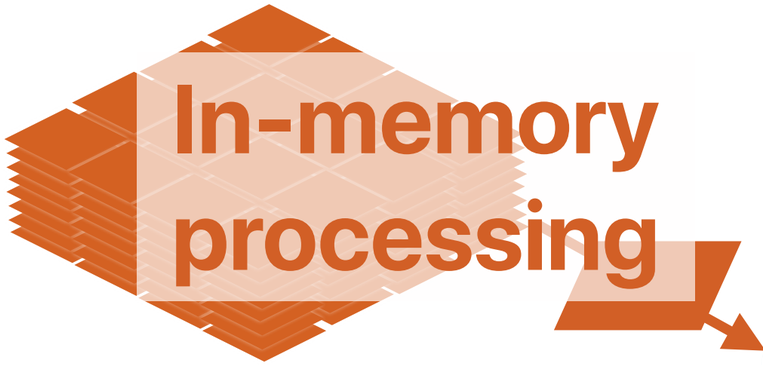
Cross the boundary

Hung-Wei Tseng

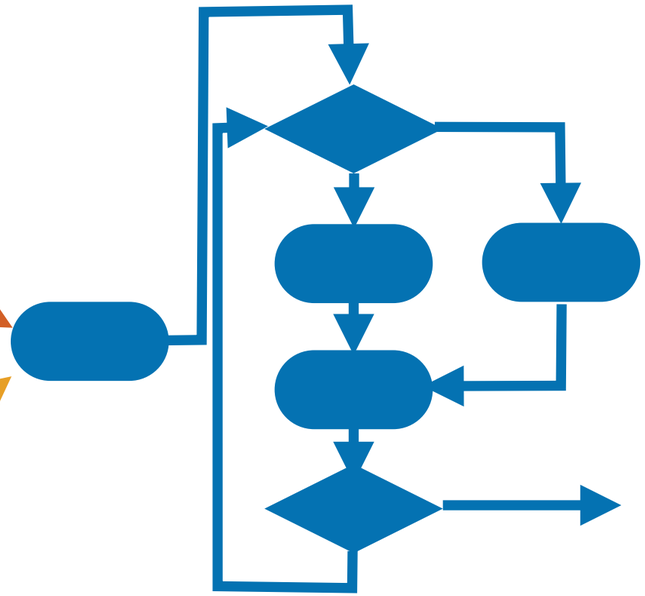
Recap: From the software perspective

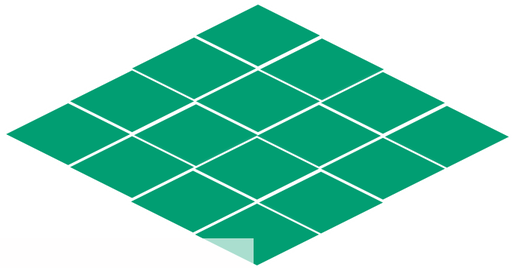
In-Storage Processing

In-memory processing



How about this process?

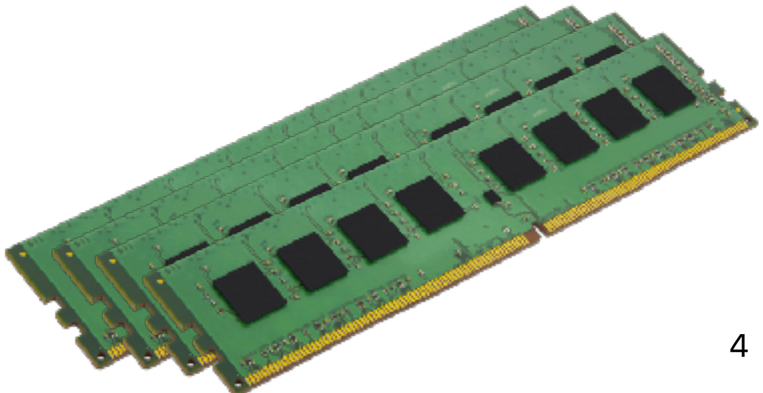




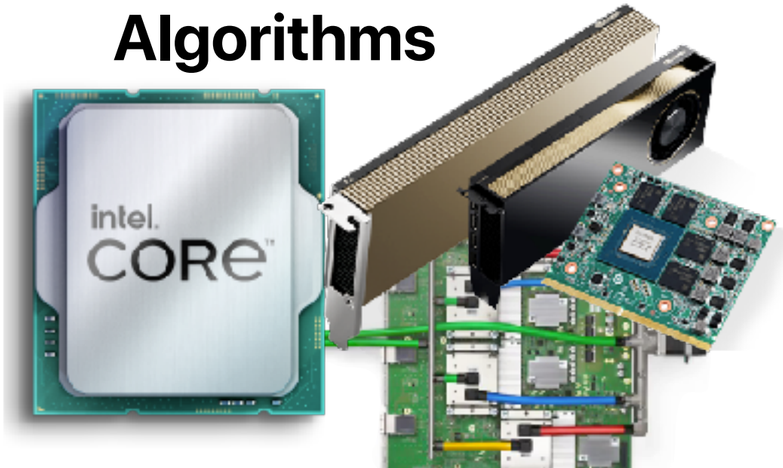
Hardware Accelerators

Source “data"

“Data” structures

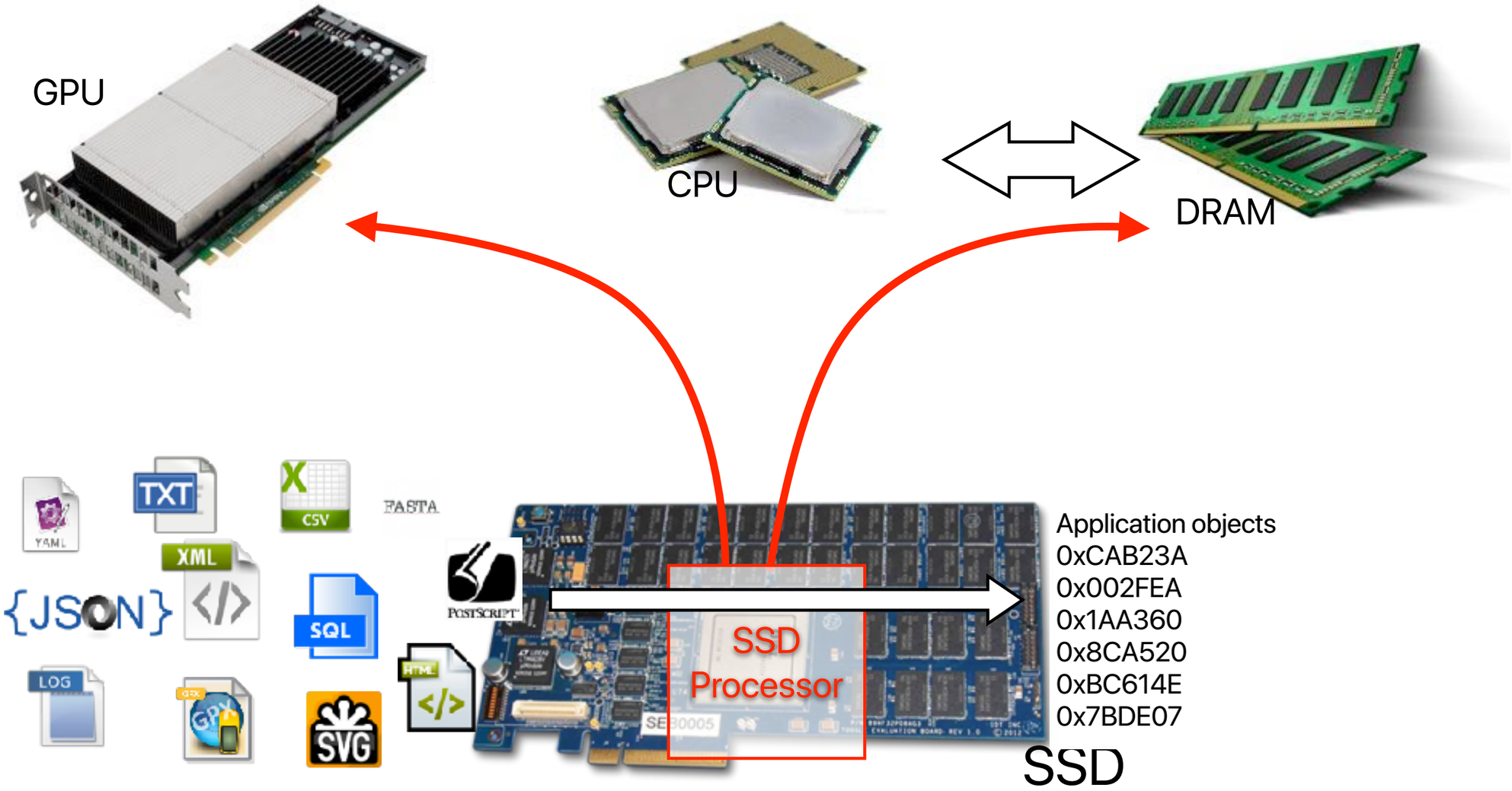


Algorithms

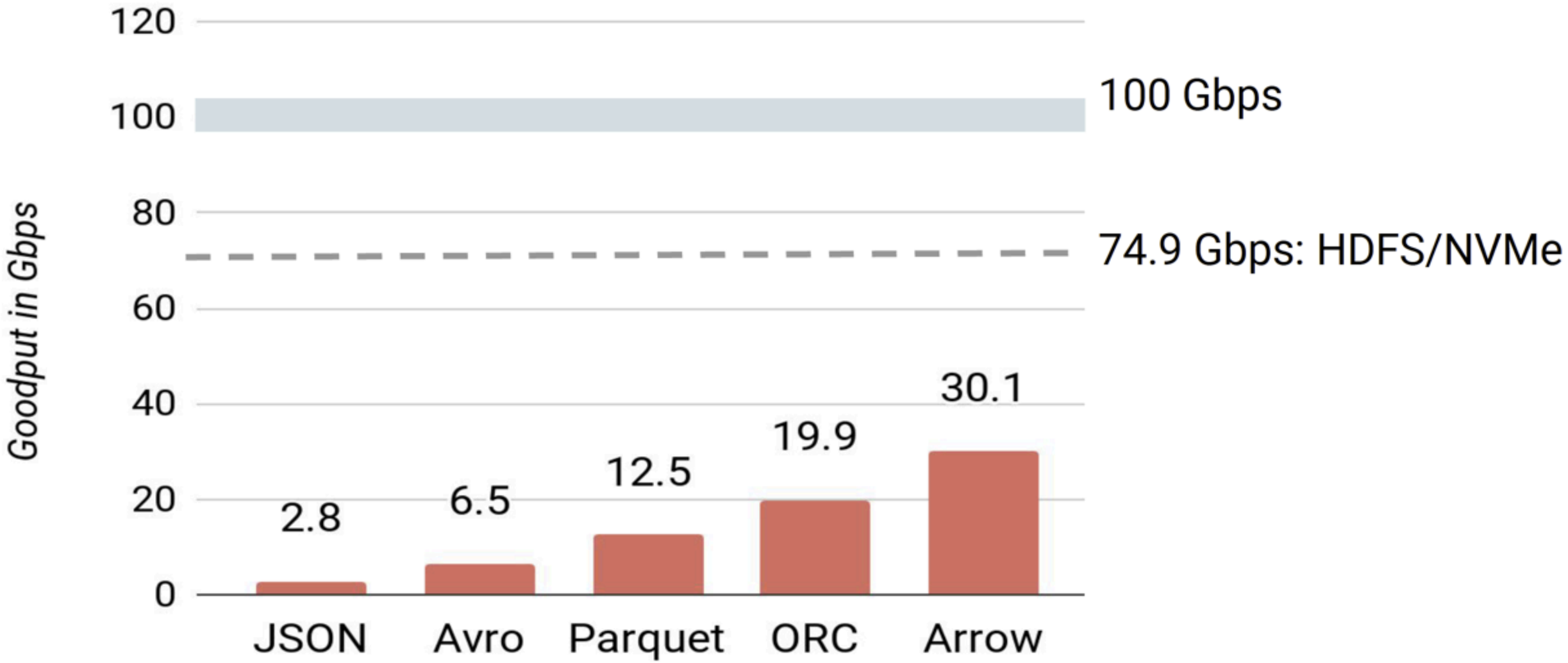


Result

Morpheus

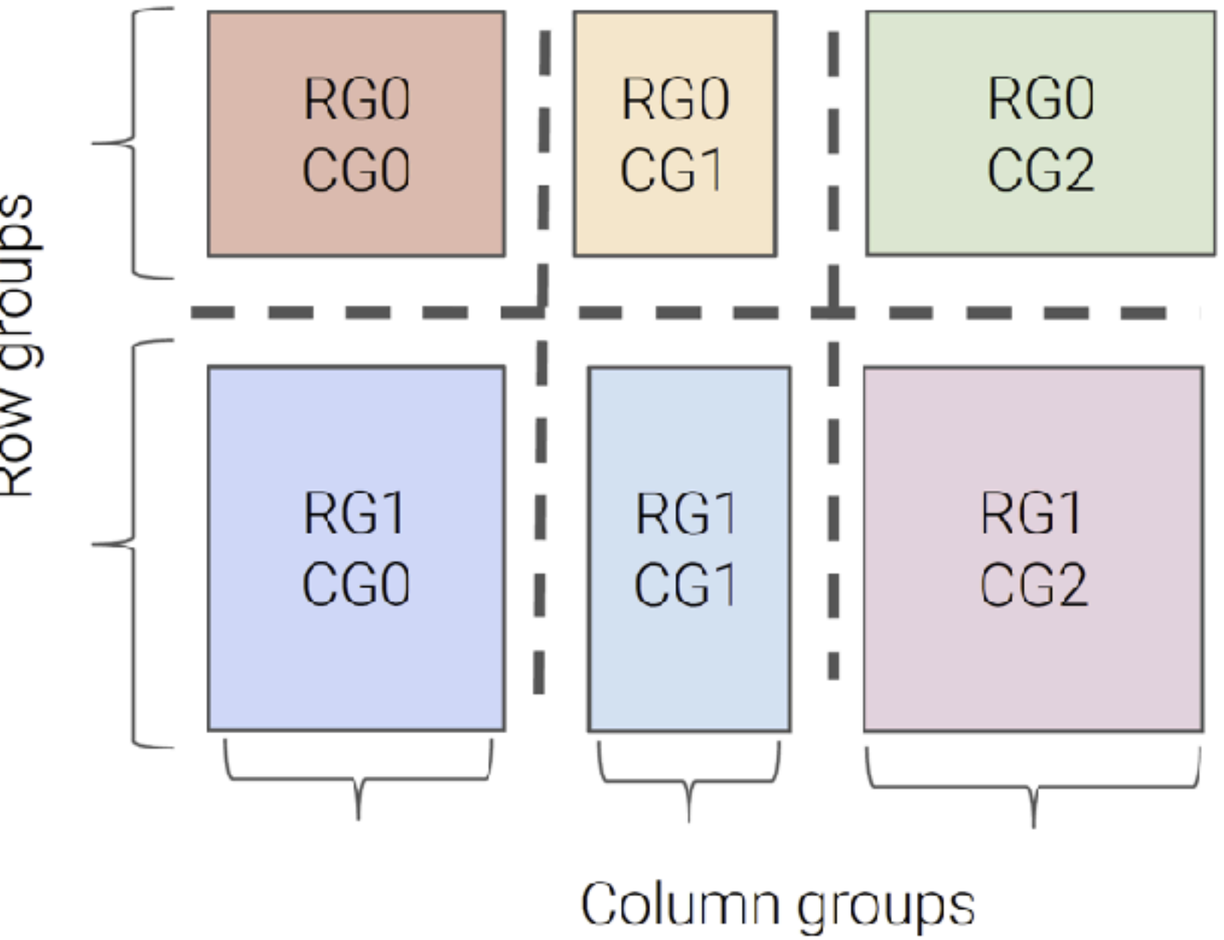

GPU
CPU
DRAM
YAML
TXT
CSV
FASTA
XML
{JSON}
SQL
LOG
GPX
SVG
HTML
POSTSCRIPT
SSD
Processor
Application objects
0xCAB23A
0x002FEA
0x1AA360
0x8CA520
0xBC614E
0x7BDE07
SSD

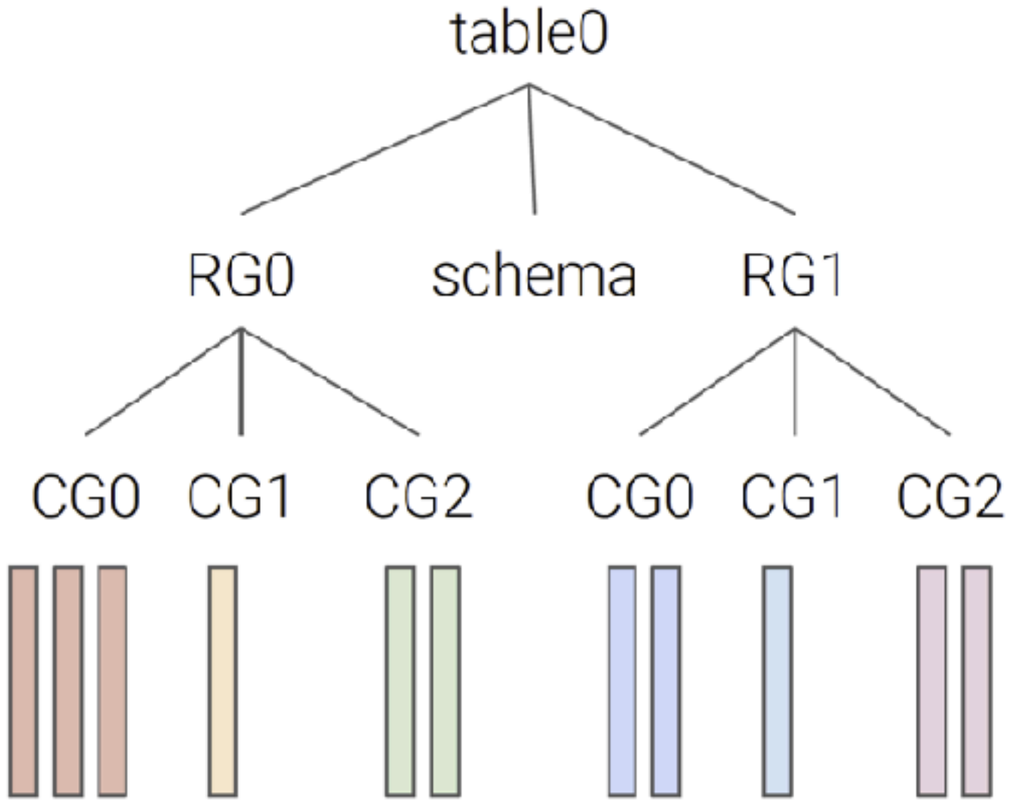
The deserialization performance on popular formats



https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_trivedi.pdf

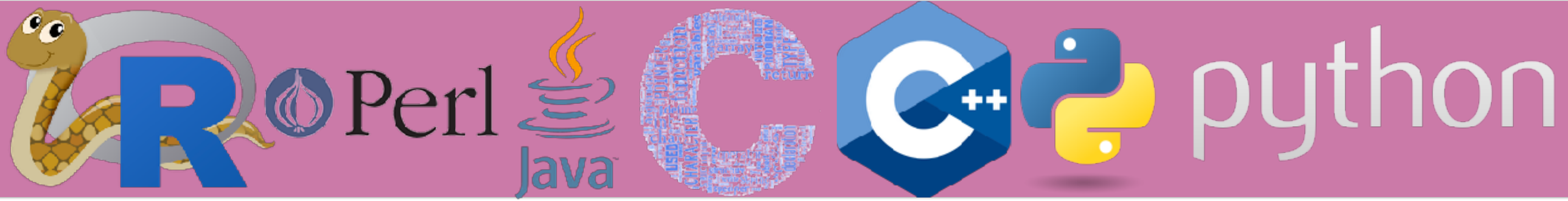
Albis: Not just row groups, but also column groups

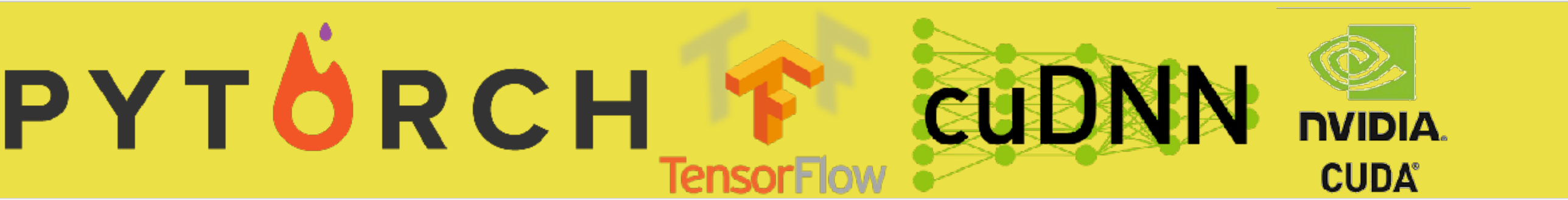




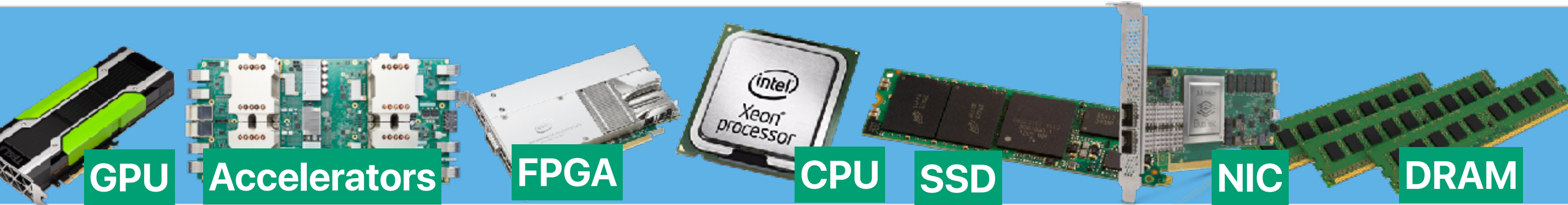
https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_trivedi.pdf


DO NOT CROSS


R
Perl
Java
C
C++
python


PYTORCH
TensorFlow
cuDNN
NVIDIA
CUDA


Mac OS
Linux
Windows
android


GPU
Accelerators
FPGA
intel
Xeon processor
CPU
SSD
NIC
DRAM

3:00

Is layering design a good approach? Why or why not?

Outline

- Pros and cons of layering
- Cross-layer design example
- Pros and cons of approximate computing
- Cross-layer design for approximate computing

Pros/Cons of Layering

Pros/Cons of Layering

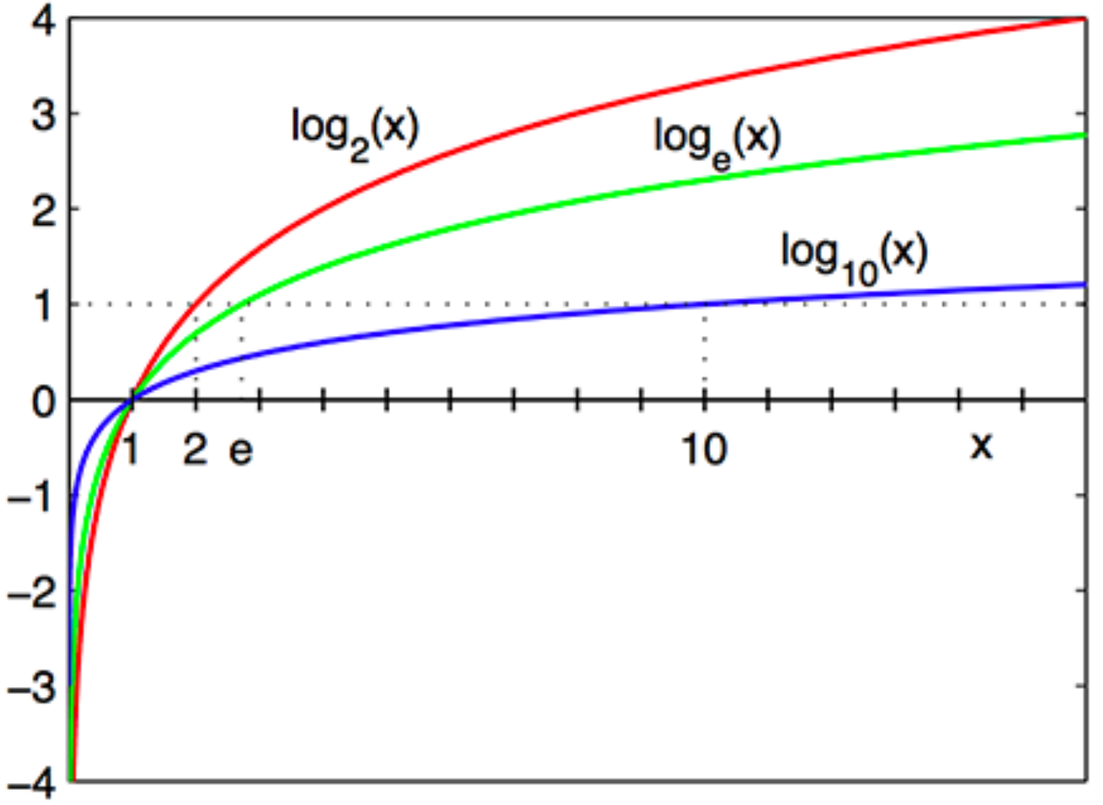
- Good
  - Ease of Debugging (T.H.E. by Dijkstra)
- Bad
  - Performance
  - Sub-optimal optimizations


R
Perl
Java
C
C++
python
PYTORCH
TensorFlow
cuDNN
NVIDIA
CUDA
Mac OS
Linux
Windows
ANDROID
? ? ? ?
GPU
Accelerators
FPGA
intel
Xeon processor
CPU
SSD
NIC
DRAM

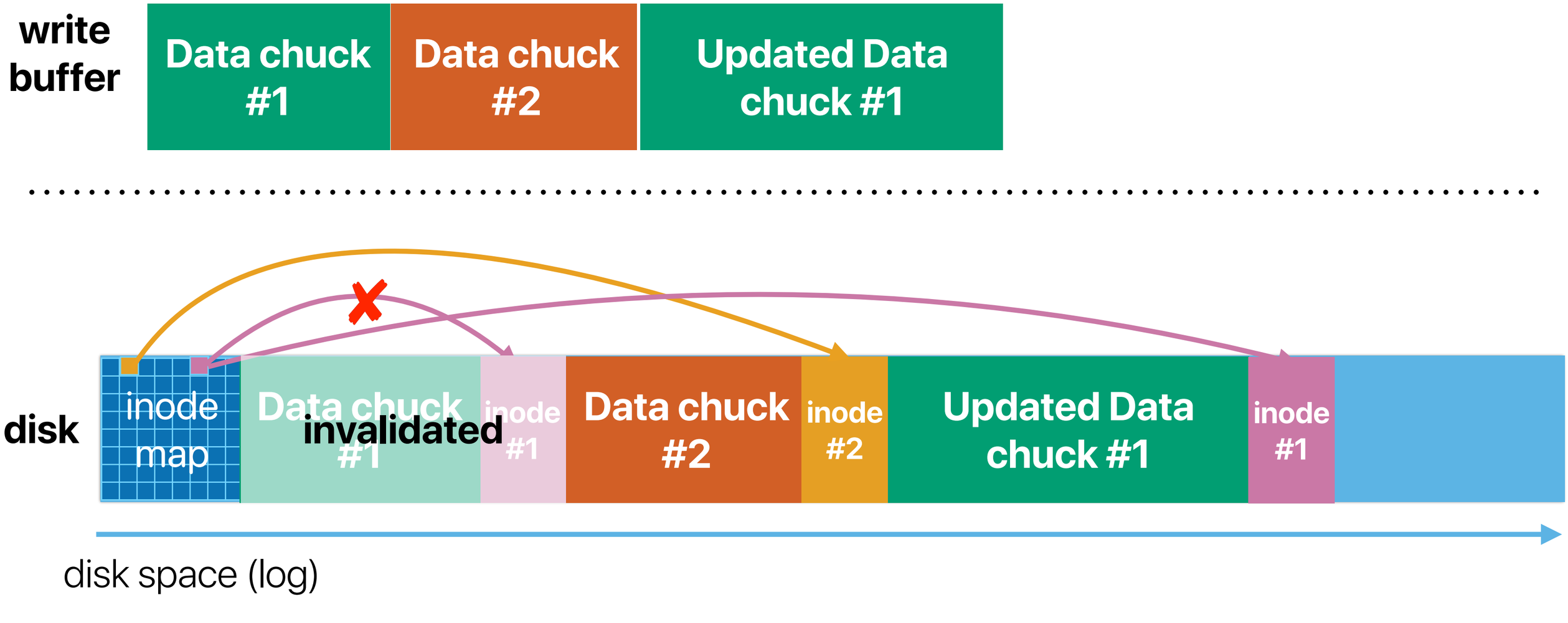
? ? ? ?

Don’t stack your log on my log

Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman
SanDisk Corporation


4
3
2
1
0
−1
−2
−3
−4
log2(x)
loge(x)
log10(x)
1
2
e
10
x

LFS in motion



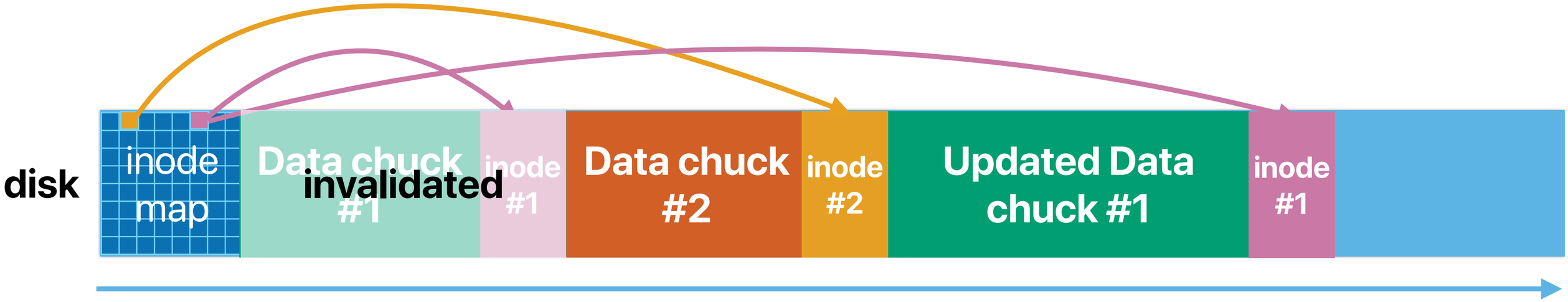
3:00

What's the good things and bad things about "Logs"

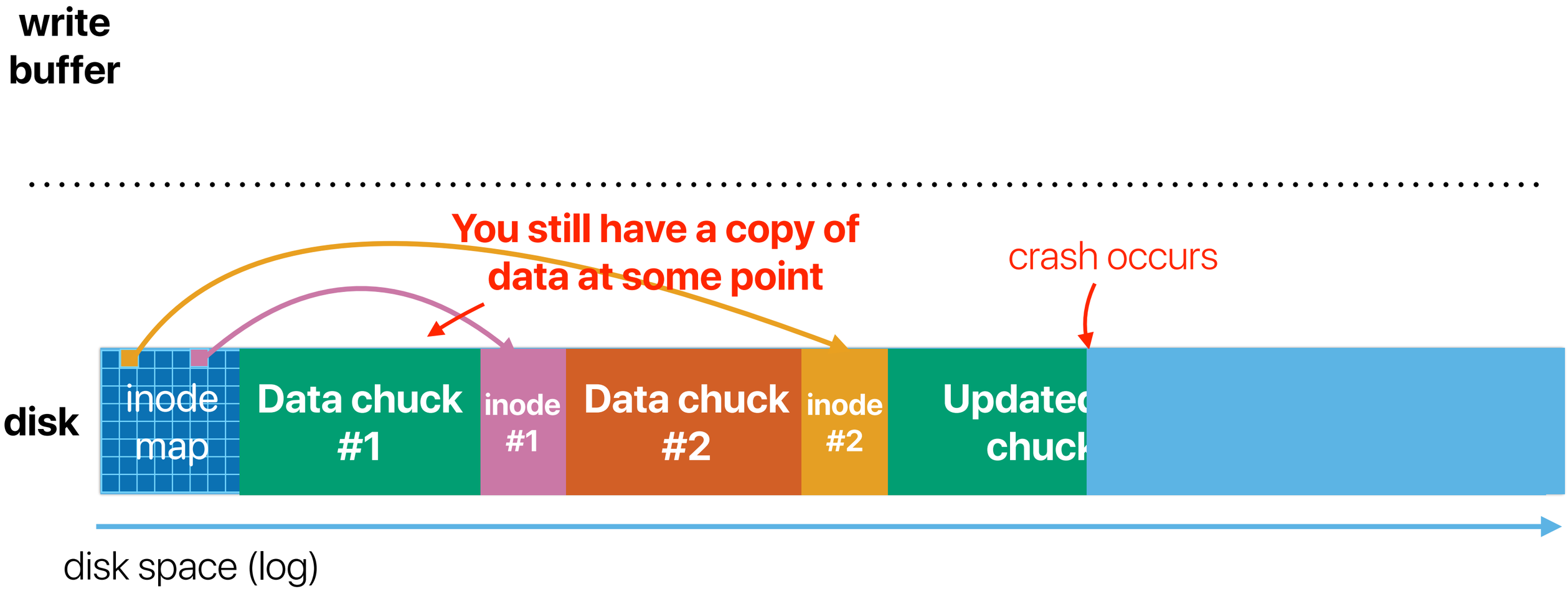
Pros/Cons of Logs

Log

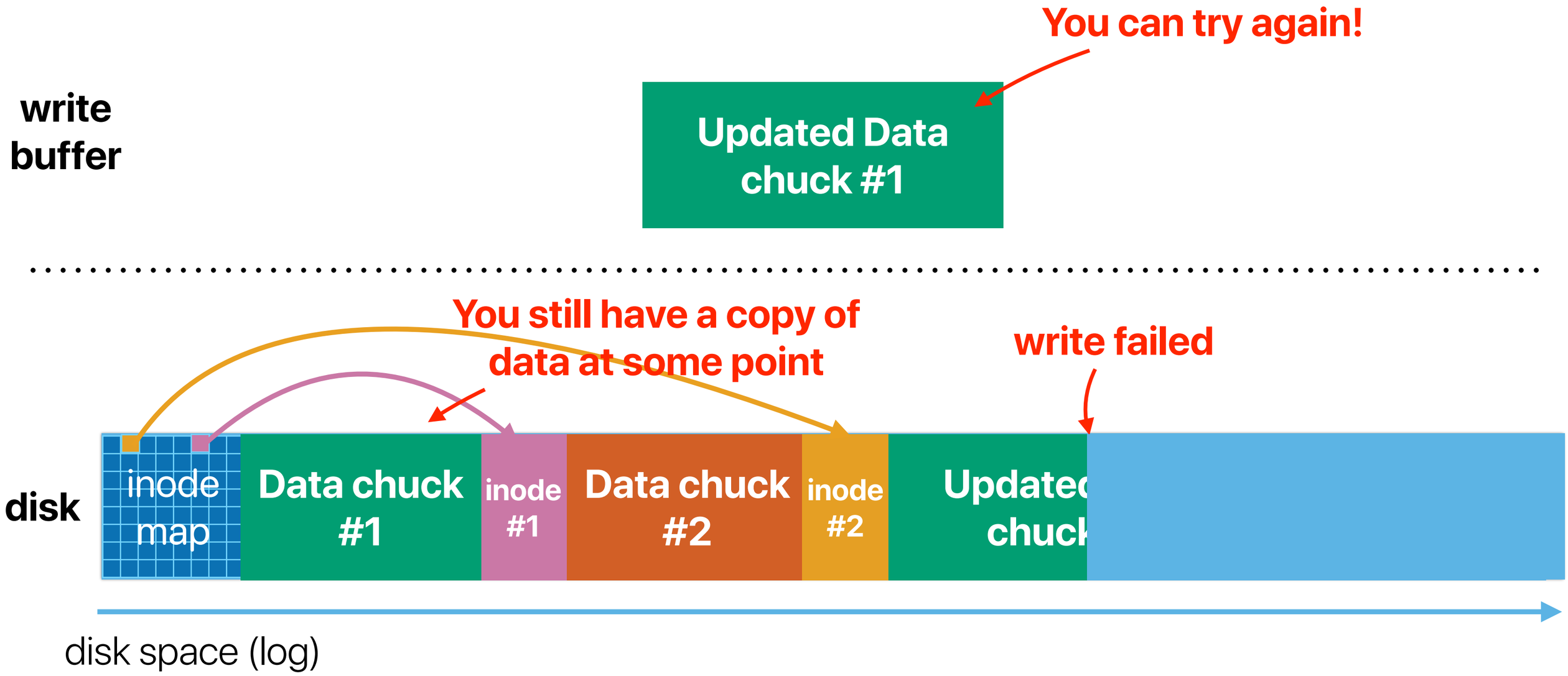
- An append only data structure that records all changes
- Advantages of logs
  - Better performance — always sequential accesses
  - Faster writes — you just need to append without sanitize existing data first
  - Ease of recovery — you can find previous changes within the log
- Disadvantage of logs — you will need to explicit perform garbage collections to reclaim spaces



LFS v.s. crash

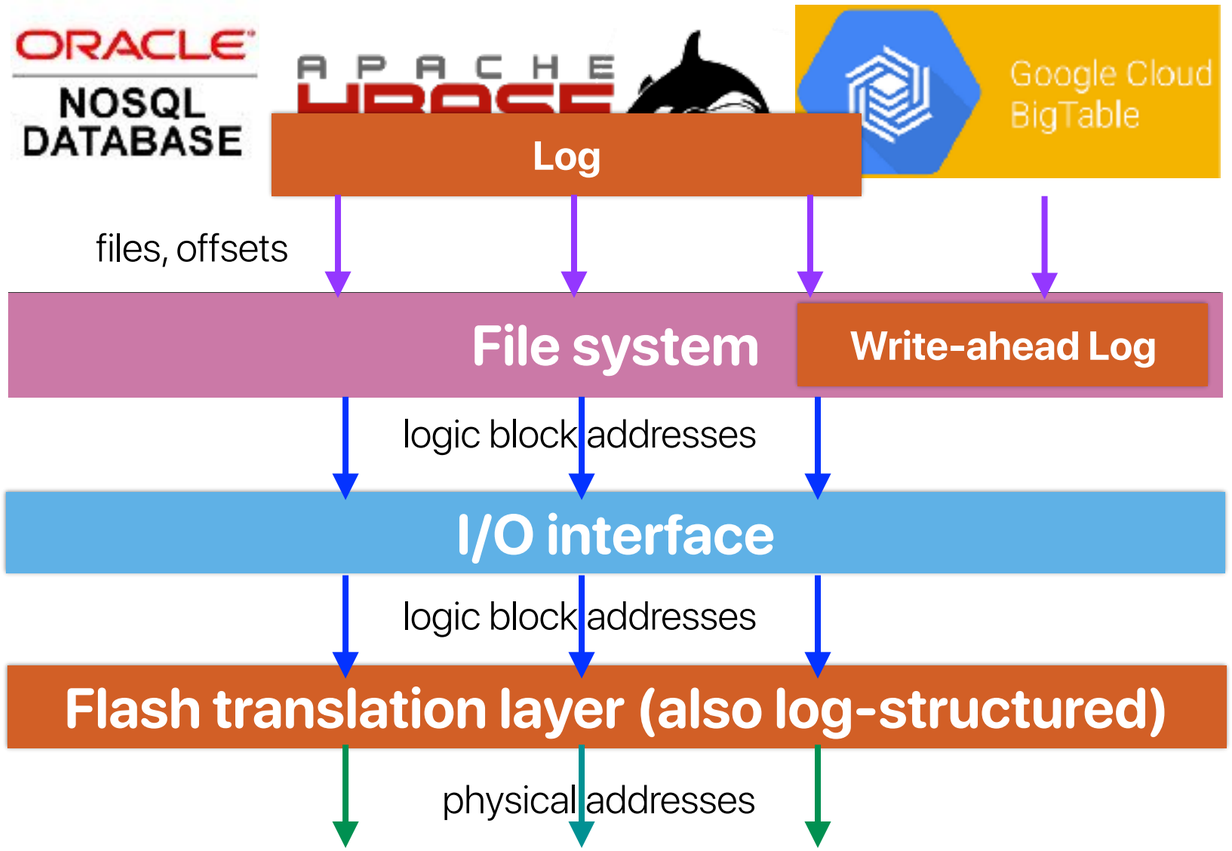

write buffer
You still have a copy of data at some point
crash occurs
disk
inode map
Data chuck #1
inode #1
Data chuck #2
inode #2
Updated chuck
disk space (log)

LFS v.s. write failed

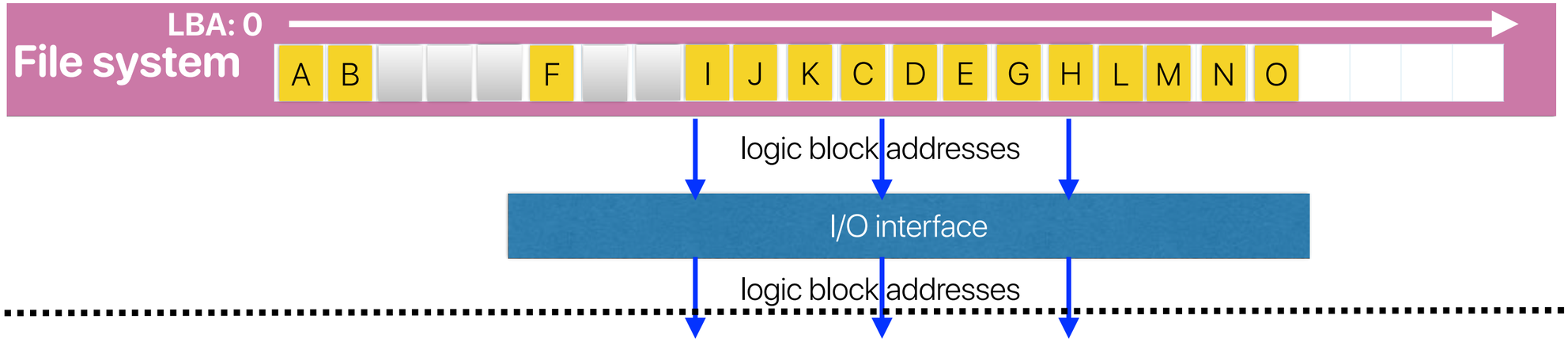


Why should we care about this paper?

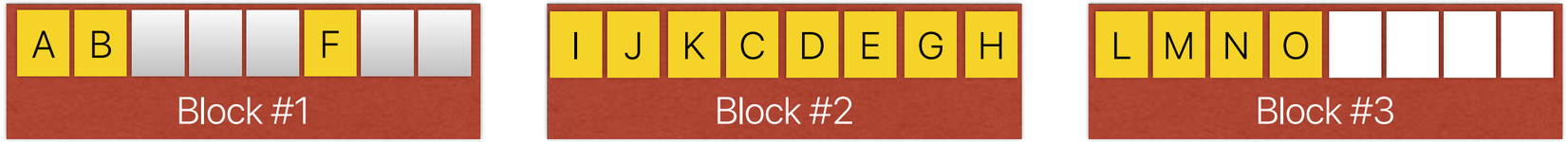
- Log is everywhere
  - Application: database
  - File system
  - Flash-based SSDs
- They can interfere with each other!
- An issue with software engineering nowadays



For example, garbage collection



SSD



invalid

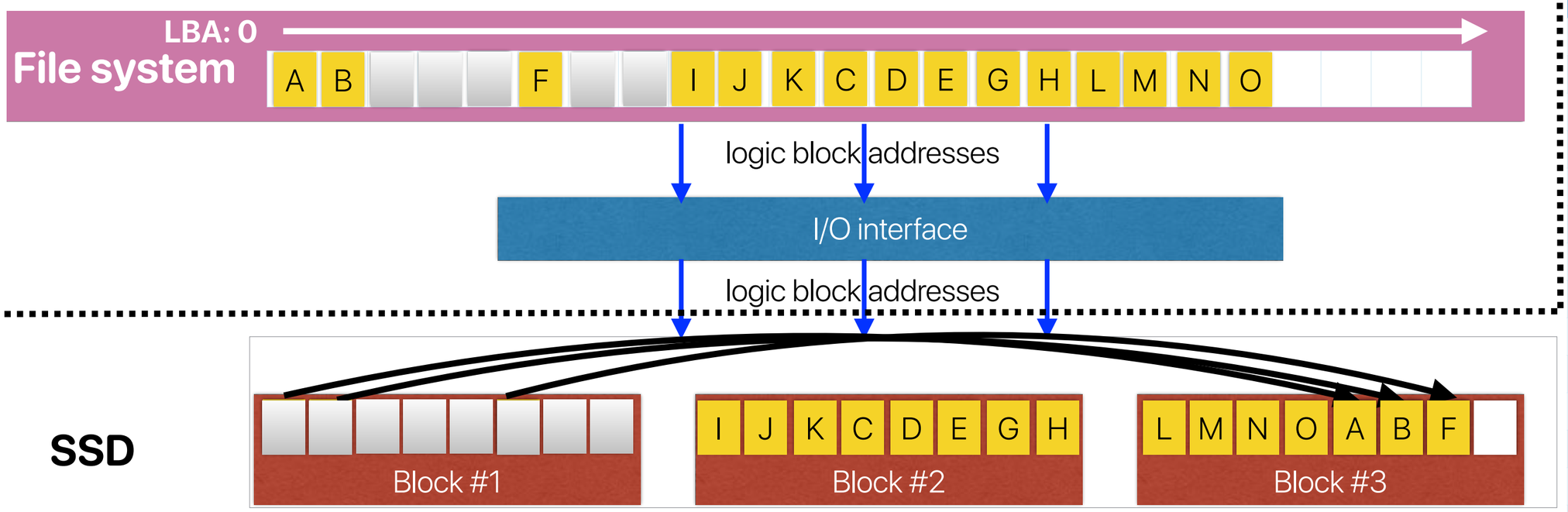
valid

free

FTL mapping table

LBA	block #	page #
0	1	0
1	1	1
2	-	-
3	-	-
4	-	-
5	1	5
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	-	-
21	-	-
22	-	-
23	-	-

Now, SSD wants to reclaim a block



invalid

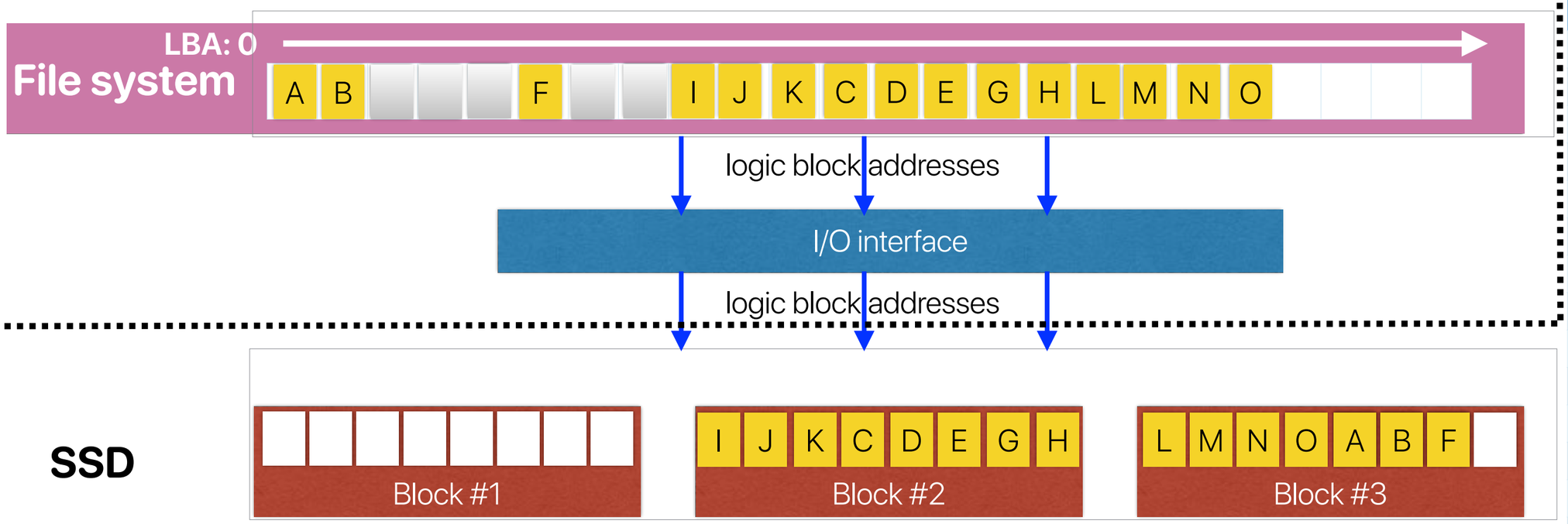
valid

free

FTL mapping table

LBA	block #	page #
0	3	4
1	3	5
2	-	-
3	-	-
4	-	-
5	3	6
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	-	-
21	-	-
22	-	-
23	-	-

Garbage collection on the SSD done!



FTL mapping table

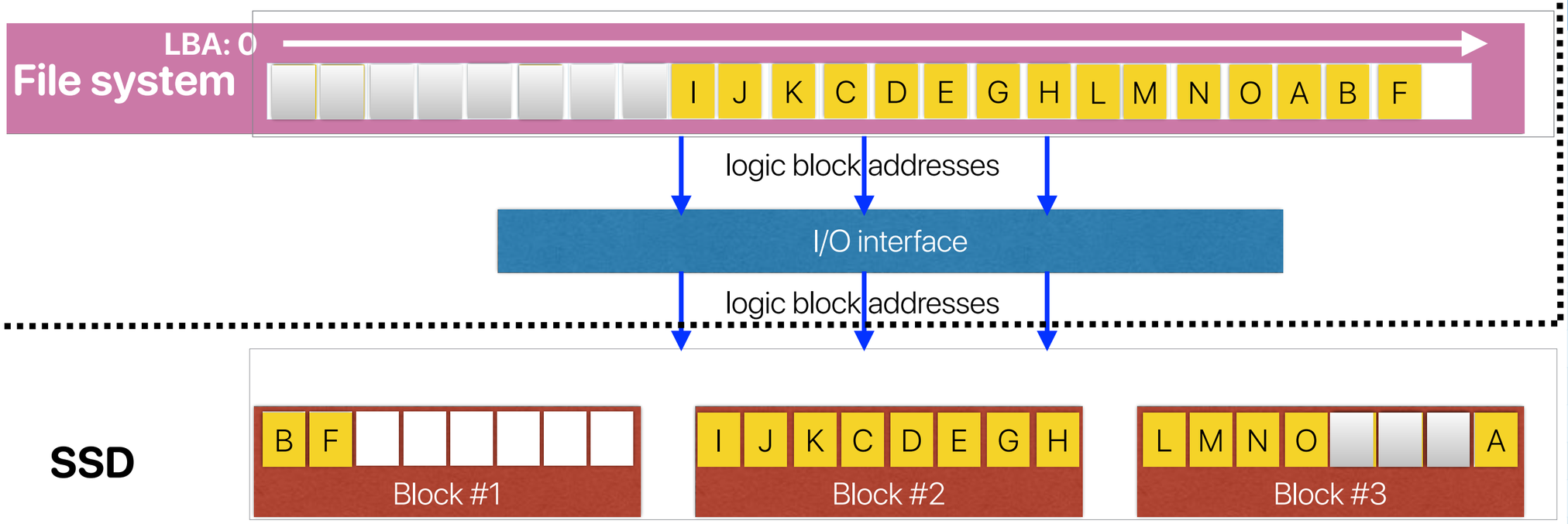
LBA	block #	page #
0	3	4
1	3	5
2	-	-
3	-	-
4	-	-
5	3	6
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	-	-
21	-	-
22	-	-
23	-	-

invalid

valid

free

What will happen if the FS wants to perform GC?



We could have avoided writing the stale A, B, F if they are coordinated!

invalid

valid

free

FTL mapping table

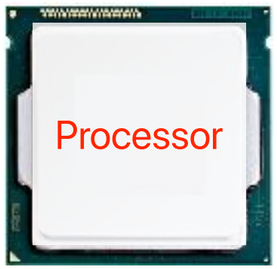
LBA	block #	page #
0	-	-
1	-	-
2	-	-
3	-	-
4	-	-
5	-	-
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	3	7
21	1	0
22	1	1
23	-	-

All problems in computer science can be solved by another level of indirection

–David Wheeler

...except for the problem of too many layers of indirection.

When the conventional interface meets modern applications



ORACLE NOSQL DATABASE

APACHE HBASE

Google Cloud BigTable

keys

Mapping Tables

Transaction support

Caches

Logs

files, offsets

File systems

logic block addresses

Read/Write

logic block addresses

Flash translation layer

physical addresses

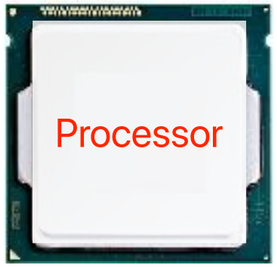
Address mapping

Log

When the conventional interface meets modern applications

- Log on log problems
  - Database logs, FS and FTL are all log-structured
  - Redundant functions can cause suboptimal performance
- Data translations/lookup in each layer
  - Waste resources to maintain the mapping
  - Addition latency for each data access

When the conventional interface meets modern applications



ORACLE NOSQL DATABASE

APACHE HBASE

Google Cloud BigTable

keys

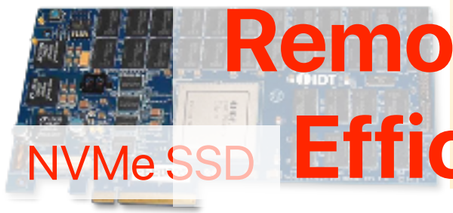
Mapping Tables | Transaction support | Caches | Logs

files, offsets

File systems

logic block addresses

Key-value Store



Avoid multiple indirections

Remove duplicated optimizations

Efficient data synchronization

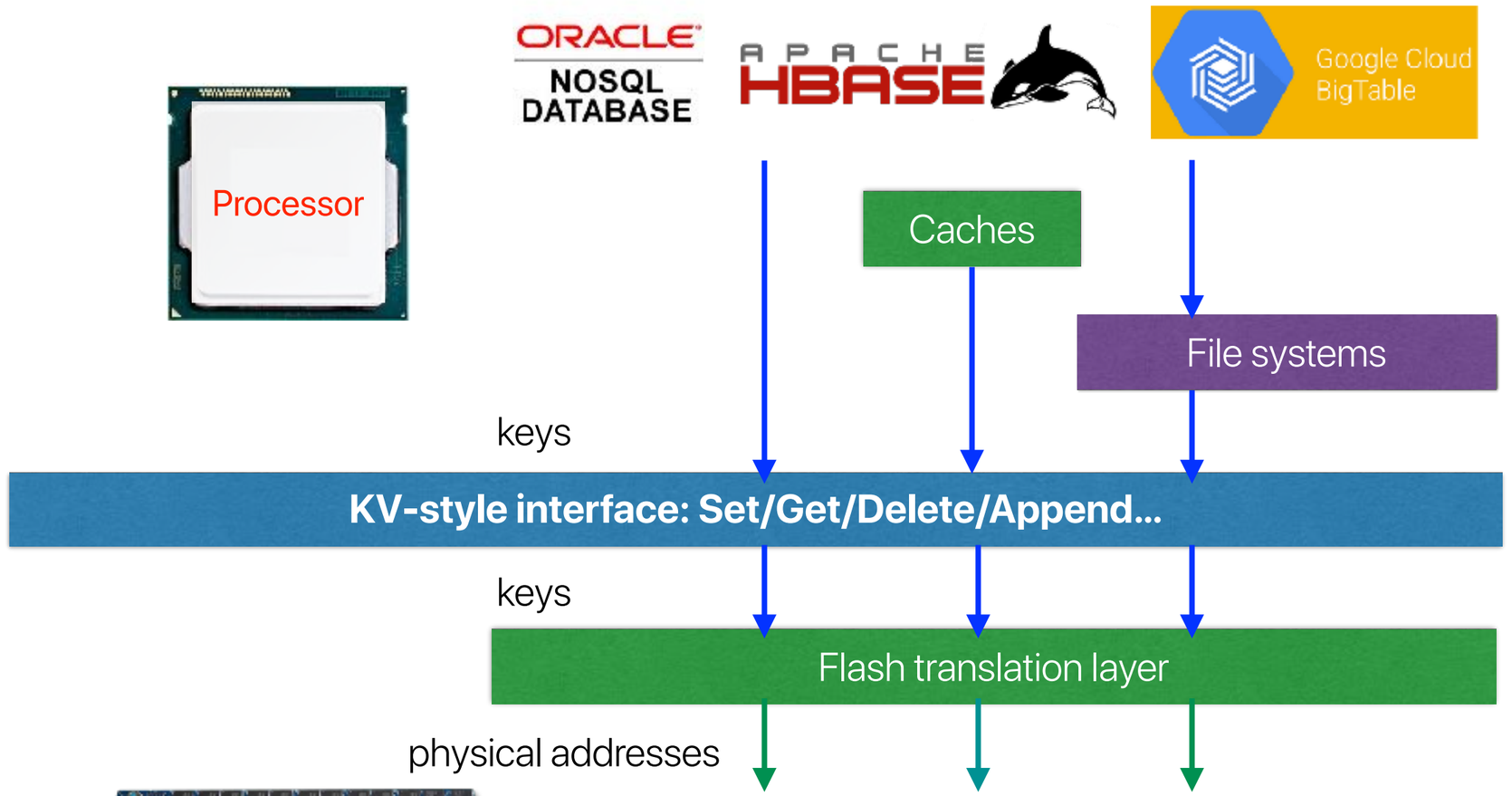
What if the SSD understands key-value?

- Remove the indirections
  - No need to maintain a mapping table
  - No need to translate between logical block addresses and physical addresses
- Avoid duplicated functions
  - Flash translation layer already does address remapping
  - Flash translation layer already maintain the physical memory as logs
- More efficient transaction support in hardware
  - No software synchronization overhead
  - The storage device itself exploits internal parallelism better

KAML

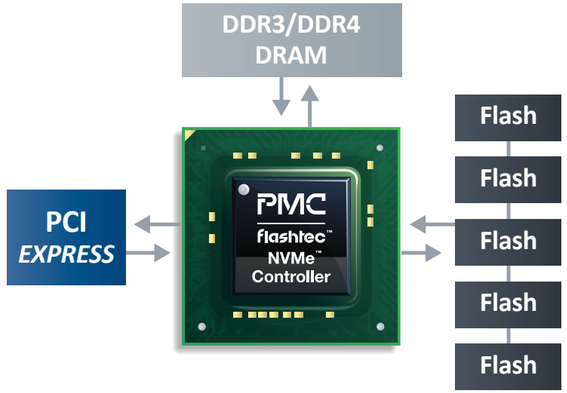
- Yanqin Jin, **Hung-Wei Tseng**, Steven Swanson and Yannis Papakonstantinou. KAML: A Flexible, High-Performance Key-Value SSD. In HPCA 2017.

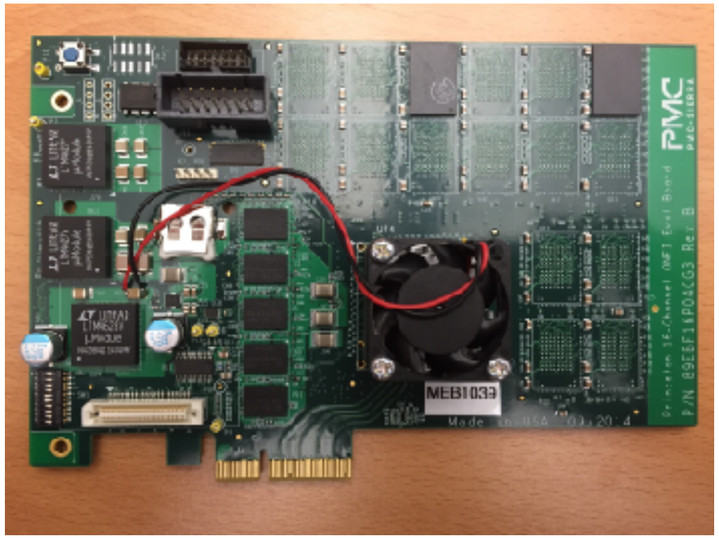
KAML: A Flexible, High-Performance Key-Value SSD



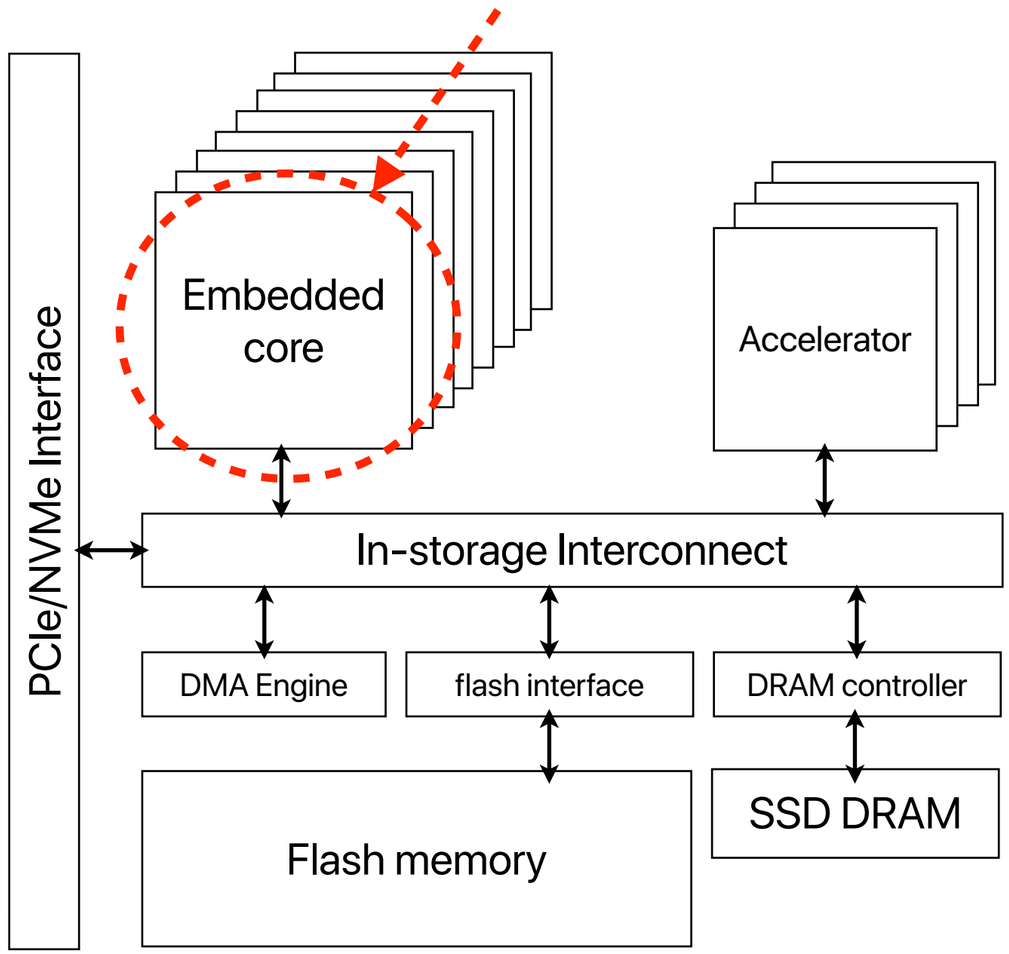
- Yanqin Jin, **Hung-Wei Tseng**, Steven Swanson and Yannis Papakonstantinou. KAML: A Flexible, High-Performance Key-Value SSD. In HPCA 2017.

Implementing KAML

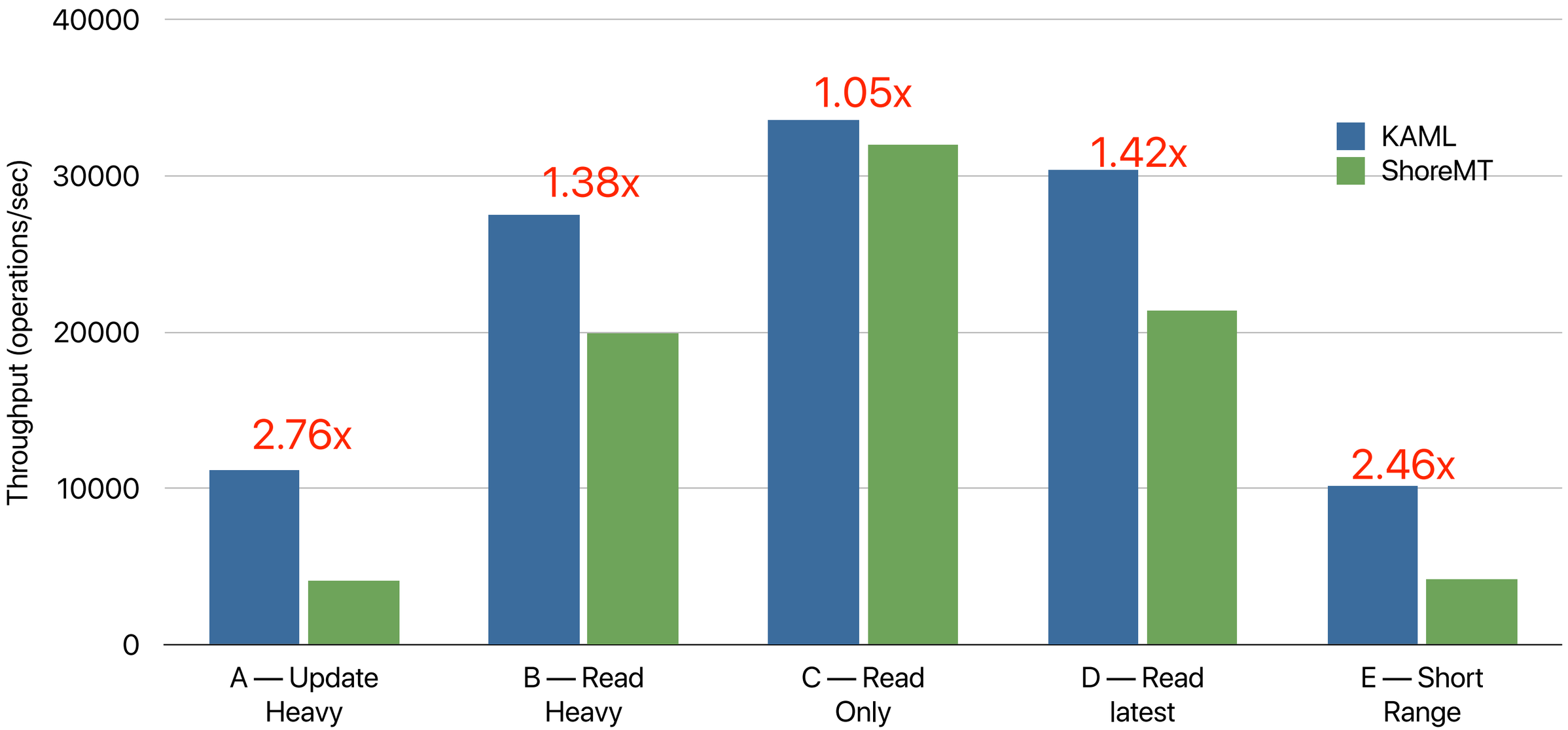




Managing key-value-like commands

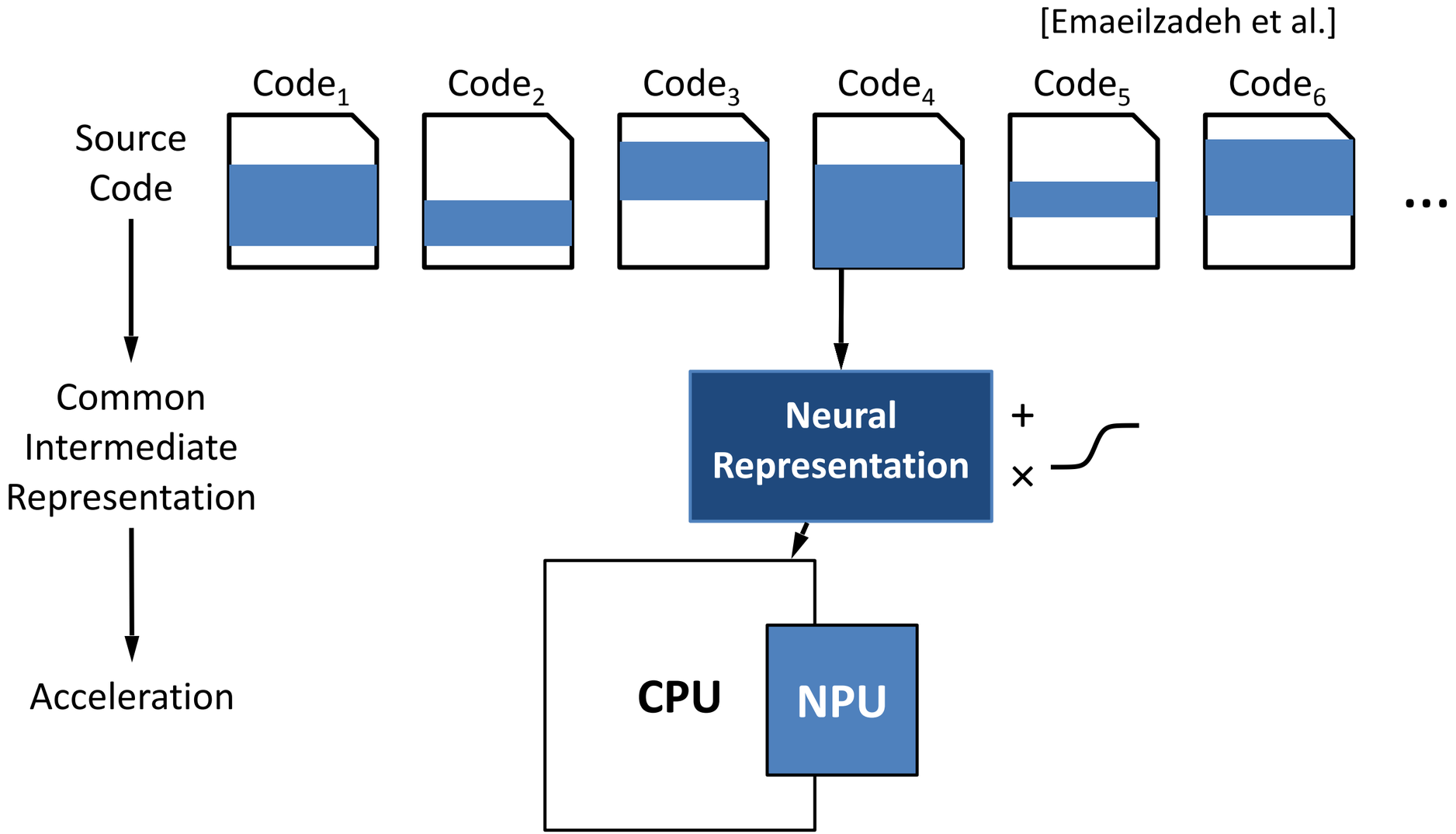


YCSB


40000
30000
20000
10000
0
Throughput (operations/sec)
KAML
ShoreMT
2.76x
1.38x
1.05x
1.42x
2.46x
A — Update Heavy
B — Read Heavy
C — Read Only
D — Read latest
E — Short Range

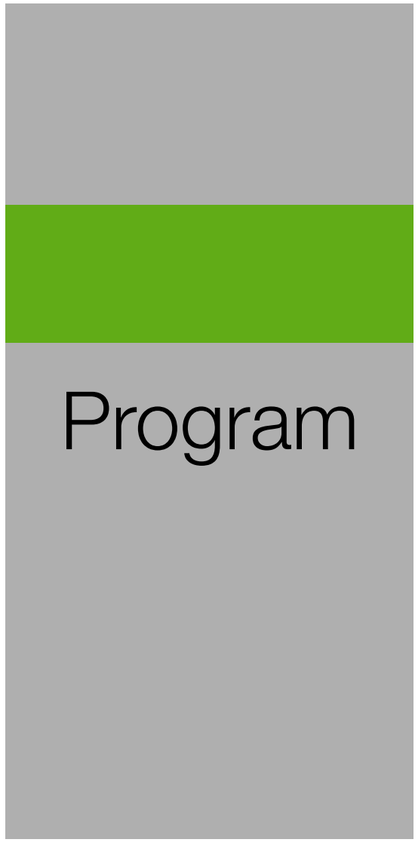
Approximate Computing

Neural Algorithmic Transformation



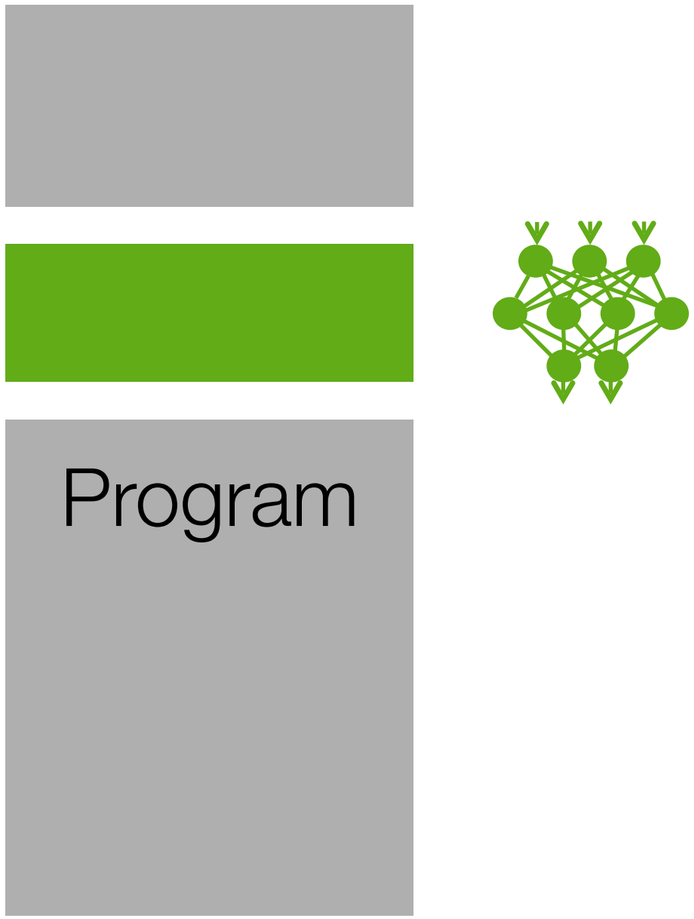
Program execution as a learning problem [Esmaeilzadeh NPU]

Program execution as a learning problem [Esmaeilzadeh NPU]



Find an approximate program component

Program execution as a learning problem [Esmaeilzadeh NPU]



Find an approximate program component

Compile the program and train a neural network

Program execution as a learning problem [Esmaeilzadeh NPU]

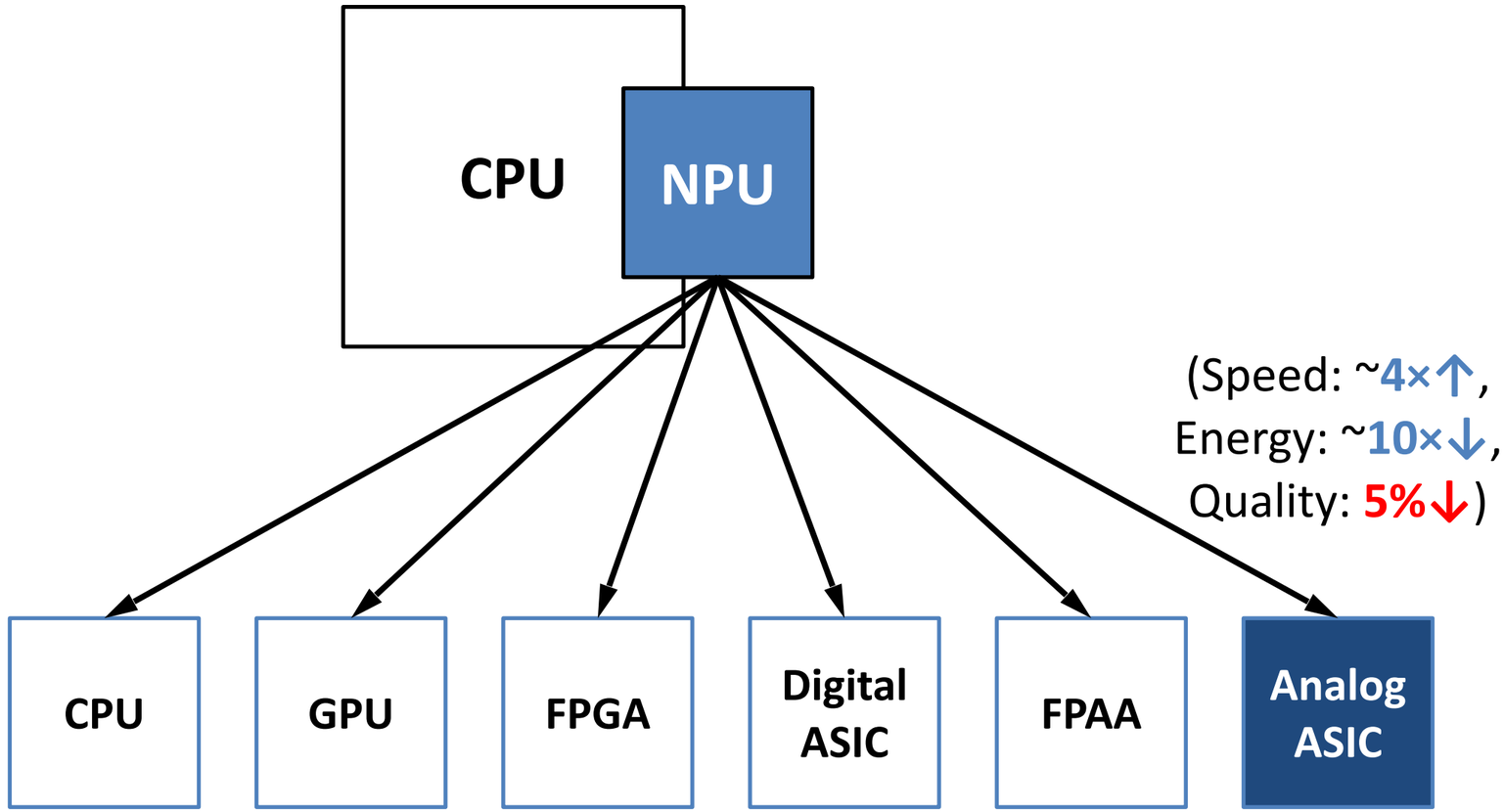
Find an approximate program component

Compile the program and train a neural network

Execute on a fast Neural Processing Unit (NPU)

Neural Acceleration

[Emaeilzadeh et al.]



(Speed: ~4×↑,
Energy: ~10×↓,
Quality: 5%↓)

3:00

Remind yourselves Approximate Computing — why/when is it a good idea? why/when is it a bad idea?

Good things about Approximate Computing?

Good things about Approximate Computing

- Reduce circuit size
- Avoid complex computation
- Smaller data
- Can use “unreliable” hardware

Bad things about Approximate Computing?

Bad things about Approximate Computing

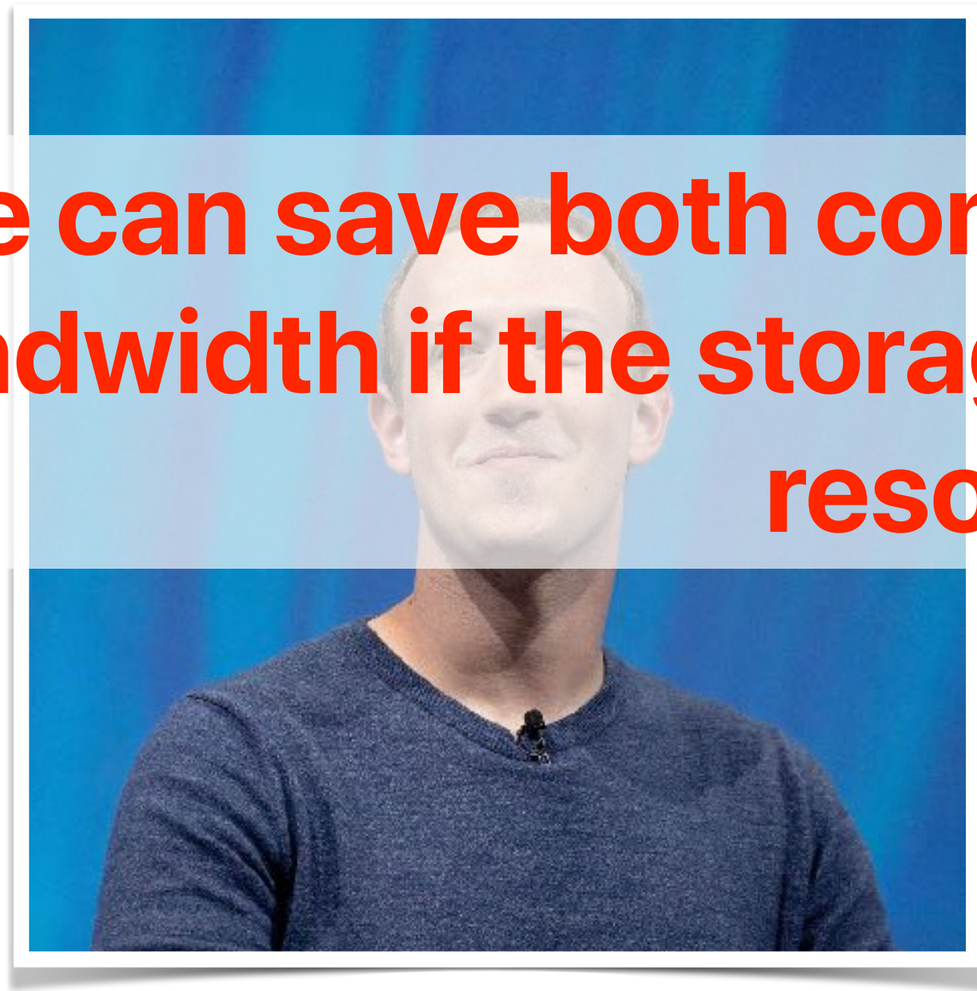
- Result
- Re-computation due to low quality
- Limited applications

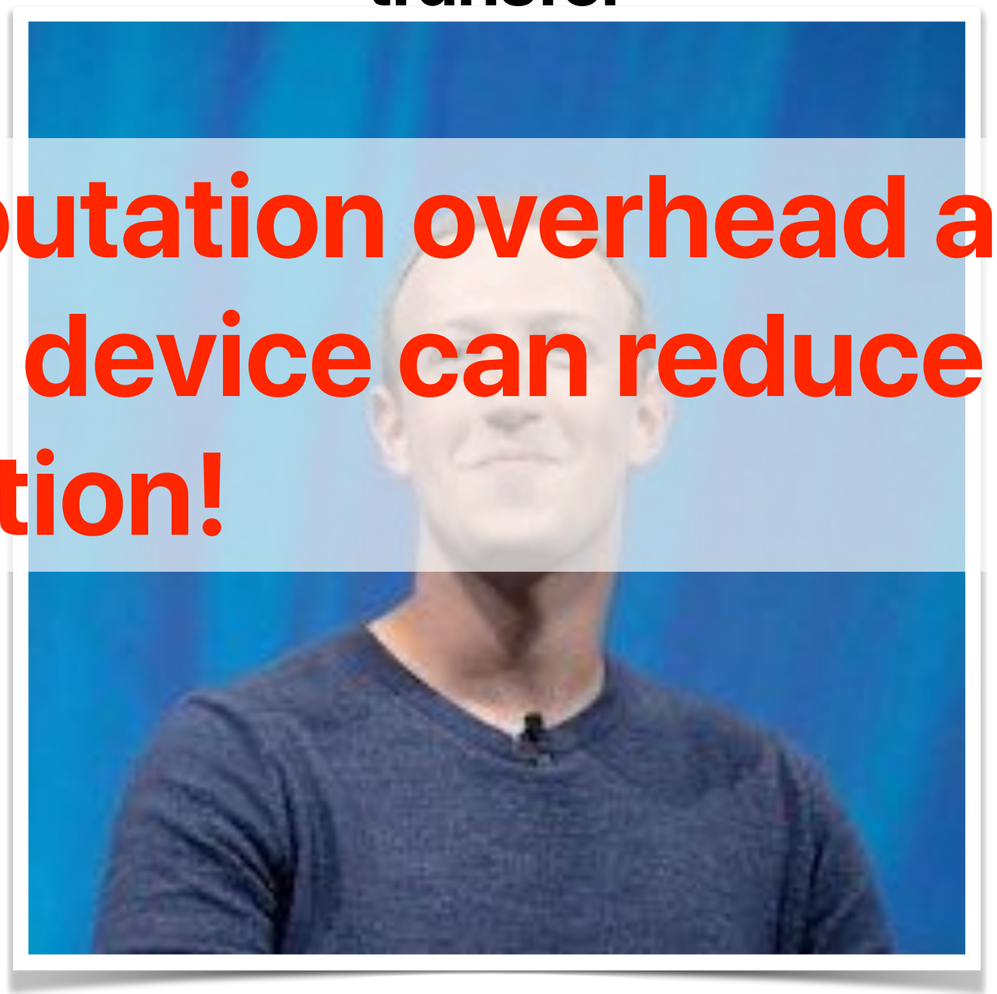
What is “approximate computing”?

- Building ***acceptable*** systems out of unreliable/inaccurate hardware and software components
  - Compute, storage and communication
  - Inaccuracy can be deterministic or non-deterministic
- Trade fidelity of results to achieve better efficiency and performance
  - Enable new applications or meet constraints (cost, power, etc.)
- In summary, embrace widespread output quality trade-off and “closer to physics”, non-deterministic hardware

We don’t need full resolution in many cases

Only need 25% bytes to transfer





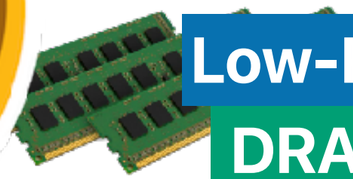
We can save both computation overhead and bandwidth if the storage device can reduce the resolution!

Accelerators in General-Purpose Computers

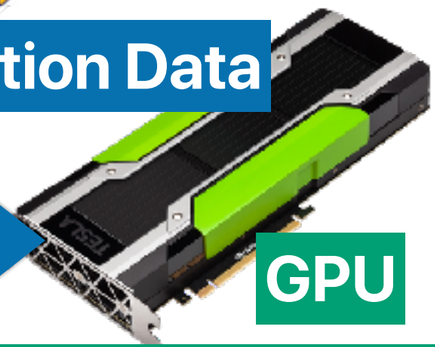
Retrieve Raw Data

Adjust Data Resolutions

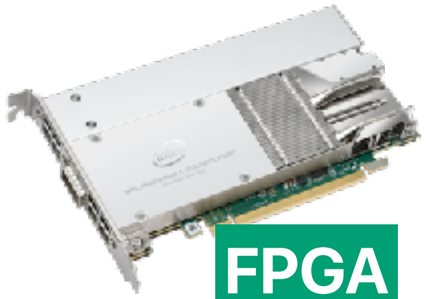
Low-Resolution Data



DRAM



GPU

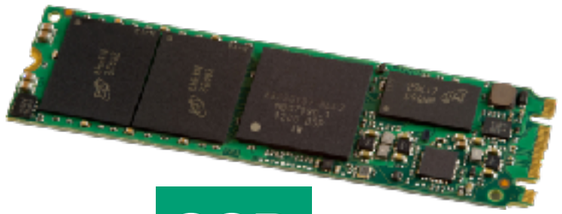


FPGA

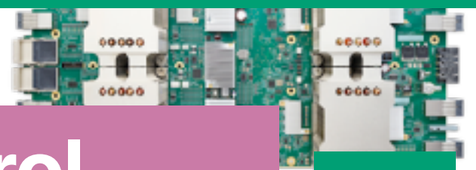
Raw Data

CPU

Compute



SSD



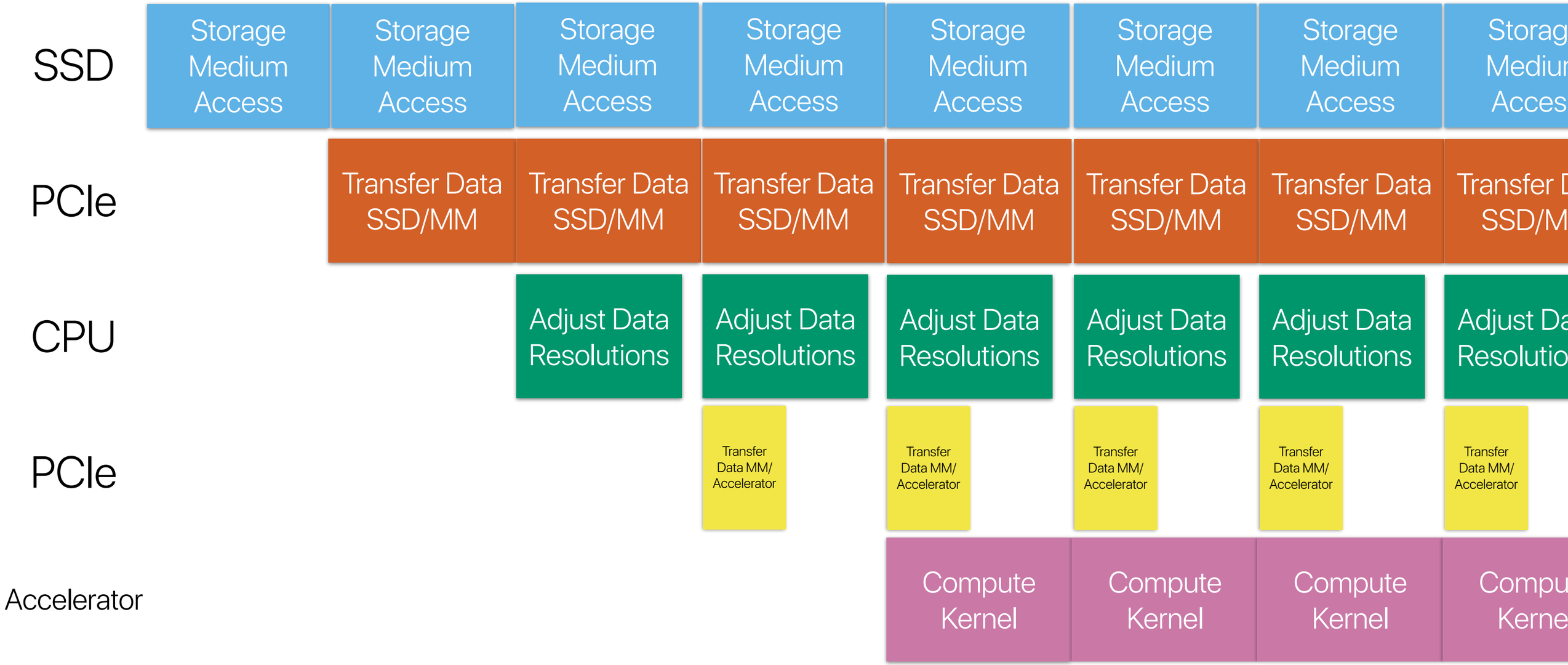
TPU

System Interconnect (e.g., PCIe)

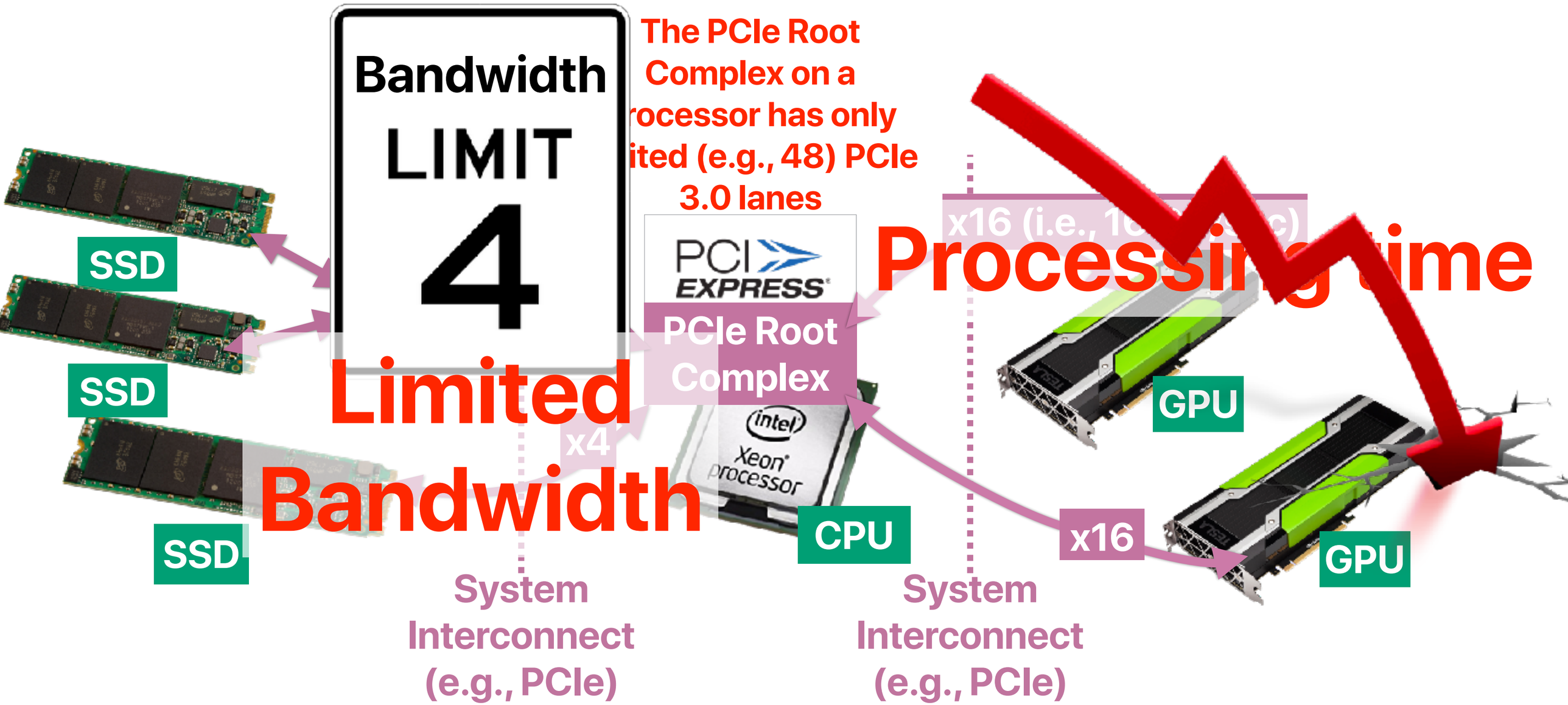
Quality Control

(e.g., PCIe)

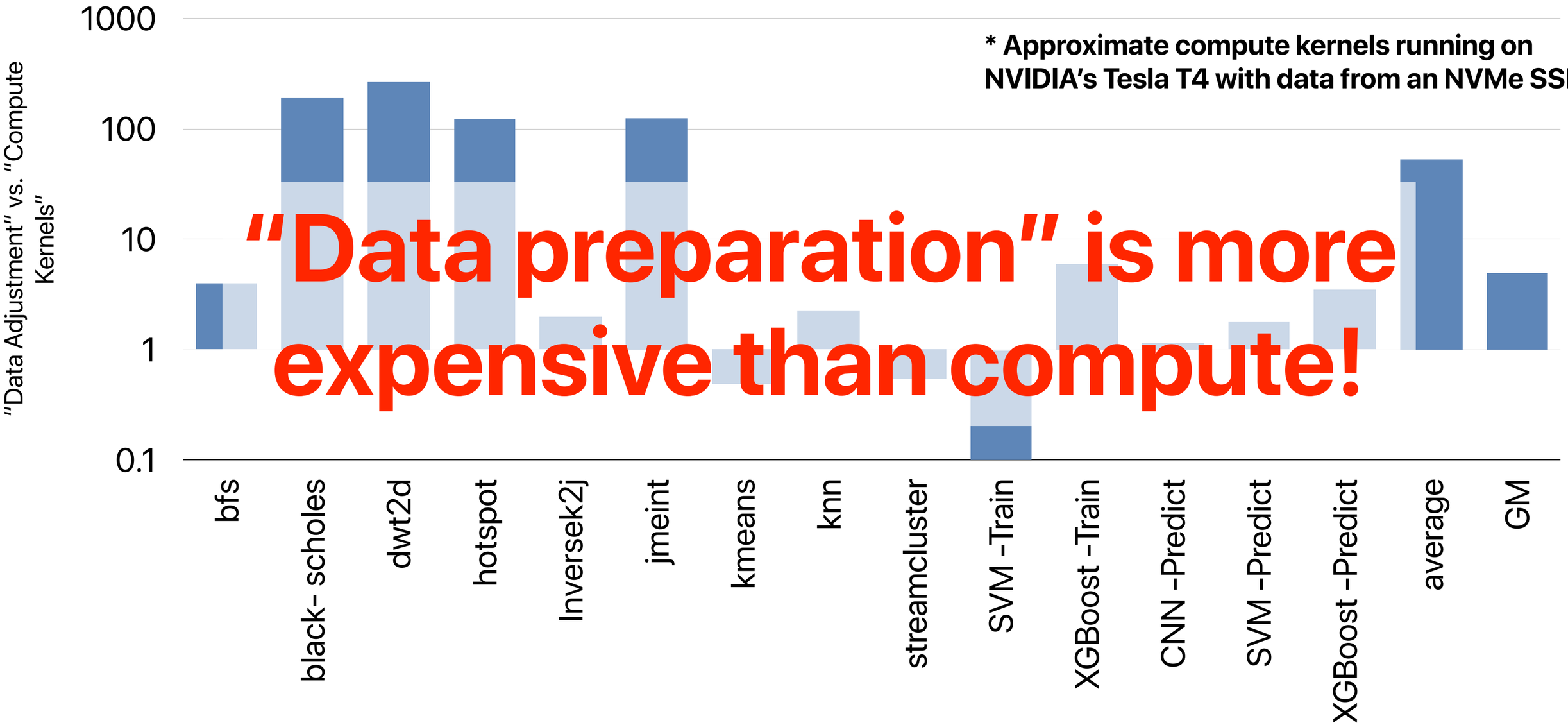
Data processing pipeline in modern computers



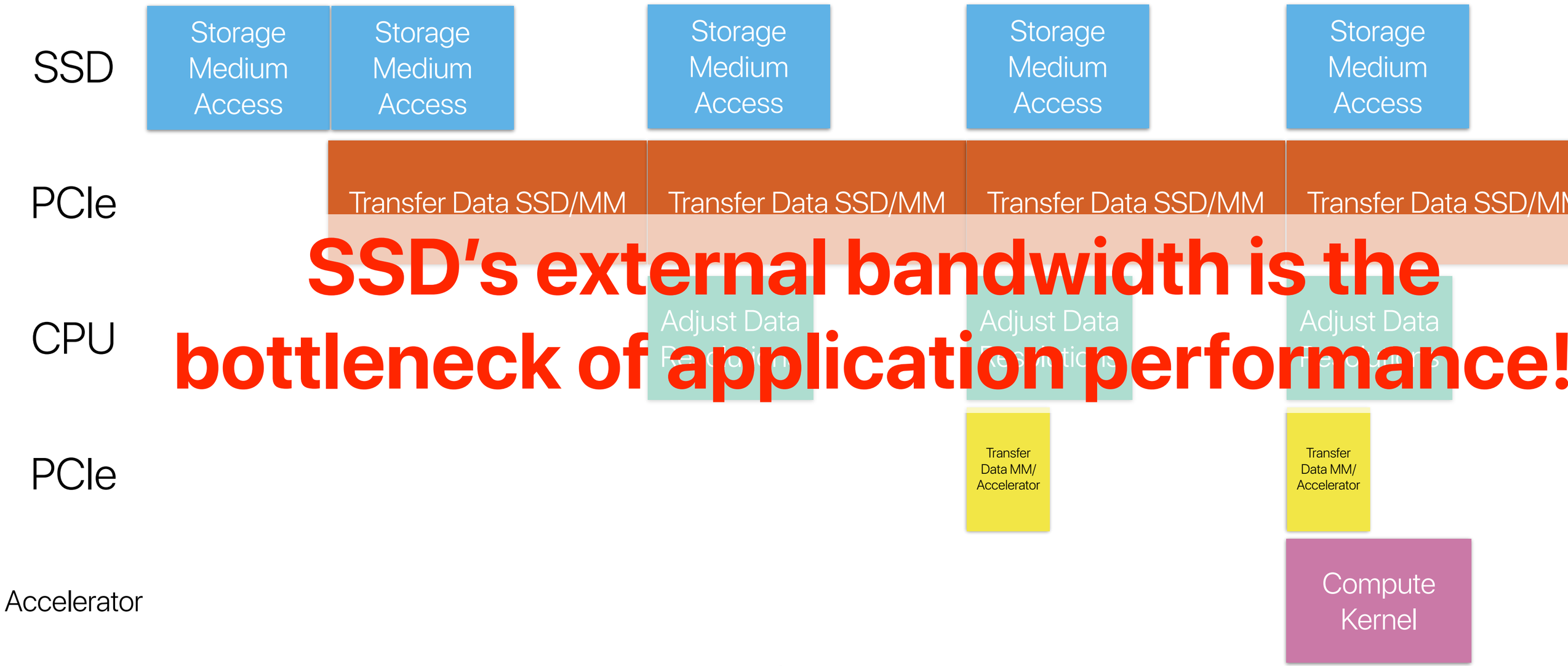
Limited Interconnect Bandwidth



Data Preparation Can be 100x More Than Compute Kernels



The “actual” pipeline …



SSD’s external bandwidth is the bottleneck of application performance!

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)

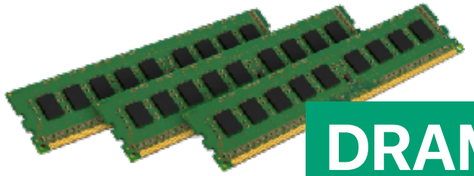
Varifocal Storage: A Holistic System Architecture

Retrieve Raw Data

Adjust Data Resolutions

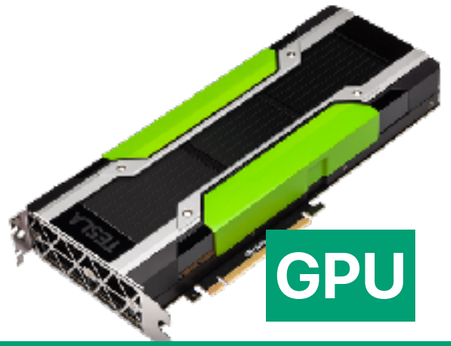
Varifocal Storage (VS)

System Interconnect (e.g., PCIe)

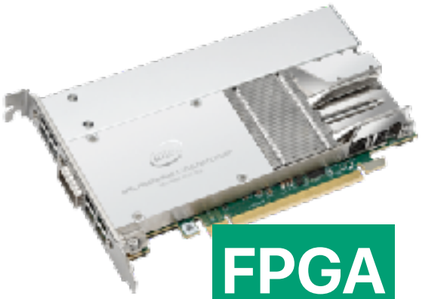


DRAM

CPU



GPU



FPGA

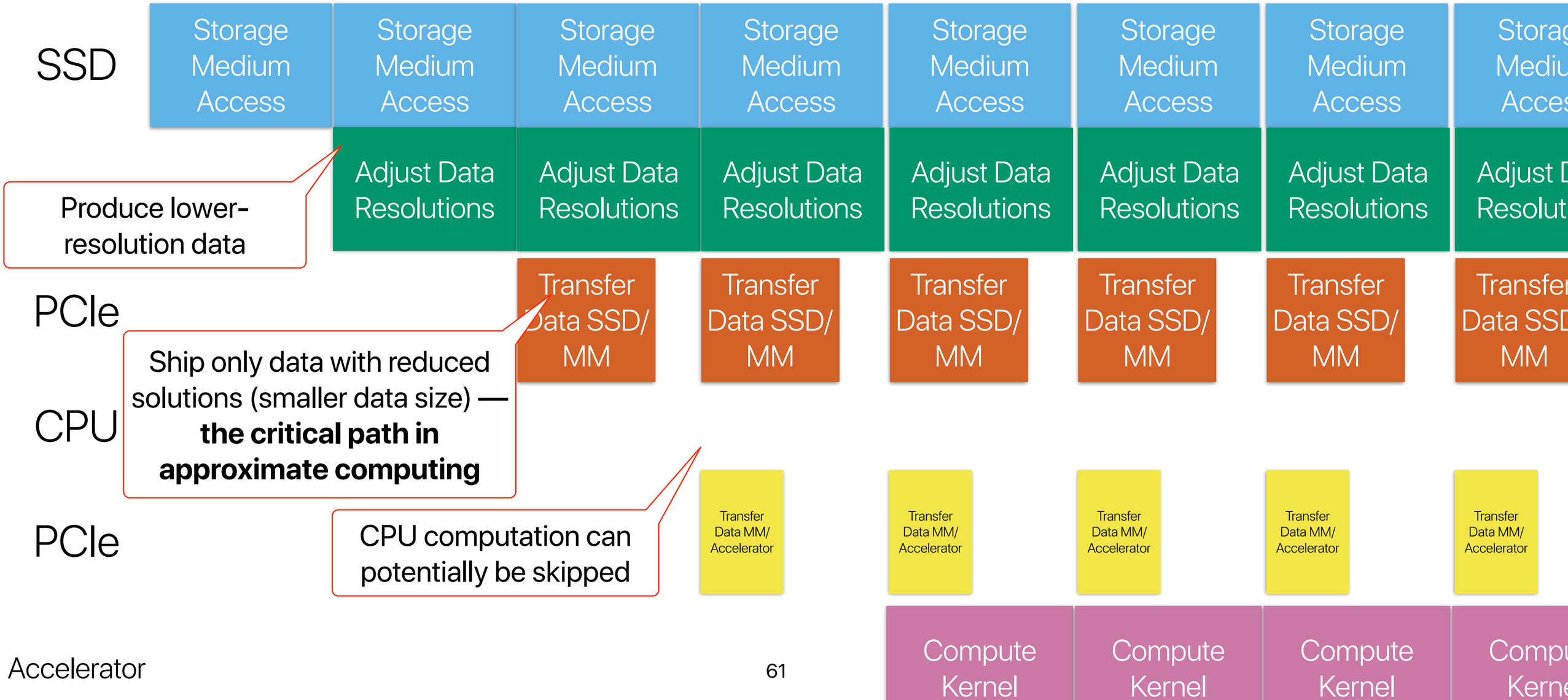
Compute

TPU

Quality Control

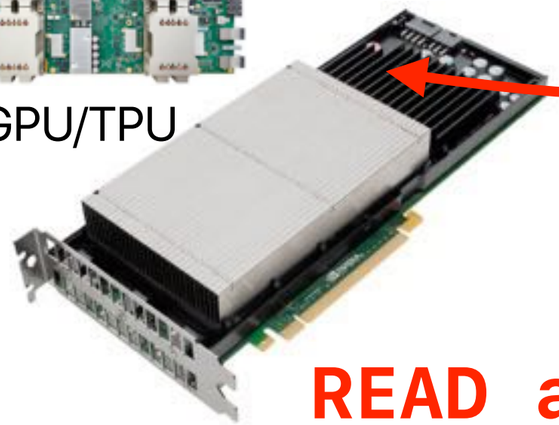
System Interconnect (e.g., PCIe)

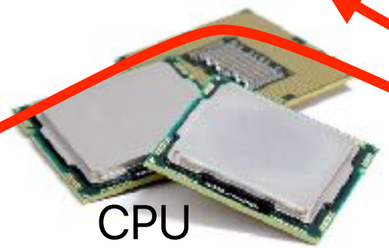
Data processing with ISP



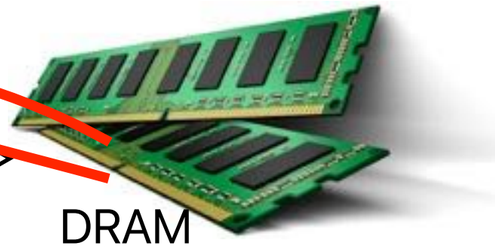
The “concept” of Varifocal Storage

GPU/TPU



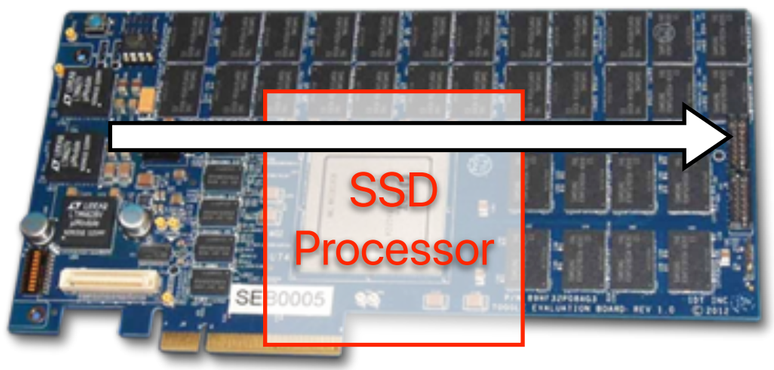


CPU



DRAM

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



extension of
NVMe interface

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

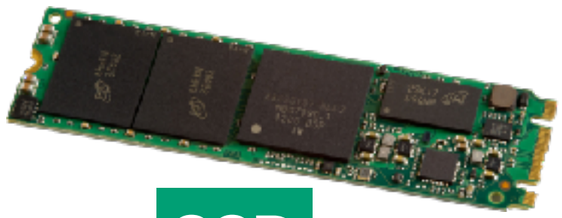
Better-resolution

50% Resolution

Re-compute

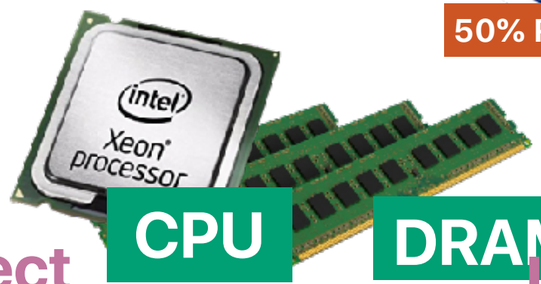
Test

Bear



SSD

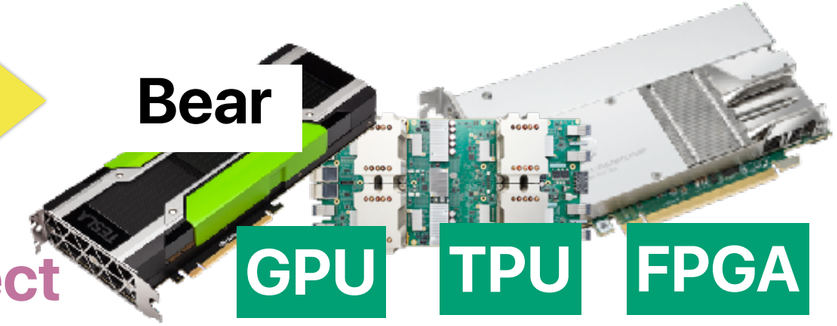
System Interconnect (e.g., PCIe)



CPU

DRAM

System Interconnect (e.g., PCIe)

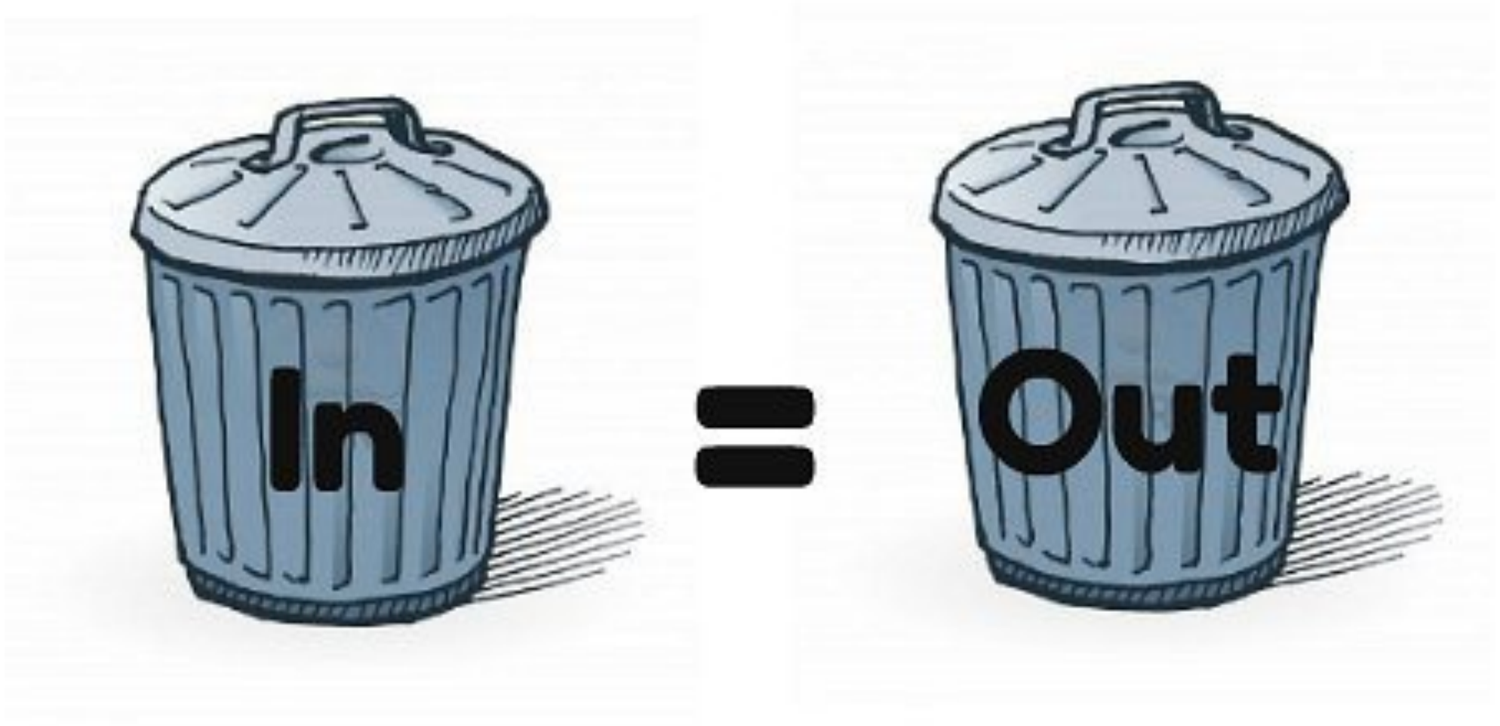


GPU

TPU

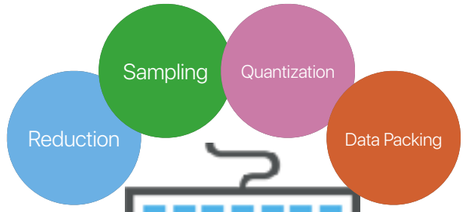
FPGA

Garbage in, garbage out



Quality Control

Autofocus



Raw Data

Lower-resolution

25% Resolution

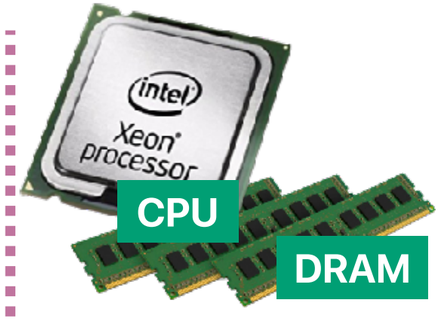
Reject if it becomes garbage

Better-resolution

50% Resolution

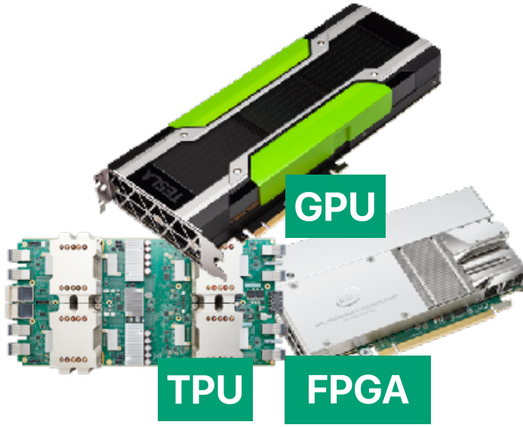
Use this resolution for the same dataset later

Varifocal Storage (VS)



CPU

DRAM



GPU

TPU

FPGA

Pass!

Compute

Test

Bear

System Interconnect (e.g., PCIe)

System Interconnect (e.g., PCIe)

iFilter

Autofocus on Operator A

25% Resolution

Autofocus on Operator B

45% Resolution

Autofocus on Operator C

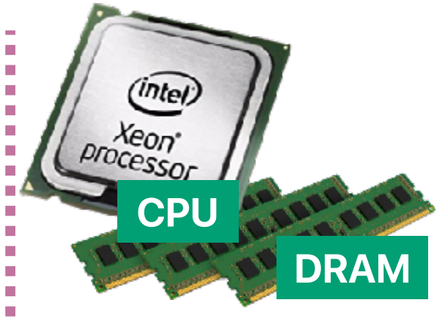
50% Resolution

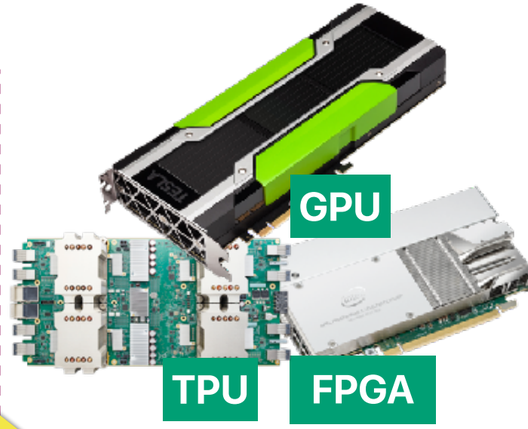
Varifocal Storage (VS)

Pass!

Use this resolution/ operator for the same dataset later

Pass!





Send Smaller One & Compute

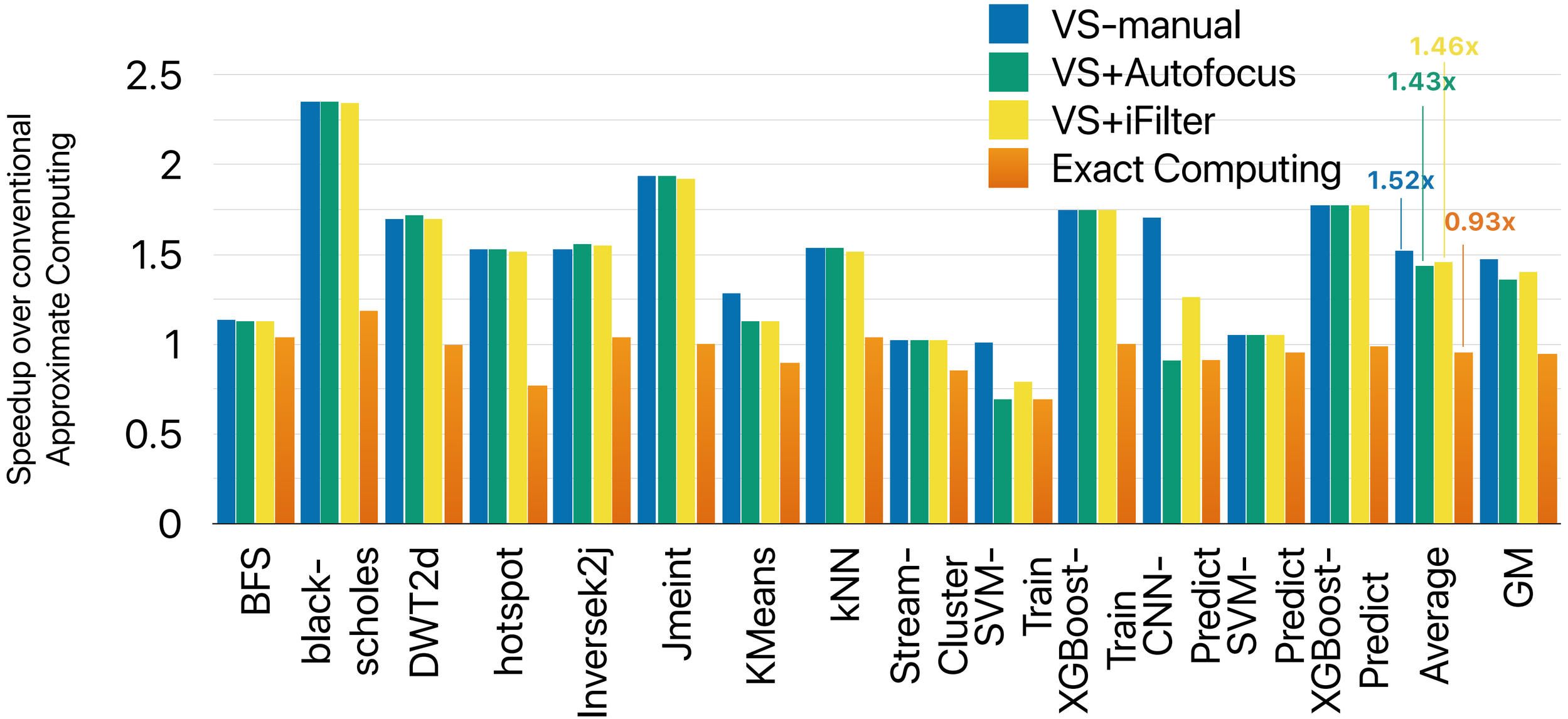
Test

Bear

System Interconnect (e.g., PCIe)

System Interconnect (e.g., PCIe)

Performance of Varifocal Storage (VS)



Announcement

- Final project presentation — 3/21 11:30a-2:30p (SSC 125)
  - Please book your order on the calendar https://calendar.google.com/calendar/u/0/selfsched?sstoken=UURHeVJHWDhlUXhhfGRlZmF1bHR8ZGQ5ODhhOGRhYjU1NGRjOWY1ODM3MmZiZjlmNmE0ODU
  - Pizza provided
- No lecture next Tuesday, but we'll be back on Thursday

Electrical
Computer Science
Engineering

277

つづく