CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 12: Tools for understanding parallel I/O performance – DXT Explorer and Drishti

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

02/16/2023

Today's class

- Any questions?

- Class presentation topic

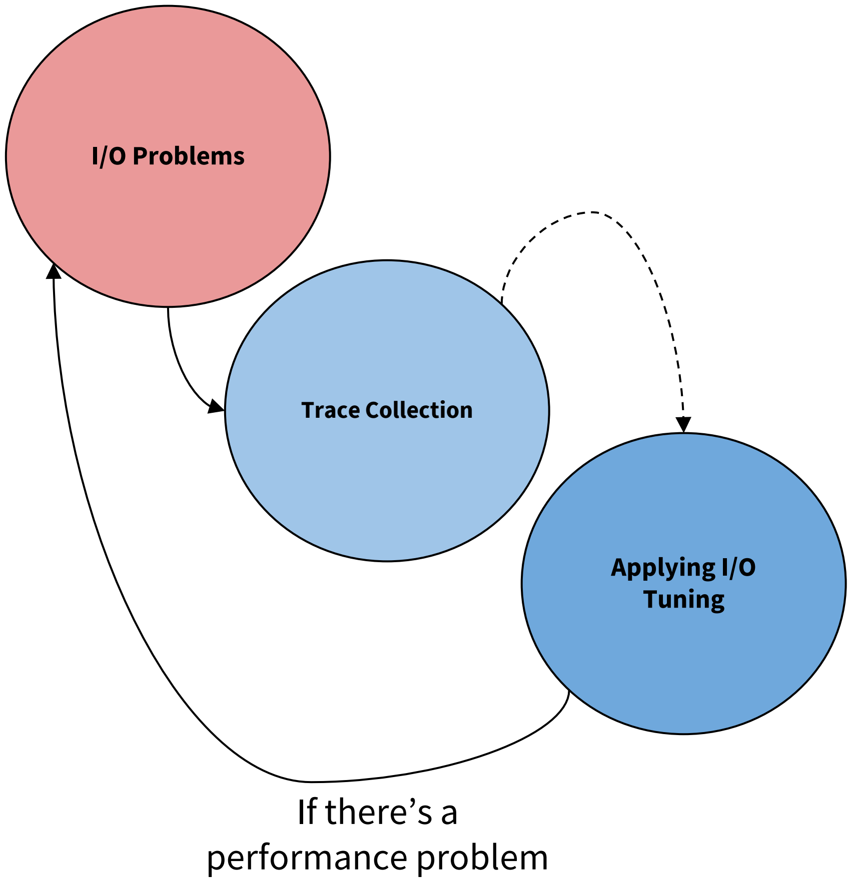
- Today's class –
  - Tools for understanding parallel I/O performance – DXT Explorer and Drishti

Tools for understanding parallel I/O performance

- Darshan (ANL)
- Darshan Extended Trace (DXT) -- Intel, LBNL, & ANL
- DXT Explorer -- LBNL
- Drishti -- LBNL

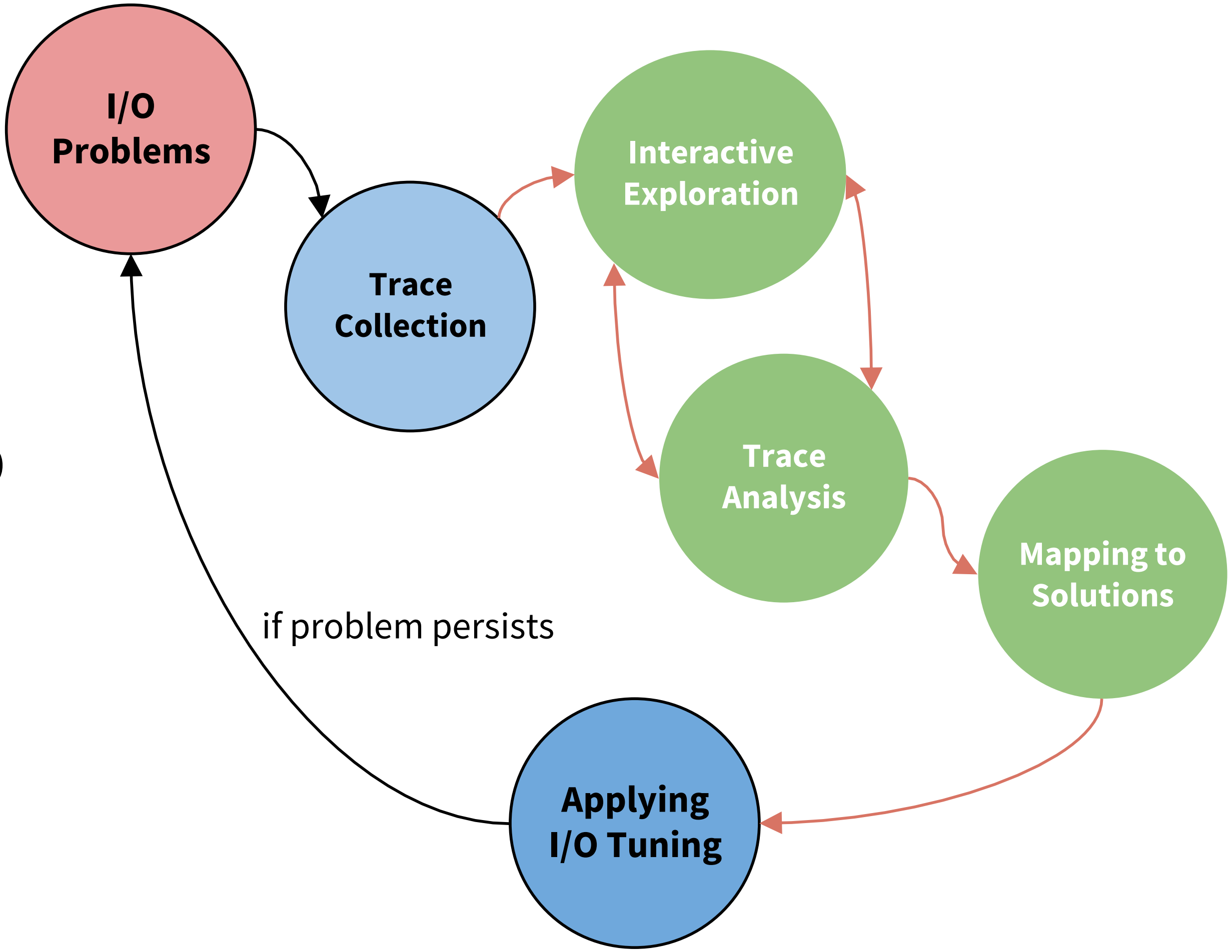
Path to understand I/O performance and optimize

- There are several tools available to trace I/O performance
  - Darshan
  - Recorder
- Gap between the trace collection, analysis, and tuning
- A solution to close this gap requires
  - Analyzing the collected metrics and traces
  - Automatically diagnosing the root cause of poor performance
  - Providing user recommendations

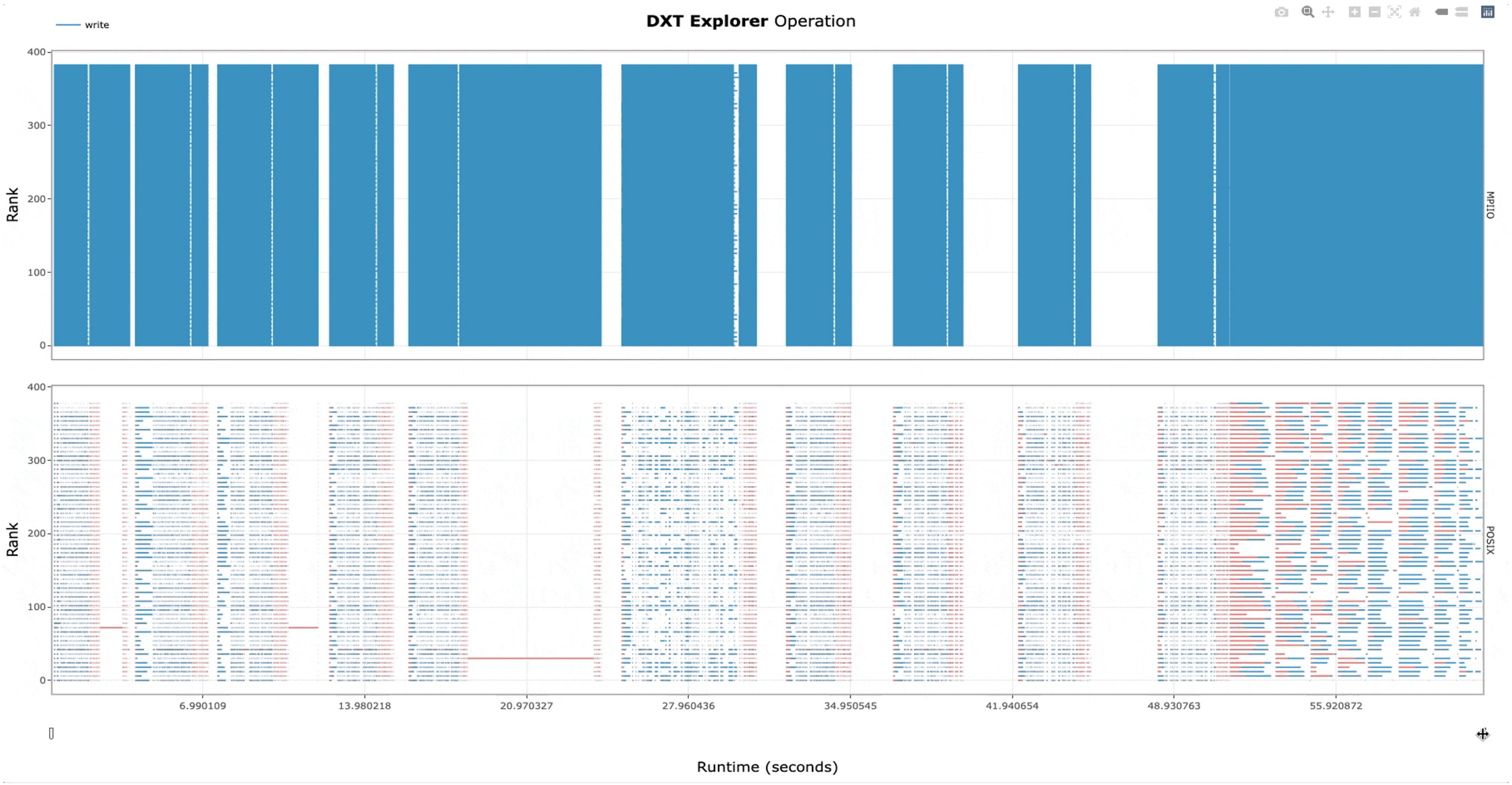


DXT Explorer

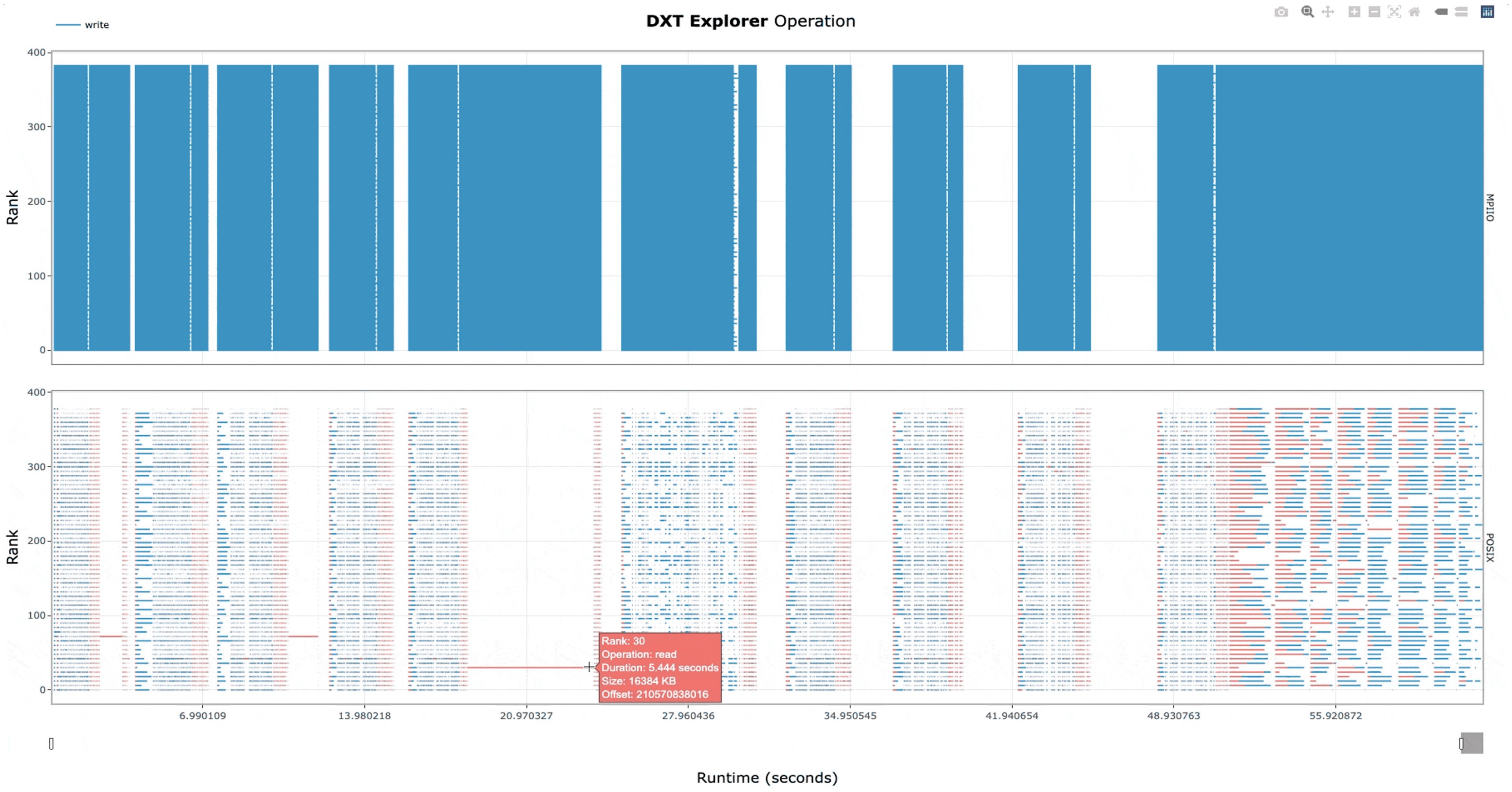
- **DXT Explorer**
  - Analyze the I/O traces interactively
  - Diagnose and highlight the bottlenecks
  - Provides an actionable set of recommendations
- Provides an interactive component to I/O traces
  - Users can visually inspect the I/O behavior
    - Zoom in areas of interest
  - End users provided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



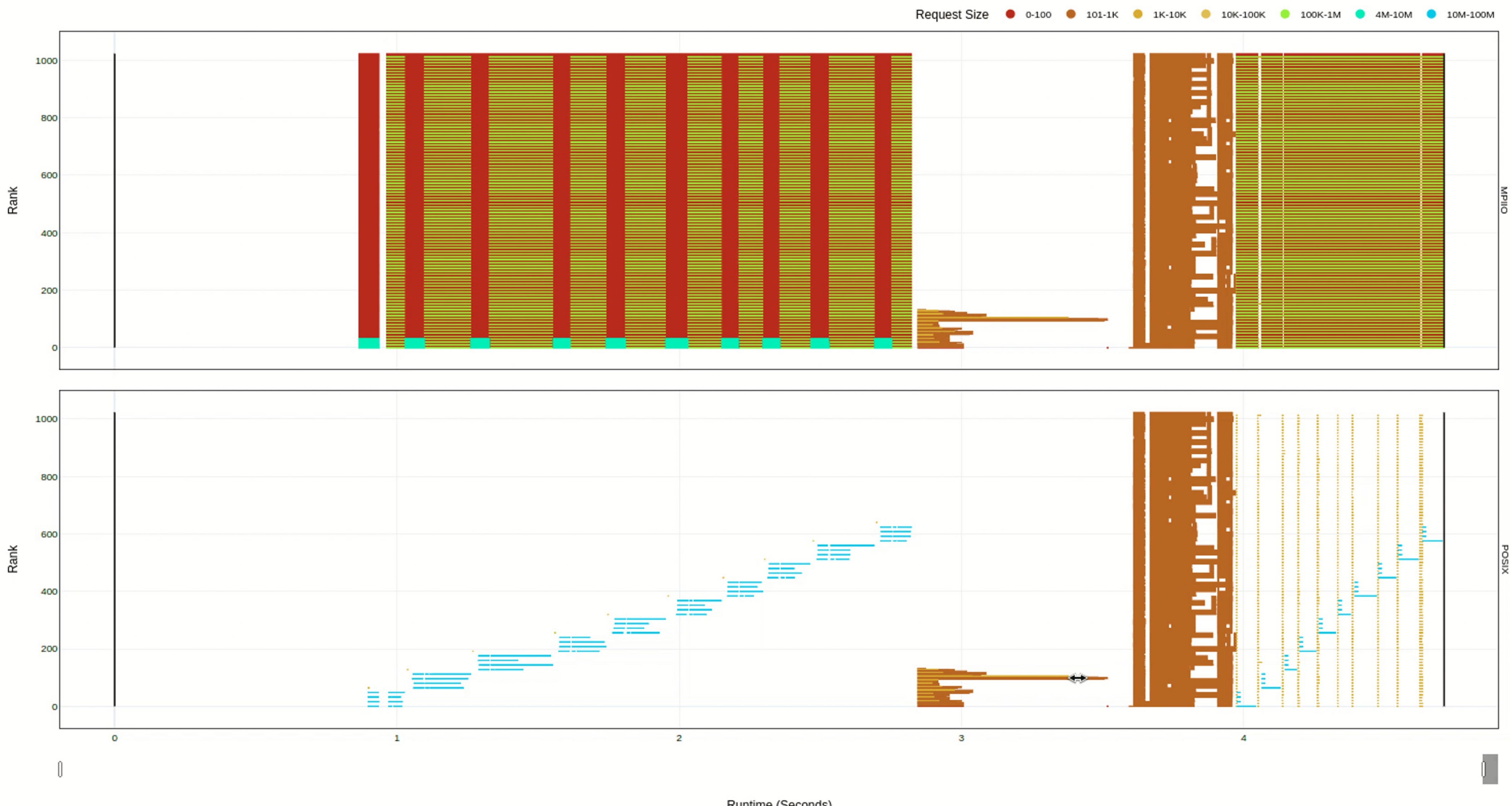
Focus on what matters!



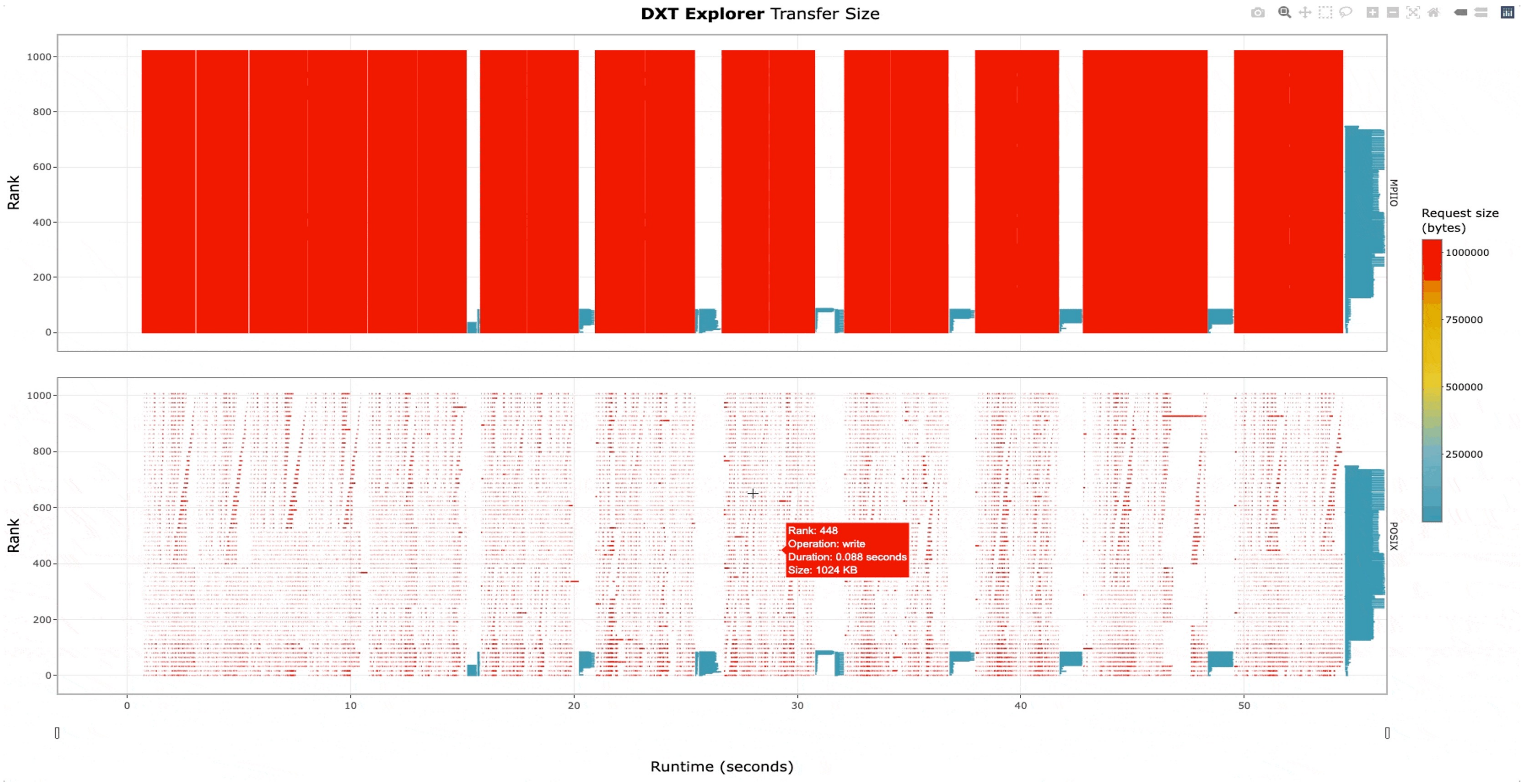
Visualize data transfers between I/O layers



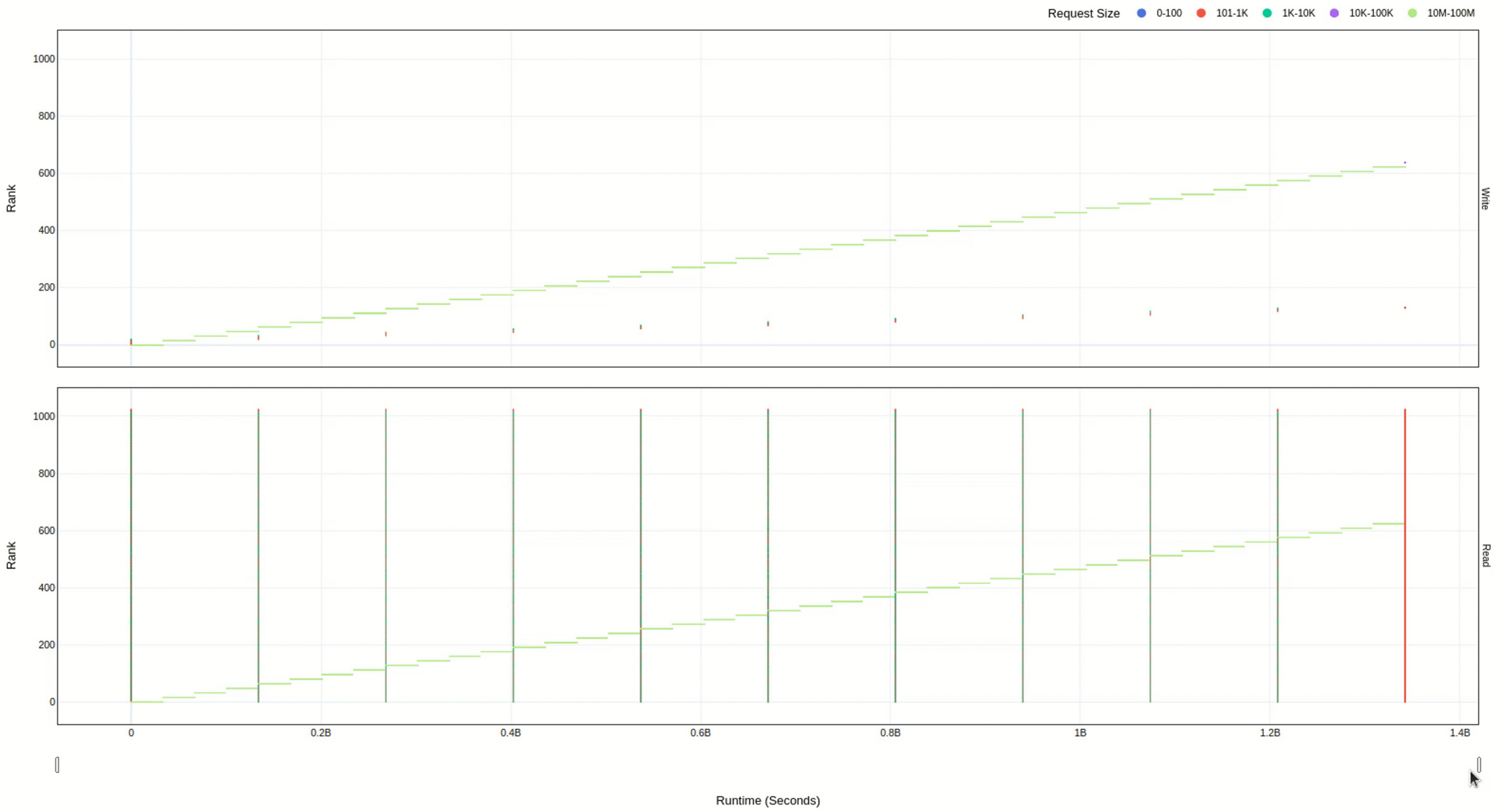
Data size transfers – more views



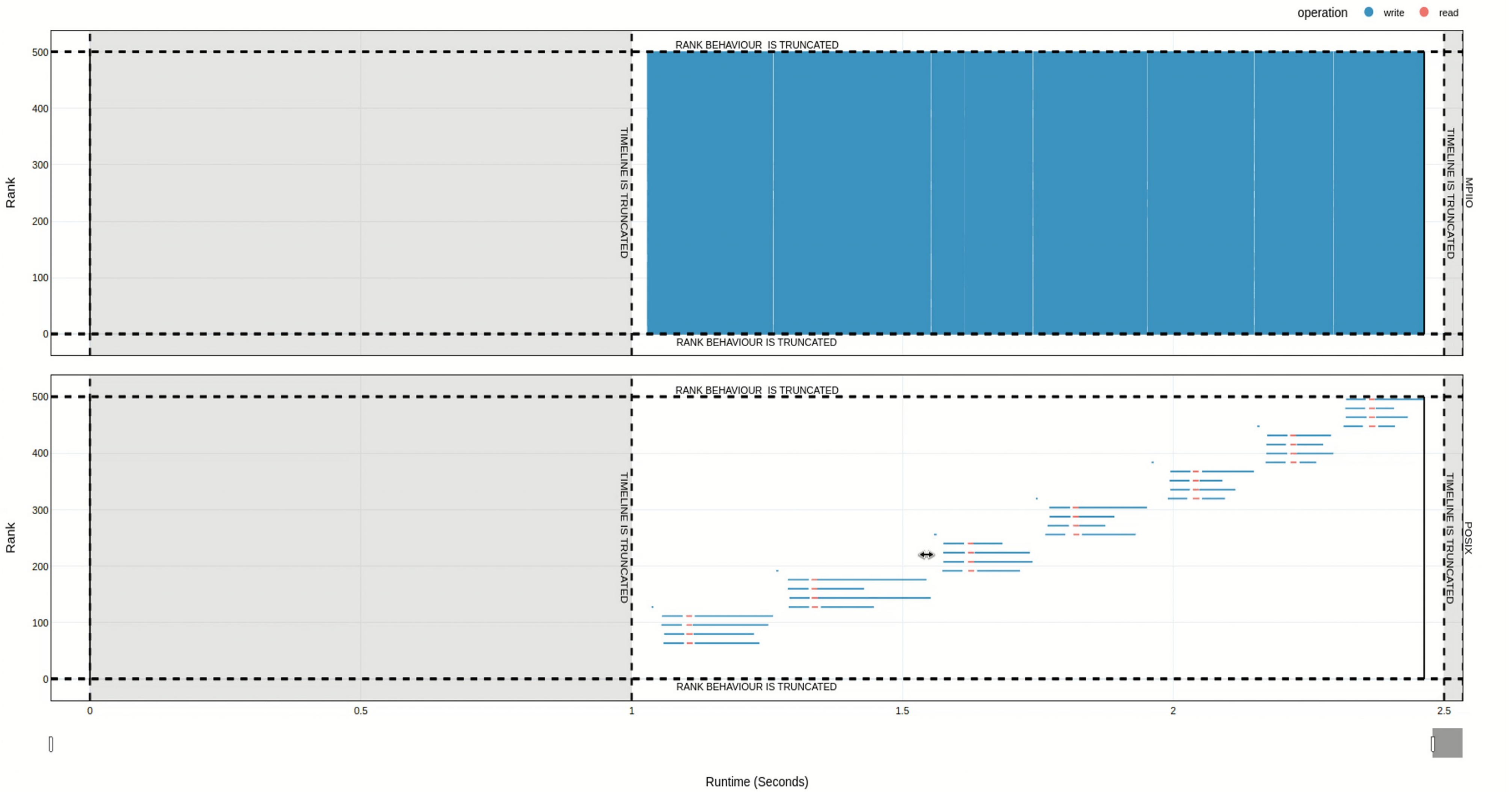
I/O size impacts performance



Spatiality of I/O calls



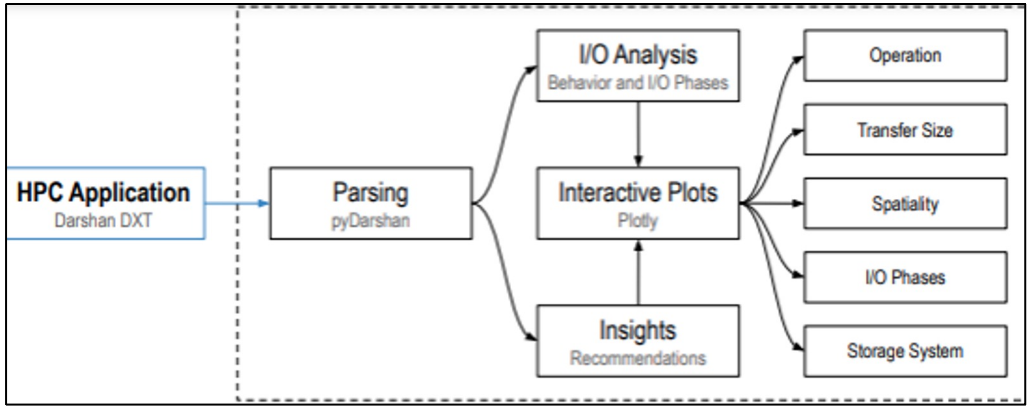
Examining selective location of plots



Explore the timeline by **zooming in and out** and observing how the **MPI-IO** calls are translated to the **POSIX** layer. Visualize relevant information in the context of each I/O call (rank, operation, duration, request size, and OSTs if Lustre) by hovering over a given operation.

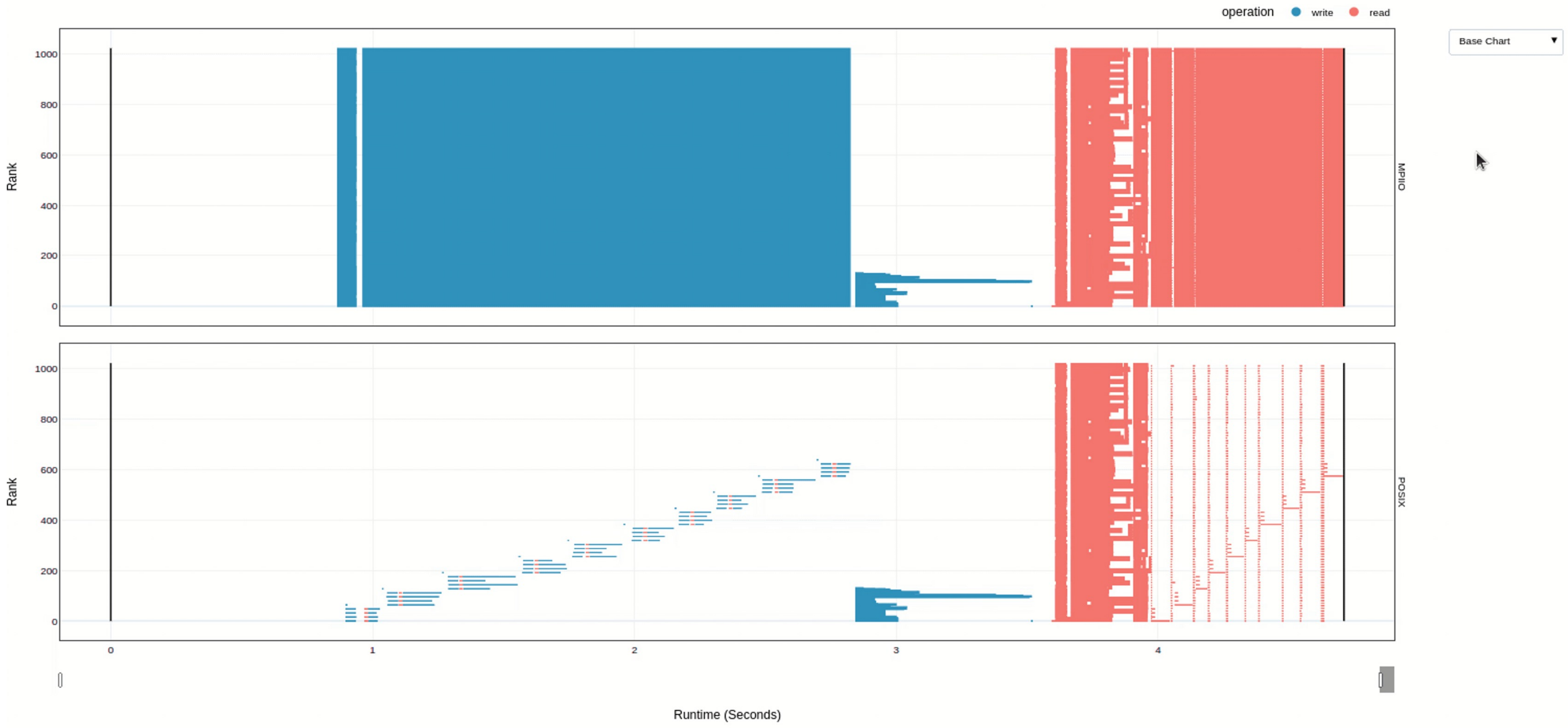
DXT Explorer v2.0

- New features we seek in DXT Explorer 2.0:
  - Provide an extensible framework so new visualizations and analysis could be easily integrated
  - Identify and highlight common root causes of I/O performance problems
  - Provide a set of actionable items or recommendations based on the detected I/O bottlenecks
  - Understand how the file system is accessed by the ranks involved in I/O operations
  - Detect and characterize the distinct I/O phases of an application throughout its execution
  - Include support for other tracing applications such as Recorder



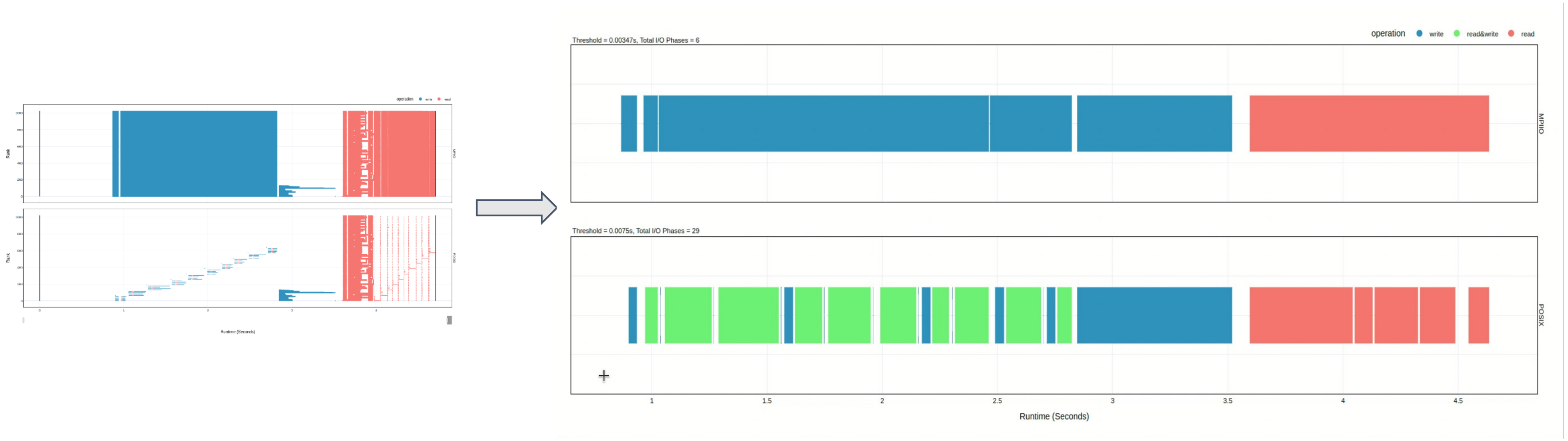
BERKELEY LAB

Multi Layered Plots (contd …)

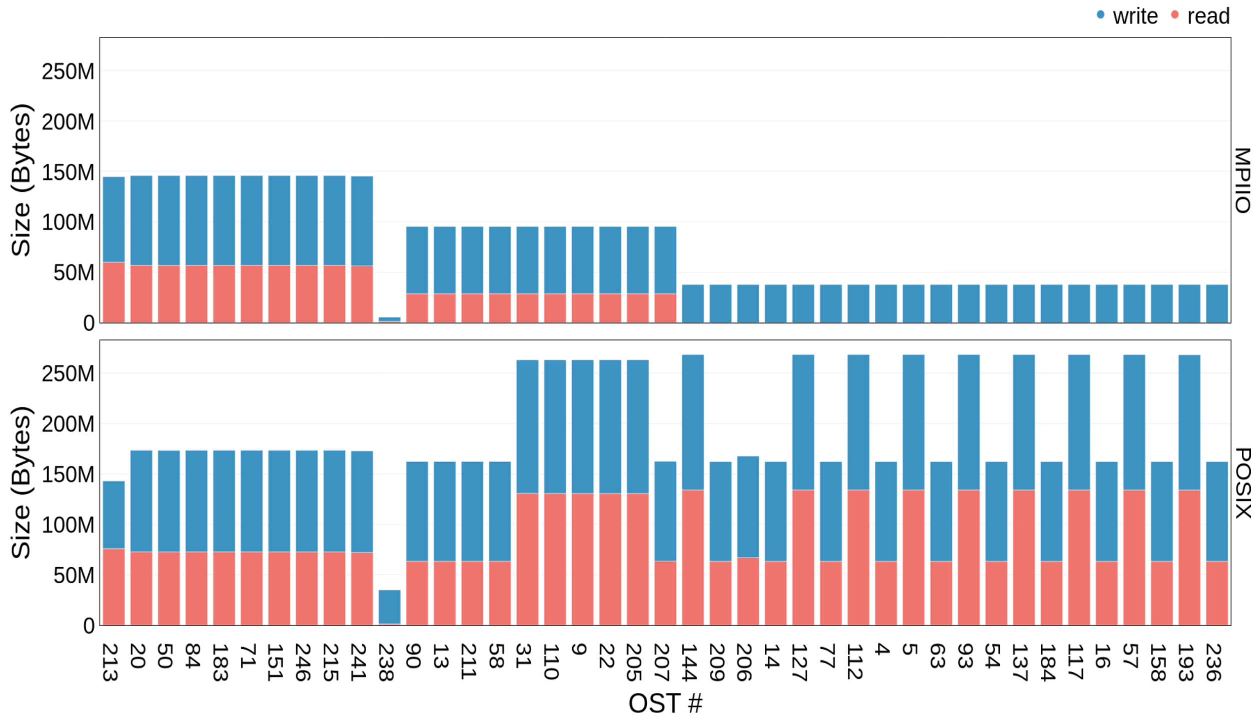


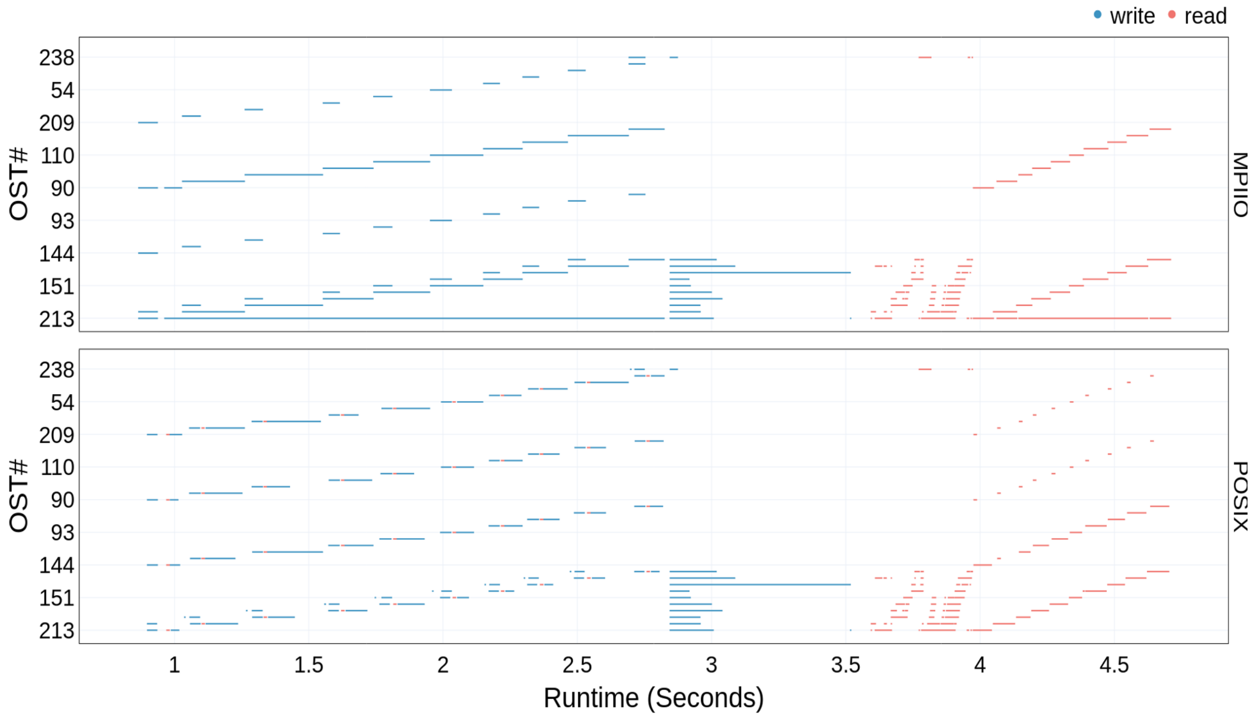
I/O Phases

- An I/O phase is continuous amount of time where an application is performing I/O
- Factors outside an application's scope could cause an I/O phase to take longer
  - Network Interference
  - Storage system congestion



File system usage





Drishti - Guiding end-users in the I/O optimization journey

Level	Interface	Detected Behavior
HIGH	STDIO	High STDIO usage* (> 10% of total transfer size uses STDIO)
OK	POSIX	High number* of sequential read operations (≥ 80%)
OK	POSIX	High number* of sequential write operations (≥ 80%)
INFO	POSIX	Write operation count intensive* (> 10% more writes than reads)
INFO	POSIX	Read operation count intensive* (> 10% more reads than writes)
INFO	POSIX	Write size intensive* (> 10% more bytes written then read)
INFO	POSIX	Read size intensive* (> 10% more bytes read then written)
WARN	POSIX	Redundant reads
WARN	POSIX	Redundant writes
HIGH	POSIX	High number* of small† reads (> 10% of total reads)
HIGH	POSIX	High number* of small† writes (> 10% of total writes)
HIGH	POSIX	High number* of misaligned memory requests (> 10%)
HIGH	POSIX	High number* of misaligned file requests (> 10%)
HIGH	POSIX	High number* of random read requests (> 20%)
HIGH	POSIX	High number* of random write requests (> 20%)
HIGH	POSIX	High number* of small† reads to shared-files (> 10% of reads)
HIGH	POSIX	High number* of small† writes to shared-files (> 10% of writes)
HIGH	POSIX	High metadata time* (one or more ranks spend > 30 seconds)
HIGH	POSIX	Rank 0 heavy workload
HIGH	POSIX	Data transfer imbalance between ranks (> 15% difference)
HIGH	POSIX	Stragglers detected among the MPI ranks
HIGH	POSIX	Time imbalance* between ranks (> 15% difference)
WARN	MPI-IO	No MPI-IO calls detected from Darshan logs
HIGH	MPI-IO	Detected MPI-IO but no collective read operation
HIGH	MPI-IO	Detected MPI-IO but no collective write operation
WARN	MPI-IO	Detected MPI-IO but no non-blocking read operations
WARN	MPI-IO	Detected MPI-IO but no non-blocking write operations
OK	MPI-IO	Detected MPI-IO and collective read operations
OK	MPI-IO	Detected MPI-IO and collective write operations
HIGH	MPI-IO	Detected MPI-IO and inter-node aggregators
WARN	MPI-IO	Detected MPI-IO and intra-node aggregators
OK	MPI-IO	Detected MPI-IO and one aggregator per node

* Trigger has a threshold that could be further tunned. Default value in parameters.
† Small requests are consider to be < 1MB.

```
Drishti

DRISHTI v.0.3

JOB              1190243
EXECUTABLE       bin/8_benchmark_parallel
DARSHAN          jlbez_8_benchmark_parallel_id1190243_7-23-45631-11755726114084236527_1.darshan
EXECUTION DATE   2021-07-23 16:40:31+00:00 to 2021-07-23 16:40:32+00:00 (0.00 hours)
FILES            6 files (1 use STDIO, 2 use POSIX, 1 use MPI-IO)
PROCESSES        64
HINTS            romio_no_indep_rw=true cb_nodes=4

1 critical issues, 5 warnings, and 5 recommendations

METADATA

▶ Application is read operation intensive (6.34% writes vs. 93.66% reads)
▶ Application might have redundant read traffic (more data read than the highest offset)
▶ Application might have redundant write traffic (more data written than the highest offset)

OPERATIONS

▶ Application issues a high number (285) of small read requests (i.e., < 1MB) which represents
37.11% of all read/write requests
  ↪ 284 (36.98%) small read requests are to "benchmark.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 mostly uses consecutive (2.73%) and sequential (90.62%) read requests
▶ Application mostly uses consecutive (19.23%) and sequential (76.92%) write requests
▶ Application uses MPI-IO and read data using 640 (83.55%) collective operations
▶ Application uses MPI-IO and write data using 768 (100.00%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
  ↪ Recommendations:
    ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations (e.g.,
MPI_File_iread(), MPI_File_read_all_begin/end(), or MPI_File_read_at_all_begin/end())
▶ Application could benefit from non-blocking (asynchronous) writes
  ↪ Recommendations:
    ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations (e.g.,
MPI_File_iwrite(), MPI_File_write_all_begin/end(), or MPI_File_write_at_all_begin/end())
▶ Application is using inter-node aggregators (which require network communication)
  ↪ Recommendations:
    ↪ Set the MPI hints for the number of aggregators as one per compute node (e.g.,
cb_nodes=32)

2022 | LBL | Drishti report generated at 2022-08-05 14:27:16.422368 in 0.973 seconds
```

Common I/O optimization techniques

Lustre Striping

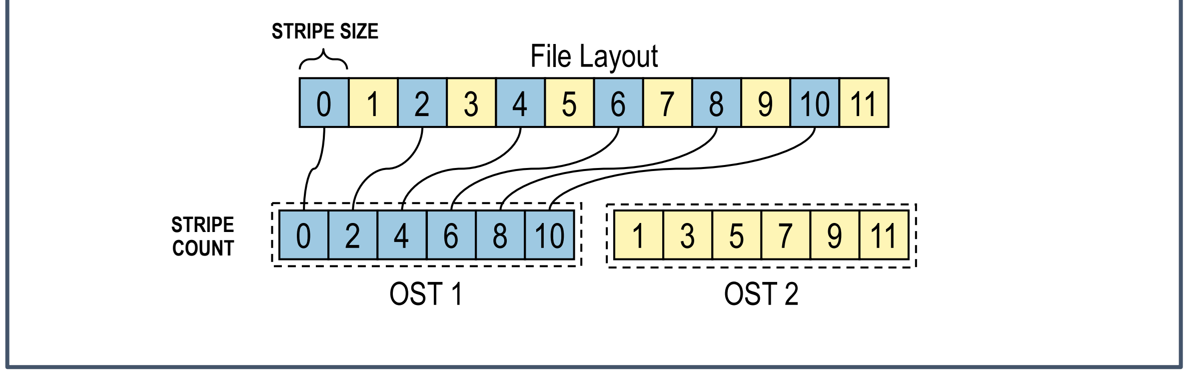
Collective I/O

HDF5 Alignment

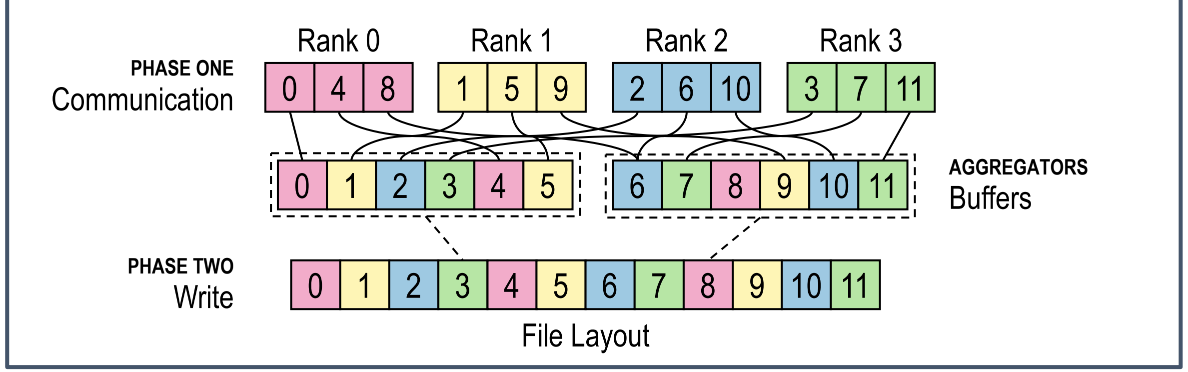
HDF5 Defer Metadata Flush

Common I/O optimization techniques

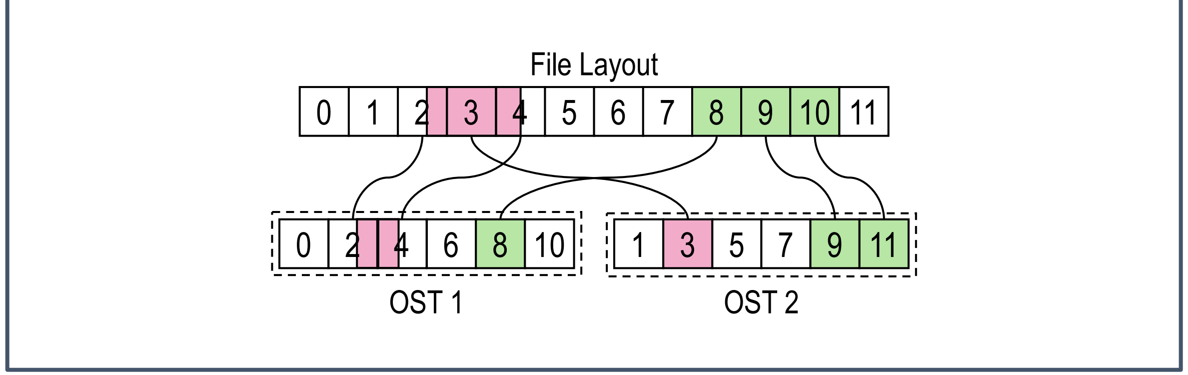
Lustre Striping



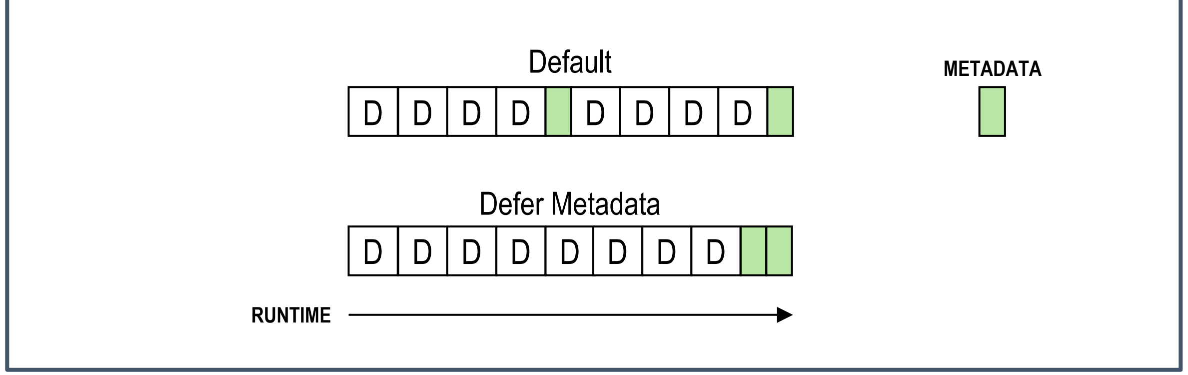
Collective I/O



HDF5 Alignment



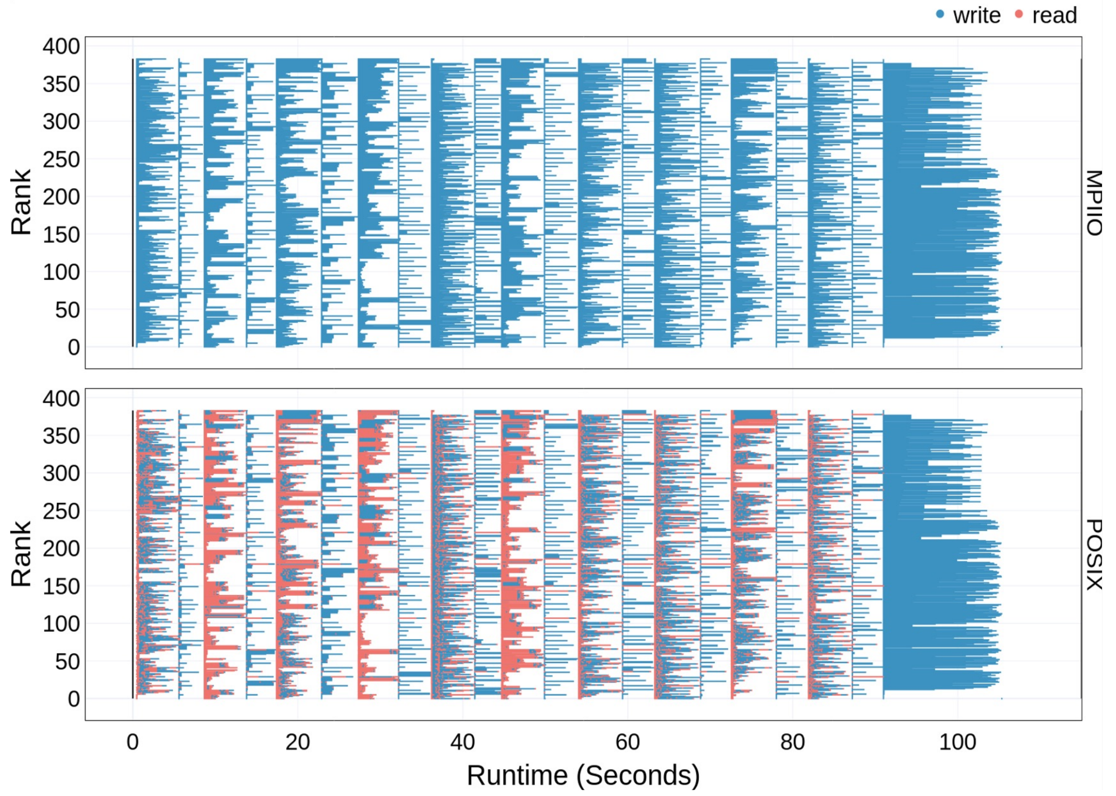
HDF5 Defer Metadata Flush



OpenPMD

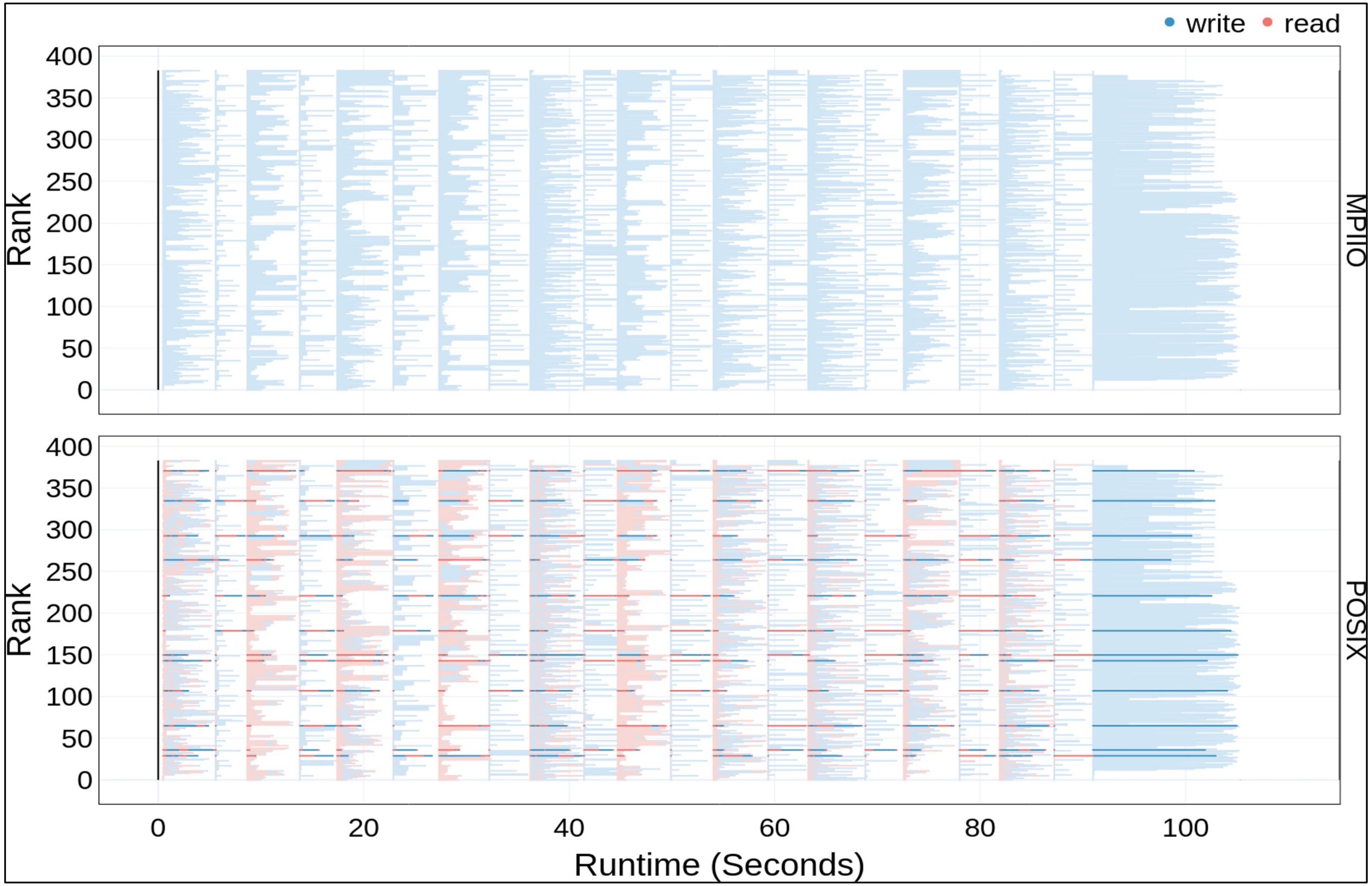
- Majority of the read and write requests are small
  - I/O calls are not using the MPI-IO's collective option

```
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
```



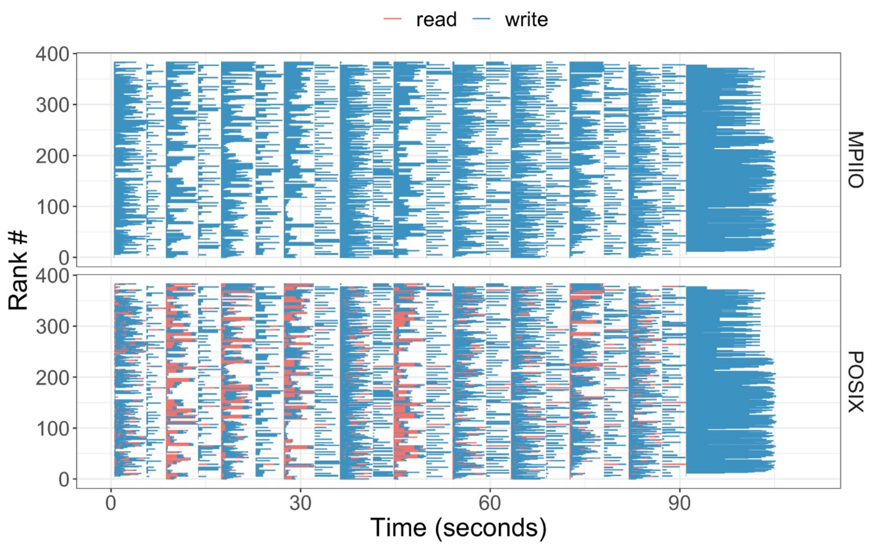
OpenPMD

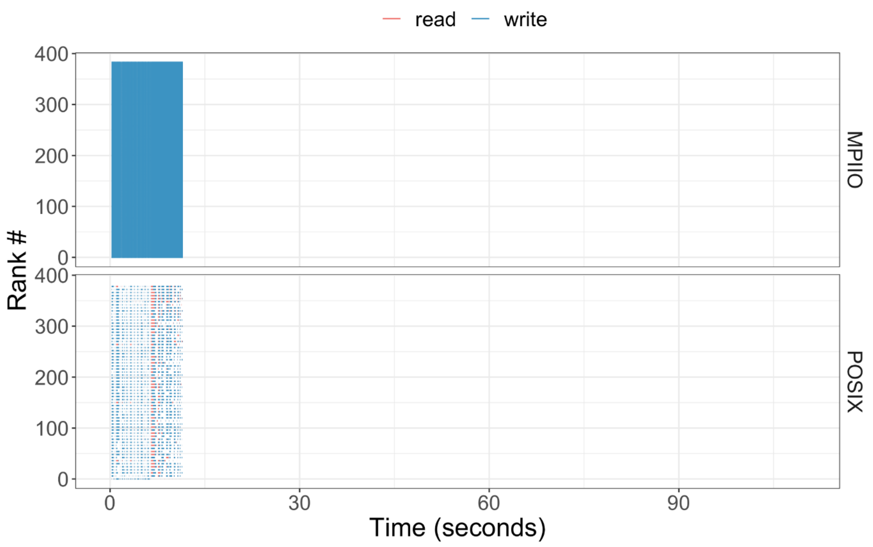
- Unbalanced data accesses among MPI ranks

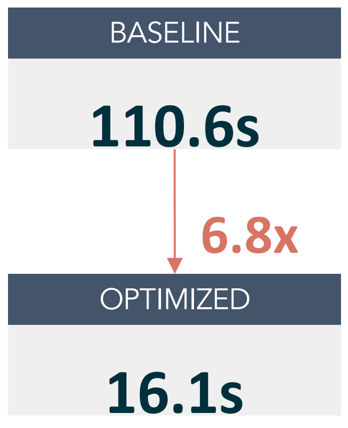


OpenPMD - Optimizations

- 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

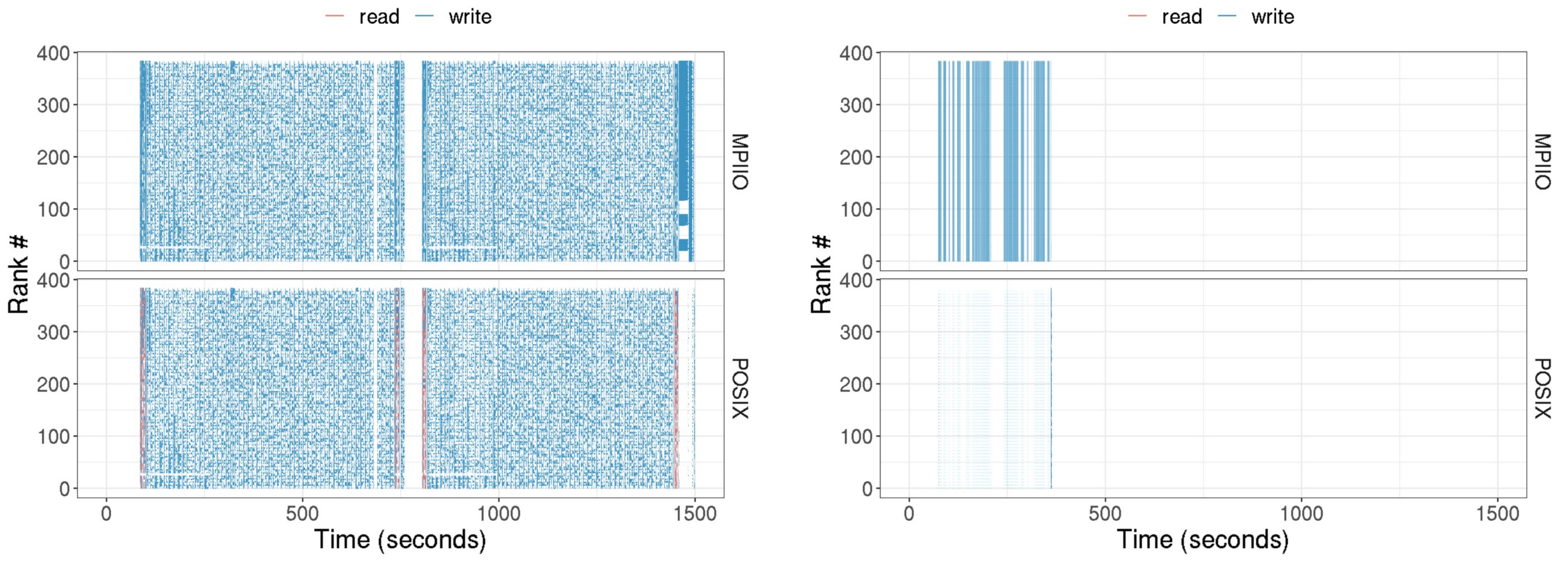






FLASH
HDF5 tuning

- 2 checkpoint files (≈2.3TB each) and 2 plot file (≈14GB each)
- FLASH was not using collective MPI-IO calls
- **Optimizations**: collective I/O, HDF5 alignment, and defer metadata flush



BASELINE

1495s

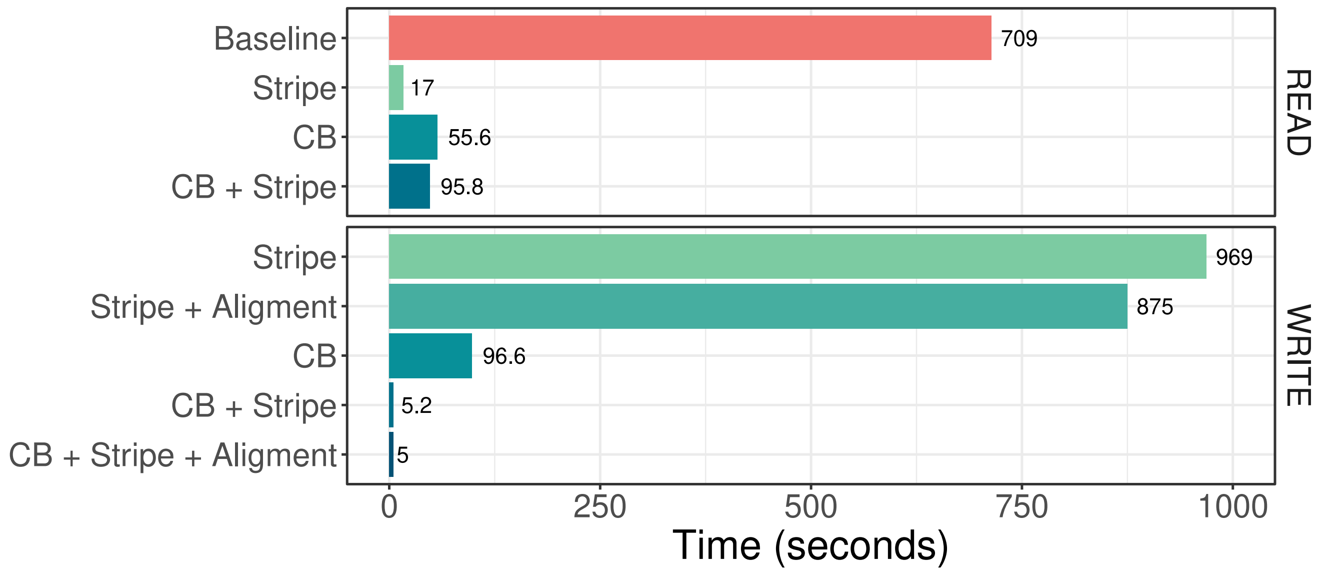
4.1x

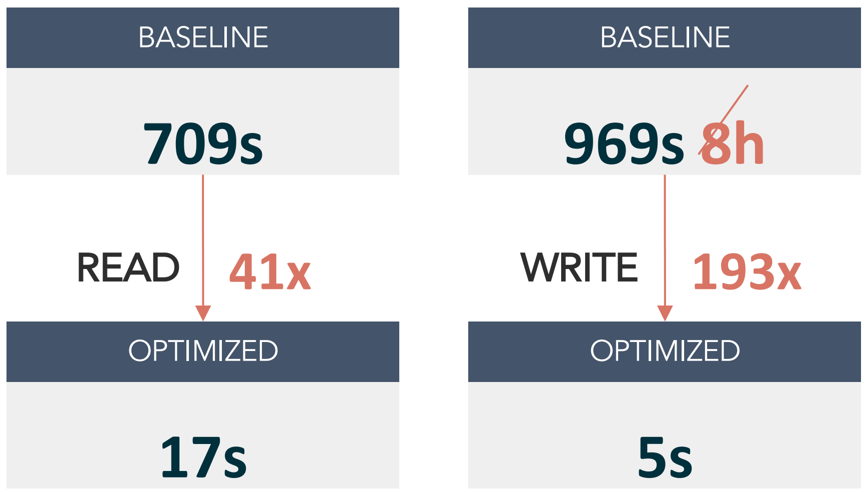
OPTIMIZED

361s

Block-cyclic
Combining multiple techniques

- **Cori** with 32 nodes ⨉ 32 ranks per node = 1024 MPI ranks
  - Square matrix with 81250 ⨉ 81250 with FP64 data, total of ≈50GB
  - Block-cyclic data with 128 ⨉ 128 with 1024 processes in a 32 ⨉ 32 process grid
- Lustre striping, MPI-IO collective buffering, and HDF5 alignment **optimizations**





Summary of today's class

- DXT Explorer: A visualization tool for Darshan Extended Traces

- Drishti: A tool for showing performance optimizations based on Darshan logs