PARALLEL I/O AND PORTABLE DATA FORMATS

HDF5

04.11.2024 | Jolanta Zjupa (j.zjupa@fz-juelich.de)

Brief history

- Hierarchical Data Format, Version 5
- **1987** beginning of development by the Graphics Foundations Task Force (GFTF) at the National Center for Supercomputing Applications (NCSA)
- **1998** HDF5 Version 1.0.0 released
- **2005** HDF Group is spinning off from NCSA 'as a non-profit corporation supporting open source software and non-proprietary data formats'
- **2012** Parallel HDF5 designed to work with MPI (Message Passing Interface protocol) and MPI I/O libraries

File organisation

HDF5 file structure corresponds in many respects to a Unix/Linux file system (FS)

HDF5		Unix/Linux fs
Group	↔	Directory
Data set	↔	File

/DataSet1

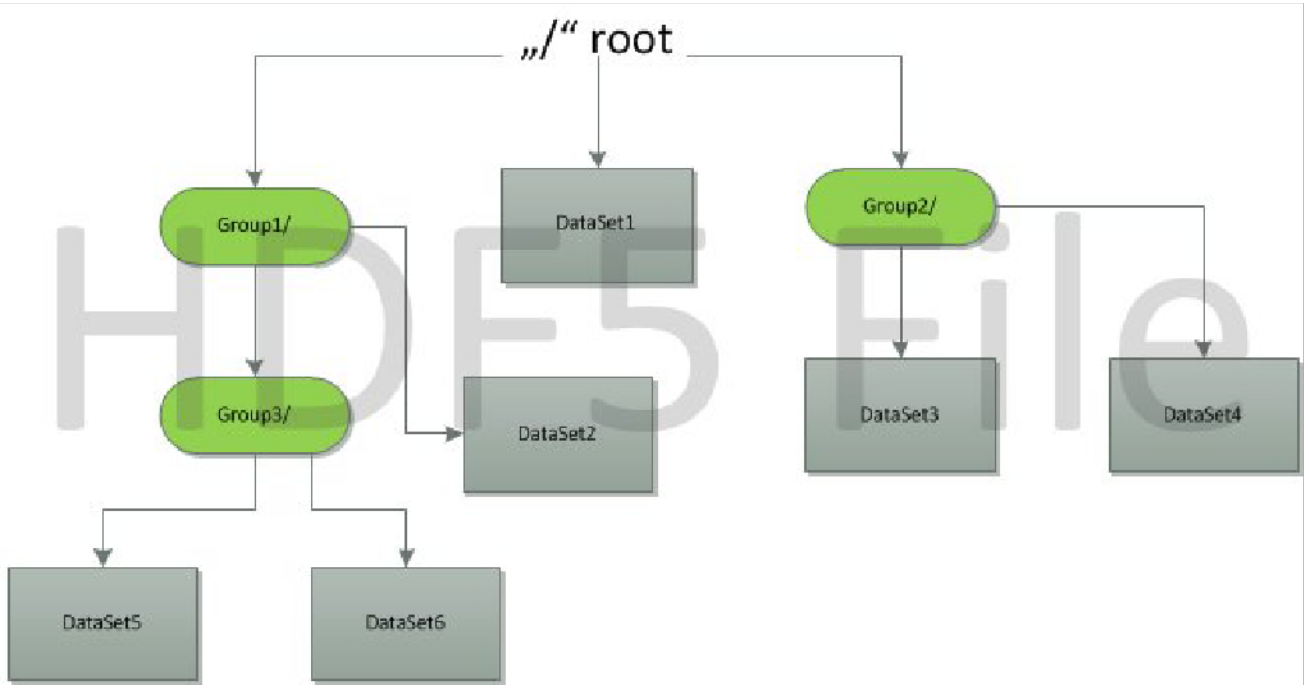
/Group1/DataSet2

/Group1/Group3/DataSet5

/Group1/Group3/DataSet6

/Group2/DataSet3

/Group2/Dataset4

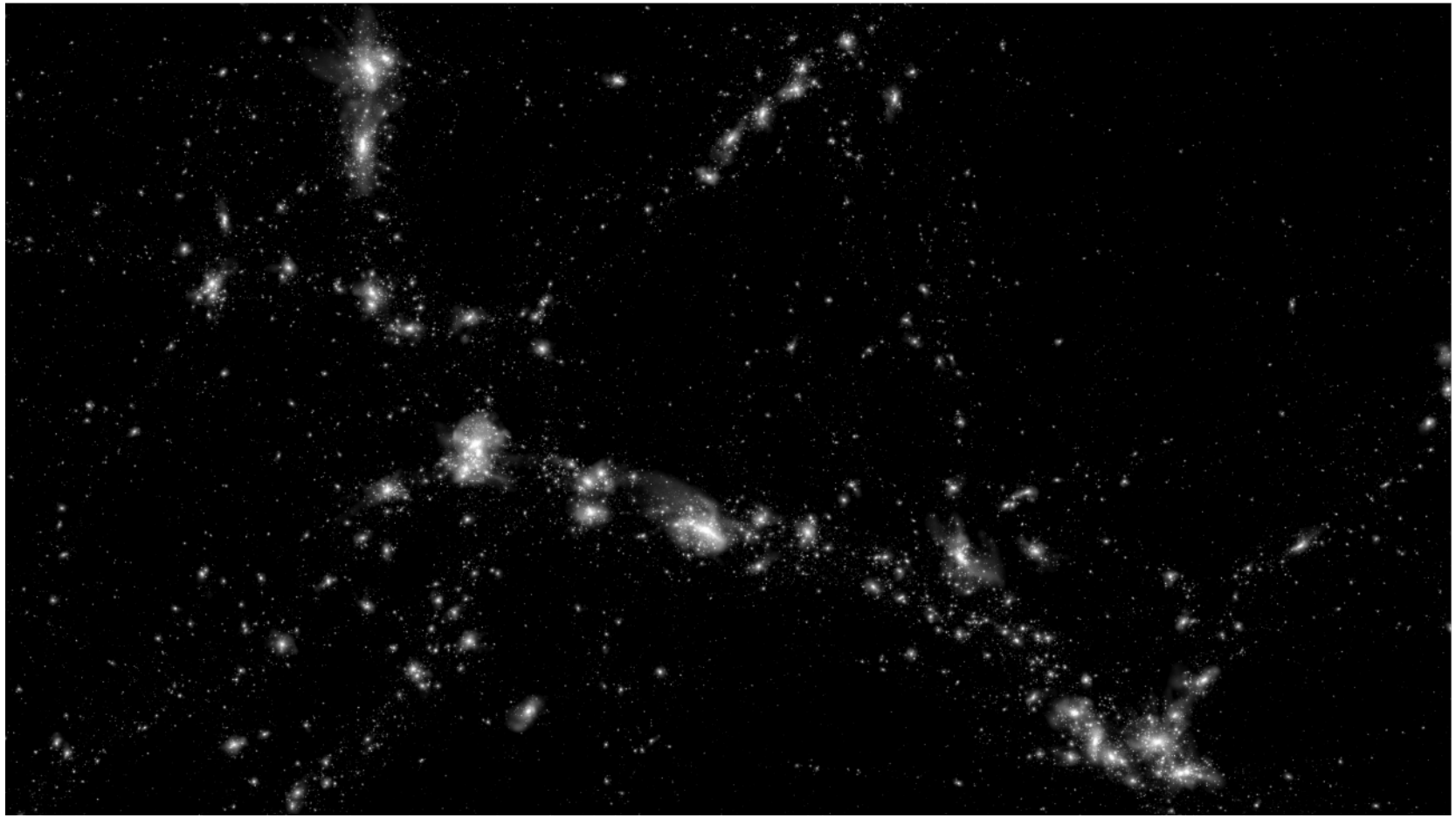


HDF5 features

- Hierarchical Data Format
- Consists of 4 main objects: (1.) Files (2.) Groups (3.) Datasets (4.) Attributes
- Self-describing data model which allows the management of complex data sets
- Portable file format
- Available on a variety of platforms
- Supports C, C++, Fortran 90, Python and Java
- Supports parallel I/O
- Provides tools to operate on HDF5 files and data

Getting started with HDF5

Install HDF5, or on the JSC systems: `module load HDF5 h5py`



scientific test dataset form:
https://www.tng-project.org/

containing:
- Haloes (‘Groups’)
- Galaxies (‘Subhaloes’)
- their phys. properties

Resources

HDF5 - C/C++, Fortran, Java

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

- User Guide
- Reference Manual

h5py - Python

https://docs.h5py.org/en/stable/index.html

Tools

https://docs.hdfgroup.org/archive/support/HDF5/doc/RM/Tools.html

HDF5 tools

```
$ h5ls /path/to/file.hdf5
```

```
$ h5ls TNG100n455Dark/fof_subhalo_tab_099.0.hdf5
Group                    Group
Subhalo                  Group
```

only top level

```
$ h5ls -r TNG100n455Dark/fof_subhalo_tab_099.0.hdf5
/Group                   Group
/Subhalo                 Group
/Subhalo/SubhaloMass     Dataset {9650}
/Subhalo/SubhaloPos      Dataset {9650, 3}
```

recursive

```
$ h5ls -r TNG100n455Dark/fof_subhalo_tab_099.0.hdf5/Subhalo
/SubhaloMass             Dataset {9650}
/SubhaloPos              Dataset {9650, 3}
```

HDF5 tools

```
$ h5ls /path/to/file.hdf5
```

```
$ h5ls -d TNG100n455Dark/fof_subhalo_tab_099.0.hdf5/Subhalo/SubhaloPos
SubhaloPos               Dataset {9650, 3}
    Data:
        845.843, 26355.2, 18311.9, 119.587, 24624.1, 16892.6, 815.786, 26724.8,
17484.6, 236.536, 26494, 15934.1, 764.69, 26543.3, 15541.7, 1429.23, 26329.4,
19048.1, 888.743, 26572.1, 17897.3, ....
```

dataset

HDF5 tools

```
$ h5dump /path/to/file.hdf5
```

```
$ h5dump –H TNG100n455Dark/fof_subhalo_tab_099.0.hdf5
```

only header

```
HDF5 "TNG100n455Dark/fof_subhalo_tab_099.0.hdf5" {
GROUP "/" {
  GROUP "Group" {
  }
  GROUP "Subhalo" {
      DATASET "SubhaloMass" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 9650 ) / ( 9650 ) }
      }
      DATASET "SubhaloPos" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 9650, 3 ) / ( 9650, 3 ) }
      }
  }
}
}
```

HDF5 tools

```
$ h5dump /path/to/file.hdf5
$ h5dump –g Subhalo TNG100n455Dark/fof_subhalo_tab_099.0.hdf5   group
$ h5dump –d Subhalo/SubhaloPos
                          TNG100n455Dark/fof_subhalo_tab_099.0.hdf5 dataset
HDF5 "TNG100n455Dark/fof_subhalo_tab_099.0.hdf5" {
DATASET "Subhalo/SubhaloPos" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SIMPLE { ( 9650, 3 ) / ( 9650, 3 ) }
   DATA {
   (0,0): 845.843, 26355.2, 18311.9,
   (1,0): 119.587, 24624.1, 16892.6,
   (2,0): 815.786, 26724.8, 17484.6,
   ...
```

Getting started with HDF5

Install HDF5, or on the JSC systems: `module load HDF5 h5py`

Include HDF5 module

C
```
#include <hdf5.h>
```

F
```
use hdf5
```

Py
```
import h5py
```

Compile C/C++, Fortran Code (Python no compilation needed)

C
```
gcc/mpicc -lhdf5 program.c -o program.exec
```

F
```
gfortran/mpif90 -lhdf5_fortran program.f90 -o program.exec
```

HDF5 also comes with compiler wrappers:

`[C]: h5cc/h5pcc, [C++] h5c++, [F90]: h5fc/h5pfc`

Fortran HDF5

The HDF5 library interface needs to be initialized (e.g. global variables) by calling `H5OPEN_F` before it can be used in your code and closed (`H5CLOSE_F`) at the end.

Fortran

```
H5OPEN_F(STATUS)
  INTEGER, INTENT(OUT) :: STATUS

H5CLOSE_F(STATUS)
  INTEGER, INTENT(OUT) :: STATUS
```

`status` returns `0` if successful

API naming scheme & Library specific types

Prefix	Operates on
H5F	Files
H5G	Group
H5S	Dataspaces
H5D	Datasets
H5A	Attributes
H5P	Property lists
H5T	Datatypes
...	...
H5	Library functions: general-purpose functions

Type	Description
[C] hid_t [F] INTEGER(HID_T)	Object identifier
[C] herr_t [F] INTEGER	Function return value
[C] hsize_t [F] INTEGER(HSIZE_T)	Used for dimensions

[Python]: no explicit usage of types.

Datatypes [C]

predefined datatypes

`H5T_NATIVE_CHAR`

`H5T_NATIVE_INT`

`H5T_NATIVE_FLOAT`

`H5T_NATIVE_DOUBLE`

`...`

`H5T_C_S1`

`...`

`H5T...()` functions to manage HDF5 datatypes

File [C]

```
C  hid_t H5Fopen( const char * filename, unsigned flags, hid_t fapl_id )
```

```
C  hid_t H5Fcreate( const char * filename, unsigned flags, hid_t fcpl_id,
   hid_t fapl_id )
```

[in]	filename	Name of the file to open / create	[out]	file_id	File identifier
[in]	flags	File access flags			
[in]	fcpl_id	File creation property list identifier			
[in]	fapl_id	File access property list identifier			

flags

H5F_ACC_RDONLY	Allows read-only access to file	**OPEN**
H5F_ACC_RDWR	Allows read and write access to file	**OPEN**
H5F_ACC_TRUNC	Truncate file, if it already exists, erasing all data previously stored in the file	**CREATE**
H5F_ACC_EXCL	Fail if file already exists	**CREATE**

```
C  herr_t H5Fclose( hid_t file_id )
```

[in]	file_id	File identifier	[out]	status	success/error

File [Python]

Open and create are handled by one / the same function call

Py
```
file = h5py.File('filename', mode)
```

[in]	filename	Name of the file to open / create	[out]	file	h5py file object
[in]	mode	File access mode			

mode

r	Readonly, file must exist (default)
r+	Read/write, file must exist
w	Create file, truncate if exists
w- / x	Create file, fail if exists
a	Read/write if exists, create otherwise

Py
```
file.close()
```

Group [C]

```c
hid_t H5Gopen( hid_t loc_id, const char * name, hid_t gapl_id )
```

```c
hid_t H5Gcreate( hid_t loc_id, const char * name, hid_t lcpl_id, hid_t gcpl_id, hid_t gapl_id )
```

[in]	loc_id	Location identifier: may be that of a file, group, dataset, named datatype, or attribute
[in]	name	Name of the group to open / create
[in]	lcpl_id	Link creation property list identifier
[in]	gcpl_id	Group creation property list identifier
[in]	gapl_id	Group access property list identifier
[out]	group_id	Group identifier

Note: upon file open / create the `root` group `'/'` is auto-created
operations on `file_id` are equivalent to operations on the `root` group

```c
herr_t H5Gclose( hid_t group_id )
```

[in]	group_id	Group identifier	[out]	status	success/error

Group [Python]

Py
```
grp = file['/path/to/groupname']
```
OPEN

[in]	obj	h5py object	[out]	grp	h5py group object
[in]	name	Name of the group (incl. path)			

Note: path can be *absolute* or *relative*

Py
```
grp = file.create_group('/path/to/groupname')
or
grp = file.create_group('path/to/groupname')
```
CREATE

Py
```
subgrp = outfile.create_group('(/)path/to/groupname/subgroupname')
or
subgrp = grp.create_group('(/)path/to/groupname/subgroupname')
or
subgrp = grp.create_group('subgroupname')
```
CREATE

Dataspace [C]

C
```
hid_t H5Screate( H5S_class_t type )
```

[in] `type` Type of dataspace to be created: `H5S_SCALAR` (single element), `H5S_SIMPLE` (array of elements), and `H5S_NULL` (no data element)

[out] `dspace_id` Dataspace identifier

C
```
hid_t H5Screate_simple( int rank, const hsize_t dims[], const hsize_t maxdims[] )
```

[in] `rank` Number of dimensions of dataspace

[in] `dims` Array specifying the size of each dimension

[in] `maxdims` Array specifying the maximum size of each dimension

[out] `dspace_id` Dataspace identifier

C
```
herr_t H5Sclose( hid_t dspace_id )
```

[in] `dspace_id` Dataspace identifier

[out] `status` success/error

Dataset [C]

```c
hid_t H5Dopen( hid_t loc_id, const char * name, hid_t dapl_id )
```

```c
hid_t H5Dcreate( 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 )
```

[in]	loc_id	Location identifier: may be that of a file, group, dataset, named datatype, or attribute
[in]	name	Name of the dataset to open / create
[in]	type_id	Datatype identifier
[in]	space_id	Dataspace identifier
[in]	lcpl_id	Link creation property list identifier
[in]	dcpl_id	Dataset creation property list identifier
[in]	dapl_id	Dataset access property list identifier
[out]	dset_id	Dataset identifier

```c
herr_t H5Dclose( hid_t dset_id )
```

[in]	dset_id	Dataset identifier	[out]	status	success/error

Dataset [C]

C
```
herr_t H5Dread( hid_t dset_id, hid_t mem_type_id, hid_t mem_space_id,
hid_t file_space_id, hid_t dxpl_id, void * buf )
```

C
```
herr_t H5Dwrite( hid_t dset_id, hid_t mem_type_id, hid_t mem_space_id,
hid_t file_space_id, hid_t dxpl_id, const void * buf )
```

[in]	dset_id	Identifier of the dataset to read from
[in]	mem_type_id	Identifier of the memory datatype
[in]	mem_space_id	Identifier of the memory dataspace
[in]	file_space_id	Identifier of the dataset's dataspace in the file
[in]	dxpl_id	Dataset transfer property list identifier
[in/out]	buf	Buffer with data to be written to the file
[out]	status	success/error

Note on `status`: non-negative value if successful; otherwise, returns a negative value.

Dataset [Python]

Py
```
dset = file['(/)path/to/dataset']
dset = grp['(/)path/to/dataset']
```
OPEN

Py
```
dset = obj.create_dataset('(/)path/to/dataset', data=data)
```
CREATE

[in]	obj	h5py object	[out]	dset	h5py dataset object
[in]	dataset	Name of the dataset (incl. path)			
[in]	data	any data			

Note: path can be *absolute* or *relative* (same as for group)

Py
```
obj['(/)path/to/dataset'] = data
```

- no `dset` output object

Dataset [Python]

Py `dset = obj.create_dataset('name', shape, dtype=dtype)` **CREATE**

[in]	obj	h5py object	[out]	dset	h5py dataset object
[in]	dataset	Name of the dataset (incl. path)			
[in]	shape	shape of data			
[in]	dtype	data type (*optional*, defaults to float `'f'`)			

Examples: `shape`

`1.) np.array([0,1,2,3,4]) - shape = (5,)`

`2.) np.array([[0,1,2],[0,1,2],[0,1,2]]) - shape = (3,3)`

Attribute [C]

C
```
hid_t H5Aopen( hid_t loc_id, const char * attr_name, hid_t aapl_id )
```

C
```
hid_t H5Acreate( hid_t loc_id, const char * attr_name, hid_t type_id, hid_t space_id, hid_t acpl_id, hid_t aapl_id )
```

[in]	loc_id	Location identifier: may be that of a file, group, dataset, or named datatype
[in]	attr_name	Name of attribute
[in]	type_id	Attribute datatype identifier (needed)
[in]	space_id	Dataspace identifier (needed)
[in]	acpl_id	Attribute creation property list identifier
[in]	aapl_id	Attribute access property list identifier
[out]	attr_id	Attribute identifier

C
```
herr_t H5Aclose( hid_t attr_id )
```

[in]	attr_id	Dataset identifier	[out]	status success/error

Attribute [C]

```c
herr_t H5Aread( hid_t attr_id, hid_t type_id, void * buf )
```

```c
herr_t H5Awrite( hid_t attr_id, hid_t type_id, const void * buf )
```

[in]	attr_id	Attribute identifier
[in]	type_id	Datatype (in-memory) identifier
[in/out]	buf	Buffer for data to be written / read

Attribute [Python]

Py
```
obj.attr['name'] = data
```

[in/out]	obj	h5py object, e.g. file, group, dataset, attribute
[in]	name	Name of the attribute
[in]	data	any data

Note: Datasets in Python have following descriptive attributes attached:

`shape, size, ndim, dtype, nbytes`

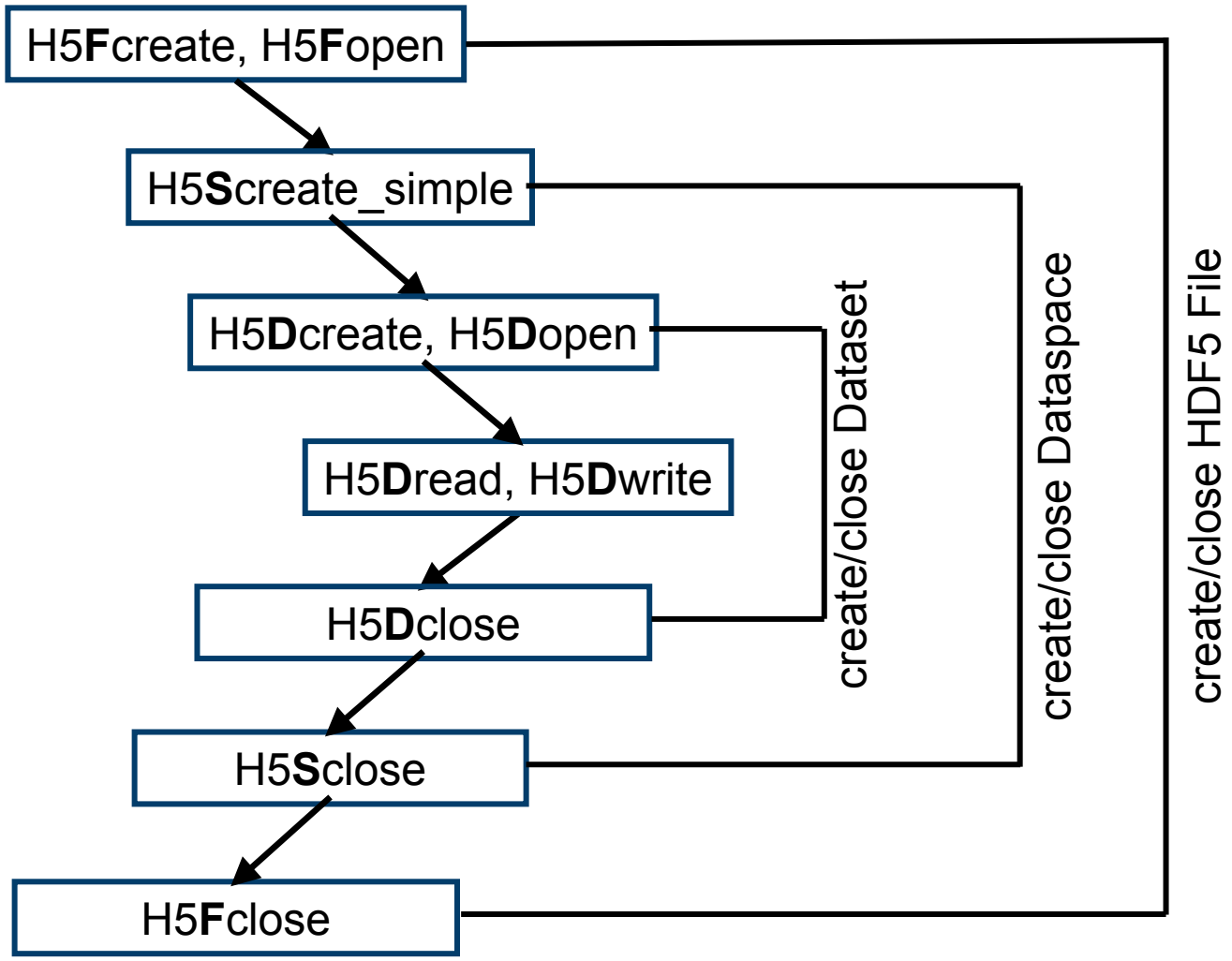
More Python Goodies

Py
```
obj.keys()
```

[in]	obj	h5py object

- prints the names of all next level h5py objects

Example workflow procedure



Exercises

Exercise 0

Inspect the dataset under:
`/p/project1/training2403/ParIO_course_material/exercises/HDF5/`
`TNG100n455Dark`
with HDF5 tools h5ls and h5dump

Exercise 1

- classroom discussion -
`/p/project1/training2403/ParIO_course_material/exercises/HDF5/`
`{C/Python}``/exc01.`**`{c/py}`**
understand the code structure & HDF5 routines, and fill out the *TODO's*

PARALLEL I/O AND PORTABLE DATA FORMATS

PARALLEL HDF5

04.11.2024 | Jolanta Zjupa (j.zjupa@fz-juelich.de)

Parallel HDF5

- File in parallel is opened on a MPI Communicator
- PHDF5 file identifier = MPI I/O file handle, all operations on file handle
- Different files can be opened on different communicators
- Collective calls need to called by all participating MPI processes.
- All routines that modify the HDF5 file structure are collective calls:
  - File operations e.g. `H5Fopen`, `H5Fcreate`, `H5Fclose`
  - Object creation e.g. `H5Dcreate`, `H5Dclose`
  - Object structure modification
- Data transfer (e.g. `H5Dwrite`, `H5Dread`) can be collective or independent
- Property list(s) used to control file access mechanism

File access property list

C
```
hid_t H5Pcreate( hid_t cls_id )
```

[in] cls_id Property list class identifier

[out] pl_id property list identifier

C
```
herr_t H5Pset_fapl_mpio( hid_t fapl_id, MPI_Comm comm, MPI_Info info )
```

[in] fapl_id File access property list identifier
[in] comm MPI communicator
[in] info MPI info object

Code example

C
```
hid_t fapl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl_id, MPI_COMM_WORLD, MPI_INFO_NULL);

hid_t file_id = H5Fcreate("f.hdf5", H5F_ACC_TRUNC, H5P_DEFAULT, fapl_id);
```

Data transfer property list

C
```
hid_t H5Pcreate( hid_t cls_id )
```

[in] cls_id Property list class identifier

[out] pl_id property list identifier

C
```
herr_t H5Pset_dxpl_mpio( hid_t dxpl_id, H5FD_mpio_xfer_t xfer_mode )
```

[in] dxpl_id Dataset transfer property list identifier
[in] xfer_mode Transfer mode: H5FD_MPIO_INDEPENDENT (default) or H5FD_MPIO_COLLECTIVE

Code example

C
```
hid_t dxpl_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(dxpl_id, H5FD_MPIO_COLLECTIVE);

H5Dwrite( dset_id, dtype_id, mem_space_id, file_space_id, dxpl_id, buf )
```

mem_space & file_space

Reminder: Dataspace – dimensions & length (in each dimension)

C

```
hid_t H5Screate_simple( int rank, const hsize_t dims[], const hsize_t
maxdims[] )
```

[in]	rank	Number of dimensions of dataspace
[in]	dims	Array specifying the size of each dimension
[in]	maxdims	Array specifying the maximum size of each dimension
[out]	dspace_id	Dataspace identifier

mem_space: Dataspace of the data that given MPI process wants to write in / read from a dataset

file_space: Dataspace in the HDF5 file for the data to be write in / read from

Hyperslabs

C
```
H5Sselect_hyperslab( long space_id, int op, byte[]/long[] start,
byte[]/long[] stride, byte[]/long[] count, byte[]/long[] block )
```

[in/out]	space_id	Identifier of dataspace selection to modify
[in]	op	Operation to perform on current selection
[in]	start	Offset of start of hyperslab
[in]	stride	Hyperslab stride
[in]	count	Number of blocks included in hyperslab
[in]	block	Size of block in hyperslab

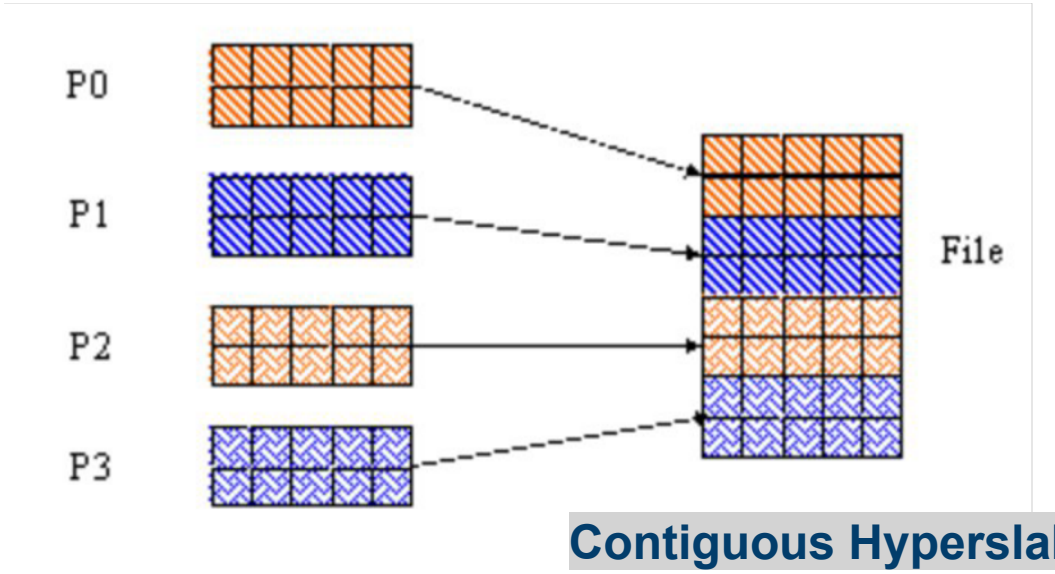
- Changes the input (file) dataspace `space_id` to the specified subselection, returning the modified (file) dataspace `space_id` as an output

Example: Contiguous Hyperslab

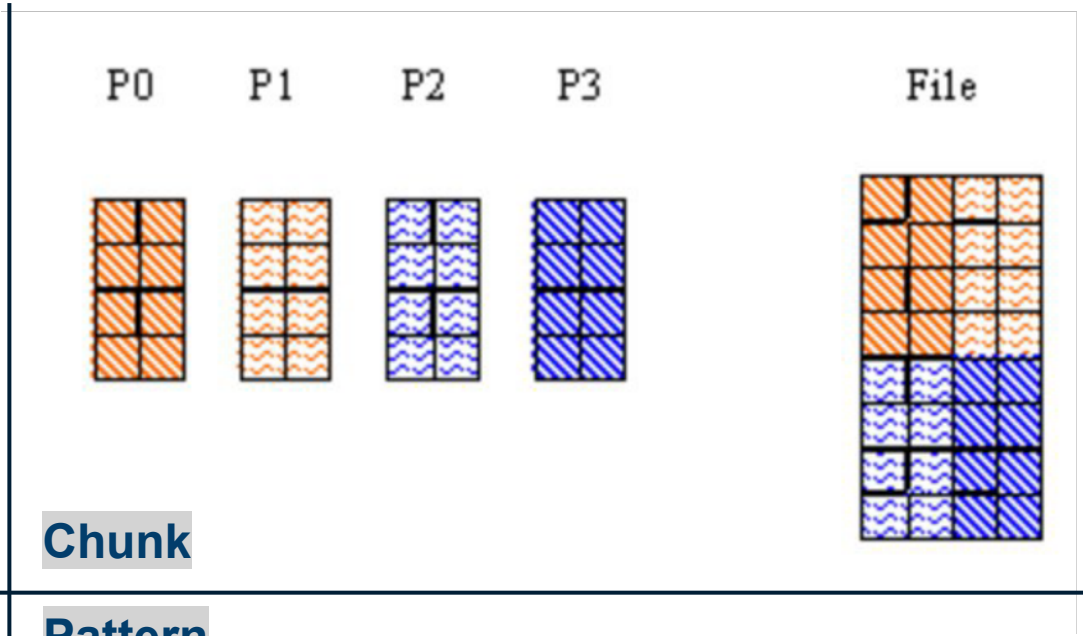
C
```
H5Sselect_hyperslab(dspace_id, H5S_SELECT_SET, &displs[rank], NULL,
&count_on_rank, NULL);
```

Hyperslabs

https://hyoklee.github.io/hdf5/develop/_intro_par_h_d_f5.html

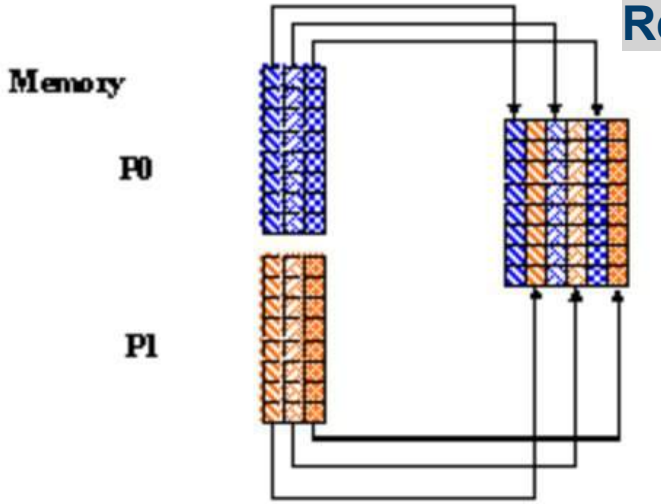


Contiguous Hyperslab



Chunk

Regularly Spaced Data



Pattern

Parallel h5py Workflow [Python]

1.) Open / create file shared across MPI processes

Py
```
file = h5py.File('filename', mode, driver='mpio', comm=MPI.COMM_WORLD)
```

[in]	filename	Name of the file to open / create	[out]	file	h5py file object
[in]	mode	File access mode			
[in]	driver	set to mpio opens a file shared across MPI processes participating in			
[in]	comm	MPI_Comm communicator			

2.) Create ***file*** dataset

Py
```
dset = file.create_dataset('dset_name', shape, dtype=dtype)
```
WRITE

3.) Write process data into the desired place in the file dataset

Py
```
dset[indexing] = data
```
WRITE

For **read**:

1.) open dataset (*Note:* reads in full dataset on all processes)

2.) select the subset the process needs through indexing (into new np.array())

3.) release the dataset (close the h5py file, or set `dset=0` and call `gc.collect()`)

Exercises

Exercise 2

- classroom discussion -
`/p/project1/training2403/ParIO_course_material/exercises/HDF5/`**`{C/Python}`**`/exc02.`**`{c/py}`**
understand the code structure & HDF5 routines, and fill out the *TODO's*

Exercise 3

- classroom discussion -
`/p/project1/training2403/ParIO_course_material/exercises/HDF5/`**`{C/Python}`**`/exc03.`**`{c/py}`**
understand the code structure & HDF5 routines, and fill out the *TODO's*