A Brief Introduction to Parallel HDF5

Quincey Koziol & Scot Breitenfeld
The HDF Group
koziol@hdfgroup.org, brtnfld@hdfgroup.org

http://bit.ly/ParallelHDF5-MPBC-2015

Recent Parallel HDF5 Success Story

- Performance of VPIC-IO on NCSA Blue Waters
  - I/O Kernel of a Plasma Physics application
- 56 GB/s I/O rate in writing 5TB data using 5K cores with multi-dataset write optimization
- VPIC-IO kernel running on **298,048 cores**
  - ~10 Trillion particles
  - **291 TB, single file**
  - 1 GB stripe size and 160 Lustre OSTs
  - 52 GB/s
  - 53% of the “practical” peak performance

http://bit.ly/ParallelHDF5-MPBC-2015

Outline

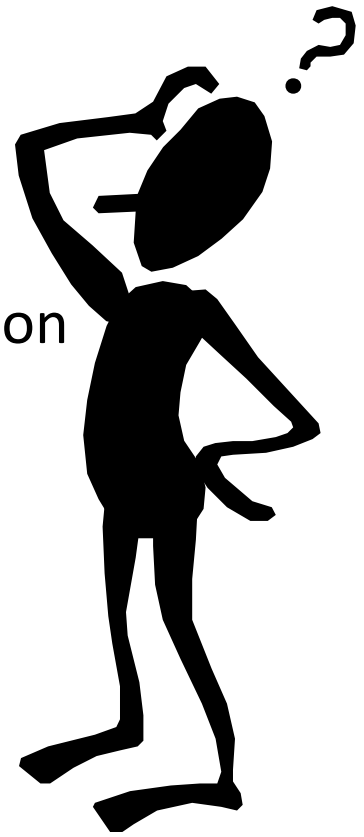
- Quick Intro to HDF5
- Overview of Parallel HDF5 design
- Parallel Consistency Semantics
- PHDF5 Programming Model
- Examples
- Performance Analysis
- Parallel Tools
- Details of upcoming features of HDF5

http://bit.ly/ParallelHDF5-MPBC-2015

QUICK INTRO TO HDF5

What is HDF5?

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



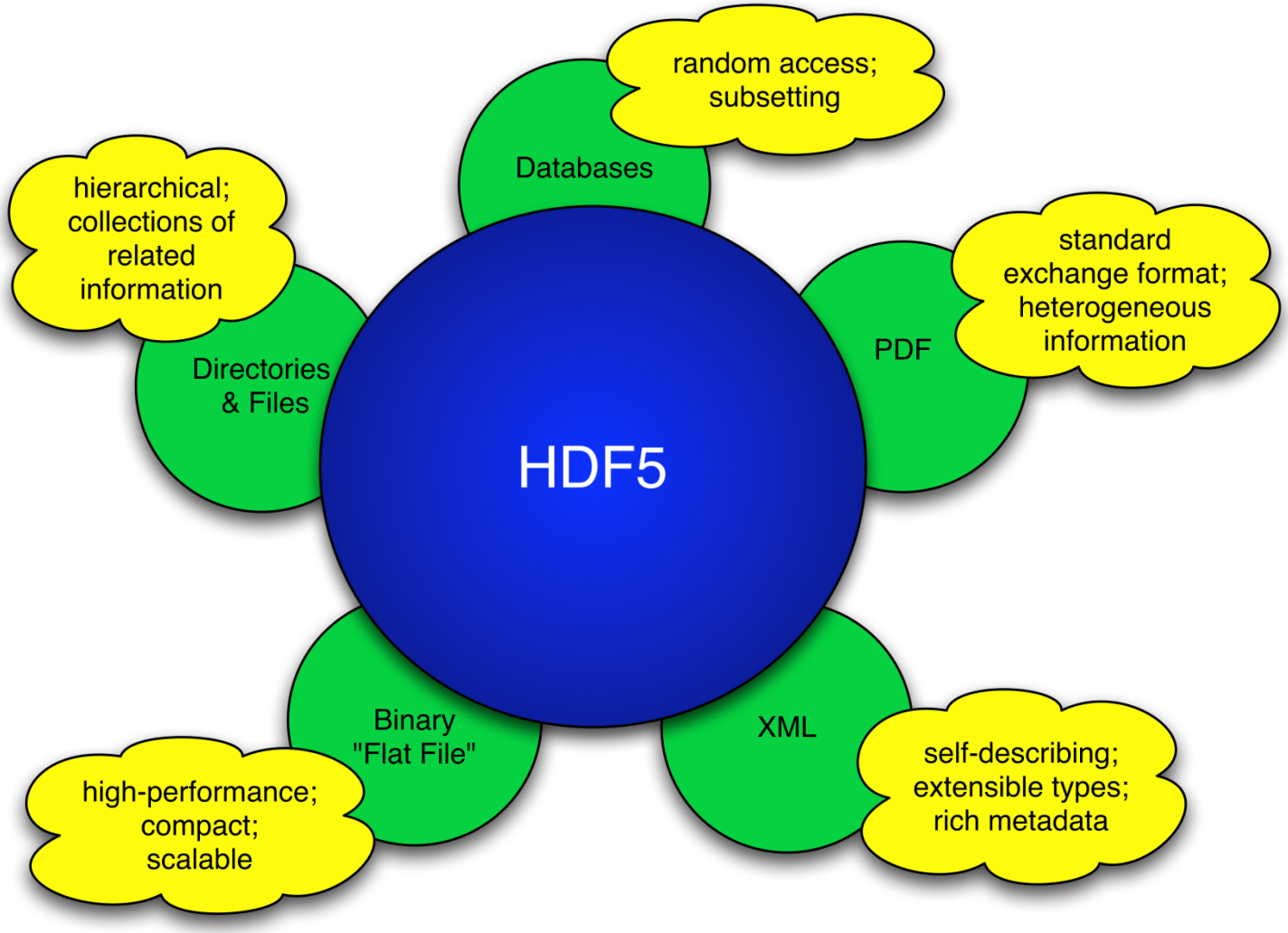
http://bit.ly/ParallelHDF5-MPBC-2015

What is HDF5, in detail?

- **A versatile data model** that can represent very complex data objects and a wide variety of metadata.
- **An open source software library** that runs on a wide range of computational platforms, from cell phones to massively parallel systems, and implements a high-level API with C, C++, Fortran, and Java interfaces.
- **A rich set of integrated performance features** that allow for access time and storage space optimizations.
- **Tools and applications** for managing, manipulating, viewing, and analyzing the data in the collection.
- **A completely portable file format** with no limit on the number or size of data objects stored.

http://bit.ly/ParallelHDF5-MPBC-2015

HDF5 is like …


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

Why use HDF5?

- Challenging data:
  - Application data that pushes the limits of what can be addressed by traditional database systems, XML documents, or in-house data formats.
- Software solutions:
  - For very large datasets, very fast access requirements, or very complex datasets.
  - To easily share data across a wide variety of computational platforms using applications written in different programming languages.
  - That take advantage of the many open-source and commercial tools that understand HDF5.
  - Enabling long-term preservation of data.

Who uses HDF5?

- Examples of HDF5 user communities
  - Astrophysics
  - Astronomers
  - NASA Earth Science Enterprise
  - US Dept. of Energy Labs
  - Supercomputing Centers in US, Europe and Asia
  - Synchrotrons and Light Sources in US and Europe
  - Financial Institutions
  - NOAA
  - Engineering & Manufacturing Industries
  - Many others
- For a more detailed list, visit
  - http://www.hdfgroup.org/HDF5/users5.html

The HDF Group

- Established in 1988
  - 18 years at University of Illinois' National Center for Supercomputing Applications
  - 8 years as independent non-profit company:

    "The HDF Group"
- The HDF Group owns HDF4 and HDF5
  - HDF4 & HDF5 formats, libraries, and tools are open source and freely available with BSD-style license
- Currently employ 37 people
  - *Always looking for more developers!*

HDF5 Technology Platform

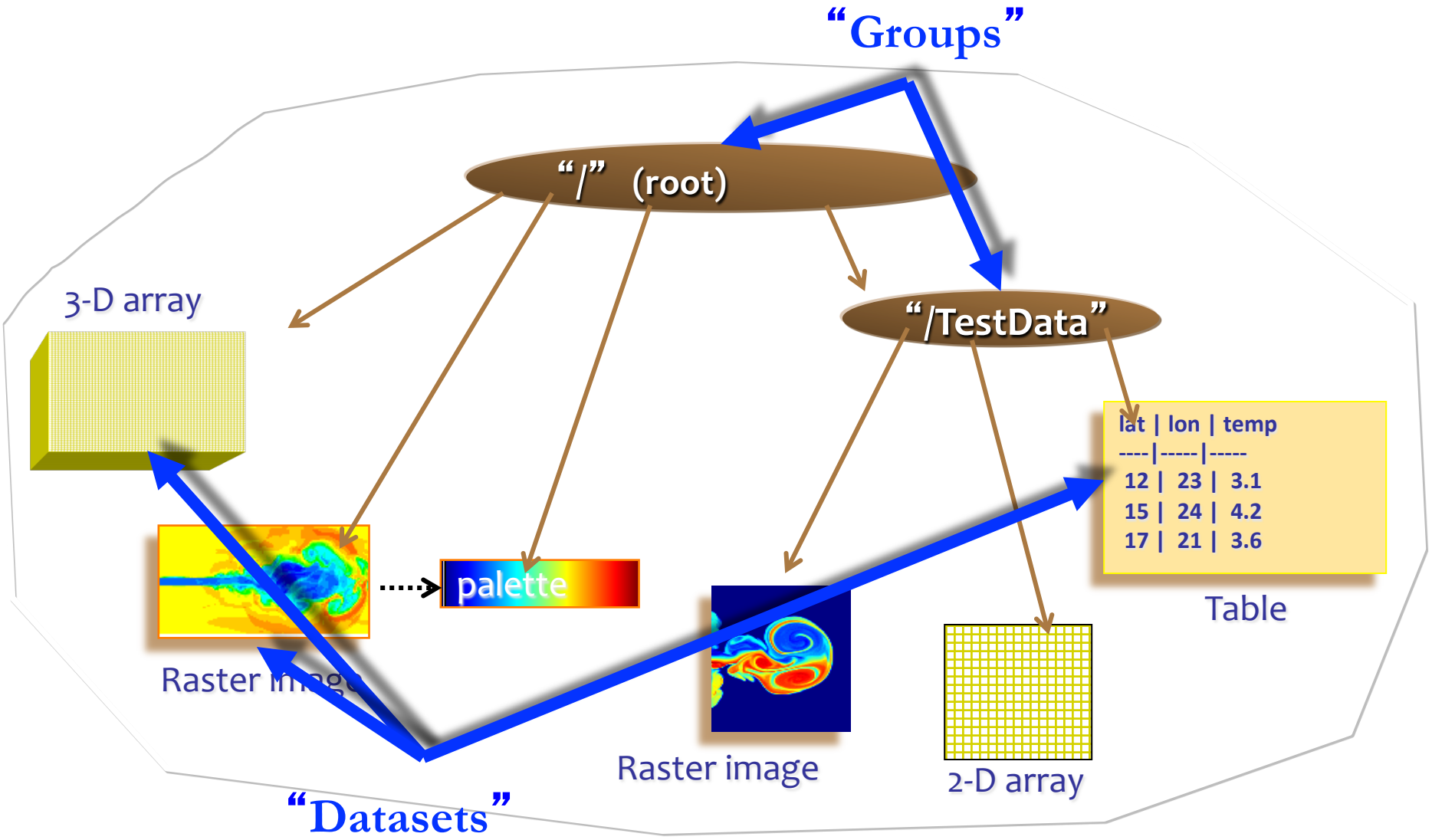
- HDF5 Abstract Data Model
  - Defines the “building blocks” for data organization and specification
  - Files, Groups, Links, Datasets, Attributes, Datatypes, Dataspaces
- HDF5 Software
  - Tools
  - Language Interfaces
  - HDF5 Library
- HDF5 Binary File Format
  - Bit-level organization of HDF5 file
  - Defined by HDF5 File Format Specification

HDF5 Data Model

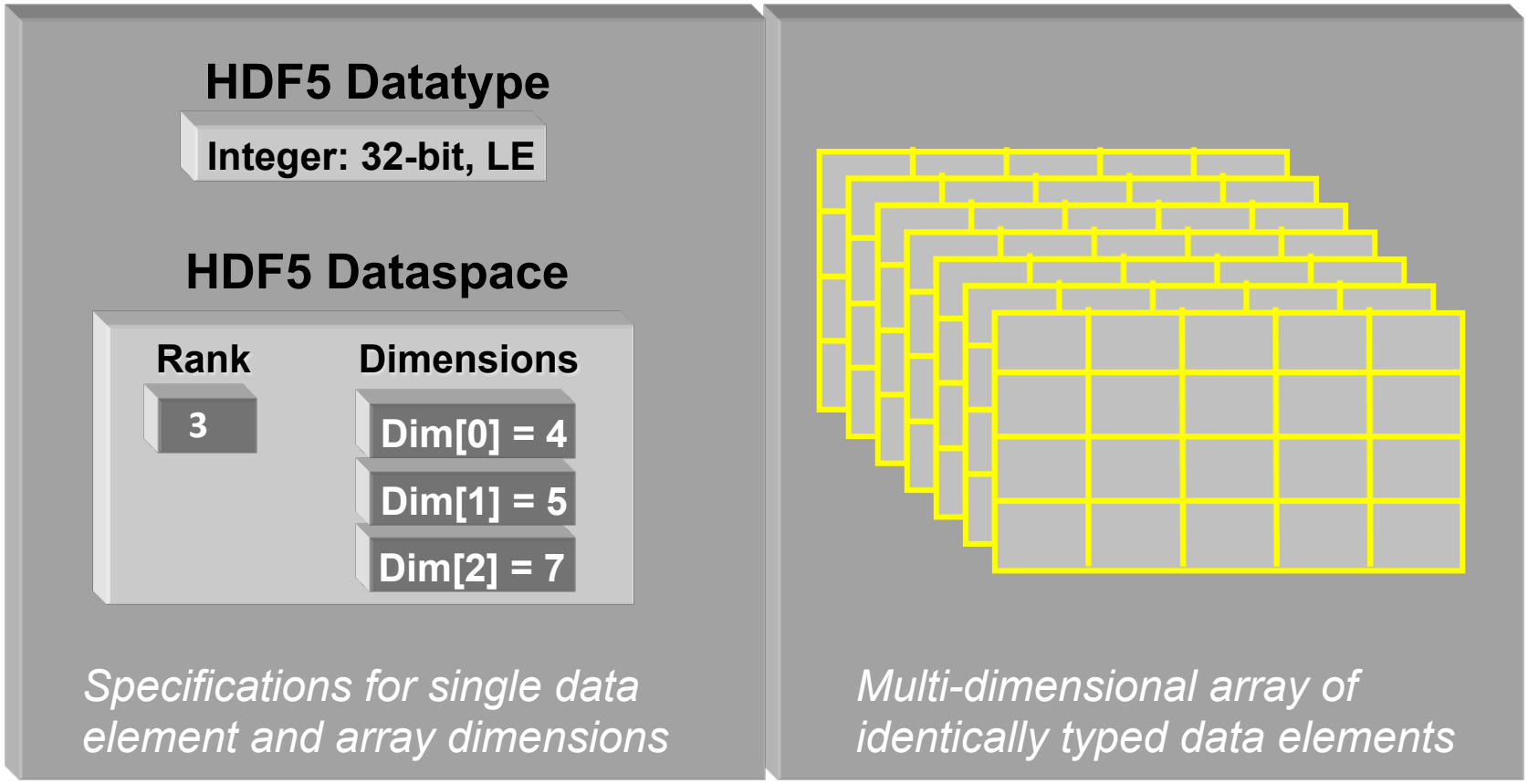
- File – Container for objects
- Groups – provide structure among objects
- Datasets – where the primary data goes
  - Data arrays
  - Rich set of datatype options
  - Flexible, efficient storage and I/O
- Attributes, for metadata

Everything else is built essentially from these parts.

Structures to organize objects



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 Attributes

- Typically contain user metadata
- Have a name and a value
- Attributes “decorate” HDF5 objects
- Value is described by a datatype and a dataspace
- Analogous to a dataset, but do not support partial I/O operations; nor can they be compressed or extended

HDF5 Technology Platform

- HDF5 Abstract Data Model
  - Defines the “building blocks” for data organization and specification
  - Files, Groups, Links, Datasets, Attributes, Datatypes, Dataspaces

- HDF5 Software
  - Tools
  - Language Interfaces
  - HDF5 Library

- HDF5 Binary File Format
  - Bit-level organization of HDF5 file
  - Defined by HDF5 File Format Specification

HDF5 Software Distribution

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

- Latest release: HDF5 1.8.15

HDF5 source code:

- Written in C, and includes optional C++, Fortran 90 APIs, and High Level APIs
- Contains command-line utilities (h5dump, h5repack, h5diff, ..) and compile scripts

HDF5 pre-built binaries:

- When possible, includes C, C++, F90, and High Level libraries. Check ./lib/libhdf5.settings file.
- Built with and require the SZIP and ZLIB external libraries

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 Interfaces:

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 optionally defined
  - ✓Creation properties (e.g., use chunking storage)
  - ✓Access properties

Basic Functions

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*

Useful Tools For New Users

OVERVIEW OF PARALLEL HDF5 DESIGN

Parallel HDF5 Requirements

- Parallel HDF5 should allow multiple processes to perform I/O to an HDF5 file at the same time
  - Single file image for all processes
  - Compare with one file per process design:
    - Expensive post processing
    - Not usable by different number of processes
    - Too many files produced for file system
- Parallel HDF5 should use a standard parallel I/O interface
- Must be portable to different platforms

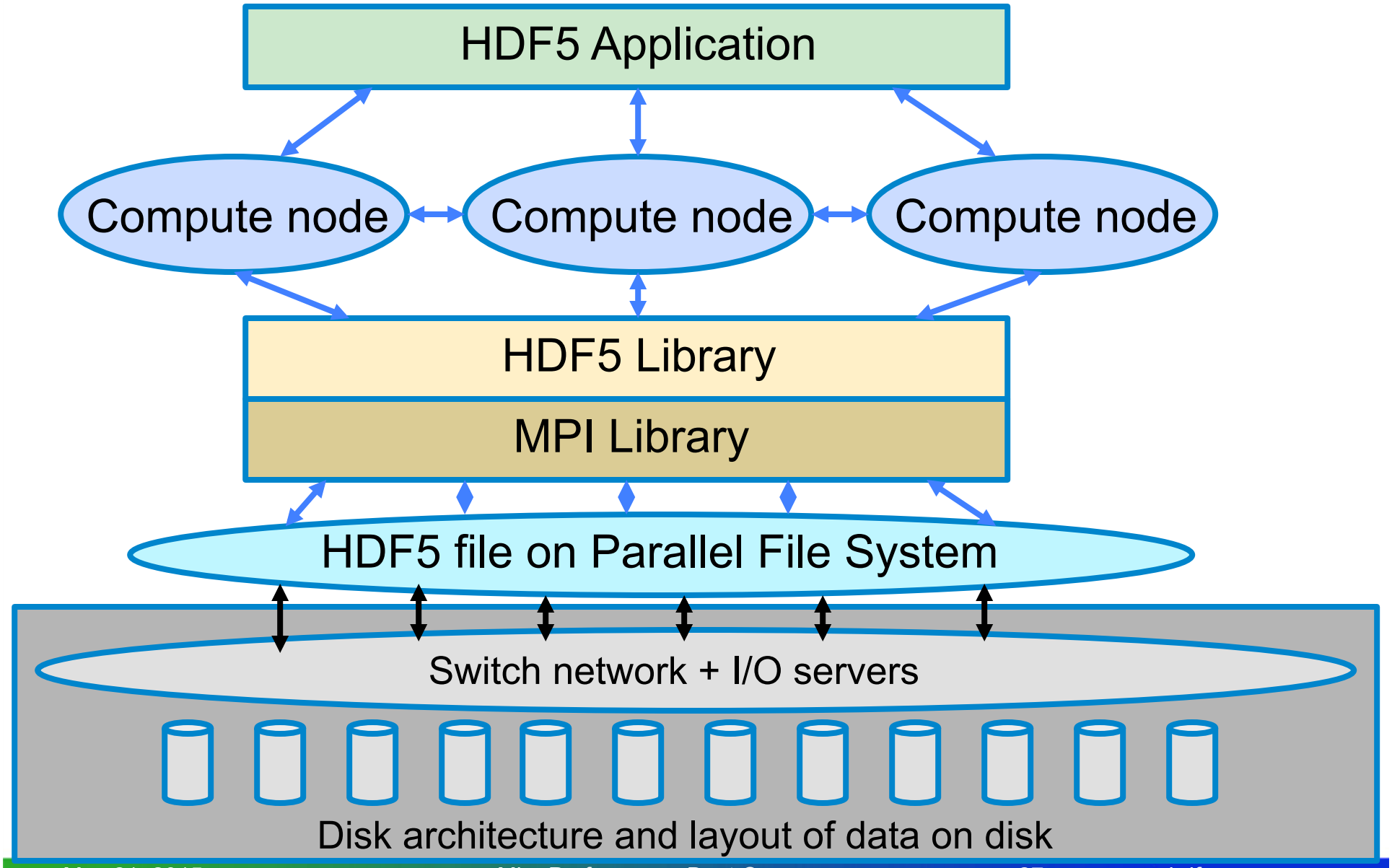
Design requirements, cont

- Support Message Passing Interface (MPI) programming
- Parallel HDF5 files compatible with serial HDF5 files
  - Shareable between different serial or parallel platforms

Design Dependencies

- MPI with MPI-IO
  - MPICH, OpenMPI
  - Vendor's MPI
- Parallel file system
  - IBM GPFS
  - Lustre
  - PVFS

PHDF5 implementation layers



MPI-IO VS. HDF5

MPI-IO

- MPI-IO is strictly an I/O API
- It treats the data file as a “linear byte stream” and each MPI application needs to provide its own file and data representations to interpret those bytes

MPI-IO

- All data stored are *machine dependent* except the “external32” representation
- External32 is defined in Big Endianness
  - Little-endian machines have to do the data conversion in both read or write operations
  - 64-bit sized data types may lose information

MPI-IO vs. HDF5

- HDF5 is data management software
- It stores data and metadata according to the HDF5 data format definition
- HDF5 file is self-describing
  - Each machine can store the data in its own native representation for efficient I/O without loss of data precision
  - Any necessary data representation conversion is done by the HDF5 library automatically

PARALLEL HDF5 CONSISTENCY SEMANTICS

Consistency Semantics

- Consistency Semantics: Rules that define the outcome of multiple, possibly concurrent, accesses to an object or data structure by one or more processes in a computer system.

Parallel HDF5 Consistency Semantics

- Parallel HDF5 library defines a set of consistency semantics to let users know what to expect when processes access data managed by the library.
  - When the changes a process makes are actually visible to itself (if it tries to read back that data) or to other processes that access the same file with independent or collective I/O operations

Parallel HDF5 Consistency Semantics

- Same as MPI-I/O semantics

Process 0	Process 1
MPI_File_write_at()	
MPI_Barrier()	MPI_Barrier()
	MPI_File_read_at()

- Default MPI-I/O semantics doesn't guarantee atomicity or sequence of above calls!
- Problems may occur (although we haven't seen any) when writing/reading HDF5 metadata or raw data

Parallel HDF5 Consistency Semantics

- MPI I/O provides atomicity and sync-barrier-sync features to address the issue
- PHDF5 follows MPI I/O
  - H5Fset_mpio_atomicity function to turn on MPI atomicity
  - H5Fsync function to transfer written data to storage device (in implementation now)

Parallel HDF5 Consistency Semantics

- For more information see “Enabling a strict consistency semantics model in parallel HDF5” linked from the HDF5 *H5Fset_mpi_atomicity* Reference Manual page[1]

[1] http://www.hdfgroup.org/HDF5/doc/RM/Advanced/PHDF5FileConsistencySemantics/PHDF5FileConsistencySemantics.pdf

HDF5 PARALLEL PROGRAMMING MODEL

How to compile PHDF5 applications

- h5pcc – HDF5 C compiler command
  - Similar to mpicc
- h5pfc – HDF5 F90 compiler command
  - Similar to mpif90
- To compile:
  - % h5pcc h5prog.c
  - % h5pfc h5prog.f90

Programming restrictions

- PHDF5 opens a parallel file with an MPI communicator
  - Returns a file ID
  - Future access to the file via the file ID
  - All processes must participate in collective PHDF5 APIs
  - Different files can be opened via different communicators

Collective HDF5 calls

- All HDF5 APIs that modify structural metadata are collective!
  - File operations
    - `H5Fcreate, H5Fopen, H5Fclose, etc`
  - Object creation
    - `H5Dcreate, H5Dclose, etc`
  - Object structure modification (e.g., dataset extent modification)
    - `H5Dset_extent, etc`
- http://www.hdfgroup.org/HDF5/doc/RM/CollectiveCalls.html

Other HDF5 calls

- Array data transfer can be collective or independent
  - Dataset operations: `H5Dwrite, H5Dread`
- Collectiveness is indicated by function parameters, not by function names as in MPI API

What does PHDF5 support ?

- After a file is opened by the processes of a communicator
  - All parts of file are accessible by all processes
  - All objects in the file are accessible by all processes
  - Multiple processes may write to the same data array
  - Each process may write to individual data array

PHDF5 API languages

- C and F90, 2003 language interfaces
- Most platforms with MPI-IO supported. e.g.,
  - IBM BG/x
  - Linux clusters
  - Cray

Programming model

- HDF5 uses access property list to control the file access mechanism
- General model to access HDF5 file in parallel:
  - Set up MPI-IO file access property list
  - Open File
  - Access Data
  - Close File

Moving your sequential application to the HDF5 parallel world

MY FIRST PARALLEL HDF5 PROGRAM

Example of Serial HDF5 C program

Example of Parallel HDF5 C program

WRITING PATTERNS - EXAMPLE

Parallel HDF5 tutorial examples

- For sample programs of how to write different data patterns see:

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

Programming model

- Each process defines memory and file hyperslabs using `H5Sselect_hyperslab`
- Each process executes a write/read call using hyperslabs defined, which can be either collective or independent
- The hyperslab parameters define the portion of the dataset to write to:
  - Contiguous hyperslab
  - Regularly spaced data (column or row)
  - Pattern
  - Blocks

Four processes writing by rows

```
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
```

Parallel HDF5 example code

```
71      /*
72       * Each process defines dataset in memory and writes it to the
73       * hyperslab in the file.
74       */
75      count[0] = dims[0] / mpi_size;
76      count[1] = dims[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);
```

Two processes writing by columns

Four processes writing by pattern

Four processes writing by blocks

```
HDF5 "SDS_blk.h5" {
GROUP "/" {
   DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
      DATA {
         1, 1, 2, 2,
         1, 1, 2, 2,
         1, 1, 2, 2,
         1, 1, 2, 2,
         3, 3, 4, 4,
         3, 3, 4, 4,
         3, 3, 4, 4,
         3, 3, 4, 4
```

Complex data patterns

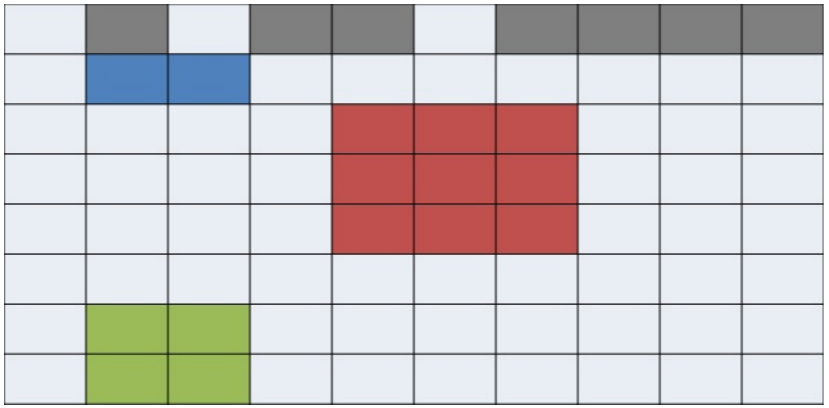
HDF5 doesn't have restrictions on data patterns and data balance

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1	2	3	4				
9	10	11	12				
17	18	19	20				
25	26	27	28				
				37	38	39	40
				45	46	47	48
				53	54	55	56
				61	62	63	64

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Examples of irregular selection



- 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

PERFORMANCE ANALYSIS

Performance analysis

- Some common causes of poor performance
- Possible solutions

My PHDF5 application I/O is slow

"Tuning HDF5 for Lustre File Systems" by Howison, Koziol, Knaak, Mainzer, and Shalf[1]

- Chunking and hyperslab selection
- HDF5 metadata cache
- Specific I/O system hints

[1]http://www.hdfgroup.org/pubs/papers/howison_hdf5_lustre_iasds2010.pdf

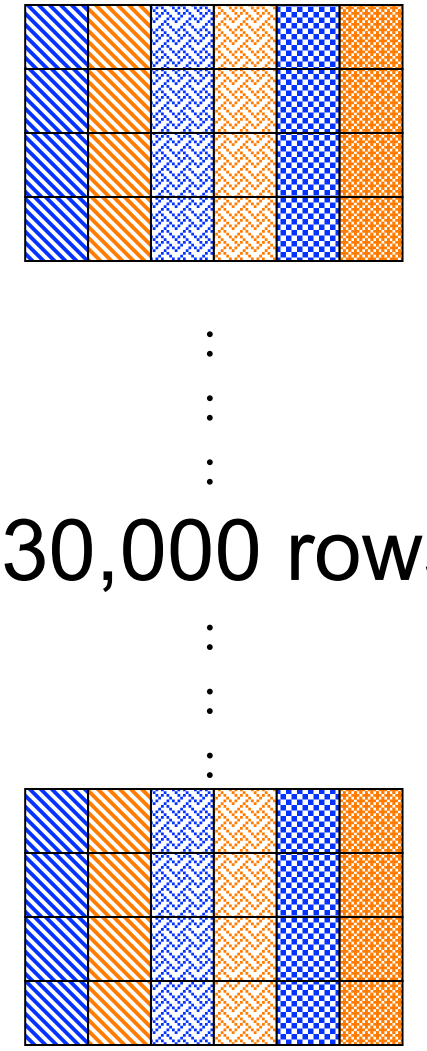
INDEPENDENT VS. COLLECTIVE RAW DATA I/O

Collective vs. independent calls

- MPI definition of collective calls:
  - All processes of the communicator must participate in calls in the right order. E.g.,
    - Process1 Process2
    - call A(); call B(); call A(); call B(); **right**
    - call A(); call B(); call B(); call A(); **wrong**
- Independent means not collective ☺
- Collective is not necessarily synchronous, nor must require communication

Independent vs. collective access

- User reported independent data transfer mode was much slower than the collective data transfer mode
- Data array was tall and thin: 230,000 rows by 6 columns



Debug Slow Parallel I/O Speed(1)

- Writing to one dataset
  - Using 4 processes == 4 columns
  - HDF5 datatype is 8-byte doubles
  - 4 processes, 1000 rows == 4x1000x8 = 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.
- Difference of 2 seconds for 1000 more rows = 32,000 bytes.
- Speed of 16KB/sec!!! *Way too slow.*

Debug slow parallel I/O speed(2)

- Build a version of PHDF5 with
  - ./configure --enable-debug --enable-parallel …
  - This allows the tracing of MPIO I/O calls in the HDF5 library.
- E.g., to trace
  - MPI_File_read_xx and MPI_File_write_xx calls
  - % setenv H5FD_mpio_Debug “rw”

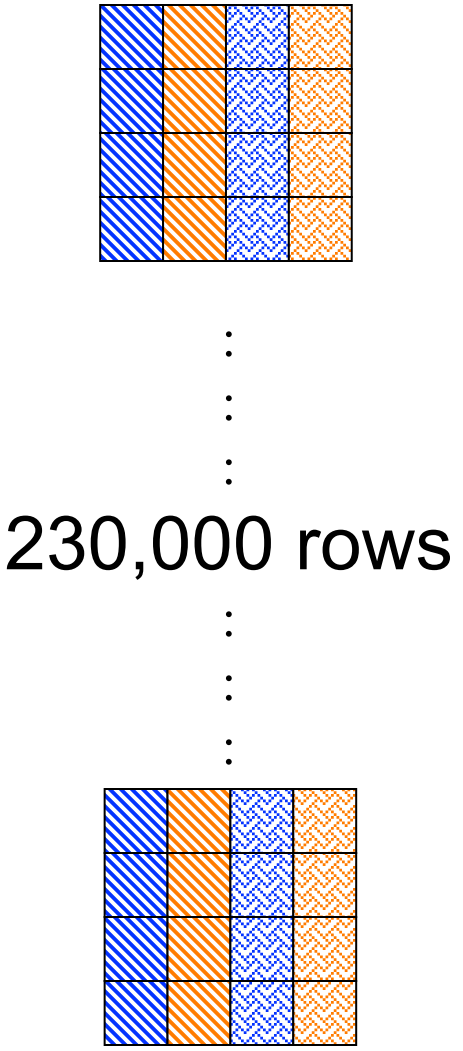
Debug slow parallel I/O speed(3)

```
% 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
…
```

- Total of 4000 of these little 8 bytes writes == 32,000 bytes.

Independent calls are many and small

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

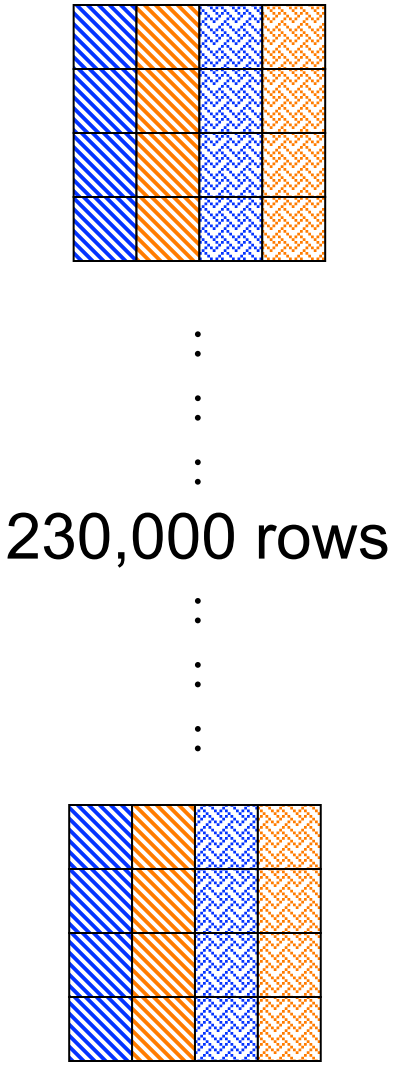


Debug slow parallel I/O speed (4)

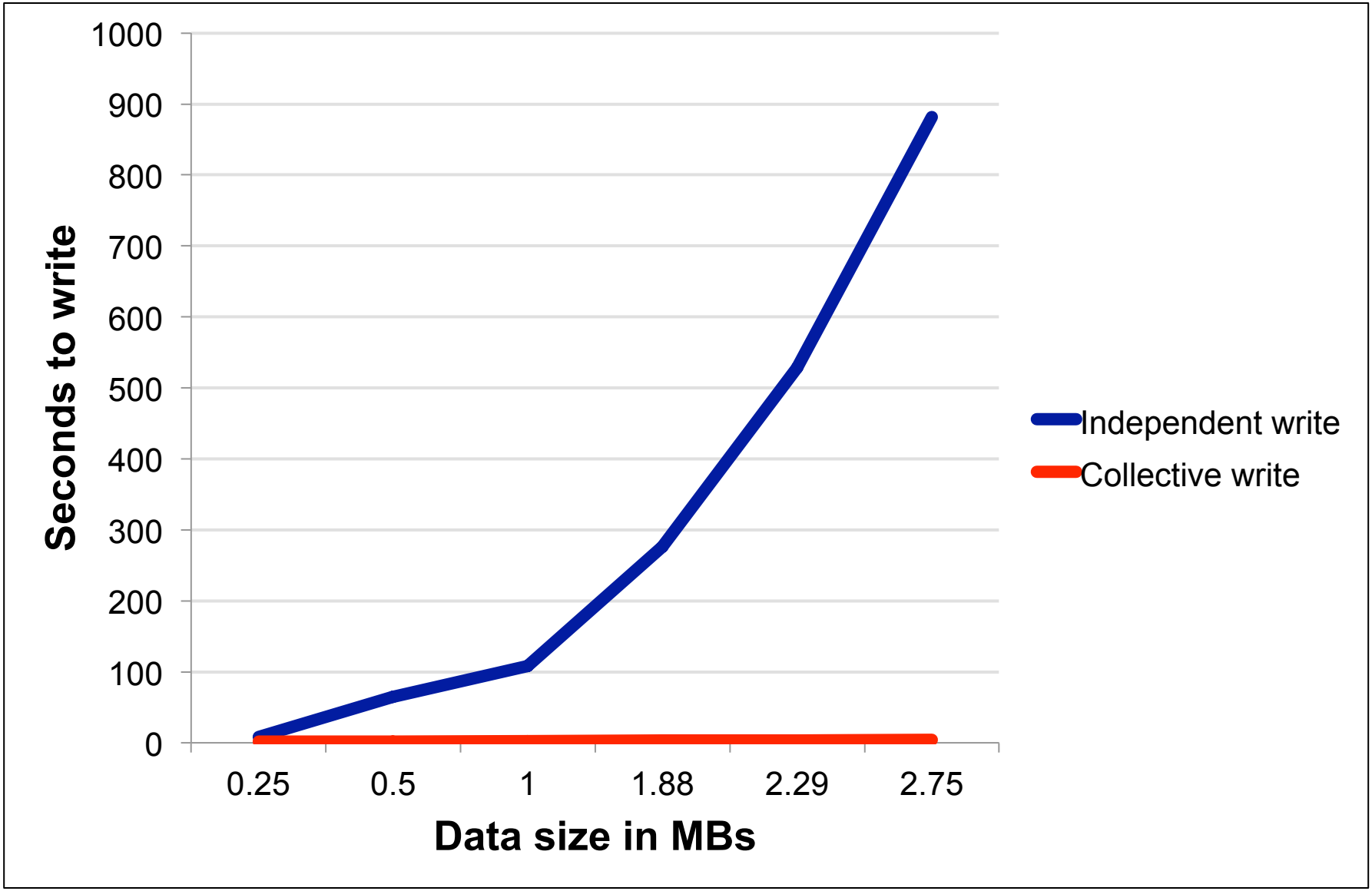
```
% setenv H5FD_mpio_Debug 'rw'
% mpirun -np 4 ./a.out 1000     # Indep., Chunked by column.
in H5FD_mpio_write  mpi_off=0         size_i=96
in H5FD_mpio_write  mpi_off=0         size_i=96
in H5FD_mpio_write  mpi_off=0         size_i=96
in H5FD_mpio_write  mpi_off=0         size_i=96
in H5FD_mpio_write  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
…
Execution time: 0.011599 s.
```

Use collective mode or chunked storage

- Collective I/O will combine many small independent calls into few but bigger calls
- Chunks of columns speeds up too



Collective vs. independent write



Collective I/O in HDF5

- Set up using a Data Transfer Property List (DXPL)
- All processes must participate in the I/O call (H5Dread/write) with a selection (which could be a NULL selection)
- Some cases where collective I/O is not used even when the use asks for it:
  - Data conversion
  - Compressed Storage
  - Chunking Storage:
    - When the chunk is not selected by a certain number of processes

Enabling Collective Parallel I/O with HDF5

```
/* Set up file access property list w/parallel I/O access */
fa_plist_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fa_plist_id, comm, info);

/* Create a new file collectively */
file_id = H5Fcreate(filename, H5F_ACC_TRUNC,
       H5P_DEFAULT, fa_plist_id);

/* <omitted data decomposition for brevity> */

/* Set up data transfer property list w/collective MPI-IO */
dx_plist_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(dx_plist_id, H5FD_MPIO_COLLECTIVE);

/* Write data elements to the dataset */
status = H5Dwrite(dset_id, H5T_NATIVE_INT,
       memspace, filespace, dx_plist_id, data);
```

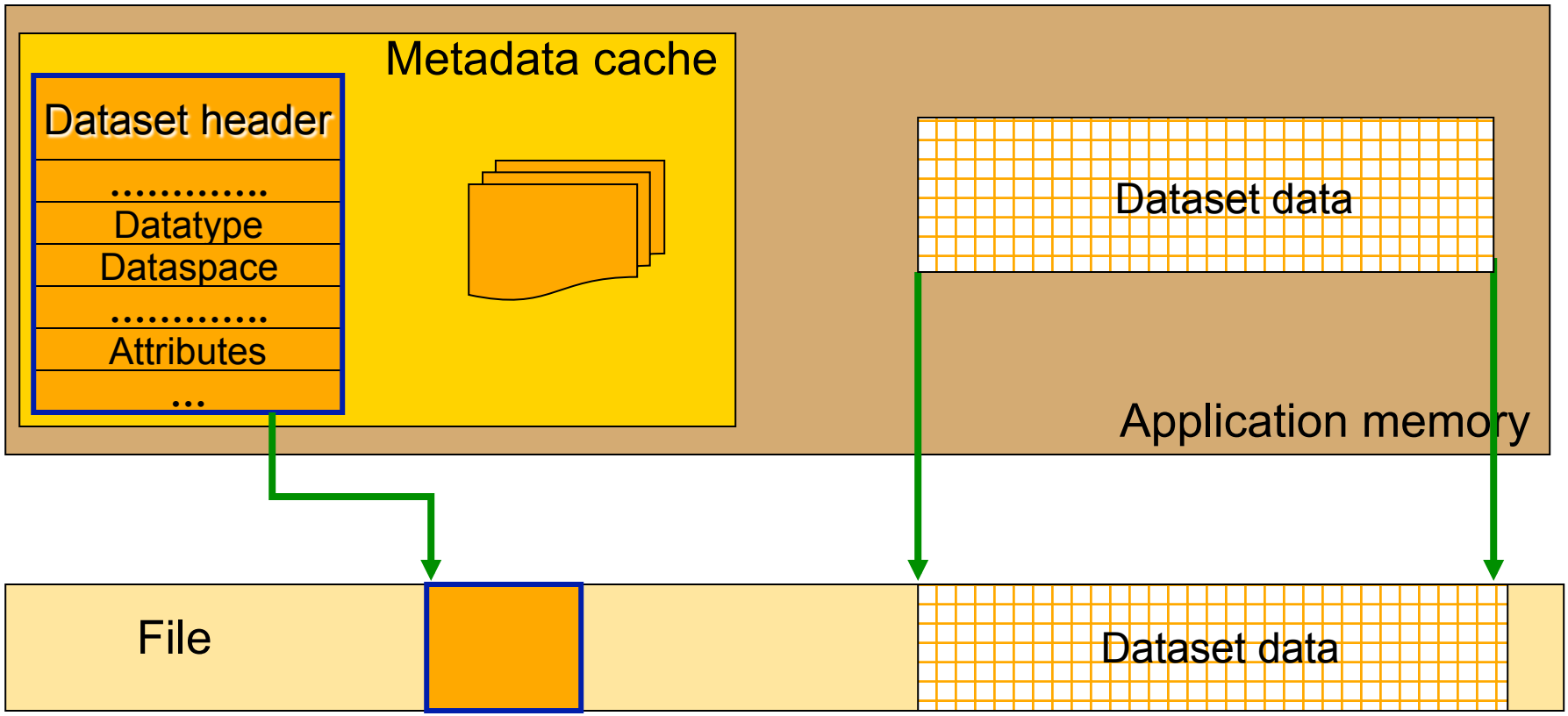
Collective I/O in HDF5

- Can query Data Transfer Property List (DXPL) after I/O for collective I/O status:
  - H5Pget_mpio_actual_io_mode
    - Retrieves the type of I/O that HDF5 actually performed on the last parallel I/O call
  - H5Pget_mpio_no_collective_cause
    - Retrieves local and global causes that broke collective I/O on the last parallel I/O call
  - H5Pget_mpio_actual_chunk_opt_mode
    - Retrieves the type of chunk optimization that HDF5 actually performed on the last parallel I/O call. This is not necessarily the type of optimization requested

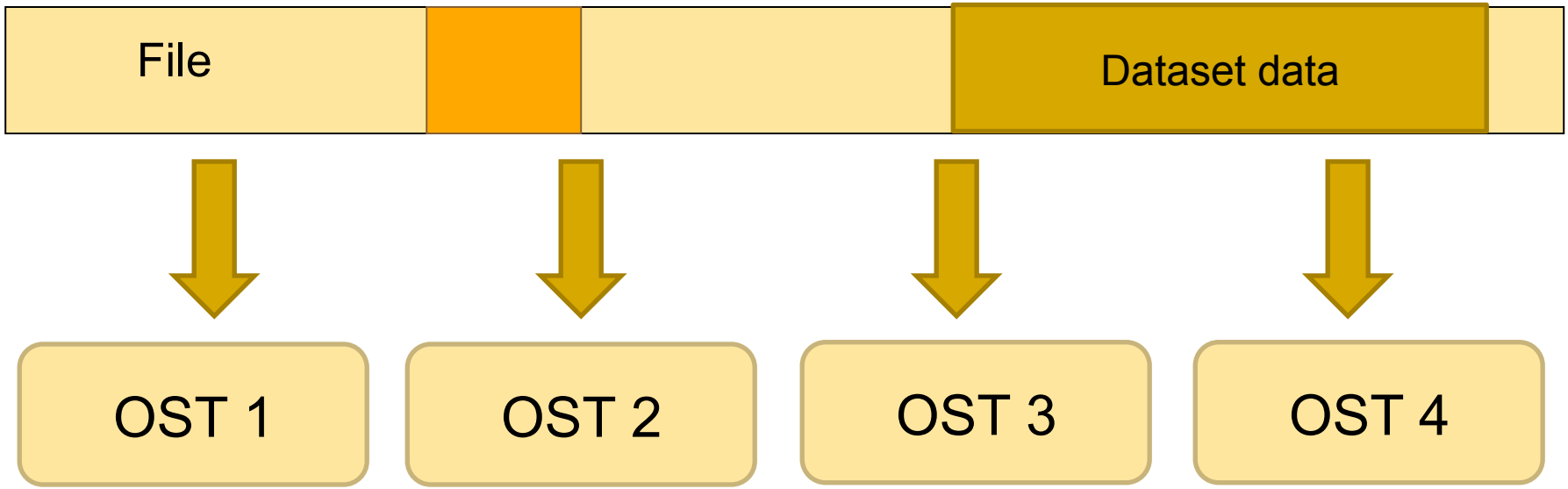
EFFECT OF HDF5 STORAGE

Contiguous storage

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



On a parallel file system

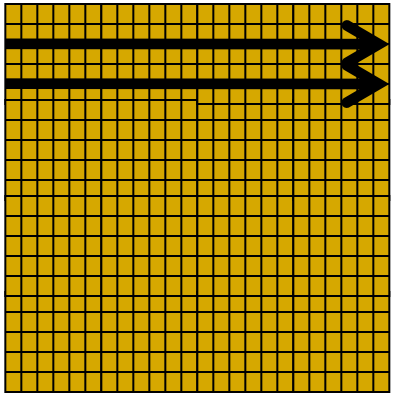


The file is striped over multiple OSTs depending on the stripe size and stripe count that the file was created with.

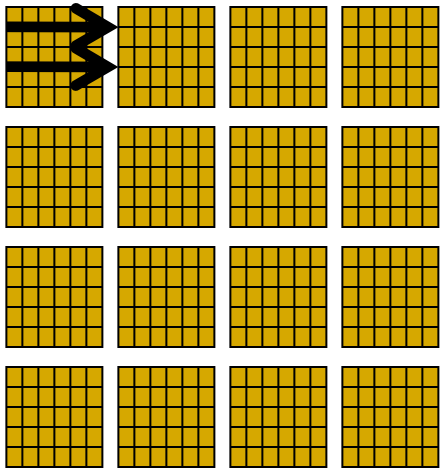
Chunked storage

- Data is stored in chunks of predefined size
  - Two-dimensional instance may be referred to as data tiling
- HDF5 library writes/reads the whole chunk

Contiguous

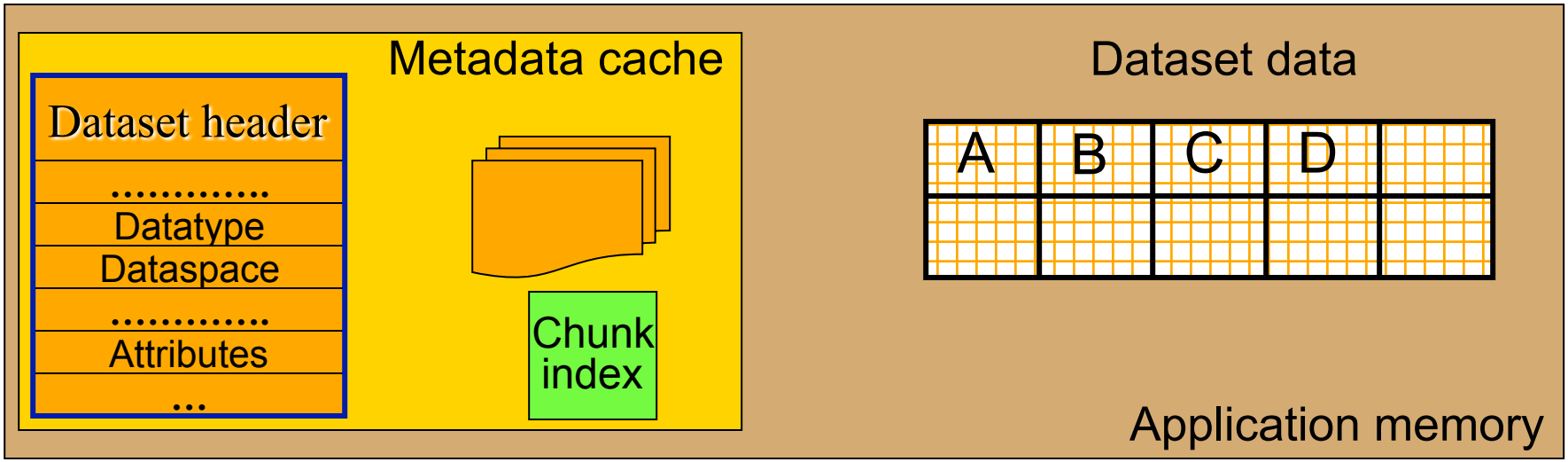


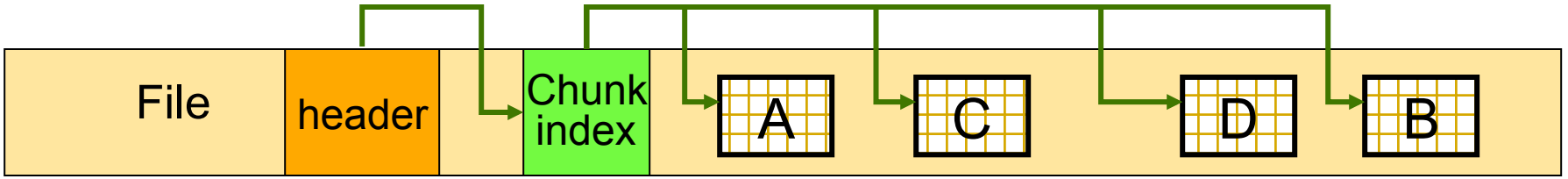
Chunked



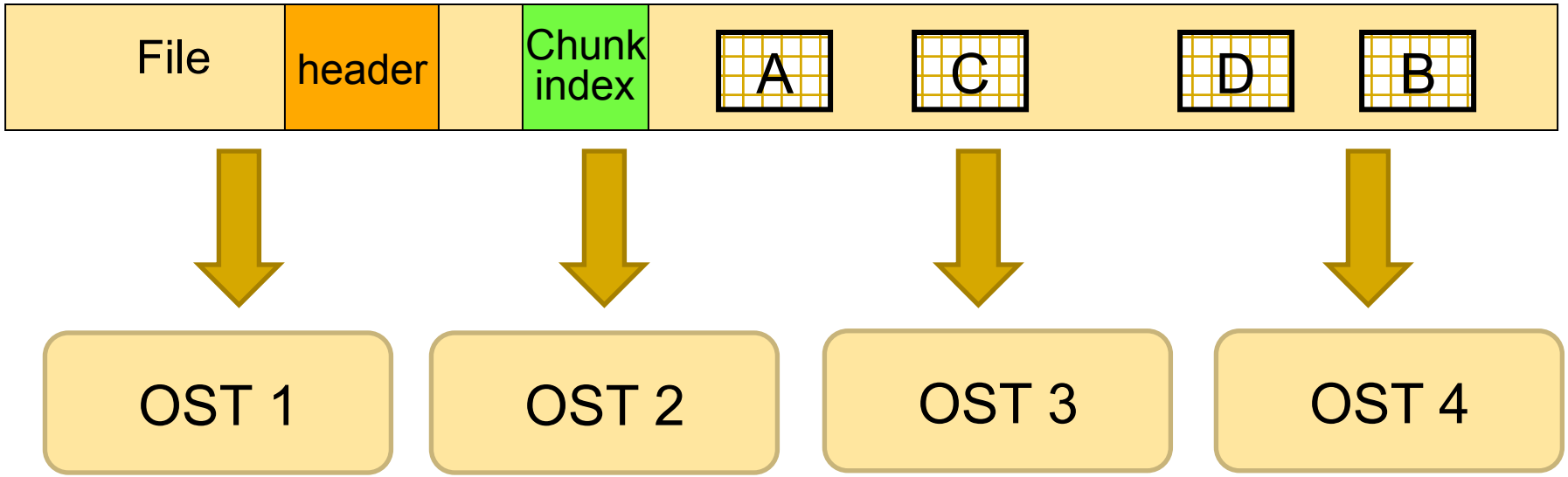
Chunked storage (cont.)

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





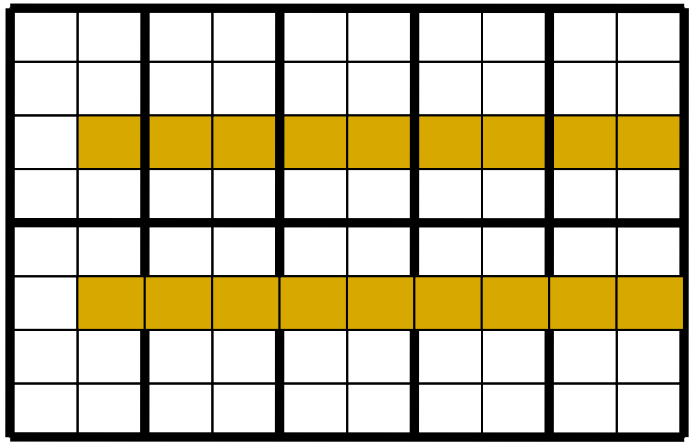
On a parallel file system

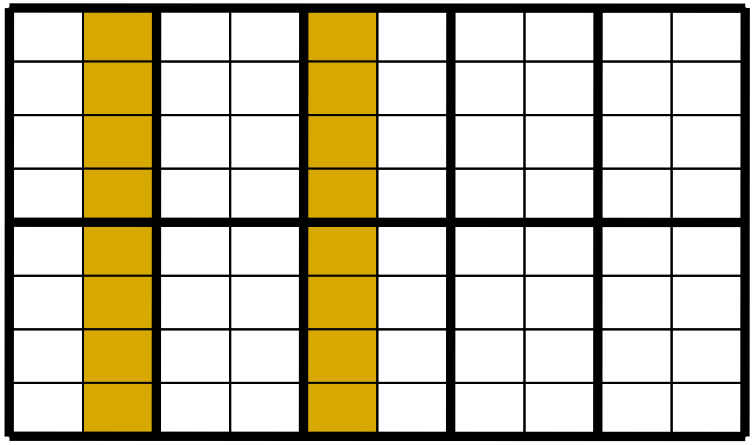


The file is striped over multiple OSTs depending on the stripe size and stripe count that the file was created with

Which is better for performance?

- It depends!!
- Consider these selections:



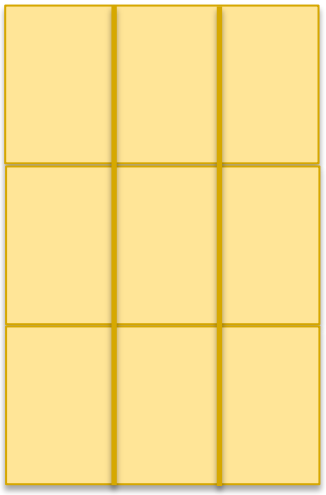


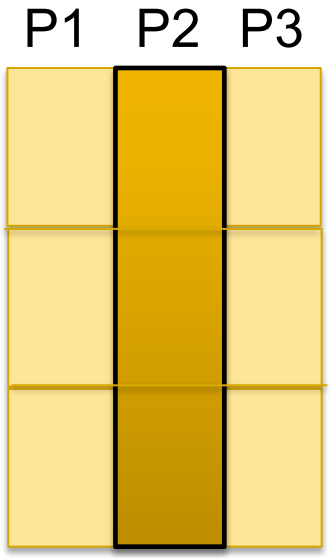
- If contiguous: 2 seeks
- If chunked: 10 seeks

- If contiguous: 16 seeks
- If chunked: 4 seeks

Add to that striping over a Parallel File System, which makes this problem very hard to solve!

Chunking and hyperslab selection



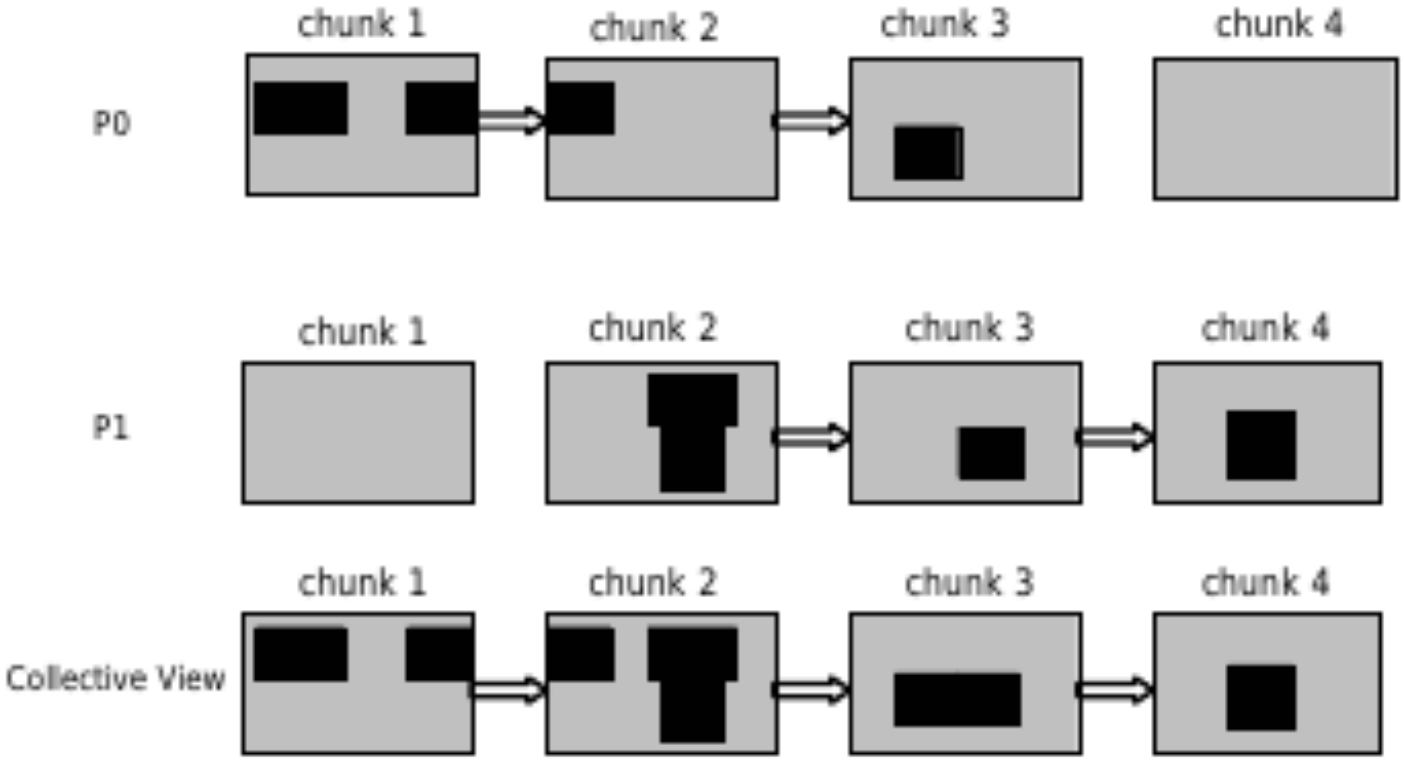


- When writing or reading, try to use hyperslab selections that coincide with chunk boundaries.

Parallel I/O on chunked datasets

- Multiple options for performing I/O when collective:
  - Operate on all chunks in one collective I/O operation: “Linked chunk I/O”
  - Operate on each chunk collectively: “Multi-chunk I/O”
  - Break collective I/O and perform I/O on each chunk independently (also in “Multi-chunk I/O” algorithm)

Linked chunk I/O

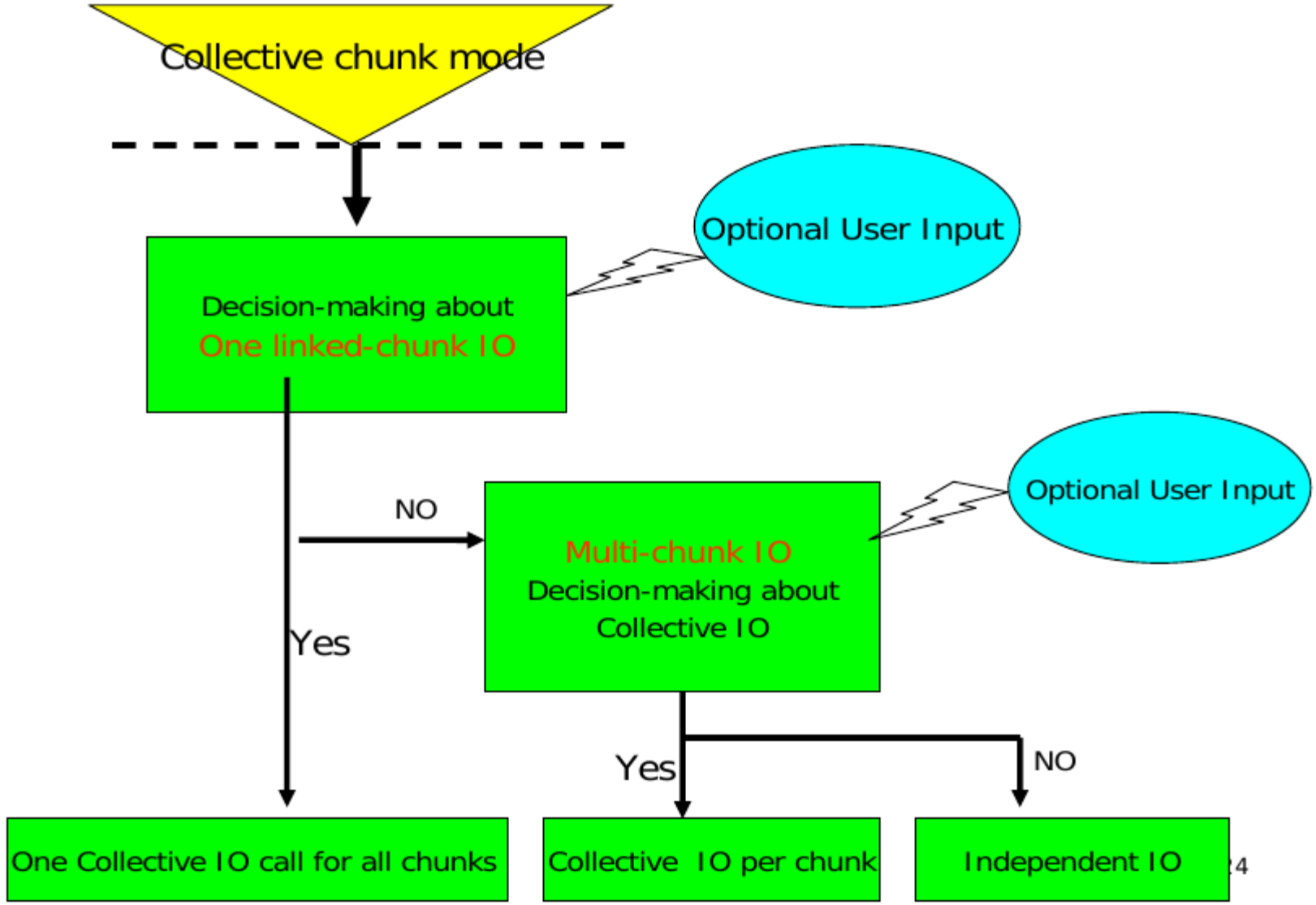


- One MPI Collective I/O Call

Multi-chunk I/O

- Collective I/O per chunk
- Determine for each chunk if enough processes have a selection inside to do collective I/O
- If not enough, use independent I/O

Decision making



EFFECT OF HDF5 METADATA CACHE

Parallel HDF5 and Metadata

- Metadata operations:
  - Creating/removing a dataset, group, attribute, etc…
  - Extending a dataset's dimensions
  - Modifying group hierarchy
  - etc …
- All operations that modify metadata are collective, i.e., all processes have to call that operation:
  - If you have 10,000 processes running your application, and one process needs to create a dataset, ***ALL*** processes must call H5Dcreate to create 1 dataset.

Space allocation

- Allocating space at the file's EOF is very simple in serial HDF5 applications:
  - The EOF value begins at offset 0 in the file
  - When space is required, the EOF value is incremented by the size of the block requested.
- Space allocation using the EOF value in parallel HDF5 applications can result in a race condition if processes do not synchronize with each other:
  - Multiple processes believe that they are the sole owner of a range of bytes within the HDF5 file.
- Solution: Make it Collective

Example

- Consider this case, where 2 processes want to create a dataset each.

H5Dcreate(D1)

H5Dcreate(D2)

Each call has to allocate space in file to store the dataset header.

Bytes 4 to 10 in the file are free

Bytes 4 to 10 in the file are free

Conflict!

Example

P1

P2

H5Dcreate(D1)

H5Dcreate(D1)

Allocate space in file to store the dataset header.

Bytes 4 to 10 in the file are free.

Create the dataset.

H5Dcreate(D2)

H5Dcreate(D2)

Allocate space in file to store the dataset header.

Bytes 11 to 17 in the file are free.

Create the dataset.

Metadata cache

- To handle synchronization issues, all HDF5 operations that could potentially modify the metadata in an HDF5 file are required to be collective
  - A list of these routines is available in the HDF5 reference manual: http://www.hdfgroup.org/HDF5/doc/RM/CollectiveCalls.html

Managing the metadata cache

- All operations that **modify** metadata in the HDF5 file are collective:
  - All processes will have the same dirty metadata entries in their cache (i.e., metadata that is inconsistent with what is on disk).
  - Processes are not required to have the same clean metadata entries (i.e., metadata that is in sync with what is on disk).
- Internally, the metadata cache running on process 0 is responsible for managing changes to the metadata in the HDF5 file.
  - All the other caches must retain dirty metadata until the process 0 cache tells them that the metadata is clean (i.e., on disk).

Example

- Metadata Cache is clean for all processes:

P0	P1	P2	P3
E1	E1	E4	E12
E2	E7	E6	E32
E3	E8	E1	E1
E4	E2	E5	E4

Example

- All processes call H5Gcreate that modifies metadata entry E3 in the file:

P0	P1	P2	P3
E3	E3	E3	E3
E1	E1	E4	E12
E2	E7	E6	E32
E4	E8	E1	E1

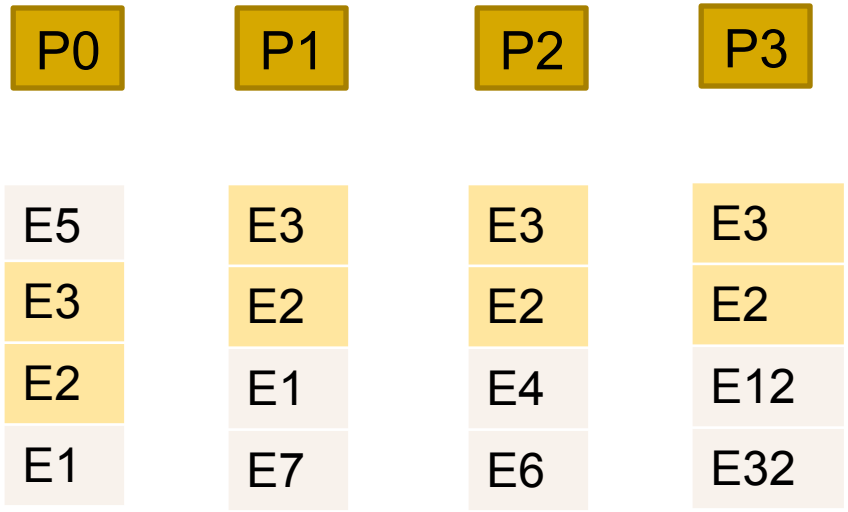
Example

- All processes call H5Dcreate that modifies metadata entry E2 in the file:

P0	P1	P2	P3
E3	E3	E3	E3
E2	E2	E2	E2
E1	E1	E4	E12
E4	E7	E6	E32

Example

- Process 0 calls H5Dopen on a dataset accessing entry E5



Flushing the cache

- Initiated when:
  - The size of dirty entries in cache exceeds a certain threshold
  - The user calls a flush
- The actual flush of metadata entries to disk is currently implemented in two ways:
  - Single Process (Process 0) write
  - Distributed write

Single Process (Process 0) write

- All processes enter a synchronization point.
- Process 0 writes all the dirty entries to disk while other processes wait
- Process 0 marks all the dirty entries as clean
- Process 0 broadcasts the cleaned entries to all processes, which mark them as clean too

Distributed write

- All processes enter a synchronization point.
- Process 0 broadcasts the metadata that needs to be flushed to all processes
- Each process algorithmically determines which metadata cache entries it should write, and writes them to disk independently
- All processes mark the flushed metadata as clean

PARALLEL TOOLS

Parallel tools

- h5perf
  - Performance measuring tool showing I/O performance for different I/O APIs

h5perf

- An I/O performance measurement tool
- Tests 3 File I/O APIs:
  - 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 upper limits

Useful parallel HDF5 links

- Parallel HDF information site

  http://www.hdfgroup.org/HDF5/PHDF5/
- Parallel HDF5 tutorial available at

  http://www.hdfgroup.org/HDF5/Tutor/
- HDF Help email address

  help@hdfgroup.org

UPCOMING FEATURES IN HDF5

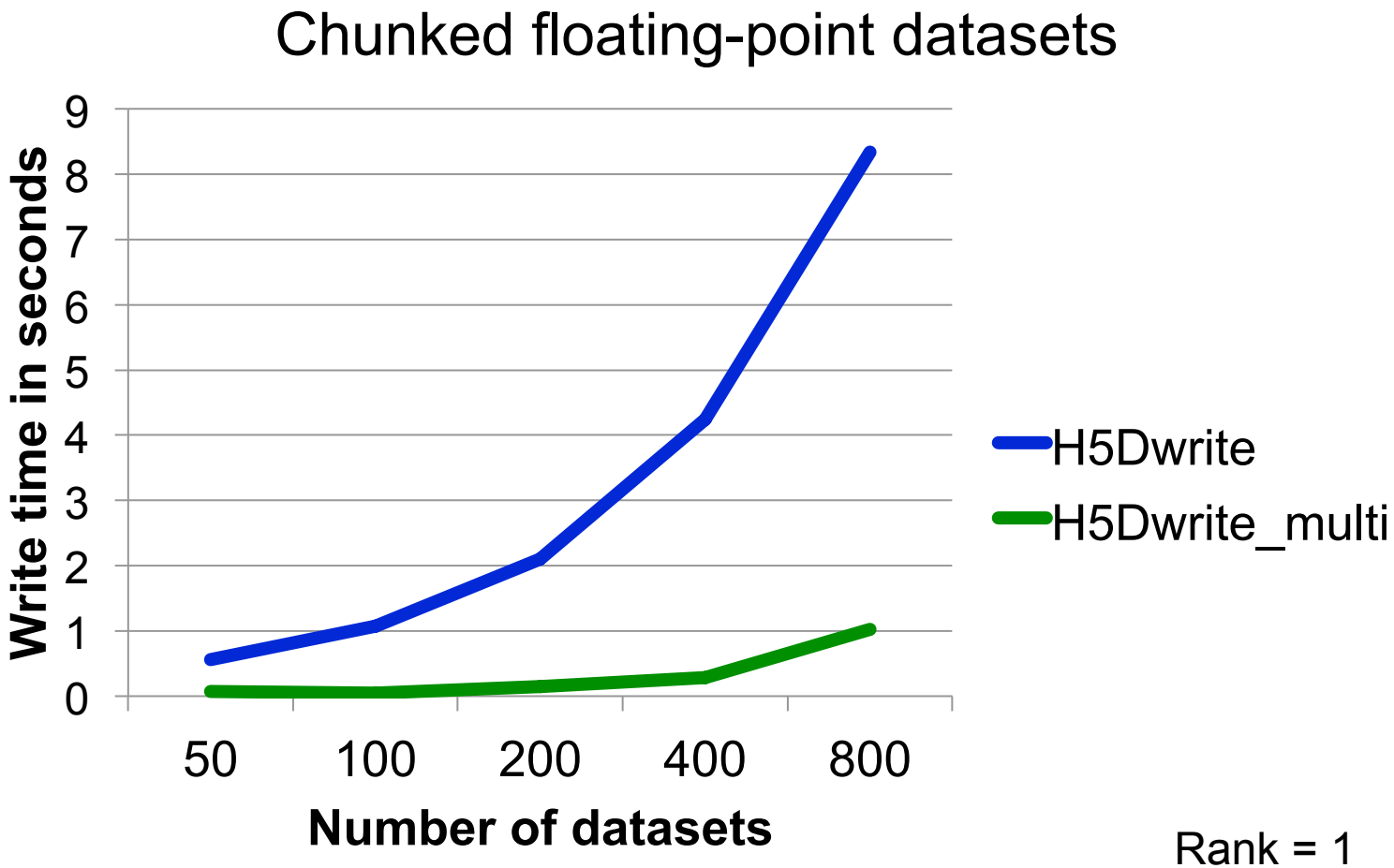
PHDF5 Improvements in Progress

- Multi-dataset read/write operations
- Avoiding file truncation
- Collective object open (avoiding “metadata read storm”)
- Metadata aggregation and page buffering

PHDF5 Improvements in Progress

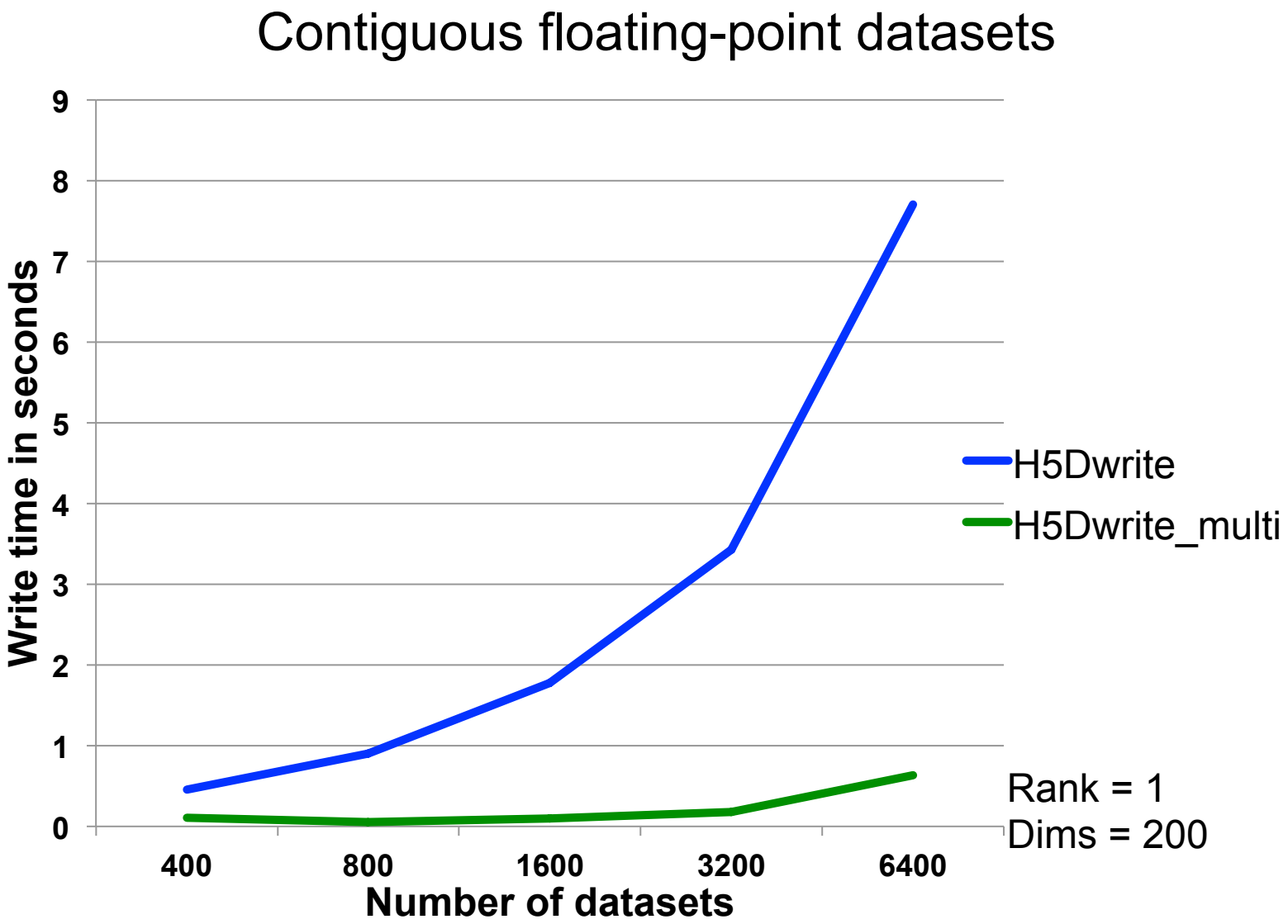
- Multi-dataset read/write operations
  - Allows single collective operation on multiple datasets
    - Similar to PnetCDF “write-combining” feature
  - H5Dmulti_read/write(<array of datasets, selections, etc>)
  - Order of magnitude speedup

H5Dwrite vs. H5Dwrite_multi



Rank = 1
Dims = 200
Chunk size = 20

H5Dwrite vs. H5Dwrite_multi



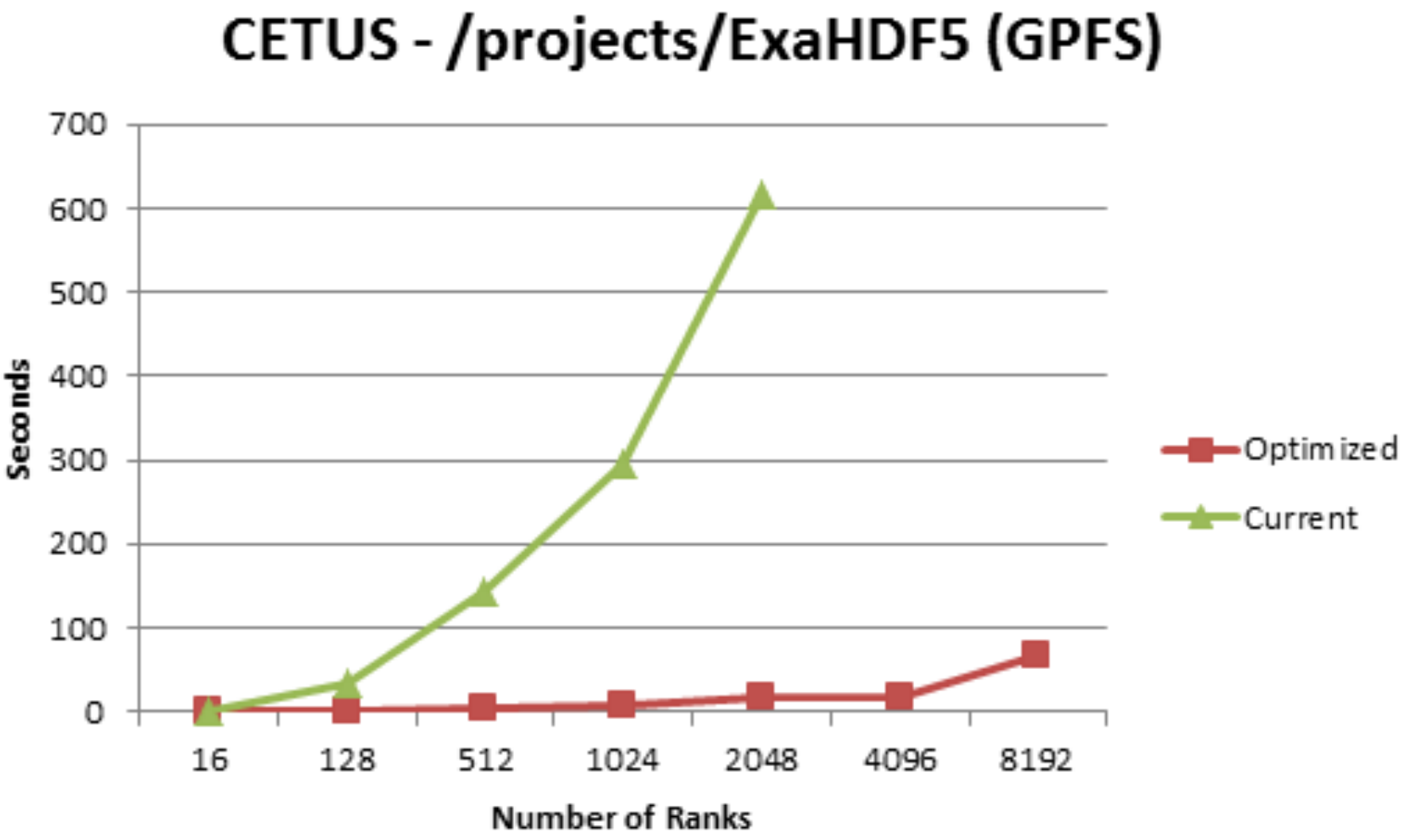
PHDF5 Improvements in Progress

- Avoid file truncation
  - File format currently requires call to truncate file, when closing
  - Expensive in parallel (MPI_File_set_size)
  - Change to file format will eliminate truncate call

PHDF5 Improvements in Progress

- Collective Object Open
  - Currently, object open is independent
  - All processes perform I/O to read metadata from file, resulting in I/O storm at file system
  - Change will allow a single process to read, then broadcast metadata to other processes

Collective Object Open Performance



Research Focus -

AUTOTUNING

Autotuning Background

- Software Autotuning:
  - Employ empirical techniques to evaluate a set of alternative mappings of computation kernels to an architecture and select the mapping that obtains the best performance.
- Autotuning Categories:
  - Self-tuning library generators such as ATLAS, PhiPAC and OSKI for linear algebra, etc.
  - Compiler-based autotuners that automatically generate and search a set of alternative implementations of a computation
  - Application-level autotuners that automate empirical search across a set of parameter values proposed by the application programmer

HDF5 Autotuning

- Why?
  - Because the dominant I/O support request at NERSC is poor I/O performance, many/most of which can be solved by enabling Lustre striping, or tuning another I/O parameter
  - *Users shouldn't have to figure this stuff out!*
- Two Areas of Focus:
  - Evaluate techniques for autotuning HPC application I/O
    - File system, MPI, HDF5
  - Record and Replay HDF5 I/O operations

Autotuning HPC I/O

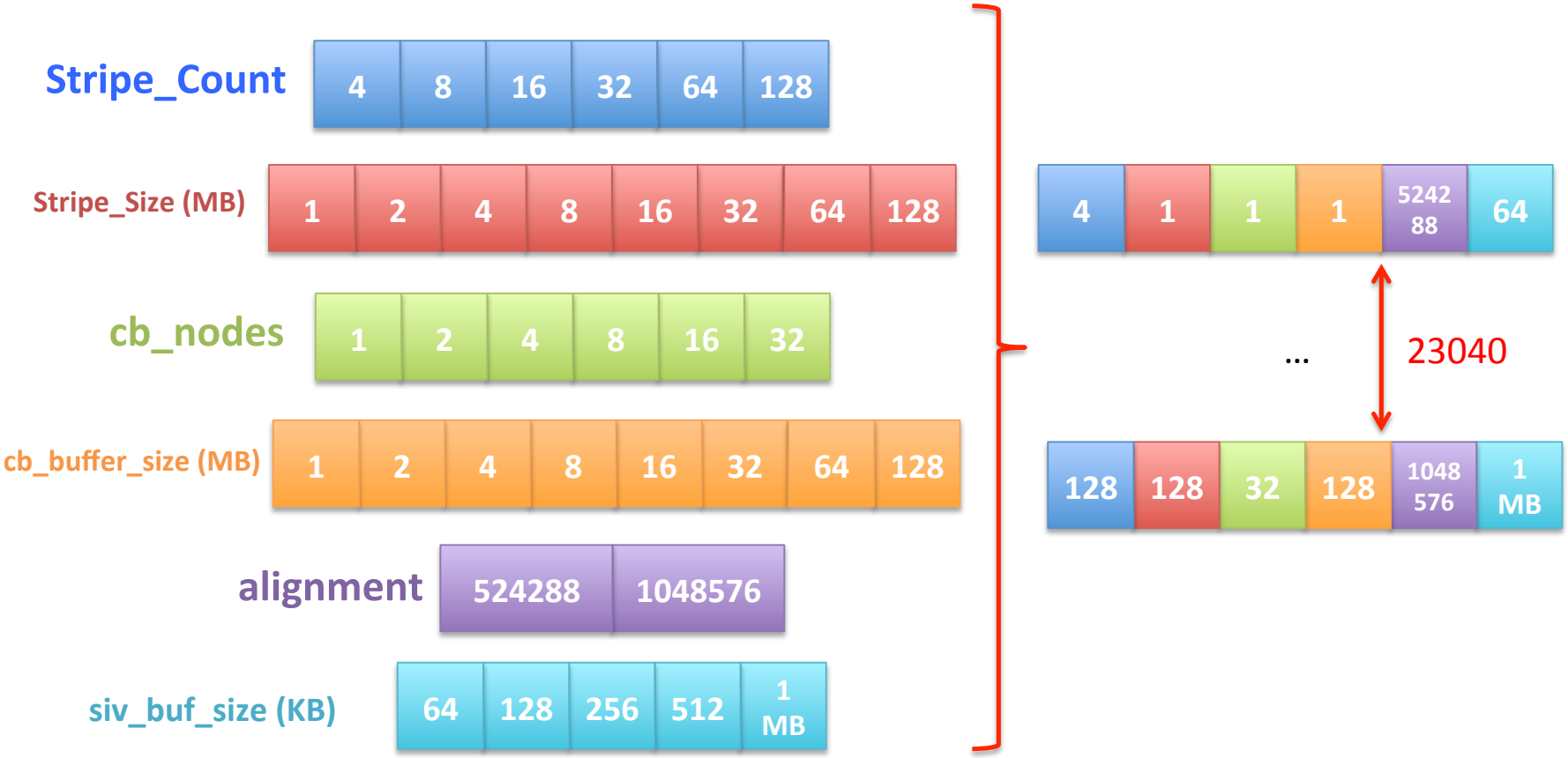
- Goal: Avoid tuning each application to each machine and file system
  - Create I/O autotuner library that can inject "optimal" parameters for I/O operations on a given system
- Using Darshan* tool to create wrappers for HDF5 calls
  - Application can be dynamically linked with I/O autotuning library
  - No changes to application or HDF5 library
- Using several HPC applications currently:
  - VPIC, GCRM, Vorpal

* - http://www.mcs.anl.gov/research/projects/darshan/

Autotuning HPC I/O

- Initial parameters of interest
  - File System (Lustre): stripe count, stripe unit
  - MPI-I/O: Collective buffer size, coll. buffer nodes
  - HDF5: Alignment, sieve buffer size

The whole space visualized



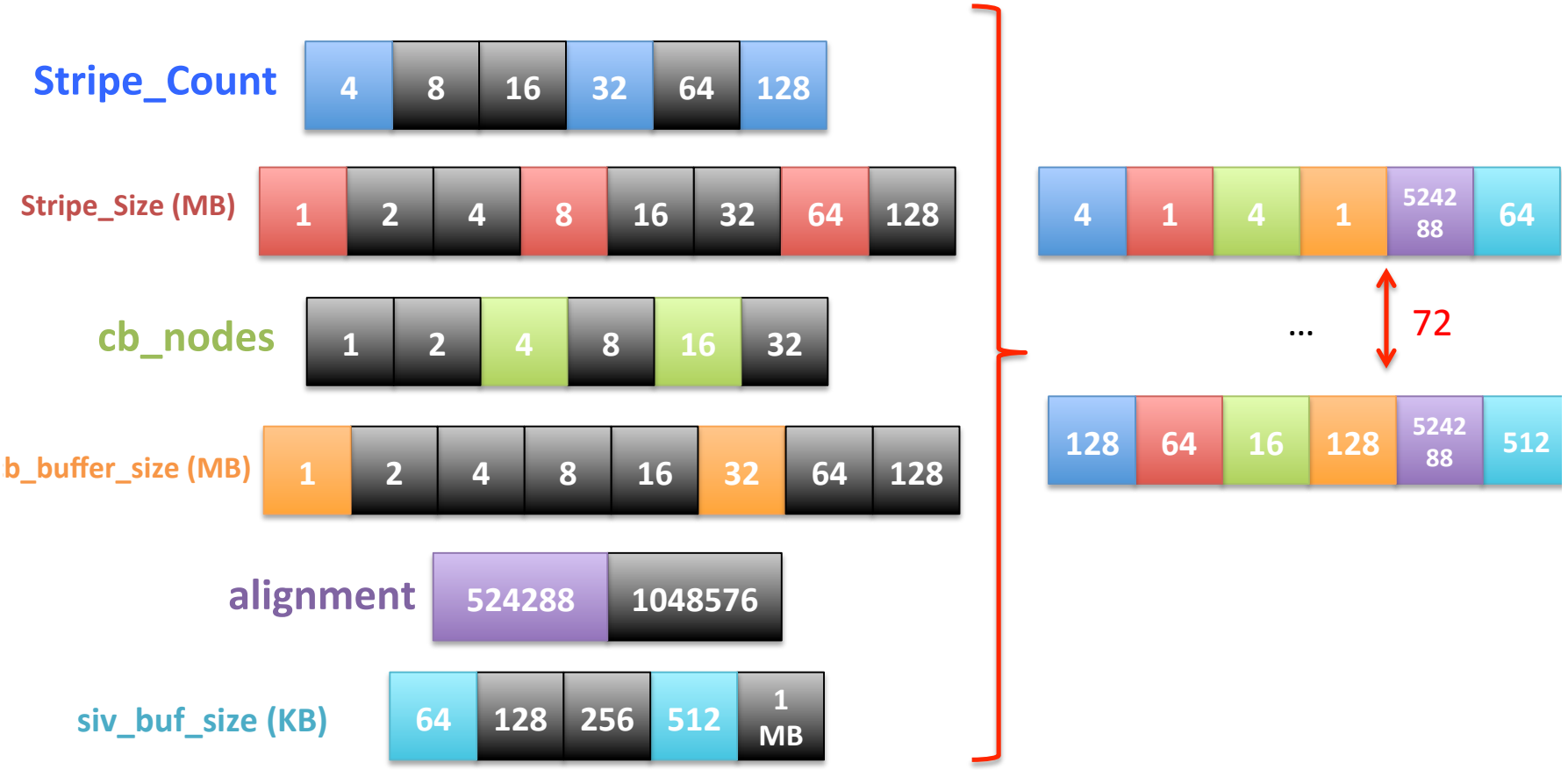
Autotuning HPC I/O

- Autotuning Exploration/Generation Process:
  - Iterate over running application many times:
    - Intercept application's I/O calls
    - Inject autotuning parameters
    - Measure resulting performance
- Analyze performance information from many application runs to create configuration file, with best parameters found for application/ machine/file system

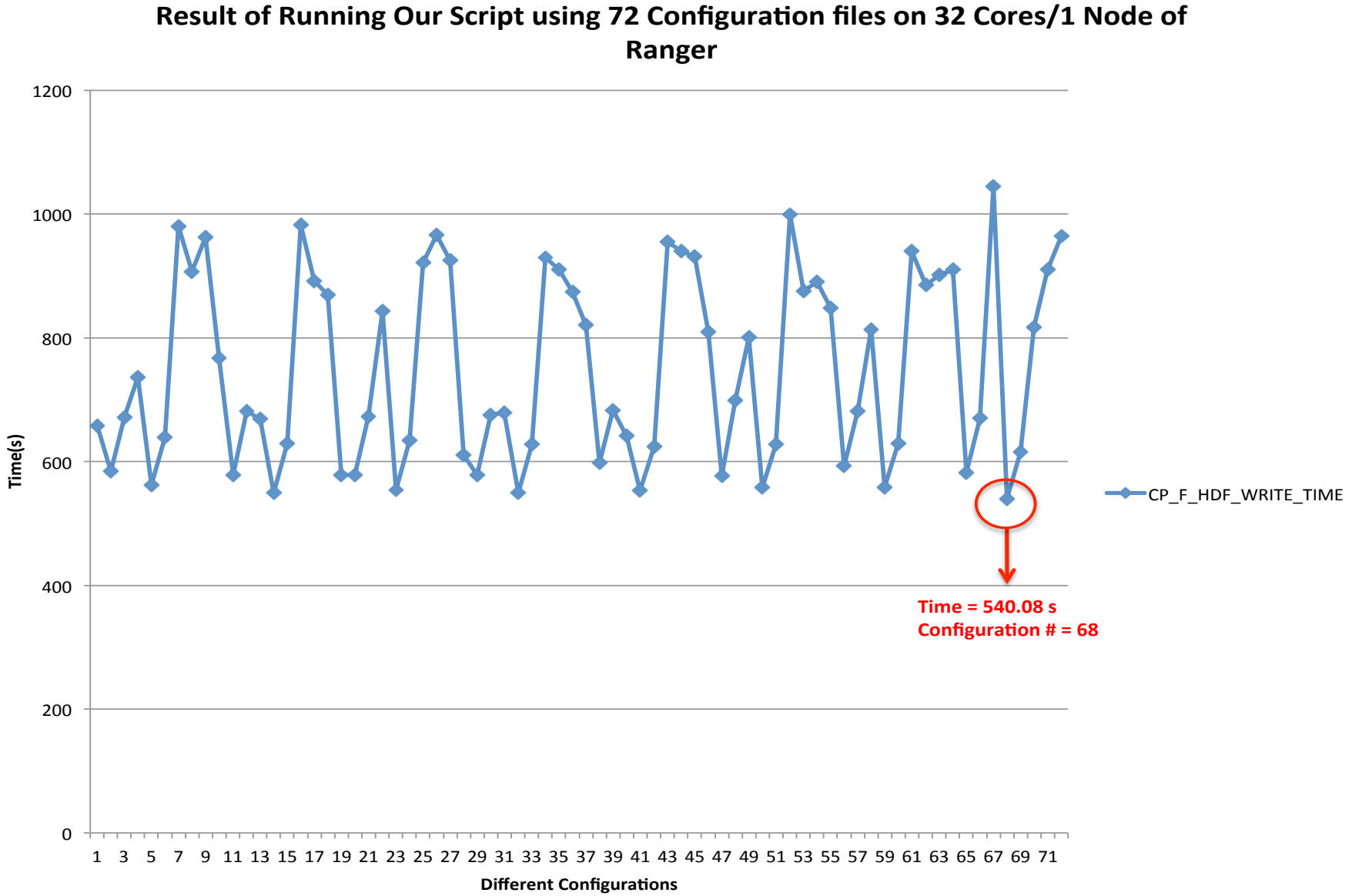
Autotuning HPC I/O

- Using the I/O Autotuning Library:
  - Dynamically link with I/O autotuner library
  - I/O autotuner library automatically reads parameters from config file created during exploration process
  - I/O autotuner automatically injects autotuning parameters as application operates

Smaller set of space visualized



Autotuning HPC I/O



Configuration #68

```
<Parameters>
        <High_Level_IO_Library>
                <sieve_buf_size> 524280 </sieve_buf_size>
                <alignment> 262144,524288 </alignment>
                <!-- H5Pset_alignment function gets 2 args: (Threshold, Alignment) -
        </High_Level_IO_Library>

        <Middleware_Layer>
                <cb_buffer_size> 134217728 </cb_buffer_size>
                <cb_nodes> 16 </cb_nodes>
        </Middleware_Layer>

        <Parallel_File_System>
                <striping_factor> 32 </striping_factor>
                <striping_unit> 8388608 </striping_unit>
        </Parallel_File_System>
</Parameters>
```

Autotuning in HDF5

- “Auto-Tuning of Parallel IO Parameters for HDF5 Applications”, Babak Behzad, et al, poster @ SC12
- "Taming Parallel I/O Complexity with Auto-Tuning”, Babak Behzad, et al, SC13

Autotuning HPC I/O

- Remaining research:
  - Determine “speed of light” for I/O on system and use that to define “good enough” performance
  - Entire space is too large to fully explore, we are now evaluating genetic algorithm techniques to help find “good enough” parameters
  - How to factor out “unlucky” exploration runs
  - Methods for avoiding overriding application parameters with autotuned parameters

Other HDF5 Improvements in Progress

- Single-Writer/Multiple-Reader (SWMR)
- Virtual Object Layer (VOL)
- Virtual Datasets

Single-Writer/Multiple-Reader (SWMR)

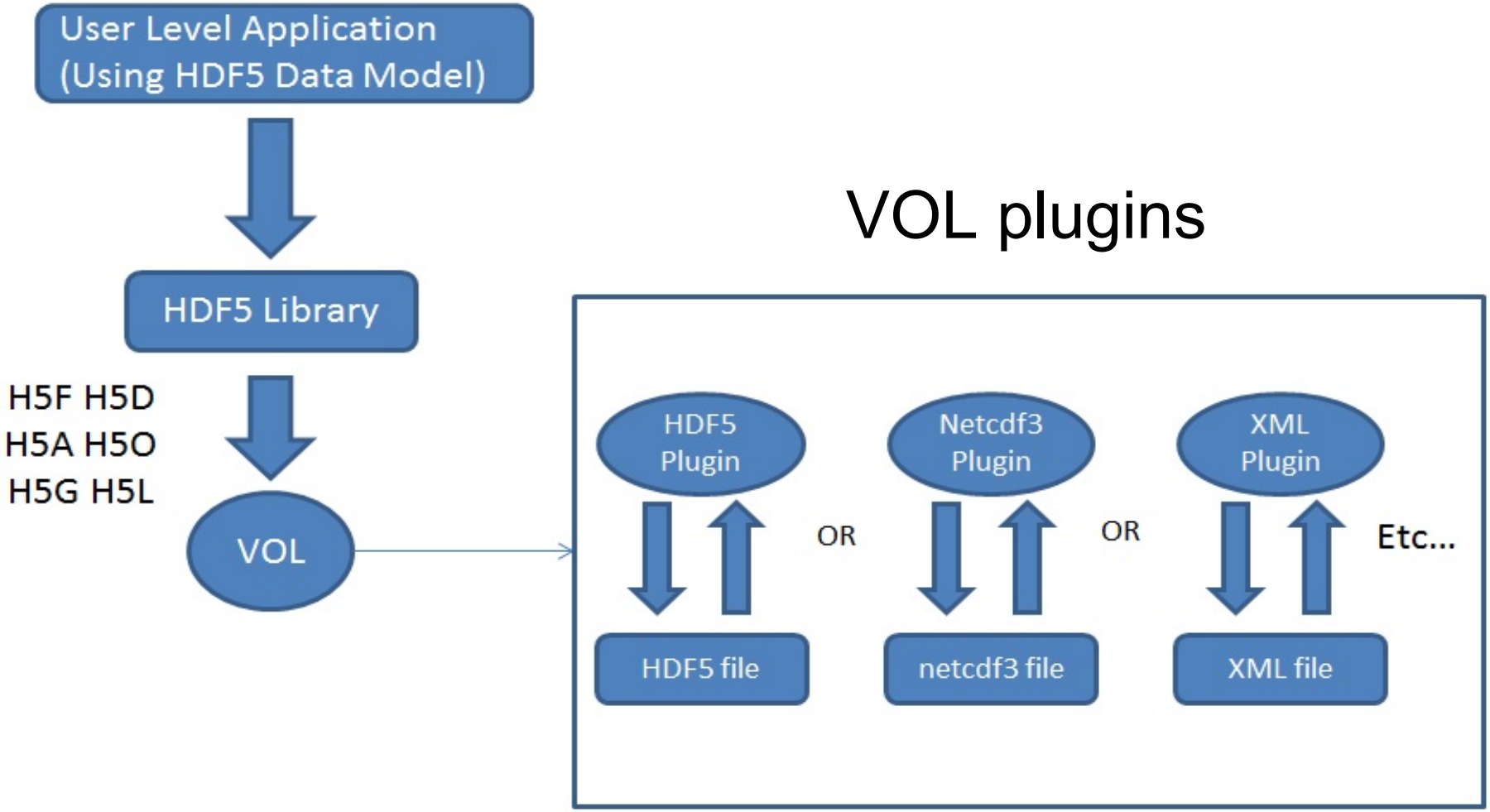
- Improves HDF5 for Data Acquisition:
  - Allows simultaneous data gathering and monitoring/analysis
  - Focused on storing data sequences for high-speed data sources
- Supports ‘Ordered Updates’ to file:
  - Crash-proofs accessing HDF5 file
  - Possibly uses small amount of extra space

VIRTUAL OBJECT LAYER (VOL)

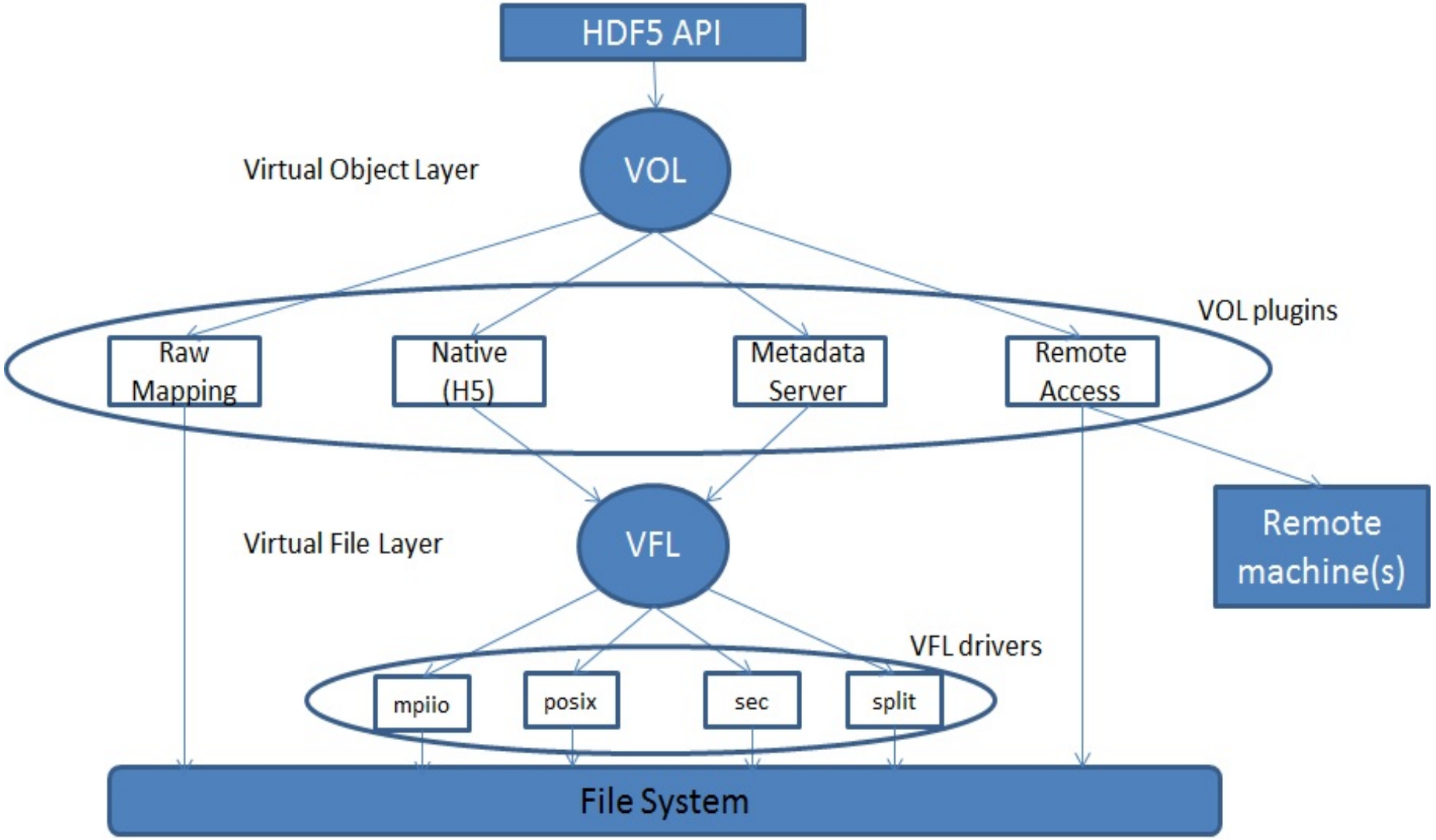
Virtual Object Layer (VOL)

- Goal
  - Provide an application with the HDF5 data model and API, but allow different underlying storage mechanisms
- New layer below HDF5 API
  - Intercepts all API calls that can touch the data on disk and routes them to a VOL plugin
- Potential VOL plugins:
  - Native HDF5 driver (writes to HDF5 file)
  - Raw driver (maps groups to file system directories and datasets to files in directories)
  - Remote driver (the file exists on a remote machine)

VOL Plugins


User Level Application
(Using HDF5 Data Model)
HDF5 Library
H5F H5D
H5A H5O
H5G H5L
VOL
VOL plugins
HDF5
Plugin
Netcdf3
Plugin
XML
Plugin
OR
OR
Etc...
HDF5 file
netcdf3 file
XML file

Virtual Object Layer



Why not use the VFL?

- VFL is implemented below the HDF5 abstract model
  - Deals with blocks of bytes in the storage container
  - Does not recognize HDF5 objects nor abstract operations on those objects
- VOL is layered right below the API layer to capture the HDF5 data model

Sample API Function Implementation

```
hid_t H5Dcreate2 (hid_t loc_id, const char *name,
hid_t type_id, hid_t space_id, hid_t lcpl_id, hid_t
dcpl_id, hid_t dapl_id) {
/* Check arguments */
      …
/* call corresponding VOL callback for H5Dcreate */
 dset_id = H5_VOL_create (TYPE_DATASET, …);
/*
 Return result to user (yes the dataset is created,
 or no here is the error)
*/
 return dset_id;
}
```

Work in progress: VOL

CONSIDERATIONS

VOL Plugin Selection

- Use a pre-defined VOL plugin:

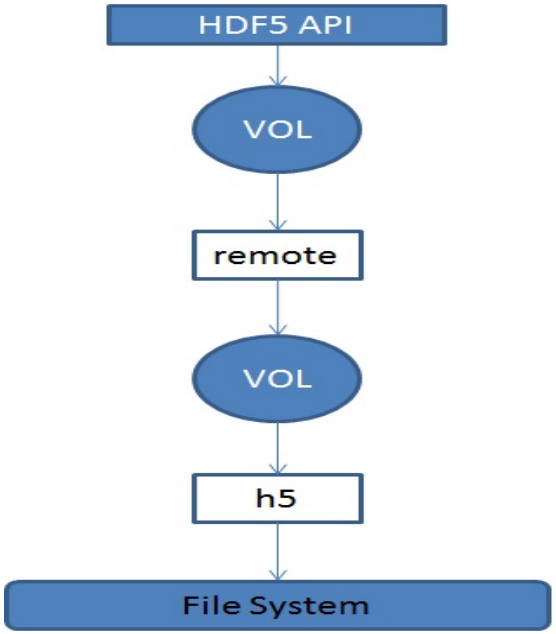
```
hid_t fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mds_vol(fapl, …);
hid_t file = H5Fcreate("foo.h5", …, …, fapl);
H5Pclose(fapl);
```

- Register user defined VOL plugin:

```
H5VOLregister (H5VOL_class_t *cls)
H5VOLunregister (hid_t driver_id)
H5Pget_plugin_info (hid_t plist_id)
```

Interchanging and Stacking Plugins

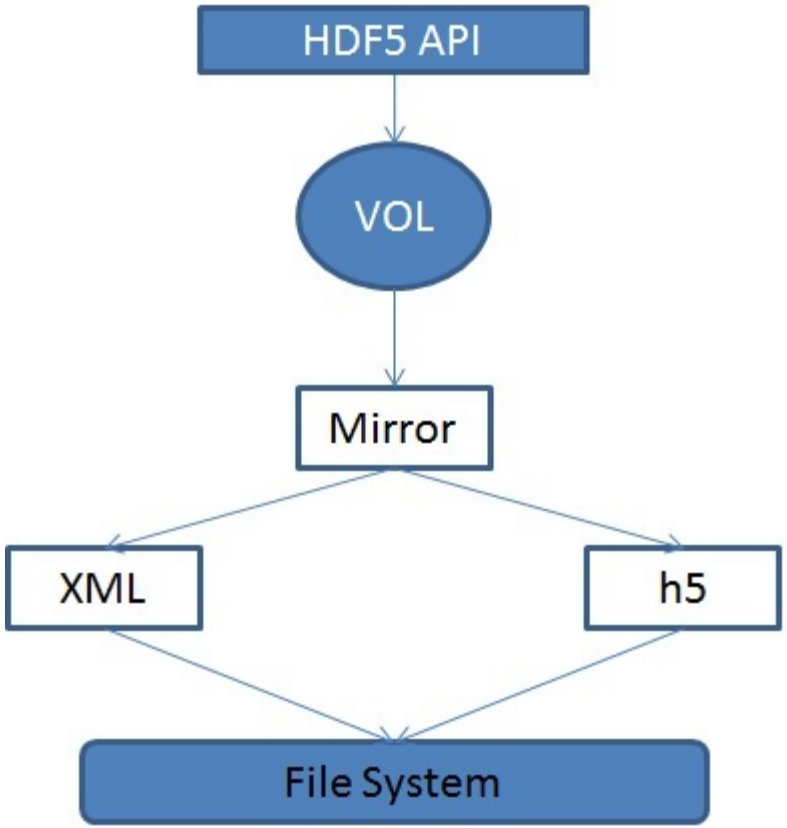
- Interchanging VOL plugins
  - Should be a valid thing to do
  - User's responsibility to ensure plugins coexist
- Stacking plugins



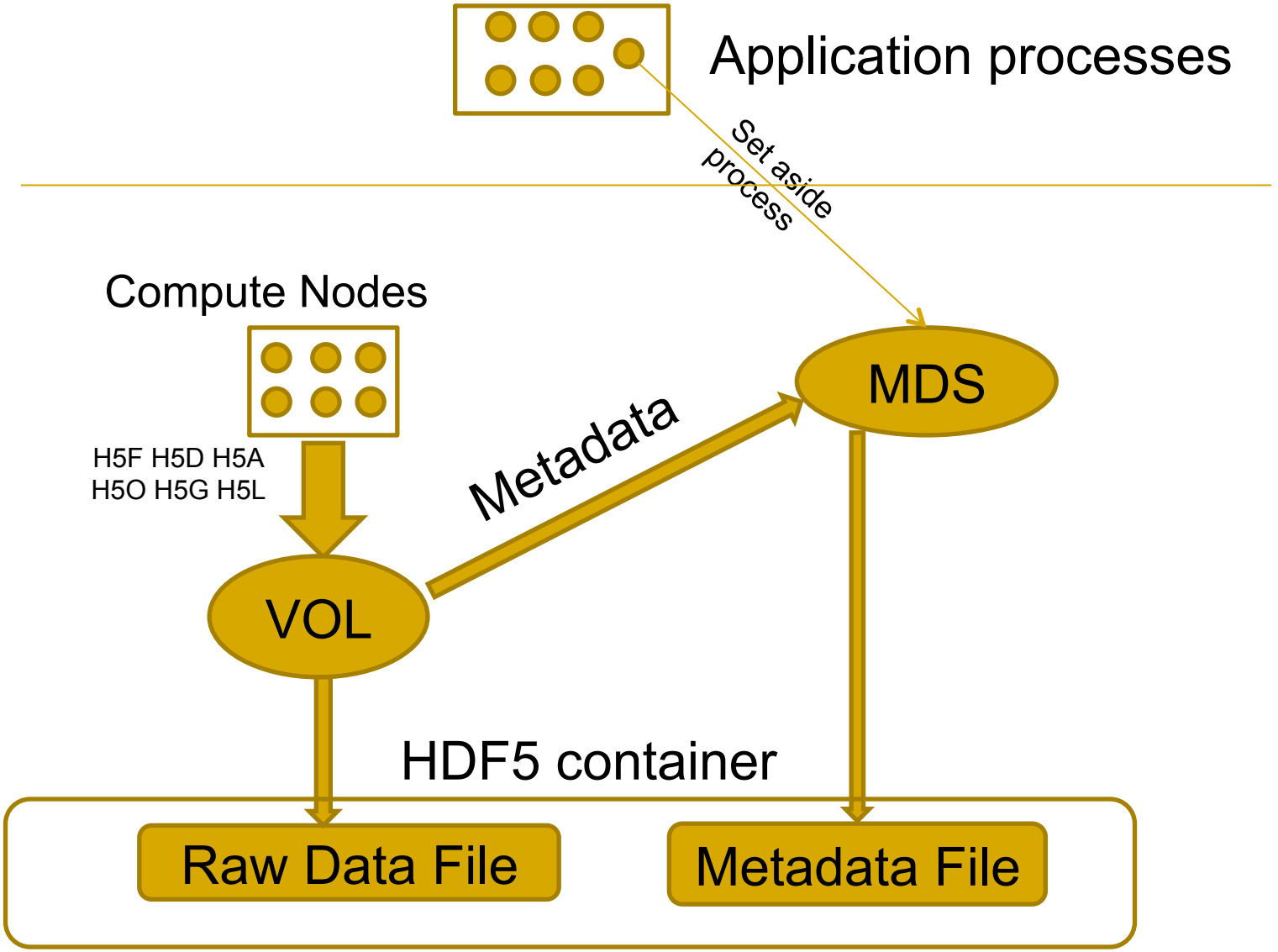
- Stacking should make sense.
- For example, the first VOL plugin in a stack could be a statistics plugin, that does nothing but gather information on what API calls are made and their corresponding parameters.

Mirroring

- Extension to stacking
- HDF5 API calls are forwarded through a mirror plugin to two or more VOL plugins



Sample Plugins: Metadata Server



Raw Plugin

- The flexibility of the virtual object layer provides developers with the option to abandon the single file, binary format like the native HDF5 implementation.
- A “raw” file format could map HDF5 objects (groups, datasets, etc …) to file system objects (directories, files, etc …).
- The entire set of raw file system objects created would represent one HDF5 container.

Remote Plugin

- A remote VOL plugin would allow access to files located on a server.
- Prototyping two implementations:
  - Web-services via RESTful access:

    http://www.hdfgroup.org/projects/hdfserver/
  - Native HDF5 file access over sockets:

    http://svn.hdfgroup.uiuc.edu/h5netvol/trunk/

Implementation

- VOL Class
  - Data structure containing general variables and a collection of function pointers for HDF5 API calls
- Function Callbacks
  - API routines that potentially touch data on disk
  - H5F, H5D, H5A, H5O, H5G, H5L, and H5T

Implementation

- We will end up with a large set of function callbacks:
  - Lump all the functions together into one data structure OR
  - Have a general class that contains all common functions, and then children of that class that contain functions specific to certain HDF5 objects OR
  - For each object have a set of callbacks that are specific to that object (This is design choice that has been taken).

Filters

- Need to keep HDF5 filters in mind
- Where is the filter applied, before or after the VOL plugin?
  - Logical guess now would be before, to avoid having all plugins deal with filters

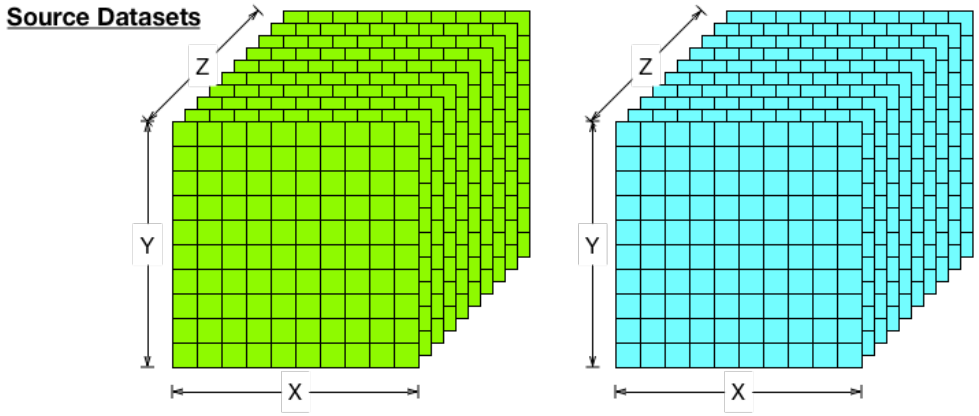
Current status of VOL

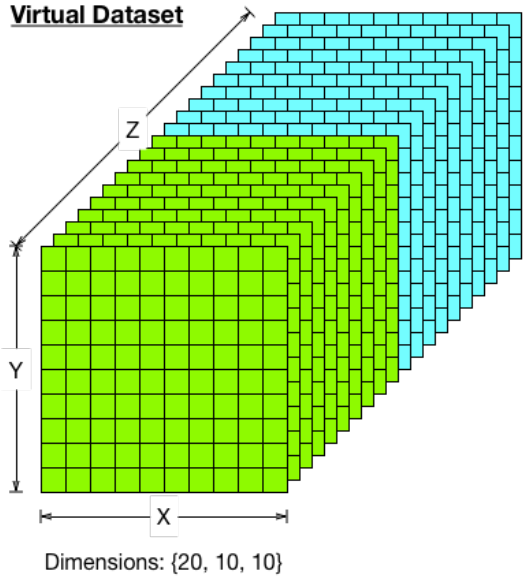
- ?

Virtual Datasets

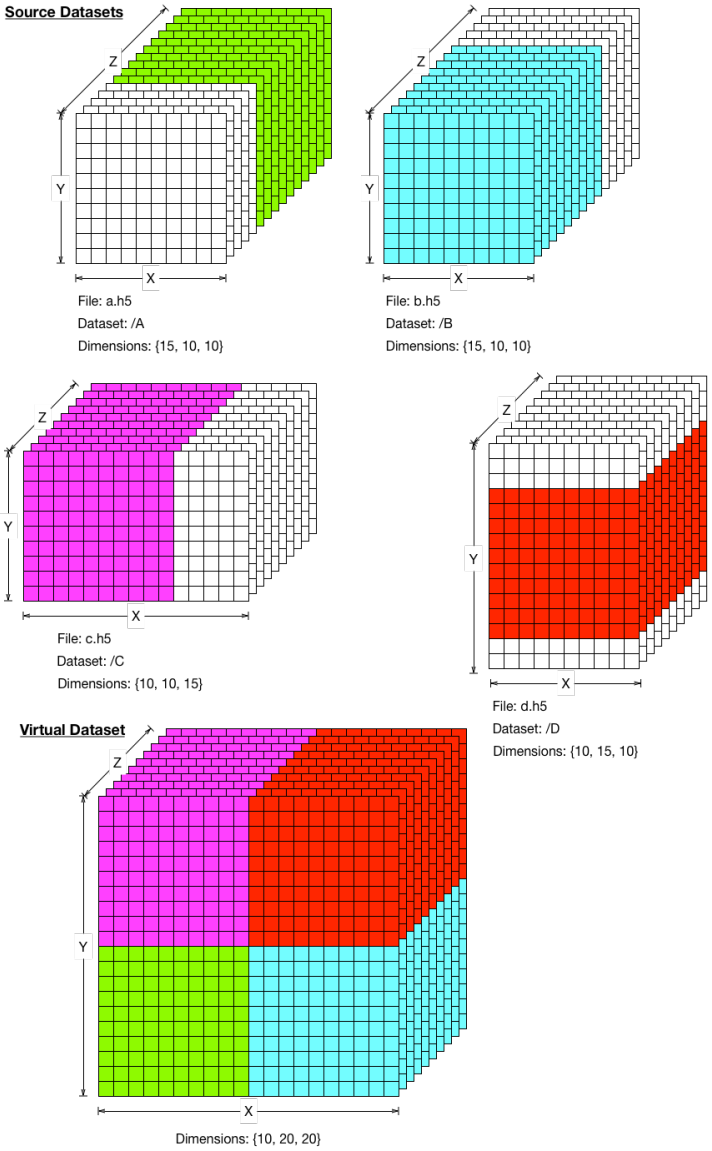
- Mechanism for creating a composition of multiple *source* datasets, while accessing through single *virtual* dataset
- Modifications to source datasets are visible to virtual dataset
  - And writing to virtual dataset modifies source datasets
- Can have subset within source dataset mapped to subsets within virtual dataset
- Source and virtual datasets can have unlimited dimensions
- Source datasets can be virtual datasets themselves

Virtual Datasets, Example 1

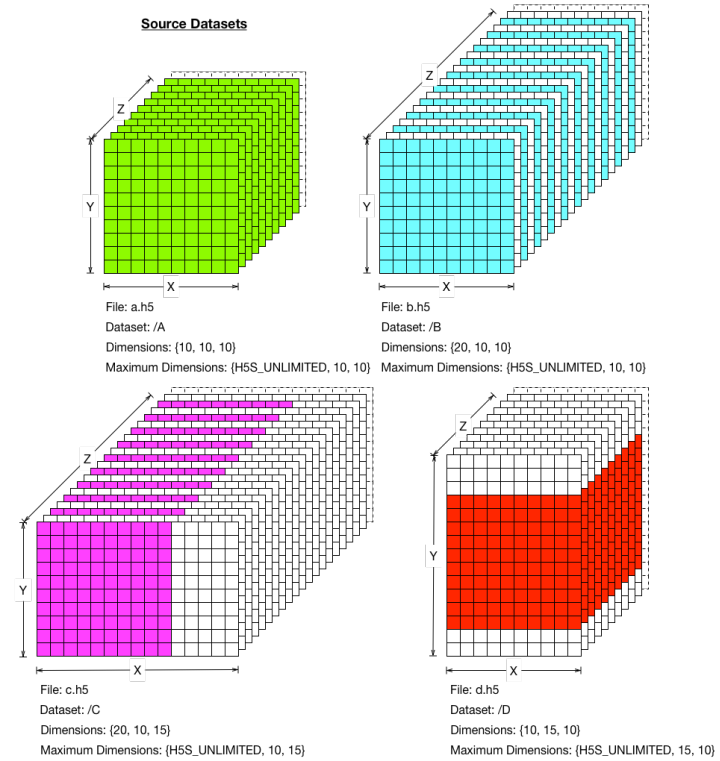


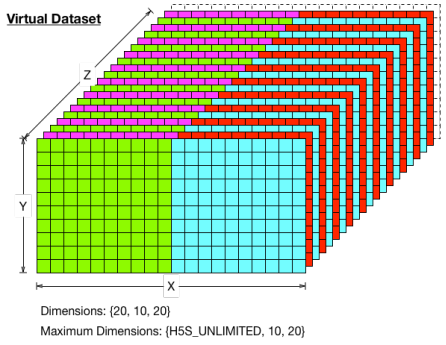


Virtual Datasets, Example 2



Virtual Datasets, Example 3





HDF5 Roadmap

- Concurrency
  - Single-Writer/Multiple-Reader (SWMR)
  - Internal threading
- Virtual Datasets
- Virtual Object Layer (VOL)
- Data Analysis
  - Query / View / Index APIs
- Native HDF5 client/server

"The best way to predict the future is to invent it."

– Alan Kay

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

Thank You!

Questions?

Codename "HEXAD"

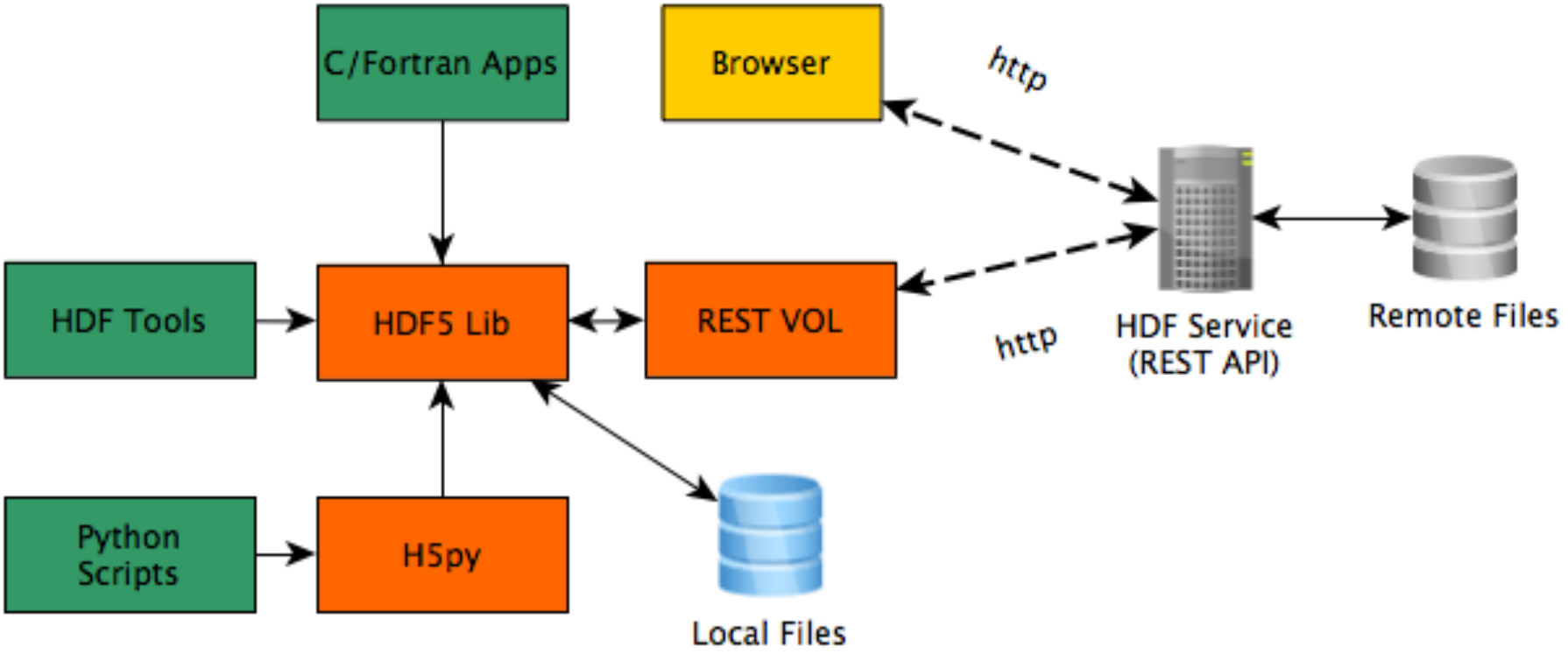
- Excel is a great frontend with a not so great rear ;-)
- We've fixed that with an **H**DF5 **Ex**cel **Ad**d-in
- Let's you do the usual things including:
  - Display content (file structure, detailed object info)
  - Create/read/write datasets
  - Create/read/update attributes
- Plenty of ideas for bells an whistles, e.g., HDF5 image & PyTables support
- Send in* your **Must Have**/*Nice To Have* feature list!
- Stay tuned for the beta program

* help@hdfgroup.org

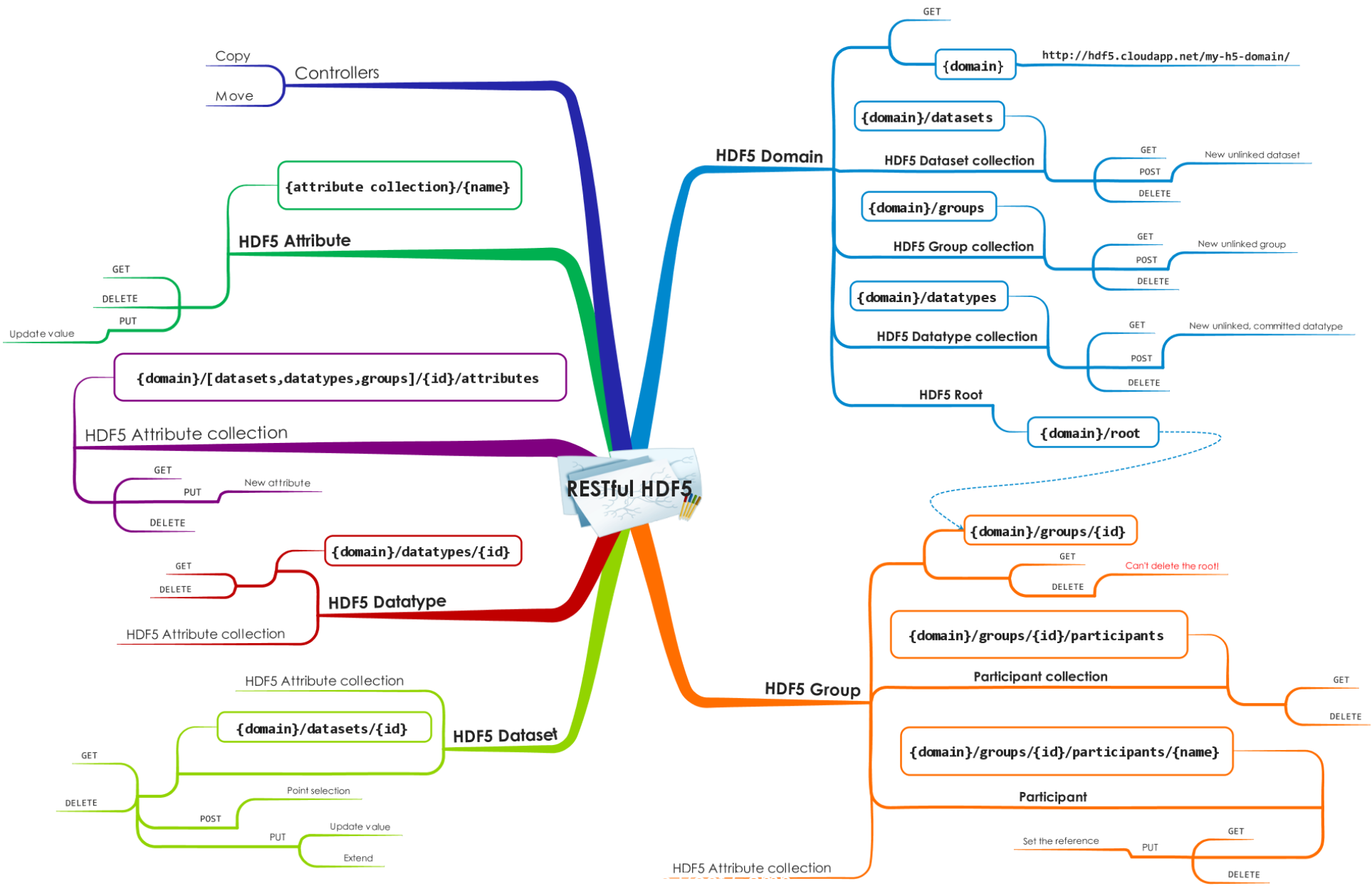
HDF Server

- REST-based service for HDF5 data
- Reference Implementation for REST API
- Developed in Python using Tornado Framework
- Supports Read/Write operations
- Clients can be Python/C/Fortran or Web Page
- Let us know what specific features you'd like to see. E.g. VOL REST Client Plugin

HDF Server Architecture



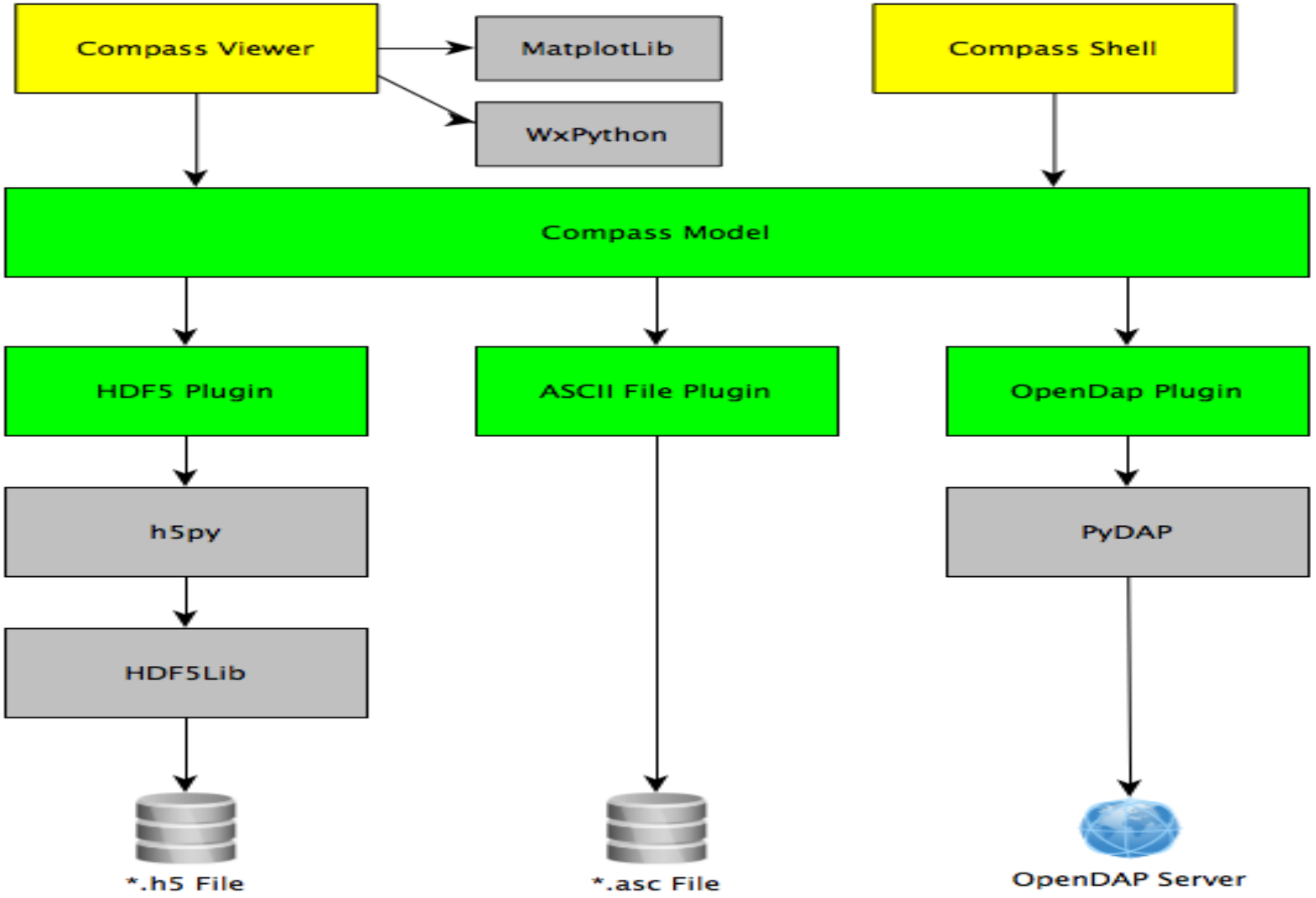
Restless About HDF5/REST



HDF Compass

- “Simple” HDF5 Viewer application
- Cross platform (Windows/Mac/Linux)
- Native look and feel
- Can display extremely large HDF5 files
- View HDF5 files and OpenDAP resources
- Plugin model enables different file formats/ remote resources to be supported
- Community-based development model

Compass Architecture



The Poisson Problem

- The Poisson Problem is a simple PDE

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

given $u(x, y) = g(x, y)$ on the boundary

- Solve the Poisson equation numerically over a region by discretizing it in the *x* and *y* directions to obtain a grid of points

- Compute the approximate solution values at these points

http://bit.ly/1FzDeEZ

The Poisson Problem

- We may replace the partial derivatives by numerical approximations involving first-order finite difference

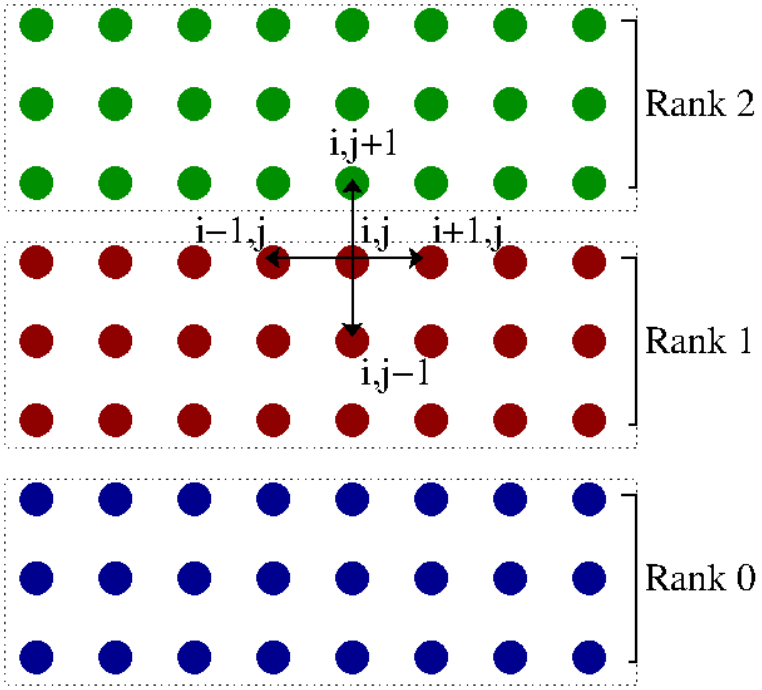
$$u(i-1,j)+u(i,j+1)+u(i+1,j)-4u(i,j)=f(i,j)$$

- We can write a Jacobi iteration as

$$u^{k+1}(i,j)=1/4[u^k(i-1,j)+u^k(i,j+1)+u^k(i,j-1)+u^k(i+1,j)-h^2f(i,j)]$$

http://bit.ly/1FzDeEZ

One-dimension Decomposition of Domain



Algorithm

… Figure out communication and decomposition of the domain

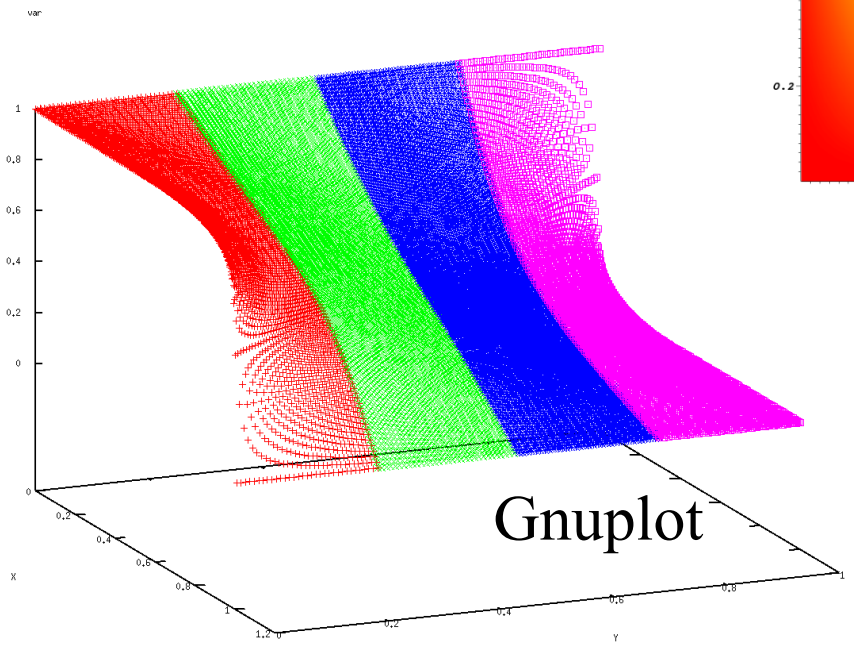
```
DO WHILE ( diffnorm > tolerance )
 …..Actually do the computation
END DO
```

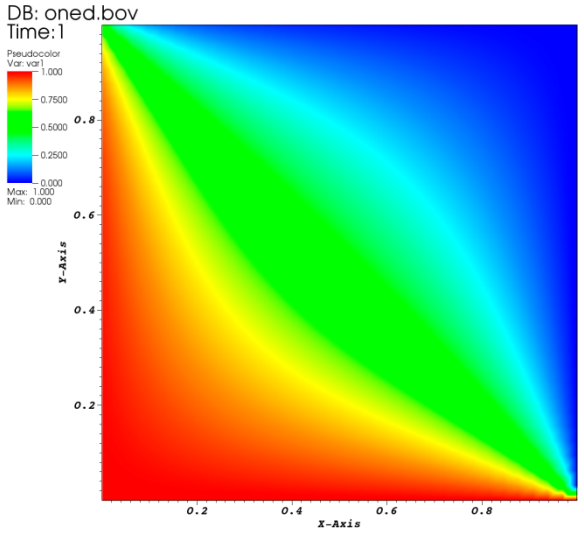
… After our ground breaking solution we need to save the solution

http://bit.ly/1FzDeEZ

Method I: One File per Process

```
OPEN(10,file='oned'//ichr4//'.dat', ACCESS='STREAM')

dx = 1.0_dp/REAL(nx, KIND=dp)
x = 0.0_dp
DO i = 0, nx+1
   y = (sy-1)*dx
   DO j = sy, ey+1
      WRITE(10) b(i,j)
      y = y + dx
   ENDDO
   x = x + dx
ENDDO
close(10)
```





Method II: Write to a Single HDF File

```
SUBROUTINE WRITE_HDF(b, nx, sy, ey, my_id, nprocs, MPI_COMM_WORLD)

  INTEGER :: nx, sy, ey, my_id, nprocs
  REAL (KIND=dp), DIMENSION (0:nx+1,sy-1:ey+1), TARGET :: b

  CHARACTER(LEN=7), PARAMETER :: filename = "oned.h5"  ! File name
  CHARACTER(LEN=3), PARAMETER :: dsetname = "Var" ! Dataset name

  INTEGER(HID_T) :: file_id       ! File identifier
  INTEGER(HID_T) :: dset_id       ! Dataset identifier
  INTEGER(HID_T) :: filespace     ! Dataspace identifier in file
  INTEGER(HID_T) :: memspace      ! Dataspace identifier in memory
  INTEGER(HID_T) :: plist_id      ! Property list identifier

  INTEGER(HSIZE_T), DIMENSION(2) :: dimsf ! Dataset dimensions.

   INTEGER(HSIZE_T), DIMENSION(2) :: count
   INTEGER(HSSIZE_T), DIMENSION(2) :: offset
   INTEGER :: rank = 2 ! Dataset rank
  TYPE(c_ptr) :: f_ptr
   INTEGER :: error, error_n  ! Error flags
   INTEGER :: MPI_COMM_WORLD
```

Method II: Write to a Single HDF File

dimsf = (/nx+2,nx+2/)

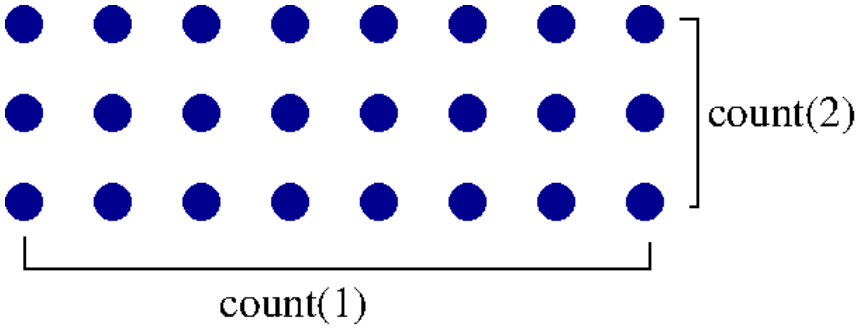
```
! Initialize FORTRAN predefined datatypes
CALL h5open_f(error)

! Setup file access property list with parallel I/O access.
CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, error)
CALL h5pset_fapl_mpio_f(plist_id, MPI_COMM_WORLD, MPI_INFO_NULL, error)
! Create the file collectively.
CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, error, access_prp = plist_id)
CALL h5pclose_f(plist_id, error)
! Create the data space for the  dataset.
CALL h5screate_simple_f(rank, dimsf, filespace, error)
! Create the dataset with default properties.
CALL h5dcreate_f(file_id, dsetname, H5T_NATIVE_DOUBLE, filespace, dset_id, error)
CALL h5sclose_f(filespace, error)
```

Method II: Write to a Single HDF File

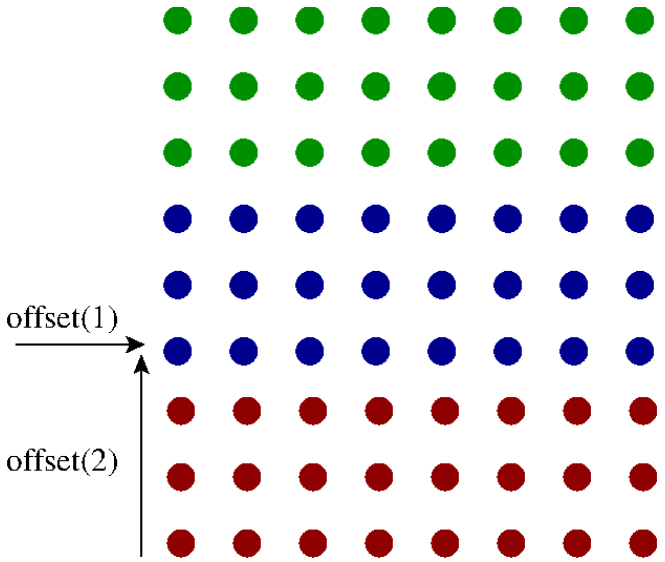
! Each process defines dataset in memory and writes it to the hyperslab
! in the file.

COUNT(1) = nx+2
COUNT(2) = ey-sy + 1
offset(1) = 0
offset(2) = my_id * COUNT(2)
CALL h5screate_simple_f(rank, count, memspace, error)

Proc 1 (memory space)



File



Method II: Write to a Single HDF File

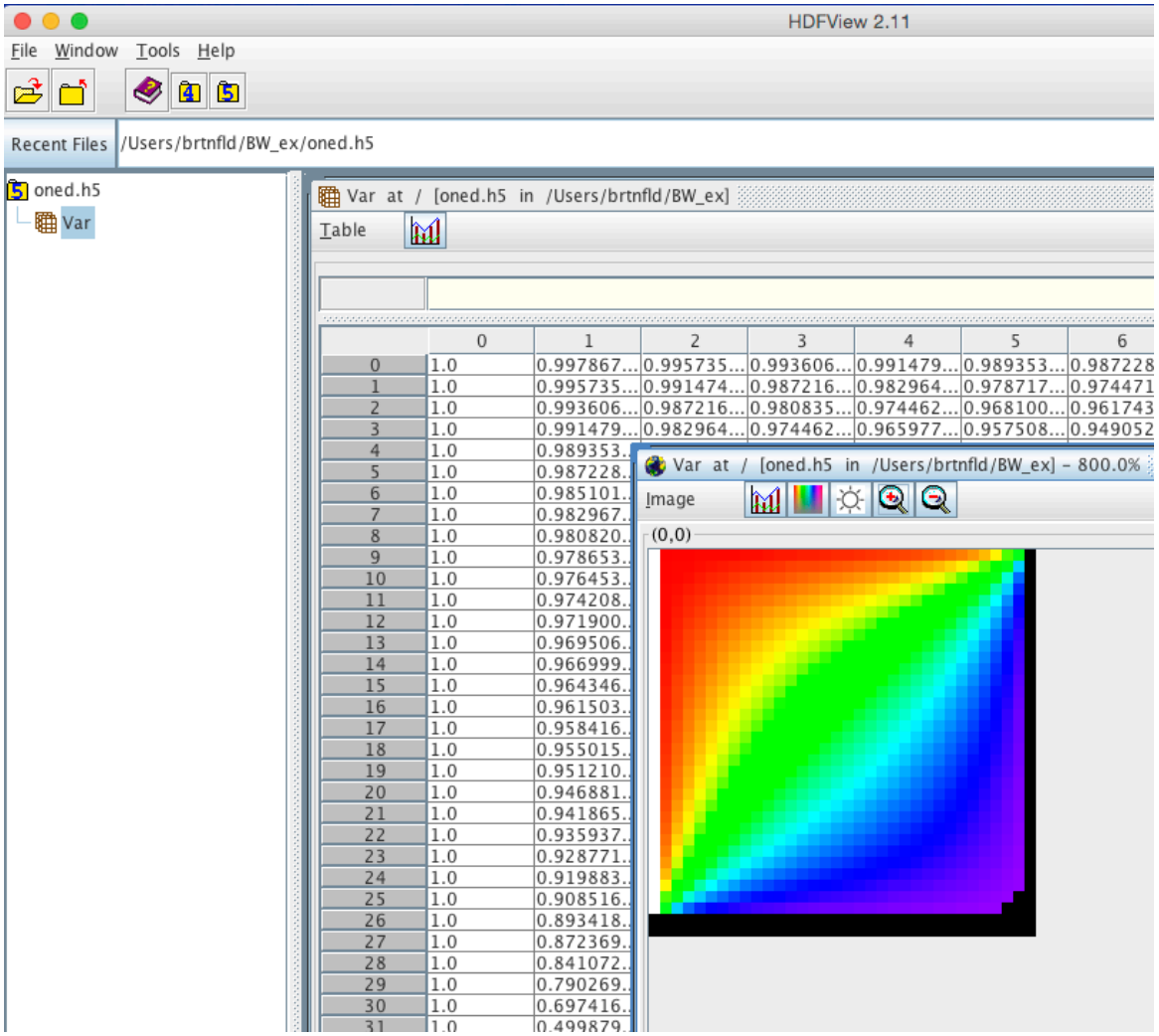
```
! Select hyperslab in the file.
CALL h5dget_space_f(dset_id, filespace, error)
CALL h5sselect_hyperslab_f (filespace, H5S_SELECT_SET_F, offset, &
count, error)

! Create property list for collective dataset write
CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, error)
CALL h5pset_dxpl_mpio_f(plist_id, H5FD_MPIO_COLLECTIVE_F, error)

! Write the dataset collectively.
f_ptr = C_LOC(b(0,sy))
CALL h5dwrite_f(dset_id, H5T_NATIVE_DOUBLE, f_ptr, error, &
    file_space_id = filespace, mem_space_id = memspace, &
    xfer_prp = plist_id)
! Write the dataset independently.
!   CALL h5dwrite_f(dset_id, H5T_NATIVE_DOUBLE, f_ptr, error, &
!                   file_space_id = filespace, mem_space_id = memspace)
```

Method II: Write to a Single HDF File

```
! Close dataspaces.
CALL h5sclose_f(filespace, error)
CALL h5sclose_f(memspace, error)

! Close the dataset and property list.
CALL h5dclose_f(dset_id, error)
CALL h5pclose_f(plist_id, error)

! Close the file.
CALL h5fclose_f(file_id, error)

! Close FORTRAN predefined datatypes.
CALL h5close_f(error)
```

Method II: Write to a Single HDF File

- All the processes create one HDF5 file
- View data with HDFView



Don't Reinvent the Wheel…

- Use libraries that utilize HDF, but represent the scientific data using a set of conventions, i.e.
  - How to represent meshes
  - Variable definitions
  - Multiple datasets
  - Component definitions
- Some parallel formats using HDF
  - CFD: CGNS
  - Meshless Methods: H5Part
  - FEM: MOAB
  - General: NetCDF
- Hides the complexity of HDF

CGNS example

```
/* ================================== */
 /* ==    **WRITE THE CGNS FILE **          == */
 /* ================================== */

cgp_open(fname, CG_MODE_WRITE, &fn);

cg_base_write(fn, "Base 1", cell_dim, phys_dim, &B);
cg_zone_write(fn, B, "Zone 1", nijk, Unstructured, &Z);
```

CGNS example

```
/* ======================================== */
/* == (A) WRITE THE NODAL COORDINATES    == */
 /* ======================================== */

 count = nijk[0]/comm_size;
 min = count*comm_rank+1;
 max = count*(comm_rank+1);

 cgp_coord_write(fn,B,Z,CGNS_ENUMV(RealDouble),"CoordinateX",&Cx);
 cgp_coord_write(fn,B,Z,CGNS_ENUMV(RealDouble),"CoordinateY",&Cy);
 cgp_coord_write(fn,B,Z,CGNS_ENUMV(RealDouble),"CoordinateZ",&Cz);
 Cvec[0] = Cx;
 Cvec[1] = Cy;
 Cvec[2] = Cz;
 cgp_coord_multi_write_data(fn, B, Z, Cvec, &min,&max, Coor_x, Coor_y, Coor_z);
```

CGNS example

```
/* ================================== */
/* == (B) WRITE THE CONNECTIVITY TABLE  == */
/* ================================== */

start = 1;
end = nijk[1];

cgp_section_write(fn,B,Z,"Elements",PENTA_6,start,end,0,&S);
count = nijk[1]/comm_size;
emin = count*comm_rank+1;
emax = count*(comm_rank+1);
cgp_elements_write_data(fn, B, Z, S, emin, emax, elements);
```

CGNS example

```
/* ========================================== */
/* == (C) WRITE THE FIELD DATA                == */
/* ========================================== */
count = nijk[0]/comm_size;

cg_sol_write(fn, B, Z, "Solution", Vertex, &S);

cgp_field_write(fn,B,Z,S,CGNS_ENUMV(RealDouble),"MomentumX",&Fx) ;
cgp_field_write(fn,B,Z,S,CGNS_ENUMV(RealDouble),"MomentumY",&Fy);
cgp_field_write(fn,B,Z,S,CGNS_ENUMV(RealDouble),"MomentumZ",&Fz);
Fvec[0] = Fx;
Fvec[1] = Fy;
Fvec[2] = Fz;

cgp_field_multi_write_data(fn,B,Z,S,Fvec,&min,&max,
                           3,Data_Fx,Data_Fy,Data_Fz);
```

Reading the file is just as easy…