Organizing Chaos: HDF5 for Scientific Data

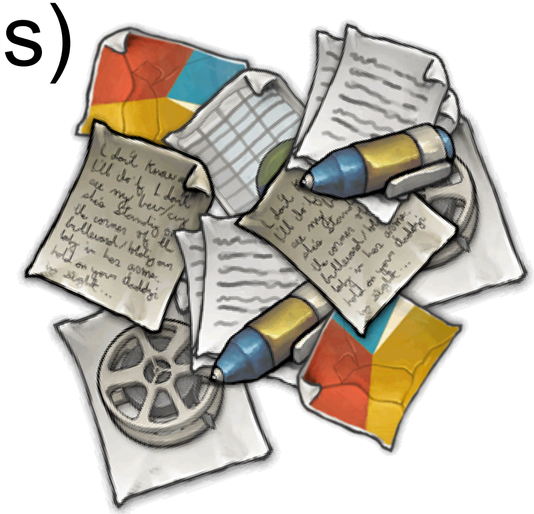
M. Scot Breitenfeld

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_run***n*th**.csv
  - metadata_run1.txt … metadata_run***n*th**.txt
  - images_run1_001.png… images_run***n*th**_001.png
- **Accessing data is slow and complicated.**
  - To find images from a specific run, you have to open and parse multiple files

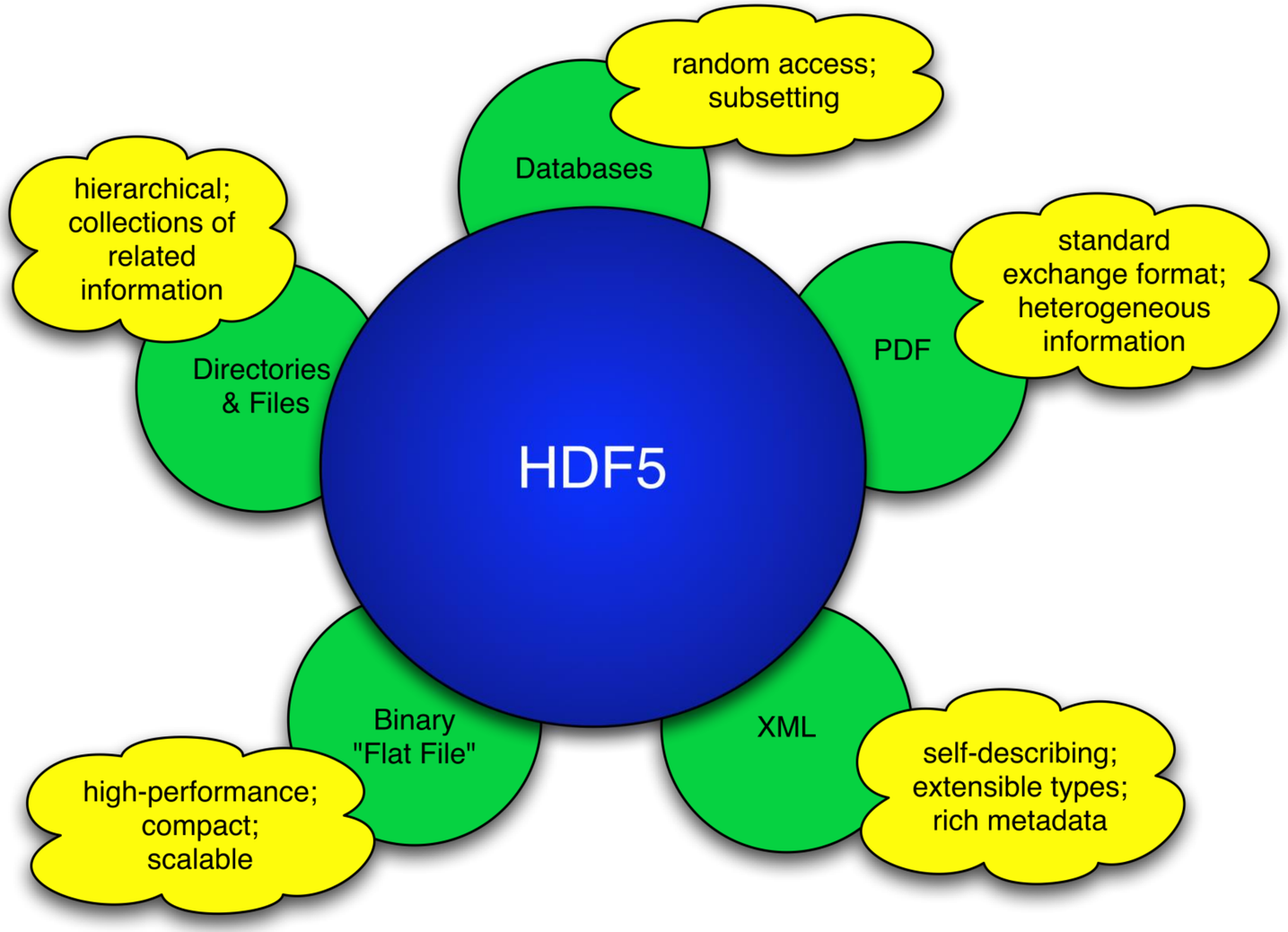
Talk Outline

- Understanding HDF5: The Essentials
- Introducing HDF5 into ML workflows
- Scaling Up: Parallel HDF5 and Advanced 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)

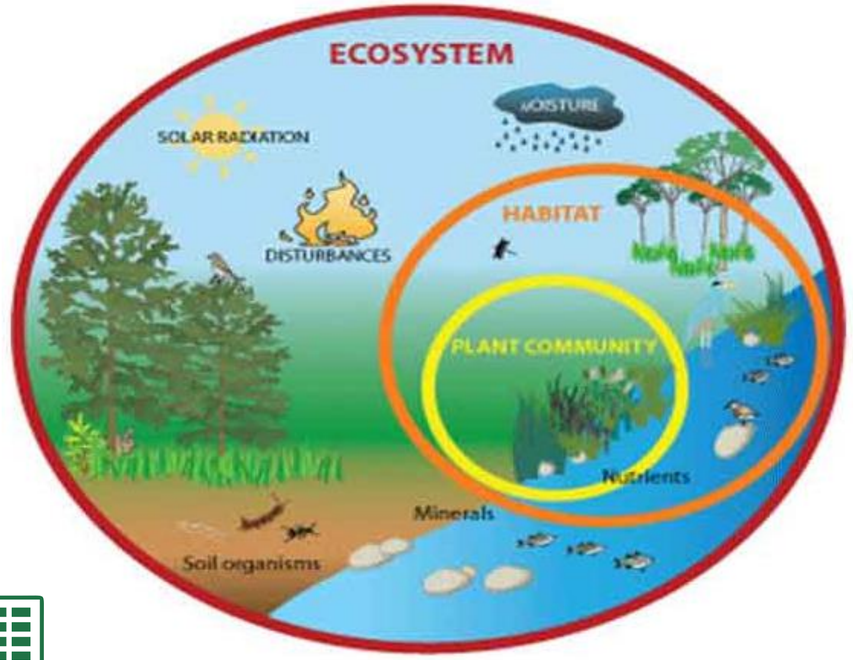
HDF5 is like …

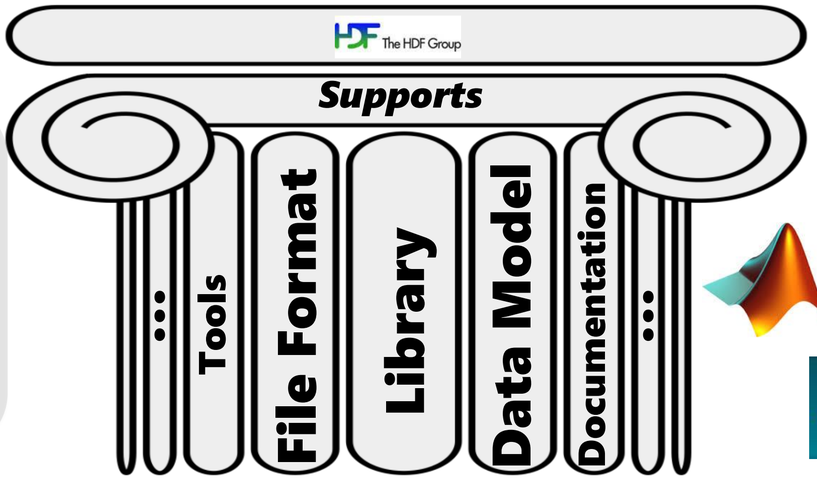

HDF5
Databases
random access; subsetting
Directories & Files
hierarchical; collections of related information
PDF
standard exchange format; heterogeneous information
Binary "Flat File"
high-performance; compact; scalable
XML
self-describing; extensible types; rich metadata

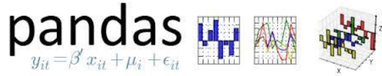
HDF5 is designed for…

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

HDF5 Ecosystem



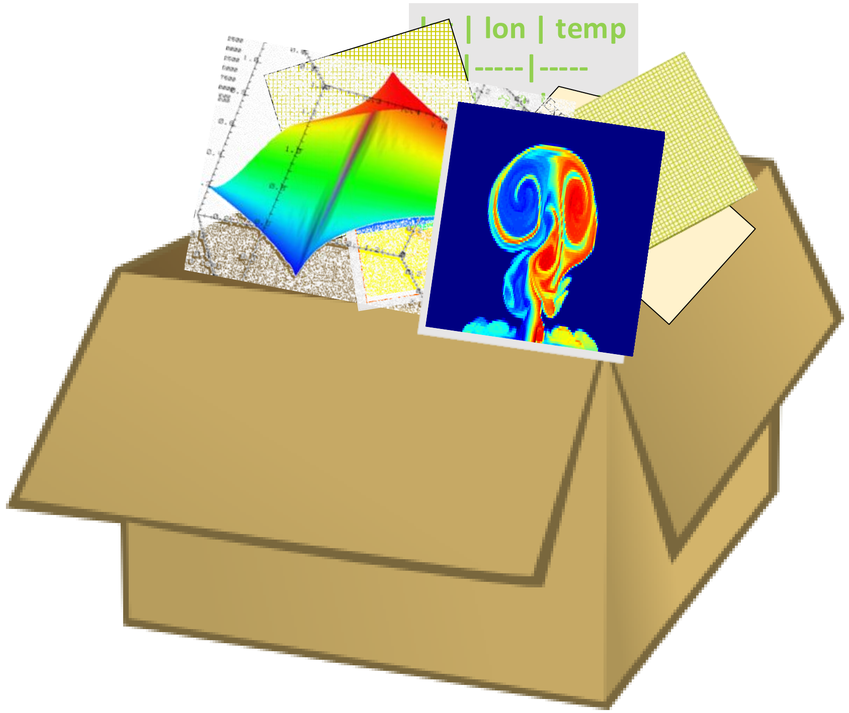




HDF5 Data model

HDF5 File

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



HDF5 Data Model

Dataset –
Organize and contain data elements

Dataspace –
Describes logical layout of the data elements

Attribute –
User-defined metadata

HDF5 Objects

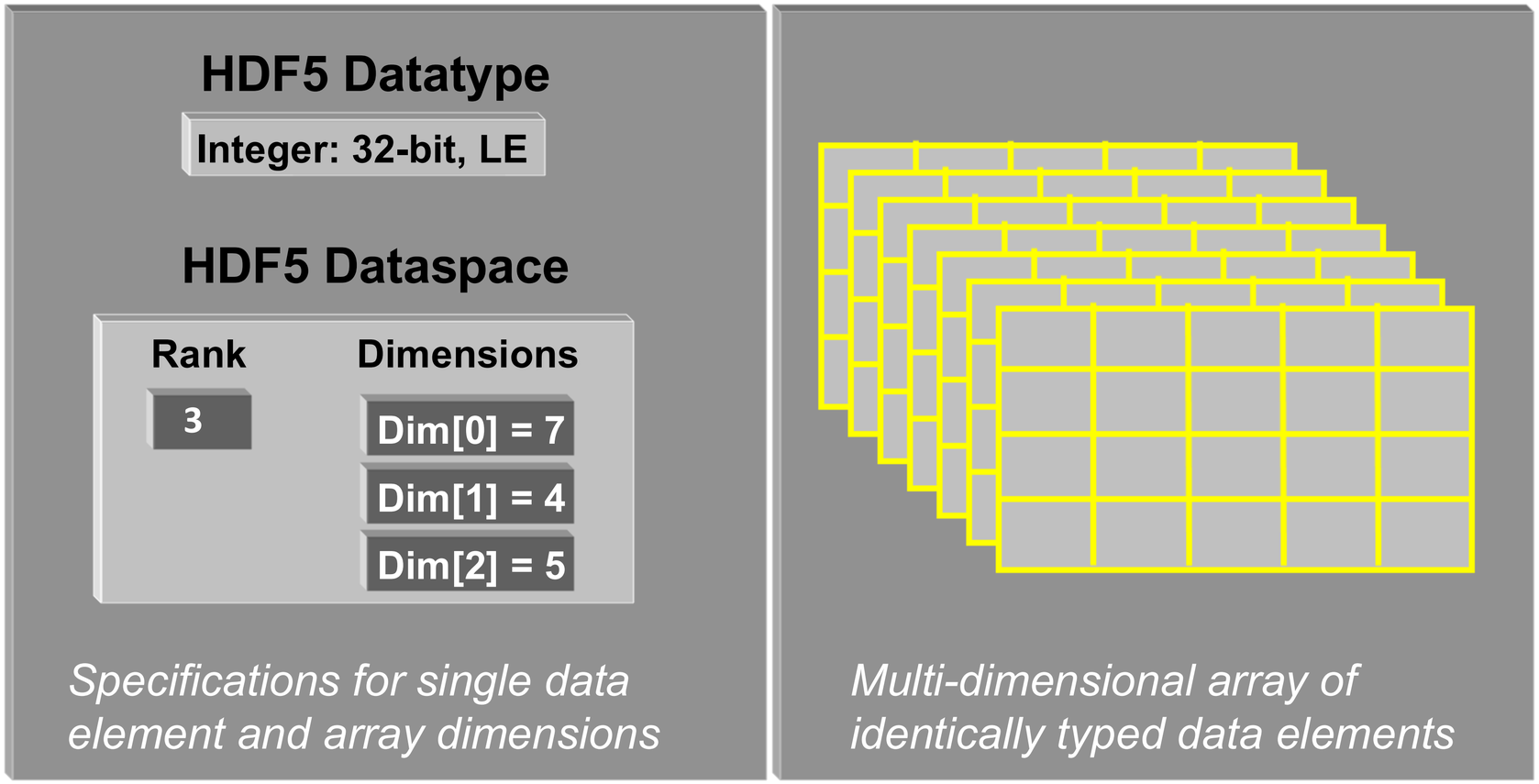
File

Datatype –
Describes individual data elements

Link –
Organize data objects

Group –
Organize data objects

HDF5 Dataset



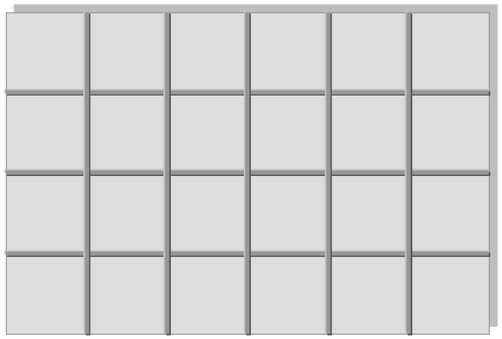
- HDF5 datasets **organize and contain** data elements
  - HDF5 datatype describes individual data elements
  - HDF5 dataspace describes the logical layout of the data elements

HDF5 Dataspace

Two roles:

(1) Spatial information for Datasets and Attributes

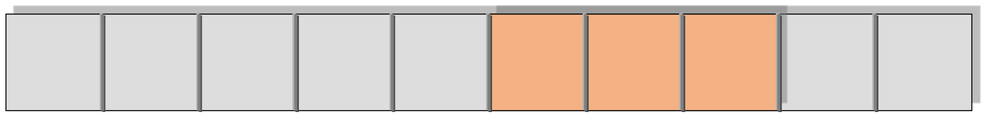
- Empty sets and scalar values
- Multidimensional arrays
  - Rank and dimensions
- A permanent part of object definition



Rank = 2

Dimensions = 4 x 6

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



Rank = 1

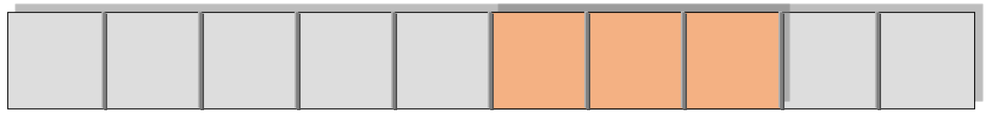
Dimension = 10

How to describe a subset in HDF5?

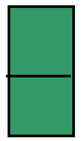
- Before writing and reading a subset of data, one must describe it to the HDF5 Library.
- The HDF5 APIs and documentation refer to a subset as a "***selection***," for example "*hyperslab* selection."
- If specified, HDF5 performs I/O on a selection *only* and not on all dataset elements.

Describing elements for I/O: HDF5 Hyperslab

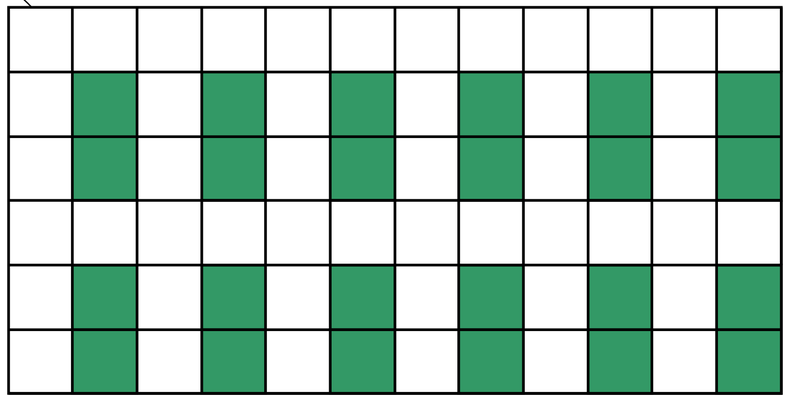
- *Everything is "measured" in the number of elements; 0-based*
- Example 1-dim:
  - Start - starting location of a hyperslab (5)
  - Block - block size (3)



- 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)

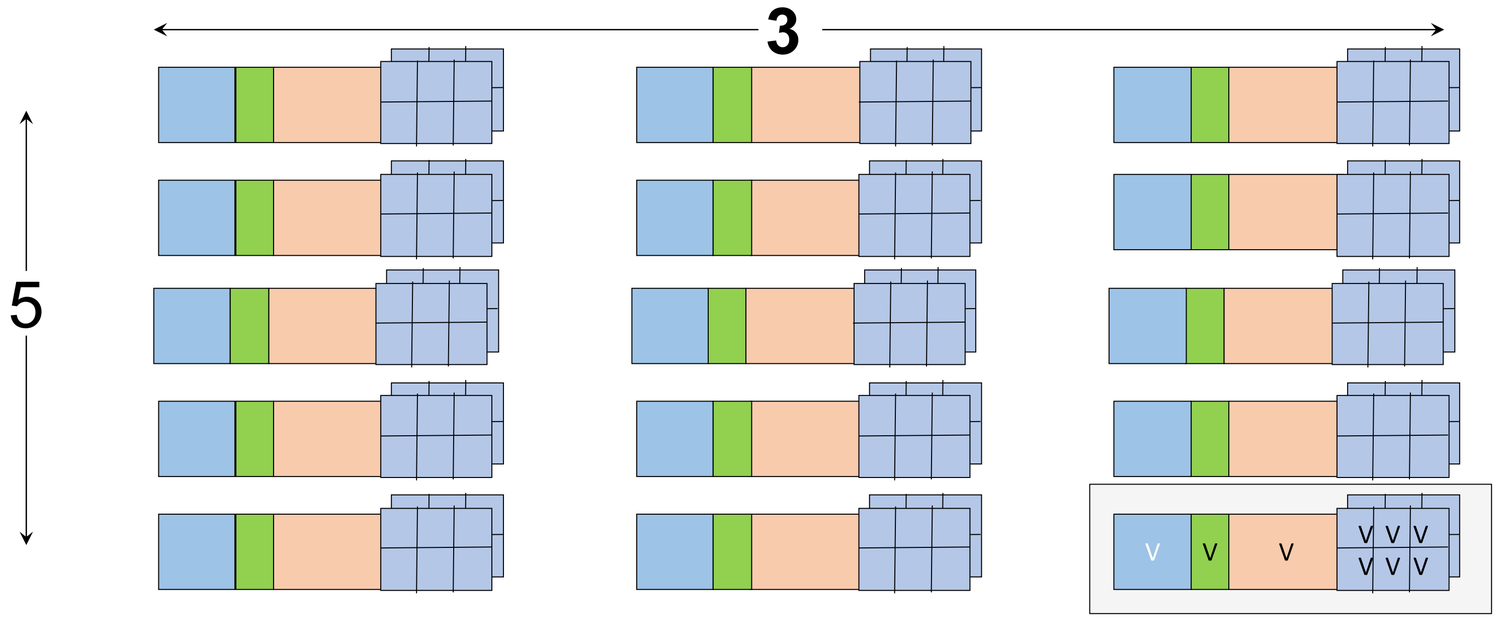


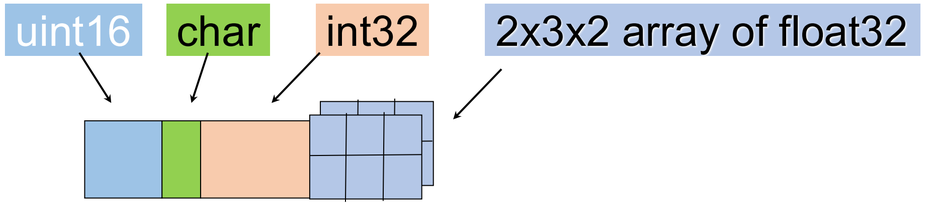
- All other selections are built using set operations

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 symbols to describe atomic datatypes

HDF5 Dataset with Compound Datatype

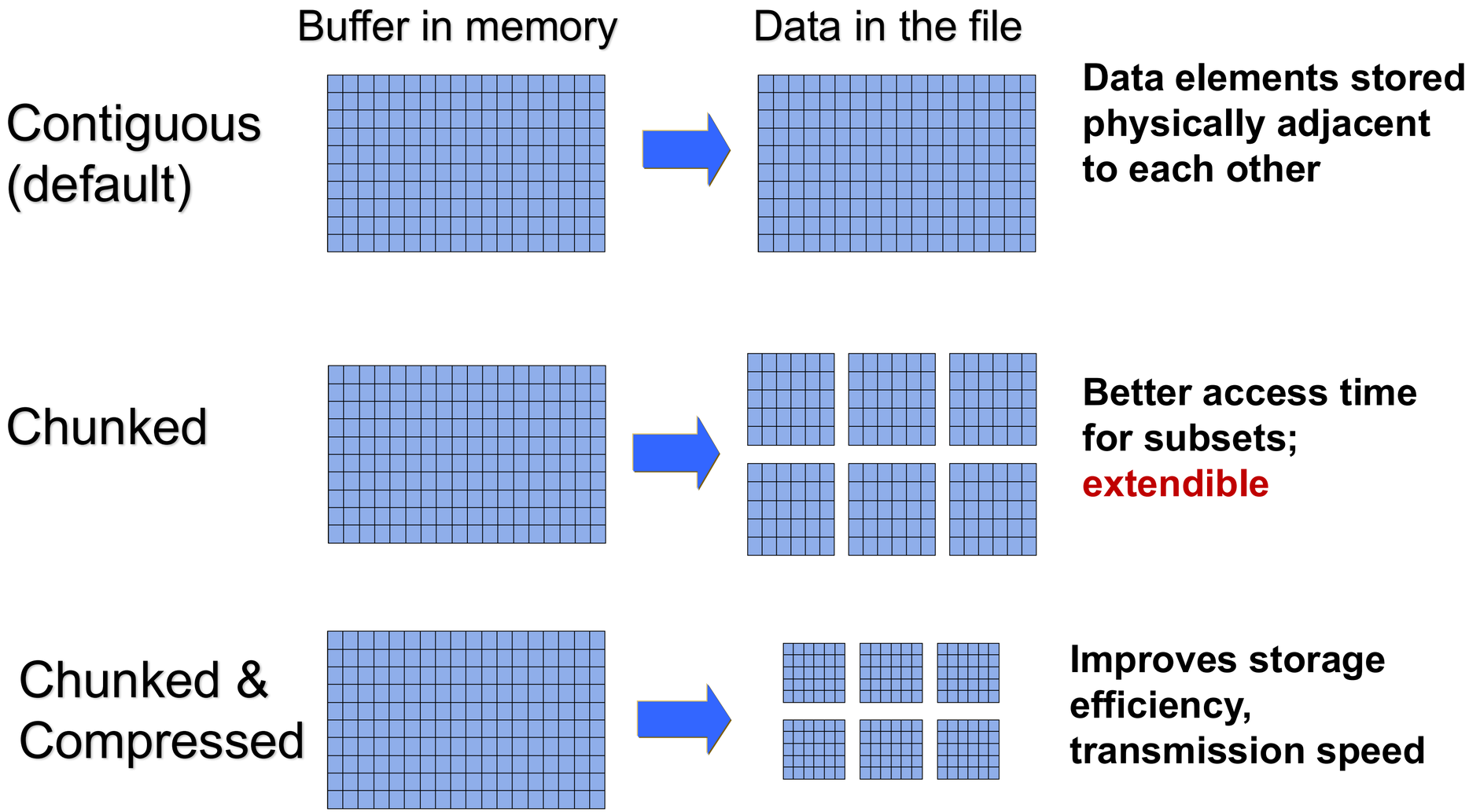




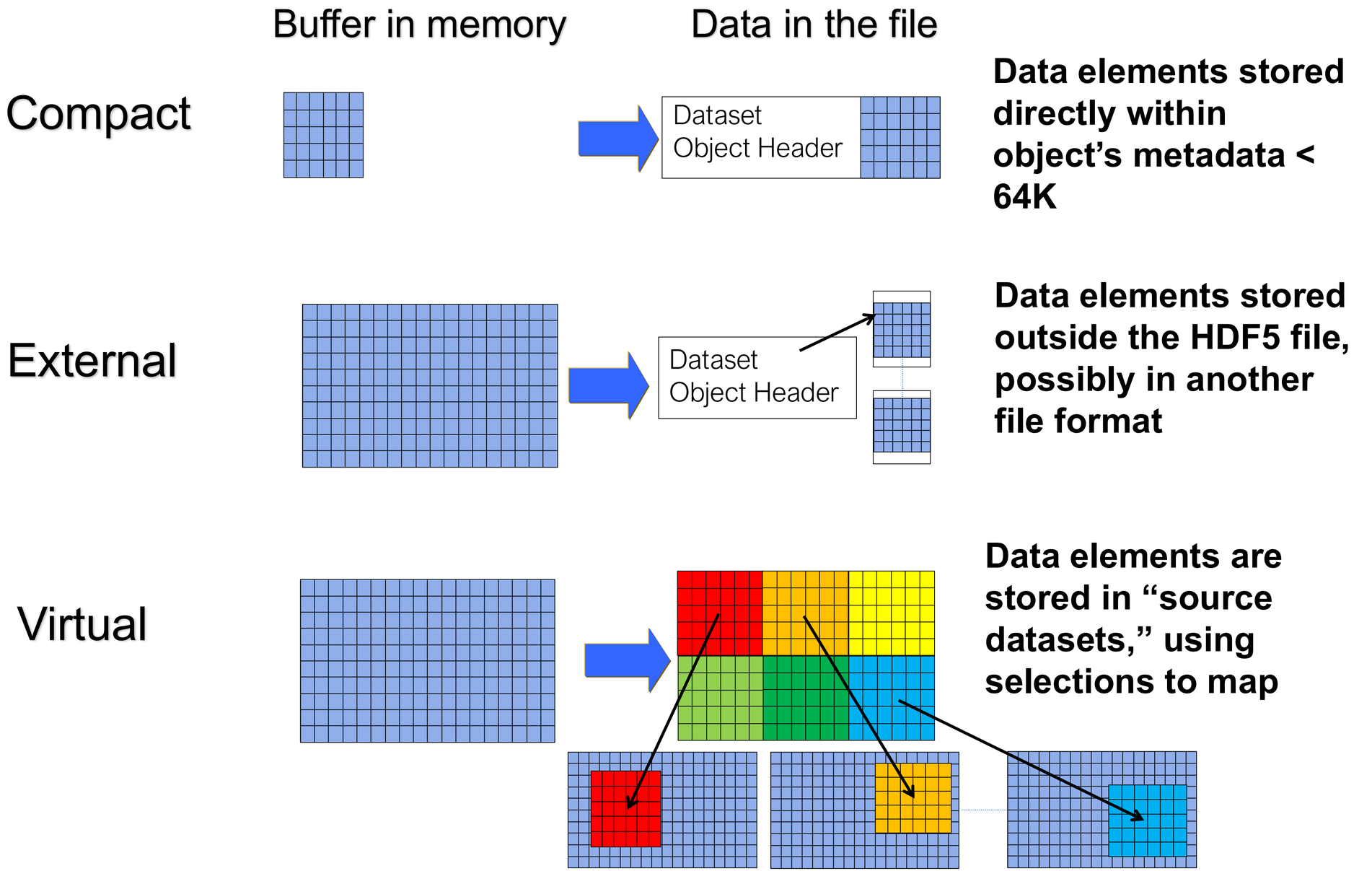
Compound Datatype:

Dataspace: Rank = 2
Dimensions = 5 x 3

How are data elements stored? (1/2)



How are data elements stored? (2/2)



Data elements stored directly within object's metadata < 64K

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

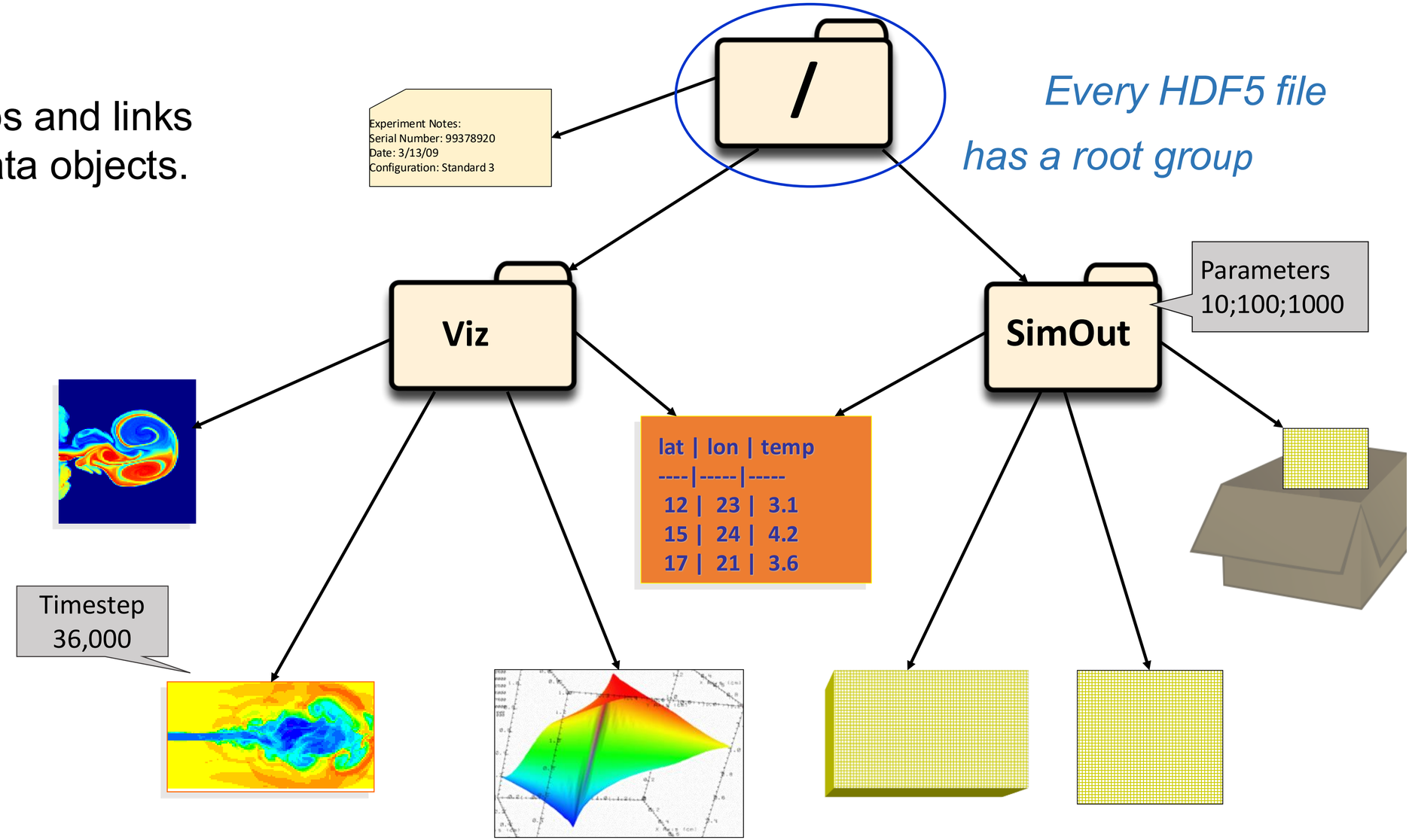
Data elements are stored in "source datasets," using selections to map

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: 2.2.0 (Retired versions ~~1.8, 1.10, 1.12, 1.14~~)

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:

- Python → h5py
- Rust → https://github.com/redclaw-quantum/clawhdf5
- Contemporary C++, including support for MPI I/O
  - https://github.com/ess-dmsc/h5cpp, https://github.com/steven-varga/h5cpp
  - https://github.com/highfive-devs/highfive

Useful Tools For New Users

h5dump

Tool to “dump” or display contents of HDF5 files

Scripts to compile applications:

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

HDFView:

Java browser to view HDF5 file

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

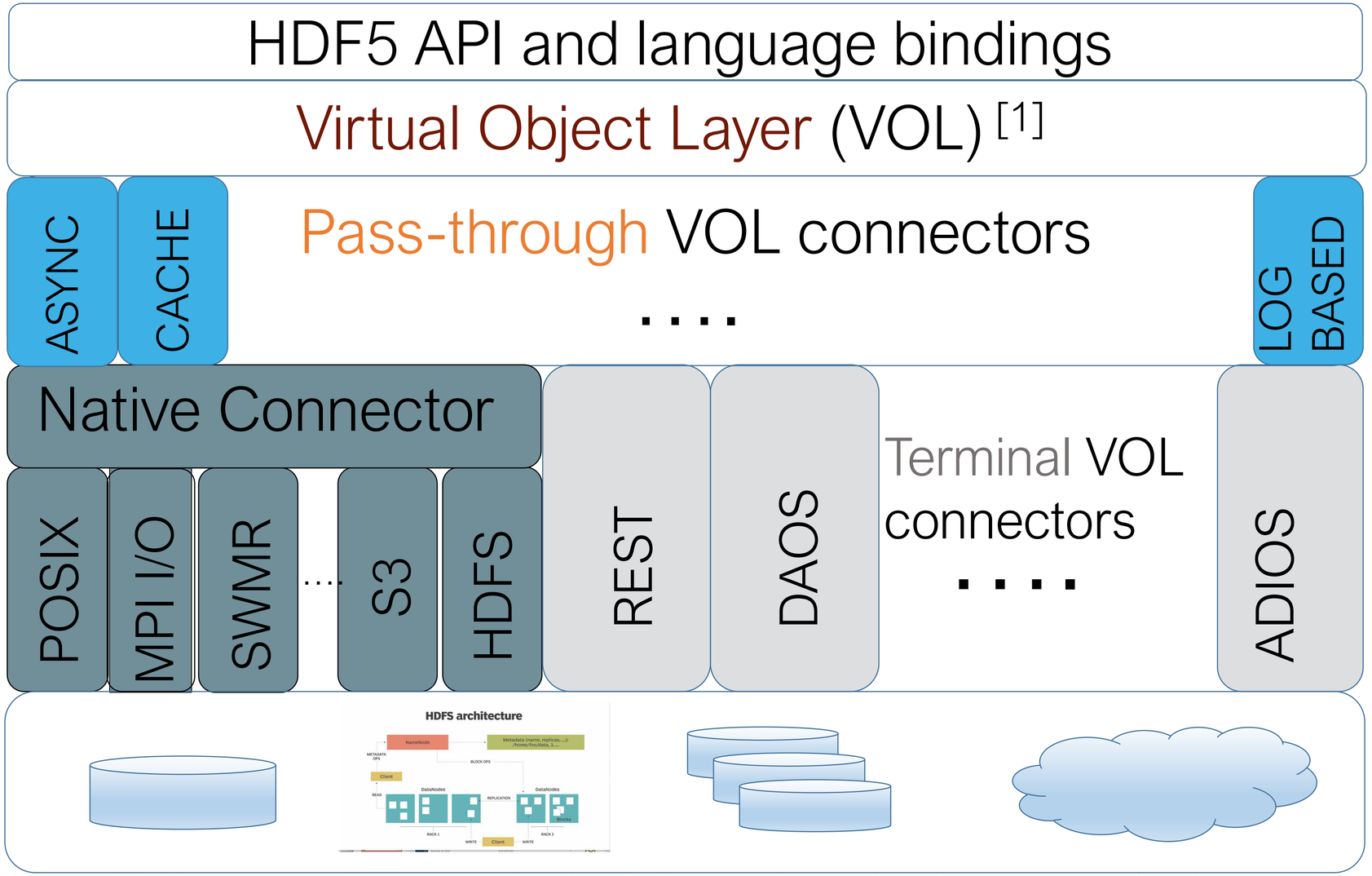
HDF5 Examples (C, Fortran, Java, Python, Matlab, ...)

https://support.hdfgroup.org/documentation/hdf5/latest/_h_d_f5_examples.html

HDF5 Library Architecture (1.12.0 +)

HDF5 Core Library

VFDs



[1] https://portal.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 : Dataset interface *e.g.,* **H5Dread**

H5F : File interface *e.g.,* **H5Fopen**

H5S : data Space 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 optionally 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

RULE #1: You Don't Always Need A DIY HDF5 Implementation

If your scientific field already has a community library built on HDF5, it has likely tuned the chunking, compression, and metadata conventions (i.e., standards) your data needs—better than a from-scratch implementation would.

HDF5

the general-purpose engine

Domain library layer

Encodes your field's schemas, coordinate systems, and access patterns on top of HDF5 — so you inherit years of tuning instead of rebuilding it.

Use HDF5 directly only when your data doesn't fit an existing convention.

1. **EARTH & CLIMATE SCIENCE** — **NetCDF-4**

   CF metadata conventions, coordinate/grid semantics, and dimension scales layered over HDF5.

2. **NEUROSCIENCE** — **NWB / HDMF (PyNWB)**

   Standardized schemas for neurophysiology recordings, built and maintained with HDF5 as the storage backend.

3. **COMPUTATIONAL FLUID DYNAMICS** — **CGNS**

   Mesh and solution-field conventions for CFD codes, so files are portable across solvers.

4. **SINGLE-CELL GENOMICS** — **AnnData**

   Annotated data matrices for scRNA-seq, with HDF5 (.h5ad) as the on-disk format.

MISC RULES: HDF5 Dataset I/O

- Issue large I/O requests
  - At least as large as the file system block size

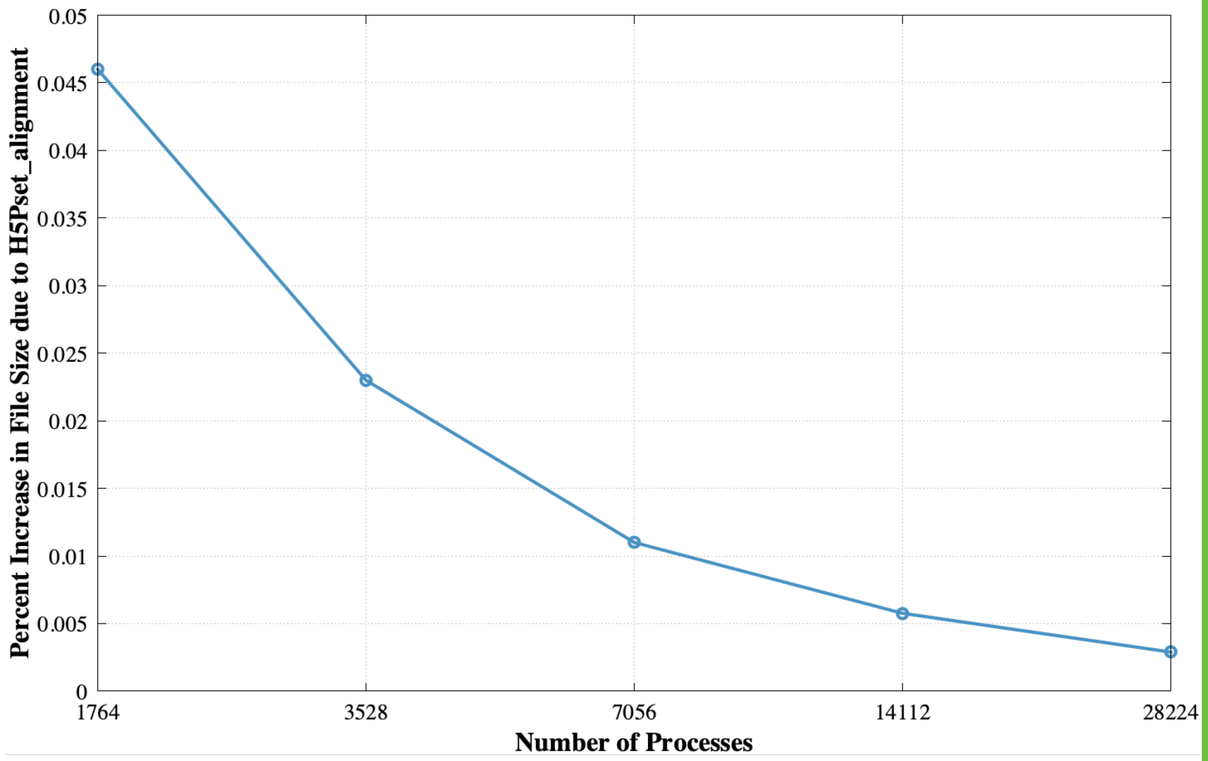
- Avoid **dataspace conversion** ℹ️
  - One-dimensional buffer in memory to a two-dimensional array in the file

- 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()*

ℹ️ Can break collective operations; check what mode was used
H5Pget_mpio_actual_io_mode, and why
H5Pget_mpio_no_collective_cause

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



Fast Reduced-Precision Conversions in HDF5 for ML workflows

Hardware-friendly **bfloat16 / FP8 / FP6 / FP4** conversions via HDF5's public **`H5Tregister`** API — up to **280×** faster*

**Benchmarked on Ryzen 9 9950X3D + RTX 5060 Ti (Blackwell)*

THE PROBLEM

Why the default path is slow

HDF5's general converter checks byte order, format, and exception callbacks **per element** — too generic for the compiler to auto-vectorize.

FLOAT32 — 32 BITS

BFLOAT16 — 16 BITS

bfloat16 keeps the sign, exponent, and top 7 mantissa bits of a float32 — conversion is just a cast and a 16-bit shift.

Same trick applies to FP8 / FP6 / FP4

Register a fast path

```
                H5Tregister(H5T_PERS_HARD,
    "flt_bf16", H5T_NATIVE_FLOAT,
    H5T_FLOAT_BFLOAT16LE,
    conv_float_bfloat16);
```

(register the reverse direction — bfloat16 → float — the same way)

- Call it once at startup and every subsequent H5Tconvert() call uses it
- A plain C loop is auto-vectorized by the compiler (confirmed via objdump).
- Falls back to HDF5's general loop for strided or misaligned buffers.

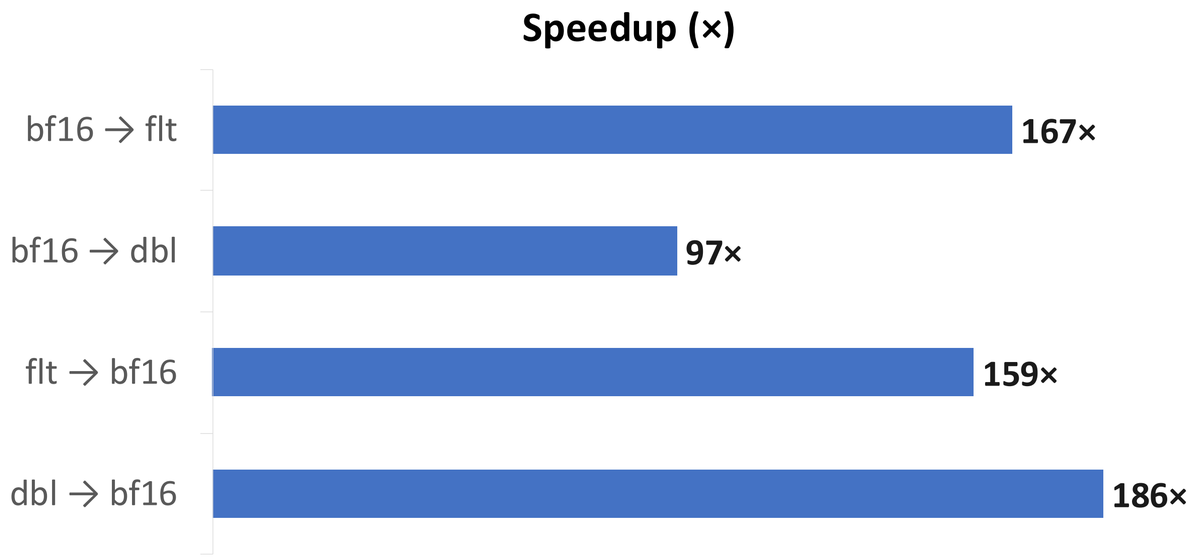
- Performance is not the issue; correctness is….
- Testing random values, tie cases, low-payload NaNs, and zero-length calls

✓ **0 mismatches for float-to-bfloat16 / 1,000,000 vs. software path (used same rounding convention)**

RESULTS

30 × –280 × faster, every precision type

BFLOAT16



134 × –180 × on GCC 15 & Clang 21 alike — bound by memory bandwidth, not compute (shift-and-cast is too cheap to be the bottleneck.

FP8 / FP6 / FP4 (stored in single byte)

130–165 ×
widening (256-entry lookup table)
Trivially vectorizable

37–57 ×
narrowing (exponent rebias + clamp)
Branchy -> vectorizes less cleanly

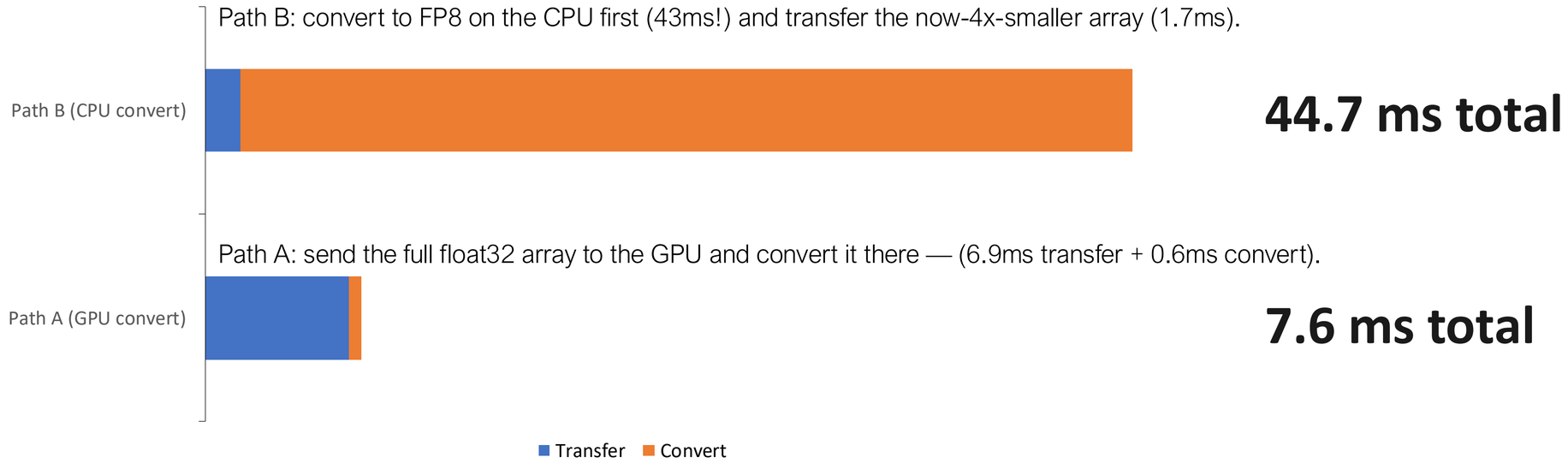
FP6 / FP4 have no x86 hardware path at all — a plain C loop still wins big.

Baseline: HDF5's general loop, same data — all registered via H5Tregister

Blackwell GPUs push it further

float → FP8 on an RTX 5060 Ti: 67 × faster than the CPU fast path (0.64 ms vs. 42.9 ms).

- Branchy exponent-rebias code that limits CPU vectorization isn't a bottleneck for GPU SIMT execution



Path A wins 5.9 × — GPU conversion is negligible next to the transfer. Pipelined: 7.92 ms, near the 6.94 ms PCIe ceiling.

- Slightly counterintuitive to “transfer less data”, but GPU conversion is essentially free next to the PCIe transfer, while CPU narrowing is expensive relative to everything else

CUDA 13.1 · H100+ (FP8) / RTX 50-series+ (FP6, FP4)

RECOMMENDATIONS

What should scientists use?

WHICH TYPE

- **bfloat16:** safe default, ~0.4% error (mantissa precision) -> file size and I/0 time ~halved
- **FP8:** ML inference and training only, needs scaled data (or range-analyzed)
- **FP6 / FP4:** GPU-only ML formats, no CPU hardware path

HOW TO CONVERT

- **CPU:** register once at startup (H5Tregister)
- **GPU:** transfer float32, convert on device, never shrink on the CPU first
- **Round trips:** pipeline multi-chunk round trips across CUDA streams

BOTTOM LINE

30–280 ×

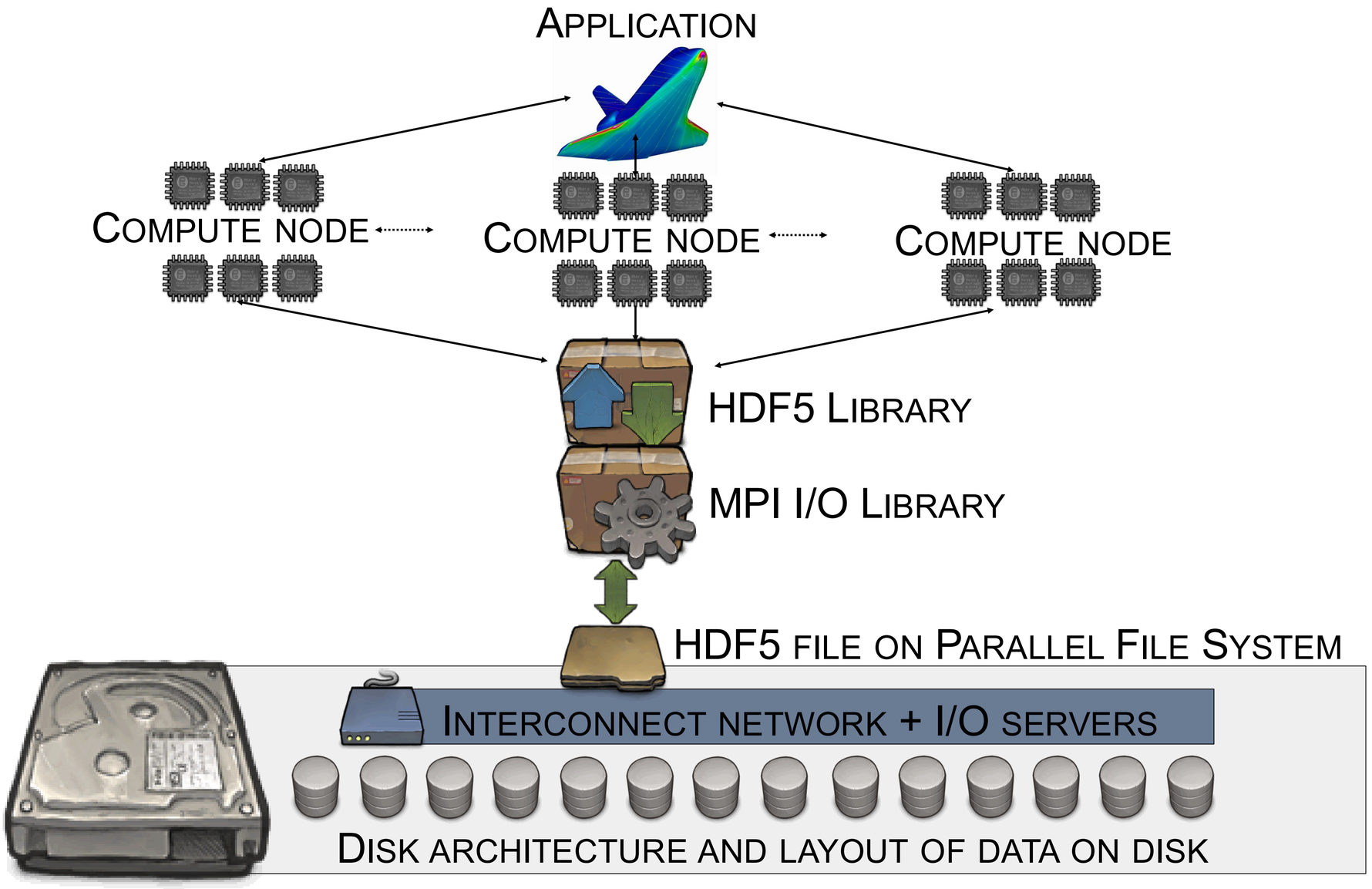
CPU, public API

67 ×

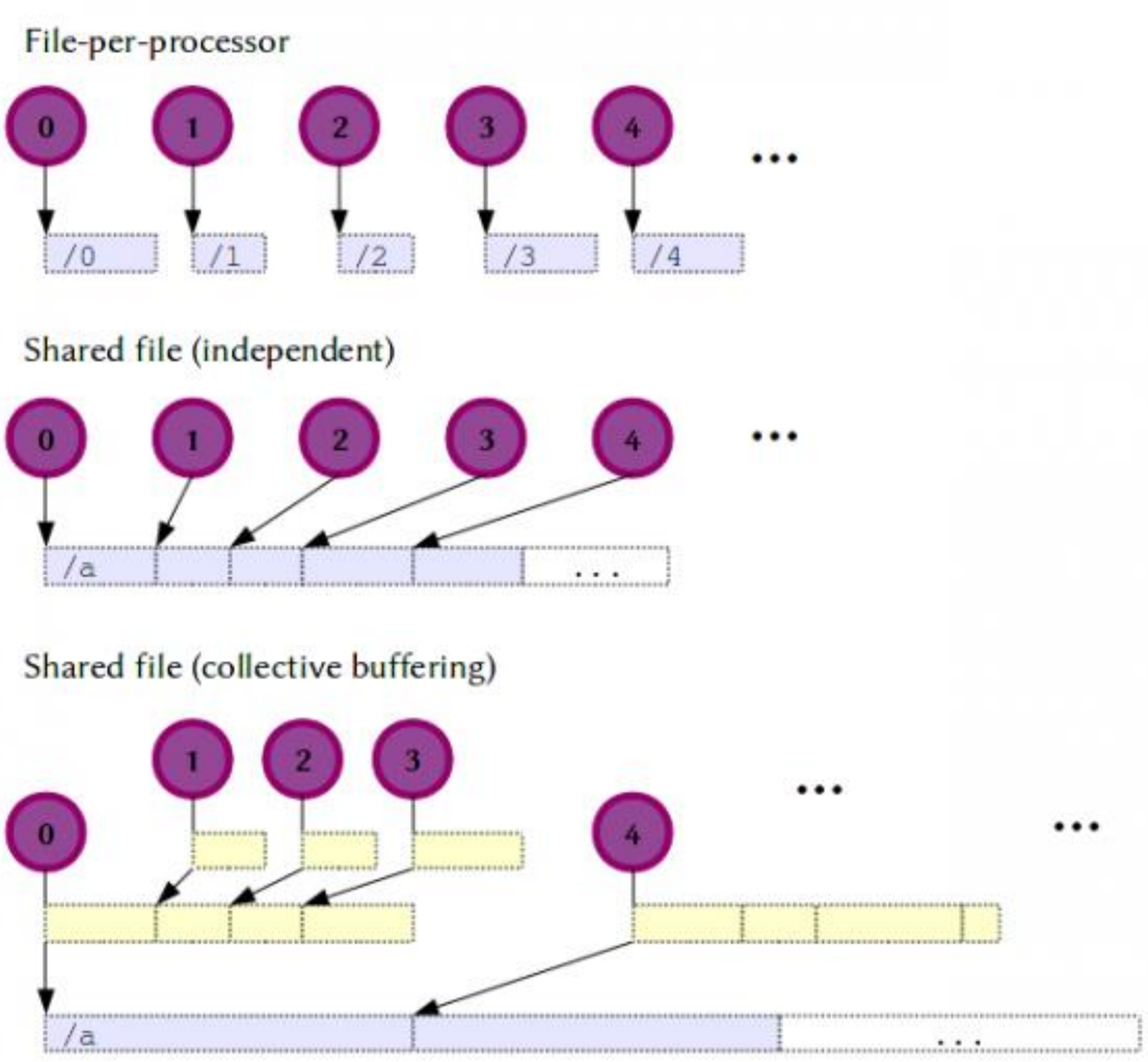
Further improvement on GPU

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”

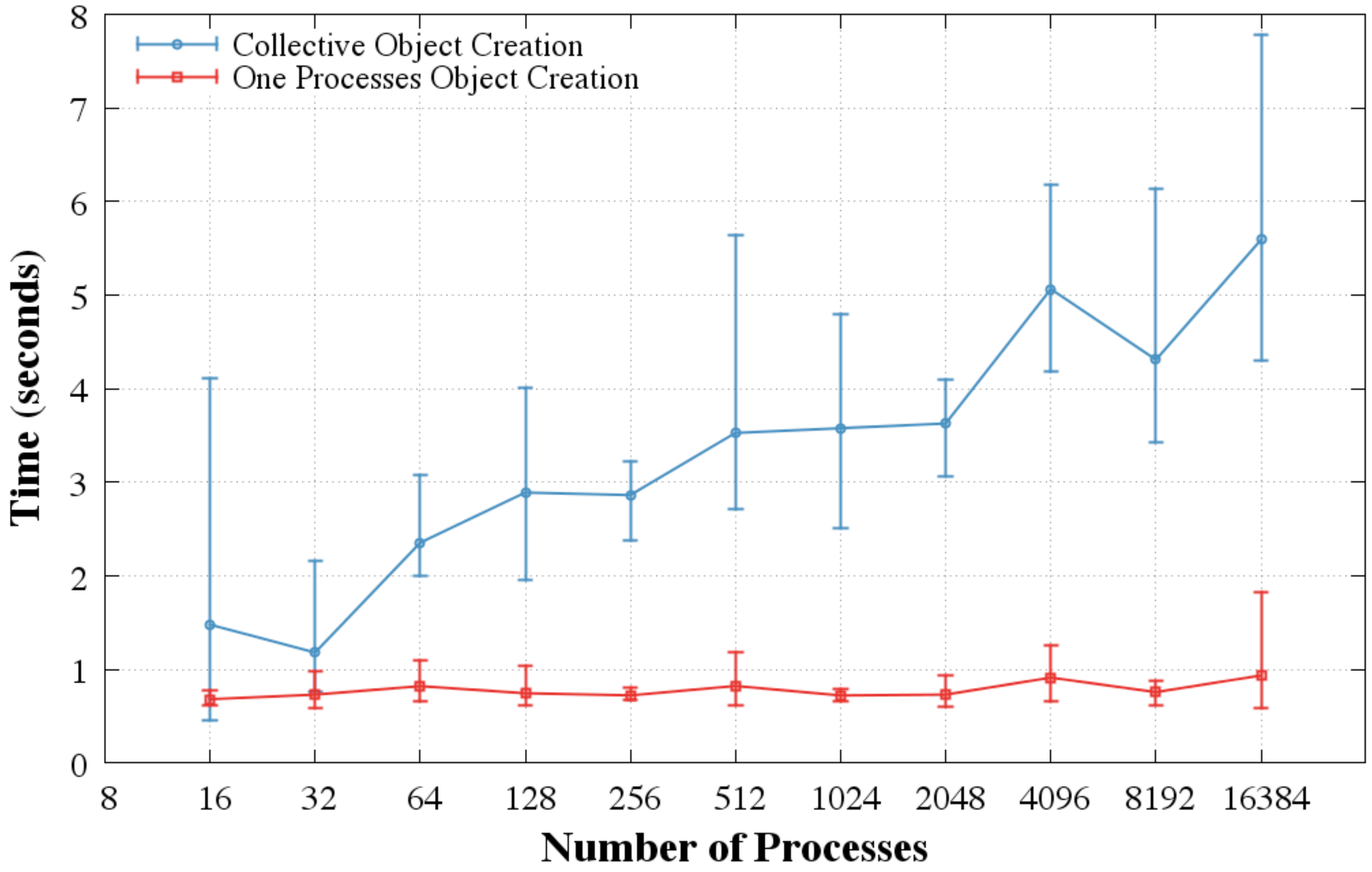
Parallel HDF5 Schema

- 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!

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

  - 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.

Object Creation (Collective vs. Single Process)



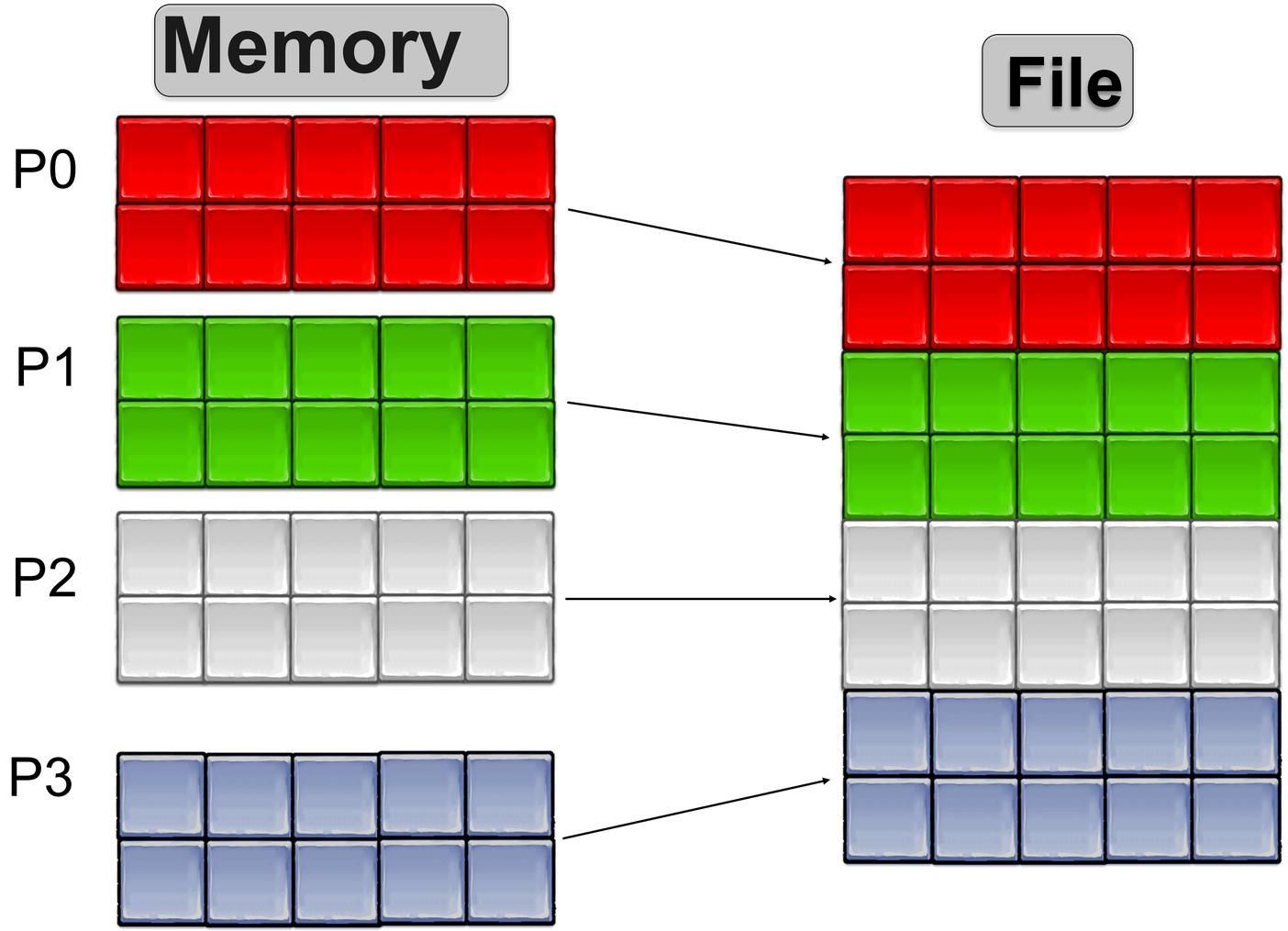
Collective vs. Independent Operations

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

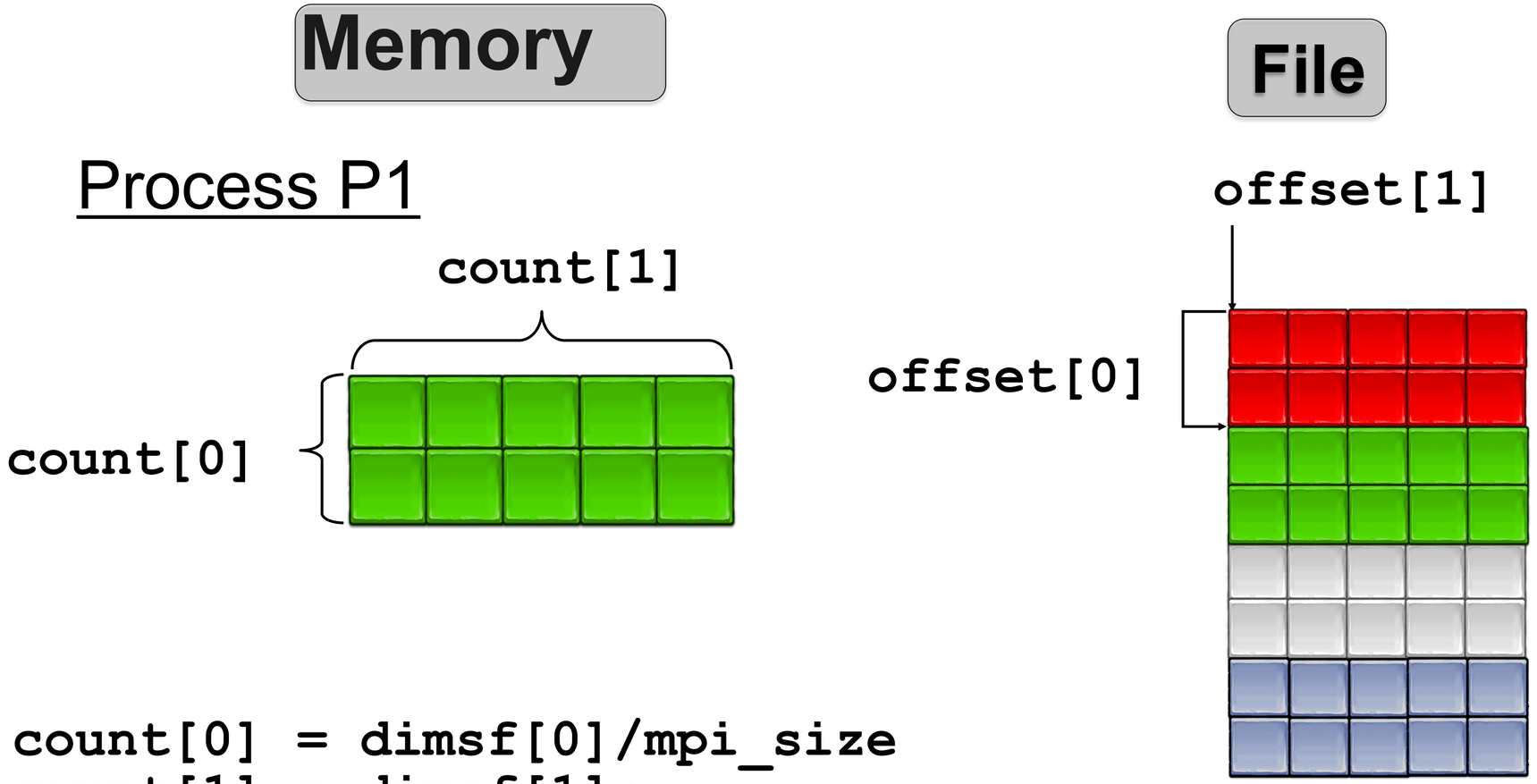
	Process1	Process2	
	**call A(); call B();**	**call A(); call B();**	**…CORRECT**
	**call A(); call B();**	**call B(); call A();**	**…WRONG**

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

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*

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

C Example: Collective write and read

```
 95  /*
 96   * Create property list for collective dataset write.
 97   */
 98  plist_id = H5Pcreate(H5P_DATASET_XFER);
->99  H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);
100
101  status = H5Dwrite(dset_id, H5T_NATIVE_INT,
102                    memspace, filespace, plist_id, data);
103  /*
104   * Collective dataset read.
105   */
106
->107 status = H5Dread(dset_id, H5T_NATIVE_INT,
108                      memspace, filespace, plist_id, data);
109
```

Writing by rows: *Output of h5dump*

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

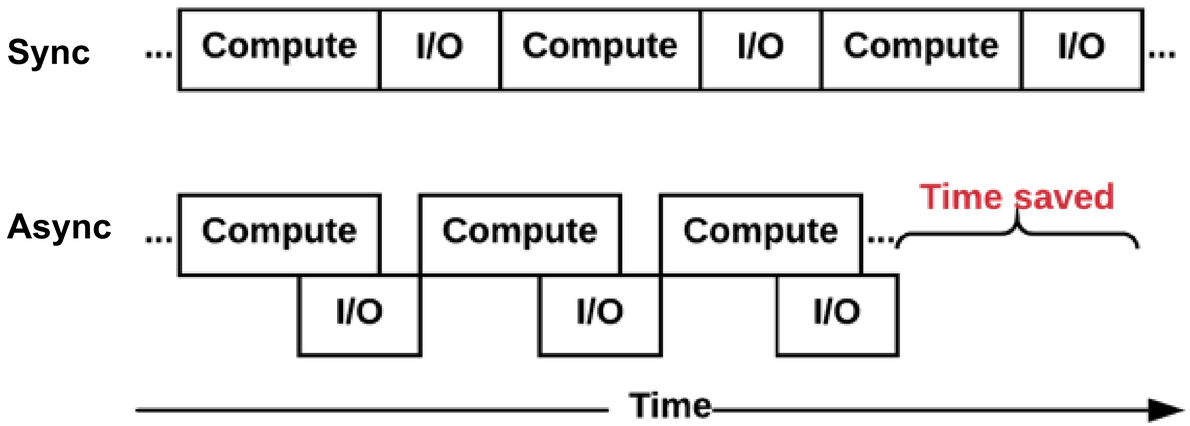
General HDF5 Best Practices and Case Studies for Parallel Performance

Strongly Recommended Options

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

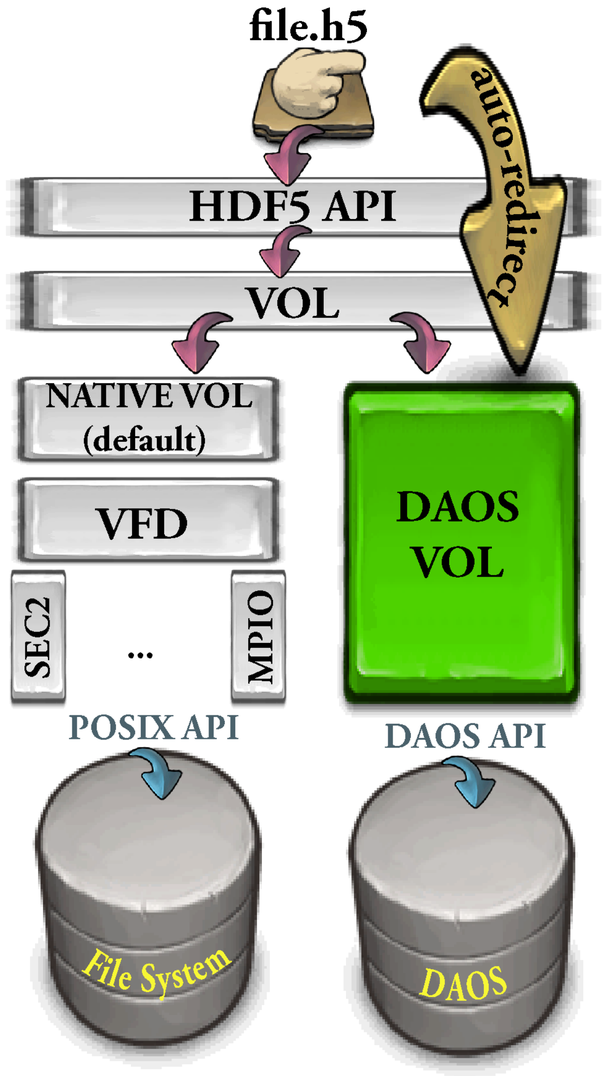
- Set HDF5 to never fill chunks (H5Pset_fill_time with H5D_FILL_TIME_NEVER)

Features: Asynchronous I/O



- Allows asynchronous operations for HDF5 applications:
  - Applications use the *_async* versions for the **H5** APIs
  - Return “request tokens” to applications to track I/O tasks.
- Requires a VOL (async or DAOS) which supports asynchronous I/O, otherwise defaults to synchronous I/O.

Feature: DAOS VOL Connector

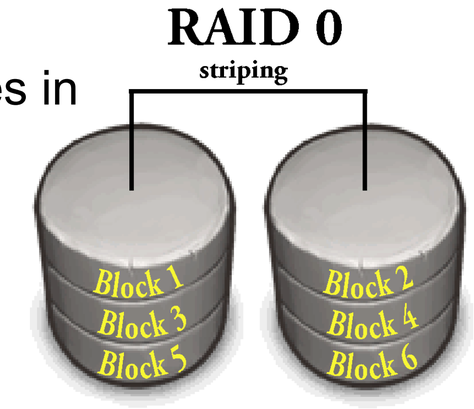


- HDF5 VOL connector for I/O to Distributed Asynchronous Object Storage (DAOS)

- Deployed at ANL.
- Minimal code changes needed to use, enable via environment variables or through HDF5 APIs.
- HDF5 tools are supported
  - h5dump, h5ls, h5diff, h5repack, h5copy, etc.
- Supports async I/O

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

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

- HDF5 stub file
  - Appears as a normal HDF5 file; only contains HDF5 superblock information and 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
```

THANK YOU!

Questions & Comments?

Need Help?

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

HDF Helpdesk – help@hdfgroup.org

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.