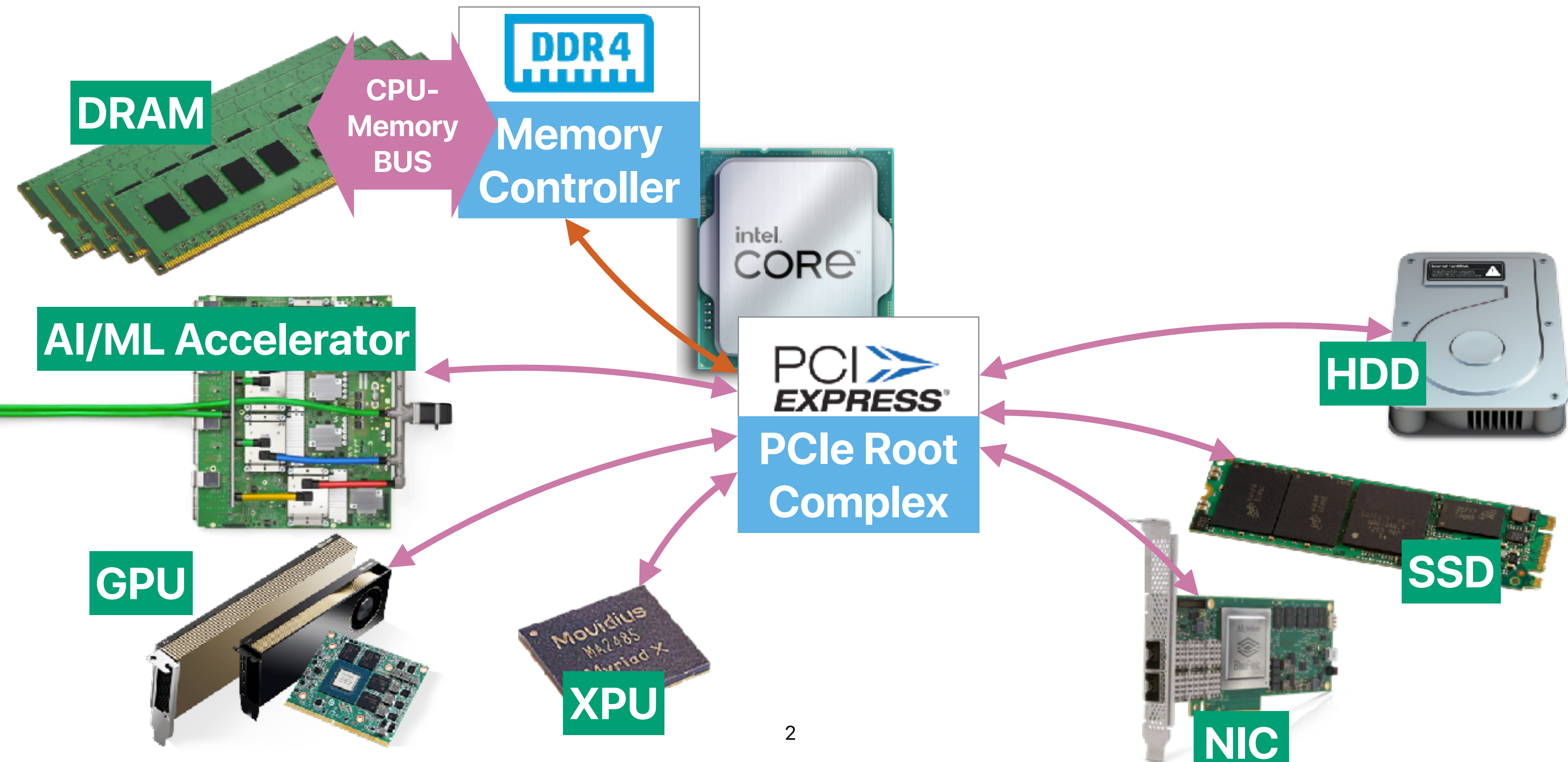


The roofline model and its implications

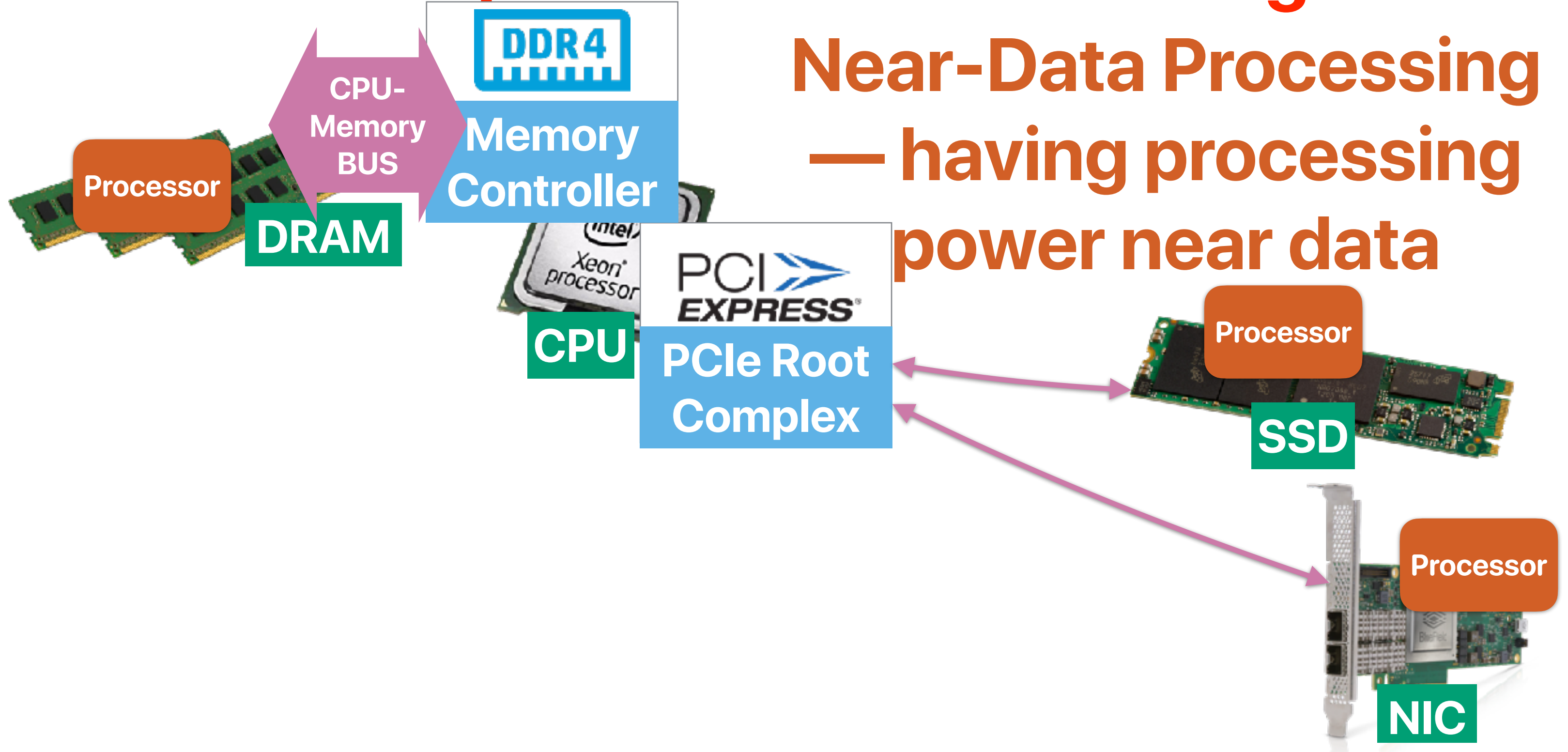
Hung-Wei Tseng

Recap: The "real" datapath in CPU-centric computing

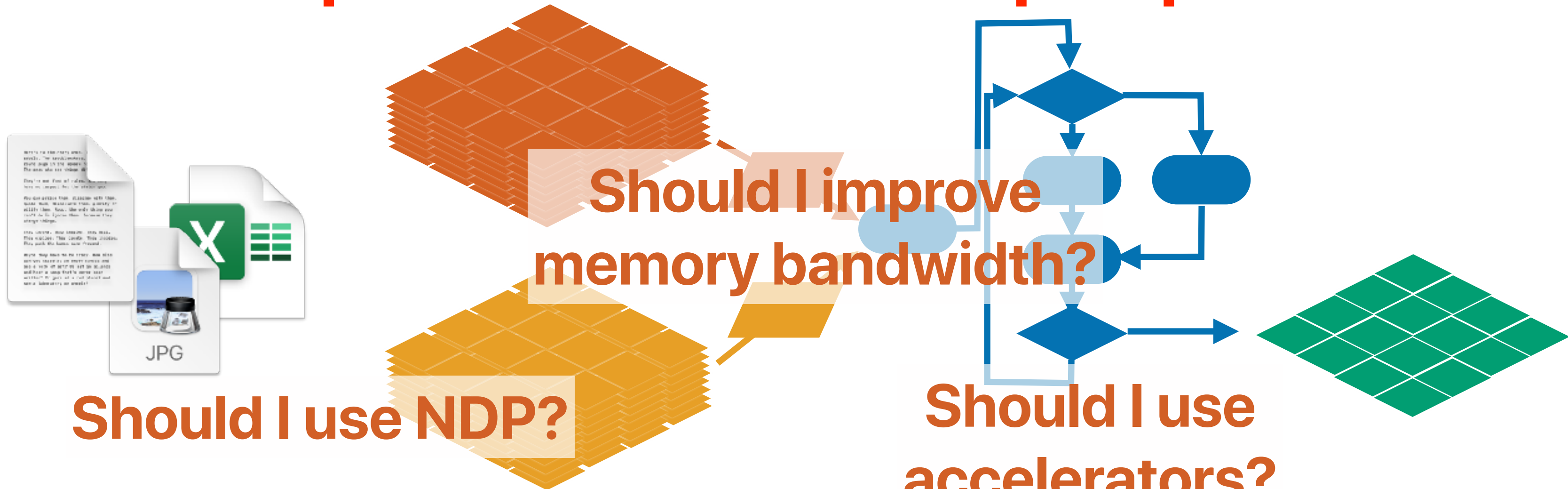


Recap: Near-Data Processing

Near-Data Processing
— having processing
power near data



Recap: From the software perspective



Source "data"

"Data" structures

Algorithms

Result

**How to decide if we should go
Compute-centric? Data-centric?**

Outline

- The roofline model
- Why heterogeneous computer architectures
 - GPUs
 - TPUs

3:00

**What's the maximum daily revenue of a
boba tea shop?**

Ideas?

- How many customers we can attract each day
- How many cups we can make each day
- How much each cup costs

Ideas?

bytes of

- How many **data** we can **supply each cycle**
- How many **OPs** we can **perform each cycle**
- How **many OPs each byte of data need**

Operational Intensity

- Number of operations each byte of data would need for an algorithm
- Changes with algorithm

Example — 32-bit 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 =

$$2 \times M \times N \times P \text{ ops}$$

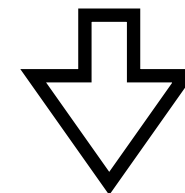
- Total size of data accesses =

$$4 \times (2 \times N + 1) \times M \times P \text{ bytes}$$

- Average operations per byte =

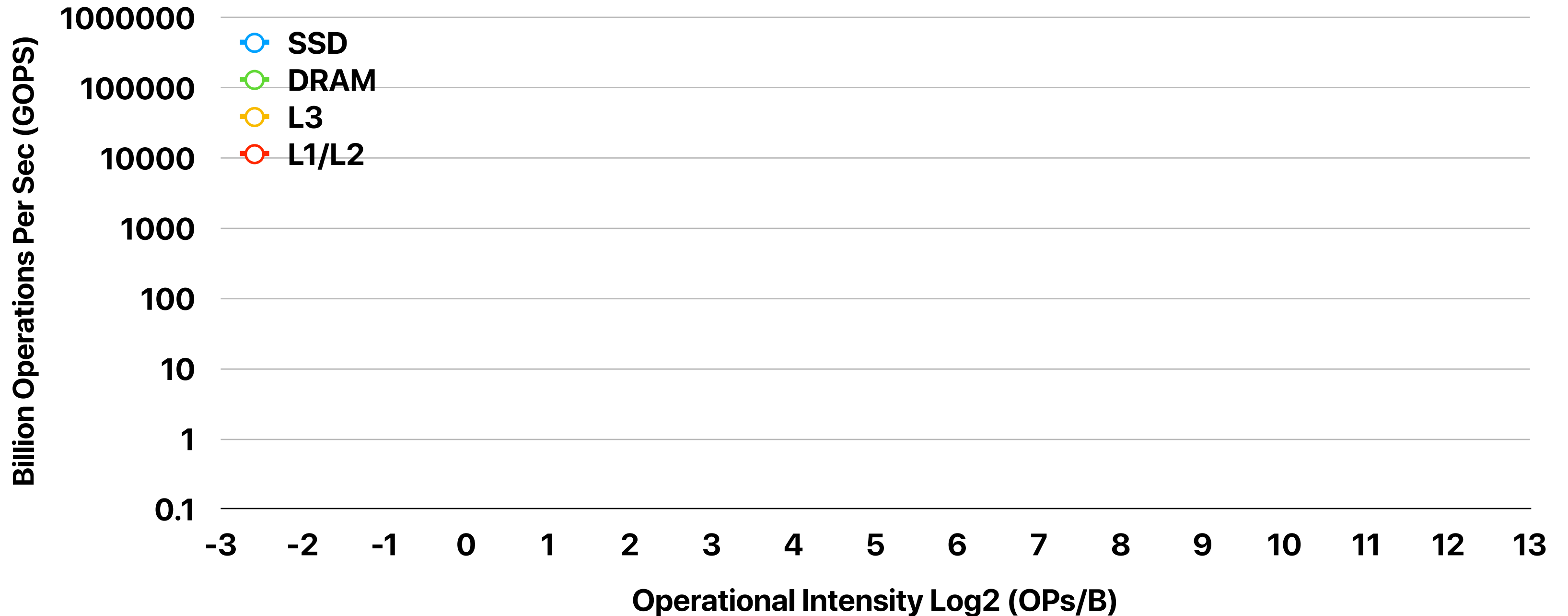
$$\frac{2 \times M \times N \times P}{4 \times (2 \times N + 1) \times M \times P} = \frac{1}{4} = 0.25$$

```
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];  
        }  
    }  
}
```

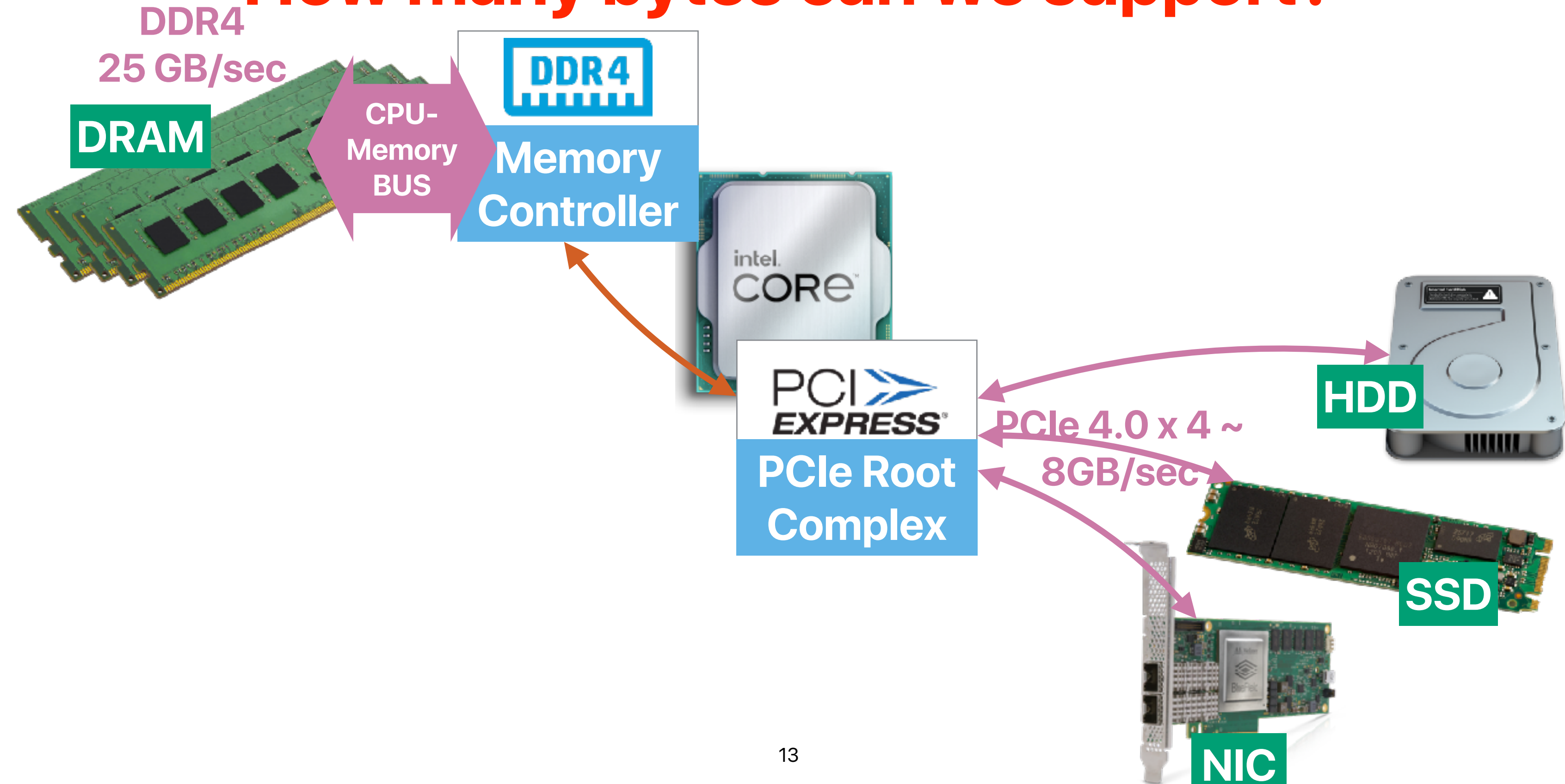


```
for(i = 0; i < M; i++) {  
    for(j = 0; j < P; j++) {  
        sum = 0.0;  
        for(k = 0; k < N; k++) {  
            sum += a[i][k]*b[k][j];  
        }  
        c[i][j] = sum;  
    }  
}
```

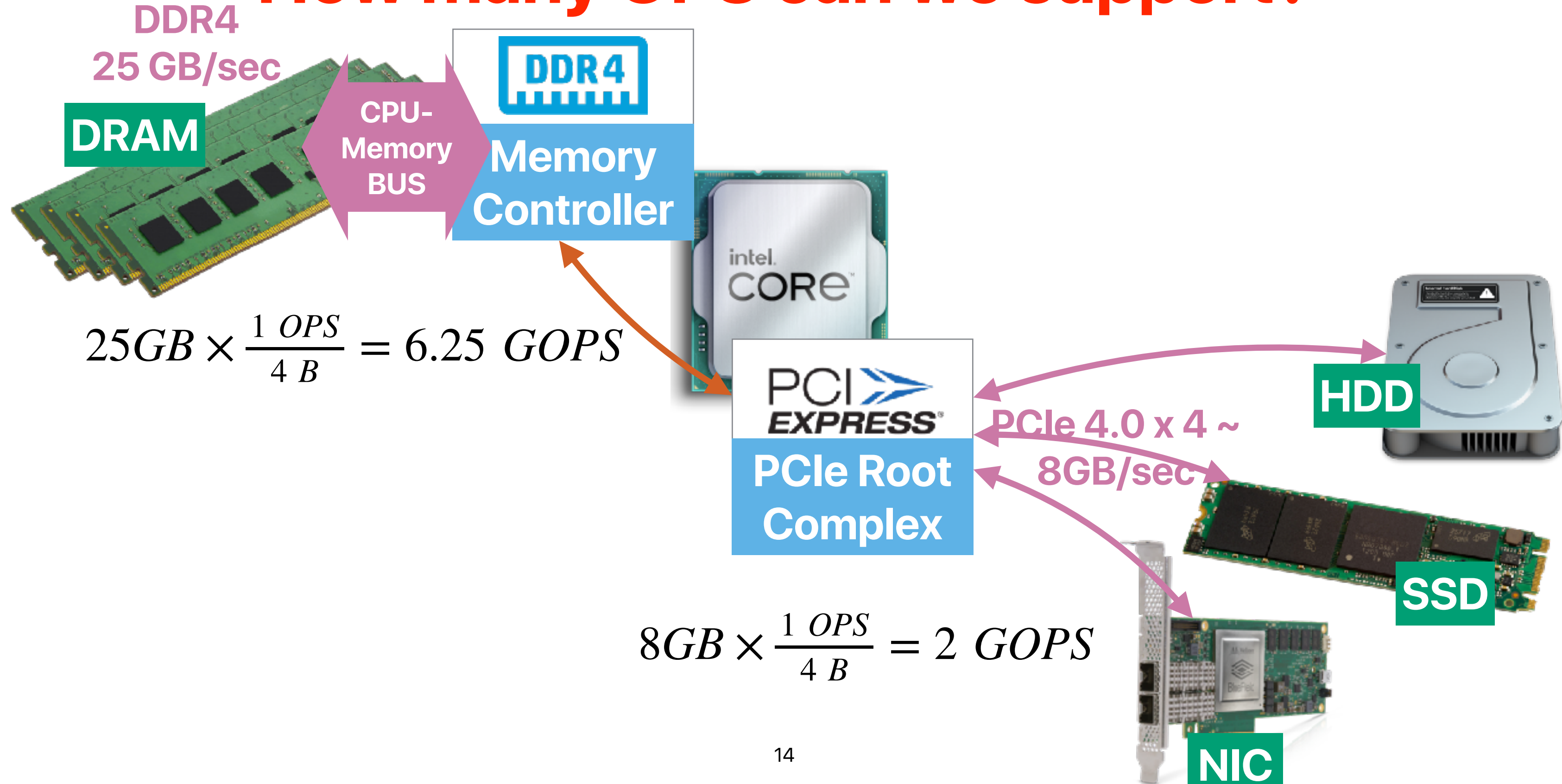
How fast can we supply data?



How many bytes can we support?



How many OPS can we support?

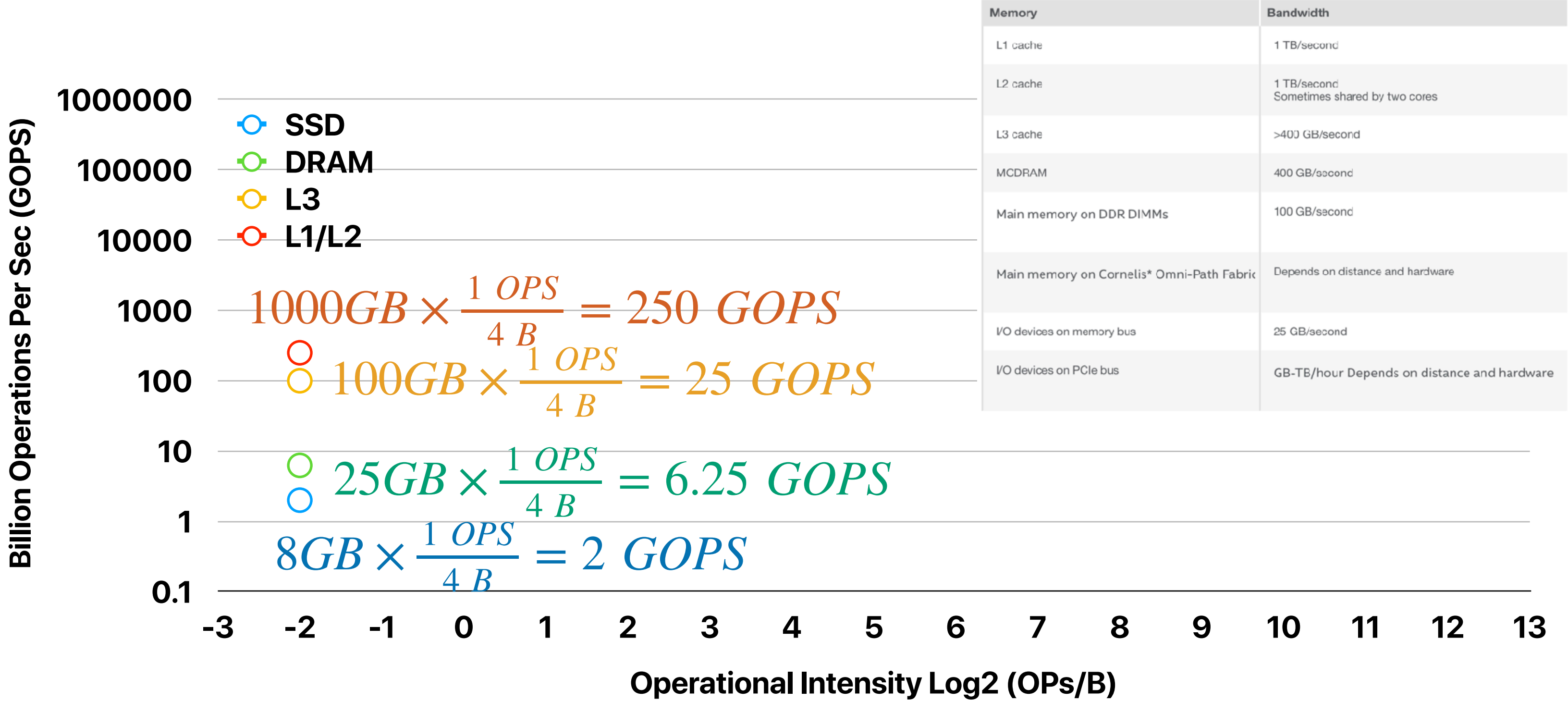


Size, Latency, and Bandwidth of Memory Subsystem Components

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

Sustainable OPS if we assume unlimited computing resource



Sustainable OPS if we assume unlimited computing resource

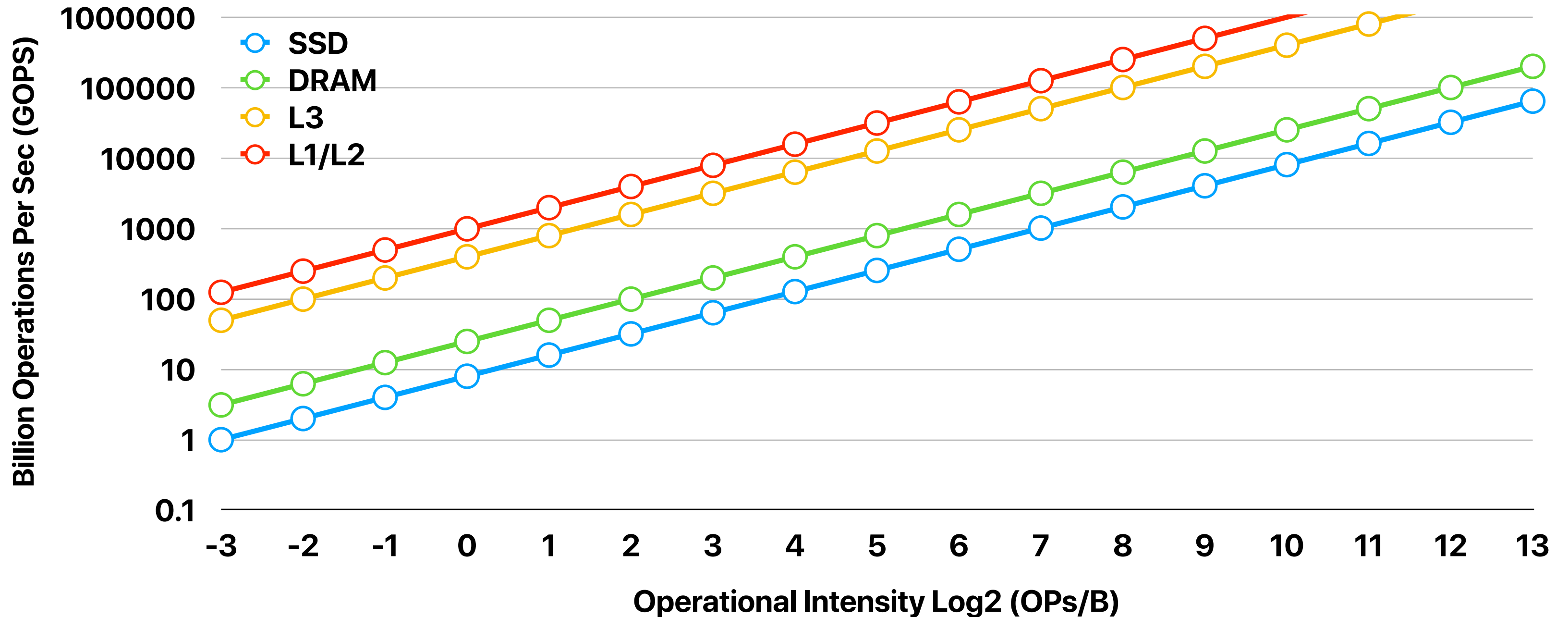
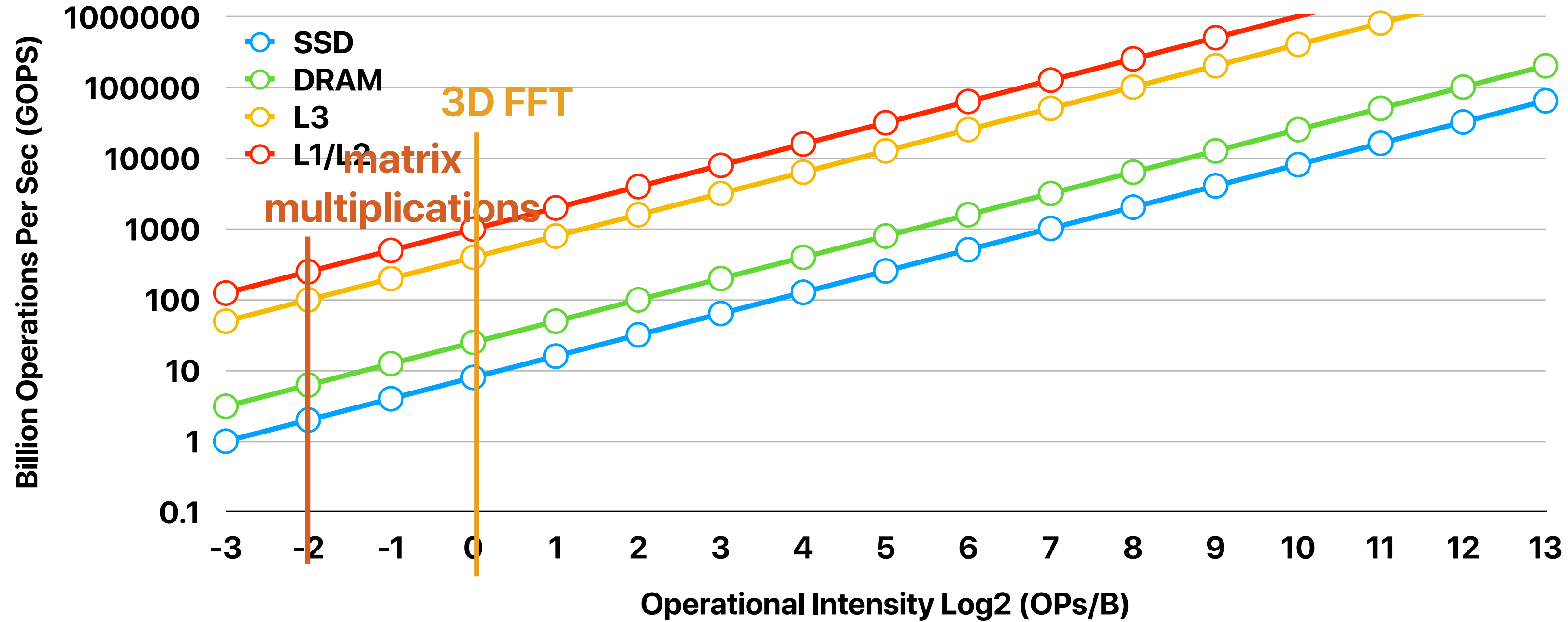


Table 2: Characteristics of four floating-point kernels.

Name	Operational Intensity	Description
SpMV²⁹	0.17 to 0.25	Sparse Matrix-Vector multiply: $y = A \cdot x$ where A is a sparse matrix and x, y are dense vectors; multiplies and adds equal.
LBMHD²⁸	0.70 to 1.07	Lattice-Boltzmann Magnetohydro-dynamics is a structured grid code with a series of time steps.
Stencil¹²	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).

Sustainable OPS if we assume unlimited computing resource



Transformer Inference

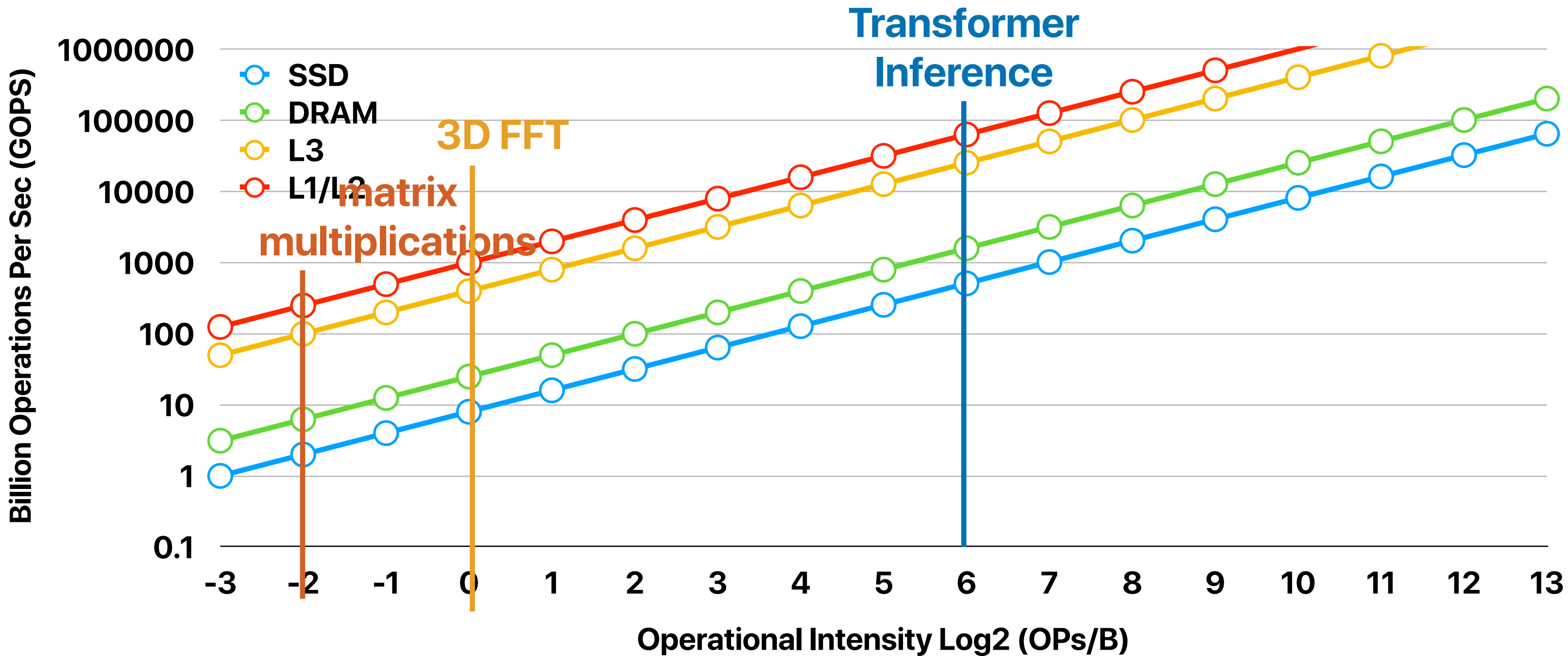
Let N be the sequence length, d be the dimension of a single attention head

- The total compute in floating point operations is approximately: $4(N^2)d + 3N^2$
 - **Attention Calculation** $(A = QK^T)$: $2(N^2)d$ FLOPs - Q and K have dimensions $N \times d$, and their dot product requires $2N^2d$ floating point operations.
 - **Softmax Operation** $A = \text{softmax}(A)/\text{sqrt}(d_k)$: $3N^2$ FLOPs - The attention matrix A has size N^2 . Calculating the exponent for each element requires N^2 FLOPs, and the normalization (sum and division across rows) requires an additional $2N^2$ FLOPs.
 - **Output Calculation** $O = A \cdot V$: $2(N^2)d$ FLOPs - A is of size $N \times N$ and V is $N \times d$. The dot product between A and V takes $2N^2d$ FLOPs.
- The total memory movement in bytes is approximately: $8N^2 + 8Nd$
 - **Attention Calculation** $A = QK^T$: $4Nd + 2N^2$ bytes - Q and K , each of size $N \times d$, require $2Nd$ bytes each in FP16 precision. The resulting attention matrix A , of size N^2 , takes $2N^2$ bytes.
 - **Softmax Operation** $A = \text{softmax}(A)/\text{sqrt}(d_k)$: $4N^2$ bytes - The input and output of the softmax operation, both of size N^2 , require $2N^2$ bytes each in FP16.
 - **Output Calculation** $O = AV$: $2N^2 + 4Nd$ bytes - The attention matrix A of size N^2 requires $2N^2$ bytes, and the vectors V and output O , both of size $N \times d$, require $2Nd$ bytes each.

Llama 7B: $N = 4096, d = 128$

$$\frac{4N^2d + 3N^2}{8N^2 + 8Nd} = \frac{4Nd + 3N}{8N + 8d} = 62.4242424 \approx 2^6$$

Sustainable OPS if we assume unlimited computing resource



**Finding another limitation — the
roofline model on the computing part**

How fast is a CPU?



The diagram illustrates the internal architecture of an Intel x86 Core. It is organized into several horizontal layers. At the top is the 'I-TLB + I-Cache' and 'Predict' block. Below this is the 'MSROM', 'Decode', and 'μop Cache' block. The 'μop Queue' follows. The next layer contains 'Allocate / Rename / Move Elimination / Zero Idiom'. Below that is the 'Scheduler'. The core is divided into two main execution units: 'INT' (Integer) and 'VEC' (Vector). The 'INT' unit includes 13 ports (Port 00 to Port 11) and various ALU, LEA, Shift, Mul, Mul-HI, JMP, and IDIV instructions. The 'VEC' unit includes FMA, FMA52, ALU, Shift, AMX, fpDIV, Shuffle, and FADD instructions. A '48KB Data Cache' and '1.25MB/2MB ML Cache' are also shown. A red box highlights the FMA, FMA52, and AMX blocks in the VEC unit.

New

Performance

x86 Core

A Step Function in CPU Architecture
Performance For the Next Decade of
Compute

A significant IPC boost at high power efficiency

Wider **Deeper** **Smarter**

- Better supports large data set and large code footprint applications
- Enhanced power management improves frequency and power
- Machine Learning Technology:** Intel® AMX – Tile Multiplication

All in a tailored scalable architecture to serve the
full range of Laptops to Desktops to Data Centers

Architecture Day 2021

intel.

50

How fast is a CPU?

- Clock rate: 3 GHz — 3B cycles/instructions per “ALU”
- 2 FMA units can perform 2 256-bit vector floating point operations per core (128 pairs of floating point numbers)
- 384B floating point operations per second — or say 384 GFLOPS

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

$$\bullet \quad \text{Attainable GFlops/sec} = \min \left\{ \frac{\text{Peak Floating-Point Performance}}{\text{Peak Memory Bandwidth} \times \text{Operational Intensity}} \right\}$$

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

$$8GB \times \frac{1 \text{ OPS}}{4 \text{ B}} = 2 \text{ GOPS} < 384 \text{ GFLOPS}$$

- If data are currently stored in DRAM

- We can supply data in 25GB/sec per module
- The supplied data per second needs per module

$$25GB \times \frac{1 \text{ OPS}}{4 \text{ B}} = 6.25 \text{ GOPS} < 384 \text{ GFLOPS}$$

- If we have 2 modules

$$50GB \times \frac{1 \text{ OPS}}{4 \text{ B}} = 14.5 \text{ GOPS} < 384 \text{ GFLOPS}$$

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

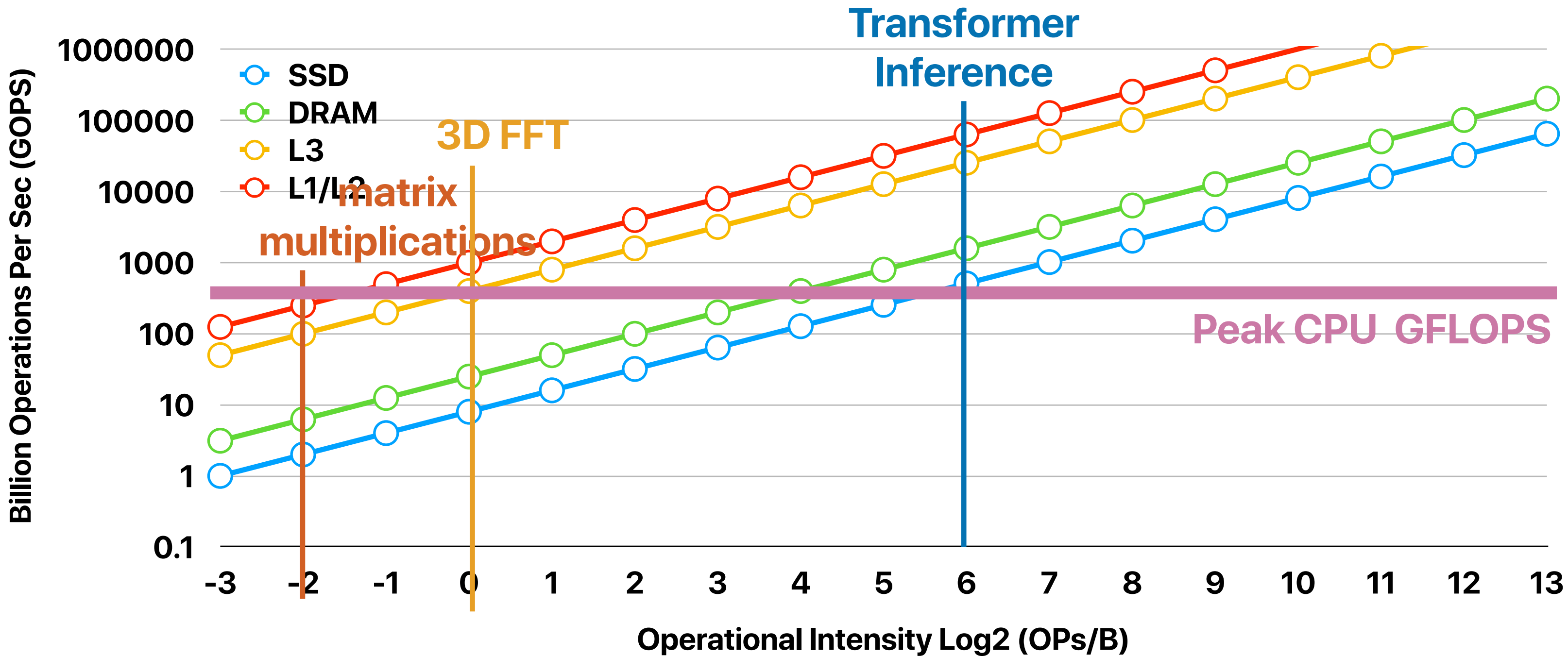
Clock cycles between row activate and reads/writes

tRP

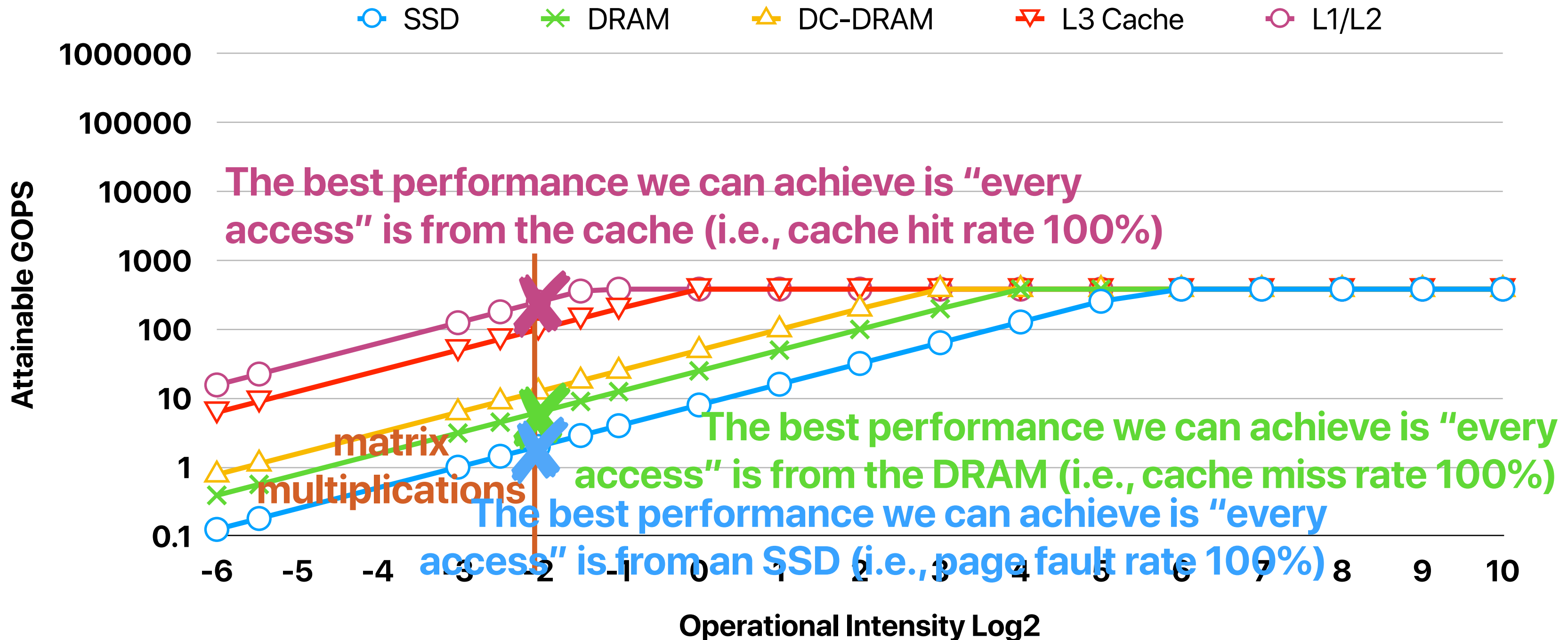
Clock cycles between row precharge and activate

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]

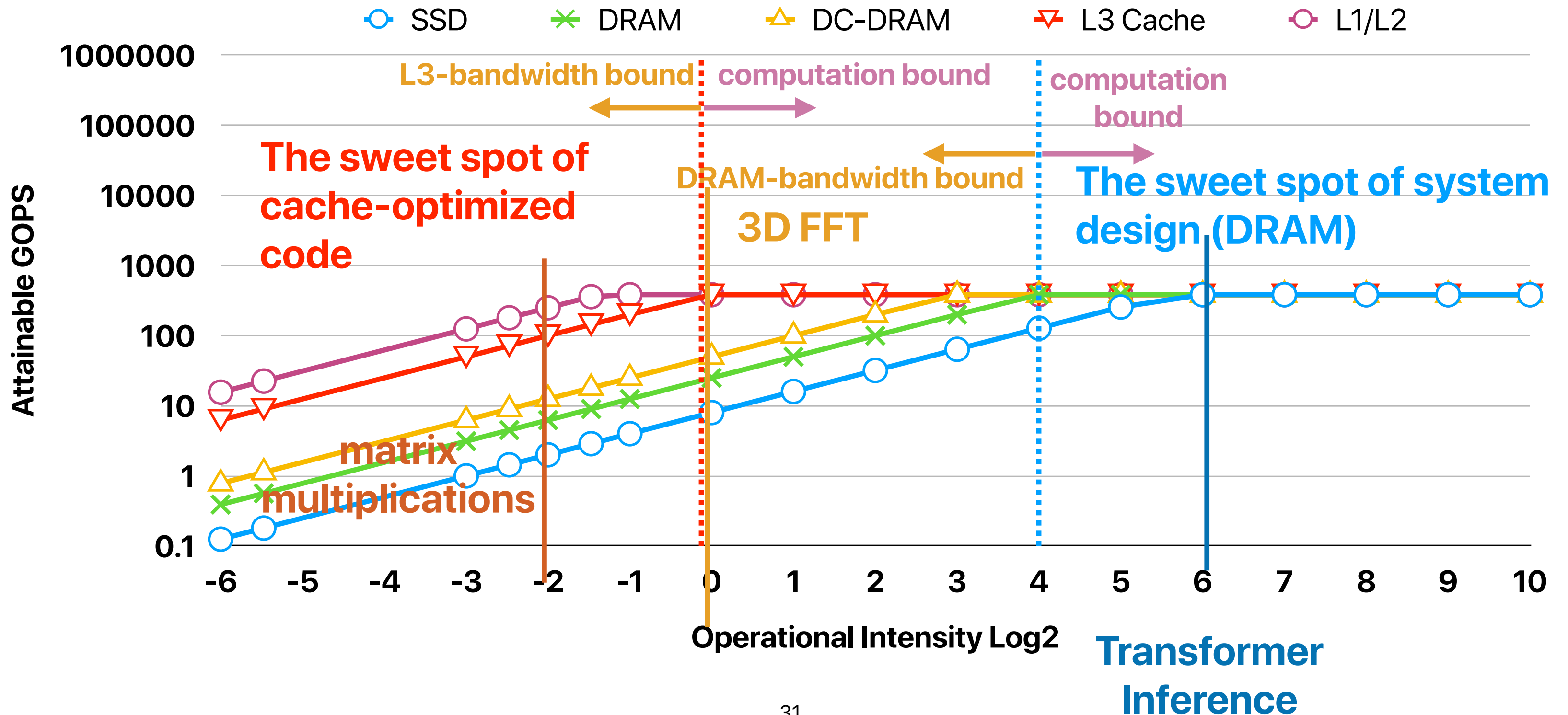
Sustainable OPS if we assume unlimited computing resource



The roofline under various configurations



The roofline under various configurations



1.6 BFLOAT16 FLOATING-POINT FORMAT

Intel® Deep Learning Boost (Intel® DL Boost) uses bfloat16 format (BF16). Figure 1-6 illustrates BF16 versus FP16 and FP32.

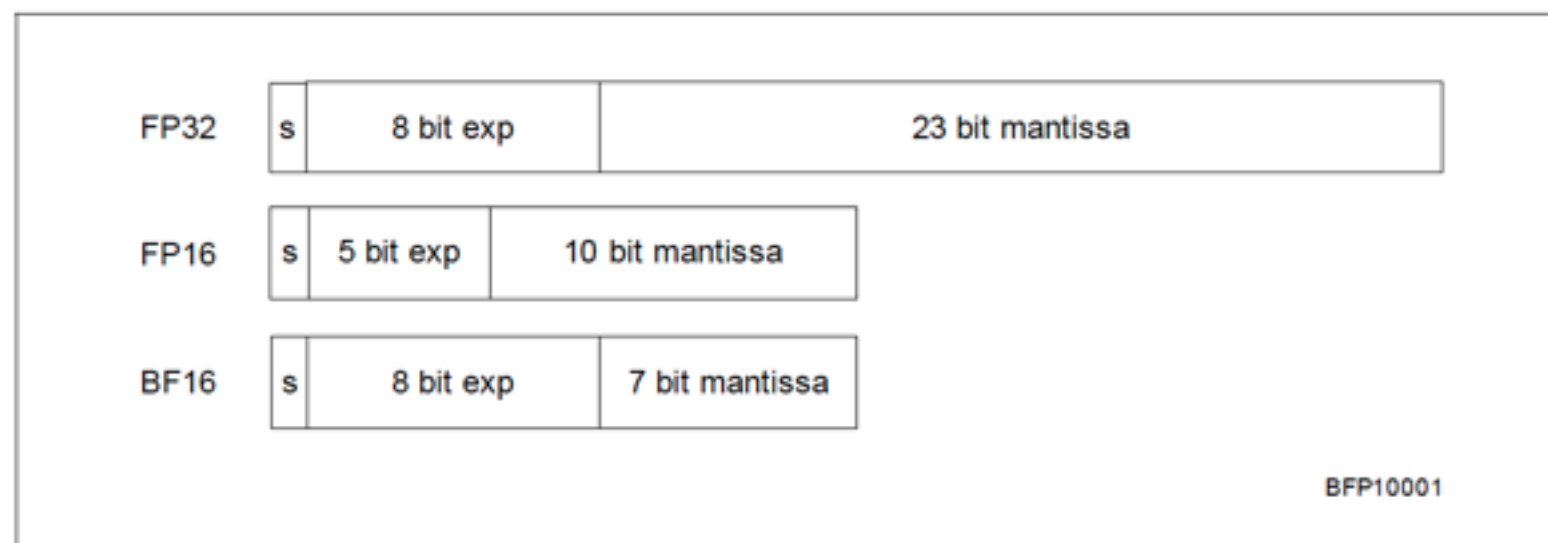


Figure 1-6. Comparison of BF16 to FP16 and FP32

 Run in Google Colab

 View source on GitHub

 Download notebook

Overview

Mixed precision is the use of both 16-bit and 32-bit floating-point types in a model during training and use less memory. By keeping certain parts of the model in the 32-bit types for numeric stability, you can achieve a lower step time and train equally as well in terms of the evaluation metrics such as accuracy. To use the Keras mixed precision API to speed up your models, using this API can improve performance times on modern GPUs, 60% on TPUs and more than 2 times on latest Intel CPUs.

How does “half-precision” floating point changes the roofline and system design?

16-bit 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 =

$$2 \times M \times N \times P \text{ ops}$$

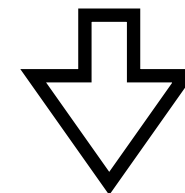
- Total size of data accesses =

$$2 \times (2 \times N + 1) \times M \times P \text{ bytes}$$

- Average operations per byte =

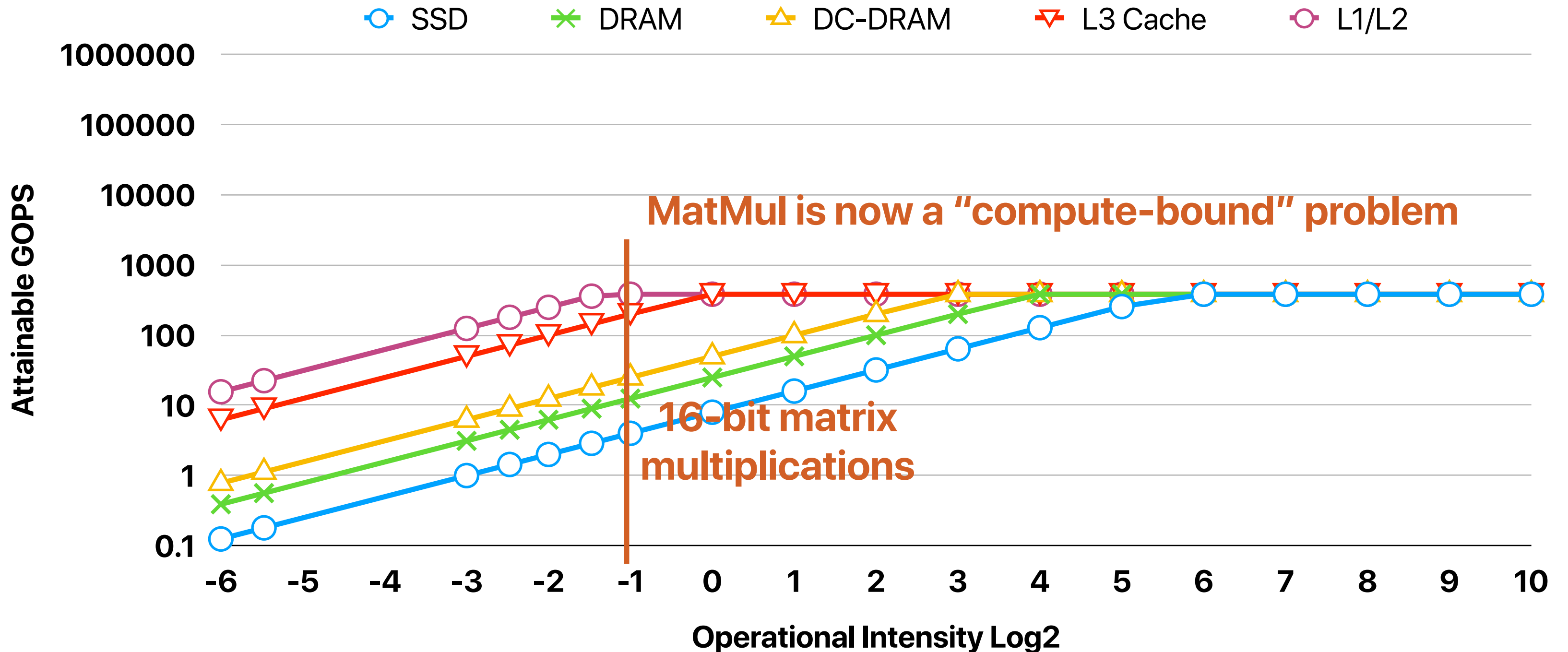
$$\frac{2 \times M \times N \times P}{2 \times (2 \times N + 1) \times M \times P} = \frac{1}{2} = 0.5$$

```
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;  
    }  
}
```

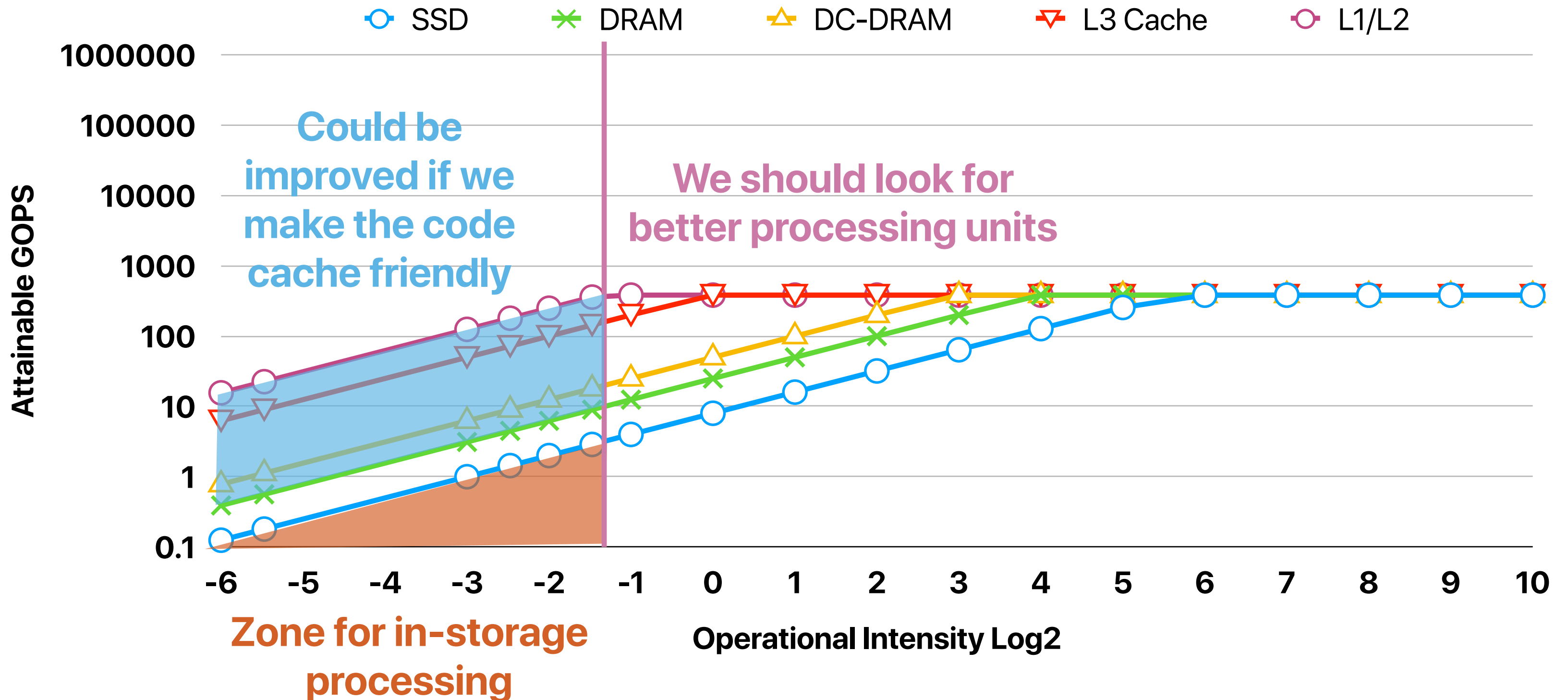
Where does 16-bit sit?



**Reduced precision makes a problem
more “compute-bound” and less
memory bandwidth dependent**

Recap on the roofline model, how do you decide if we want to go for improved processors or alternative architectures (e.g., near-data processing)

Where is my application?



When should we use
1. Performance equation
2. Amdahl's law
3. Roofline model

PE v.s. Amdahl's law v.s. Roofline model

- Performance equation
 - Useful to compare the two implementations and identify the reason of improvement
 - Cannot guide what to do
- Amdahl's law
 - Only tell us where to improve
 - Does not tell us by how much can we improve on "f"
- Roofline model
 - Tell us the roofline of improvement
 - Tell us if we maximize the improvement
 - Only work for each software component

Summary of the roofline model

- The roofline model defines the “best performance” we can achieve under a certain architecture
 - Tells us how good we’re in optimizing our code
 - We can never surpass the roofline without changing the hardware
- The roofline model helps us to identify which direction to go for optimization
 - Memory-bounded
 - Can we more efficiently use cache?
 - Can we reduce the data volume?
 - Compute-bounded
 - Can we make the algorithm more efficient?
 - Can we use more or other processing units?

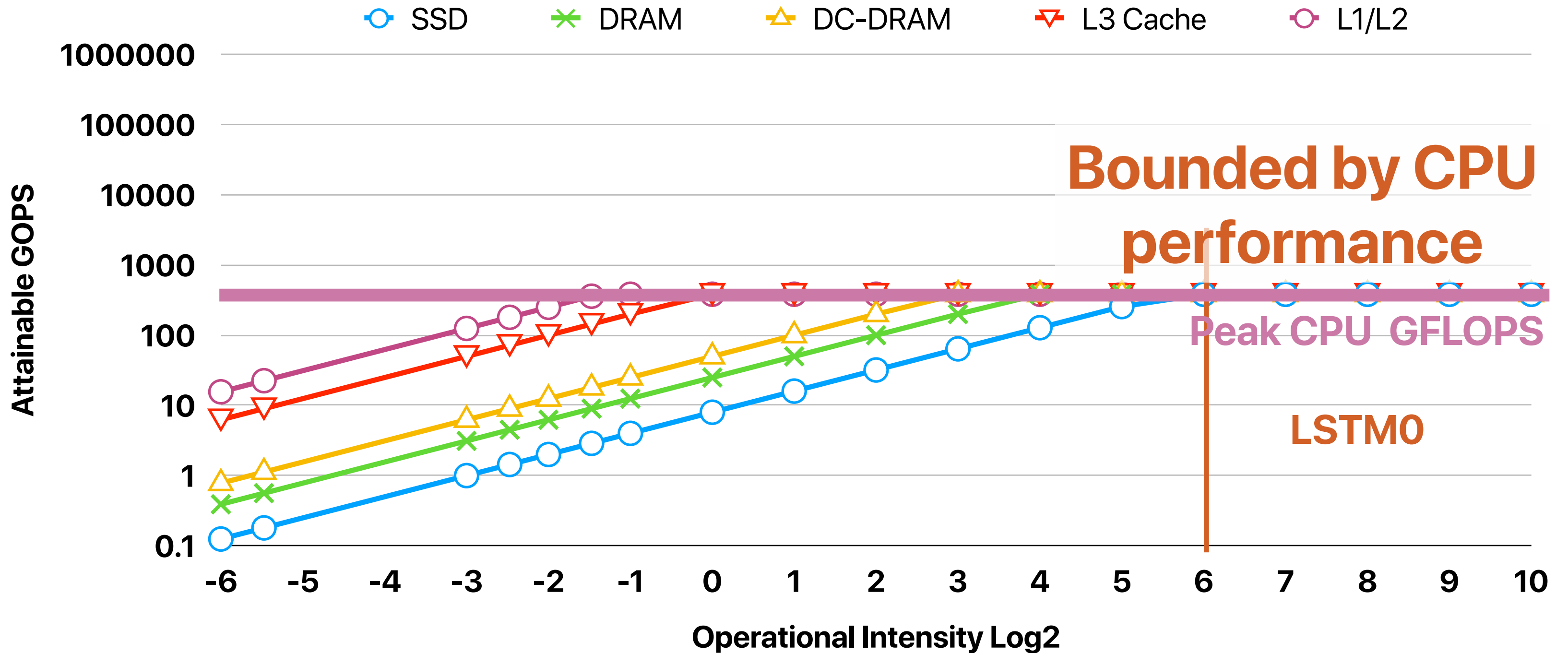
What's the operational intensity of modern workloads?

Table 1. Six DNN applications (two per DNN type). MLP stands for multi-layer perceptron, and LSTM stands for long short-term memory.

Name	Lines of Code	Weights	TPU Ops / Weight Byte	Operational Intensity Log2
MLP0	100	20 M	200	7.64385618977472
MLP1	1,000	5 M	168	7.39231742277876
LSTM0	1,000	52 M	64	6
LSTM1	1,500	34 M	96	6.58496250072116
CNN0	1,000	8 M	2,888	11.4958550268872
CNN1	1,000	100 M	1,750	10.7731392067197

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, May./Jun. 2018, doi: 10.1109/MM.2018.032271057.

Where is LSTM0 in the roofline?



Don't forget...

- Form your group and discuss the project ideas
- Check the schedule/Google Spaces for up-to-date reading list and submit the summary!
- Project ideas
 - Accelerating applications through AI/ML accelerators (e.g., EdgeTPUs or tensor cores)
 - Accelerating applications through innovative parallel programming models that hardware accelerators enable and evaluate the result using FPGAs
 - Static tools to perform roofline analysis on AI/ML workloads
 - Accelerating applications through intelligent storage devices (smartSSD)
 - Literature review (more than 20 papers) on a certain topic related to this class
 - Anything related to what we discussed in this class!
- Decide & discuss your topics of interests with me before the 3rd week

Electrical Computer Science Engineering

277

つくづく

