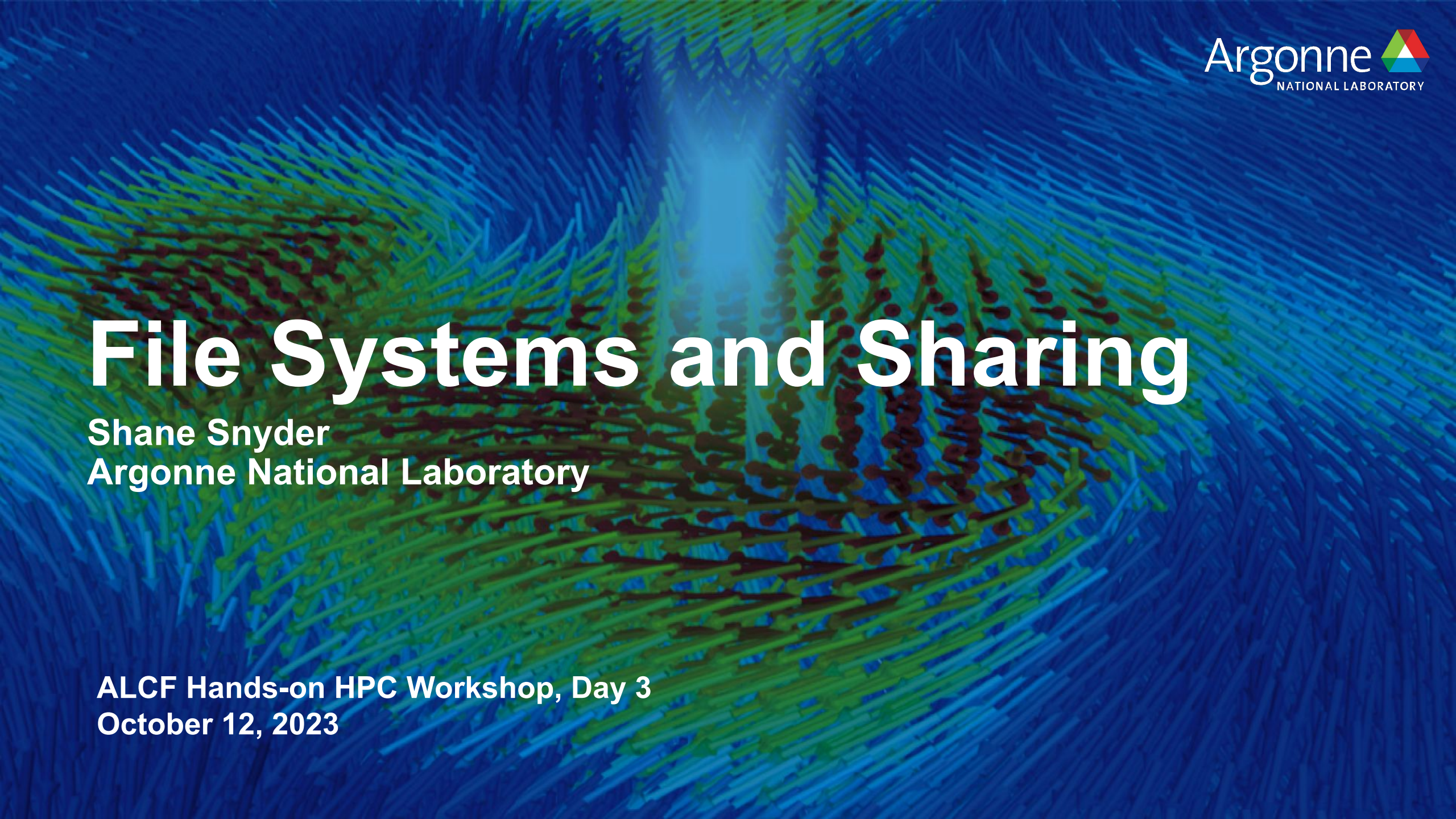


File Systems and Sharing

Shane Snyder
Argonne National Laboratory

ALCF Hands-on HPC Workshop, Day 3
October 12, 2023

Managing scientific data

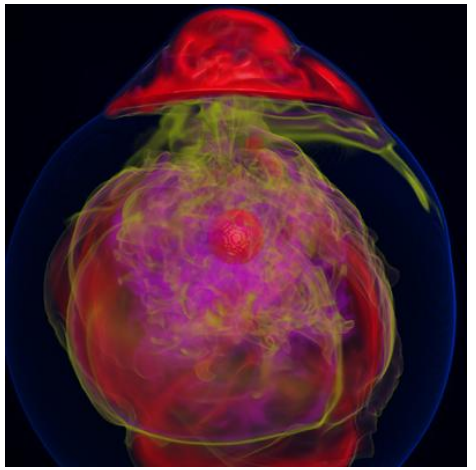
HPC applications spanning various scientific disciplines have a range of diverse data management needs

- 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

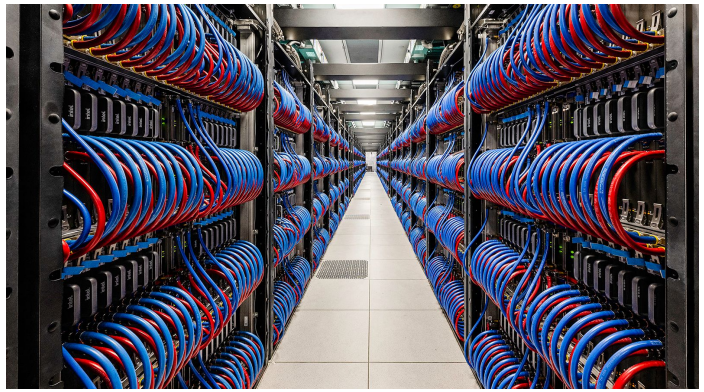
Meanwhile, hardware trends have enabled novel, high-performance storage system designs that promise increased productivity to HPC apps

ALCF and other HPC facilities deploy vast amounts of storage resources to help meet the I/O needs of HPC applications

- Today, we'll introduce the basics of the HPC data managements stack at ALCF and try to establish some best practices for using it effectively



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

Parallel file systems

Parallel file systems (PFSes) have long been offered by HPC facilities as a general-purpose tool for persisting users' data

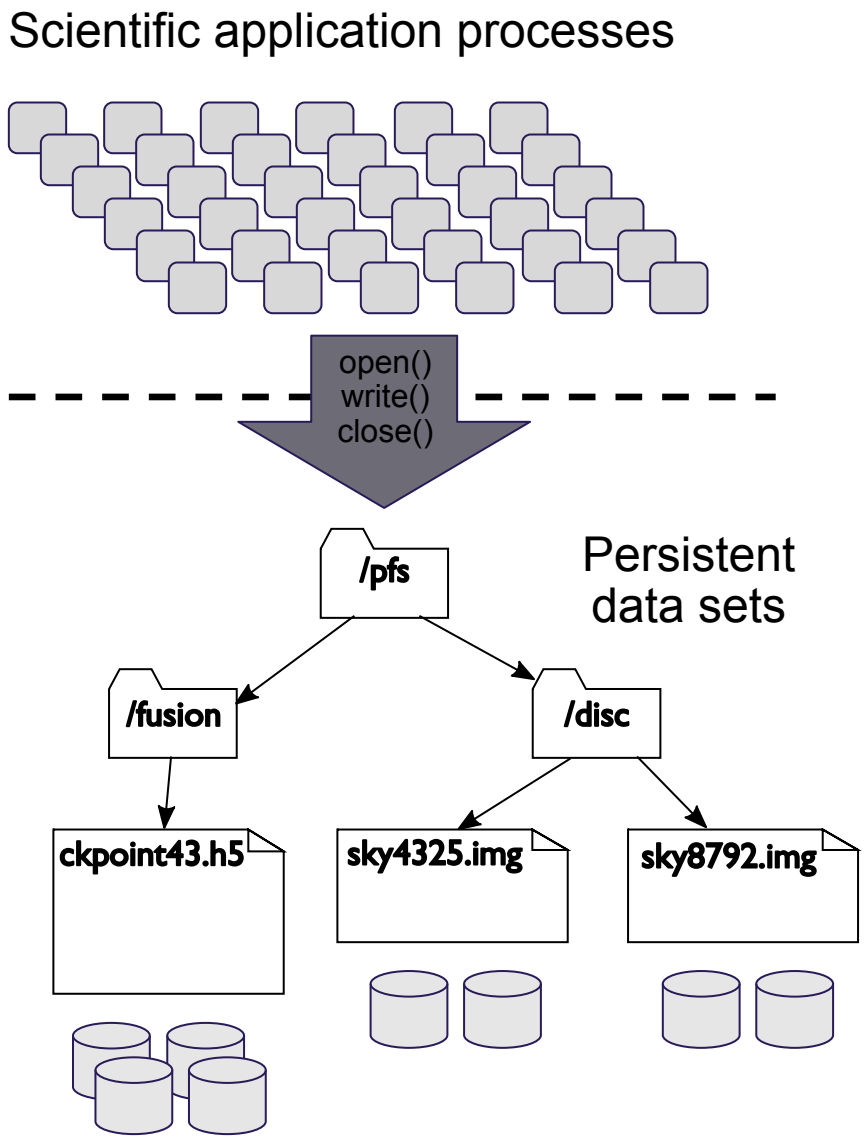
- Users store data in a familiar file/directory hierarchy, but with much more aggregate capacity and performance relative to a local FS

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

Popular PFSes available on modern HPC systems include:

- Lustre
- GPFS (a.k.a Spectrum Scale)
- BeeGFS



Parallel file systems: Lustre

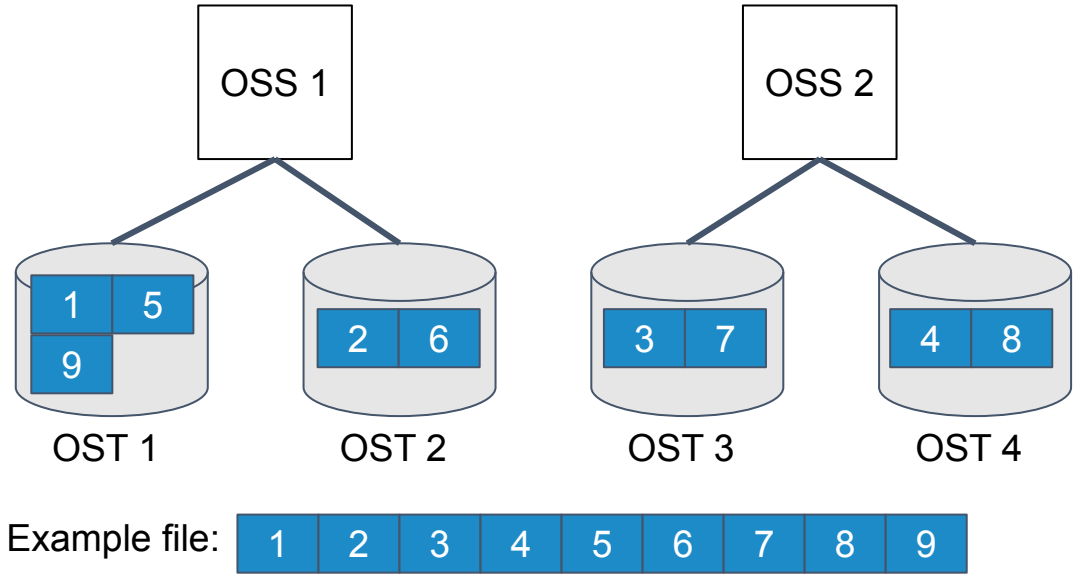
Lustre is currently the preferred scratch file system in production at the ALCF (as well as at many other large-scale HPC facilities)

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

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

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

Achieving the best performance with Lustre sometimes requires thoughtful file striping settings

- Larger files tend to be benefit from larger stripe counts (i.e., more storage resources)

Parallel file systems: Lustre

Achieving the best performance with Lustre sometimes requires thoughtful file striping settings

- Larger files tend to be benefit from larger stripe counts (i.e., more storage resources)

Default Lustre stripe settings may not be optimal! On ALCF systems, files default to a stripe width of 1, meaning they are stored on a single OST. The onus is on users to ensure stripe settings are set appropriately.

Stripe settings can be modified using the `lfs` tool:

```
lfs setstripe -S <size> -c <count> <file/dir name>
```

Parallel file systems: Lustre

Achieving the best performance with Lustre sometimes requires thoughtful file striping settings

- Larger files tend to be benefit from larger stripe counts (i.e., more storage resources)

```
$ mkdir stripe4
$ lfs getstripe stripe4/
stripe4/
stripe_count:  1 stripe_size:   1048576 pattern:       0 stripe_offset: -1
```

By default (on this file system), new files/directories are set to use a stripe count of 1.

NOTE: Stripe settings applied to a directory are inherited by all files within it.

Parallel file systems: Lustre

Achieving the best performance with Lustre sometimes requires thoughtful file striping settings

- Larger files tend to be benefit from larger stripe counts (i.e., more storage resources)

```
$ mkdir stripe4
$ lfs getstripe stripe4/
stripe4/
stripe_count:  1 stripe_size:   1048576 pattern:       0 stripe_offset: -1

$ lfs setstripe -c 4 stripe4/
$ lfs getstripe stripe4/
stripe4/
stripe_count:  4 stripe_size:   1048576 pattern:       raid0 stripe_offset: -1
```

Using the `setstripe` command we can override the default striping and request more storage resources.

Parallel file systems: Lustre

Achieving the best performance with Lustre sometimes requires thoughtful file striping settings

- Larger files tend to be benefit from larger stripe counts (i.e., more storage resources)

Recent Lustre versions have introduced a new feature called progressive file layouts (PFL) that enables a more flexible file striping strategy

- PFL replaces traditional static striping with a dynamic approach that increases the stripe size as the file offset increases
- Small files stored on a single OST, large files can grow to stripe across many OSTs

PFL not enabled by default on ALCF Lustre volumes yet, but something to keep in mind.

PFL can provide default stripe settings that achieve reasonable performance for a variety of I/O workloads, but knowledgeable users can still achieve the best performance by thoughtful tuning of striping.

ALCF Polaris file systems

ALCF offers a number of file systems to Polaris users:

- Grand (Lustre)
  - Temporary storage of I/O intensive data
  - 100 PB aggregate capacity, 650 GB/sec transfer rate
  - 160 OSTs, 40 MDTs
- Eagle (Lustre)
  - Temporary storage of I/O intensive data
  - Community sharing with Globus (more later)
  - 100 PB aggregate capacity, 650 GB/sec transfer rate
  - 160 OSTs, 40 MDTs
- Home (Lustre)
  - General-purpose storage of data that is not I/O-intensive (e.g., binaries, source code, etc.)
  - Regular backups to tape
- Node local storage (XFS)
  - Temporary, compute node-local storage for jobs
  - 2 SSDs with total capacity of 3.2 TB
  - Users must copy data somewhere persistent at job end

ALCF Polaris file systems

ALCF offers a number of file systems to Polaris users:

- Grand (Lustre)
  - Temporary storage of I/O intensive data
  - 100 PB aggregate capacity, 650 GB/sec transfer rate
  - 160 OSTs, 40 MDTs
- Eagle (Lustre)
  - Temporary storage of I/O intensive data
  - Community sharing with Globus (more later)
  - 100 PB aggregate capacity, 650 GB/sec transfer rate
  - 160 OSTs, 40 MDTs
- Home (Lustre)
  - General-purpose storage of data that is not I/O-intensive (e.g., binaries, source code, etc.)
  - Regular backups to tape
- Node local storage (XFS)
  - Temporary, compute node-local storage for jobs
  - 2 SSDs with total capacity of 3.2 TB
  - Users must copy data somewhere persistent at job end

Users should always carefully consider whether their usage of production storage resources matches their intended use.

- ➢ Maximize app performance
- ➢ Maximize system efficiency
- ➢ Ensure data integrity

I/O libraries: interacting with file systems

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

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

I/O libraries: interacting with file systems

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

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- 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.

I/O libraries: interacting with file systems

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

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- 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 the large-scale parallel file access

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

I/O libraries: interacting with file systems

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

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- 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 the large-scale 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.

I/O libraries: parallel I/O capabilities

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

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

MPI is actually 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 offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

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

I/O libraries: parallel I/O capabilities

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

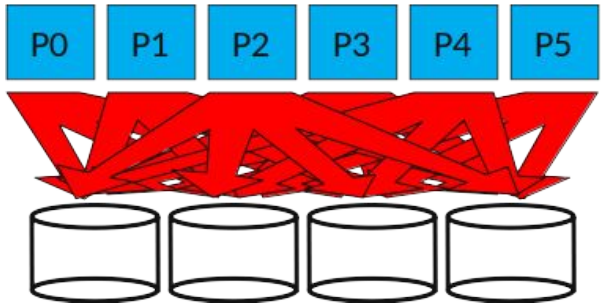
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually 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 offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

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



I/O libraries: parallel I/O capabilities

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

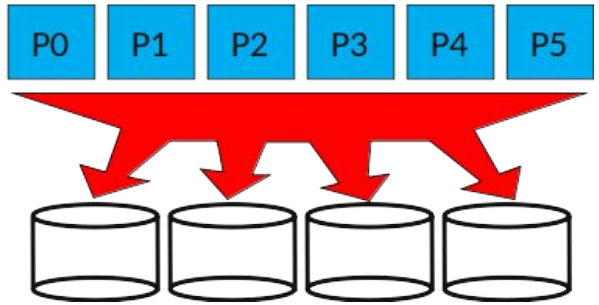
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually 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 offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

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



I/O libraries: parallel I/O capabilities

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

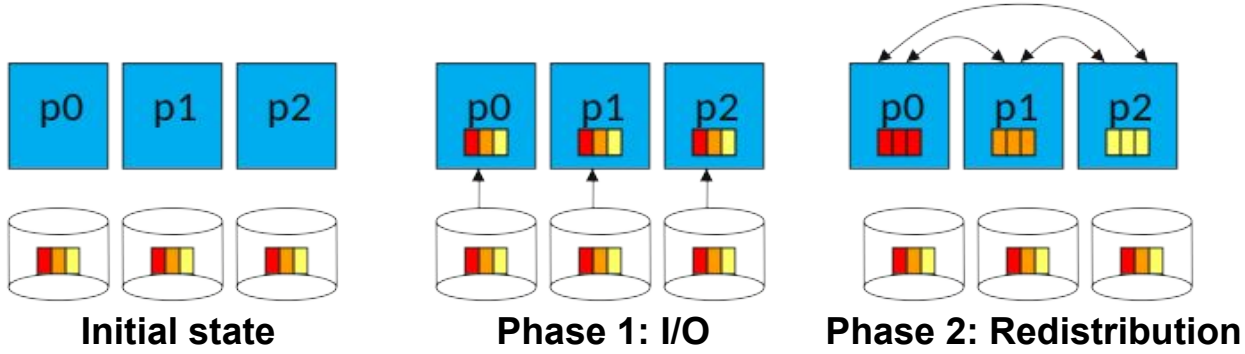
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually 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 offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

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



Two-phase collective I/O algorithm

I/O libraries: parallel I/O capabilities

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

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

MPI is actually 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 offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

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

More on MPI-IO this afternoon in Breakout Session 1 (Rob Latham, ANL)

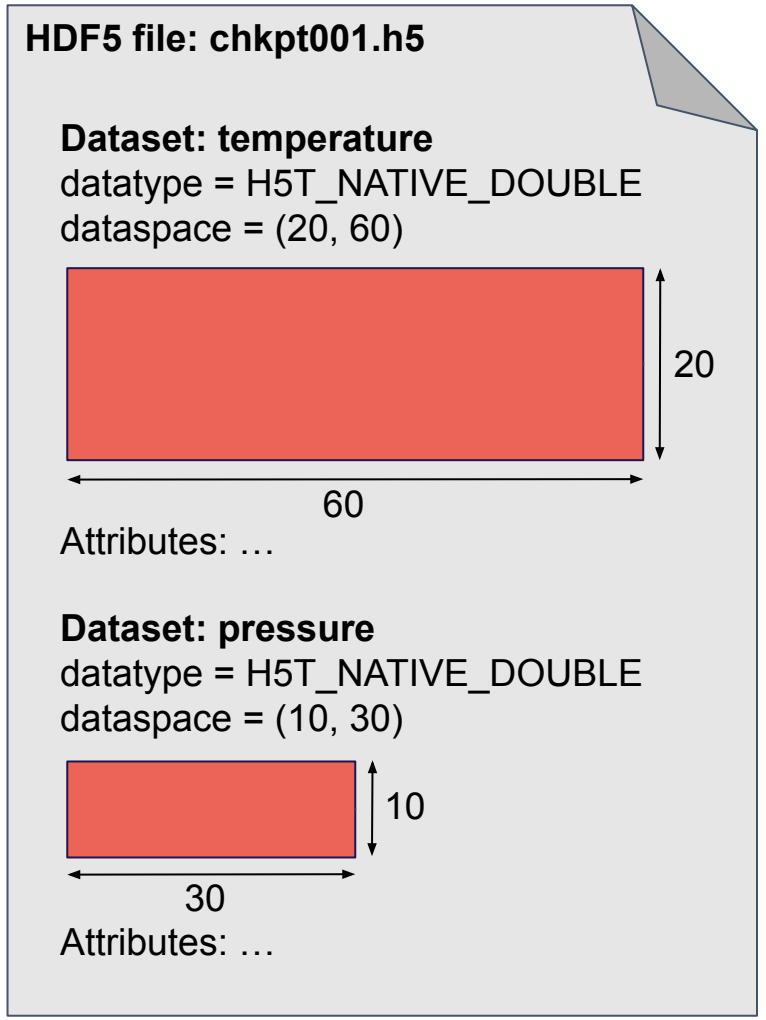
I/O libraries: scientific data management abstractions

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 amounts of scientific data

- Enables 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 log-structured storage, using an object store rather than file system



I/O libraries: scientific data management abstractions

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

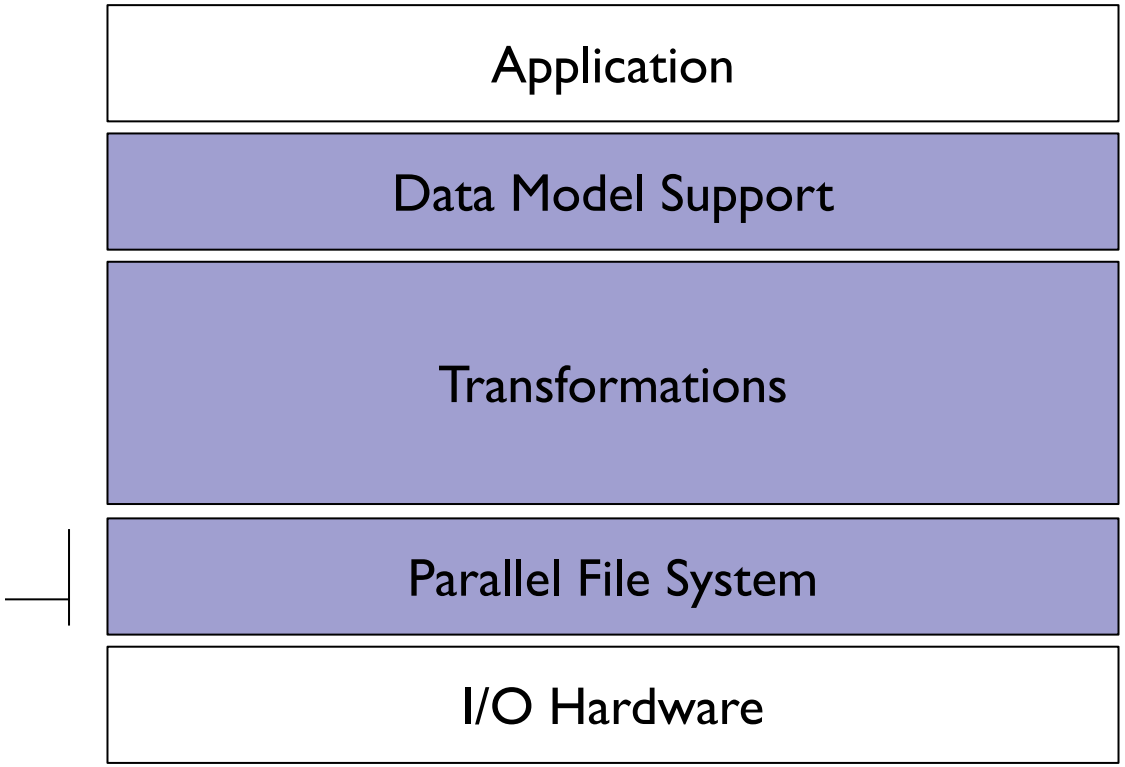
- Storing independent data products in unique files or manually serializing collections of data products into a single file is often untenable

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

- Enables 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 log-structured storage, using an object store rather than file system

More on HDF5 this afternoon in Breakout Session 1 (Rob Latham, ANL)

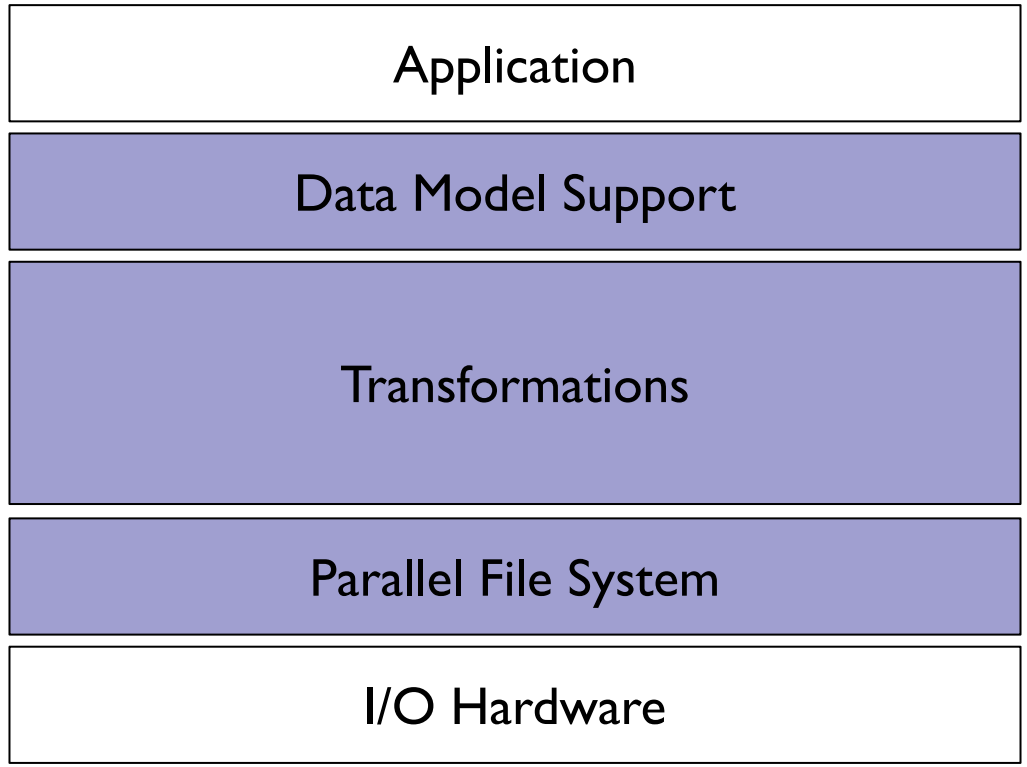
Putting it all together: the HPC I/O stack



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

Lustre, GPFS

Putting it all together: 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

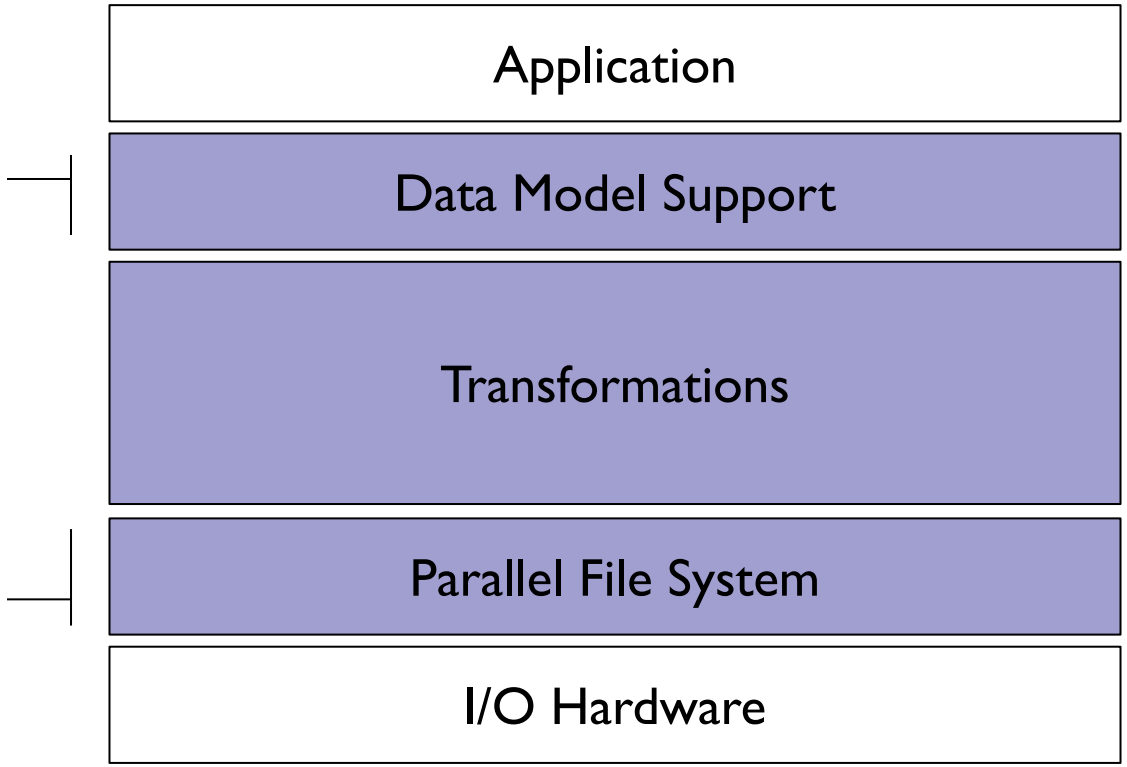
I/O Middleware organizes and transforms accesses from many processes, especially those using collective I/O.

MPI-IO

Putting it all together: the HPC I/O stack

Data Model Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS



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

MPI-IO

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

Lustre, GPFS

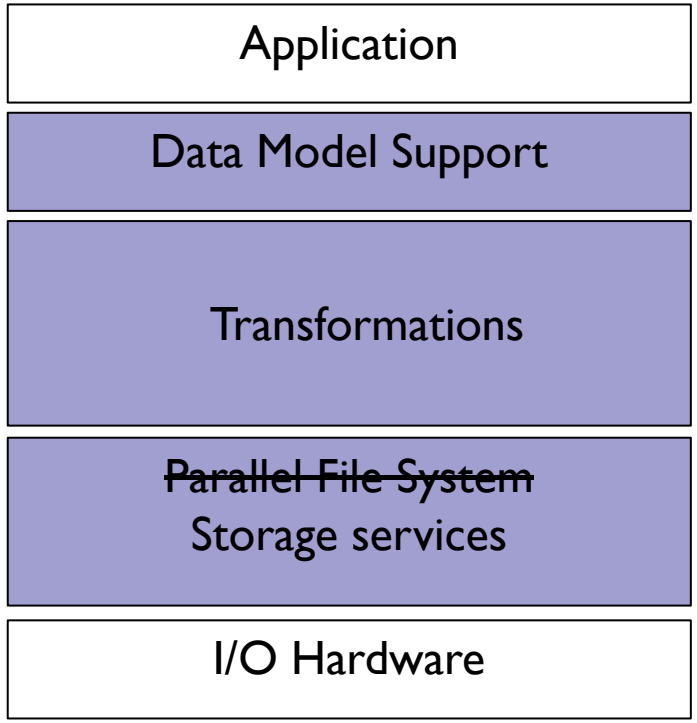
Emerging storage technologies: DAOS

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

ALCF Aurora will feature Intel's **DAOS** storage system, a first-of-a-kind object-based storage system for large-scale HPC platforms

- Leverages both SCM and SSDs for storage

DAOS provides a range of different interfaces to users



Emerging storage technologies: DAOS

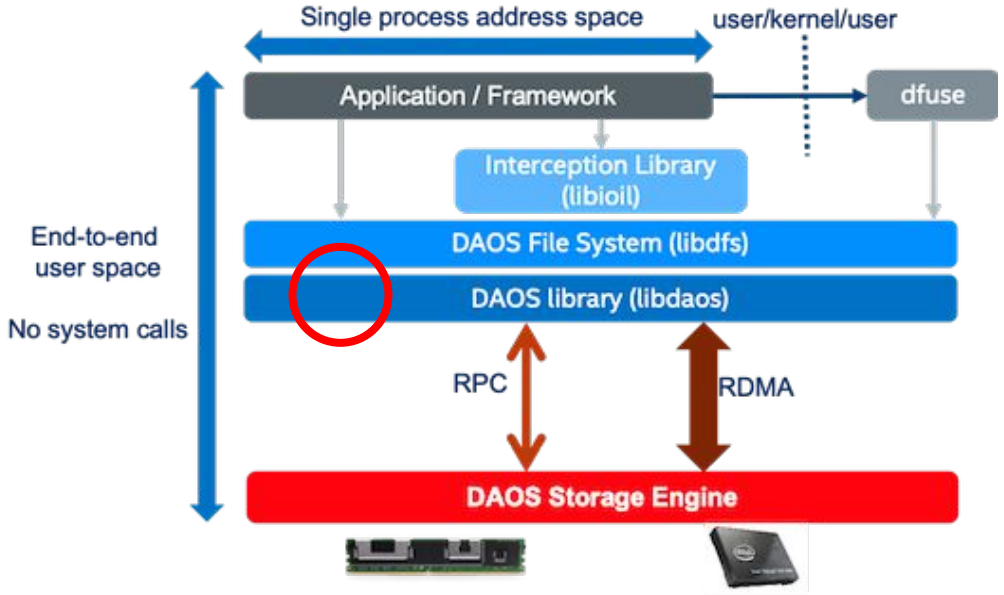
HPC storage technology is changing to meet the needs of diverse application workloads and to embrace emerging storage trends

ALCF Aurora will feature Intel's **DAOS** storage system, a first-of-a-kind object-based storage system for large-scale HPC platforms

- Leverages both SCM and SSDs for storage

DAOS provides a range of different interfaces to users

- **Direct usage of native DAOS object (libdaos) interface**
  - Native library for directly accessing objects (arrays or key-val stores) rather than files



Various DAOS access methods.

Figure courtesy of Intel

Emerging storage technologies: DAOS

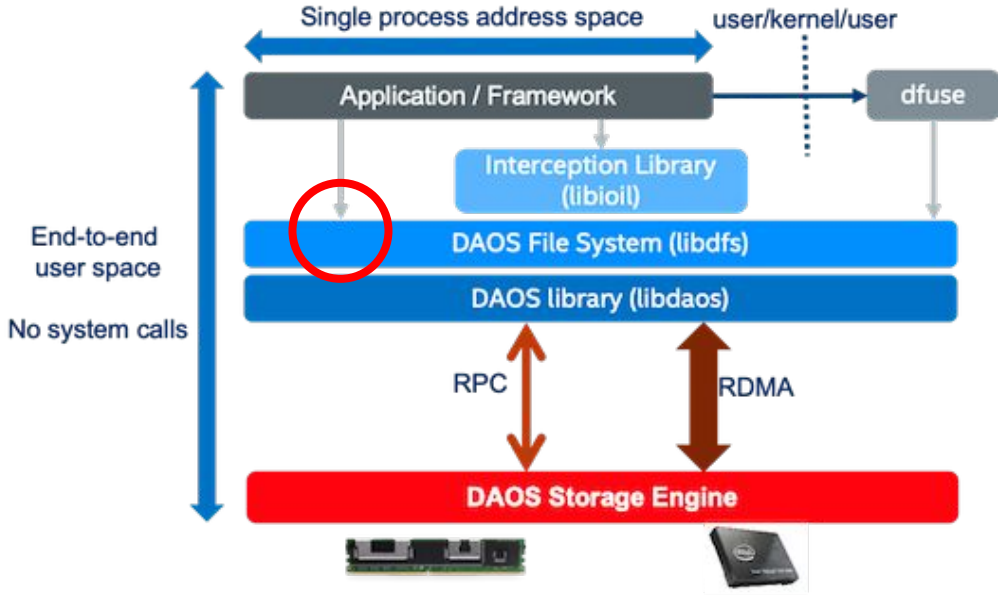
HPC storage technology is changing to meet the needs of diverse application workloads and to embrace emerging storage trends

ALCF Aurora will feature Intel's **DAOS** storage system, a first-of-a-kind object-based storage system for large-scale HPC platforms

- Leverages both SCM and SSDs for storage

DAOS provides a range of different interfaces to users

- **Direct usage of POSIX-like DAOS file system (libdfs) interface**
  - POSIX-like file and directory abstractions implemented over the native DAOS object interface



Various DAOS access methods.

Figure courtesy of Intel

Emerging storage technologies: DAOS

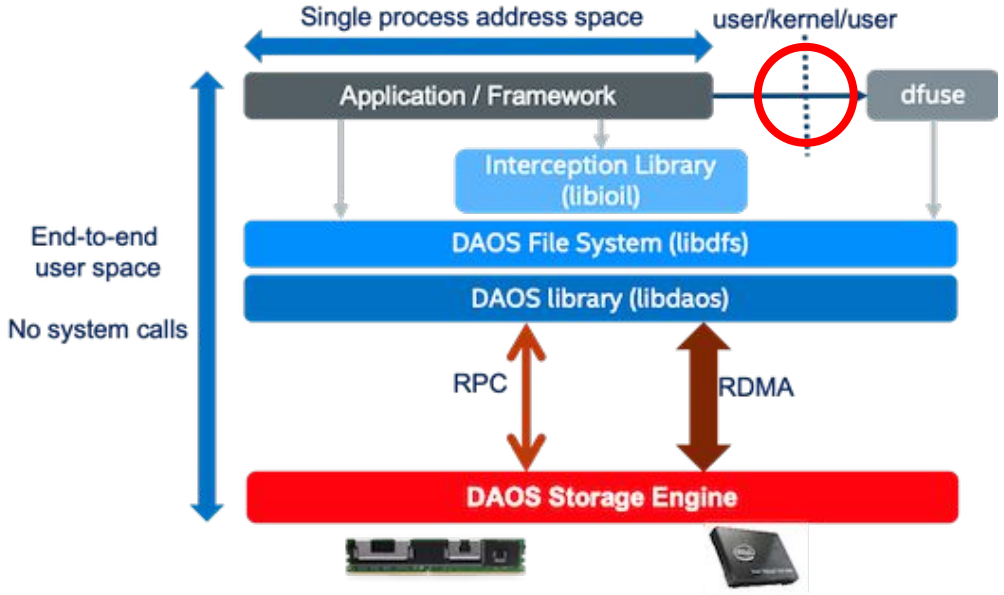
HPC storage technology is changing to meet the needs of diverse application workloads and to embrace emerging storage trends

ALCF Aurora will feature Intel's **DAOS** storage system, a first-of-a-kind object-based storage system for large-scale HPC platforms

- Leverages both SCM and SSDs for storage

DAOS provides a range of different interfaces to users

- **Legacy POSIX support using FUSE**
  - For legacy apps already using POSIX access
  - No app modifications needed



Various DAOS access methods.

Figure courtesy of Intel

Tools: analyzing application I/O behavior

Application-level analysis tools are critical to *better understanding I/O behavior* and *informing potential tuning decisions*

Darshan is a lightweight I/O characterization tool commonly deployed at HPC facilities, including ALCF systems

- Transparent, low-overhead instrumentation of multiple layers of the HPC I/O stack
- Detailed counters/timers/statistics for each file accessed by the app stored in a condensed log
- Analysis tools for inspecting and presenting key information about I/O behavior (e.g., the Darshan job summary tool, **right**)

e3sm_io (2022-03-02)

Darshan Summary Report

Job Summary

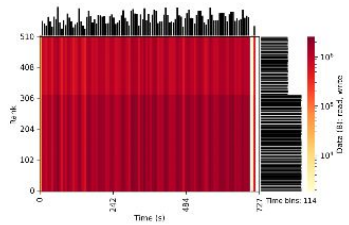
Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200

Darshan Log Information

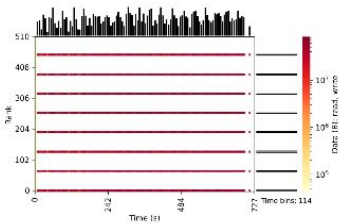
Log Filename	e3sm_io_heatmap_only.darshan
Runtime Library Version	3.3.1
Log Format Version	3.21
POSIX (ver=4) Module Data	20.34 KiB
MPI-IO (ver=3) Module Data	0.49 KiB
PNETCDF_FILE (ver=2) Module Data	0.11 KiB
LUSTRE (ver=1) Module Data	9.36 KiB
STDIO (ver=2) Module Data	0.08 KiB
APMPI (ver=1) Module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

I/O Summary

Heat Map: HEATMAP MPIIO



Heat Map: HEATMAP POSIX



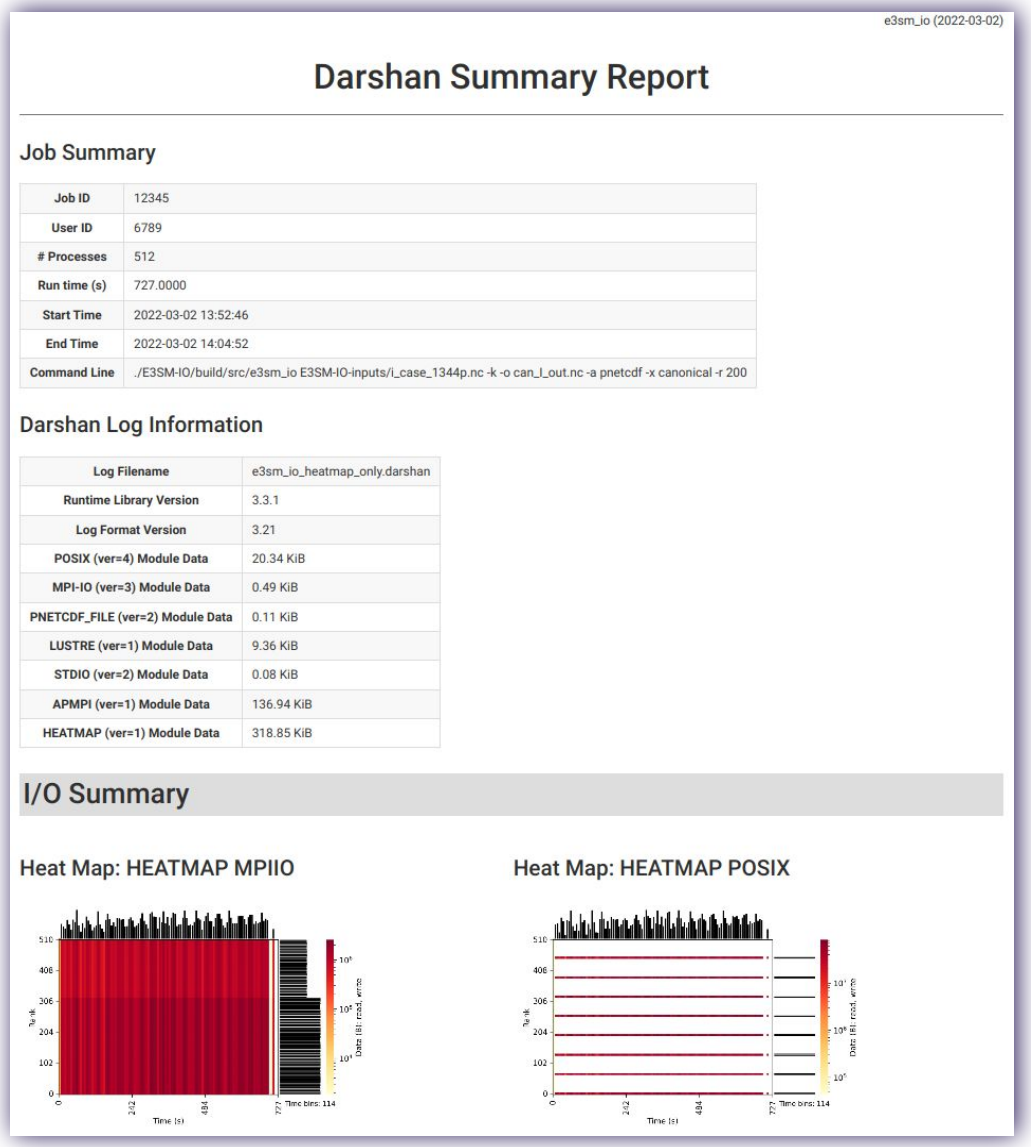
Tools: analyzing application I/O behavior

Application-level analysis tools are critical to *better understanding I/O behavior* and *informing potential tuning decisions*

Darshan is a lightweight I/O characterization tool commonly deployed at HPC facilities, including ALCF systems

- Transparent, low-overhead instrumentation of multiple layers of the HPC I/O stack
- Detailed counters/timers/statistics for each file accessed by the app stored in a condensed log
- Analysis tools for inspecting and presenting key information about I/O behavior (e.g., the Darshan job summary tool, **right**)

More on Darshan this afternoon in Breakout Session 3


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

Other tools

There are some other notable tools that may be of use for gaining more insights into the I/O behavior of an application:

- **Drishti:** github.com/hpc-io/drishti-io
  - Command-line tool transforming Darshan log data into a set of tuning recommendations based on common HPC I/O pitfalls
- **DXT Explorer:** github.com/hpc-io/dxt-explorer
  - Interactive web-based trace analysis tool for Darshan DXT trace data
- **TAU**: http://www.cs.uoregon.edu/research/tau/
  - General call profiling/tracing toolkit for HPC applications, including I/O routines
  - Tools for visualizing profiles/traces and detecting bottlenecks, etc.
  - See: https://hps.vi4io.org/_media/events/2019/sc19-analyzing-tau.pdf
- **Recorder**: github.com/uiuc-hpc/Recorder
  - Multi-level detailed traces and corresponding trace viz tools

Sharing data with collaborators

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

Sharing data with collaborators

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

More on Globus this afternoon in Breakout Session 2 (Greg Nawrocki, University of Chicago)

A recap

Today we have covered various software technologies that comprise the HPC I/O stack, which is used to persist, manage, and share large-scale scientific datasets

- *Parallel file systems* offer high-performance, scalable file storage
- *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 app data abstractions
- *Tools* are available for better understanding and, ideally, improving I/O performance
- *File transfer/sharing* mechanisms to enable collaboration

Always consider facility documentation and other resources to help understand general best practice and reach out on support channels for help if you're stuck!

Check out these hands-on sessions this afternoon to learn more:

- **Darshan** (room 1406, Shane Snyder, ANL)
- **MPI-IO** & **HDF5** (room 1404, Rob Latham, ANL)
- **Globus** (room 1405, Greg Nawrocki, University of Chicago)

Thank you!