HDTREE

Schema Evolution and the TTree in HDF5

Tom Eichlersmith

he/him/his

University of Minnesota

eichl008@umn.edu

CHEP 2023

May 8, 2023

Outline

Helpful Aspects of the TTree

Interface to Data

Data viewed hierarchically in-memory while still being well organized on-disk.

Schema-Evolution

User-defined classes can evolve with version numbers.

"Ragged" Data

Directly represent HEP's common data "awkwardness" where variables change shape from event-to-event

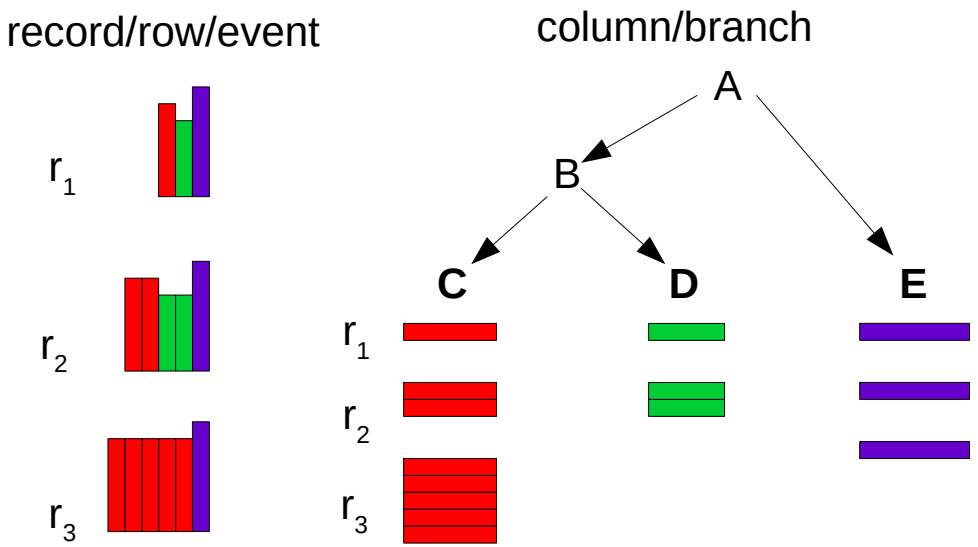
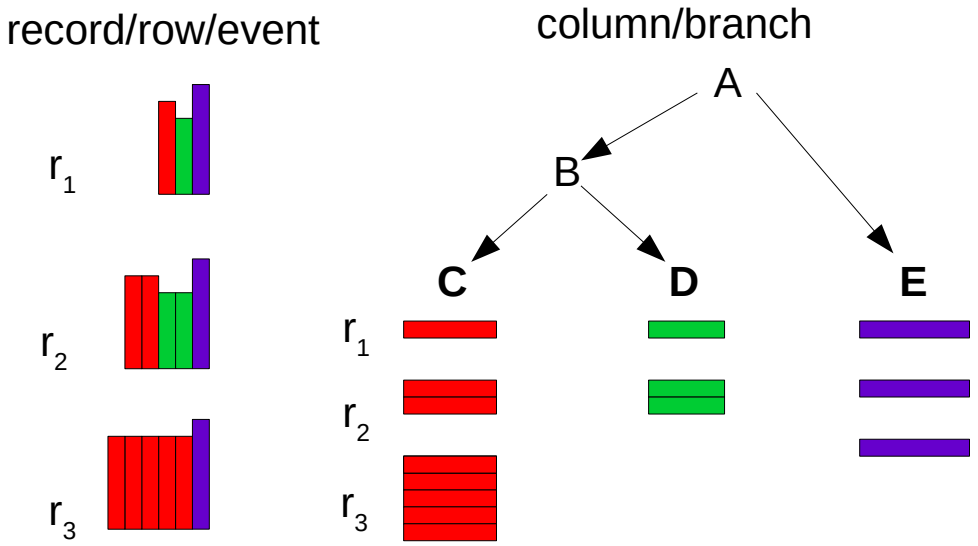


Figure: Credit to J. Pivarski from talk at 2019 CHEP for diagram idea. ▸ Pivarski 2019 CHEP .

Why re-implement?

1. Wish to separate the *format* from the *API implementation*
2. Wish to implement *native* APIs in other languages
3. Take lessons learned from using and developing with ROOT – intentionally avoid supporting TTree API to focus on a more modern API



HDF5

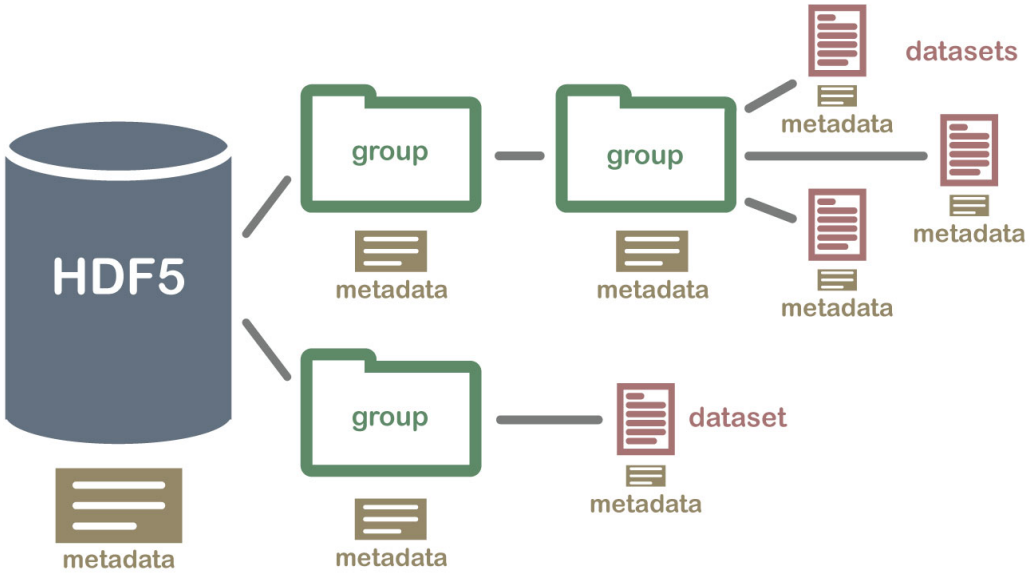


Figure: Diagram of how users can organize data within an HDF5 file. Leah A. Wasser "Hierarchical Data Formats - What is HDF5?" *Neon Science*, Oct 2022. online .

Performance

Chunking and compression available to user configuration.

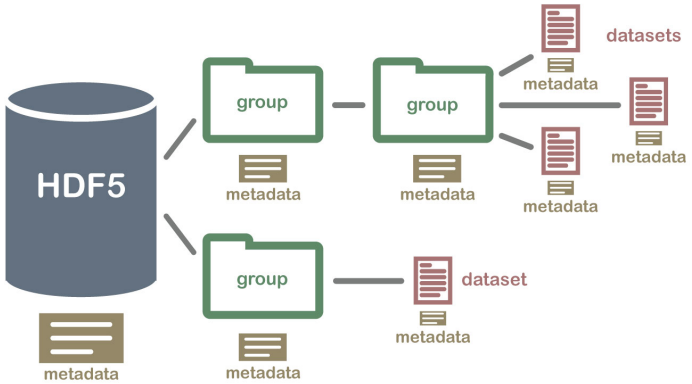
Usability

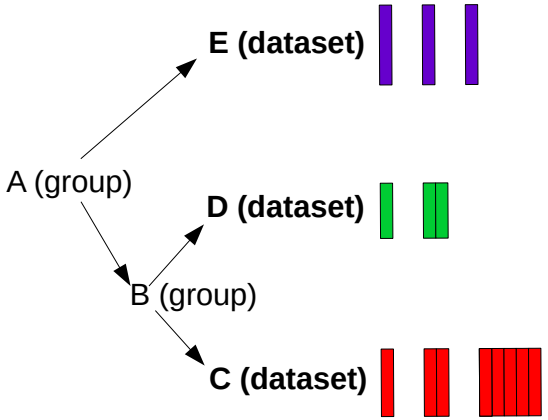
C, C++, and Fortran Official APIs - Python, Rust, Julia, and more provided by community

Support

Industry-supported, popular file format already used within HEP and other research disciplines

HDTree Meta-Format





1. Atomic data written into HDF5 datasets
2. Description of that data written into HDF5 attributes
3. Datasets organized into groups based on hierarchical in-memory ownership

Specification of HDTree Meta-Format documented online ▸ tomeichlersmith.github.io/hdtree

HDTree C++ API

First API implementation for this meta-format based on C++17 and HighFive[1]

Capabilities

- Read and write the HDTree meta-format
- Default compression, chunking, and caching behavior similar to ROOT's handling of TTrees
- Schema evolution of user-defined classes

Distinctions from ROOT

- **No** separate dictionary file required – "binding" user classes to organize data is completely contained in class definition
- **No** user-juggling of memory addresses
- Follows RAII principles – i.e. **No** need to Close files or explicitly Write trees

[1] a header-only, template-focused C++ API for HDF5 – GitHub: ▸ BlueBrain/HighFive

Usability

of the HDTree C++ API

```cpp
auto tree = hdtree::Tree::save(
  "my-file.hdf5","/path/to/tree");
auto& i_entry = tree.branch<int>(
  "i_entry");
auto& rand_data = tree.branch<
  std::vector<float>
  >("rand_data");
for (std::size_t i{0}; i < 5; i++)
{
  *i_entry = i;
  rand_data->resize(i);
  for (float& pt : *rand_data) {
    pt = i*i;
  }
  tree.save();
}
```

Code Task: Write a tree with two branches: entry index and an array whose size is the index and content is the index squared.

- Create `tree` and new `branch`es, accessing branches through handles
- Handles act like smart pointers to underlying data to avoid unnecessary copying
- Final writing and closing done when `tree` is destructed

User Classes

Force documentation of what on-disk data is

```cpp
class MyData {
  float x_;
  friend class hdtree::access;
  template <typename Branch>
  void attach(Branch& b) {
    b.attach("x", x_);
  }
 public:
  hdtree_class_version(2);
  MyData() = default;
  MyData(float x)
    : x_{x} {}
  void clear() {
    x_ = 0.;
  }
};
```

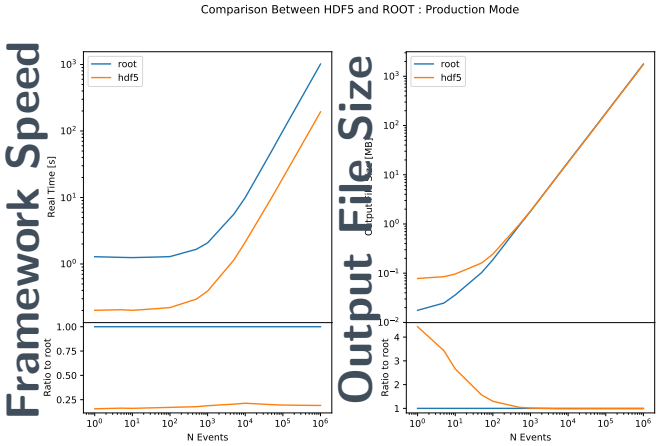
- Templated `attach` method not necessary but does reduce code boilerplate
- Optional definition of class version – library reports version discrepancies
- `clear` method for resetting class to an "unset" state

Interaction with `Tree` (and underlying `Branch`) the same.

```cpp
auto& my_data = tree.branch<std::
    vector<MyData>>("my_data");
```

Generally – on par with ROOT

- Compared writing and reading of variable-length `std::vector<float>` for a range of entries
- Both close and the faster one depends on computer I run on



Specifics of test highly matters

Embedding this style of serialization into a more generalized event-processing framework led to a vast difference in speed.

HDF5-based Serialization ▸ LDMX-Software/fire

ROOT-based Serialization ▸ LDMX-Software/Framework

Summary

HDTree

- Utilize TTree-like data organization and access patterns while gaining the benefit of an industry-standard file format.
- C++ API currently available with performance on-par (if not exceeding) ROOT.

- ✓ Implementing specialized APIs for HDF5 files is easier
- ✓ HDF5 files are supported by other data science libraries (e.g. pandas and pytorch)
- ✓ Develop data processing frameworks with less fear of memory issues

Moving Forward

- More thorough performance testing
- Start Python API based on well-supported and more general h5py package