Parallel I/O with HDF5 and Performance Tuning Techniques

The HDF Group

M. Scot Breitenfeld

Outline

The HDF Group

- A brief overview of past general best practices for HDF5
- Recent best practice findings for parallel performance

Resources

The HDF Group

- HDF5 home page: http://hdfgroup.org/HDF5/
  - HDF forum, webinars, YouTube channel, help@hdfgroup.org
- HDF5 Jira: https://jira.hdfgroup.org, GitHub issue tracker.
- Documentation: https://docs.hdfgroup.org/hdf5/develop/
  - Online tutorials https://portal.hdfgroup.org/display/HDF5/Introduction+to+Parallel+HDF5
  - In-person tutorials
    - Super Computing Conference (MPI IO)
    - National Laboratories (Argonne Training Program on Extreme-Scale Computing (ATPESC))
- HDF5 repo: https://github.com/HDFGroup/hdf5
- Latest releases: https://portal.hdfgroup.org/display/support/Downloads
  - HDF5 1.8.22
  - HDF5 1.10.8
  - HDF5 1.12.2
  - HDF5 1.13.1 (pre-production 1.14 release)

Useful pre-tuned third-party alternatives

The HDF Group

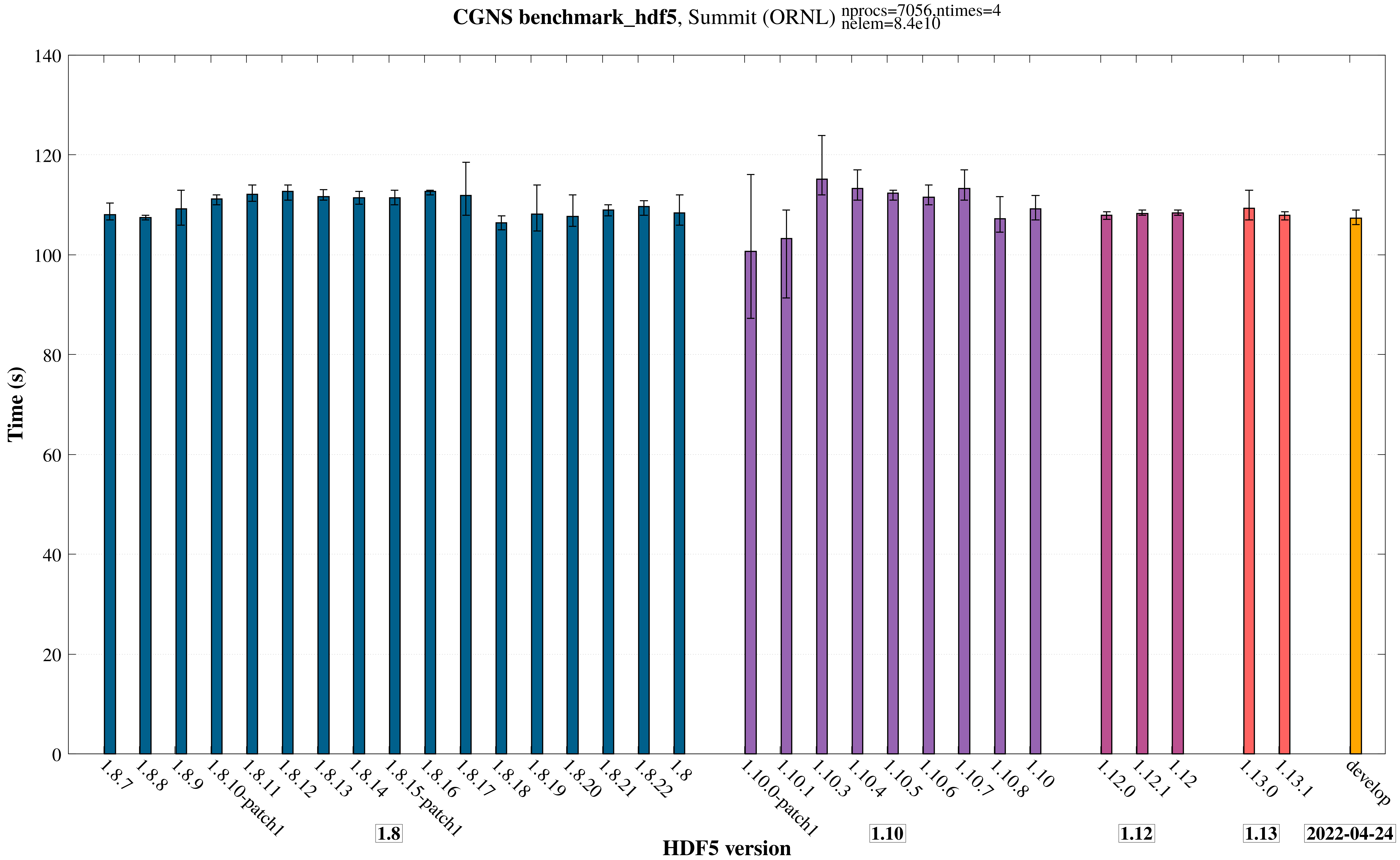
- Don’t open the hood, consider,
  - Alternatives to the C-API, Fine choices: h5py, rhdf5, H5CPP, HDF5.jl, etc.
  - Third-party HDF5 based libraries (netCDF, CGNS)

- CGNS = Computational Fluid Dynamics (CFD) General Notation System
- An effort to standardize CFD input and output data, including:
  - Grid (both structured and unstructured), flow solution
  - Connectivity, boundary conditions, auxiliary information.
- Two parts:
  - A standard format for recording the data
  - Software that reads, writes, and modifies data in that format.
- An American Institute of Aeronautics and Astronautics Recommended Practice

CGNS cfd data standard

AIAA Shaping the Future of Aerospace

Useful for monitoring HDF5 Performance

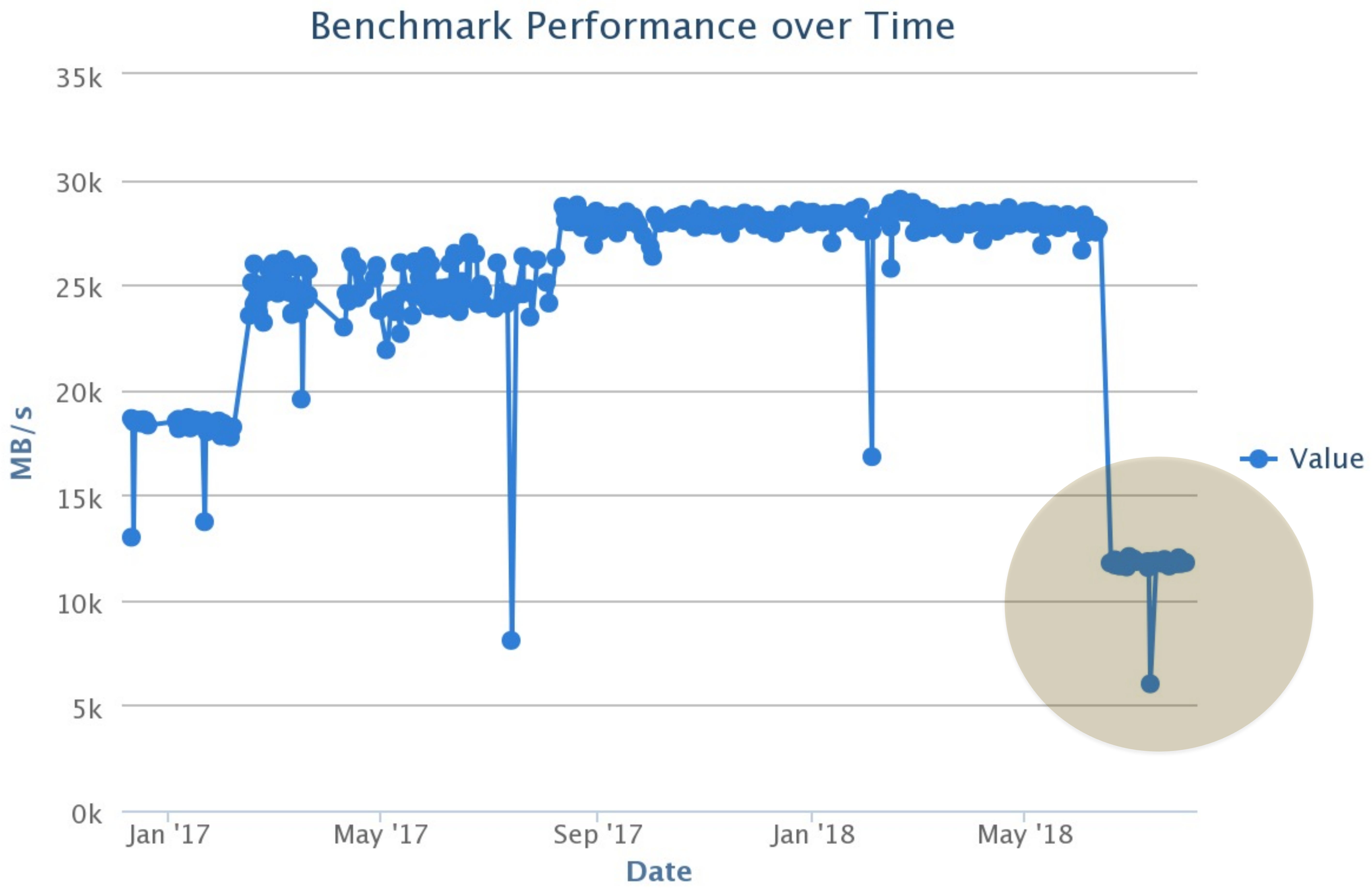


Past Performance Best Practice Findings

Effects of Software/Hardware Changes

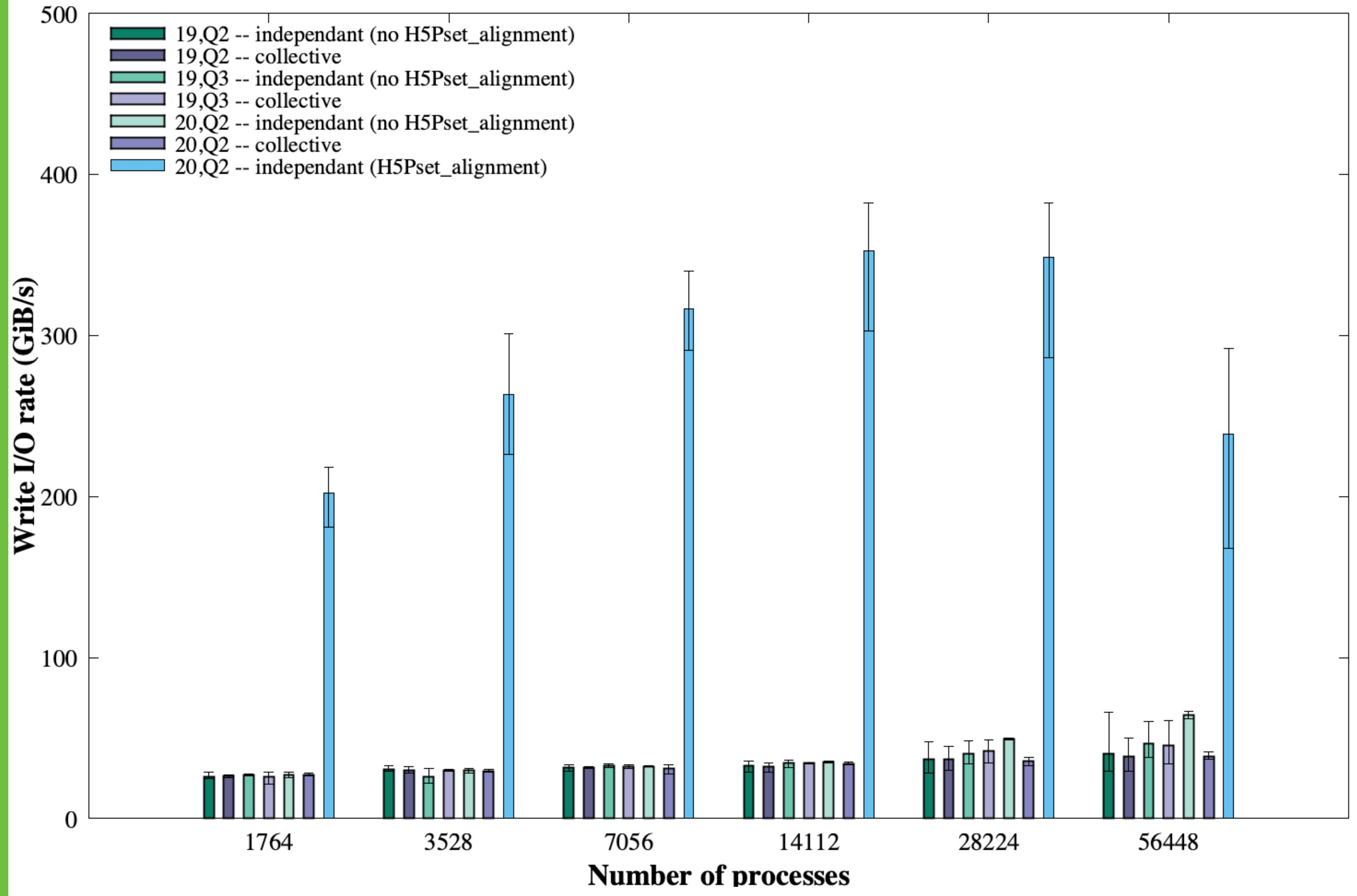
The HDF Group

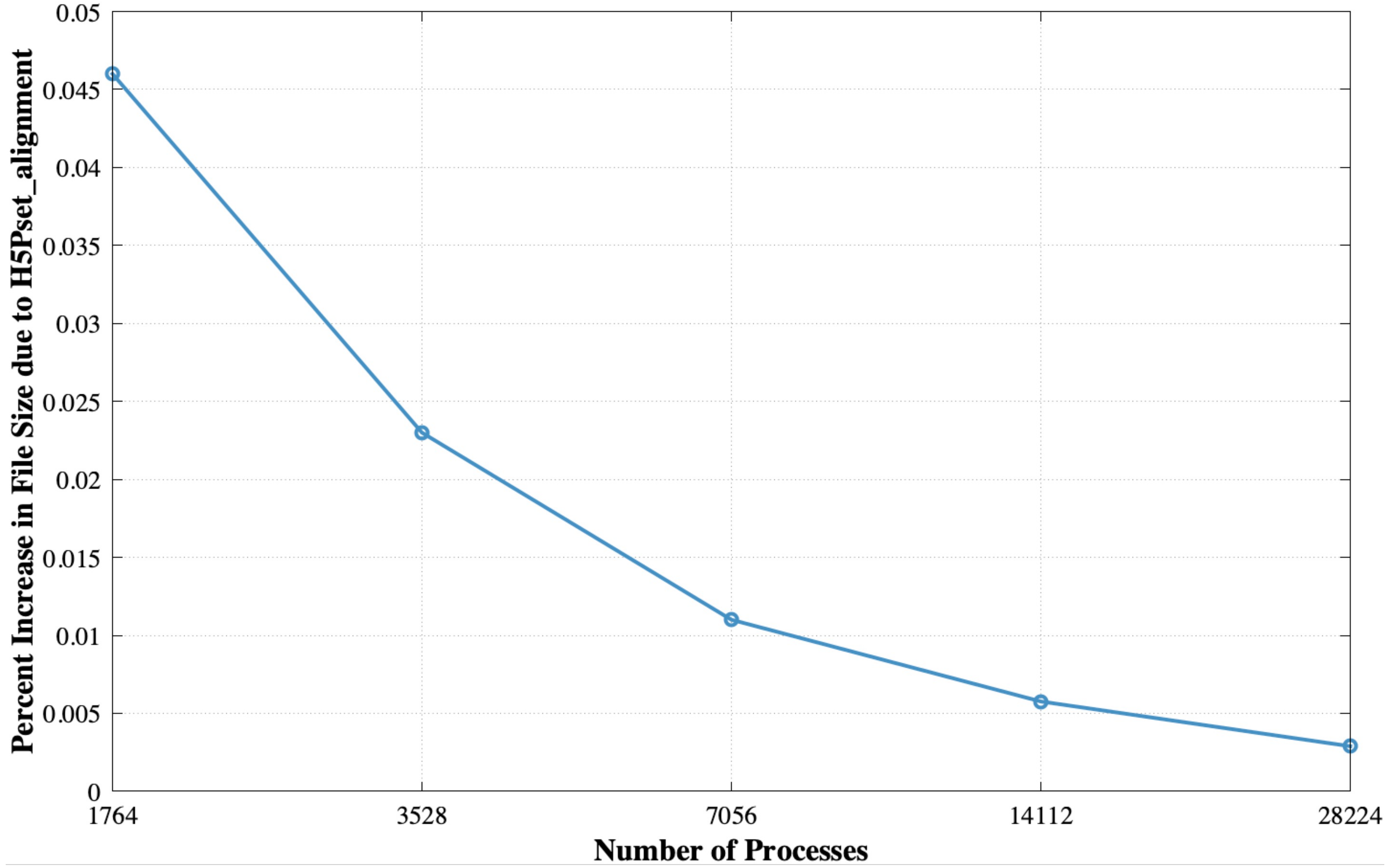
- Poor/Improved performance can be a result of FS changes
- Single shared file using MPI-IO performance degradation [Byna, NERSC].



Effects of influencing object's in the file layout

• H5Pset_alignment – controls the alignment of file objects on addresses.

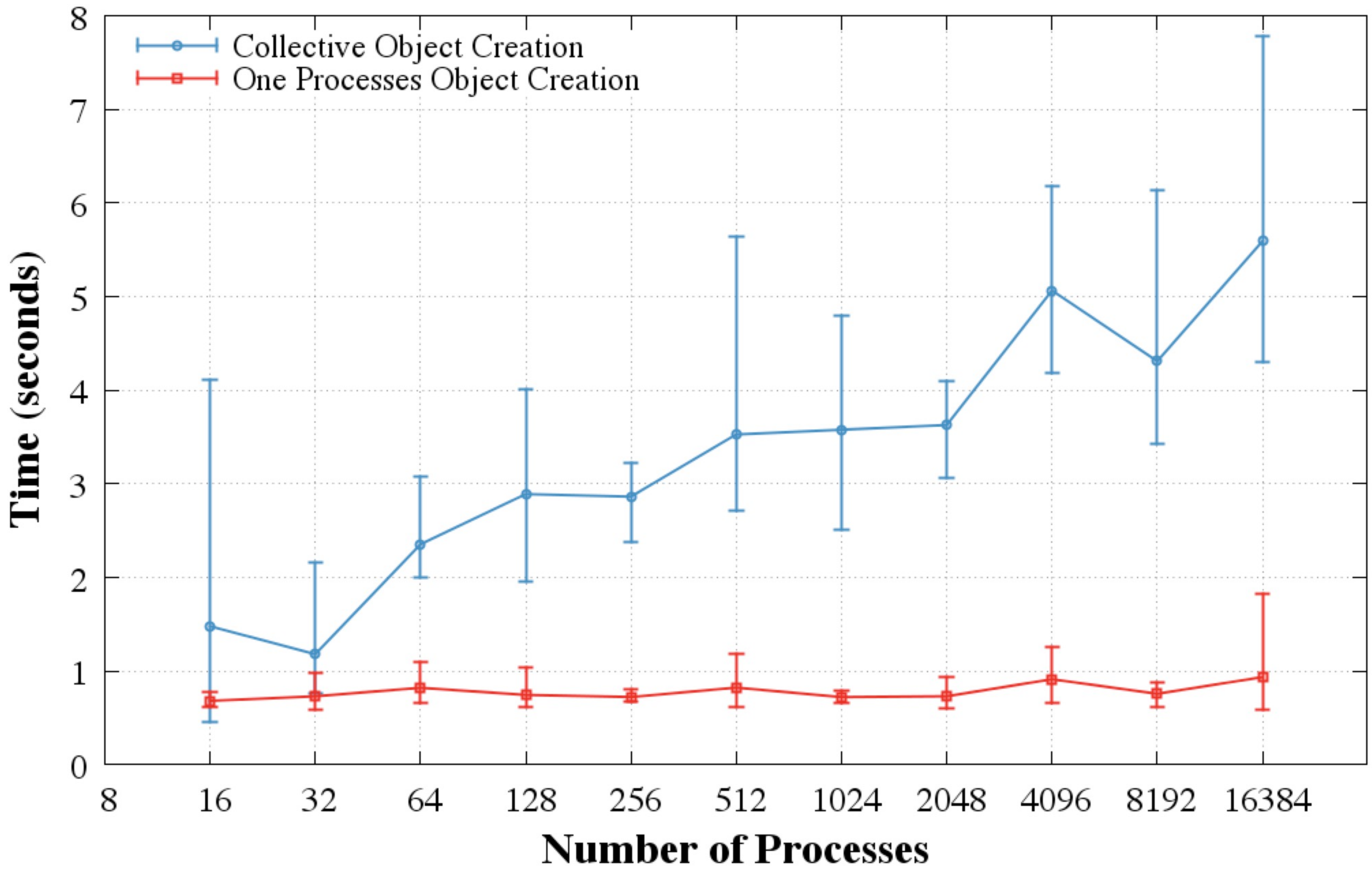




VPIC, Summit, ORNL

Object Creation (Collective vs. Single Process)

The HDF Group



CAUTION: Object Creation (Collective vs. Single Process)

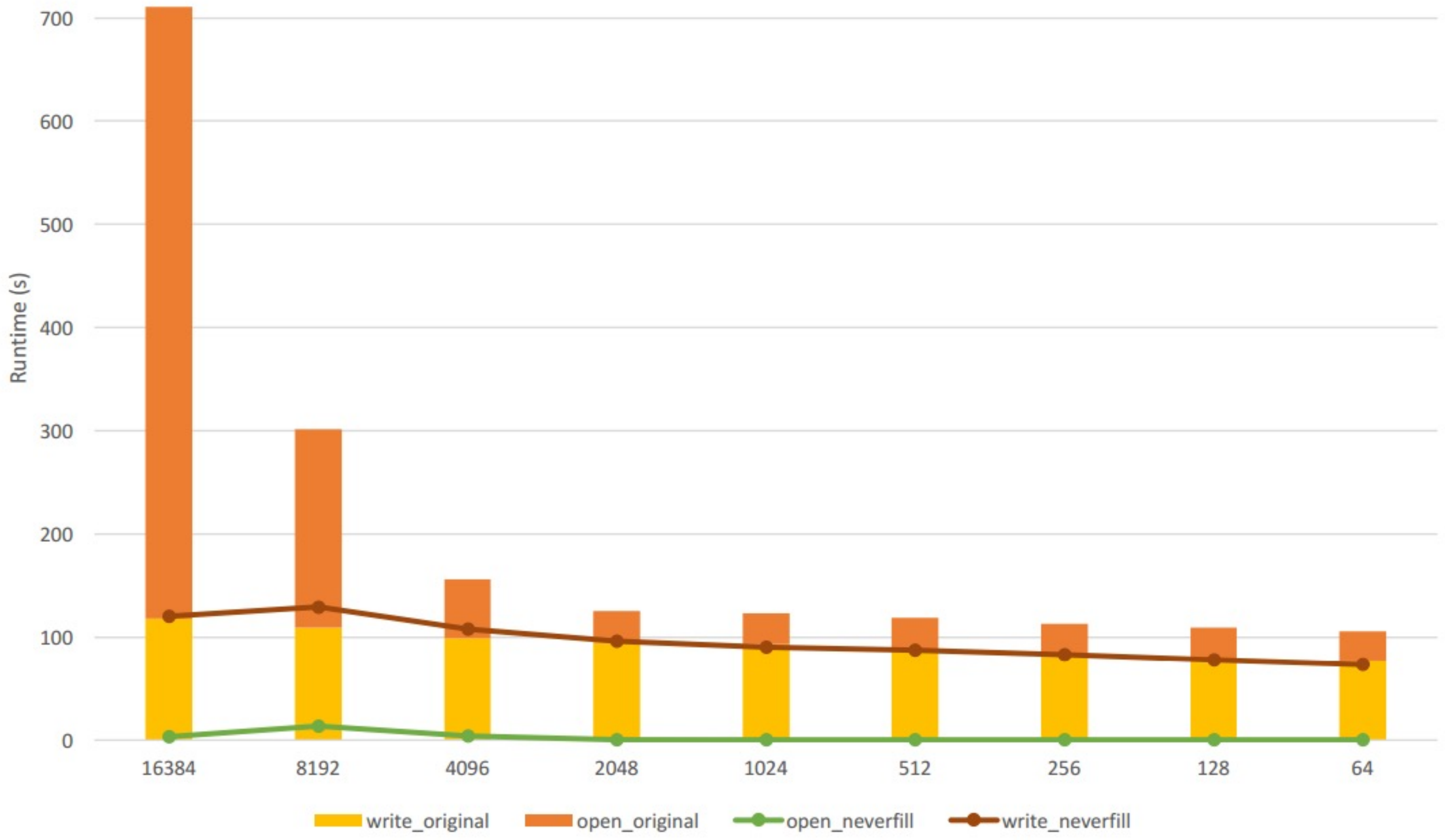
The HDF Group

- In sequential mode, HDF5 allocates chunks incrementally, i.e., when data is written to a chunk for the first time.
  - Chunk is also initialized with the default or user-provided fill value.
- In the parallel case, chunks are always allocated when the dataset is created (not incrementally).
  - The more ranks there are, the more chunks need to be allocated and initialized/written, resulting in a slowdown.

CAUTION: Object Creation (SEISM-IO, Blue Waters—NCSA)

The HDF Group

Set HDF5 to never fill chunks (H5Pset_fill_time with H5D_FILL_TIME_NEVER)



Challenging HDF5 Use Cases

- Ideally, HDF5 parallel performance should be comparable (or better) to raw binary I/O.
- Issues with third-party libraries (netCDF, CGNS…) using HDF5:
  - Can be metadata heavy due to the need to conform to a standard format.
  - The standard's format may dictate raw data output patterns.
    - May lead to optimal write performance but poor read performance, or vice-versa.
- Mitigating performance issues
  - Calls for HDF5 metadata can result in many small reads and writes.
  - Implement new features in HDF5 to address metadata performance
    - Collective metadata, using the core file driver for metadata creation, etc…
  - Work with third-party libraries to use parallel file system-friendly HDF5 schemes.

Improve the performance of reading/writing H5S_all selected datasets

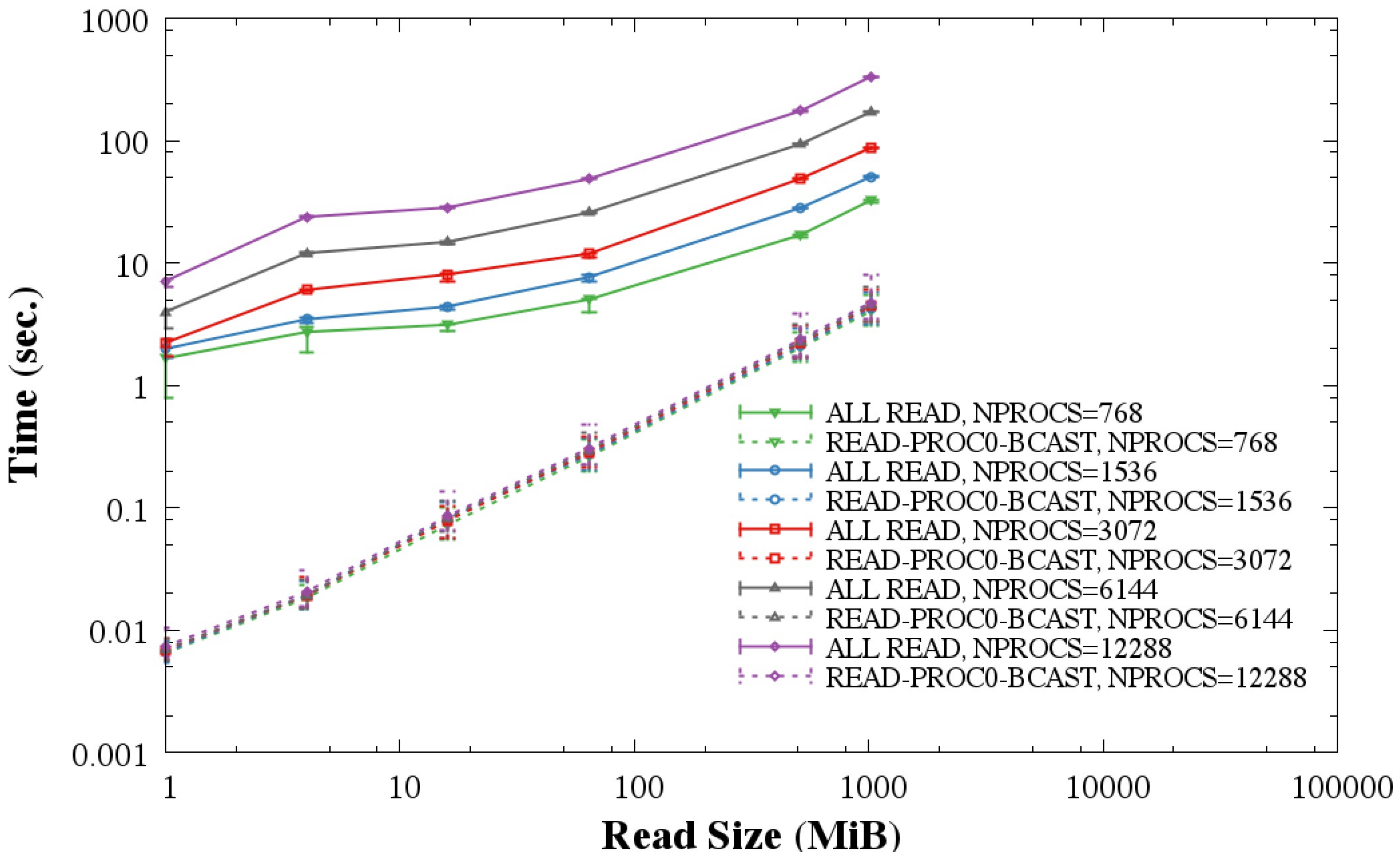
The HDF Group

(1) New in HDF5 1.10.5

- If:
  - All the processes are reading/writing the same data
  - And the dataset is less than 2GB
- Then
  - The lowest process id in the communicator will read and broadcast the data or write the data.

(2) Use of compact storage, or

- For compact storage, this same algorithm gets used.



HDF5 Dataset I/O

The HDF Group

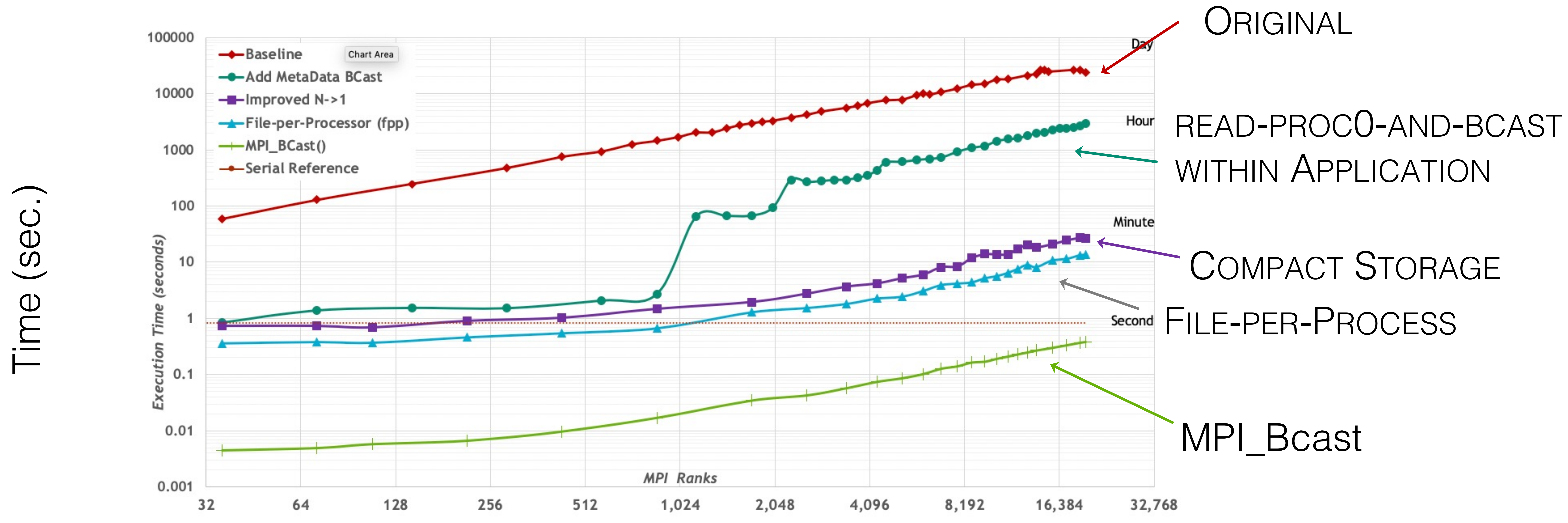
- Issue large I/O requests
  - At least as large as the file system block size
- Avoid **datatype conversion** ⓘ
  - Use the same data type in the file as in memory
- Avoid **dataspace conversion** ⓘ
  - One dimensional buffer in memory to two-dimensional array in the file

ⓘ Can break collective operations; check what mode was used H5Pget_mpio_actual_io_mode, and why H5Pget_mpio_no_collective_cause

HDF5 Dataset – Storage Type

- Use **contiguous storage** if no data will be added and compression is not used
  - Data will not be cached by HDF5
- Use **compact** storage when working with small data (<64K)
  - Data becomes part of HDF5 internal metadata and is cached (metadata cache)
- Avoid data duplication to reduce file sizes.
  - Use links to point to datasets stored in the same or external HDF5 file
  - Use VDS to point to data stored in other HDF5 datasets

SCALING OPTIMIZATIONS



Greg Sjaardema, Sandia National Labs

HDF5 Dataset – Chunked Storage

The HDF Group

- Chunking is required when using extendibility and/or compression and other filters
- **I/O** is always performed **on a whole chunk**
- Understand how **chunking cache** works https://portal.hdfgroup.org/display/HDF5/Chunking+in+HDF5 and consider
  - Do you access the same chunk often?
  - What is the best chunk size (especially when using compression)?

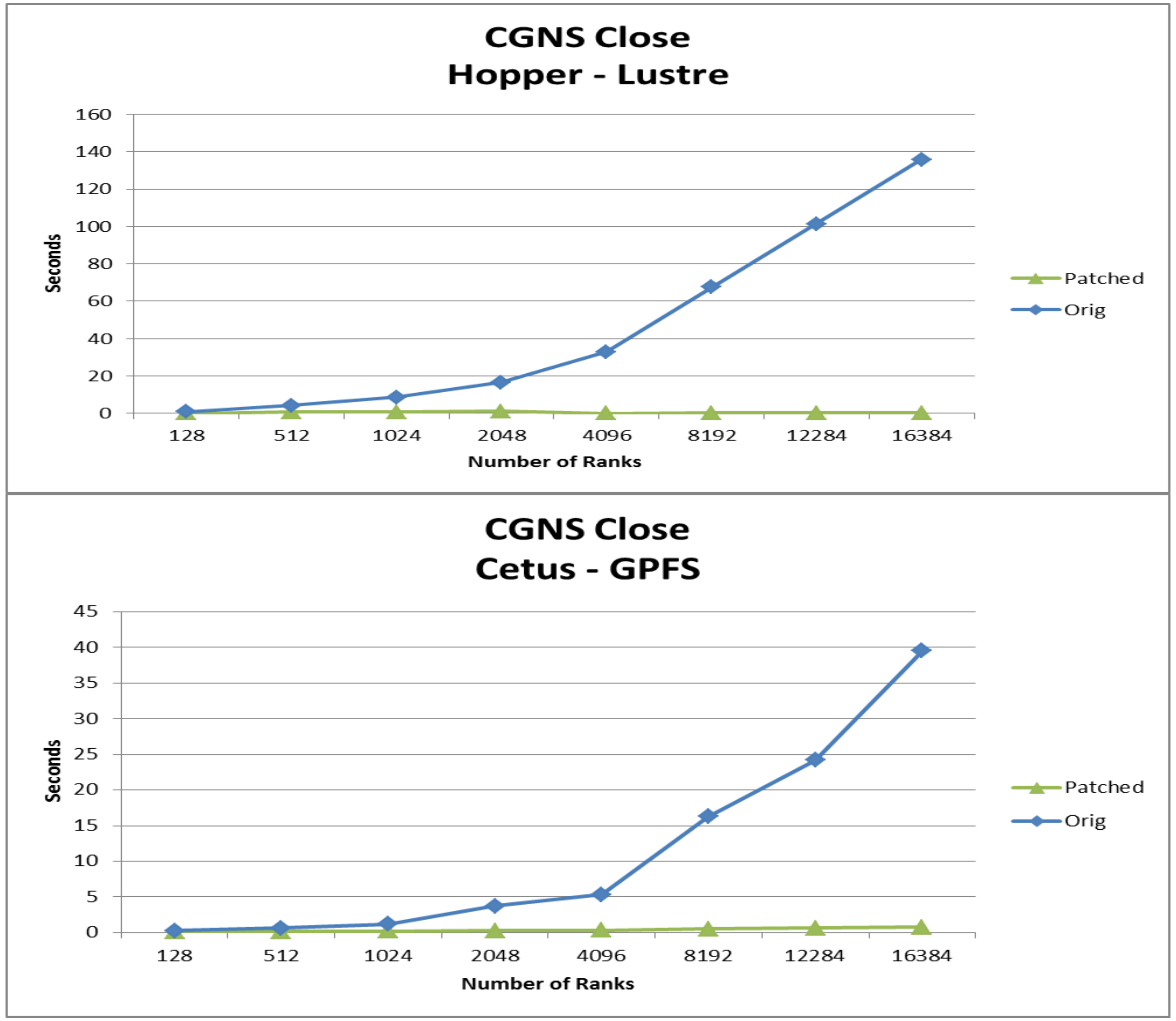
Write Metadata Collectively

The HDF Group

- **Symptoms:** Many users reported that `H5Fclose()` is very slow and doesn't scale well on parallel file systems.
- **Diagnosis:** HDF5 metadata cache issues very small accesses (one write per entry). We know that parallel file systems don't do well with small I/O accesses.
- **Solution:** Gather up all the entries of an epoch, create an MPI-derived datatype, and issue a single collective MPI write.

H5P_SET_COLL_METADATA_WRITE	Establishes I/O mode property setting, collective or independent, for metadata writes
H5P_GET_COLL_METADATA_WRITE	Retrieves I/O mode property setting for metadata writes
H5P_SET_ALL_COLL_METADATA_OPS	Establishes I/O mode, collective or independent, for metadata read operations
H5P_GET_ALL_COLL_METADATA_OPS	Retrieves I/O mode for metadata read operations

Closing a CGNS File …



New General HDF5 Best Practices Effecting Parallel Performance

HDF5 Fundamentals – A Simple Problem

The HDF Group

- Writing multiple 2D array variables over time:

ACROSS P processes arranged in a **R x C** process grid
    FOREACH step 1 .. **S**
     FOREACH count 1 .. **A**
        CREATE a double **ARRAY** of size **[X,Y]** | [R*X,C*Y] (Strong | Weak)
        (**WRITE** | **READ**) the **ARRAY** (**to** | **from**) an HDF5 file

Fundamentals – Missing Information

- How are the array variables represented in HDF5?
  - 2D, 3D, 4D datasets
  - Are the extents known a priori?
  - How are the dimensions ordered?
  - Groups?
- What order is the data written, and is the data read the same way?
- What's the storage layout?
  - How many physical files?
  - Contiguous or chunked, etc.
  - Is the data compressible?
- What's the file system or data store?
- Collective vs. independent MPI-IO

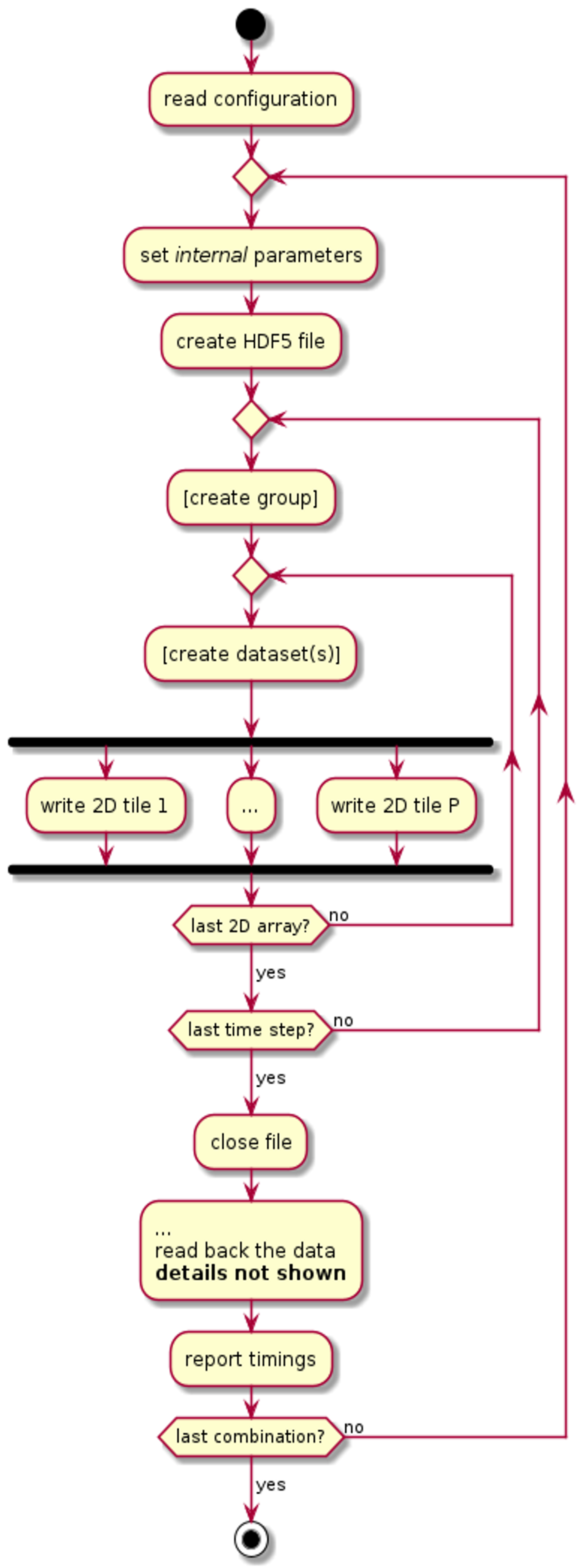
Other Sources of Performance Variability

- Hardware
- System configuration and activity of other users
- **<u>HDF5 property lists</u>**
  - Nearly 180 APIs
  - Controls storage properties for HDF5 objects
  - Controls in-flight HDF5 behavior
  - About 100 *`H5Pset_*`* functions
    - $\leq p_1 * \ldots * p_{100}$ combinations!
    - How many are tested?
  - What does *`H5P_DEFAULT`* mean?
    - (No, you can't control that one)
  - What is the effect of using `H5P_DEFAULT`?

https://portal.hdfgroup.org/display/HDF5/Property+Lists

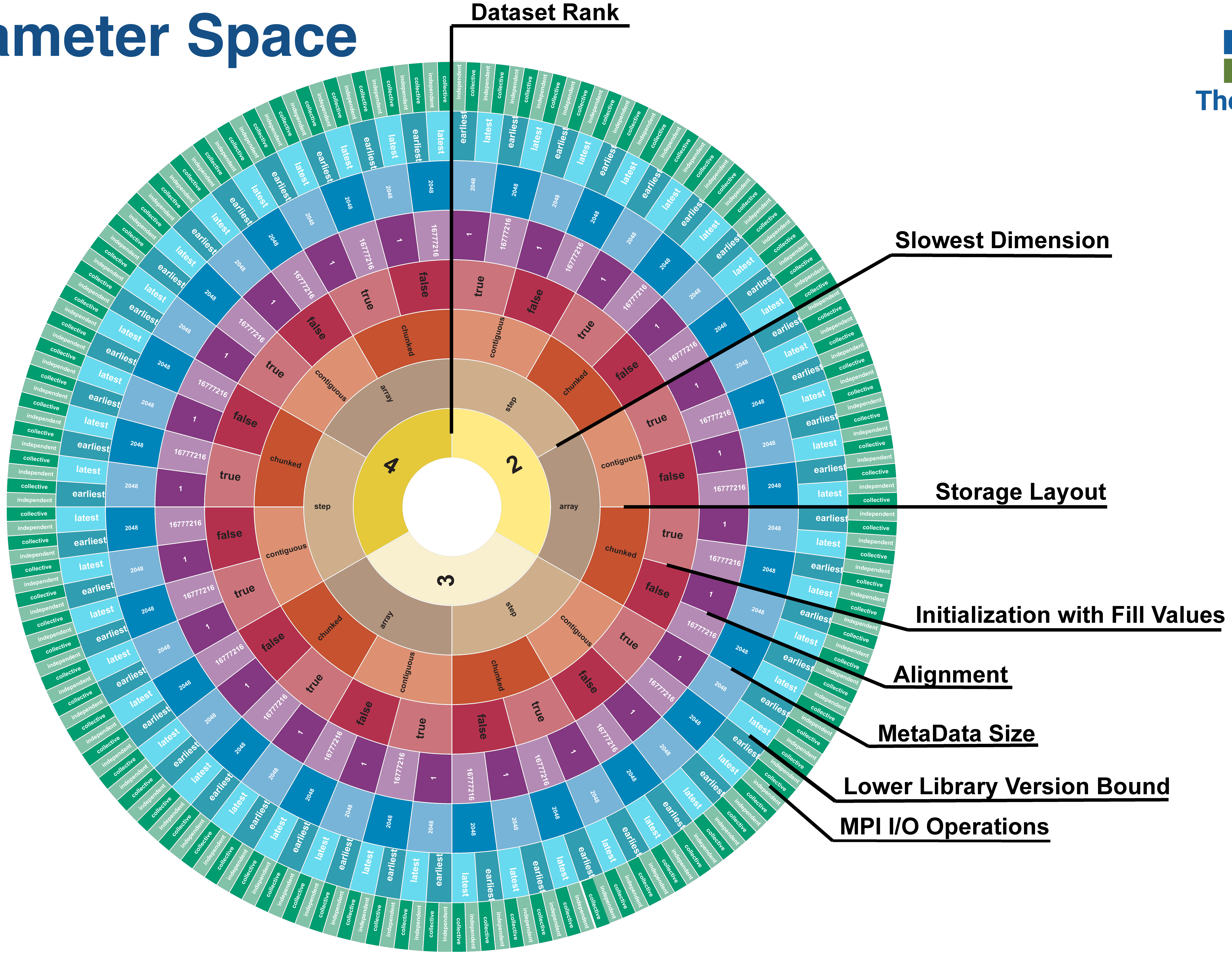
Back to the earlier example -- Application Model

- Good or bad news:
  - There are *several* different ways to handle the data in HDF5, for example:
    - Many 2D datasets or attributes
    - A few 3D datasets
    - A 4D dataset
  - There are many ways to use HDF5 properties
    - Chunking
    - Data alignment
    - Metadata block size
    - Collective/Independent I/O
  - Ideally, performance would be more or less the same
  - **HDF5 I/O**[1] test explores the HDF5 parameter space

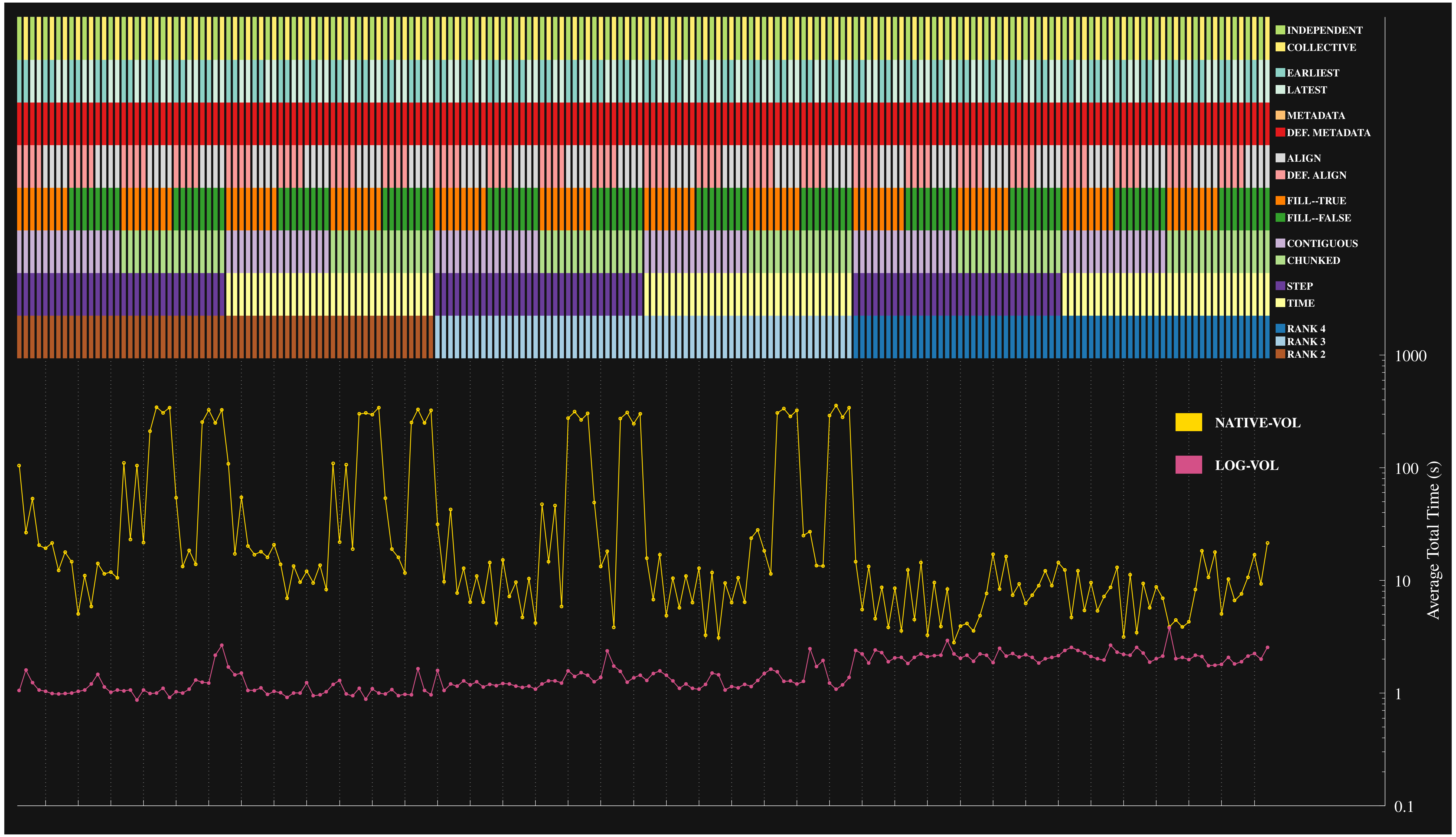


1 https://github.com/HDFGroup/hdf5-iotest

HDF5 Parameter Space

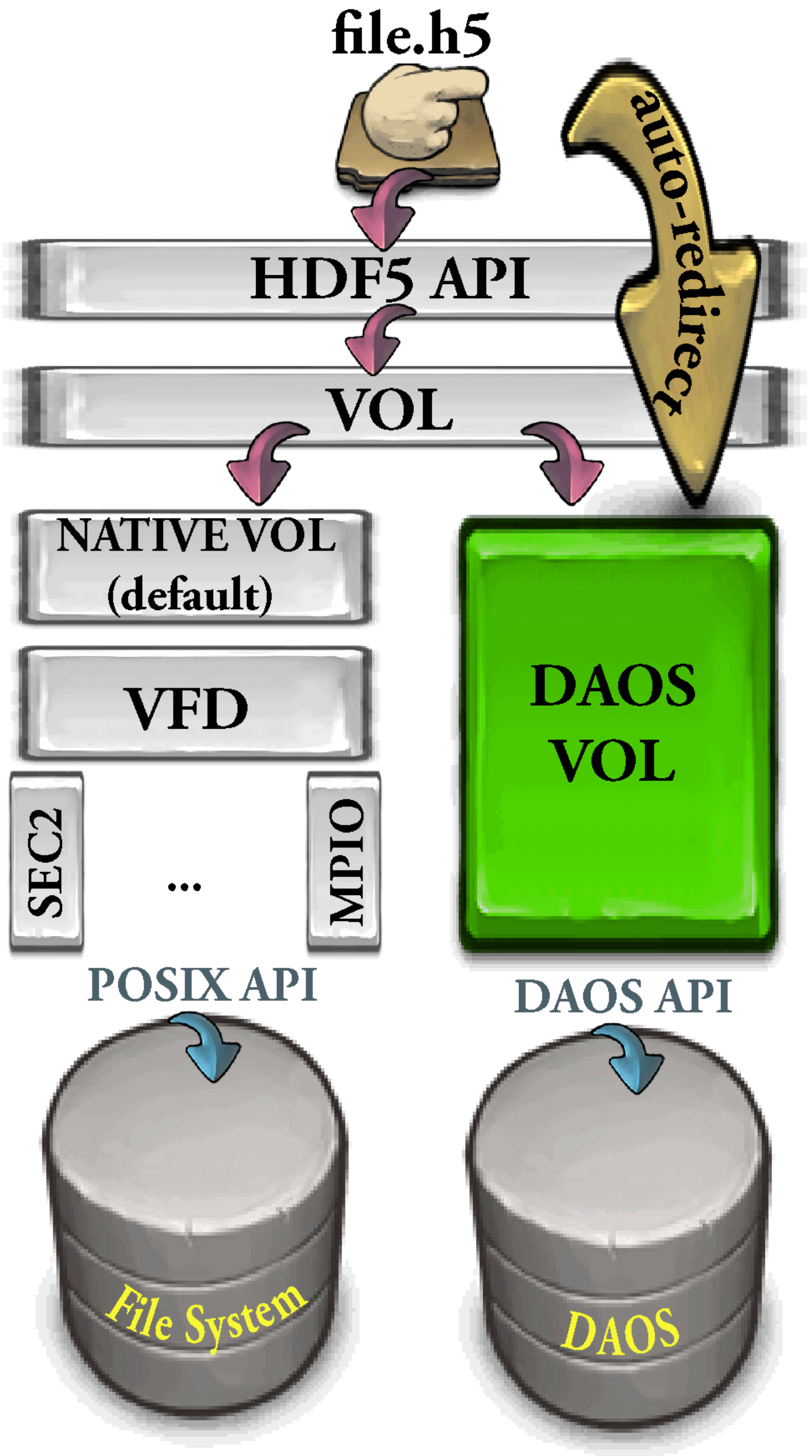


VOLs can help eliminate performance variability



Total time (read & write) in the HDFspace set for Cori on 512 ranks, LOG-BASED VOL

DAOS VOL Connector



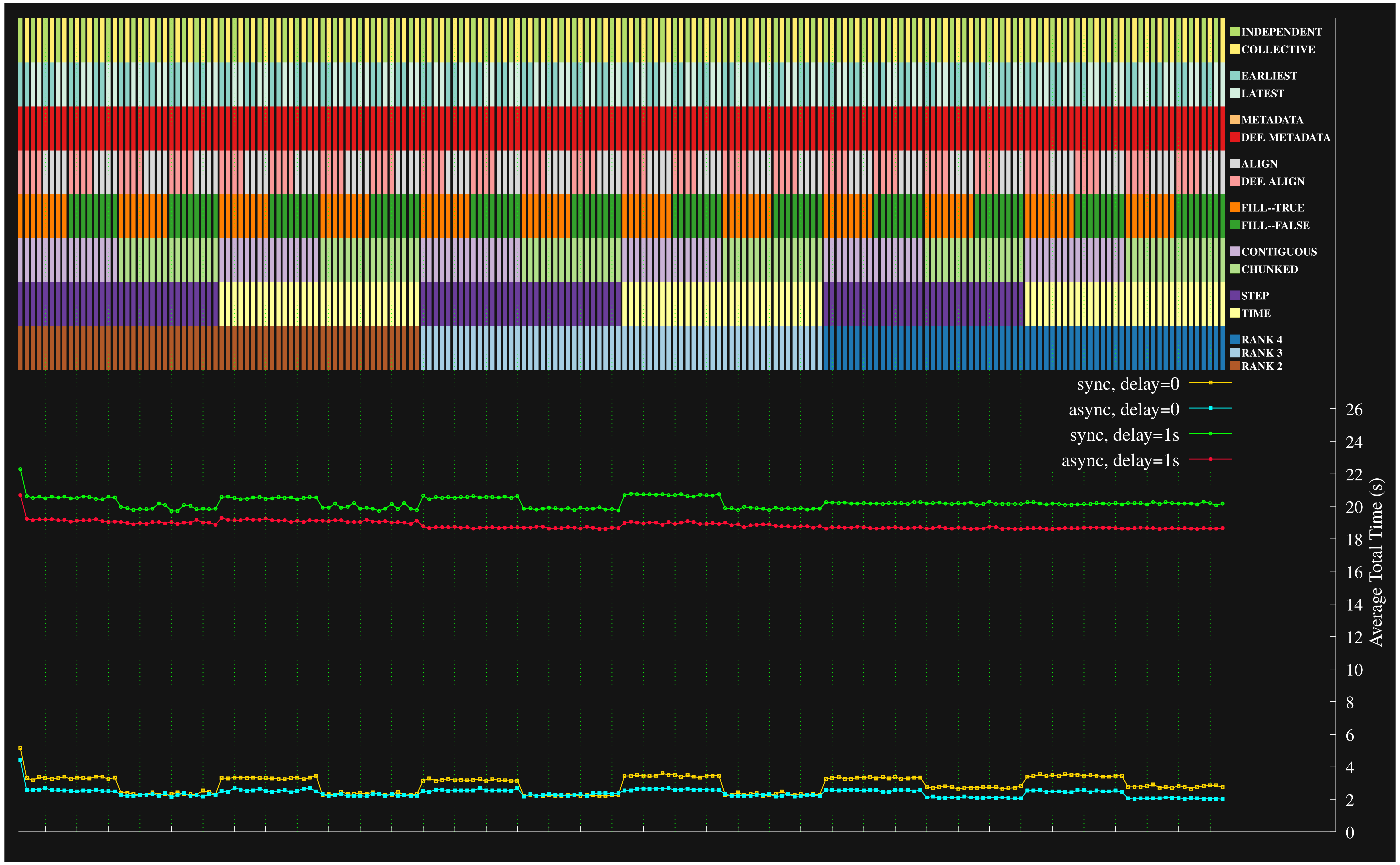
- HDF5 VOL connector for I/O to Distributed Asynchronous Object Storage (DAOS)

  https://github.com/HDFGroup/vol-daos

- Set to be deployed at ANL.
- Minimal code changes needed to use, enabled via environment variables or through HDF5 APIs.
- HDF5 tools are supported
  - h5dump, h5ls, h5diff, h5repack, h5copy, etc.
- Supports async I/O

VOLs can help eliminate performance variability

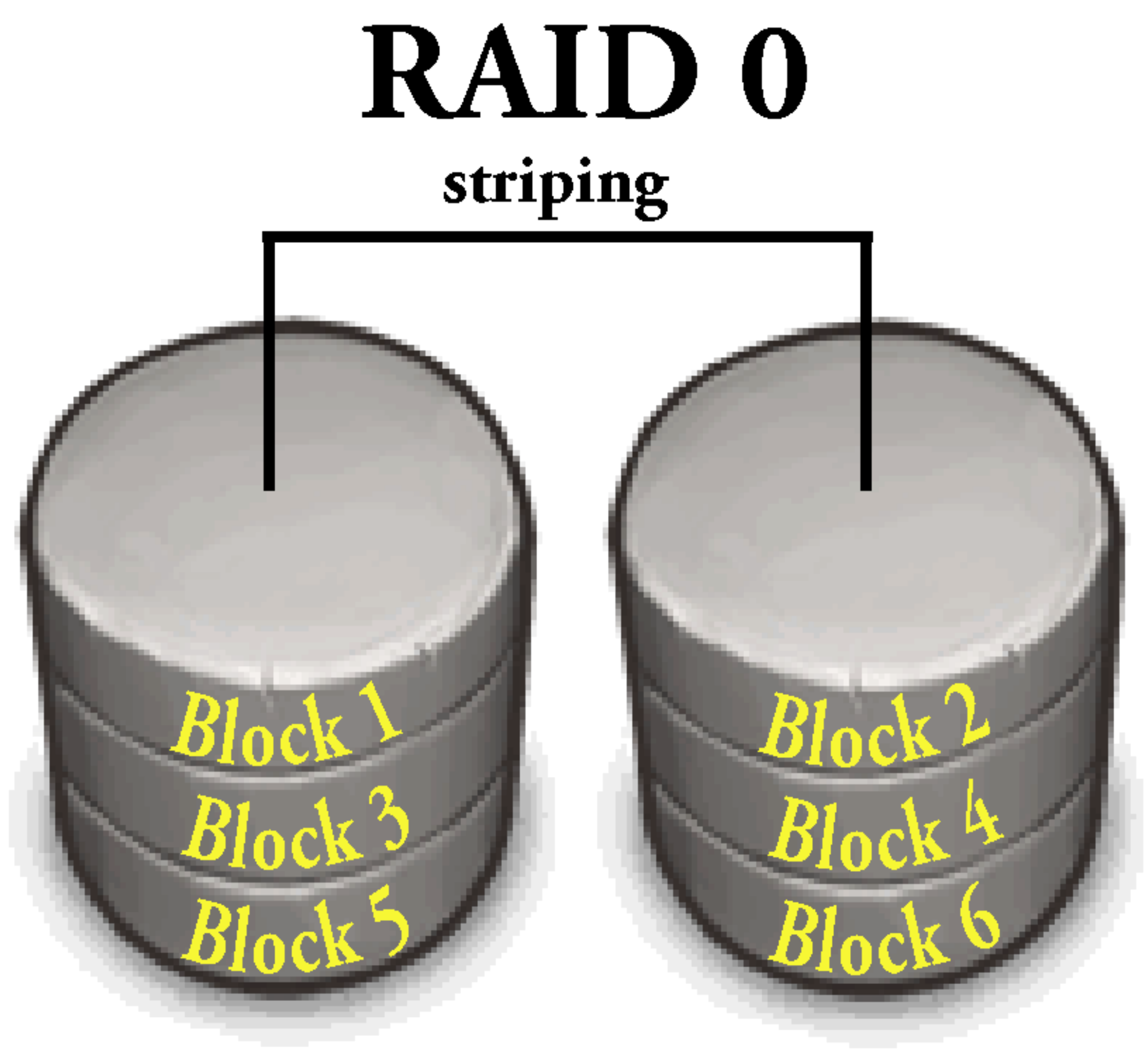
The HDF Group



Total time (read & write) in the HDFspace set for ANL 144 ranks, with no delay and one second delay for a compute phase, DAOS-VOL.

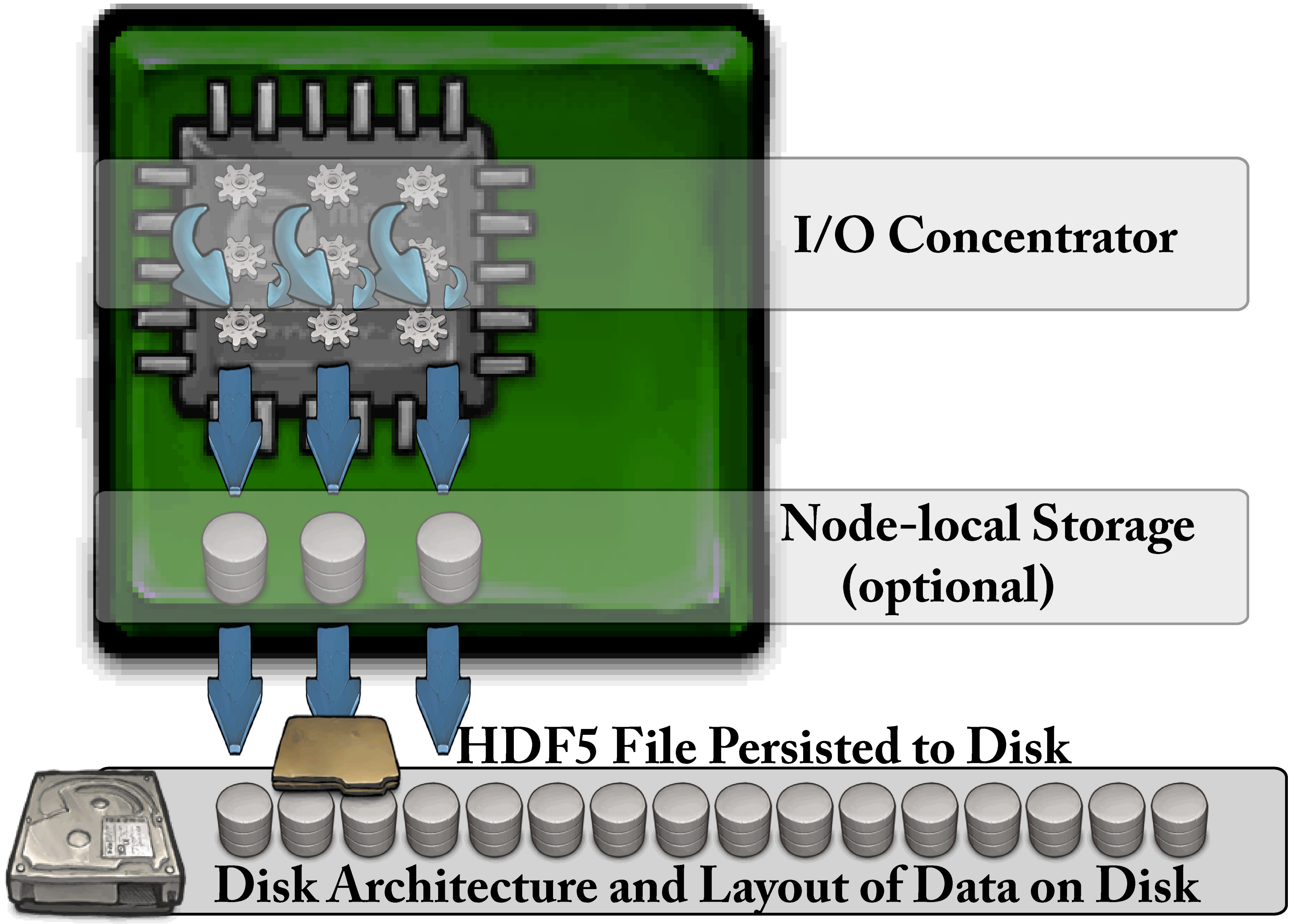
Subfiling

- Subfiling is a compromise between file-per-process (*fpp*) and a single shared file (*ssf*)
  - Multiple files organized as a Software RAID-0 Implementation
    1. Configurable "stripe-depth" and "stripe-set size"
    2. A default "stripe-set" is created by using 1 file per node
    3. A default "stripe-depth" is 32MB
  - One metadata (.h5) file *stitching* the small files together in the current implementation



- Benefits
  - Better use of parallel I/O subsystem
  - Reduces the complexity of *fpp*
  - Reduced locking and contention issues to improve performance at larger processor counts over *ssf*

Subfiling



For Subfiling, the HDF5 content is separated into two components:

1. **The Metadata –** the metadata is embedded in subfiles.
2. **The RAW data –** is written logically to a RAID-0 file and is spread over several individual files, each managed by an I/O concentrator.

The resulting collection can be read using Sub-filing or eventually coalesced via a post-processing step into a single HDF5 file (***h5fuse.sh***).

a. I/O Concentrators are implemented as independent threads attached to a normal HDF5 process.
b. MPI is utilized for communicating between HDF5 processes and the set of I/O Concentrators.
c. Because of (b), applications need to use *MPI_Init_thread* with *MPI_THREAD_MULTIPLE* to initialize the MPI library.

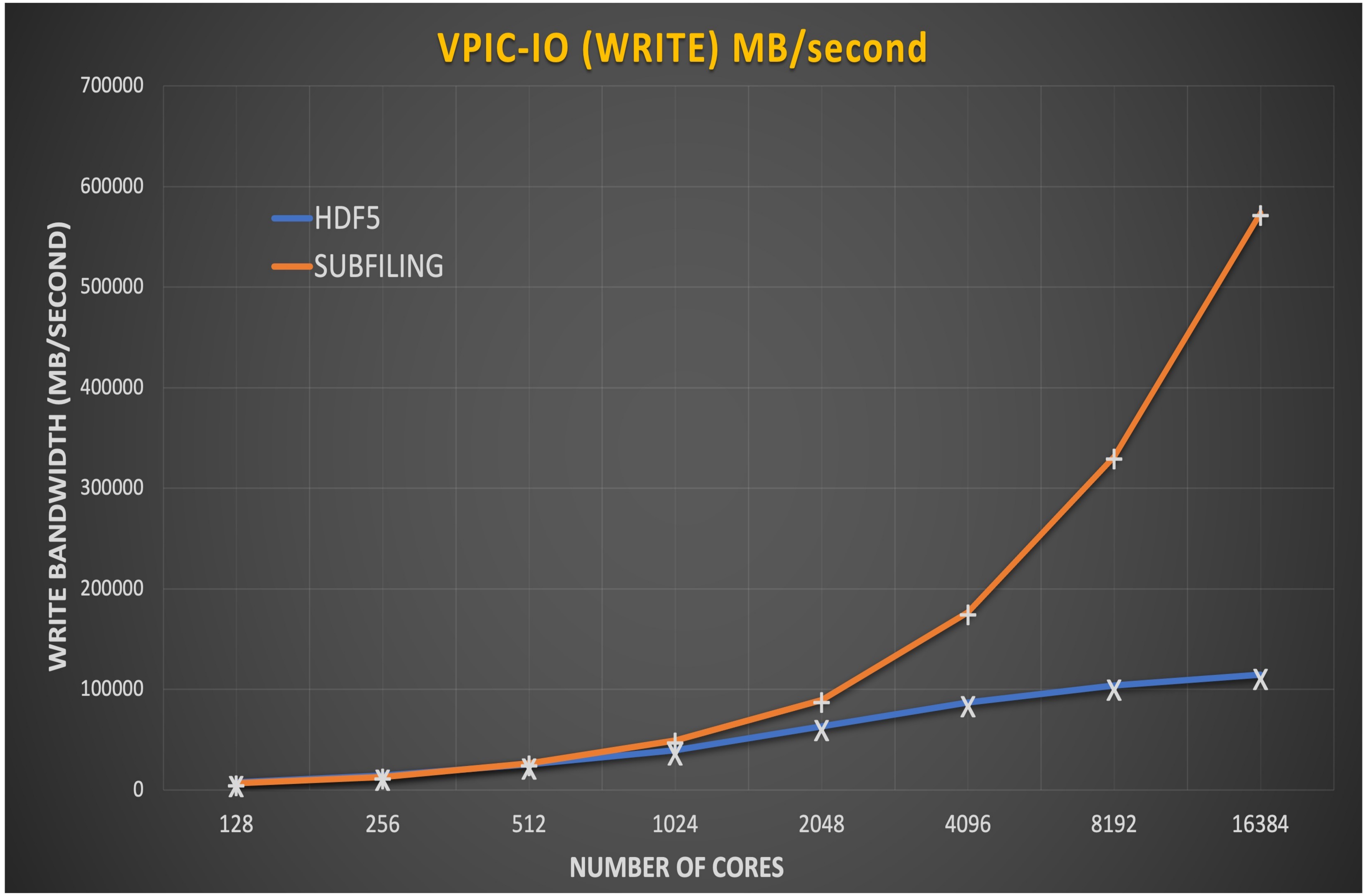
Subfiling

The HDF Group

Initial Results

(**h5bench – vpicio**)

- Parallel runs on *SUMMIT* show results from 256 to 16384 cores.
- The number of *Subfiles* utilized ranges from **6** (for a 256 MPI rank application run) to **391** (for the 16K MPI rank application), based on 42 cores per node.



THANK YOU!

Questions & Comments?