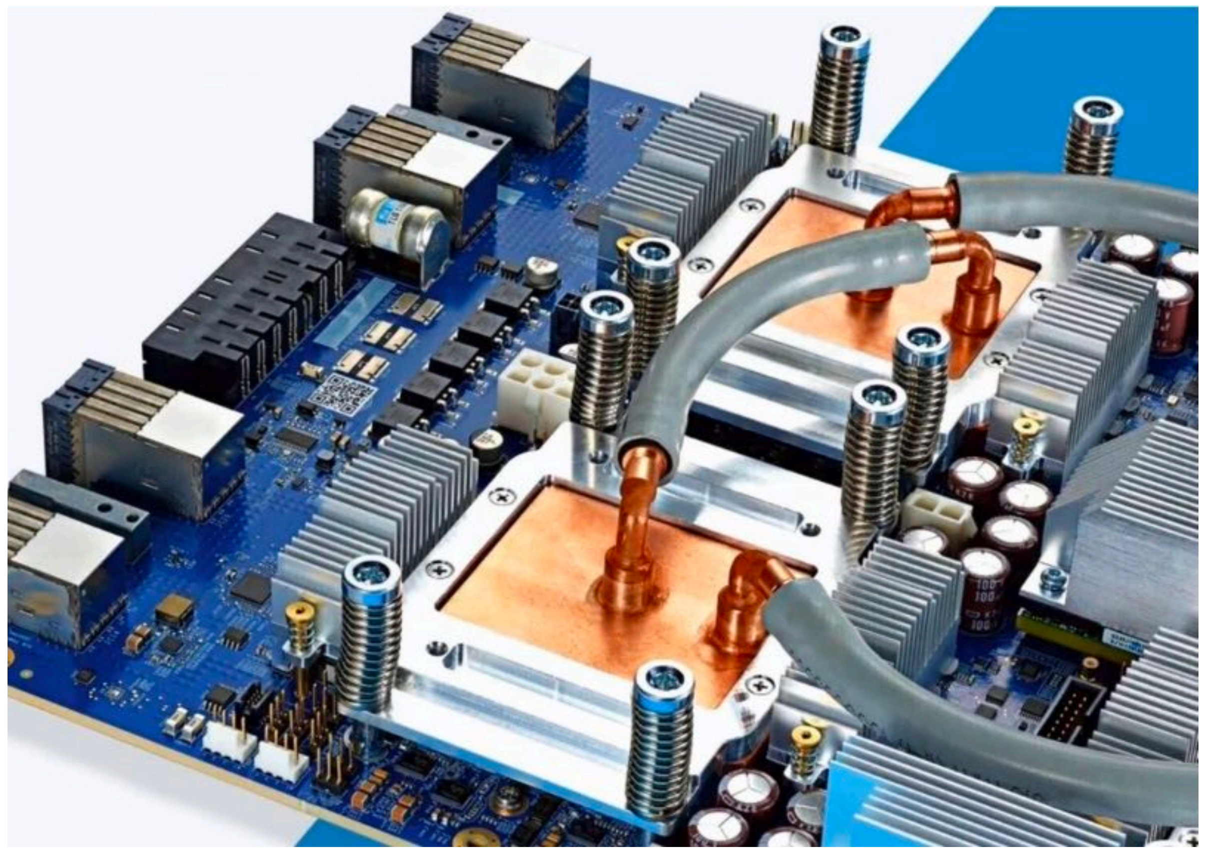
Lecture 7:

Hardware Acceleration of DNNs

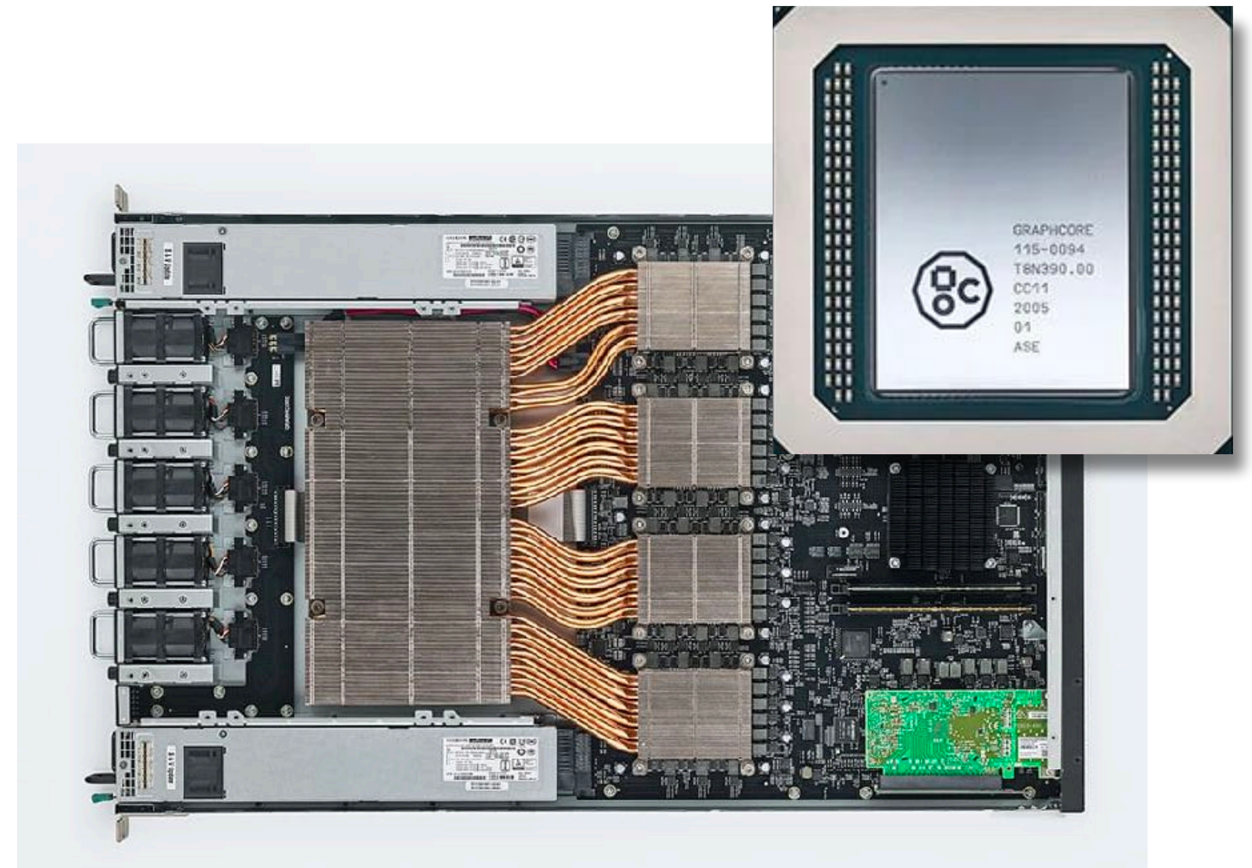
Visual Computing Systems

Stanford CS348K, Spring 2021

Hardware acceleration of DNN inference/training



Google TPU3

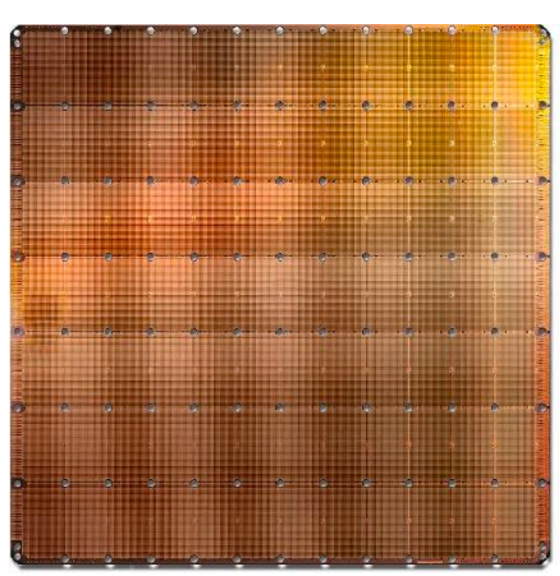


GraphCore IPU

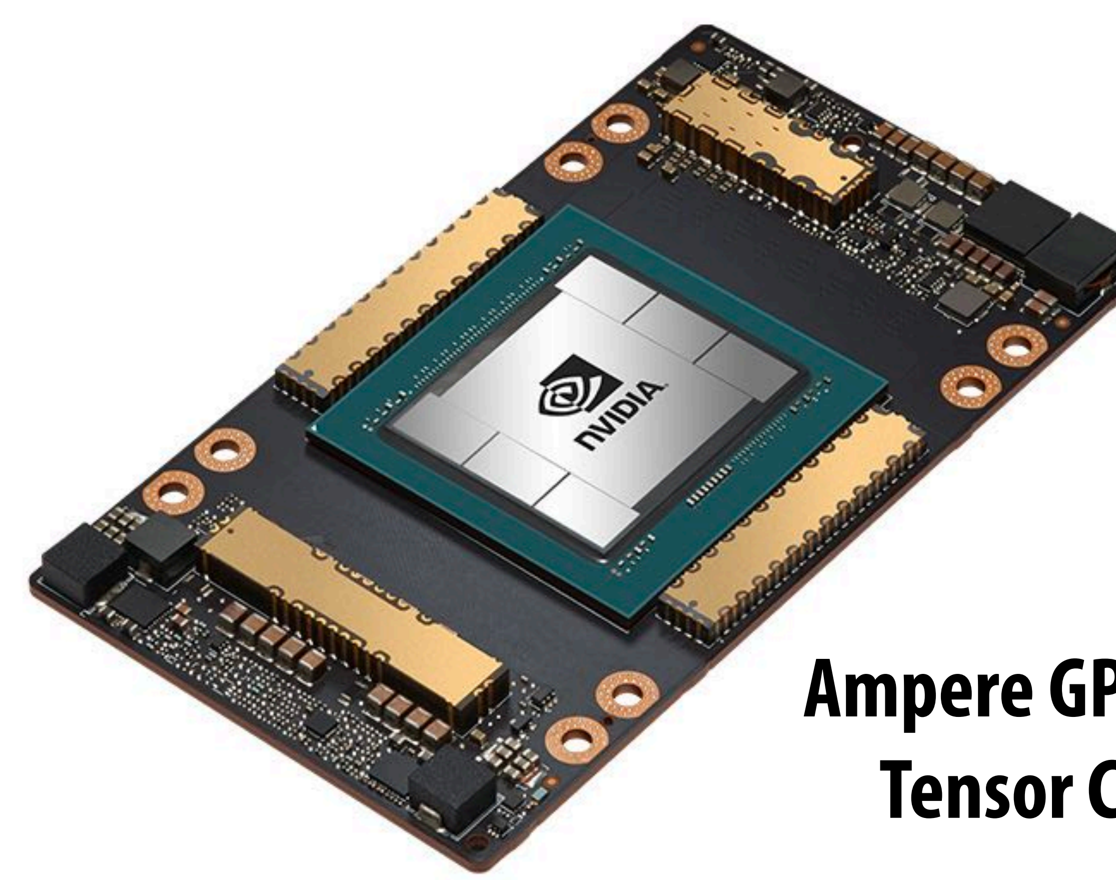
Apple Neural Engine

Intel Deep Learning
Inference Accelerator

SambaNova
Cardinal SN10



Cerebras Wafer Scale Engine



Ampere GPU with
Tensor Cores

Investment in AI hardware

SambaNova Systems Raises $676M in Series D, Surpasses $5B Valuation and Becomes World's Best-Funded AI Startup

SoftBank Vision Fund 2 leads round backing breakthrough platform that delivers unprecedented AI capability and accessibility to customers worldwide

April 13, 2021 09:00 AM Eastern Daylight Time

PALO ALTO, Calif.--(BUSINESS WIRE)--SambaNova Systems, the company building the industry's most advanced software, hardware and services to run AI applications, today announced a $676 million Series D funding round led by SoftBank Vision Fund 2*. The round includes additional new investors Temasek and GIC, plus existing backers including funds and accounts managed by BlackRock, Intel Capital, GV (formerly Google Ve

"We're here to revolutionize the AI market, and this round greatly accelerates that mission"

Tweet this

This Se
and ro

Now th
world,
legacy
hardwa
solutio

"We're here to revolutionize the AI market, and this round gre
founder and CEO. "Traditional CPU and GPU architectures ha
to solve humanity's greatest technology challenges, a new ap
to see a wealth of prudent investors validate that."

SambaNova's flagship offering is Dataflow-as-a-Service (Daa
to jump-start enterprise-level AI initiatives, augmenting organ
centers, allowing the organization to focus on its business objectives instead of infrastructure.

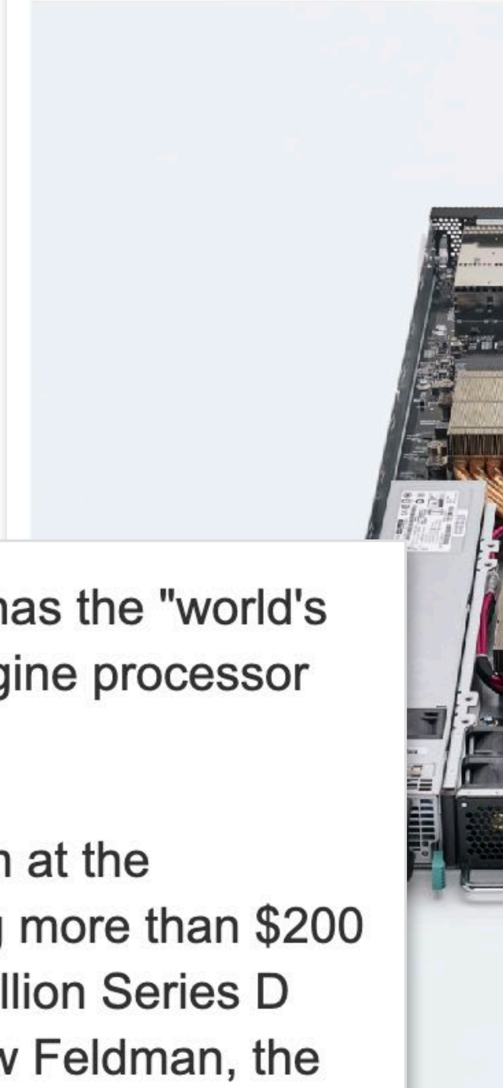
Artificial intelligence chip startup Cerebras Systems claims it has the "world's fastest AI supercomputer," thanks to its large Wafer Scale Engine processor that comes with 400,000 compute cores.

The Los Altos, Calif.-based startup introduced its CS-1 system at the Supercomputing conference in Denver last week after raising more than $200 million in funding from investors, most recently with an $88 million Series D round that was raised in November 2018, according to Andrew Feldman, the founder and CEO of Cerebras who was previously an executive at AMD.

AI chipmaker Graphcore raises $222M at a $2.77B valuation and puts an IPO in its sights

Ingrid Lunden @ingridlunden / 10:59 PM PST • December 28, 2020

Comment



Applications based on artificial intelligence — whether they are systems running autonomous services, platforms being used in drug development or to predict the spread of a virus, traffic management for 5G networks or something else altogether — require an unprecedented amount of computing power to run. And today, one of the big names in the world of designing and

Groq Closes $300 Million Fundraise

Wed, April 14, 2021, 6:00 AM · 4 min read

With Investment Co-Led by Tiger Global Management and D1 Capital, Groq Is Well Capitalized for Accelerated Growth

MOUNTAIN VIEW, Calif., April 14, 2021 /PRNewswire/ -- Groq Inc., a leading innovator compute accelerators for artificial intelligence (AI), machine learning (ML) and high performance computing, today announced that it has closed its Series C fundraising closed $300 million in new funding, co-led by Tiger Global Management and D1 Capita participation from The Spruce House Partnership and Addition, the venture firm foun by Lee Fixel. This round brings Groq's total funding to $367 million, of which $300 mill has been raised since the second-half of 2020, a direct result of strong customer endorsement since the company launched its first product.

groq™

Groq logo

Intel Acquires Artificial Intelligence Chipmaker Habana Labs

Combination Advances Intel's AI Strategy, Strengthens Portfolio of AI Accelerators for the Data Center

SANTA CLARA Calif., Dec. 16, 2019 – Intel Corporation today announced that it has acquired Habana Labs, an Israel-based developer of programmable deep learning accelerators for the data center for approximately $2 billion. The combination strengthens Intel's artificial intelligence (AI) portfolio and accelerates its efforts in the nascent, fast-growing AI silicon market, which Intel expects to be greater than $25 billion by 2024[1].

"This acquisition advances our AI strategy, which is to provide customers with solutions to fit every performance need – from the intelligent edge to the data center," said Navin Shenoy, executive vice president and general manager of the Data Platforms Group at Intel. "More specifically, Habana turbo-charges our AI offerings for the data center with a high-performance training processor family and a standards-based programming environment to address evolving AI workloads."

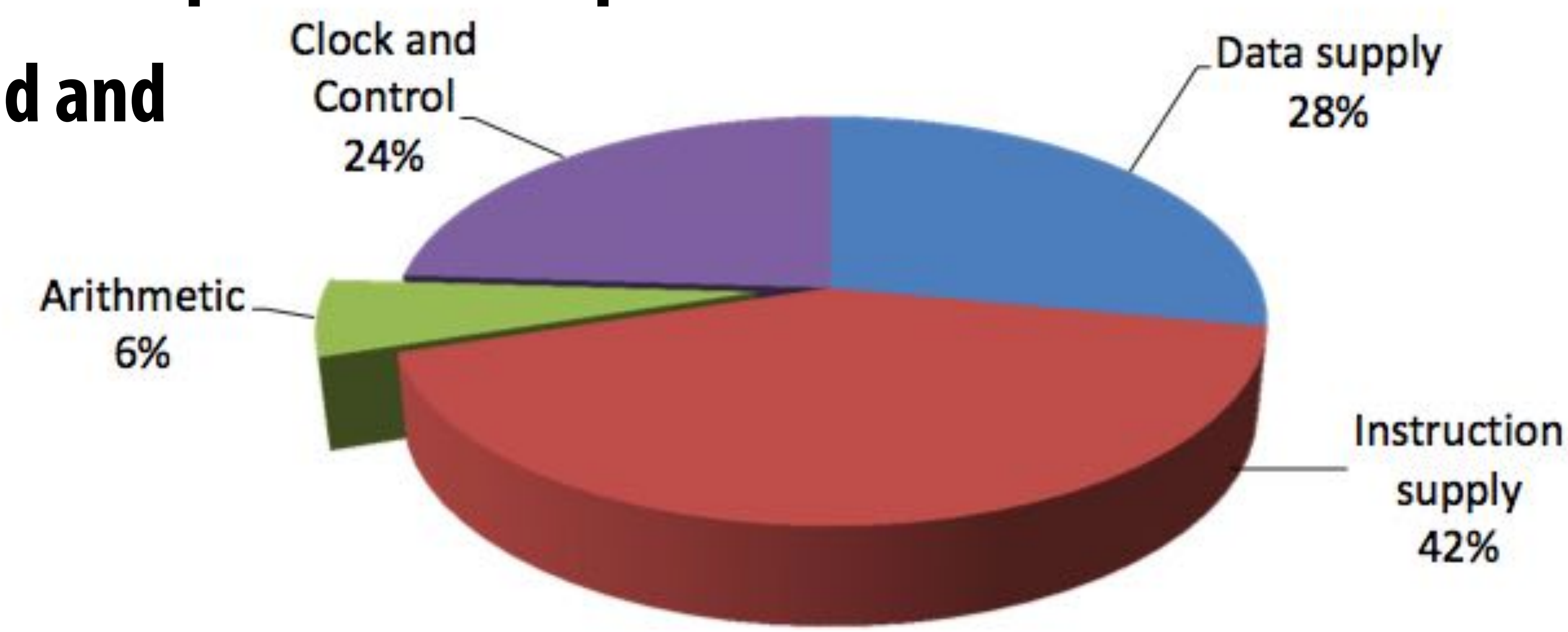
Two computer architecture reminders

Compute specialization = energy efficiency

- Rules of thumb: compared to high-quality C code on CPU...

- Throughput-maximized processor architectures: e.g., GPU cores
  - Approximately 10x improvement in perf / watt
  - Assuming code maps well to wide data-parallel execution and is compute bound

- Fixed-function ASIC ("application-specific integrated circuit")
  - Can approach 100-1000x or greater improvement in perf/watt
  - Assuming code is compute bound and and is not floating-point math



Efficient Embedded Computing [Dally et al. 08]
[Figure credit Eric Chung]

[Source: Chung et al. 2010 , Dally 08]

Data movement has high energy cost

- **Rule of thumb in modern system design: always seek to reduce amount of data movement in a computer**

- **"Ballpark" numbers**

  [Sources: Bill Dally (NVIDIA), Tom Olson (ARM)]

  - Integer op: ~ 1 pJ *
  - Floating point op: ~20 pJ *
  - Reading 64 bits from small local SRAM (1mm away on chip): ~ 26 pJ
  - Reading 64 bits from low power mobile DRAM (LPDDR): ~1200 pJ

- **Implications**
  - Reading 10 GB/sec from memory: ~1.6 watts
  - Entire power budget for mobile GPU: ~1 watt
    (remember phone is also running CPU, display, radios, etc.)
  - iPhone 6 battery: ~7 watt-hours (note: my Macbook Pro laptop: 99 watt-hour battery)
  - Exploiting locality matters!!!

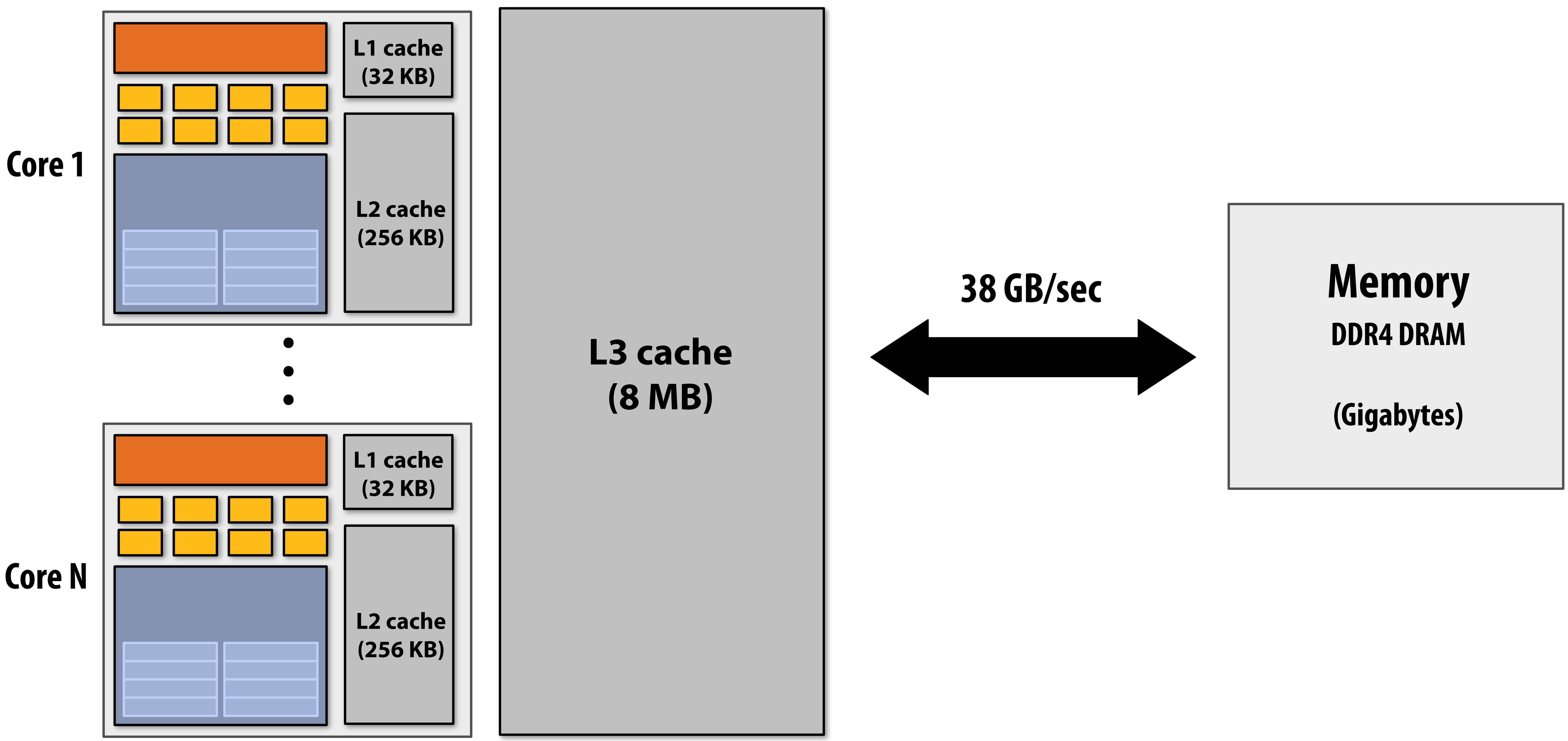
* Cost to just perform the logical operation, not counting overhead of instruction decode, load data from registers, etc.

On-chip caches locate data near processing

Processors run efficiently when data is resident in caches

Caches reduce memory access latency *

Caches reduce the energy cost of data access



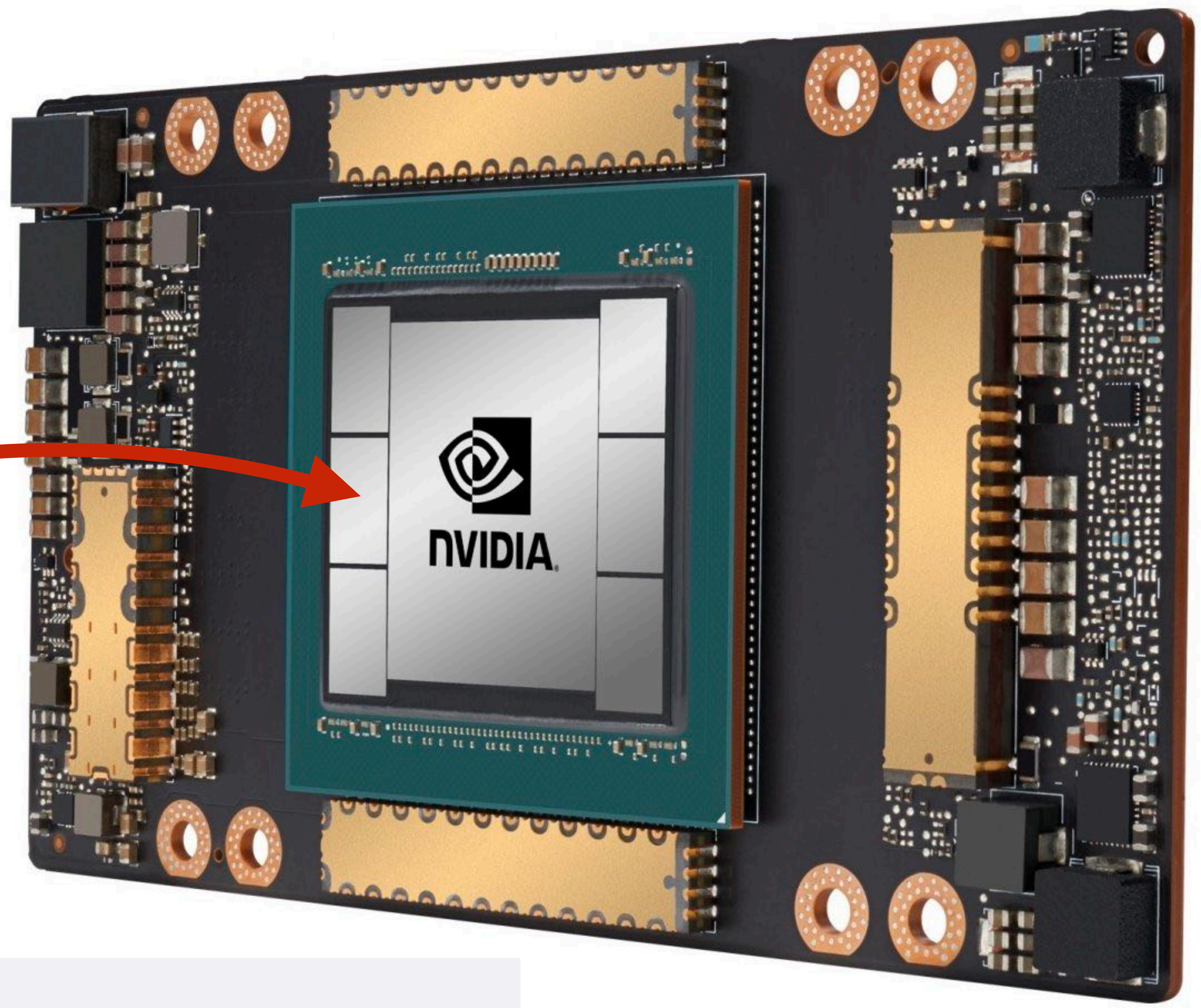
* Caches also provide high bandwidth data transfer to CPU

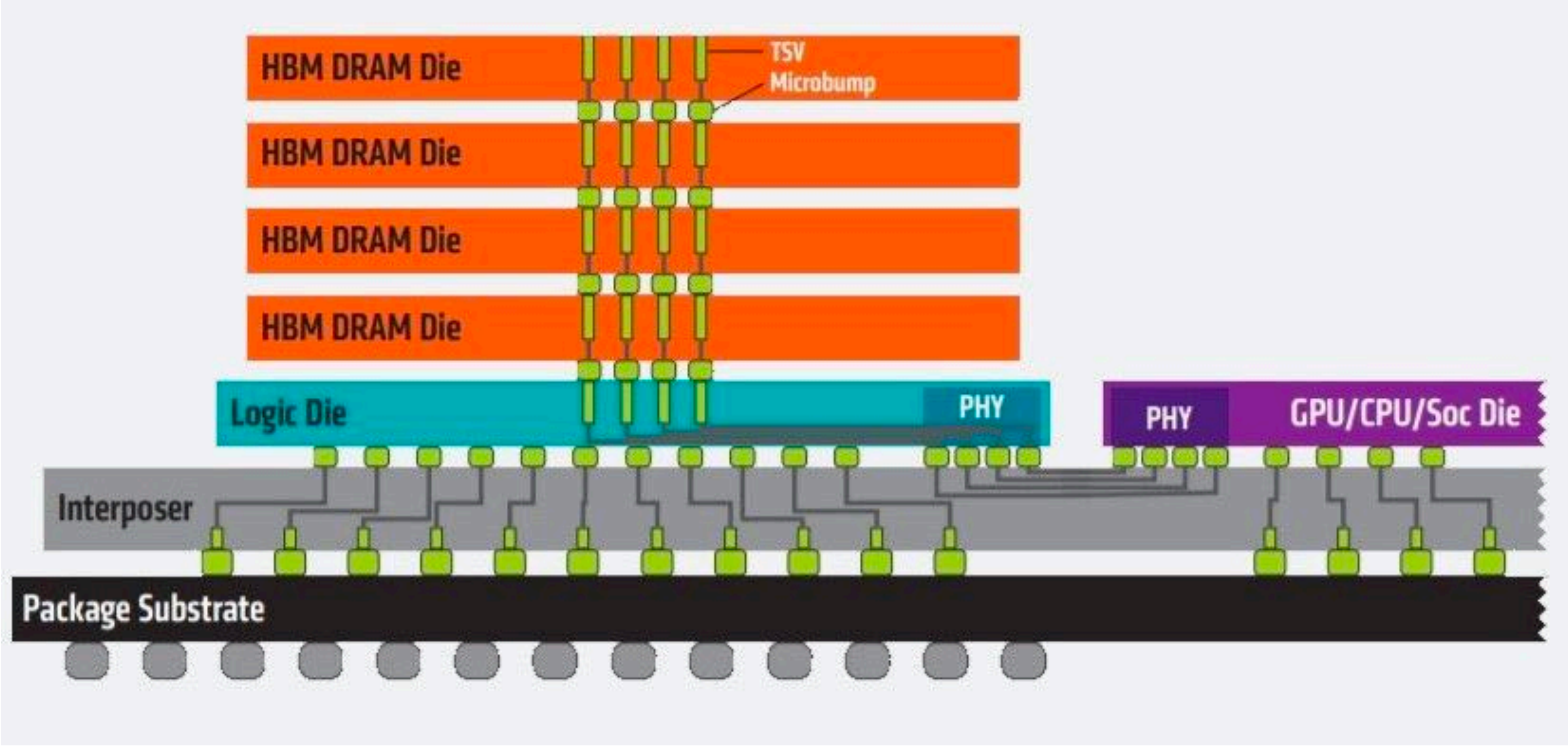
Memory stacking locates memory near chip

Example:
NVIDIA A100 GPU

Up to 80 GB HMB2 stacked memory
2 TB/sec memory bandwidth

Also note: A100 has 40 MB L2 cache
(increased from 6.1 MB on V100)





Improving hardware efficiency for DNN operations

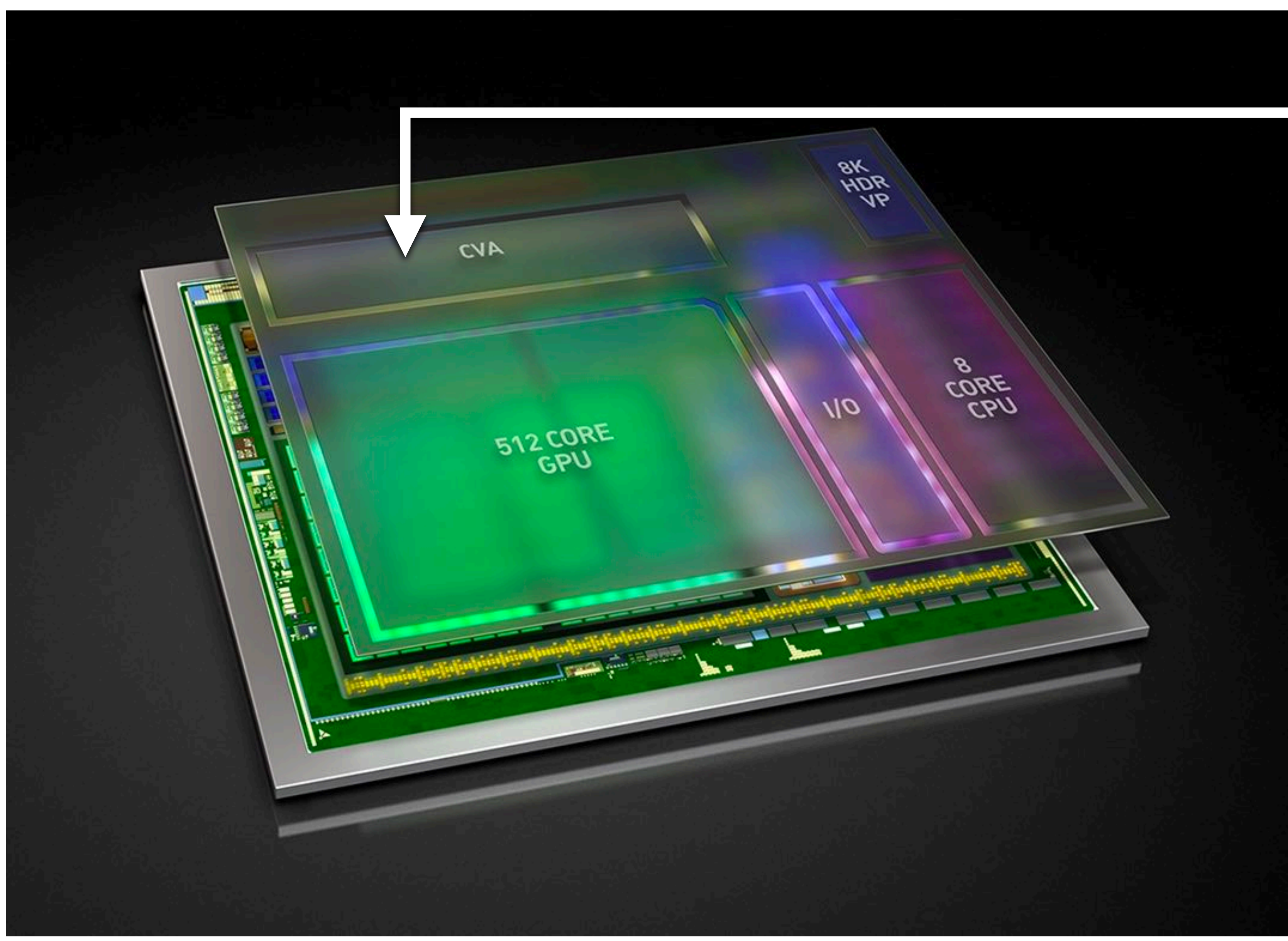
Amortize overhead of instruction stream control using more complex instructions

- Fused multiply add ($ax + b$)
- 4-component dot product $x = A$ dot B
- 4x4 matrix multiply
  - $AB + C$ for 4x4 matrices A, B, C

- Key principle: amortize cost of instruction stream processing across many operations of a single complex instruction

Efficiency estimates *

- Estimated overhead of programmability (instruction stream, control, etc.)
  - Half-precision FMA (fused multiply-add) 2000%
  - Half-precision DP4 (vec4 dot product) 500%
  - Half-precision 4x4 MMA (matrix-matrix multiply + accumulate) 27%



NVIDIA Xavier (SoC for automotive domain)

Features a Computer Vision Accelerator (CVA), a custom module for deep learning acceleration (large matrix multiply unit)

~ 2x more efficient than NVIDIA V100 MMA instruction despite being highly specialized component. (includes optimization of gating multipliers if either operand is zero)

* Estimates by Bill Dally using academic numbers, SysML talk, Feb 2018

Ampere GPU SM (A100)

Single instruction to perform 2x8x4x8 int16 + 8x8 int32 ops

Each SM core has:

64 fp32 ALUs (mul-add)

32 int32 ALUs

4 "tensor cores"

Execute 8x4 x 4x8 matrix mul-add instr

A x B + C for matrices A,B,C

A, B stored as fp16, accumulation with fp32 C

There are 108 SM cores in the GA100 GPU:

6,912 fp32 mul-add ALUs

432 tensor cores

1.4 GHz max clock

= 19.5 TFLOPs fp32

+ 312 TFLOPs (fp16/32 mixed) in tensor cores

Google TPU
(version 1)

Google's TPU (v1)

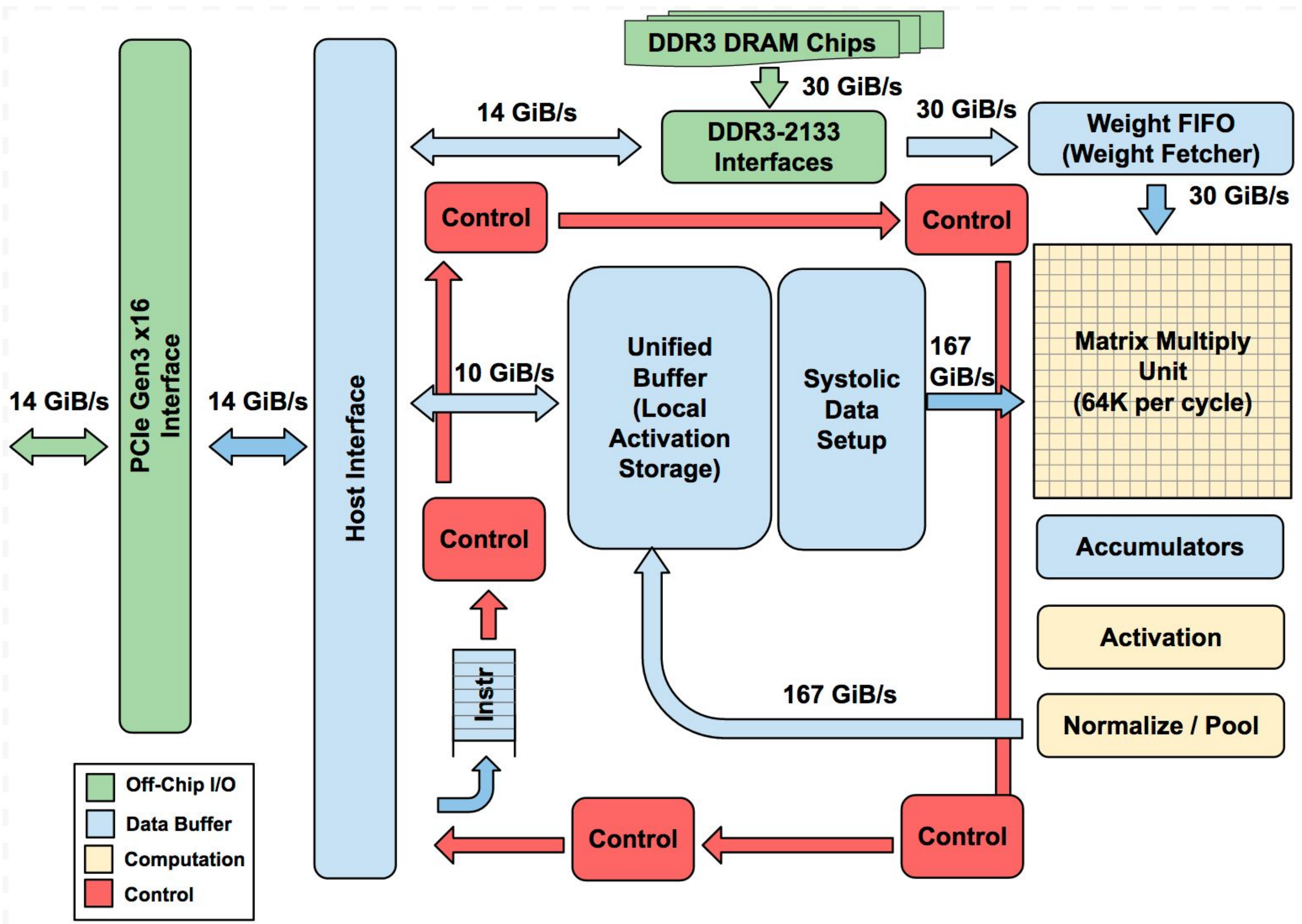
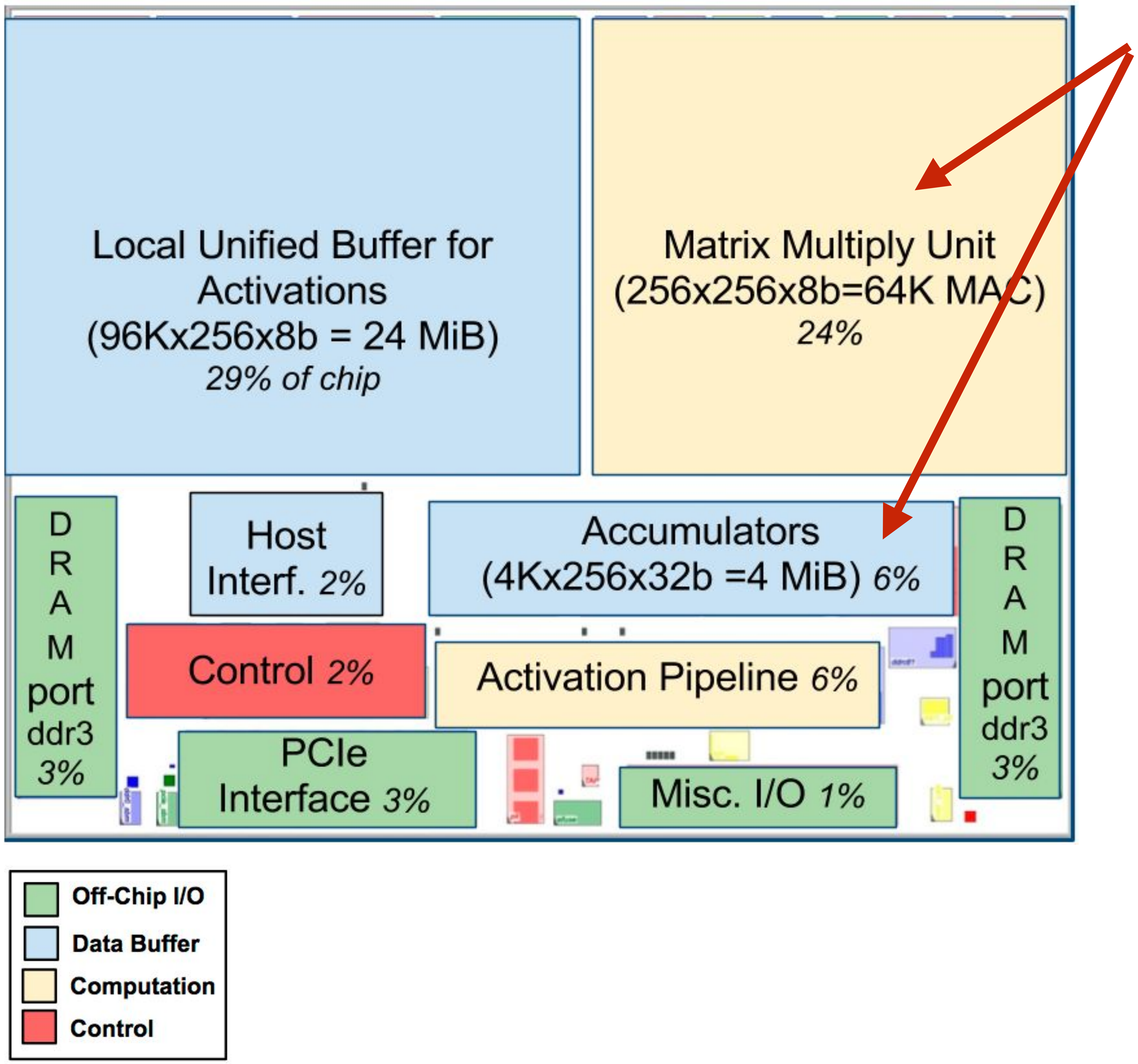


Figure credit: Jouppi et al. 2017

TPU area proportionality



Arithmetic units ~ 30% of chip

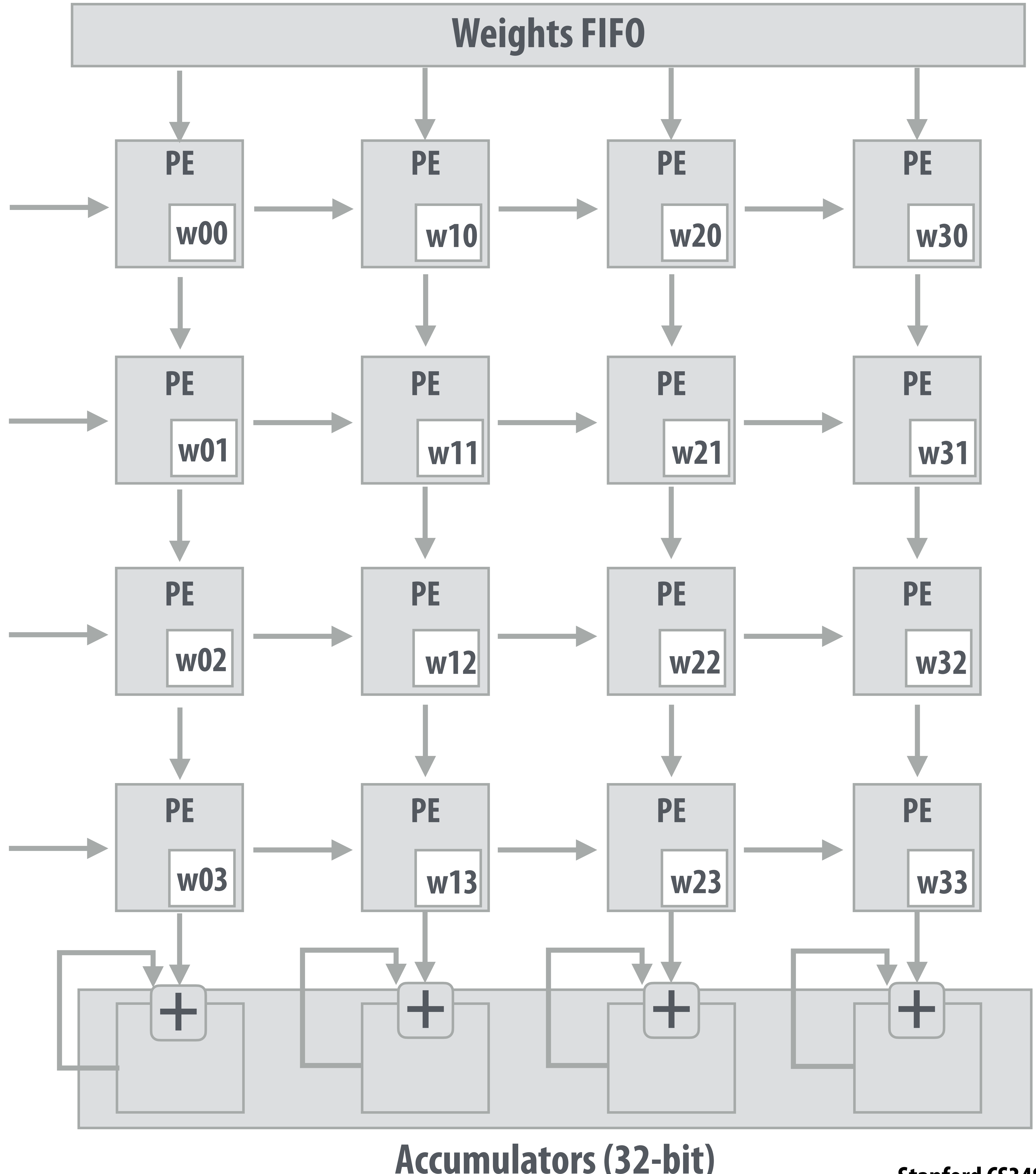
Note low area footprint of control

Key instructions:

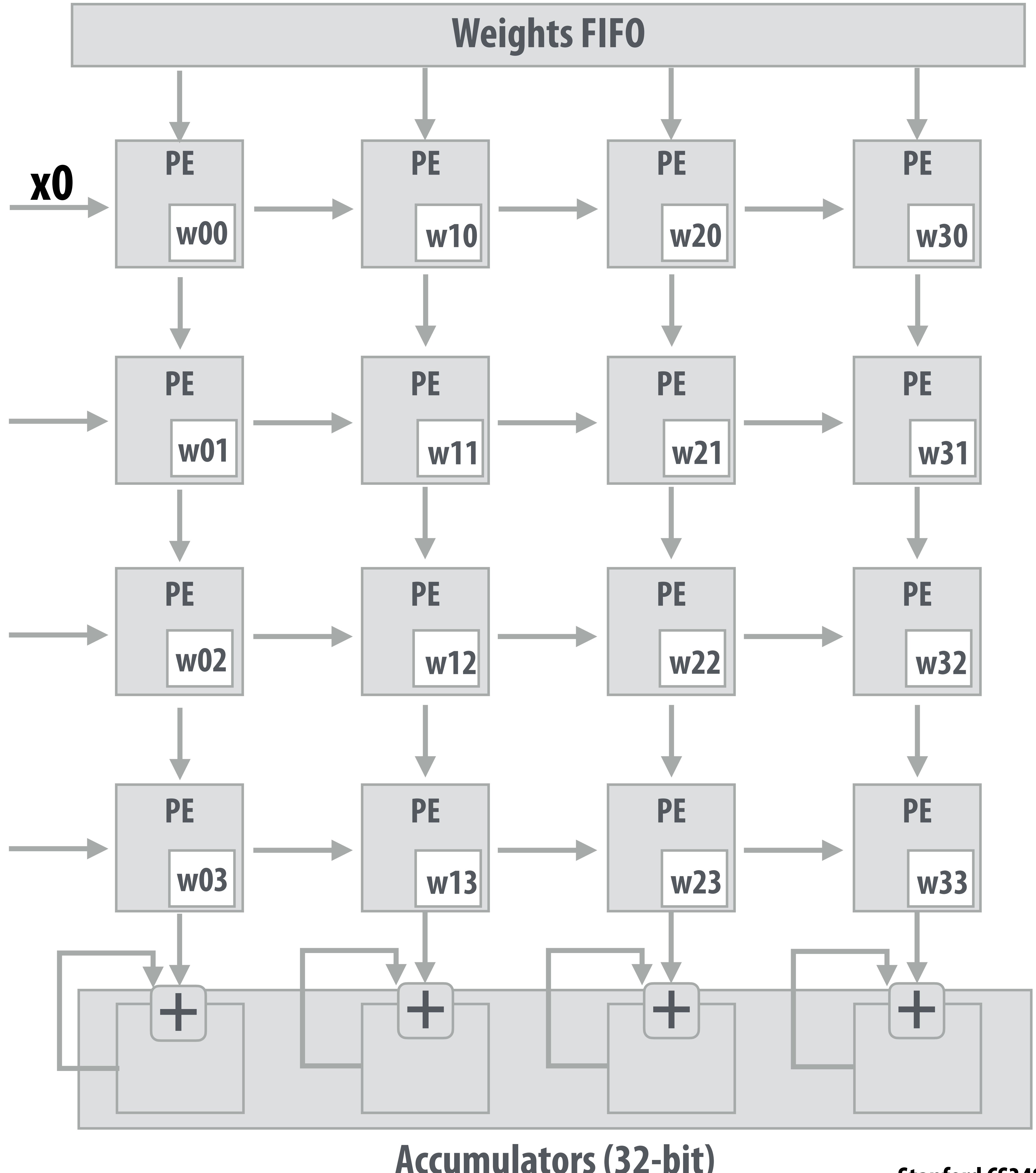
- **read host memory**
- **write host memory**
- **read weights**
- **matrix_multiply / convolve**
- **activate**

Figure credit: Jouppi et al. 2017

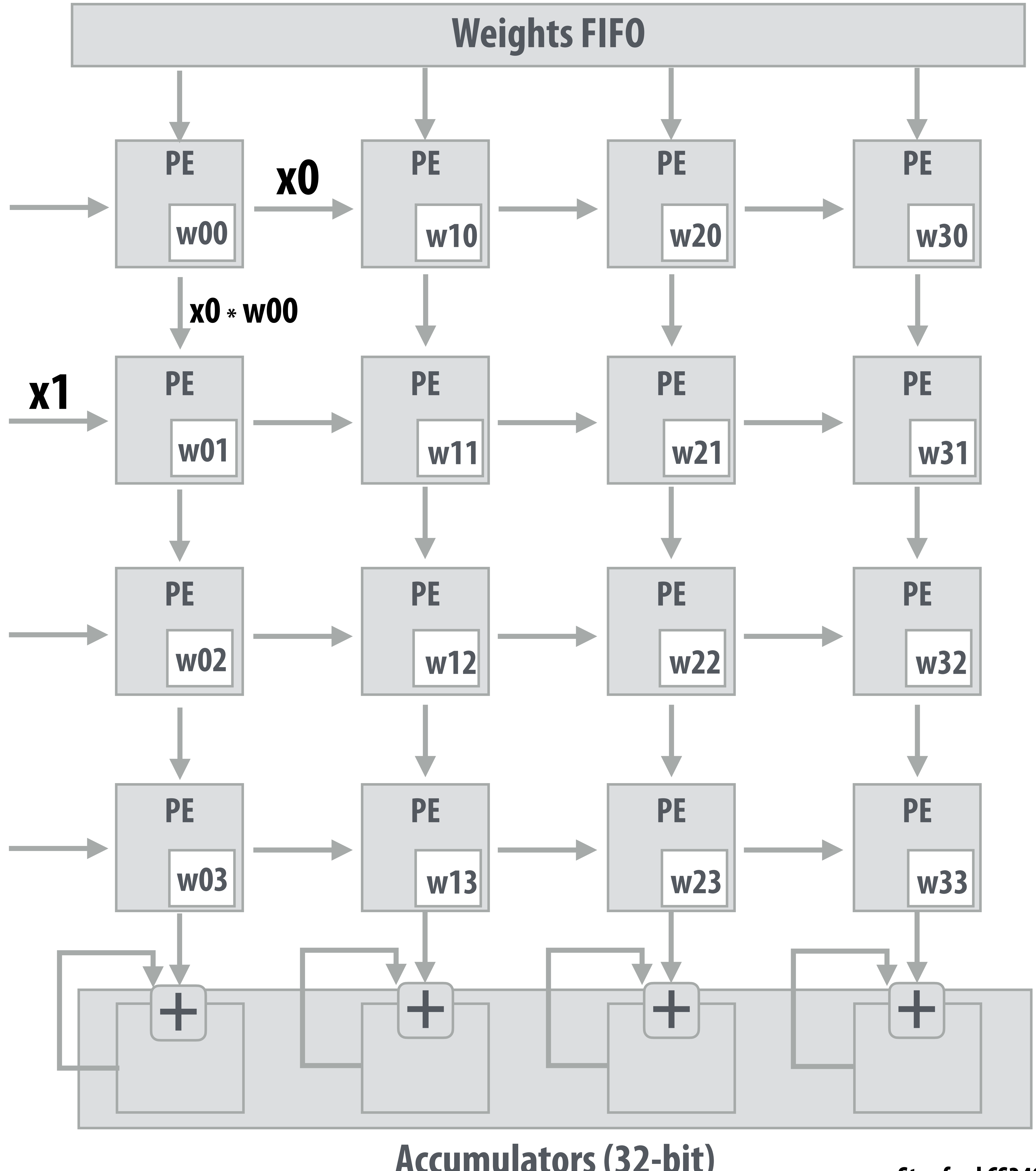
Systolic array (matrix vector multiplication example: $y=Wx$)



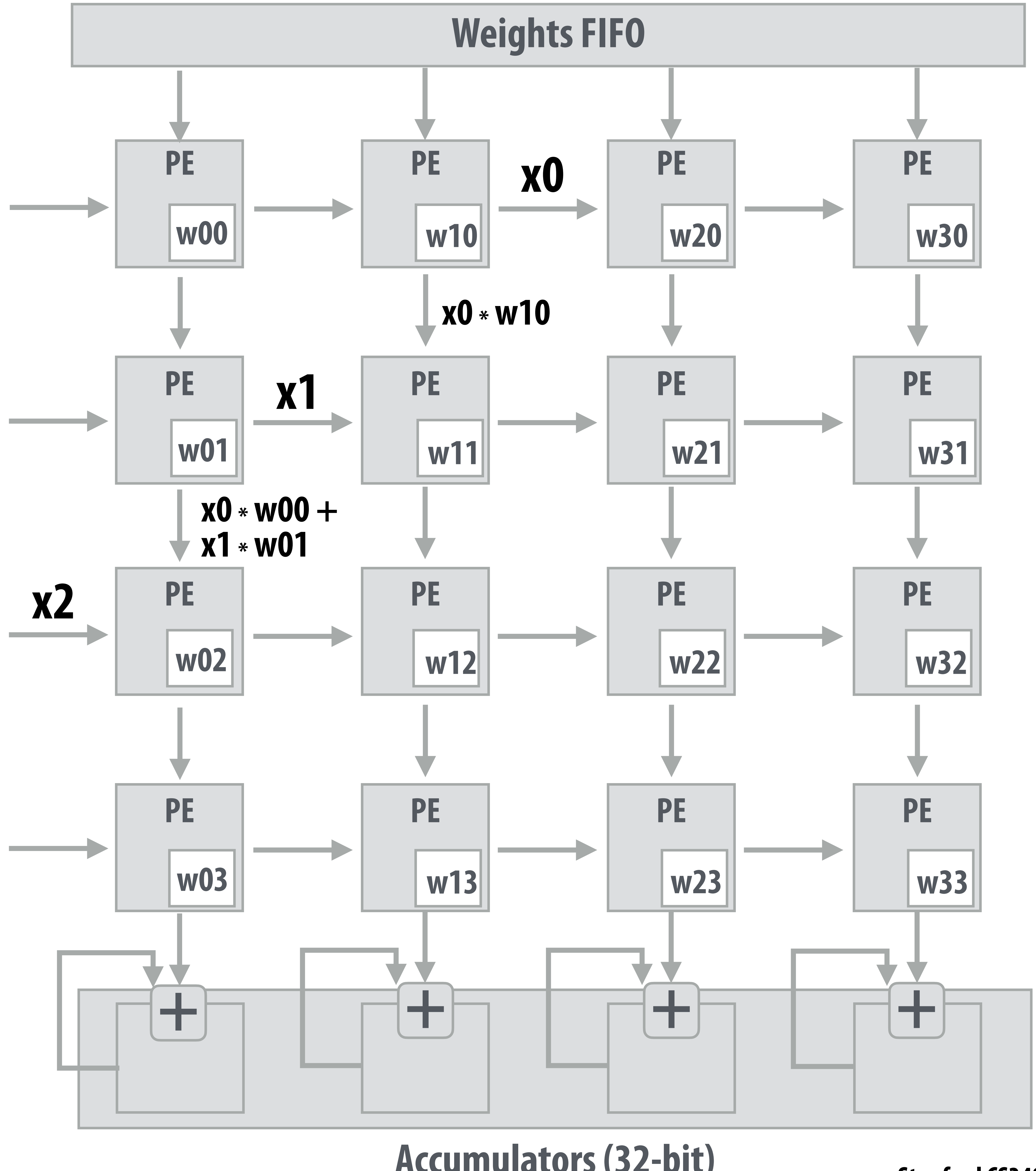
Systolic array (matrix vector multiplication example: $y=Wx$)



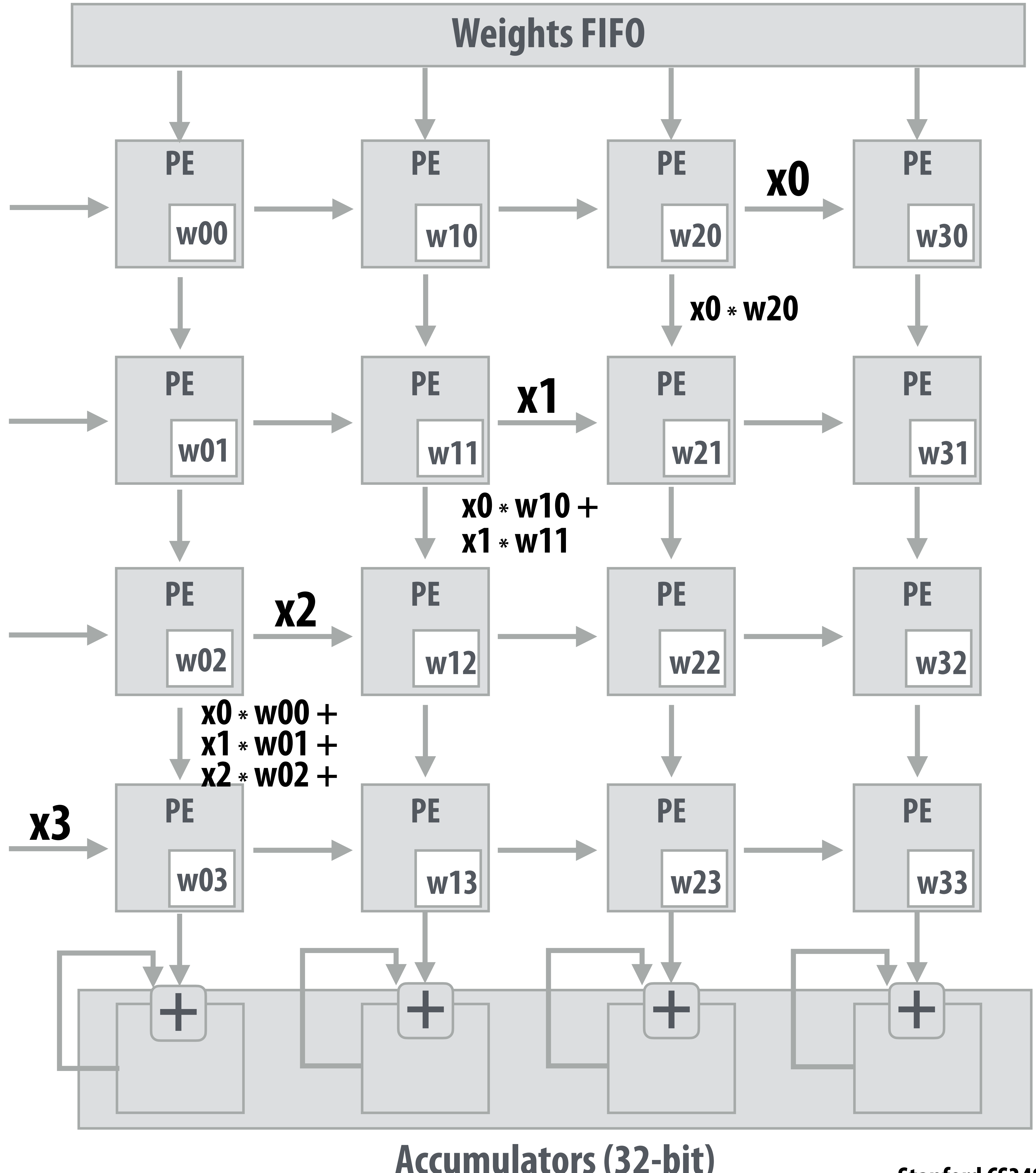
Systolic array (matrix vector multiplication example: $y=Wx$)



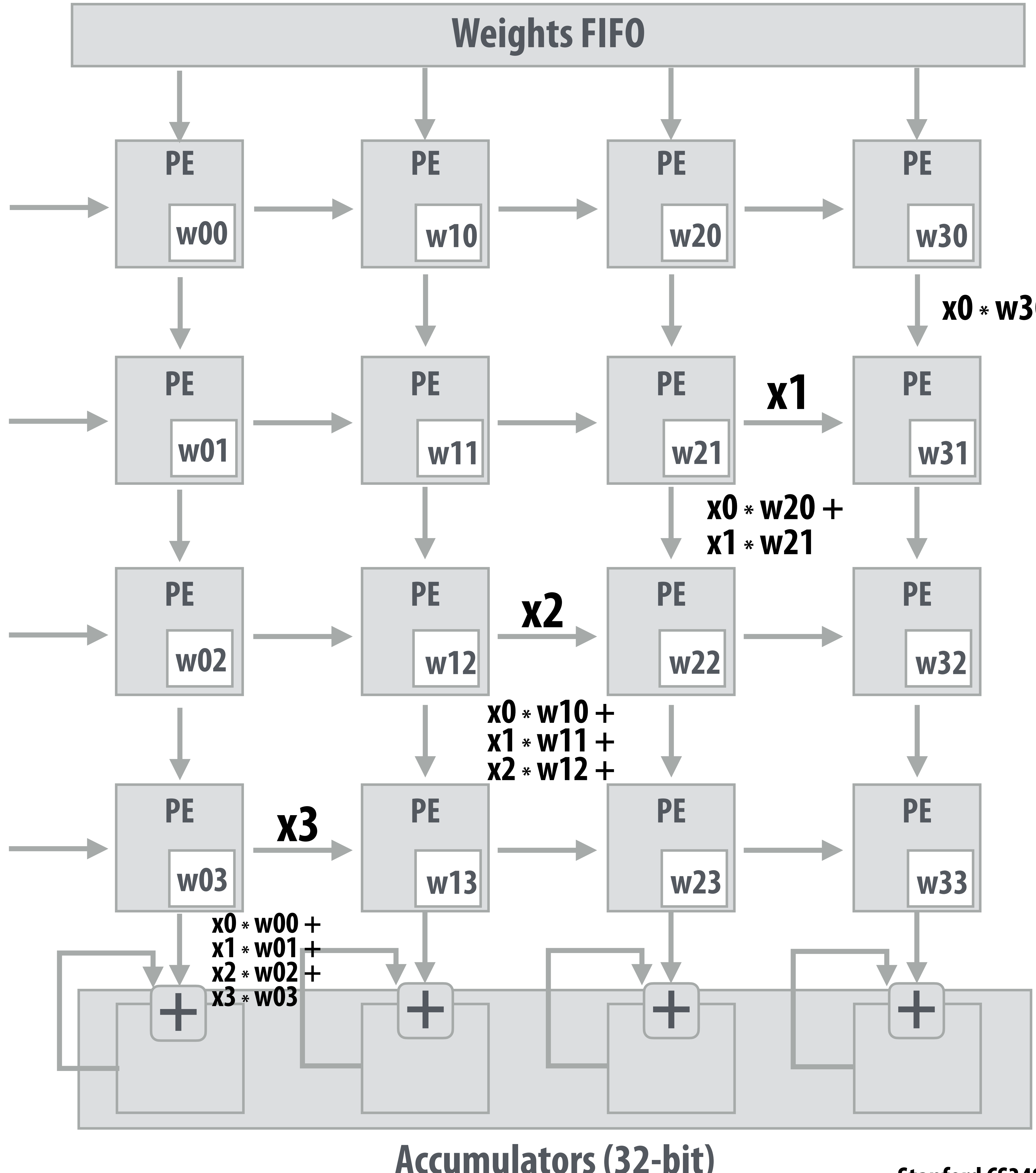
Systolic array (matrix vector multiplication example: $y=Wx$)



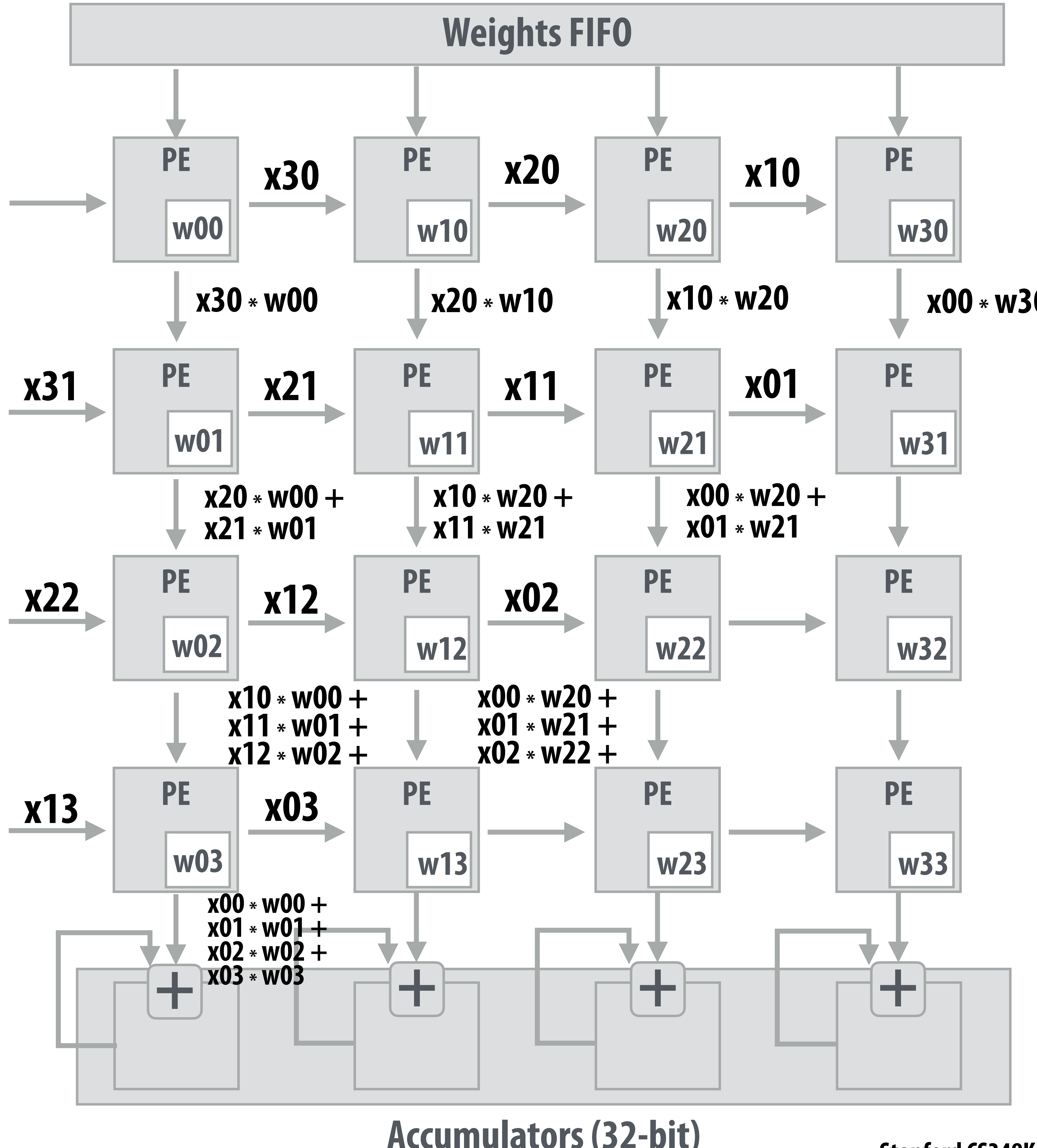
Systolic array (matrix vector multiplication example: $y = Wx$)



Systolic array (matrix vector multiplication example: $y=Wx$)



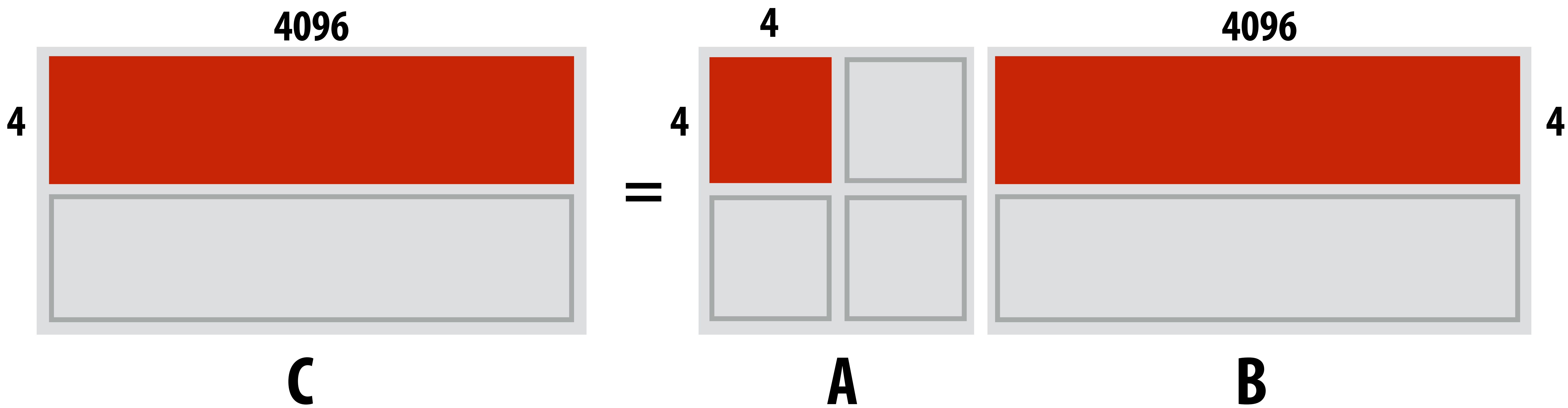
Systolic array (matrix matrix multiplication example: $Y=WX$)



Notice: need multiple 4x32bit accumulators to hold output columns

Building larger matrix-matrix multiplies

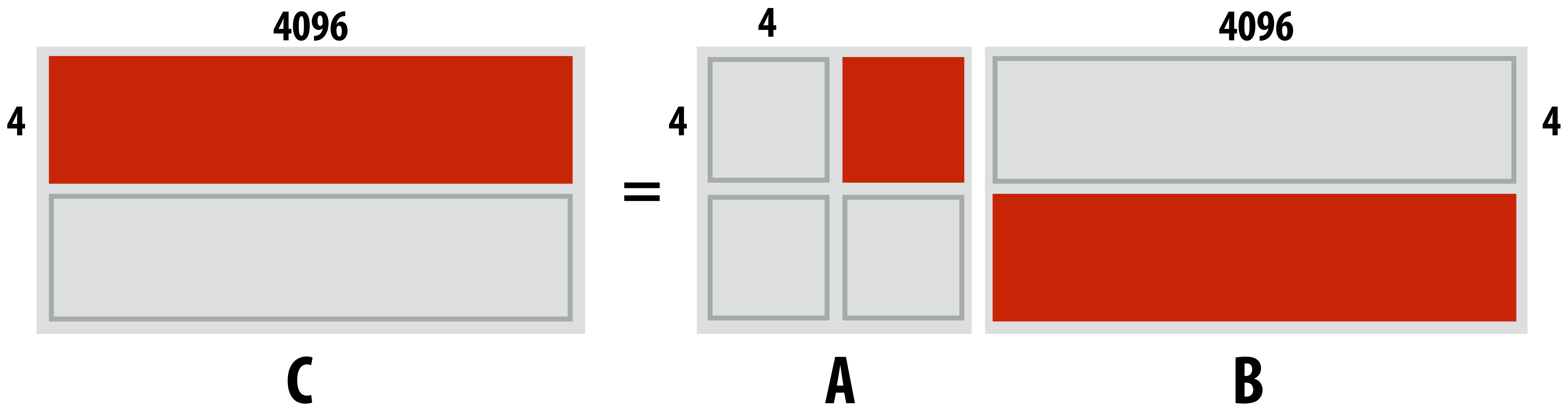
Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

Building larger matrix-matrix multiplies

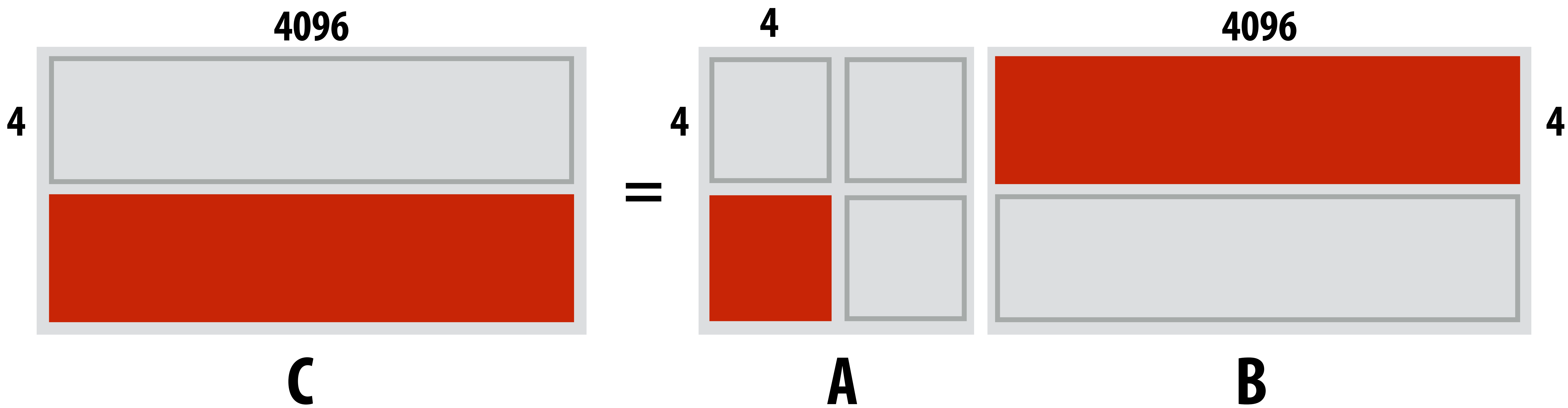
Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

Building larger matrix-matrix multiplies

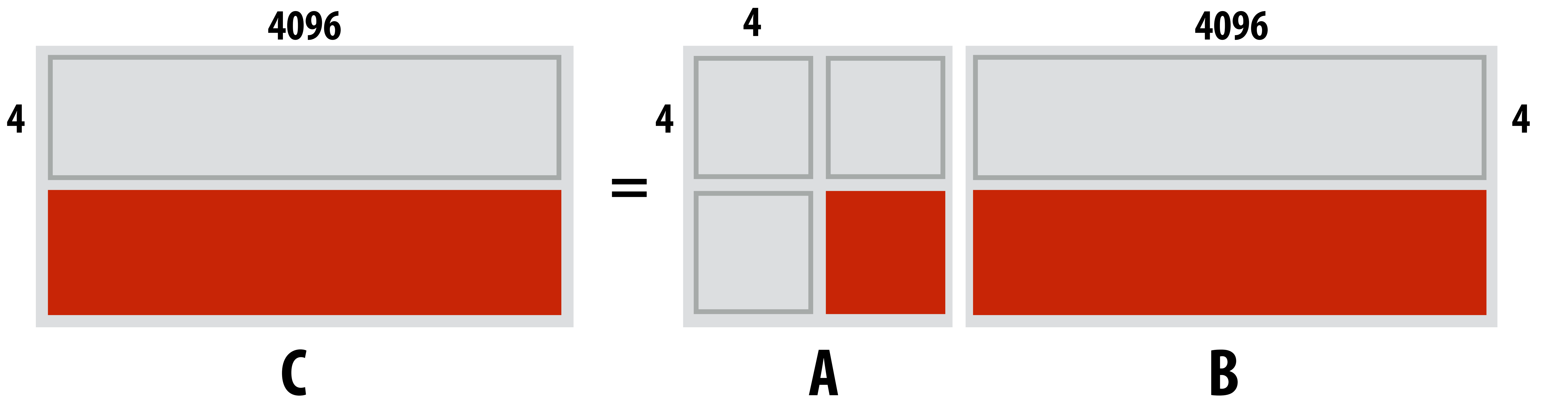
Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

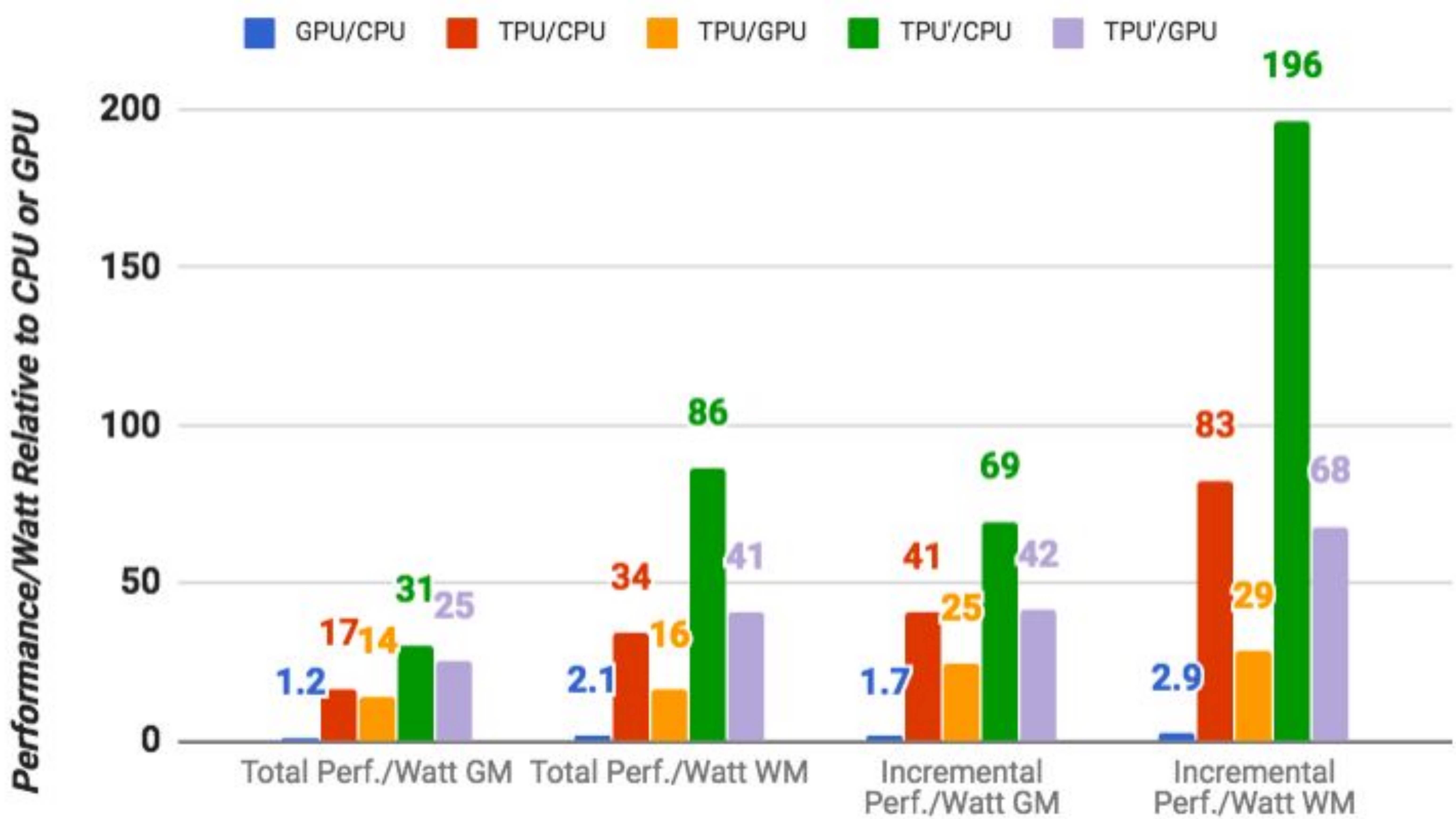
Building larger matrix-matrix multiplies

Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

TPU Performance/Watt



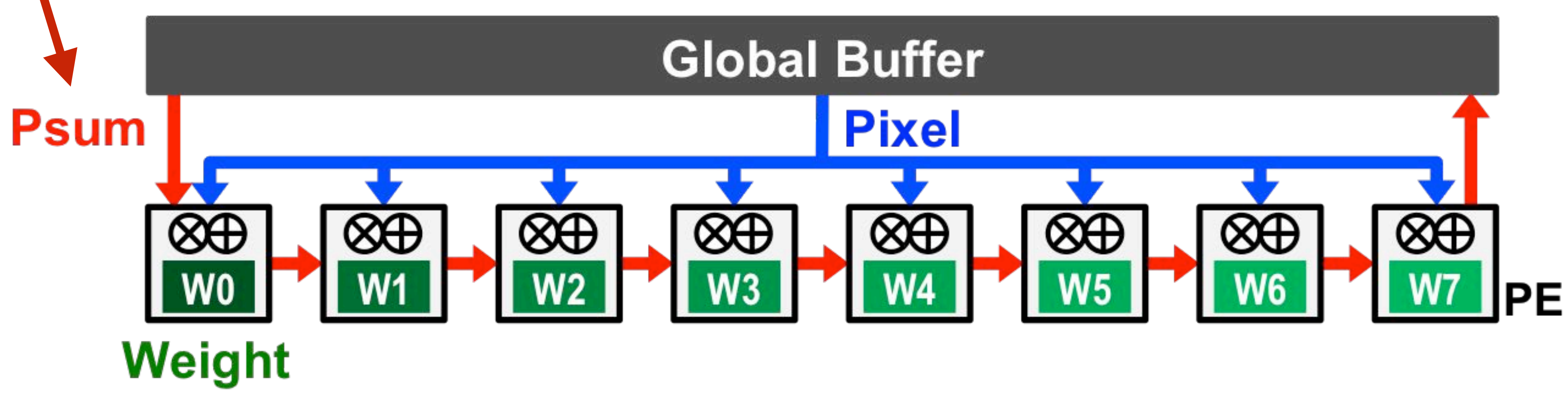
GM = geometric mean over all apps
WM = weighted mean over all apps

total = cost of host machine + CPU
incremental = only cost of TPU

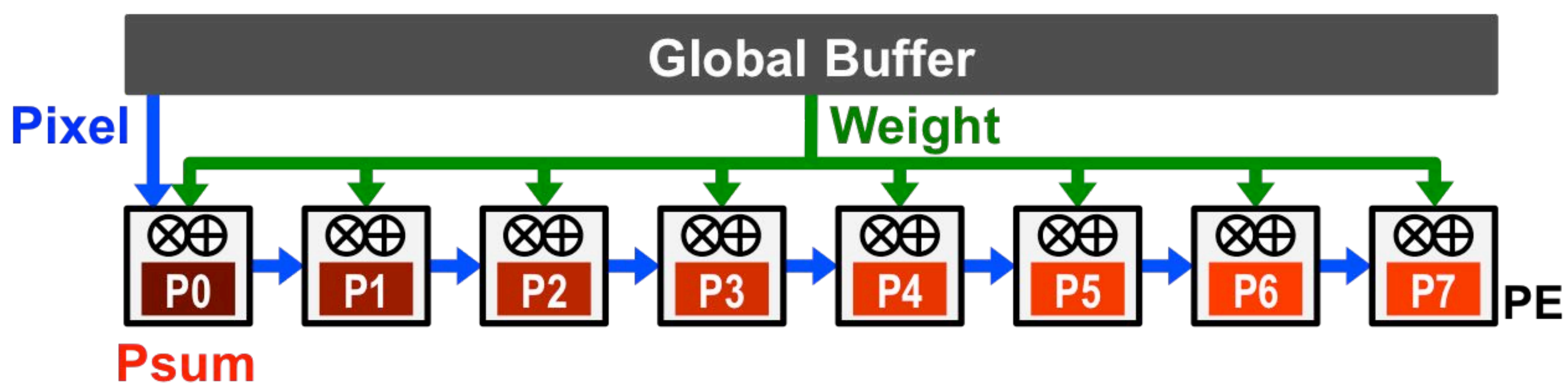
Figure credit: Jouppi et al. 2017

Alternative scheduling strategies

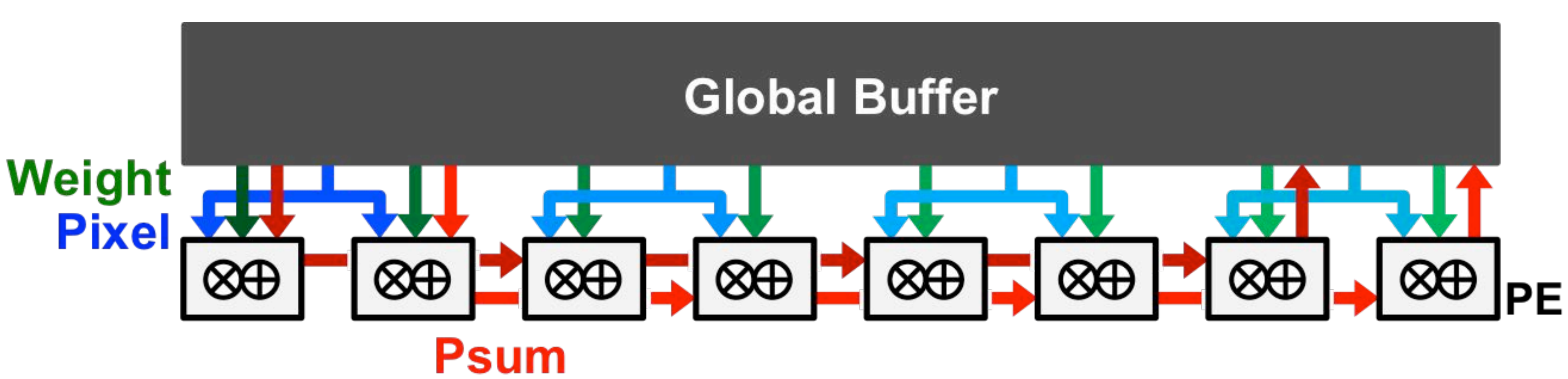
Psum = partial sum



(a) Weight Stationary



(b) Output Stationary



(c) No Local Reuse

TPU (v1) was "weight stationary":
weights kept in register at PE
each PE gets different pixel
partial sum pushed through array (array has one output)

"Output stationary":
each PE computes one output
push input pixel through array
each PE gets different weight
each PE accumulates locally into output

Takeaway: many DNN accelerators can be characterized by the data flow of input activations, weights, and outputs through the machine. (Just different "schedules"!)

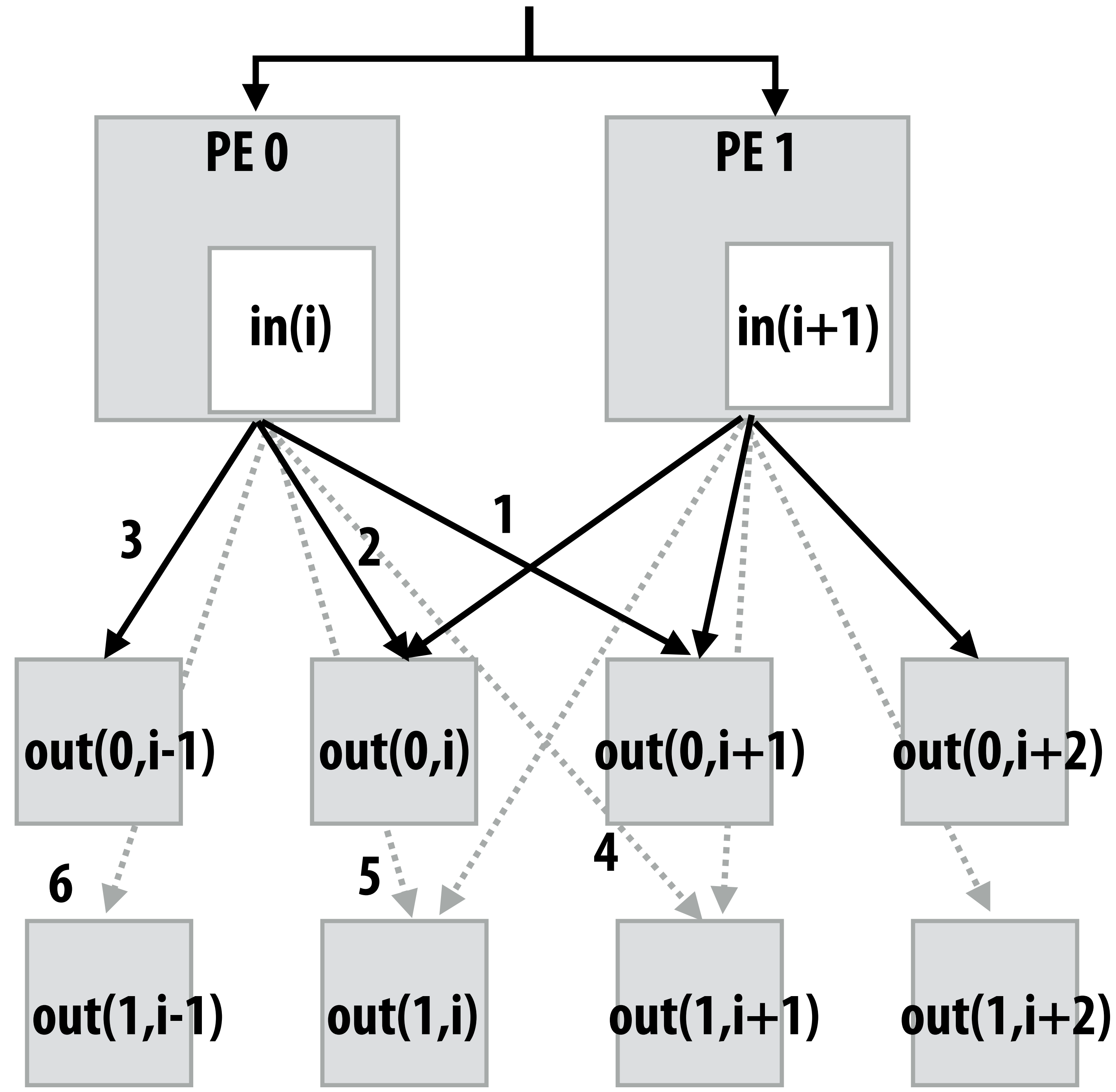
Figure credit: Sze et al. 2017

Input stationary design (dense 1D conv example)

Assume:
1D input/output
3-wide filters
2 output channels (K=2)

Stream Order	Weight
6	w(1,2)
5	w(1,1)
4	w(1,0)
3	w(0,2)
2	w(0,1)
1	w(0,0)

Stream of weights (2 1D filters of size 3)

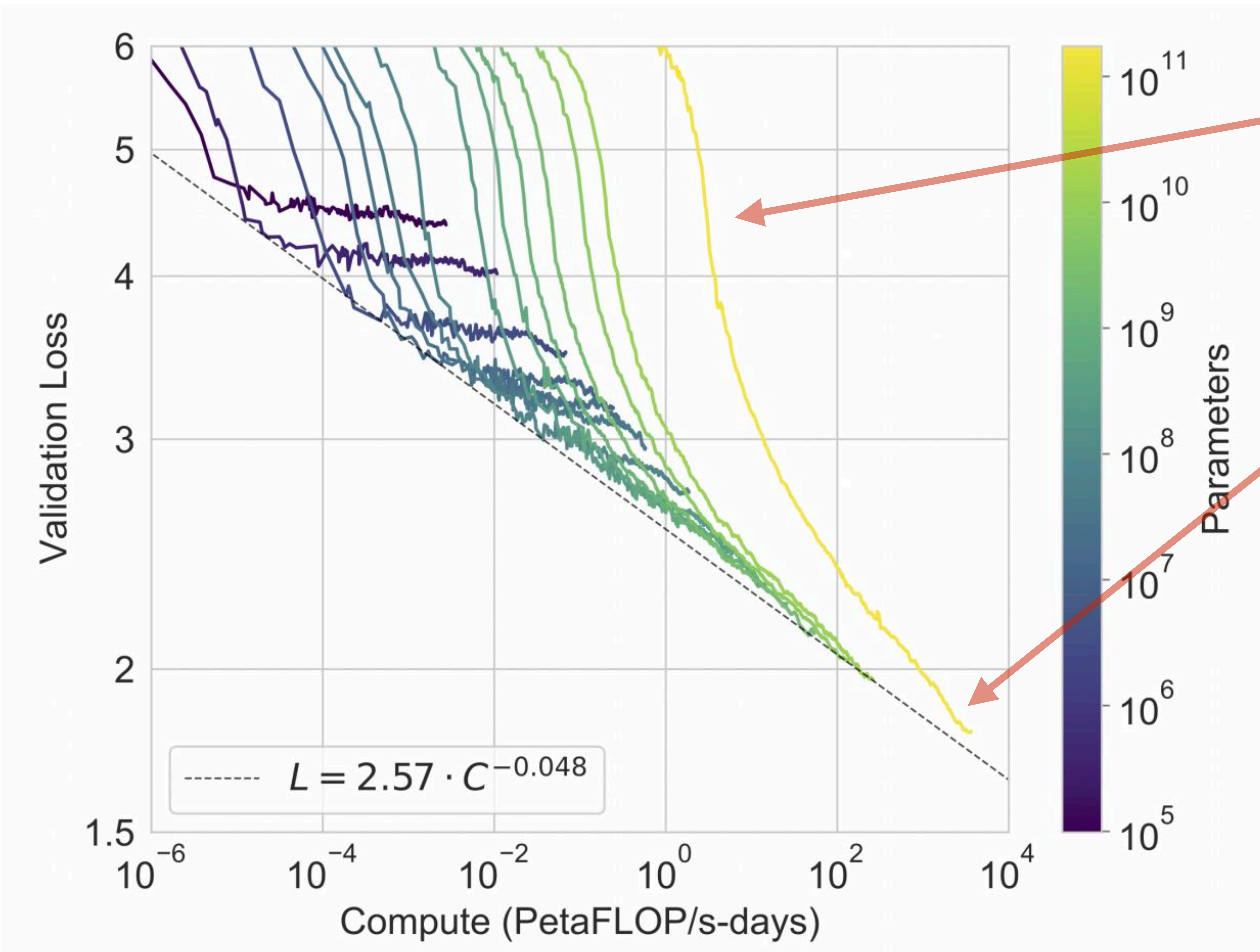


Processing elements (implement multiply)

Accumulators (implement +=)

Scaling up (for training big models)

Example: GPT-3 language model



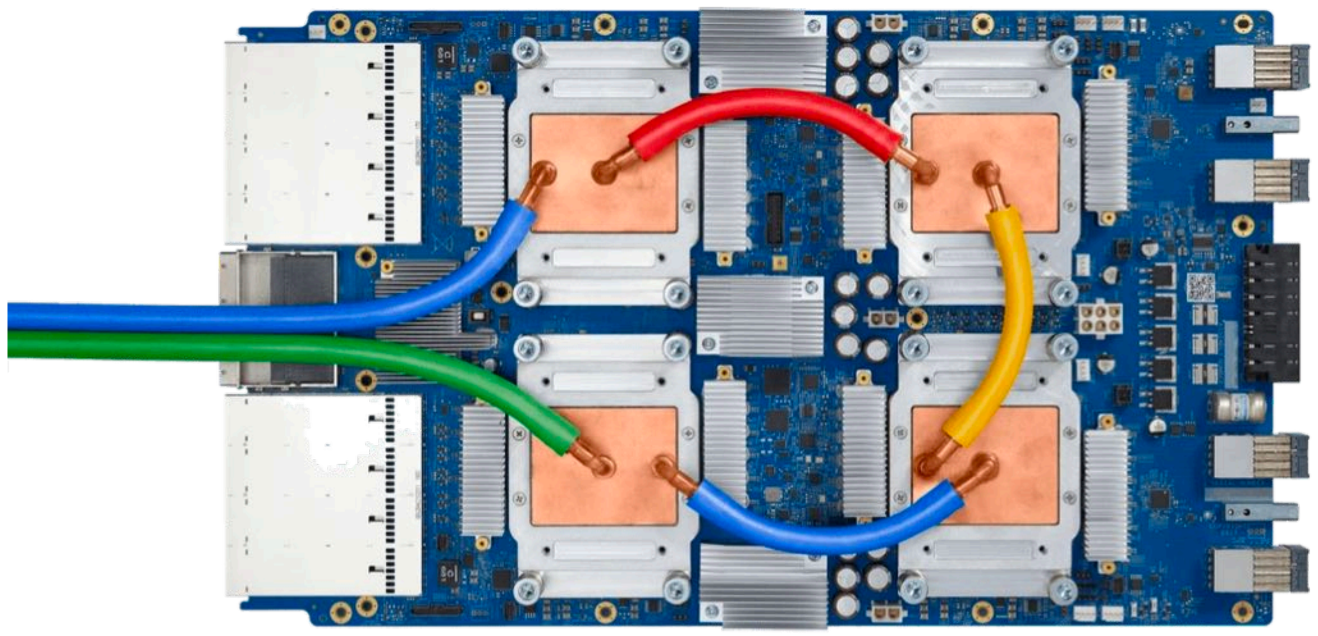
(Amount of training — note this is log scale)

Very big models + More training = Better accuracy

Power law effect: exponentially more compute to take constant step in accuracy

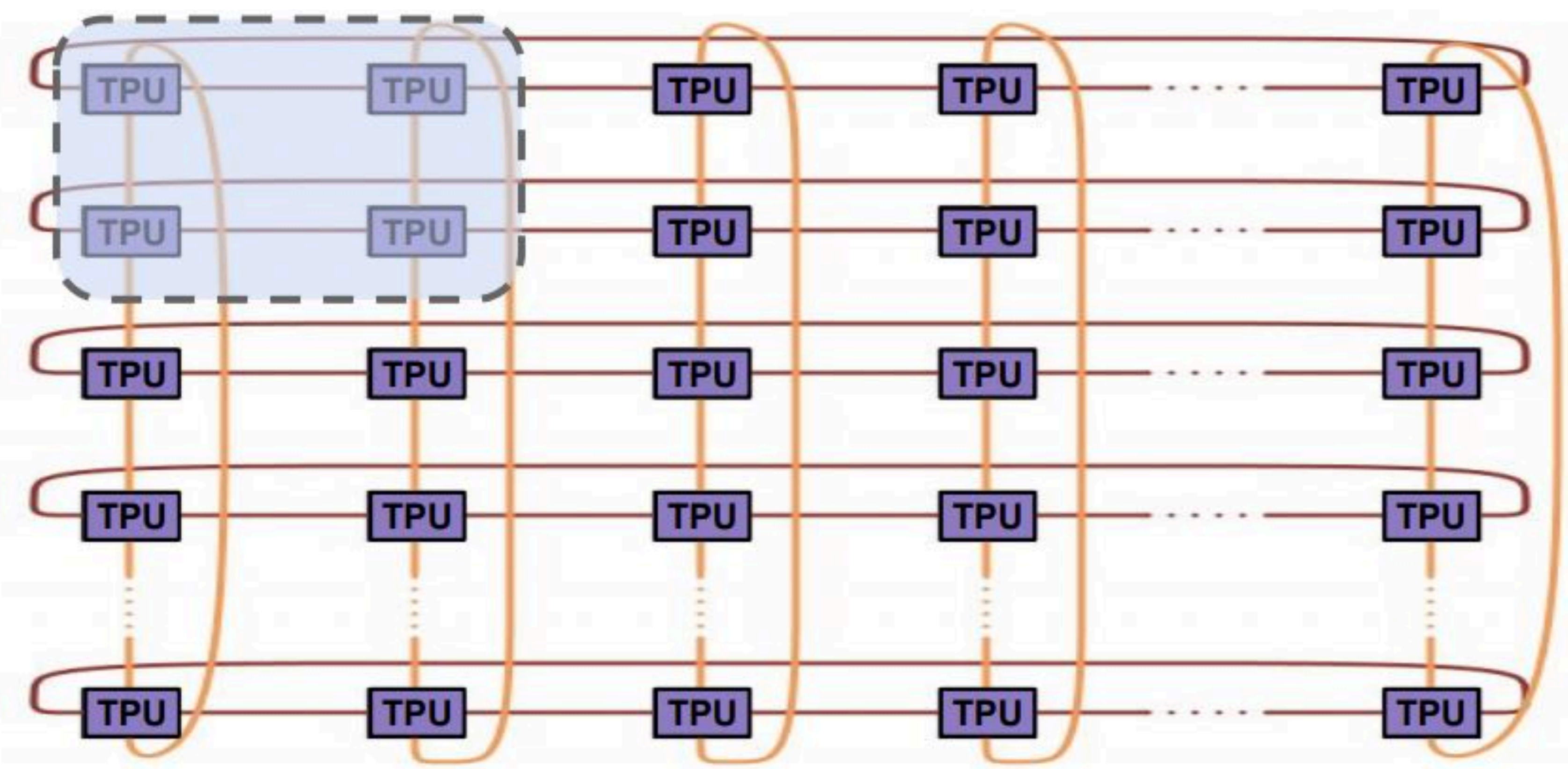
TPU v3 supercomputer

TPU v3 board
4 TPU3 chips

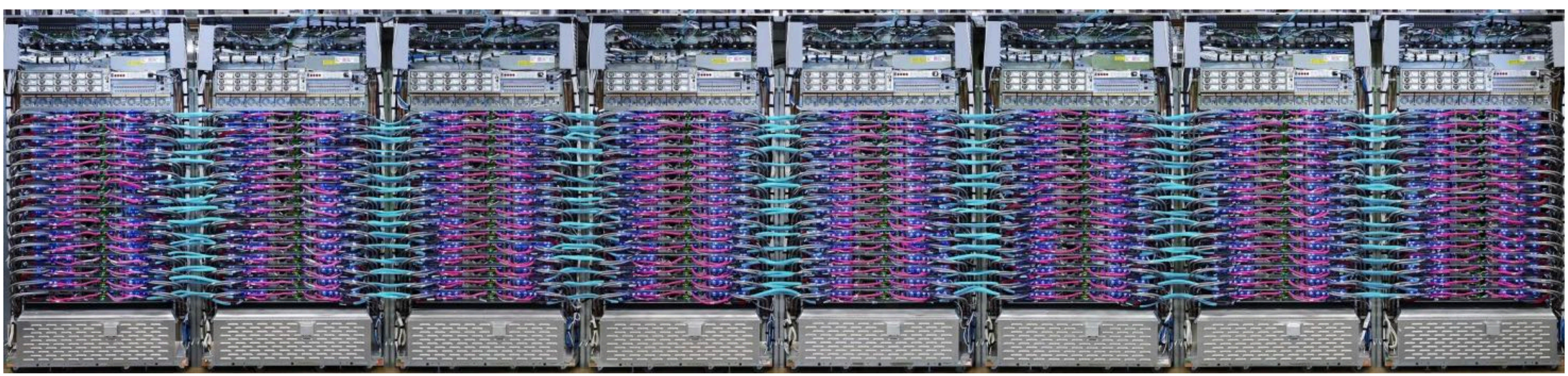


One TPU v3 board

TPUs connected by
2D Torus interconnect



TPU supercomputer (1024 TPU v3 chips)



Additional examples of “AI chips”

Key ideas:

1. Huge numbers of compute units

2. Huge amounts of on-chip storage to maintain input weights and intermediate values

GraphCore MK2 GC200 IPU

IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

1472 independent IPU-Core™

8832 independent program threads executing in parallel

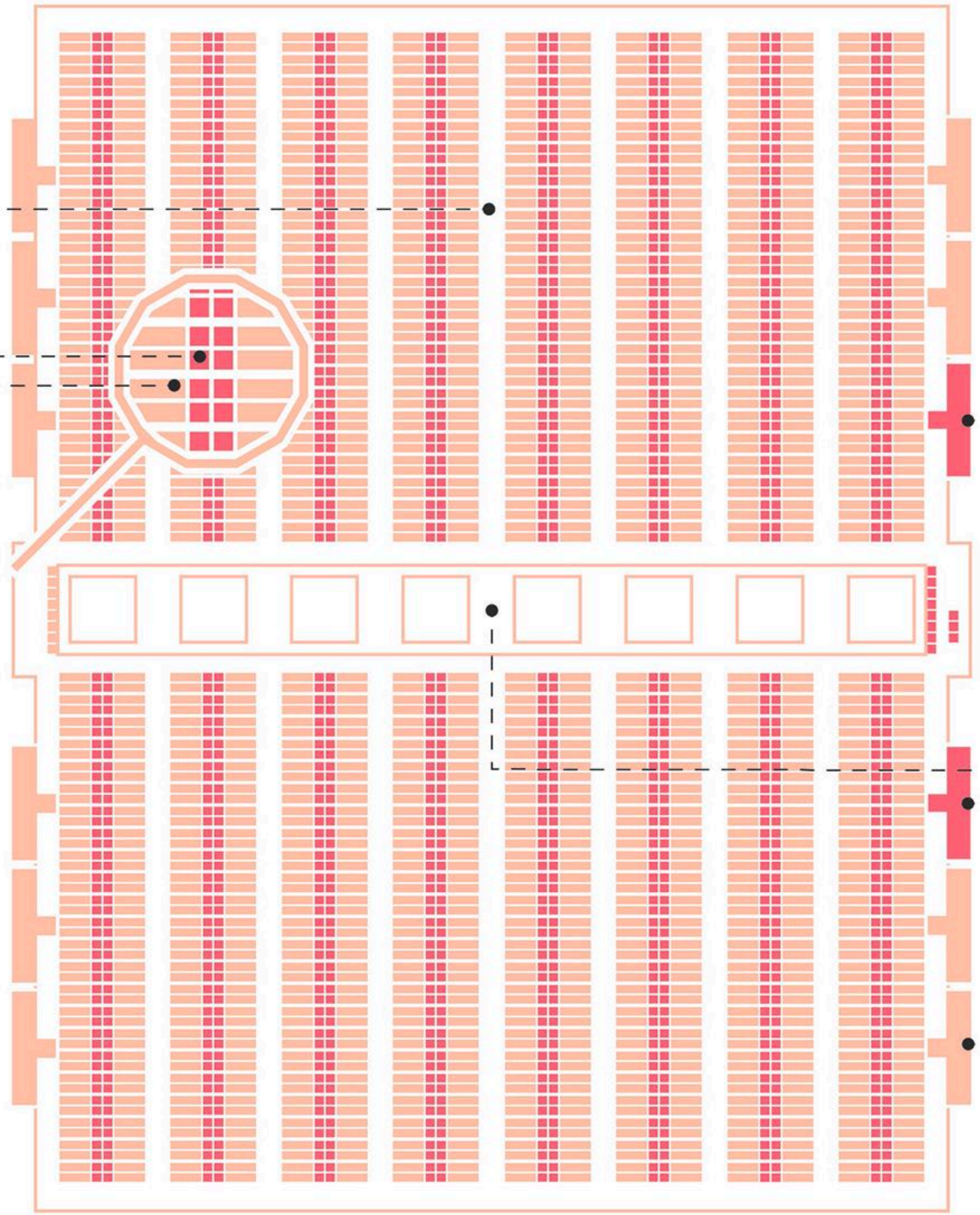
In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

47.5TB/s memory bandwidth per IPU

900 MB on-chip storage

Access to off-chip DDR4



IPU-Exchange™

8 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

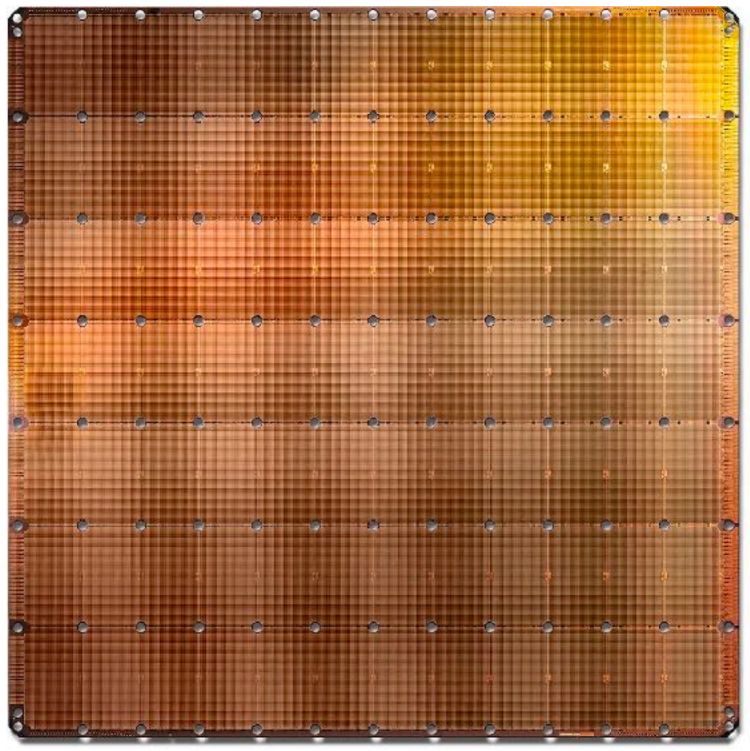
PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

IPU-Links™

10 x IPU-Links,
320GB/s chip to chip bandwidth

(59B transistors similar size to A100 GPU)

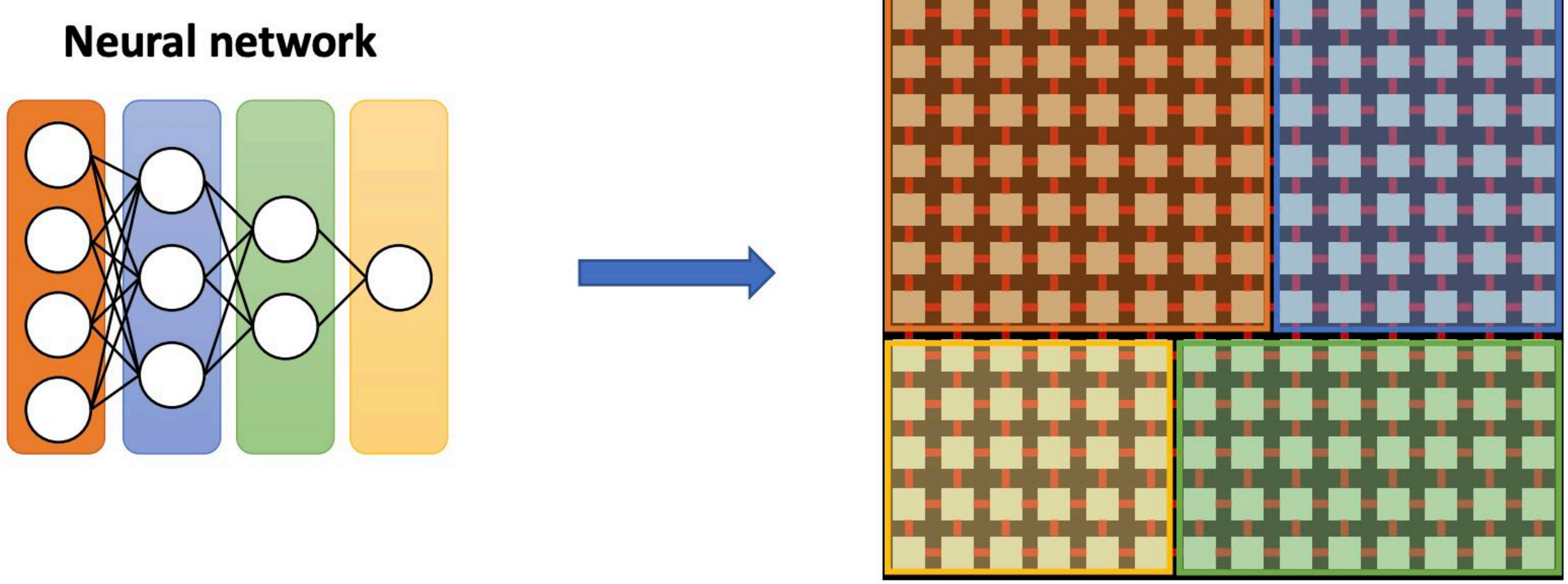
Cerebras Wafer-Scale Engine (WSE)



Tightly interconnected tile of chips (entire wafer)
Many more transistors (1.2T) than largest single chips
(Example: NVIDIA A100 GPU has 54B)

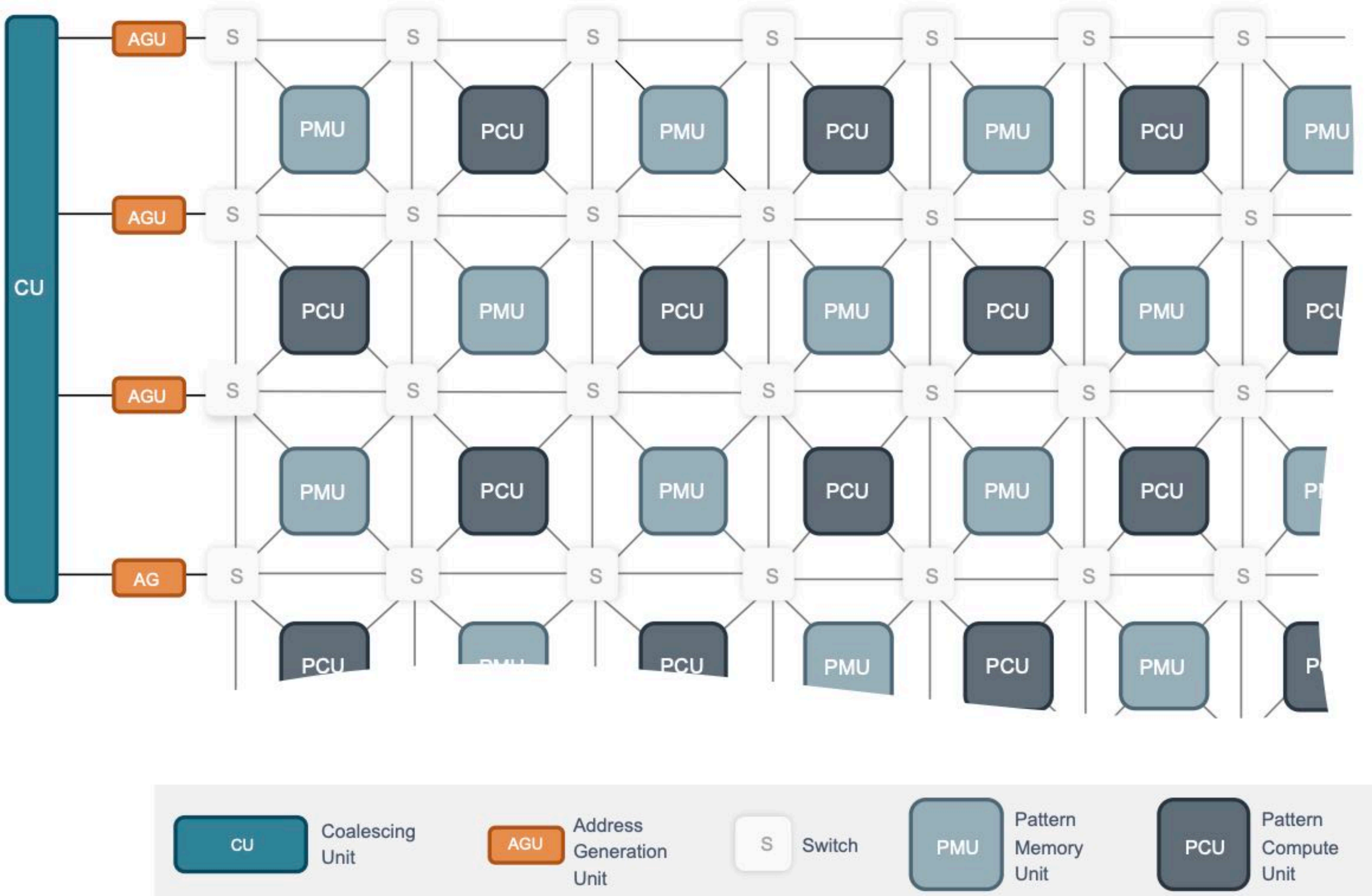
	Cerebras WSE
Chip size	46,225 mm^2
Cores	400,000
On chip memory	$\boxed{\text{18 Gigabytes}}$
Memory bandwidth	9 Petabytes/S
Fabric bandwidth	100 Petabits/S

Compilation of DNN to platform involves "laying out" DNN layers in space on processing grid.

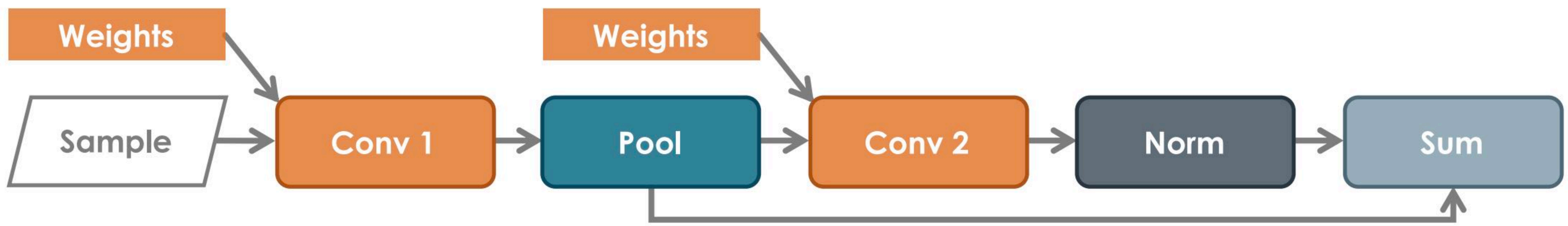


SambaNova reconfigurable dataflow unit

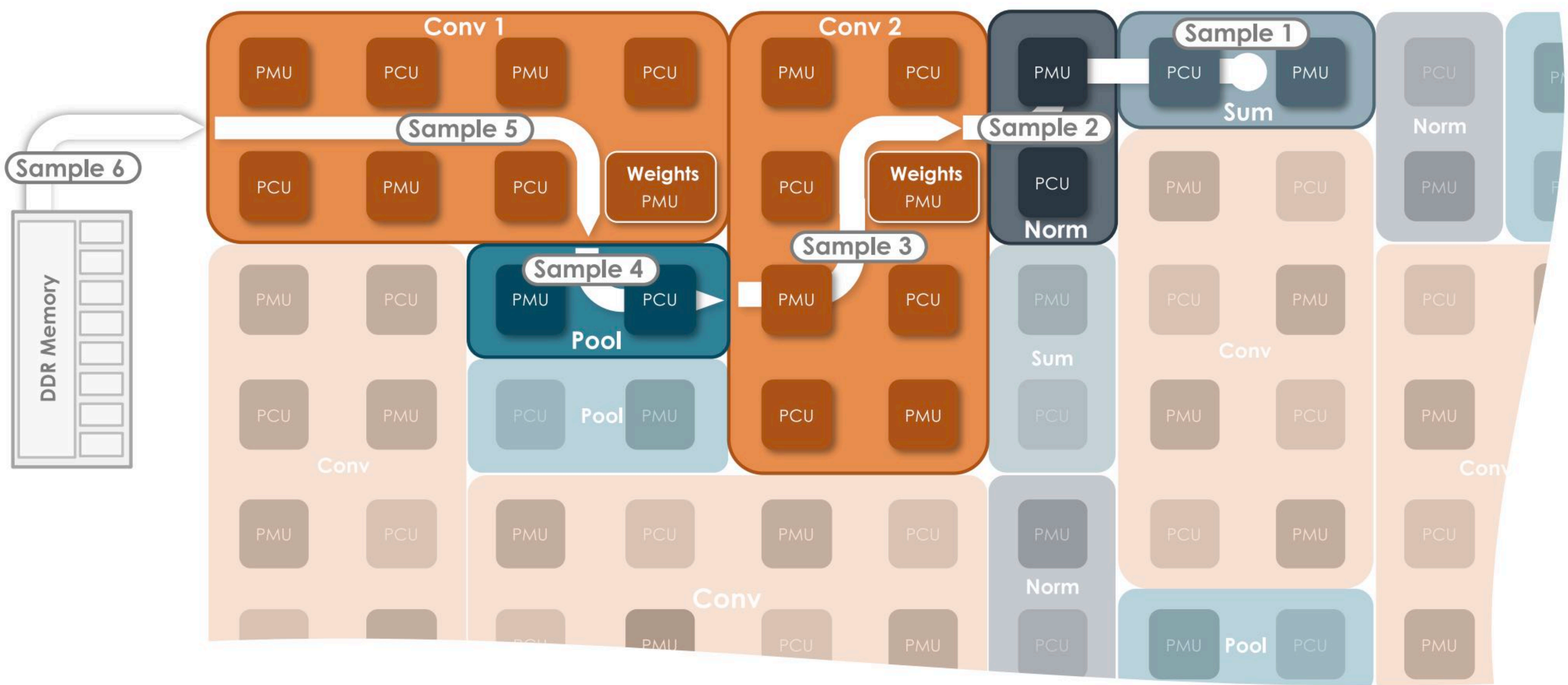
Again, notice tight integration of storage and compute



Another example of spatial layout



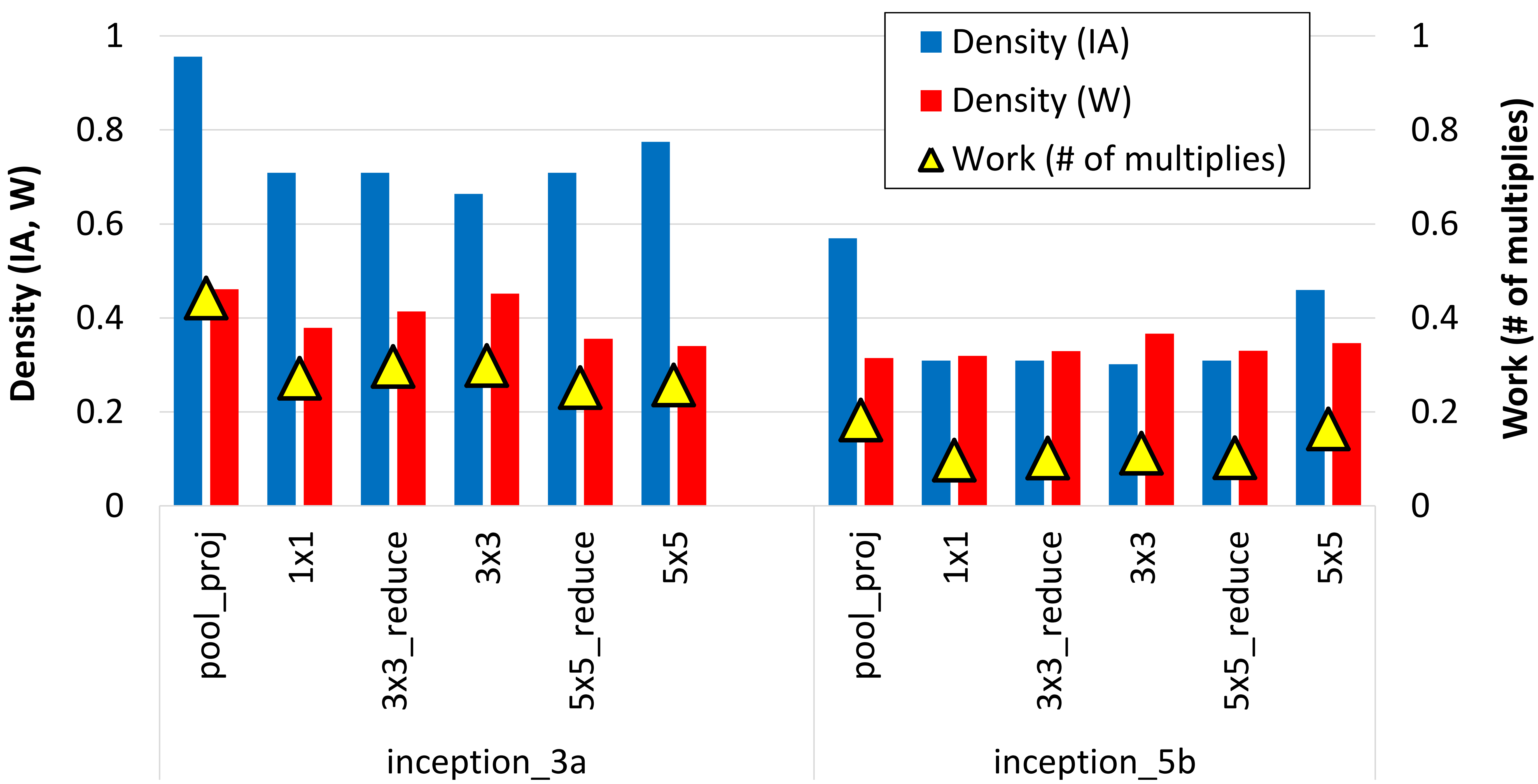
Notice: inter-layer communication occurs through on-chip interconnect, not through off-chip memory.



Exploiting sparsity

Architectural tricks for optimizing for sparsity

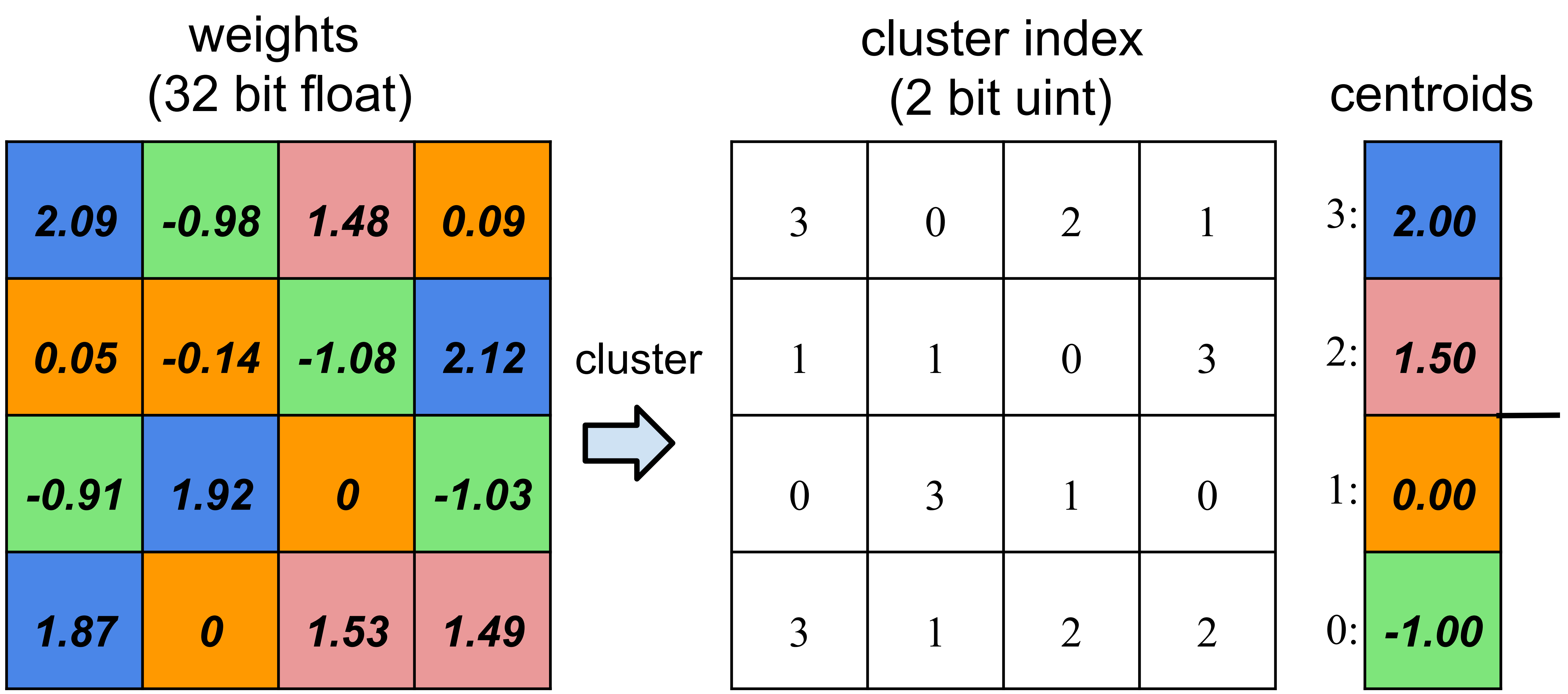
- Consider operation: result += x*y
- If hardware determines contents of register x or register y is zero…
  - Don't fire ALU (save energy)
  - Don't move data from register file to ALU (save energy)
  - But ALU is idle (computation doesn't run faster, optimization only saves energy)



(b) GoogLeNet

Recall: model compression

- Step 1: sparsify weights by truncating weights with small values to zero
- Step 2: compress surviving non-zeros
  - Cluster weights via k-means clustering
  - Compress weights by only storing index of assigned cluster (lg(k) bits)



[Han et al.]

[Figure credit: Han ICLR16]

Sparse, weight-sharing fully-connected layer

$$b_i = ReLU\left(\sum_{j=0}^{n-1} W_{ij} a_j\right)$$

Fully-connected layer:
Matrix-vector multiplication of activation vector a against weight matrix W

$$b_i = ReLU\left(\sum_{j \in X_i \cap Y} S[I_{ij}] a_j\right)$$

Sparse, weight-sharing representation:
I_{ij} = index for weight W_{ij}
S[] = table of shared weight values
X_i = list of non-zero indices in row i
Y = list of non-zero indices in vector a

Note: activations can be sparse due to ReLU

Sparse-matrix, vector multiplication

Represent weight matrix in compressed sparse column (CSC) format to exploit sparsity in activation vector:

```
for each nonzero a_j in a:
    for each nonzero M_ij in column M_j:
      b_i += M_ij * a_j
```

More detailed version (assumes CSC matrix):

```
int16* a_values;    // dense
PTR*   M_j_start;   // column j
int4*  M_j_values;
int4*  M_j_indices;
int16* lookup; // lookup table for
               // cluster values (from
               // deep compression paper)
```

```
for j=0 to length(a):
   if (a[j] == 0) continue;   // scan to next nonzero
   col_values = M_j_values[M_j_start[j]]; // j-th col
   col_indices = M_j_indices[M_j_start[j]]; // row idx in col
   col_nonzeros = M_j_start[j+1] - M_j_start[j];
   for i=0, i_count=0 to col_nonzeros:
      i    += col_indices[i_count];
      b[i] += lookup[col_values[i_count]] * a_values[j];
```

* Recall from deep compression paper: there is a unique lookup table for each chunk of matrix values

Parallelization of sparse-matrix-vector product

Stride rows of matrix across processing elements

Output activations strided across processing elements

$$\vec{a}\ \begin{pmatrix} 0 & 0 & \mathbf{a_2} & 0 & a_4 & a_5 & 0 & a_7 \end{pmatrix}$$

$$\times$$

$$\begin{array}{r} PE0 \\ PE1 \\ PE2 \\ PE3 \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}
\begin{pmatrix}
w_{0,0} & 0 & \boldsymbol{w_{0,2}} & 0 & w_{0,4} & w_{0,5} & w_{0,6} & 0 \\
0 & w_{1,1} & \mathbf{0} & w_{1,3} & 0 & 0 & w_{1,6} & 0 \\
0 & 0 & \boldsymbol{w_{2,2}} & 0 & w_{2,4} & 0 & 0 & w_{2,7} \\
0 & w_{3,1} & \mathbf{0} & 0 & 0 & w_{0,5} & 0 & 0 \\
0 & w_{4,1} & \mathbf{0} & 0 & w_{4,4} & 0 & 0 & 0 \\
0 & 0 & \mathbf{0} & w_{5,4} & 0 & 0 & 0 & w_{5,7} \\
0 & 0 & \mathbf{0} & 0 & w_{6,4} & 0 & w_{6,6} & 0 \\
w_{7,0} & 0 & \mathbf{0} & w_{7,4} & 0 & 0 & w_{7,7} & 0 \\
w_{8,0} & 0 & \mathbf{0} & 0 & 0 & 0 & 0 & w_{8,7} \\
w_{9,0} & 0 & \mathbf{0} & 0 & 0 & 0 & w_{9,6} & w_{9,7} \\
0 & 0 & \mathbf{0} & 0 & w_{10,4} & 0 & 0 & 0 \\
0 & 0 & \boldsymbol{w_{11,2}} & 0 & 0 & 0 & 0 & w_{11,7} \\
w_{12,0} & 0 & \boldsymbol{w_{12,2}} & 0 & 0 & w_{12,5} & 0 & w_{12,7} \\
w_{13,0} & w_{13,2} & \mathbf{0} & 0 & 0 & 0 & w_{13,6} & 0 \\
0 & 0 & \boldsymbol{w_{14,2}} & w_{14,3} & w_{14,4} & w_{14,5} & 0 & 0 \\
0 & 0 & \boldsymbol{w_{15,2}} & w_{15,3} & 0 & w_{15,5} & 0 & 0
\end{pmatrix}
=
\vec{b}\begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \\ -b_8 \\ -b_9 \\ b_{10} \\ -b_{11} \\ -b_{12} \\ b_{13} \\ b_{14} \\ -b_{15} \end{pmatrix}
\overset{ReLU}{\Rightarrow}
\begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \\ 0 \\ 0 \\ b_{10} \\ 0 \\ 0 \\ b_{13} \\ b_{14} \\ 0 \end{pmatrix}$$

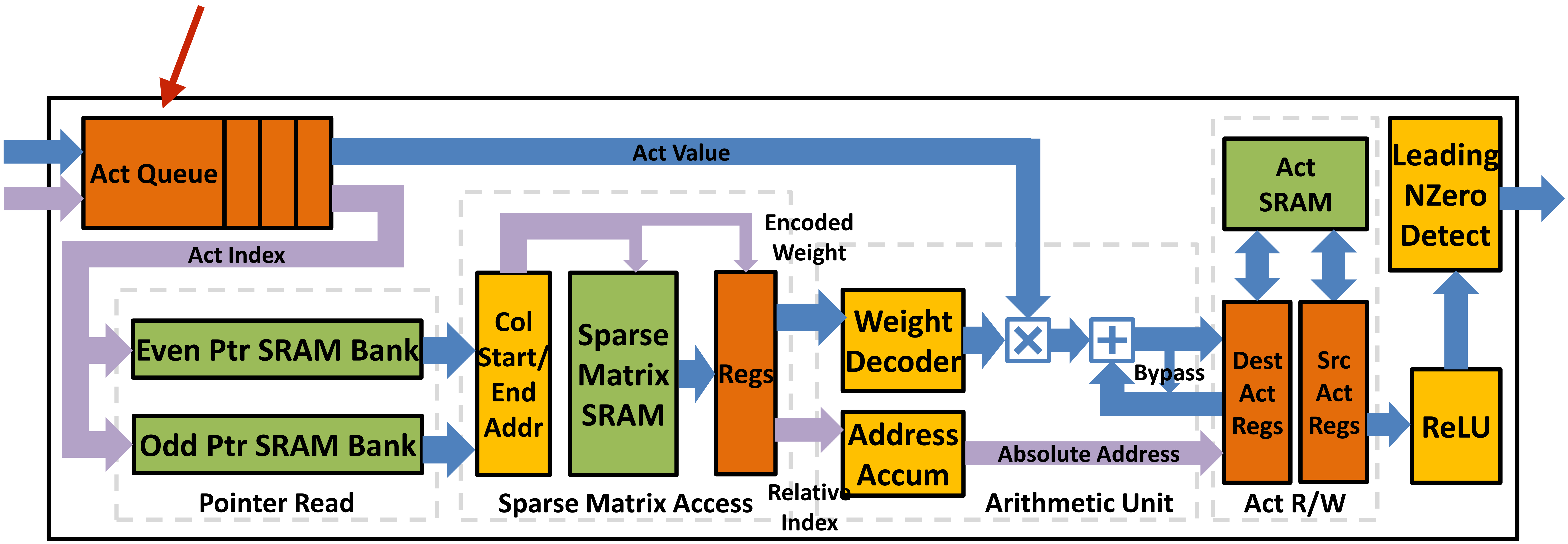
Weights stored local to PEs. Must broadcast non-zero a_j's to all PEs

Accumulation of each output b_i is local to PE

Efficient Inference Engine (EIE) for quantized sparse/matrix vector product

Custom hardware for decoding compressed-sparse representation

Tuple representing non-zero activation (a_j, j) arrives and is enqueued



EIE efficiency

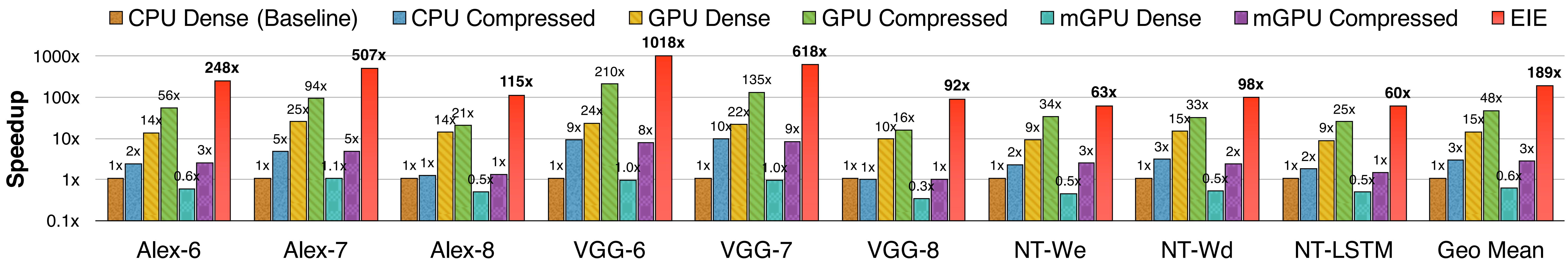
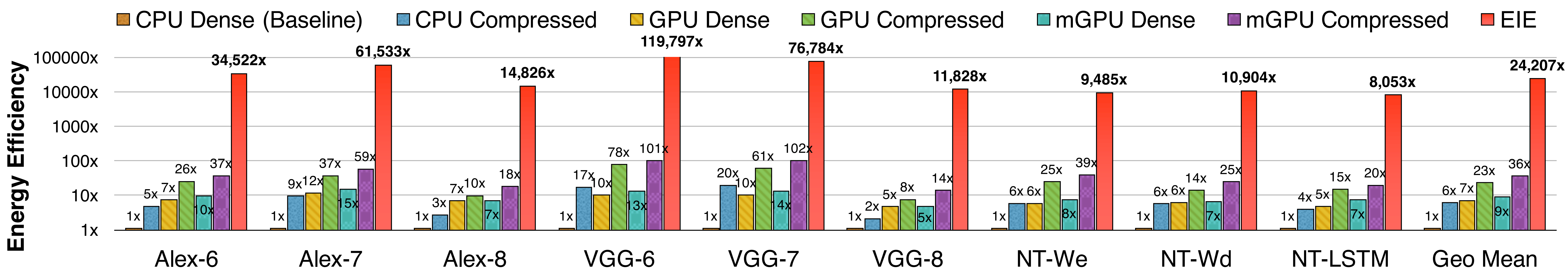


Figure 6. Speedups of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model. There is no batching in all cases.



CPU: Core i7 5930k (6 cores)
GPU: GTX Titan X
mGPU: Tegra K1

Warning: these are not end-to-end numbers: just fully connected layers!

Sources of energy savings:

- Compression allows all weights to be stored in SRAM (reduce DRAM loads)
- Low-precision 16-bit fixed-point math (5x more efficient than 32-bit fixed math)
- Skip math on input activations that are zero (65% less math)

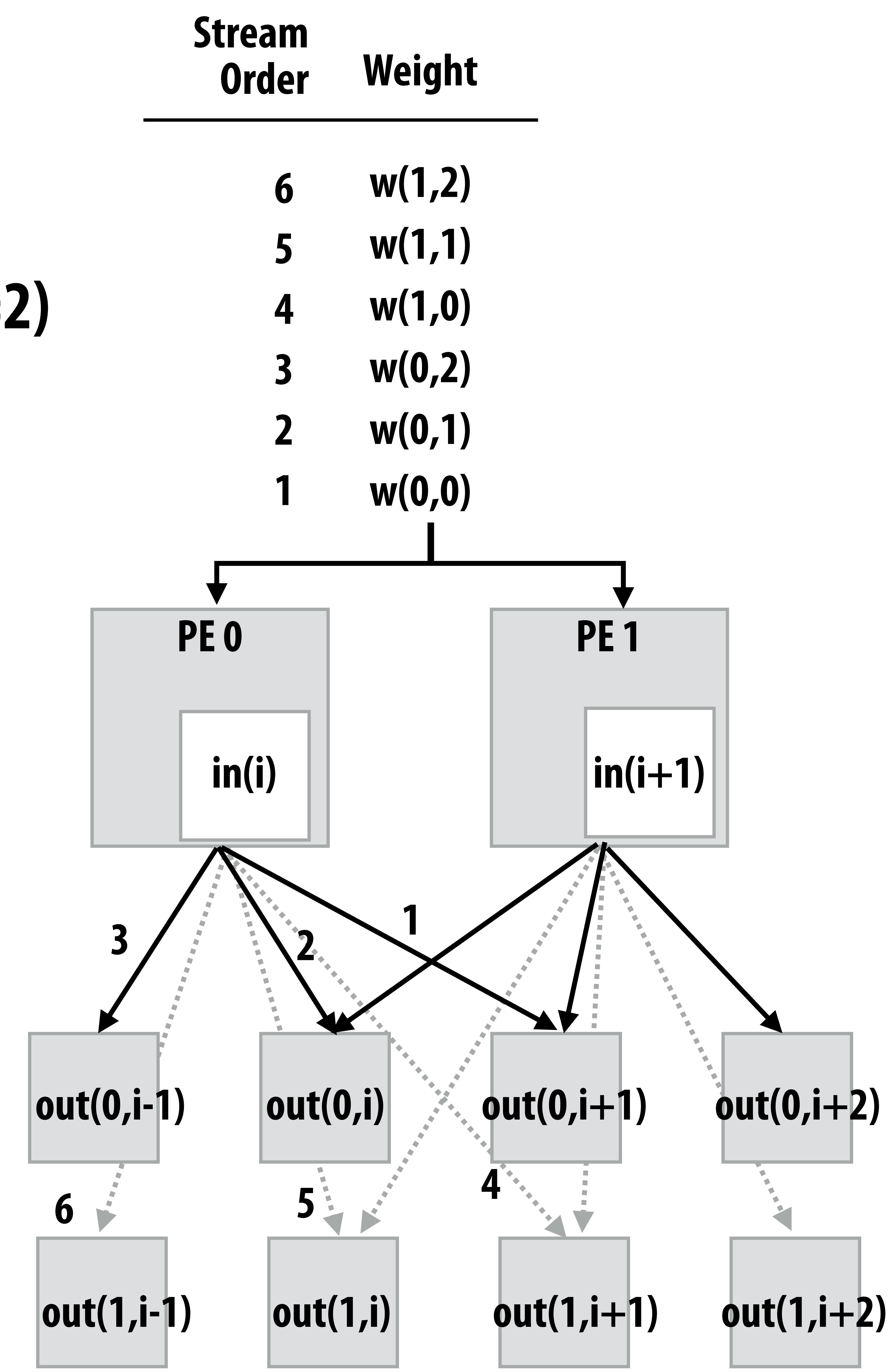
Reminder: input stationary design (dense 1D)

Assume:

1D input/output

3-wide filters

2 output channels (K=2)



Stream of weights (2 1D filters of size 3)

Processing elements (implement multiply)

Accumulators (implement +=)

Input stationary design (sparse example)

Assume:

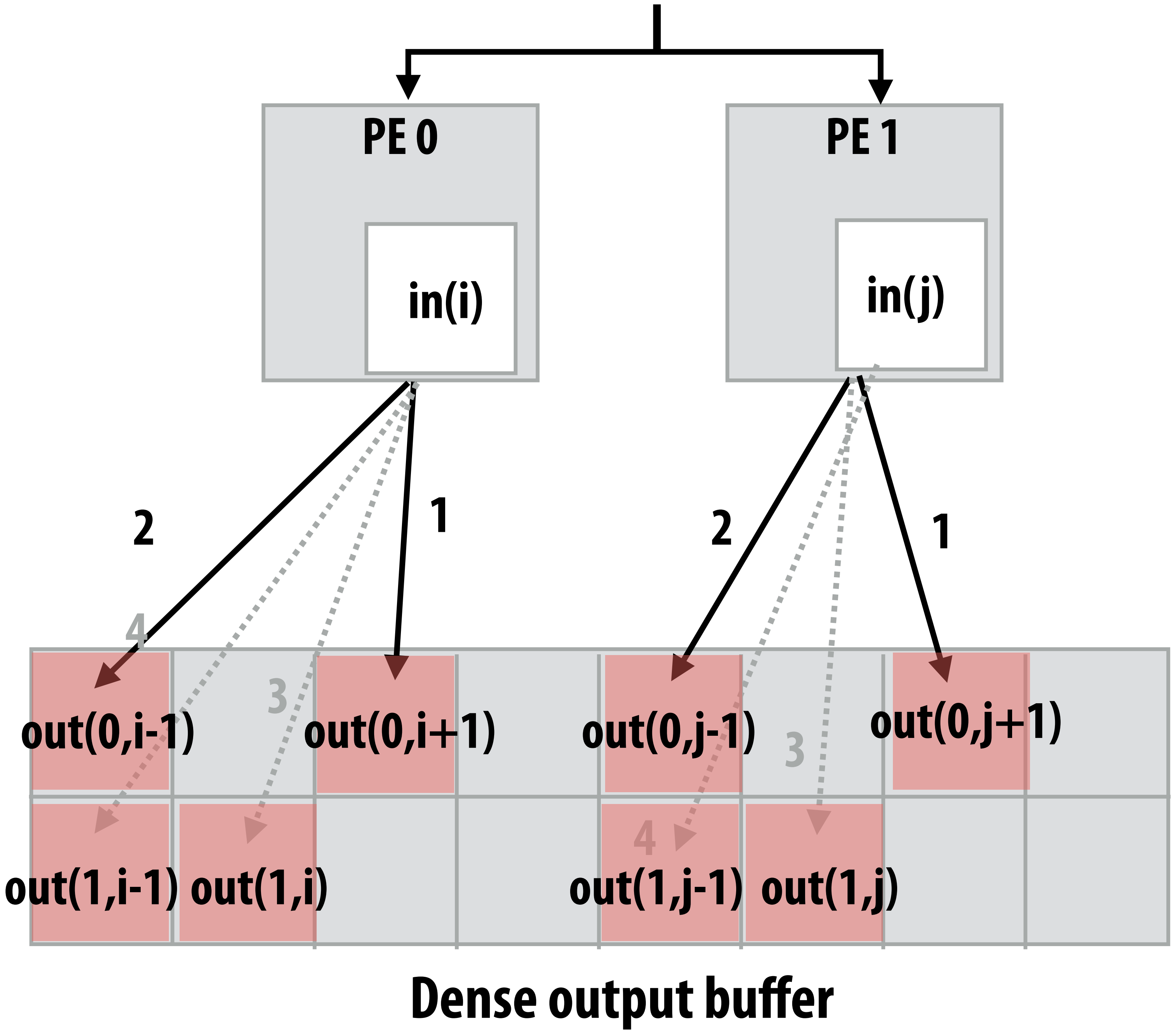
1D input/output

3-wide SPARSE filters

2 output channels (K=2)

Stream Order	Weight
4	w(1,2)
3	w(1,1)
2	w(0,2)
1	w(0,0)

Stream of sparse weights (2 filters, each with 2 non-zeros)



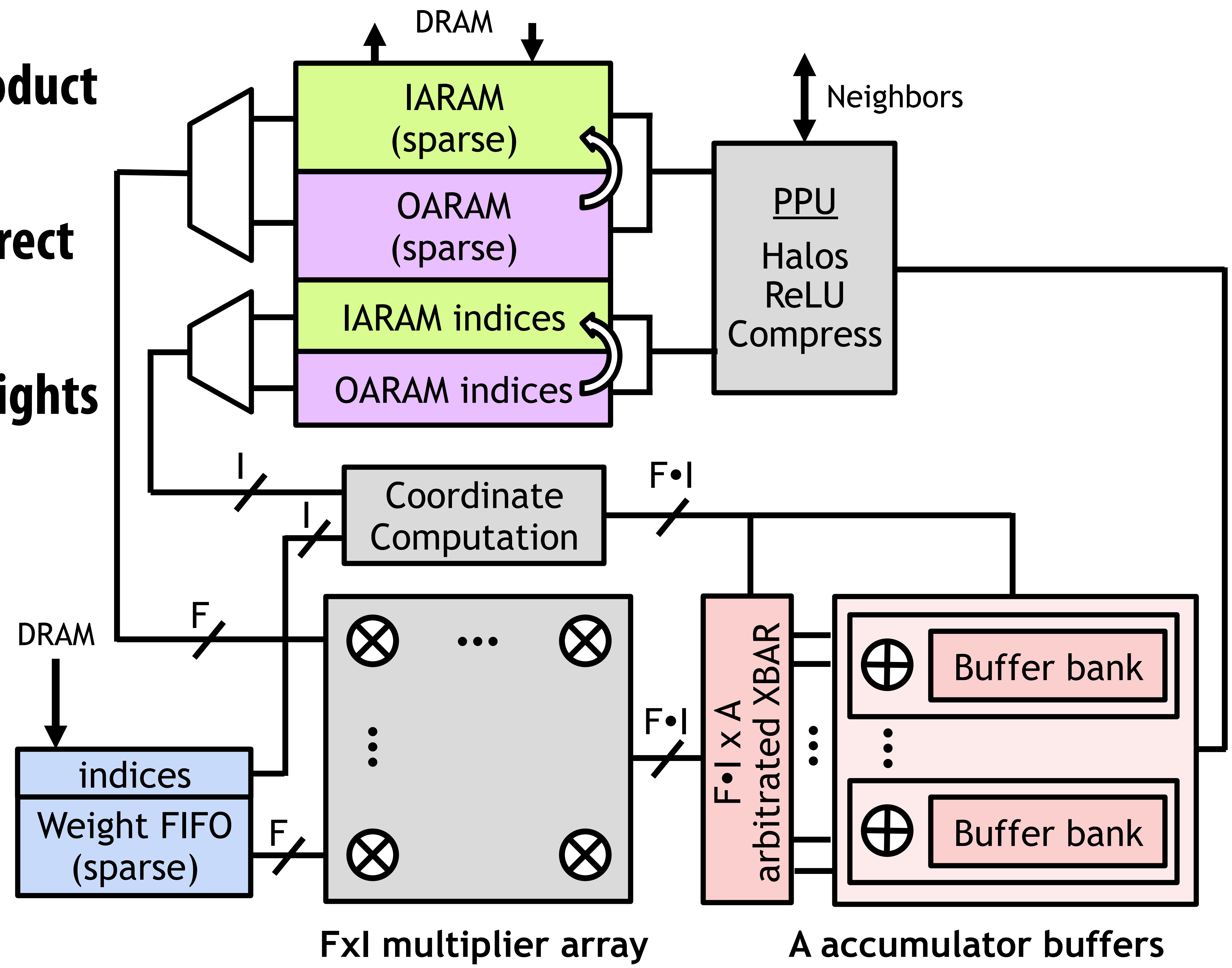
Processing elements

Accumulators (implement +=)

Note: accumulate is now a scatter

SCNN: accelerating sparse conv layers

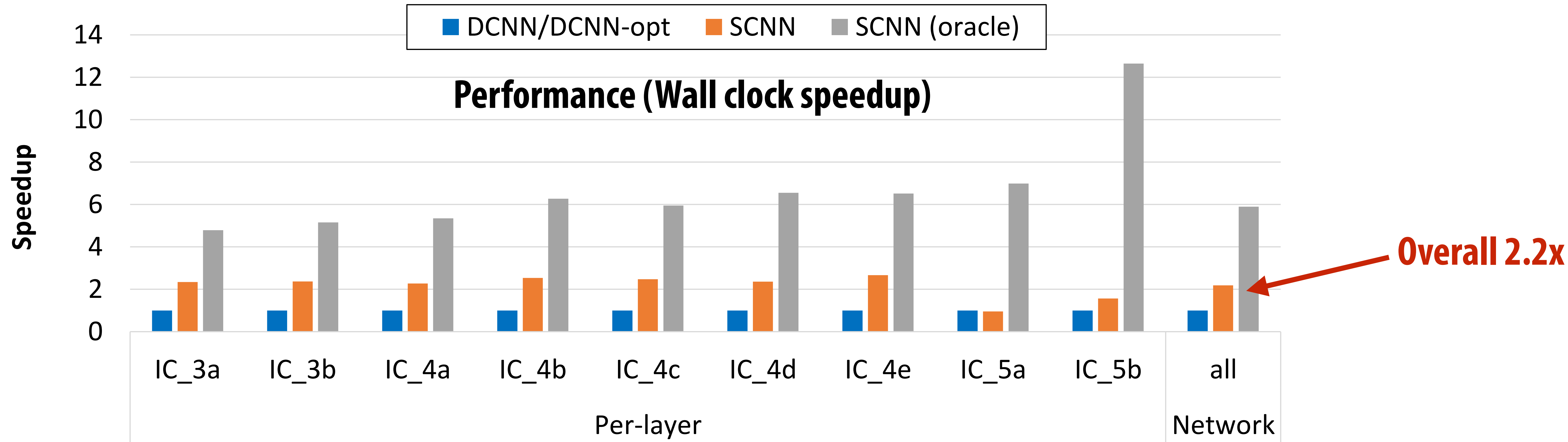
- Like EIE: assume both activations and conv weights are sparse
- Weight stationary design:
  - Each PE receives:
    - A set of I input activations from an input channel: a list of I (value, (x,y)) pairs
    - A list of F non-zero weights
  - Each PE computes: the cross-product of these values: P x I values
  - Then scatters P x I results to correct accumulator buffer cell
  - Then repeat for new set of F weights (reuse I inputs)
- Then, after convolution:
  - ReLU sparsifies output
  - Compress outputs into sparse representation for use as input to next layer

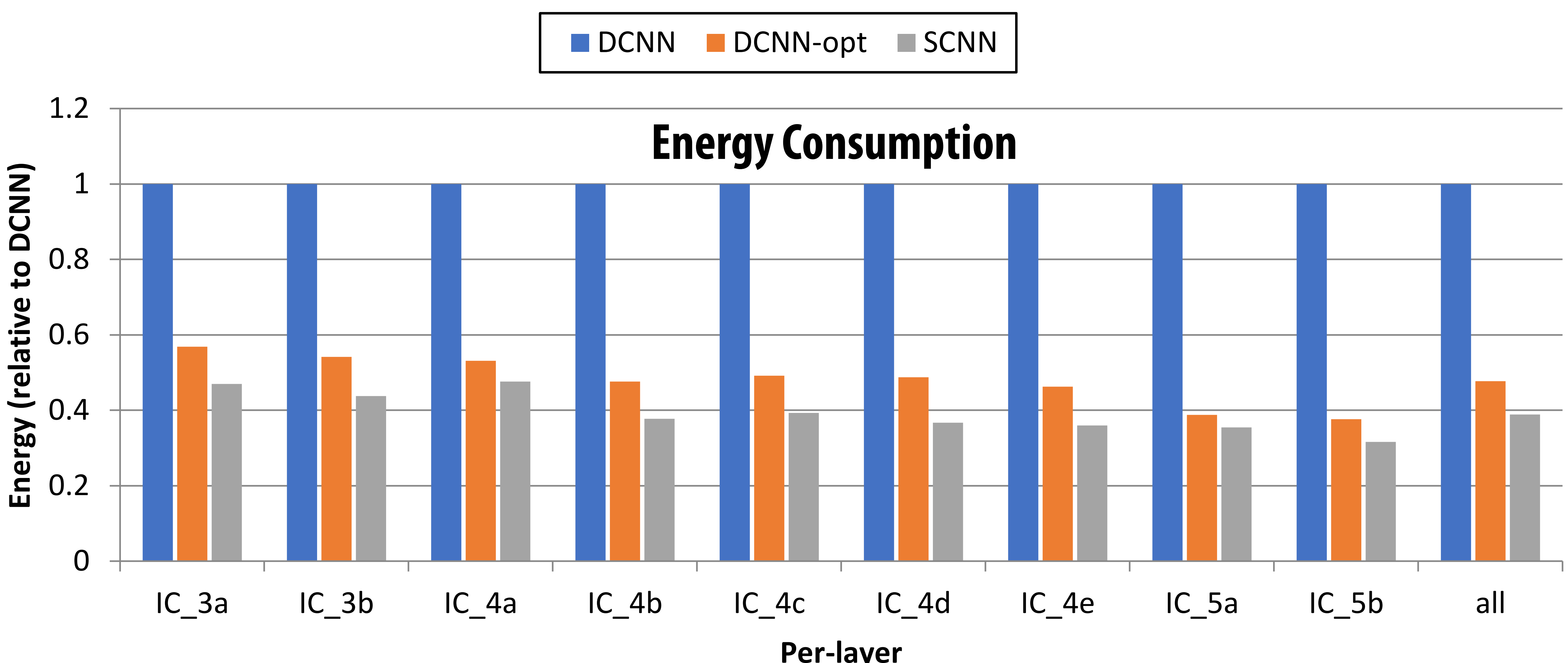


[Parashar et al. ISCA17]

[Parashar et al. ISCA17]

SCNN results (on GoogLeNet)





DCNN = dense CNN evaluation

DCNN-opt = includes ALU gating, and compression/decompression of activations

Summary of hardware accelerators for efficient inference

- Specialized instructions for dense linear algebra computations
  - Reduce overhead of control (compared to CPUs/GPUs)
- Reduced precision operations (cheaper computation + reduce bandwidth requirements)
- Systolic / dataflow architectures for efficient on-chip communication
  - Different scheduling strategies: weight-stationary, input/output stationary, etc.
- Huge amounts of on-chip memory to avoid off-chip communication
- Exploit sparsity in activations and weights
  - Skip computation involving zeros
  - Hardware to accelerates decompression of sparse representations like compressed sparse row/column