YAML as an ADL: YADL (?)

The F.A.S.T. analysis tools

Ben Krikler
7th May 2019

Analysis Description Languages for the LHC:
indico.cern.ch/event/769263/

Outline of this talk

Why YAML?

How do the tools work?

What's next?

Why focus on YAML?

- An existing markup: many parsers exist
- A superset of JSON
  - Static object description (dicts, lists, numbers, strings)
  - Adds anchors and references: reuse common occurrences
- Easier to read than JSON:
  - Can write without brackets and braces
  - Indentation to imply nesting (c.f. python)
- Naturally declarative: No “control flow” (e.g. no for loops)
- Widely used to describe pipeline configuration:
  - gitlab-CI, travis-CI, Azure CI/CD, Ansible, Kubernetes, etc
  - HEPData: YAML for reproducible Data

How do we use YAML at this point?

Four separate types of YAML file, to answer:

1. What are your datasets?
2. ***How to process them into tables (histograms and cut-flow efficiencies)?***
3. How to convert histograms into fitting inputs?
4. How to visualise outputs?

"Hang on Ben, isn't that just a set of configuration files?"
Much of this project is building towards a DSL *but that's something of a secondary consequence* of the key goal behind all of this….

Ask not “what does my code need me to do for it”

but “what do I want my code to do for me”

The F.A.S.T tools...

FAST codebase

A minimal-viable product to develop these ideas

- Changing rapidly and often
- Some sizeable changes on the roadmap

Developed largely by myself and a couple of others

But being used in some form for **2 CMS analyses** and students on **DUNE, FCC, LUX-ZEPLIN** experiments

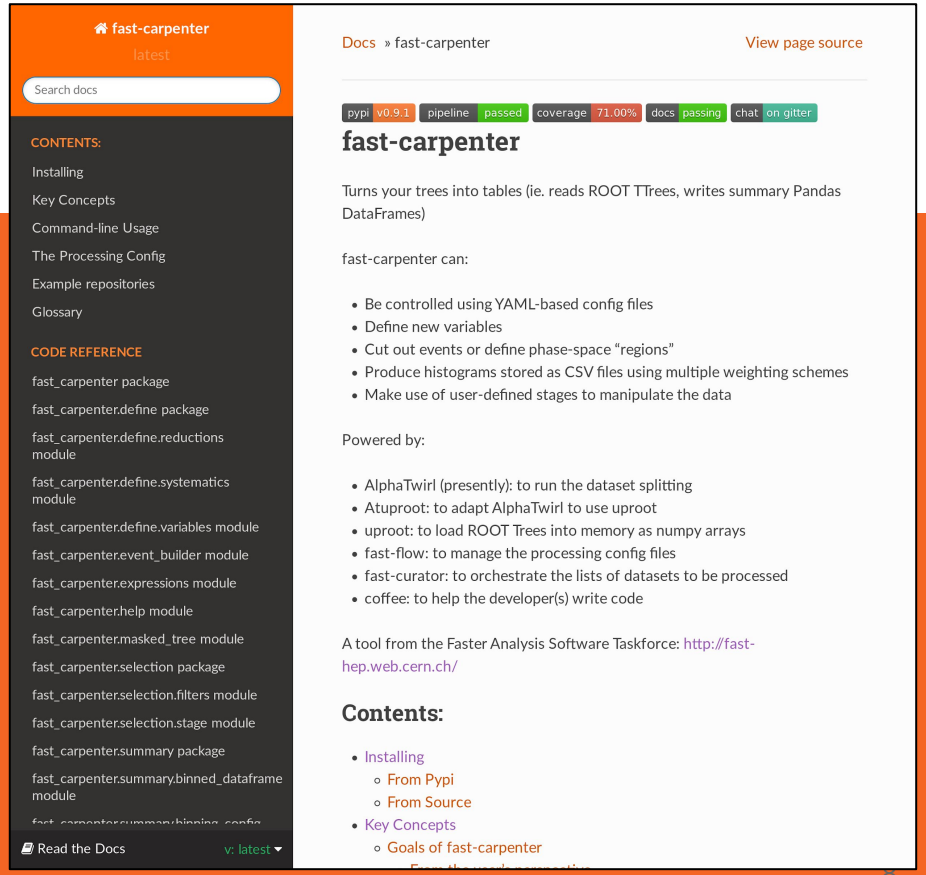
- Have seen students copy snippets verbatim into talks to other collaboration members

Code to handle configs and execute all written in Python

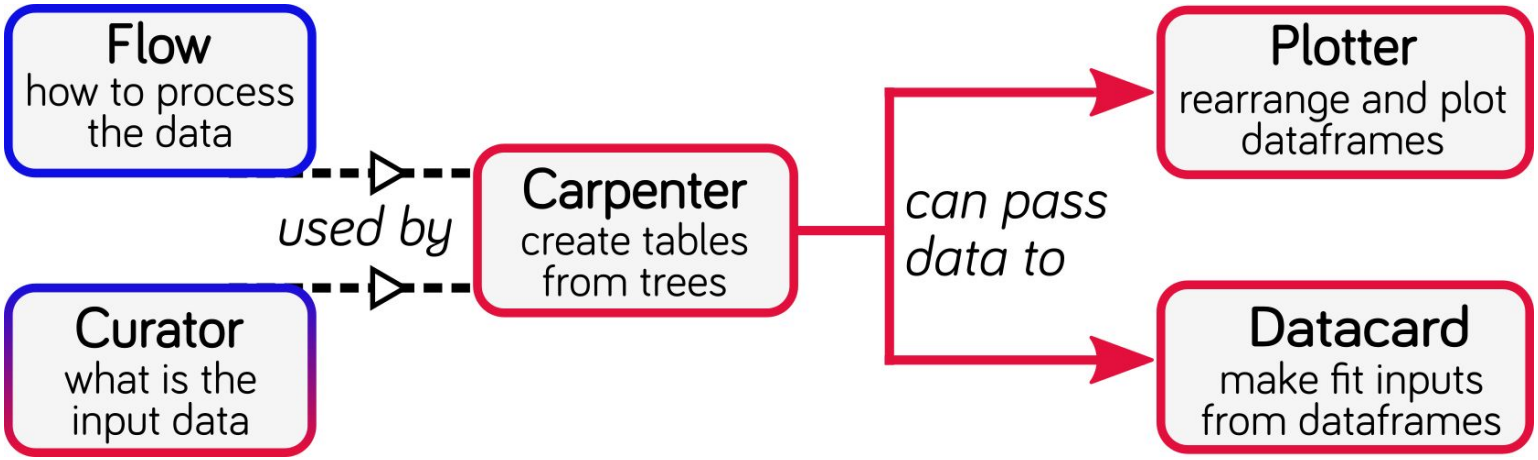
- Use existing tools as much as possible
- My goal: code written by me $\rightarrow 0$

Where to find the code

- Docs: fast-carpenter.readthedocs.io/
- All on CERN's gitlab, likely to move to github soon
  - https://gitlab.cern.ch/fast-hep/public
  - Main package: gitlab.cern.ch/fast-hep/public/fast-carpenter
- Demo repository where most examples in this talk come from: gitlab.cern.ch/fast-hep/public/fast_cms_public_tutorial



FAST codebase interplay



Changes since IRIS-HEP presentation (4th March)

https://indico.cern.ch/event/802182/contributions/3334624/

1. Documentation on readthedocs
2. Stage to produce a ghast
3. Import mechanism for processing description
4. Optimisation of various stages behind the scenes

How it works...

Anatomy of the processing description

What type of action to take at each step:

- Stage1 = A built-in stage of fast-carpenter
- Stage2 = A stage imported from a python module
- IMPORT = Import a list of stages and their descriptions from another YAML file

For each stage named above:

- Provide a dictionary of keyword arguments
- Passed through to stage's init method

```yaml
stages:
  - Stage1: StageFromBackend
  - Stage2: module.that.provides.some.Stage
  - IMPORT: "{this_dir}/another_description.yaml"


Stage1:
  keyword: value
  another_keyword: [a, list, of, values]


Stage2:
  arg1: 35
  arg2:
      takes: ["a", "dict"]
      with: 3
      different: keys
```

Stages section: What do you want to do with the data?

```
stages:
  # Just defines new variables
  - BasicVars: fast_carpenter.Define
  # A custom class to form the invariant mass of a
  # two-object system
  - DiMuons: cms_hep_tutorial.DiObjectMass
  # Filled a binned dataframe
  - NumberMuons: fast_carpenter.BinnedDataframe
  # Select events by applying cuts
  - EventSelection: fast_carpenter.CutFlow
  # Fill another binned dataframe
  - DiMuonMass: fast_carpenter.BinnedDataframe
```

(Currently) a *sequence* of stages and their descriptions.

Each stage:

- Can be any python importable class
- Should implement three or four key processing methods
- Fast-carpenter provides several stages

For example:

1. Define some variables
2. Make a histogram
3. Cut out some events
4. Make another histogram

Define Stage: fast_carpenter.Define

```yaml
BasicVars:
  variables:
      - Muon_Pt: "sqrt(Muon_Px ** 2 + Muon_Py ** 2)"
      - IsoMuon_Idx: (Muon_Iso / Muon_Pt) < 0.10
      # This next variable will create a single
      # number for each event, using a set of inputs
      # whose length varies for each event
      - NIsoMuon:
          formula: IsoMuon_Idx
          reduce: count_nonzero
      - HasTwoMuons: NIsoMuon >= 2
      # Capture first muon's Pt, padded
      # with NaNs if NMuon < 1
      - Muon_lead_Pt: {reduce: 0, formula: Muon_Pt}
      # Capture second muon's Pt, padded
      # with NaNs if NMuon < 2
      - Muon_sublead_Pt: {reduce: 1, formula: Muon_Pt}
```

- Combines uproot + numexpr (v2)
  - Presents a dict-like object to numexpr, containing uproot tree and other new variables
  - https://gitlab.cern.ch/fast-hep/public/fast-carpenter/blob/master/fast_carpenter/expressions.py
  - All input variable in expression need same “jaggedness”
  - In future: numpy-like broadcasting across jaggedness
- Additional reductions: object-level variables (jagged arrays) to event-level
- Adds variables into tree using replaced `itervalues` method
- Can some of this become central functionality within uproot(-methods) in the future?

Select events

fast_carpenter.CutFlow

```
DiMu_controlRegion:
    weights: {nominal: weight}
    selection:
        All:
          - {reduce: 0, formula: Muon_pt > 30}
          - leadJet_pt > 100
          - All:
            - DiMuon_mass > 60
            - DiMuon_mass < 120
          - Any:
            - nCleanedJet == 1
            - DiJet_mass < 500
            - DiJet_deta < 2
```

Masks events from subsequent stages

Produces a cut-flow summary with:

- Raw and weighted yields
- Inclusive and exclusive yields to each cut

Selection is specified as a nested dictionary of All and Any and a list of cuts

- Inspired by Tai Sakuma's approach in AlphaTwirl

Individual cuts use same scheme as variable definition

```
EventSelection:
  weights: {weighted: EventWeight}
  selection:
      All:
      - NIsoMuon >= 2
      - triggerIsoMu24 == 1
      - {reduce: 0, formula: Muon_Pt > 25}
```

Select events

fast_carpenter. CutFlow

Resulting cut-flow outputs from EventSelection config on last slide

```
>>> import pandas as pd
>>> pd.read_csv("cuts_EventSelection-weighted.csv", header=[0, 1], index_col=[0, 1, 2])
```

			passed_incl		passed_excl		totals_excl	
			unweighted	EventWeight	unweighted	EventWeight	unweighted	EventWeight
dataset	depth	cut						
data	0	All	15995.0	15995.000000	15995.0	15995.000000	469384.0	469384.000000
	1	NIsoMuon >= 2	16208.0	16208.000000	16208.0	16208.000000	469384.0	469384.000000
		triggerIsoMu24 == 1	469384.0	469384.000000	16208.0	16208.000000	16208.0	16208.000000
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	229710.0	229710.000000	15995.0	15995.000000	16208.0	16208.000000
dy	0	All	37263.0	16628.843750	37263.0	16628.843750	77729.0	34115.511719
	1	NIsoMuon >= 2	37559.0	16829.451172	37559.0	16829.451172	77729.0	34115.511719
		triggerIsoMu24 == 1	77729.0	34115.511719	37559.0	16829.451172	37559.0	16829.451172
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	73374.0	32168.121094	37263.0	16628.843750	37559.0	16829.451172
qcd	0	All	0.0	0.000000	0.0	0.000000	142.0	79160.507812
	1	NIsoMuon >= 2	0.0	0.000000	0.0	0.000000	142.0	79160.507812
		triggerIsoMu24 == 1	142.0	79160.507812	0.0	0.000000	0.0	0.000000
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	16.0	6014.819336	0.0	0.000000	0.0	0.000000
single_top	0	All	110.0	5.676235	110.0	5.676235	5684.0	311.622986
	1	NIsoMuon >= 2	111.0	5.748312	111.0	5.748312	5684.0	311.622986
		triggerIsoMu24 == 1	5684.0	311.622986	111.0	5.748312	111.0	5.748312
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	5278.0	290.494965	110.0	5.676235	111.0	5.748312
ttbar	0	All	206.0	47.293686	206.0	47.293686	36941.0	7929.475586
	1	NIsoMuon >= 2	226.0	51.629749	226.0	51.629749	36941.0	7929.475586
		triggerIsoMu24 == 1	4515.0	1001.804932	206.0	47.293686	226.0	51.629749
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	5067.0	1109.433960	206.0	47.293686	206.0	47.293686
wjets	0	All	1.0	0.311917	1.0	0.311917	109737.0	209603.531250
	1	NIsoMuon >= 2	1.0	0.311917	1.0	0.311917	109737.0	209603.531250
		triggerIsoMu24 == 1	109737.0	209603.531250	1.0	0.311917	1.0	0.311917
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	99016.0	191354.781250	1.0	0.311917	1.0	0.311917
ww	0	All	243.0	12.577849	243.0	12.577849	4580.0	229.949570
	1	NIsoMuon >= 2	244.0	12.639496	244.0	12.639496	4580.0	229.949570
		triggerIsoMu24 == 1	4580.0	229.949570	244.0	12.639496	244.0	12.639496
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	4214.0	212.997131	243.0	12.577849	244.0	12.639496
wz	0	All	623.0	13.157759	623.0	13.157759	3367.0	69.927917
	1	NIsoMuon >= 2	623.0	13.157759	623.0	13.157759	3367.0	69.927917
		triggerIsoMu24 == 1	3367.0	69.927917	623.0	13.157759	623.0	13.157759
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	3125.0	65.436157	623.0	13.157759	623.0	13.157759
zz	0	All	1232.0	8.985804	1232.0	8.985804	2421.0	16.922522
	1	NIsoMuon >= 2	1235.0	8.998816	1235.0	8.998816	2421.0	16.922522
		triggerIsoMu24 == 1	2421.0	16.922522	1235.0	8.998816	1235.0	8.998816
		{'formula': 'Muon_Pt > 25', 'reduce': 0}	2325.0	16.362473	1232.0	8.985804	1235.0	8.998816

Fill a histogram

fast_carpenter.BinnedDataFrame
fast_carpenter.BuildAghast

```yaml
NumberMuons:
  dataset_col: true
  binning:
      - {in: NMuon, out: nMuons}
      - {in: NIsoMuon, out: nIsoMuons}
  weights: [EventWeight, EventWeight_NLO_up]

DiMuonMass:
  dataset_col: true
  binning:
      - in: DiMuon_Mass
        out: dimu_mass
        bins: {low: 60, high: 120, nbins: 60}
  weights: {weighted: EventWeight}
```

- Binning scheme:
  - Assume variable already discrete (eg. NumberMuons)
  - Equal-width bins over a range (eg. DiMuonMass)
  - List of bin edges
  - Special: bin by dataset name
- Weighting schemes:
  - None
  - Single weight variable
  - List of weight variables
  - Mapping of names to input variable
- Output written to disk:
  - Pandas to produce a dataframe (csv)
  - Also to a Ghast for future tooling

Fill a histogram: Resulting CSV from DiMuonMass

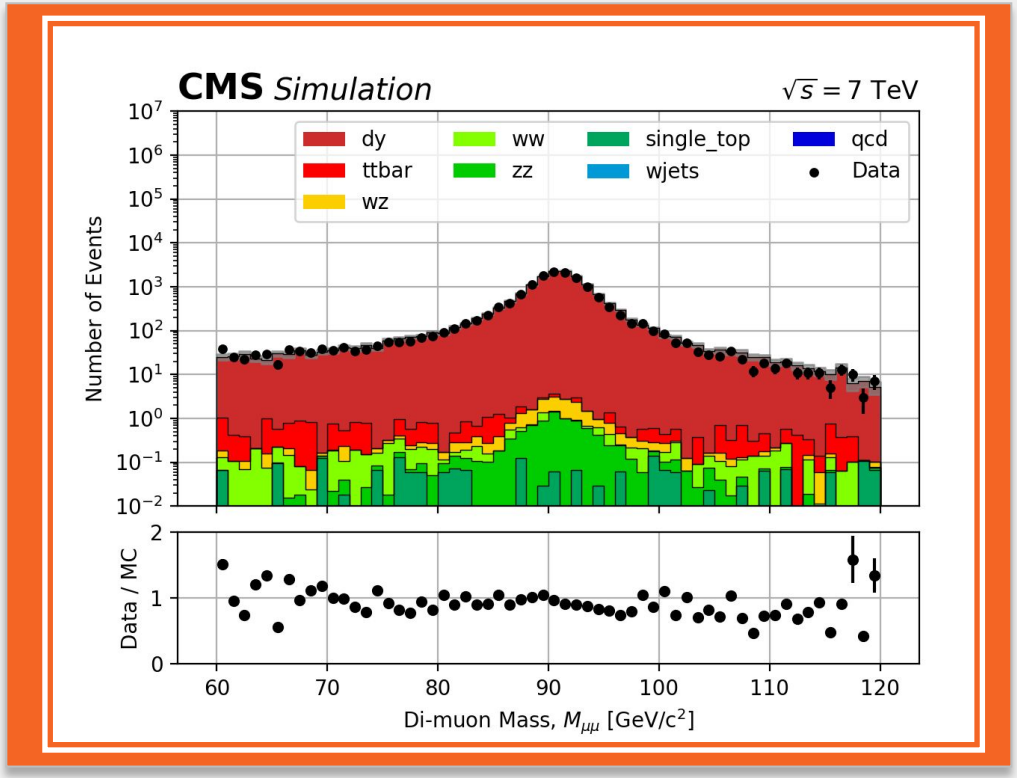
Showing only first three rows for each dataset (using groupby operation)

```
>>> import pandas as pd
>>> df = pd.read_csv('tbl_dataset.dimu_mass--weighted.csv')
>>> print(df.groupby('dataset').nth([0, 1, 2]).set_index('dimu_mass', append=True))
                                n  weighted:sumw  weighted:sumw2
dataset    dimu_mass
data       (-inf, 60.0]  993.0            NaN             NaN
           (60.0, 61.0]   38.0            NaN             NaN
           (61.0, 62.0]   25.0            NaN             NaN
dy         (-inf, 60.0]  821.0     655.570801     1017.549133
           (60.0, 61.0]   56.0      23.963226       12.091142
           (61.0, 62.0]   56.0      25.572840       13.094129
qcd        (-inf, 60.0]    0.0       0.000000        0.000000
           (60.0, 61.0]    0.0       0.000000        0.000000
           (61.0, 62.0]    0.0       0.000000        0.000000
single_top (-inf, 60.0]   32.0       1.741041        0.100682
           (60.0, 61.0]    1.0       0.065288        0.004263
           (61.0, 62.0]    1.0       0.005831        0.000034
ttbar      (-inf, 60.0]   49.0      11.392980        3.072051
           (60.0, 61.0]    3.0       0.840432        0.236490
           (61.0, 62.0]    2.0       0.319709        0.075986
wjets      (-inf, 60.0]    1.0       0.311917        0.097292
           (60.0, 61.0]    0.0       0.000000        0.000000
           (61.0, 62.0]    0.0       0.000000        0.000000
ww         (-inf, 60.0]   61.0       3.600221        0.221474
           (60.0, 61.0]    1.0       0.063284        0.004005
           (61.0, 62.0]    2.0       0.102053        0.005617
wz         (-inf, 60.0]   15.0       0.320914        0.007842
           (60.0, 61.0]    2.0       0.053328        0.001424
           (61.0, 62.0]    0.0       0.000000        0.000000
zz         (-inf, 60.0]   47.0       0.360053        0.002981
           (60.0, 61.0]    0.0       0.000000        0.000000
           (61.0, 62.0]    0.0       0.000000        0.000000
```

Turning outputs into plots: fast-plotter

- Philosophy of: easy to produce basic plots, tools to support final publication-quality:
  - Command-line tool with good defaults and simple configuration
  - Written in lots of small functions that can help a user in a dedicated script / notebook
- Plot on the right with:

```
fast_plotter -y log \
-c plot_config.yml \
-o tbl_*.csv
```



Plot of DiMuonMass binned dataframe from last slide

User-defined stages

```
stages:
  - BasicVars: fast_carpenter.Define
  - DiMuons: cms_hep_tutorial.DiObjectMass
  - Histogram: BinnedDataframe
```

- This is a growing MVP
- Previous steps not able to capture all analysis needs (yet), eg:
  - More complex variable definition (e.g. invariant masses)
  - Scale factor look-ups
- But a stage needn't belong to fast_carpenter
  - Break out of declarative YAML to full, imperative python
- Any importable python class with the correct interface can be used:
  - __init__ method accepts at least a name and output directory path
  - An `event` method
  - Optionally: `begin`, `end`, and `collector` methods
  - Collector used to write to disk if wanted
- Example: [fast_cms_public_tutorial/cms_hep_tutorial/__init__.py]

Describe your datasets: fast-curator

```yaml
import:
  - "{this_dir}/WW.yml"
  - "{this_dir}/WZ.yml"
datasets:
  - eventtype: data
    Files: [input_files/HEPTutorial/files/data.root]
    name: data
    nevents: 469384
  - files: [input_files/HEPTutorial/files/dy.root]
    name: dy
    nevents: 77729
  - files: [input_files/HEPTutorial/files/qcd.root]
    name: qcd
    nevents: 142
defaults:
  eventtype: mc
  nfiles: 1
  tree: events
```

Mainly for scaling up to full analysis

- Many input files form a dataset
- Many datasets make up an analysis

Shouldn't need to produce dataset descriptions too often

- Analyses keep dataset files in repo, easy to review

Command line tool to help you:

- Eg. wild-card on the command line, including xrootd files (contributed to pyxrootd)

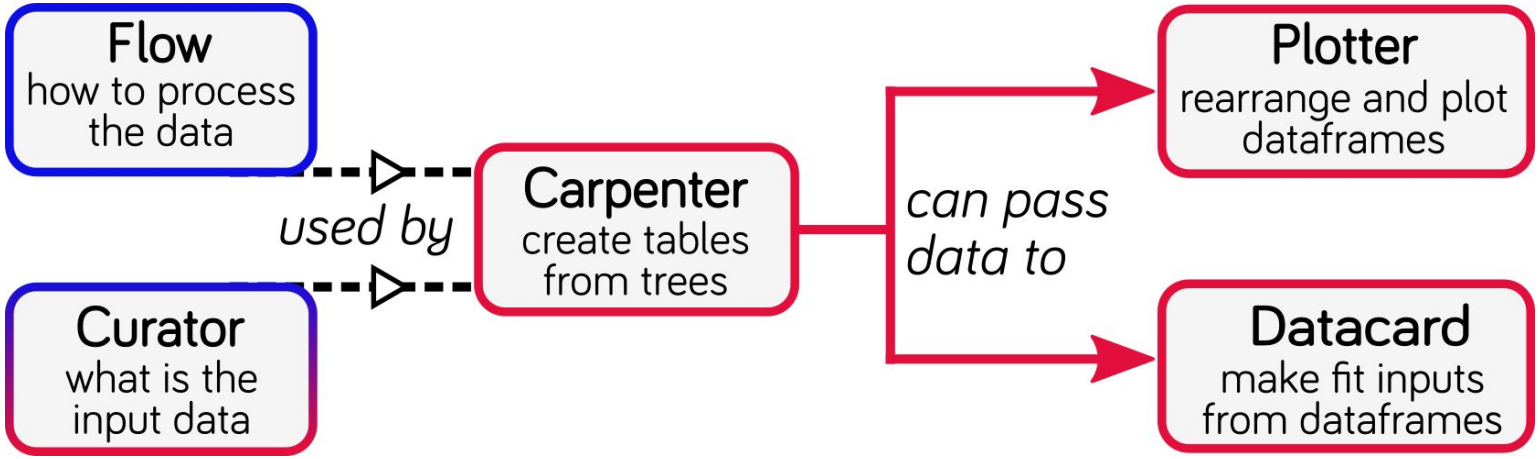
Dataset types can be "data" or "MC"

- Event weights only applied to MC

Can include additional meta-data e.g.:

- Cross-sections
- Number of original events (for processing skimmed inputs)
- Event tree name

FAST codebase interplay



- **fast-flow**: Library code only
- **fast-curator**: Executables to write and validate YAML
- **fast-carpenter**: Main entry-point. Default backend and stages
- **fast-plotter**: Executable with good defaults, small functions to assist custom scripts
- **fast-datacard**: Specific to CMS Higgs Combine inputs at this point

Just how “fast” is this?

Compared to C++ example analysis, single core

- Fast-carpenter: 6 seconds, ~100 lines of analysis-specific code
- C++ example: 3 seconds, >600 lines of code

Apples-to-apples comparison is tricky

Many places ripe for optimisation:

- Overly-general histogram filling using pandas
- Combining multi-processed jobs’ output
- Caching and optimisation of variable definitions (a la histbook ?)

Going forwards...

Really using YAML as an ADL

YAML descriptions from previous slides specifically tied to fast-carpenter and friends.

Could this be “standardised” into a full language = YADL

Stage provides the same interface and outputs: its implementing the YADL standard for such a stage, e.g.:

- Variable definition expressions
- Cut-flows with nested dictionaries

Fast-flow already provides a “backend” mechanism

- Develop further: allow user to select backend
- E.g.: AlphaTwirl (current), Spark, RDataFrame

Roadmap for changes

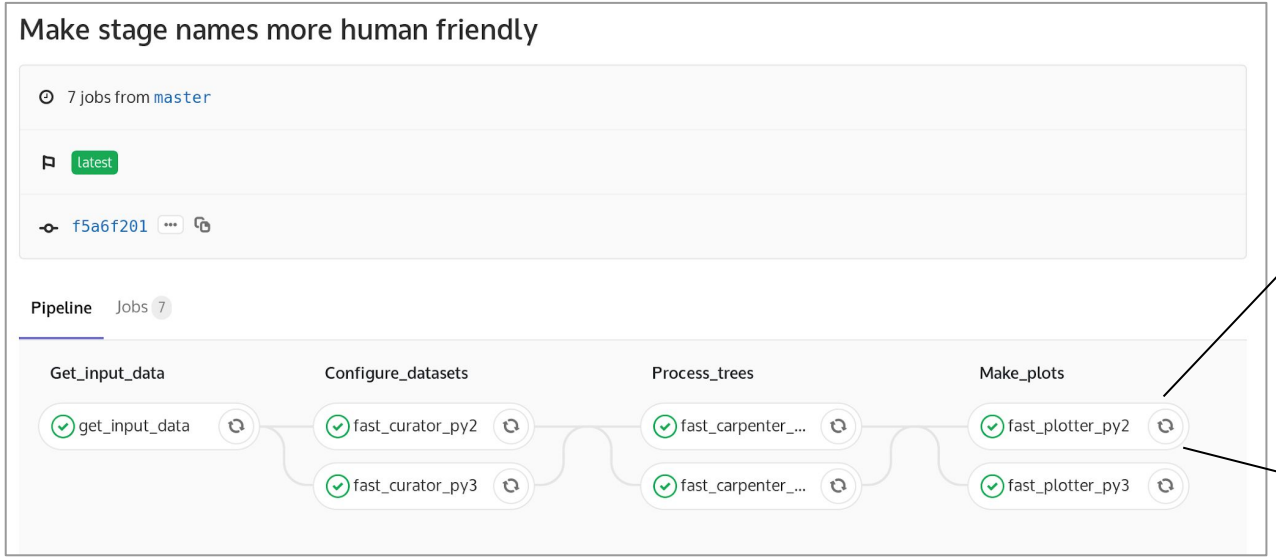
1. Generalised data-space concept:
   - Not just individual (jagged)-arrays
   - Tables, dataframes, ghasts, etc
   - Remove uproot tree “branch injection” hacks
2. Improved expressions:
   - Pre-process on top of numexpr
   - Move reductions into expressions
   - Separate package?
3. Flow control model
   - Stages to become more functional
   - Move to PARSL, SAGA-Python
   - Support Spark integration
4. Type system for configurations
   - Via Python 3 type annotations?
5. Collaboration with similar efforts?
   - E.g. using functions from Coffea

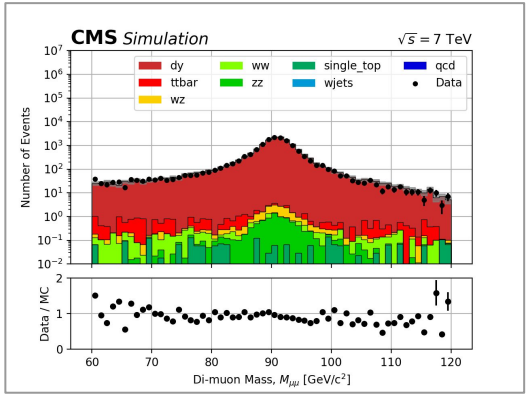
Summary

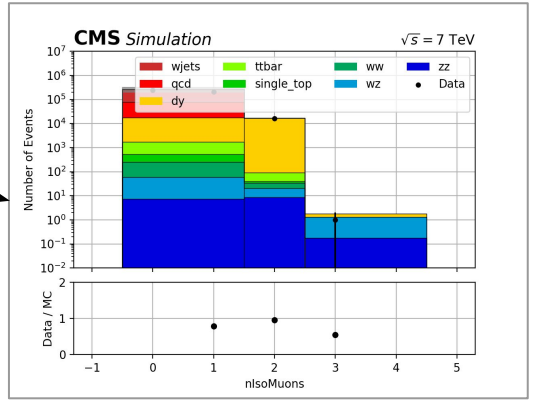
- Have introduced the FAST codebase
  - Incorporating uproot, awkward array, numexpr, aghast
  - Being used on CMS and several other experiments
- YAML-based analysis description
  - Datasets, processing, plotting steps
  - Not too much work to “standardize” this beyond existing backend
- About twice as slow as equivalent C++ analysis (single core)
  - But lots of room for optimisation
- Resources
  - Code: gitlab.cern.ch/fast-hep/public/fast-carpenter/
  - Installing: pypi.org/project/fast-carpenter/
  - Docs: fast-carpenter.readthedocs.io/
  - Gitter: gitter.im/FAST-HEP/community

Thank You

“Analysis in a CI pipeline”







- To run this:
  - Demo analysis in a pipeline
  - The gitlab-ci config
  - Script tying the commands together
- Feasible for huge datasets unclear, but can happily manage subsets of data for testing

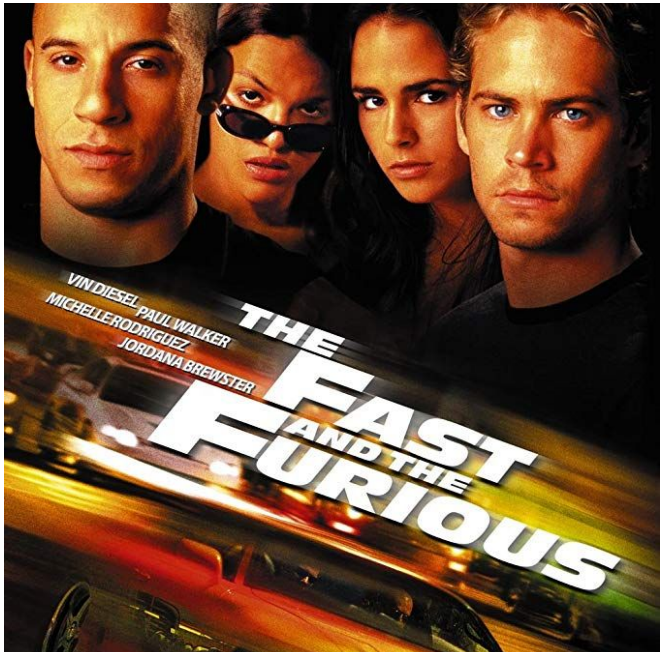
Fill a histogram: Technical implementation details

- First load necessary branches into pandas dataframe
- Then one highly general function to
  - Discretize (i.e. bin) variables if needed (using pandas.cut)
  - Aggregate (groupby) and produce counts, sum of (multiple) weights, and sum of square of (multiple) weights
- This covers all cases but not optimal in many common uses, e.g.:
  - Single variable to bin on
  - Unweighted counts
- Can optimise behind the scenes
  - https://iscinumpy.gitlab.io/post/histogram-speeds-in-python/
  - Config file doesn't have to change

FAST = Faster Analysis Software Taskforce

- Group of HEP researchers
- Primarily working for UK institutes
- Started around May 2017
- Use of 1 to 3-day “hack-shops” to test new ideas

- Goals: Try to help improve HEP analysis software
  a. Simplicity
  b. Speed
  c. Documentation
  d. Automation



https://www.imdb.com/title/tt0232500/

No connection to this work, just an excuse to include an image

FAST Hack-shops

- Halfway between hackathon and a workshop
- Small (<10) group of people working together for 1 to 3 days
- People / pairs working on self-defined projects (eg. set up PARSL within fast-carpenter)
- Mainly for trying out new tools, experimenting with new ideas
- 4 hack-shops held throughout last two years, but only within FAST
- Would there be interest to extend this to HSF / IRIS-HEP / scikit-hep more broadly?

A little bit of context