Parallel HDF5 Programming Interface

Albert Cheng, Elena Pourmal

NCSA

Outline

- Overview of Parallel HDF5 Design
- Setting up Parallel Environment
- Creating and Accessing a File
- Creating and Accessing a Dataset
- Writing and Reading Hyperslabs

PHDF5 Requirements

- **PHDF5 files compatible with serial HDF5 files**
  - **Shareable between different serial or parallel platforms**
- **Single file image to all processes**
  - **One file per process design is undesirable**
    - **Expensive post processing**
    - **Not useable by different number of processes**
- **Standard parallel I/O interface**
  - **Must be portable to different platforms**

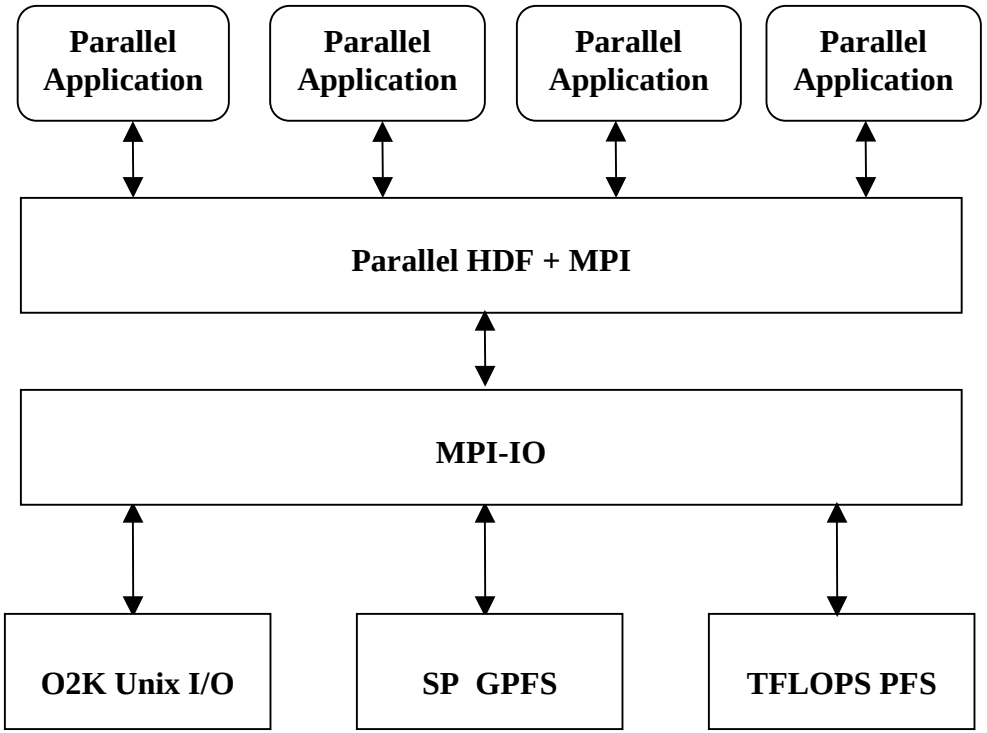
PHDF5
Initial Target

- **Support for MPI programming**
- **Not for shared memory programming**
  - **Threads**
  - **OpenMP**
- **Has some experiments with**
  - **Thread-safe support for Pthreads**
  - **OpenMP if called “correctly”**

Implementation Requirements

- **No use of Threads**
  - **Not commonly supported (1998)**
- **No reserved process**
  - **May interfere with parallel algorithms**
- **No spawn process**
  - **Not commonly supported even now**

PHDF5 Implementation Layers



User Applications

HDF library

Parallel I/O layer

Parallel File systems

Programming Restrictions

- **Most PHDF5 APIs are collective**
- **MPI definition of collective: All processes of the communicator must participate in the right order.**
- **PHDF5 opens a parallel file with a communicator**
  - **Returns a file-handle**
  - **Future access to the file via the file-handle**
  - **All processes must participate in collective PHDF5 APIs**
  - **Different files can be opened via different communicators**

Examples of PHDF5 API

- **Examples of PHDF5 collective API**
  - **File operations: H5Fcreate, H5Fopen, H5Fclose**
  - **Objects creation: H5Dcreate, H5Dopen, H5Dclose**
  - **Objects structure: H5Dextend (increase dimension sizes)**
- **Array data transfer can be collective or independent**
  - **Dataset operations: H5Dwrite, H5Dread**

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 write to the same data array**
  - **Each process writes to individual data array**

PHDF5 API Languages

- C and F90 language interfaces
- Platforms supported: IBM SP2 and SP3, Intel TFLOPS, SGI Origin 2000, Mpich.

Creating and Accessing a File
Programming model

- **HDF5 uses access template object to control the file access mechanism**
- **General model to access HDF5 file in parallel:**
  - *Setup access template*
  - *Open File*
  - *Close File*

Setup access template

Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information

C:

```
herr_t H5Pset_fapl_mpio(hid_t plist_id,
       MPI_Comm comm,  MPI_Info info);
```

F90:

```
h5pset_fapl_mpio_f(plist_id, comm, info);

integer(hid_t) :: plist_id
integer        :: comm, info
```

C Example
Parallel File Create

```
23      comm = MPI_COMM_WORLD;
24      info = MPI_INFO_NULL;
26      /*
27       * Initialize MPI
28       */
29      MPI_Init(&argc, &argv);
33      /*
34       * Set up file access property list for MPI-IO access
35       */
36      plist_id = H5Pcreate(H5P_FILE_ACCESS);
37      H5Pset_fapl_mpio(plist_id, comm, info);
38
42      file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC,
           H5P_DEFAULT, plist_id);
49      /*
50       * Close the file.
51       */
52      H5Fclose(file_id);
54      MPI_Finalize();
```

F90 Example
Parallel File Create

```
23 comm = MPI_COMM_WORLD
24 info = MPI_INFO_NULL
26 CALL MPI_INIT(mpierror)
29 !
30 ! Initialize FORTRAN predefined datatypes
32 CALL h5open_f(error)
34 !
35 ! Setup file access property list for MPI-IO access.
37 CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, error)
38 CALL h5pset_fapl_mpio_f(plist_id, comm, info, error)
40 !
41 ! Create the file collectively.
43 CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id,
       error, access_prp = plist_id)
45 !
46 ! Close the file.
49 CALL h5fclose_f(file_id, error)
51 !
52 ! Close FORTRAN interface
54 CALL h5close_f(error)
56 CALL MPI_FINALIZE(mpierror)
```

Creating and Opening Dataset

- **All processes of the MPI communicator open a dataset by a *collective call***
  - **`C: H5Dcreate or H5Dopen; H5Dclose`**
  - **`F90: h5dfreate_f or h5dopen_f; h5dclose_f`**
- **All processes of the MPI communicator extend dataset with unlimited dimensions by a *collective call***
  - **`C: H5Dextend`**
  - **`F90: h5dextend_f`**

C Example
Parallel Dataset Create

```
56  file_id = H5Fcreate(…);
57  /*
58   * Create the dataspace for the dataset.
59   */
60  dimsf[0] = NX;
61  dimsf[1] = NY;
62  filespace = H5Screate_simple(RANK, dimsf, NULL);
63
64  /*
65   * Create the dataset with default properties collective.
66   */
67  dset_id = H5Dcreate(file_id, “dataset1”, H5T_NATIVE_INT,
68                      filespace, H5P_DEFAULT);

70  H5Dclose(dset_id);
71  /*
72   * Close the file.
73   */
74  H5Fclose(file_id);
```

F90 Example
Parallel Dataset Create

```
43 CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id,
       error, access_prp = plist_id)
73 CALL h5screate_simple_f(rank, dimsf, filespace, error)
76 !
77 ! Create the dataset with default properties.
78 !
79 CALL h5dcreate_f(file_id, “dataset1”, H5T_NATIVE_INTEGER,
                     filespace, dset_id, error)

90 !
91 ! Close the dataset.
92 CALL h5dclose_f(dset_id, error)
93 !
94 ! Close the file.
95 CALL h5fclose_f(file_id, error)
```

Accessing a Dataset

- All processes that have opened dataset may do collective I/O
- Each process may do independent and arbitrary number of data I/O access calls
  - `C:  H5Dwrite and H5Dread`
  - `F90: h5dwrite_f and h5dread_f`

Accessing a Dataset
Programming model

- **Create and set dataset transfer property**
  - **C: `H5Pset_dxpl_mpio`**
    - **`H5FD_MPIO_COLLECTIVE`**
    - **`H5FD_MPIO_INDEPENDENT`**
  - **F90: `h5pset_dxpl_mpio_f`**
    - **`H5FD_MPIO_COLLECTIVE_F`**
    - **`H5FD_MPIO_INDEPENDENT_F`**
- **Access dataset with the defined transfer property**

C Example: Collective write

```
 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);
```

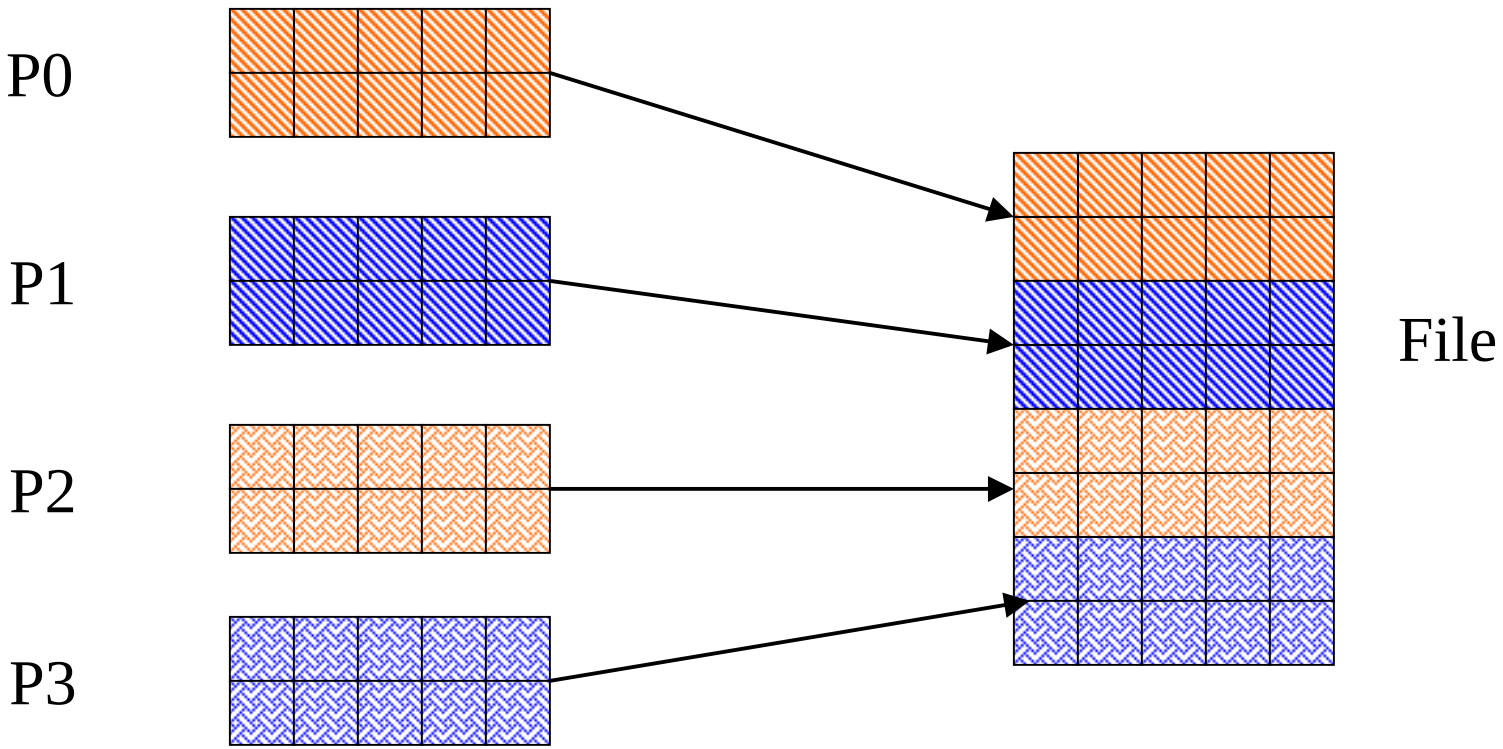
F90 Example: Collective write

```
87  !
88  ! Create property list for collective dataset write
89  !
90  CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, error)
91  CALL h5pset_dxpl_mpio_f(plist_id, &
                            H5FD_MPIO_COLLECTIVE_F, error)
92
93  !
94  ! Write the dataset collectively.
95  !
96  CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, data, &
                    error, &
                    file_space_id = filespace, &
                    mem_space_id = memspace, &
                    xfer_prp = plist_id)
```

Writing and Reading Hyperslabs
Programming model

- **Each process defines memory and file hyperslabs**
- **Each process executes partial write/read call**
  - **Collective calls**
  - **Independent calls**

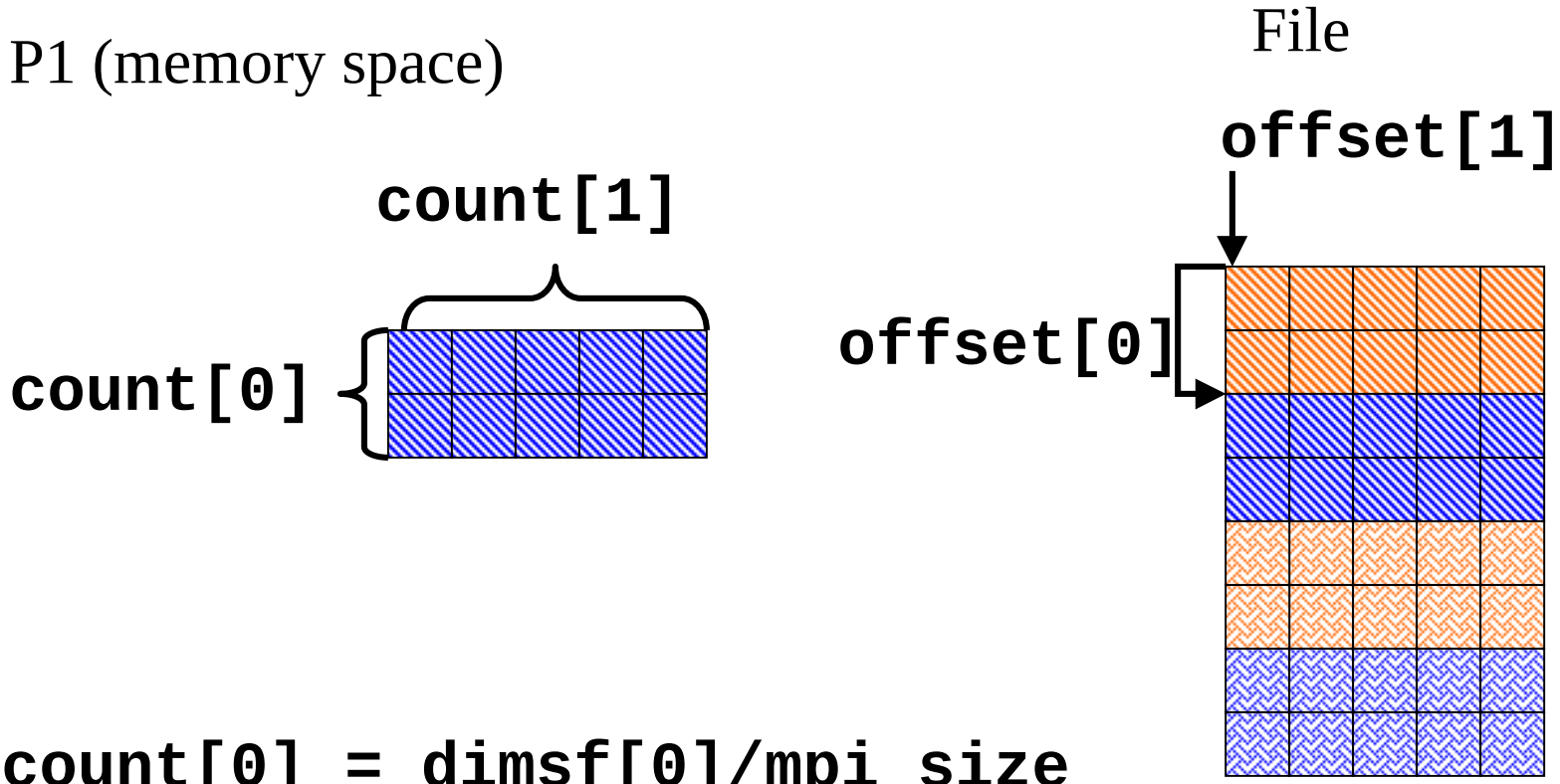
Hyperslab Example 1
Writing dataset by rows



Writing by rows
Output of h5dump utility

```
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
      }
   }
}
}
```

Example 1
Writing dataset by rows

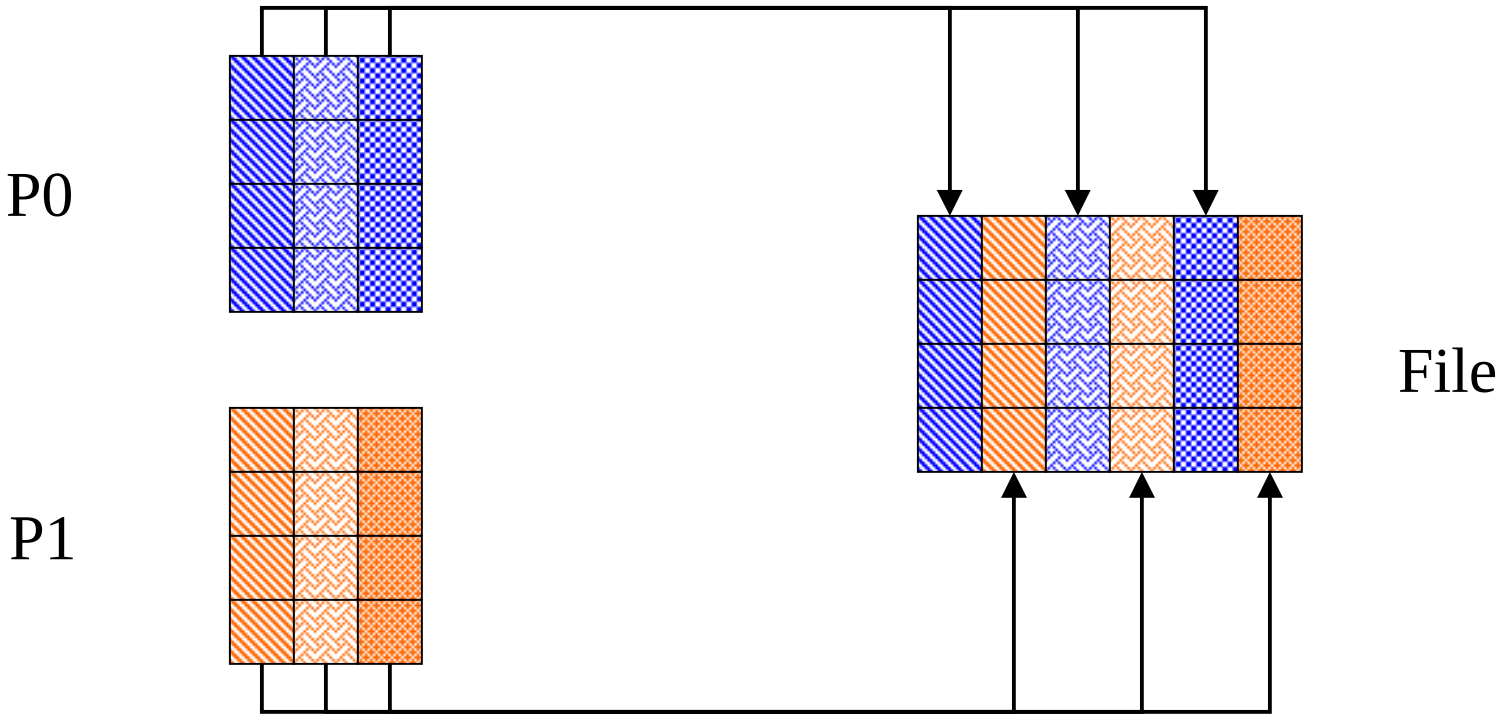



```
count[0] = dimsf[0]/mpi_size
count[1] = dimsf[1];
offset[0] = mpi_rank * count[0];
offset[1] = 0;
```

C Example 1

```
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);
```

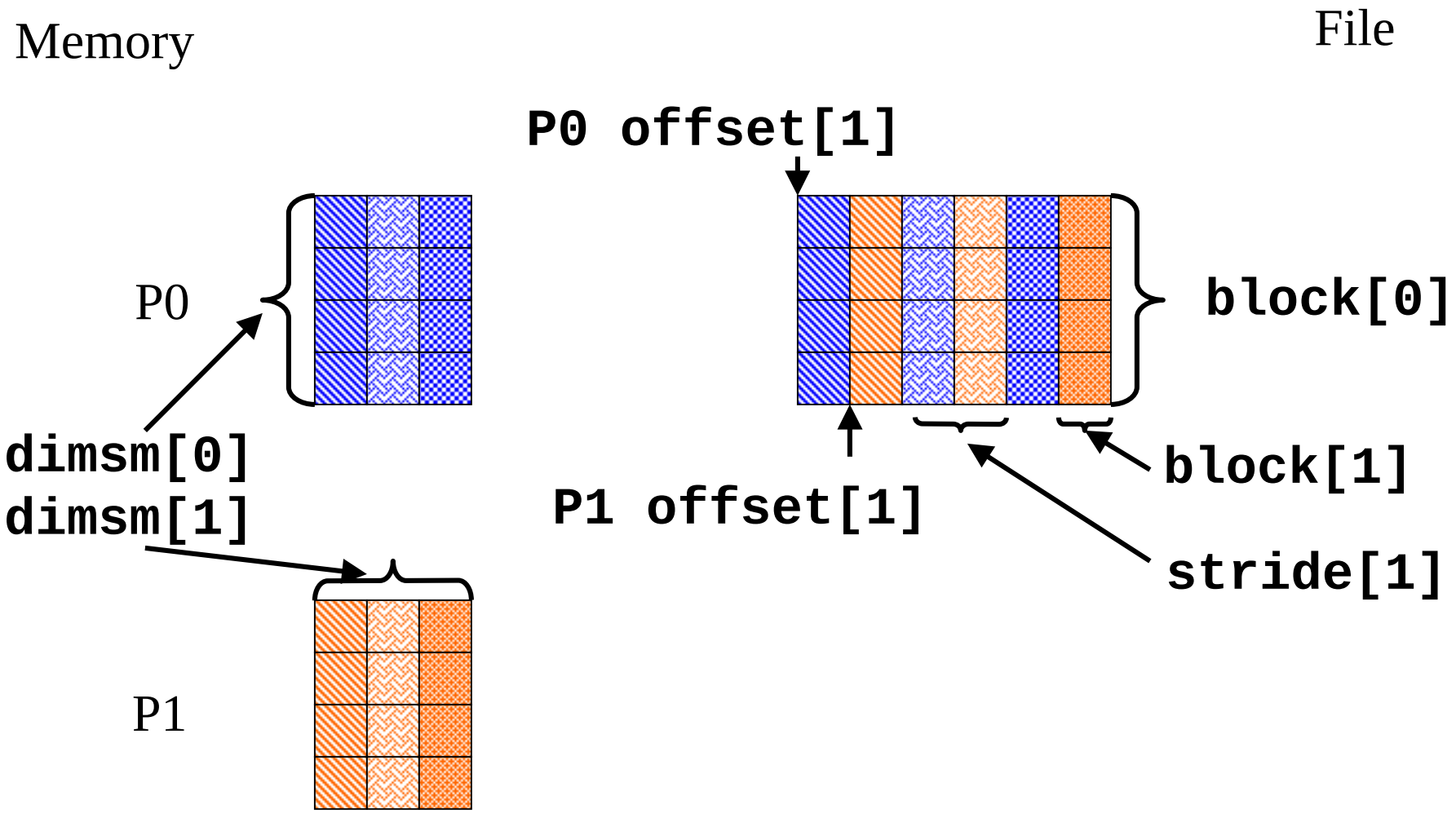
Hyperslab Example 2
Writing dataset by columns



Writing by columns
Output of h5dump utility

```
HDF5 "SDS_col.h5" {
GROUP "/" {
   DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 8, 6 ) / ( 8, 6 ) }
      DATA {
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200,
         1, 2, 10, 20, 100, 200
      }
   }
}
}
```

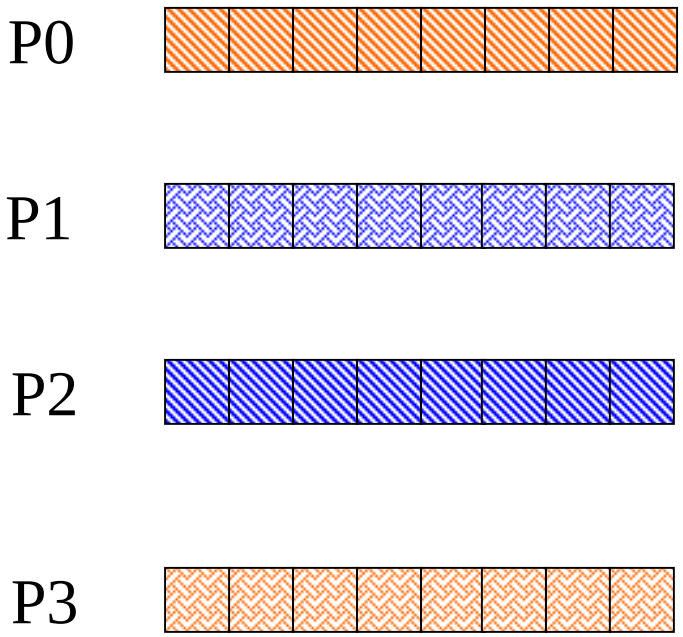
Example 2
Writing dataset by column

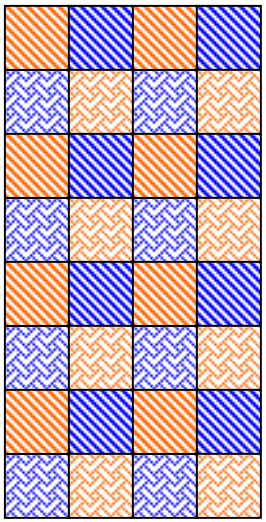


C Example 2

```
85          /*
86           * Each process defines hyperslab in
           * the file
88           */
89          count[0] = 1;
90          count[1] = dimsm[1];
91          offset[0] = 0;
92          offset[1] = mpi_rank;
93          stride[0] = 1;
94          stride[1] = 2;
95          block[0] = dimsf[0];
96          block[1] = 1;
97
98          /*
99           * Each process selects hyperslab.
100          */
101         filespace = H5Dget_space(dset_id);
102         H5Sselect_hyperslab(filespace,
              H5S_SELECT_SET, offset, stride,
              count, block);
```

Hyperslab Example 3
Writing dataset by pattern





Writing by Pattern
Output of h5dump utility

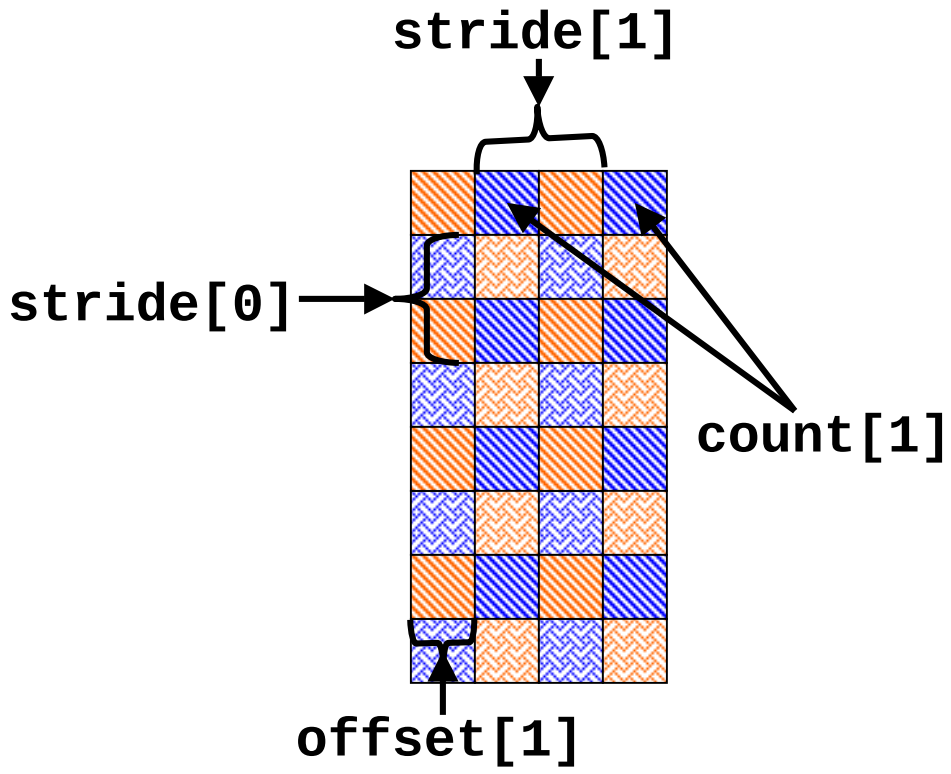
```
HDF5 "SDS_pat.h5" {
GROUP "/" {
   DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
      DATA {
         1, 3, 1, 3,
         2, 4, 2, 4,
         1, 3, 1, 3,
         2, 4, 2, 4,
         1, 3, 1, 3,
         2, 4, 2, 4,
         1, 3, 1, 3,
  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

Example 3
Writing dataset by pattern

Memory

File

P2

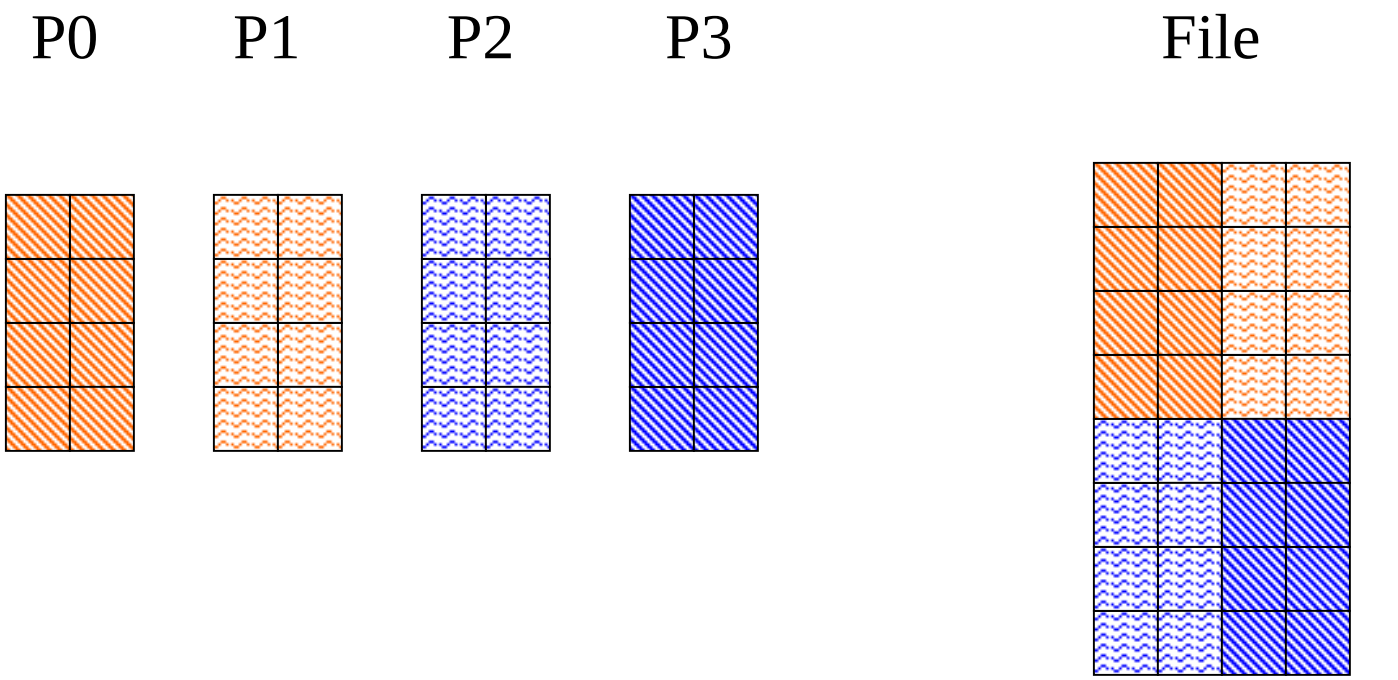



```
offset[0] = 0;
offset[1] = 1;
count[0] = 4;
count[1] = 2;
stride[0] = 2;
stride[1] = 2;
```

C Example 3
Writing by pattern

```
 90      /* Each process defines dataset in memory and
           * writes it to the hyperslab
 91        * in the file.
 92        */
 93      count[0] = 4;
 94      count[1] = 2;
 95      stride[0] = 2;
 96      stride[1] = 2;
 97      if(mpi_rank == 0) {
 98         offset[0] = 0;
 99         offset[1] = 0;
100      }
101      if(mpi_rank == 1) {
102         offset[0] = 1;
103         offset[1] = 0;
104      }
105      if(mpi_rank == 2) {
106         offset[0] = 0;
107         offset[1] = 1;
108      }
109      if(mpi_rank == 3) {
110         offset[0] = 1;
111         offset[1] = 1;
112      }
```

Hyperslab Example 4
Writing dataset by chunks



Writing by Chunks
Output of h5dump utility

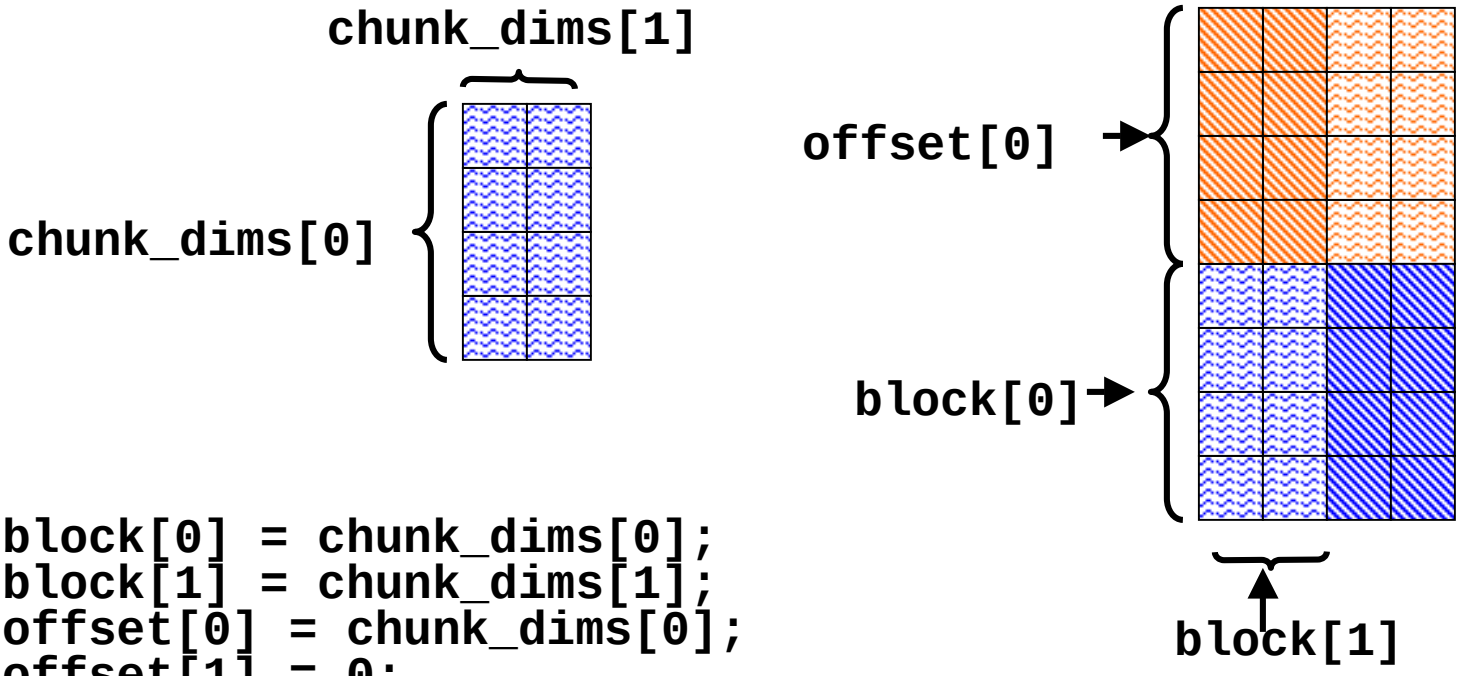
```
HDF5 "SDS_chnk.h5" {
GROUP "/" {
   DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
      DATA {
         1, 1, 2, 2,
         1, 1, 2, 2,
         1, 1, 2, 2,
         1, 1, 2, 2,
         3, 3, 4, 4,
         3, 3, 4, 4,
         3, 3, 4, 4,
 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

Example 4
Writing dataset by chunks

Memory

File

P2




```
block[0] = chunk_dims[0];
block[1] = chunk_dims[1];
offset[0] = chunk_dims[0];
offset[1] = 0;
```

C Example 4
Writing by chunks

```
 97      count[0] = 1;
 98      count[1] = 1 ;
 99      stride[0] = 1;
100      stride[1] = 1;
101      block[0] = chunk_dims[0];
102      block[1] = chunk_dims[1];
103      if(mpi_rank == 0) {
104         offset[0] = 0;
105         offset[1] = 0;
106      }
107      if(mpi_rank == 1) {
108         offset[0] = 0;
109         offset[1] = chunk_dims[1];
110      }
111      if(mpi_rank == 2) {
112         offset[0] = chunk_dims[0];
113         offset[1] = 0;
114      }
115      if(mpi_rank == 3) {
116         offset[0] = chunk_dims[0];
117         offset[1] = chunk_dims[1];
118      }
```

HDF5 Tuning knobs

Albert Cheng

NCSA

File Level Knobs

- **H5Pset_meta_block_size**
- **H5Pset_alignment**
- **H5Pset_fapl_split**
- **H5Pset_cache**
- **H5Pset_fapl_mpio**

H5Pset_meta_block_size

- Sets the minimum metadata block size allocated for metadata aggregation. The larger the size, the fewer the number of small data objects in the file.
- The aggregated block of metadata is usually written in a single write action and always in a contiguous block, potentially significantly improving library and application performance.
- Default is 2KB

H5Pset_alignment

- Sets the *alignment* properties of a file access property list so that any file object greater than or equal in size to *threshold* bytes will be aligned on an address which is a multiple of alignment. This makes significant improvement to file systems that are sensitive to data block alignments.
- Default values for threshold and alignment are one, implying no alignment. Generally the default values will result in the best performance for single-process access to the file. For MPI-IO and other parallel systems, choose an alignment which is a multiple of the disk block size.

H5Pset_fapl_split

- Sets file driver to store metadata and raw data in two separate files, metadata and raw data files.
- Significant I/O improvement if the metadata file is stored in Unix file systems (good for small I/O) while the raw data file is stored in Parallel file systems (good for large I/O).
- Default is no split.

H5Pset_cache

- **Sets:**
  - **the number of elements (objects) in the meta data cache**
  - **the number of elements, the total number of bytes, and the preemption policy value in the raw data chunk cache**
- **The right values depend on the individual application access pattern.**
- **Default for preemption value is 0.75**

H5Pset_fapl_mpio

- MPI-IO hints can be passed to the MPI-IO layer via the Info parameter.
- E.g., telling Romio to use 2-phases I/O speeds up collective I/O in the ASCI Red machine.

Data Transfer Level Knobs

- **H5Pset_buffer**
- **H5Pset_sieve_buf_size**
- **H5Pset_hyper_cache**

H5Pset_buffer

- Sets the maximum size for the type conversion buffer and background buffer used during data transfer. The bigger the size, the better the performance.
- Default is 1 MB.

H5Pset_sieve_buf_size

- Sets the maximum size of the data sieve buffer. The bigger the size, the fewer I/O requests issued for raw data access.
- Default is 64KB

H5Pset_hyper_cache

- **Indicates whether to cache hyperslab blocks during I/O, a process which can significantly increase I/O speeds.**
- **Default is to cache blocks with no limit on block size for serial I/O and to not cache blocks for parallel I/O.**

Chuck Cache Effect by H5Pset_cache

- Write one integer dataset 256x256x1024 (256MB)
- Using chunks of 256x16x1024 (16MB)
- Two tests of
  - Default chunk cache size (1MB)
  - Set chunk cache size 16MB

Chunk Cache Time Definitions

- Total: time to open file, write dataset, close dataset and close file.
- Dataset Write: time to write the whole dataset
- Chunk Write: best time to write a chunk
- User Time: total Unix user time of test
- System Time: total Unix system time of test

Chunk Cache size Results

Integer dataset size 256x256x1024 (256MB)					
Chunk size 256x16x1024 (16MB)					
Cache buffer size (MB)	Chunk write time (s)	Dataset write time (s)	Total time (s)	User time (s)	System time (s)
1	132.58	2450.25	2453.09	14	2200.1
16	0.376	7.83	8.27	6.21	3.45

Chunk Cache Size Summary

- **Big chunk cache size improves performance**
- **Poor performance mostly due to increased system time**
  - **Many more I/O requests**
  - **Smaller I/O requests**

Access Methods

Albert Cheng
NCSA

Remote Resource Access

- **Three experimental Virtual File Drivers added**
  - **Globus (Grid)**
  - **Grid Storage (Grid-experiment)**
  - **Storage Resource Broker (SDSC)**

Grid File Driver

- Use Globus 1.1.2 GASS interface
- Can do HTTP access
  - Generally read access
  - Can do write if the HTTP server permits
- No partial access yet
- No FTP support

Grid Storage File Driver

- Contributed by Alliance Relativity group and Albert Einstein Institute at Germany
- Use GLOBUS experimental software
- Supports FTP protocol
- Can do partial access with DPSS server

Storage Resource Broker File Driver

- Use the SRB by SDSC
- Use the low-level file API
- Partial access supported
- Use SRB server
- SRB knows the file only by filename and nothing else

Storage Resource Broker File Driver (cont.)

- Tested with latest SRB (1.1.7)
- Define Metadata of an HDF5 file
  - All attributes of the Root group?
  - All attributes of all objects (datasets, groups, datatypes, …)?
  - XML representation of the whole file, except the DATA part