Enabling Agile Analysis of I/O Performance Data with PyDarshan

Jakob Luettgau
luettgauj@acm.org
Inria
Rennes, France

Shane Snyder
ssnyder@mcs.anl.gov
Argonne National Laboratory
Lemont, IL, USA

Tyler Reddy
treddy@lanl.gov
Los Alamos National Laboratory
Los Alamos, NM
USA

Nikolaus Awtrey
Los Alamos National Laboratory
Los Alamos, NM
USA

Kevin Harms
harms@alcf.anl.gov
Argonne National Laboratory
Lemont, IL, USA

Jean Luca Bez
jlbez@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, CA, USA

Rui Wang
rwang@anl.gov
Argonne National Laboratory
Lemont, IL, USA

Rob Latham
robl@mcs.anl.gov
Argonne National Laboratory
Lemont, IL, USA

Philip Carns
carns@mcs.anl.gov
Argonne National Laboratory
Lemont, IL, USA

ABSTRACT

Modern scientific applications utilize numerous software and hardware layers to efficiently access data. This approach poses a challenge for I/O optimization because of the need to instrument and correlate information across those layers. The Darshan characterization tool seeks to address this challenge by providing efficient, transparent, and compact runtime instrumentation of many com-

and rate of data produced by scientific instruments and simulations [9, 20, 40]. At the same time, new workloads are emerging that stress I/O systems differently from traditional applications [12, 22]. The variety of workloads present on modern platforms include simulations, experimental data analysis, inference and training of machine learning, and hybrid workloads incorporating elements of each. Many applications run into I/O bottlenecks that require

Overview

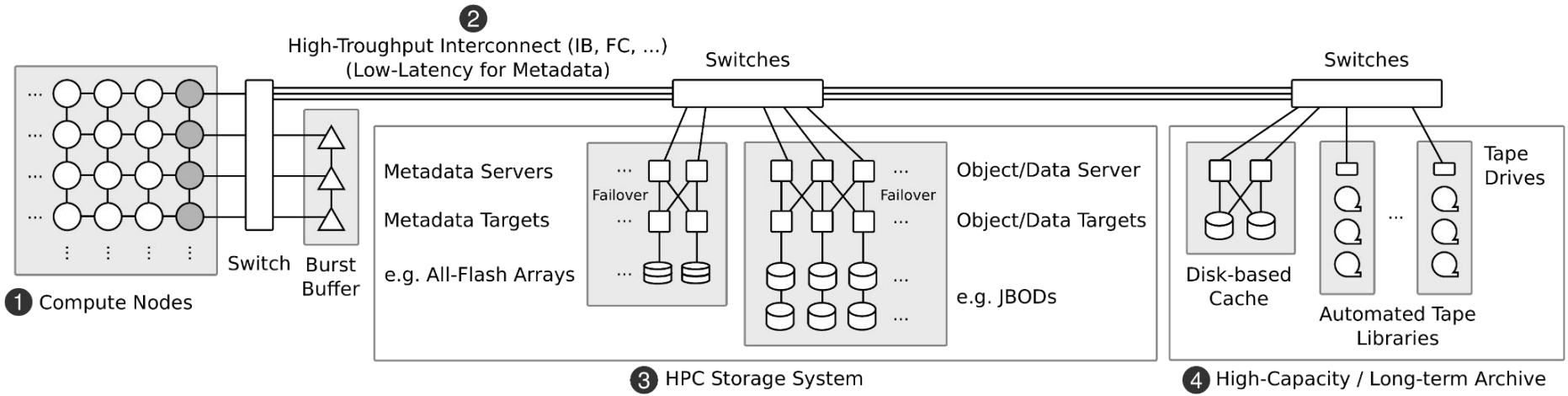
- Darshan in Brief
- PyDarshan Design and APIs

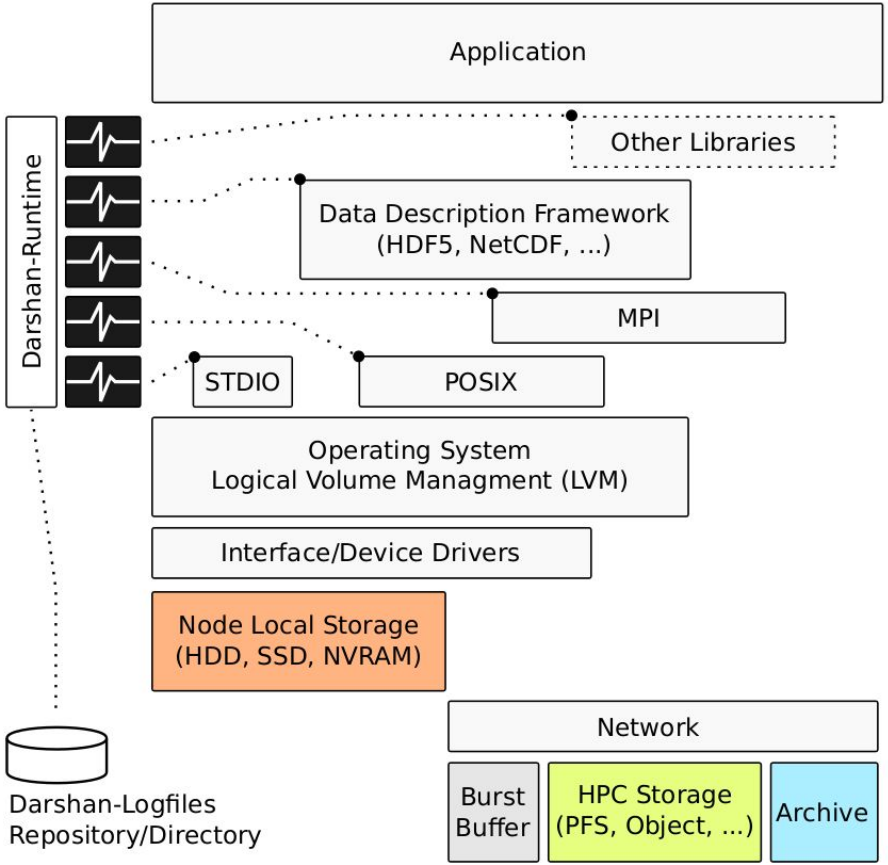
- Case Studies
  - Use Case 1: **Enhancing single job summaries** with HTML reports and modular templates for more interactivity using a large-scale run of the E3SM
  - Use Case 2: **Enabling custom analysis tools** building on top of Darshan using the examples of DXT Explorer and Drishti
  - Use Case 3: **Customizing I/O analysis of workflows** using the example of ATLAS AthenaMP, a high-energy physics simulation
  - Use Case 4: **Enabling the analysis of large bodies of Darshan logs** with hundreds of thousands of jobs on the Cori and Theta supercomputers

HPC I/O and Storage

A state of the art data center with multiple storage services/tiers. Increasingly heterogeneous;

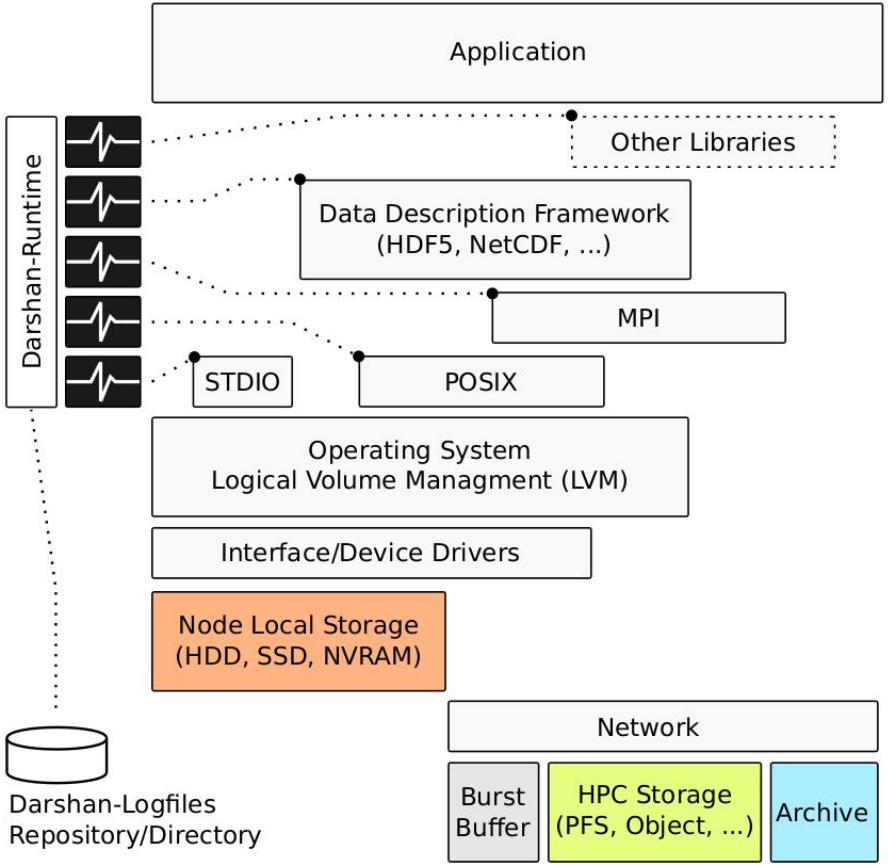
New storage/data services emerging. **This perspective is often unknown to users.**

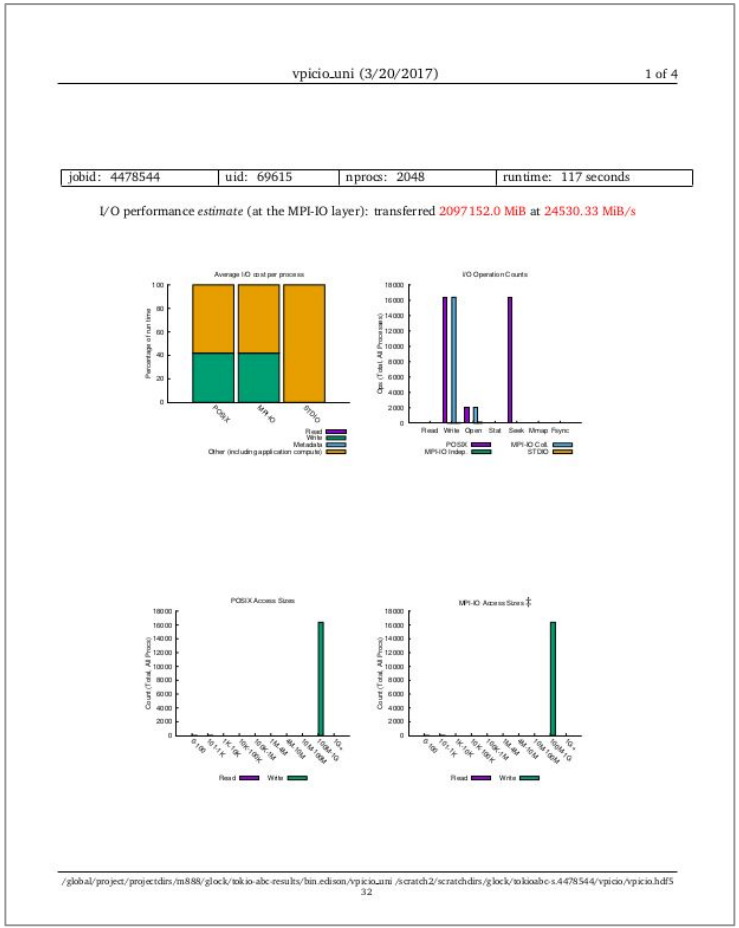





```
#!/bin/bash

export LD_PRELOAD=libdarshan.so
./app
```

Software Perspective: Data passes through **various middleware** layers to storage. Instrumentation, for example, using **Darshan**.





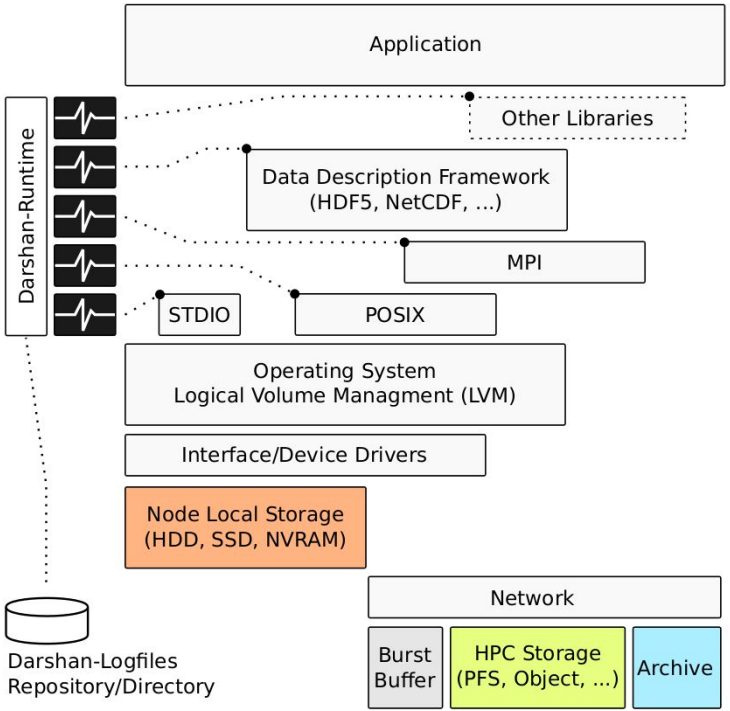
vpicio_uni (3/20/2017) 1 of 4

jobid: 4478544	uid: 69615	nprocs: 2048	runtime: 117 seconds

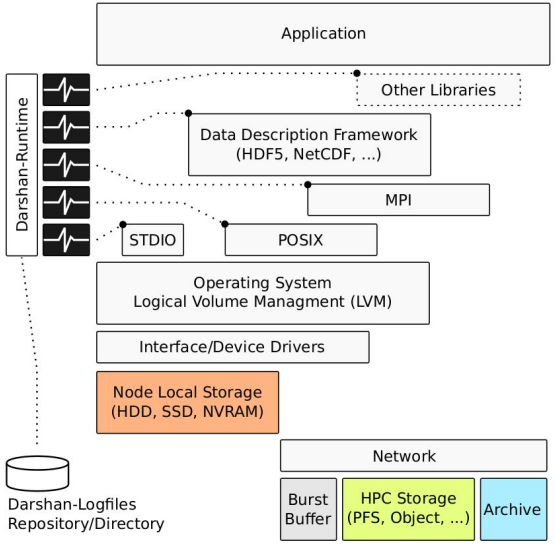
I/O performance *estimate* (at the MPI-IO layer): transferred 2097152.0 MiB at 24530.33 MiB/s

/global/project/projectdirs/m888/glock/tokio-abc-results/bin.edison/vpicio_uni /scratch2/scratchdirs/glock/tokioabc-s.4478544/vpicio/vpicio.hdf5
32

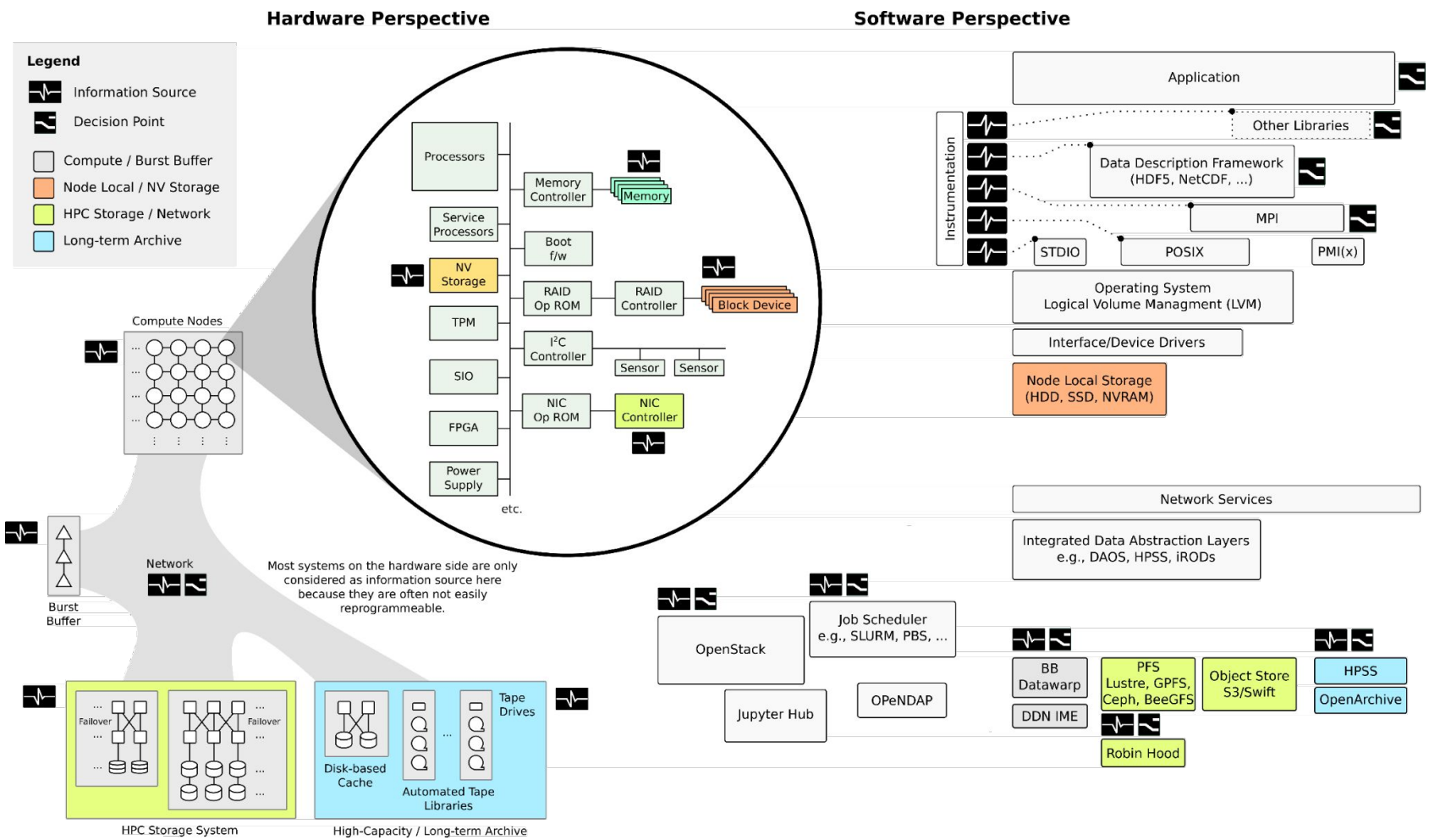
Software Perspective: Instrumentation, for example, using Darshan offers **job/app granularity! Interpretation requires understanding of execution context.**



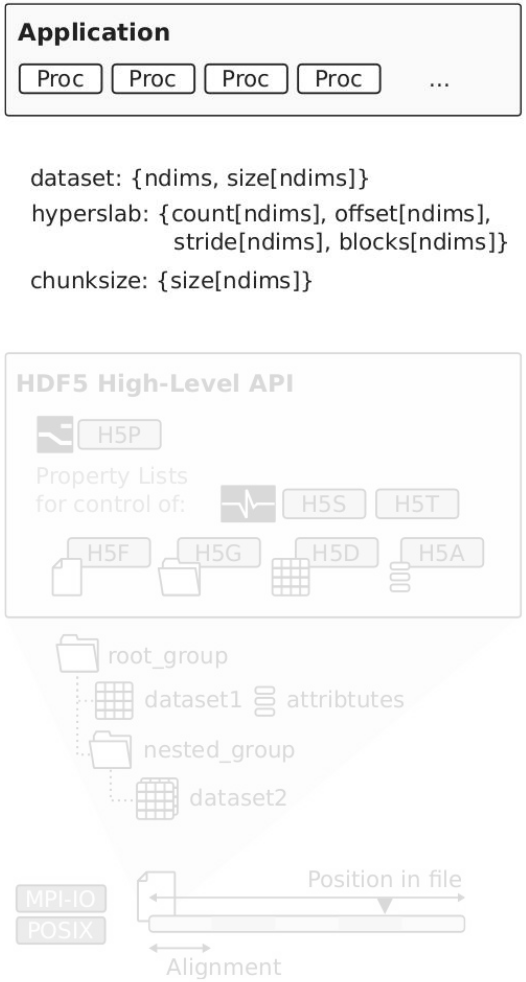
Need to consider broader execution context!



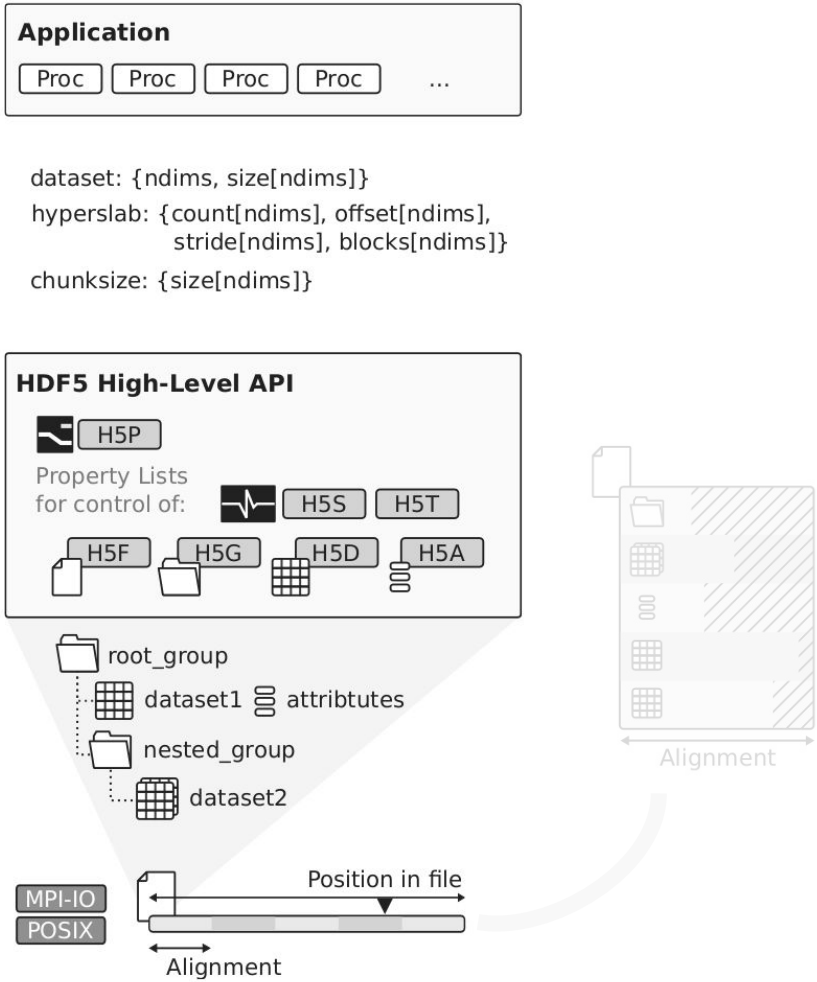
Need to consider broader execution context!



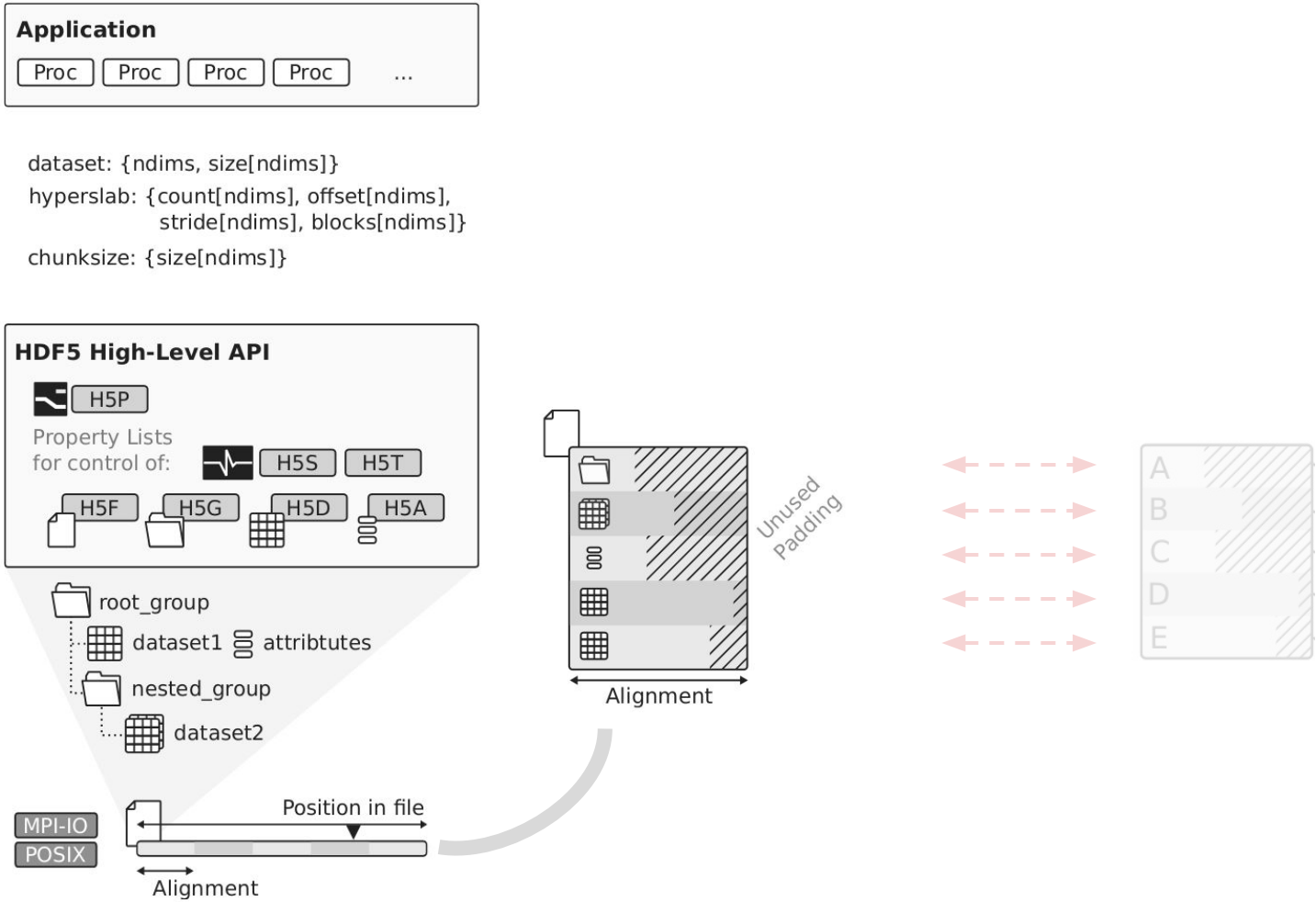
Holistic Perspective: Many technologies, many layers, many instrumentation/tunable APIs.



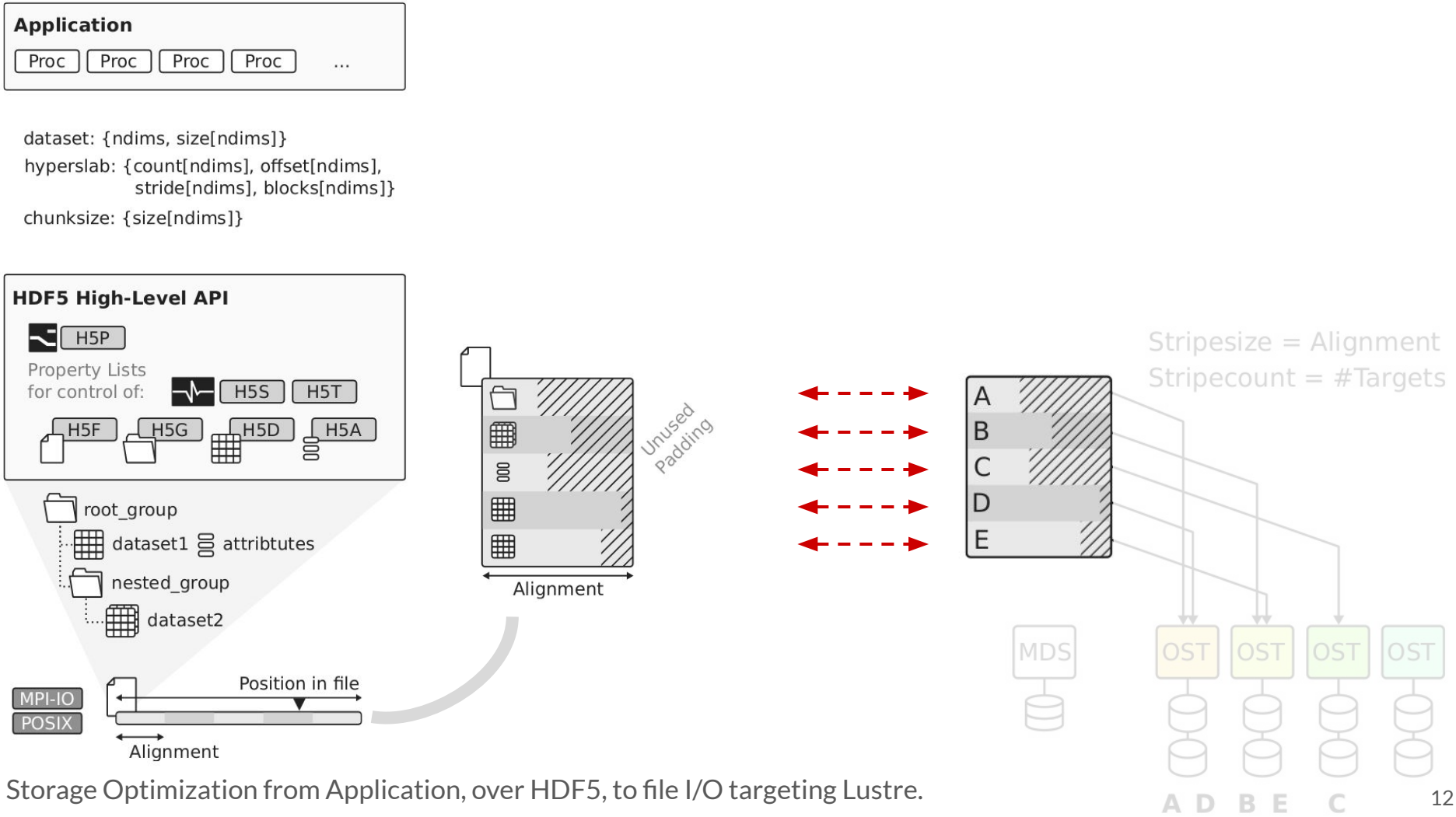
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.



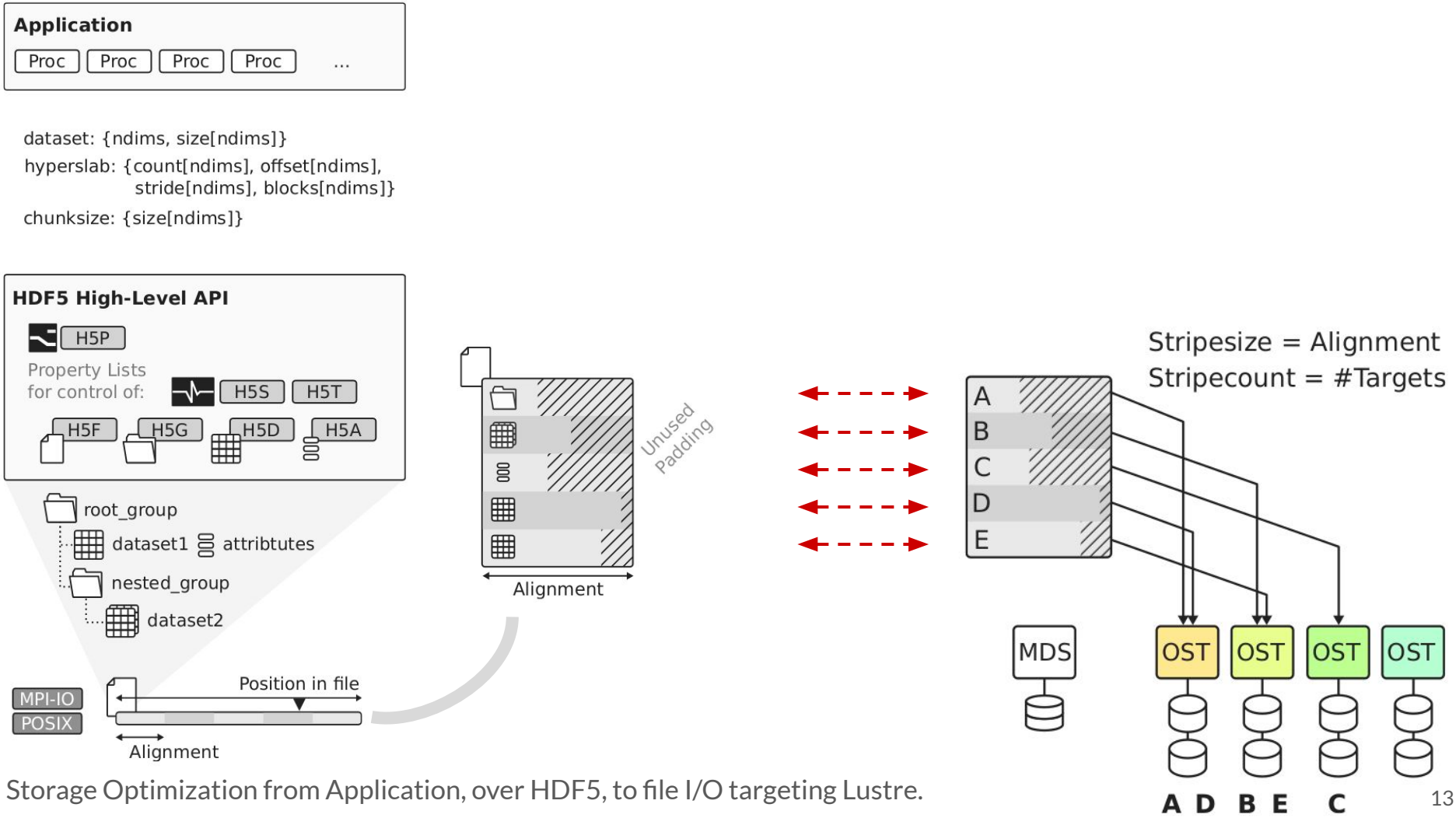
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.



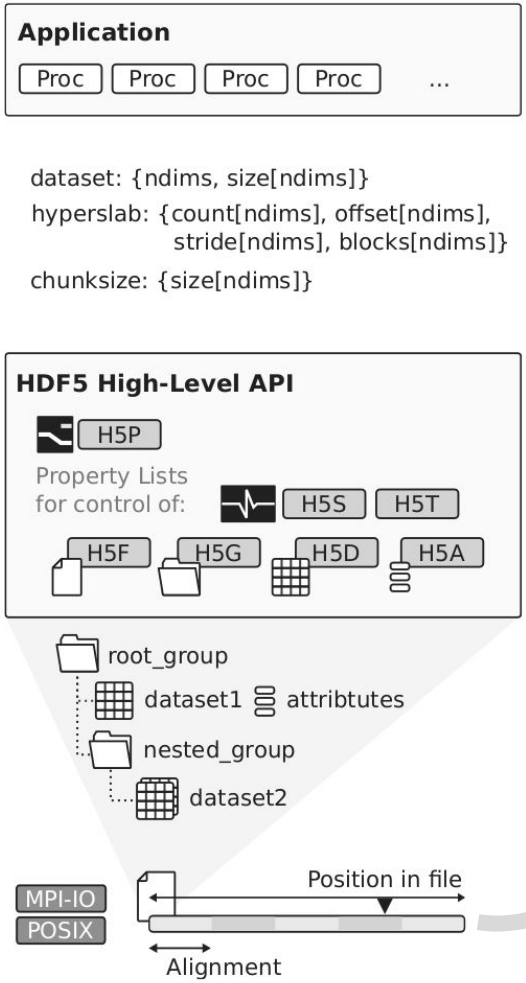
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.



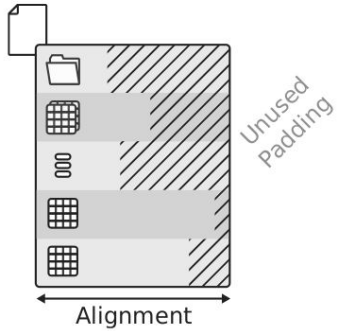
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.

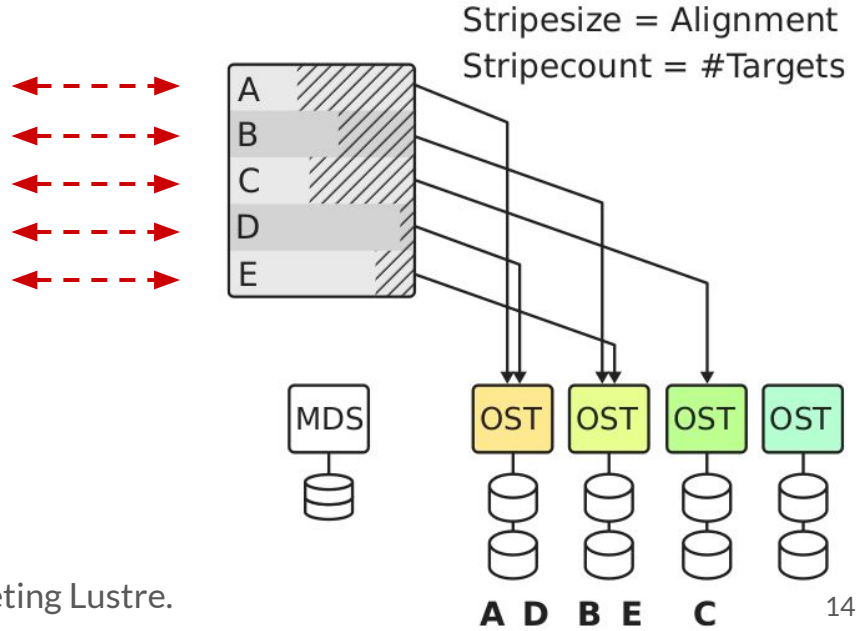


Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.



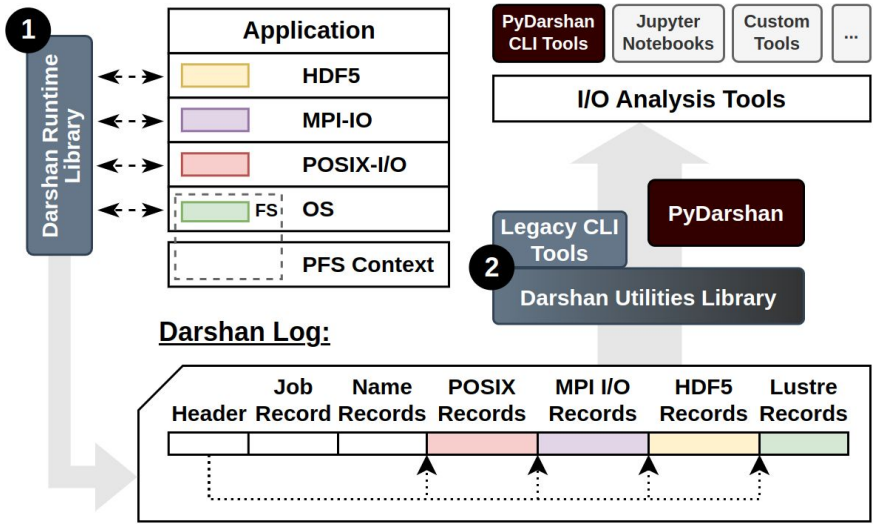
Logical perspective is lost as objects from an HDF5 perspective become just **byte-streams** from a file system perspective.





Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.

Darshan's Architecture



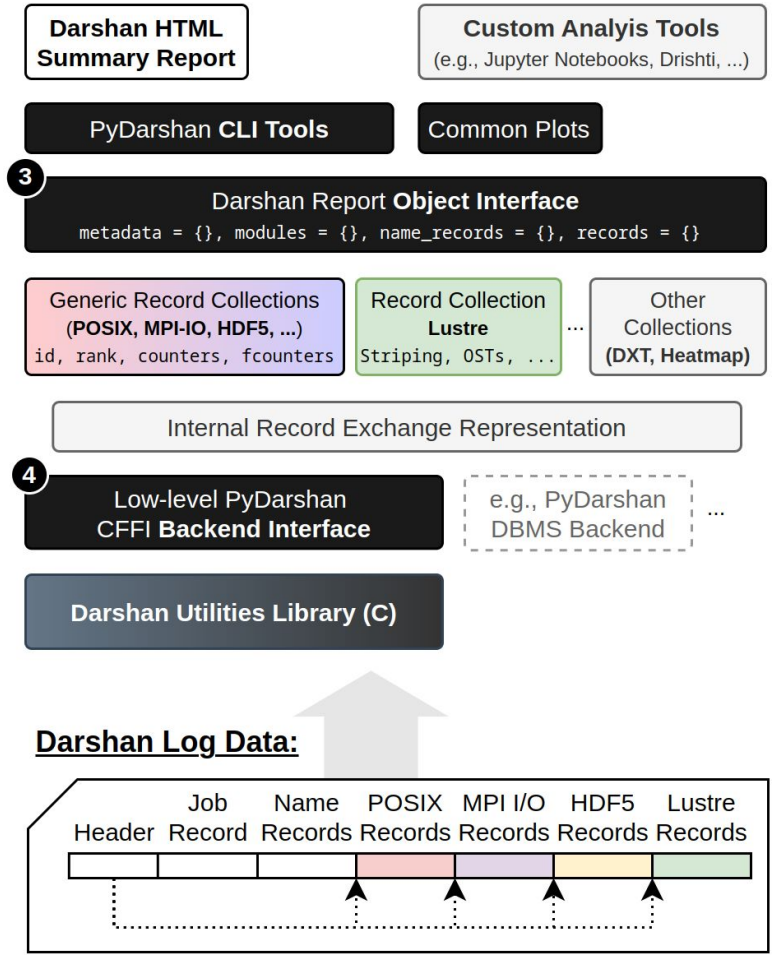
1. Multi-Layer Runtime Library for Instrumentation
   - POSIX, STDIO, MPI-IO, HDF-5, LUSTRE, …
   - Custom Modules
2. Darshan Utilities
   - CLI Tool Collection
     i. Parser
     ii. DXT Parser
     iii. Merge/Filter
   - PyDarshan
     i. Python API
     ii. Fine-Granular Access
     iii. Bridge to Python's data analysis and machine learning libraries/ecosystem

PyDarshan in Detail

- PyDarshan CLI Interface
- Darshan Report/Log Objects
- Darshan Record Collections
- Darshan CFFI-Backend to access binary data

Facilitate:

- Interactive Visualisation
- Modernization of Reports (HTML)
- Common Plots for Reuse
- Analysis in Jupyter Notebooks
- Fine-Grained Data Loading for Analysis and Machine Learning Libraries



PyDarshan API Usage Example

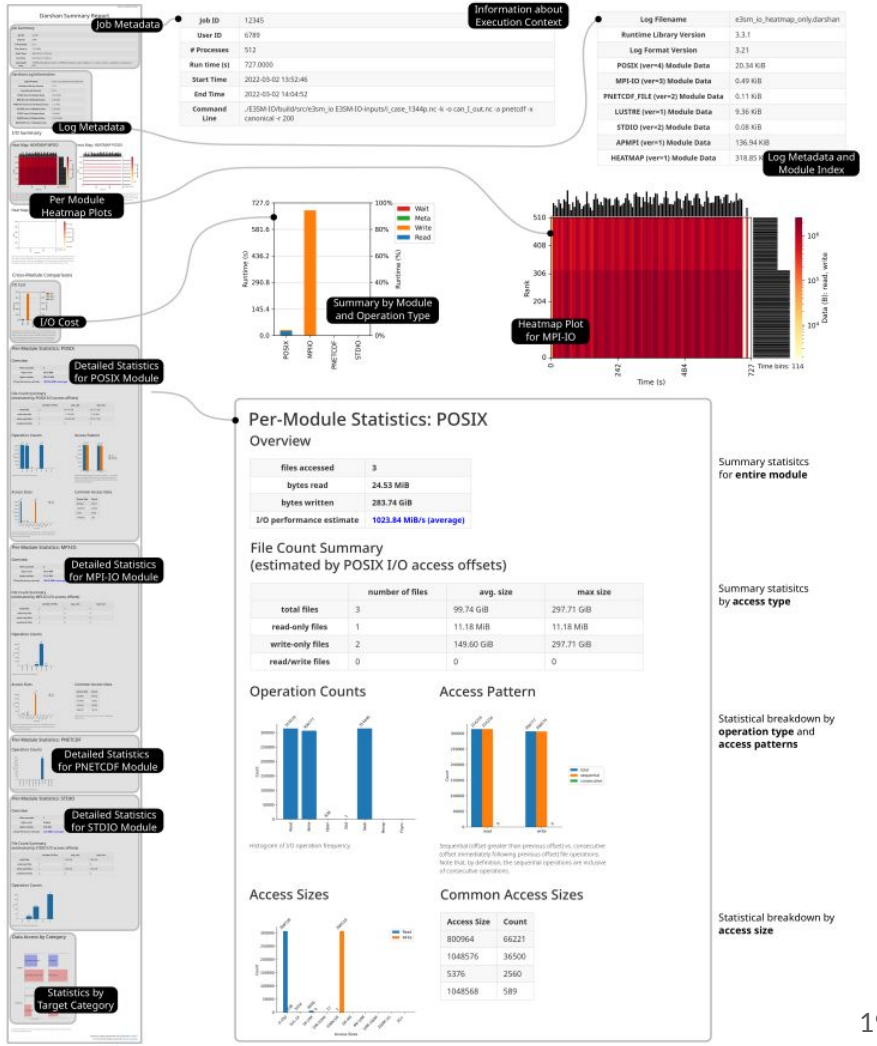
```
# open a Darshan log file and read all data
with darshan.DarshanReport(filename, read_all=True) as report:

    # print the metadata dict for this log
    print('metadata: ', report.metadata)

    # print job runtime and nprocs
    print('run_time: ', report.metadata['job']['run_time'])
    print('nprocs: ', report.metadata['job']['nprocs'])

    # print modules contained in the report
    print('modules: ', list(report.modules.keys()))

    # export POSIX module records to DataFrame and print
    posix_df = report.records['POSIX'].to_df()
    display(posix_df)
```

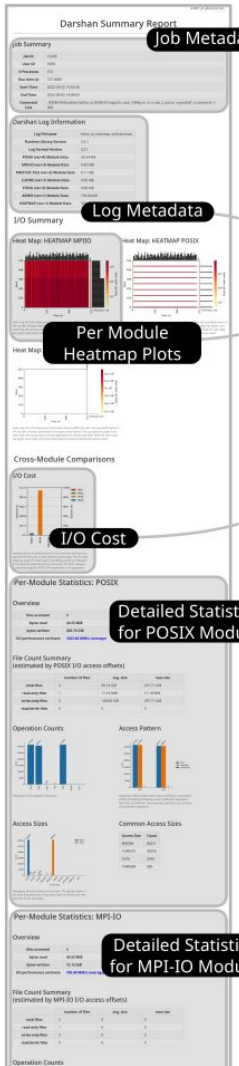
PyDarshan API Usage Example (Low-Level)

```
import darshan.backend.cffi_backend  as darshanll

log = darshanll.log_open( 'example.darshan' )

# Access various job information
darshanll.log_get_job(log)
# Example Return:
# {'jobid': 4478544, 'uid': 69615,
# 'start_time': 1490000867, 'end_time': 1490000983,
# 'metadata': {'lib_ver': '3.1.3', 'h': 'romio_no_indep_rw=true;cb_nodes=4'}}

# Access available modules and modules
darshanll.log_get_modules(log)
# Example Return:
# {'POSIX': {'len': 186, 'ver': 3, 'idx': 1},
#  'MPI-IO': {'len': 154, 'ver': 2, 'idx': 2},
#  'LUSTRE': {'len': 87, 'ver': 1, 'idx': 6},
#  'STDIO': {'len': 3234, 'ver': 1, 'idx': 7}}

# Access different record types as numpy arrays, with integer and float counters separated
# Example Return: {'counters': array([...], dtype=uint64), 'fcounters': array([...])}
posix_record = darshanll.log_get_record(log , 'POSIX')
mpiio_record = darshanll.log_get_record(log , 'MPI-IO')
stdio_record = darshanll.log_get_record(log , 'STDIO')
# ...

darshanll.log_close(log)
```

Use Case 1:
Enhancing single job summaries with HTML reports

Allowing modular templates for more interactivity using a large-scale run of the E3SM





Job Metadata

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

Information about Execution Context

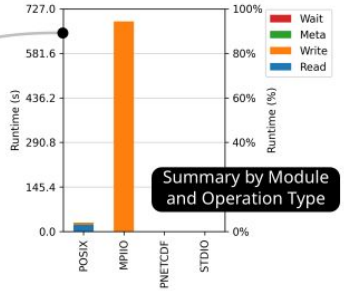
Log Filename	e3sm_io_heatmap_only.darshan
Runtime Library Version	3.3.1
Log Format Version	3.21
POSIX (ver=4) Module Data	20.34 KiB
MPI-IO (ver=3) Module Data	0.49 KiB
PNETCDF_FILE (ver=2) Module Data	0.11 KiB
LUSTRE (ver=1) Module Data	9.36 KiB
STDIO (ver=2) Module Data	0.08 KiB
APMPI (ver=1) Module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

Log Metadata and Module Index

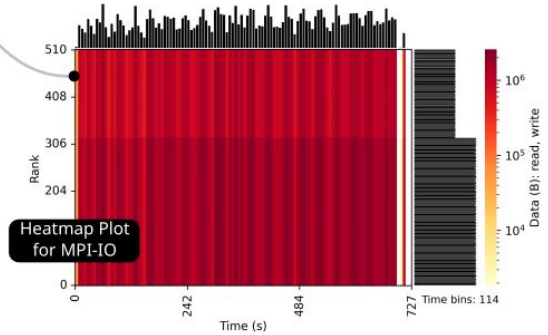
Log Metadata

Per Module Heatmap Plots

I/O Cost



Summary by Module and Operation Type



Heatmap Plot for MPI-IO

Detailed Statistics for POSIX Module

Detailed Statistics for MPI-IO Module

Per-Module Statistics: POSIX

Overview

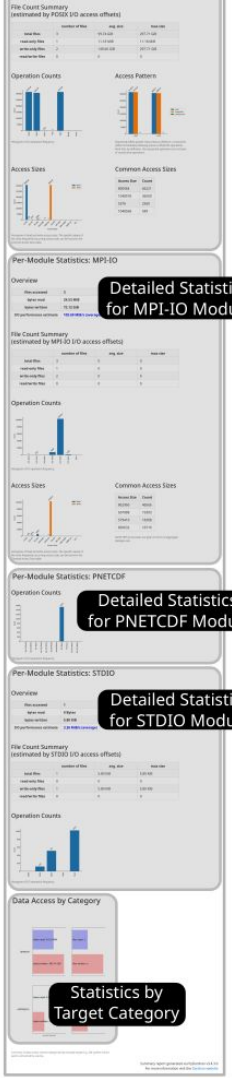
files accessed	**3**
bytes read	**24.53 MiB**
bytes written	**283.74 GiB**
I/O performance estimate	**1023.84 MiB/s (average)**

Summary statisitcs for **entire module**

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

Summary statisitcs by **access type**



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)**

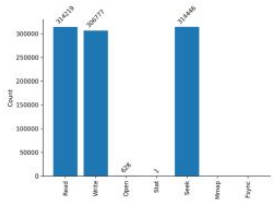
Summary statisitcs for **entire module**

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

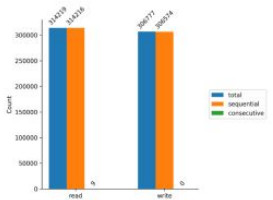
Summary statisitcs by **access type**

Operation Counts



Histogram of I/O operation frequency.

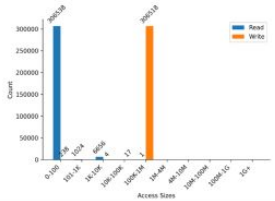
Access Pattern



Sequential (offset greater than previous offset) vs. consecutive (offset immediately following previous offset) file operations. Note that, by definition, the sequential operations are inclusive of consecutive operations.

Statistical breakdown by **operation type** and **access patterns**

Access Sizes



Common Access Sizes

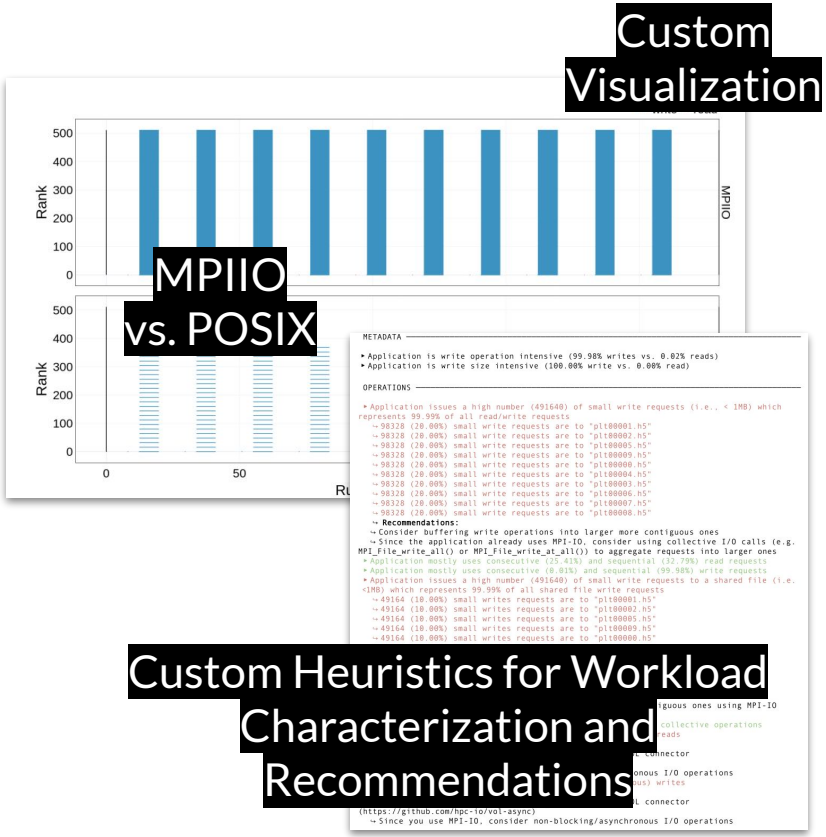
Access Size	Count
800964	66221
1048576	36500
5376	2560
1048568	589

Statistical breakdown by **access size**

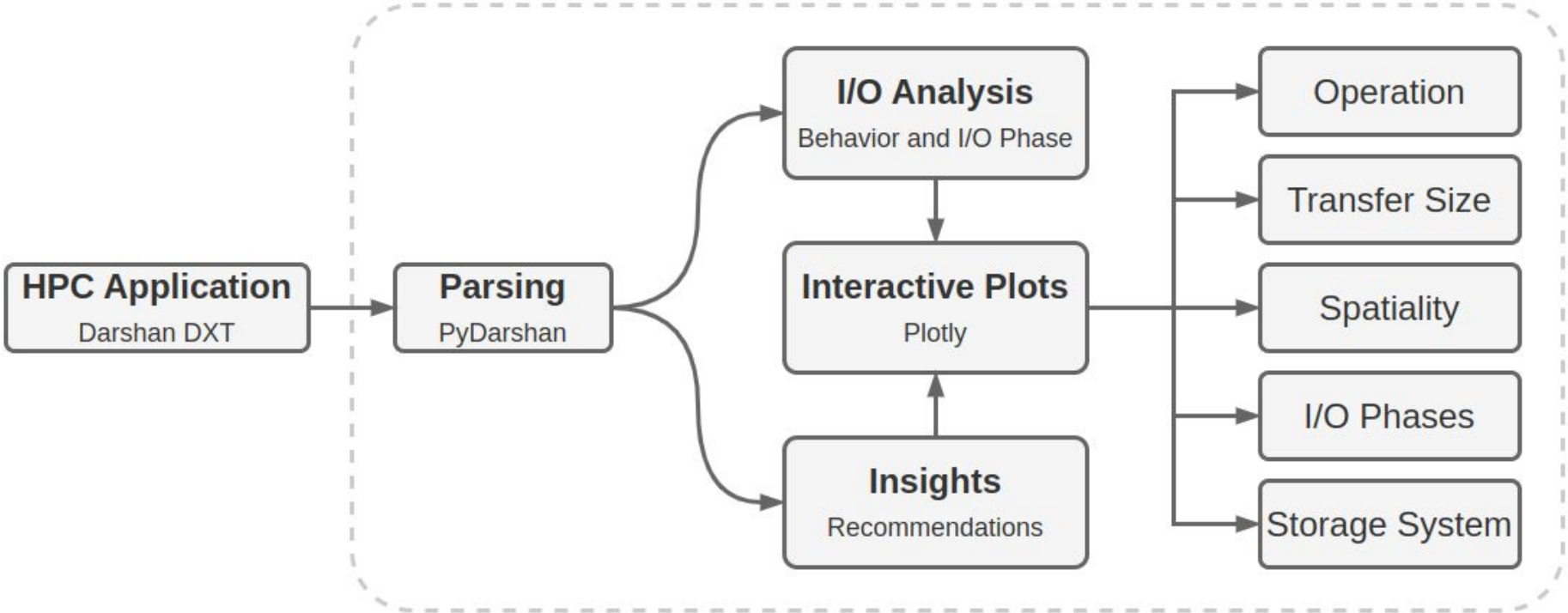
Use Case 2:

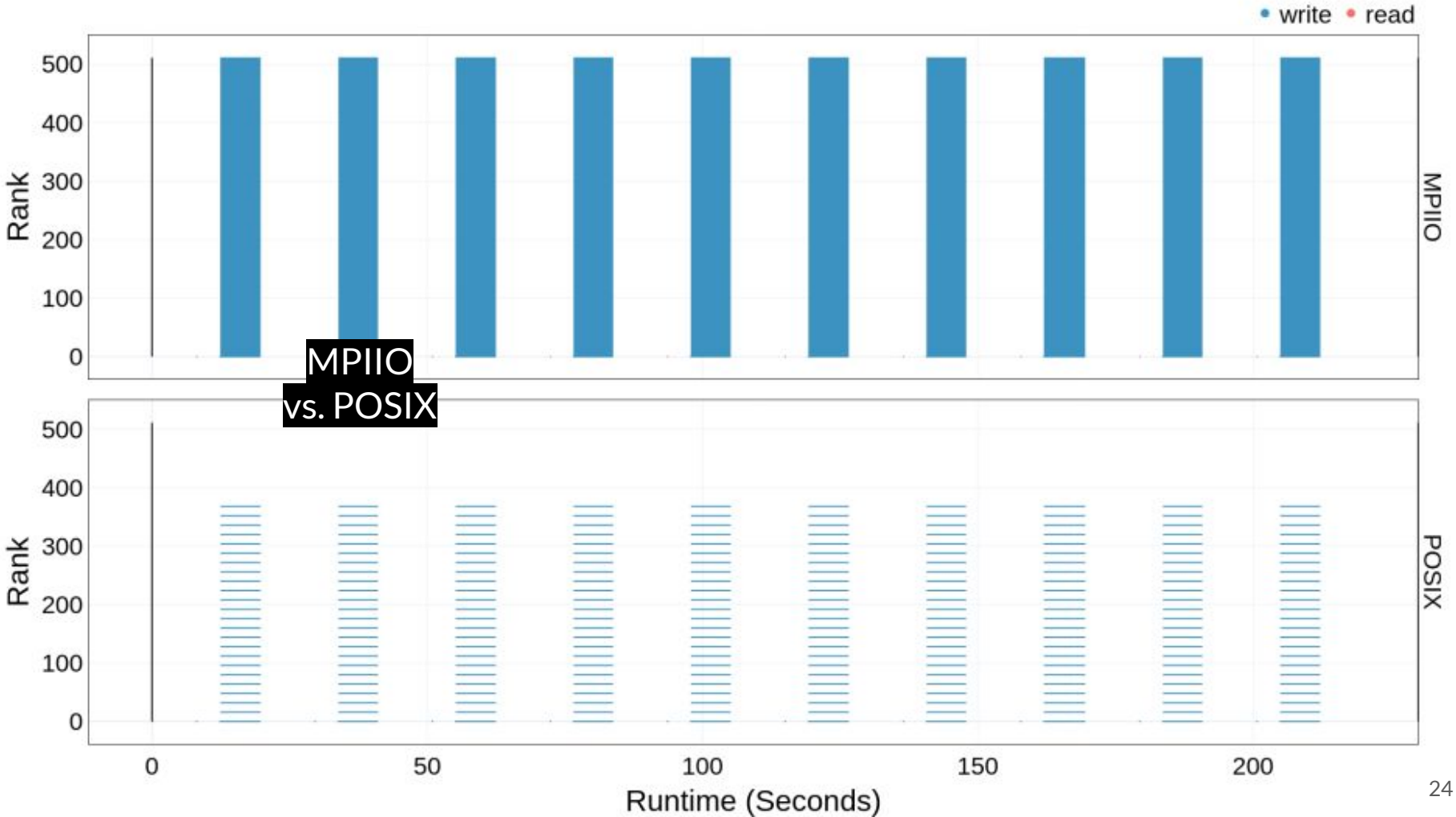
Enabling custom analysis tools building on top of Darshan

Here using the examples of
DXT Explorer and Drishti:
https://github.com/hpc-io/dxt-explorer
https://github.com/hpc-io/drishti-io



Drishti Overview





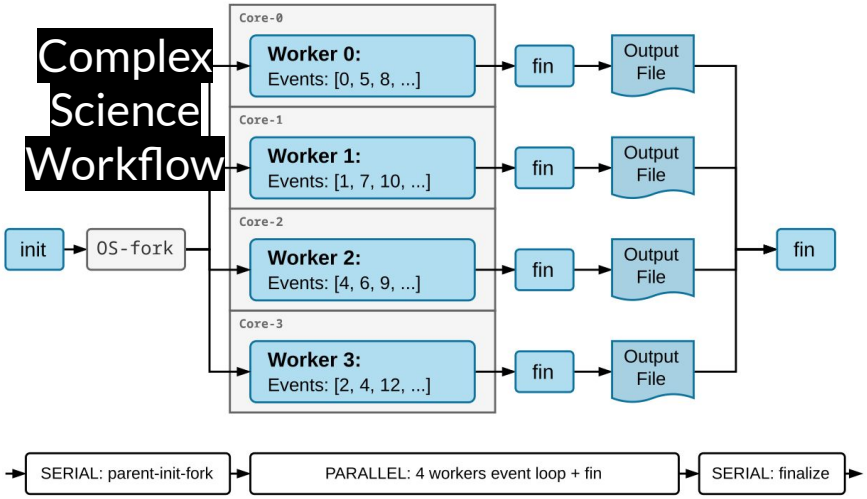
MPIIO
vs. POSIX

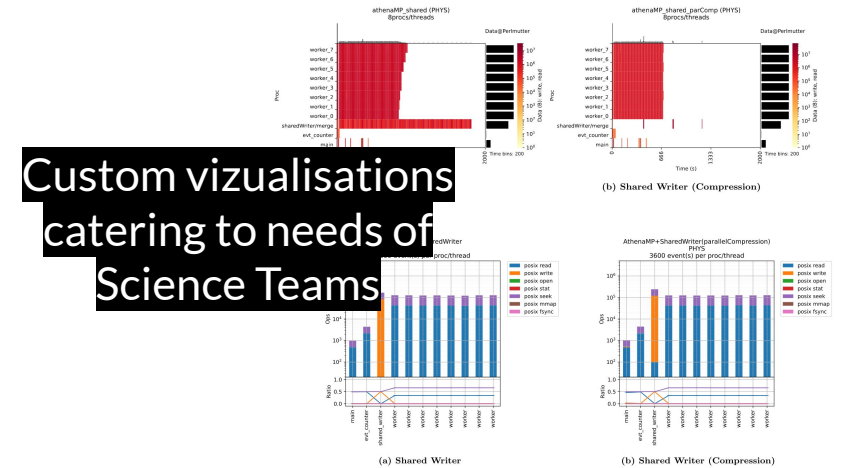
```
METADATA ──────────────────────────────────────────────────────────────────────

▸ Application is write operation intensive (99.98% writes vs. 0.02% reads)
▸ Application is write size intensive (100.00% write vs. 0.00% read)

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

 ▸ Application issues a high number (491640) of small write requests (i.e., < 1MB) which
represents 99.99% of all read/write requests
   ↪ 98328 (20.00%) small write requests are to "plt00001.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00002.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00005.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00009.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00000.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00004.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00003.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00006.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00007.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00008.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 (25.41%) and sequential (32.79%) read requests
 ▸ Application mostly uses consecutive (0.01%) and sequential (99.98%) write requests
 ▸ Application issues a high number (491640) of small write requests to a shared file (i.e.
 <1MB) which represents 99.99% of all shared file write requests
   ↪ 49164 (10.00%) small writes requests are to "plt00001.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00002.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00005.h5"
```

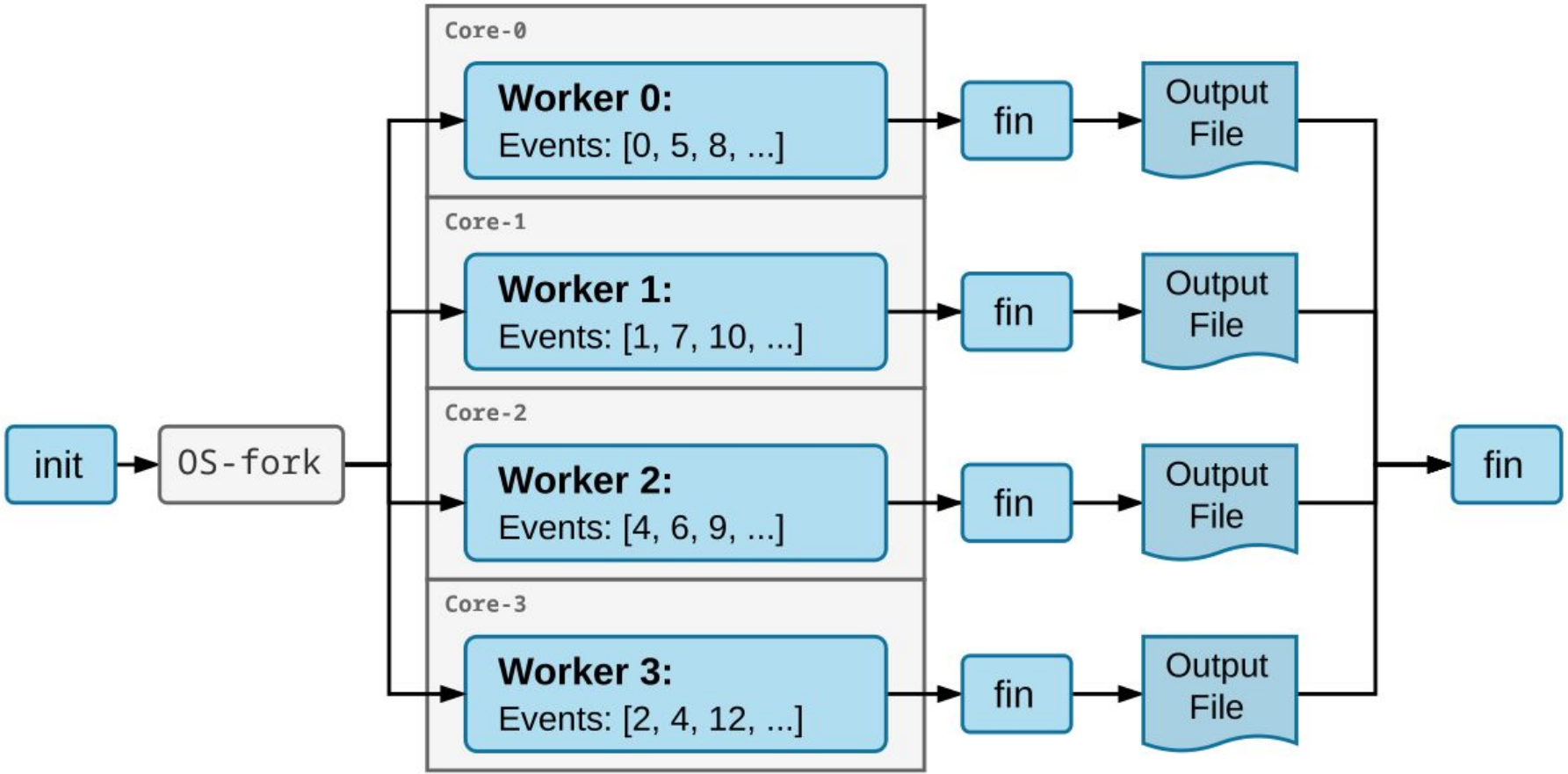
Custom Heuristics for Workload Characterization and Recommendations

Use Case 3: **Customizing I/O analysis of workflows**

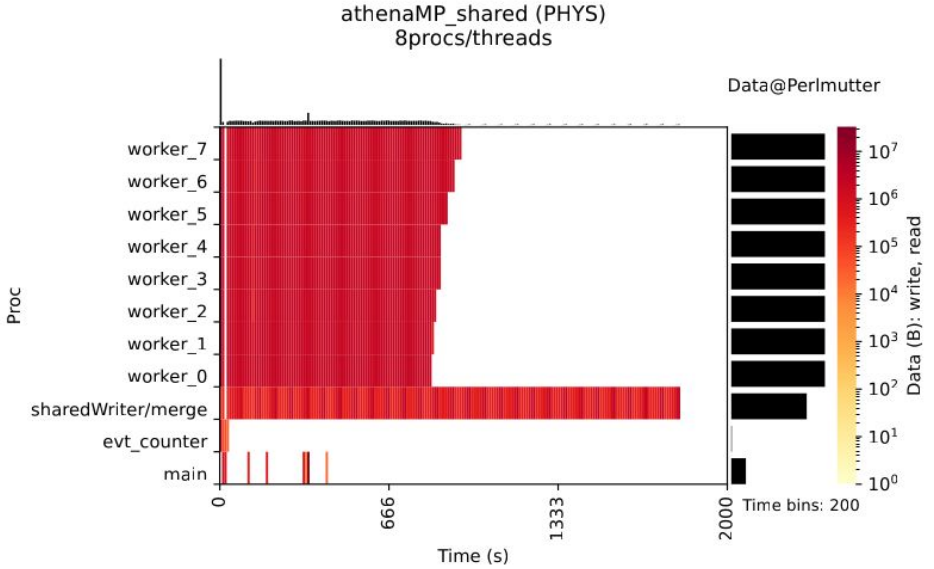
Using the example of ATLAS AthenaMP, a high-energy physics simulation.



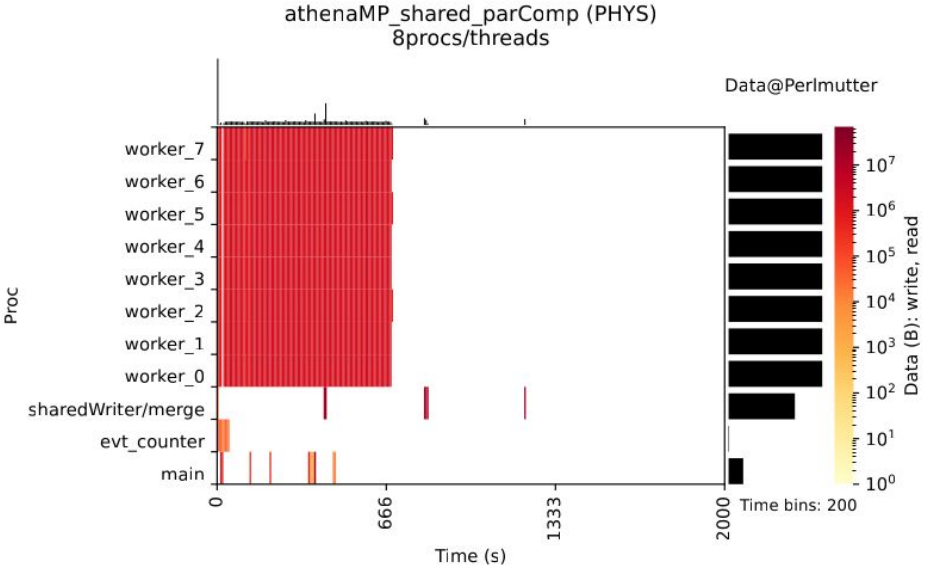




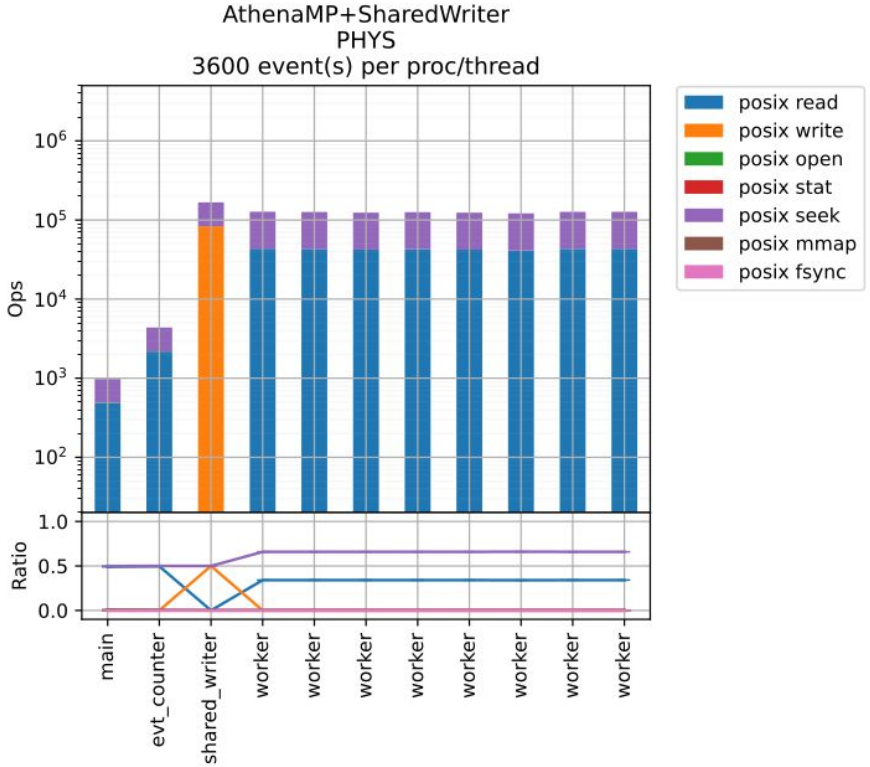
SERIAL: parent-init-fork → PARALLEL: 4 workers event loop + fin → SERIAL: finalize



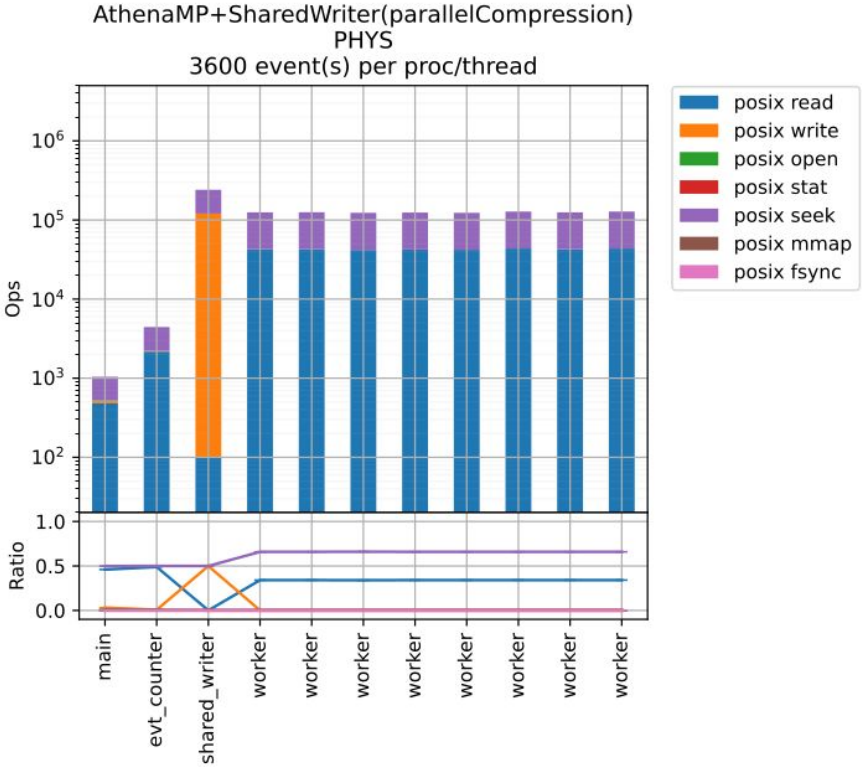
(a) Shared Writer



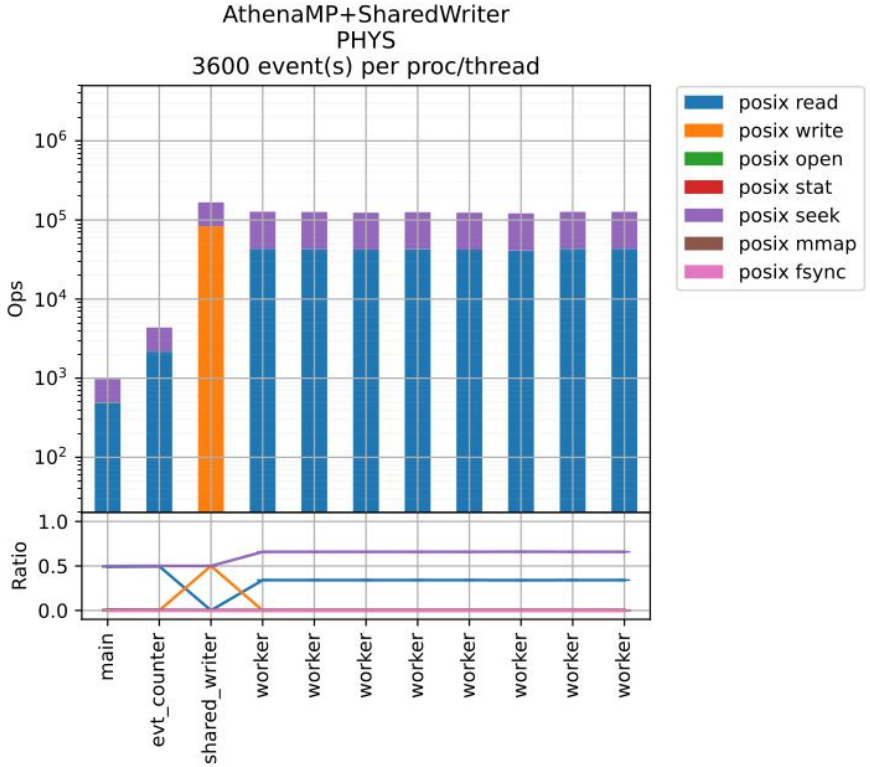
(b) Shared Writer (Compression)



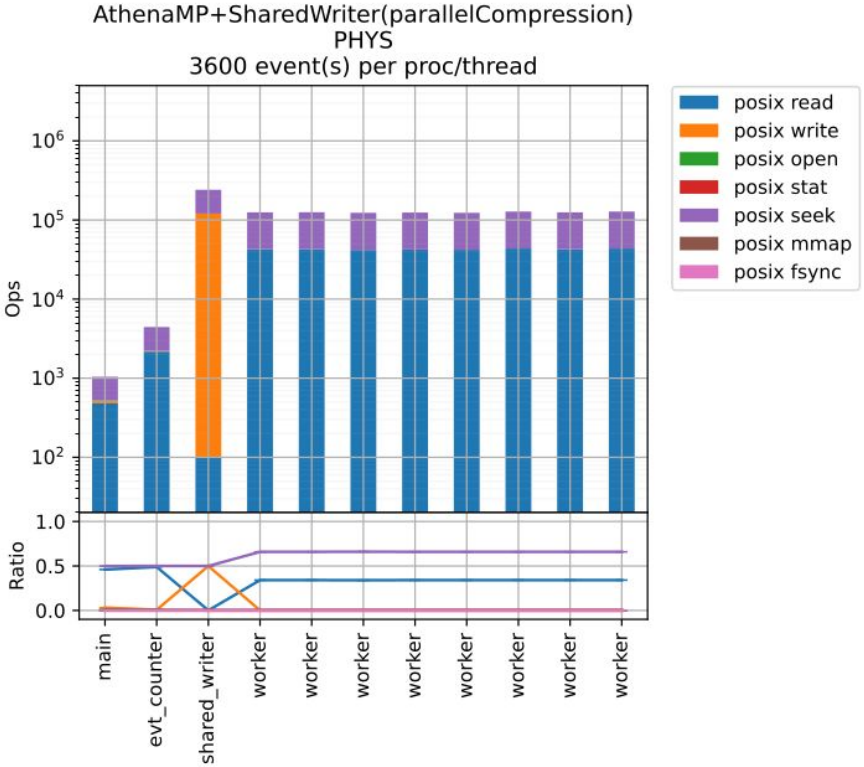
(a) Shared Writer



(b) Shared Writer (Compression)



(a) Shared Writer



(b) Shared Writer (Compression)

```
[...]

Out of memory: Kill process 43805
Killed process 43805

$ |
```

Use Case 4: **Enabling the Analysis of large bodies of Darshan logs**

Scaling to hundreds of thousands of jobs on the Cori and Theta supercomputers

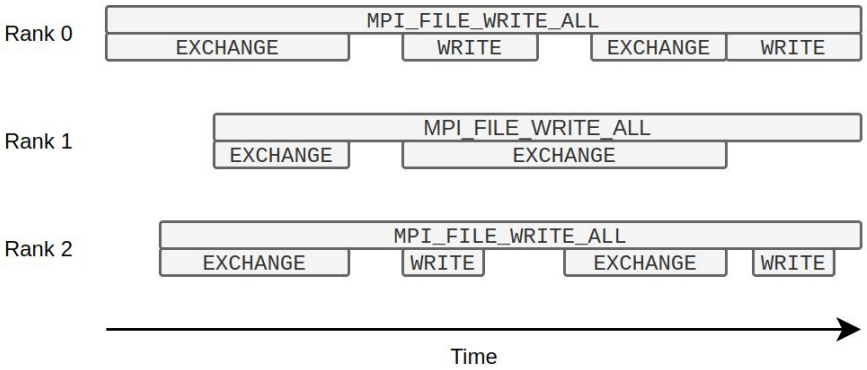
```
[...]

Out of memory: Kill process 43805
Killed process 43805

$ |
```

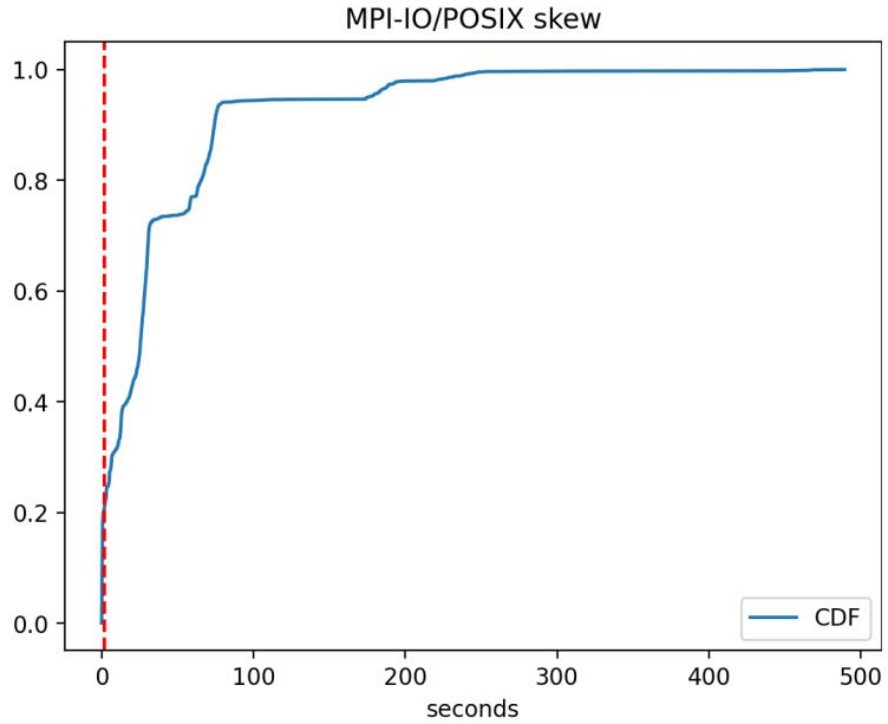
Cori		Theta	
Jobs	Total (%)	**Jobs**	Total (%)
158592	30.61	3430	45.16
75742	14.62	2695	35.48
267228	51.59	2282	30.04
136664	26.38	1197	15.76
207346	40.03	3201	42.14
322130	62.19	3203	42.17

Site-Wide Analysis: MPI I/O Collective Skew Across Ranks

Collective operations across ranks routinely have to wait for POSIX I/O in other ranks to finish. This use case demonstrates how PyDarshan can be used to quantify the skew that occurs between the MPI I/O and POSIX layers.

Example:





Level	Interface	Detected Behavior	Cori		Theta	
			Jobs	Total (%)	Jobs	Total (%)
HIGH	STDIO	High STDIO usage (> 10% of total transfer size uses STDIO)	158592	30.61	3430	45.16
INFO	POSIX	Write operation count intensive (> 10% more writes than reads)	75742	14.62	2695	35.48
INFO	POSIX	Read operation count intensive (> 10% more reads than writes)	267228	51.59	2282	30.04
INFO	POSIX	Write size intensive (> 10% more bytes written then read)	136664	26.38	1197	15.76
INFO	POSIX	Read size intensive (> 10% more bytes read then written)	207346	40.03	3201	42.14
HIGH	POSIX	High number of small ($< 1MB$) read requests (> 10% of total read requests)	322130	62.19	3203	42.17
HIGH	POSIX	High number of small ($< 1MB$) write requests (> 10% of total write requests)	309868	59.82	3386	44.58
HIGH	POSIX	High number of misaligned memory requests (> 10%)	191214	36.91	5012	65.99
HIGH	POSIX	High number of misaligned file requests (> 10%)	344503	66.51	4974	65.49
WARN	POSIX	Redundant reads	13113	2.53	605	7.96
WARN	POSIX	Redundant writes	1407	0.27	40	0.52
HIGH	POSIX	High number of random read requests (> 20%)	174148	33.62	738	9.71
OK	POSIX	High number of sequential read operations ($\geq$ 80%)	169283	32.68	4273	56.26
HIGH	POSIX	High number of random write requests (> 20%)	2038	0.39	10	0.13
OK	POSIX	High number of sequential write operations ($\geq$ 80%)	342581	66.14	4275	56.28
HIGH	POSIX	High number of small ($< 1MB$) reads to shared-files (> 10% of total reads)	273274	52.76	2286	30.09
HIGH	POSIX	High number of small ($< 1MB$) writes to shared-files (> 10% of total writes)	46384	8.95	1300	17.11
HIGH	POSIX	High metadata time (at least one rank spends > 30 seconds)	19443	3.75	364	4.79
HIGH	POSIX	Data transfer imbalance between ranks causing stragglers (> 15% difference)	286811	55.37	3136	41.29
HIGH	POSIX	Time imbalance between ranks causing stragglers (> 15% difference)	201206	38.84	2559	33.69
HIGH	POSIX	Write imbalance (> 30%) when accessing individual files	3178	0.61	578	7.61
HIGH	POSIX	Read imbalance (> 30%) when accessing individual files	5265	1.01	652	8.58
WARN	MPI-IO	No MPI-IO calls detected from Darshan logs	511731	98.80	6518	85.81
HIGH	MPI-IO	Detected MPI-IO but no collective read operation	3162	0.61	31	0.40
HIGH	MPI-IO	Detected MPI-IO but no collective write operation	612	0.11	1	0.013
OK	MPI-IO	Detected MPI-IO and collective read operations	1329	0.25	743	9.78
OK	MPI-IO	Detected MPI-IO and collective write operations	3266	0.63	1009	13.28
WARN	MPI-IO	Detected MPI-IO but no non-blocking read operations	6208	1.19	1077	14.18
WARN	MPI-IO	Detected MPI-IO but no non-blocking write operations	6208	1.19	1077	14.18

Summary

- PyDarshan Design and APIs
- Case Studies
  - Use Case 1: **Enhancing single job summaries** with HTML reports and modular templates for more interactivity using a large-scale run of the E3SM
  - Use Case 2: **Enabling custom analysis tools** building on top of Darshan using the examples of DXT Explorer and Drishti
  - Use Case 3: **Customizing I/O analysis of workflows** using the example of ATLAS AthenaMP, a high-energy physics simulation
  - Use Case 4: **Enabling the analysis of large bodies of Darshan logs** with hundreds of thousands of jobs on the Cori and Theta supercomputers

Find out more or contribute at:
https://www.mcs.anl.gov/research/projects/darshan/
https://github.com/darshan-hpc/darshan

Try it yourself:

```
pip install darshan
```