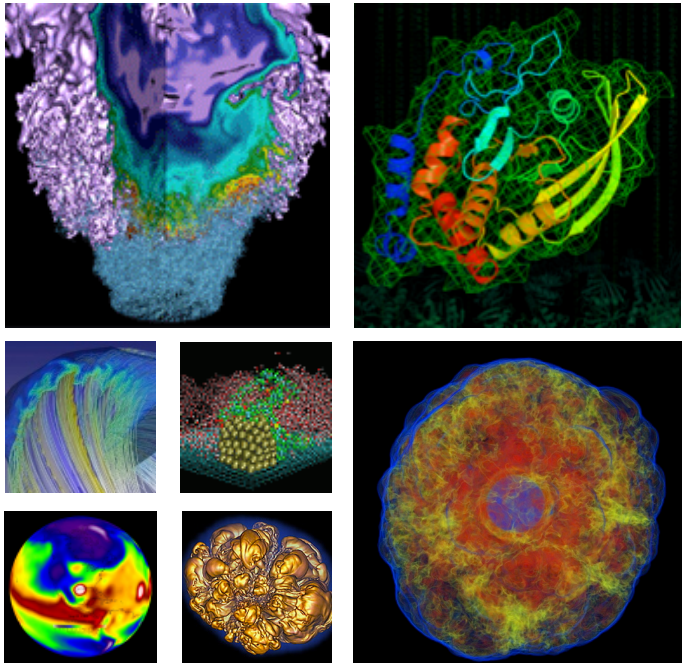
HDF5 and H5py Tutorial



Jialin Liu Data Analytics & Service Group

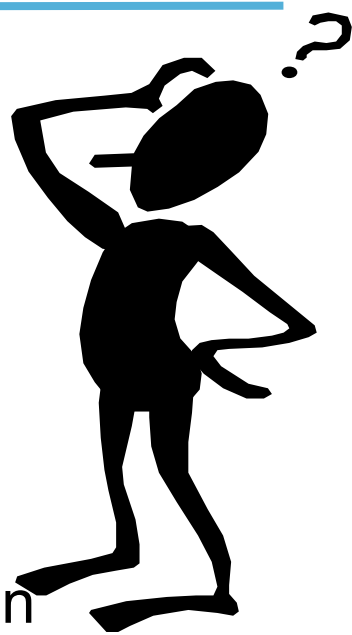
Feb 22, 2017

Goals

- Introduce you to HDF5
  - HDF5 data model
- Python Interface of HDF5: H5py
  - Basic usage
  - Best practice

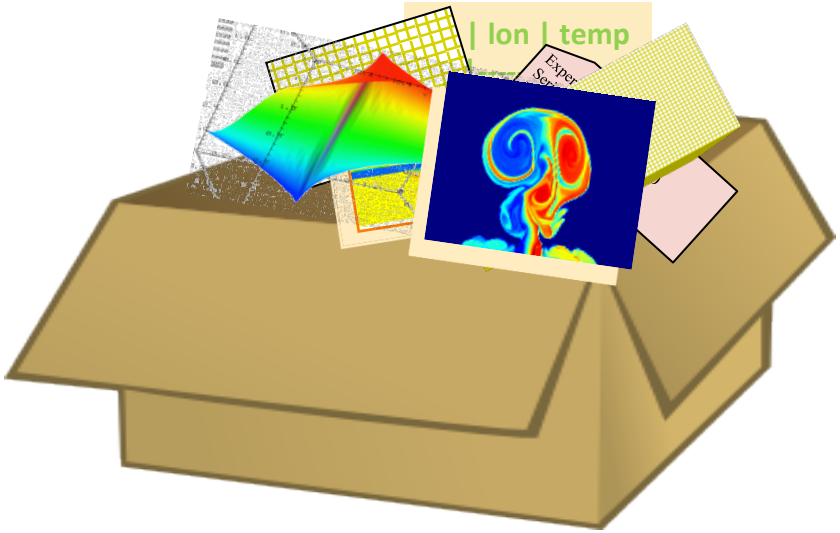
What is HDF5?

- HDF5 == Hierarchical Data Format, v5
- Open file format
  - Designed for high volume or complex data
- Open source software
  - Works with data in the format
- A data model
  - Structures for data organization and specification

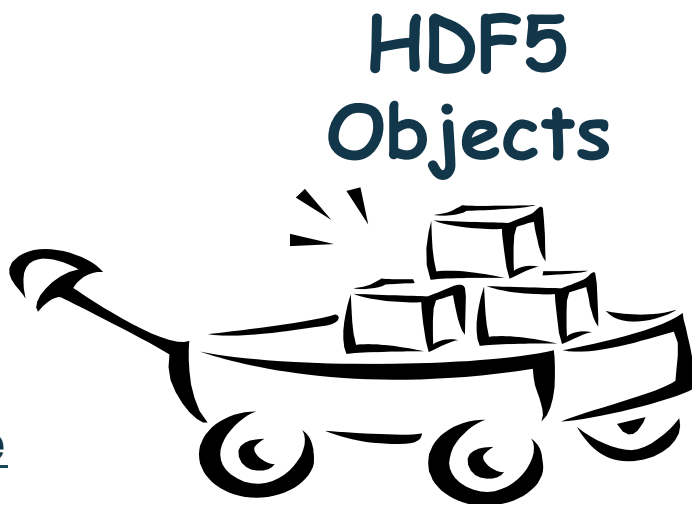


HDF5 File

An HDF5 file is a **container** that holds data objects.



HDF5 Data Model



Dataset

Group

Attribute

Link

Datatype

Dataspace

File

HDF5 Dataset

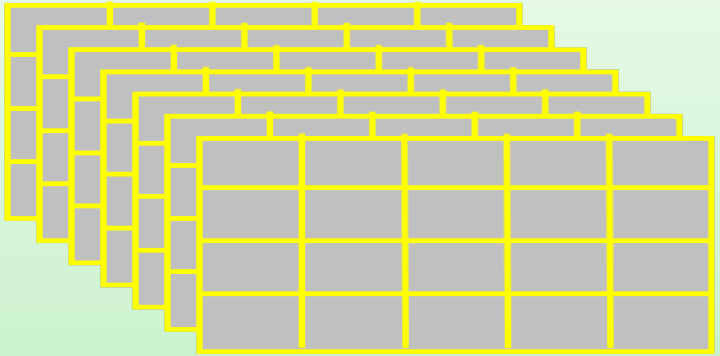
HDF5 Datatype

Integer: 32-bit, LE

HDF5 Dataspace

Rank	Dimensions
2	Dim[0] = 4
	Dim[1] = 5

Specifications for single data element and array dimensions

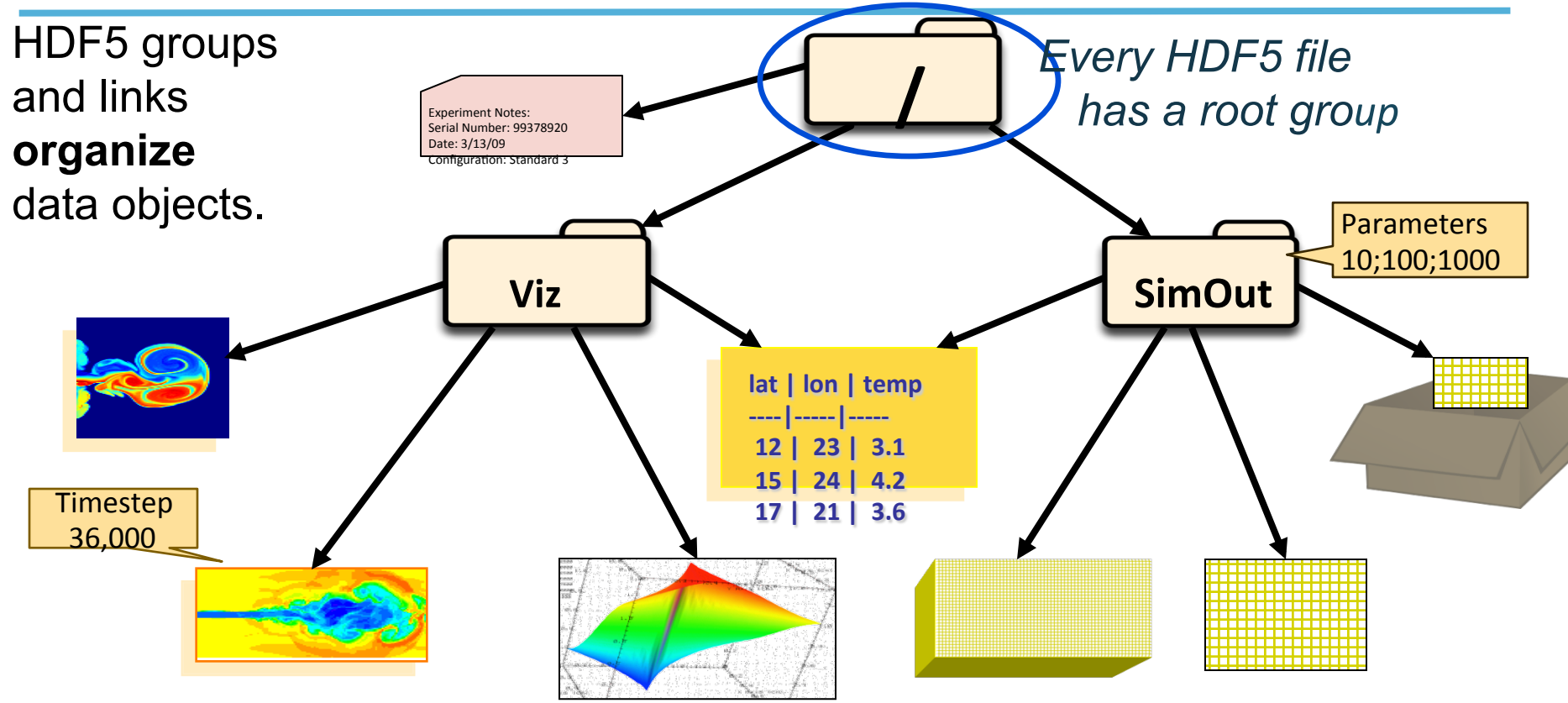


Multi-dimensional array of identically typed data elements

- HDF5 datasets **organize and contain** data elements.
  - HDF5 datatype describes individual data elements.
  - HDF5 dataspace describes the logical layout of the data elements.

HDF5 Groups and Links

HDF5 groups and links **organize** data objects.



HDF5 Attributes

- Typically contain user metadata
- Have a name and a value
- Attributes “decorate” HDF5 objects
- Value is described by a datatype and a dataspace
- Analogous to a dataset, but do not support partial I/O operations; nor can they be compressed or extended

> h5dump

H5py

The h5py package is a Pythonic interface to the HDF5 binary data format.

- H5py provides easy-to-use high level interface, which allows you to store huge amounts of numerical data,
- Easily manipulate that data from NumPy.
- H5py uses straightforward NumPy and Python metaphors, like dictionary and NumPy array syntax.

H5py at NERSC

```
module load python/2.7-anaconda
or
module load python/3.5-anaconda
```

Serial h5py

Anaconda includes h5py package

- H5py 2.6.0
- Built-in hdf5 library, 1.8.17
- Easy use with other packages
- *No parallel support*

Works on both Edison and Cori

H5py at NERSC

We provide **parallel h5py** at NERSC

```
module load python/2.7-anaconda
module load h5py-parallel
or
module load python/3.5-anaconda
module load h5py-parallel
```

H5py-parallel @ NERSC

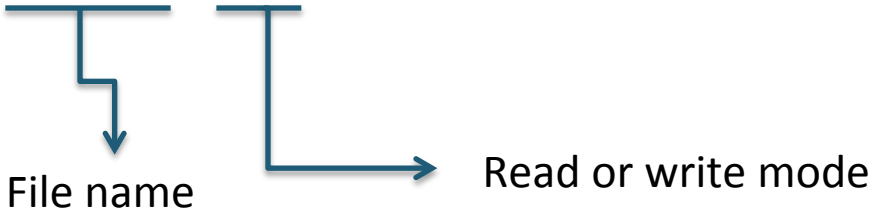
- H5py 2.6.0
- Compiled with cray-hdf5-parallel/1.8.16
- No conflict with anaconda's serial h5py
  - Import h5py (perfectly fine)
  - Can use together with anaconda
- Up to date features

Basic Usage

Basic Usage of H5py

- **Serial H5py**

```
1 import h5py
2 fx=h5py.File('output.h5','w')
```



Read with H5py

Basic Usage of H5py: Dealing with File

- **Parallel H5py**

```
1 from mpi4py import MPI
2 import h5py
3 fx=h5py.File('output.h5','w',driver='mpio',comm=MPI.COMM_WORLD)
```

Need MPI for parallel h5py

No difference with serial h5py

Only on Compute node

Playing with the data: More than numpy

```
dh5   = h5py.File('4857-55711.h5')
dflux = dh5['4857/55711/4/coadd']['FLUX']
```

Use h5py in Astronomy

Path to the dataset

Dataset name

Returned a dataset object

Playing with the data: More than numpy

```
dh5   = h5py.File('4857-55711.h5')
dflux = dh5['4857/55711/4/coadd']['FLUX']
```

Slicing

```
dall = dh5['4857/55711/4/coadd'][()]
```

Chunking

```
>>> dset = f.create_dataset("chunked", (1000, 1000), chunks=(100, 100))
```

Compression

```
>>> dset = f.create_dataset("zipped", (100, 100), compression="gzip")
```

Best Practice

Strive for Best Performance!
at NERSC

Really Challenging: More than Single IO Layer

h5py

mpi4py

numpy

1. Optimal HDF5 file creation

```
1 f = h5py.File('name.hdf5', libver='earliest') # most compatible
2 f = h5py.File('name.hdf5', libver='latest')   # most modern
```

2.25X

Choose the most modern format [optional]

2. Speedup the I/O with collective I/O

```
dset[start:end,:]=temp
```

Independent IO

```
1 with dset.collective:
2  dset[start:end,:]=temp
```

Collective IO

2X

Collective IO reduces the IO contention on server side

3. Use low-level API in H5py

```
1 space=h5py.h5s.create_simple((100,))
2 plist=h5py.h5p.create(h5py.h5p.DATASET_CREATE)
3 plist.set_alloc_time(h5py.h5d.ALLOC_TIME_EARLY)
```

Get closer to the HDF5 C library, fine tuning

4. Avoid type casting on the fly

```
1 dset = f.require_dataset('field',shape,'f')
```

vs

```
1 dset = f.require_dataset('field',shape,'f8')
```

Reduces the IO time, from 527 seconds to 1.3 seconds when writing a 100x100x100 array with 5x5x5 procs

High Performance H5py with Sample Codes

http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/h5py/

Thanks

Stay Tuned