ARGONNE ATPESC2025
EXTREME-SCALE COMPUTING

Beyond the Data Swamp – Finding Order with HDF5

M. Scot Breitenfeld

The HDF Group

Why?

You're a scientist with massive, complex data.

- Multi-dimensional arrays (particle trajectories)
- Images
- Metadata (experiment parameters, timestamps)

- **Storing it in separate files is a mess.**
  - trajectories_run1.csv … trajectories_runn^{th}.csv
  - metadata_run1.txt … metadata_runn^{th}.txt
  - images_run1_001.png… images_runn^{th}_001.png

- **Accessing data is slow and complicated.**
  - To find images from a specific run, you have to open and parse multiple files

Talk Outline

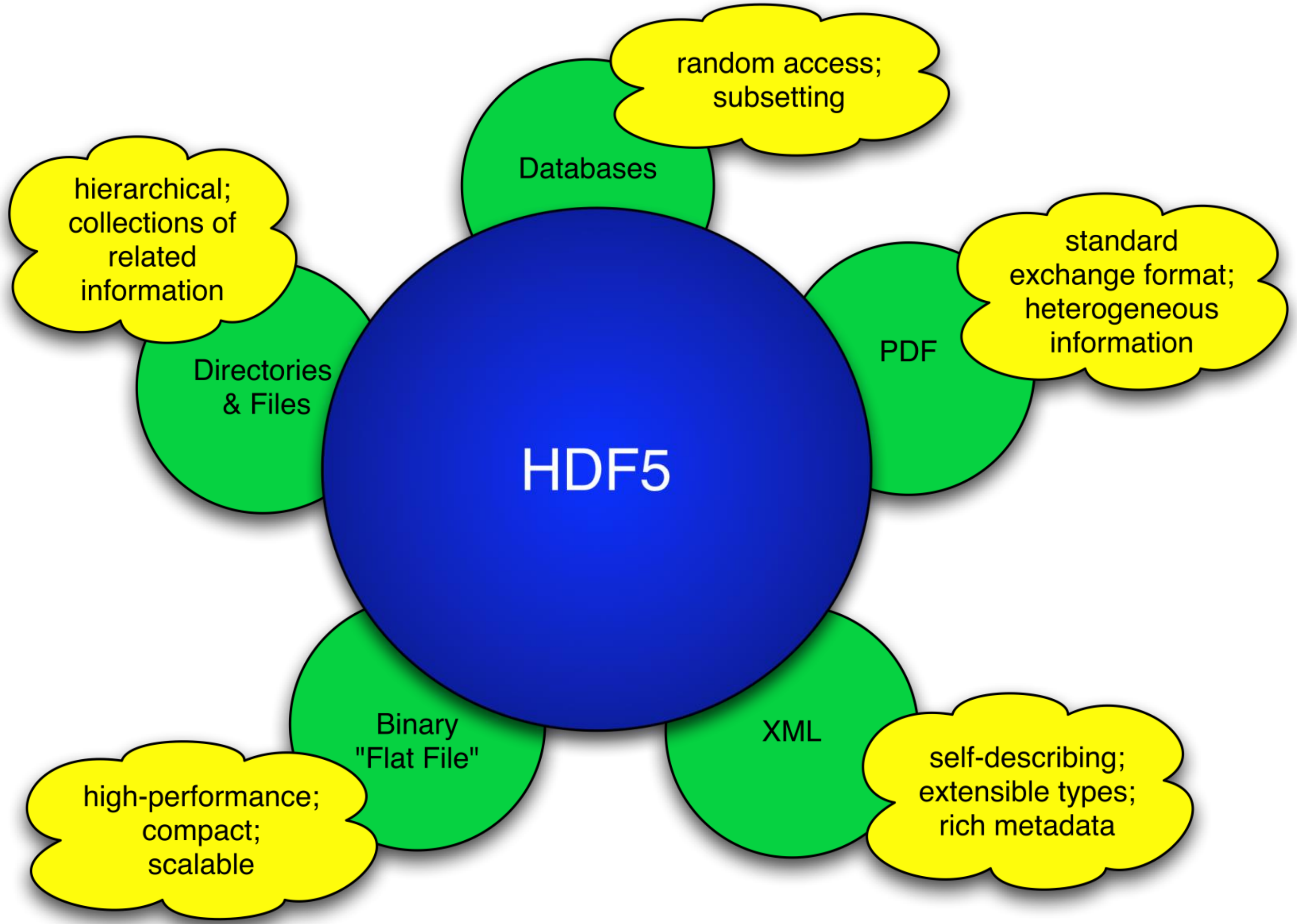
- Foundations of HDF5
  - Introduction to
    - HDF5 data model, software, and architecture
    - HDF5 programming model
  - Overview of general best practices

- Overview of parallel HDF5
  - Introduction to HDF5 parallel I/O
  - New features

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 stores : Object Stores (DAOS, AWS S3, AZURE, Ceph, etc.), memory hierarchies and other storage devices
  3. Open **file format** (POSIX storage only)

HDF5 is like …

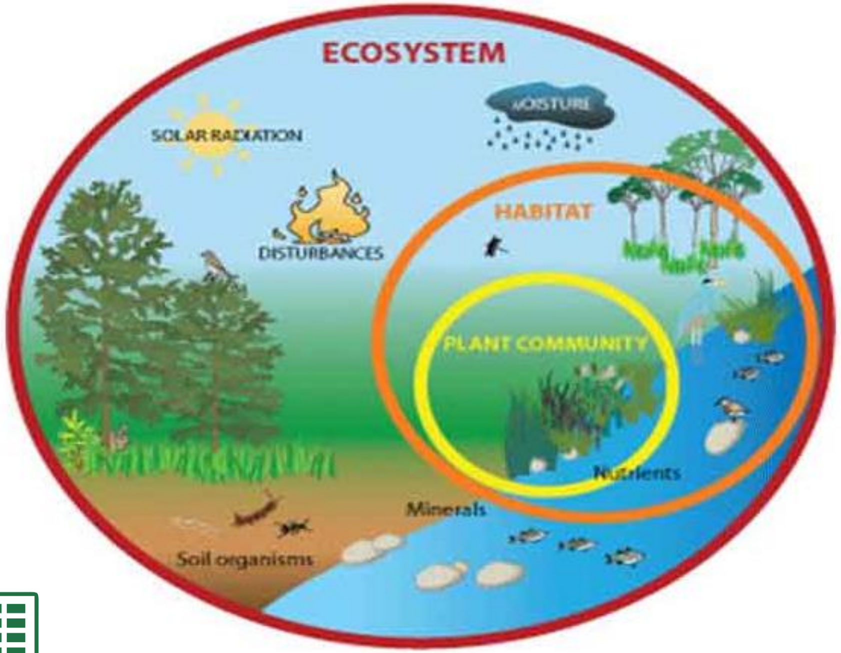


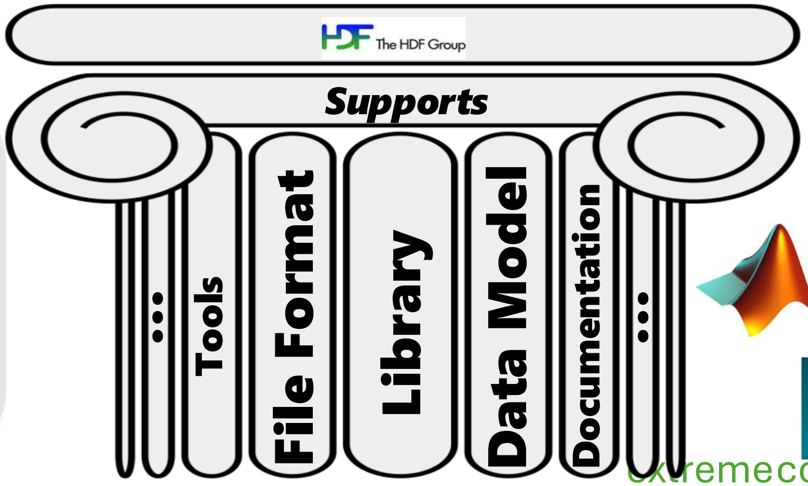
random access;
subsetting
Databases
hierarchical;
collections of
related
information
Directories
& Files
HDF5
PDF
standard
exchange format;
heterogeneous
information
Binary
"Flat File"
XML
high-performance;
compact;
scalable
self-describing;
extensible types;
rich metadata

HDF5 is designed for…

- High-volume and complex data
  - HDF5 files of GB+ sizes are common
- Every size and type of system (portable)
  - Works on embedded systems ➪desktops/laptops ➪ 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 as a Transition Layer

Concepts

Variables

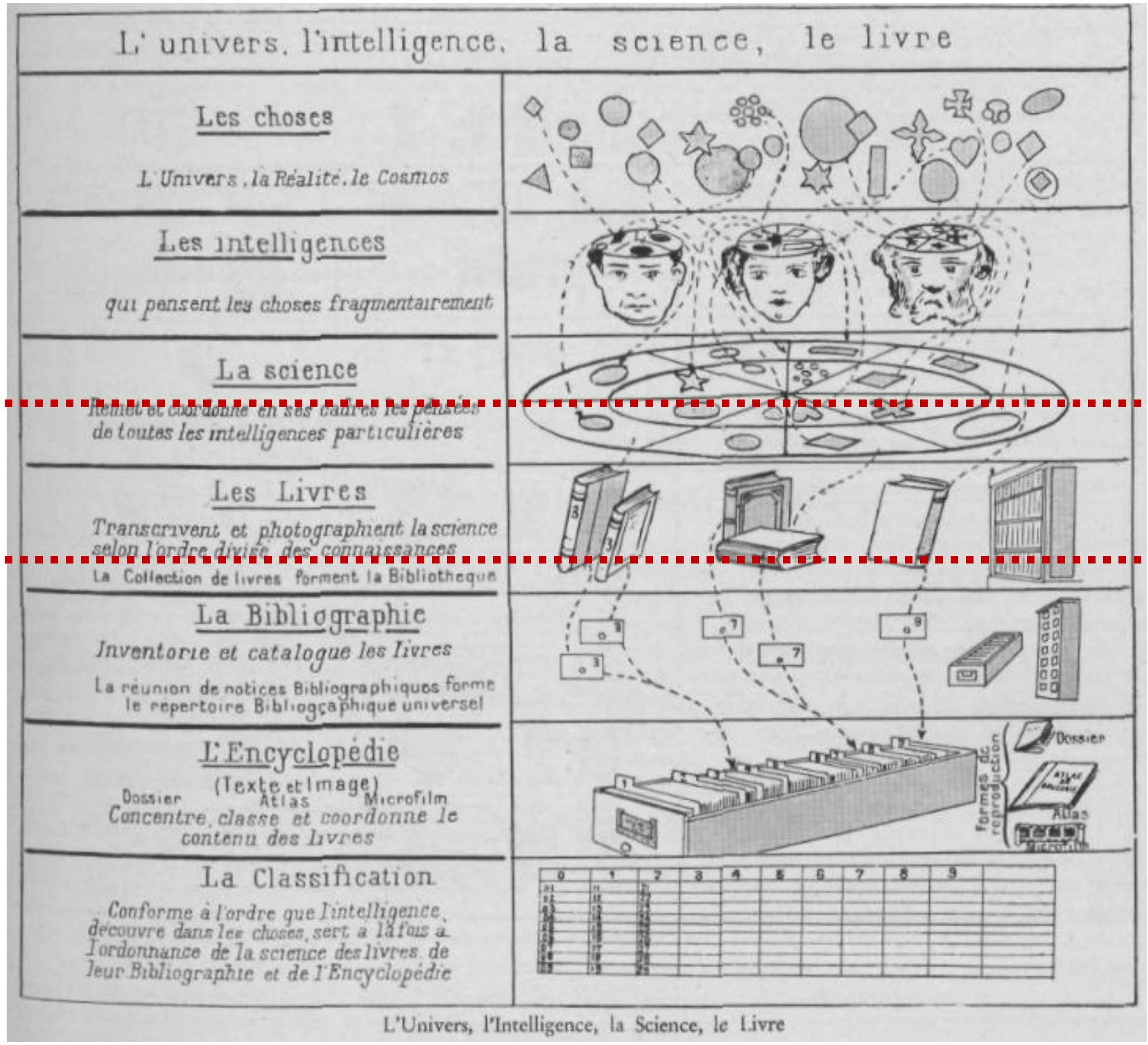
Symbols

HDF5

Signs

Representations

Encodings

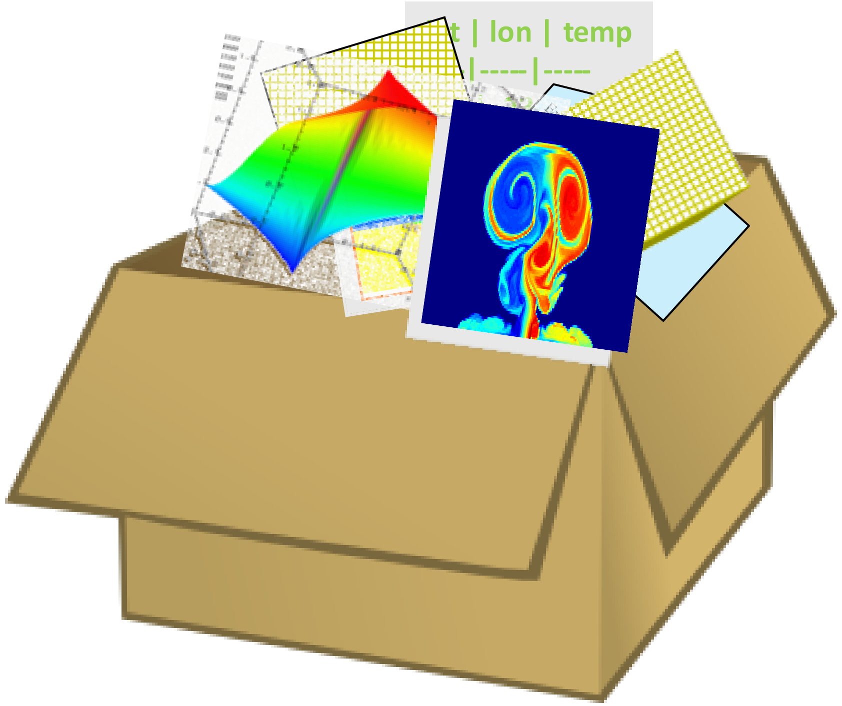


Contexts

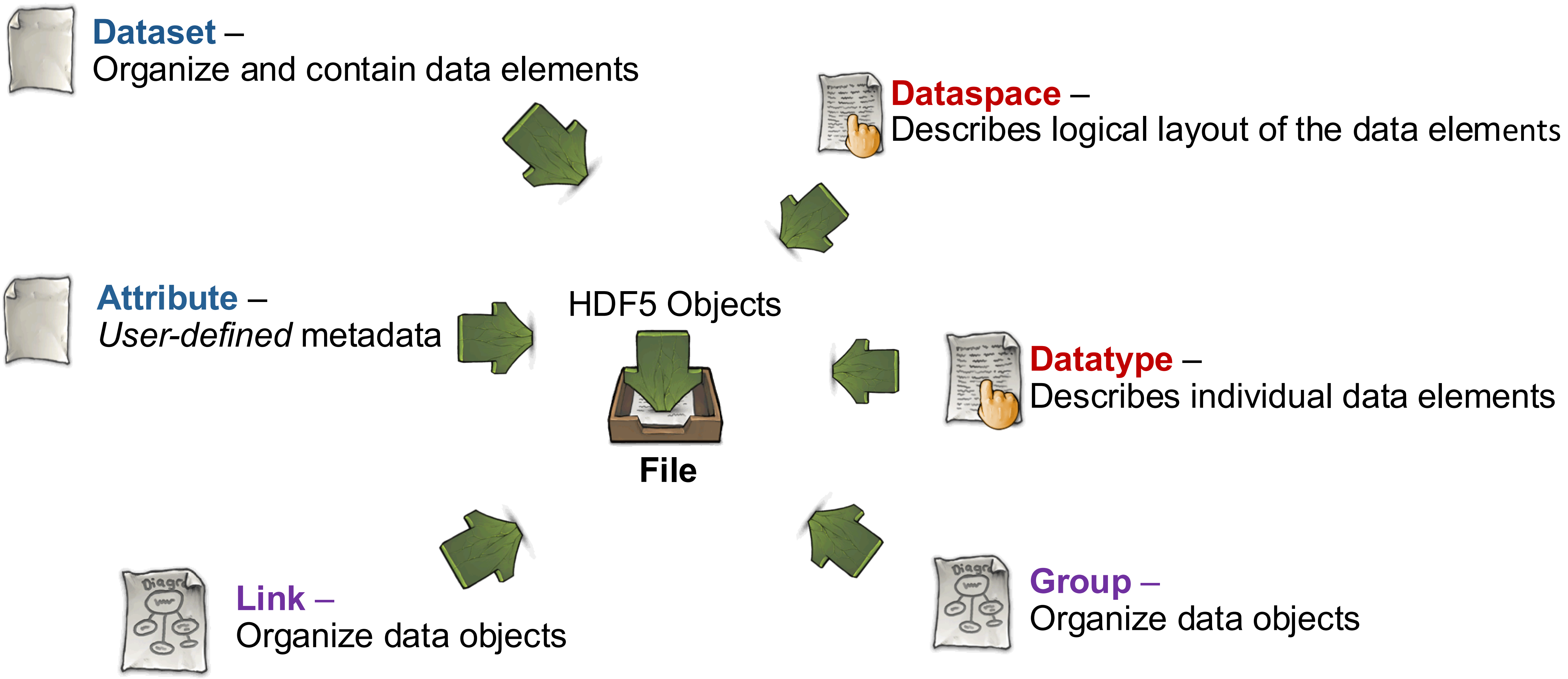
Self-describing Data

HDF5 File

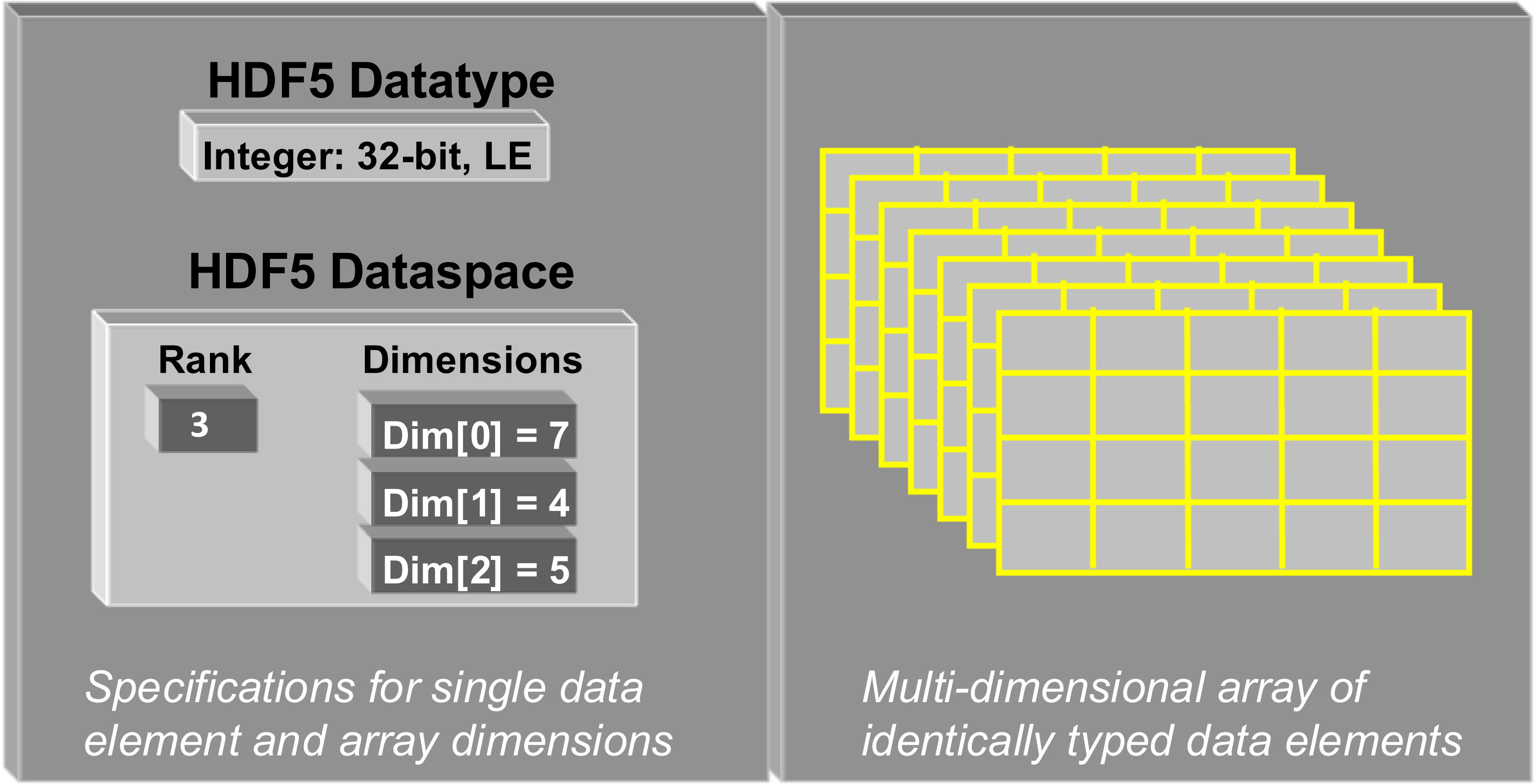
An HDF5 file is a **container** that holds data objects.



HDF5 Data Model



HDF5 Dataset



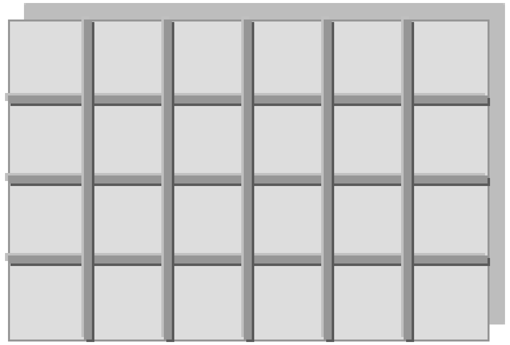
- HDF5 datasets **organize and contain** data elements
  - HDF5 datatype describes individual data elements
  - HDF5 dataspace describes the logical layout of the data elements

HDF5 Dataspace

Two roles:

<u>(1) Spatial information for Datasets and Attributes</u>

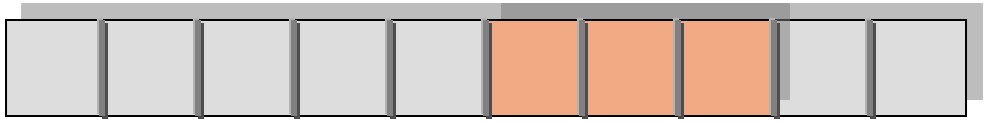
- Empty sets and scalar values
- Multidimensional arrays
    - Rank and dimensions
- A permanent part of object definition



Rank = 2

Dimensions = 4 x 6

<u>(2) Partial I/O</u>: Dataspace and subset describe the application’s data buffer and data elements participating in I/O



Rank = 1

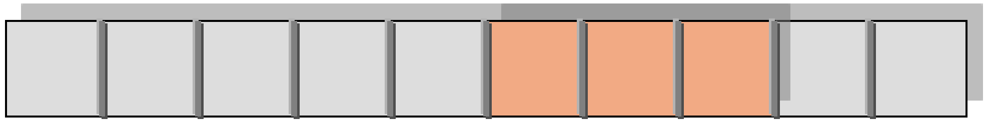
Dimension = 10

How to describe a subset in HDF5?

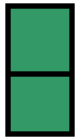
- Before writing and reading a subset of data, one must describe it to the HDF5 Library.
- The HDF5 APIs and documentation refer to a subset as a "***selection***," for example "*hyperslab* selection."
- If specified, HDF5 performs I/O on a selection *only* and <u>not</u> on all dataset elements.

Describing elements for I/O: HDF5 Hyperslab

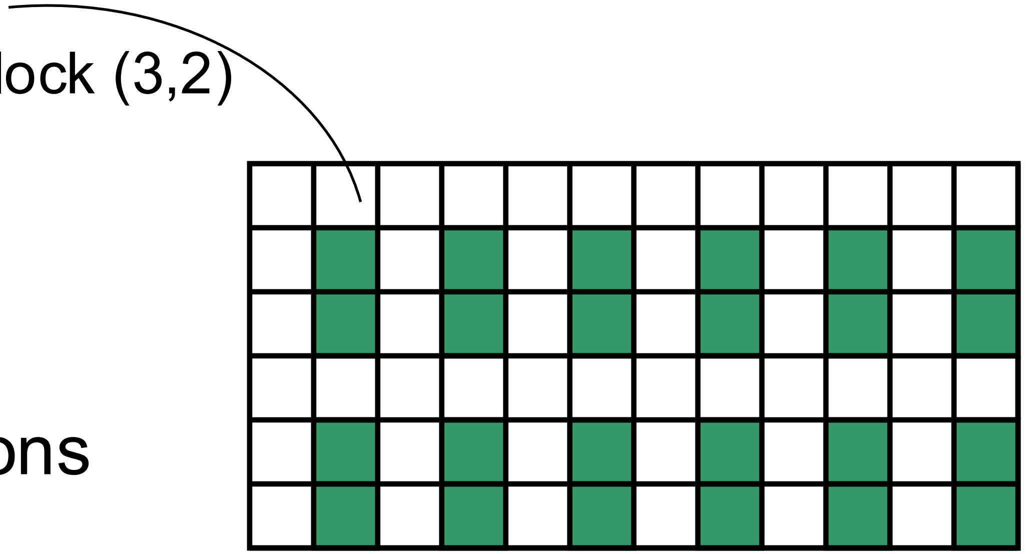
- *Everything is "measured" in the number of elements; 0-based*
- Example 1-dim:
  - Start - starting location of a hyperslab (5)
  - Block - block size (3)



- Example 2-dim:
  - Start - starting location of a hyperslab (1,1)
  - Stride - number of elements that separate each block (3,2)
  - Block - block size (2,1)



  - Count - number of blocks (2,6)

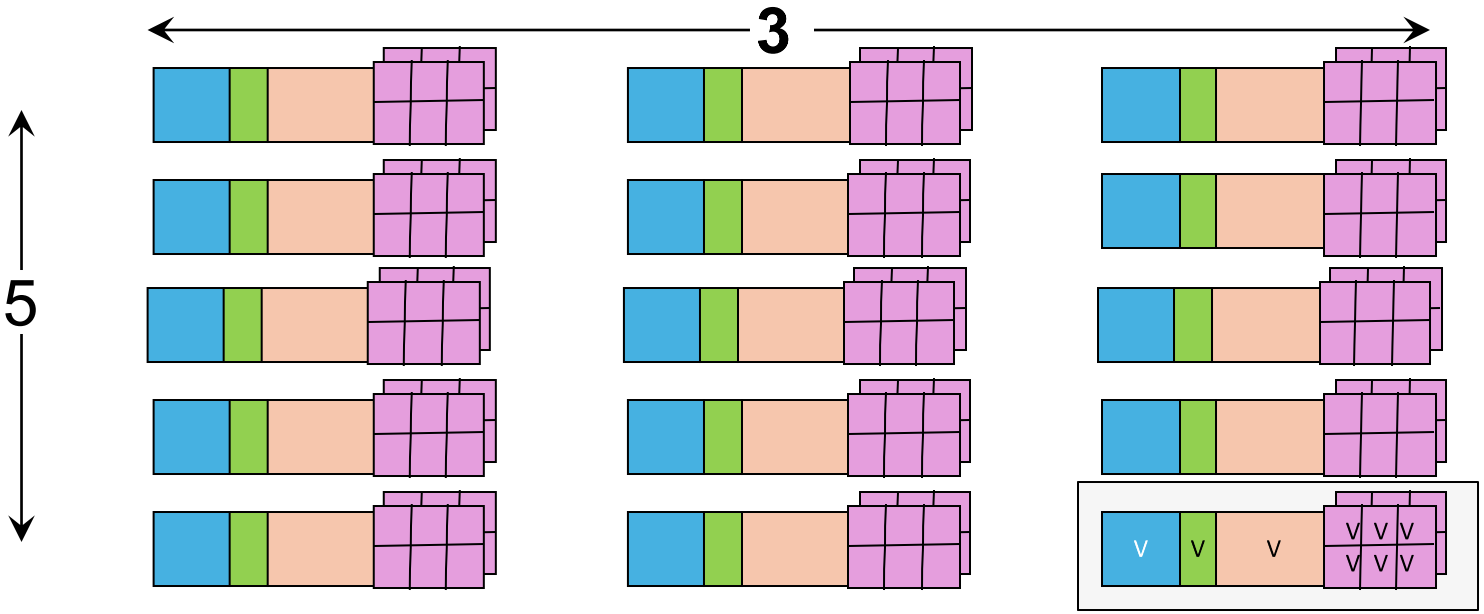


- All other selections are built using set operations

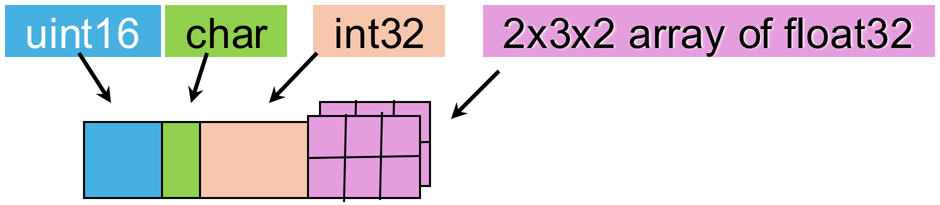
HDF5 Datatypes

- 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)
- HDF5 library provides predefined variables to describe atomic datatypes

HDF5 Dataset with Compound Datatype

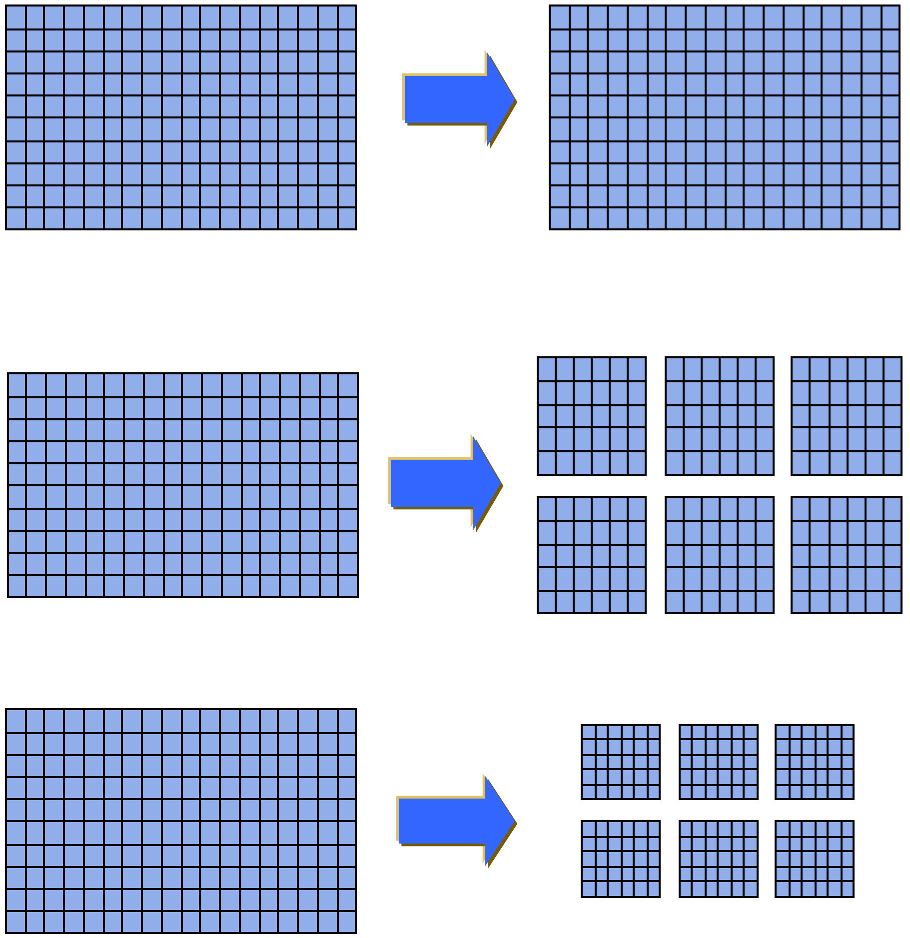


Compound Datatype:



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**

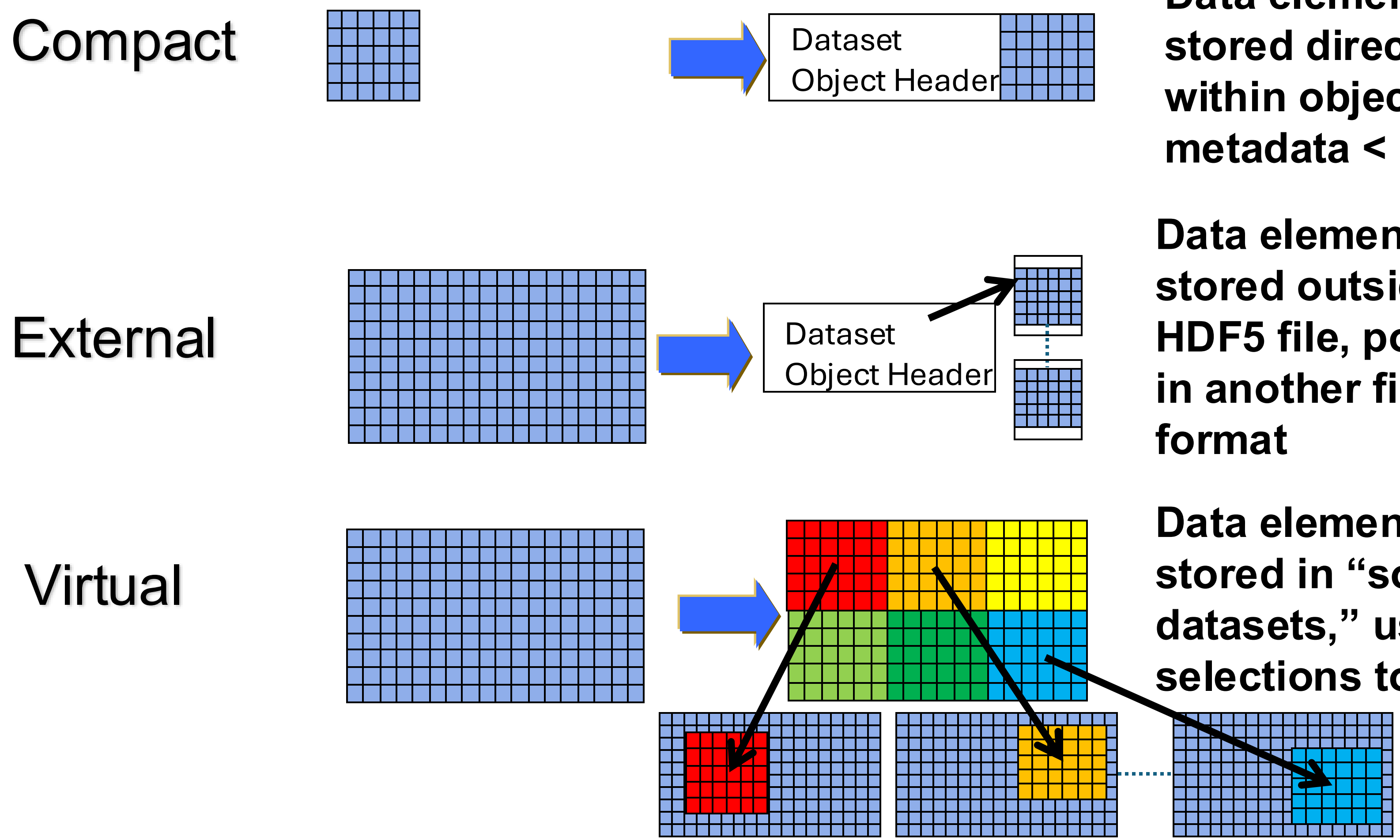
Compression and filters in HDF5

- GZIP and SZIP (free version is available from German Climate Computing Center)
- Other compression methods registered with The HDF Group
  - https://github.com/HDFGroup/hdf5_plugins/blob/master/docs/RegisteredFilterPlugins.md
  - **BZIP2, JPEG, LZF, BLOSC, MAFISC, LZ4, Bitshuffle, SZ and ZFP**, etc.
  - The ones 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: **Data elements stored directly within object's metadata < 64K**

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

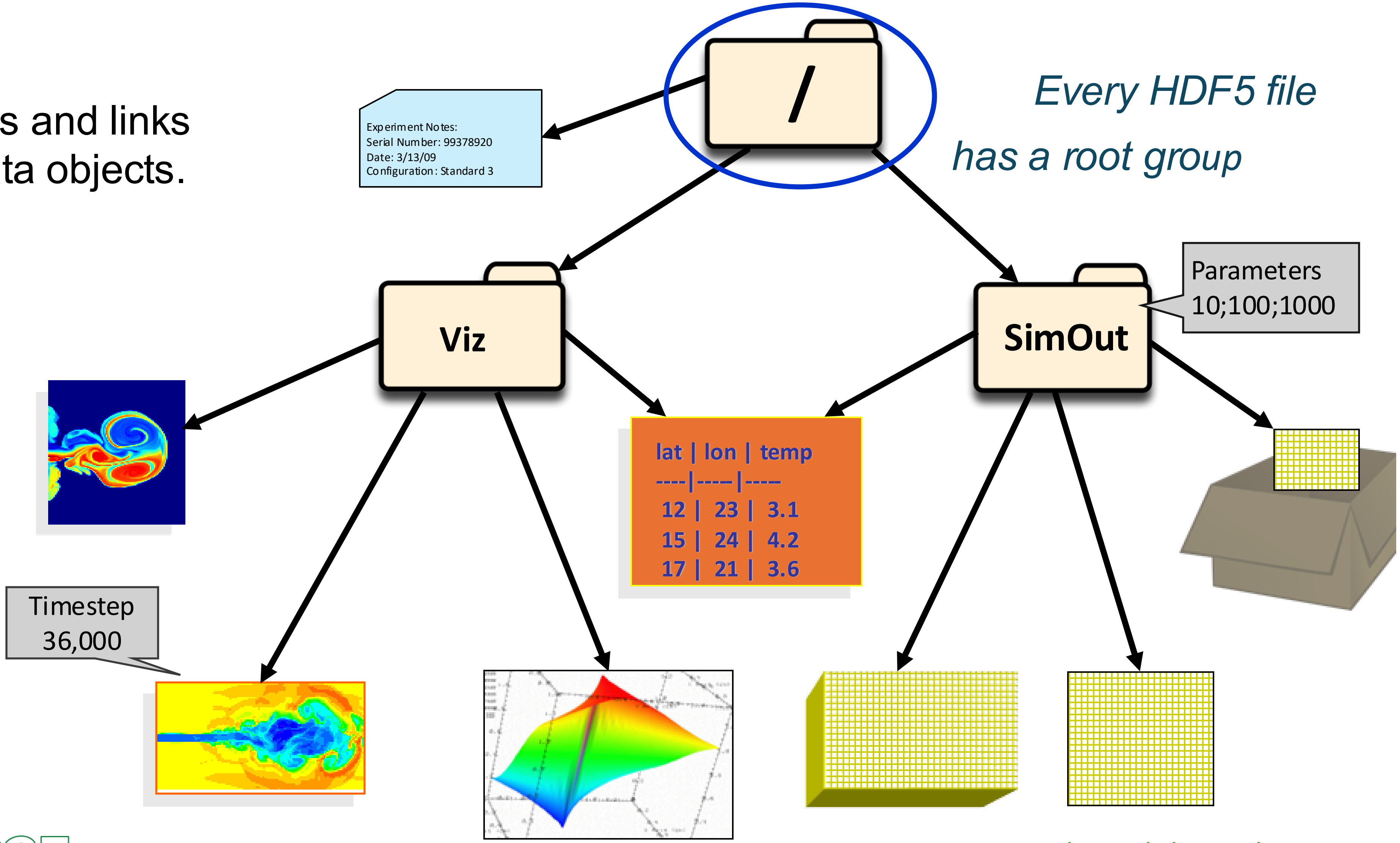
Virtual: **Data elements are stored in "source datasets," using selections to map**

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.



HDF5 software and architecture

HDF5 Software

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

- Latest releases: 1.14.6 (Retired versions ~~1.8, 1.10, 1.12~~), Coming Soon 2.0

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)
- Contemporary C++, including support for MPI I/O
  - https://github.com/ess-dmsc/h5cpp, https://github.com/steven-varga/h5cpp

Useful Tools For New Users

h5dump

Command line tool to “dump” or display the contents of HDF5 files

Scripts to compile applications:

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

HDFView:

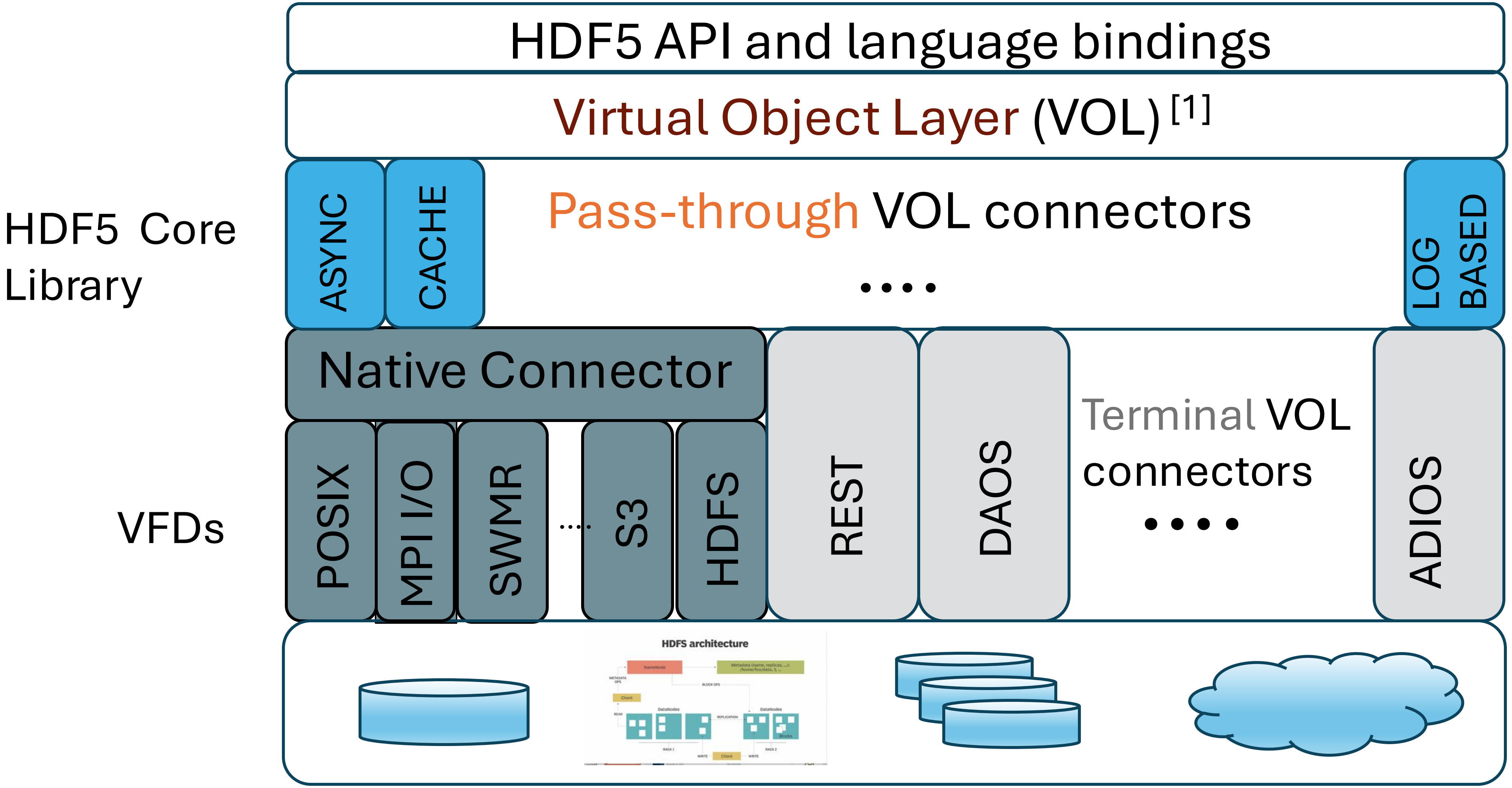
Java browser to view HDF5 file

https://www.hdfgroup.org/downloads/hdfview/

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

https://github.com/HDFGroup/hdf5/tree/develop/HDF5Examples

HDF5 Library Architecture (1.12.0 +)



[1] https://support.hdfgroup.org/documentation/hdf5-docs/registered_vol_connectors.html

HDF5 Programming model and API

The General HDF5 API

- C, FORTRAN, Java, and C++
- The APIs 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

- 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
- Properties (H5**P**) of an object are <u>optionally</u> defined
  - Creation properties (e.g., use chunking storage)
  - Access properties (e.g., using MPI I/O driver to access file)

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
  - If conversion is necessary, increase datatype conversion buffer size (default 1MB) with *H5Pset_buffer()*
- Avoid **dataspace conversion**
  - One-dimensional buffer in memory to a 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)
- 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**
  - Make your chunks the "right" size
    - Goldilocks Principle: Not too big, nor too small
- Understand how **chunking cache** works

  https://support.hdfgroup.org/documentation/hdf5-docs/advanced_topics/chunking_in_hdf5.html

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()*?
- `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

Terminology

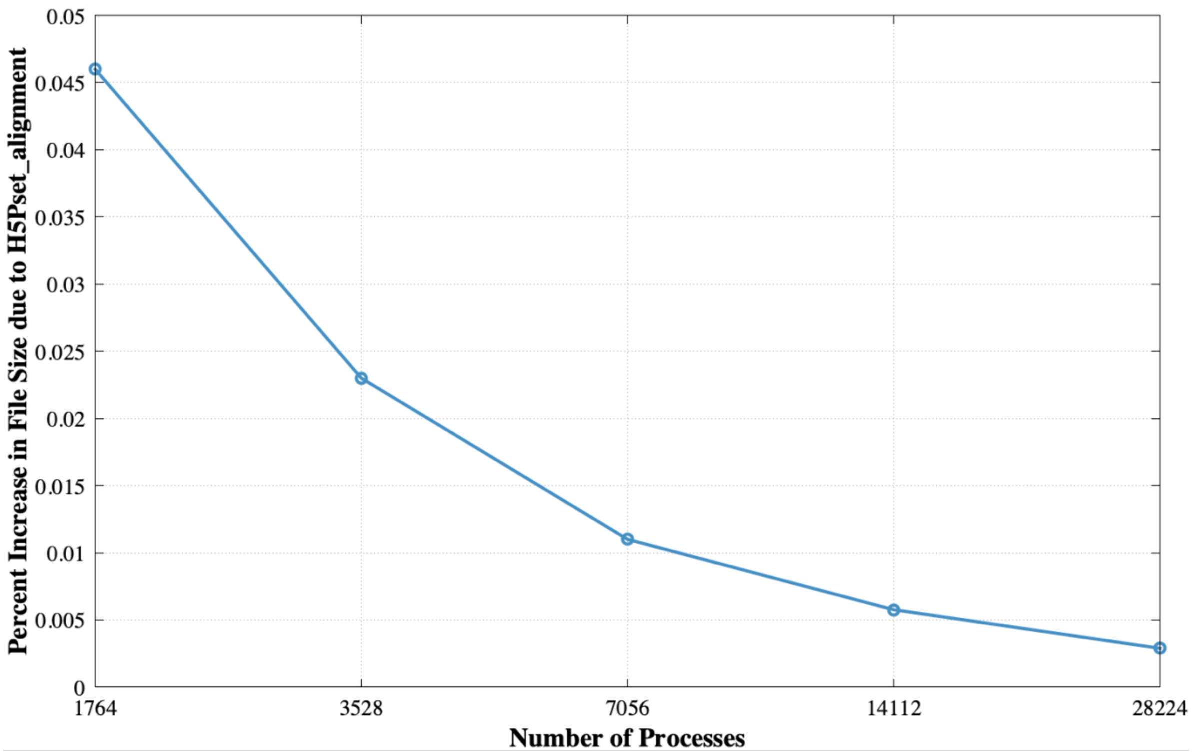
- DATA – “problem-size” data, e.g., large arrays
- METADATA – is an overloaded term
- In this presentation:

  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.

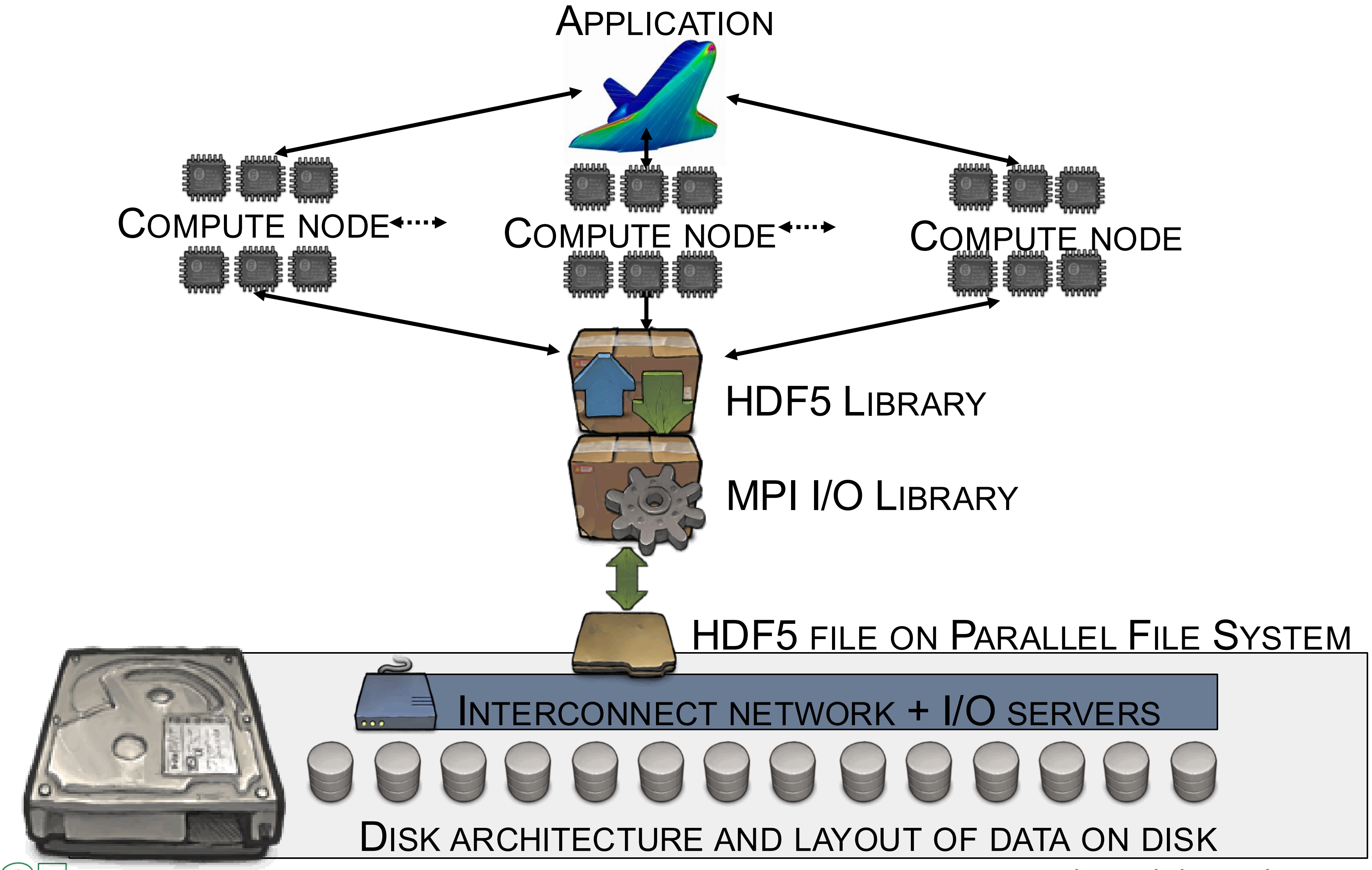
General HDF5 Efficiency

- Faster HDF5 Performance: **Metadata**
  - Use the "latest" file format features
    - *H5Pset_libver_bounds()*
  - Increase the size of metadata data structures
    - *H5Pset_istore_k(), H5Pset_sym_k(), etc.*
  - Aggregate metadata into larger blocks
    - *H5Pset_meta_block_size()*
  - Align objects in the file
    - *H5Pset_alignment()*
  - Control metadata cache
  - Paged allocation and page buffering (serial only)
    - Aggregate and align metadata and small data, perform I/O in aligned pages
    - See File Space Management Documentation https://support.hdfgroup.org/documentation/hdf5-docs/advanced_topics/FileSpaceManagement.html

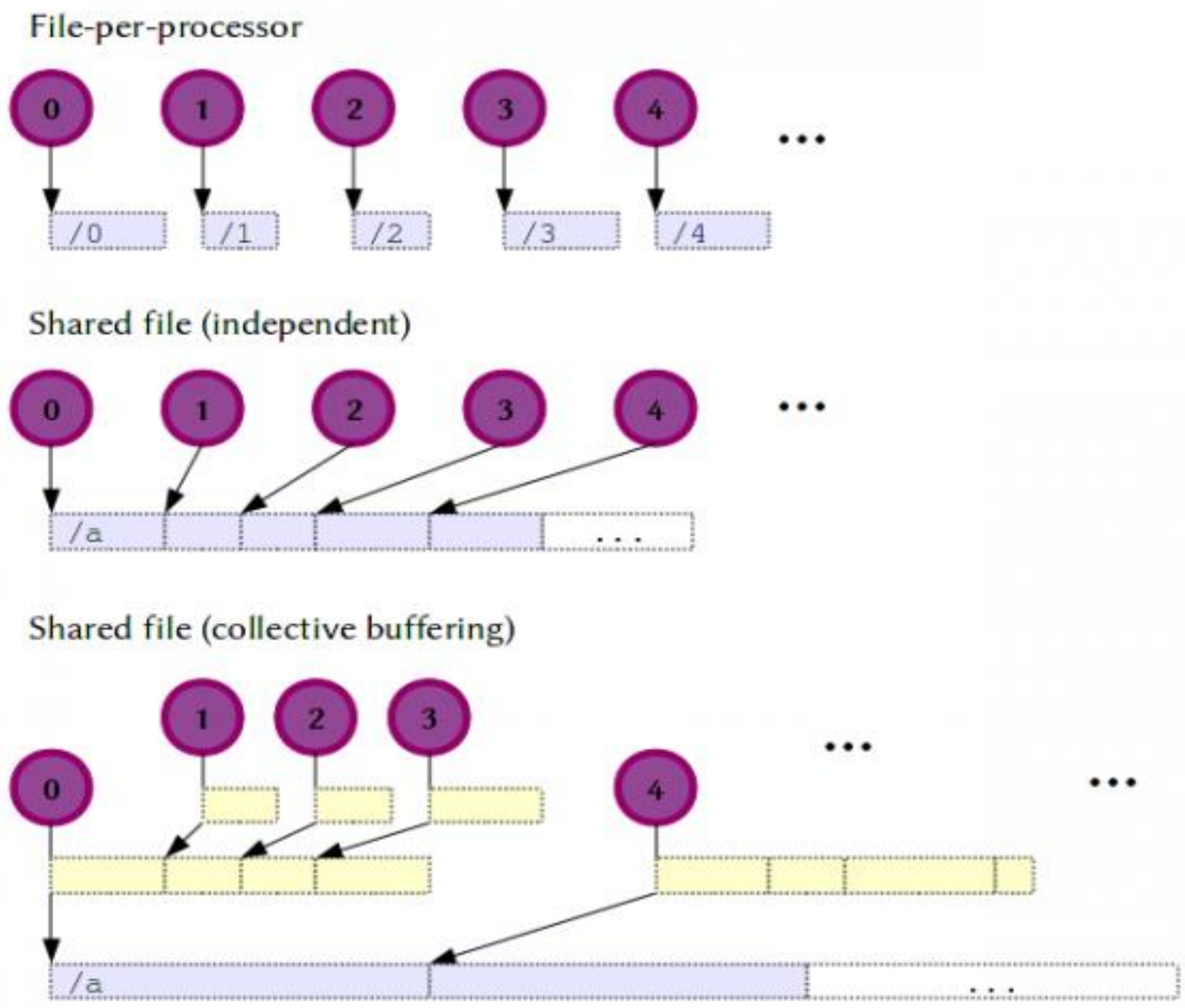


Parallel I/O with HDF5

PHDF5 implementation layers



Types of Application I/O to Parallel File Systems



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

- Maintained code base, performance and data portability
  - Rely on HDF5 to optimize for the underlying storage system

Parallel HDF5 (PHDF5) vs. Serial HDF5

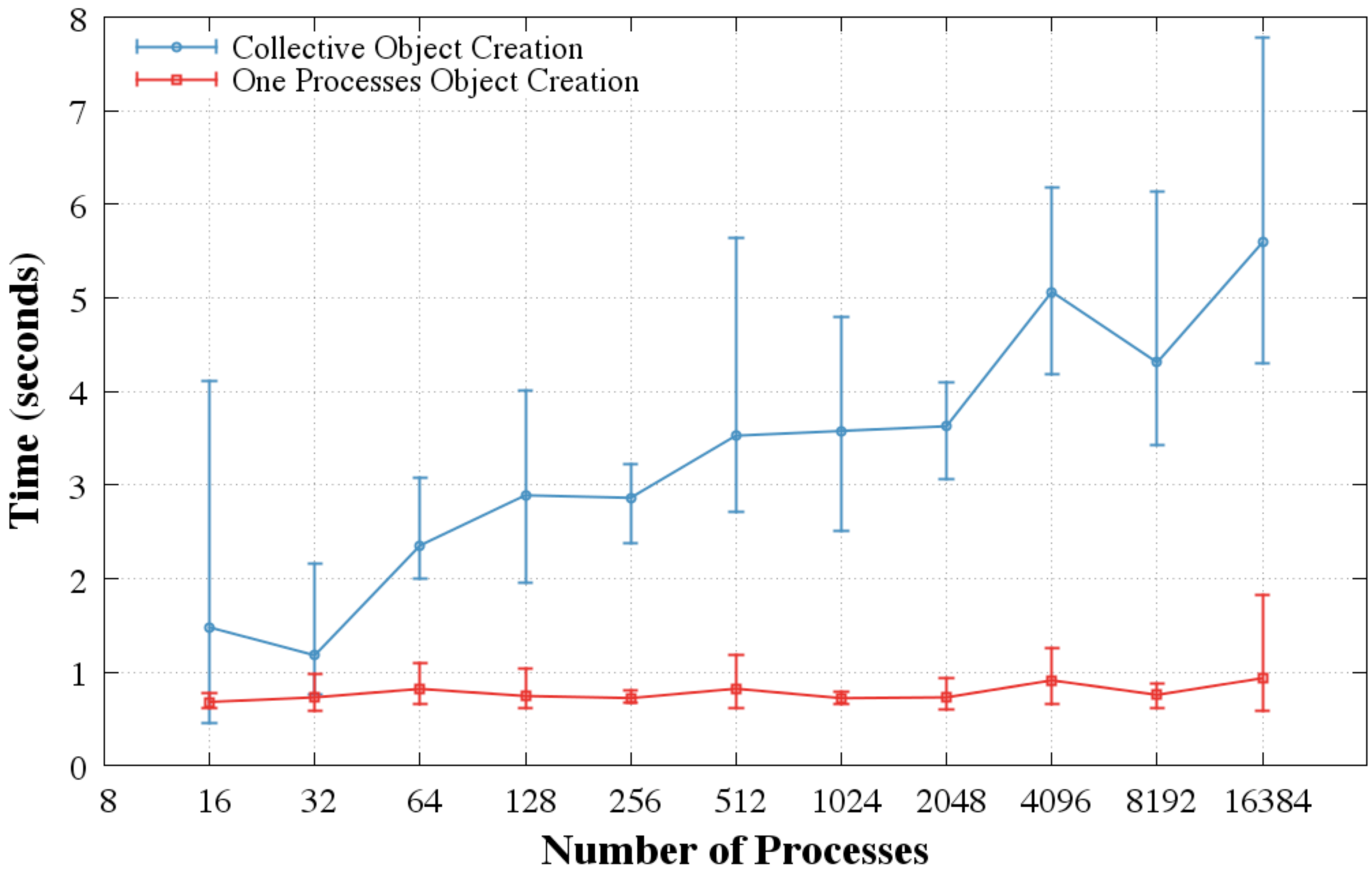
- PHDF5 allows multiple MPI processes in an MPI application to perform I/O to a single HDF5 file
- PHDF5 uses a standard parallel I/O interface (MPI-IO)
- Portable to different platforms
- PHDF5 files ARE HDF5 files conforming to the HDF5 file format specification
- The PHDF5 API consists of:
  - The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

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.
  - Raw data operations can either be collective or independent
    - For collective, all processes must participate, but they don’t need to read/write data.

    https://support.hdfgroup.org/documentation/hdf5/latest/collective_calls.html#sec_collective_calls_func

Object Creation (Collective vs. Single Process)



Collective vs. Independent Operations

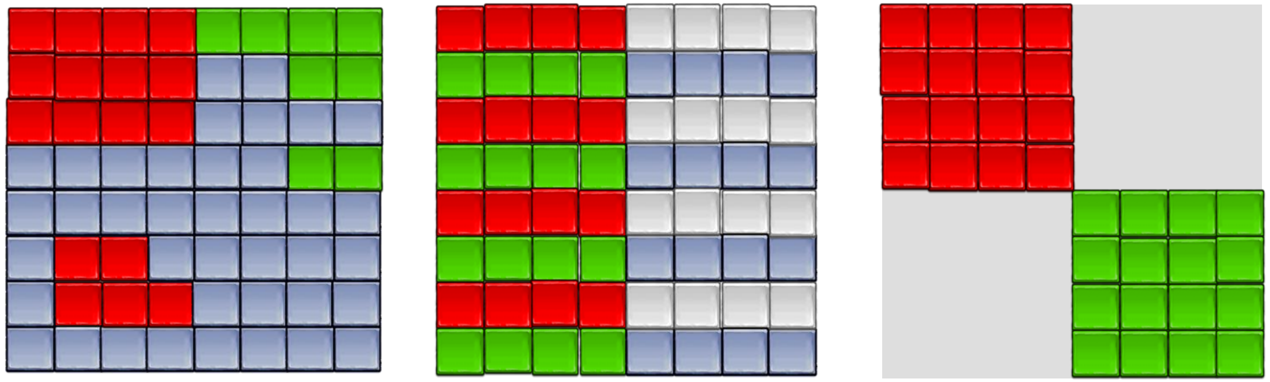
- MPI Collective Operations:
  - All processes of the communicator must participate in the right order. E.g.,

	Process1	Process2	
✓	call A(); call B();	call A(); call B();	...CORRECT
⛔	call A(); call B();	call B(); call A();	...WRONG

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

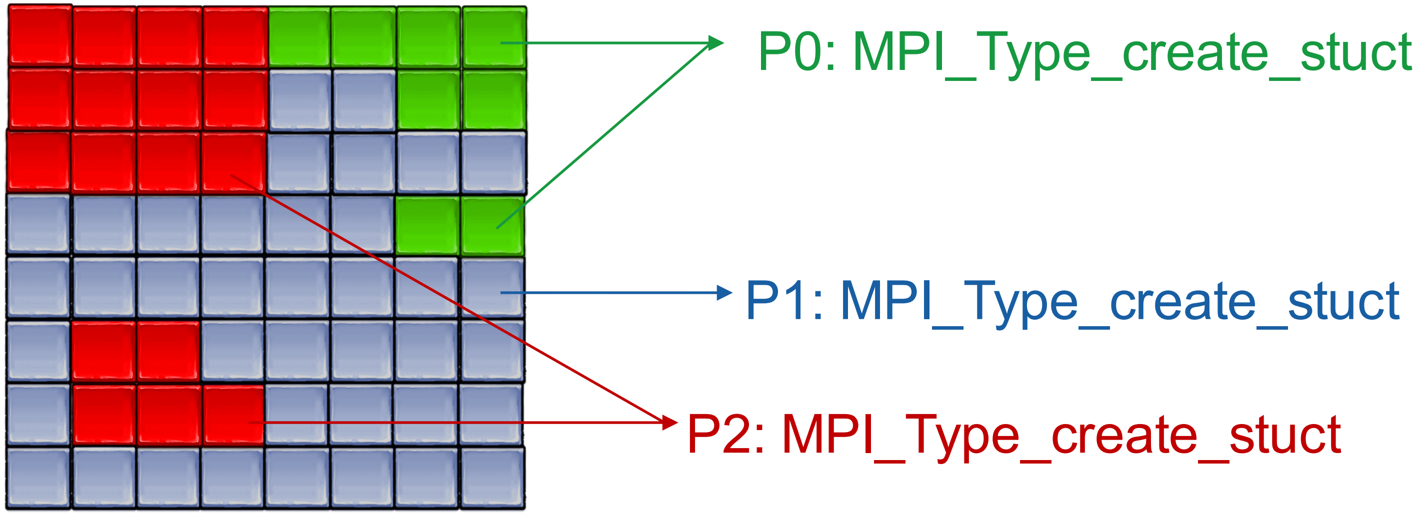
General HDF5 Programming Parallel Model for raw data I/O

- Distributed memory model: data is split among processes
- 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, or Blocks



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

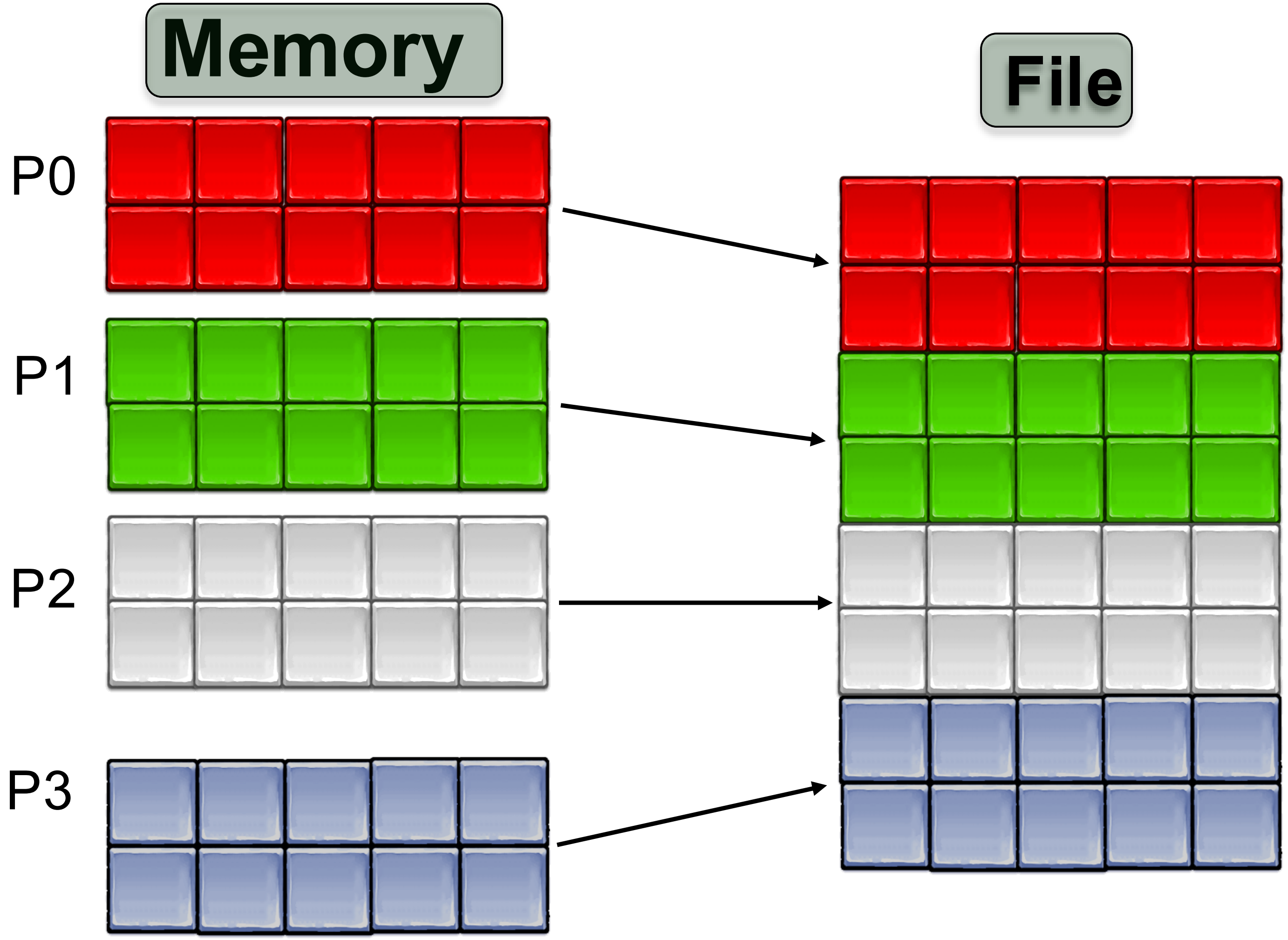
Examples of irregular selection



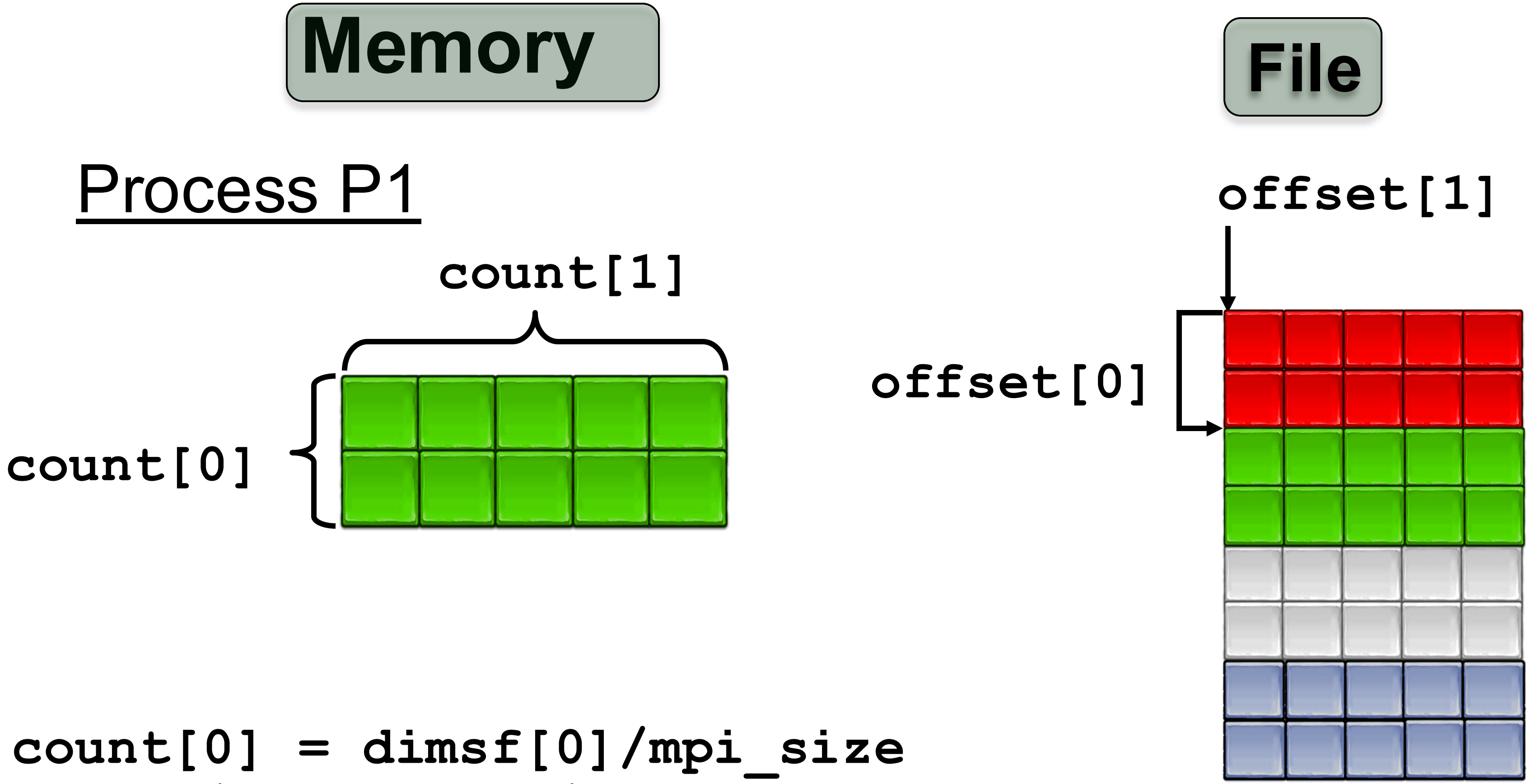
Internally…

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



Example 1: *Writing dataset by rows*




```
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*

```
/*
 * Each process defines dataset in memory and
 * writes it to the hyperslab
 * in the file.
 */
 count[0] = dimsf[0]/mpi_size;
 count[1] = dimsf[1];
 offset[0] = mpi_rank * count[0];
 offset[1] = 0;
 memspace = H5Screate_simple(RANK,count,NULL);

/*
 * Select hyperslab in the file.
 */
 filespace = H5Dget_space(dset_id);
 H5Sselect_hyperslab(filespace,
     H5S_SELECT_SET,offset,NULL,count,NULL);
```

C Example: Collective write and read

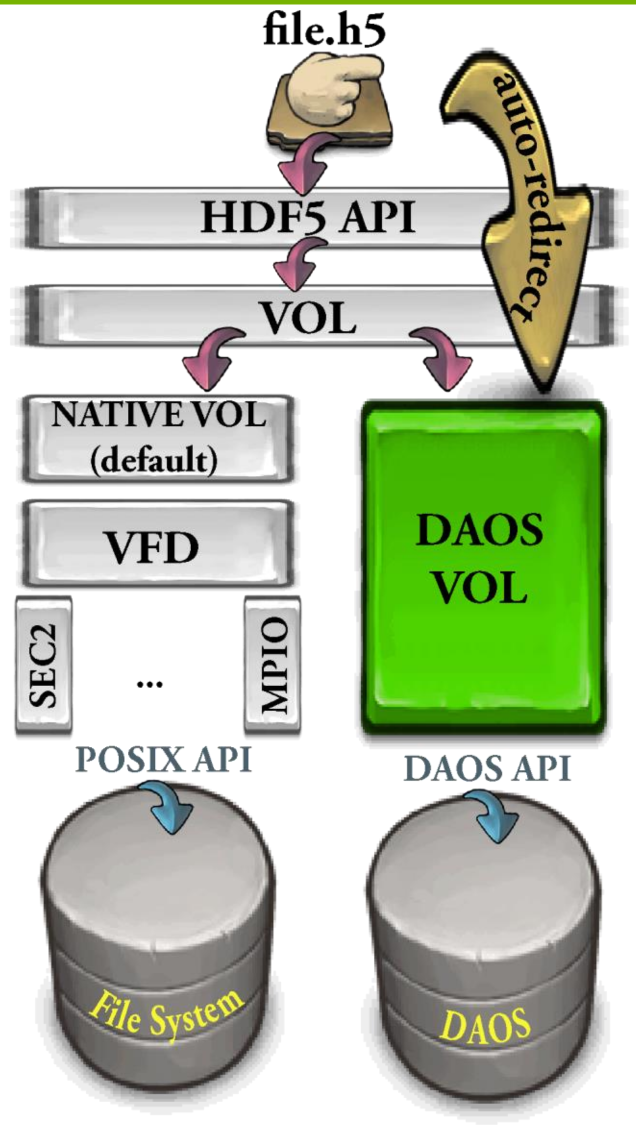
```
/*
 * Create property list for collective dataset write.
 */
plist_id = H5Pcreate(H5P_DATASET_XFER);
-> H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

status = H5Dwrite(dset_id, H5T_NATIVE_INT,
               memspace, filespace, plist_id, data);
/*
 * Collective dataset read.
 */

-> status = H5Dread(dset_id, H5T_NATIVE_INT,
                     memspace, filespace, plist_id, data);
```

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
      }
   }
}
```

The Main Event: DAOS and HDF5

DAOS VOL Connector



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

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

VOL vs. MPI-IO Driver

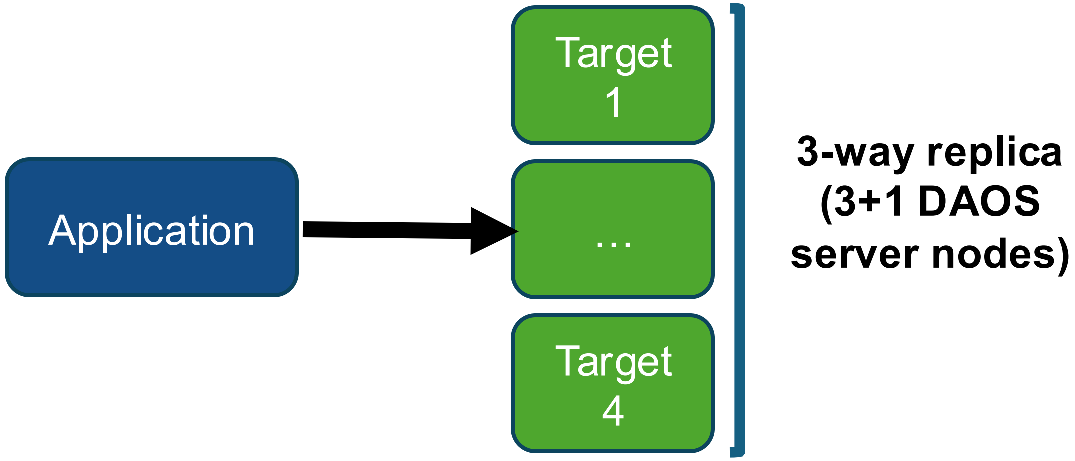
Feature	HDF5 DAOS VOL	DOAS MPI-IO Driver
Performance	**Highest** (low-latency, high throughput)	**Good**(limited by MPI-IO overhead)
Data Path	**Direct**: HDF5 ⇨ DAOS	**Indirect**: HDF5 ⇨ MPI-IO ⇨ DAOS
DAOS Features	Full Access (Native Async, etc.)	Limited Access (Generic Interface)
Code Changes	Recommended for new/modernized code	Minimal to None for existing code
Primary Use Case	Performance-critical applications	Legacy application compatibility
Crash Handling	**Direct & Native:** Leverages DAOS's robust, transparent recovery.	**Indirect & Abstracted:** Relies on the MPI layer, complicating recovery.

DAOS VOL – Data placement and Replication

- Multiple options
  - **Chunking enabled by default for contiguous datasets**, controlled with:

    `H5Pset_chunk()`
  - Set DAOS object class per DAOS object to control number of targets used for storing object (= **sharding**) as well as the number of **replicas** (for recovery) :

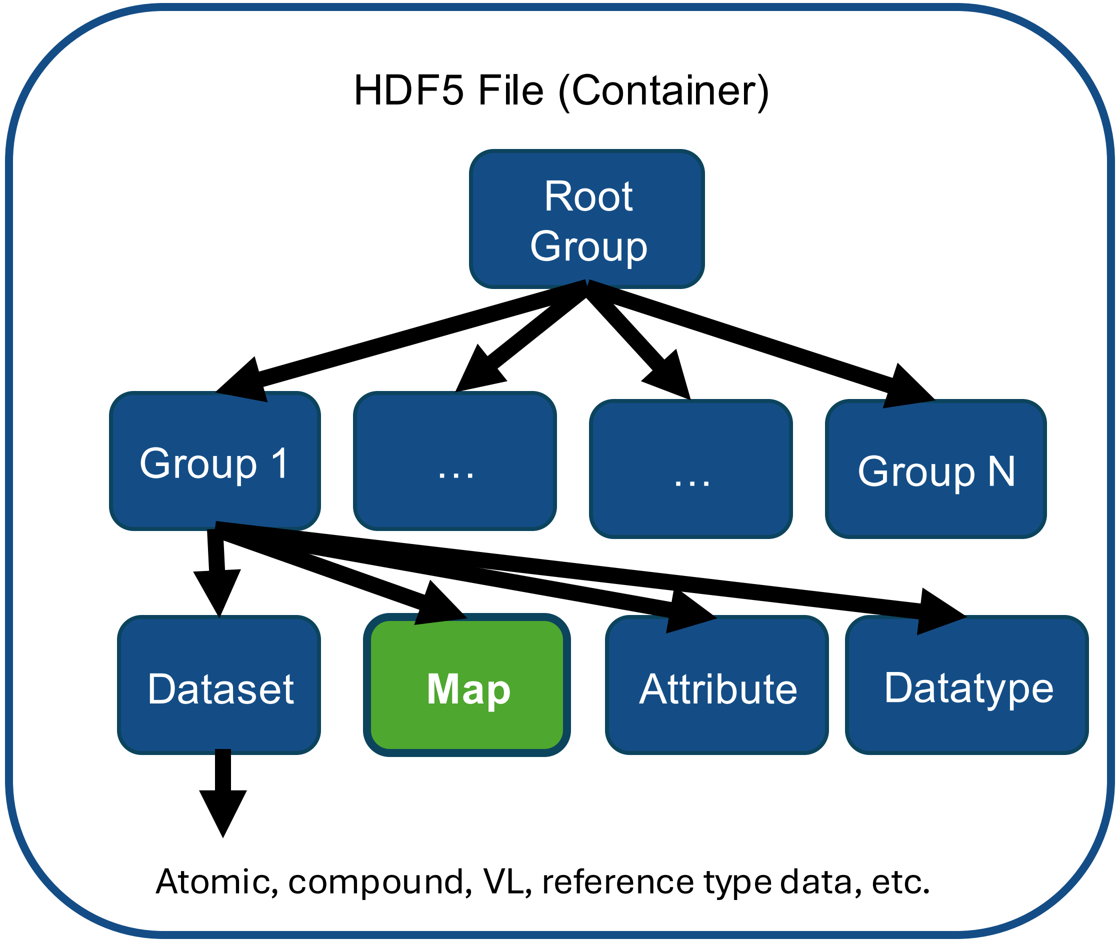
    `H5daos_set_object_class()`
    - Default for datasets is to shard across all available targets, with no replication



target ≠ storage node:
multiple storage targets per node

DAOS VOL – HDF5 Objects and Features

- The majority of HDF5 features are currently supported, except:
  - Native file format specific APIs
  - Compression filters
- Additional features implemented
  - Map objects (enabled by K/V objects)
  - File deletion
  - Independent metadata
    - HDF5 objects can be created independently
    - Enabled with:
      ***H5daos_set_all_ind_metadata_ops*()**
  - Asynchronous I/O



DAOS VOL – Async I/O with DAOS

- Enables asynchrony using an ***E**vent **S**et* API
  - Implemented at the DAOS connector level
  - Uses DAOS task engine (does not necessarily need additional progress thread)
  - HDF5 API can return before the operation completes, placing the operation in an "event set"
- Asynchrony must be <u>explicitly controlled</u> by the application
  - Similar to existing async APIs, such as MPI non-blocking
  - Place async tasks in an Event Set (H5ES)
  - Use async versions of all routines that may block
  - Applications are expected to rework/optimize their code to avoid memcpy, avoid memory modifications of async buffers, and correct async error handling.

DAOS VOL – Getting started

- Buil t using HDF5 version1.14.x, compatible with v2.0 coming soon.

```
CC=mpicc configure --enable-parallel --disable-static --enable-map-api
```

- Build the DAOS VOL

```
#!/bin/bash
export HDF5_ROOT=$HOME/packages/hdf5/build/hdf5
cmake -D CMAKE_BUILD_TYPE=Release -D BUILD_EXAMPLES=TRUE \
      -D CMAKE_INSTALL_PREFIX=$PWD -D CMAKE_C_COMPILER=mpicc ..
make -j 8 install
```

DAOS VOL – Getting started – Using it

- Creation and use of HDF5 files in DAOS
  - Minimal or no code changes for the application developer (<u>if only looking for compatibility</u>)
  - Two ways to select the DAOS connector:
    1. HDF5 file access property list (*recommended for new files or when manipulating multiple VOLs*)

       1. `H5Pset_fapl_daos()`
       2. `Include daos_vol.h and daos.h, link to libhdf5_vol_daos.so`

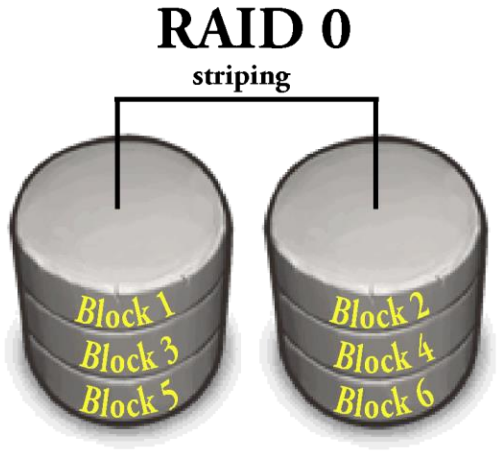
    2. Environment variable

```
HDF5_VOL_CONNECTOR=daos
HDF5_PLUGIN_PATH=/path/to/connector/folder/lib
DAOS_POOL = <pool uuid>
```

GO TO Aurora;

Subfiling

- An MPI-based parallel file driver is used to split an HDF5 file across a collection of subfiles in equally sized data segment stripes.
    - Data stripe size is the amount of data (in bytes) that can be written to a subfile before data is placed in the next subfile in a round-robin (default) fashion
    - Defaults to 1 subfile per machine node with 32MiB data stripes



Subfiling is a compromise between file-per-process (*fpp*) and a single shared file (*ssf*)

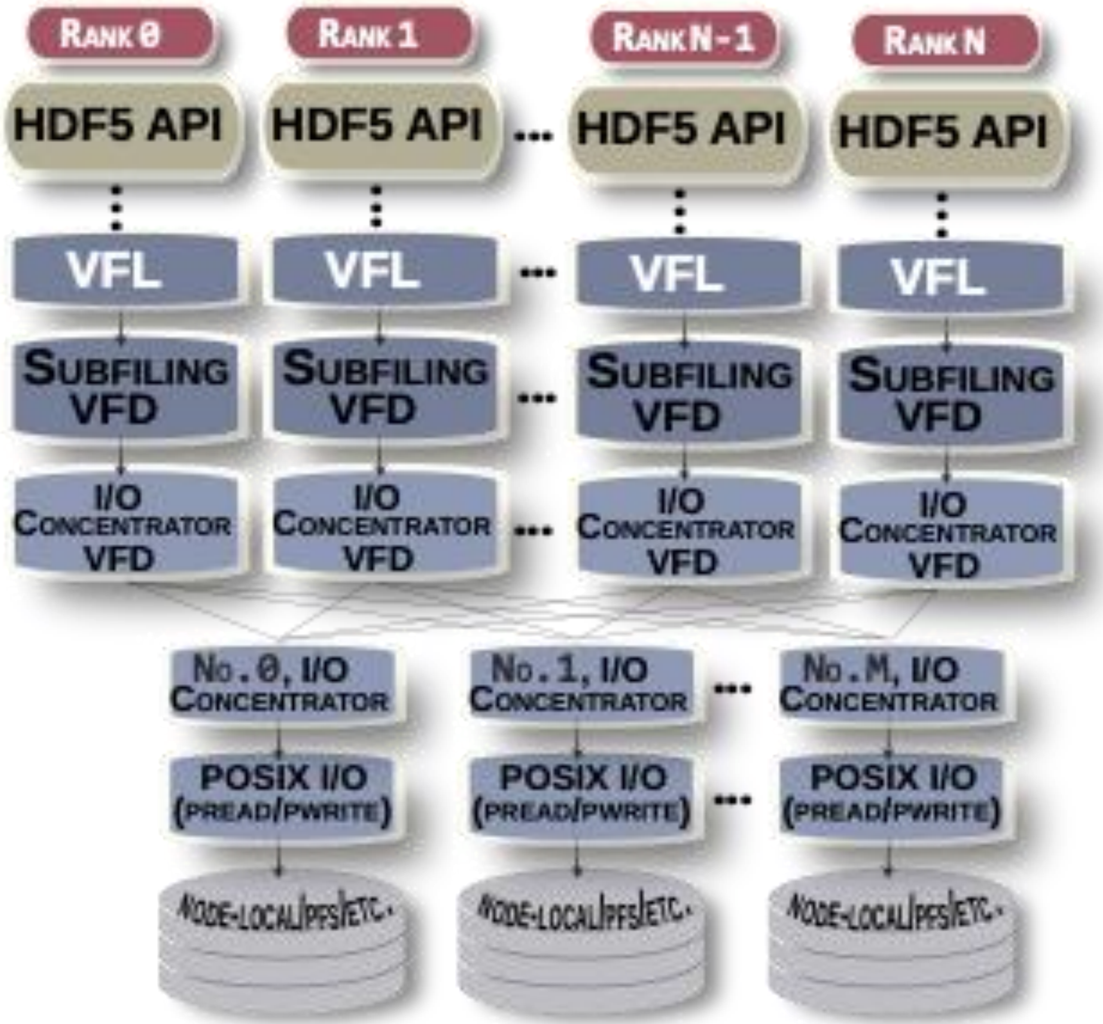
Minimize the locking issues of *ssf* approach

Avoid some complexity and reduce total number of files compared to *fpp* approach

Designed to be flexible and configurable for different machines

What is it? (continued)

- Uses "I/O concentrators" - a subset of available MPI ranks that control subfiles and operate I/O worker thread pools.
  - N-to-1 mapping from subfiles -> I/O concentrator ranks
  - I/O from non-I/O-concentrator MPI ranks is forwarded to the appropriate I/O concentrator based on offset in the logical HDF5 file
  - Default: Subfiles are assigned round-robin across the available I/O concentrator ranks



Subfiling Output Files per Logical HDF5 File

- HDF5 stub file
  - Appears as a normal HDF5 file; only contains HDF5 superblock information and subfiling parameter information
  - Useful for compatibility with HDF5 applications that read initial bytes of file, e.g., CGNS, NetCDF4
  - Inode value of stub file used to generate unique filenames for configuration file and subfiles

```
bash-5.1$ ls
outFile.h5
outFile.h5.subfile_12190989.config
outFile.h5.subfile_12190989_1_of_4
outFile.h5.subfile_12190989_2_of_4
outFile.h5.subfile_12190989_3_of_4
outFile.h5.subfile_12190989_4_of_4
```

Subfiling Output Files per Logical HDF5 File

Subfiling configuration text file

- A simple configuration file detailing the subfiling parameters for an existing file
- Validated against subfiling parameters stored in HDF5 stub file once logical HDF5 file has been opened
- Useful for external tooling to get subfiling parameter information

```
bash-5.1$ ls
outFile.h5
outFile.h5.subfile_12190989.config
outFile.h5.subfile_12190989_1_of_4
outFile.h5.subfile_12190989_2_of_4
outFile.h5.subfile_12190989_3_of_4
outFile.h5.subfile_12190989_4_of_4
```

```
stripe_size=1048576
aggregator_count=4
subfile_count=4
hdf5_file=/home/jhenderson/subfiling/outFile.h5
subfile_dir=/home/jhenderson/subfiling
outFile.h5.subfile_12190989_1_of_4
outFile.h5.subfile_12190989_2_of_4
outFile.h5.subfile_12190989_3_of_4
outFile.h5.subfile_12190989_4_of_4
```

Subfiles

Contains all the file data, including superblock information duplicated in HDF5 stub file

Subfiling

- Subfiling file driver is set on a File Access Property List

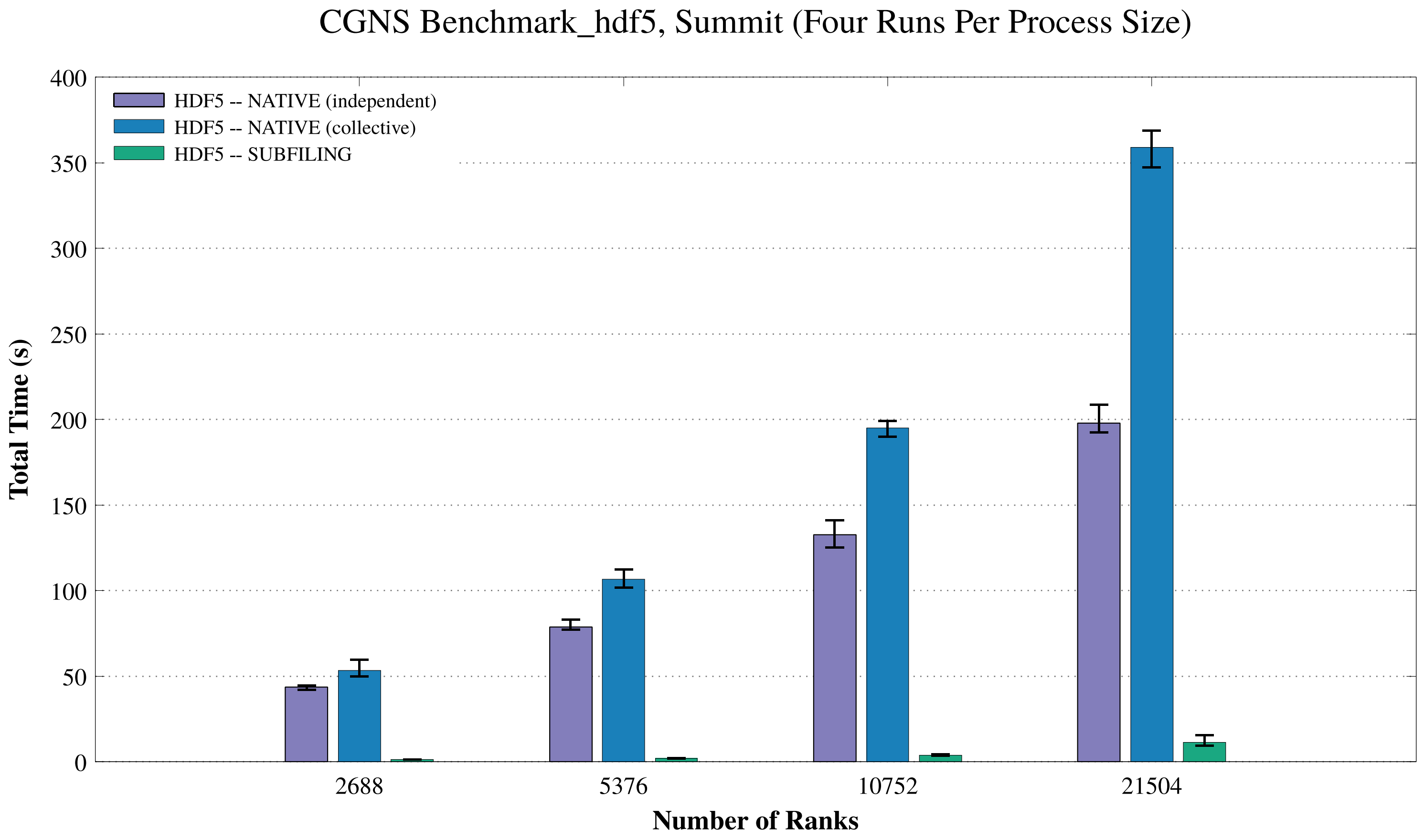
```
plist_id = H5Pcreate(H5P_FILE_ACCESS);
status = H5Pset_fapl_subfiling(plist_id, vfd_config);
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
H5Pclose(plist_id);
```

- Environment variables control options:
- **H5FD_SUBFILING_IOC_PER_NODE**– Number of I/O concentrators per node.
- **H5FD_SUBFILING_STRIPE_SIZE** – Maximum contiguous block of data that can be written to a single I/O Concentrator before moving on to the next IOC.
- **H5FD_IOC_THREAD_POOL_SIZE** – Sets the number of I/O Concentrator helper threads. **The default is four pool threads**.
- **H5FD_SUBFILING_CONFIG_FILE_PREFIX** — Sets the prefix of the configuration file. Useful when using node-local storage.
- **H5FD_SUBFILING_SUBFILE_PREFIX** – Sets the prefix for the subfiles. Useful when using node-local storage

Subfiling

- (CGNS[1] **benchmark_hdf5**)
- The default settings for Subfiling were used, one subfile per node.

Number of Ranks	HDF5 File Size
21504	53 GiB
10752	27 GiB
5376	14 GiB
2688	6.6 GiB

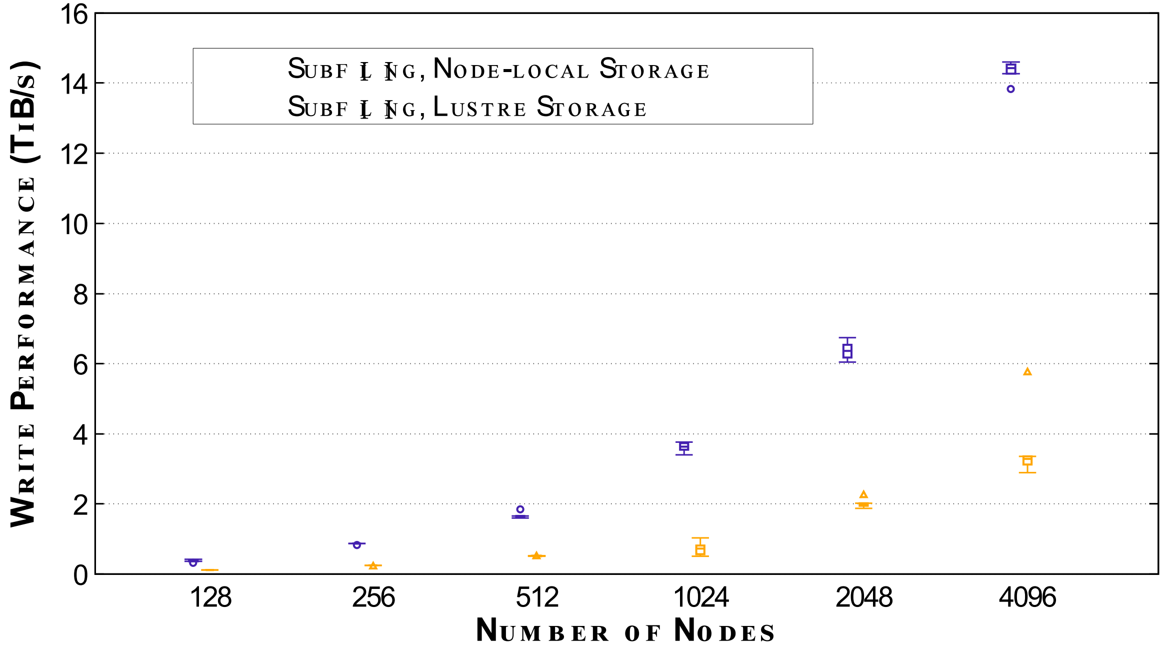


[1] CGNS = Computational Fluid Dynamics (CFD) General Notation System, cgns.org

Subfiling – ExaMPM [1] (Cabana [2]) on Frontier (OLCF)

- GPU computation engine
  - — Kokkos is used to transfer memory between GPU and CPUs
- Subfilings *pwrite* throughput for 4096 nodes

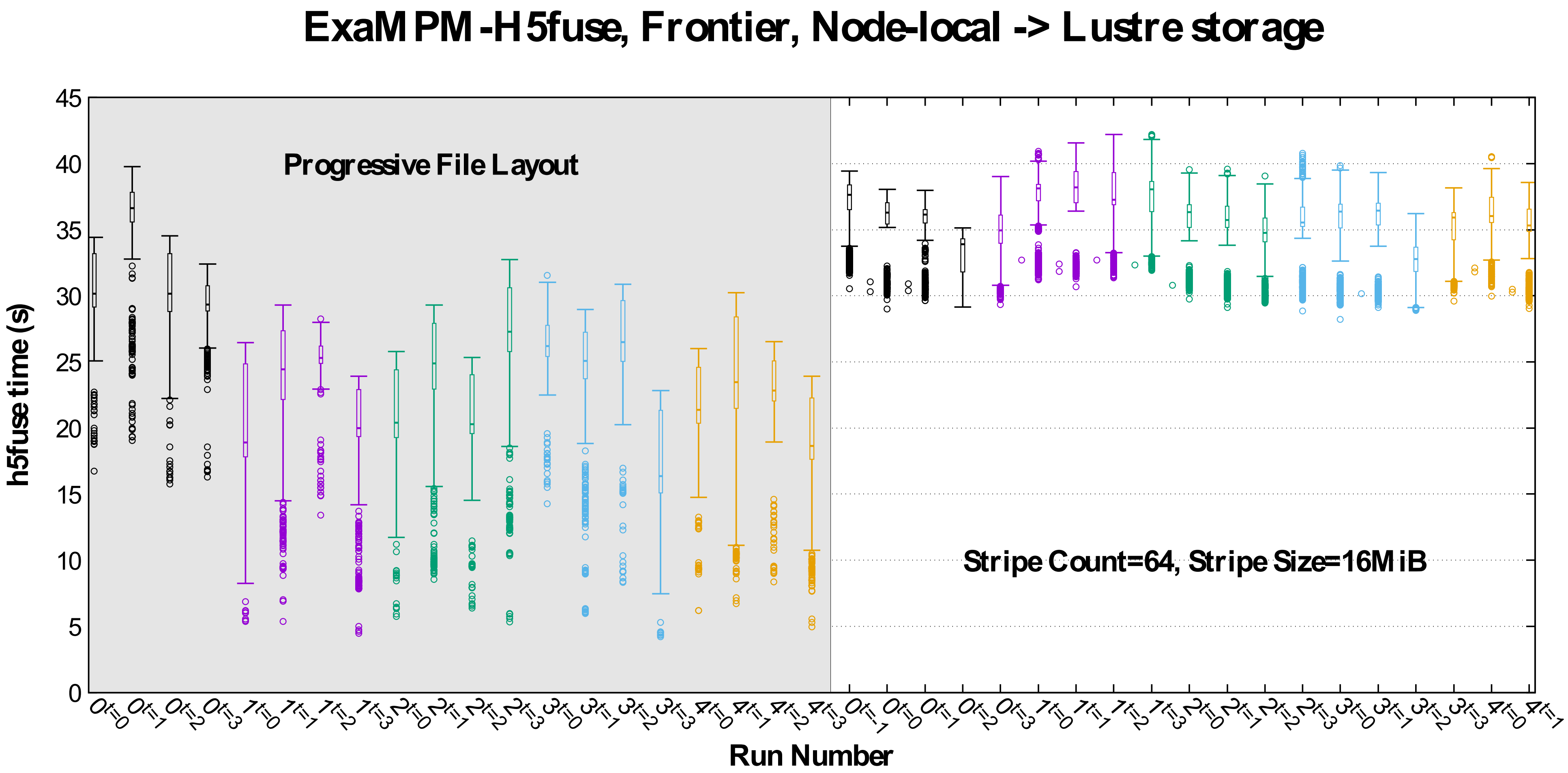
Number of Nodes	Size (GiB)	
	Per Output	Total
128	122	610
256	195	975
512	482	2410
1024	981	4905
2048	1950	9750
4096	2083	10415



[1] https://github.com/ECP-copa/ExaMPM ,
[2] https://github.com/ECP-copa/Cabana

Subfiling

- (Cabana/ExaMPM)



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

ARGONNE

ATPESC2025

EXTREME-SCALE COMPUTING

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicagoArgonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.