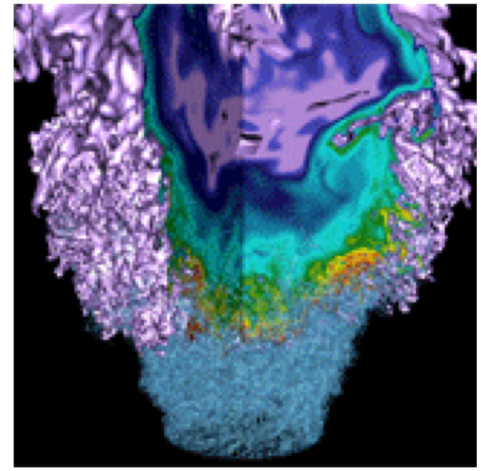
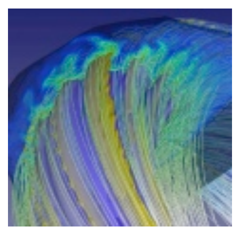
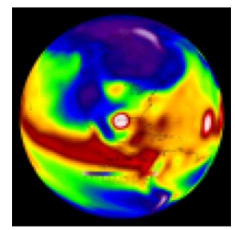
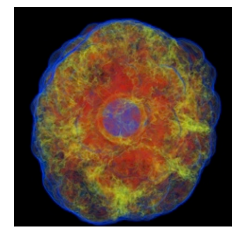
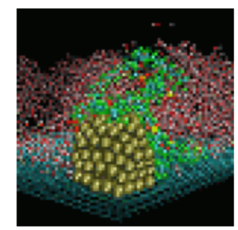
Scalable HDF5











Quincey Koziol

ATPESC

August 3, 2018

koziol@lbl.gov

Why HDF5?

- **Have you ever asked yourself:**
  - How will I deal with one-file-per-processor in the petascale era?
  - Do I need to be an “MPI and Lustre pro” to do my research?
  - Where is my checkpoint file?
- **HDF5 hides all complexity so you can concentrate on Science**
  - Optimized I/O to single shared file*

* Explorations for “multi-file” HDF5 storage are underway as well.

Goal

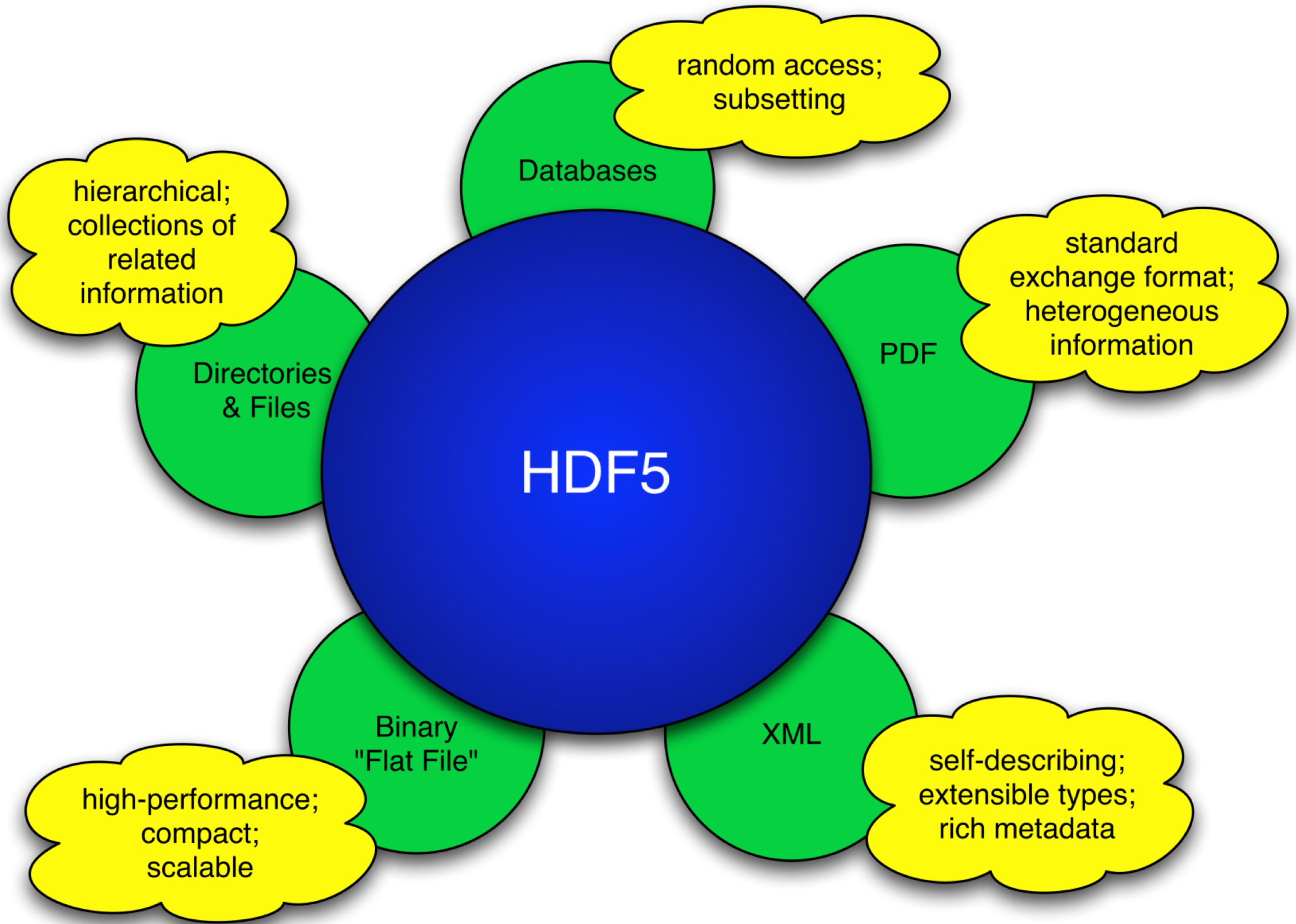
- **Introduce you to HDF5**
  - HDF5 Overview
  - HDF5 Programming Overview
  - Intro to Scalable HDF5

WHAT IS HDF5?

What is HDF5?

- **HDF5 == Hierarchical Data Format, v5**
- **Open file format**
  - Designed for high volume or complex data
- **Open source software**
  - Works with data in the format
- **An extensible data model**
  - Structures for data organization and specification

HDF5 is like …

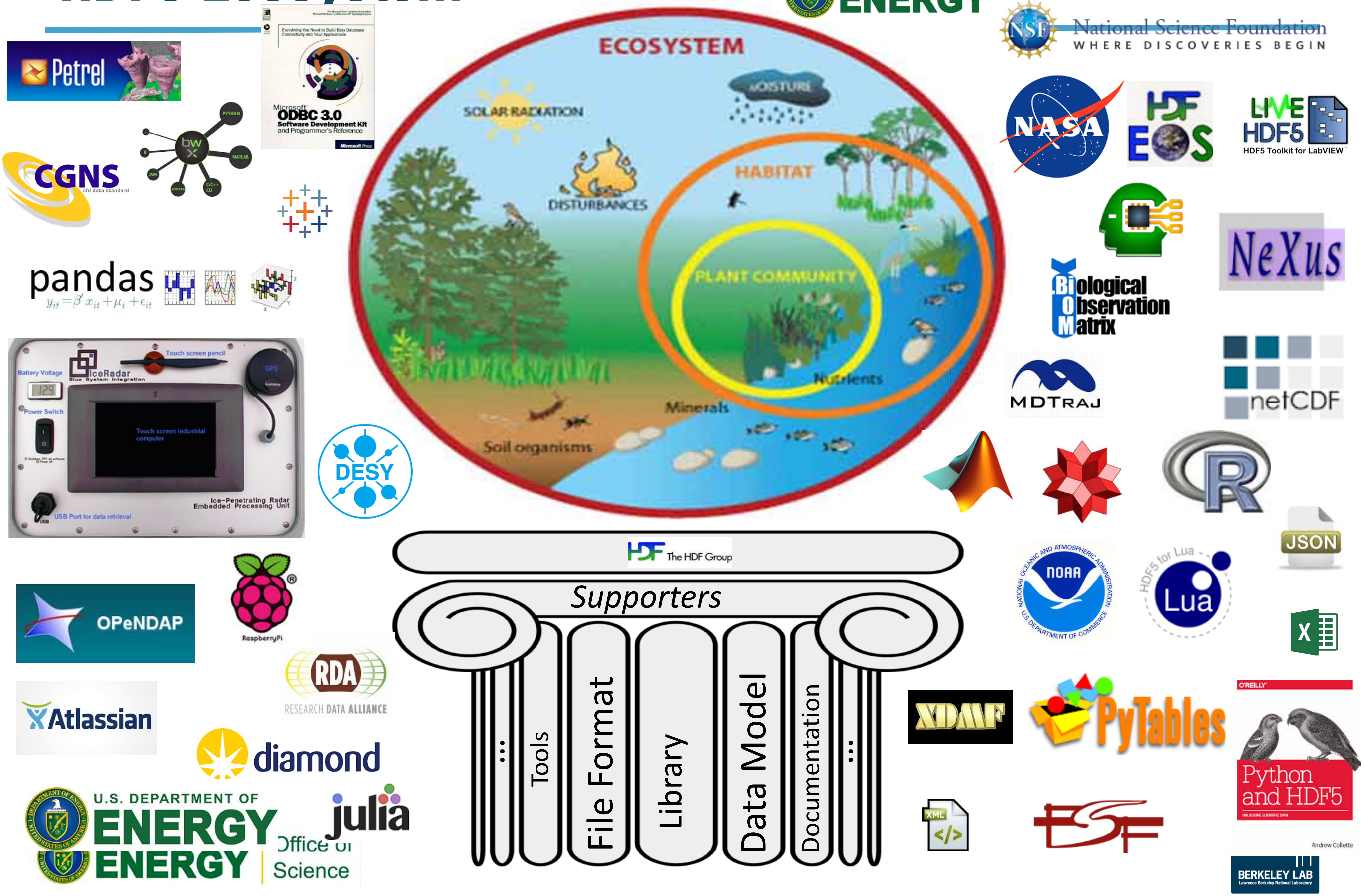


random access;
subsetting
Databases
hierarchical;
collections of
related
information
Directories
& Files
standard
exchange format;
heterogeneous
information
PDF
HDF5
Binary
"Flat File"
XML
high-performance;
compact;
scalable
self-describing;
extensible types;
rich metadata

HDF5 is designed …

- **for high volume and/or complex data**
- **for every size and type of system (portable)**
- **for flexible, efficient storage and I/O**
- **to enable applications to evolve in their use of HDF5 and to accommodate new models**
- **to support long-term data preservation**

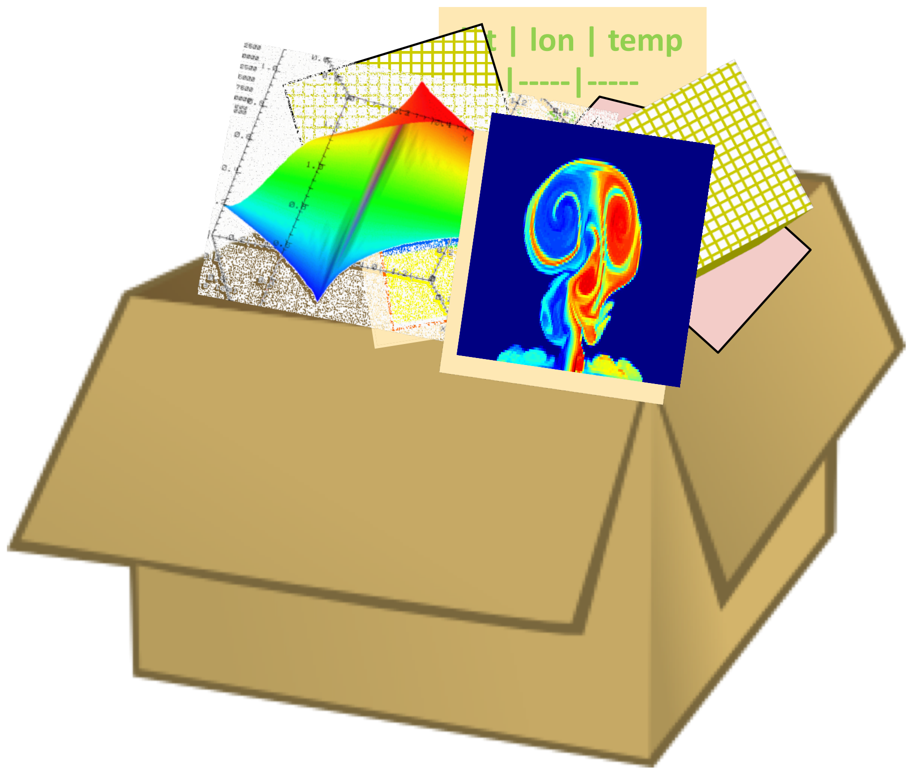
HDF5 Ecosystem



HDF5 DATA MODEL

HDF5 File

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



HDF5 Data Model

Dataset

Link

Group

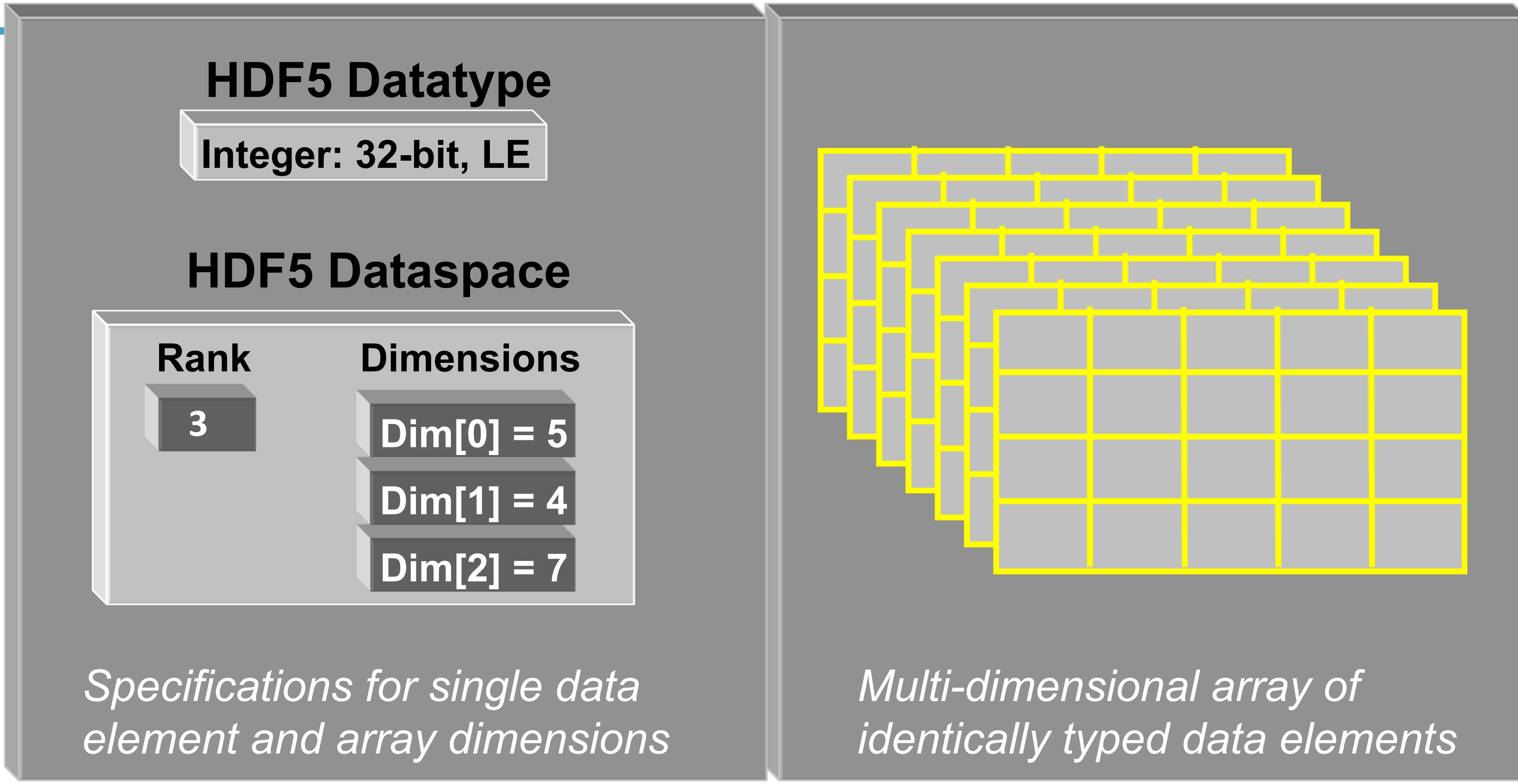
Datatype

Attribute

Dataspace

File

HDF5 Dataset



- 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

- **Describes the logical layout of the elements in an HDF5 dataset**
  - NULL
    - no elements
  - Scalar
    - single element
  - Simple array (*most common*)
    - multiple elements organized in a rectangular array
      - rank = number of dimensions
      - dimension sizes = number of elements in each dimension
      - maximum number of elements in each dimension
        - » may be fixed or unlimited

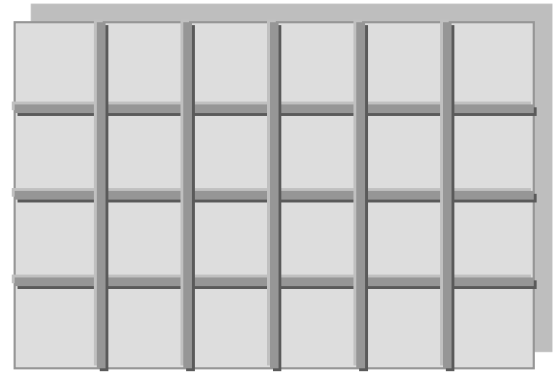
HDF5 Dataspace

Two roles:

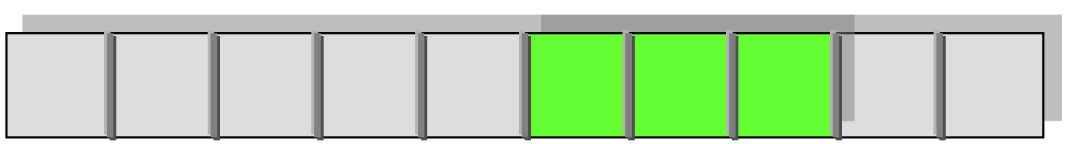
Spatial information for Datasets and Attributes

– Rank and dimensions

– Permanent part of object definition



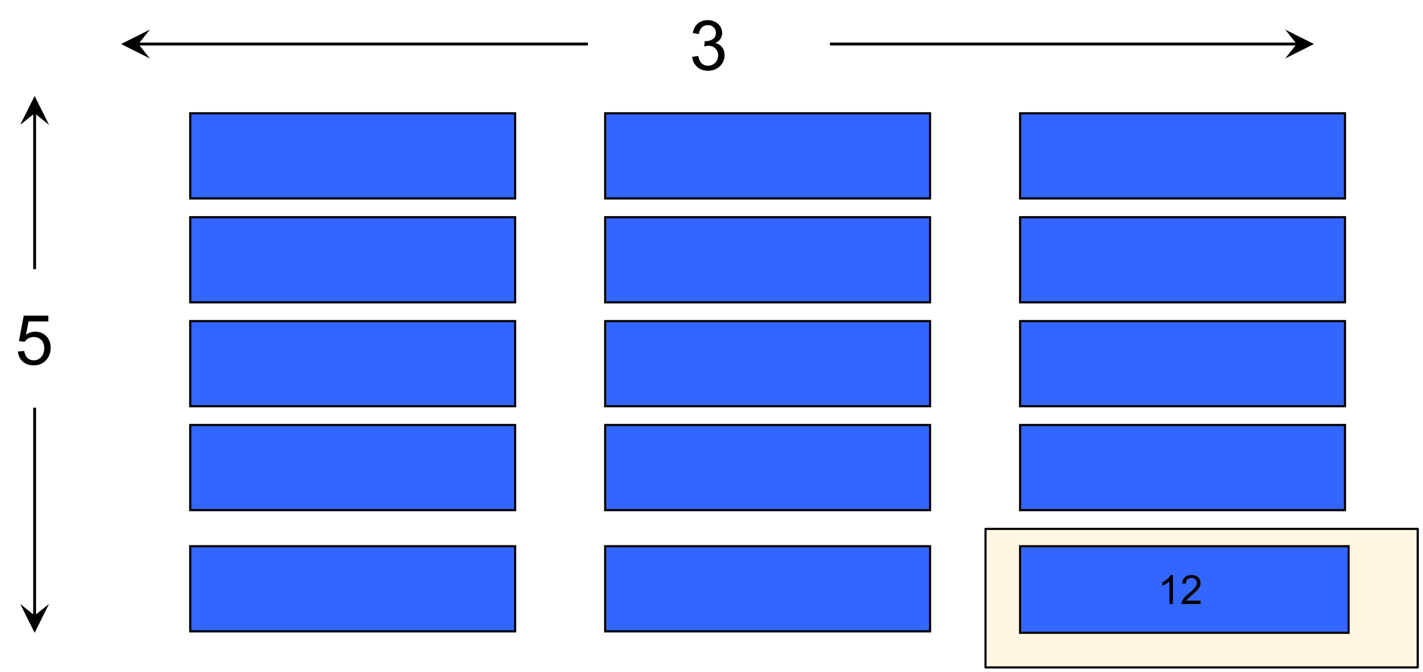
Partial I/O: Dataspace and selection describe application’s data buffer and data elements participating in I/O



HDF5 Datatypes

- **Describe individual data elements in an HDF5 dataset**
- **Wide range of datatypes supported**
  - Integer
  - Float
  - Enum
  - Array (similar to matrix)
  - User-defined (e.g., 12-bit integer, 16-bit float)
  - Variable-length types (e.g., strings, vectors)
  - Compound (similar to C structs)
  - More …

HDF5 Dataset

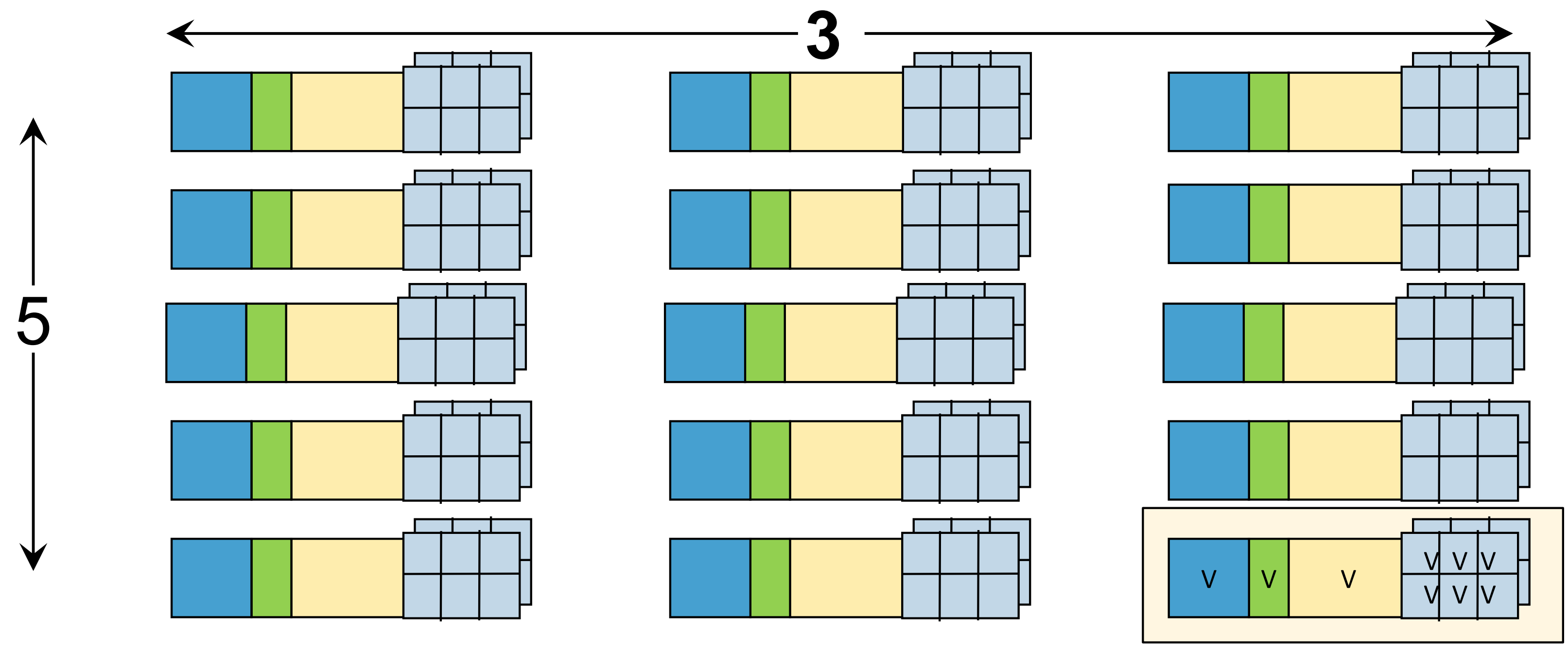


Datatype: 32-bit Integer

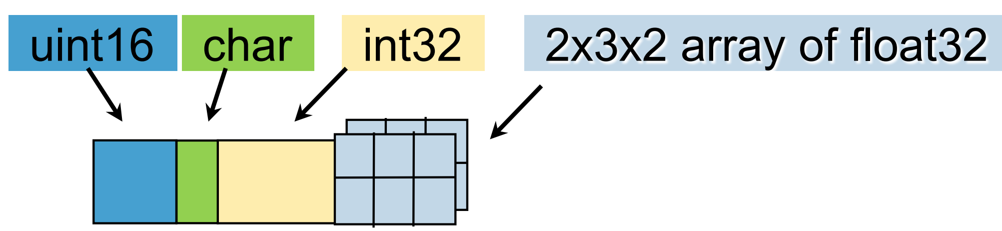
Dataspace: Rank = 2
Dimensions = 3 x 5

Note that this is declared in C as: "array[5][3]" and as "array(3)(5)" in FORTRAN

HDF5 Dataset with Compound Datatype



Compound Datatype:

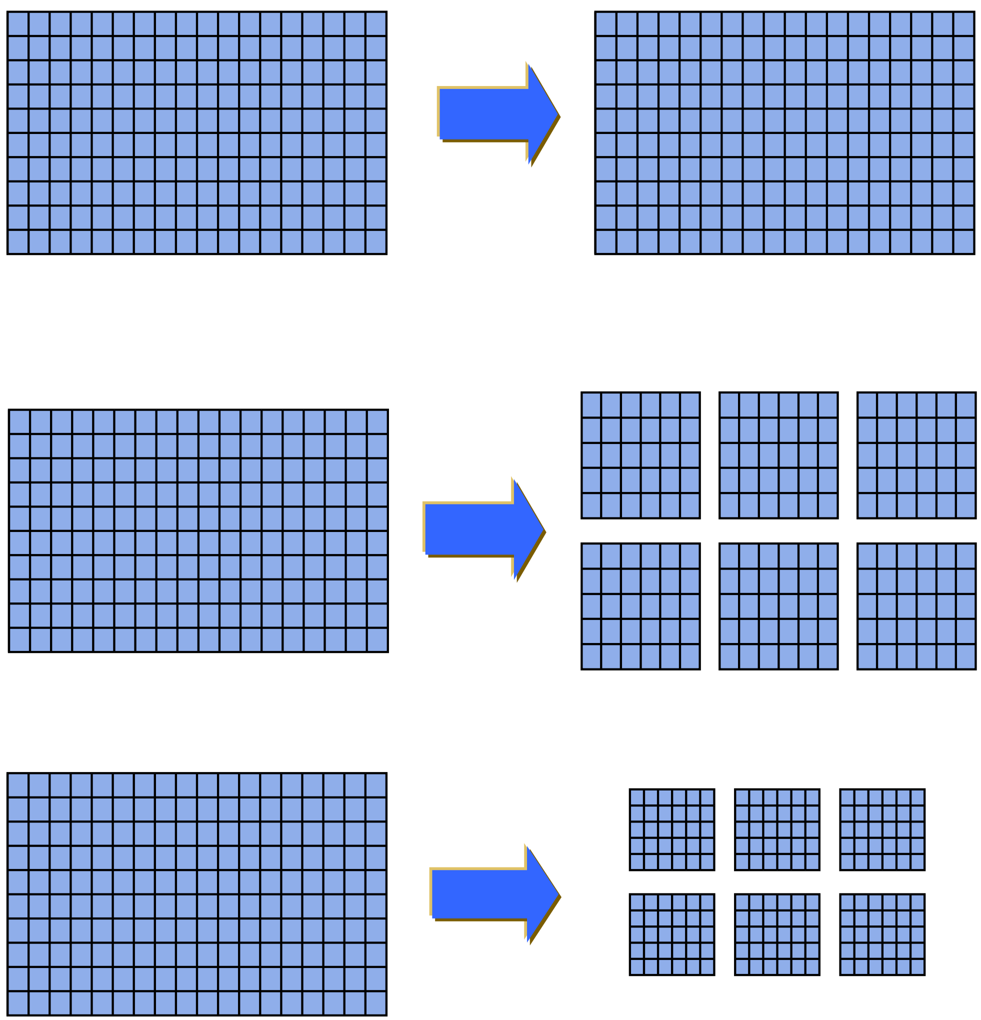


Dataspace: Rank = 2

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

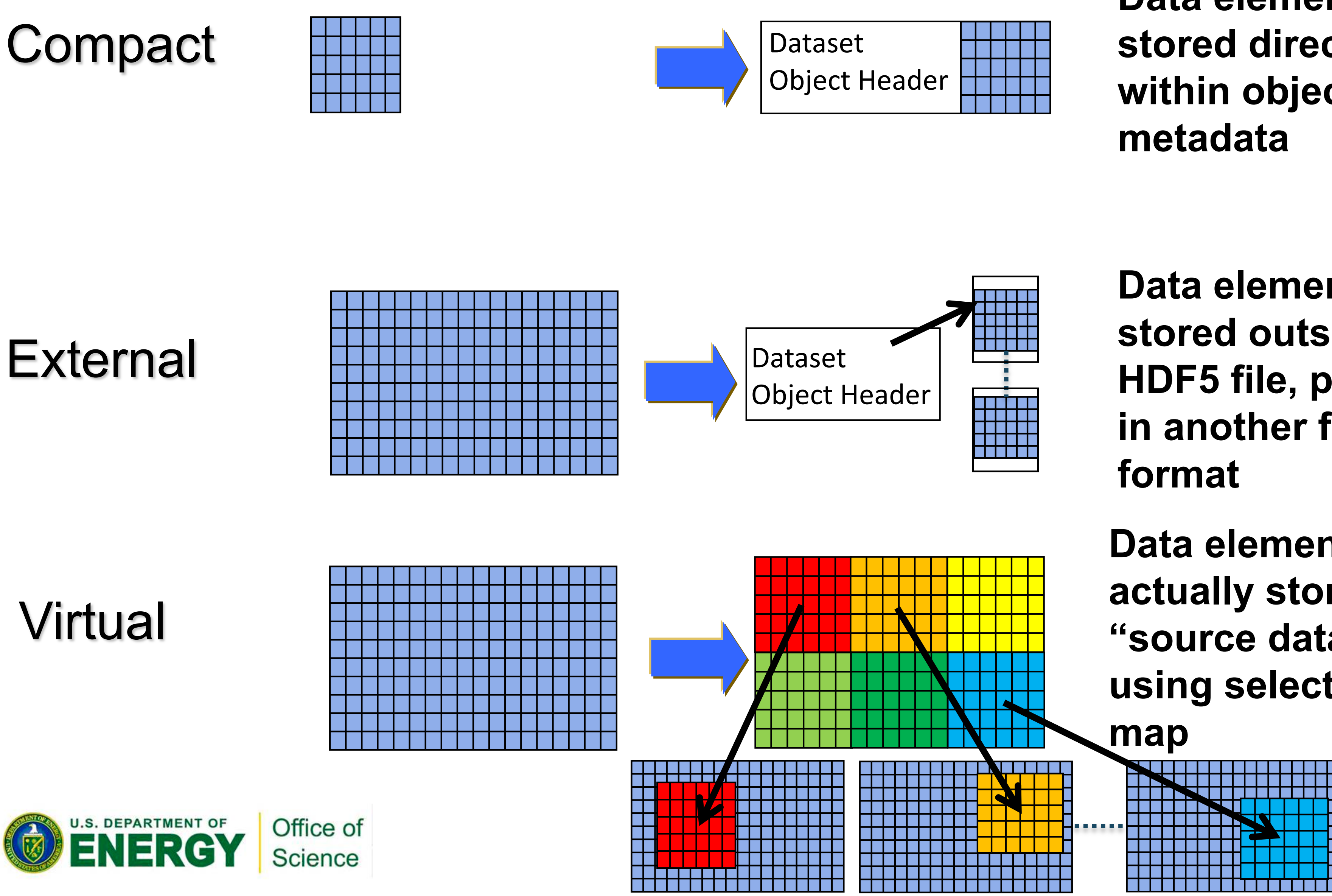
Chunked & Compressed
Improves storage efficiency, transmission speed

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

How are data elements stored? (2/2)

Buffer in memory | Data in the file



Data elements stored directly within object's metadata

Data elements stored outside the HDF5 file, possibly in another file format

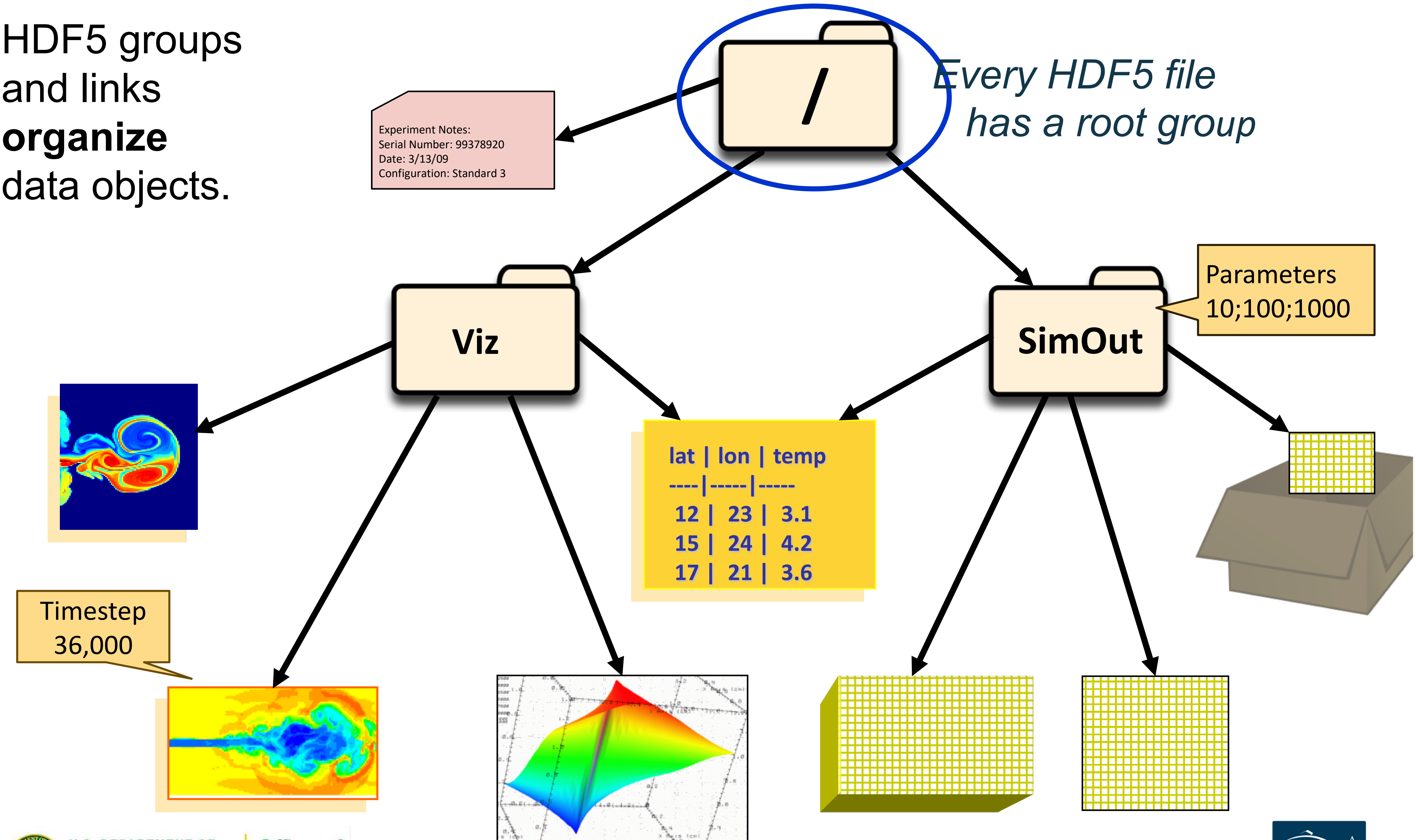
Data elements actually stored in "source datasets", using selections to map

HDF5 Attributes

- **Attributes "decorate" HDF5 objects**
- **Typically contain *user* metadata**
- **Similar to Key-Values:**
  - Have a unique name (for that object) and a value
- **Analogous to a dataset**
  - "Value" is an array 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.



HDF5 SOFTWARE

HDF5 Home Page

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

- Latest release: HDF5 1.10.2 (1.10.2 coming August 2018)

HDF5 source code:

- Written in C, and includes optional C++, Fortran APIs, and High Level APIs
- Contains command-line utilities (h5dump, h5repack, h5diff, ..) and compile scripts

HDF5 pre-built binaries:

- When possible, include C, C++, Fortran, and High Level libraries. Check ./lib/libhdf5.settings file.
- Built with and require the SZIP and ZLIB external libraries

Useful Tools For New Users

`h5dump`:

Tool to "dump" or display contents of HDF5 files

`h5cc, h5c++, h5fc`:

Scripts to compile applications

HDFView:

Java browser to view HDF5 files

http://www.hdfgroup.org/hdf-java-html/hdfview/

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

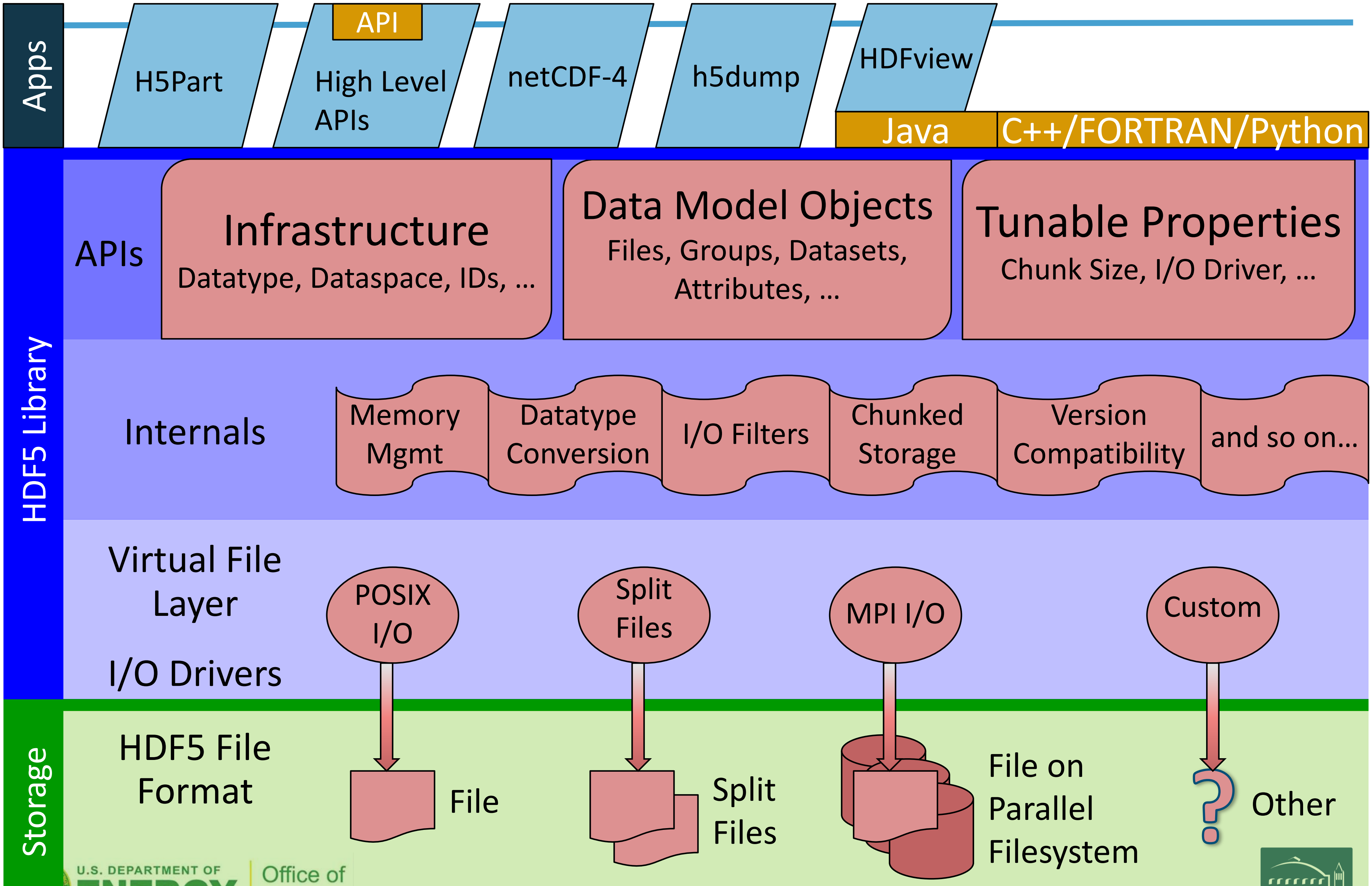
https://www.hdfgroup.org/HDF5/examples/

HDF5 PROGRAMMING MODEL AND API

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

HDF5 Software Layers & Storage



The General HDF5 API

- **C, FORTRAN, Java, C++, and .NET bindings**
- **IDL, MATLAB, Python (H5Py, PyTables)**
- **C routines begin with prefix: H5?**

  ? is a character corresponding 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 HDF5 API

- **For flexibility, the API is extensive**
  - ✓ 300+ functions

Victorinox Swiss Army Cybertool 34

- **This can be daunting… but there is hope**
  - ✓ A few functions can do a lot
  - ✓ Start simple
  - ✓ Build up knowledge as more features are needed

General Programming Paradigm

- **Object is opened or created**
- **Object is accessed, possibly many times**
- **Object is closed**

- **Properties of object are <u>optionally</u> defined**
  - ✓ Creation properties (e.g., use chunking storage)
  - ✓ Access properties

Basic Functions

H5Fcreate (H5Fopen)	***create (open) File***
H5Screate_simple/H5Screate	***create dataSpace***
H5Dcreate (H5Dopen)	***create (open) Dataset***
H5Dread, H5Dwrite	***access Dataset***
H5Dclose	***close Dataset***
H5Sclose	***close dataSpace***
H5Fclose	***close File***

Other Common Functions

DataSpaces:	**H5Sselect_hyperslab (Partial I/O)** **H5Sselect_elements (Partial I/O)** **H5Dget_space**
DataTypes:	**H5Tcreate, H5Tcommit, H5Tclose** **H5Tequal, H5Tget_native_type**
Groups:	**H5Gcreate, H5Gopen, H5Gclose**
Attributes:	**H5Acreate, H5Aopen_name,** **H5Aclose, H5Aread, H5Awrite**
Property lists:	**H5Pcreate, H5Pclose** **H5Pset_chunk, H5Pset_deflate**

Tools

PARALLEL HDF5

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

Why Parallel HDF5?

- **Take advantage of high-performance parallel I/O while reducing complexity**
  - Add a well-defined layer to the I/O stack
  - Keep the dream of a single or a few shared files alive
  - “Friends don’t let friends use one file per process!”
- **Make performance portable**

What We'll Cover Here

- **Parallel vs. serial HDF5**
- **Implementation layers**
- **HDF5 files (= composites of data & metadata) in a parallel file system**
- **PHDF5 I/O modes: collective vs. independent**
- **Data and metadata I/O**

What We Won't Cover

- **Consistency semantics**
- **Virtual Object Layer (VOL)**
- **Automatic tuning**
- **Single Writer / Multiple-Reader (SWMR)**
- **Virtual Datasets (VDS)**
- **Asynchronous I/O**
- **Independent Metadata Modification**
- **…**

Come see me this evening or after the presentation!

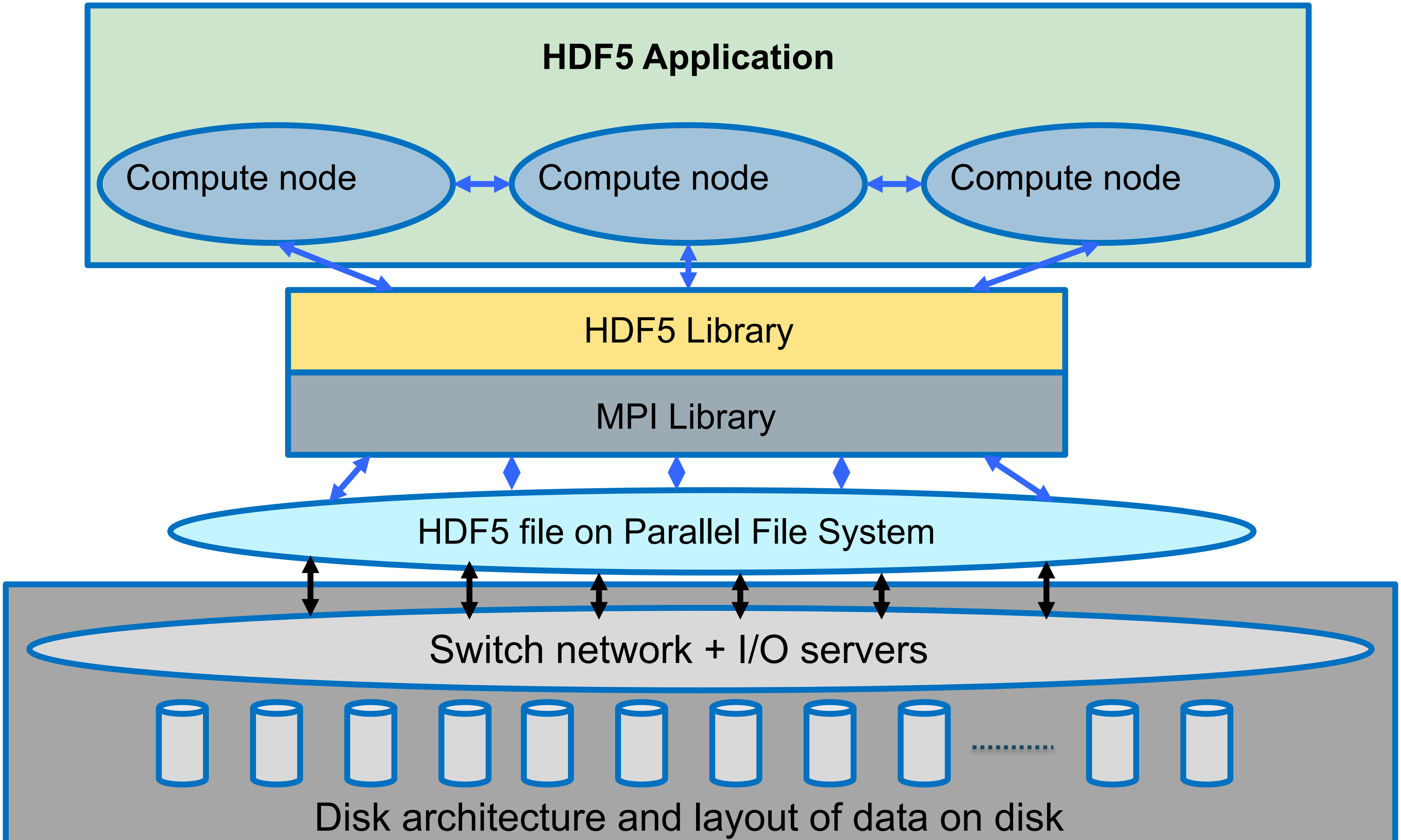
(MPI-)Parallel vs. Serial HDF5

- **PHDF5 allows multiple MPI processes in an MPI application to perform I/O to a single HDF5 file**
- **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 “etiquette”

Standard HDF5 "Skeleton"

H5Fcreate (H5Fopen)	create (open) File
H5Screate_simple/H5Screate	create dataSpace
H5Dcreate (H5Dopen)	create (open) Dataset
H5Dread, H5Dwrite	access Dataset
H5Dclose	close Dataset
H5Sclose	close dataSpace
H5Fclose	close File

PHDF5 Implementation Layers



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();
```

PHDF5 Etiquette

- PHDF5 opens a shared file with an MPI communicator
- Returns a file handle
- All future access to the file via that file handle
- All processes must participate in collective PHDF5 APIs
- Different files can be opened via different communicators
- All HDF5 APIs that modify structural metadata are collective! (file ops., object structure and life-cycle)

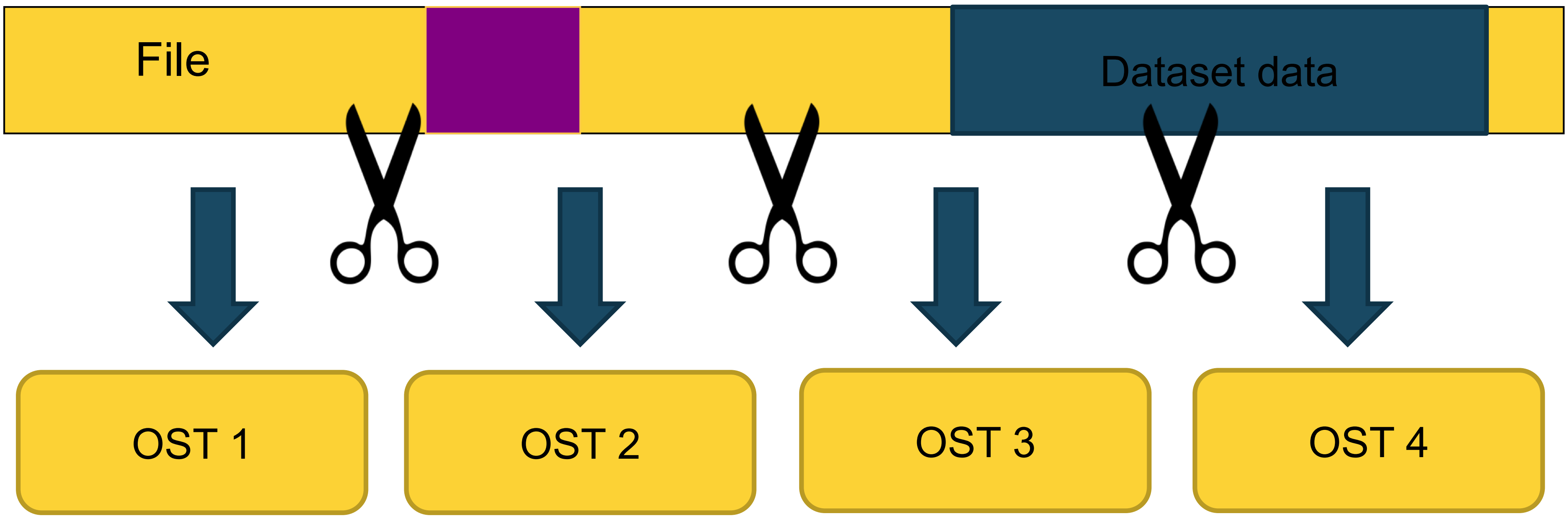
https://www.hdfgroup.org/HDF5/doc/RM/CollectiveCalls.html

Parallel HDF5 tutorial examples

- **For simple examples how to write different data patterns see**

http://www.hdfgroup.org/HDF5/Tutor/parallel.html

In a Parallel File System

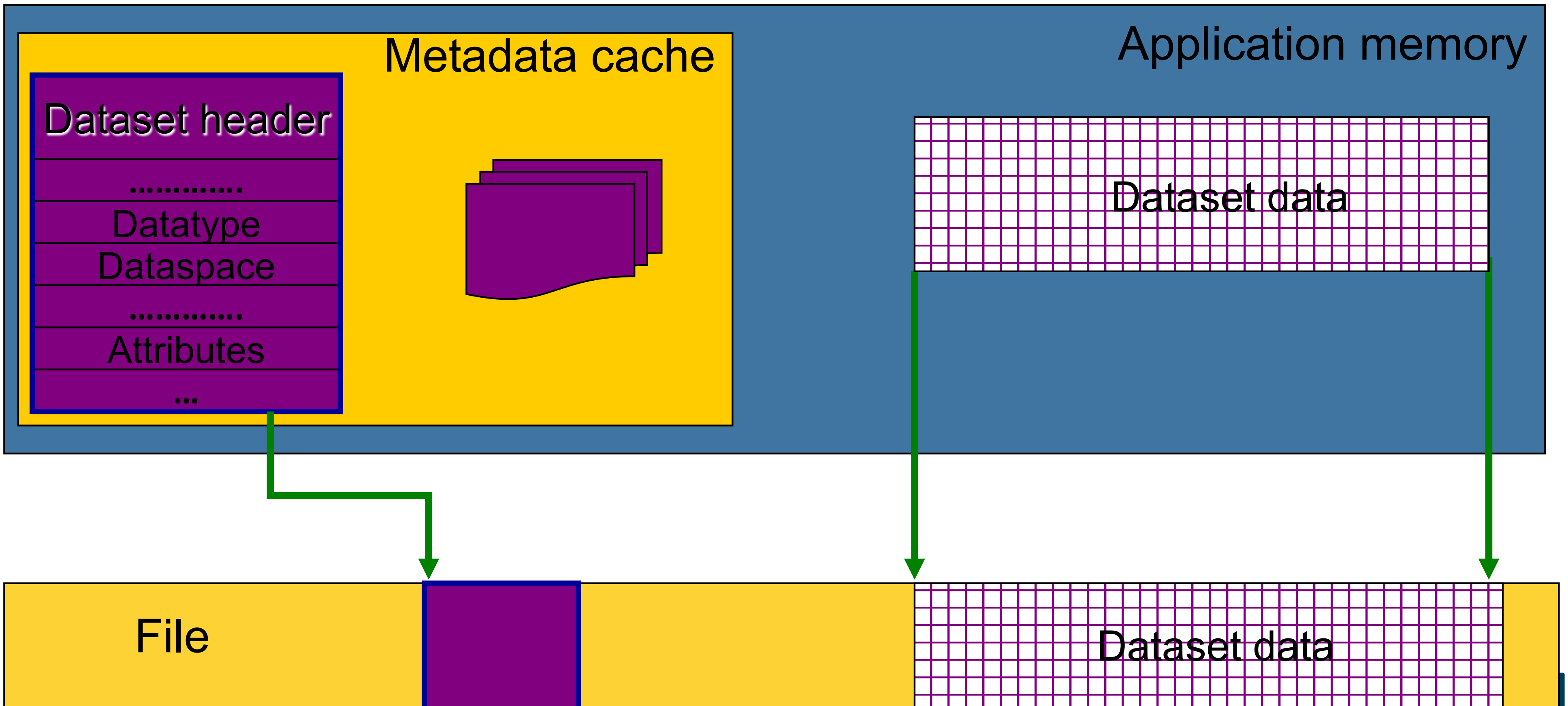


The file is striped over multiple “disks” (e.g. Lustre OSTs) depending on the stripe size and stripe count with which the file was created.

And it gets worse before it gets better…

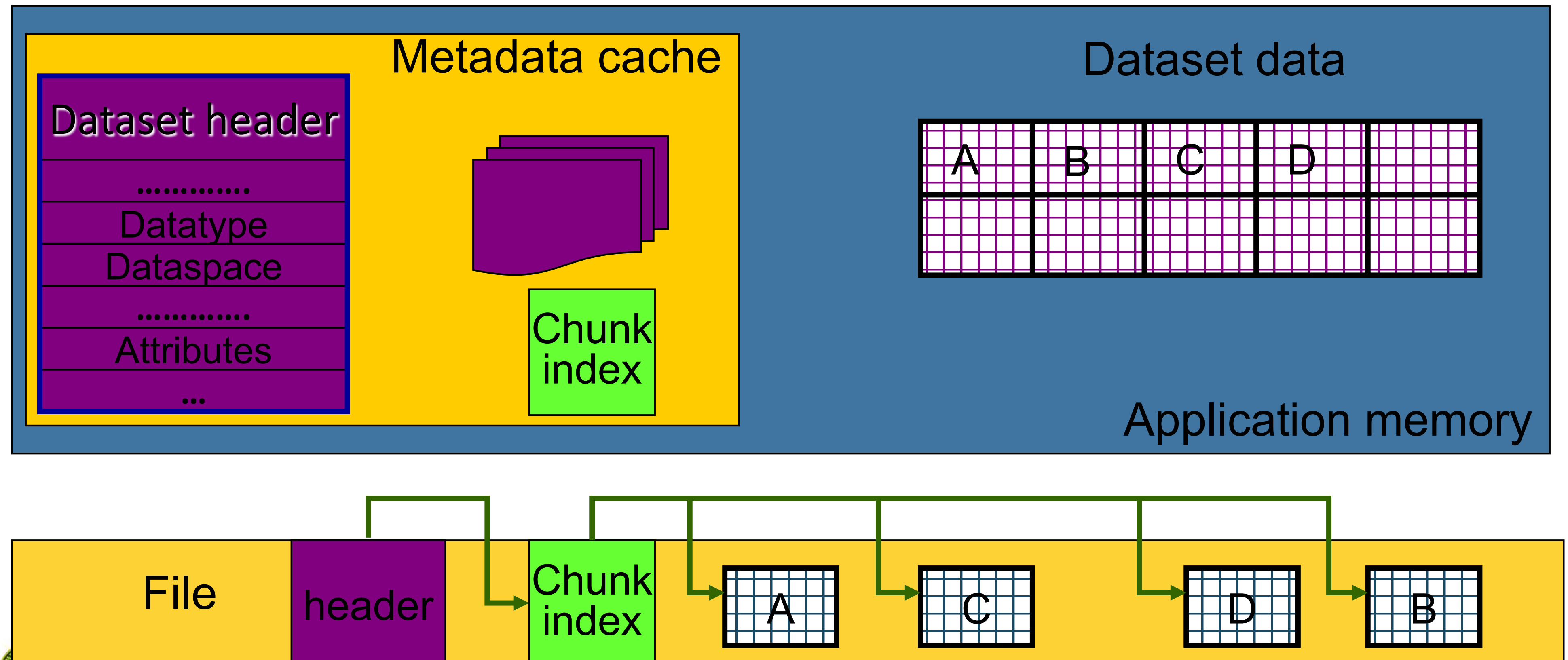
Contiguous Storage

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

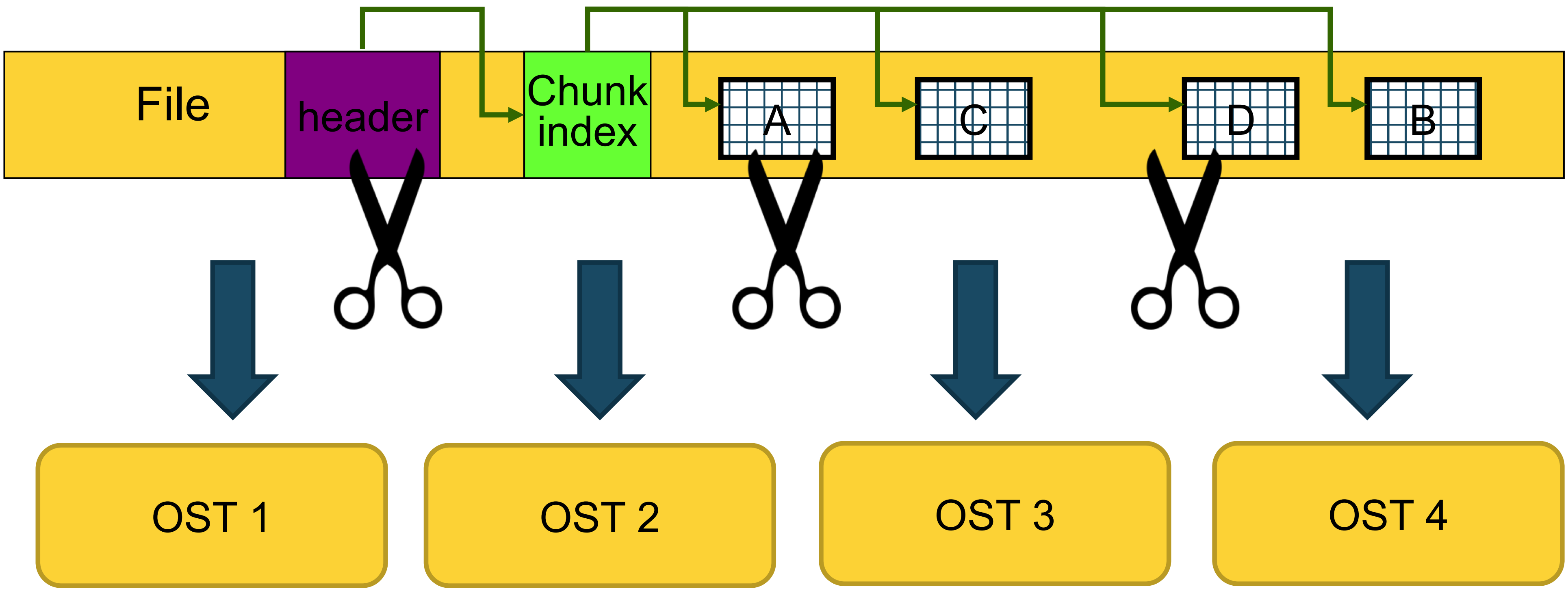


Chunked Storage

- **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.

Collective vs. Independent I/O

- **Collective I/O attempts to combine multiple smaller independent I/O ops into fewer larger ops.**
  - Neither mode is preferable *a priori*
- **MPI definition of collective calls:**
  - All processes of the communicator must participate in calls in the same order:

Process 1	Process 2	
call A(); → call B();	call A(); → call B();	**right**
call A(); → call B();	call B(); → call A();	**wrong**

  - Independent calls are not collective ☺
  - Collective calls are not necessarily synchronous, nor must they require communication
    - It could be that only internal state for the communicator changes

Data and Metadata I/O

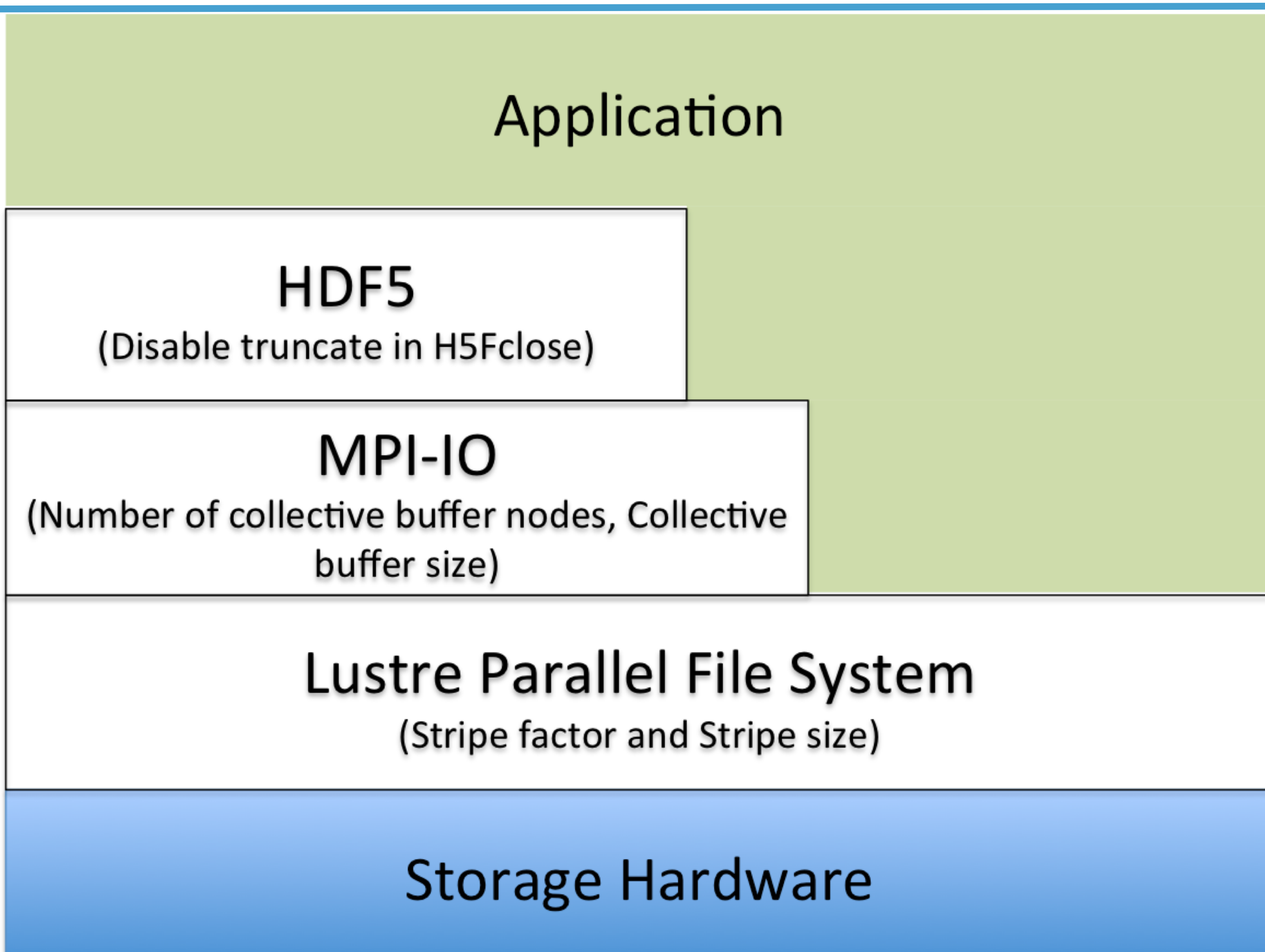
Data

- **Problem-sized**
- **I/O can be independent or collective**
- **Improvement targets:**
  - Avoid unnecessary I/O
  - I/O frequency
  - Layout on disk
    - Different I/O strategies for chunked layout
  - Aggregation and balancing
  - Alignment

Metadata

- **Small**
- **Reads can be independent or collective**
- **All modifying I/O must be collective**
- **Improvement targets:**
  - Metadata design
  - Use the latest library version, if possible
  - Metadata cache
    - In desperate cases, take control of evictions

Don’t Forget: It’s a Multi-layer Problem



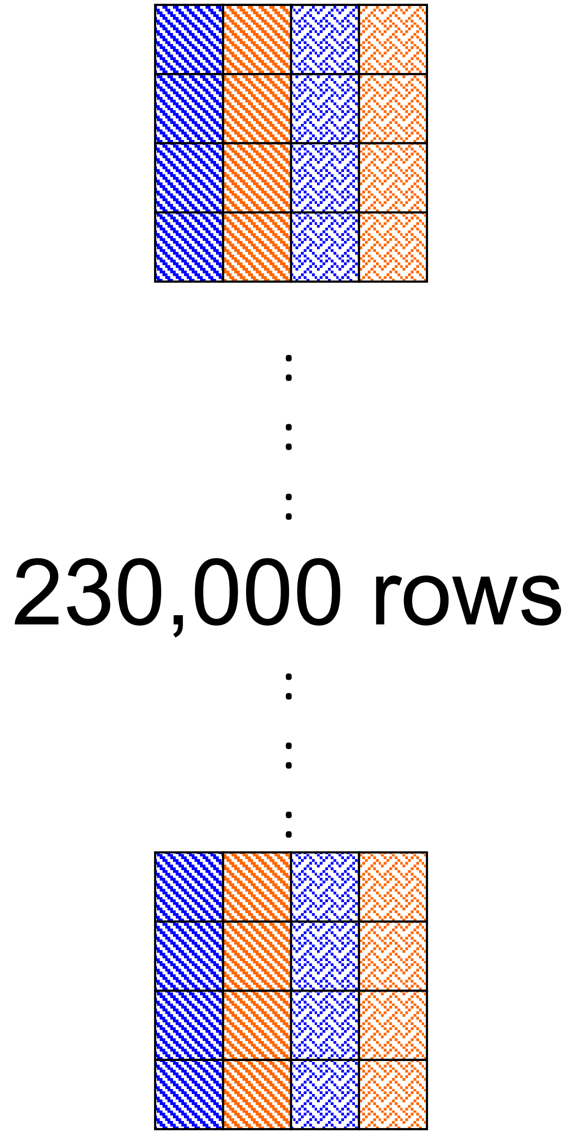
Tools

DIAGNOSTICS AND INSTRUMENTATION

A Textbook Example

User reported:

- **Independent data transfer mode is much slower than the collective data transfer mode**
- **Data array is tall and thin: 230,000 rows by 4 columns**



Symptoms

Writing to one dataset

- 4 MPI processes → 4 columns
- Datatype is 8-byte floats (doubles)
- 4 processes x 1000 rows x 8 bytes = 32,000 bytes

```
% mpirun -np 4 ./a.out 1000
```

➢ Execution time: 1.783798 s.

```
% mpirun -np 4 ./a.out 2000
```

➢ Execution time: 3.838858 s. (linear scaling)

- **2 sec. extra for 1000 more rows = 32,000 bytes.**

  16KB/sec → Way too slow!!!

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

"Poor Man's Debugging"

- **Build a version of PHDF5 with**
- `./configure --enable-debug --enable-parallel …`
- **This allows the tracing of MPIO I/O calls in the HDF5 library such as `MPI_File_read_xx` and `MPI_File_write_xx`**
- **Don't forget to `% setenv H5FD_mpio_Debug "rw"`**
- **You'll get something like this…**

Independent and Contiguous

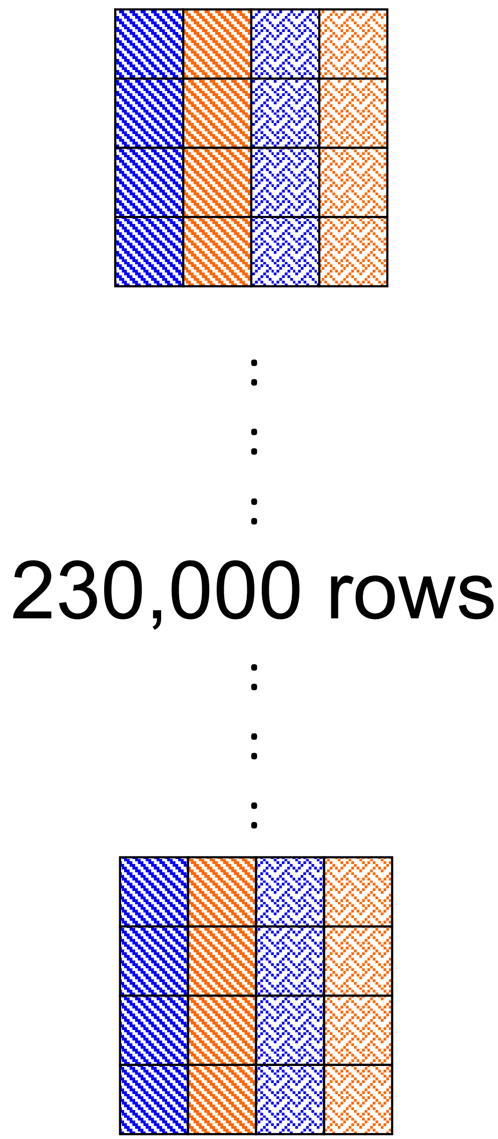
```
% setenv H5FD_mpio_Debug ’rw’
% mpirun -np 4 ./a.out 1000    # Indep.; contiguous.
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=2056  size_i=8
in H5FD_mpio_write  mpi_off=2048  size_i=8
in H5FD_mpio_write  mpi_off=2072  size_i=8
in H5FD_mpio_write  mpi_off=2064  size_i=8
in H5FD_mpio_write  mpi_off=2088  size_i=8
in H5FD_mpio_write  mpi_off=2080  size_i=8
…
```

- A total of 4000 of these 8 bytes writes == 32,000 bytes.

Plenty of Independent and Small Calls

Diagnosis:

- **Each process writes one element of one row, skips to next row, writes one element, and so on.**
- **Each process issues 230,000 writes of 8 bytes each.**



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
…
```

Metadata

Dataset elements

Metadata

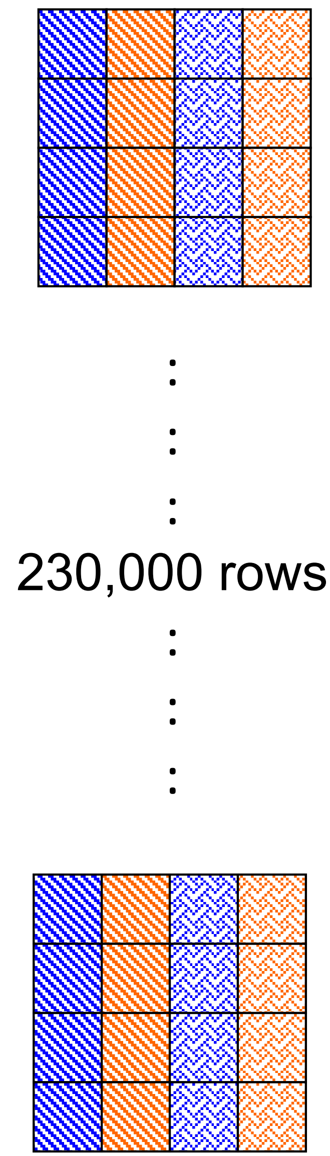
- **Execution time: 0.011599 s.**

Use Collective Mode or Chunked Storage

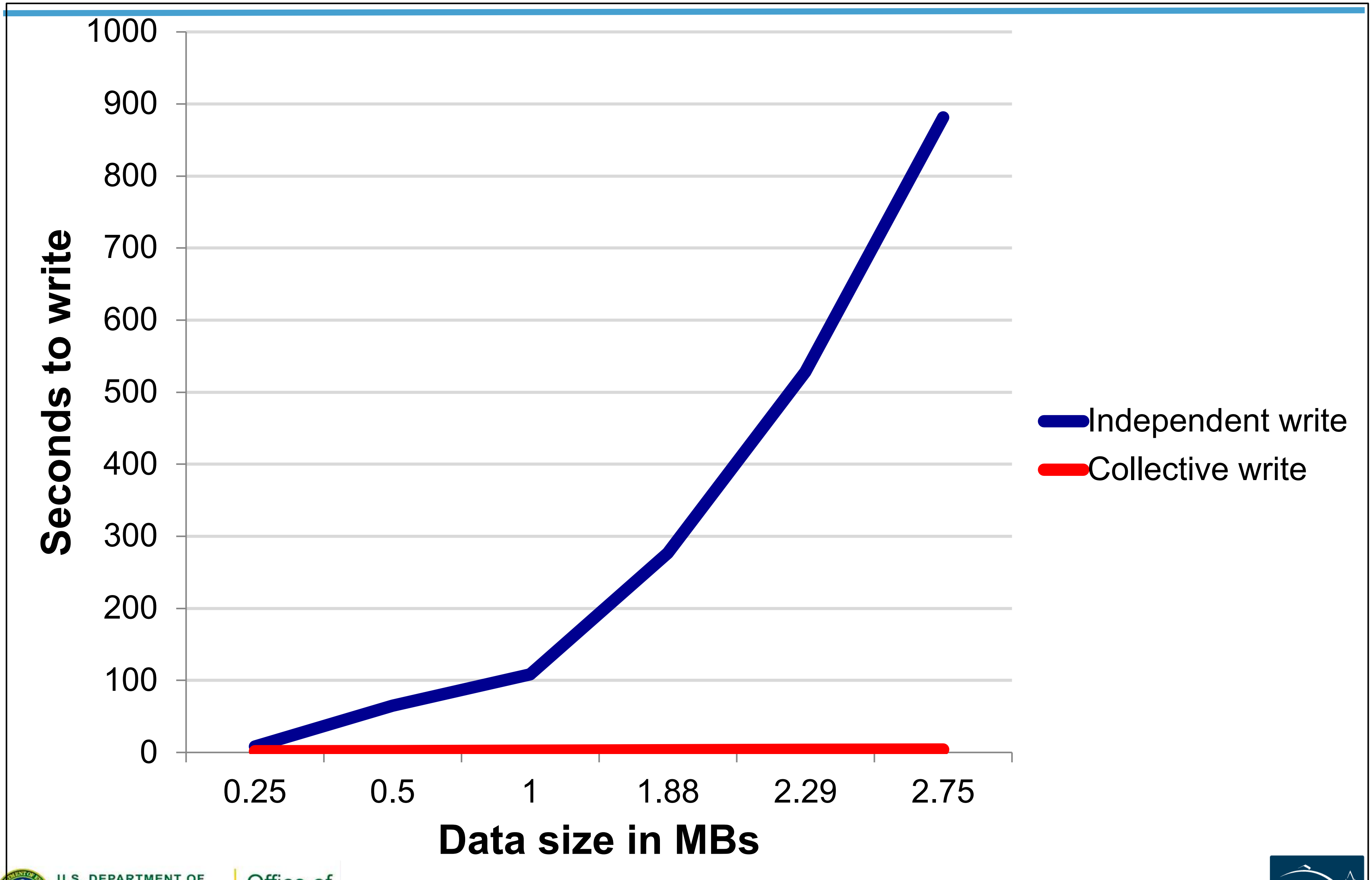
Remedy:

- **Collective I/O will combine many small independent calls into few but bigger calls**

- **Chunks of columns speeds up too**



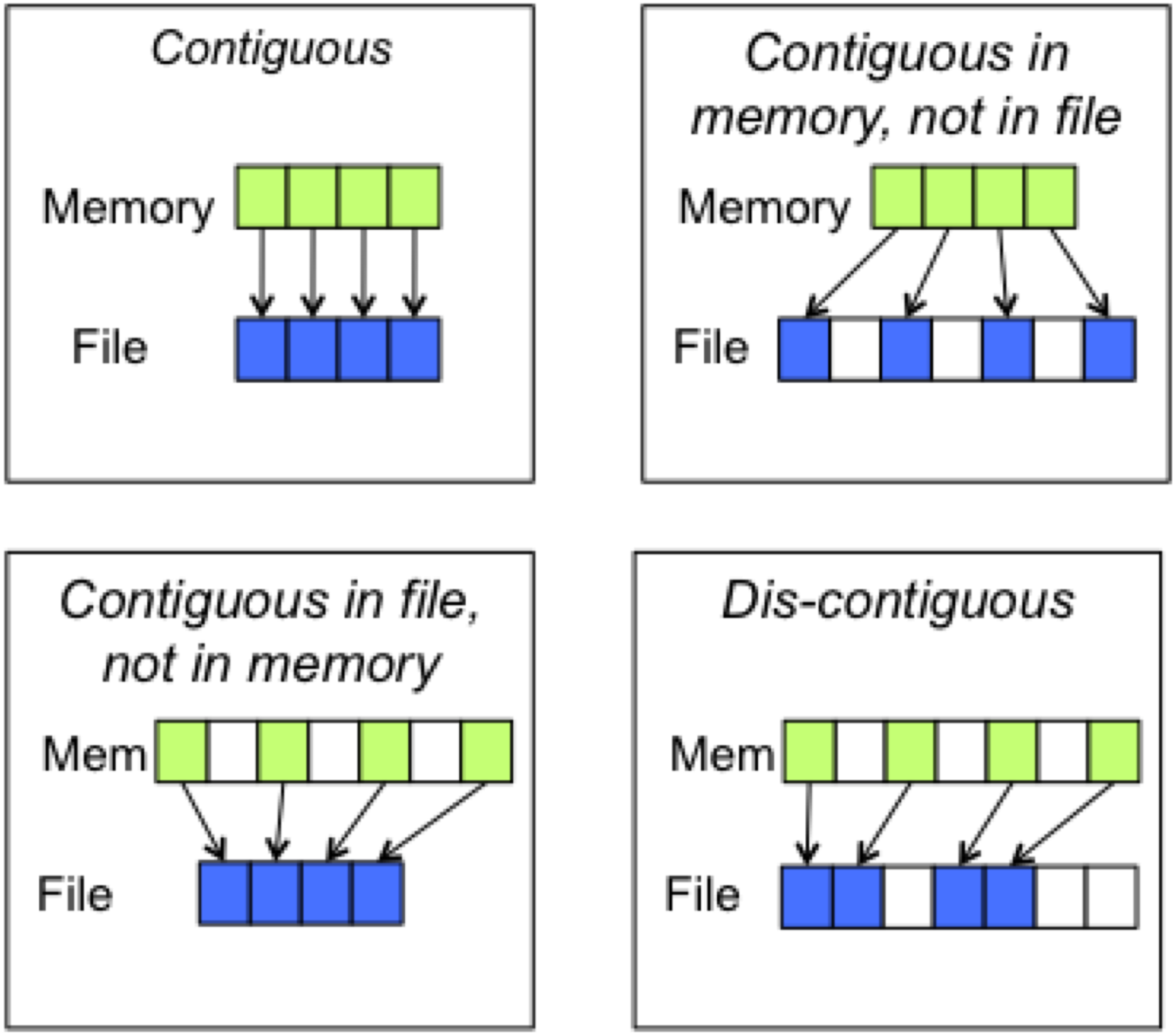
Collective vs. independent write



Back Into the Real World…

- **Two kinds of tools:**
  - I/O benchmarks for measuring a system's I/O capabilities
  - I/O profilers for characterizing applications' I/O behavior
- **Two examples:**
  - h5perf (in the HDF5 source code distro)
  - Darshan (from Argonne National Laboratory)
- **Profilers have to compromise between**
  - A lot of detail → large trace files and overhead
  - Aggregation → loss of detail, but low overhead

I/O Patterns



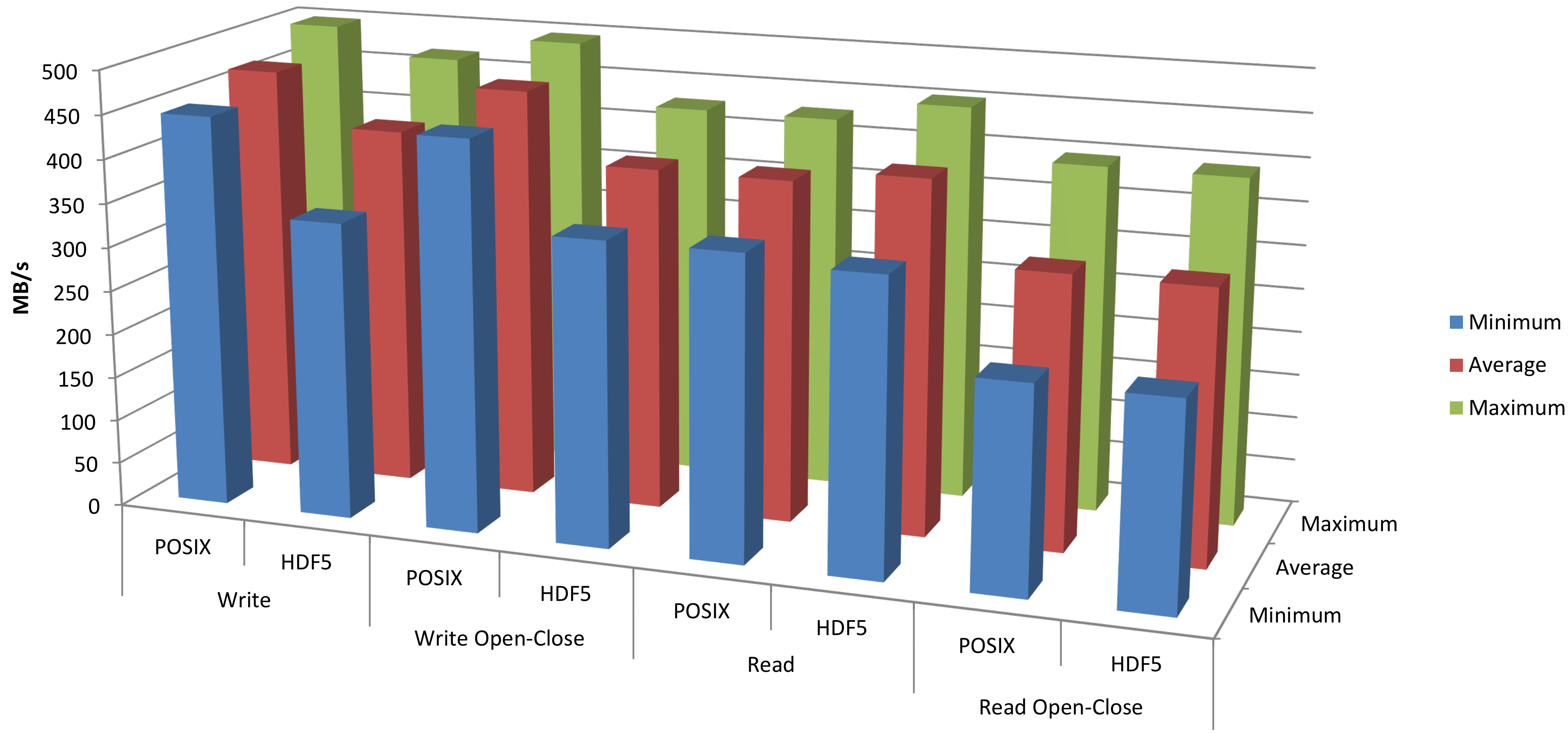
Contiguous
Memory
File
Contiguous in memory, not in file
Memory
File
Contiguous in file, not in memory
Mem
File
Dis-contiguous
Mem
File

h5perf(_serial)

- **Measures performance of a filesystem for different I/O patterns and APIs**
- **Three File I/O APIs for the price of one!**
  - POSIX I/O (open/write/read/close…)
  - MPI-I/O (MPI_File_{open,write,read,close})
  - HDF5 (H5Fopen/H5Dwrite/H5Dread/H5Fclose)
- **An indication of I/O speed ranges and HDF5 overheads**
- **Expectation management…**

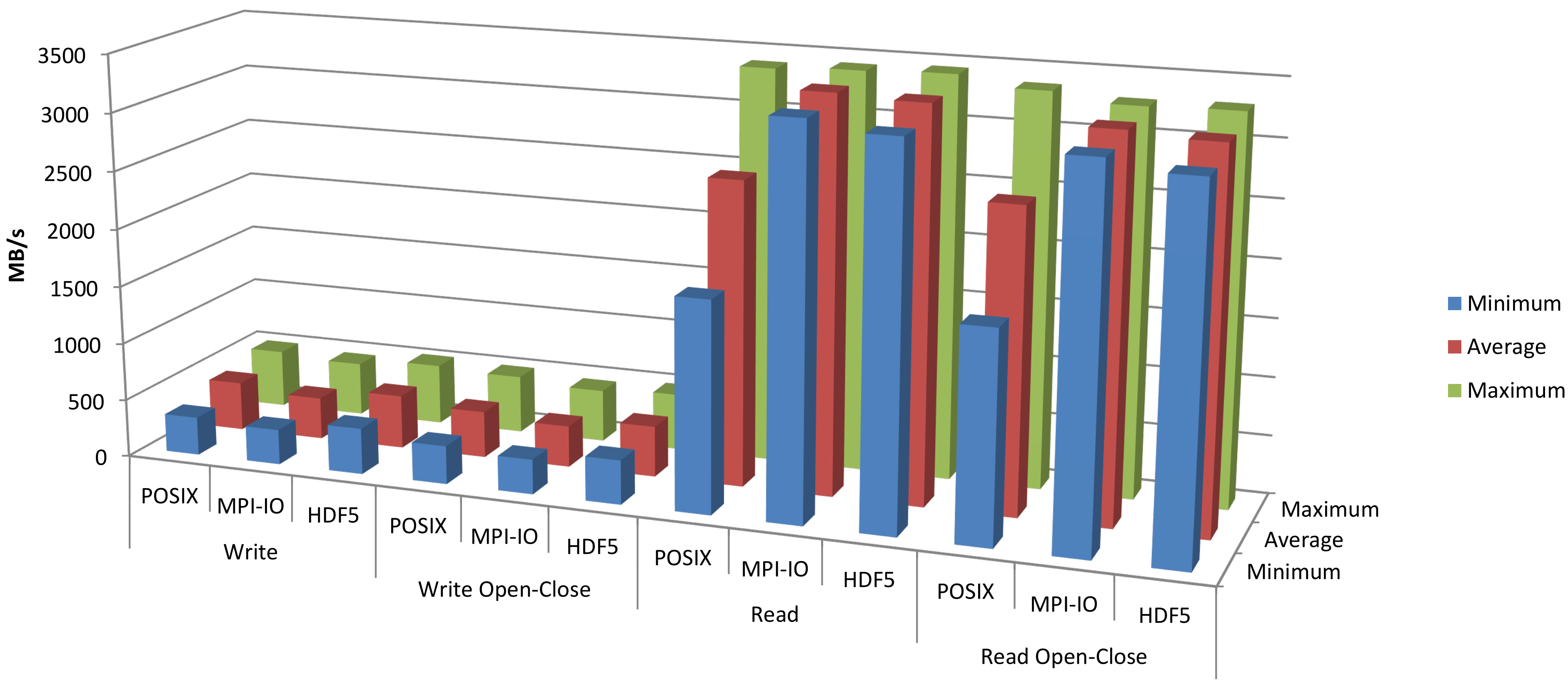
A Serial Run

h5perf_serial, 3 iterations, 1 GB dataset, 1 MB transfer buffer, HDF5 dataset contiguous storage, HDF5 SVN trunk, NCSA BW



A Parallel Run

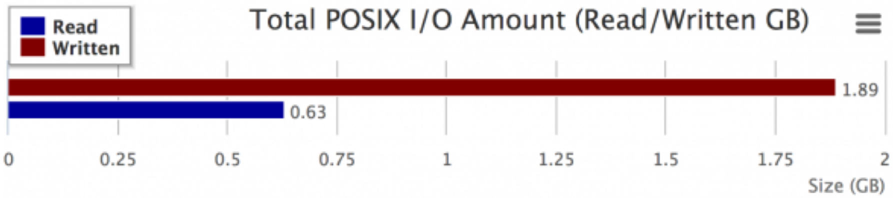
h5perf, 3 MPI processes, 3 iterations, 3 GB dataset (total), 1 GB per process, 1 GB transfer buffer, HDF5 dataset contiguous storage, HDF5 SVN trunk, NCSA BW

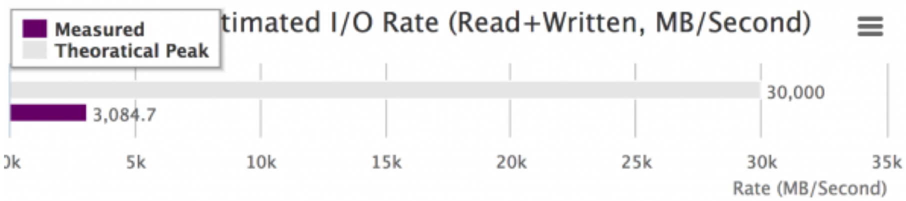


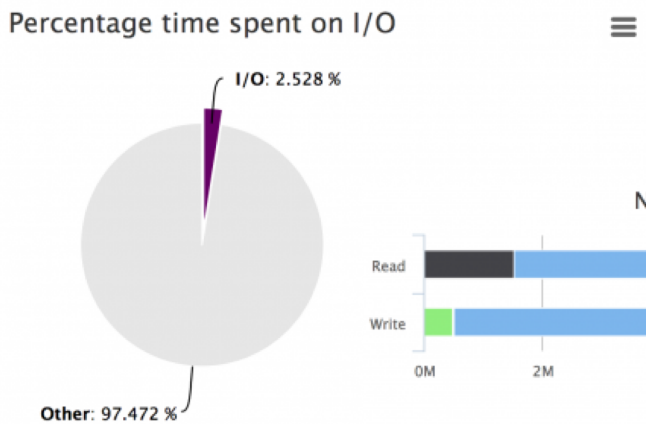
Darshan (ANL)

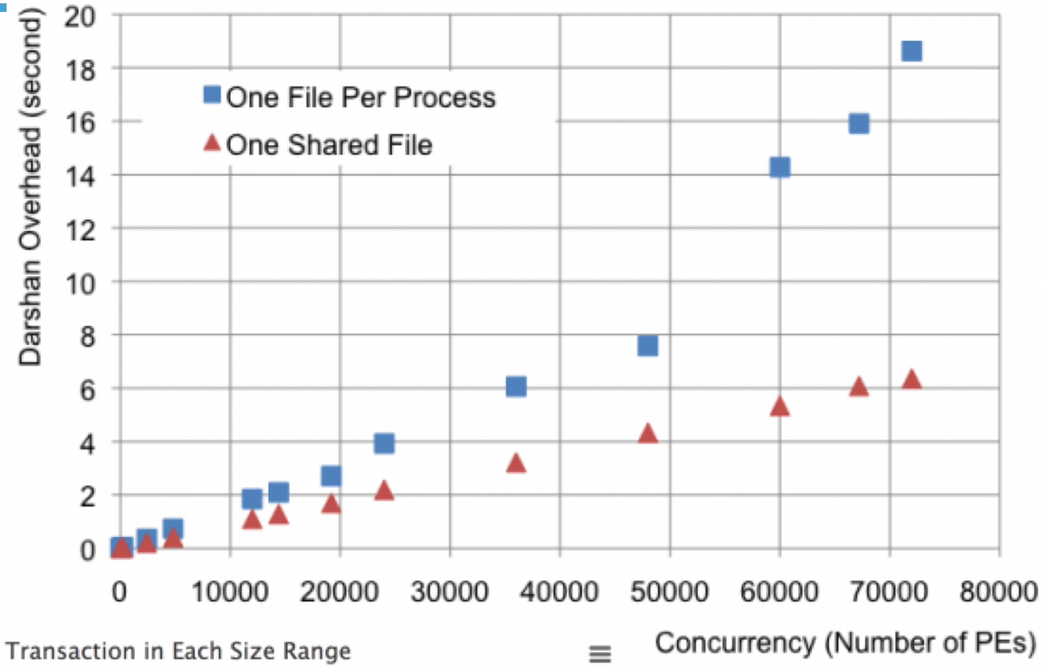
- **Design goals:**
  - Transparent integration with user environment
  - Negligible impact on application performance
- **Provides aggregate figures for:**
  - Operation counts (POSIX, MPI-IO, HDF5, PnetCDF)
  - Datatypes and hint usage
  - Access patterns: alignments, sequentiality, access size
  - Cumulative I/O time, intervals of I/O activity
- **Does not provide I/O behavior over time**
- **Excellent starting point, maybe not your final stop**

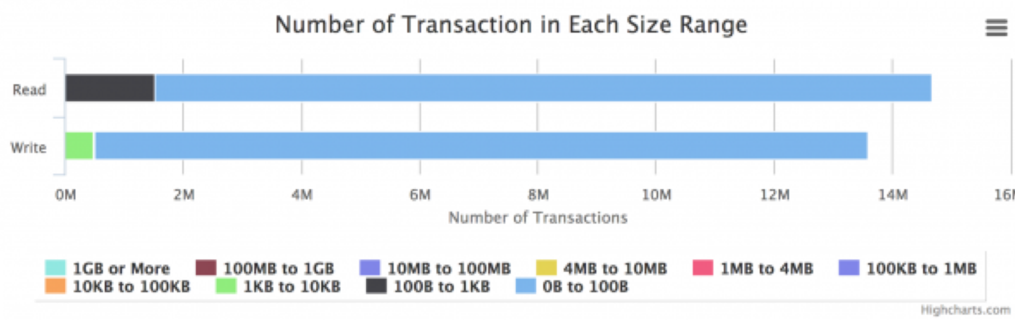
Darshan Sample Output

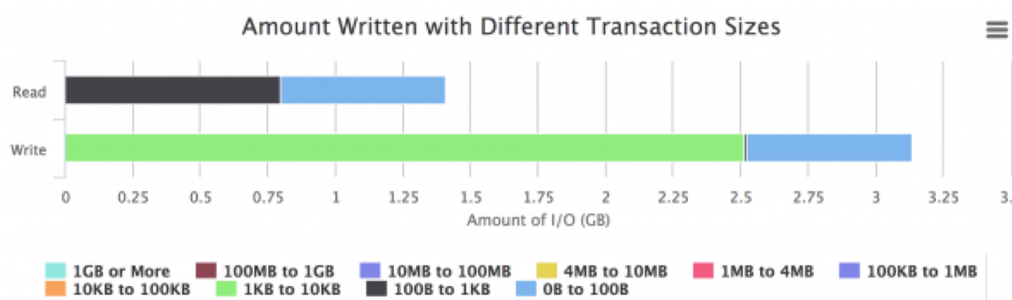












Source: [NERSC]

EXAMPLES

Standard Questions

- **What I/O layers are involved and how much control do I have over them?**
- **Which ones do I tackle in which order?**
  - Are there any low-hanging fruit?
- **What's my baseline (for each layer) and what are my metrics?**
- **Which tool(s) will give me the information I need?**
- **When do I stop?**

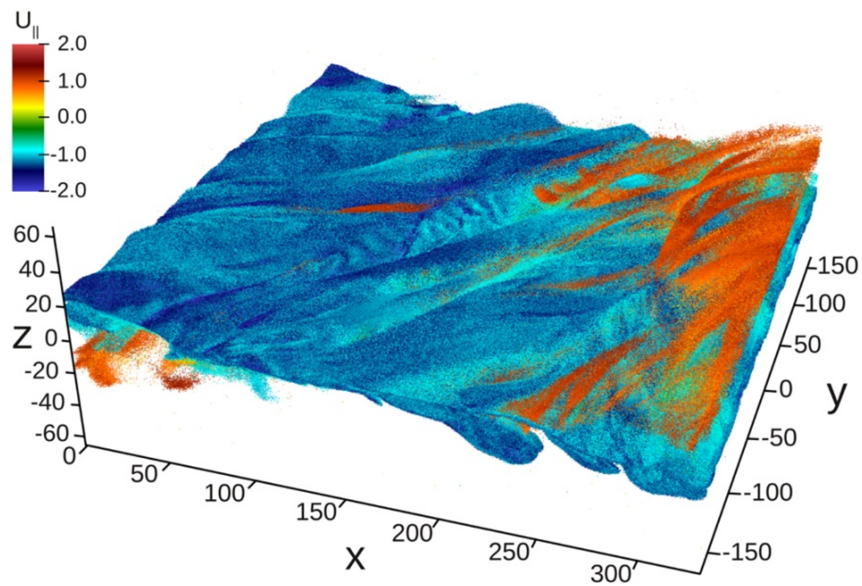
- **New information ➔ New answers (maybe) : Need to keep an open mind!**

Reference:

[Trillion Particles, 120,000 cores, and 350 TBs: Lessons Learned from a Hero I/O Run on Hopper](),
By Suren Byna (LBNL) et al., 2015.

Examples

VPIC



Layers

Application

HDF5
(Disable truncate in H5Fclose)

MPI-IO
(Number of collective buffer nodes, Collective buffer size)

Lustre Parallel File System
(Stripe factor and Stripe size)

Storage Hardware

"Application I/O Structure"

- **Total control over all layers**
- **Challenge: large output files**
- **Metric: write speed (throughput)**
- **Computationally intensive → Need an I/O kernel**
- **H5Part multiple dataset writes**

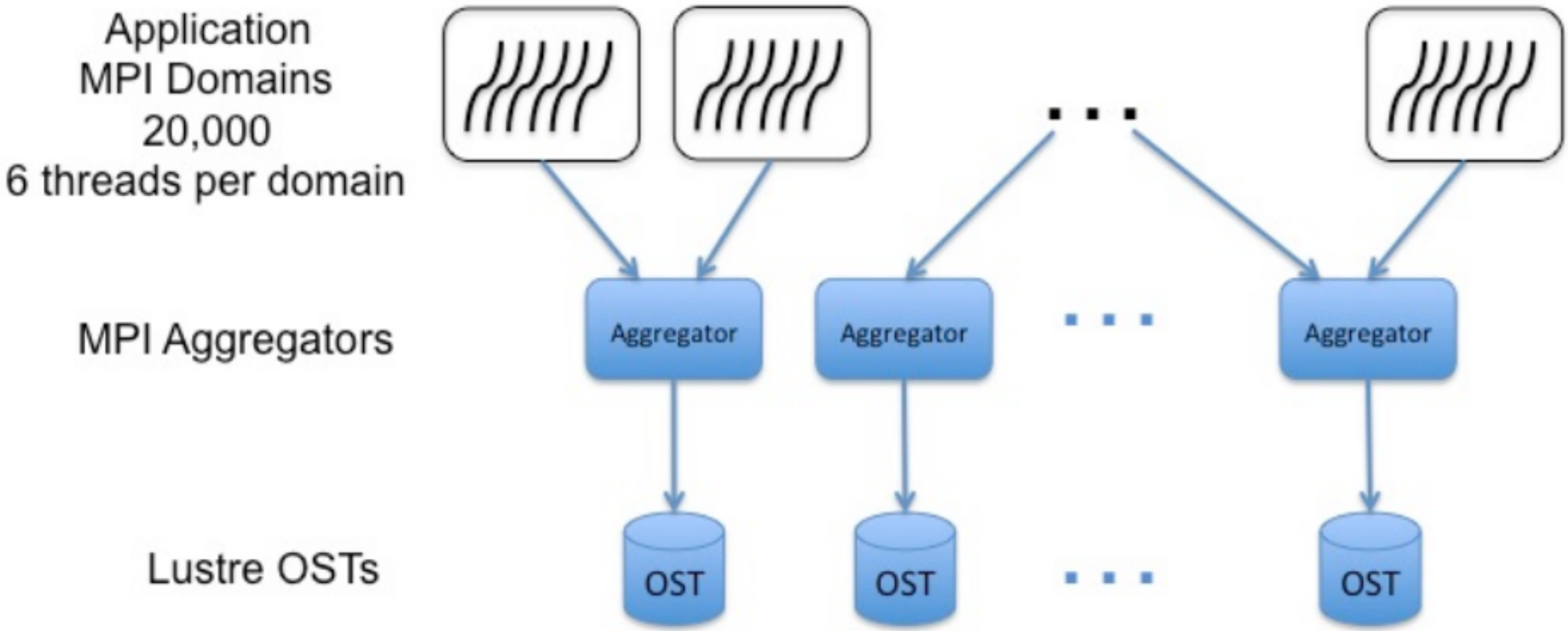
```
h5pf = H5PartOpenFileParallel (fname, H5PART_WRITE |
                   H5PART_FS_LUSTRE, MPI_COMM_WORLD);
H5PartSetStep (h5pf, step);
H5PartSetNumParticlesStrided (h5pf, np_local, 8);

H5PartWriteDataFloat32 (h5pf, "dX", Pf);
H5PartWriteDataFloat32 (h5pf, "dY", Pf+1);
H5PartWriteDataFloat32 (h5pf, "dZ", Pf+2);
H5PartWriteDataInt32   (h5pf, "i",  Pi+3);
H5PartWriteDataFloat32 (h5pf, "Ux", Pf+4);
H5PartWriteDataFloat32 (h5pf, "Uy", Pf+5);
H5PartWriteDataFloat32 (h5pf, "Uz", Pf+6);
H5PartWriteDataFloat32 (h5pf, "q",  Pf+7);

H5PartCloseFile (h5pf);
```

- **"Game plan":**
  - MPI-IO / Lustre tuning
    - Low hanging fruit (relatively)
    - Pair MPI aggregators with Lustre OSTs
    - Match MPI-IO buffer sizes and Lustre stripe size
  - Worry about HDF5 (H5Part)

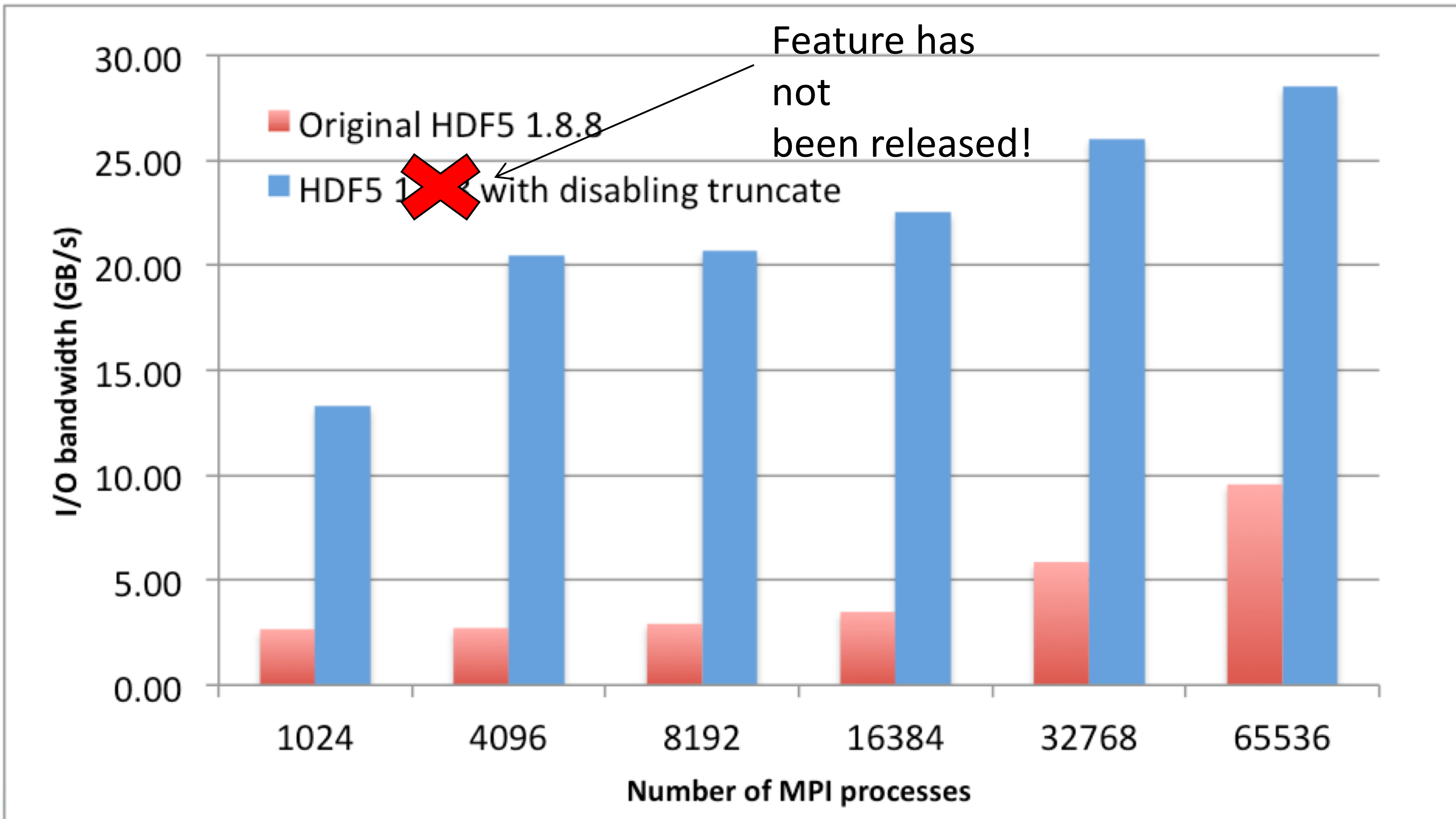
I/O Aggregation



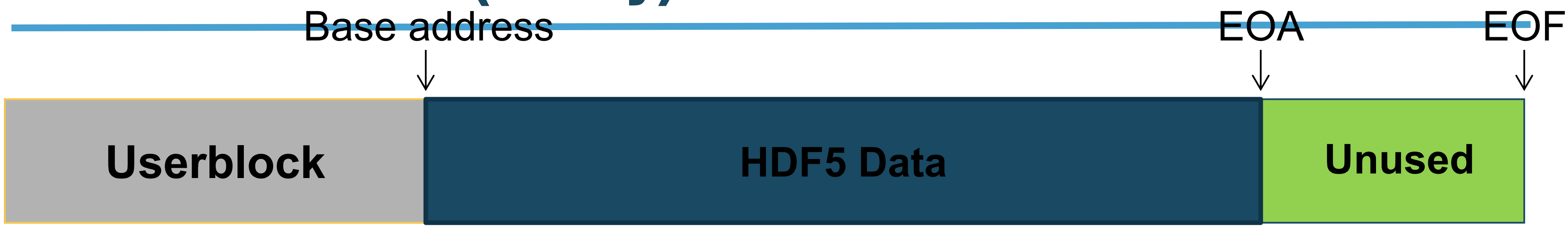
Closing HDF5 File …

Q: How long does it take to close/flush an HDF5 file?

A: A lot longer than you might expect!



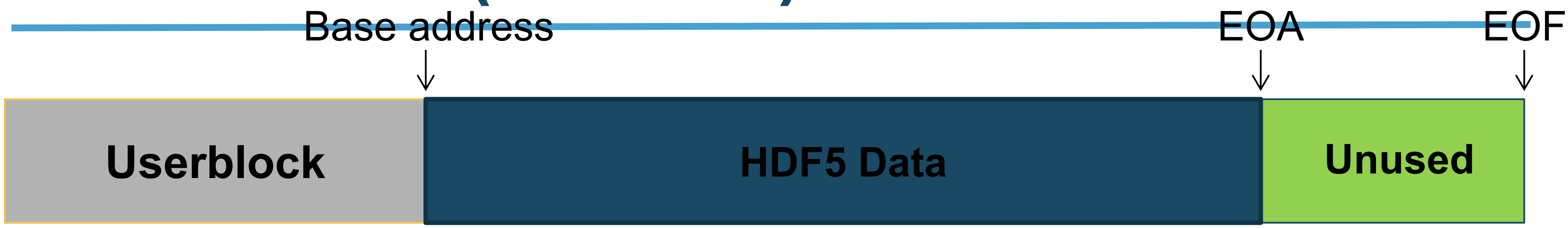
File Truncation (Today)



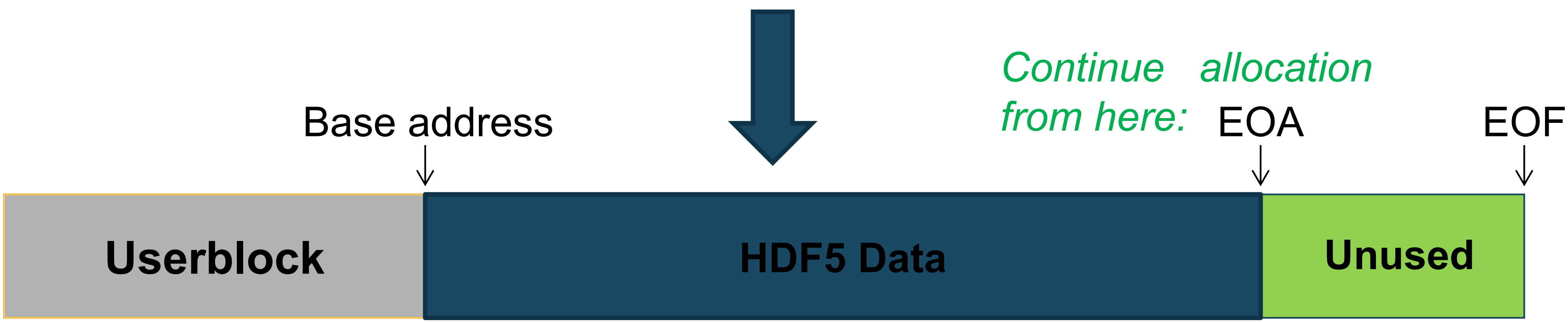
A call to `H5Fflush` or `H5Fclose` triggers a call to `ftruncate` (serial) or `MPI_File_set_size` (parallel), which can be fairly expensive.

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

File Truncation (Tomorrow)



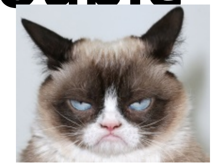
A call to `H5Fflush` or `H5Fclose` triggers both values (EOA, EOF) to be saved in the file and **no** truncation takes place, IF the file was created with the "avoid truncation" property set.



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

Multi-Dataset I/O - Motivation

- HDF5 accesses elements in one dataset at a time
- Many HPC applications access data in multiple datasets in every time step
- Frequent small-size dataset access ➔ Big Trouble (≠Big Data)



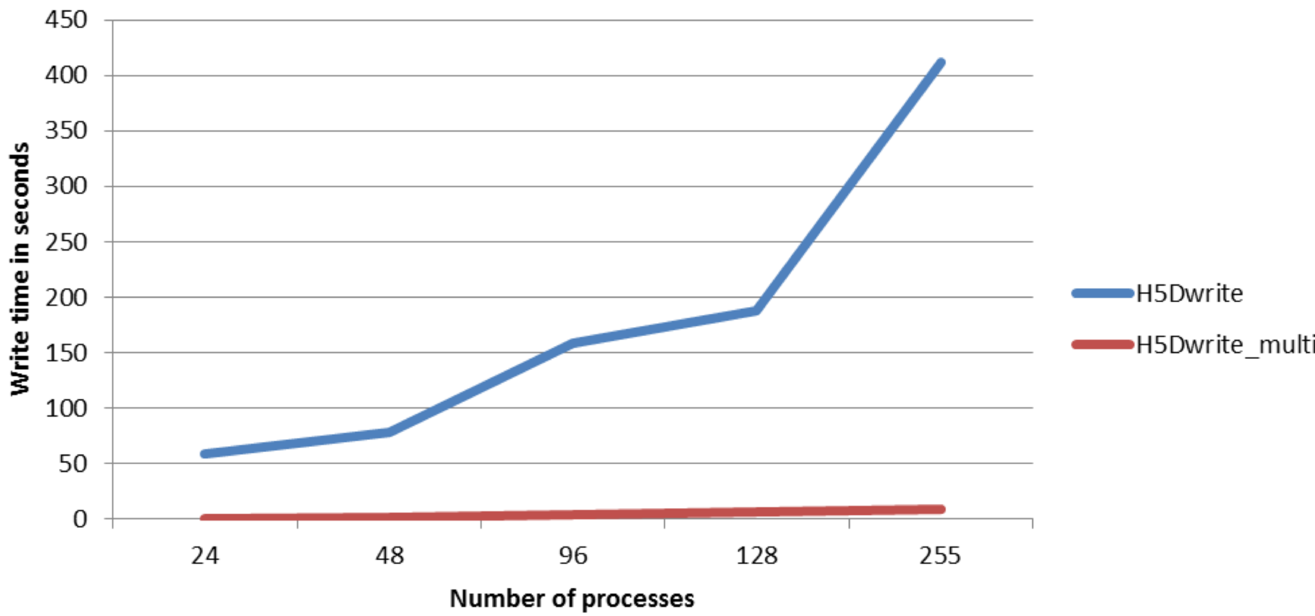
- Parallel file systems tend not to like that.
- Idea: Let users to do more I/O per HDF5 call!
- Two New API routines:
  - `H5Dread_multi()`
  - `H5Dwrite_multi()`

Not a new idea: PnetCDF has supported this for some time…

Sample Results

The plot shows the performance difference between using a single `H5Dwrite()` multiple times and using `H5Dwrite_multi()` once on 30 chunked datasets.

(On Hopper @ NERSC, a Cray XE-6 with Lustre file system)



Run	Code	Nodes	Cores	File Size	Stripe Size	Write Time	Total HDF Time	Throughput
Max Throughput	Multiple Dataset	320	5,120	5 TB	1 GB	91.08 s	167.78 s	56.21 GB/s
Code Comparison	Single Dataset	320	5,120	5 TB	128 MB	116.94 s	117.37 s	43.78 GB/s
Hero Run	Single Dataset	9,314	298,048	291 TB	1 GB	5,763.14 s	5,779.89 s	51.81 GB/s

TABLE I

COMPARISON OF VPIC-IO KERNEL PARAMETERS AND OBSERVED IO THROUGHPUT.

Reference:

Parallel and Large-scale Simulation Enhancements to CGNS, By Scot Breitenfeld, The HDF Group, 2015.

Examples

CGNS

CFD Standard

- **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**

Shaping the Future of Aerospace

CGNS Storage Evolution

- CGNS data was originally stored in ADF ('Advanced Data Format')
- ADF lacks parallel I/O or data compression capabilities
- Doesn't have HDF5's support base and tools
- HDF5 superseded ADF as the official storage mechanism
- CGNS introduced parallel I/O APIs w/ parallel HDF5 in 2013
- Poor performance of the new parallel APIs in most circumstances

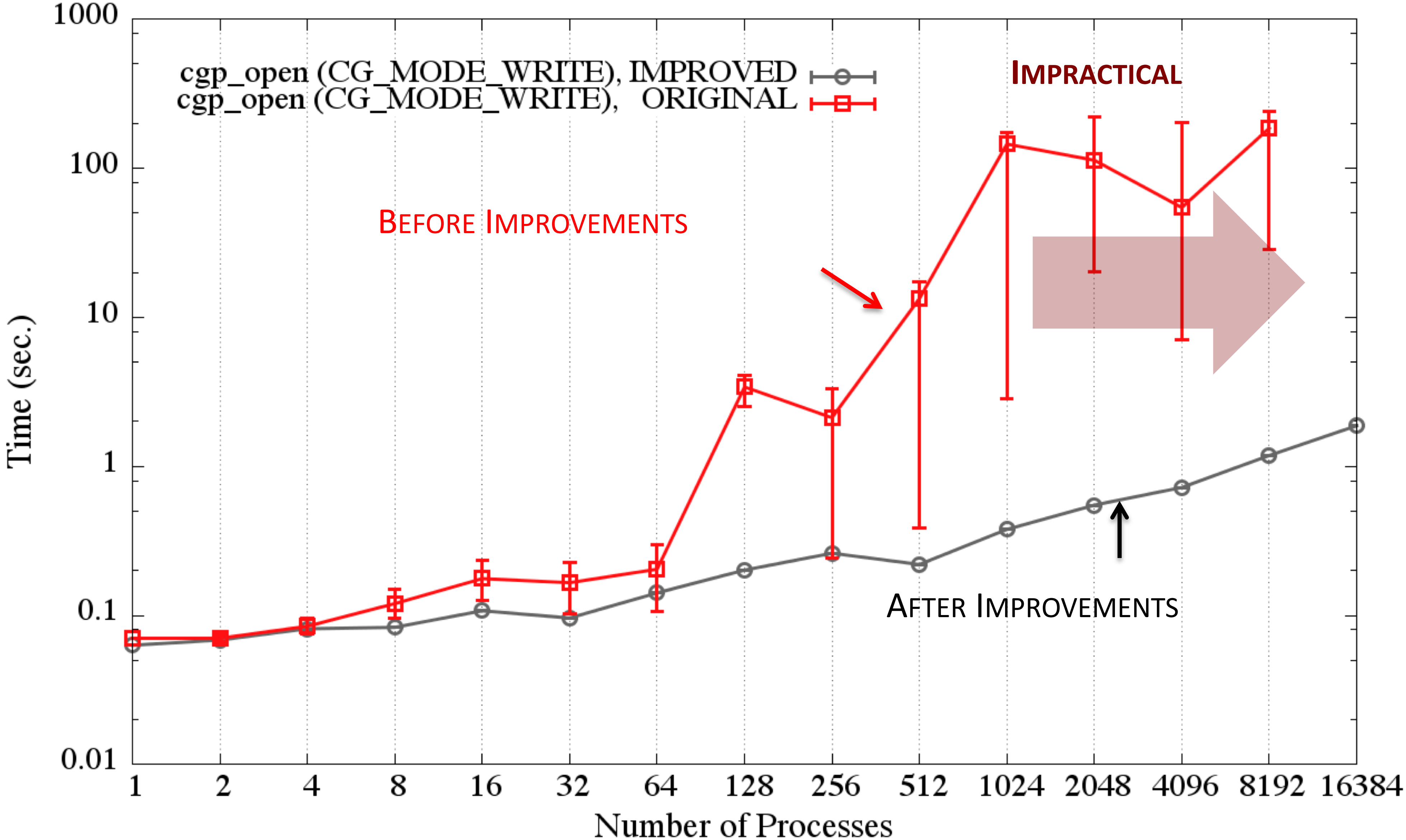
- In 2014, NASA provided funding for The HDF Group with the goal to improve the under-performing parallel capabilities of the CGNS library.

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

Opening CGNS File …



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");
```

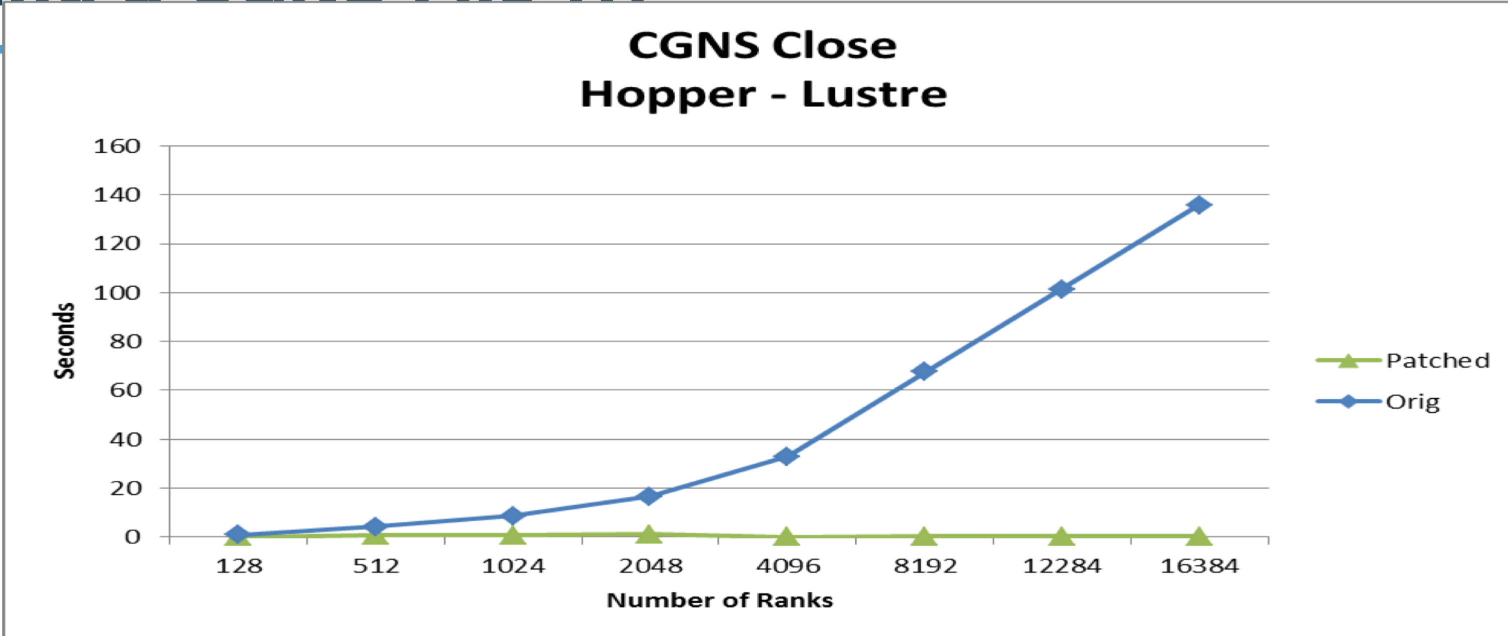
Metadata Read Storm Problem (II)

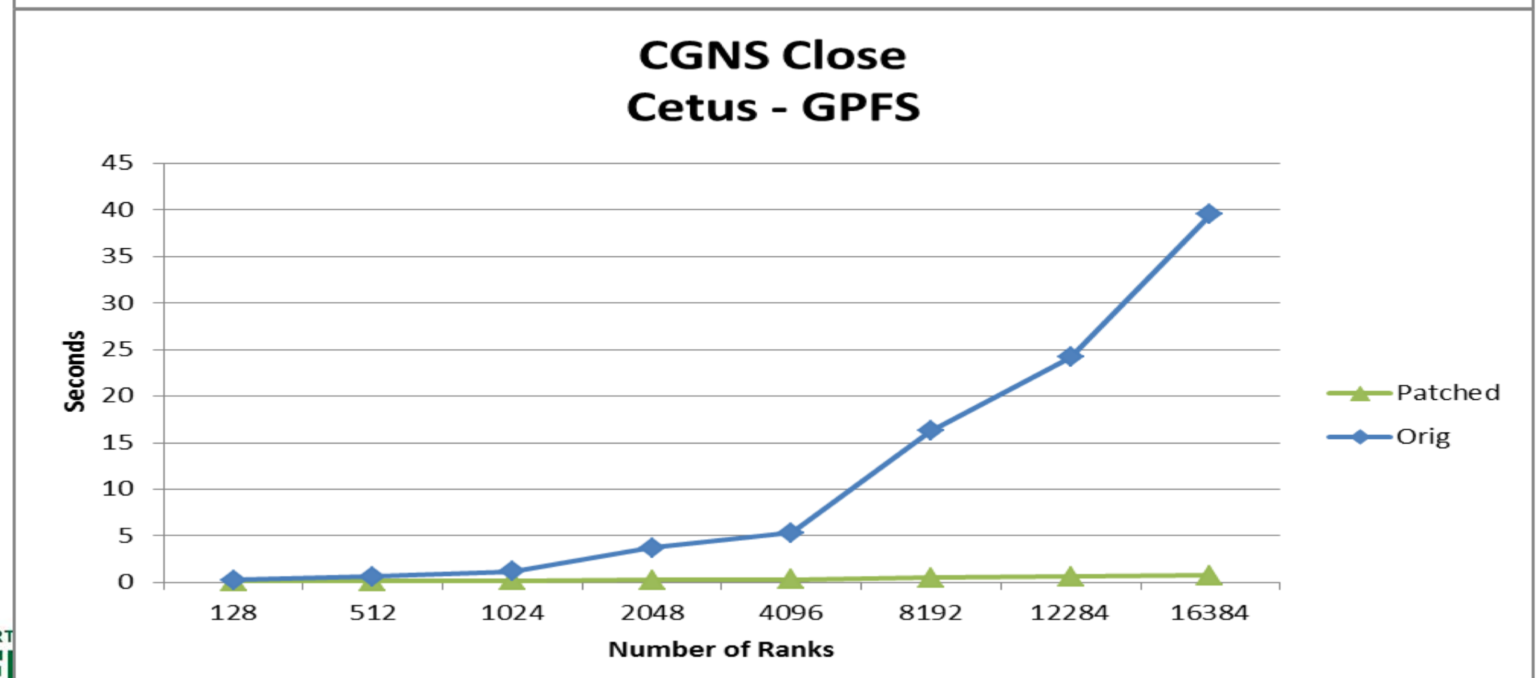
- **Metadata read operations are treated by the library as independent read operations.**
- **Consider a very large MPI job size where all processes want to open a dataset that already exists in the file.**
- **All processes**
  - Call `H5Dopen("/G1/G2/D1");`
  - Read the same metadata to get to the 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
- 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

Closing a 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.**

HDF5 Roadmap

- **Concurrency**
  - Single-Writer / Multiple-Reader (SWMR)
  - Asynchronous I/O
  - Internal threading
- **Virtual Object Layer**
- ~~**Virtual Datasets**~~
- **Query & Indexing**
- **Native HDF5 client/server**

- Performance
  - ~~Scalable chunk indices~~
  - ~~Metadata aggregation and Page buffering~~
  - Variable-length records
- Fault tolerance
- Parallel I/O
- I/O Autotuning

Questions, Comments, Feedback?

Thank You!

ExaHDF5 – Features

- Virtual Object Layer (VOL) integration into HDF5
- Caching and prefetching – Data Elevator VOL
- Topology-aware I/O
- Asynchronous I/O
- Independent metadata updates
- Workflow supporting features – SWMR
- Querying HDF5 data and metadata
- Interoperability with other file formats
    - PnetCDF/netCDF, ADIOS
- Maintenance and release support
- ECP engagement w/ AD, ST, and HT
    - Consulting and performance tuning for applications

ExaHDF5 – Development timeline