What you store matters

Hung-Wei Tseng

Recap: The "Winning Formula" of In-Storage Processing

Conventional Model

ISP Model

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

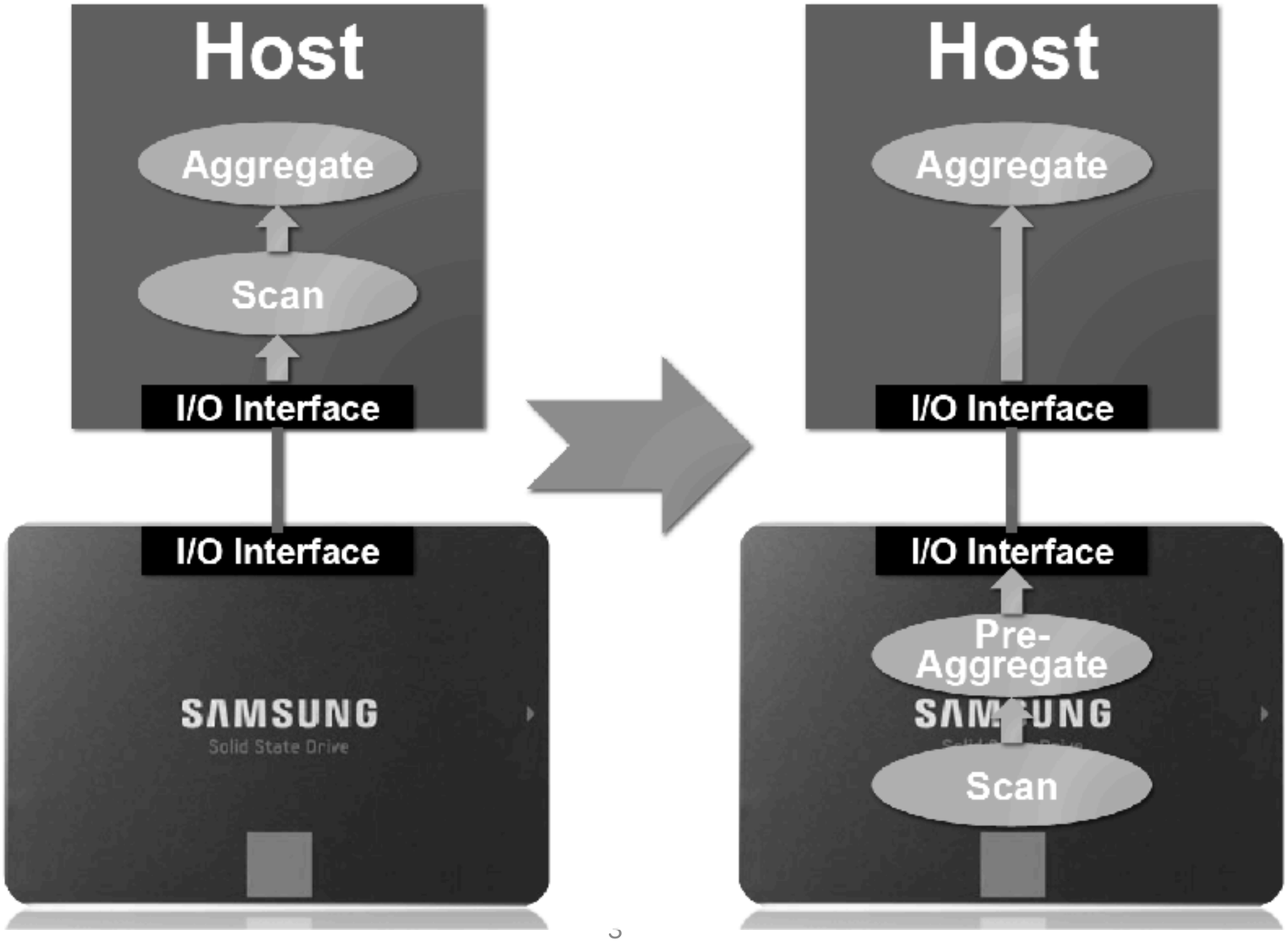
The ISP processor must have efficient internal access to the device

The ISP computation cannot take too long

The ISP should reduce data volume

The ISP should offload the processor

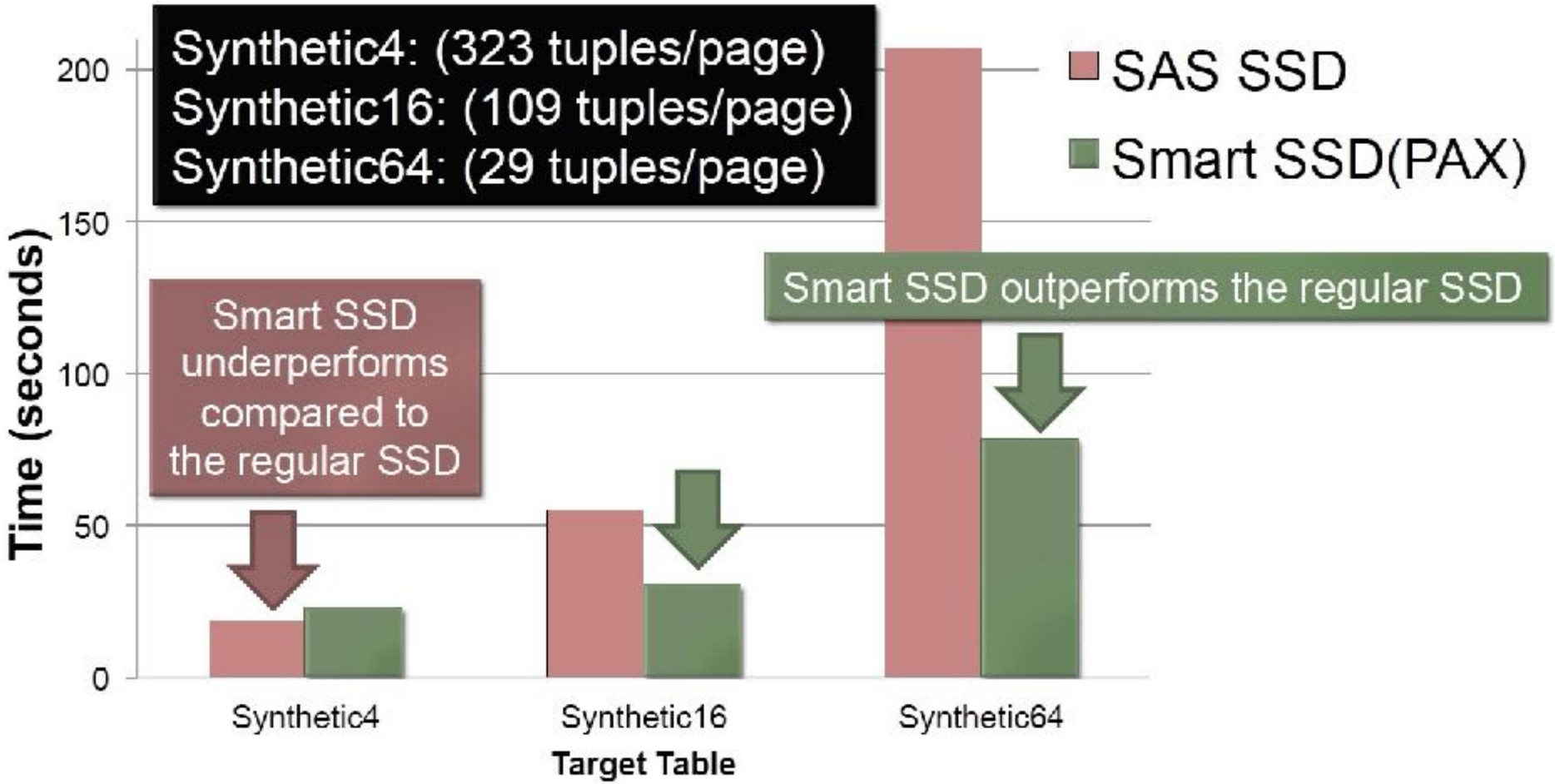
Recap: SmartSSDs



Query Processing on Smart SSDs , SIGMOD 2013

Effect of the # of Tuples on a Page

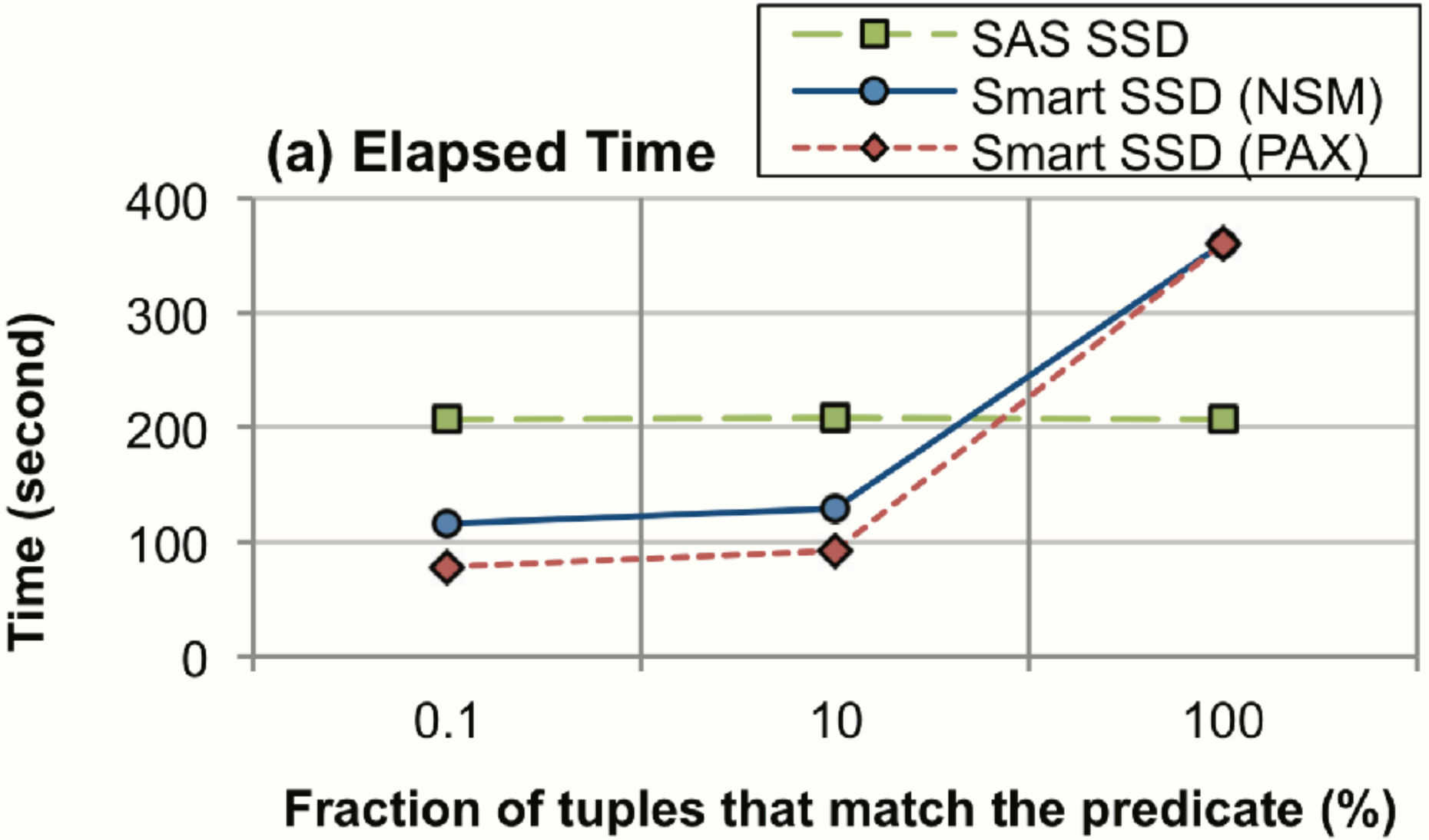
```
SELECT COLUMN_2, FROM [Synthetic_table] WHERE COLUMN_1 < [VALUE]
```



Benefit depends on the dataset size

Query Processing on Smart SSDs , SIGMOD 2013

SSD becomes a bottleneck

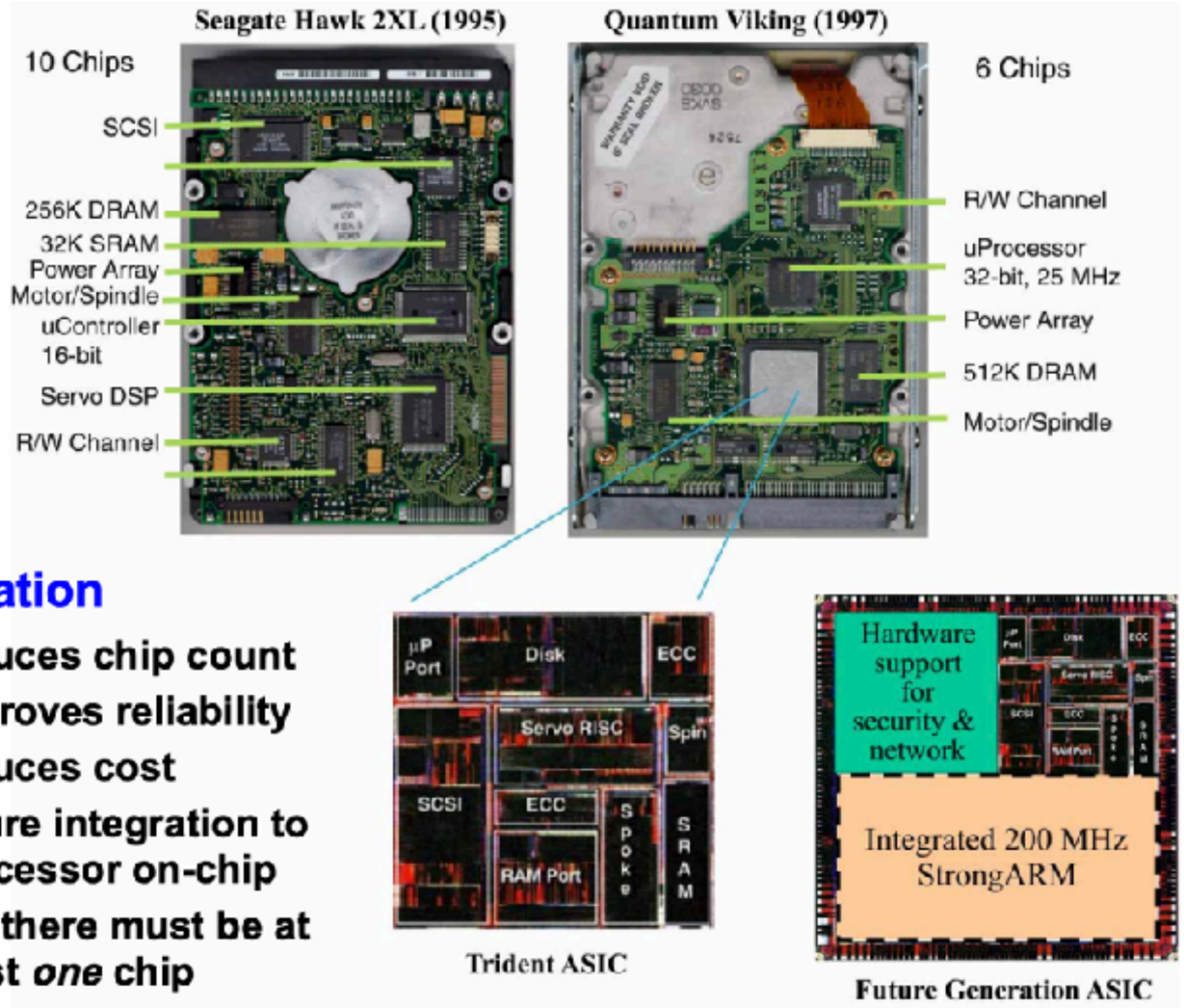


Memory and compute in SSD is a bottleneck

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

Recap: Active Disks

Evolution of Disk Drive Electronics



Recap: 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”
- Independent advances like distributed storage is similar to active disk
- New features at added cost may not be used by many and convincing system vendors to implement support is hard
- Business model — hard disk drives focus on low cost per unit of data

Recap: Challenges of ISP

- Programming is still an issue
  - Do you want to program at the level of FPGAs?
- Applications
  - Low compute intensity
  - Volume reduction after pre-processing
- Hardware capability and cost
  - What kind of processors can we have in the storage device?

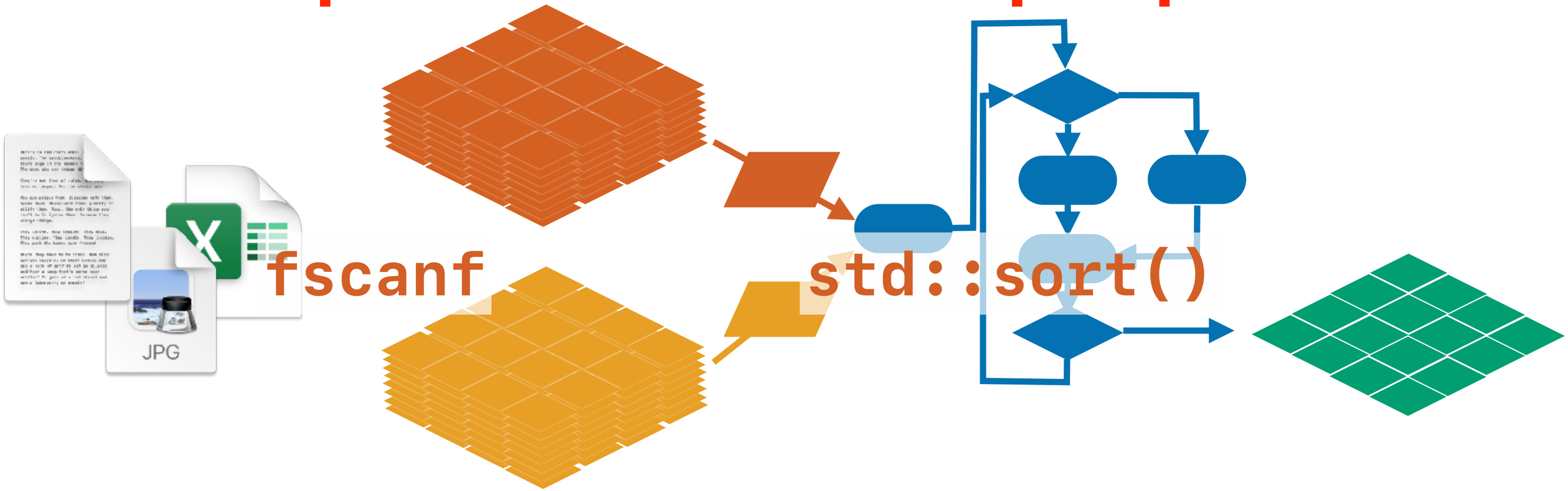
Case: where do you expect the program to spend the most time?

3:00

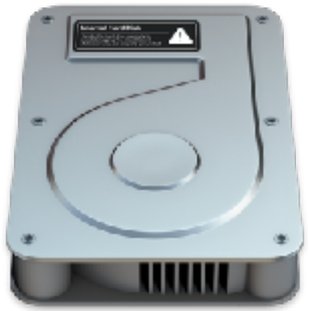
```cpp
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
#include <ctime>
#include <iostream>
#include <climits>
#include <sys/time.h>

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;
}
```

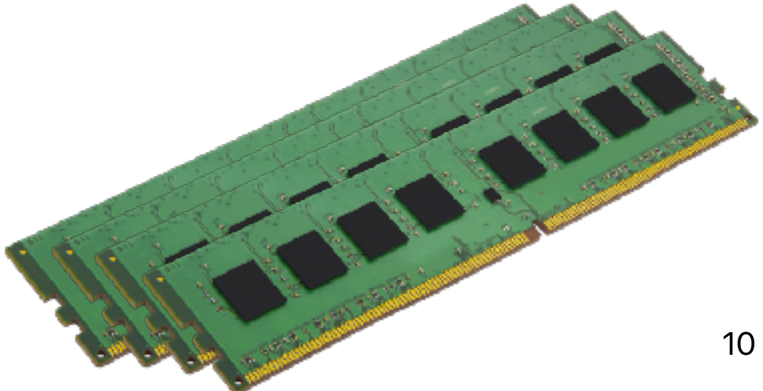
https://github.com/hungweitseng/EE277/blob/main/demo/sort/arraySort.cpp

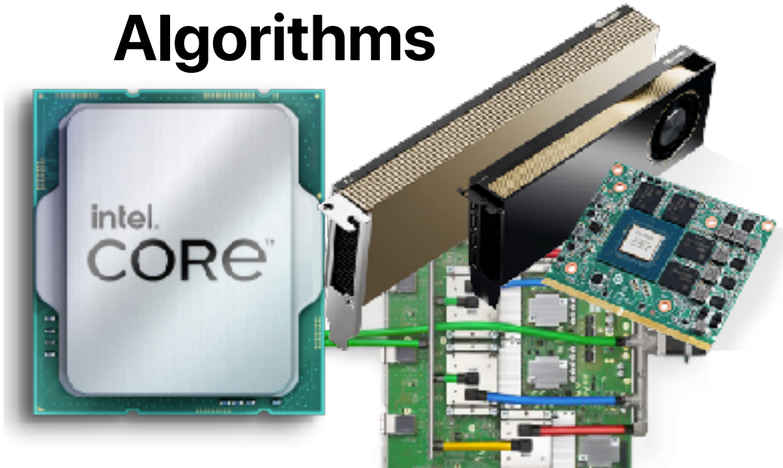
Recap: From the software perspective



Source “data" **“Data” structures** **Algorithms** **Result**







ASCII v.s. Binary

ASCII

“12345678”

⇩

0x3231 0x3433 0x3635 0x3837 0x000a

Binary

12345678

⇩

0xBC614E

What's the arithmetic intensity?

```c
int string_to_int(char *str)
{
    int num = 0;

    // converting string to number
    for (int i = 0; str[i] != '\0'; i++) {
        num = num * 10 + (str[i] - 48);
    }

    // at this point num contains the converted number
    printf("%d\n", num);
    return 0;
}
```

Arithmetic intensity == 3

https://www.geeksforgeeks.org/convert-string-to-int-in-c/

What's the arithmetic intensity?

```cpp
#include <bits/stdc++.h>
using namespace std;

int partition(int arr[],int low,int high)
{
  //choose the pivot

  int pivot=arr[high];
  //Index of smaller element and Indicate
  //the right position of pivot found so far
  int i=(low-1);

  for(int j=low;j<=high;j++)
  {
    //If current element is smaller than the pivot
    if(arr[j]<pivot)
    {
      //Increment index of smaller element
      i++;
      swap(arr[i],arr[j]);
    }
  }
  swap(arr[i+1],arr[high]);
  return (i+1);
}
```

```cpp
// The Quicksort function Implement

void quickSort(int arr[],int low,int high)
{
  // when low is less than high
  if(low<high)
  {
    // pi is the partition return index of pivot

    int pi=partition(arr,low,high);

    //Recursion Call
    //smaller element than pivot goes left and
    //higher element goes right
    quickSort(arr,low,pi-1);
    quickSort(arr,pi+1,high);
  }
}
```

Arithmetic intensity == $\frac{lg(n) \times n}{n \times 4\ Bytes} = \frac{lg(n)}{4}$

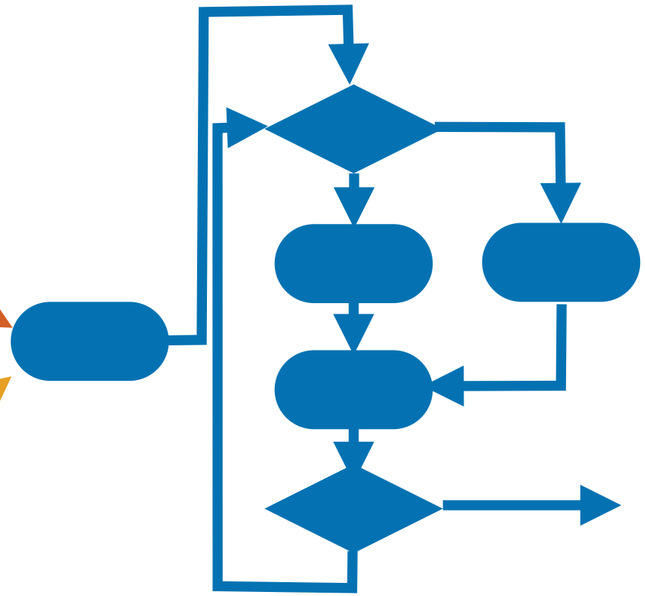
https://www.geeksforgeeks.org/convert-string-to-int-in-c/

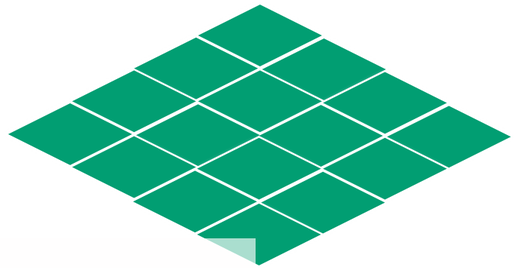
Recap: From the software perspective

In-Storage Processing

In-memory processing

How about this process?

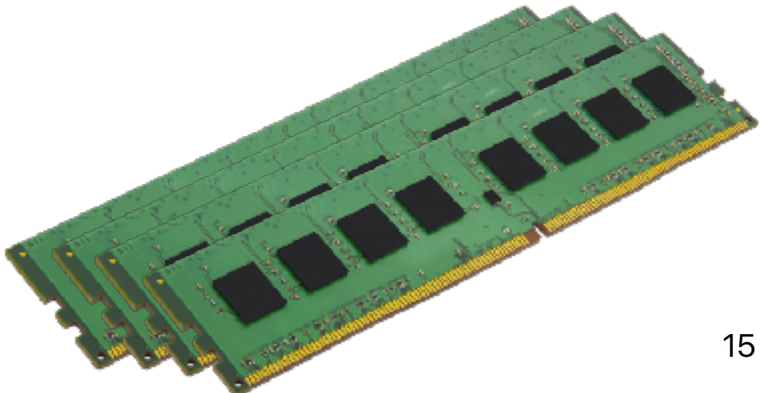




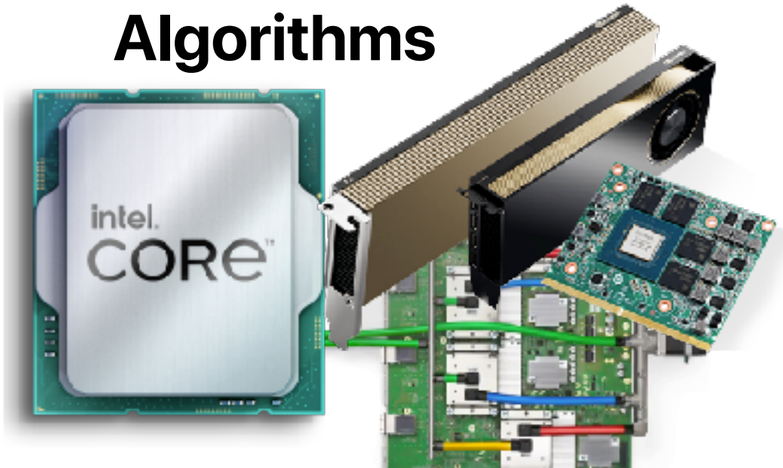
Hardware Accelerators

Source “data"

“Data” structures



Algorithms



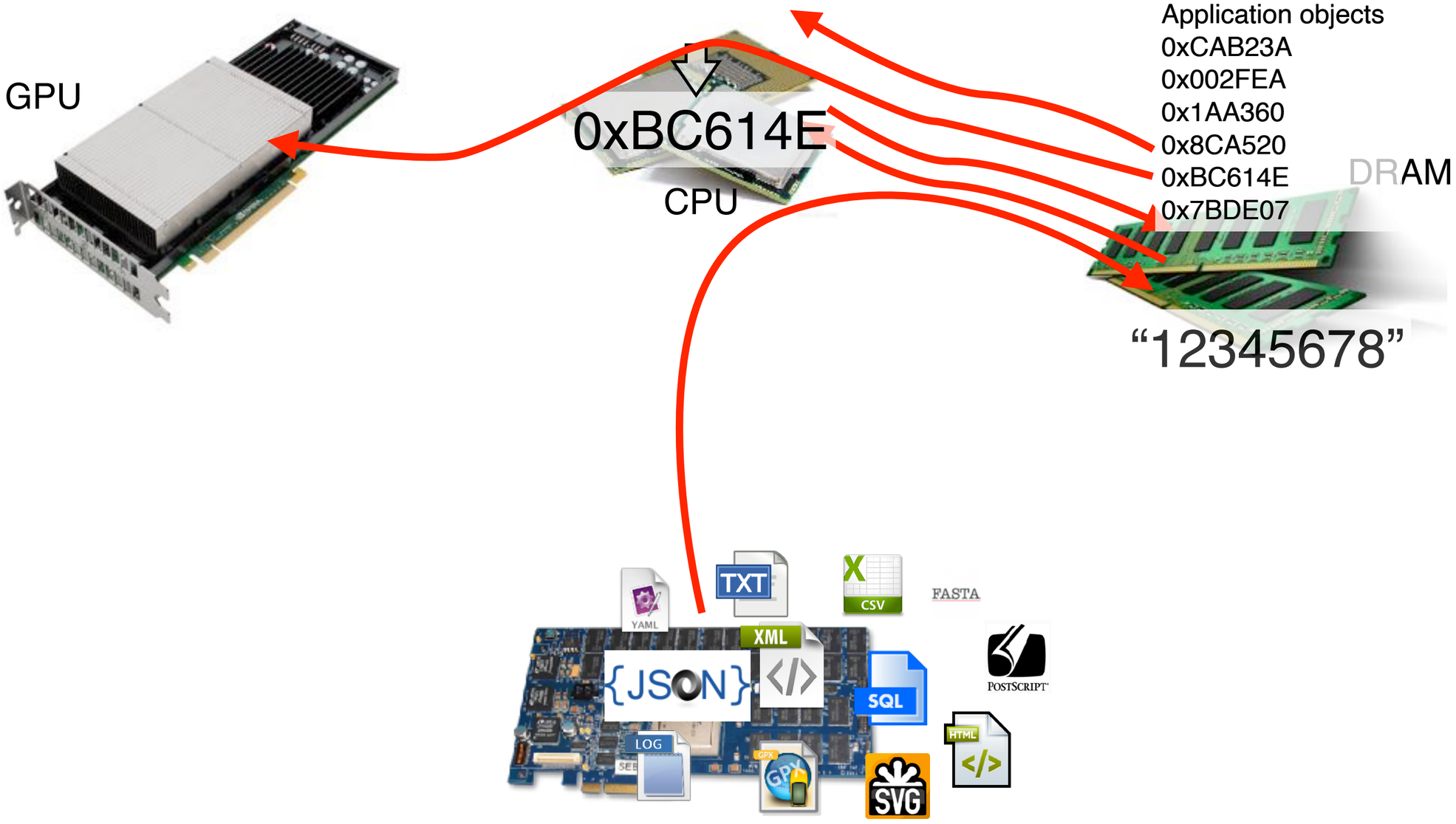
Result

3:00

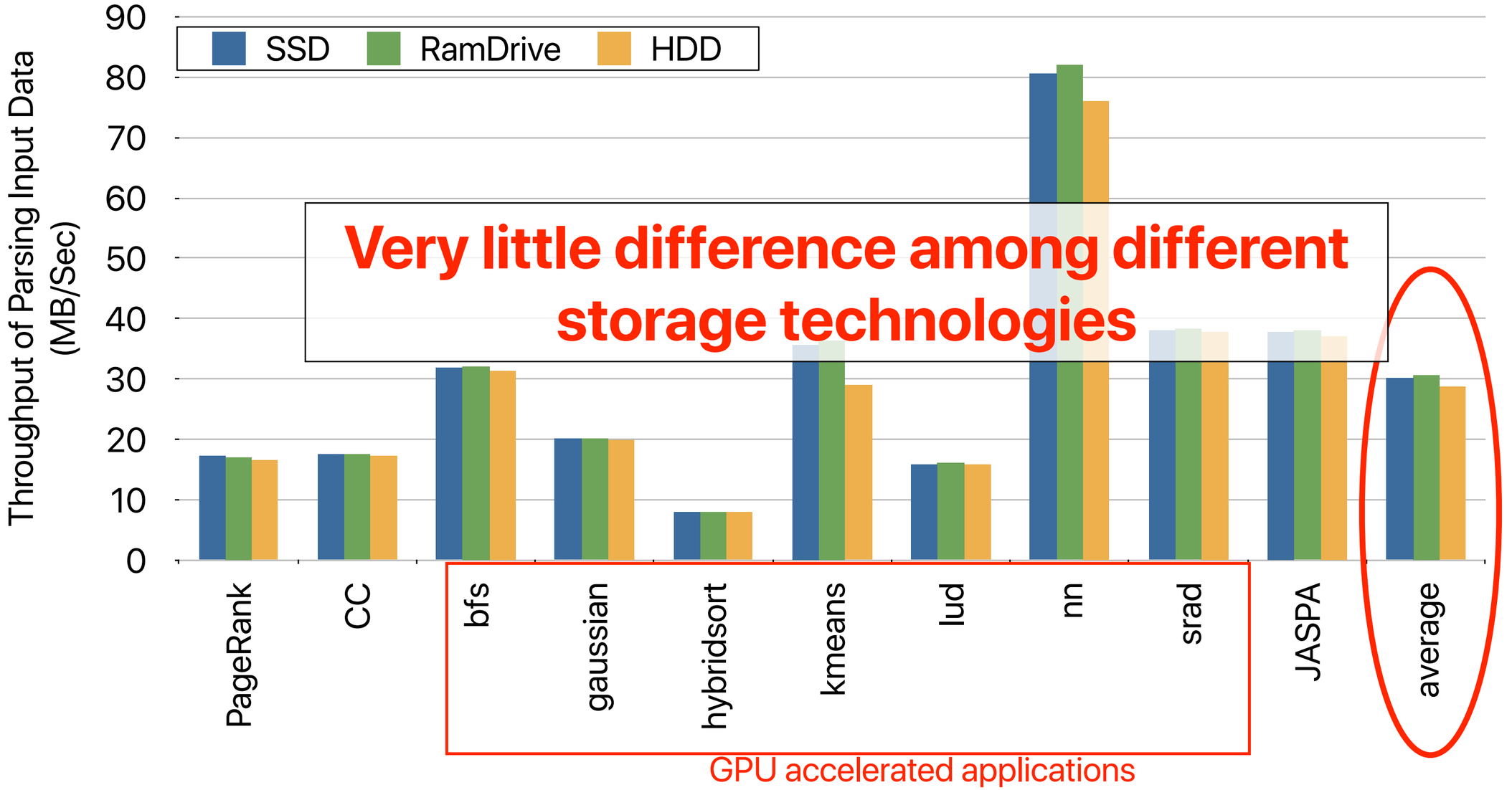
Can you identify a few file formats? What're they used for?

There are lots of data formats

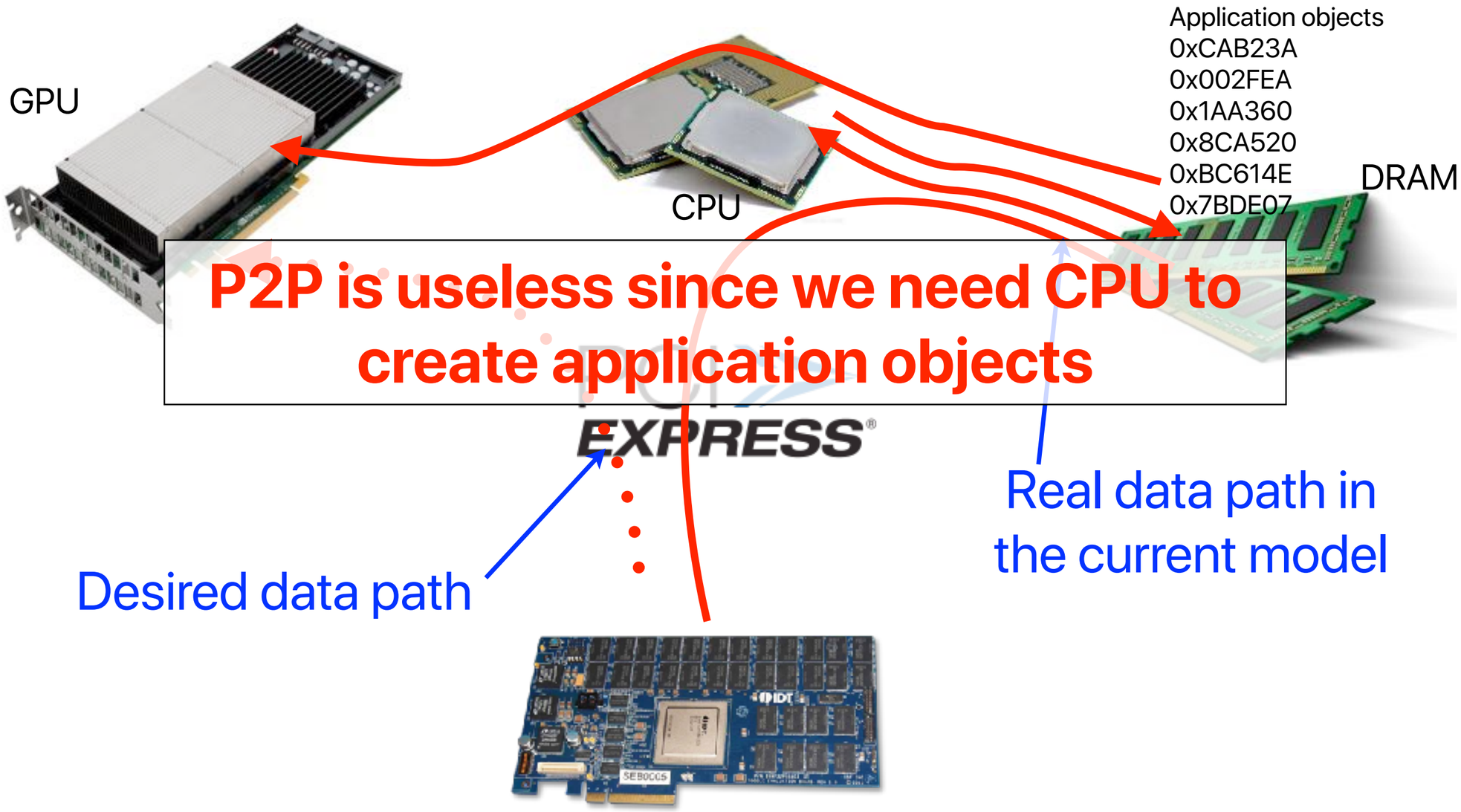
Processing “ASCII” data



High-speed storage is useless



P2P is also useless

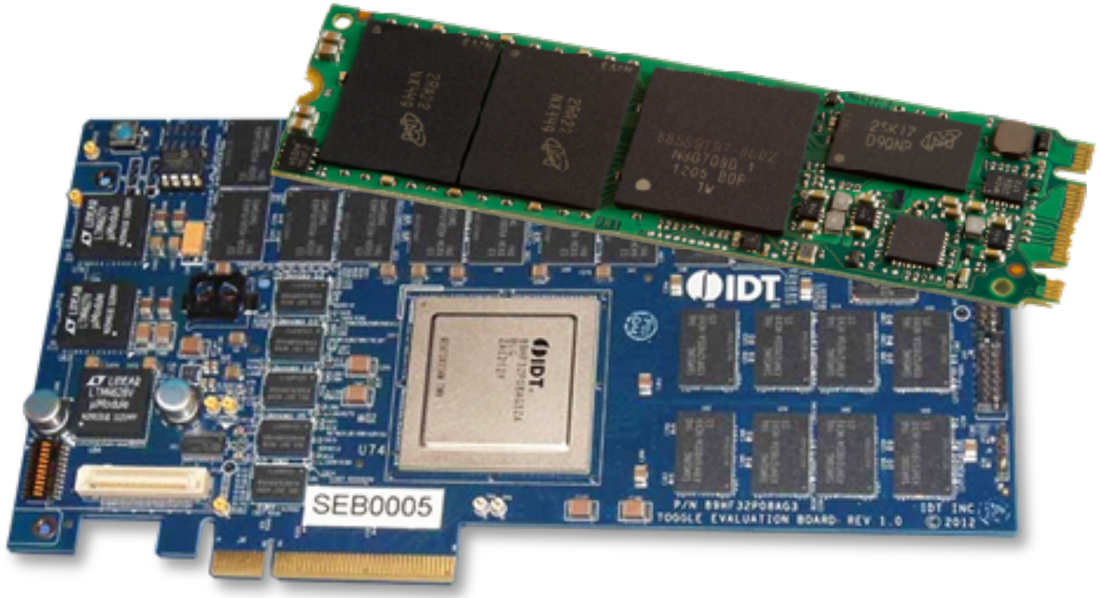


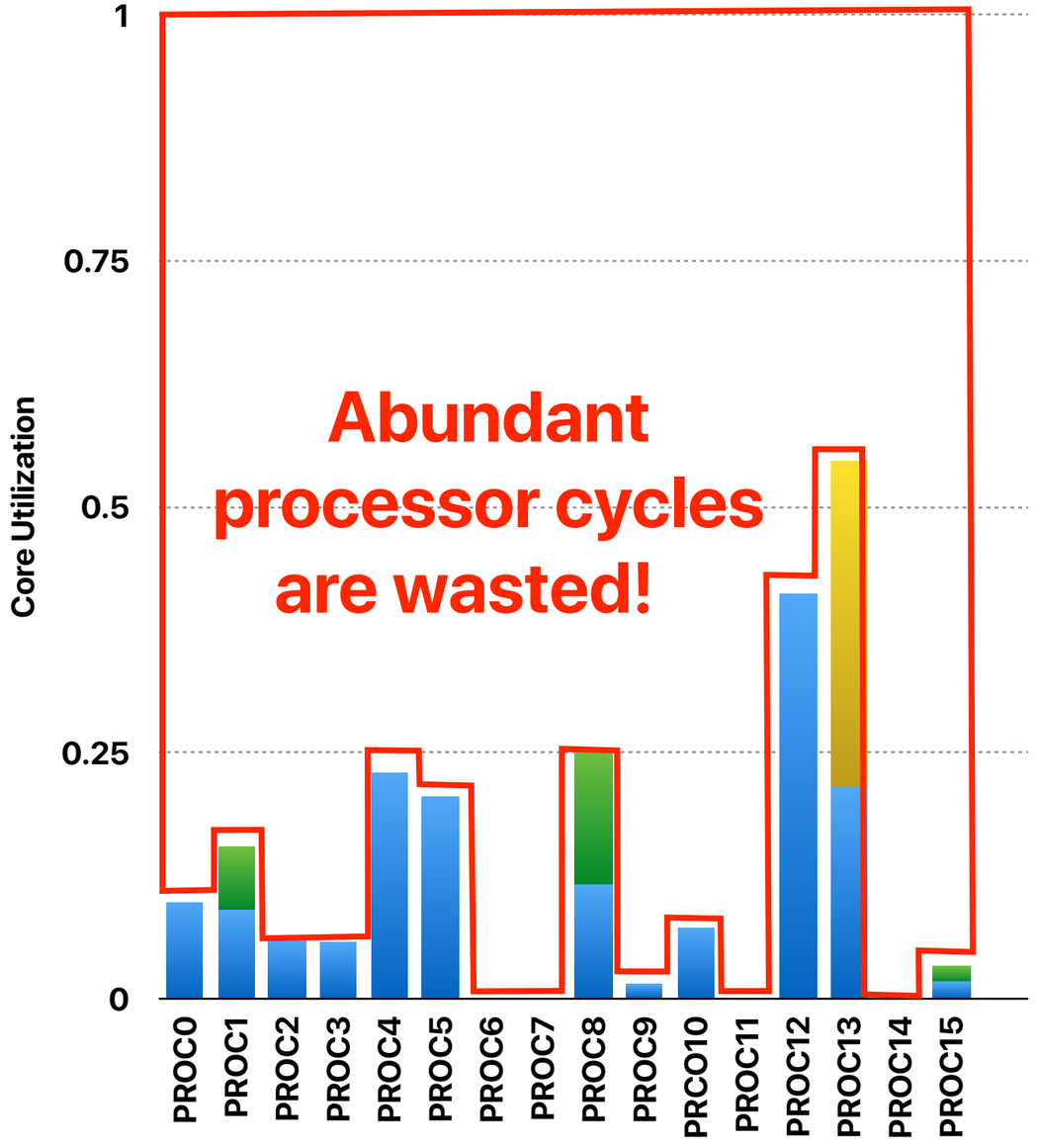
What have you learned?

- 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

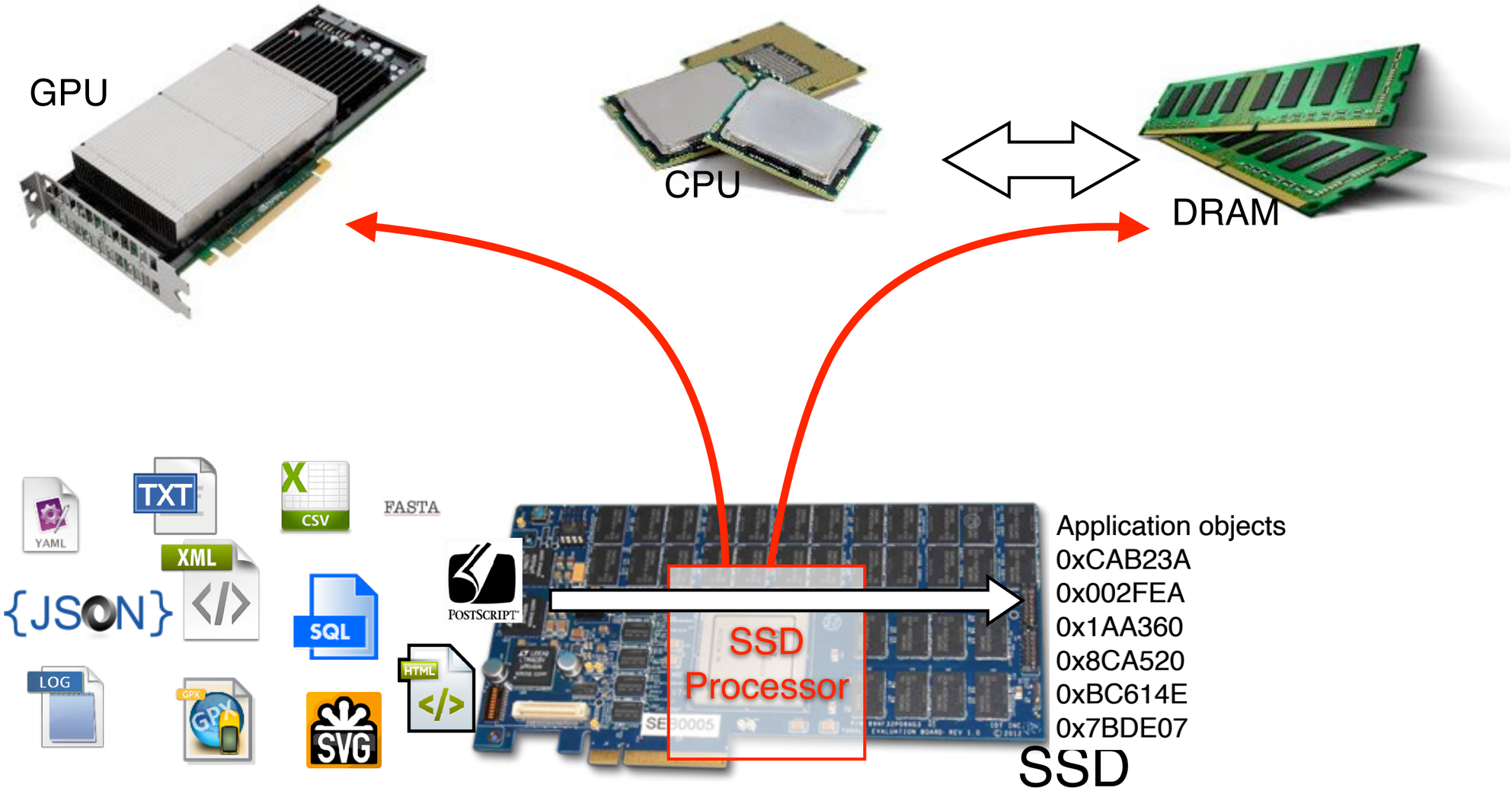




Example — Morpheus✳

✳ 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).

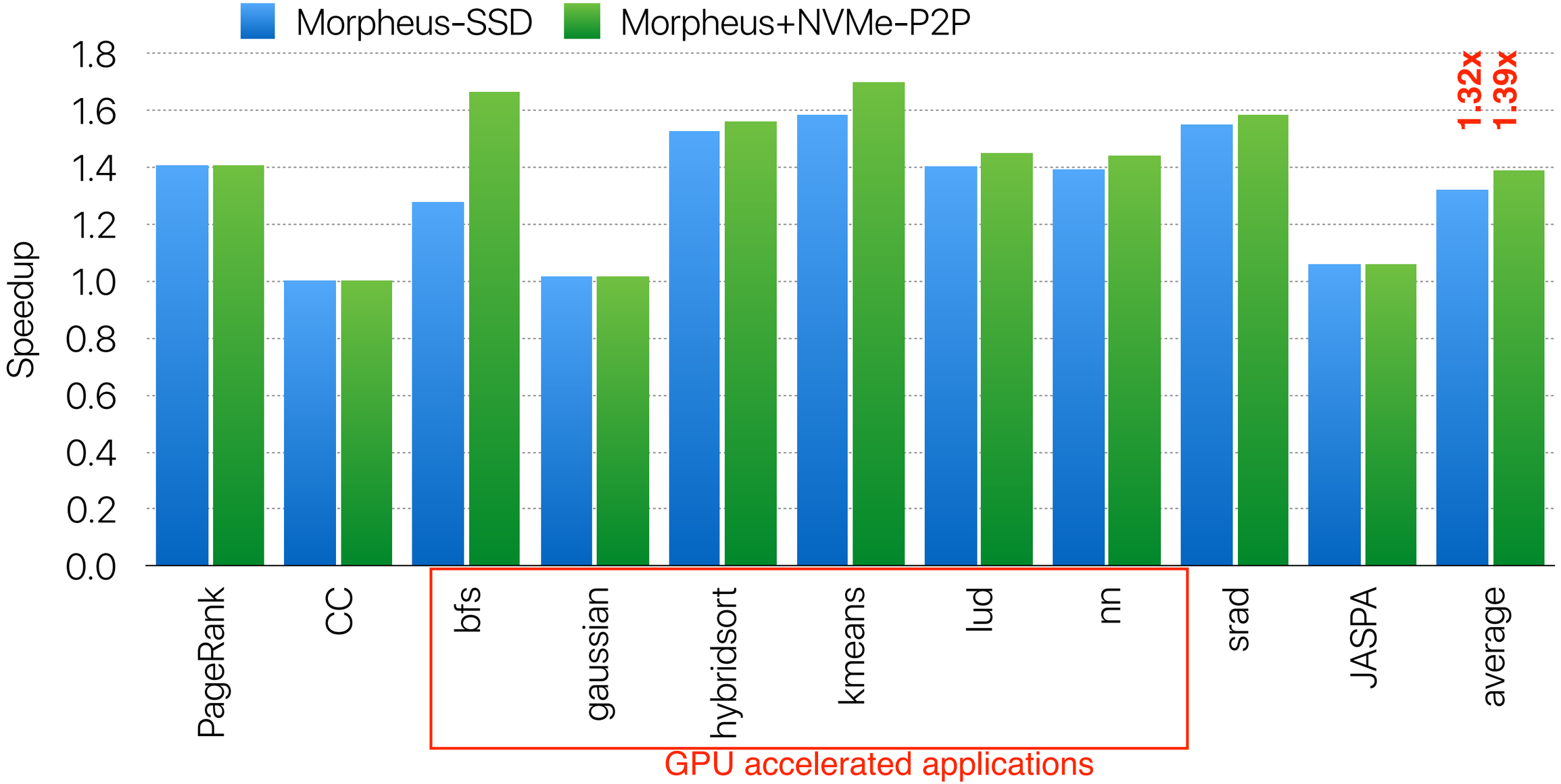
Morpheus


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

Performance



What if we just store them right at the beginning?

3:00

If you could define the data storage format, what’re you going to design for storing big data?

Ideas for big data storage?

Ideas for big data storage?

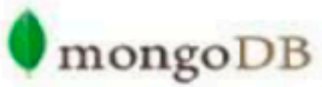
- Low overhead
  - Binary encoded
  - No compression
- Make no assumption to applications/algorithms
- Or at least fit most applications
  - Column-store? Row-store? Or?
  - Sparse? Dense?
- Easy for architectural optimizations (e.g., cache) to work
  - Properly chunked to fit page/cache block size

Modern Data Processing Stack

Relational Engines

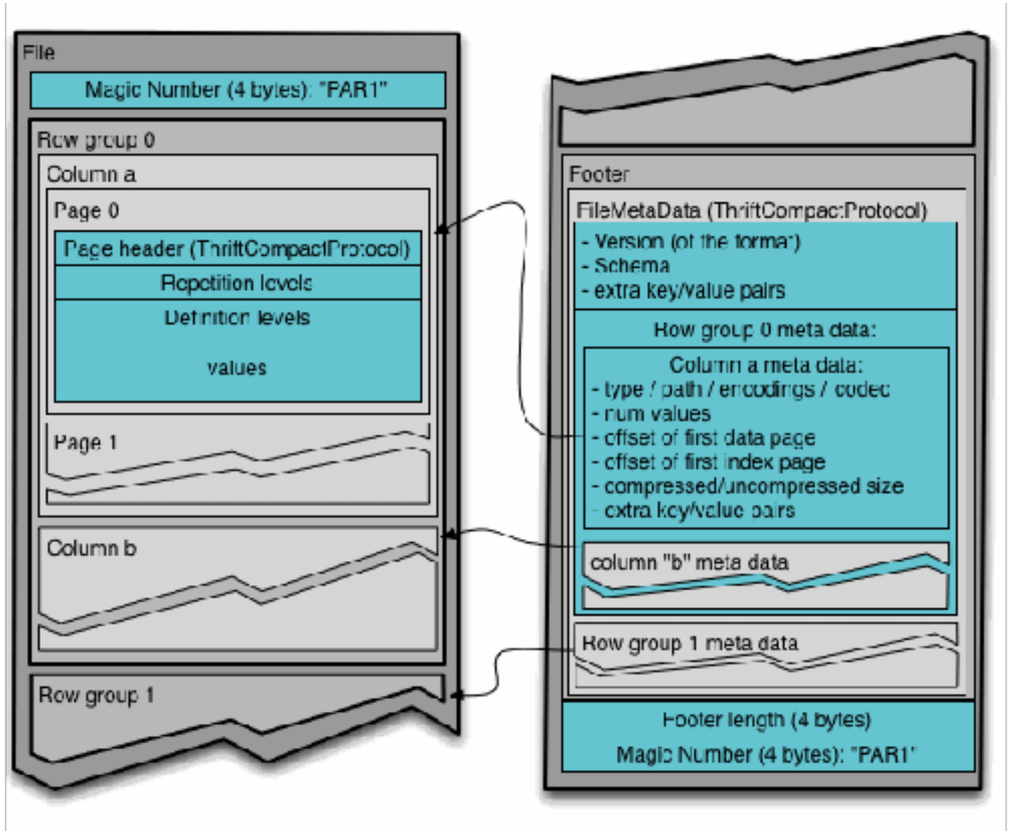
File Formats

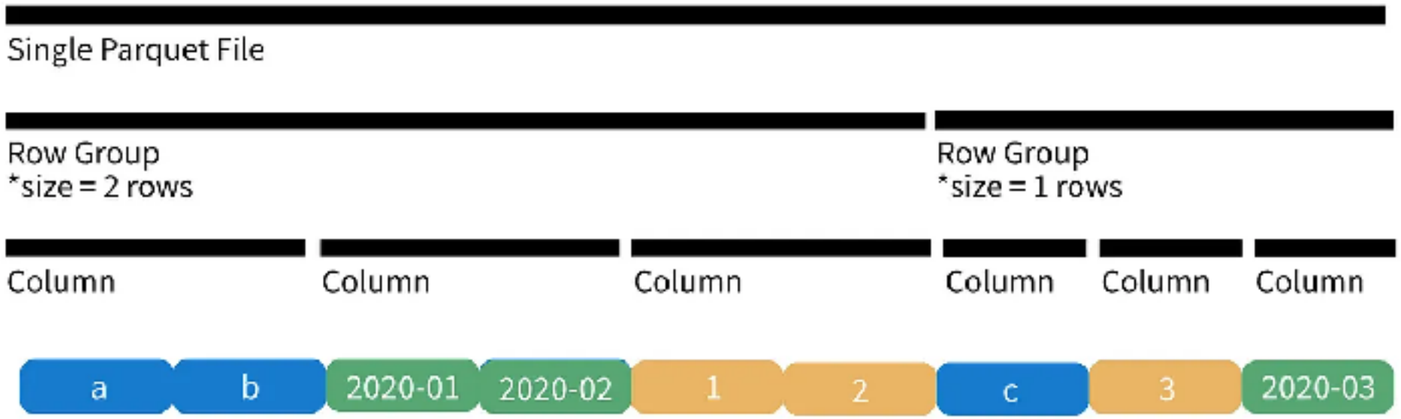
Distributed Storage



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

Apache Parquet





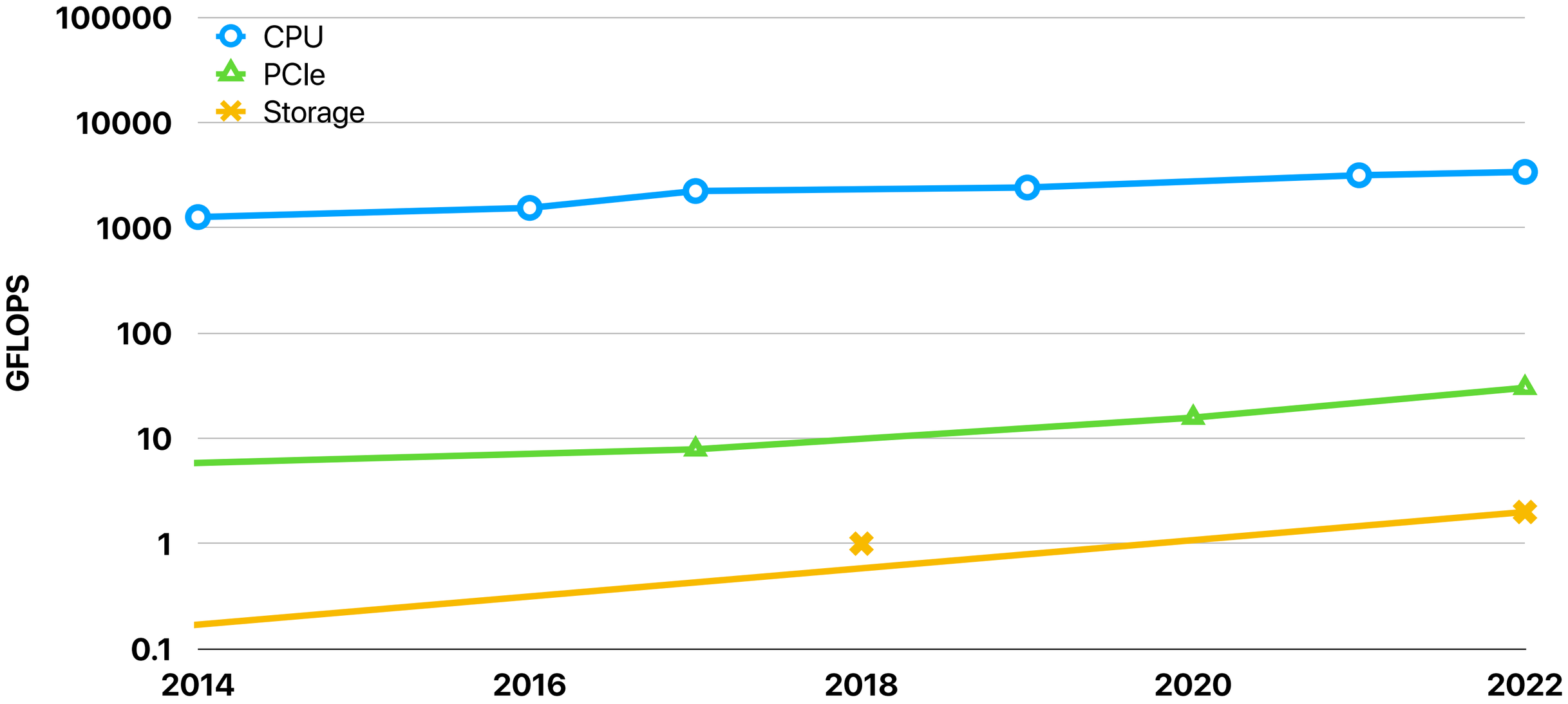
https://github.com/apache/parquet-format

3:00

Parquet is designed in 2013 — can you compare the computer systems at that time v.s. now?

Changes in the last decade?

CPU performance staggered



Datacenters prefer large chunks

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

2.5 Chunk Size

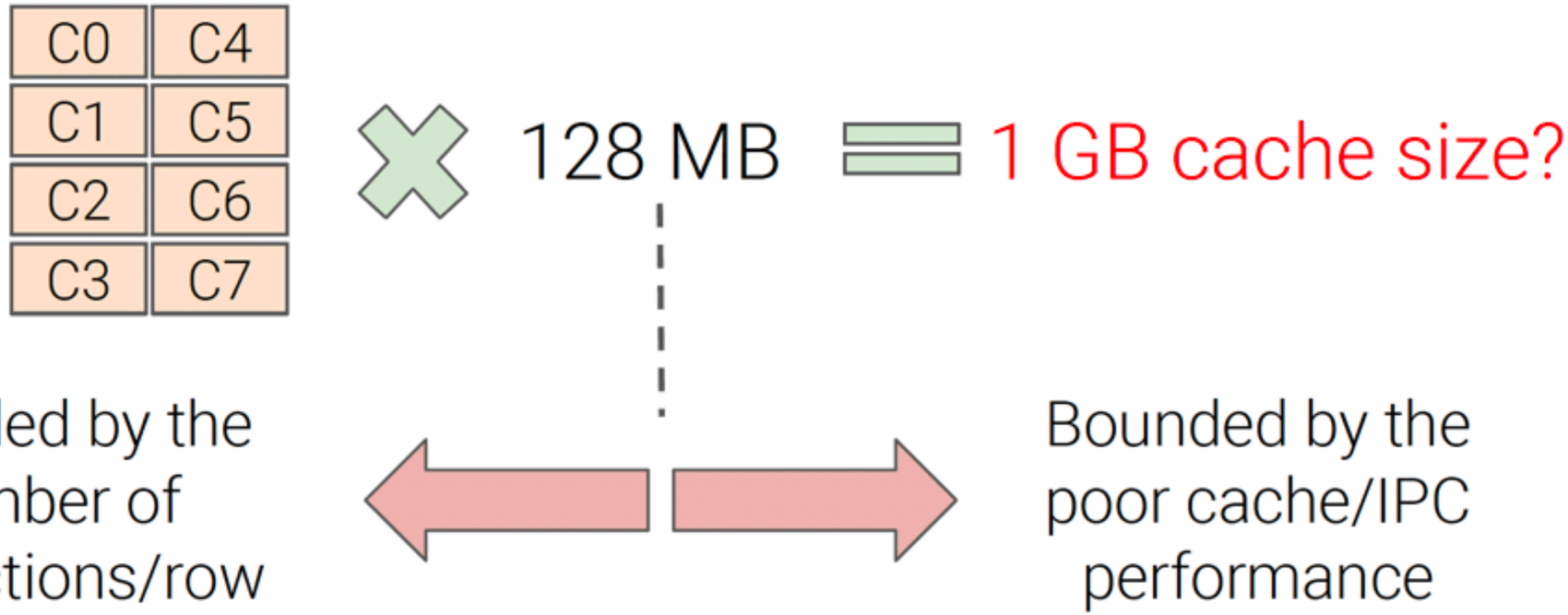
Chunk size is one of the key design parameters. We have chosen 64 MB, which is much larger than typical file system block sizes. Each chunk replica is stored as a plain Linux file on a chunkserver and is extended only as needed. Lazy space allocation avoids wasting space due to internal fragmentation, perhaps the greatest objection against such a large chunk size.

Finding a needle in Haystack: Facebook's photo storage

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel,
Facebook Inc.
{doug, skumar, hcli, jsobel, pv}@facebook.com

Each Store machine manages multiple physical volumes. Each volume holds millions of photos. For concreteness, the reader can think of a physical volume as simply a very large file (100 GB) saved as '/hay/haystack_<logical volume id>'. A Store machine can access a photo quickly using only the id of the corresponding logical volume and the file offset at which the photo resides. This knowledge is the keystone of

But leads to bad CPU cache performance



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

Multi-channel SSD — small random I/O ain't slow anymore

Flashtec™ NVMe2032 and NVMe2016 Controllers

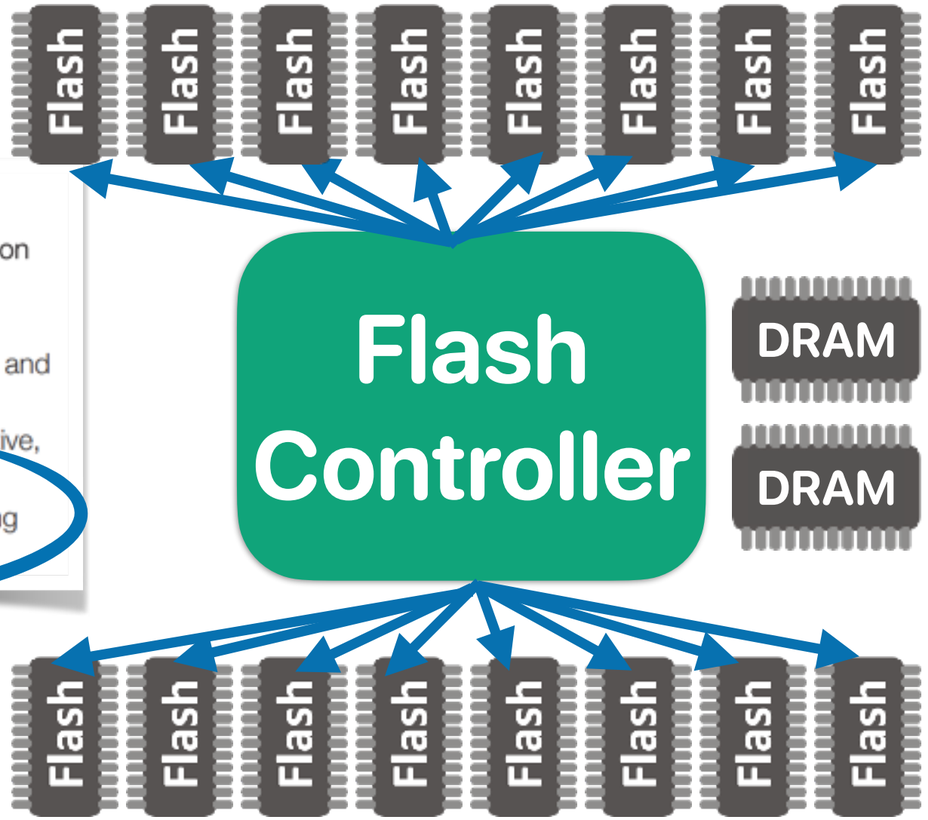
32- and 16-Channel PCIe Flash Controller Pr

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



Each ~500MB/sec
8 channels == 4GB/sec
16 channels == 16 GB/sec

Micron

- Organization
  - Page size x8: 18,592 bytes (16,384 + 2208 bytes)
  - Block size: 2304 pages, (36,864K + 4968K bytes)
  - Plane size: 4 planes x 504 blocks
  - Device size: 512Gb: 2016 blocks; 1Tb: 4032 blocks; 2Tb: 8064 blocks; 4Tb: 16,128 blocks; 8Tb: 32,256 blocks

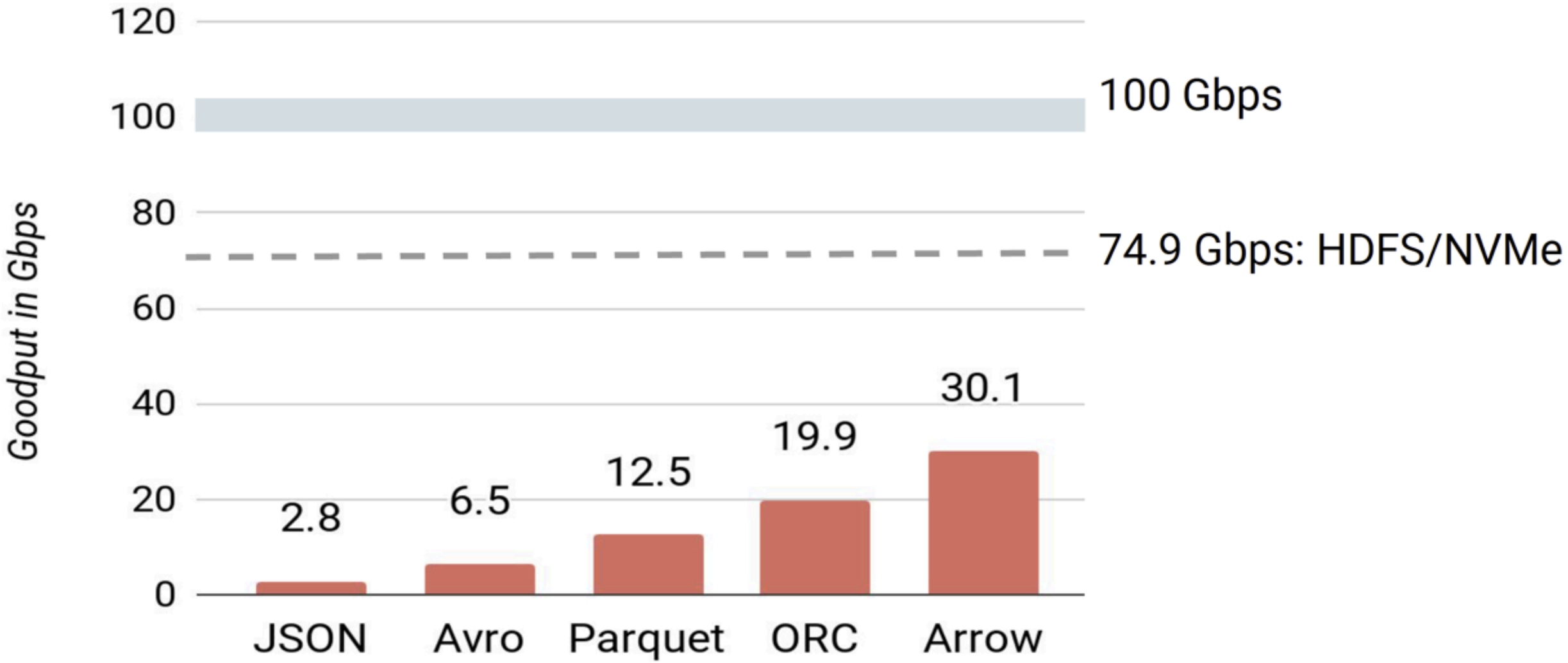
TLC 512Gb-8Tb NAND – Array Characteristics

is suspended or ongoing					
READ PAGE operation time without/with V_{PP}	t_R	57/41	60	µs	2, 12
SNAP READ operation time	t_{RSNAP}	27	37	µs	
Cache read busy time	t_{RCBSY}	11	60	µs	2, 8, 12

Changes in the last decade

- CPU speed v.s. I/O?
- Sequential I/O v.s. Random I/O?
- Remote I/O?
- Metadata lookup?
- Runtime systems

The deserialization performance on popular formats



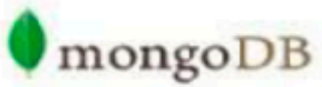
https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_trivedi.pdf

Modern Data Processing Stack

Relational Engines

File Formats

Distributed Storage

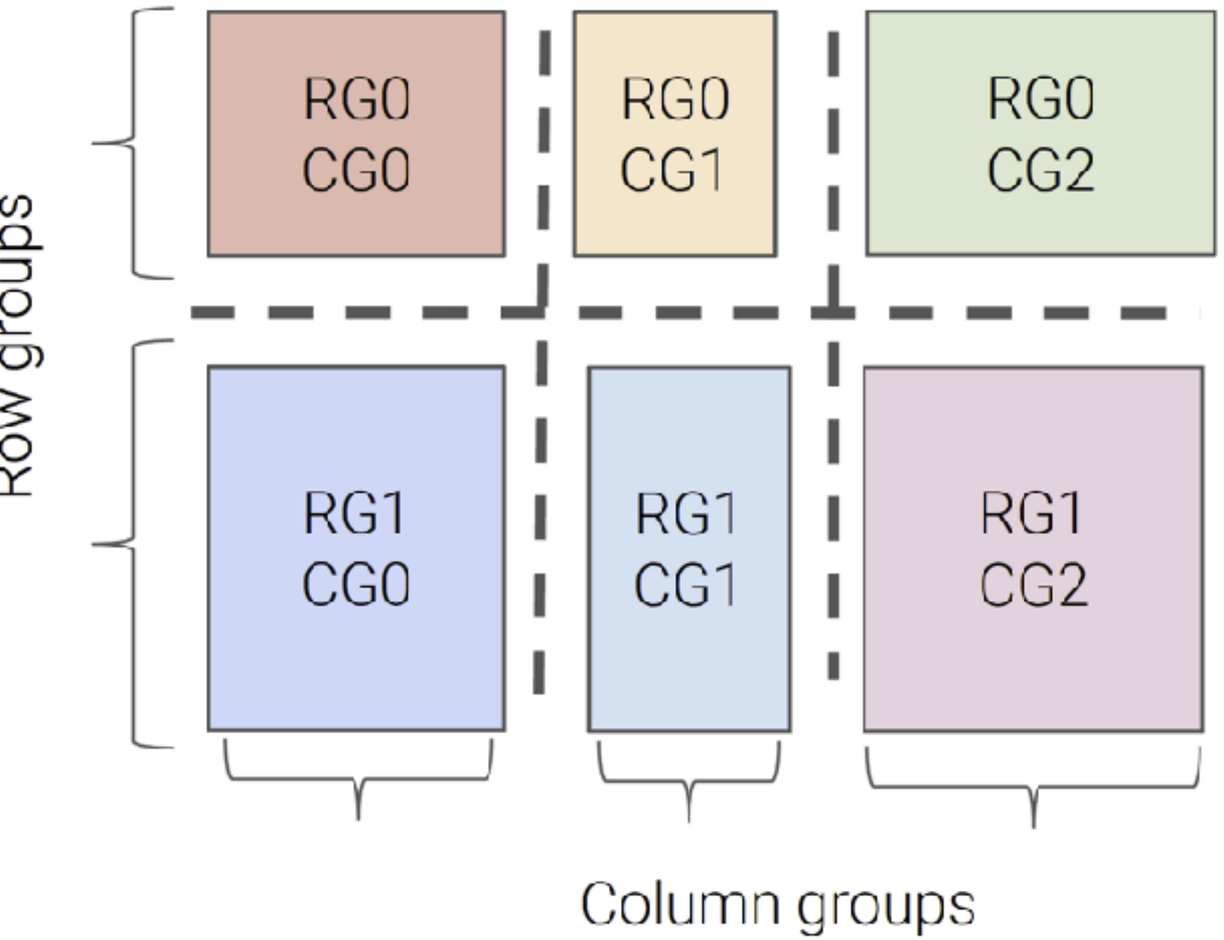


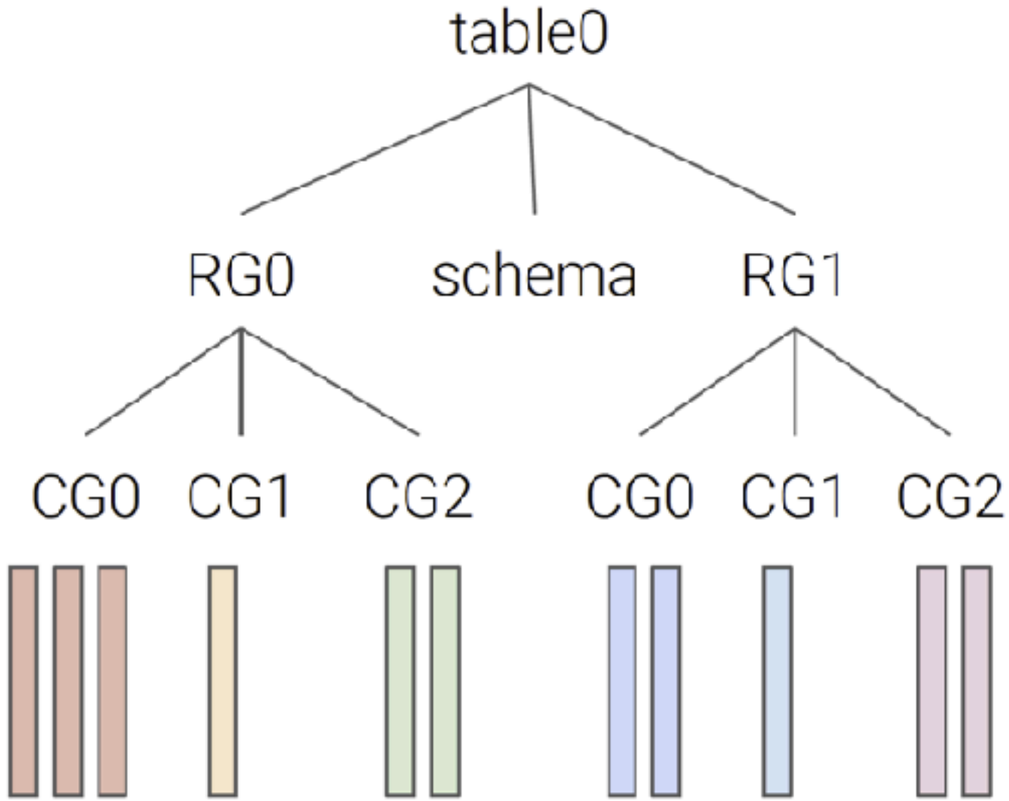
https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_trivedi.pdf

Albis

- Albis - A file format to store relational tables for read-heavy analytics workloads
  - Supports all basic primitive types with data and schema
  - Nested schemas are flattened and data is stored in the leaves
- Three fundamental design decisions:
  - Avoid CPU pressure, i.e., no encoding, compression, etc.
  - Simple data/metadata management on the distributed storage
  - Carefully managed runtime - simple row/column storage with a binary API

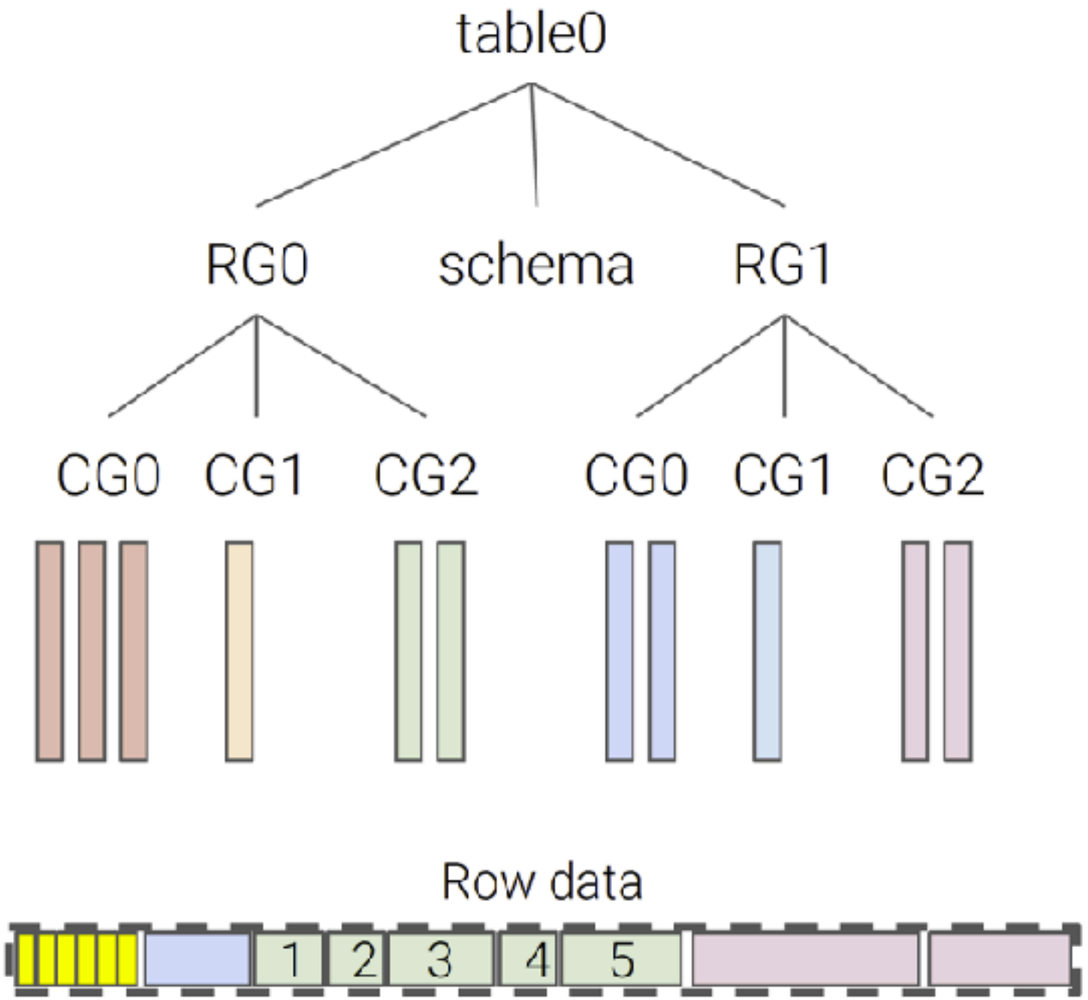
Not just row groups, but also column groups





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

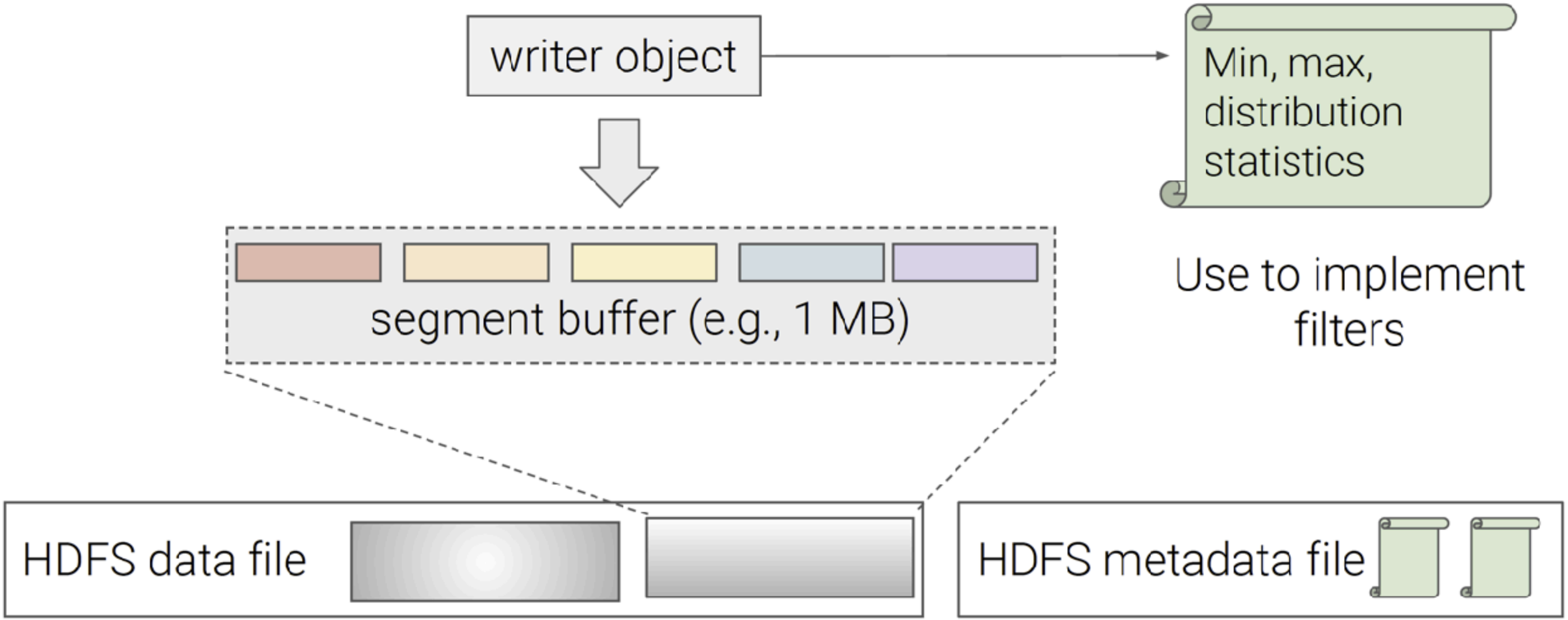
Reading a row



1. Read schema file
2. Check projection to figure out which files to read
    a. Complete CGs
    b. Partial CGs
3. Evaluate filters to skip segments
4. Materialize values
    a. Skip value materialization in partial CG reads

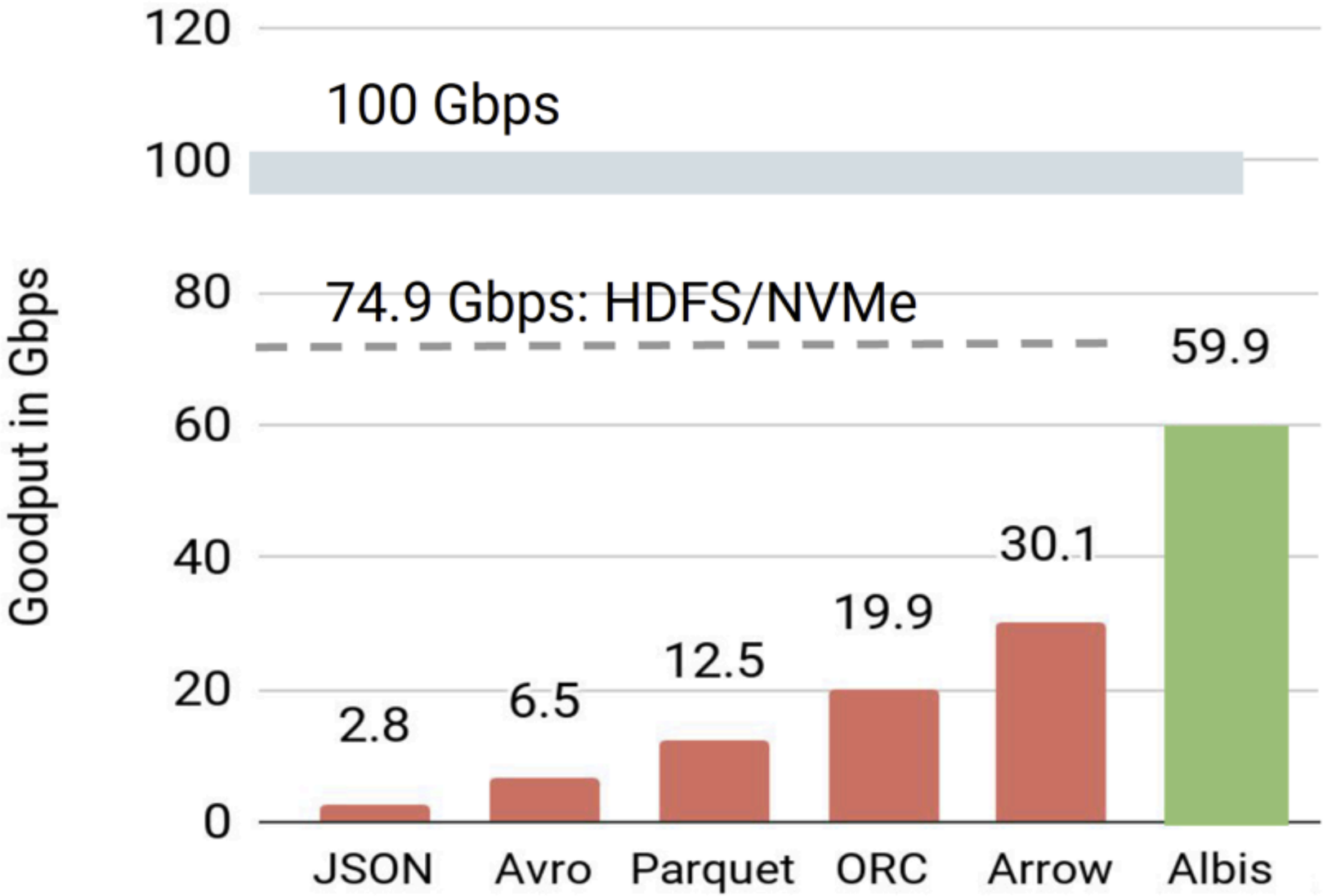
https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_trivedi.pdf

Writing a row



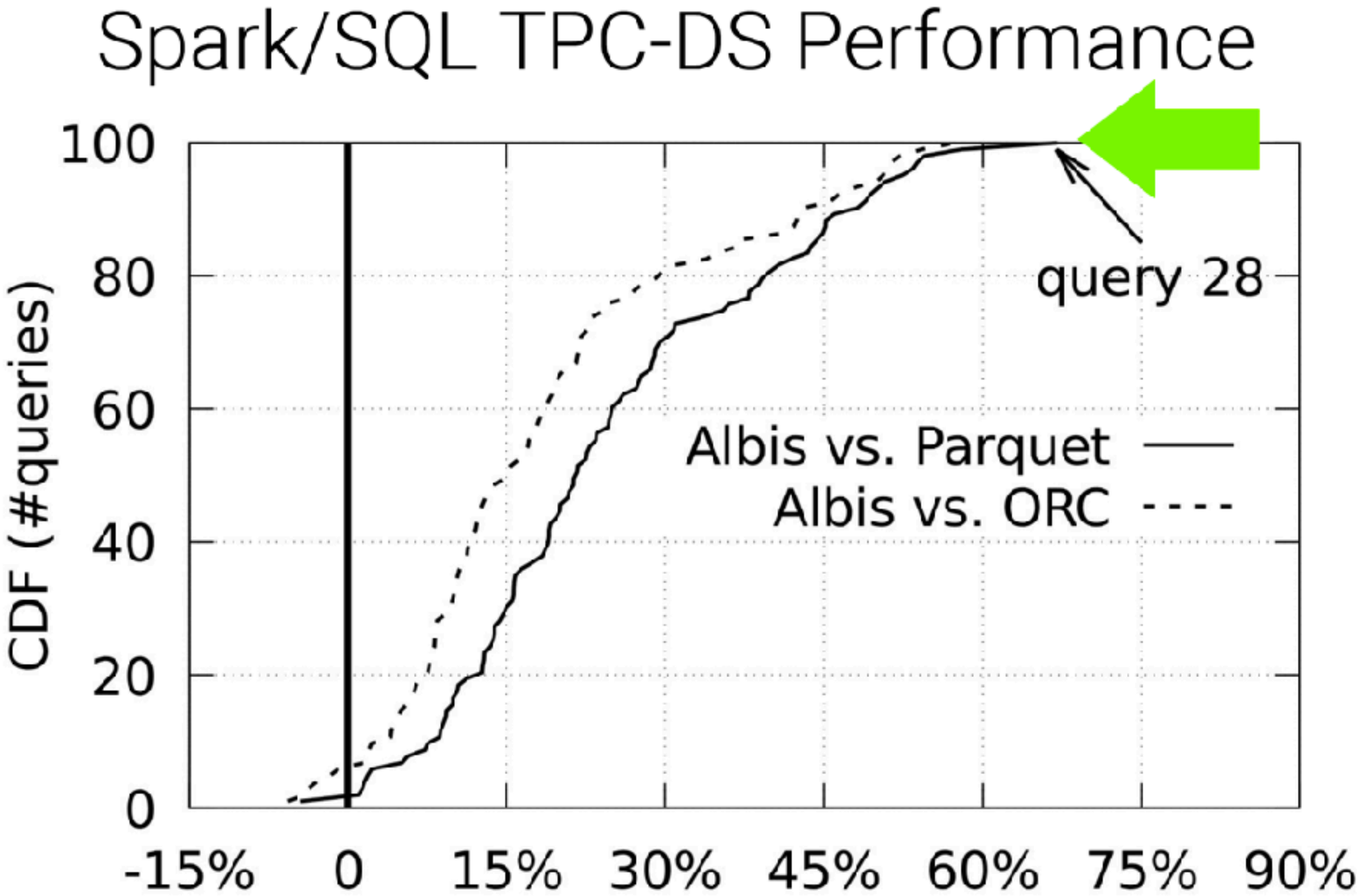
https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_trivedi.pdf

Performance of Albis



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

Albis v.s Parquet



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

Protobufs

Announcement

- Final project presentation — 3/21 11:30a-2:30p (SSC 125)
- No lecture next Tuesday, but we'll be back on Thursday
- CEN Tech Talk this Friday @ 12:30p /WCH 205/206 by Zoe Wang from Google

Zoe Wang

Technical Program Manager at Google

Friday, March 1, 2024
12:30 pm - 1:30 pm

FOOD WILL BE PROVIDED!

Winston Chung Hall
205/206

Guest speaker biography: Zoe is a technical program manager at Google. Her career has been focused on Machine Learning (ML) productionization. Currently she works with her team bringing ML models to mobile devices that power some of AI features for Pixel and other edge

Electrical
Computer Science
Engineering

277

つづく