高性能计算程序设计基础

任课教师：黄聃（Huang，Dan）

Introduction to HDF5

Slides by Reuben D. Budiardja
Scientific Computing
National Institute for Computational Sciences

High Level I/O Libraries

- Provide an appropriate abstraction for domain
  - Multidimensional datasets
  - Typed variables
  - attributes
- Self-describing, structured file format
- Map to middleware interface
  - Encourage collective I/O
- Provide optimization that middleware cannot

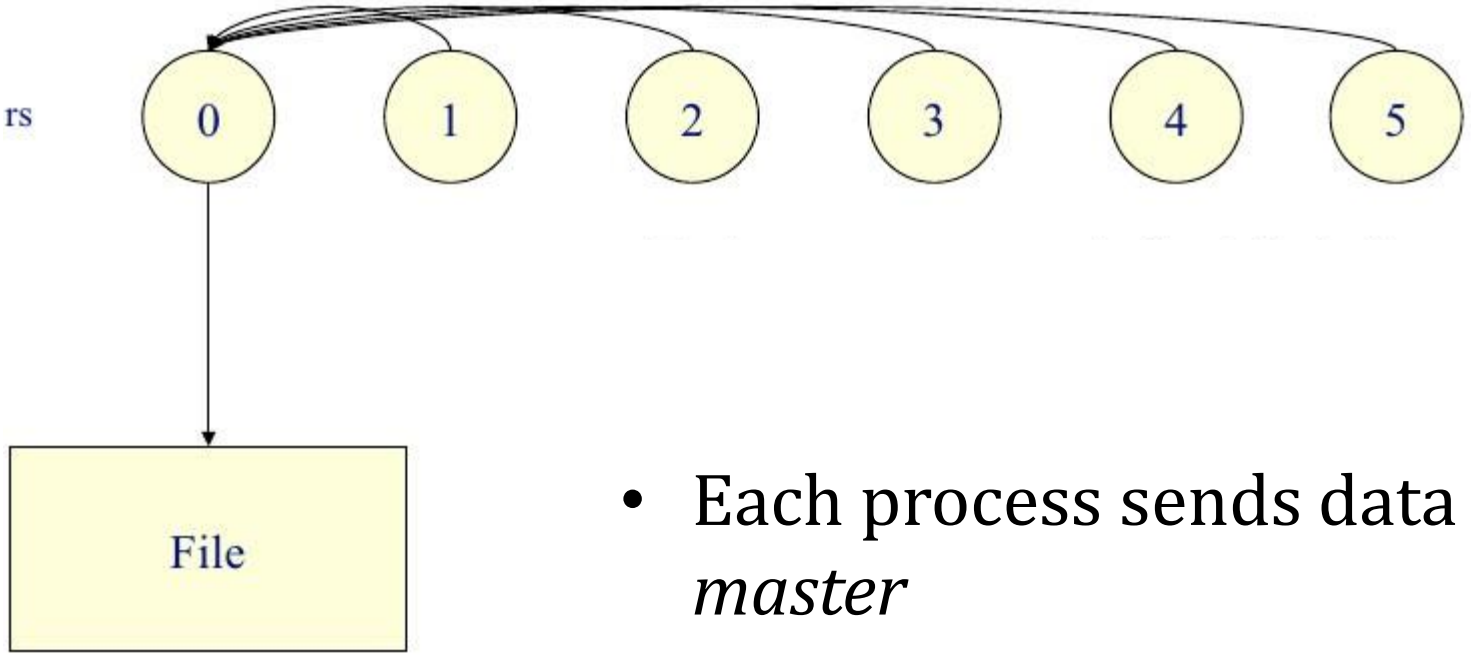
High Level I/O Libraries

- Provide an appropriate abstraction for domain
  - Multidimensional datasets
  - Typed variables
  - attributes
- Self-describing, structured file format

E.g. HDF5, NetCDF, Adios

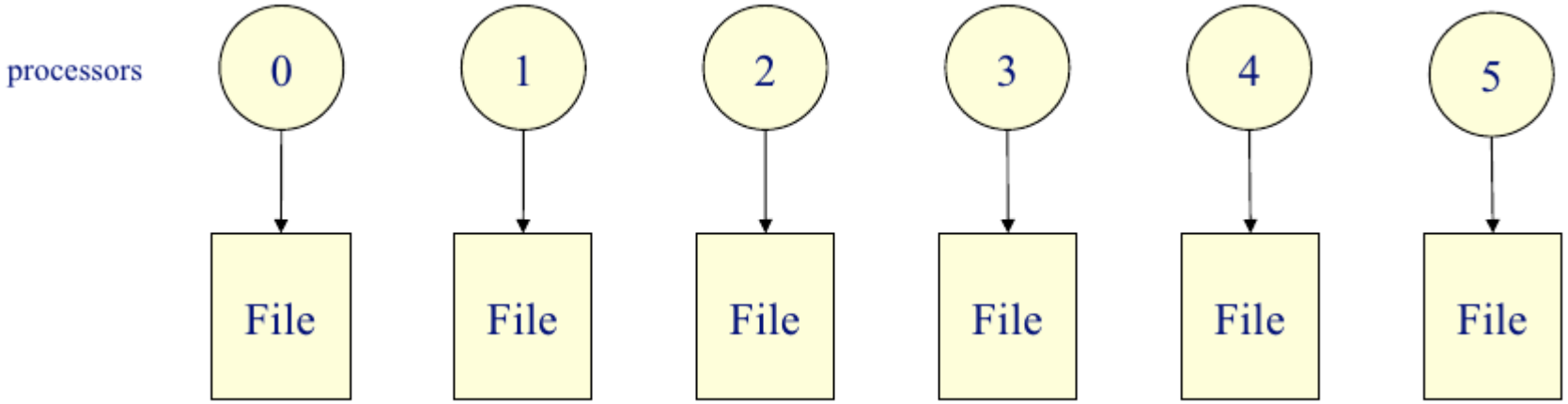
- Map to middleware interface
  - Encourage collective I/O
- Provide optimization that middleware cannot

Serial I/O



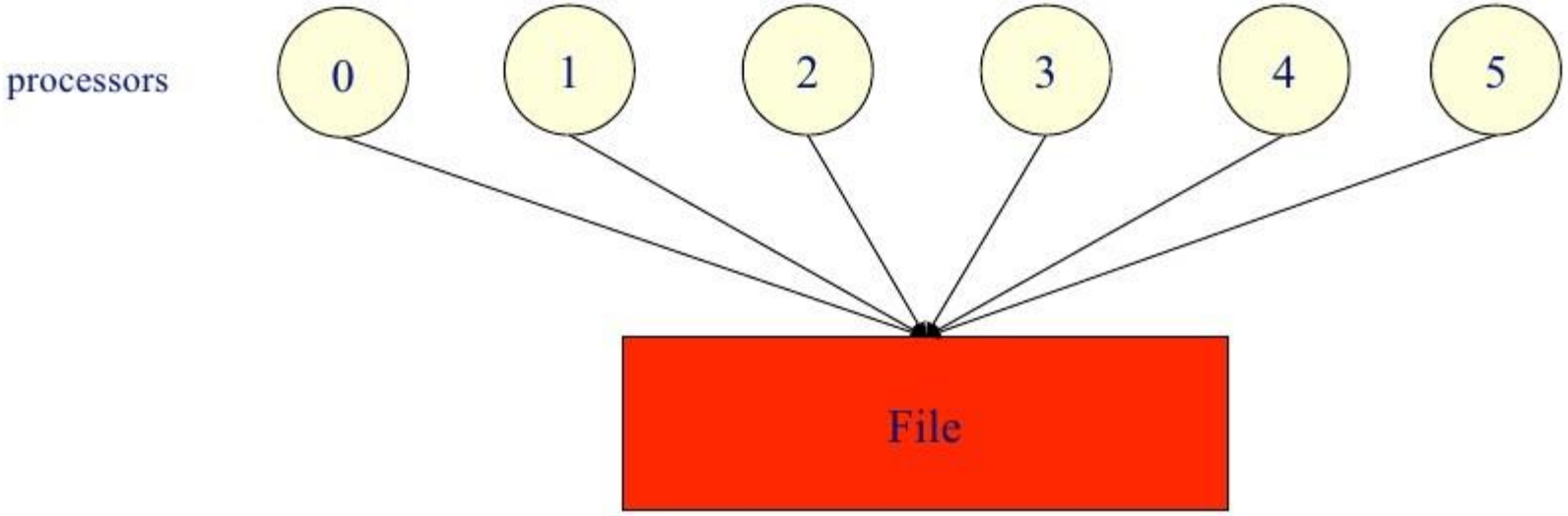
- Each process sends data to *master*
- *Master* writes data to file

Parallel I/O Multi-File



- Each process writes their own file

Parallel I/O Shared File



- Each process performs I/O to a single shared file

Common I/O File Format

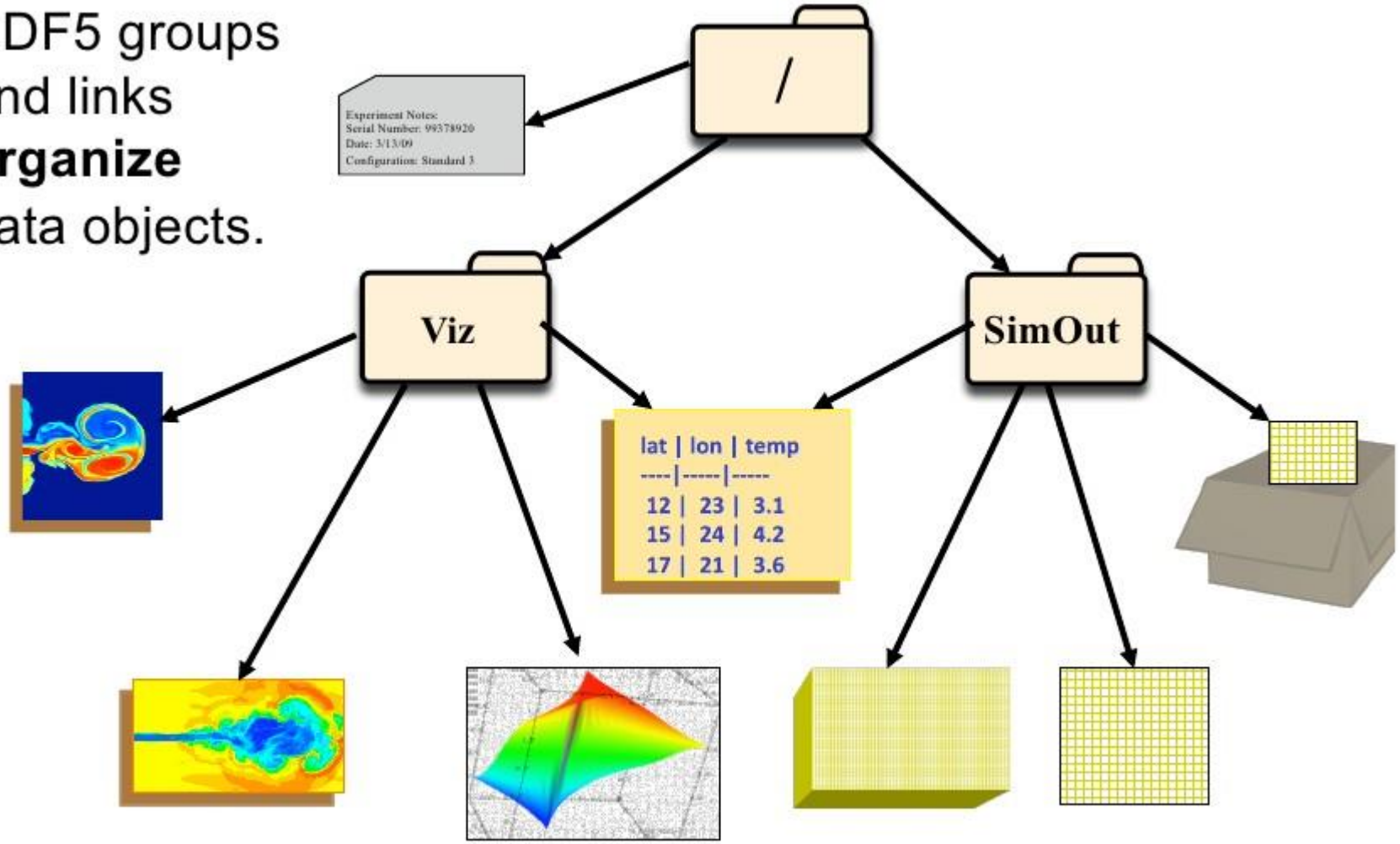
- ASCII
  - Slow
  - Takes too much space, no compression
  - Inaccurate
- Binary
  - Non-portable (byte-ordering? types? precisions?)
  - Not future proof
  - Can be parallel I/O via MPI-IO
- Self-Describing formats
  - HDF5, NetCDF, ADIOS
  - Portable
  - Parallel I/O support in the library

What is HDF5 ?

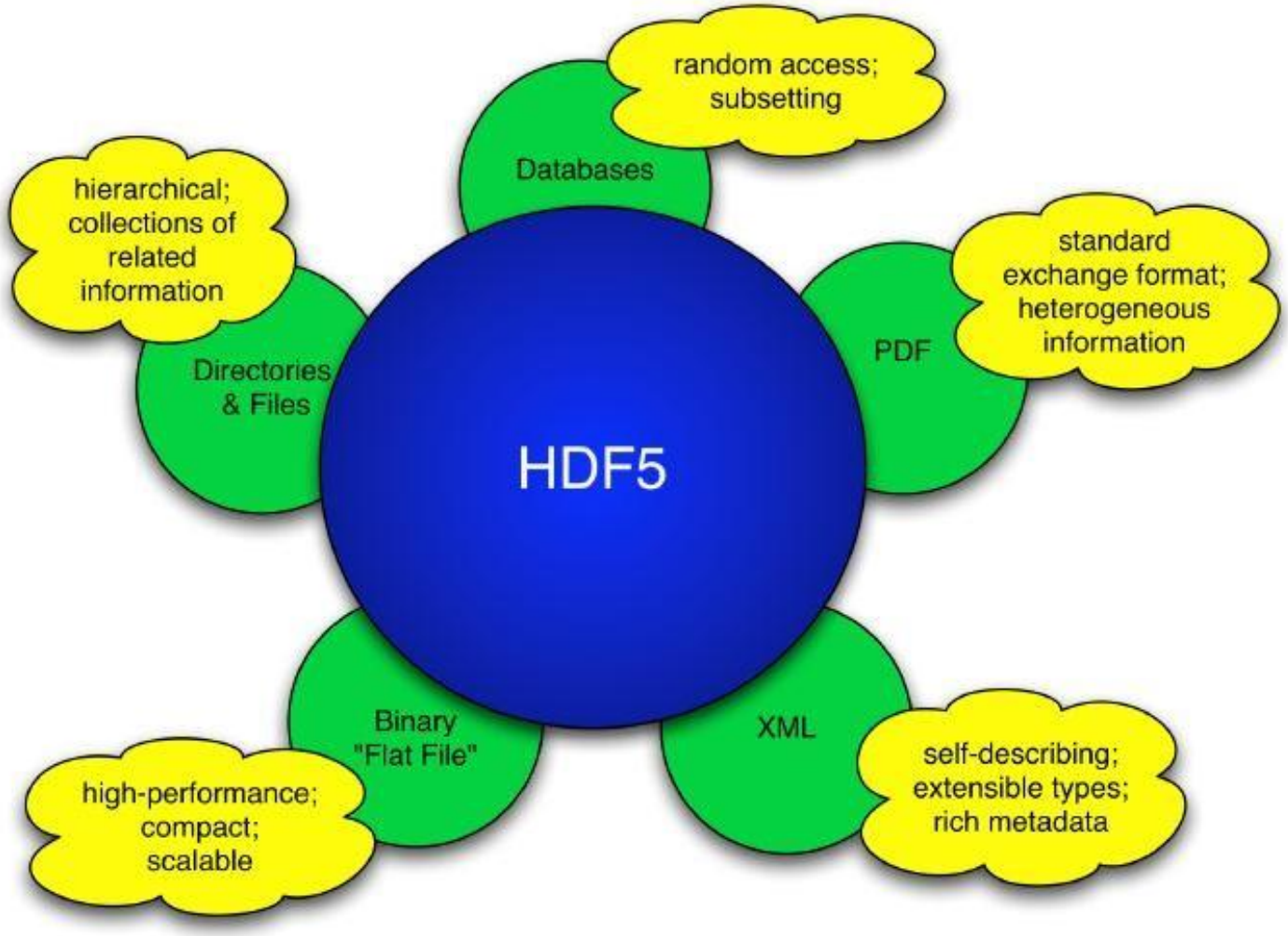
- Hierarchical Data Format, v5
- Open file format
  - Designed for high volume or complex data
- Open Source Software
  - Works with data in the format
- A data model
  - Structures for data organization and specification

HDF5 as Container of Objects

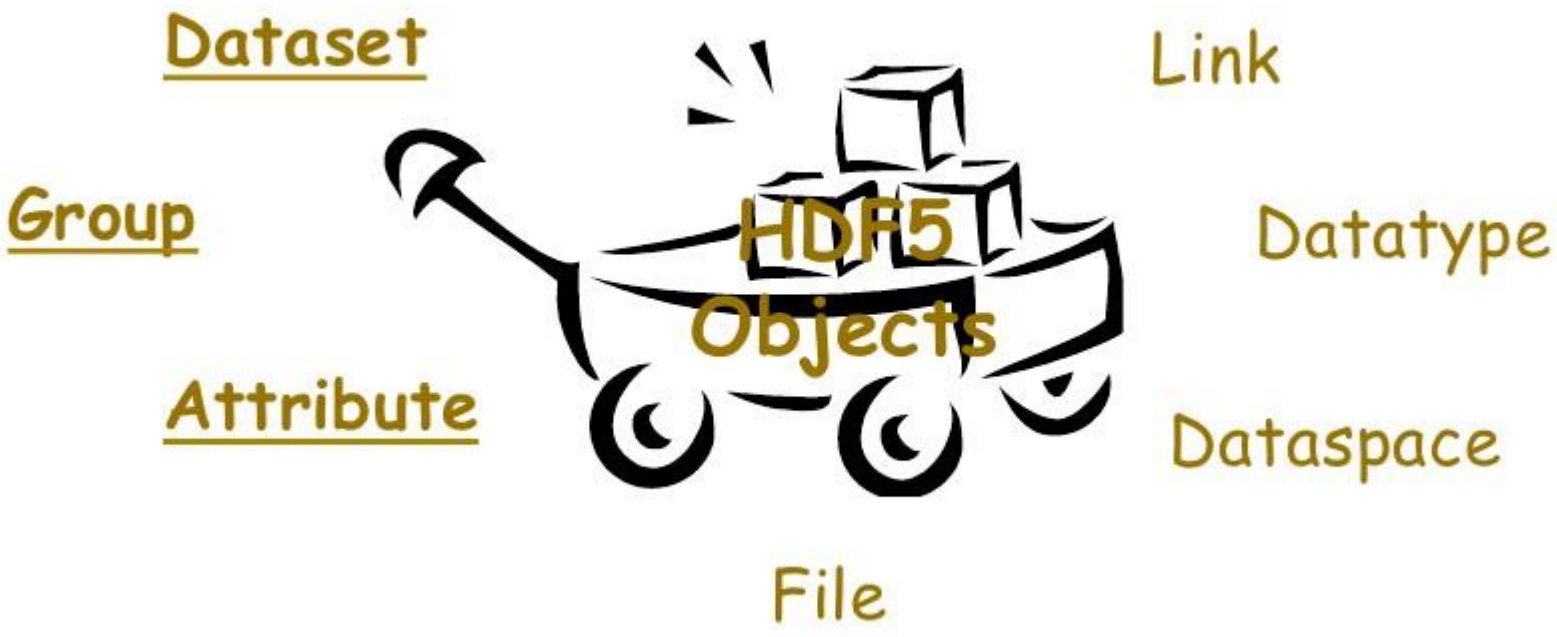
HDF5 groups and links **organize** data objects.



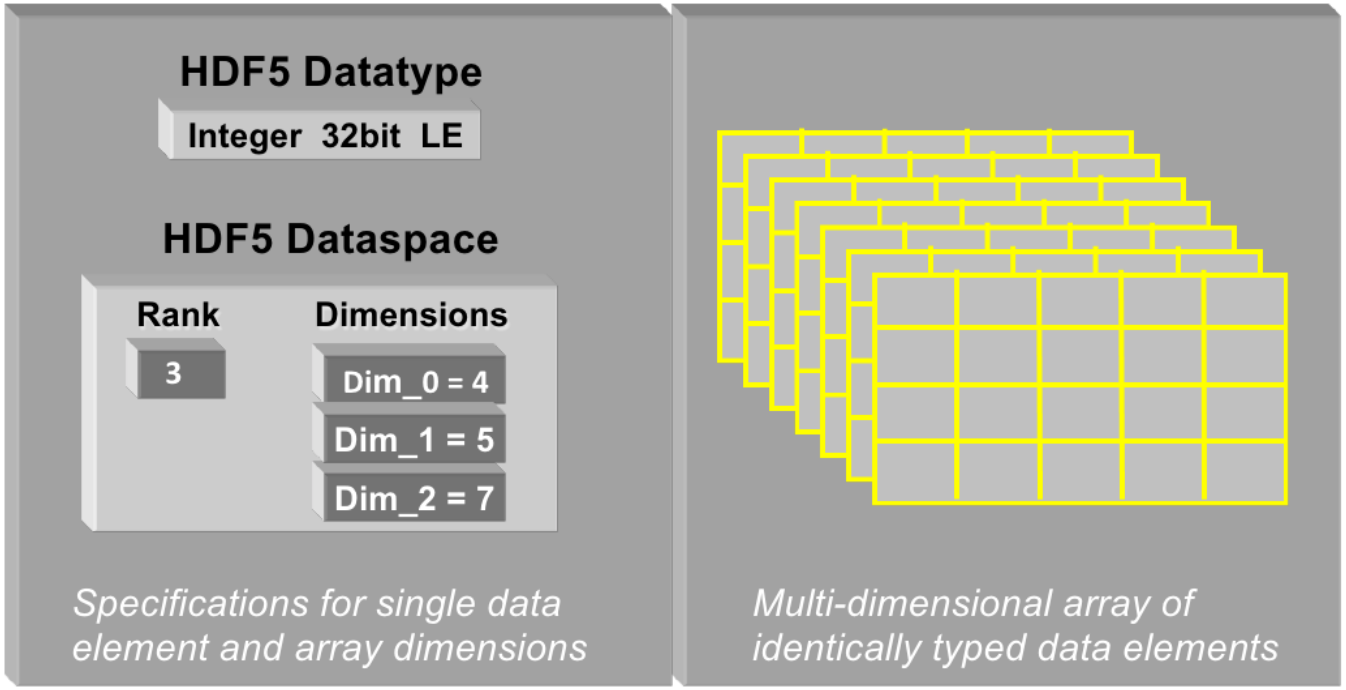
HDF5 is like . . .


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 Data Model



HDF5 Dataset



- HDF5 datasets organize and contain “raw data values”
  - HDF5 datatype describes individual data elements
  - HDF5 dataspace describes the logical layout of the data elements

HDF5 Datatypes

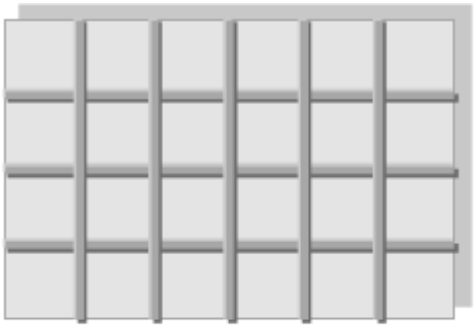
- Describes individual data elements
- Wide range of datatypes supported:
  - Integer
  - Float
  - Character
  - Array
  - User-defined (e.g. 13-bit integer)
  - Variable length (e.g. strings)
  - Compound (similar to C structs)
  - Opaque
  - References to objects / dataset regions

HDF5 Dataspace

- Describes the logical layout of the elements in an HDF5 dataset, e.g.:
  - Scalar
  - Simple array (*most common*)
    - Multiple elements organized in a rectangular array
      - Rank = number of dimensions
      - Dimension sizes = number of elements in each dimension
      - Maximum number of elements in each dimensions
        - May be fixed or unlimited

HDF5 Dataspace

- Dataspace contains spatial information



Rank = 2

Dimensions = 4x6

- Necessary for partial I/O:

Rank = 1

Dimension = 10

HDF5 Attributes

- Typically contain user metadata
- Have a *name* and *value*
- Attributes "decorate" HDF5 objects
- Use case: variable name of an HDF5 object
  - E.g.: pressure, density
  - Group name

HDF5 Group

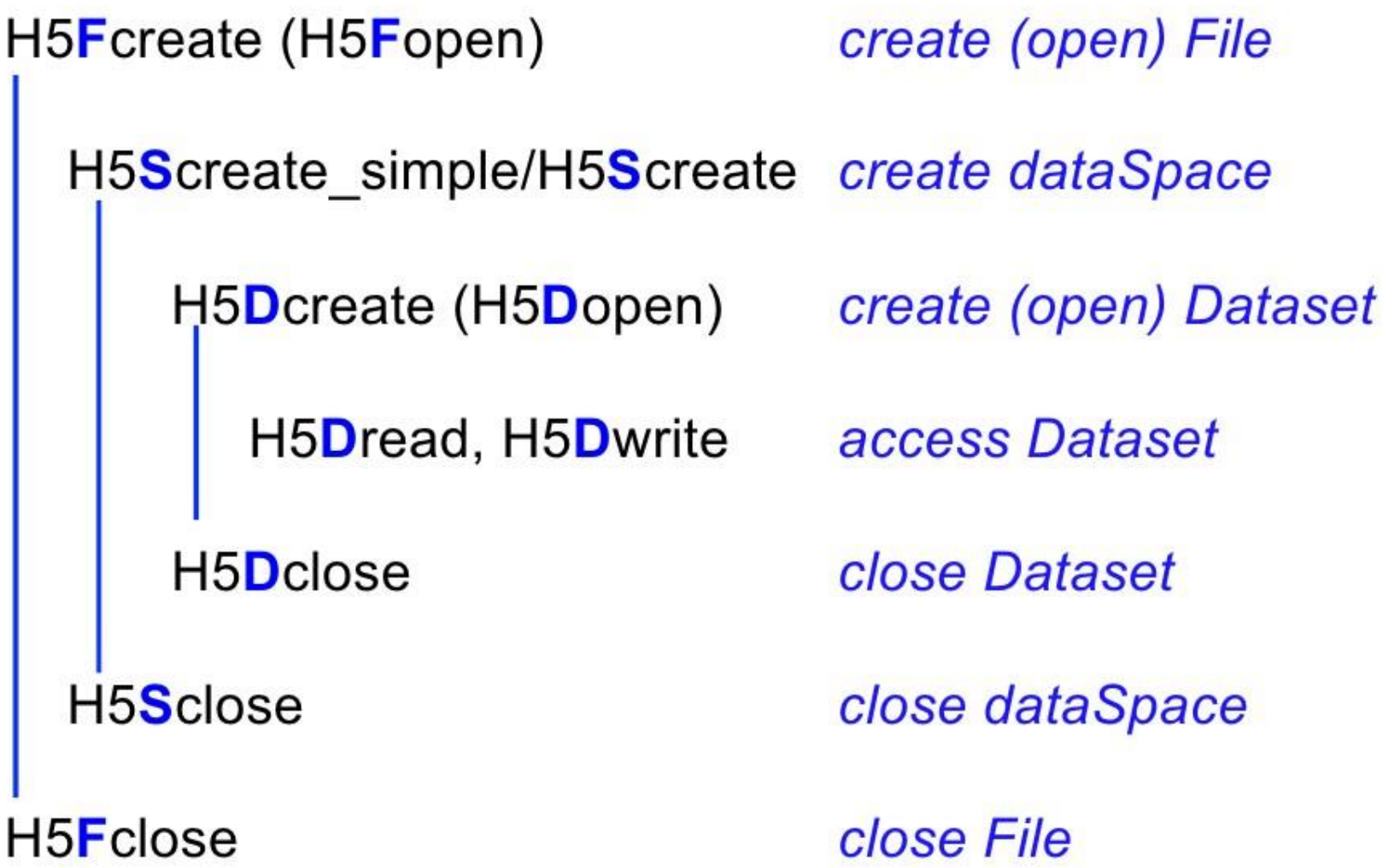
- HDF5 Group organize data objects
- Multiple objects can be *group*-ed
- Think of it as *virtual directories* in your HDF5 file

General Programming Paradigm

- Object is created or opened
- Object is accessed (read, written), possibly many times
- Object is closed

- Properties of object are *optionally* defined (can use the default)

HDF5 Basic Function



HDF5: Create a File

```
hid_t      file_id;
herr_t     status;


file_id = H5Fcreate ("file.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_Default);

status = H5Fclose (file_id);
```

HDF5 Example Code

```
#include "hdf5.h"
int main() {
hid_t   file_id, dataset_id, dataspace_id, group_id;
hsize_t dims[2];
herr_t  status;

file_id = H5Fcreate ("file.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
group_id = H5Gcreate (file_id, "fluid", H5P_DEFAULT, H5P_DEFAULT,H5P_DEFAULT);

dims[0] = 4;
dims[1] = 6;
dataspace_id = H5Screate_simple (2, dims, NULL);
dataset_id = H5Dcreate (group_id, "pressure",H5T_NATIVE_FLOAT,
                                dataspace_id, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

status = H5Dclose (dataset_id);
status = H5Sclose (dataspace_id);
status = H5Gclose (group_id);

group_id = H5Gcreate (file_id, "particle", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
status = H5Gclose (group_id);
status = H5Fclose (file_id);
}
```

h5dump

```
[rbudiard@darter1 hdf5]$ h5dump file.h5
HDF5 "file.h5" {
GROUP "/" {
   GROUP "fluid" {
      DATASET "pressure" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
         DATA {
         (0,0): 0, 0, 0, 0, 0, 0,
         (1,0): 0, 0, 0, 0, 0, 0,
         (2,0): 0, 0, 0, 0, 0, 0,
         (3,0): 0, 0, 0, 0, 0, 0
         }
      }
   }
   GROUP "particle" {
   }
}
}
```

Partial I/O → Parallel I/O

```
[rbudiard@darter1 hdf5]$ h5dump file.h5
HDF5 "file.h5" {
GROUP "/" {
   GROUP "fluid" {
      DATASET "pressure" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
         DATA {
         (0,0): 0, 0, 0, 0, 0, 0,
         (1,0): 0, 0, 0, 0, 0, 0,
         (2,0): 0, 0, 0, 0, 0, 0,
         (3,0): 0, 0, 0, 0, 0, 0
         }
      }
   }
   GROUP "particle" {
   }
}
}
```

Proc 0

Proc 1

Proc 0

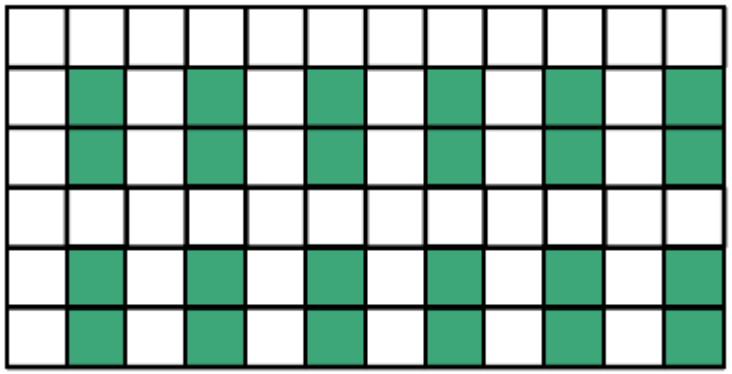
Proc 0

Describing a Subset in HDF5

- Before writing / reading a subset, need to describe it to HDF5 library
- HDF5 APIs refer to a subset as “selection” or “hyperslab”
- If specified, HDF5 will perform I/O on a *selection* or *hyperslab* only, not on all elements of a dataset
- Hyperslab selection is especially important for Parallel I/O in HDF5

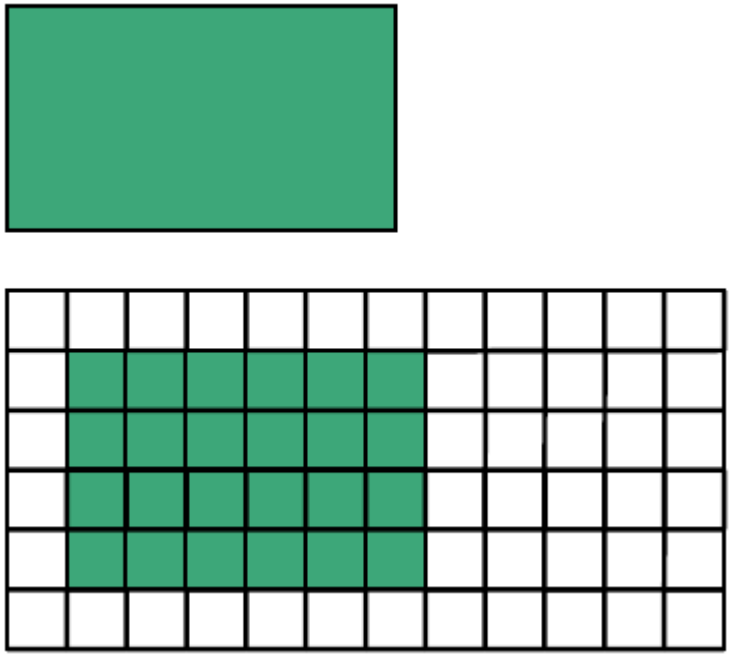
Hyperslab Description

- *Everything is measured in number of elements*
- ***start*** – starting location of a hyperslab (1,1)
- ***stride*** – number of elements that separate each block (3,2)
- ***count*** – number of blocks (2, 6)
- ***block*** – block size (2,1)



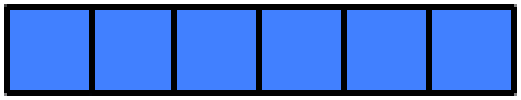
Example: Hyperslab Description

- We can describe the following hyperslab in two ways:
- As *several* blocks:
  - stride (1,1)
  - count (2,6)
  - block (2, 1)
- As *one* block:
  - stride (1,1)
  - count (1,1)
  - block (4,6)
- No performance penalty for either way

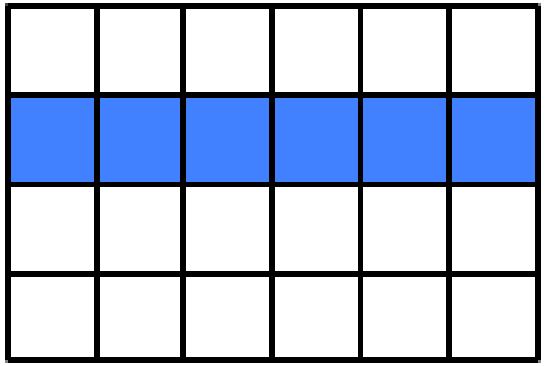


Partial I/O – Writing A Row

- Data in memory space: 1-dim array of size 6



- File space selection (hyperslab):
  start = {1, 0}; stride = {1, 1}; count = {1, 6}; block = {1, 1}



HDF5 Example Code: Writing A Row

```
...
hid_t mspace_id, fspace_id;
hsize_t start[2], count[2], mdims[1] = {6};
float   data[6] = {3.1, 3.2, 3.3, 3.4, 3.5, 3.6};

file_id = H5Fcreate ("file.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
group_id = H5Gcreate (file_id, "fluid", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

dims[0] = 4;
dims[1] = 6;
dataspace_id = H5Screate_simple (2, dims, NULL);
dataset_id = H5Dcreate (group_id, "pressure", H5T_NATIVE_FLOAT,
                        dataspace_id, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

mspace_id = H5Screate_simple(1, mdims, NULL);
fspace_id = H5Dget_space(dataset_id);

start[0] = 1; start[1] = 0;
count[0] = 1; count[1] = 6;
status = H5Sselect_hyperslab(fspace_id, H5S_SELECT_SET, start, NULL, count, NULL);
H5Dwrite(dataset_id, H5T_NATIVE_FLOAT, mspace_id, fspace_id, H5P_DEFAULT, data);

status = H5Dclose (dataset_id);
status = H5Sclose (dataspace_id);
status = H5Gclose (group_id);

group_id = H5Gcreate (file_id, "particle", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
status = H5Gclose (group_id);
status = H5Fclose (file_id);
```

h5dump: Writing A Row

```
HDF5 "file2.h5" {
GROUP "/" {
   GROUP "fluid" {
      DATASET "pressure" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
         DATA {
         (0,0): 0, 0, 0, 0, 0, 0,
         (1,0): 3.1, 3.2, 3.3, 3.4, 3.5, 3.6,
         (2,0): 0, 0, 0, 0, 0, 0,
         (3,0): 0, 0, 0, 0, 0, 0
         }
      }
   }
   GROUP "particle" {
   }
}
}
```

When / How To Use HDF5 ?

- Bundle / organize your data of multiple variables, multiple time-slices, to a single file
- Write in self-describing, future-proof, portable format
- To do parallel I/O where each process write their own contribution
- Write optimize I/O without having to be an “MPI-IO and Lustre expert”
- Hide complexity to concentrate on Science