Modern Storage

Sai Narasimhamurthy (Seagate, UK)
sai.narasimhamurthy@seagate.com

Jean-Thomas Acquaviva (DDN, France)
jtacquaviva@ddn.com

Konstantinos Chasapis (DDN, Germany)
kchasapis@ddn.com

Foreword

This session has been designed to be vendor agnostic and only reflects the personal views of the authors.
The content cannot be interpreted as a commitment from their respective companies.

Outline

9:00am

- Infrastructure hardware: - 30 minutes -KC
    - Storage devices characteristics
    - Storage devices evolution
    - Importance of software in infrastructure
    - Resulting stack and standardization aspects
    - New applications
- Infrastructure software - 30 minutes - Sai
    - posix
    - mpi-io
    - netcdf
    - object
- Storage trend and possible futures
    - Deep and multi-tier storage hierarchy
    - Technical challenges
        - metadata, data policies, fault tolerance
        - perspective - Storage Class Memory

10:00am KC

- Introduction to Darshan - 30 minutes -
    - Why, Install, HOWTO
    - Darshan DXT

10:30am virtual break

10:45am - KC

- Hands-on session - 1H -
    - 4 differents code to analyse

12:00 wrap-up

Storage devices characteristics

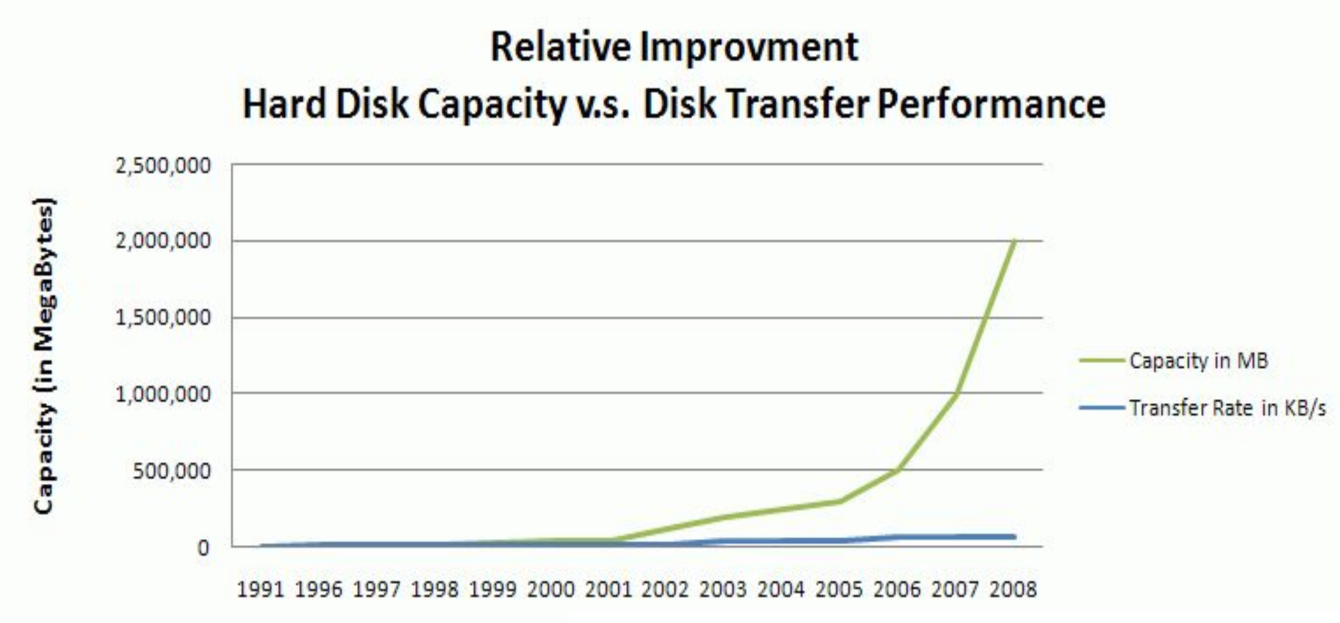
- Storage Medium
  - Magnetic Tapes, Hard Disk Drive (HDD)
  - Solid State Drive (SSD), Non-Volatile Memory (NVM)
  - Intel Optane Memory based on 3D Xpoint
- Throughput
  - How many bytes can it pass per second
- Latency
  - How much time does it need to perform one operation
- IOPS
  - How many operations performed per second
- Capacity
  - How much data can it store
- Connection Type
  - Which type of connection/protocol does it use

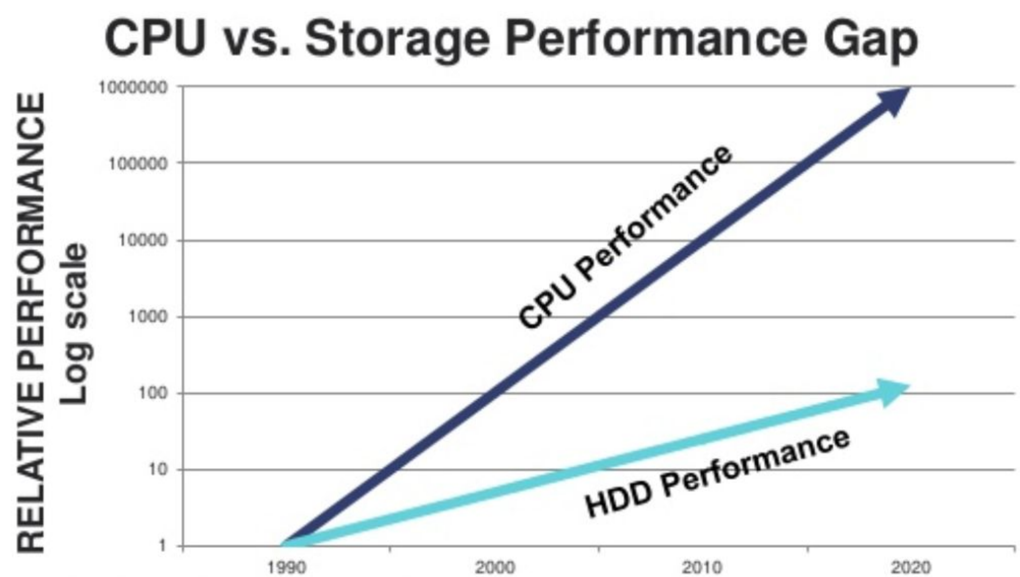
Important Performance Considerations

Storage Medium - Capacity - Connection Type
Throughput - Latency - IOPS

- Different Storage device technologies fall at different points in these parameter space
- Possibility to aggregate storage devices to achieve different throughput and Capacity
- For example – Possibility of taking advantage of multiple hard disks in parallel to achieve better throughput (& Capacity)
- Latency and IOPS are very device dependent

Historical Performance of HDDs





Ref: Intel

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

SSDs now offer better near-line performance than HDDs

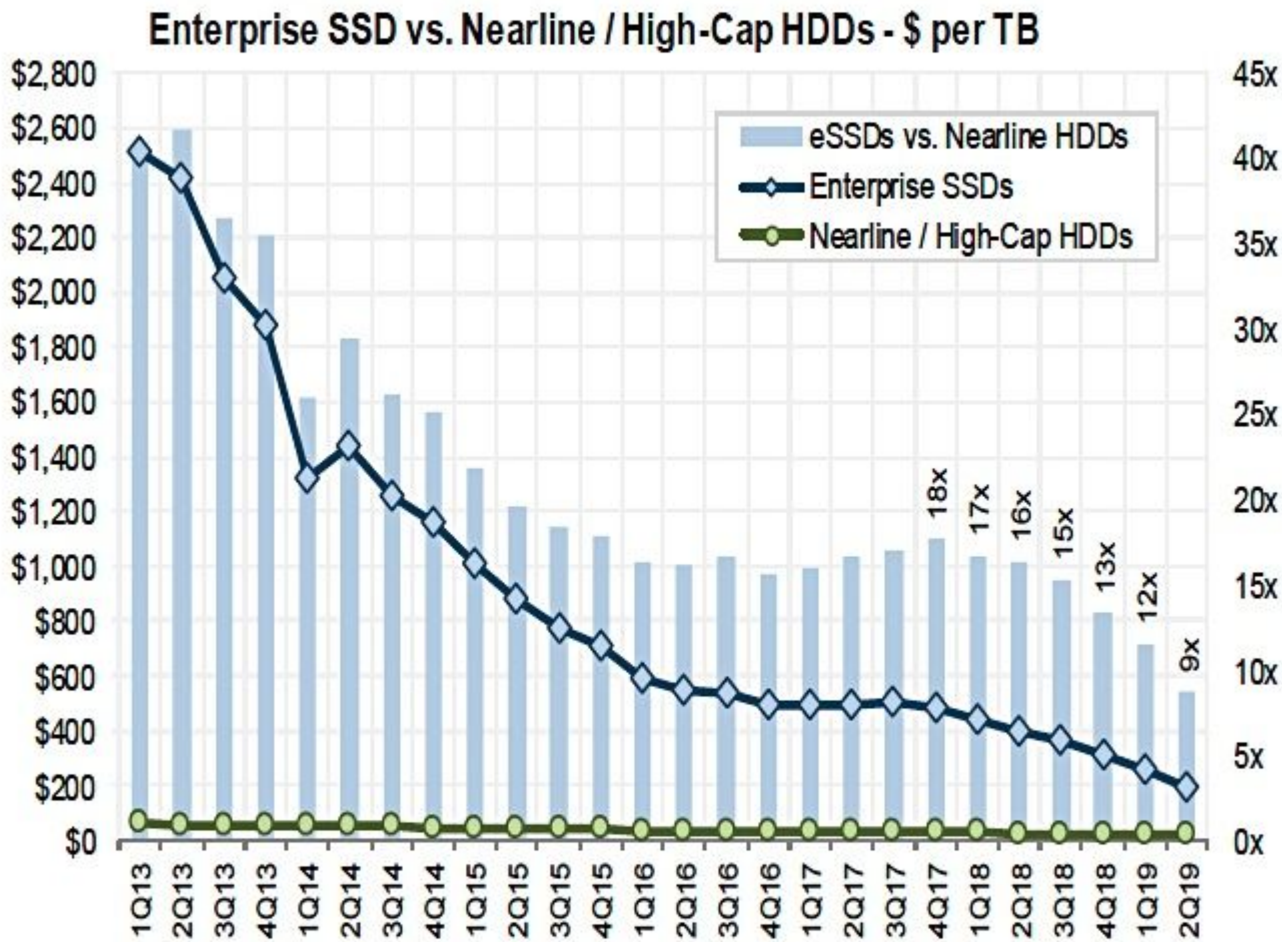
SSD vs. HDD

Usually 10,000 or 15,000 rpm SAS drives

SSD		HDD
0.1 ms	**Access Times** SSDs exhibit virtually no access time	**5.5-8.0 ms**
SSDs deliver at least **6000 io/s**	**Random I/O Performance** SSDs are at least 15 times faster than HDDs	HDDs reach up to **400 io/s**
SSDs have a failure rate of less than **0.5%**	**Reliability** This makes SSDs 4-10 times more reliable	HDDs failure rate fluctuates between **2-5%**
SSDs consume between **2 and 5 watts**	**Energy Savings** This means that on a large server, approximately 100 watts are saved	HDDs consume between **6 and 15 watts**
SSDs have an average I/O wait of **1%**	**CPU Power** You will have an extra 6% of CPU power for other operations	HDDs average I/O wait is about **7%**
The average service time for an I/O request while running a backup remain below **20 ms**	**Input/Output Request Times** SSDs allow for much faster data access	The I/O request time with HDDs during backup rises up to **400-500 ms**
SSD backups take about **6 hours**	**Backup Rates** SSDs allow for 3-5 times faster backup for your data	HDD backups take up to **20-24 hours**

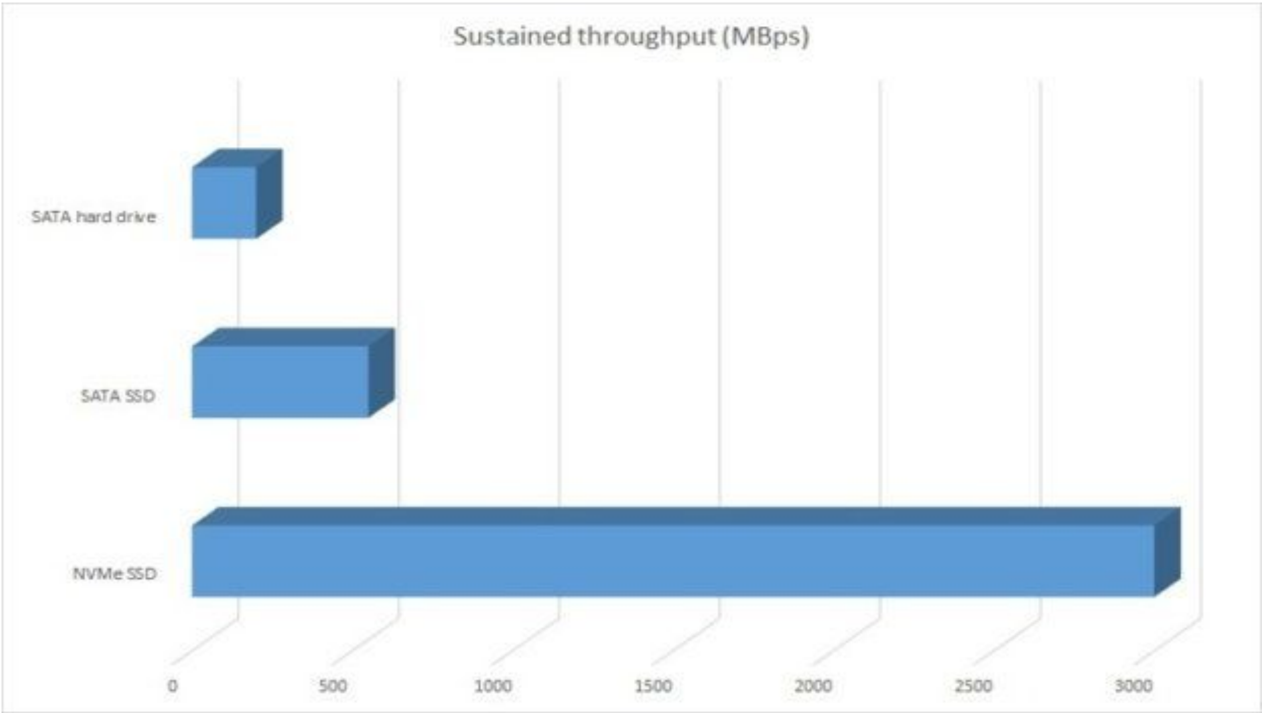
Ref: Enterprise Storage Forum

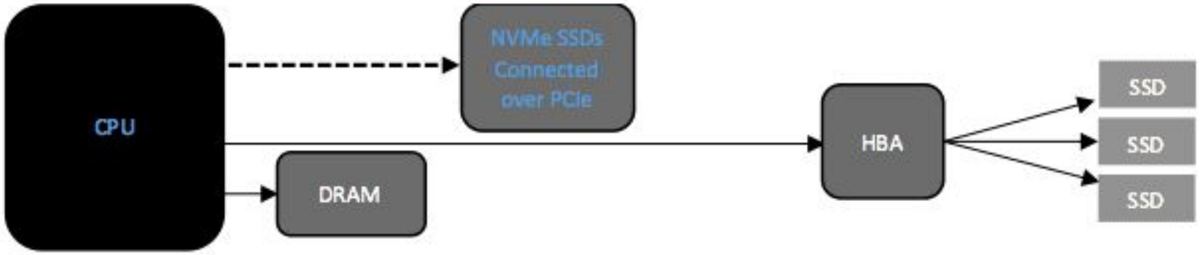
HDD VS SSD cost comparison



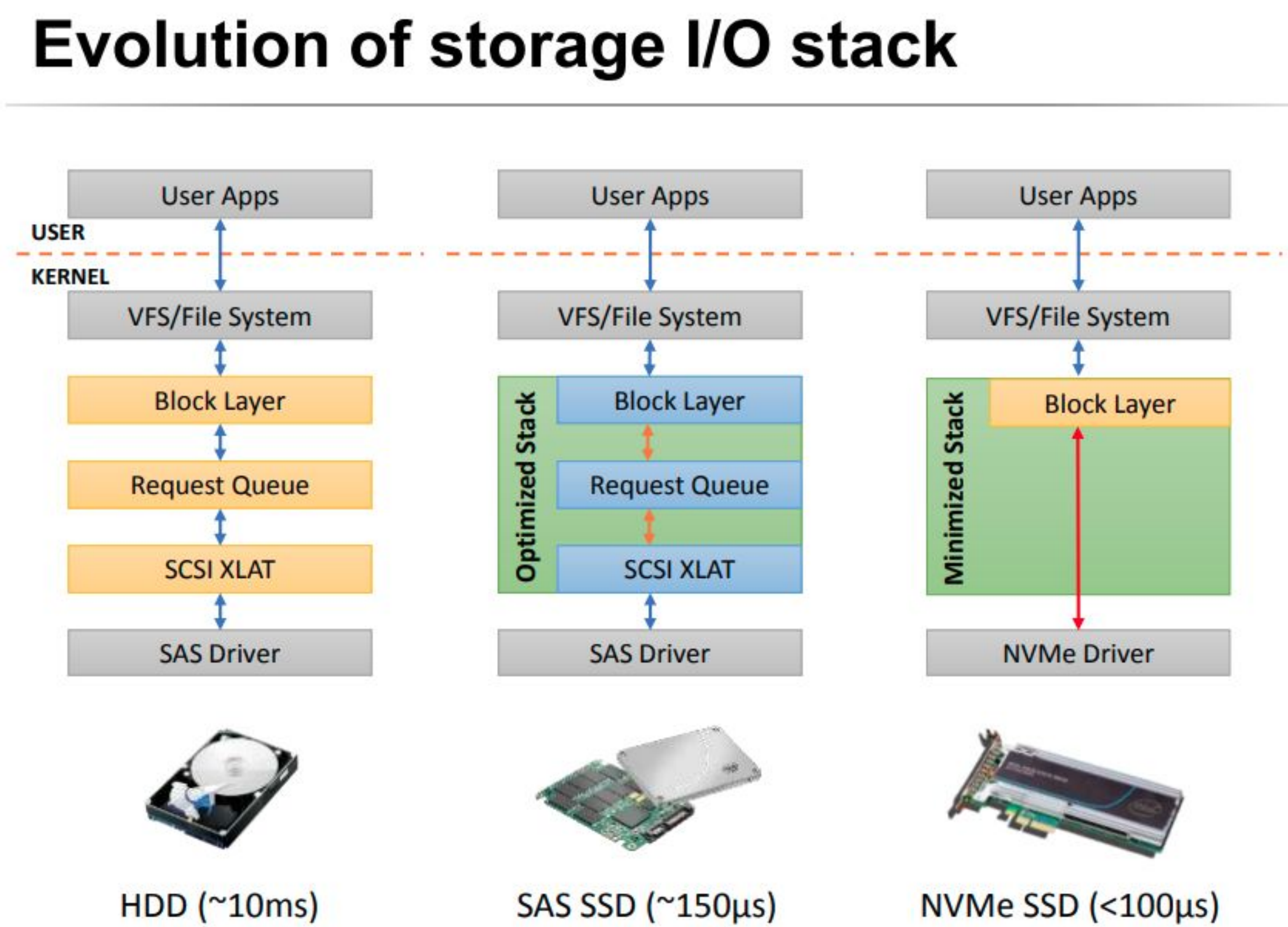
Source: IDC; TrendFocus; Wells Fargo Securities, LLC

The arrival of NVMe protocol for accessing SSDs





NVMe gain over SSD is software based



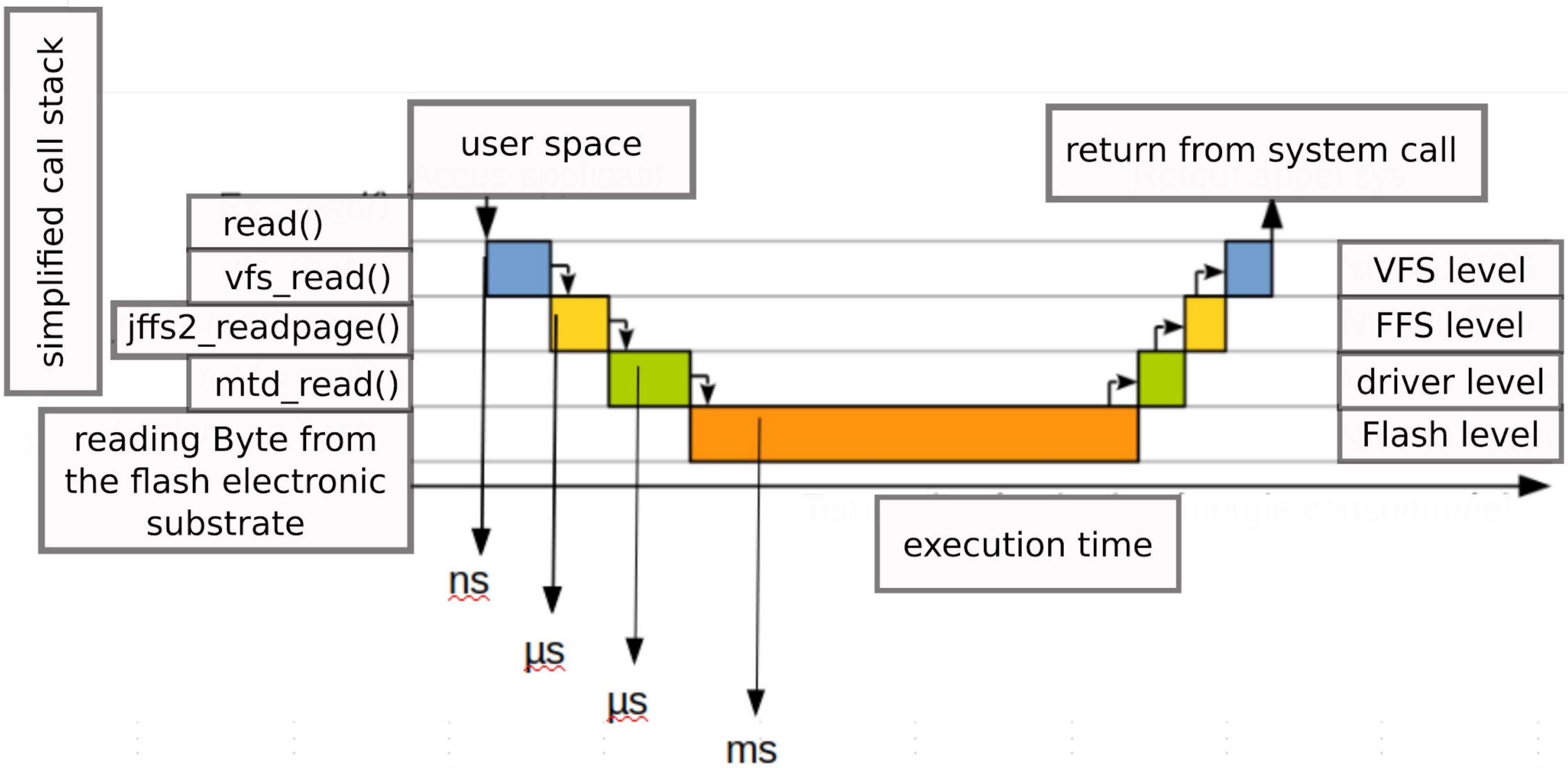
Ref: Flash Memory Summit

Access Latency Compared to DRAM

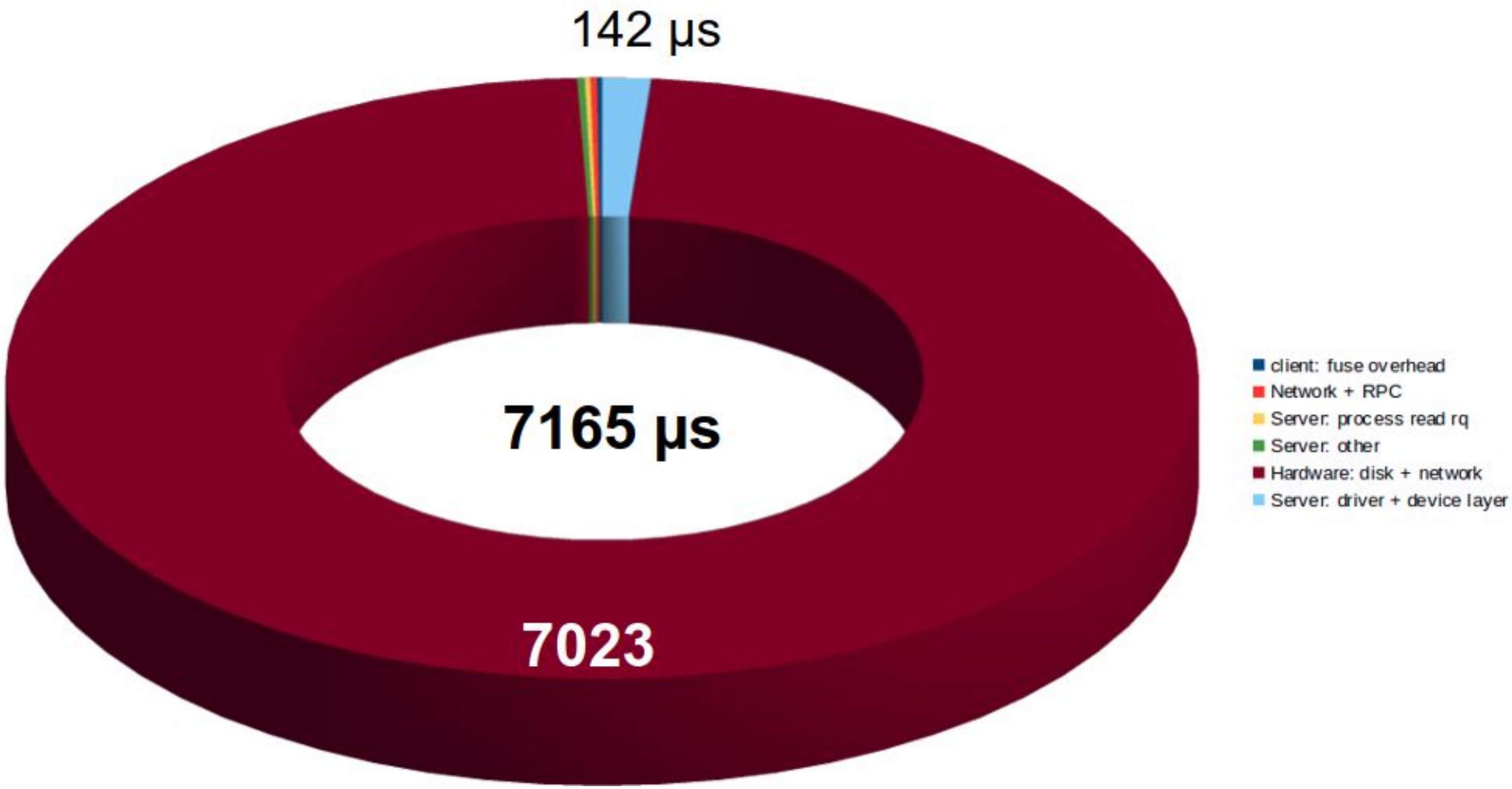
CPU register	O(0.1ns)
Cache	O(1ns~10ns)
DRAM	O(10ns~100ns)
SCM	O(1000ns)
NVM	O(10μs~100μs)
HDD	O(10ms)

Dramatic latency difference between memory and different storage mediums

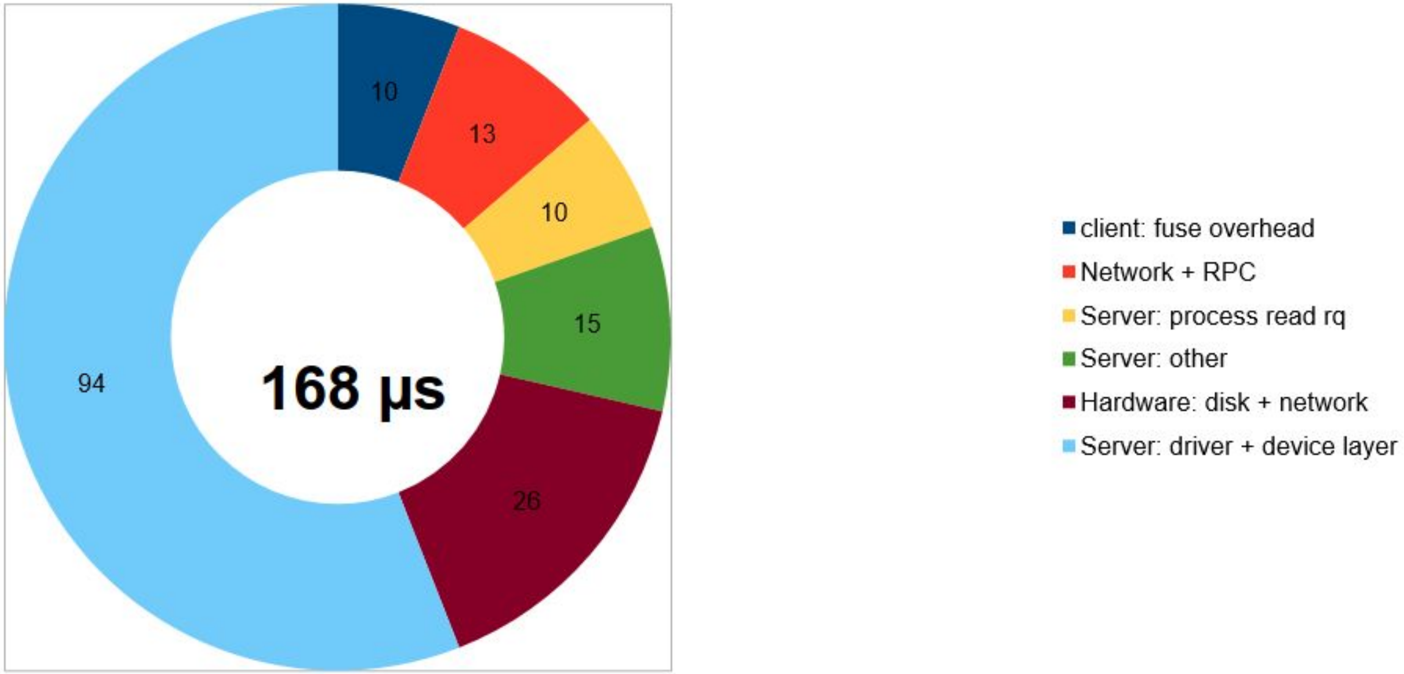
I/O path call trace



Random 4K Read: HDD 10K RPM



Random 4K Read: NVMe latency



I/O behavior performance impact

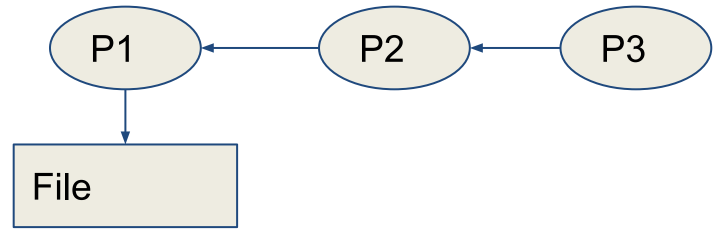
- The performance observed by the application might be orders of magnitude lower than the maximum performance the storage system can offer
  - Implications factors
    - File Access pattern
    - File I/O model

- File access pattern categories
  - Sequential / strided
  - Random
  - Request size
  - Aligned

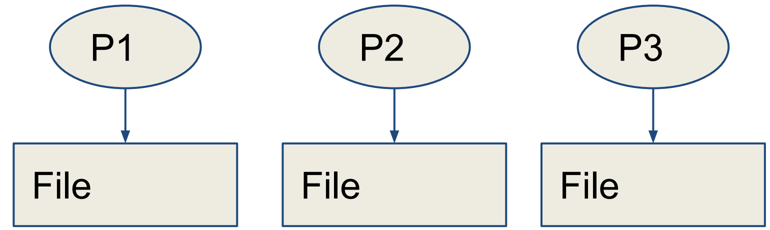
- File I/O models
  - File per process
  - Shared file model
    - Serial I/O (each process sends data to a single master)
    - Parallel I/O (each process performs I/O directly to the file)

Serial and parallel I/O

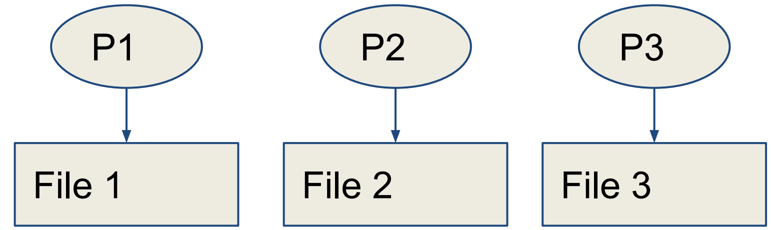
Serial I/O



Parallel Shared file I/O



Parallel file per process



IO500 Benchmark

Community developed and maintained benchmark which aims to covers various I/O workloads

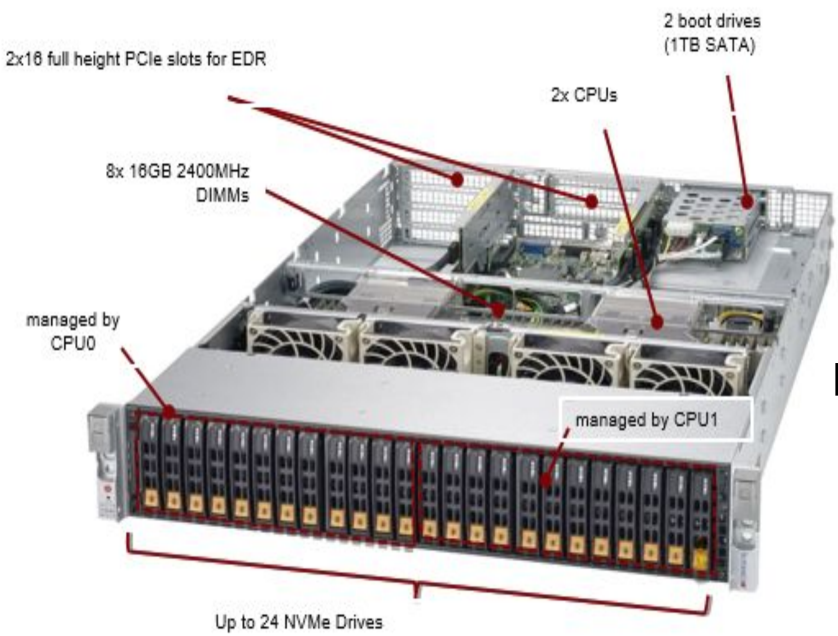
- Workloads
    - IOEasy: Applications with well optimized I/O patterns
    - IOHard: Applications that require a random workload
    - MDEasy: Metadata/small objects
    - MDHard: Small files (3901 bytes) in a shared directory
    - Find: Finding relevant objects based on patterns

Additional analysis for write performance

Taking further the IO500 idea of easy / hard pattern

- Write access
  - Random / Sequential access
  - Small (4K) / Medium (32k) / Large (1MB)
  - Single Shared File / File Per Process

Parallel I/O Apple-to-Apple Comparison

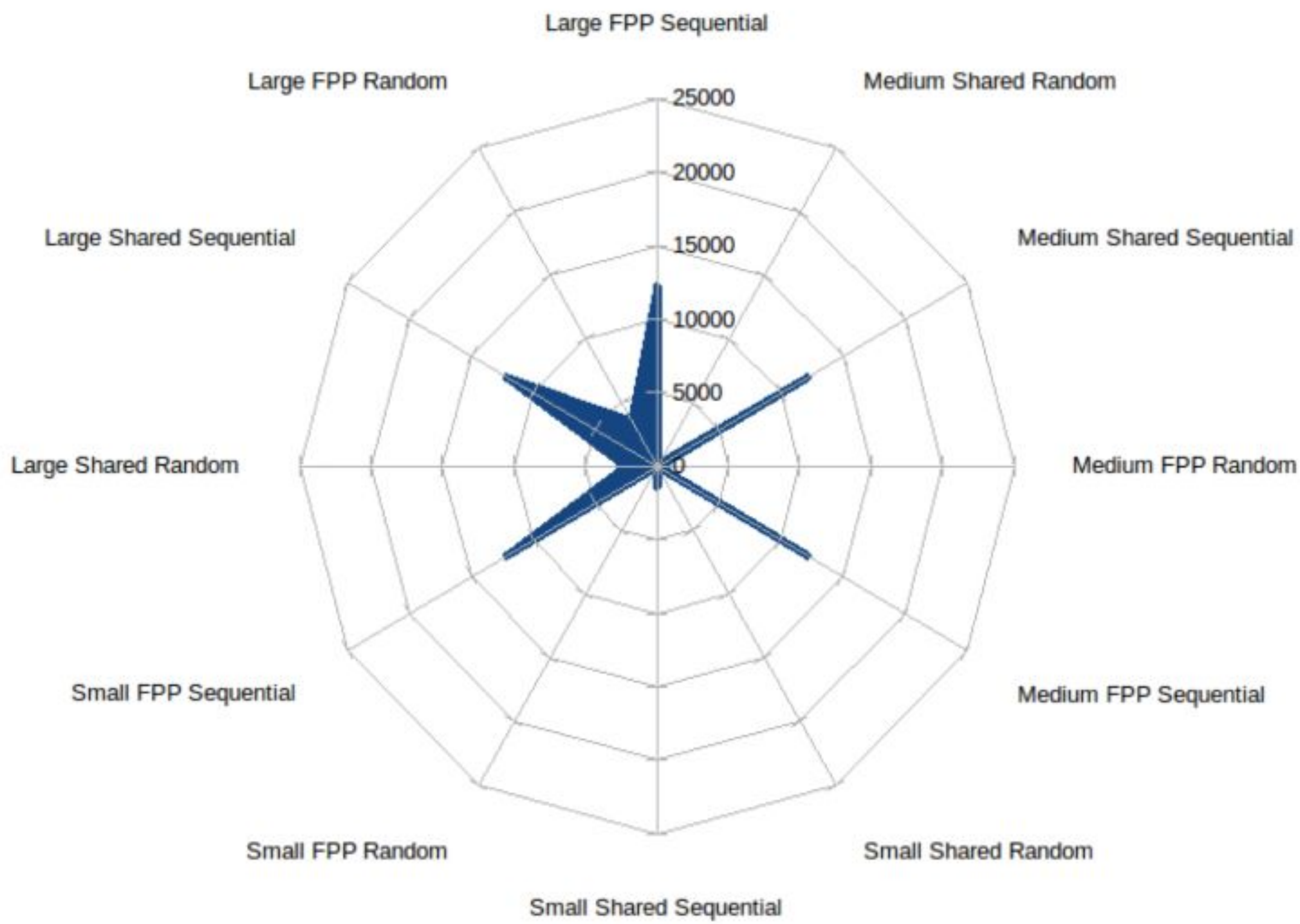


- 2U - Dual-socket Intel
- Performance is around 20GB/s
- 24 NVMe
- 2 x16 EDR

Peak network performance:
2 EDR = 25 GB/s

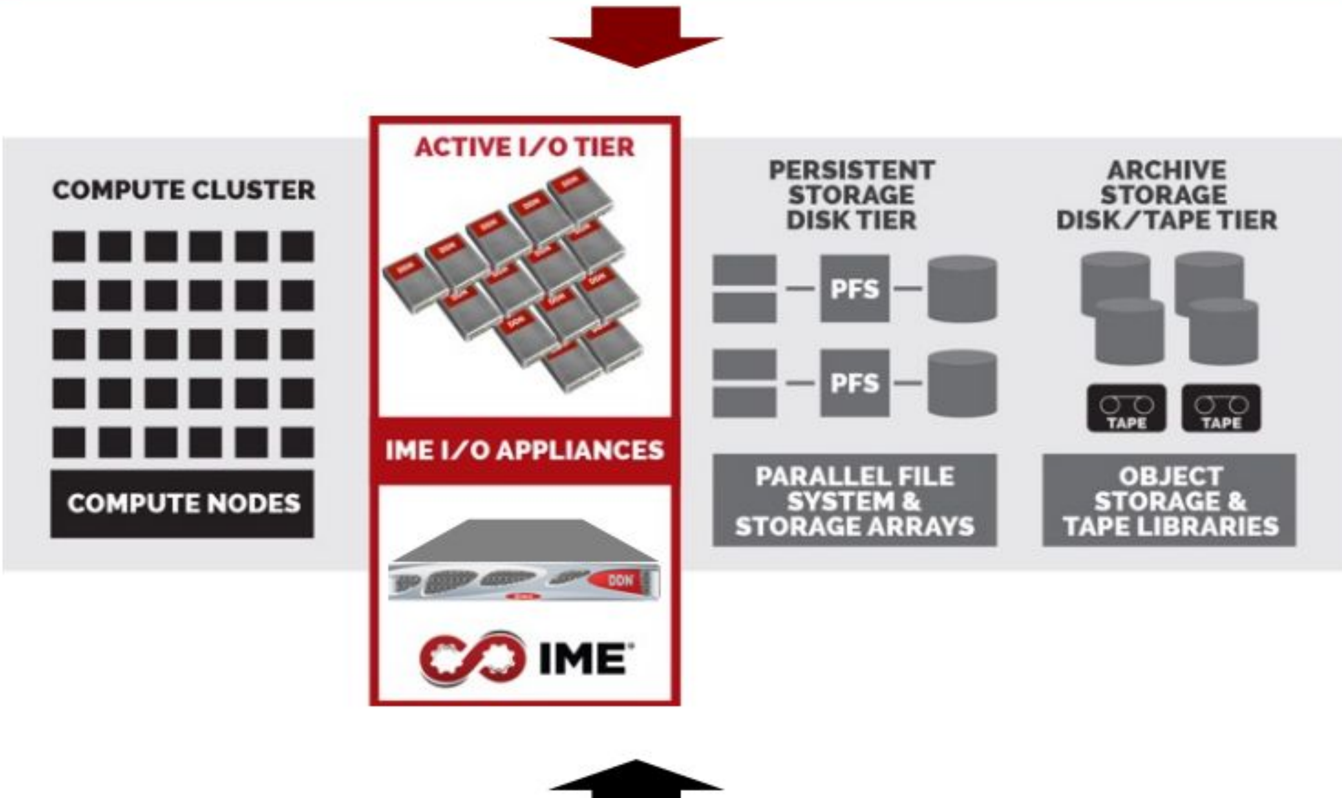
Results obtained with Lustre

Results obtained with Spectrum Scaler v4



Introducing an FLASH cache

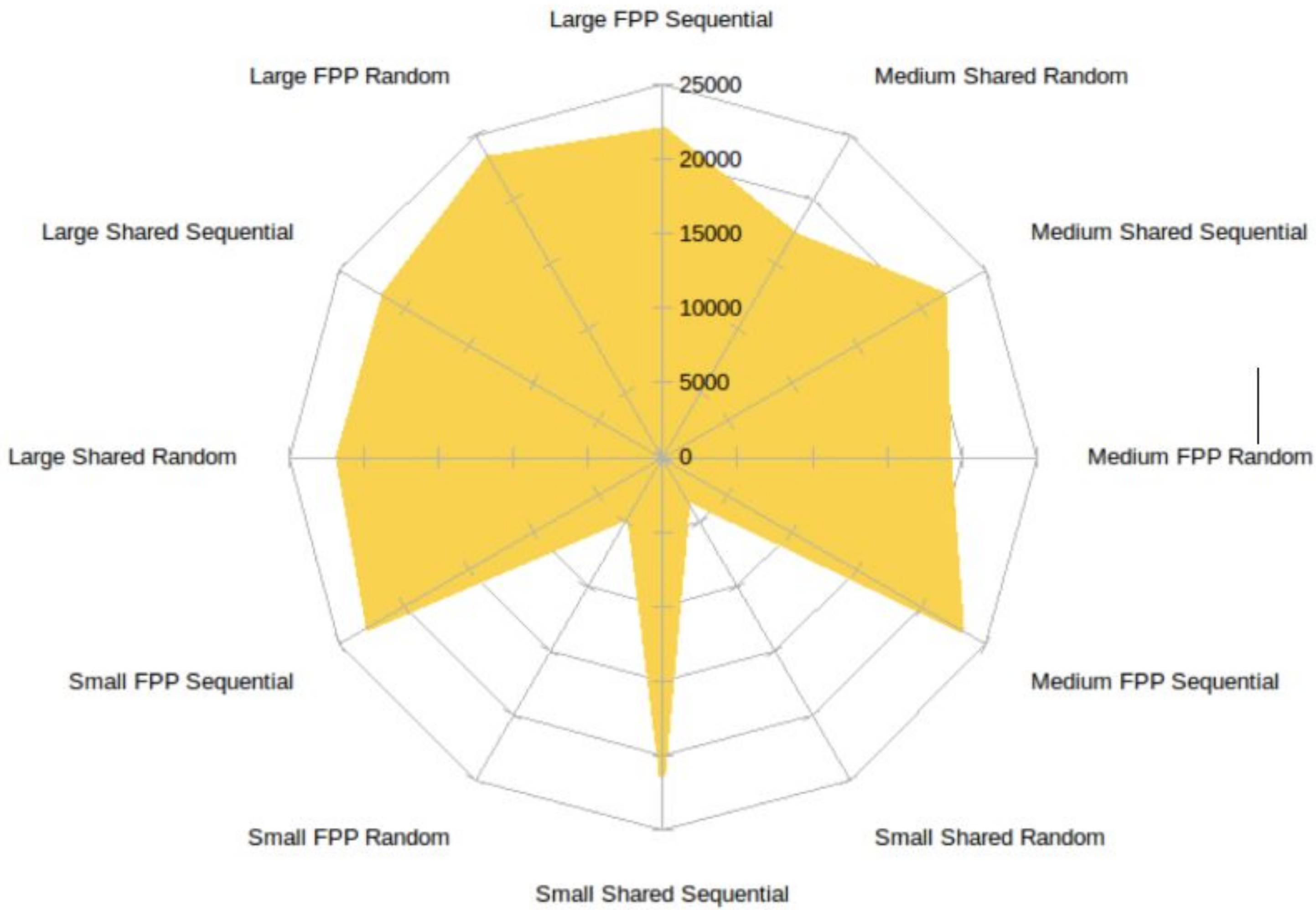
IME's *Active I/O Tier*, is inserted right between
Compute and the parallel file system



IME software intelligently virtualizes disparate NVMe SSDs into
a single pool of shared memory that accelerates I/O, PFS & Applications

Results obtained with IME 1.2

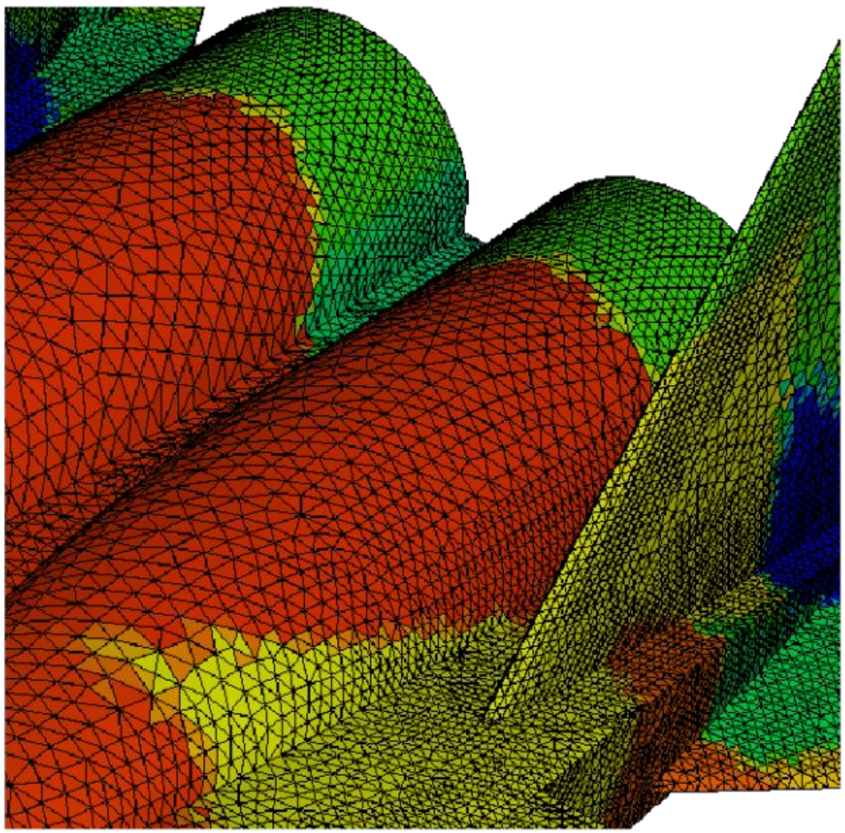
→ thinner software layer brings better performance



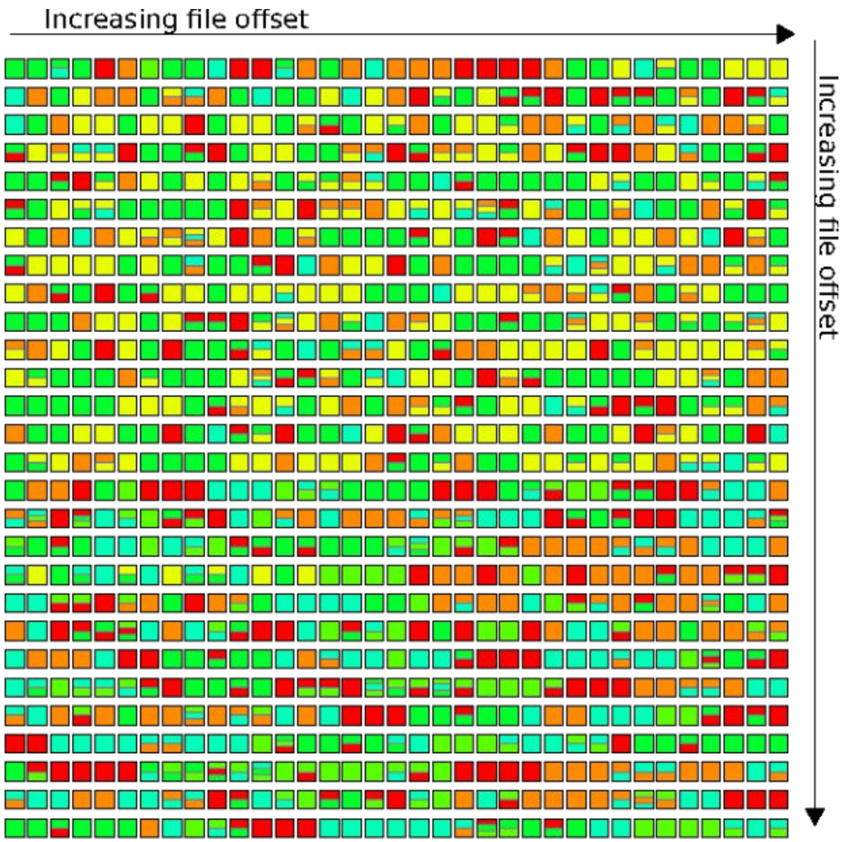
Is random small write an artifact?

Single shared file sparse access is intrinsic to many simulation workloads

-> Byte addressability is a performance enabling feature of flash device



(a) Decomposed mesh



(b) File mapping

Flash as an additional layer in the storage stack

Complexity of storage stack

With multiple layers the requirements of POSIX consistency are more challenging to implement

-> *de facto* standard: NFS open()/close() consistency

Posix Test Suite PJD	EXT4 compliance	GPFS compliance	Lustre compliance	NFSv3 compliance	NFSv4 compliance	Union-FS compliance	IME FUSE compliance
CHFLAGS	100%	100%	100%	100%	100%	100%	100%
CHMOD	100%	96%	99%	99%	100%	77%	91%
CHOWN	100%	100%	97%	99.7%	100%	82%	92%
FTRUNCATE	100%	100%	100%	100%	100%	89%	95%
GRANULAR	100%	100%	100%	100%	100%	100%	100%
LINK	100%	100%	100%	100%	100%	89%	95%
MKDIR	100%	100%	99%	100%	100%	88%	94%
MKFIFO	100%	100%	99%	100%	100%	87%	94%
MKNOD	100%	100%	99%	100%	100%	86%	92%
OPEN	100%	100%	98%	100%	99%	91%	96%
FALLOCATE	100%	100%	100%	100%	100%	100%	100%
RENAME	100%	100%	100%	100%	100%	90%	98%
RMDIR	100%	100%	100%	100%	100%	89%	94%
SYMLINK	100%	97%	100%	97%	97%	90%	94%
TRUNCATE	100%	100%	100%	100%	100%	90%	95%
UNLINK	100%	100%	100%	100%	100%	88%	94%
UTIMENSAT	100%	96%	98%	99%	98%	87%	90%
TOTAL	**100%**	**99.3%**	**99.3%**	**99.7%**	**99.7%**	**90%**	**94.9%**

New applications?

- ➔ New workloads
  - ◆ Numerical simulation tends to be write driven and bandwidth limited, AI is read driven: latency becomes the new challenge.
    - Magnum I/O from NVIDIA to reduce the critical path and keep the GPU busy
  - ◆ On-Prem / Cloud interoperability, mix of on-premise and cloud dataflow require connectors and interoperable system
- ➔ New usages
  - ◆ Objects and files are two data API to support
  - ◆ Container, part of the interoperability discussion
  - ◆ Python, numerical apps and AI workload are now routinely scripted in Python

Outline

9:00am

- Infrastructure hardware: - 30 minutes -KC
  - Storage devices characteristics
  - Storage devices evolution
  - Importance of software in infrastructure
  - Resulting stack and standardization aspects
  - New applications
- Infrastructure software - 30 minutes - Sai
  - posix
  - mpi-io
  - netcdf
  - object
- Storage trend and possible futures
  - Deep and multi-tier storage hierarchy
  - Technical challenges
    - metadata, data policies, fault tolerance
    - perspective - Storage Class Memory

10:00am KC

- Introduction to Darshan - 30 minutes -
  - Why, Install, HOWTO
  - Darshan DXT

10:30am virtual break

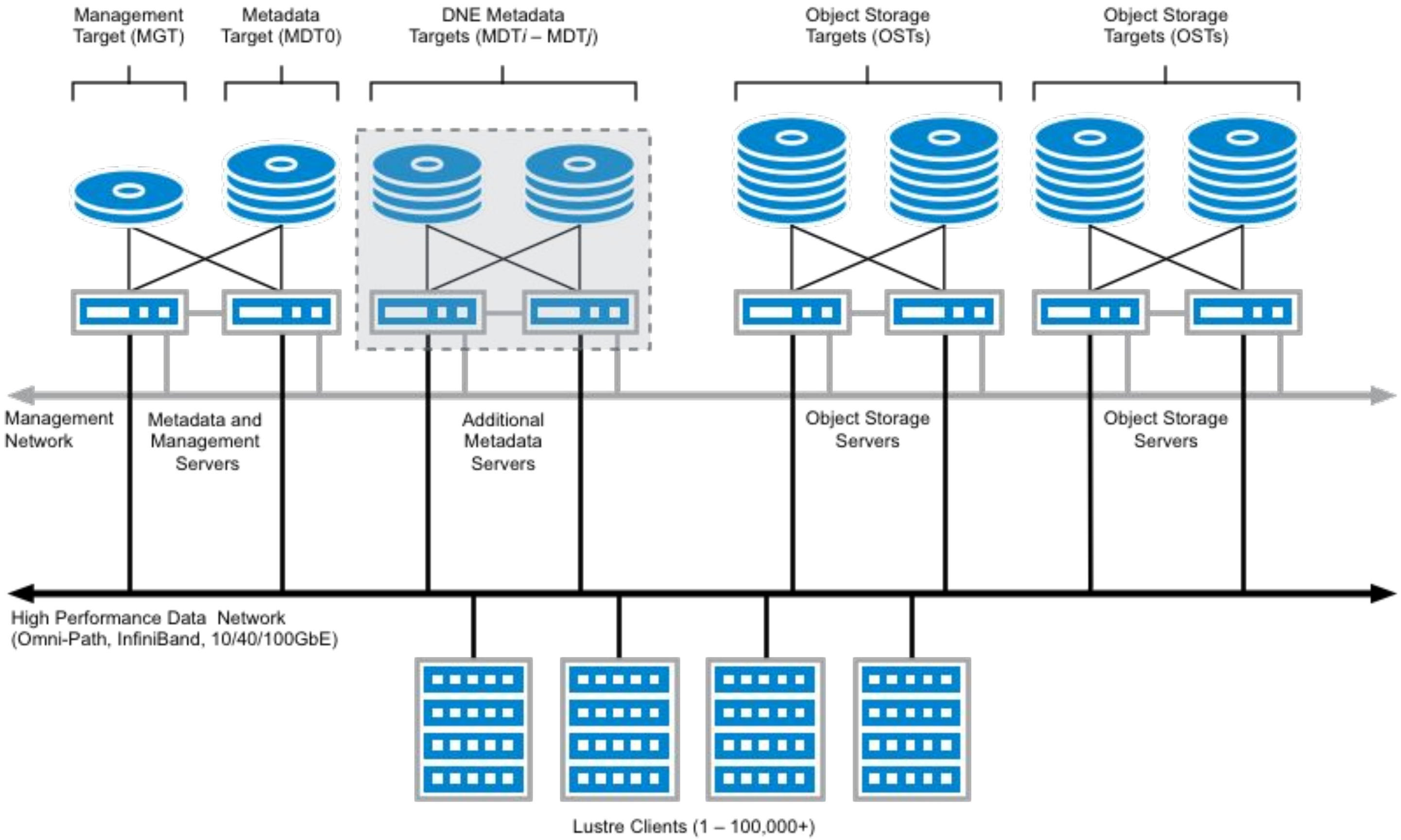
10:45am - KC

- Hands-on session - 1H -
  - 4 differents code to analyse

12:00 wrap-up

Infrastructure Software

- Parallel File Systems



- Posix compliant
- Data exposed to clients as hierarchical files

Ref: Lustre wiki

POSIX

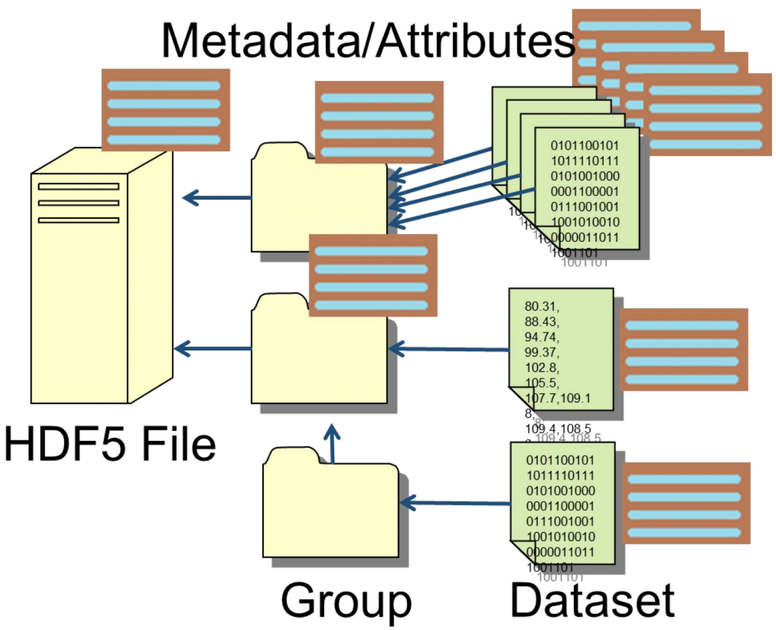
- POSIX Compliancy and problems
  - POSIX is an IEEE standard
  - POSIX I/O API and POSIX I/O Semantics
    - Read(), write(), open(), close(), etc
    - State: Open() before Read()/Write()
      - Overkill when there are millions of processes wanting to read/write a file
    - Prescriptive/inflexible metadata
      - All files in a directory have same metadata
      - Not easy to have additional data descriptions
    - Consistency
      - Read() always returns the latest write()
        - Write() required to block an application until its “committed”
        - Extreme performance penalty
    - Will be good to avoid these problems within HPC
      - One solution: Object stores

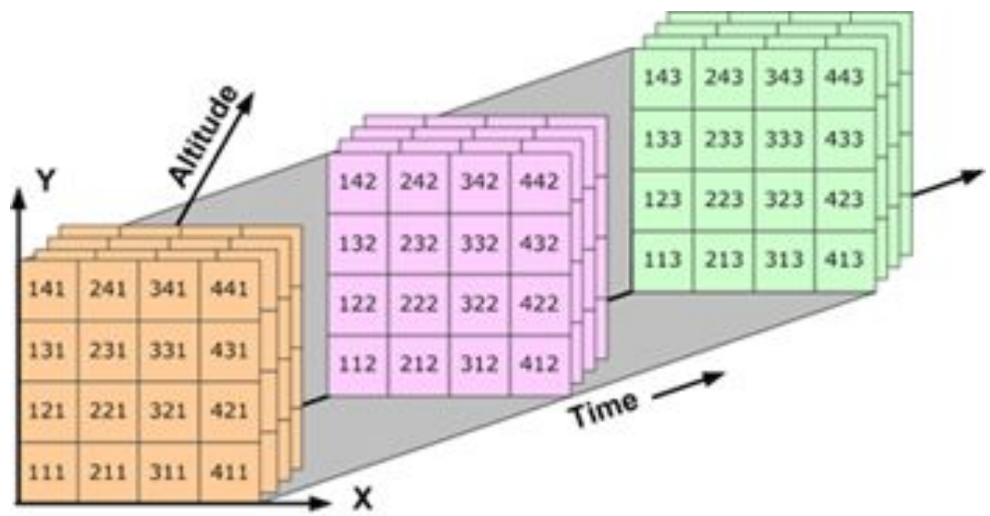
Object stores

- Organization of data as “objects” rather than hierarchical files
  - Without pre-defined structure, it’s a “flat” organization of data as objects
  - Objects can have any user defined metadata
  - Can overlay and impose any structure on the organization of data
    - For example: Hierarchical files as needed by POSIX, different data formats such as HDF5, NetCDF etc (described next)
  - Consistency can be relaxed & tunable
  - Key Value stores can be used to describe metadata (& can of course be very fast!)
  - Provides a foundation to build multiple “views” – such as POSIX, S3, HDF5

Data Formats (eg : HDF, NetCDF)

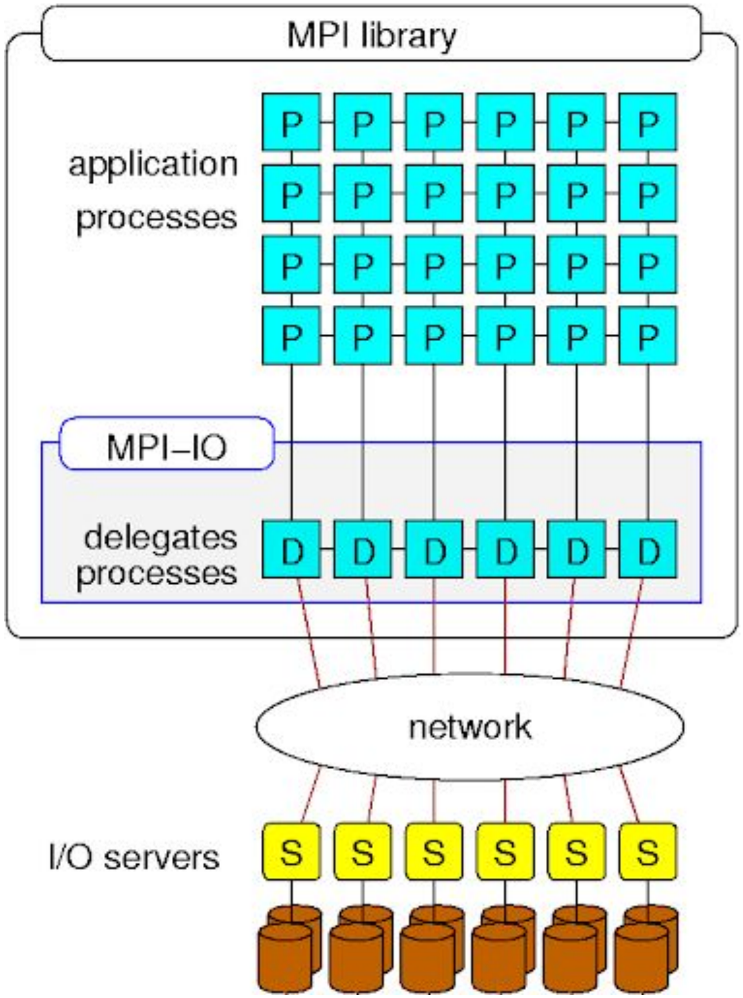
- Designed to store and manage large amounts of data
- Used a lot of represent data in the scientific community





MPI-IO

- Two fundamental ways to for parallel I/O in multi-process/Message Passing applications parallel I/O
  - File per process
  - Shared file
- MPI-IO provides a mechanism to access a parallel file system to store data from application processes
- Collective I/O



Ref: Northwesterm University

Storage Trends and Possible Futures

- NVRAM in I/O Stack
- Advent of Object stores in HPC
- In-Storage Computing
- Quality of Service
- Federation of data stores
- Advanced Telemetry
- A place for all storage types in HPC!

Deep and Multi-tier Storage Hierarchy

- Storage landscape is changing
- New storage devices (and memories) are now appearing on the scene
- How can the applications get the best out of persistent storage?
- How to Mix and match each of the different storage technologies to give the best performance for applications?
- We next look at the different individual storage technologies

Hierarchical Storage Systems

- Achieving as many possible performance/capacity points within a single storage system

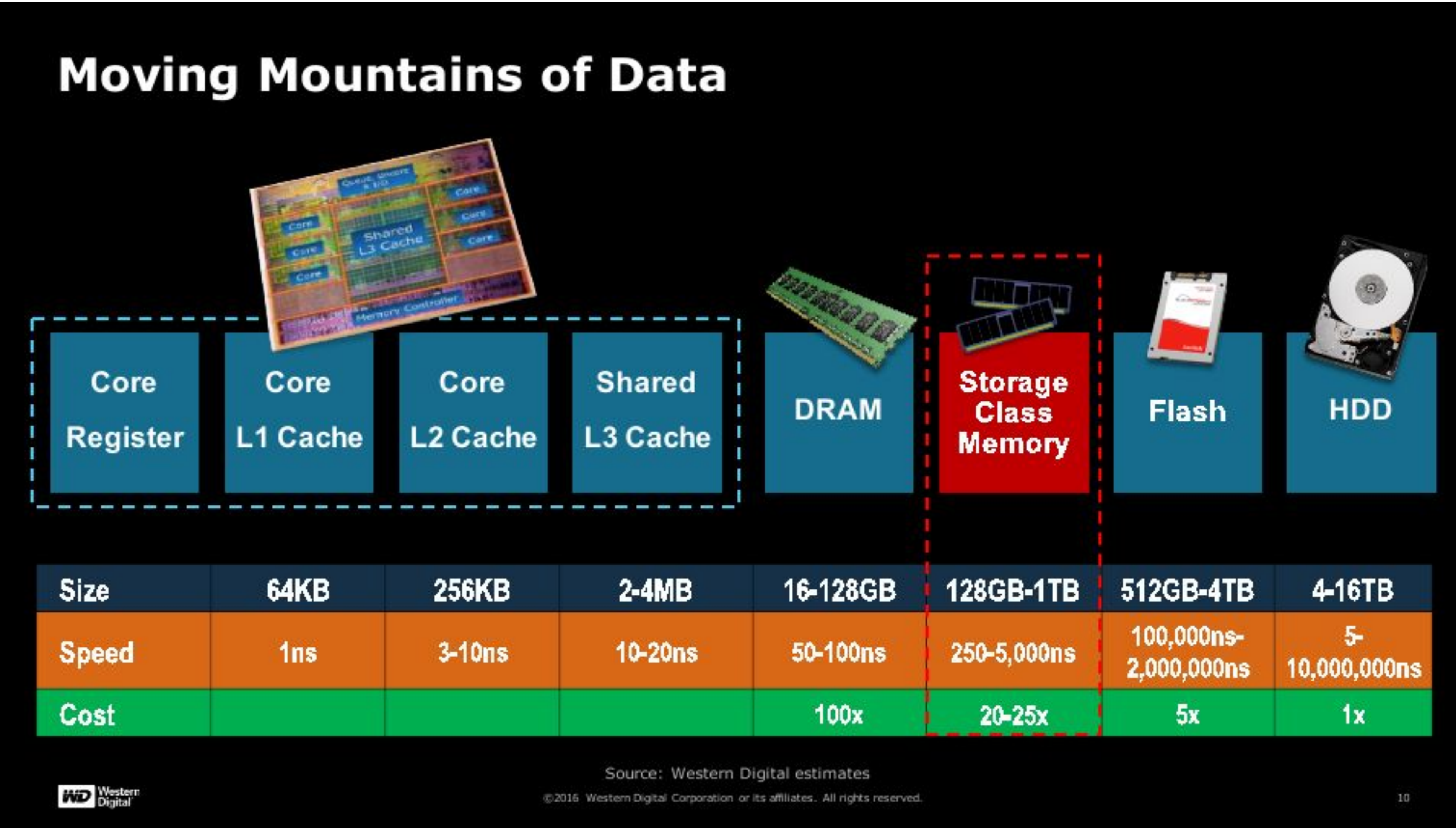
- The different tiers of storage devices technologies stacked in the same storage system
  - Organized as “Tiers”

- Applicable to a wide variety of workloads
  - Eg: Archival workloads can use the lowest most tiers
  - Transactional workloads can use the highest performance tiers

- Can use a combination of infrastructure software
  - Eg: Parallel File systems, Object Stores, Tape file systems

- Data moved between the tiers based on policy
  - User driven
  - Machine Learning based (Automated)

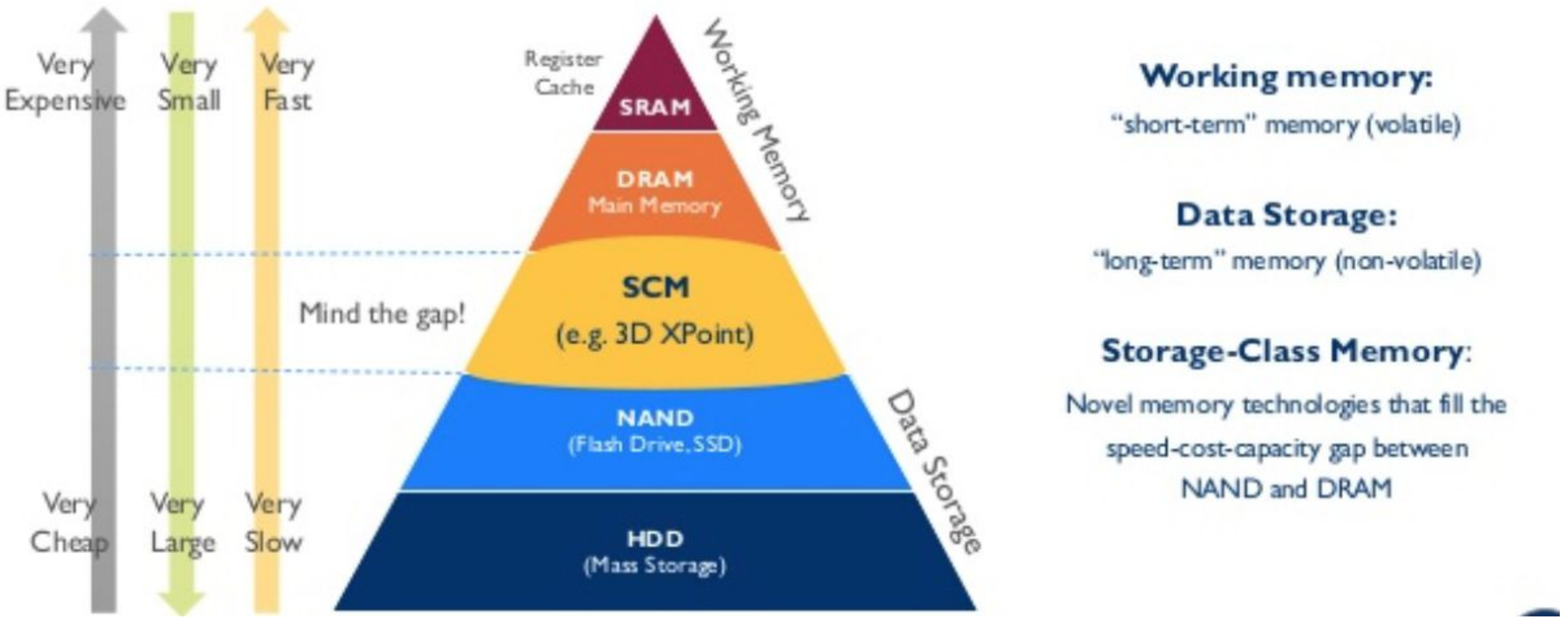
Arrival of Storage Class Memories



	Core Register	Core L1 Cache	Core L2 Cache	Shared L3 Cache	DRAM	Storage Class Memory	Flash	HDD
Size	64KB	256KB	2-4MB	16-128GB	128GB-1TB	512GB-4TB	4-16TB	
Speed	1ns	3-10ns	10-20ns	50-100ns	250-5,000ns	100,000ns-2,000,000ns	5-10,000,000ns	
Cost				100x	20-25x	5x	1x	

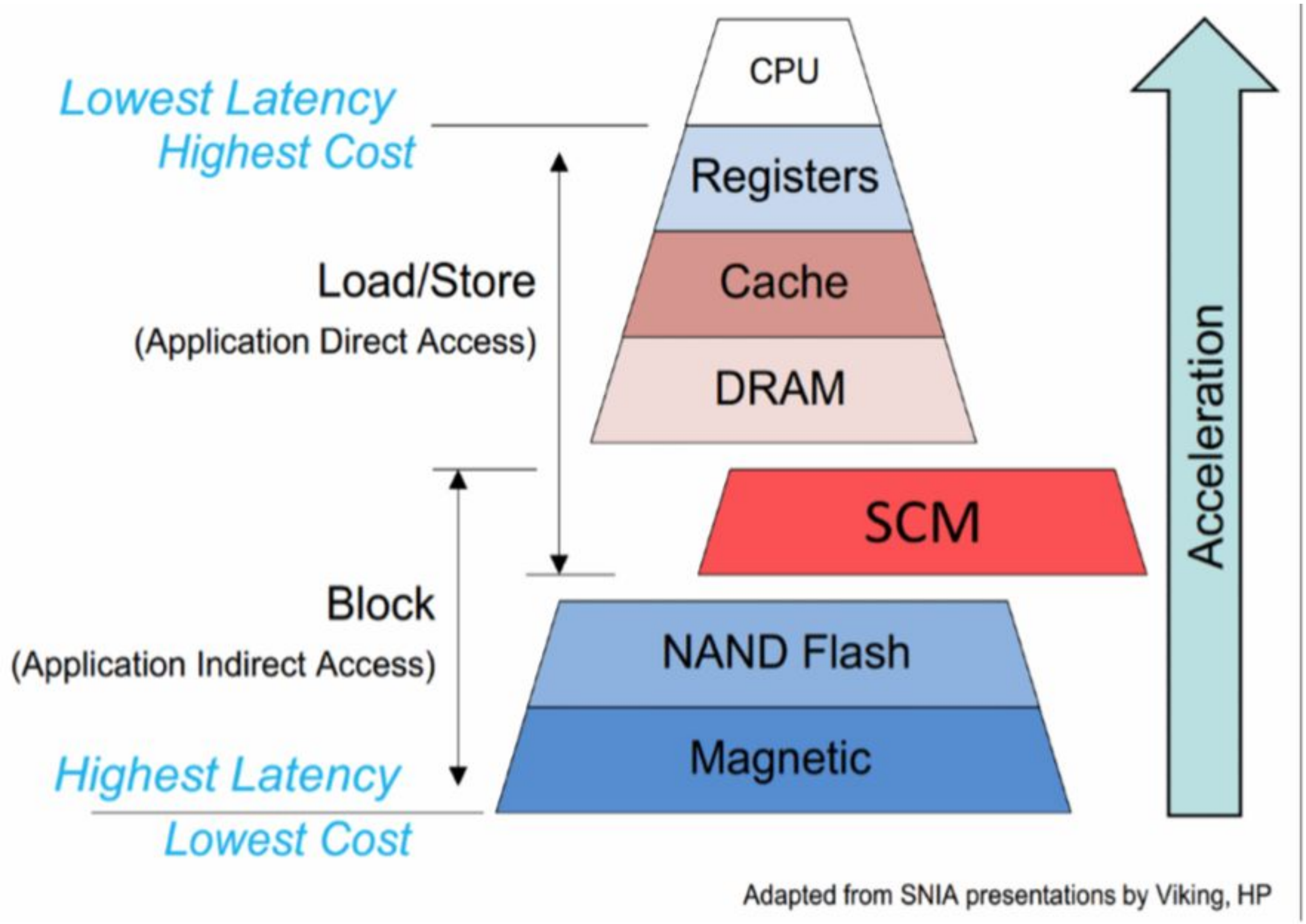
Ref: Western Digital

SCM in the stack



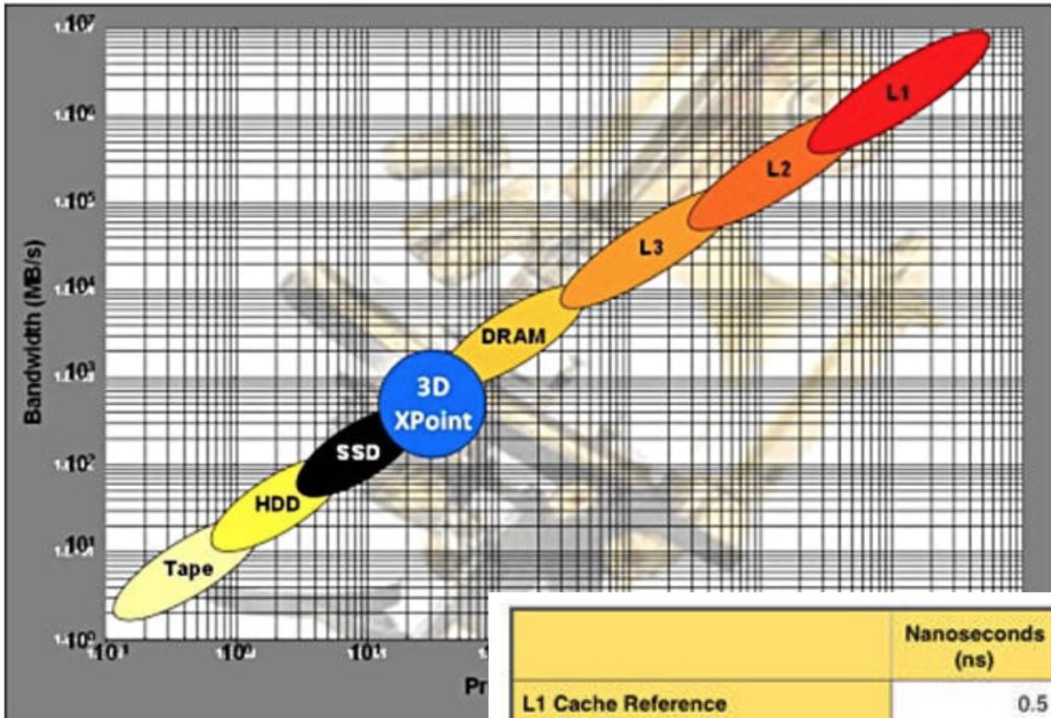
Ref: Status of Memory industry report , 2019

SCM Usage



Ref: Flash Memory Summit

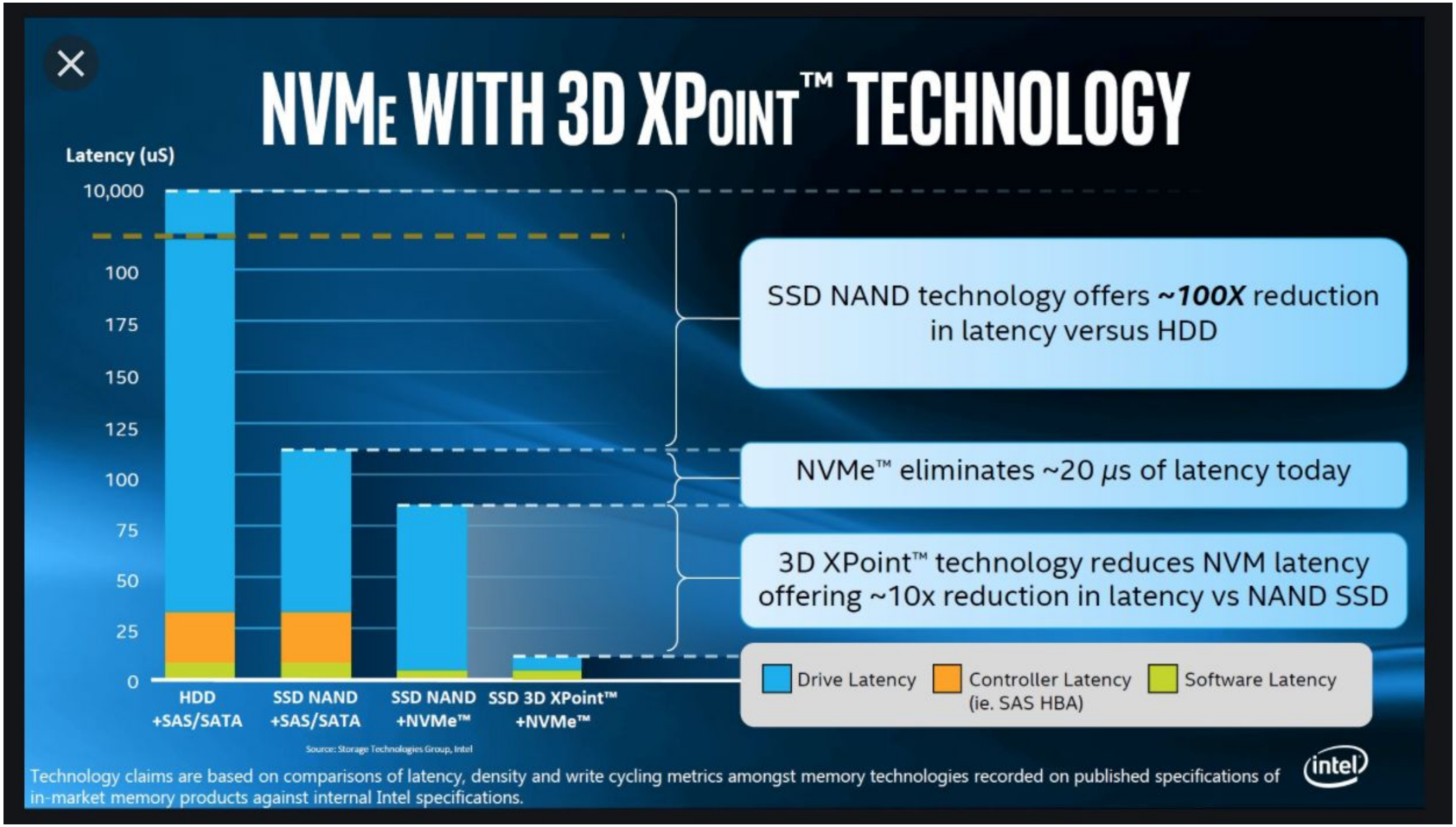
3DXPoint Technology (SCM)



	Nanoseconds (ns)	Microseconds (µs)	Milliseconds (ms)	If L1 Access is 1 second
L1 Cache Reference	0.5			1 sec
L2 Cache Reference	7			14 secs
DRAM Access	200			6 mins, 40 secs
Intel Octane 3D XPoint	7,000	7		3 hours, 53 mins, 20 secs
Micron 9100 NVMe PCIe SSD Write	30,000	30		16 hours, 40 mins
Mangstor NX NVMeF Array Write	30,000	30		16 hours, 40 mins
DSSD D5 NVMeF Array	100,000	100		2 days, 7 hours, 33 mins, 20 secs
Mangstor NX NVMeF Array Read	110,000	110		2 days, 13 hours, 6 mins, 40 secs
NVMe PCIe SSD Read	110,000	110		2 days, 13 hours, 6 mins, 40 secs
Micron 9100 NVMe PCIe SSD Read	120,000	120		2 days, 18 hours,m40 mins
Disk Seek	10,000,000	10,000	10	7 months, 10 days, 11 hours, 33 mins, 20 secs
DAS Disk Access	100,000,000	100,000	100	6 years, 4 months, 19 hours, 33 mins, 20 secs
SAN Array Access	200,000,000	200,000	200	9 years, 6 months, 2 days, 17 hours, 20 mins

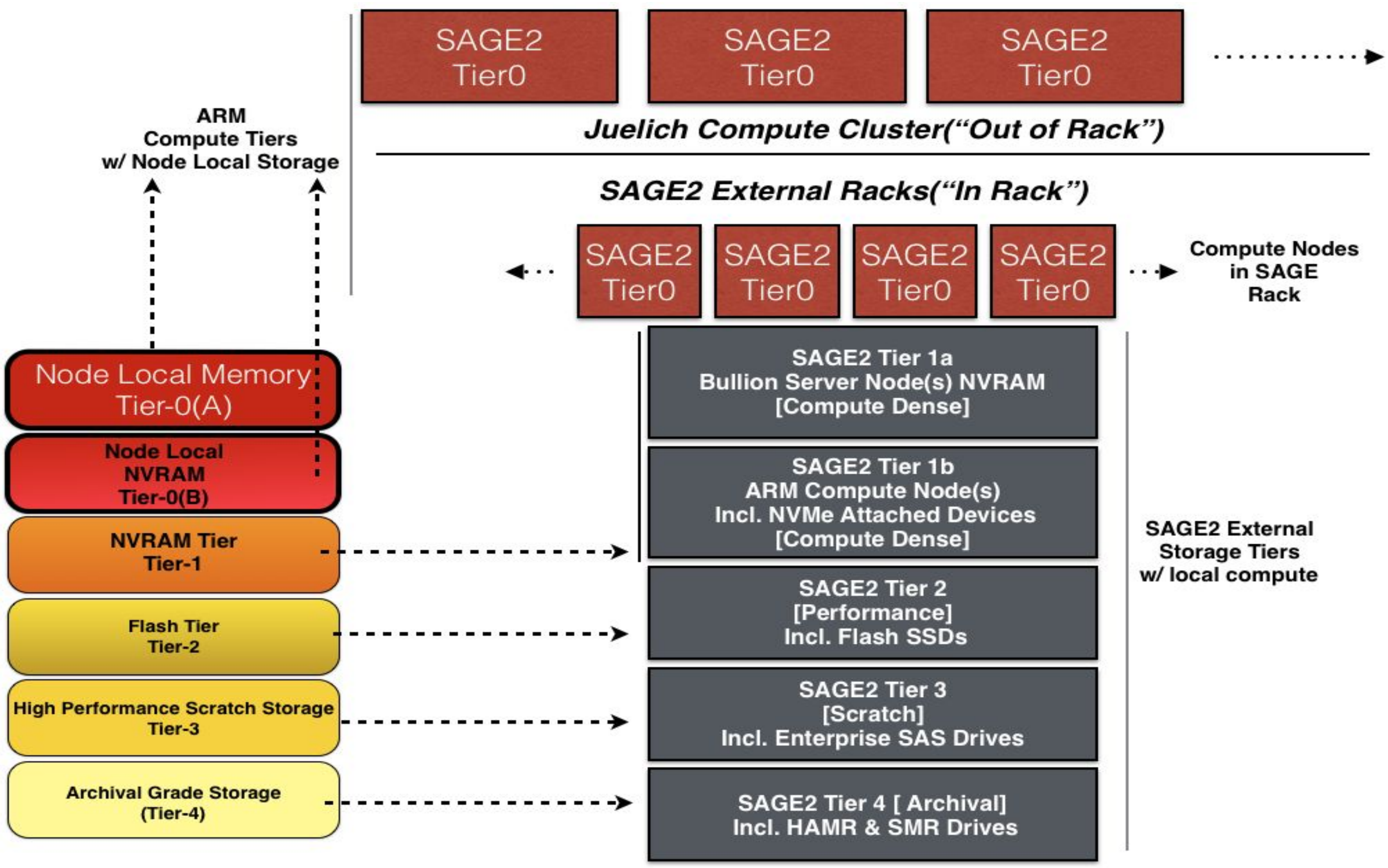
Ref: The Register

3DXPoint Performance



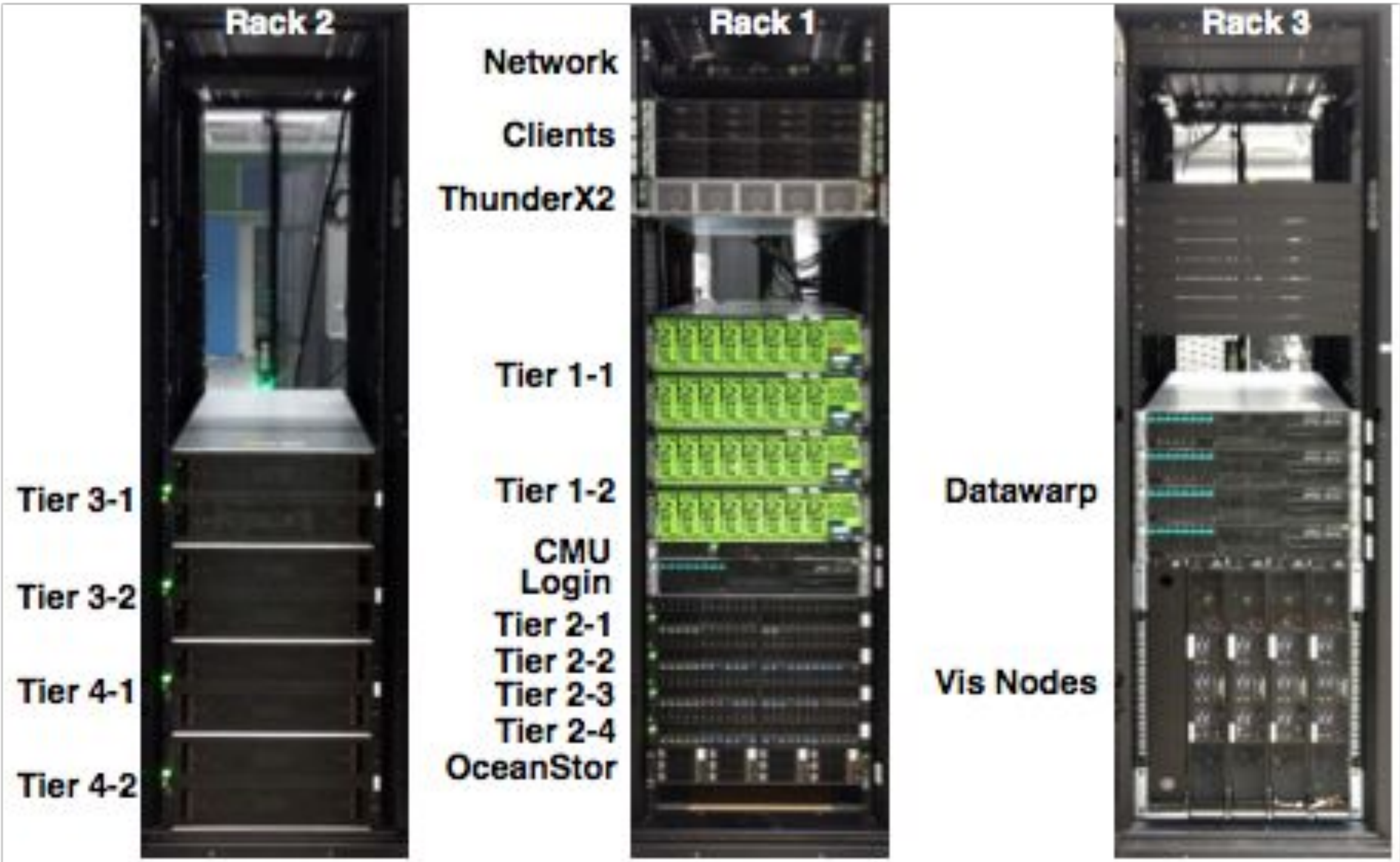
Ref: Intel

Example Hierarchical Storage System
SAGE System at Juelich

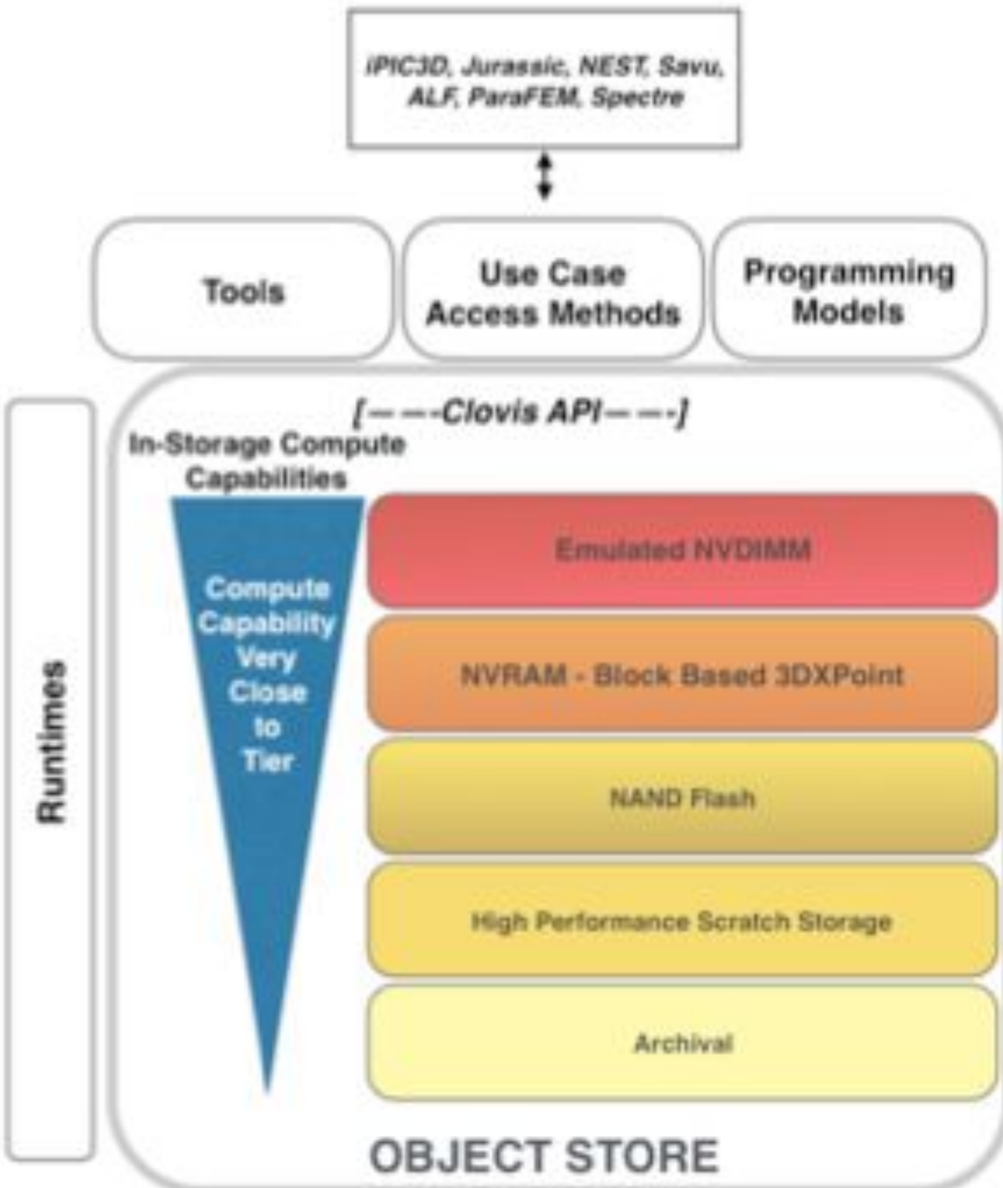


Example Hierarchical Storage System

SAGE System at Juelich



Hierarchical Storage – SAGE Stack Example

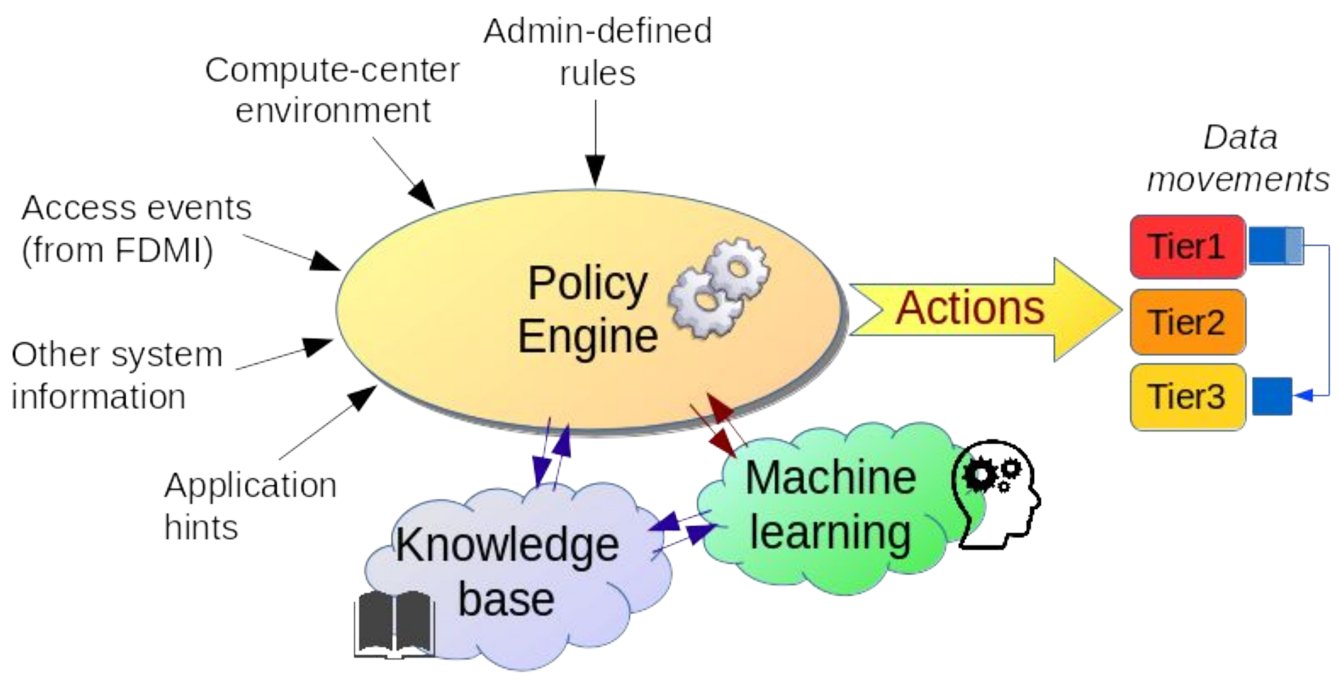


Technical Challenges in Hierarchical Storage Systems

- Fault Tolerance
  - Hierarchical storage systems can have multiple possible faults (Different storage tiers have different fault tolerance characteristics)
  - Infrastructure needed to handle
    - Software failures (handled within file systems, object stores, etc)
    - Hardware Failures
      - Storage devices & techs have their own techs (eg: RAID for HDD)
      - Network RAID, PDRAID, & Erasure coding

Technical Challenges in Hierarchical Storage Systems

- Data Policies (Key questions)
  - How long to retain data in a tier?
  - When to migrate the data to a lower tier?
  - How to deal with Tape tiers?
  - Usage of specialized Hierarchical storage managers (example below, used in SAGE)



Outline

9:00am

- Infrastructure hardware: - 30 minutes -KC
  - Storage devices characteristics
  - Storage devices evolution
  - Importance of software in infrastructure
  - Resulting stack and standardization aspects
  - New applications
- Infrastructure software - 30 minutes - Sai
  - posix
  - mpi-io
  - netcdf
  - object
- Storage trend and possible futures
  - Deep and multi-tier storage hierarchy
  - Technical challenges
    - metadata, data policies, fault tolerance
    - perspective - Storage Class Memory

10:00am KC

- Introduction to Darshan - 30 minutes -
  - Why, Install, HOWTO
  - Darshan DXT

10:30am virtual break

10:45am - KC

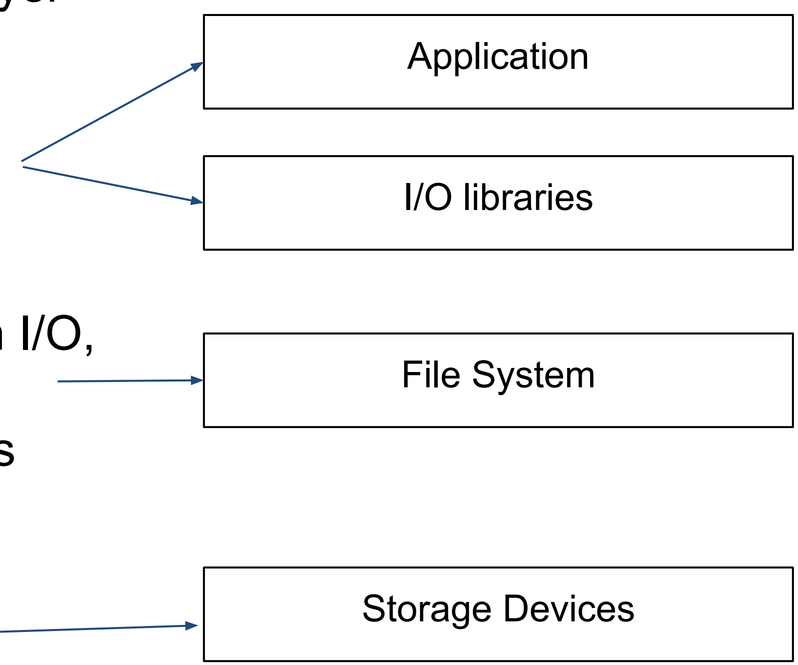
- Hands-on session - 1H -
  - 4 differents code to analyse

12:00 wrap-up

I/O tracing and monitor possibilities

- I/O path consist of several layers
  - Different options to monitor each layer
- Applications level tracing
  - Tools: Darshan, Scalatrace etc.
- File System instrumentation
  - Not always the same as application I/O, libraries may modify the I/O pattern
  - Tools: Lustre and GPFS diagnostics
- Block device instrumentation
  - I/O requests to the actual devices
  - Tools: Linux block tracing, etc.

Typical I/O stack



I/O stack tuning

- I/O the access pattern in all layer impacts performance

- Applications developers
  - Control the I/O request from the Application to the I/O libraries and the file system
  - Can **not** control how file system will internally translate their I/O requests
    - However, the **fadvice()** to pass hints for access patterns
- File System developers / System Admins
  - Control file system request to storage devices

Typical I/O stack

Application

I/O libraries

fadvise()

File System

Storage Devices

Application level I/O monitoring with Darshan

- What is Darshan
  - Name means “sight” or “vision” in Sanskrit
  - Lightweight, scalable I/O characterization tool
  - Transparently captures application I/O access pattern information
  - Open source library and runtime
    - Developed and maintained at Argonne National Laboratory

Key features

- Captures several I/O interfaces
  - POSIX I/O, MPI-IO, and limited HDF5 and PNetCDF
- Instrumentation on compile time or at run time
- Compatible with popular compilers and MPI implementations
- File system agnostic
  - Can be used with any file system
- Does not impact application performance in measurable way
  - Use it on production runs
- No need for applications code modification

Components

- **darshan-runtime**
  - Used to capture I/O statistics while the application is running
  - Installed on an HPC system to instrument MPI applications
    - Installation steps vary depending on the platform
- **darshan-util**
  - Use to annayle Darshan log files
  - Installed on a workstation to analyze Darshan log files
    - (log files themselves are portable)
  - Installation is generic for almost any unix-like platform

Compilation and Installation process

- System-wide (available to all users)
- User's home directory (no root access required)
  - There is no difference in functionality
- Download source code from
  - https://www.mcs.anl.gov/research/projects/darshan/download/
- `tar -zxvf darshan-$version.tar.gz`
- Compile Darshan runtime, use the same compiler as your application
  - `cd darshan-$version/darshan-runtime`
  - `./configure CC=mpicc --prefix=$installation-dir --with-log-path-by-env=DARSHAN_LOGPATH --with-jobid-env=NONE --with-mem-align=128`
  - `make && make install`
- Compile Darshan util
  - `cd darshan-$version/darshan-util`
  - `./configure --prefix=$installation-dir`
  - `make && make install`

How to use it

- The simplest method to use Darshan is to build a dynamic executable that is dynamically linked with the MPI library
    - To determine if your executable is dynamic or not:
        - `ldd a.out`
        - `libmpi.so.1 => /$inst_path/libmpi.so.1 [ ... ]`
- Set log path directory
    - `export DARSHAN_LOGPATH=./`
- Then prefix the MPI execution command with the Darshan library
    - `LD_PRELOAD=$path/libdarshan.so mpirun -np 4 a.out`
- Each job instrumented with Darshan produces a single log file
    - Application must call MPI_Finalize() to generate the log file
- Darshan command line utilities are used to analyze these log files
- Online doc:
    - https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-runtime.html

Log file analysis tools

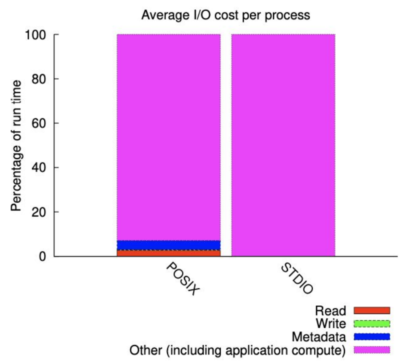
- **darshan-job-summary.pl**
  - creates pdf file with graphs useful for initial analysis
  - packages needed: Perl, pdflatex, epstopdf, and gnuplot
- **darshan-summary-per-file.sh**
  - similar to above, but creates a separate pdf file for each file opened by the application
- **darshan-parser**
  - dumps all information into ascii (text) format

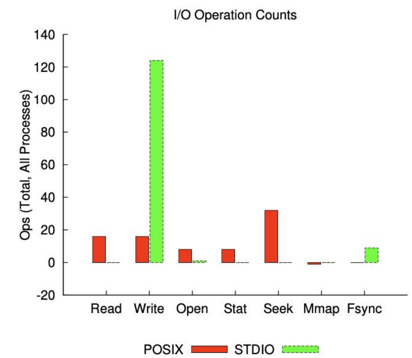
- Online documentation at
  - https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html

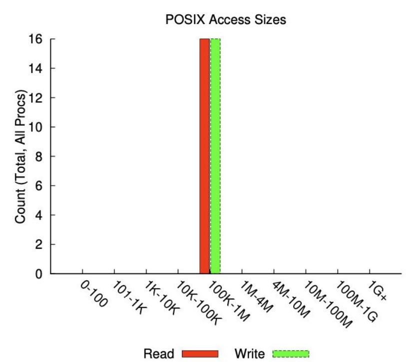
Darshan job summary example

jobid: 202545	uid: 1008	nprocs: 4	runtime: 1 seconds

I/O performance *estimate* (at the POSIX layer): transferred 8.0 MiB at 101.99 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 2.71 MiB/s







Darshan job parser example

#<module><rank><record id> <counter> <value><file name> <mount pt> <fs type>

POSIX	-1	22..36	POSIX_OPENS	8	file	/home	nfs4
POSIX	-1	22..36	POSIX_FILENOS	0	file	/home	nfs4
POSIX	-1	22..36	POSIX_DUPS	0	file	/home	nfs4
POSIX	-1	22..36	POSIX_READS	16	file	/home	nfs4
POSIX	-1	22..36	POSIX_WRITES	16	file	/home	nfs4
POSIX	-1	22..36	POSIX_SEEKS	32	file	/home	nfs4
POSIX	-1	22..36	POSIX_STATS	8	file	/home	nfs4
POSIX	-1	22..36	POSIX_MMAPS	-1	file	/home	nfs4
POSIX	-1	22..36	POSIX_FSYNCS	0	file	/home	nfs4
POSIX	-1	22..36	POSIX_MODE	436	file	/home	nfs4
POSIX	-1	22..36	POSIX_BYTES_READ	4194304	file	/home	nfs4

Darshan eXtended Tracing (DXT) module

- "Advanced" Darshan to report every intercepted call
- Not on by default, to enable
    - export DXT_ENABLE_IO_TRACE=1
- I/O Traces appear as a time series
- Special tool for post process analysis
    - darshan-dxt-parser
- Provide tools for applying different types of analyses to the logs.
- Provides different levels of granularity
    - DXT_TRIGGER_CONF_PATH environment variable to notify DXT of the path of the configuration file
        - file triggers: trace files based on regex matching of file paths
        - rank triggers: trace files based on regex matching of ranks
        - dynamic triggers: trace files based on runtime analysis of I/O characteristics (e.g., frequent small or unaligned I/O accesses

darshan-dxt-parser example output

```
# ***************************************************
# DXT_POSIX module data
# ***************************************************

# DXT, file_id: 16457598720760448348, file_name: /tmp/test/testFile
# DXT, rank: 0, hostname: shane-thinkpad
# DXT, write_count: 4, read_count: 4
# DXT, mnt_pt: /, fs_type: ext4
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX       0  write       0                           0       262144      0.0029       0.0032
 X_POSIX       0  write       1          262144      262144      0.0032       0.0035
 X_POSIX       0  write       2          524288      262144      0.0035       0.0038
 X_POSIX       0  write       3          786432      262144      0.0038       0.0040
 X_POSIX       0   read       0                           0       262144      0.0048       0.0048
 X_POSIX       0   read       1                262144      262144      0.0049       0.0049
 X_POSIX       0   read       2                524288      262144      0.0049       0.0050
 X_POSIX       0   read       3                786432      262144      0.0050       0.005
```

Other darshan tools

- **darshan-convert:**
  - converts an existing log file to the newest log format
- **darshan-diff:**
  - provides a text diff of two Darshan log files, comparing both job-level metadata and module data records between the files
- **darshan-analyzer:**
  - walks an entire directory tree of Darshan log files and produces a summary of the types of access methods used in those log files
- **dxt_analyzer**:
  - plots the read or write activity of a job using data obtained from Darshan's DXT modules (if DXT is enabled)

Hands on tutorial

- Download virtual machine
    - https://rb.gy/n82oex
- Download sample applications
    - https://github.com/kchasapis/esiwace_demo_darshan

Guidelines for optimizing I/O

- Large request size
- Avoid single shared file for parallel file systems
- Sequential I/O performs always better
- For MPI-I/O Collective I/O results in better performance

The ESiWACE2 is on Zenodo, the Open Access repository for our results
https://zenodo.org/communities/esiwace

Interested in getting in touch?
Twitter: https://twitter.com/esiwace
Website: www.esiwace.eu

ESiWACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988