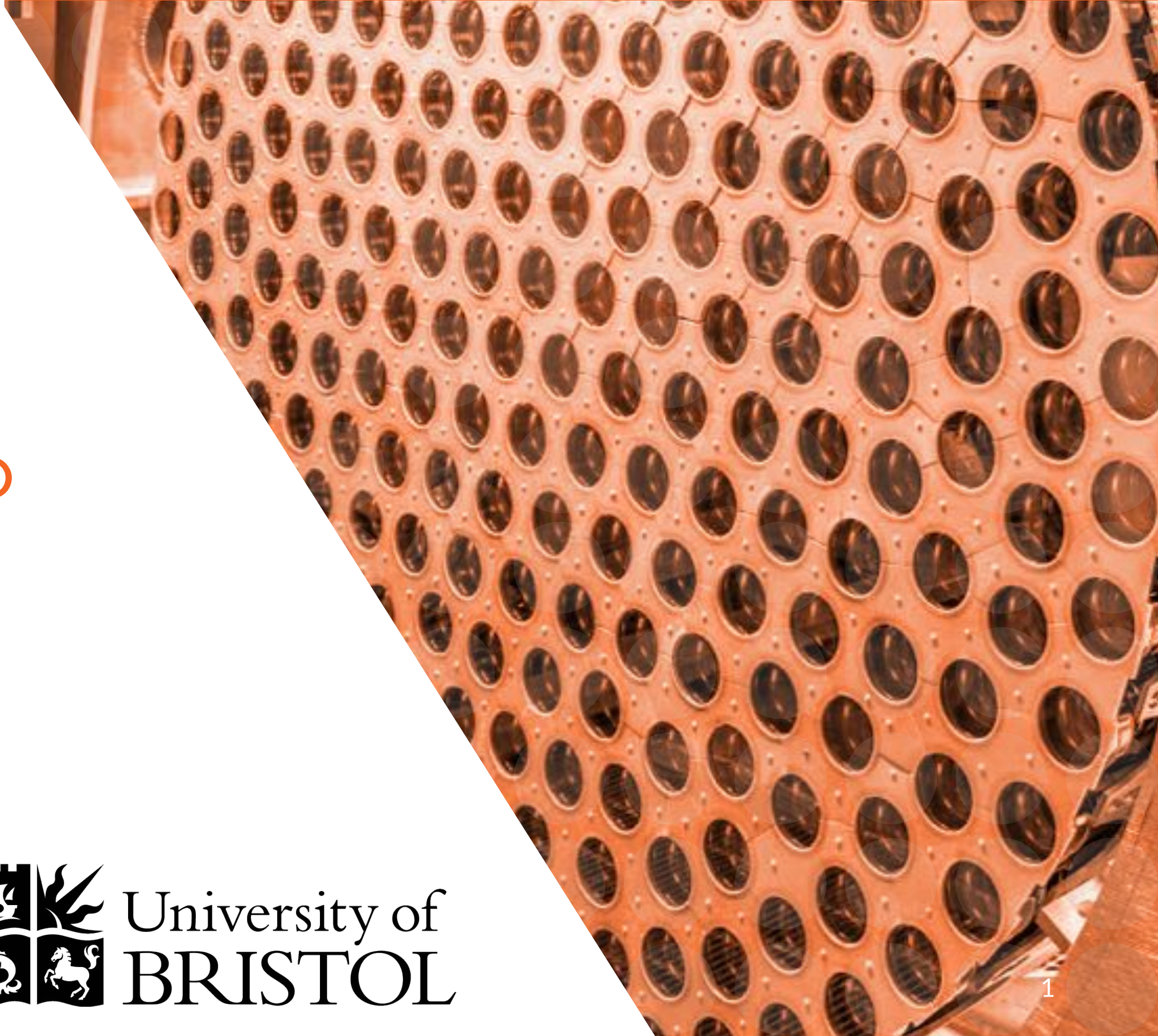
The FAST-HEP toolkit

Ben Krikler, PyHEP 2019

17th October 2019



Goals of this talk

Give you a sense of:

1. the big picture
2. how these tools work
3. where we want to go
4. how this fits in to the rest of the ecosystem

The High-level Overview

Or: Repeating some themes we've already heard

F.A.S.T = Faster Analysis Software Taskforce

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

Properties of an Ideal FAST Analysis: FLAMERSP

1. **Flexibility**: It should be very easy change parts of an analysis, e.g. selection, input data (incl. structure), and to prototype new ideas
2. **Learnability**: A new user should be able to produce meaningful results, e.g. new plots, within a week
3. **Automation**: Use Continuous Integration tools to automate the validation of the analysis, publication of documentation, and performance monitoring
4. **Modularity:** If a new package becomes available, improving the functionality or performance of some part of the analysis, it should be relatively easy to replace the current version with the new package.
5. **Expressiveness**: An analyst should be asking "what do I want to study" and not "how do I implement this"
6. **Reproducibility**: Once an analysis has been run, it should be easy to repeat this, and therefore easy to document what was done
7. **Summarizing**: quick and easy production of plots & tables to inspect data
8. **Performance:** Analysis code should run quickly, processing events at MHz rates


http://www.moviequotesdb.com
do not try & bend the spoon.
that's impossible. Instead...
only try to realize the truth.
Then you'll see, that it is not
the spoon that bends, it is
ONLY YOURSELF!

This is backwards for us:

Physicists have bent ourselves to think in ways that the code dictates: “I want to see this, how must I write that…”

Instead: How can we make the spoon itself bend for us?

Or in other words….

Less: “How do I have to write this”

More: “What do I want to see”

Declarative programming

- We’re most familiar with imperative programming:
  - “Loop over each event, add this to that if something is true, etc”
- Declarative languages the user says WHAT, the interpretation decides the HOW
  - Wikipedia: “Declarative programming [...] expresses the logic of a computation without describing its control flow.”
- Allows:
  - Optimisation behind the scenes
  - More mathematical description of the analysis
  - More concise definition
  - Fewer bugs
  - Easier to reproduce and share

Describing analysis with YAML

- 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

JSON


```
[{"martin":{"name": "Martin Devloper",
    "job": "Developer",`
    "Skills": ["python", "perl", "pascal"]}
,{"tabitha":{"name": "Tabitha Bitumen", "job":
"Developer", "Skills": ["lisp", "fortran",
"erlang"]}}]
```


YAML


```
-  martin:
    name: Martin Devloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
-  tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

“But this is PyHEP and you’re talking about YAML...”

- I want to write and “own” the least amount of code possible:
  - less maintenance
  - more sharing
- All backend code fully python-based

Currently:

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

uproot

Awkward Array

NumPy

numexpr

at

Near future:

Parsl

Further future (?):

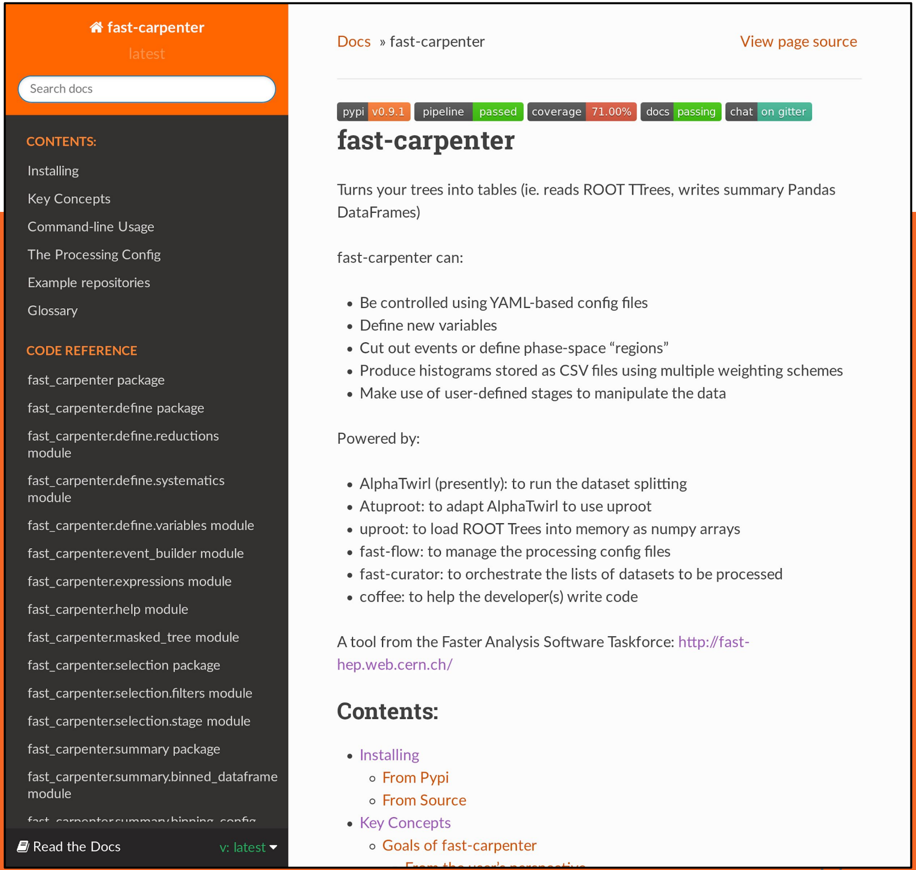
APACHE Spark™

PyHEADTAIL

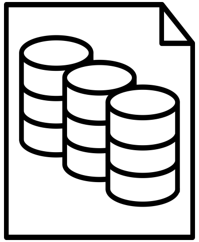
How do you use it?

Where to find the code

- Docs: fast-carpenter.readthedocs.io/
- All public on github:
    - github.com/fast-hep/
    - Main package: github.com/fast-hep/fast-carpenter
- On PyPI, e.g. fast-carpenter
- Docker image with all tools: fasthep/fast-hep-docker
- Clonable demo analysis repository:
    - gitlab.cern.ch/fast-hep/public/fast_cms_public_tutorial
    - github.com/fast-hep/fast_cms_public_tutorial (prelim.)

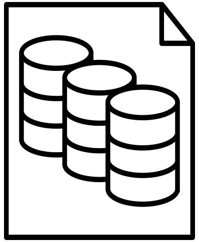


Step 1:
`fast_curator`



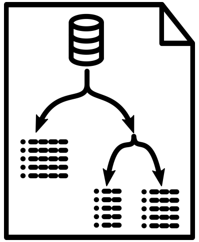
Dataset
description

Step 1:
`fast_curator`



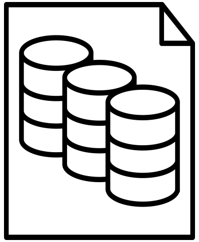
Dataset description

Step 2:
`fast_carpenter`



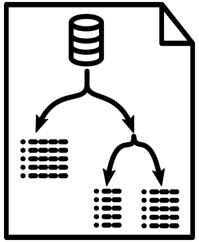
Analysis description

Step 1:
`fast_curator`



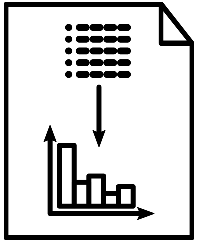
Dataset description

Step 2:
`fast_carpenter`



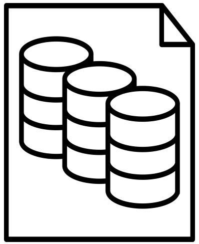
Analysis description

Step 3:
`fast_plotter`
`fast_datacard`



Plotting and postprocessing

Step 1:
fast_curator



Dataset description

Start with a root tree

- Ah, but I have many
- Ah but I need meta-data:
- E.g. cross-section, integrated exposure, calibration source

Curator: adiabatic from 1 to many files

Dataset descriptions don't change often

- Track descriptions in repo, easy to review

Command line tool to help write YAML

- Wild-card on the command line, including xrootd files (contributed to pyxrootd)
- Hooks in place for experiment-specific catalogues, e.g. CMS DAS
- Integrate with Rucio

Regardless of other FAST-HEP tools, generally useful for analysis

Dataset description

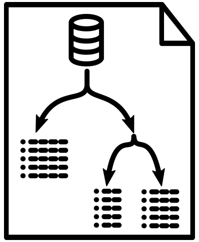
```yaml
defaults:
  eventtype: mc
  nfiles: 1
  tree: events
datasets:
  - eventtype: data
    Files: [input_files/HEPTutorial/files/data.root]
    name: data
    nevents: 469384
  - files:
      - input_files/HEPTutorial/files/dy.root
      - input_files/HEPTutorial/files/dy_2.root
    name: dy
    nevents: 77729
  - files: [input_files/HEPTutorial/files/qcd.root]
    name: qcd
    nevents: 142
import:
  - "{this_dir}/WW.yml"
  - "{this_dir}/WZ.yml"
```

- Default values for all datasets
- Meta data: tree name(s), data or MC

- Each dataset has a list of files
- A unique name

- Can Import other dataset files
- Build complex nested dataset descriptions

Step 2:
`fast_carpenter`



Analysis description

Take yous trees and make them into tables

- Just like a carpenter

Table = Pandas DataFrame

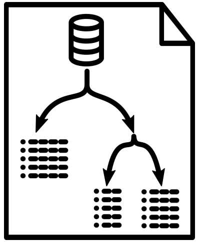
Two main types of table for now:

- Histogram
- Cutflow

Cover most typical particle physics analyses

- BUT: very easy to break out to imperative python when needed

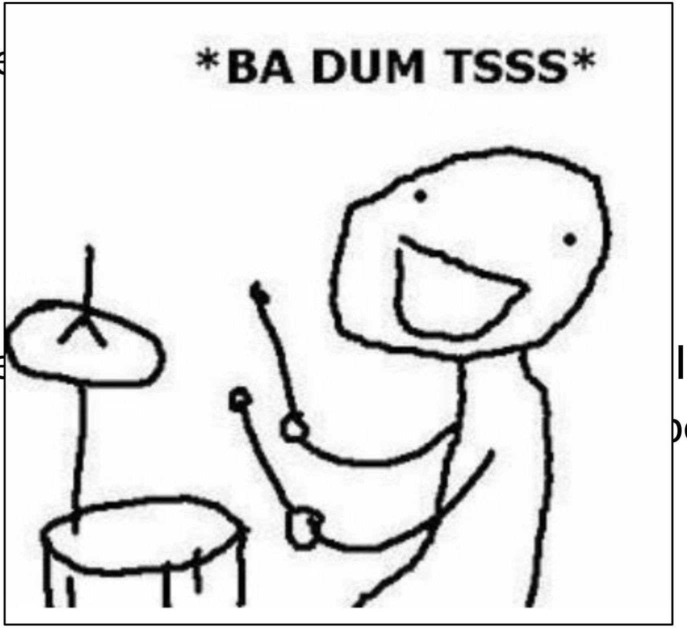
Step 2:
`fast_carpenter`



Analysis description

Take yous trees and make them into tables

- Just like a carpenter



Table

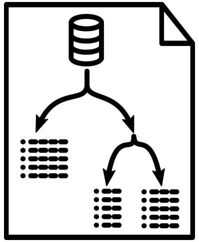
Two

-
-

Cove ... lyses

- ... perative

Step 2:
`fast_carpenter`



Analysis
description

Take yous trees and make them into tables

- Just like a carpenter

Table = Pandas DataFrame

Two main types of table for now:

- Histogram
- Cutflow

Cover most typical particle physics analyses

- BUT: very easy to break out to imperative python when needed

Describe what to do with the data

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: BinnedDataframe
```

The sequence of stages wanted

Each stage:

- Any python importable class
- Duck-typed interface
- Default stages from fast-carpenter

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
      - NIsoMuon:
          formula: IsoMuon_Idx
          reduce: count_nonzero
      - IsoMuPtSum:
          formula: Muon_Pt
          reduce: sum
          mask: IsoMuon_Idx
      - HasTwoMuons: NIsoMuon >= 2
      - Muon_lead_Pt: {reduce: 0, formula: Muon_Pt}
      - Muon_sublead_Pt: {reduce: 1, formula: Muon_Pt}
```

Mathematical description of operations

Operates on arrays of data

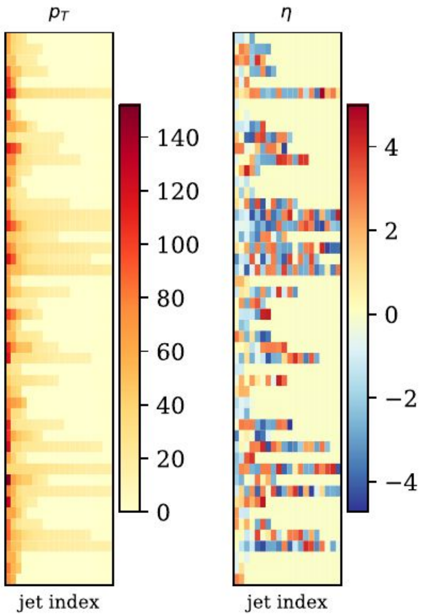
- Uses uproot + numexpr (v2)
- Reductions: go from object-level variables (jagged arrays) to event-level
- Masks: remove objects failing some condition

Support for jaggedness as much as uproot / awkward

- E.g. reducing a 3D jagged array → 2D jagged array, same formula

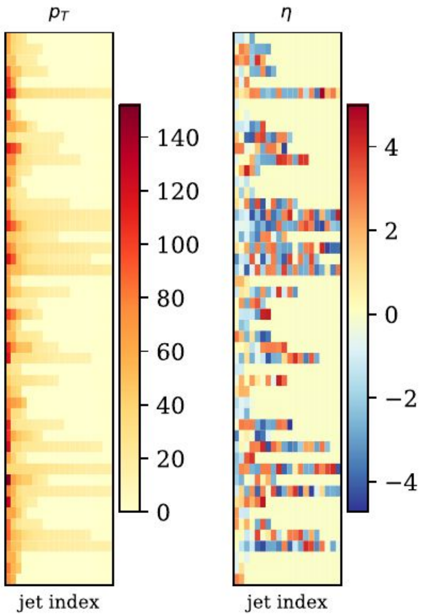
Biggest gap: operations between collections

Define Stage:
fast_carpenter.Define



From Joosep Pata's talk yesterday

Define Stage:
fast_carpenter.Define

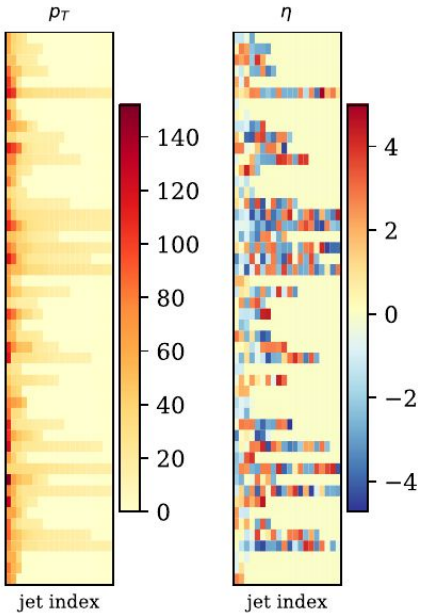


From Joosep Pata's talk yesterday

```yaml
- Muon_Pt: "sqrt(Muon_Px ** 2 + Muon_Py ** 2)"
- IsoMuon_Idx: (Muon_Iso / Muon_Pt) < 0.10
- HasTwoMuons: NIsoMuon >= 2
```

- Simple operations
- Preserve the "jaggedness"

Define Stage:
fast_carpenter.Define



From Joosep Pata's talk yesterday

```
- Muon_Pt: "sqrt(Muon_Px ** 2 + Muon_Py ** 2)"
- IsoMuon_Idx: (Muon_Iso / Muon_Pt) < 0.10
- HasTwoMuons: NIsoMuon >= 2
```

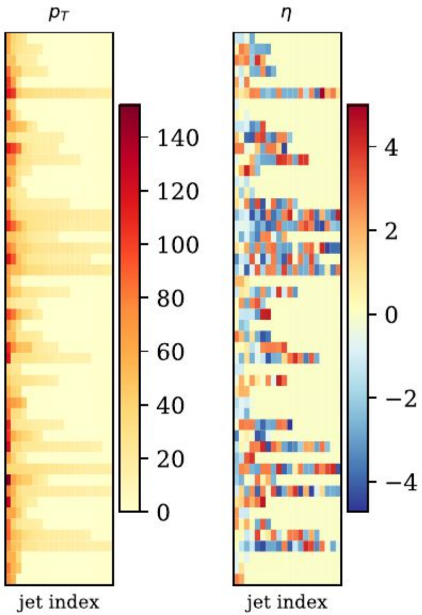
- Simple operations
- Preserve the "jaggedness"

Take the Nth object
(on the deepest dimension)

```
- Muon_lead_Pt: {reduce: 0, formula: Muon_Pt}
- Muon_sublead_Pt: {reduce: 1, formula: Muon_Pt}
```

Define Stage:

fast_carpenter.Define



From Joosep Pata's talk yesterday

```yaml
- Muon_Pt: "sqrt(Muon_Px ** 2 + Muon_Py ** 2)"
- IsoMuon_Idx: (Muon_Iso / Muon_Pt) < 0.10
- HasTwoMuons: NIsoMuon >= 2
```

- Simple operations
- Preserve the "jaggedness"

Take the Nth object
(on the deepest dimension)

```yaml
- Muon_lead_Pt: {reduce: 0, formula: Muon_Pt}
- Muon_sublead_Pt: {reduce: 1, formula: Muon_Pt}
```

```yaml
- NIsoMuon:
    formula: IsoMuon_Idx
    reduce: count_nonzero

- IsoMuPtSum:
    formula: Muon_Pt
    reduce: sum
    mask: IsoMuon_Idx
```

- Reduce dimensionality with a function
- Mask out objects in the event

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
```

Remove events from subsequent stages

Produces a cut-flow summary table

- Weighted / raw counts

Selection is specified as a nested dictionary of **All**, **Any** and a list of expressions

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:
  binning:
      - {in: NMuon, out: nMuons}
      - {in: NIsoMuon, out: nIsoMuons}
  weights: [EventWeight, EventWeight_NLO_up]

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

- Binning scheme:
  - Assume variable already discrete (eg. NumberHits)
  - Equal-width bins over a range (eg. DiMuonMass)
  - List of bin edges
- Event weights
  - Multiple weight schemes add columns
- Output written to disk:
  - Pandas to produce a dataframe in any format
  - Also (experimentally) to a Ghast

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
```

All built-in stages

- Full list of stages can be found with:
  `$ fast_carpenter --help-stages`
- Can get full help for specific stage e.g.:
  `$ fast_carpenter --help-stages-full CutFlow`

- **Define**: Create new variables
- **SystematicWeights**: Create event weights with systematic variations from multiple sources
- **CutFlow**: Remove events failing cuts and summarize # of events passing each cut
- **SelectPhaseSpace**: Like CutFlow but creates mask without applying it
- **BinnedDataframe**: Creates a binned pandas dataframe that can be fed into fast-plotter
- **BuildAghast**: Like BinnedDataframe but result is a Ghast

User-defined stages

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

...

DiMuons:
    mask: IsoMuon_Idx
```

- 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

User-defined stages

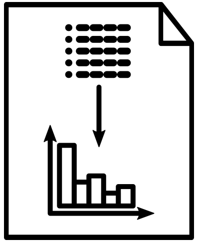
Parameters controlled from analysis description

```
from uproot_methods import TLorentzVectorArray
import numpy as np


class DiObjectMass():
    def __init__(self, name, out_dir, collection="Muon", mask=None, out_var=None):
        self.name = name
        self.out_dir = out_dir
        self.mask = mask
        self.collection = collection

        self.branches = [self.collection + "_" + var for var in ["Px", "Py", "Pz", "E"]]
        if out_var:
            self.out_var = out_var
        else:
            self.out_var = "Di{}_Mass".format(collection)
```

User-defined stages

```python
def event(self, chunk):
    # Get the data as a pandas dataframe
    px, py, pz, energy = chunk.tree.arrays(self.branches, outputtype=tuple)

    # Rename the branches so they're easier to work with here
    if self.mask:
        mask = chunk.tree.array(self.mask)
        px = px[mask]
        py = py[mask]
        pz = pz[mask]
        energy = energy[mask]

    # Find the second object in the event (which are sorted by Pt)
    has_two_obj = px.counts > 1

    # Calculate the invariant mass
    p4_0 = TLorentzVectorArray(px[has_two_obj, 0], py[has_two_obj, 0],
                               pz[has_two_obj, 0], energy[has_two_obj, 0])
    p4_1 = TLorentzVectorArray(px[has_two_obj, 1], py[has_two_obj, 1],
                               pz[has_two_obj, 1], energy[has_two_obj, 1])
    di_object = p4_0 + p4_1

    # insert nans for events that have fewer than 2 objects
    masses = np.full(len(chunk.tree), np.nan)
    masses[has_two_obj] = di_object.mass

    # Add this variable to the tree
    chunk.tree.new_variable(self.out_var, masses)
    return True
```

Step 3:
`fast_plotter`
`fast_datacard`



Plotting and postprocessing

fast-plotter:

- Easy to produce basic plots, tools to support final publication-quality
- Command-line tool with reasonable defaults and simple configuration
- Written in lots of small functions: can be used in custom scripts / notebooks

fast-datacard:

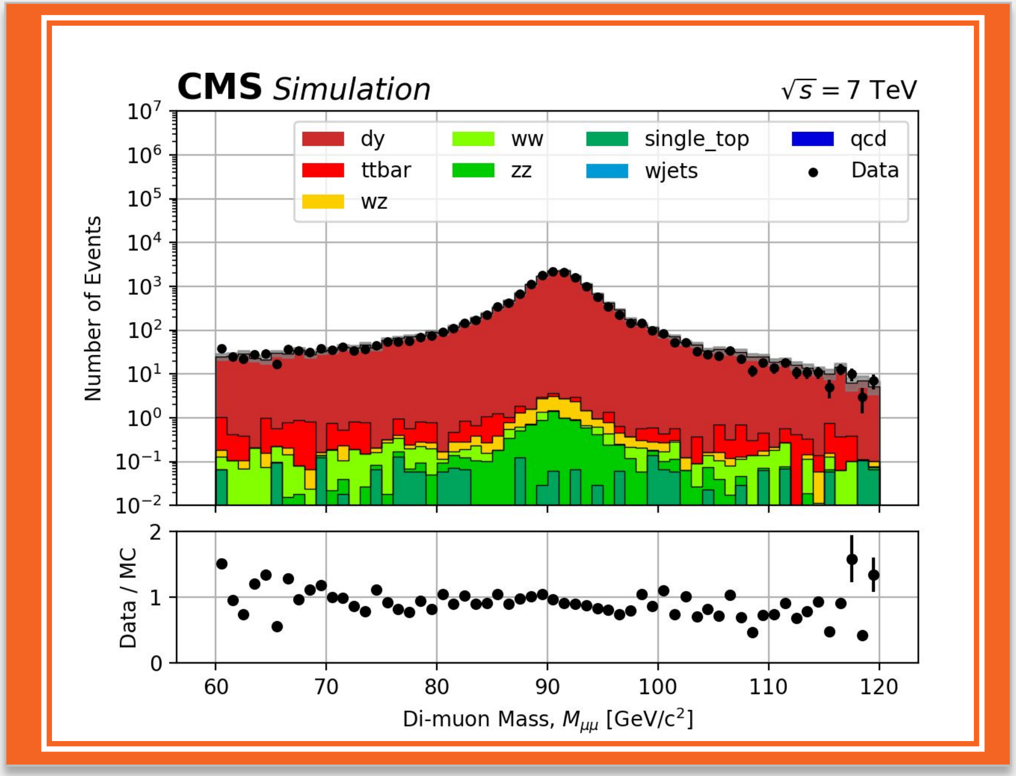
- Bring resulting DataFrames into CMS' Combine fitting procedures

Turning outputs into plots: fast-plotter

- Plot on the right with:

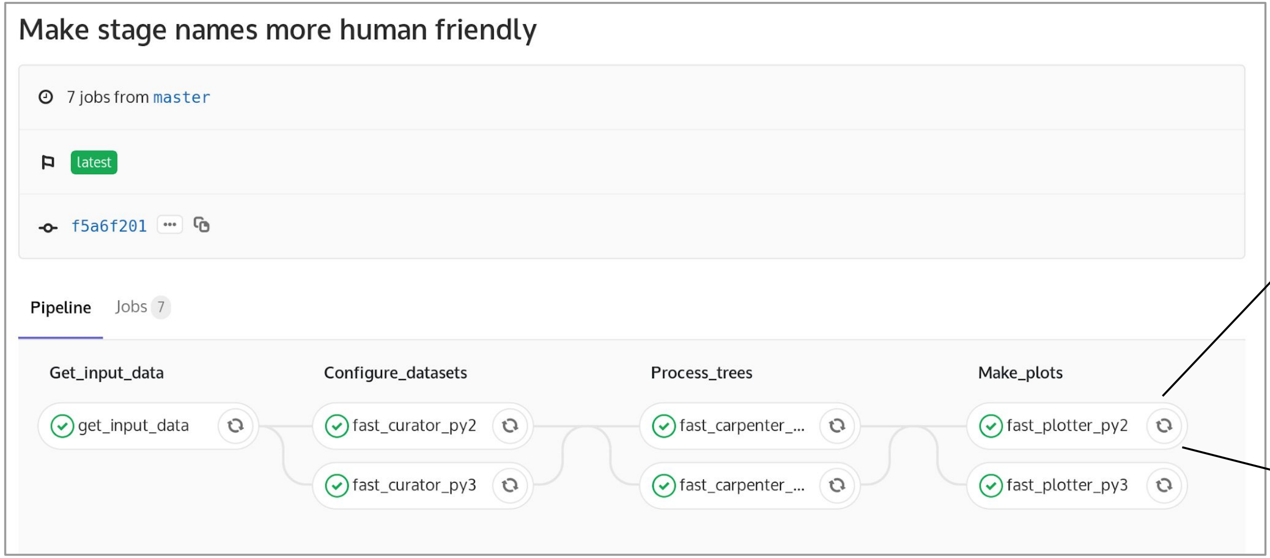
```
fast_plotter -y log \
-c plot_config.yml \
-o tbl_*.csv
```

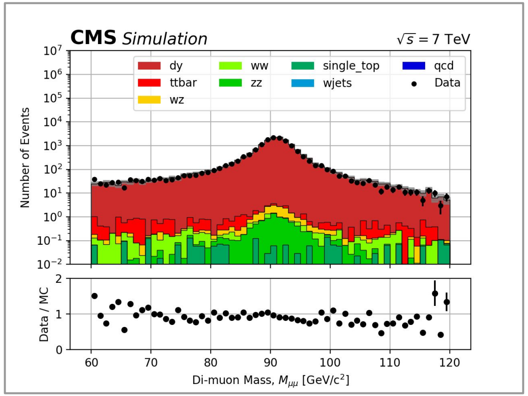
- YAML config:
  - Colour scheme, axis labels
  - Dataset definition
  - Annotation

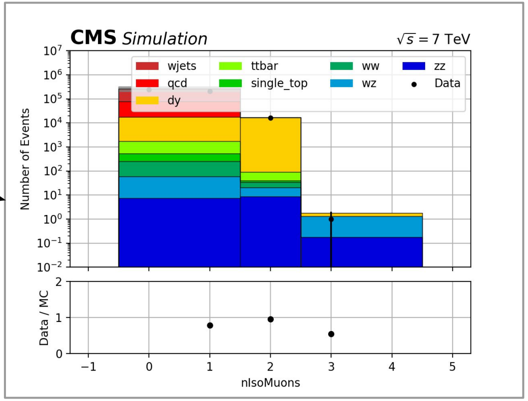


Plot of DiMuonMass binned dataframe from last slide

"Analysis in a CI pipeline"







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

Where are we and what's next

Current FAST-HEP codebase

Demonstrate the previous principles

- A Minimal Viable Product where we're continually adding features
- Hope to cover most analyses using just YAML
- Easy to add user features when FAST-HEP doesn't include

Developed largely by myself, Luke Kreczko, and others

- Contributions growing from various activities

Being used for **2 CMS analyses**, **LUX-ZEPLIN** getting going, design studies for **DUNE, FCC** experiments

- New features being fed back to core packages from analysis-specific repositories

Just how “fast” is this?

In general: as quick as a C++ equivalent

For example, the demo repo:

- Fast-carpenter: 6 seconds
- C++ example: 4 seconds

Much optimisation possible under the hood

At this level, the main advantage not the speed of execution:

- Readability, reproducibility, portability
- From demo repo: 100 lines of YAML vs > 600 of C++

Major changes

Next milestone: PARSL backend

- Experimental version: https://gist.github.com/benkrikler/dc1d2b1fa291b8250a6a07be2b7fc7fa
- Expect first integrated version in next few weeks
- Many benefits anticipated:
  - More control over job splitting and merging
  - Caching
  - DAG monitoring
  - More parallel processing options

Version 1.0: Generalised data-space

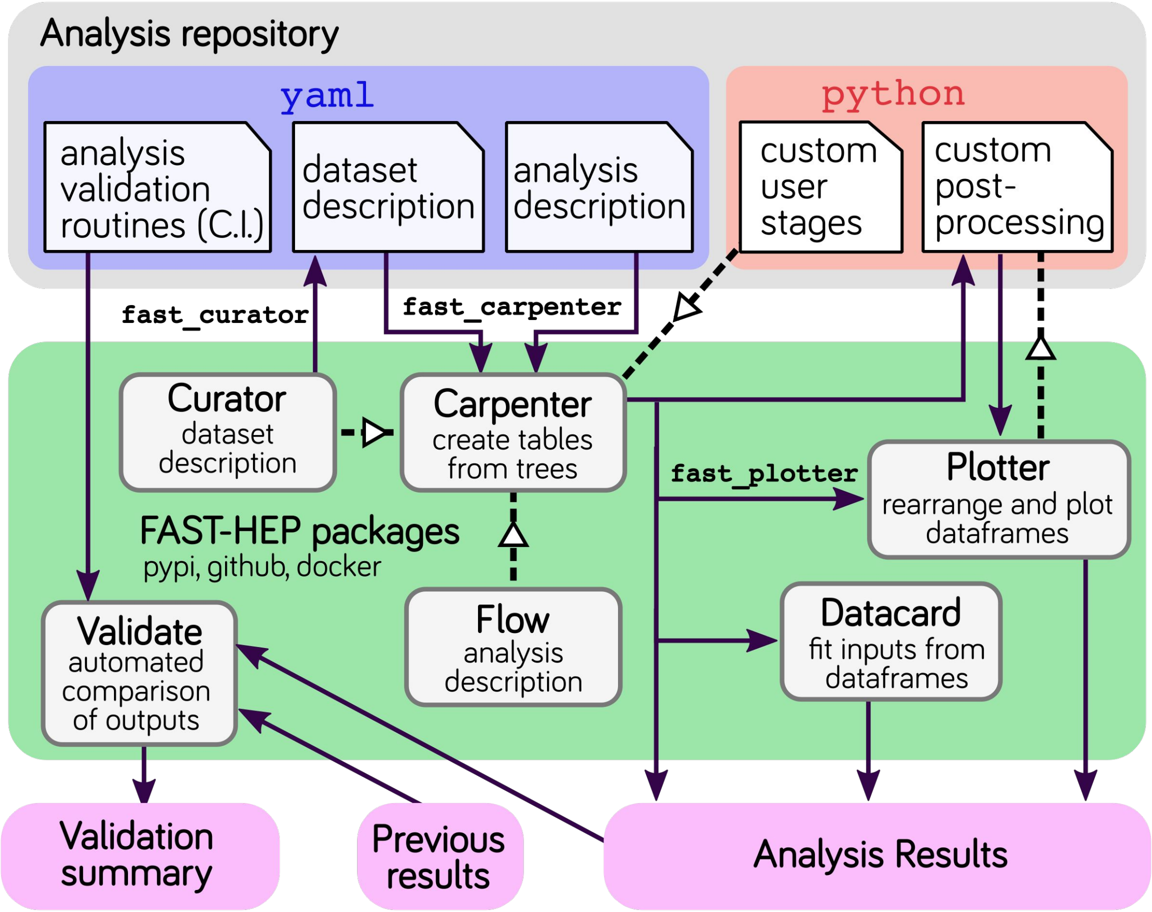
- Not just passing around root tree + other variables
- Pass full dataframes
- Include plotting and fitting in carpenter

Summary

- Have introduced the FAST codebase
  - 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
- As fast as C++ analysis speed
  - Lots of room for optimisations
- Resources
  - Code: github.com/fast-hep/fast-carpenter
  - PiPI: pypi.org/project/fast-carpenter/
  - Docs: fast-carpenter.readthedocs.io/
  - Gitter: gitter.im/FAST-HEP/community

Thank You

Interplay in a typical user's analysis repo



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

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