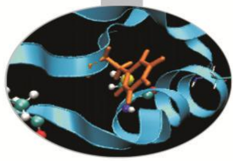
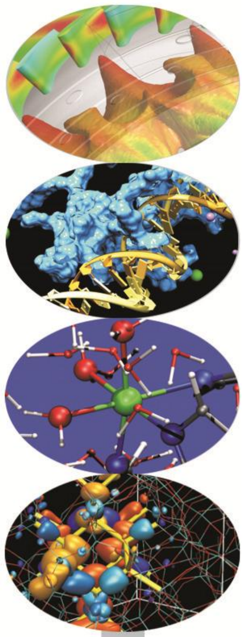


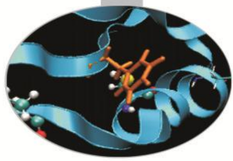
# Hierarchical Data Format 5:



Giusy Muscianisi – g.muscianisi@cineca.it

*SuperComputing Applications and Innovation Department*

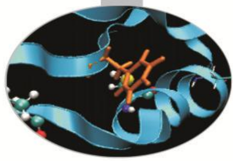
*May 17th, 2013*



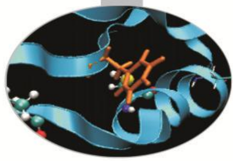
## Outline

- What is HDF5 ?
- Overview to HDF5 Data Model and File Structure
- Overview to HDF5 APIs

# Outline



- What is HDF5 ?
- Overview to HDF5 Data Model and File Structure
- Overview to HDF5 APIs



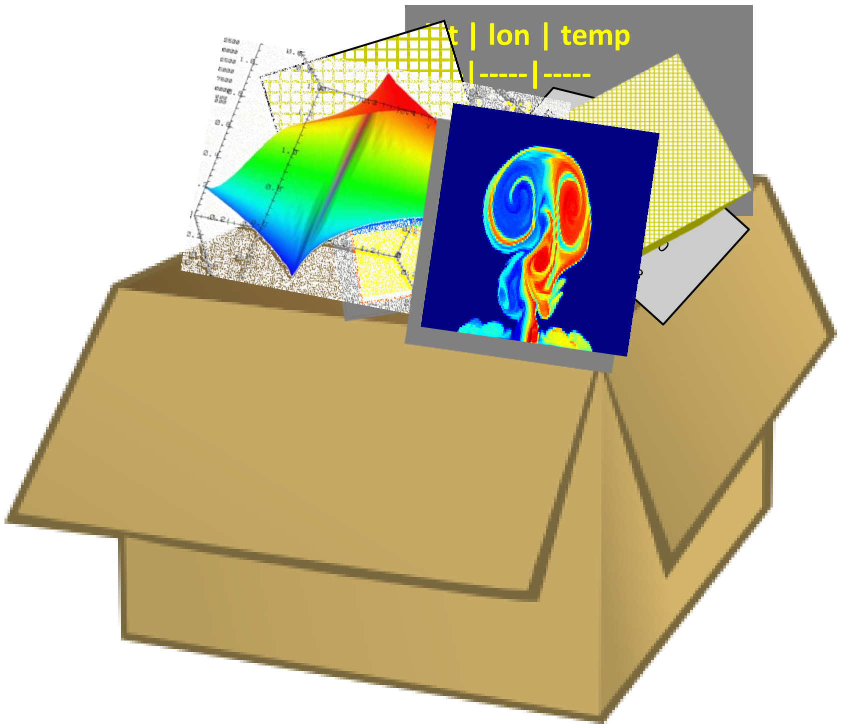
## HDF5

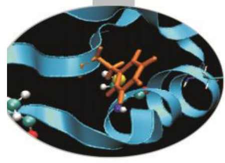
- HDF5 : Hierarchical Data Format  version 5
- File format for storing scientific data
  - To store and organize all kinds of data
  - To share data, to port files from one platform to another
  - To overcome a limit on number and size of the objects in the file
- Software for accessing scientific data
  - Flexible I/O library (parallel, remote, etc.)
  - Efficient storage
  - Available on almost all platforms
  - C, F90, C++ , Java APIs
  - Tools (HDFView, utilities)

# HDF5 File

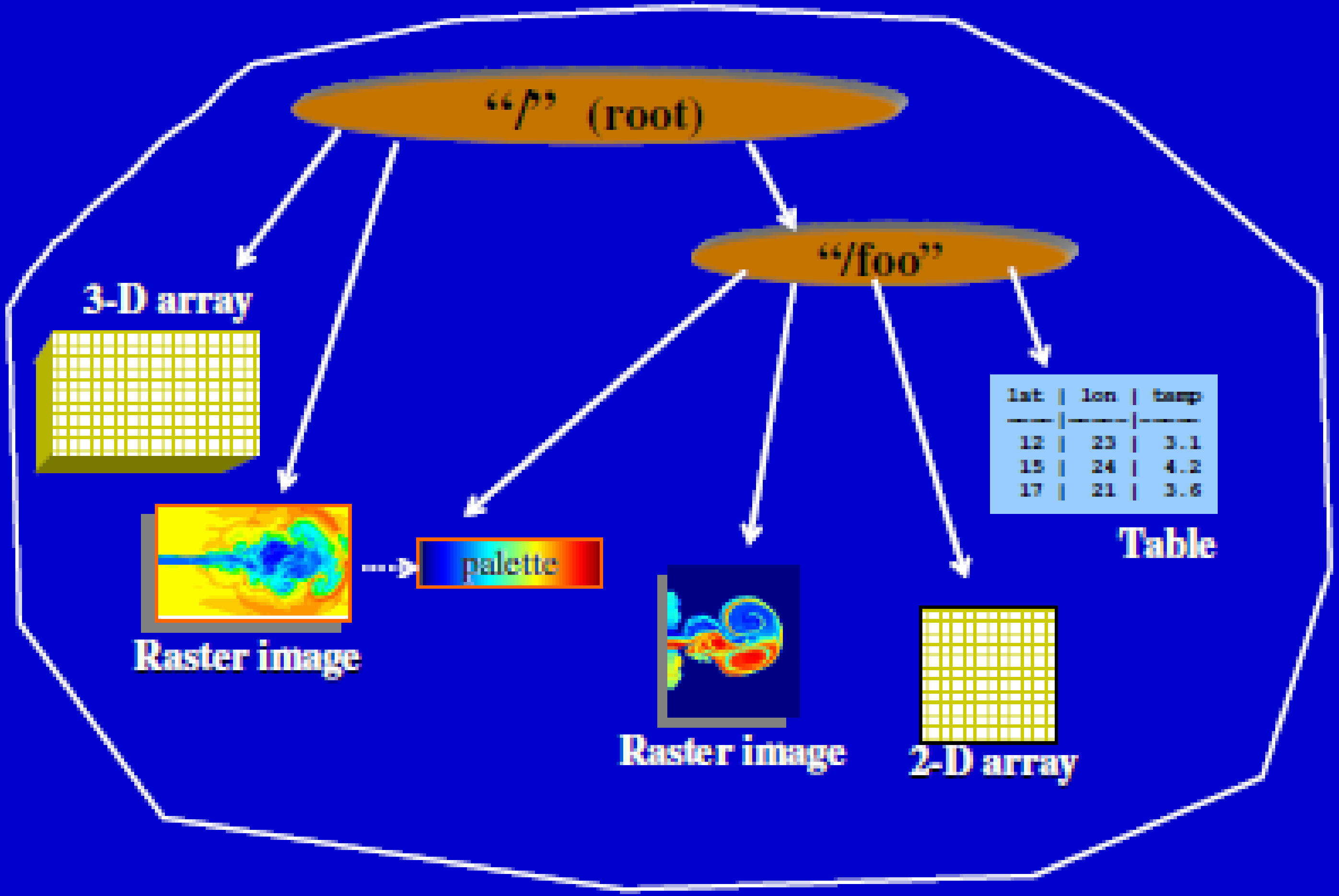
An HDF5 file is a binary file containing scientific data and supporting metadata.

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



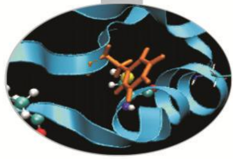


## Example HDF5 file



```
 lat | lon | temp
-----|-----|-----
  12 |  23 |  3.1
  15 |  24 |  4.2
  17 |  21 |  3.6
```

# Outline



- What is HDF5 ?
- Overview to HDF5 Data Model and File Structure
- Overview to HDF5 APIs

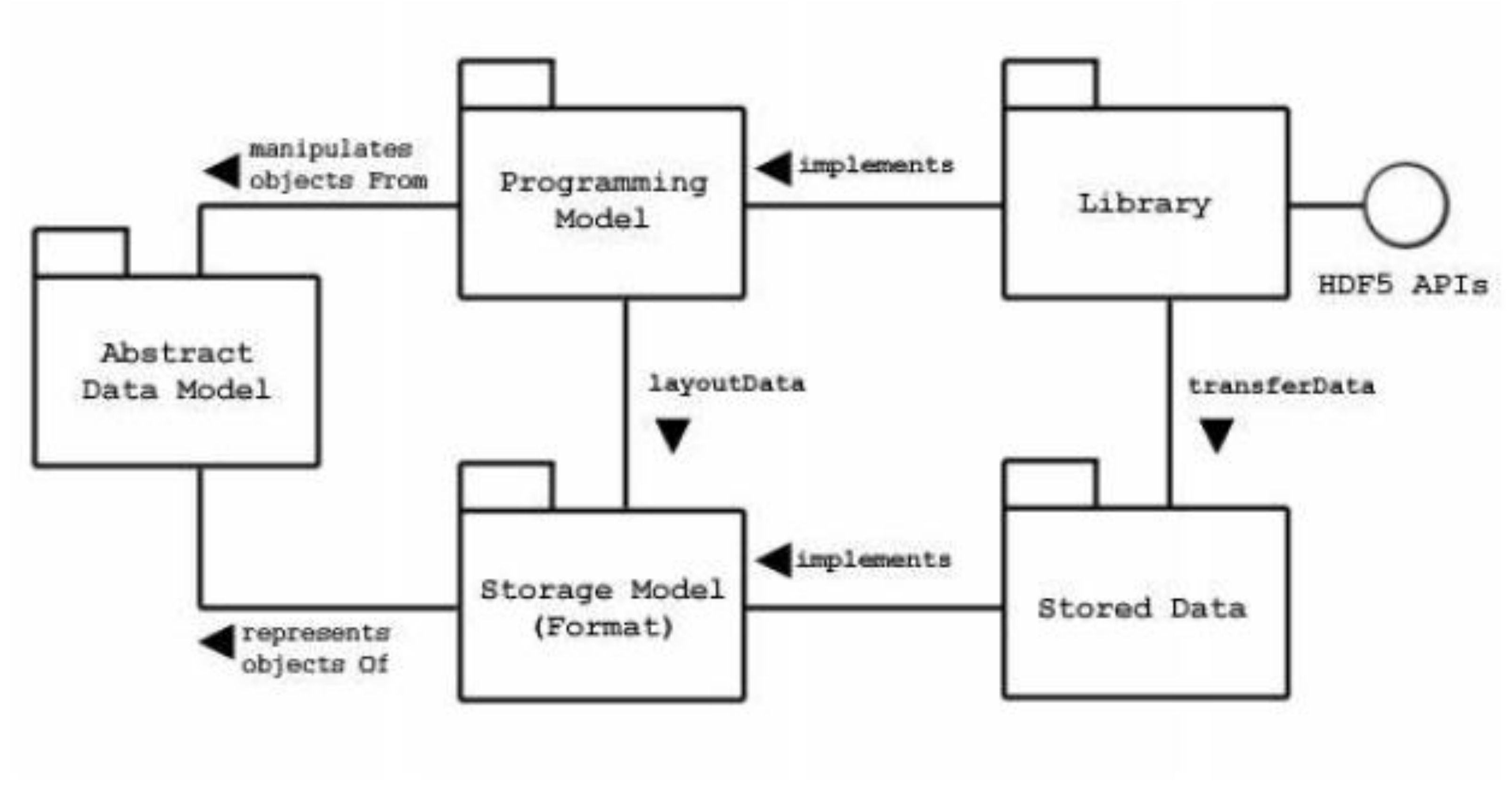
# HDF Data Model & Implementation

- HDF implements a model for managing and storing data.
- The model includes
  - an abstract data model and an abstract storage model (the data format)
  - libraries to implement the abstract model and to map the storage model to different storage mechanisms.
- The HDF5 library
  - provides a programming interface to a concrete implementation of the abstract models.
  - implements a model of data transfer, i.e., efficient movement of data from one stored representation to another stored representation.

SuperComputing Applications and Innovation

# HDF Data Model & Implementation

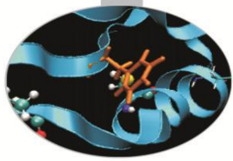
Relationships between the model and implementation



# HDF Data Model & Implementation

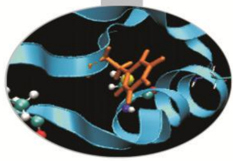
- The **Abstract Data Model** is a conceptual model of data, data types, and data organization. The abstract data model is independent of storage medium or programming environment.
- The **Storage Model** is a standard representation for the objects of the abstract data model. The HDF5 File Format Specification defines the storage model.
- The **Programming Model** is a model of the computing environment and includes platforms from small single systems to large multiprocessors and clusters. The programming model manipulates (instantiates, populates, and retrieves) objects from the abstract data model.
- The **Library** is the concrete implementation of the programming model. The Library exports the HDF5 APIs as its interface.
  - In addition to implementing the objects of the abstract data model, the Library manages data transfers from one stored form to another.
  - Data transfer examples include reading from disk to memory and writing from memory to disk.

# Abstract Data Model

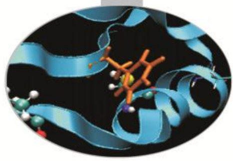


- The abstract data model (ADM) defines concepts for defining and describing complex data stored in files.
- The ADM is a very general model which is designed to conceptually cover many specific models.
- Many different kinds of data can be mapped to objects of the ADM, and therefore stored and retrieved 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.

# Abstract Data Model



- **File** - a contiguous string of bytes in a computer store (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

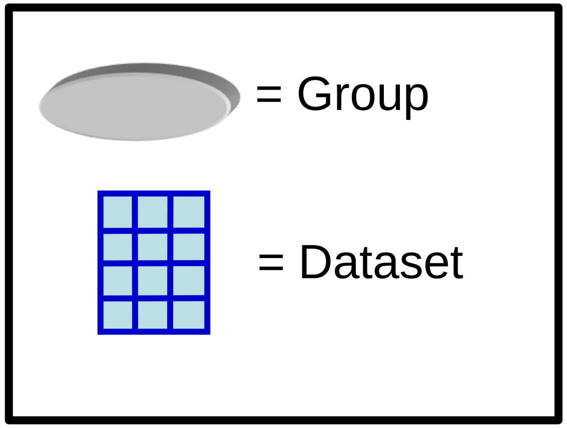
- An HDF5 file is a container for storing a variety of scientific data
- Is composed of two primary types of objects
  - **Groups**: a grouping structure containing zero or more HDF5 objects, together with supporting metadata
  - **Datasets**:  a multidimensional array of data elements, together with supporting metadata

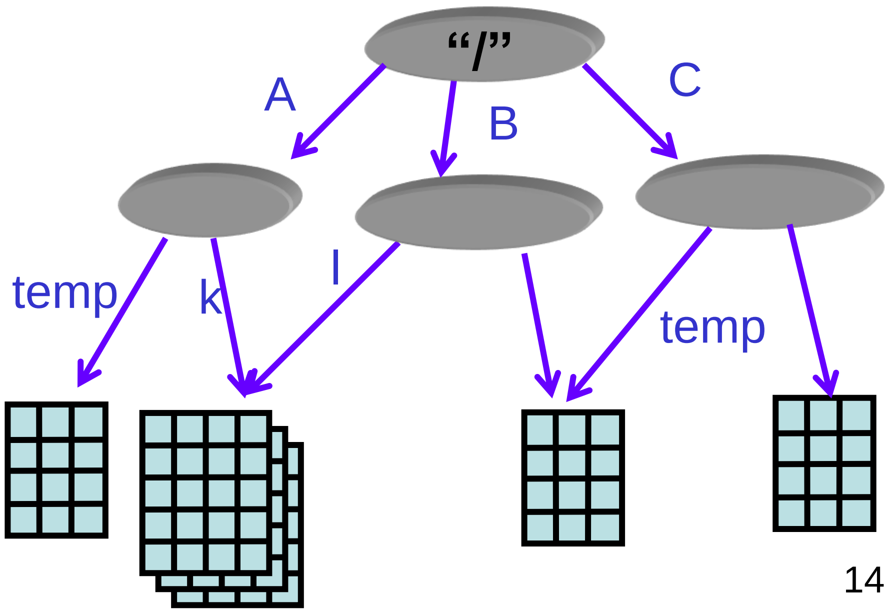
- Any HDF5 group or dataset may have an associated attribute list
  - **Attribute**: a user-defined HDF5 structure that provides extra information about an HDF5 object.

# HDF5 Groups

## A 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

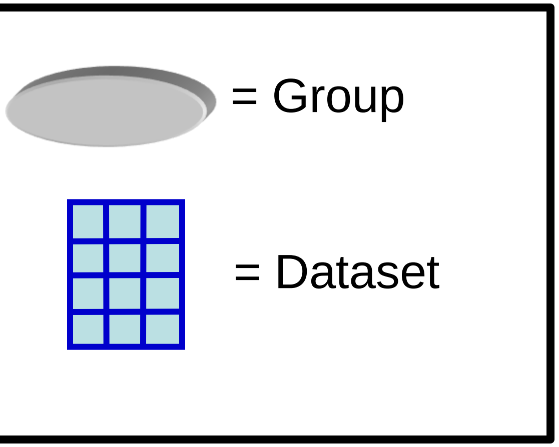


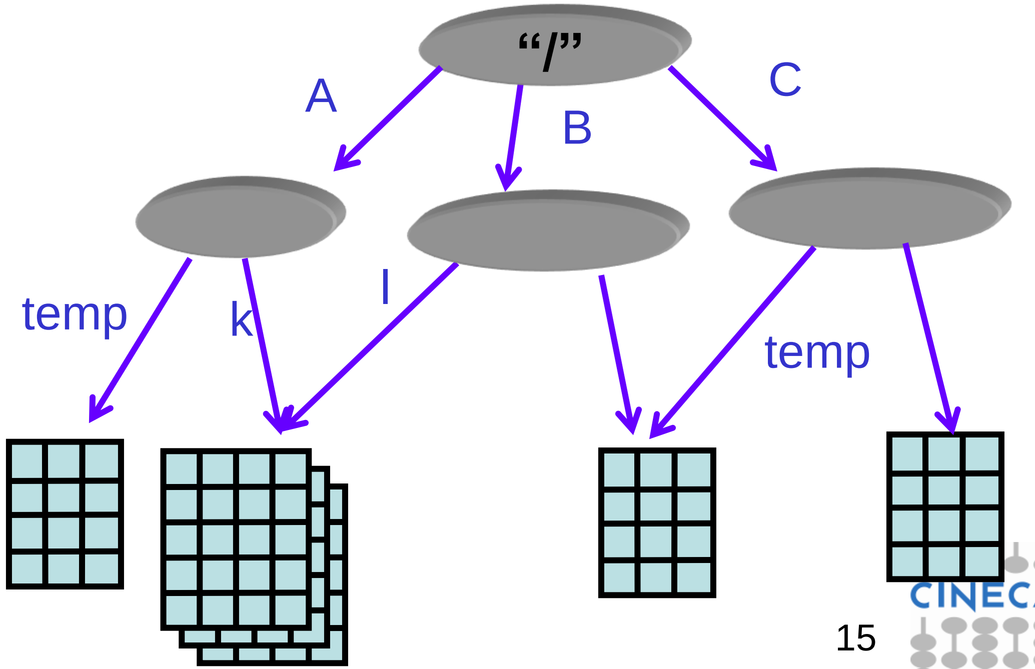


# HDF5 Groups

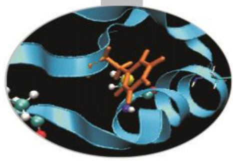
HDF5 objects are identified and located by their pathnames:

```
/         (signifies the root group)
/A        (signifies a member of the root group called A)
/A/temp  (signifies a member of the group A, which in turn is a
   member of the root group)

/A/k and  /B/l are the same
```





# HDF5 Dataset

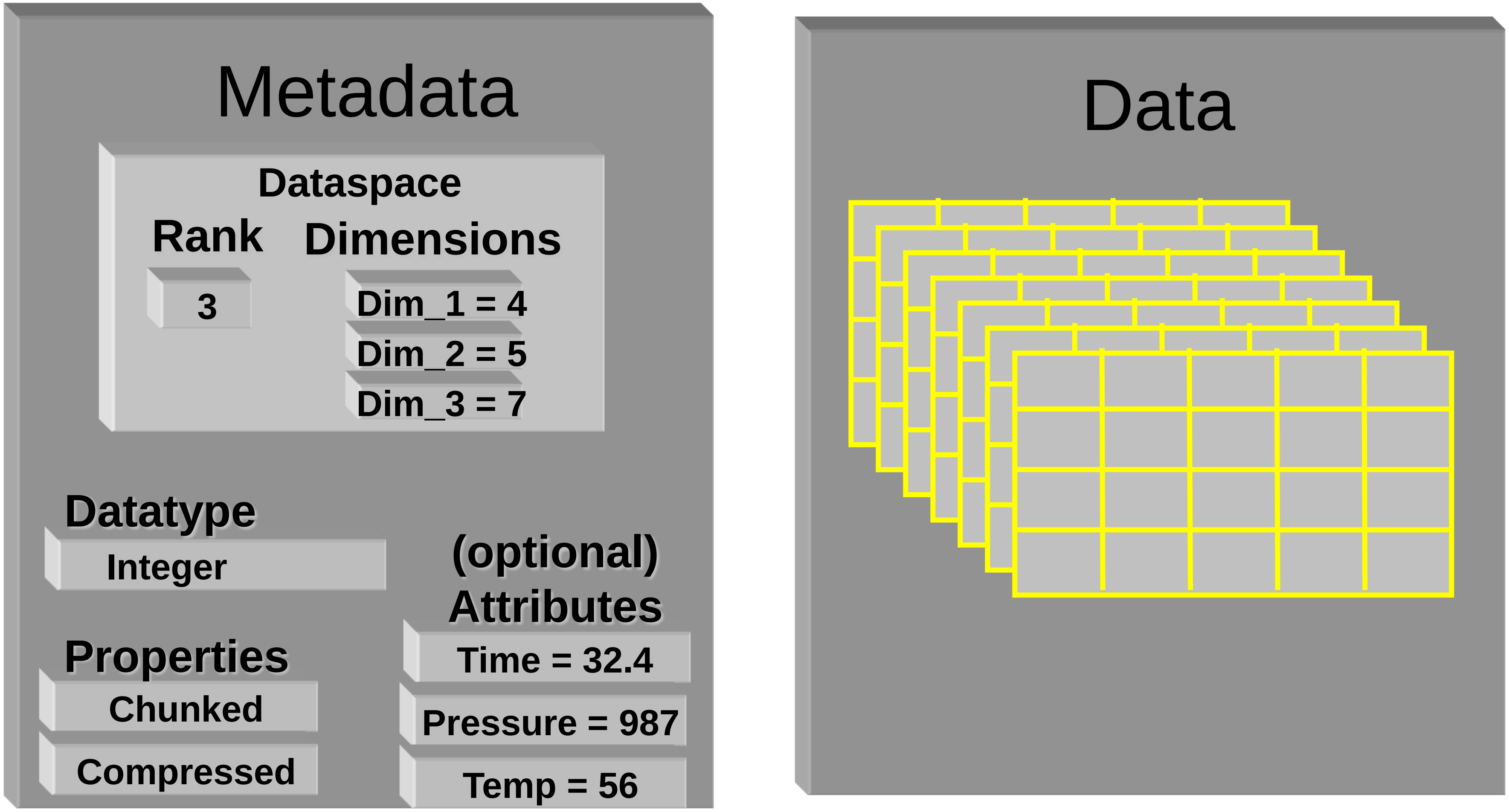


**Object used to organize and contain your "raw data values".**

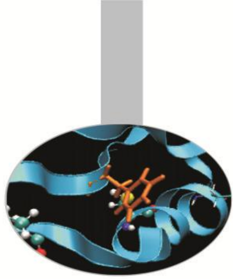
They consist of:

- Your raw data
- Metadata describing the raw data:
  - **Dataspace:** information to describe the logical layout of the data elements
  - **Datatype:** information to interpret the data
  - **Properties:** characteristics of the data
  - **Attributes :** additional optional information that describes the data

# HDF5 Dataset



# HDF5 Dataspaces

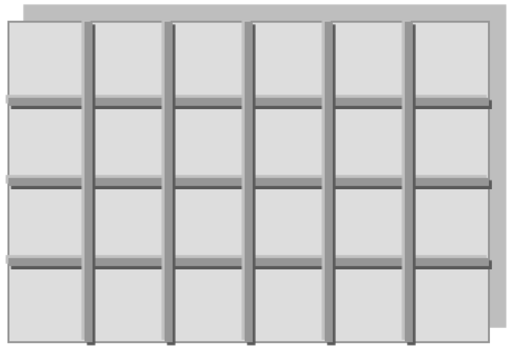


**An HDF5 Dataspace describes the logical layout for the data elements:**

- Array
  - multiple elements in dataset organized in a multi-dimensional (rectangular) array
  - maximum number of elements in each dimension may be fixed or unlimited
- NULL
  - no elements in dataset
- Scalar
  - single element in dataset

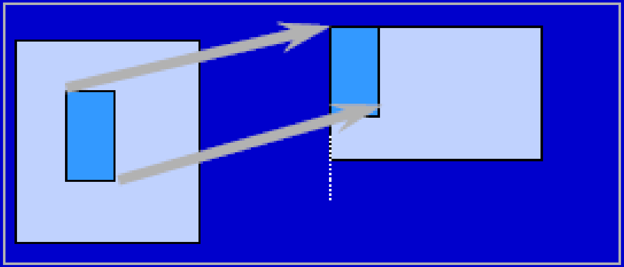
# HDF5 Dataspaces

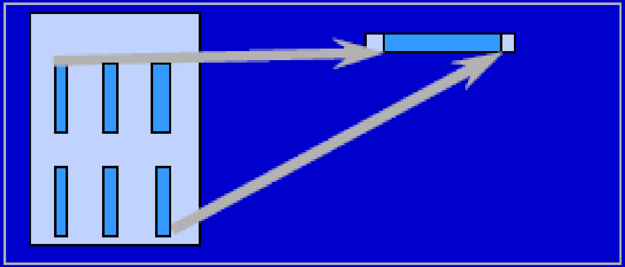
- Dataspace – spatial info about a dataset
  - Rank and dimensions
    - Permanent part of dataset definition
  - Subset of points, for partial I/O
    - Needed only during I/O operations
- Apply to datasets in memory or in the file

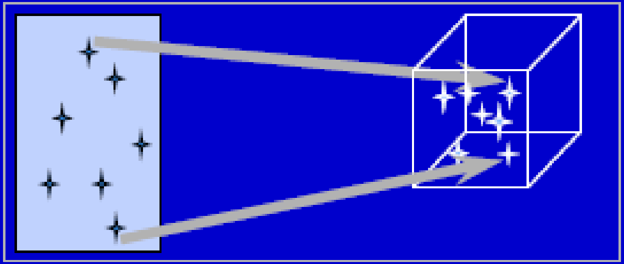


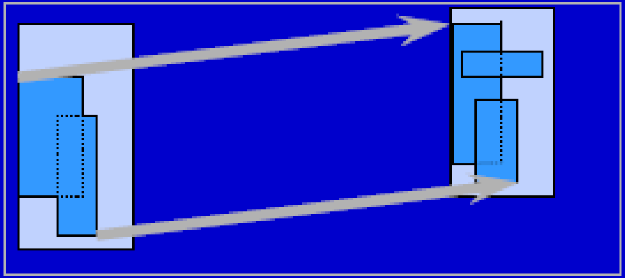
Rank =2
Dimensions = 4 x 6

## Sample Mappings between File Dataspaces and Memory Dataspaces

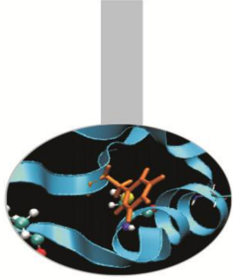

(a) Hyperslab from a 2D array to the corner of a smaller 2D array


(b) Regular series of blocks from a 2D array to a contiguous sequence at a certain offset in a 1D array


(c) A sequence of points from a 2D array to a sequence of points in a 3D array.


(d) Union of hyperslabs in file to union of hyperslabs in memory.

# HDF5 Datatypes

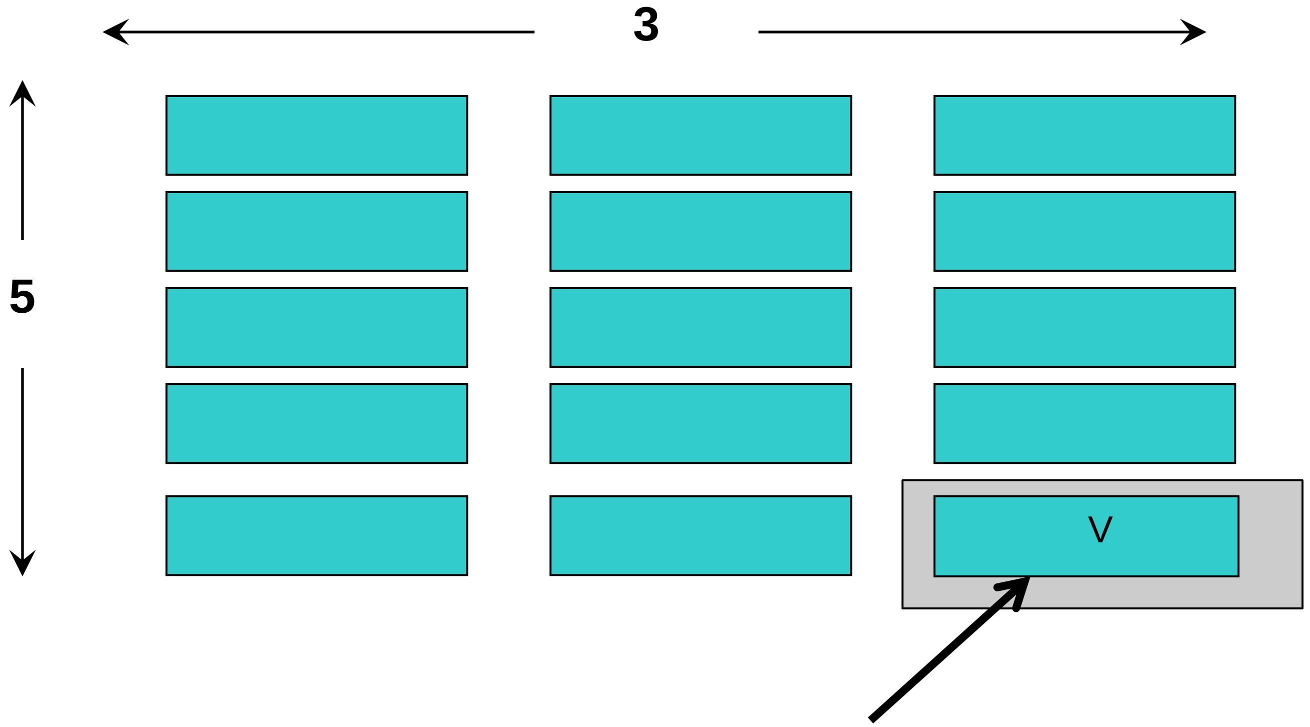


**The HDF5 datatype describes how to interpret individual data elements.**

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 - names mapped to integers
- opaque
- compound (similar to C structs)

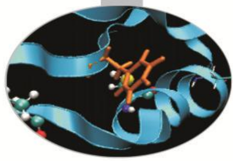
# HDF5 Dataset



**Datatype:** **16-byte integer**

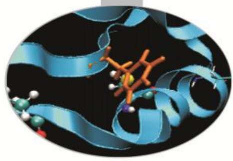
**Dataspace:** **Rank = 2**
**Dimensions = 5 x 3**

## HDF5 Properties



- Properties (also known as Property Lists) are characteristics of HDF5 objects that can be modified
- Default properties handle most needs
- By changing properties one can take advantage of the more powerful features in HDF5

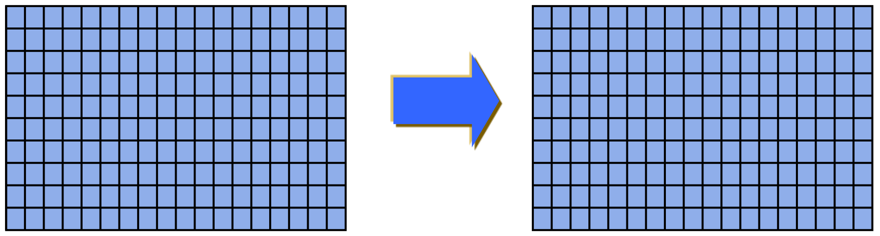
# HDF5 Properties



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

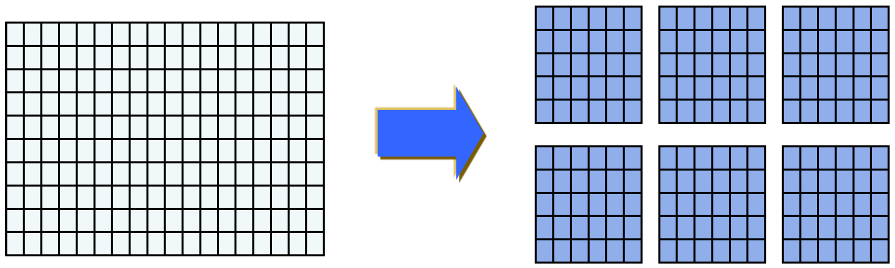
# Storage Properties

Contiguous (default)



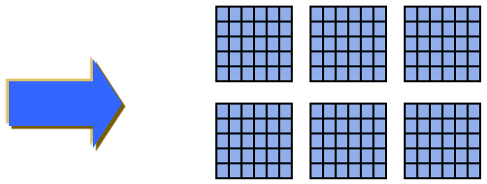
Data elements stored physically adjacent to each other

Chunked



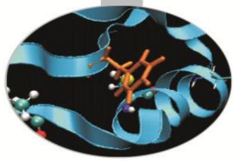
Better access time for subsets; extensible

Chunked & Compressed



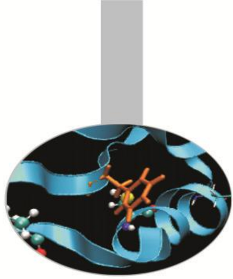
Improves storage efficiency, transmission speed

# HDF5 Attributes



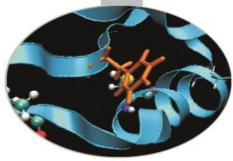
- An HDF5 attribute has a name and a value
- Attributes typically contain user metadata
- Attributes may be associated with
  - HDF5 groups
  - HDF5 datasets
  - HDF5 named datatypes
- An attribute's value is described by a datatype and a dataspace
- Attributes are analogous to datasets except…
  - they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

## Outline



- What is HDF5 ?
- Overview to HDF5 Data Model and File Structure
- Overview to HDF5 APIs

# The General HDF5 API

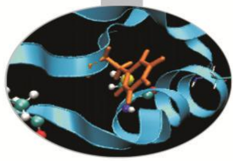


- The HDF5 library provides several interfaces, or APIs.
  - These APIs provide routines for creating, accessing, and manipulating HDF5 files and objects.
- The library itself is implemented in C.
  - To facilitate the work of FORTRAN 90, C++ and Java programmers, HDF5 function wrappers have been developed 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 |

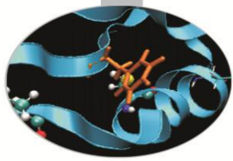
# Order of Operations



- The library imposes an order on the operations by argument dependencies
  - Example: A file must be opened before a dataset because the dataset open call requires a file handle as an argument

- Objects can be closed in any order, and reusing a closed object will result in an error

# HDF5 C Programming Issue



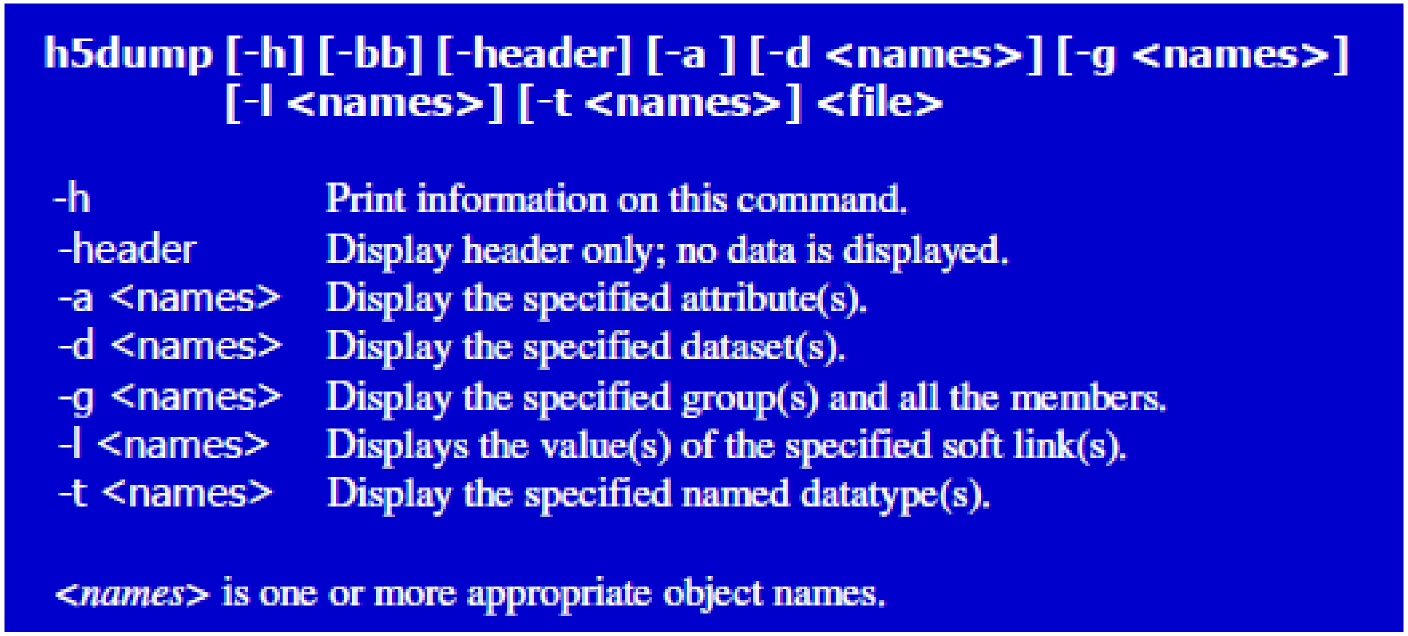
For portability, HDF5 library has its own defined types:

| | |
|---|---|
| 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 lenght datatype |

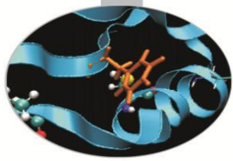
For C, include #include hdf5.h at the top of your HDF5 application

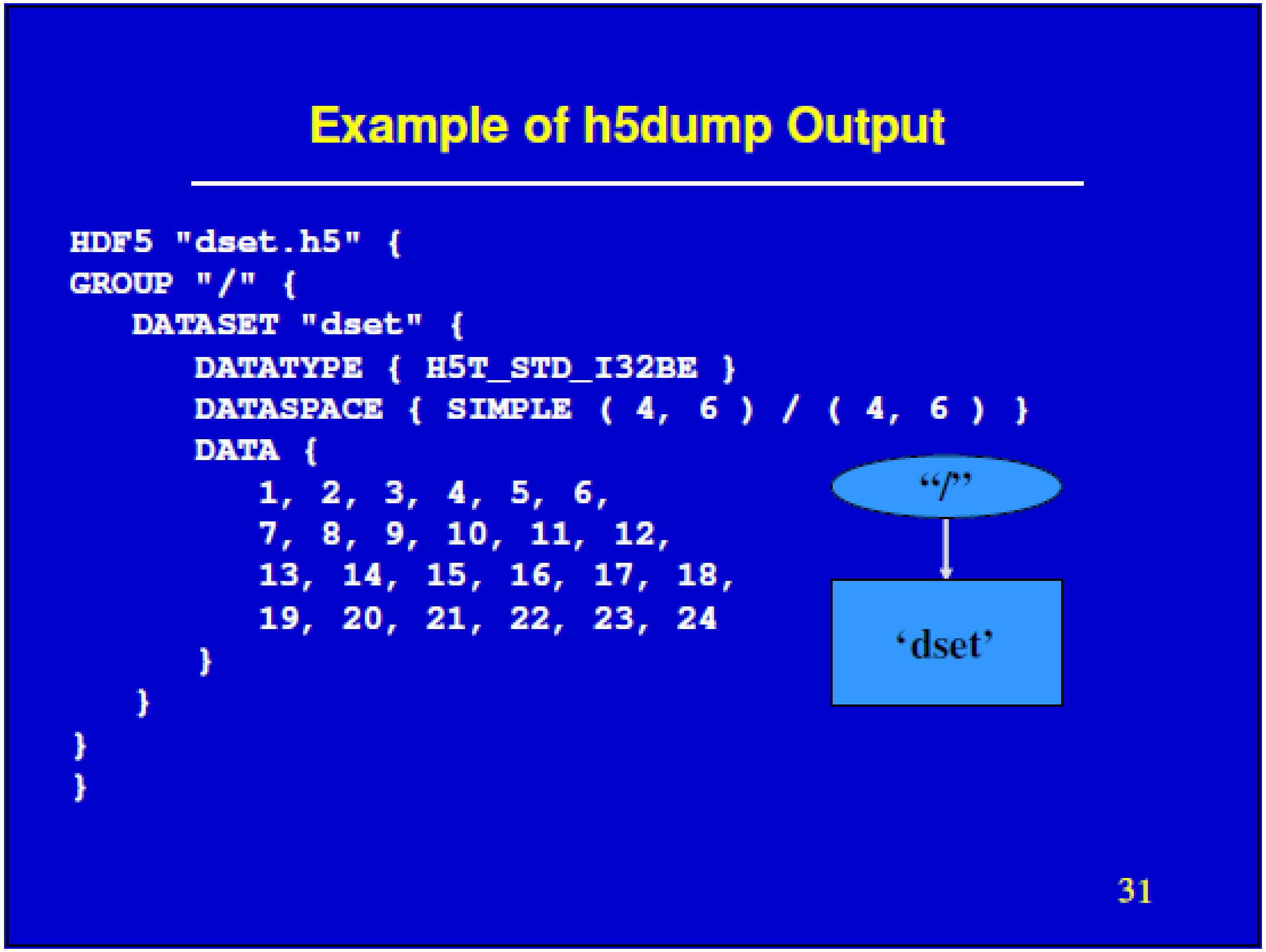
For Fortran, USE HDF5

# h5dump
# command-line Utility for Viewing HDF5 Files



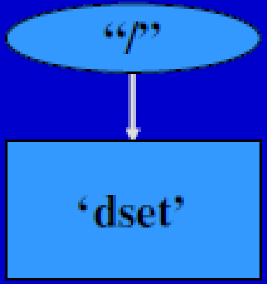
```
h5dump [-h] [-bb] [-header] [-a ] [-d <names>] [-g <names>]
          [-l <names>] [-t <names>] <file>

 -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.
```





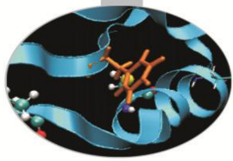
## Example of h5dump Output

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



31

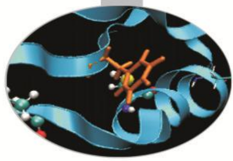
## Create an HDF5 File



- To create an HDF5 file, an application 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:
  - Specify File Creation and Access property lists, if necessary
  - Create a file
  - Close the file and property lists, if necessary

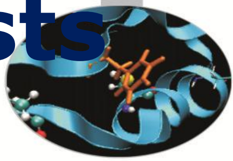
# File access mode



- When creating a file, the file access mode specifies the action to take if the file already exists:
  - H5F_ACC_TRUNC specifies that 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 specifies that the 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.

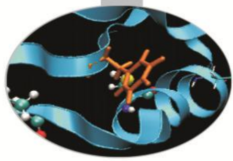
# File access/creation Property Lists



- A property list is a collection of values that can be passed to HDF5 functions at lower layers of the library

- **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

- **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

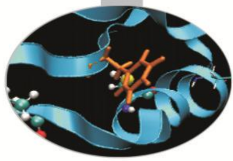


## Binding of H5Fcreate

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

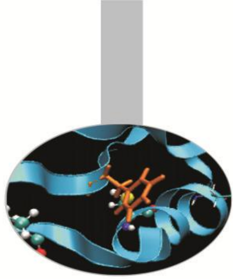
```
IN name : Name of the file to access
IN flags : File access flags
IN create_id : File creation property list
               identifier
IN access_id : File access property list
               identifier
```

# Binding of H5Fclose



```
herr_t H5Fclose(hid_t file_id)

    IN file_id : Identifier of the file to terminate
                        access to
```

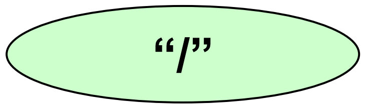
# Example 1



```
#include <hdf5.h>
#define FILE "file.h5‟
main() {
   hid_t file_id; /* file identifier */
   herr_t status;

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

   /* Terminate access to the file. */
   status = H5Fclose (file_id);

}
```

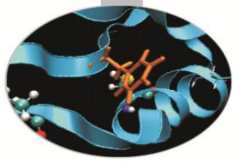
# Example 1: h5dump Output

```
HDF5 "file.h5" {
GROUP "/" {
}
}
```



When a HDF5 is created, the “/” root group is created by default.

# Use Groups

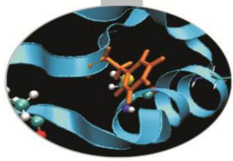


- HDF5 groups provide a mechanism for organizing meaningful and extendable sets of datasets within an HDF5 file.

- An HDF5 group is a structure containing zero or more HDF5 objects.

- To create a group, the calling program must:
  - Obtain the location identifier where the group is to be created
  - Create the group
  - 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

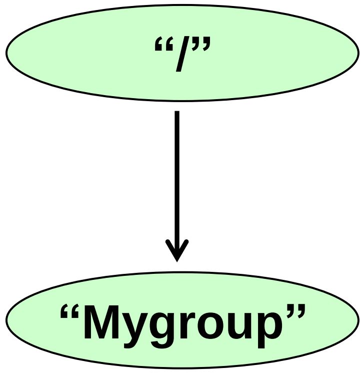
**gapl_id** : Group access property list identifier (No group access properties have been implemented at this time; use H5P_DEFAULT.)

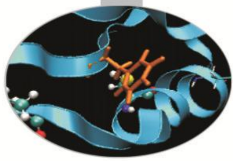
# Example 2

```
#include "hdf5.h"
#define FILE "group.h5"

int main() {
hid_t        file_id, group_id;  /* identifiers */
herr_t       status;

file_id = H5Fcreate(FILE, 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);

status = H5Fclose(file_id);
}
```

## Example 2: h5dump Output

```
HDF5 "group.h5" {
GROUP "/" {
  GROUP "Mygroup" {
  }
}
}
```





## Open an existing Group

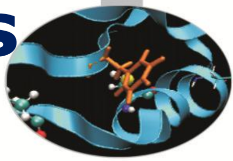
```
hid_t  H5Gopen( hid_t loc_id, const char
   * name, hid_t gapl_id )
```

**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.)

## Groups: Absolute & Relative Names



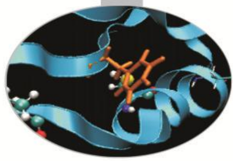
- To create an HDF5 object, we have to specify the location where the object is to be created. This location is determined by the identifier of an HDF5 object and the name of the object to be created.

- 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

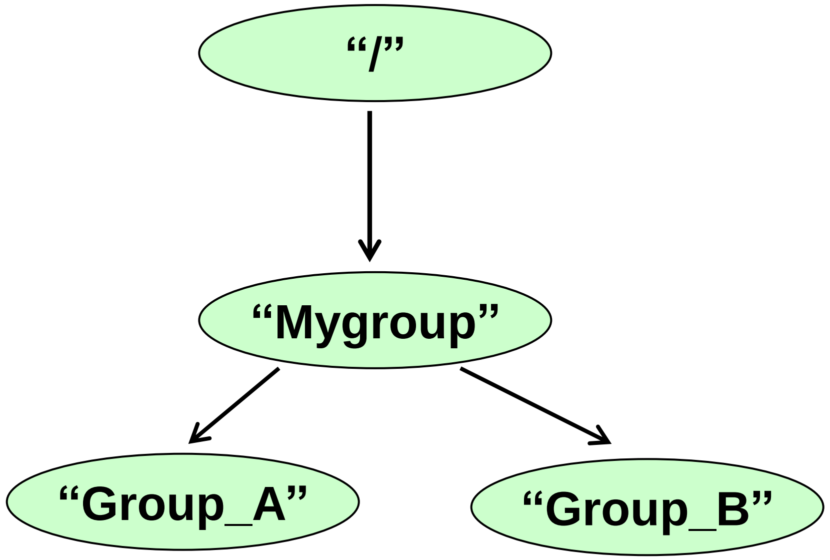
```
. . .
/* Create group "MyGroup" in the root group using absolute name. */
group1_id = H5Gcreate(file_id, "MyGroup", H5P_DEFAULT, H5P_DEFAULT,
   H5P_DEFAULT);

/* Create group "Group_A" in group "MyGroup" using absolute name. */
group2_id = H5Gcreate(file_id, "/MyGroup/Group_A", H5P_DEFAULT,
   H5P_DEFAULT, H5P_DEFAULT);

/* Create group "Group_B" in group "MyGroup" using relative name. */
group3_id = H5Gcreate(group1_id, "Group_B", H5P_DEFAULT, H5P_DEFAULT,
   H5P_DEFAULT);

/* Close groups. */
status = H5Gclose(group1_id);
status = H5Gclose(group2_id);
status = H5Gclose(group3_id);
. . .
```



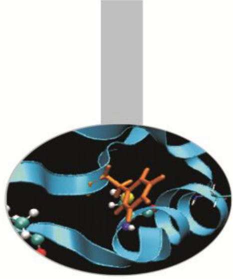
SuperComputing Applications and Innovation

# Example 3: h5dump Output

```
HDF5 "groups.h5" {
GROUP "/" {
   GROUP "MyGroup" {
       GROUP "Group_A" {
       }
       GROUP "Group_B" {
       }
   }
}
}
```



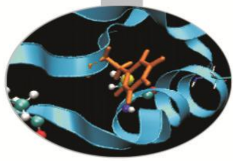
# Step to use Datasets 1/2



- A dataset is a multidimensional array of data elements, together with supporting metadata.

- To create an empty dataset (no data written) 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:
     1. Define a datatype or specify a pre-defined datatype.
     2. Define a dataspace (shape of the array of the dataset).
     3. 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.

# Step to use Datasets 1/2



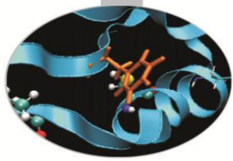
Regarding to the definition of the dataset characteristics:

1. Define a datatype or specify a pre-defined datatype.
2. Define a dataspace.
3. Specify the property list(s) or use the default

Note that:

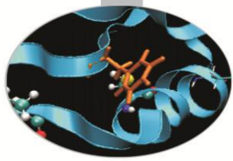
- In HDF5, datatypes and dataspaces are independent objects which are created separately from any dataset that they might be attached to.
- Because of this, the creation of a dataset requires the definition of the **datatype** and **dataspace**.

## Datatypes



- A **datatype** is a collection of properties, all of which can be stored on disk, and which, when taken as a whole, provide 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



Examples:

**H5T_IEEE_F64LE** Eight-byte, little-endian, IEEE floating-point

**H5T_IEEE_F32BE** Four-byte, big-endian, IEEE floating point

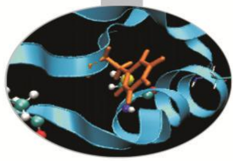
**H5T_STD_I32LE** Four-byte, little-endian, signed two's complement integer

**H5T_STD_U16BE** Two-byte, big-endian, unsigned 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

# Standard Predefined Datatype



Examples:

**`H5T_IEEE_F64LE`** Eight-byte, little-endian, IEEE floating-point

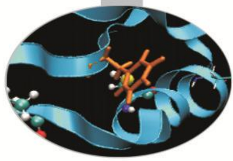
**`H5T_IEEE_F32BE`** Four-byte, big-endian, IEEE floating point

**`H5T_STD_I32LE`** Four-byte, little-endian, signed two's complement integer

**`H5T_STD_U16BE`** Two-byte, big-endian, unsigned integer

**Architecture** **Programming Type**

# Native Predefined Datatype



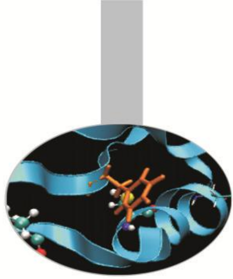
Examples of predefined native types in C:

| | |
|---|---|
| **`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, which means that 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 IN: Number of dimensions of dataspace

dims IN: An array of the size of each dimension

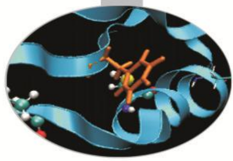
maxdims IN: An array of the maximum size of each dimension.

A value of H5S_UNLIMITED specifies the unlimited dimension.

A value of NULL specifies that *dims and maxdims* are the same.

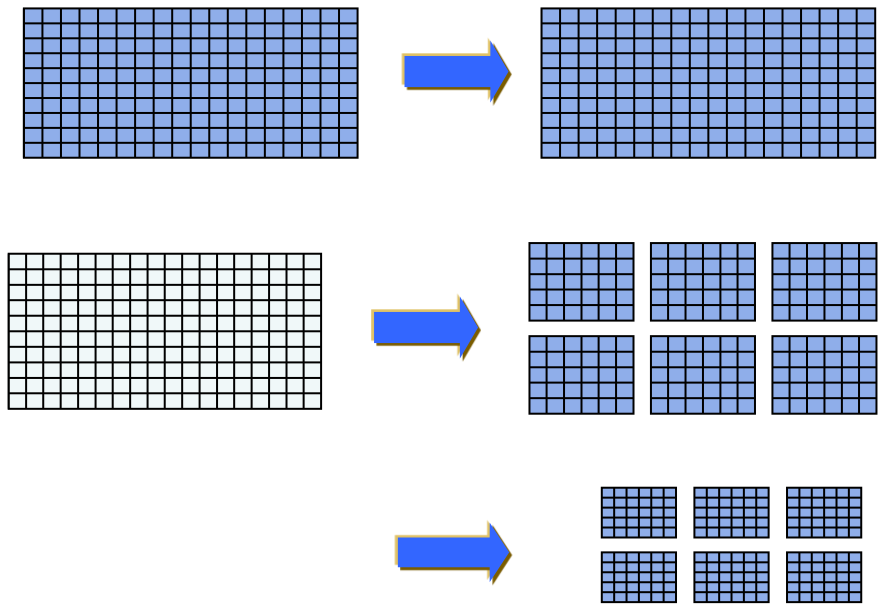
**Simple Datespace:**
a regular N-dimensional array of data points

## Property Lists



- Property lists are a mechanism for modifying the default behavior when creating or 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



Contiguous (default)

Data elements stored physically adjacent to each other

Chunked

Better access time for subsets; extensible

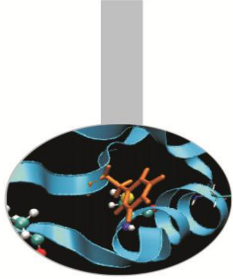
Chunked & Compressed

Improves storage efficiency, transmission speed

**H5P_DEFAULT: contiguous**

***Dataset creation property list:***

**information on how to organize data in storage.**



## Property List example

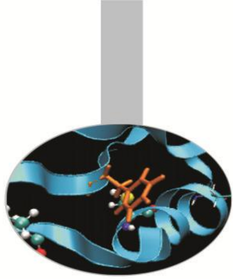
Create the dataset creation property list, add the gzip compression filter (deflate) and set the chunk size:

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

NOTE:

The property “create_plist_id” will be passed when the dataset will be created

# 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** IN: Location identifier

**name** IN: Dataset name

**dtype_id** IN: Datatype identifier

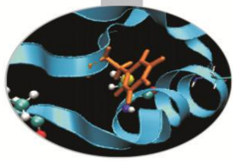
**space_id** IN: Dataspace identifier

**lcpl_id** IN : Link creation property list

**dcpl_id** IN : Dataset creation property list

**dapl_id** IN: Dataset access property list

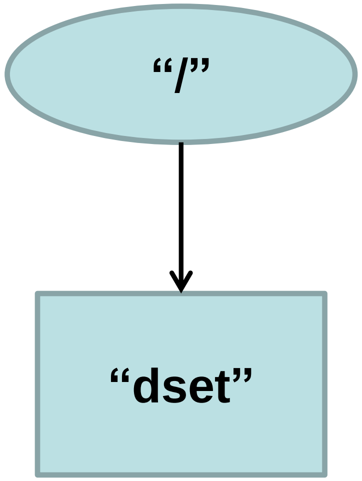
# Create an empty, chunked, 4x6 Dataset: Example 4



```
hid_t file_id, dataset_id, dataspace_id;
hid_t dcpl   /* dataset creation property */
hsize_t dims[2]={4,6};
herr_t status;

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

dataspace_id = H5Screate_simple (2, dims, NULL);

dcpl = H5Pcreate (H5P_DATASET_CREATE);
status = H5Pset_chunk (dcpl, 2, chunk);

dataset_id = H5Dcreate(file_id,"dset",H5T_STD_I32BE, dataspace_id,
   H5P_DEFAULT, dcpl, H5P_DEFAULT);

status = H5Dclose (dataset_id); status = H5Sclose (dataspace_id);
status = H5Pclose (dcpl); status = H5Fclose (file_id);
```

# Example 4: h5dump Output

```
HDF5 "dset.h5" {
GROUP "/" {
   DATASET "dset" {
        DATATYPE { H5T_STD_I32BE }
        DATASPACE { SIMPLE ( 4, 6 ) / ( 4, 6 ) }
        DATA {
               0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0
        }
   }
}
}
```

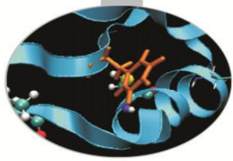


# Dataset IO operations

- During a dataset I/O operation, the library transfers raw data between memory and the file.
- The data in memory can have a datatype different from that of the file and can also be of a different size (i.e., the data in memory is a subset of the dataset elements, or vice versa).

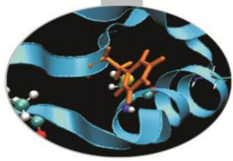
- Therefore, to perform read or write operations, the application program must specify:
  - The dataset
  - The dataset's datatype in memory
  - The dataset's dataspace in memory
  - The dataset's dataspace in the file
  - The dataset transfer property list.
  - The data buffer

- The data transfer property list is used to control various aspects of the I/O, such as caching hints or collective I/O information.



# Dataset IO operations

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

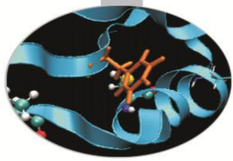


## Dataset IO operations

- 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 (file-memory, memoryfile)
  - 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 IO

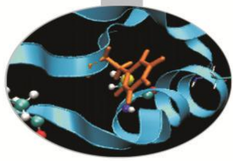


- Selected elements (called selections) from source are mapped (read/written) to the selected elements in destination

- 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

# Binding Open Dataset

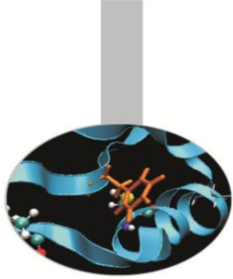


```
hid_t H5Dopen (hid_t loc_id, const char *name)

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

**NOTE**:

File datatype and dataspace are known when a dataset is opened

# Binding Write Dataset

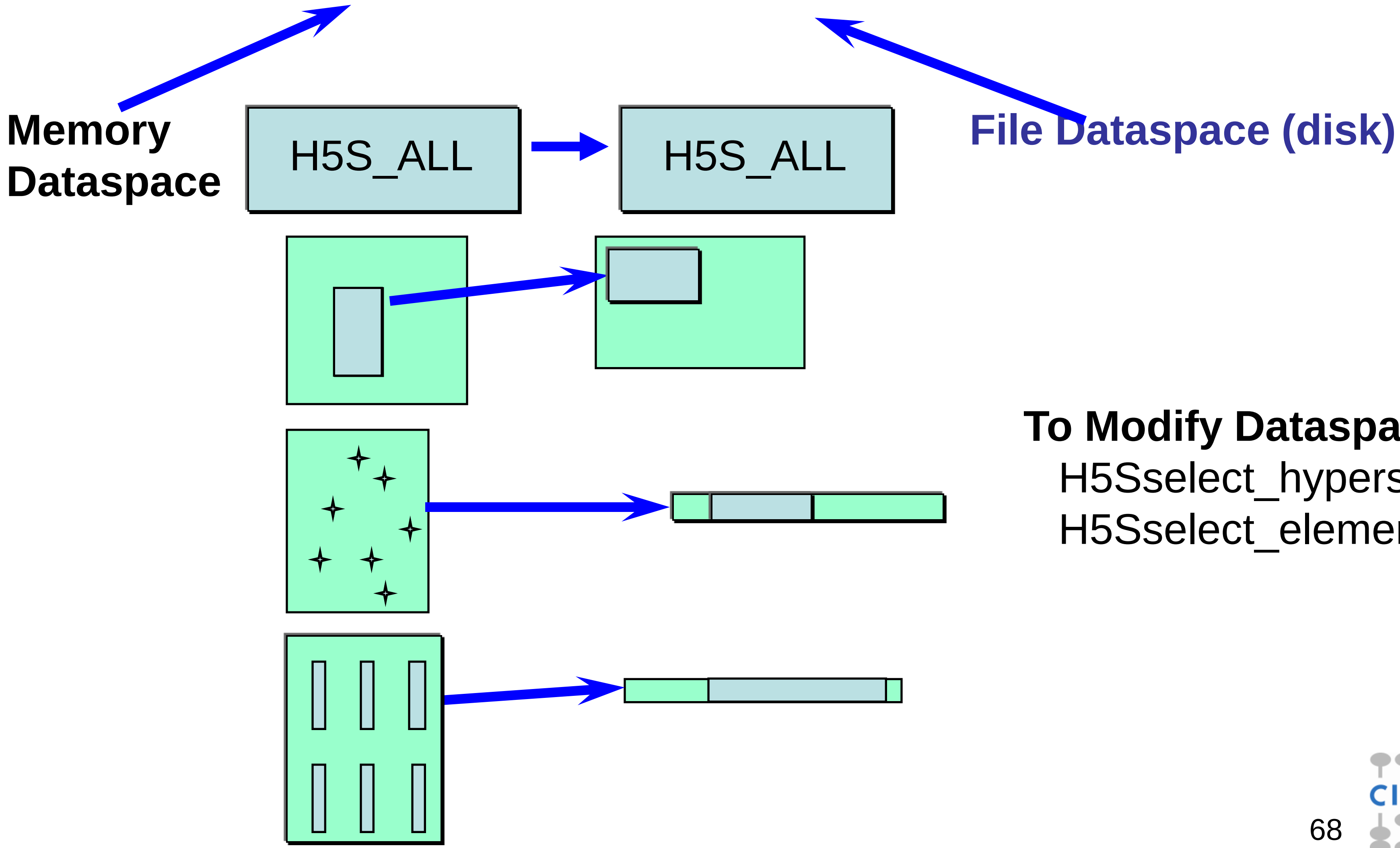


```
herr_t H5Dwrite (hid_t dataset_id, hid_t
   mem_type_id, hid_t mem_space_id, hid_t
   file_space_id, hid_t xfer_plist_id, const void *
   buf )
```

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

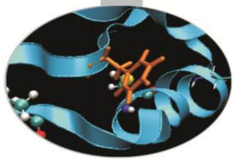
# Partial I/O

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



**To Modify Dataspace:**
H5Sselect_hyperslab
H5Sselect_elements

# Binding 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** IN: Identifier of the dataset to read to

**mem_type_id** IN: Identifier of memory datatype of the dataset

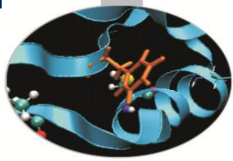
**mem_space_id** IN: Identifier of the memory dataspace (or H5S_ALL)

**file_space_id** IN: Identifier of the file dataspace (or H5S_ALL)

**xfer_plist_id** IN: Identifier of the data transfer properties to use (or H5P_DEFAULT)

**buf** IN: Buffer with data to be written to the file

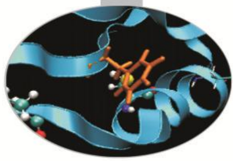
# Writing to an existing Dataset: Example 5



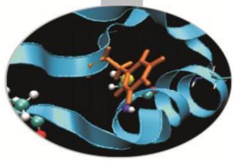
```
hid_t file_id, dataset_id;
herr_t status;
int i, j, dset_data[4][6];

/* Initialize buffer */
for (i = 0; i < 4; i++)
   for (j = 0; j < 6; j++)
        dset_data[i][j] = i * 6 + j + 1;

/* Open existing file and dataset */
file_id = H5Fopen ("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);
dataset_id = H5Dopen (file_id, "dset");

/* Write to dataset
status = H5Dwrite (dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
   H5P_DEFAULT, dset_data);
```

# Example 5: h5dump Output

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



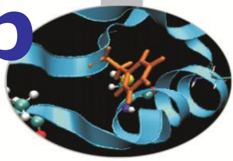
# R/W to a Subset of a Dataset



- 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.

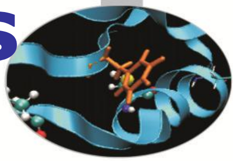
# Binding of H5Sselect_hyperslab



```
herr_t H5Sselect_hyperslab(hid_t space_id, H5S_selop
   er_t op, const hsize_t *start, const hsize_t
   *stride, const hsize_t *count, const hsize_t
   *block )
```

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

# Binding of H5Sselect_elements



```
herr_t H5Sselect_elements( hid_t space_id, H5S_selop
   er_t op, size_t num_elements, const hsize_t
   *coord )
```

**space_id** IN: Identifier of the dataspace.

**op** IN: Operator specifying how the new selection is to be combined with the existing selection for the dataspace.

**num_elements** IN: Number of elements to be selected.

**coord** IN: 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.

# Example: R/W to a Subset of a Dataset

1. In an HDF5 file, creates an 8 x 10 integer dataset, with a simple dataspace.
2. Initialize and write data in such dataset; print the data written and then close all.
3. Re-open the file and the dataset.
4. Re-write a portion of such dataset, with dimension 3x4.
5. Print the new dataset and then close all

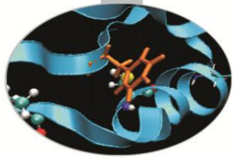
| Contents of Original Dataset Created | Contents of Dataset After 3 x 4 Subset Written |
|---|---|
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 5 5 5 5 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 5 5 5 5 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 5 5 5 5 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |

# Example R/W to a Subset of a Dataset

Hints for point 4. :

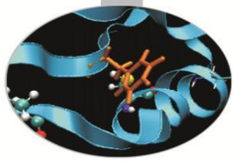
- Specify size and shape of subset to write.
- Create memory space with size of subset. Get file dataspace and select subset from file dataspace.
- Write  the subset of data to the dataset
- Then read the entire dataset back from the file

# Example R/W to a Subset of a Dataset



| Contents of Original Dataset Created | Contents of Dataset After 3 x 4 Subset Written |
|---|---|
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 5 5 5 5 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 5 5 5 5 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 5 5 5 5 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |
| 1 1 1 1 1 2 2 2 2 2 | 1 1 1 1 1 2 2 2 2 2 |

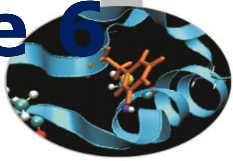
http://www.hdfgroup.org/ftp/HDF5/examples/introductory/C/h5_subset.c



# Creating Datasets in Groups

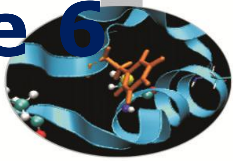
- We have shown how to create groups, datasets, and attributes.
- In this section, we show how to create datasets in groups.
- Recall that
  - H5Dcreate/h5dcreate_f creates a dataset at the location specified by a location identifier and a name.
  - similar toH5Gcreate/h5gcreate_f, the location identifier can be a file identifier or a group identifier and the name can be relative or absolute.
- The location identifier and the name together determine the location where the dataset is to be created. If the location identifier and name refer to a group, then the dataset is created in that group.

## Creating Datasets in Groups: Example 6



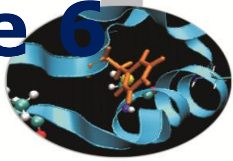
```
#include "hdf5.h"
#define FILE "groups.h5"
int main() {
   hid_t file_id, group_id, dataset_id, dataspace_id; /* identifiers */
   hsize_t dims[2];
   herr_t status;
   int i, j, dset1_data[3][3], dset2_data[2][10];

   /* Initialize the first dataset. */
   for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
                dset1_data[i][j] = j + 1;
   /* Initialize the second dataset. */
   for (i = 0; i < 2; i++)
        for (j = 0; j < 10; j++)
                dset2_data[i][j] = j + 1;
```

# Creating Datasets in Groups: Example 6



```
/* Open an existing file. */
file_id = H5Fopen(FILE, H5F_ACC_RDWR, H5P_DEFAULT);
/* Create the data space for the first dataset. */
dims[0] = 3; dims[1] = 3;
dataspace_id = H5Screate_simple(2, dims, NULL);

/* Create a dataset in group "MyGroup". */
dataset_id = H5Dcreate(file_id, "/MyGroup/dset1", H5T_STD_I32BE,
     dataspace_id, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

/* Write the first dataset. */
status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
     H5P_DEFAULT, dset1_data);
/* Close the data space for the first dataset. */
status = H5Sclose(dataspace_id);
/* Close the first dataset. */
status = H5Dclose(dataset_id);
```

# Creating Datasets in Groups: Example 6

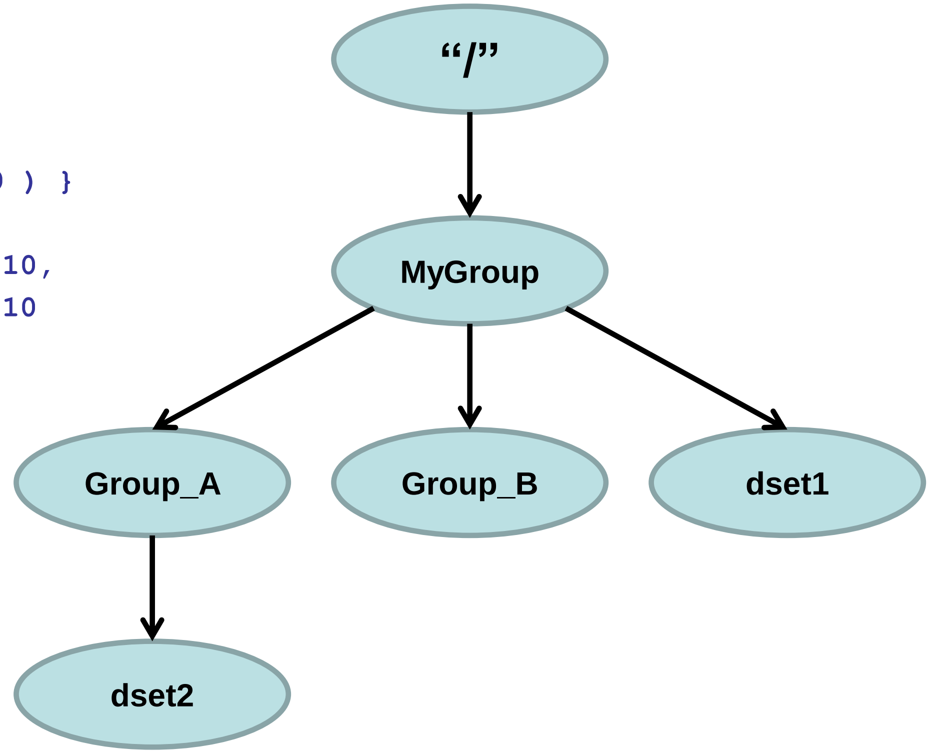


```
/* Open an existing group of the specified file. */
group_id = H5Gopen(file_id, "/MyGroup/Group_A", H5P_DEFAULT);
/* Create the data space for the second dataset. */
dims[0] = 2; dims[1] = 10;
dataspace_id = H5Screate_simple(2, dims, NULL);
/* Create the second dataset in group "Group_A". */
dataset_id = H5Dcreate(group_id, "dset2", H5T_STD_I32BE, dataspace_id,
     H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
/* Write the second dataset. */
status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
     H5P_DEFAULT, dset2_data);
/* Close the data space for the second dataset. */
status = H5Sclose(dataspace_id);
/* Close the second dataset */
status = H5Dclose(dataset_id);
/* Close the group. */
status = H5Gclose(group_id);
/* Close the file. */
status = H5Fclose(file_id); }
```

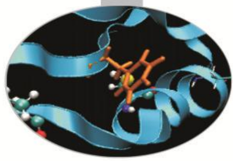
SuperComputing Applications and Innovation

## Example 6: h5dump Output

```
HDF5 "groups.h5" {
GROUP "/" {
GROUP "MyGroup" {
GROUP "Group_A" {
    DATASET "dset2" {
           DATATYPE { H5T_STD_I32BE }
           DATASPACE { SIMPLE ( 2, 10 ) / ( 2, 10 ) }
           DATA {
                      1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                      1, 2, 3, 4, 5, 6, 7, 8, 9, 10
           }
    }
}
GROUP "Group_B" {
}
DATASET "dset1" {
    DATATYPE { H5T_STD_I32BE }
    DATASPACE { SIMPLE ( 3, 3 ) / ( 3, 3 ) }
    DATA {
           1, 2, 3,
           1, 2, 3,
           1, 2, 3
    }
 }
 }
 }
}
```



# Attributes

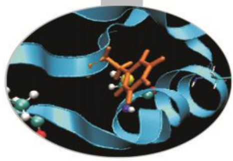


**Attributes** are small datasets that can be used to describe the nature and/or the intended usage of the object they are attached to.

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

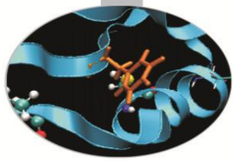


The steps to create an attribute are as follows:

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.

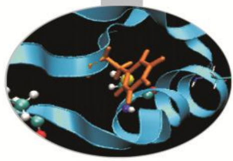
# Example 7. 1/2

```
#include “hdf5.h”
#define FILE “dset.h5”

main(){
   hid_t file_id, dataset_id, attribute_id, dataspace_id; /*
   identifier */
   hsize_t dims;
   int attr_data[2];
   herr_t status;

   /* Initialize the attribute data */
   attr_data[0] = 100;  attr_data[1] = 200;

   /* Open an existing file */
   file_id = H5FOpen(FILE, H5F_ACC_RDWR, H5P_DEFAULT);

   /* Open an existing dataset */
   dataset_id = H5DOpen(file_id, “/dset”);
```

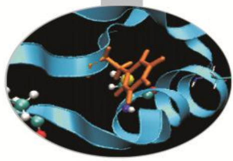


# Example 7. 2/2

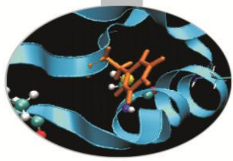
```
    /* Create the data space for the attribute */
    dims = 2;
    dataspace_id = H5Screate_simple(1, &dims, NULL);

    /* Create a dataset attribute */
    attribute_id = H5Acreate(dataset_id, “attr”, H5T_STD_I32BE,
    dataspace_id, H5P_DEFAULT);

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

    /* Close the attribute, dataspace
       dataset and file */
    status = H5Aclose(attribute_id);
    status = H5Sclose(dataspace_id);
    status = H5Dclose(dataset_id);
    status = H5Fclose(file_id);
}
```



# Example 7: h5dump Output



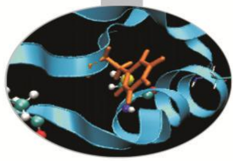
```
HDF5 "dset.h5" {
GROUP "/" {
DATASET "dset" {
DATATYPE { H5T_STD_I32BE }
DATASPACE { SIMPLE ( 4, 6 ) / ( 4, 6 ) }
DATA {
   1, 2, 3, 4, 5, 6,
   7, 8, 9, 10, 11, 12,
   13, 14, 15, 16, 17, 18,
   19, 20, 21, 22, 23, 24
}
ATTRIBUTE "attr" {
   DATATYPE { H5T_STD_I32BE }
   DATASPACE { SIMPLE ( 2 ) / ( 2 ) }
   DATA {
        100, 200
   }
}
}
}
}
```



# 1.8 vs 1.6, main differences

- 1.8 is backward compatible, provided at compile time you add:

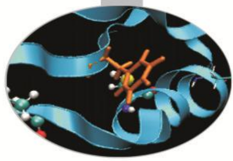
  -D H5_USE_16_API
- Support to
  - External Links, Links in a group that link to objects in a different HDF5 file
  - User-defined Links
  - Dedicated Link Interface Link API (H5L) for directly managing links
  - Enhanced Attribute Handling Faster access and more compact storage
  - Object Copying: Copying an HDF5 object to a new location within a file or in a different file
  - Dedicated Object Interface
  - C++ and Fortran Wrapper Improvements
  - ……..



# Usefull links

| | |
|---|---|
| **The HDF Group Page:** | http://hdfgroup.org/ |
| **HDF5 Home Page:** | http://hdfgroup.org/HDF5/ |
| | |
| **HDF Helpdesk:** | help@hdfgroup.org |
| **HDF Mailing Lists:** | http://hdfgroup.org/services/support.html |

**1.8 vs 1.6:**
http://www.hdfgroup.org/HDF5/doc/ADGuide/WhatsNew180.html



## QUESTIONS ???