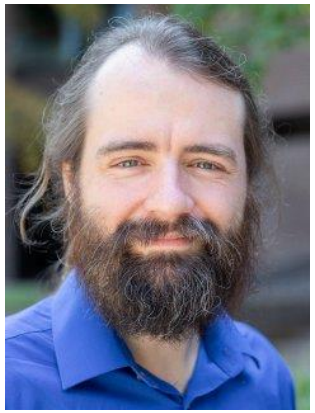
Writing files with uproot

Pratyush Das
Institute of Engineering
and Management, Kolkata

Jim Pivarski
Princeton University

History of uproot



Project started by Jim Pivarski in 2017 as a temporary replacement for Bulk I/O.

uproot 1 could read TTrees with single leaf branches and fully split C++ objects stored in a TTree.

1.0.0

jpivarski released this on Sep 14, 2017 · 1899 commits to master since this release

This is a usable version of the package. It has:

- branch reading into arrays with all three types of compression (zlib, lzma, lz4);
- full control over dtype conversion during the copy from the file into arrays;
- variable-length string reading (`TLeafC`);
- memory-mapped files, regular file reading, and XRootD;
- conveniences for querying directory objects, subdirectories, branch names, branch-to-counter associations, etc;
- parallel basket reading and decompression, which accelerates lzma;

2.0.5

jpivarski released this on Nov 25, 2017 · 1495 commits to master since this release

uproot 2.x is a complete rewrite of the system.

- Unlike 1.x, this version first reads the ROOT file's streamer info and uses that to deserialize almost all classes into Python objects. Apart from corner cases involving the bootstrapping classes (TFile, TKey, TStreamer*, etc.), the deserialization code will never be out of date.
- It also replaces the spaghetti of special cases for numerical branches and string-valued branches with a generic "Interpretation" system, where different kinds of branches get custom classes (possibly generated from streamers) to turn the bytes on disk into different Python objects.
- Jagged arrays (array of arbitrary-length arrays) are on the same footing as Numpy arrays, which is a basis for reading any arbitrary-length content (e.g. using classes generated from streamers).
- All manipulations that must touch individual entries with Python code (e.g. making a jagged array of strings or `std::vector<...>`) is Numba-accelerated. If you have Numba installed, it will run as fast as Numpy calls (compiled code).
- File-reading mechanism replaced with separated "source" and "cursor" to emulate memory mapped access for all file types, including XRootD.
- Parallel executor and cache options simplified and made systematic across all method argument lists.
- Any dict-like object may be a cache; memory-based and disk-based dict subclasses with LRU eviction policies included.

All old tests work. Reference documentation written. Tutorials in progress.

By the end of 2017, uproot 2 was released following the release of uproot 1.6.3.

uproot 2 had a lot of the things users now associate with uproot like JaggedArrays.

This is where I come in…

My history with uproot

Hi!

I worked under the supervision of Jim Pivarski as a DIANA-HEP fellow in summer, 2018.

uproot could now write ROOT files with TObjStrings(for debugging) and histograms.

3.0.0

jpivarski released this on Sep 20, 2018 · 909 commits to master since this release

We can write `TObjString` and `TH1*` to ROOT files. This is the first non-beta release of uproot 3!

uproot 3.0 is released. Jim modularized uproot with array handling being split out into the awkward-array package and object methods into uproot-methods.

I came back as an IRIS-HEP fellow in summer, 2019 and added TTree writing functionality to uproot.

3.10.0

reikdas released this 23 days ago · 62 commits to master since this release

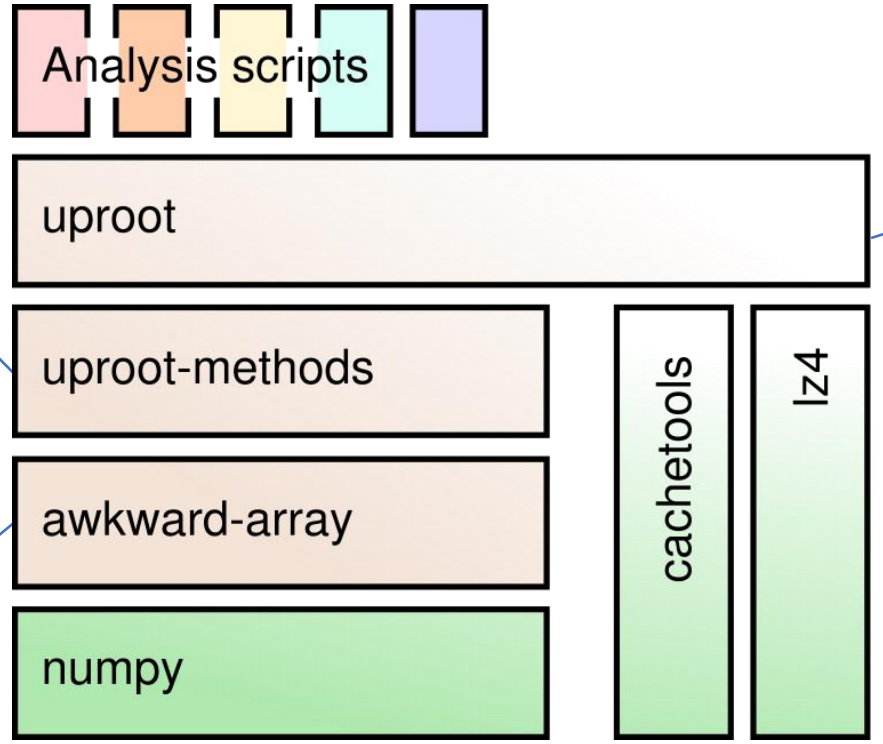
uproot can now write TTrees containing baskets with flat data!

uproot is *just* ROOT I/O

uproot is strictly concerned with ROOT I/O – all other functionality is handled by other libraries.

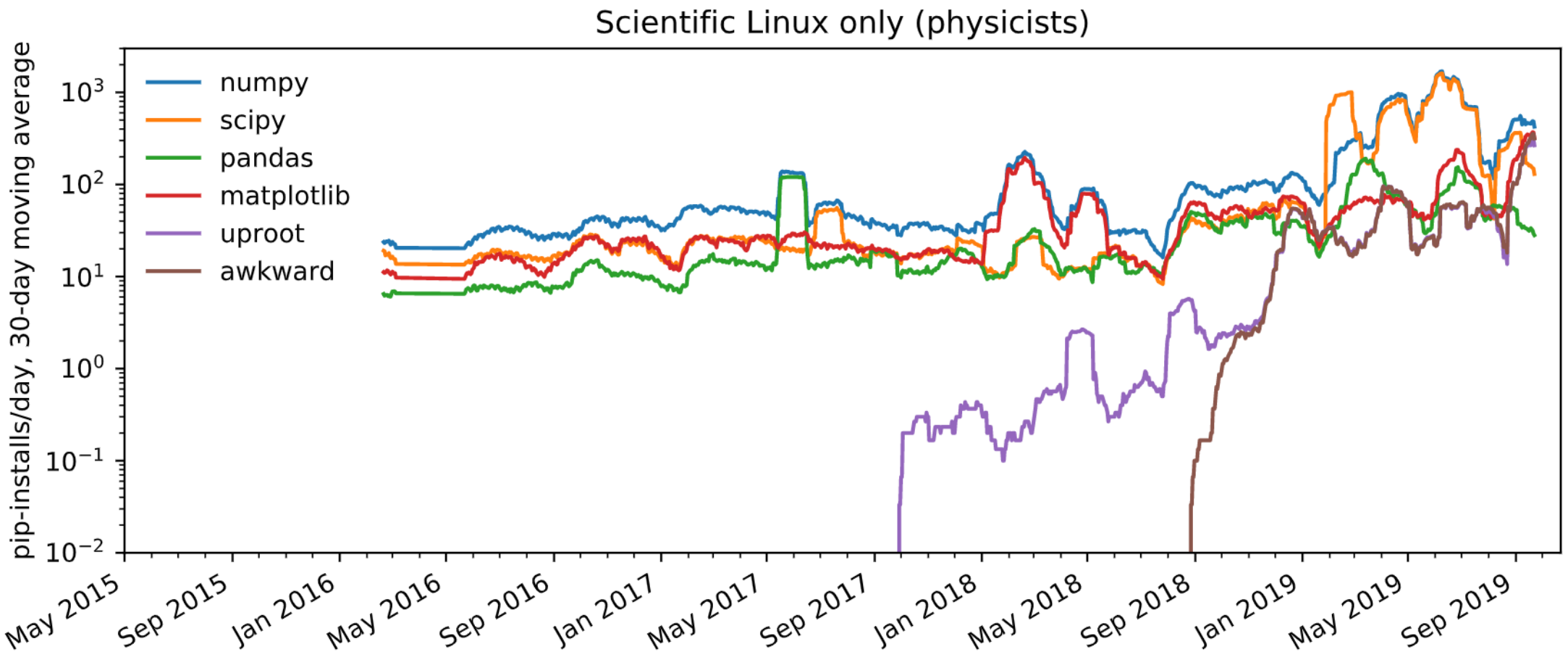
uproot-methods is home to physics methods read from ROOT files.

awkward-array for array manipulation beyond numpy with Jagged and Lazy Arrays.

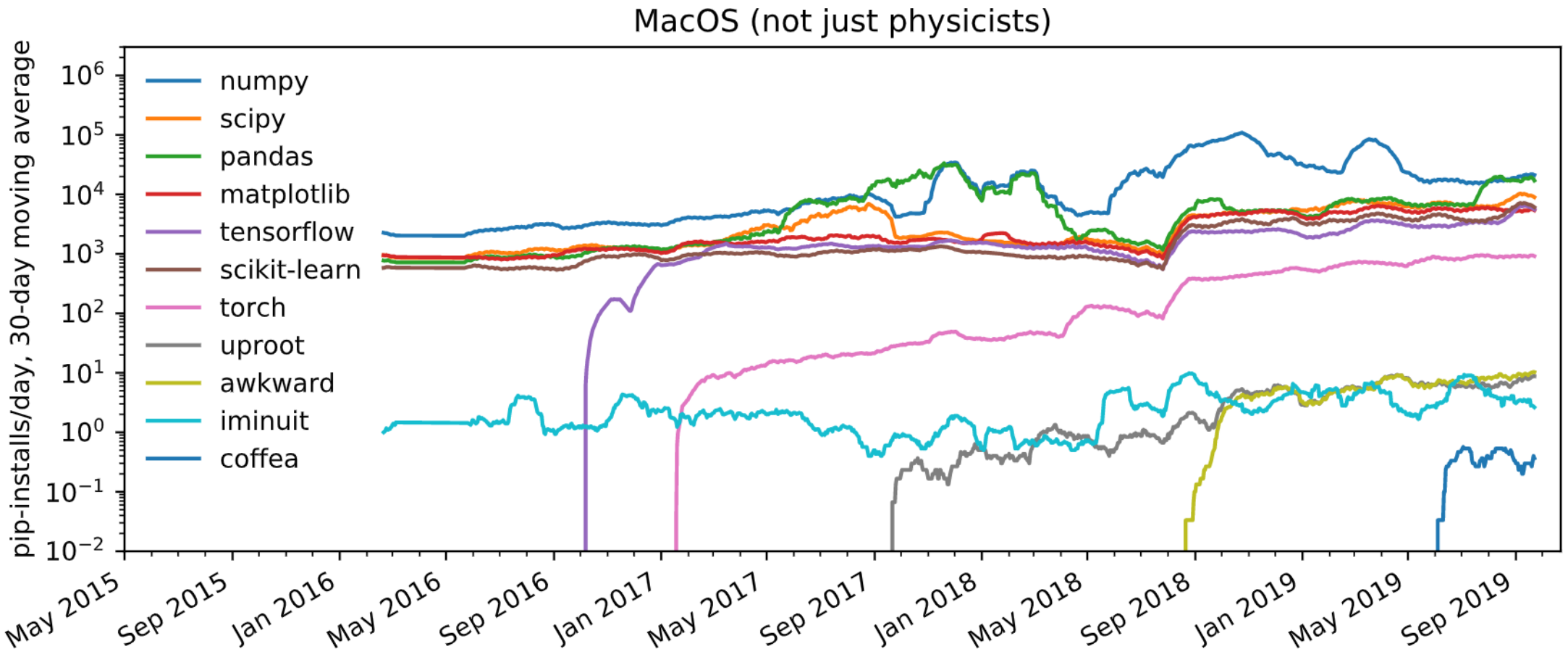


uproot is the layer most users interact with.

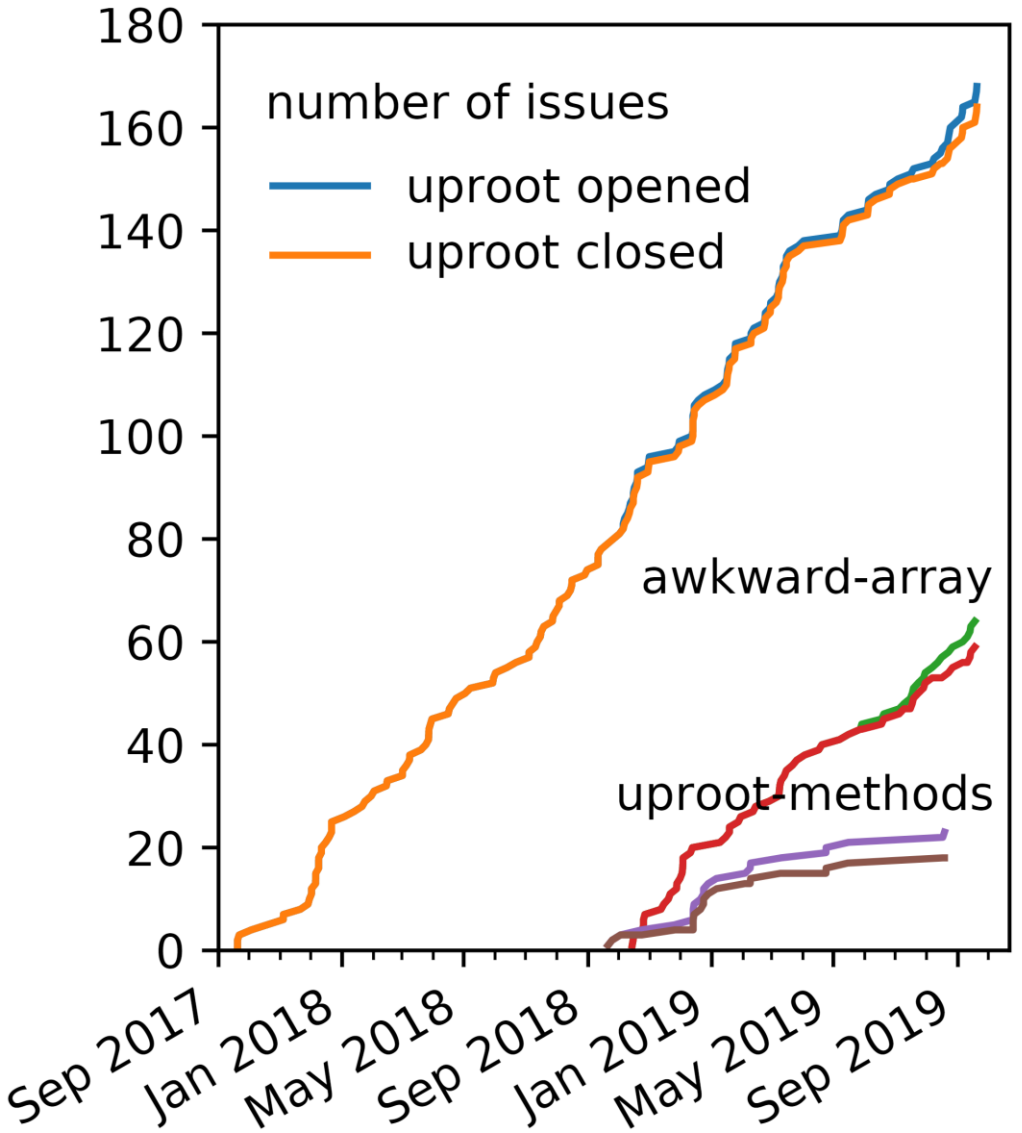
A lot of people are using uproot



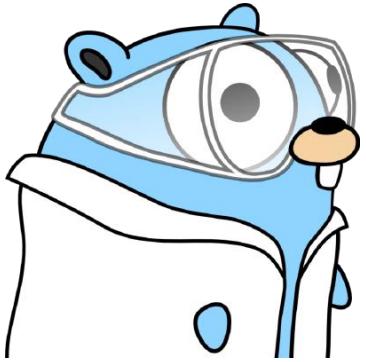
A lot of people are using uproot



A lot of people are using uproot



Go-HEP

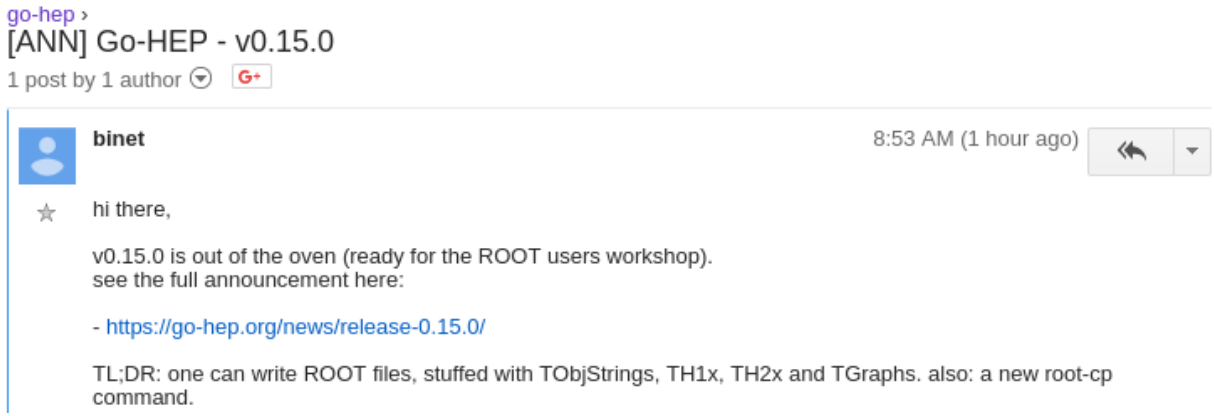


Similar to uproot, Go-HEP has a pure Go implementation of ROOT I/O (part of a larger ecosystem).

Author – Sebastien Binet

Both the reading and writing code of uproot are heavily inspired by Go-HEP.

uproot and Go-HEP announced file writing around the same time in 2018.



More recently, I looked at Go-HEP's implementation of ROOT's WriteObjectAny() to implement one in uproot for writing TTrees and Histogram bin labels.

uproot writing

uproot writing is aimed at the end-user physicist:

- uproot writing has a more limited scope than uproot reading – Object types useful for analysis like histograms, TTrees, TLorentzVectors and std::vectors. (Currently able to write strings, histograms and TTrees)

  Not complex types like AODs.

- Optimized for “write-once”, not “file as a database” or “partial write recovery”.

- Single threaded writing.

- Simple dict-like interfaces wherever possible.

Writing files with uproot

To write to a ROOT file in uproot, the file must be opened for writing using uproot.create, uproot.recreate or uproot.update.

Compression is set by uproot.ZLIB(n), uproot.LZMA(n), uproot.LZ4(n) or None (where n is the compression level).

```
file = uproot.recreate("tmp.root", compression=uproot.ZLIB(4))
```

file acts like a Python dict where the key is the TKey name and the value is the object being written.

```
file["name"] = "Some object, like a TObjString."
```

Writing histograms

Histograms(1D, 2D, 3D, TProfile, TProfile2D, TProfile3D) can be written in the same way.

uproot also recognizes numpy histograms, which may have come from other libraries.

```
file["from_numpy"] = numpy.histogram(numpy.random.normal(0, 1, 10000))
```

```
file["from_numpy"].show()

#                        0                                        2993.6
#                        +-------------------------------------------------+
# [-inf, -3.6179)     0  |                                                 |
# [-3.6179, -2.8738)  22 |                                                 |
# [-2.8738, -2.1296)  127|**                                               |
# [-2.1296, -1.3854)  632|***********                                      |
# [-1.3854, -0.64124) 1814|*********************************               |
# [-0.64124, 0.10294) 2851|*****************************************************|
# [0.10294, 0.84711)  2602|************************************************ |
# [0.84711, 1.5913)   1391|*************************                        |
# [1.5913, 2.3355)    464|********                                         |
# [2.3355, 3.0796)    85 |**                                               |
# [3.0796, 3.8238)    12 |                                                 |
# [3.8238, inf]       0  |                                                 |
#                        +-------------------------------------------------+
```

(More details in uproot README)

Writing TTrees – 2 phases

- Declaration phase

```
file = uproot.recreate("example.root")
file["tree"] = uproot.newtree({"branch1": int,
                               "branch2": numpy.int32,
                               "branch3": uproot.newbranch(numpy.float64, title="My title")})
```

- Filling phase

```
file["tree"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]),
                     "branch2": [11, 12, 13, 14, 15],
                     "branch3": numpy.array([21, 22, 23, 24, 25])})
```

Writing baskets - extend

The suggested interface of writing baskets to the TTree using the extend method.

```
: #Same as previous slide
file["tree"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]),
                     "branch2": [11, 12, 13, 14, 15],
                     "branch3": numpy.array([21, 22, 23, 24, 25])})
```

The extend method takes a dictionary where key is the name of the branch and the value of the dictior numpy array of a list of data to be written to the file.

REMEMBER TO ADD EQUAL NUMBER OF ENTRIES TO EACH BRANCH!

```
: file["tree"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]),
                     "branch2": [11, 12, 13],
                     "branch3": numpy.array([21, 22, 23, 24])})
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-5-6280911ee6cb> in <module>
      1 file["tree"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]),
      2                      "branch2": [11, 12, 13],
----> 3                      "branch3": numpy.array([21, 22, 23, 24])})

~/uproot/uproot/write/objects/TTree.py in extend(self, branchdict)
     81         first = next(values)
     82         if all(len(first) == len(value) for value in values) == False:
---> 83             raise Exception("Baskets of all branches should have the same length"
     84
     85         #Convert to numpy arrays of required dtype

Exception: Baskets of all branches should have the same length
```

Writing baskets – low level interface

If you want, you can write a basket to only one branch. But remember to add equal number of basket data to the other branches as well because ROOT assumes that all branches have equal number of basket data and will not read the non-uniform baskets.

```
f["tree"]["branch1"].newbasket([1, 2, 3])
```

Add 3 more basket data to branch2 and branch3!

```
f["tree"]["branch2"].newbasket([11, 12, 13])
f["tree"]["branch3"].newbasket([21, 22, 23])
```

The newbasket method does not(cannot) perform any checks on whether the number of basket data being added to the branches are equal in length or even if basket data is being added to all the branches.

Be careful when using this method!

TTree writing – compression

Baskets can be compressed using a similar interface to the one used when compressing the file.

Baskets are compressed according to the file compression, when no compression is specified.

‘compression’ parameter can be set at the

- TTree level

```
f["t"] = uproot.newtree(branchdict, compression=uproot.LZ4(4))
```

- TBranch level

```
branchdict = {"Branch": uproot.newbranch(">i4", compression=uproot.LZMA(3))}
```

Each branch can have its own compression algorithm.

Baskets are compressed according to the tree compression if no Branch compression is specified.

TTree writing – active improvements

Users reported several issues -

- root file, written with uproot, number of events in tree issue — 5
  #359 opened 9 days ago by marinang
- root file, written with uproot, size issue — 26
  #345 by marinang was closed 6 days ago
- Error reading with uproot root files, with TTree, written with uproot — 7
  #340 by marinang was closed 20 days ago
- Reading a file written with uproot: TTreeCache::{AddBranch,DropBranch}: unknown branch -> * bug — 31
  #352 by douglasdavis was closed 8 days ago

.. and others.

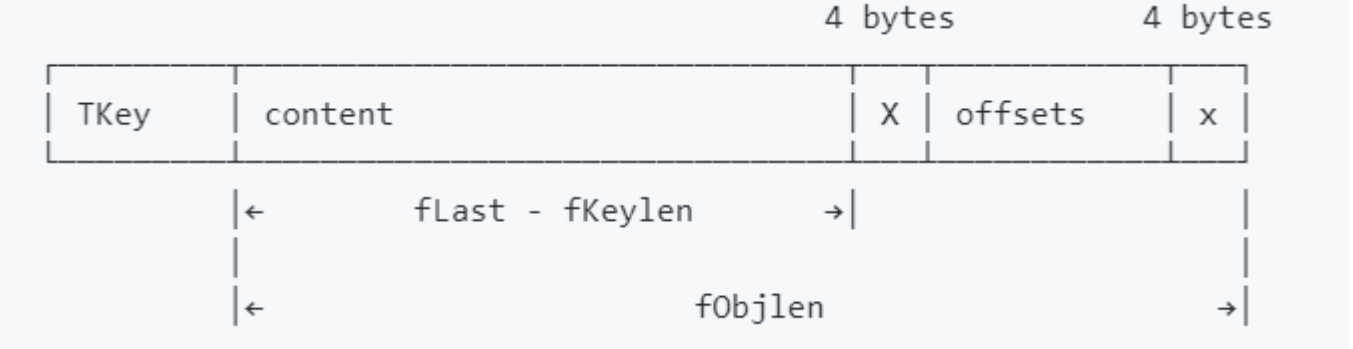
Resulted in several improvements -

- Resize all other branches when one branch overflows ✓
  #368 by reikdas was merged 6 days ago • Approved
- TTree fLeaves now holds references to fLeaves of all branches correctly ✓
  #361 by reikdas was merged 8 days ago • Approved
- Issue352 ✓
  #358 by reikdas was merged 10 days ago • Approved
- Fix issue340 ✓
  #343 by reikdas was merged 20 days ago

Extension - Jagged Arrays

Implementing baskets with Jagged Arrays is simply an extension of the current interface.

Serialization of Jagged Arrays is almost understood (by Jim) -



Offsets – 32 bit integers that indicate the starting byte of each entry.

Only thing left to learn – The 2 blocks of 4 bytes, marked x in the above diagram.

(Should be able to implement once I get a break from school, on some free weekend or if a user stresses that they need it!)

Open areas for work?

We don’t have plans for introducing any major new features to uproot.
There *are* some bug fixes/corner cases or features that are nice to have -

- Write TTrees with multi-dimensional array data `writing-improvements`
  #355 opened 12 days ago by reikdas
- Write TTrees with Jagged Array data `writing-improvements`
  #354 opened 12 days ago by reikdas

!

- Unable to compress data larger than 2^24 bytes `bug` `writing-improvements`
  #333 opened on Sep 9 by reikdas
- ROOT cannot append objects to empty files created by uproot
  #316 opened on Aug 15 by reikdas
- Objects written by uproot and ROOT are read in a different "cycle" order by ROOT `bug`
  #224 opened on Feb 8 by reikdas
- Nested TDirectories `writing-improvements`
  #138 opened on Sep 20, 2018 by jpivarski
- Write more histogram, profile, and graph types `writing-improvements`
  #137 opened on Sep 20, 2018 by jpivarski
- Block management `writing-improvements`
  #135 opened on Sep 20, 2018 by jpivarski

Contributions welcome!

Hear from users – buffered writing?

- Currently, the ROOT TBaskets are as large as the arrays you give to the extend method.

- Do you want an intermediate interface to accumulate arrays before writing them as a single basket to the ROOT file?

Concluding Remarks

- uproot can write TObjStrings, histograms, TTrees with flat data and TTrees with jagged arrays(soon?).
- uproot is in maintenance mode.
- No major features planned for the future.
- uproot 4 coming in early 2020 when the current awkward-array will be replaced by awkward1.0 currently being written by Jim.

Transition `import awkward` → `import awkward0`
and `import awkward1` → `import awkward` in early 2020.

THANK YOU!