Features of HDF5

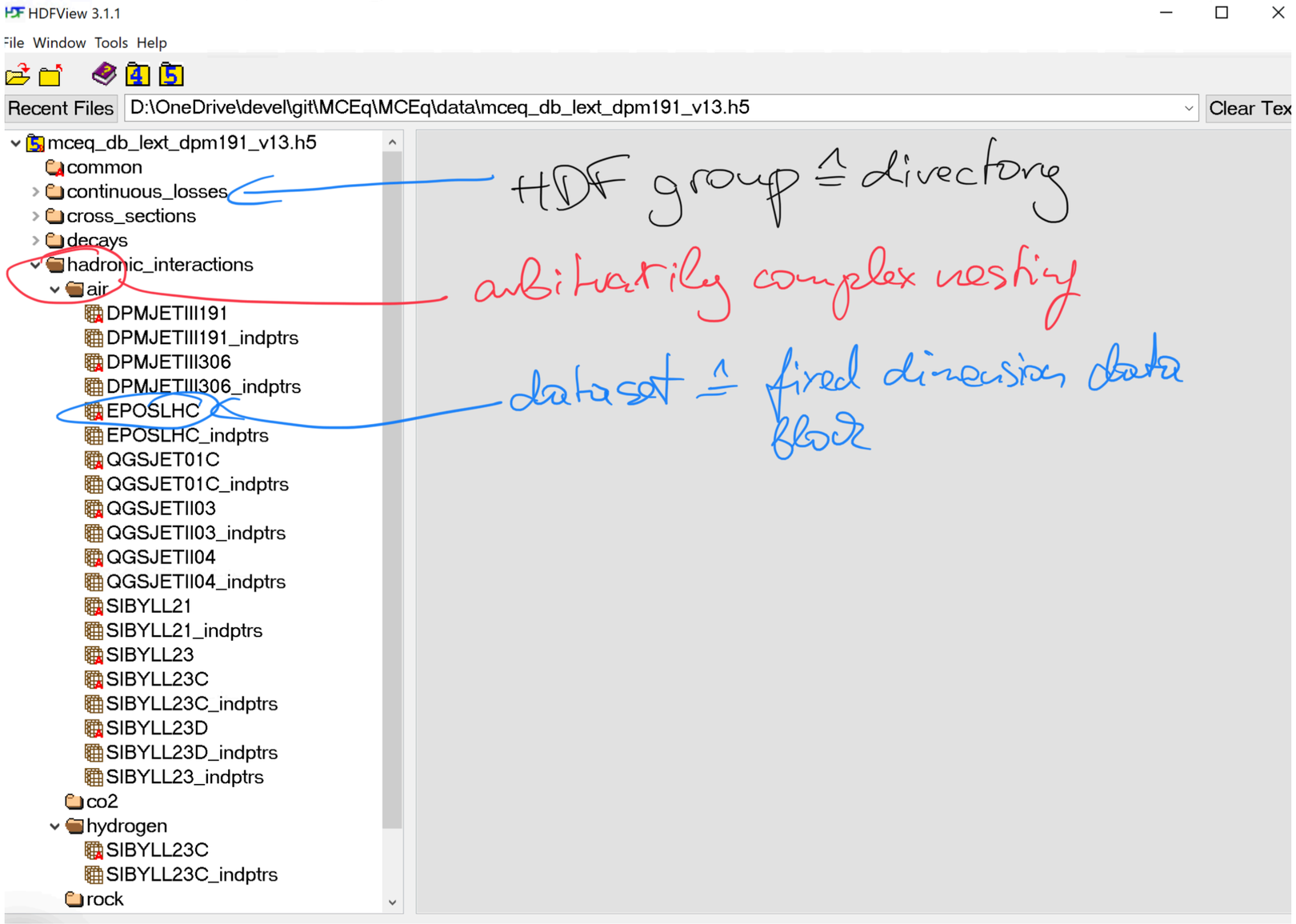
Anatoli Fedynitch, 2020/09/24

Key features of HDF5

- Hierarchical data format = like a file system
- Metadata + data together
- Cross-platform: Linux, Windows, Mac
- Multi-language: Fortran, C, C++, Java, Python, Julia, etc..; thread-safe
- Architectures (official): x86, amd64, ppc64; (community): arm64
- Open-source: BSD 3-clause license

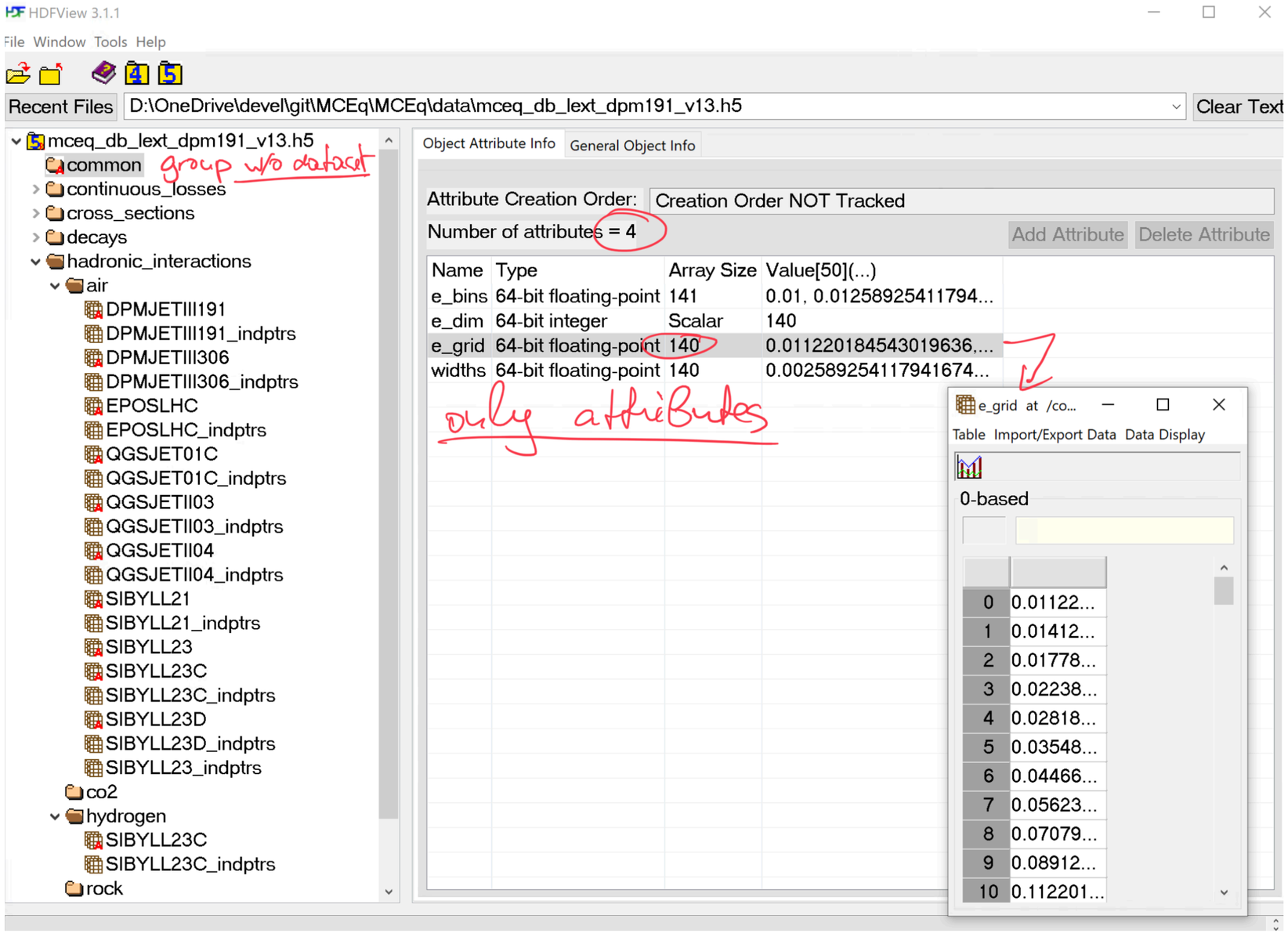
Example MCEq database layout

- Used official tool to view file “HDFView”
- MCEq stores all data in one file
- File is opened/closed many times
- Only data requested by program flow is read from disk



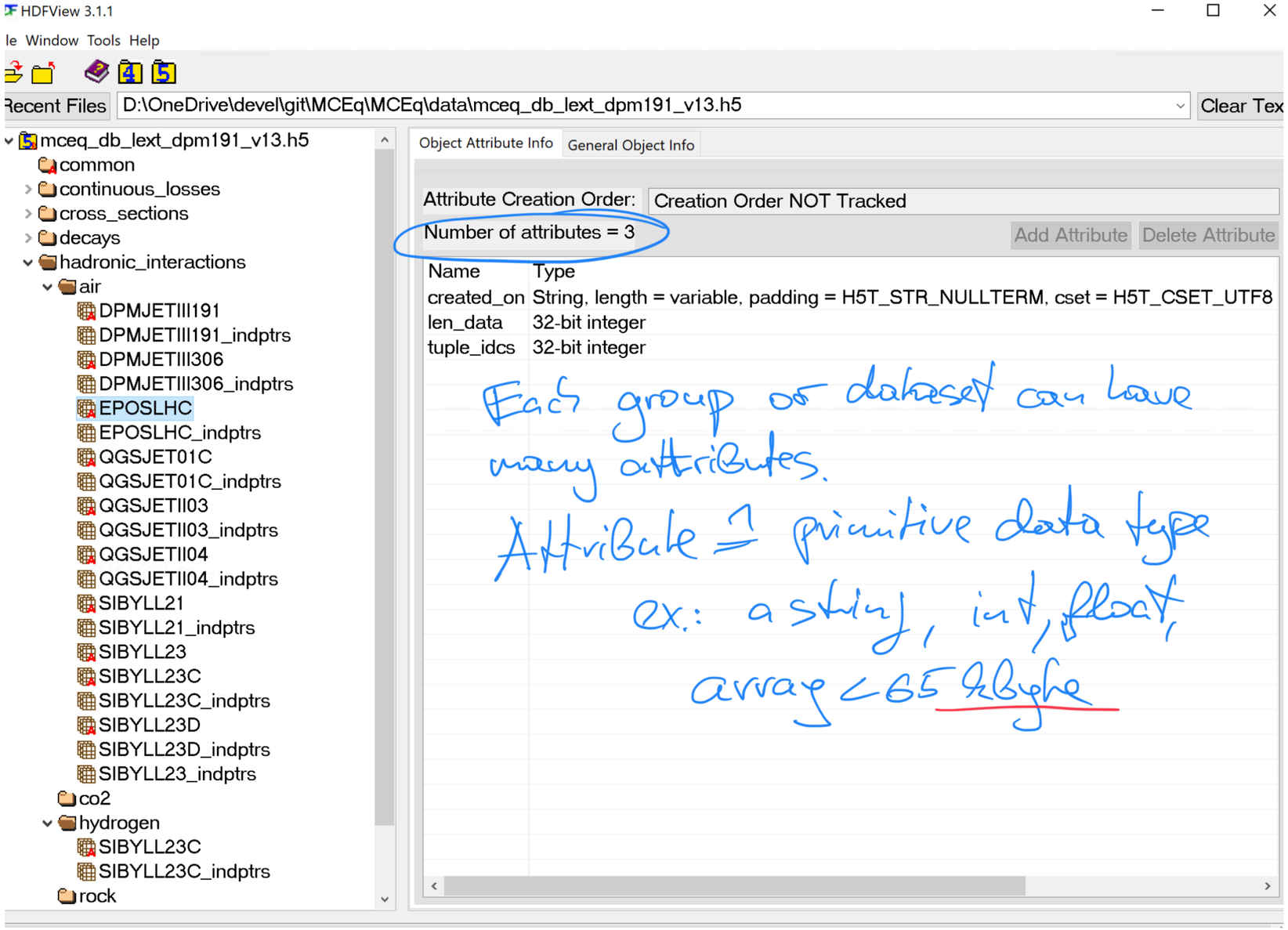
HDF group

- Element to organize hierarchy
- Contains groups and datasets
- Can have many “properties”



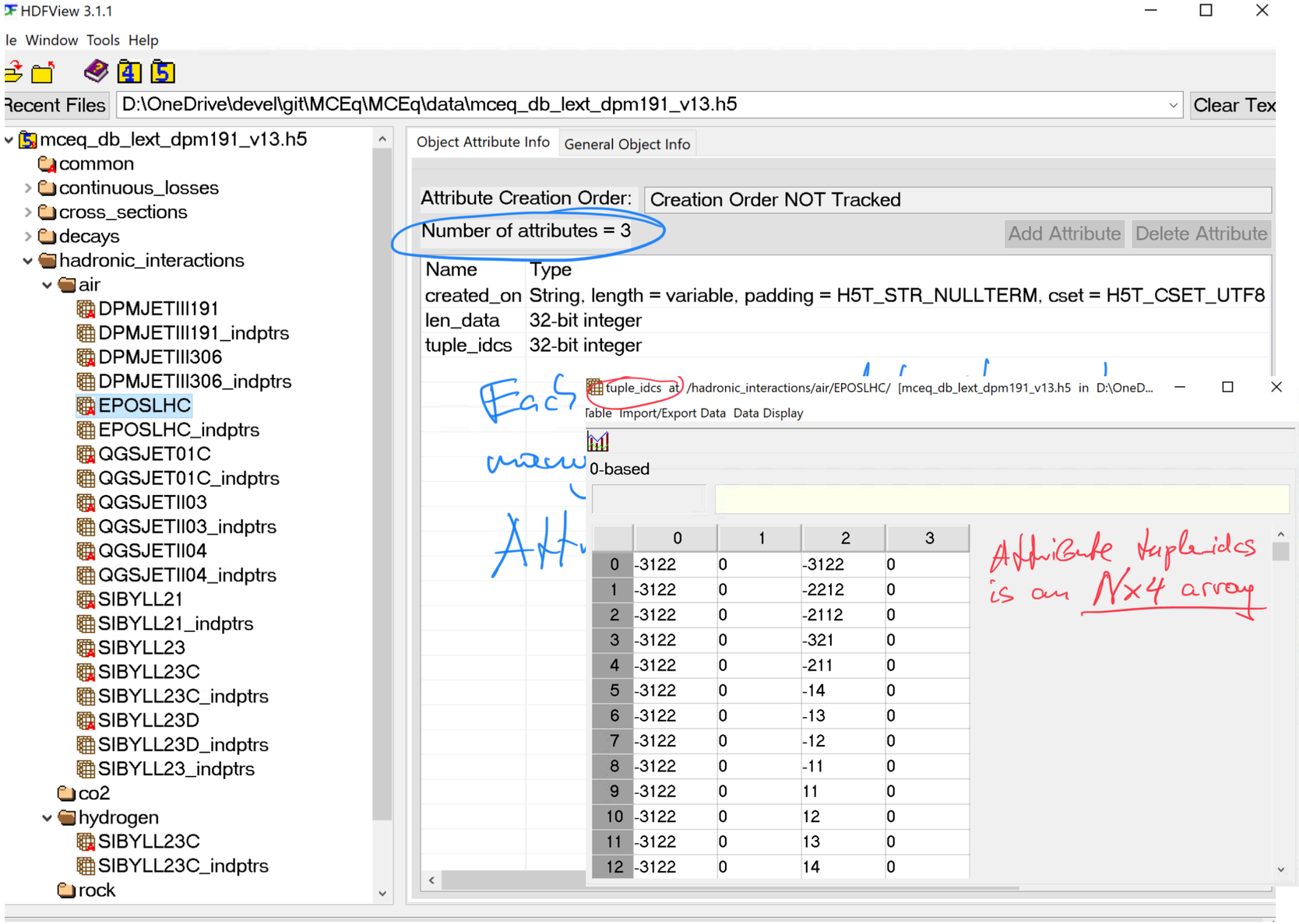
HDF dataset

- Is like a “datatype”
- Can be a primitive array of floats, into, etc
- Can have many “properties”



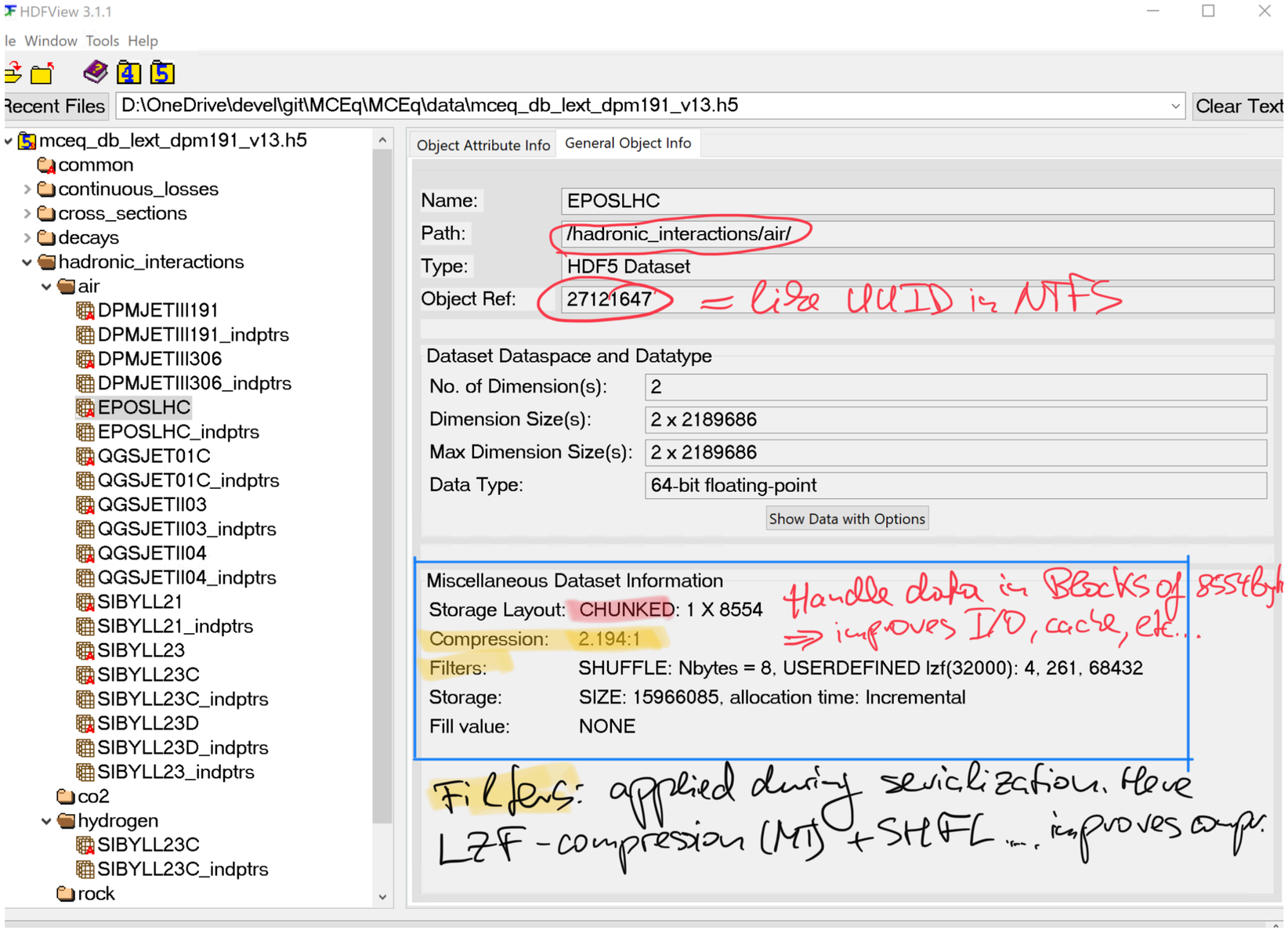
HDF dataset

- Is like a “datatype”
- Can be a primitive array of floats, into, etc
- Can have many “properties”



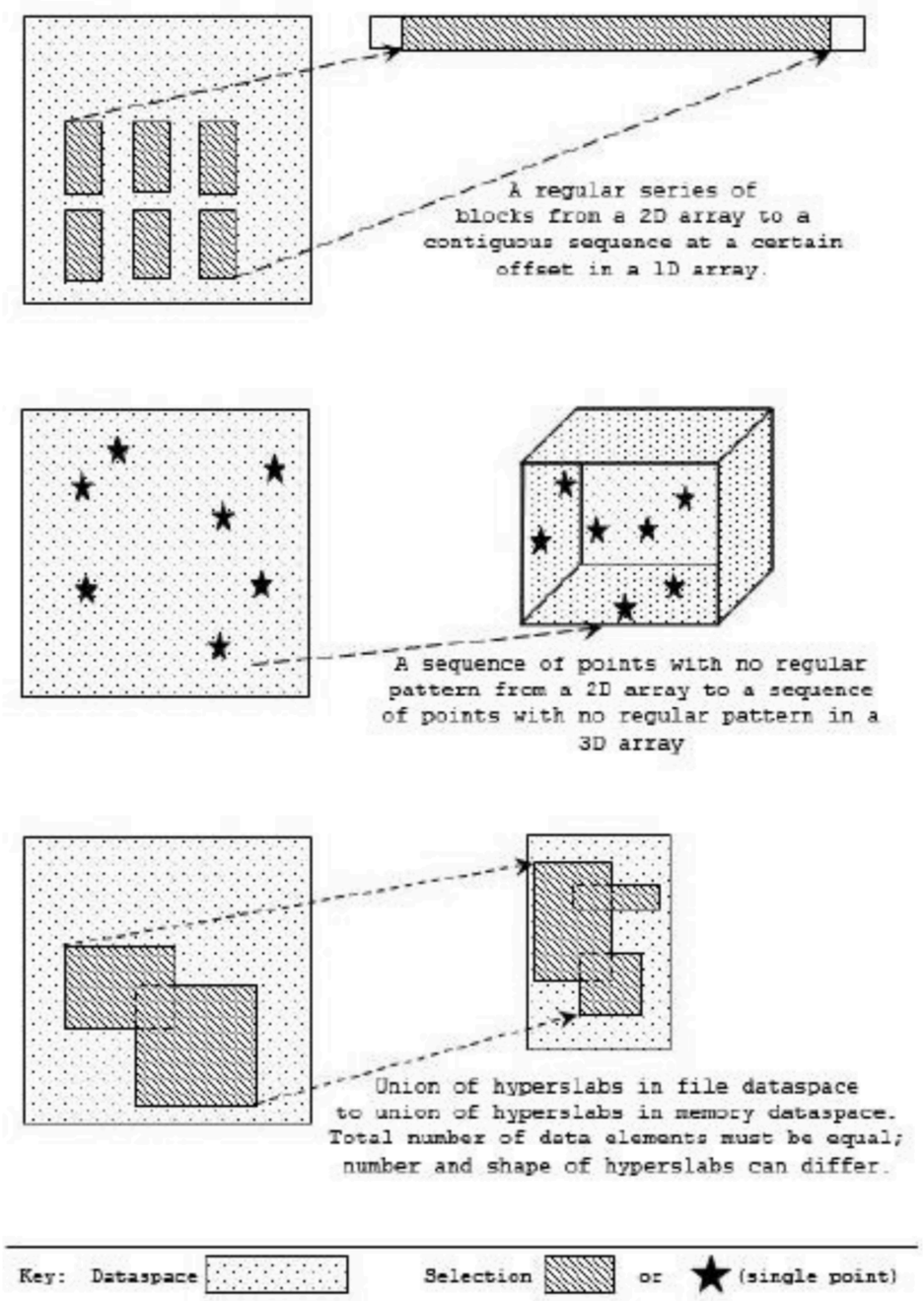
HDF dataset

- Is like a “datatype”
- Can be a primitive array of floats, into, etc
- Can have many “properties”
- Customization options (Filters) for read/write



Reading and writing to parts of a dataset

- The HDF library can read parts of dataset defined by:
  - Ranges
  - Sequence of points
  - Block selections
  - Periodic selections, etc...
- One can think of a method to read particles only at the coordinates of detectors, for example...



Multi-file and nested structures

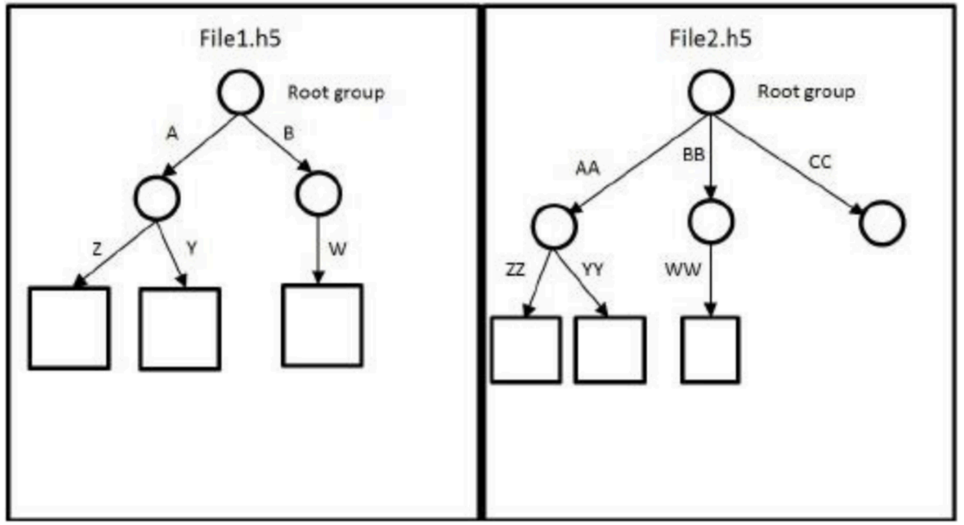


Figure 3-3. Two separate files

The figure below shows the two files after File2 has been mounted File1 at the group that is the target of the B link.

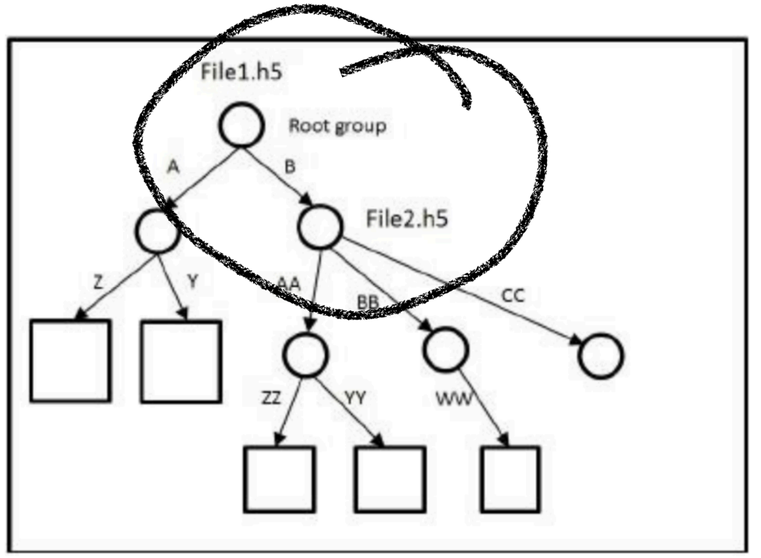
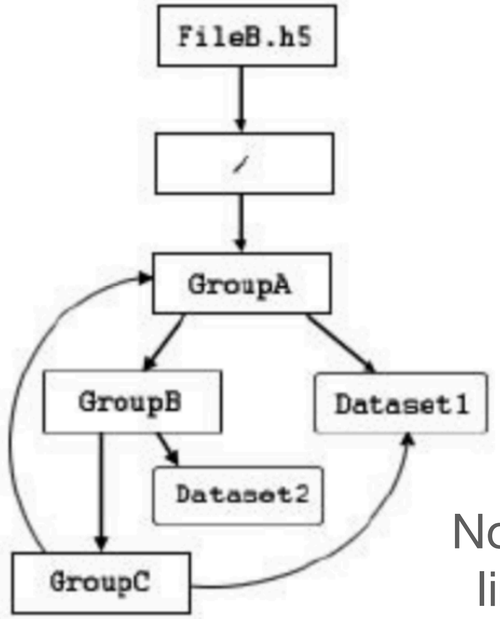


Figure 3-4. File2 mounted on File1



No obvious limitation

Figure 4-2. A file with a circular reference

Low-level file drivers

- Splitting I/O into multiple files can be handled by the low-level driver
- Split (subset of Multi) can write metadata and data into separate files
- MPI driver for control and comms
- Fine-grained control over interaction with file system

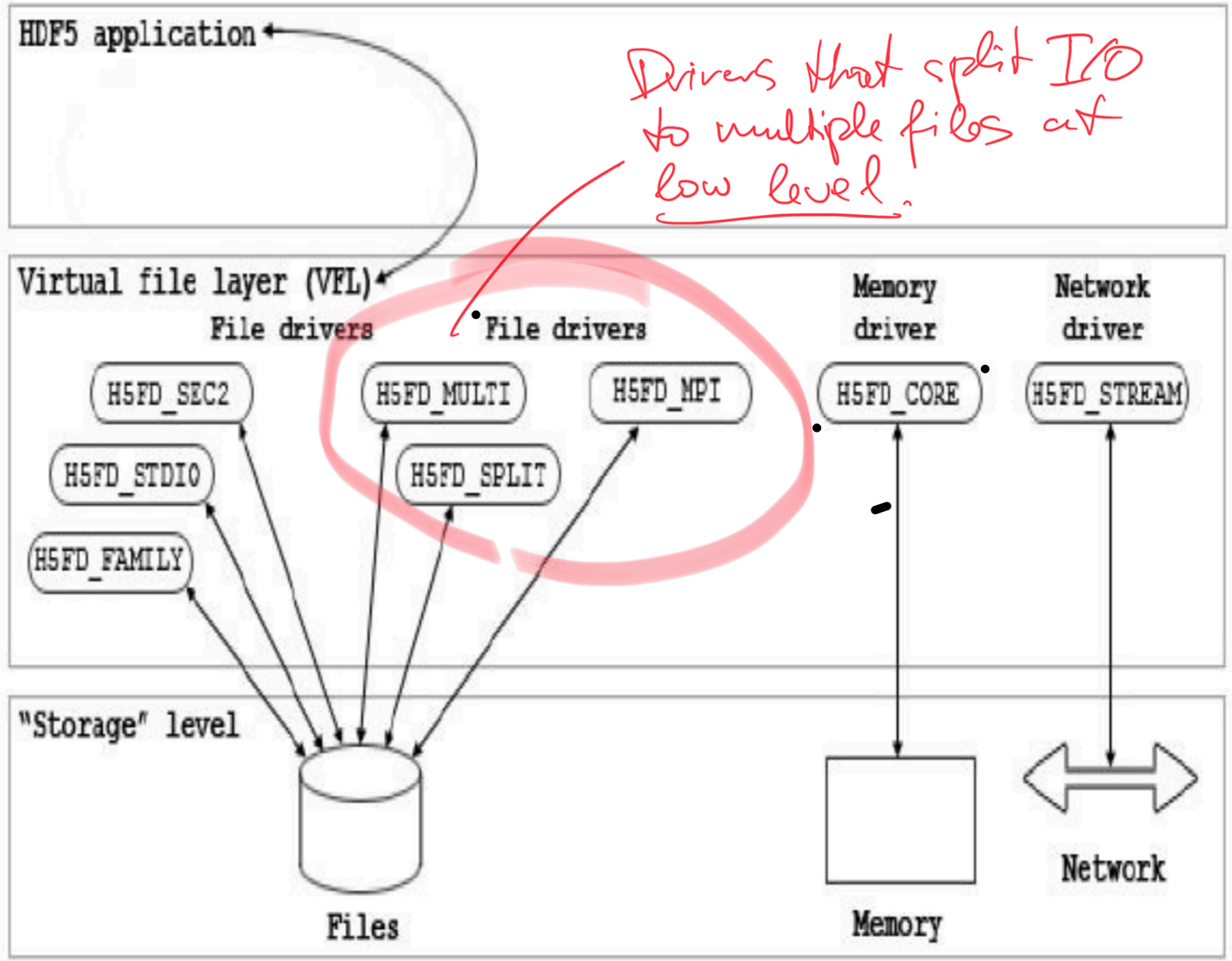


Figure 3-2. I/O path from application to VFL and low-level drivers to storage

Very easy to use in user applications

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np

import h5py
```

In [2]:

```
hdf_file = h5py.File("../../MCEq/MCEq/data/mceq_db_lext_dpm191_v13.h5", "r")
```

In [3]:

```
print(hdf_file['cross_sections']['air']['EPOSLHC'])
```

```
<HDF5 dataset "EPOSLHC": shape (140, 3), type "<f8">
```

In [4]:

```
assert hdf_file['cross_sections/air/EPOSLHC'] == hdf_file['cross_sections']['air']['EPOSLHC']
```

In [5]:

```
#Access attributes of group or dataset, another dictionary
hdf_file['common'].attrs.keys()
```

Out[5]:

```
<KeysViewHDF5 ['e_bins', 'e_dim', 'e_grid', 'widths']>
```

In [29]:

```
e_grid = hdf_file['common'].attrs['e_grid']
cs = hdf_file['cross_sections/air/DPMJETIII191']
```

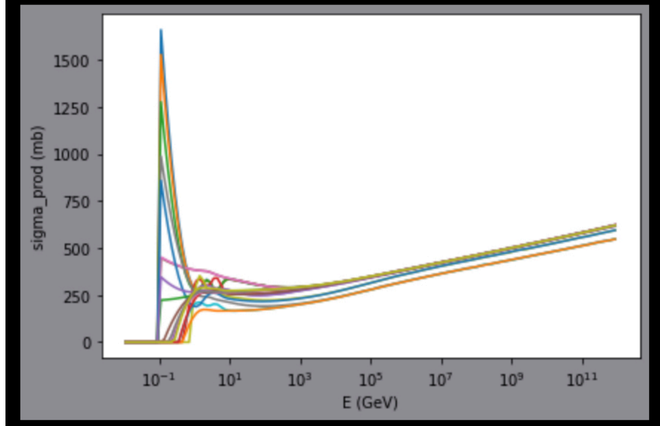
Read attribute is same as reading dataset

In [36]:

```
for ip, pidx in enumerate(hdf_file['cross_sections/air/DPMJETIII191'].attrs['projectiles']):
    plt.semilogx(e_grid, cs[:,ip], label=str(pidx))
plt.xlabel('E (GeV)')
plt.ylabel('sigma_prod (mb)')
plt.tight_layout()
```

projectiles is an iterable list.

immediate access as numpy array



Summary

- HDF5 is a widely adopted, high-performance, open-source file format.
- The features to organize structured data and meta data should be well suited for CORSIKA8.
- In my experience it is smooth to interact with. Speed is good and optimization is possible, but quite technical job.
- Binaries can be bundled with CORSIKA.
- It is very easy to read and write with standard options. In my experience multi-TB single files were handled without issues.
- No experience with MPI, multi-file, etc. —> no explicit recommendation