hepfile

Wrapping HDF5 to give ROOT-like functionality for HEP datasets and more

Matt Bellis
Siena College, Department of Physics and Astronomy
European HDF5 User Group Meeting
5/31/2022

Relevant links

https://github.com/mattbellis/hepfile

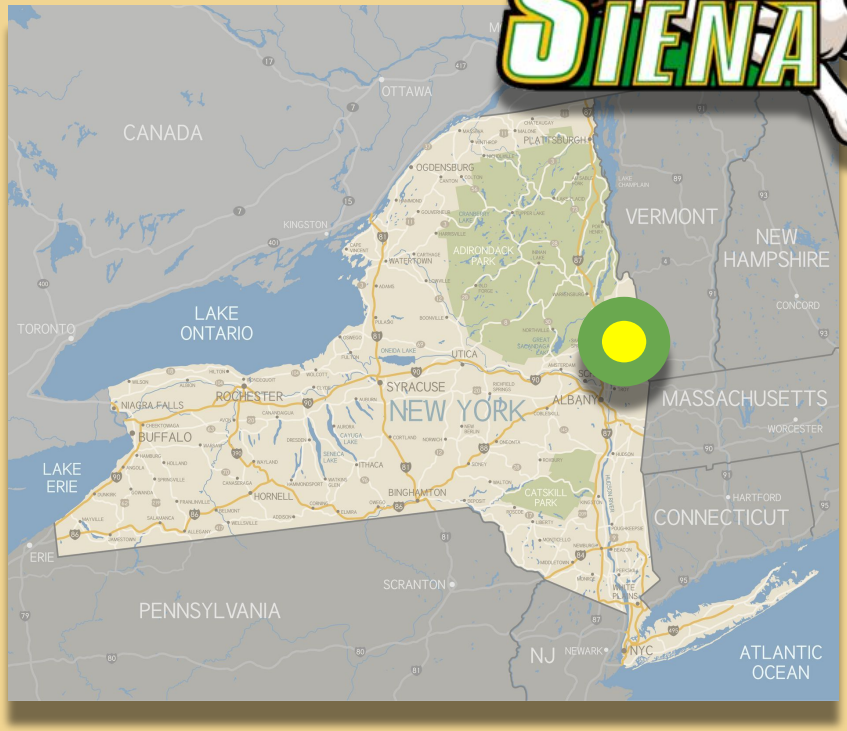
https://hepfile.readthedocs.io/en/latest/index.html

Colab notebook example

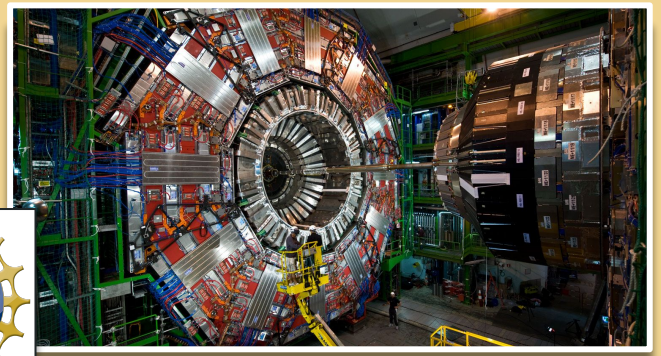
Siena College

Undcrgraduatc-only institution in upstate-NY

~3000 students



Member of CMS experiment since 2013



So much thanks to my student collaborators!

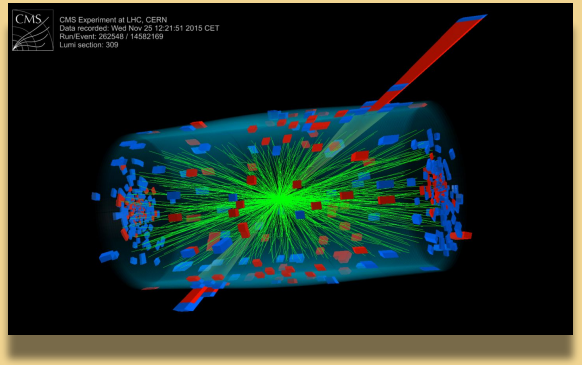
- Willow Hagen
- Matt Dreyer (Cornell)
  - Supported on a DIANA-HEP fellowship in 2021
- Ryan Mikulec
- Gabriella Tamayo

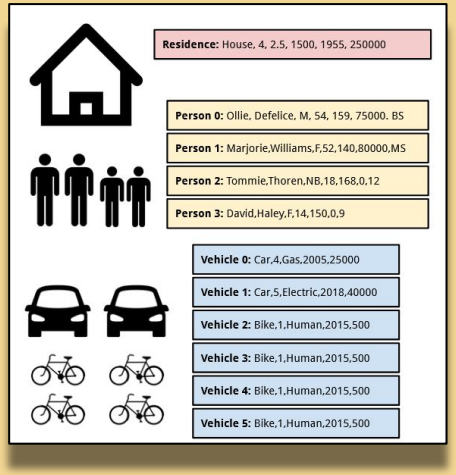
What is the problem we're trying to solve?

- HEP (High Energy Physics) data is heterogeneous and complicated!

- HEP-native example is particle physics experiment
  - Sensors find muons, electrons, jets, etc.
  - Each particle has specific data attached to it (momentum, charge, etc.)
  - Each **event** (collision) might have different numbers of these particles

- Consider a census of a town, with data gathered per household
  - Each **household** has people, cars, and place of residence
  - Each person has name, gender, and age
  - Each car has age and license plate
  - Each house has # of bedrooms and bathrooms





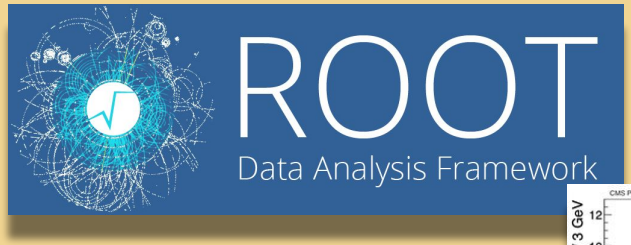
How to solve it

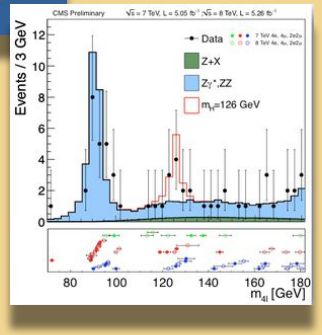
Current solutions: ROOT!

Problems: *Monolithic*

PyROOT helps but still tied to a file format that is bound to the analysis toolkit

Makes it difficult to interface with non-HEP people (e.g. broader computing community)





uproot

Awkward Array

Alternative ROOT-file approaches: uproot/awkward ecosystem

But this is still using the ROOT file format (uproot).

Early attempt - h5hep

- Since 2013, maintained Particle Physics Playground
  - Simplified particle physics data (zipped text files)
  - CMS, BaBar, CLEO
  - Custom python accessors with knowledge of file (text) structure
  - Getting complicated to maintain!

- Package originally called h5hep (*2017*)
  - Begun in earnest at HEP Software Foundation workshop in Annecy, FR. https://indico.cern.ch/event/613093/

- I approached it from the UI/UX/API standpoint first
  - What would I want to type? *Then figure it out!*
  - Didn't want to write an entire file structure from scratch!

- Leaned in to wrapping HDF5 file format
  - Robust; been around; someone else had done the heavy lifting, h5py
  - Datasets stored in group, stored in groups, etc. Very HEP-like

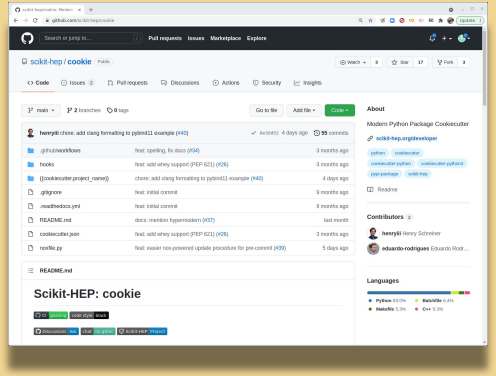
Tried HDF5 as far back as 2014 but couldn't figure out how to use efficiently!

By 2018, h5hep was used in PPP

In 2021, proposed fellowship project through DIANA-HEP and worked with Matt Dryer (Cornell sophomore)

What we had

- Basics of hepfile
- Could be pip installed
- Basics of ReadThe Docs



What we wanted to have

- Add some functionality
  - Store strings
  - Add attributes
- Improve documentation
- Use Henry Schreiner's scikit-cookiecutter to make the package more robust for distribution
- Make code more robust and fault-tolerant
- Add necessary unit tests
- Get CI working
- Submit to JOSS! *(Journal of Open Source Software)*

hepfile is born!

- h5hep ⟶ hepfile (Summer 2021)
- **Define a *schema***
  - How data is organized
  - What metadata needs to be stored to organize the data
- **Define minimal *useful* API for flexibility**
- *Thcn*, implement it in
  - Python
  - HDF5
- Define structure of two python *dictionaries* to help with packing/unpacking data
  - Analogous to ROOT's TTree/Leaf/Branch

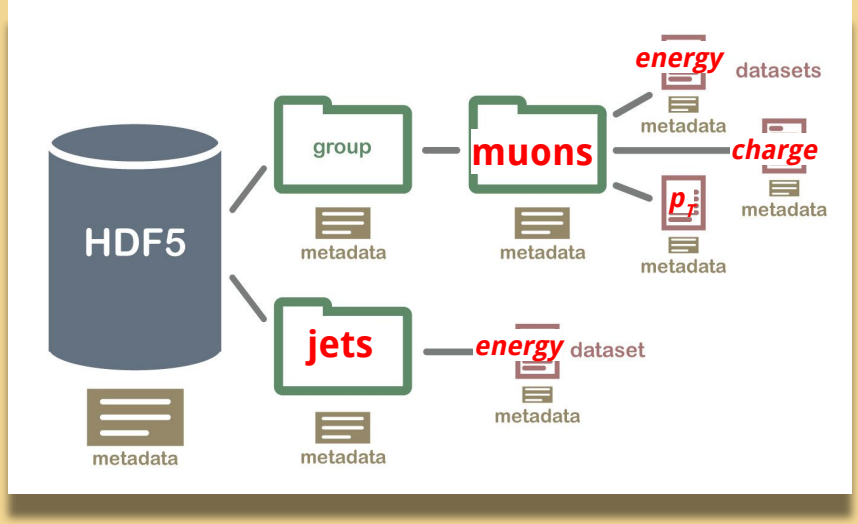
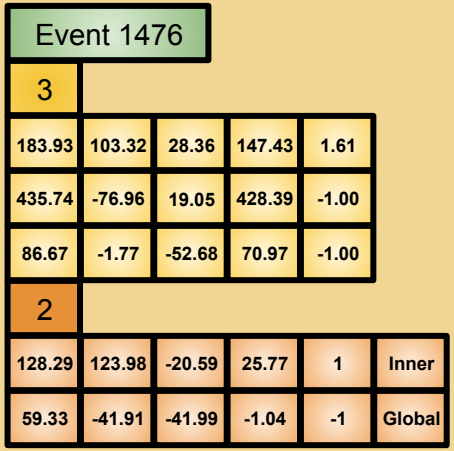
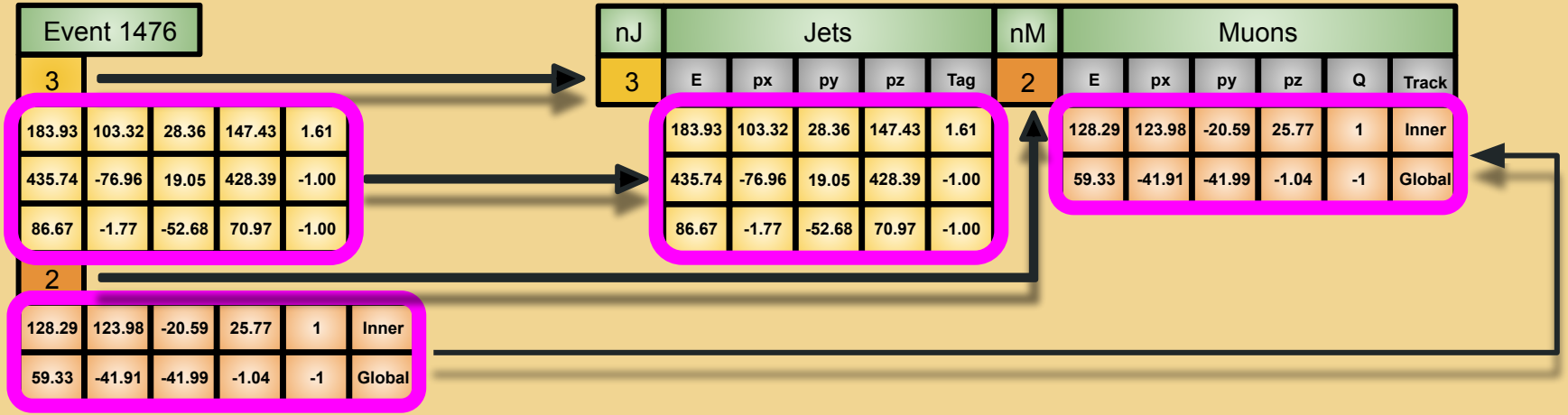


Image credit :
https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5



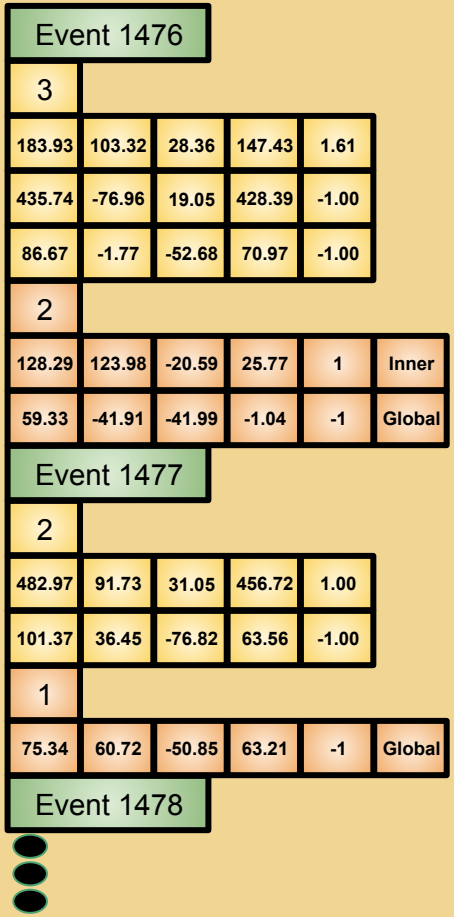
Text representation

hepfile



Text representation

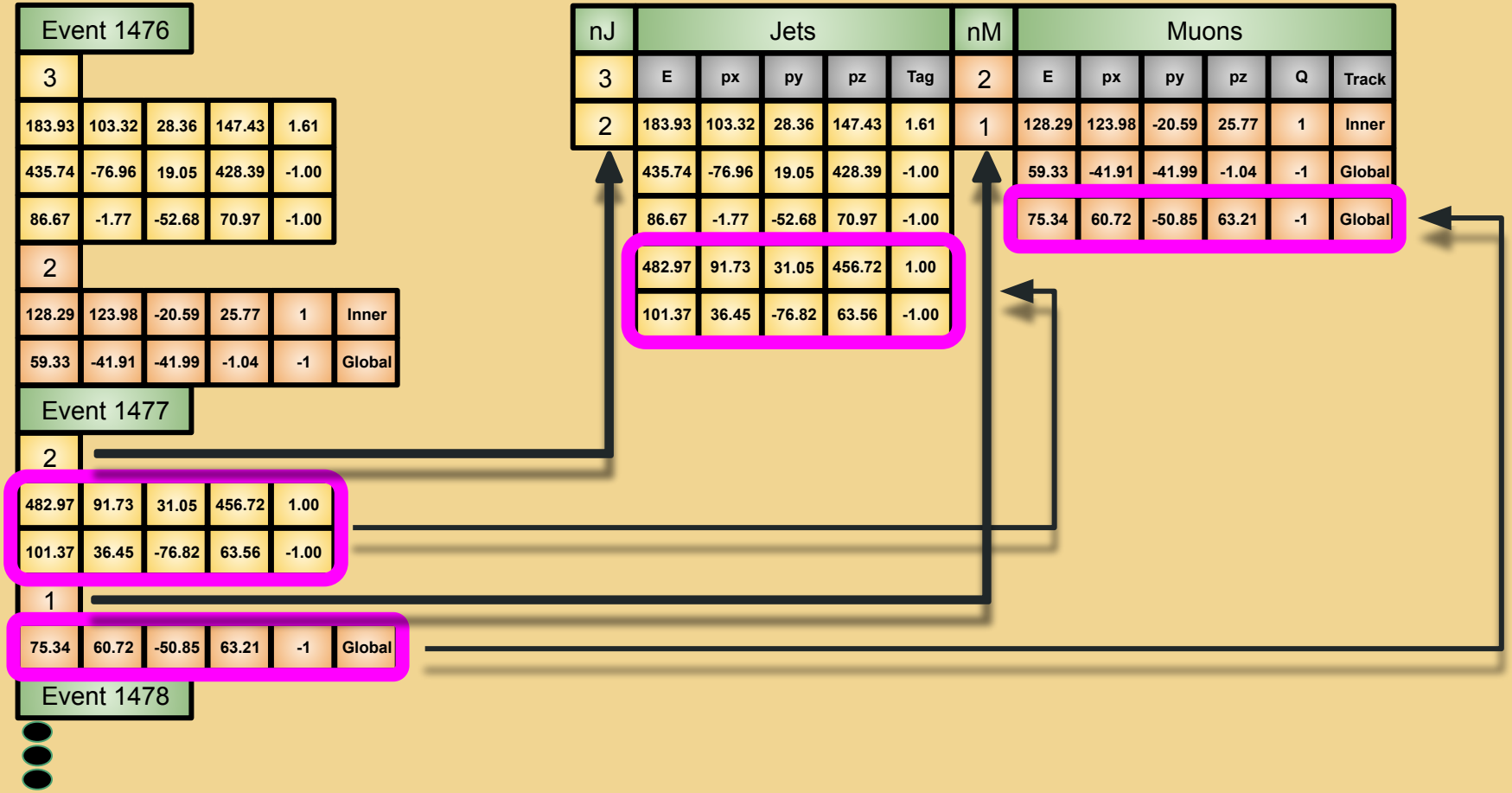
hepfile



Text representation

nJ	Jets					nM	Muons					
3	E	px	py	pz	Tag	2	E	px	py	pz	Q	Track
2	183.93	103.32	28.36	147.43	1.61	1	128.29	123.98	-20.59	25.77	1	Inner
	435.74	-76.96	19.05	428.39	-1.00		59.33	-41.91	-41.99	-1.04	-1	Global
	86.67	-1.77	-52.68	70.97	-1.00							

hepfile



Text representation

hepfile

hepfile - API

```
hepfile.create_group(my_data, 'my_group', counter = 'my_counter')
```

```
hepfile.create_dataset(my_data, 'my_dataset', group = 'my_group', dtype = str)
```

```
hepfile.create_dataset(my_data, ['data1', 'data2'] , group = 'my_group')
```

```
for i in range(5)
    my_bucket['my_group/my_dataset'] = 'yes'
    my_bucket['my_group/data1'] = 1.0
    my_bucket['my_group/data2'] = 2.0

my_bucket['my_unique'] = 3

hepfile.pack(my_data, my_bucket)
```

```
hepfile.write_to_file('my_file.hdf5', my_data)
```

hepfile - Read The Docs is getting populated!

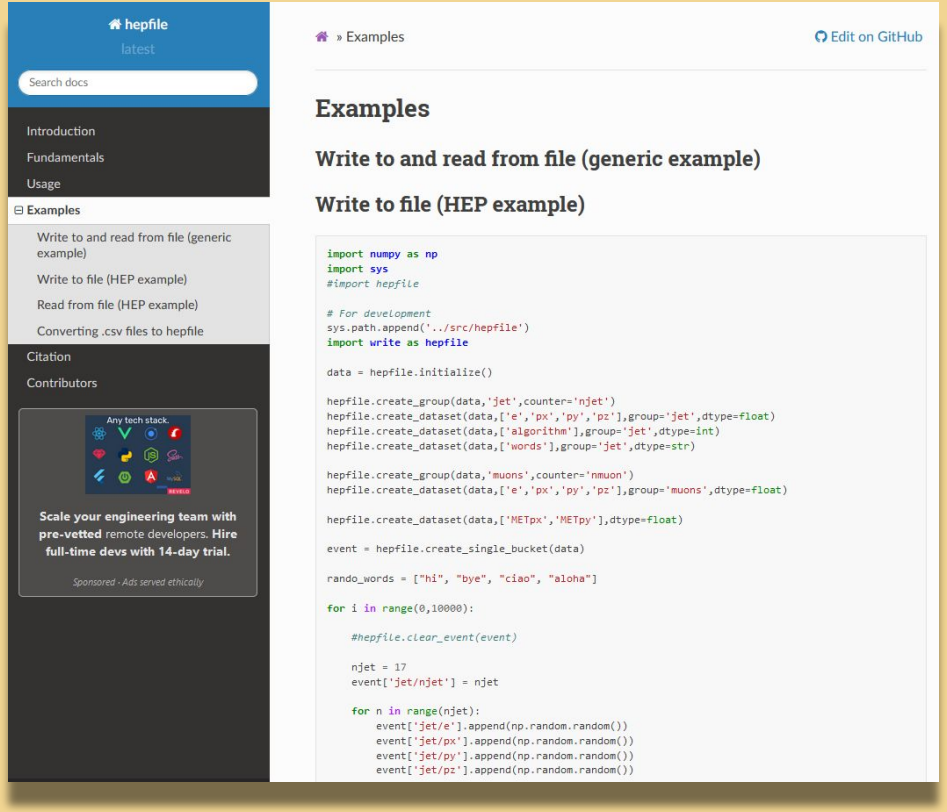
Writing Data with hepfile

Before anything, we extract the data from the .csv files. (Since *houses* will not be its own group, it is not completely necessary to extract *houses_ID*.)

```
people = np.loadtxt('sheet1.csv', unpack=True, dtype=str,
                    delimiter=",", comments = '$')[:, 1:]

vehicles = np.loadtxt('sheet2.csv', unpack=True, dtype=str,
                    delimiter=",", comments = '$')[:, 1:]

houses = np.loadtxt('sheet3.csv', unpack=True, dtype=str,
                    delimiter=",", comments = '$')[:, 1:]

people_ID = people[0][1:].astype(np.int32)
vehicles_ID = vehicles[0][1:].astype(np.int32)
houses_ID = houses[0][1:].astype(np.int32)
```

We create the dictionary where we will be storing our data, and then create the groups inside it. For brevity, the **counter** for the buckets will be *ID*. It is fine to repeat the name of the counter because hepfile will store the counter dataset as `f'{groupname}/ID'`.

```
town = hepfile.initialize()
hepfile.create_group(town, 'people', counter = 'ID')
hepfile.create_group(town, 'vehicles', counter = 'ID')
```



Installation

Local install and development

- Clone from Github
- https://github.com/mattbellis/hepfile

```
git clone https://github.com/mattbellis/hepfile
cd hepfile
flit install
```

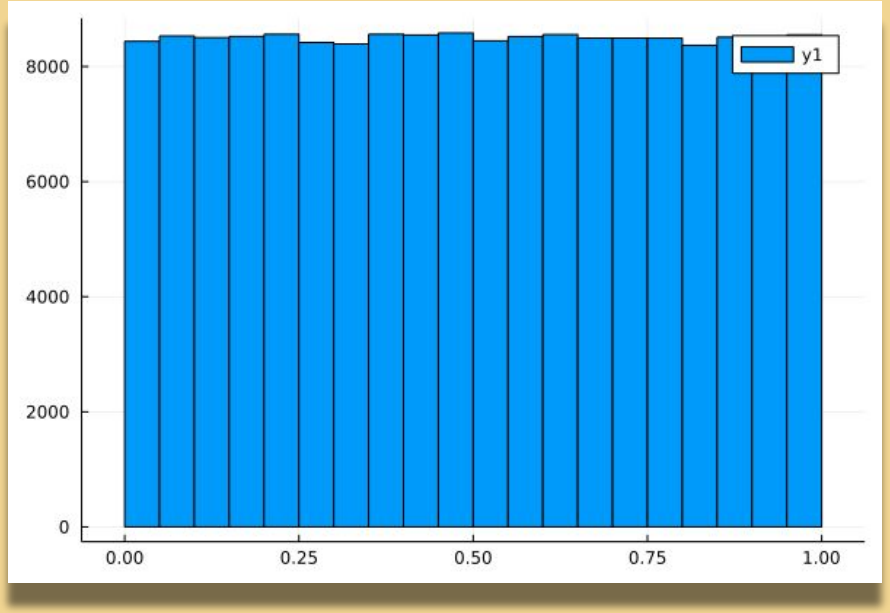
```
pip install hepfile
```

```
pip install hepfile[awkward]
```

In development

Julia test case

```
using HDF5
using Plots

fname = "output.h5"
fid = h5open(fname, "r")

group_names = keys(fid)

for name in group_names
    println(name)
end

jet = read(fid,"jet")
jet_fields = keys(jet)

for field in jet_fields
  println(field)
end

e = jet["e"]

h = histogram(e,bins=25)

savefig("julia_plot_output.png")

gui()
```



With a standard underlying file format (HDF5), it makes it easier for other languages to extract data from the file (*assuming there are HDF5 tools written already*)

Summary

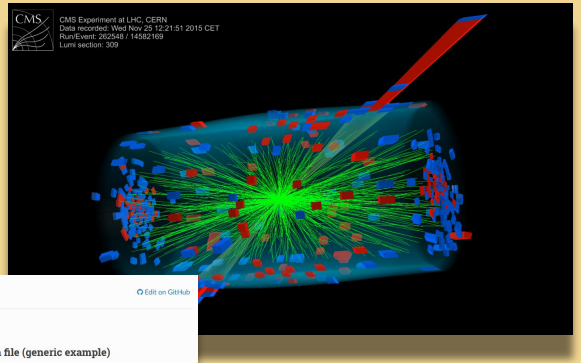
hepfile is in use in Particle Physics Playground (outreach)

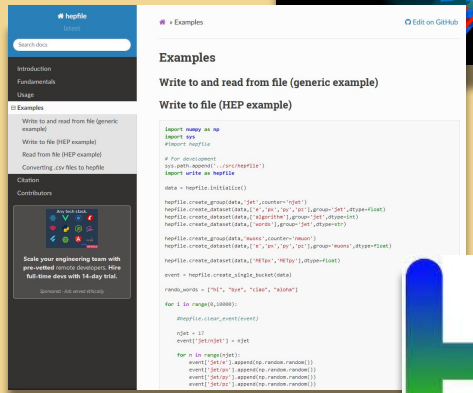
I'm using it regularly for CMS analysis with skimmed data

Interest from others!

To do

- A few more features to add
- Refactor internal storage *(slightly)*
- Finish documentation
- Submit to JOSS





Thank you for your time!

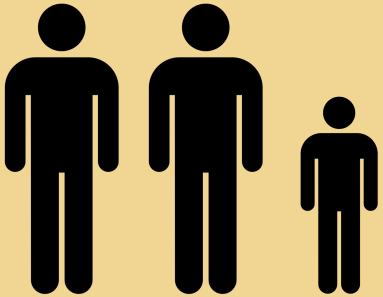
Backup slides

What is hepfile?

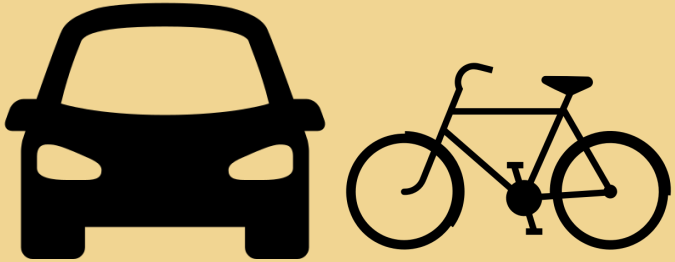
- Data organized into events/buckets
- hepfile groups data of similar types together in datasets
    - Keep bucket data using ‘counter’ field in dataset
- Pack takes buckets -> groups and datasets
- Unpack takes groups and datasets -> buckets
    - Extracts specific bucket ‘i’ from datasets

Residence

People



Vehicles



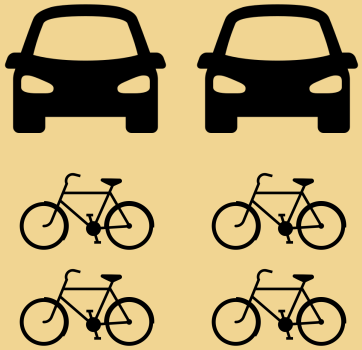
Residence: House, 4, 2.5, 1500, 1955, 250000

Person 0: Ollie, Defelice, M, 54, 159, 75000. BS

Person 1: Marjorie,Williams,F,52,140,80000,MS

Person 2: Tommie,Thoren,NB,18,168,0,12

Person 3: David,Haley,F,14,150,0,9



Vehicle 0: Car,4,Gas,2005,25000

Vehicle 1: Car,5,Electric,2018,40000

Vehicle 2: Bike,1,Human,2015,500

Vehicle 3: Bike,1,Human,2015,500

Vehicle 4: Bike,1,Human,2015,500

Vehicle 5: Bike,1,Human,2015,500

Residence: House, 4, 2.5, 1500, 1955, 250000

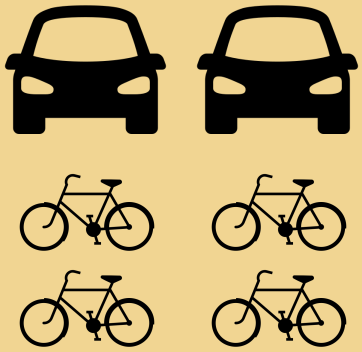
pack

Person 0: Ollie, Defelice, M, 54, 159, 75000. BS

Person 1: Marjorie,Williams,F,52,140,80000,MS

Person 2: Tommie,Thoren,NB,18,168,0,12

Person 3: David,Haley,F,14,150,0,9



Vehicle 0: Car,4,Gas,2005,25000

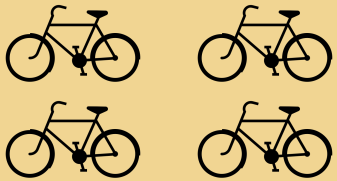
Vehicle 1: Car,5,Electric,2018,40000

Vehicle 2: Bike,1,Human,2015,500

Vehicle 3: Bike,1,Human,2015,500

Vehicle 4: Bike,1,Human,2015,500

Vehicle 5: Bike,1,Human,2015,500



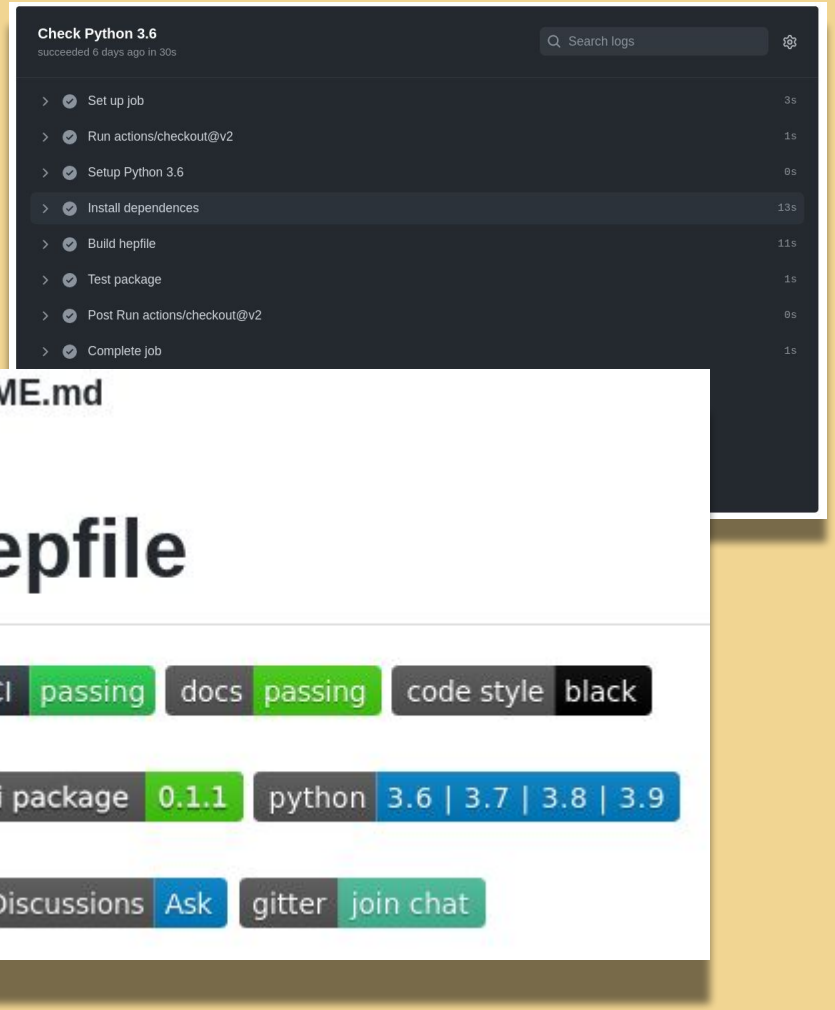
pack


Person 0: Ollie, Defelice,
Person 2: Tommie,Thoren,NB,
Vehicle 3: Bike,
Vehicle 4: Bike, 1,Human,2015,500

Unit tests

- *Unit tests* ensure functionality of the program and use cases are considered

- Continuous Integration
  - Evaluates whether every github commit keeps all unit tests working
  - Using Github Actions for this



hepfile - Read The Docs is getting populated!

hepfile.read module

hepfile.read.calculate_index_from_counters(*counters*)

hepfile.read.get_file_metadata(*filename*)

Get the file metadata and return it as a dictionary

hepfile.read.get_nbuckets_in_data(*data*)

Get the number of buckets in the data dictionary.

This is useful in case you've only pulled out subsets of the data

hepfile.read.get_nbuckets_in_file(*filename*)

Get the number of buckets in the file.

hepfile.read.load(*filename=None, verbose=False, desired_datasets=None, subset=None*)

Reads all, or a subset of the data, from the HDF5 file to fill a data dictionary. Returns an empty dictionary to be filled later with data from individual buckets.

Parameters:
- **filename** (*string*) – Name of the input file
- **verbose** (*boolean*) – True if debug output is required

hepfile.write.pack(*data, bucket, AUTO_SET_COUNTER=True, EMPTY_OUT_BUCKET=True, STRICT_CHECKING=False, verbose=False*)

Takes the data from an bucket and packs it into the data dictionary, intelligently, so that it can be stored and extracted efficiently. (This is analagous to the ROOT TTree::Fill() member function).

Parameters:
- **data** (*dict*) – Data dictionary to hold the entire dataset EDIT.
- **bucket** (*dict*) – bucket to be packed into data.
- **EMPTY_OUT_BUCKET** (*bool*) – If this is *True* then empty out the *bucket* container in preparation for the next iteration. We used to ask the users to do this "by hand" but now do it automatically by default. We allow the user to not do this, if they are running some sort of debugging.

hepfile.write.write_file_metadata(*filename, mydict={}, write_default_values=True, append=True*)

Writes file metadata in the attributes of an HDF5 file

Args: **filename** (string): Name of output file

mydict (dictionary): Metadata desired by user

write_default_values (boolean): True if user wants to write/update the

default metadata: date, hepfile version, h5py version, numpy version, and Python version, false if otherwise.

append (boolean): True if user wants to keep older metadata, false otherwise.

Returns: **hdoutfile** (HDF5): File with new metadata