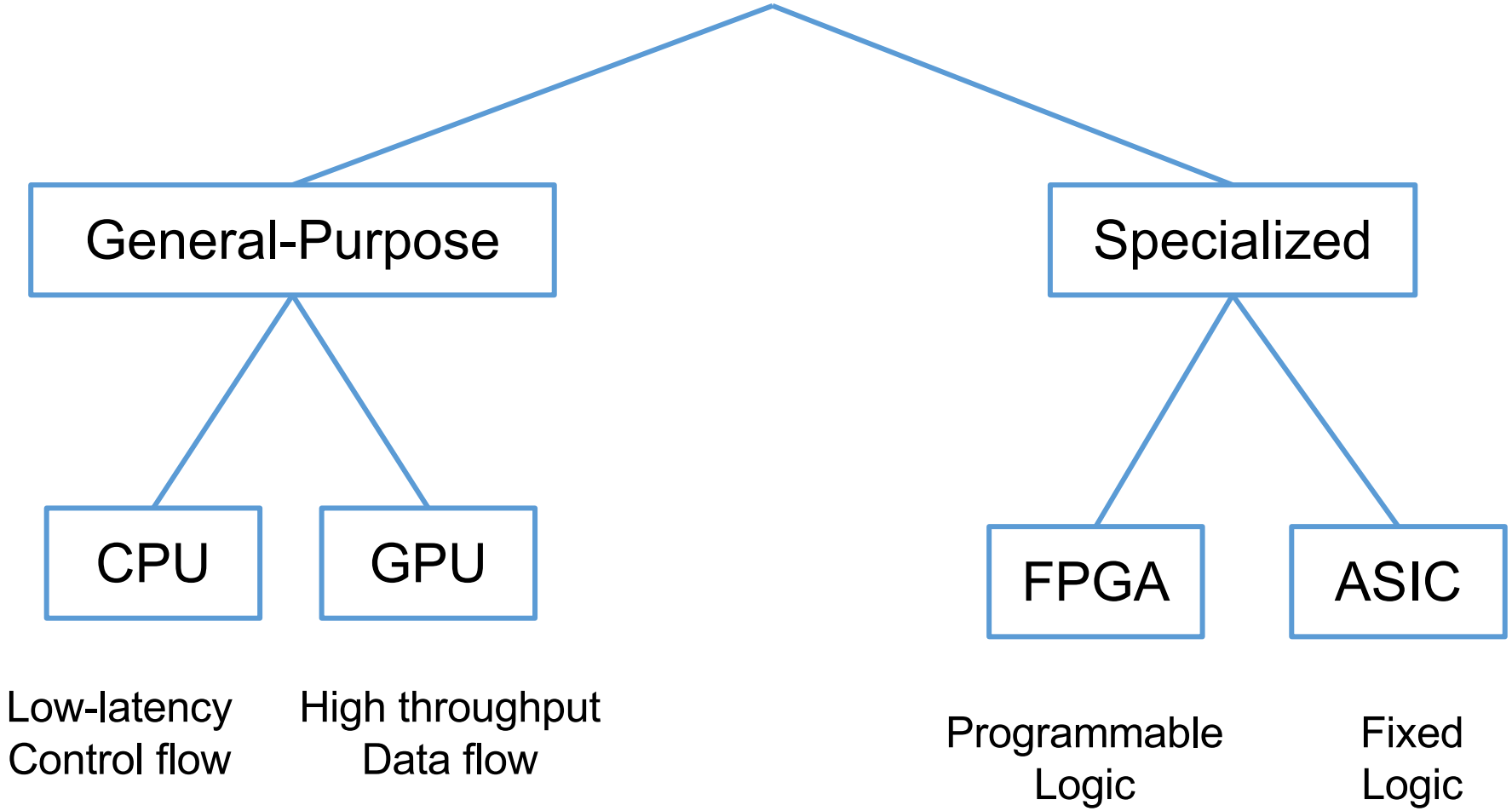
SFU

CMPT 450/750: Computer Architecture
Fall 2023
Domain-Specific Architecture I
How did we get here ?
What are they ?

Alaa Alameldeen & Arrvindh Shriraman

SFU

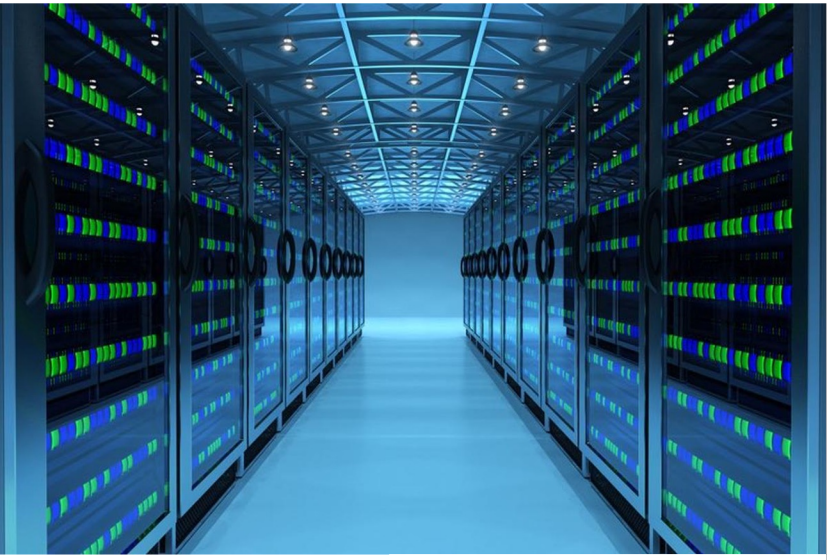
Hardware Types

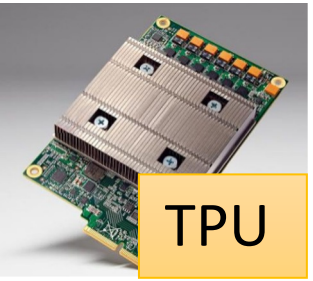


Adapted from Song Han "Efficient Methods and Hardware for Deep Learning" Stanford 2017

Specialized Hardware

SFU





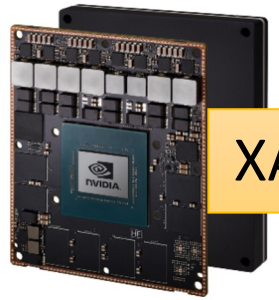
TPU

FPGA

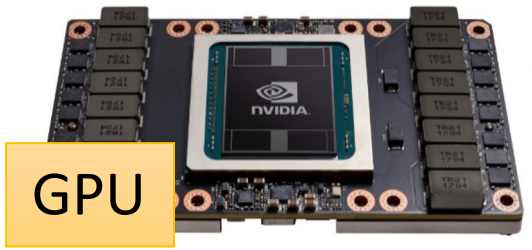
A12 BIONIC

NPU

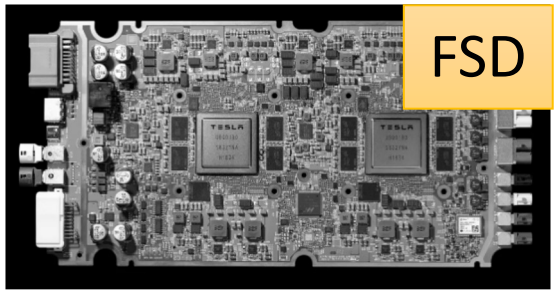
855+ Qualcomm snapdragon



XAVIER



GPU



FSD

SFU

Learning Objectives

By the end of this lecture, you should be able to:

1. Calculate important performance metrics for hardware

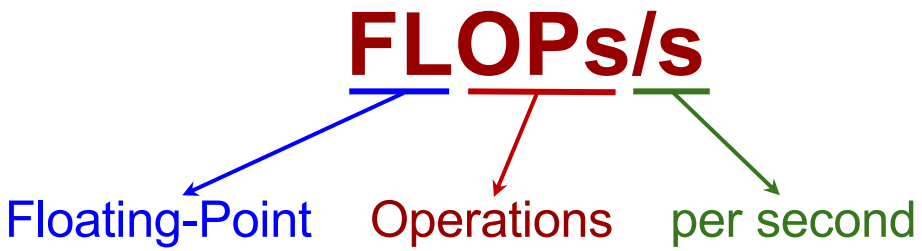
2. Optimize the compute and memory efficiency of hardware

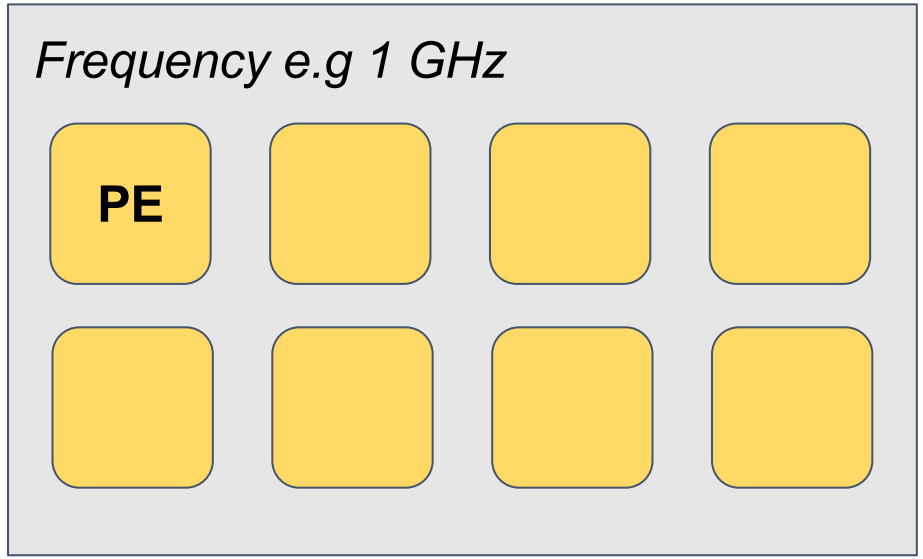
3. Analyze emerging hardware architectures

SFU

Hardware Metrics & Roofline

Compute Performance Metrics



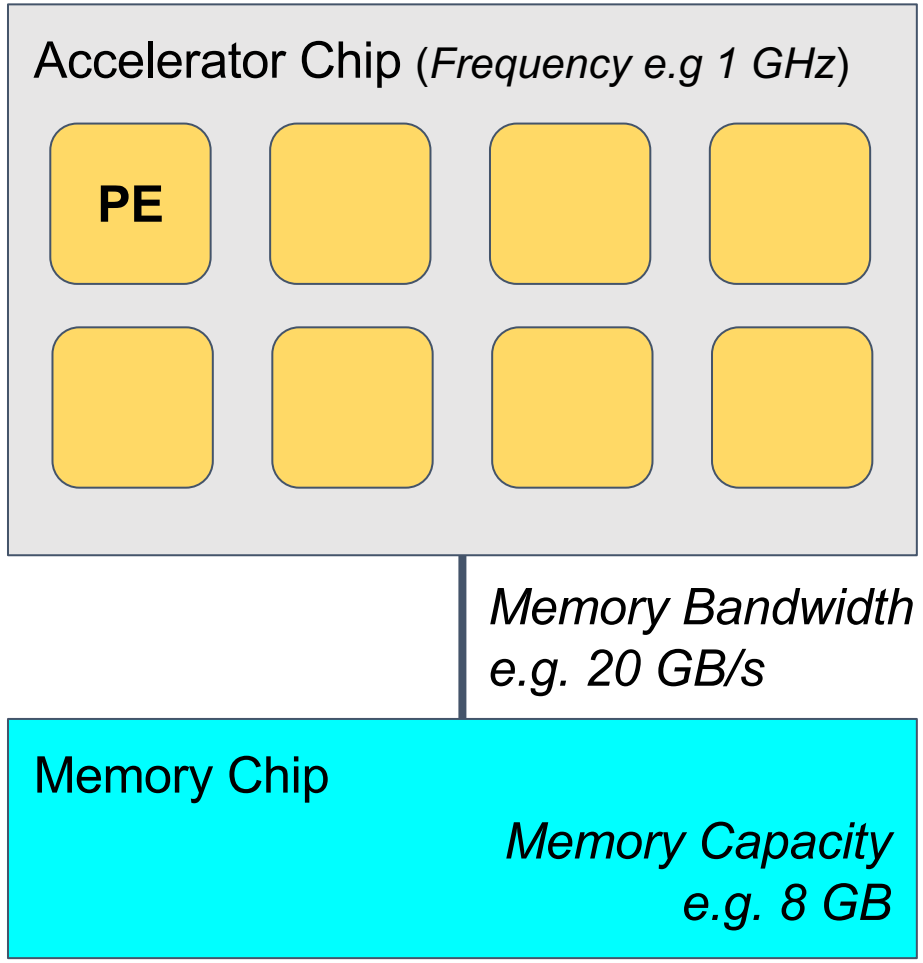


- MACs/s: Multiply-accumulate Ops/s
  - Half FLOPs/s
- OPs/s: for non floating-point operations
- Chips are often labeled with “peak FLOPs/s”
  - Not achievable under normal workloads
  - Very rough indication of performance

$$\frac{\text{operations}}{\text{second}} = \underbrace{\left(\frac{1}{\frac{\text{cycles}}{\text{operation}}} \times \frac{\text{cycles}}{\text{second}}\right)}_{\text{for a single PE}} \times \text{number of PEs}$$

Memory Performance Metrics

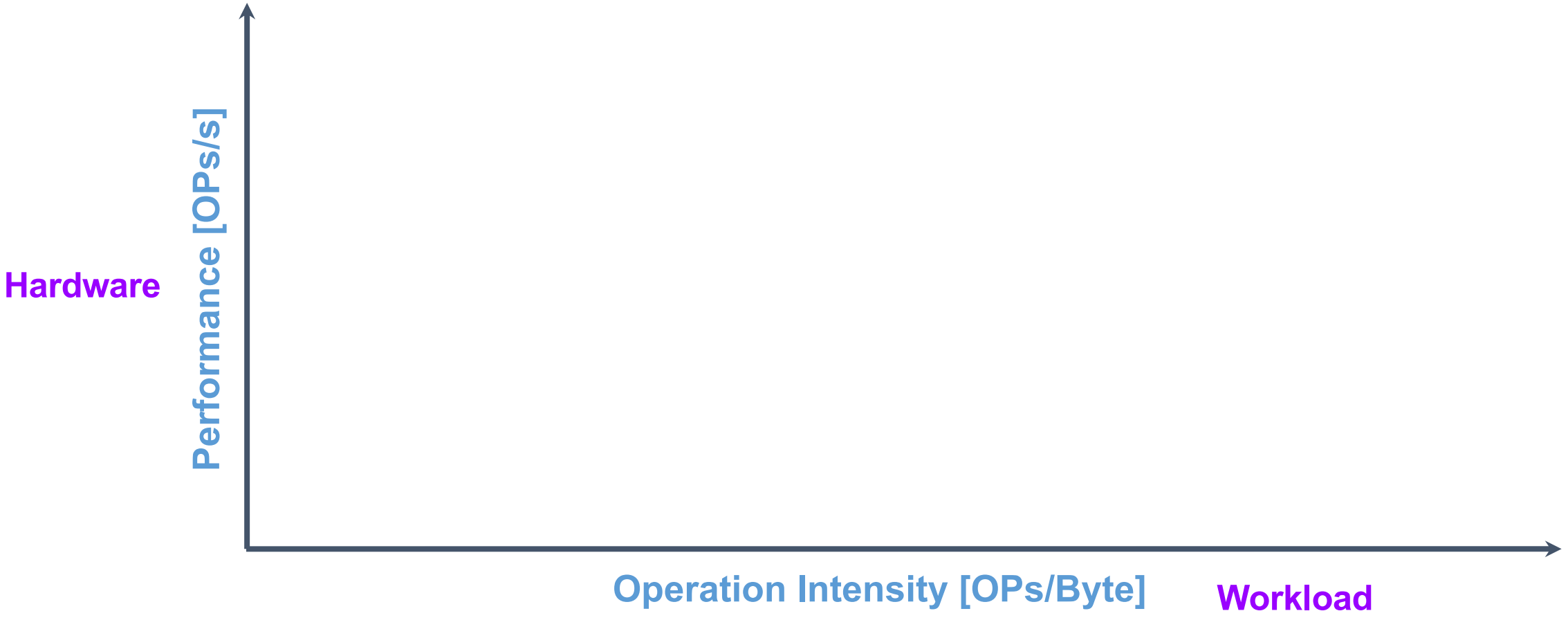
- Memory capacity [GB]
- Memory bandwidth [GB/s]
  - Transfer speed from memory chip to compute chip
- More complicated because there is a *memory hierarchy*
  - Showing “external”/”main” memory
  - Can have caches, local memory, registers with much higher bandwidth



SFU

Roofline Plot

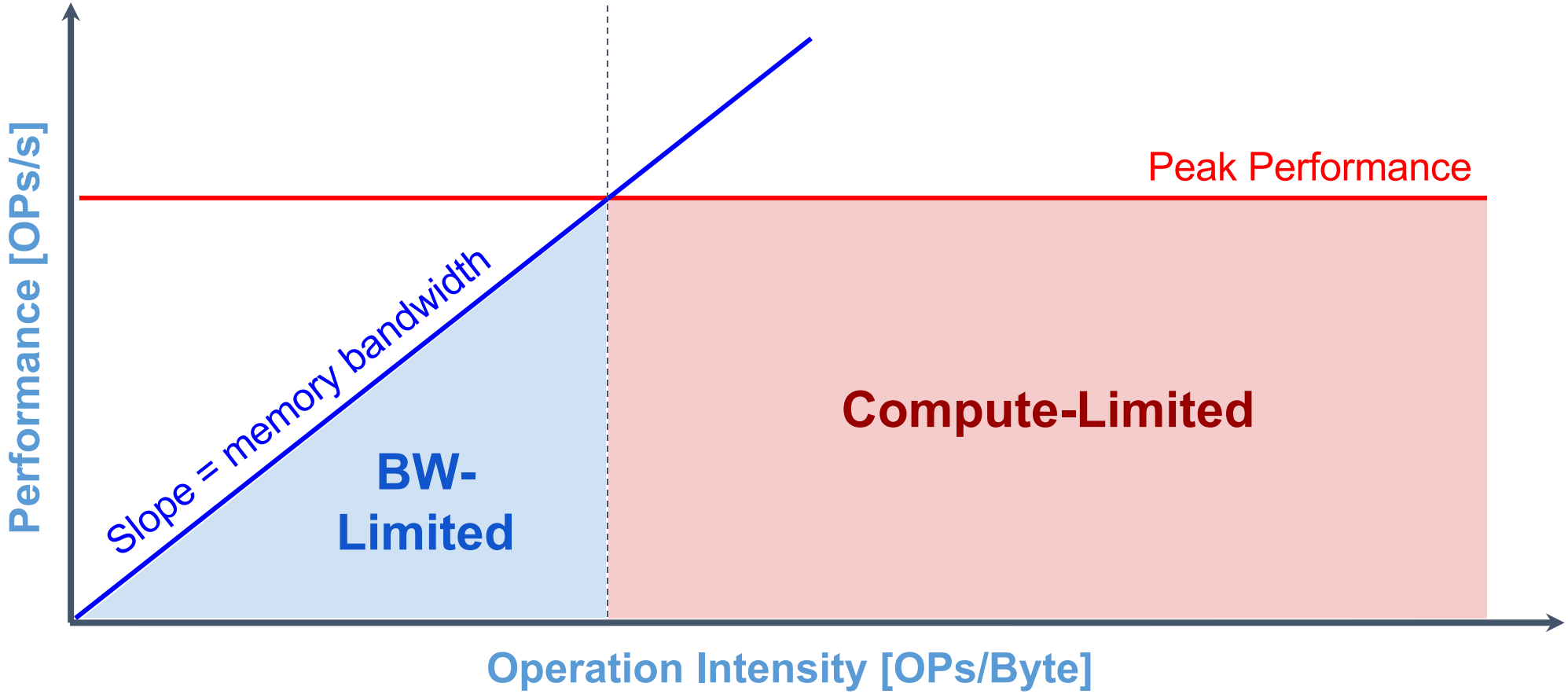
Characterize the performance of a given hardware device across different workloads



SFU

Roofline Plot

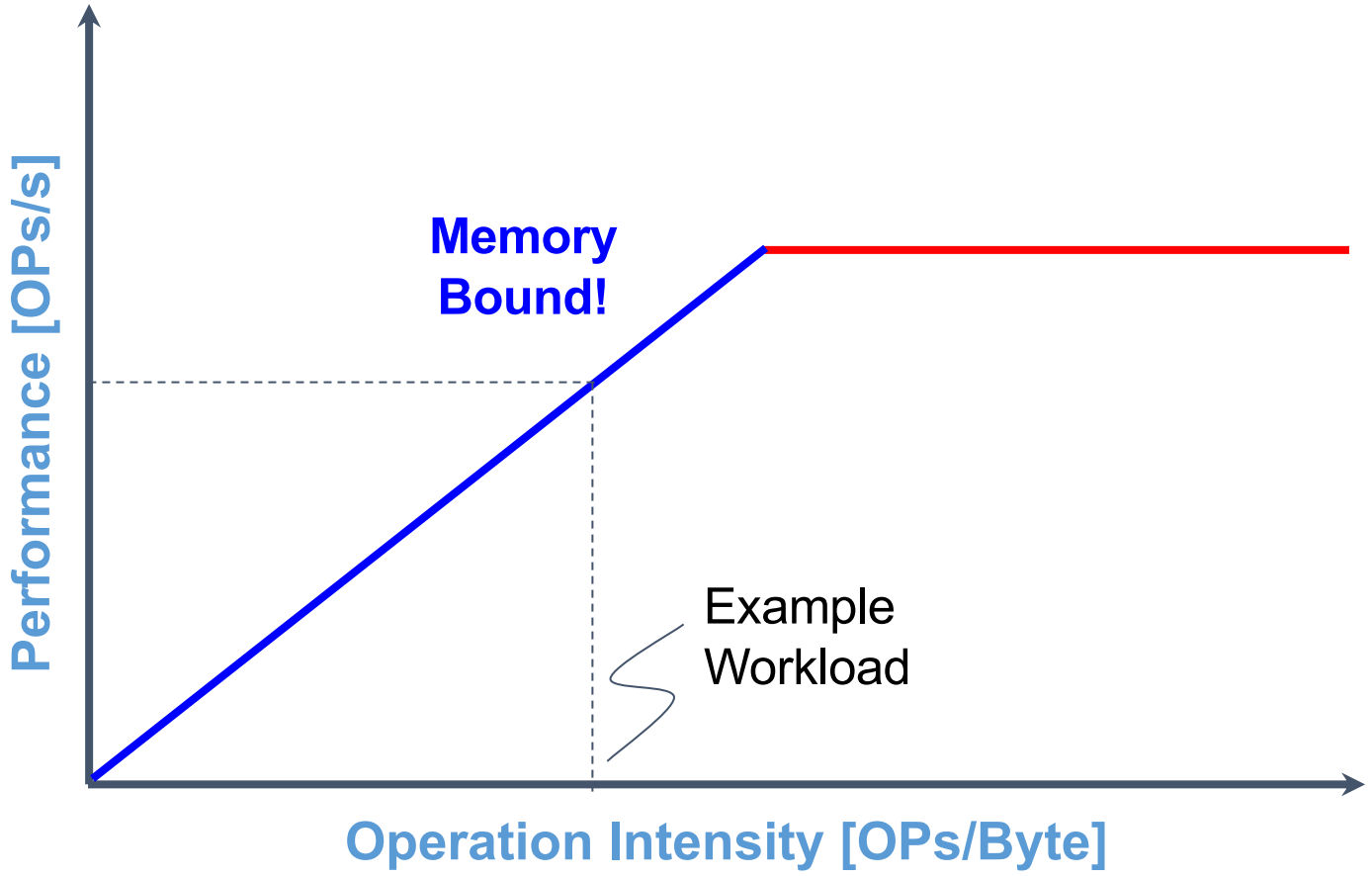
Characterize the performance of a given hardware device across different workloads

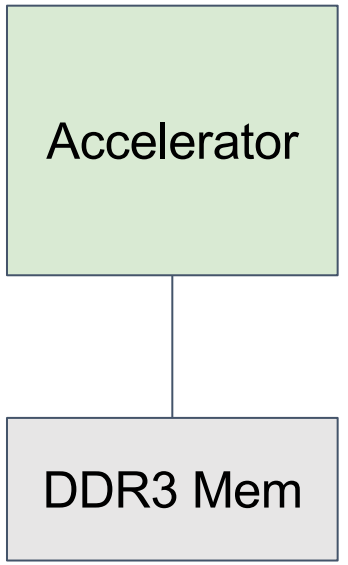


SFU

Roofline Plot

Characterize the performance of a given hardware device across different workloads





SFU

What is OPs/Byte of a DNN?

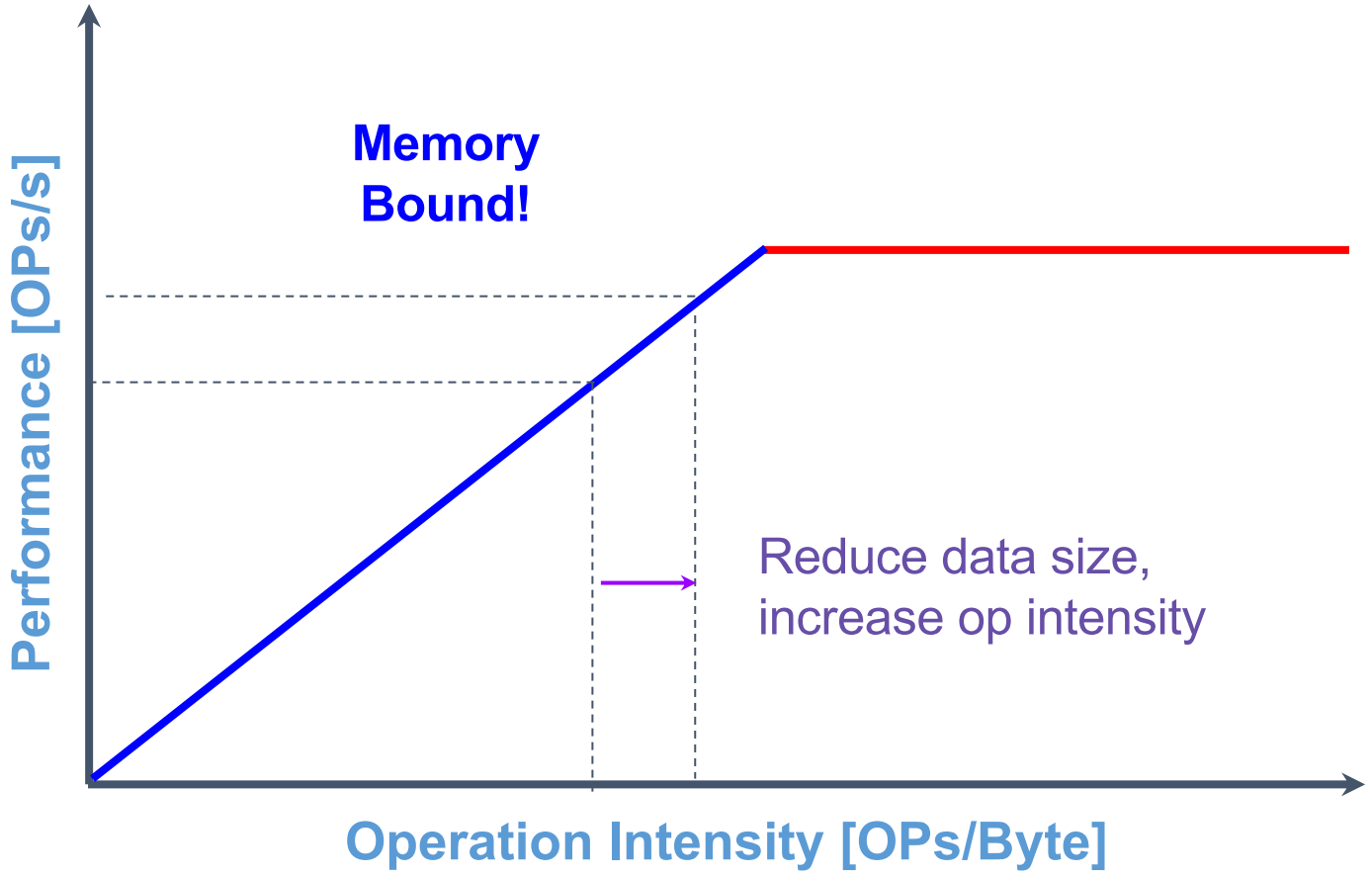
- Operational intensity [OPs/Byte] quantifies the ratio of computations to memory footprint of a DNN
- Total number of operations = multiplications + additions
- Total memory footprint = size of parameters + size of activations

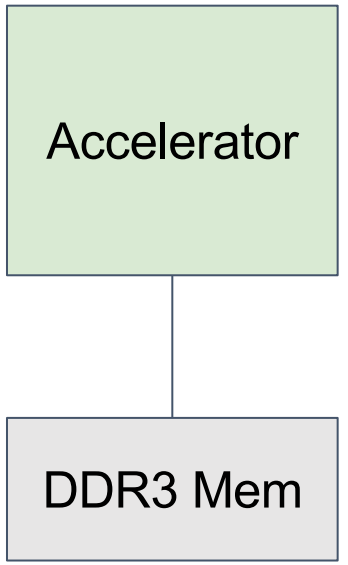
$$\text{Operational Intensity} = \frac{\text{Total number of operations}}{\text{Total memory footprint}}$$

SFU

Roofline Plot

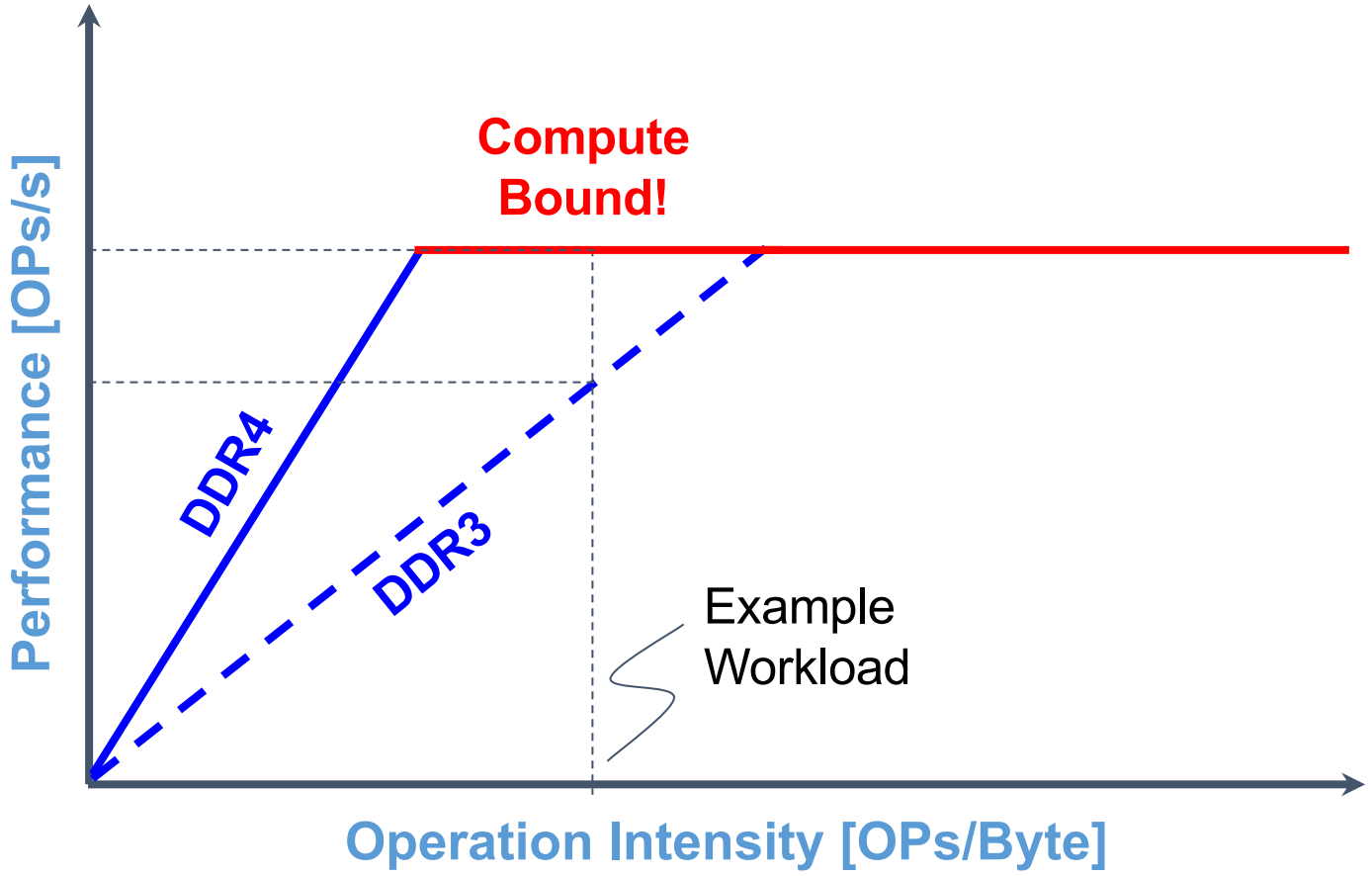
Compressed data format e.g. reduced precision

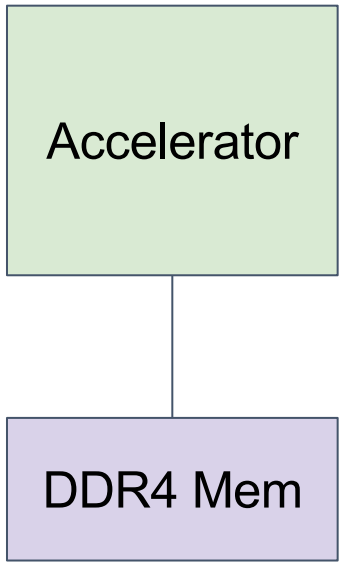




Roofline Plot

Faster memory chip increases slope of roofline

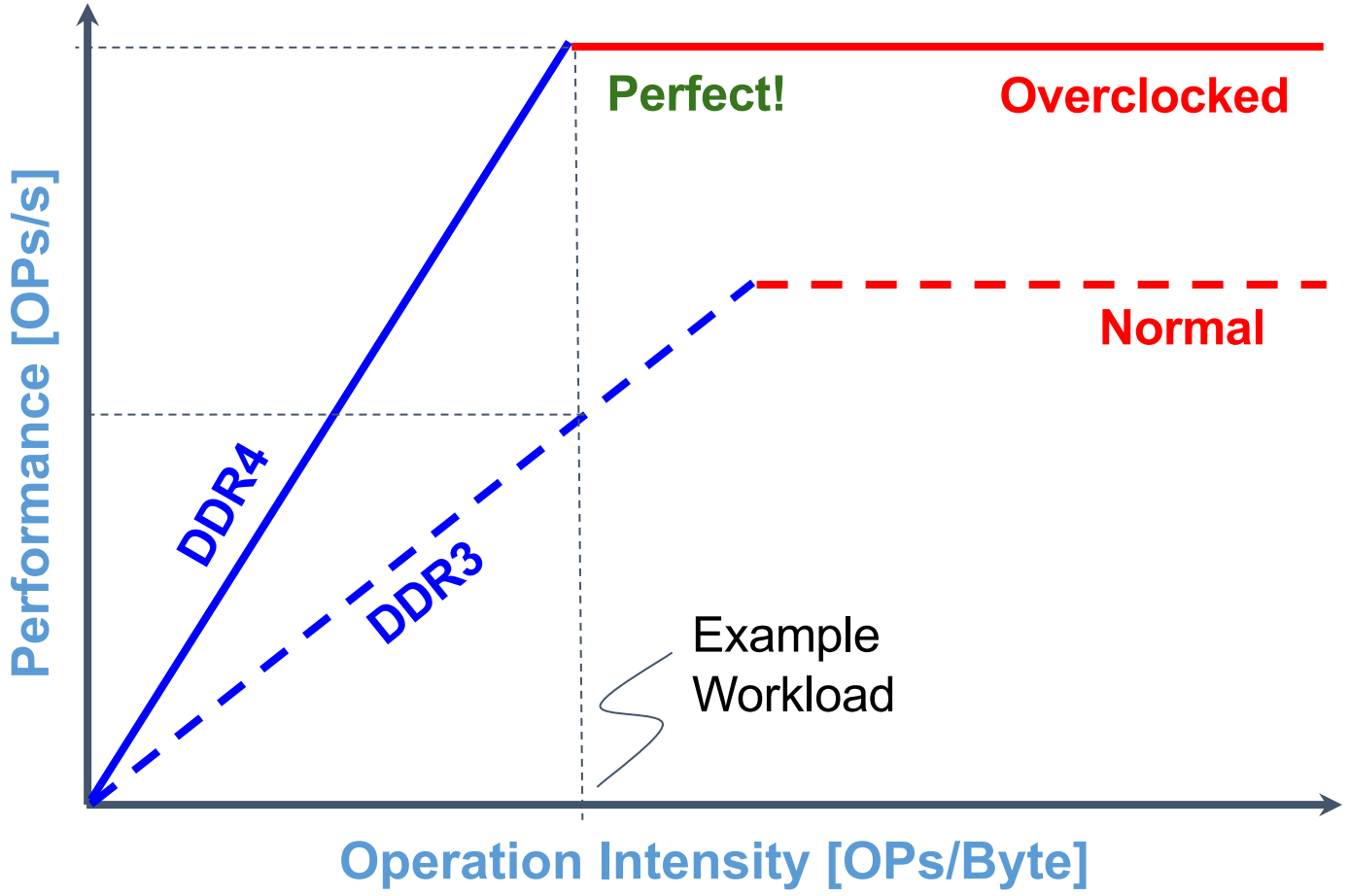


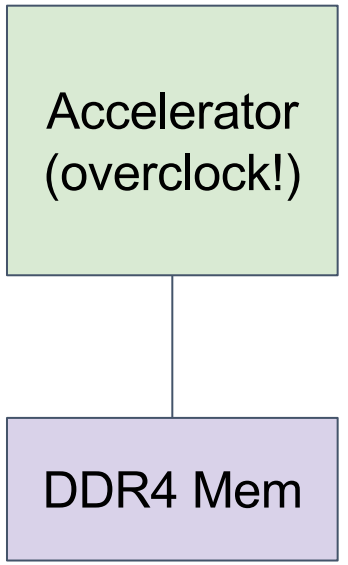


SFU

Roofline Plot

Raise the roofline by increasing the speed/throughput of compute

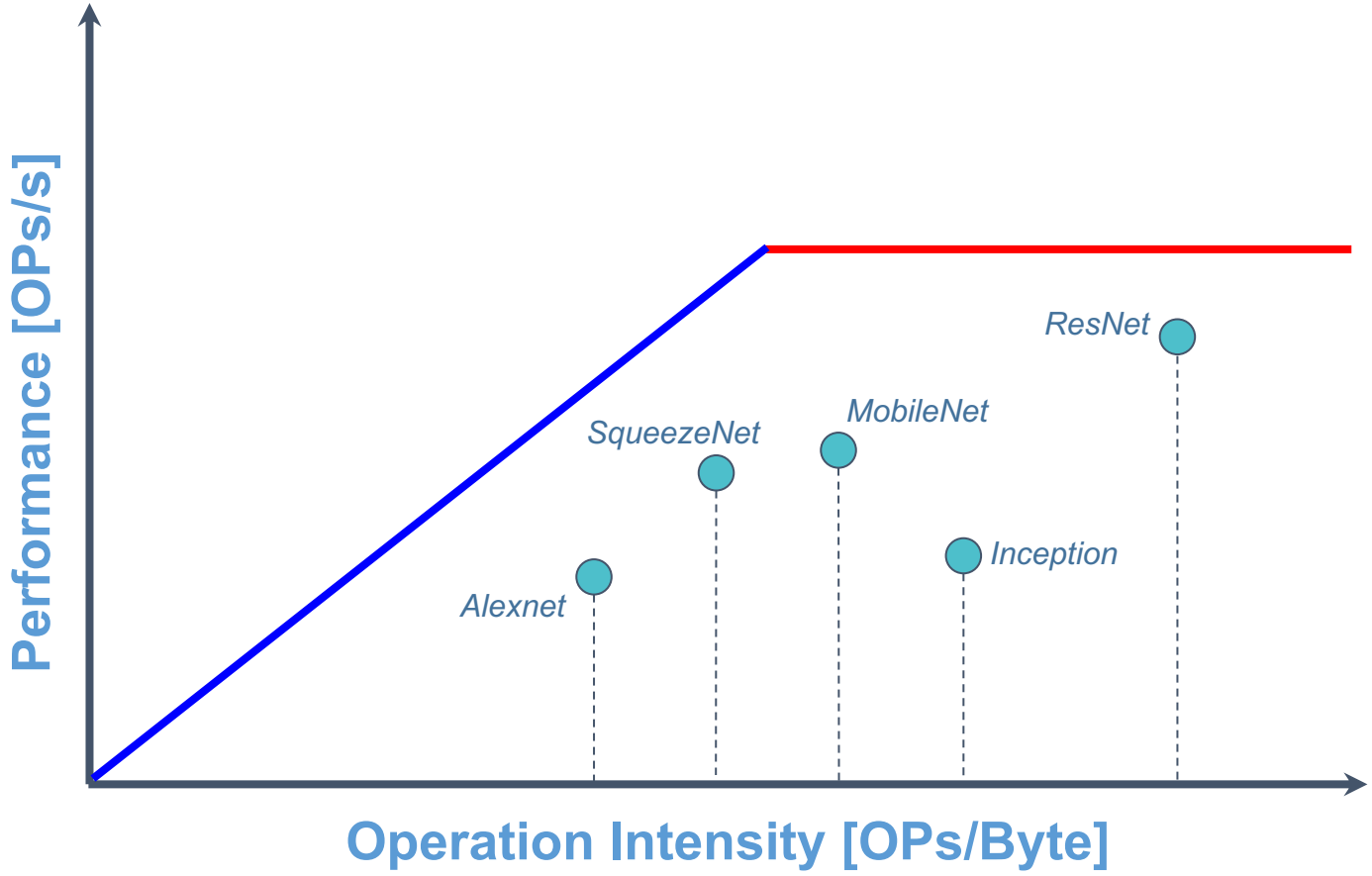




SFU

Roofline Example

Measured performance is (by definition) below the roofline.

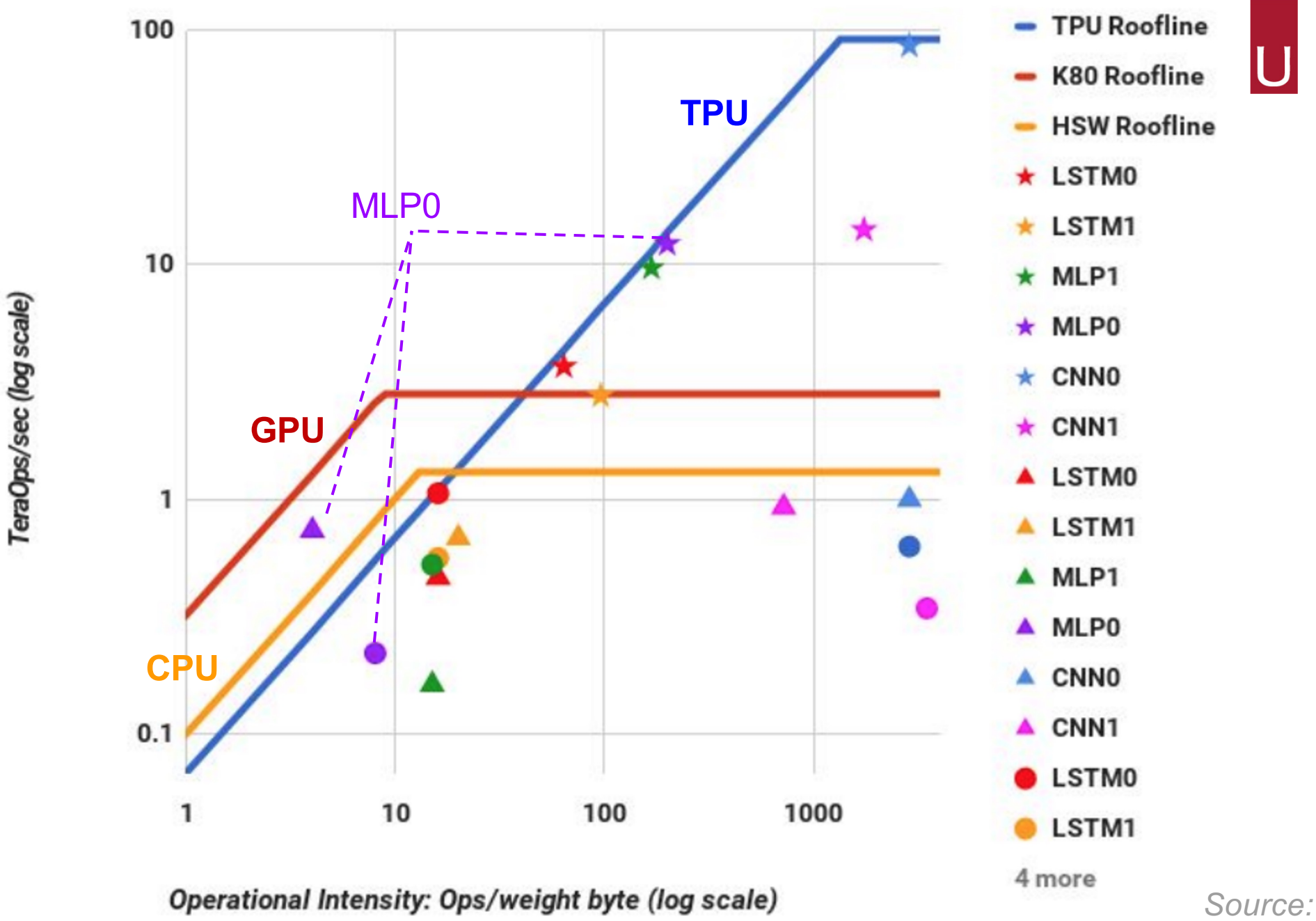


Achieved Performance can be limited by:

- Memory access efficiency
  - E.g.: uncoalesced reads - most DRAM chips require successive reads, each of a specific width to use maximum bandwidth.
- Compute utilization
  - E.g.: In DNN, MAC array hardcoded to 16 channels per tile but first layer has 3 channels
  - Overhead of control logic
- Complexity
  - Control flow and data hazards may stall execution even if the hardware is available

note: points are not plotted in their correct place and are just for illustrative purposes

Google TPU Roofline



Source: Google

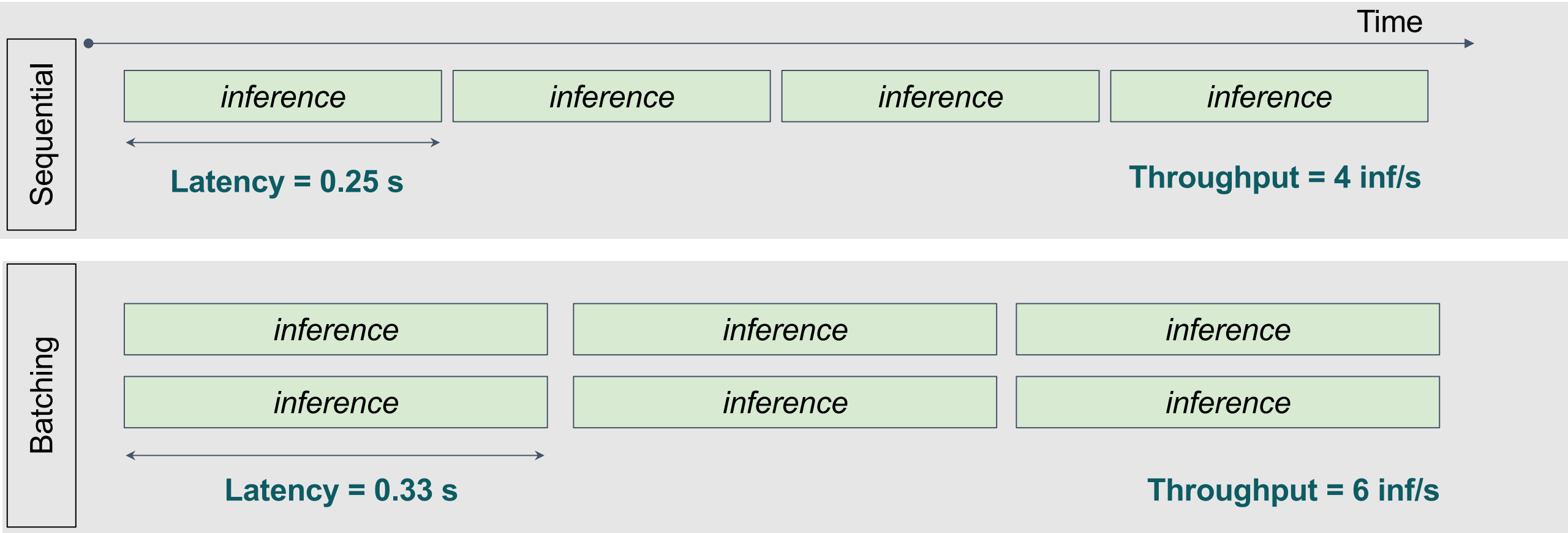
Metrics Summary (so far)

Metric	Hardware	DNN
Peak Performance [OPs/s]	●	
Memory Bandwidth [GB/s]	●	
Operational Intensity [OP/B]		●
HW Utilization	●	●
Throughput [OPs/s]	●	●
Latency [seconds]	●	●

Adapted from: Sze et al.

Throughput and Latency

- **Latency**: Number of seconds per inference (unit = seconds)
- **Throughput**: Number of inferences per second (unit = inference/second)



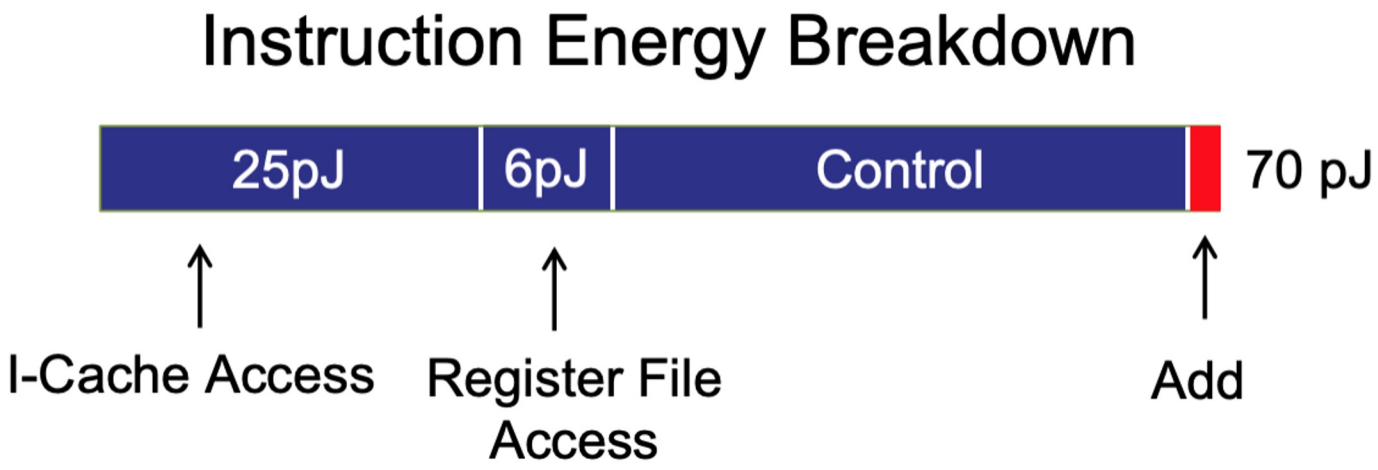
SFU

2. Hardware Efficiency

SFU

Where does the Energy go?

- Energy breakdown of an add instruction in a 45nm CPU
- How can we optimize this?



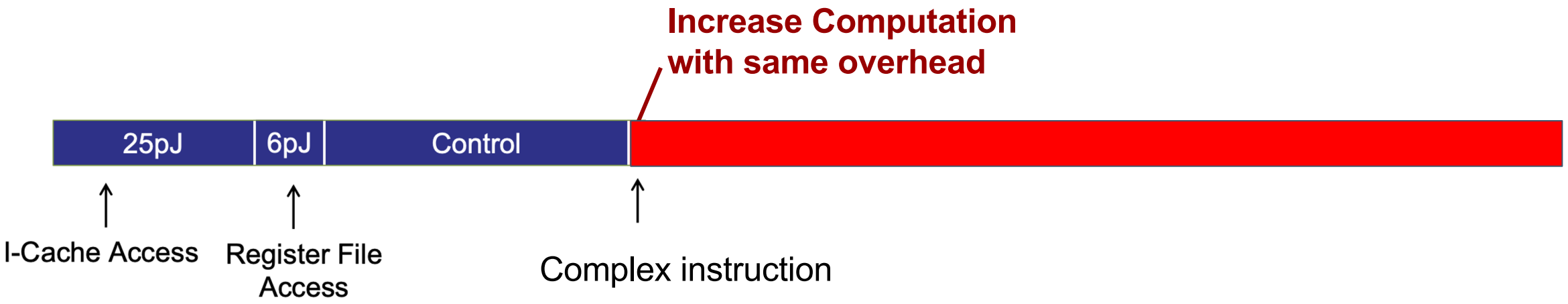
Source: Mark Horowitz "Computing's Energy Problem (and What we can do about it)" ISSCC 2014

Hardware Efficiency Approaches

SFU

1. Arithmetic
   - Specialized Instructions: To amortize overhead.
   - Lower precision (Quantization)
2. Memory
   - Locality: Move data to inexpensive on-chip memory.
   - Reuse: To avoid expensive memory fetches.
3. Ineffectual Operations
   - Sparsity: Skip useless operations

Amortize Overhead



Half-precision Fused Multiply-Add

4-way dot-product

16x16 matrix multiplication

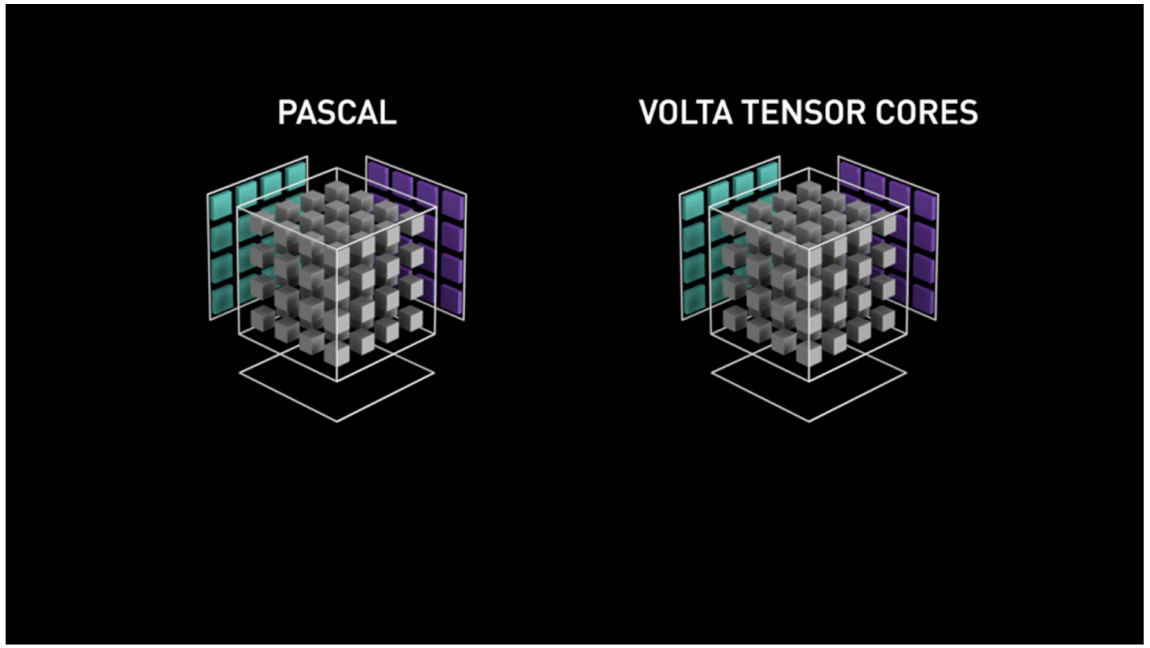
Operation	Energy**	Overhead*
HFMA	1.5pJ	2000%
HDP4A	6.0pJ	500%
HMMA	110pJ	27%

Source: Bill Dally "Hardware for Deep Learning" SysML 2018

“Special” Instruction Examples

SFU

GPU

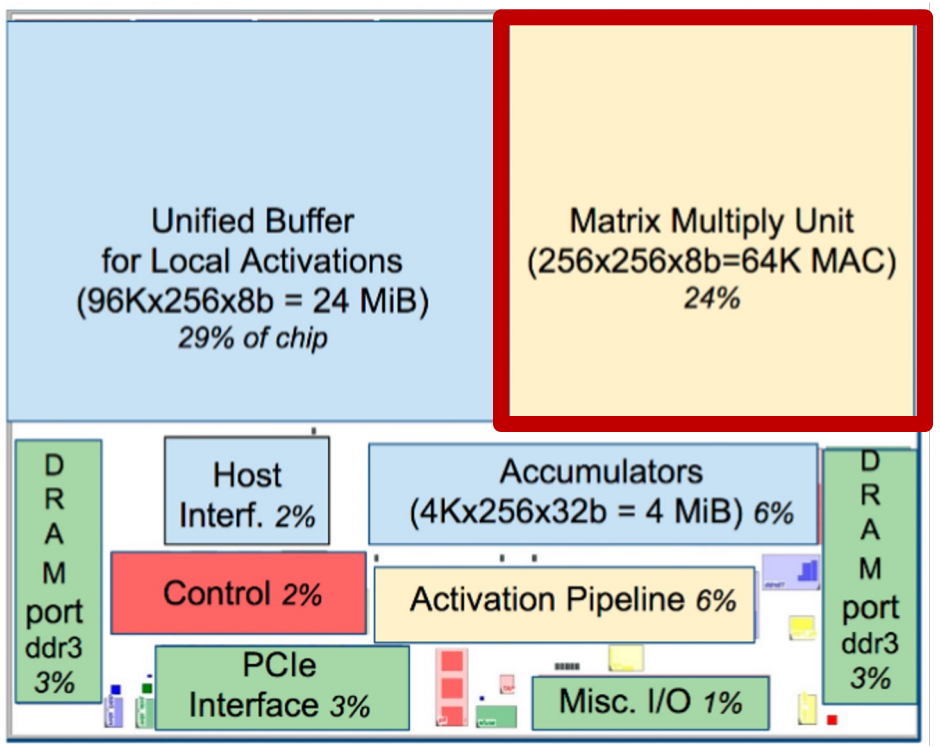


16x16 = 256* MAC/cycle

*~ 500 tensor cores per GPU

Source: Nvidia

ASIC (TPUv1)

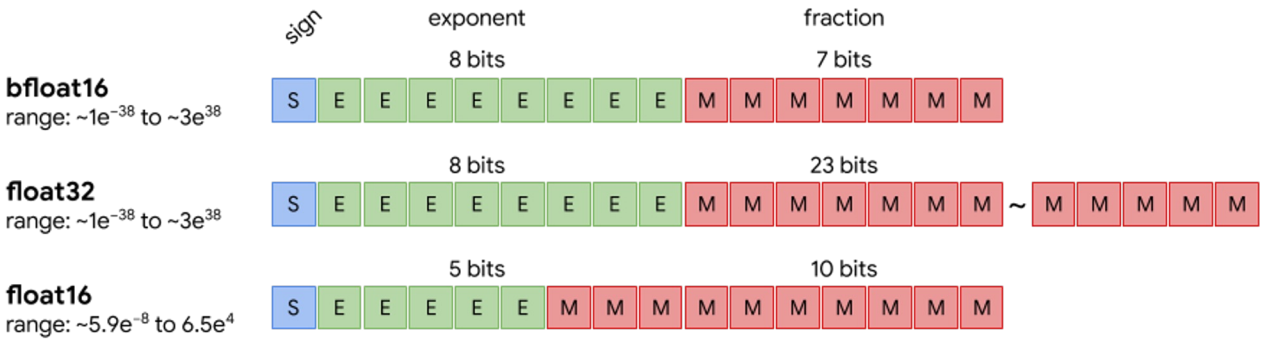


256x256 = 64 kMAC/cycle

Source: Google

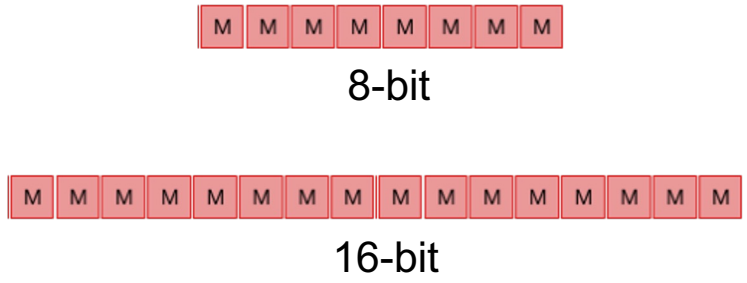
Numerical Format and Precision

Floating Point



- IEEE standard includes FP32 and FP16
- Many exotic FP numbers in DNN
  - E.g. bfloat, minifloat

Integer



- Whole numbers only
- (typically) much cheaper circuit area and power

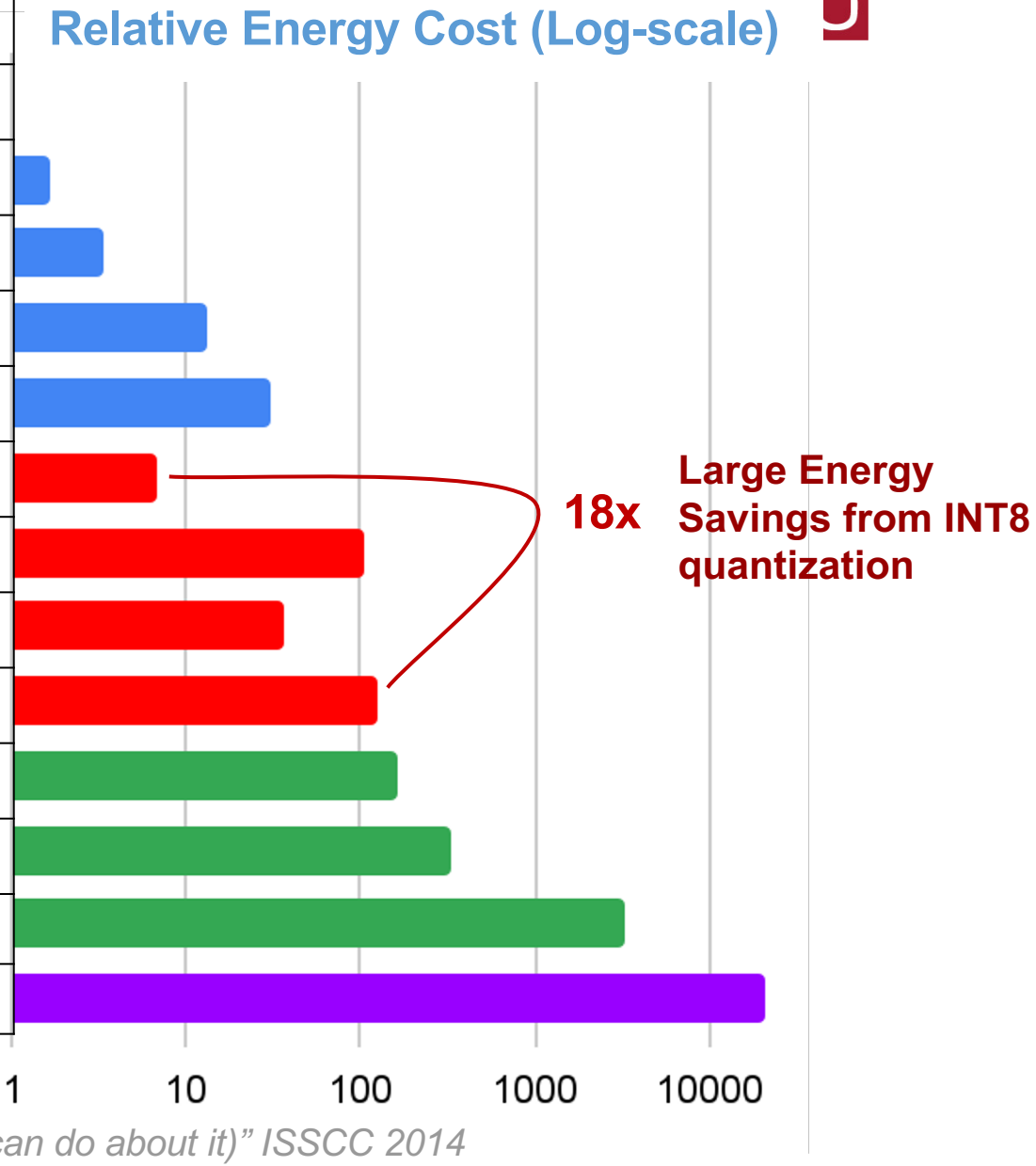
Source: Google

Cost of Arithmetic Operations

Operation	Precision	Energy (pJ)
Addition	INT8	0.03
	INT16	0.05
	INT32	0.1
	FP16	0.4
	FP32	0.9
Multiplication	INT8	0.2
	INT32	3.1
	FP16	1.1
	FP32	3.7
32-bit SRAM Read (8 kB)		5
32-bit SRAM Read (32 kB)		10
32-bit SRAM Read (1 MB)		100
32-bit DRAM Read		640

4x

16x



Adapted from Mark Horowitz "Computing's Energy Problem (and What we can do about it)" ISSCC 2014

SFU

QUESTION

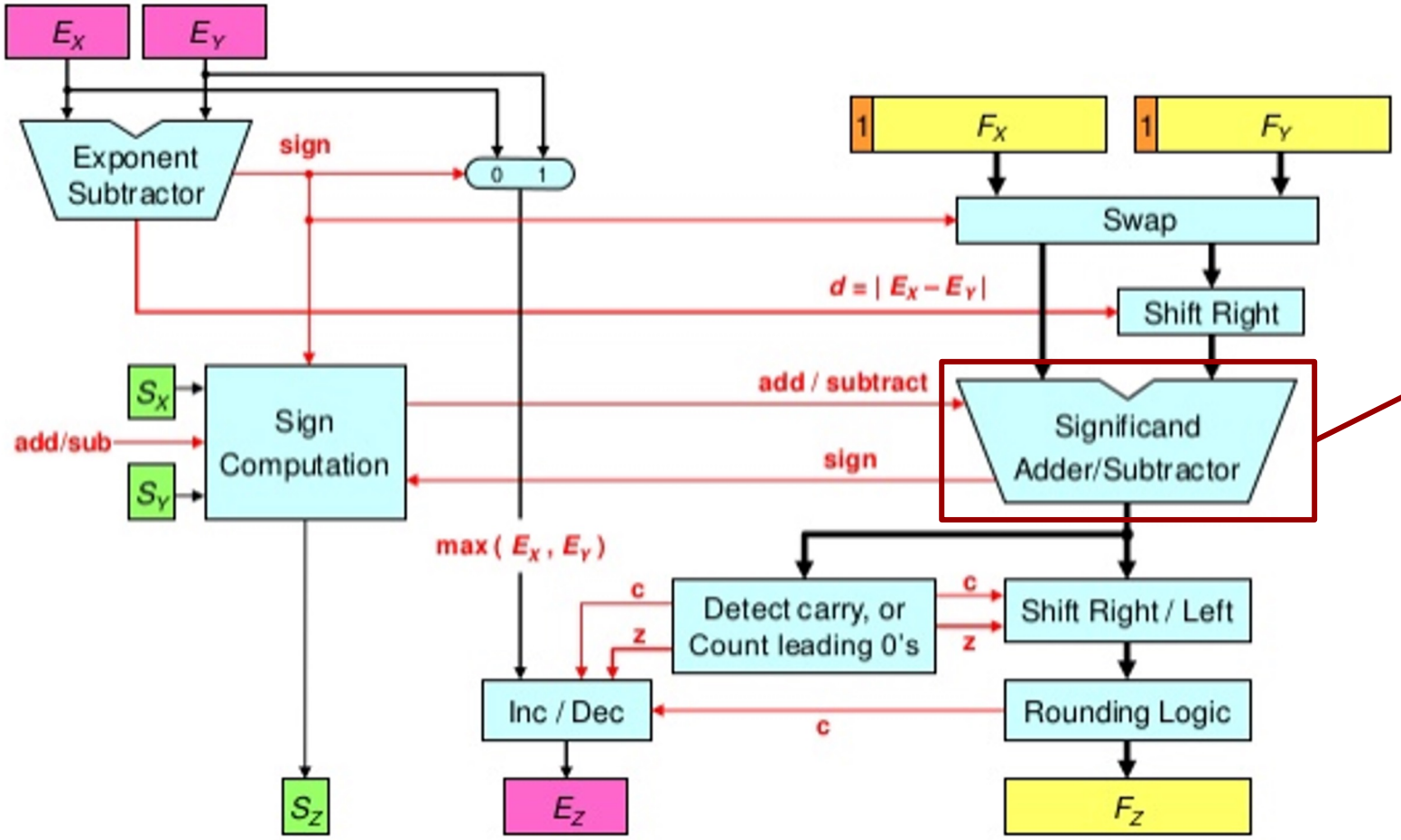
Why is floating-point add so expensive compared to integer add?

Operation	Precision	Energy (pJ)
Addition	INT8	0.03
	INT16	0.05
	INT32	0.1
	FP16	0.4
	FP32	0.9
Multiplication	INT8	0.2
	INT32	3.1
	FP16	1.1
	FP32	3.7

9x

1.2x

Floating-Point Addition



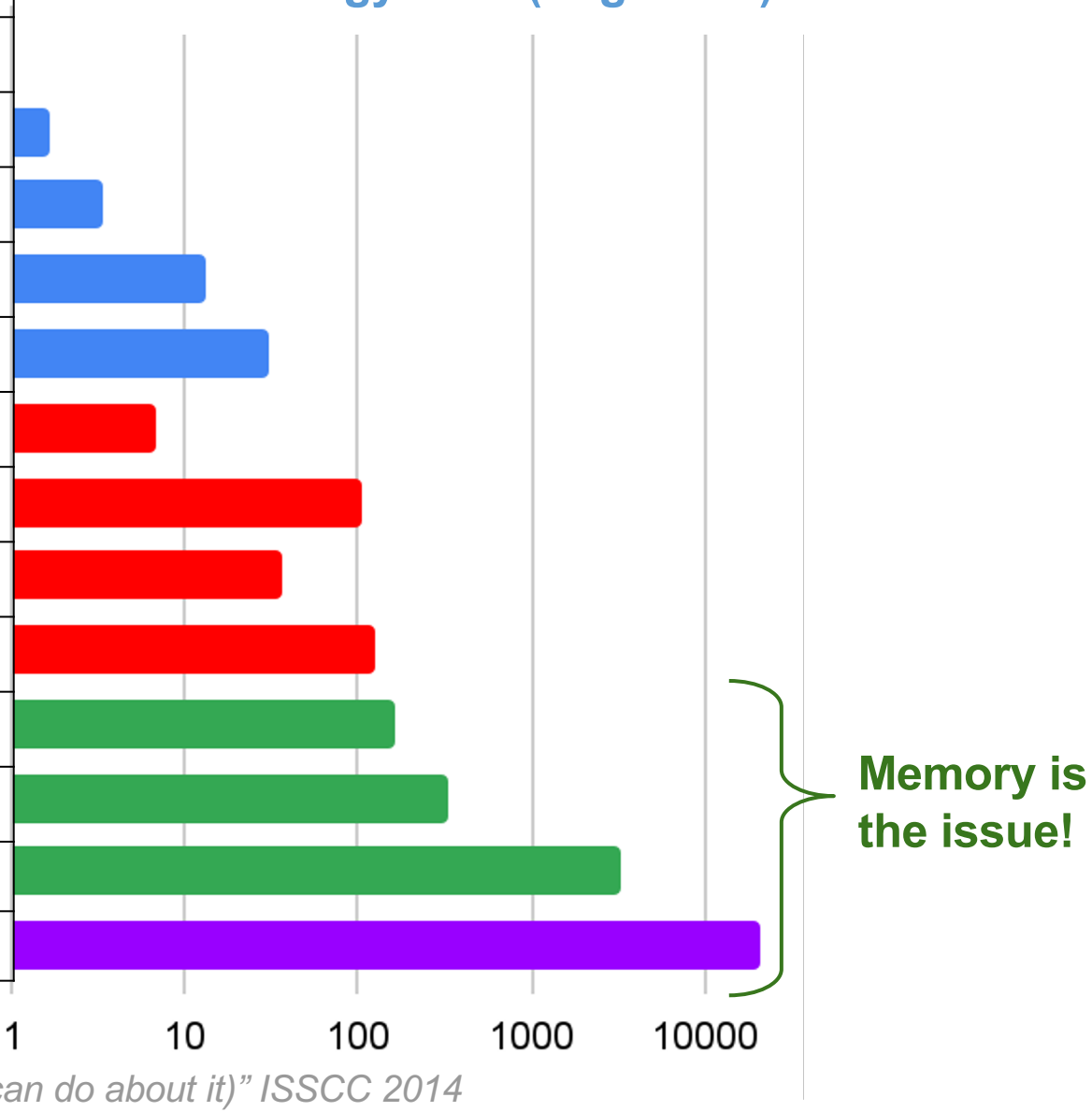
SFU

2. Hardware Efficiency

1. Arithmetic
   - Specialized Instructions: To amortize overhead. ✓
   - Lower precision (Quantization) ✓
2. Memory
   - Locality: Move data to inexpensive on-chip memory.
   - Reuse: To avoid expensive memory fetches.
3. Ineffectual Operations
   - Sparsity: Skip useless operations
   - Compressed Sparse Column (CSC) Format

Cost of Arithmetic Operations

Operation	Precision	Energy (pJ)
Addition	INT8	0.03
	INT16	0.05
	INT32	0.1
	FP16	0.4
	FP32	0.9
Multiplication	INT8	0.2
	INT32	3.1
	FP16	1.1
	FP32	3.7
32-bit SRAM Read (8 kB)		5
32-bit SRAM Read (32 kB)		10
32-bit SRAM Read (1 MB)		100
32-bit DRAM Read		640

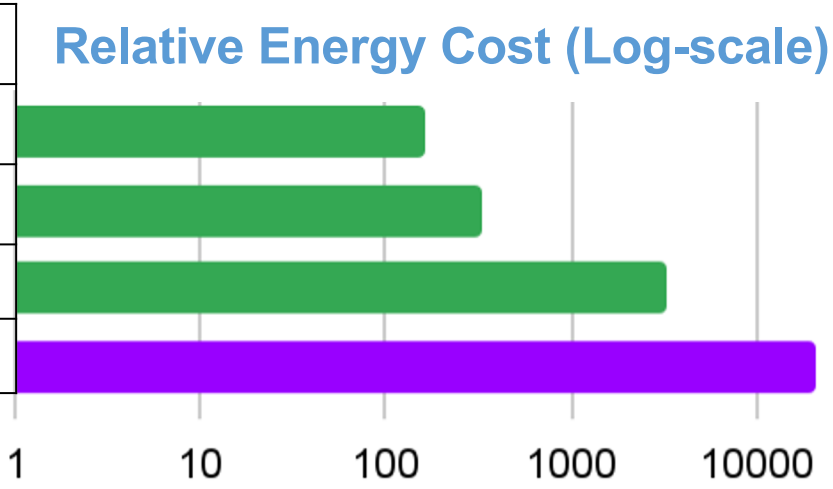


Adapted from Mark Horowitz "Computing's Energy Problem (and What we can do about it)" ISSCC 2014

Memory Hierarchy Optimizations

1. Get data close to the computation. (LOCALITY)
2. Once data is close - perform all computations with this data. (REUSE)

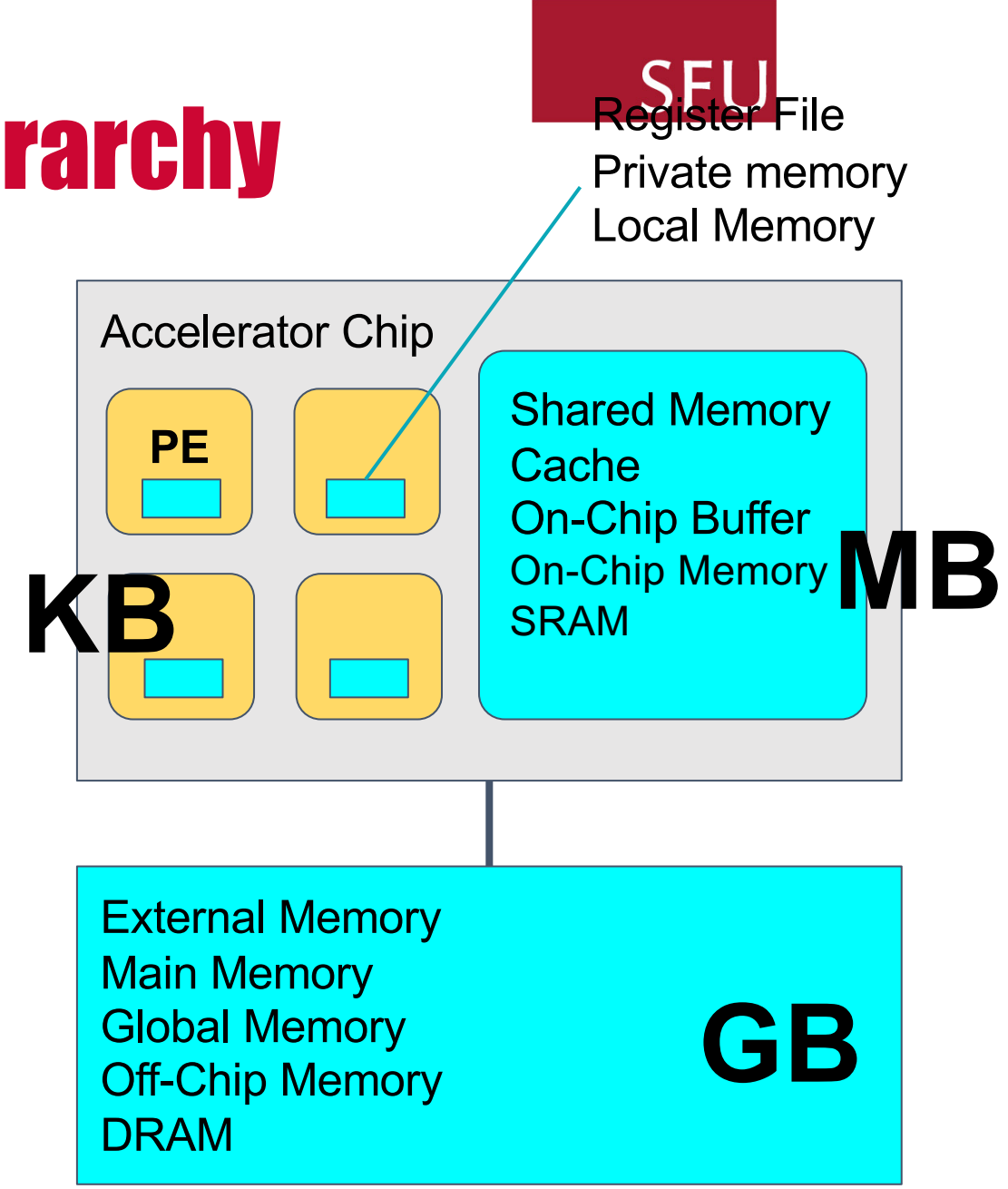
Operation	Energy (pJ)
32-bit SRAM Read (8 kB)	5
32-bit SRAM Read (32 kB)	10
32-bit SRAM Read (1 MB)	100
32-bit DRAM Read	640



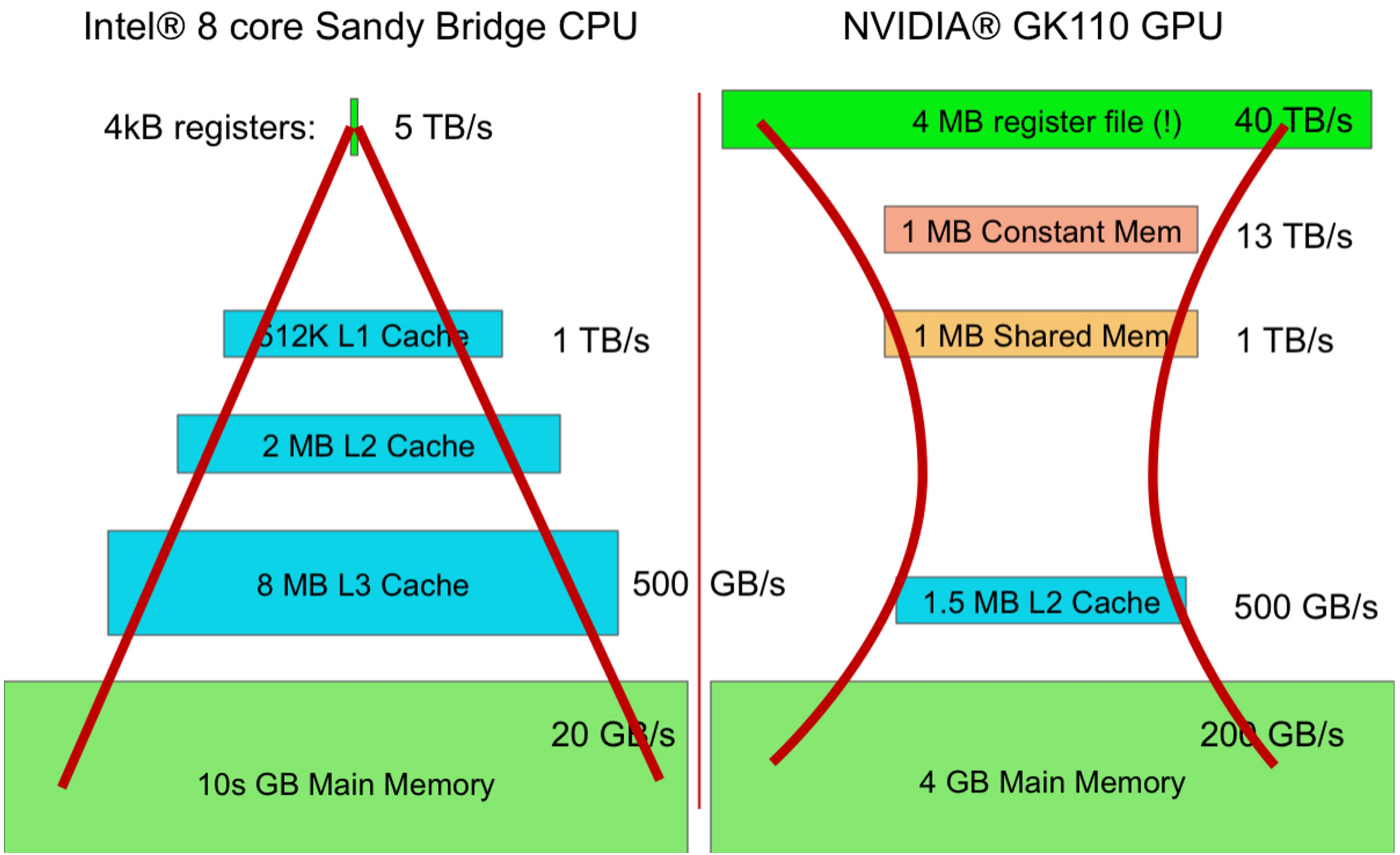
Memory Hierarchy

Why do we have a memory hierarchy?

- The closer you get to compute, the more $$ and scarce the memory resource becomes
- In *most* cases, the DNN parameters live off chip and are fetched layer-by-layer or tile-by-tile
- Data locality: how to get data close to the PEs (to keep them fully utilized)



Memory Hierarchy Examples

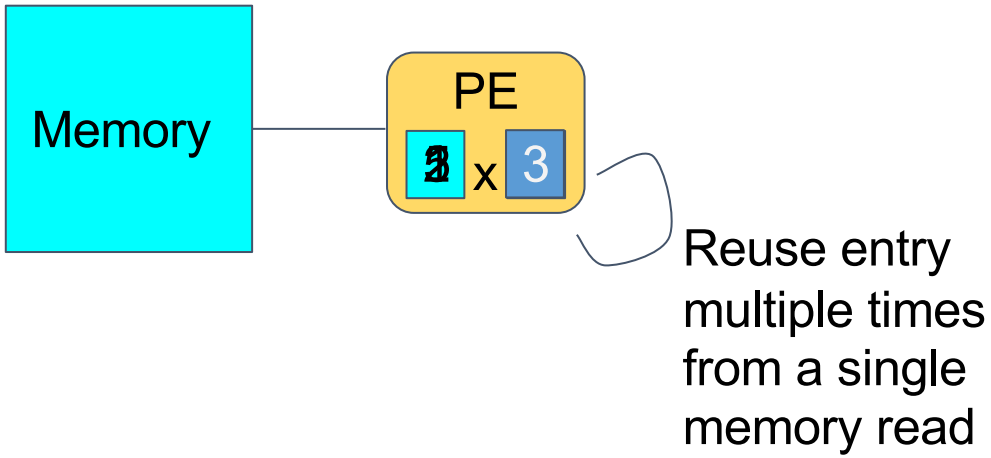


Source: Nvidia

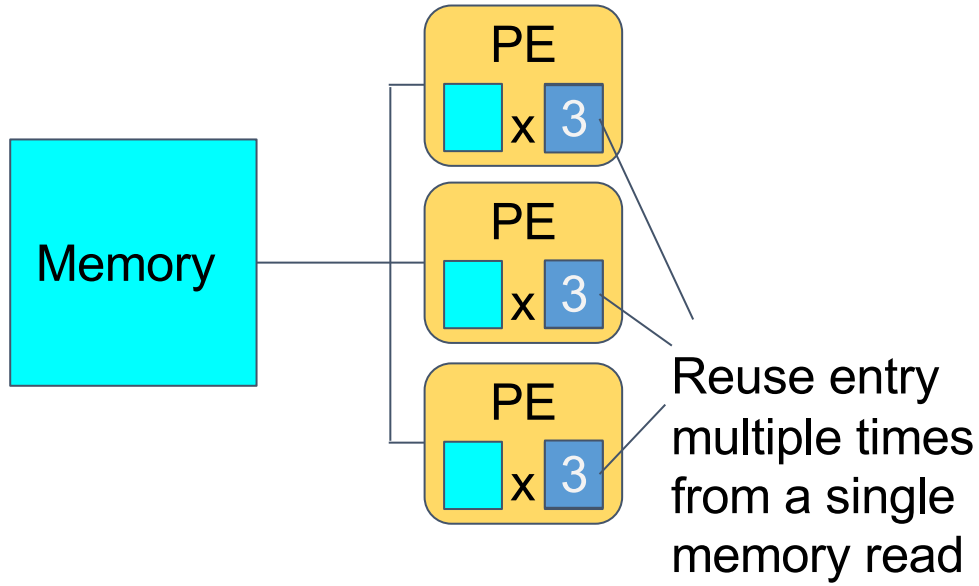
Data Reuse

- Temporal Reuse
- Spatial Reuse

Read once from memory, use same data multiple times by same PE



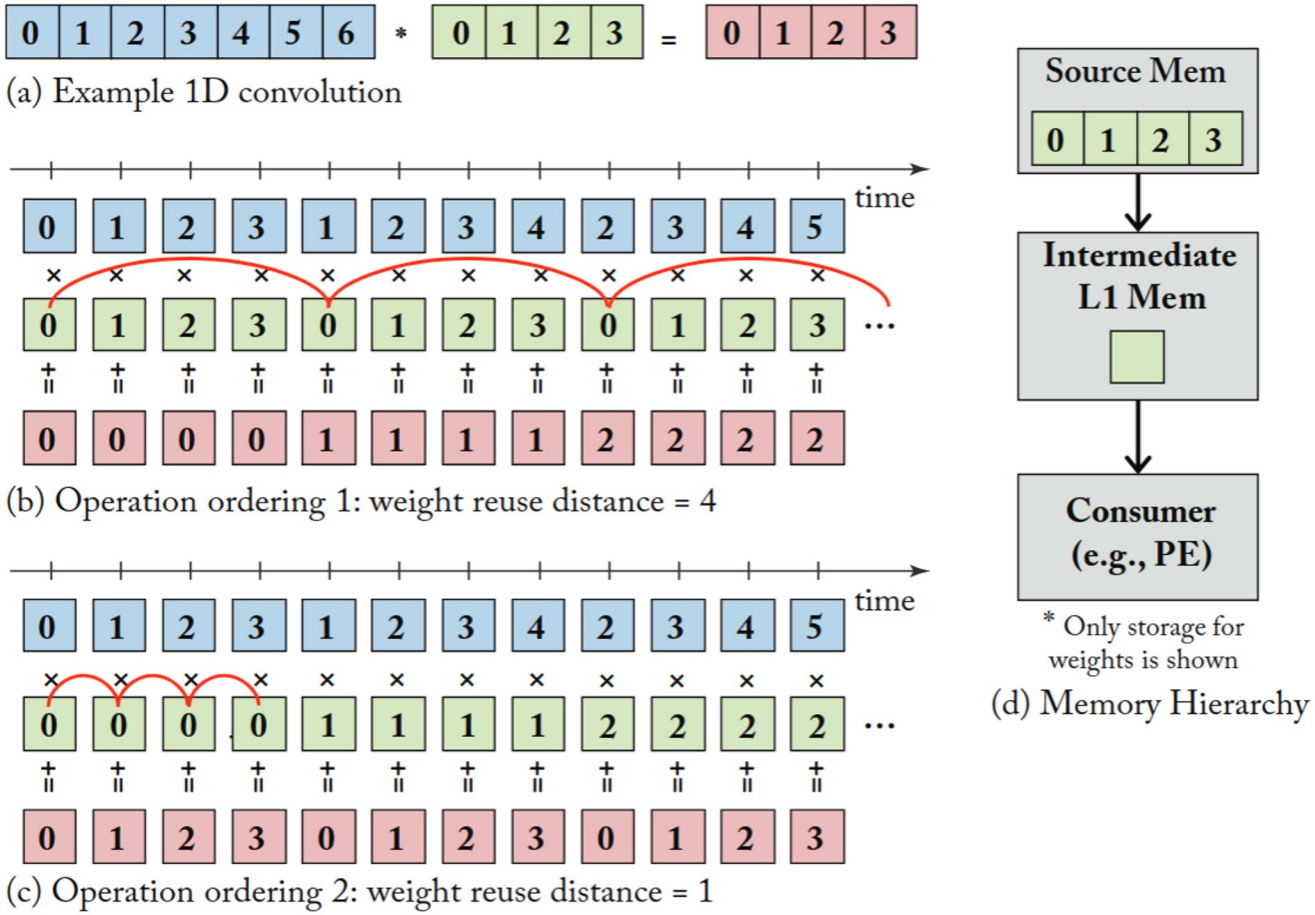
Read once from memory, use same data multiple times by multiple PEs



Temporal Reuse Example

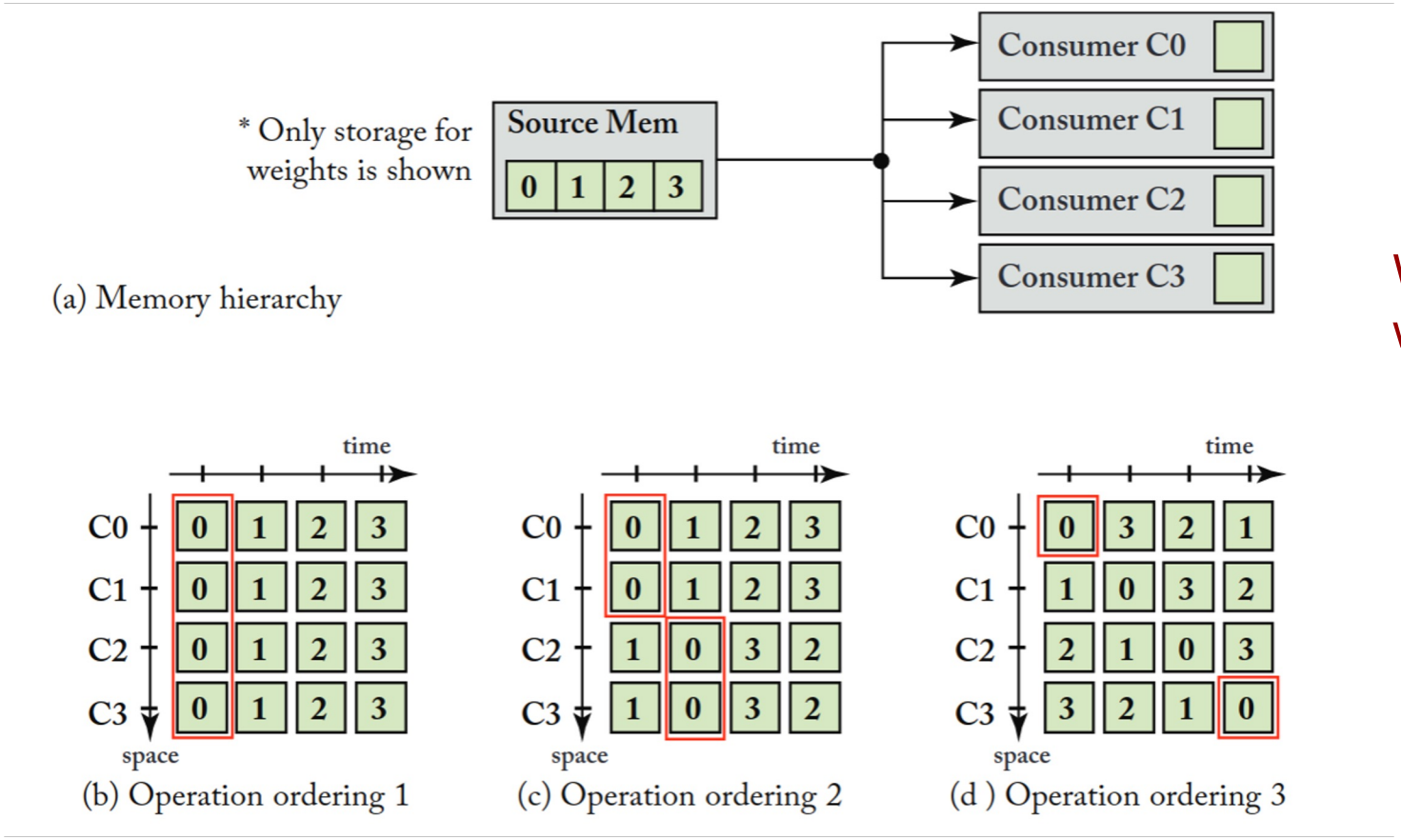
Reuse Distance

- Number of memory accesses
- Size of your on-chip memory
- Smaller is better



Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

Spatial Reuse Example

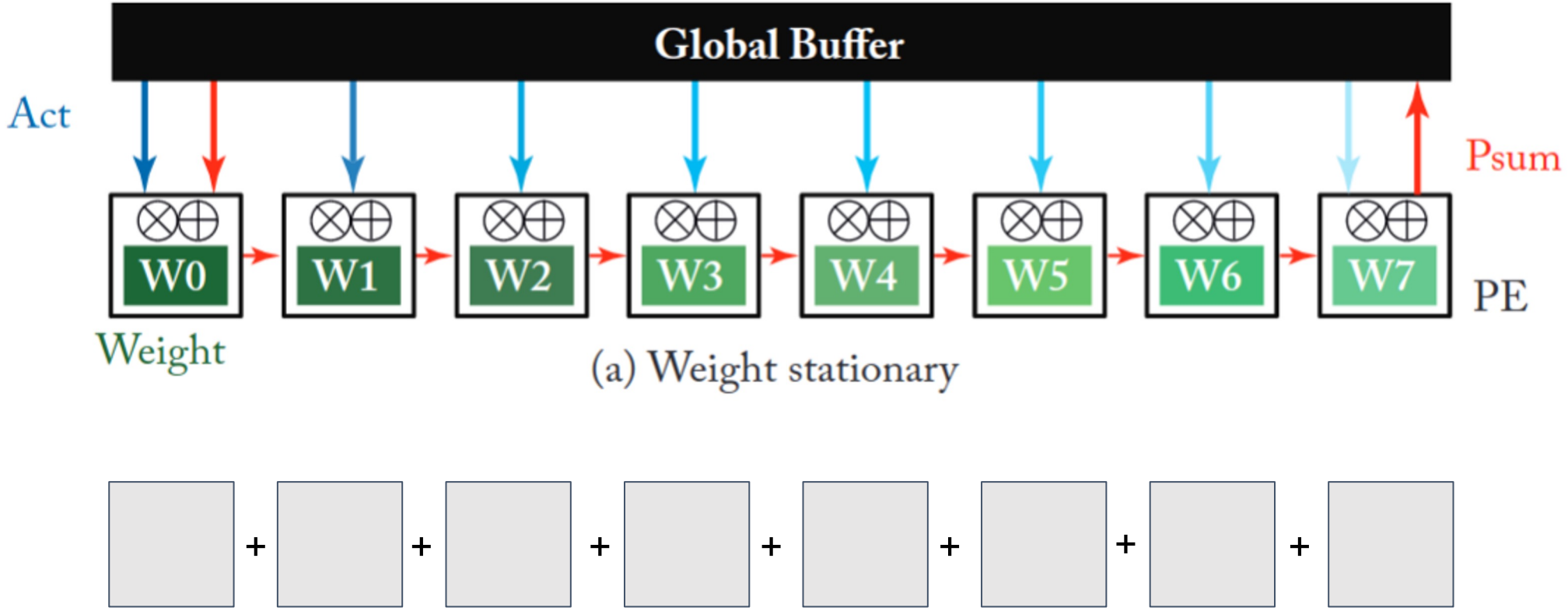


Why a smaller reuse distance with spatial reuse?

- Fewer memory read ports
  - Less area
- Fewer reads
  - Less power

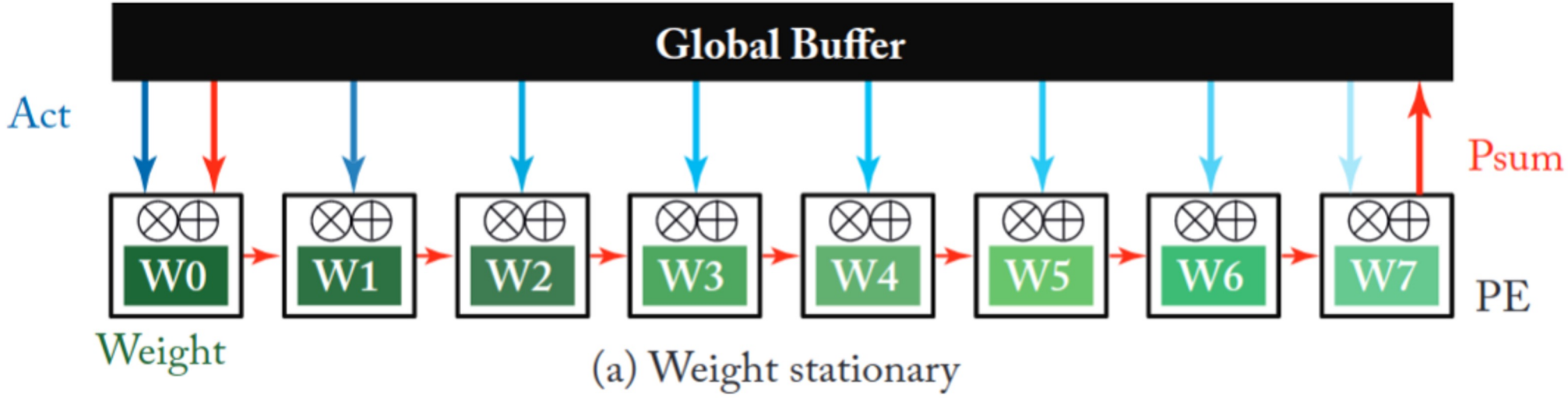
Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

What to Keep Stationary? Weights



Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

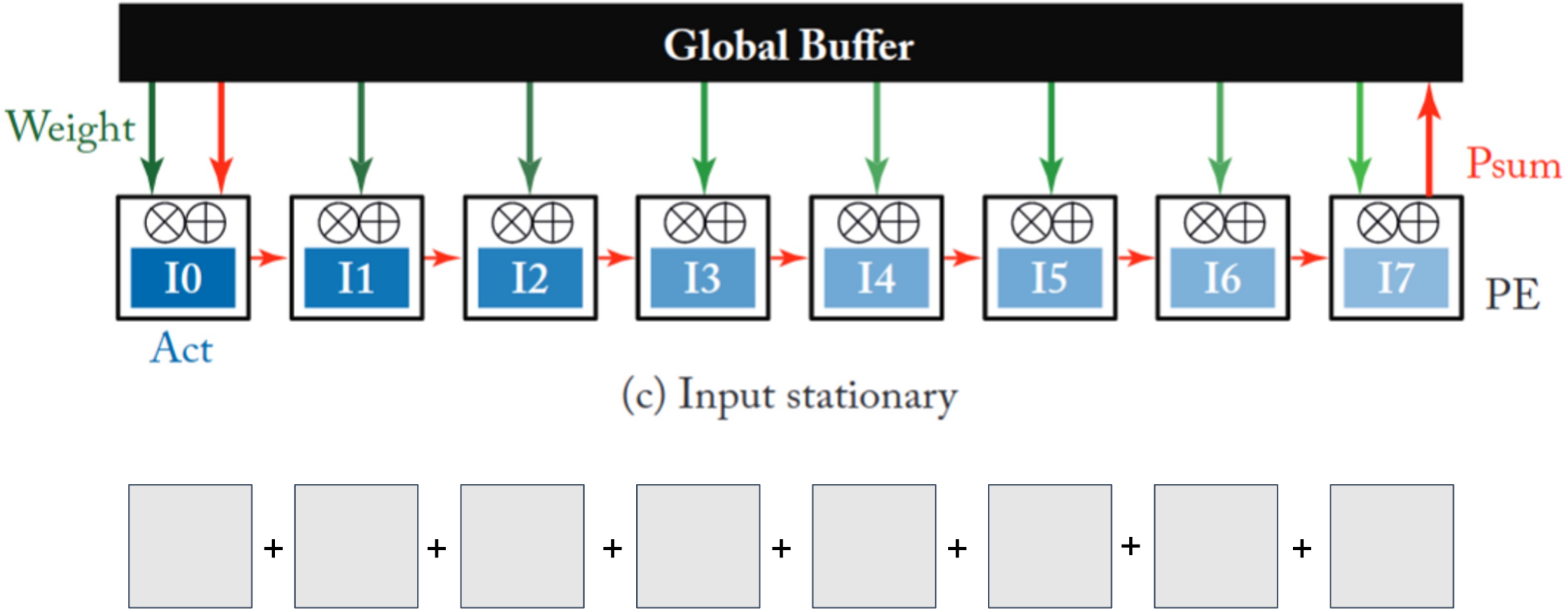
What to Keep Stationary? Weights



a0w0 + a1w1 + a2w2 + a3w3 + a4w4 + a5w5 + a6w6 + a7w7

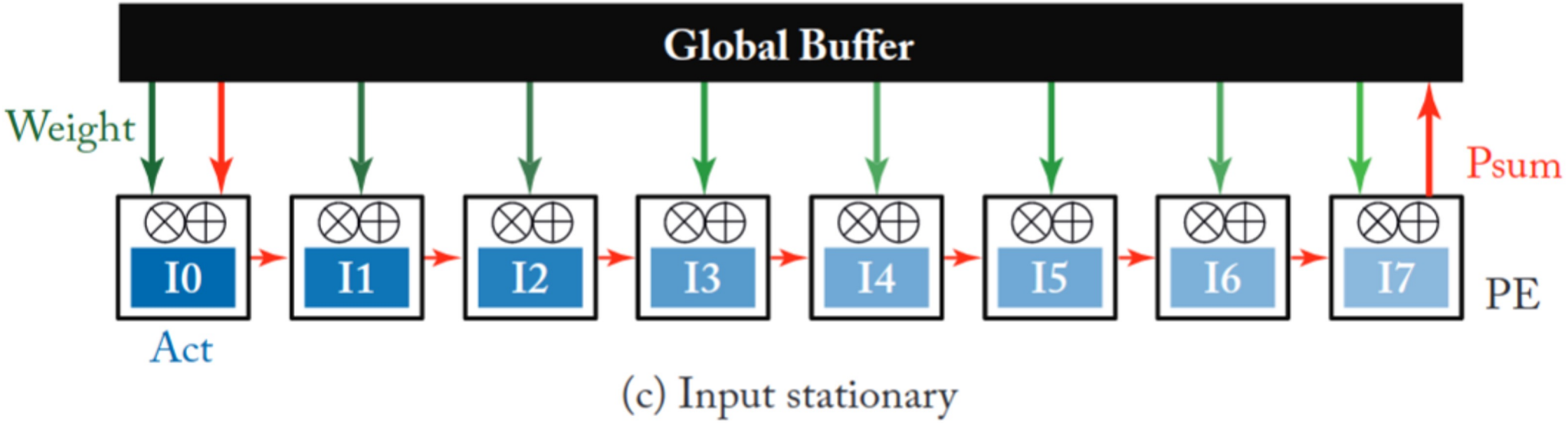
Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

What to Keep Stationary? Inputs



Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

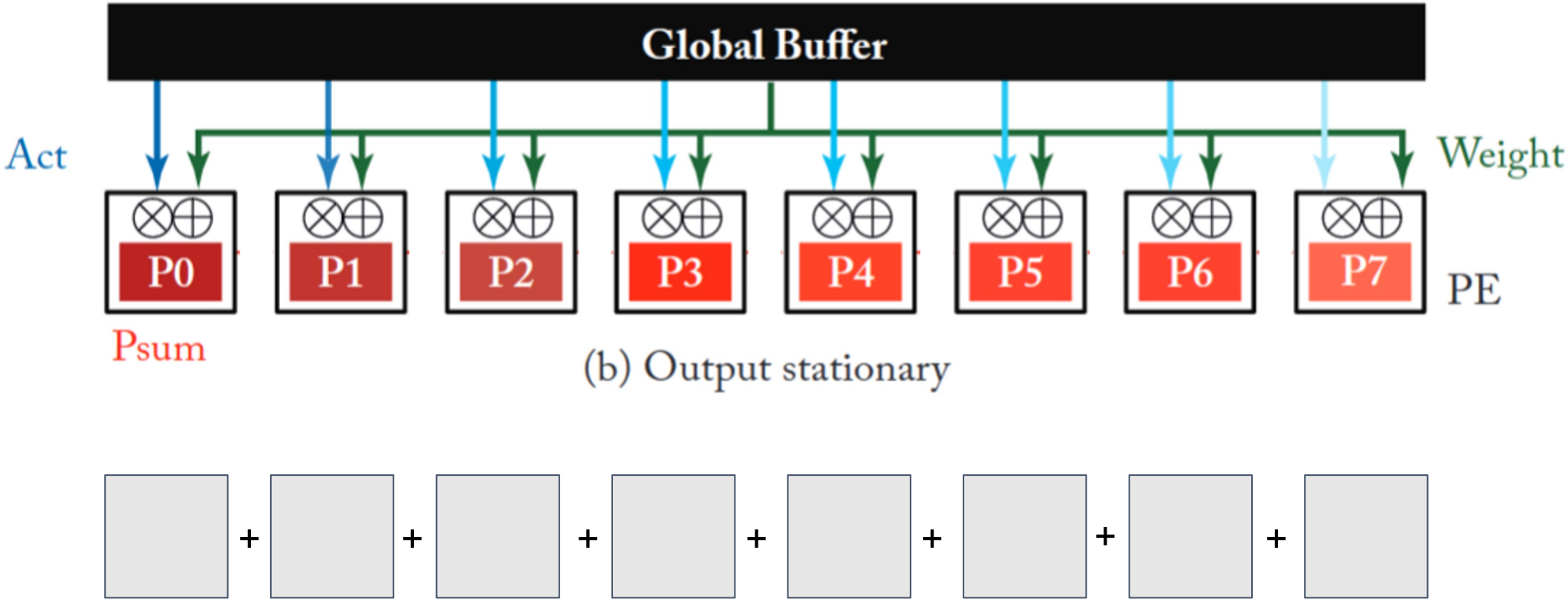
What to Keep Stationary? Inputs



a0w0 + a1w1 + a2w2 + a3w3 + a4w4 + a5w5 + a6w6 + a7w7

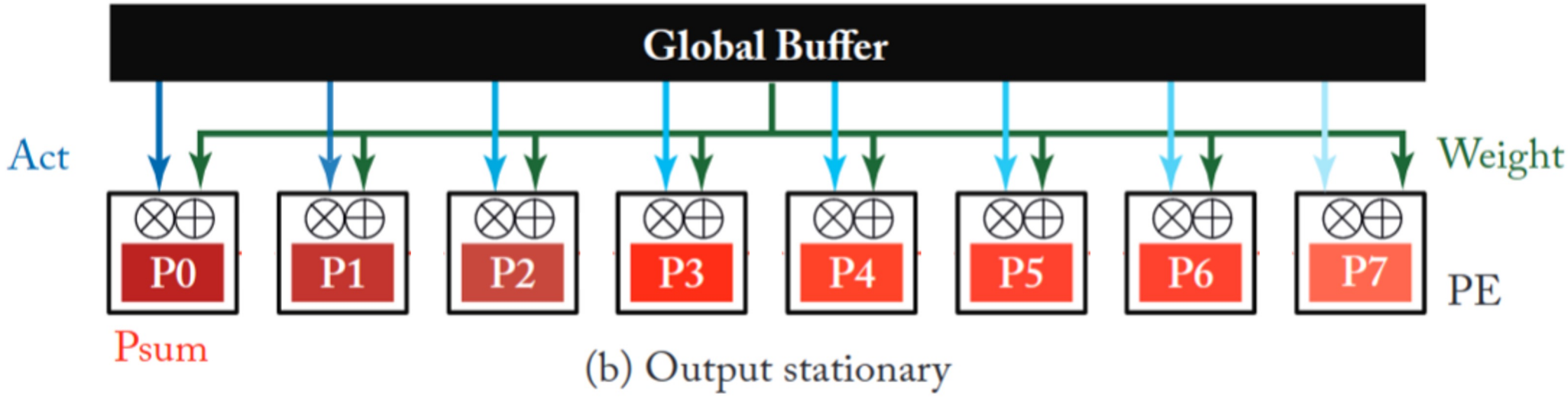
Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

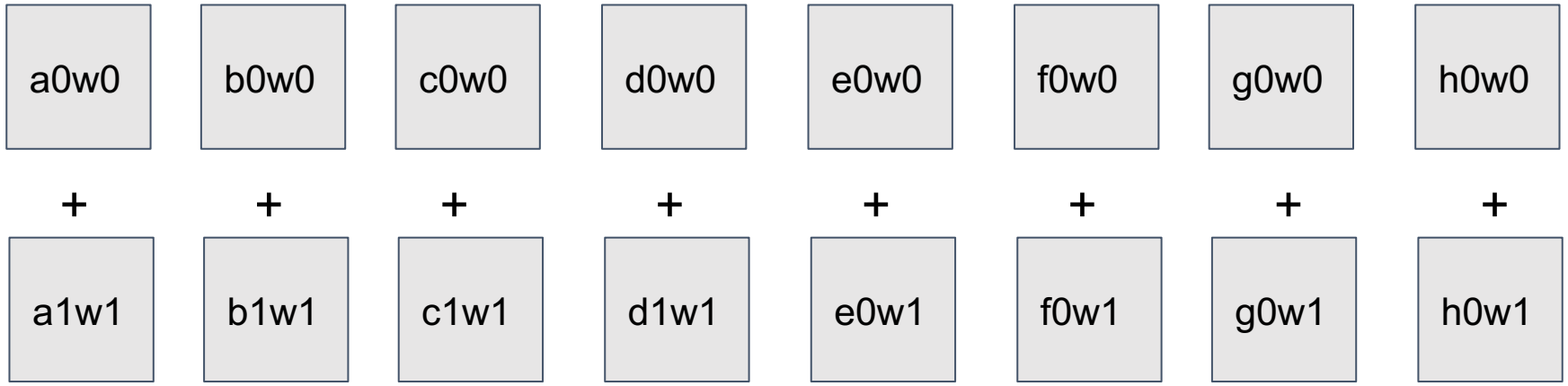
What to Keep Stationary? Outputs



Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

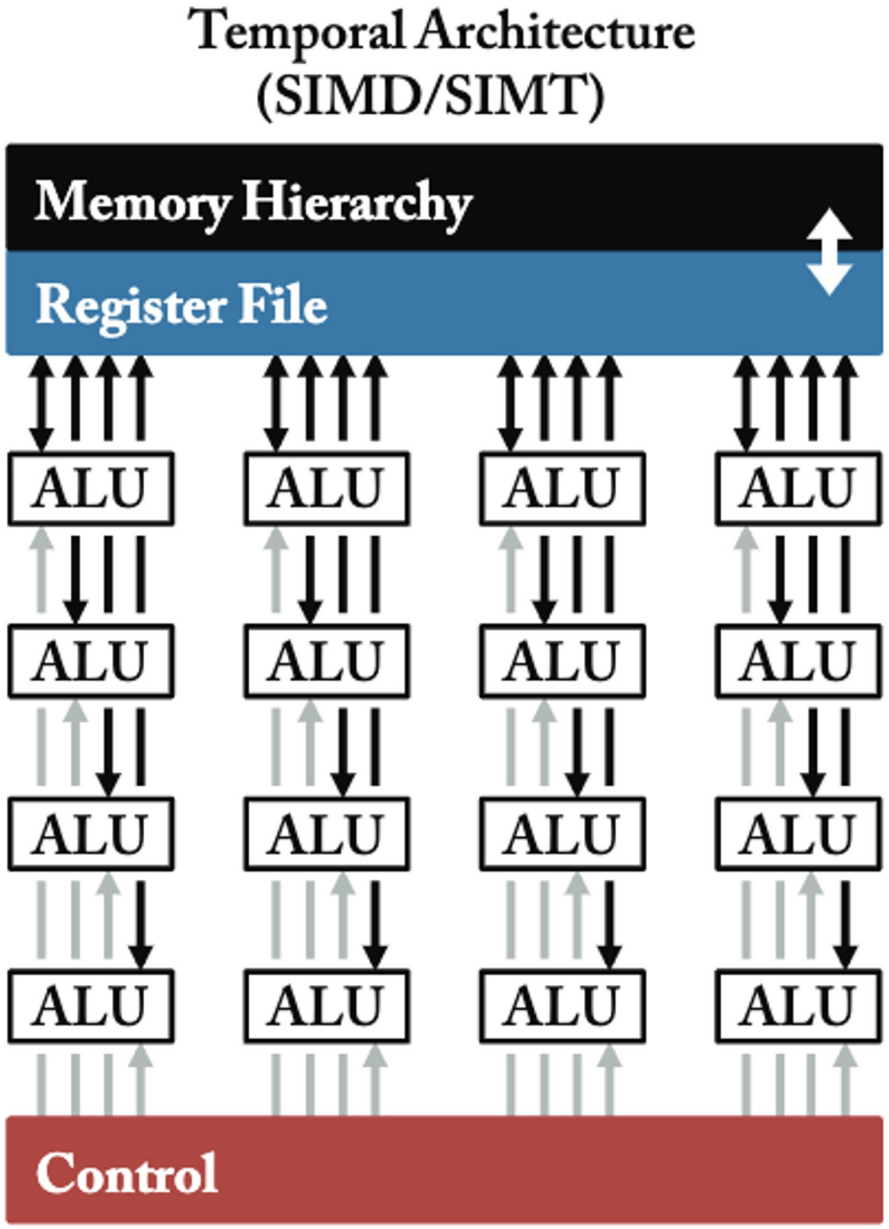
What to Keep Stationary? Outputs

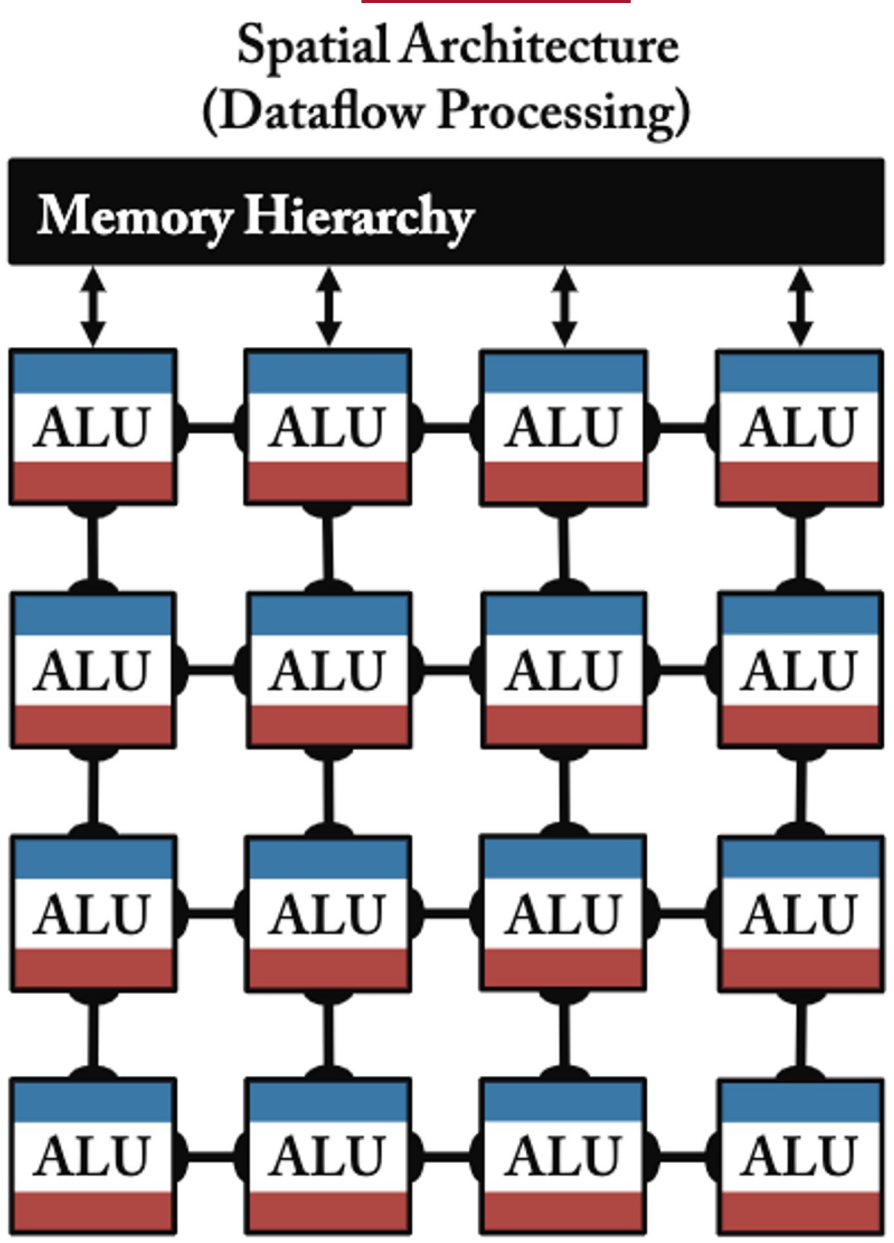




Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

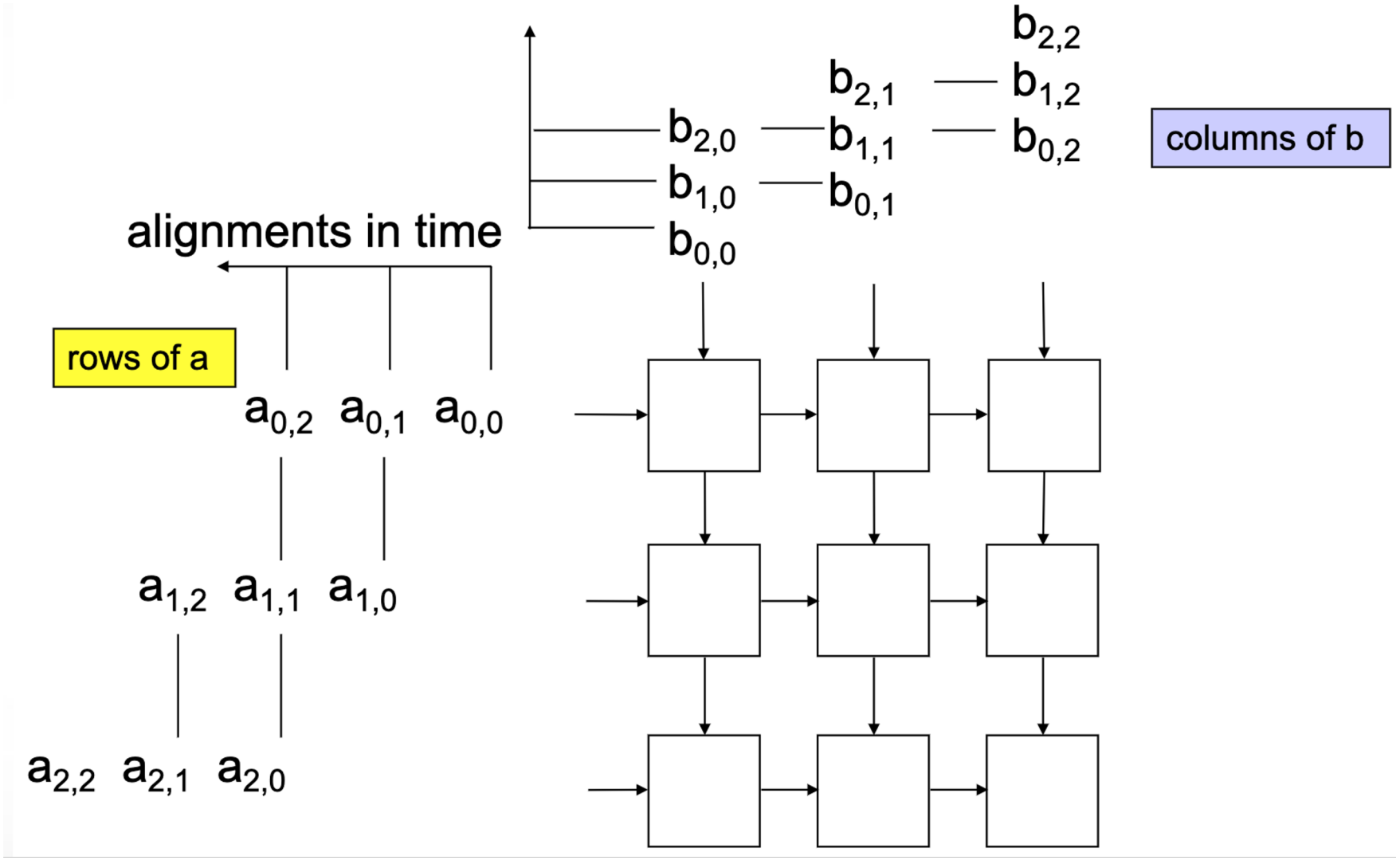
Highly Parallel Architectures





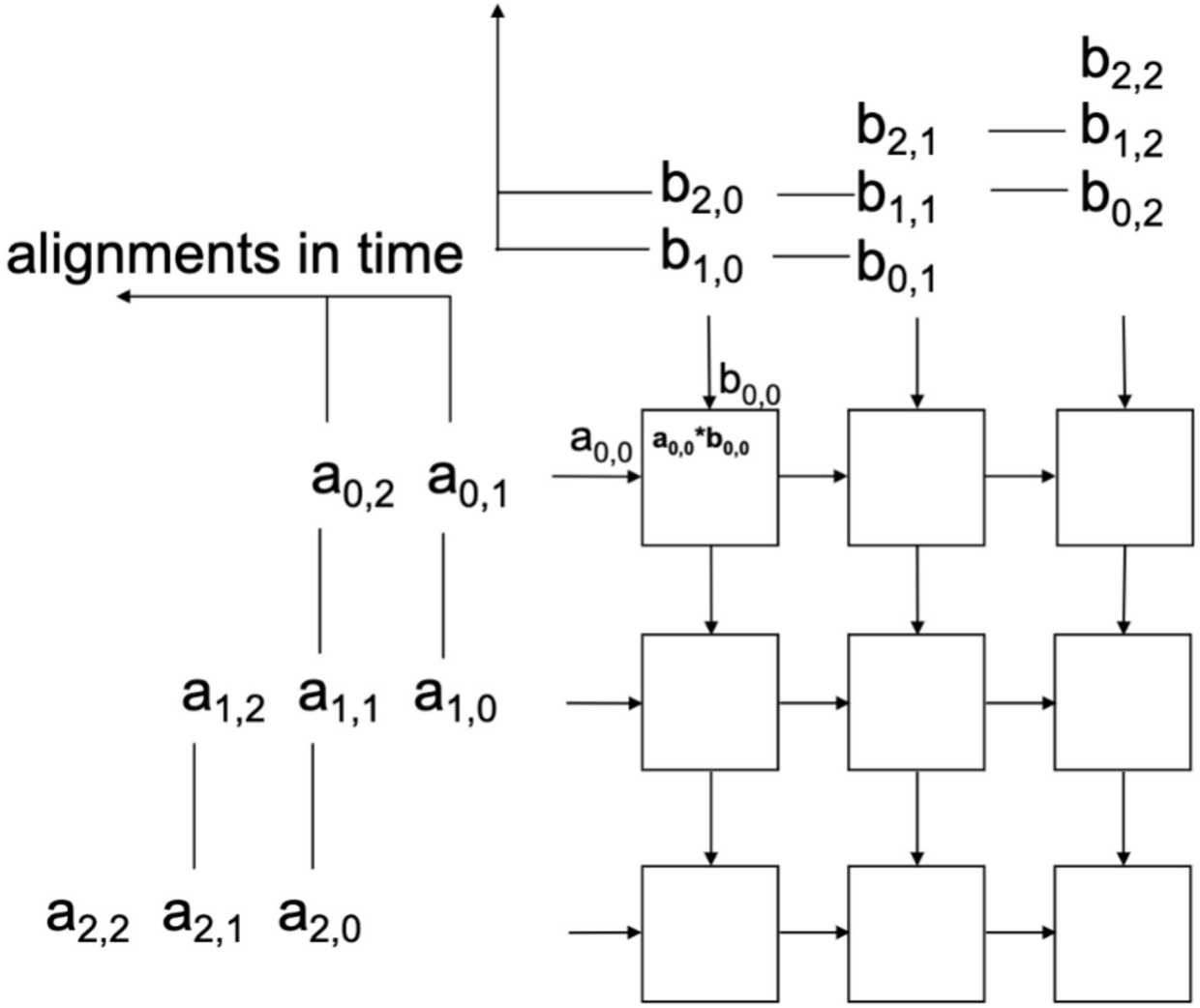
Source: V. Sze et al. "Efficient Processing of Deep Neural Networks" 2020

Systolic Array: Matrix Multiply Example

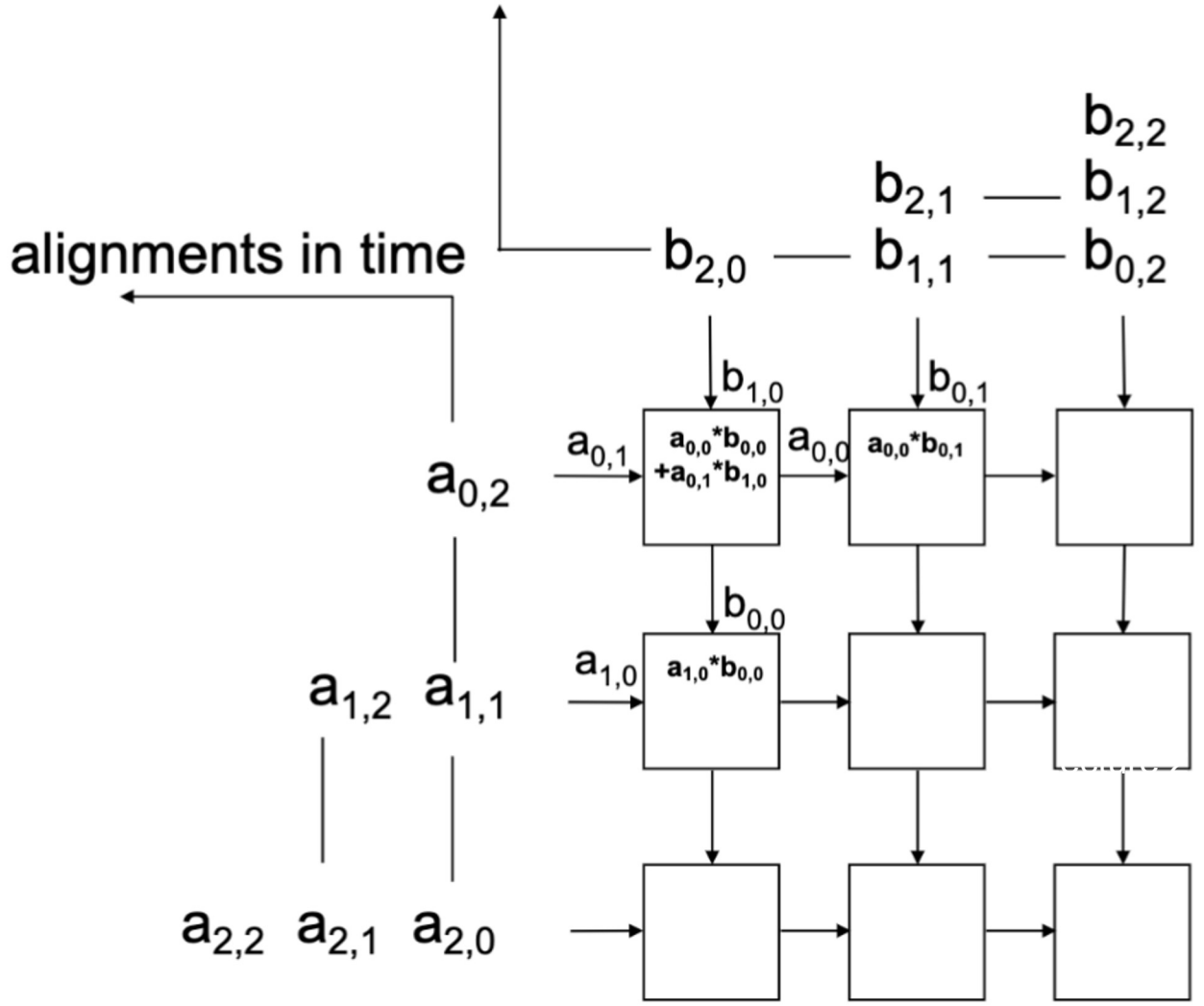


SFU

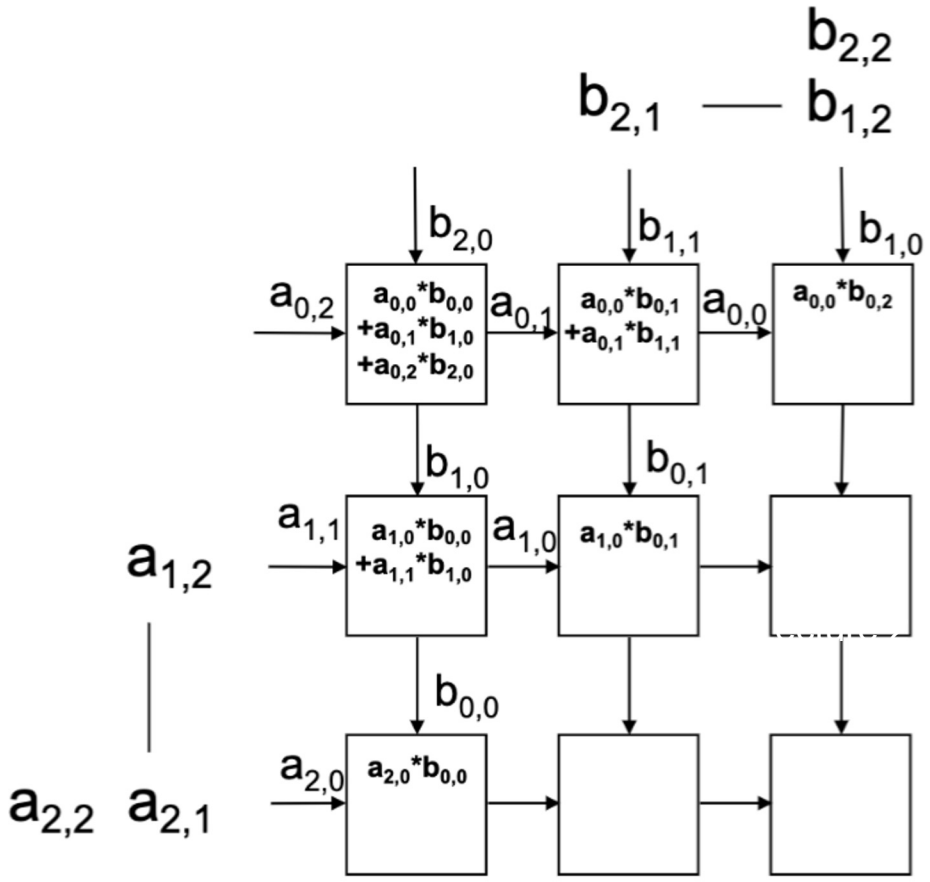
Systolic Array: Matrix Multiply Example



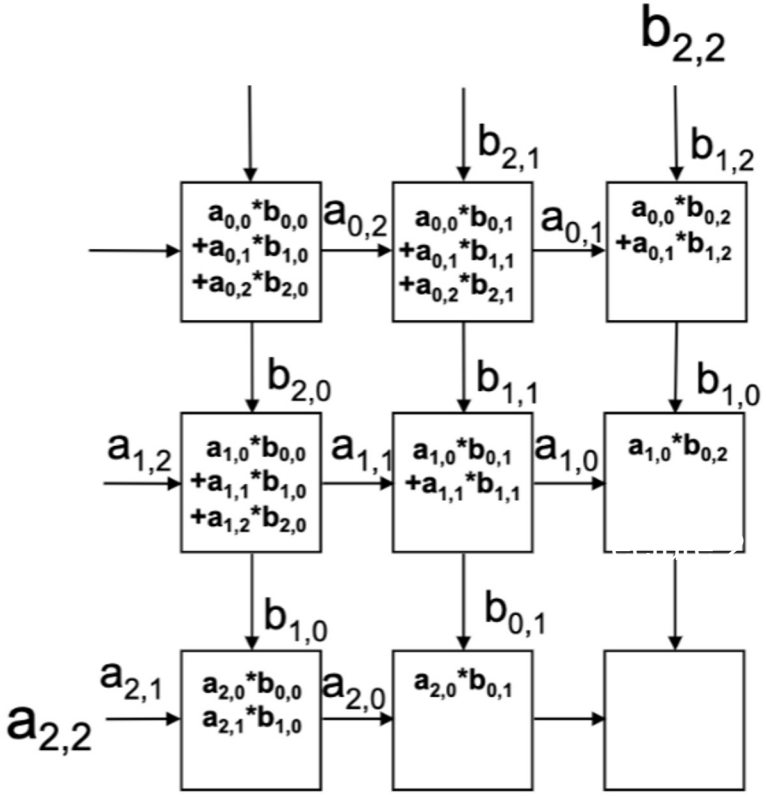
Systolic Array: Matrix Multiply Example



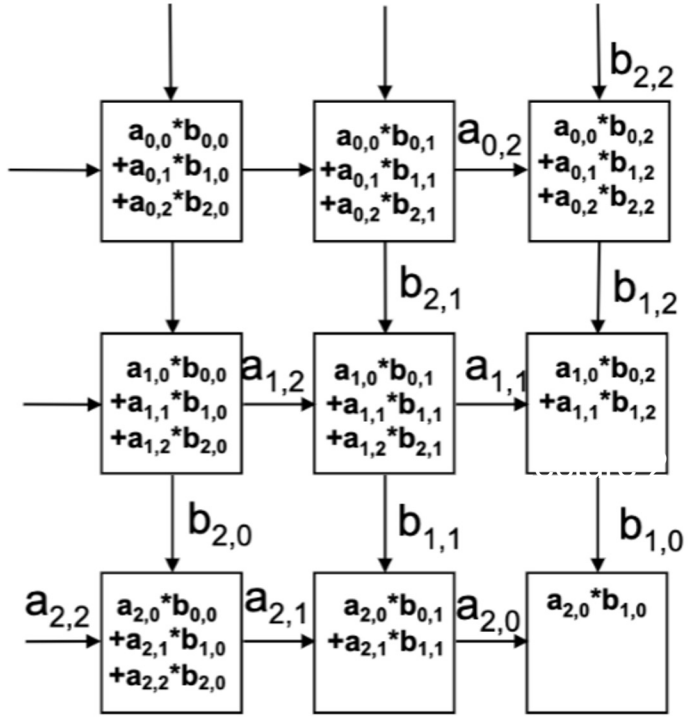
Systolic Array: Matrix Multiply Example



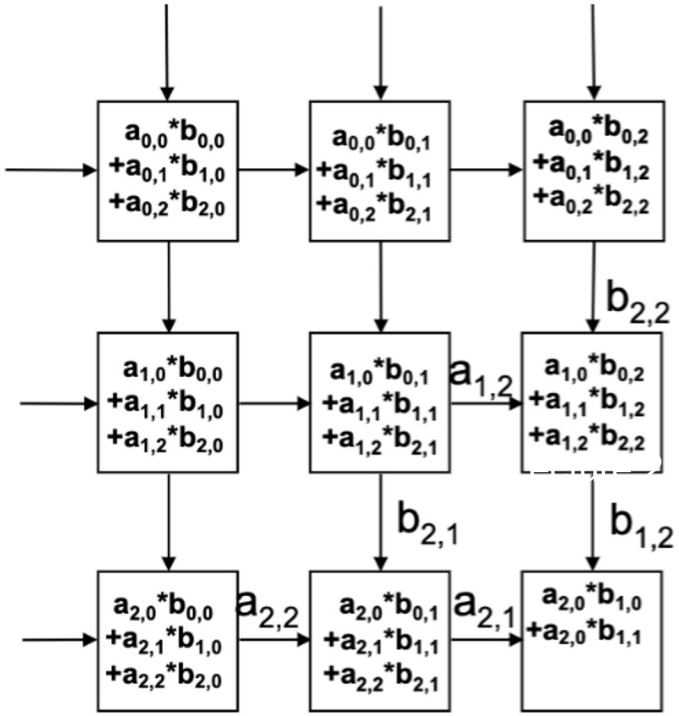
Systolic Array: Matrix Multiply Example



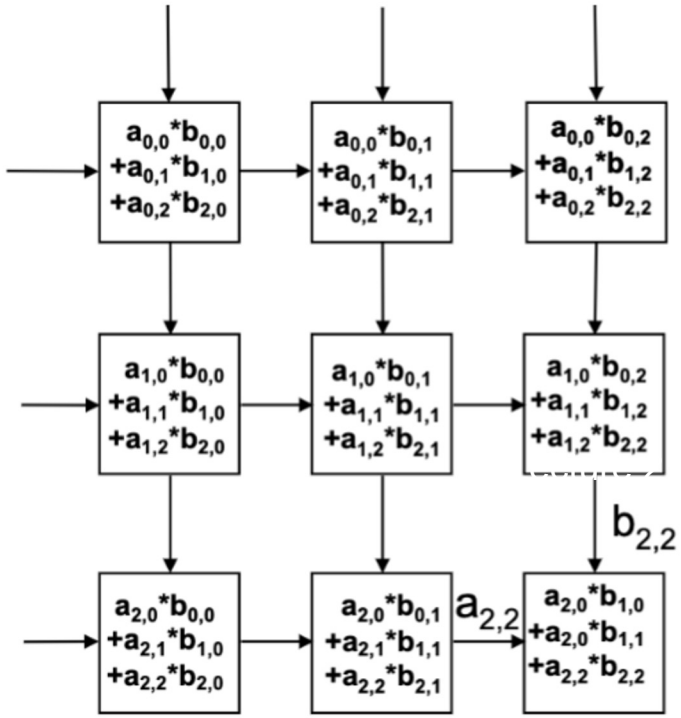
Systolic Array: Matrix Multiply Example



Systolic Array: Matrix Multiply Example



Systolic Array: Matrix Multiply Example



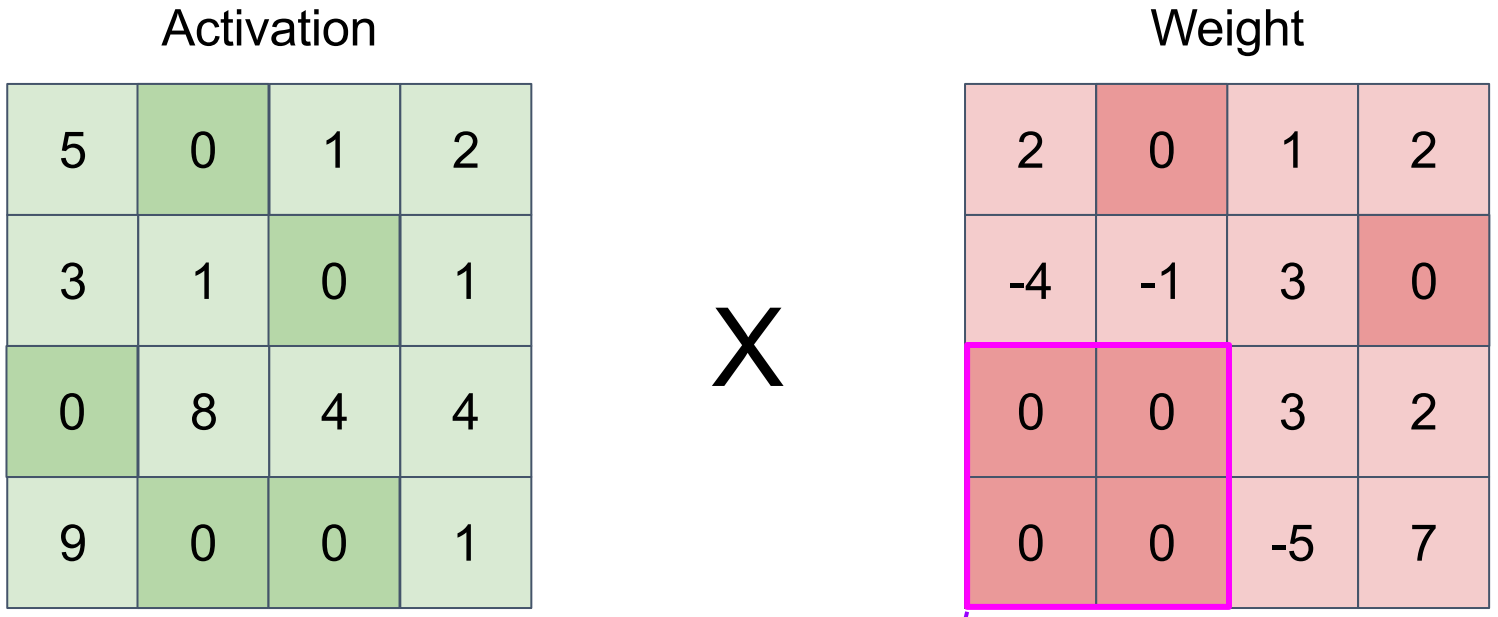
_____ stationary?

SFU

2. Hardware Efficiency

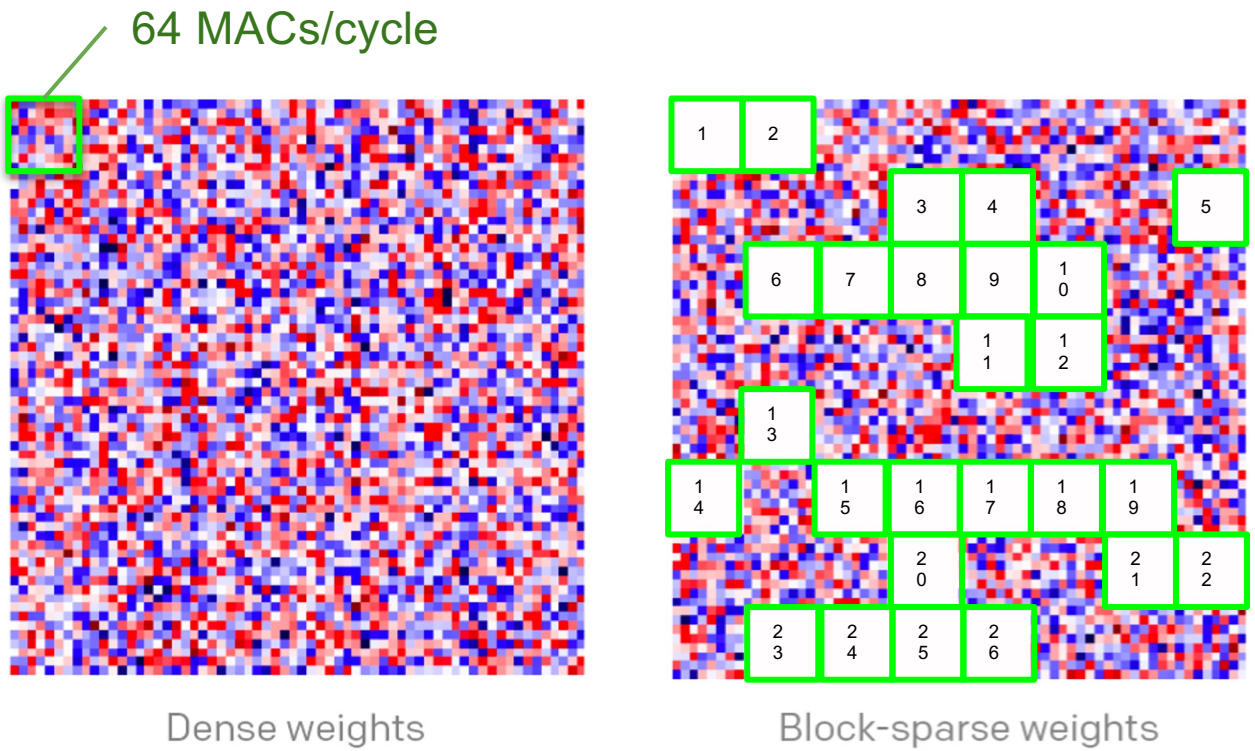
1. Arithmetic
   - Specialized Instructions: To amortize overhead. ✓
   - Lower precision (Quantization) ✓
2. Memory
   - Locality: Move data to inexpensive on-chip memory. ✓
   - Reuse: To avoid expensive memory fetches. ✓
3. Ineffectual Operations
   - Sparsity: Skip useless operations
   - Compressed Sparse Column (CSC) Format

Kinds of Sparsity



Coarse-grained "Block" Sparsity

- All DNN accelerators are parallel
  - Multiple MACs/cycle
- The smallest unit of computation that can be skipped is a large block (*recall* *amortized overhead*)
- Example:
  - Systolic array with 64 MACs/cycle
    - 8x8 pattern
  - 64x64 matrix = 4096 MACs
  - Total # cycles = 64 cycles
  - Block sparsity pattern needs to skip blocks of 8x8
  - Speedup = 64/(64-26) = 1.7X faster



Dense weights

Block-sparse weights

Source: Open AI

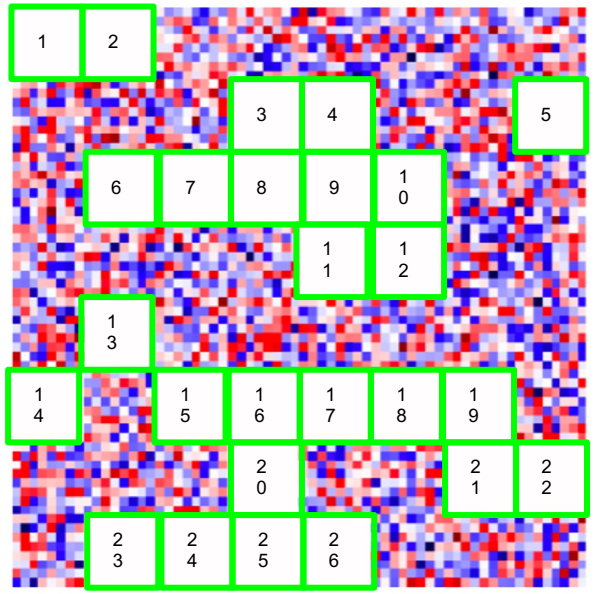
Coarse-grained "Block" Sparsity

- All DNN accelerators are parallel
  - Multiple MACs/cycle
- The smallest unit of computation that can be skipped is a large block (*recall* amortized overhead)
- Example:
  - Systolic array with 64 MACs/cycle
    - 8x8 pattern
  - 64x64 matrix = 4096 MACs
  - Total # cycles = 64 cycles
  - Block sparsity pattern needs to skip blocks of 8x8
  - Speedup = 64/(64-26) = 1.7X faster

Source: Open AI

0	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0
1	0	0	0	0	0	1	1
1	1	1	1	0	0	1	1
1	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0
1	1	1	0	1	1	0	0
1	0	0	0	0	1	1	1

Corresponding sparsity pattern



Block-sparse weights

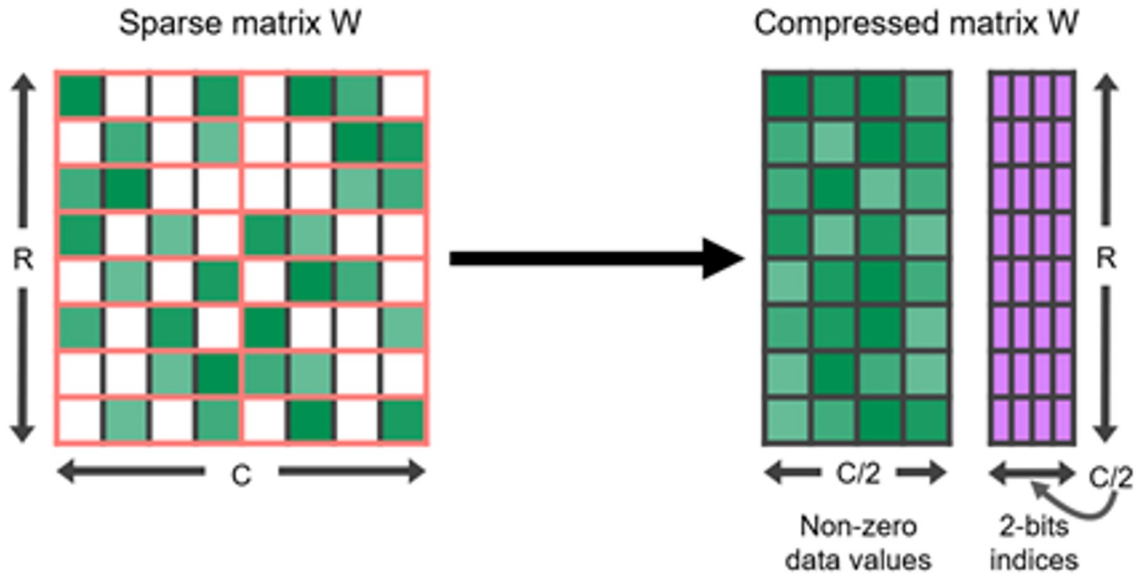
Simplest way to leverage sparsity with low overhead

⇨ Single bit per 8x8 block (1/64 = 1.6% overhead)

⇨ Simple control logic because entire block is skipped

Fine-grained Sparsity in Ampere GPUs

- Very recently, fine-grained sparsity was added to Tensor Cores on Nvidia GPUs
- 2 elements for every block of 4 elements can be zero
- Requires retraining to regain accuracy
- Overhead?
  - 2 bits per 8-bit element
  - 12.5% memory overhead
  - Control logic? Performance improvement? Power savings?



Source: Nvidia

SFU

QUESTION

What is the performance improvement of 50% fine-grained sparsity on Nvidia GPUs?

1. 2.0 X
2. 1.5 X
3. 1.2 X
4. 0.5 X

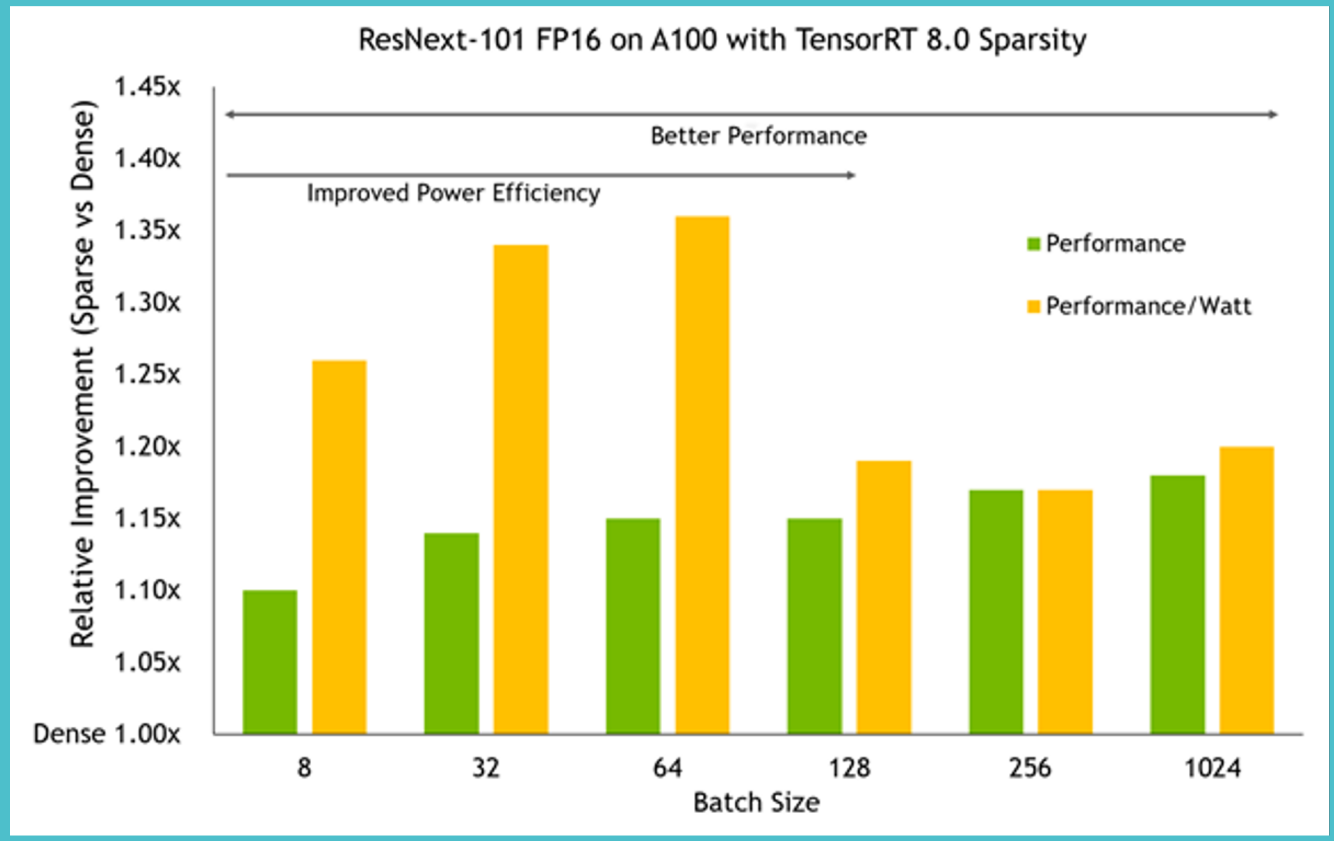
SFU

QUESTION

What is the performance improvement of 50% fine-grained sparsity on Nvidia GPUs?

1. 2.0 X
2. 1.5 X
3. 1.2 X
4. 0.5 X

Even though we skip half the computations, there is overhead to support sparsity, like figuring out where all the zeroes are to be able to skip



SFU

2. Hardware Efficiency

1. Arithmetic
   - Specialized Instructions: To amortize overhead. ✓
   - Lower precision (Quantization) ✓
2. Memory
   - Locality: Move data to inexpensive on-chip memory. ✓
   - Reuse: To avoid expensive memory fetches. ✓
3. Ineffectual Operations
   - Sparsity: Skip useless operations ✓
   - Compressed Sparse Column (CSC) Format

Compressed Sparse Column (CSC) Format

Example

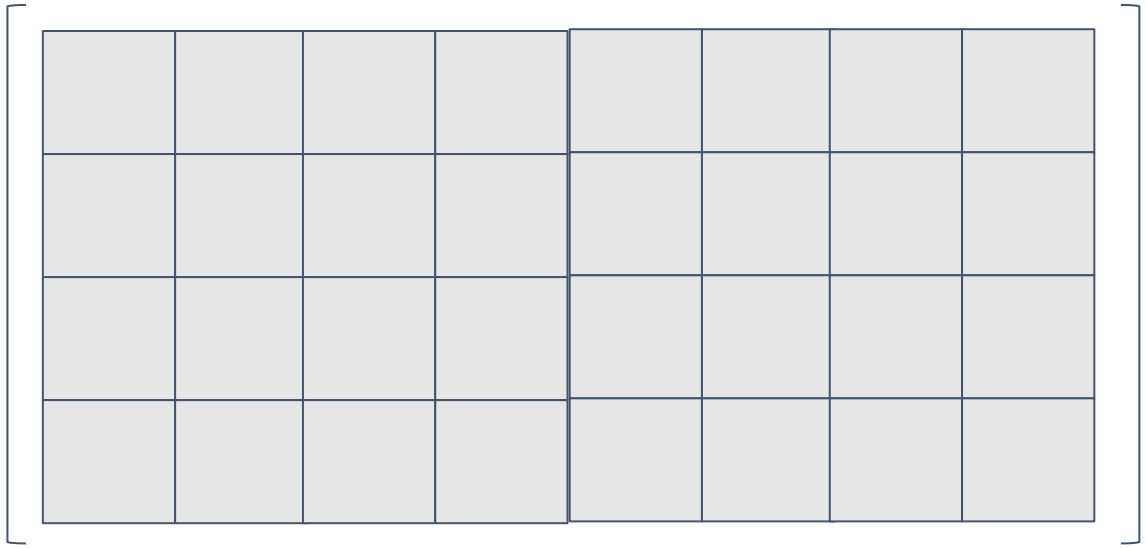
$$[0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\mathbf{0},0,0,3]$$

$$\text{as } v = [1,2,\mathbf{0},3],\ z = [2,0,\mathbf{15},2].\ v \text{ and } z \text{ of all columns}$$

Source: Han et al. EIE: Efficient Inference Engine

Compressed Sparse Column (CSC) Format

Virtual Weight	a	b	c	d	e	f	g	h	i	j	k	l	m
Relative Row Index	0	1	0	1	0	2	0	0	0	2	0	2	0
Column Pointer	0	3	4	6	6	8	10	11	13				



Source: Han et al. EIE: Efficient Inference Engine

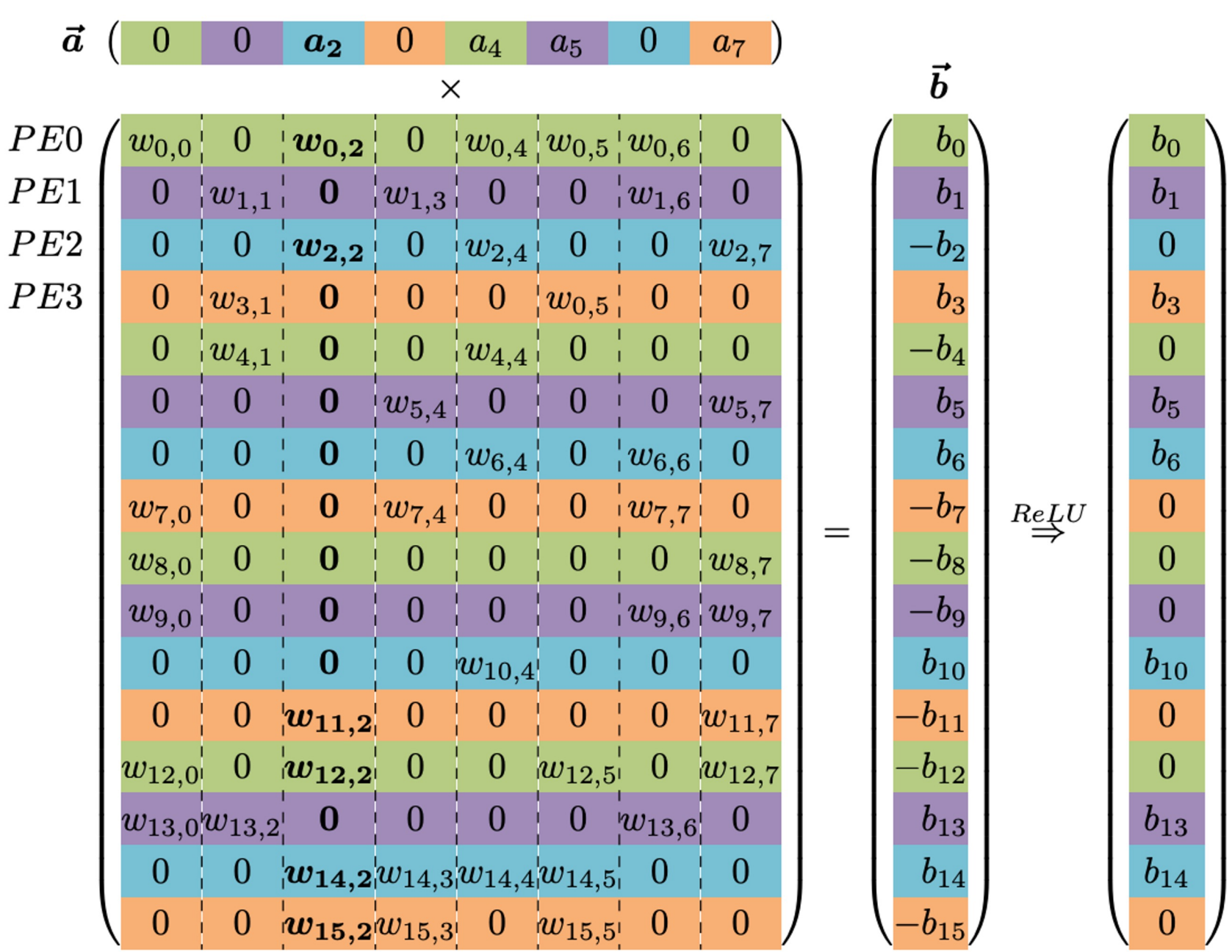
Compressed Sparse Column (CSC) Format

Virtual Weight	a	b	c	d	e	f	g	h	i	j	k	l	m
Relative Row Index	0	1	0	1	0	2	0	0	0	2	0	2	0
Column Pointer	0	3	4	6	6	8	10	11	13				

a	0	e	0	g	i	k	0
0	d	0	0	h	0	0	0
b	0	0	0	0	0	0	l
c	0	f	0	0	j	0	m

Source: Han et al. EIE: Efficient Inference Engine

Interleaving



SFU

2. Hardware Efficiency

1. Arithmetic
   - Specialized Instructions: To amortize overhead. ✓
   - Lower precision (Quantization) ✓
2. Memory
   - Locality: Move data to inexpensive on-chip memory. ✓
   - Reuse: To avoid expensive memory fetches. ✓
3. Ineffectual Operations
   - Sparsity: Skip useless operations ✓
   - Compressed Sparse Column (CSC) Format ✓

SFU

Guidelines Domain-Specific Architectures (DSAs)

1. Use dedicated memory to minimize the distance over which data is moved.
2. Invest the resources saved from dropping advanced microarchitectural optimizations into more arithmetic units or bigger memories.
3. Use the easiest form of parallelism that matches the domain.
4. Reduce data size and type to the simplest needed for the domain.
5. Use a domain-specific programming language to port code to the DSA.

Source: Computer Architecture: A Quantitative Approach, Chapter 7

SFU

Putting it all together