IRIS-HEP Fellow Project Proposal: Reading & RNTuples in Uproot

The foundation of any analysis software and framework lies in reading and parsing `.root` files. Right now in 2022, the vast majority of physics data is stored in the `TTree` (or collection of) objects inside `.root` files.[1]

The `TTree` design is mostly columnar, but with each column broken into `basket`s such that appending or modifying entries does not imply complete re-written of the on on-disk bits. This is critically important as the Tier-0 of CERN's storage facilities uses tapes, which has super low read/write speed and no random read/write at all.

The next evolution of data serialization format [2] takes the name `RNTuple`, which draws inspiration from some industry-standard formats such as Apache Arrow/Parquet. According to the plan as of 2021 [3], it will be ready for production use by 2024, and analysis users can expect to run into samples around or a few years afterward.

As the spec is fixed and ROOT/C++ implementation of `RNTuple` is maturing, it is time to implement the reading and possibly writing of `RNTuple` object in foundational packages for alternative analysis workflow – `uproot`, to deliver a painless transition for physics users and allowing a buffer in time for fixing performance and or usage problems before large scale adoption of `RNTuple` itself.

We know from user feedback that it's important `uproot` is able to provide complete support for `TTree` reading and writing without dependency on ROOT/C++ stack. This is because the development environment for a physics analysis user is hard to debug at best, and users value the independence provided by `uproot` and the analysis systems stemming from it. In fact, we'd argue that without timely support of RNTuple, `uproot` and friends would fall out of usage quickly once the migration to RNTuple happens.

Besides the importance of keeping `uproot` and the entire analysis ecosystem in Python usable in the coming years, implementing RNTuple reading and writing also provides an opportunity to test the specification itself. Unlike previous endeavors such as TTree, RNTuple had a clear top-down design and a structured specification at its core. Learning from examples in industry, Arrow has multiple independent implementations [4], which helped verify the correctness of the specification itself. We believe everyone would benefit from an independent verification.

Estimated Timeline:

Week 1 & 2: Understanding RNTuple implementation and building scaffolding/debug utilities
Week 3 & 4: Make a minimal collection of functions interactively parse/read a few RNTuple samples
Week 5 & 6: Organize the functions into a new class and interface design
Week 7: Source more complex RNTuple test files, debug edge cases with complex serialized objects

+ Housekeeping: clean up code, easy optimizations, enrich docs and examples

Week 8 & 9: Investigate writing of RNTuple sample and try to keep TTree convention/api familiarity
Week 10 & 11: Implement writing RNTuple, retrospectively harmonize reading and writing when possible
Week 12: Clean up code and docs of writing.

[1]: There are “raw” physics data not readable by vanilla ROOT such as xAOD and EDM (Event Data Model), but those are rarely used directly by physics analysis users also due to the sheer size and lack of high-level variables.
[2]: https://arxiv.org/abs/2003.07669
[3]: https://indico.jlab.org/event/420/contributions/7701/
[4]: https://github.com/apache/arrow