CH 380: Lecture 8
Scientific Data and I/O Libraries

Dr. Christopher Simmons
csim@ices.utexas.edu

Scientific Data Files

- Most of us need to read and write data at some point in order to do some useful analysis:
  - Read geometry configurations for simulation driven research
  - Data mining algorithms
  - Post-process generated data (simple statistics or complicated iso-surface animations)
  - Continuing from a restart file (does everyone know what a restart file is?)
- Data is often written to disk as a byte-stream (C coded I/O) or as sequential I/O (Fortran code):
  - In a byte stream the data is written contiguously to disk
  - In (Fortran standard) sequential files, the data is also written sequentially to disk, but a 4-byte record-size value is written at the beginning and end of each record
- Note: Fortran Direct I/O doesn't have record markers, but all records (writes/reads) can only have one size, making it unpopular

Scientific Data Files

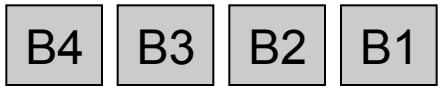
- There are three main aspects of data files that effect portability:
  - floating point representation
    - some older machines, such as the Cray SV1, employ their own representation for floating point numbers (17-bit exponent, 47 bit mantissa), while most machines now use the **IEEE 754** representation (11-bit exponent, 52-bit mantissa)
  - **byte-ordering**
  - I/O file formatting (eg. Fortran record markers), order of the variables that are written

Scientific Data Elements

- Byte Ordering
  - a **"*byte*"** is the lowest addressable storage unit
  - A **"*word*"** refers to a group of bytes, and is often used to represent a number:
    - in C, floats are usually 4 bytes and double are 8 bytes
    - in Fortran, reals are often 4 bytes, and double precision variables are 8 bytes
  - There are two different ways that the individual bytes of a word are stored in disk and memory. Note, this ordering has nothing to do with the individual bits in each byte-- only the order of how the bytes are stored.
  - These two orderings are called ***little*** and ***big Endian*** storage order

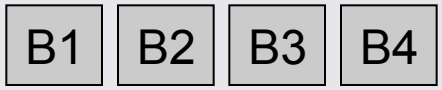
Little vs Big Endian Storage

Word (least significant digits in B1)



Little Endian

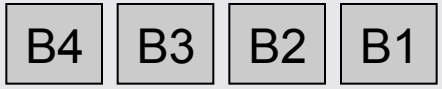
Little Endian means the bytes are stored from the least significant byte to the most, beginning at the lowest address. The word is stored "little end first"



Base Address +0 +1 +2 +3

Big Endian

Big Endian means the bytes are stored from the most significant byte to the lowest, beginning at the lowest address. The word is stored "big end first"



Base Address +0 +1 +2 +3

Little vs Big Endian Storage

- Example ordering of common architectures:

Processor	Endianess
Pentium, Xeon	little-endian
Itanium 2	little-endian
AMD, Athlon, Opteron	little-endian
Opteron	little-endian
POWERPC (Blue Gene, G5 - Old MACs)	big-endian
Power3/4/5/6/7 (AIX)	big-endian
MIPS (IRIX)	big-endian
Cray X1	big-endian

Q: what happens if you read big endian files on a little endian platform?

Little vs. Big Endian Conversion Utilities

- Utilities exist for converting between formats at runtime (e.g. compiler dependent options or C macros)
- Be sure to read the compiler man pages: Intel Fortran compiler option for converting I/O to big-endian output for predefined I/O units:

Intel Environment Variable (units=10,11,12):
C SHELL: setenv F_UFMTENDIAN 10,11,12
BASH: export F_UFMTENDIAN=10,11,12

- But, why bother with such craziness when there are portable, extensible, and efficient I/O libraries available for scientific computing?

Portable I/O Libraries

- Platform independent I/O Libraries
  - Low-level
    - XDR (low-level)
  - Scientific Orientation (*arrays of numbers*)
    - NetCDF
    - HDF

Platform Independent Binary Files

- XDR ("eXternal Data Representation)
  - Developed originally by SUN
  - Using XDR API gives platform independent binary files
  - `man xdr_vector`
  - `man xdr_string`
- NETCDF – Unidata Network Common Data Form
  - Common format used in Climate/Weather/Ocean applications
  - http://my.unidata.ucar.edu/content/software/netcdf/docs.html
- HDF - Hierarchical Data Format developed by NCSA
  - extends beyond scientific arrays (can include graphics)
  - Example uses: many scientific apps, Boeing flight test data; special effects (smoke, wind, clouds) for the *Lord of the Rings* trilogy

netCDF Overview

- netCDF stands for Network Common Data Format
  - a binary data format standard for exchanging scientific data
  - developed and supported by Unidata/UCAR (University Corporation for Atmospheric Research - a nonprofit consortium of 66 universities)
- NetCDF data is:
  - **Self-Describing**: file includes information about the data it contains.
  - **Portable:** file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers
  - **Appendable:** data may be appended to a properly structured netCDF file without copying the dataset or redefining its structure.
  - **Sharable:** one writer and multiple readers may simultaneously access the same netCDF file.
  - **Archivable:** access to all earlier forms of netCDF data will be supported by current and future versions of the software
- Data access in netCDF is direct, not sequential
  - This design means that a small subset of a large dataset may be accessed efficiently, without first reading through all the preceding data (and when the order in which data is read is different from the way it is written)

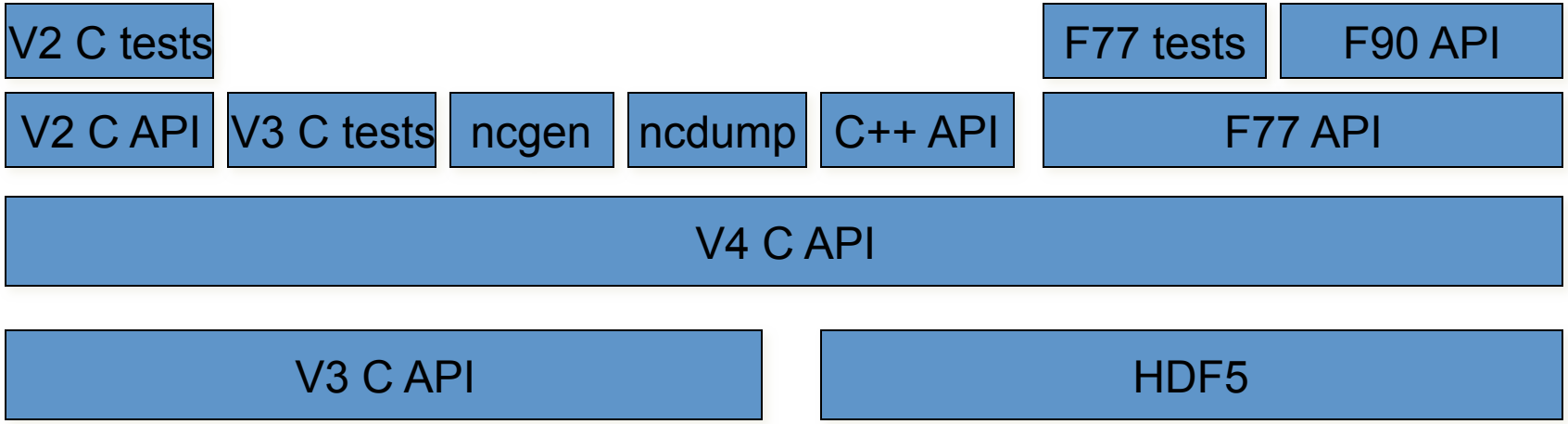
netCDF Overview

- netCDF provides an effective way to store and retrieve scientific data through binaries as well as common language bindings
- APIs exist for reading and writing data:
  - C/C++
  - Fortran
  - Java
  - Perl, MATLAB, Python, and Ruby
- netCDF is available as a precompiled library on TACC systems (*which provides the C/C++ and Fortran interfaces*)

Why Use Standard I/O libraries?

- Facilitate the use of common datasets by distinct applications
- Permit datasets to be transported between or shared by dissimilar computers transparently (without translation)
- Reduce programming effort usually spent interpreting formats
- Reduce errors from misinterpreting data and ancillary data
- Output data from one application into another
- Establish an interface standard which simplifies the inclusion of new software into existing system
- Performance tuned – some I/O libraries support parallel I/O

Software Architecture of NetCDF-4



- Fortran, C++ and V2 APIs are all built on top of the C API.
- Other language APIs (perl, python, MatLab, etc.) also use the C API

netCDF Dataset Objects

- Variables: named arrays
  - name, type, shape, attributes, values
  - Fixed sized variables: array of fixed dimensions
  - Record variables: array with its most-significant dimension UNLIMITED
  - Coordinate variables: 1-D array with the same name as its dimension
- Dimensions
  - name, length
- Attributes - “metadata about the data”
  - name, type, values, length
  - Variable attributes
  - Global attributes

netCDF Dataset Objects

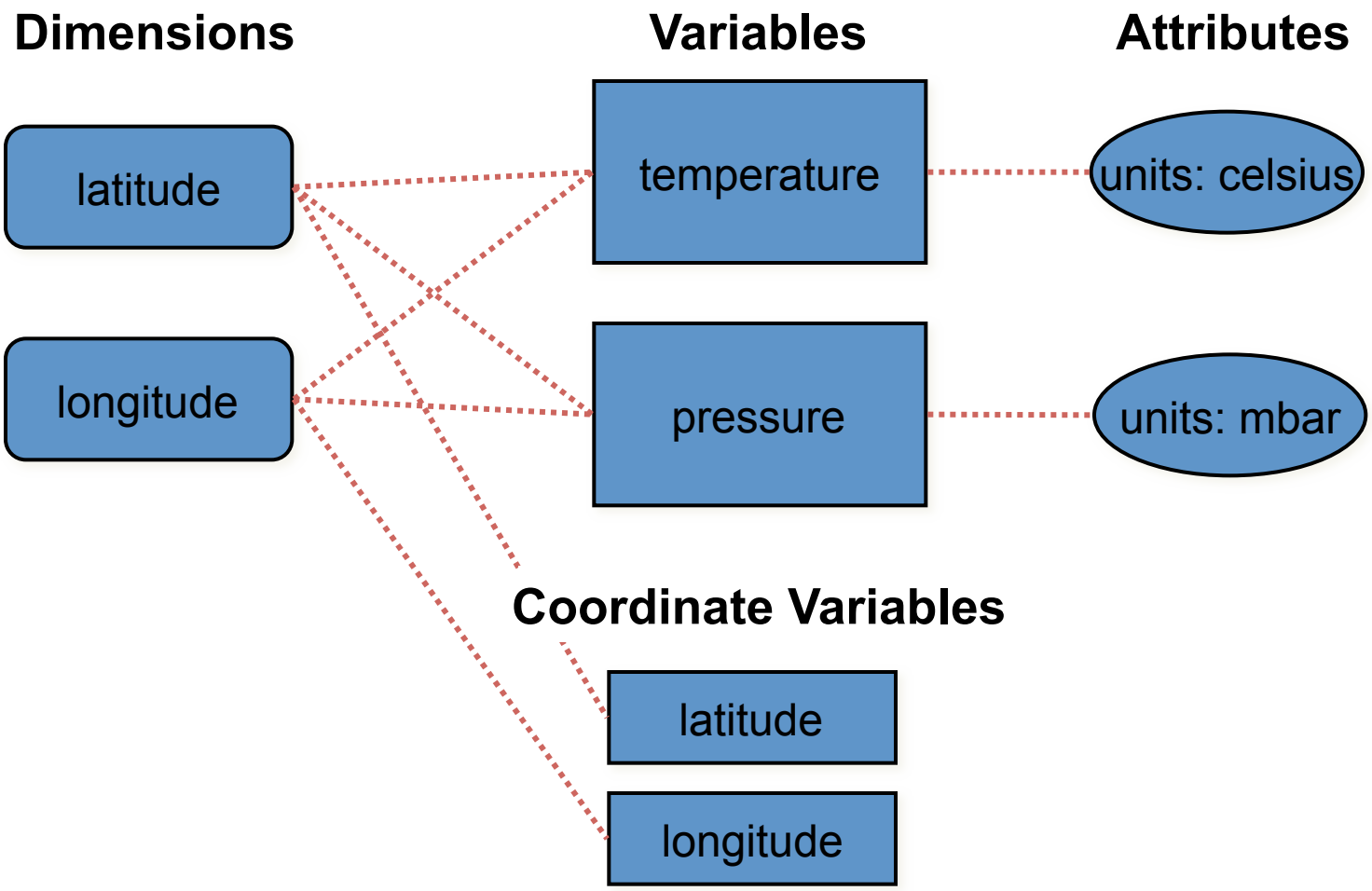
- Coordinate Variable – a 1D variable with the same name as a dimension, which stores values for each dimension value (for example, “latitude”)

- Unlimited Dimension – a dimension which has no maximum size
  - Data can always be extended along the unlimited dimension
  - Recommended when creating a new netCDF dataset that may be extended later

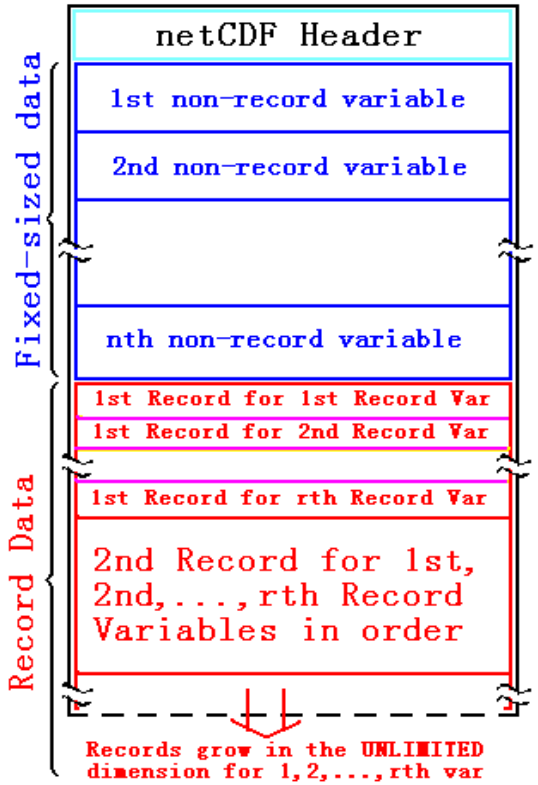
netCDF Example

- Suppose you want to store temperature and pressure values on a 2D latitude/longitude grid
- In addition to the data *(ie. Temperature and Pressure),* you want to store information about the exact values of the lat/lon grid
- You may also have additional data to store, for example, the units of the data values

netCDF Example



netCDF File Format - binary



- Header
  - –Number of record variables
  - –Dimension list
  - –Global attribute list
  - –Variable list
- Data (row-major)
  - –Fixed-sized data

    data for each variable is stored contiguously
  - –Record data

    a variable number of fixed-size records, each of which contains one record for each of the record variables in order.
- Both use extended XDR

Accessing netCDF Data

- Access is direct - efficient
- Can access elements of arrays
- Can stride through arrays to pick up portions of data

Important NetCDF Functions

- nc_create and nc_open: used to create and open files.
- nc_enddef, nc_close: used to end definition of and close files
- nc_def_dim, nc_def_var, nc_put_att_*: used to define dimensions, variables, and attributes
- nc_inq, nc_inq_var, nc_inq_dim, nc_get_att_* :used to learn about dims, vars, and atts.
- nc_put_vara_* , nc_get_vara_*: used to write and read data.

Attributes

- Attributes are 1-D arrays of any of the netCDF types
- Read/write them with functions like: nc_get_att_float and nc_put_att_int
- Attributes may be attached to a variable, or may be global to the file

Useful commands - ncdump

- Ncdump useful for quick glimpse of what kind of data file contains - queries and displays the file's metadata

```
> ncdump -h wrfinput_d01 | less
netcdf wrfinput_d01 {
dimensions:
        Time = UNLIMITED ; // (1 currently)
        DateStrLen = 19 ;
        west_east = 424 ;
        south_north = 299 ;
        west_east_stag = 425 ;
.....................................
float V(Time, bottom_top, south_north_stag, west_east) ;
                V:FieldType = 104 ;
                V:MemoryOrder = "XYZ" ;
                V:description = "y-wind component" ;
                V:units = "m s-1" ;
                V:stagger = "Y" ;
```

NetCDF Compilations

- Once you start using the netCDF API, you will need to start linking your application against the library
- Also need to include the appropriate header file in your application (or you can access a F90 module if you have built a Fortran interface):

```
#include "netcdf.h"
```

- Example Intel compile (after "module load netcdf" on TACC system):

```
> icc -I$(TACC_NETCDF_INC)
 -L$(TACC_NETCDF_LIB) example.c -lnetcdf
```

Useful commands - ncdump

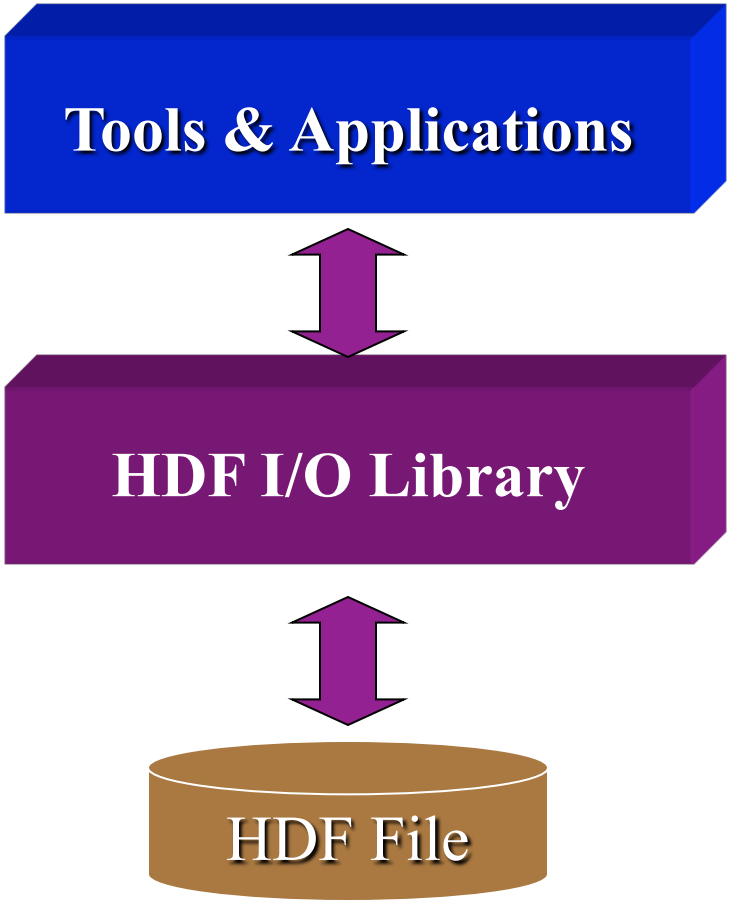
- In addition to displaying metadata, ncdump can show all data values, or values for a requested variable:

```
>ncdump -v V wrfinput_d01 | less

data:

 V =
  -5.943535, -5.954164, -5.959685, -5.960224, -5.957259, -5.952528,
   -5.947741, -5.944627, -5.944635, -5.947225, -5.949976, -5.950274,
   -5.945539, -5.933652, -5.914801, -5.890519, -5.861694, -5.829077,
   -5.793406, -5.755308, -5.714824, -5.671347, -5.624143, -5.57258,
   -5.516121, -5.452713, -5.380764, -5.30263, -5.222619, -5.145173,
   -5.074714, -5.012296, -4.95566, -4.903169, -4.853917, -4.806993,
   -4.761831, -4.719834, -4.681822, -4.64602, -4.610251, -4.57234,
   -4.530297, -4.483756, -4.434185, -4.383417, -4.333199, -4.285205,
```

HDF Library

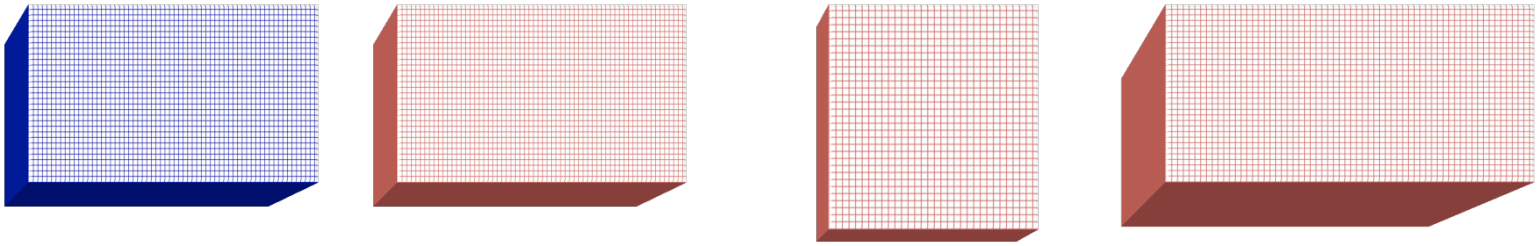
What is HDF?

- HDF = Hierarchical Data Format
  - format and software for scientific data (developed to aid scientists and programmers in the storing, transfer and distribution of data sets)
  - stores images, multidimensional arrays, tables, etc. (*you can mix and match all of these structures in HDF5)*
  - emphasis on storage and I/O efficiency
  - free and commercial software support
  - emphasis on standards
  - users from many engineering and scientific fields



An HDF File:
A Collection of Scientific Data Objects

Four separate 3-D arrays



Typically include metadata to describe the objects

HDF History

- HDF4 - Based on original 1988 version of HDF
  - backwards compatible with all earlier versions
  - 6 basic objects: *raster image*, *multidimensional array (SDS), palette, group (Vgroup), table (Vdata), annotation*
  - limits on file size (< 2GB) and number of objects (<20K)
- HDF5 - First released in 1998
  - new format(s) and library are ***not compatible*** with HDF4
  - includes only 2 basic primitive objects (groups and datasets)
    - No limit on the HDF5 file size and number of objects in the file
    - HDF5 file is portable across all computing platforms

HDF Supported Languages

- C
- Wrappers:
  - C++
  - Fortran90
  - Java
  - Python (and others)
- Vendors' compilers (Oracle, Cray, IBM, HP, etc.)
- PGI, Intel, Pathscale and Absoft (Fortran)
- GNU C

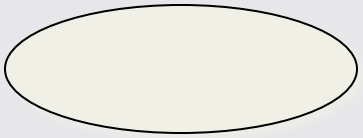
HDF5 File Objects (conceptual view)

- Containers for storing scientific data
  - Primary Objects:
    - Groups
    - Datasets
  - Secondary Objects:
    - Datatypes
    - Dataspaces
- Additional means to organize data
    - Attributes
    - Sharable objects
    - Storage and access properties

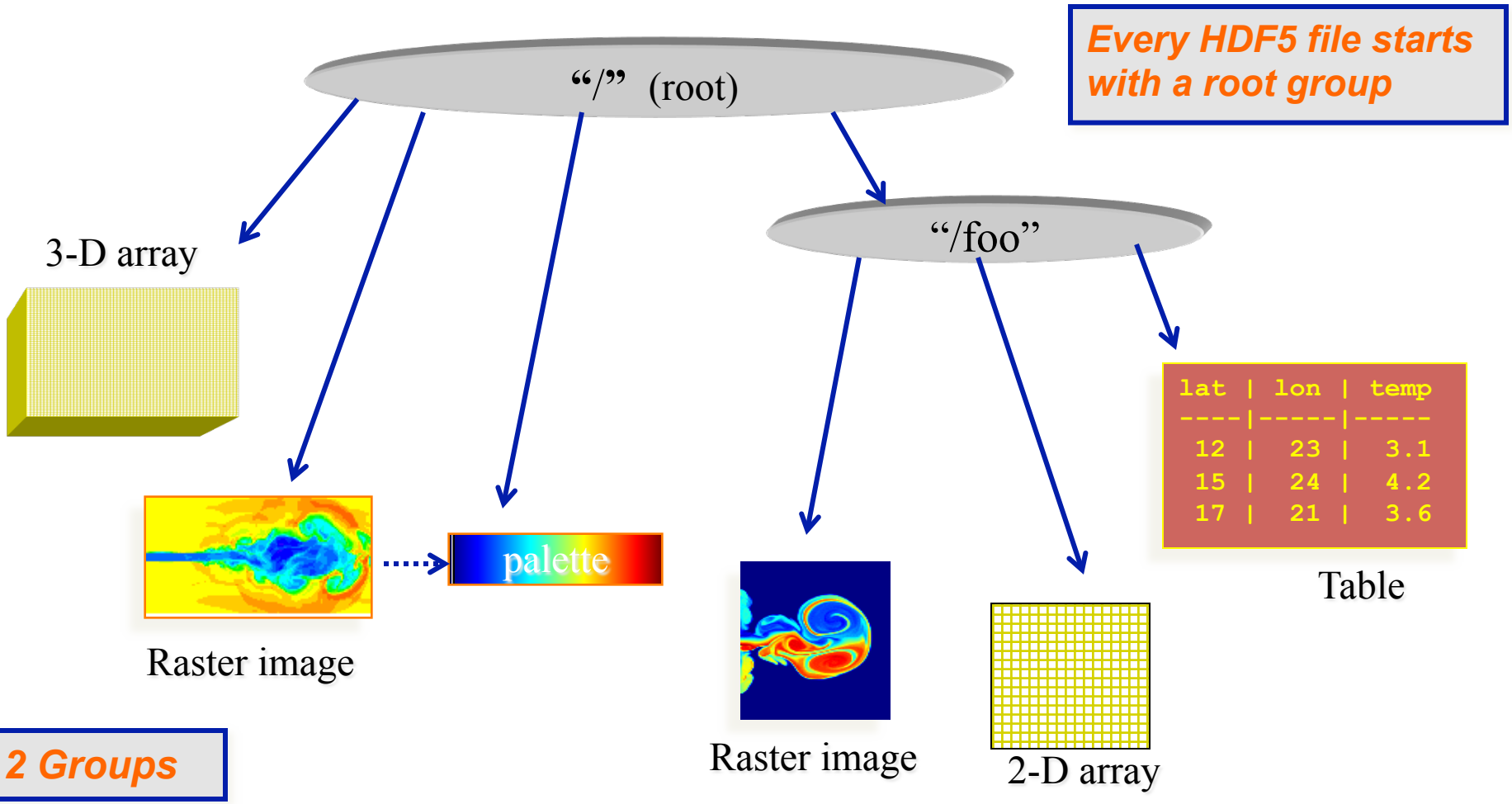
HDF5 Data Model

- Dataset
  - multidimensional array of elements, together with supporting metadata

- Group
  - directory-like structure containing datasets, groups, other objects



Example HDF5 File Contents



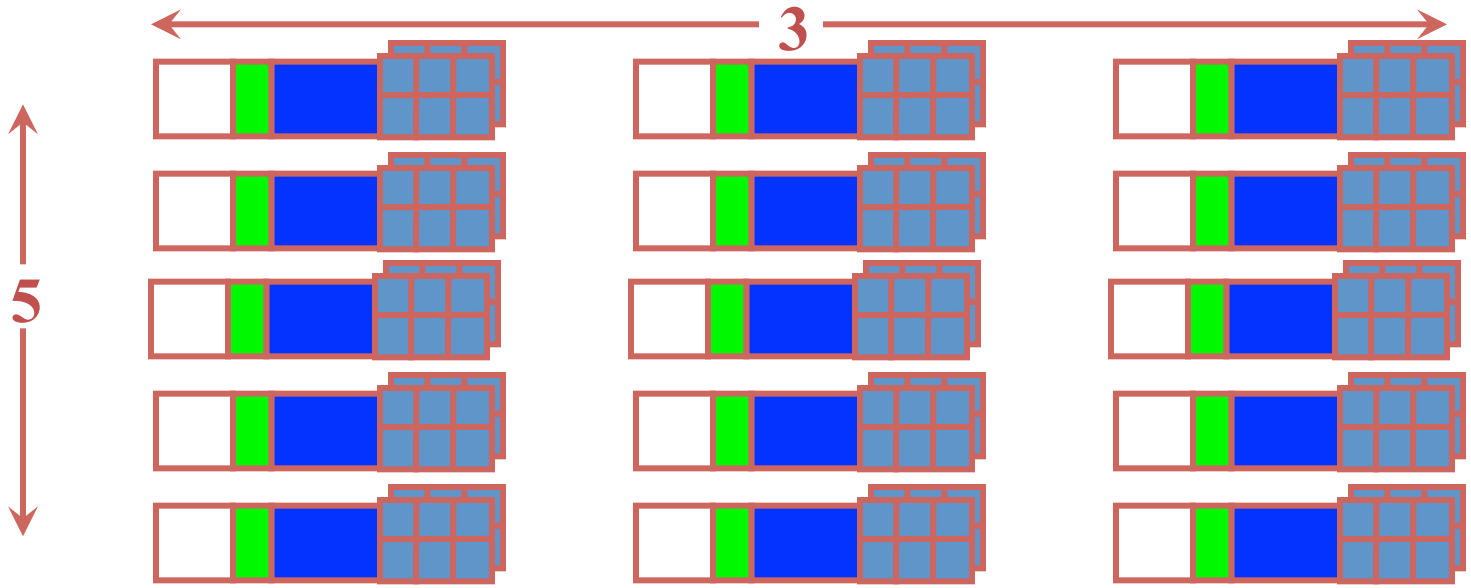
Components of an HDF Dataset

- Array
  - an ordered collection of identically typed data items distinguished by their indices (subscripts)
- Dataspace
  - information about the size and shape of a dataset array and selected parts of the array
- User-defined attribute list
- Special storage options
  - extendable, chunked, compressed, external

HDF5 Datatypes

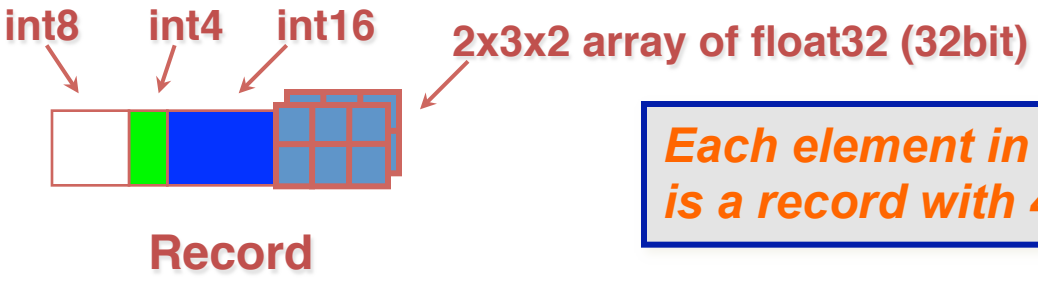
- Atomic types
  - standard integer & float
  - user-definable scalars (e.g. 13-bit integer)
  - variable length types (e.g. strings)
  - pointers - references to objects/dataset regions
  - enumeration - names mapped to integers
- Compound types
  - Comparable to C structs
  - Members can be atomic or compound types
  - Members can be multidimensional

HDF5 dataset: Example Array of Records



Dimensionality: 5 x 3

Datatype:



Each element in 2D array is a record with 4 values

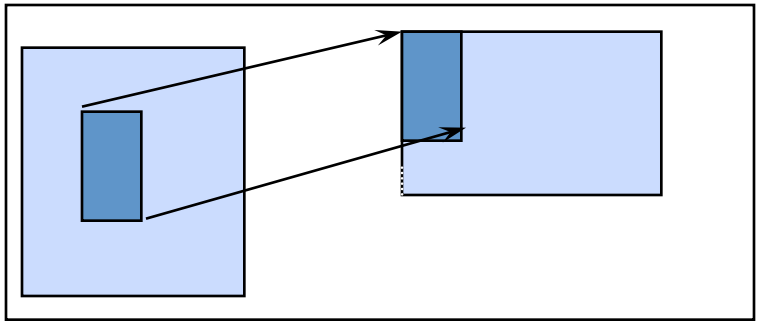
HDF Data Spaces

- A dataset *dataspace* describes the dimensionality of the dataset. Dimensions of a dataset can be:
  - *fixed* (unchanging), or
  - *unlimited*, which means that they are extendible (i.e. they can grow larger)
- Properties of a *dataspace* consist of:
  - the *rank* (number of dimensions) of the data array,
  - the *actual sizes of the dimensions* of the array
  - the *maximum sizes of the dimensions* of the array
    - For a fixed-dimension dataset, the actual size is the same as the maximum size of a dimension.
    - When a dimension is unlimited, the maximum size is set to the value H5P_UNLIMITED

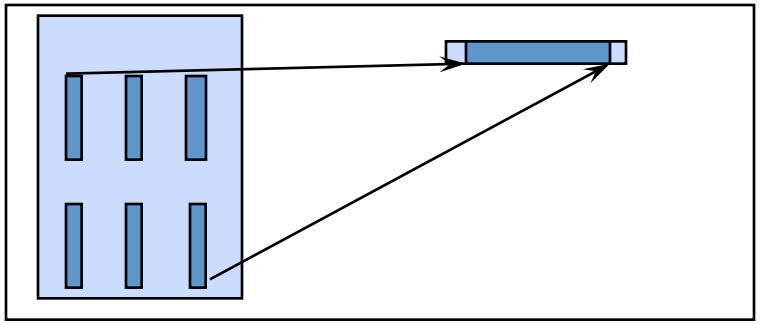
HDF Data Spaces

- A *dataspace* can also describe portions of a dataset, making it possible to do partial I/O operations on *selections*
- *Selection* is supported by the dataspace interface: given an n-dimensional dataset, there are currently four ways to do partial selection:
  - Select a logically contiguous n-dimensional hyperslab
  - Select a non-contiguous hyperslab consisting of elements or blocks of elements (hyperslabs) that are equally spaced
  - Select a union of hyperslabs
  - Select a list of independent points
- Note: to perform partial read/write operations on the data, you must provided: file dataspace, file dataspace selection, memory dataspace and memory dataspace selection *(ie. a mapping between the file layout and desired memory layout)*

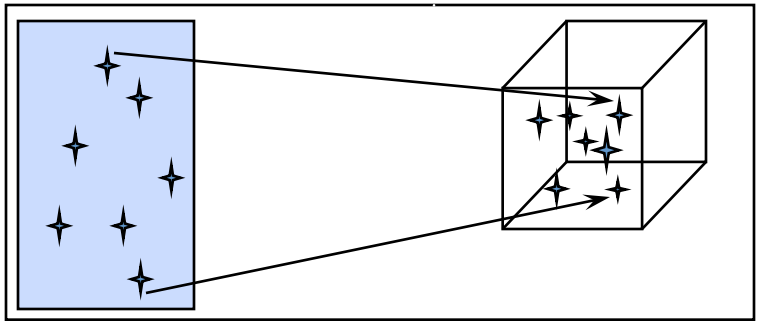
Example Mappings



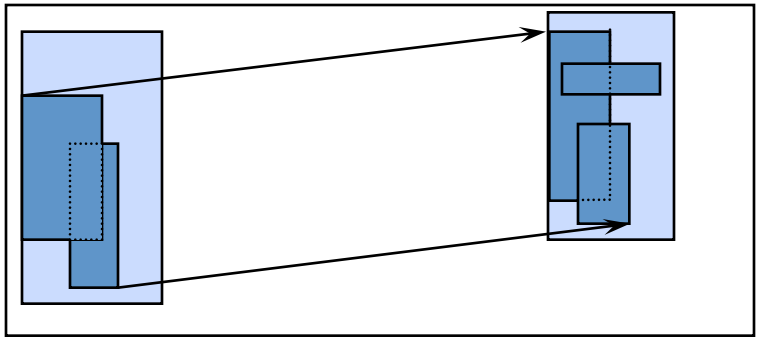
(a) A hyperslab from a 2D array to the corner of a smaller 2D array



(b) A 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. Number of elements must be equal.

HDF Attributes

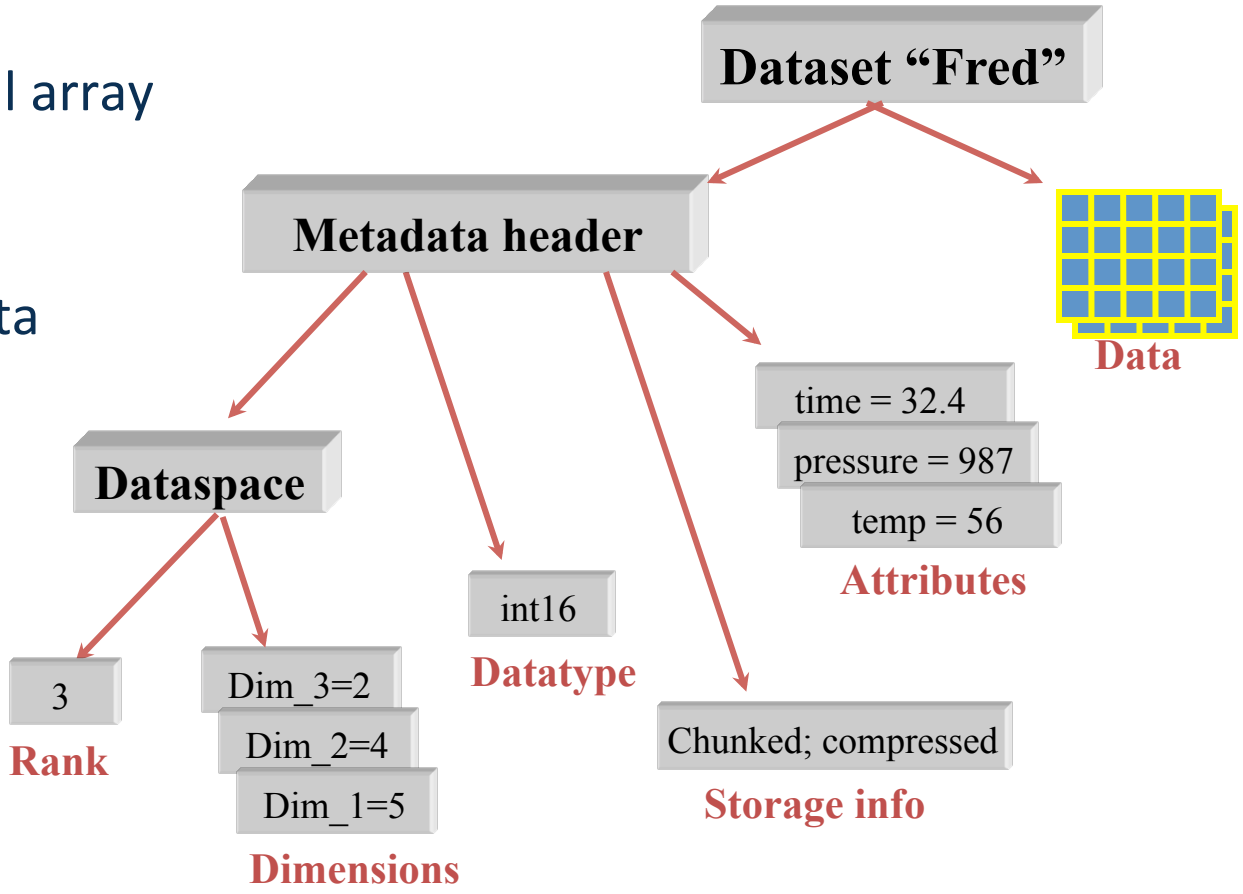
- Are small pieces of data
- Attached to datasets or groups
- Operations are scaled-down versions of the dataset operations
  - Not extendible
  - No compression
  - No partial I/O

Storage Options

- The HDF5 format makes it possible to store data in a variety of ways
- The default storage layout format is *contiguous*, meaning that data is stored in the same linear way that it is organized in memory
- Two other storage layout formats are currently defined for HDF5:
  - ***compact*** - used when the amount of data is small and can be stored directly in the object header
  - ***chunked*** - involves dividing the dataset into equal-sized "chunks" that are stored separately
    - achieves good performance when accessing subsets of the datasets, even when the subset to be chosen is orthogonal to the normal storage order of the dataset.
    - makes it possible to compress large datasets and still achieve good performance when accessing subsets of the dataset
    - makes it possible efficiently to extend the dimensions of a dataset in any direction.

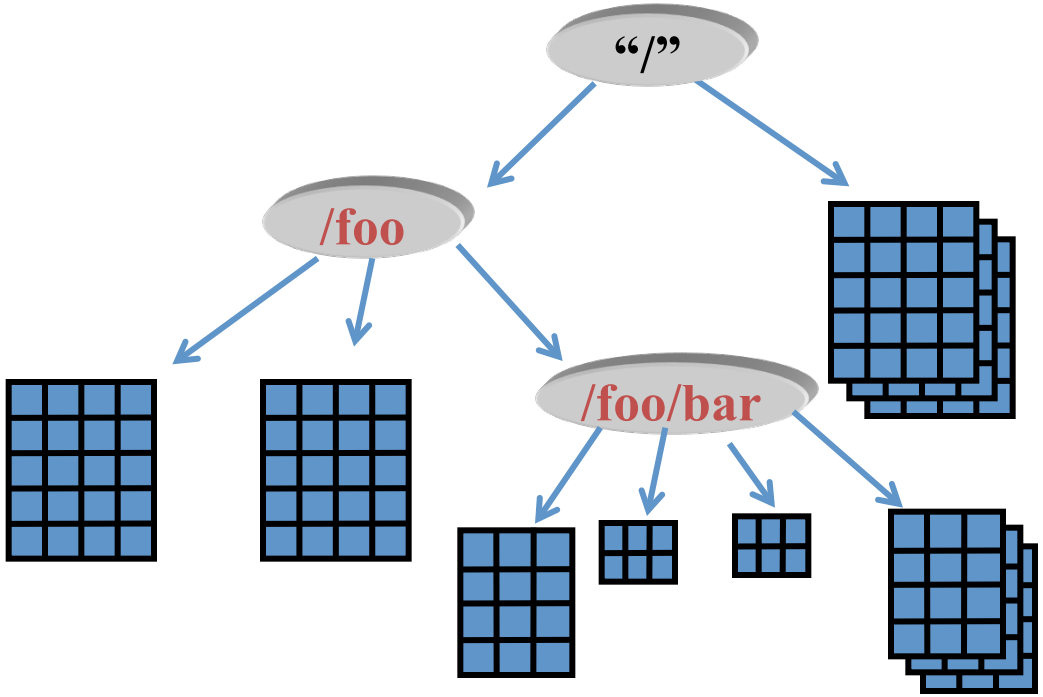
Dataset Components – putting it all together

- a multidimensional array of data elements
- header with all necessary metadata
  - datatype
  - dataspace
  - attributes
  - storage info



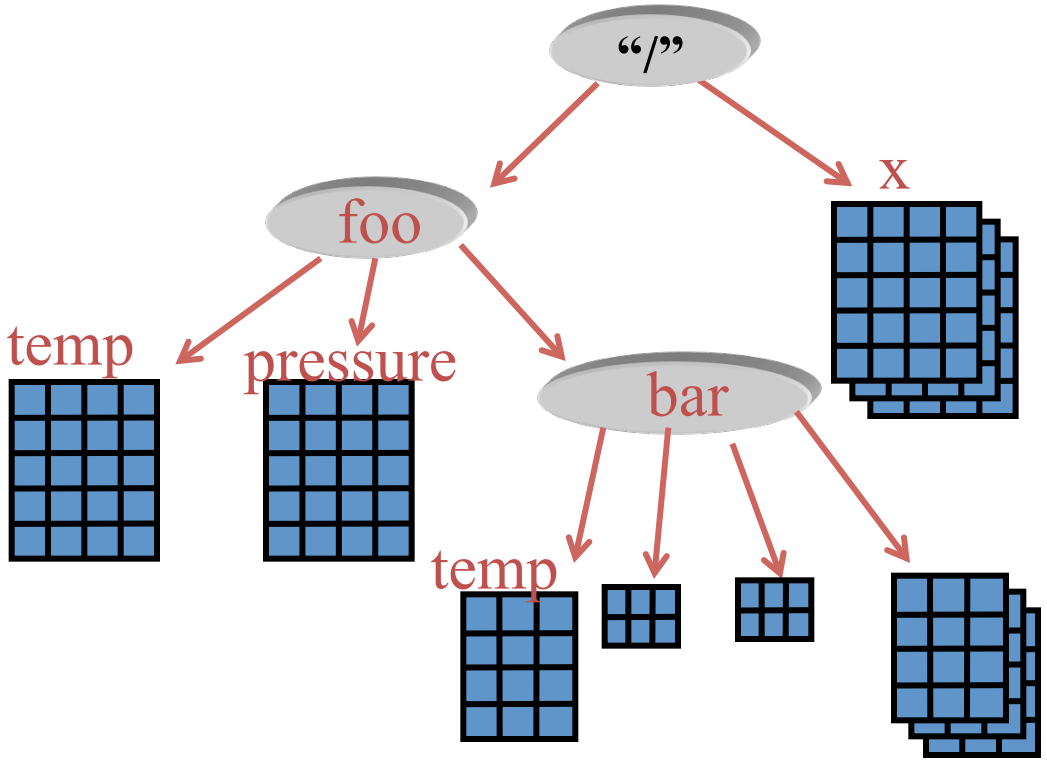
Groups

- A mechanism for collections of related objects
- Every file starts with a root group
- Similar to UNIX directories
- Can have attributes

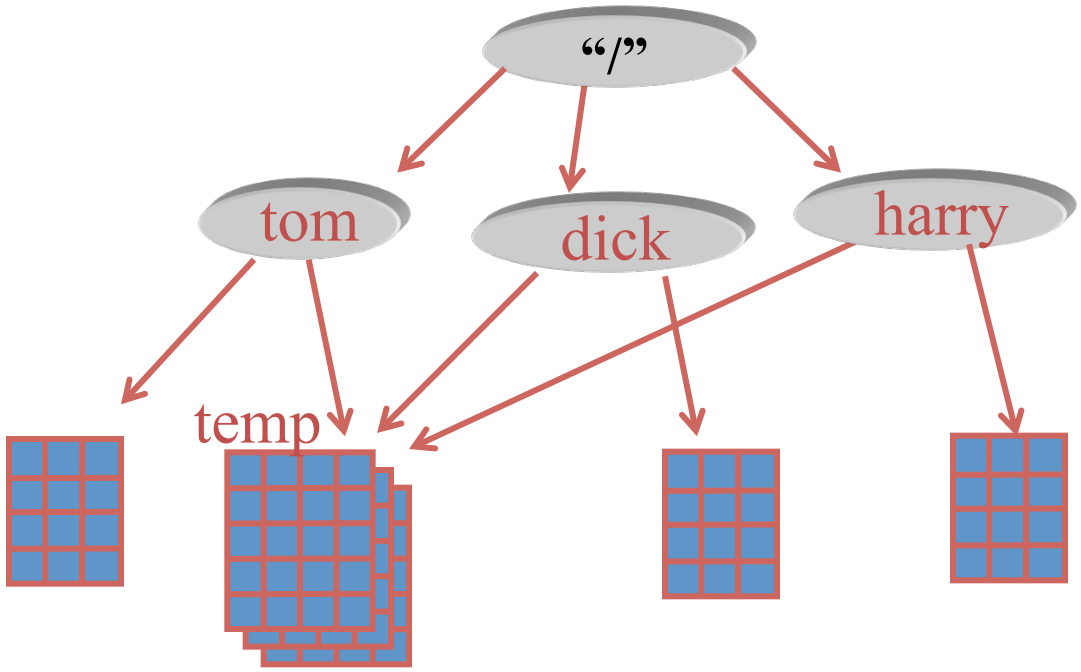


HDF5 objects are identified and located by their *pathnames*

/ (root)
/x
/foo
/foo/temp
/foo/pressure
/foo/bar/temp



HDF Members Can be Shared



Note: Three pathnames identify the same object

/tom/temp
/dick/temp
/harry/temp

The General Programming Paradigm

1. Objects are first opened or created
2. objects are accessed (read/write)
3. Objects are closed

```
CALL h5fopen_f (“myfile”, H5F_ACC_RDWR_F, file_id, err)
CALL h5dopen_f (file_id, “velocity”, dset_id, err)
CALL h5dread_f (dset_id, H5T_NATIVE_INTEGER, data, err)
CALL h5dclose_f (dset_id, error)
CALL h5fclose_f (file_id, error)
```

Anything that can be set from the API can also be queried

Atomic Data Types

- The library has predefined native atomic types:

H5T_NATIVE_CHAR	H5T_NATIVE_INT8
H5T_NATIVE_USHORT	H5T_NATIVE_UINT16
H5T_NATIVE_INT	H5T_NATIVE_INT32
H5T_NATIVE_LONG	H5T_NATIVE_LLONG
H5T_NATIVE_FLOAT	H5T_NATIVE_FLOAT32
H5T_NATIVE_DOUBLE	H5T_NATIVE_FLOAT64
H5T_NATIVE_STRING	H5T_NATIVE_TIME
H5T_NATIVE_DATE	H5T_NATIVE_BITFIELD
H5T_NATIVE_OPAQUE	H5T_NATIVE_INT64

- Some non-native types will be added for common architectures.
- New types can be derived from existing types

Useful HDF5 Binaries

- These binaries are generally built during the normal HDF installation
  - **h5ls** - lists contents of HDF5 file
  - **h5dump** - higher level view of the file
  - **h5diff** - show difference between two HDF files

```
lonestar4--> h5dump SDS.h5
HDF5 "SDS.h5" {
GROUP "/" {
 DATASET "IntArray" {
    DATATYPE  H5T_STD_I32LE
    DATASPACE  SIMPLE { ( 5, 6 ) / ( 5, 6 ) }
    DATA {
    (0,0): 0, 1, 2, 3, 4, 5,
    (1,0): 1, 2, 3, 4, 5, 6,
    (2,0): 2, 3, 4, 5, 6, 7,
    (3,0): 3, 4, 5, 6, 7, 8,
    (4,0): 4, 5, 6, 7, 8, 9
    }
 }
}
}
```

HDF Compilations

- Once you start using the HDF API, you will need to start linking your application against the library
- Also need to include the appropriate header file in your application:

  #include "hdf5.h"

- See “module help hdf5” for example on Stampede:

```
To use the HDF5 library, compile the source code with the option:

 -I$TACC_HDF5_INC

and add the following options to the link step:

 -Wl,-rpath,$TACC_HDF5_LIB -L$TACC_HDF5_LIB -lhdf5 -lz
```

References

- NetCDF Software: http://www.unidata.ucar.edu/software/netcdf/
- NetCDF Documentation: http://www.unidata.ucar.edu/software/netcdf/docs/
- NetCDF Version 4: http://www.unidata.ucar.edu/software/netcdf/netcdf-4/
- HDF Group: http://www.hdfgroup.org/HDF5/
- SILO: https://silo.llnl.gov
- VisIt: https://visit.llnl.gov