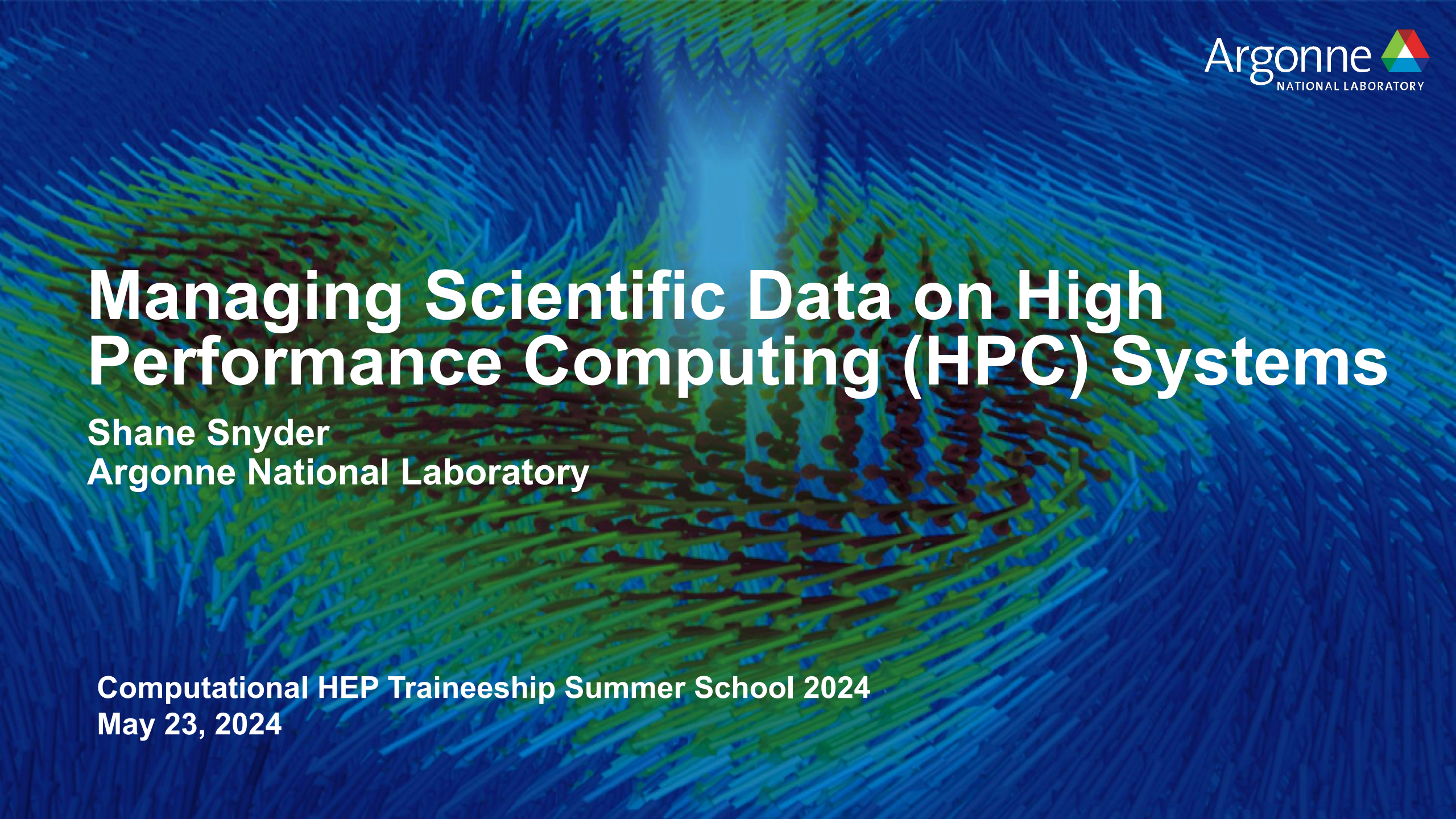


# Managing Scientific Data on High Performance Computing (HPC) Systems

Shane Snyder
Argonne National Laboratory

Computational HEP Traineeship Summer School 2024
May 23, 2024

# Managing scientific data

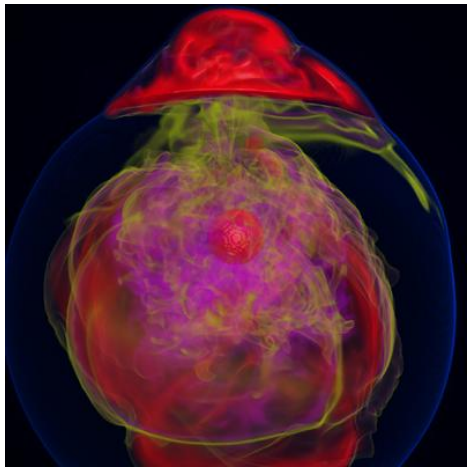
HPC applications span various scientific disciplines and have a range of diverse data management needs.

- An explosion of scientific data (both in terms of volume and in diversity of access patterns) is compounding the *I/O bottleneck*, a longstanding performance impediment on HPC systems.

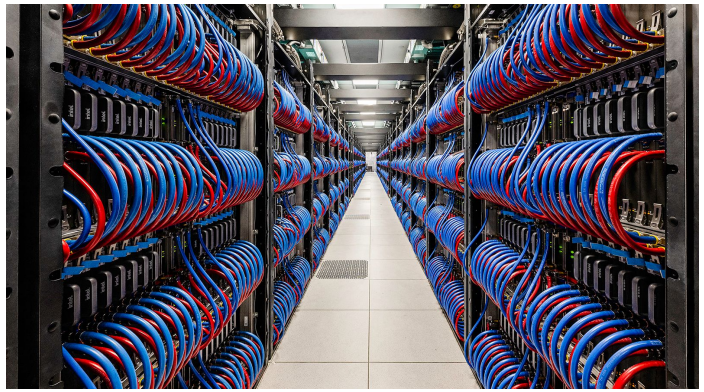
Meanwhile, hardware trends have enabled novel, high-performance storage system designs that promise increased productivity to HPC apps.

HPC facilities deploy vast amounts of storage resources to help meet the I/O needs of scientific applications.

- Today, I'll introduce the basics of the HPC I/O stack and how it can be used to efficiently manage/access data.



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma's visualization team at UC Davis.



HPE/Cray Aurora system at the ALCF

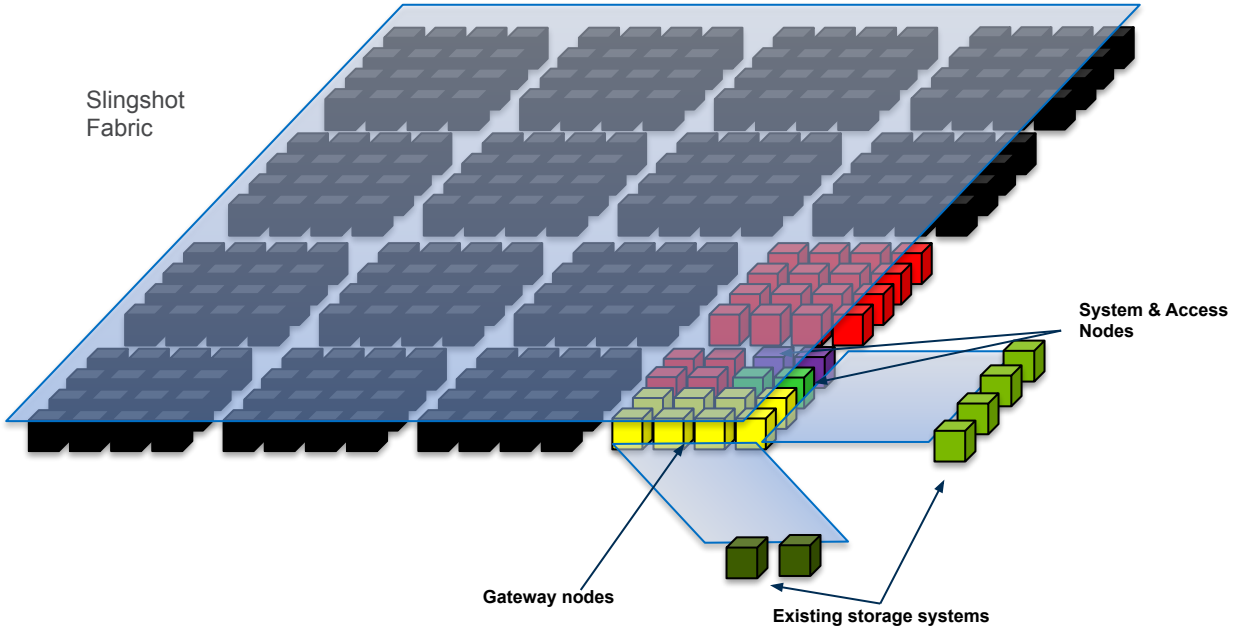
# An example HPC system: ALCF Aurora

Aurora is the new exascale HPC system at the ALCF:

- 10,624 compute nodes (CPU+GPU)
- HPE Slingshot high speed interconnect
- On-fabric DAOS storage
  - 220 PB @ ≥ 25 TB/s
- Connectivity to shared Lustre file storage
  - 100 PB @ 650 GB/s

Typically, HPC systems like Aurora are used by large jobs (i.e., spanning many nodes) attempting to solve complex science problems by aggregating lots of computational power.

- The MPI message passing library has traditionally been used to enable communication/coordination across the processes comprising these large jobs.



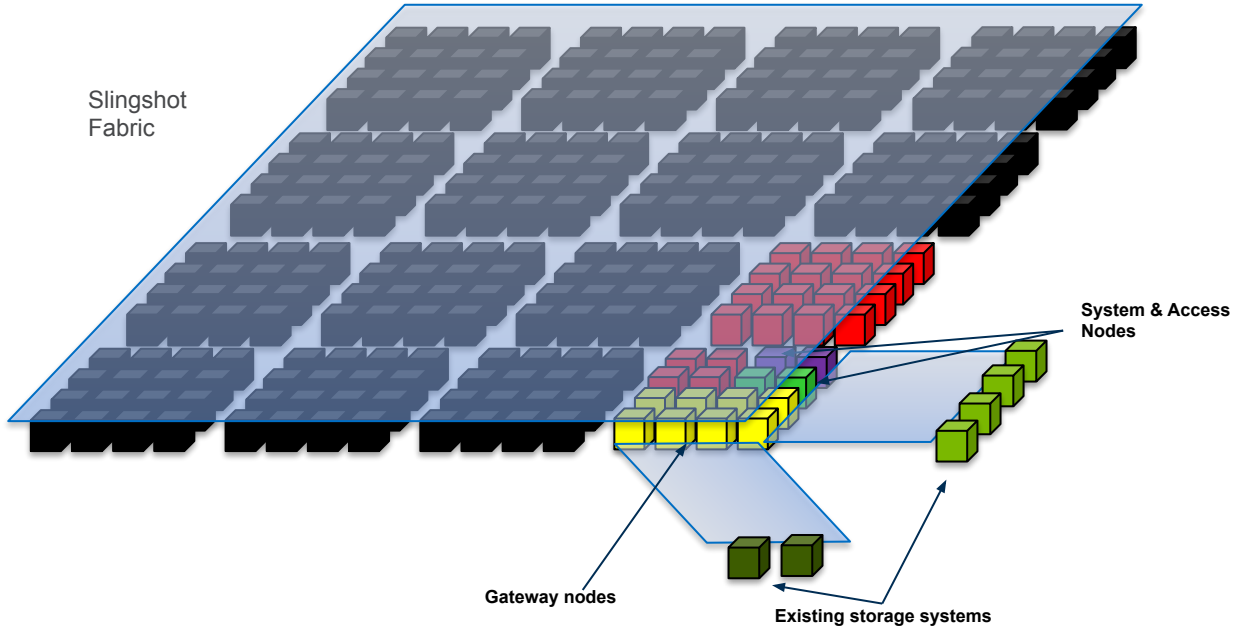
# An example HPC system: ALCF Aurora

Aurora is the new exascale HPC system at the ALCF:

- 10,624 compute nodes (CPU+GPU)
- HPE Slingshot high speed interconnect
- On-fabric DAOS storage
  - 220 PB @ ≥ 25 TB/s
- Connectivity to shared Lustre file storage
  - 100 PB @ 650 GB/s

Typically, HPC systems like Aurora are used by large jobs (i.e., spanning many nodes) attempting to solve complex science problems by aggregating lots of computational power.

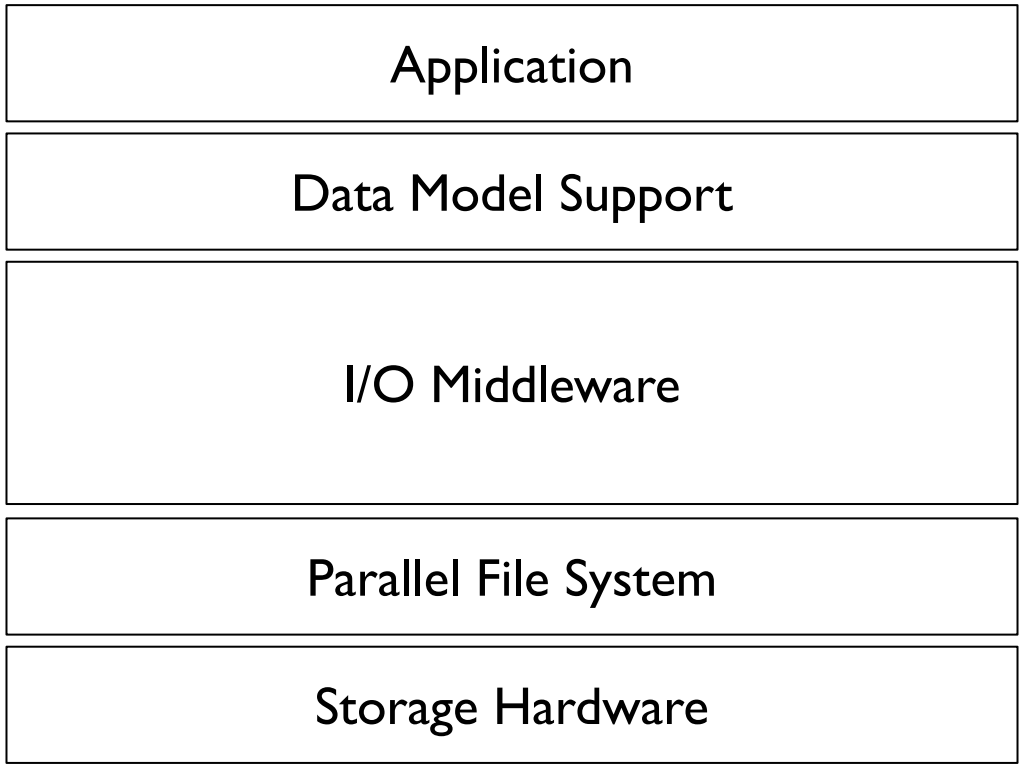
- The MPI message passing library has traditionally been used to enable communication/coordination across the processes comprising these large jobs.



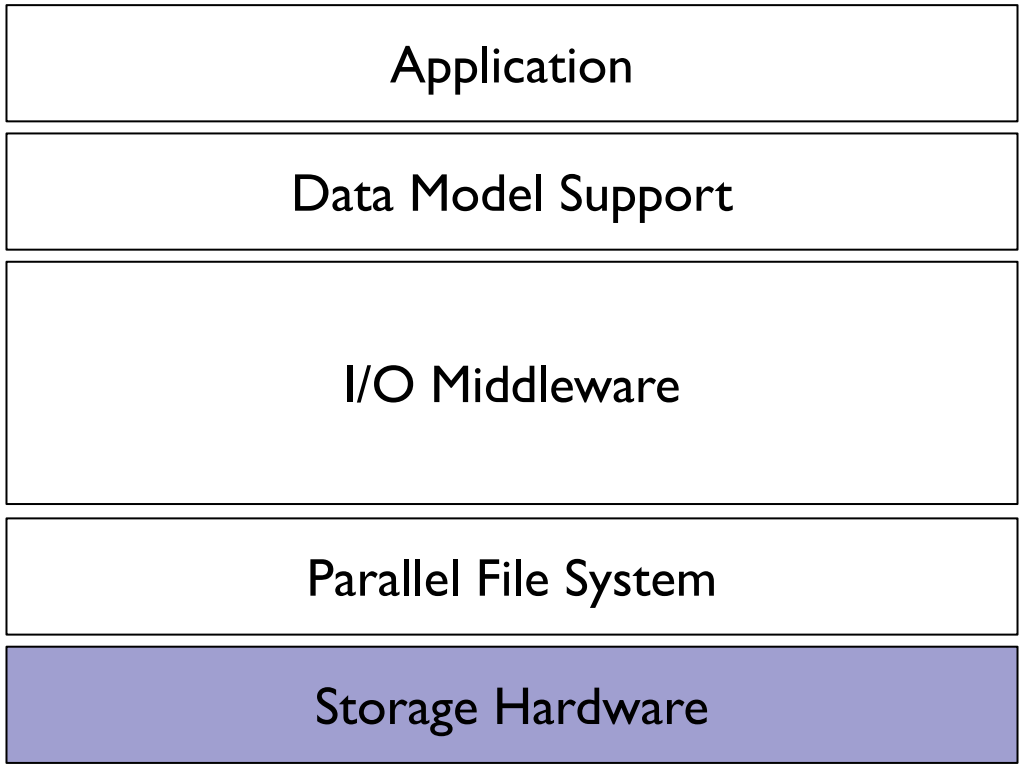

**This historical generality is becoming less true as HPC takes on more diverse computational workloads, e.g.:**

- **AI/ML**
- **Workflow systems (e.g., HEP)**

# The HPC I/O stack



# The HPC I/O stack




**Storage Hardware** stores raw bytes on different storage devices.

*HDDs, SSDs*

# HPC storage hardware

Traditionally, HPC storage systems were exclusively based on large arrays of hard disk drives (**HDDs**).

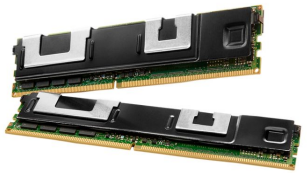
- Still commonly used in higher capacity storage tiers

Higher performance flash-based storage (i.e., **SSDs**) has become increasingly prevalent in HPC storage systems in recent years.

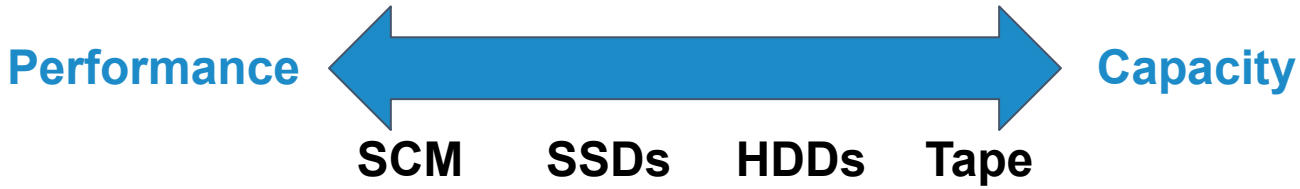
- Originally, deployed as a smaller performance storage tier, but decreasing costs have recently enabled all-flash storage systems


Storage class memory (**SCM**) is now being integrated into the HPC storage hierarchy.

- Performance somewhere in between that of SSDs and of DRAM
- Used as an accelerated storage component in a hybrid storage system



**Tape** storage is often offered as a cost-effective storage option for archival data.



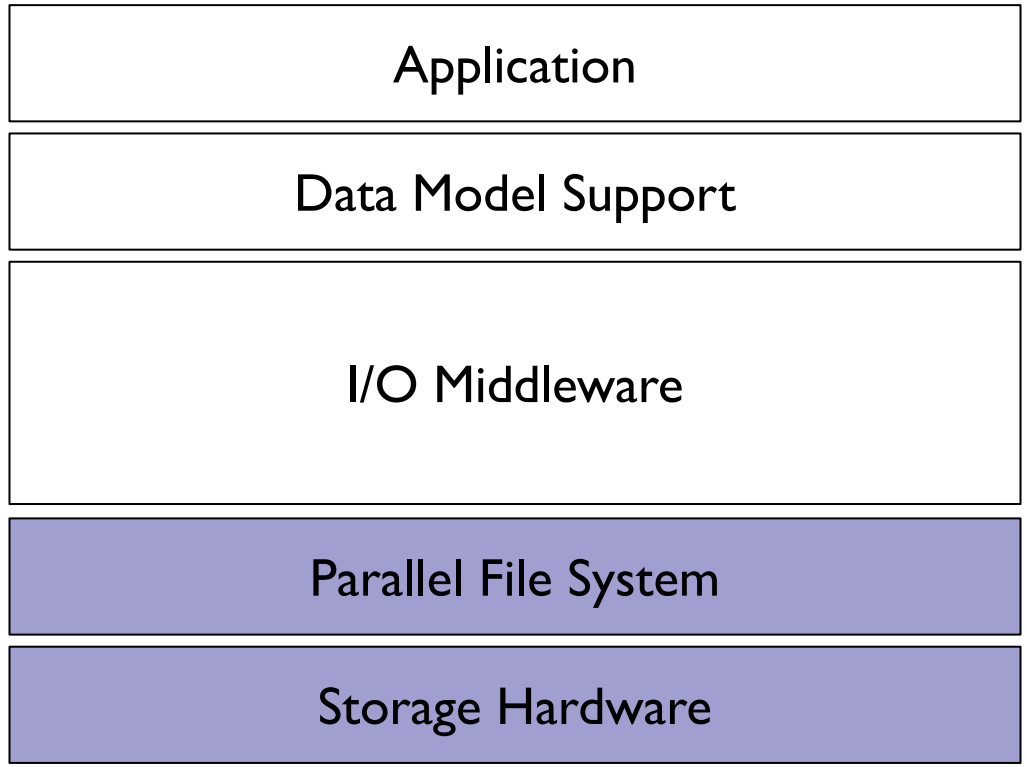
# HPC storage hardware

HPC storage systems are usually deployed as massive appliances that provide all necessary storage hardware/software and that can be scaled out to meet I/O demands of the system:

- All-disk, all-flash, hybrid, and tiered storage hardware
- Networking infrastructure for internal storage/management and for external connection to the HPC system
- Parallel file system software (e.g., Lustre)
- Storage system management and profiling software

HPE/Cray ClusterStor storage appliance

# The HPC I/O stack




**Parallel file system** maintains logical file model and provides efficient access to data using a POSIX-like interface.

*Lustre, GPFS*

**Storage Hardware** stores raw bytes on different storage devices.

*HDDs, SSDs*

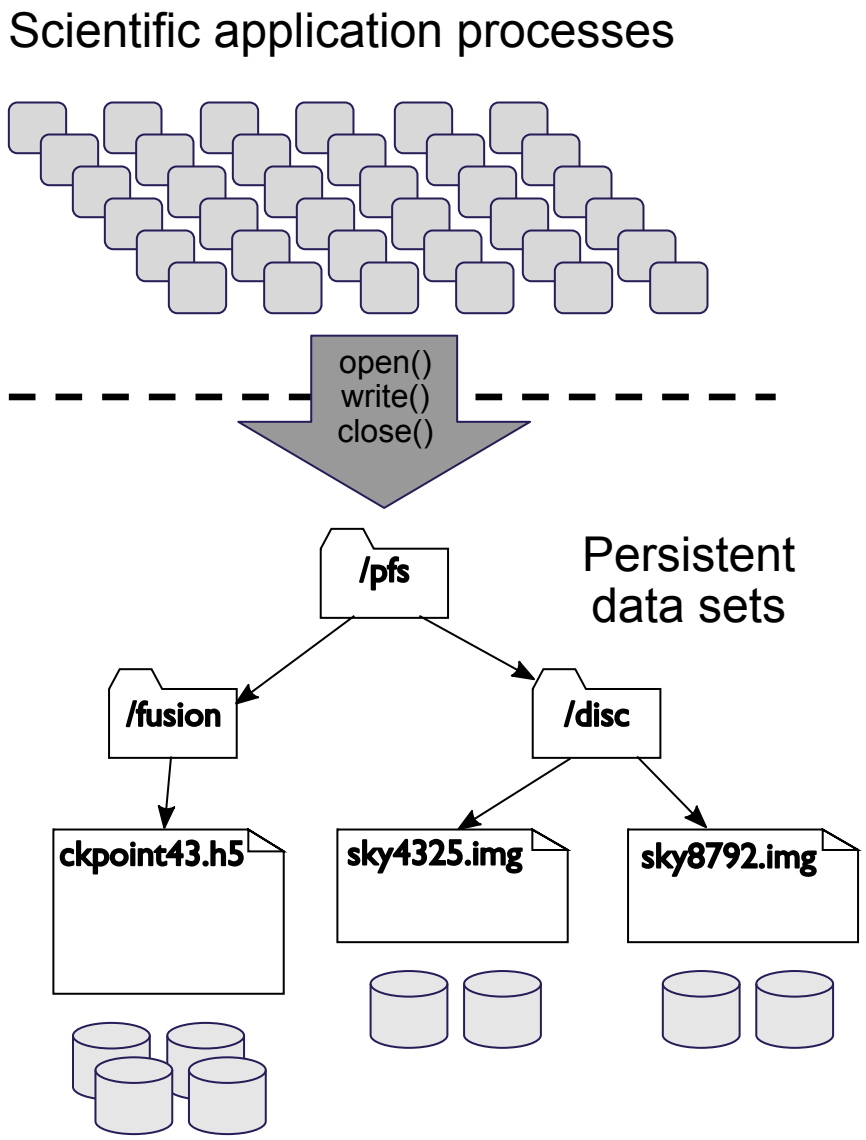
# Parallel file systems

Parallel file systems (PFSes) have been traditionally offered by HPC facilities as a one-size-fits-all solution for storing users data.

- Users interact with a familiar file/directory storage hierarchy, but with much more aggregate capacity and performance relative to a local file system.

PFSes offer a number of attractive characteristics that have led to their widespread usage in HPC:

- **High performance** - parallel I/O paths enable aggregate performance of many storage resources using high speed interconnects
- **Scalability** - storage resources can be scaled to meet demands of current and future applications
- **Reliability** - failover mechanisms to ensure availability of data in face of failures



# Parallel file systems: Lustre

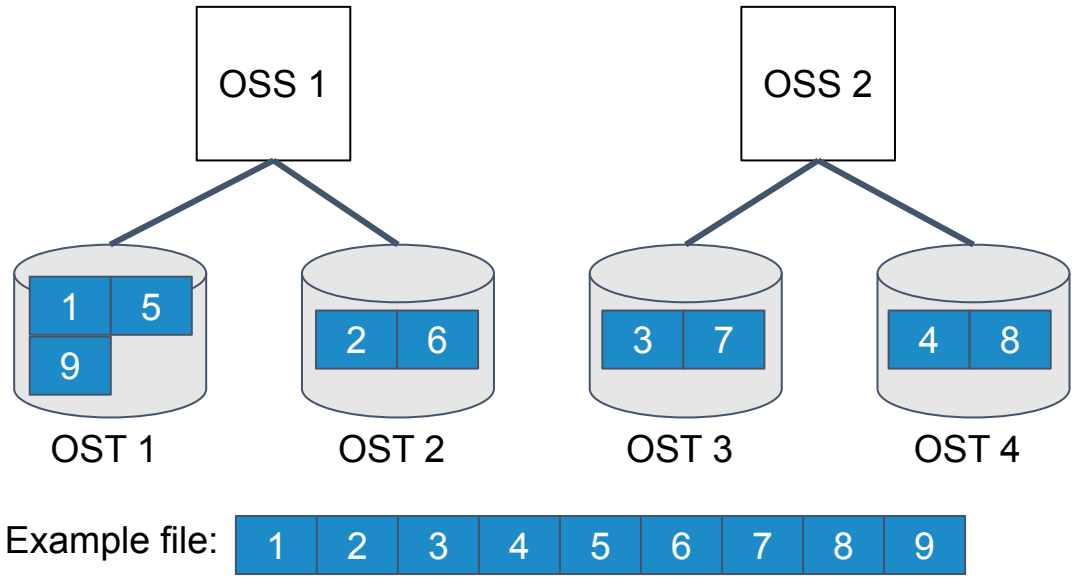
**Lustre** is a high-performance, scalable PFS that is commonly deployed on HPC systems.

Lustre's design is centered around a metadata storage service and an object storage service:

- Metadata servers (MDSes) manage metadata targets (MDTs).
  - Maintain filesystem namespace and key file metadata
- Object storage servers (OSSes) manage object storage targets (OSTs).
  - Provide bulk storage for file data

Lustre clients coordinate with metadata servers to set/query file layout, but then interact strictly with storage servers for reading/writing data.

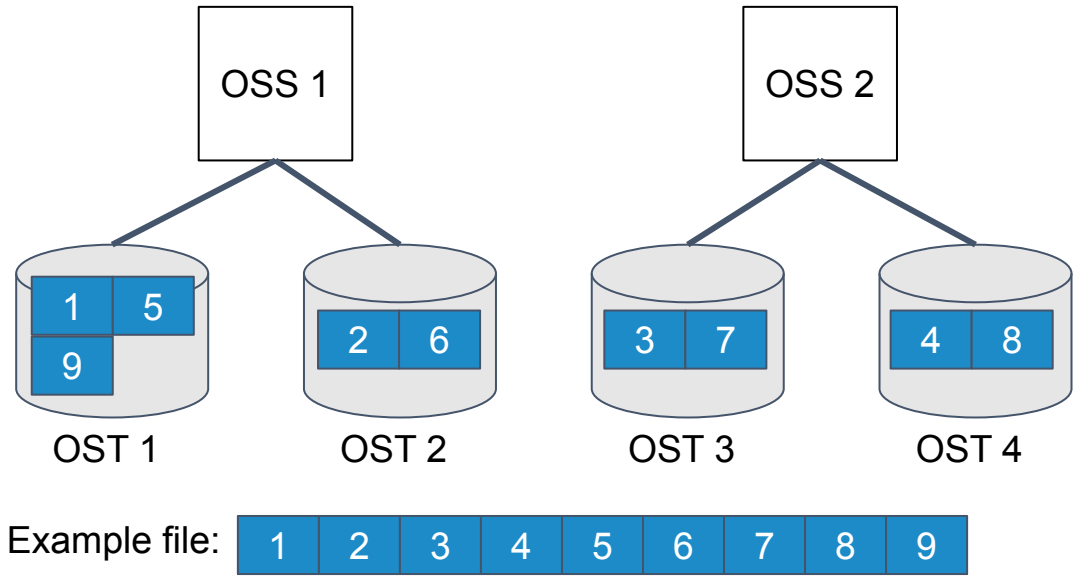
Lustre files are broken into *stripes*, with file stripes round-robin distributed over 1 or more OSTs.



# Parallel file systems: Lustre

**Lustre file striping is something that users can often tune directly for each file to achieve better I/O performance:**

- Larger files typically benefit from being stored across many stripes (i.e., more storage resources).
- Smaller files typically benefit from being stored on a single stripe (i.e., less overhead from accessing multiple servers).



# Parallel file systems: the POSIX interface

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a familiar POSIX-like interface.

**POSIX** (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations for raw file access
  - `fopen()`, `fread()`, `fseek()`, `fclose()` operations for buffered file access (often used for text)
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

# Parallel file systems: the POSIX interface

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a familiar POSIX-like interface.

**POSIX** (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations for raw file access
  - `fopen()`, `fread()`, `fseek()`, `fclose()` operations for buffered file access (often used for text)
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

**This semantic is tricky to enforce for PFSes where potentially hundred of thousands of clients collectively access and cache file contents.**

# Parallel file systems: the POSIX interface

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a familiar POSIX-like interface.

**POSIX** (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations for raw file access
  - `fopen()`, `fread()`, `fseek()`, `fclose()` operations for buffered file access (often used for text)
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

POSIX was never designed or necessarily intended for parallel file access.

- Inflexible, strong consistency requirements often lead PFSes to implement elaborate locking protocols or to eschew strong consistency entirely.

# Parallel file systems: the POSIX interface

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a familiar POSIX-like interface.

**POSIX** (Portable Operating System Interface)

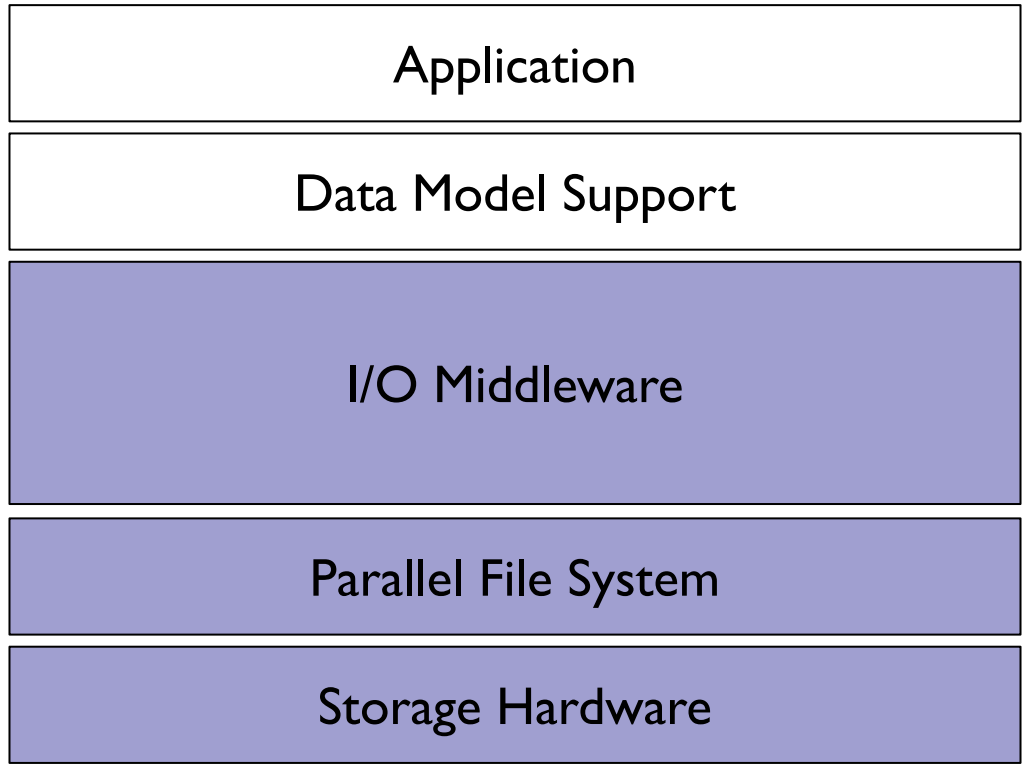
- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations for raw file access
  - `fopen()`, `fread()`, `fseek()`, `fclose()` operations for buffered file access (often used for text)
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

POSIX was never designed or necessarily intended for parallel file access.

- Inflexible, strong consistency requirements often lead PFSes to implement elaborate locking protocols or to eschew strong consistency entirely.

**To avoid performance or consistency issues, best practice in the HPC community typically involves avoiding concurrent access of overlapping regions of a file. However, "false sharing" can still lead to performance inefficiencies.**

# The HPC I/O stack




**Parallel file system** maintains logical file model and provides efficient access to data using a POSIX-like interface.

*Lustre, GPFS*

**I/O Middleware** organizes and transforms accesses from many processes, especially those using collective I/O.

*MPI-IO*

**Storage Hardware** stores raw bytes on different storage devices.

*HDDs, SSDs*

# I/O middleware: parallel I/O capabilities

I/O middleware bridges the gap between application data abstractions and the storage system, typically transforming application accesses into optimized storage system requests.

The **MPI-IO** interface was designed to help address needs for parallel I/O support by HPC applications.

- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO enables flexible and performant I/O strategies:

- Independent operations
- Collective operations
- Non-blocking operations
- Optimizations
  - General and system-specific

# I/O middleware: parallel I/O capabilities

I/O middleware bridges the gap between application data abstractions and the storage system, typically transforming application accesses into optimized storage system requests.

The **MPI-IO** interface was designed to help address needs for parallel I/O support by HPC applications.

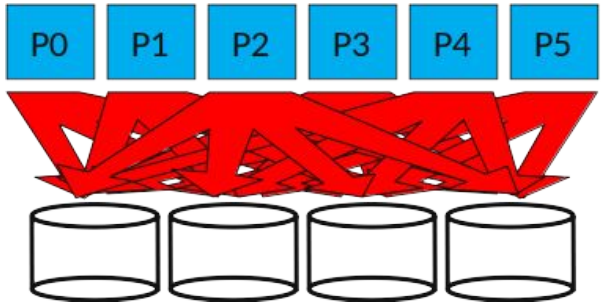
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO enables flexible and performant I/O strategies:

- **Independent operations**
- Collective operations
- Non-blocking operations
- Optimizations
  - General and system-specific



# I/O middleware: parallel I/O capabilities

I/O middleware bridges the gap between application data abstractions and the storage system, typically transforming application accesses into optimized storage system requests.

The **MPI-IO** interface was designed to help address needs for parallel I/O support by HPC applications.

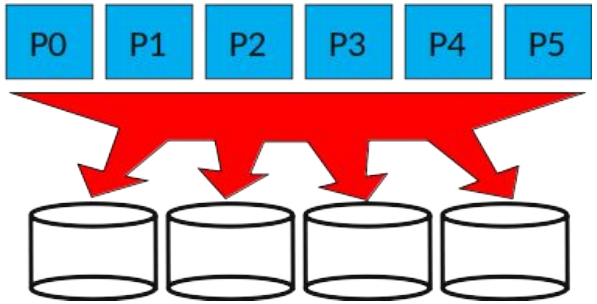
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO enables flexible and performant I/O strategies:

- Independent operations
- **Collective operations**
- Non-blocking operations
- Optimizations
  - General and system-specific



# I/O middleware: parallel I/O capabilities

I/O middleware bridges the gap between application data abstractions and the storage system, typically transforming application accesses into optimized storage system requests.

The **MPI-IO** interface was designed to help address needs for parallel I/O support by HPC applications.

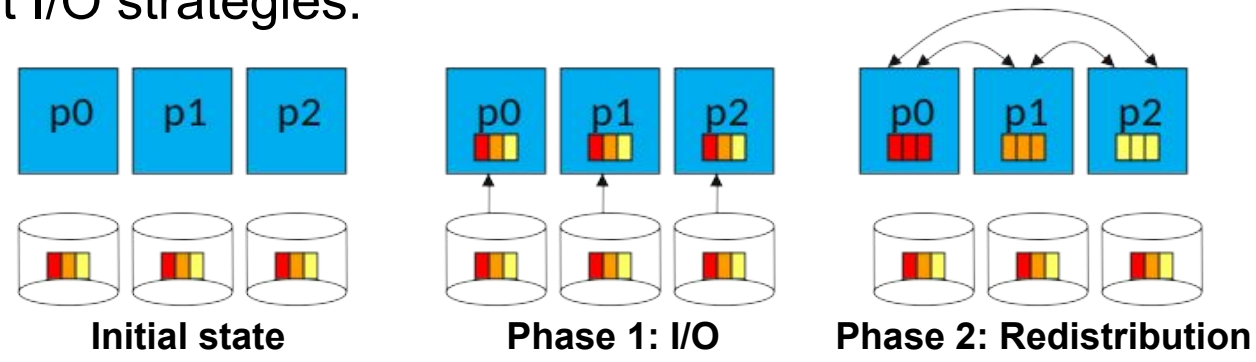
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO enables flexible and performant I/O strategies:

- Independent operations
- Collective operations
- Non-blocking operations
- **Optimizations**
  - General and system-specific




Two-phase collective I/O algorithm

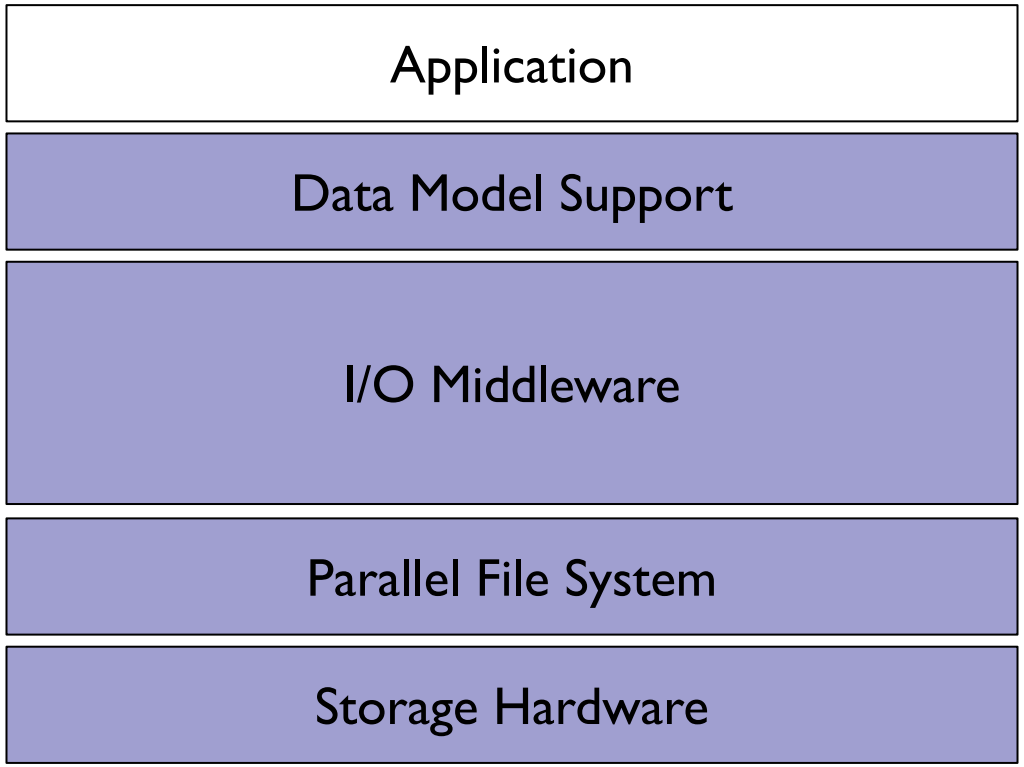
# The HPC I/O stack

**Data Model** maps application abstractions onto storage abstractions and provides data portability.

*HDF5, Parallel netCDF*

**Parallel file system** maintains logical file model and provides efficient access to data using a POSIX-like interface.

*Lustre, GPFS*




**I/O Middleware** organizes and transforms accesses from many processes, especially those using collective I/O.

*MPI-IO*

**Storage Hardware** stores raw bytes on different storage devices.

*HDDs, SSDs*

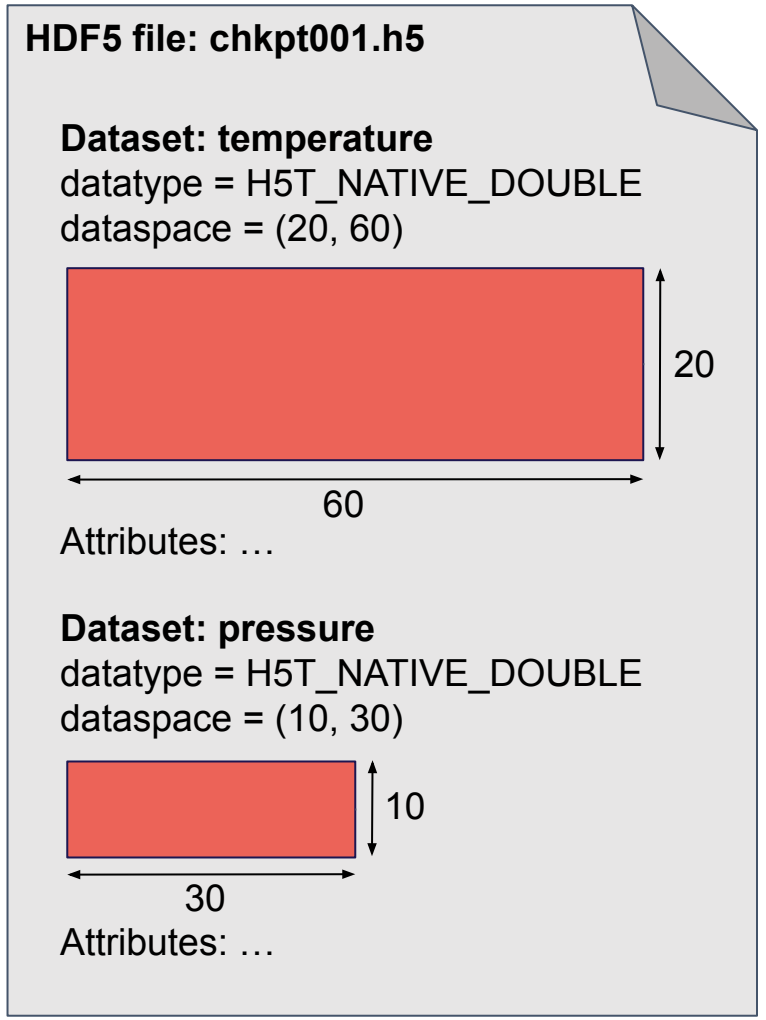
# Data models: scientific data abstractions

MPI-IO is a step in the right direction, but application scientists often prefer richer data management abstractions than simple files.

- Storing independent data products in unique files or manually serializing collections of data products into a single file is often untenable.

**HDF5** is a popular data management library and file format that specializes in storing large volumes of scientific data.

- Enable storage of multi-dimensional datasets, attributes, etc. in an HDF5 file (more like a “container”)
- Interfaces allow for access of individual dataset elements, subarrays, or entire datasets
- Support for collective I/O (using MPI-IO) or independent I/O (using MPI-IO or POSIX)
- VOL layer allows abstract implementation of storage for HDF5 objects
  - e.g., using async operations, using an object store rather than file system

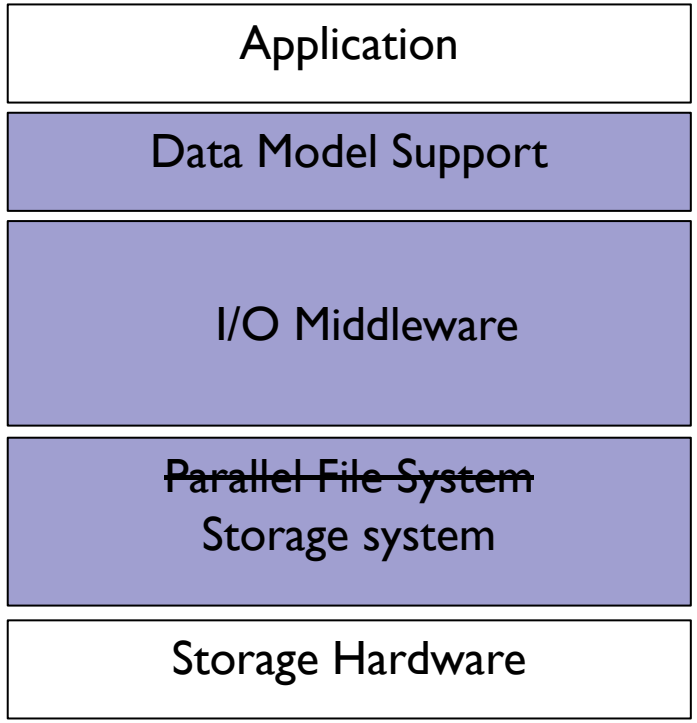


# Emerging storage technologies: DAOS

HPC storage technology is changing to meet the needs of diverse application workloads and to embrace emerging storage trends.

ALCF Aurora features Intel **DAOS**, a first-of-a-kind object-based storage system for large-scale HPC platforms.

- Leverages both SCM and SSDs for high-performance storage



# Emerging storage technologies: DAOS

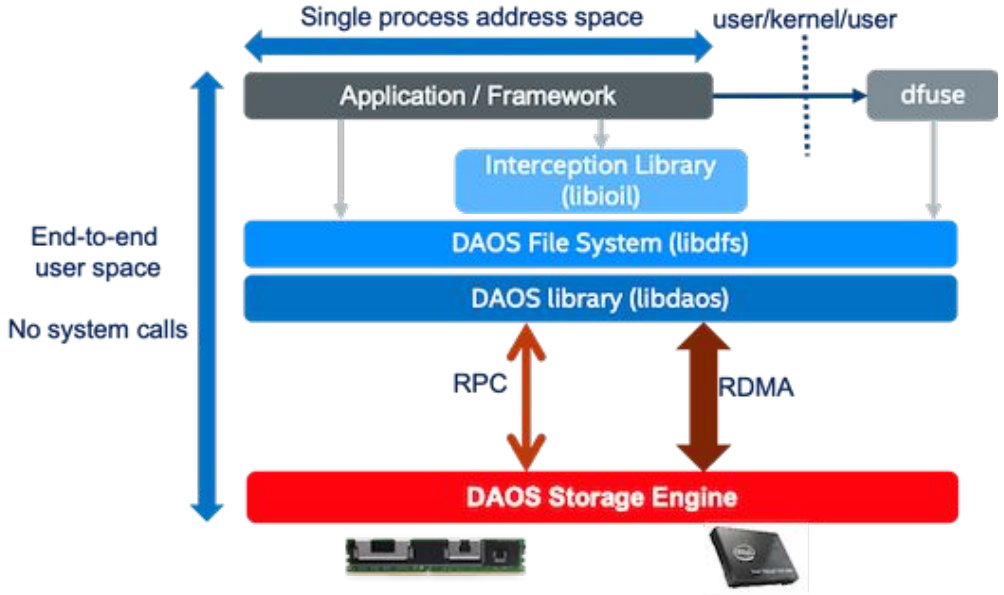
HPC storage technology is changing to meet the needs of diverse application workloads and to embrace emerging storage trends.

ALCF Aurora features Intel **DAOS**, a first-of-a-kind object-based storage system for large-scale HPC platforms.

- Leverages both SCM and SSDs for high-performance storage

DAOS offers multiple I/O interfaces to users:

- Filesystem emulation API allowing legacy POSIX file access to DAOS storage
- Native object-based APIs (e.g.., key-val, array) offering more powerful semantics compared to POSIX-like file APIs
  - Data locality, replication strategy, etc.




Various DAOS access methods.

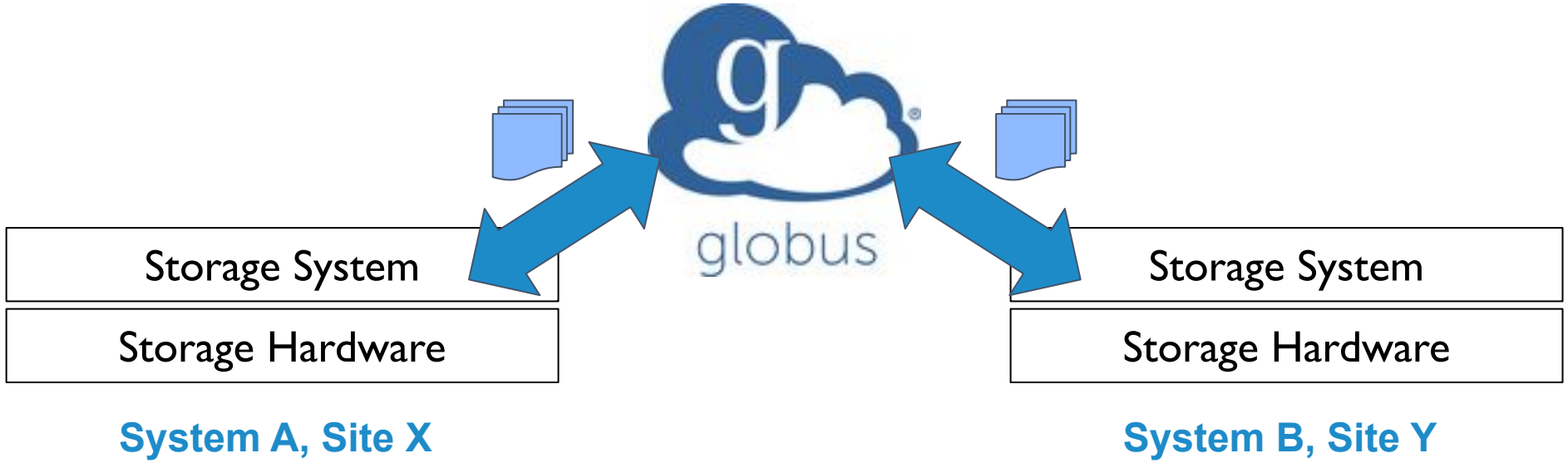
Figure courtesy of Intel

# Sharing data with collaborators: Globus

**Globus** is a platform for managing research data, enabling the moving, sharing, and archiving of large volumes of data among distributed sites.

- Manages data transfers between endpoints
- Monitors performance and errors
- Retries and corrects errors, where possible
- Reports status back to users

Globus can be easily accessed either using CLI tools or a web-interface.

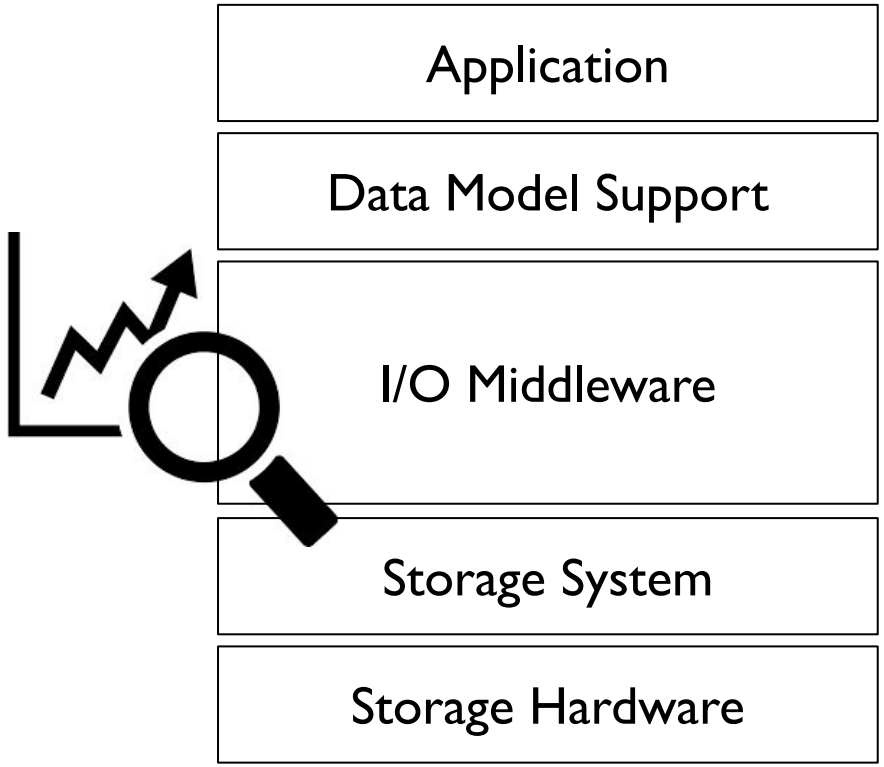


# Tools: understanding and improving I/O behavior

Application performance monitoring and analysis tools are critical to *better understanding I/O behavior* and *informing potential tuning decisions*, answering questions like:

- How much of your run time is spent reading and writing files?
- Does it get better, worse, or is it the same as you scale up?
- Does it get better, worse, or is it the same across platforms?
- How should you prioritize I/O tuning to get the most bang for your buck?

As a starting point to better answering these sorts of questions, I recommend using a popular tool called **Darshan**[1].




[1] https://www.mcs.anl.gov/research/projects/darshan/

# Tools: Darshan application I/O characterization

Darshan is a lightweight I/O characterization tool that is commonly deployed at HPC facilities to provide important details on I/O behavior of user jobs.

It has 2 primary components:

1. **Darshan runtime library**
   - Transparently and scalably instrument multiple layers of the application I/O stack by intercepting I/O calls and recording file access statistics
   - At application shutdown, generate a compressed log containing detailed I/O statistics for each file accessed by the application

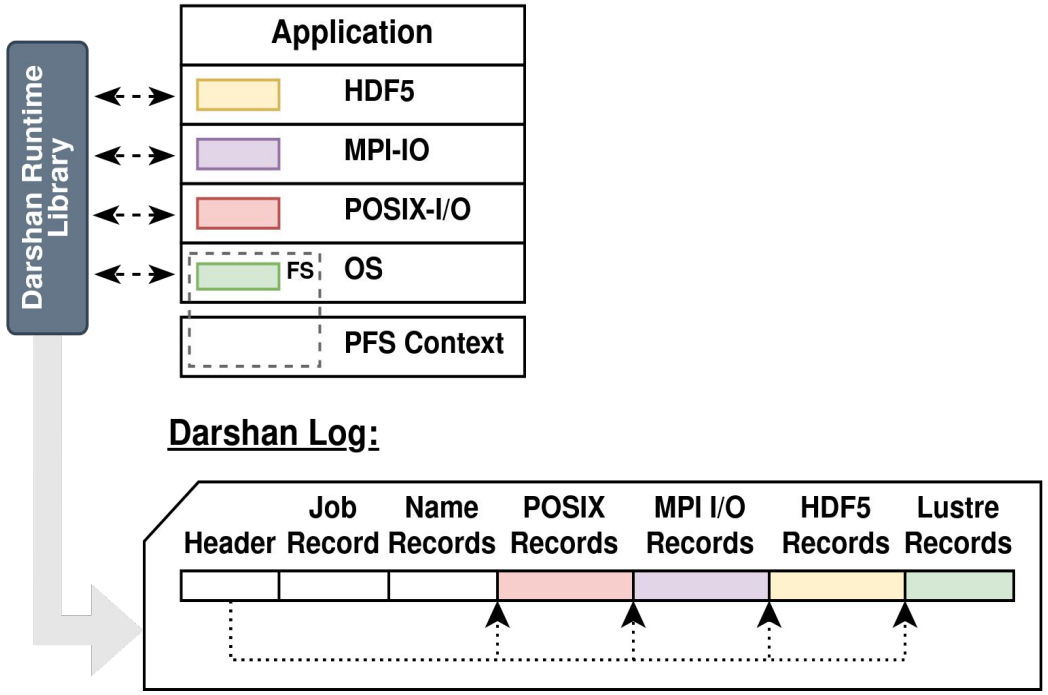



Figure courtesy Jakob Luettgau (Inria)

# Tools: Darshan application I/O characterization

Darshan is a lightweight I/O characterization tool that is commonly deployed at HPC facilities to provide important details on I/O behavior of user jobs.

It has 2 primary components:

2. **Darshan log analysis tools**
   - Tools for inspecting and presenting key information about I/O behavior
   - Recent development effort on PyDarshan, a framework for extracting and presenting key Darshan log data using popular Python packages (pandas, matplotlib, etc.)

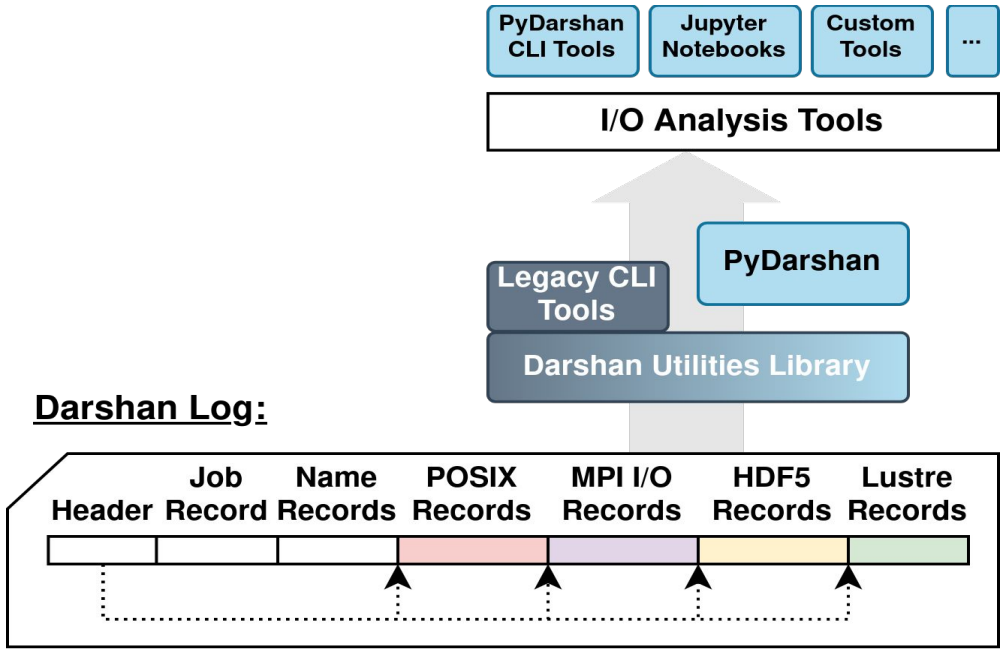
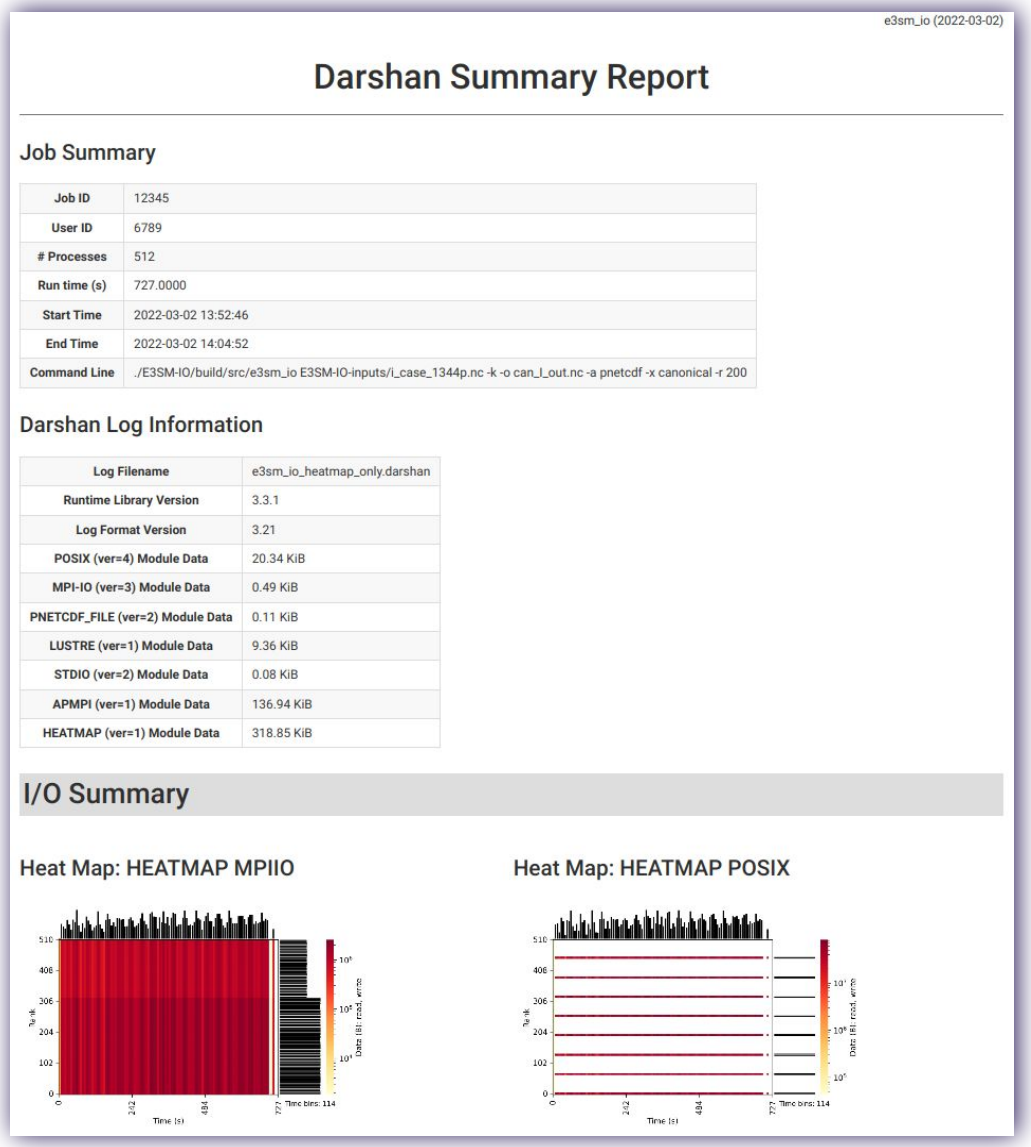



Figure courtesy Jakob Luettgau (Inria)

# Tools: Darshan job summary

**A helpful starting point for Darshan log file analysis is the PyDarshan job summary tool.**

**It generates an HTML report providing key details on I/O performance and access characteristics that can be used to better understand the application's I/O behavior.**



e3sm_io (2022-03-02)

## Darshan Summary Report

### Job Summary

| | |
|---|---|
| Job ID | 12345 |
| User ID | 6789 |
| # Processes | 512 |
| Run time (s) | 727.0000 |
| Start Time | 2022-03-02 13:52:46 |
| End Time | 2022-03-02 14:04:52 |
| Command Line | ./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200 |

### Darshan Log Information

| | |
|---|---|
| Log Filename | e3sm_io_heatmap_only.darshan |
| Runtime Library Version | 3.3.1 |
| Log Format Version | 3.21 |
| POSIX (ver=4) Module Data | 20.34 KiB |
| MPI-IO (ver=3) Module Data | 0.49 KiB |
| PNETCDF_FILE (ver=2) Module Data | 0.11 KiB |
| LUSTRE (ver=1) Module Data | 9.36 KiB |
| STDIO (ver=2) Module Data | 0.08 KiB |
| APMPI (ver=1) Module Data | 136.94 KiB |
| HEATMAP (ver=1) Module Data | 318.85 KiB |

### I/O Summary

Heat Map: HEATMAP MPIIO

Heat Map: HEATMAP POSIX

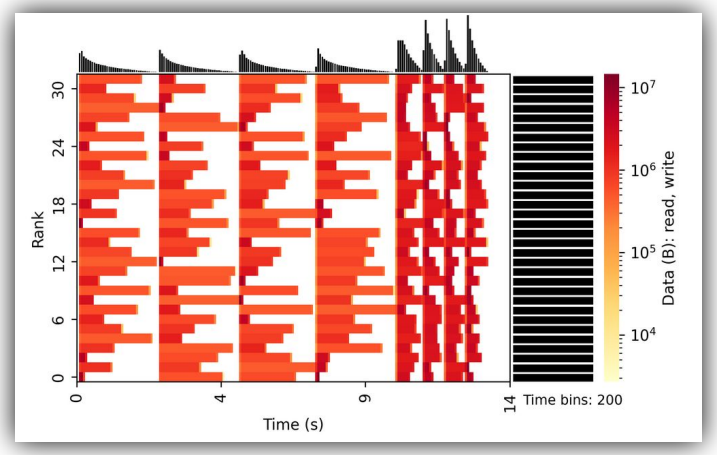
# Tools: Darshan job summary

**Detailed job metadata**

## Job Summary

| | |
|---|---|
| **Job ID** | 12345 |
| **User ID** | 6789 |
| **# Processes** | 512 |
| **Run time (s)** | 727.0000 |
| **Start Time** | 2022-03-02 13:52:46 |
| **End Time** | 2022-03-02 14:04:52 |
| **Command Line** | ./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_134 |

**Heatmaps of I/O activity**




## Per-Module Statistics: POSIX

### Overview

| | |
|---|---|
| **files accessed** | 3 |
| **bytes read** | 24.53 MiB |
| **bytes written** | 283.74 GiB |
| **I/O performance estimate** | 1023.84 MiB/s (average) |

### File Count Summary
### (estimated by POSIX I/O access offsets)

| | number of files | avg. size | max size |
|---|---|---|---|
| **total files** | 3 | 99.74 GiB | 297.71 GiB |
| **read-only files** | 1 | 11.18 MiB | 11.18 MiB |
| **write-only files** | 2 | 149.60 GiB | 297.71 GiB |
| **read/write files** | 0 | 0 | 0 |

**Comprehensive I/O statistics for multiple layers of the I/O stack**

# HPC I/O best practices

Achieving the best I/O performance on HPC systems can be a challenging task:

- Deep storage software stack must effectively transform diverse application access patterns into performant low-level storage access

Here are some general best practices that should position you to get the best I/O performance:

- **Use high-level I/O libraries (e.g., HDF5) for managing data, rather than low-level POSIX APIs.**
  - Expressive interfaces and portable file formats more suitable for managing scientific data
  - Transparently implement general and platform-specific optimizations to I/O workloads

# HPC I/O best practices

Achieving the best I/O performance on HPC systems can be a challenging task:

- Deep storage software stack must effectively transform diverse application access patterns into performant low-level storage access

Here are some general best practices that should position you to get the best I/O performance:

- **Use high-level I/O libraries (e.g., HDF5) for managing data, rather than low-level POSIX APIs.**
  - Expressive interfaces and portable file formats more suitable for managing scientific data
  - Transparently implement general and platform-specific optimizations to I/O workloads
- **Use larger access sizes and limit the overall number of operations issued to storage systems.**
  - HPC storage resources are typically remote and have high latency costs that can stall applications

# HPC I/O best practices

Achieving the best I/O performance on HPC systems can be a challenging task:

- Deep storage software stack must effectively transform diverse application access patterns into performant low-level storage access

Here are some general best practices that should position you to get the best I/O performance:

- **Use high-level I/O libraries (e.g., HDF5) for managing data, rather than low-level POSIX APIs.**
  - Expressive interfaces and portable file formats more suitable for managing scientific data
  - Transparently implement general and platform-specific optimizations to I/O workloads
- **Use larger access sizes and limit the overall number of operations issued to storage systems.**
  - HPC storage resources are typically remote and have high latency costs that can stall applications
- **Be aware of intended usage of storage resources, in terms of performance, data lifetime, etc.**
  - Some file systems are intended for high-performance access (e.g., scratch or project file systems) while others are not (e.g., home file systems)
  - Higher performance tiers of a storage system may purge old data periodically to effectively manage capacity

# HPC I/O best practices

Achieving the best I/O performance on HPC systems can be a challenging task:

- Deep storage software stack must effectively transform diverse application access patterns into performant low-level storage access

Here are some general best practices that should position you to get the best I/O performance:

- **Use high-level I/O libraries (e.g., HDF5) for managing data, rather than low-level POSIX APIs.**
  - Expressive interfaces and portable file formats more suitable for managing scientific data
  - Transparently implement general and platform-specific optimizations to I/O workloads
- **Use larger access sizes and limit the overall number of operations issued to storage systems.**
  - HPC storage resources are typically remote and have high latency costs that can stall applications
- **Be aware of intended usage of storage resources, in terms of performance, data lifetime, etc.**
  - Some file systems are intended for high-performance access (e.g., scratch or project file systems) while others are not (e.g., home file systems)
  - Higher performance tiers of a storage system may purge old data periodically to effectively manage capacity
- **Use monitoring/analysis tools like Darshan to better understand I/O access characteristics and attained performance for your application.**

# HPC I/O best practices

Achieving the best I/O performance on HPC systems can be a challenging task:

- Deep storage software stack must effectively transform diverse application access patterns into performant low-level storage access

Here are some general best practices that should position you to get the best I/O performance:

- **Use high-level I/O libraries (e.g., HDF5) for managing data, rather than low-level POSIX APIs.**
  - Expressive interfaces and portable file formats more suitable for managing scientific data
  - Transparently implement general and platform-specific optimizations to I/O workloads
- **Use larger access sizes and limit the overall number of operations issued to storage systems.**
  - HPC storage resources are typically remote and have high latency costs that can stall applications
- **Be aware of intended usage of storage resources, in terms of performance, data lifetime, etc.**
  - Some file systems are intended for high-performance access (e.g., scratch or project file systems) while others are not (e.g., home file systems)
  - Higher performance tiers of a storage system may purge old data periodically to effectively manage capacity
- **Use monitoring/analysis tools like Darshan to better understand I/O access characteristics and attained performance for your application.**
- **Always check facility documentation for site-specific best practices and don't be afraid to ask for help/clarification using support channels.**

# HPC I/O performance: what's possible?

The IO500 foundation manages an I/O benchmark suite that is used to test and rank the attained I/O performance for HPC storage systems under different access patterns:

| # | INFORMATION | | | | | | IO500 | |
|---|---|---|---|---|---|---|---|---|
| | SYSTEM | INSTITUTION | STORAGE VENDOR | FILESYSTEM TYPE | CLIENT NODES | CLIENT TOTAL PROCS | BW | MD |
| 1 | Aurora | Argonne National Laboratory | Intel | DAOS | 300 | 62400 | 10,066.09 | 102,785.41 |
| 2 | SuperMUC-NG-Phase2-EC | LRZ | Lenovo | DAOS | 90 | 6480 | 742.90 | 8,472.60 |
| 3 | Shaheen III | King Abdullah University of Science and Technology | HPE | Lustre | 2080 | 16640 | 709.52 | 895.35 |
| 4 | Leonardo | EuroHPC-CINECA | DDN | EXAScaler | 2000 | 16000 | 807.12 | 521.79 |
| 5 | Lise | Zuse Institute Berlin | Megware | DAOS | 10 | 960 | 65.01 | 1,620.13 |

Recent IO500 results from ISC'24 conference showing the current rankings of fastest production HPC storage systems.

https://io500.org/list/isc24/production

# HPC I/O performance: what's possible?

The IO500 foundation manages an I/O benchmark suite that is used to test and rank the attained I/O performance for HPC storage systems under different access patterns:

| # | INFORMATION | | | | | | IO500 | |
|---|---|---|---|---|---|---|---|---|
| | SYSTEM | INSTITUTION | STORAGE VENDOR | FILESYSTEM TYPE | CLIENT NODES | CLIENT TOTAL PROCS | BW | MD |
| 1 | Aurora | Argonne National Laboratory | Intel | DAOS | 300 | 62400 | 10,066.09 | 102,785.41 |
| 2 | SuperMUC-NG-Phase2-EC | LRZ | Lenovo | DAOS | 90 | 6480 | 742.90 | 8,472.60 |
| 3 | Shaheen III | King Abdullah University of Science and Technology | HPE | Lustre | 2080 | 16640 | 709.52 | 895.35 |
| 4 | Leonardo | EuroHPC-CINECA | DDN | EXAScaler | 2000 | 16000 | 807.12 | 521.79 |
| 5 | Lise | Zuse Institute Berlin | Megware | DAOS | 10 | 960 | 65.01 | 1,620.13 |

**ALCF Aurora current #1!**

# HPC I/O performance: what's possible?

The IO500 foundation manages an I/O benchmark suite that is used to test and rank the attained I/O performance for HPC storage systems under different access patterns:

| # | INFORMATION | | | | | | IO500 | |
|---|---|---|---|---|---|---|---|---|
| | SYSTEM | INSTITUTION | STORAGE VENDOR | FILESYSTEM TYPE | CLIENT NODES | CLIENT TOTAL PROCS | BW | MD |
| 1 | Aurora | Argonne National Laboratory | Intel | DAOS | 300 | 62400 | 10,066.09 | 102,785.41 |
| 2 | SuperMUC-NG-Phase2-EC | LRZ | Lenovo | DAOS | 90 | 6480 | 742.90 | 8,472.60 |
| 3 | Shaheen III | King Abdullah University of Science and Technology | HPE | Lustre | 2080 | 16640 | 709.52 | 895.35 |
| 4 | Leonardo | EuroHPC-CINECA | DDN | EXAScaler | 2000 | 16000 | 807.12 | 521.79 |
| 5 | Lise | Zuse Institute Berlin | Megware | DAOS | 10 | 960 | 65.01 | 1,620.13 |

**Both DAOS and Lustre are well represented in the current rankings**

# HPC I/O performance: what's possible?

The IO500 foundation manages an I/O benchmark suite that is used to test and rank the attained I/O performance for HPC storage systems under different access patterns:

| # | INFORMATION | | | | | | IO500 | |
|---|---|---|---|---|---|---|---|---|
| | SYSTEM | INSTITUTION | STORAGE VENDOR | FILESYSTEM TYPE | CLIENT NODES | CLIENT TOTAL PROCS | BW | MD |
| 1 | Aurora | Argonne National Laboratory | Intel | DAOS | 300 | 62400 | 10,066.09 | 102,785.41 |
| 2 | SuperMUC-NG-Phase2-EC | LRZ | Lenovo | DAOS | 90 | 6480 | 742.90 | 8,472.60 |
| 3 | Shaheen III | King Abdullah University of Science and Technology | HPE | Lustre | 2080 | 16640 | 709.52 | 895.35 |
| 4 | Leonardo | EuroHPC-CINECA | DDN | EXAScaler | 2000 | 16000 | 807.12 | 521.79 |
| 5 | Lise | Zuse Institute Berlin | Megware | DAOS | 10 | 960 | 65.01 | 1,620.13 |

**Aurora attained performance of nearly 10 TiB/sec bandwidth and over 100 million I/O ops/sec**

# HPC I/O performance: what's possible?

The IO500 foundation manages an I/O benchmark suite that is used to test and rank the attained I/O performance for HPC storage systems under different access patterns:

| # | INFORMATION | | | | | | IO500 | |
|---|---|---|---|---|---|---|---|---|
| | SYSTEM | INSTITUTION | STORAGE VENDOR | FILESYSTEM TYPE | CLIENT NODES | CLIENT TOTAL PROCS | BW | MD |
| 1 | Aurora | Argonne National Laboratory | Intel | DAOS | 300 | 62400 | 10,066.09 | 102,785.41 |
| 2 | SuperMUC-NG-Phase2-EC | LRZ | Lenovo | DAOS | 90 | 6480 | 742.90 | 8,472.60 |
| 3 | Shaheen III | King Abdullah University of Science and Technology | HPE | Lustre | 2080 | 16640 | 709.52 | 895.35 |
| 4 | Leonardo | EuroHPC-CINECA | DDN | EXAScaler | 2000 | 16000 | 807.12 | 521.79 |
| 5 | Lise | Zuse Institute Berlin | Megware | DAOS | 10 | 960 | 65.01 | 1,620.13 |

**Achieving these massive performance numbers requires a large number of clients/nodes**

**Aurora attained performance of nearly 10 TiB/sec bandwidth and over 100 million I/O ops/sec**

# A recap

Today we have covered various technologies that comprise the HPC I/O stack, which is used to persist, manage, and share large-scale scientific datasets, a critical aspect of high-performance computing.

- *Storage hardware* with different performance characteristics aggregated into massive systems that store raw data.
- *Parallel file systems* offer high-performance, scalable file storage over provided storage hardware
- *I/O libraries* provide interfaces for managing data at different abstraction layers
  - *POSIX* provides a portable, performant low-level file system interface
  - *MPI-IO* introduces capabilities for parallel access of files
  - *HDF5* provides a data management interface more closely aligned with application data abstractions
- *File transfer/sharing* mechanisms to enable collaboration
- *Tools* are available for better understanding and, ideally, improving I/O performance

Additionally, we discussed some common best practices for achieving good I/O performance and provided some general performance expectations for production HPC systems.

# Thank you!