CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Effective IO (best practices)

John Biddiscombe
CSCS, 19/03/2013

With many slides taken from The HDF Group presentations.

Objectives

- **Learn something about the options available**
  - Some of the formats that exist already (Scientific anyway)
  - Some of the APIs (HDF5)

- **Good and Bad ways of storing data**
  - Random access
  - Serialization/Parallelization

- **Portability**
  - Ensure that others can read your data
  - Not machine/language specific
  - Especially Fortran programmers!

Representation

- **Binary vs Text/Human readable**
  - Who is your data for?
    - Scripts (text/csv)
    - (machine) Analysis/Visualization (binary)
    - Statistics
    - Queries (SQL)
    - Information
    - Key Value storage
  - Validation
    - How Import is the data
  - Is final size important
    - Humans don't read GB/TB
    - ASCII generally ends up >3 times less efficient (s.p.)

Lessons learned from experience

- **Use a well defined API**
  - Because then it's easy for others to access your data

- **Self describing formats are 'in'**
  - What do we mean by self-describing anyway
    - (Internal) Metadata
  - They all come provided with an API already
  - HDF5, netCDF, XML, SQL database tables
  - Home grown formats are not 'in'

- **One file with everything** *'you'll thank me later'*
  - Or at least one per time step
  - Because lots of small files get lost, ***or copies diverge***
  - Because *someone* ***can edit parts of files*** and ruin provenance

- **Headers/Metadata for pre-analysis browsing**
  - So the user knows what's there before loading it all
  - So the reader can be as clever as possible

Metadata

- **Data that describes the rest of the data**
  - Allows sizes/types/internal storage format to be 'known'
    - This is all handled for us by the API
- **Conventions**
  - Are really all about the metadata and the API
  - Allows **true sharing** of data between **communities**
  - Think about the problems that the people like the IPCC have
    - N institutions
    - M simulations
    - X parameter studies
    - Y models
- **Provenance**
  - Will you need the data in N years
  - Do you/others need to regenerate the data in part or even full

More or Less

- **Disk space is basically cheap**
  - Unless you're a climate scientist (Or other big supercomputer user)
  - *But memory isn't so readily available*
    - *don't assume all can be loaded in one go*
    - *avoid forcing decision making after load*
    - *This is why metadata/headers/info is great*

- **It's easy to delete things later**
  - But harder to decide what's important enough to keep

- **Provenance**
  - In Supercomputing …
  - Rerunning a simulation is becoming 'easier' than storing everything
  - So metadata about the simulation is important
  - From compile options to input parameters

A terrible file format

- **Fluent (CFD solver) CASE files**
  - A series of TAGS followed by data lists
  - Lists may have sub tags
  - Minimal information about list sizes
  - Random access practically impossible
  - Have to read everything to get the small bit you want

  - Parallel code can’t make best use of resources.
    - You could parse the info once, build header tables and then indedx everything, but …

CASE Format example

This section indicates the pairings of periodic faces on periodic boundaries. Grids without periodic boundaries do not have sections of this type. The format of the section is as follows:

```
(18 (first-index last-index periodic-zone shadow-zone)( f00 f01 f10
f11 f20 f21 . . . ))

ex:.
(18 (1 2b a c) ( 12 1f 13 21 ad 1c2 . . . ))
```

The declaration section for cells is similar to that for nodes.

```
(12 (zone-id first-index last-index type element-type))

ex.
(12 (0 1 3e3 0))
```

More on bad file formats

- **Reasons for bad file formats**
  - Commercial advantage -Proprietary word/spreadsheet formats.
  - Some compression possibilities.
  - Support of clever algorithms (b-tree).
  - Close intertwining with existing code.

- **Consequences of bad file formats**
  - Forced use of provider's software.
  - Restriction on unexpected uses.
  - Binary formats are generally not editable by a simple standard means.
  - Patent lock-in (LZW vs gzip/flate in PDF).
  - Platform lock-in.

Sob story : Copying/Editing bits of files

A recent example

- **Simulation Inputs**
  - Bunch of files (many text/binary in combination)
  - Some quite large
  - Text files reference other input files (master input file)

- **Simulation Outputs**
  - Bunch of files (many text/binary in combination)
  - Text files reference some of the input files

- **Hand edits to output text files to correct paths/symlinks**
  - Only received a subset of the data
  - Crucial symlinked files missing
  - Received other files to replace them (edit ***this*** and ***that*** they say)
  - Nobody really knows if I'm using the right stuff
  - Wasted hours on crashes and dodgy plots

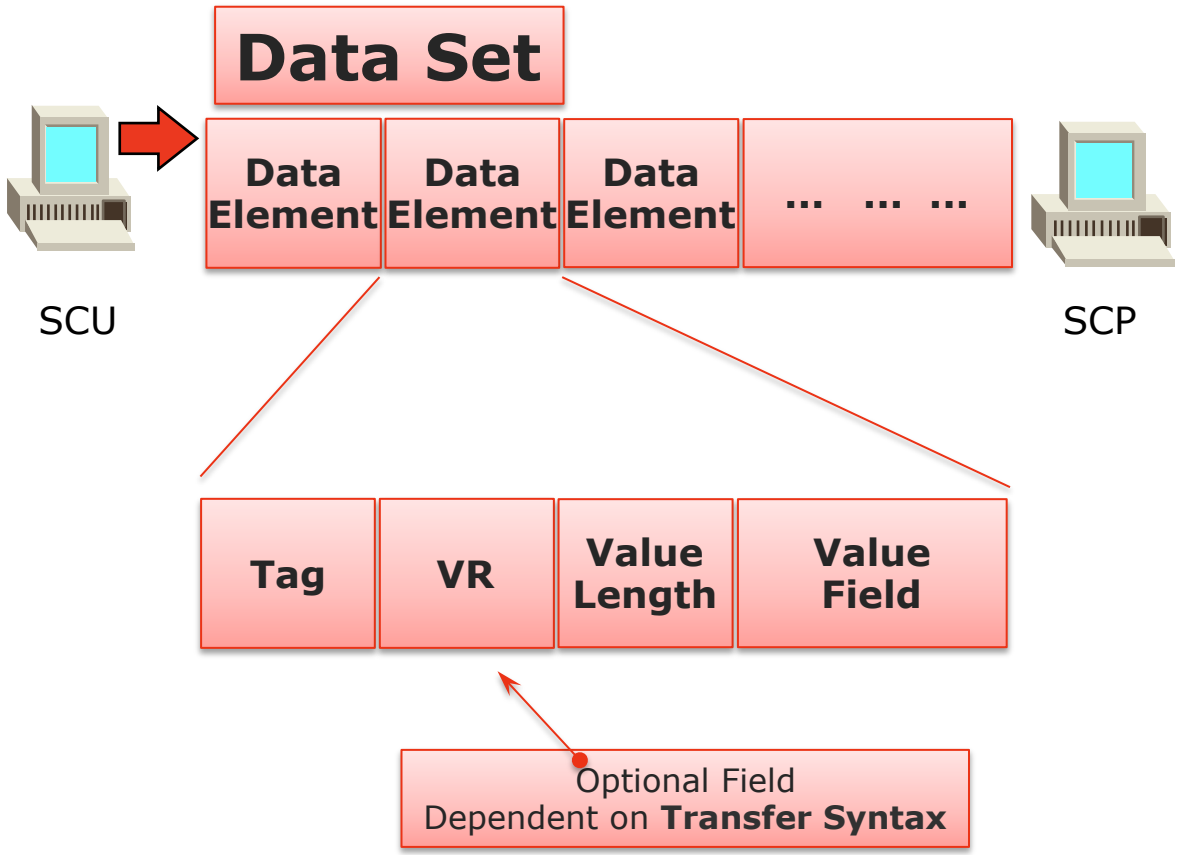
How to solve that problem

- **Missing symlinks caused by management practices**
  - Restricted access to data
  - Not much we can directly do about that
- **Resolving correct input/parameters**
  - Database
  - Key/ID ‘generation’ for known parameters
  - Maintenance/management nightmare
  - Adding new params?
- **Learn from established data generators**
  - Invite someone from CERN next time
  - Or invite me there for a visit!

Some formats for example

- **DICOM,**
  - Digital image communications in medicine
- **Used in every hospital (probably)**
- **Allows customization**
  - New imaging machines are always being built
  - Everything is a metadata tag really (and its hierarchical)
  - You can add anything, but use the Dictionary
  - A standard is available
- **Actually goes well beyond a format, but we won't worry about that.**
  - Covers network protocols as well, so machines can talk
- **It is shockingly complex and awful**
  - But it works
  - Standardization (and policing)
- **The file is really just another provider (API again)**

DICOM : Medical Imaging



Attributes are assigned a unique Tag (group + element)

tag=(0010, 0010)

groups deals with patient information for example

element is patient name for example

Attribute encoding is defined by Value Representation (VR).

Data Element Types

TYPE 1

Required Data Elements
Value Field shall not be zero length

TYPE 1C

Type 1C elements have the same requirements as Type 1 elements under certain specified conditions

Type 2

Required Data Elements
Value Field may be zero length

Type 2C

Type 2C elements have the same requirements as Type 2 elements under certain specified conditions

Type 3

Optional Data Elements

Example tags

Group / Element	Type	Attribute Name	Attribute Desc.
0008 , 0060	1	Modality	Orig. Modality
0020 , 2810	1	Rows	# of rows in image
0010 , 0010	2	Patient Name	Pat. Full Name
0010 , 0030	2	Patient Birthdate	Pat. Date of Birth
0010 , 1030	3	Patient Weight	Wt. of Patient kg
0010 , 2180	3	Patient Occupation	Occ. of Patient

Reality differs slightly

- **Manufacturers invent new stuff**
  - Add Tags anywhere they like
- **Publish vendor specific tags**
  - Users have to manage
- **Eventually new tags become part of standard**
  - Everyone is happy
  - Years of misery and broken files are forgotten
- **But more seriously**
  - Lives depend on the data
  - so irregularities are flagged/refused
  - Regulatory body issues new standards/tags/dictionaries

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

XML : As a Data File Format?

- **Surprisingly flexible**
- **Can be Validated**
- **Great for short documents**
- **Schema can be used to constrain 'types' of fields, attributes**
- **Can be used to provide 'metadata' along with heavy binary data**
- **Tree-like structures are easily represented**
  - Think multi-block data
- **VTK for example uses XML for in-house format**
  - Binary data base64 encoded and can be compressed (zlib)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

VTK XML, covers most of the bases

```
<VTKFile version="0.1" byte_order="BigEndian" type="PolyData" compressor="vtkZLibDataCompressor" >
  <PolyData >
    <Piece NumberOfStrips="0" NumberOfVerts="0" NumberOfPolys="12" NumberOfLines="0" NumberOfPoints="8" >
      <Points >
        <DataArray Name="Points" type="Float32" NumberOfComponents="3" format="appended" offset="0"/>
      </Points >
      <PointData >
        <DataArray Name="VertexNormals" type="Float32" NumberOfComponents="3" format="appended" offset="60"/>
        <DataArray Name="VertexColors" type="Float32" NumberOfComponents="4" format="appended" offset="136"/>
      </PointData >
      <Polys >
        <DataArray Name="connectivity" type="Int32" format="appended" offset="220"/>
        <DataArray Name="offsets" type="Int32" format="appended" offset="320"/>
      </Polys >
    </Piece >
  </PolyData >
  <AppendedData >
    _AQAAAACAAABgAAAAGQAAAA==eJxjYEAGDfaobHQ+ulqYGIyNKQYAP8sI9Q==
    AQAAAACAAABgAAAAJQAAAA==eJwzOytsb4bA+9HY6HwUtd9ZTEHYngEIkNWDx
    AQAAAACAAACAAAAAKwAAAA==eJwzNja2NwZiBjBoANEwDOMz4ODbI/Ghcjjlk
    AQAAAACAAACQAAAAOAAAAA==eJxNjFsKADAIw+r7/jeeY/mYECySKkm2OITe1
  </AppendedData >
</VTKFile >
```

Databases, Tables, SQL

- **Consist of several/many tables**
  - Data is stored as row/column
- **Databases are relational**
  - Relating information in one table to another
- **Downside is usually having to have a database 'server'**
  - configuration
  - Barrier to entry
- **Standalone operation possible**
  - SQLite
  - Qt has internal SQLite type data server

Table description

Field	Type	Null	Key	Default	Extra
job_id	varchar(32)	NO	PRI	NULL	
username	varchar(16)	YES		NULL	
group_id	varchar(16)	YES		NULL	
ctime	int(11)	NO	PRI	0	
start	int(11)	NO	PRI	0	
end	int(11)	YES		NULL	
mppwidth	int(11)	YES		NULL	
mppdepth	tinyint(4)	YES		NULL	
mppnppn	tinyint(4)	YES		NULL	
req_ncpus	smallint(6)	YES		NULL	
req_mem	bigint(20) unsigned	YES		NULL	
jobname	varchar(64)	YES		NULL	
is_pp	tinyint(1)	YES		0	
place	varchar(64)	YES		NULL	

Queries can be made from console or code

```
SELECT job FROM accounting WHERE user='biddisco'

std::string fullquery =
    std::string("SELECT ja.job_id, ja.create_time,
       ja.destroy_time, ya.yod_id, yapl.processor_id ") +
    std::string("FROM yod_accounting ya, job_accounting ja,
       yod_accounting_processor_list yapl ") +
    std::string("WHERE ja.create_time < ") + endstring +
    std::string("AND ja.destroy_time >= ") + startstring +
    std::string("AND ya.partition_id = ja.partition_id ") +
    std::string("AND ya.yod_id = yapl.yod_id ");
```

- **The Database server (API again) handles the searching**
  - Building indexes can be expensive
  - May depend on what type of query you plan
  - Indexes can be conditional, or user defined

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

SQL in use from Cray SDB

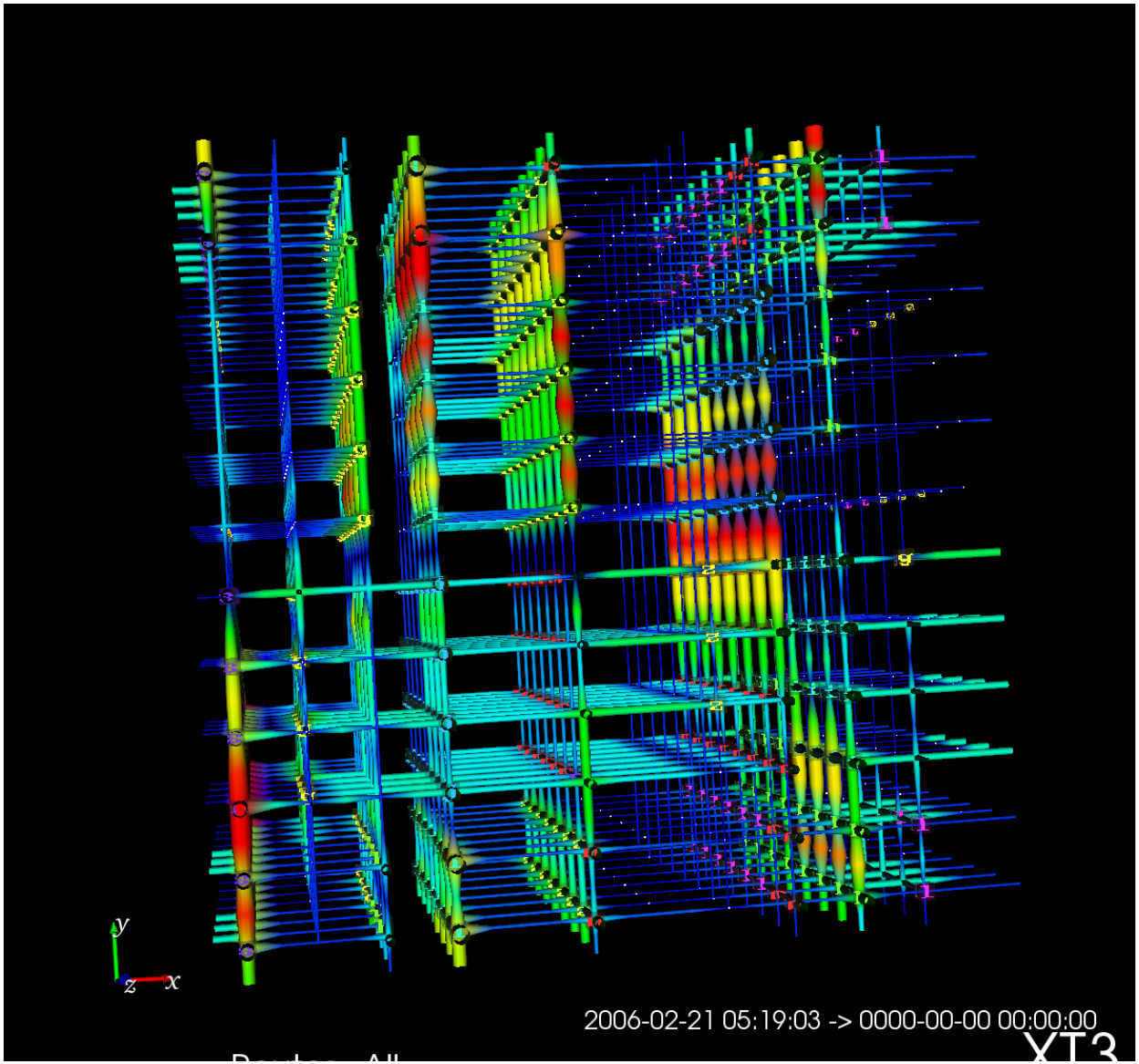
Topology from database

Jobs from Database

Routing from Database

Scientists are not usually big users of SQL type data, but it can be very convenient for structured or relational queries/data.

Not ideal for ‘large’ datasets



Key Value storage

- **The next big thing in 'Big Data'**
  - Sometimes known as NoSQL, but usually supports SQL
- **Removed the need for certain synchronization operations**
  - Size of 'value' part of data not defined in advance
  - Keys and values are just data arrays on unspecified type
- **All based on hashing of keys to find data**
  - Keys can map to multiple hashes
- **Often Used where fault tolerance is important**
  - Cloud computing for example
  - Amazon google ….
- **Might be interesting for asynchronous Parallel IO**
  - (Because hyperslabs/pieces are not united, see later)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

HDF5 – for all your data needs?

Over the last 8 years I've used it extensively (and even extended it)

Never really had any complaints

Written files of over 32TB in size (for fun mostly)
Would not recommend that for most usage

H stands for hierarchical
D for Data
F for Format
5 must be how many version they got through

See also, ADIOS, pnetCDF, netCDF5, more slides on those later

Following slides taken from hdfgroup presentations

What is HDF5?

- **File format for storing scientific data**
  - To store and organize all kinds of data
  - To share data , to port files from one platform to another
  - To overcome a limit on number and size of the objects in the file

- **Software/API for accessing scientific data**
  - Flexible I/O library (parallel, remote, etc.)
  - Efficient storage
  - Available on almost all platforms
  - C, F90, C++ , Java, Python (H5py) APIs
  - Tools (HDFView, utilities)

HDF5 file

- **Primary Objects**
  - Groups
  - Datasets

- **Additional means to organize data**
  - Attributes
  - Sharable objects
  - Storage and access properties

HDF5 Dataset

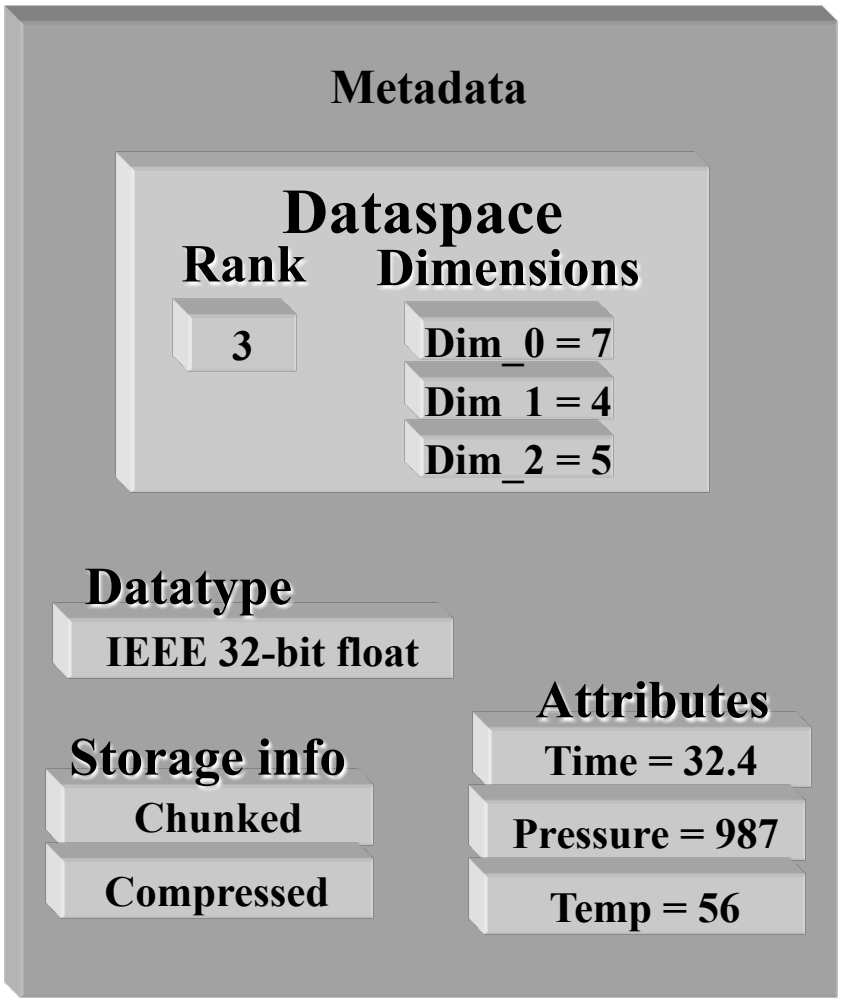
- **HDF5 dataset – data array and metadata**

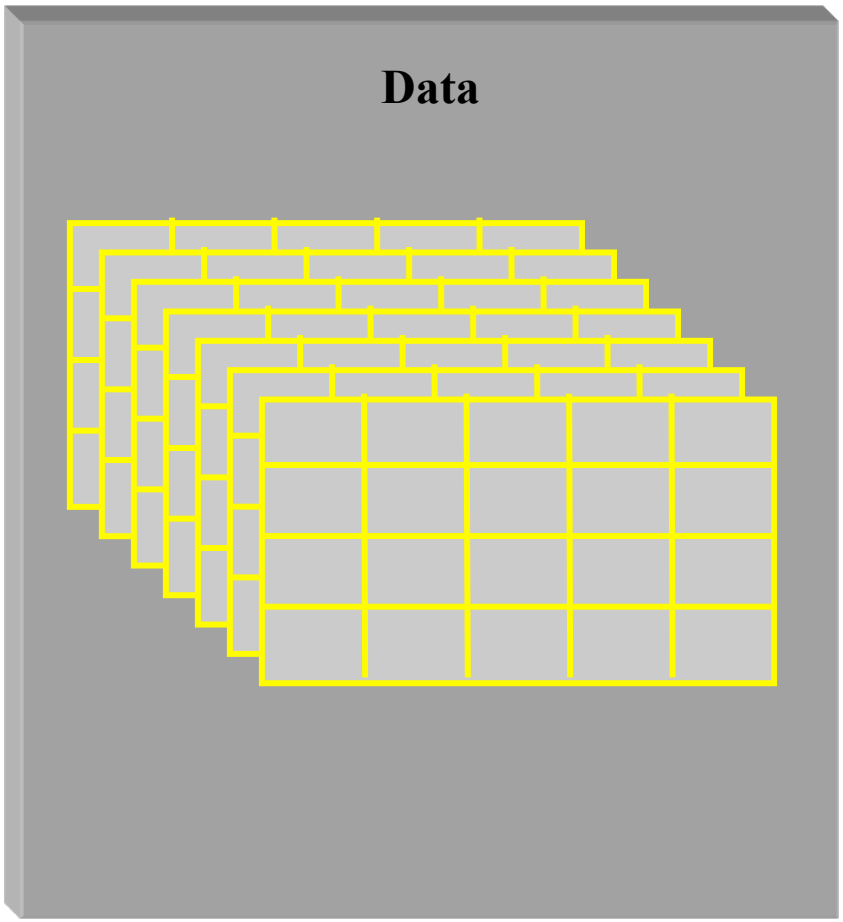
- **Data array**
  - ordered collection of identically typed data items distinguished by their indices

- **Metadata**
  - Dataspace
    - rank, dimensions, other spatial info about dataset
  - Datatype
  - Attribute list – user-defined metadata
  - Special storage options – how array is organized and stored in the file

Dataset Components

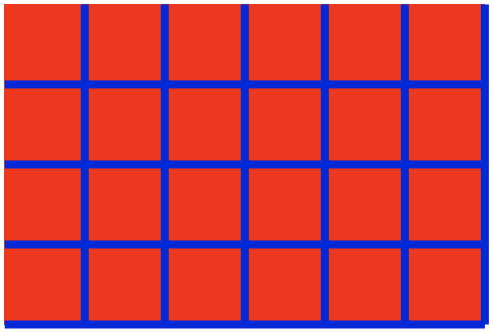
Example Matrix A(7,4,5)





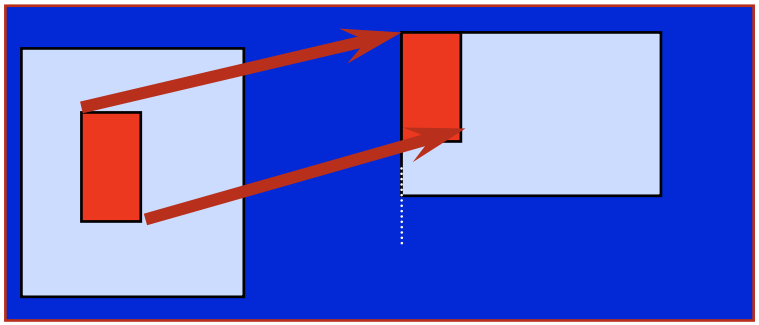
Dataspaces

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

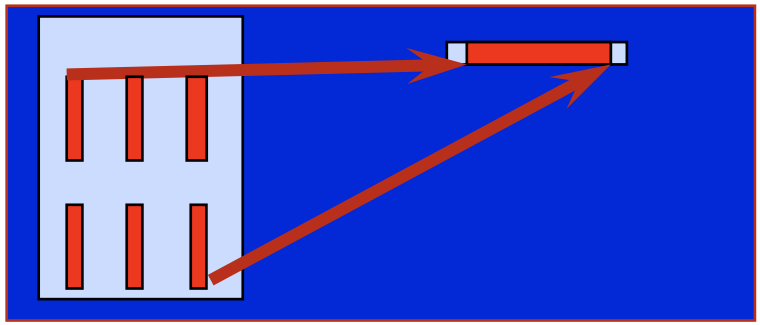


Rank = 2
Dimensions = 4x6

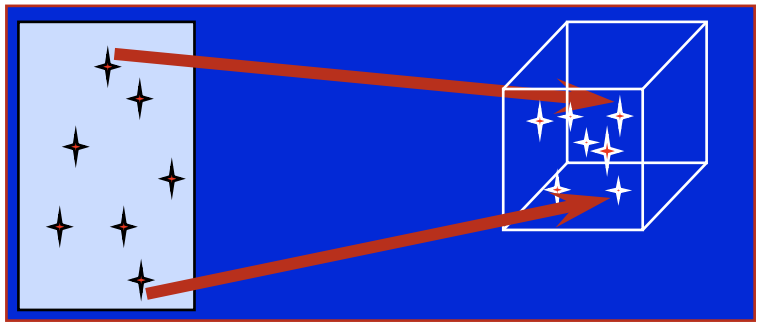
Mappings between File and Memory Dataspaces



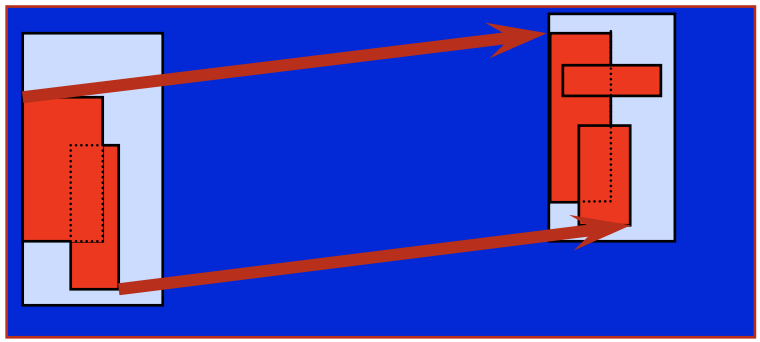
(a) Hyperslab from a 2D array to the corner of a smaller 2D array



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



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



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

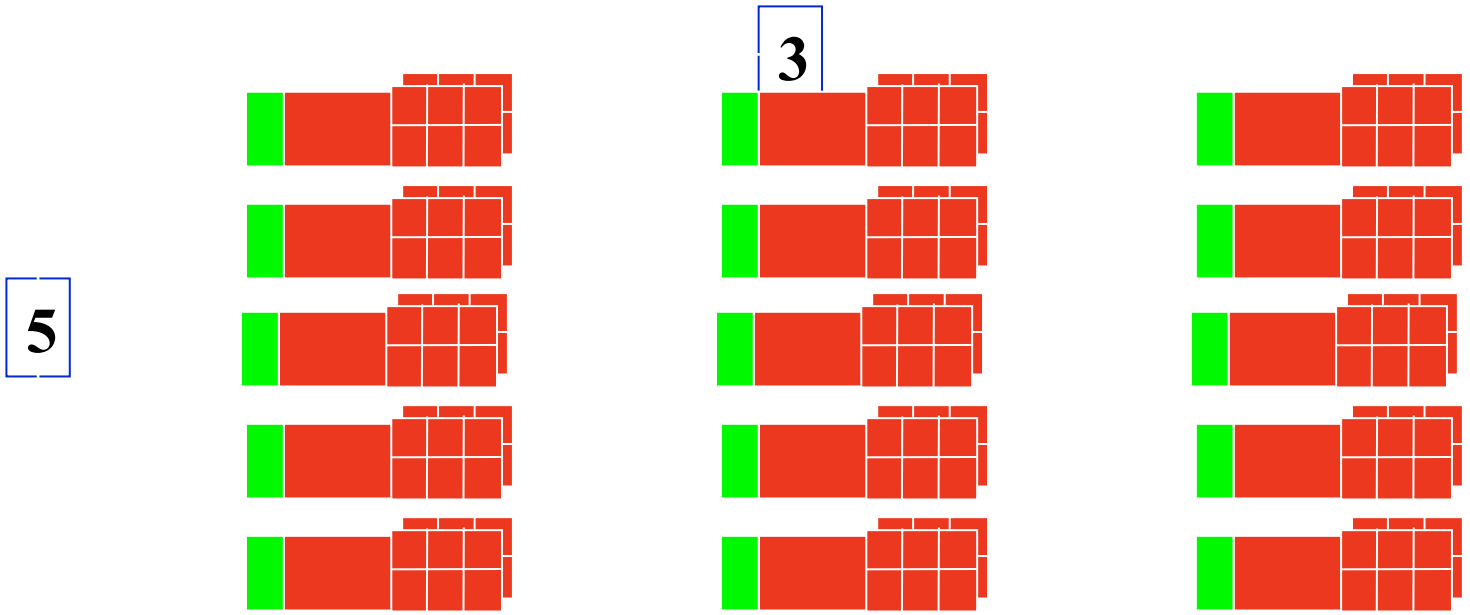
Datatypes (array elements)

- **Datatype – how to interpret a data element**
  - Permanent part of the dataset definition

- **HDF5 atomic types**
  - normal integer & float
  - user-definable integer and float (e.g. 13-bit integer)
  - variable length types (e.g. strings)
  - pointers - references to objects/dataset regions
  - enumeration - names mapped to integers
  - array

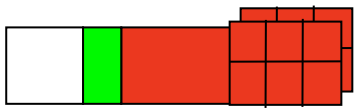
- **HDF5 compound types**
  - Comparable to C structs
  - Members can be atomic or compound types

HDF5 dataset: array of records



Dimensionality: 5 x 3

Datatype: int8 int4 int16 — 2x3x2 array of float32



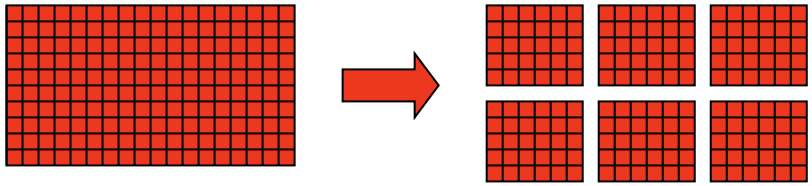
Record

Attributes

- **Attribute – 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
- **Optional for the dataset definition**
- **Can be overwritten, deleted, added during the "life" of a dataset**

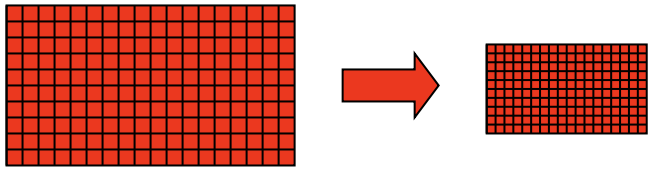
Special Storage Options

chunked



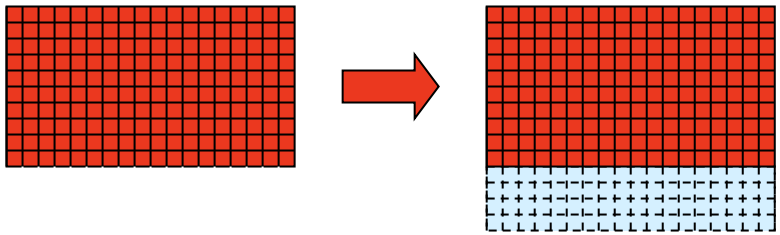
Better subsetting access time; extendable

compressed



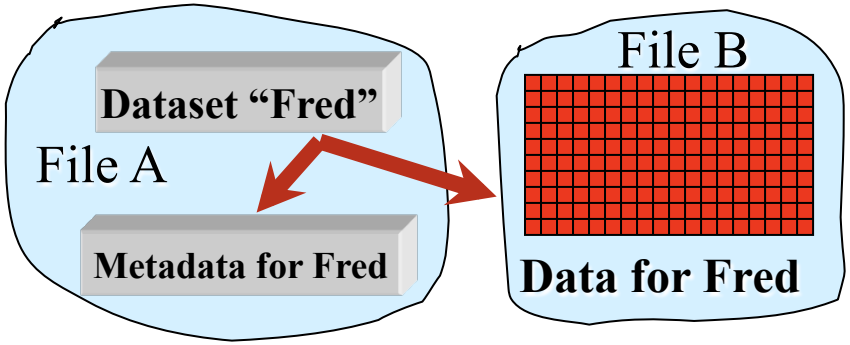
Improves storage efficiency, transmission speed

extendable



Arrays can be extended in any direction

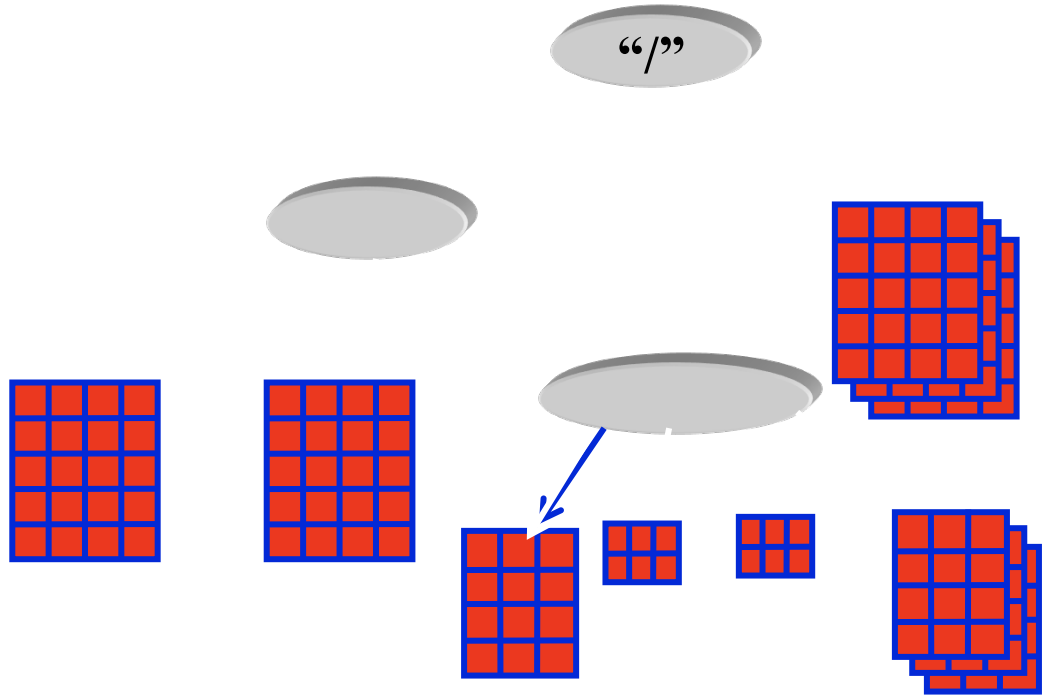
External file



Metadata in one file, raw data in another.

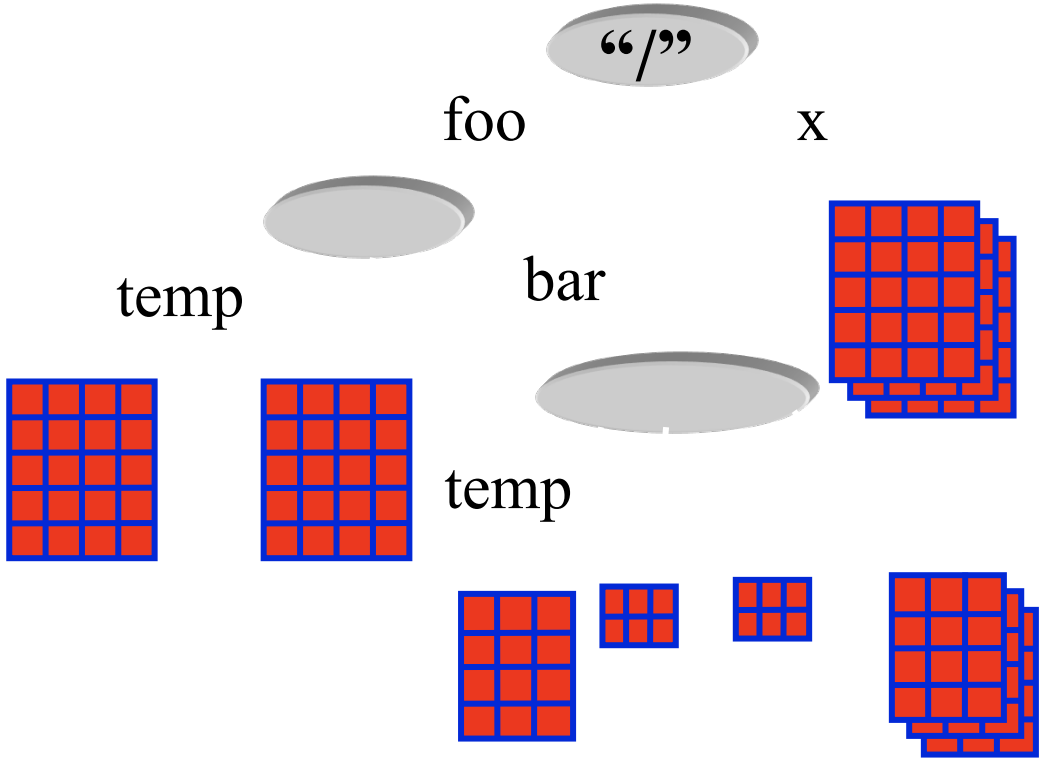
Groups

- **Group – a mechanism for describing collections of related objects**

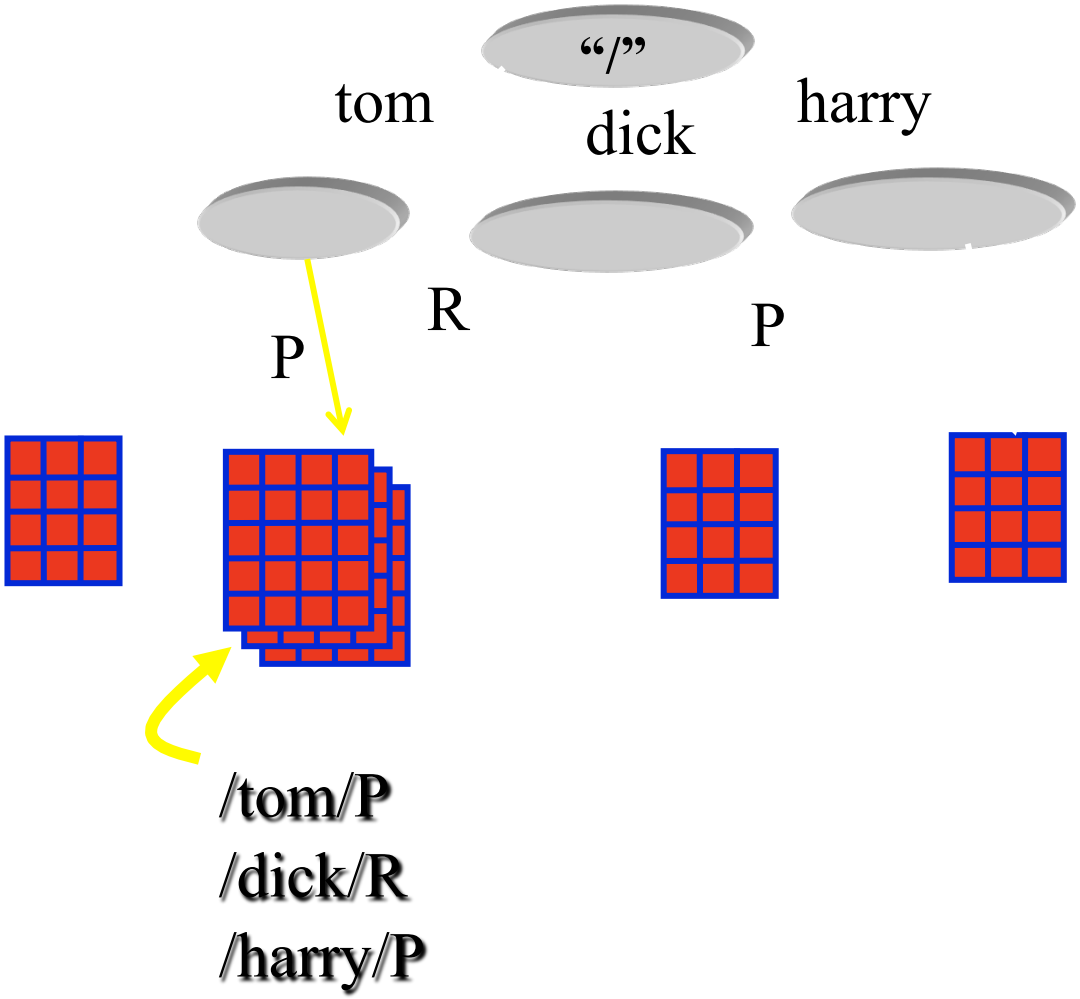


Objects are identified/located by pathnames

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



Groups & members of groups can be shared



Fortran vs C ordering

- **FORTRAN is Colum Major**
  - myLovelyData[10,20,30]
- **C is Row Major**
  - myLovelyData[30][20][10]

- **HDF5 assumes C style arrays**

- **Oh no, my data has been munged!**
  - No it hasn't
  - The arrays are ordered in memory order of fastest changing index

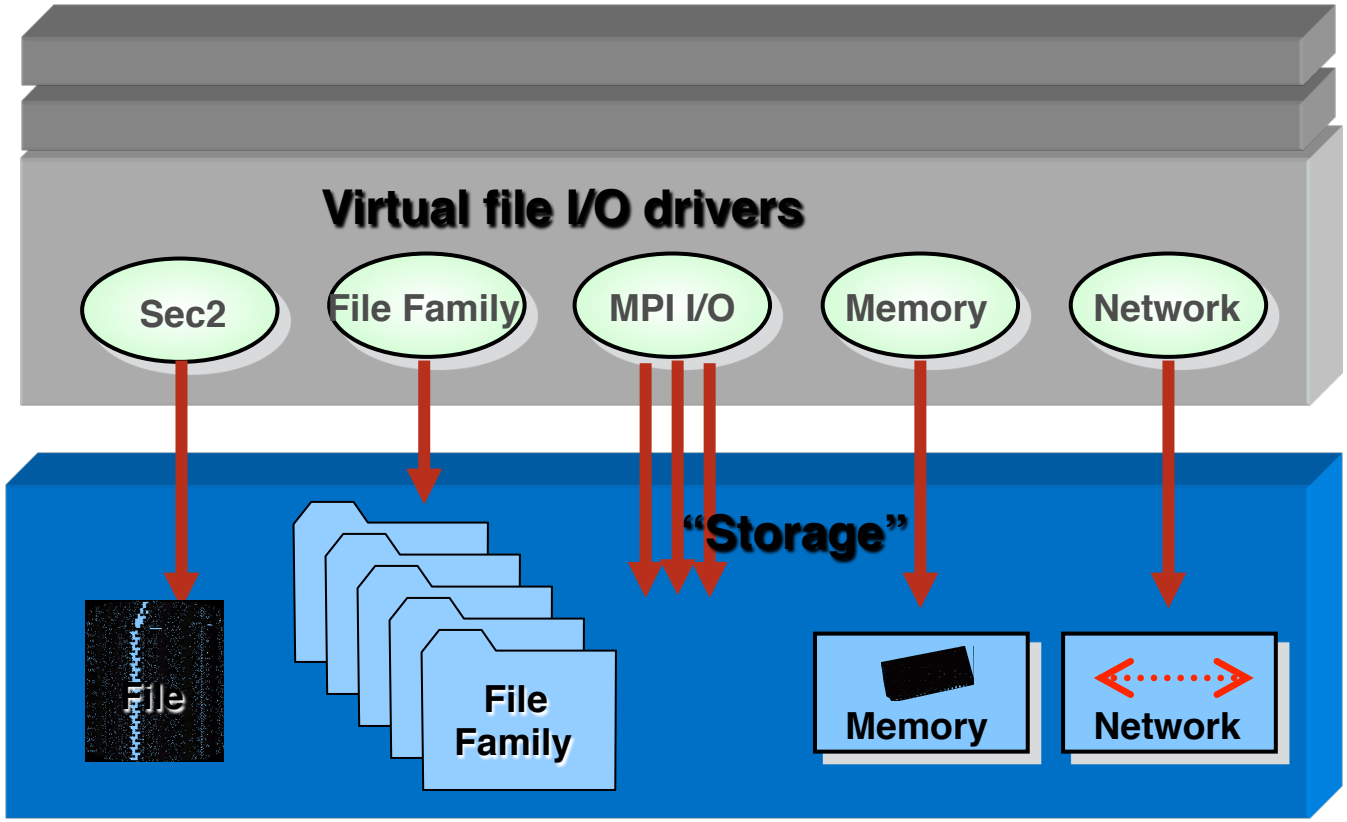
- **HDF5 does not state {Dim0 = X, Dim1 = Y, Dim2 = Z}**
  - It says, dim0 has {10}, dim1 has {20}, dim2 has {30}

It is up to the person reading/writing the data to interpret

This is why we have conventions/metadata/etc

Virtual file I/O layer

- **A public API for writing I/O drivers**
- **Allows HDF5 to interface to disk, the network, memory, or a user-defined device**



H5Dump

```
usage: h5dump [OPTIONS] files
  OPTIONS
     -h, --help           Print a usage message and exit
     -n, --contents       Print a list of the file contents and exit
     -B, --superblock     Print the content of the super block
     -H, --header         Print the header only; no data is displayed
     -A, --onlyattr       Print the header and value of attributes
     -i, --object-ids     Print the object ids
     -r, --string         Print 1-byte integer datasets as ASCII
     -e, --escape         Escape non printing characters
     -V, --version        Print version number and exit
     -a P, --attribute=P  Print the specified attribute
                          If an attribute name contains a slash (/), escape the
                          slash with a preceding backslash (\).
                          (See example section below.)
     -d P, --dataset=P    Print the specified dataset
     -y, --noindex        Do not print array indices with the data
     -p, --properties     Print dataset filters, storage layout and fill value
     -f D, --filedriver=D Specify which driver to open the file with
     -g P, --group=P      Print the specified group and all members
     -l P, --soft-link=P  Print the value(s) of the specified soft link
     -o F, --output=F     Output raw data into file F
     -b B, --binary=B     Binary file output, of form B
```

H5Dump cont.

```
     -t P, --datatype=P   Print the specified named datatype
     -w N, --width=N      Set the number of columns of output. A value of 0
(zero)
                          sets the number of columns to the maximum (65535).
                          Default width is 80 columns.
     -m T, --format=T     Set the floating point output format
     -q Q, --sort_by=Q    Sort groups and attributes by index Q
     -z Z, --sort_order=Z Sort groups and attributes by order Z
     -M L, --packedbits=L Print packed bits as unsigned integers, using mask
                          format L for an integer dataset specified with
                          option -d. L is a list of offset,length values,
                          separated by commas. Offset is the beginning bit in
                          the data value and length is the number of bits of
                          the mask.
     -R, --region         Print dataset pointed by region references
     -x, --xml            Output in XML using Schema
     -u, --use-dtd        Output in XML using DTD
     -D U, --xml-dtd=U    Use the DTD or schema at U
     -X S, --xml-ns=S      (XML Schema) Use qualified names n the XML
                          ":": no namespace, default: "hdf5:"
                          E.g., to dump a file called `-f', use h5dump -- -f
     --enable-error-stack Prints messages from the HDF5 error stack as they
                          occur.
     --no-compact-subset  Disable compact form of subsetting and allow the use
```

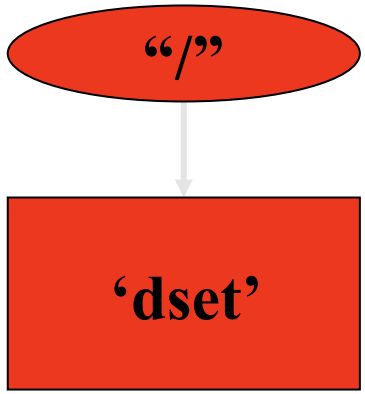
H5Dump output

```
h5dump.exe -H /c/data/Francis-Seeds.h5part
HDF5 "c:/data/Francis-Seeds.h5part" {
GROUP "/" {
   GROUP "Step#0" {
      ATTRIBUTE "TimeValue" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
      DATASET "Coords_0" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 5095 ) / ( 5095 ) }
      }
      DATASET "Coords_1" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 5095 ) / ( 5095 ) }
      }
      DATASET "Coords_2" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 5095 ) / ( 5095 ) }
      }
      DATASET "Pressure" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 5095 ) / ( 5095 ) }
      }
   }
}
}
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

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
      }
   }
}
}
```



Xdmf Generator from H5Dump XML

```
<Grid Name="Particles">
   <Topology TopologyType="Polyvertex">
   </Topology>
   <Geometry GeometryType="X_Y_Z">
     <DataItem Name="x">/Step#0/X</DataItem>
     <DataItem Name="y">/Step#0/Y</DataItem>
     <DataItem Name="z">/Step#0/Z</DataItem>
   </Geometry>
   <Attribute AttributeType="Vector" Name="Velocity">
     <DataItem Function="JOIN($0, $1, $2)" ItemType="Function">
       <DataItem >/Step#0/VX</DataItem>
       <DataItem >/Step#0/VY</DataItem>
       <DataItem >/Step#0/VZ</DataItem>
     </DataItem>
   </Attribute>
   <Attribute>
     <DataItem>/Step#0/P</DataItem>
   </Attribute>
   <Attribute>
     <DataItem>/Step#0/Smooth</DataItem>
   </Attribute>
 </Grid>
```

Pause

Steps to Create a File

1. **Specify File Creation and Access Property Lists, if necessary**

2. **Create a file**
3. **Close the file and the property lists, if necessary**

Property Lists

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

- **File Creation Property List**
  - Controls file metadata
  - Size of the user-block, sizes of file data structures, etc.
  - Specifying H5P_DEFAULT uses the default values

- **Access Property List**
  - Controls different methods of performing I/O on files
  - Unbuffered I/O, parallel I/O, etc.
  - Specifying H5P_DEFAULT uses the default values.

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);
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Example 1

```
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);
```

Location ID
Same as with H5GCreate

Terminate access to the File

Steps to Create a Dataset

1. Obtain location ID where dataset is to be created
2. Define dataset characteristics (datatype, dataspace, dataset storage properties, if necessary)
3. Create the dataset
4. Close the datatype, dataspace, and property list, if necessary
5. 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 storage (creation) properties (e.g. chunked and compressed, by default storage is contiguous)

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 (DT) are the same on all platforms**
- **These are DT handles generated at run-time**
- **Used to describe DT in the HDF5 calls**
- **DT are not used to describe application data buffers**

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 all platforms**
- **These are DT handles generated at run-time**

Dataspaces

- **Dataspace: size and shape of dataset**
- **Rank: number of dimension**
- **Dimensions: sizes of all dimensions**

Permanent – part of dataset definition
API handles this

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.

Writing arrays

- **Should you write the data as it appears in memory**
- **Or should you collect it up and write it out 'nicely'**
- **{x},{y},{z} stored as 3 arrays in memory**
- **Post processing using a tool like ParaView**
  - {x,y,z} stored as tuples in memory
  - Fast Output, slower input
  - Write once, read many times
  - Not really an issue, but worth remembering

Dataspace create example (3-vector array)

```
hid_t dataset;
hid_t &diskshape = H5FileId->diskshape;
hid_t &memshape  = H5FileId->memshape;
      // note, shapes might just be (1D in this example)
      // memshape = H5Screate_simple(1, count1_mem, NULL);
      //
if (memshape!=H5S_ALL) {
  if (H5Sclose(memshape)<0) vtkErrorMacro(<<"memshape :
      HANDLE_H5S_CLOSE_ERR");
      memshape = H5S_ALL;
  }
herr_t r=0;
int Nt = data->GetNumberOfTuples();
int Nc = data->GetNumberOfComponents();
hsize_t     count1_mem[] = { Nt*Nc };
hsize_t     count2_mem[] = { Nt };
hsize_t     offset_mem[] = { 0 };
hsize_t     stride_mem[] = { Nc };
```

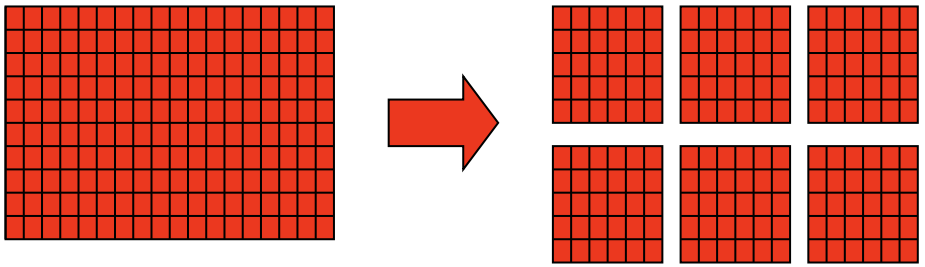
Continued

```
if (Nc>1) {
   sprintf(buffer,"_%i", c);
   name = name.append(buffer);
   if (!this->VectorsWithStridedWrite) {
     this->CopyFromVector(c, data, component);
     finalData = component;
   }
   else {
     offset_mem[0] = c;
     r = H5Sselect_hyperslab(
       memshape, H5S_SELECT_SET, offset_mem, stride_mem,
        count2_mem, NULL);
   }
 }
```

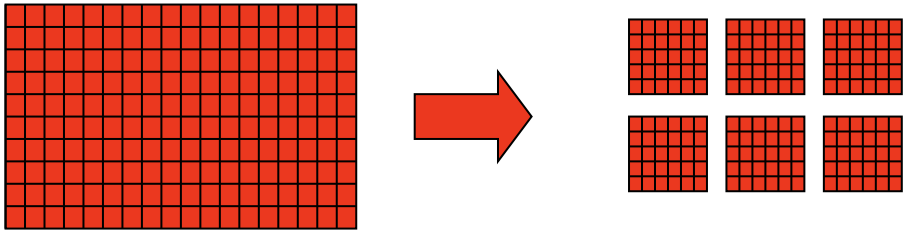
Dataset Creation Property List

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

Chunked



Chunked & compressed



HDF5 Chunking and compression

Chunking is required for compression and other filters

HDF5 filters modify data during I/O operations

Filters provided by HDF5:

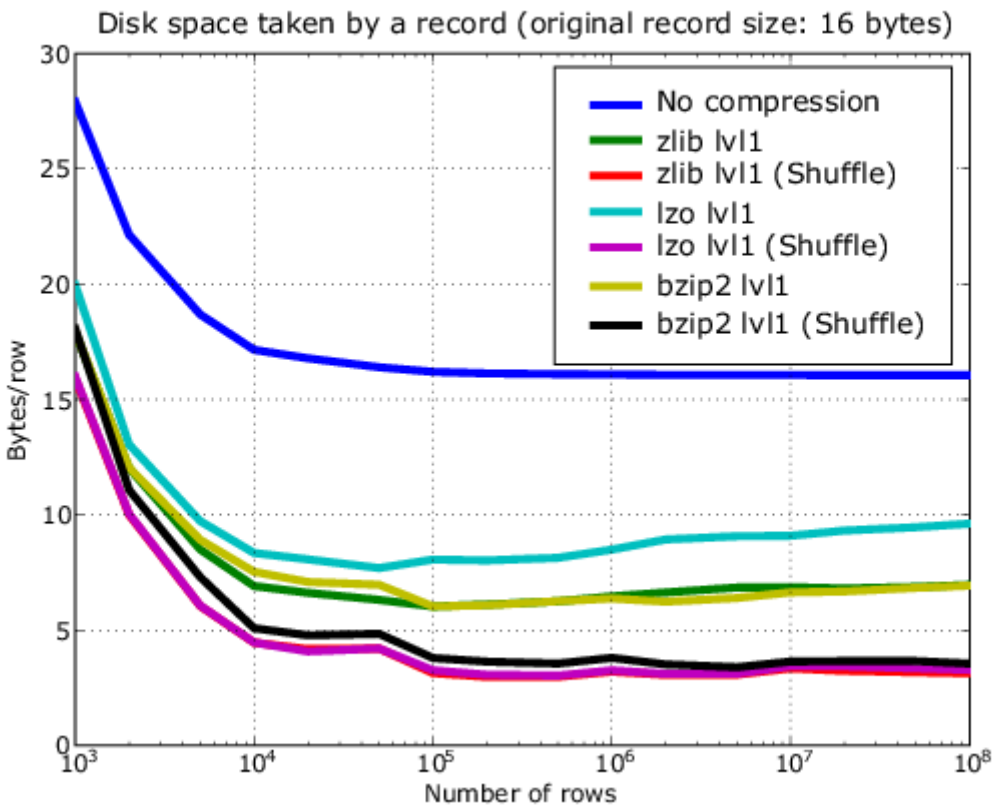
- Checksum (H5Pset_fletcher32)
- Data transformation (in 1.8.*)
- Shuffling filter (H5Pset_shuffle)

Compression (also called filters) in HDF5

- Scale + offset (in 1.8.*) (H5Pset_scaleoffset)
- N-bit (in 1.8.*) (H5Pset_nbit)
- GZIP (deflate) (H5Pset_deflate)
- SZIP (H5Pset_szip)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Compression Shuffle filter saving

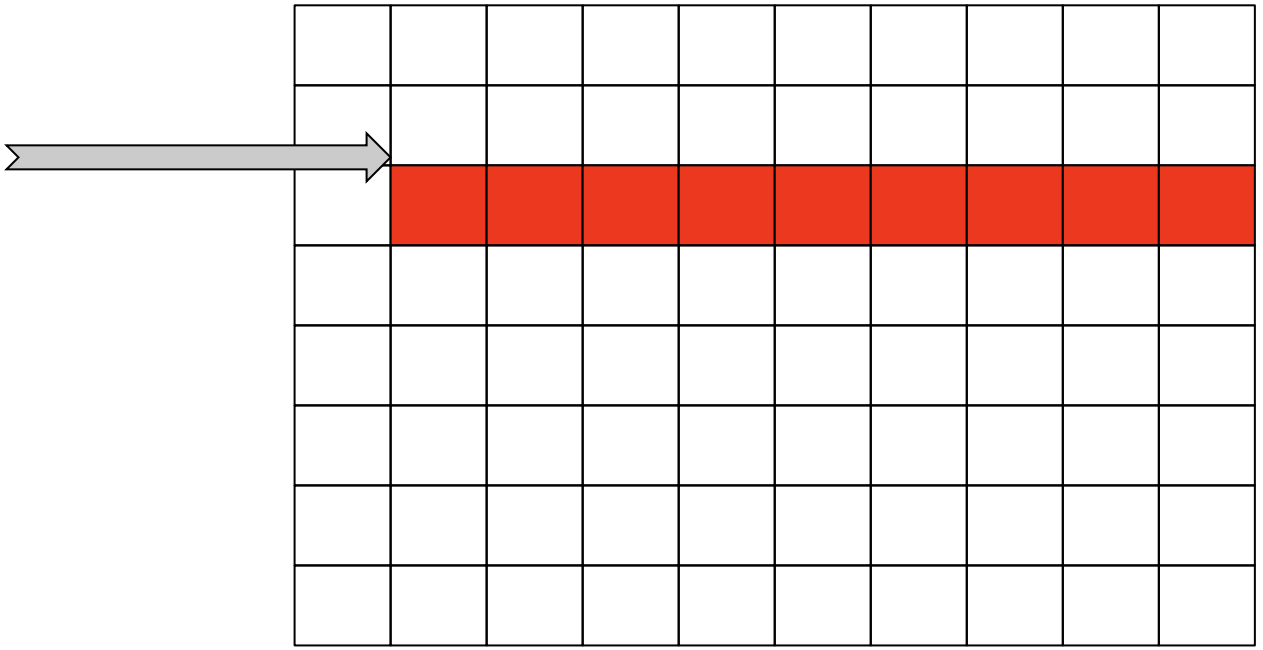


"The shuffle filter de-interlaces a block of data by reordering the bytes. All the bytes from one consistent byte position of each data element are placed together in one block; all bytes from a second consistent byte position of each data element are placed together a second block; etc.

For example, given three data elements of a 4-byte datatype stored as 012301230123, shuffling will re-order data as 000111222333. This can be a valuable step in an effective compression algorithm because the bytes in each byte position are often closely related to each other and putting them together can increase the compression ratio. "

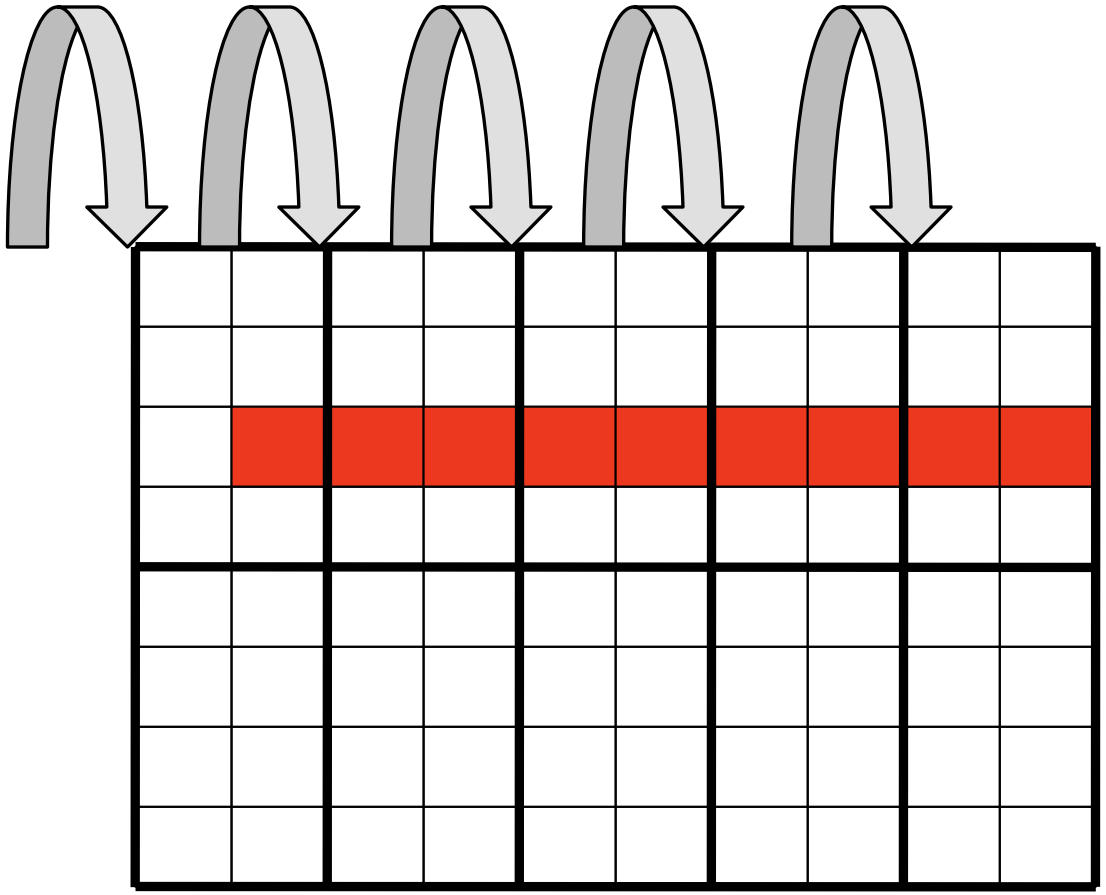
Source : PyTables website

Accessing a row in contiguous dataset



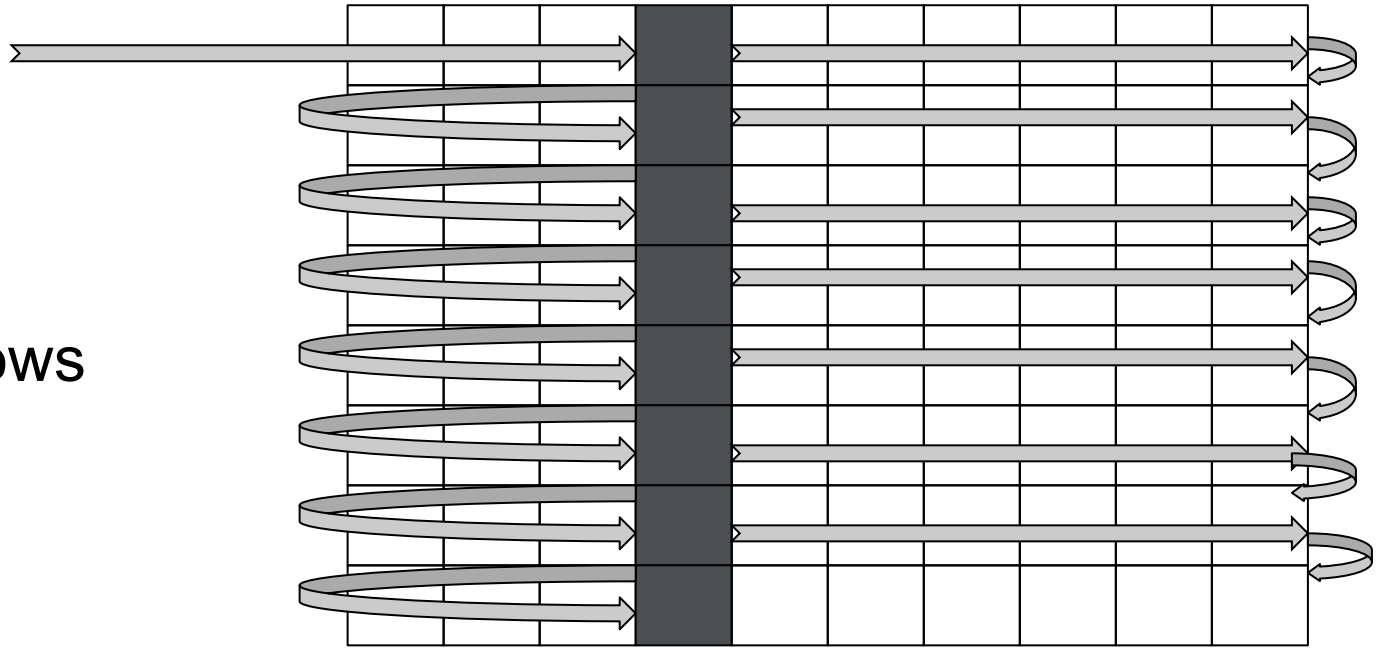
One seek is needed to find the starting location of row of data.
Data is read/written using one disk access.

Accessing a row in chunked dataset



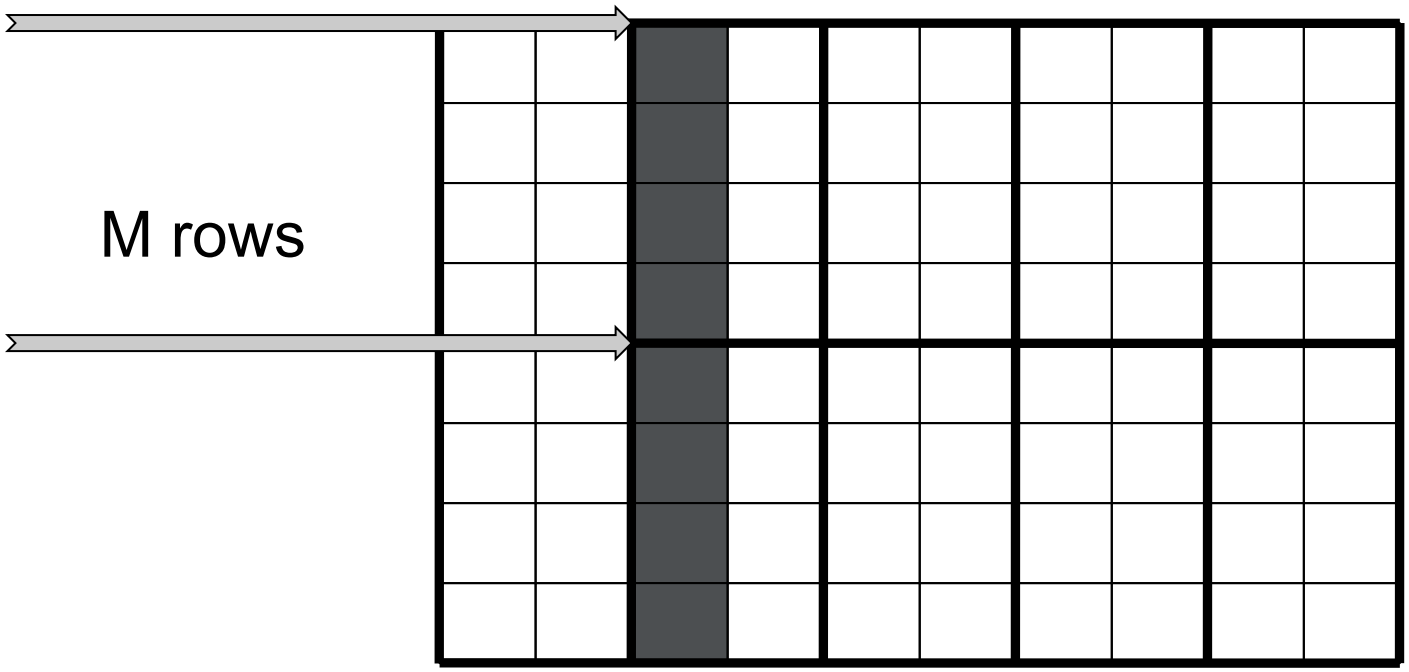
Five seeks is needed to find each chunk. Data is read/written using five disk accesses. Chunking storage is less efficient than contiguous storage.

Accessing data in contiguous dataset



M seeks are needed to find the starting location of the element. Data is read/ written using M disk accesses. Performance may be very bad.

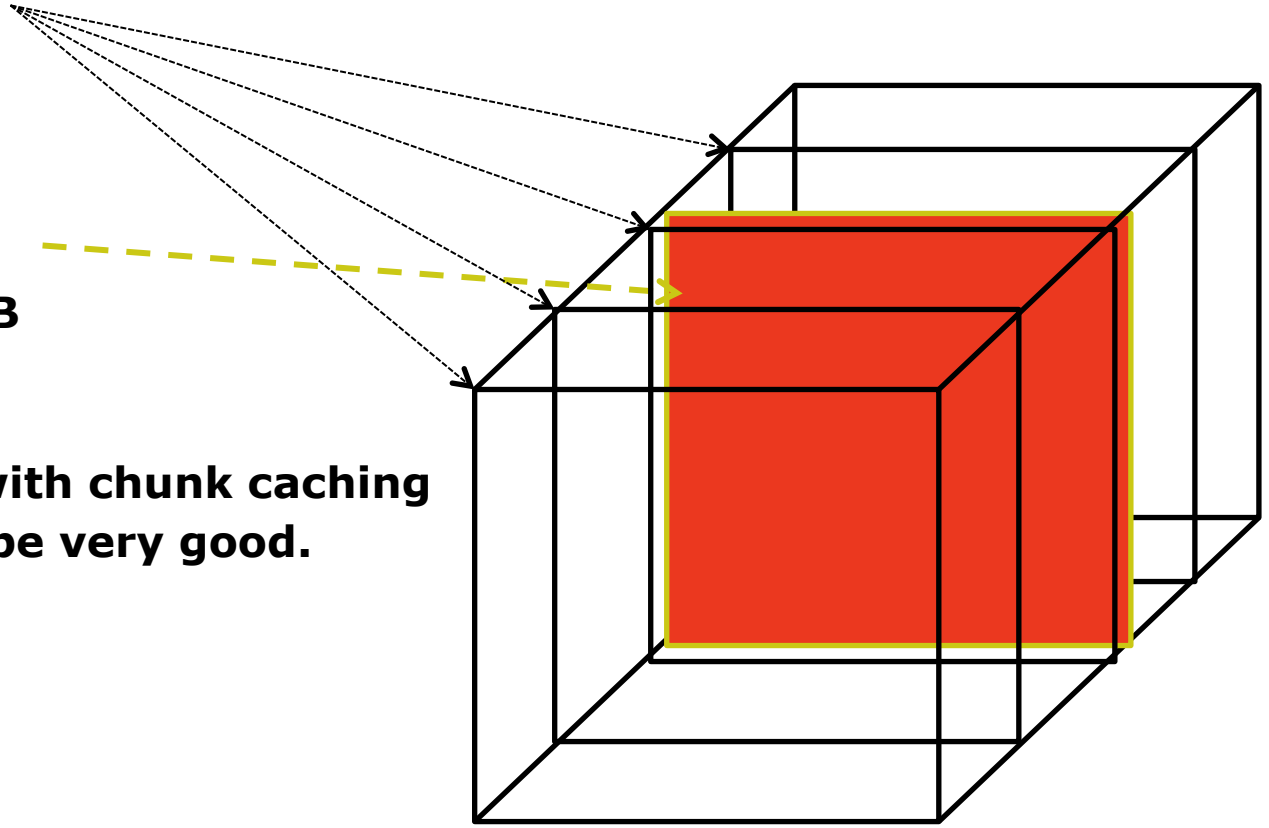
Motivation for chunking storage



Two seeks are needed to find two chunks. Data is read/written using two disk accesses. For this pattern chunking helps with I/O performance.

Example of slice writing by chunk

- **20 Chunks**
- **1000 slices**
- **Chunk size ~ 2MB**
- **Total size ~ 40MB**
- **Each plane ~ 40KB**
- **When combined with chunk caching performance can be very good.**



Property List Example

- **Creating a dataset with ``deflate'' compression**

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

Remaining Steps to Create a Dataset

3. **Create the dataset**
4. **Close the datatype, dataspace, and property list, if necessary**
5. **Close the dataset**

Example 2 – Create an empty 4x6 dataset

```
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);

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

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

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

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Example 2 – Create an empty 4x6 dataset

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


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

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


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


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

Create a dataspace

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Example 2 – Create an empty 4x6 dataset

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


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);


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


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

rank

current dims

Set maxdims
to current dims

Example 2 – Create an empty 4x6 dataset

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


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

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);


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

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Example 2 – Create an empty 4x6 dataset

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


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

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

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

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

Create a dataset

Pathname

Datatype

Dataspace

Property list (default)

Example 2 – Create an empty 4x6 dataset

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


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

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


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);
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Example2: h5dump Output

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
            }
      }
 }
 }
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Writing and Reading Datasets

Dataset I/O

- **Dataset I/O involves**
  - reading or writing
  - all or part of a dataset
  - Compressed/uncompressed

- **During I/O operations data is translated between the source & destination (file-memory, memory-file)**

  - Datatype conversion
    - data types (e.g. 16-bit integer => 32-bit float)
  - Dataspace conversion
    - dataspace (e.g. 10x20 2d array => 200 1d array)

Partial I/O

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

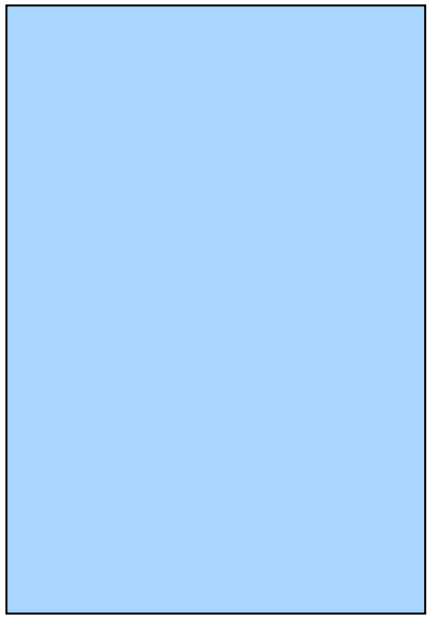
- **Selection**
  - Selections in memory can differ from selection in file
  - Number of selected elements is always the same in source and destination

- **Selection can be**
  - Hyperslabs (contiguous blocks, regularly spaced blocks)
  - Points
  - Results of set operations (union, difference, etc.) on hyperslabs or points

Reading Dataset into Memory from File

File

2D array of 16-bit ints



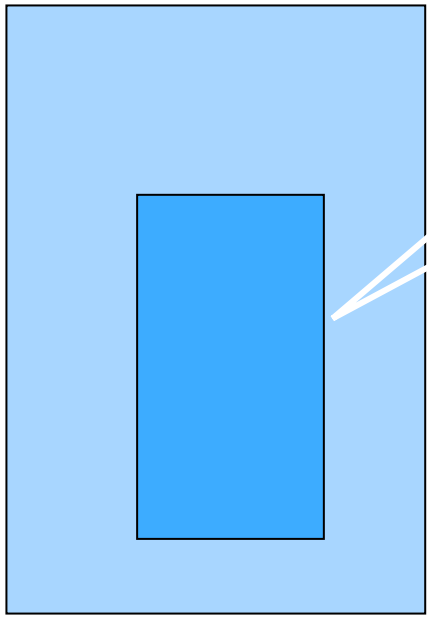
Memory

3D array of 32-bit ints

Reading Dataset into Memory from File

File

2D array of 16-bit ints

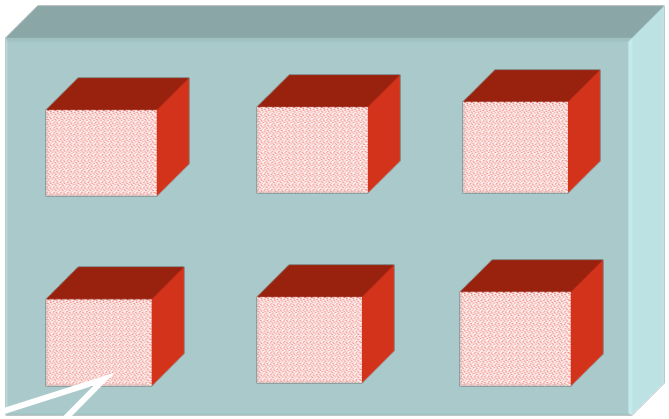


2-d array

Regularly spaced series of cubes

Memory

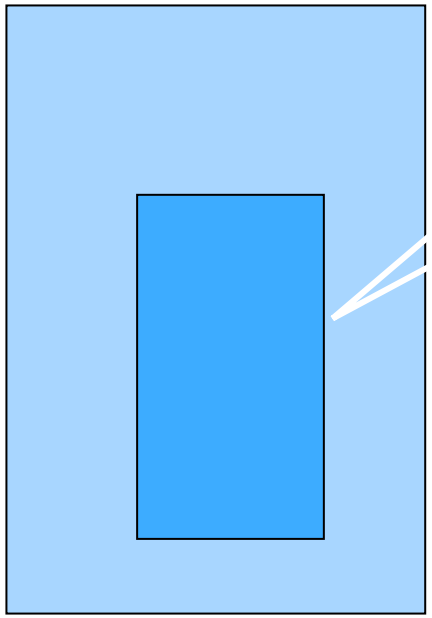
3D array of 32-bit ints



Reading Dataset into Memory from File

File

2D array of 16-bit ints

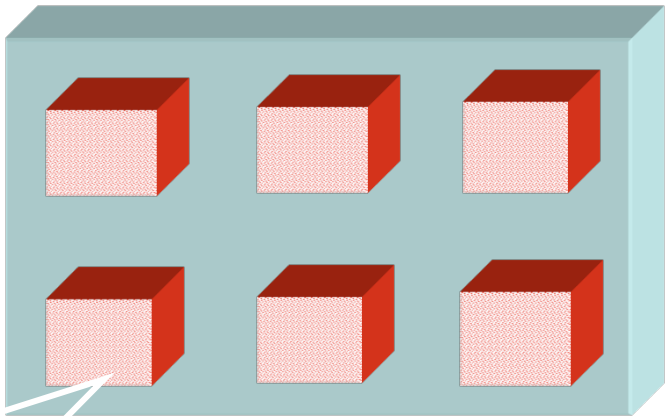


2-d array

Regularly spaced series of cubes

Memory

3D array of 32-bit ints

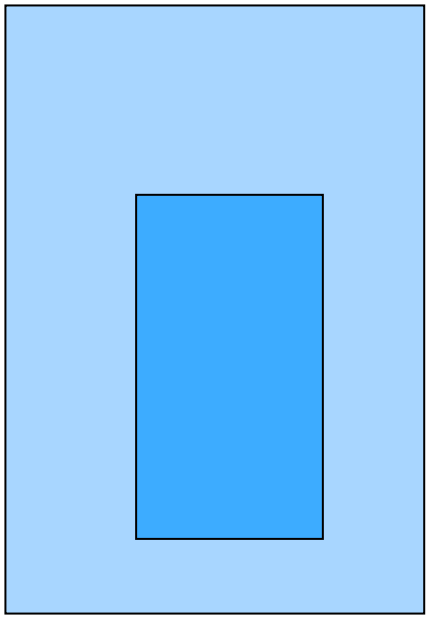


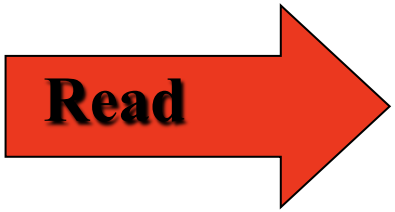
The only restriction is that the number of selected elements on the left be the same as on the right.

Reading Dataset into Memory from File

File

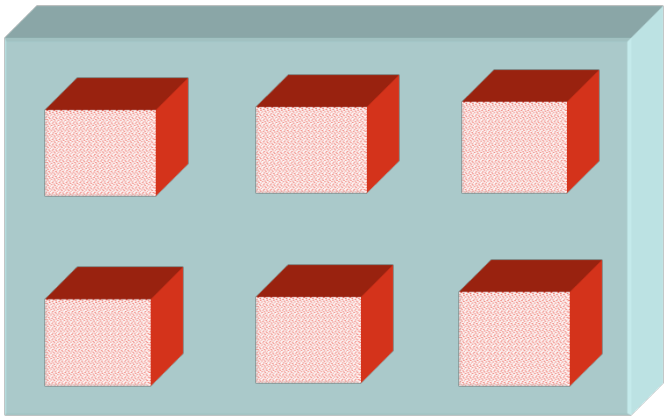
2D array of 16-bit ints





Memory

3D array of 32-bit ints



Steps for Dataset Writing/Reading

1. **If necessary, open the file to obtain the file ID**
2. **Open the dataset to obtain the dataset ID**
3. **Specify**
   - Memory datatype
   - ! *Library "knows" file datatype – do not need to specify* !
   - Memory dataspace (optional, needed for partial I/O)
   - File dataspace (optional, needed for partial I/O)
   - Transfer properties (optional)
4. **Perform the desired operation on the dataset**
5. **Close dataspace, datatype and property lists**

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.

Example 3 – Writing to an existing dataset

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


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


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


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

Example 3 – Writing to an existing dataset

```
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;
```

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

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

Example 3 – Writing to an existing dataset

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


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");


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

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Example 3 – Writing to an existing dataset

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


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


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

Write to dataset

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

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

H5Py

- **Python access to HDF5 datasets**
- **Removes most of the annoying create/close calls**
- **No need for property lists stuff**
- **Everything is wrapped nicely for you**

H5Py example snippet

```
# open output filename_out
fout = h5py.File(filename_out, 'w')

# create simulation group in output file
simulation = fout.create_group('simulation')

# convert simulation data to cgs and write to output file
date = fin['/simulation/date'][:]
fout.create_dataset('/simulation/date',data=date)
dx = fin['/simulation/dx'][:]
fout.create_dataset('/simulation/dx',data=dx)
num  = fin['/simulation/num'][:]
fout.create_dataset('/simulation/num',data=num)
time = fin['/simulation/time'][:]*(dx/c)
fout.create_dataset('/simulation/time',data=time)
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

netCDF and conventions

Parallel IO

- **Same ideas as before**
- **Selections based on hyperslabs**
  - Each process uses its own piece
- **Collective or independent?**
  - Property list used to set transfer type
  - Underlying mechanism uses MPIIO VFD
- **Metadata operations are collective in HDF5**
  - This is the main obstacle to scalability
- **Understand the file system and buffering**
  - Striping
  - Collective buffering nodes, IO/OSS nodes
  - Buffer sizes

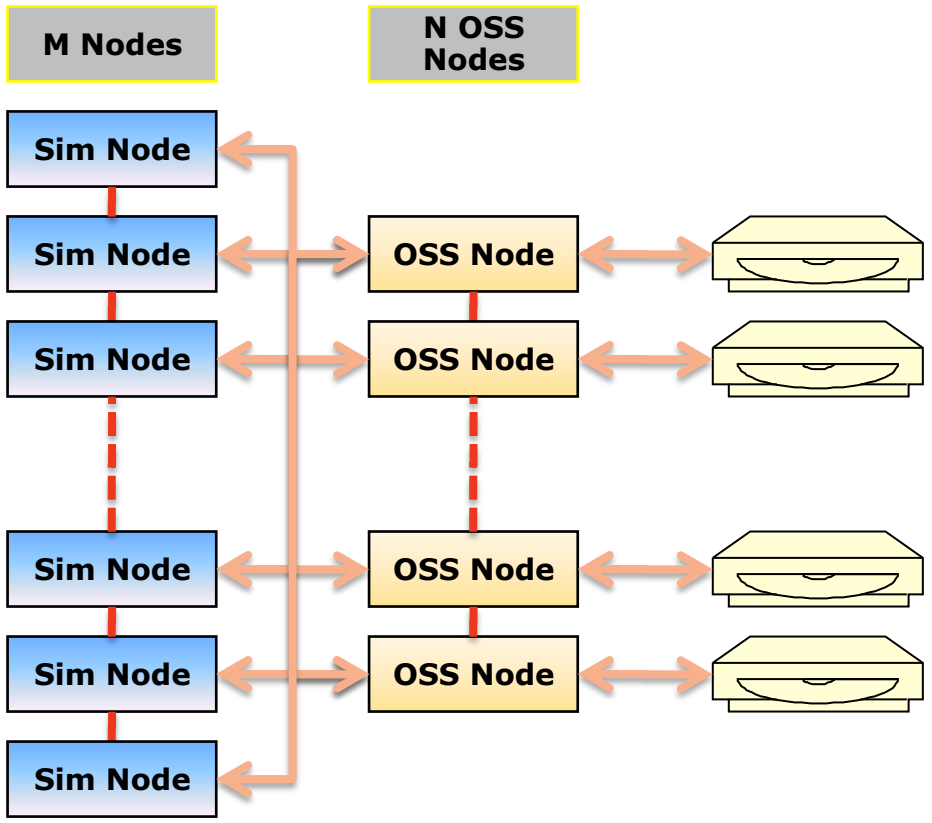
For cutting edge performance Check out ADIOS

Similar API and concepts, but more experimental for developers

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Parallel IO (c.f. Lustre)

HDF5 Parallel IO looks like this



Loose Coupling

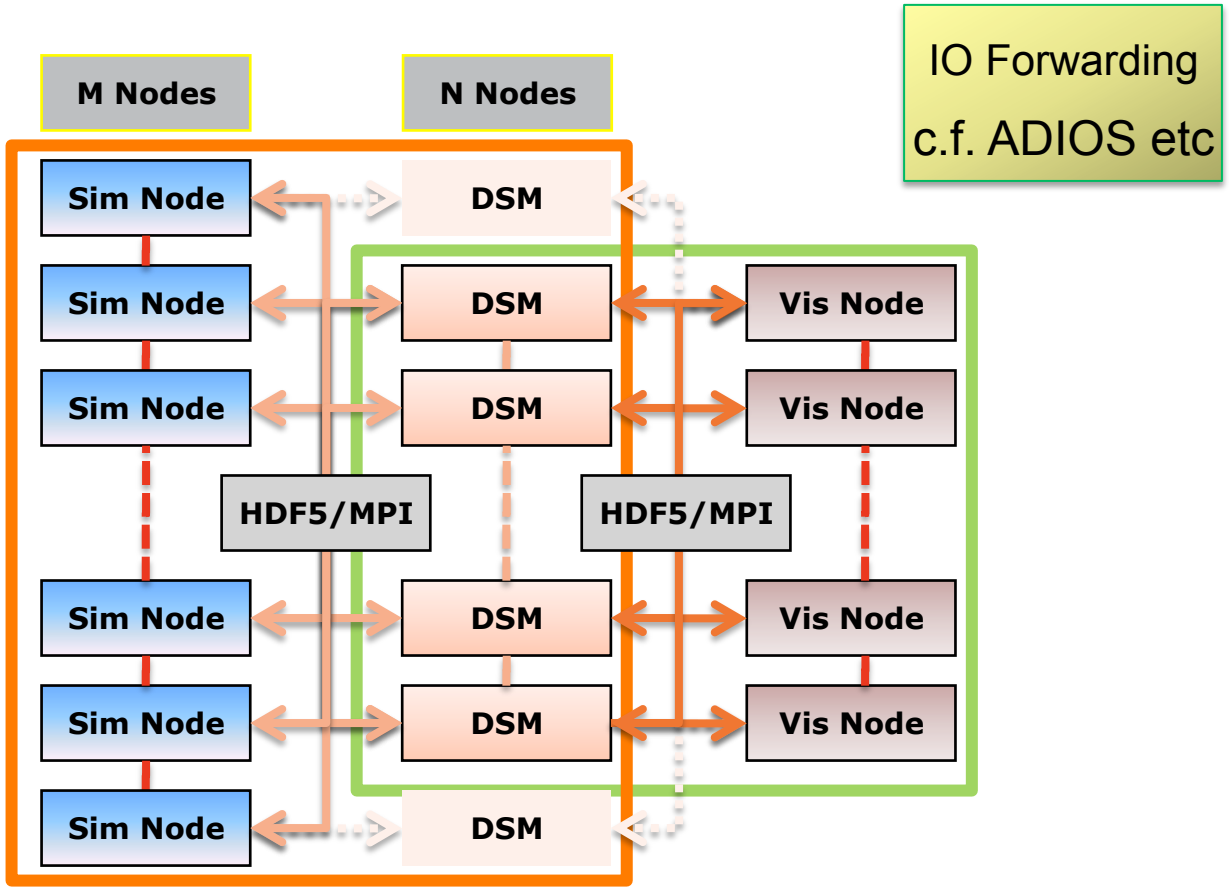
All IO goes through OSS nodes

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

DSM Implementation (à la Data-Staging)

DSM Model could looks like this

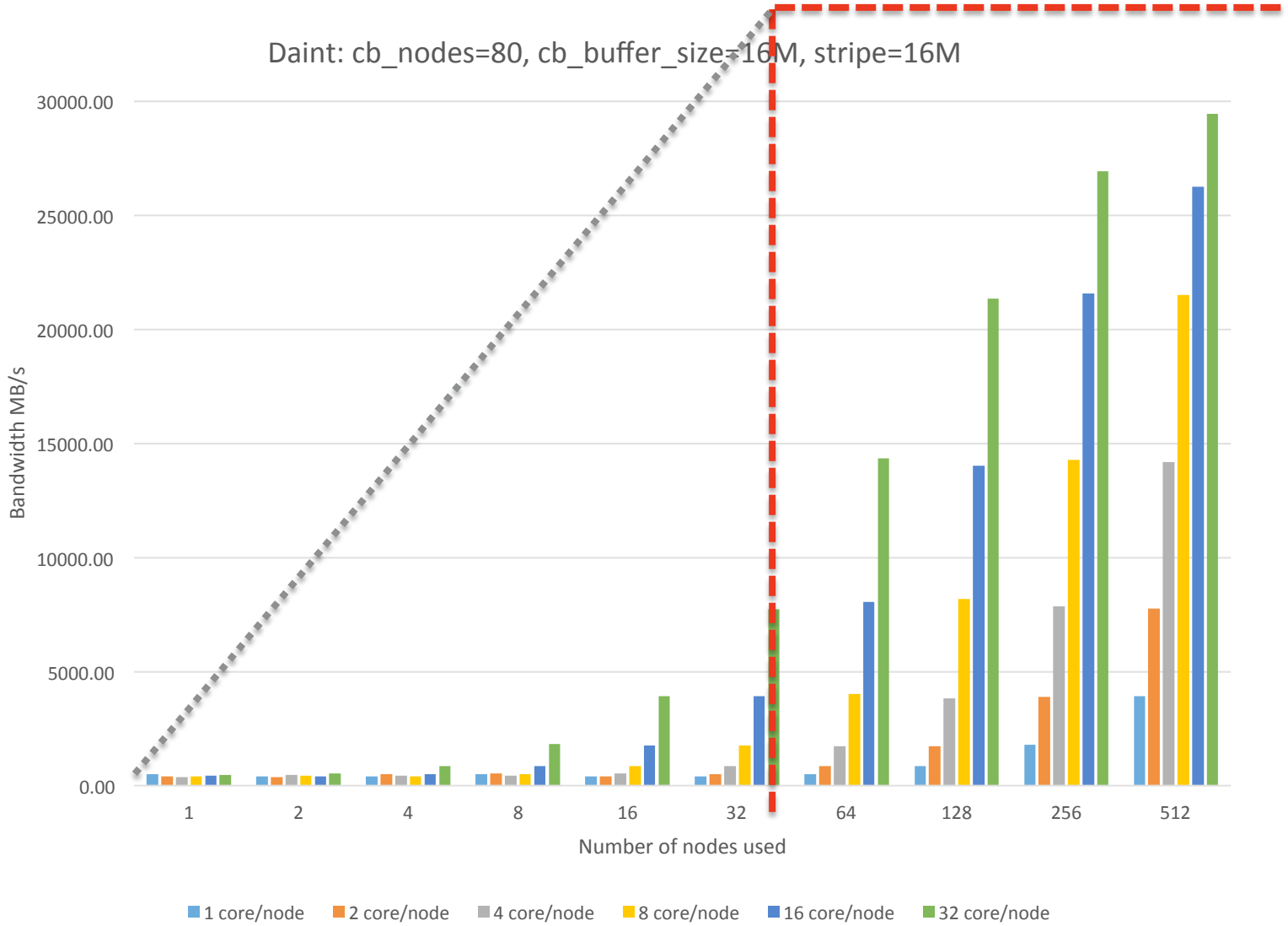
Just replace the IO layer in HDF5



HDF IO intercepted and routed to DSM

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

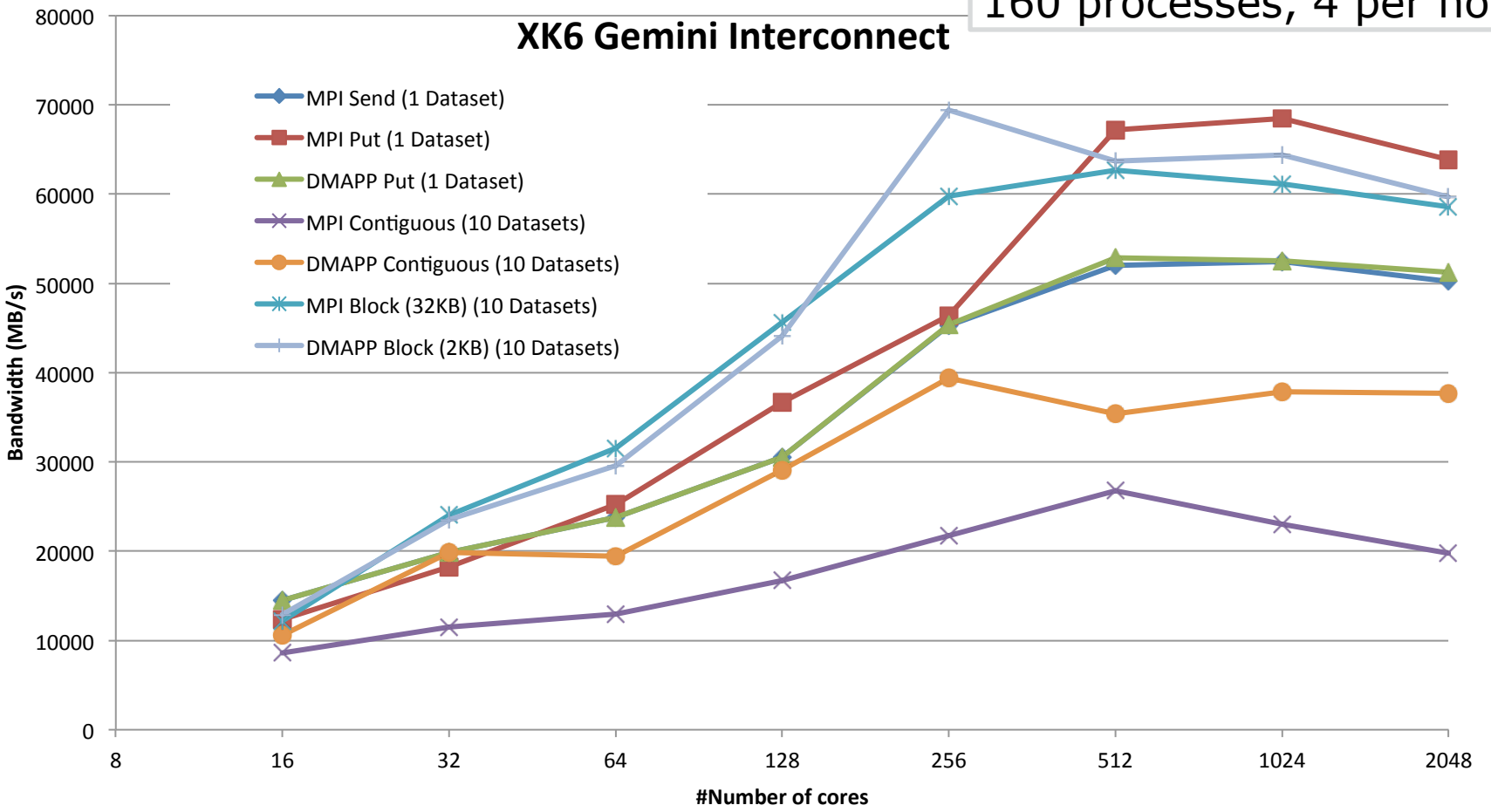
IO performance of HDF5 on Daint



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

DSM Transfer rates on Cray XK6 (Todi)

DSM 40 nodes
160 processes, 4 per node



Serialization of IO calls impacts the IO BW

Future HDF5 VOL / Function Shipping

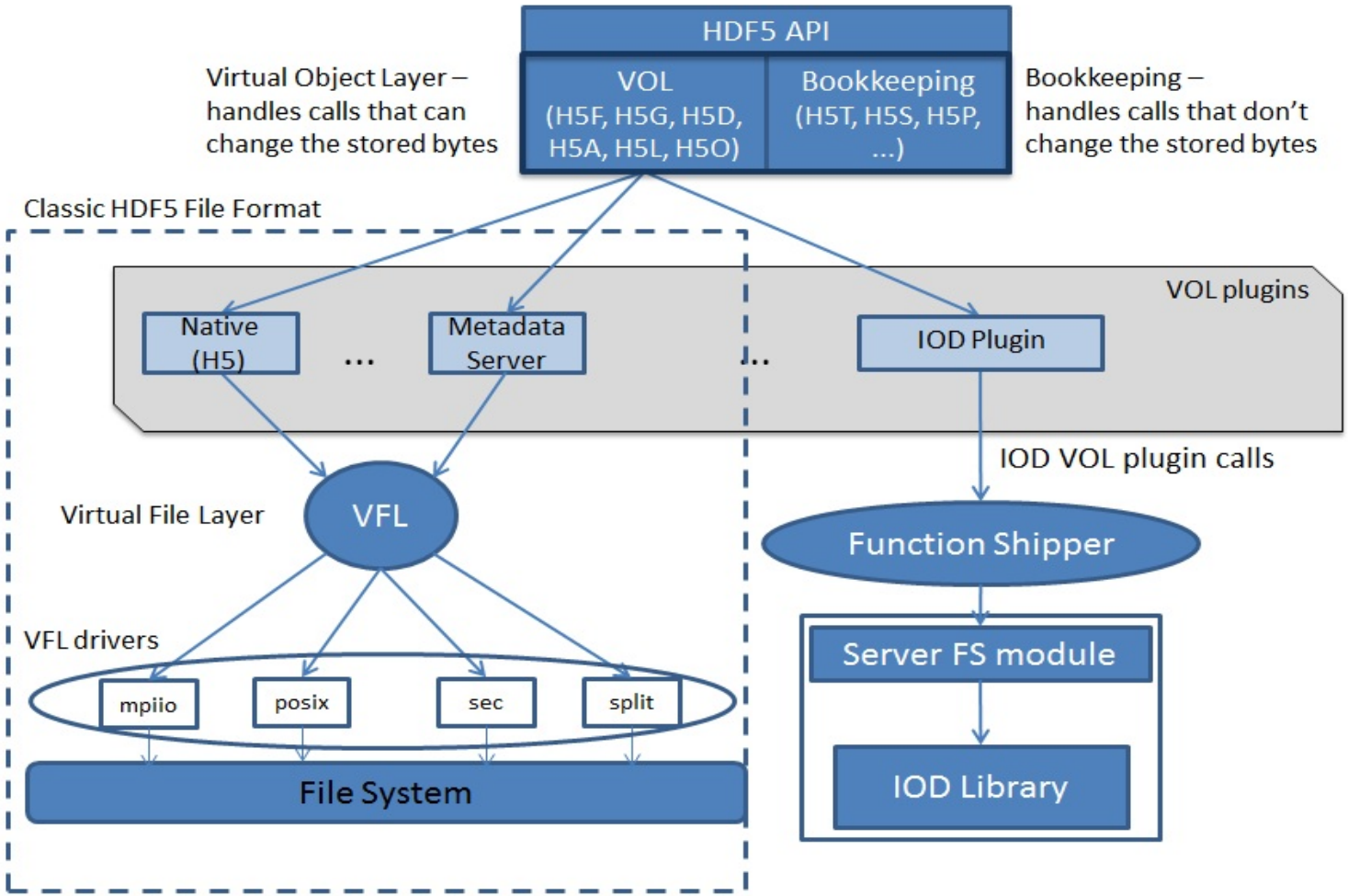


Image courtesy The HDF Group

FS + VOL

- Asynchronous + function shipping
  - Overall speed improvements, wait on request completion
  - Help HDFG to get this working on EPFL BG-Q + Daint
  - BG-Q MPI/PAMI/RDMA IBVerbs library for network layer
- Object level storage
  - Moving away from File addresses means we no longer need to synchronize the metadata before writing chunks - Scalability
- Blobs/Chunks/Pieces
  - Pieces do not have to map to addresses, therefore they can be variable size – on the fly compression permitted. Not previously possibly using hyperslabs.

NetCDF

- **Latest versions (4.x) built onto HDF5**
- **All previous discussions apply**
- **Interesting because of 'Conventions'**
  - Associating Dimensions with coordinate variables

NetCDF Jargon

- **"Variable"**
  - a multi-dimensional array of data, of any of 6 types (char, byte, short, int, float, or double).
- **"Dimension"**
  - information about an axis: it's name and length.
- **"Attribute"**
  - a 1D array of metadata.
- **"Coordinate Variable"**
  - a 1D variable with the same name as a dimension, which stores values for each dimension value.
- **"Unlimited Dimension" – a dimension which has no maximum size.**
  - Data can always be extended along the unlimited dimension

In brief

- **Variables**
  - are the basic data units. Each dimension has a name, data type, and ***shape***, which is a list of dimensions and determines whether the variable is a scalar (0 dimensions), a vector (1 dimension), a matrix (2 dimensions), etc.

- **Dimensions**
  - are used to describe physical dimensions like time, latitude, longitude, height, etc. Each dimension has a name and a length.

In brief

- **Attributes**
  - are used to store metadata. Most attributes provide information about a specific variable. Global attributes can be used to provide metadata about the dataset as a whole.

- **Attributes**
  - have an associated variable, a name, data type, length and value. All attributes are treated as vectors.

What Comes with NetCDF

- **NetCDF comes with 4 language APIs: C, C++, Fortran 77, and Fortran 90.**
  - Other language APIs (perl, python, MatLab, etc.) use the C API.
  - Another implementation uses a Java special – OpenDAP etc
- **Tools ncgen and ncdump.**
- ncdump is very handy, it tells you about your file
  - ncgen reads a textual representation of a netCDF dataset and generates the corresponding binary netCDF file or a C or FORTRAN program to create the netCDF dataset.
- **Tests.**
  - It's nice to know your build worked
- **Documentation.**
  - It nice to find instructions bundled
  - A language-specific guide is provided for C, C++, Fortran 77, and Fortran 90 users.
  - All documentation can be found at: http://my.unidata.ucar.edu/content/software/netcdf/docs
- CDL – Common Data Language

Parts of netCDF file

DIMENSIONS:
dimensions:
  lat = 64 ;
  lon = 128 ;
  time = 12 ;

FILE ATTRIBUTES:
global attributes:
  :title = “Temp: 1999" ;
  :source = “NCAR” ;
  :conventions = "None" ;

VARIABLES:
Names , Attributes, Coordinates
variables:
  float lat(lat) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
  float lon(lon) ;
    lon:long_name = "longitude" ;
    lon:units = "degrees_east" ;
  int time(time) ;
    time:long_name = "time" ;
    time:units = "Month of Year" ;
  float T(time, lat, lon) ;
   T:long_name = “Temperature”;
   T:units = “C" ;
   T:missing_value = 1.e+20f ;
   T:_FillValue = 1.e+20f

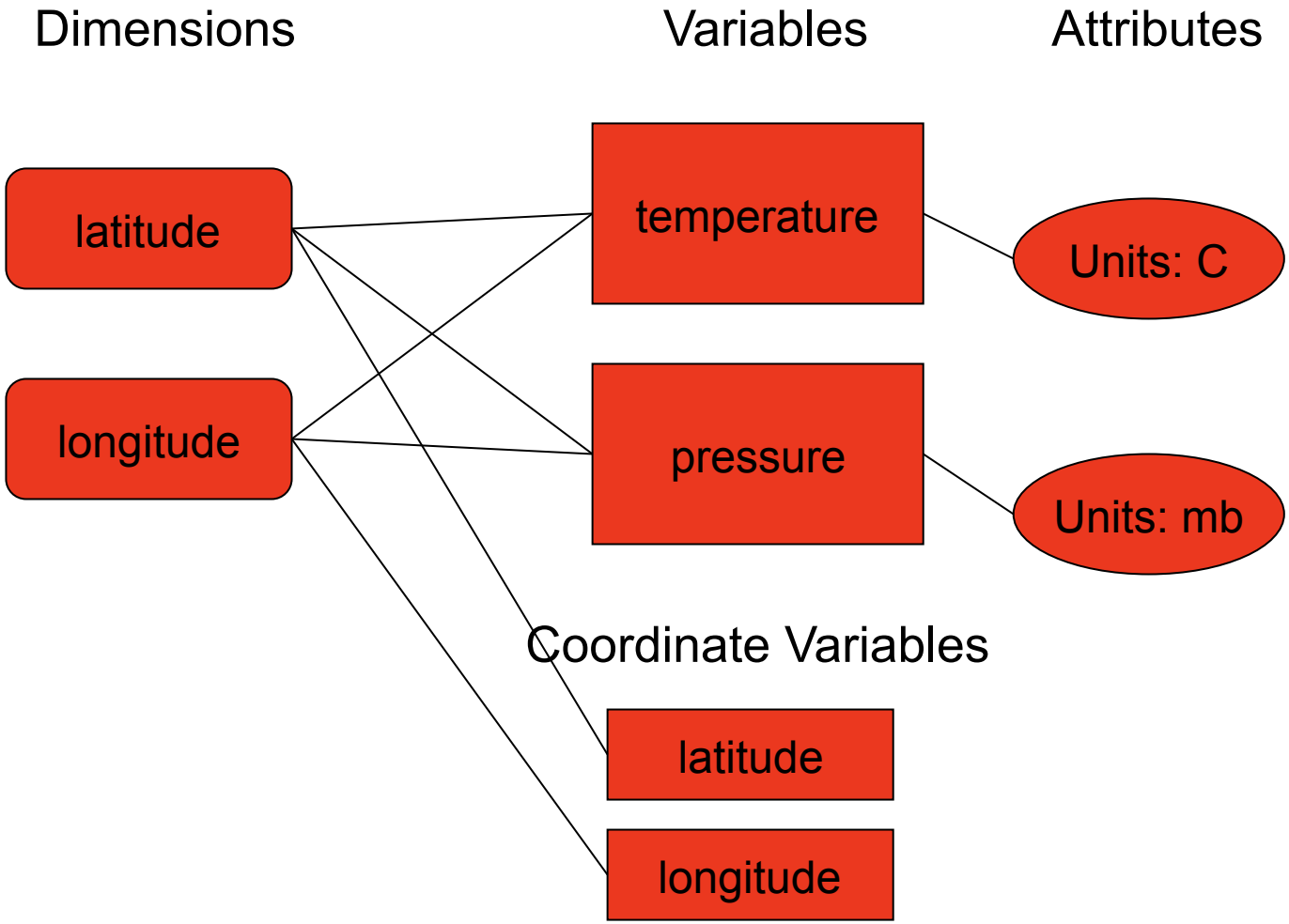
Detailed Look netCDF Variable

```
Variable:  u
Type: float
Total Size:  65536 bytes
             16384 values
Number of Dimensions:  3
Dimensions and Sizes:   [time | 2] x  [lat | 64] x [lon | 128]
Coordinates:
             time:  [ 1 .. 7 ]
             lat:  [ -87.8638 ..  87.8638 ]
             lon: [ 0 .. 357.185]
Number of Attributes: 5
    _FillValue :        1e36
    units :             m/s
    long_name  :        Zonal Wind Component
    short_name :        U
    missing_value :    1e36
```

NetCDF Example

- **Suppose a user wants to store temperature and pressure values on a 2D latitude/longitude grid.**
- **In addition to the data, the user wants to store information about the lat/lon grid.**
- **The user may have additional data to store, for example the units of the data values.**

NetCDF Model Example



NetCDF Conventions

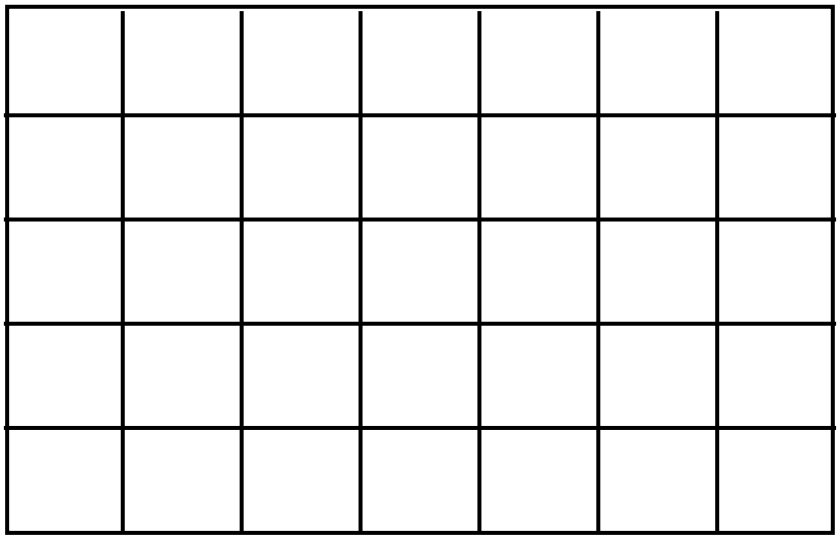
- **Unidata (http://www.unidata.ucar.edu)**
- offers a repository and will maintain WWW links for sets of netCDF conventions, as supported by the global `Conventions' attribute described in the Attribute Conventions section of the netCDF User's Guide.
- **Examples**
- CF Conventions (Climate and Forecast)
- COARDS Conventions (Cooperative Ocean/Atmosphere Research Data Service)
- GDV Conventions (Gridded data)
- GDT Conventions
- CDC netCDF Conventions for Gridded Data (Climate diagnostics centre)
- NCAR-RAF Conventions for Aircraft Data (National Center for Atmospheric Research)
- NUWG Conventions (netCDF Users Working Group)
- PMEL-EPIC Conventions (Pacific Marine Environmental Laboratory)
- Others in development

Best practice

- **Conventions = "COARDS" (or chosen favourite)**
- **Coordinate variables**
  - No missing values or other tricks. Keep simple
  - Use Units and long_name
- **Variable Grouping**
  - You may structure the data in a netCDF file in different ways, for example putting related parameters into a single variable by adding an extra dimension. Standard visualization and analysis software may have trouble breaking that data out.
- **Variable Attributes**
  - For each variable where it makes sense, add a units attribute, using the udunits conventions, if possible. The units attribute is a character string that specifies the units used for the variable's data. (UDUNITS is a library for manipulating units of physical qualities)
  - For each variable where it makes sense, add a long_name attribute, which is a human-readable descriptive name for the variable. This could be used for labeling plots, for example.

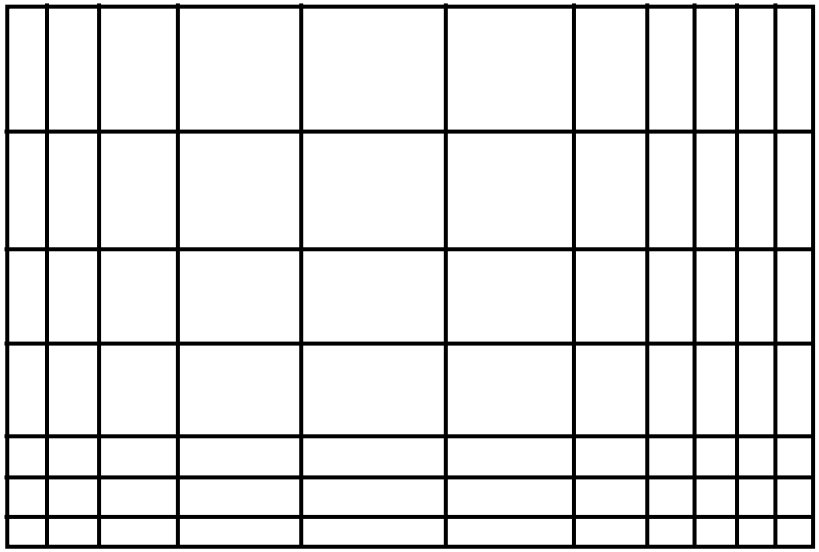
If the variable is simple

- **Then we get a regular grid**
- In as many dimensions as we like



If the variable is less simple

- **Then we get a rectilinear grid**
- In as many dimensions as we like



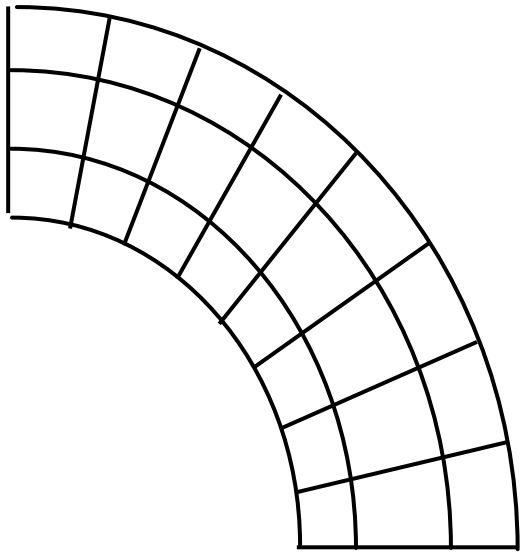
- For example - Variable : Height - 1,6,7,9,23,23.6…
- Coordinate : Pressure using Dimension Height/x/y/etc
- Can get multi resolution grids in one dataset

But It can be complicated

- **A dimension can be associated with a variable**
  - The variable might be 2 dimensional!
  - Variable = Temperature
  - Dimension 1 : Lon
  - Dimension 2 : Lat
  - Variable Lat (or lon)
  - Dimension 1 : X
  - Dimension 2 : Y
  - Temperature is now defined over a lon(x,y)/lat(x,y) grid but referenced indirectly via x/y
  - The grid can now be arbitrarily shaped/warped, with cells changing size (near poles usually)

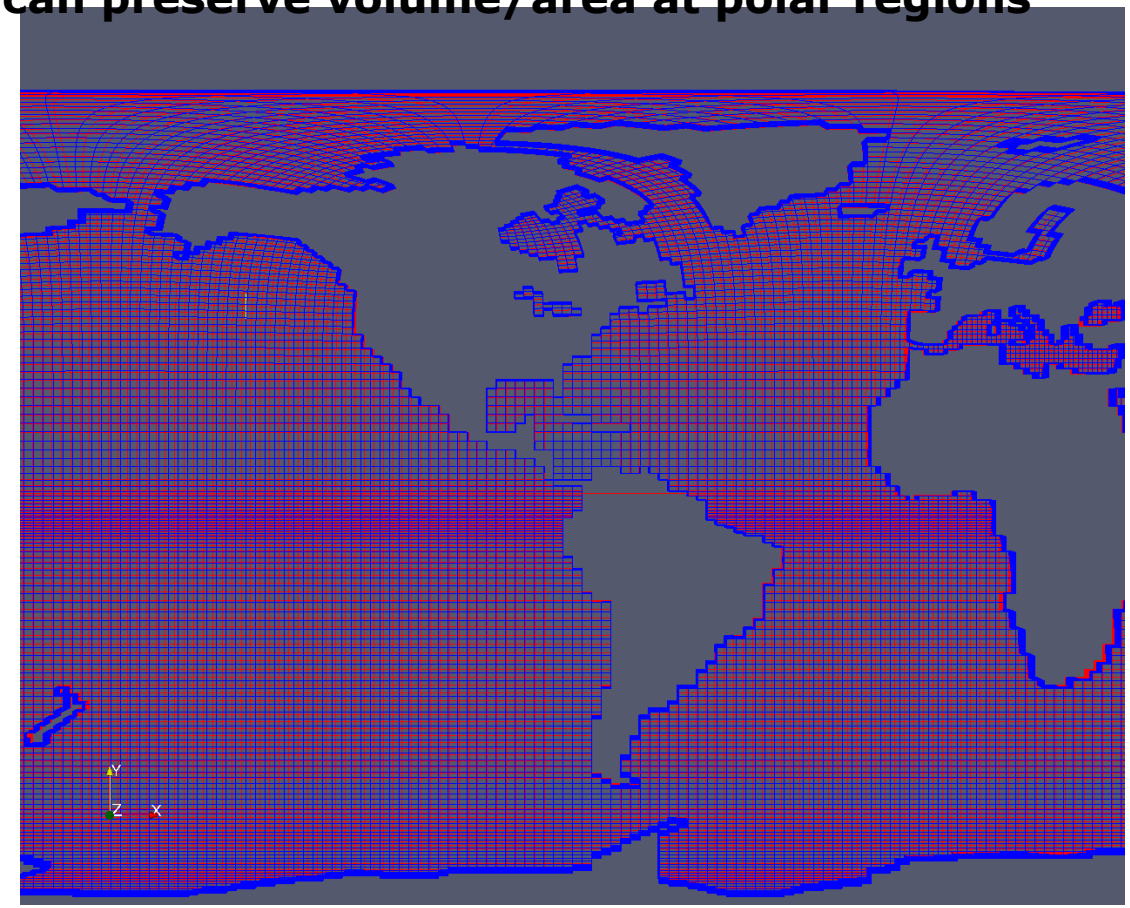
With multiple coordinate dimensions

- **Grids can be distorted or warped in useful ways**
- In as many dimensions as we like



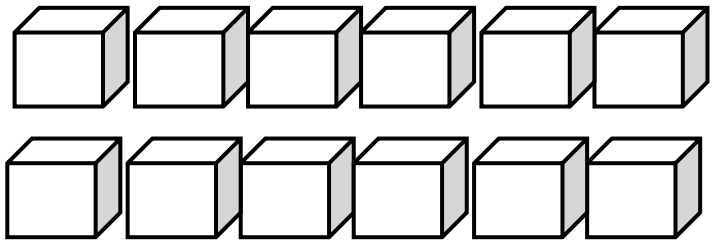
A Warped grid

- **We can preserve volume/area at polar regions**



But it can be more complicated

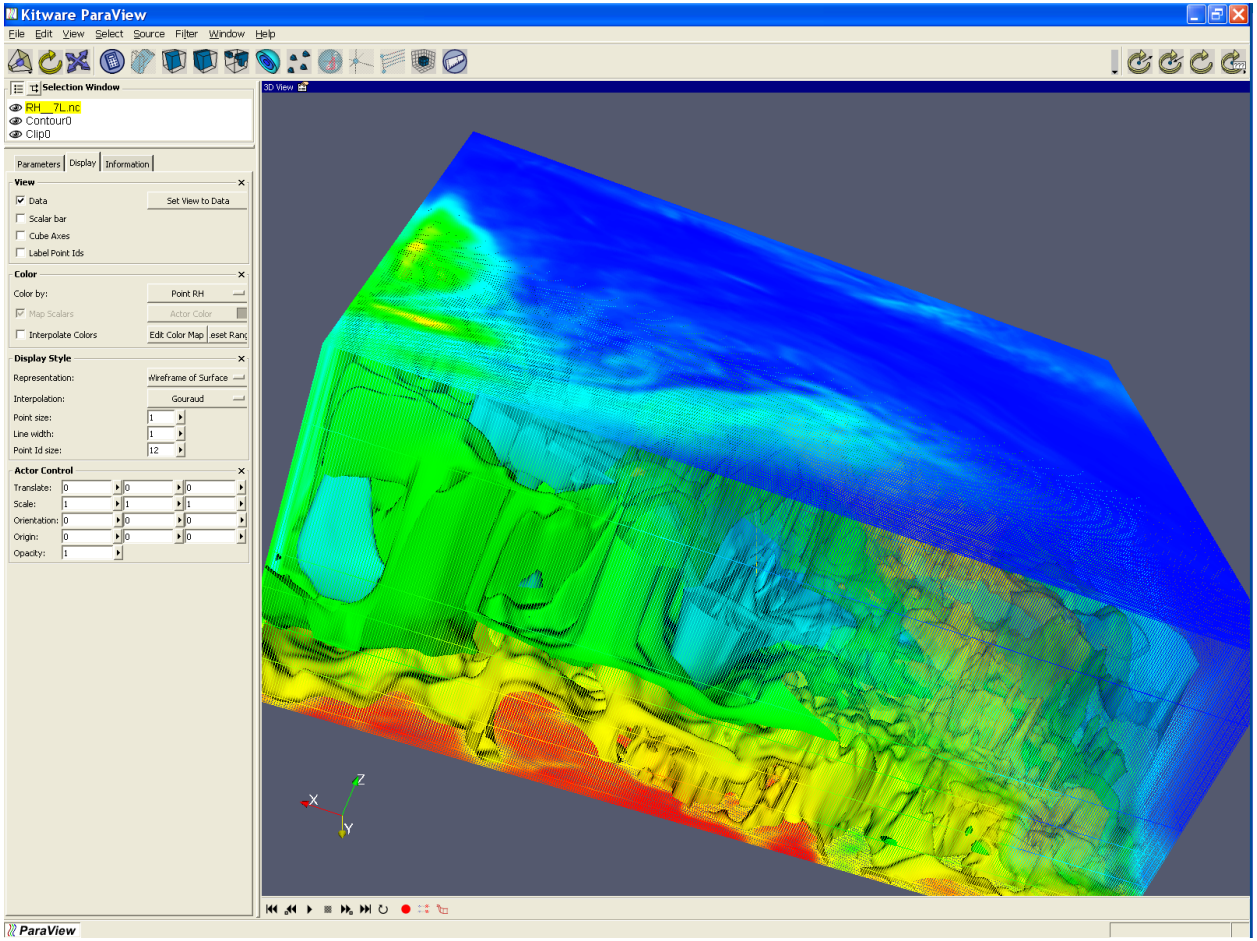
- **A variable may have a bounds var**
- Lon/Lat can now have Lon_Bounds + Lat_Bounds
- And each value is referenced by x[0] x[1] and of course y[0] y[1]
- Now the grid can have true “Cells”
- The corners of one cell do not have to lie on the corners of the neighbouring cells



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Regular grids in action

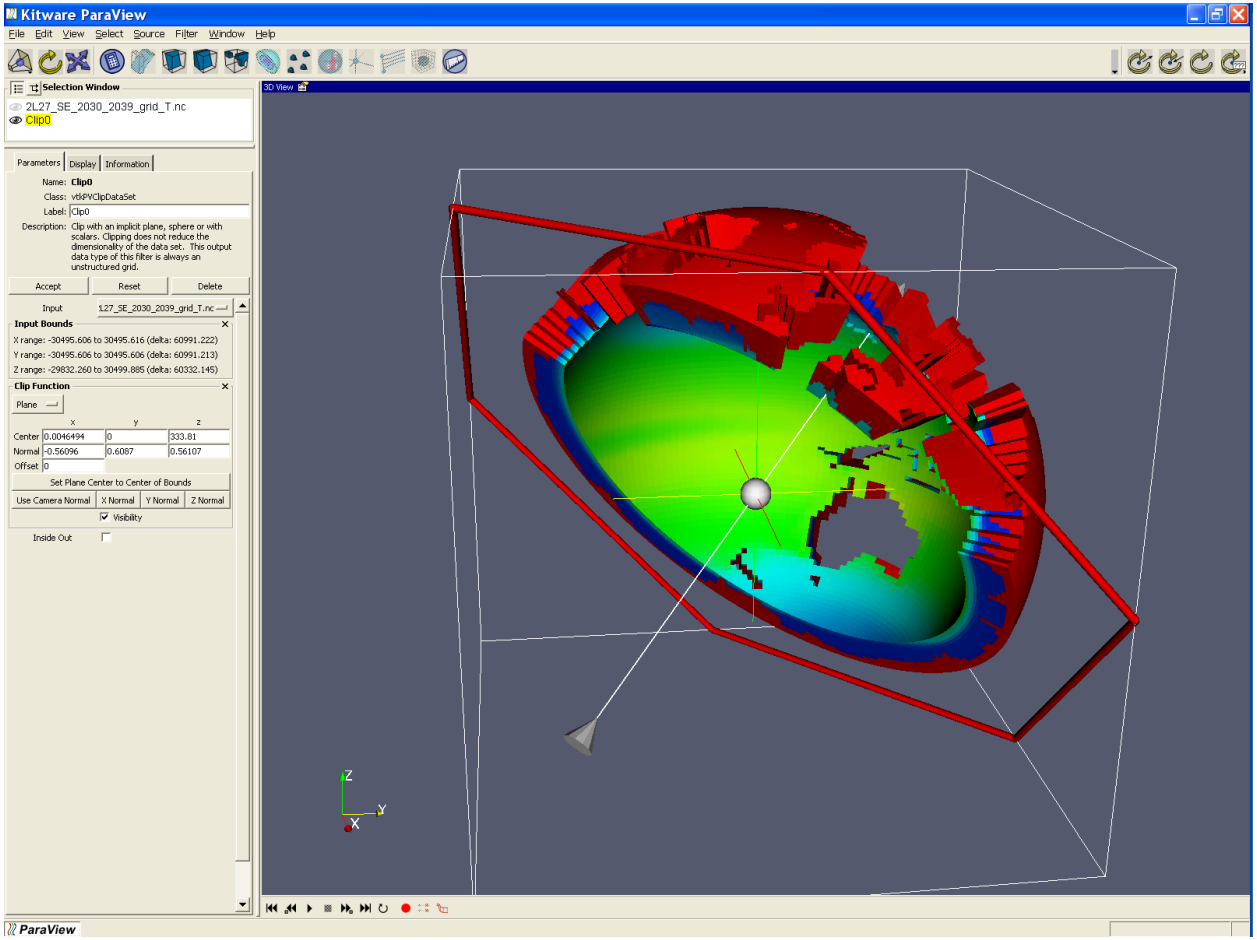
- **An example of daily meteo output : Relative humidity**



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Arbitrary mesh/grids in action

- **An ocean temperature model (inside out -)**



Concluding remarks

- **Use an API**
  - Someone else has done most of the work
  - Everyone can share the data
  - You get features you could not have done alone

- **Store as much metadata about the data as you can**
  - You'll thank me later
  - PhD student data from N years ago might prove useful