CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 16: Pre-spring break Recap

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

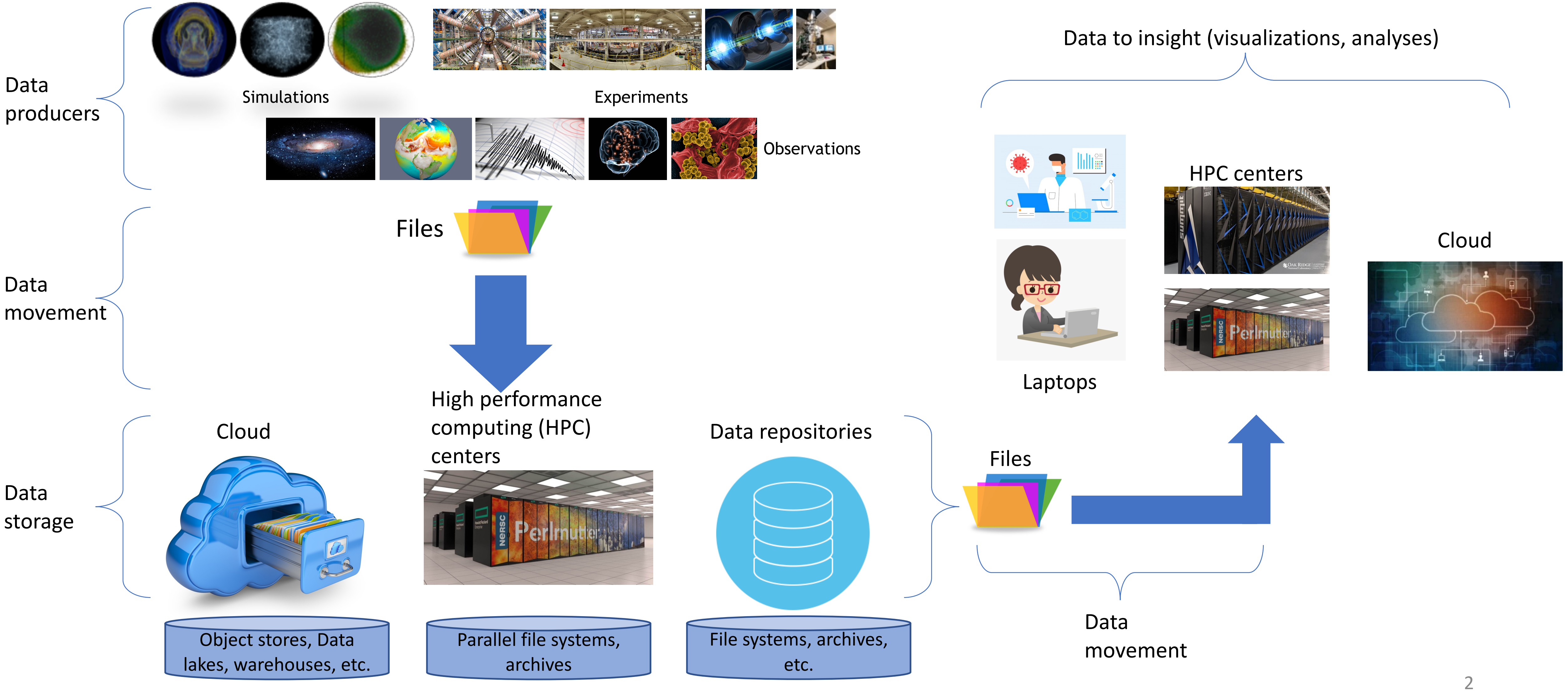
https://sbyna.github.io

03/07/2023

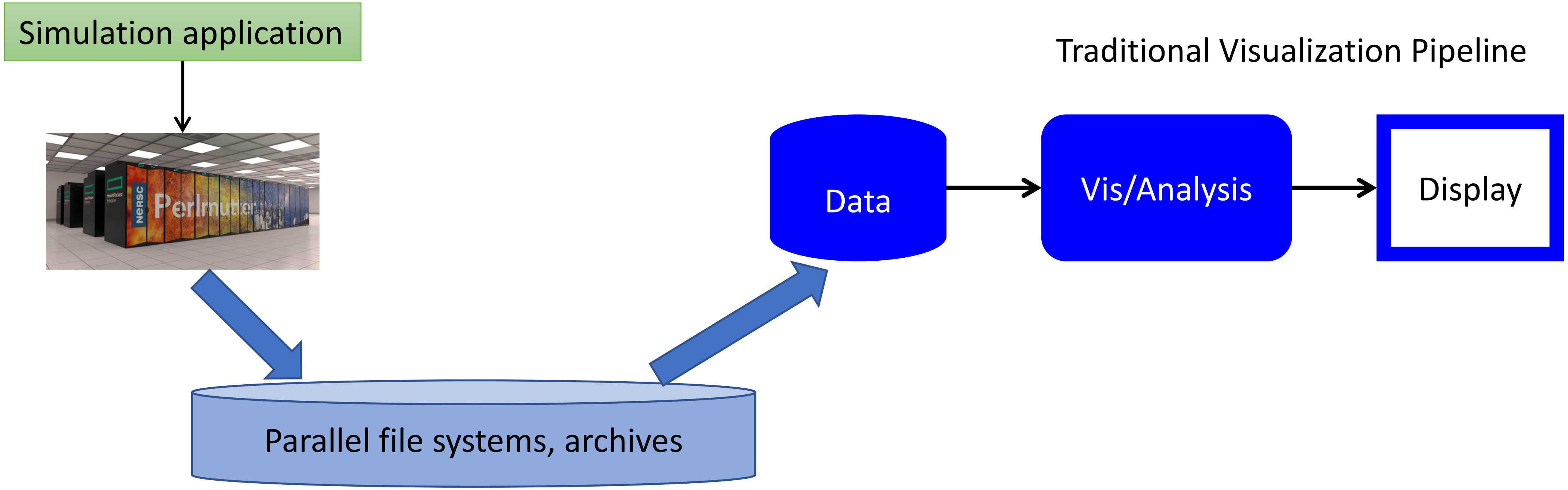
Today's class

- Any questions?
- Progress of the project
- Recap of all the topics discussed so far

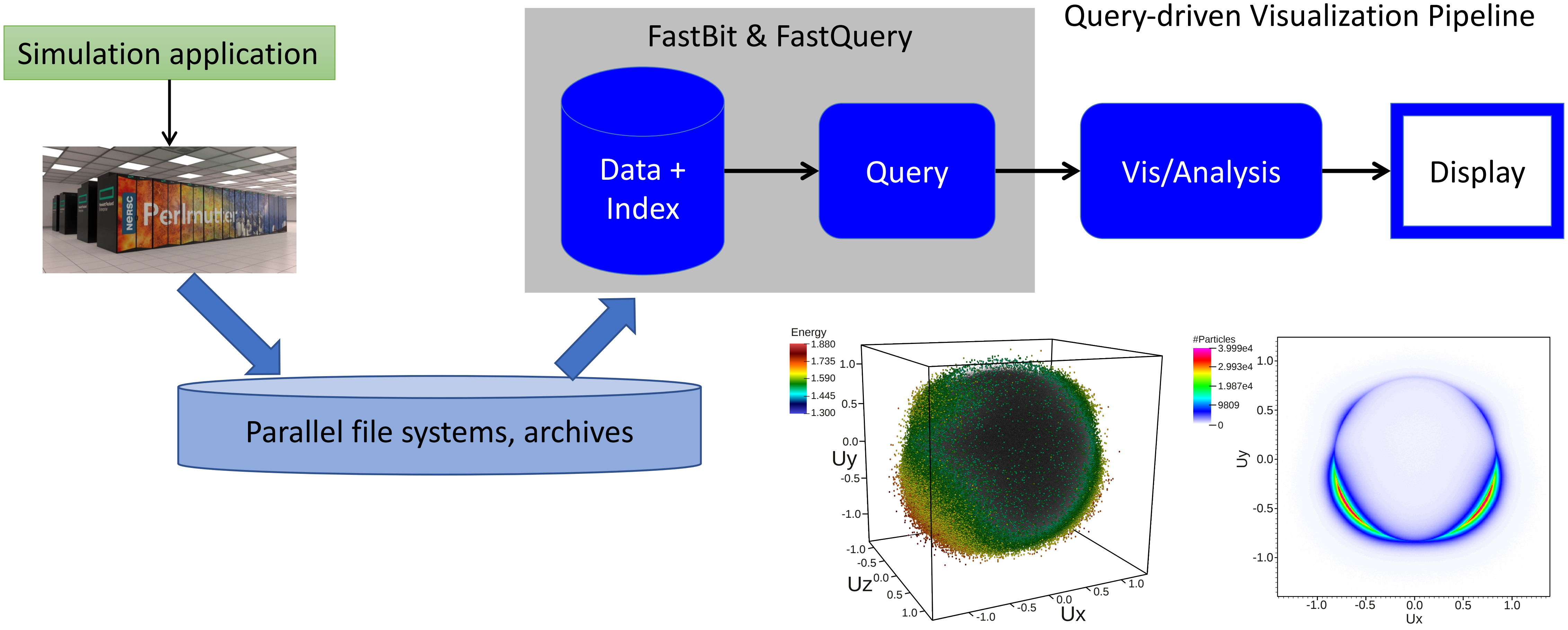
Data life cycle - An overview



Data life cycle of a plasma physics simulation and analysis

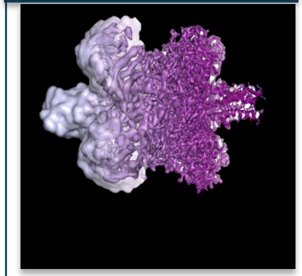


Data life cycle of a plasma physics simulation and analysis



Data life cycle of experimental & observation use cases

Acquire

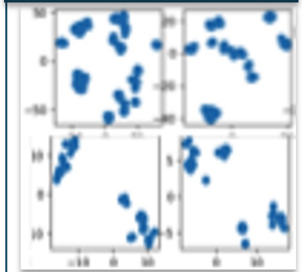


Collect from sensors, experiments, simulations

Transfer

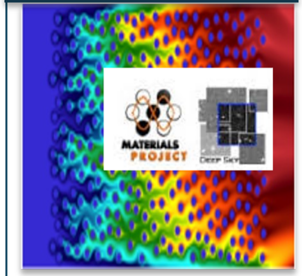
Move from instrument to computing center (supercomputing / cloud)

Clean



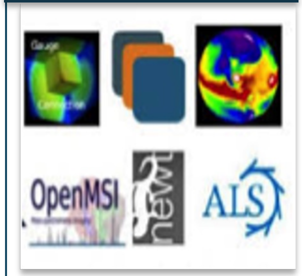
Organize, annotate, filter, encrypt, compress

Use/Reuse



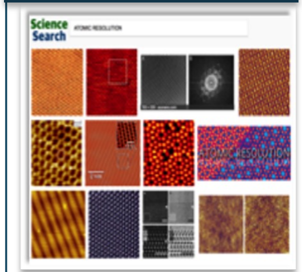
Analyze, mine, model, learn, infer, derive, predict

Publish



Disseminate & aggregate, using portals, databases

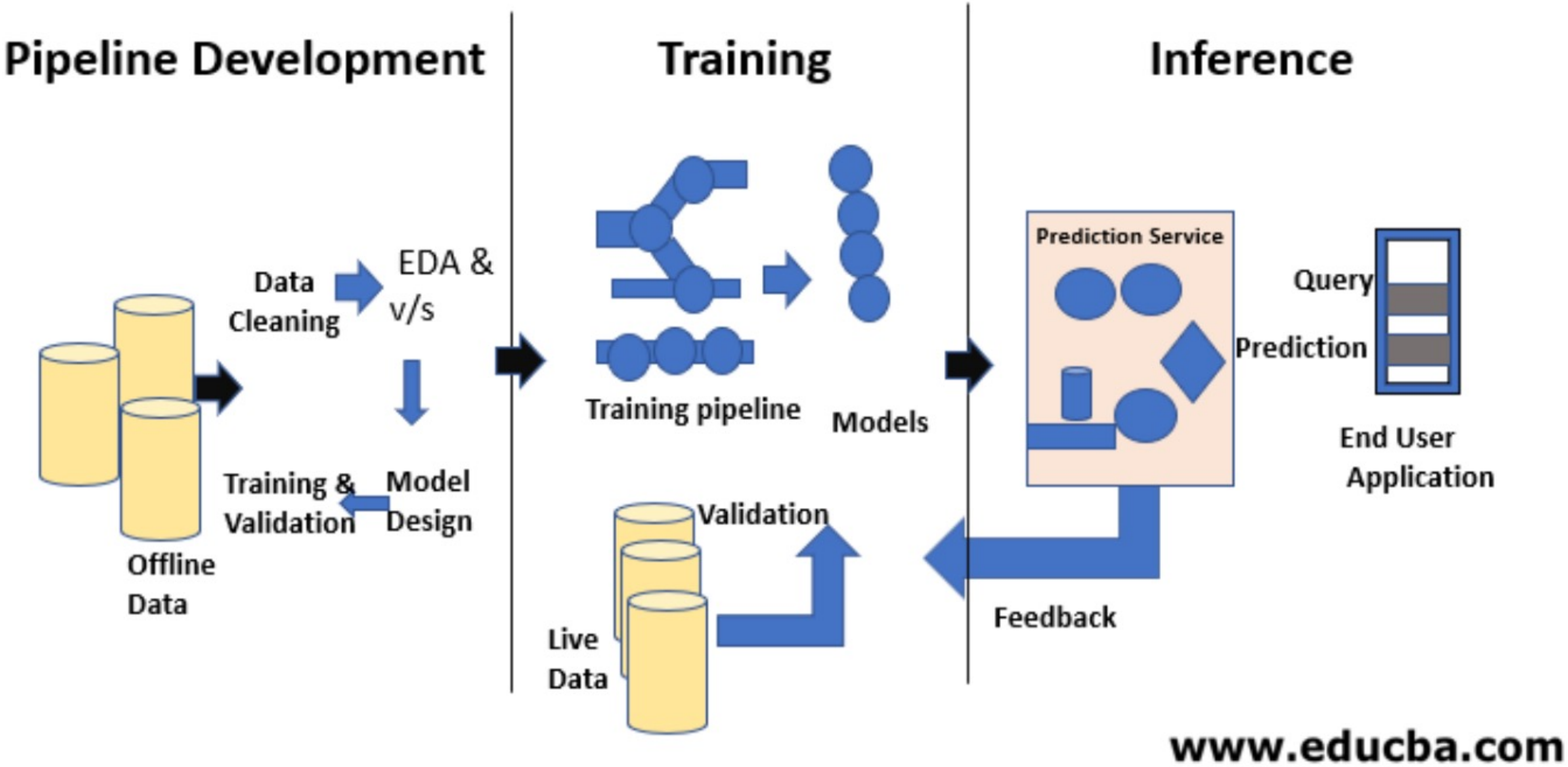
Preserve



Index, curate, age, track provenance, purge

From a slide from Debbie Bard's SuperFacility presentation

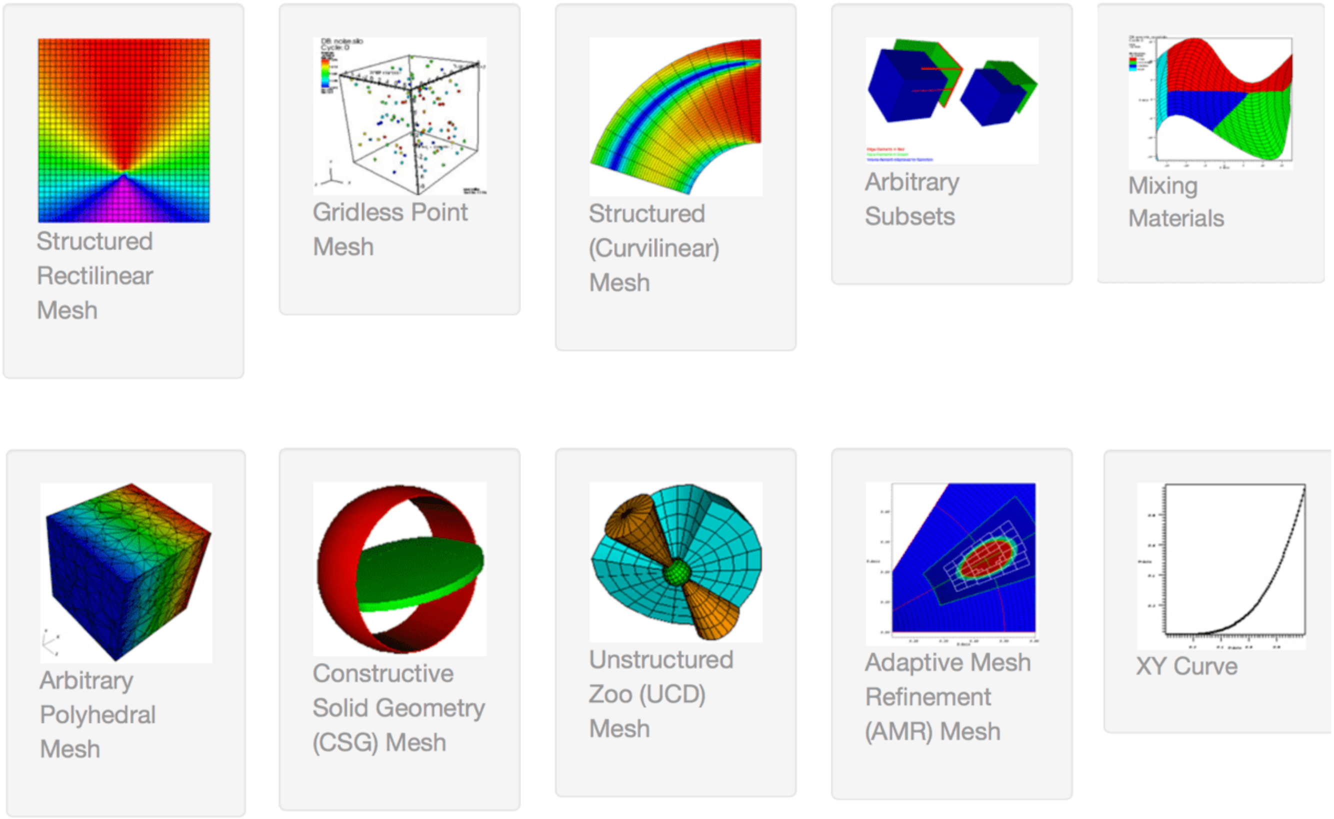
Machine learning life cycle



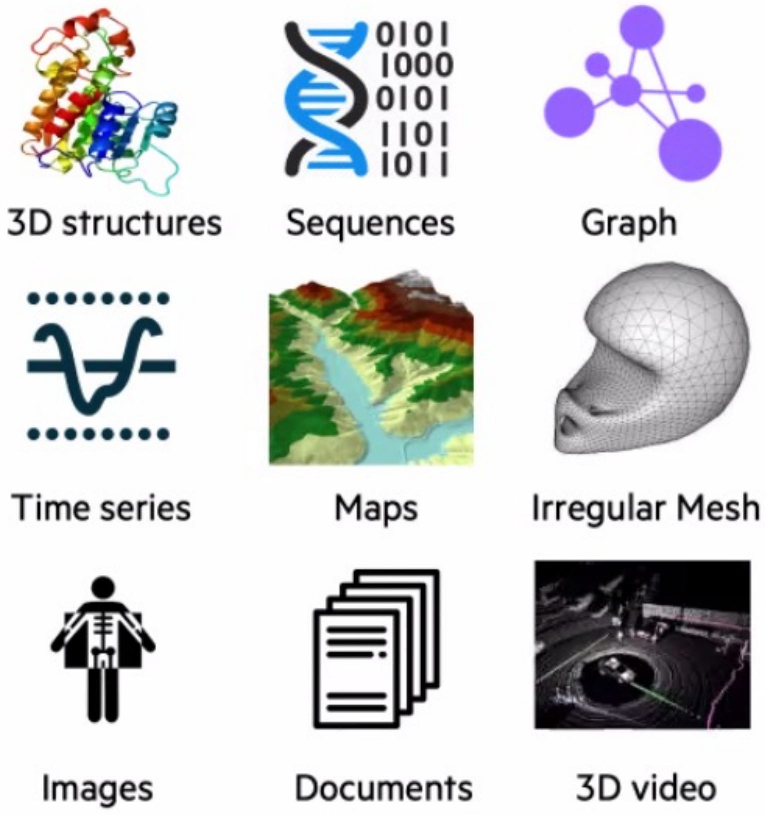
Source: https://www.educba.com/machine-learning-life-cycle/

What does scientific data look like

Traditional types of data - modeling and simulation



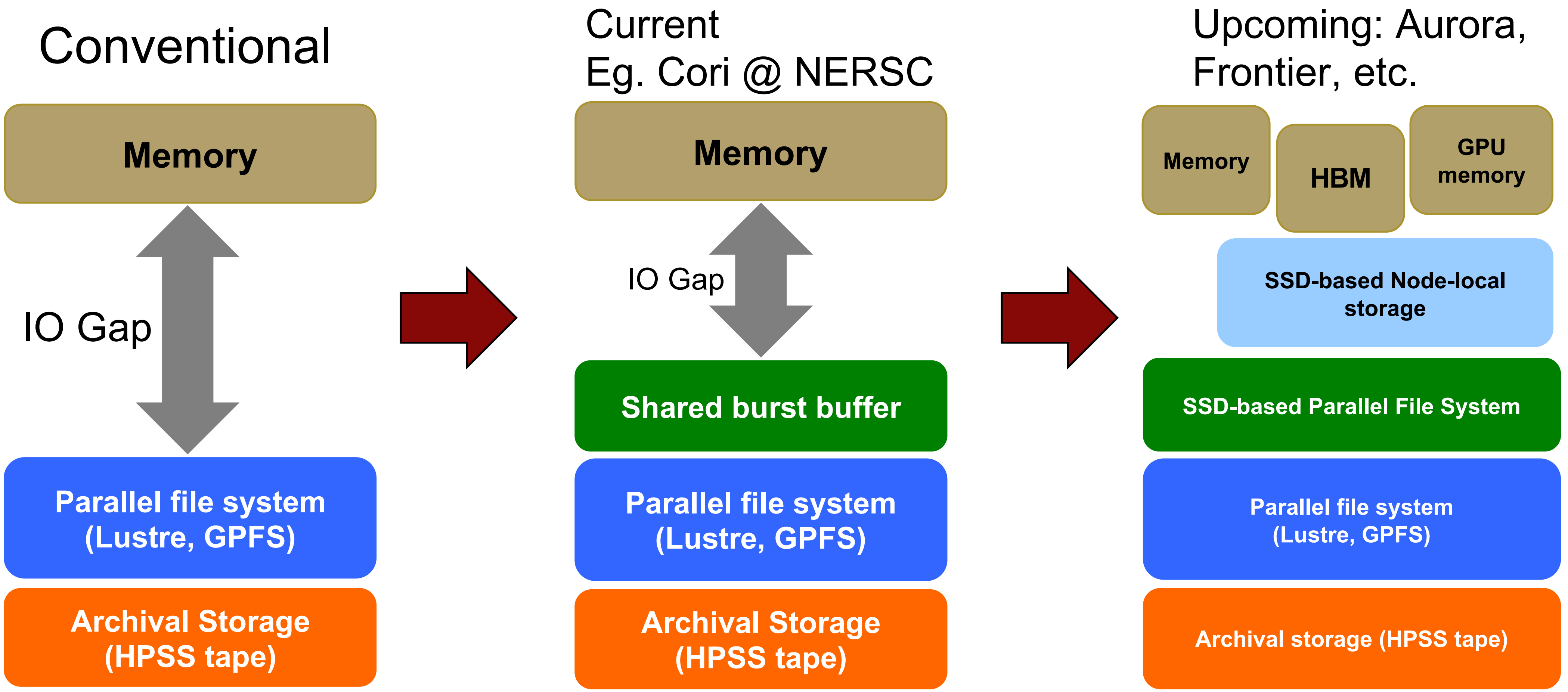
Typical data used for AI / ML



Source: MACSIO, LLNL
https://github.com/LLNL/MACSio/blob/master/doc/scientific_data_objects.png

Source: Rangan Sukumar's slides presented at Monterey Data Workshop on 04/21/2022

Storage systems in high performance compute systems



Different types of parallelism – Flynn's taxonomy

- Problem – Data stream
- Work – Instruction stream

- Single
- Multiple

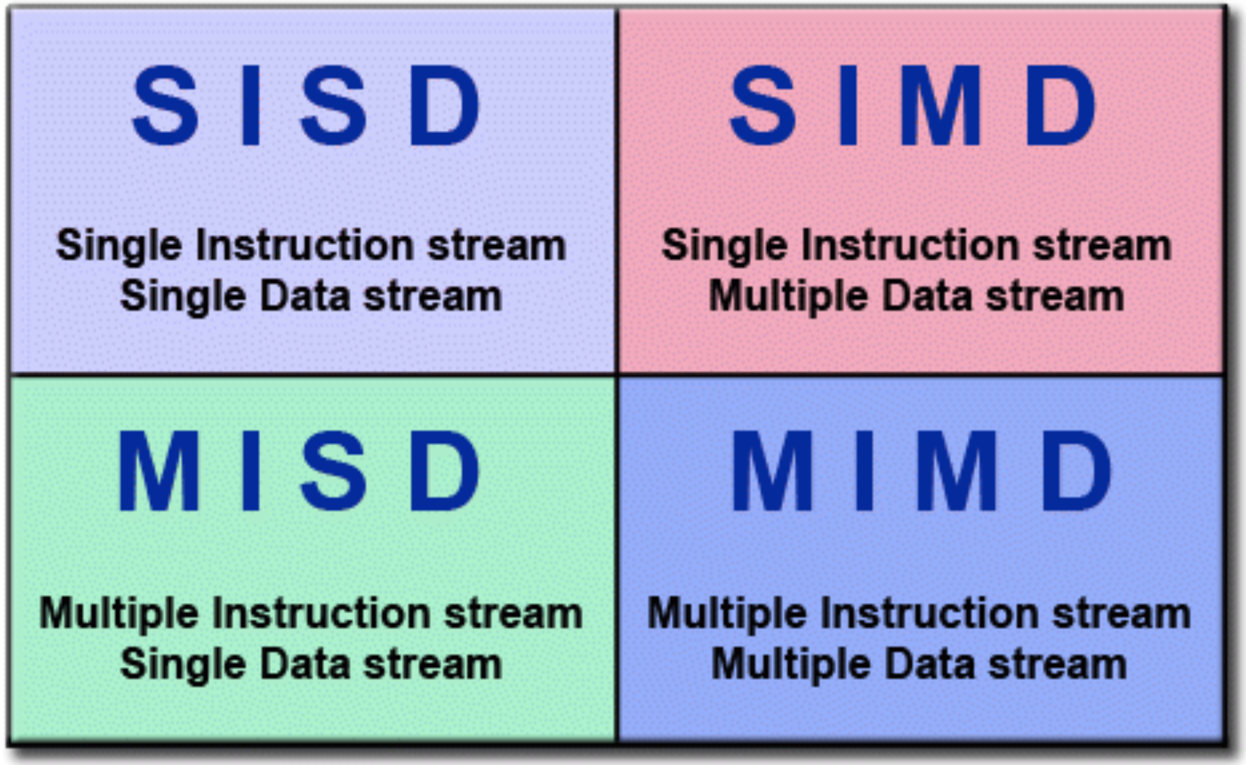
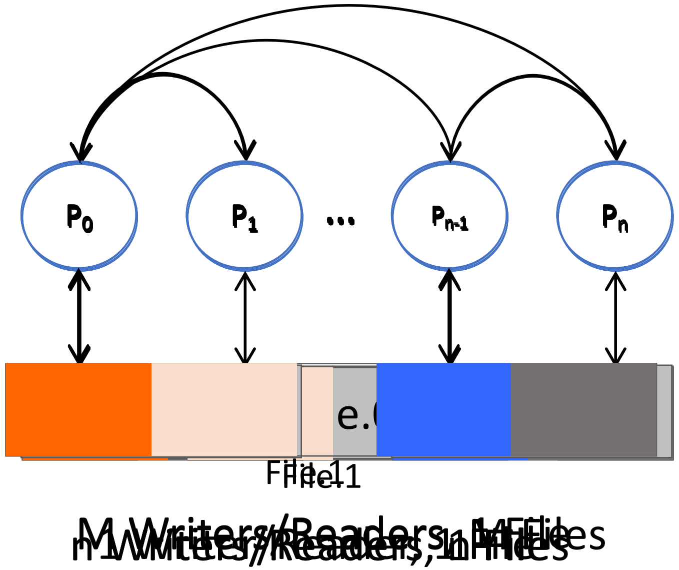


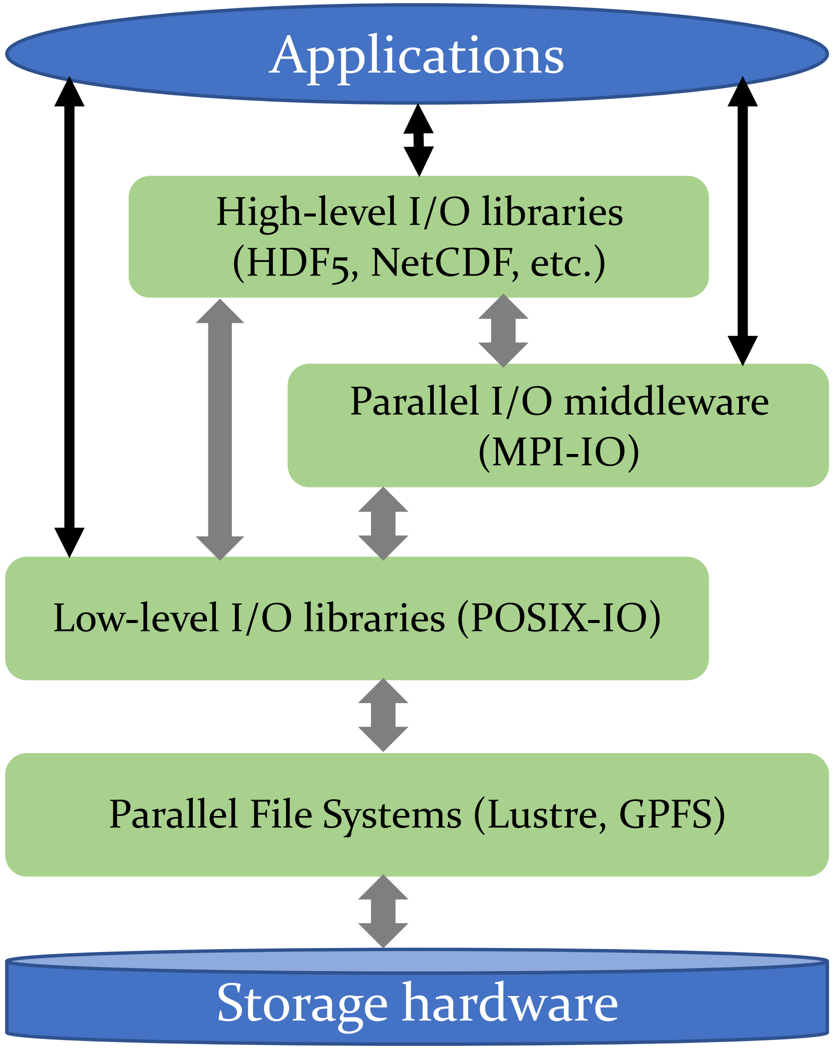
Image from LLNL parallel computing tutorial
https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial

Parallel I/O – Application view

- **Types of parallel I/O**
  - 1 writer/reader, 1 file
  - N writers/readers, N files (File-per-process)
  - N writers/readers, 1 file
  - M writers/readers, 1 file
    - Aggregators
    - Two-phase I/O
  - M aggregators, M files (file-per-aggregator)
    - Variations of this mode

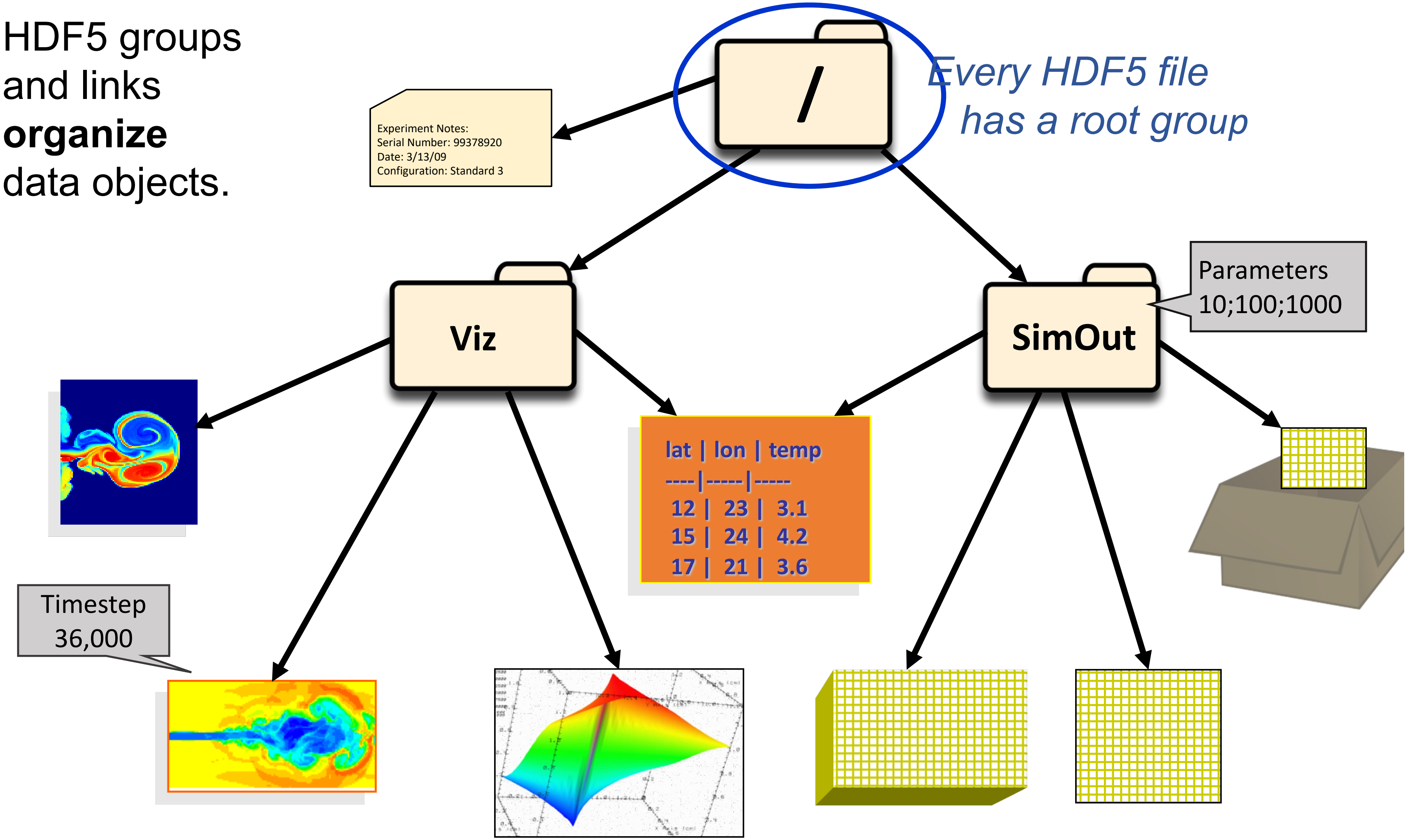


Data storage and access – Software layers in HPC systems

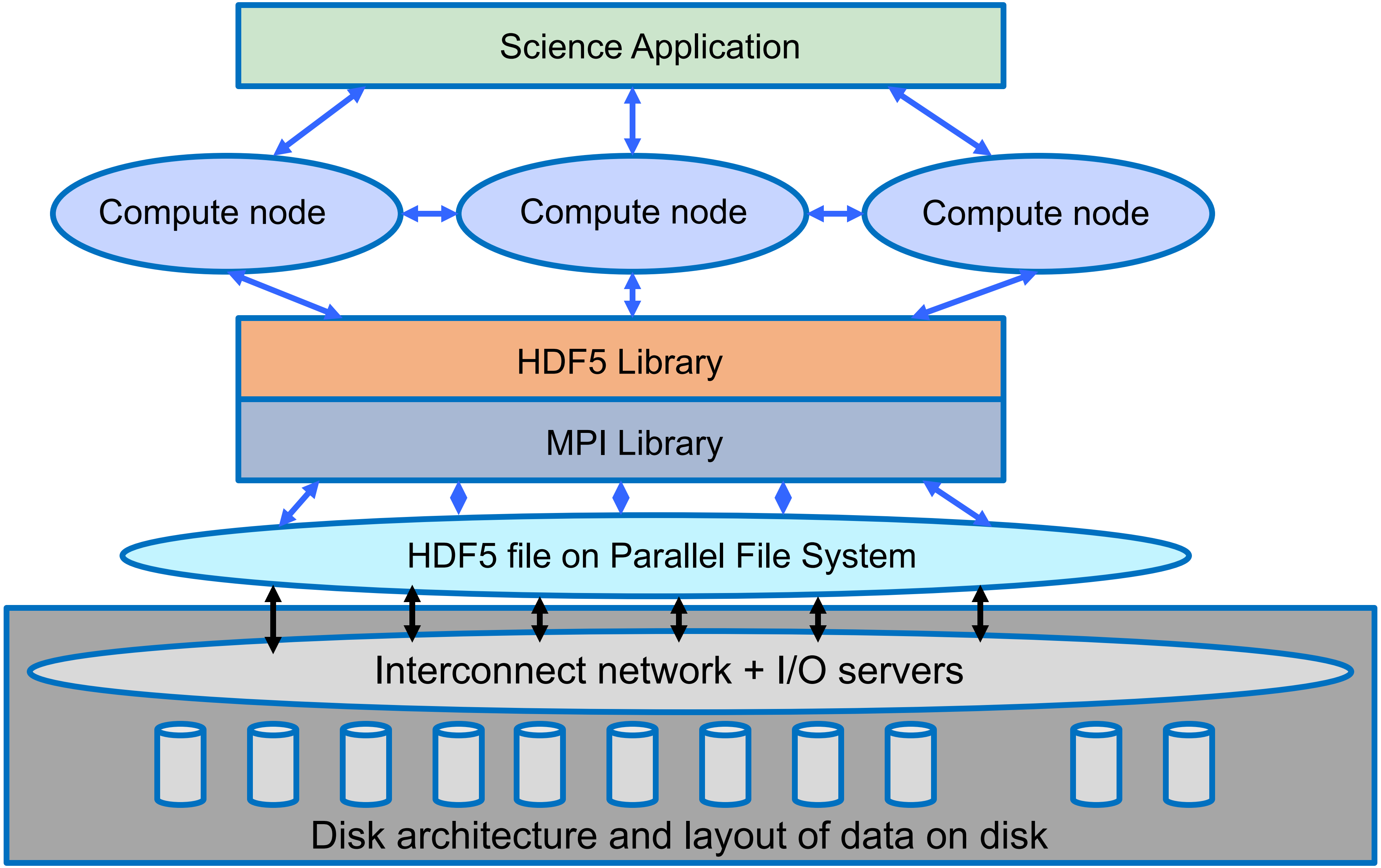


HDF5 Groups and Links

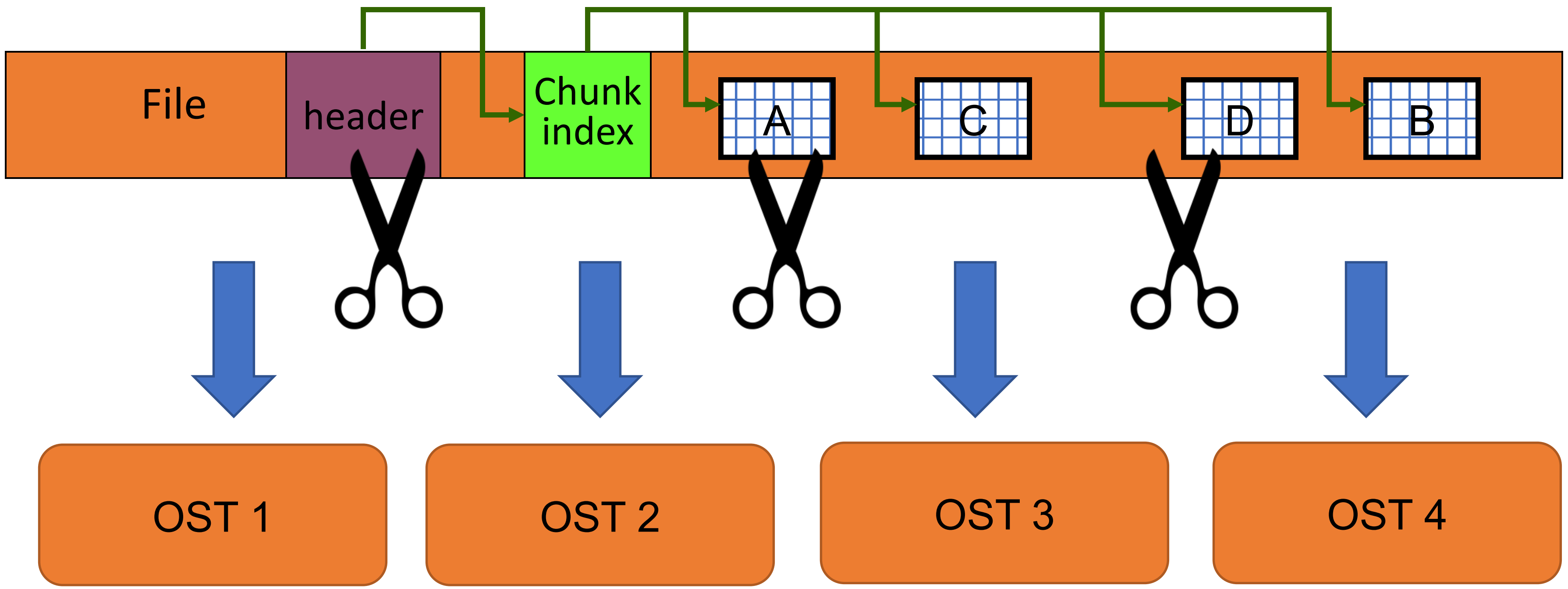
HDF5 groups and links **organize** data objects.



PHDF5 Implementation Layers



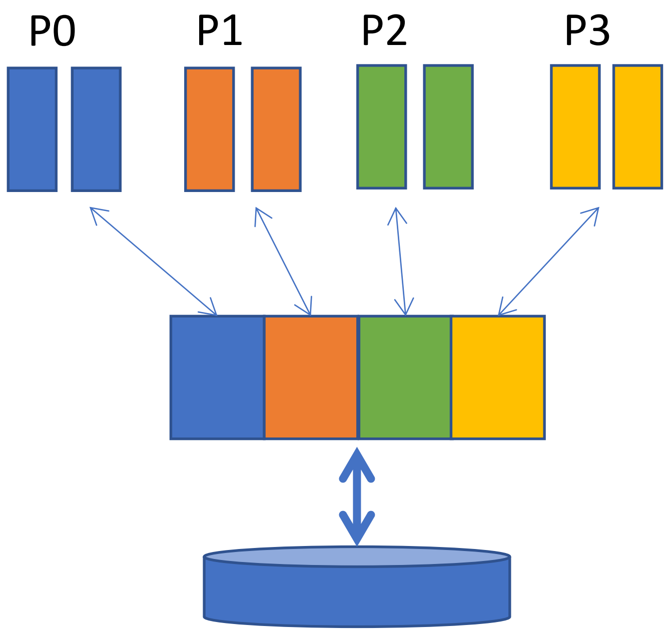
In a Parallel File System



The file is striped over multiple OSTs depending on the stripe size and stripe count with which the file was created.

MPI-IO performance optimizations – Collective buffering

- Also known as two-phase I/O
- A few processes aggregate data to temporary buffers and the data is then written to file (collective write operations)



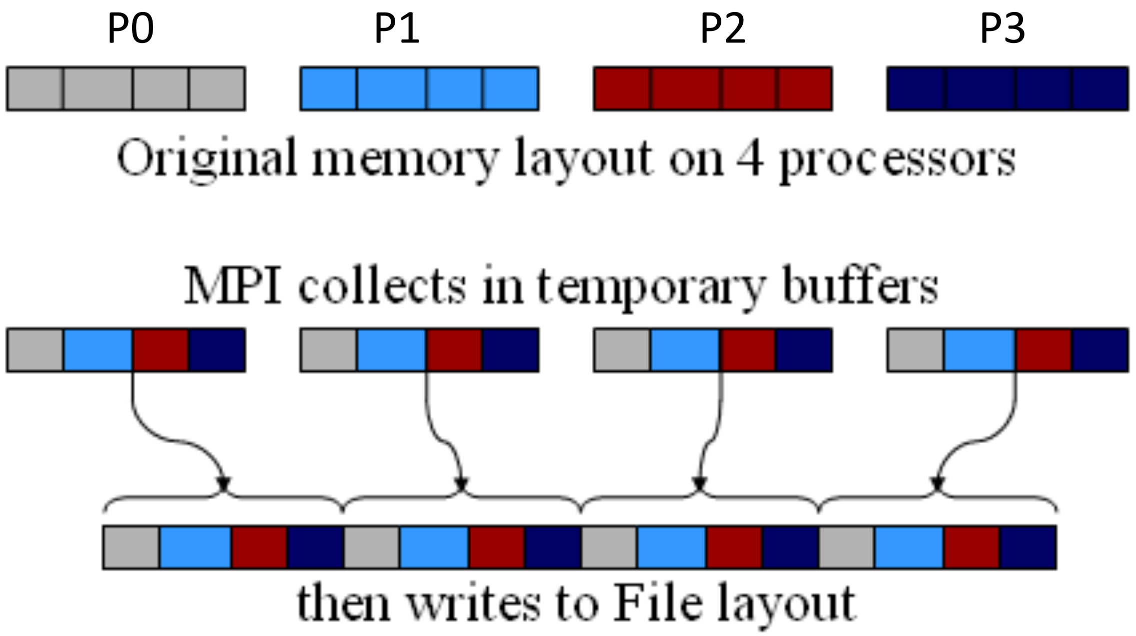


Image from https://cvw.cac.cornell.edu/ParallelIO/choreography

NetCDF - to store multiple arrays in a single file with metadata

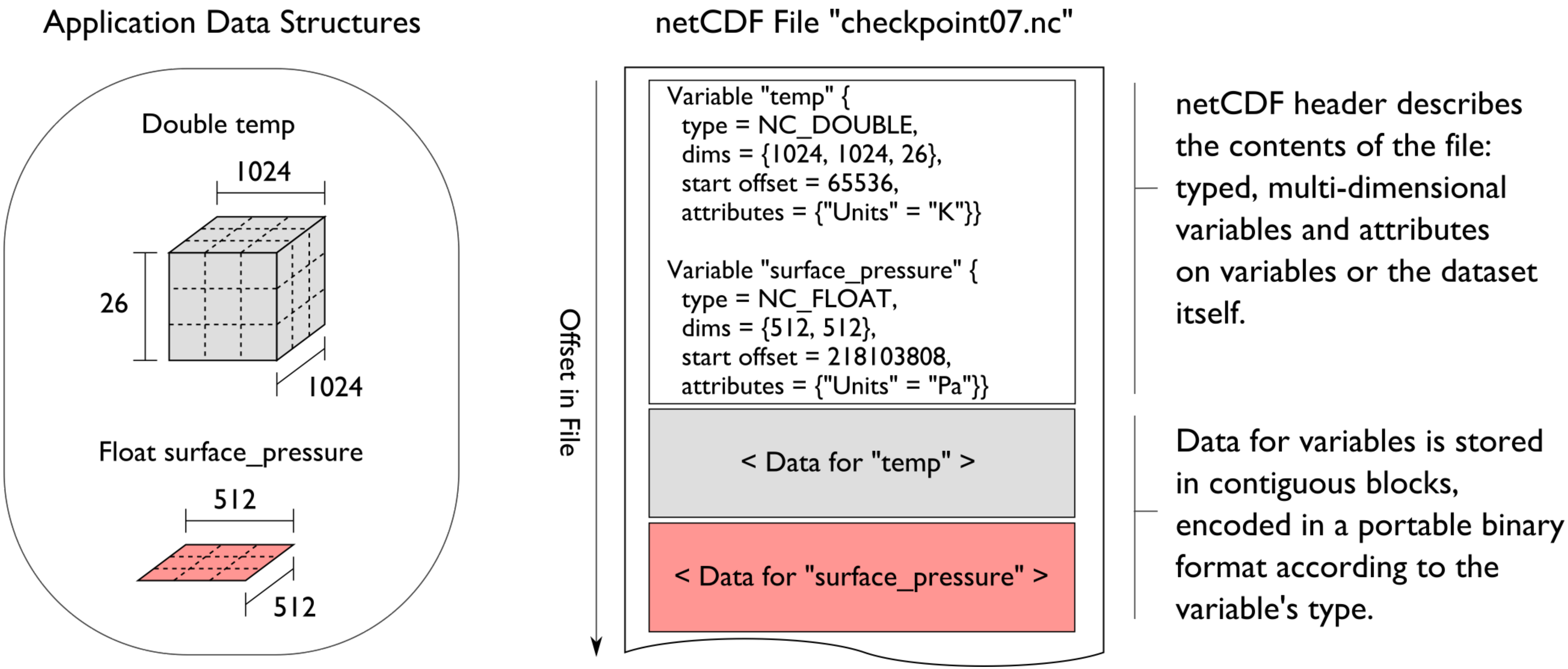


Image from Pnetcdf tutorial / Bill Gropp's slides on MPI-IO

PnetCDF

- PnetCDF is a high-performance parallel I/O library for accessing NetCDF files
- Parallel I/O library by using MPI-IO

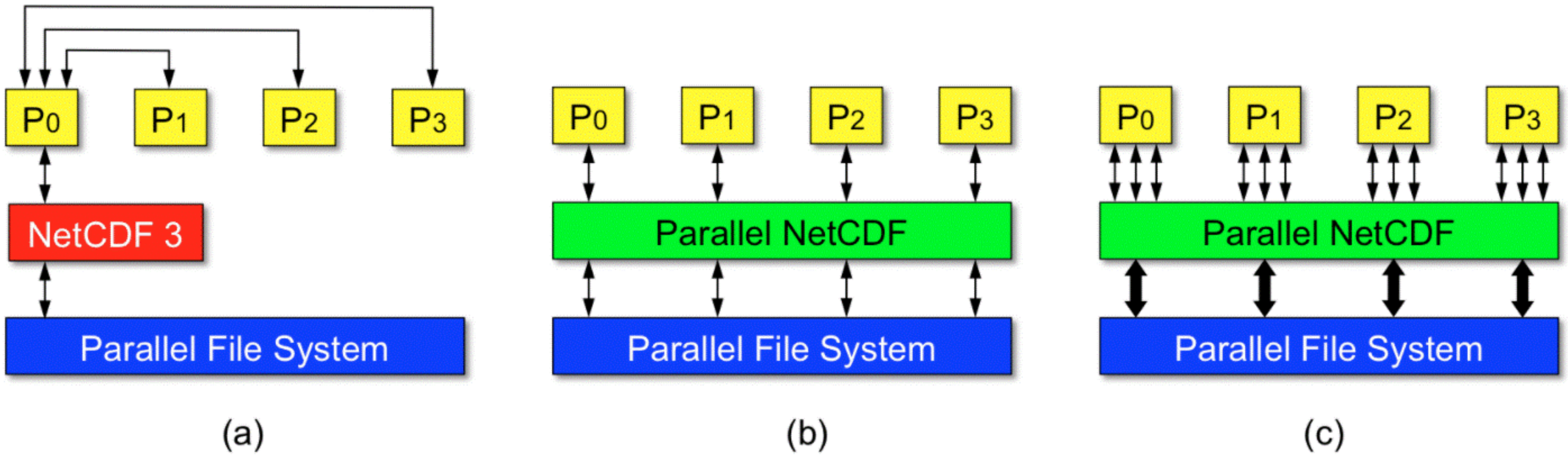


Figure 1. Comparison of data access between using sequential netCDF and PnetCDF. (a) Write operation is carried out through one of the clients when using the sequential netCDF prior to version 4.0. (b) PnetCDF enables concurrent write to parallel file systems. (c) Through nonblocking I/O, PnetCDF can aggregate multiple requests into large ones so a better performance can be achieved.

Image source: cucis.ece.northwestern.edu/projects/PnetCDF

ADIOS2

- ADaptable I/O System 2
- Development led by Oak Ridge National Laboratory

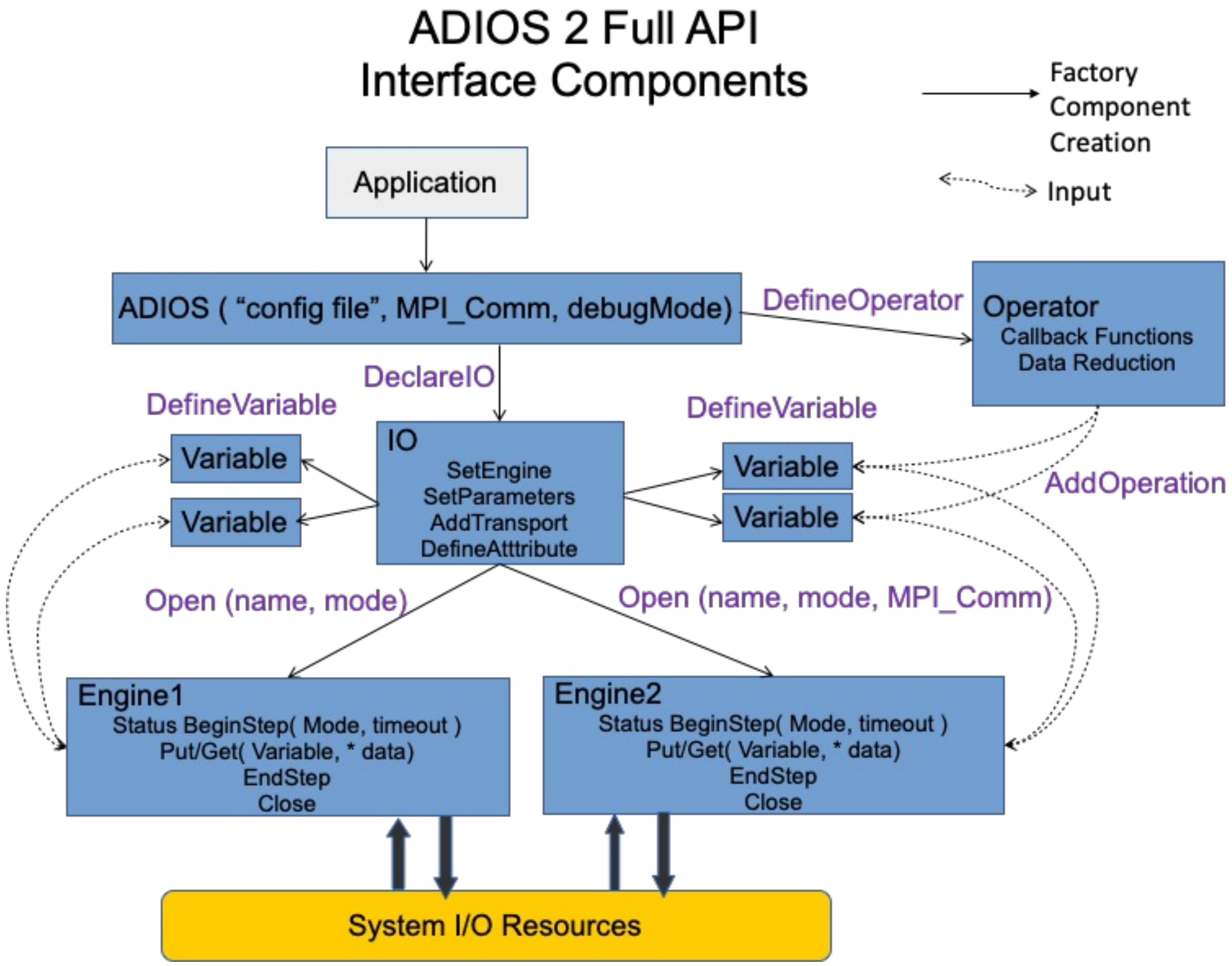


Image source: https://adios2.readthedocs.io/

File system – Lustre architecture

- Lustre
- Main components
  - Metadata server (MDS)
  - Object storage servers (OSS)
  - Object storage targets (OSTs)

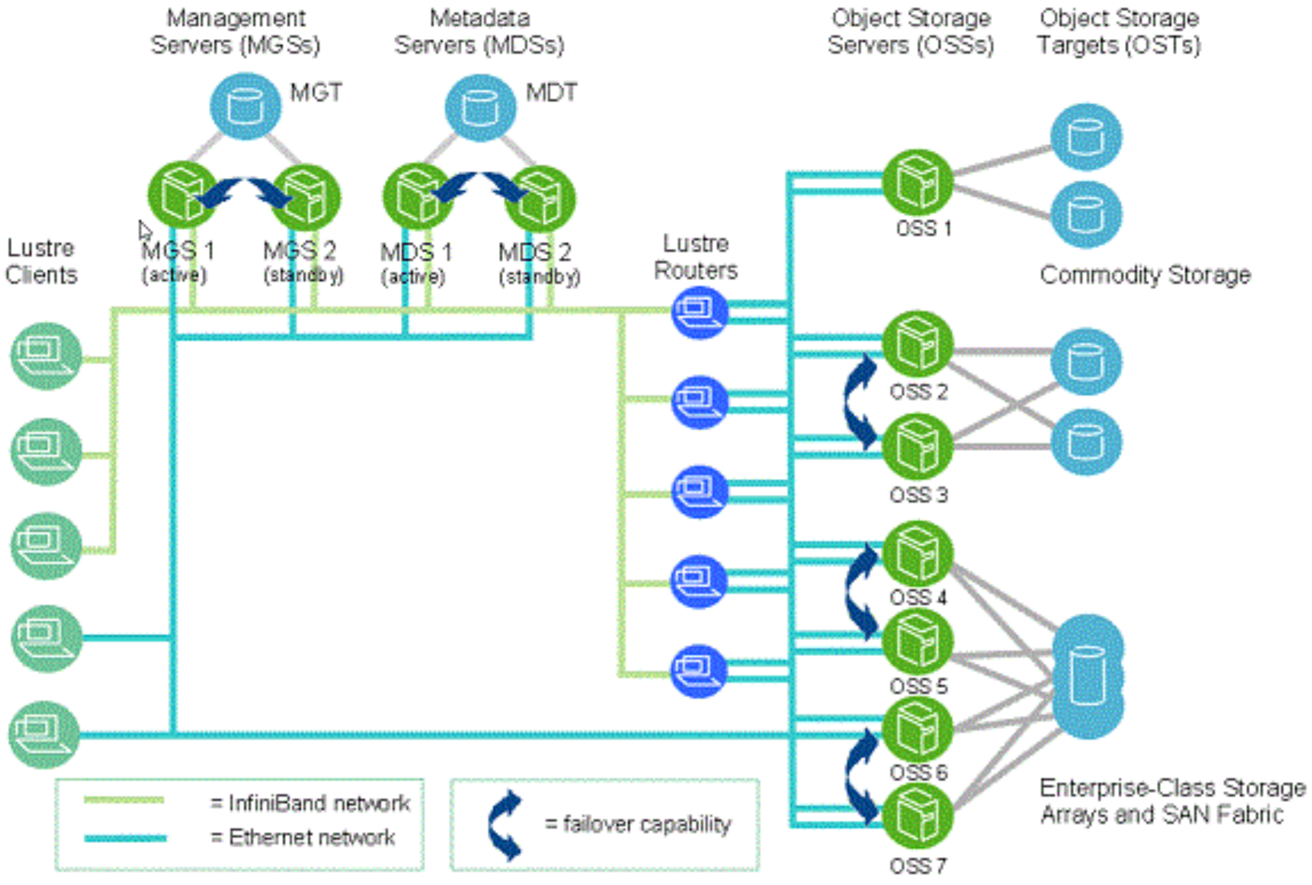
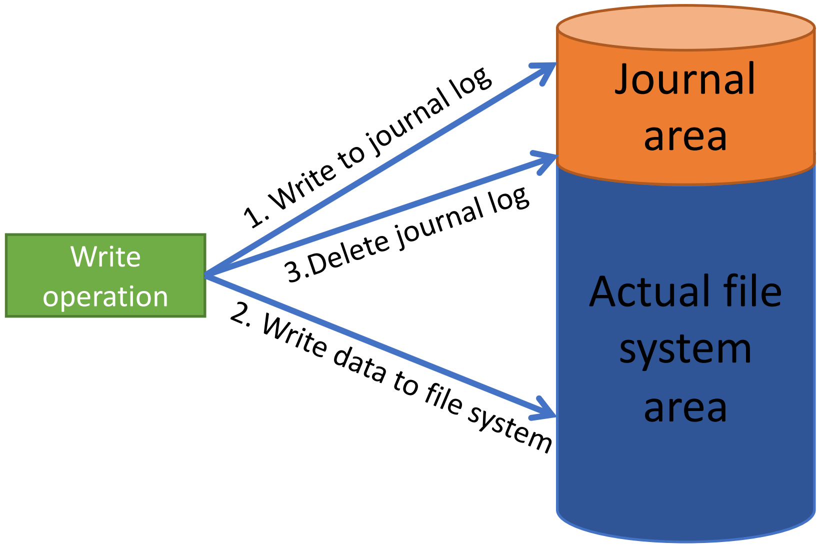


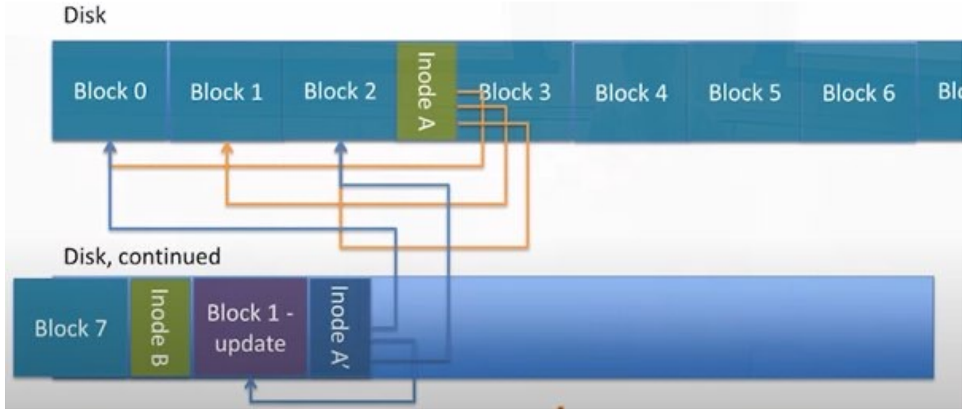
Image from: https://www.weka.io/learn/lustre/lustre-file-system-explained

Journaling File System

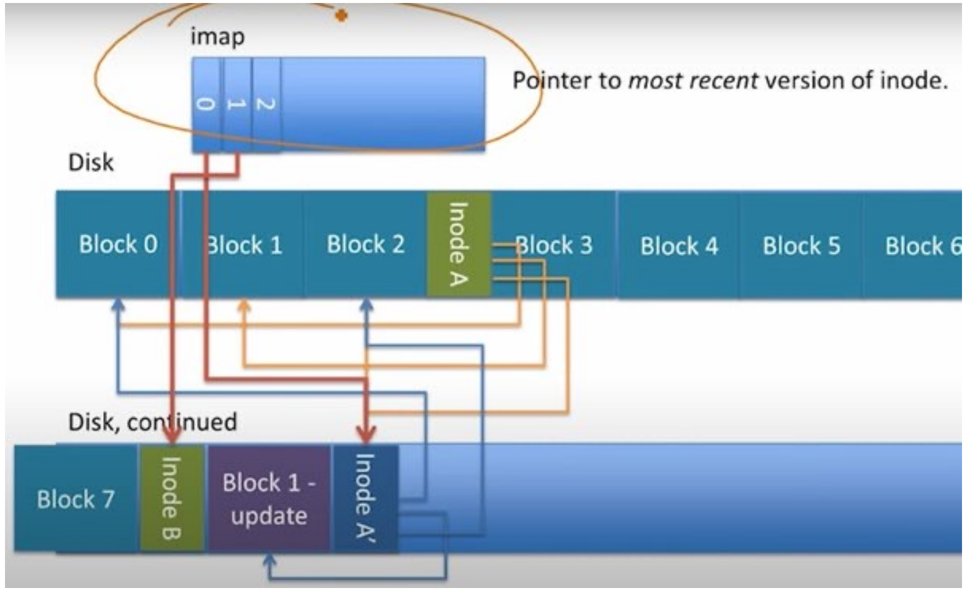
- A journal to keep track of uncommitted file system operations
- A separate data structure is used for keeping track of records – Journal



Log-structured File System



- Instead of making changes to the journal and file system separately, logs are embedded into the file system
- Blocks of data are never modified
  - An update operation places a new block at the end of the file
  - Writes always go to the end of the file



More details: https://pages.cs.wisc.edu/~remzi/OSTEP/file-lfs.pdf

Image source: https://www.quora.com/What-is-the-difference-between-a-journaling-vs-a-log-structured-file-system

Factors that impact parallel I/O performance

Applications

- Number of MPI ranks
- Number of I/O requests
- Size of I/O requests
- Number of files
- Number of metadata calls
  - File open and close requests
- Number of seek operations
- Contiguous / non-contiguous requests
  - Number of seeks
- Alignment of I/O request with
  - File block
  - Sub-files
- Shared file or multiple files
- …

High-level I/O library

- Metadata operations for self-describing property
- Location of metadata
- How many processes are participating in metadata or data operations
- Alignment in file offsets
- Hyperslab selections
  - contiguous / non-contiguous?
  - complex hyperslabs construction cost
- Chunking
  - Chunk size
  - Number of chunks
- Sub-files
  - How many? How's the data aggregated?
- Compression used or not?
  - What's the compression / decompression cost?
  - Where is compression / decompression executed?
- Does a file need to be exact size of data or can it have some gaps?
- Cache metadata or not?

MPI-IO

- Contiguous / non-contiguous accesses
- Number of I/O requests
- Size of I/O requests
- POSIX consistency semantics
- Synchronous / Asynchronous I/O calls
- Collective or independent
- If collective:
  - Number of aggregators
  - Aggregator placement
  - Aggregation buffer size
  - Aggregator to file system mapping – network connections and block sizes

File systems

- Number of storage servers
- Number of metadata servers
- Number of storage targets (stripe count)
- Block size on storage server
- Page size on storage target
- Amount of contiguous data stored on a storage target (stripe size)
- Traffic on storage targets
- Fullness of storage targets
- Fragmentation on storage targets

Darshan – How does it work?

- `darshan-runtime` and `darshan-util`

- Instrumentation of I/O calls
  - At link time of application OR
  - At runtime (using LD_PRELOAD)

- Collects file access statistics
  - HDF5, MPI-IO, POSIX-IO, File system layers
  - Computes statistics
  - Compresses the logs and writes

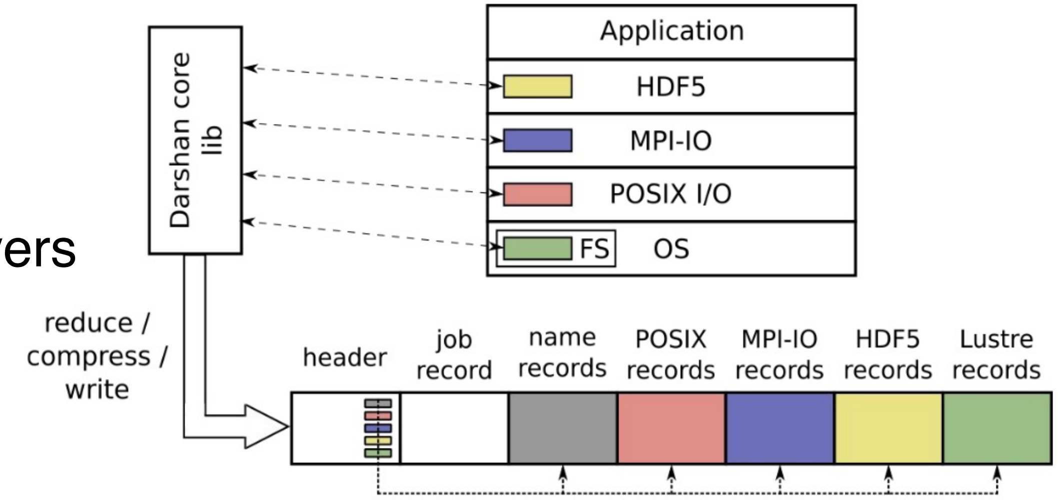
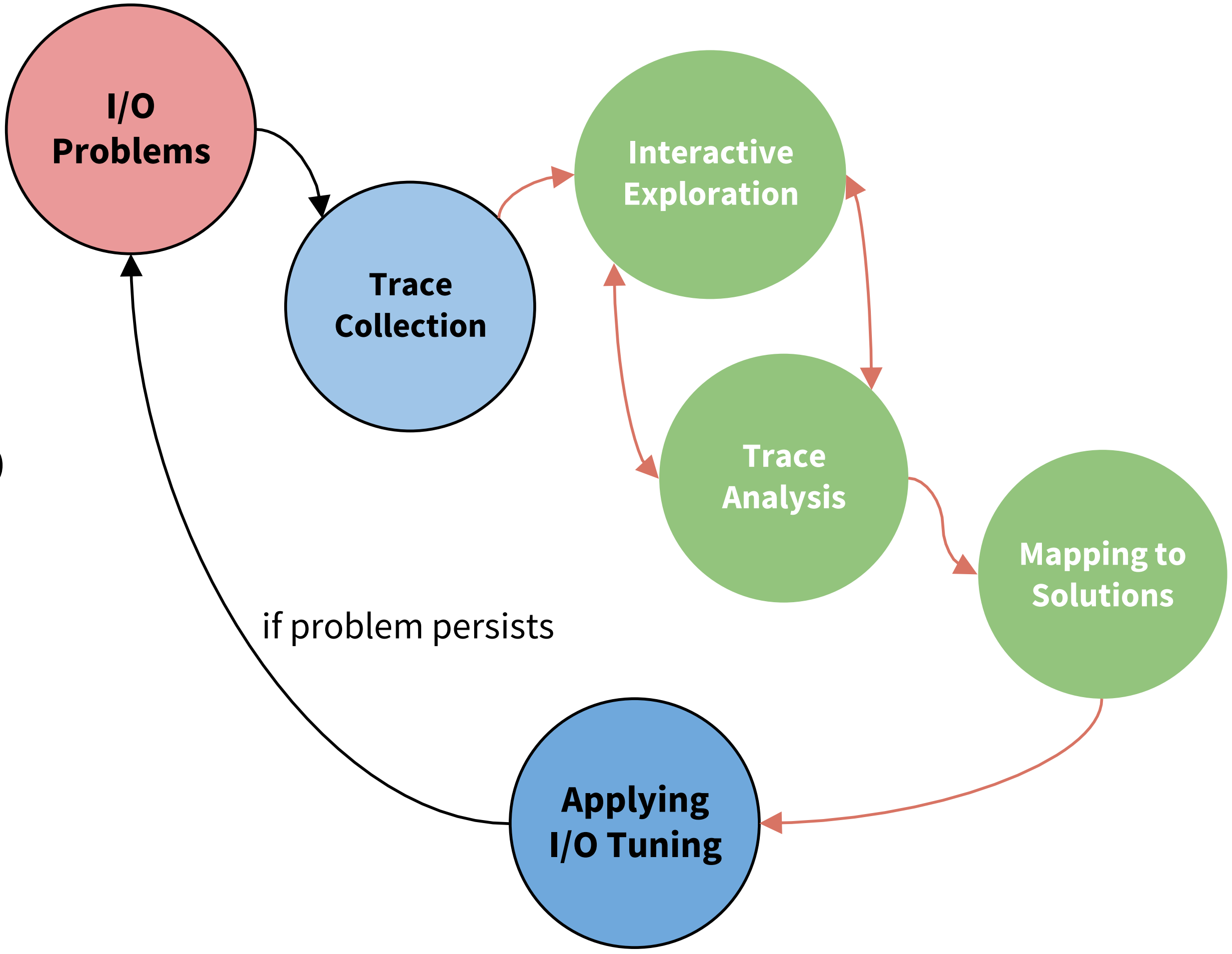


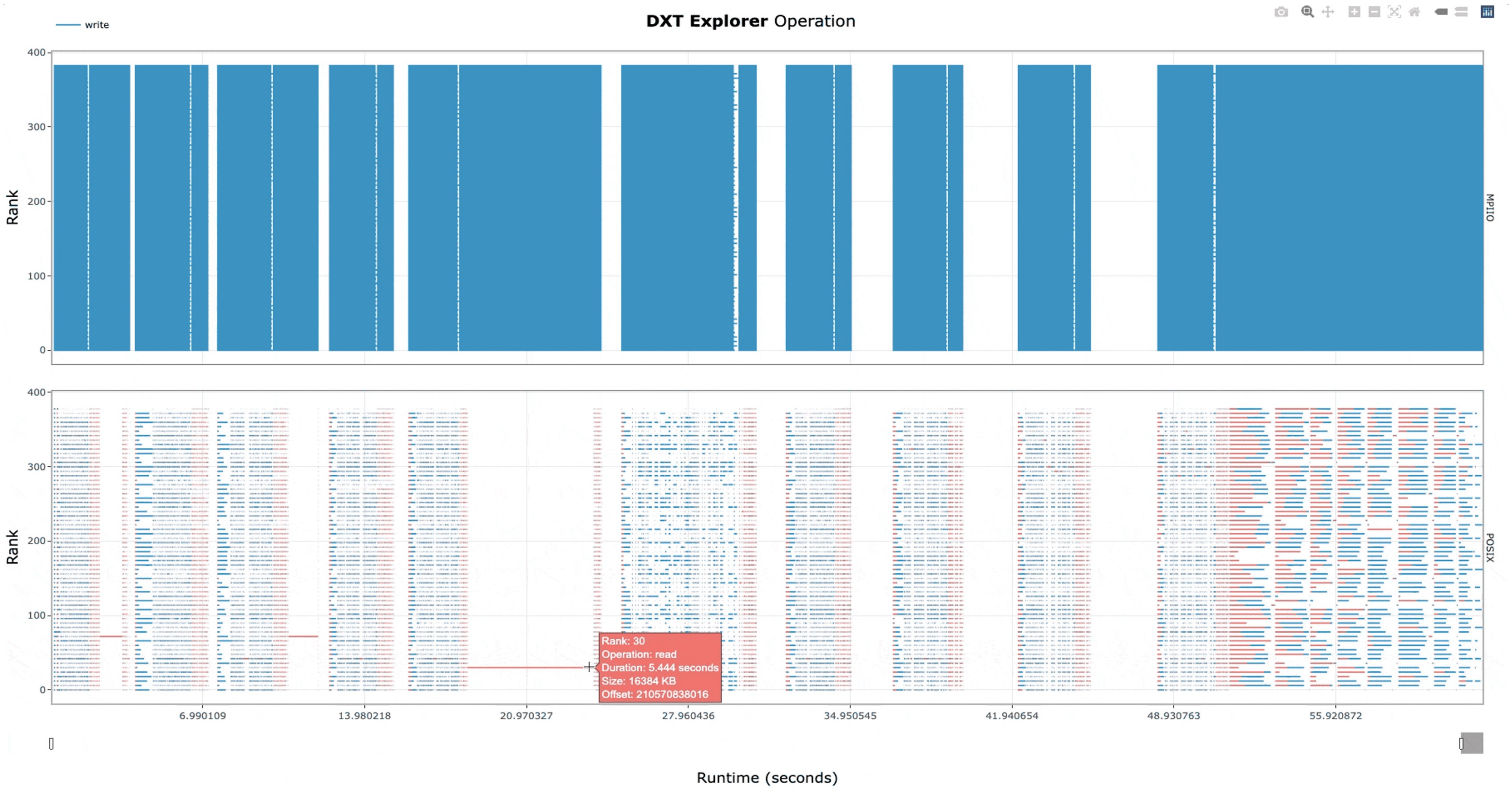
Image from Shane Snyder's Darshan tutorial

DXT Explorer

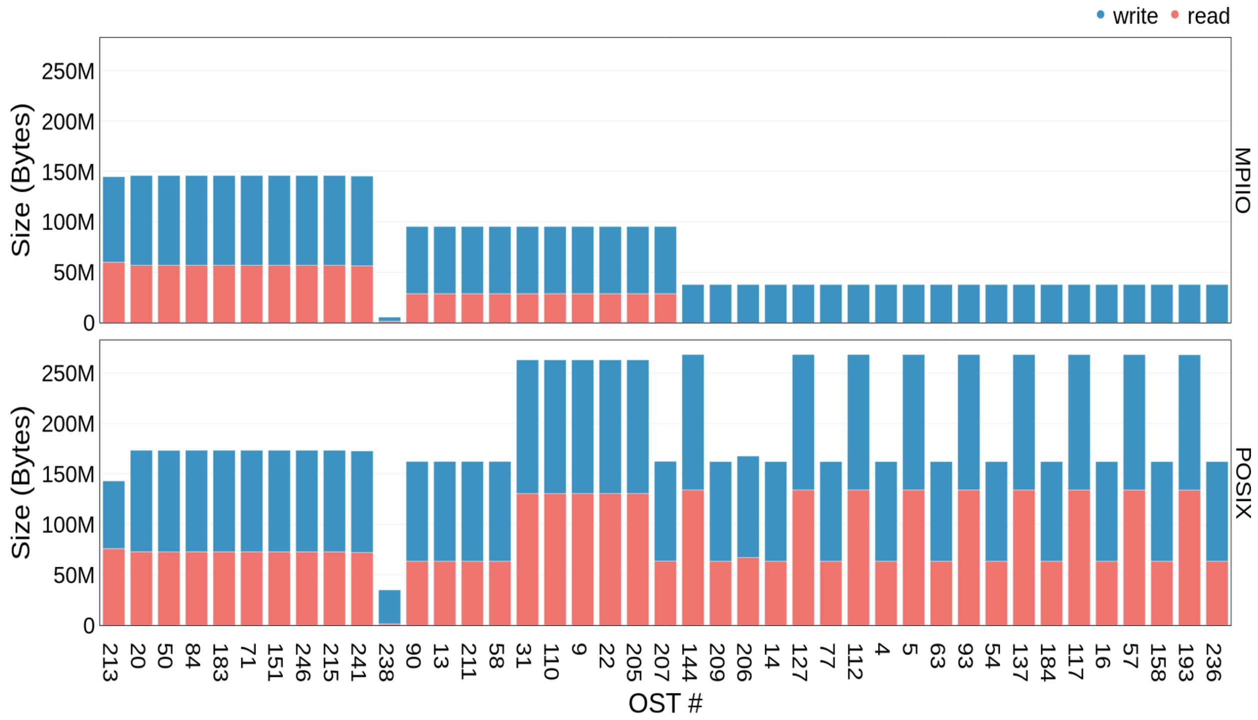
- **DXT Explorer**
  - Analyze the I/O traces interactively
  - Diagnose and highlight the bottlenecks
  - Provides an actionable set of recommendations
- Provides an interactive component to I/O traces
  - Users can visually inspect the I/O behavior
    - Zoom in areas of interest
  - End users provided with solution recommendations
    based on detected bottlenecks

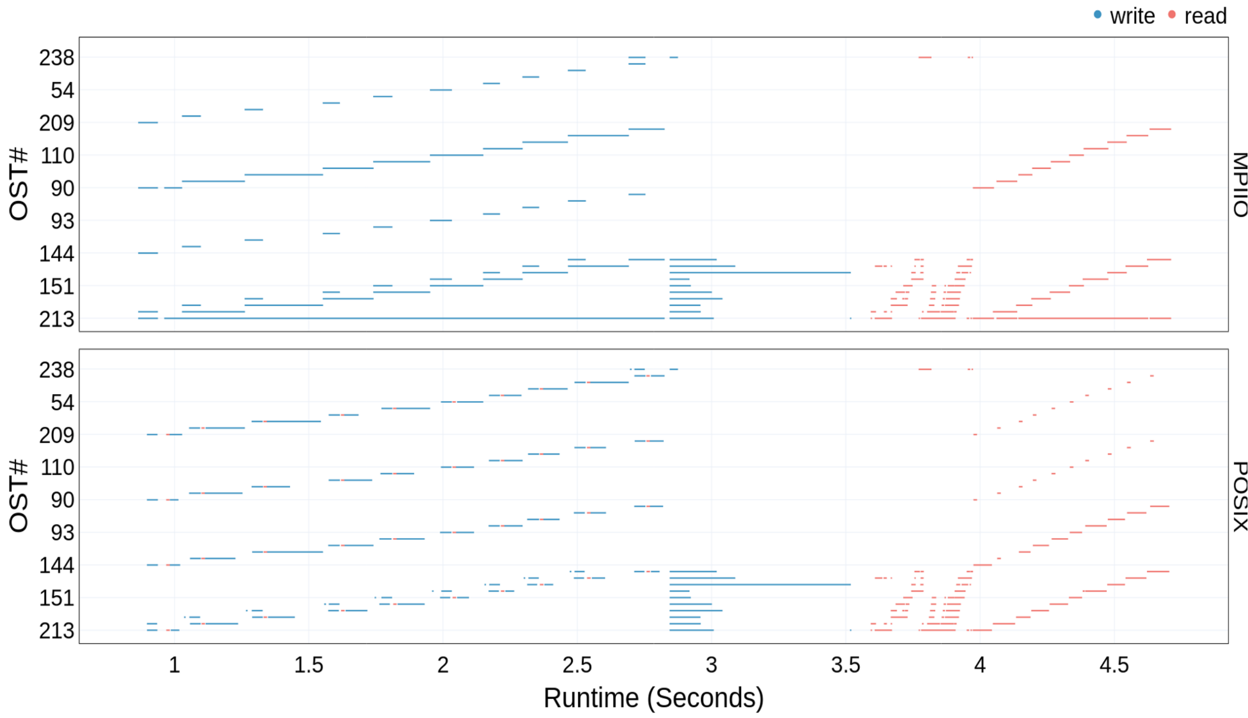


Visualize data transfers between I/O layers



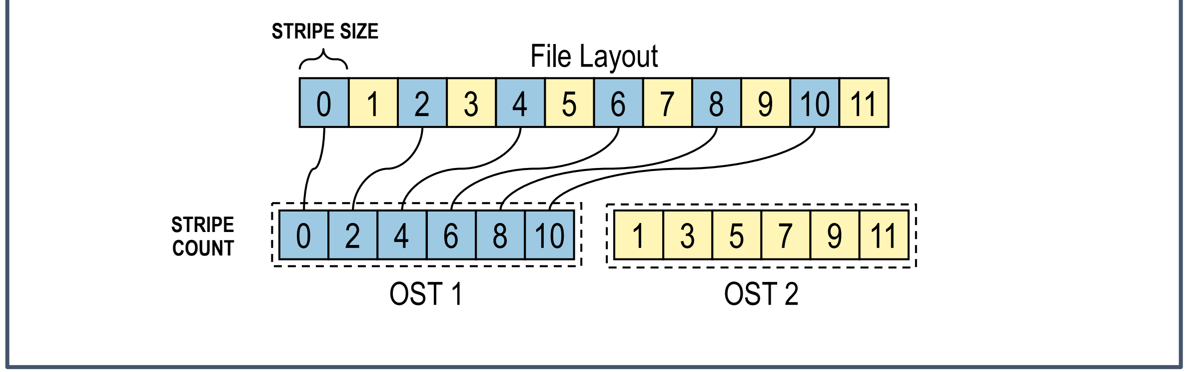
File system usage



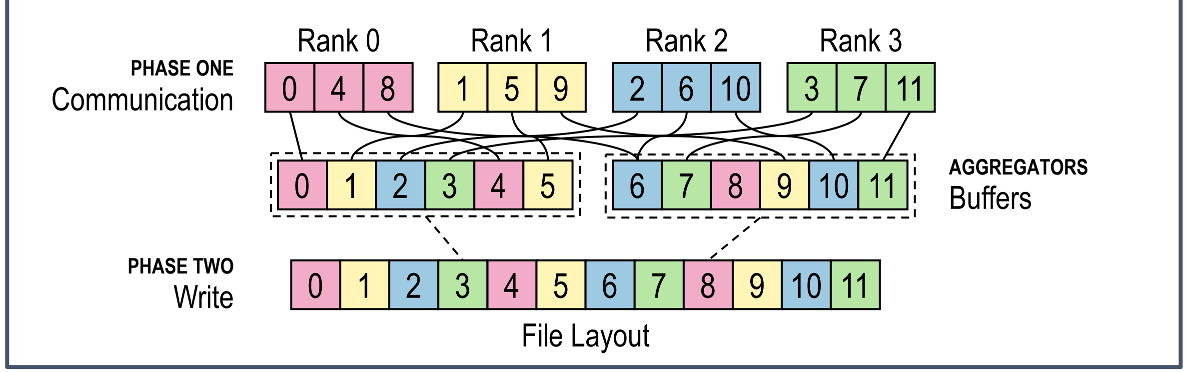


Common I/O optimization techniques

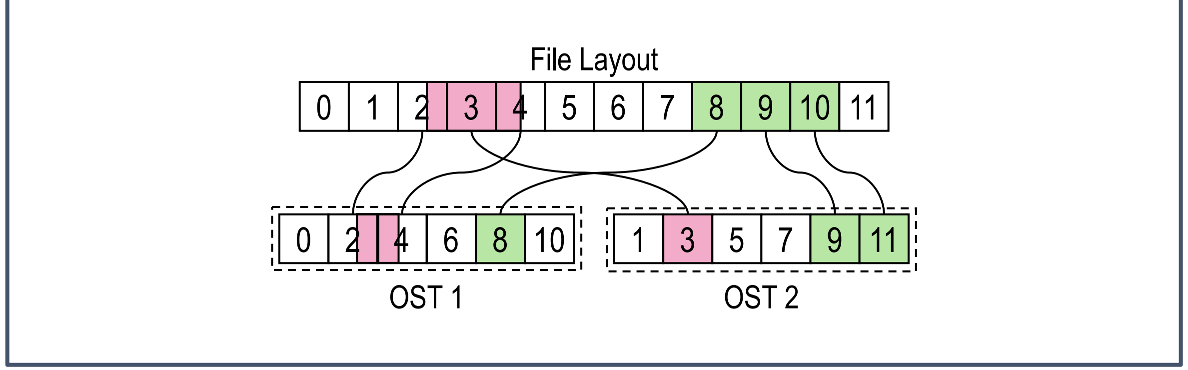
Lustre Striping



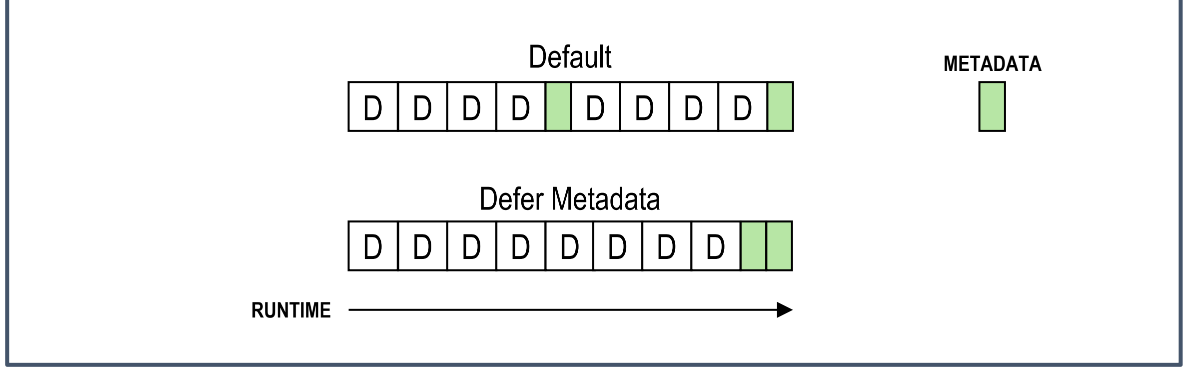
Collective I/O



HDF5 Alignment



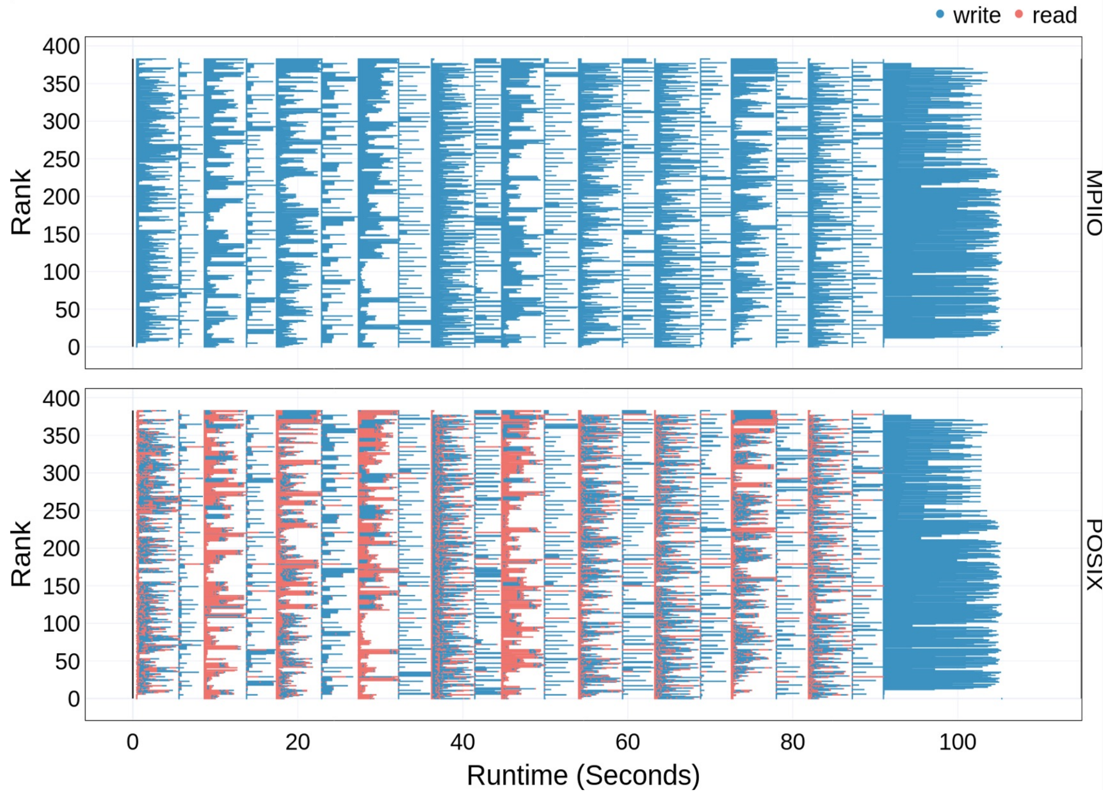
HDF5 Defer Metadata Flush



OpenPMD

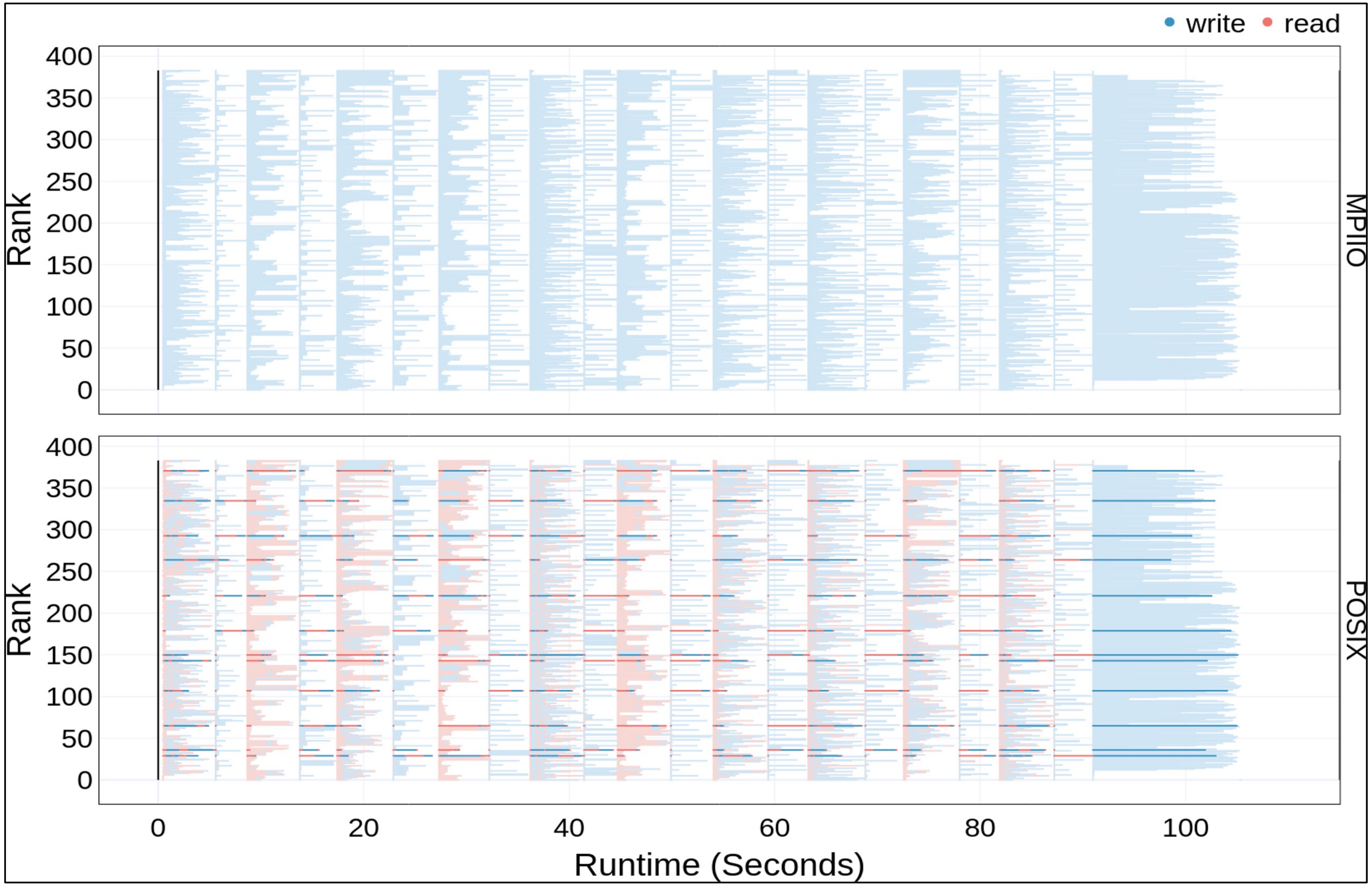
- Majority of the read and write requests are small
  - I/O calls are not using the MPI-IO's collective option

```
METADATA ────────────────────────────────────────────────────────────

▸ Application is write operation intensive (60.83% writes vs. 39.17% reads)
▸ Application is write size intensive (64.15% write vs. 35.85% read)
▸ Application issues a high number (100.00%) of misaligned file requests
   ↪ Recommendations:
  ↪ Consider aligning the requests to the file system block boundaries

OPERATIONS ──────────────────────────────────────────────────────────

▸ Application issues a high number (275840) of small read requests (i.e., < 1MB) which
represents 100.00% of all read/write requests
   ↪ 275840 (100.00%) small read requests are to "8a_parallel_3Db_0000001.h5"
   ↪ Recommendations:
  ↪ Consider buffering read operations into larger more contiguous ones
  ↪ Since the appplication already uses MPI-IO, consider using collective I/O calls (e.g.
MPI_File_read_all() or MPI_File_read_at_all()) to aggregate requests into larger ones
▸ Application issues a high number (427386) of small write requests (i.e., < 1MB) which
represents 99.75% of all read/write requests
   ↪ 275840 (64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"
   ↪ Recommendations:
  ↪ Consider buffering write operations into larger more contiguous ones
  ↪ Since the application already uses MPI-IO, consider using collective I/O calls (e.g.
MPI_File_write_all() or MPI_File_write_at_all()) to aggregate requests into larger ones
▸ Application mostly uses consecutive (97.67%) and sequential (2.16%) read requests
▸ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
▸ Detected read imbalance when accessing 1 individual files.
   ↪ Load imbalance of 55.23% detected while accessing "8a_parallel_3Db_0000001.h5"
   ↪ Recommendations:
  ↪ Consider better balancing the data transfer between the application ranks
  ↪ Consider tuning the stripe size and count to better distribute the data
  ↪ If the application uses netCDF and HDF5 double-check the need to set NO_FILL values
  ↪ If rank 0 is the only one opening the file, consider using MPI-IO collectives
▸ Application uses MPI-IO and write data using 7680 (92.50%) collective operations
▸ Application could benefit from non-blocking (asynchronous) reads
   ↪ Recommendations:
  ↪ Since you use HDF5, consider using the ASYNC I/O VOL connector
(https://github.com/hpc-io/vol-async)
  ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations
▸ Application could benefit from non-blocking (asynchronous) writes
   ↪ Recommendations:
  ↪ Since you use HDF5, consider using the ASYNC I/O VOL connector
(https://github.com/hpc-io/vol-async)
  ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations
```



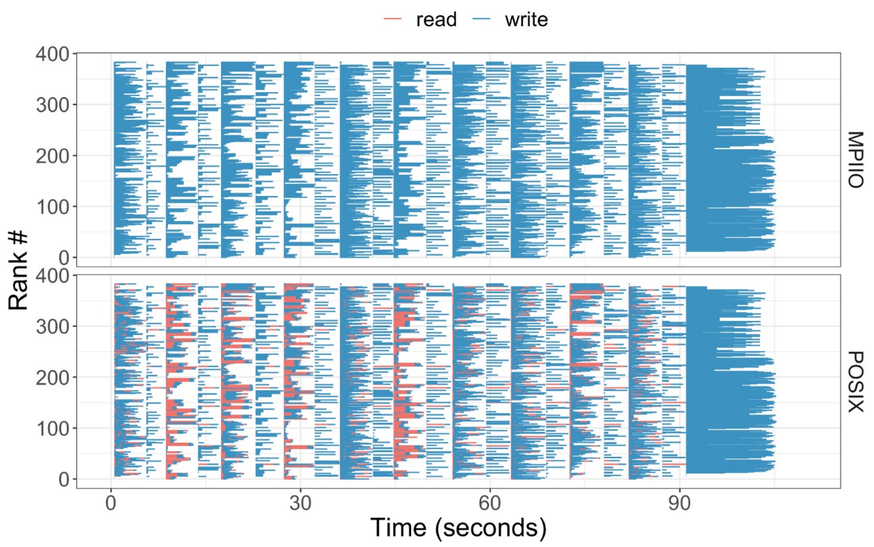
OpenPMD

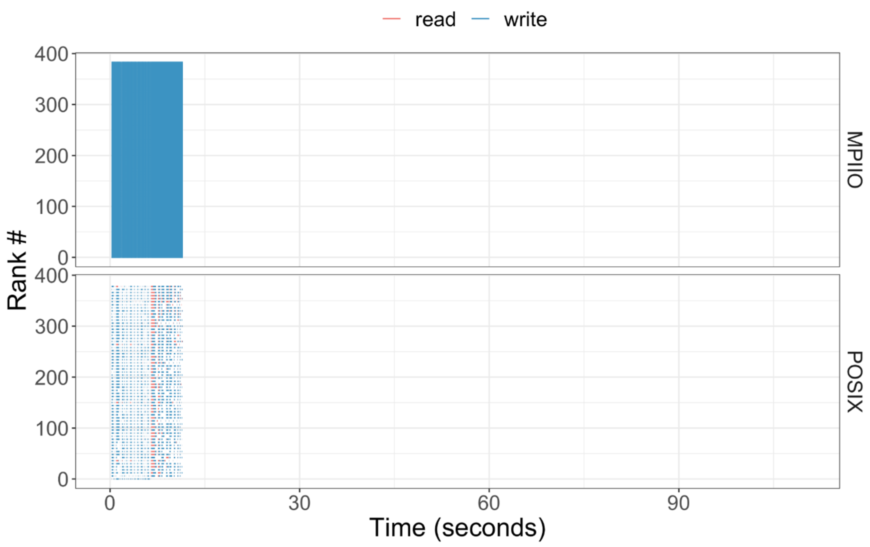
- Unbalanced data accesses among MPI ranks

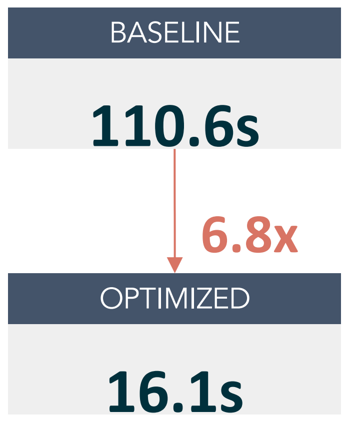


OpenPMD - Optimizations

- Collective HDF5 metadata were not actually collective due to an issue introduced in HDF5 1.10.5
  - Fixed that issue by using HDF5 1.10.4 and then enabling collective metadata I/O
- DXT Explorer 2.0 suggested larger buffer sizes
  - Used ROMIO hints to set the aggregators to **1 agg/node** and set the **cb_buffer_size** to 16 MB
  - Used GPFS **large block** I/O
- With HDF5 1.10.4 combined with other optimizations gives a total of 6.8x speedup from baseline

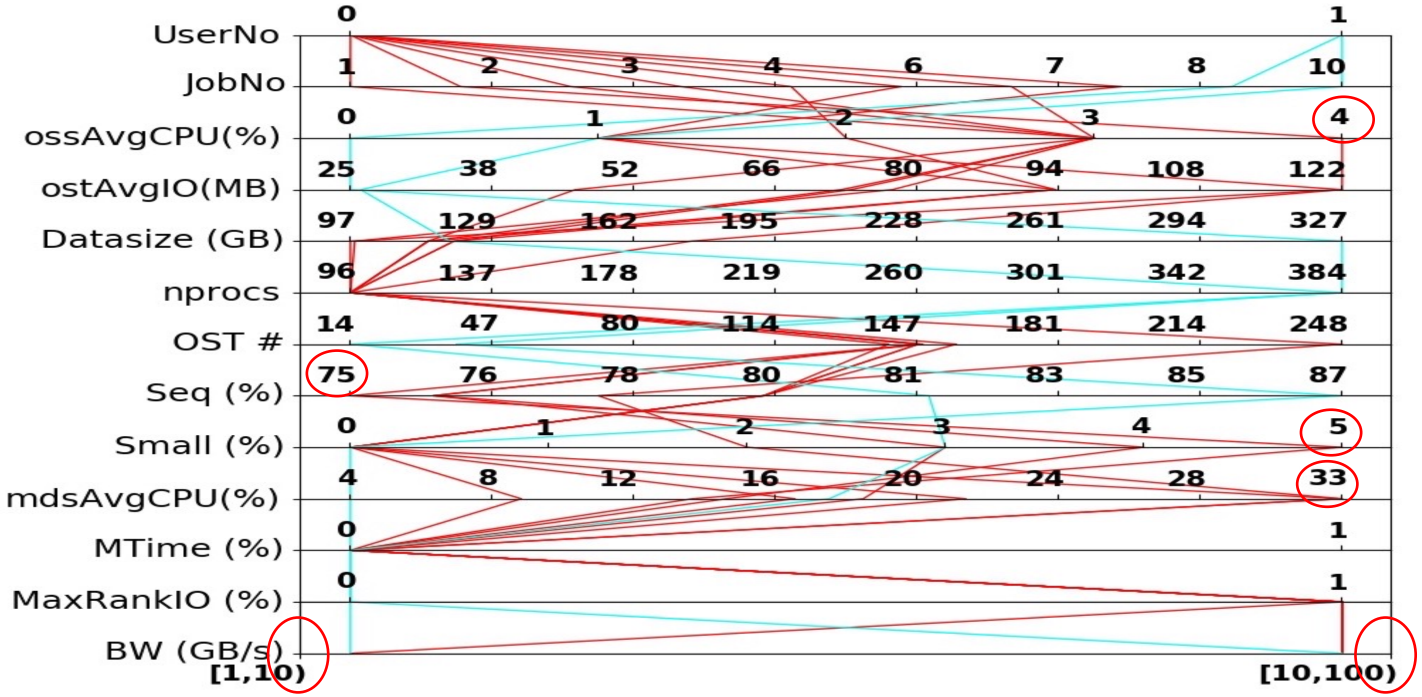






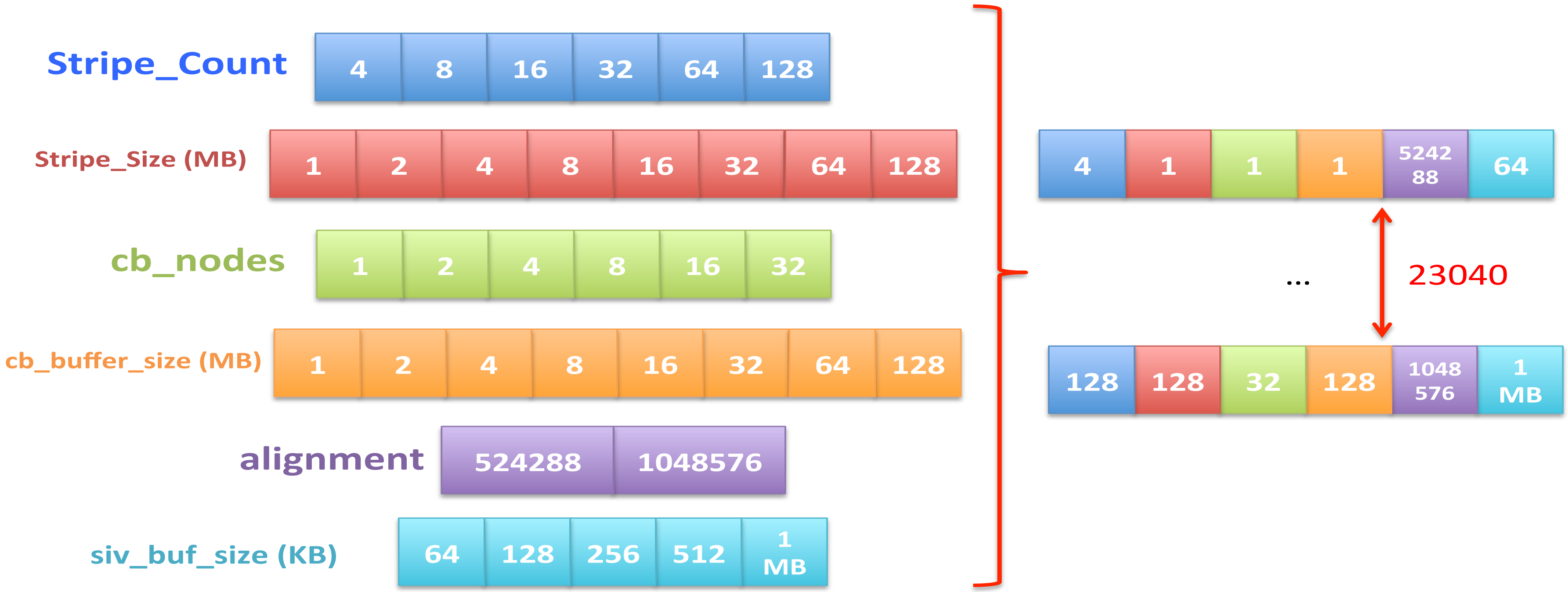
Application-Level Analysis of Cosmology1 IO

- Cosmology1's IO is well-formed: all jobs have high sequential IO (>75%), low small IO ratio (< 5%). Low metadata and storage server CPU utilization (<4% and <33%), etc.
- *However, IO bandwidth varies between [1,10) GB/s and [10,100)GB/s, which needs a job-level analysis.*



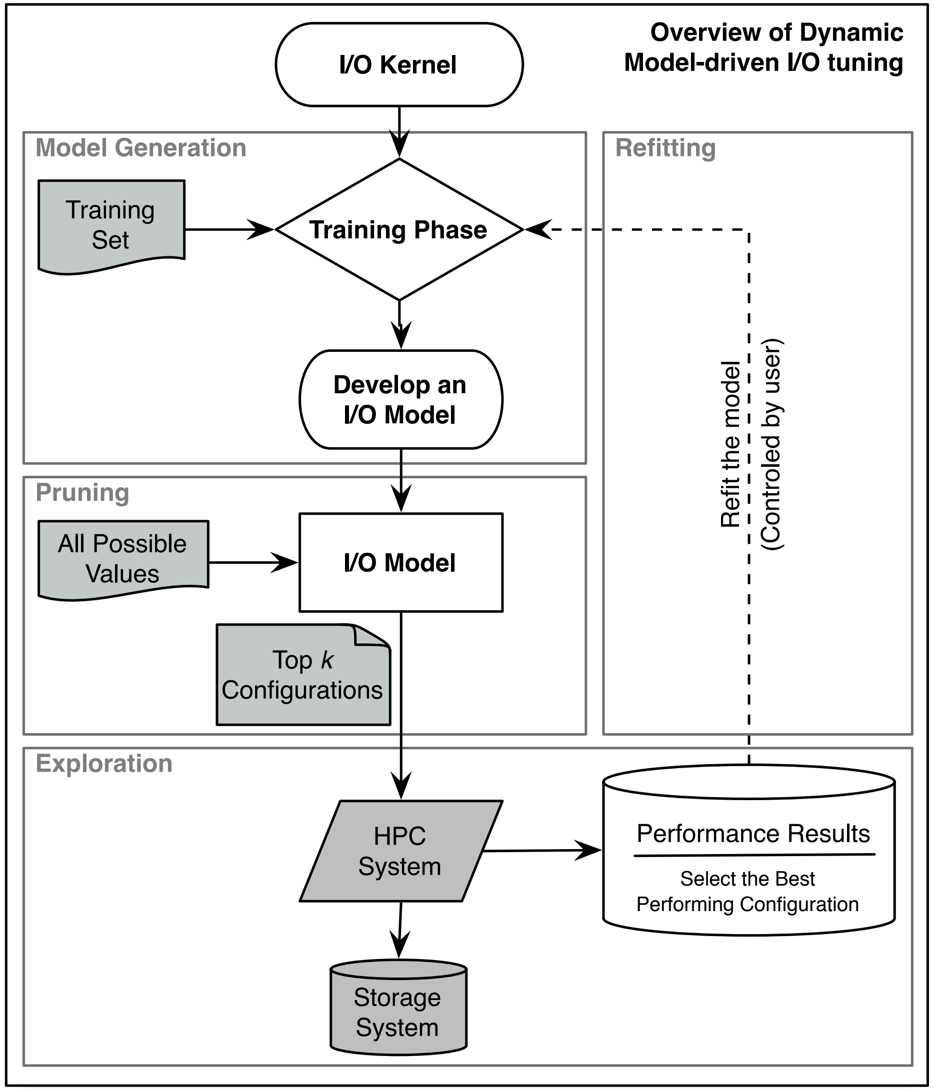
Tuning parameter space

The whole space visualized



Dynamic Model-driven Auto-tuning

- **Auto-tuning using empirical performance models of I/O**
- **Steps**
  - **Training phase to develop an I/O model**
  - **Pruning phase to select the top-K configurations**
  - **Exploration phase to select the best configuration**
  - **Refitting step to refine performance model**



Summary of today's class

- Today's class
  - Data life cycle
  - Data structures used in science data
  - Storage systems
  - Parallelism and parallel I/O
  - High-level parallel I/O libraries
  - Factors that impact the parallel I/O performance
  - Tuning parallel I/O configurations to optimize performance

- Next class: Class presentations