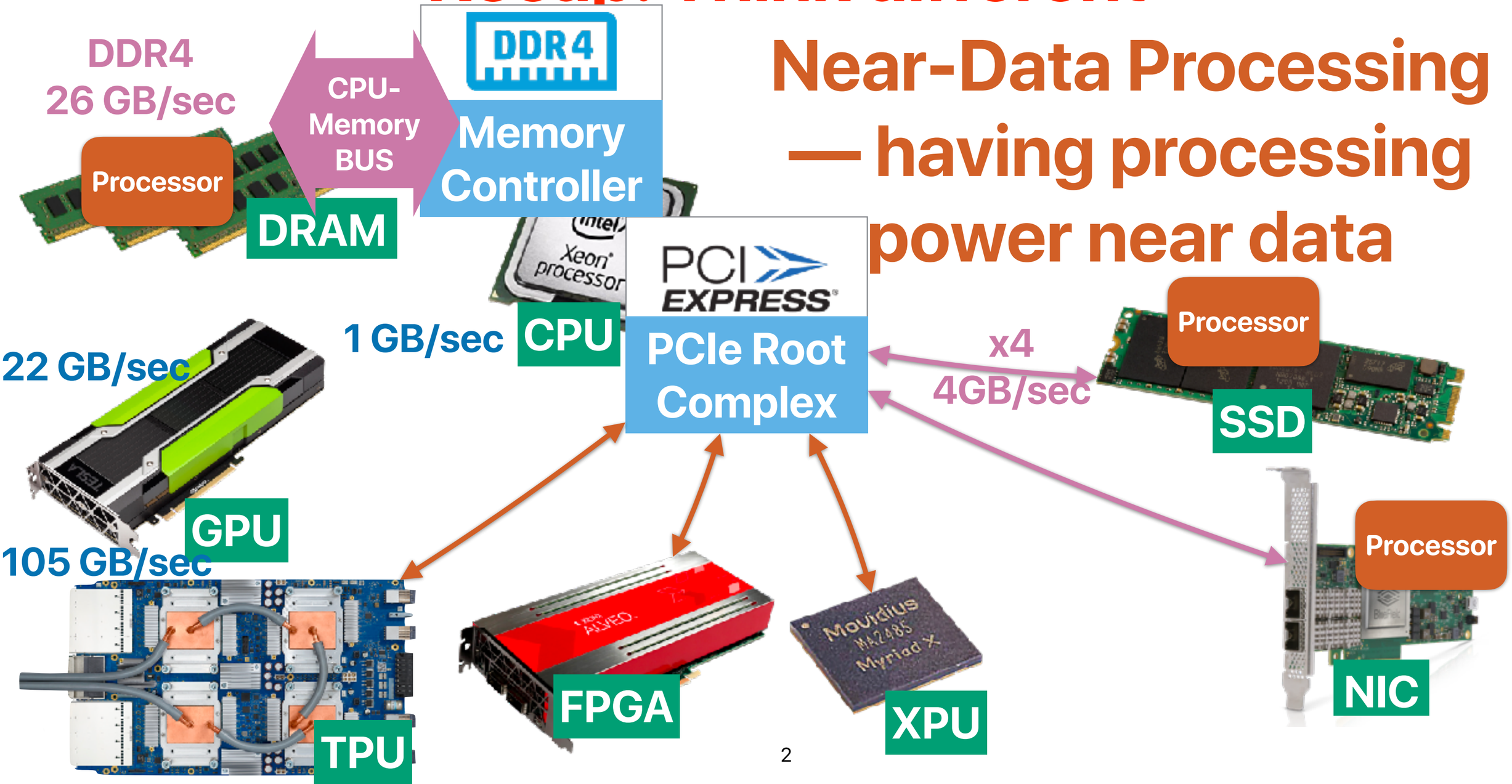
Think Different (2): In-Storage Processing — Practice

Hung-Wei Tseng

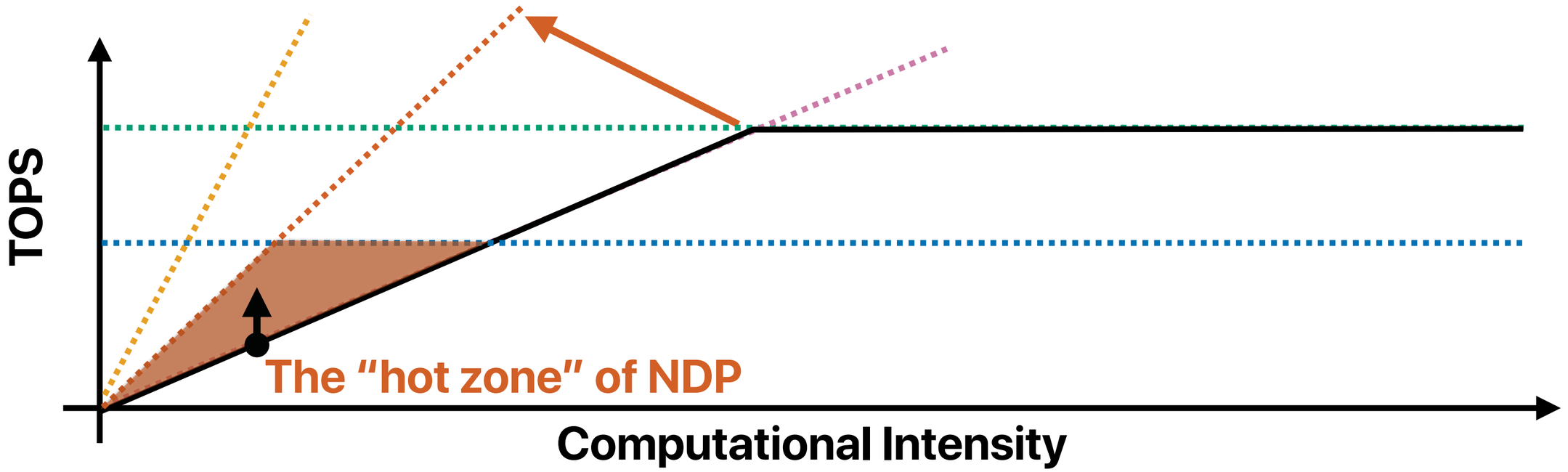
Recap: Think different

Near-Data Processing — having processing power near data



Reviewing the roofline model

min(Peak OPS of target computing resource, Peak memory bandwidth × computational intensity ÷ reduction of data volume, Peak OPS of target NDP device, Peak device internal bandwidth × computational intensity of NDP program)



Recap: The "Winning Formula" of Near-Data Processing

Conventional Model

NDP Model

- The NDP computation can help offload computation
- The NDP computation can help reduce data volume
- The NDP computation can facilitate data preprocessing
- The NDP computation itself cannot be too slow

Not the cases of Near-Data Processing

Conventional Model

NDP Model

- The NDP processing exceeds NDP computing resources' capabilities
- The NDP processing does not help reduce data volume
- The NDP device itself does not offer rich internal bandwidth to the computing resource near-by

Why do we need controllers in SSDs?

- Interfacing with the interconnect
- Maintaining the block device abstraction
  - Mapping between physical & logical addresses
  - Exploiting channel-level parallelism
- Dealing with the “weird” device characteristics
  - Variance in reads
  - Very long write/program latency
  - Limited erase counts
  - Garbage collections

Why? Multiple channels to improve bandwidth

Flashtec™ NVMe2032 and NVMe2016 Controllers

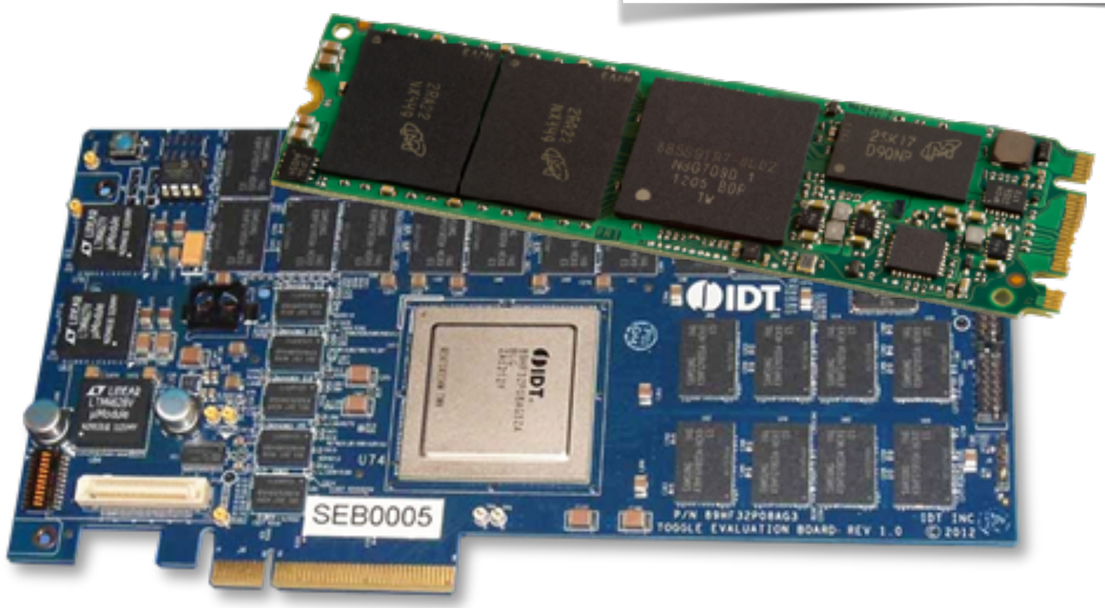
32- and 16-Channel PCIe Flash Controller Pr

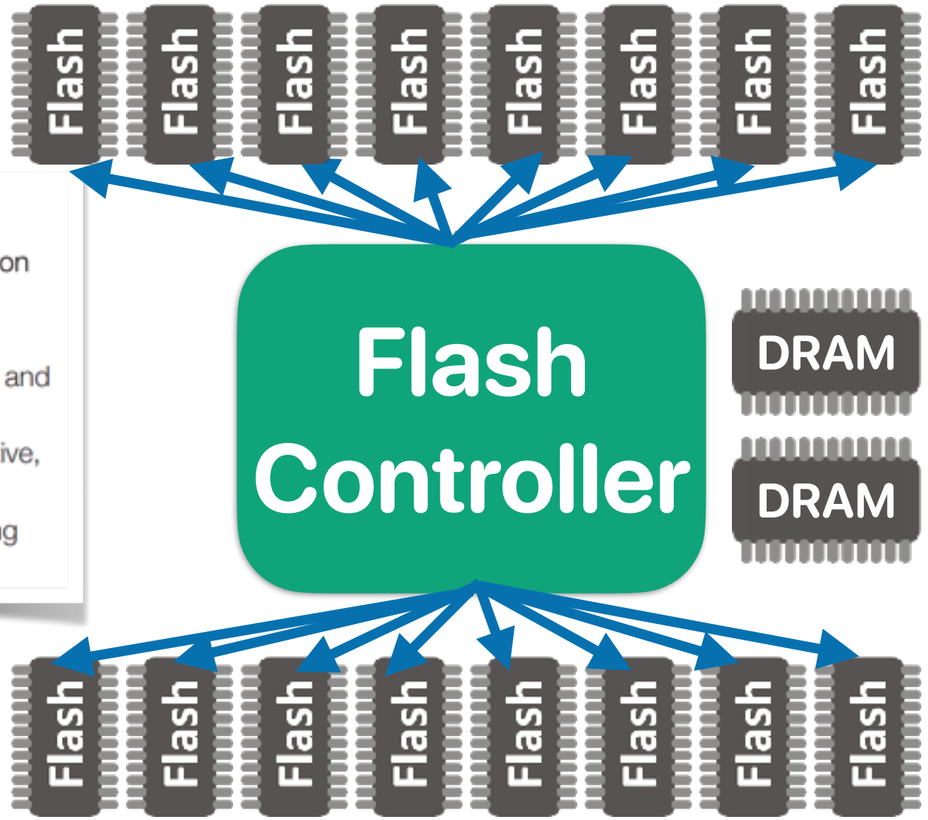
Summary

The Flashtec™ 2nd generation NVMe Controller Family enables the
and data centers to realize the highest performance SSDs utilizing n
technologies. Combining world-class capacity and flexibility, the Flas
reliable choice. The Flashtec NVMe2032 and NVMe2016 controllers
Express (NVMe) host interface and are optimized for high-performan
operations, performing all flash management operations on-chip an
processing and memory resources.

Features

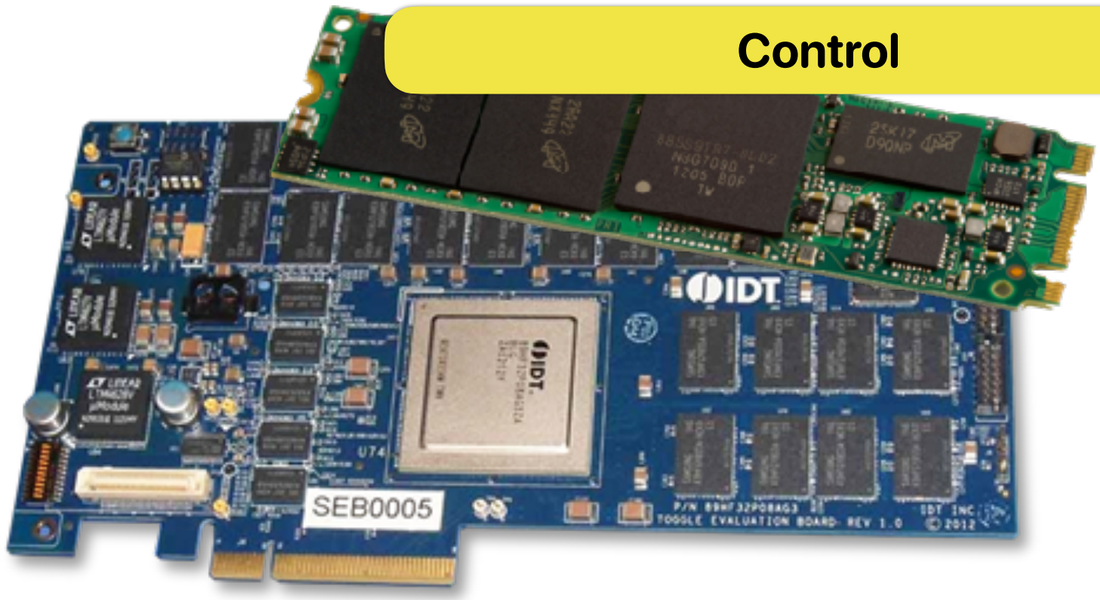
- Flashtec NVMe2032 controller can achieve up to 1 million random read IOPS on 4 KB operations
- Up to 20 TB Flash capacity using 256 GB Flash
- SLC, MLC, Enterprise MLC, and TLC Flash with toggle and ONFI interface
- PCIe Gen 3 x8 or dual independent PCIe Gen 3 x4 (active, active/standby) host interface
- 16 and 32 independent Flash channels, each supporting up to 8 CE

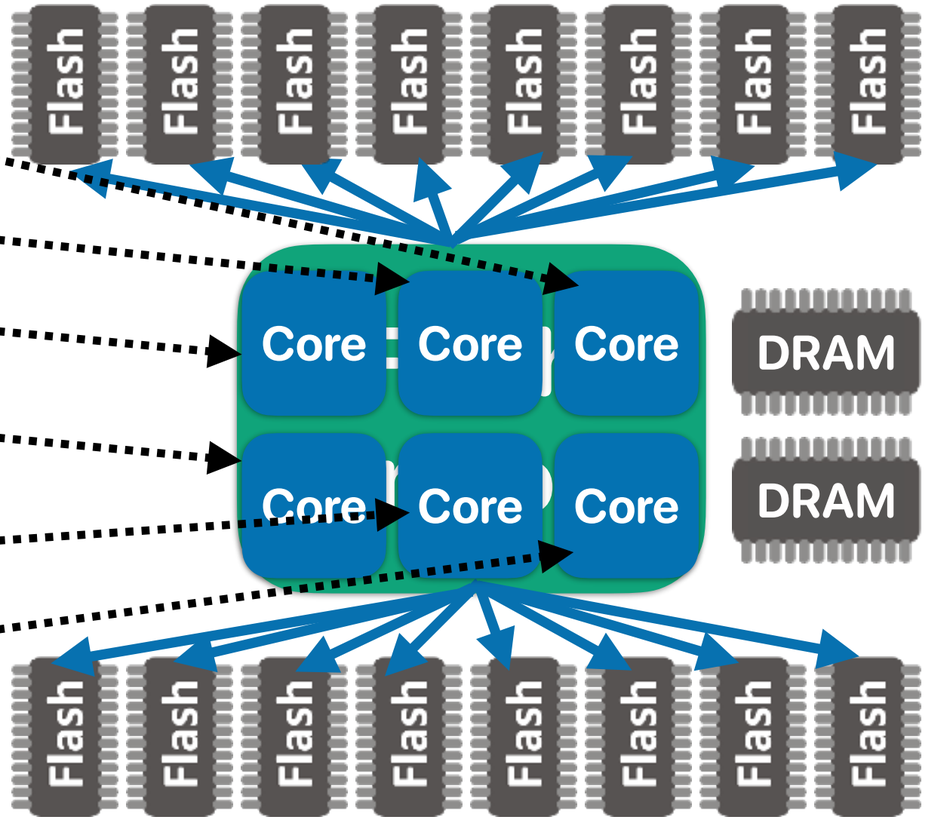




SSD Controller Architectures

- Parse NVMe/SATA/PCIe commands
- Address mapping/lookup
- Wear-leveling/Garbage Collection
- Read
- Program
- Control





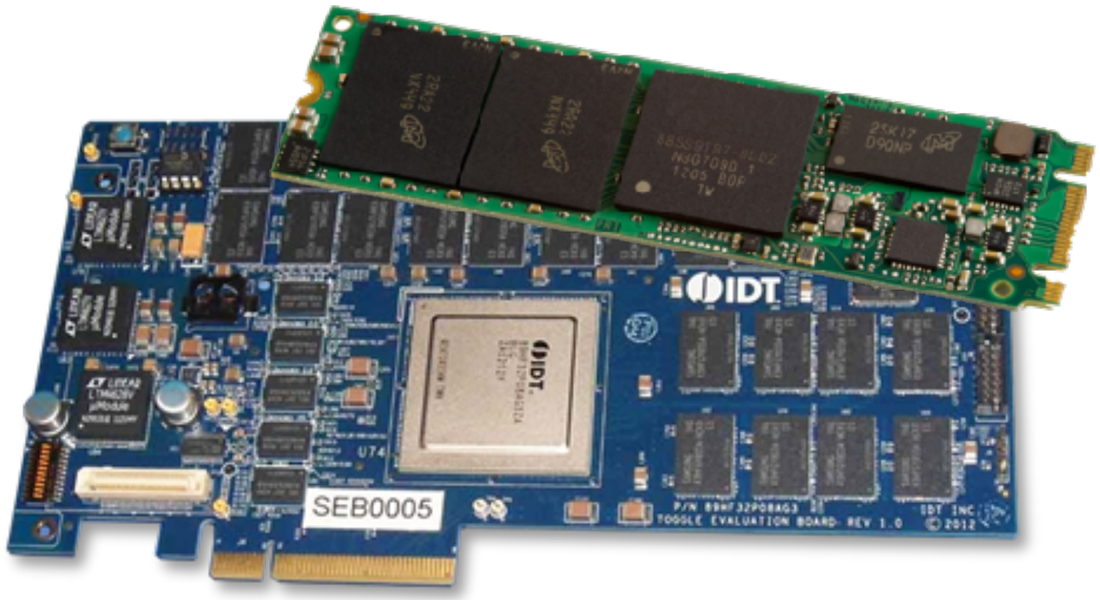
3:00

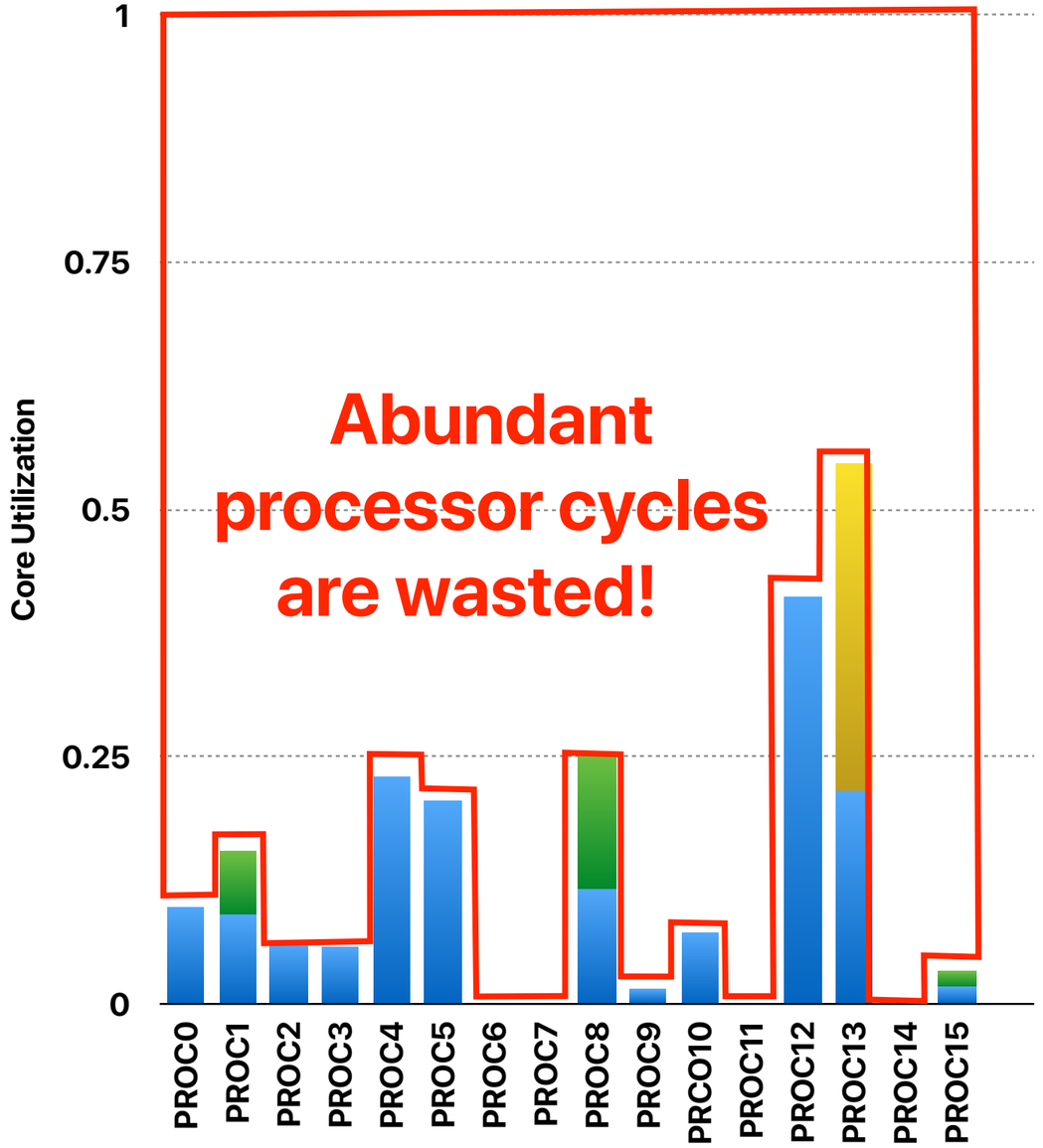
What is Firmware? Why firmware?

Firmware

- What is firmware?
  - A software program directly interacts with hardware without an operating system
  - Typically a program works in between hardware and application "software"
- Why firmware?
  - You don't need to implement an application-specific IC for each feature
  - You can upgrade the firmware to get new features (or fix a bug)
  - General-purpose processors are so cheap and offering "good enough" performance for most tasks

Are we using them “efficiently”?



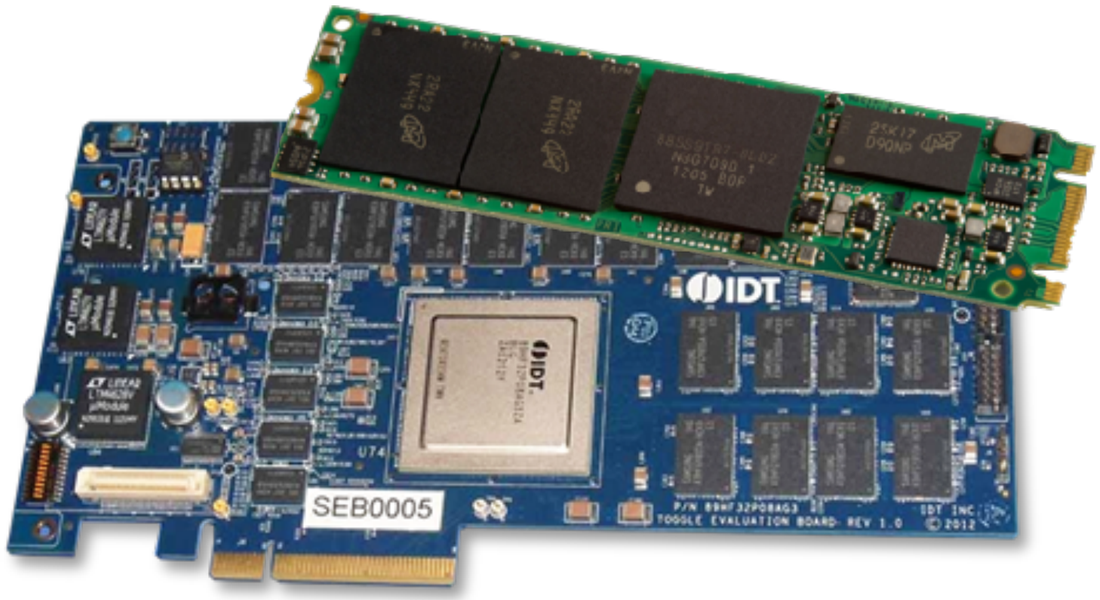


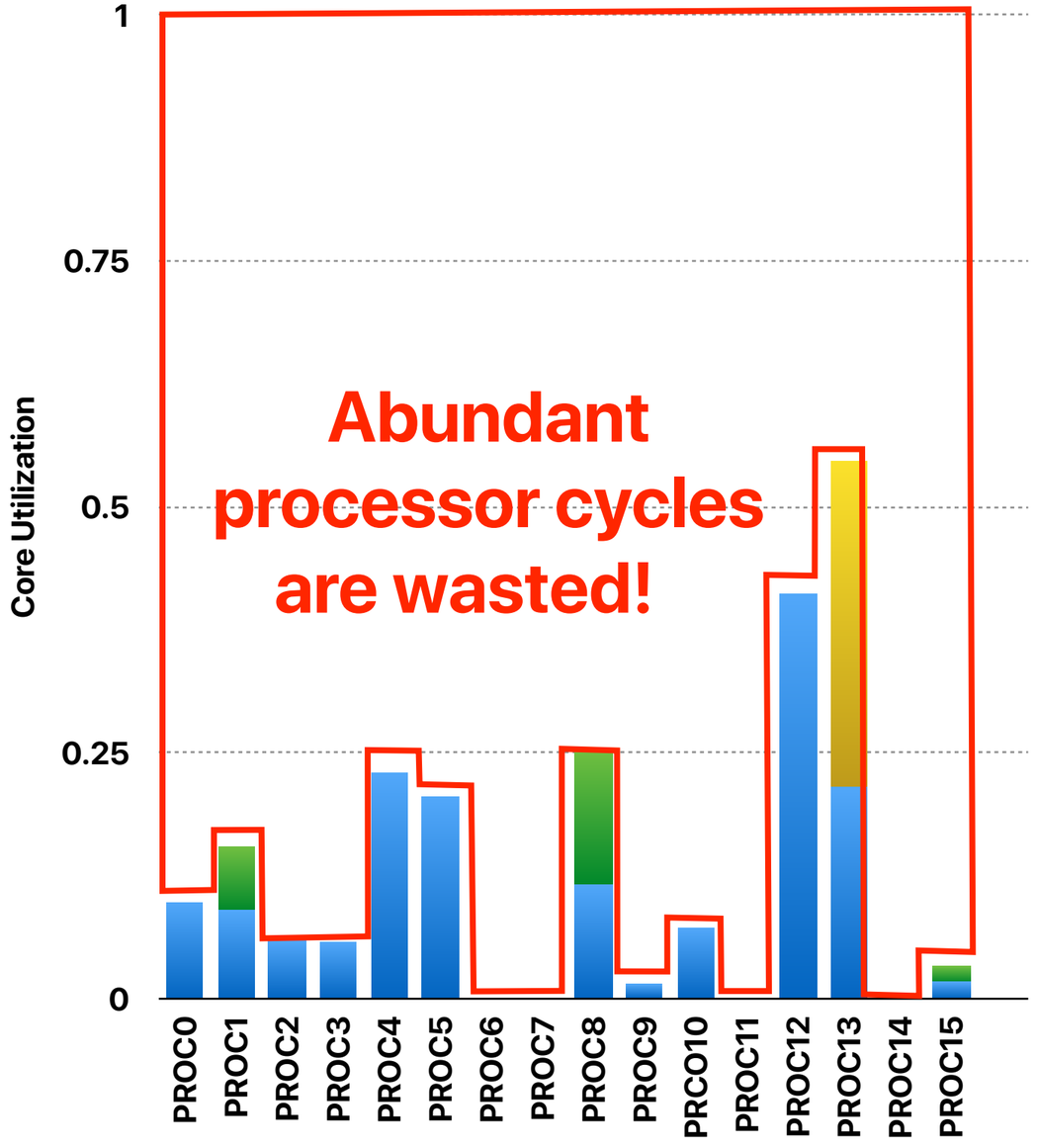
3:00

Why are we under utilizing these processors?

Are we using them “efficiently”?

- Firmware does not rely on OS
- Utilization is not optimized as well
- We can “tweak” and “extend” the firmware to perform computation!





3:00

If you can extend the firmware — what are you going to do to improve performance?

What applications/features do you have in mind?

NVMe

- The standard of PCIe SSD devices now
  - Provides multiple command queues to better support multithreading hardware
  - Allows more parallelism inside the SSD
- The "payload" of a PCIe packet

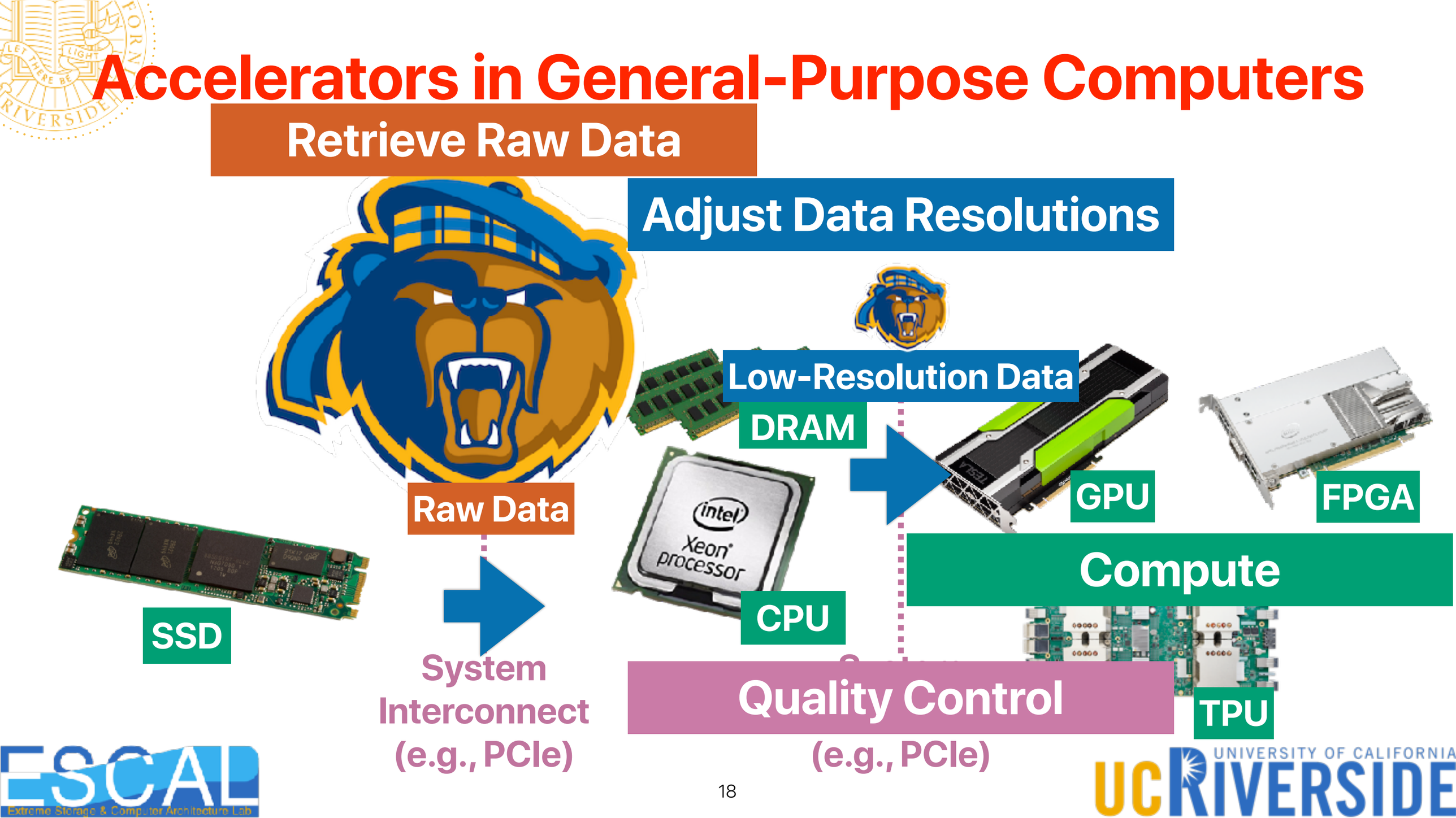
We can implement "new commands" by using these fields

0	8	16	32	48	63
OPCODE	FLAGS	command id	Namespace ID		
reserved					
metadata					
PRP1					
PRP2					
Start LBA					
length		control	Dataset management		
Reference tag			App tag	App mask	

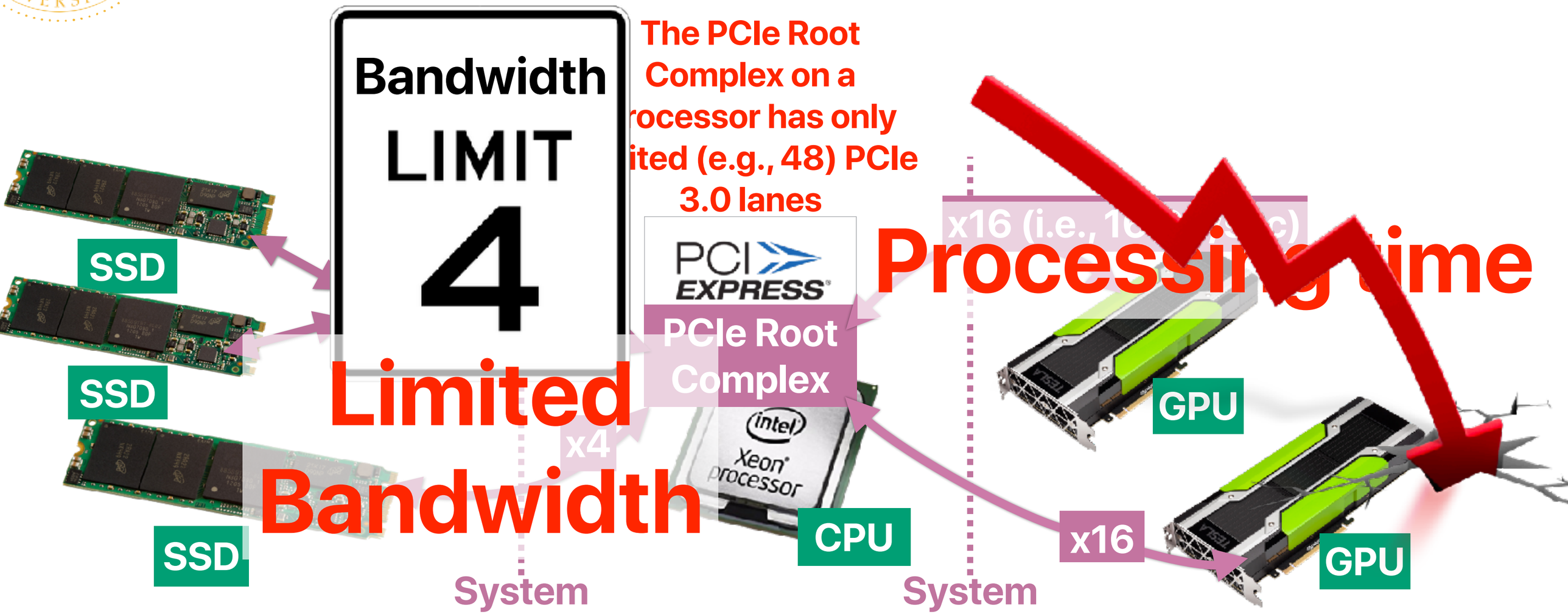
Example — Varifocal Storage✳

✳ Yu-Ching Hu, Murtuza Lokhandwala, Te I and Hung-Wei Tseng. Dynamic Multi-Resolution Data Storage. In the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019 (Best paper nominee)

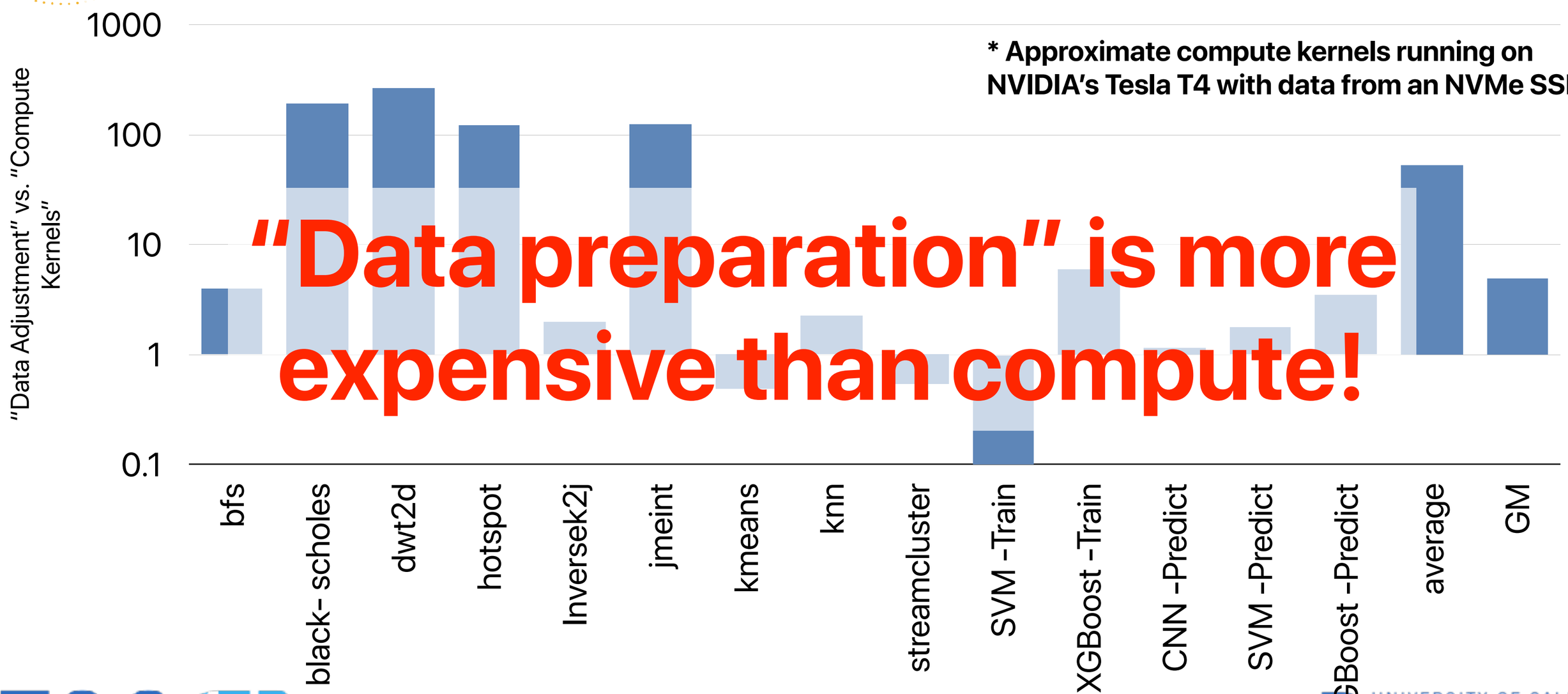
Accelerators in General-Purpose Computers



Limited Interconnect Bandwidth

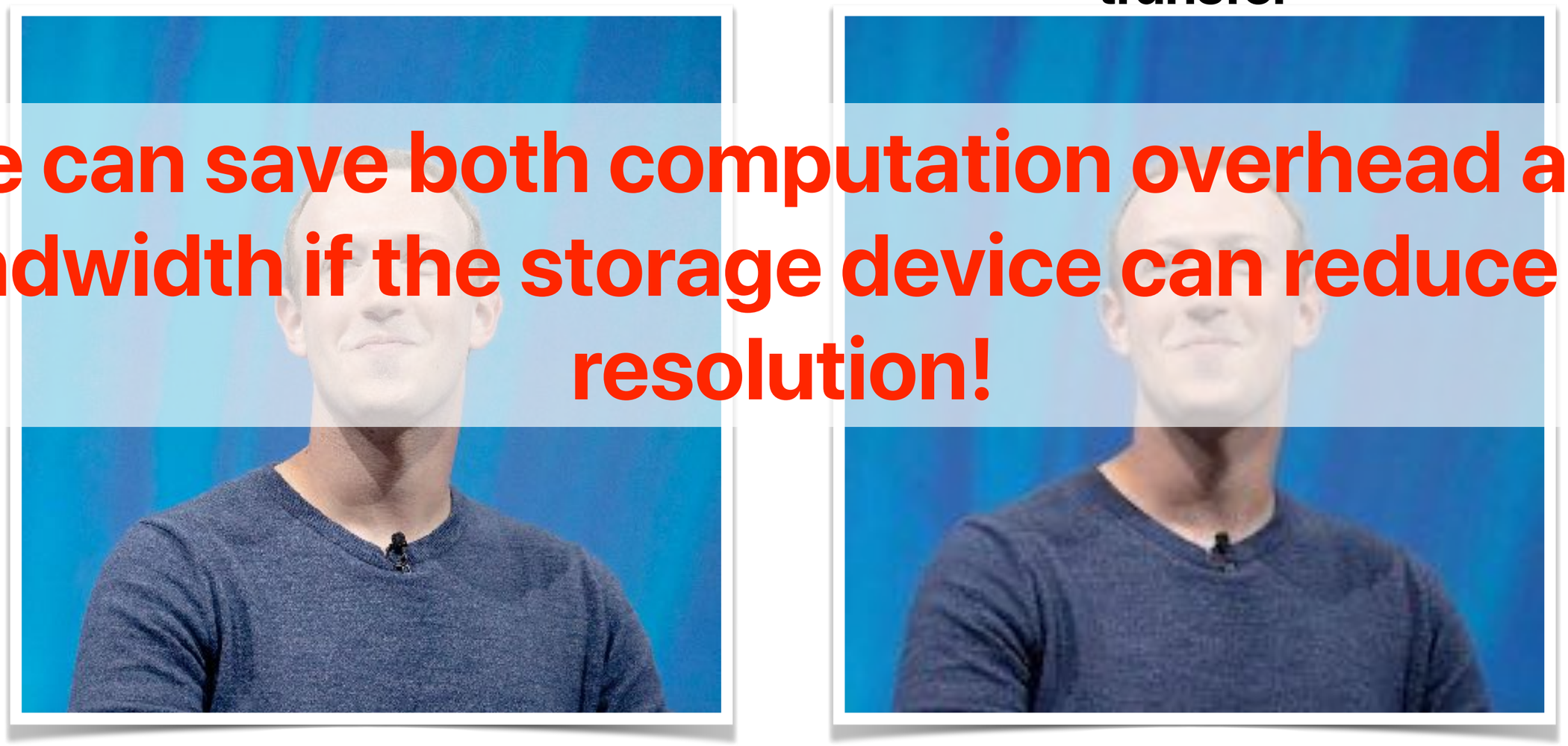


Data Preparation Can be 100x More Than Compute Kernels



We don’t need full resolution in many cases

Only need 25% bytes to transfer



We can save both computation overhead and bandwidth if the storage device can reduce the resolution!

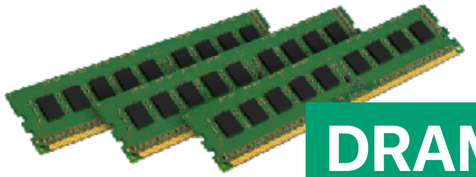
Varifocal Storage: A Holistic System Architecture

Retrieve Raw Data

Adjust Data Resolutions

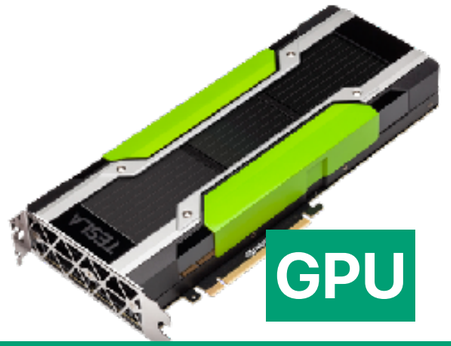
Varifocal Storage (VS)

System Interconnect (e.g., PCIe)

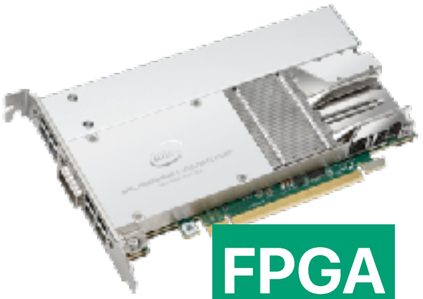


DRAM

CPU

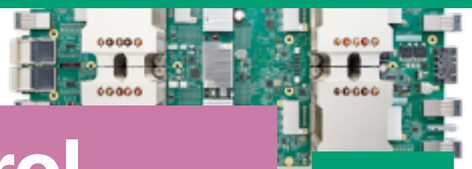


GPU



FPGA

Compute

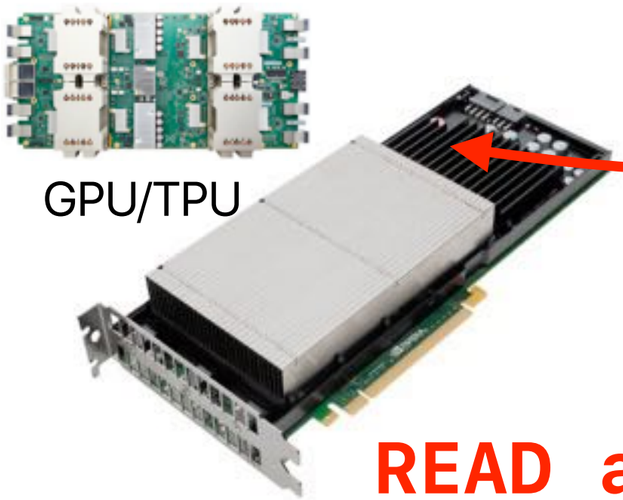


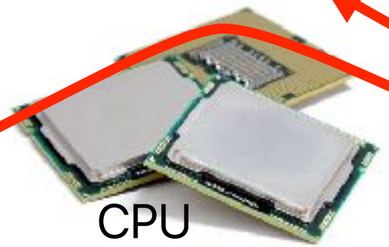
TPU

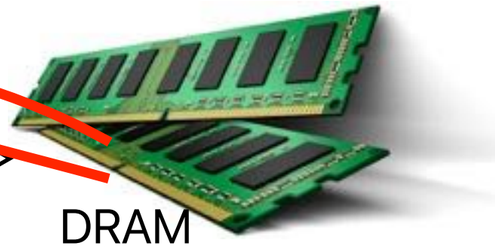
Quality Control

System Interconnect (e.g., PCIe)

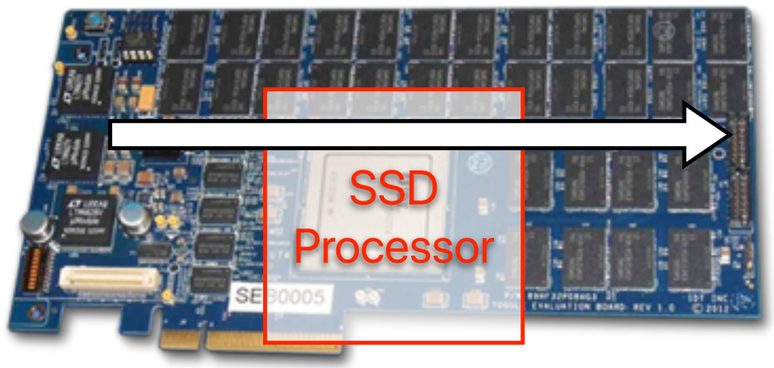
The “concept” of Varifocal Storage







`READ address(mark_zuckerberg), reduce(25%)`



extension of
NVMe interface

The “Winning Formula” of Near-Data Processing

$$\frac{8GB}{8GBps} + \frac{8GB \times 0.25}{2GOPS} + \frac{2GB}{2.5GBps} + \frac{2GB \times 1}{1000GOPS} = 2.4$$

$$\frac{DataVolume_{raw}}{Bandwidth_{device}} + \frac{DataVolume_{raw} \times ComputationIntensity_{NDP}}{ComputationThroughput_{device}} + \frac{DataVolume_{afterNDP}}{Bandwidth_{device2host}} + \frac{DataVolume_{afterNDP} \times ComputationIntensity_{afterNDP}}{ComputationThroughput}$$

$$<= \frac{DataVolume_{raw}}{Bandwidth_{device2host}} + \frac{DataVolume_{raw} \times ComputationIntensity}{ComputationThroughput}$$

$$\frac{8GB}{2.5GBps} + \frac{8GB \times 1}{1000GOPS} = 3.2$$

NVMe

- The standard of PCIe SSD devices now
  - Provides multiple command queues to better support multithreading hardware
  - Allows more parallelism inside the SSD
- The "payload" of a PCIe packet

VS_READ

REDUCE/QUTIZATION/
PACKING/SAMPLING

PARAMETER

0	8	16	32	48	63
OPCODE	FLAGS	command id	Namespace ID		
reserved					
metadata					
PRP1					
PRP2					
Start LBA					
length		control	Dataset management		
Reference tag			App tag	App mask	

Programming model of Varifocal Storage

```
int setup(int argc, char **argv)  {
    // Skip – the rest of code ...
    int infile;

    // VS: Declare VS variables
    struct vs_operator op[1];
    struct vs_feedback fb[1];

    // Open a file descriptor
    infile = open(filename, O_RDONLY, "0600");

    // Read precise data from the file descriptor
    read(infile, &npoints,   sizeof(int));
    read(infile, &nfeatures, sizeof(int));

    // Skip – some other initialization code ...

    // VS: set parameters for desired operator
    // PACKING(default)/PACKING_AF(autofocus)/VS_IF(iFilter)
    op[0].op = PACKING;
    op[0].resolution = HALF;

    // Skip – some other initialization code ...
    // VS: apply the desired VS operator for the file
    vs_setup(infile, &op, "%f");
    // VS: read data processed by the VS operator
    vs_read(infile, buf, npoints*nfeatures*sizeof(float), &fb);
    // VS: disable the usage of VS operator for the file
    vs_release(infile);
    // Skip – the rest of code ...
    // VS: use approximate kernel if the operator succeed
    if(fb[0].resolution == op[0].resolution)
        cluster_approximate(…);
    else
        cluster(…);
    // Skip – the rest of code ...
}
```

```
void test_distributed_page_rank(char* graphfilename,
int num_ofVertex, int num_ofEdges, int iterations)
{
    FILE *fin;
    vs_stream input_stream;
    void **arg_list;
    fin = fopen(graphfilename, "r");
    vs_input_stream = vs_stream_create(fin);
    Edge *edge_array = (Edge *)malloc(
                              sizeof(Edge)*num_ofEdges);
    inputApplet(input_stream, edge_array);
    // The rest of code ...

}
```

```
StorageApp int inputApplet
(ms_stream ssd_input_stream, void *edge_array)
{
    Edge ssd_edge_array[4096];
    int i = 0;
    while(vs_scanf(ssd_input_stream, "%d %d",
                   &ssd_edge_array[i%4096].first,
                   &ssd_edge_array[i%4096].second)==2)
    {
        i++;
        if(i % 4096 == 0)
        {
            vs_memcpy(edge_array, ssd_edge_array,
                      sizeof(Edge)*4096);
            edge_array = sizeof(Edge)*4096;
        }
    }
    vs_memcpy(edge_array, ssd_edge_array,
                          sizeof(Edge)*(i%4096));
    return i;
}
```

Programmer has to implement

Traditional Quality Control

Raw Data

Lower-resolution

25% Resolution

Compute

Test

Tiger

Reject Result

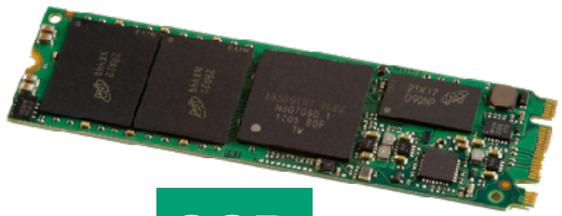
Better-resolution

50% Resolution

Re-compute

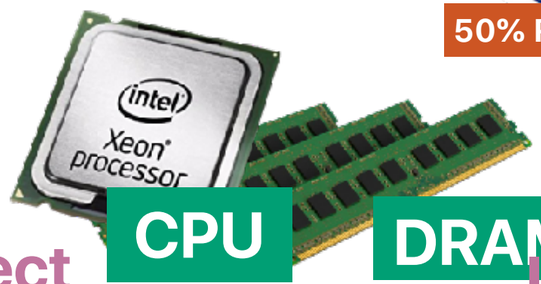
Test

Bear



SSD

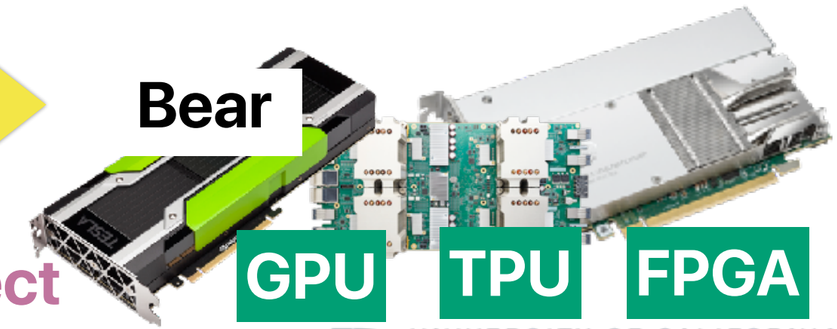
System Interconnect (e.g., PCIe)



CPU

DRAM

System Interconnect (e.g., PCIe)

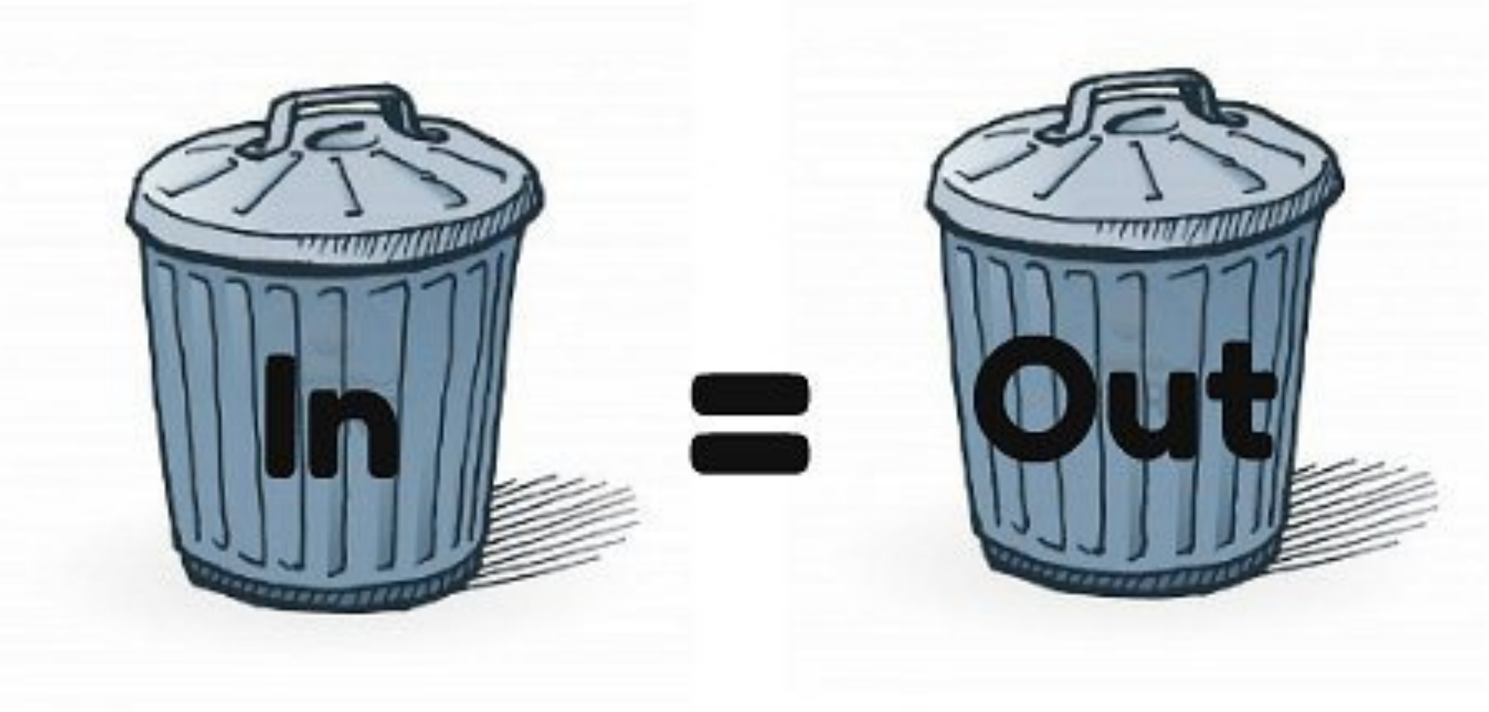


GPU

TPU

FPGA

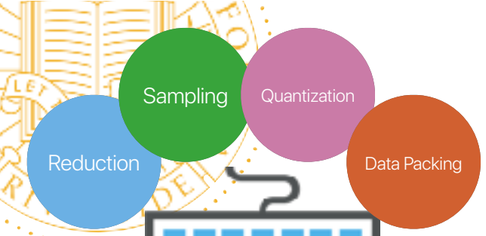
Garbage in, garbage out



Quality Control

Autofocus
iFilter

Autofocus



Raw Data

Lower-resolution

25% Resolution

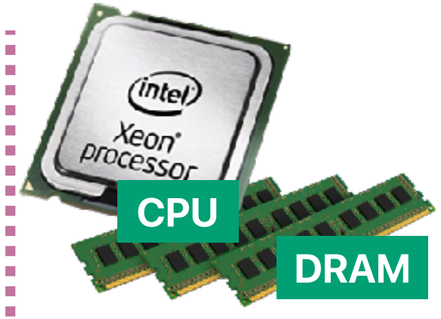
Reject if it becomes garbage

Better-resolution

50% Resolution

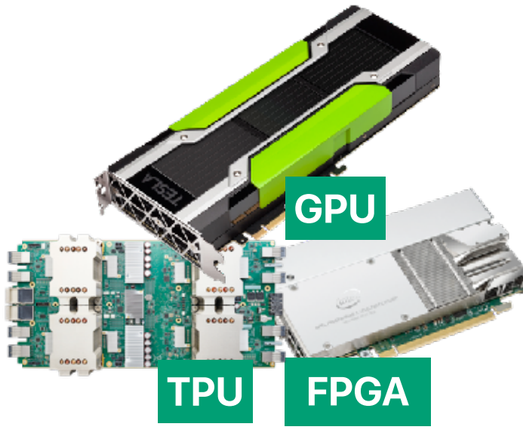
Use this resolution for the same dataset later

Varifocal Storage (VS)



CPU

DRAM



GPU

TPU

FPGA

Pass!

Compute

Test

Bear

System Interconnect (e.g., PCIe)

System Interconnect (e.g., PCIe)

ESCAL
Extreme Storage & Computer Architecture Lab

UNIVERSITY OF CALIFORNIA
UC RIVERSIDE

iFilter

Autofocus on Operator A

25% Resolution

Autofocus on Operator B

45% Resolution

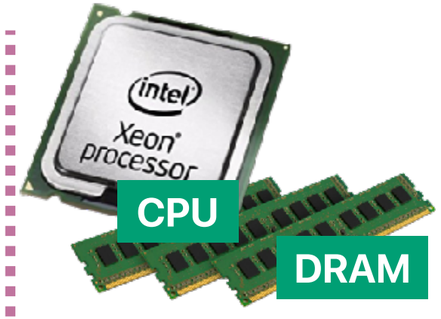
Autofocus on Operator C

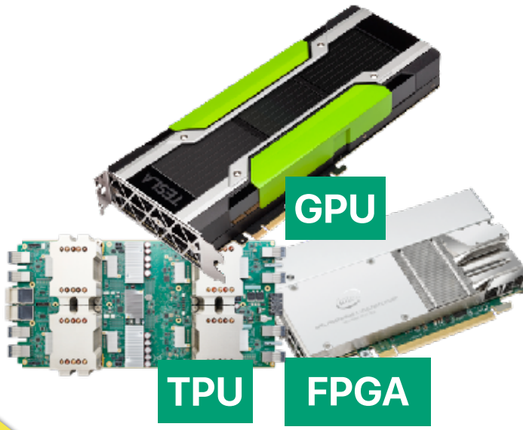
50% Resolution

Pass!

Use this resolution/ operator for the same dataset later

Pass!





Send Smaller One & Compute

Test

Bear

System Interconnect (e.g., PCIe)

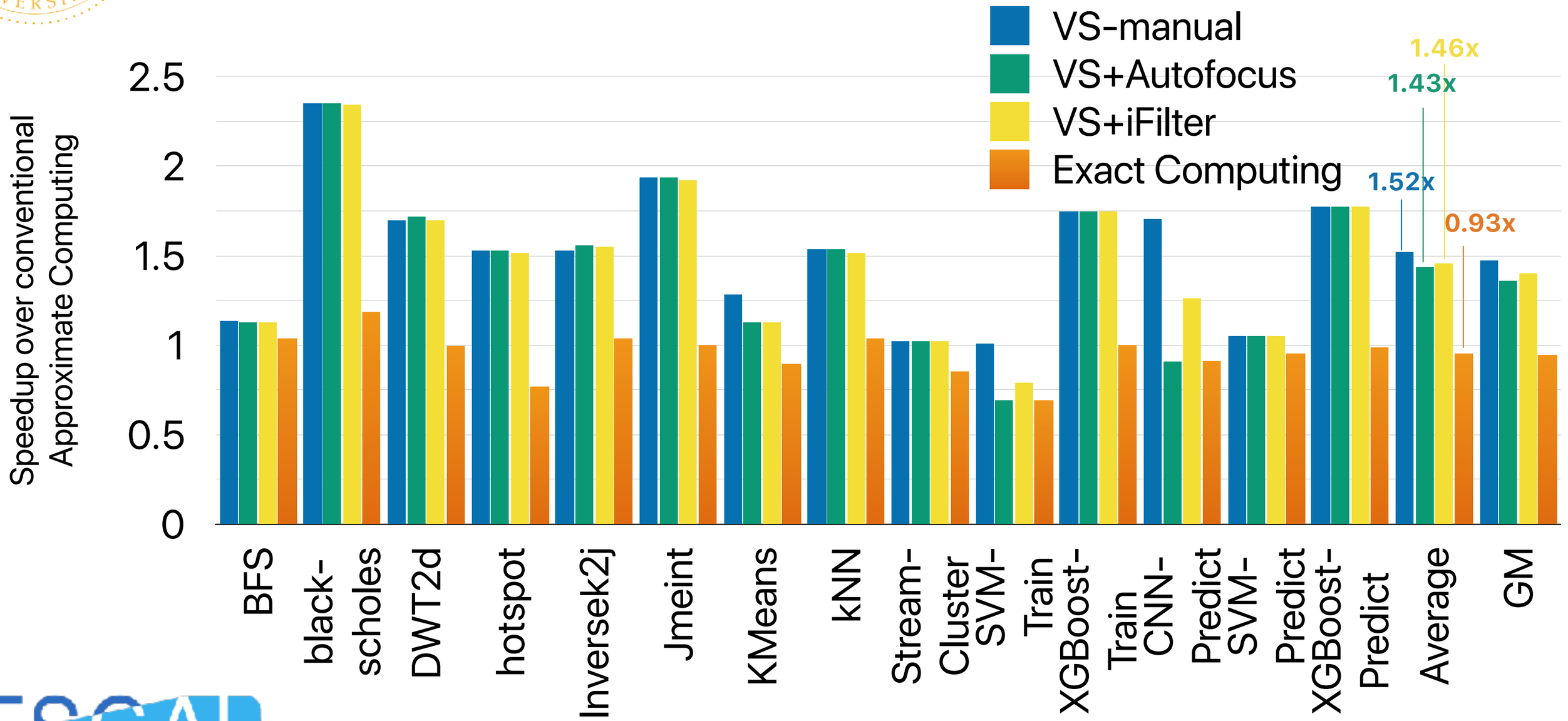
System Interconnect (e.g., PCIe)

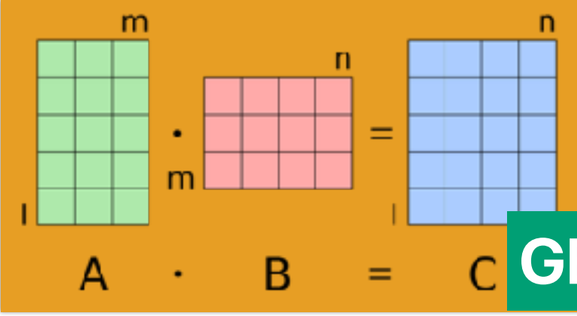
Varifocal Storage (VS)

ESCAL Extreme Storage & Computer Architecture Lab

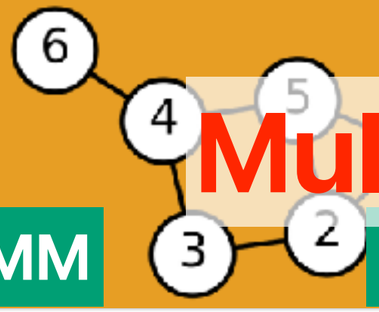
UNIVERSITY OF CALIFORNIA UCRIVERSIDE

Performance of Varifocal Storage (VS)

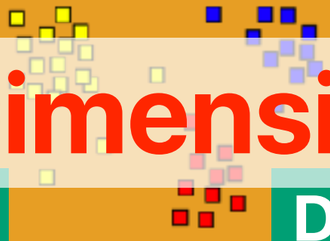




GEMM



Graph

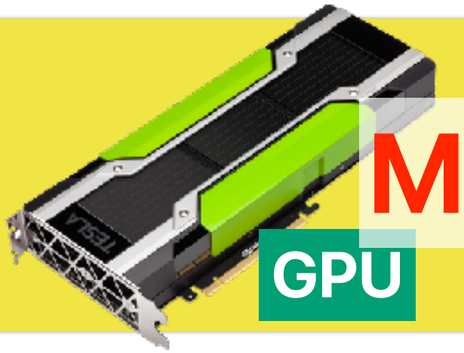


Data Analysis

Multi-Dimensional

Datasets/Compute Kernels/Applications

CPU



GPU

TCU

TPU

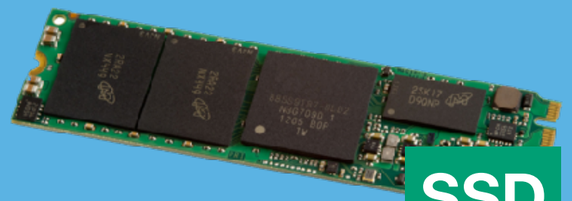
Multi-Dimensional

Compute Units

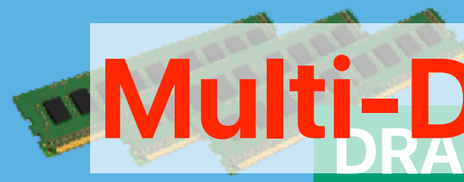
```
pread(fd, buf, 512, 0)
f.read(512)
f.read(&mut buffer[..])
```

One-dimension!!! (start LBA, length)

Storage Interface



SSD



DRAM

Multi-Dimensional

Storage Devices

Examples of Storage Interface

```c
int fd = open("data.bin", O_RDONLY);
float *buf = (float *) malloc(512);
pread(fd, buf, 512, 0);
```

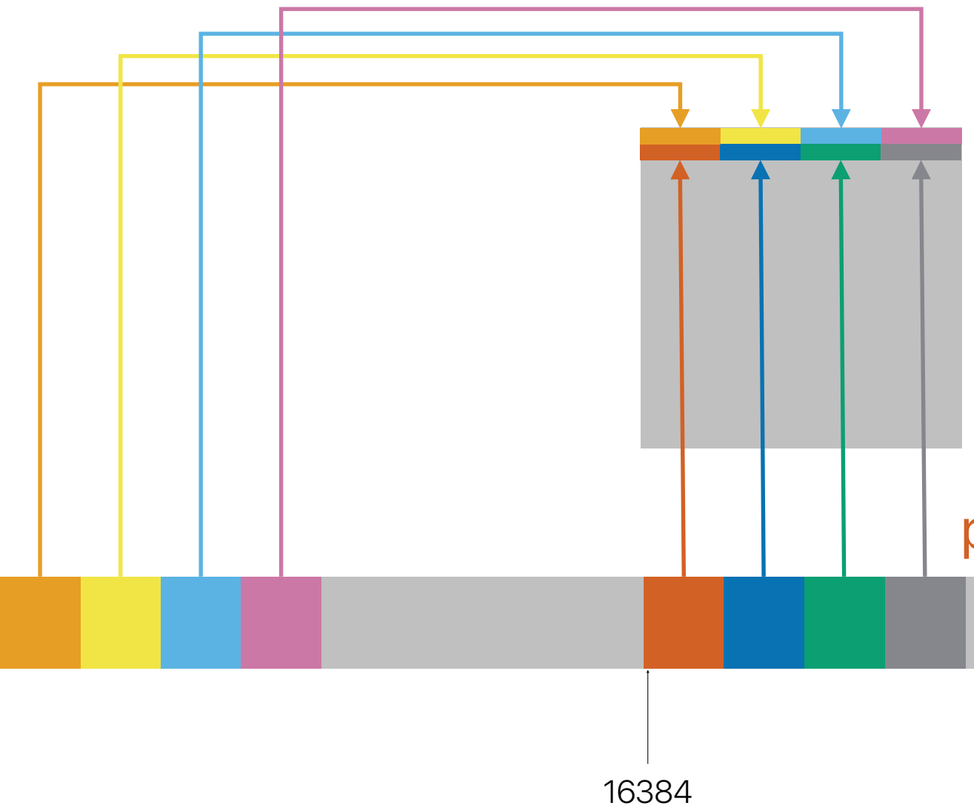

```python
f = open('matrix.bin')
f.read(512)
f.close()
```

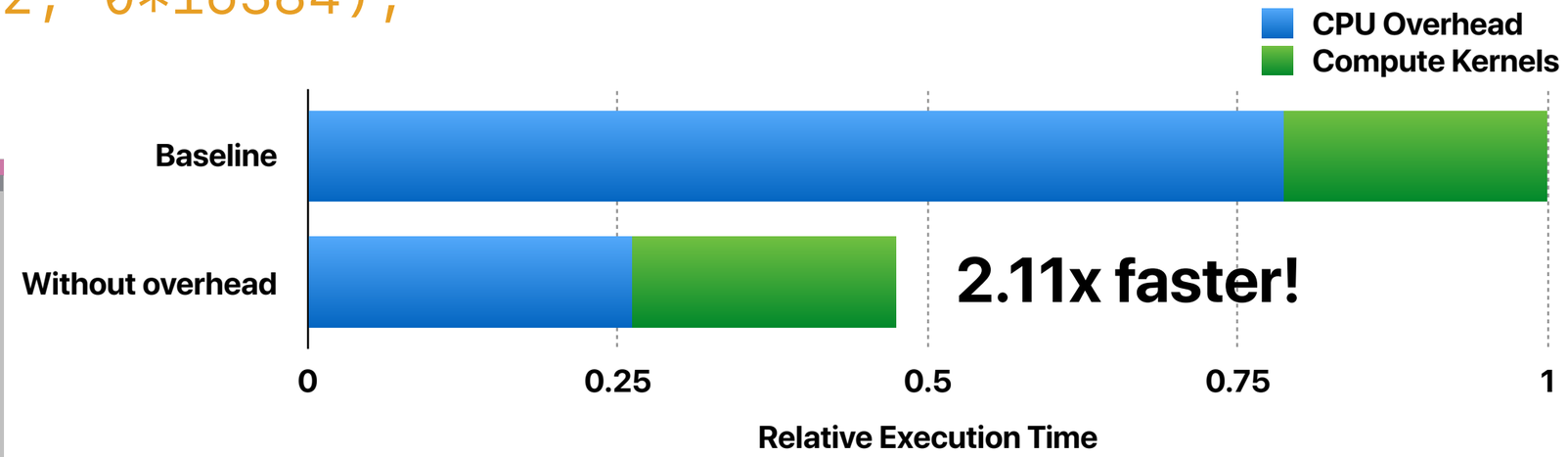

```rust
let mut f = File::open("matrix.bin")?;
let mut buffer = [0; 10];
let n = f.read(&mut buffer[..])?;
```

opcode	flags	cmd		namespace			
reserved							
metadata							
PRP1							
PRP2							
Start LBA							
length		control		Dataset management			
Reference tag				App tag		App mask	

Problem 1: the overhead of marshalling data

```
pread(fd, buf+0*8192, 8192, 0*16384);
```



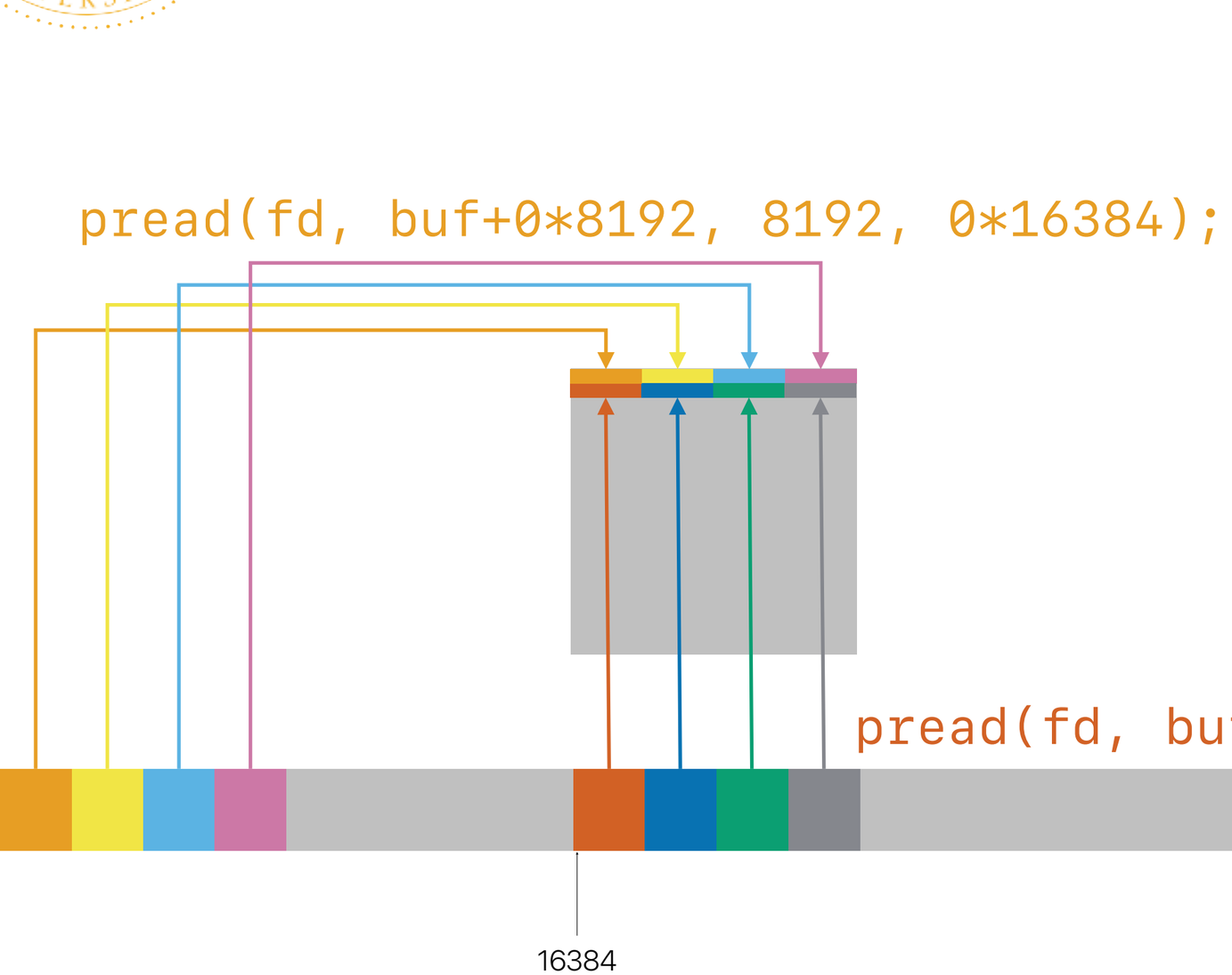


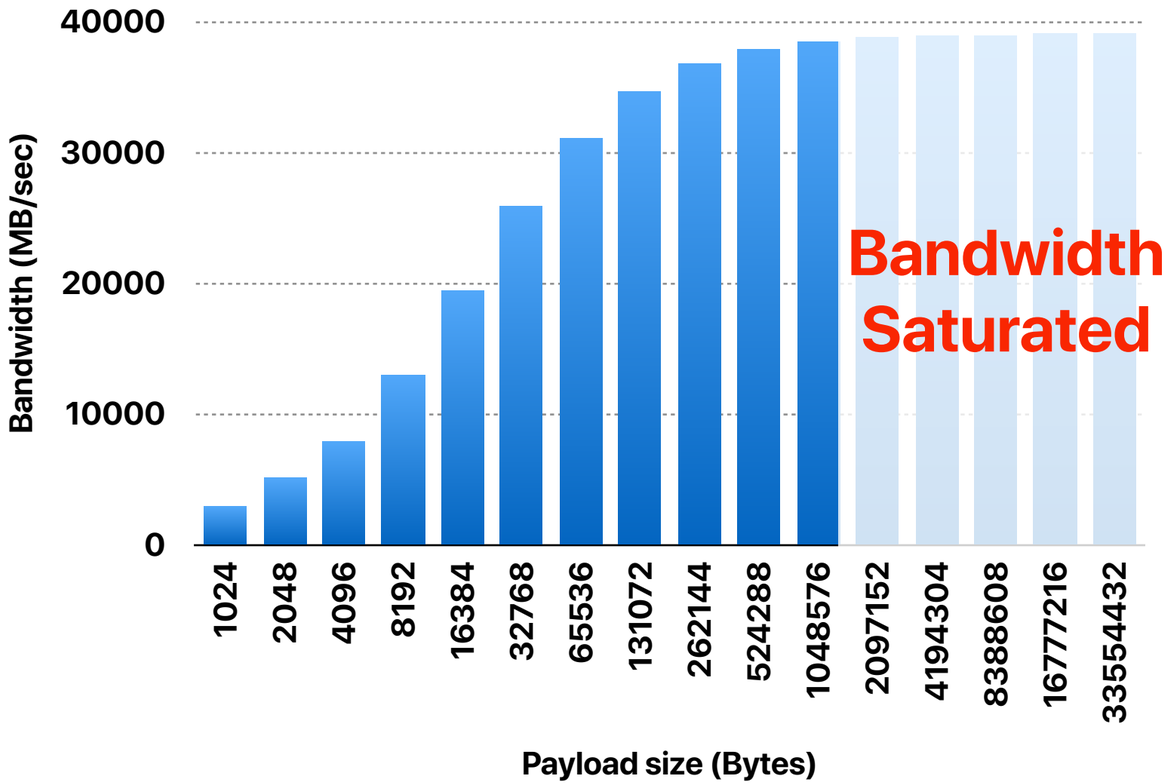

```
pread(fd, buf+1*8192, 8192, 1*16384);
```

16384

address space

Problem 2: under-utilized interconnect bandwidth





Problem 3: under-utilized internal bandwidth

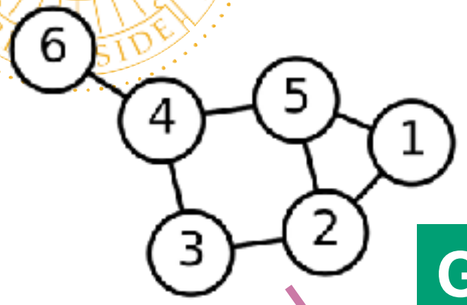
0th row, [0, 2047]	0th row, [2048, 4095]	0th row, [4096, 6143]	0th row, [6144, 8191]	0th row, [8192, 10239]	0th row, [10240, 12287]	0th row, [12288, 14335]	0th row, [14336, 16383]
1st row, [0, 2047]	1st row, [2048, 4095]	1st row, [4096, 6143]	1st row, [6144, 8191]	1st row, [8192, 10239]	1st row, [10240, 12287]	1st row, [12288, 14335]	1st row, [14336, 16383]
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
16383rd row, [0, 2047]	16383rd row, [2048, 4095]	16383rd row, [4096, 6143]	16383rd row, [6144, 8191]	16383rd row, [8192, 10239]	16383rd row, [10240, 12287]	16383rd row, [12288, 14335]	16383rd row, [14336, 16383]
Channel #0	Channel #1	Channel #2	Channel #3	Channel #4	Channel #5	Channel #6	Channel #7

These channels are idle

Example — NDS✳

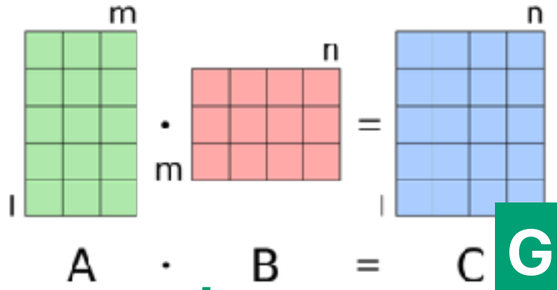
✳ Yu-Chia Liu and Hung-Wei Tseng. 2021. NDS: N-Dimensional Storage. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21)

NDS: N-Dimensional Storage



Graph

read/write	
coordinate	1, 0
dimensionality	M, N

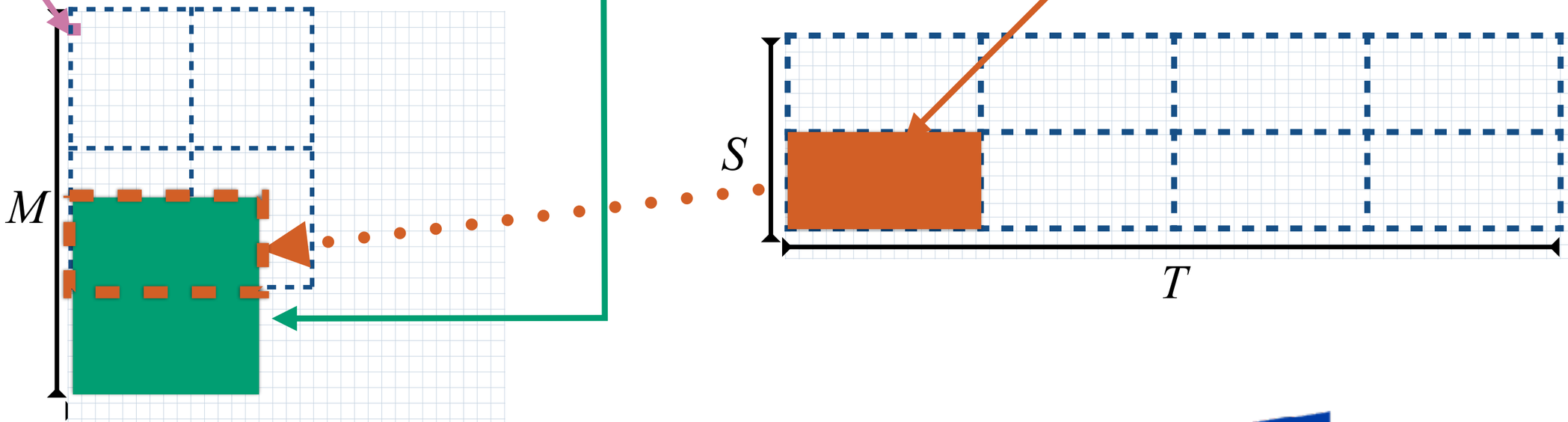


GEMM

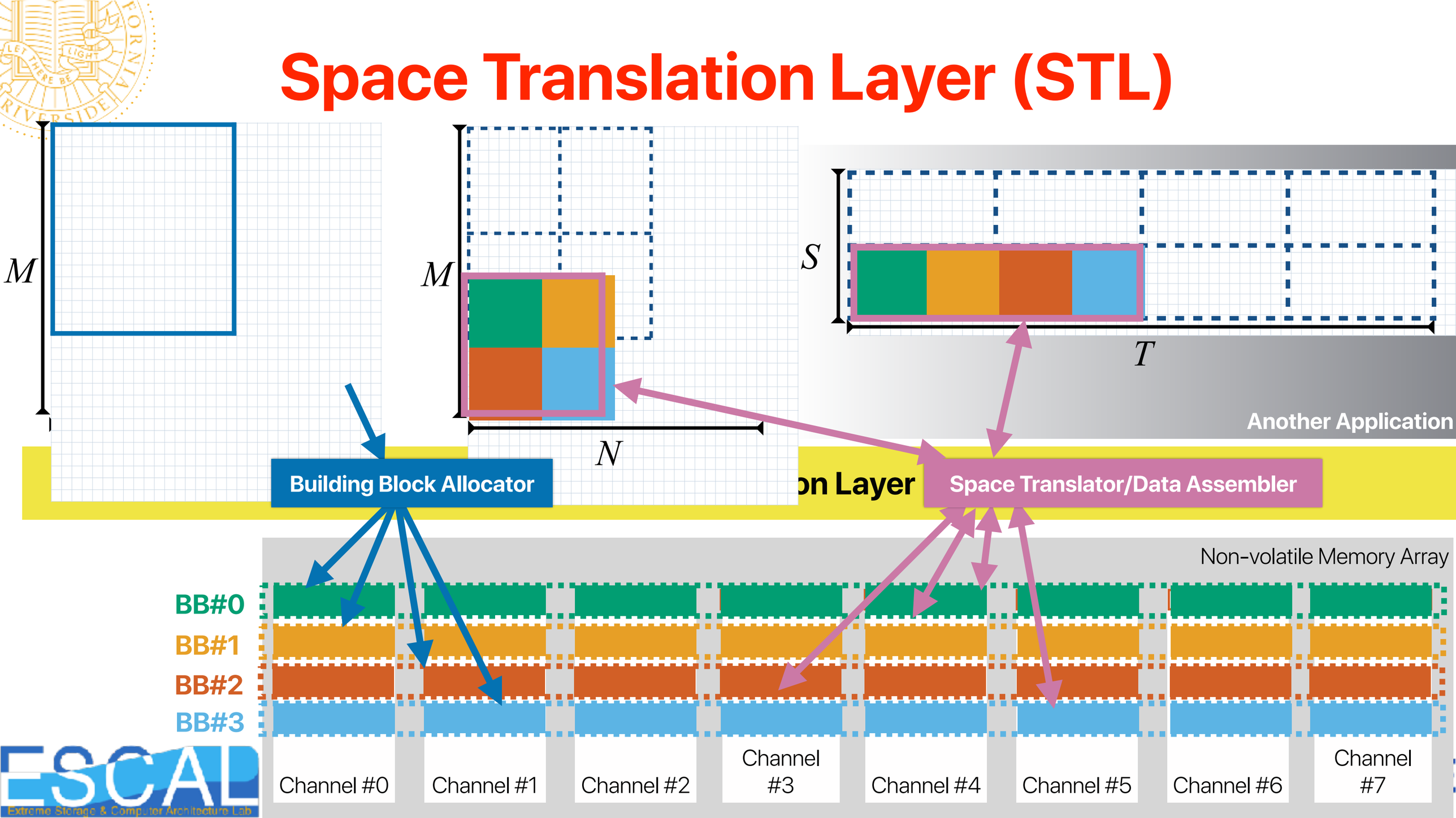
read/write	
coordinate	1, 0
sub-dimensionality	M/2, N/2
dimensionality	M, N

?

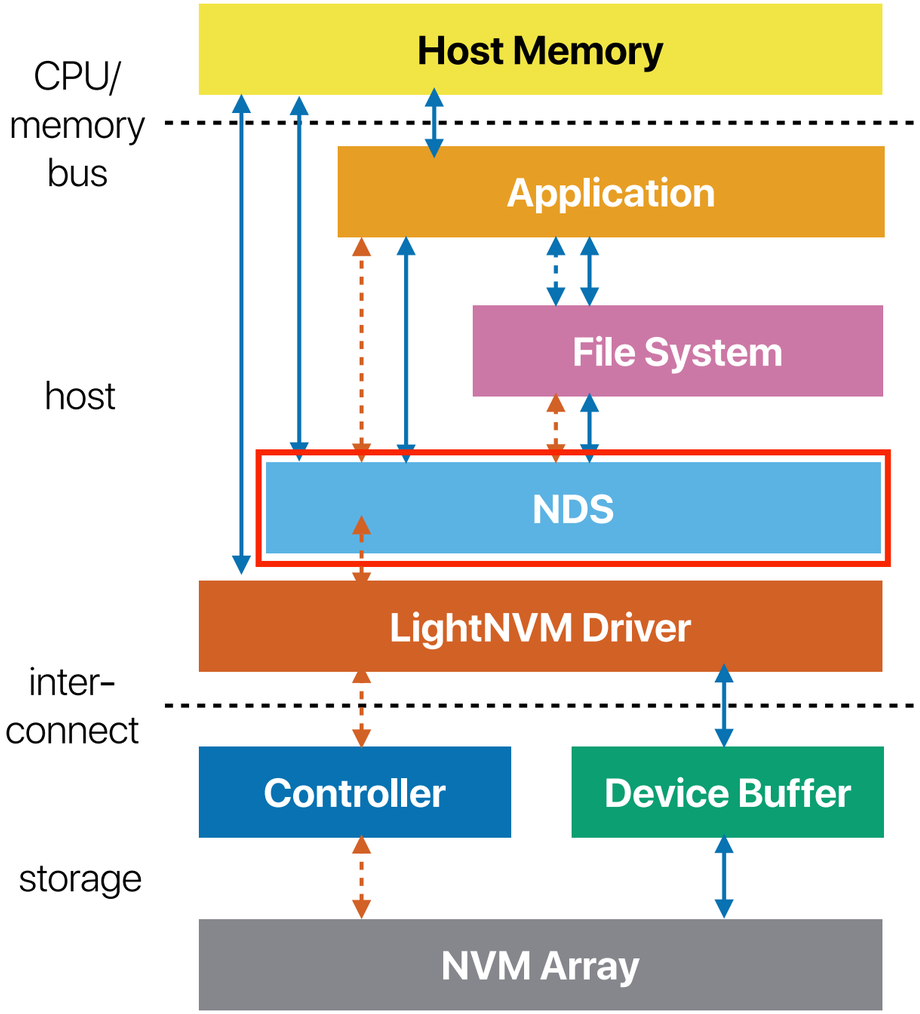
read/write	
coordinate	1, 0
sub-dimensionality	S/2, T/4
dimensionality	S, T



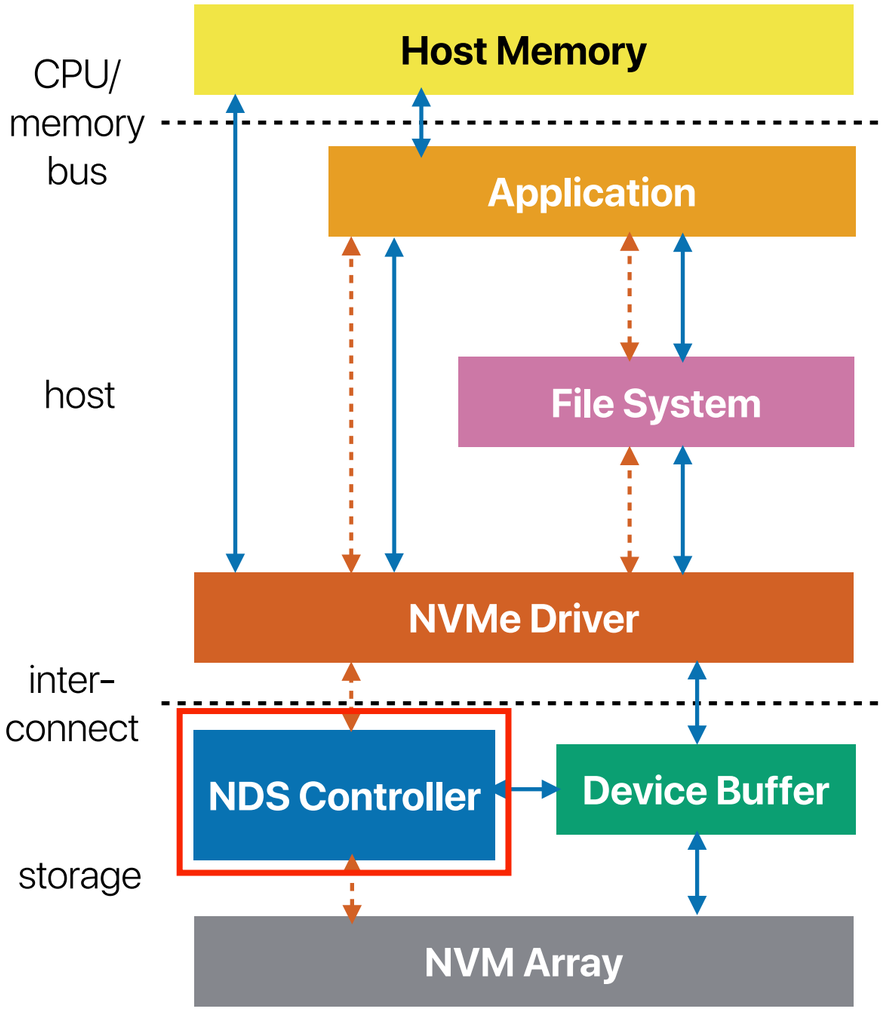
Space Translation Layer (STL)



NDS Implementations

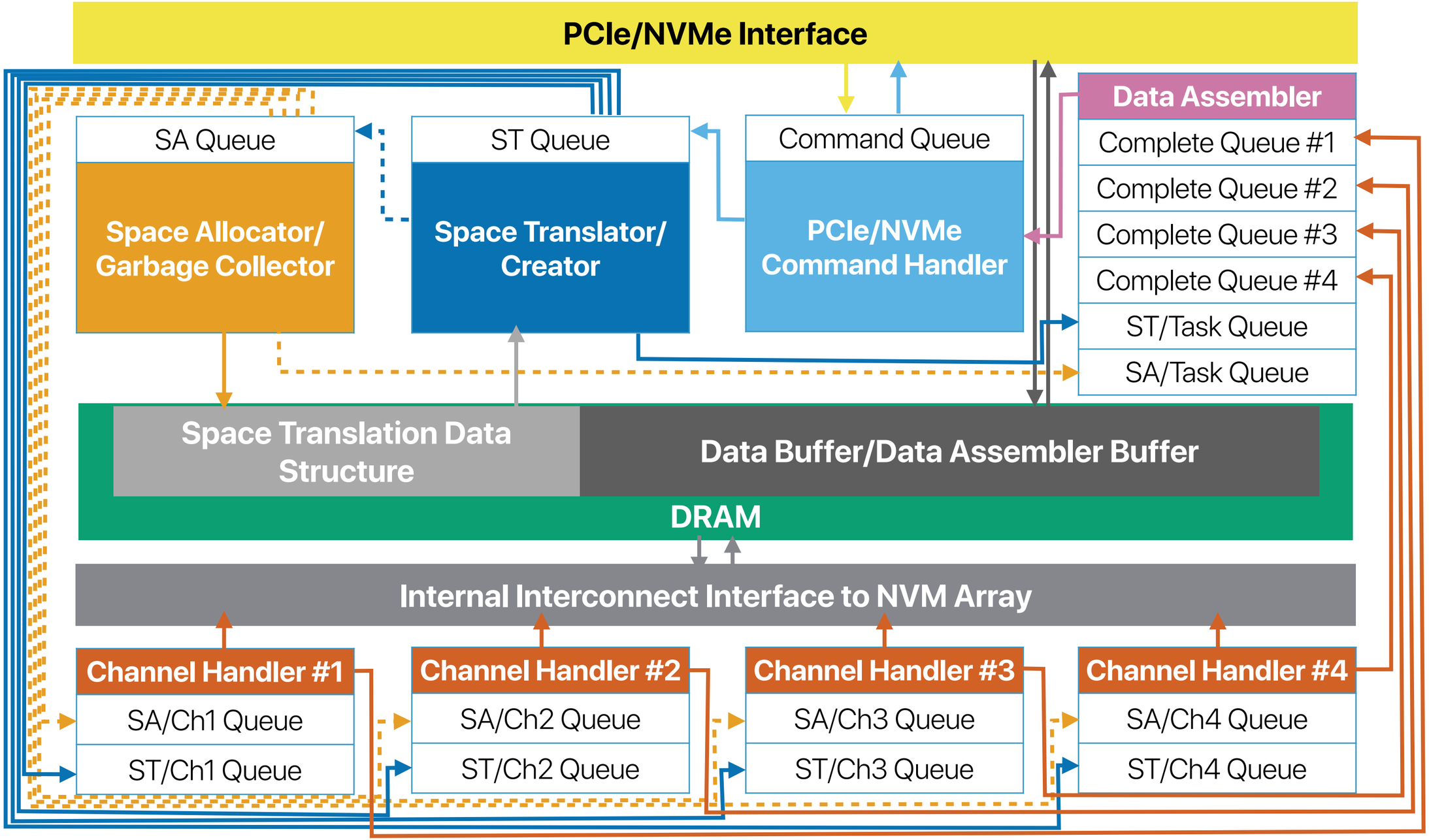


Software NDS

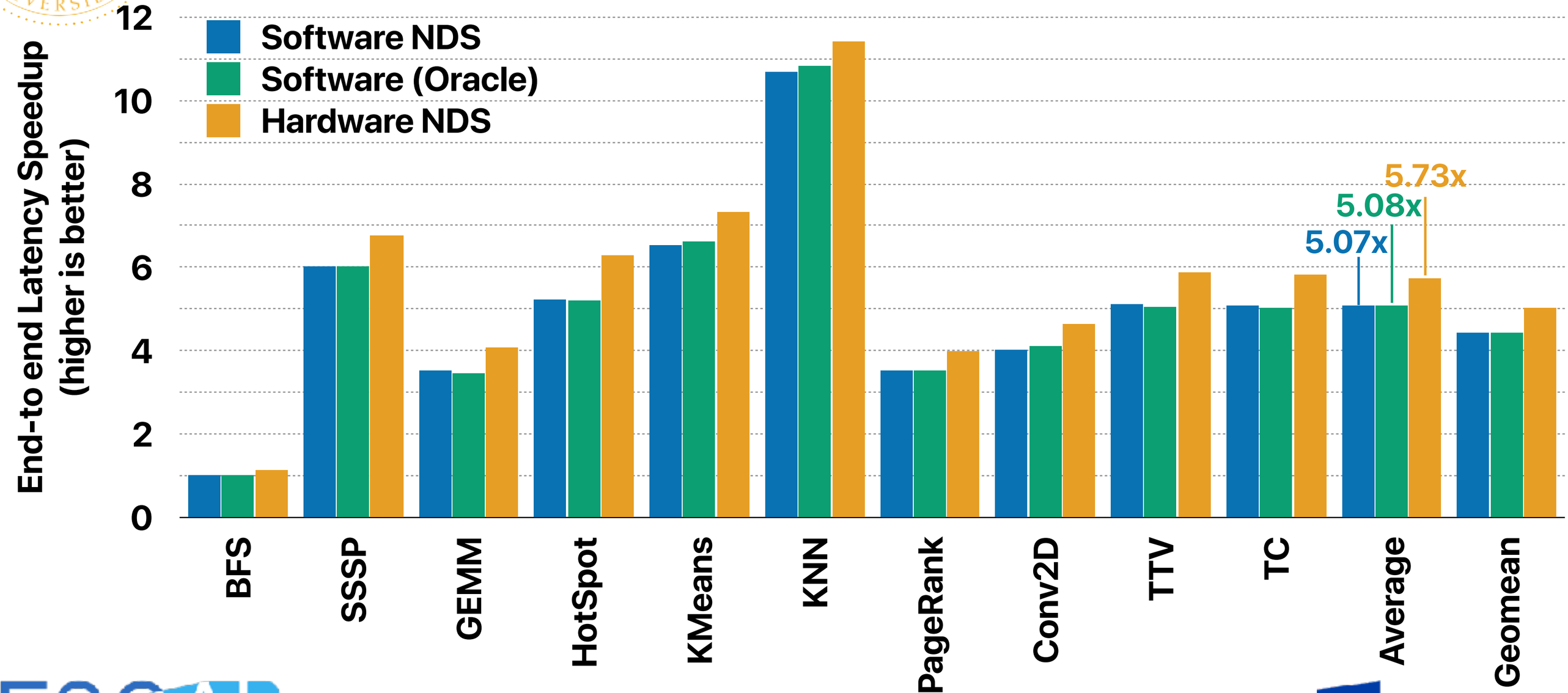


Hardware NDS

control path
data path


PCIe/NVMe Interface
SA Queue
ST Queue
Command Queue
Data Assembler
Complete Queue #1
Complete Queue #2
Complete Queue #3
Complete Queue #4
ST/Task Queue
SA/Task Queue
Space Allocator/ Garbage Collector
Space Translator/ Creator
PCIe/NVMe Command Handler
Space Translation Data Structure
Data Buffer/Data Assembler Buffer
DRAM
Internal Interconnect Interface to NVM Array
Channel Handler #1
Channel Handler #2
Channel Handler #3
Channel Handler #4
SA/Ch1 Queue
ST/Ch1 Queue
SA/Ch2 Queue
ST/Ch2 Queue
SA/Ch3 Queue
ST/Ch3 Queue
SA/Ch4 Queue
ST/Ch4 Queue

Performance of NDS



Electrical
Computer Science
Engineering

277

つづく