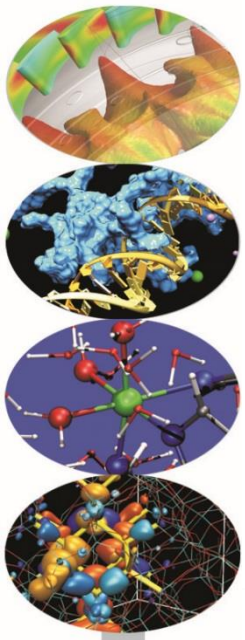
CINECA SCAI
SuperComputing Applications and Innovation

HDF5: theory & practice/1



Giorgio Amati, Mario Tacconi
SCAI Dept.

Cineca
TRAINING
High Performance
Computing 2016

CINECA

Agenda

- ✓ HDF5: introduction
- ✓ Using the API (serial)
- ✓ Using the API (parallel)
- ✓ Tools
- ✓ Some comments

HFD5: some history…

- **Hierarchical Data Format** is a set of file formats and libraries designed to store and organize large amounts of numerical data
- Originally developed at the NCSA, it is supported by the non-profit HDF Group (www.hdfgroup.org), whose mission is to ensure continued development of HDF5 technologies
- There are two major versions of HDF
  - ✓ HDF4 (old, but still supported)
  - ✓ HDF5
- Last HDF5 releases:
  - ✓ 1.10.0 (first release of the new minor revision 1.10)
  - ✓ 1.8.16 (last release of the minor revision 1.8)

HFD5: some history…

- **Hierarchical Data Format** is a hierarchical, filesystem-like data format.
  - ✓ resources in an HDF5 file are accessed using the syntax */path/to/resource*. Metadata are stored in the form of user-defined, named attributes attached to groups and datasets.
  - ✓ More complex storage APIs representing images and tables can then be built up using datasets, groups and attributes.

h5dump: tool for HDF5

h5dump [-h] [-bb] [-header] [-a] [-d <names>] [-g <names>] [-l <names>] [-t <names>] <file>

-h	Print information on this command.
-header	Display header only; no data is displayed.
-a <names>	Display the specified attribute(s).
-d <names>	Display the specified dataset(s).
-g <names>	Display the specified group(s) and all the members.
-l <names>	Displays the value(s) of the specified soft link(s).
-t <names>	Display the specified named datatype(s).

<names> is one or more appropriate object names.

- You can also use **`h5ls`**

A look inside an hdf5 file/1

- **h5dump -H u_00001000.h5**

```
HDF5 "u_00001000.h5" {
GROUP "/" {
   GROUP "field" {
      DATASET "rho" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 64, 1, 64 ) / ( 64, 1, 64 ) }
      }
      DATASET "u" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 64, 1, 64 ) / ( 64, 1, 64 ) }
      }
      DATASET "v" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 64, 1, 64 ) / ( 64, 1, 64 ) }
      }
   }
.. ..
```

A look inside an hdf5 file/2

- **h5dump -H u_00001000.h5**

```
…
   GROUP "time" {
      DATASET "timestep" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
   }
}
}
```

- **h5dump -d /time/timestep u_00001000.h5**

```
HDF5 "u_00001000.h5" {
DATASET "/time/timestep" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   DATA {
   (0): 1000
   }
```

A look inside an hdf5 file/3

```
h5dump -o rho.txt -d /field/rho  u_00001000.h5
```

```
$ cat rho.txt  | head

   (0,0,0): 0.99735, 0.997346, 0.997344, 0.997345, 0.997349, 0.997359,
   (0,0,6): 0.997376, 0.9974, 0.997431, 0.997471, 0.997519, 0.997577,
   (0,0,12): 0.997642, 0.997716, 0.997799, 0.99789, 0.99799, 0.998097,
   (0,0,18): 0.998212, 0.998335, 0.998464, 0.9986, 0.998743, 0.998891,
   (0,0,24): 0.999045, 0.999204, 0.999367, 0.999534, 0.999704, 0.999877,
   (0,0,30): 1.00005, 1.00023, 1.00041, 1.00058, 1.00076, 1.00093, 1.00111,
   (0,0,37): 1.00128, 1.00145, 1.00161, 1.00177, 1.00192, 1.00207, 1.00221,
   (0,0,44): 1.00235, 1.00248, 1.0026, 1.00271, 1.00281, 1.0029, 1.00299,
   (0,0,51): 1.00306, 1.00313, 1.00319, 1.00324, 1.00328, 1.00331, 1.00334,
...
```

- With **`h5import`** you can import also data in existing hdf5 file

Endiannes: no more problem

- BiGEndian vs. LittleEndian
- Now we don't care anymore about endianness

- Galileo (Intel, LittleEndian)

```
[gamati01@node165 ~]$ h5dump sample.h5 | grep "(95,0,30)"
      (95,0,30):0.0956433, 0.095775, 0.0958987, 0.0960148, 0.0961237,
      (95,0,30):0.0956738, 0.0958036, 0.0959251, 0.0960391, 0.096146,
      (95,0,30):0.0956364, 0.0957697, 0.0958949, 0.0960127, 0.0961234,
```

- Fermi (IBM, BigEndian)

```
[gamati01@fen08 ~]$ h5dump sample.h5 | grep "(95,0,30)"
      (95,0,30):0.0956433, 0.095775, 0.0958987, 0.0960148, 0.0961237,
      (95,0,30):0.0956738, 0.0958036, 0.0959251, 0.0960391, 0.096146,
      (95,0,30):0.0956364, 0.0957697, 0.0958949, 0.0960127, 0.0961234,
```

Abstract Data Model

- The abstract data model (ADM) defines concepts for defining and describing complex data stored in files.
- The ADM is a very general model which is designed to conceptually cover many specific models.
- Many different kinds of data can be mapped to objects of the ADM, and therefore stored and retrieved using HDF5.
- The ADM is not, however, a model of any particular problem or application domain. Users need to map their data to the concepts of the ADM.

✓ NETCDF → uses HDF5 under the hood

Abstract Data Model

- ✓ **File**: a contiguous string of bytes in a computer storage device (memory, disk, etc.), and the bytes represent zero or more objects of the model
- ✓ **Group**: a collection of objects (including groups)
- ✓ **Dataset**: a multidimensional array of data elements with attributes and other metadata
- ✓ **Dataspace**: a description of the dimensions of a multidimensional array
- ✓ **Datatype**: a description of a specific class of data element including its storage layout as a pattern of bits
- ✓ **Attribute**: a named data value associated with a group, dataset, or named datatype
- ✓ **Property List:** a collection of parameters (some permanent and some transient) controlling options in the library
- ✓ **Link** - the way objects are connected

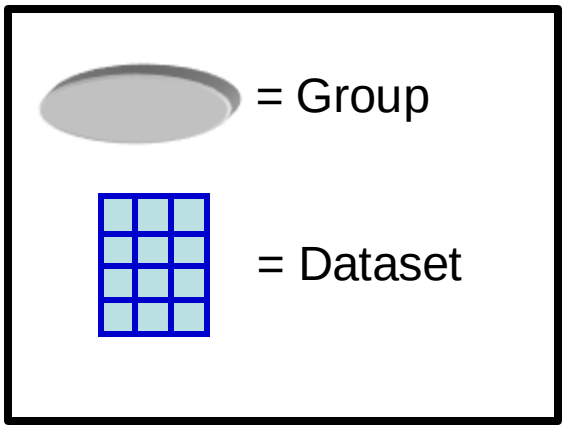
HDF5 file

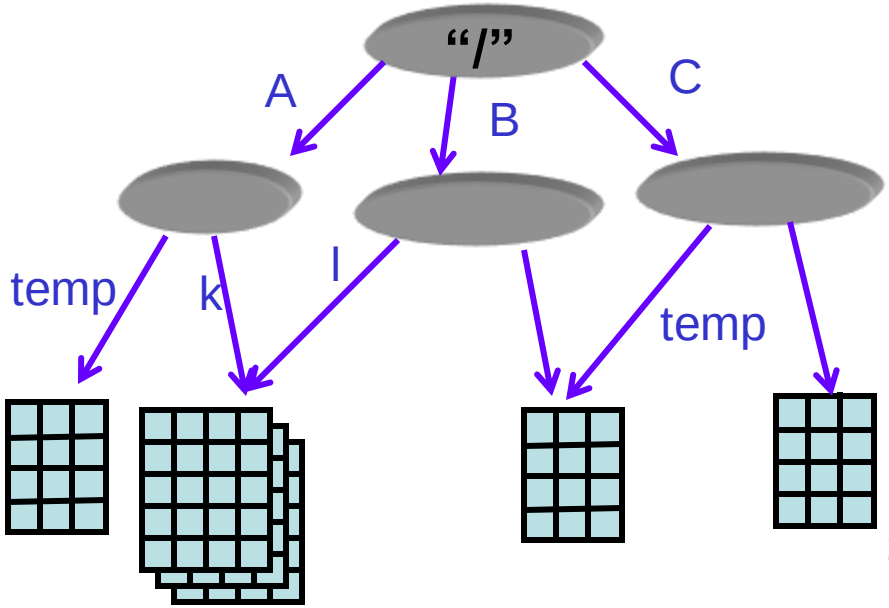
- An HDF5 file is a "*container*" for storing a variety of (scientific) data
- Is composed of two primary types of objects:
  - ✓ Groups: a grouping structure containing zero or more HDF5 objects, together with supporting metadata
  - ✓ Datasets: a multidimensional array of data elements, together with supporting metadata
- Any HDF5 group or dataset may have an associated attribute list
  - ✓ Attribute: a user-defined HDF5 structure that provides extra information about an HDF5 object.

HDF5 Groups

grouping structure containing zero or more HDF5 objects

- ✓ Used to organize collections
- ✓ Every file starts with a root group
- ✓ Similar to UNIX directories
- ✓ Path to object defines it
- ✓ Objects can be shared: /A/k and /B/l are the same

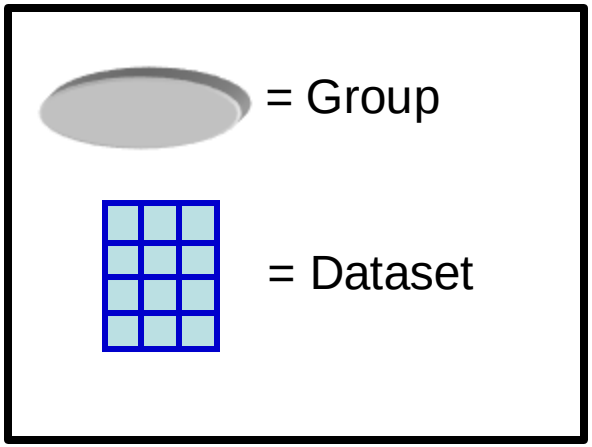


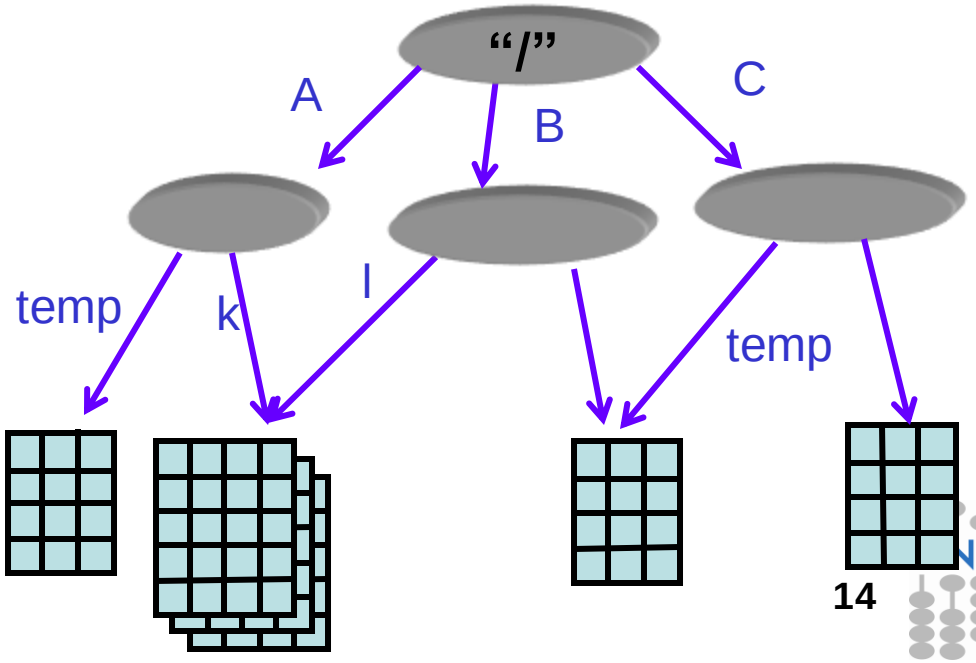


HDF5 Groups

HDF5 objects are identified and located by their pathnames:

- `/` (the root group)
- `/A` (a member of the root group called A)
- `/A/temp` (a member of the group A, which is itself a member of the root group)
- `/A/k` & `/B/l` are the same object





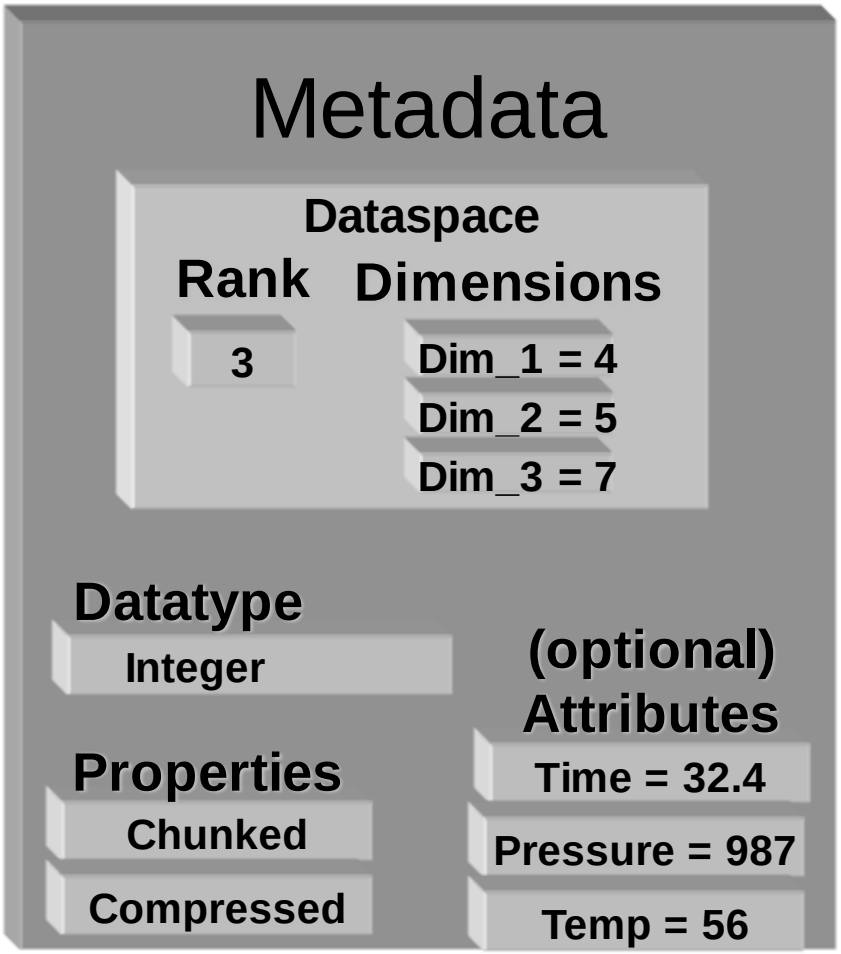
HDF5 Dataset

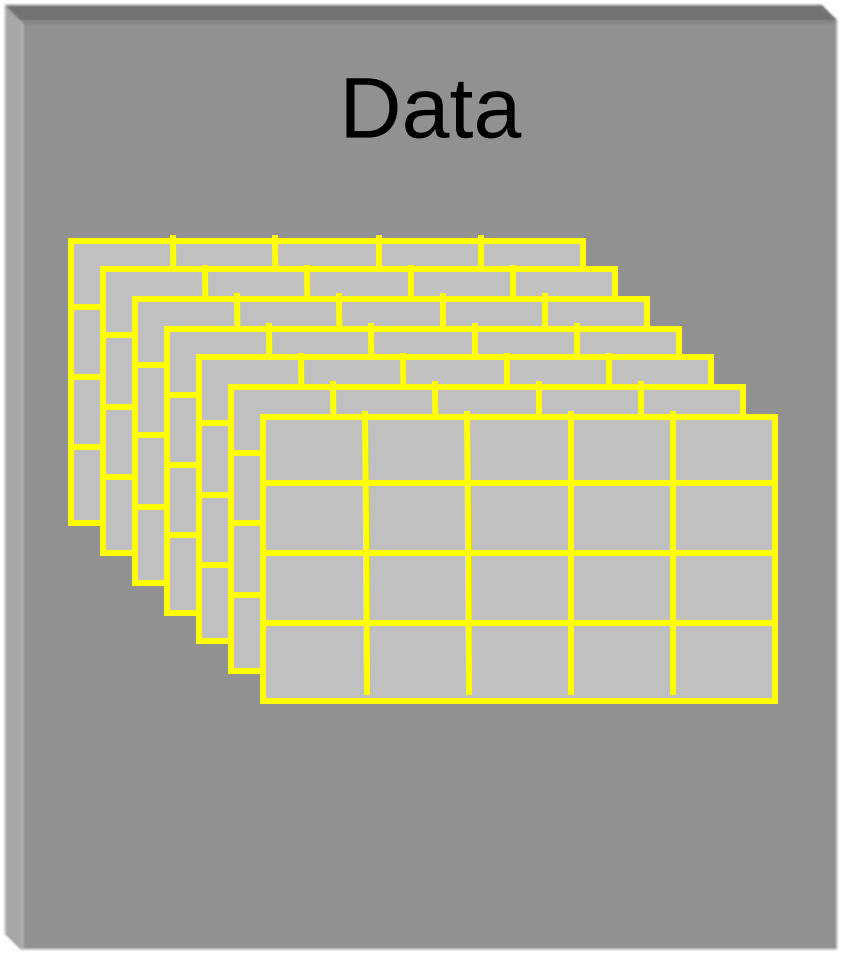
Dataset are object used to organize and contain your "*raw data values*".

They consist of:

- Raw data
- Metadata describing the raw data:
  - ✓ **Dataspace:** information to describe the logical layout of the data elements
  - ✓ **Datatype:** information to interpret the data
  - ✓ **Property list:** allows to fine-tune the dataset object behavior
  - ✓ **Attributes :** additional optional information that describes the data

HDF5 Dataset





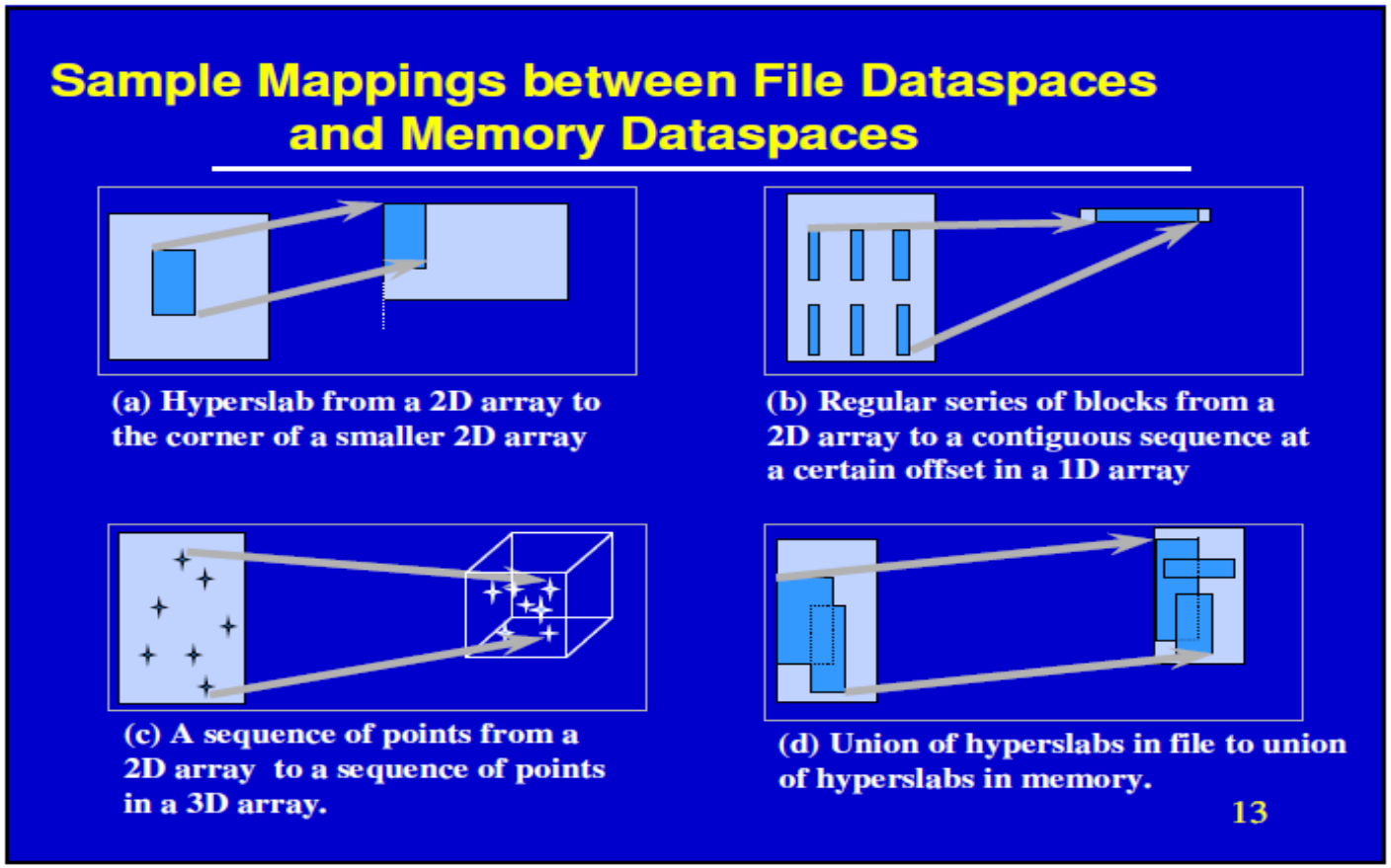
HDF5 Dataspaces

An HDF5 Dataspace describes the logical layout for the data elements:

- **Array**
  - ✓ multiple elements in dataset organized according to a multi-dimensional (rectangular) array
  - ✓ maximum number of dimensions: 32
  - ✓ maximum number of elements in each dimension may be fixed or even unlimited
- **Scalar**
  - ✓ single element in dataset
- **Null**
  - ✓ no elements in dataset

HDF5 Dataspaces

- ✓ **Memory dataspace**: logical layout of data in memory
- ✓ **File dataspace**: logical layout of data in the file

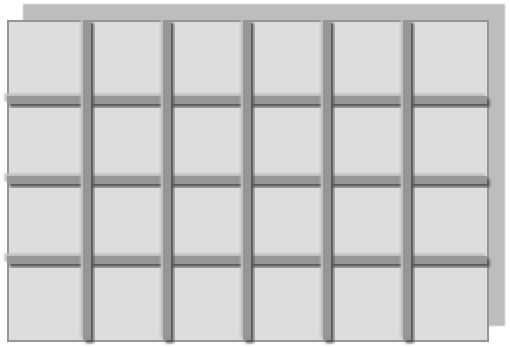


HDF5 Dataspaces

- Dataspace –> spatial info about a dataset
  - Rank and dimensions
    - ✓It is a permanent part of the dataset definition
  - Subset of points, for partial I/O
    - ✓Needed only during I/O operations

- Apply to datasets

Rank =2
Dimensions = 4 x 6

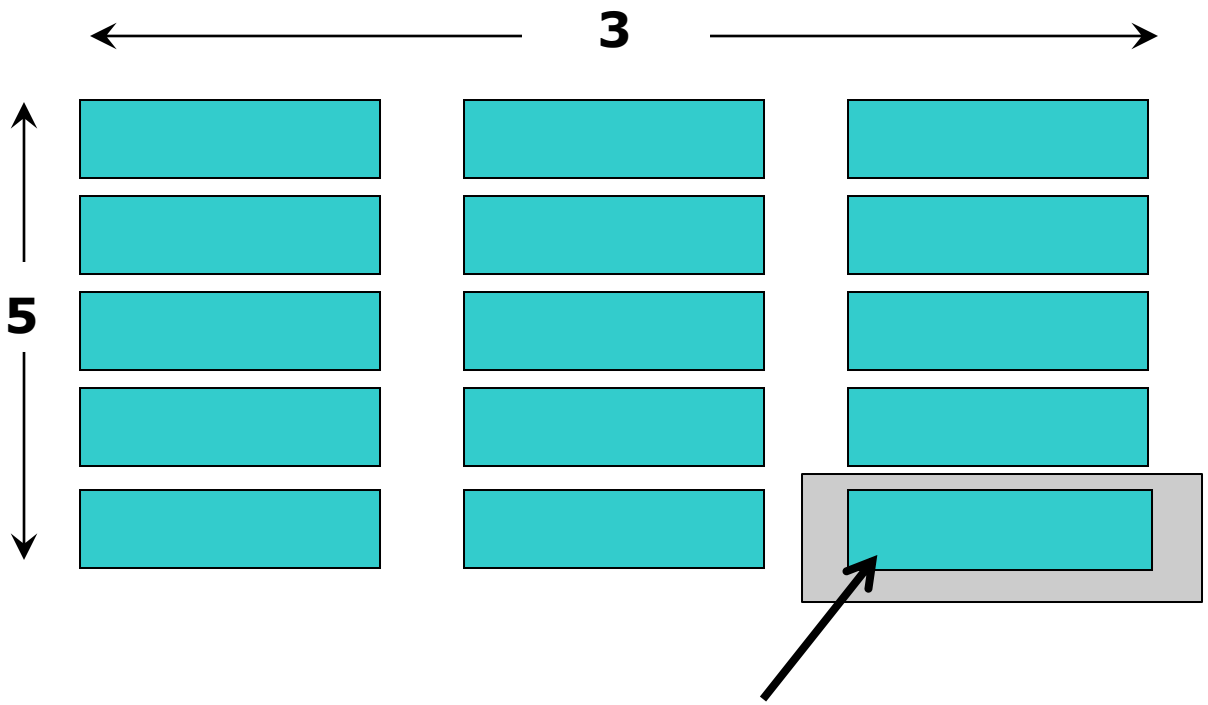


HDF5 Datatypes

- HDF5 datatype describes how to interpret data elements.
- HDF5 datatypes include:
  - ✓ integer, float, unsigned, bitfield, …
  - ✓ user-definable (e.g., 13-bit integer)
  - ✓ variable length types (e.g., strings)
  - ✓ references to objects/dataset regions
  - ✓ enumerations (e.g. names mapped to integers)
  - ✓ opaque (un-interpreted data, sequence of bits)
  - ✓ compound (similar to "C" structs or "**Fortran**" derived types and common blocks)

HDF5 Dataset

Dataspace: Rank = 2, Dimensions = 5x3



Datatype: 16-byte integer

HDF5 Properties

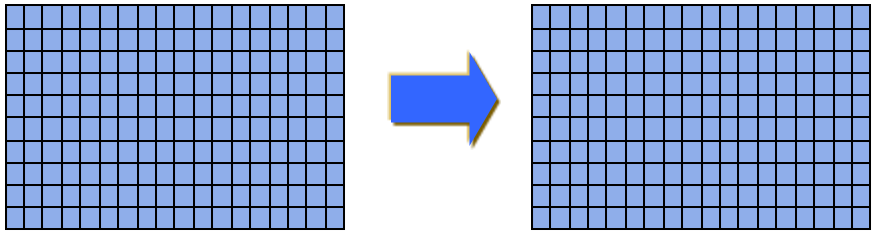
- Properties (also known as Property Lists) are HDF5 objects characteristics objects, that can be modified/added
- Usually default properties handle most needs
- By changing properties one can take advantage of the more advanced powerful features in HDF5 (i.e. compression)

HDF5 Properties: example

- HDF5 Dataset properties
  - ✓ I/O and Storage Properties (filters)
- HDF5 File properties
  - ✓ I/O and Storage Properties (drivers)
- HDF5 Datatypes properties
  - ✓ Compound
  - ✓ Variable Length
  - ✓ Reference to object and dataset region

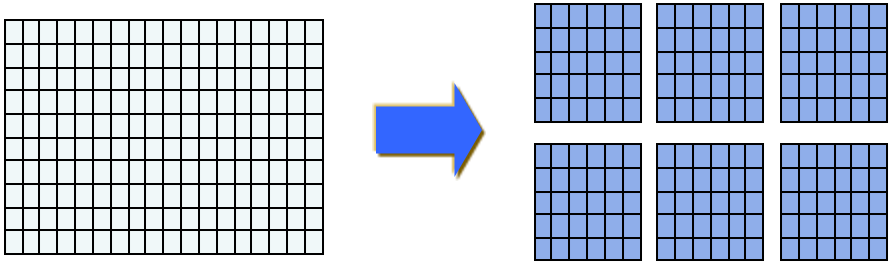
Storage Properties

Contiguous (default)



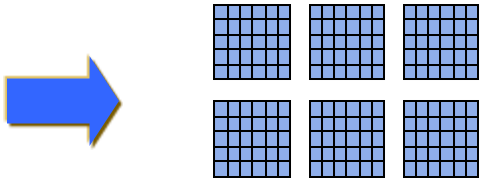
Data elements stored physically adjacent to each other

Chunked



Better access time for subsets; extensible

Chunked & Compressed



Improves storage efficiency, transmission speed

HDF5 Attributes

- An HDF5 attribute has a name and a value
- Attributes typically contain user metadata
- Attributes may be associated with
  - ✓ HDF5 groups
  - ✓ HDF5 datasets
  - ✓ HDF5 named datatypes
- An attribute's value is described by a datatype and a dataspace
- Attributes are analogous to datasets except…
  - ✓ they are NOT extensible
  - ✓ they do NOT support compression or partial I/O

Agenda

- ✓ HDF5: introduction
- ✓ Using the API (serial)
- ✓ Using the API (parallel)
- ✓ Tools
- ✓ Some comments

The General HDF5 API

- The HDF5 library provides several interfaces, or APIs.
  - ✓ These APIs provide routines for creating, accessing, and manipulating HDF5 files and objects.
- The library itself is implemented in C.
  - ✓ To facilitate the work of F90, F2003, C++ and Java programmers, HDF5 function wrappers have been developed in each of these languages.
- All C routines in the HDF5 library begin with a prefix of the form **H5***, where * is one or two uppercase letters indicating the type of object on which the function operates
- The FORTRAN wrappers come in the form of subroutines that begin with **h5** and end with **_f**
- Example APIs:
  - **H5D** : **D**ataset interface e.g. **H5Dread**
  - **H5F** : **F**ile interface e.g. **H5Fopen**
  - **H5S** : data**S**pace interface e.g. **H5Sclose**

Order of Operations

- The library imposes an order on the operations by argument dependencies
  - ✓ e.g. A file must be opened before a dataset because the dataset open call requires a file handle as an argument
- Remember always to close objects to avoid memory leaks
  - ✓ No warning about it….
- Objects can be closed in any order. However, using a closed object will result in an error
- In Fortran the HDF5 library must be explicitly initialized (**`h5open_f`**) and finalized (**`h5close_f`**)

HDF5 Programming issue

For portability, HDF5 library has its own defined types:

`hid_t`: object identifiers (native integer)

`hsize_t`: size used for dimensions (unsigned long or insigned long long)

`hssize_t`: for specifying coordinates and sometimes for dimensions (signed long or signed long long)

`herr_t`: function return value

`hvl_t`: variable length datatype

In order to give access to your application to the HDF5 types and API you need to:

- include the header file in C: `#include <hdf5.h> /* C API */`
- Use the HDF5 module in Fortran: `use hdf5 ! Fortran 95 API`

Create an HDF5 File

- To create an HDF5 file, you must specify
  - ✓ a file name;
  - ✓ a file access mode;
  - ✓ a file creation property list;
  - ✓ a file access property list.
- The steps to create and close an HDF5 file are as follows:
  1. Specify File Creation and Access property lists, if necessary
  2. Create a file
  3. Close the file and property lists, if necessary

File access mode

When creating a file, the file access mode defines the HDF5 runtime behavior during the file creation process:

- ✓ **H5F_ACC_TRUNC**: if the file already exists, the current contents will be deleted so that the application can **rewrite** the file with new data.
- ✓ **H5F_ACC_EXCL:** open will **fail** if the file already exists. If the file does not already exist, the file access parameter is ignored.
- ✓ In either case, the application has both read and write access to the successfully created file.

There are two different access modes for opening existing files:

- ✓ **H5F_ACC_RDONLY:** specifies that the application has read access but will **not be allowed to write** any data.
- ✓ **H5F_ACC_RDWR:** specifies that the application **has read and write access.**

File access/creation property lists

- HDF5 object have a series of default properties. A property list is an object that allows the modification of the default properties of a HDF5 object.
- **File Creation Property List**
  - Controls file metadata: information about size of the user-block, size of file data structures, etc.
  - Specifying **H5P_DEFAULT** uses the default values
- **File Access Property List**
  - Controls different methods of performing I/O on files
  - Unbuffered I/O, parallel I/O, etc.
  - Specifying **H5P_DEFAULT** uses the default value

C:Binding of H5Fcreate

```
hid_t H5Fcreate(const char *name, unsigned
  flags, hid_t create_id, hid_t access_id)
```

- ✓ **In) name:** Name of the file to access
- ✓ **In) flags:** File access flags
- ✓ **In) create_id:** File creation property list identifier
- ✓ **In) access_id:** File access property list identifier

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

- ✓ **In) file_id:** Identifier of the file to terminate access

Fortran:Binding of H5Fcreate

```
call h5fcreate_f(file_name, access_flag,
  file_id, hdferr, create_id, access_id)
```

- ✓ **In) name:** Name of the file to access
- ✓ **In) access_flag:** File access flags
- ✓ **Out) file_id:** File identifier
- ✓ **Out) hfderr:** error code
- ✓ **In) create_id:** File creation property list identifier
- ✓ **In) access_id:** File access property list identifier

```
call h5fclose_f(file_id, hdferr)
```

- ✓ **In) file_id:** Identifier of the file to terminate access
- ✓ **Out) hdfderr:** error code

Info about HDF5

- Simple way to take a look to the HD5 library you are using

```
h5cc -showconfig
            SUMMARY OF THE HDF5 CONFIGURATION
            =================================
General Information:
                   HDF5 Version: 1.8.11
…
Languages:
                 Fortran: yes
…
Features:
               Parallel HDF5: no
               High Level library: yes
                     Threadsafety: no
               Default API Mapping: v18
…
```

How to compile

- Compilation asks for a lot of options/libraries to include
- Use a wrapper!
- Serial:
  - ✓ **h5cc filename.c**
  - ✓ **h5fc filename.f90**
- Parallel:
  - ✓ **h5pcc filename.c**
  - ✓ **h5pfc filename.f90**

```
h5cc -show
gcc -O3 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_BSD_SOURCE -
L/home/giorgio/LOCAL_SW/hdf5_serial/lib
/home/giorgio/LOCAL_SW/hdf5_serial/lib/libhdf5_hl.a
/home/giorgio/LOCAL_SW/hdf5_serial/lib/libhdf5.a -ldl -lm -Wl,-rpath
-Wl,/home/giorgio/LOCAL_SW/hdf5_serial/lib
```

Example 1 (C)

```
#include <hdf5.h>
int main() {
char H5FILE_NAME[128];
hid_t file_id; /* file identifier */
herr_t status;
/* filename */
sprintf(H5FILE_NAME,"RUN/%s.h5","my_first_file");
printf("creating h5 file: %s...", H5FILE_NAME);
/* Create a new file using default properties. */
file_id = H5Fcreate
   (H5FILE_NAME,H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAULT);
/* Terminate access to the file. */
status = H5Fclose (file_id);
printf(" ...done\n");
}
```

Example 1 (F90)

```
      use HDF5
!
      integer(hid_t) :: file_id       !! file identifier
      integer        :: hdferr
!
      write(6,*) "writing h5 file ", file_name
!
! open the fortran interface for hdf5
      call h5open_f(hdferr)
! create the file using default properties.
      call h5fcreate_f(file_name,H5F_ACC_TRUNC_F,file_id, hdferr, &
                       H5P_DEFAULT_F, H5P_DEFAULT_F)
! close the file
      call h5fclose_f(file_id, hdferr)
! close the interface
      call h5close_f(hdferr)
```

Example 1: h5dump output

```
~/file my_first_file.h5
my_first_file.h5: Hierarchical Data Format
(version 5) data

~/h5dump my_first_file.h5
HDF5 "my_first_file.h5" {
GROUP "/" {
}
}
```

Example 1: file access

```
H5Fcreate(H5FILE_NAME,H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAULT);
```

- **H5F_ACC_TRUNC**: Truncate file, if it already exists, erasing all data previously stored in the file.
- **H5F_ACC_EXCL**: Fail if file already exists. **H5F_ACC_TRUNC** and **H5F_ACC_EXCL** are mutually exclusive
- **H5F_ACC_RDONLY**: Open file as read-only, if it already exists, and fail, otherwise
- **H5F_ACC_RDWR**: Open file for read/write, if it already exists, and fail, otherwise

Error handling (serial_ex1.better.c)

Good practice: always check the results of an operation

```
#include <assert.h>
…
file_id = H5Fcreate(H5FILE_NAME,H5F_ACC_EXCL,H5P_DEFAULT,H5P_DEFAULT);
assert(file_id != -1);
…
/* Terminate access to the file. */
status = H5Fclose (file_id);
if (status != 0) {
    printf(".... Error in H5Fclose\n");
  } else {
    printf(" ... done\n");
  }
```

Use Groups

- HDF5 groups provide a mechanism for organizing meaningful and extendable sets of datasets within an HDF5 file.
- An HDF5 group is a structure containing zero or more HDF5 objects.
- To create a group, the calling program must:
  1. Obtain the location identifier where the group is to be created
  2. Create the group
  3. Close the group

Binding of H5Gcreate

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

- ✓ **loc_id**: file or parent group identifier
- ✓ **name**: absolute or relative name of the new group
- ✓ **lcpl_id**: Link creation property list identifier
- ✓ **gcpl_id**: Group creation property list identifier
- ✓ **gapl_id**: Group access property list identifier (No group access properties have been implemented at this time; use **H5P_DEFAULT**.)

Example 2 (C)

```
int main() {
char H5FILE_NAME[128];
hid_t file_id;  /* file identifier */
hid_t group_id; /* group identifier */
herr_t status;
/* filename */
sprintf(H5FILE_NAME,"RUN/%s.h5","my_second_file");
printf("creating h5 file: %s with group...",H5FILE_NAME);
/* Create a new file using default properties. */
file_id =
   H5Fcreate(H5FILE_NAME,H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAULT);
/* Create a group named "/MyGroup" in the file. */
group_id =
   H5Gcreate(file_id,"/MyGroup",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
/* Close the group. */
status = H5Gclose(group_id);
/* Terminate access to the file. */
status = H5Fclose (file_id);
```

Example 2: h5dump Output

```
~/h5dump my_second_file.h5
HDF5 "my_second_file.h5" {
GROUP "/" {
    GROUP "MyGroup" {
    }
}
}
```

Open an existing Group

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

- **loc_id**: File or group identifier specifying the location of the group to be opened
- **name**: Name of the group to open
- **gapl_id**: Group access property list identifier (No group access properties have been implemented at this time; use **H5P_DEFAULT**.)

Absolute & Relative Names

- To create an HDF5 object, we have to specify the location where the object is to be created. This location is determined by the identifier of an HDF5 object and the name of the object to be created.
- The name of the created object can be either an absolute name or a name relative to the specified identifier.
- HDF5 object names are a slash-separated list of components:
  - ✓ component names may be any length except zero and may contain any character except slash (/) and the null terminator.
  - ✓ a full name may be composed of any number of component names separated by slashes, with any of the component names being the special name . (a dot or period).
  - ✓ A name which begins with a slash is an absolute name which is accessed beginning with the root group of the file; all other names are relative names and the named object is accessed beginning with the specified group.

Example 3 (C)

```
…
hid_t group_id;              /* group identifier */
hid_t group1_id, group2_id; /* sub-group identifier */
herr_t status;
…
/* Create a group named "/MyGroup" in the file. */
group_id = H5Gcreate(file_id,
   "/MyGroup",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
/* Create sub-group "Group_A" in group "MyGroup" using absolute name.
   */
group1_id = H5Gcreate(file_id, "/MyGroup/Group_A", H5P_DEFAULT,
   H5P_DEFAULT, H5P_DEFAULT);
/* Create sub group "Group_B" in group "MyGroup" using relative name.
   */
group2_id = H5Gcreate(group_id, "Group_B", H5P_DEFAULT, H5P_DEFAULT,
   H5P_DEFAULT);
/* Close all the groups. */
status = H5Gclose(group_id);
status = H5Gclose(group1_id);
status = H5Gclose(group2_id);
…
```

Example 3: h5dump Output

```
~/h5dump my_third_file.h5
HDF5 "my_third_file.h5" {
GROUP "/" {
   GROUP "MyGroup" {
      GROUP "Group_A" {
      }
      GROUP "Group_B" {
      }
   }
}
}
```

Exercise

Create an h5 file with the following structure:

```
.
|-- 100
|   |-- PRESSURE
|   `-- VELOCITY
|-- 200
|   |-- PRESSURE
|   `-- VELOCITY
|-- 300
|   |-- PRESSURE
|   `-- VELOCITY
`-- PARAMETERS
    |-- SIZE
    `-- VISCOSITY
```

Exercise: (a) solution

```
#include <hdf5.h>
int main() {
char H5FILE_NAME[128];
hid_t file_id;              /* file identifier */
hid_t group1_id, group2_id, group3_id, groupP_id;  /* group identifier */
hid_t group1P_id, group1V_id; /* sub-group identifier */
hid_t group2P_id, group2V_id; /* sub-group identifier */
hid_t group3P_id, group3V_id; /* sub-group identifier */
hid_t groupPS_id, groupPV_id; /* sub-group identifier */
herr_t status;

/* filename */
sprintf(H5FILE_NAME,"RUN/%s.h5","my_file");
printf("creating h5 file: %s with group...",H5FILE_NAME);

/* Create a new file using default properties. */
file_id = H5Fcreate (H5FILE_NAME,H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAULT);
```

Exercise: (a) solution

```
/* Create a group  */
group1_id = H5Gcreate(file_id, "/100",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
group2_id = H5Gcreate(file_id, "/200",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
group3_id = H5Gcreate(file_id, "/300",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
groupP_id = H5Gcreate(file_id, "/PARAMETERS",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);

/* Create sub-group */
group1P_id = H5Gcreate(file_id, "/100/PRESSURE", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
group2P_id = H5Gcreate(file_id, "/200/PRESSURE", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
group3P_id = H5Gcreate(file_id, "/300/PRESSURE", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
groupPS_id = H5Gcreate(file_id, "/PARAMETERS/SIZE", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

group1V_id = H5Gcreate(group1_id, "VELOCITY", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
group2V_id = H5Gcreate(group2_id, "VELOCITY", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
group3V_id = H5Gcreate(group3_id, "VELOCITY", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
groupPV_id = H5Gcreate(groupP_id, "VISCOSITY", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

/* Close all the groups. */
status = H5Gclose(group1_id);
```

Exercise: output

```
~/h5dump my_file.h5
HDF5 "my_file.h5" {
GROUP "/" {
   GROUP "100" {
      GROUP "PRESSURE" {
      }
      GROUP "VELOCITY" {
      }
   }
   GROUP "200" {
      GROUP "PRESSURE" {
      }
      GROUP "VELOCITY" {
      }
…
   GROUP "PARAMETERS" {
      GROUP "SIZE" {
      }
      GROUP "VISCOSITY" {
      }
…
```

That's all folks (for now)!!

- ✓ It's lunch time!!!!

Datasets/1

- A dataset is a multidimensional array of data elements, together with supporting metadata.
- To create an empty dataset (no data) the following steps need to be taken:
  1. Obtain the location id where the dataset is to be created.
  2. Define or specify the dataset characteristics:
     - ✓ Define a datatype or specify a pre-defined datatype.
     - ✓ Define a dataspace (shape of the array of the dataset).
     - ✓ Specify the property list(s) or use the default.
  3. Create the dataset.
  4. Close the datatype, the dataspace, and the property list(s) if necessary.
  5. Close the dataset.

Datasets/2

- Regarding to the definition of the dataset characteristics:
  - ✓ Define a datatype or specify a pre-defined datatype.
  - ✓ Define a dataspace.
  - ✓ Specify the property list(s) or use the default
- Note that:
  - In HDF5, datatypes and dataspaces are independent objects which are created separately from any dataset that they might be attached to.
  - Because of this, the creation of a dataset requires the definition of the datatype and dataspace.

Datatypes

- A **datatype** is a collection of properties, all of which can be stored on disk which provide a complete information for data conversion to or from that datatype.

- There are two categories of datatypes in HDF5:
  - Pre-defined: These datatypes are opened and closed by HDF5.
  - Derived: These datatypes are created or derived from the pre-defined types. (To use them, see the Datatype Interface H5T)

Standard predefined Datatype

H5T_IEEE_F64LE 8-byte, little-endian, IEEE floating-point

H5T_IEEE_F32BE 8-byte, big-endian, IEEE floating point

H5T_STD_U16BE 4-byte, big-endian, unsigned integer

H5T_STD_I32LE 4-byte, little-endian, signed two's complement integer

NOTE:

- ✓ These datatypes (DT) are the same on all platforms
- ✓ These are DT handles generated at run-time
- ✓ Used to describe DT in the HDF5 calls
- ✓ DT cannot be used to describe application data buffers

Native Predefined Datatype

Examples of predefined native types in C:

H5T_NATIVE_INT	(int)
H5T_NATIVE_FLOAT	(float)
H5T_NATIVE_UINT	(unsigned int)
H5T_NATIVE_LONG	(long)
H5T_NATIVE_CHAR	(char)

NOTE:

- ✓ These datatypes are NOT the same on all platforms
- ✓ These are DT handles generated at run-time

Dataspaces

- A dataspace describes the layout of the data array.
- A dataspace is either
  - ✓ simple dataspace: a regular N-dimensional array of data points,
  - ✓ complex dataspace: a more general collection of data points organized in another manner
- The dimensions of a dataset:
  - ✓ can be fixed (unchanging)
  - ✓ or they may be unlimited (they are extensible).
- A dataspace can also describe a portion of a dataset (hyper-slab), making it possible to do partial I/O operations on selections.

Creating a Simple Dataspace

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

- **rank**: Number of dimensions of dataspace
- **dims**: An array of the size of each dimension
- **maxdims**: An array of the maximum size of each dimension.
  - ✓ A value of H5S_UNLIMITED specifies the un-limited dimension.
  - ✓ A value of NULL specifies that *dims and maxdims* are the same.

Simple Dataspace: a regular N-dimensional array of data points

Property Lists

- Property lists are a mechanism for modifying the default behavior when creating/accessing objects.
- The following property lists can be specified when creating a dataset:
  - ✓ Dataset Creation Property List: When creating a dataset, HDF5 allows the user to specify how raw data is organized and/or compressed on disk.
  - ✓ Link Creation Property List: The link creation property list governs creation of the link(s) by which a new dataset is accessed and the creation of any intermediate groups that may be missing.
  - ✓ Dataset Access Property List: Dataset access property lists are properties that can be specified when accessing a dataset.

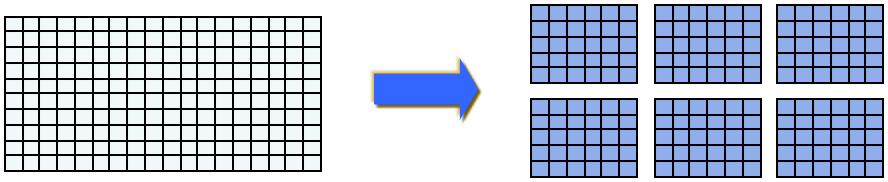
Dataset creation property list

Dataset creation property list: information on how to organize data in storage (**`H5P_DEFAULT`** → contiguous)

Contiguous (default)

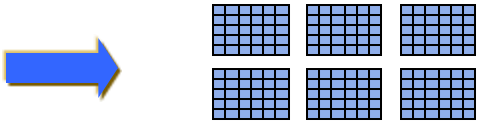
Data elements stored physically adjacent to each other

Chunked



Better access time for subsets; extensible

Chunked & Compressed



Improves storage efficiency, transmission speed

Property List example/1

- Create a link creation property list
- Allow for the creation of missing intermediate groups
- Use it to create a group and its parent groups with a single call to **H5Gcreate**

…

```
link_ppl = H5Pcreate(H5P_LINK_CREATE);
status = H5Pset_create_intermediate_group(link_ppl,1);
```

…

- The property **link_ppl** modifies how the group is created:

…

```
char gpath[]=''/this/is/a/very/long/path/to/my/GroupA'';
groupA_id = H5Gcreate(file_id,gpath,link_ppl,H5P_DEFAULT,
            H5P_DEFAULT);
```

Example 3.1 (C)

```
…
hid_t group_id;              /* group identifier */
hid_t group1_id, group2_id; /* sub-group identifier */
herr_t status;
…
/* Create a link creation property list */
link_pl = H5Pcreate(H5P_LINK_CREATE);
/* Allow for the creation of intermediate missing groups */
status = H5Pset_create_intermediate_group(link_pl, 1);

/* Create sub-group "Group_A" in group "MyGroup" using absolute name. */
group1_id=H5Gcreate(file_id,"/MyGroup/Group_A",link_pl,H5P_DEFAULT,
   H5P_DEFAULT);
/* Create sub group "Group_B" in group "MyGroup" using relative name. */
group2_id=H5Gcreate(group_id, "Group_B",link_pl,H5P_DEFAULT, H5P_DEFAULT);

/* Close all the HDF% objects */
status = H5Pclose(link_pl);
…
```

Homework...

- Like the previous exercise
- But using links ….

```
.
|-- 100
|    |-- PRESSURE
|    `-- VELOCITY
|-- 200
|    |-- PRESSURE
|    `-- VELOCITY
|-- 300
|    |-- PRESSURE
|    `-- VELOCITY
`-- PARAMETERS
     |-- SIZE
     `-- VISCOSITY
```

Creating a Dataset

```
hid_t H5Dcreate (hid_t loc_id, const char
  *name, hid_t dtype_id, hid_t space_id, hid_t
   lcpl_id, hid_t dcpl_id, hid_t dapl_id)
```

- ✓ **loc_id**: Location identifier
- ✓ **name**: Dataset name
- ✓ **dtype_id**: Datatype identifier
- ✓ **space_id**: Dataspace identifier
- ✓ **lcpl_id**: Link creation property list
- ✓ **dcpl_id**: Dataset creation property list
- ✓ **dapl_id**: Dataset access property list

Example 4 (C)

```
hid_t dataset_id;           /* dataset identifier */
hid_t dataspace_id;         /* dataspace identifier */
hid_t dcpl;                 /* dataset creation property */
hsize_t dims[2]={4,6};
…
/* create simple dataspace */
dataspace_id = H5Screate_simple (2, dims, NULL);
/* create dataset creation property list */
dcpl = H5Pcreate (H5P_DATASET_CREATE);
/* create datset */
dataset_id = H5Dcreate(file_id,"dset",H5T_STD_I32BE, dataspace_id,
   H5P_DEFAULT, dcpl, H5P_DEFAULT);
/* Close dataset et al... */
status = H5Dclose (dataset_id);
status = H5Sclose (dataspace_id);
status = H5Pclose (dcpl);
…
```

Example 4: h5dump Output

```
HDF5 "my_fourth_file.h5" {
GROUP "/" {
   GROUP "MyGroup" {
      GROUP "Group_A" {
      }
      GROUP "Group_B" {
      }
   }
   DATASET "dset" {
      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
...
```

Property List example/2

- ✓ create the dataset creation property list
- ✓ add the **gzip** compression filter (deflate)
- ✓ set the chunk size

```
create_plist_id = H5Pcreate(H5P_DATASET_CREATE);
status = H5Pset_deflate(create_plist_id,9);
status = H5Pset_chunk(create_plist_id,ndims,chunk_dims);
```

- The property **create_plist_id** will be passed when the dataset will be created

Dataset I/O operations

- During a dataset I/O operation, the library transfers raw data between memory and the file.
- Data in memory can have a different datatype from that of the file and can also have a different size (i.e., the data in memory is a subset of the dataset elements, or vice versa).
- To perform read/write operations, you must specify:
  - ✓ dataset
  - ✓ dataset's datatype in memory
  - ✓ dataset's dataspace in memory
  - ✓ dataset's dataspace in the file
  - ✓ dataset transfer property list.
  - ✓ data buffer
- Data transfer property list is used to control various aspects of the I/O, such as caching hints or collective I/O information.

Dataset I/O operations

- The steps to read from or write to a dataset are as follows:
  1. Obtain the dataset identifier.
  2. Specify the memory datatype.
  3. Specify the memory dataspace.
  4. Specify the file dataspace.
  5. Specify the transfer properties.
  6. Perform the desired operation on the dataset.
  7. Close the dataset.
  8. Close the dataspace, datatype, and property list if necessary.

Dataset I/O operations

- Dataset I/O involves
  - ✓ reading or writing
  - ✓ all or part of a dataset
  - ✓ Compressed/uncompressed
- During I/O operations data is translated between the source & destination
  - ✓ Datatype conversion
    - data types (e.g. 16-bit integer => 32-bit integer) of the same class
  - ✓ Dataspace conversion
    - dataspace (e.g. 10x20 2d array => 200 1d array)

Partial I/O

- Selected elements (called selections) from source are mapped (read/written) to the selected elements in destination
- Selection
  - ✓ Selections in memory can differ from selection in file
  - ✓ Number of selected elements is always the same in source and destination
- Selection can be
  - ✓ Hyperslabs (contiguous blocks, regularly spaced blocks)
  - ✓ Points
  - ✓ Results of set operations (union, difference, etc.) on hyperslabs or points

Open Dataset

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

- **loc_id**: Identifier of the file or group in which to open a dataset
- **name**: The name of the dataset to access

NOTE:
File datatype and dataspace are known when a dataset is opened

Write Dataset

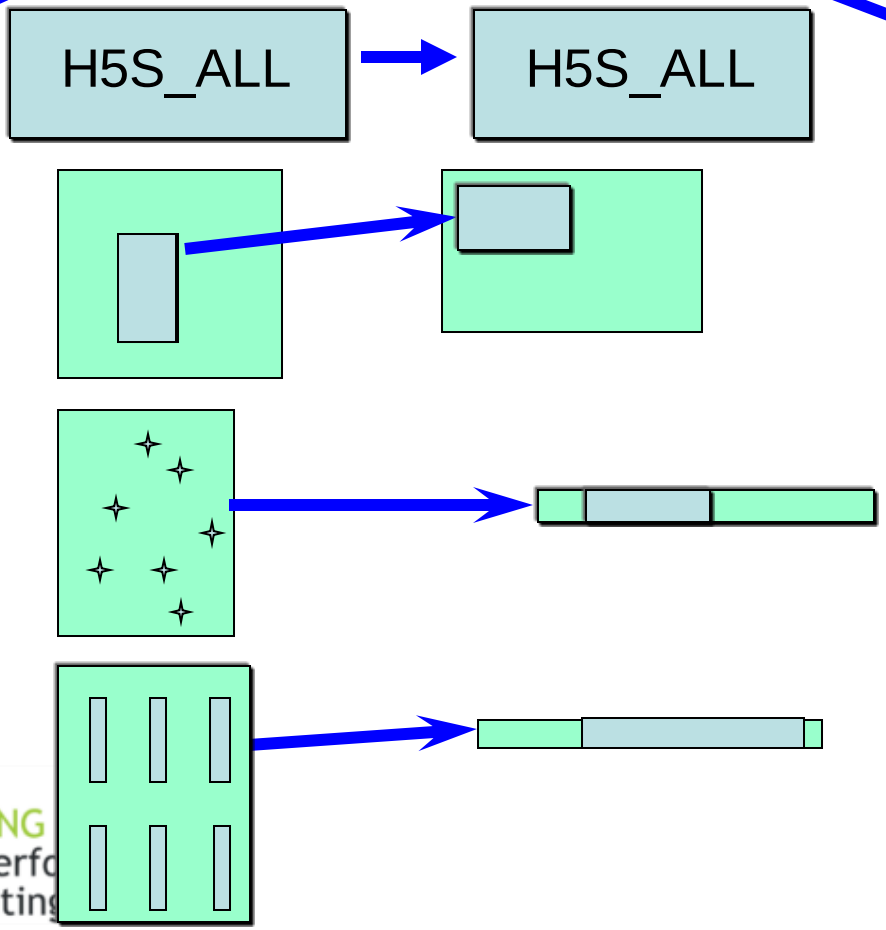
```
herr_t H5Dwrite (hid_t dataset_id, hid_t
  mem_type_id, hid_t mem_space_id, hid_t
  file_space_id, hid_t xfer_plist_id, const
  void * buf )
```

- ✓ **dataset_id**: Identifier of the dataset to write to
- ✓ **mem_type_id**: Identifier of memory datatype of the dataset
- ✓ **mem_space_id**: Identifier of the memory dataspace (or **H5S_ALL**)
- ✓ **file_space_id**: Identifier of the file dataspace (or **H5S_ALL**)
- ✓ **xfer_plist_id**: Identifier of the data transfer properties to use (or **H5P_DEFAULT**)
- ✓ **buf**: Buffer with data to be written to the file

Partial I/O

```
status = H5Dwrite (dataset_id,
   H5T_NATIVE_INT,H5S_ALL, H5S_ALL, H5P_DEFAULT, buffer);
```

Memory Dataspace



File Dataspace (disk)

To Modify Dataspace:
H5Sselect_hyperslab
H5Sselect_elements

Read Dataset

```
herr_t H5Dread (hid_t dataset_id, hid_t
  mem_type_id, hid_t mem_space_id, hid_t
  file_space_id, hid_t xfer_plist_id, const
  void * buf )
```

- ✓ **dataset_id**: Identifier of the dataset to read to
- ✓ **mem_type_id**: Identifier of memory datatype of the dataset
- ✓ **mem_space_id**: Identifier of the memory dataspace (or **H5S_ALL**)
- ✓ **file_space_id**: Identifier of the file dataspace (or **H5S_ALL**)
- ✓ **xfer_plist_id**: Identifier of the data transfer properties to use (or **H5P_DEFAULT**)
- ✓ **buf**: Buffer with data to be written to the file

Example 5 (C)

```
….
/* data to copy */
for (i = 0; i < 4; i++)
        for (j = 0; j < 6; j++)
                dset_data[i][j] = i * 6 + j + 1;
…
/* open an existing file */
file_id = H5Fopen (H5FILE_NAME,H5F_ACC_RDWR,H5P_DEFAULT);
/* open an existing dataset */
dataset_id = H5Dopen (file_id,"dset",H5P_DEFAULT);
/* Write to dataset */
status = H5Dwrite (dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
   H5P_DEFAULT, dset_data);
/* close dataset */
status = H5Dclose (dataset_id);
```

Example 5: h5dump Output

```
HDF5 "my_fourth_file.h5" {
GROUP "/" {
   GROUP "MyGroup" {
      GROUP "Group_A" {
      }
      GROUP "Group_B" {
      }
   }
   DATASET "dset" {
      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
      }
   }
}
```

About compression

```
gamati01@node013$ h5cc serial_ex4.c ; ./a.out
creating h5 file: RUN/my_fourth_file.h5 with group & dataset......done
gamati01@node013$ h5cc serial_ex5.c;  ./a.out
opening an h5 file: RUN/my_fourth_file.h5 ...and adding dataspace...done
gamati01@node013$ ls -lrt RUN/
total 4608
-rw-r--r-- 1 gamati01 interactive 4198144 16 mag 14:02 my_fourth_file.h5

gamati01@node013.pico:[TEMP]$ h5cc serial_ex4.1.c;  ./a.out
creating h5 file: RUN/my_fourth_file.h5 with group & dataset......done
gamati01@node013.pico:[TEMP]$ h5cc serial_ex5.c; ./a.out
opening an h5 file: RUN/my_fourth_file.h5 ...and adding dataspace...done
gamati01@node013.pico:[TEMP]$ ls -lrt RUN/
total 0
-rw-r--r-- 1 gamati01 interactive 93066 16 mag 14:05 my_fourth_file.h5

gamati01@node013.pico:[RUN]$ h5diff  uncompressed.h5 compressed.h5
gamati01@node013.pico:[RUN]$
```

R/W to a Subset of a Dataset

- HDF5 allows you to read from or write to a portion or subset of a dataset.
- This is done by selecting a subset of the dataspace of the dataset, and then using that selection to read from or write to the dataset.
- There are two types of selections in HDF5, hyperslab selections and element selections,
  - ✓ The **`H5Sselect_hyperslab`** call selects a logically contiguous collection of points in a dataspace, or a regular pattern of points or blocks in a dataspace.
  - ✓ The **`H5Sselect_elements`** call selects elements in an array.

H5Sselect_hyperslab

```
herr_t H5Sselect_hyperslab(hid_t space_id, H5S
  _seloper_t op, const hsize_t *start, const
  hsize_t *stride, const hsize_t *count, const
  hsize_t *block )
```

- ✓ **space_id**: Identifier of dataspace selection to modify
- ✓ **op**: Operation to perform on current selection.
- ✓ **start**: Offset of start of hyperslab
- ✓ **count**: Number of blocks included in hyperslab.
- ✓ **stride**: Hyperslab stride.
- ✓ **block**: Size of block in hyperslab.

H5Sselect_elements

```
herr_t H5Sselect_elements( hid_t space_id, H5S
  _seloper_t op, size_t num_elements, const
  hsize_t *coord )
```

- ✓ **space_id**: Identifier of the dataspace.
- ✓ **op**: Operator specifying how the new selection is to be combined with the existing selection for the dataspace.
- ✓ **num_elements**: Number of elements to be selected.
- ✓ **coord**: A pointer to a buffer containing a serialized copy of a 2-dimensional array of zero-based values specifying the coordinates of the elements in the point selection.

Write a Subset of a Dataset

1. Re-open my_fourth_file.h5 the file and the dataset
2. Creates an 2 x 4 integer dataset, starting from (1,1) position
3. write the value `1000+(i+j)` in such dataset;

Original dataset

```
 1,  2,  3,  4,  5,  6,
 7,  8,  9, 10, 11, 12,
13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24
```

New Dataset

```
 1,    2,    3,    4,    5,  6,
 7, 1000, 1001, 1002, 1003, 12,
13, 1004, 1005, 1006, 1007, 18,
19,   20,   21,   22,   23, 24
```

Example 6 (C)

```
…
hsize_t dim_sub[2],start[2],count[2];
…
/* Create the dataspace for the subset of data */
dim_sub[0] = 2; dim_sub[1] = 4;
memspace_id = H5Screate_simple (2,dim_sub,NULL);

/* Selection, in the dataspace of the dataset,
   of the hyperslab where write the new subset of data */
dataspace_id = H5Dget_space (dataset_id);
start[0] = 1; start[1] = 1;
count[0] = dim_sub[0]; count[1] = dim_sub[1];
status = H5Sselect_hyperslab (dataspace_id, H5S_SELECT_SET, start,
   NULL, count, NULL);

/* Write hyperslab to dataset */
status = H5Dwrite (dataset_id, H5T_NATIVE_INT, memspace_id,
   dataspace_id, H5P_DEFAULT, sub_data);
…
```

Example 6: h5dump Output

```
HDF5 "my_fourth_file.h5" {
GROUP "/" {
   GROUP "MyGroup" {
      GROUP "Group_A" {
      }
      GROUP "Group_B" {
      }
   }
   DATASET "dset" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }
      DATA {
      (0,0): 1, 2, 3, 4, 5, 6,
      (1,0): 7, 1000, 1001, 1002, 1003, 12,
      (2,0): 13, 1004, 1005, 1006, 1007, 18,
      (3,0): 19, 20, 21, 22, 23, 24
      }
...
```

Read a Subset of a Dataset

1. Open the file and the dataset
2. Creates an 2 x 4 integer dataset starting from (1,1) position
3. Read the value of such dataset;

Original dataset

```
 1,  2,  3,  4,  5,  6,
 7,  1000,  1001, 1002, 1003, 12,
13, 1004, 1005, 1006, 1007, 18,
19, 20, 21, 22, 23, 24
```

Values

```
1000, 1001, 1002, 1003,
1004, 1005, 1006, 1007,
```

Example 7(C)

```
…
hsize_t dim_sub[2],start[2],count[2];
…
/* Create the dataspace for the subset of data */
dim_sub[0] = 2; dim_sub[1] = 4;
memspace_id = H5Screate_simple (2,dim_sub,NULL);

/* Selection, in the dataspace of the dataset,
   of the hyperslab where write the new subset of data */
dataspace_id = H5Dget_space (dataset_id);
start[0] = 1; start[1] = 1;
count[0] = dim_sub[0]; count[1] = dim_sub[1];
status = H5Sselect_hyperslab (dataspace_id, H5S_SELECT_SET, start,
   NULL, count, NULL);

/* Write hyperslab to dataset */
status = H5Dread(dataset_id, H5T_NATIVE_INT, memspace_id,
   dataspace_id, H5P_DEFAULT, sub_data);
…
```

Example 7: Output

```
gamati01@node013.pico:[SERIAL]$ h5cc serial_ex6.c  -o ex6
gamati01@node013.pico:[SERIAL]$ h5cc serial_ex7.c  -o ex7
gamati01@node013.pico:[SERIAL]$ ./ex7
opening h5 file: RUN/my_fourth_file.h5 ......and reading an hyperslab ...
--------------------------
 8, 9, 10, 11,
14, 15, 16, 17,
--------------------------
 ...done
gamati01@node013.pico:[SERIAL]$ ./ex6
opening h5 file: RUN/my_fourth_file.h5 ......and modify a part of
   dataspace......done
gamati01@node013.pico:[SERIAL]$ ./ex7
opening h5 file: RUN/my_fourth_file.h5 and reading an hyperslab
--------------------------
1000, 1001, 1002, 1003,
1004, 1005, 1006, 1007,
--------------------------
...done
```

Attributes

Attributes are small datasets that can be used to describe the nature and/or the intended usage of the object they are attached to.

- ✓ Creating an attribute is similar to creating a dataset. To create an attribute, the application must specify the object which the attribute is attached to, the datatype and dataspace of the attribute data, and the attribute creation property list.
- ✓ Attributes may only be read or written as an entire object; no partial I/O is supported. Therefore, to perform I/O operations on an attribute, the application needs only to specify the attribute and the attribute's memory datatype.

Steps to create an attribute

The steps to create an attribute are as follows:

1. Obtain the object identifier that the attribute is to be attached to.
2. Define the characteristics of the attribute and specify the attribute creation property list.
   - ✓ Define the datatype.
   - ✓ Define the dataspace.
   - ✓ Specify the attribute creation property list.
3. Create the attribute.
4. Close the attribute and datatype, dataspace, and attribute creation property list, if necessary.

Example 8 (C)

```
/* open an existing dataset */
dataset_id = H5Dopen (file_id,"dset",H5P_DEFAULT);

/* Create the dataspace for the attribute */
dataspace_id = H5Screate_simple(1, &dims, NULL);

/* Create a dataset attribute */
attribute_id = H5Acreate(dataset_id,"attr",H5T_NATIVE_INT,
               dataspace_id,H5P_DEFAULT,H5P_DEFAULT);

/* Write the attribute data */
status = H5Awrite(attribute_id,H5T_NATIVE_INT,attr_data);

/* close dataset et al. */
status = H5Aclose(attribute_id);
...
```

Example 8: output

```
$ h5dump RUN/my_fourth_file.h5
HDF5 "RUN/my_fourth_file.h5" {
GROUP "/" {
…
ATTRIBUTE "attr" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         DATA {
         (0): 100, 200
         }
      }
   }
}
}
```

Attributes: Do not abuse!

CFD code, each velocity dump has also:

```
1. ATTRIBUTE "GIT_BRANCH"
2. ATTRIBUTE "GIT_DIFF"
3. ATTRIBUTE "GIT_HASH"
4. ATTRIBUTE "GIT_NOT_COMMITTED"
5. ATTRIBUTE "GIT_NOT_STAGED"
6. ATTRIBUTE "LBE3D_CMAKE_FLAGS"
7. ATTRIBUTE "LBE3D_VERSION_MAJOR"
8. ATTRIBUTE "LBE3D_VERSION_MINOR"
9. ATTRIBUTE "MISC_EMAIL"
10. ATTRIBUTE "MISC_OWNER"
11. ATTRIBUTE "MISC_UUID"
12. ATTRIBUTE "ORIG_FILE_NAME"
13. ATTRIBUTE "lbe_diag_nsteps"
14. ATTRIBUTE "lbe_force_gravity_x"
15. ATTRIBUTE "lbe_rho_1"
```

…

Attributes: Do not abuse!

```
17. ATTRIBUTE "lbe_rho_2"
18. ATTRIBUTE "lbe_steps"
19. ATTRIBUTE "lbe_sx"
20. ATTRIBUTE "lbe_sy"
21. ATTRIBUTE "lbe_sz"
22. ATTRIBUTE "lbe_tau_1"
23. ATTRIBUTE "lbe_tau_2"
24. ATTRIBUTE "scmc_bc_mirror_z_m_rho_nohom_stripes_width"
25. ATTRIBUTE "scmc_bc_mirror_z_m_rho_nohom_wetting"
26. ATTRIBUTE "scmc_bc_mirror_z_m_rho_wetting"
27. ATTRIBUTE "scmc_coupling_g12"
28. ATTRIBUTE "scmc_init_droplet_n"
29. ATTRIBUTE "scmc_init_droplet_r0"
30. ATTRIBUTE "scmc_init_droplet_x0"
31. ATTRIBUTE "scmc_init_droplet_y0"
32. ATTRIBUTE "scmc_init_droplet_z0"
33. ATTRIBUTE "temperature_beta_1"
34. ATTRIBUTE "temperature_beta_2"
```

Example 9 (C)

Working with strings…

```
const char* days[] = {"Mon","Tue","Wed","Thu","Fri","Sat","Sun"};

/* create the fixed lenght string datatype */
str_type = H5Tcopy(H5T_C_S1);
status = H5Tset_size(str_type, 3);
…
for (i=0; i<7; ++i) {
index[0]=i;
  status = H5Sselect_elements(dataspace_id, H5S_SELECT_SET, 1, index);
  /* write dataset */
  status = H5Dwrite(dataset_id, str_type, strspace_id, dataspace_id,
   H5P_DEFAULT, days[i]);
}
status = H5Tclose (str_type);
```

Example 9: output

```
$ h5dump RUN/my_6th_file.h5
HDF5 "my_6th_file.h5" {
GROUP "/" {
   GROUP "Week" {
      GROUP "Days" {
      }
   }
   DATASET "dset" {
      DATATYPE  H5T_STRING {
         STRSIZE 3;
         STRPAD H5T_STR_NULLTERM;
         CSET H5T_CSET_ASCII;
         CTYPE H5T_C_S1;
      }
      DATASPACE  SIMPLE { ( 7 ) / ( 7 ) }
      DATA {
      (0): "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
      }
```

API 1.8 vs.1.6

- 1.6 API
  - ✓ `dset_id = H5Dcreate(file_id, "dataset1", H5T_NATIVE_INT, filespace, H5P_DEFAULT);`
- 1.8 API
  - ✓ `dset_id = H5Dcreate(file_id, "dataset1", H5T_NATIVE_INT, filespace, H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);`
- Read documentation please!!!!!

Using API 1.6 with 1.8 library

- 1.8 is backward compatible, provided at compile time you add: **`-DH5_USE_16_API`**
- Support to
  - ✓ External Links, Links in a group that link to objects in a different HDF5 file
  - ✓ User-defined Links
  - ✓ Dedicated Link Interface Link API (H5L) for directly managing links
  - ✓ Enhanced Attribute Handling Faster access and more compact storage
  - ✓ Object Copying: Copying an HDF5 object to a new location within a file or in a different file
  - ✓ Dedicated Object Interface
  - ✓ C++ and Fortran Wrapper Improvements
  - ✓ ……

Using API 1.8 with 1.6 library

```
serial_ex2.c(16): error #140: too many arguments in
   function call
  group_id = H5Gcreate( file_id,
  "/MyGroup",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
                                     ^

compilation aborted for serial_ex2.c (code 2)
```

- HFD5 library on compiled with API v16
- Best practice
  - Recompile the library with 1.8 API
  - You can use all 1.8 improvements
- Else
  1. Rewrite the code using 1.6 API interface
  2. Use **`-DH5Dopen_vers=2`** and similar preprocessing flag
  - In both case you cannot use the 1.8 API improvements

API 1.10

- HDF5 1.10.0 is the new minor revision of the HDF API
- New features:
  - Concurrent access to an HDF5 file: single-writer / multiple reader (SWMR). The SWMR feature enables user to read data while concurrently writing it.
  - Virtual Dataset (VDS). The VDS feature enables access to data stored in dataset in different HDF5 files using the standard H5D interface without rewriting or rearranging data.
  - Several under-the-hood optimizations.
  - Backward compatibility with the 1.8 data format
- Issues: the version 1.8 is not forward compatible with the 1.10 HDF5 files.