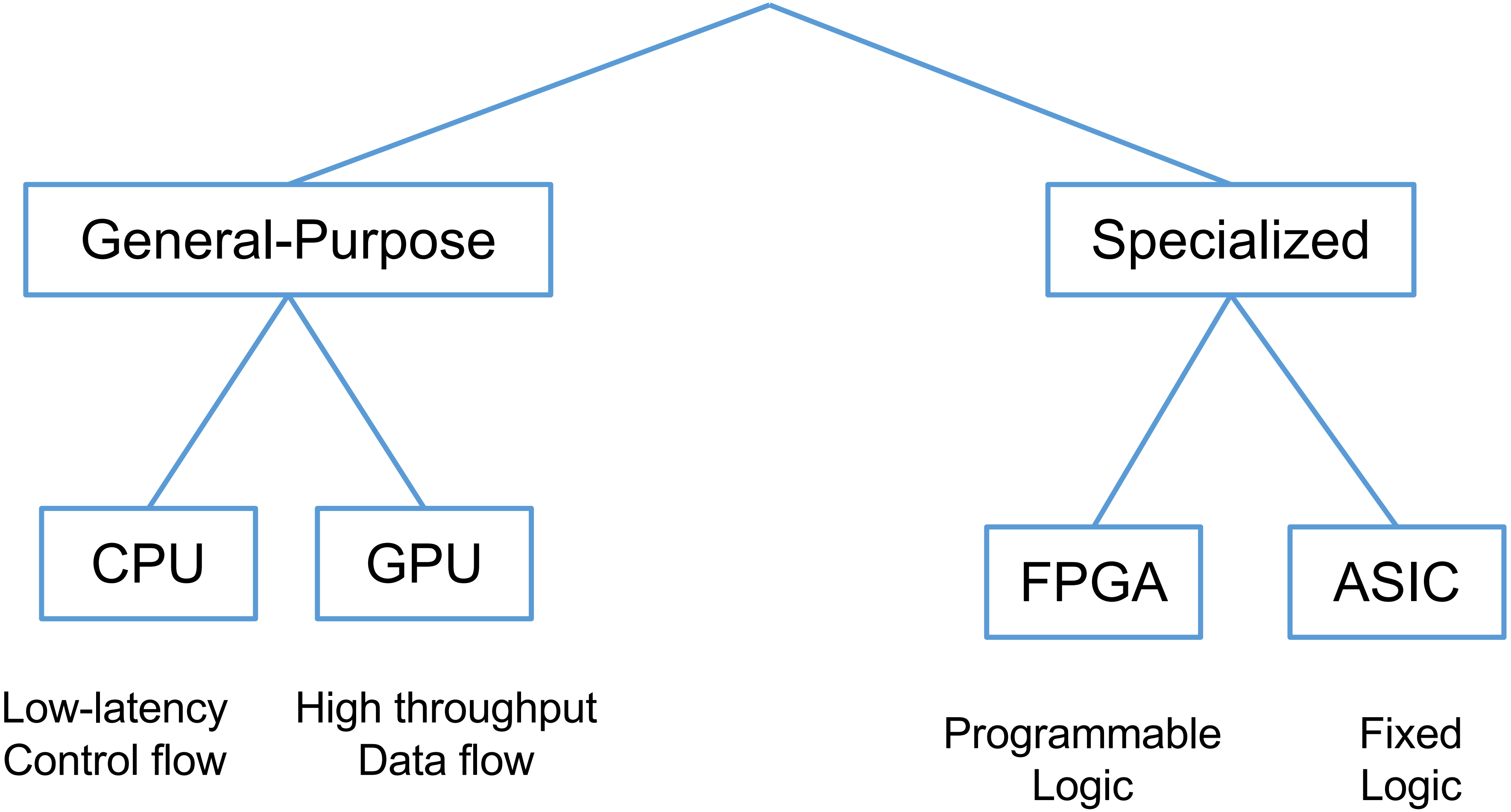
CMPT 450/750: Computer Architecture
Fall 2024
Domain-Specific Architecture I
How did we get here ?
What are they ?

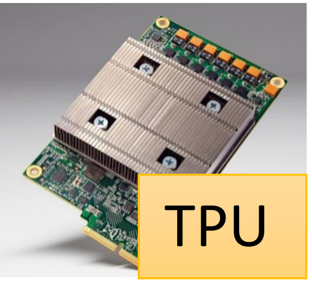
Alaa Alameldeen & Arrvindh Shriraman

Hardware Types

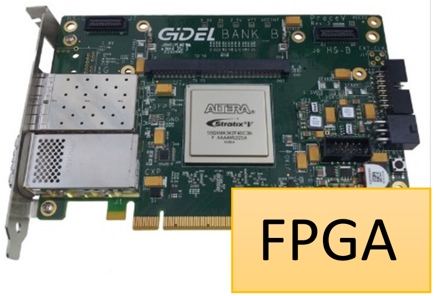


Adapted from Song Han "Efficient Methods and Hardware for Deep Learning" Stanford 2017

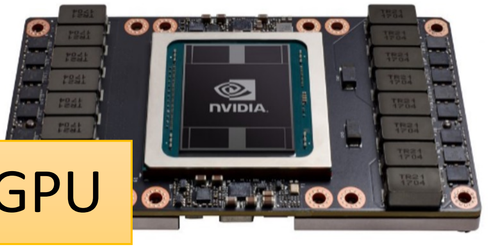
Specialized Hardware



TPU

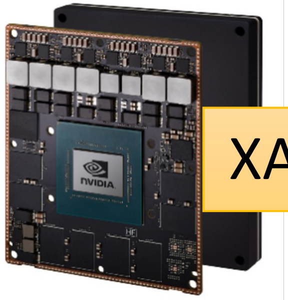


FPGA

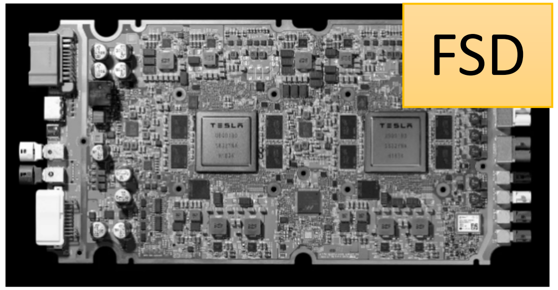


GPU

NPU



XAVIER



FSD

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

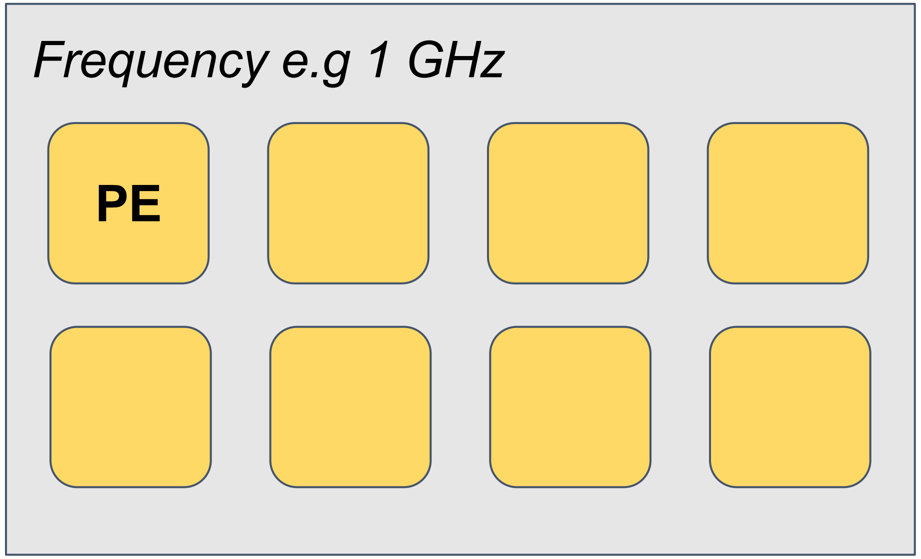
Hardware Metrics & Roofline

Compute Performance Metrics

FLOPs/s

Floating-Point Operations per second

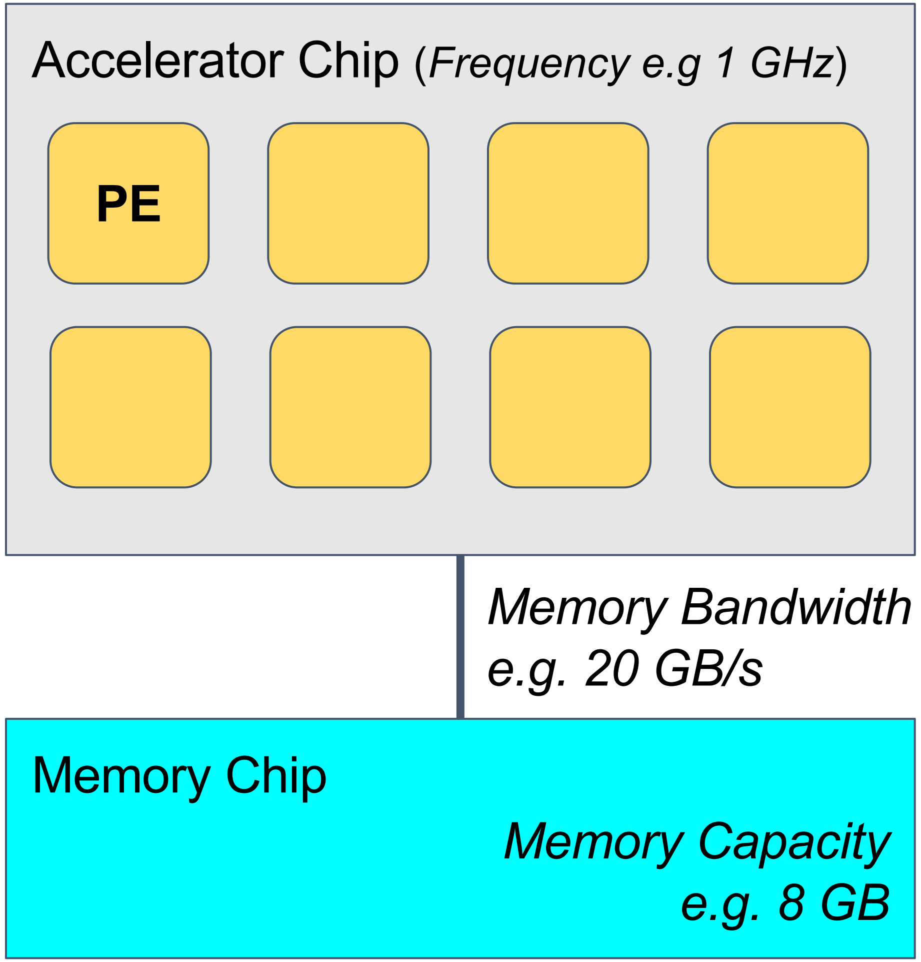
- MACs/s: Multiply-accumulate Ops/s
  - Half FLOPs/s
- OPs/s: for non floating-point operations
- Chips are often labeled with “peak FLOPs/s”
  - Not achievable under normal workloads
  - <u>Very rough</u> indication of performance



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

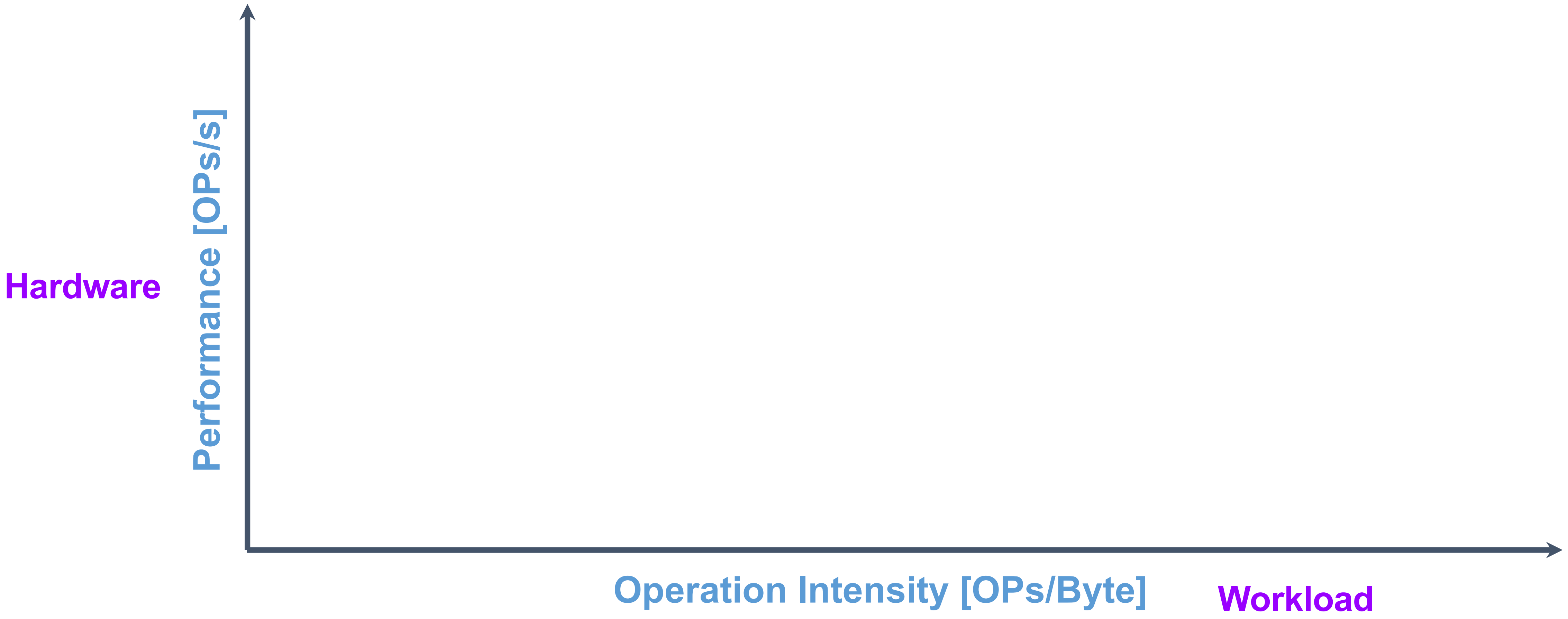
Memory Performance Metrics

- 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



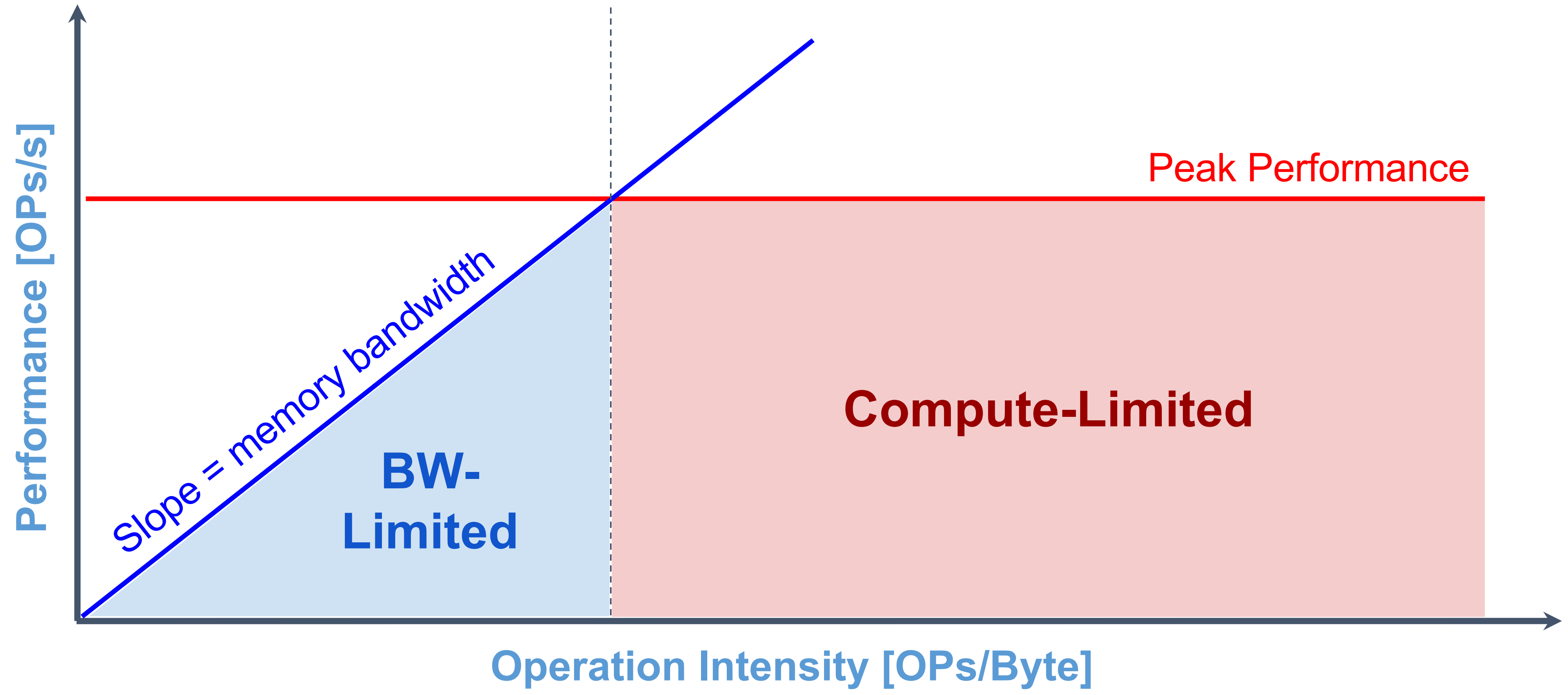
Roofline Plot

Characterize the performance of a given hardware device across different workloads



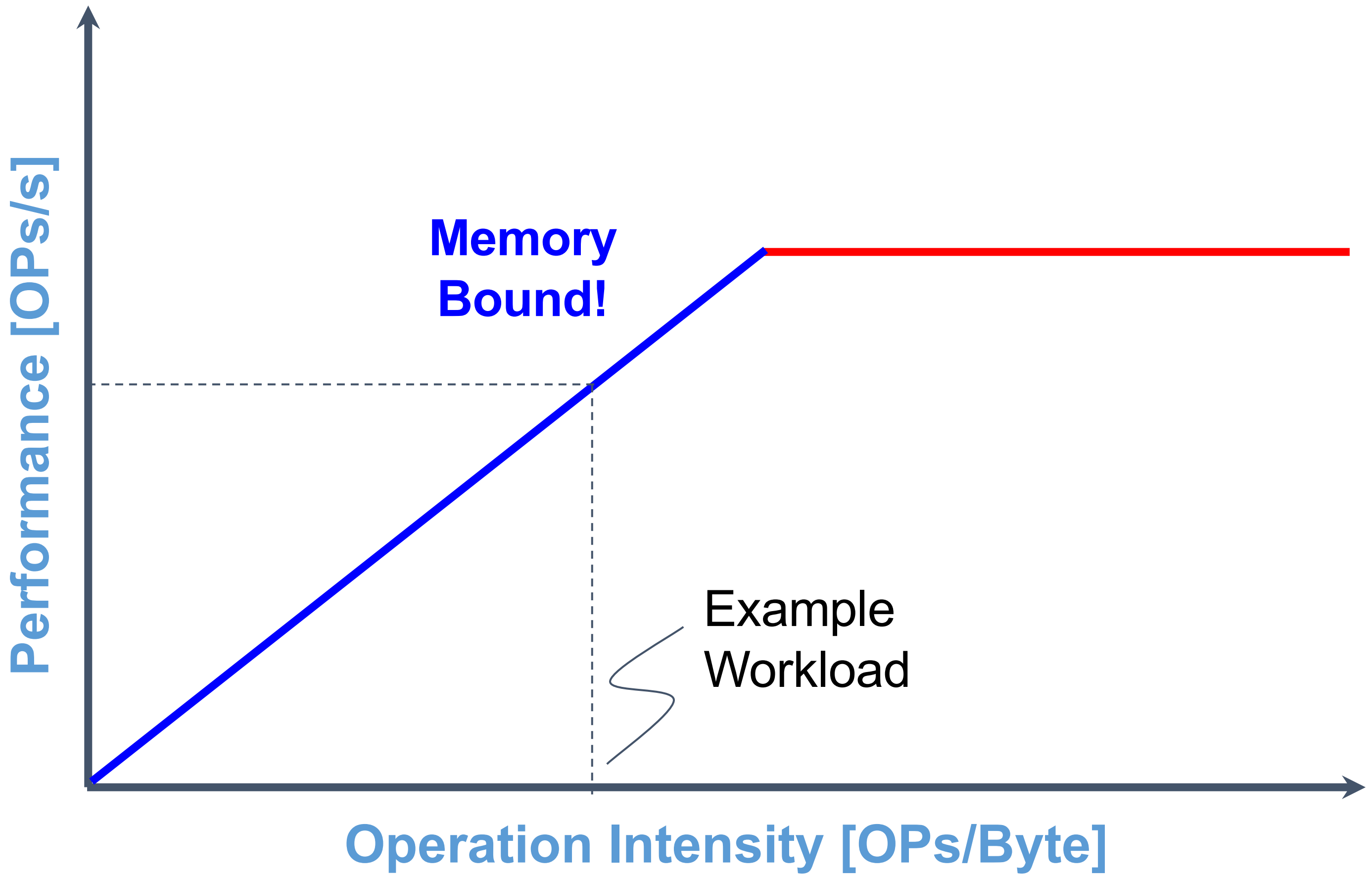
Roofline Plot

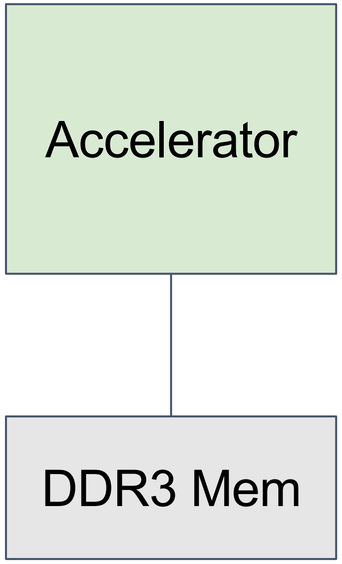
Characterize the performance of a given hardware device across different workloads



Roofline Plot

Characterize the performance of a given hardware device across different workloads





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

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

QUESTION

How can you speed up a memory-bound application?

1. Use a larger memory chip
2. Use a faster memory chip
3. Add more multipliers
4. Use lower numerical precision

SFU

QUESTION

How can you speed up a memory-bound application?

1. ~~Use a larger memory chip~~
2. Use a faster memory chip

Gets data on chip faster

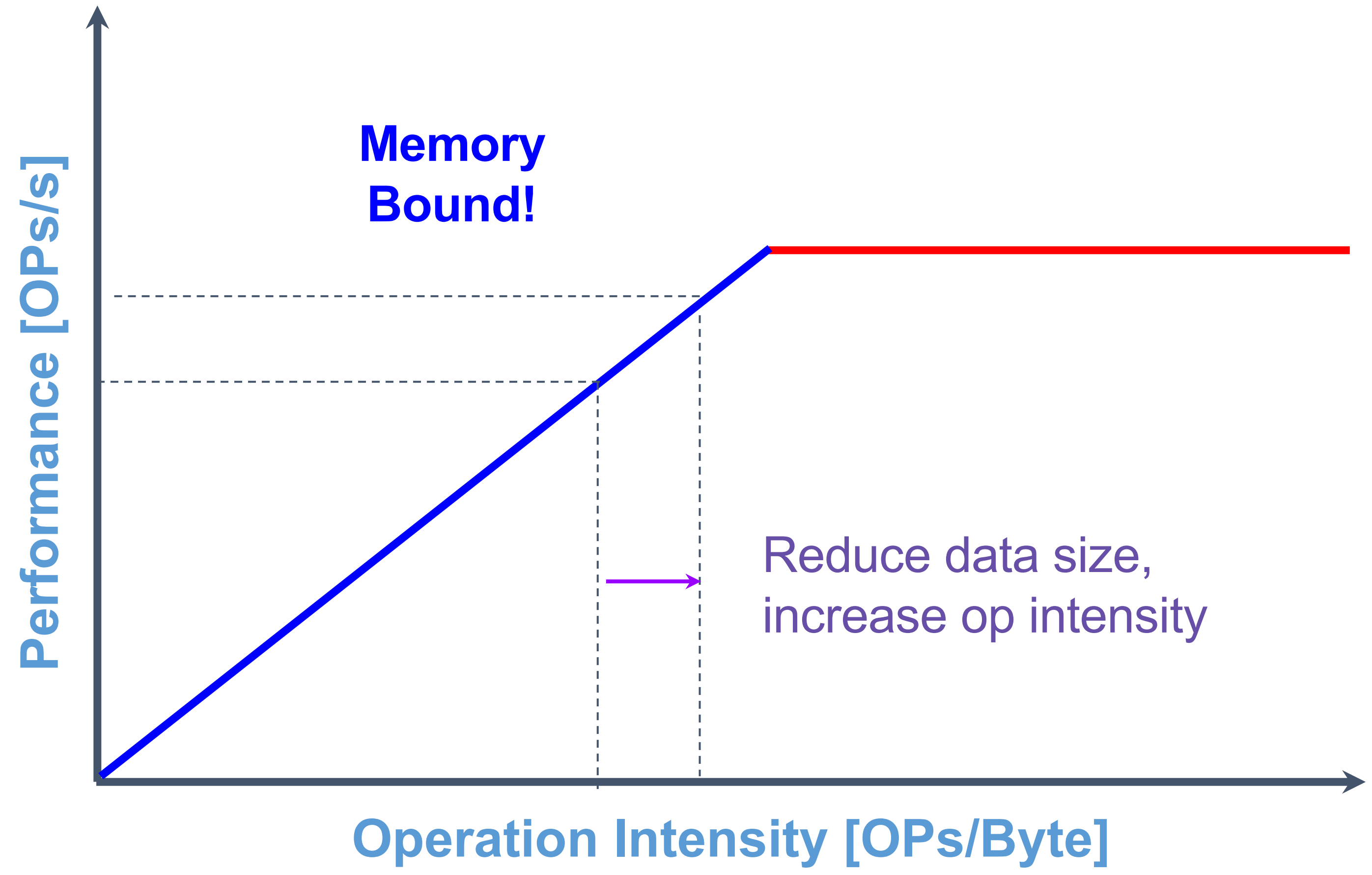
3. ~~Add more multipliers~~
4. Use lower numerical precision

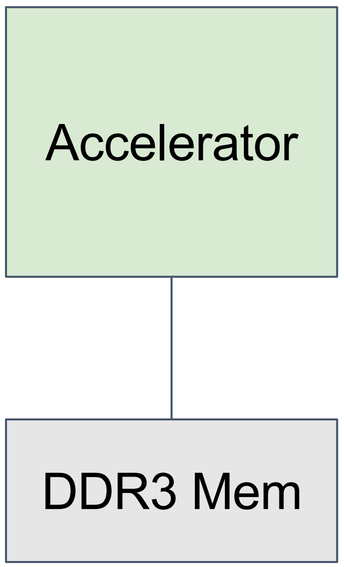
Data becomes smaller, so transport is faster

?

Roofline Plot

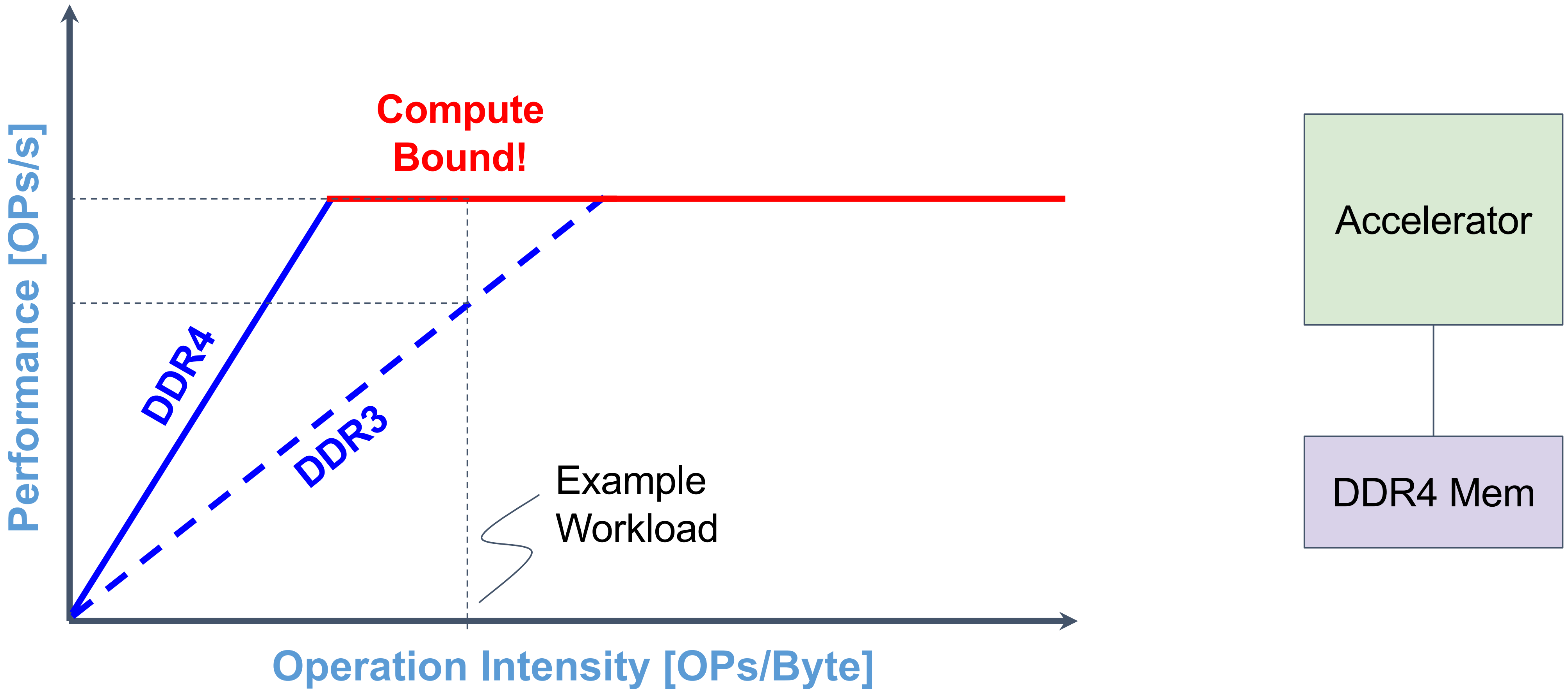
Compressed data format e.g. reduced precision





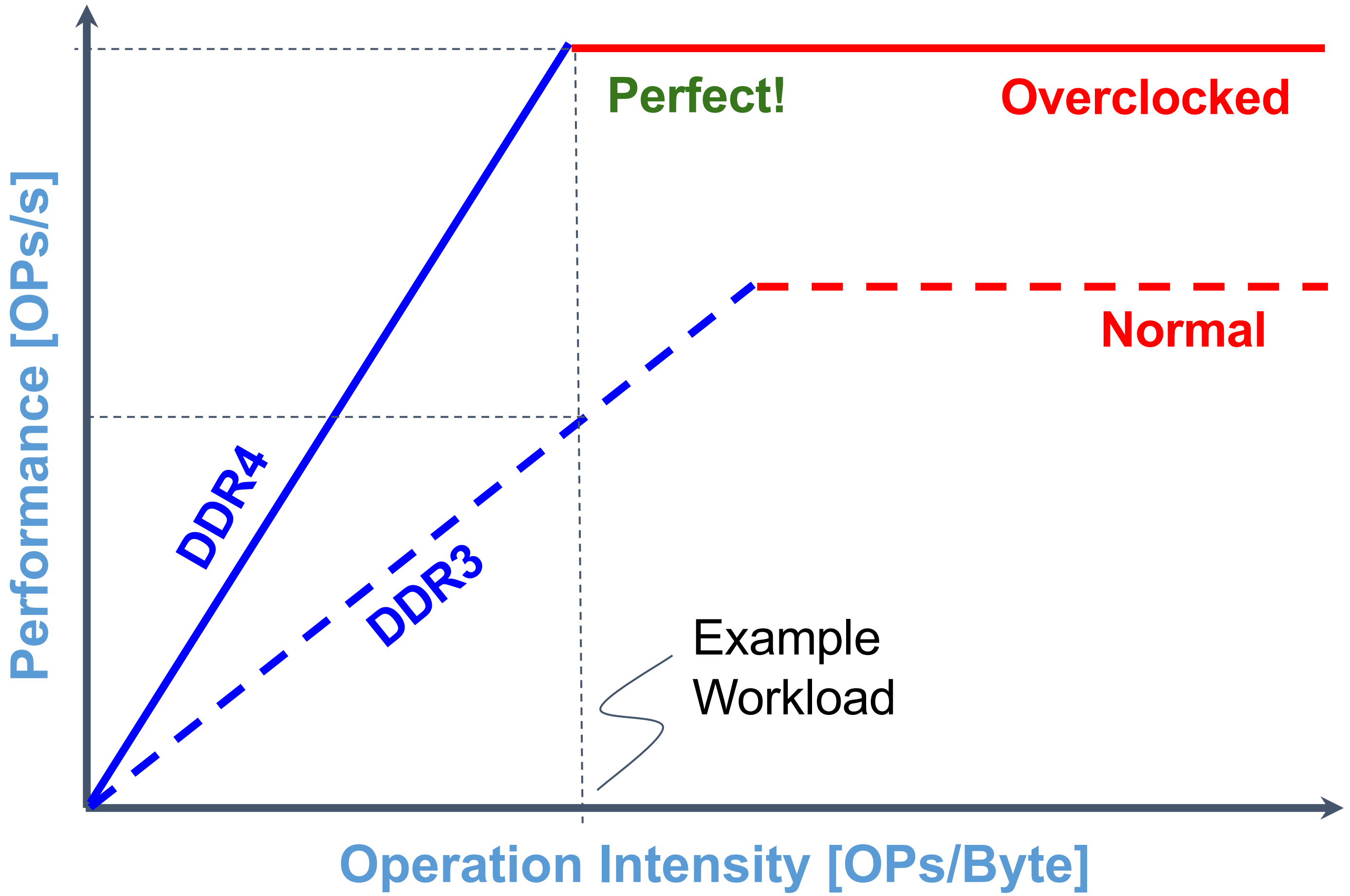
Roofline Plot

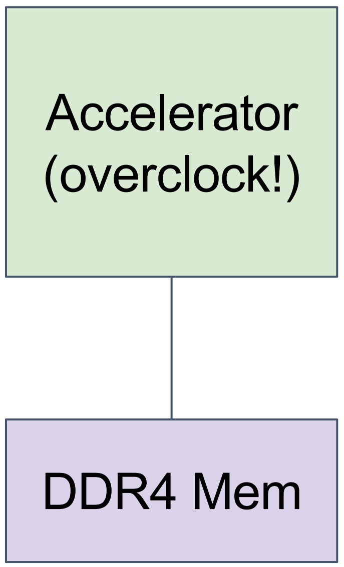
Faster memory chip increases slope of roofline



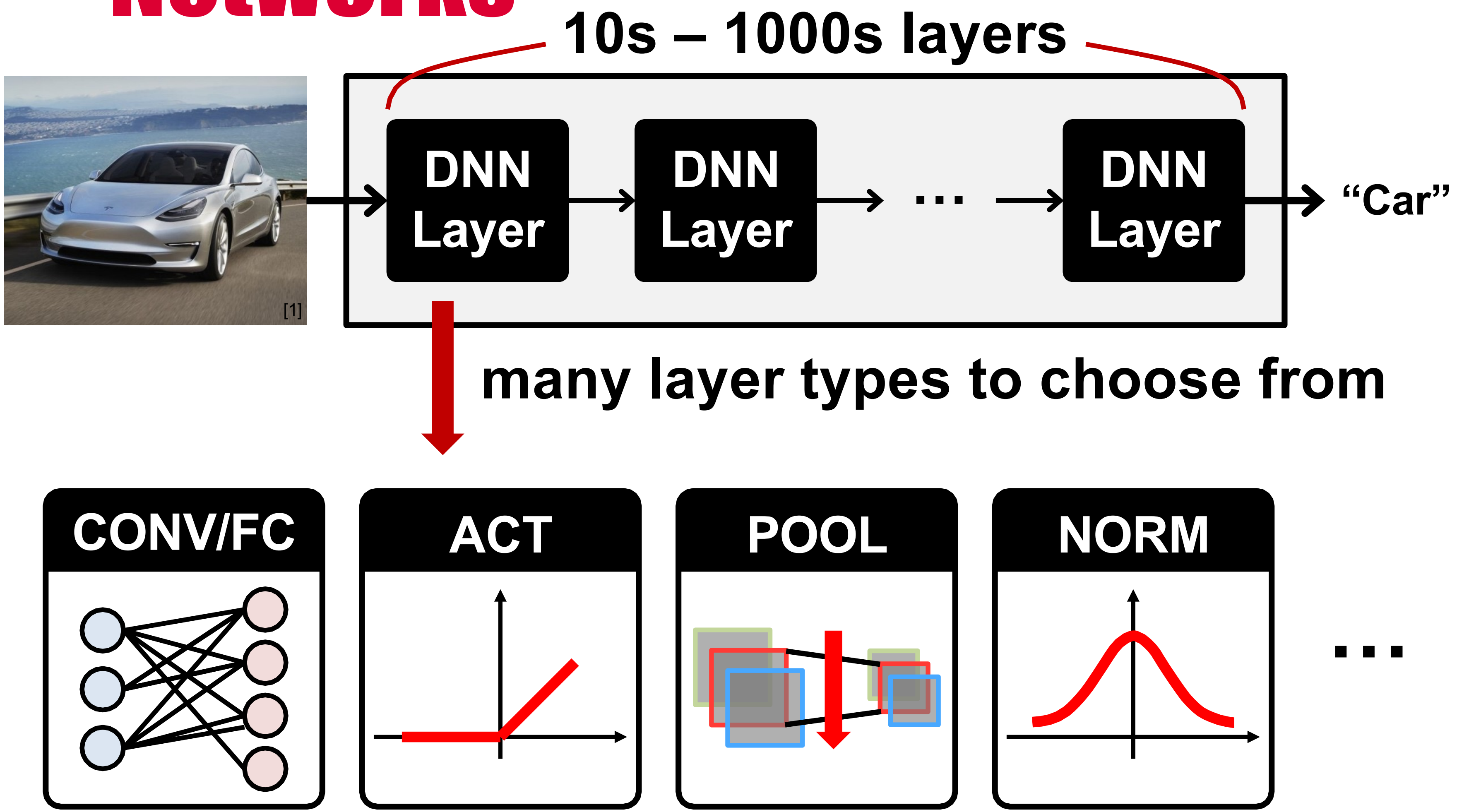
Roofline Plot

Raise the roofline by increasing the speed/throughput of compute





Primer on Deep Neural Networks

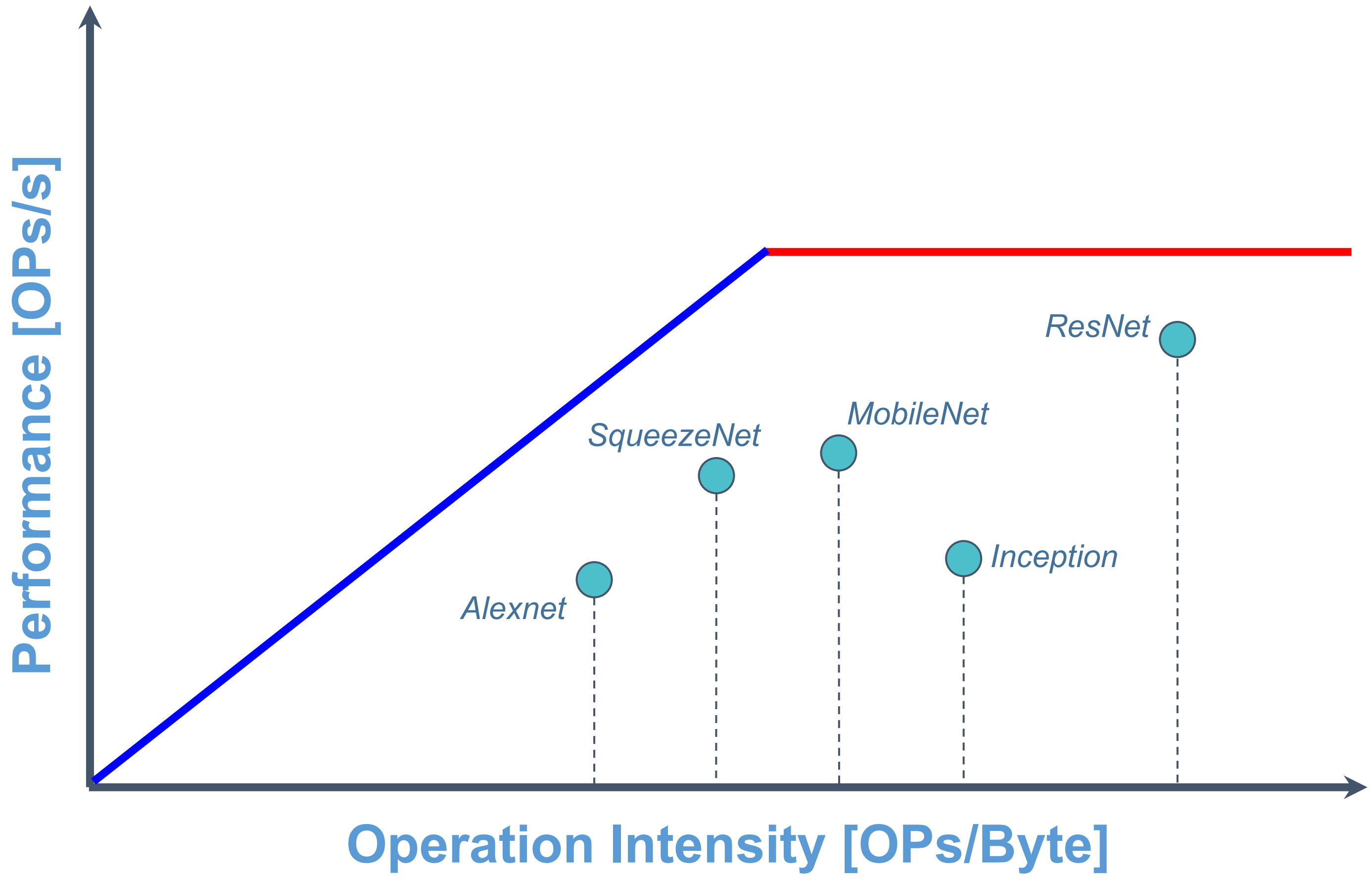


[1] Tesla

SFU

Roofline Example

Measured performance is (by definition) below the roofline.

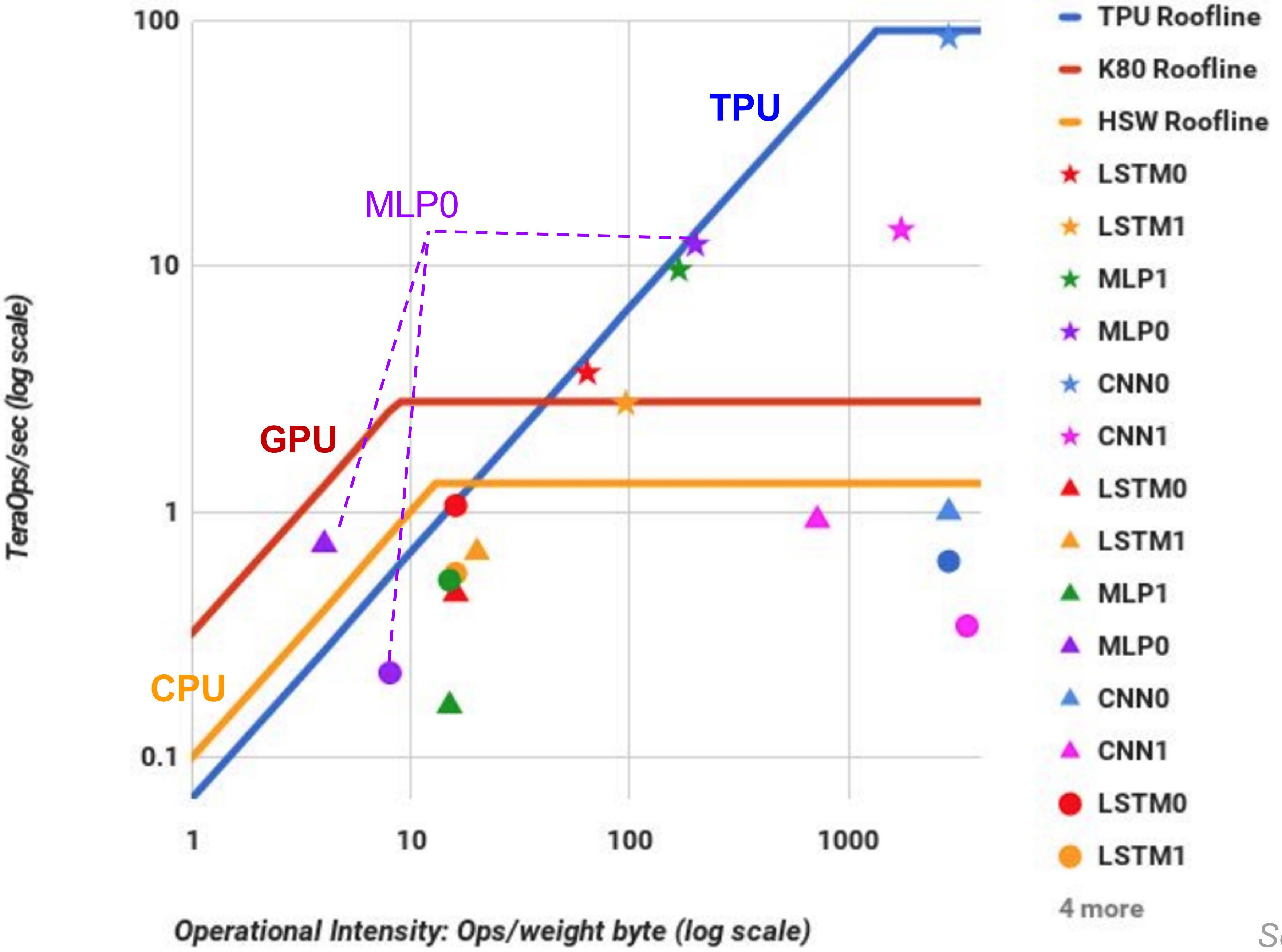


Achieved Performance can be limited by:

- 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

SFU

QUESTION

How can the same DNN have a different operational intensity on different hardware?

1. Different supported numerical precisions on each device
2. Different memory bandwidths on each device
3. Different number of PEs on each device
4. Different on-chip memory hierarchy on each device

?

SFU

QUESTION

How can the same DNN have a different operational intensity on different hardware?

1. Different supported numerical precisions on each device
2. ~~Different memory bandwidths on each device~~
3. ~~Different number of PEs on each device~~
4. ~~Different on-chip memory hierarchy on each device~~

Size of data affects ops/byte

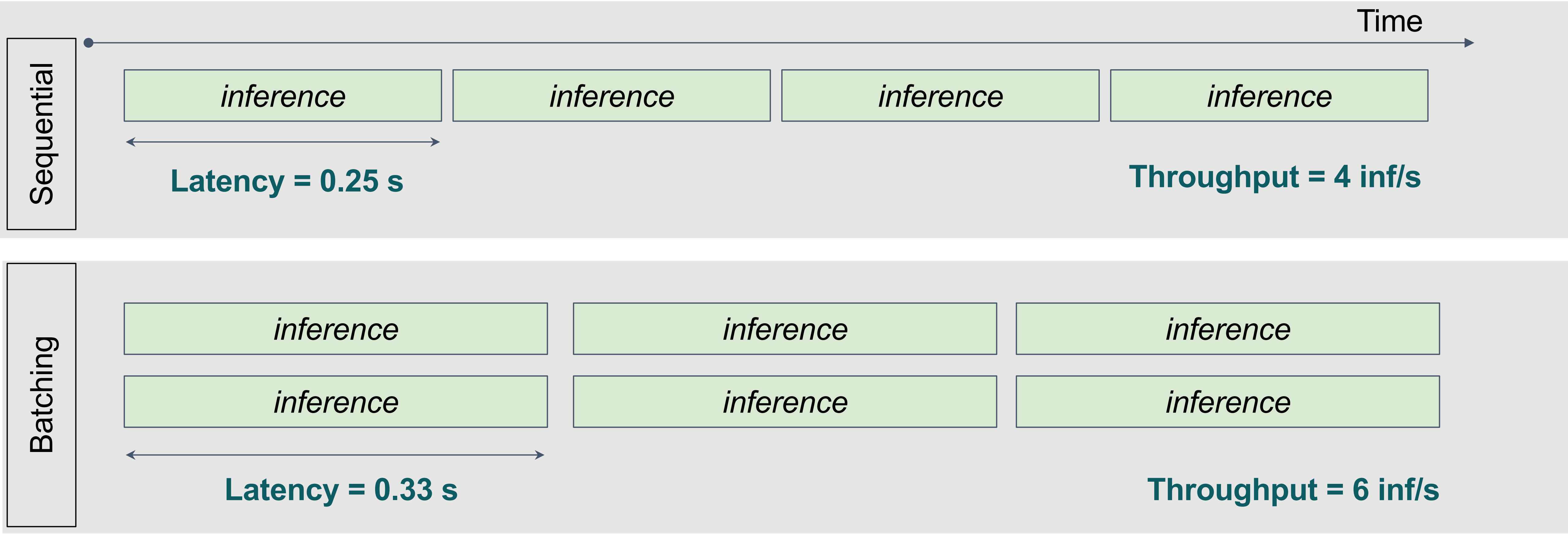
?

Metrics Summary (so far)

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

Throughput and Latency

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



2. Hardware Efficiency

SFU

Information lost necessitating more complex hardware

PYTHON

```
np.add(arr1, arr2)
```

C/C++

```
for(i = 0;i < n;i++)
 res[i] = arr1[i] + arr2[i]
```

ISA

```
.Loop:
 lw     a5, 0(a2)        # *(arr1+i)
 lw     a6, 0(a3)        # *(arr2+i)
 add    a0, a5, a6
 sw     a0, 0(a4)
# Bump pointers.
   addi   a2, a2, 4
   addi   a3, a3, 4
   addi   a4, a4, 4
   addi   a1, a1, 1
   bne    a1, a3, loop
```

SFU

Information lost necessitating more complex hardware

PYTHON

```
np.add(arr1, arr2)
```

C/C++

```
for(i = 0;i < n;i++)
 res[i] = arr1[i] + arr2[i]
```

ISA

```
.Loop:
lw      a5, 0(a2)        # *(arr1+i)
lw      a6, 0(a3)        # *(arr2+i)
 add     a0, a5, a6
sw      a0, 0(a4)
# Bump pointers.
   addi    a2, a2, 4
   addi    a3, a3, 4
   addi    a4, a4, 4
   addi    a1, a1, 1
   bne     a1, a3, loop
```

Global reg

Load/Store Queues

Branch Predictor to find loop paralleism

Why ISAs suck ?

SFU

Register naming introduced dependencies

Need register renaming hardware

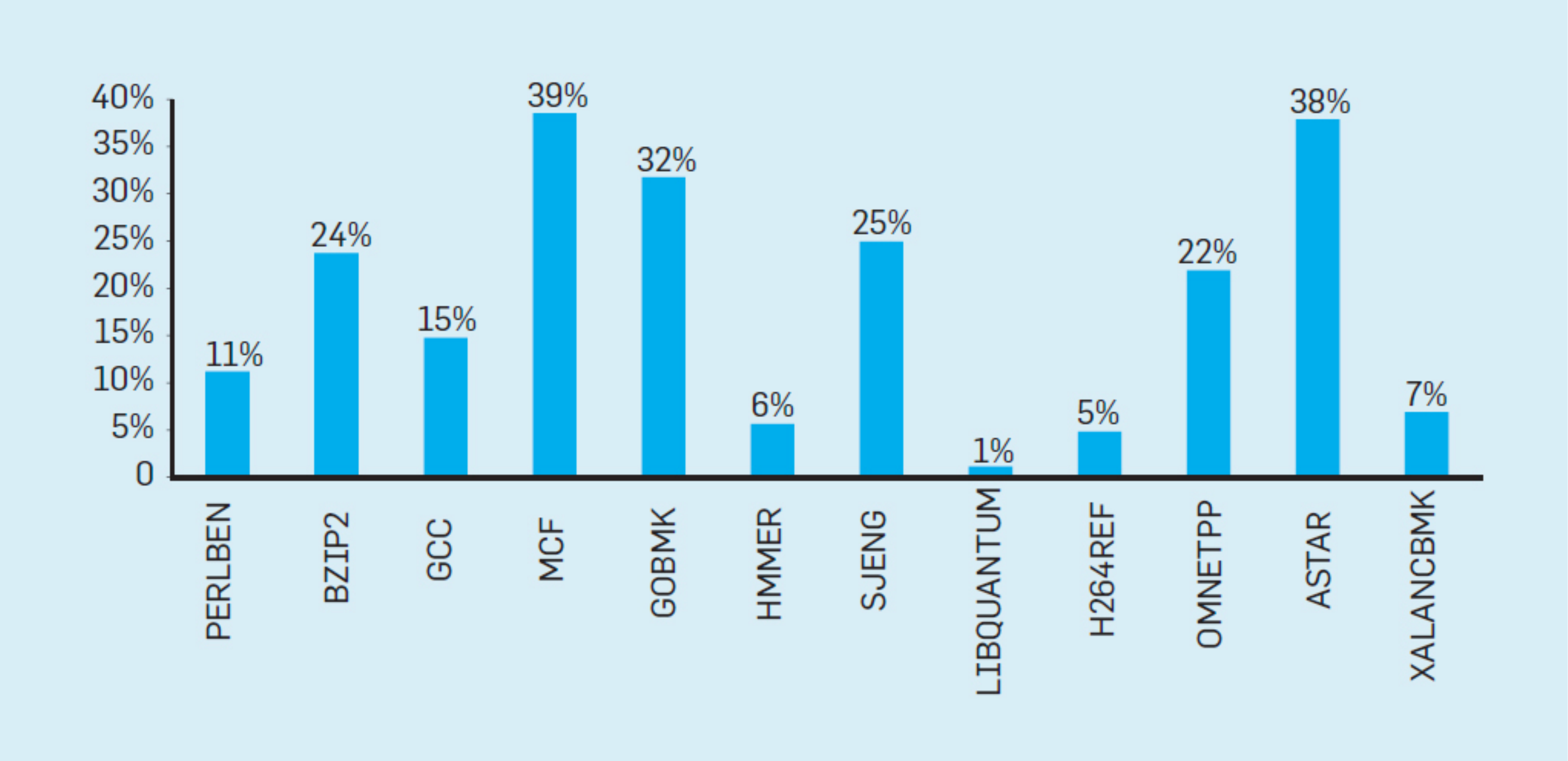
```
#pragma clang unroll_count(10)
for(int i = 0;i < 10;i++)
res[i] = arr1[i] + arr2[i];
}
```

```
res[0] = arr1[0] + arr2[0];
res[1] = arr1[1] + arr2[1];
…..
res[9] = arr1[9] + arr2[9];
```

```
lw  a6, 0(a0)
lw  a4, 0(a1)
lw  a5, 4(a0)
lw  a3, 4(a1)
add a4, a4, a6
sw  a4, 0(a2)
add a6, a3, a5
lw  a7, 8(a0)
lw  a5, 8(a1)
lw  a3, 12(a0)
lw  a4, 12(a1)
sw  a6, 4(a2)
add a5, a5, a7
sw  a5, 8(a2)
add a6, a4, a3
lw  a7, 16(a0)
lw  a5, 16(a1)
lw  a3, 20(a0)
lw  a4, 20(a1)
```

Wasted instructions

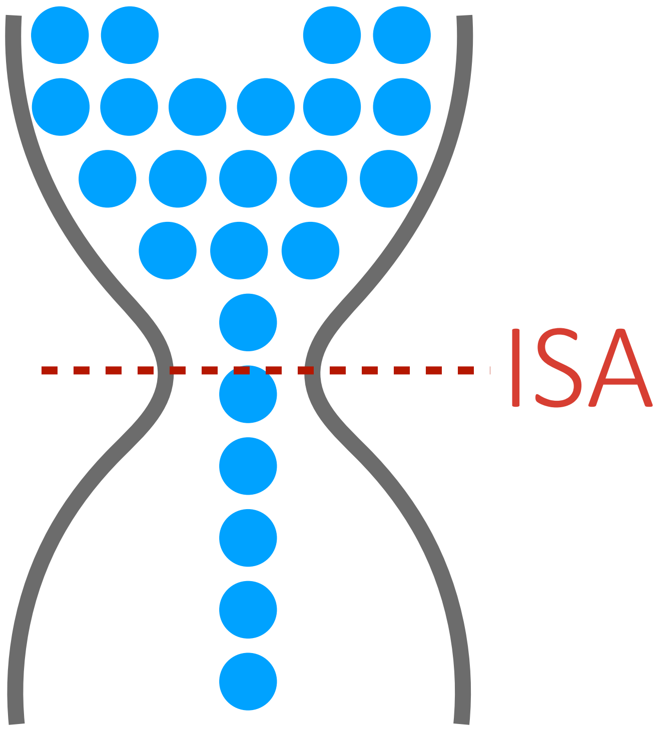
SFU



https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

Why ISAs suck ?

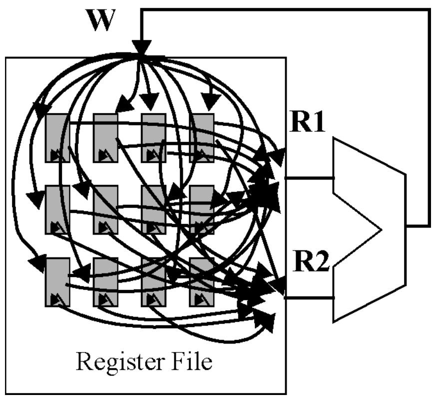
SFU



Why OOOs suck.

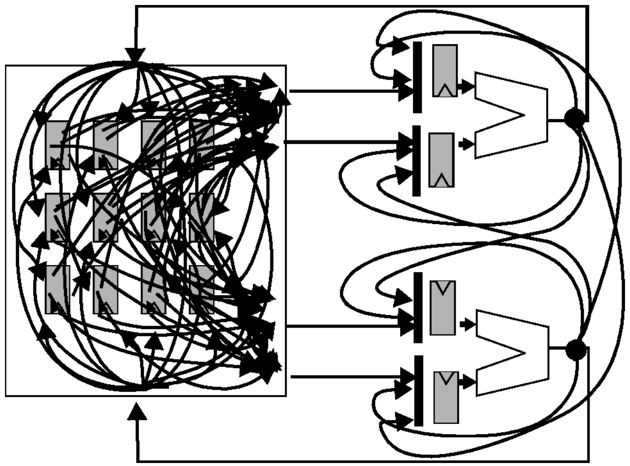
Is technology scaling dead/dying ?

Are DSAs/Accelerators The Solution?



mul $2,$3,$4

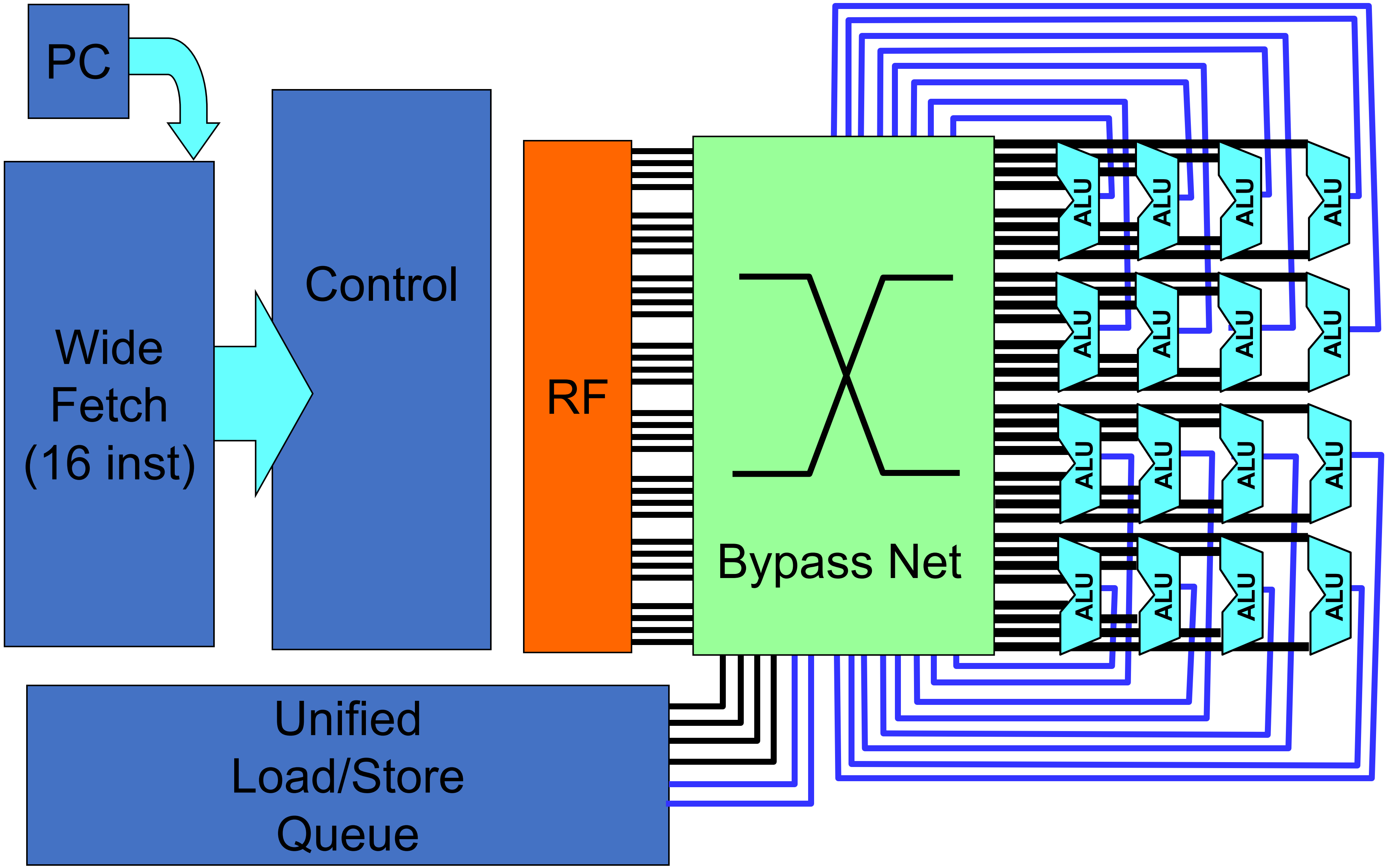
add $6,$5,$2



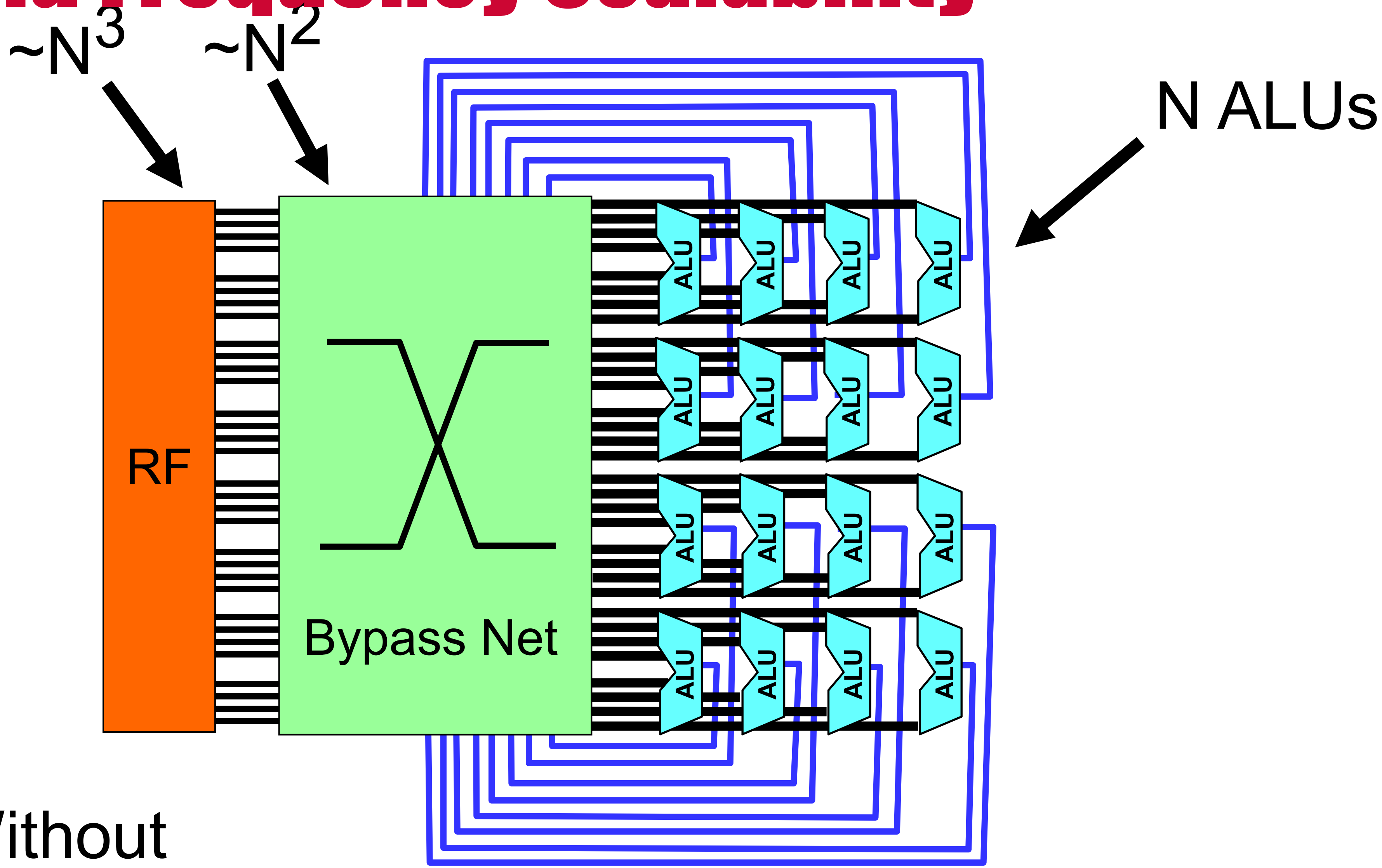
What's great about superscalar microprocessors? →

Fast low-latency tightly-coupled networks

(0-1 cycles of latency, no occupancy)


PC
Wide
Fetch
(16 inst)
Control
RF
Bypass Net
ALU
Unified
Load/Store
Queue

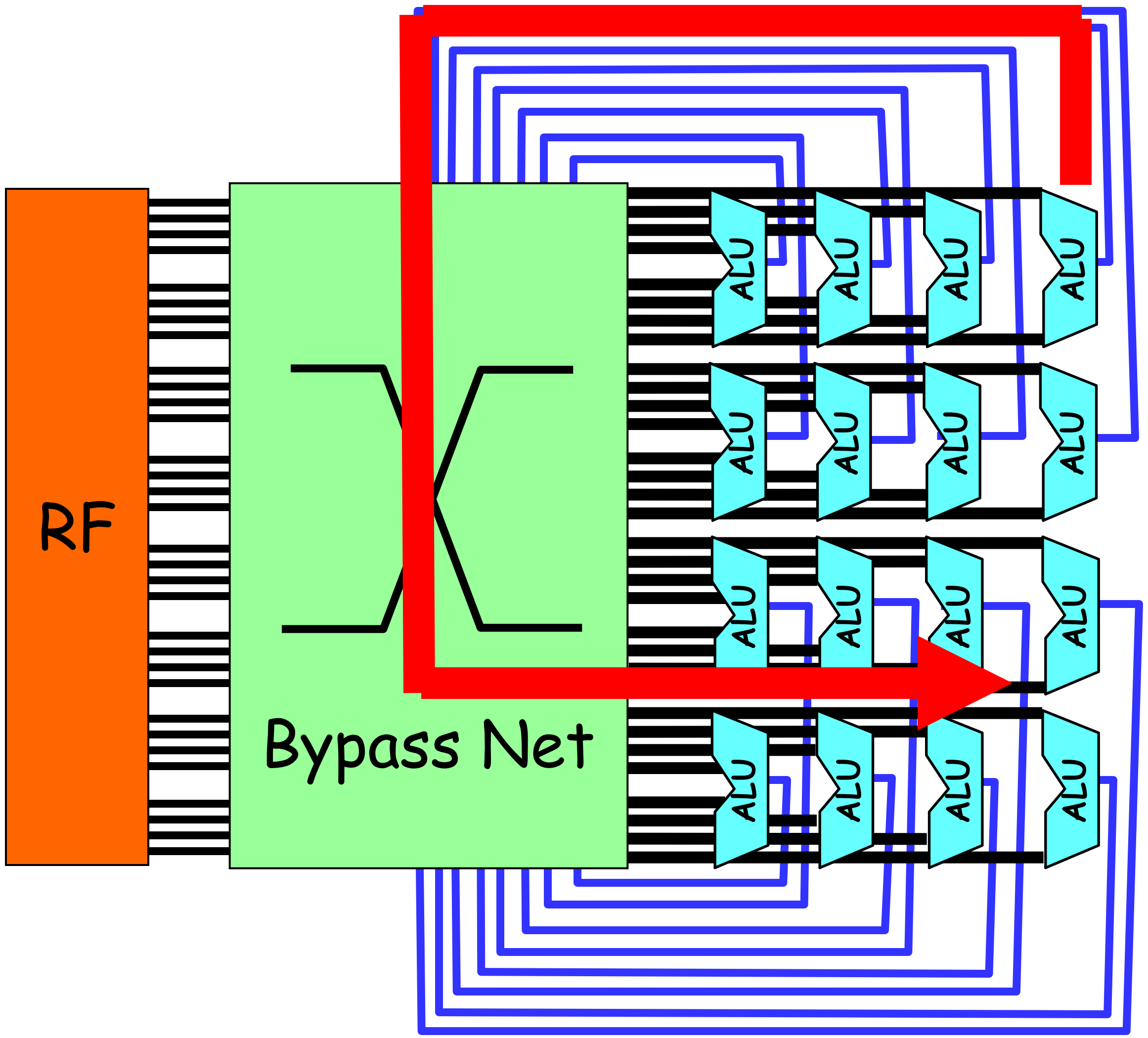
Area and Frequency Scalability

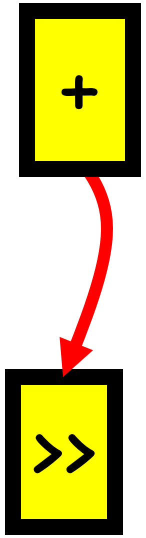


Without modification, freq decreases linearly or worse.

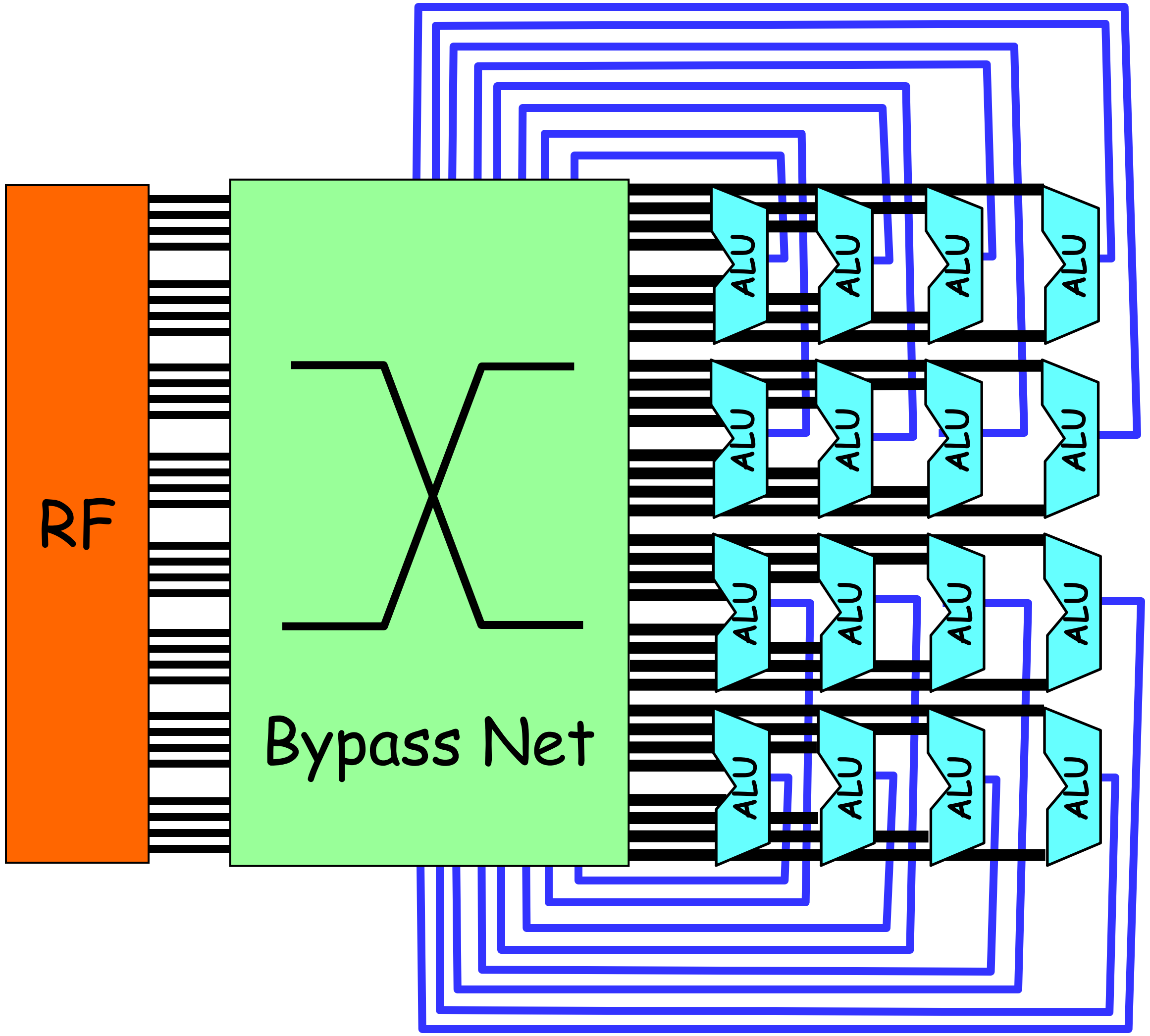
Global Operand Routing

SFU



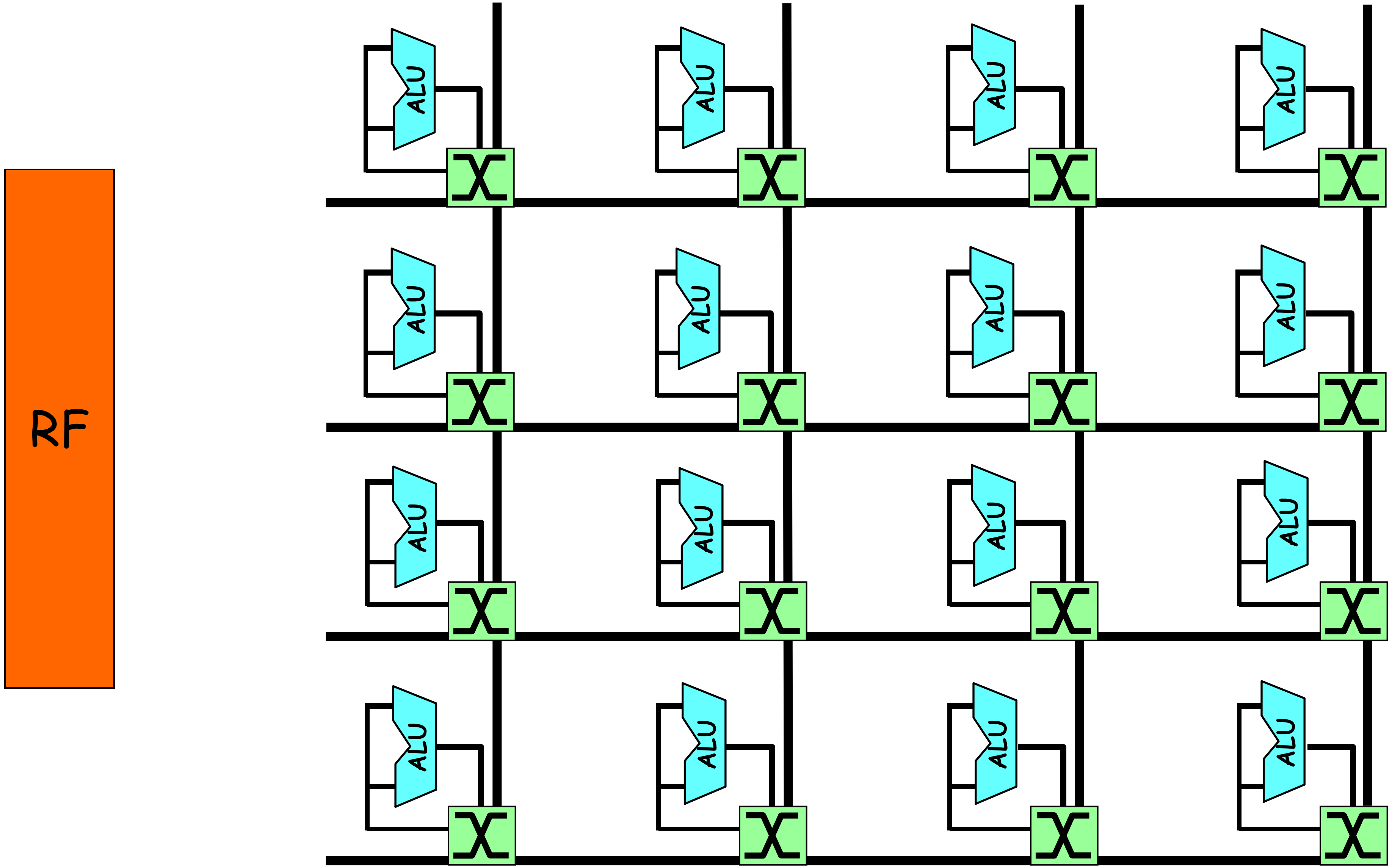


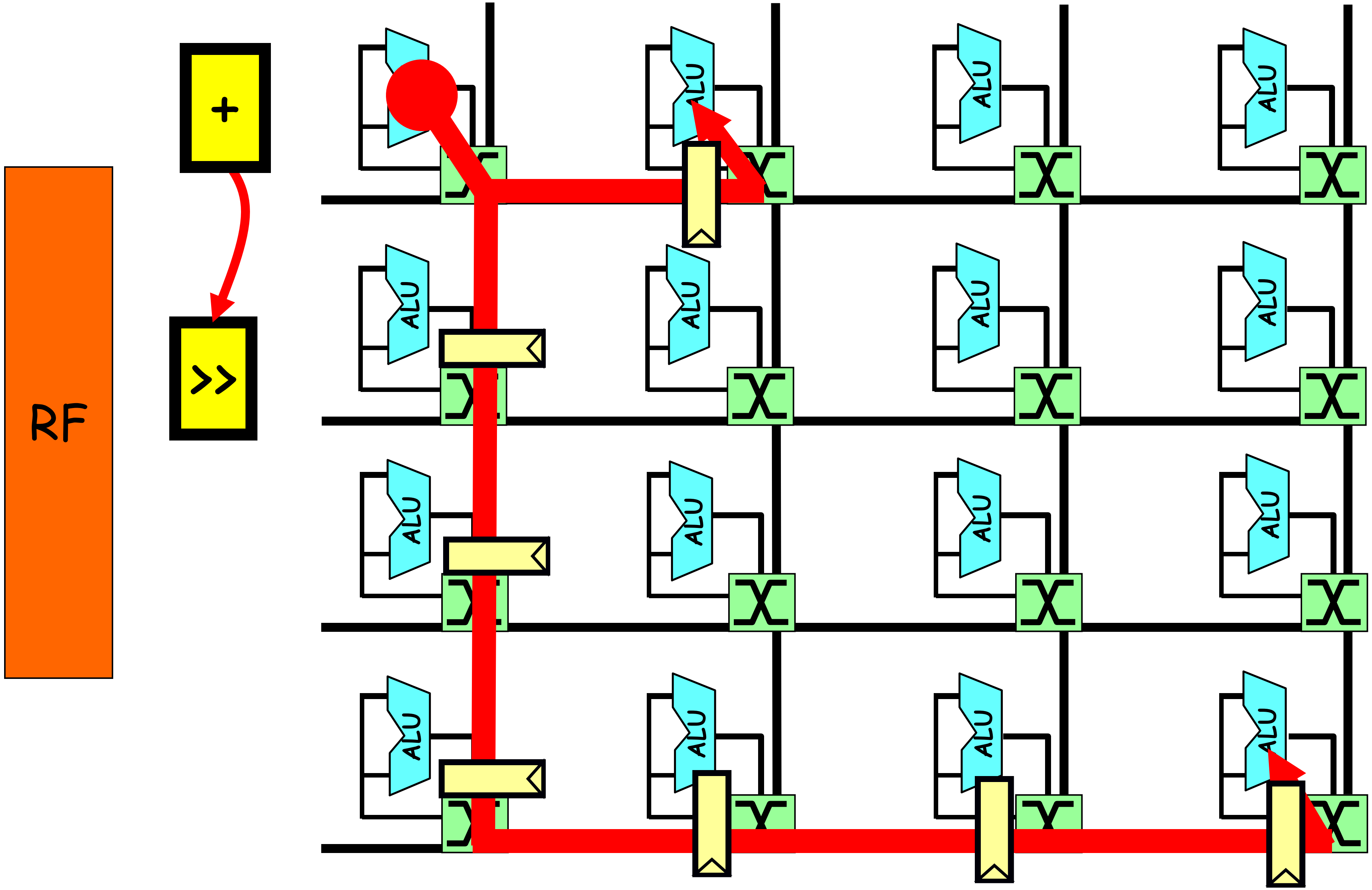
Idea 1 : Make operand routing local



Idea 1: Make operand routing local

SFU




RF
+
>>
ALU

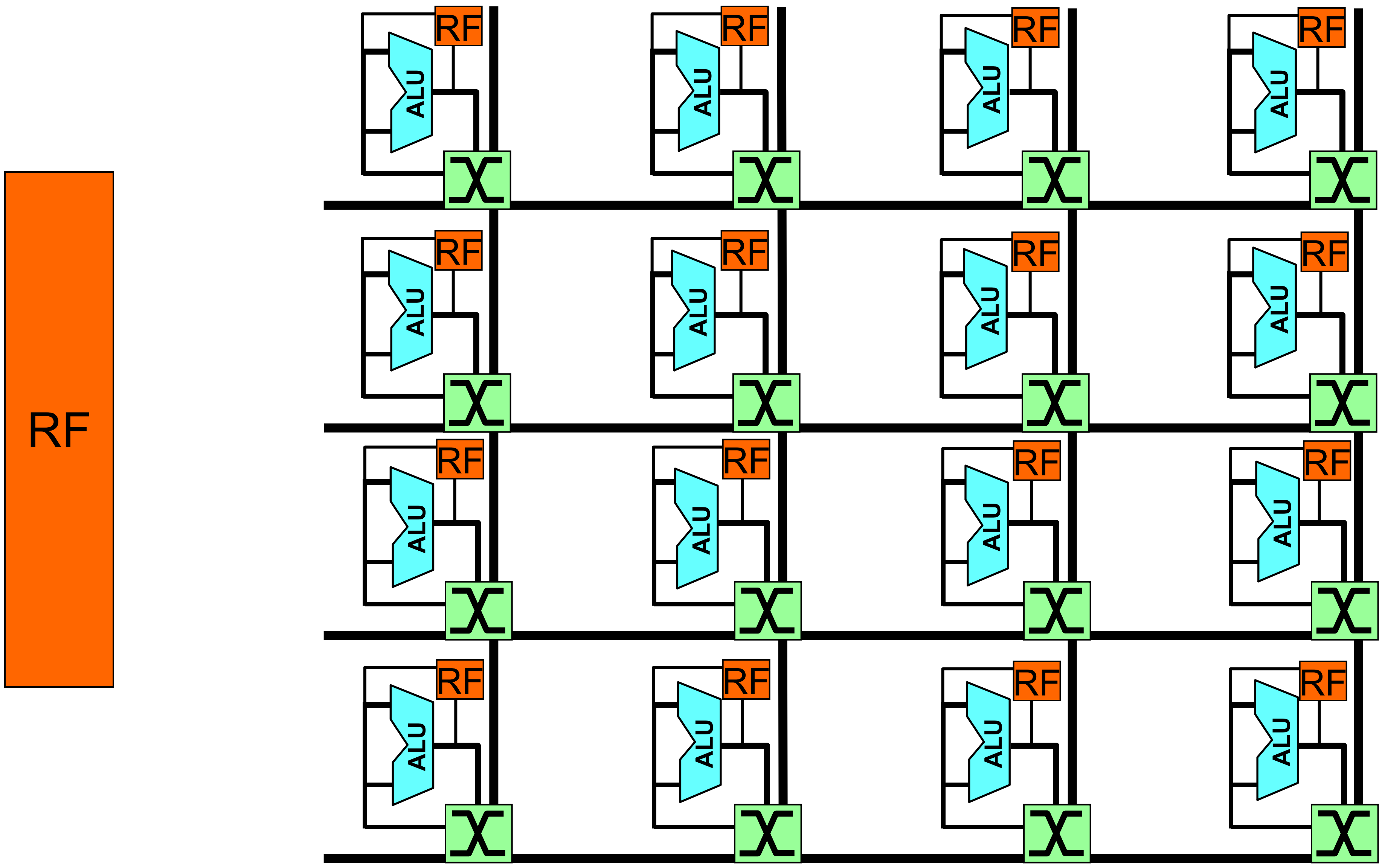
Operand Latency

Time for operand to travel between instructions mapped to different ALUs.

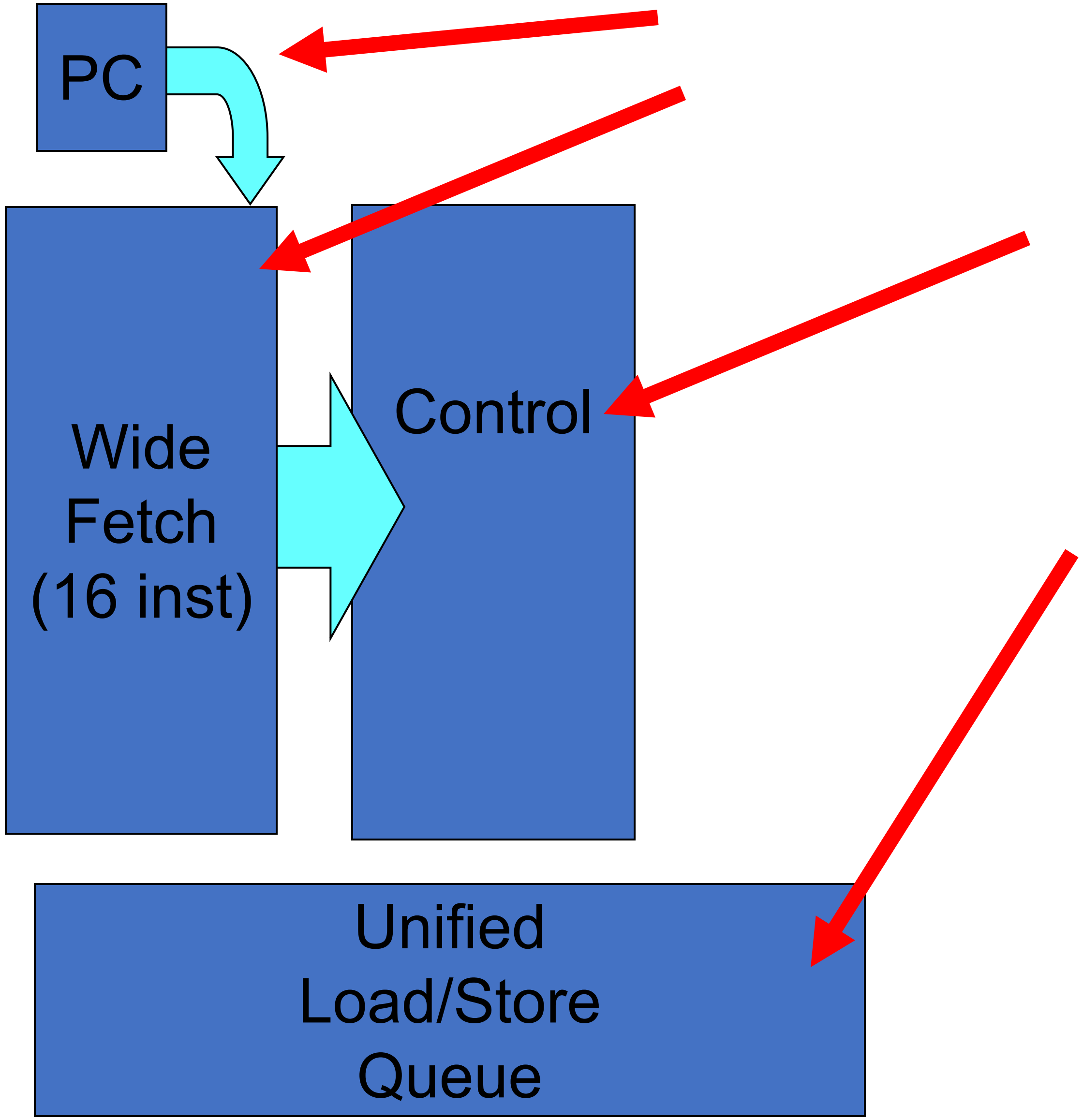
	Un-pipelined crossbar	Point-to-Point Routed Mesh Network
Non-local Placement	~ N	~ $N^{1/2}$
Locality-Driven Placement	~ N	~ 1

Latency bonus if we map communicating instructions nearby so communication is local.

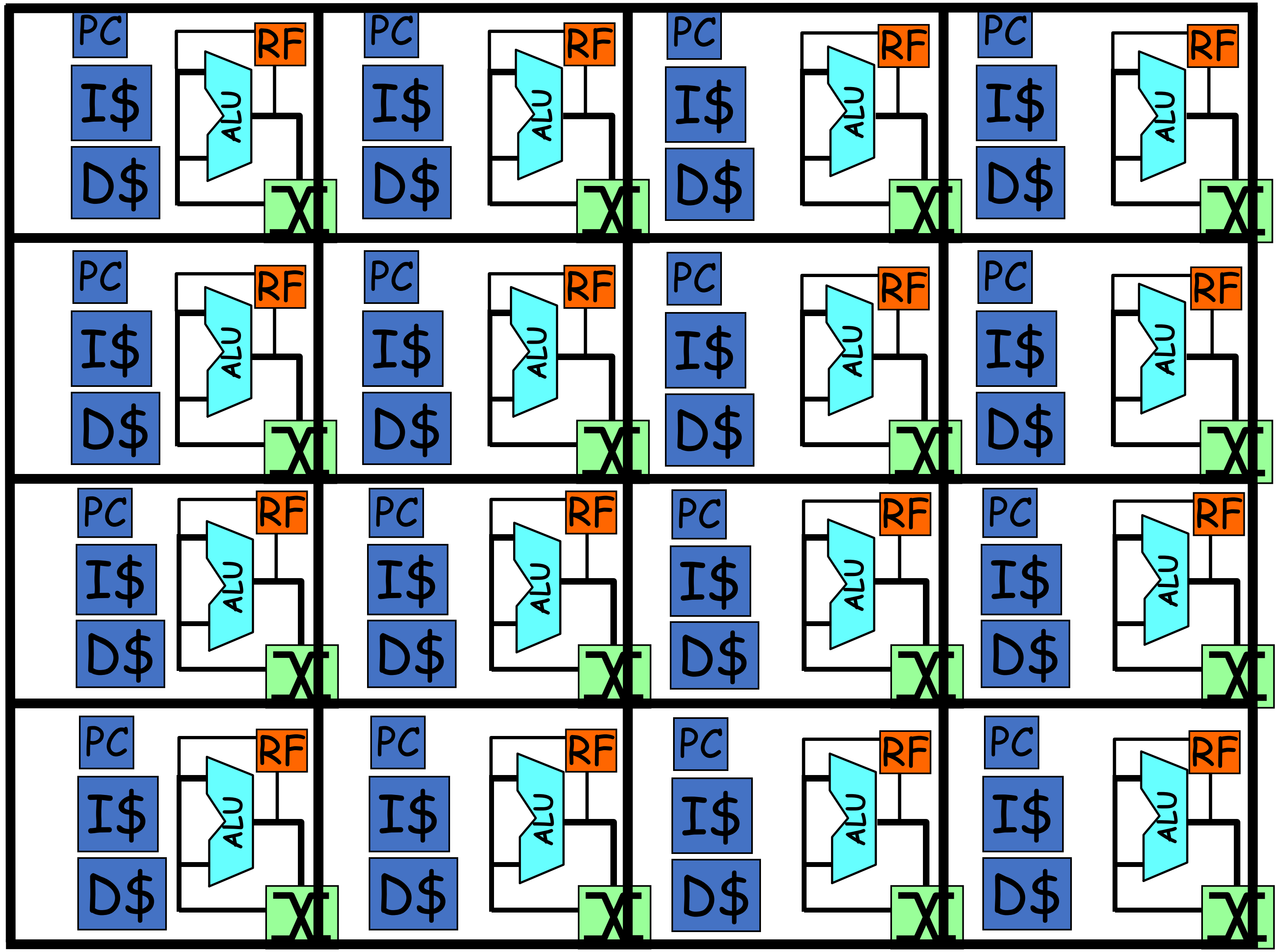
Distribute the Register File



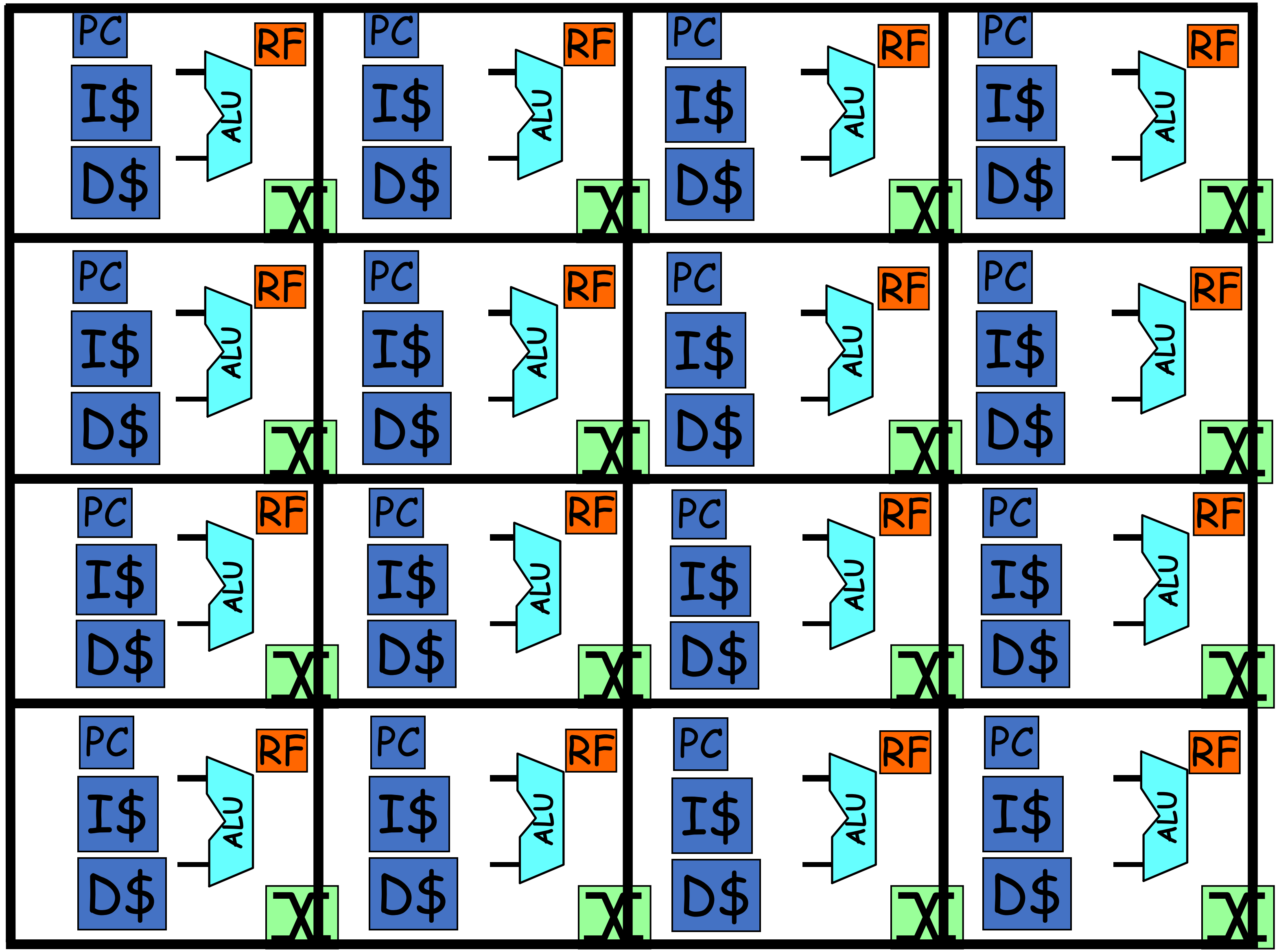
More Scalability Problems



Tiles (precursor to multicore)



Multicore (what was practical)



SFU

Widespread Assumption: Microarchitecture was the cause of the power problem

More cores on a chip

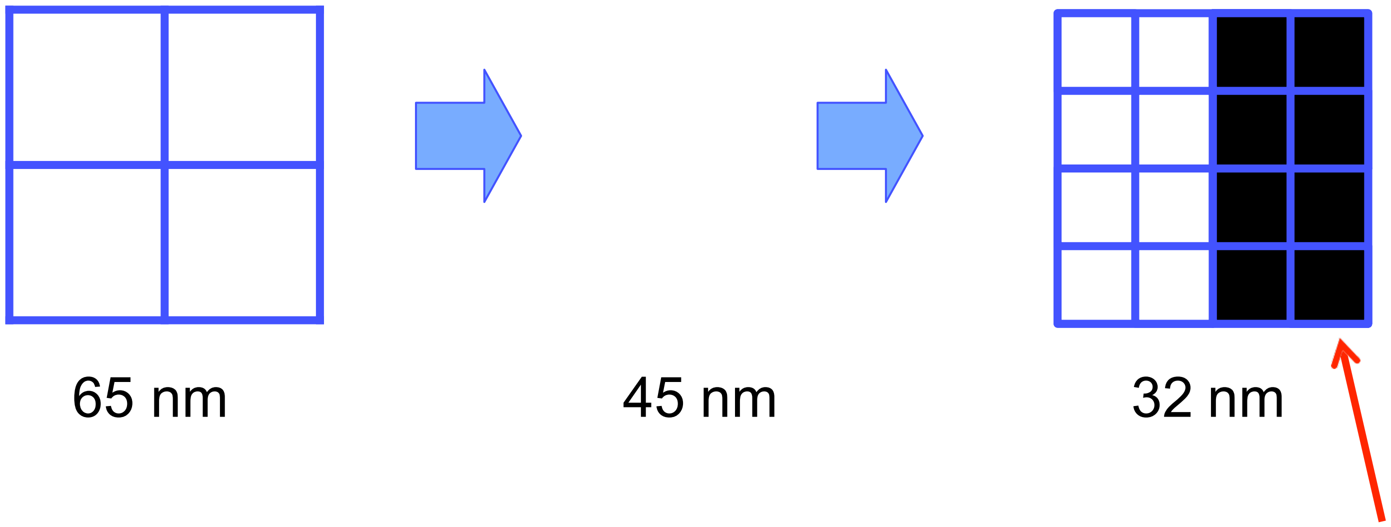
Each core ; 40%↓Ghz = 0.25x Power↓

Overall Performance = 4 cores * 0.6x/core = 2.4x

But actually, that's not what's happening

4 cores
1.8 GHz

8 cores
>= 1.8 GHz



1.4x cores per generation,
flat or slightly growing frequency

Dark or Dim
Silicon ("uncore")

SFU

Why OOOs suck.

Is technology scaling dead/dying ?

Are DSAs/Accelerators The Solution?

Scaling 101: Moore's Law

90 65 45 32 22 16 11 8 nm

$$S = \frac{22}{16} = \sim 1.4x$$

Scaling 101:
Transistors scale as S^2

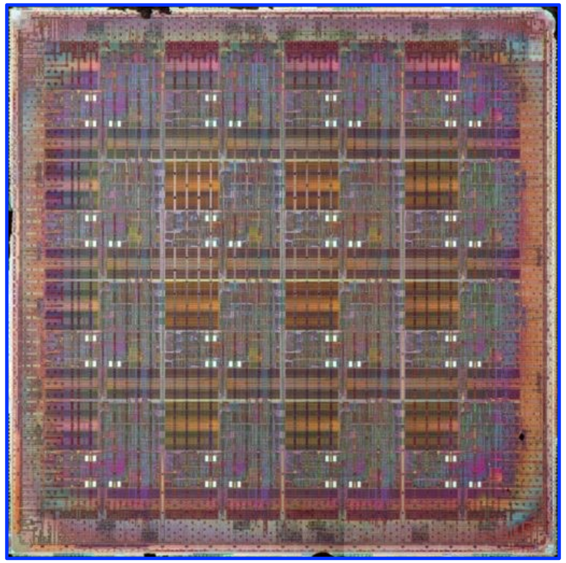
180 nm

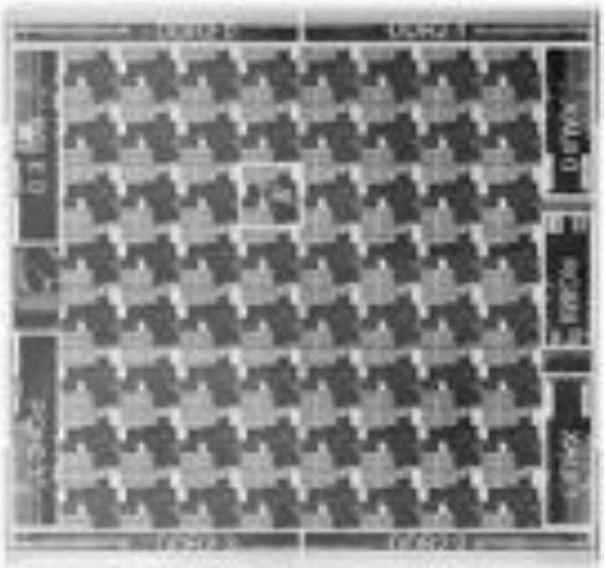
16 cores

S = 2x
Transistors = 4x

90 nm

64 cores





Advanced Scaling:

If S=1.4x … **Scale by S^3 = 2.8x"**

S^3

S^2

Design of Ion-Implanted MOSFETs with Very Small Dimensions
Dennard et al, 1974

S

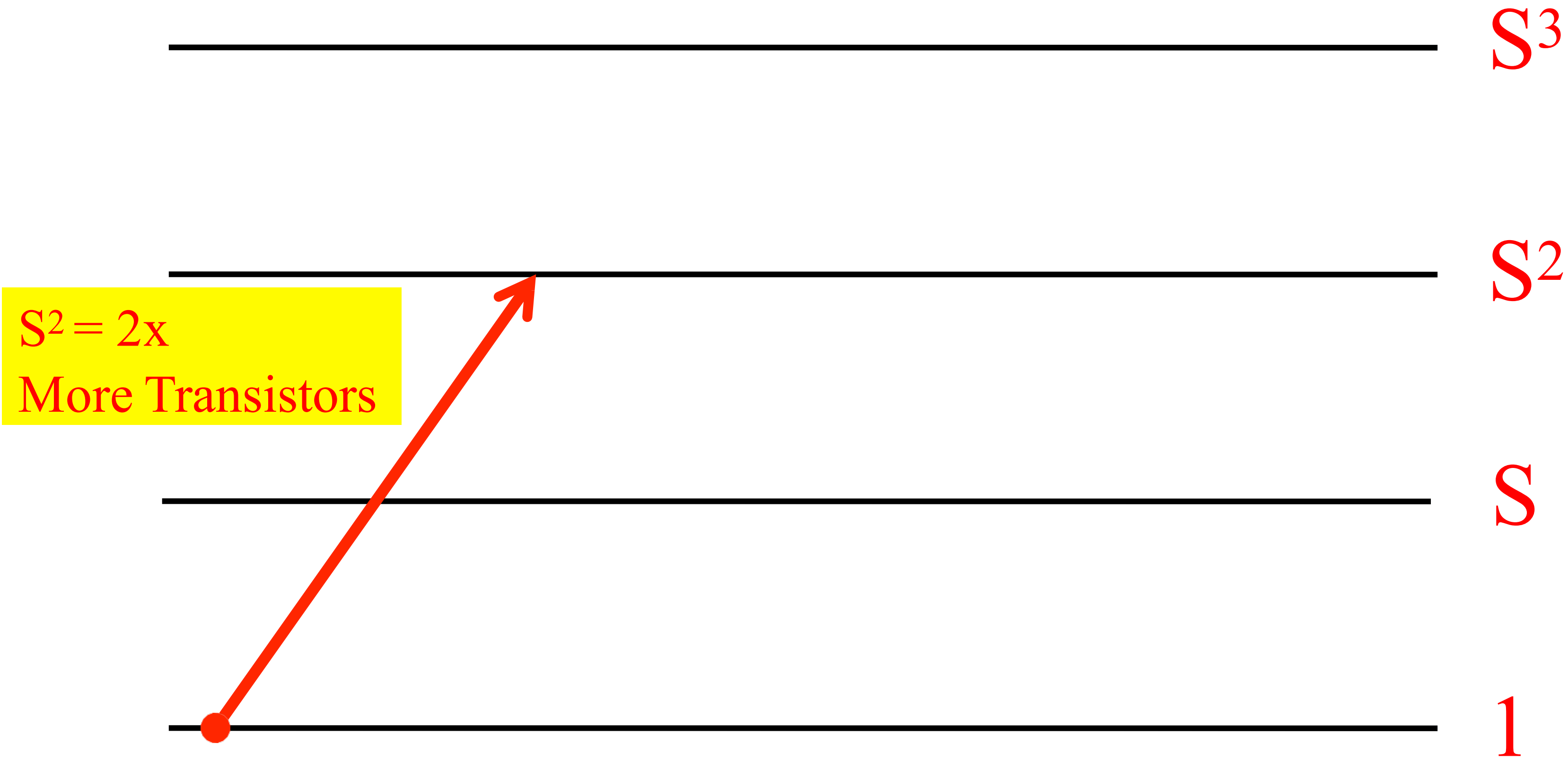
1

Dennard: "Computing Capabilities

Advanced Scaling:

If $S=1.4x$ …

Scale by S^3 = 2.8x”

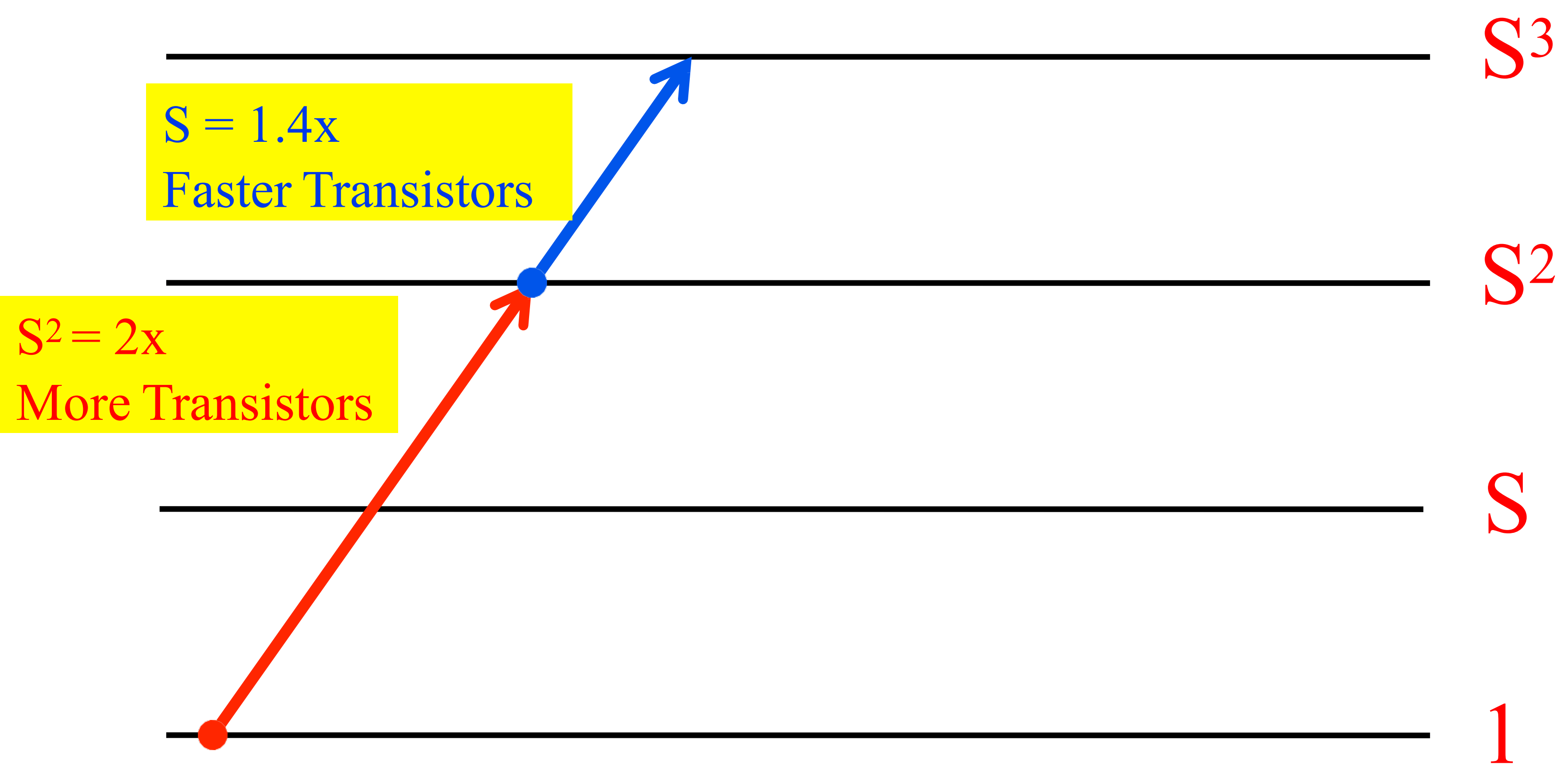


Dennard: “Computing Capabilities

Advanced Scaling:

If S=1.4x …

Scale by S^3 = 2.8x”

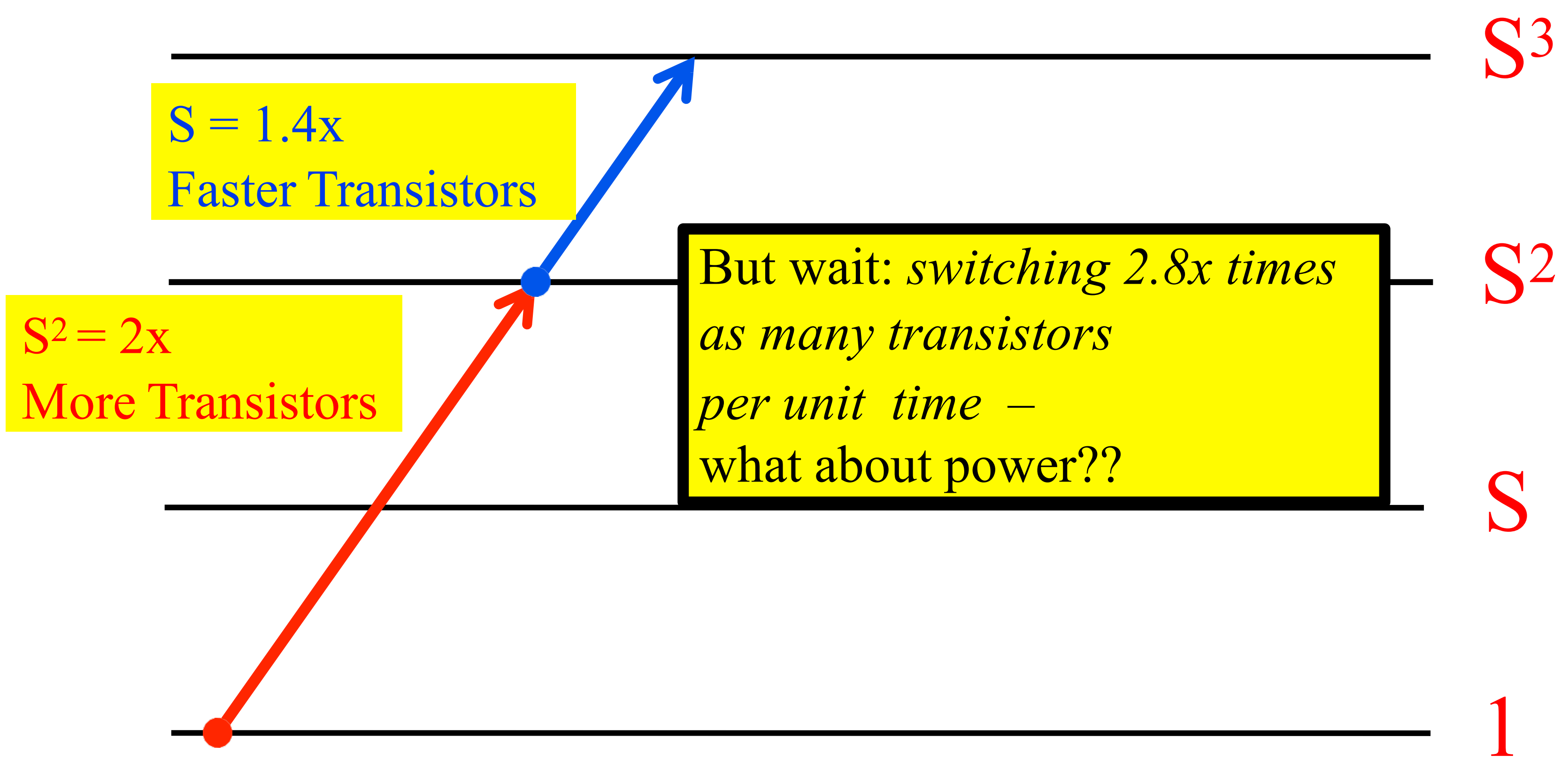


Dennard: “Computing Capabilities

Advanced Scaling:

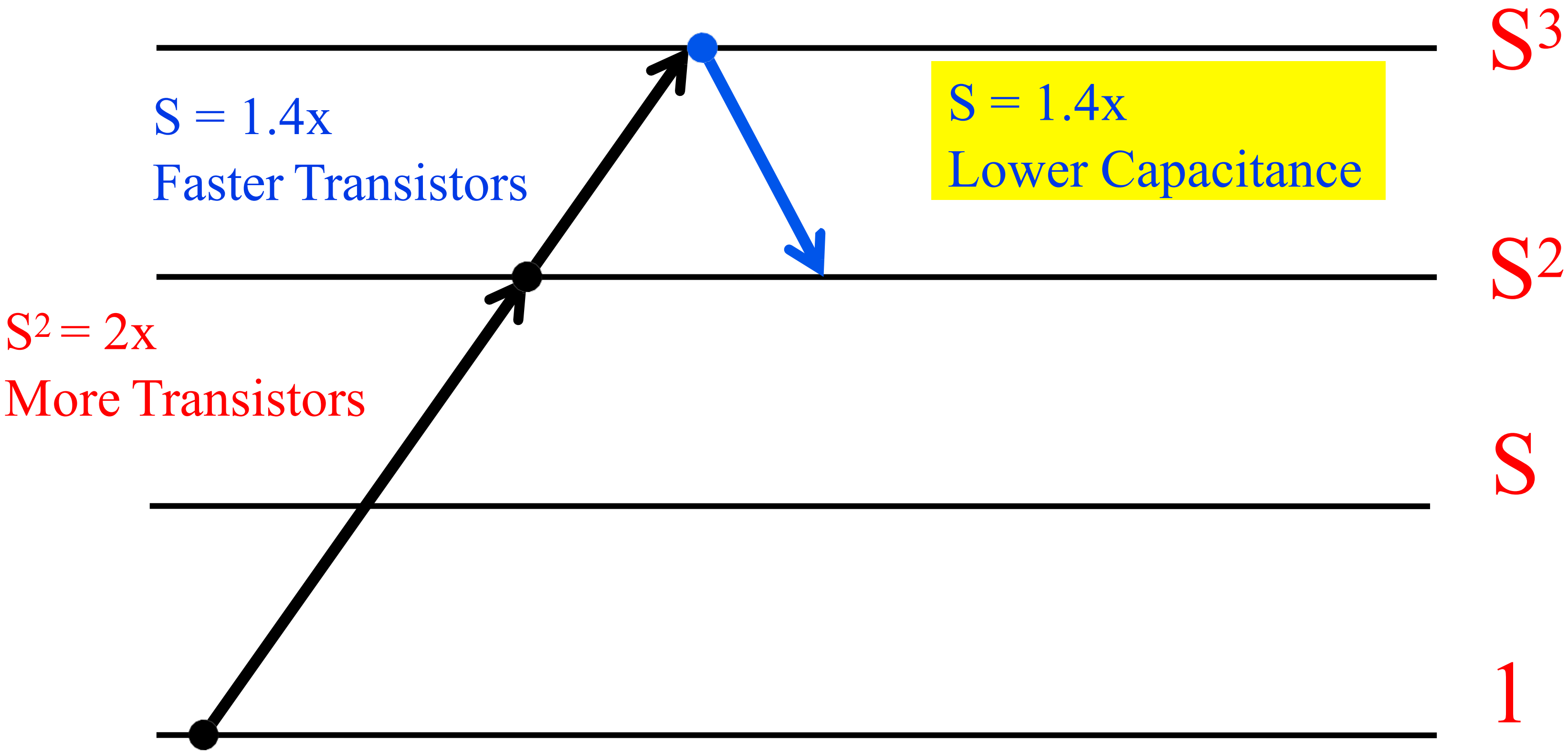
If S=1.4x …

Scale by S^3 = 2.8x"



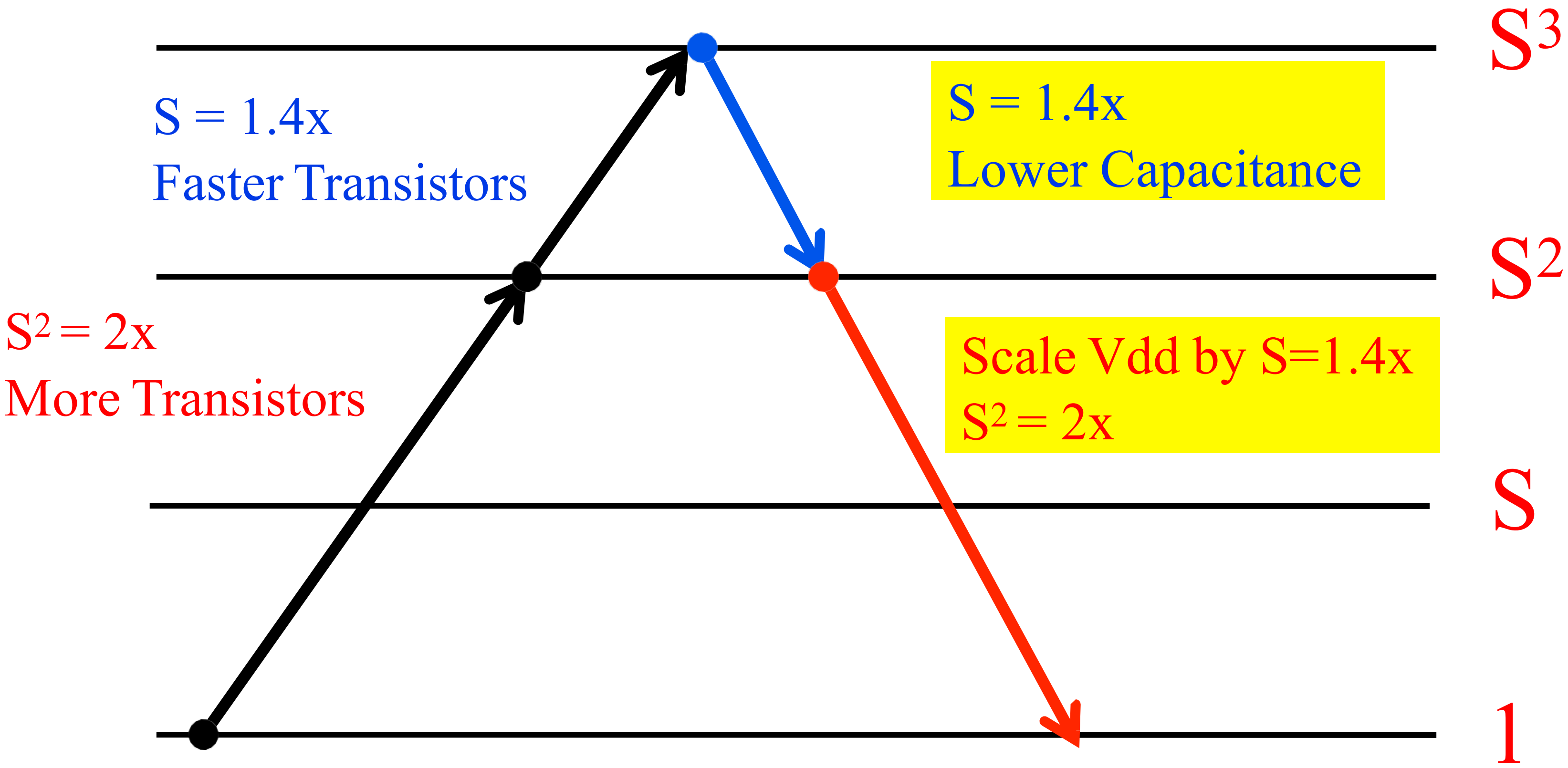
"We can keep power consumption constant"

Dennard:



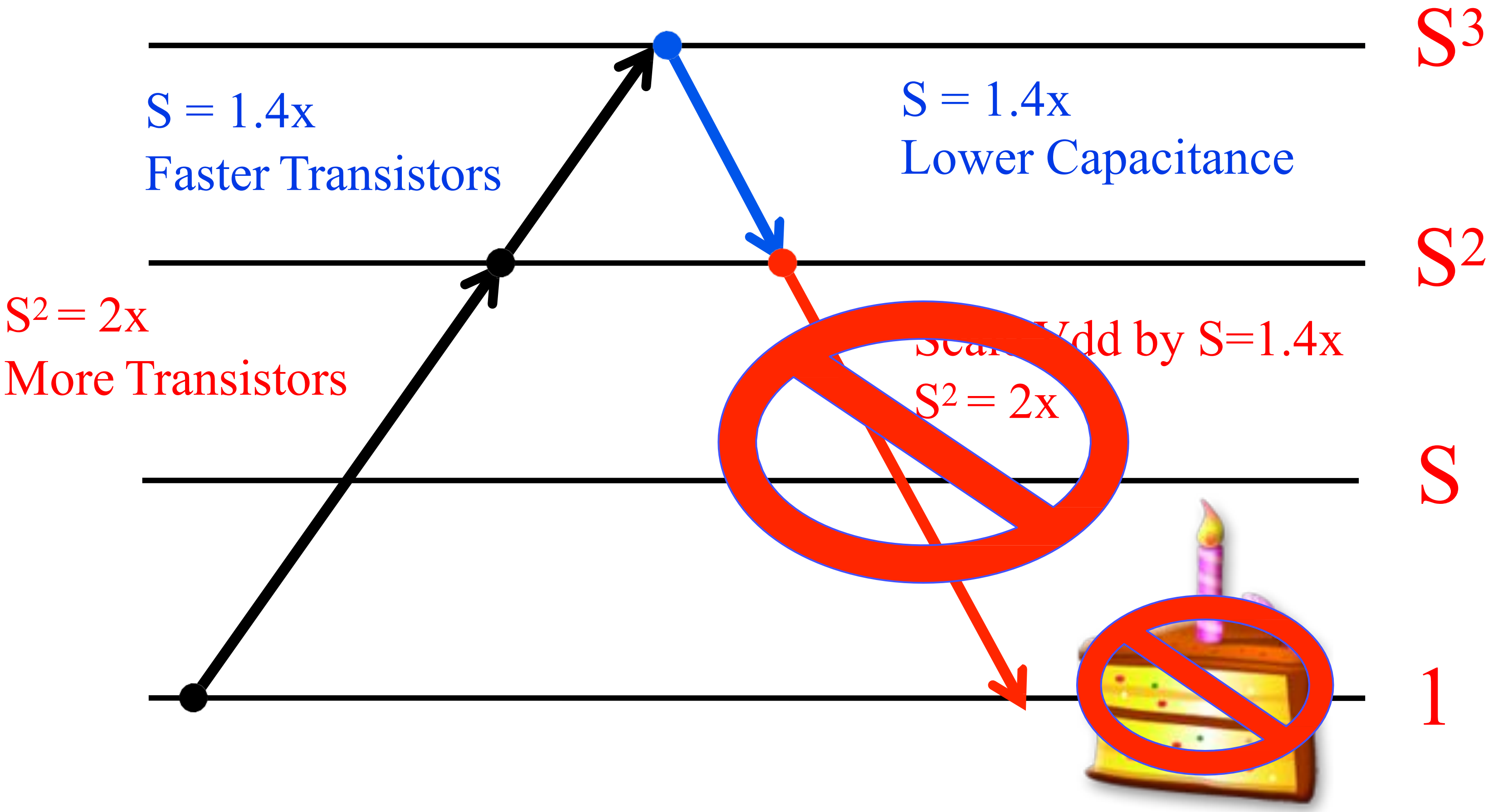
"We can keep power consumption constant"

Dennard:



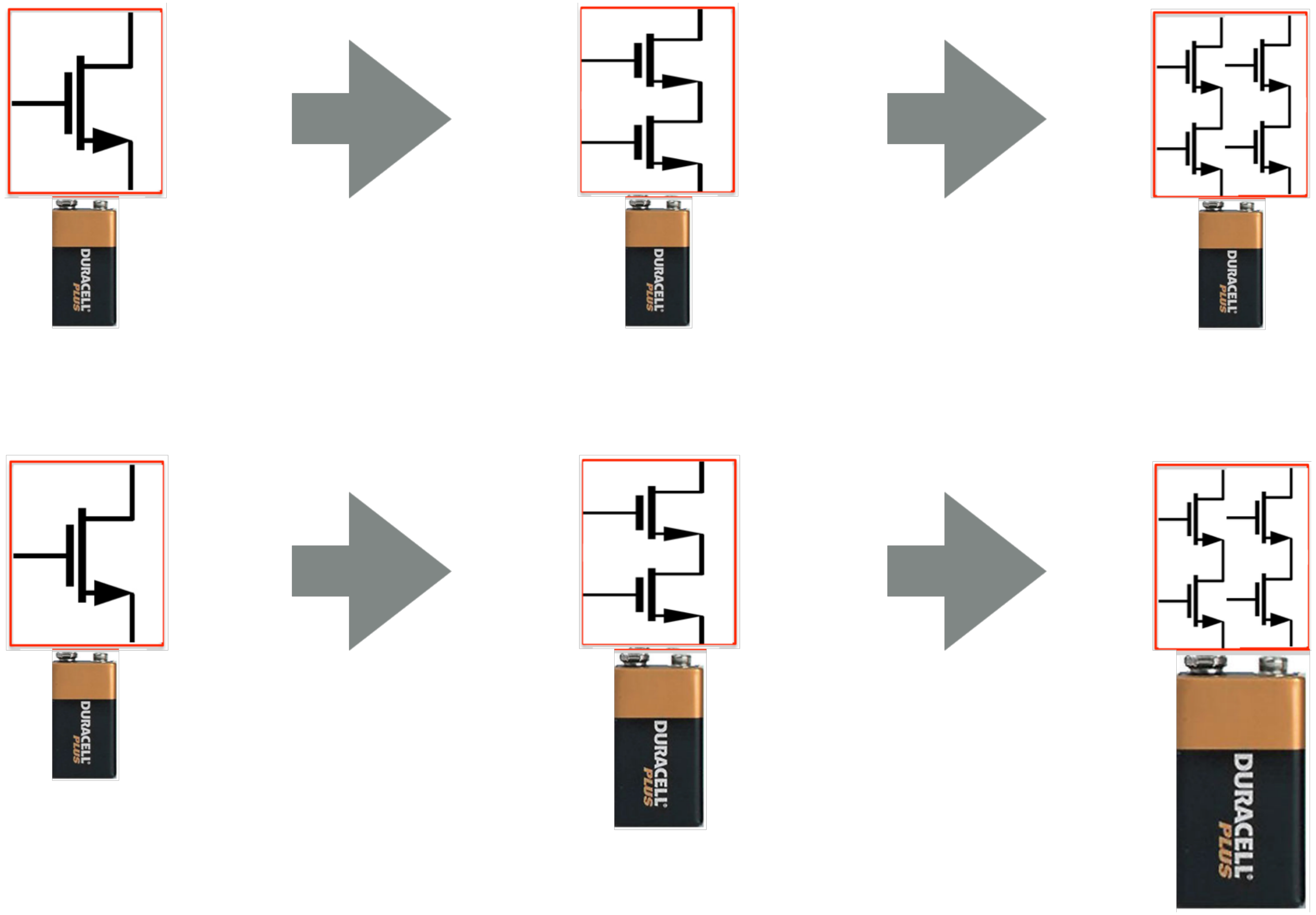
"We can keep power consumption constant"

Fast forward to 2005:



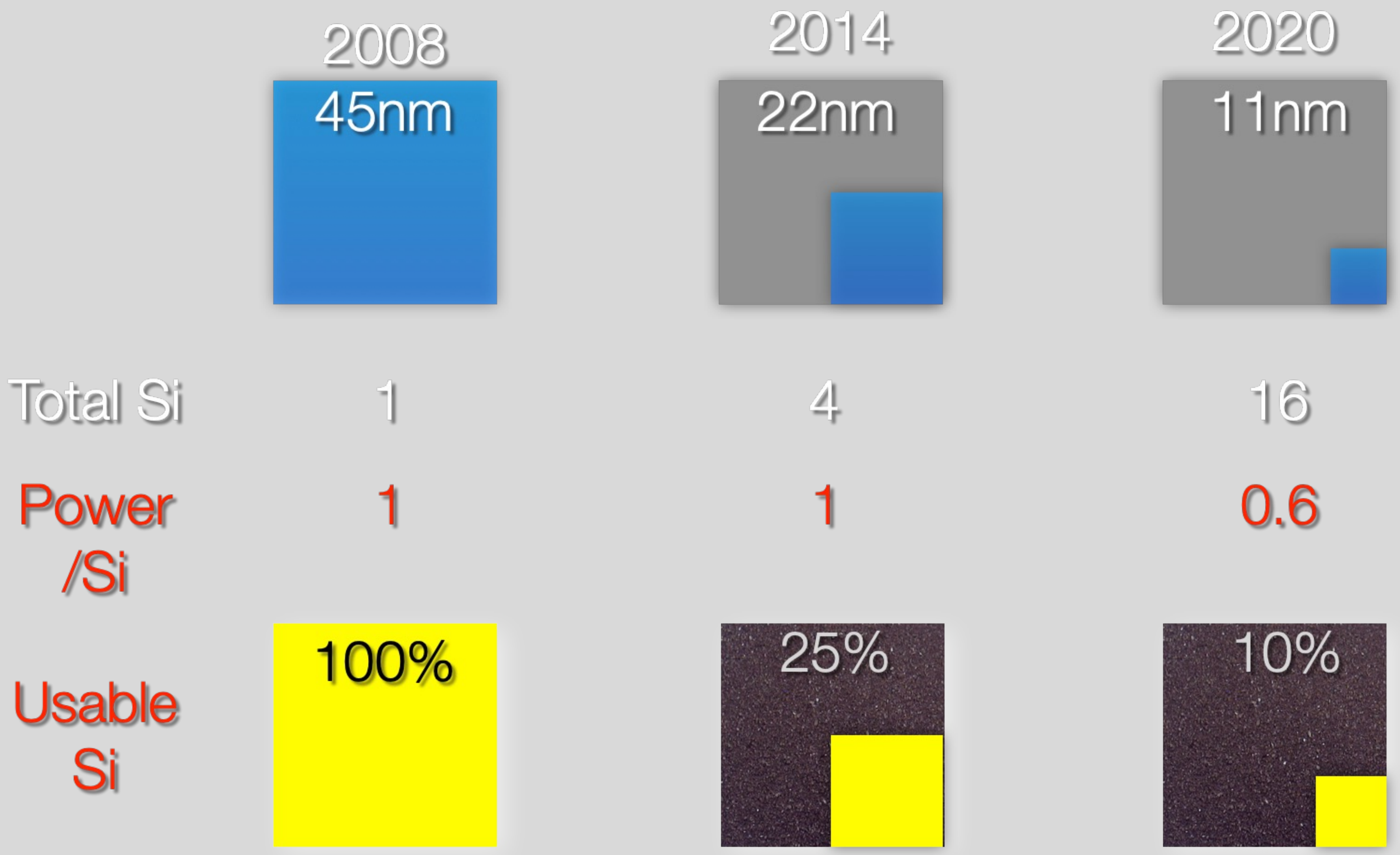
Leakage Prevents Us From Scaling Voltage

Utilization Wall

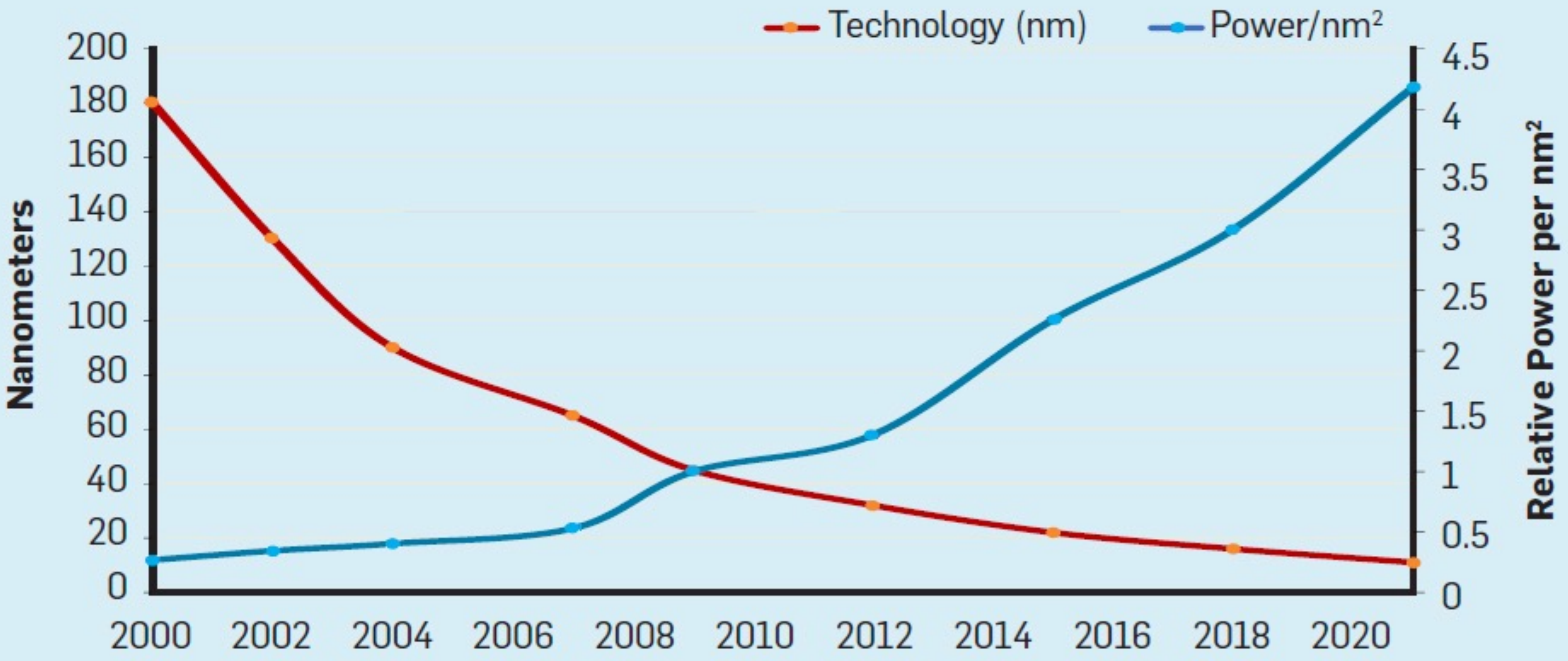


We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.



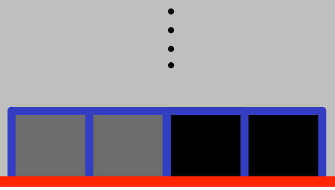
Transistors vs Power



https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

Multicore has hit the Utilization Wall

Spectrum of tradeoffs between # of cores and frequency



2x4 cores @ 1.8 GHz

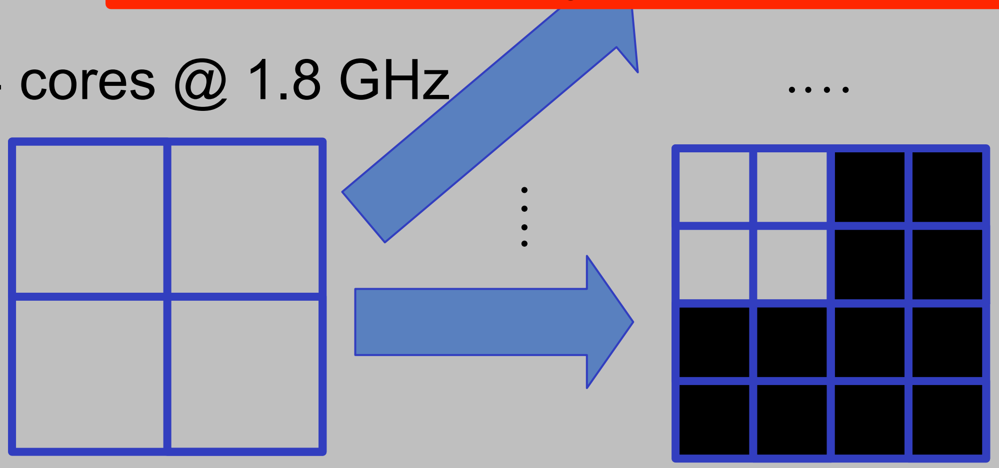
m)

Exa

65 r

The utilization wall will change the way everyone builds chips.

4 cores @ 1.8 GHz



4 cores @ 2x1.8 GHz
(12 cores dark)

65 nm

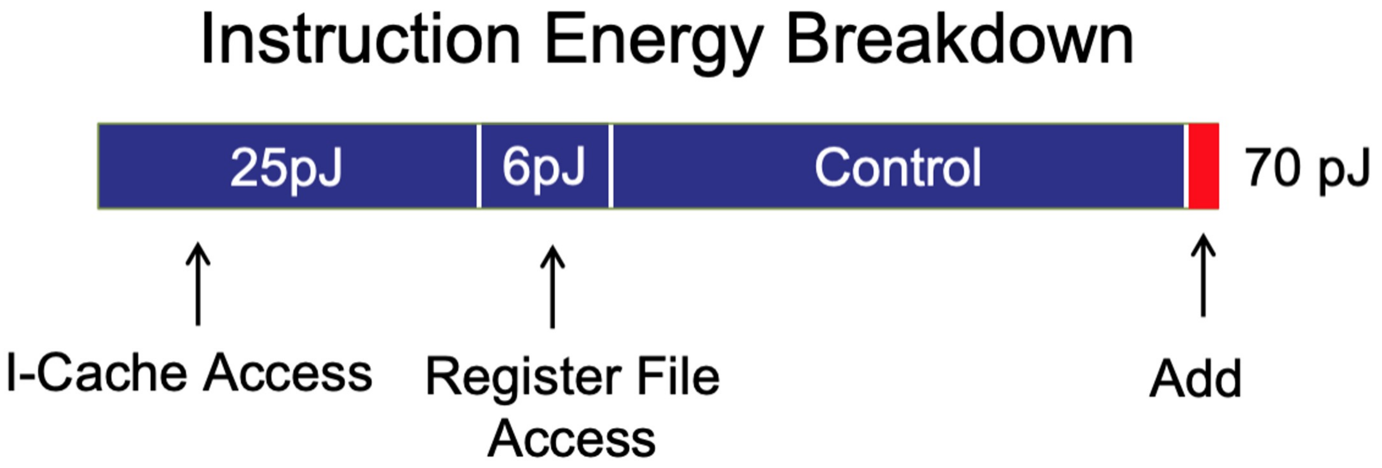
32 nm

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

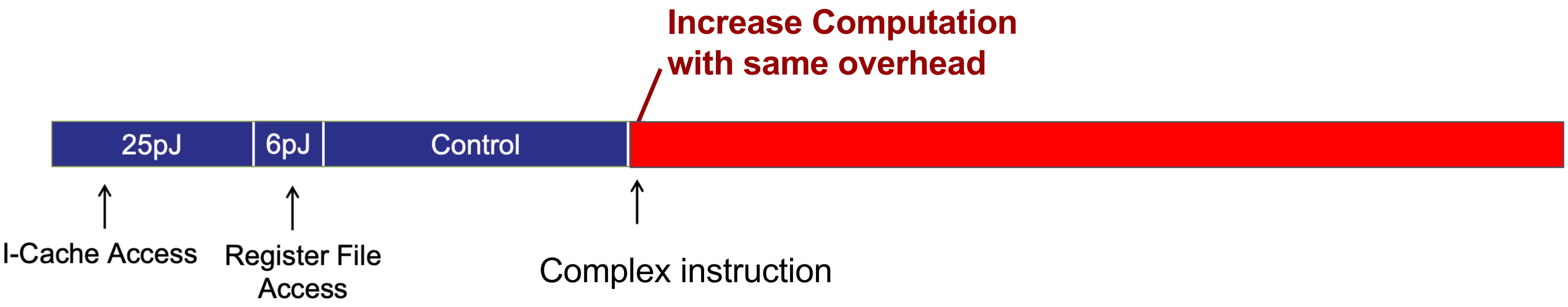
Where does the Energy go?

- 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

Amortize Overhead



Half-precision Fused Multiply-Add

4-way dot-product

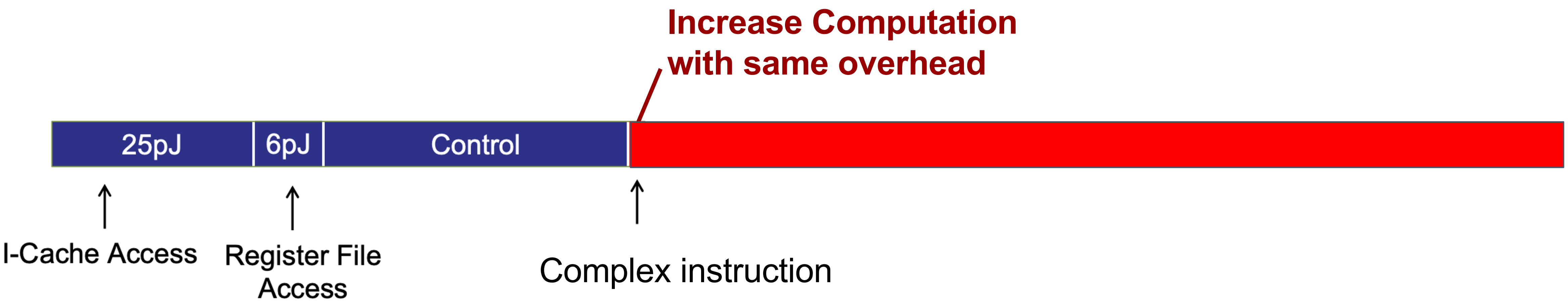
16x16 matrix multiplication

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

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

SFU

Amortize Overhead



Half-precision
Fused Multiply-Add

4-way dot-product

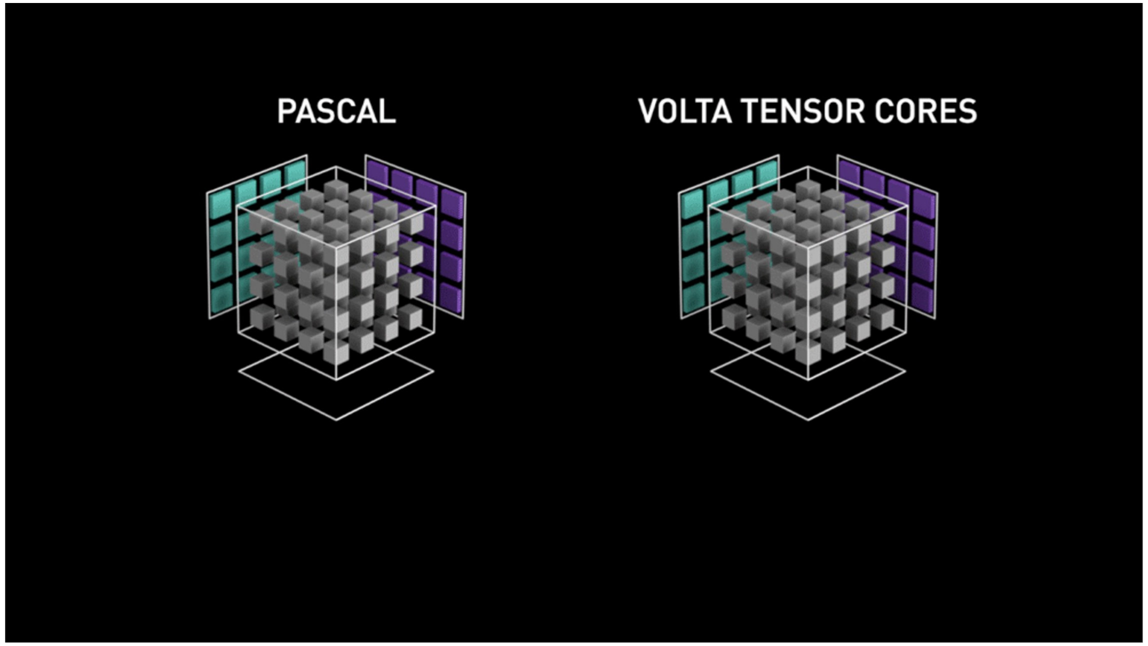
16x16 matrix
multiplication

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

GPU

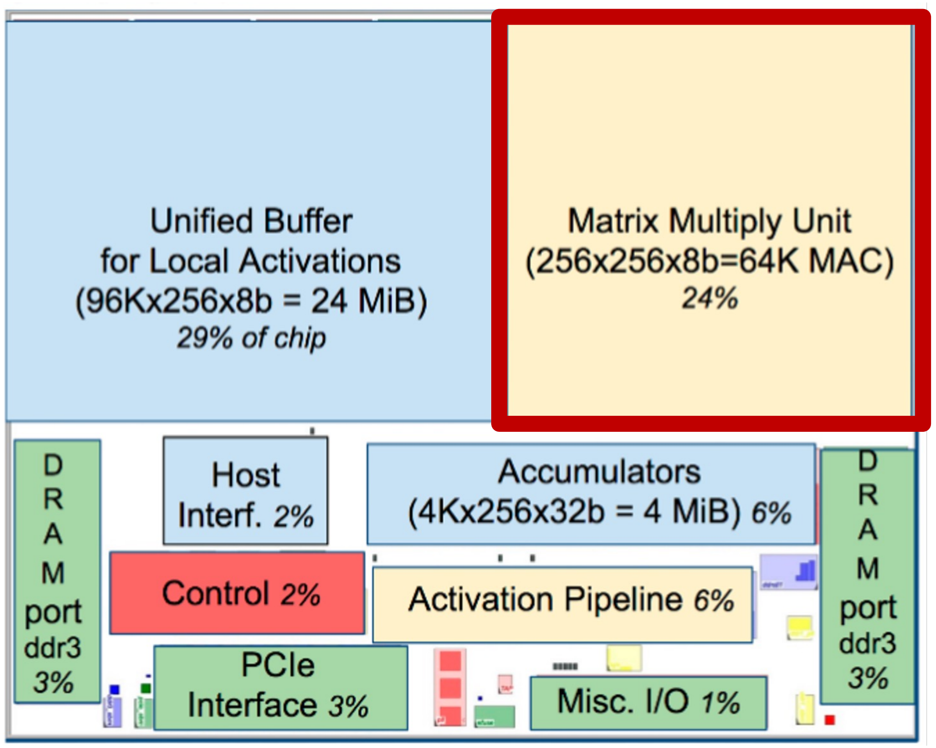


16x16 = 256* MAC/cycle

*~ 500 tensor cores per GPU

Source: Nvidia

ASIC (TPUv1)



256x256 = 64 kMAC/cycle

Source: Google

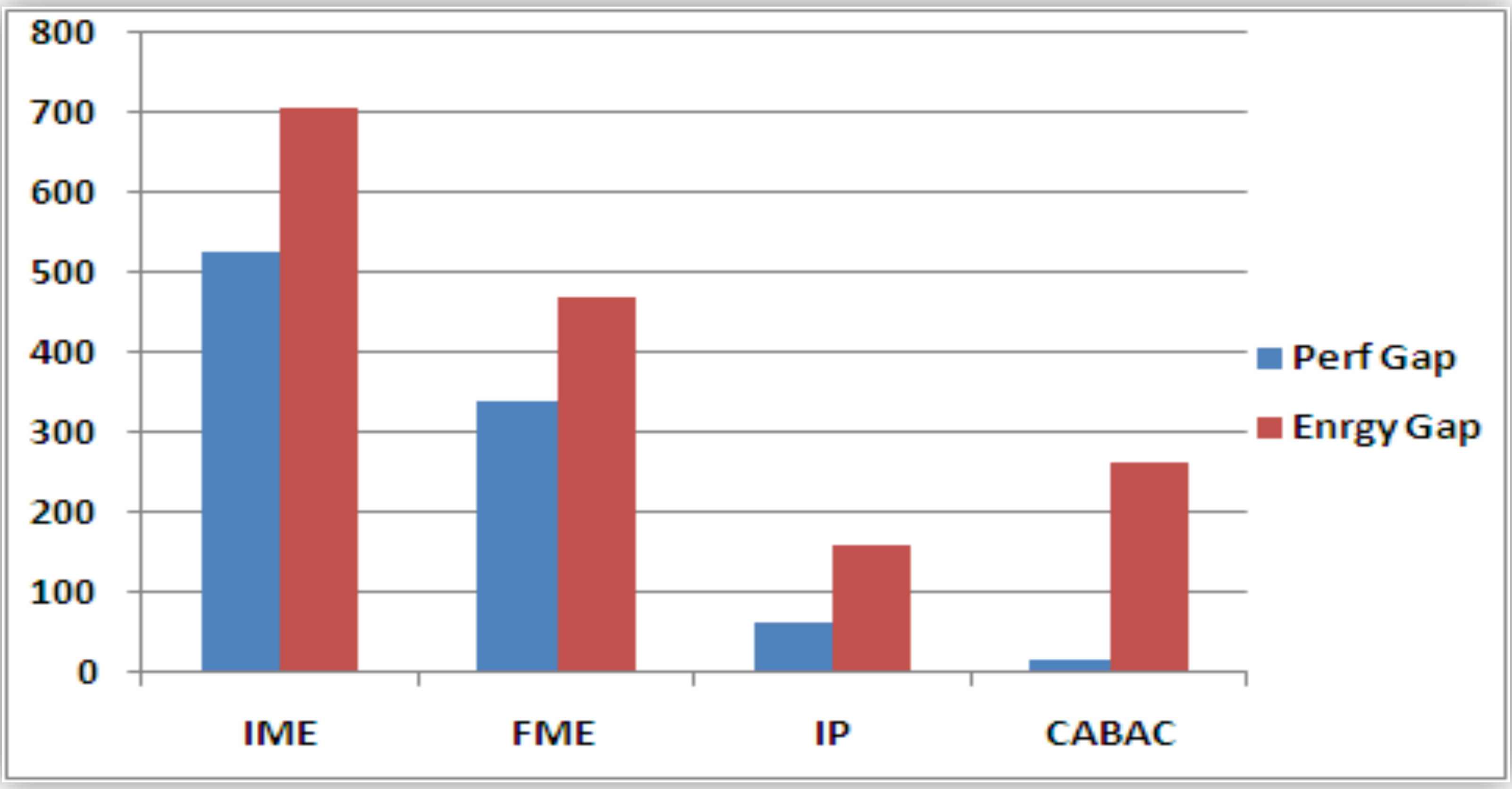
Multicore vs. ASIC

Huge efficiency gap

- 4-proc CMP 250x slower
- 500x extra energy

Manycore doesn't help

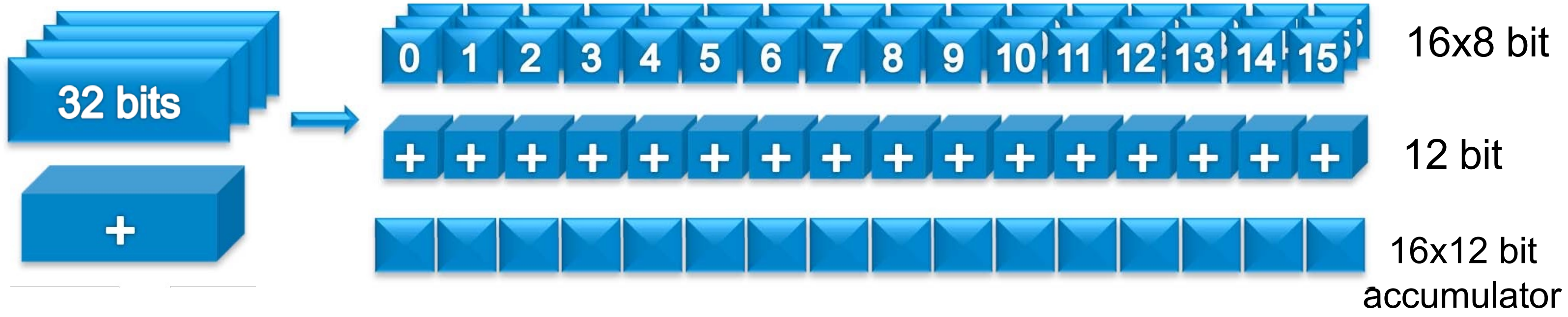
- Energy/frame remains same
- Performance improves



Opt 1: SIMD, VLIW and Horizontal Fusion

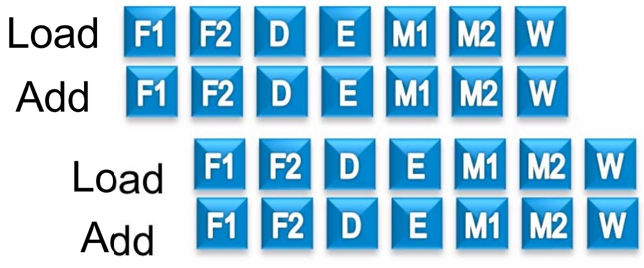
SIMD

- Up to 18-way SIMD in reduced precision



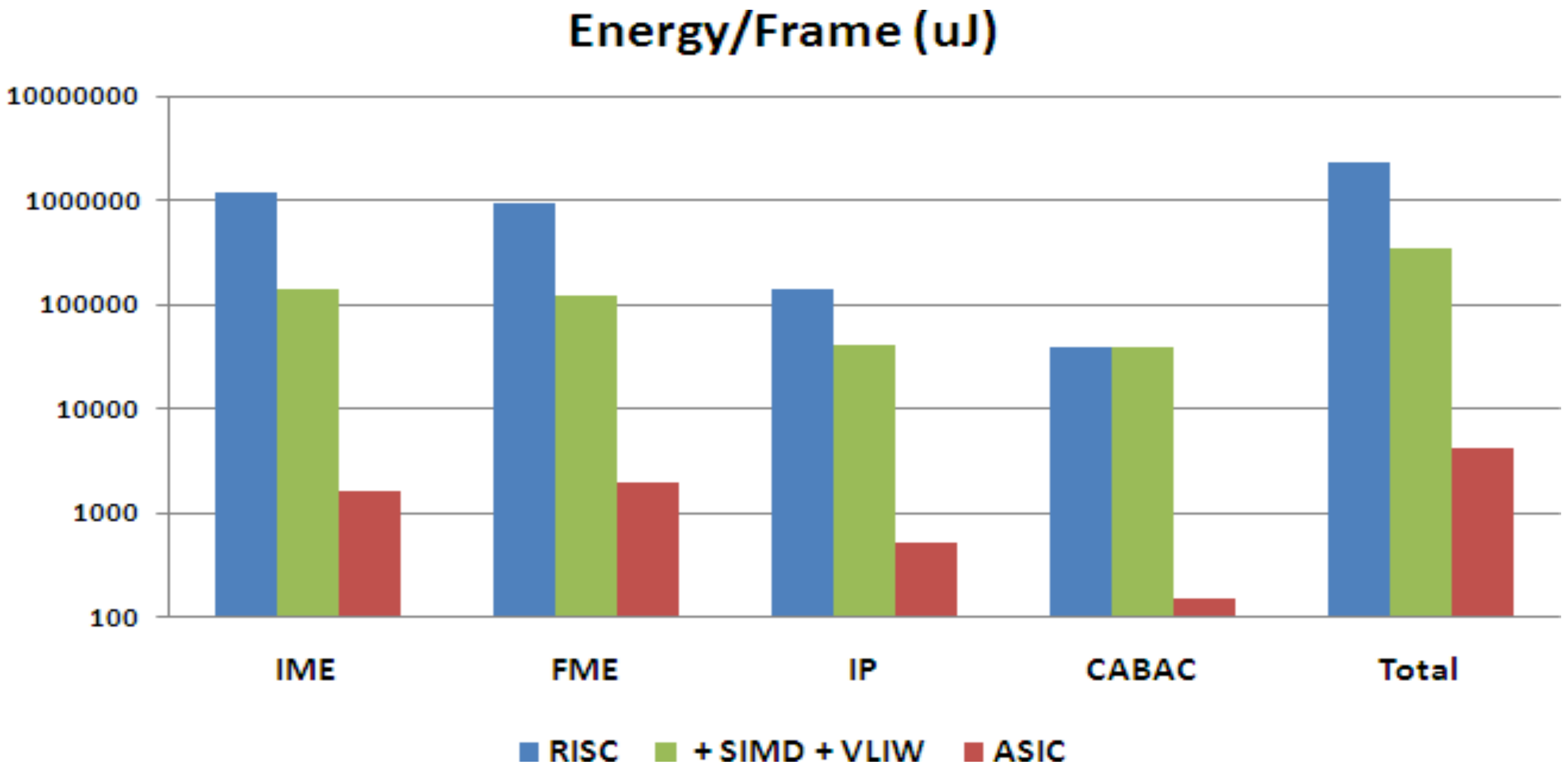
VLIW

- Up to 3-slot VLIW



SIMD and ILP - Results

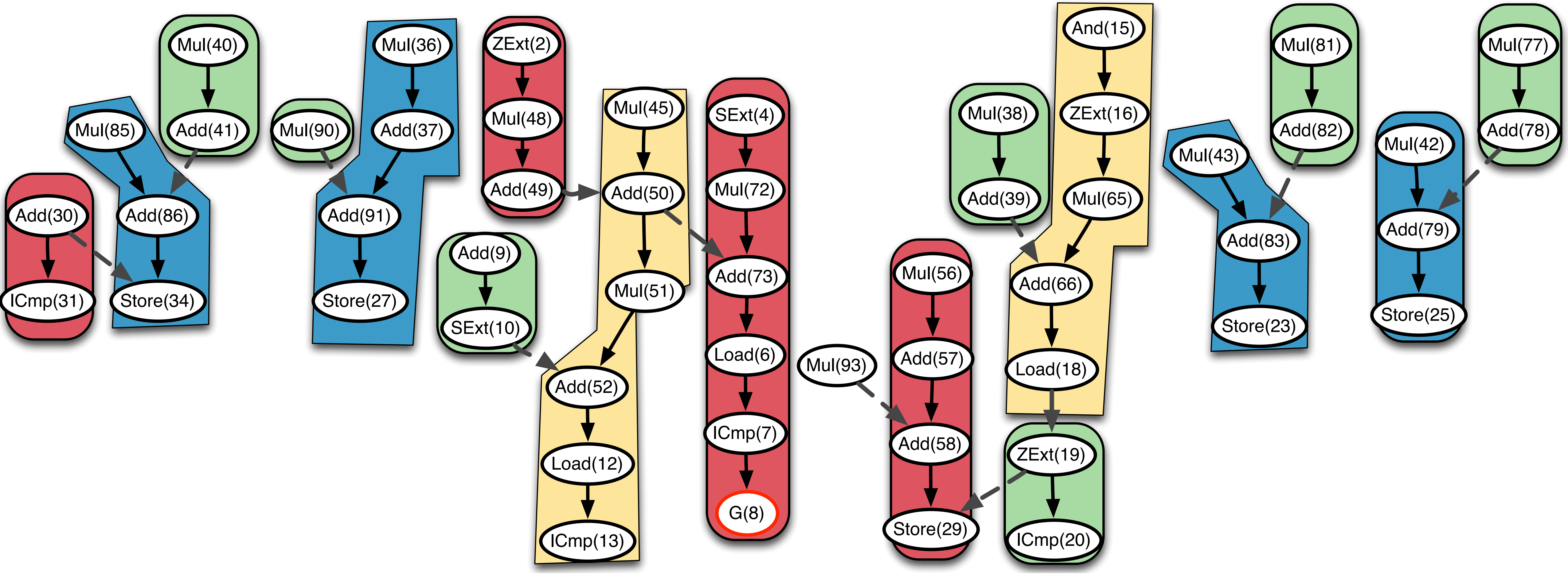
SFU



Order of magnitude improvement in performance, energy

- For data parallel algorithms
- But ASIC still better by roughly 2 orders of magnitude

Opt 2: Op Fusion



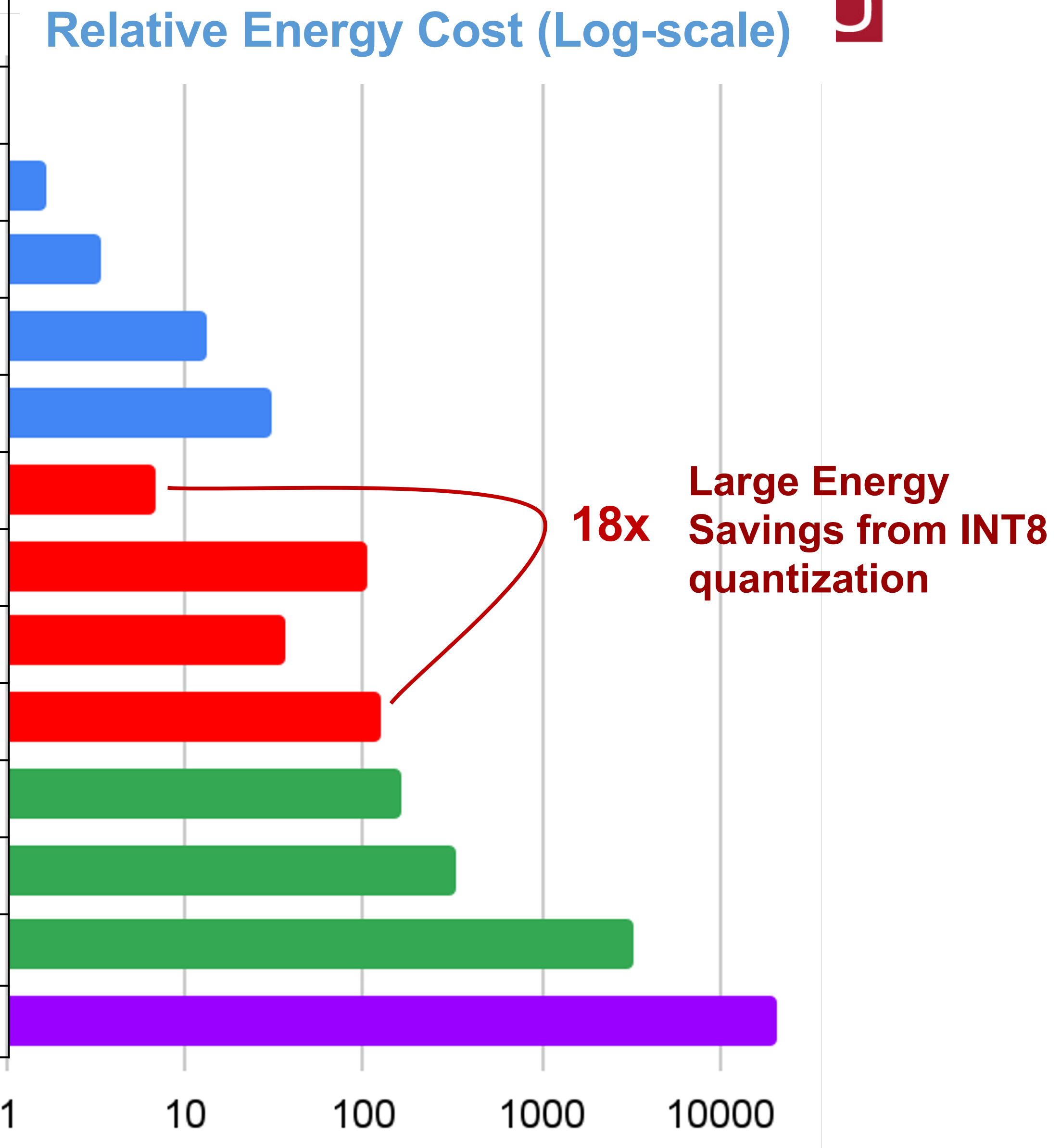
Chainsaw: Von-Neumann Accelerators, Amirali et al., MICRO 2016

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

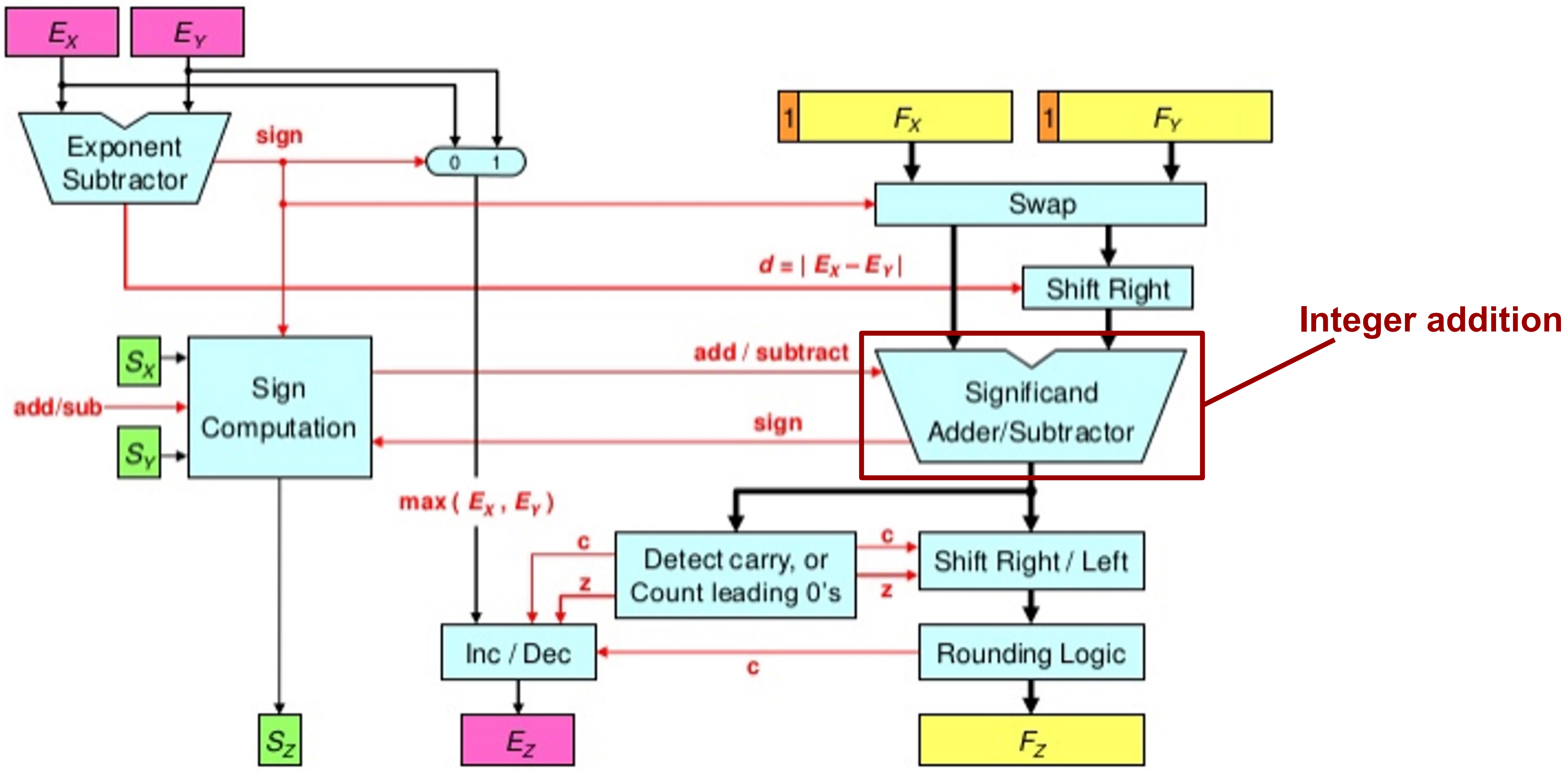
Why is floating-point add so expensive compared to integer add?

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

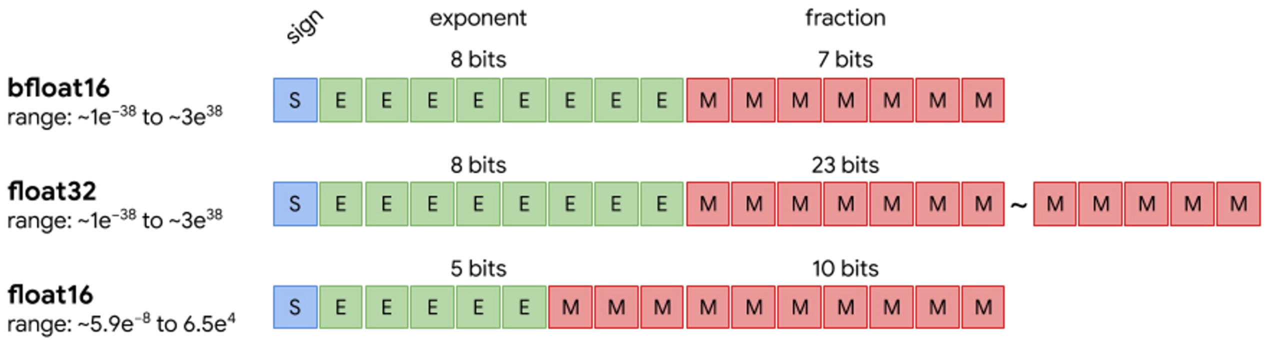
Floating-Point Addition



Numerical Format and Precision

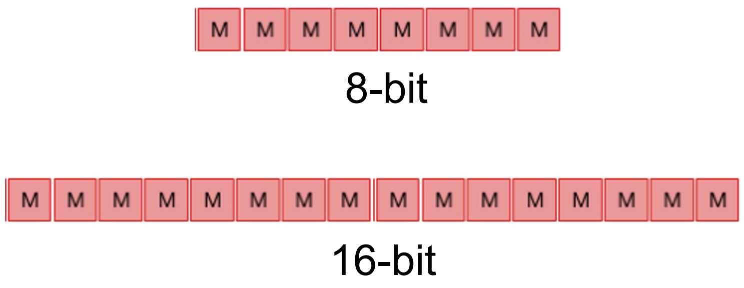
SFU

Floating Point



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

Integer



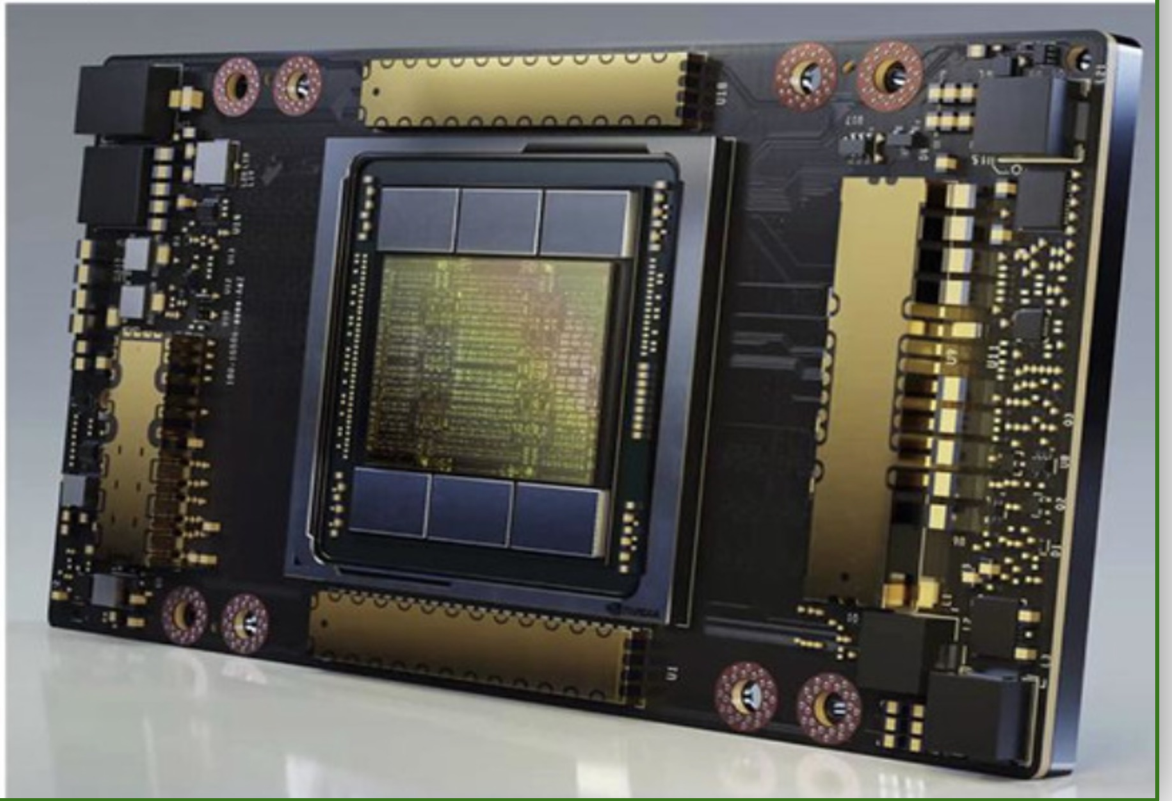
- Whole numbers only
- (typically) much cheaper circuit area and power

Source: Google

Ampere (2020)

Sparsity!
BF16 & TF32!
156 / 312 TFLOPS (TF32) (dense/sparse)
312 / 624 TFLOPS (FP16 or BF16)
624 / 1,248 TOPS (Int 8)
1,248 / 2,496 TOPS (Int 4)
2TB/s (HBM)
400W

3.12 TOPS/W (Int 8)
6.24 TOPS/W (Int 4)



Block Floating Point

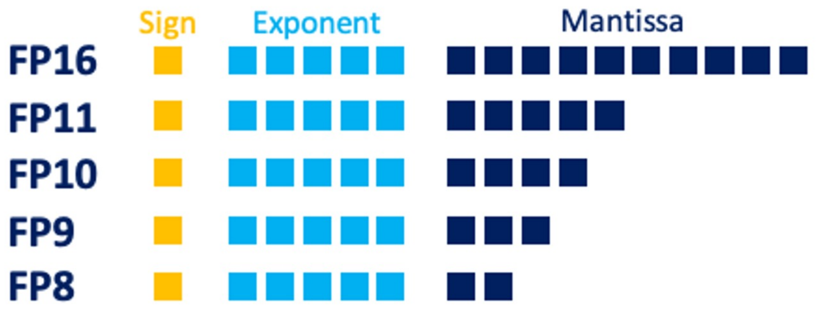
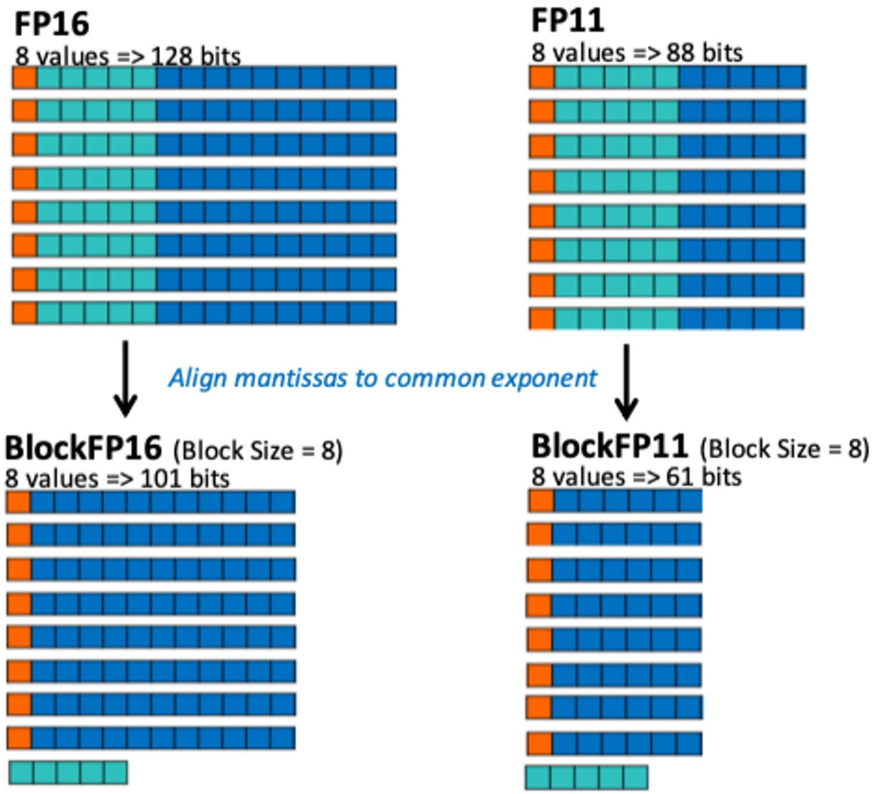


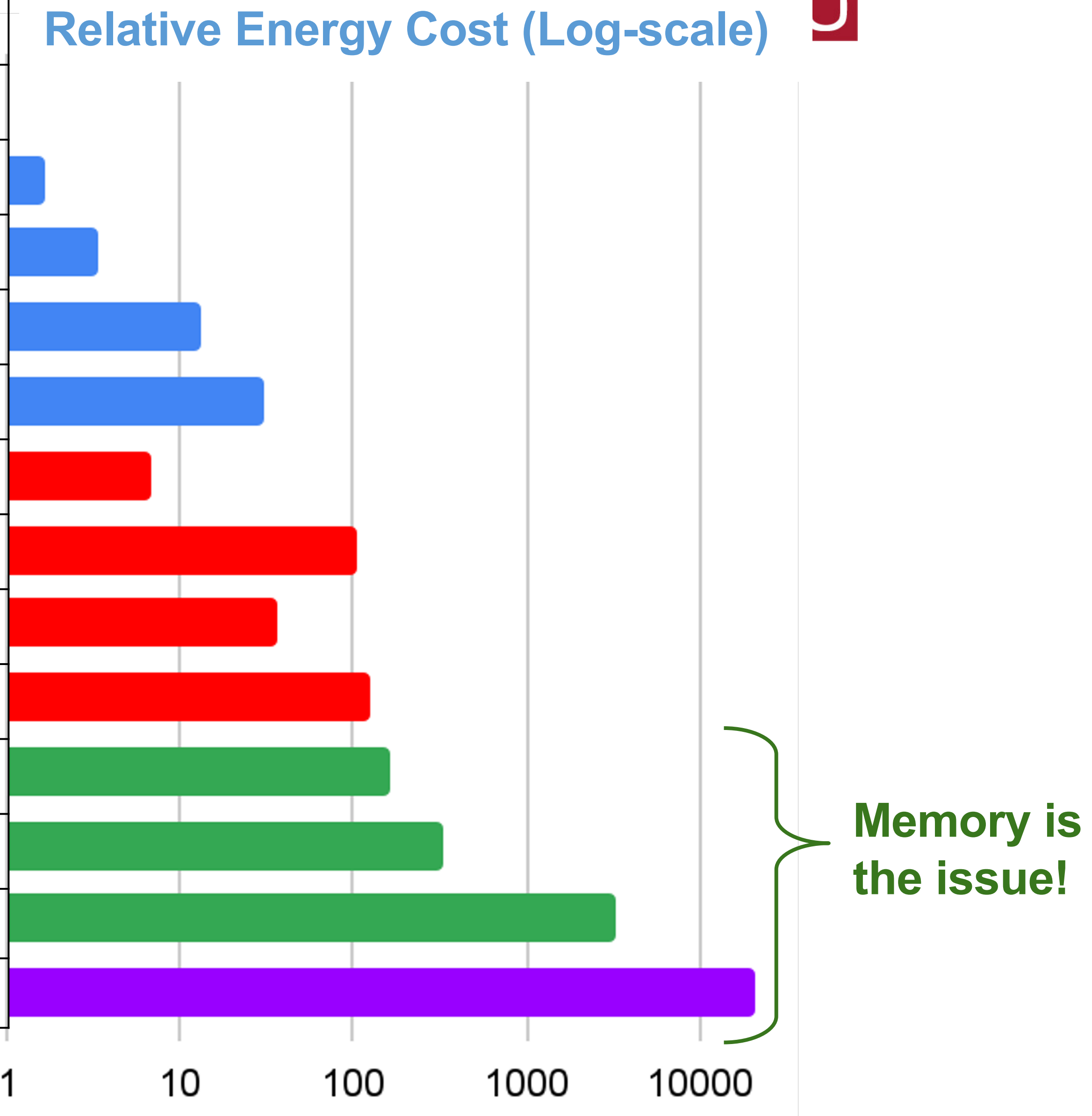
Figure 3: Reduced-precision floating point sign/exponent/mantissa breakdown.



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

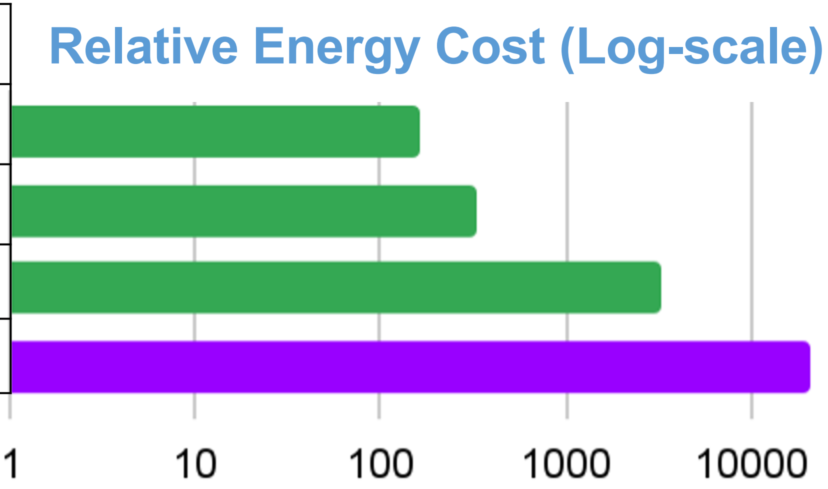


Adapted from Mark Horowitz "Computing's Energy Problem (and What we can do about it)" ISSCC 2014

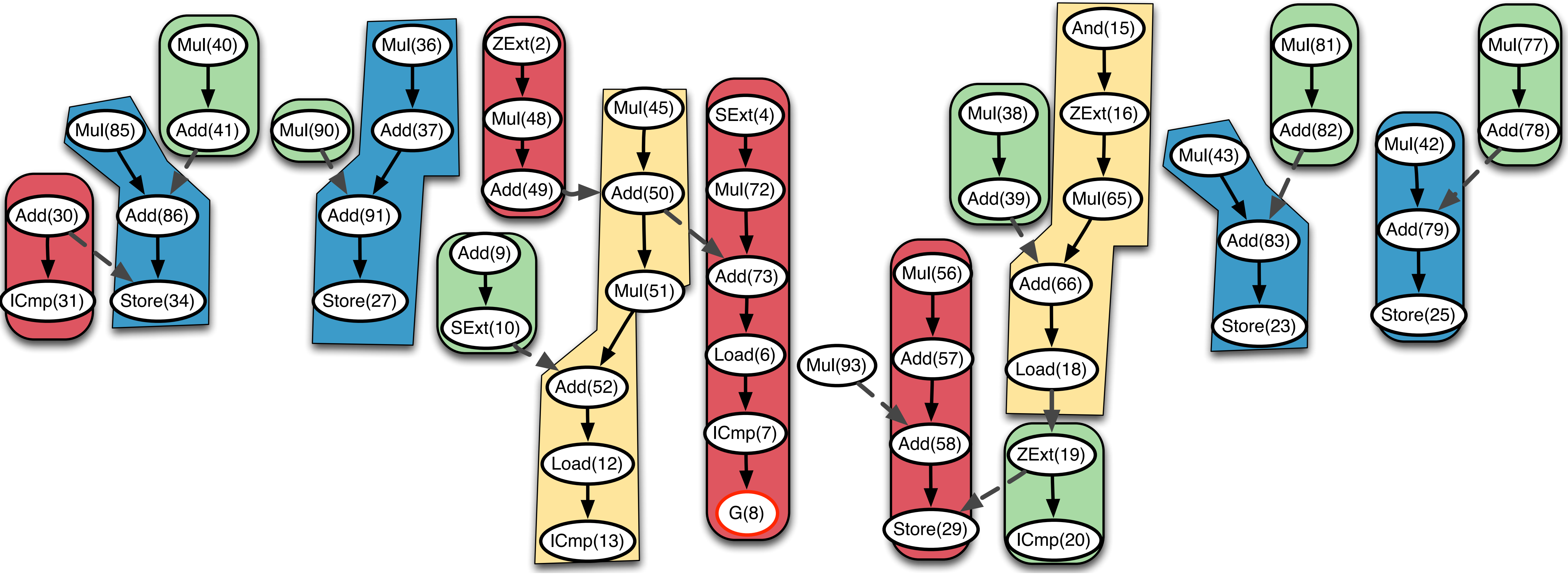
Memory Hierarchy Optimizations

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

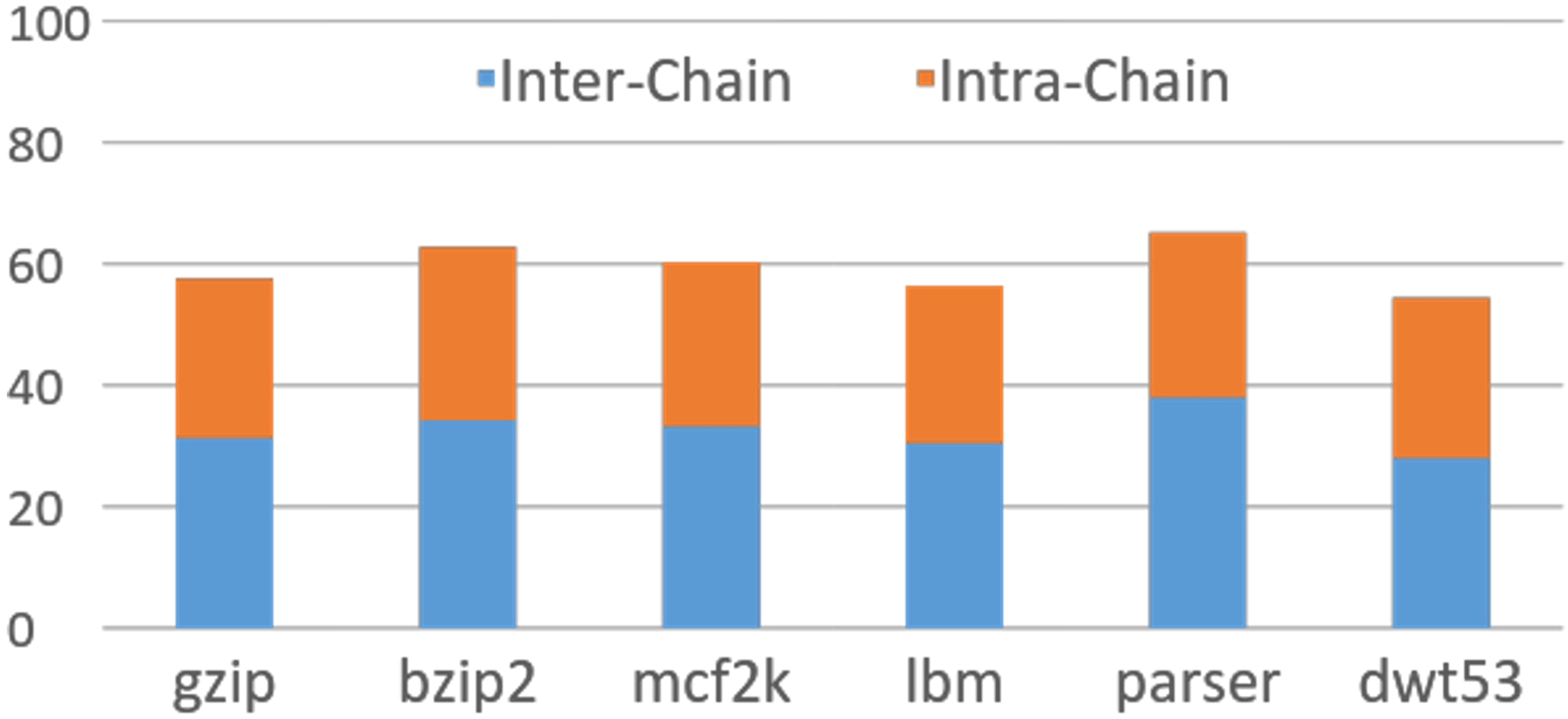


Opt 2: Op Fusion



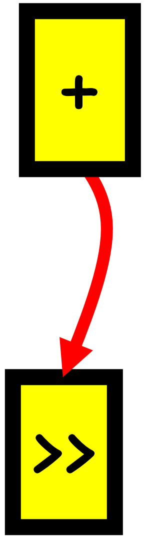
Chainsaw: Von-Neumann Accelerators, Amirali et al., MICRO 2016

Opt 2: Op Fusion



Reduces 40% of data movement energy

“Magic” Instructions



Create specialized data storage structures

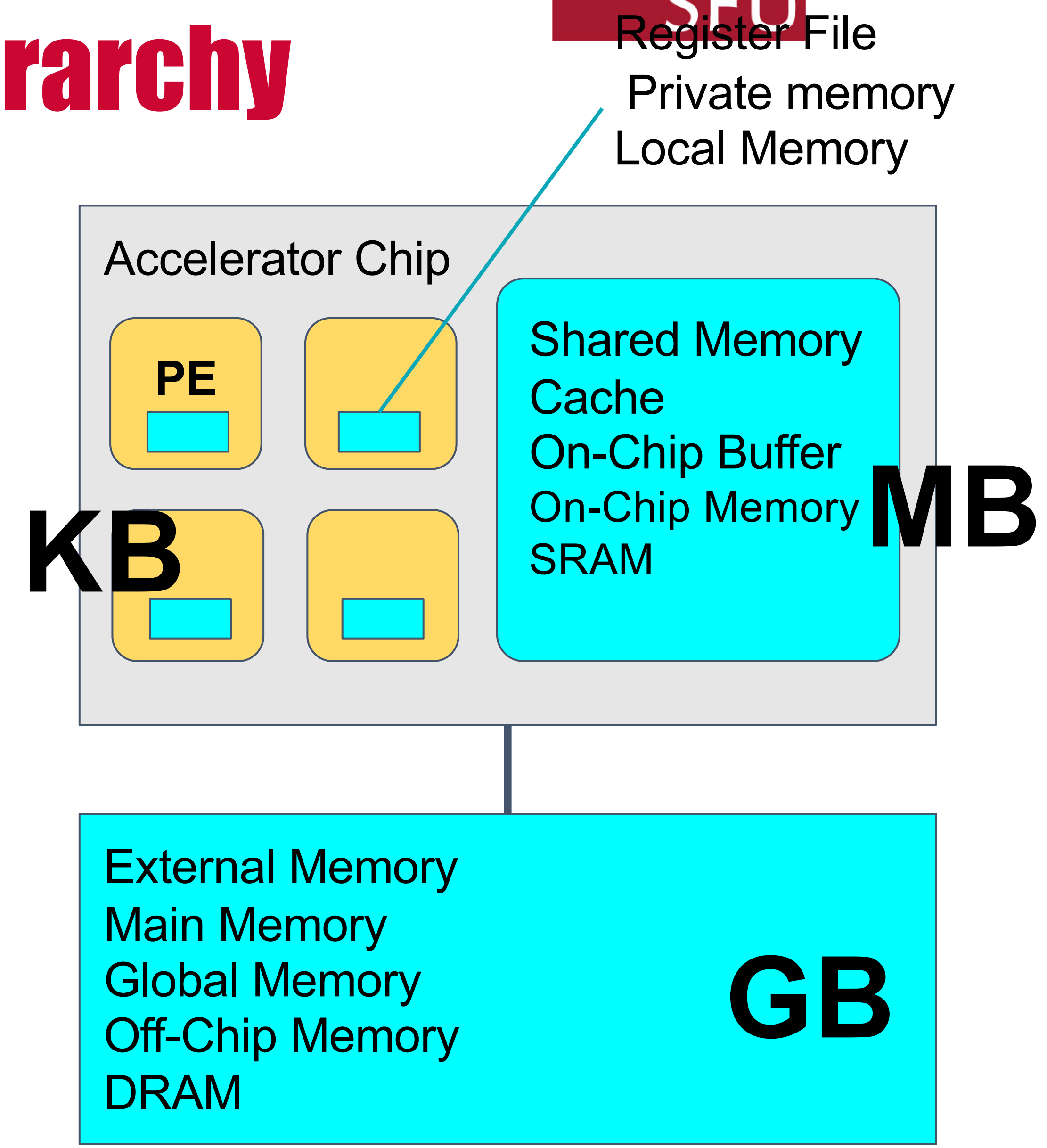
- Require modest memory bandwidth to keep full
- Internal data motion is hard wired
- Use all the local data for computation

Arbitrary new low-power compute operations Large effect on energy efficiency and performance

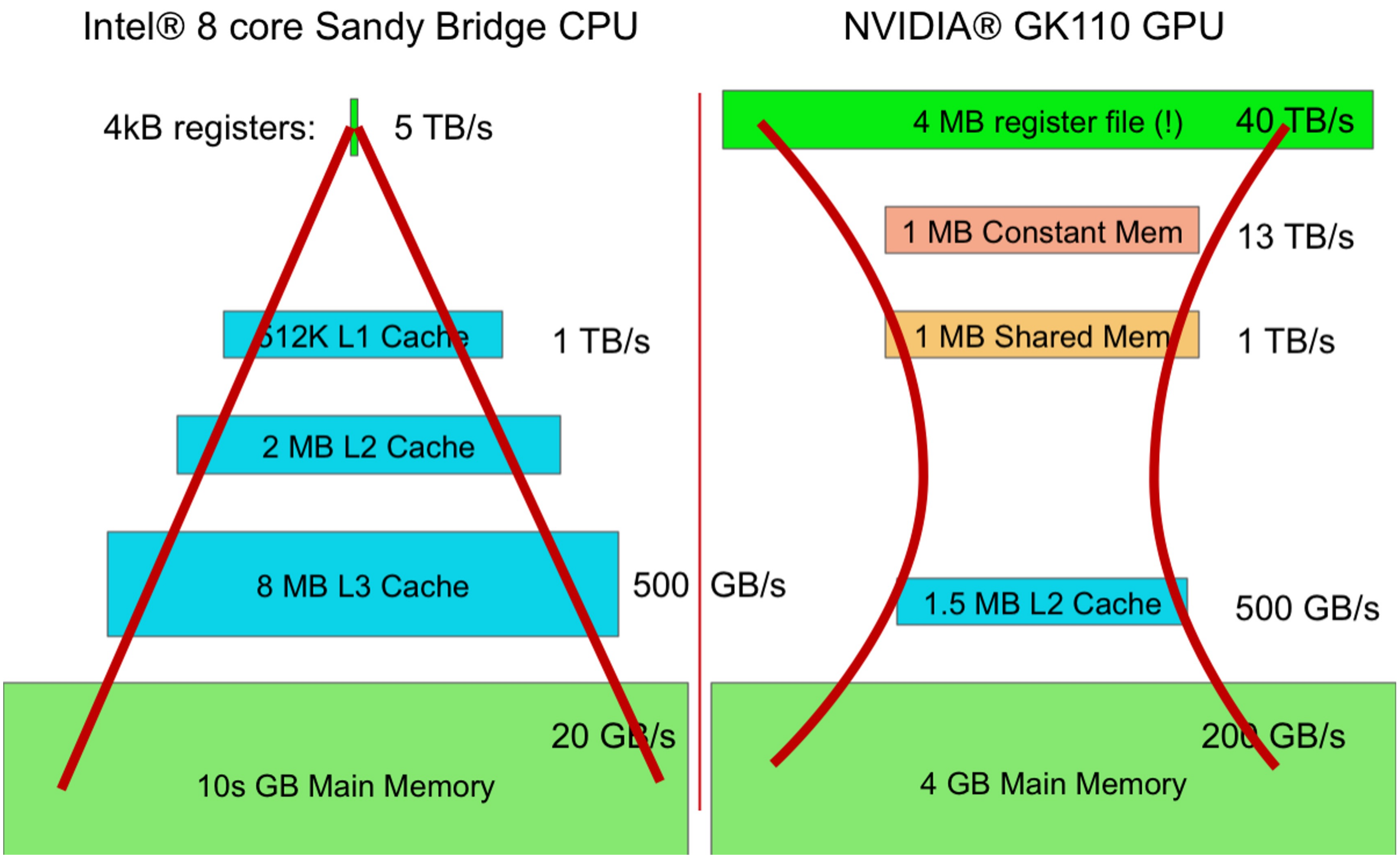
Memory Hierarchy

Why do we have a memory hierarchy?

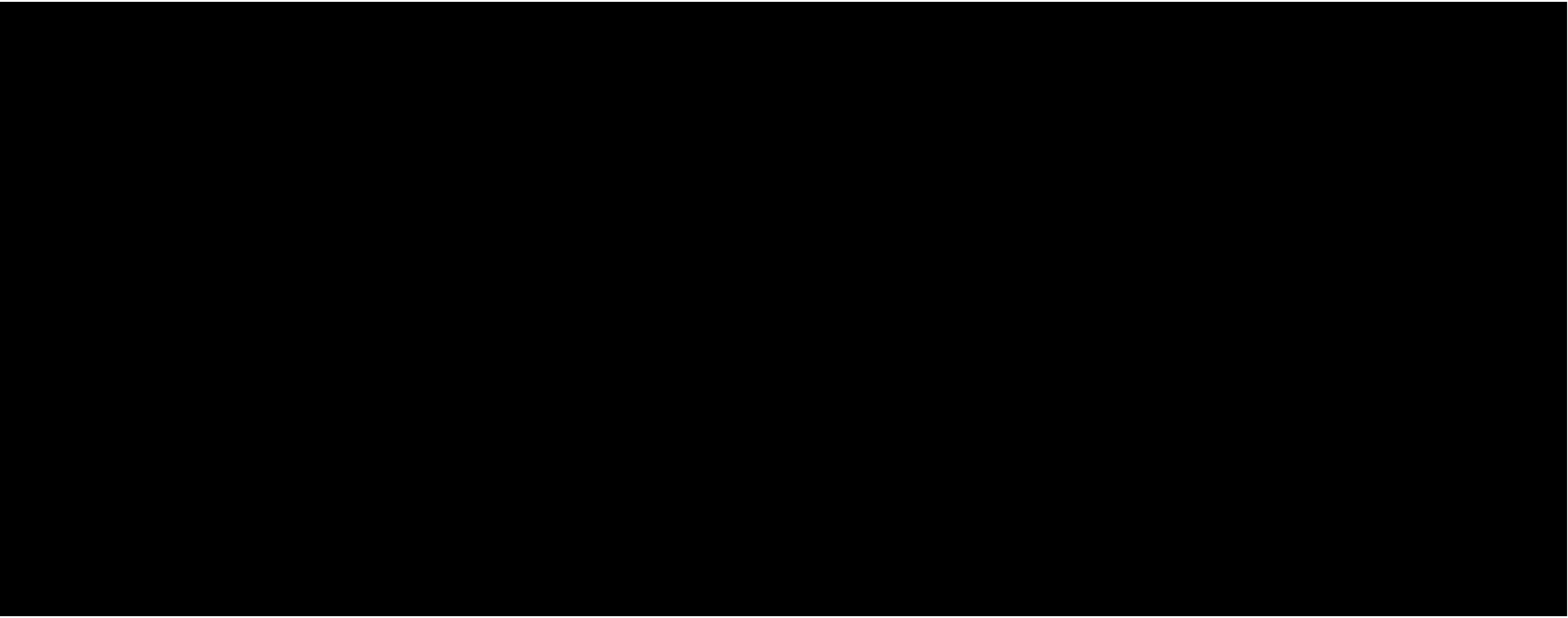
- The closer you get to compute, the more $$ and scarce the memory resource becomes
- In *most* cases, the DNN parameters live off chip and are fetched layer-by-layer or tile-by-tile
- Data locality: how to get data close to the PEs (to keep them fully utilized)



Memory Hierarchy Examples



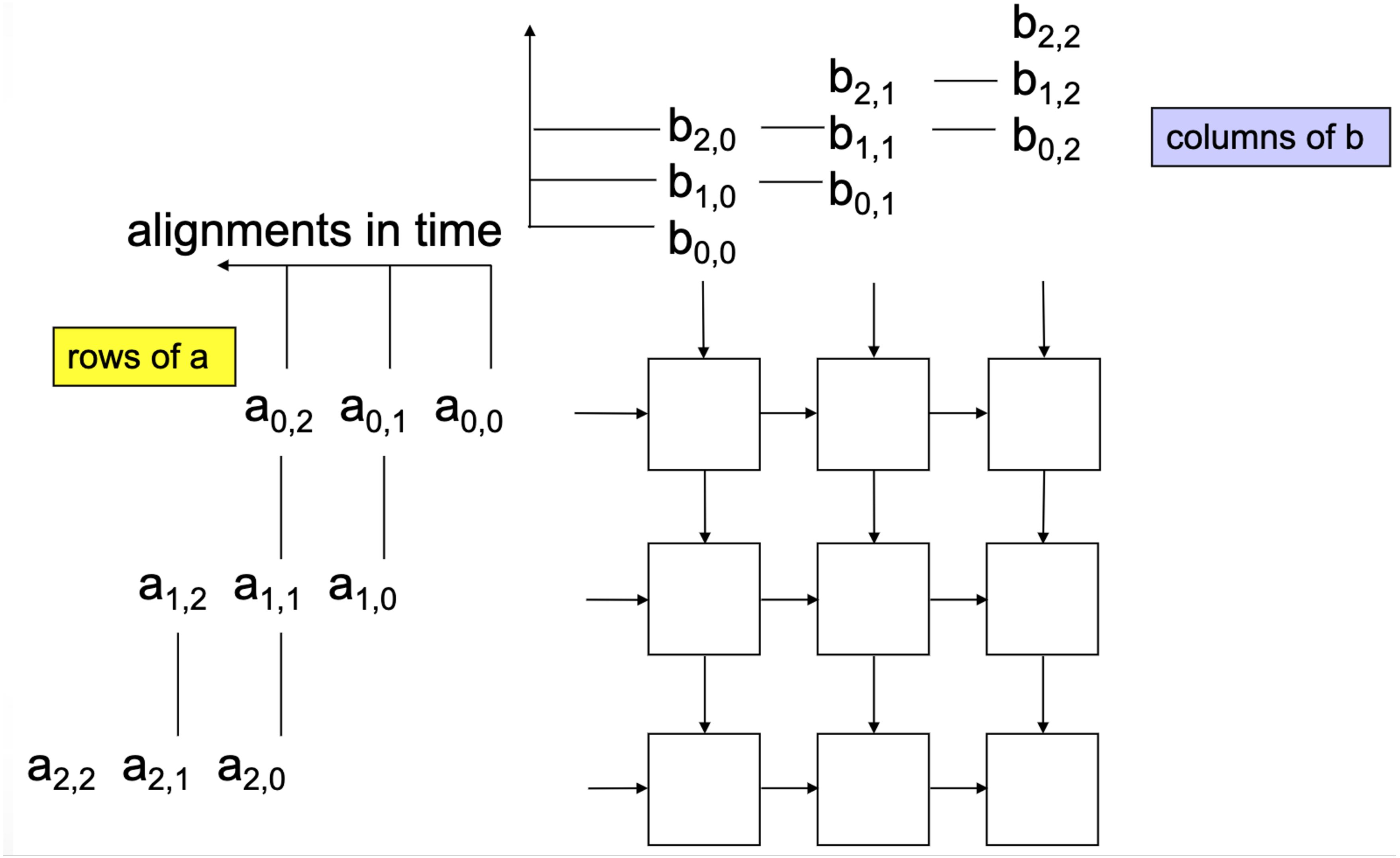
Source: Nvidia



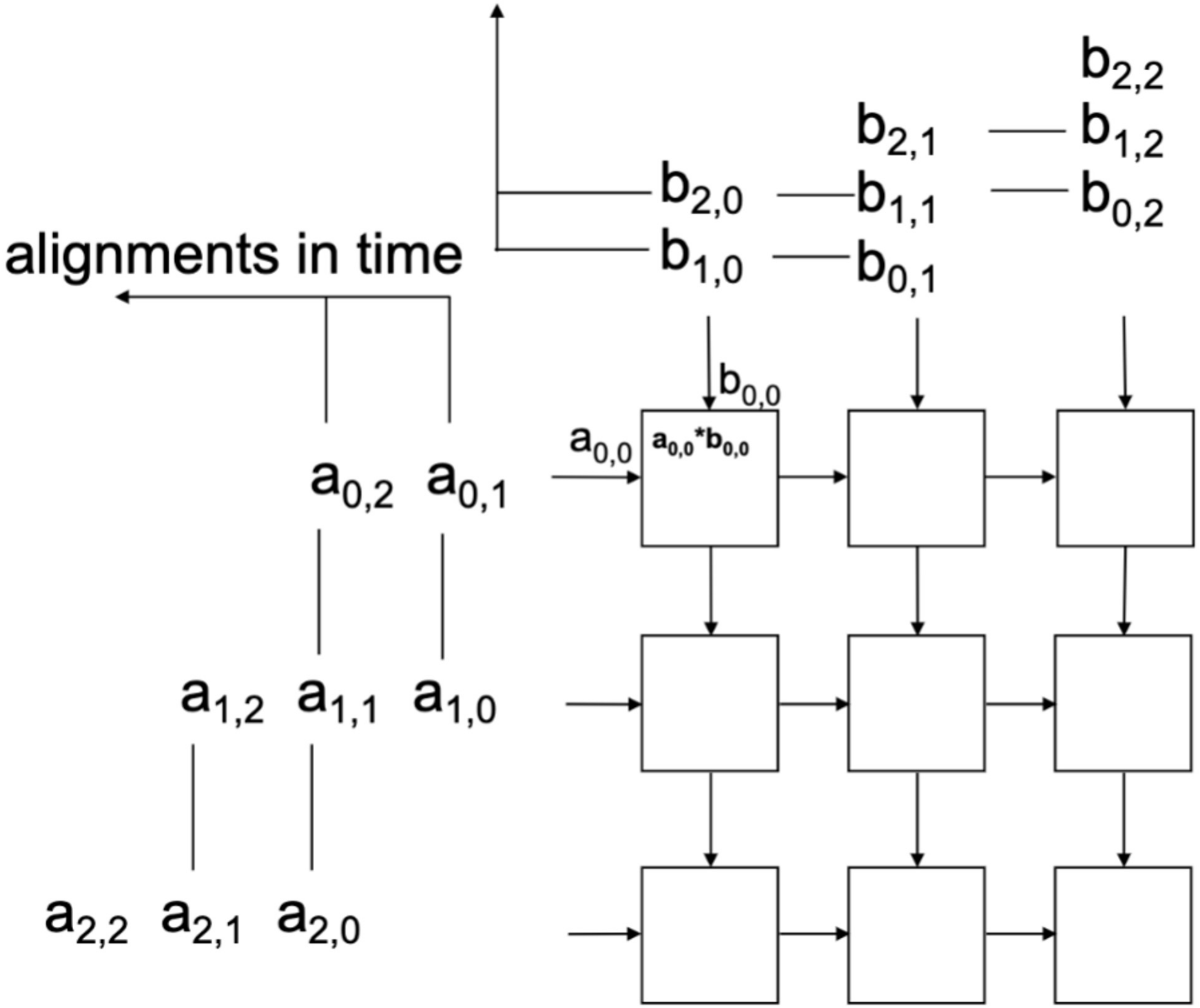
http://matrixmultiplication.xyz

```
function GEMM(alpha, A, B, beta, C)
for i = 0 to m − 1 # Loop over rows of A and C
  for j = 0 to n − 1 # Loop over columns of B and C
    for k = 0 to k − 1 # Loop over columns of A and rows of B
       temp = temp + A[i][k] ∗ B[k][j]
    end for
    temp = C[i][j]
   end for
end for
```

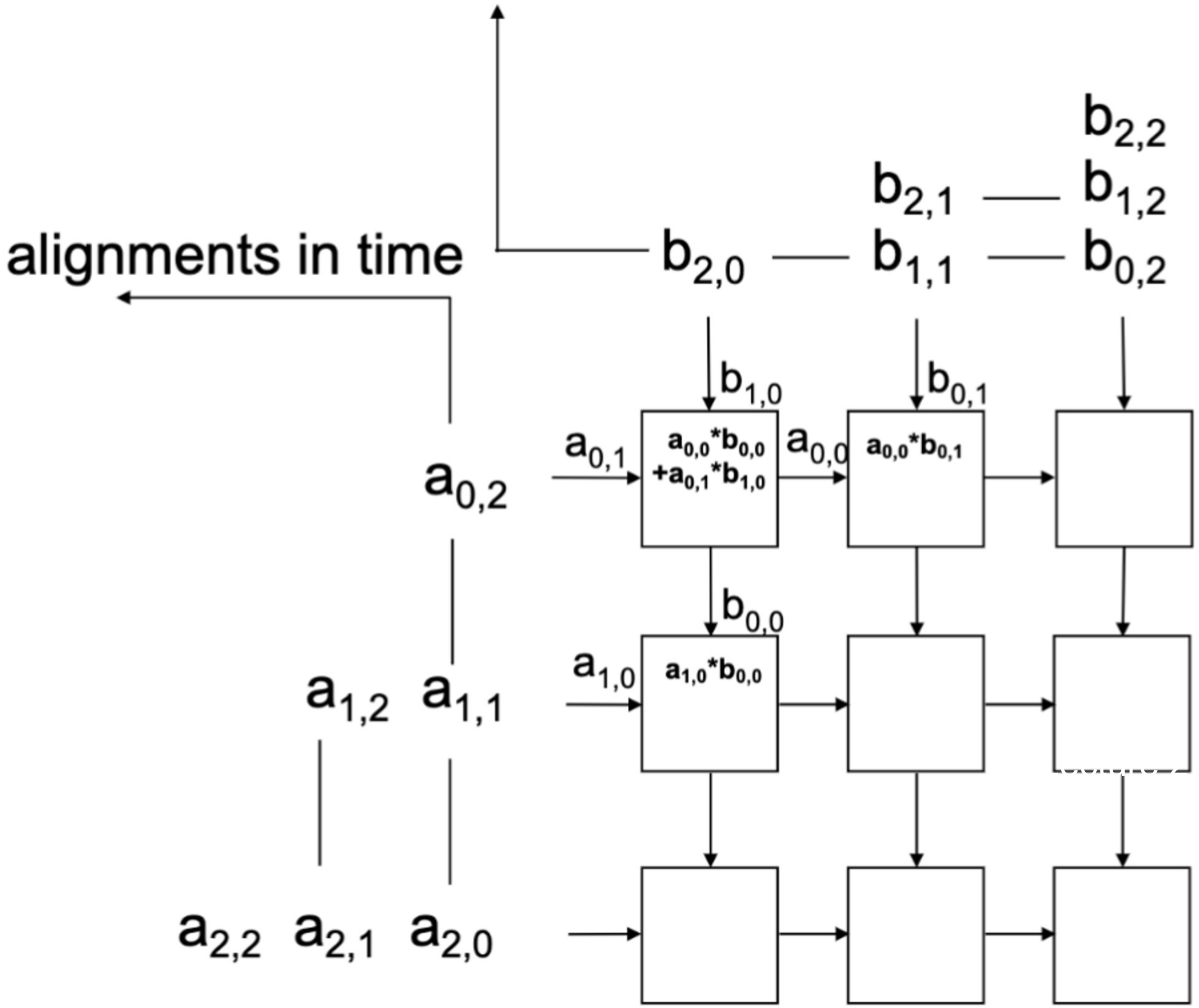
Systolic Array: Matrix Multiply Example



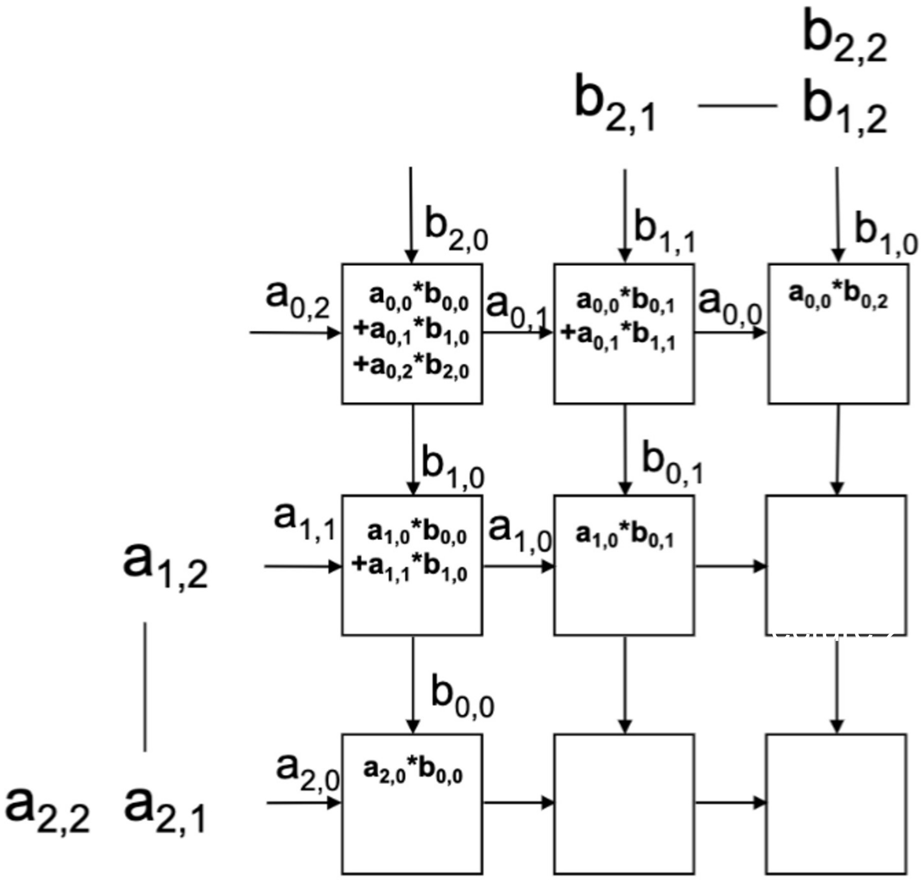
Systolic Array: Matrix Multiply Example



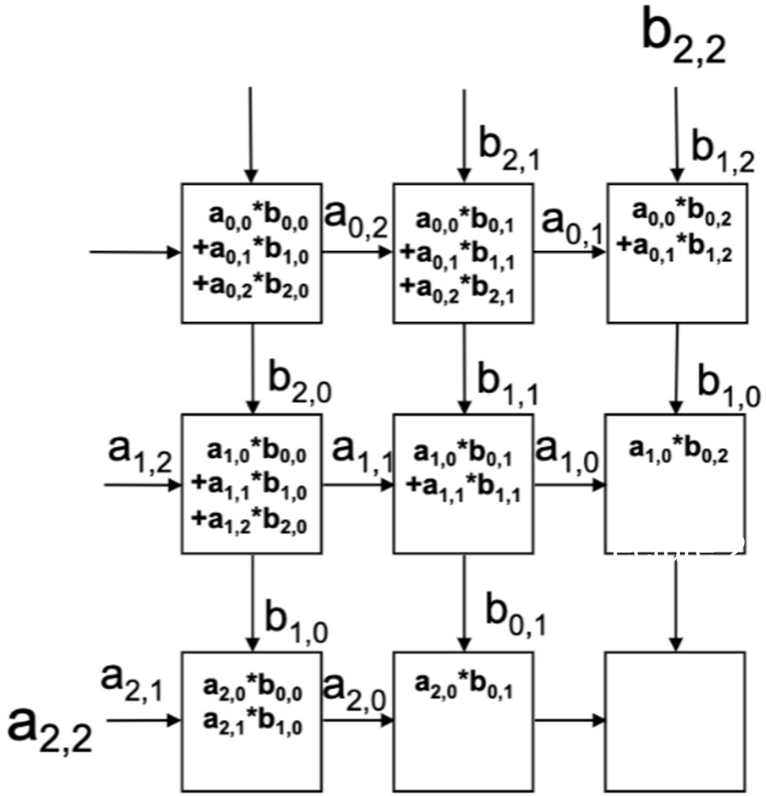
Systolic Array: Matrix Multiply Example



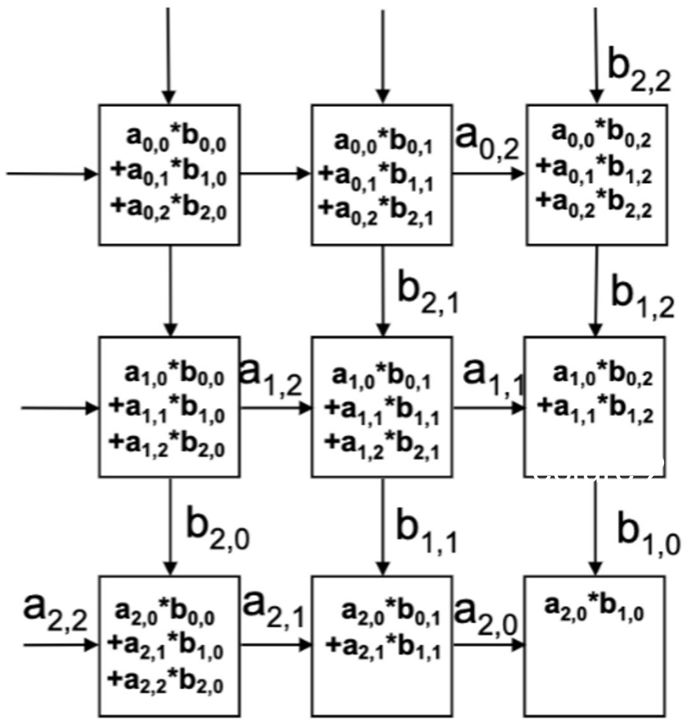
Systolic Array: Matrix Multiply Example



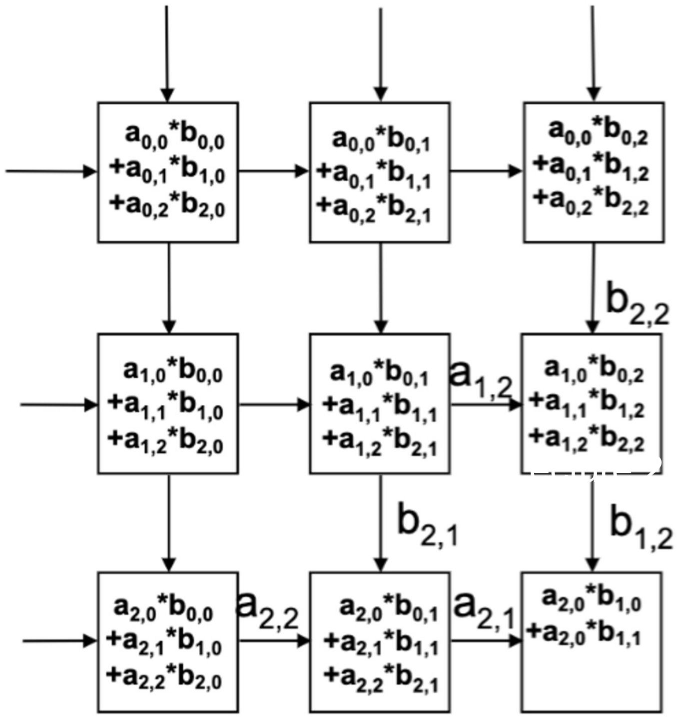
Systolic Array: Matrix Multiply Example



Systolic Array: Matrix Multiply Example

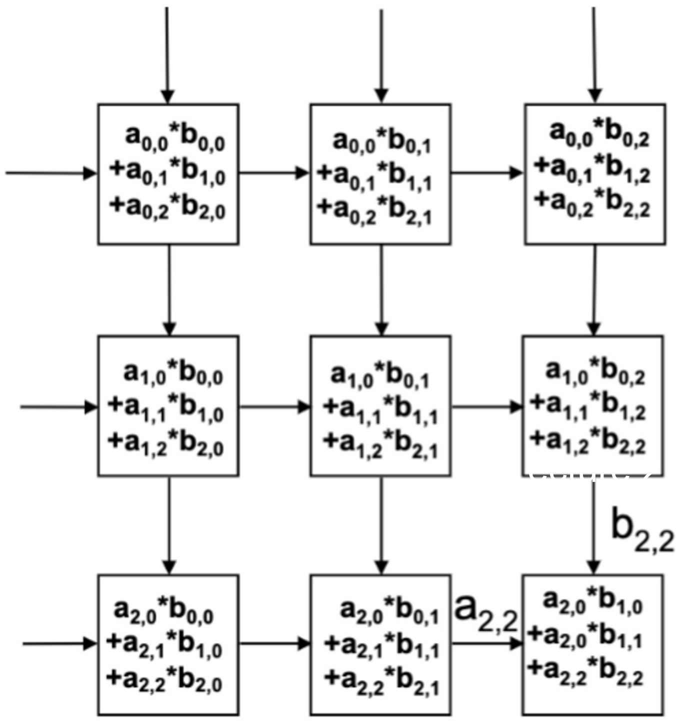


Systolic Array: Matrix Multiply Example



Systolic Array: Matrix Multiply Example

______ stationary?



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

Activation

5	0	1	2
3	1	0	1
0	8	4	4
9	0	0	1

X

Weight

2	0	1	2
-4	-1	3	0
0	0	3	2
0	0	-5	7

Activation Sparsity

Sparse activation functions (e.g. ReLU)

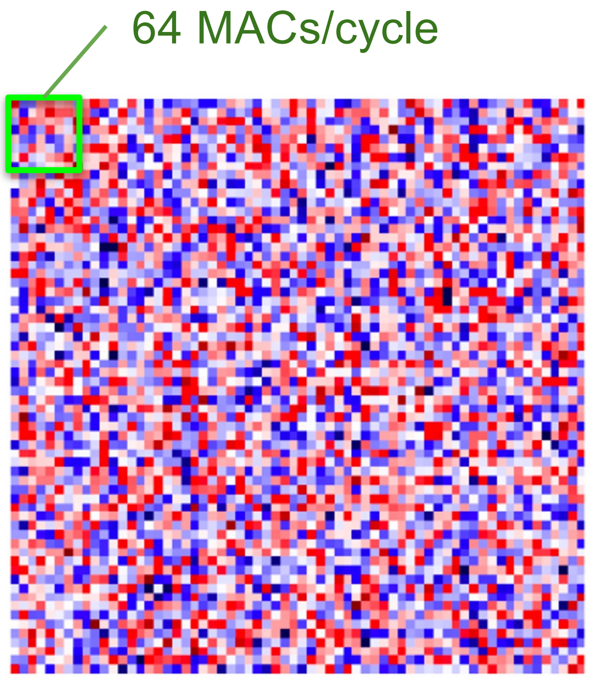
Weight Sparsity

Block Sparsity

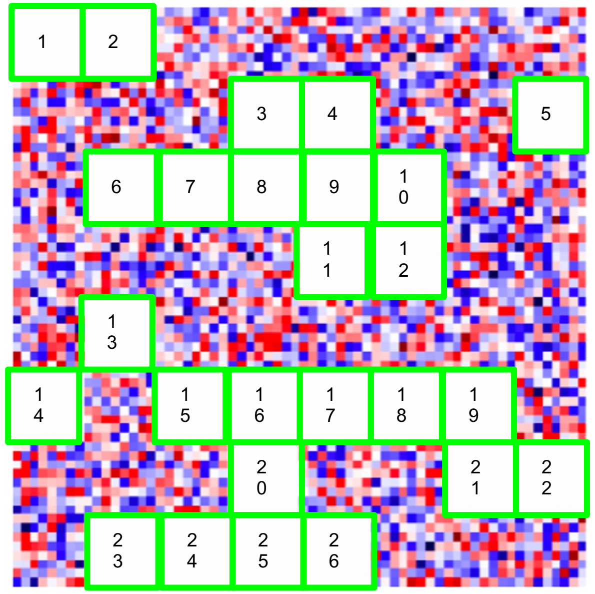
Pruning (covered in later lectures)

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

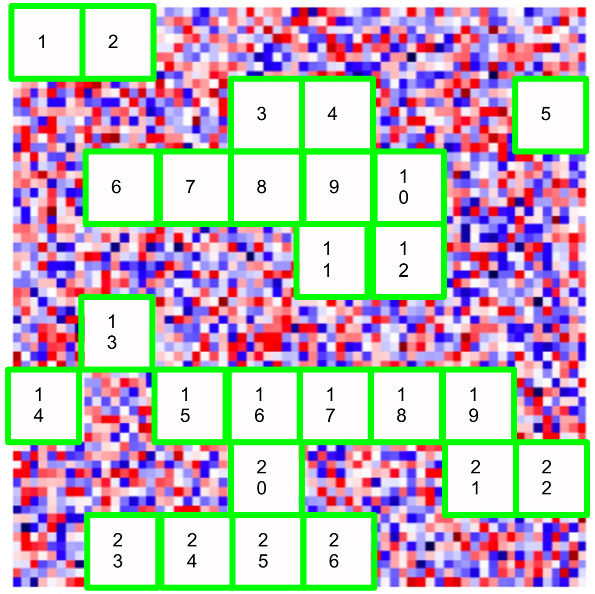
Coarse-grained “Block” Sparsity

- 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

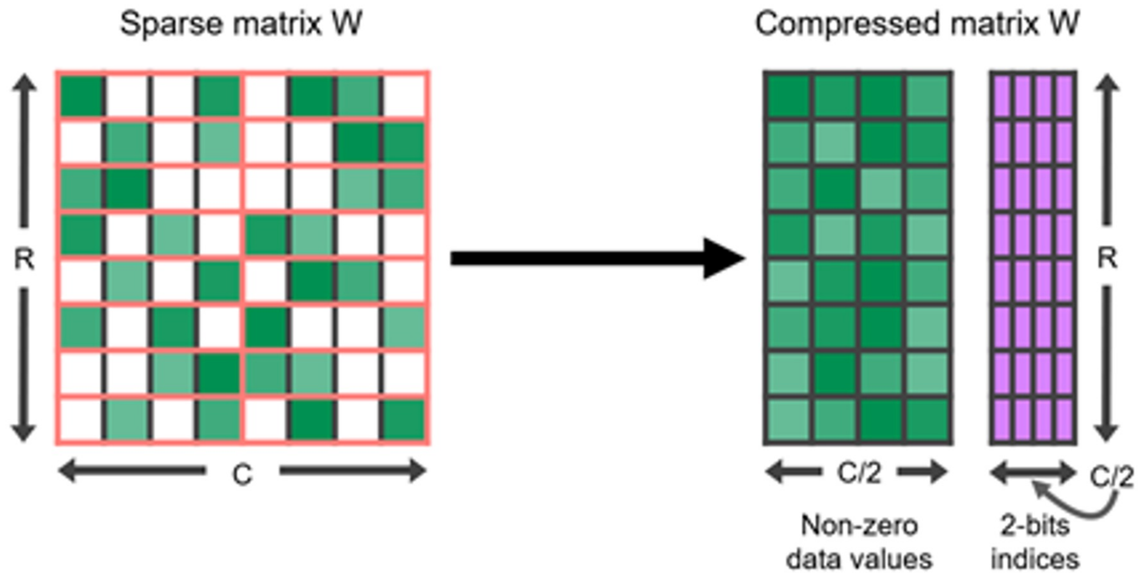
⇨ Single bit per 8x8 block (1/64 = 1.6% overhead)

⇨ Simple control logic because entire block is skipped

Fine-grained Sparsity in Ampere GPUs

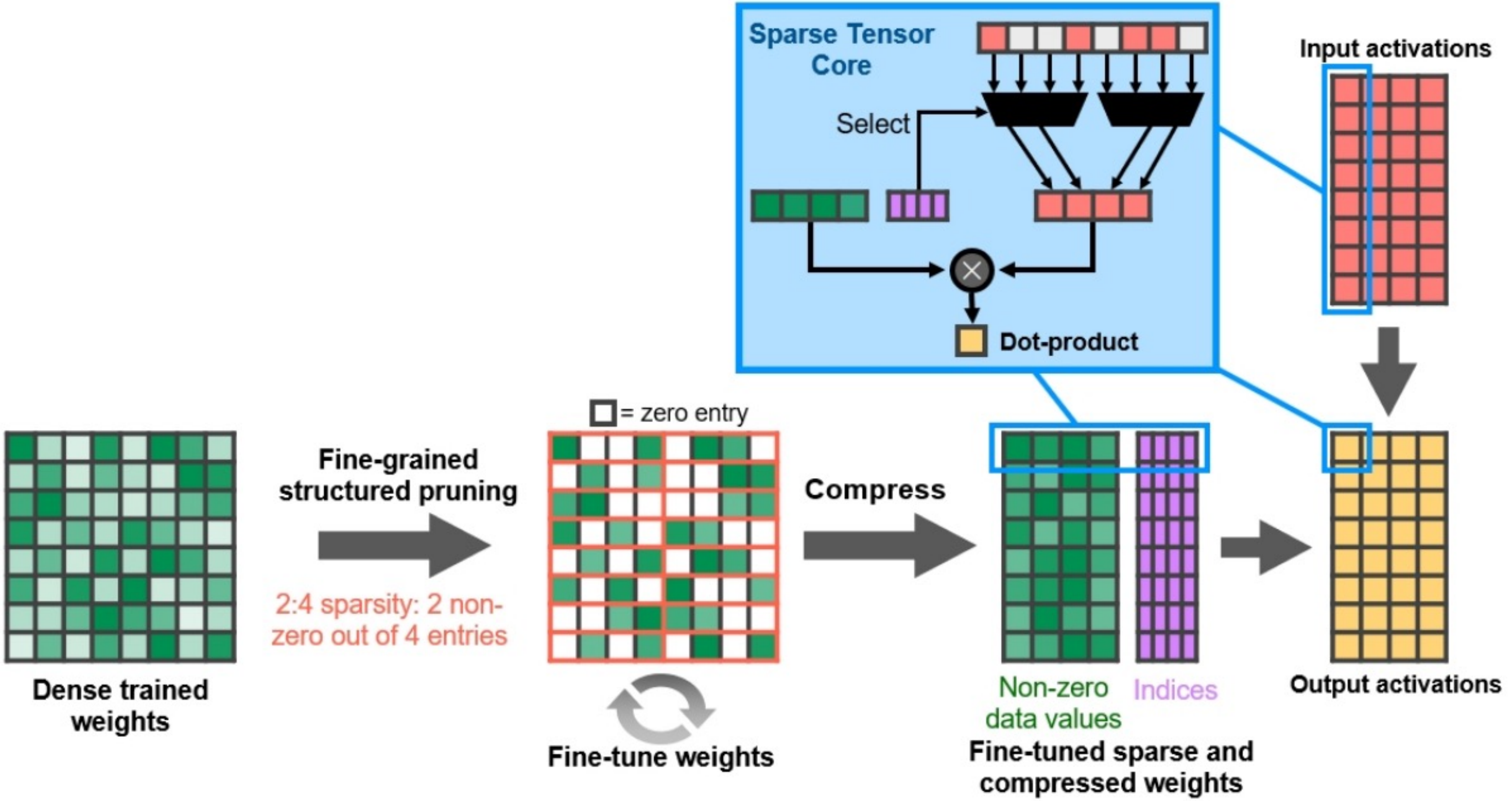
SFU

- 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

Nvidia sparsity support



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