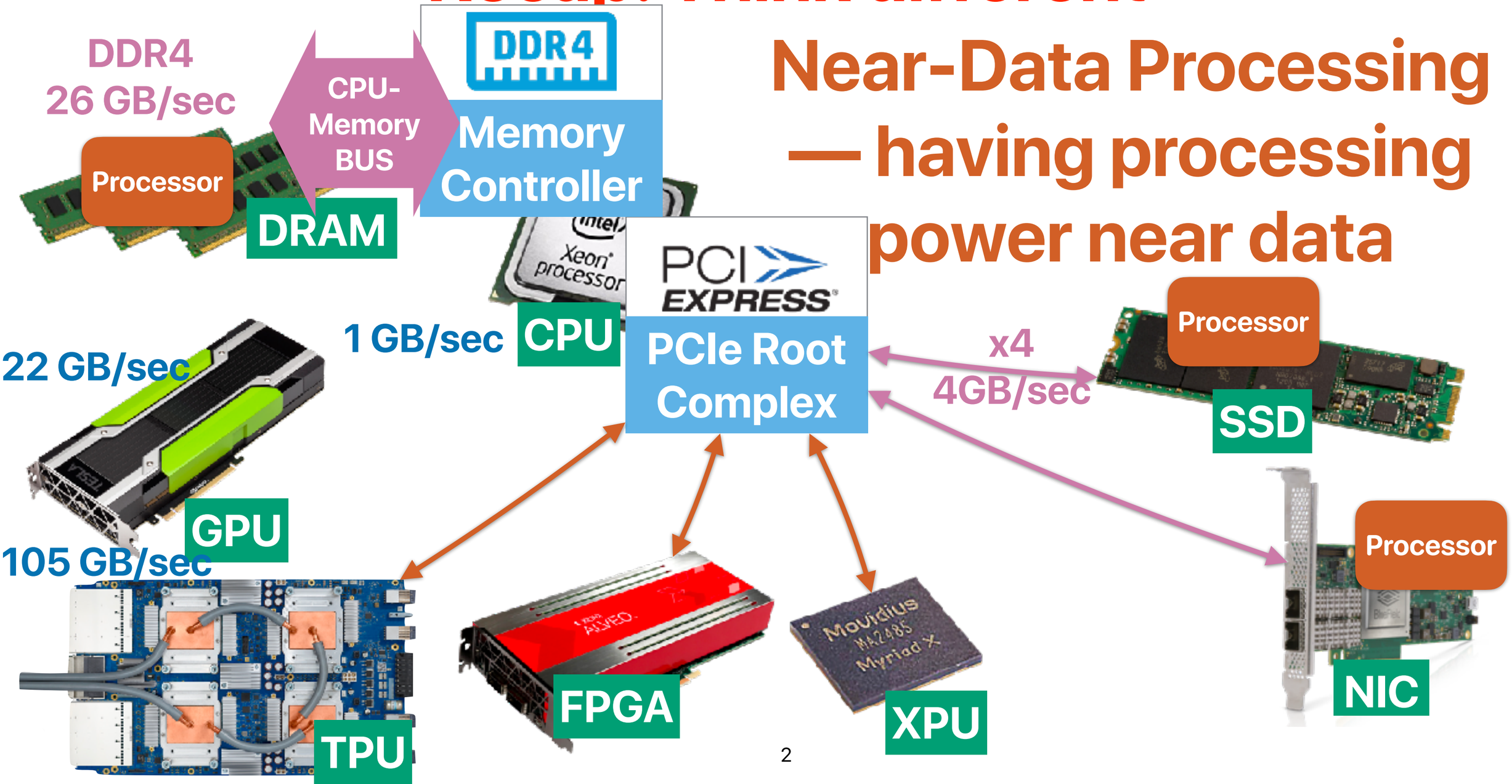
Think Different (3): In-Storage Processing (3)

Hung-Wei Tseng

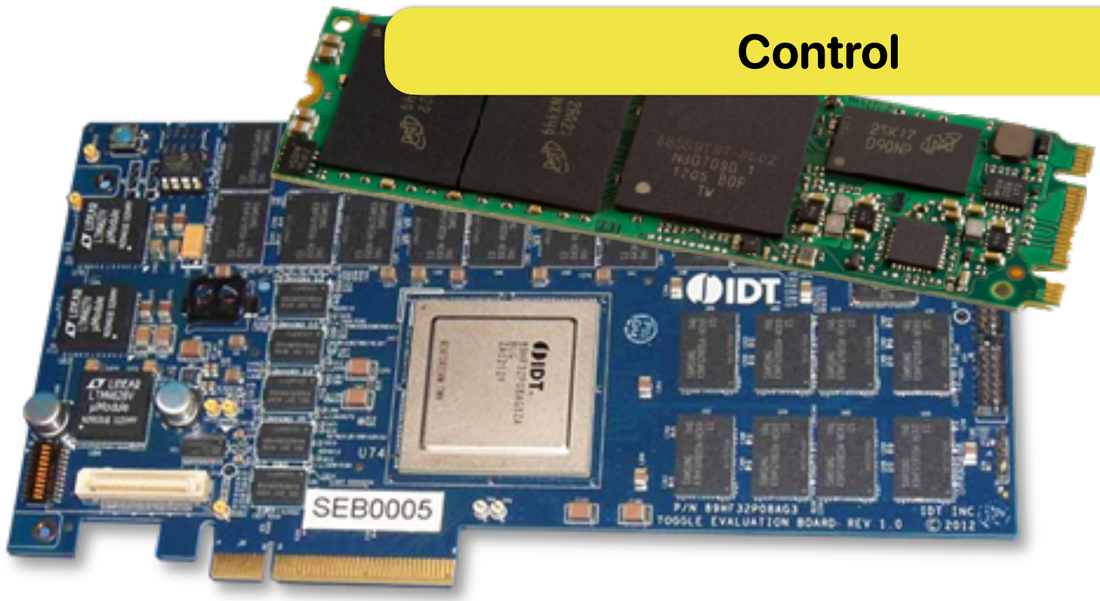
Recap: Think different

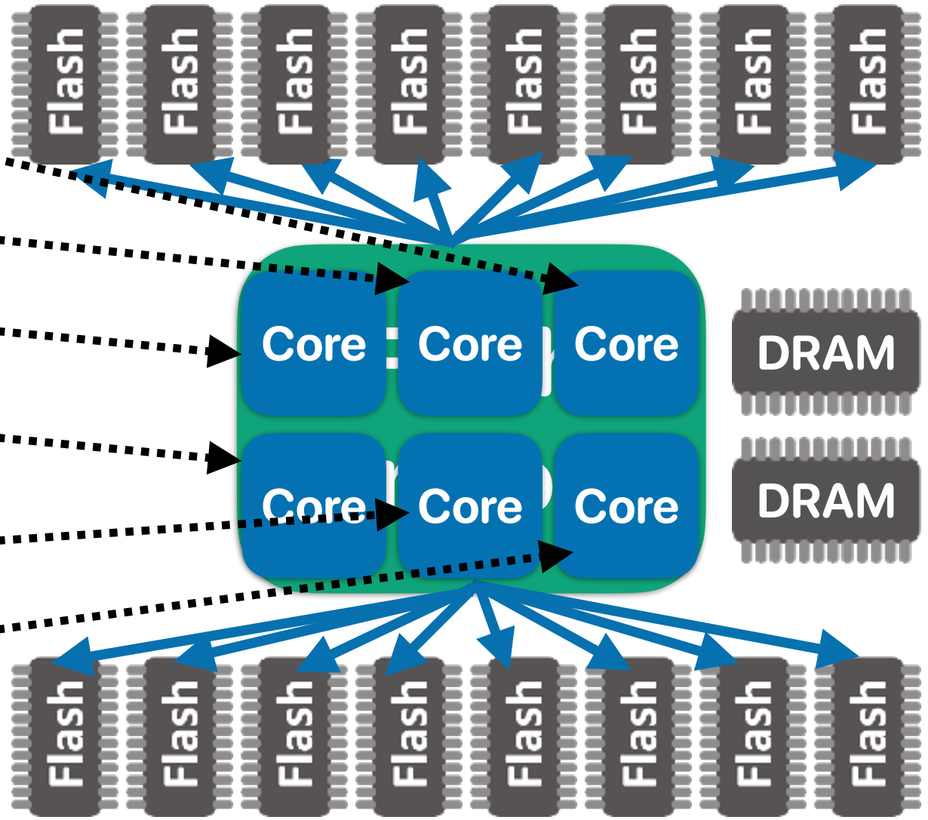
Near-Data Processing — having processing power near data



Recap: SSD Controller Architectures

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





Recap: Firmware

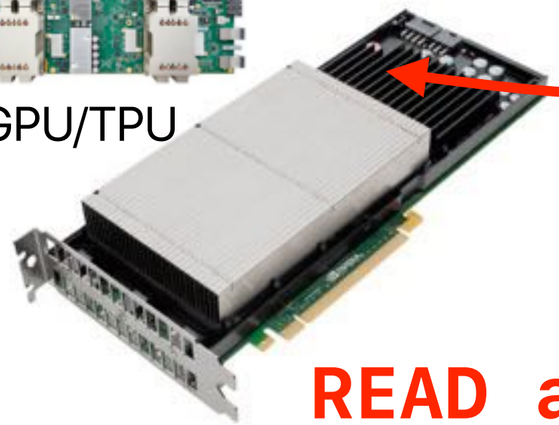
- What is firmware?
  - A software program directly interacts with hardware without an operating system
  - Typically a program works in between hardware and application "software"
- Why firmware?
  - You don't need to implement an application-specific IC for each feature
  - You can upgrade the firmware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

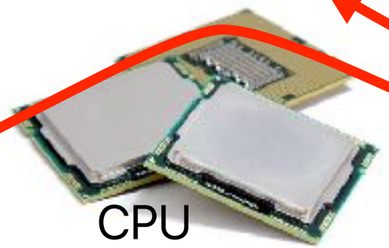
Recap: What applications/features do you have in mind?

- Simple arithmetic operations — if the result is smaller than the raw data and the computation is not too intense
- Reduce precision
- Create application objects

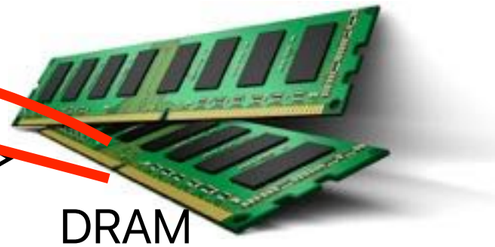
The “concept” of Varifocal Storage

GPU/TPU



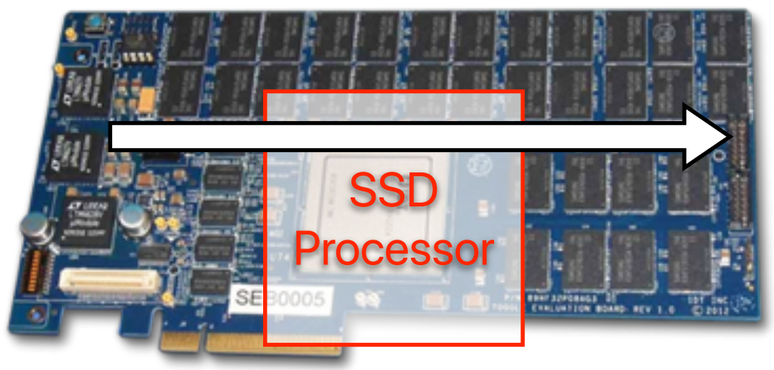


CPU



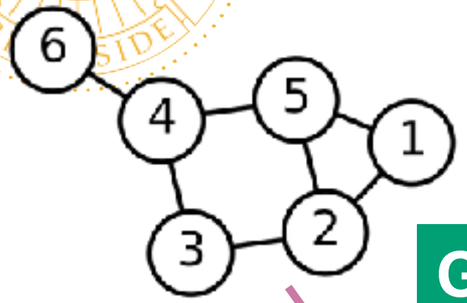
DRAM

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



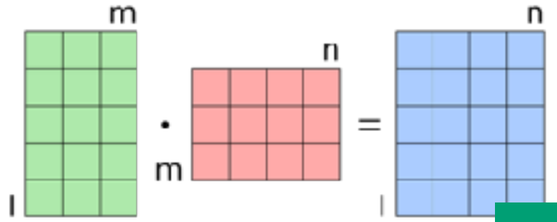
extension of NVMe interface

NDS: N-Dimensional Storage



Graph

read/write	
coordinate	1, 0
dimensionality	M, N

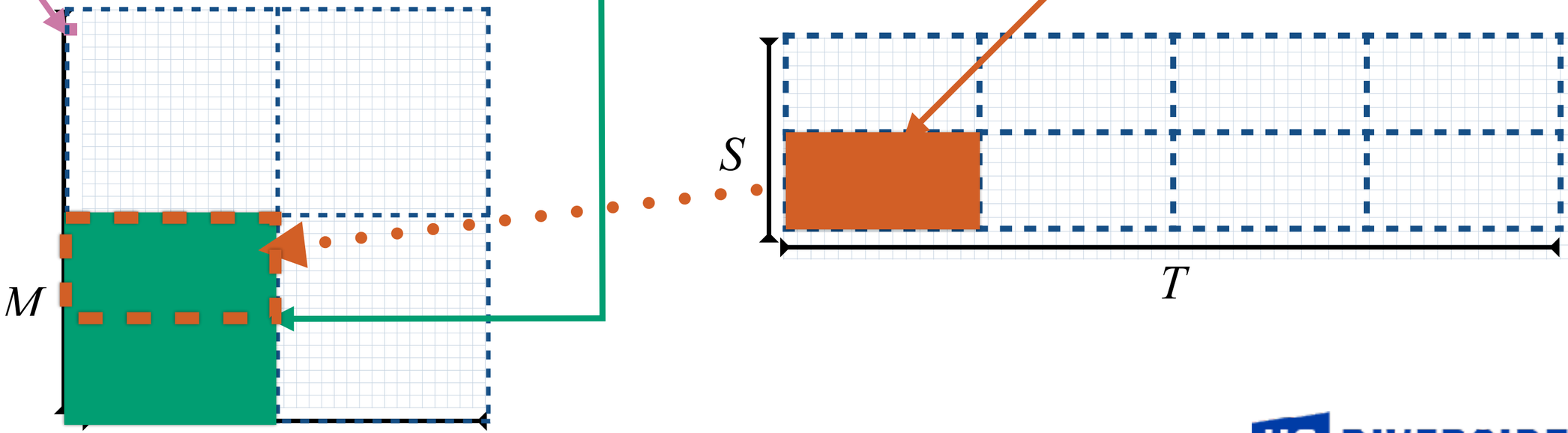


GEMM

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



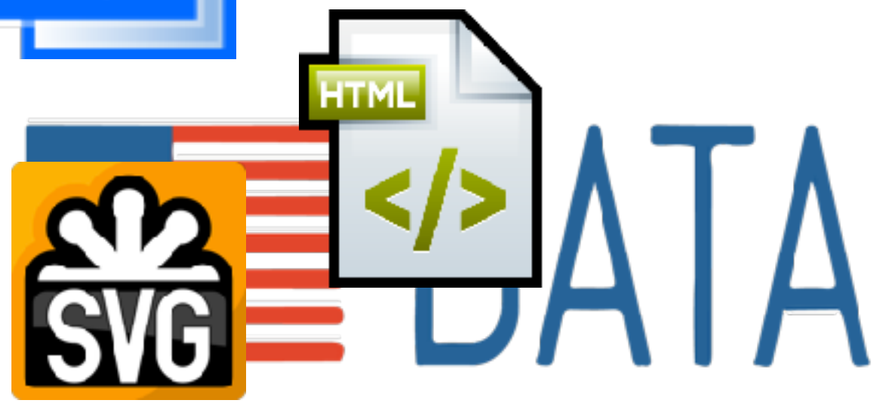
Case: what do you expect the performance of the program looks like?

3:00

```
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
#include <ctime>
#include <iostream>
#include <climits>
#include <sys/time.h>

int main(int argc, char **argv)
{
    // input data
    unsigned array_size = 100000000;
    FILE *fp;
    int *data, *sorted;
    int option = atoi(argv[1]);
    double total_time;
    struct timeval time_start, time_end;
    if(argc > 2)
        array_size = atoi(argv[2]);
    long long sum = 0;
    fp = fopen(argv[1],"r");
    data = (int *)malloc(sizeof(int)*array_size);
    sorted = (int *)malloc(sizeof(int)*array_size);

    gettimeofday(&time_start, NULL);
    for(int i=0;i<array_size;i++)
    {
        fscanf(fp, "%d", &data[i]);
    }
    gettimeofday(&time_end, NULL);
    total_time = ((time_end.tv_sec * 1000000 +
time_end.tv_usec) - (time_start.tv_sec * 1000000 +
time_start.tv_usec));
    fprintf(stderr,"Scan Latency: %.0lf us, Goodput
(Integers Per Second): %lf\n",total_time,
array_size*1000000/total_time);

    gettimeofday(&time_start, NULL);
    std::sort(data, data + array_size);
    gettimeofday(&time_end, NULL);
    total_time = ((time_end.tv_sec * 1000000 +
time_end.tv_usec) - (time_start.tv_sec * 1000000 +
time_start.tv_usec));
    fprintf(stderr,"Sort Latency: %.0lf us, Goodput
(Integers Per Second): %lf\n",total_time,
array_size*1000000/total_time);
    fprintf(stderr, "Samping points: data[0: %d, mid: %d,
end: %d]\n", data[0], data[array_size>1],
data[array_size-1]);
    fclose(fp);
    return 0;
}
```

https://github.com/hungweitseng/EE277/blob/main/demo/sort/arraySort.cpp

There are lots of data formats



ASCII

“12345678”

⇩

0x3231 0x3433 0x3635 0x3837 0x000a

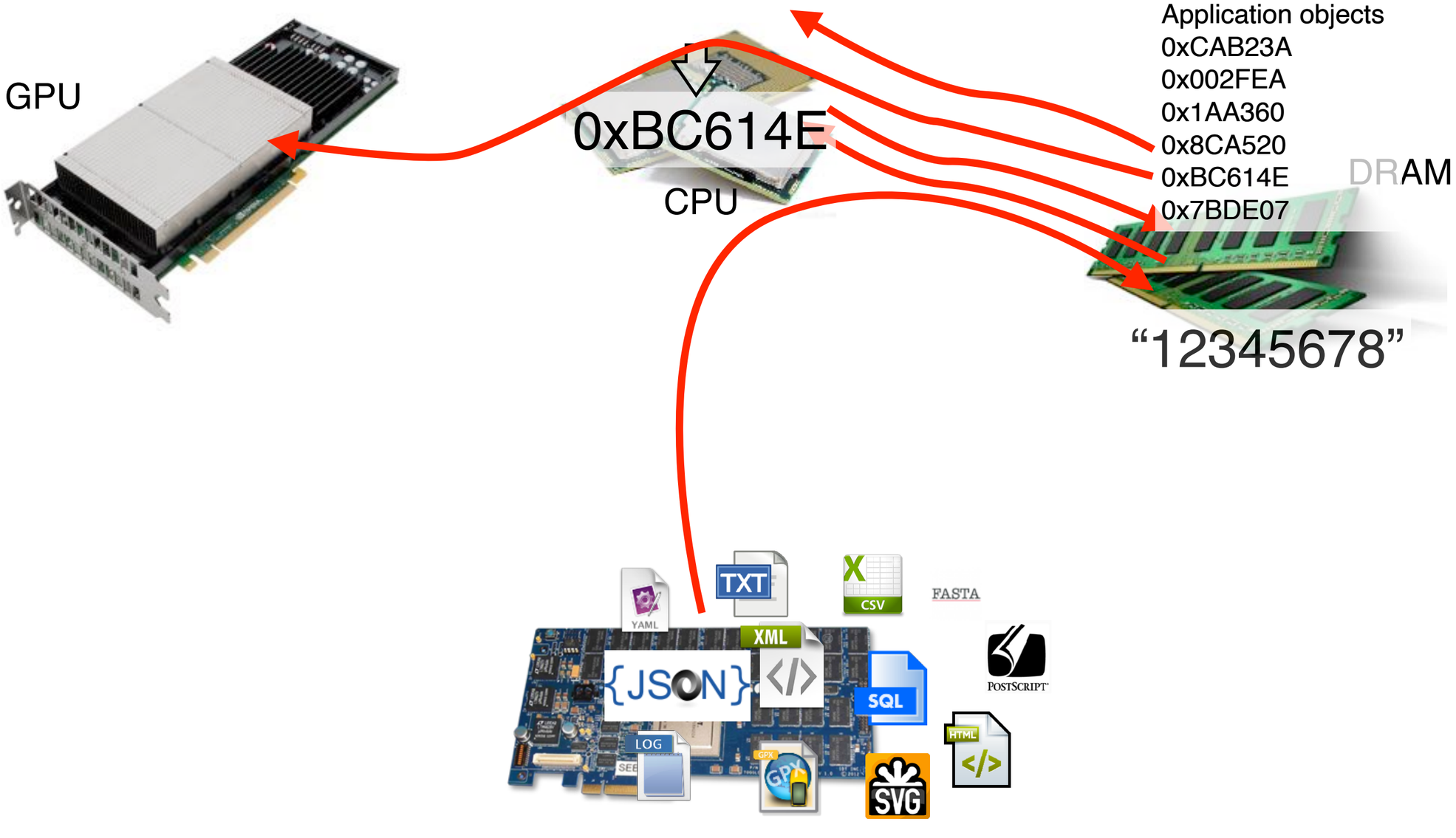
Binary

12345678

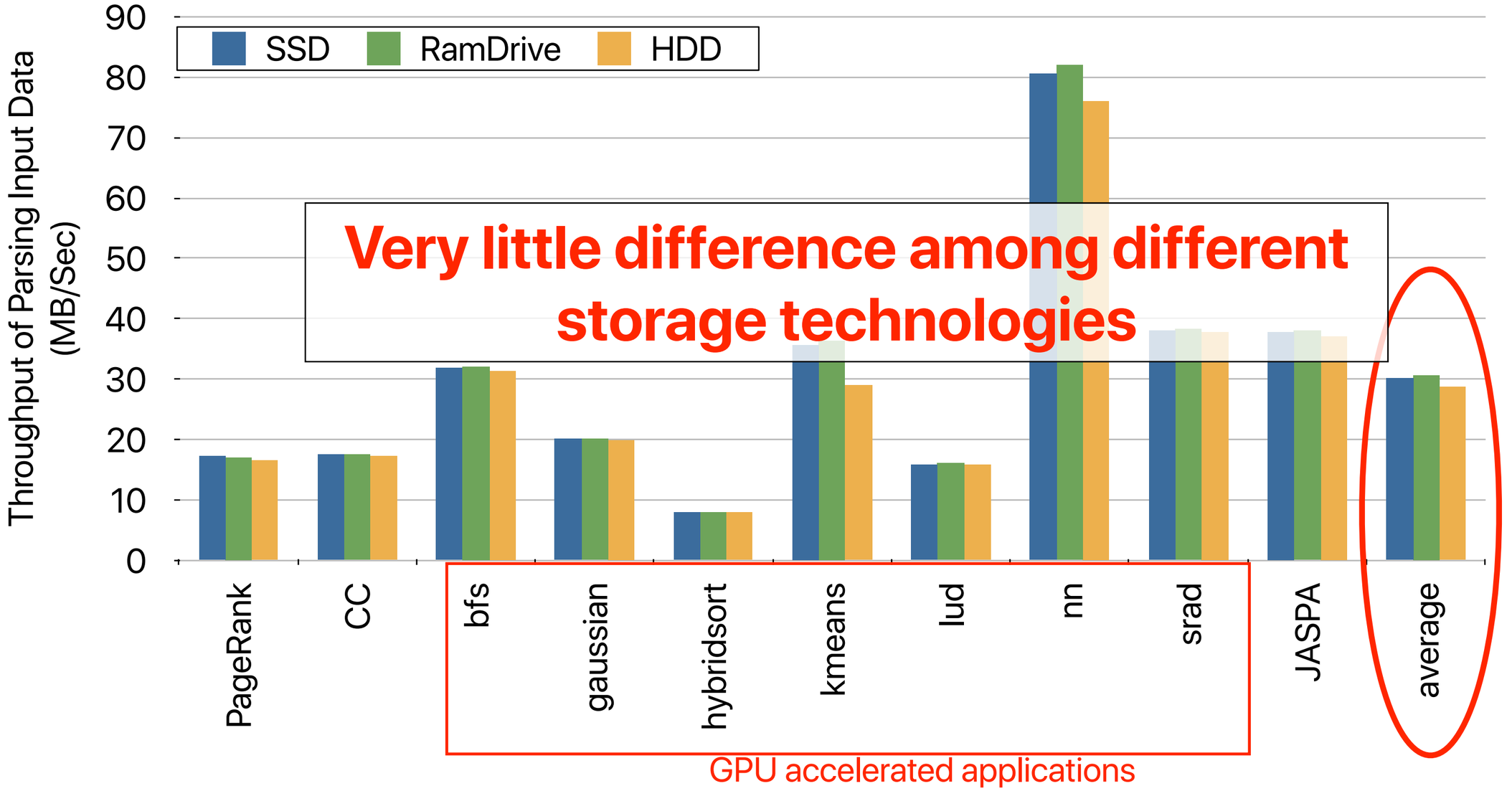
⇩

0xBC614E

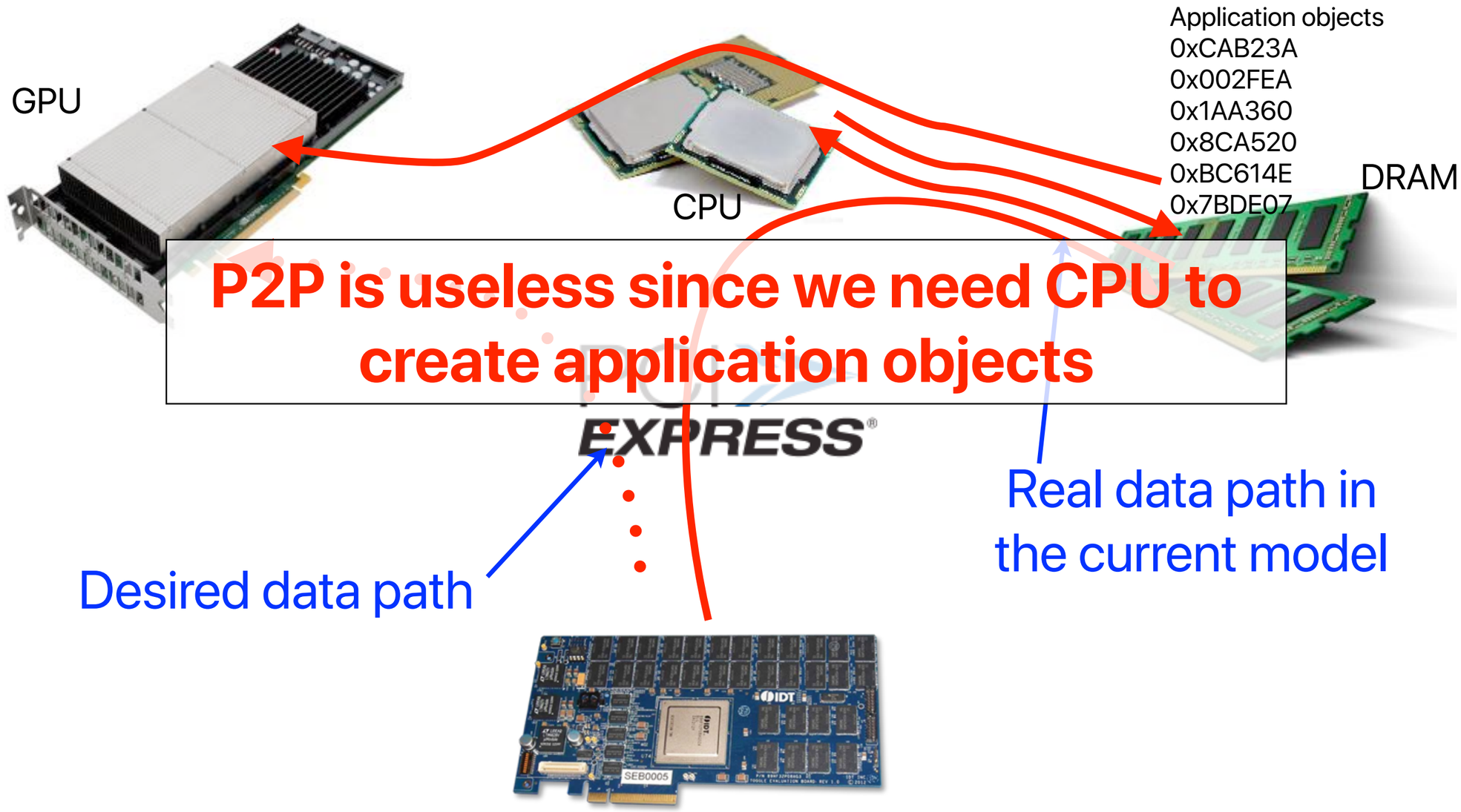
Processing “ASCII” data



High-speed storage is useless



P2P is also useless

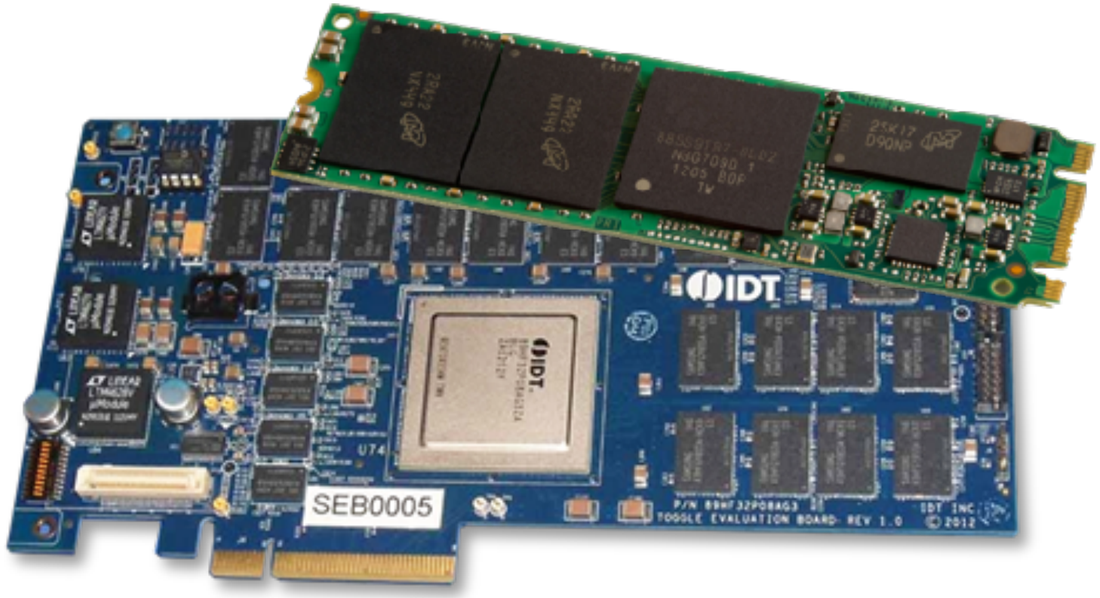


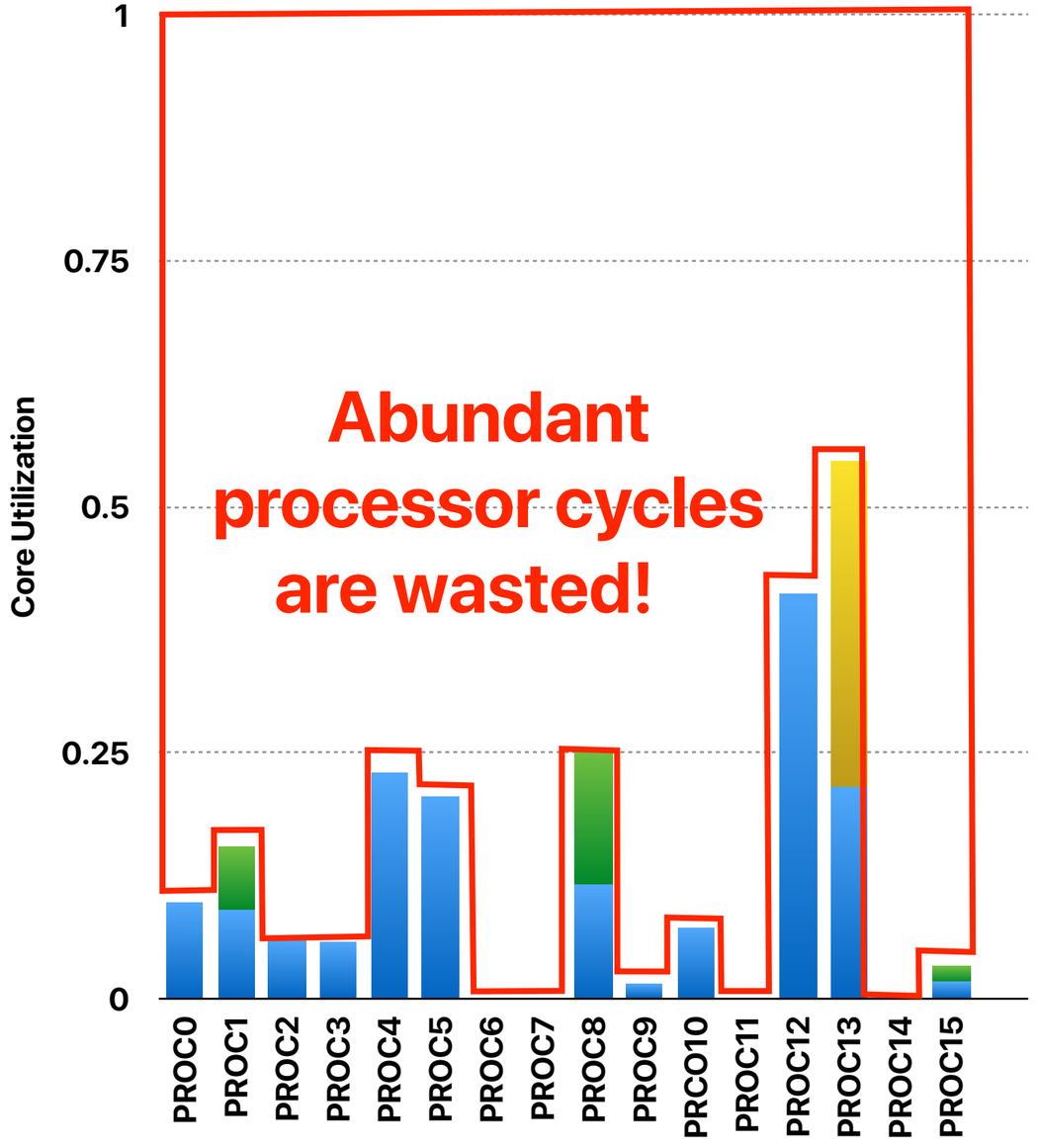
What have you learned?

- Parsing the input takes as long as the computation (or longer if your sort is optimized)
- The good-put of parsing is way lower than the I/O bandwidth
- If you don’t store data in the most appropriate format, it does not matter what kind of storage device you use

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!
- Applications cannot use full bandwidth sometimes

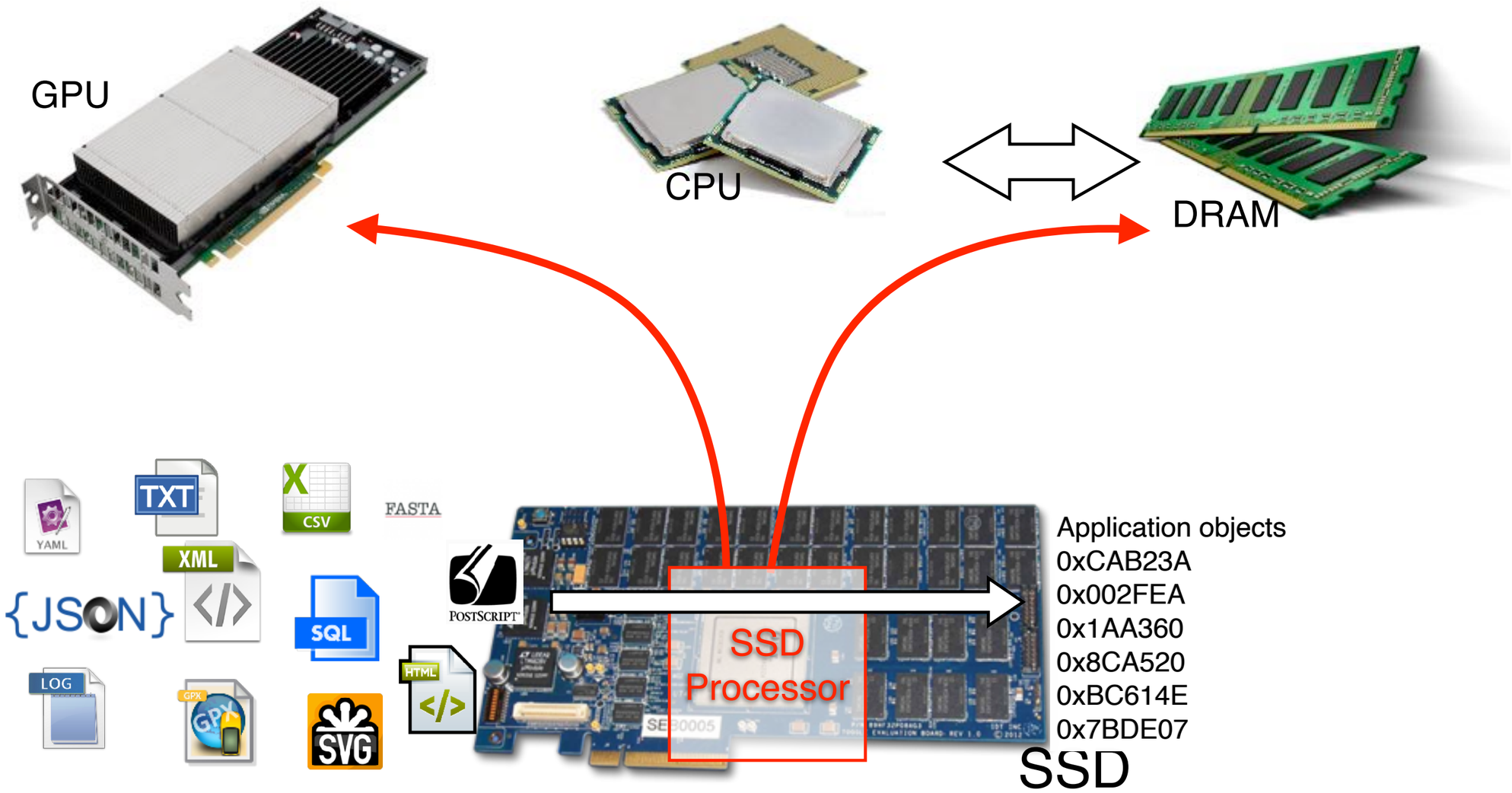




Example — Morpheus✳

✳ Hung-Wei Tseng, Qianchen Zhao, Yuxiao Zhou, Mark Gahagan, and Steven Swanson. 2016. Morpheus: creating application objects efficiently for heterogeneous computing. In Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16).

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

What's the benefit?

- Enabling P2P communication
- Reducing the data volume
- Lowering the energy consumption
- Bypassing host system overhead

Samsung SmartSSD

PRODUCT BRIEF

AMD XILINX

SmartSSD® COMPUTATIONAL STORAGE DRIVE

OVERVIEW

The Samsung SmartSSD computational storage drive (CSD) is the industry's first adaptable computational storage platform. It empowers a new breed of software developers to easily build innovative hardware-accelerated solutions in familiar high-level languages. The SmartSSD dramatically accelerates data intensive applications by 10X or more by pushing compute to where the data lives.

At the heart of the SmartSSD is the Xilinx Adaptive Platform, a powerful toolkit which enables customers to create customized, scalable applications to solve a broad range of data center problems.

CHALLENGE

Turning Big Data Into Fast Data

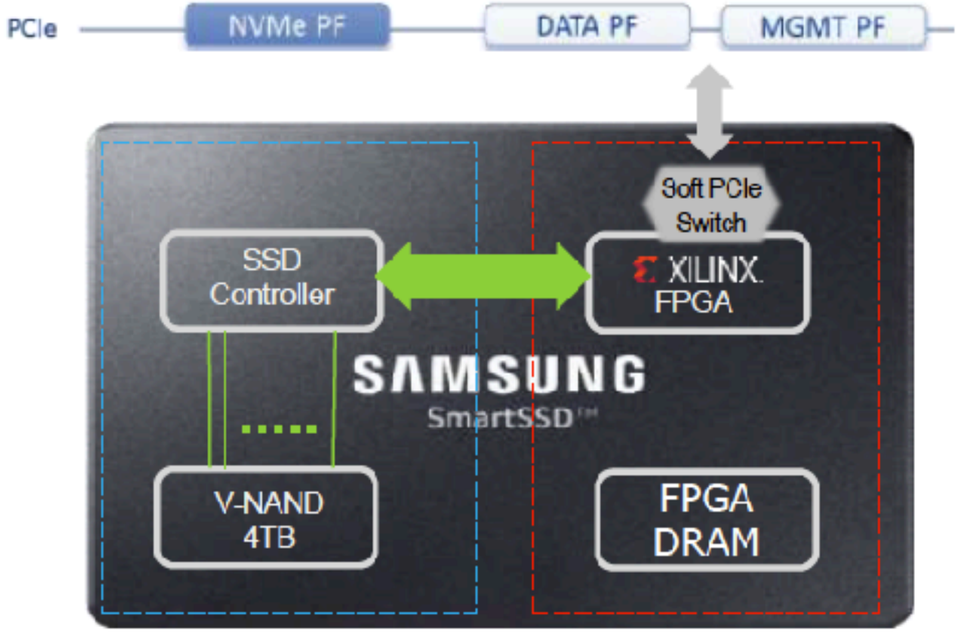
The global datasphere will grow from 45 zettabytes in 2019 to 175 by 2025, and nearly 30% of the world's data will need real- time processing. This data explosion provides tremendous opportunity for business insights, but the sheer volume

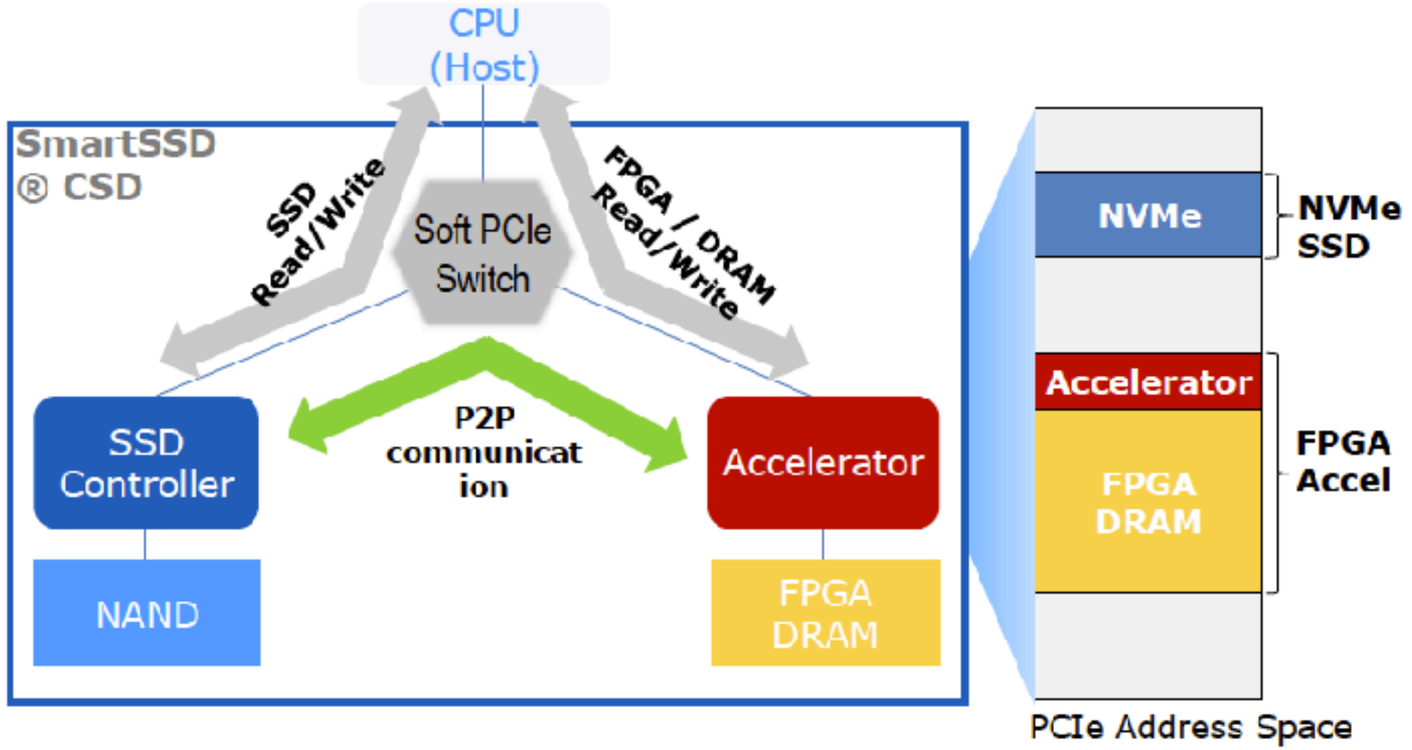
TARGET APPLICATIONS

- AI/ML Inference
- Big Data Analytics

SmartSSD® CSD HW Architecture

- Peer-to-peer (P2P) communication enables unlimited concurrency
  - SSD:Accelerator data transfers use internal data path
    - Save precious L2:DRAM Bandwidth (Compute Nodes) • Scale without costly x86 frontend (Storage Nodes)
    - Avoid the unnecessary funneling and data movement of standalone accelerators
  - FPGA DRAM is exposed to Host PCIe address space
    - NVMe commands can securely stream data from SSD to FPGA peer-to-peer





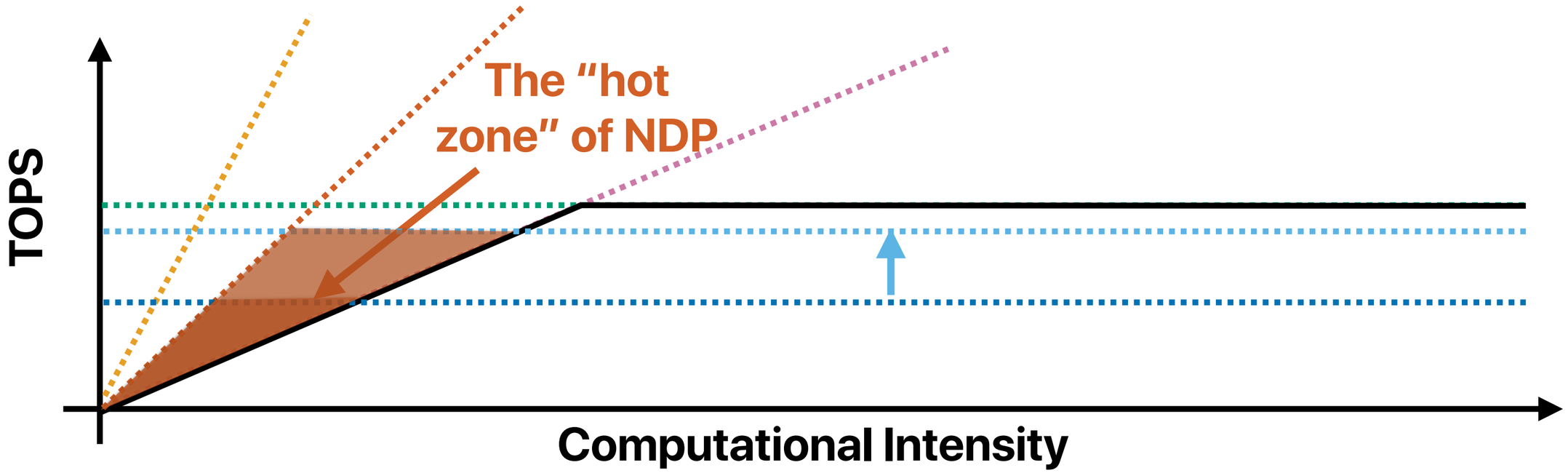
Reviewing the roofline model

Peak OPS of target computing resource

Peak memory bandwidth × computational intensity ÷ reduction of data volume

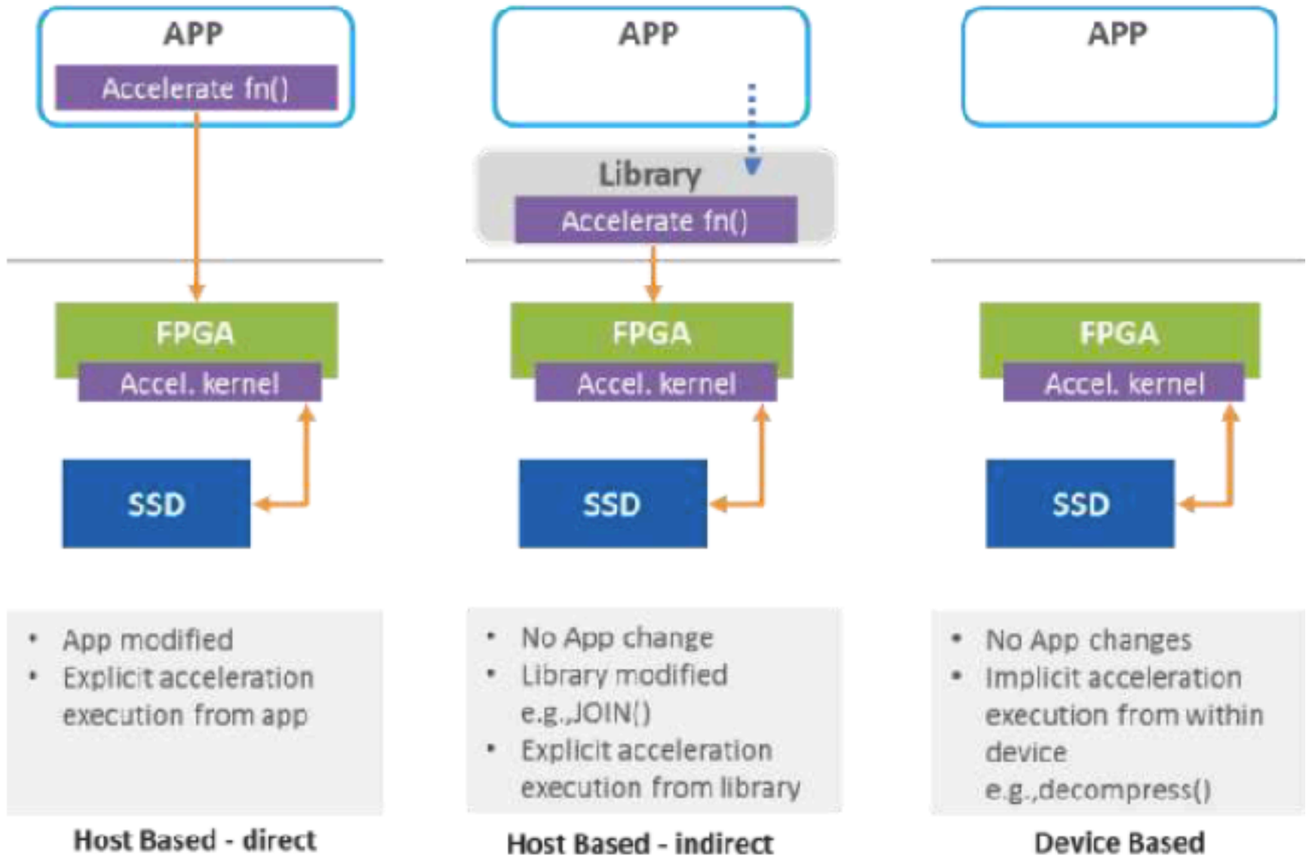
Peak OPS of target NDP device

Peak device internal bandwidth × computational intensity of NDP program



Developing on SmartSSD® CSD

- Frameworks supported by partners
  - Spark, Kafka available
  - FFmpeg coming
  - Many more in development
- Supported OSes
  - Linux
  - FreeBSD
  - Windows Server
- Ease of porting for SDAccel OpenCL developers
- Vivado-friendly for RTL developers



SmartSSD® OpenCL Programming in 5 Steps

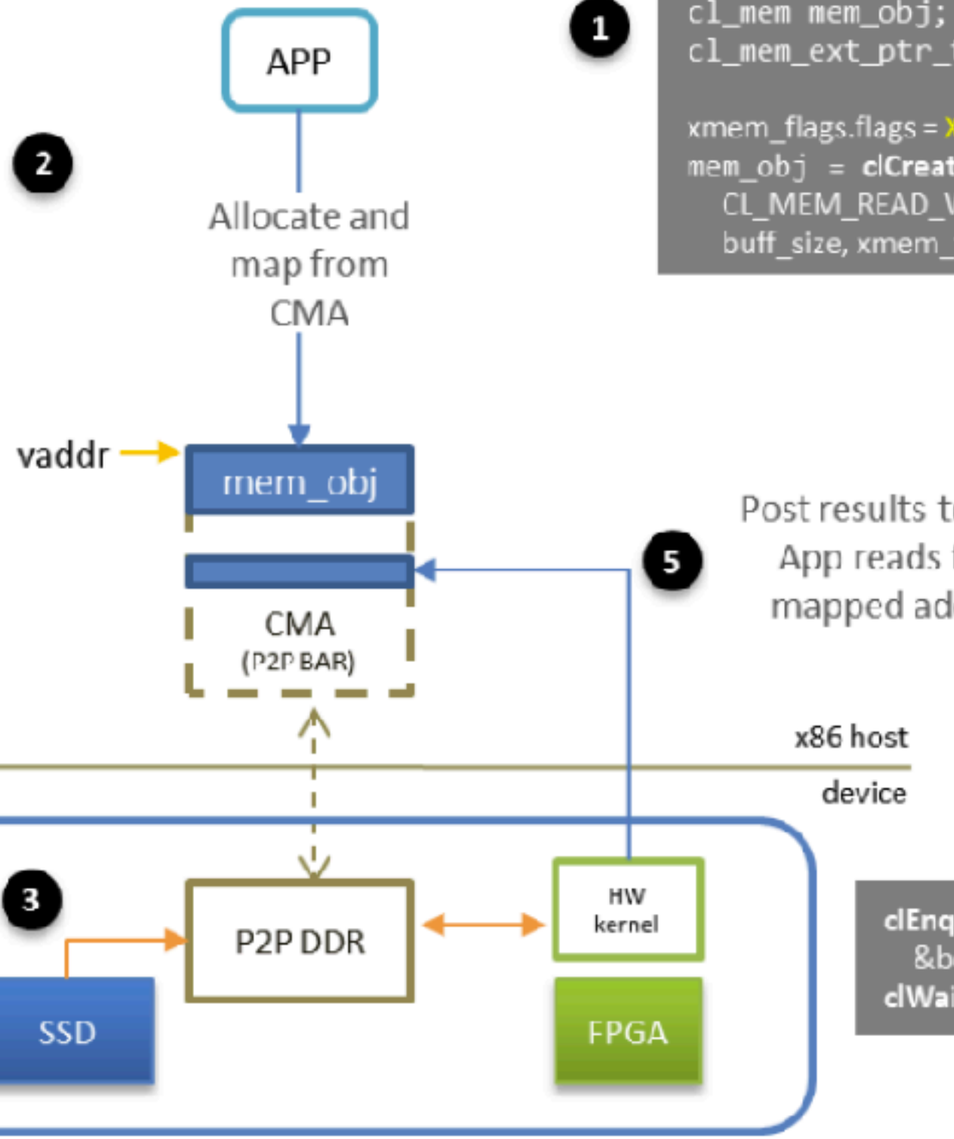
- Secure, P2P data movement
  - Data moves in/out of SSD only under control of storage stack (NVMe)

```
cl_mem mem_obj;
cl_mem_ext_ptr_t xmem_flags = {0};

xmem_flags.flags = XCL_MEM_EXT_P2P_BUFFER;
mem_obj = clCreateBuffer(dev_cxt,
  CL_MEM_READ_WRITE | CL_MEM_EXT_PTR_XILINX,
  buff_size, xmem_flags, &err);
```

1

```
void *vaddr;

vaddr = clEnqueueMapBuffer(cmd_queue, cl_buff,
  CL_TRUE, host_access, 0, buff_size, 0, NULL, NULL,
  &err);
```

2




```
bytes_read = read(fd, vaddr, buff_size);
```

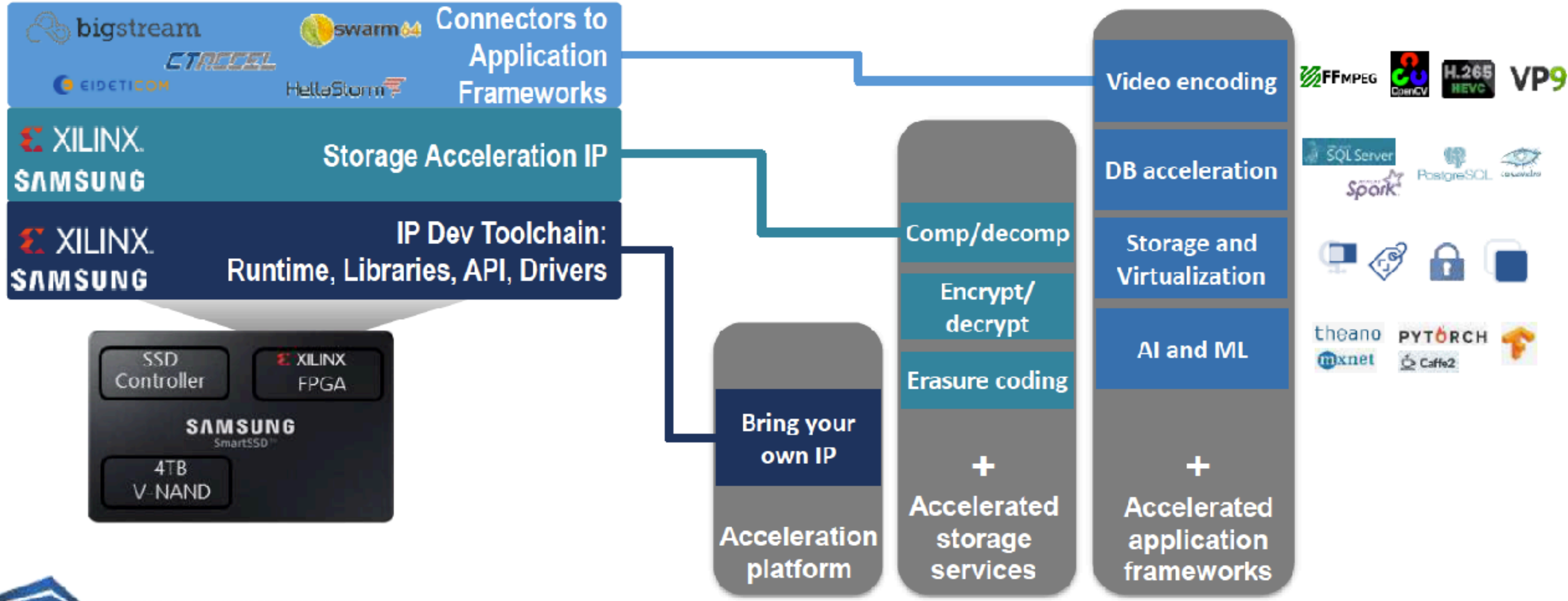
Initiate peering transfers from SSD

4

```
clEnqueueTask(cmd_queue, hw_kernel, 0, NULL,
  &bc_event);
clWaitForEvents(1, &bc_event);
```

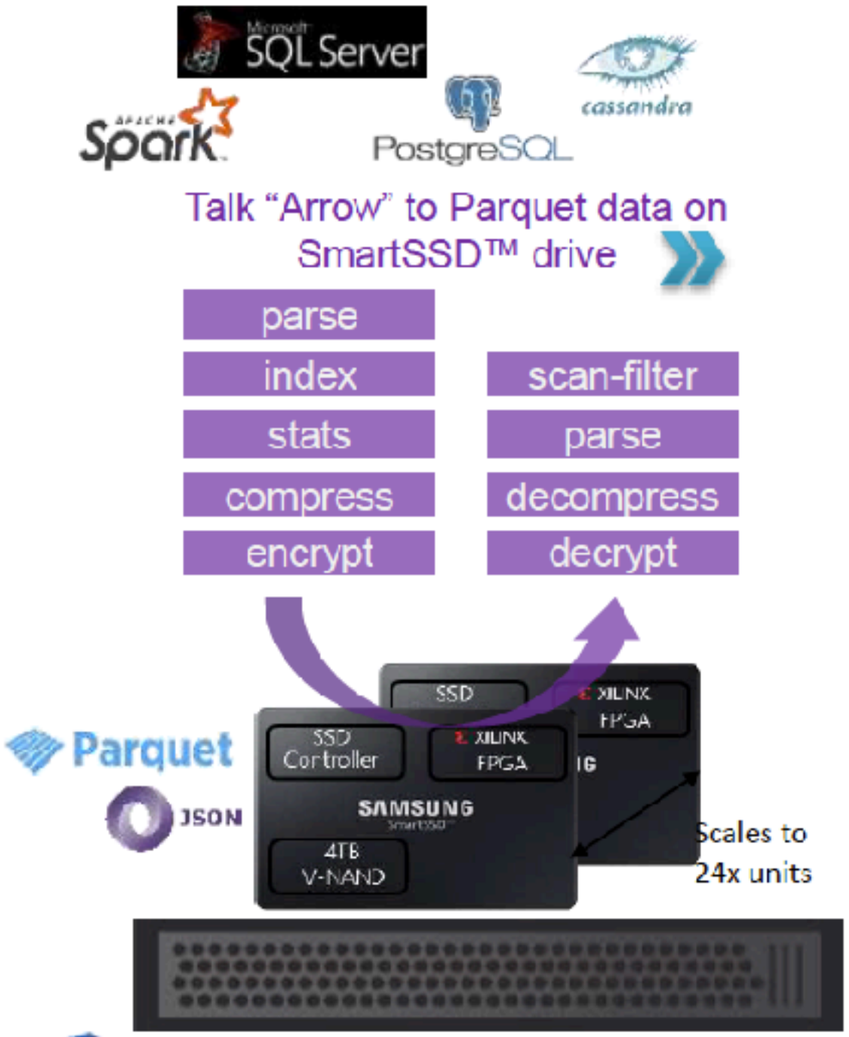
SmartSSD® CSD Use Cases and Ecosystem

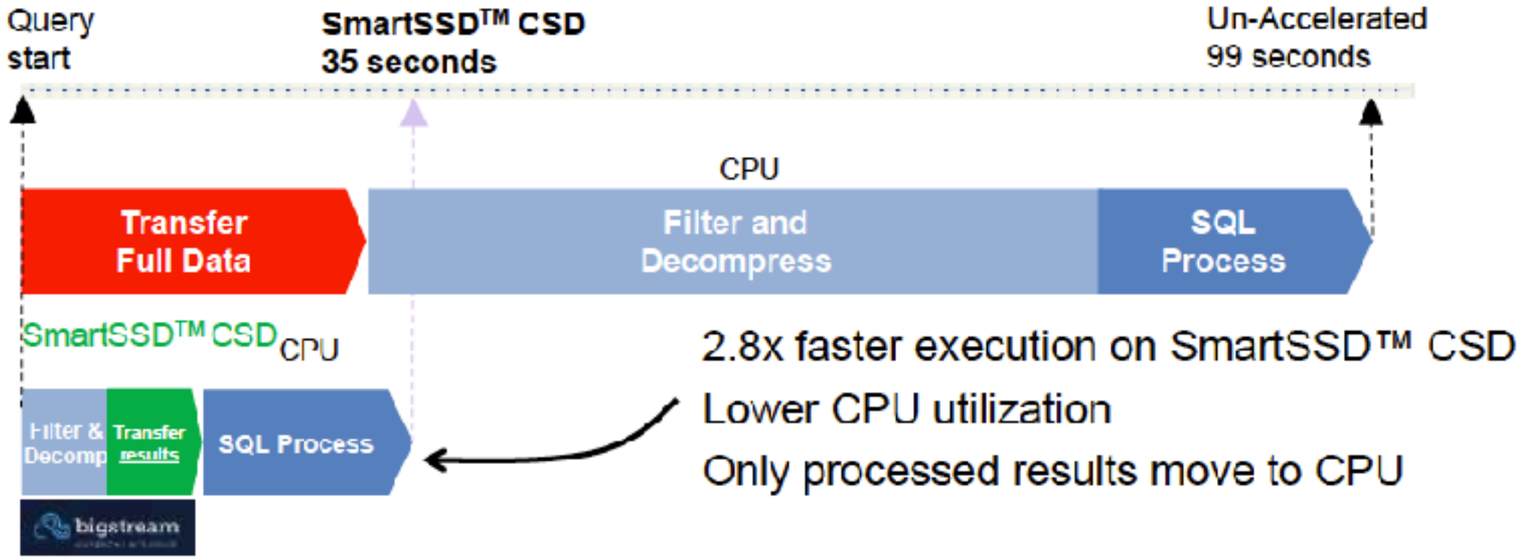
- **Storage Services:** Comp/Decomp, Encryp/Decrypt, Metadata management, Erasure Coding,
- **Real-time Analytics & Biz Intelligence:** DB Query (Spark, PostgreSQL, ..), Log analytics, genomics, physics
- **Rich Media and ML:** transcoding, live streaming, object detection

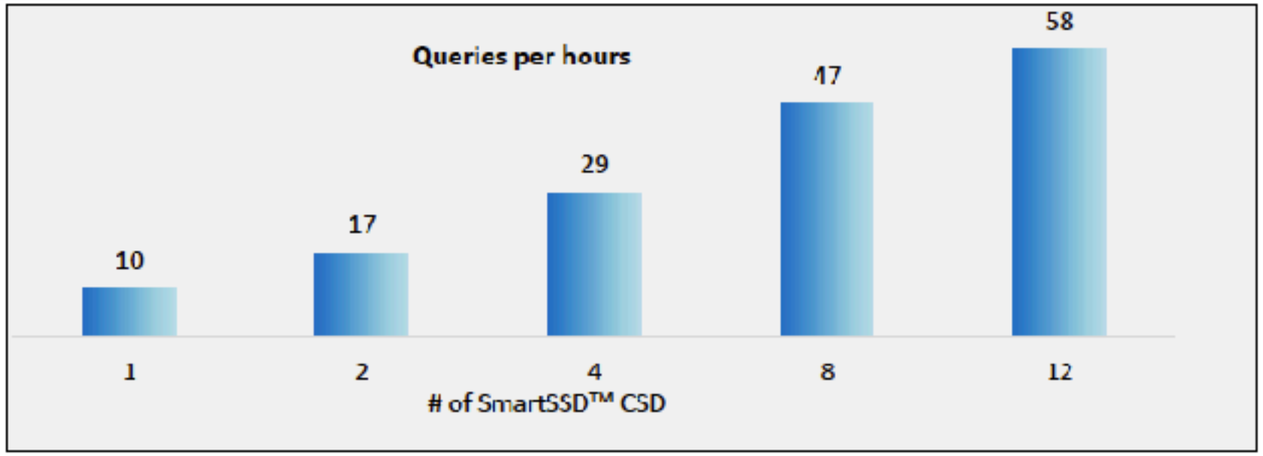


SmartSSD® CSD accelerates DB and Analytics

- **Scale to larger data sets with fewer servers**







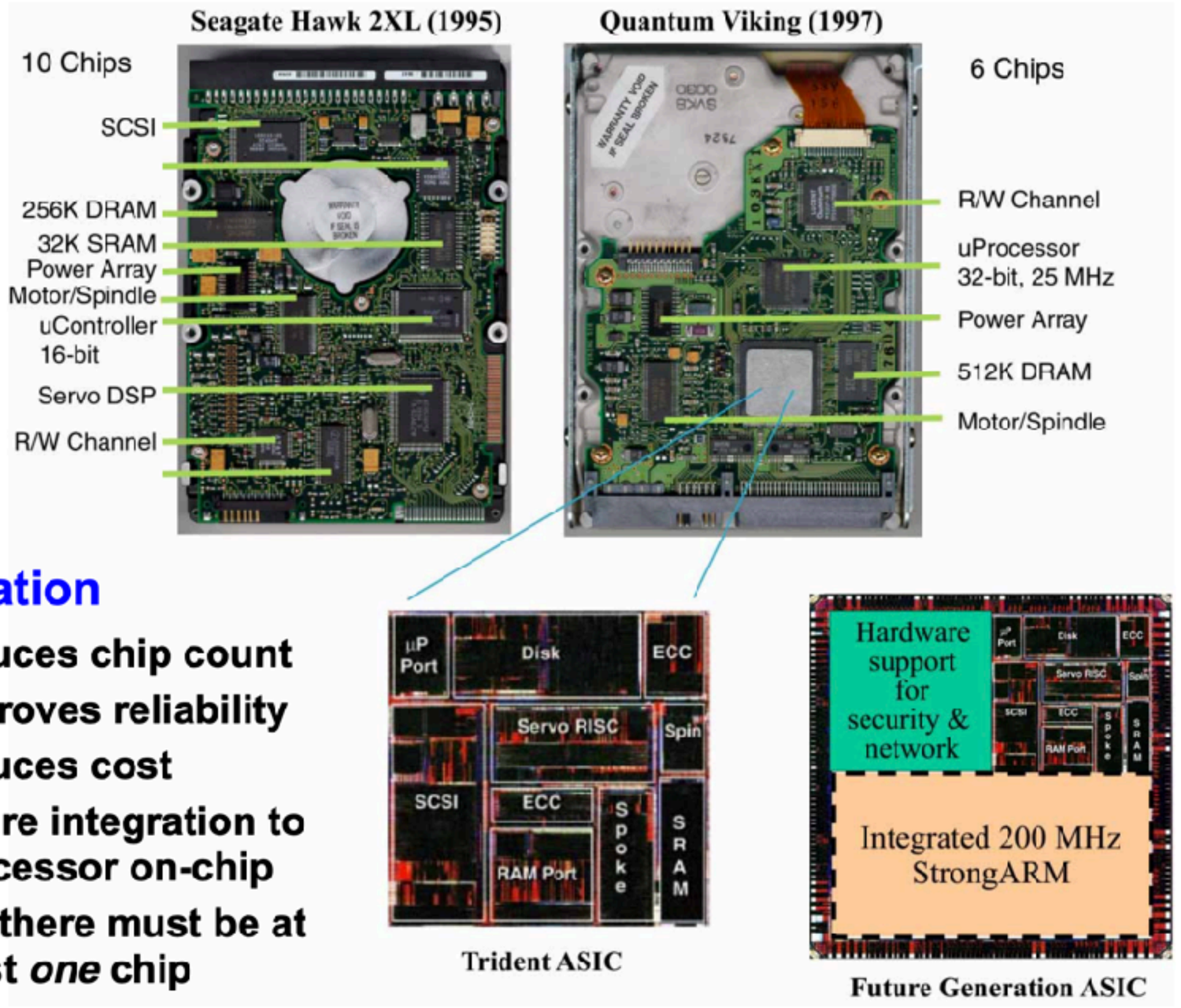
Performance scales with each SmartSSD™ CSD

In-storage processing isn't a new idea…

- Active Disks: Anurag Acharya, Mustafa Uysal, and Joel Saltz. 1998. Active disks: programming model, algorithms and evaluation. In Proceedings of the eighth international conference on Architectural support for programming languages and operating systems (ASPLOS VIII). Association for Computing Machinery, New York, NY, USA, 81–91.

http://www.cs.cmu.edu/~riedel/defense/talk99_ActiveDisksForDatabases.pdf

Evolution of Disk Drive Electronics

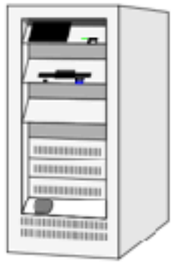


http://www.cs.cmu.edu/~riedel/defense/talk99_ActiveDisksForDatabases.pdf

Opportunity

TPC-D 300 GB Benchmark, Decision Support System

Database Server

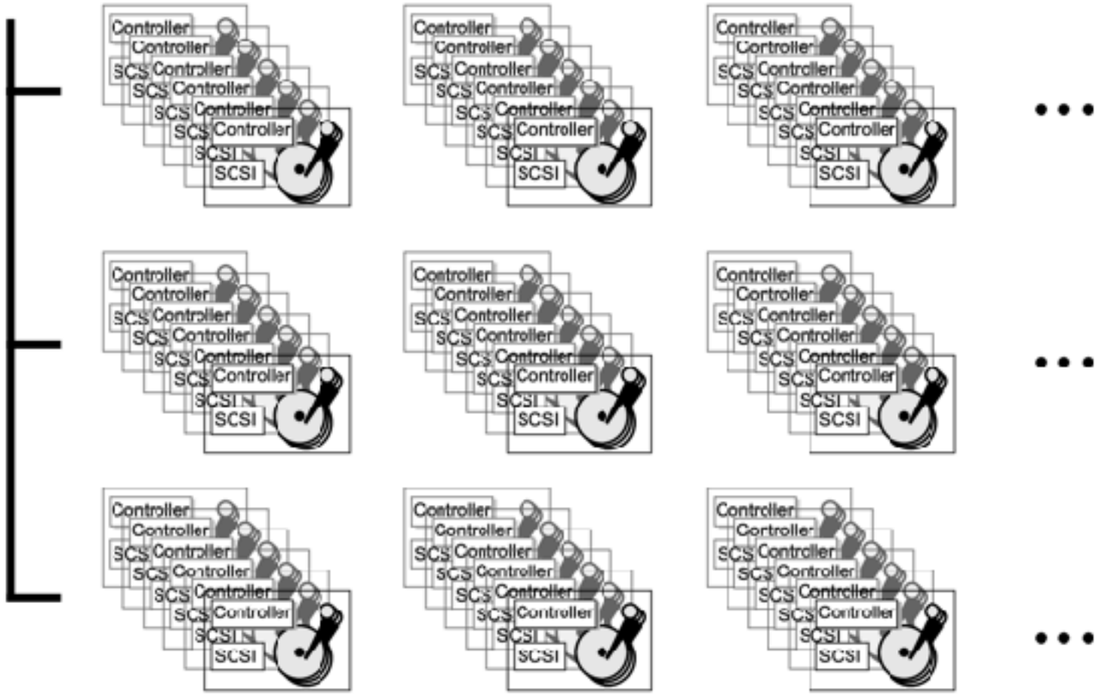


Digital AlphaServer 8400 = 7,344 total MHz
- 12 x 612 MHz 21164
- 8 GB memory
- 3 64-bit PCI busses — *3 x 266 = 798 MB/s*
- 29 FWD SCSI controllers — *29 x 40 = 1,160 MB/s*

Storage
- 520 rz29 disks
- 4.3 GB each
- 2.2 TB total

= 104,000 total MHz
(with 200 MHz drive chips)

= 5,200 total MB/s
(at 10 MB/s per disk)



3:00

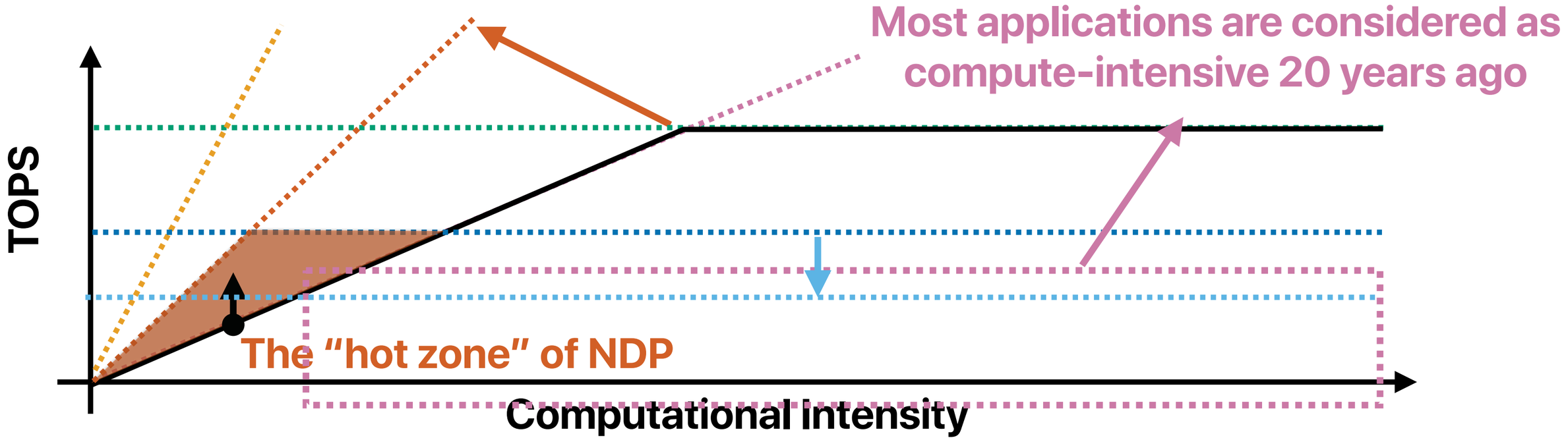
Why “active disks” did not work out?

Why Active Disks did not work out?

- Computation still dominates at that era
- Magnetic disks do not really have too much to do with “internal bandwidth”

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)



3:00

Are you fascinated by in-storage processing? Why or why not? What do you have in mind to make ISP more attractive?

Limitations of in-storage processing

- Programming
- Processor capabilities
  - Only gives you “moderate” performance gain, not orders-of-magnitude

References

- SmartSSDs
  - Query Processing: Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. 2013. Query processing on smart SSDs: opportunities and challenges. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13). Association for Computing Machinery, New York, NY, USA, 1221–1230.
  - MapReduce: Kang, Yangwook, Yang-suk Kee, Ethan L. Miller, and Chanik Park. "Enabling cost-effective data processing with smart SSD." In 2013 IEEE 29th symposium on mass storage systems and technologies (MSST), pp. 1-12. IEEE, 2013.
- BlueDBM: Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. 2015. BlueDBM: an appliance for big data analytics. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15). Association for Computing Machinery, New York, NY, USA, 1–13. DOI:https://doi.org/10.1145/2749469.2750412
- Biscuit: A Framework for Near-Data Processing of Big Data Workloads
  - Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: a framework for near-data processing of big data workloads. In Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)
- Summarizer
  - Gunjae Koo, Kiran Kumar Matam, Te I, H. V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: trading communication with computing near storage. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17).

Electrical
Computer Science
Engineering

277

つづく