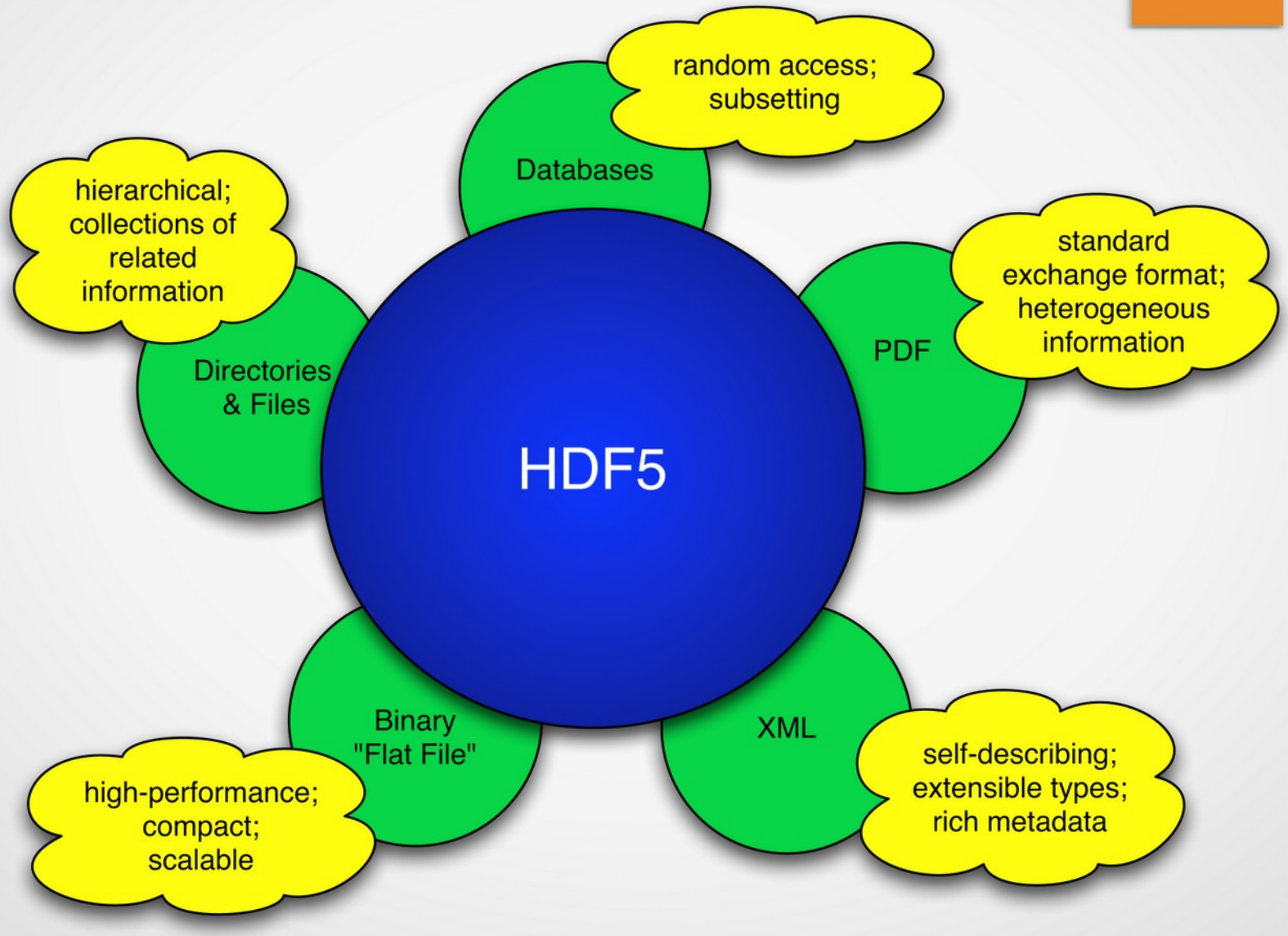
HDF5

Introduction to HDF5
and
a few advanced HDF5 features
for GLCS

What is HDF5?

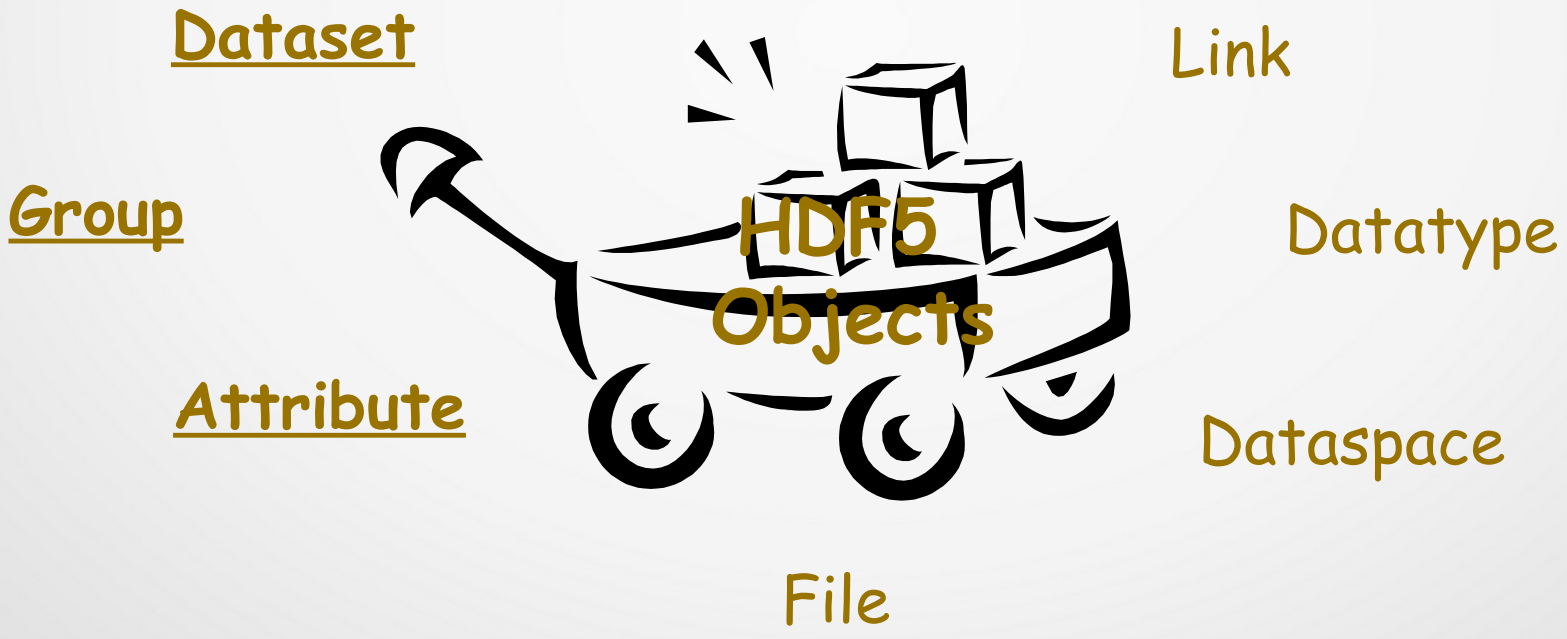
- HDF5 == Hierarchical Data Format, v5
- Open file format
  - Designed for high volume or complex data
- Open source software
  - Works with data in the format
- An extensible data model
  - Structures for data organization and specification

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

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



HDF5 Dataset

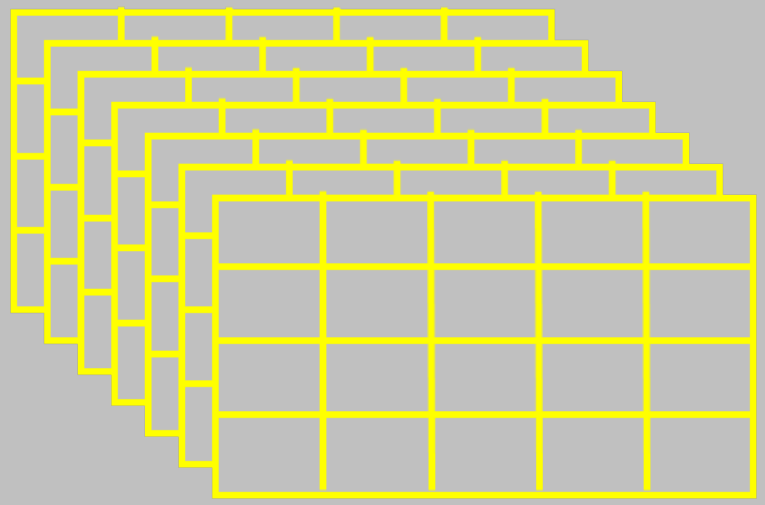
HDF5 Datatype

Integer: 32-bit, LE

HDF5 Dataspace

Rank	Dimensions
3	Dim[0] = 4
	Dim[1] = 5
	Dim[2] = 7

Specifications for single data element and array dimensions



Multi-dimensional array of identically typed data elements

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

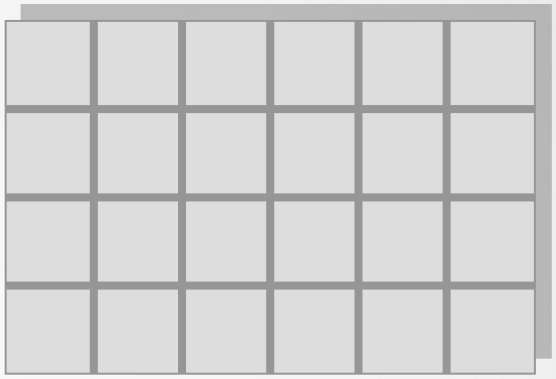
HDF5 Dataspace

- Describes the logical layout of the elements in an HDF5 dataset
  - NULL
    - no elements
  - Scalar
    - single element
  - 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 dimension
        - may be fixed or unlimited

HDF5 Dataspace

Two roles:

- Dataspace contains spatial information
  - Rank and dimensions
  - Permanent part of dataset definition



Rank = 2
Dimensions = 4x6

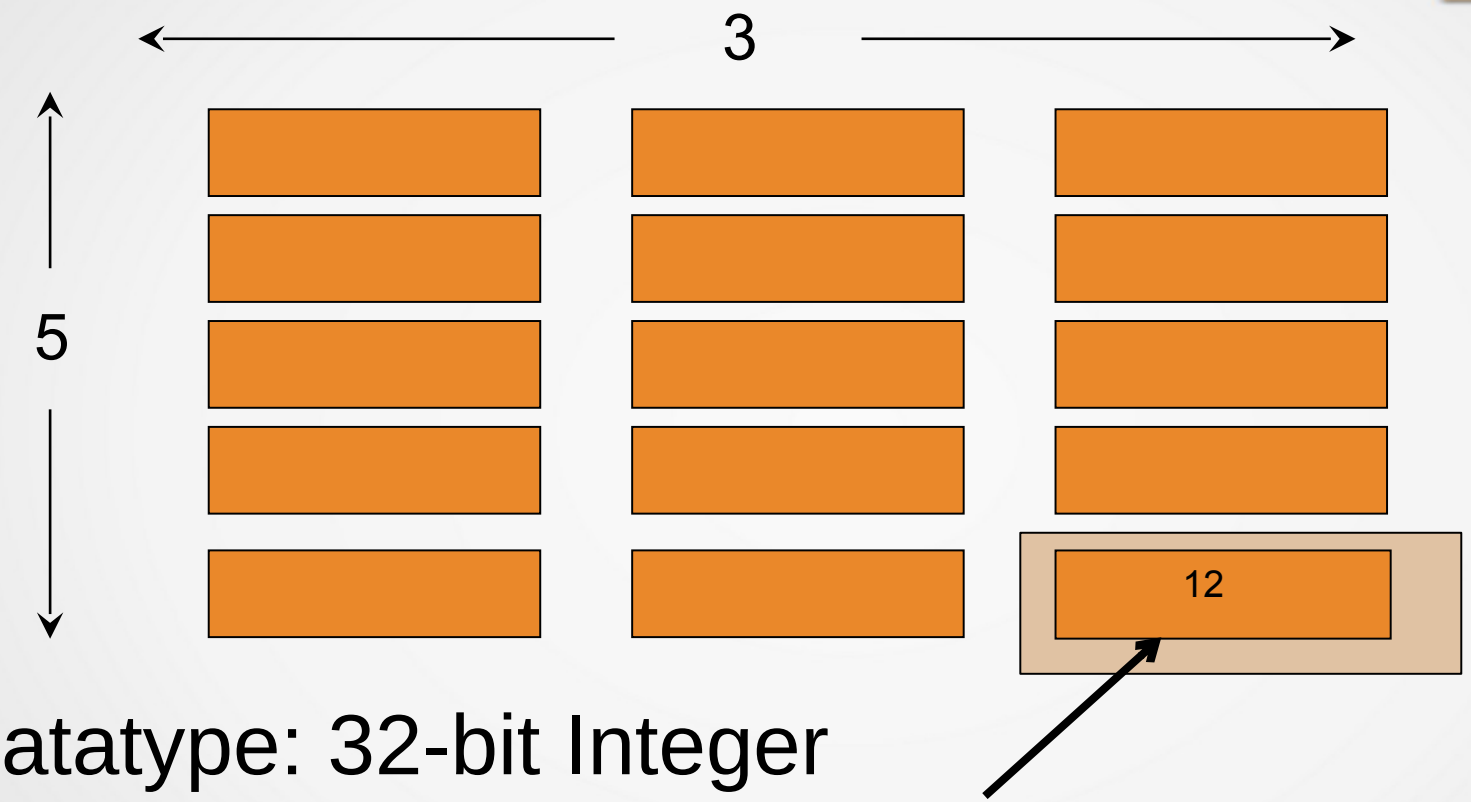
- Partial I/0: Dataspace describes application's data buffer and data elements participating in I/O

Rank = 1
Dimension = 10

HDF5 Datatypes

- Describe individual data elements in an HDF5 dataset
- Wide range of datatypes supported
  - Integer
  - Float
  - Enum
  - Array
  - User-defined (e.g., 13-bit integer)
  - Variable-length types (e.g., strings, vectors)
  - Compound (similar to C structs)
  - More …

HDF5 Dataset



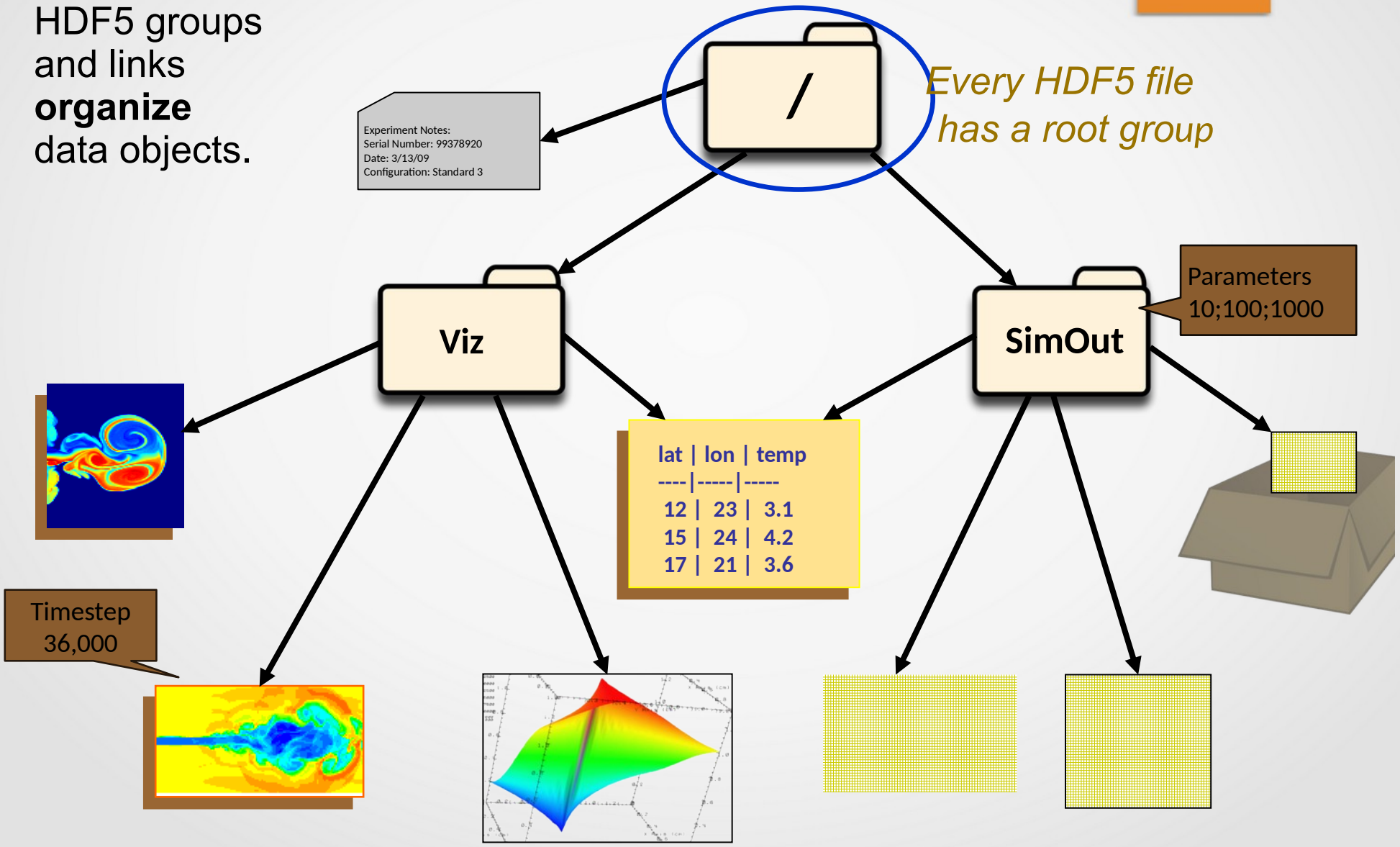
- Datatype: 32-bit Integer
- Dataspace:
  - Rank = 2
  - Dimensions = 5 x 3

HDF5 Attributes

- Typically contain user metadata
- Have a name and a value
- Attributes “decorate” HDF5 objects
- Value is described by a datatype and a dataspace
- Analogous to a dataset, but
  - do not support partial I/O operations
  - can not be compressed or extended

HDF5 Groups and Links

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



Useful Tools

- h5dump:
  - Tool to “dump” or display contents of HDF5 files
- h5cc, h5c++, h5fc:
  - Scripts to compile applications
- HDFView:
  - Java browser to view HDF5 files
  - http://www.hdfgroup.org/hdf-java-html/hdfview/

The General HDF5 API

- Prefix: H5?
  - ? is a character corresponding to the type of object the function acts on

- Example:
  - H5D: Dataset interface — *e.g.*, H5Dread
  - H5F: File interface — *e.g.*, H5Fopen
  - H5S: DataSpace interface — *e.g.*, H5Sclose

The HDF5 API

- For flexibility, the API is extensive
  - 300+ functions
- This can be daunting… but there is hope
  - A few functions can do a lot
  - Start simple
  - Build up knowledge as more features are needed

General Programming Paradigm

- Object is opened or created
- Object is accessed, possibly many times
- Object is closed

- Properties of object are optionally defined
  - Creation properties (e.g., use chunking storage)
  - Access properties

Basic Functions

H5Fcreate (H5Fopen)	*create (open) File*
H5Screate_simple/H5Screate	*create dataSpace*
H5Dcreate (H5Dopen)	*create (open) Dataset*
H5Dread, H5Dwrite	*access Dataset*
H5Dclose	*close Dataset*
H5Sclose	*close dataSpace*
H5Fclose	*close File*

Other Common Functions

- DataSpaces:
  - H5Sselect_hyperslab (Partial I/O)
  - H5Sselect_elements (Partial I/O)
  - H5Dget_space
- Groups:
  - H5Gcreate, H5Gopen, H5Gclose
- Property lists:
  - H5Pcreate, H5Pclose

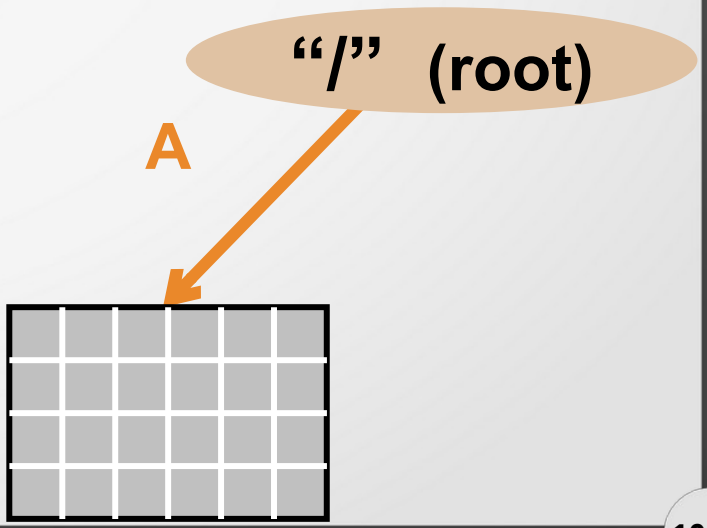
Code: Create a File

```
hid_t file_id = H5Fcreate("file.h5", H5F_ACC_TRUNC,
                                     H5P_DEFAULT,H5P_DEFAULT);
// …
H5Fclose (file_id);
```

“/” (root)

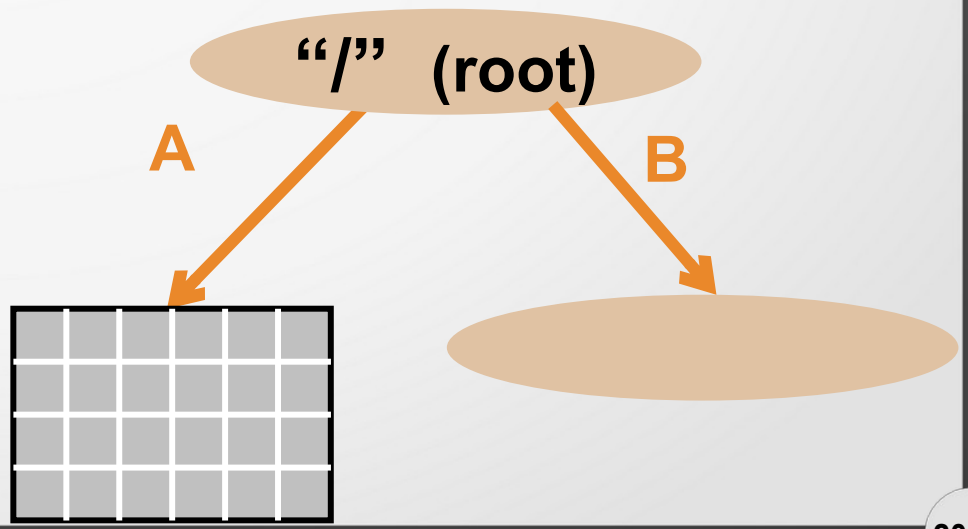
Code: Create a Dataset

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

hsize_t dims[2] = {4, 6};
hid_t dataspace_id = H5Screate_simple(2, dims, NULL);

hid_t dataset_id = H5Dcreate(file_id, "A", H5T_STD_I32BE,
                             dataspace_id, H5P_DEFAULT,
                             H5P_DEFAULT, H5P_DEFAULT);
// …

H5Dclose(dataset_id);

H5Sclose(dataspace_id);

H5Fclose(file_id);
```



Code: Create a Group

```
// …

hid_t file_id = H5Fopen("file.h5", H5F_ACC_RDWR,
                        H5P_DEFAULT);

hid_t group_id = H5Gcreate(file_id,"B", H5P_DEFAULT,
                           H5P_DEFAULT, H5P_DEFAULT);

// …

H5Gclose(group_id);
H5Fclose(file_id);
```



Output of h5dump

```
$ h5dump file.h5

HDF5 "file.h5" {
GROUP "/" {
   DATASET "A" {
      DATATYPE  H5T_STD_I32BE
      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 "B" {
   }
}
}
```

Example Code - H5Dwrite

```
int wdata[4][6];
for (i = 0; i < 4; i++)
  for (j = 0; j < 6; j++)
    wdata[i][j] = i * 6 + j + 1;

// …

H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
         H5P_DEFAULT, wdata);
```

Output of h5dump after writing

```
$ h5dump file.h5

HDF5 "file.h5" {
GROUP "/" {
   DATASET "A" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
      DATA {
      (0,0): 1, 2, 3, 4, 5, 6,
      (1,0): 7, 8, 9, 10, 11, 12,
      (2,0): 13, 14, 15, 16, 17, 18,
      (3,0): 19, 20, 21, 22, 23, 24
      }
   }
   GROUP "B" {
   }
}
}
```

How to write a row?

```
$ h5dump file.h5

HDF5 "file.h5" {
GROUP "/" {
   DATASET "A" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
      DATA {
      (0,0): 0, 0, 0, 0, 0, 0,
      (1,0): 7, 8, 9, 10, 11, 12,
      (2,0): 0, 0, 0, 0, 0, 0,
      (3,0): 0, 0, 0, 0, 0, 0
      }
   }
   GROUP "B" {
   }
}
}
```

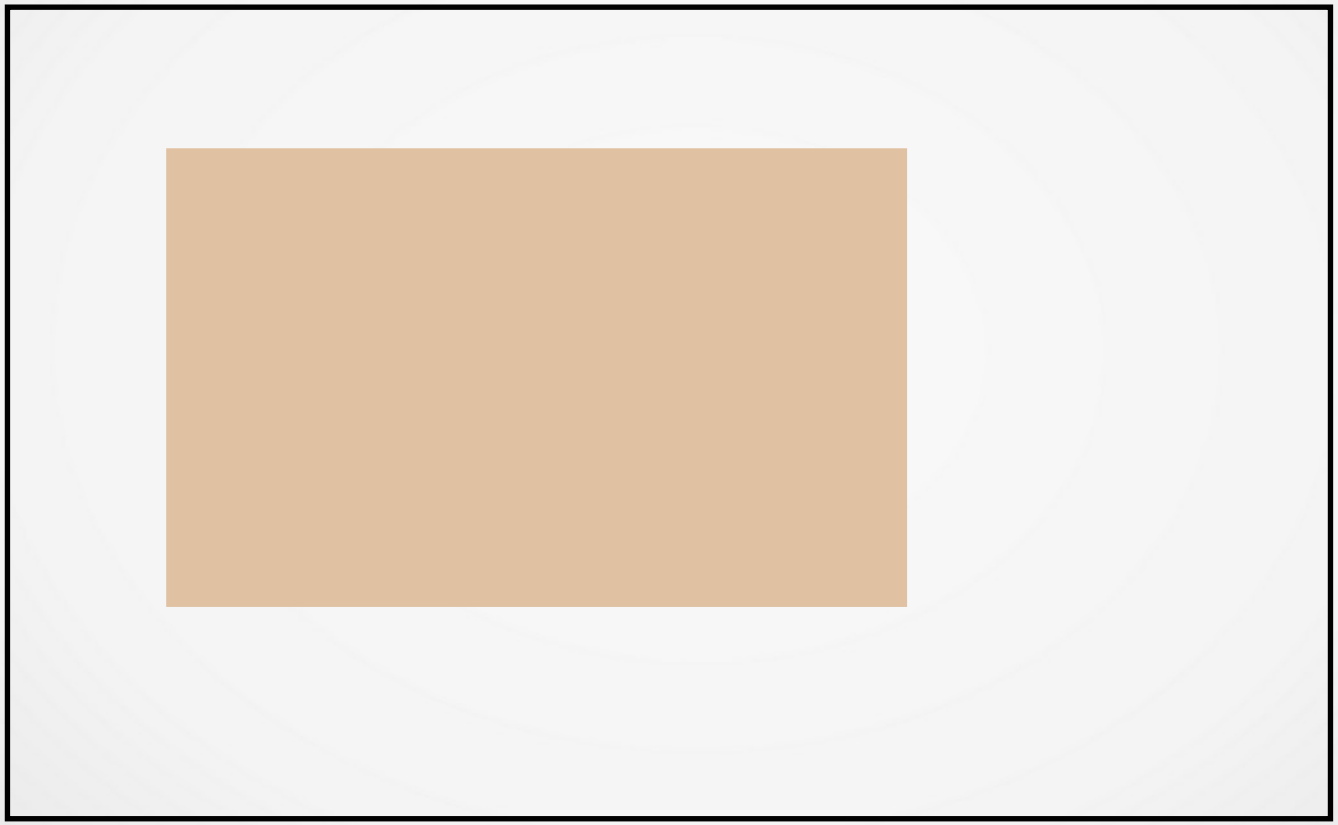
Describe a Subset in HDF5

- Before writing and reading a subset of data one has to describe it to the HDF5 Library
- HDF5 APIs and documentation refer to a subset as a “selection” or “hyperslab selection”
- If specified, HDF5 Library will perform I/O on a selection only and not on all elements of a dataset.

Types of Selections in HDF5

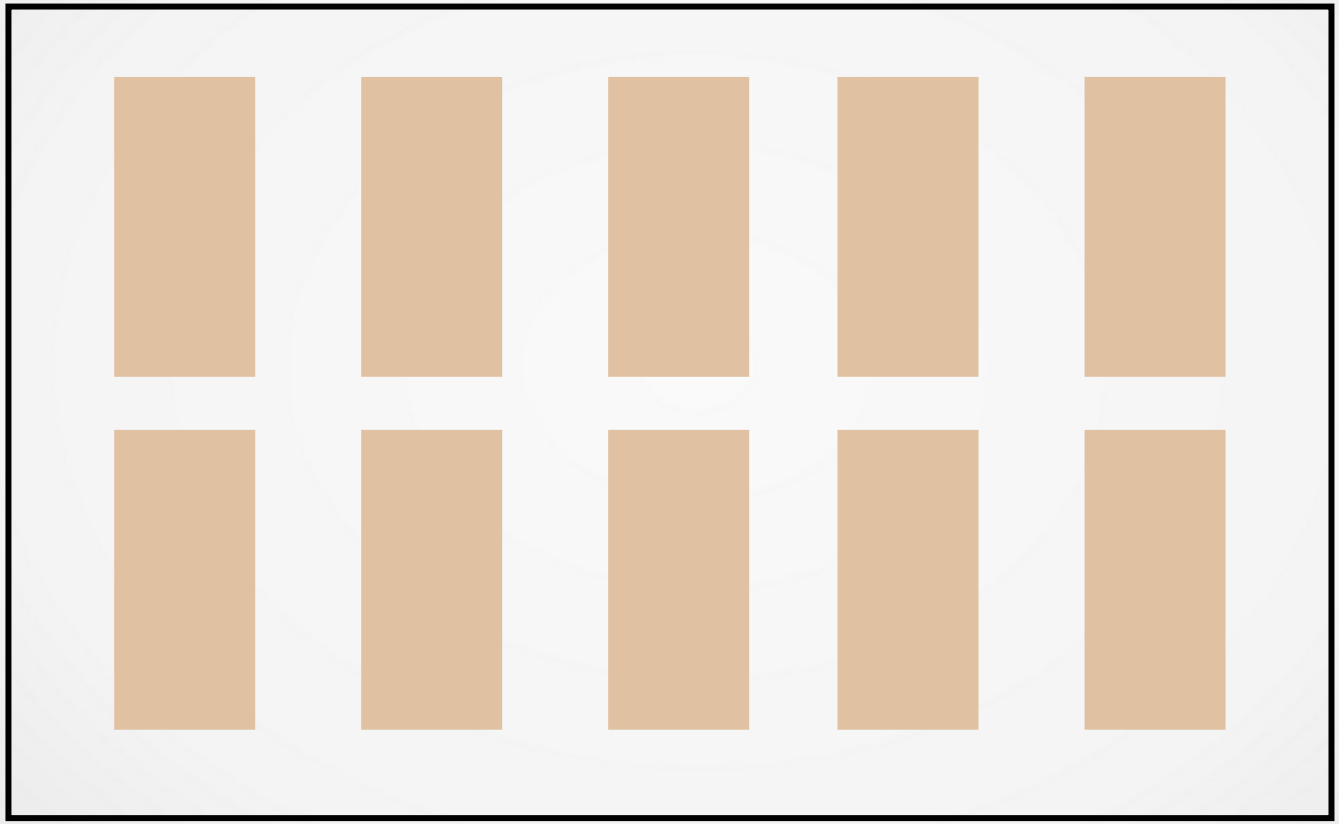
- Two types of selections
  - Hyperslab selection
    - Regular hyperslab
    - Simple hyperslab
    - Result of set operations on hyperslabs (union, difference, …)
  - Point selection
- Hyperslab selection is especially important for doing parallel I/O in HDF5

Simple Hyperslab



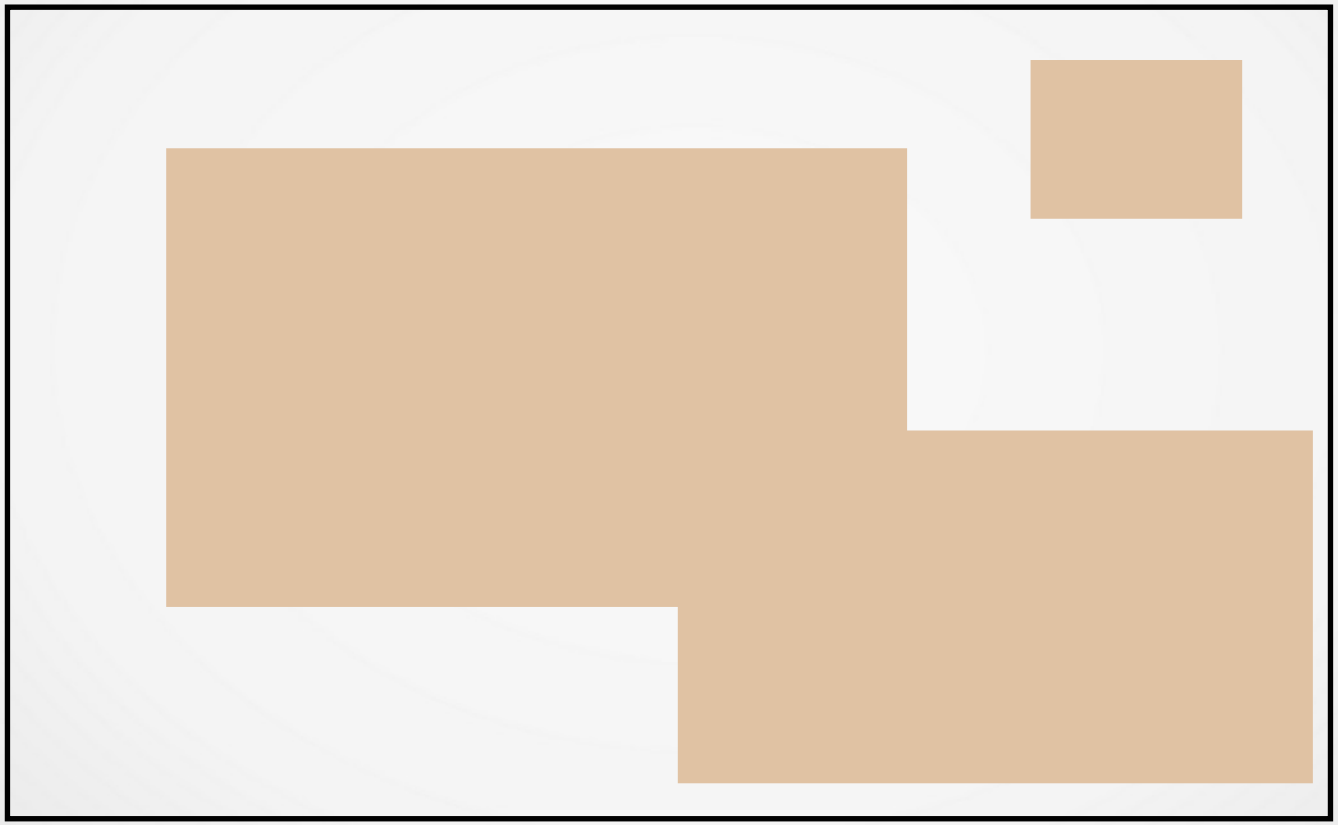
Contiguous subset or sub-array

Regular Hyperslab



Collection of regularly spaced blocks of equal size

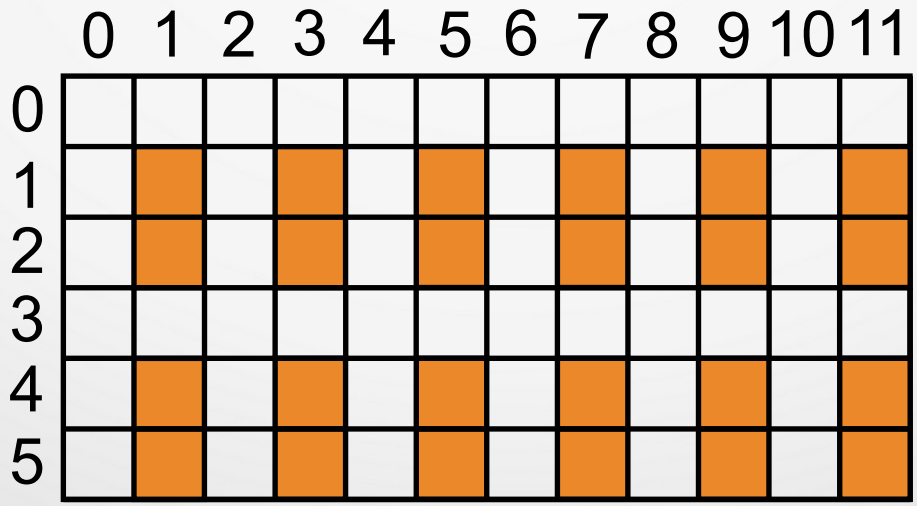
Hyperslab Selection



Result of union operation on three simple hyperslabs

HDF5 Hyperslab Description

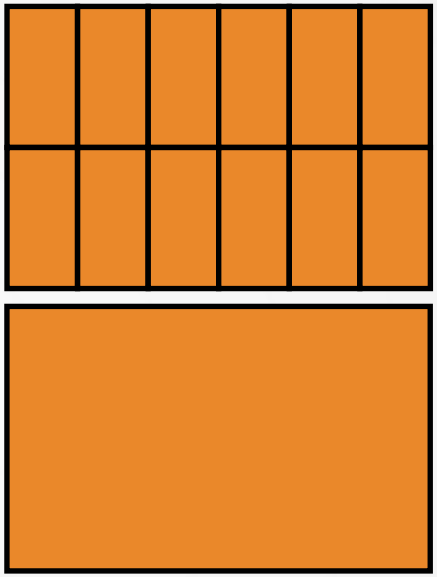
- Everything is “measured” in number of elements
  - Uses row-major ordering (C order) for coordinates
  - Example:
    - Start - starting location of a hyperslab (1,1)
    - Stride - number of elements that separate each block (3,2)
    - Count - number of blocks (2,6)
    - Block - block size (2,1)

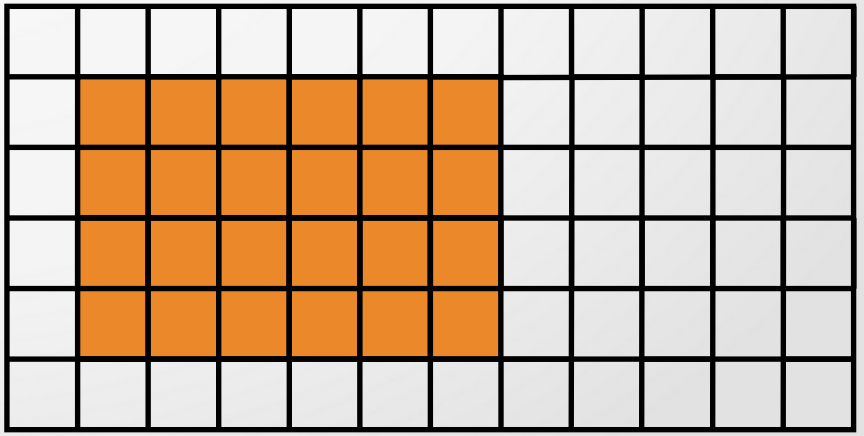


Simple Hyperslab Description

Two ways to describe a simple hyperslab

- As several blocks
  - Stride – (1,1)
  - Count – (2,6)
  - Block – (2,1)
- As one block
  - Stride – (1,1)
  - Count – (1,1)
  - Block – (4,6)

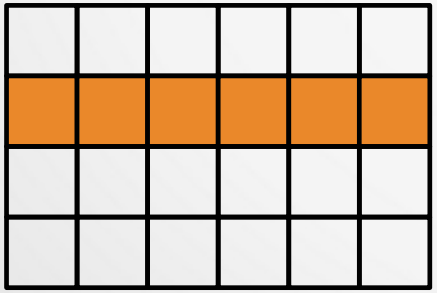




Writing a row

- Memory space selection is 1-dim array of size 6

- File space selection
  - start = {1,0}, stride = {1,1}, count = {1,6}, block = {1,1}



- Number of elements selected in memory must be the same as selected in the file

Writing a row

```
// …
hsize_t dims[1] = {6};
hid_t mspace_id = H5Screate_simple(1, dims, NULL);


hid_t fspace_id = H5Dget_space(dataset_id);


hsize_t start[2] = {1, 0};
hsize_t count[2] = {1, 6};
H5Sselect_hyperslab(fspace_id, H5S_SELECT_SET, start, NULL, count,
                    NULL);


H5Dwrite(dataset_id, H5T_NATIVE_INT, mspace_id, fspace_id,
         H5P_DEFAULT, wdata);
```

Parallel HDF5

- Open files with a MPI communicator
  - Most file operations become a MPI collective
  - All parts of the file are accessible by all processes
  - All objects in the file are accessible by all processes
  - Multiple processes can write to the same dataset
- Use partial I/O to write your local data block
  - But no overlap !

//HDF5 : Opening a file

For all processes of the MPI communicator :

- Set up an access template object
  - hid_t H5Pcreate(H5P_FILE_ACCESS)
- Set up the access template for parallel I/O
  - `H5Pset_fapl_mpio(hid_t fapl_id, MPI_Comm comm, MPI_Info info)`
    - `comm` & `info` passed to `MPI_FILE_OPEN`
    - `Info` : usually `MPI_INFO_NULL`
- Open the file using the access template
- Close the file

Parallel HDF5 : Example

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

hid_t file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT,
                          plist_id);

// …

H5Pclose(plist_id);

H5Fclose(file_id);

MPI_Finalize();
```

Parallel dataset access

- Create a file transfer property list
  - `hid_t H5Pcreate(H5P_DATASET_XFER)`
- Set the data transfer mode to independent I/O collective I/O
  - `H5Pset_dxpl_mpio (hid_t dxpl_id, H5FD_mpio_xfer_t xfer_mode)`
    - `dxpl_id`: Data transfer property list identifier
    - `xfer_mode`: Transfer mode:
      - `H5FD_MPIO_INDEPENDENT`: independent I/O access
      - `H5FD_MPIO_COLLECTIVE`: collective I/O access
- Access the dataset with the defined transfer property list
  - All processes that have opened a dataset may do collective I/O.
  - Each process may do an independent and arbitrary number of data I/O access calls
    - `H5Dwrite / H5Dread`

Parallel access: example

```
// …
hid_t filespace = H5Screate_simple(RANK, dimsf, NULL);
hid_t dset_id = H5Dcreate(file_id, DATASETNAME, H5T_NATIVE_INT,
                          filespace, H5P_DEFAULT, H5P_DEFAULT,
                          H5P_DEFAULT);
hid_t plist_id = H5Pcreate(H5P_DATASET_XFER);
                 H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

H5Dwrite(dset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, plist_id, data);

H5Dclose(dset_id);
H5Sclose(filespace);
H5Pclose(plist_id);
```

Your turn now

- Get the TD description :

https://raw.githubusercontent.com/jbigot/glcs_2020-2021/master/td1.pdf

- Set-up your environment :
  - Install docker

```
sudo apt install docker.io
sudo gpasswd -a $USER docker
```

  - Get the TD environment script

```
wget https://raw.githubusercontent.com/jbigot/glcs_2020-2021/master/glcs-td1.sh
```

  - Generate the TD directory

```
bash glcs-td1.sh
```