CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 10: Parallel I/O Performance

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

02/09/2023

Today's class

- Any questions?

- Class presentation topic

- Today's class –
  - Parallel I/O performance

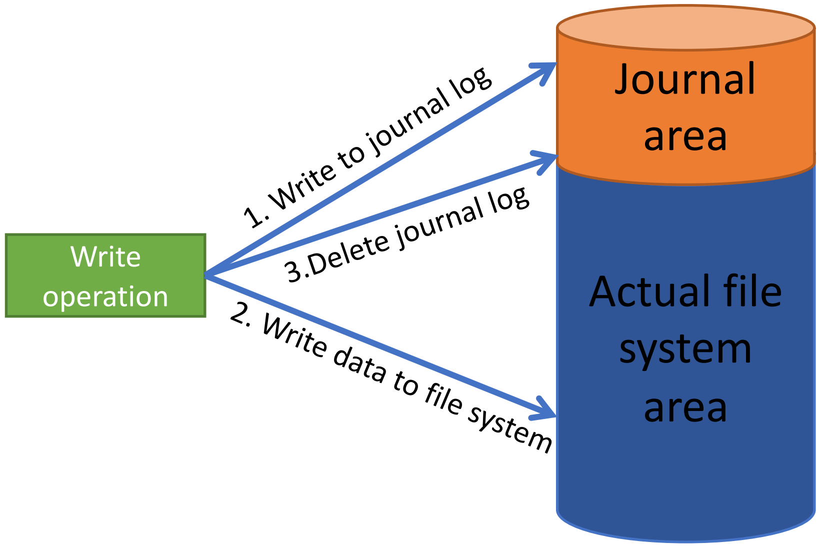
File system – Fault tolerance requirements

- In computing systems, crashes happen
  - Power outage, software bugs, hardware bugs, etc.
- Maintaining consistency in case of a failure is crucial

- Fault tolerance
  - Maintain functionality and data structures consistent
  - Example: If a file system is in the middle of a write operation, it must record it and recover it when the fault is resolved
- Two approaches
  - Journaling
  - Log-structured

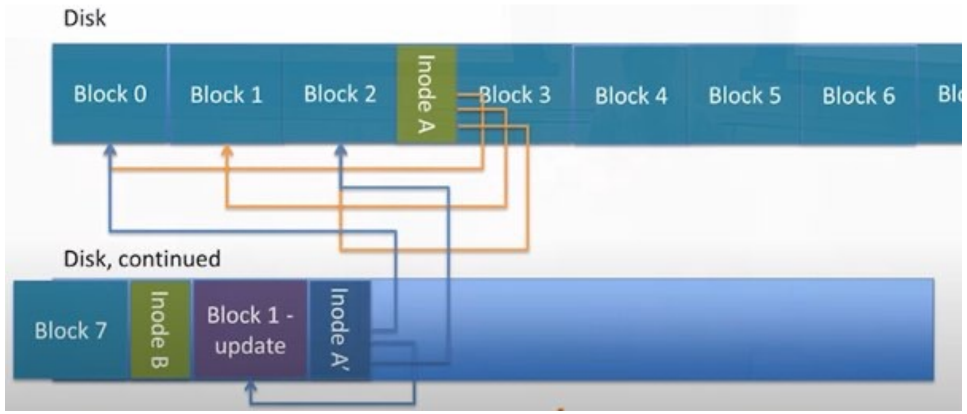
Journaling File System

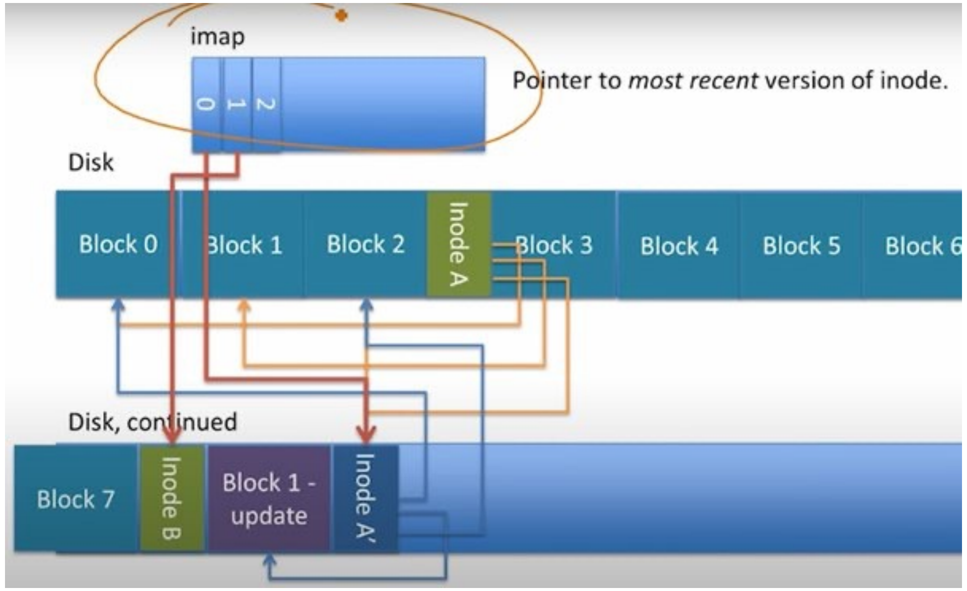
- A journal to keep track of uncommitted file system operations
- A separate data structure is used for keeping track of records – Journal



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





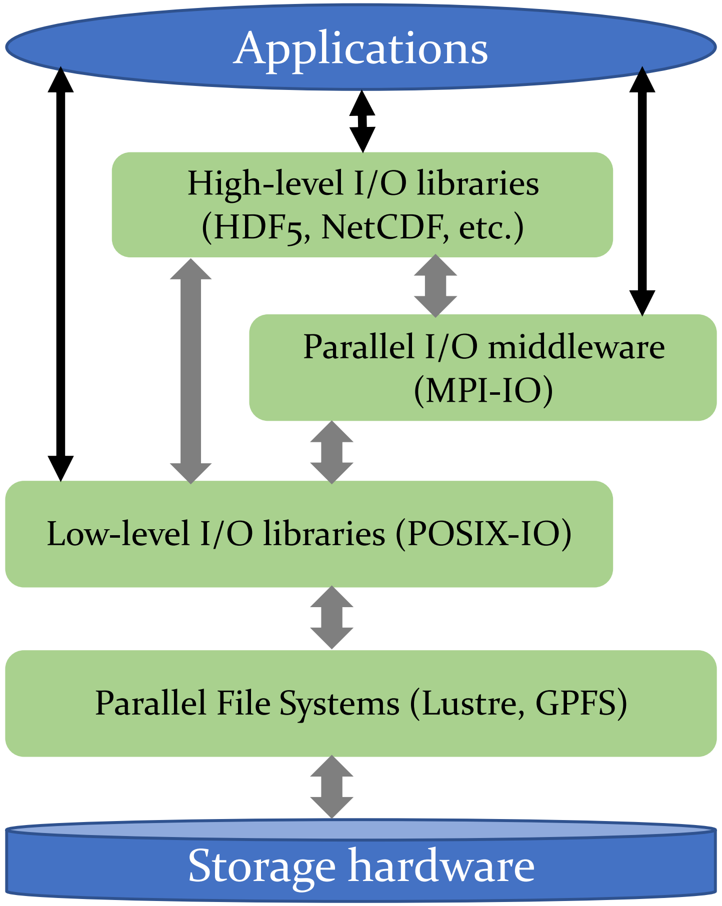
More details: https://pages.cs.wisc.edu/~remzi/OSTEP/file-lfs.pdf

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

Parallel I/O performance

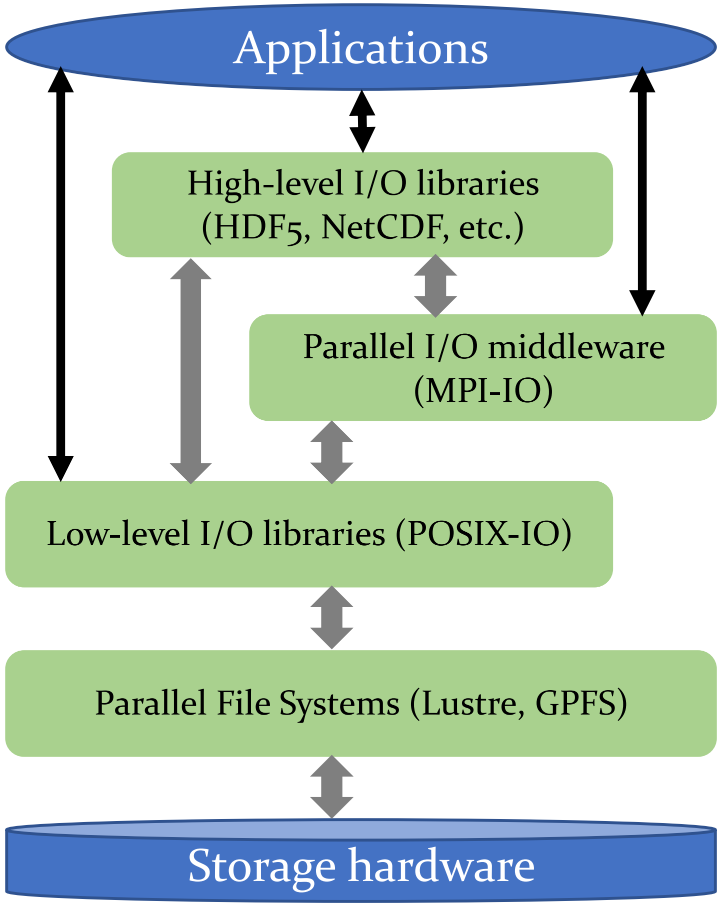
Parallel I/O performance – Factors that impact from Application level

- 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
- …



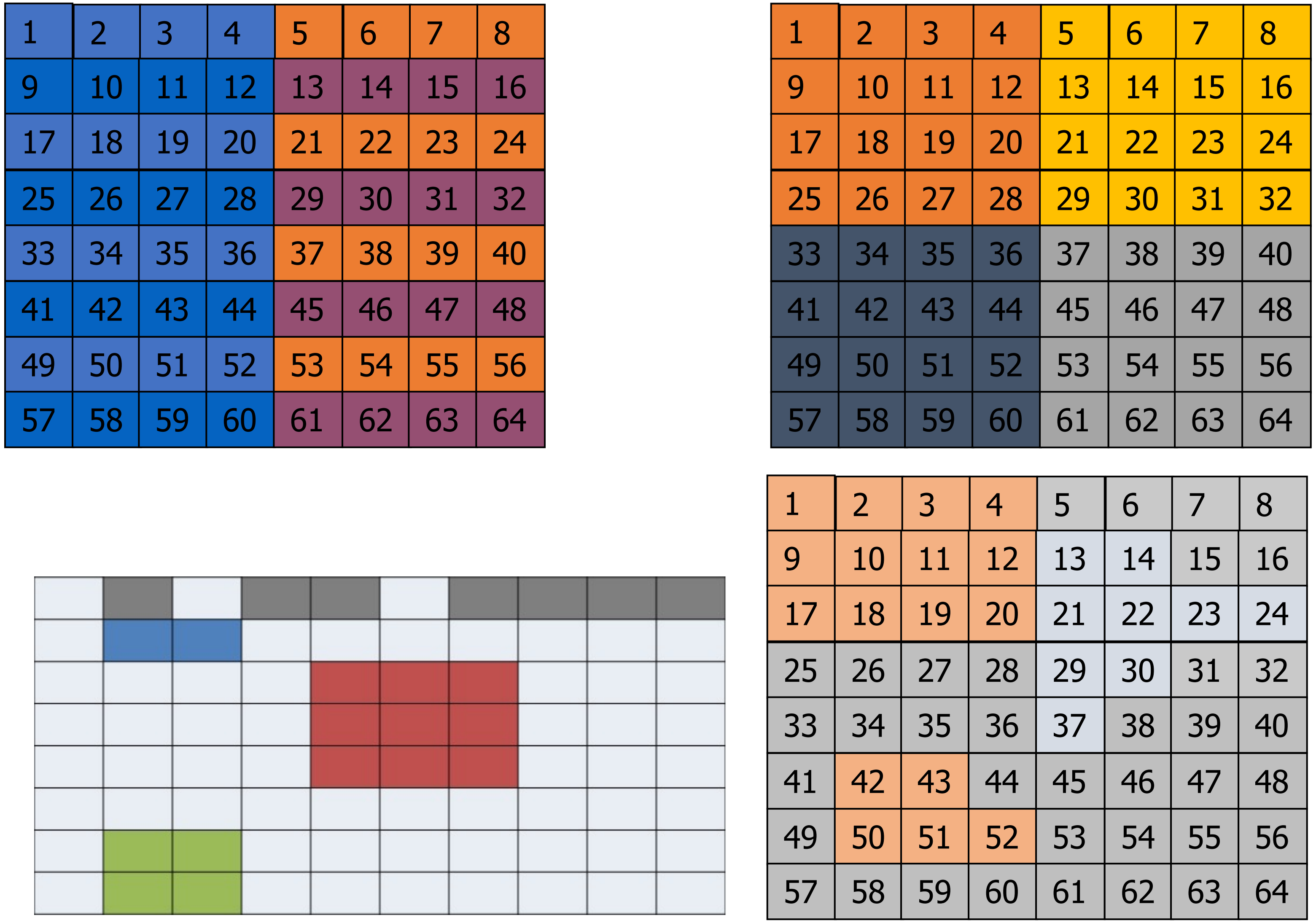
Parallel I/O performance – Factors that impact – HL I/O library

- 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?
  - File need to be exact size or can it have some gaps?
  - Cache metadata or not?

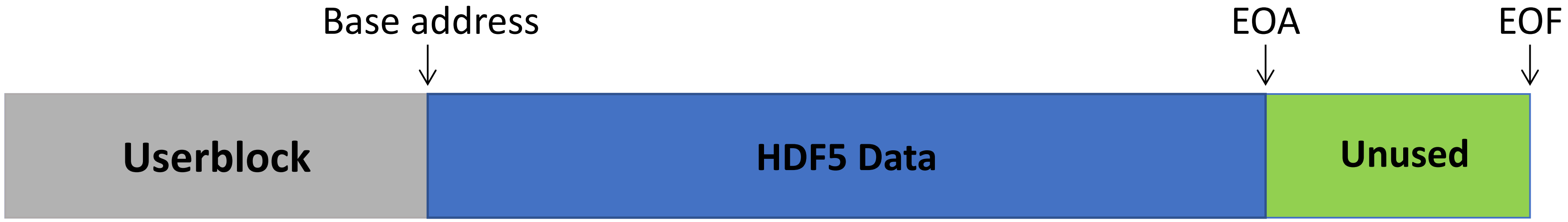


Hyperslabs can be complex

- HDF5 doesn't have restrictions on data patterns and data balance
- Internally, the HDF5 library creates an MPI datatype for each lower dimension in the selection and then combines those types into one giant structured MPI datatype

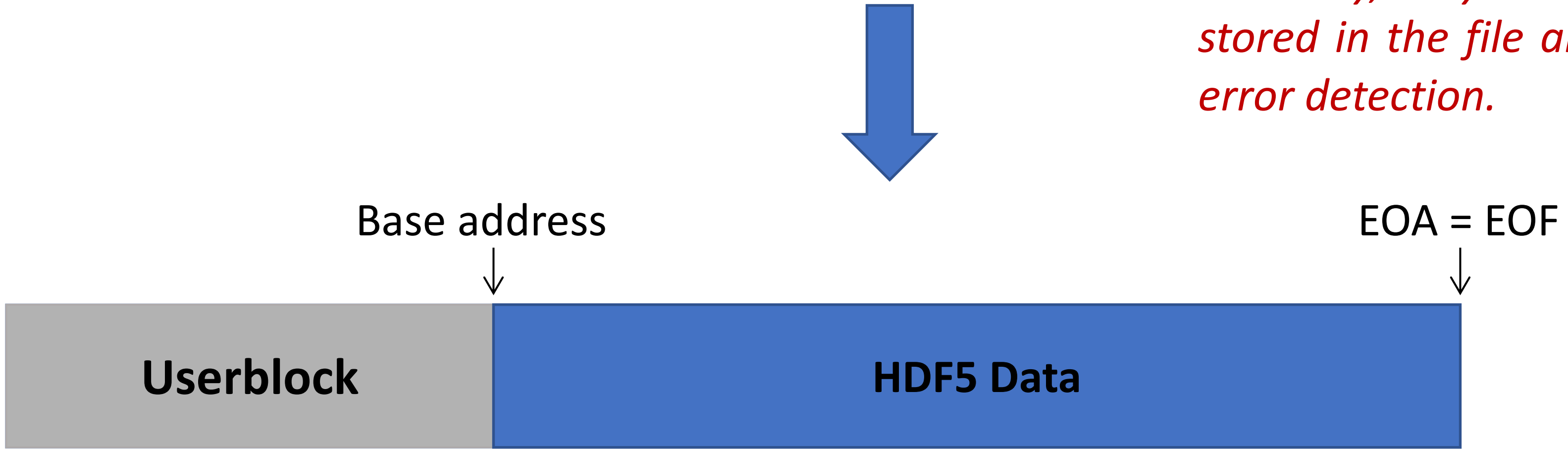


File Truncation Needed or not?



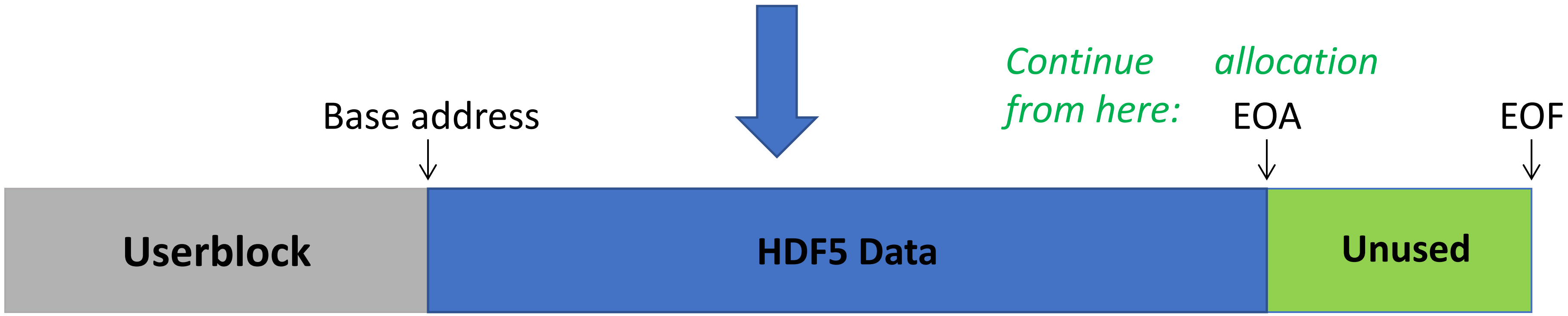
A call to `H5Fflush` or `H5Fclose` triggers a call to `ftruncate` (serial) or `MPI_File_set_size` (parallel), which can be fairly expensive.

*Currently, only **one** number is stored in the file and used for error detection.*



File Truncation Needed or not?

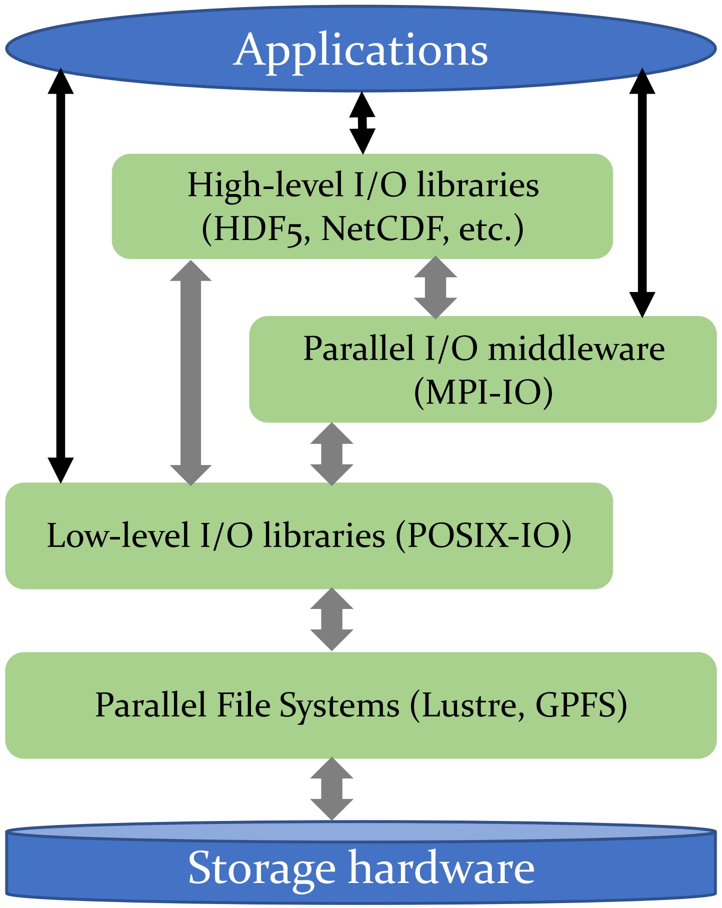
A call to `H5Fflush` or `H5Fclose` triggers both values (EOA, EOF) to be saved in the file and **no** truncation takes place, IF the file was created with the “avoid truncation” property set.



Caveat: Incompatible with older versions of the library. Requires HDF5 library version 1.12 or later.

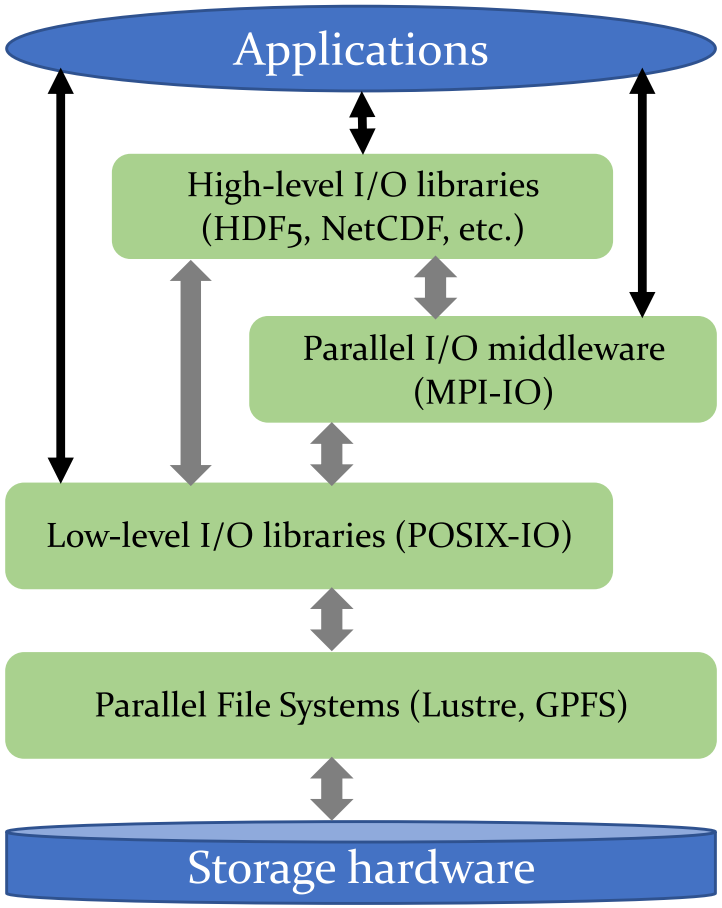
Parallel I/O performance – Factors that impact – MPI-IO layer

- 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



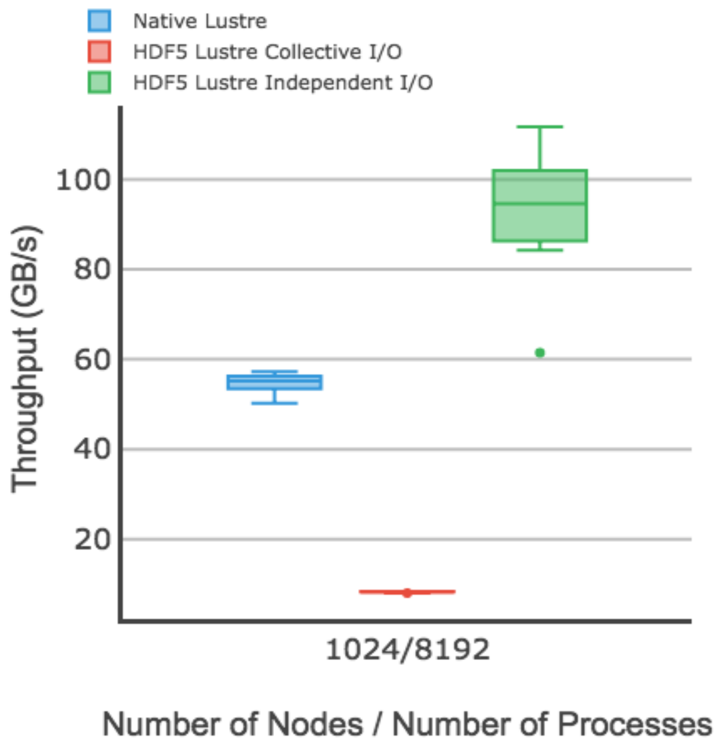
Parallel I/O performance – Factors that impact – Parallel file system

- 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



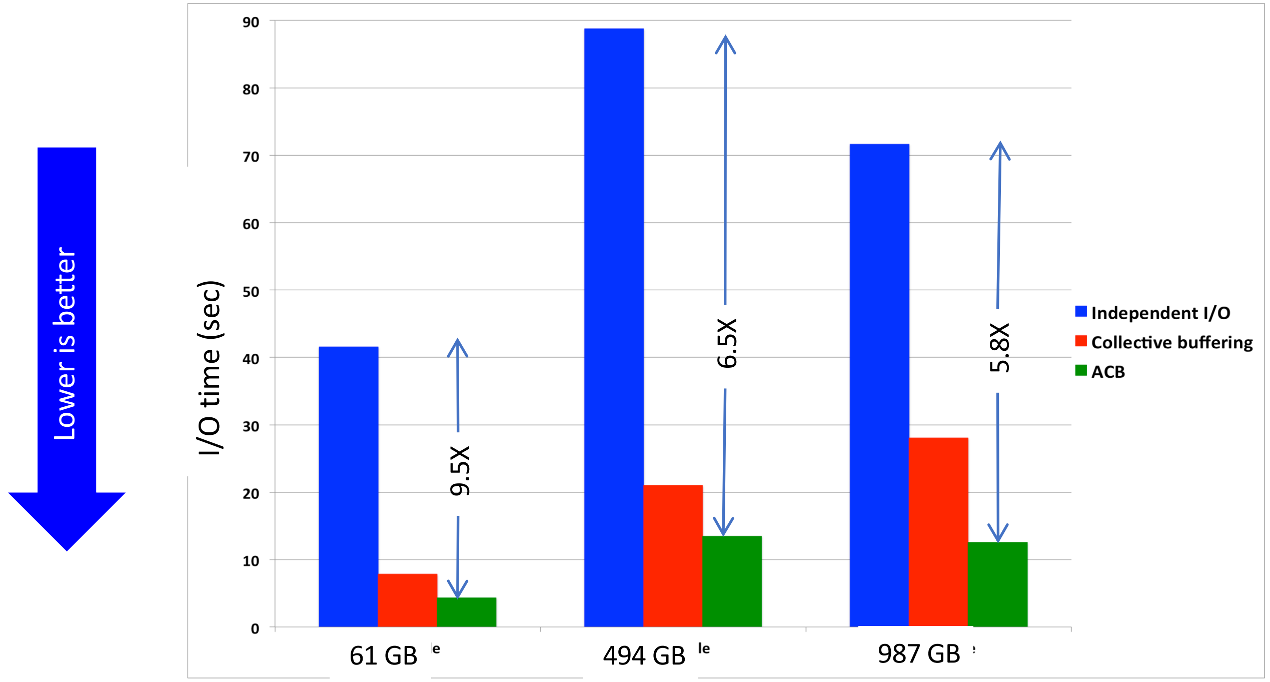
Parallel I/O performance optimization – Use cases

AMReX I/O
(collective vs. independent)



Native: File per node

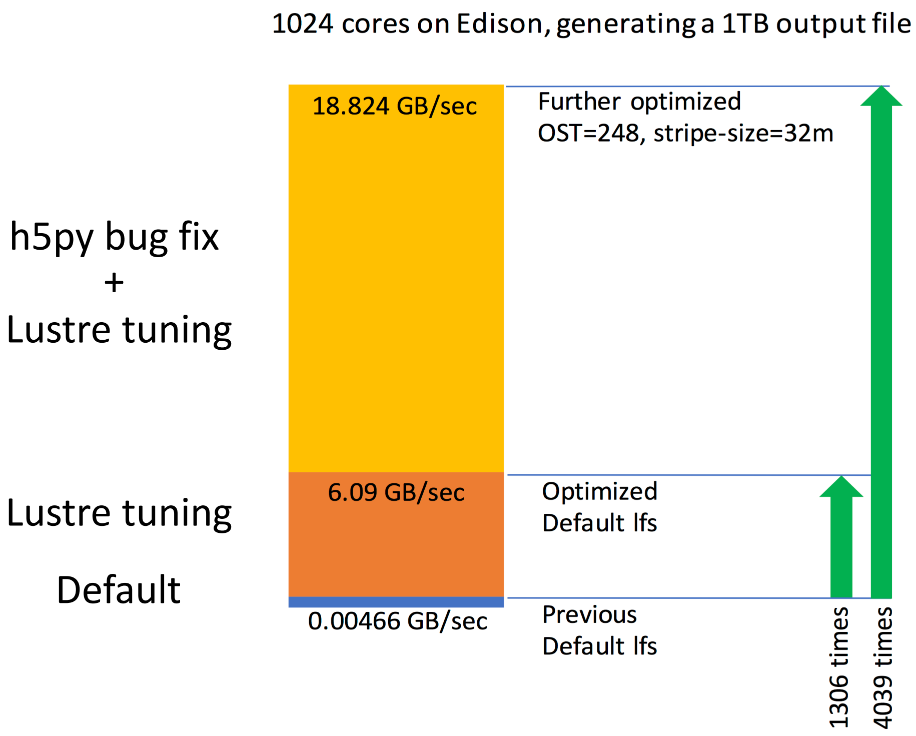
Chombo I/O
(collective vs. independent)



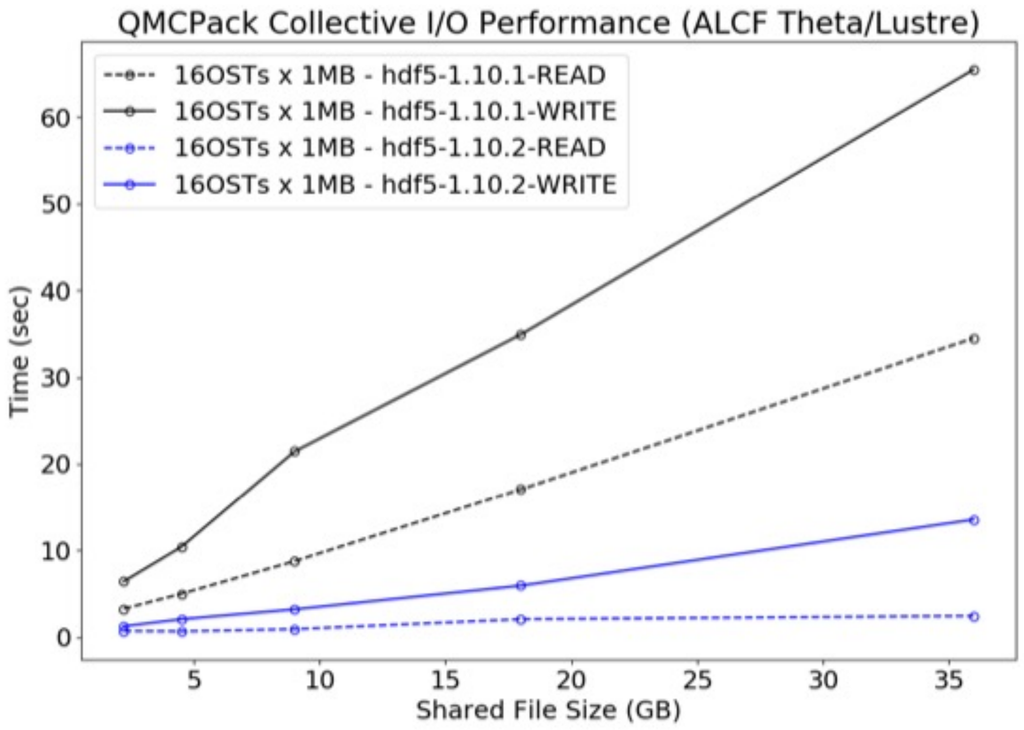
ACB: Aggregated Collective Buffering

- Three-phase I/O
  - Application + MPI-IO two-phase I/O

WarpX (Tuning Lustre)



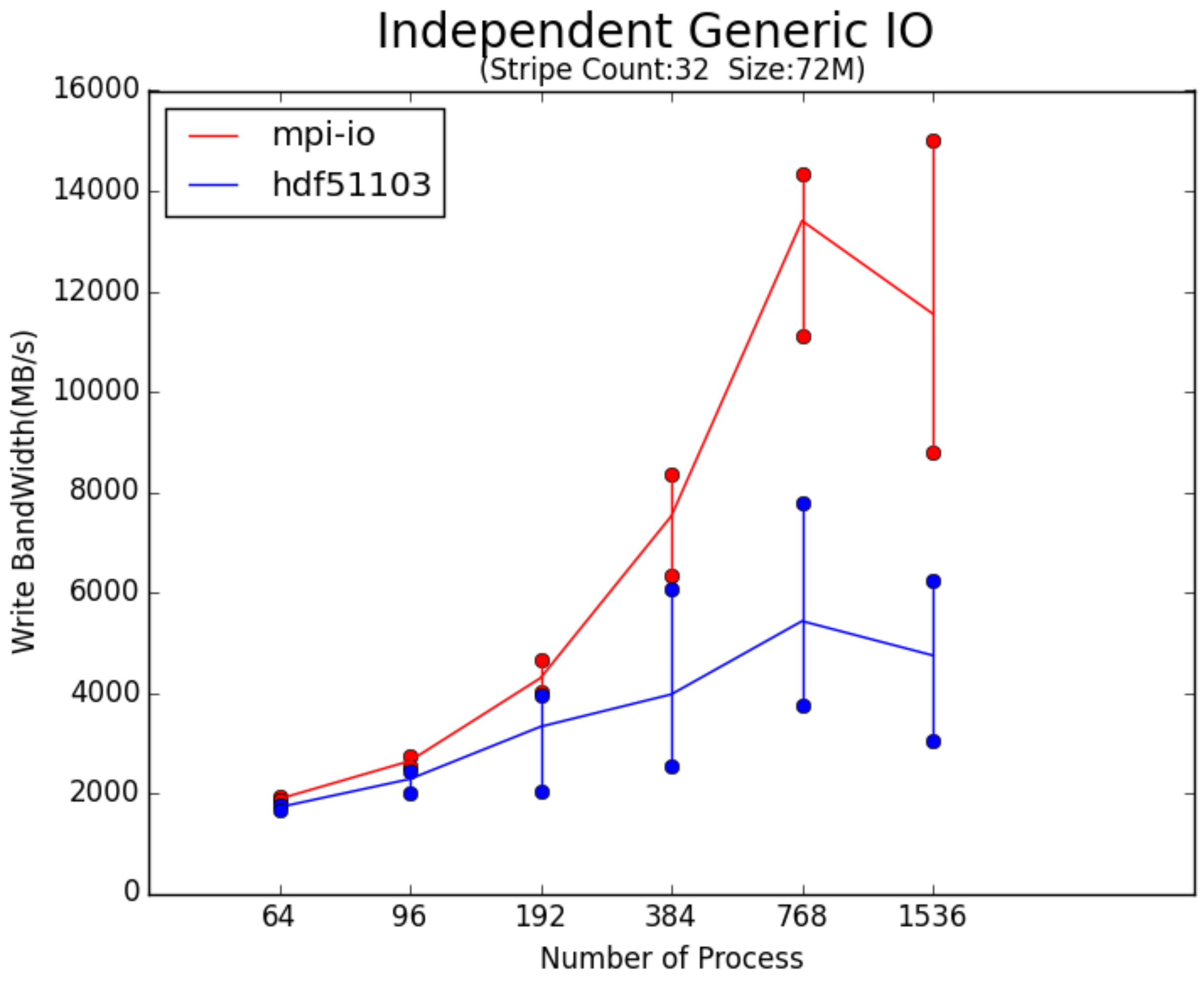
QMCPACK
(HDF5 version dependences, best to use newest version)



HACC – Data layout

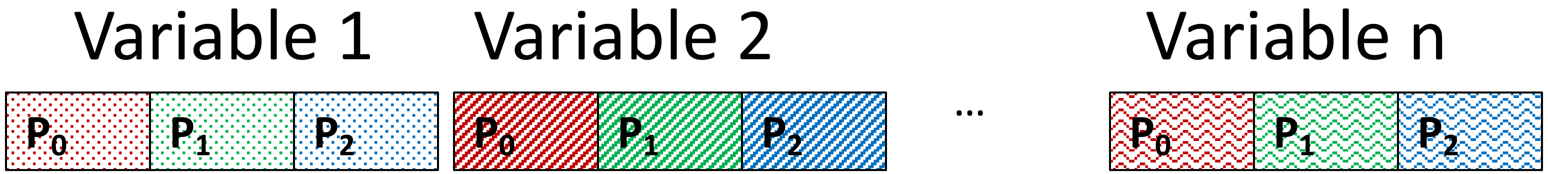
Benchmark:

- 9 1-D variables with the same number of elements (~1e9).
- Total file size is about 40GB.
- Can switch between writing with MPI-IO or HDF5.
- Used independent IO for write.

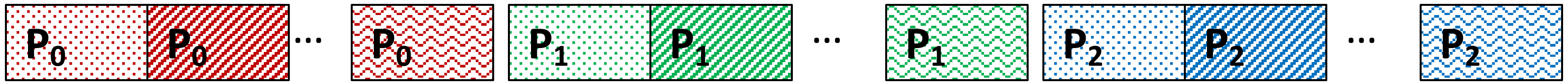


HACC Write Patterns

Pattern 1 – General HDF5 pattern

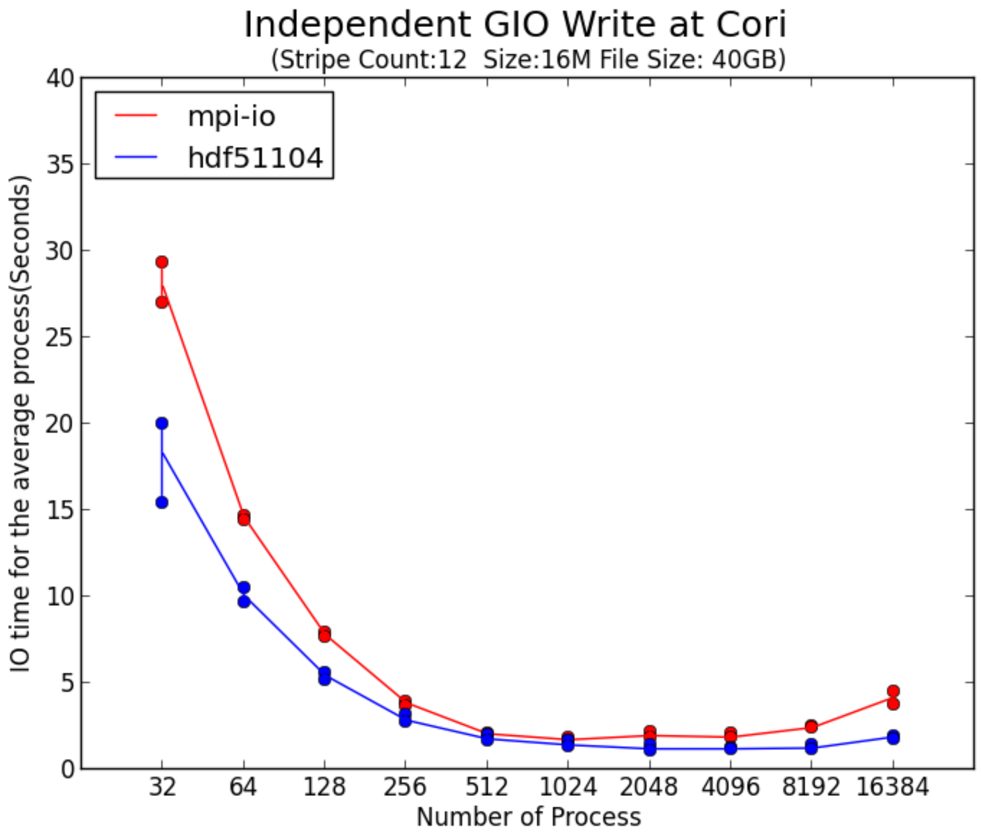


Pattern 2- HACC MPI-IO pattern



HDF5 Pattern 2 Implementation

- Use HDF5 compound datatype, then one big HDF5 write for each process



CGNS Performance Problems

- Opening an existing file
  - CGNS reads the entire HDF5 file structure, loading a lot of (HDF5) metadata
  - Reads occur independently on ALL ranks competing for the same metadata

    ➔"Read Storm"

- Closing a CGNS file
  - Triggers HDF5 flush of a large amount of small metadata entries
  - Implemented as iterative, independent writes, an unsuitable workload for parallel file systems

Metadata Read Storm Problem (I)

- All metadata “write” operations are required to be collective:

✗

```
if(0 == rank)
    H5Dcreate(“dataset1”);
else if(1 == rank)
    H5Dcreate(“dataset2”);
```

✓

```
/* All ranks have to call */
H5Dcreate(“dataset1”);
H5Dcreate(“dataset2”);
```

- Metadata read operations are not required to be collective

✓

```
if(0 == rank)
    H5Dopen(“dataset1”);
else if(1 == rank)
    H5Dopen(“dataset2”);
```

✓

```
/* All ranks have to call */
H5Dopen(“dataset1”);
H5Dopen(“dataset2”);
```

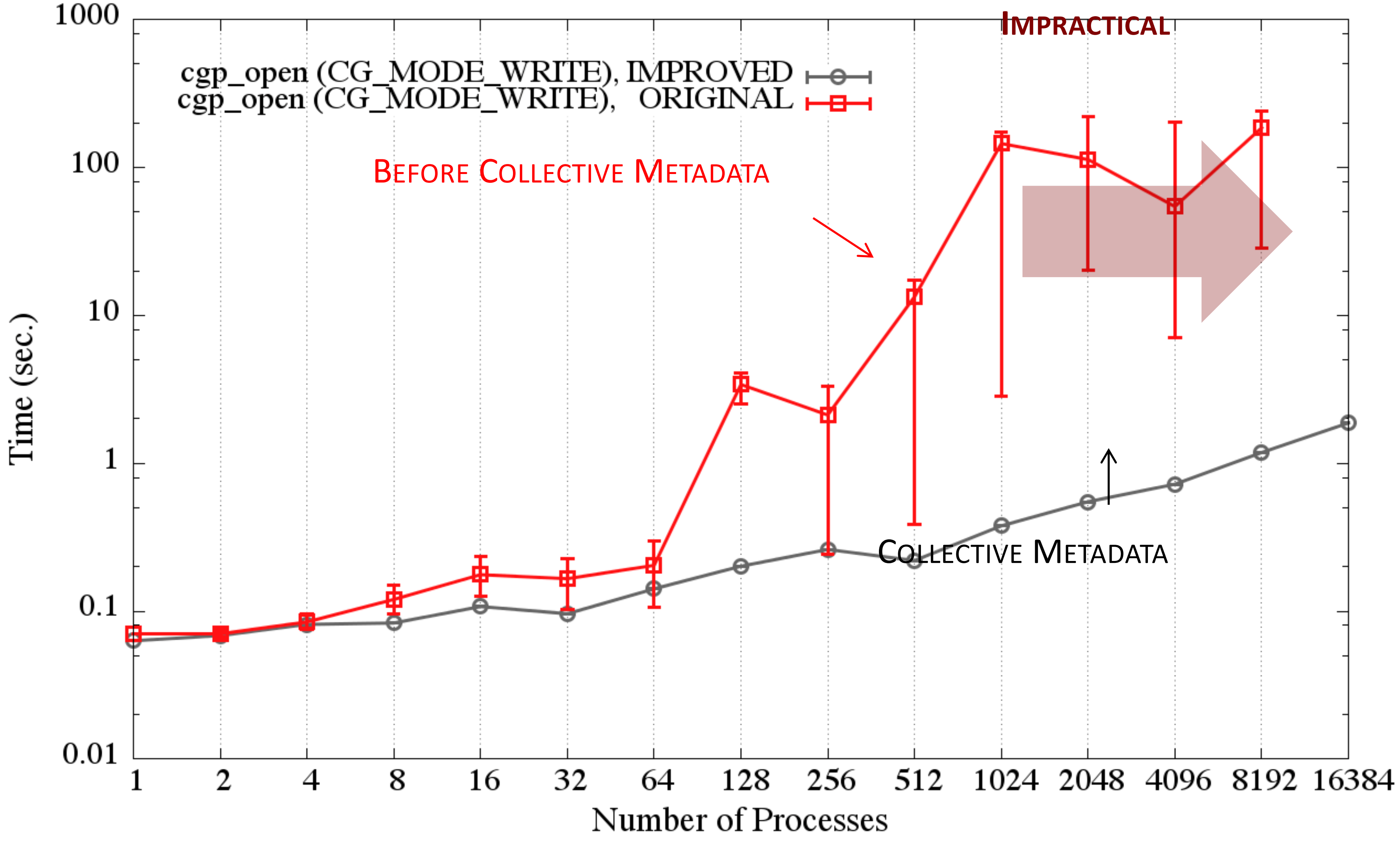
Metadata Read Storm Problem (II)

- Metadata read operations are treated by the library as independent read operations.
- Consider a very large MPI job size where all processes want to open a dataset that already exists in the file.
- All processes
  - Call `H5Dopen("/G1/G2/D1");`
  - Read the same metadata to get to the dataset and the metadata of the dataset itself
    - IF metadata not in cache, THEN read it from disk.
  - Might issue read requests to the file system for the same small metadata.
- ➔ READ STORM

Avoiding a Read Storm

- Hint that metadata access is done collectively
  - `H5Pset_coll_metadata_write`, `H5Pset_all_coll_metadata_ops`
- A property on an access property list
- If set on the file access property list, then all metadata read operations will be required to be collective
- Can be set on individual object property list
- If set, MPI rank 0 will issue the read for a metadata entry to the file system and broadcast to all other ranks

Opening CGNS File …



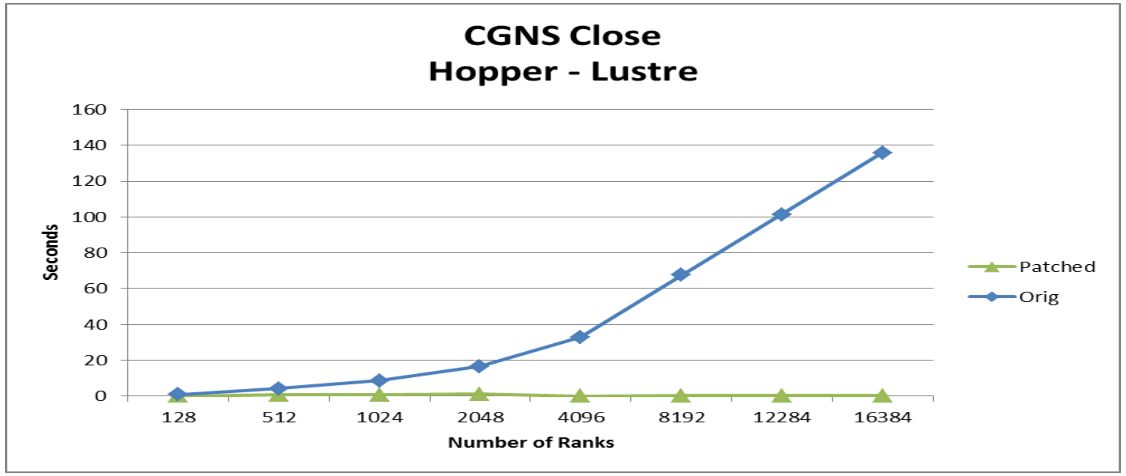
Write Metadata Collectively!

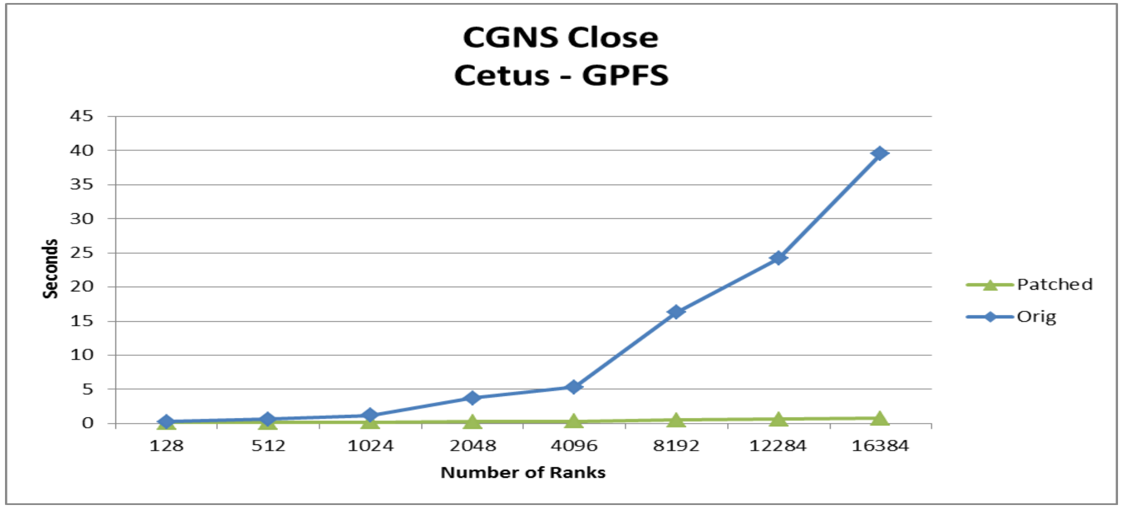
- **Symptoms:** Many users reported that `H5Fclose()` is very slow and doesn't scale well on parallel file systems.

- **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.

Closing a CGNS File …





Summary of today's class

- Parallel I/O performance factors and some application tuning examples

- Next Class – Tracing parallel I/O performance, visualizing