Columnar file I/O with hepconvert and Uproot

Zoë Bilodeau[1], Jim Pivarski[1]

[1]Princeton University

Support for this work was provided by NSF cooperative agreements OAC-1836650 and PHY-2323298 (IRIS-HEP).

PRINCETON UNIVERSITY

Main Projects: hepconvert and Uproot feature

- hepconvert:
  - Columnar conversion package in Python
  - Worked with Uproot and ROOT files; served as a precursor to more in-depth work with Uproot
- Uproot:
  - Worked on adding a new feature: Adding new TBranches to an existing TTree

hepconvert: Time Spent on Columnar File Conversions

- Unnecessary time and energy from physicists to convert between file formats
- Even basic conversions require multiple lines of code, multiple file I/O packages
  - There are a number of common modifications that take extra time
- Many users are writing very similar code

What is hepconvert?

- High-level Python converter between **ROOT**, **Parquet**, (and eventually) and **HDF5**

- Uses common I/O packages
  - Uproot
  - Awkward
  - h5py
  - Dask-awkward

Goal: Quick, Simple File Conversions

- Main goal of hepconvert is **convenience**
- Blocks of code -> single function call
  - One package
  - Memory management and compression handled
  - Parameters for customization
- User input oriented

Overview of Features:

- Features added at **user request**
  - Converters between Parquet and ROOT
  - ROOT to ROOT
  - Common file manipulations
    - Add/remove data
    - Hadd-like functionality
    - Change compression
  - Address common issues
  - Command Line Interface

Memory Management: Batches

- For large files, it is necessary to read and write data in batches
- Can take time depending on file structure and I/O package;
  - Each “batch” is a different structure
  - Always require multiple lines of code/loops

TTree (ROOT)		
Entries	Branch 1	Branch 2
1		
2		
3		
4		
5		
6		

Parquet File		
Row-groups	Column 1	Column 2
1		
2		

Work with ROOT files:

- Pure Python; users don’t need ROOT
- Writing capabilities of Uproot
  - Currently works with **flat TTrees, NanoAOD-like** files
  - One level deep

Note

The small but growing list of data types can be written as TTrees is:

- dict of NumPy arrays (flat, multidimensional, and/or structured), Awkward Arrays containing one level of variable-length lists and/or one level of records, or a Pandas DataFrame with a numeric index
- a single NumPy structured array (one level deep)
- a single Awkward Array containing one level of variable-length lists and/or one level of records
- a single Pandas DataFrame with a numeric index

Parquet to ROOT

- One Parquet file -> one TTree
  - Now have merge_parquet; could merge data from multiple Parquet files to one TTree
- Writing capabilities of Awkward Array
  - Compression settings and many other options available
  - ak.to_parquet()

Parquet file to ROOT file:

```
>>> hepconvert.root_to_parquet("out_file.parquet", "in_file.root")
```

ROOT to Parquet

- One TTree -> one Parquet File
- Can merge TTrees, or specify one TTree to be written
- Step-size becomes row-group size
- Options:
  - Branch skimming, branch slimming

ROOT file to Parquet file:

```
>>> hepconvert.root_to_parquet("out_file.parquet", "in_file.root")
```

Awkward Feature: Iterative Writing to Parquet Files

- Re-implemented **ak.to_parquet_row_groups()**
- Writes data to parquet files in batches (row-groups)
- Pass data as an iterable over data rather than array

```
ak.to_parquet_row_groups(
    (i for i in f[tree].iterate(step_size=step_size,)),
    out_file,
)
```

Copy (and modify) ROOT Files

```
>>> hepconvert.copy_root("out_file.parquet", "in_file.root")
```

- Features for altering files
  - Automatically groups branches to avoid duplicate counter branches when writing with Uproot
    - Instead of manually choosing and grouping branches with ak.zip()
  - Branch-skimming, TTree removal, Branch removal
    - Wildcarding supported
  - Can either write to a new file or return a writable uproot object in memory to then work with
  - Change compression type
  - Run from command-line

Merging TTrees and Histogram Summing (hadd-like)

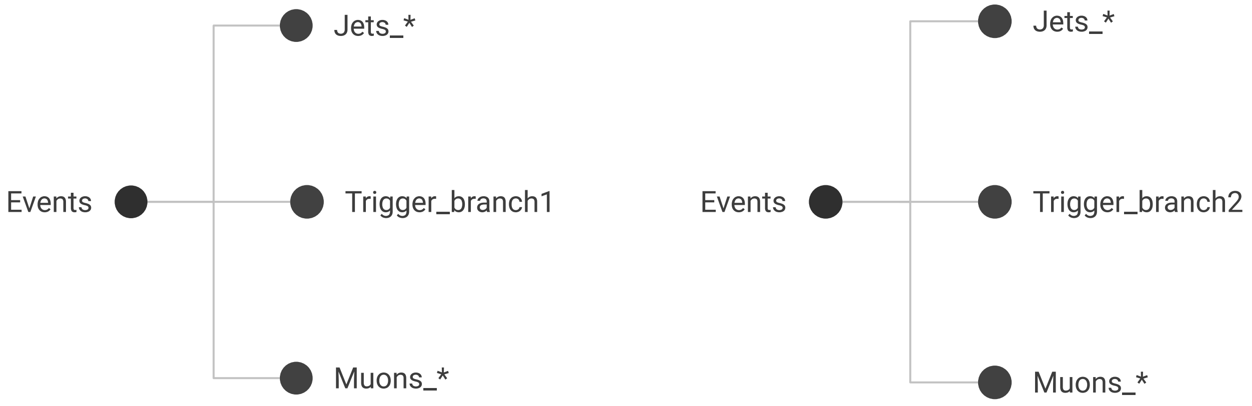
- `add_histograms()`:
  - Sums contents of histograms in many files
  - Writes to a new file
- `merge_root()`:
  - Merges like TTrees, sums histograms from many files
  - Branch skimming, branch slimming, cuts, etc.
  - Customizable parameters similar to hadd
    - union, append, same_names
- Not dependent on ROOT!

Uproot Feature: Add Branches to an Existing TTree

- **Goal: Add one or more branches to an existing TTree**

```
>>> uproot.add_branches('tree', {branch1: data, branch2: data})
```

- **Example of use:**
  - Addresses common issue with CMS data
    - Users wanted to merge NanoAOD files with mismatched branches
    - Can backfill with booleans

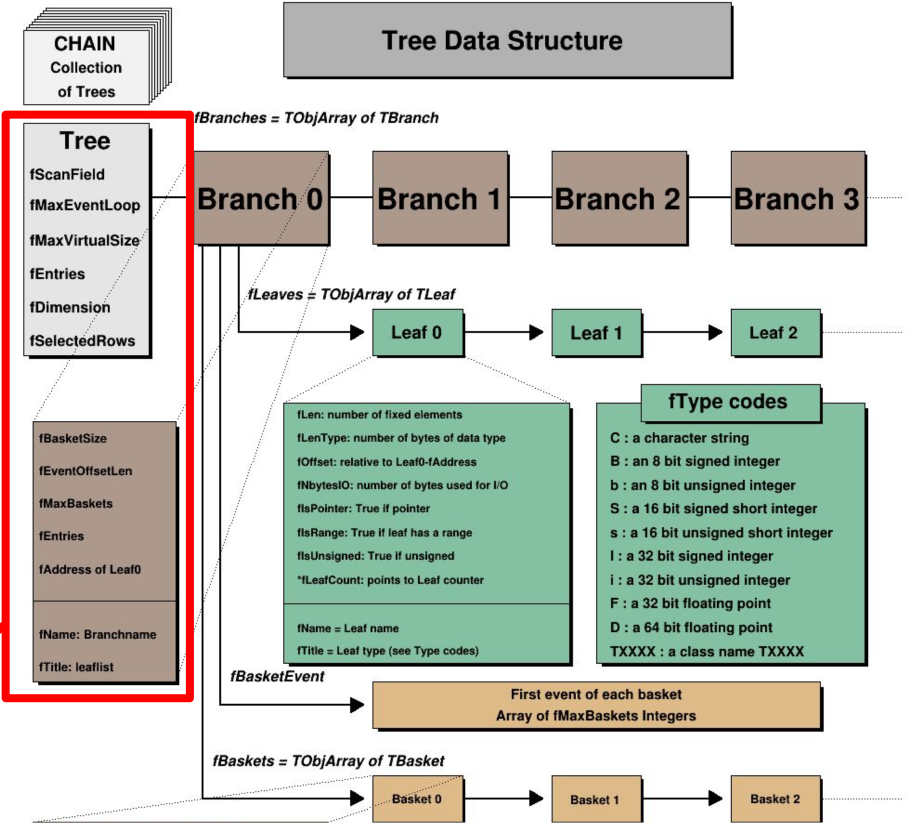


Copying TTree and Old Branches

Challenge: Addressing Robustness

- Rewrites TTree metadata
  - Can only handle most recent ROOT versions (generally after 2017)
- Copies branches from original TTree; copying process does not depend on branch type/content

Rewritten (new TTree object created)



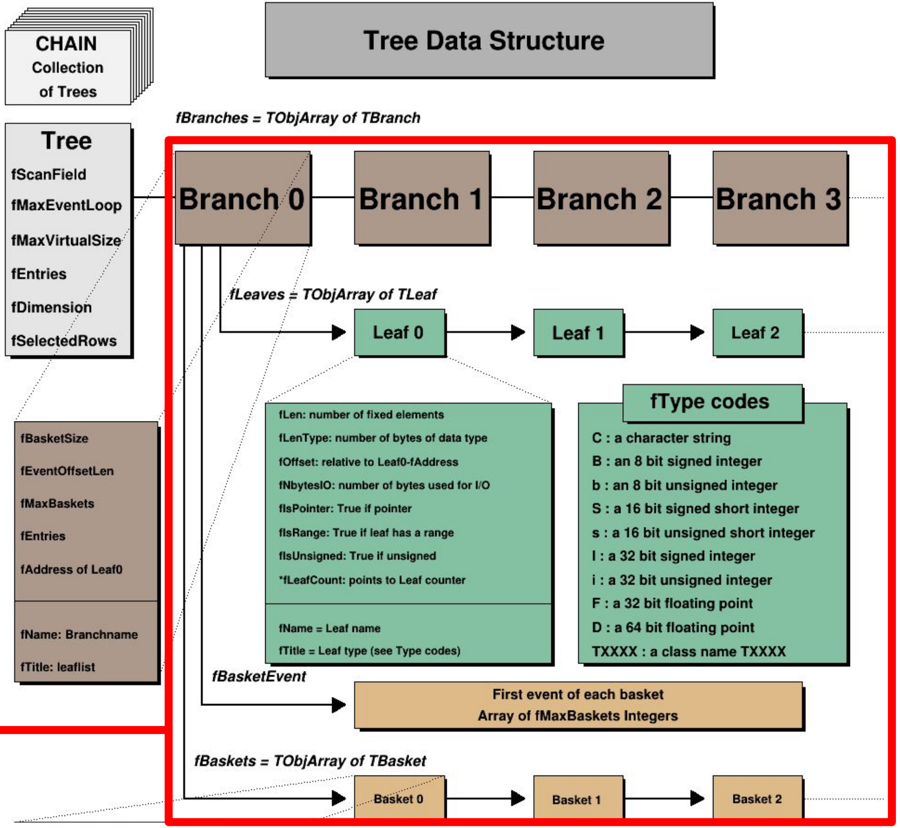
https://root.cern.ch/root/htmldoc/guides/users-guide/Trees.html

Copying TTree and Old Branches

Challenge: Addressing Robustness

- Rewrites TTree metadata
  - Can only handle most recent ROOT versions (generally after 2017)
- Copies branches from original TTree; copying process does not depend on branch type/content

Bytes are copied, reference numbers updated



https://root.cern.ch/root/htmldoc/guides/users-guide/Trees.html

Copying TTree and Old Branches

Data rewritten or updated:

- Information describing TTree and data
  - New TTree object with updated metadata
- Reference numbers
  - For objects referenced in multiple places

Data directly copied:

- Entirety of TBranches, including TLeaves and TBaskets

Copying was done using Uproot's reading ability; it recognizes objects and can find and skip over portions (i.e. a TLeaf within a TBranch)

Copying TTree and Old Branches

Challenge: Changing TTree metadata size

- Files made with ROOT can have smaller TTree metadata; when copying this should always be changed to the larger size
- This can shift all TRefs as they depend on object’s position in chunks; problem to update
- Difficult to diagnose

Adding New Branches

- They are appended to the end of the TTree, should not affect previous data
- Can only add version 13 TBranches, reasonable limitation of Uproot
- Serialized very similarly to when a new TTree is written
- Can add multiple at once