



The HDF Group



HDF5 Advanced Topics

Elena Pourmal

The HDF Group

The 13th HDF and HDF-EOS Workshop

November 3-5, 2009



Outline

- HDF5 Datatypes
- Partial I/O



HDF5 Datatypes

Overview



HDF5 Datatypes

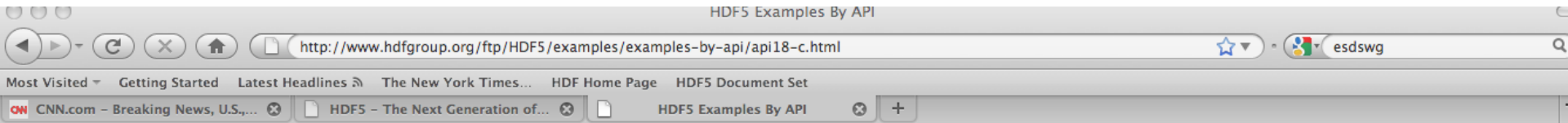
- An HDF5 datatype
 - Required description of a data element
 - the set of possible values it can have
 - Example: enumeration datatype
 - the operations that can be performed
 - Example: type conversion cannot be performed on opaque datatype
 - how the values of that type are stored
 - Example: values of variable-length type are stored in a heap in a file
 - Stored in the file along with the data it describes
 - Not to be confused with the C, C++, Java and Fortran types



HDF5 Datatypes Examples

- We provide examples how to create, write and read data of different types

<http://www.hdfgroup.org/ftp/HDF5/examples/examples-by-api/api18-c.html>



The HDF Group

HDF5 Examples By API

HDF5 1.8 C Examples





HDF5 Datatypes Examples

Datatypes:

Read / Write Array Datatypes (Attribute):	h5ex_t_arrayatt.c	[Output]
Read / Write Array Datatypes (Dataset):	h5ex_t_array.c	[Output]
Read / Write Bitfield Datatypes (Attribute):	h5ex_t_bitatt.c	[Output]
Read / Write Bitfield Datatypes (Dataset):	h5ex_t_bit.c	[Output]
Read / Write Compound Datatypes (Attribute):	h5ex_t_cmpdatt.c	[Output]
Read / Write Compound Datatypes (Dataset):	h5ex_t_cmpd.c	[Output]
Commit Named Datatype and Read Back:	h5ex_t_commit.c	[Output]
Convert Between Datatypes in Memory:	h5ex_t_convert.c	[Output]
Read / Write Complex Compound Datatype (Attribute):	h5ex_t_cpxcmpdatt.c	[Output]
Read / Write Complex Compound Datatype (Dataset):	h5ex_t_cpxcmpd.c	[Output]
Read / Write Enumerated Datatypes (Attribute):	h5ex_t_enumatt.c	[Output]
Read / Write Enumerated Datatypes (Dataset):	h5ex_t_enum.c	[Output]
Read / Write Floating Point Datatype (Attribute):	h5ex_t_floatatt.c	[Output]
Read / Write Floating Point Datatype (Dataset):	h5ex_t_float.c	[Output]



HDF5 Datatypes

- When HDF5 Datatypes are used?
 - To describe application data in a file
 - H5Dcreate, H5Acreate calls
 - Example:
 - A C application stores integer data as 32-bit little-endian signed two's complement integer; it uses H5T_SDT_I32LE with the H5Dcreate call
 - A C applications stores double precision data “as is” in the application memory; it uses H5T_NATIVE_DOUBLE with the H5Acreate call
 - A Fortran application stores array of real numbers as 64-bit big-endian IEEE format; it uses H5T_IEEE_F64BE with h5dcreate_f call; HDF5 library will perform all necessary conversions



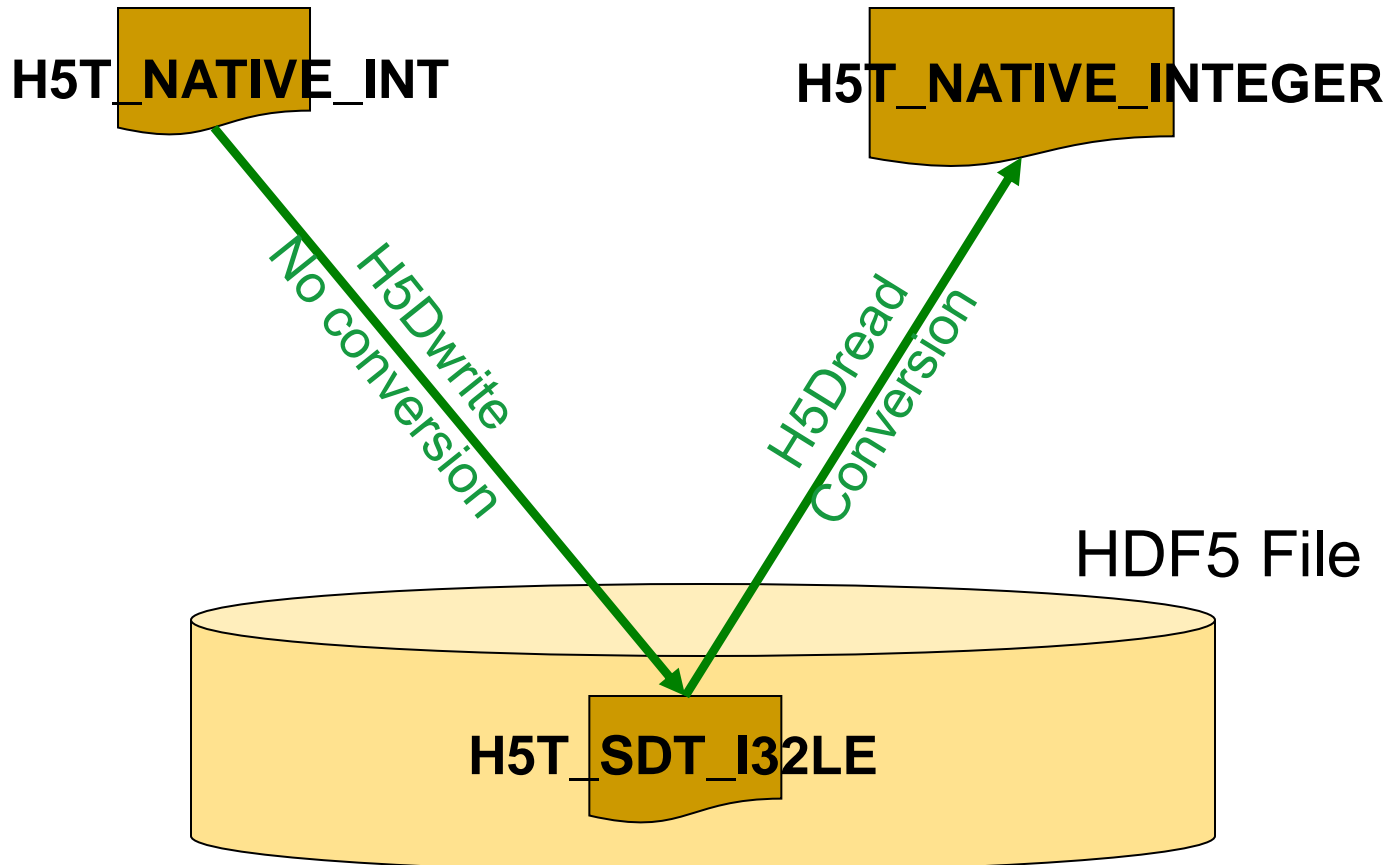
HDF5 Datatypes

- When HDF5 Datatypes are used?
 - To describe application data in memory
 - Data buffer to be written or read into with H5Dwrite/ H5Dread and H5Awrite/H5Aread calls
 - Example:
 - C application reads data from the file and stores it in an integer buffer; it uses H5T_NATIVE_INT to describe the buffer.
 - A Fortran application reads floating point data from the file and stores it an integer buffer; it uses H5T_NATIVE_INTEGER to describe the buffer;
- HDF5 library performs datatype conversion; overflow/underflow may occur.

Example

C Array of integers on Linux platform
 Native integer (**H5T_NATIVE_INT**)
 is little-endian, **4** bytes

Fortran Array of integers on AIX platform
 Native integer (**H5T_NATIVE_INTEGER**)
 is big-endian, **8** bytes



Data is stored as little-endian, converted to big-endian on read



HDF5 Datatypes

HDF5/H5T API Specification

http://www.hdfgroup.org/HDF5/doc/RM/RM_H5T.html

H5T: Datatype Interface

Datatype Object API Functions

These functions create and manipulate the datatype which describes elements of a dataset. In the following lists, *italic type* indicates a configurable macro.

The C Interfaces:

<i>General Datatype Operations</i>	<i>Atomic Datatype Properties</i>	<i>Compound Datatype Properties</i>
<ul style="list-style-type: none">• H5Tcreate• H5Topen• H5Topen1 *• H5Topen2• H5Tcommit• H5Tcommit1 *• H5Tcommit2• H5Tcommit anon	<ul style="list-style-type: none">• H5Tset size• H5Tget order• H5Tset order• H5Tget precision• H5Tset precision• H5Tget offset• H5Tset offset• H5Tget pad	<ul style="list-style-type: none">• H5Tget nmembers• H5Tget member class• H5Tget member name• H5Tget member index• H5Tget member offset• H5Tget member type• H5Tinsert• H5Tpack

Done



Example: Writing/reading an Array to HDF5 file

- Calls you've already seen in Intro Tutorial
 - `H5LTmake_dataset`
 - `H5Dwrite`, `H5Dread`
- APIs to handle specific C data type
 - `H5LTmake_dataset_<*>`
 - `<*>` is one of "char", "short", "int", "long", "float", "double", "string"
 - All data array is written (no sub-setting)
 - Data stored in a file as it is in memory
 - `H5LTread_dataset`,
`H5LTread_dataset_<*>`



Example: Read data into array of longs

```
#include "hdf5.h"
#include "hdf5_hl.h"
int main( void ){
    long *data;

    ...
    /* Open file from ex_lite1.c */
    file_id = H5Fopen ("ex_lite1.h5", H5F_ACC_RDONLY,
        H5P_DEFAULT);
    /* Get information about dimensions to allocate memory buffer */
    status = H5LTget_dataset_ndims(file_id,"/dset",rank);
    status = H5LTget_dataset_info(file_id,"/dset",dims,dt_class,dt_size);
    /* Allocate buffer to read data in */
    data = (long*)malloc(...
    /* Read dataset */
    status = H5LTread_dataset_long(file_id,"/dset",data);
    /
```



Example: Read data into array of longs

```
#include hdf5.h

...
long *rdata;

....
/* Open file and dataset. */
file_id = H5Fopen ("ex_lite1.h5", H5F_ACC_RDONLY, H5P_DEFAULT);
dset_id = H5Dopen (file, "/dset", H5P_DEFAULT);

/* Get information about dimensions to allocate memory buffer */
space = H5Dget_space (dset);
rank = H5Sget_simple_extent_dims (space, dims, NULL);

status = H5Dread (dset, H5T_NATIVE_LONG, H5S_ALL, H5S_ALL,
    H5P_DEFAULT, rdata);
```



Basic Atomic HDF5 Datatypes



Basic Atomic Datatypes

- Integers & floats
- Strings (fixed and variable size)
- Pointers - references to objects and dataset regions
- Bitfield
- Opaque

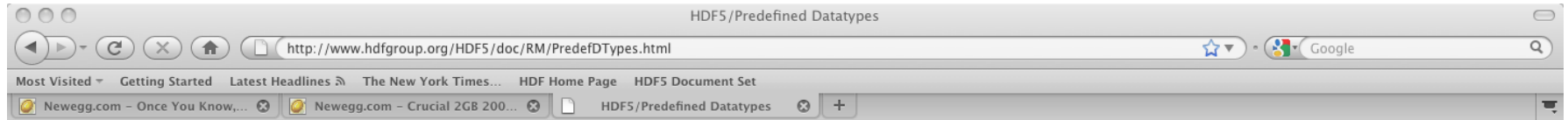


HDF5 Predefined Datatypes

- HDF5 Library provides predefined datatypes (symbols) for all basic atomic datatypes except opaque datatype
 - H5T_<arch>_<base>
 - Examples:
 - H5T_IEEE_F64LE
 - H5T_STD_I32BE
 - H5T_C_S1, H5T_FORTRAN_S1
 - H5T_STD_B32LE
 - H5T_STD_REF_OBJ, H5T_STD_REF_DSETREG
 - H5T_NATIVE_INT
- *Predefined datatypes **do not have constant values**; initialized when library is initialized*



HDF5 Pre-defined Datatypes



Predefined native datatypes

These are the datatypes detected by `h5detect`. Their names differ from other HDF5 datatype names as follows:

- Instead of a class name, precision, and byte order as the last component, they have a C-like datatype name.
- If the datatype begins with `u`, then it is the unsigned version of the integer datatype; other integer datatypes are signed.
- The datatype `LLONG` corresponds to C's `long_long` and `LDOUBLE` is `long_double`. These datatypes might be the same as `LONG` and `DOUBLE`, respectively.

```
H5T_NATIVE_CHAR          H5T_NATIVE_FLOAT
H5T_NATIVE_SCHAR         H5T_NATIVE_DOUBLE
H5T_NATIVE_UCHAR         H5T_NATIVE_LDOUBLE

H5T_NATIVE_SHORT         H5T_NATIVE_B8
H5T_NATIVE_USHORT        H5T_NATIVE_B16
                          H5T_NATIVE_B32
                          H5T_NATIVE_B64
                          H5T_NATIVE_OPAQUE
```

Done



HDF5 Predefined Datatypes

IEEE floating point datatypes

- 32-bit and 64-bit
- Big-endian and little-endian

```
H5T_IEEE_F32BE
H5T_IEEE_F32LE
H5T_IEEE_F64BE
H5T_IEEE_F64LE
```

Standard datatypes

- Signed integer (2's complement), unsigned integer, and bitfield
- 8-bit, 16-bit, 32-bit, and 64-bit
- Big-endian and little-endian

```
H5T_STD_I8BE      H5T_STD_U8BE      H5T_STD_B8BE
H5T_STD_I8LE      H5T_STD_U8LE      H5T_STD_B8LE
H5T_STD_I16BE     H5T_STD_U16BE     H5T_STD_B16BE
H5T_STD_I16LE     H5T_STD_U16LE     H5T_STD_B16LE
H5T_STD_I32BE     H5T_STD_U32BE     H5T_STD_B32BE
H5T_STD_I32LE     H5T_STD_U32LE     H5T_STD_B32LE
H5T_STD_I64BE     H5T_STD_U64BE     H5T_STD_B64BE
H5T_STD_I64LE     H5T_STD_U64LE     H5T_STD_B64LE
```



HDF5 integer datatype

- HDF5 supports 1,2,4,and 8 byte signed and unsigned integers in memory and in the file
- Support differs by language
- C language
 - All C integer types including **C99 extended integer types** (when available)
 - Examples:
 - H5T_NATIVE_INT16 for **int16_t**
 - H5T_NATIVE_INT_LEAST64 for **int_least64_t**
 - H5T_NATIVE_UINT_FAST16 for **uint_fast16_t**



HDF5 integer datatype

- Fortran language
 - In memory supports only Fortran “integer”
 - Examples:
 - `H5T_NATIVE_INTEGER` for `integer`
 - In the file supports all HDF5 integer types
 - Example: one-byte integer has to be represented by integer in memory; can be stored as one-byte integer by creating an appropriate dataset (`H5T_SDT_I8LE`)
 - Next major release of HDF5 will support ANY kinds of Fortran integers



HDF5 floating-point datatype

- HDF5 supports 32 and 64-bit floating point IEEE big-endian, little-endian types in memory and in the file
- Support differs by language
- C language
 - `H5T_IEEE_F64BE` and `H5T_IEEE_F32LE`
 - `H5T_NATIVE_FLOAT`
 - `H5T_NATIVE_DOUBLE`
 - `H5T_NATIVE_LDOUBLE`



HDF5 floating-point datatype

- Fortran language
 - In memory supports only Fortran “real” and “double precision” (obsolete)
 - Examples:
 - H5T_NATIVE_REAL for **real**
 - H5T_NATIVE_DOUBLE for **double precision**
 - In the file supports all HDF5 floating-point types
 - Next major release of HDF5 will support ANY kinds of Fortran reals



HDF5 string datatype

- HDF5 strings are characterized by
 - The way each element of a string type is stored in a file
 - NULL terminated (C type string)
char *mystring[]="Once upon a time";
HDF5 stores <Once upon a time/0>
 - Space padded (Fortran string)
character(len=16) :: mystring='Once upon a time'
HDF5 stores <Once upon a time> and adds spaces if required
 - The sizes of elements in the same dataset or attribute
 - Fixed-length string
 - Variable-length string



Example: Creating fixed-length string

- C Example: “Once upon a time” has 16-characters

```
string_id = H5Tcopy(H5T_C_S1);  
H5Tset_size(string_id, size);
```

- Size value have to include accommodate “/0”, i.e., size=17 for “Once upon a time” string
- Overhead for short strings, e.g., “Once” will have extra 13 bytes allocated for storage
- Compressed well



Example: Creating variable-length string

- C example:

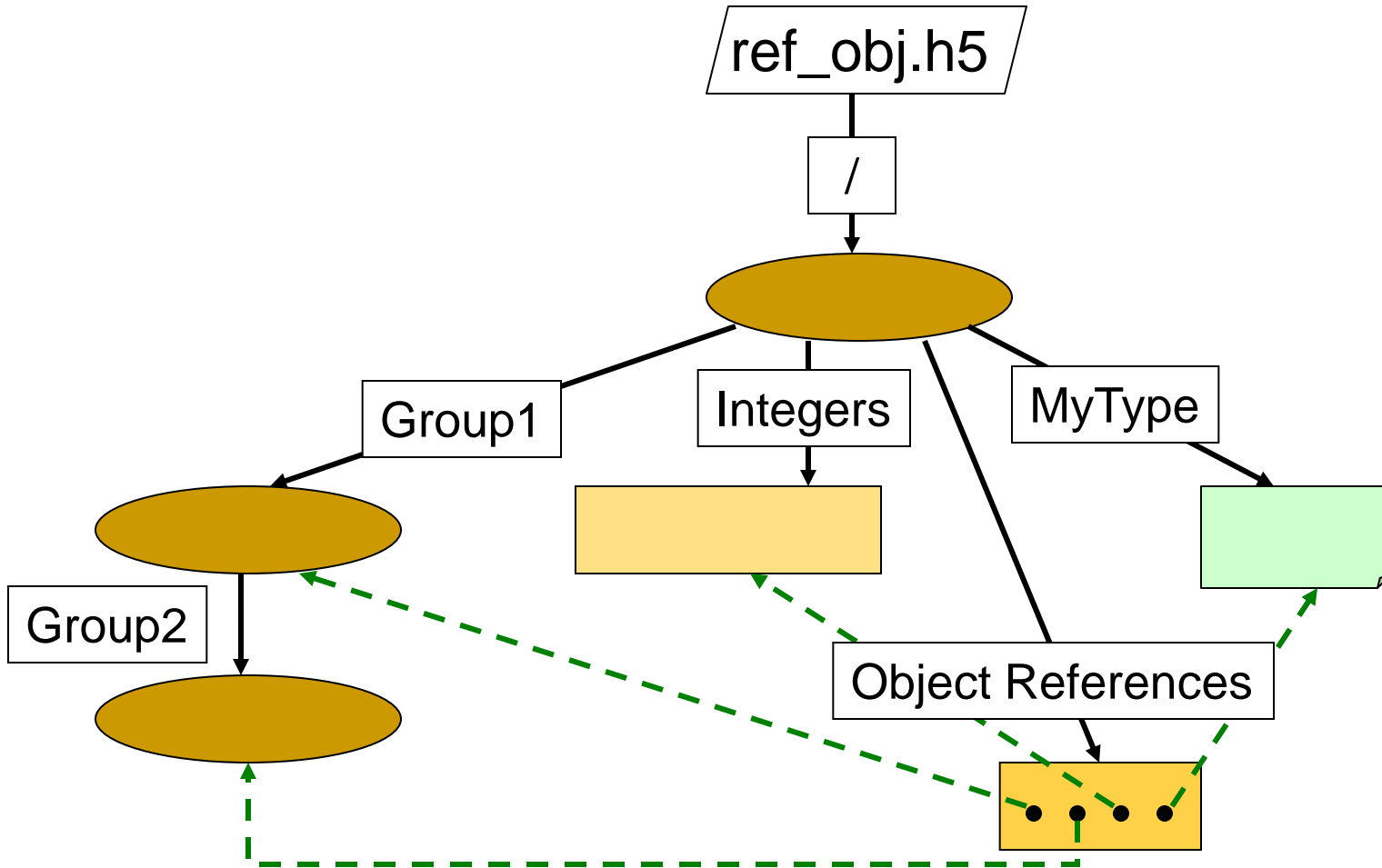
```
string_id = H5Tcopy(H5T_C_S1);  
H5Tset_size(string_id, H5T_VARIABLE);
```

- Overhead to store and access data
- Cannot be compressed (may be in the future)



Reference Datatype

- Reference to an HDF5 object
 - Pointer to a group or a dataset in a file
 - Predefined datatype `H5T_STD_REG_OBJ` describe object references





Reference to Object

```
h5dump -d /"object_reference" ref_obj.h5
```

```
DATASET "OBJECT_REFERENCES" {  
    DATATYPE  H5T_REFERENCE  
    DATASPACE  SIMPLE { ( 4 ) / ( 4 ) }  
    DATA {  
        (0): GROUP 808 /GROUP1 , GROUP 1848 /GROUP1/GROUP2 ,  
        (2): DATASET 2808 /INTEGERS , DATATYPE 3352 /MYTYPE  
    }  
}
```



Reference to Object

- Create a reference to group object

```
H5Rcreate(&ref[1], fileid, "/GROUP1/GROUP2",  
H5R_OBJECT, -1);
```

- Write references to a dataset

```
H5Dwrite(dsetr_id, H5T_STD_REF_OBJ, H5S_ALL,  
H5S_ALL, H5P_DEFAULT, ref);
```

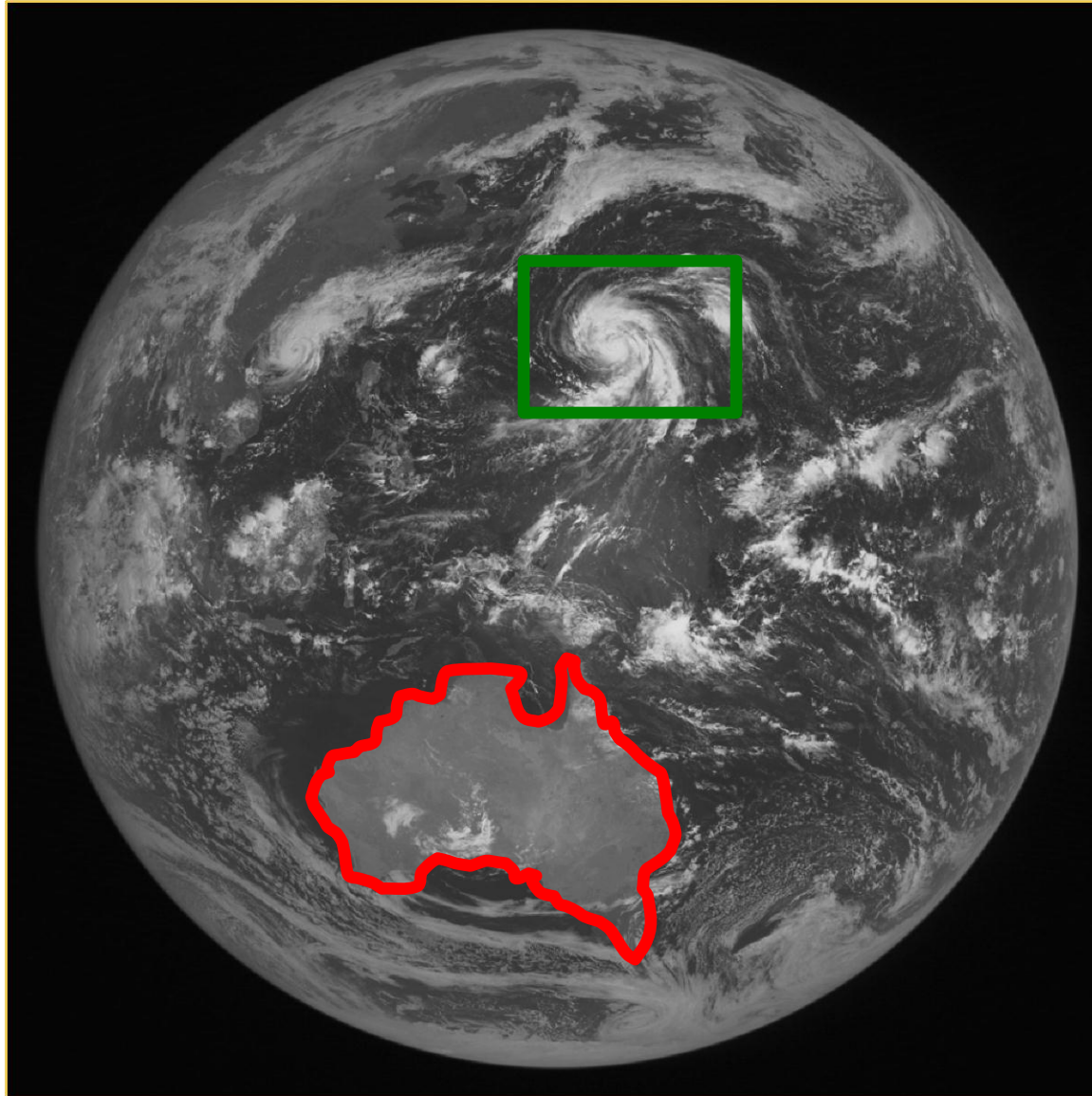
- Read reference back with H5Dread and find an object it points to

```
type_id = H5Rdereference(dsetr_id, H5R_OBJECT,  
&ref[3]);
```

```
name_size = H5Rget_name(dsetr_id, H5R_OBJECT,  
&ref_out[3], (char*)buf, 10);
```



Saving Selected Region in a File



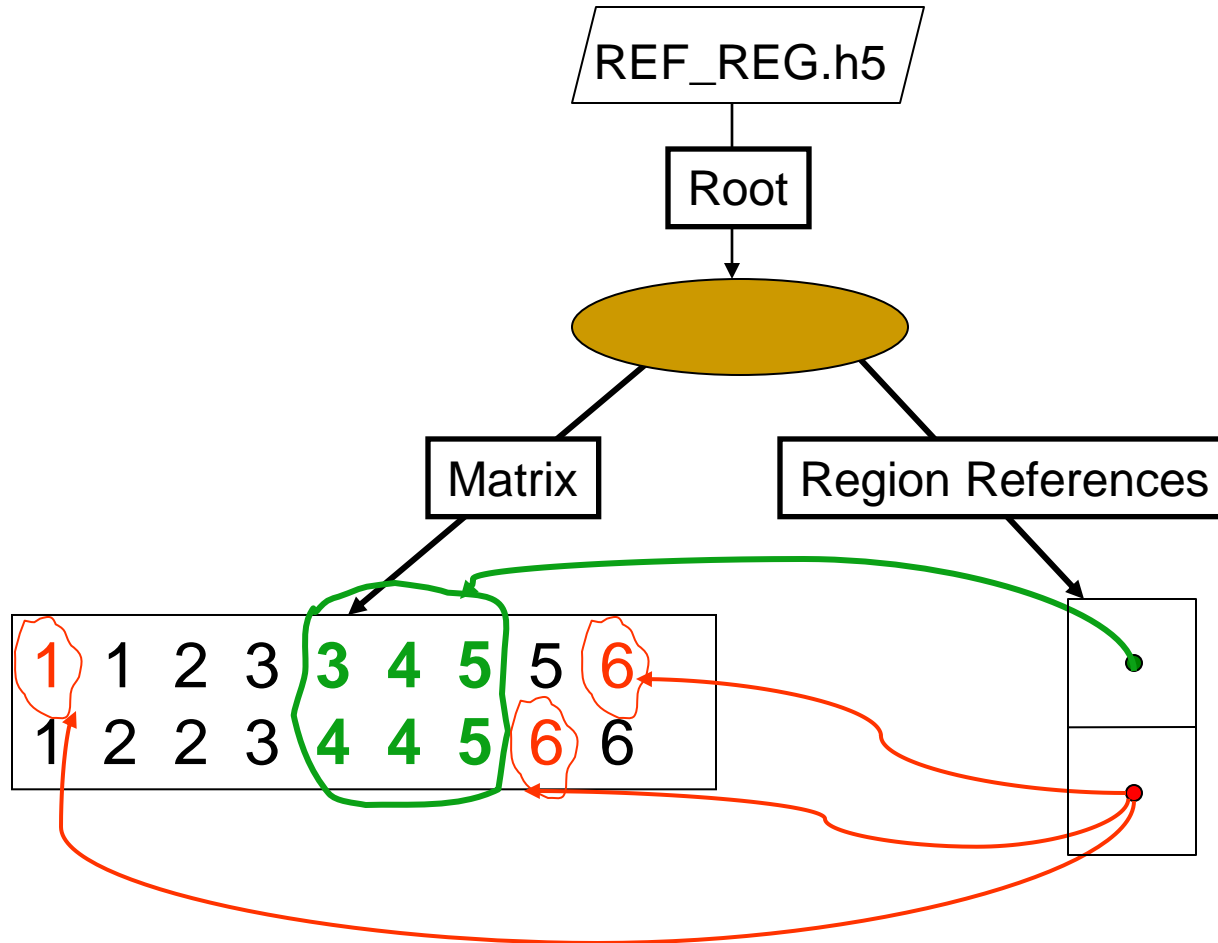


Reference Datatype

- Reference to a dataset region (or to selection)
 - Pointer to the dataspace selection
 - Predefined datatype `H5T_STD_REF_DSETREG` to describe regions



Reference to Dataset Region





Reference to Dataset Region

Example

```
dsetr_id = H5Dcreate(file_id,  
"REGION REFERENCES", H5T_STD_REF_DSETREG,  
...);
```

```
H5Sselect_hyperslab(space_id,  
H5S_SELECT_SET, start, NULL, ...);  
H5Rcreate(&ref[0], file_id, "MATRIX",  
H5R_DATASET_REGION, space_id);
```

```
H5Dwrite(dsetr_id, H5T_STD_REF_DSETREG,  
H5S_ALL, H5S_ALL, H5P_DEFAULT, ref);
```



Reference to Dataset Region

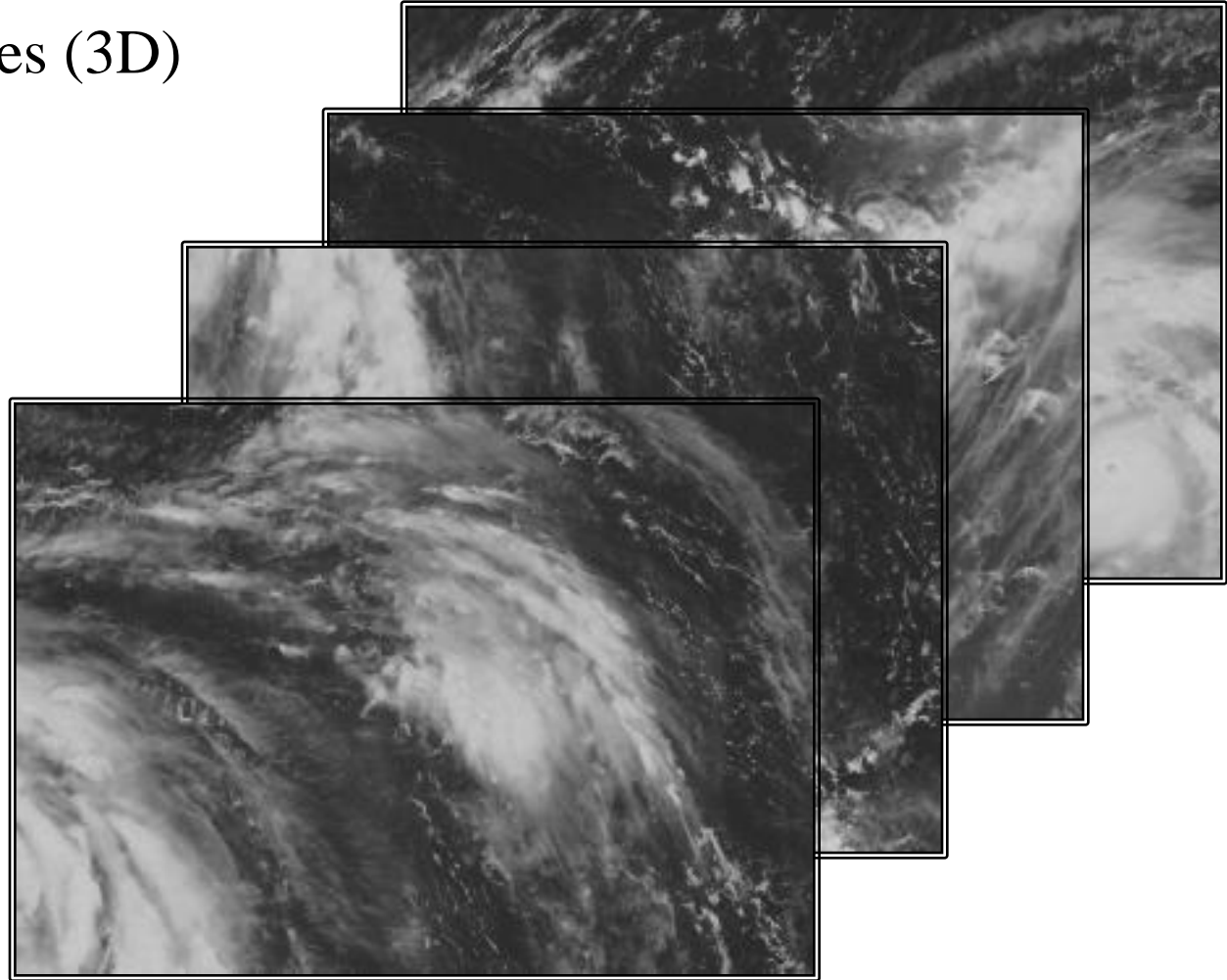
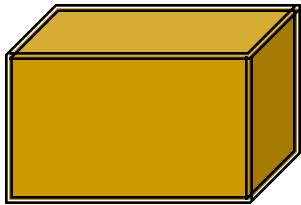
```
HDF5 "REF_REG.h5" {
GROUP "/" {
  DATASET "MATRIX" {
    .....
  }
  DATASET "REGION_REFERENCES" {
    DATATYPE H5T_REFERENCE
    DATASPACE SIMPLE { ( 2 ) / ( 2 ) }
    DATA {
      (0) : DATASET /MATRIX { (0,3) - (1,5) },
      (1) : DATASET /MATRIX { (0,0) , (1,6) , (0,8) }
    }
  }
}
}
```



Part II

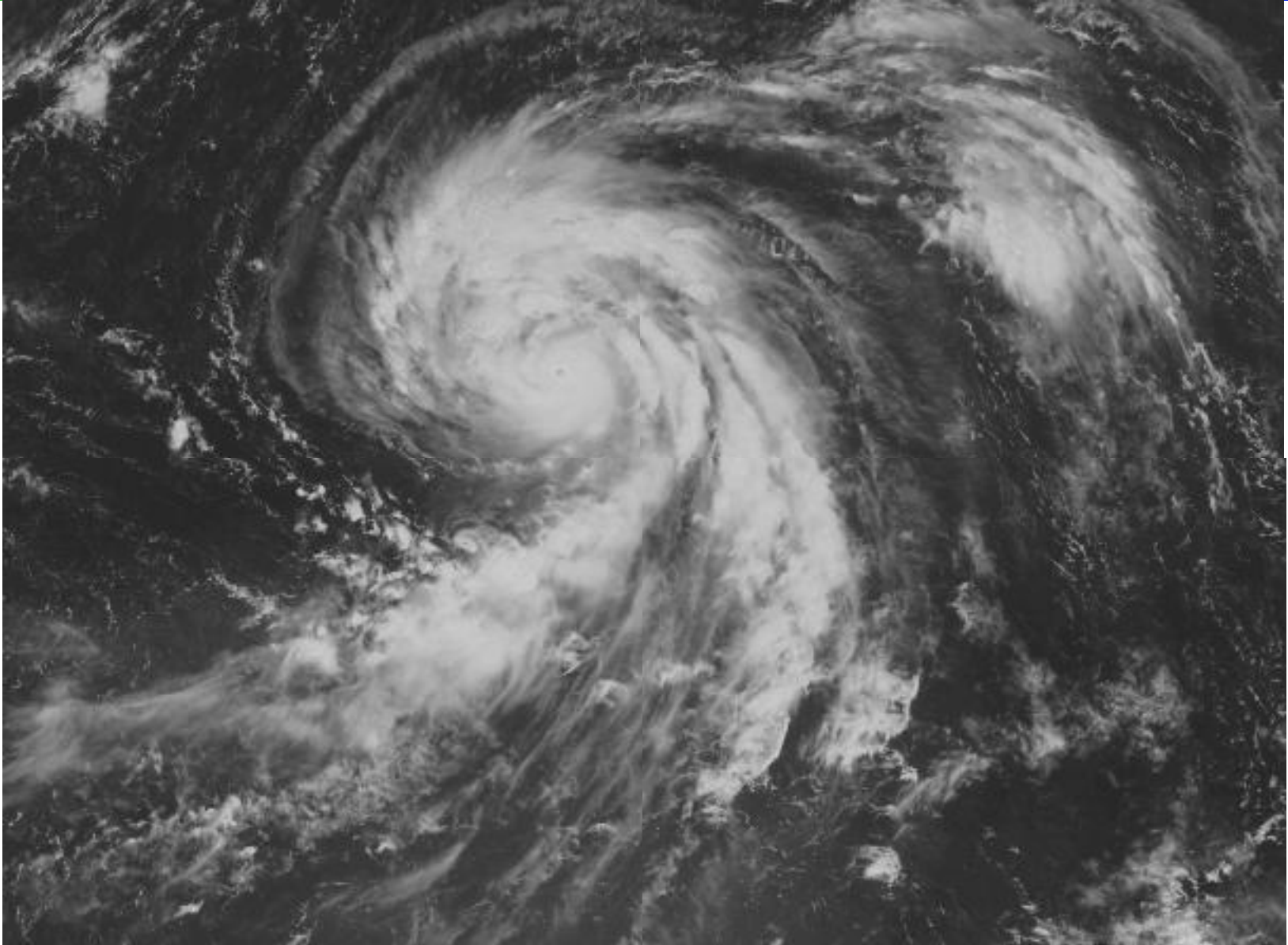
Working with subsets

Array of images (3D)



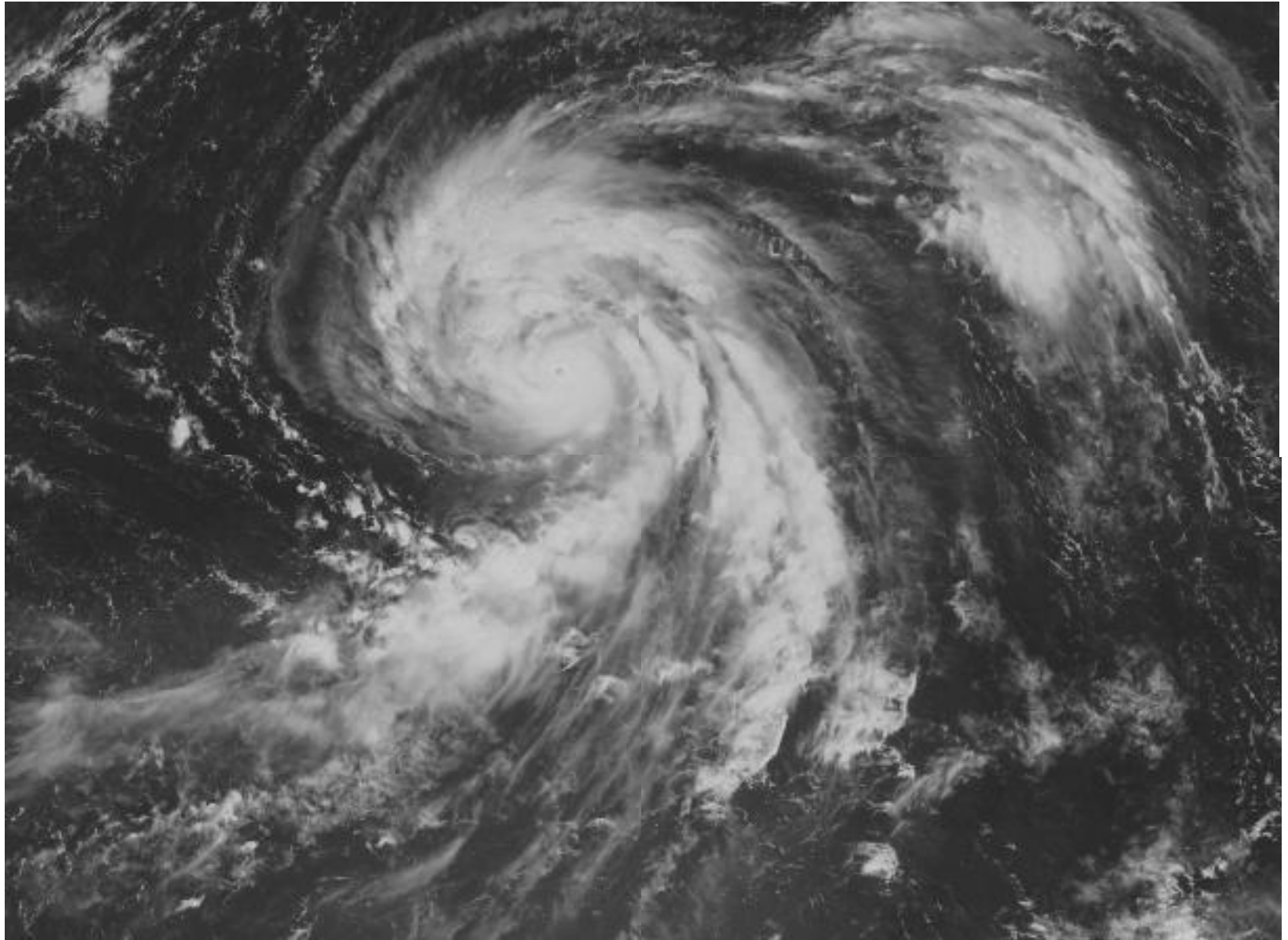


Display data another way ...



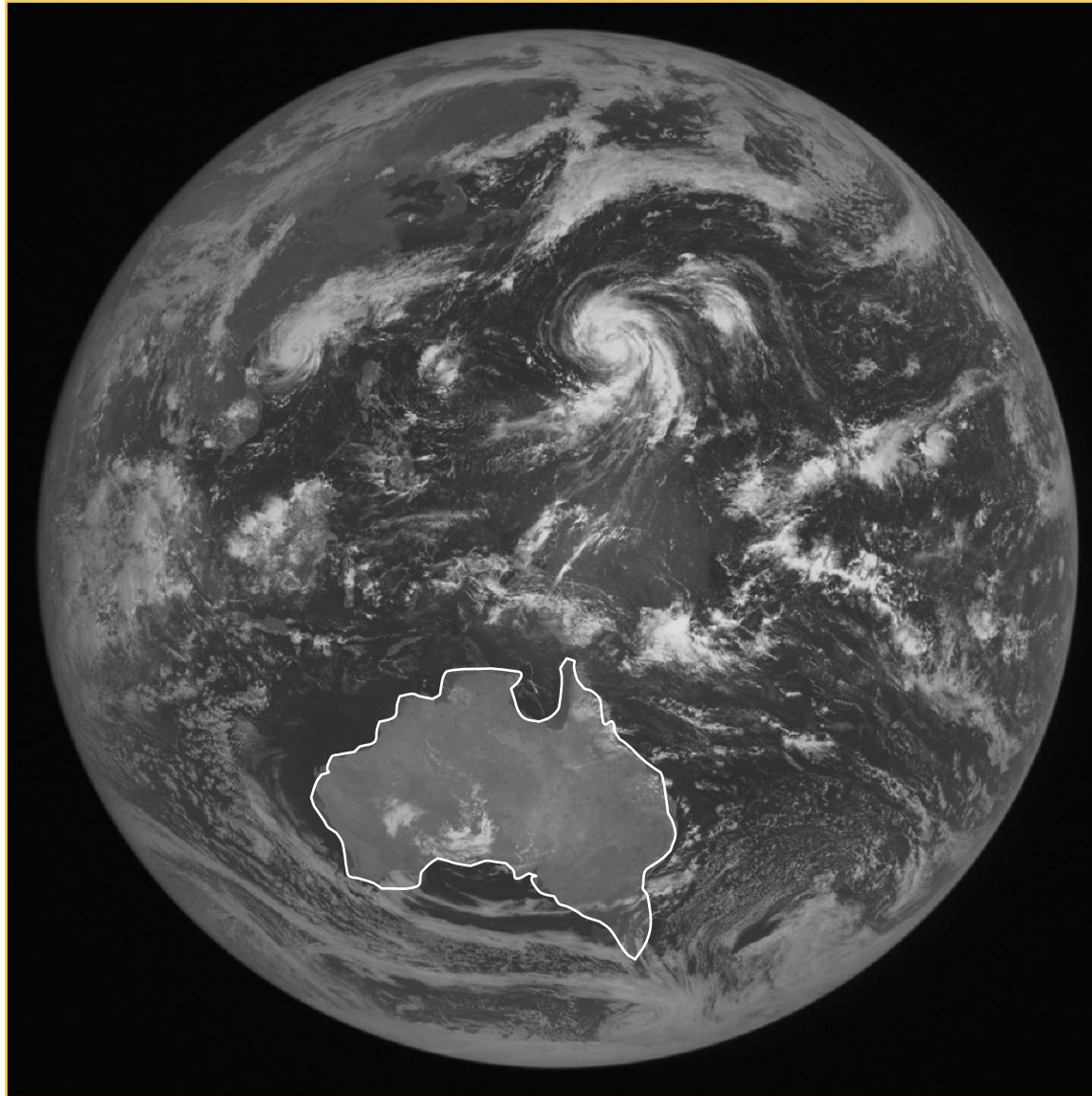


Data is too big to read....





Refer to a region...





HDF5 Library Features

- HDF5 Library provides capabilities to
 - Describe subsets of data and perform write/read operations on subsets
 - Hyperslab selections and partial I/O
 - Store descriptions of the data subsets in a file
 - Object references
 - Region references
 - Use efficient storage mechanism to achieve good performance while writing/reading subsets of data
 - Chunking, compression



Partial I/O in HDF5



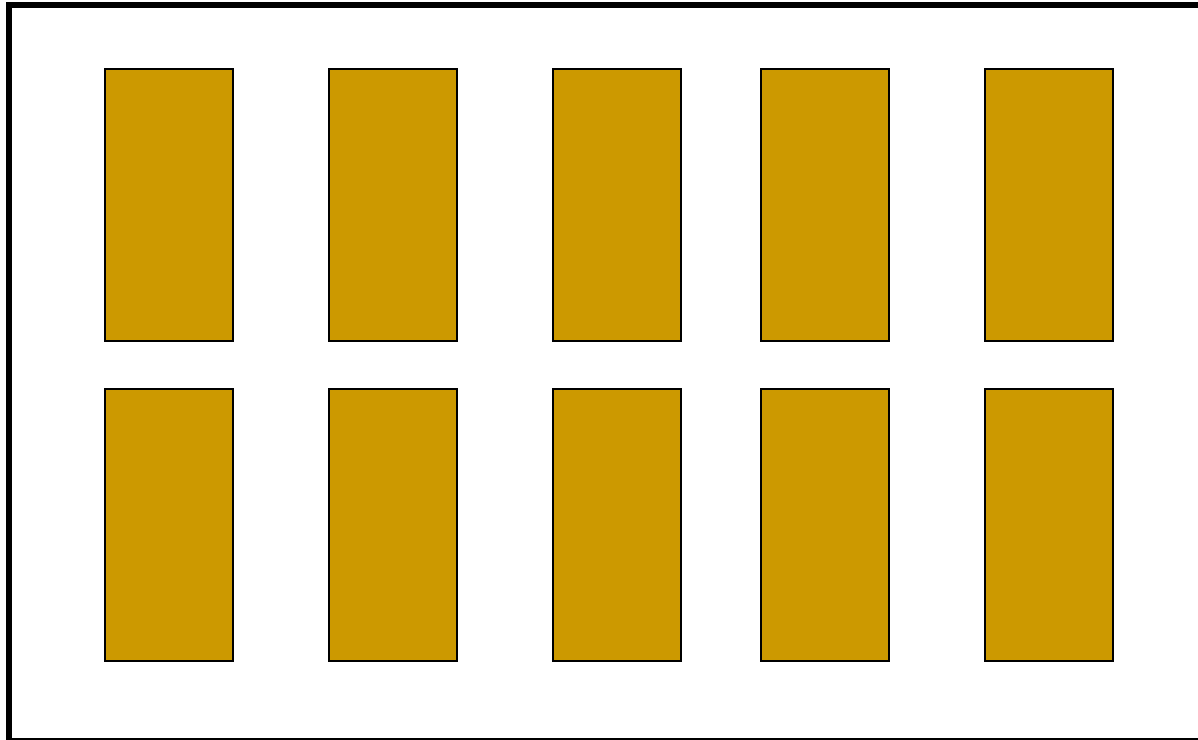
How to Describe a Subset in HDF5?

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

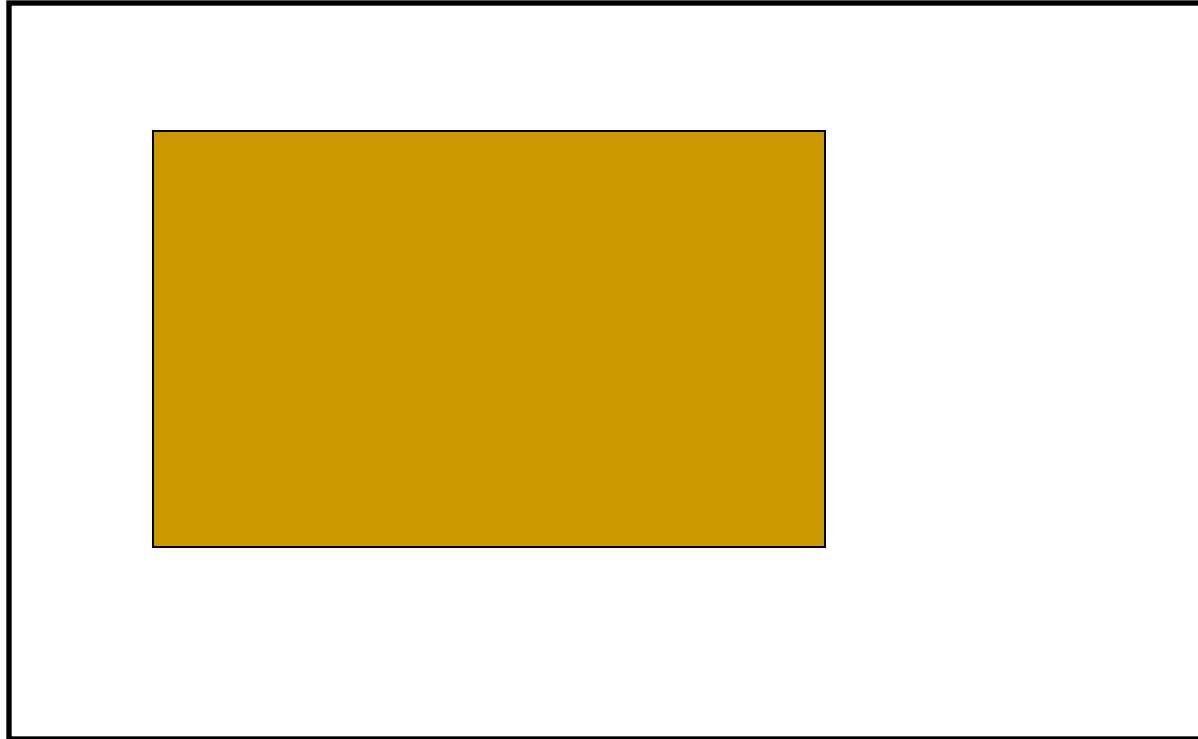


Types of Selections in HDF5

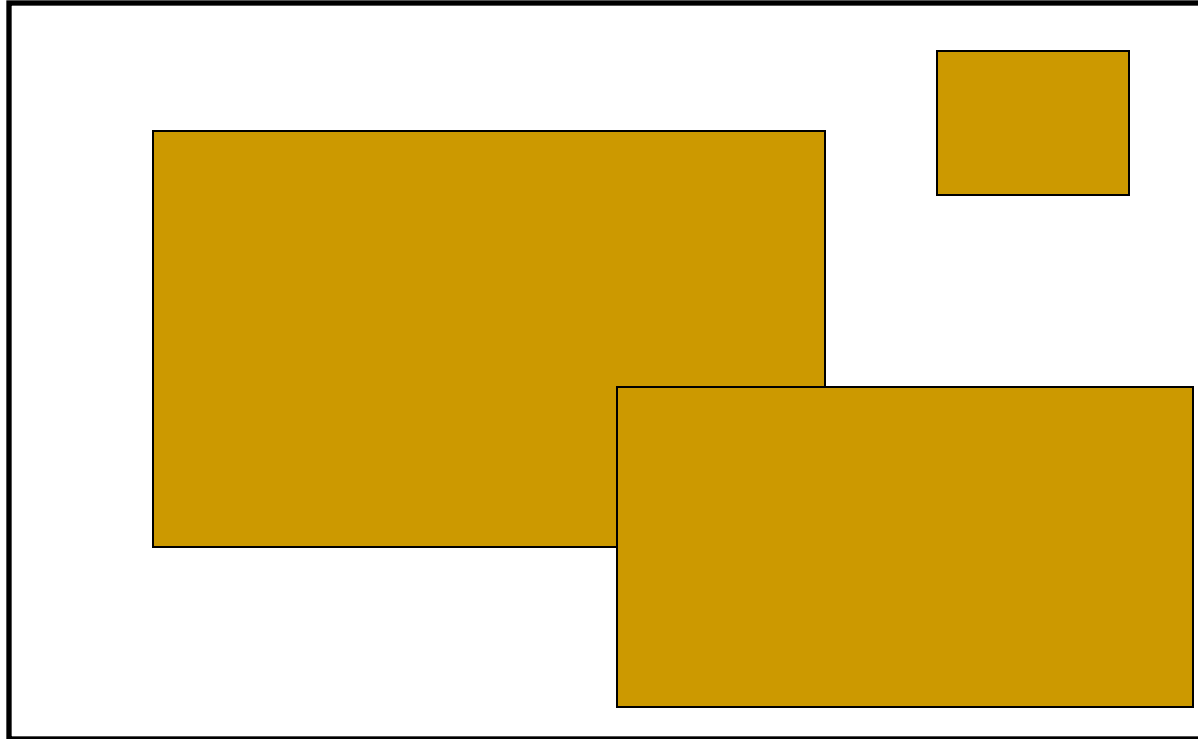
- Two types of selections
 - Hyperslab selection
 - Regular hyperslab
 - Simple hyperslab
 - Result of set operations on hyperslabs (union, difference, ...)
 - Point selection
- Hyperslab selection is especially important for doing parallel I/O in HDF5 (See Parallel HDF5 Tutorial)



Collection of regularly spaced blocks of equal size




Contiguous subset or sub-array

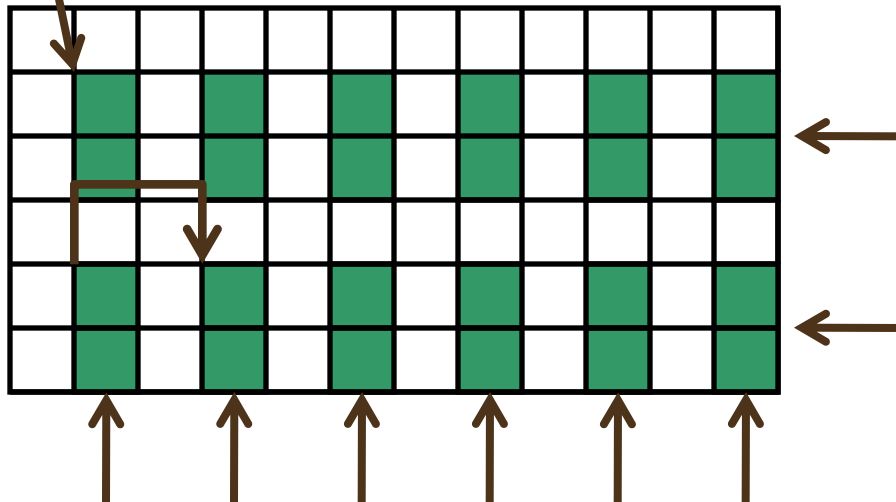


Result of union operation on three simple hyperslabs



Hyperslab Description

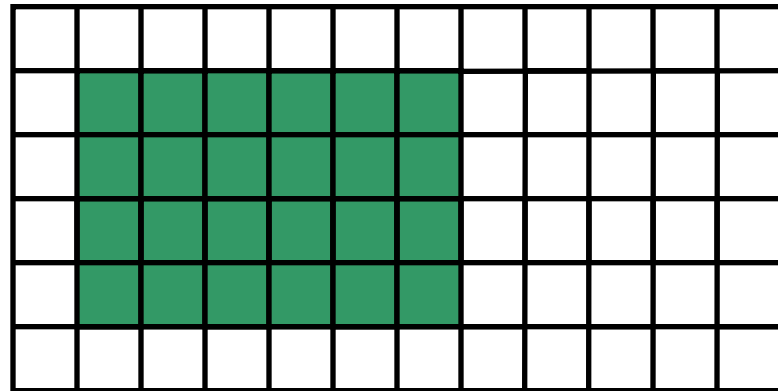
- Start - starting location of a hyperslab (1,1)
- Stride - number of elements that separate each block (3,2)
- Count - number of blocks (2,6)
- Block - block size (2,1) 
- *Everything is “measured” in number of elements*





Simple Hyperslab Description

- Two ways to describe a simple hyperslab
- As *several* blocks
 - **Stride** – (1,1)
 - **Count** – (2,6)
 - **Block** – (2,1)
- As *one* block
 - **Stride** – (1,1)
 - **Count** – (1,1)
 - **Block** – (4,6)



No performance penalty for one way or another



H5Sselect_hyperslab Function

space_id Identifier of dataspace

op Selection operator
H5S_SELECT_SET or H5S_SELECT_OR

start Array with starting coordinates of hyperslab

stride Array specifying which positions along a dimension to select

count Array specifying how many blocks to select from the dataspace, in each dimension

block Array specifying size of element block
(NULL indicates a block size of a single element in a dimension)



Reading/Writing Selections

Programming model for reading from a dataset in a file

1. Open a dataset.
2. Get file dataspace handle of the dataset and specify subset to read from.
 - a. **H5Dget_space** returns file dataspace handle
 - a. File dataspace describes array stored in a file (number of dimensions and their sizes).
 - b. **H5Sselect_hyperslab** selects elements of the array that participate in I/O operation.
3. Allocate data buffer of an appropriate shape and size



Programming model (continued)

4. Create a memory dataspace and specify subset to write to.
 1. Memory dataspace describes data buffer (its rank and dimension sizes).
 2. Use **H5Screate_simple** function to create memory dataspace.
 3. Use **H5Sselect_hyperslab** to select elements of the data buffer that participate in I/O operation.
5. Issue **H5Dread** or **H5Dwrite** to move the data between file and memory buffer.
6. Close file dataspace and memory dataspace when done.



Example : Reading Two Rows

Data in a file
4x6 matrix

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

Buffer in memory
1-dim array of length 14

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
----	----	----	----	----	----	----	----	----	----	----	----	----	----



Example: Reading Two Rows

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

`start` = {1,0}
`count` = {2,6}
`block` = {1,1}
`stride` = {1,1}

```
filespace = H5Dget_space (dataset);  
H5Sselect_hyperslab (filespace, H5S_SELECT_SET,  
                    start, NULL, count, NULL)
```



Example: Reading Two Rows

```
start[1] = {1}  
count[1] = {12}  
dim[1]   = {14}
```

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
----	----	----	----	----	----	----	----	----	----	----	----	----	----

```
memspace = H5Screate_simple(1, dim, NULL);  
H5Sselect_hyperslab (memspace, H5S_SELECT_SET,  
                    start, NULL, count, NULL)
```



Example: Reading Two Rows

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

```
H5Dread (... , ..., memspace, filespace, ..., ...);
```

-1	7	8	9	10	11	12	13	14	15	16	17	18	-1
----	---	---	---	----	----	----	----	----	----	----	----	----	----



Things to Remember

- *Number* of elements selected in a file and in a memory buffer *must be the same*
 - `H5Sget_select_npoints` returns number of selected elements in a hyperslab selection
- HDF5 partial I/O is tuned to move data between selections that have the same dimensionality; *avoid* choosing *subsets* that have *different ranks* (as in example above)
- *Allocate a buffer of an appropriate size* when reading data; use `H5Tget_native_type` and `H5Tget_size` to get the correct size of the data element in memory.



Thank You!



Acknowledgements

This work was supported by cooperative agreement number NNX08AO77A from the National Aeronautics and Space Administration (NASA).

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author[s] and do not necessarily reflect the views of the National Aeronautics and Space Administration.



Questions/comments?