

# Data representation approaches for robotics: storage and logging

# Content

- CDF – Common Data Format
- HDF5 – Hierarchical Data Format version 5
- NetCDF – Network Common Data Form
- XDMF – Extensible Data Model and Format

CDF

CDF: concepts and data model

# CDF: model

- CDF is conceptual data abstraction for storing, manipulating and accessing multidimensional data sets.
- It does not emphasize actual physical format
- It defines form and function rather than specification of bits and bytes.
- The content of CDF is composed of 2 parts:
  - Collection of variables (data objects) comprising of scalars, vectors, and dimensional arrays.
  - Collection of attributes of these variables or CDF in global terms (metadata).
- CDF groups data by variables whose values are conceptually organized into arrays.
- CDF variable is a generic name or an object that represents data
- It does not have any scientific context associated with it. It can be data representing independent variable, dependent variable, time/date value, image, XML file etc.

# CDF: variables

- Variables or relationships between them are described through attributes
- The dimensionality of variable depends how it is specified by a user, e.g. scalars are 0-dim, image data array is 2-dim, volume data array is 3-dim. Maximum number of dimensions allowed is 10.
- Array for a particular data is called "*variable record*"
- Collection of arrays is called "*CDF record*".
- A CDF file/dataset can contain multiple "CDF records"
- There are two types of variables in CDF: **rVariables** and **zVariables**
- Every rVariable in CDF file/dataset must have the same number of dimensions and dimension sizes
- Zvariables may have a different number of dimensions or dimension sizes. They are more efficient (save more disk space) than rVariables, which are kept for backward compatibility.

# CDF: attributes

- While CDF variables represent the data, attributes are a mechanism to “describe these data”.
- There are two types of attributes: **global** and **variable** attributes
  - Global attributes, **gAttributes**, are used to describe CDF file/dataset. Example: file creation date, file author, source of data, and data set documentation
  - Variable attributes, **vAttributes**, describe individual variables **rVariables** and **zVariables**. Example: a field name for the variable, the valid minimum and maximum, the units in which the variable data values are stored, the format in which the data values are to be displayed, and a fill value for errant or missing data.
  - A gAttribute may contain multiple entries called “**gEntries**”. Example for this could be a modification history kept in some gAttribute X. Any additional changes could be taken into account by additional gEntries.
  - The entries of vAttributes correspond to each variables in CDF dataset. Each “**vEntry**” corresponds to each rVariable and zEntry to zVariable

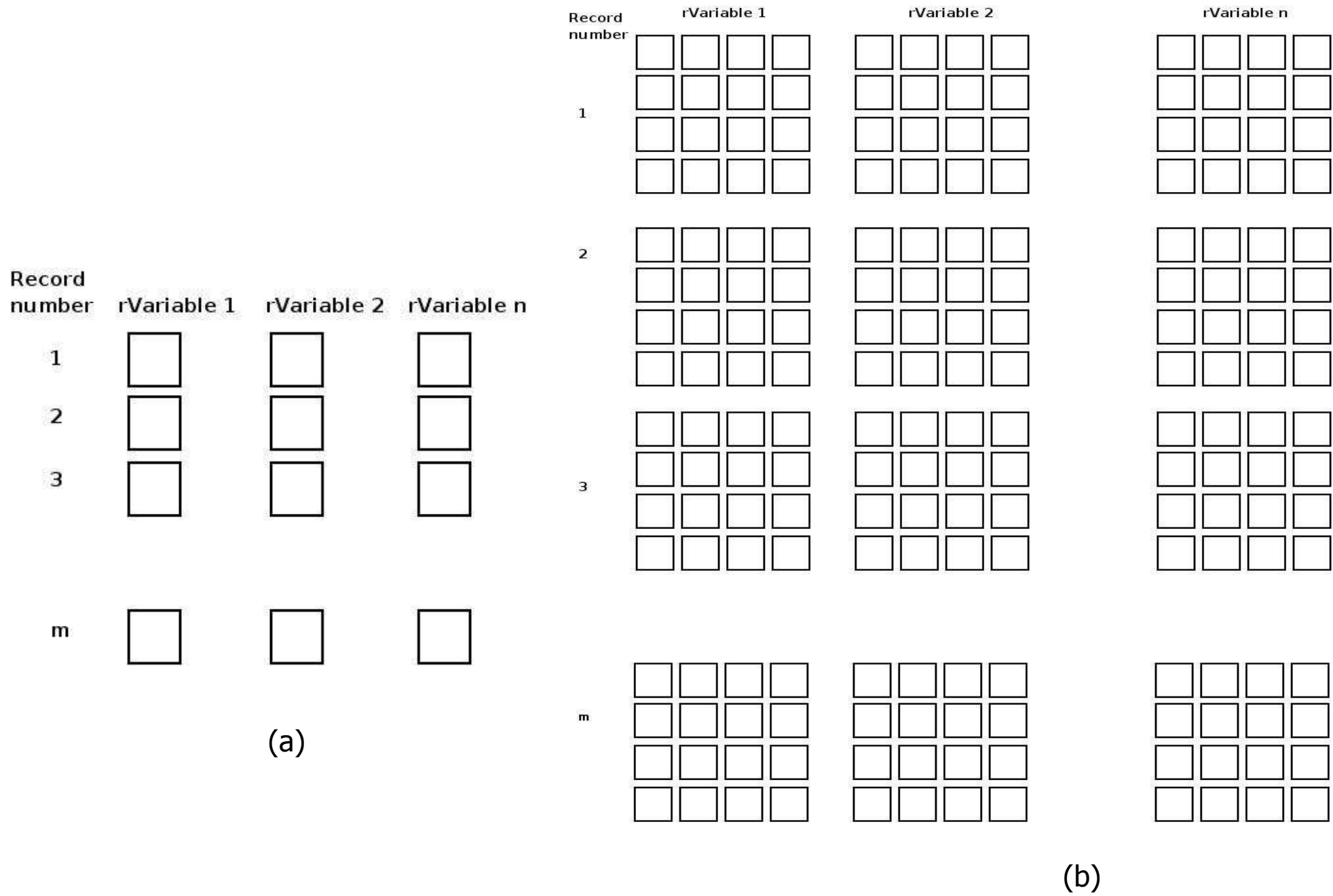


Figure 1. (a) 0-dimensional rVariables and (b) 2-dimensional rVariables in dataset

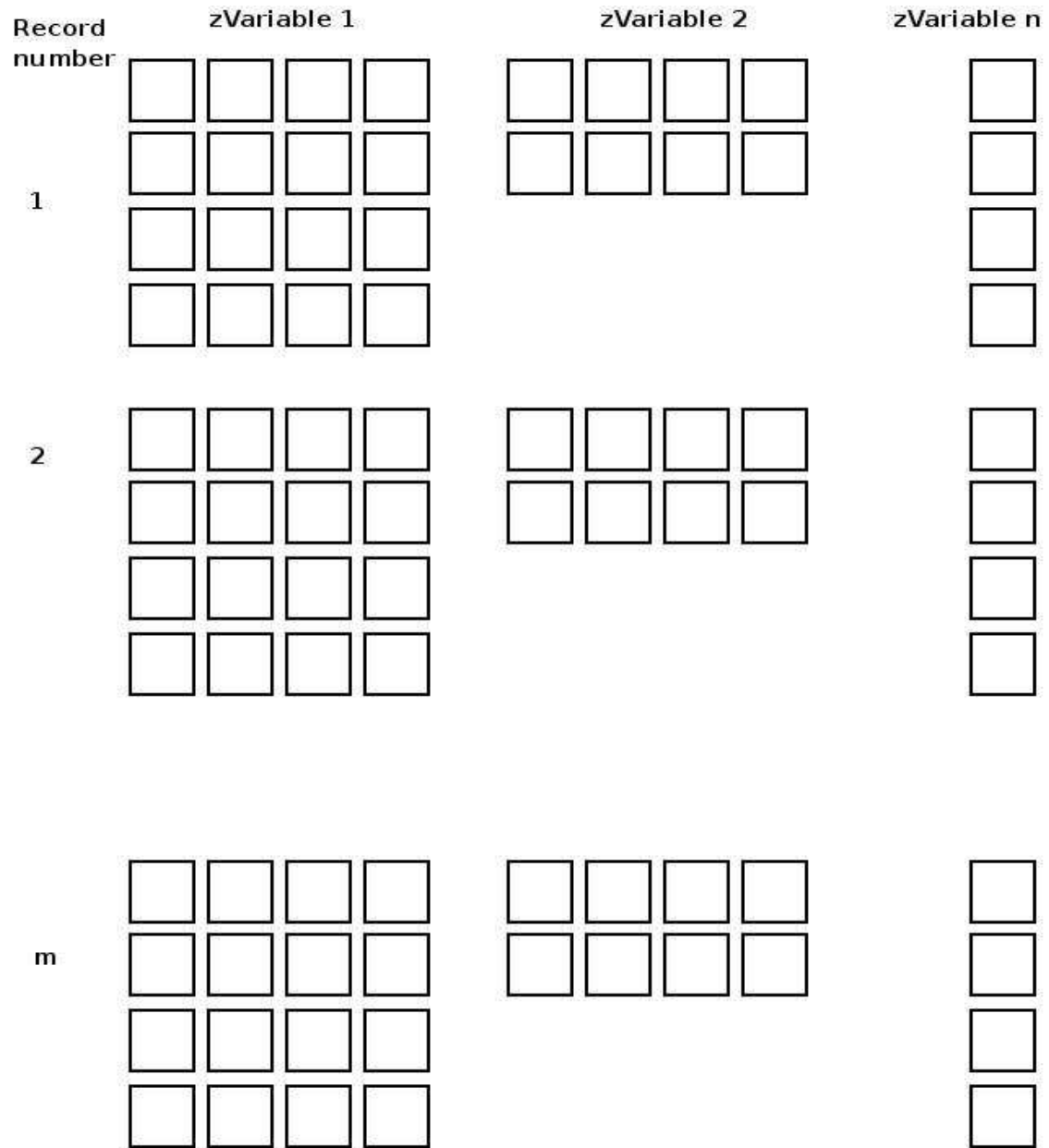


Figure 2. zVariables in dataset

CDF: technical details and  
programming tools

# CDF: r/w

- CDF library reads and writes to open files in 10240-byte blocks
- Maintains a cache of 10240 bytes for each open file
- In single-file CDFs an indexing scheme is used to keep track of where a variable's record is.
- The way variables and attributes stored in CDF file may be noncontiguous
- CDF dataset may have either ***single or multiple file formats***, where metadata and raw data are stored in separate files

# CDF: data compression and sparseness

- CDF library supports data compression for CDF dataset or individual variables (in single file format)
- Supports access to compressed data
- Compression/decompression are carried out automatically, user is not aware of it (one has to specify first time only).
- CDF library supports different type of compression algorithms
- In single file format two types of sparseness are allowed for CDF variables:
  - Sparse records – when variable is specified as having sparse records only those records are actually written to that variable will be stored.
  - Sparse arrays

# CDF: data accessibility and data types

- CDF supports several types of variable access modes:
  - *Single value* access allows only **one value** to be read/written to a variable with a single library call
  - *Hyper access* allows more than **one value** to be read/written to a variable with a single library call
  - *Sequential access* provides a way to sequentially read/write the values physically stored for a variable
  - *Multiple variable access* allows an application to read from/write to **several variables** in a single operation
- CDF supports a variety of data types consistent with the types in C and Fortran. All data types are based on an 8-bit byte. The size of a data type is the same across all platforms
- The types include: integers, floats, char and epoch equivalent time

# CDF: API and tools

- CDF has two levels of access
  - Through programming interface
  - High-level toolkit based on programming interface. It allows to create, browse, modify, export/import CDF data to/from text or XML file.
- CDF library comes with C, Java and Fortran APIs, which provide essential framework on which data analysis SW can be developed
- Supports creation of files bigger than 2 Gb.
- Protection of data integrity using simple redundancy check, asserting bits of the message/data send around.
- The same CDF file can be accessed without any modifications on many platforms as (**data encoding**): VAX (OpenVMS and POSIX shell), Sun (SunOS & Solaris), DECstation (ULTRIX), DEC Alpha (OSF/1 or Tru64 & OpenVMS), Silicon Graphics Iris and Power Series (IRIX), IBM RS6000 series (AIX), HP 9000 series (HP-UX), NeXT (Mach), PC (DOS, Windows 3.x, Windows NT/95/98/2000/XP, Linux, Cygwin & MinGW), Macintosh (MacOS X or Linux) or on **network XDR**.

# CDF: API and tools

- CDF library supports two types of interfaces
  - Standard interface (old and new versions) – is based on internal interface but much simpler, easy to learn and less efficient
  - Internal interface – is callable both from C and Fortran and consists of one routine in the form of CDFlib and CDF lib respectively
- CDF also has interfaces to such commercial software as IDL (Interactive Data Language), Matlab, Application Visualization System, FlexPro
- To such public domain software as OpenDX, CDAWlib, CDFx,
- DTWS (Data Translation Web Services) application client supports conversion between CDF and other data formats as netCDF, FITS, CDFML, ASCII, HDF4
- CDF Toolkit includes – CDFedit, CDFconvert, CDFexport, SkeletonCDF, SkeletonTable and many other
- CDF also provides XML based data and meta data description language, **CDFML**

**HDF5**

# HDF5: concepts and data model

# HDF5: abstract data model

- HDF implements a model for managing and storing data, it is domain data independent and can be tailored to some particular problem
- HDF model is composed of abstract data model and abstract storage model and libraries that implement this model (Programming model)
- Abstract data model is a conceptual model of data, data types and data organization. It is independent of storage model
- Storage model is a representation of the objects in abstract data model and it is defined in HDF5 file specs.
- Programming model manipulates objects from the abstract data model
- Library and storage data are concrete implementations of the respective models

# HDF5: abstract data model

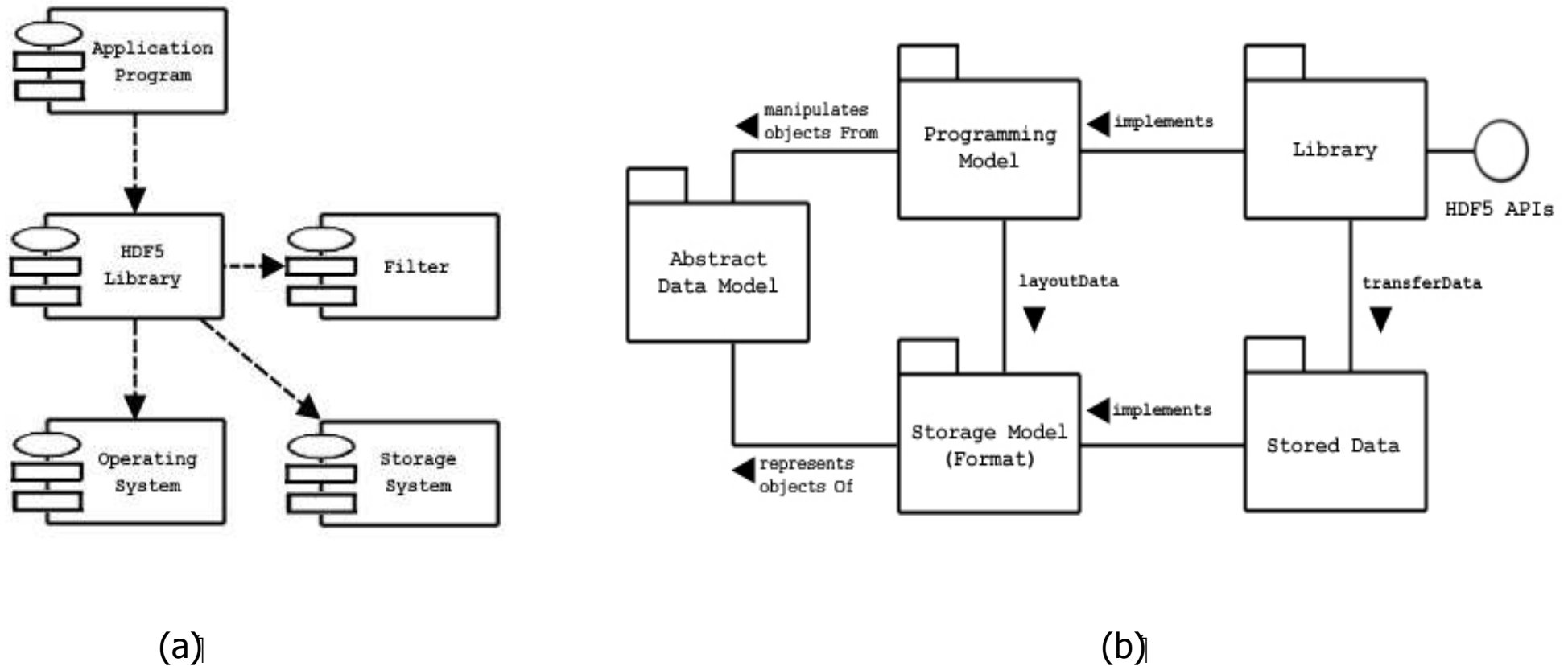


Figure 3. (a) Dependencies between application, OS and HDF5 library and (b) relationship between HDF5 models and implementations

# HDF5: application and library concepts

- The storage approach in HDF5 can be viewed across three levels
- **Level1** - Application program uses data structures that represent the problem and algorithms. These include tables, variables, arrays and meshes
- **Level2** – HDF5 library implements the objects of abstract data model. These include groups, datasets and attributes
- **Level3** – HDF5 specific storage model. On this level objects/data are stored on physical medium. These include header blocks, free lists, data blocks, B-tree etc
- The elements on the top layers are mapped to the ones on the layers below

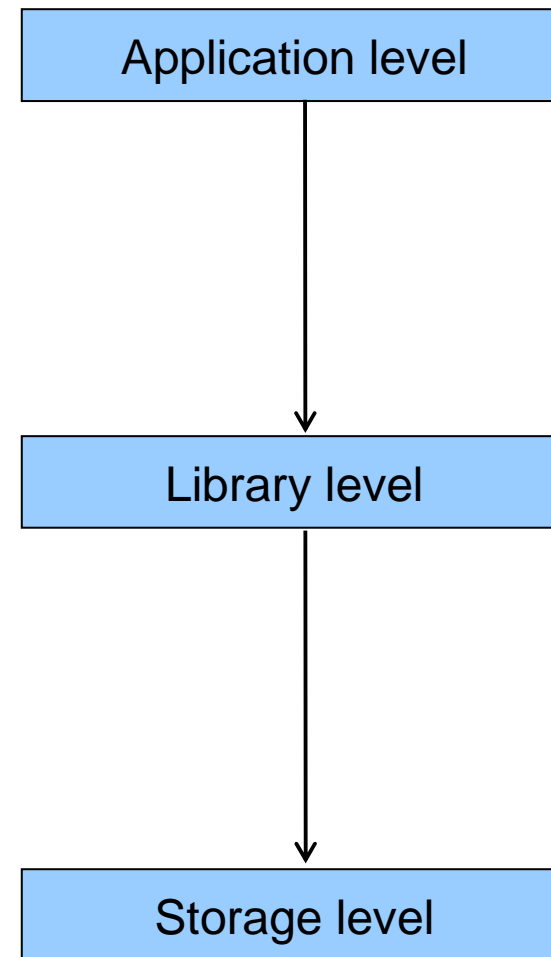
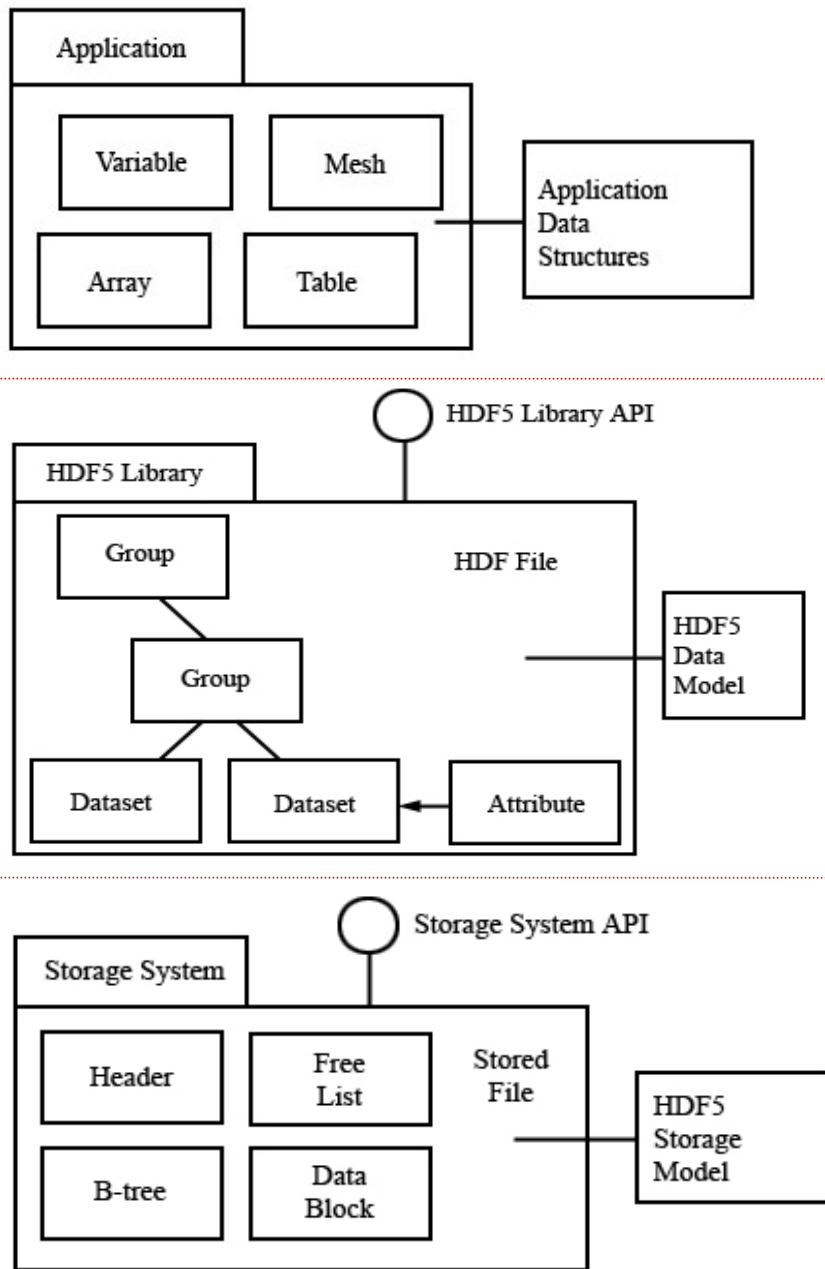


Figure 4. Layered view of data structures and associations between them

# HDF5: data storage concepts

- Abstract data model defines several concepts for storing complex data. These are:
  - **File** – a contiguous string of bytes in computer memory. The bytes represent zero or more objects of the model
  - **Group** – a group of objects
  - **Dataset** – multidimensional array of Data Elements, with Attributes and metadata
  - **Datatype** – a description of a specific class of data element, including its storage layout
  - **Dataspace** – description of the dimensions of a multidimensional array
  - **Attribute** – a named data value associated with a group, dataset or named datatype
  - **Property List** – a collection of parameters controlling options in the library. Some are permanently stored as part of the object; others are transient and apply to a specific access.

# HDF5: technical details and programming tools

# HDF5: objects

- A HDF5 file is the container for a collection of objects(groups, datasets)
- Every HDF5 file has at least one object, the **root group**
- All objects are members of the root group and have **unique ID** within a single file
- HDF5 file can be “mounted” as the part of another file
- Group membership is implemented via **Link Objects** which actually points to some **Named Object**
- Each link points to exactly one object, whereas there can be possibly many links pointing to the same object
- There are 3 types of Named Objects: Group, Dataset and Named datatype

# HDF5: relation between concepts

- Dataset is a multidimensional array of Data Elements. The shape of this array is described by Dataspace object (e.g current size and maximum size of the array)
- A Data Element is a single unit of data that can be an array, number, character or a record of heterogeneous data elements
- A Data Element is a set of bits, the layout of which is described by the Datatype
- The Dataset object manages the storage and access to the Data. It also maps between the conceptual array of elements and the actual stored data.

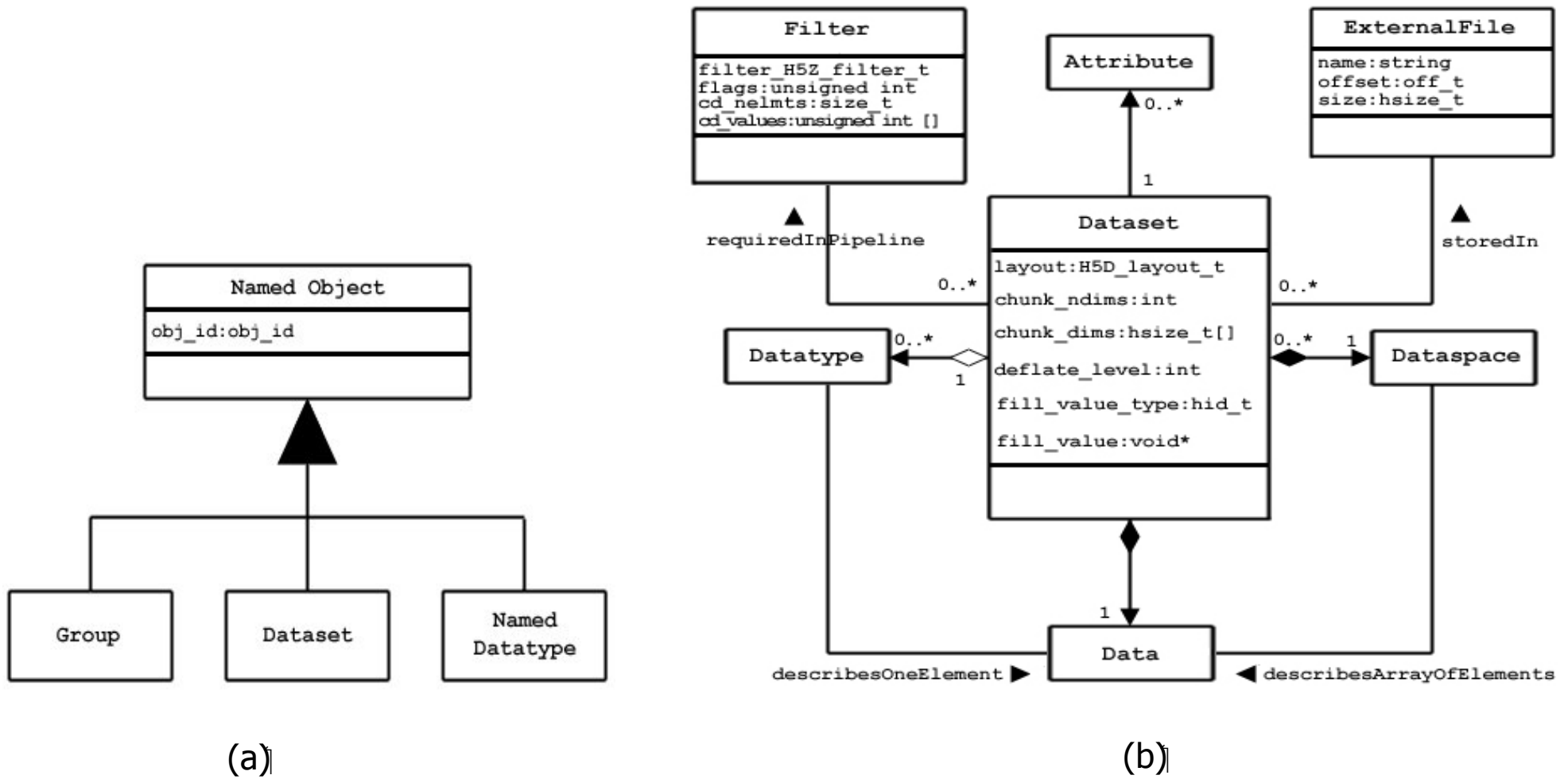


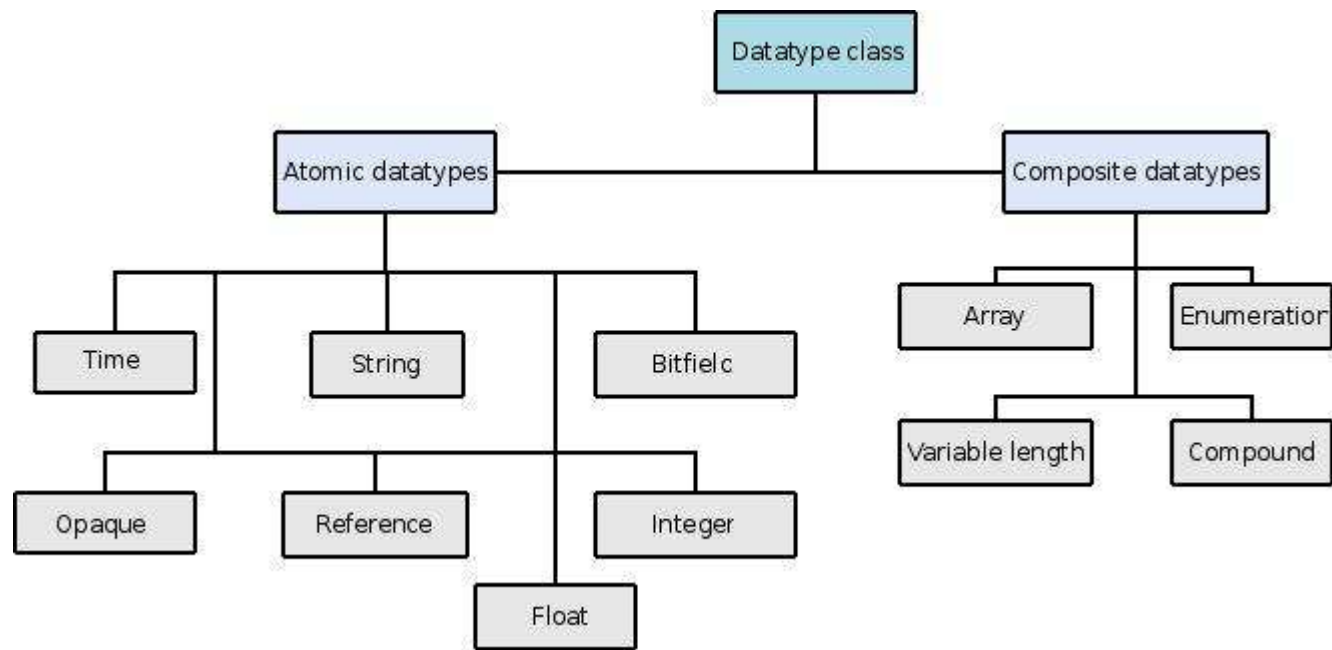
Figure 5. (a) Types of named objects and (b) relationship among HDF5 concepts that form a Dataset

# HDF5: object attributes and property lists

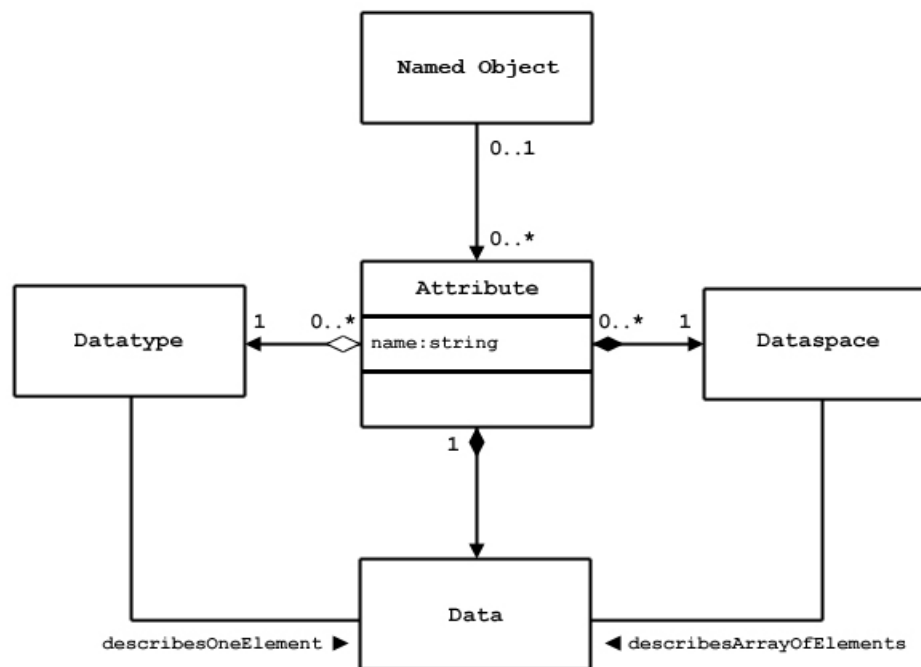
- HDF5 Named objects may have zero or more user defined Attributes, which are used to document the objects
- Attributes of an object are stored with the object and have a name and data
- Data is described in the same way as Dataset
- Conceptually an Attribute is similar to a Dataset apart from the following limitations:
  - An attribute can only be accessed via the object it belongs to and doesn't bear any meaning outside its scope
  - An Attribute should not be more than 1000 bytes
  - Attributes themselves can not have attributes
- Attributes of the object can be accessed by *name*, *index* or by *iterating through*
- *Property Lists* are conceptually equivalent to Attributes. They store information **relevant to the library**
- *Property Lists* control file creation, access and mounting, dataset creation and transfer behaviors

# HDF5: datatypes

- There are two different concepts of data types in HDF5:
  - Atomic datatypes – indivisible, a single object: number, string etc
  - Composite datatypes – composed of multiple atomic datatype elements



(a)



(b)

Figure 6. (a) Data types in HDF5 and (b) Organization of HDF5 object attribute

# HDF5: file format concepts

- HDF5 format defines:
  - how the object of the Abstract Data Model are mapped to the *linear address*
  - A number of objects for managing the storage, including, *header blocks, B-trees, and heaps*
- The specification can be organized in 3 levels:
  - **Level 0:** File Signature and Super Block - defines the header block for the file, a signature, version info, VFL driver
  - **Level 1:** File Infrastructure - defines the data structures used throughout the file: the B-trees, heaps, and groups.
  - **Level 2:** Data Object - defines the data structure for storing the data objects and data: Data Object Headers (metadata), Shared Data Object Headers (metadata), Data Object Data Storage

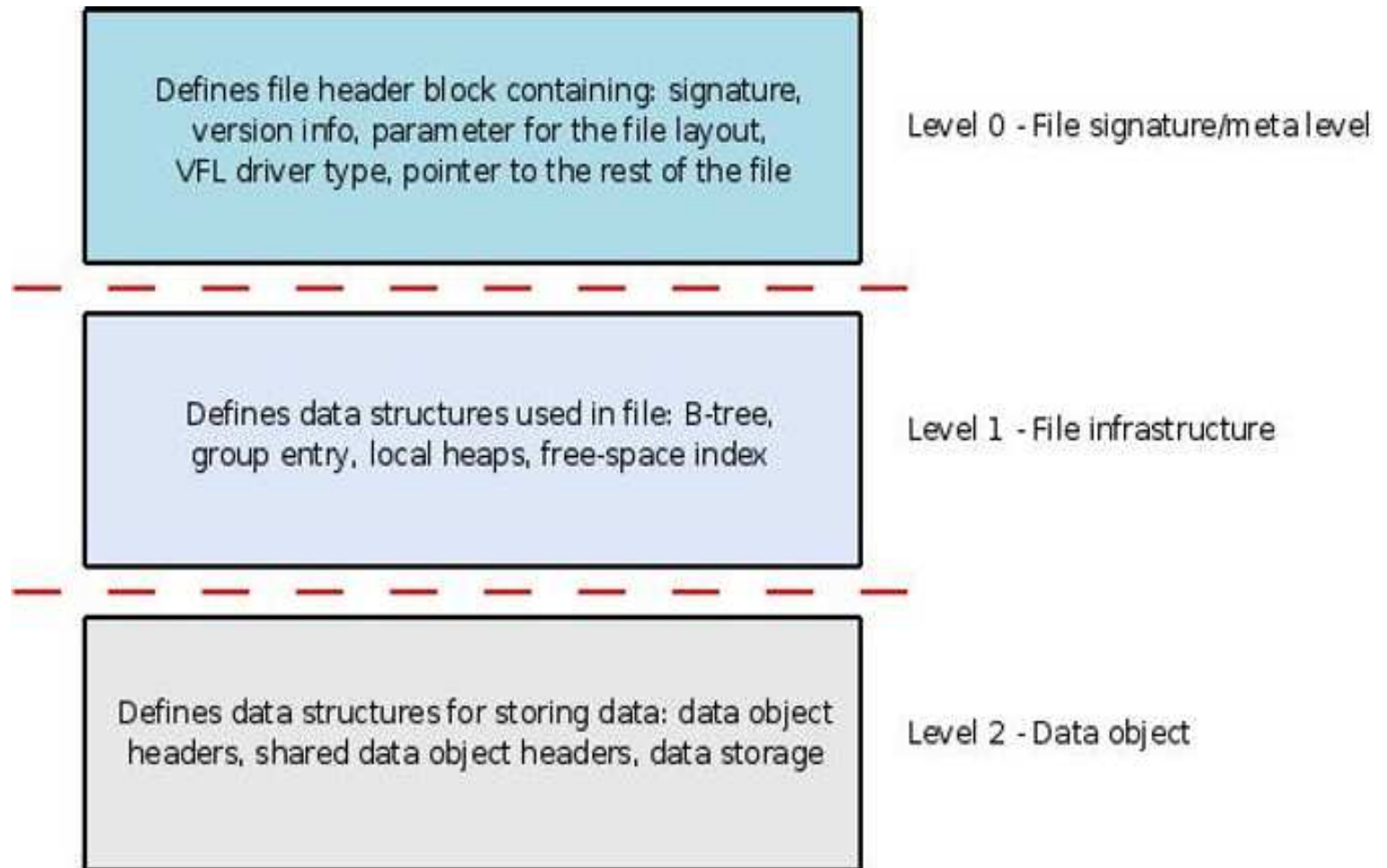


Figure 7. HDF5 file organization

# HDF5: virtual file layer

- VFL is an open interface that allows different concrete storage models to be selected.
- VFL defines an abstract model and API for random access storage, an API to plug in alternative VFL drivers.
- HDF5 library defines 6 different VFL drivers:

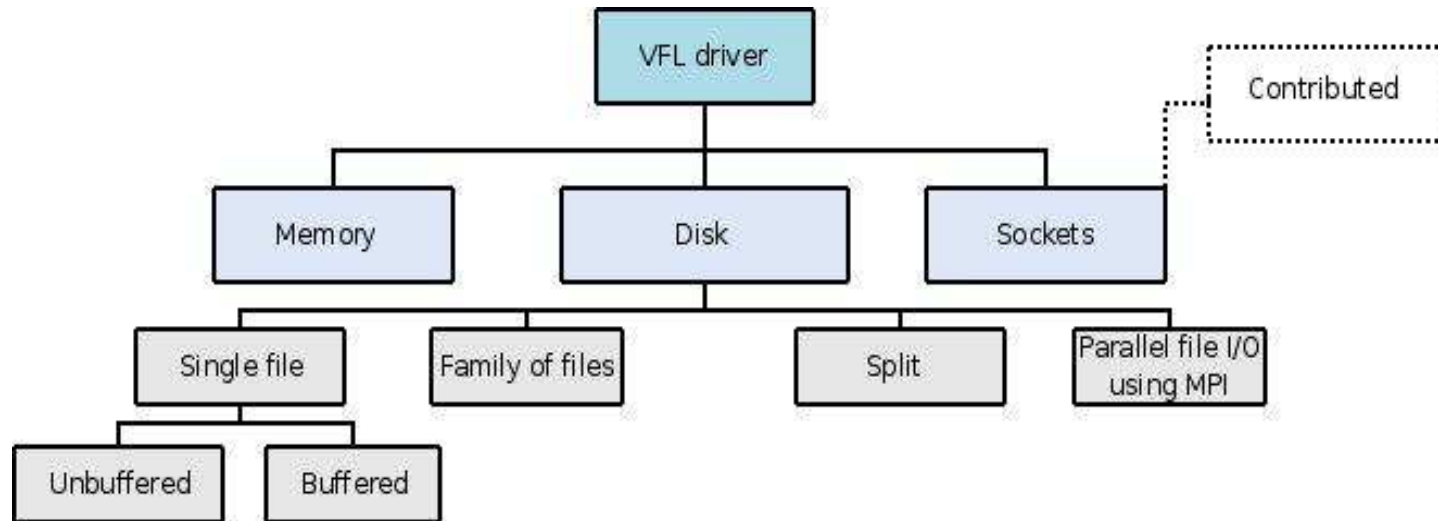


Figure 8. Conceptual hierarchy of HDF5 VFL drivers

# HDF5: virtual file layer

- HDF5 library impl. data transfer between different storage media
- Library reads/writes using VFL drivers, manages caches of metadata, data I/O pipeline including compression and transfer of data

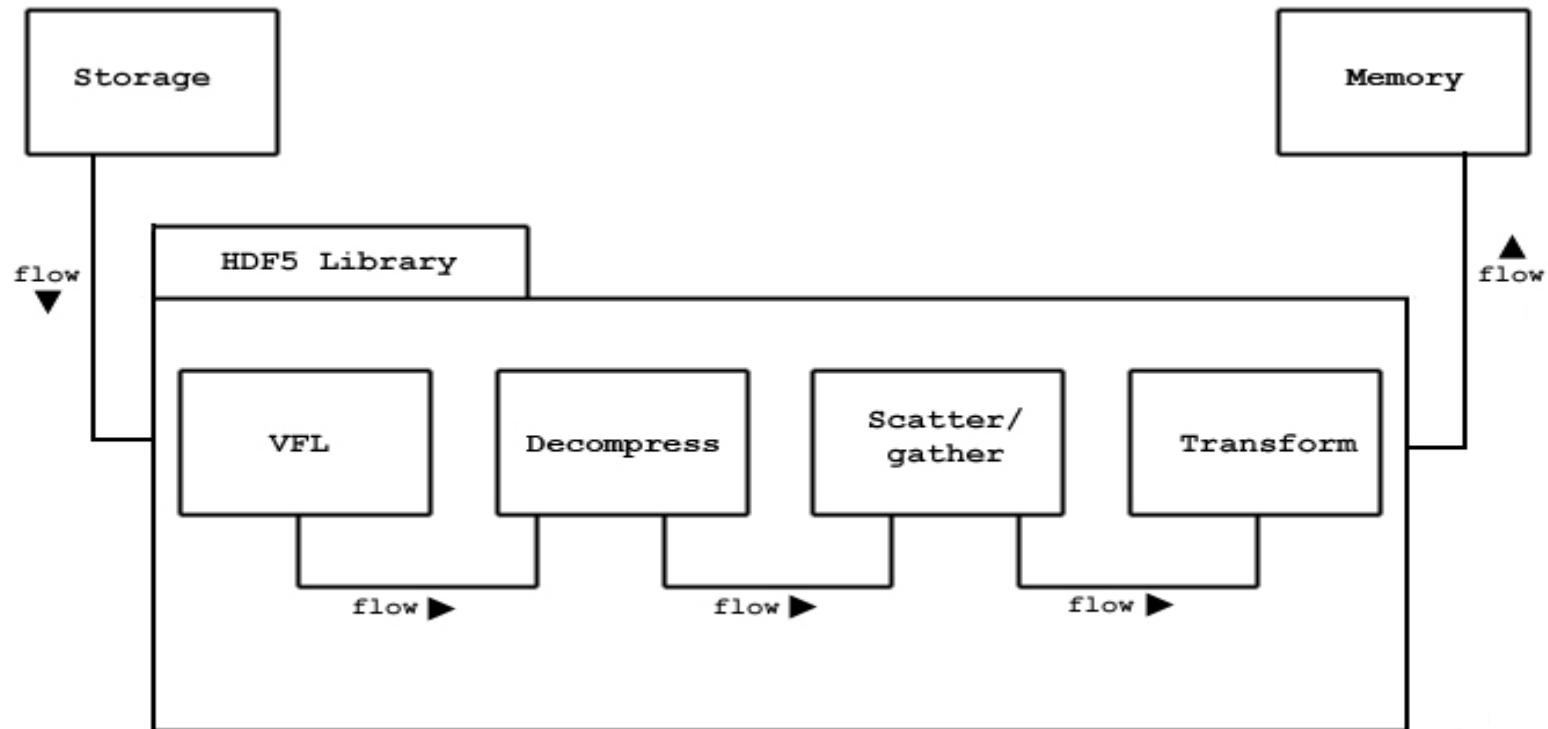


Figure 9. Example for HDF5 data transfer pipeline

# HDF5: APIs and tools

- HDF5 library is implemented in portable C and provides C++, Java, Fortran90 APIs
- Provides Data Description Language (similar CDFML, NcML & CDL)
- Conversion software, browsing and editing software (HDFView), Web-browser plugin
- HDF5 is not multithreaded, though thread safe (using PThreads)
- Utilities:
  - gif2h5 - Converts a GIF file into HDF5.
  - h5import - Imports ASCII or binary data into HDF5.
  - h5diff - Compares two HDF5 files and reports the differences.
  - h5repack - Copies an HDF5 file to a new file with or without compression/chunking.
  - h5cc, h5fc, h5c++ - Simplifies compiling an HDF5 application
  - h5debug - Debugs an existing HDF5 file at a low level.
  - h5stat - Displays object and metadata information for an HDF5 file

netCDF

netCDF: concepts and data model

# netCDF: intro

- Network Common Data Form is a data model and a library
- It allows to manipulate array-oriented data in a self-describing and portable manner
- Array values may be accessed directly without the knowledge how the actual data stored. Metadata is stored with the data.
- NetCDF physical representation of data is platform independent
- NetCDF supports 3 different file formats:
  - netCDF classic binary format
  - netCDF 64-bit offset format
  - netCDF-4 or HDF5 format (can be understood by netCDF only)
- To achieve network transparency netCDF classic and 64-bit formats use representation similar XDR

# netCDF: data model

- a NetCDF dataset contains dimensions, variables and attributes
- All 3 components have name and an ID number by which they can be identified
- NetCDF library allows simultaneous access to multiple netCDF datasets which are identified by the dataset ID
- NetCDF-4 format supports expanded data model, which includes named groups (from HDF5)
- Each group acts as an entire netCDF dataset in the classic model
- Each group may have attributes, dimensions, variables and other groups
- Dimensions are shared between variables in child groups if they are defined in parent group
- NetCDF-4 also supports atomic and compound data types as defined in HDF5
- Variables, groups and types share a namespace, thus within the same group they should have unique names

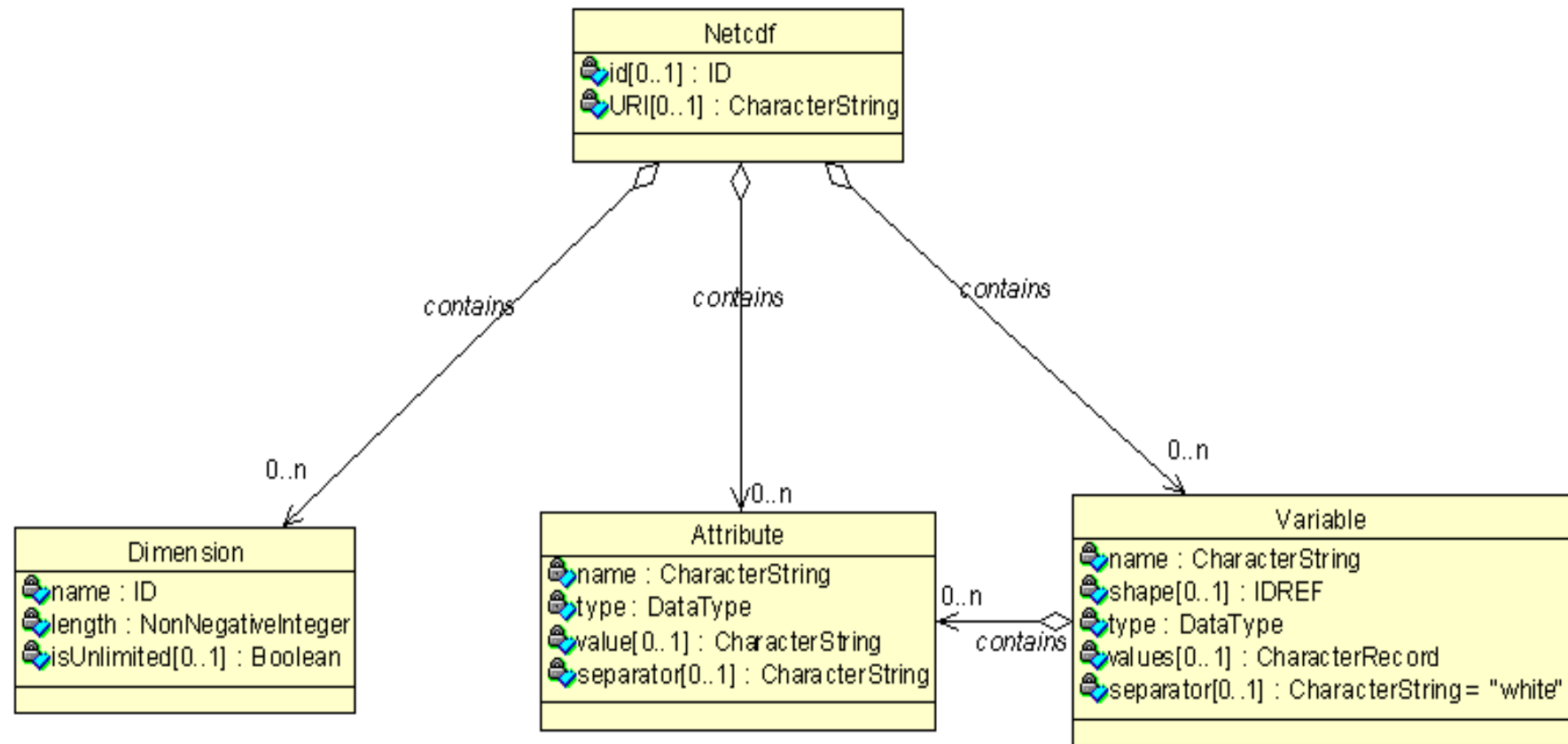


Figure 10. netCDF data object model

# netCDF: data access

- Access to data in classic and 64-bit offset format is direct
- Access to data in netCDF-4 format is buffered by the HDF5 layer
- To read a small subset of data one doesn't need to read the whole dataset
- In C and Fortran interfaces repetitive datasets access is performed through dataset Ids. The same holds for variables
- NetCDF interface supports several types of direct access
  - Access to all elements
  - Access to individual elements, specified with an index vector
  - Access to array sections, specified with an index vector, and count vector
  - Access to sub-sampled array sections, specified with an index vector, count vector and stride vector
  - Access to mapped array sections, specified with an index vector, count vector, stride vector and an index mapping vector

# netCDF: CDL

- NetCDF uses CDL (network common data form language) to describe its datasets
- CDL is accompanied by several utilities which help
  - to generate netCDF datasets
  - to extract meta information from a dataset
  - to generate C and Fortran sources that can produce netCDF dataset
- CDL description is composed of 3 components:
  - Dimensions – declares netCDF dimensions
  - Variables – declares netCDF variables and attributes
  - Data – declares variable data

# netCDF: CDL dimensions and variables

- CDL dimensions are used
  - to represent a real physical dimension: time, latitude, height ..
  - to index other quantities: station or model-run-number
- A dimension has both a name and a length (arbitrary positive integer)
- A variable with UNLIMITED dimension can grow to any length along that dimension
- A variable is used to store the bulk of the data in a netCDF dataset.
- A variable represents an array of values of the same type
- A variable has a name, a type, attributes(optional) and a shape defined by its dimensions
- netCDF classic and 64-bit offset formats support six types: byte, character, short, int, float and double
- netCDF-4 supports arrays of variable length, user defined compound types

# netCDF: CDL attributes (metadata)

- NetCDF attributes store data about the data
- Attributes provide info about specific variables or a dataset (global attributes)
- Attributes are identified by a name/ID of a variable and attribute name
- Global attributes are identified together by the attribute name and a blank variable name in CDL and null global variable in C and Fortran
- Attribute has an associated variable, a name, a data type, a length, and a value. All attributes are treated as vectors
- Attributes can have all types supported by netCDF for external data
- Attributes are more dynamic than variables and dimensions: they can be deleted or modified after being created
- Attributes can at most be one dimensional
- Attributes can not have attributes themselves

# netCDF: technical details and programming tools

# netCDF: file format (1)

- NetCDF datasets are stored as single file including both meta-data and data
- NetCDF classic and 64-bit offset dataset file is composed of two parts:
  - A header – contains all the information about dimensions, attributes, and variables, including their names, types and other charac.
  - A data section – contains fixed-size data, non-unlimited dimension variable data, unlimited dimension variable data
- Both the header and data parts of the file are encoded using extended XDR

## netCDF: file format (2)

- NetCDF 64-bit supports files bigger than 2GB on LFS platforms
- NetCDF-4 files are HDF5 files in every way
  - Groups in netCDF-4 file correspond to HDF5 groups
  - Variables and attributes correspond to identically named HDF5 datasets and attributes
  - In netCDF-4 special datasets are used to hold metadata
- NetCDF-4 uses HDF5 parallel programming model
- NetCDF-4 supports any file, variable sizes

# netCDF: utilities, APIs and other SW

- The 4 main utilities accompanied with netCDF and File Array Notation packages
  - ncdump – reads a netCDF dataset and prints a textual representation of the information in the dataset
  - ncgen – reads a CDL representation of a netCDF dataset and generates corresponding binary file or a C/Fortran sources to create the dataset
  - ncmeta – prints selected metadata from one or more netCDF datasets
  - Ncrob – performs various operations (copy, sum, mean, max, min ...) with data read from/written to text files/or selected parts of netCDF variables or attributes
- It comes with C, Fortran, C++, Java APIs and libraries
- Other interfaces include MATLAB, Octave, Objective-C, Perl, Python, Tcl/Tk etc

# netCDF: NcML description language

- NcML is an XML representation of netCDF metadata,
- NcML is similar to the netCDF CDL (network Common data form Description Language), except, it uses XML syntax.
- It is a work in progress and has support:
  - In NetCDF-Java library (version 2.2 and above), the NcML Core Schema and NcML Dataset have been combined into a single NcML-2.2 Schema
  - NcML-Gml is an extension of NcML core schema, based on GML grammar. It uses both NcML and GML to create a bridge to GIS Systems.

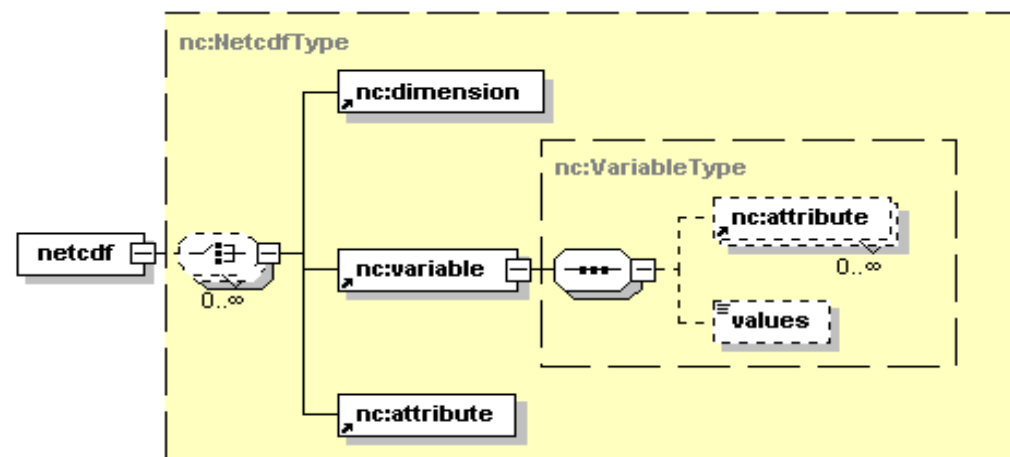


Figure 11. netCDF Meta Object Model

**XDMF**

# XDMF: concepts and data model

# XDMF: concepts (1)

- XDMF defines two types of data:
  - Light data – is description of the data (metadata)
  - Heavy data – refers to data itself (values)
- These two types are stored using different mechanisms
  - Light data uses XML format
  - Heavy data uses HDF5 format
- Calculated, time varying data values are described as attributes of the grid
- Storing meta information using XML allows tools to determine required resources for heavy data
- In this model HPC data is viewed as hierarchy of Domains.
- A domain must contain at least one grid
- A Grid is the basic representation of both the geometric and measured values.
- Grid is the group of Structured and Unstructured elements which is described by its topology and geometry

## XDMF: concepts (2)

- Grid may have one or more Attributes
- Attributes store any other value associated with the grid
- They may reference to individual cells of the Grid
- In addition to raw values, data can refer to Format (array dimension) or Model/Semantics (how the data is to be used, xyz coordinates via Vector components)
- XDMF view XML as a “personalized HTML” with some special rules.
- It is made of three components: elements, entities and processing
- These elements are represented in the form of <tag>CDATA</tag>
- CDATA is typically where the values are stored
- Any XDMF element can have a *Name* and *Reference* attributes
- XDMF elements contain one or more *Domain* elements
- *Domain* can have one or more *Grid* elements and each *Grid* contains a *Topology*, *Geometry* and zero or more *Attribute* elements

# XDMF: "DataItem" (1)

- Topology specifies the connectivity of the grid
- Geometry specifies the location of the grid nodes
- Attributes specify values such as scalars and vectors that are located at the *node, edge, face, cell, center* or *grid center*
- To specify actual values for topology, geometry and attributes XDMF defines a *DataItem* element
- DataItem provides actual values or the physical storage (HDF5 files). They specify the **format** of the data
- There are six types of DataItems:
  - Uniform – specifies a single array of values (this is default for DataItem type)
  - Collection – a one dimension array of DataItems
  - Tree – a hierarchical structure of DataItems
    - continued on the next page

## XDMF: "DataItem" (2)

- HyperSlab – contains two DataItems, the 1<sup>st</sup> specifies start, stride and count indexes of the 2<sup>nd</sup> DataItem
- Coordinates – contains two DataItems, the 1<sup>st</sup> selects the parametric coordinates of the 2<sup>nd</sup> DataItem
- Function – calculates an expression
- Below is an example of data stored in 3 dimensional array of pressure values in HDF5

```
< DataItem ItemType= 'Uniform' Format= 'HDF' NumberType= 'Float'  
Precision= '8' Dimensions= '64 128 256'>  
    OutputData.h5:/ Results/ Iteration 100/ Part 2/ Pressure  
</DataItem>
```

# XDMF: "Grid"

- Grid element conveys information on the model of the data
- It is a container for the information on 2D/3D points, structured/unstructured topology and assigned values
- There are 4 types of Grids:
  - Uniform – a homogeneous single grid (i.e. Pile of triangles). It must contain a Topology and Geometry elements
  - Collection – an array of Uniform grids all with the same Attributes. These can be specified as Spatial or Temporal
  - Tree – a hierarchical group
  - SubSet – a portion of another Grid

# XDMF: "Topology" (1)

- The Topology describes the general organization of the data (connectivity)
- It is the part of the computation grid that is invariant with rotation, translation and scale
- It allows to specify an Order for unstructured grids.
- The following types of the ordered cells are supported:
  - Linear
    - Polyvertex – a group of unconnected points
    - Polyline – a group of line segments
    - Polygon, Triangle, Quadrilateral, Tetrahedron, Pyramid, Wedge, Hexahedron
  - Quadratic
    - Edge\_3 – quadratic line with 3 nodes
    - Tri\_6, Quad\_8, Tet\_10, Pyramid\_13, Wedge\_15, Hex\_20
  - Arbitrary
    - Mixed – amixture of unstructured cells

# XDMF: “Topology” (2)

- The following types of the ordered cells are supported:
  - Structured
    - 2DSMesh – curvilinear
    - 2DRectMesh – Axis are perpendicular
    - 2DCoRectMesh – Axis are perpendicular and spacing is constant
    - 3DSMesh, 3DRectMesh, 3DCoRectMesh

# XDMF: "Geometry"

- The Geometry element describes the XYZ values of the mesh
- The important attribute for this element is the organization of the points. The possible values are:
  - XYZ – interlaced locations
  - XY – Z is set to 0.0
  - X\_Y\_Z – X, Y and Z are separate arrays
  - VXVYVZ – three arrays, one for each axis
  - ORIGIN\_DXDY\_DZ – six values: Ox, Oy, Oz + Dx, Dy, Dz

# XDMF: "Attribute"

- The Attribute element defines values associated with the mesh
- It supports the following types:
  - Tensor – 9 values expected
  - Tensor6 – a symmetrical tensor
  - Matrix – an arbitrary NxM matrix
  - Scalar, Vector
- These values can be centered on:
  - Node, Edge, Face, Cell, Grid
- A Grid centered Attribute might be something like "Material Type" where the value is constant everywhere in the grid
- Typically Attributes are assigned on the Node

# XDMF: "Time"

- The Time element is a child of the Grid element and specifies temporal information for the grid
- There are 4 different formats in which time can be specified
  - Single – a single time value for the entire grid
  - HyperSlab – Start, stride, count
  - List – a list of discrete times
  - Range – min, max
- Below one can see two examples showing range and hyperslab formats

```
<Time TimeType="Range">  
  <DataItem Format="XML" NumberType="Float" Dimensions="2">  
    0.0 0.5  
  </DataItem>  
</Time>
```

## Range format

```
<Time TimeType="HyperSlab">  
  <DataItem Format="XML" NumberType="Float" Dimensions="3">  
    0.0 0.000001 100  
  </DataItem>  
</Time>
```

## HyperSlab format

# XDMF: "Information"

- The Information element defines system or code specific information which does not map to the used data model
- It can address several system specific items like time, units, descriptions, etc.

# XDMF: Technical Details and Programming Tools

# XDMF: tools and APIs

- XDMF uses two major extensions to XML:
  - XInclude allows inclusion of files that are not XML (can be XML or just plain text)
  - XPath allows for elements in the XML document and API to reference specific elements in a document: `/Xdmf/Domain/Grid[@Name="Copper Plate"]`
- All valid XDMF must appear between `<Xdmf></Xdmf>`
- XDMF DTD and Schema exist but are only necessary for parser validation
- XDMF comes with C++ API, this API is also wrapped in Python, TCL and Java.
- API is not necessary to read/write XDMF data, it can be performed via ICE environment

# XDMF: Interdisciplinary Computing Environment (ICE)

- **ICE** is an effort provide a common software platform for Scientific Codes in a heterogeneous High Performance Computing environment
- It includes the **XDMF** as a common data hub where HPC codes and tools can efficiently exchange data values and meaning
- It also includes Visualization and Graphical User Interface tools.
- ICE defines distributed data buffer called "*Network Distributed Global Memory*" and located on top of OS.
- HDF5, XML and a convenience layer are used to add structure this buffer.
- Together, these layers comprise the XDMF.

# Summary

Name	Data model	Meta-data support	Data access mode	Storage model/data type support	Platform and API
<b>CDF</b>	Data is stored in arrays and composed of two parts: data values and attributes	Two types of meta-data: global attributes describing dataset and local attributes describing single variables. Provides data description language, CDFML	Four types of data access modes: single value, hyper access, sequential, Multiple variable	Data can be stored either in single file or in multiple files/data types supported are integers, floats, char and epoch equivalent time	C, Java and Fortran APIs, interfaces to MATLAB, IDL, FlexPro Supported on many platforms including windows and POSIX compliant
<b>HDF5</b>	Data is stored in arrays composed of value and attributes. It is organized in graph tree, all data descending from root group	Two types of meta-data: attributes for datasets, data types and groups and global meta-data, property lists, describe the library or application. Provides HDF5 data description language	Partial I/O (sub-setting, sampling, scatter-gather), hyperslab selection, point selection	Memory, buffered/unbuffered, family of files, split, parallel file IO and sockets/two main categories of data: atomic (string, time, int, float etc) and compound data types (array, enumeration, user defined)	C, C++, Java Fortran90 APIs, interface to MATLAB. Supported on many platforms including windows and POSIX compliant.
<b>netCDF</b>	Data is stored in arrays. It is described using attributes and dimensions. Both data values and attributes are stored together	Two types of meta-, data: variable(local) and dataset(global) attributes. Provides two data description lang. CDL and NcML	Access to all elements, to individual elements through index values, to sub-sampled array sections, to mapped array sections	All data is stored in single file/NetCDF-4 supports all data types supported by HDF5, netCDF classic and 64 bit support six basic types: char, byte, short, int, float and double	C, Fortran, C++, Java APIs, interfaces to MATLAB, Octave, Perl, Python, TCL/Tk. Supported on many platforms including windows and POSIX compliant.
<b>XDMF</b>	Defines data as a collection of nodes which form a grid. One can specify grid topology, geometry and attributes of the nodes. The real data is stored in HDF5.	All the meta-data is stored in XML format. It adopts the model of grid network. Each node is a datum . One can specify attributes of these grids and nodes.	Follows HDF5 model, meta-data access depends on the parser implementation	Data is stored in 2 separate files: XML file for meta-data and HDF5 for heavy data/data supported by HDF5	C++ API, also wrapper for the API in Python, Java and TCL. XDMF is the part of Interdisciplinary Computing environment (ICE). Supported on POSIX compliant and windows systems.