Introduction to HDF5

Francesc Alted

Consultant and PyTables creator

Outline

- Some words about me
- What is HDF5?
- Basic file structure
  - Groups, datasets, attributes and links
- The software
  - The library
  - Other tools
- A short glimpse into the C and Python APIs

Slides Provenance

- First time that I do an introduction to HDF5
- The HDF Group has already made a great job introducing HDF4/HDF5 to the public
- Asked them for permission to reuse part of their material (don't like to reinvent the wheel)
- Added some additional slides based on my own experience

A slice from The HDF Group

Me & HDF5

- Started working with HDF5 back in 2002
  - Needed it to scratch my own itch
  - The PyTables project, based on HDF5, started shortly after
- Handle large series of tabular data efficiently
  - Buffered I/O for maximum throughput
  - Very fast selections (leverage Numexpr)
  - Column indexing for top-class speed queries
- PyTables Pro is the commercial version that allows me to continue improving the package

What is HDF5?

What is HDF5?

HDF stands for Hierarchical Data Format

- A file format for managing any kind of data
  - http://www.hdfgroup.org/HDF5/doc/H5.format.html
- Software system to manage data in the format

- Designed for high volume or complex data
- Designed for every size and type of system

Brief History of HDF

1987 At NCSA (University of Illinois), a task force formed to create an architecture-independent format and library:

AEHOO (All Encompassing Hierarchical Object Oriented format)

Became HDF

Early 1990's NASA adopted HDF for Earth Observing System project

1996 DOE's ASC (Advanced Simulation and Computing) Project began collaborating with the HDF group (NCSA) to create "Big HDF" (Increase in computing power of DOE systems at LLNL, LANL and Sandia National labs, required bigger, more complex data files).

"Big HDF" became HDF5.

1998 HDF5 was released with support from National Labs, NASA, NCSA

2006 The HDF Group spun off from University of Illinois as non-profit corporation

The HDF Group

Outstanding Features of HDF5

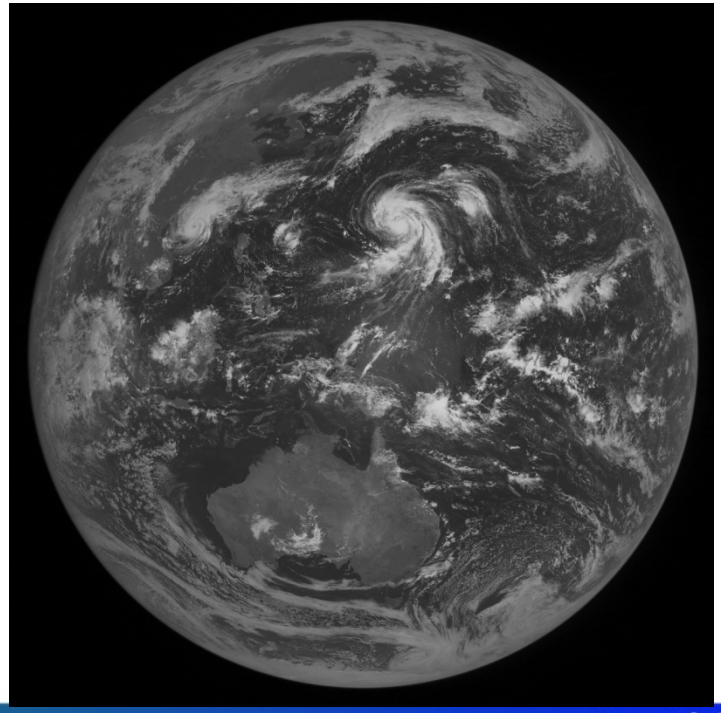
- Can store all kinds of data in a variety of ways
- Runs on most systems
- Lots of tools to access data
- Long term format support (HDF-EOS, CGNS)
- Library and format emphasis on I/O efficiency and different kinds of storage

Who uses HDF5?

- Applications that deal with big or complex data
- Over 200 different types of apps
- 2+million product users world-wide
- Academia, government agencies, industry

NASA EOS remote sense data

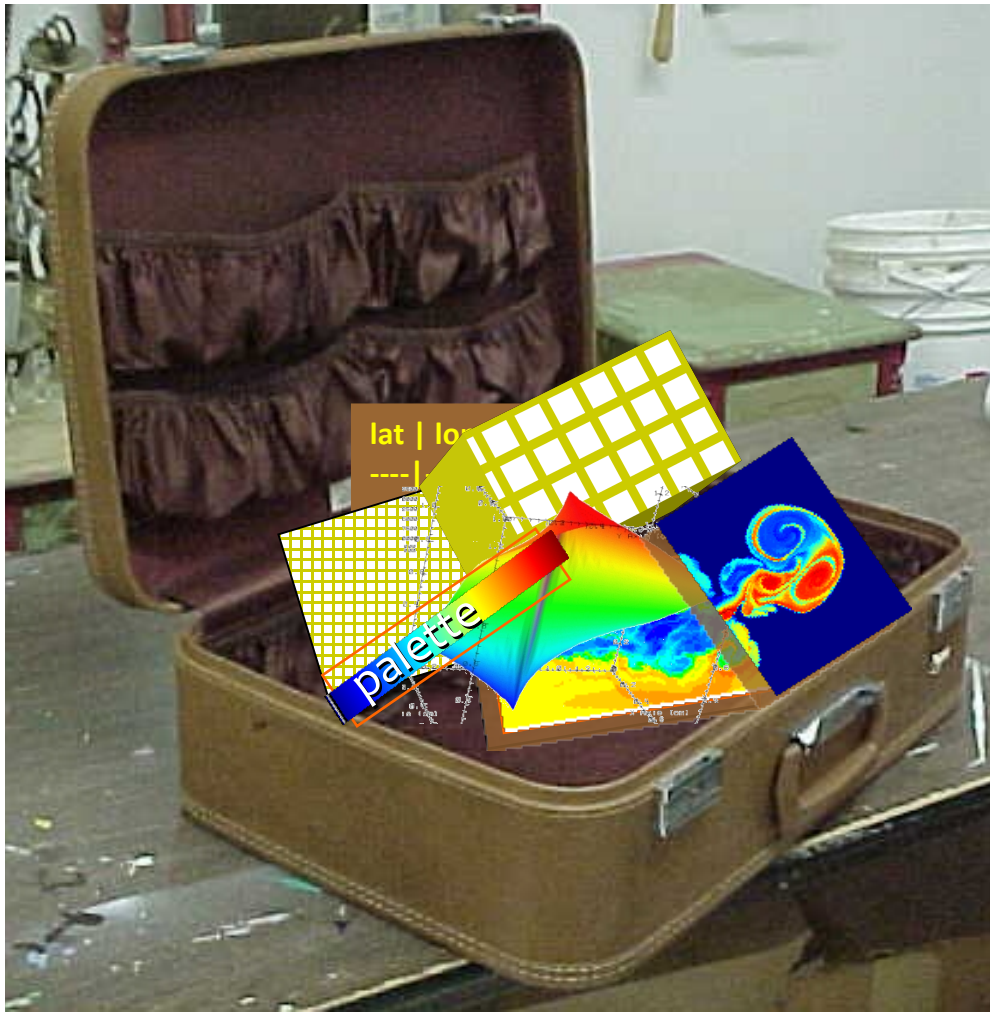
- HDF format is the standard file format for storing data from NASA's Earth Observing System (EOS) mission.
- Petabytes of data stored in HDF and HDF5 to support the Global Climate Change Research Program.



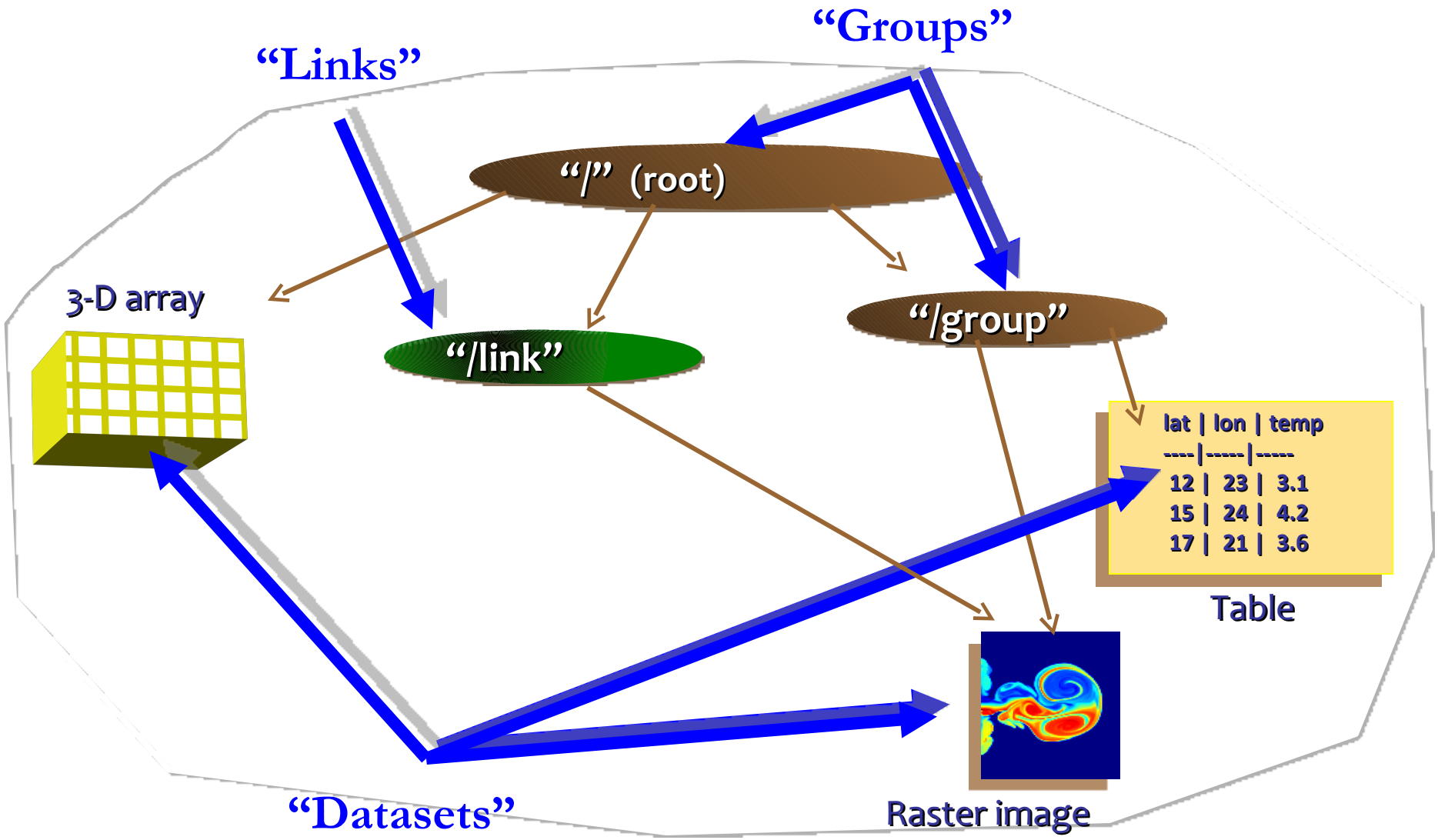
HDF5
Basic File Structure

An HDF5 “file” is a container…

…into which you can put your data objects



Structures to organize objects



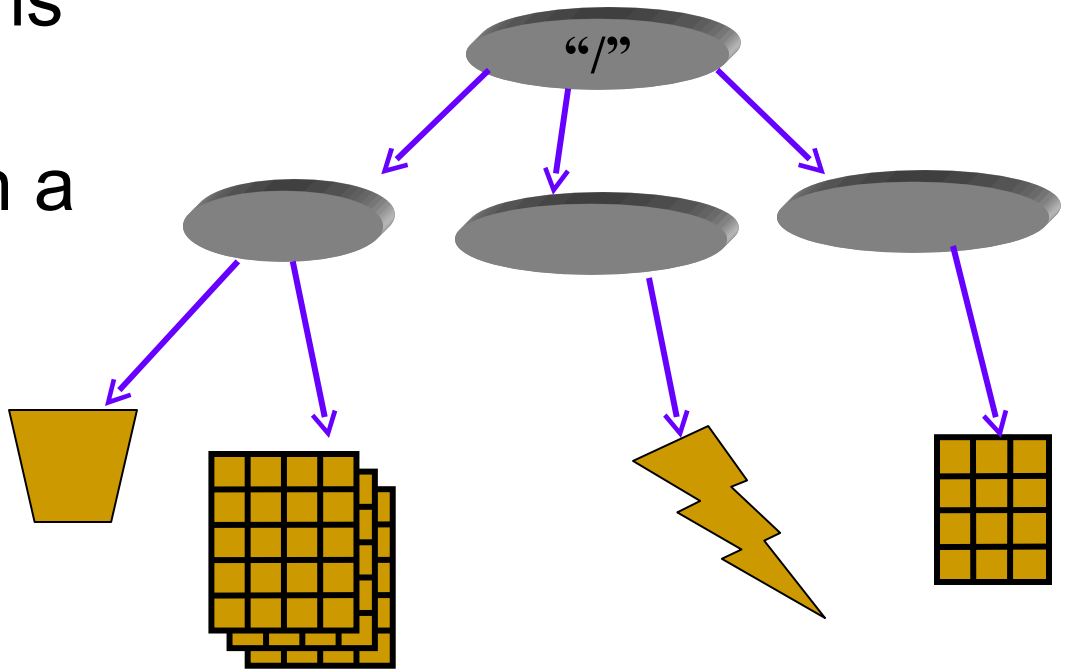
HDF5 model

- Groups – provide structure among objects
- Datasets – where the primary data goes
  - Rich set of datatype options
  - Flexible, efficient storage and I/O
- Attributes, for metadata annotations
- Links – point to other groups or datasets
  - Hard, soft and external flavors

Everything else is built essentially from these parts

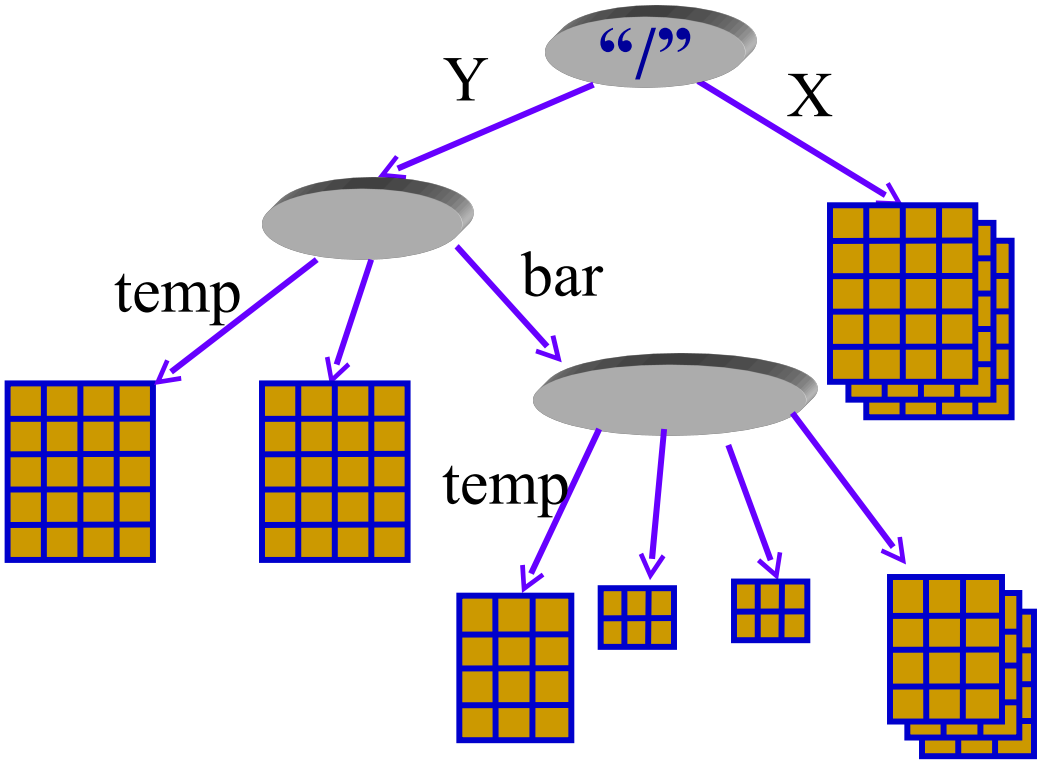
HDF5 Group

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

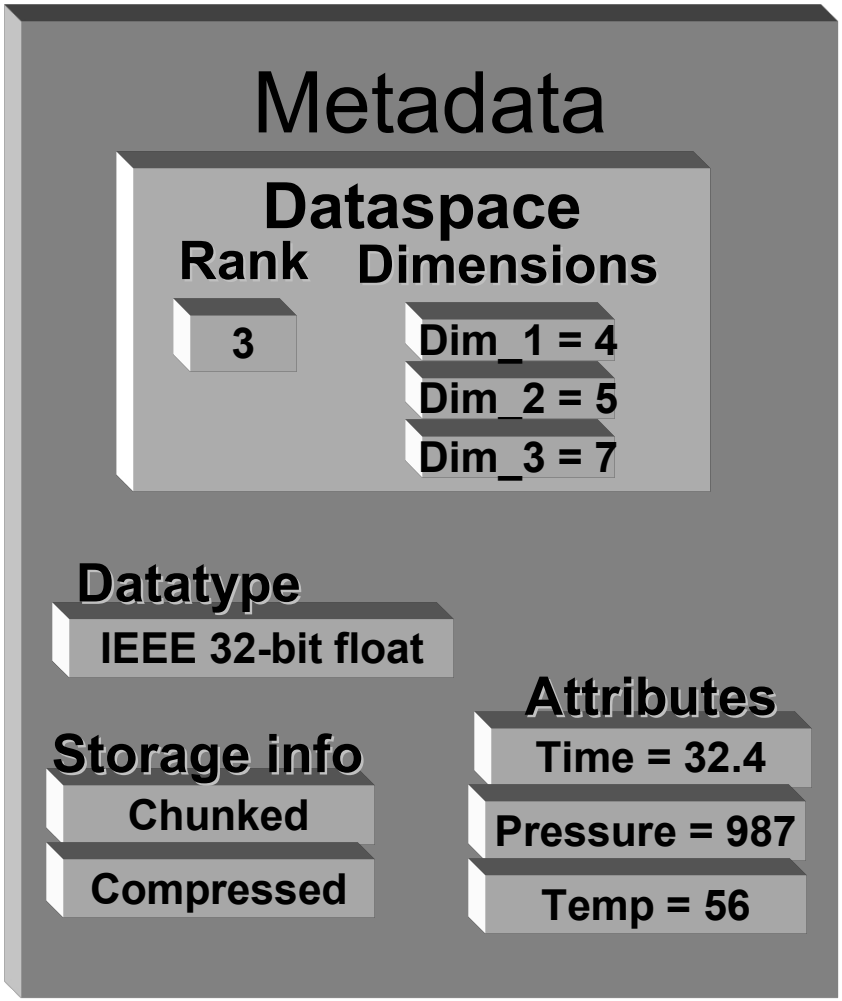


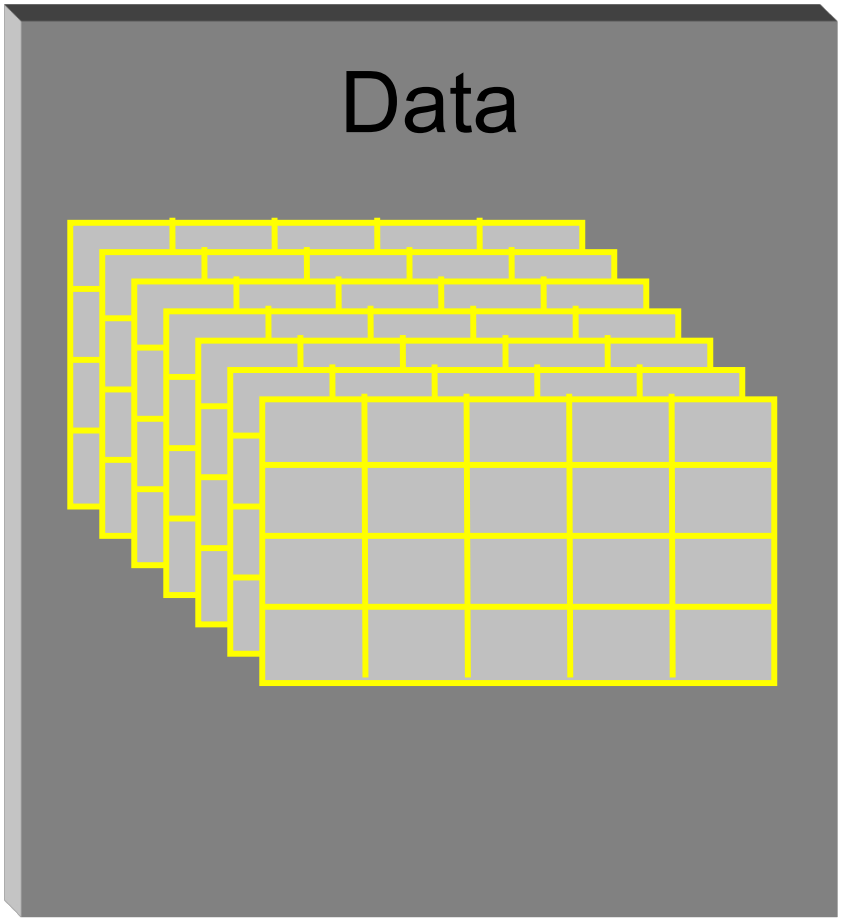
Path to HDF5 object in a file

/ (root)
/X
/Y
/Y/temp
/Y/bar/temp



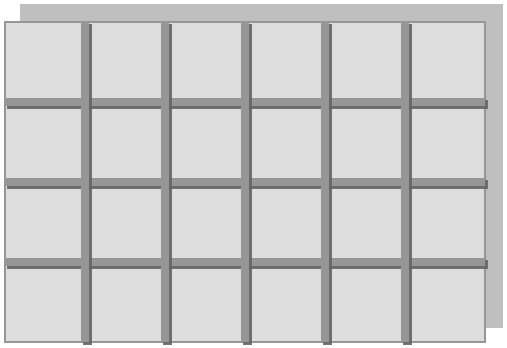
HDF5 Dataset





HDF5 Dataspace

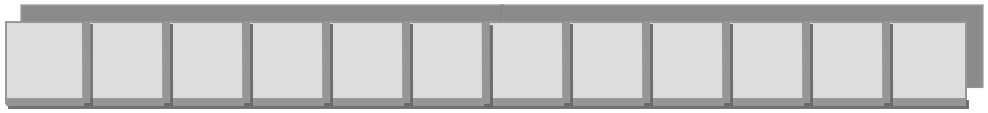
- Two roles
  - Dataspace contains spatial info about a dataset stored in a file
    - Rank and dimensions
    - Permanent part of dataset definition



Rank = 2

Dimensions = 4x6

  - Dataspace describes application's data buffer and data elements participating in I/O



Rank = 1

Dimensions = 12

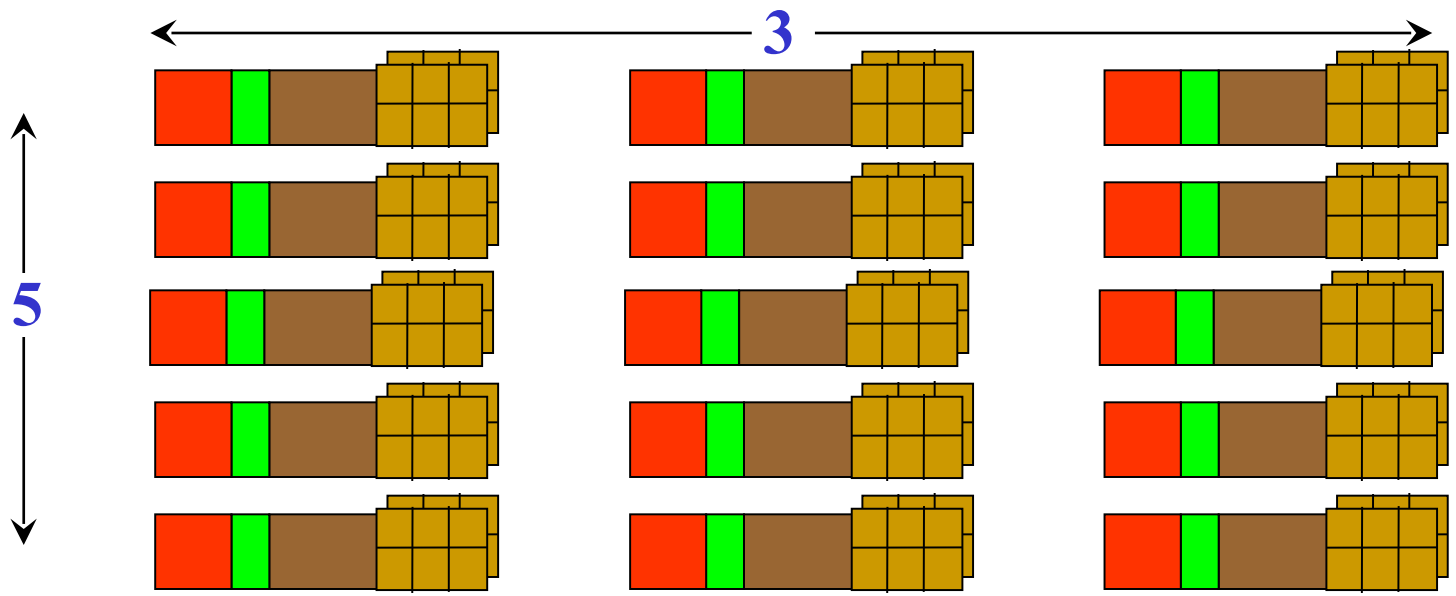
HDF5 Datatype

- Datatype – how to interpret a data element
  - Permanent part of the dataset definition
  - Two classes: **atomic** and **compound**

HDF5 Datatype

- HDF5 atomic types include
  - normal integer & float
  - user-definable (e.g., 13-bit integer)
  - variable length types (e.g., strings)
  - references to objects/dataset regions
  - enumeration - names mapped to integers
  - array
- HDF5 compound types
  - Comparable to C structs (“records”)
  - Members can be atomic or compound types

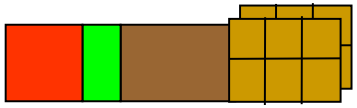
HDF5 dataset: array of records



Dimensionality: 5 x 3

Datatype:

int8 int4 int16 2x3x2 array of float32



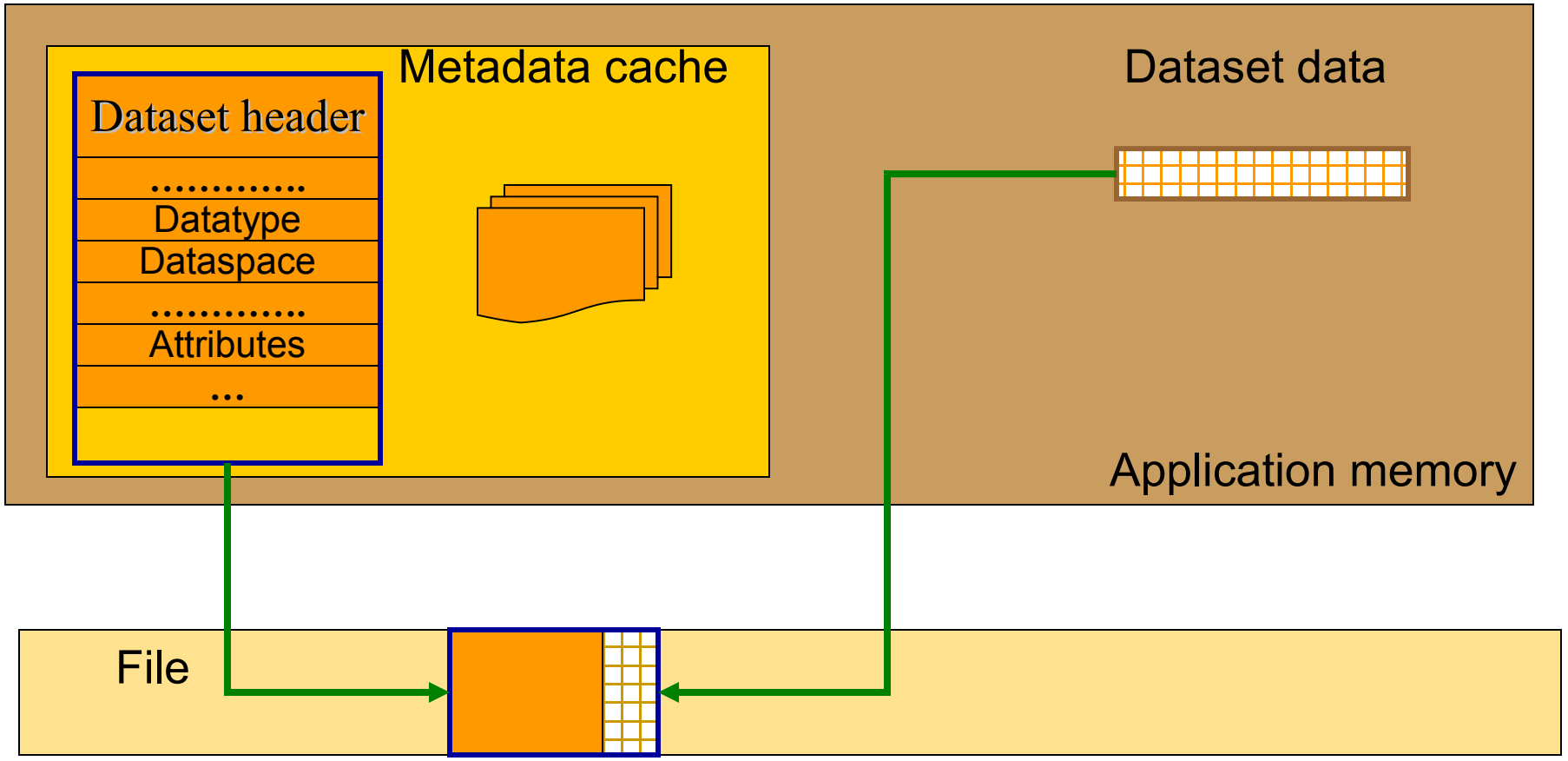
Record

HDF5 dataset storage layouts

- Compact
- Contiguous
- Chunked

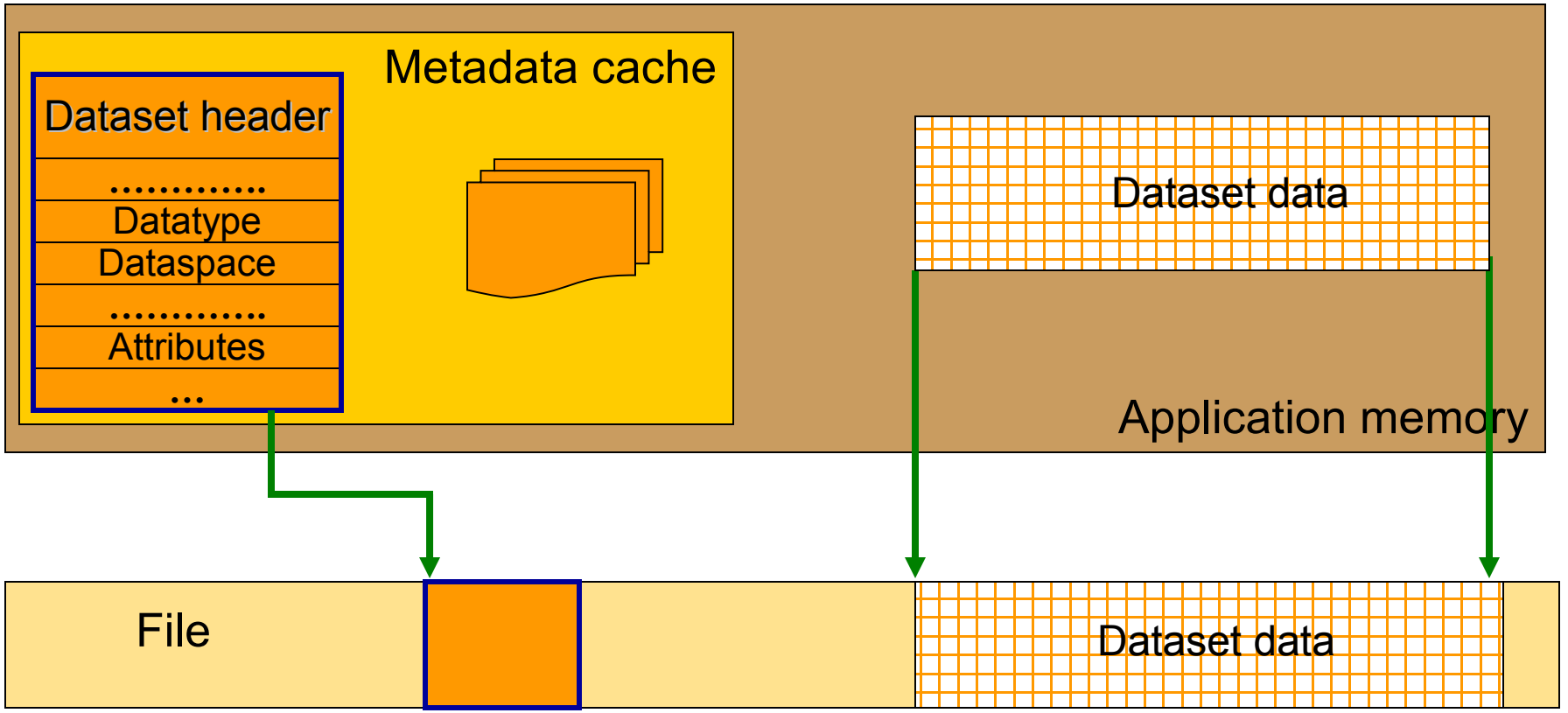
Compact storage layout

- Dataset data and metadata stored together in the object header



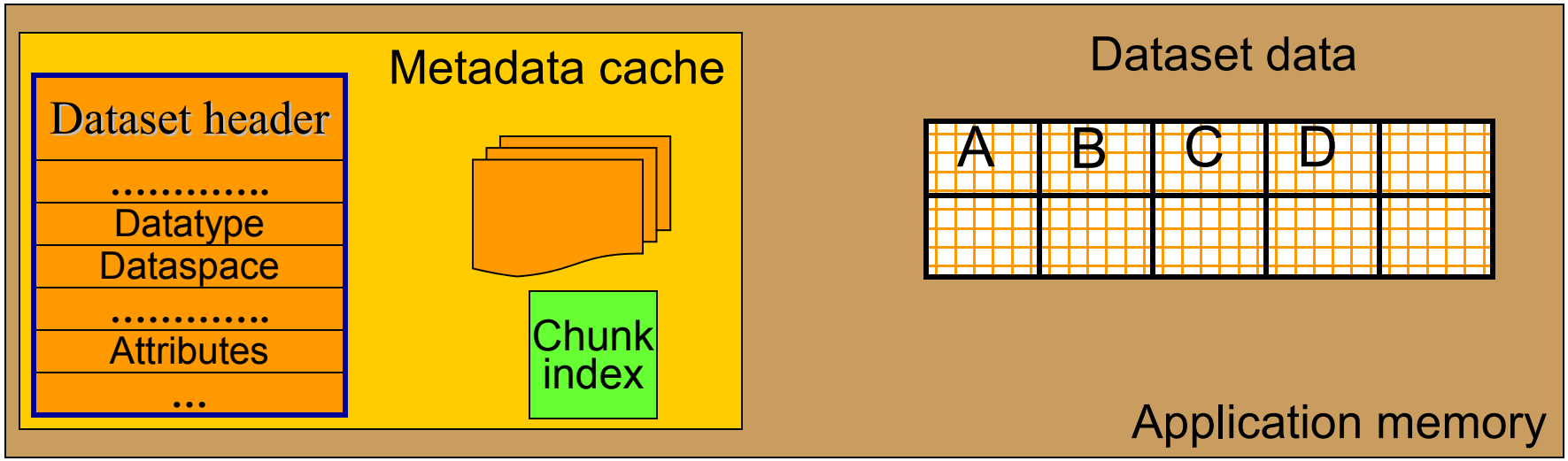
Contiguous storage layout

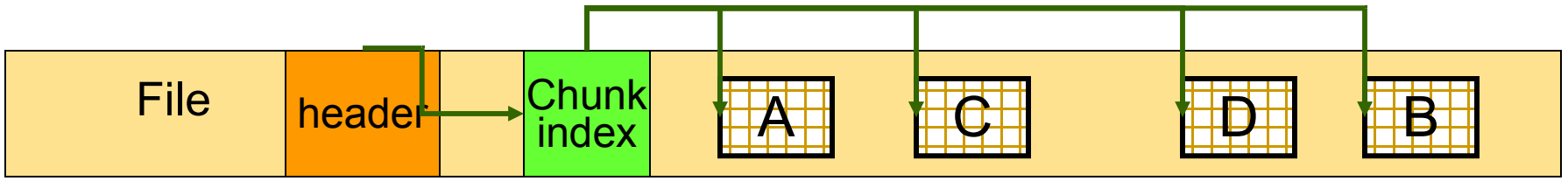
- Metadata header separate from dataset data
- Data stored in one contiguous block in HDF5 file



Chunked storage layout

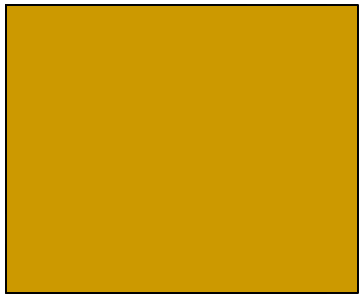
- Dataset data divided into equal sized blocks (chunks)
- Each chunk stored separately as a contiguous block in HDF5 file





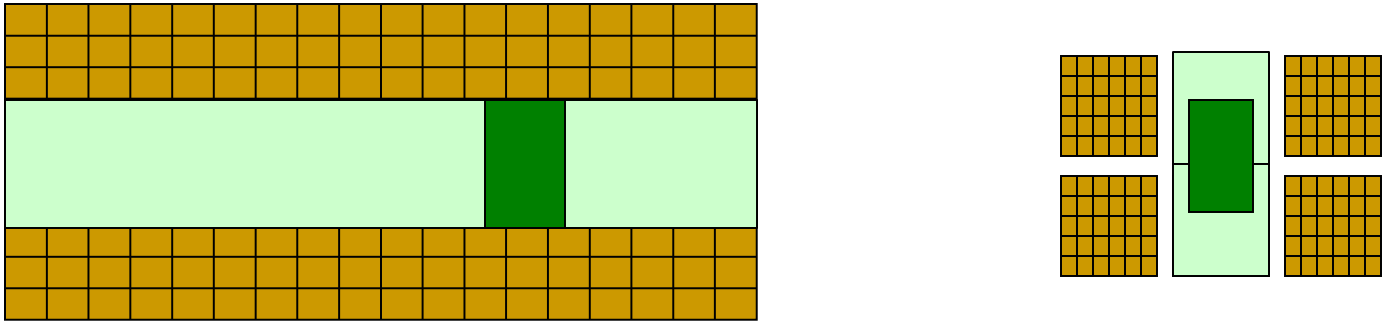
Why HDF5 Chunking?

- Chunking is required for several HDF5 features
  - Enabling compression and other filters like checksum
  - Extendible datasets



Why HDF5 Chunking?

- If used appropriately chunking improves partial I/O for big datasets

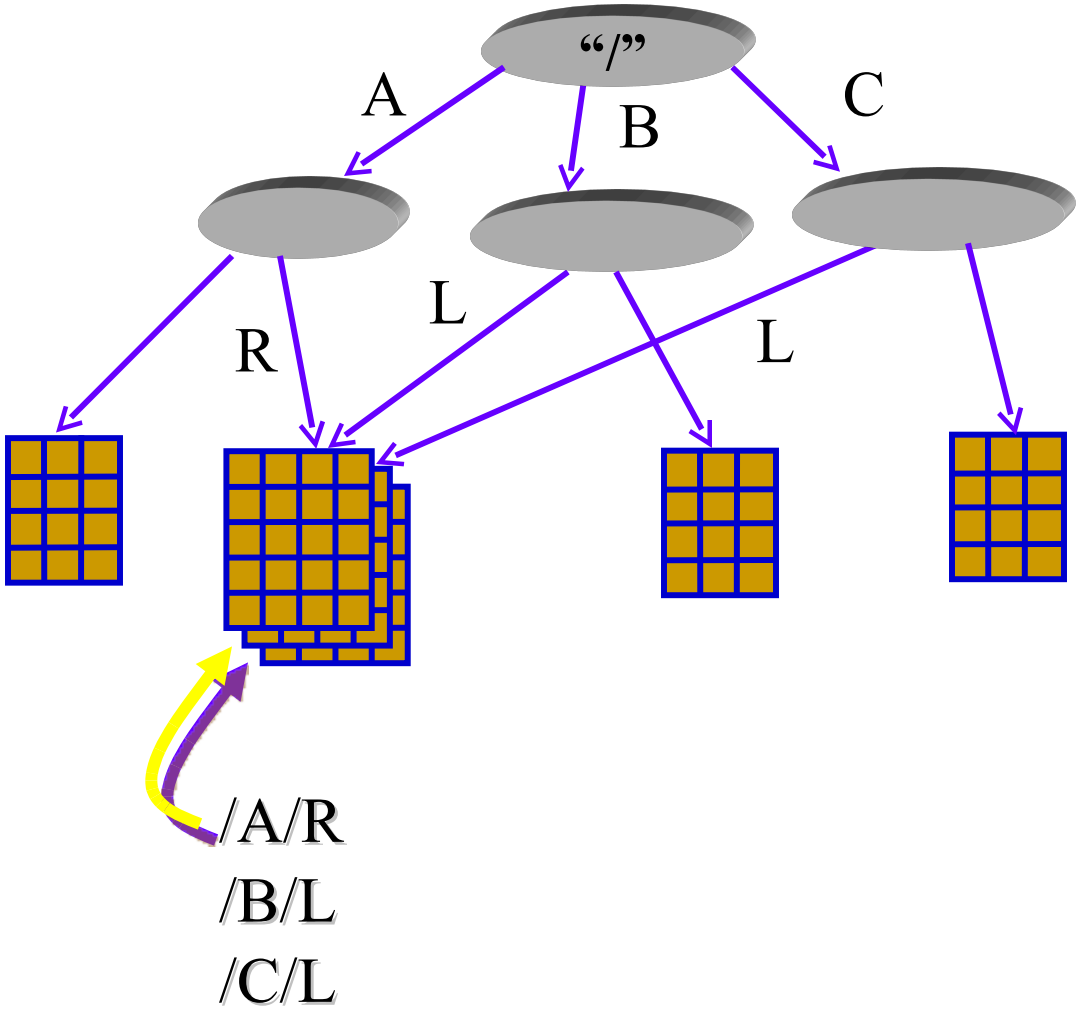


Only two chunks are involved in I/O

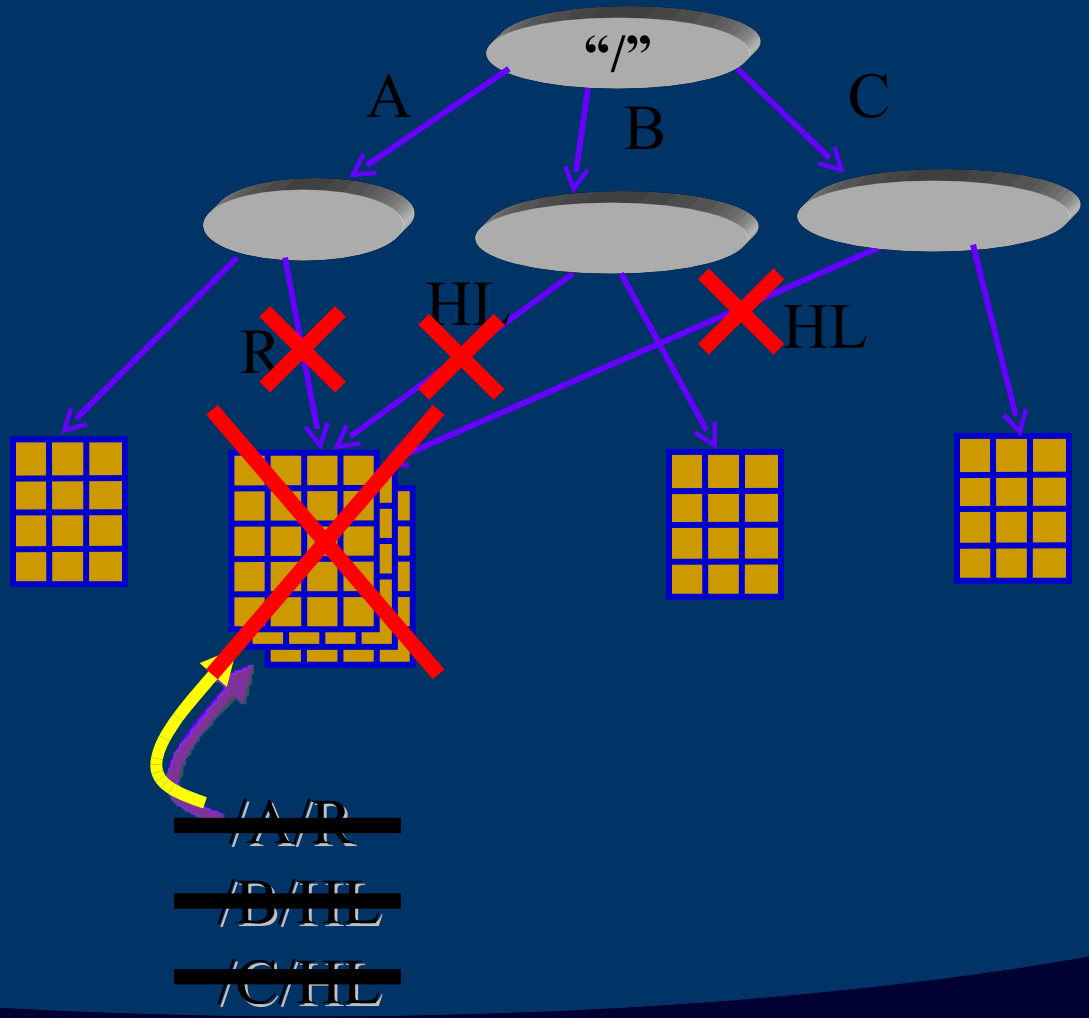
HDF5 Attribute

- Attribute – data of the form “name = value”, attached to an object by application
- Operations similar to dataset operations, but …
  - Not extendible
  - No compression or partial I/O
- Can be overwritten, deleted, added during the “life” of a dataset or a group (but not to a link)

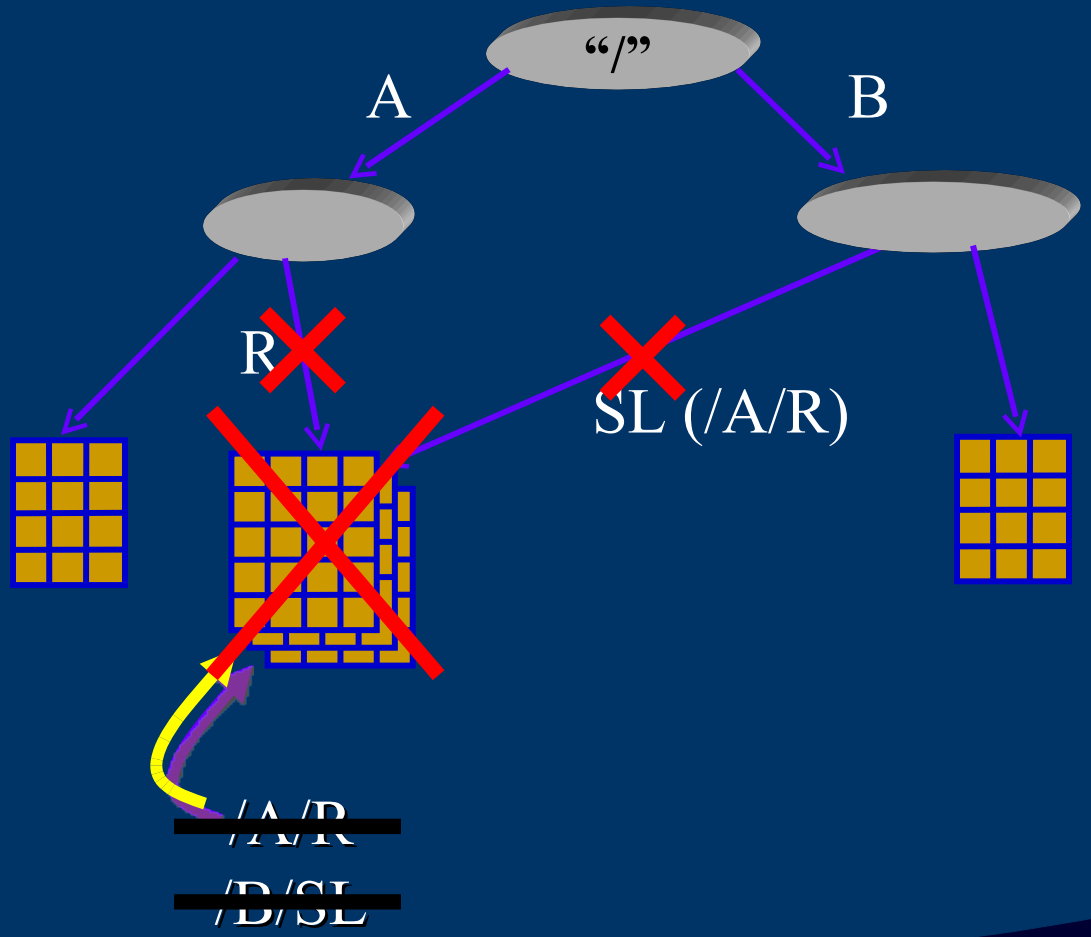
HDF5 links



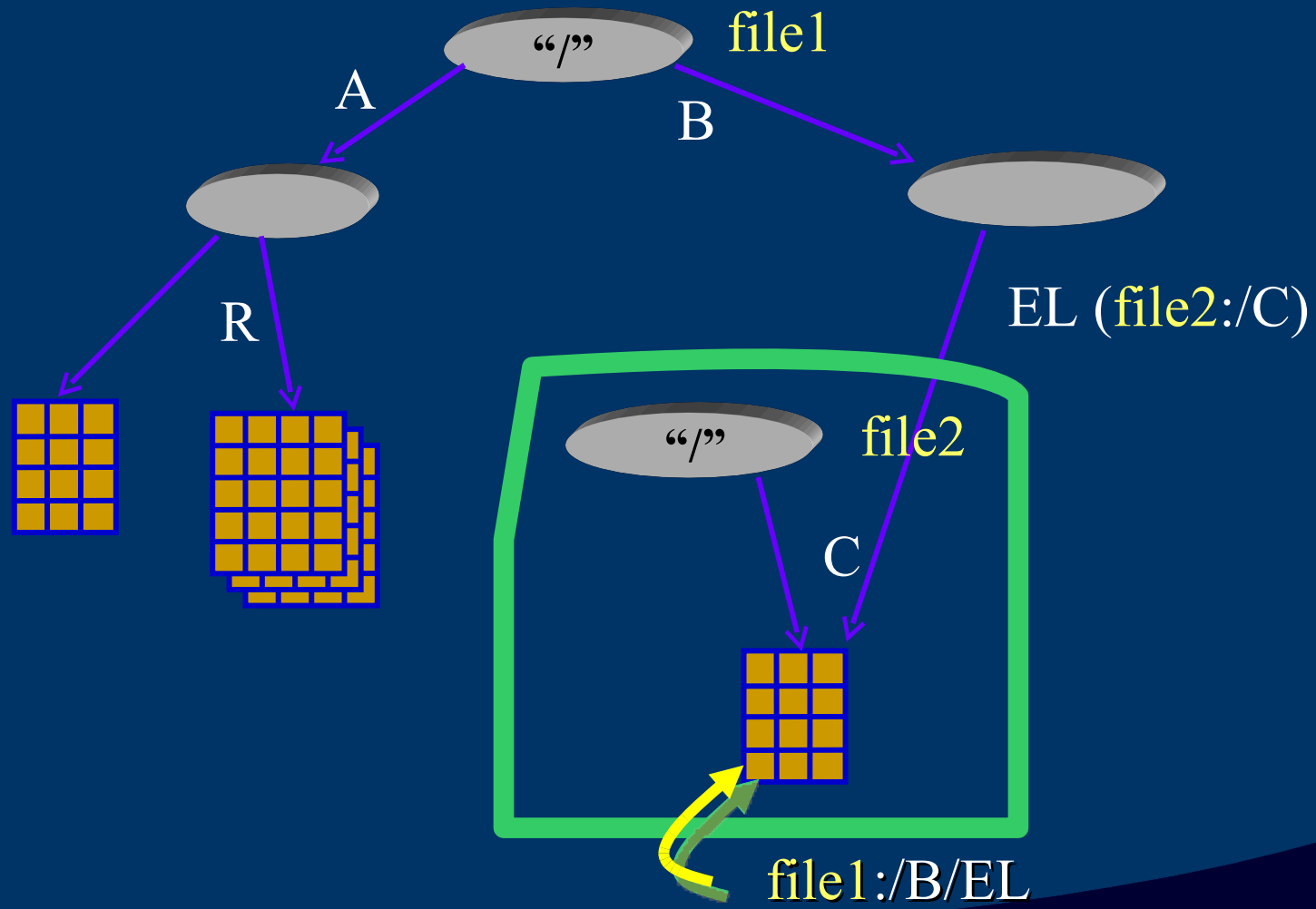
Hard links



Soft links

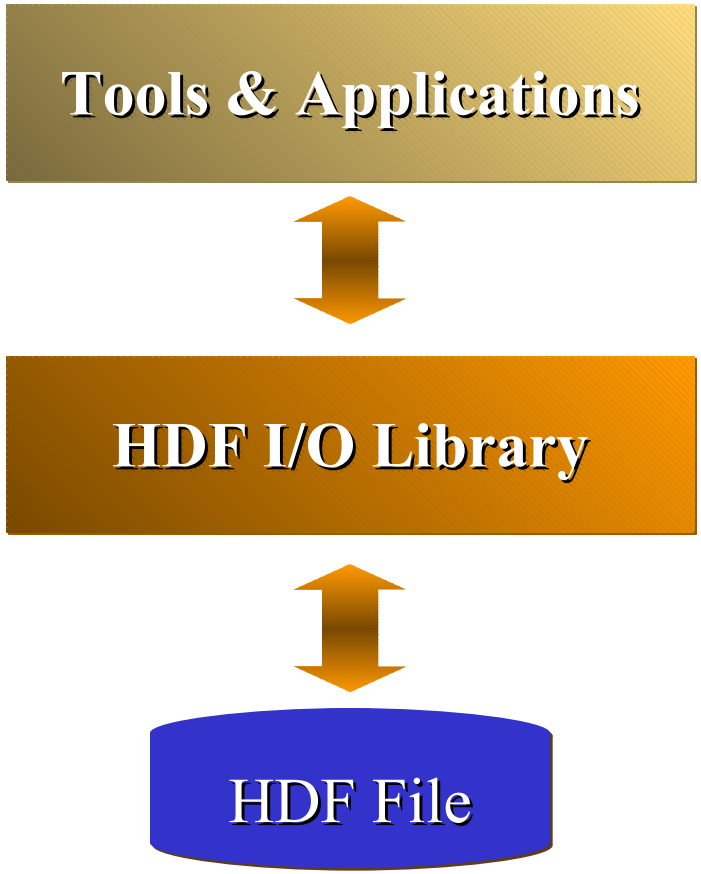


External links

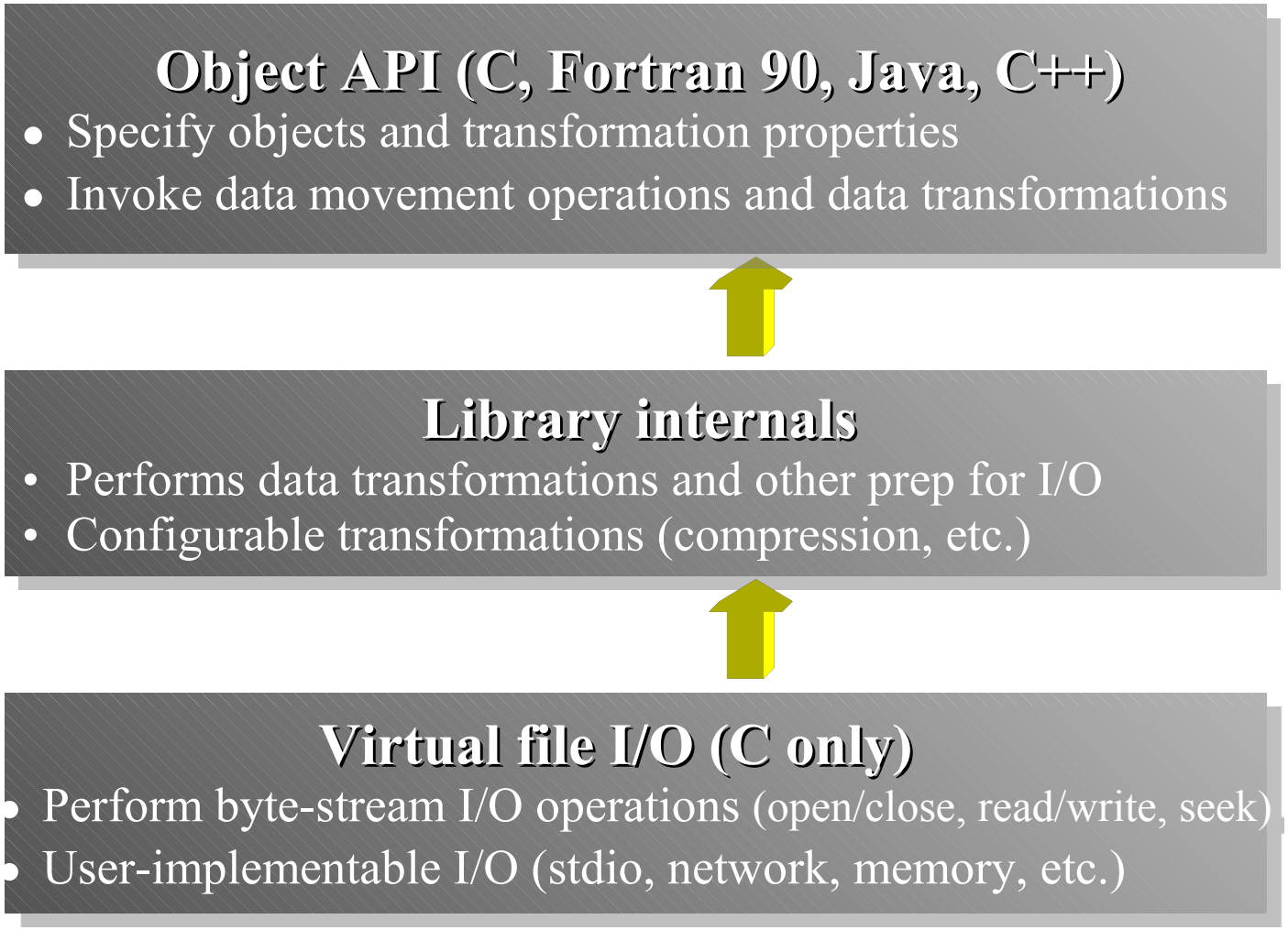


HDF5 Software

HDF5 software stack



Structure of HDF5 Library

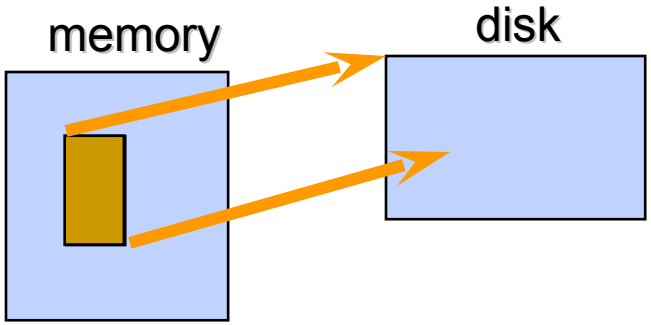


HDF5 Library Features

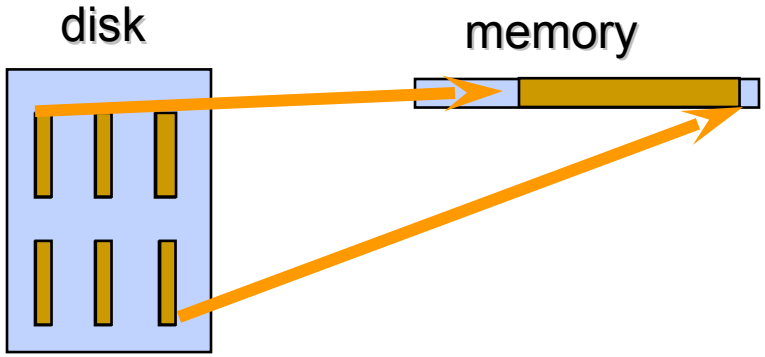
- HDF5 Library provides capabilities to
  - Describe subsets of data and perform write/read operations on subsets
    - Hyperslab selections and partial I/O
  - Layered architecture
    - Virtual I/O layers (ex. parallel I/O)
  - Use efficient storage mechanism to achieve good performance while writing/reading subsets of data
    - Chunking, compression

Partial I/O

Move just part of a dataset



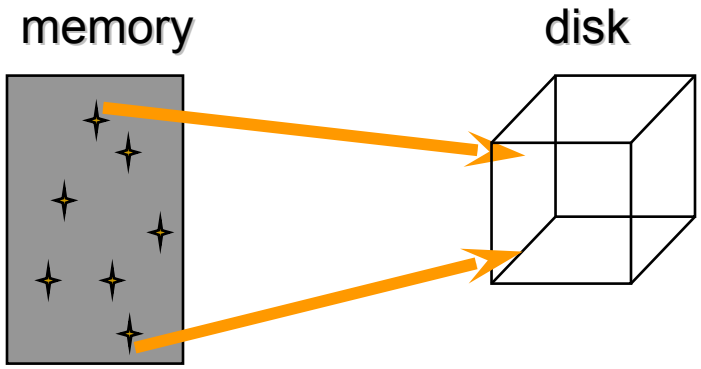
(a) Hyperslab from a 2D array to the corner of a smaller 2D array



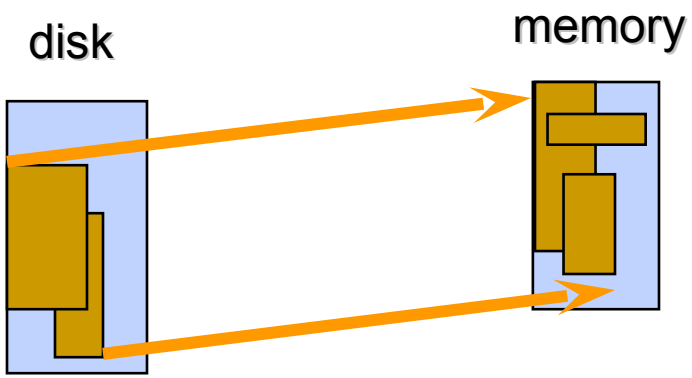
(b) Regular series of blocks from a 2D array to a contiguous sequence at a certain offset in a 1D array

Partial I/O

Move just part of a dataset

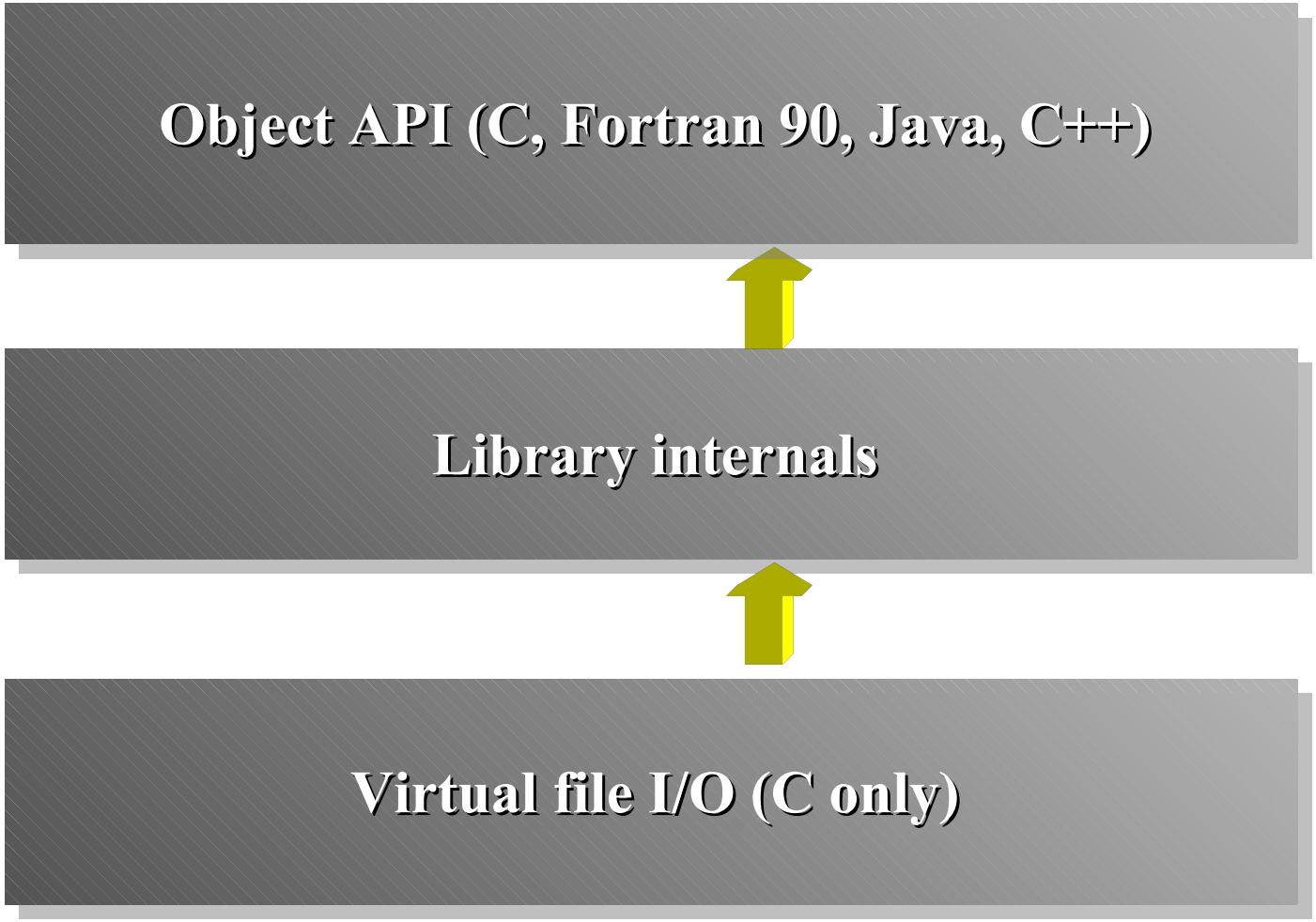


(c) A sequence of points from a 2D array to a sequence of points in a 3D array.



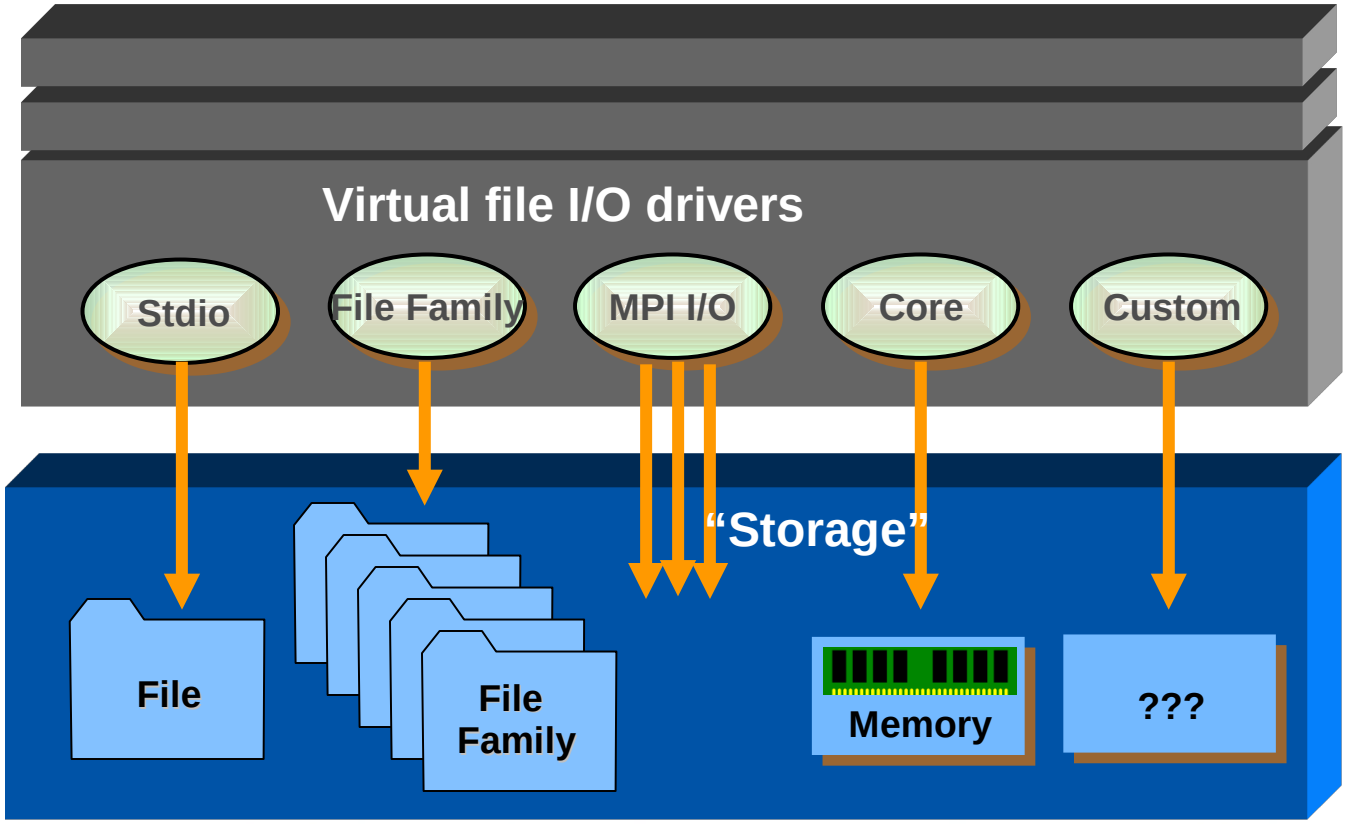
(d) Union of hyperslabs in file to union of hyperslabs in memory.

Virtual I/O layer

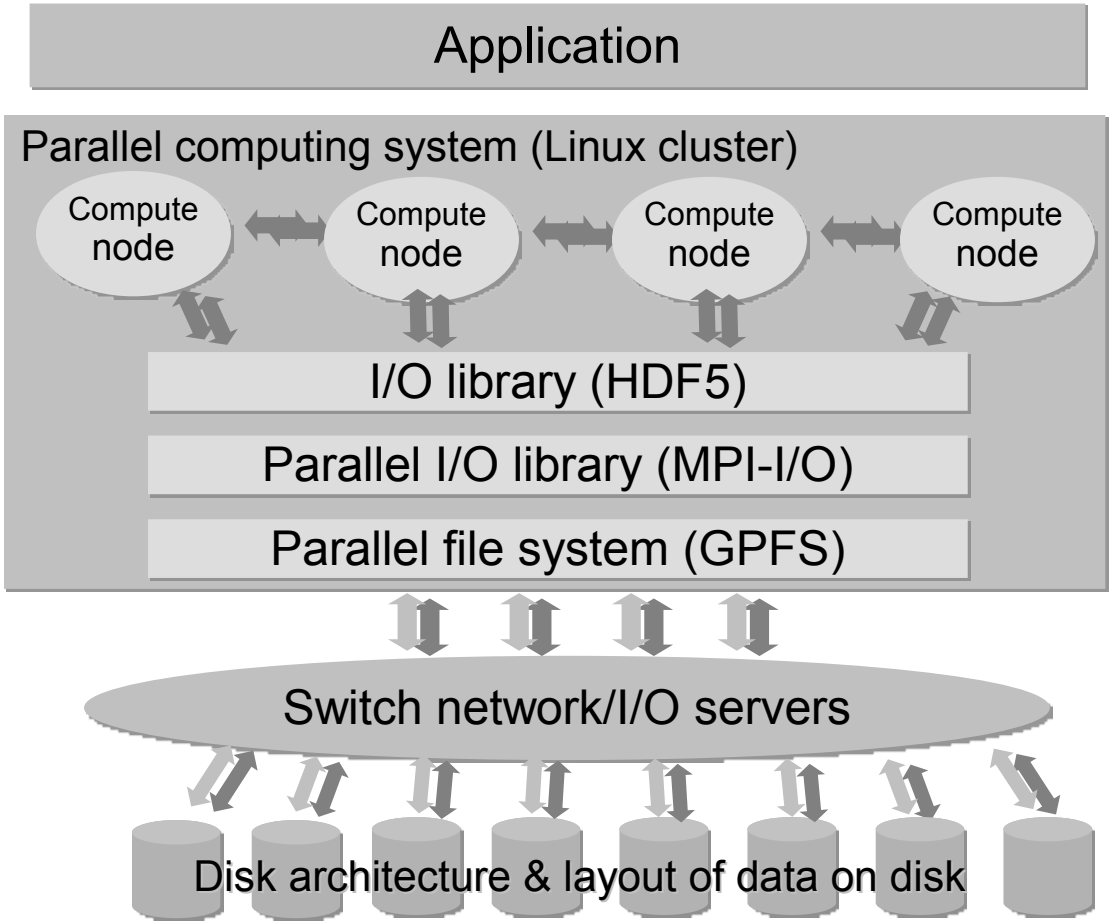


Virtual file I/O layer

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

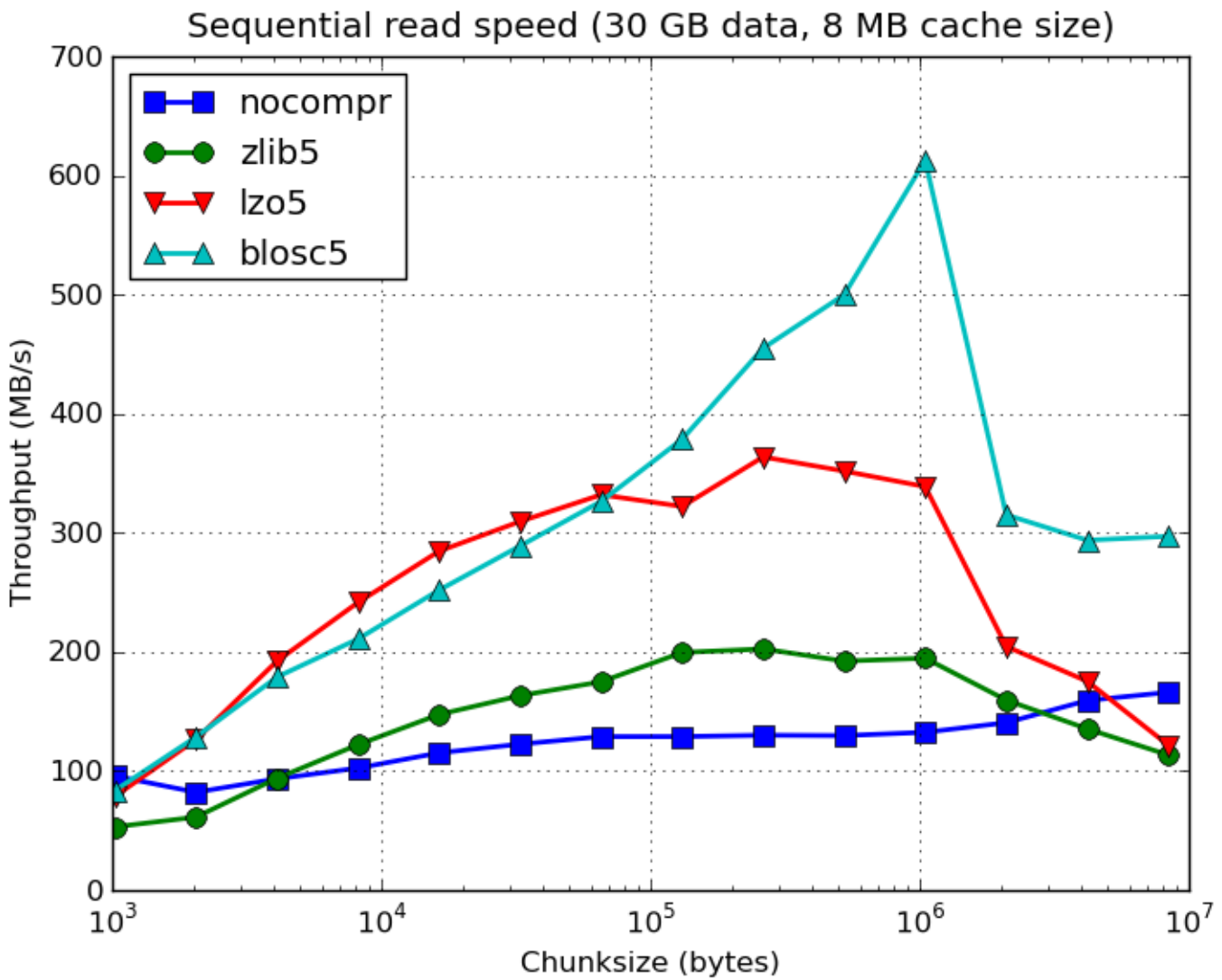


Layers – parallel example



I/O flows through many layers from application to disk.

Optimal chunksizes



Other Software

- The HDF Group
  - HDFView
  - Java tools
  - Command-line utilities (h5ls, h5dump, h5repack...)
  - Web browser plug-in
  - Regression and performance testing software
- 3rd Party (IDL, MATLAB, Mathematica, PyTables, h5py, ViTables, HDF Explorer, LabView)
- Communities (EOS, ASC, CGNS)
- Integration with other software (OpeNDAP)

A short glimpse into the HDF5 APIs (C & Python)

The General HDF5 API

- Currently **C,** Fortran 90, Java, and C++ bindings.
- C routines begin with prefix H5?

  ? is a character corresponding to the type of object the function acts on

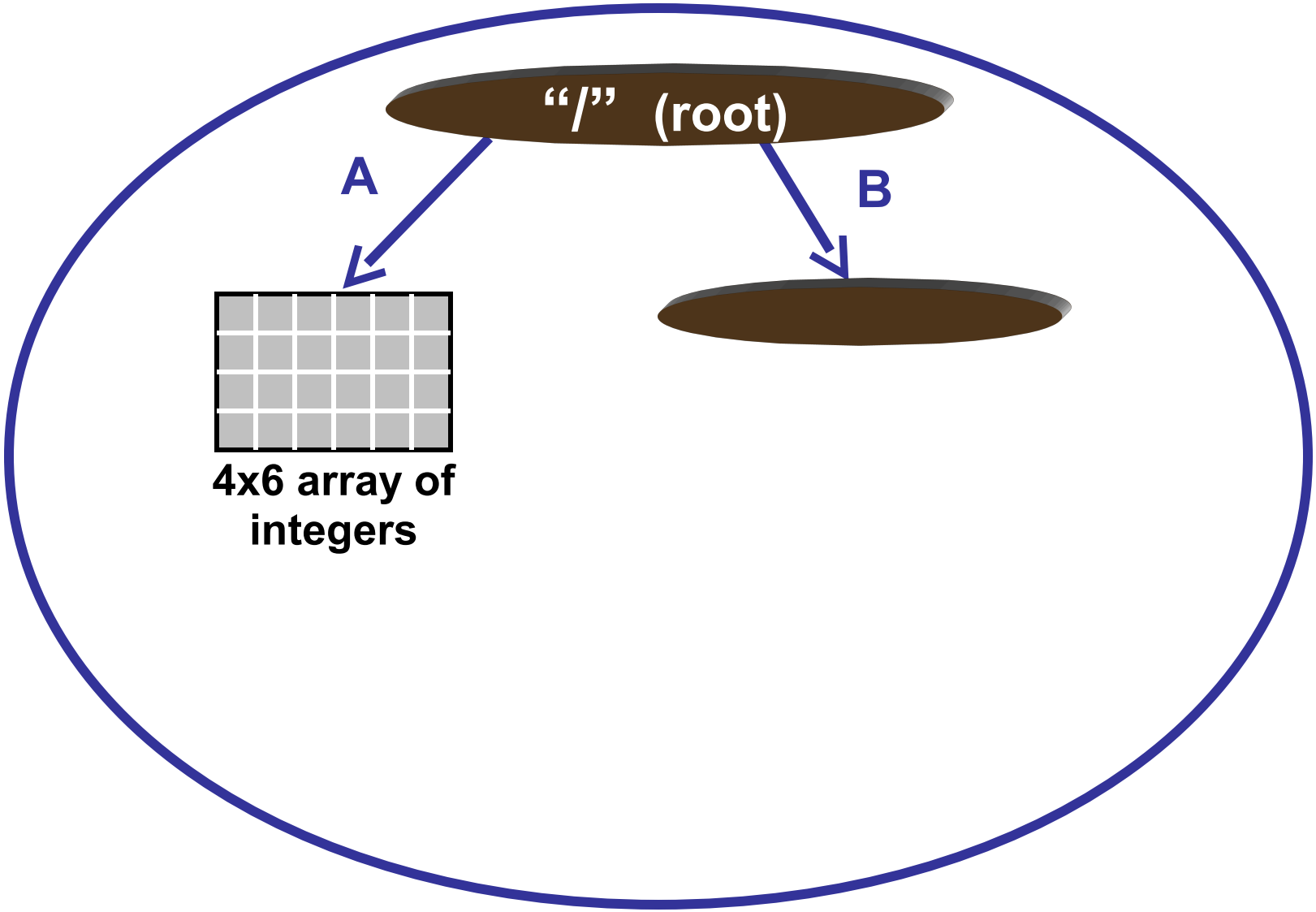
Example APIs:

H5D : **D**ataset interface *e.g.,* **H5Dread**

H5F : **F**ile interface *e.g.,* **H5Fopen**

H5S : data**S**pace interface *e.g.,* **H5Sclose**

Example: Create this HDF5 File



Code: Create a Dataset

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

4   file_id = H5Fcreate ("file.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);
```

rank

current dims

Create a dataset

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

pathname

datatype

dataspace

property list (default)

Terminate access to dataset, dataspace, file

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

Code: Create a Group

```
hid_t file_id, group_id;
...
/* Open “file.h5” */
file_id = H5Fopen(“file.h5”, H5F_ACC_RDWR,
                                H5P_DEFAULT);

/* Create group "/B" in file. */
group_id = H5Gcreate(file_id, "/B", H5P_DEFAULT,
                        H5P_DEFAULT);

/* Close group and file. */
status = H5Gclose(group_id);
status = H5Fclose(file_id);
```

Code: Create a Dataset and Group (Python/PyTables)

```
1  import tables as tb

2  file_id = tb.openFile (”file.h5", “w”)

3  dataset_id = file_id.createCArray(
                    “/”, ”A",
                    tb.Int32Atom,
                    shape=(4,6))

4   group_id = file_id.createGroup("/", "B")

# You don't need to explicitly close dataset or group!

5  file_id.close()
```

HDF5 Information

HDF Information Center

http://www.hdfgroup.org

HDF Help email address

help@hdfgroup.org

HDF users mailing lists

news@hdfgroup.org

hdf-forum@hdfgroup.org

Questions?