Modern Heterogeneous Computers: (1) Computing Units

Hung-Wei Tseng

Heterogeneous computing

- The computer system contains multiple types of processing units
  - CPU
  - GPU
  - TPU
  - FPGA

Heterogeneous computing

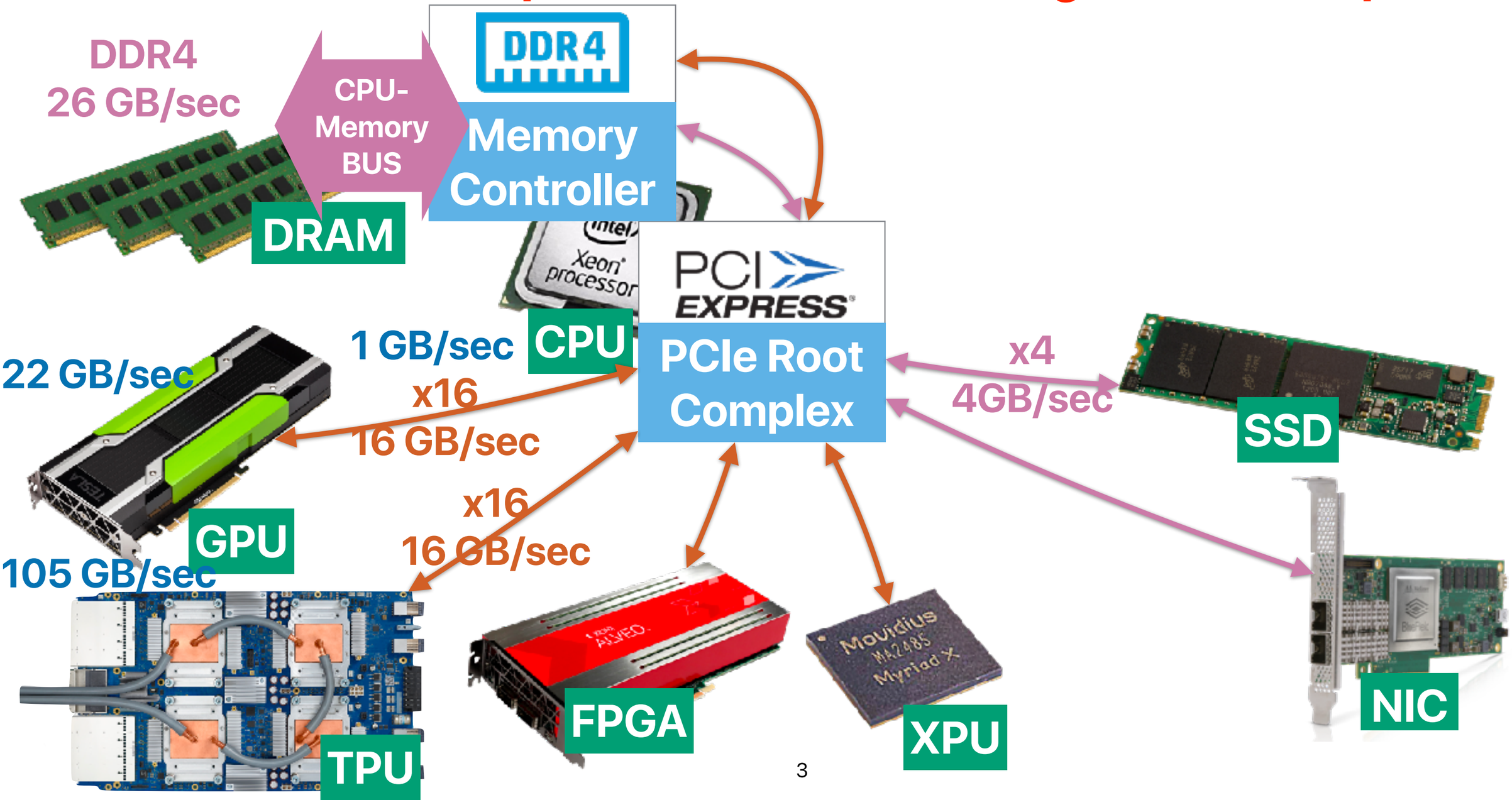
From Wikipedia, the free encyclopedia

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Heterogeneous computing" - news · newspapers · books · scholar · JSTOR *(October 2014)*
(Learn how and when to remove this template message)

Heterogeneous computing refers to systems that use more than one kind of processor or cores. These systems gain performance or energy efficiency not just by adding the same type of processors, but by adding dissimilar coprocessors, usually incorporating specialized processing capabilities to handle particular tasks.[1]

https://en.wikipedia.org/wiki/Heterogeneous_computing

Review: The "data path" in modern heterogeneous computers



3:00

Why computers become heterogeneous?

Why heterogeneous?

- Because the CPU performance is below the demand of applications
- Why not making CPUs faster? — Because we can’t!

The limitation of “conventional” von Neumann Architecture

The “conventional” von Neumman Architecture

509cbd23

00c2e800

Processor

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

Memory

Program

Instructions	Data
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

Storage

Review: How fast is a CPU?

https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Review: How fast is a CPU?

- Clock rate: 3 GHz — 3B cycles/instructions per "ALU"
- 3 floating point ALUs per core
- 9B floating point operations per second
- If we can "perfectly" parallelize a program using all 6 cores — 54 B/G floating point operations

Finding the limitation — the roofline model of CPU processing

Determining factors of performance

- The speed of computation — determined by the "OPS" (operations per second) of the processing unit
- The speed of data supply — determined by the "effective bandwidth" of the data path
- At any point, the slower one determines the performance
  - $$\text{Attainable GFlops/sec} = min \left\{ \begin{array}{l} \text{Peak Floating-Point Performance} \\ \text{Peak Memory Bandwidth} \times \text{Operational Intensity} \end{array} \right.$$

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 4 (April 2009), 65–76.

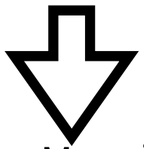
Arithmetic/operational Intensity

- The amount of computation/operations required for each byte of data
- Changes as the algorithm changes

Example — floating point matrix multiplications

- Performing $M \times N \times P$ matrix multiplications
  - The size of a is $M \times N$
  - The size of b is $N \times P$
  - The size of c is $M \times P$
- Total amount of operations = $2 \times M \times N \times P$ ops
- Total size of data accesses = $4 \times (2 \times N + 1) \times M \times P$ bytes
- Average operations per byte = $$\frac{2 \times M \times N \times P}{4 \times (2 \times N + 1) \times M \times P} = \frac{1}{8}$$

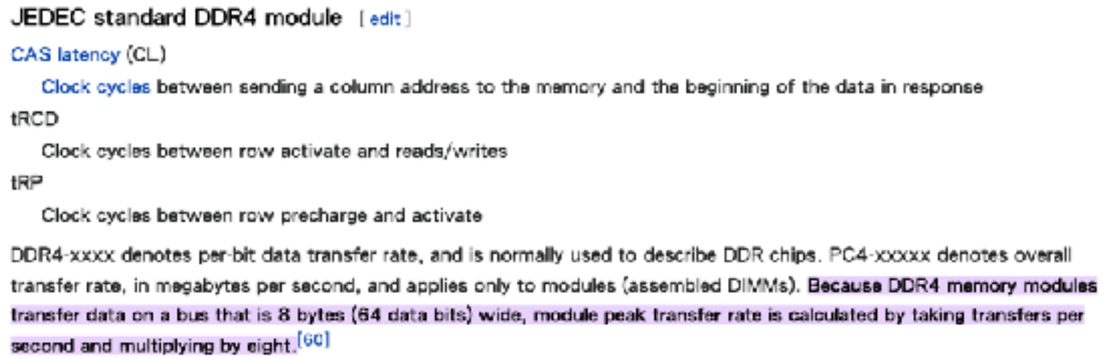
```
for(i = 0; i < M; i++) {
  for(j = 0; j < P; j++) {
    for(k = 0; k < N; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```



```
for(i = 0; i < M; i++) {
  for(j = 0; j < P; j++) {
    sum = 0.0;
    for(k = 0; k < N; k++) {
      sum += a[i][k]*b[k][j];
    }
    c[i][j] = sum;
  }
}
```

Example — matrix multiplications (cont.)

- Remember that we have a CPU core supporting 9G operations per second (OPS)
- If data are currently stored in SSD
  - We can supply data in 4GB/sec
  - The supplied data per second needs
    $$4GB \times \frac{1\ OPS}{8\ B} = 0.5\ GOPS < 9G$$
- If data are currently stored in DRAM
  - We can supply data in 25GB/sec per module
  - The supplied data per second needs per module
    $$25GB \times \frac{1\ OPS}{8\ B} = 3.12500\ GOPS < 9G$$
  - If we have 2 modules
    $$50GB \times \frac{1\ OPS}{8\ B} = 7.25\ GOPS < 9G$$



JEDEC standard DDR4 module [edit]

CAS latency (CL)

Clock cycles between sending a column address to the memory and the beginning of the data in response

tRCD

Clock cycles between row activate and reads/writes

tRP

Clock cycles between row precharge and activate

DDR4-xxxx denotes per-bit data transfer rate, and is normally used to describe DDR chips. PC4-xxxxx denotes overall transfer rate, in megabytes per second, and applies only to modules (assembled DIMMs). Because DDR4 memory modules transfer data on a bus that is 8 bytes (64 data bits) wide, module peak transfer rate is calculated by taking transfers per second and multiplying by eight.[60]

Size, Latency, and Bandwidth of Memory Subsystem Components

Assuming you have a large processor (about 16 cores), the following summarizes, for 2016, approximate data totals present in and moving through the system.

Memory	Size	Latency	Bandwidth
L1 cache	32 KB	1 nanosecond	1 TB/second
L2 cache	256 KB	4 nanoseconds	1 TB/second Sometimes shared by two cores
L3 cache	8 MB or more	10x slower than L2	>400 GB/second
MCDRAM		2x slower than L3	400 GB/second
Main memory on DDR DIMMs	4 GB-1 TB	Similar to MCDRAM	100 GB/second
Main memory on Cornelis* Omni-Path Fabric	Limited only by cost	Depends on distance	Depends on distance and hardware
I/O devices on memory bus	6 TB	100x-1000x slower than memory	25 GB/second
I/O devices on PCIe bus	Limited only by cost	From less than milliseconds to minutes	GB-TB/hour Depends on distance and hardware

https://www.intel.com/content/www/us/en/developer/articles/technical/memory-performance-in-a-nutshell.html

Example — matrix multiplications (cont.)

- Remember that we have a CPU core supporting 9G operations per second (OPS)
- If data are currently stored in SSD
  - We can supply data in 4GB/sec
  - The supplied data per second needs
    $4GB/s \times \frac{1\ OPS}{8\ B} = 0.5\ GOPS < 9G$
- If data are currently stored in DRAM
  - We can supply data in 25GB/sec per module
  - The supplied data per second needs per module
    $25GB/s \times \frac{1\ OPS}{8\ B} = 3.12500\ GOPS < 9G$
  - If we have 2 modules
    $50GB/s \times \frac{1\ OPS}{8\ B} = 7.25\ GOPS < 9G$
- If we can use cache "perfectly"
  - $400GB/s \times \frac{1\ OPS}{8\ B} = 50\ GOPS > 9G$

JEDEC standard DDR4 module [edit]

CAS latency (CL)

Clock cycles between sending a column address to the memory and the beginning of the data in response

tRCD

Clock cycles between row activate and reads/writes

tRP

Clock cycles between row precharge and activate

DDR4-xxxx denotes per-bit data transfer rate, and is normally used to describe DDR chips. PC4-xxxxx denotes overall transfer rate, in megabytes per second, and applies only to modules (assembled DIMMs). **Because DDR4 memory modules transfer data on a bus that is 8 bytes (64 data bits) wide, module peak transfer rate is calculated by taking transfers per second and multiplying by eight.**[60]

The roofline under various configurations

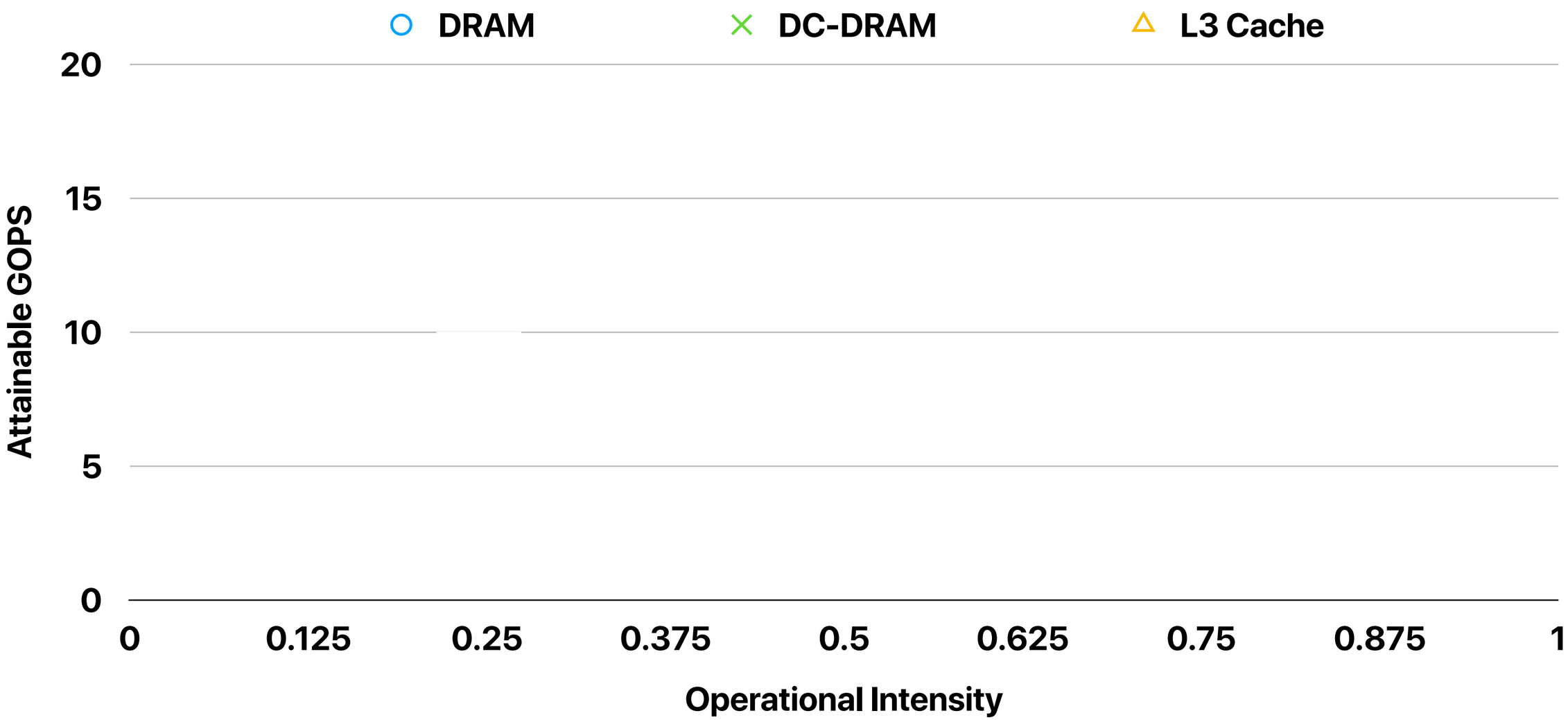
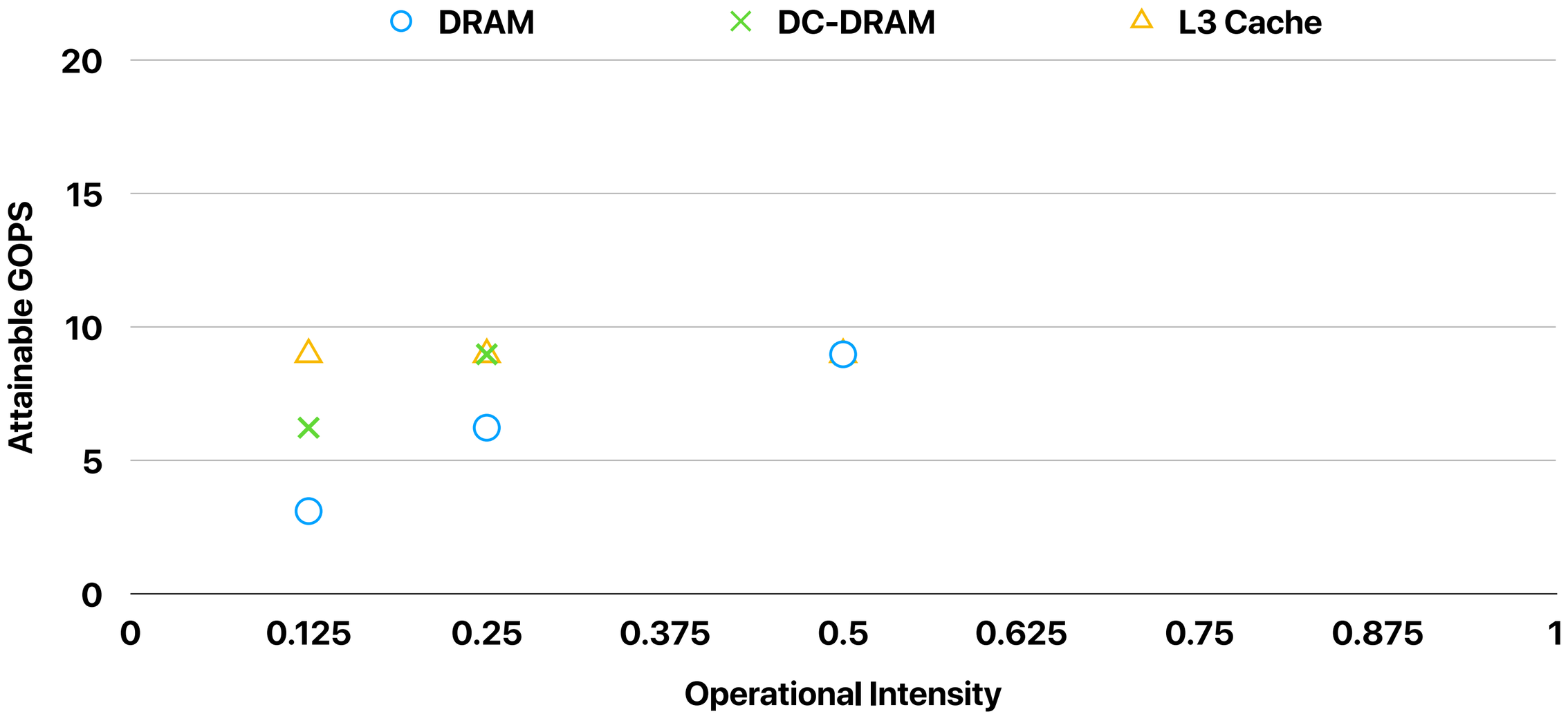


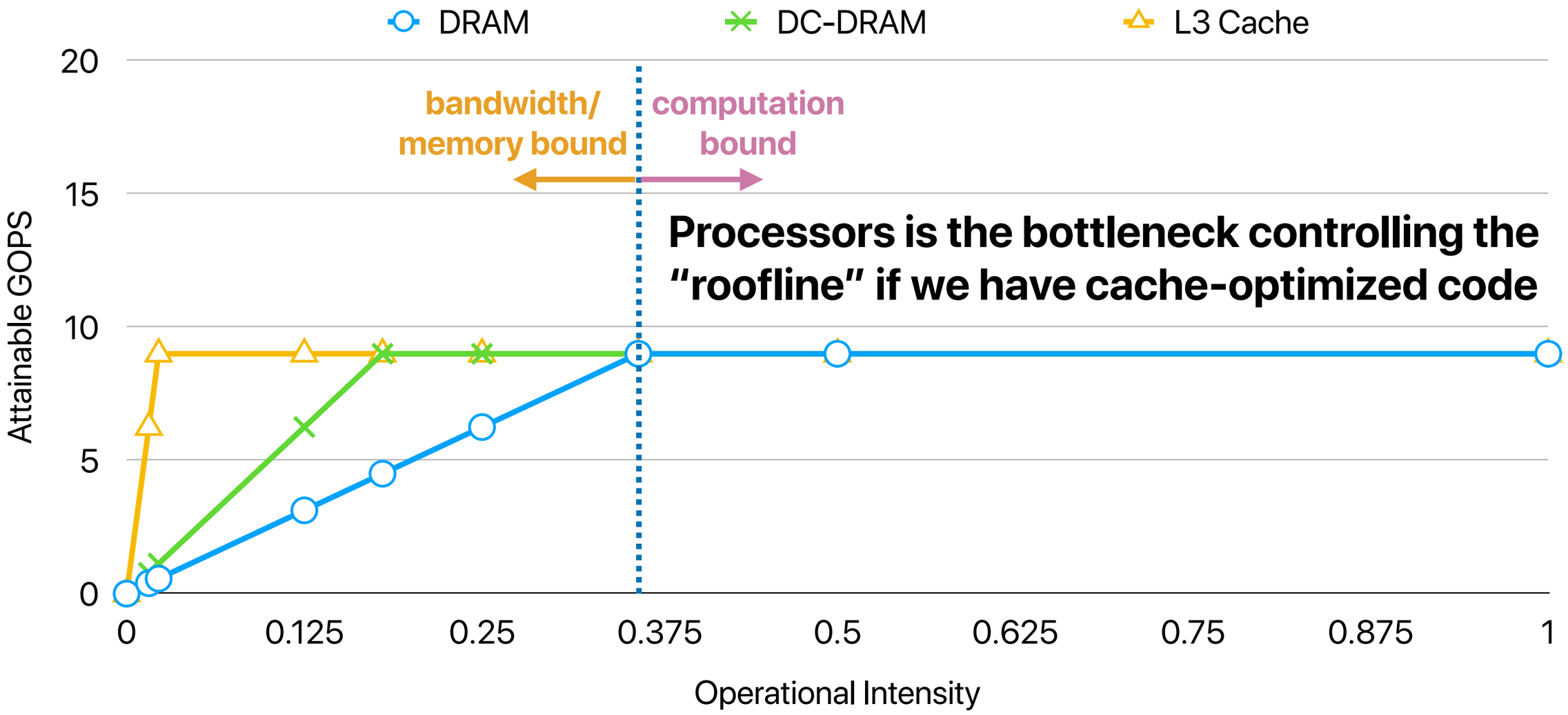
Table 2: Characteristics of four floating-point kernels.

Name	Operational Intensity	Description
SpMV[29]	0.17 to 0.25	Sparse Matrix-Vector multiply: y = A*x where A is a sparse matrix and x, y are dense vectors; multiplies and adds equal.
LBMHD[28]	0.70 to 1.07	Lattice-Boltzmann Magnetohydro-dynamics is a structured grid code with a series of time steps.
Stencil[12]	0.33 to 0.50	A multigrid kernel that updates seven nearby points in a 3D stencil for a 256^3 problem.
3D FFT	1.09 to 1.64	3D Fast Fourier Transform (2 sizes: 128^3 and 512^3).

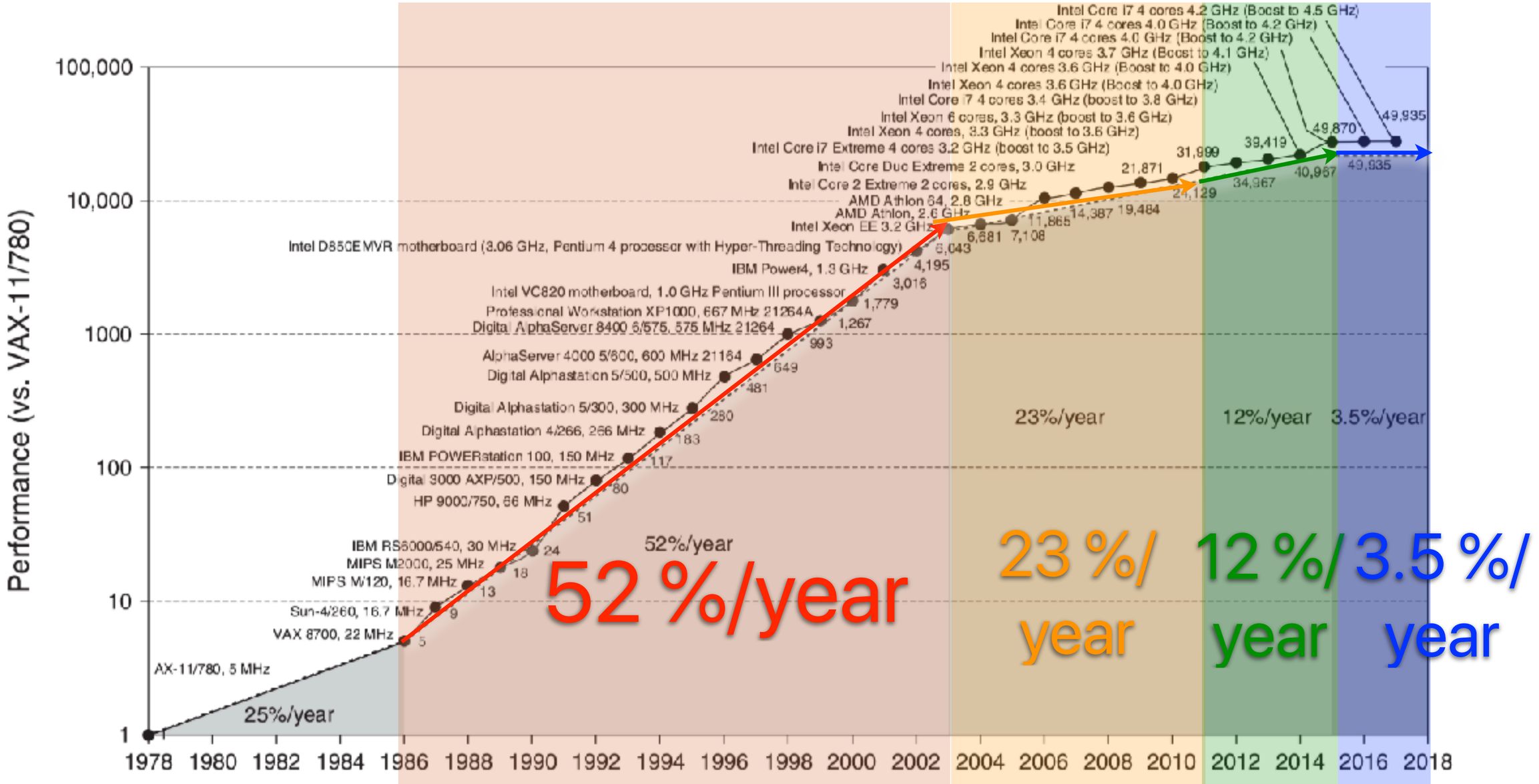
The roofline under various configurations



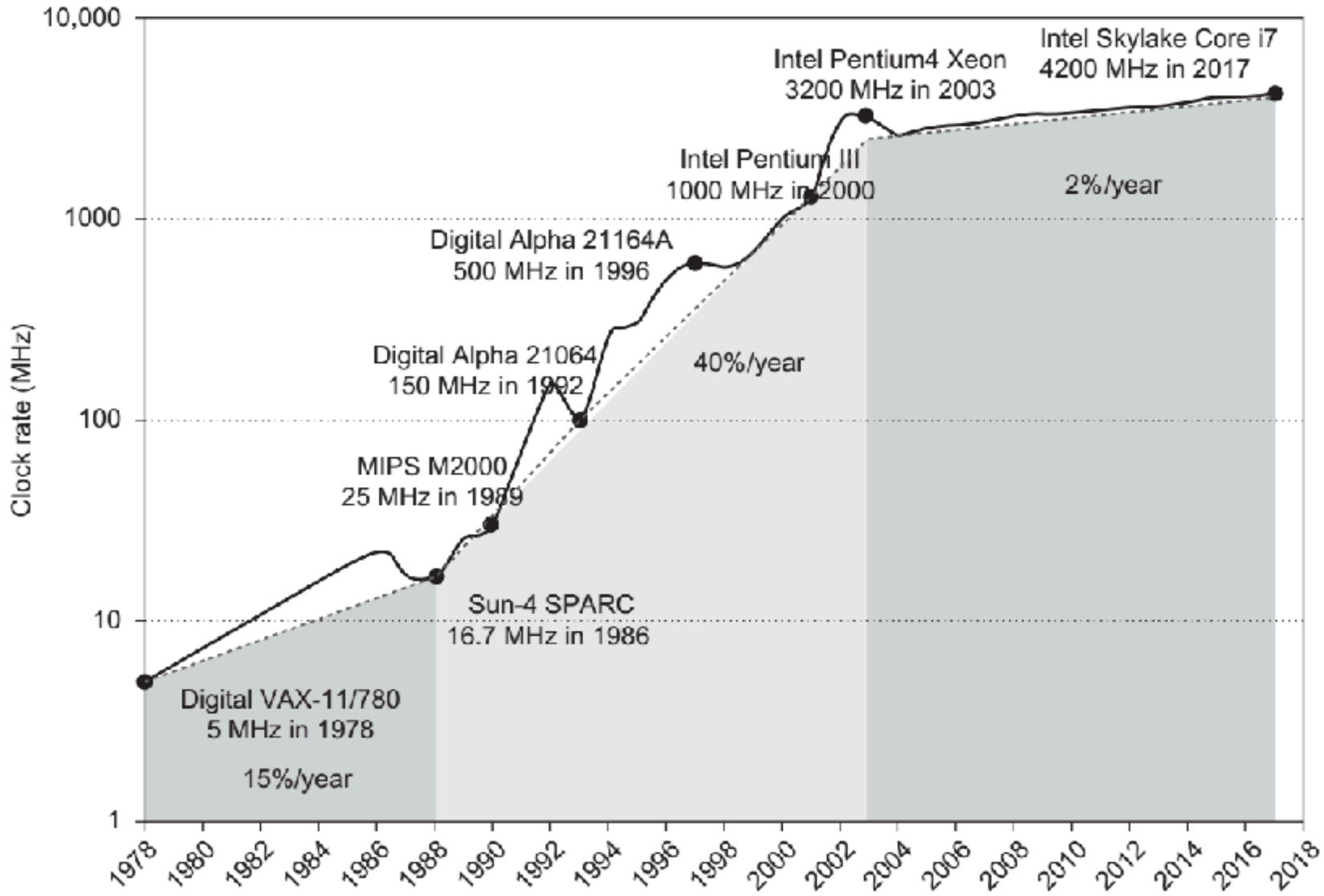
The roofline under various configurations



CPU defines the “ceiling” of the roofline…



Clock rate improvement is limited nowadays



Why is it?

Static/Leakage Power

- The power consumption due to leakage — transistors do not turn all the way off during no operation
- Becomes the **dominant** factor in the most advanced process technologies.

$$P_{leakage} \sim N \times V \times e^{-V_t}$$

  - N: number of transistors
  - V: voltage
  - V_t: threshold voltage where transistor conducts (begins to switch)

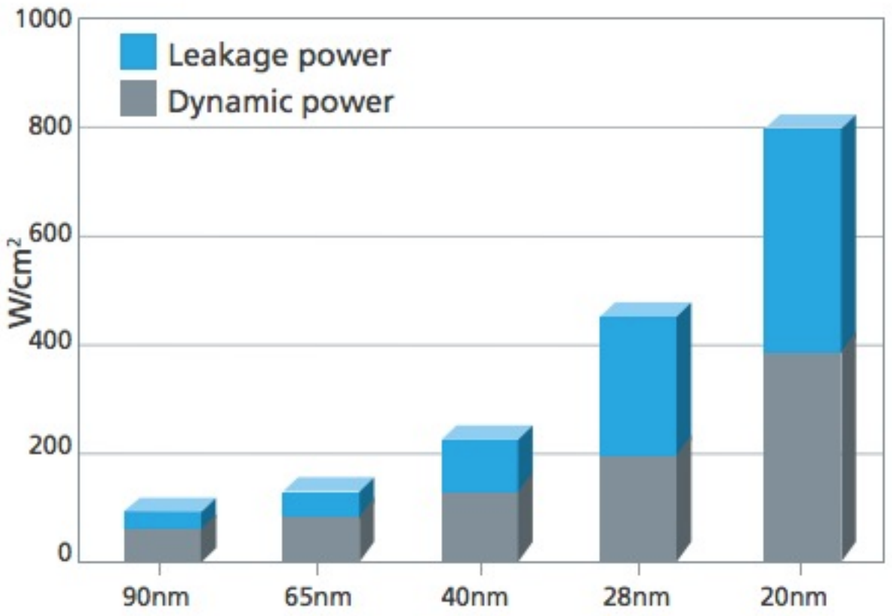


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Dennardian Broken

- Given a scaling factor S

Parameter	Relation	Classical Scaling	Leakage Limited
Power Budget		1	1
Chip Size		1	1
Vdd (Supply Voltage)		1/S	1
Vt (Threshold Voltage)	1/S	1/S	1
tex (oxide thickness)		1/S	1/S
W, L (transistor dimensions)		1/S	1/S
Cgate (gate capacitance)	WL/tox	1/S	1/S
Isat (saturation current)	WVdd/tox	1/S	1
F (device frequency)	Isat/(CgateVdd)	S	S
D (Device/Area)	1/(WL)	S^2	S^2
p (device power)	IsatVdd	$1/S^2$	1
P (chip power)	Dp	1	S^2
U (utilization)	1/P	1	$1/S^2$

Power consumption to light on all transistors

Chip

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

=49W

Dennardian Scaling

Chip

0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

=50W

Dennardian Broken

Chip

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

On ~ 50W

Off ~ 0W

Dark!

=100W!

3:00

What are the limitations enforced by the brokenness of Dennardian Scaling?

Implications from the brokenness of Dennardian Scaling

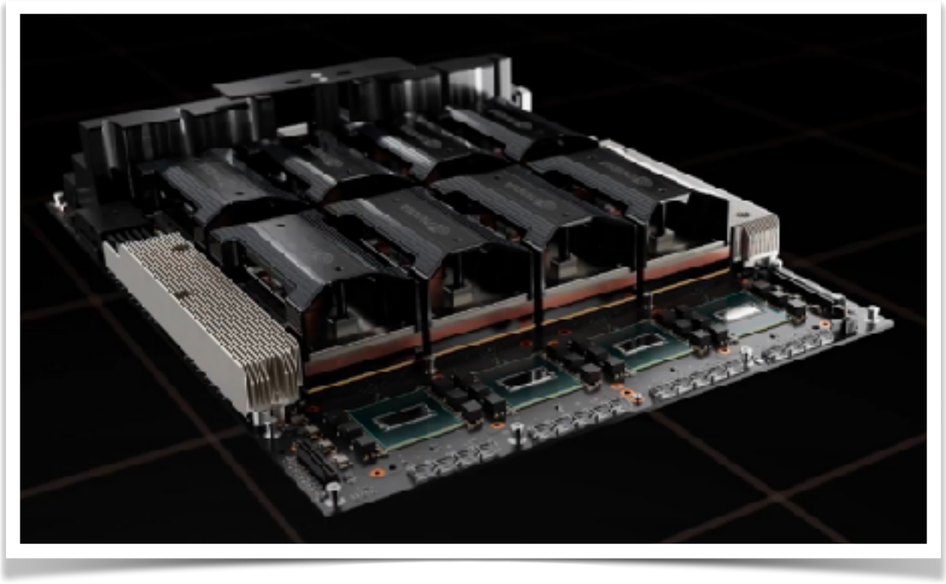
Implications from the brokenness of Dennardian Scaling

- Given the same power budget
  - Implication 1 — You can always have the same amount of circuits in a chip, regardless how Moore's Law reduces the die size
  - Implication 2 — You can put more circuits in, but you cannot activate all of them all the same time — Dark Silicon Problem
- If you want to activate more functional units at the same time, you have to raise the power budget/consumption and the thermal design power (TDP)

If you can add power budget…

NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
Boost Clock	~1.78GHz (Not Finalized)	1.41GHz	1.53GHz
Memory Clock	4.8Gbps HBM3	3.2Gbps HBM2e	1.75Gbps HBM2
Memory Bus Width	5120-bit	5120-bit	4096-bit
Memory Bandwidth	3TB/sec	2TB/sec	900GB/sec
VRAM	80GB	80GB	16GB/32GB
FP32 Vector	60 TFLOPS	19.5 TFLOPS	15.7 TFLOPS
FP64 Vector	30 TFLOPS	9.7 TFLOPS (1/2 FP32 rate)	7.8 TFLOPS (1/2 FP32 rate)
INT8 Tensor	2000 TOPS	624 TOPS	N/A
FP16 Tensor	1000 TFLOPS	312 TFLOPS	125 TFLOPS
TF32 Tensor	500 TFLOPS	156 TFLOPS	N/A
FP64 Tensor	60 TFLOPS	19.5 TFLOPS	N/A
Interconnect	NVLink 4 18 Links (900GB/sec)	NVLink 3 12 Links (600GB/sec)	NVLink 2 6 Links (300GB/sec)
GPU	GH100 (814mm2)	GA100 (826mm2)	GV100 (815mm2)
Transistor Count	80B	54.2B	21.1B
TDP	700W	400W	300W/350W
Manufacturing Process	TSMC 4N	TSMC 7N	TSMC 12nm FFN
Interface	SXM5	SXM4	SXM2/SXM3
Architecture	Hopper	Ampere	Volta

https://www.workstationspecialist.com/product/nvidia-tesla-a100/



https://www.servethehome.com/wp-content/uploads/2022/03/NVIDIA-GTC-2022-H100-in-HGX-H100.jpg

3:00

What are the potential “work-arounds” to improve performance?

If we don't want to add power budget…

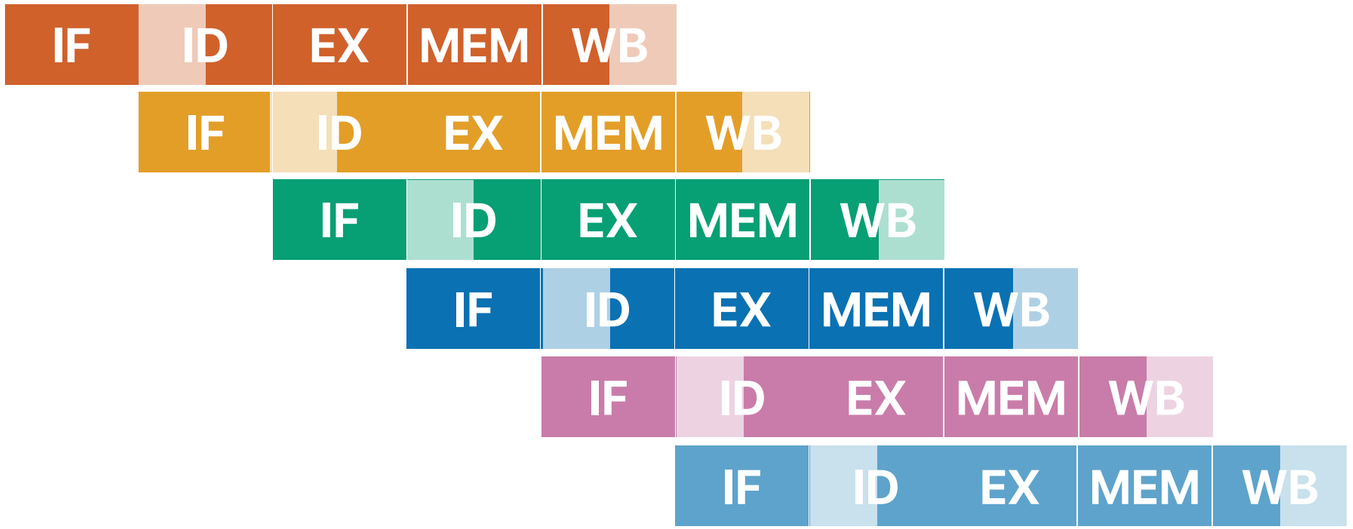
- Implication 1 — You can always have the same amount of circuits in a chip, regardless how Moore's Law reduces the die size
  - We make "specialized processors" (i.e., ASICs, accelerators) for target applications — computers become heterogeneous
- Implication 2 — You can put more circuits in, but you cannot activate all of them all the same time — Dark Silicon Problem
  - We equip more functions into each processor — processors become heterogeneous

Why accelerators can be more efficient

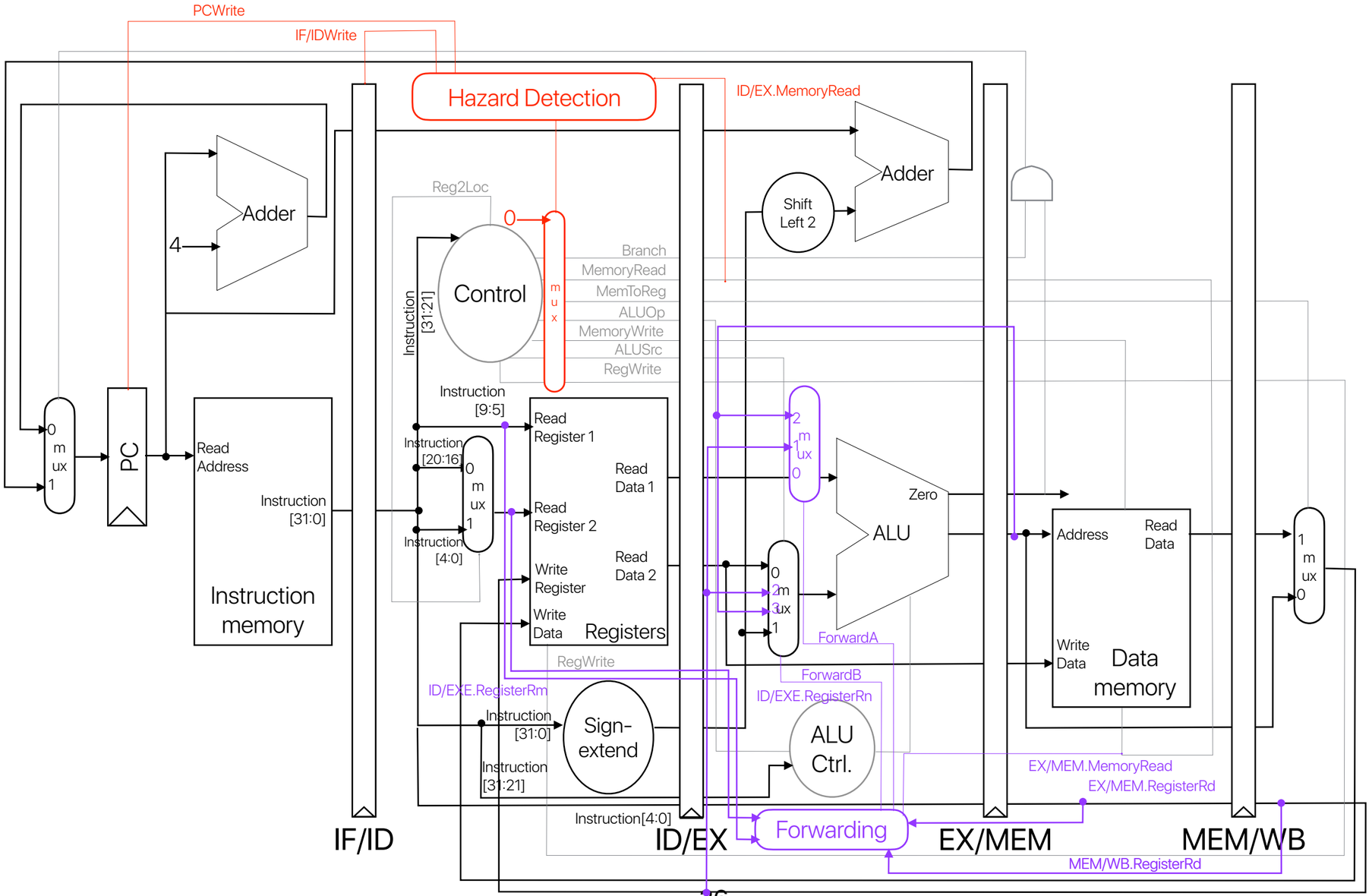
Say, we want to implement a[i] += b[i]*c[i]

- This is what we need in RISC-V in each iteration

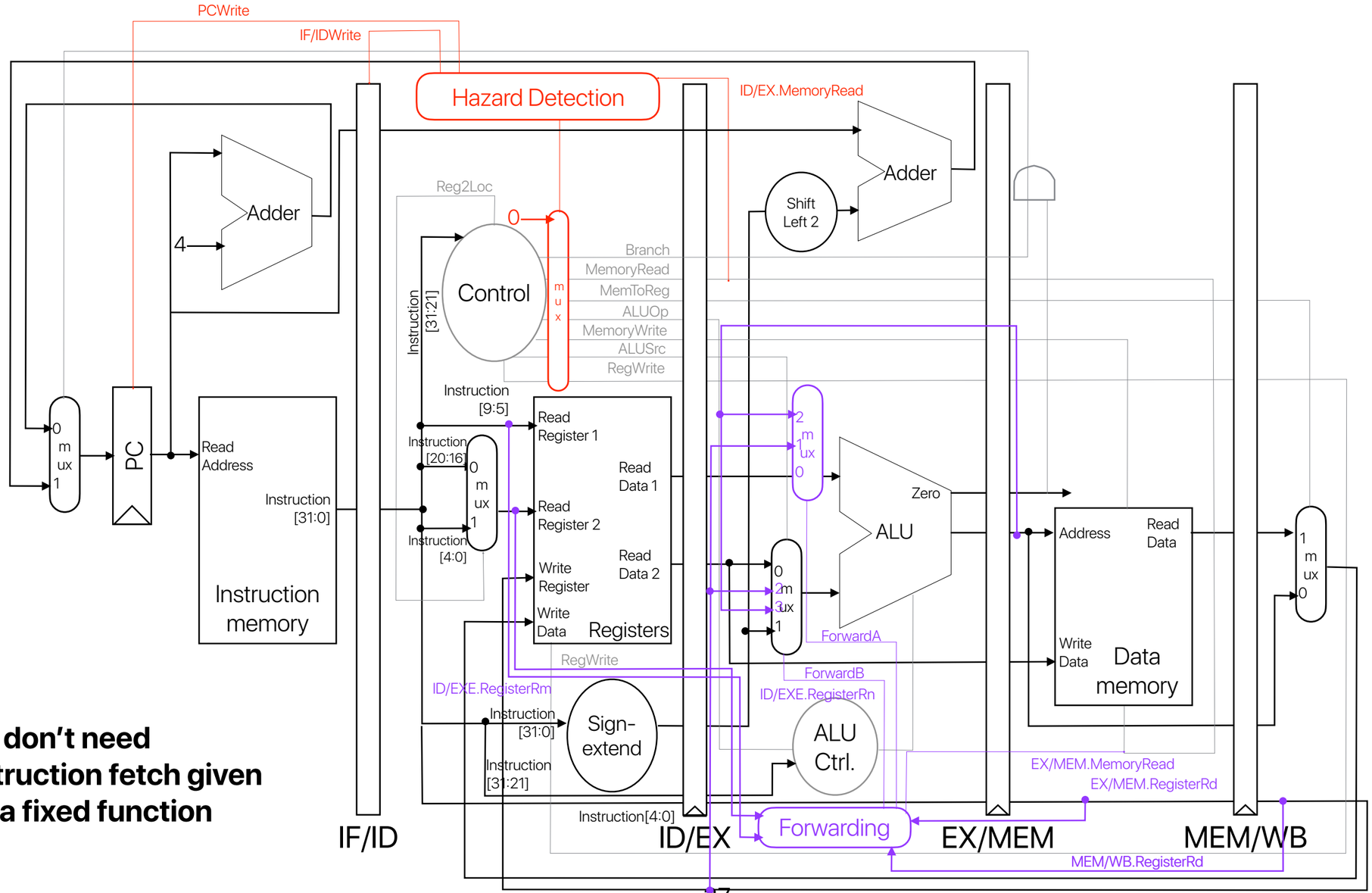
```
ld    X4, 0(X0)
ld    X5, 0(X1)
ld    X6, 0(X2)
mul   X7, X4, X5
add   X6, X6, X7
sd    X6, 0(X2)
```



General-purpose processor pipeline



Specialize the circuit

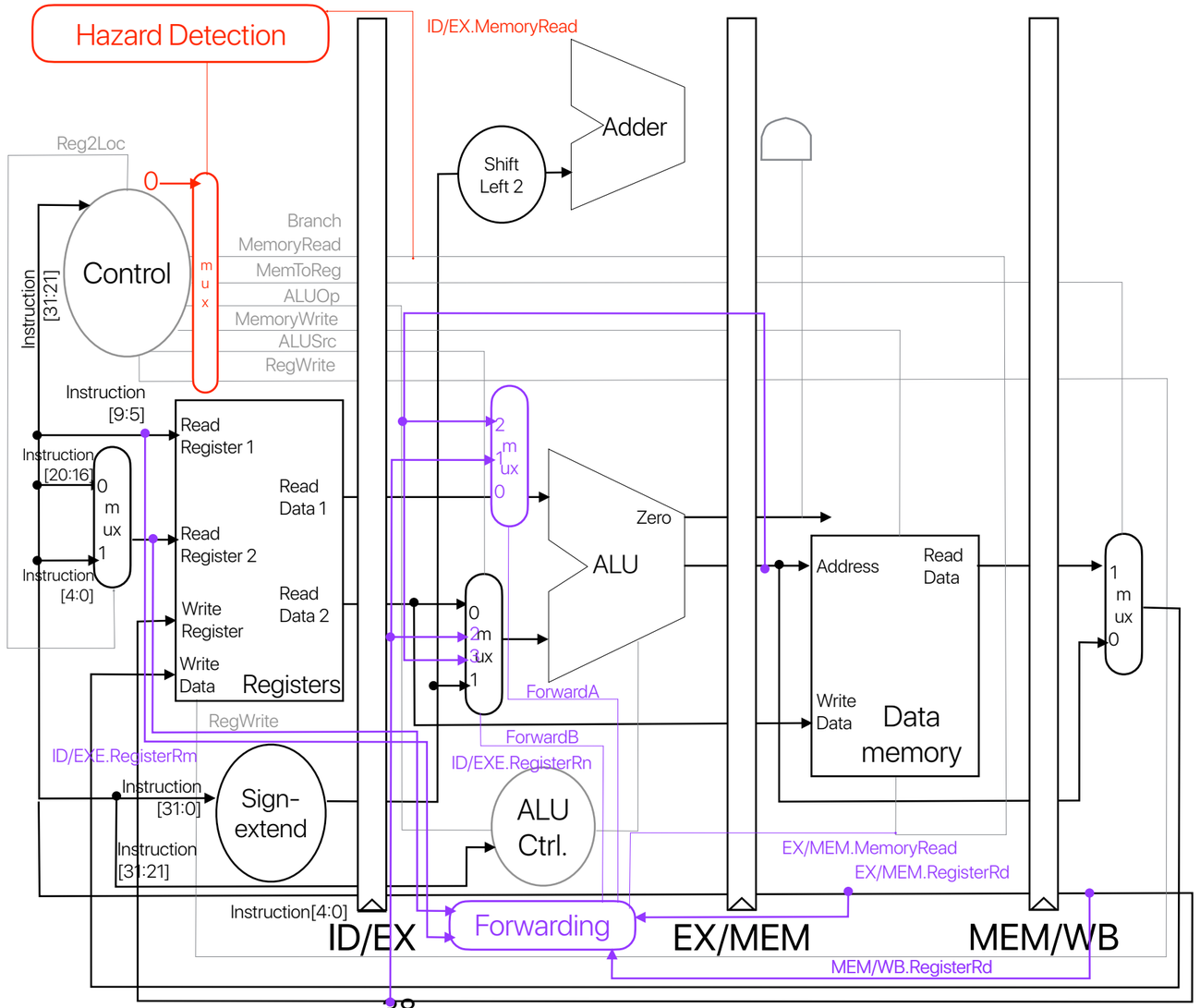


We don't need instruction fetch given it's a fixed function

Specialize the circuit

We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function

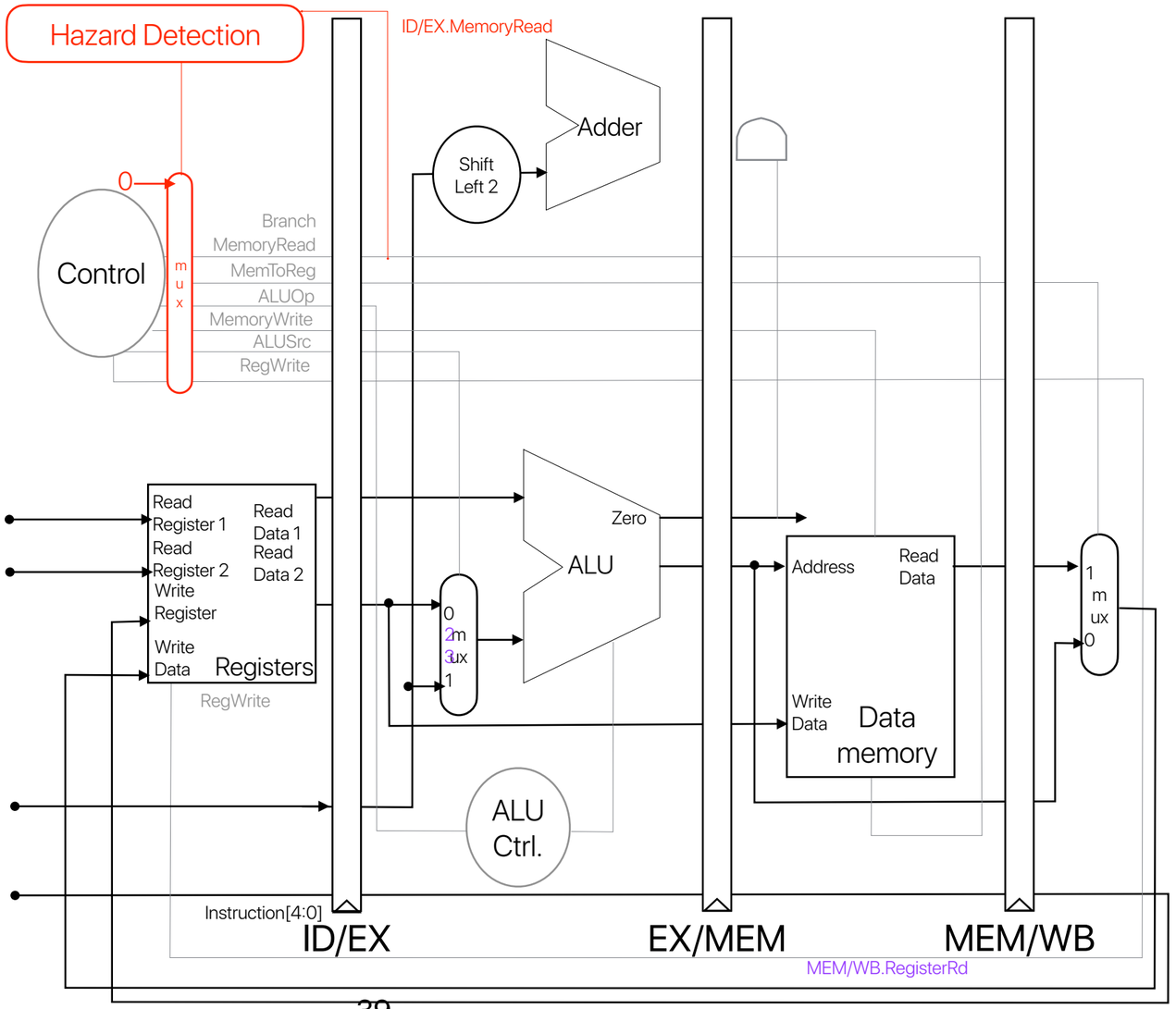


Specialize the circuit

We don't need ALUs, branches, hazard detections…

We don't need these many registers, complex control, decode

We don't need instruction fetch given it's a fixed function

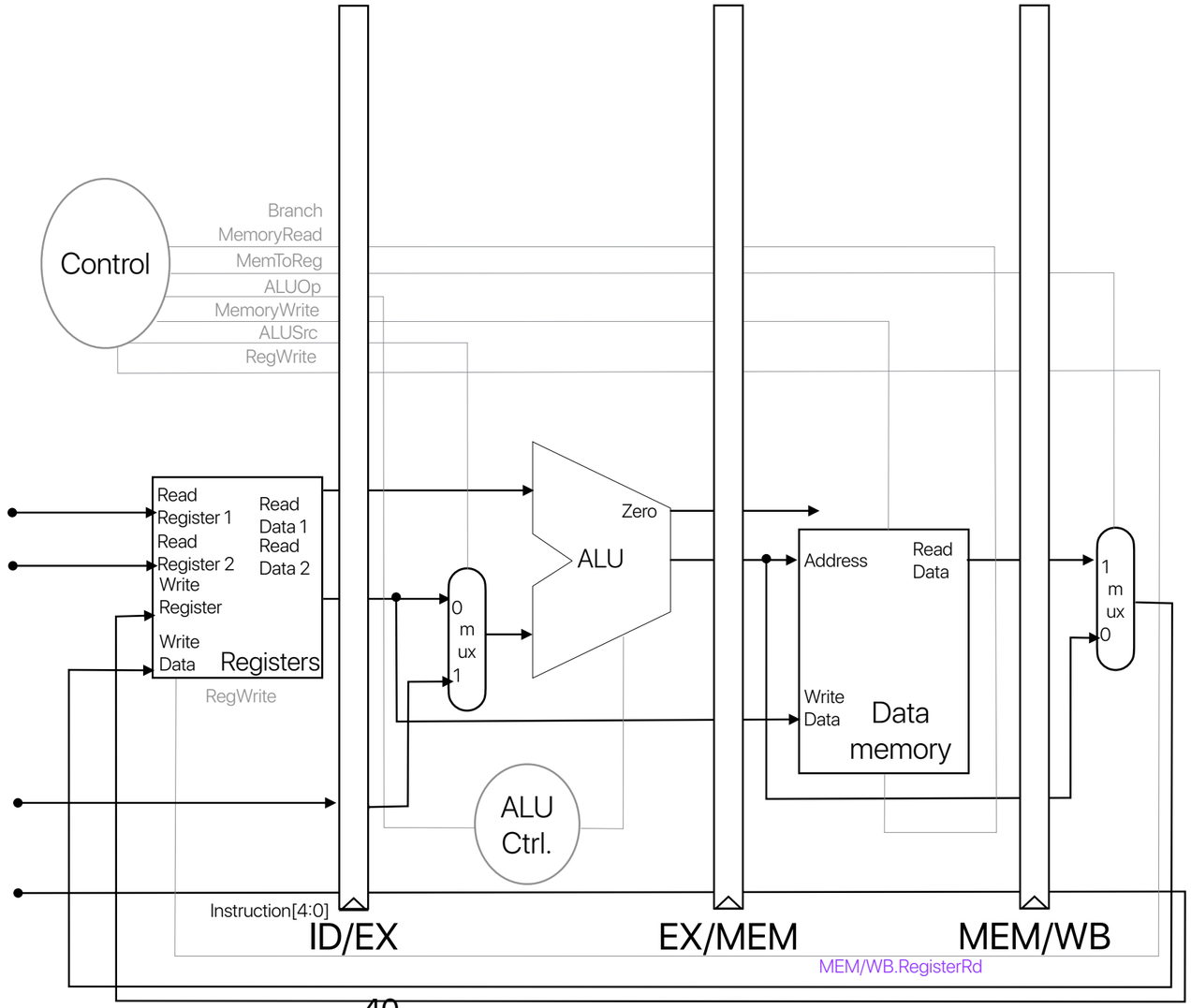


Specialize the circuit

We don’t need big ALUs, branches, hazard detections…

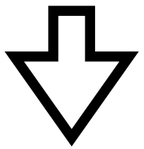
We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function

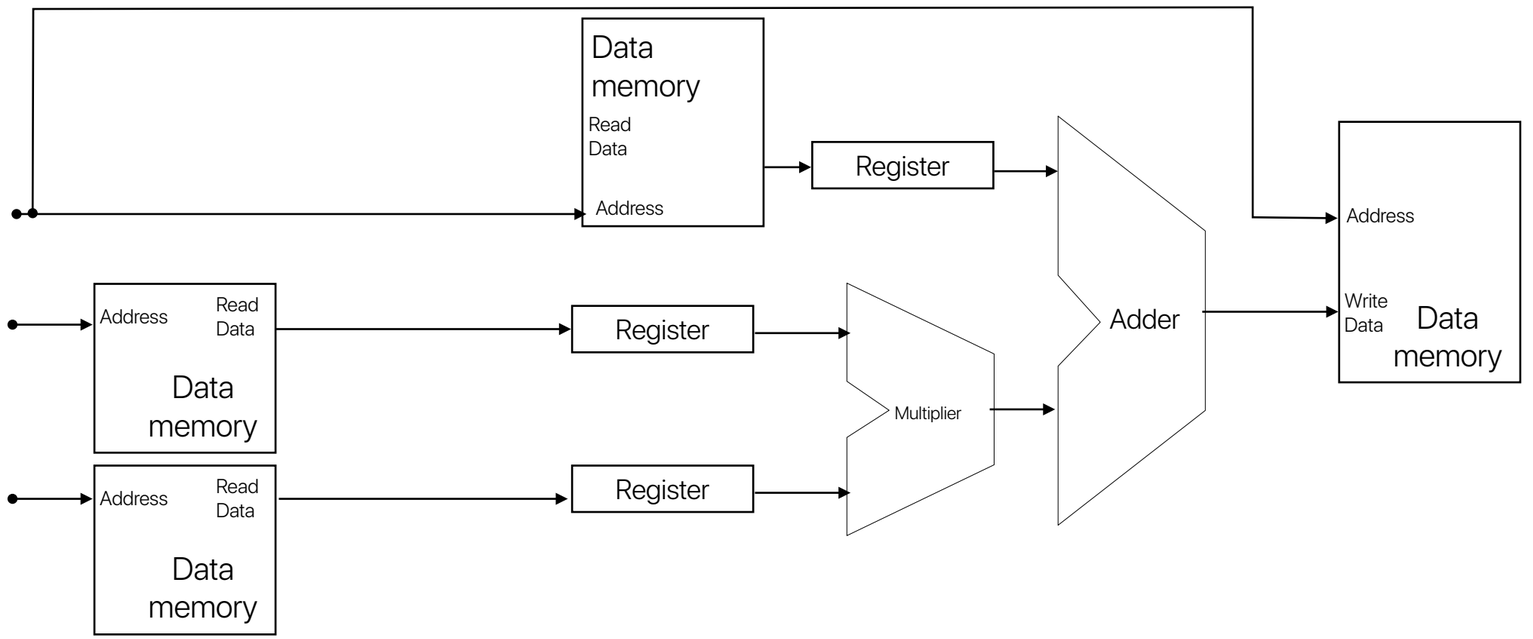


Rearranging the datapath

```
ld    X4, 0(X0)
ld    X5, 0(X1)
ld    X6, 0(X2)
mul   X7, X4, X5
add   X6, X6, X7
sd    X6, 0(X2)
```



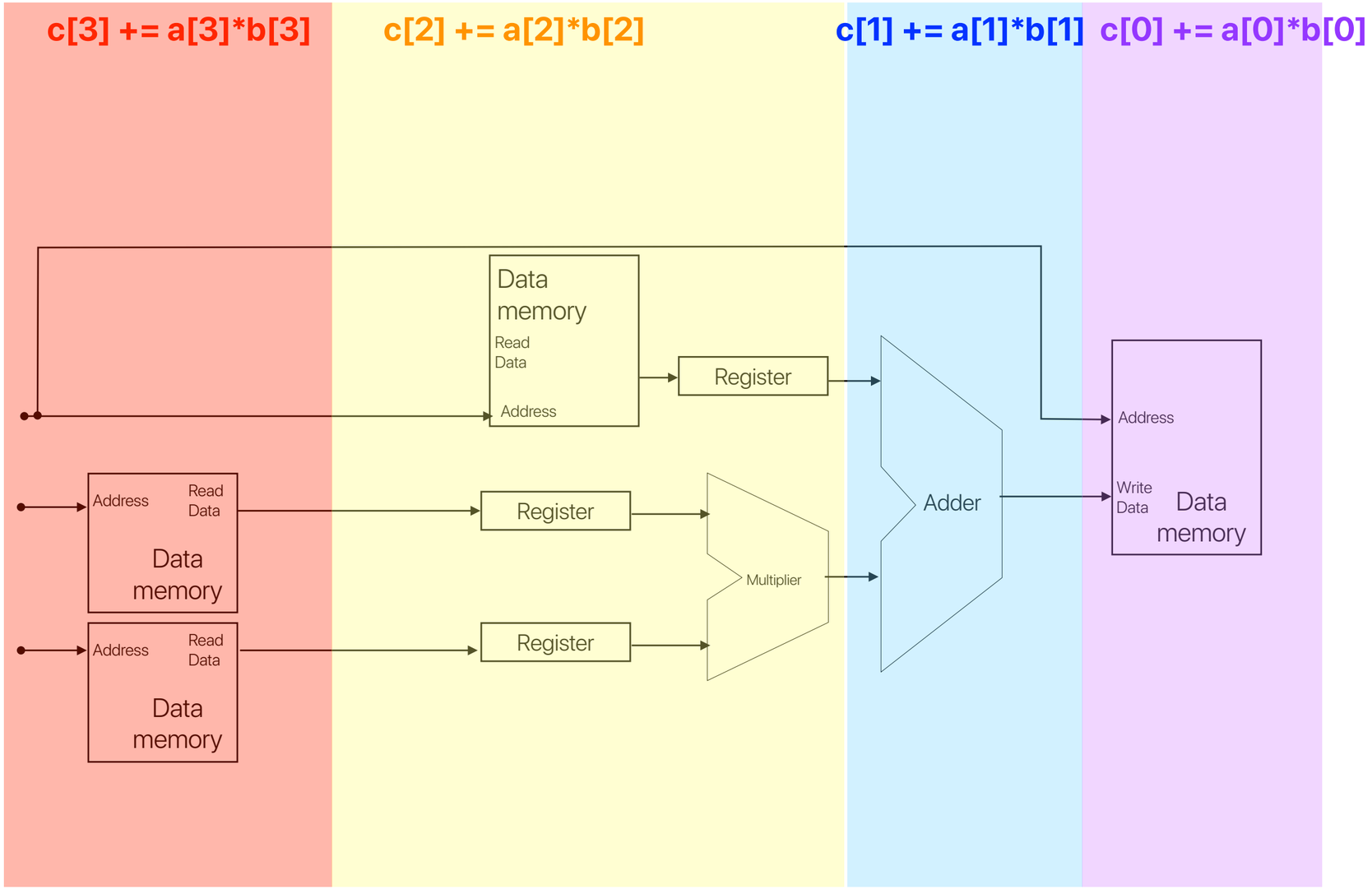
```
 mac X2,X0,X1
```



Pipeline it again!

Each stage can still be as fast as (or faster if it's simpler) the pipelined processor
— clock rate is the same or higher

But each stage is now working on what the original 6 instructions would do
— instruction count is lower



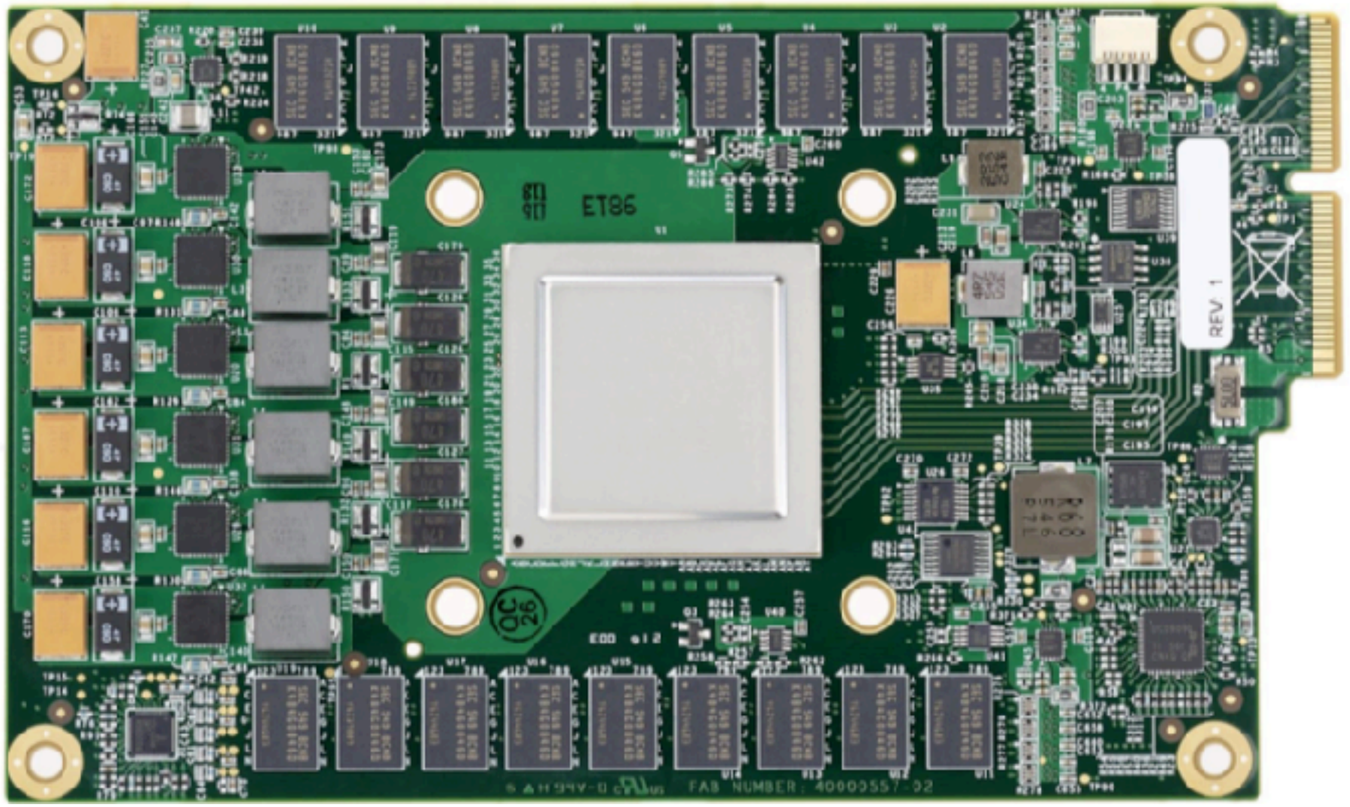
How can the idea of accelerator help?

- Performance-wise $ET = IC \times CPI \times CT$
  - The accelerator itself reduces the amount of “instructions” used to implement a program
  - The accelerator’s logic is simpler than that of a general-purpose pipeline (potentially higher clock rate)
- Power-wise $P = \alpha C V^2 f$
  - Simpler logic reduces the waste of gates
- Energy-wise $Energy = P \times ET$
  - Energy is power times execution time
  - If you improve both, it’s better anyway

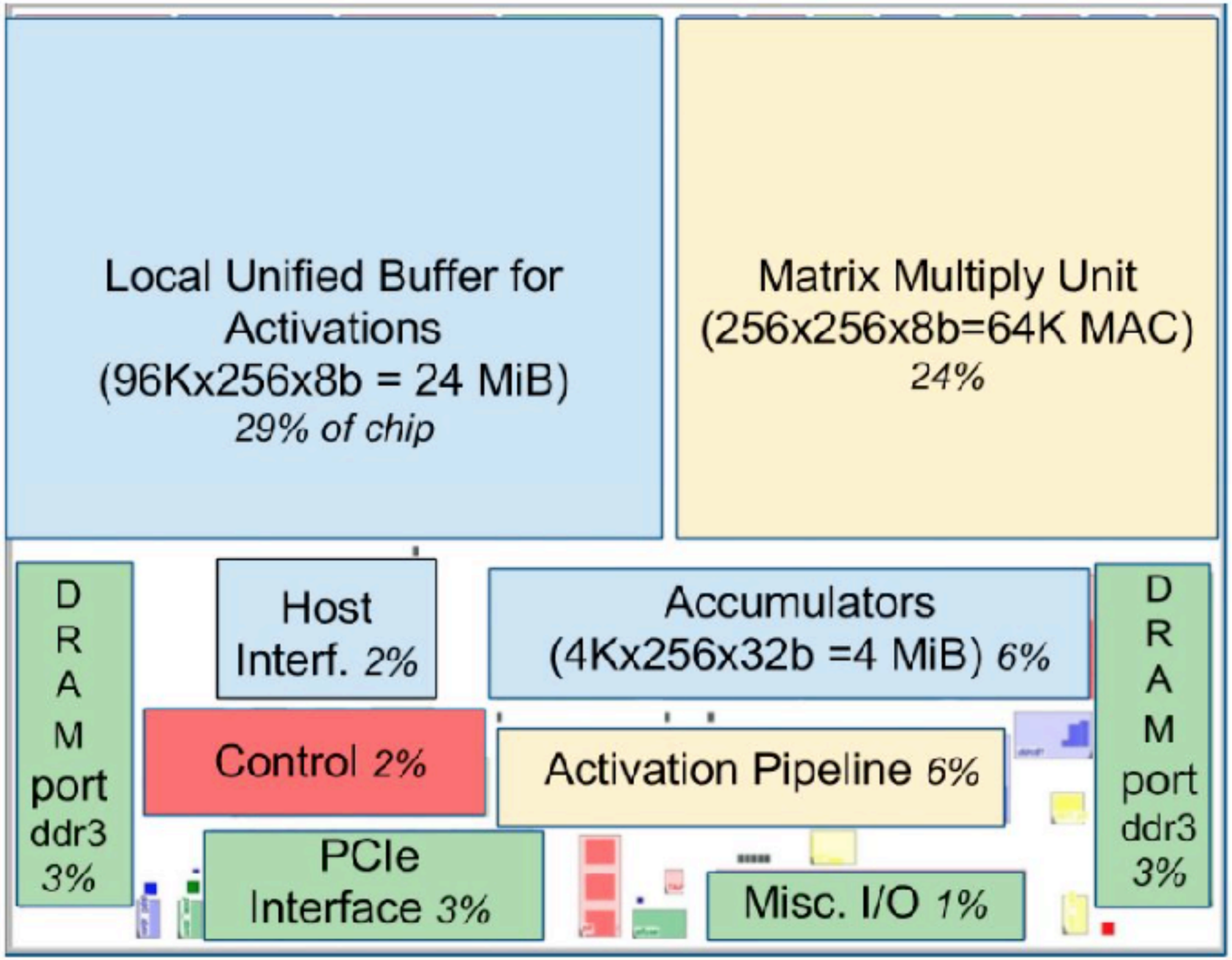
Example of a hardware accelerator — Tensor Processing Unit

- N. Jouppi, C. Young, N. Patil and D. Patterson. Motivation for and Evaluation of the First Tensor Processing Unit. In IEEE Micro, vol. 38, no. 3, pp. 10-19. 2018
- Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In ISCA '17. 2017.

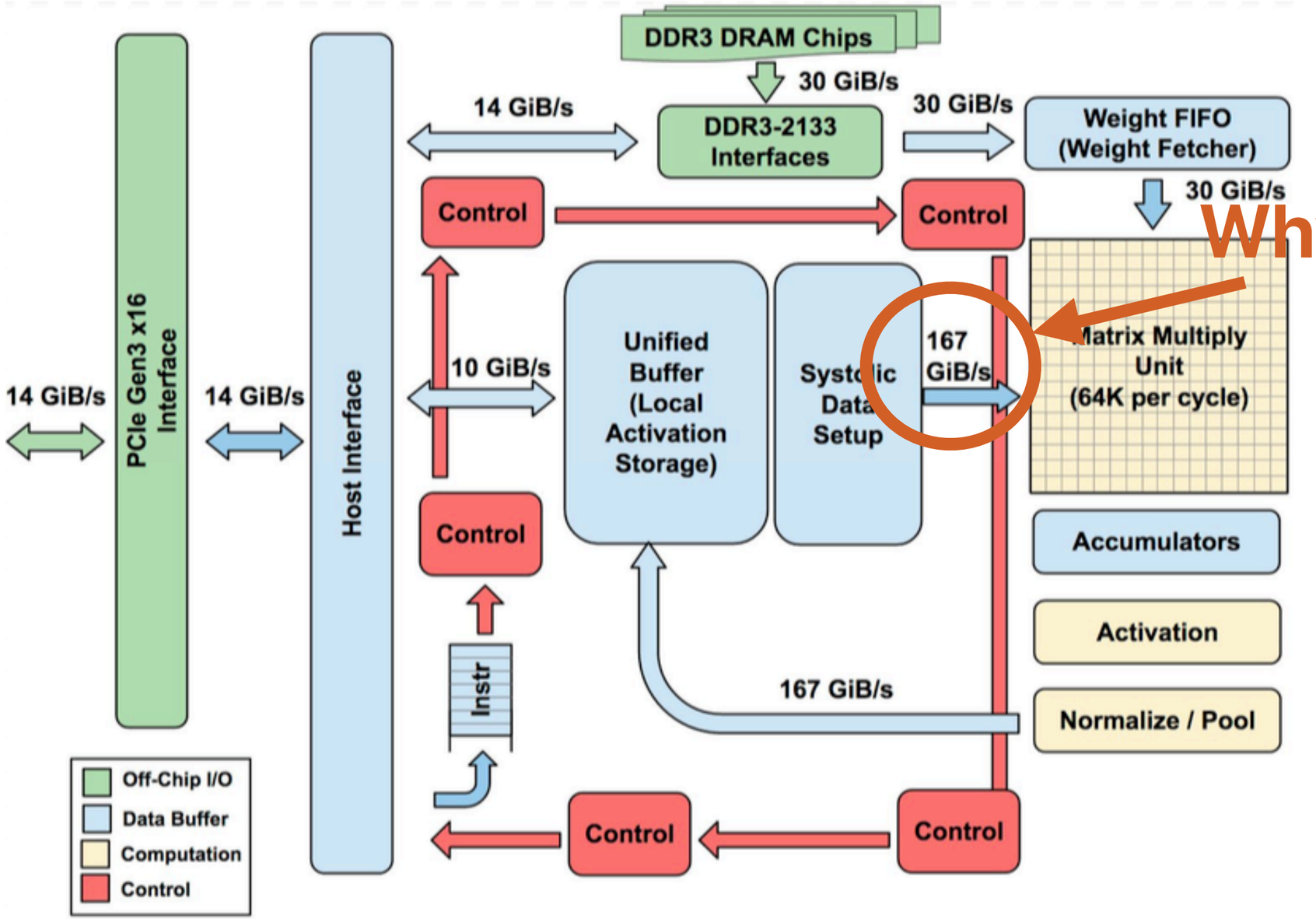
What TPU looks like



TPU Floorplan



TPU Block diagram



Electrical
Computer Science
Engineering

277

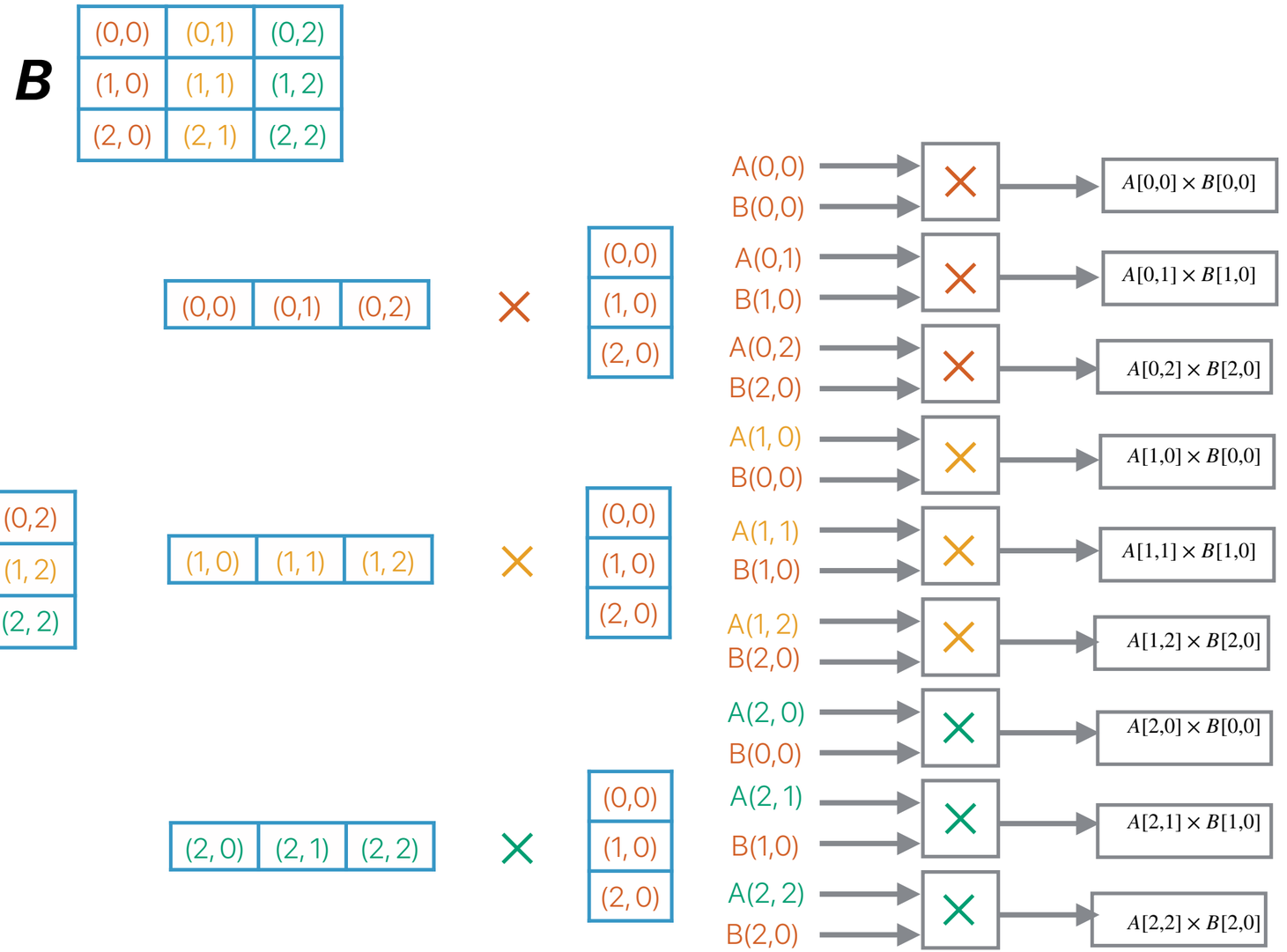
つづく

CUDA Matrix Multiplications

```
template<typename T>
__global__
void naive_matrix_multiply(const T *A, const T *B, T* C, int width, int P, int Q)
{
  int r = blockIdx.y * blockDim.y + threadIdx.y;
  int c = blockIdx.x * blockDim.x + threadIdx.x;
  // check boundry conditions
  if( r < P && c < Q){
    // do the multiplication for one row and col
    T value = 0;
    for(int k = 0; k < width; k++){
      value += A[r * width + k] * B[k * Q + c];
    }
    // store the result
    C[r * Q + c] = value;
  }
}
```

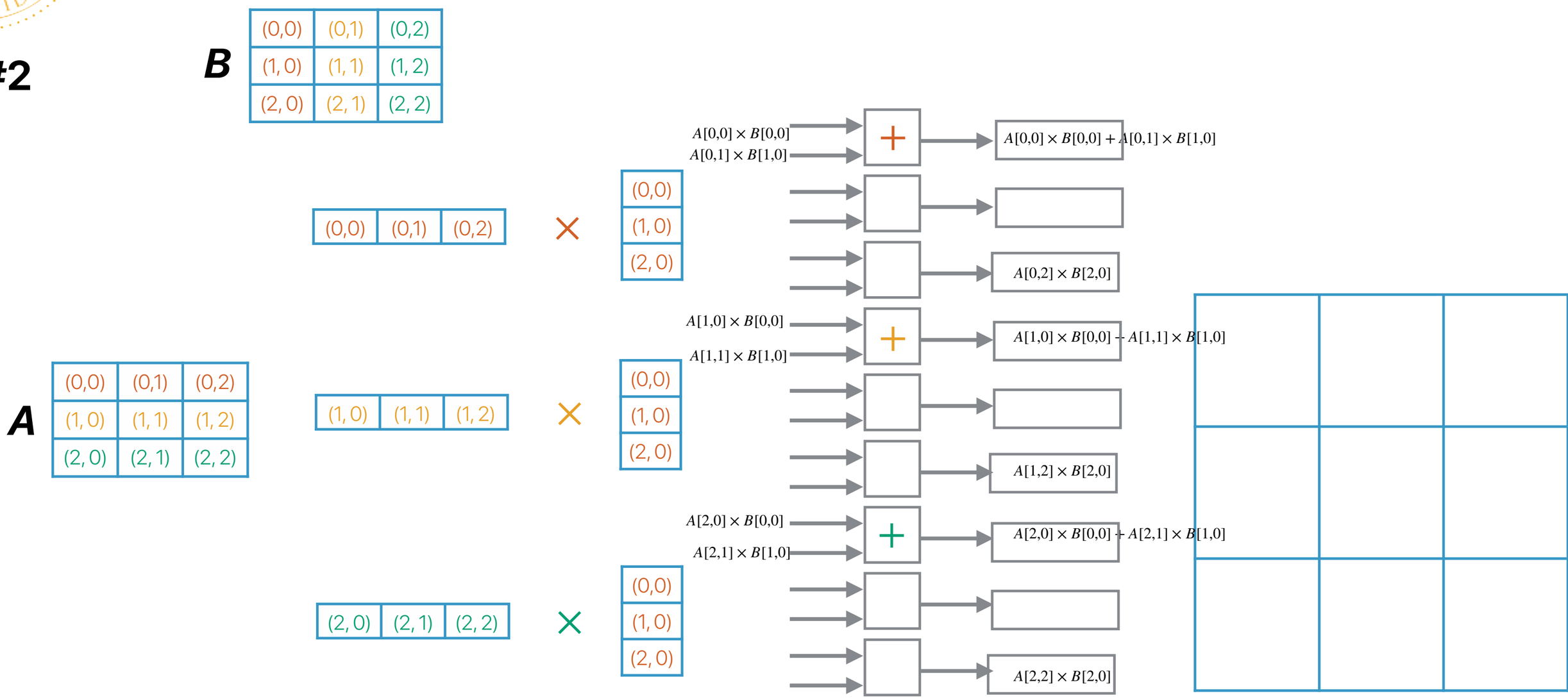
Vector processing for MM

#1



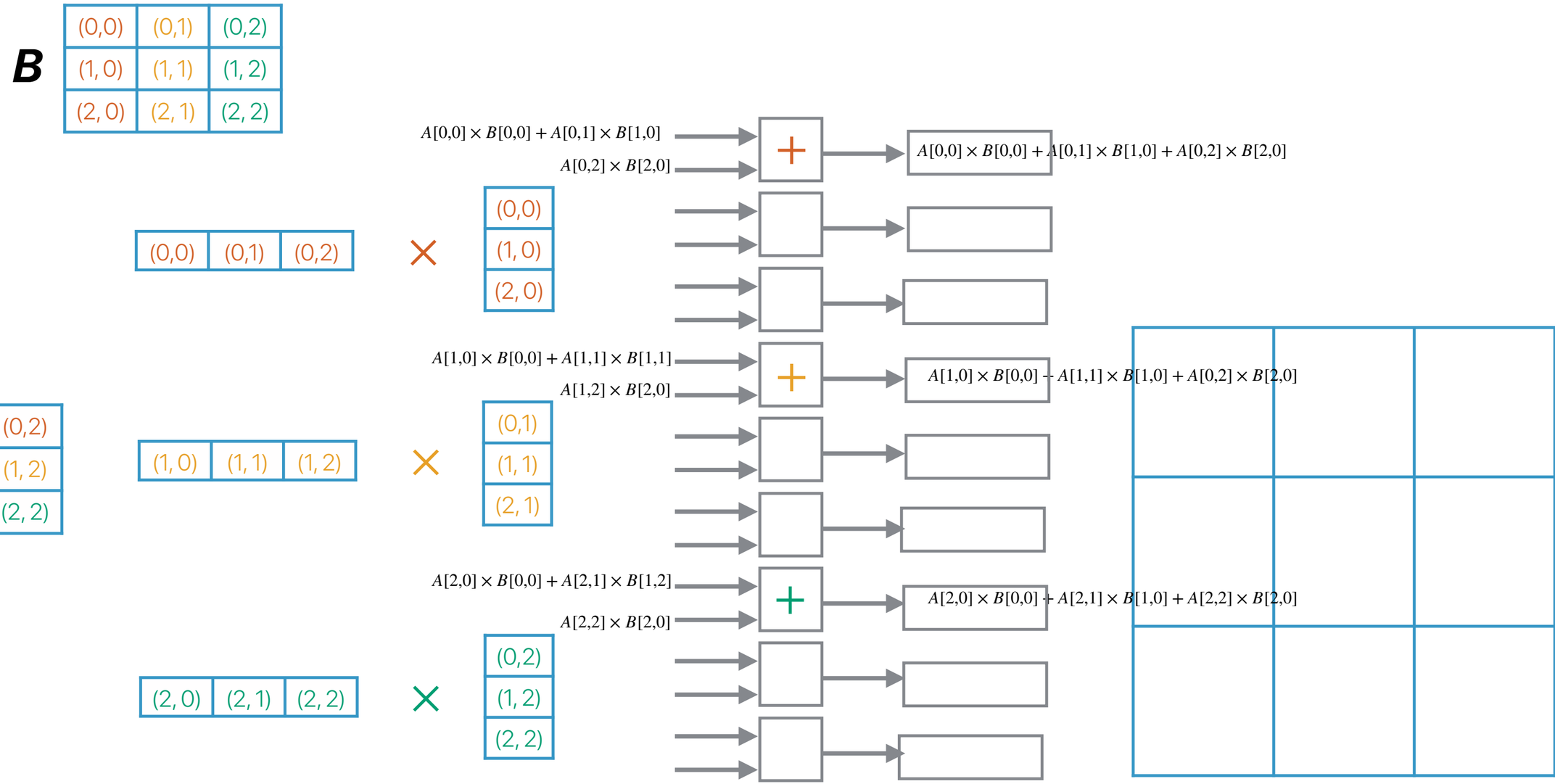
Vector processing for MM

#2



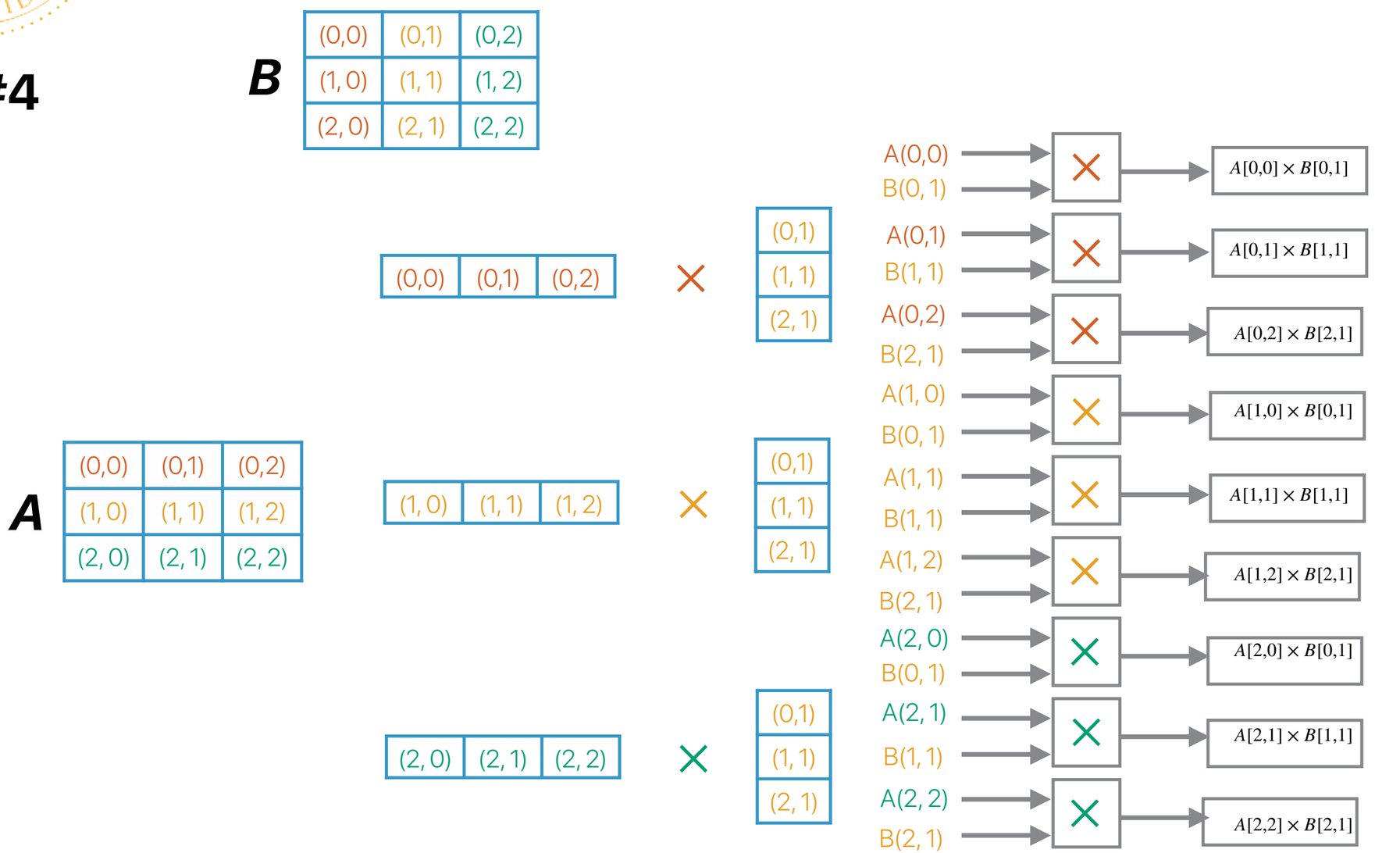
Vector processing for MM

#3



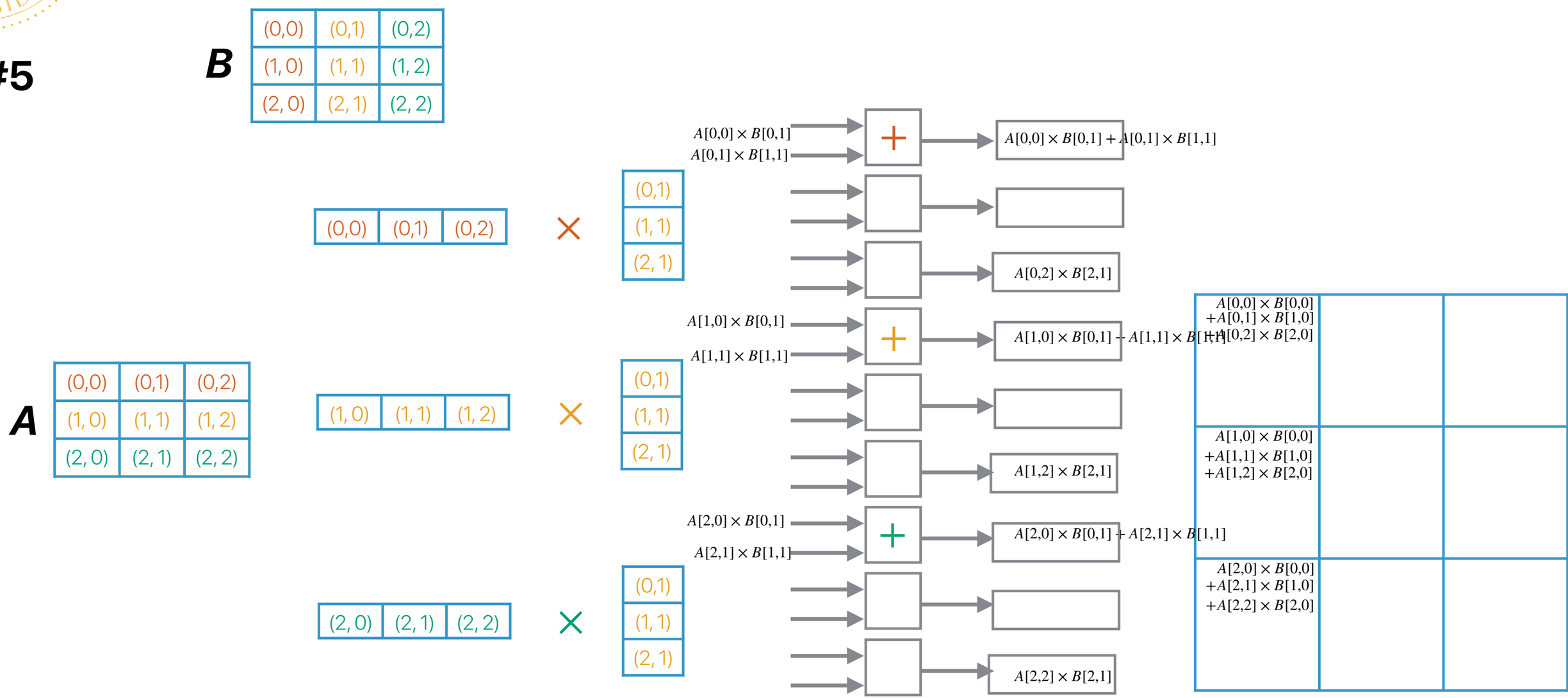
Vector processing for MM

#4



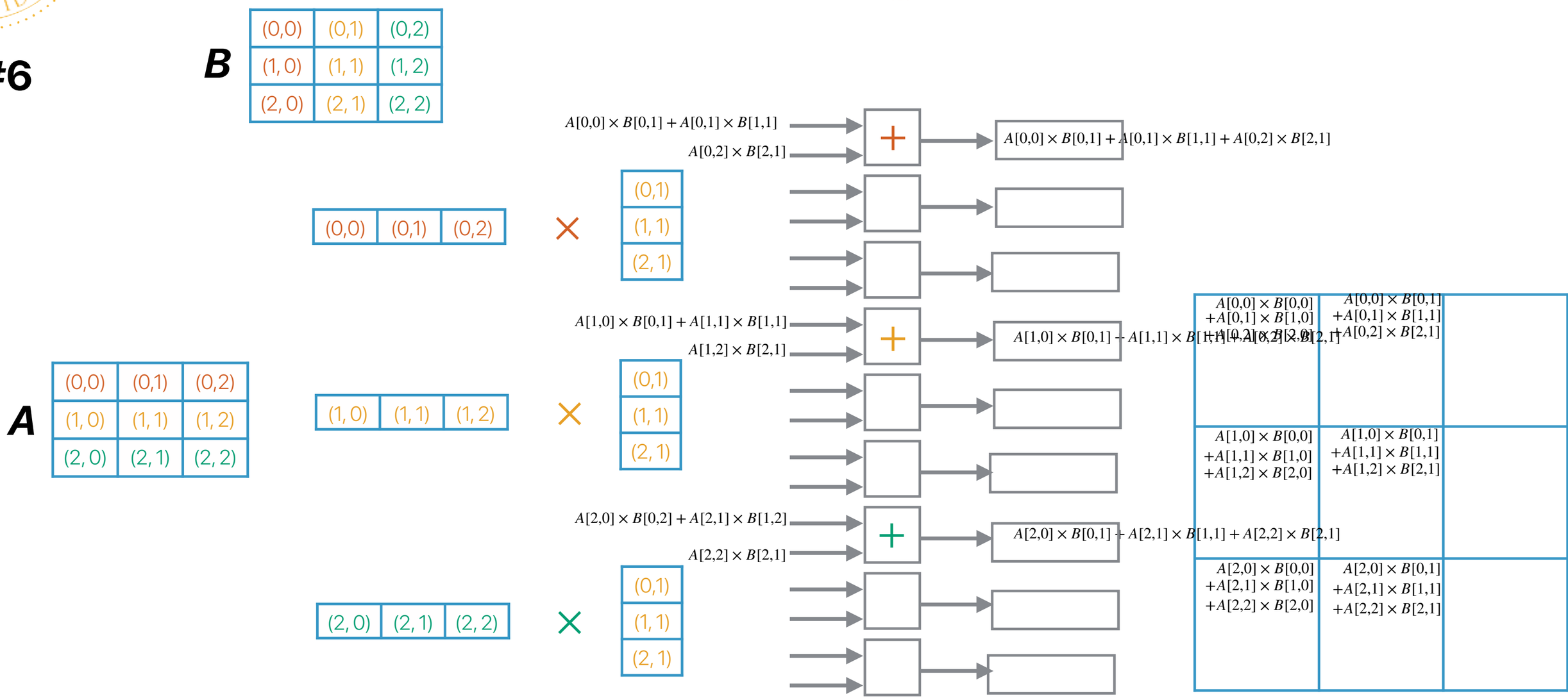
Vector processing for MM

#5



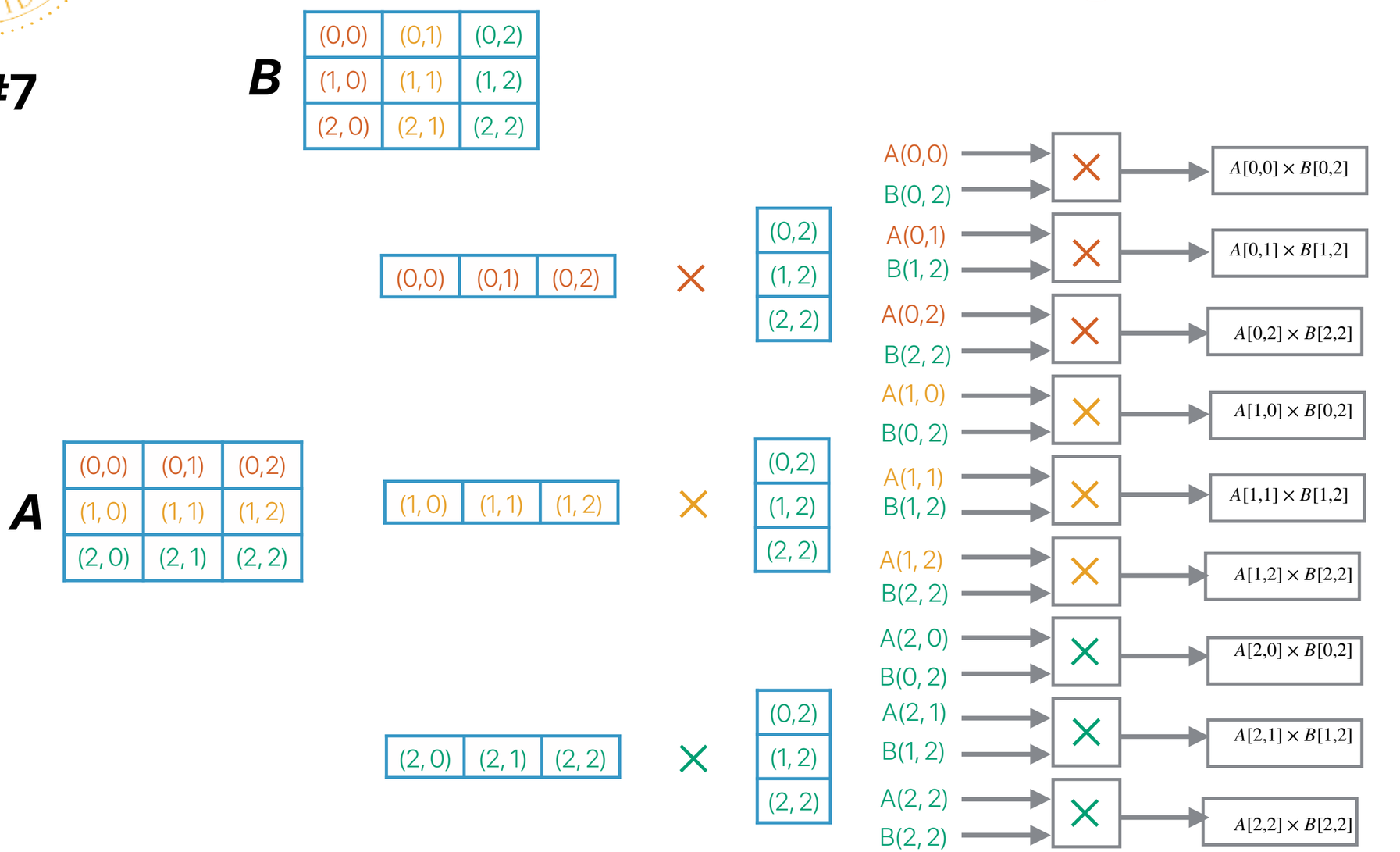
Vector processing for MM

#6



Vector processing for MM

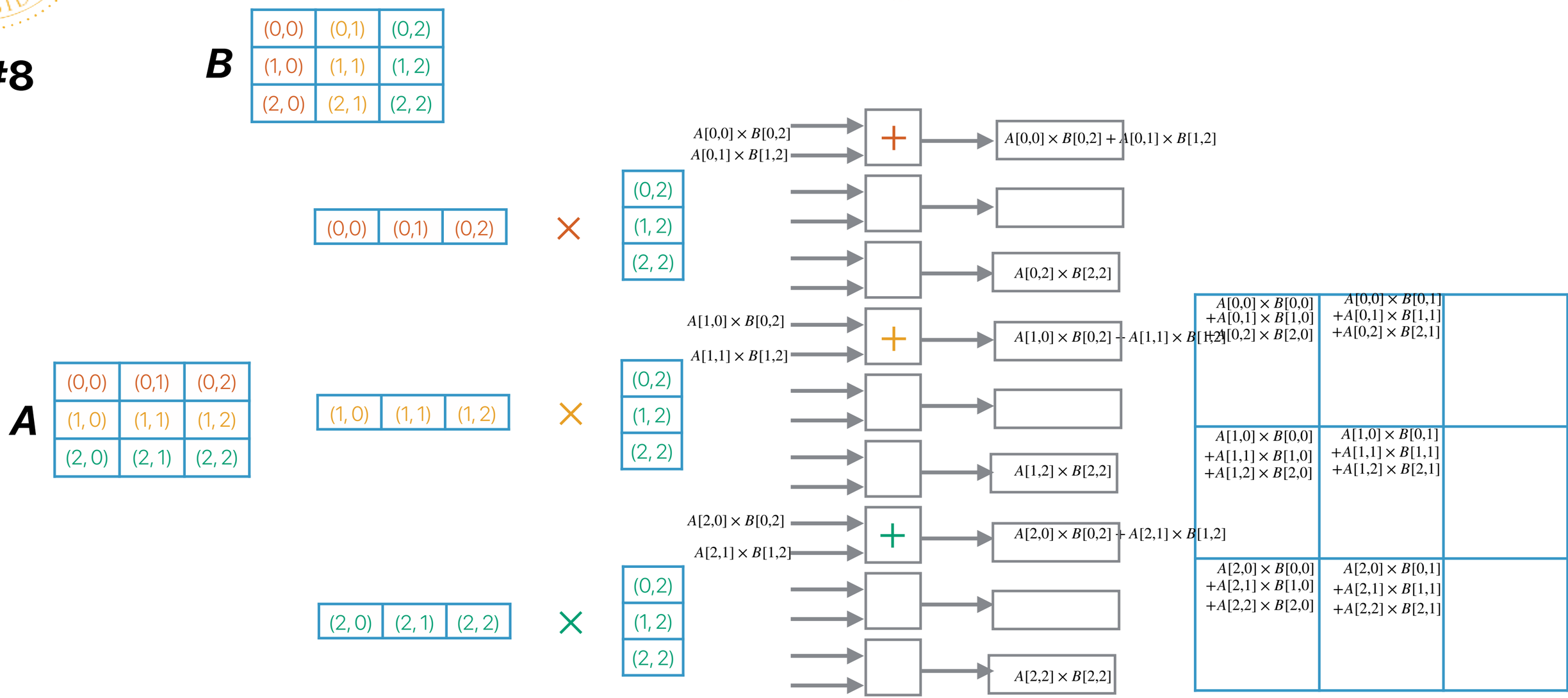
#7



$A[0,0] \times B[0,0] + A[0,1] \times B[1,0] + A[0,2] \times B[2,0]$	$A[0,0] \times B[0,1] + A[0,1] \times B[1,1] + A[0,2] \times B[2,1]$	
$A[1,0] \times B[0,0] + A[1,1] \times B[1,0] + A[1,2] \times B[2,0]$	$A[1,0] \times B[0,1] + A[1,1] \times B[1,1] + A[1,2] \times B[2,1]$	
$A[2,0] \times B[0,0] + A[2,1] \times B[1,0] + A[2,2] \times B[2,0]$	$A[2,0] \times B[0,1] + A[2,1] \times B[1,1] + A[2,2] \times B[2,1]$	

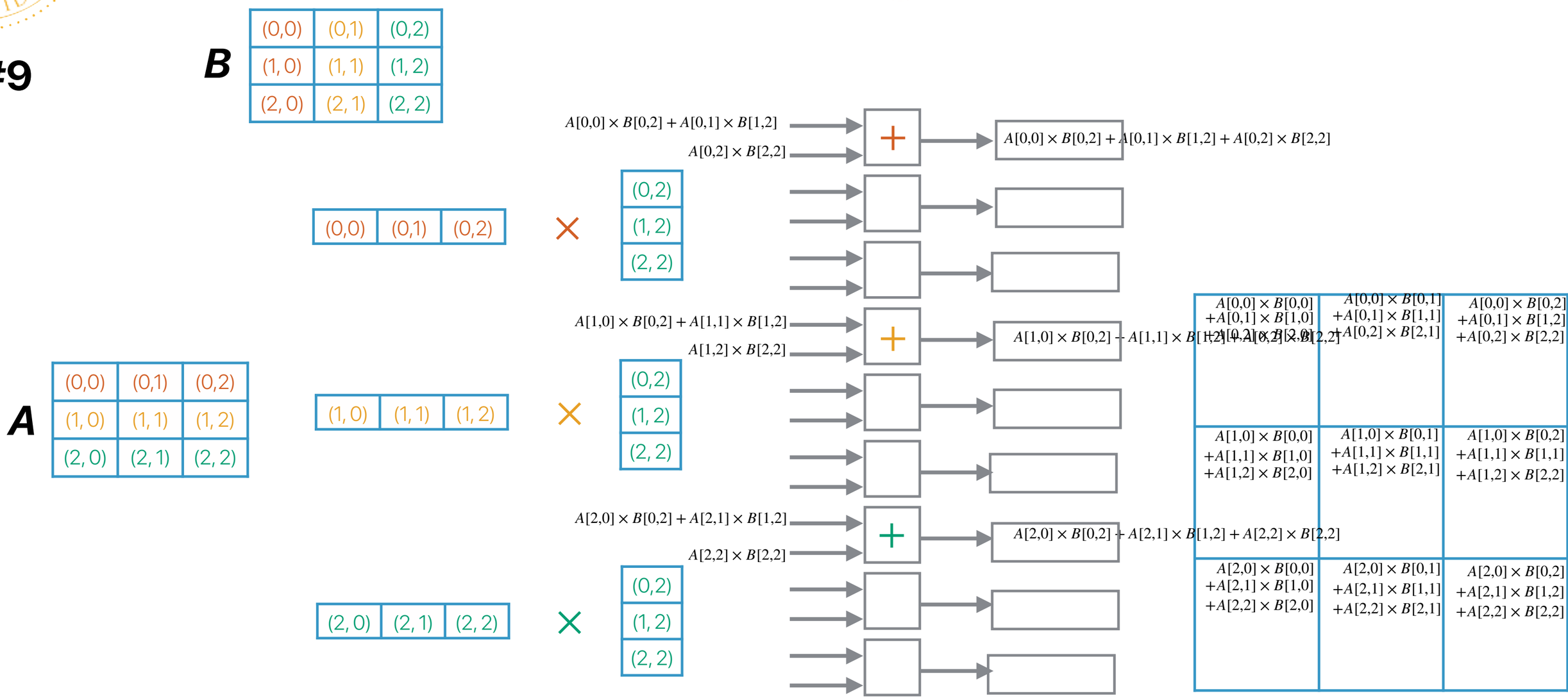
Vector processing for MM

#8



Vector processing for MM

#9



For n² matrices with n² processing elements: $n \times (1 + \lceil log_2(n) \rceil)$

CPU Performance Equation

$$Performance = \frac{1}{Execution\ Time}$$

$$Execution\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

$$ET = IC \times CPI \times CT$$

$$1GHz = 10^9 Hz = \frac{1}{10^9} sec\ per\ cycle = 1\ ns\ per\ cycle$$

$$\frac{1}{Frequency(i.e., clock\ rate)}$$