Foundations of HDF5 and Parallel I/O

July 27, 2022

M. Scot Breitenfeld

Talk Outline

- Foundations of HDF5
  - Introduction to
    - HDF5 data model, software, and architecture
    - HDF5 programming model
  - Overview of general best practices

- Overview of parallel HDF5
  - Introduction to HDF5 parallel I/O
  - General best practices and methods which affect parallel performance
  - Using Parallel I/O instrumentation for tuning

Why HDF5?

Have you ever asked yourself:

- How do I organize and share my data?
- How can I use visualization and other tools with my data?
- What will happen to my data if I need to move my application to another system?
- How will I deal with one-file-per-processor in the exascale era?
- Do I need to be an “MPI I/O and Lustre, or Object Store, etc.” pro to do my research?

- HDF5 is an answer to the questions above and can hide all complexity so you can concentrate your research

What is HDF5?

What is HDF5?

- **H**ierarchical **D**ata **F**ormat version 5 (**HDF5**)
  1. An extensible **data model**
     - Uses structures for data organization and specification
  2. Open source **software** (I/O library and tools)
     - Performs I/O on data organized according to the data model
     - Works with POSIX and other types of backing store: Object Stores (DAOS, AWS S3, AZURE, Ceph, etc.), memory hierarchies and other storage devices
  3. Open **file format** (POSIX storage only)

HDF5 is like …

HDF5

Databases
random access; subsetting

PDF
standard exchange format; heterogeneous information

XML
self-describing; extensible types; rich metadata

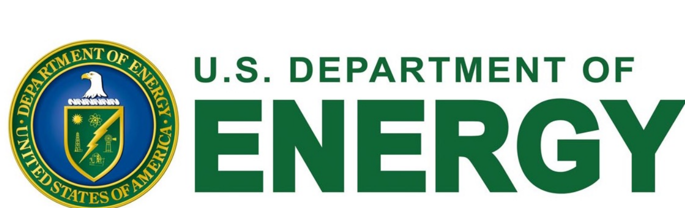
Binary "Flat File"
high-performance; compact; scalable

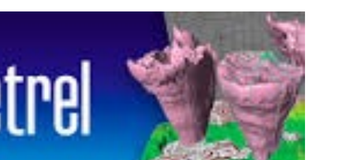
Directories & Files
hierarchical; collections of related information

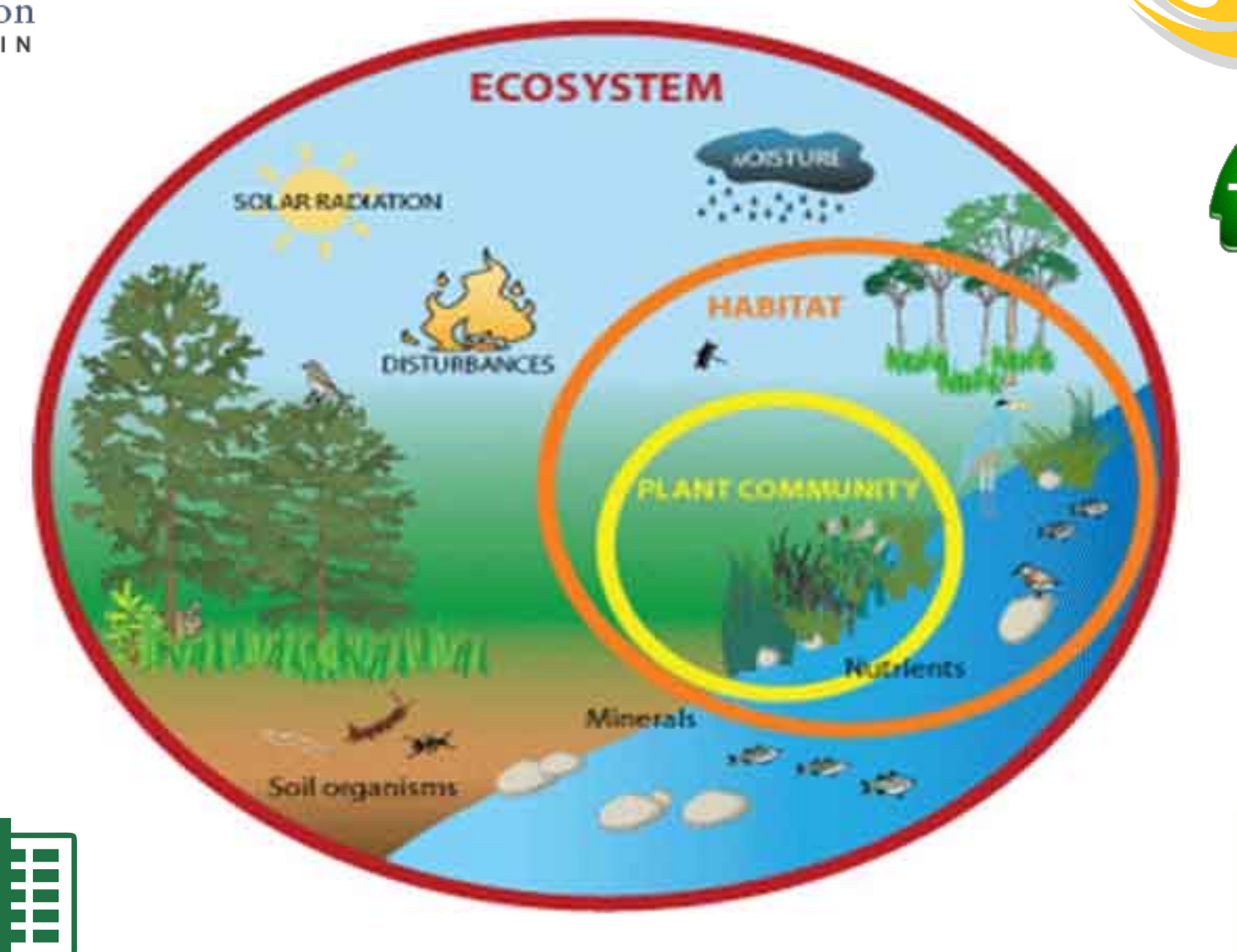
HDF5 is designed for…

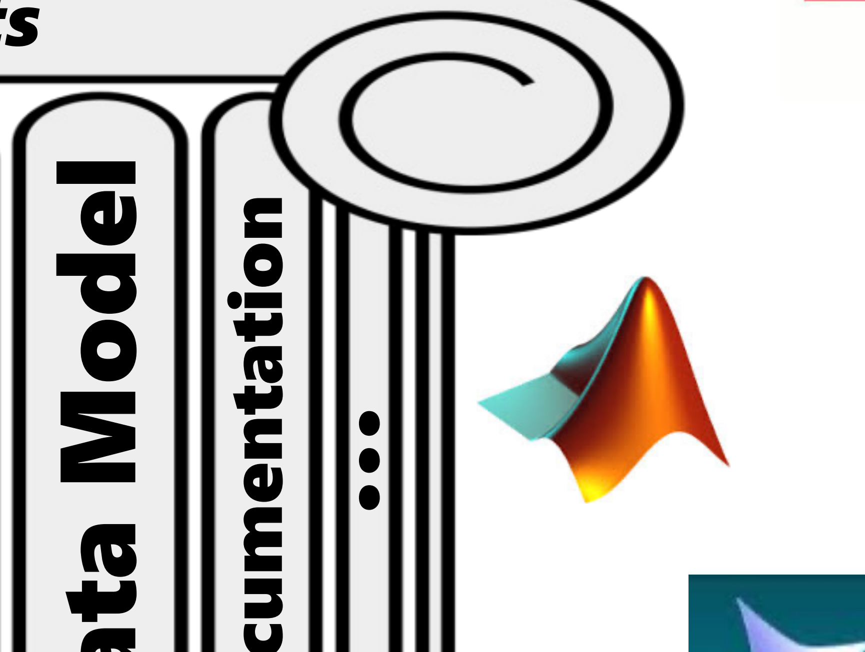
- High volume and complex data
  - HDF5 files of GBs sizes are common
- Every size and type of system (portable)
  - Works on from embedded systems, desktops and laptops to exascale systems
- Flexible, efficient storage and I/O
  - Works for a variety of backing storage
- Enabling applications to evolve in their use of HDF5 and to accommodate new models
  - Data can be added, removed and reorganized in the file
- Supporting long-term data preservation
  - Petabytes of remote sensing data including data for long-term climate research in NASA archives now

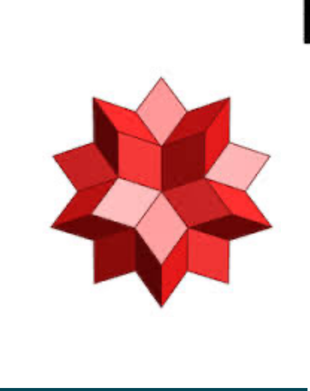
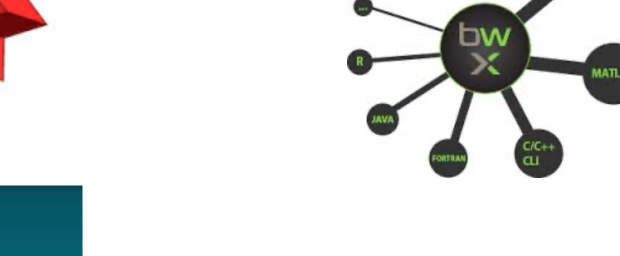
HDF5 Ecosystem

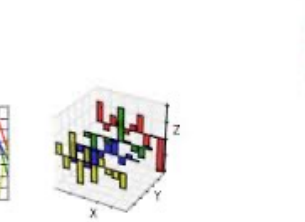








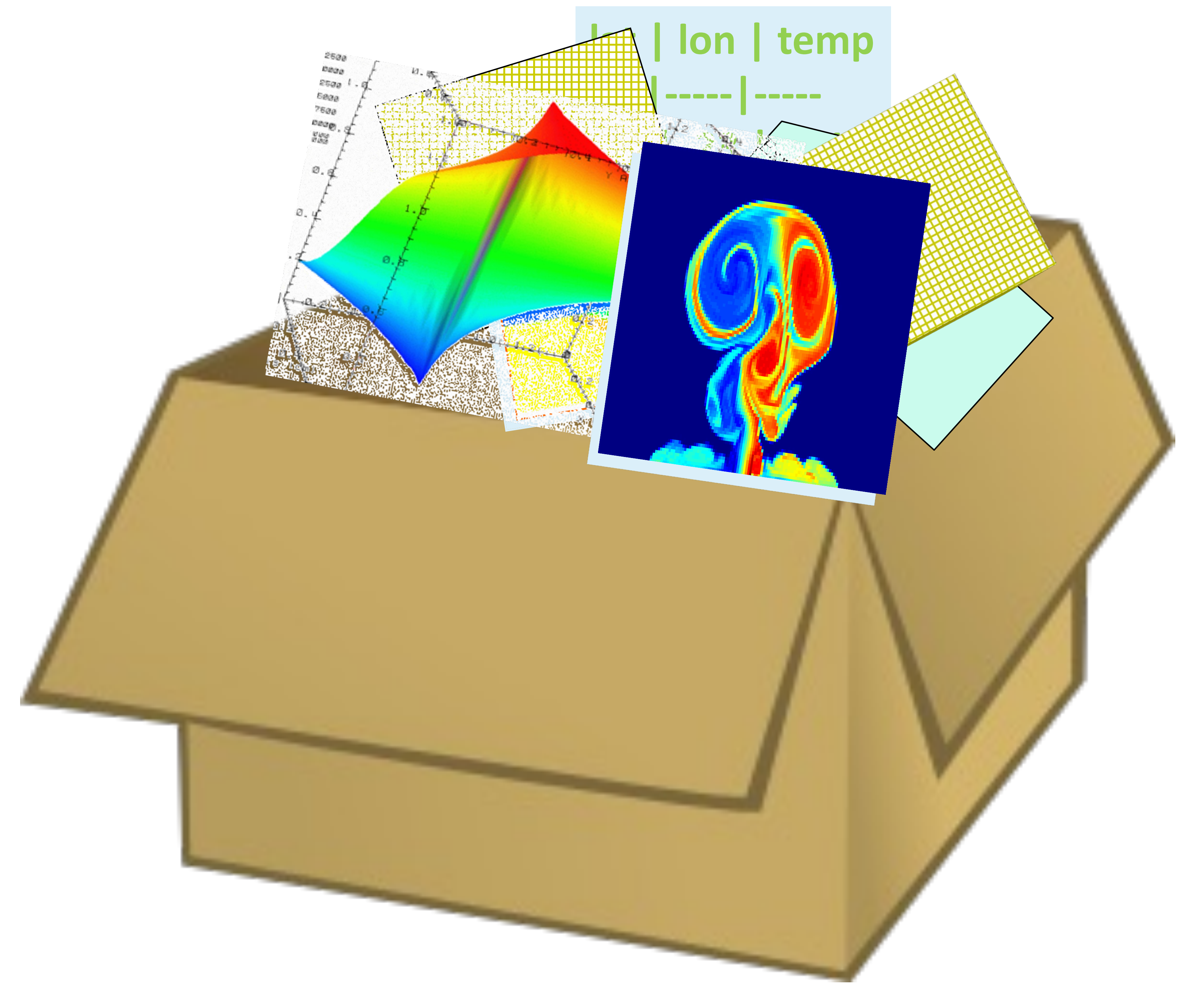





HDF5 Data model

HDF5 File

An HDF5 file is a **container** that holds data objects.



HDF5 Data Model

Dataset –
Organize and contain data elements

Dataspace –
Describes logical layout of the data elements

Attribute –
User-defined metadata

HDF5 Objects

File

Datatype –
Describes individual data elements

Link –
Organize data objects

Group –
Organize data objects

HDF5 Dataset

HDF5 Datatype

Integer: 32-bit, LE

HDF5 Dataspace

Rank	Dimensions
3	Dim[0] = 7
	Dim[1] = 4
	Dim[2] = 5

Specifications for single data element and array dimensions

Multi-dimensional array of identically typed data elements

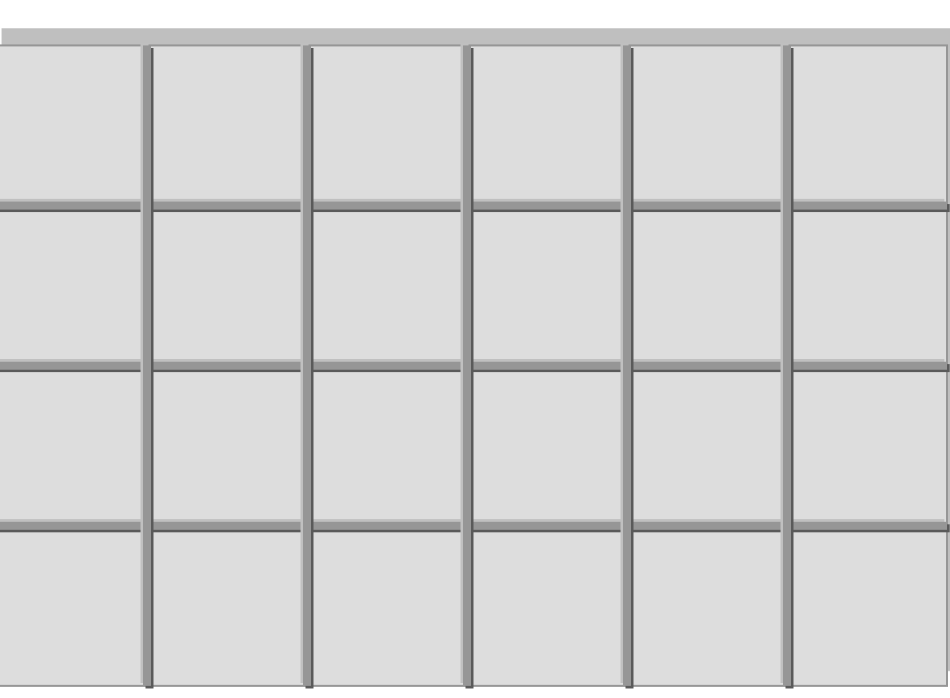
- HDF5 datasets **organize and contain** data elements
  - HDF5 datatype describes individual data elements
  - HDF5 dataspace describes the logical layout of the data elements

HDF5 Dataspace

Two roles:

(1) Spatial information for Datasets and Attributes

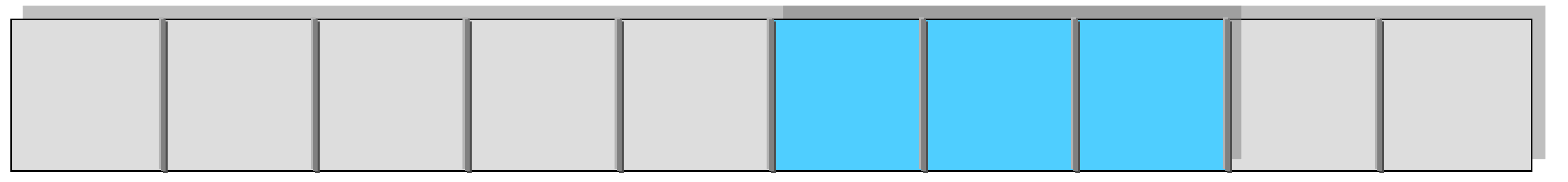
- Empty sets and scalar values
- Multidimensional arrays
  - Rank and dimensions
- A permanent part of object definition



Rank = 2

Dimensions = 4 x 6

(2) Partial I/O: Dataspace and selection describe the application's data buffer and data elements participating in I/O



Rank = 1

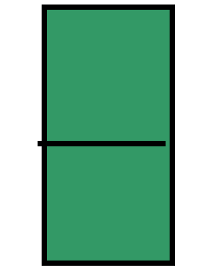
Dimension = 10

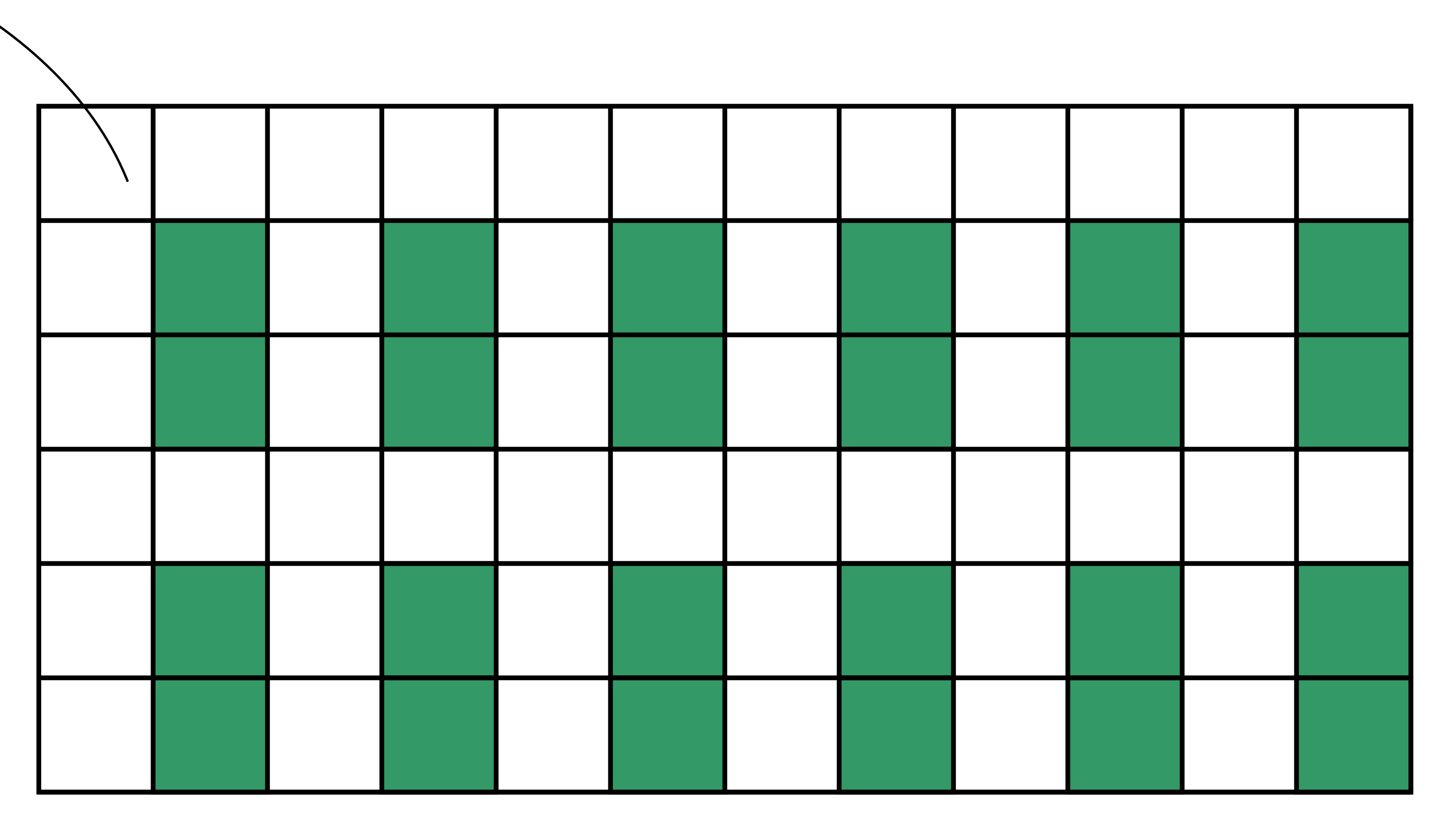
How to describe a subset in HDF5?

- Before writing and reading a subset of data, one must describe it to the HDF5 Library.
- The HDF5 APIs and documentation refer to a subset as a “selection,” for example “*hyperslab* selection.”
- If specified, HDF5 performs I/O on a selection *only* and <u>not</u> on all dataset elements.

Describing elements for I/O: HDF5 Hyperslab

- *Everything is "measured" in the number of elements; 0-based*
- Example 1-dim:
  - Start - starting location of a hyperslab (5)
  - Block - block size (3)
- Example 2-dim:
  - Start - starting location of a hyperslab (1,1)
  - Stride - number of elements that separate each block (3,2)
  - Count - number of blocks (2,6)
  - Block - block size (2,1)





- All other selections are built using set operations

HDF5 Datatypes

- Describe individual data elements in an HDF5 dataset
- A wide range of datatypes is supported
  - Atomic types: integer, floats
  - User-defined (e.g., 12-bit integer, 16-bit float)
  - Enum
  - References to HDF5 objects and selected elements of datasets
  - Variable-length types (e.g., strings, vectors)
  - Compound (similar to C's structures or Fortran's derived types)
  - Array (similar to matrix)
  - More complex types can be built from the types above
- HDF5 library provides predefined symbols to describe atomic datatypes

HDF5 Dataset with Compound Datatype

3

5

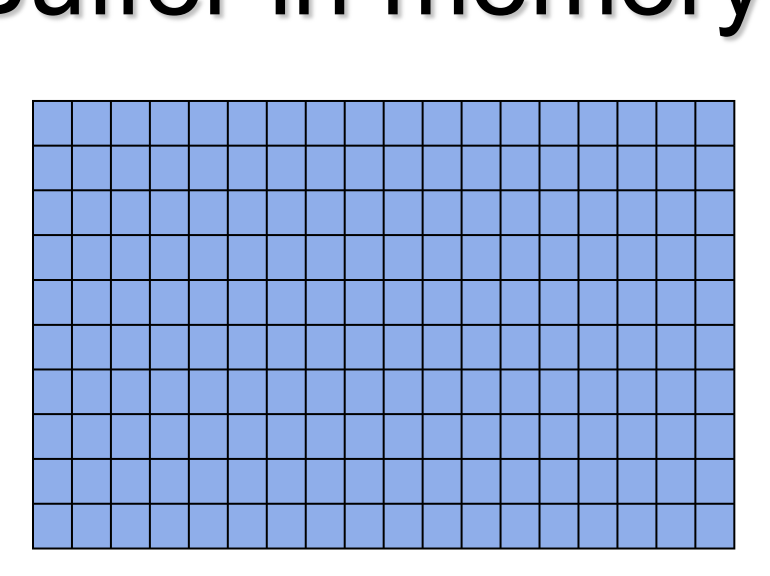
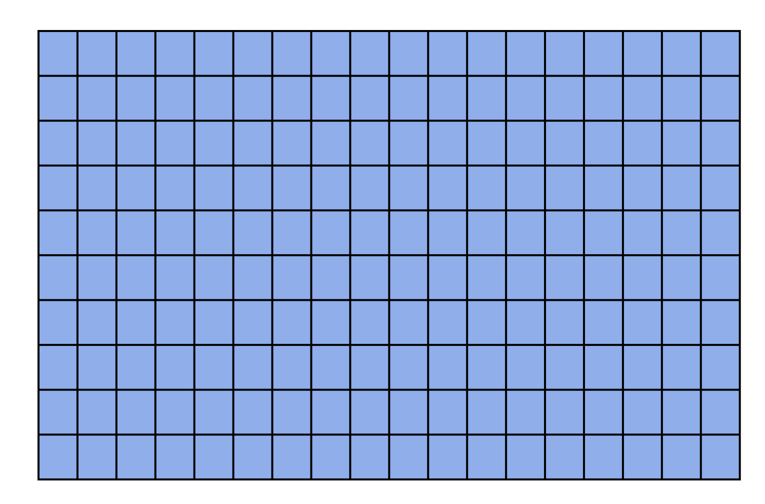
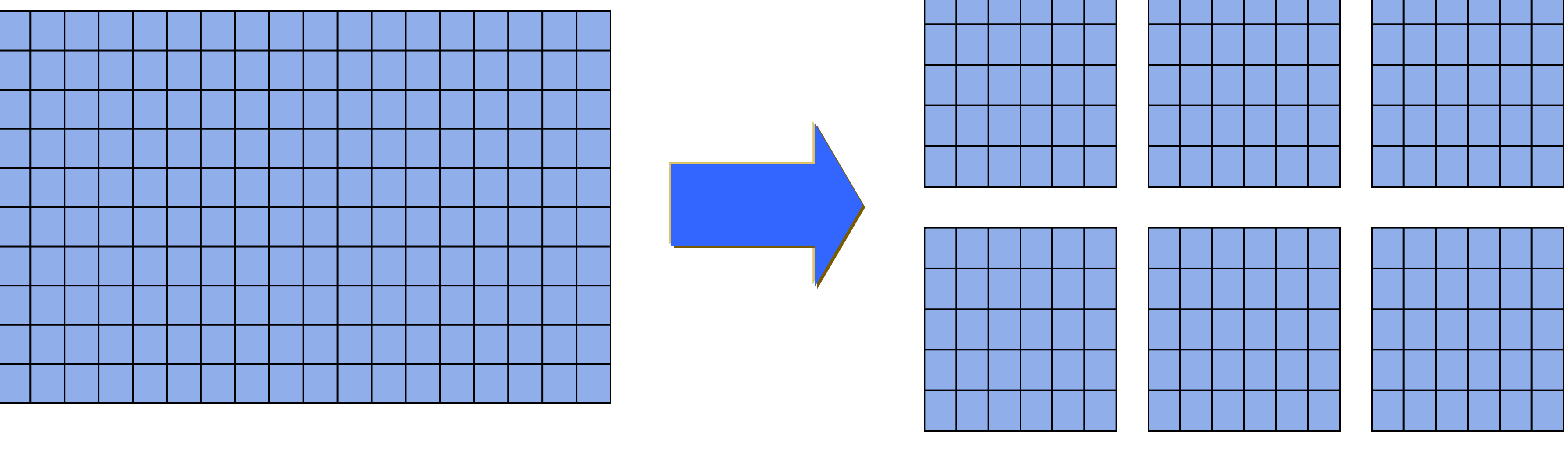
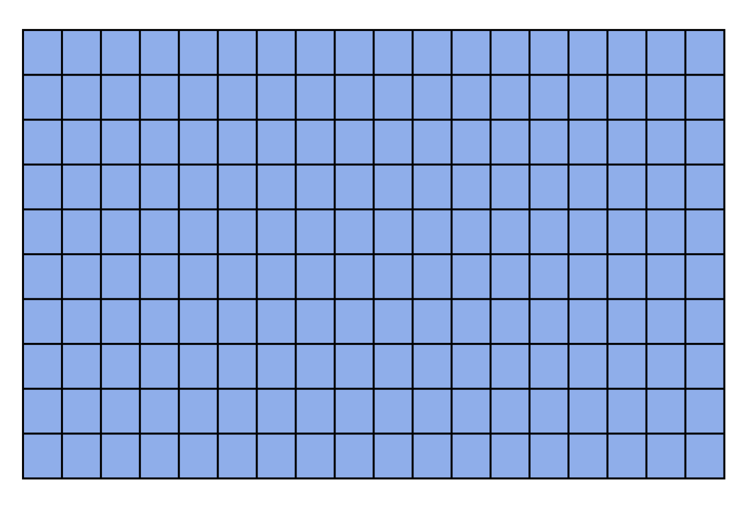
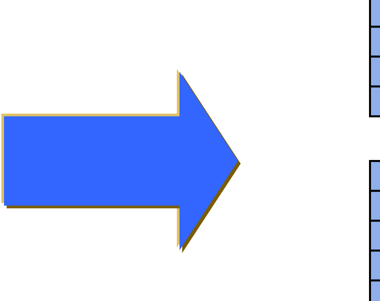
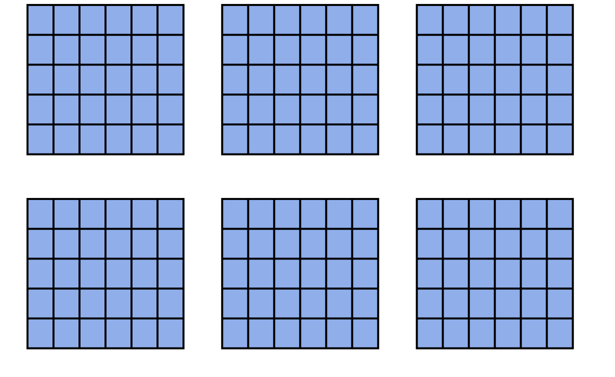
V V V
V V V
V V V

uint16 char int32 2x3x2 array of float32

Compound Datatype:

Dataspace: Rank = 2
Dimensions = 5 x 3

How are data elements stored? (1/2)

	Buffer in memory		Data in the file	
Contiguous (default)				**Data elements stored physically adjacent to each other**
Chunked				**Better access time for subsets; extendible**
Chunked & Compressed				**Improves storage efficiency, transmission speed**

Compression and filters in HDF5

- GZIP and SZIP (free version is available from German Climate Computing Center)
- Other compression methods registered with The HDF Group at https://portal.hdfgroup.org/display/support/Contributions#Contributions-filters
  - BZIP2, JPEG, LZF, BLOSC, MAFISC, LZ4, Bitshuffle, SZ and ZFP, etc.
    - The listed above are available as dynamically loaded plugins
- Filters:
  - Fletcher32 (checksum)
  - Shuffle
  - Scale+offset
  - n-bit

How are data elements stored? (2/2)

	Buffer in memory	Data in the file	
Compact		Dataset Object Header	**Data elements stored directly within object's metadata**
External		Dataset Object Header	**Data elements stored outside the HDF5 file, possibly in another file format**
Virtual			**Data elements are stored in "source datasets," using selections to map**

HDF5 Attributes

- Attributes “decorate” HDF5 objects
- Contain *user-defined* metadata
- Similar to Key-Values:
  - Have a unique name (for that object) and a value
- Analogous to a dataset
  - “Value” is described by a datatype and a dataspace
  - **Do not** support partial I/O operations; nor can they be compressed or extended

HDF5 Groups and Links

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

Every HDF5 file has a root group

/

Experiment Notes:
Serial Number: 99378920
Date: 3/13/09
Configuration: Standard 3

Viz

SimOut

Parameters
10;100;1000

lat	lon	temp
12	23	3.1
15	24	4.2
17	21	3.6

Timestep
36,000

HDF5 software and architecture

HDF5 Software

HDF5 home page: http://hdfgroup.org/HDF5/

- Latest releases: HDF5 1.8.21,1.10.9, 1.12.2, 1.13.1 (precursor to 1.14.0)

HDF5 source code:

- Available on GitHub: https://github.com/HDFGroup/hdf5
- Written in C and includes optional C++, Fortran, Java APIs, and High-Level APIs
- Contains command-line utilities (h5dump, h5repack, h5diff, ..) and compile scripts

HDF5 pre-built binaries:

- Include C, C++, Fortran, Java, and High-Level libraries when possible. Check ./lib/libhdf5.settings file.
- Built with the SZIP and ZLIB external libraries

3rd party software:

- h5py (Python)
- http://h5cpp.org/ (Contemporary C++ including support for MPI I/O)

Useful Tools For New Users

h5dump

Tool to "dump" or display contents of HDF5 files

Scripts to compile applications:

h5cc, h5c++, h5fc (*h5pcc, h5pfc – parallel variants*)

HDFView:

Java browser to view HDF5 file

https://portal.hdfgroup.org/display/HDFVIEW/HDFView

HDF5 Examples (C, Fortran, Java, Python, Matlab, ...)

https://portal.hdfgroup.org/display/HDF5/HDF5+Examples

HDF5 Library Architecture (1.12.0 +)

HDF5 Core Library

HDF5 API and language bindings

Virtual Object Layer (VOL)

Pass-through VOL connectors (e.g., Async IO)

Native Connector

VFDs

POSIX

MPI I/O

SWMR

....

S3

HDFS

REST

DAOS

....

Log Based

ADIOS

VOL connectors

HDFS architecture

NameNode

Metadata (name, replicas, ...): /home/foo/data, 3, ...

Client

DataNodes

Blocks

RACK 1

RACK 2

HDF5 Programming model and API

The General HDF5 API

- C, FORTRAN, Java, and C++
- C routines begin with the prefix: H5*

  * corresponds to the type of object the function acts on

Example Functions:

H5D : **D**ataset interface *e.g.,* **H5Dread**

H5F : **F**ile interface *e.g.,* **H5Fopen**

H5S : data**S**pace interface *e.g.,* **H5Sclose**

  - The language wrappers follow the same trend

- There are more than 300 APIs – but one can start with less than 50

General Programming Paradigm

- Properties of an object are optionally defined
  - Creation properties (e.g., use chunking storage)
  - Access properties (e.g., using MPI I/O driver to access file)
- Object is opened or created
  - Creation properties applied
  - Access properties applied
  - Supporting objects are defined (datatype, dataspace)
- Object is accessed possibly many times
  - Access property can be changed
- Object is closed

Standard HDF5 program “Skeleton”

H5**F**create (H5**F**open)	create (open) File
H5**S**create_simple/H5**S**create	create dataSpace
H5**D**create (H5**D**open)	create (open) Dataset
H5**D**read, H5**D**write	**access Dataset**
H5**D**close	close Dataset
H5**S**close	close dataSpace
H5**F**close	close File

General best practices

HDF5 Dataset I/O

- 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

- Use **contiguous storage** if no data will be added and compression is not used
  - HDF5 will not cache data
- Use **compact** storage when working with small data (<64K)
  - Data becomes part of HDF5 internal metadata and is cached (metadata cache)
- If you have **binary files** that you would like to convert to HDF5, consider **external storage** and use the h5repack tool
- 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

HDF5 Dataset – Chunked Storage

- 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)?
  - Do you need to adjust chunk cache size (1 MB default; can be set up per file or per dataset)?
  - `H5Pset_chunk_cache` sets raw data chunk cache parameters for a dataset
    - `H5Pset_chunk_cache (dapl, …);`
  - `H5Pset_cache` sets raw data chunk cache parameters for all datasets in a file
    - `H5Pset_cache (fapl, …);`
  - Investigate other parameters to control chunk cache

HDF5 Dataset – Chunked Storage (cont'd)

- Cache size is essential when doing partial I/O to avoid many I/O operations
- With the 1 MB cache size, a chunk will not fit into the cache
  - All writes to the dataset must be immediately written to the disk
  - With compression, the entire chunk must be read, and rewritten, every time a part of the chunk is written to
    - Data must also be decompressed and recompressed each time

- Without compression, the entire chunk must be written when it is first written to the file.
- To write multiple chunks at once, increase the cache size to hold more chunks

Effect of chunk cache size on read

- With compression, HDF5 reads entire chunk once for each call to **`H5Dread,`** cache size does not matter
- Without compression, HDF5's behavior depends on the cache size relative to the chunk size.
  - If the chunk fits in the cache, HDF5 reads the entire chunk once for each call to **`H5Dread`**
  - If the chunk does not fit in cache, the library reads only the data that is selected
  - Cache must be large enough to hold all the chunks to perform well.
- The optimum cache size depends on the exact shape of the data, as well as the hardware, as well as access pattern.

Will This Chunk Be Cached?

Start

Does Dataset have I/O Filters?

No

Using MPI-IO VFD with file write access?

Yes

Chunk is not cached

Yes

No

Yes

Yes

No

Does Dataset allow Partial Edge Chunks?

Yes

Is Partial Chunk, at Edge?

Chunk fits in Dataset's Chunk Cache?

No

Reading Data?

No

Has Chunk been Allocated in File?

No

No

Yes

No

Chunk is cached

Yes

Should Dataset Fill value be written?

Memory considerations

- **Open Objects**
  - Open objects use up memory. The amount of memory used may be substantial when many objects are left open. Application should:
    - Delay opening of files and datasets as close to their actual use as is feasible.
    - Close files and datasets as soon as their use is completed.
    - If writing to a portion of a dataset in a loop, close the dataspace with each iteration, as this can cause a large temporary "memory leak."

- There are APIs to determine if objects are left open. H5Fget_obj_count will get the number of open objects in the file, and H5Fget_obj_ids will return a list of the open object identifiers.

Memory considerations (cont'd)

- **Metadata Cache**
  - The metadata cache can also impact memory usage.
    - Modify the metadata cache settings to minimize the size and growth of the cache as much as possible without decreasing performance.

  - By default, the metadata cache is 2 MiB in size (maximum allowed 32 MiB per file).
    - The metadata cache can be disabled or modified.
    - Memory used for the cache is not released until the datasets or file is closed.
  - https://portal.hdfgroup.org/display/HDF5/Metadata+Caching+in+HDF5

  - See https://portal.hdfgroup.org/display/HDF5/H5P_GET_MDC_CONFIG get default MD cache configurations and https://portal.hdfgroup.org/display/HDF5/H5P_SET_MDC_CONFIG to set new configuration

  - To keep MD cache from growing, consider evicting objects on close https://portal.hdfgroup.org/display/HDF5/H5P_SET_EVICT_ON_CLOSE

Parallel I/O with HDF5

PHDF5 implementation layers

APPLICATION

COMPUTE NODE

COMPUTE NODE

COMPUTE NODE

HDF5 LIBRARY

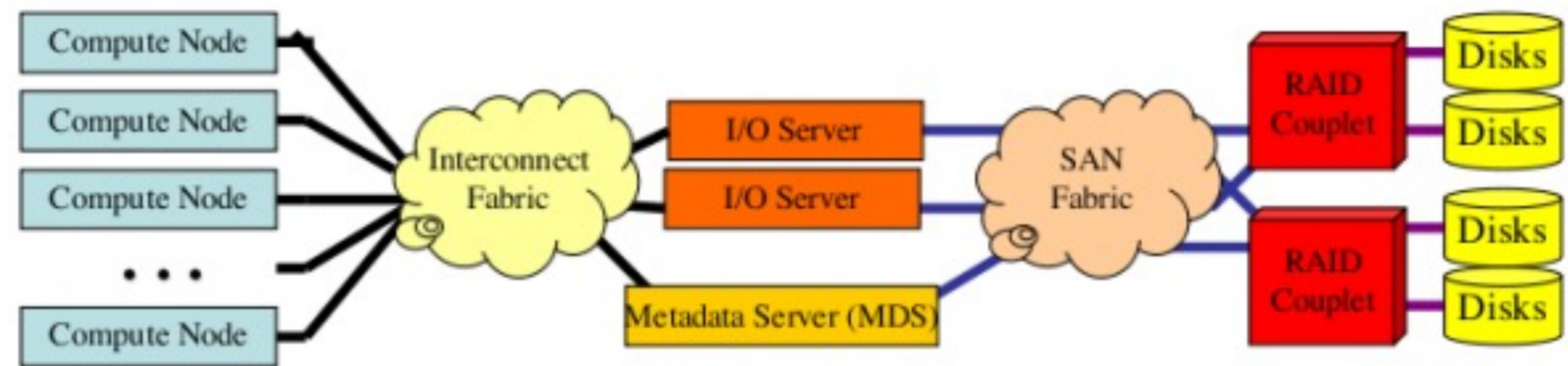
MPI I/O LIBRARY

HDF5 FILE ON PARALLEL FILE SYSTEM

INTERCONNECT NETWORK + I/O SERVERS

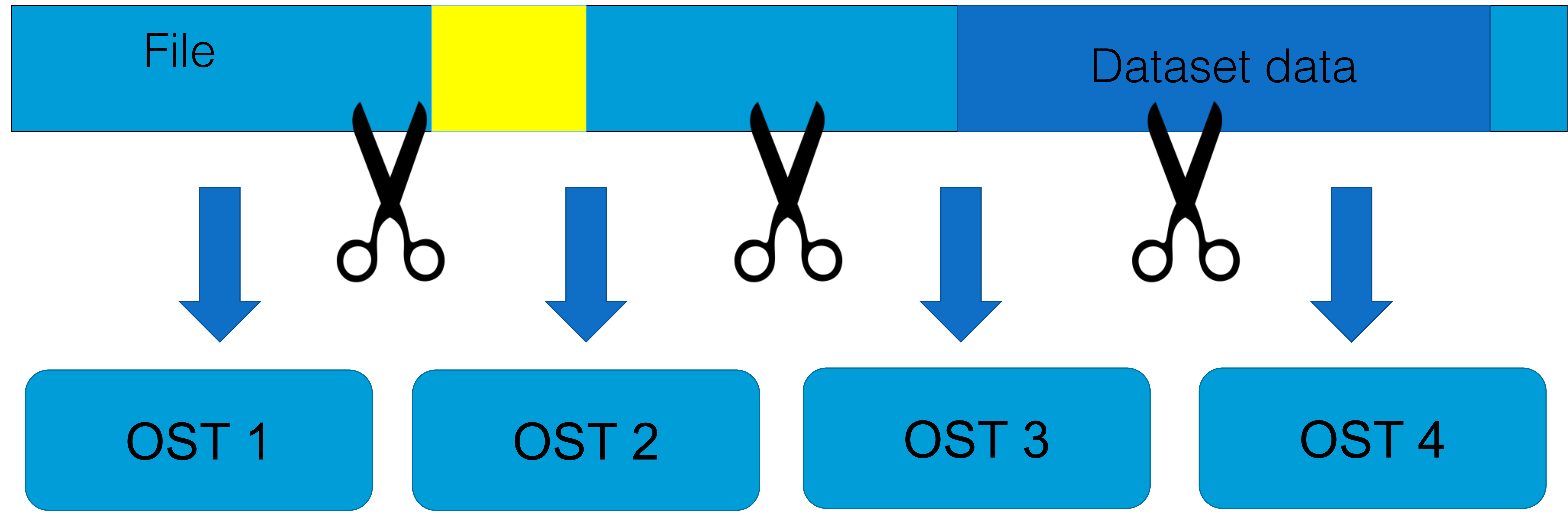
DISK ARCHITECTURE AND LAYOUT OF DATA ON DISK

Parallel File Systems – Lustre, GPFS, etc.



- Scalable, POSIX-compliant file systems designed for large, distributed-memory systems
- Uses a client-server model with separate servers for file metadata and file content

In a Parallel File System



The file is striped over multiple “disks” (e.g. Lustre OSTs) depending on the stripe size and stripe count with which the file was created.

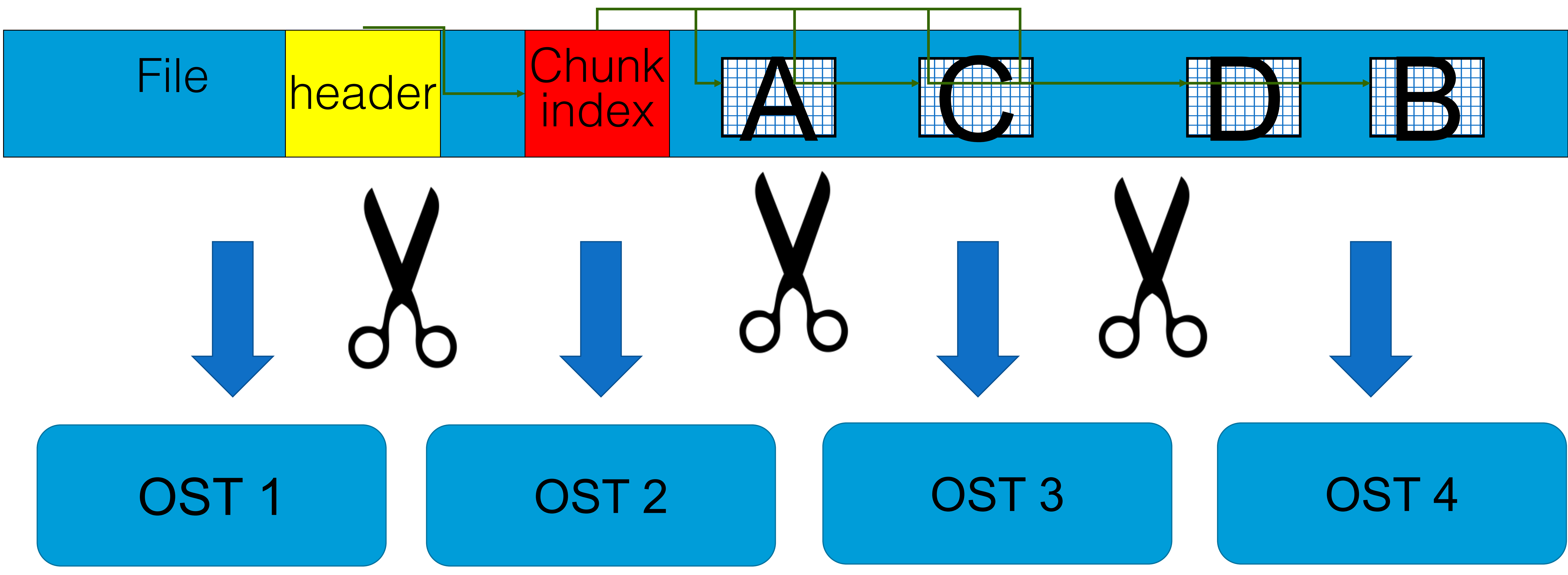
Contiguous Storage

- Metadata header separate from dataset data
- Data stored in one contiguous block in HDF5 file

Chunked Storage

- Dataset data is divided into equally sized blocks (chunks).
- Each chunk is stored separately as a contiguous block in HDF5 file.

In a Parallel File System



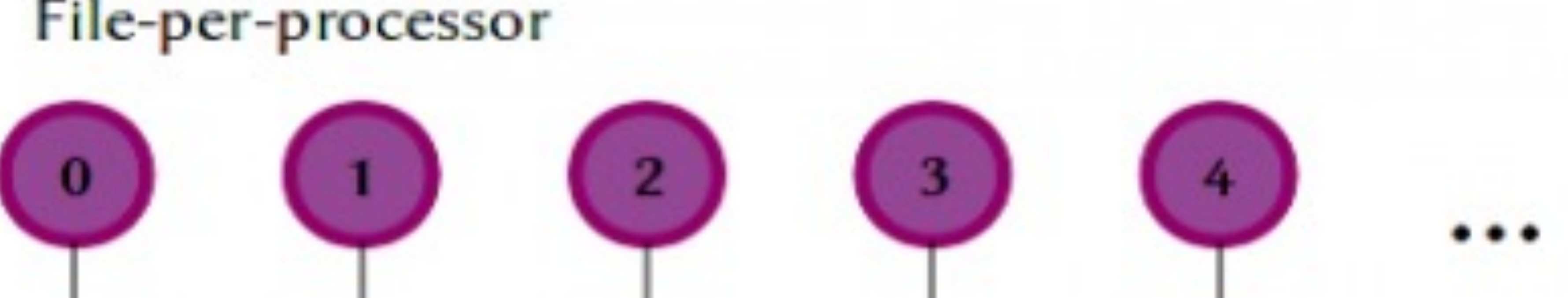
The file is striped over multiple OSTs depending on the stripe size and stripe count with which the file was created.

Compact dataset

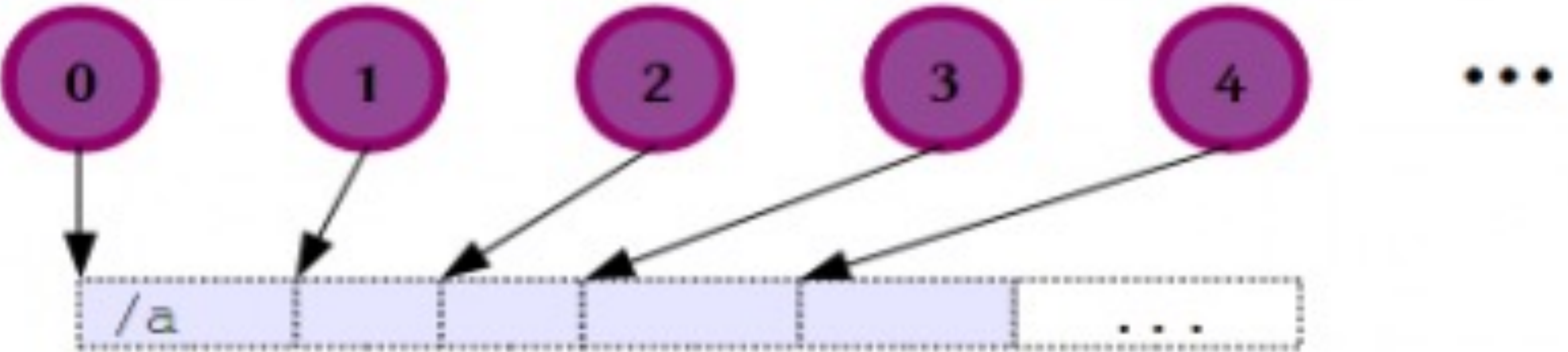
Raw data is written when object header is written

Types of Application I/O to Parallel File Systems

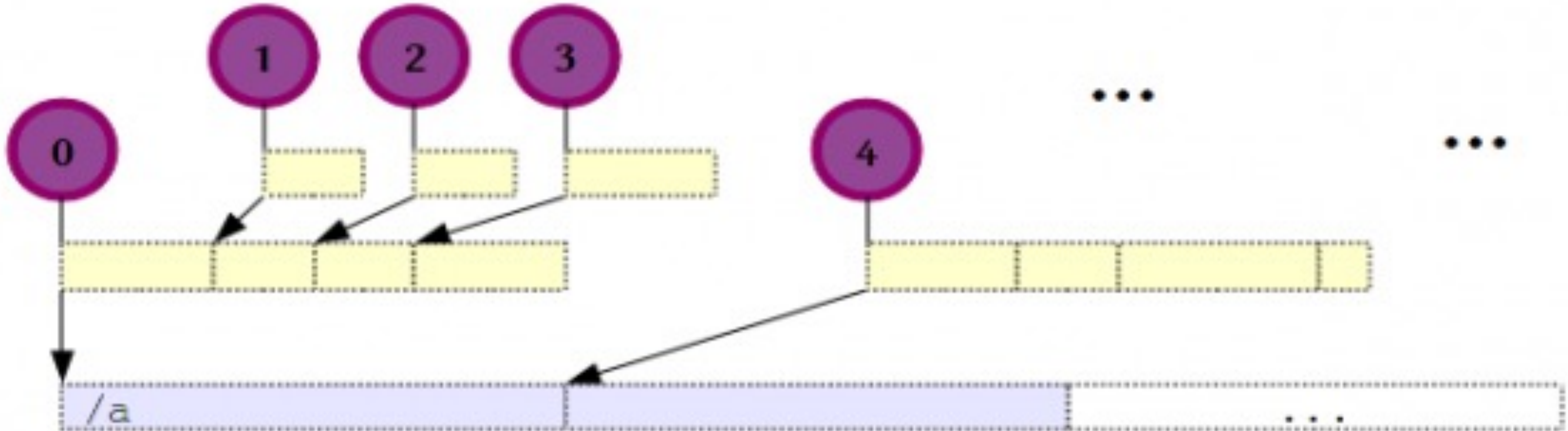
File-per-processor



Shared file (independent)



Shared file (collective buffering)



Why Parallel HDF5?

- Take advantage of high-performance parallel I/O while reducing complexity
  - Use a well-defined high-level I/O layer instead of POSIX or MPI-IO
  - Use only a single or a few shared files
    - “Friends don’t let friends use file-per-process!” 😕
- Maintained code base, performance and data portability
  - Rely on HDF5 to optimize for underlying storage system

What We'll Cover Here

- Parallel vs. Serial HDF5
- Implementation Layers
- HDF5 files (= composites of data & metadata) in a parallel environment
- Parallel HDF5 (PHDF5) I/O modes: collective vs. independent
- Data and Metadata I/O

Terminology

- DATA – “problem-size” data, e.g., large arrays
- METADATA – is an overloaded term
- In this presentation:

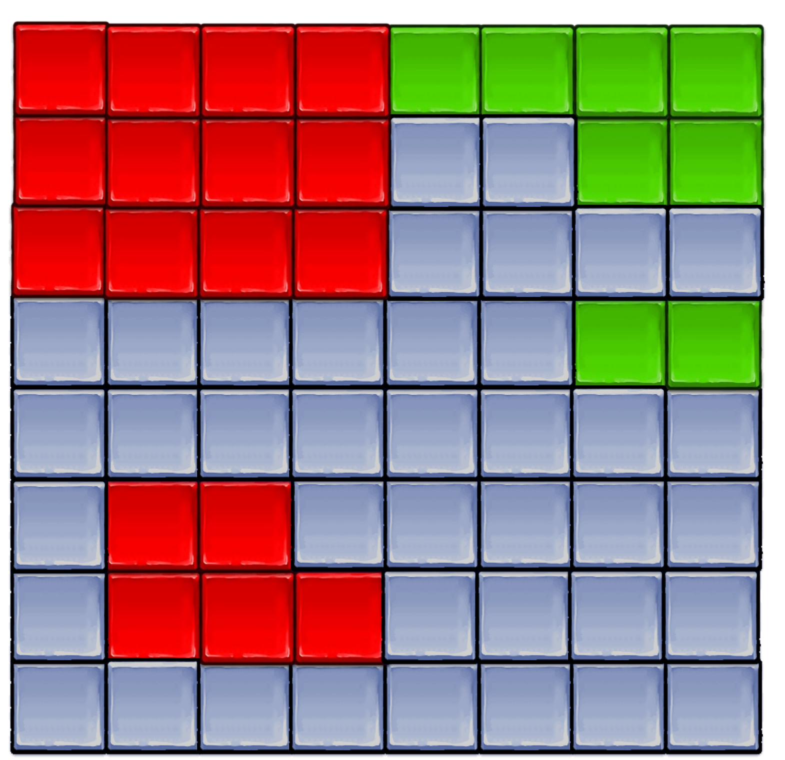
  Metadata “=“ HDF5 metadata
  - For each piece of application metadata, there are many associated pieces of HDF5 metadata
  - There are also other sources of HDF5 metadata
    - Chunk indices, heaps to store group links and indices to look them up, object headers, etc.

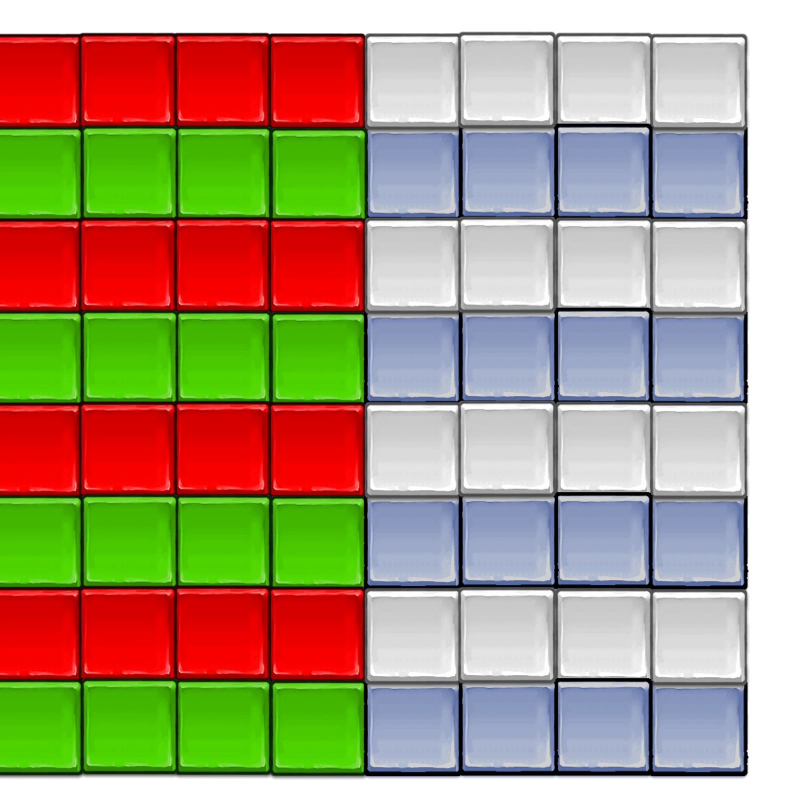
Parallel HDF5 (PHDF5) vs. Serial HDF5

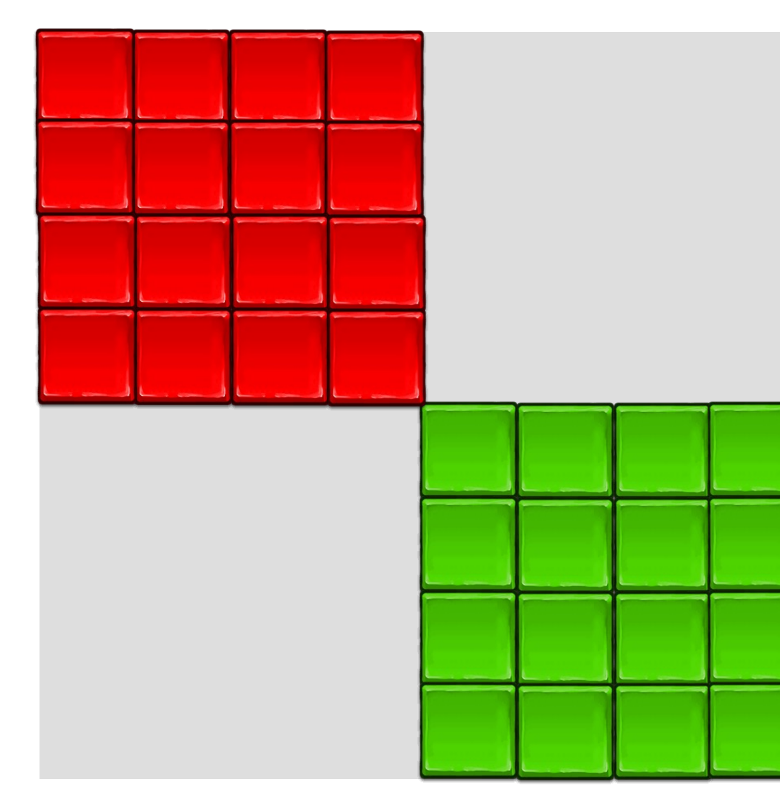
- PHDF5 allows multiple MPI processes in an MPI application to perform I/O to a single HDF5 file
- PHDF5 uses a standard parallel I/O interface (MPI-IO)
- Portable to different platforms
- PHDF5 files ARE HDF5 files conforming to the HDF5 file format specification
- The PHDF5 API consists of:
  - The standard HDF5 API
  - A few extra knobs and calls
  - A parallel "schema"

General HDF5 Programming Parallel Model for raw data I/O

- Each process defines selections in memory and in file (aka HDF5 hyperslabs) using `H5Sselect_hyperslab`
- The hyperslab parameters define the portion of the dataset to write to
  - Contiguous hyperslab
  - Regularly spaced data (column or row)
  - Pattern
  - Blocks







- Each process executes a write/read call using selections, which can be either collective or independent

Example of a PHDF5 C Program

Starting with a simple serial HDF5 program:

```
file_id = H5Fcreate(FNAME, …, H5P_DEFAULT);
space_id = H5Screate_simple(…);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id, …);


status = H5Dwrite(dset_id, H5T_NATIVE_INT, …, H5P_DEFAULT);
```

Example of a PHDF5 C Program

A parallel HDF5 program has a few extra calls:

```
MPI_Init(&argc, &argv);
…
fapl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl_id, comm, info);
file_id = H5Fcreate(FNAME, …, fapl_id);
space_id = H5Screate_simple(…);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id, …);
xf_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(xf_id, H5FD_MPIO_COLLECTIVE);
status = H5Dwrite(dset_id, H5T_NATIVE_INT, …, xf_id);
…
MPI_Finalize();
```

Parallel HDF5 Schema

- PHDF5 opens a shared file with an MPI communicator
  - Returns a file ID (as usual)
  - All future access to the file via that file ID
- Different files can be opened via different communicators
- All processes must participate in collective PHDF5 APIs
- All HDF5 APIs that modify the HDF5 namespace and structural metadata are collective!
  - File ops., group structure, dataset dimensions, object life-cycle, etc. https://support.hdfgroup.org/HDF5/doc/RM/CollectiveCalls.html
  - Raw data operations can either be collective or independent
    - For collective, all processes must participate, but they don't need to read/write data.

Object Creation (Collective vs. Single Process)

Collective Object Creation
One Processes Object Creation

Time (seconds)

Number of Processes

8 16 32 64 128 256 512 1024 2048 4096 8192 16384

CAUTION: Object Creation (Collective vs. Single Process)

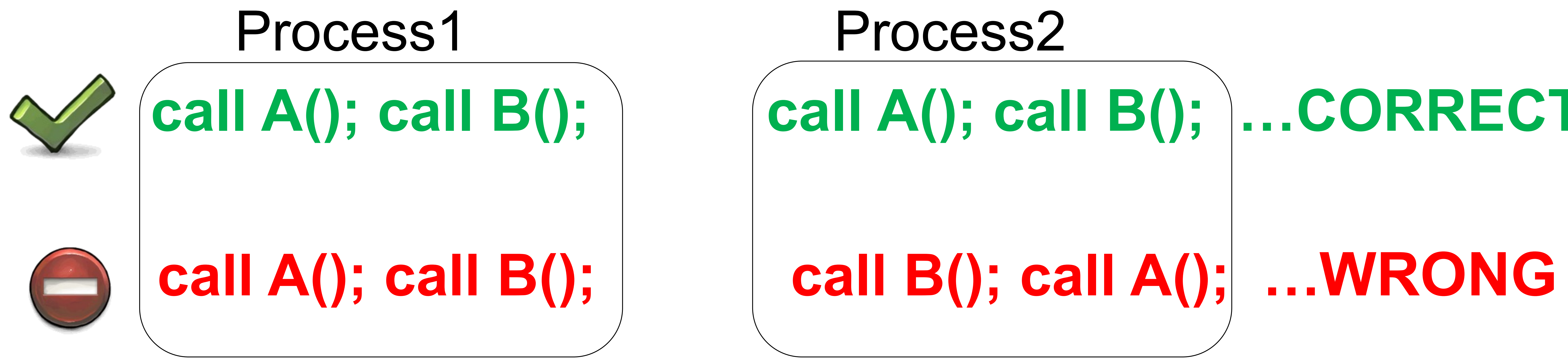
- In sequential mode, HDF5 allocates chunks incrementally, i.e., when data is written to a chunk for the first time.
  - Chunk is also initialized with the default or user-provided fill value.
- In the parallel case, chunks are always allocated when the dataset is created (not incrementally).
  - The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

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

Set HDF5 to never fill chunks (H5Pset_fill_time with H5D_FILL_TIME_NEVER)

Collective vs. Independent Operations

- MPI Collective Operations:
  - All processes of the communicator must participate, in the right order. E.g.,



- Collective I/O attempts to combine multiple smaller independent I/O ops into fewer larger ops; neither mode is preferable *a priori*

Writing and Reading Hyperslabs

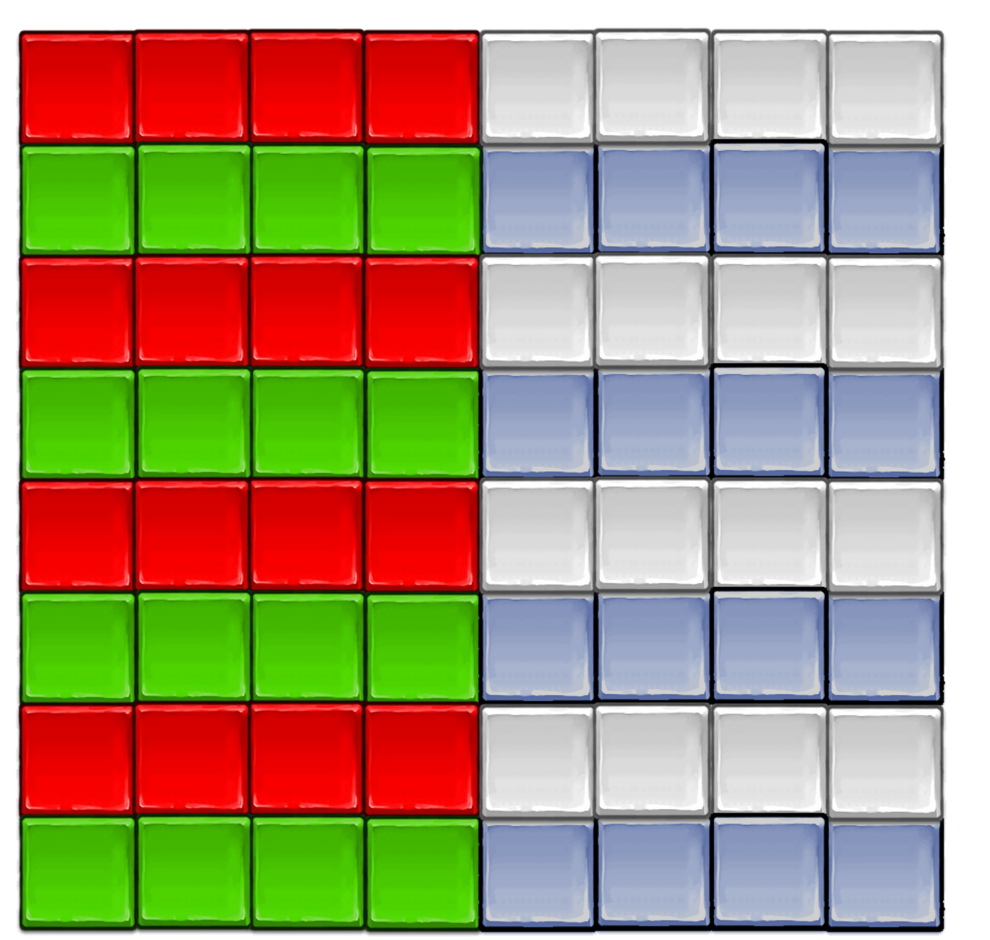
- Distributed memory model: data is split among processes
- PHDF5 uses HDF5 hyperslab model
- Each process defines memory and file hyperslabs

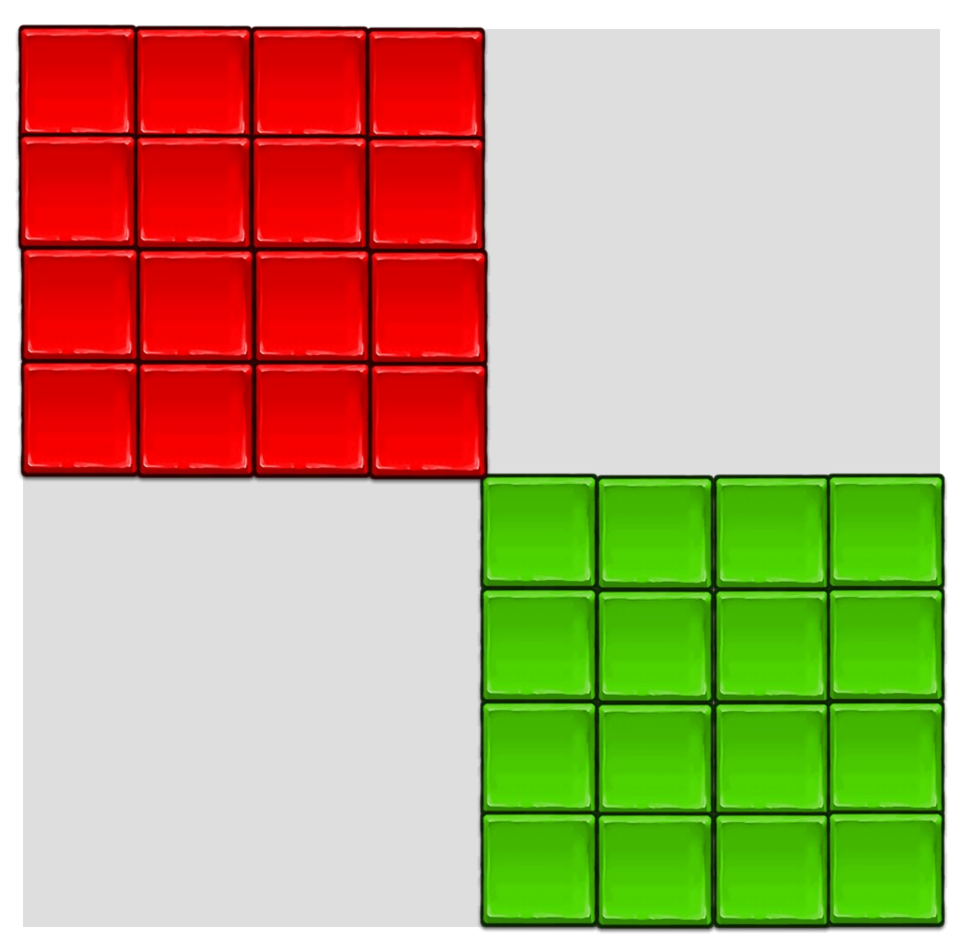
```
H5Sselect_hyperslab(space_id, H5S_SELECT_SET,
                    offset, stride, count, block)
```

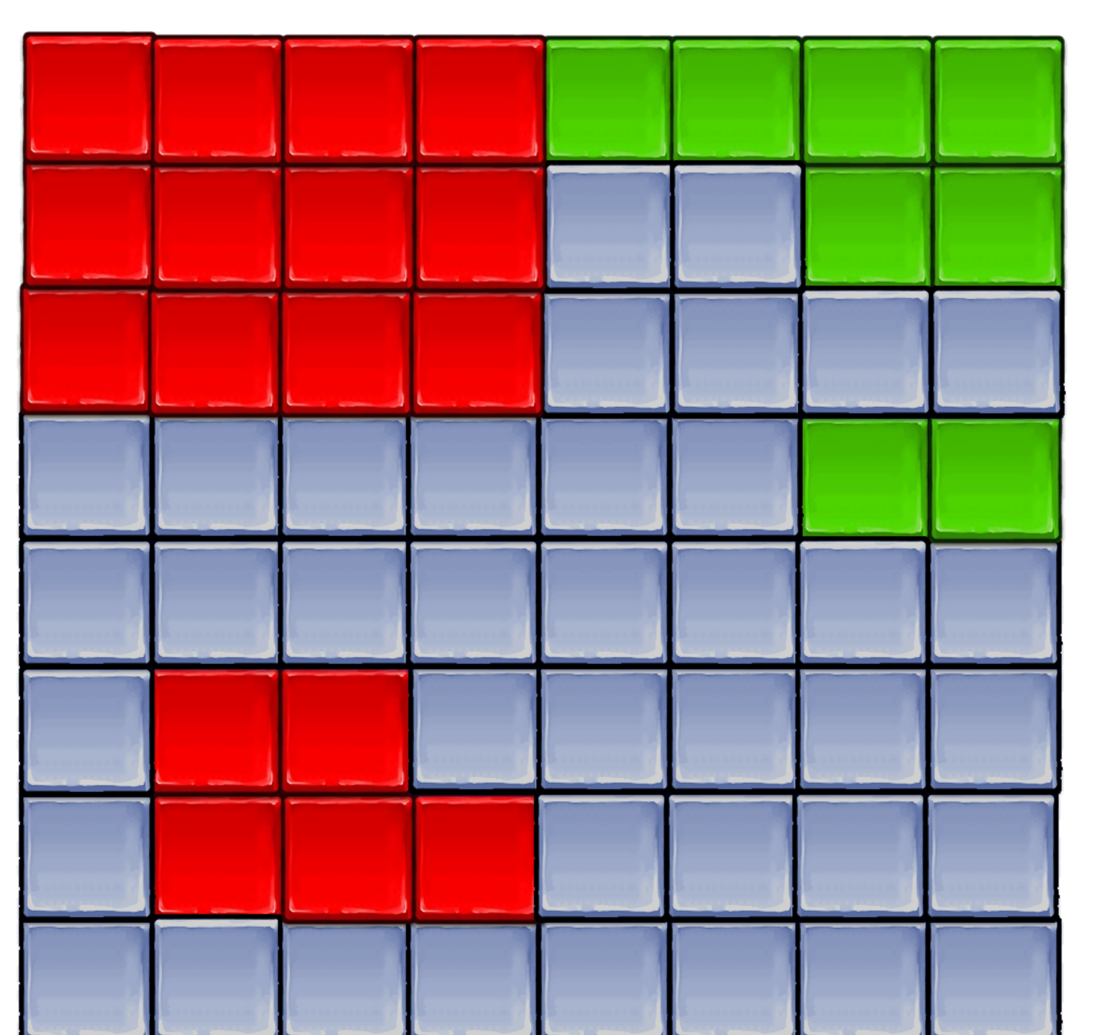
- Each process executes partial write/read call
  - Collective calls
  - Independent calls

Complex data patterns

HDF5 doesn't have restrictions on data patterns and balance







Irregular hyperslabs created by union operators

```
H5Sselect_hyperslab(space_id, op,
start, stride, count, block )
```

Complex data patterns -- Selection

H5S_SELECT_SET

H5S_SELECT_OR

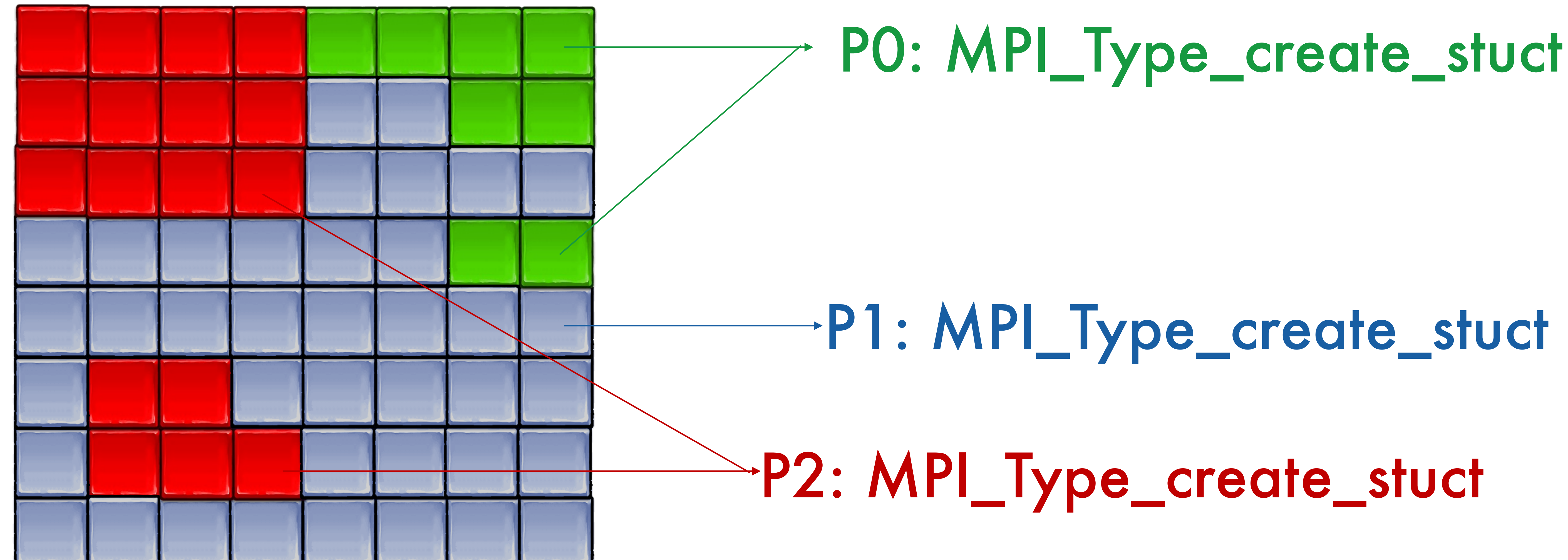
H5S_SELECT_AND

H5S_SELECT_XOR

H5S_SELECT_NOTB

H5S_SELECT_NOTA

Examples of irregular selection



Internally…

1. The HDF5 library creates an MPI datatype for each lower dimension in the selection
2. It then combines those types into one large structured MPI datatype

Example 1: Writing dataset by rows

Memory

File

P0

P1

P2

P3

Example 1: *Writing dataset by rows*

Memory

File

Process P1

count[1]

count[0]

offset[1]

offset[0]

```
count[0] = dimsf[0]/mpi_size
count[1] = dimsf[1];
offset[0] = mpi_rank * count[0];  /* = 2 */
offset[1] = 0;
```

Example 1: *Writing dataset by rows*

```
71  /*
72   * Each process defines dataset in memory and
     * writes it to the hyperslab
73   * in the file.
74   */
75   count[0] = dimsf[0]/mpi_size;
76   count[1] = dimsf[1];
77   offset[0] = mpi_rank * count[0];
78   offset[1] = 0;
79   memspace = H5Screate_simple(RANK,count,NULL);
80
81  /*
82   * Select hyperslab in the file.
83   */
84   filespace = H5Dget_space(dset_id);
85   H5Sselect_hyperslab(filespace,
         H5S_SELECT_SET,offset,NULL,count,NULL);
```

C Example: Collective write and read

```
   95  /*
   96   * Create property list for collective dataset write.
   97   */
   98  plist_id = H5Pcreate(H5P_DATASET_XFER);
 ->99  H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);
  100
  101  status = H5Dwrite(dset_id, H5T_NATIVE_INT,
  102                 memspace, filespace, plist_id, data);


  103  /*
  104   * Collective dataset read.
  105   */
  106
->107  status = H5Dread(dset_id, H5T_NATIVE_INT,
  108                 memspace, filespace, plist_id, data);
  109
```

Writing by rows: *Output of h5dump*

```
HDF5 "SDS_row.h5" {
GROUP "/" {
   DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 8, 5 ) / ( 8, 5 ) }
      DATA {
         10, 10, 10, 10, 10,
         10, 10, 10, 10, 10,
         11, 11, 11, 11, 11,
         11, 11, 11, 11, 11,
         12, 12, 12, 12, 12,
         12, 12, 12, 12, 12,
         13, 13, 13, 13, 13,
         13, 13, 13, 13, 13
      }
   }
}

}
```

General HDF5 Best Practices and Case Studies for Parallel Performance

PHDF5 Fundamentals – A Simple Problem

- 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

One Kind of Performance Hurdle

- HDF5 has a complex-looking interface
  - Complexity does not necessarily mean difficult to use
  - Users may require such complexity to achieve their goals
    - **Goal**: Self-describing share-friendly data layout
      - Tuning performance and efficiency with the constraint of using a standardized file format (netCDF, CGNS, etc.)
    - **Goal**: Fastest I/O possible
      - Tuning for check-points by minimizing metadata, large write blocks.
  - The complexity of the HDF5 workflow and underlying hardware may make the HDF5 tasks unavoidably complex.

Other Sources of Performance Variability

- Hardware
- System configuration and activity of other users

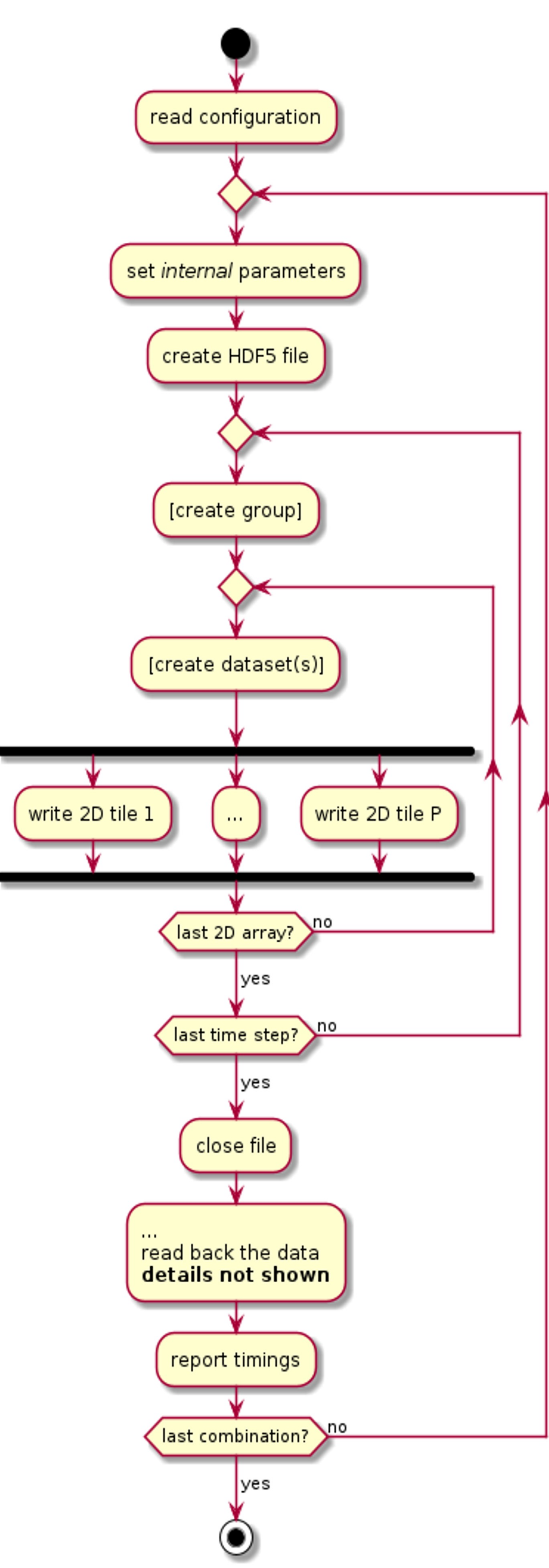
- **HDF5 property lists**
  - Nearly 180 APIs
  - Controls storage properties for HDF5 objects
  - Controls in-flight HDF5 behavior
  - About 100 *H5Pset_** functions
    - $\leq p_1 * \ldots * p_{100}$ combinations!
    - How many are tested?
  - What does *H5P_DEFAULT* mean?
  - What is the effect of using H5P_DEFAULT?

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

Back to 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

76

IO Pattern Model

Step based IO Pattern

Step1
Var1(3,4)
Var2(3,4)
Var3(3,4)
Var4(3,4)
Var 5(3,4)
Step2
Var1(3,4)
Var2(3,4)
Var3(3,4)
Var4(3,4)
Var 5(3,4)
Step1 (5, 3,4)
Step2 (5, 3,4)
Array(2, 5,3,4)

IO Pattern Model

Array based IO Pattern

Var1
Step1(3,4)
Step2(3,4)
Var2
Step1(3,4)
Step2(3,4)
Var3
Step1(3,4)
Step2(3,4)
Var4
Step1(3,4)
Step2(3,4)
Var5
Step1(3,4)
Step2(3,4)

Var 1(2,3,4)
Var2(2,3,4)
Var3(2,3,4)
Var4(2,3,4)
Var5(2,3,4)

Array(5,2, 3,4)

Performance as a function of HDF5 parameter space

independent
collective
earliest
latest
metadata
def. metadata
align
def. align
fill-true
fill-false
contiguous
chunked
step
time
Rank 4
Rank 3
Rank 2
more scalable
less scalable
1000
100
10
1
0.1
Average Total Time (s)

- Summit, weak scaling (42 to 2688)
- Best had:
  - four rank array (layout)
  - chunked
  - no fill values
  - default alignment
  - independent I/O

79

Use Case CGNS

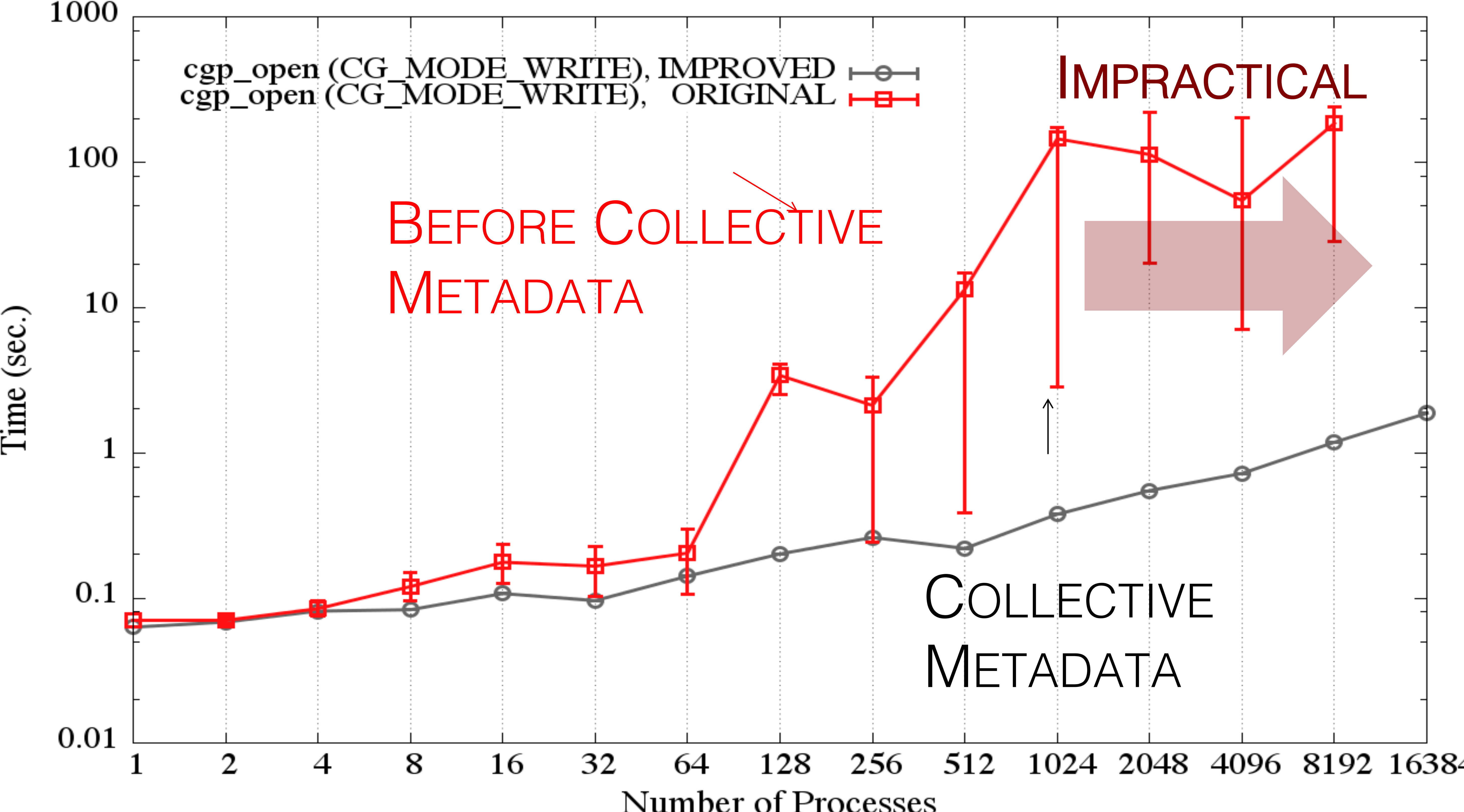
Performance tuning

- 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

Performance issue: Slow opening of an HDF5 File …

- 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”



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”);
```

HDF5 Metadata Read Storm Problem (II)

- HDF5 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
- When set, MPI rank 0 will issue the read for a metadata entry to the file system and broadcast to all other ranks

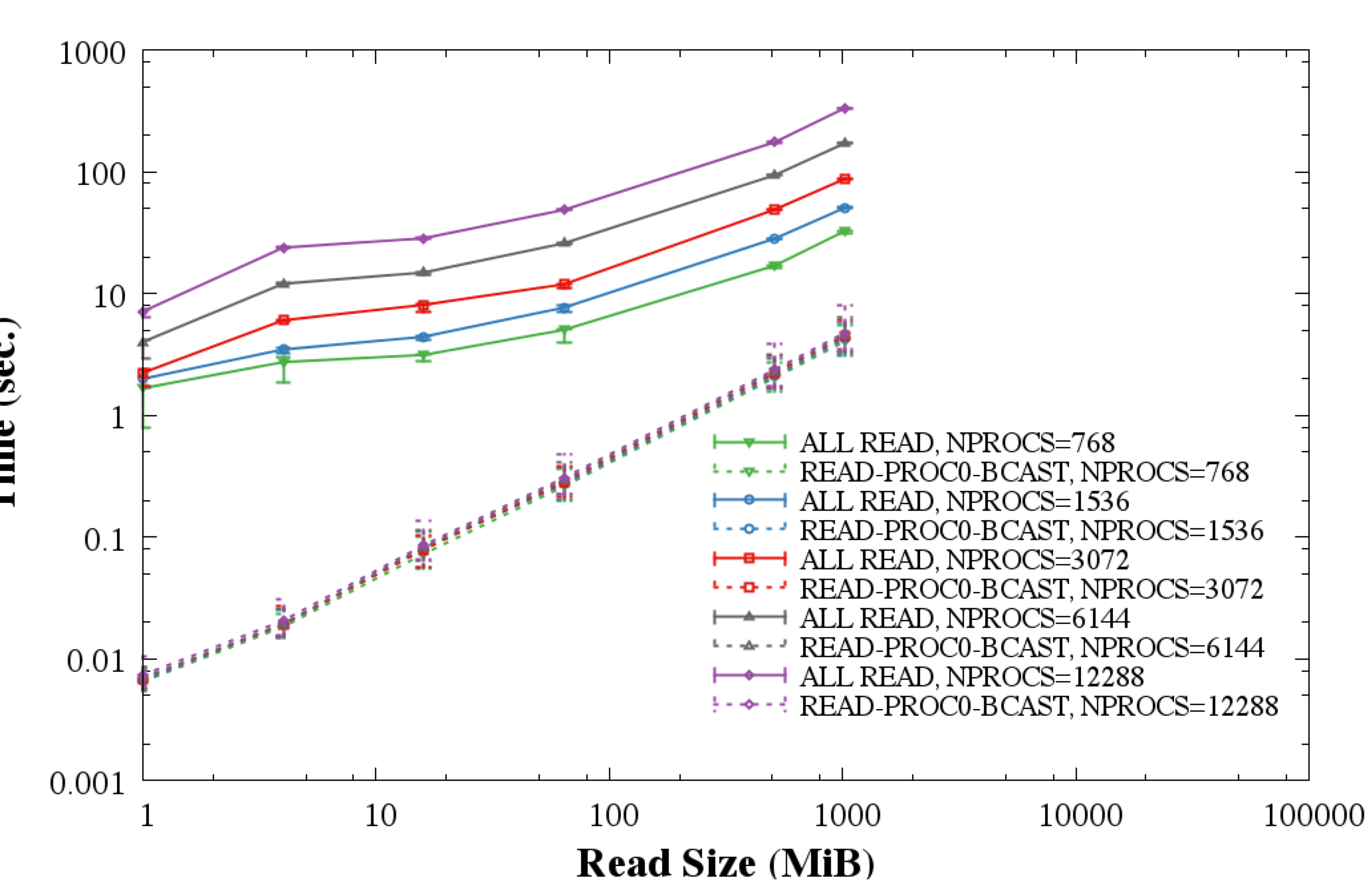
Improve the performance of reading/writing H5S_all selected datasets

(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 will write the data.

(2) Use of compact storage, or

- For compact storage, this same algorithm gets used.



SCALING OPTIMIZATIONS

The HDF Group

Time (sec.)

Execution Time (seconds)

100000
10000
1000
100
10
1
0.1
0.01
0.001

Baseline
Chart Area
Add MetaData BCast
Improved N->1
File-per-Processor (fpp)
MPI_BCast()
Serial Reference

Day
Hour
Minute
Second

MPI Ranks

32 64 128 256 512 1,024 2,048 4,096 8,192 16,384 32,768

ORIGINAL

READ-PROC0-AND-BCAST WITHIN APPLICATION

COMPACT STORAGE

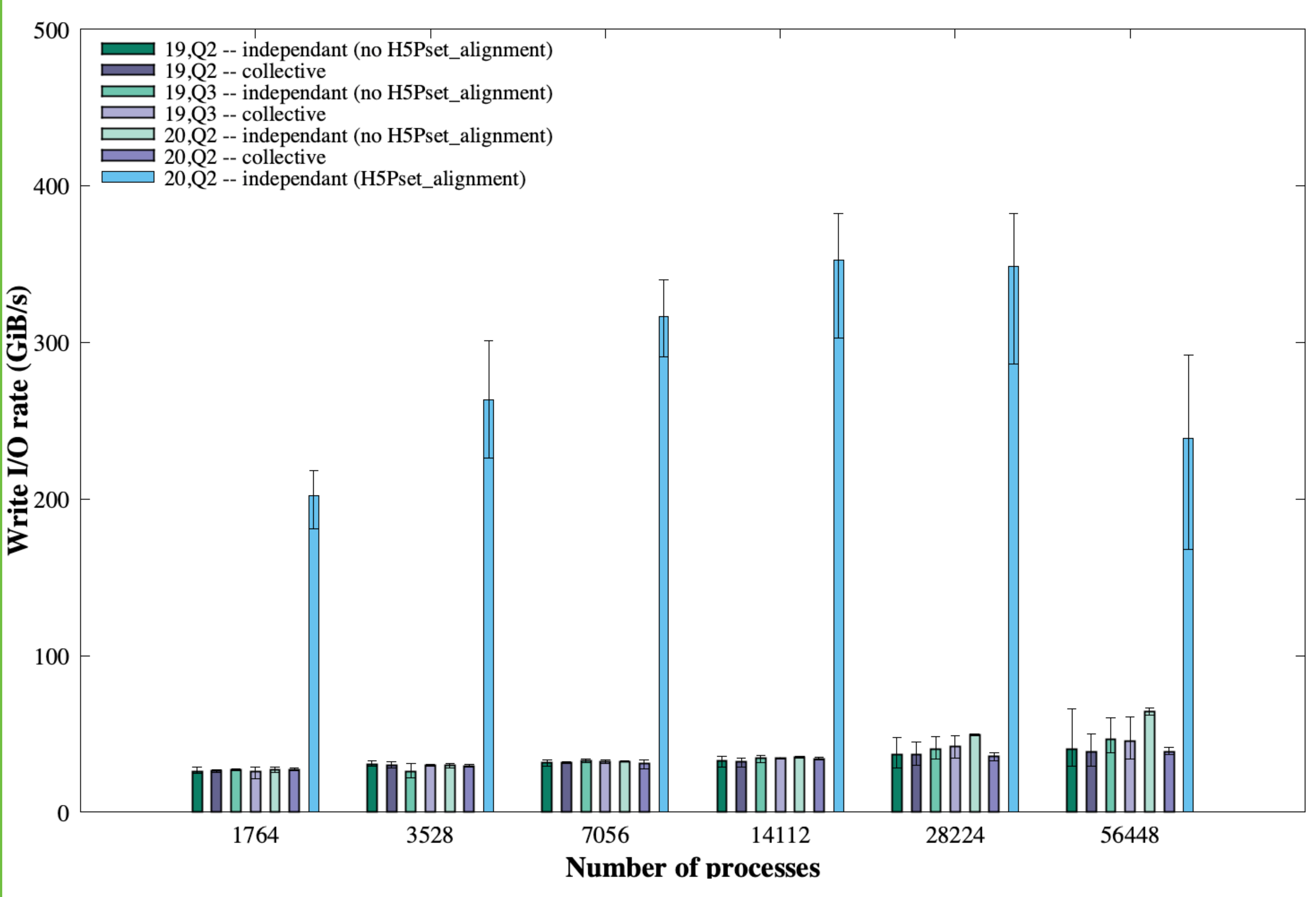
FILE-PER-PROCESS

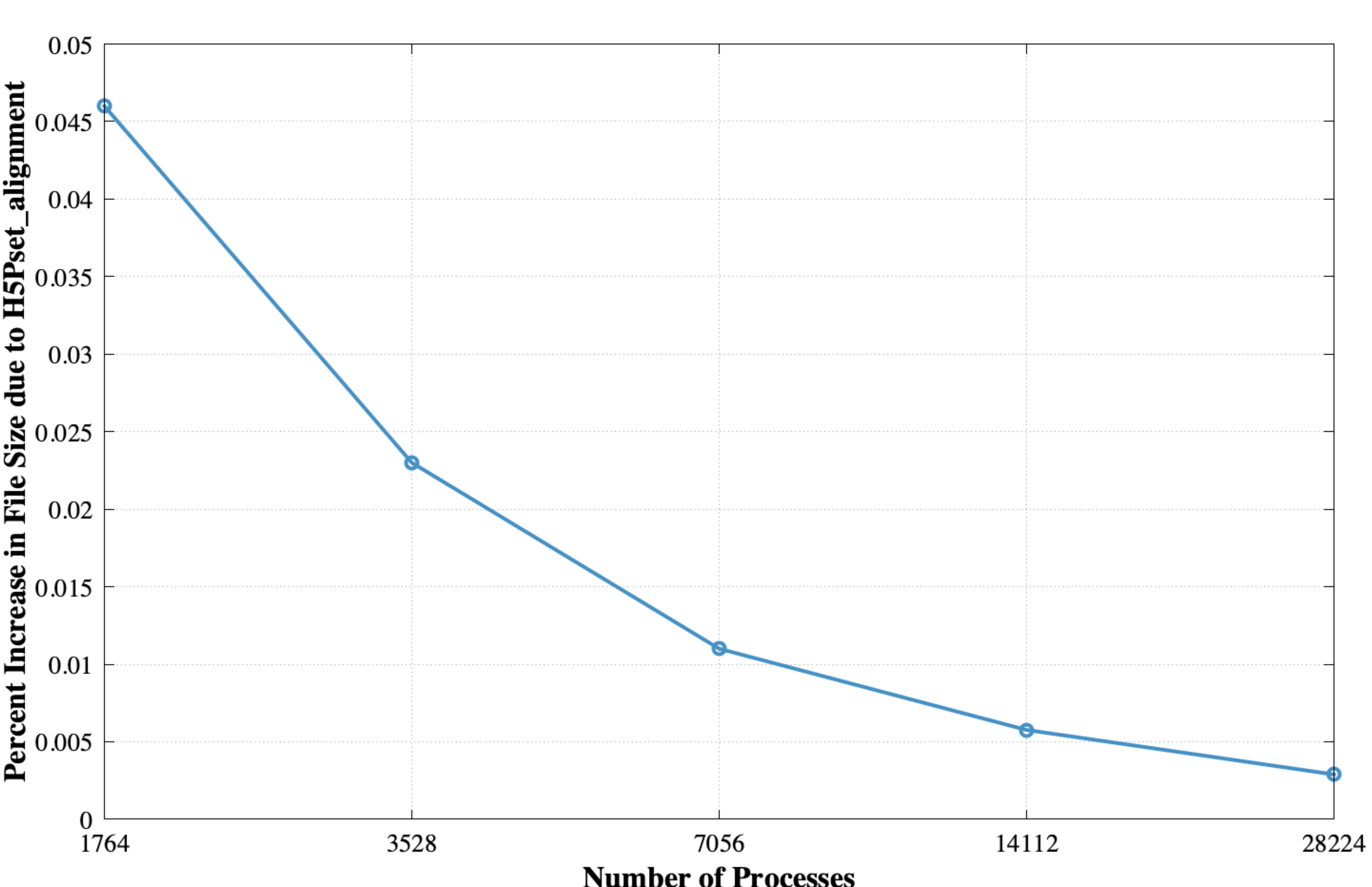
MPI_Bcast

Greg Sjaardema, Sandia National Labs

Effects of influencing object's in the file layout

- ***H5Pset_alignment*** – controls alignment of file objects on addresses.





VPIC, Summit, ORNL

How to pass hints to MPI from HDF5

- To set hints for MPI using HDF5, see: `H5Pset_fapl_mpio`
- Use the 'info' parameter to pass these kinds of low-level MPI-IO tuning tweaks.
- C Example – Controls the number of aggregators on GPFS:

```
MPI_Info info;
MPI_Info_create(&info); /* MPI hints: the key and value are strings */
MPI_Info_set(info, "bg_nodes_pset", "1");
H5Pset_fapl_mpio(plist_id, MPI_COMM_WORLD, info);
/* Pass plist_id to H5Fopen or H5Fcreate */
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
```

Diagnostics and Instrumentation Tools

I/O monitoring and profiling tools

- Two kinds of tools:
  - I/O benchmarks for measuring a system's I/O capabilities
  - I/O profilers for characterizing applications' I/O behavior
  - Profilers have to compromise between
    - A lot of detail ➔ large trace files and overhead
    - Aggregation ➔ loss of detail, but low overhead
- Examples of I/O benchmarks:
  - h5perf (in the HDF5 source code distro and binaries)
  - IOR https://github.com/hpc/ior
- Examples of profilers
  - Darshan https://www.mcs.anl.gov/research/projects/darshan/
  - Recorder https://github.com/uiuc-hpc/Recorder
  - TAU built with HDF5
    https://github.com/UO-OACISS/tau2/wiki/Configuring-TAU- to-measure-IO-libraries

"Poor Man's Debugging"

- Build a version of PHDF5 with

```
./configure --enable-build-mode=debug --enable-parallel …
```

```
setenv H5FD_mpio_Debug "rw"
```

- This allows the tracing of MPIO I/O calls in the HDF5 library such as `MPI_File_read_xx` and `MPI_File_write_xx`

- You'll get something like this…

"Poor Man's Debugging"(cont'd)
Example - Chunked by Column

```
% setenv H5FD_mpio_Debug 'rw'
% mpirun -np 4 ./a.out 1000    # Indep., Chunked by column.
in H5FD_mpio_write  mpi_off=0       size_i=96
in H5FD_mpio_write  mpi_off=0       size_i=96
in H5FD_mpio_write  mpi_off=0       size_i=96
in H5FD_mpio_write  mpi_off=0       size_i=96
in H5FD_mpio_write  mpi_off=3688    size_i=8000
in H5FD_mpio_write  mpi_off=11688   size_i=8000
in H5FD_mpio_write  mpi_off=27688   size_i=8000
in H5FD_mpio_write  mpi_off=19688   size_i=8000
in H5FD_mpio_write  mpi_off=96      size_i=40
in H5FD_mpio_write  mpi_off=136     size_i=544
in H5FD_mpio_write  mpi_off=680     size_i=120
in H5FD_mpio_write  mpi_off=800     size_i=272
…
```

HDF5 metadata (first four writes)

Dataset elements (next four writes)

HDF5 metadata (last four writes)

"Poor Man's Debugging" (cont'd)
Debugging Collective Operations

`setenv` **H5_COLL_API_SANITY_CHECK 1**

- HDF5 library will perform an MPI_Barrier() call inside each metadata operation that modifies the HDF5 namespace.
- Helps to find which rank is hanging in the MPI barrier

Use Case

Tuning PSDNS with Darshan

Darshan (ECP DataLib team)

- Design goals:
  - Transparent integration with user environment
  - Negligible impact on application performance
- Provides aggregate figures for:
  - Operation counts (POSIX, MPI-IO, HDF5, PnetCDF)
  - Datatypes and hint usage
  - Access patterns: alignments, sequentially, access size
  - Cumulative I/O time, intervals of I/O activity
- An excellent starting point

Darshan Use-Case (Blue Waters, NCSA)

- PSDNS code solves the incompressible Navier-Stokes equations in a periodic domain using pseudo-spectral methods.
- Uses custom sub-filing by collapsing the 3D in-memory layout into a 2D arrangement of HDF5 files
- Uses virtual dataset which combines the datasets distributed over several HDF5 files into a single logical dataset

Slow read times.

Ran experiments on 32,768 processes with **Darshan** 3.1.3 to create an I/O profile.

Darshan Use-Case (Blue Waters, NCSA)

…
total_POSIX_SIZE_READ_0_100: 196608
total_POSIX_SIZE_READ_100_1K: 393216
total_POSIX_SIZE_READ_1K_10K: 617472
total_POSIX_SIZE_READ_10K_100K: 32768
total_POSIX_SIZE_READ_100K_1M: 2097152
total_POSIX_SIZE_READ_1M_4M: 0
total_POSIX_SIZE_READ_4M_10M: 0
total_POSIX_SIZE_READ_10M_100M: 0
total_POSIX_SIZE_READ_100M_1G: 0
total_POSIX_SIZE_READ_1G_PLUS: 0
…

Large numbers of reads of only small amounts of data.

Multiple MPI ranks independently read data from a small restart file which contains a virtual dataset.

Darshan Use-Case (Blue Waters, NCSA)

"Broadcast" the restart file:

1. Rank 0: read the restart file as a byte stream into a memory buffer.
2. Rank 0: broadcasts the buffer.
3. All MPI ranks open the buffer as an HDF5 *file image,* and proceed as if they were performing reads against an HDF5 file stored in a file system.

Eliminates the "read storm",

….

total_POSIX_SIZE_READ_0_100: 6
total_POSIX_SIZE_READ_100_1K: 0
total_POSIX_SIZE_READ_1K_10K: 0
total_POSIX_SIZE_READ_10K_100K: 2
total_POSIX_SIZE_READ_100K_1M: 0
total_POSIX_SIZE_READ_1M_4M: 0
total_POSIX_SIZE_READ_4M_10M: 0
total_POSIX_SIZE_READ_10M_100M: 0
total_POSIX_SIZE_READ_100M_1G: 32768
total_POSIX_SIZE_READ_1G_PLUS: 0

…

Use Case

Tuning HACC (Hardware/Hybrid Accelerated Cosmology Code) with Recorder

Write Pattern Effects – Data location in the file

Pattern 1 – HDF5 pattern

Variables are **contiguously** stored in the file

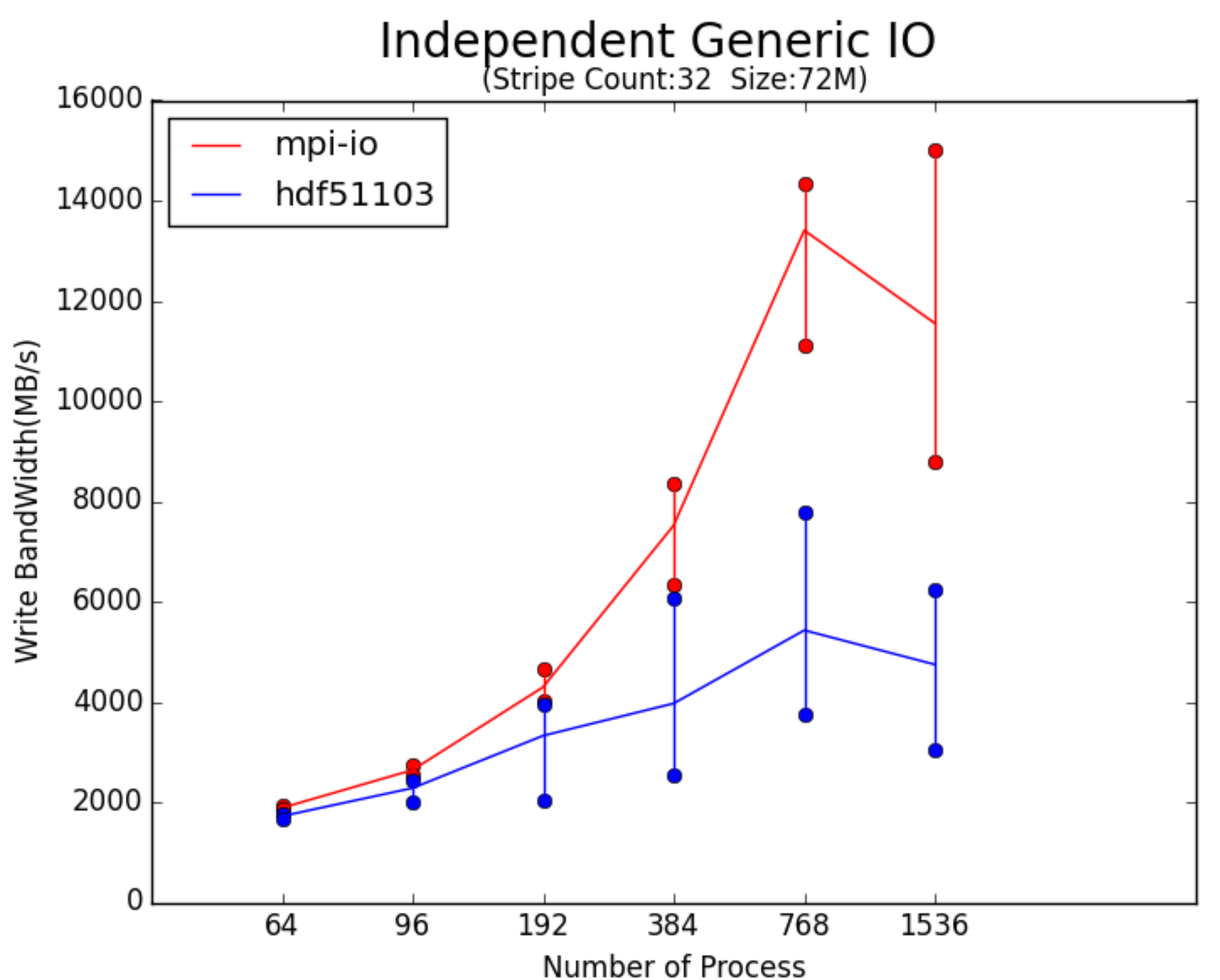
Pattern 2 – MPI-IO pattern (or HDF5 compound datatype)

Variables are **interleaved** in the file

Case Study – Data Layout Effects

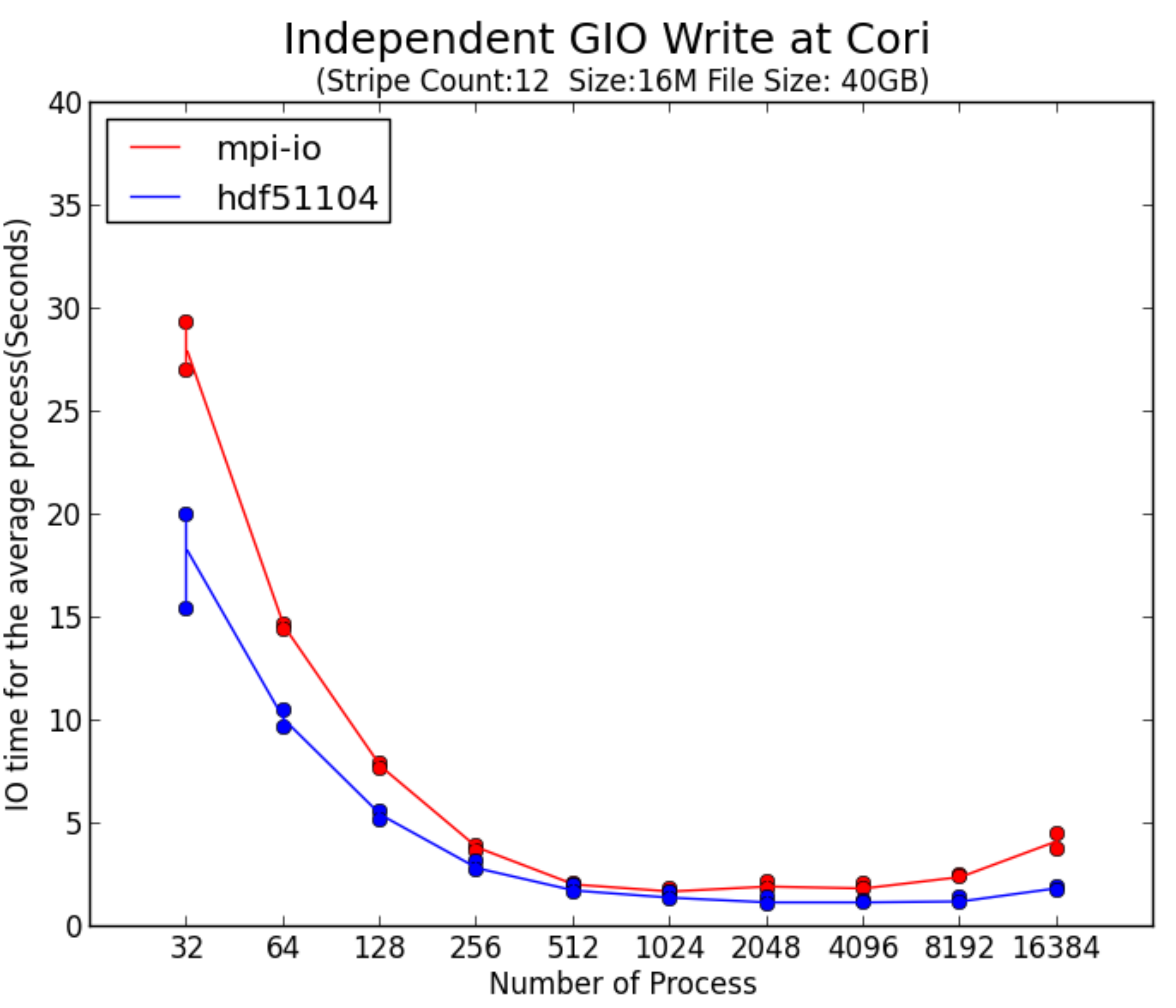
Benchmark:

- 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.



HDF5 Pattern 2 Implementation

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



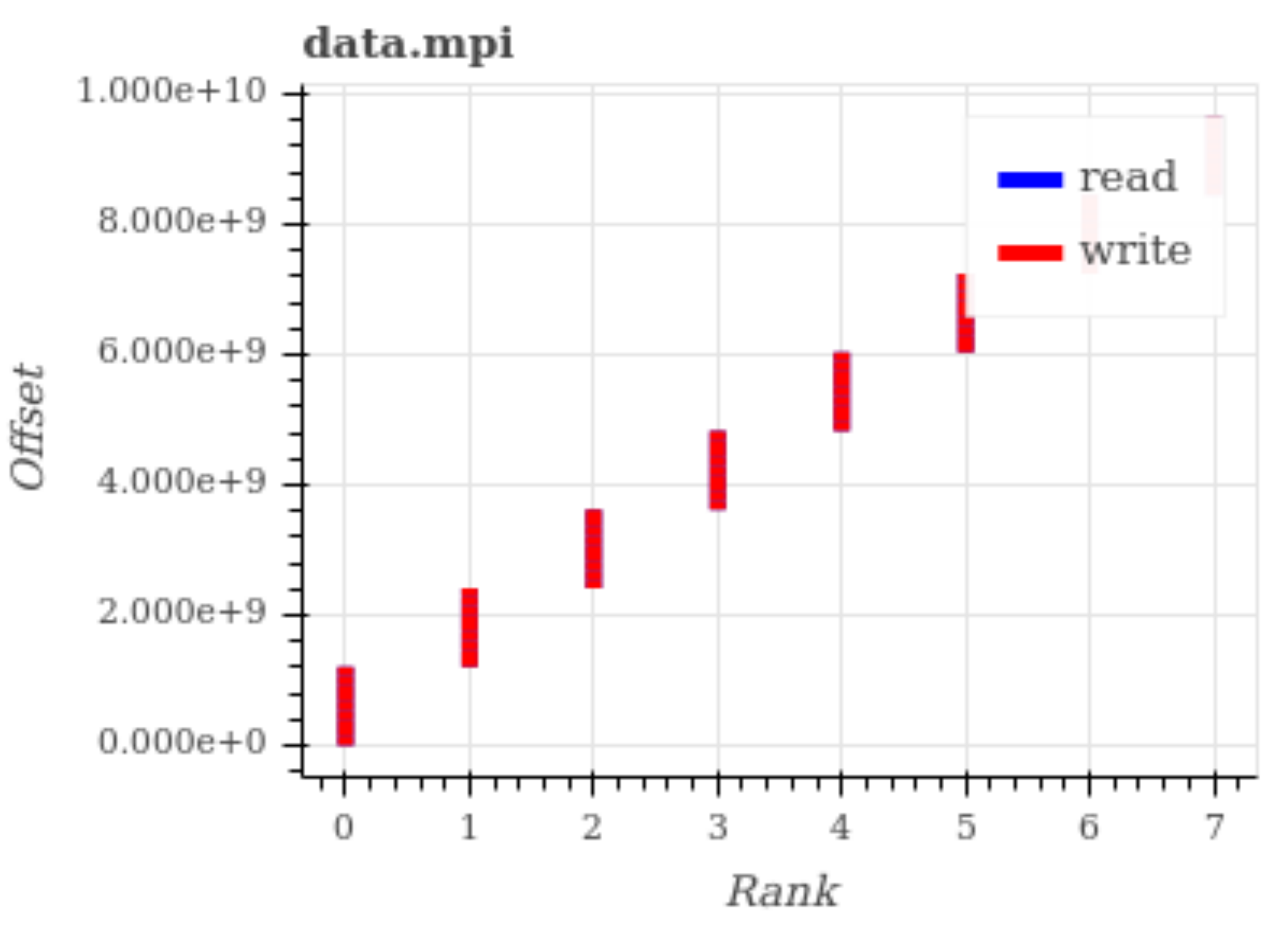
Recorder

- Multi-level I/O tracing library that captures function calls from HDF5, MPI and POSIX.
- It keeps every function and its parameters. Useful to exam access patterns.
- Built-in visualizations for access patterns, function counters, I/O sizes, etc.
- Also reports I/O conflicts such as write-after-write, write-after-read, etc. Useful for consistency semantics check (File systems with weaker consistency semantics).

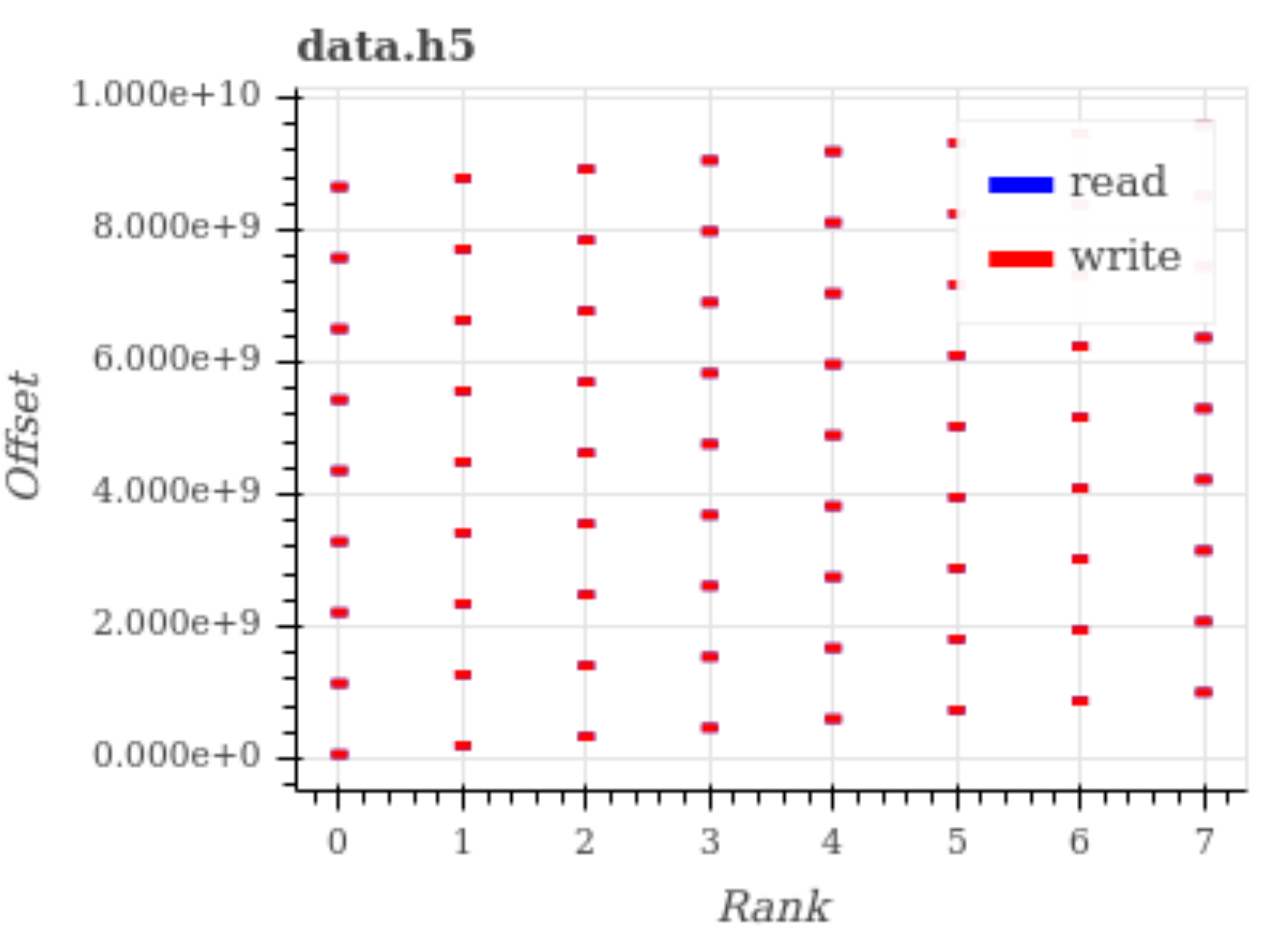
Wang, Chen, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. "Recorder 2.0: Efficient Parallel I/O Tracing and Analysis." In IEEE International Workshop on High-Performance Storage (HPS), 2020.
https://github.com/uiuc-hpc/Recorder

HACC-IO: MPI vs HDF5 – Analysis

Example of access patterns with 8 ranks writing 9GB.



MPI-IO Access Pattern



HDF5 with individual dataset

For a full presentation and related paper [1], covering an analysis of

- The effects of access patterns (independent and collective)
- Example of the analysis needed to match the performance of pure MPI IO
  - Metadata considerations

[1] https://www.hdfgroup.org/2020/08/a-study-of-hacc-io-benchmarks/

Need Help?

HDF-FORUM – https://forum.hdfgroup.org/

HDF Helpdesk – help@hdfgroup.org

Call the Doctor – Weekly HDF Clinic

https://zoom.us/meeting/register/tJwvf--gpjsqEtV0NSexRspn0NUjcNhZFmFb

THANK YOU!

Questions & Comments?