Sharable and Scalable I/O Solutions for High Performance Computing Applications

SC01

Nov 11, 2001

Presenters

- **Larry Schoof (laschoo@sandia.gov)**
  - **Sandia National Laboratories (SNL)**
- **Mike Folk (mfolk@ncsa.uiuc.edu)**
  - **National Center for Supercomputing Applications (NCSA)**
- **Albert Cheng (acheng@ncsa.uiuc.edu)**
  - **NCSA**
- **Elena Pourmal (epourmal@ncsa.uiuc.edu)**
  - **NCSA**
- **Mark Miller (miller86@llnl.gov)**
  - **Lawrence Livermore National Laboratory (LLNL)**

Outline

8:30 - 9:00	**History of I/O libraries (Schoof)**
	HDF5 (NCSA)
9:00 - 9:20	Data model and file structure (Folk)
9:20 - 10:00	API (Pourmal)
10:00 - 10:30	**Break**
10:30 - 11:15	Parallel programming model (Cheng)
11:15 - 11:45	Application tuning (Cheng)
11:45 - 12:00	Access methods (Cheng)
12:00 - 13:30	**Lunch**
	Sets And Fields (SAF)
13:30 - 14:30	Math-based data model (Miller)
14:30 - 15:00	SAF implementation into simulation application (Miller)
15:00 - 15:30	**Break**
	Performance Results
15:30 - 16:15	HDF5 and MPI-IO(Cheng)
16:15 - 17:00	SAF (Schoof)

History of I/O Libraries
An Application Perspective

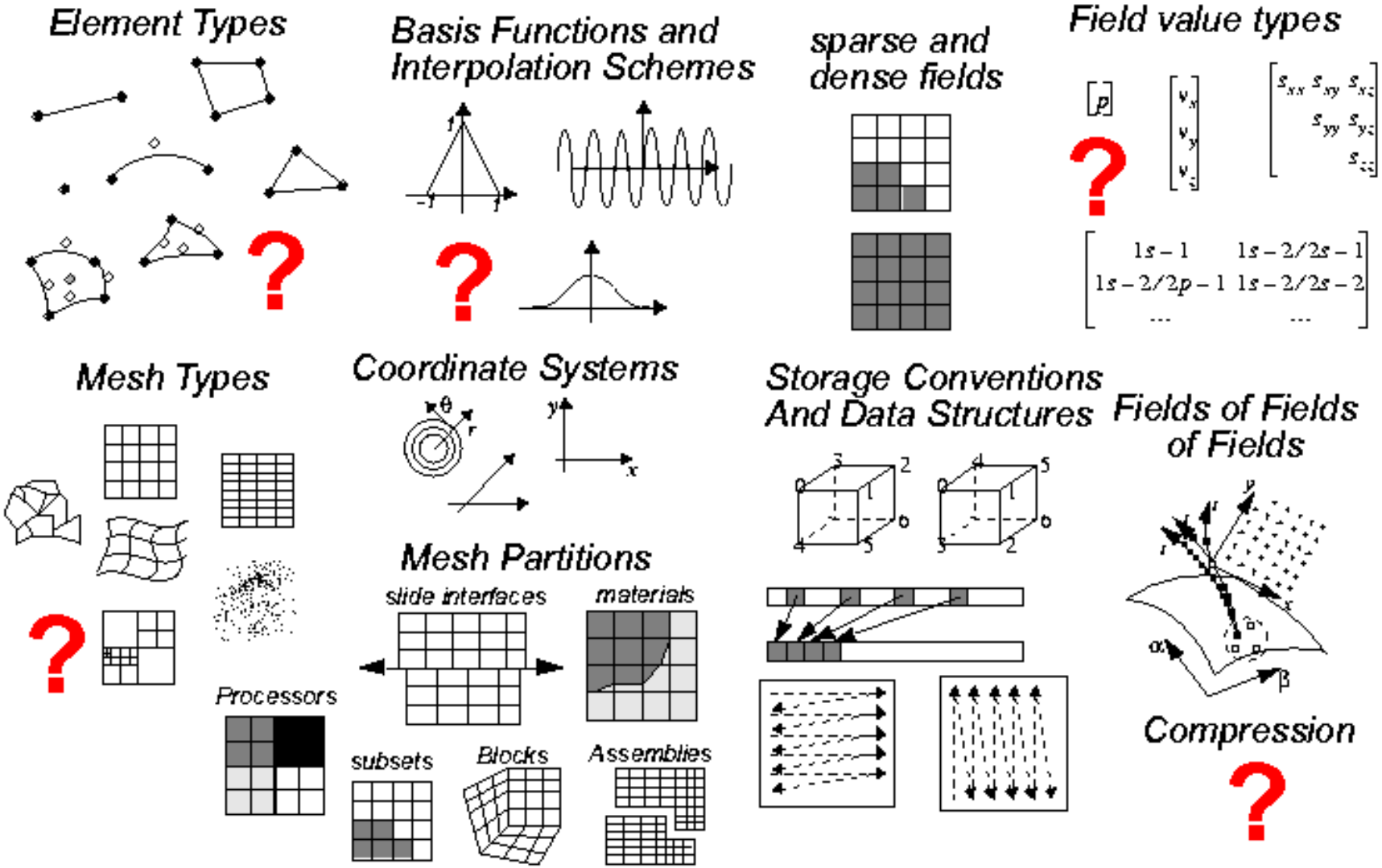
Larry Schoof

Sandia National Laboratories

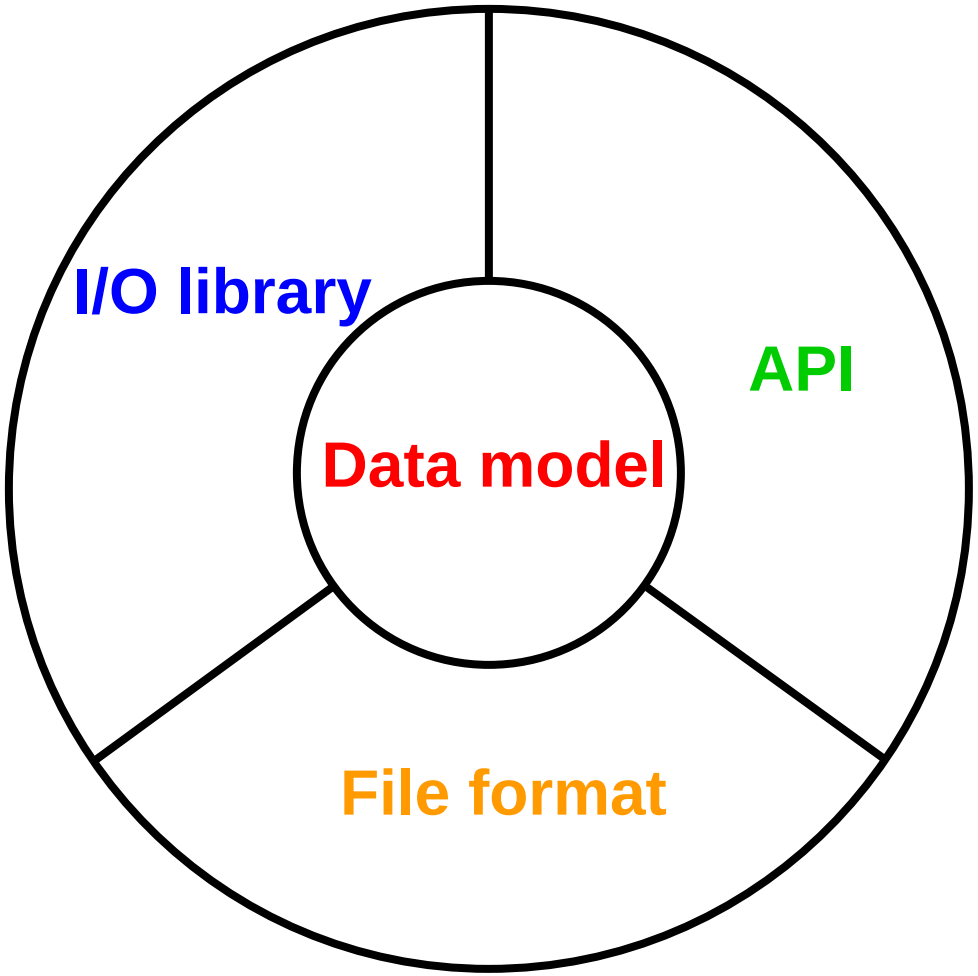
Goals for Interoperable HPC

- **Share data**
  - Application interoperability across computational physics community
  - Requires sound data model with robust data abstractions to represent diverse data
- **Share software**
  - Common data format allows development of common tools
- **Handle tera-scale data**
  - Tbyte (10^{12} bytes) - Pbyte (10^{15} bytes) data sets
  - Parallel I/O services and parallel data constructs
  - Don't copy or move data
  - Minimize data translation

The Problem

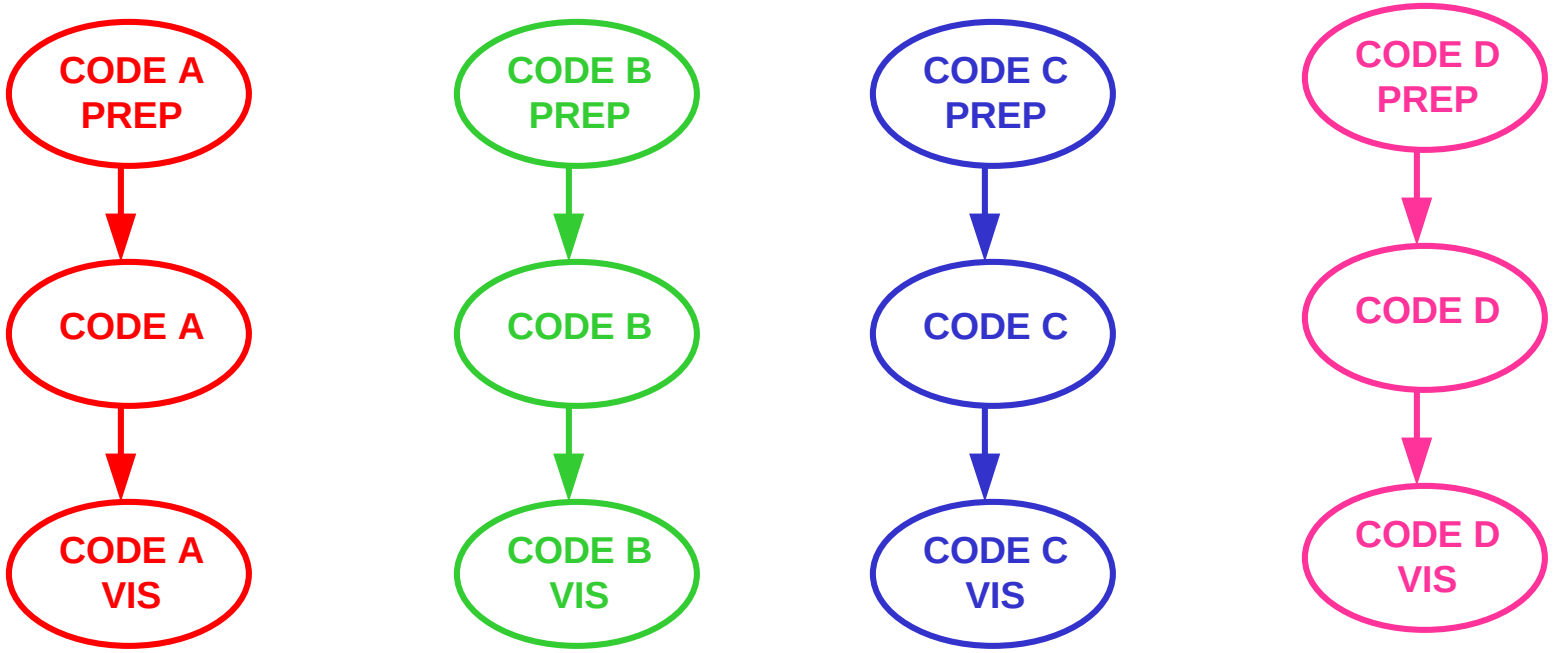

Element Types
Basis Functions and Interpolation Schemes
sparse and dense fields
Field value types
Mesh Types
Coordinate Systems
Storage Conventions And Data Structures
Fields of Fields of Fields
Mesh Partitions
slide interfaces
materials
Processors
subsets
Blocks
Assemblies
Compression

The Answer!



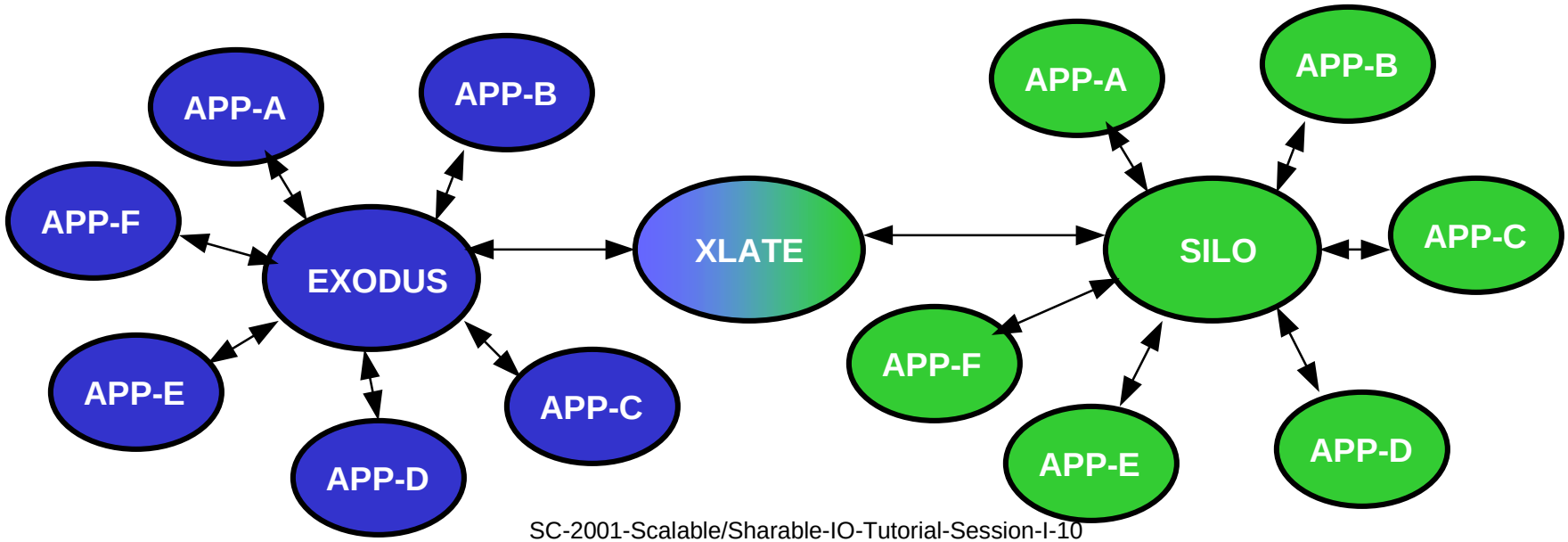
First Generation Scientific Databases (pre 1980)

- **Software fiefdoms**; “stovepipes”
- Application-code centric
- Labor intensive; **duplication of effort**
- A lot of these environments still exist !!!



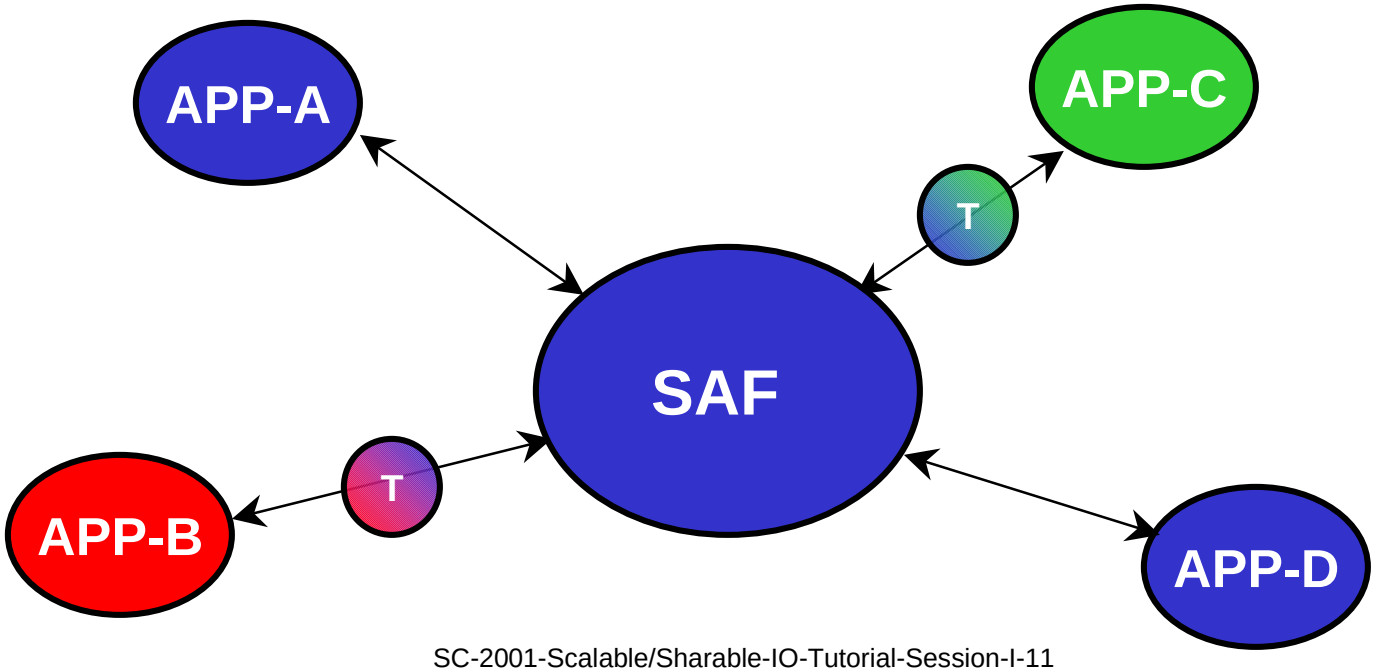
Second Generation Scientific Databases (1980 - present)

- **"Small-scale integration"**
- **Data centric, but constrained data domain (e.g., FE apps)**
- **Set of applications with which data is sharable is small**
- **May use low-level I/O layers (e.g., netCDF, PDB, HDF, etc.)**
- **Limited to small menu of supported data objects**
- **Difficult to add new objects**
- **Translators lose and duplicate data**



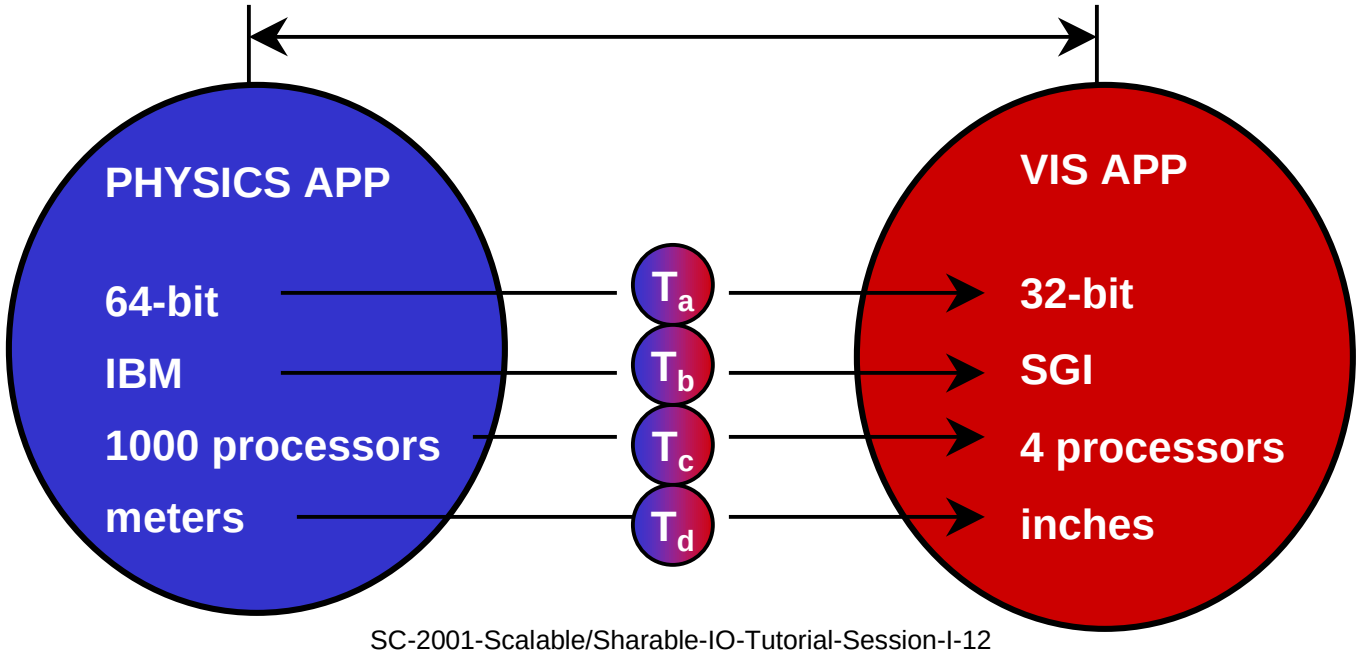
Third Generation Scientific Databases

- **"Large-scale integration"**
- **Data model based on mathematics**
- **Fully self-describing data; higher level than netCDF, HDF, etc. descriptions**
- **Operators to optionally transform data**



Transform Operators

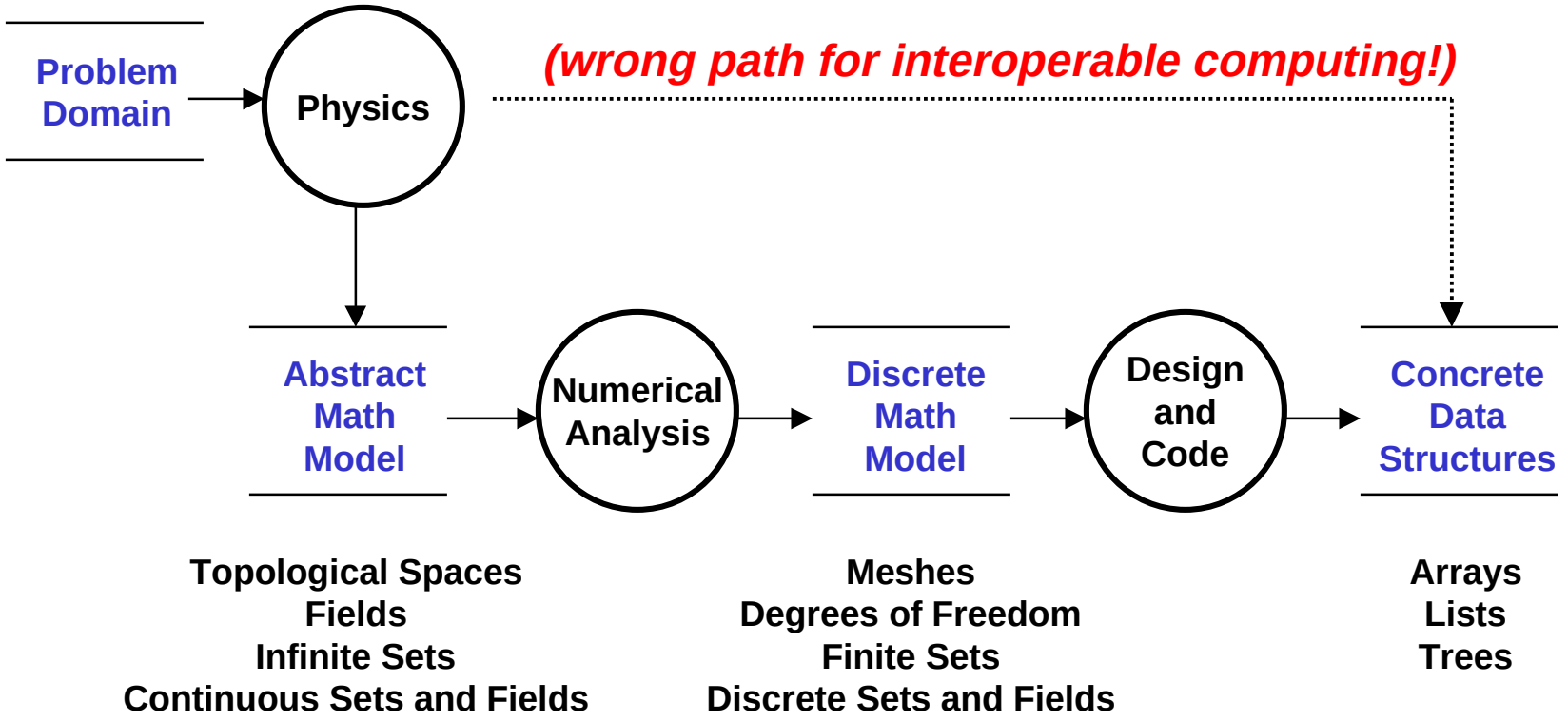
- **Different representations**
  - cpu architecture (SGI / Cray)
  - precision (32 / 64 bit)
  - indexing origin (0 / 1)
  - interleave (vector / component)
  - array order (row / column major)
  - coordinate system (spherical / cartesian)
  - mesh resolution
  - processor decomposition
  - element types (quad / tri)
  - node ordering schemes
  - mesh topology (structured / unstructured)
  - units
  - field interpolations (bicubic / bilinear)
  - etc.



What is a Data Model?

- "a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints"
- Mathematical abstractions that fully characterize an application domain and the data produced in that domain
- Data model specifies
  - mathematical objects
  - operations on the objects
- Data storage software based on the model implements
  - the objects and operations

Numerical Software Design



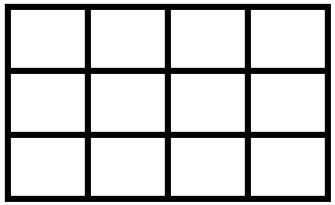
- Useful semantics defined by abstract math model
- Interface usually defined by concrete data structures
- Sharing data at the concrete level is difficult

Low-context Data Models

- netCDF (UCAR)
  - multi-dimensional arrays of ints, floats, etc
- HDF4 (NCSA)
  - Scientific Data Set: multi-dimensional arrays of ints, floats, etc
  - Vdata: collection of records of fixed-length fields (tables)
  - Vgroup: collection of related objects
  - raster: 2-dimensional raster image
  - palette: color-lookup table with 256 entries
  - annotation: text attached to objects
- HDF5 (NCSA)
  - multi-dimensional array of record structures
  - grouping structure

High-Context Data Model

MESH



SETS OF:

0-CELLS (POINTS)

1-CELLS (EDGES)

2-CELLS (FACES)

3-CELLS (VOLUMES)

VARIABLE SPACE

ALBEBRAIC TYPE:

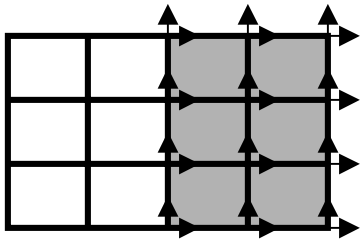
SCALAR,

VECTOR,

TENSOR,

. . .

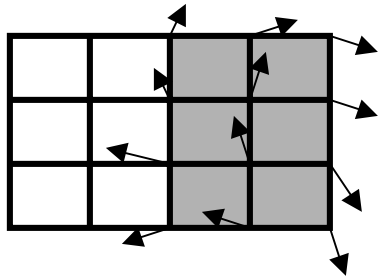
FIELD SPACE



FIELD TEMPLATE:

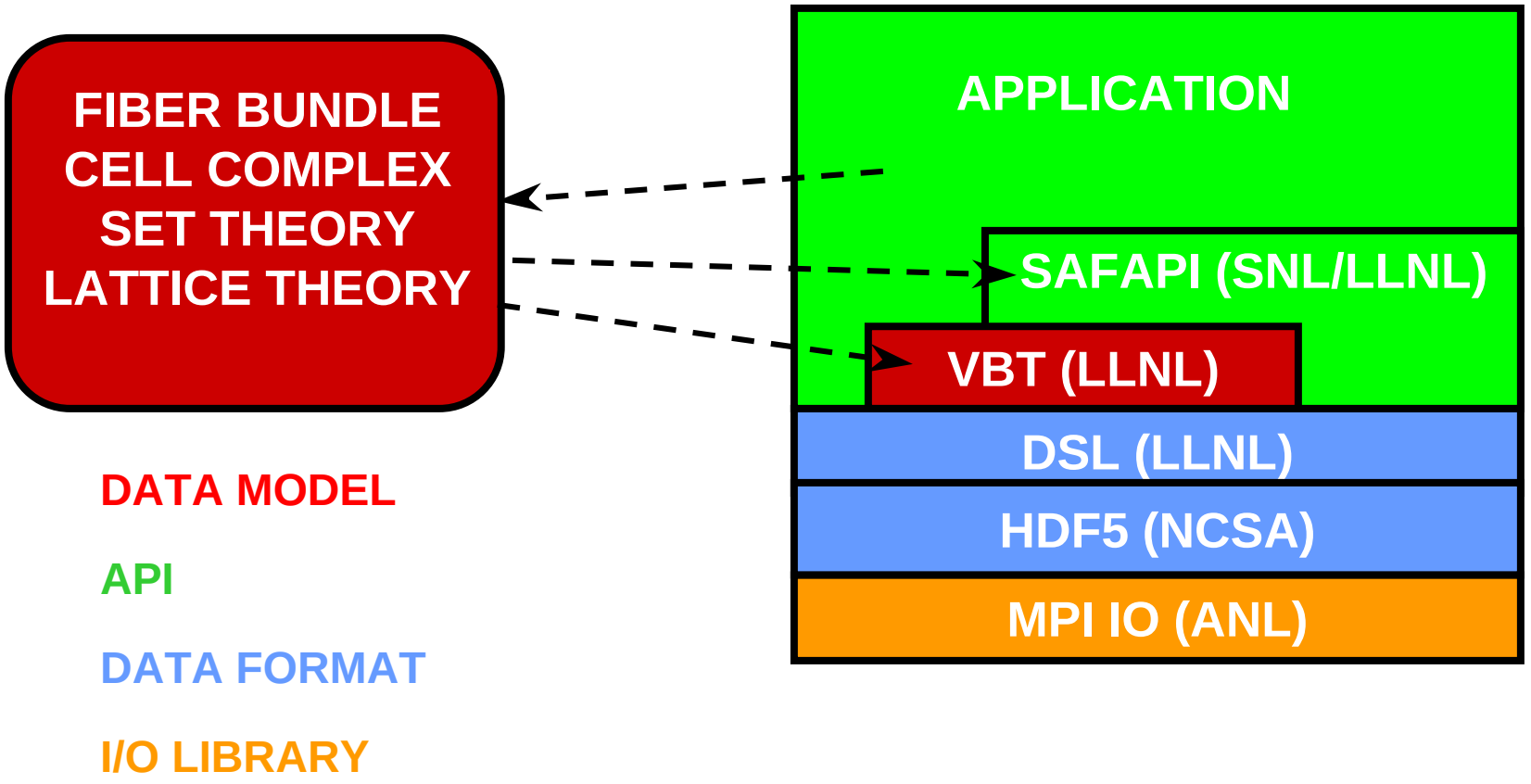
ALGEBRAIC TYPE OVER (SUBSET OF) MESH

FIELD VALUES



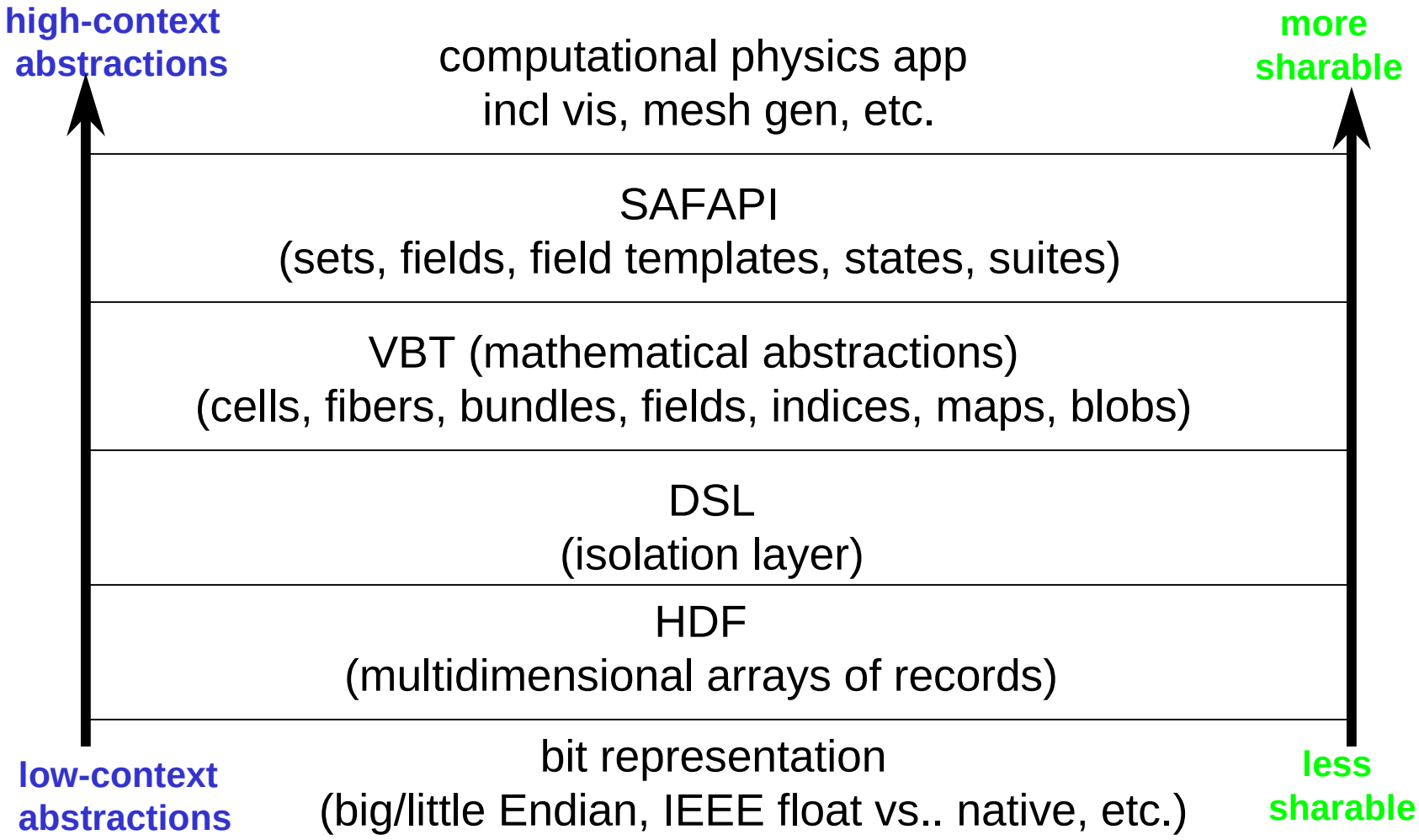
FIELD

SAF (Sets And Fields) Implementation



Data Sharability Spectrum

Data can be shared at the highest layer of common semantics.



Background Summary

- **Development of I/O software systems has been evolutionary**
- **High-context data model is critical for interoperability**
- **Complete solution must include high-performance low-context I/O layers**

HDF5

Mike Folk, Elena Pourmal, and Albert Cheng

NCSA

HDF5

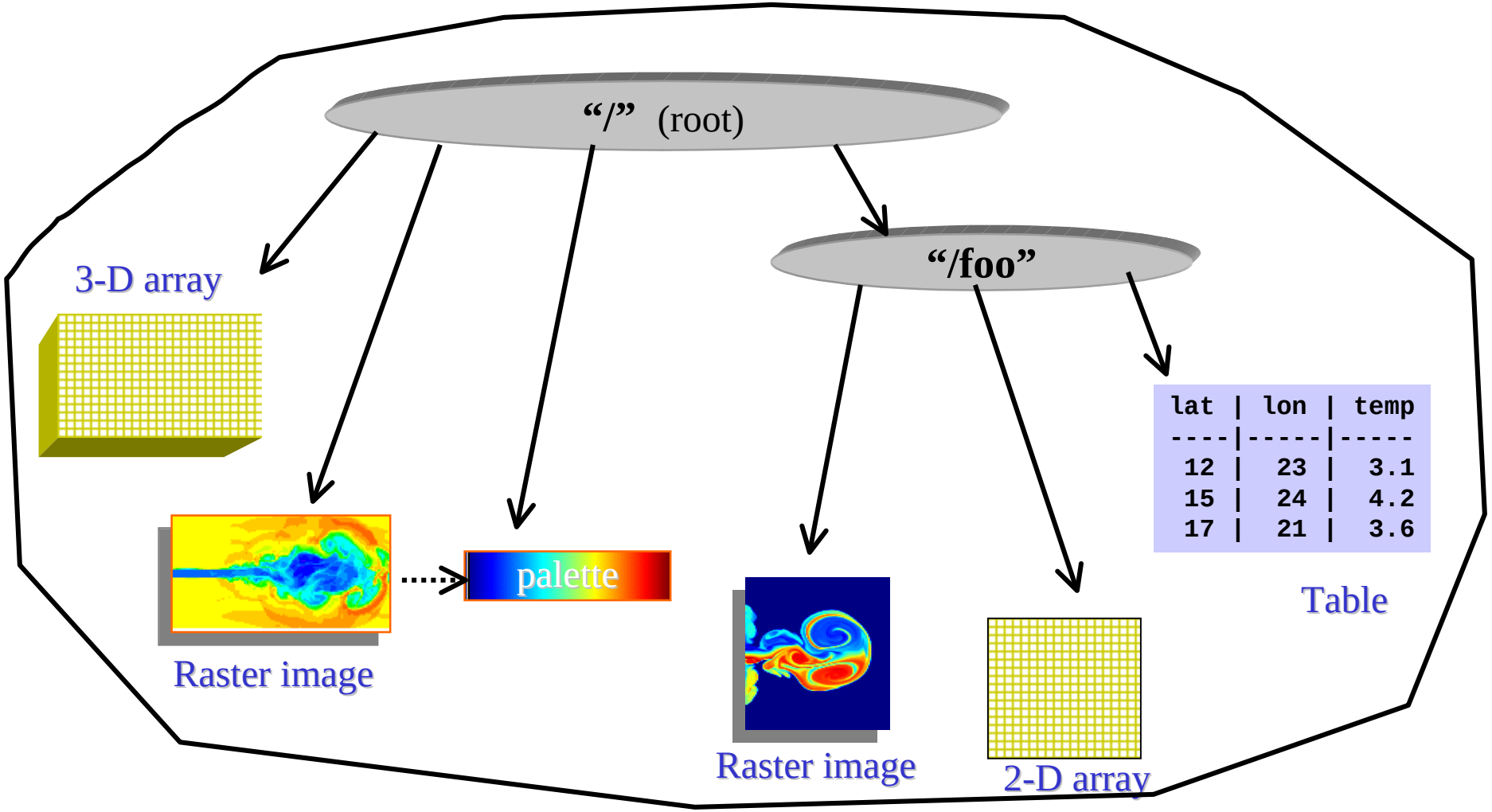
- Overview of HDF5 data model and file structure
- Application programming interface (API) for creating and accessing HDF5 objects and files. Programming model for sequential access.
- Parallel programming model for HDF5, including the use of MPI-IO
- Tuning for different applications and file systems
- Other forms of access, such as access across a computational grid

Overview of HDF5
Data Model and File Structure

What is HDF?

- Format and software for scientific data
- Stores images, multidimensional arrays, tables, etc.
- Emphasis on storage and I/O efficiency
- Free and commercial software support
- Emphasis on standards
- Users from many engineering and scientific fields

Example HDF5 file



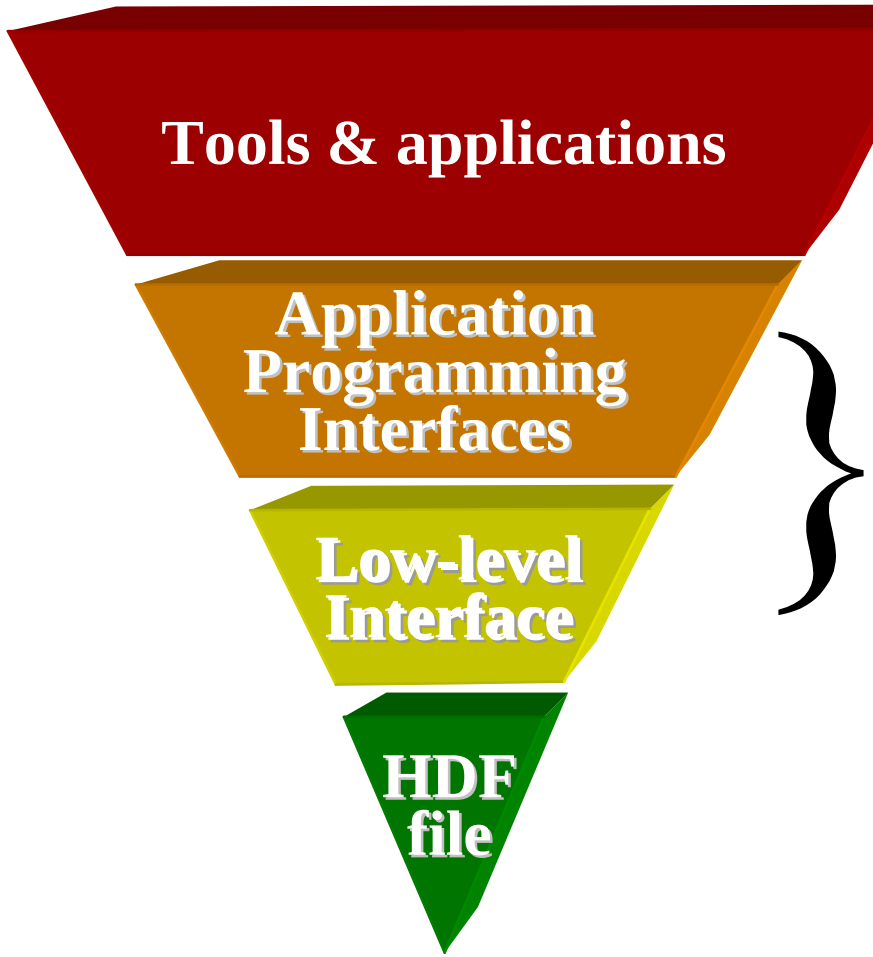
HDF4 vs. HDF5

- **HDF4 - Based on original 1988 version of HDF**
  - **Backwardly compatible with all earlier versions**
  - **6 basic objects**
    - **raster image, multidimensional array (SDS), palette, group (Vgroup), table (Vdata), annotation**
- **HDF5 - First released in 1998**
  - **New format & library – not compatible with HDF4**
  - **2 basic objects**
    - **Multidimensional array**
    - **Group**

New HDF5 Features

- **More scalable**
  - **Larger arrays and files**
  - **More objects**
- **Improved data model**
  - **New datatypes**
  - **Single comprehensive dataset object**
- **Improved software**
  - **More flexible, robust library**
  - **More flexible API**
  - **More I/O options**

HDF Software



Utilities and applications for managing, manipulating, viewing, & analyzing data.

HDF I/O library

– High-level, object-specific APIs.

– Low-level API for I/O to files, etc.

File or other data source

HDF5 Tools

- Java HDF5 Viewer/Editor
- hdf5ls - lists contents of HDF5 file
- h5dump - higher level view of file contents
- Utilities to convert to & from XML, GIF, HDF4
- Some commercial tools
- Future
  - Server-side software
  - More commercial support
  - Contributed software

HDF Supporters and Users

- ASCI Data Models and Formats (DMF) Group
- NASA Earth Science Data & Info System
- Army Research Lab DICE
  - Network Distributed Global Memory
  - HPC codes read/write into NDGM as if to HDF5
- TIKSL - Tele-immersion, collision of black holes
  - remote visualization and distributed file I/O
- NEXUS
  - exchange of neutron & synchrotron scattering data
- Many others. See:
  - http://hdf.ncsa.uiuc.edu/users.html

Major User #1: ASCI

- **ASCI Data Models and Formats (DMF) Group**
  - **open standard exchange format and I/O library**
  - **DOE tri-lab ASCI applications**
- **HDF requirements**
  - **large datasets (> a terabyte)**
  - **ASCI data types, especially meshes**
  - **good performance in massive parallel environments**

Major User #2: EOSDIS

- **ESDIS Project**
  - open standard exchange format and I/O library for EOSDIS
  - EOS applications
- **HDF requirements**
  - Earth science data types (HDF-EOS, etc,)
  - HDF tools, utilities, access software

HDF5 objects and structures

HDF5 File

- **HDF5 file: container for storing scientific data**
  - **Primary Objects:**
    - **Groups**
    - **Datasets**
  - **Secondary Objects:**
    - **Datatypes**
    - **Dataspaces**
    - **Attributes**

HDF5 Dataset

- **HDF5 dataset: data array and metadata**
- **Data array**
  - **an ordered collection of identically typed data items distinguished by their indices (subscripts)**
- **Metadata**
  - **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**
  - **datatype**

Dataset Components

Metadata

Dataspace

Rank: 3

Dimensions: Dim_1 = 4, Dim_2 = 5, Dim_3 = 7

Datatype

IEEE 32-bit float

Storage info

Chunked

compressed

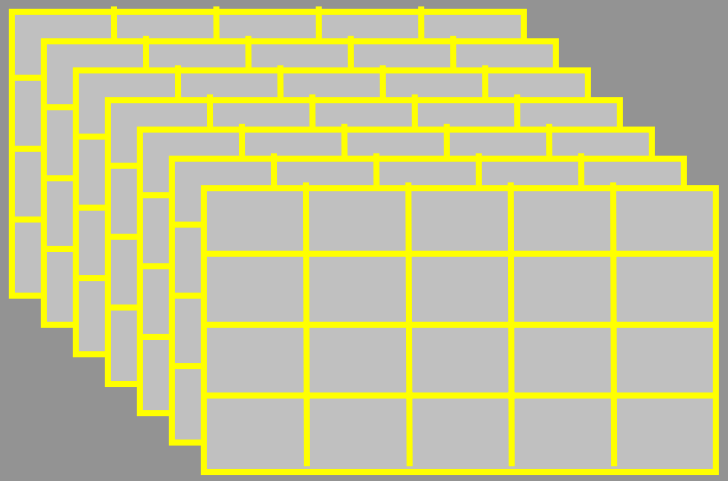
Attributes

time = 32.4

pressure = 987

temp = 56

Data



Datatypes

- **A *datatype* is a classification specifying the interpretation of a data element**
  - **Specifies for a given data element**
    - **the set of possible values it can have**
    - **the operations that can be performed**
    - **how the values of that type are stored**

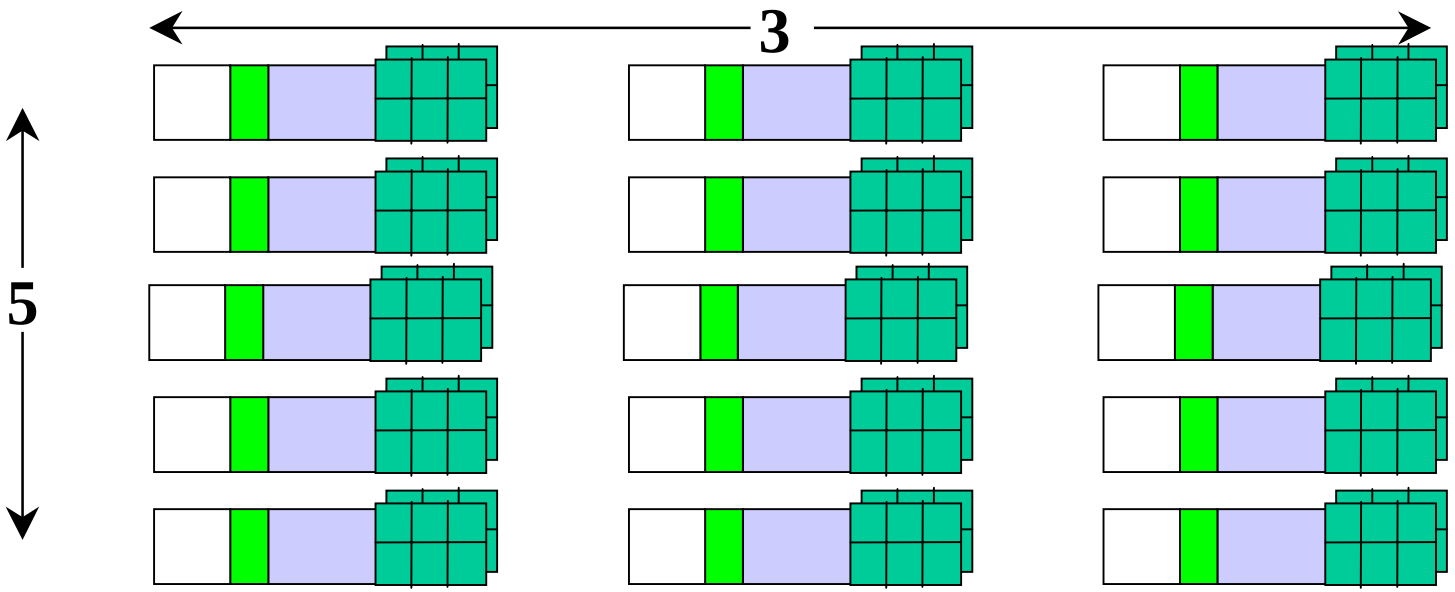
HDF5 Datatypes

- **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**
  - **array**

HDF5 Datatypes

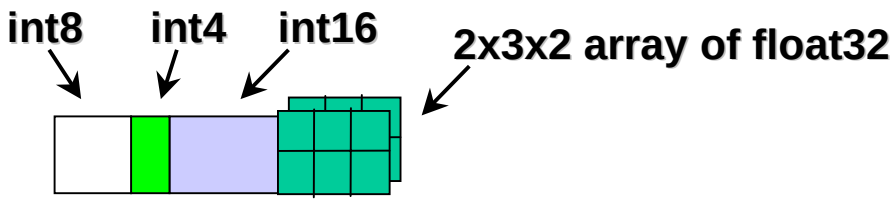
- **Compound types**
  - **Comparable to C structs**
  - **Members can be atomic or compound types**
  - **Members can be multidimensional**

HDF5 dataset: array of records



Dimensionality: 5 x 3

Datatype:



Record

Dataspaces

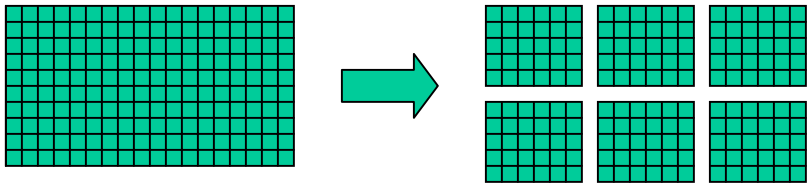
- **Dataspace: spatial information about a dataset**
  - **How elements are organized**
    - **Rank and dimensions**
    - **Permanent part of dataset definition**
  - **A subset of points, for partial I/O**
    - **Needed only during I/O operations**
- **Apply to datasets in memory or in the file**

Attributes

- An *attribute* is
  - a small piece of data of the form “name = value”
  - attached to an object
- Operations are scaled-down versions of the dataset operations
  - Not extendible
  - No compression
  - No partial I/O

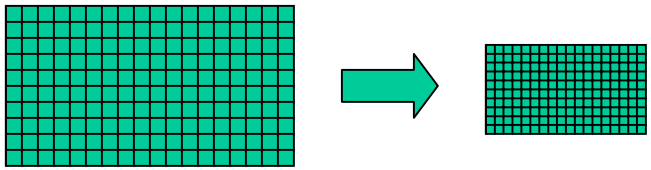
Special Storage Options

chunked



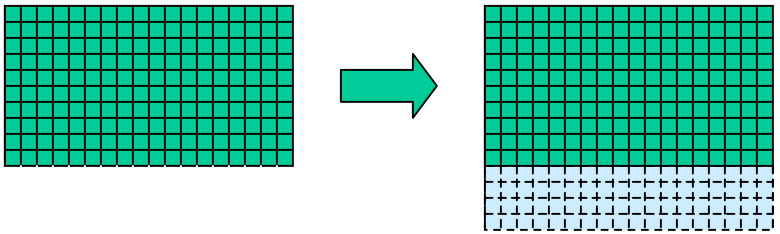
Better subsetting access time; extendable

compressed



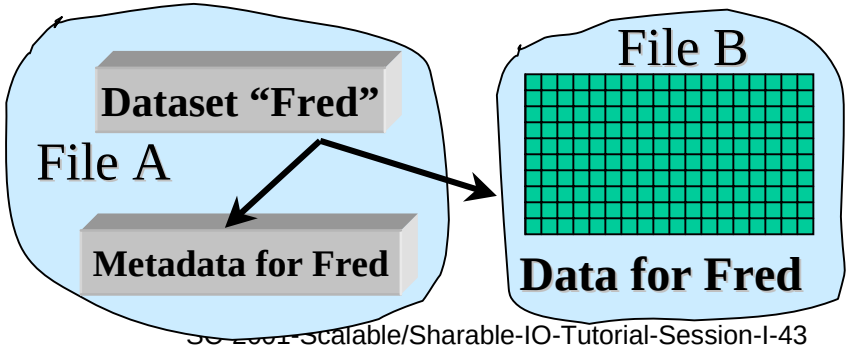
Improves storage efficiency, transmission speed

extendable



Arrays can be extended in any direction

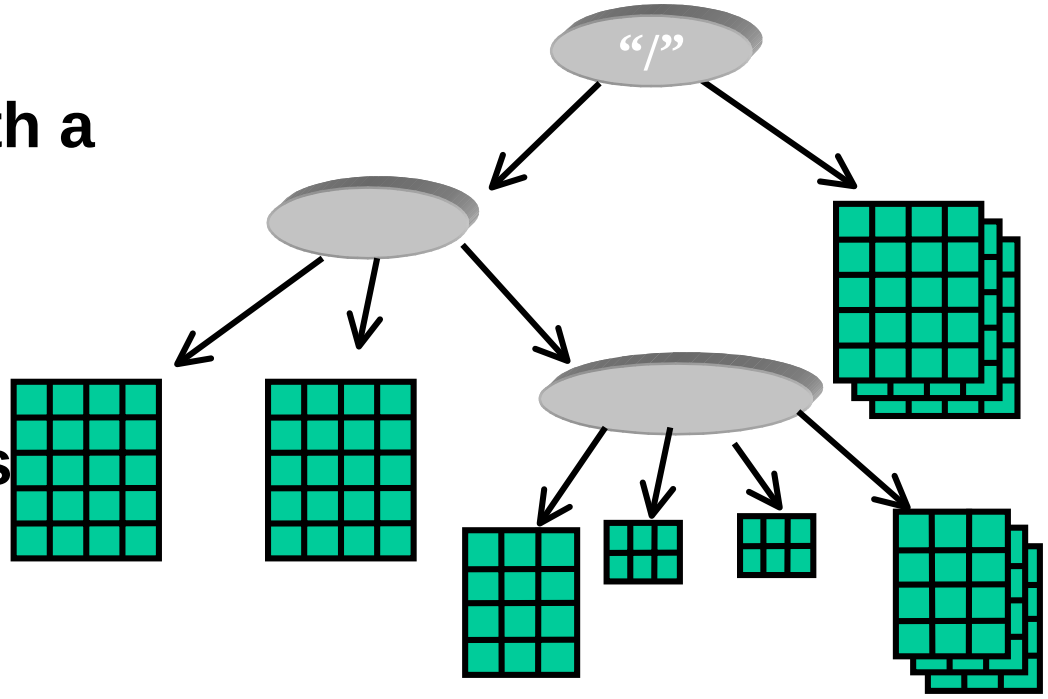
Split file



Metadata in one file, raw data in another.

Groups

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

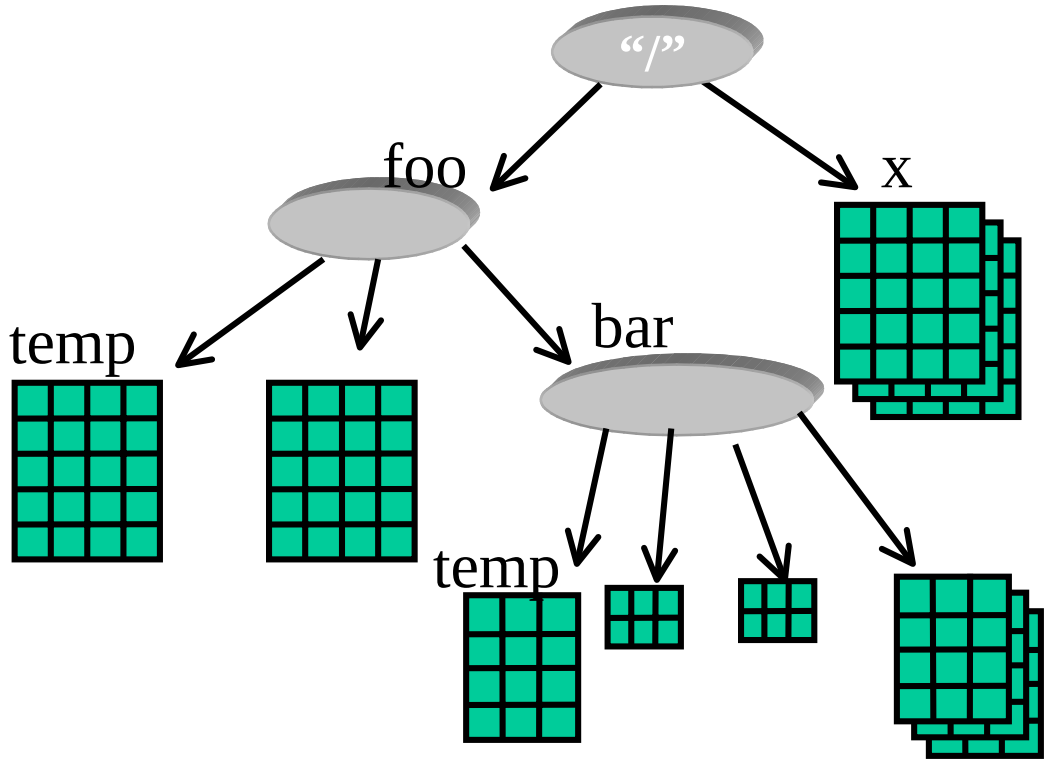


Groups

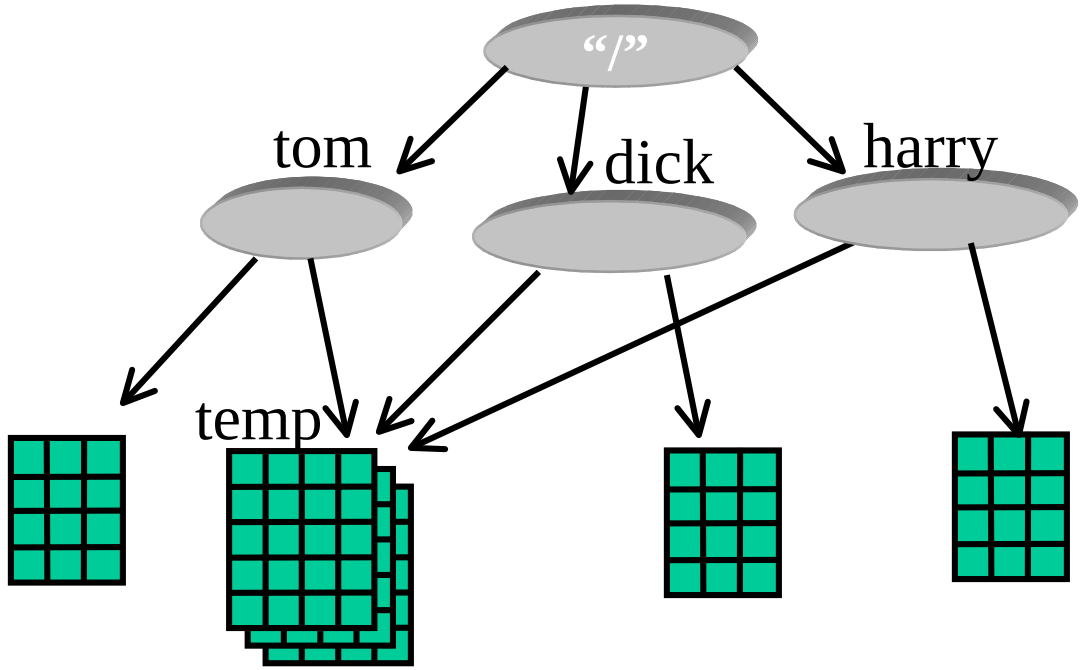
- **Similar to Unix directories**
  - **Location describe by “path”**
  - **Components separated by slashes**
  - **A root group called ‘‘/”**
  - **Absolute and relative names**
  - **Hard and soft (symbolic) links**
- **Different from Unix directories**
  - **A graph rather than a tree**
  - **No automatic ‘‘..” entries**

HDF5 objects are identified and located by their *pathnames*

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



Groups & members of groups can be shared



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

Application programming interface (API) for creating and accessing HDF5 objects and files

Programming model for sequential access

API Topics

- General info about HDF5 programming
- Creating an HDF5 file
- Creating a dataset in a file
- Writing and reading a dataset
- Creating attributes for an object
- Creating a group in a file
- Creating a dataset in a group

Goals of HDF5 Library

- **Flexible API to support a wide range of operations on data**
- **High performance access in serial and parallel computing environments**
- **Compatibility with common data models and programming languages**

The HDF5 API

- **Currently has C, Fortran 90, Java bindings C++ .**
- **C routines begin with prefix H5*, where * is a single letter indicating the object on which the operation is to be performed.**

Example APIs:

H5D : **D**ataset interface *e.g..* **H5Dread**
H5F : **F**ile interface *e.g..* **H5Fopen**
H5S : data**S**pace interface *e.g..* **H5Sclose**

The General Paradigm

- Objects are opened or created
- Then accessed
- Finally closed

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**

Return Values (for C)

Functions return a non-negative value if successful and a negative value if not.

Example Error Output:

```
HDF5-DIAG: Error detected in thread 0.  Back trace follows.
  #000: H5F.c line 1463 in H5Fopen(): unable to open file
    major(04): File interface
    minor(12): Unable to open file
  #001: H5F.c line 1058 in H5F_open(): file does not exist
    major(04): File interface
    minor(12): Unable to open file
```

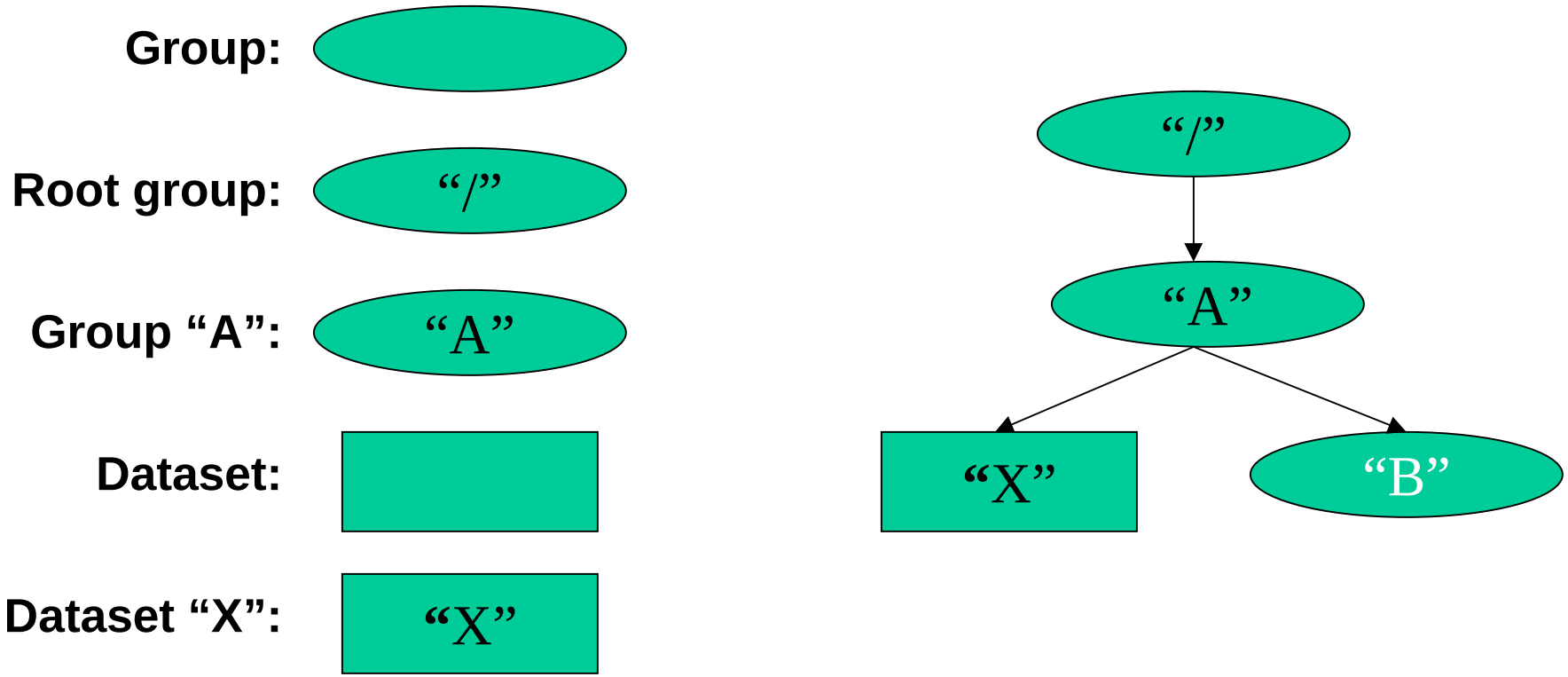
HDF5 C Programming Issues

For portability, HDF5 library have their own defined types:

hid_t:	object identifiers (native *integer*)
hsize_t:	size used for dimensions (*unsigned long* or *unsigned long long*)
hssize_t:	for specifying coordinates and sometimes for dimensions (*signed long* or *signed long long*)
herr_t:	function return value

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

Graphical Notations



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).

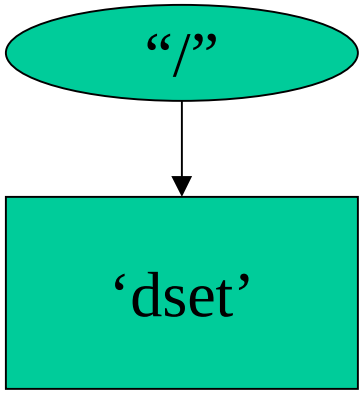
<*names*> is one or more appropriate object names.

Data Description Language (DDL)

- Describes an HDF5 file in a formal format
- Output of h5dump is in DDL
- Note: h5dump also outputs XML

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
      }
   }
}
}
```



Creating an HDF5 File

Steps to Create a File

- **Specify File Creation and Access Property Lists if necessary**
- **Create a file**
- **Close the file and the property lists, if necessary**

Property Lists

File Creation Property List: Controls file metadata (size of the user-block, sizes of file data structures, etc.).
(For **C**: Specifying `H5P_DEFAULT` uses the default values.)

Access Property List: Controls different methods of performing I/O on files.
(For **C**: Specifying `H5P_DEFAULT` uses the default values.)

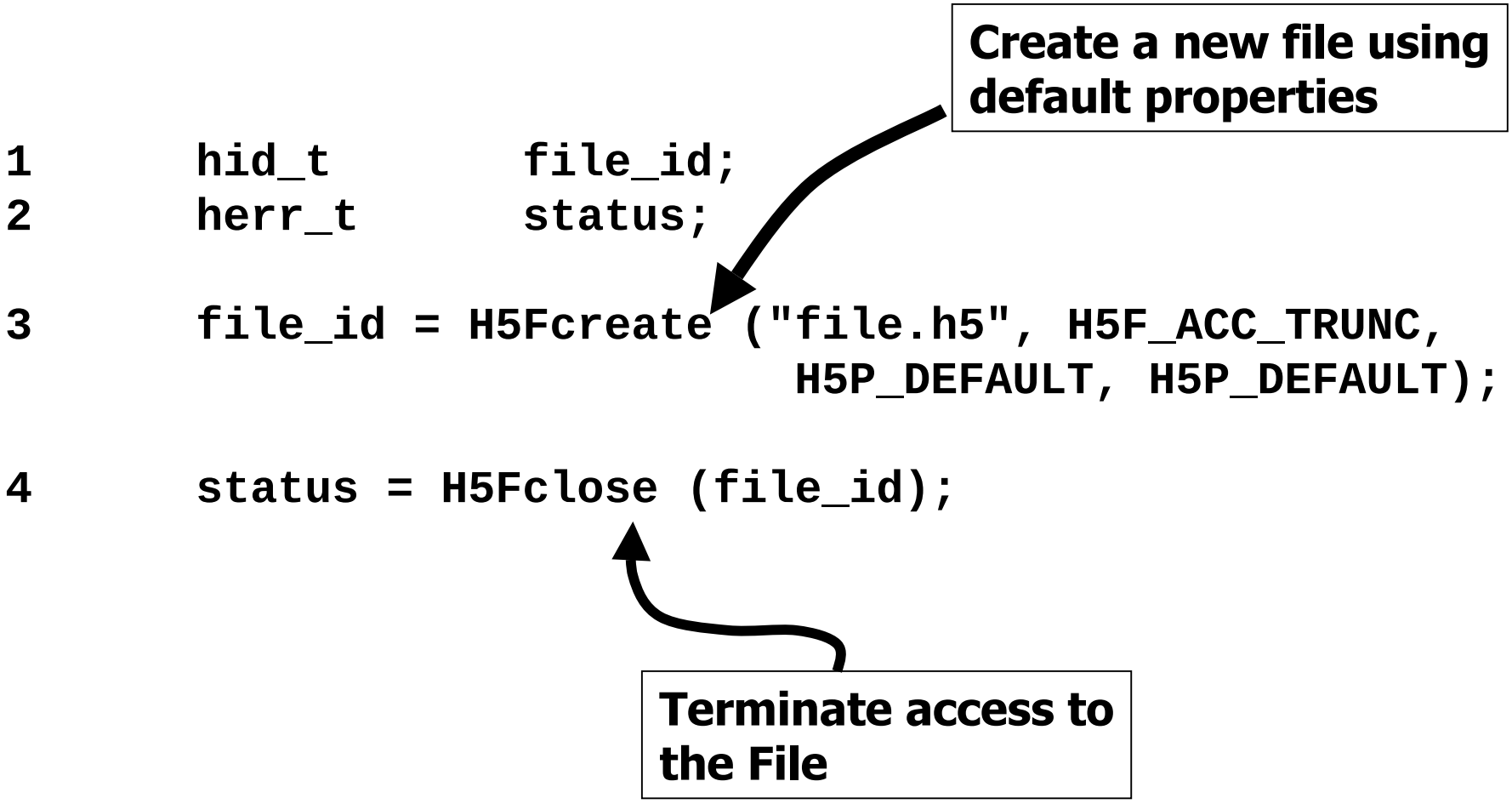
hid_t H5Fcreate (const char *name, unsigned flags, hid_t create_id, hid_t access_id)

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

herr_t H5Fclose (hid_t file_id)

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

Example 1

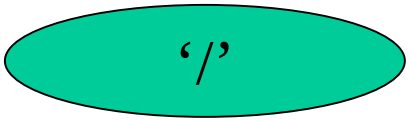

Create a new file using default properties
1 hid_t file_id;
2 herr_t status;
3 file_id = H5Fcreate ("file.h5", H5F_ACC_TRUNC,
H5P_DEFAULT, H5P_DEFAULT);
4 status = H5Fclose (file_id);
Terminate access to the File

h5_crtfile.c

```
 1   #include <hdf5.h>
 2   #define FILE "file.h5"
 3
 4   main() {
 5
 6      hid_t       file_id;   /* file identifier */
 7      herr_t      status;
 8
 9      /* Create a new file using default properties. */
10      file_id = H5Fcreate (FILE, H5F_ACC_TRUNC,
                             H5P_DEFAULT, H5P_DEFAULT);
11
12      /* Terminate access to the file. */
13      status = H5Fclose (file_id);
14   }
```

Example 1: h5dump Output

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



Create a Dataset

Dataset Components

Metadata

Dataspace

Rank

3

Dimensions

Dim_1 = 4

Dim_2 = 5

Dim_3 = 7

Datatype

IEEE 32-bit float

Storage info

Chunked

compressed

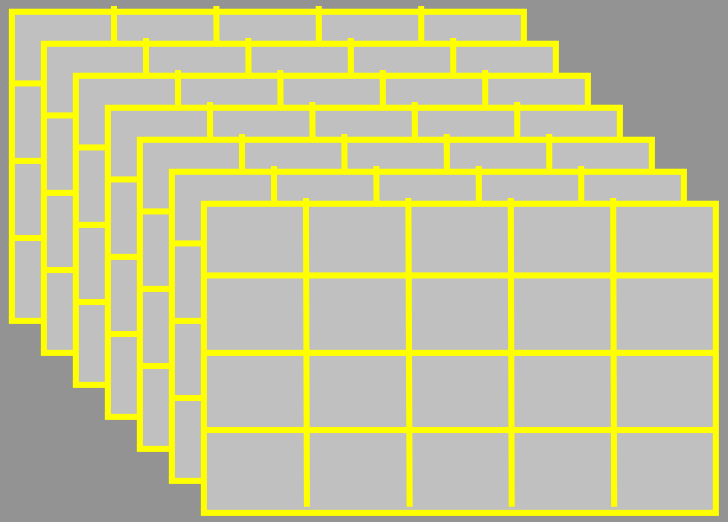
Attributes

time = 32.4

pressure = 987

temp = 56

Data



Steps to Create a Dataset

- Obtain location ID where dataset is to be created
- Define dataset characteristics (datatype, dataspace, dataset creation property list, if necessary)
- Create the dataset
- Close the datatype, dataspace, and property list, if necessary
- Close the dataset

Step 1

- **Obtain the *location identifier* where the *dataset* is to be created**

Location Identifier: the file or group identifier in which to create a dataset

Step 2

- **Define the dataset characteristics**
  - **datatype (e.g. integer)**
  - **dataspace (2 dimensions: 100x200)**
  - **dataset creation properties (e.g. chunked and compressed)**

Datatypes

A *datatype* is a collection of properties that provide complete information for data conversion to or from that datatype.

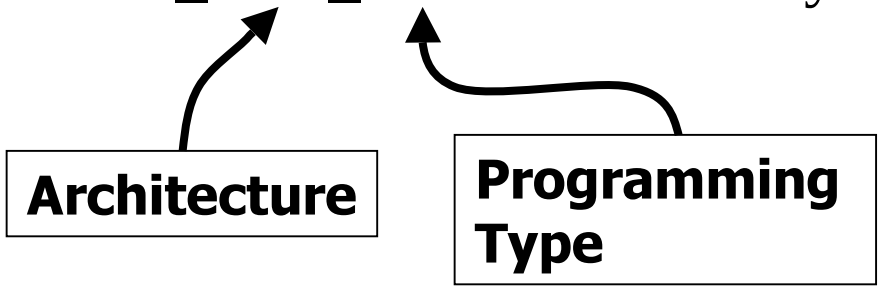
HDF5 Predefined Datatypes

- A predefined type is a handle. It is not persistent.
- Predefined types are opened and closed by HDF5.
- New types can be derived from existing types.

Standard Predefined Datatypes

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 are the same on different platforms.

Native Predefined Datatypes

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 different platforms.

Dataspaces

- **Dataspace: size and shape of dataset and subset**
  - **Dataset**
    - **Rank: number of dimension**
    - **Dimensions: sizes of all dimensions**
    - **Permanent – part of dataset definition**
  - **Subset**
    - **Size, shape and position of selected elements**
    - **Needed primarily during I/O operations**
- **Applies to arrays in memory or in the file**

Simple Dataspace

- **A *simple* dataspace is a regular N-dimensional array of data points.**

- **Currently in HDF5, only simple dataspaces are supported.**

Creating a Simple Dataspace

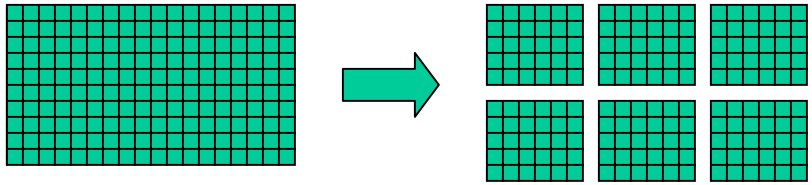
```
hid_t  H5Screate_simple (int rank,
                         const hsize_t * dims,
                         const hsize_t *maxdims)
```

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.

Dataset Creation Property List

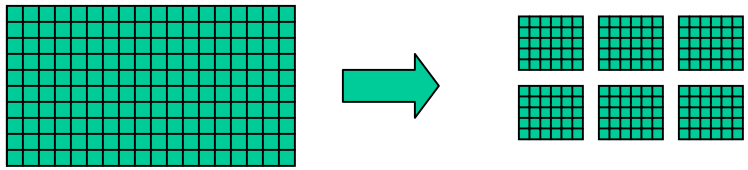
The *dataset creation property list* contains information on how to organize data in storage.

Chunked



Better subsetting access time; extendable

Chunked and compressed



Improves storage efficiency, transmission speed

Dataset Creation Property Lists

- **Dataset creation properties include**
  - **Compression**
  - **Extendibility**
  - **External storage**
  - **Chinking**
  - **Fill values**

Property List Example

- **Creating a dataset with ‘‘deflate” compression**

```
hid_t_dcpl = H5Pcreate(H5P DATASET CREATE);
H5Pset_deflate(dcpl, 9);
hid_t_v = H5Dcreate(file, ’’velocity’’, ..., dcpl);
H5Pclose(dcpl);
```

Remaining Steps to Create a Dataset

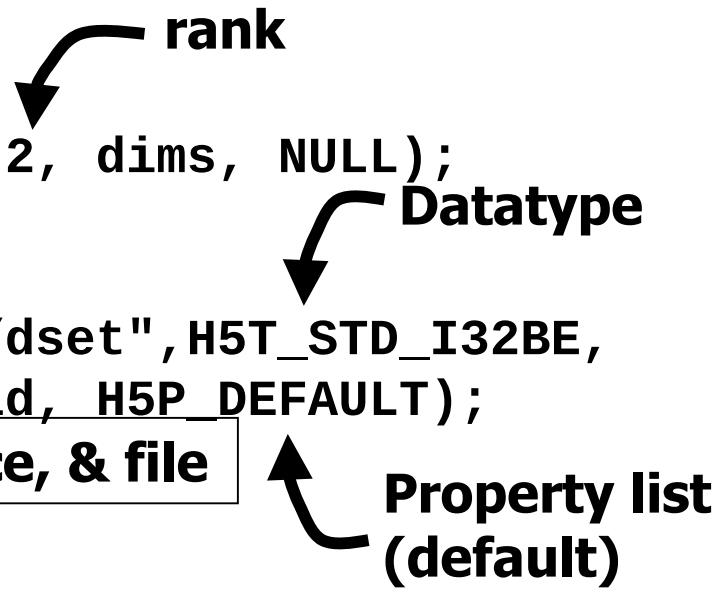
- Create the dataset
- Close the datatype, dataspace, and property list, if necessary
- Close the dataset

hid_t H5Dcreate (hid_t loc_id, const char *name,
hid_t type_id, hid_t space_id,
hid_t create_plist_id)

loc_id	**IN**: Identifier of file or group to create the dataset within
name	**IN**: The name of (the link to) the dataset to create
type_id	**IN**: Identifier of datatype to use when creating the dataset
space_id	**IN**: Identifier of dataspace to use when creating the dataset
create_plist_id	**IN**: Identifier of the dataset creation property list (or H5P_DEFAULT)

Example 2

```
1  hid_t       file_id, dataset_id, dataspace_id;
2  hsize_t     dims[2];
3  herr_t      status;

   Create a new file

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

   Create a dataspace

5  dims[0] = 4;
6  dims[1] = 6;
7  dataspace_id = H5Screate_simple (2, dims, NULL);

   Create a dataset

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

   Terminate access to dataset, dataspace, & file

9  status = H5Dclose (dataset_id);
10 status = H5Sclose (dataspace_id);
11 status = H5Fclose (file_id);
```



h5_crtdat.c

```
 1   #include <hdf5.h>
 2   #define FILE "dset.h5"
 3
 4   main() {
 5
 6      hid_t       file_id, dataset_id, dataspace_id;
 7      hsize_t     dims[2];
 8      herr_t      status;
 9
10      /* Create a new file using default properties. */
11      file_id = H5Fcreate (FILE, H5F_ACC_TRUNC,
                             H5P_DEFAULT, H5P_DEFAULT);
12
13      /* Create the data space for the dataset. */
14      dims[0] = 4;
15      dims[1] = 6;
16      dataspace_id = H5Screate_simple(2, dims, NULL);
```

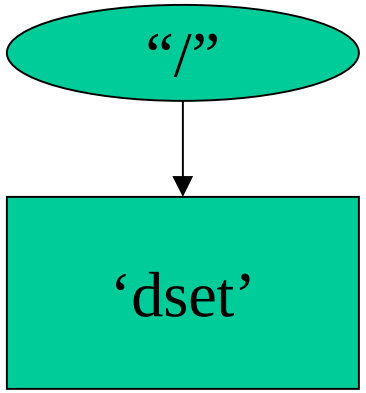
h5_crtdat.c (continued)

```
17
18    /* Create the dataset. */
19    dataset_id = H5Dcreate(file_id,"/dset",H5T_STD_I32BE,
20                           dataspace_id, H5P_DEFAULT);
21
22    /* End access to the dataset and release
         resources used by it. */
23    status = H5Dclose (dataset_id);
24
25    /* Terminate access to the data space. */
26    status = H5Sclose (dataspace_id);
27
28    /* Close the file. */
29    status = H5Fclose (file_id);
30 }
```

Example2: h5dump Output

Create an empty 4x6 dataset

```
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
            }
      }
 }
 }
```



Writing and Reading Datasets

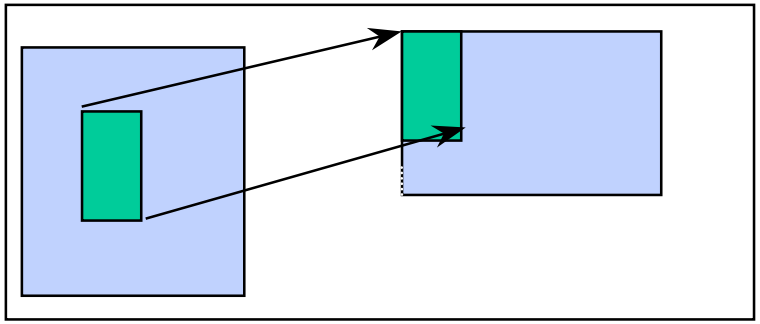
Dataset I/O

- **Dataset I/O involves**
  - **reading or writing**
  - **all or part of a dataset**
- **During I/O operations data is translated between the source & destination**
  - **data types (e.g. 16-bit integer => 32-bit integer)**
  - **dataspace (e.g. 10x20 2d array => 200 1d array)**
  - **also compressed/uncompressed, etc.**

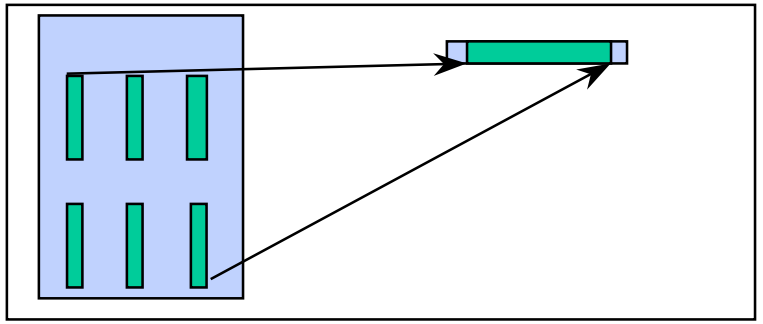
Partial I/O

- **Selection regions**
  - **Define shapes of memory and file spaces**
  - **Region in memory can differ from region in file**
- **Selection regions can be**
  - **Hyperslabs**
  - **Points**
  - **Unions of hyperslabs or points**

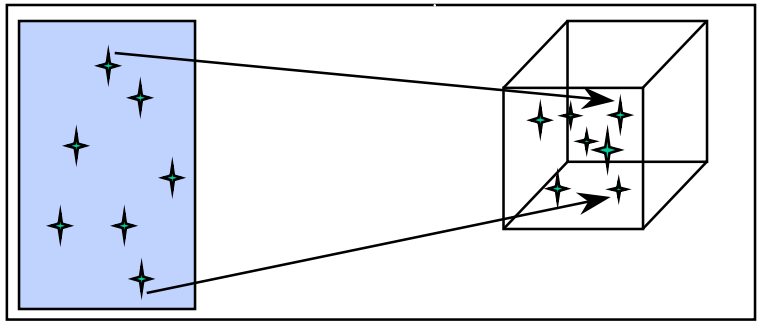
Sample Mappings between File Dataspaces and Memory Dataspaces



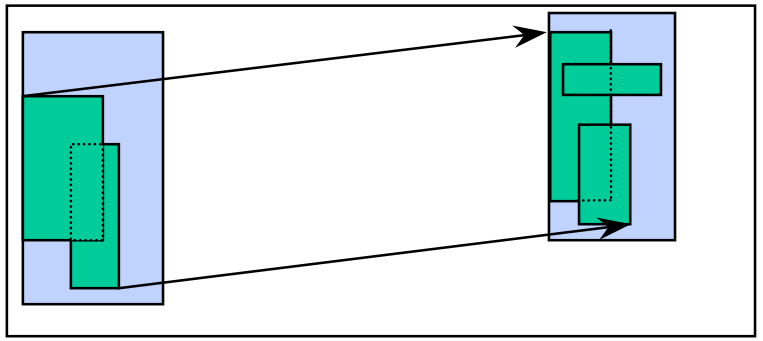
(a) A hyperslab from a 2D array to the corner of a smaller 2D array



(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.

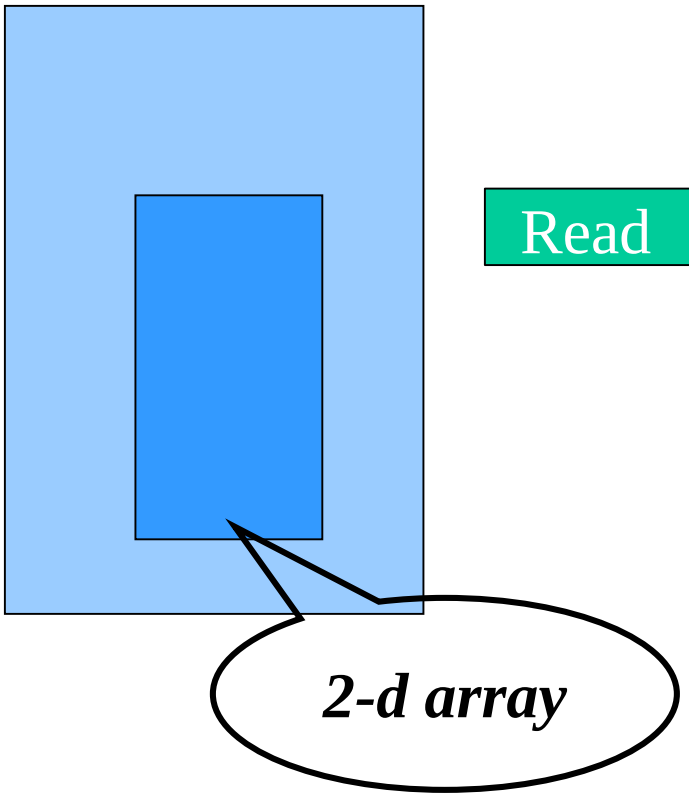


(d) Union of hyperslabs in file to union of hyperslabs in memory. Number of elements must be equal.

Reading Dataset into Memory from File

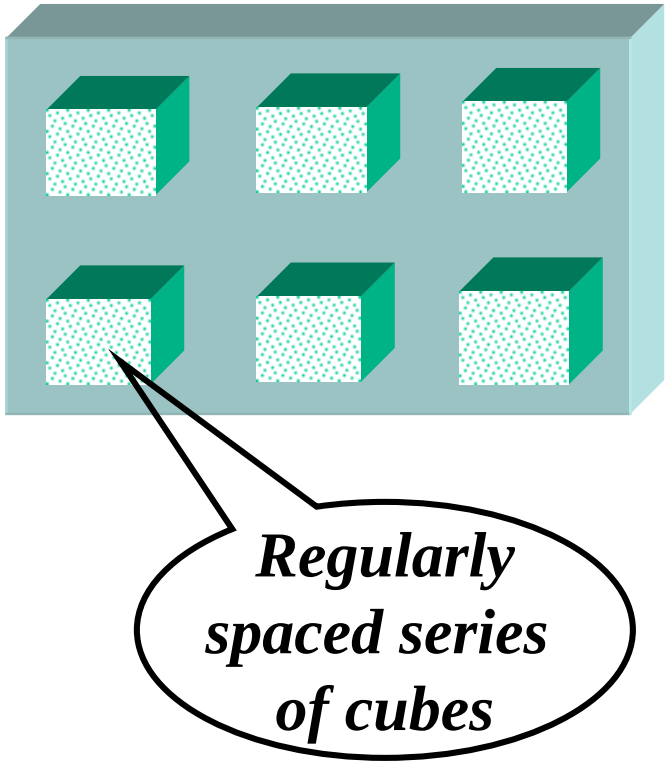
File

2D array of integers



Memory

3D array of floats



Steps for Dataset Writing/Reading

- Open the dataset to obtain the dataset ID
- Specify
  - Memory datatype
  - Memory dataspace
  - File dataspace
  - Transfer properties (optional)
- Perform the desired operation on the dataset
- Close dataspace, datatype and property lists

A dataspace can be specified in three ways

- **Create a new dataspace**
  - **H5Screate_simple()**
- **Copy an existing dataspace**
  - **H5Scopy()**
- **Retrieve a dataspace from a dataset**
  - **H5Dget_space()**

Data Transfer Property List

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

Reading and Writing

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

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,void*buf)

dataset_id	**IN**:	Identifier of dataset to read from
mem_type_id	**IN**:	Identifier of memory datatype
mem_space_id	**IN**:	Identifier of memory dataspace (or H5S_ALL for entire dataset)
file_space_id	**IN**:	Identifier of file dataspace (or H5S_ALL)
xfer_plist_id	**IN**:	Identifier of data transfer property list (or H5P_DEFAULT)
buf	**OUT**:	Buffer to store data read from file

Opening the File and Dataset

hid_t H5Fopen (const char *name, unsigned flags, hid_t access_id)

name	**IN**:	Name of the file to access
flags	**IN**:	File access flags
access_id	**IN**:	Identifier of the file access properties list (or H5P_DEFAULT)

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

Example 3

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

Initialize buffer

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

Open existing file and dataset

```
7  file_id = H5Fopen ("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);
8  dataset_id = H5Dopen (file_id, "/dset");
```

Write to and read from dataset

```
9  status = H5Dwrite (dataset_id, H5T_NATIVE_INT,
                      H5S_ALL, H5S_ALL, H5P_DEFAULT, dset_data);
10 status = H5Dread (dataset_id, H5T_NATIVE_INT,
                      H5S_ALL, H5S_ALL, H5P_DEFAULT, dset_data);
```

Close dataset and file

h5_rdwt.c

```
 1  #include <hdf5.h>
 2  #define FILE "dset.h5"
 3
 4  main() {
 5
 6    hid_t       file_id, dataset_id;  /* identifiers */
 7    herr_t      status;
 8    int         i, j, dset_data[4][6];
 9
10    /* Initialize the dataset. */
11    for (i = 0; i < 4; i++)
12       for (j = 0; j < 6; j++)
13          dset_data[i][j] = i * 6 + j + 1;
14
15    /* Open an existing file. */
16    file_id = H5Fopen (FILE, H5F_ACC_RDWR, H5P_DEFAULT);
17
18    /* Open an existing dataset. */
19    dataset_id = H5Dopen (file_id, "/dset");
20
```

h5_rdwt.c (continued)

```
21    /* Write the dataset. */
22    status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL,
23                      H5S_ALL, H5P_DEFAULT, dset_data);
24
25    status = H5Dread (dataset_id, H5T_NATIVE_INT, H5S_ALL,
26                      H5S_ALL, H5P_DEFAULT, dset_data);
27
28    /* Close the dataset. */
29    status = H5Dclose (dataset_id);
30
31    /* Close the file. */
32    status = H5Fclose (file_id);
33 }
```

Example 3: 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
      }
   }
}
}
```

Creating an Attribute

Attributes

- **Operations are scaled-down versions of the dataset operations**
  - **Not extendible**
  - **No compression**
  - **No partial I/O**

Steps to Create an Attribute

- Obtain the identifier for the object that the attribute will be attached to
- Define datatype & dataspace of the attribute
- Specify creation properties, if necessary
- Create the attribute
- Close the attribute and datatype, dataspace, and creation property list if necessary

hid_t H5Acreate (hid_t loc_id, const char *name,
hid_t type_id, hid_t space_id,
hid_t create_plist)

loc_id	**IN**:	Object (dataset, group, or named datatype) to be attached to.
*name	**IN**:	Name of attribute to create.
type_id	**IN**:	Identifier of datatype for attribute.
space_id	**IN**:	Identifier of dataspace for attribute.
create_plist	**IN**:	Identifier of creation property list (currently not used).

herr_t H5Awrite (hid_t attr_id, hid_t mem_type_id, void *buf)

attr_id	**IN:**	Identifier of an attribute to write.
mem_type_id	**IN:**	Identifier of attribute datatype (in memory).
*buf	**IN:**	Data to be written.

herr_t H5Aread (hid_t attr_id, hid_t mem_type_id, void *buf)

attr_id	**IN:**	Identifier of an attribute to read.
mem_type_id	**IN:**	Identifier of attribute datatype (in memory).
*buf	**OUT:**	Buffer for data to be read.

Example 4: C

```
1       hsize_t     dims;
2       int         attr_data[2]={100,200};
```

Open an existing file and dataset

```
3       file_id = H5Fopen ("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);
4       dataset_id = H5Dopen (file_id, "/dset");
```

Create the dataspace for the attribute and the attribute

```
5       dims = 2;
6       dataspace_id = H5Screate_simple (1, &dims, NULL);
7       attribute_id = H5Acreate (dataset_id, "attr", H5T_STD_I32BE,
                                  dataspace_id, H5P_DEFAULT);
```

Write the attribute data

```
8       status = H5Awrite (attribute_id, H5T_NATIVE_INT, attr_data);
```

Close all open interfaces

h5_crtatt.c

```
 1   #include <hdf5.h>
 2   #define FILE "dset.h5"
 3
 4   main() {
 5
 6   hid_t       file_id, dataset_id, attribute_id,
                                  dataspace_id;
 7   hsize_t     dims;
 8   int         attr_data[2];
 9   herr_t      status;
10
11   /* Initialize the attribute data. */
12   attr_data[0] = 100;
13   attr_data[1] = 200;
14
15   /* Open an existing file and dataset */
16   file_id = H5Fopen (FILE, H5F_ACC_RDWR, H5P_DEFAULT);
17   dataset_id = H5Dopen (file_id, "/dset");
18
```

h5_crtatt.c (continued)

```
19    /* Create the data space for the attribute. */
20    dims = 2;
21    dataspace_id = H5Screate_simple (1, &dims, NULL);
22
23    /* Create a dataset attribute. */
24    attribute_id = H5Acreate (dataset_id, "attr",
25                  H5T_STD_I32BE,dataspace_id, H5P_DEFAULT);
26
27    /* Write the attribute data. */
28    status = H5Awrite (attribute_id, H5T_NATIVE_INT,
                            attr_data);
29
30    /* Close the attribute, dataspace, dataset, file */
31    status = H5Aclose (attribute_id);
32    status = H5Sclose (dataspace_id);
33    status = H5Dclose (dataset_id);
34    status = H5Fclose (file_id);
35 }
```

Example 4: h5dump Output (C)

```
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
              }
           }
        }
     }
     }
```

Creating a Group

Steps to Create a Group

- Obtain the location identifier where the group is to be created
- Create the group
- Close the group

hid_t H5Gcreate (hid_t loc_id, const char *name,
size_t size_hint)

loc_id	**IN:**	Identifier of the file or group where the new group is to be created
name	**IN:**	The name to give the new group
size_hint	**IN:**	A hint for the number of bytes to reserve to store the names which will be eventually added to the new group. A value of 0 is usually adequate, but sometimes the performance can be improved by specifying another value.

Example 5: C

Create a new file using default properties and create group "MyGroup" under root group

```
1     file_id = H5Fcreate ("group.h5", H5F_ACC_TRUNC,
2                          H5P_DEFAULT, H5P_DEFAULT);
3     group_id = H5Gcreate (file_id, "/MyGroup", 0);
```

Close the group and terminate access to the file

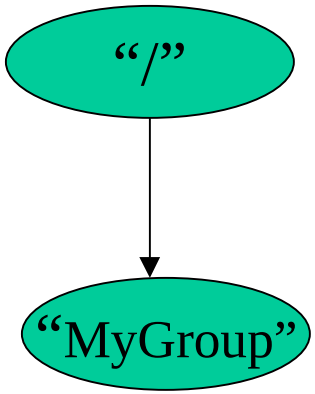
```
4       status = H5Gclose (group_id);
5       status = H5Fclose (file_id);
```

h5_crtgrp.c

```
 1   #include <hdf5.h>
 2   #define FILE "group.h5"
 3
 4   main() {
 5
 6      hid_t       file_id, group_id;  /* identifiers */
 7      herr_t      status;
 8
 9      /* Create a new file using default properties. */
10      file_id = H5Fcreate (FILE, H5F_ACC_TRUNC,
11                           H5P_DEFAULT, H5P_DEFAULT);
12
13      /* Create a group named "/MyGroup" in the file. */
14      group_id = H5Gcreate (file_id, "/MyGroup", 0);
15
16      /* Close the group and file. */
17      status = H5Gclose (group_id);
18      status = H5Fclose (file_id);
19   }19
```

Example 5: h5dump Output (C)

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



Absolute vs.. Relative Names

- **To create an HDF5 object, you must 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 can be either absolute or relative.**
- Absolute **name**
  - **begins with a "/"**
  - **is accessed beginning with the root group of the file.**
- Relative **name**
  - **does not begin with "/"**
  - **is accessed beginning with the specified location identifier.**

Absolute vs.. Relative Names

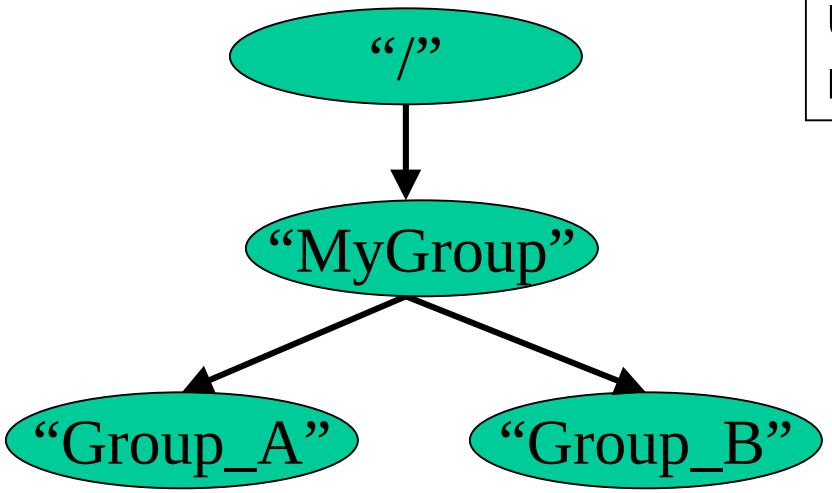
Location Type	Object Name	Description
File ID	**/foo/bar**	**Object *bar* in group *foo* in root group**
File ID	**/**	**Root group of file**
Group ID	**foo/bar**	**Object *bar* in group *foo* in specified group**

Example 6
Using absolute and relative path names

```
1   group1_id = H5Gcreate (file_id, "/MyGroup", 0);
2   group2_id = H5Gcreate (file_id, "/MyGroup/Group_A", 0);
3   group3_id = H5Gcreate (group1_id, "Group_B", 0);
```

uses absolute name

uses relative name



Creating a Dataset in a Group

hid_t H5Dcreate (hid_t loc_id, const char *name,
hid_t type_id, hid_t space_id,
hid_t create_plist_id)

The location identifier, *loc_id*, is the identifier for a group.

Opening a Group

hid_t H5Gopen (hid_t loc_id, const char *name)

loc_id	**IN:**	File or group identifier where group is to be opened
name	**IN:**	Name of group to open

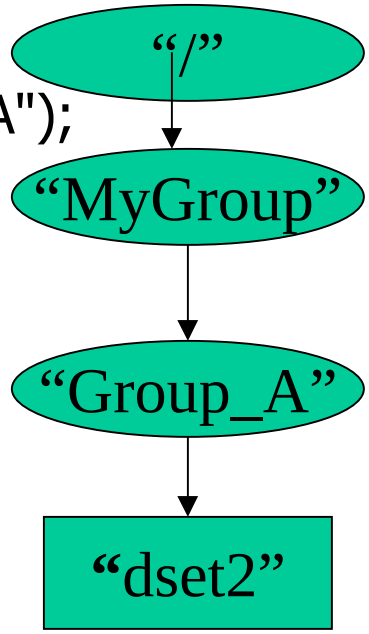
Example 7
Creating a dataset in a group

Open an existing group in a file

```
1       group_id = H5Gopen (file_id, "/MyGroup/Group_A");
```

Create a dataset in that group

```
•   dataset_id = H5Dcreate (group_id, "dset2",
•                           H5T_STD_I32BE,
•                           dataspace_id,
•                           H5P_DEFAULT);
```



Virtual File Layer: files needn't be files

- **Allows HDF5 to interface to disk, the network, memory, or a user-defined device**
- **Public API: write your own driver**