My analysis experience in LHCb

MANCHESTER 1824
The University of Manchester

Chris Burr

23rd January 2019 - Analysis Requirements Jamboree

Image: CERN-EX-66954B © 1998-2018 CERN

Who am I?

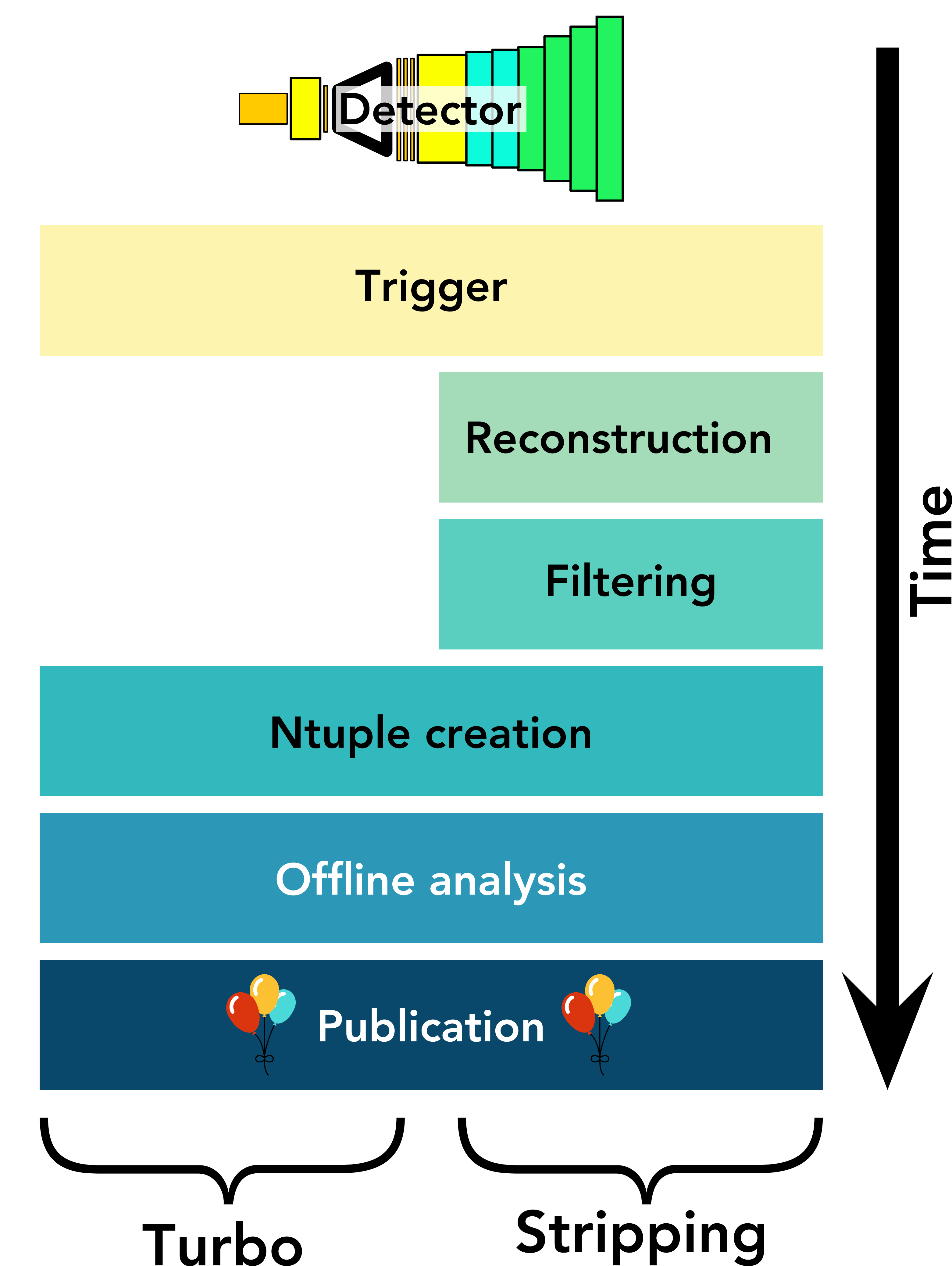
- Final year PhD student and due to submit "soon"
- "Primarily" working on analysis and detector alignment in LHCb
- Heavily involved in LHCb's Starterkit activities
  - Young people teaching master's and first year PhD students
  - Hoped that students become helpers and teachers the following year
- Generally interested in computing and analysis preservation
- This is mostly from memory so I might have forgotten details

Some things I've worked on

- My full analyses:
  - Measuring charm cross-sections in 13 TeV pp collisions — LHCB-PAPER-2015-041
  - Measuring charm cross-sections in 5 TeV pp collisions — LHCB-PAPER-2016-042
  - Search for $D^+_{(s)} \to h^\pm l^+ l'^\mp$ — Currently in internal review
- Other work:
  - Feasibility study for D0 2 phi gamma
  - Alignment studies for the LHCb Upgrade Vertex Locator
  - Alignment support for test beams
  - Optimisation of the energy test — JINST 13 (2018) no.04, P04011

Data flow

- Currently two main ways to get data from LHCb
  - Most analyses use a constant number of particles (TTrees are flat)
- Stripping:
  - Filter data using hardware trigger then software trigger
  - Run offline reconstruction
  - Filter data in centrally ran "stripping campaigns"
  - Analysts make make TTrees containing information about candidates
- Turbo stream: (LHC Run 2 onwards)
  - Offline reconstruction optimised to be fast enough for the trigger
  - Use trigger reconstruction
  - Analysts make make TTrees containing information about candidates
- I will only talk about the "offline analysis" step
  - I've done analyses using both, but there is little difference in practice



Charm cross-sections

- Measure charm cross-sections in 13 TeV pp collisions
  - Make results for four species: D0, D*+, D+, and Ds+
  - Measure in bins of the kinematics (p_T and η)
  - Around 90 measurements for each meson (some bins are skipped due to missing entries)
  - Combine results to give lots of ratios

LHCB-PAPER-2015-041

- Data was collected during the 50ns ramp using Turbo
  - Analysis was developed using Run 1 data and MC
- Signal yields ranged from 110,000 to 2,600,000
- Paper was submitted ~2 weeks after data taking finished

- 3 students developed the analysis code
  - Almost entirely written in Python
  - Heavily used PyROOT: Loading data, RooFit, plotting
  - Used wrappers for TChain, RooFit, ...
- Code was stored in a private repository on GitHub
  - 1,582 commits
- Single repository, used feature branches and pull requests
  - 177 pull requests
  - Almost always reviewed each others code, many PRs have 10+ comments
  - Incredibly educational having this review
- Used Travis CI to lint the code with flake8
  - Looking at the cross-section code for the first time in 3 years…it's actually quite nice

- Manually ran the analysis on lxplus every night during later stages
- Most code is kept inside a python module
- Executed using: `python run_analysis_framework.py […]`

```
3  for year in args.years:
4      for mode in args.modes:
5          log.info('Running for mode {0}'.format(mode))
6          setup(mode, year, no_stripping=args.no_stripping,
7                no_offline=args.no_offline)
8          train_bdt(mode, year)
9          for polarity in args.polarities:
10             read_bdt(mode, year, polarity)
11             fit(mode, year, polarity)
12             fit_yields(mode, year, polarity)
13             plot_fits(mode, year, polarity)
14             fit_systematic(mode, year, polarity)
15             sweights_fit(mode, year, polarity)
16             pid_efficiency(mode, year, polarity)
17             efficiencies_from_mc(mode, year, polarity)
18             efficiencies_by_cut(mode, year, polarity)
19             tracking_systematic(mode, year, polarity)
20             data_mc_compare(mode, year, polarity)
21             signal_window_efficiency(mode, year, polarity)
22             from submit_jug import RunIt as RunPID
23             RunPID(mode, year, polarity,
24                    'charmproduction/pid/GetToyResult_mcerp.py',
25                    False, [('PART_RPL', '"KPi"')], utilities.cpu_count(), 0)
```

The actual script is the same except **`argparse`** is used and each line is prefixed with if **`args.run_something`**

- 13 TeV code and repository was reused for a 5 TeV measurement
- Using data was collected during a special run at the end of 2015

Search for $D^+_{(s)} \to h^\pm l^+ l'^\mp$.

- Search for all decays of the form $D^+_{(s)} \to h^\pm l^+ l'^\mp$
  - h is a kaon or pion
  - l is a muon or electron
  - 28 measurements in total across 14 final states **(8 allowed in SM but very rare, 20 forbidden from LFU/LNU)**
  - 4 additional channels used for normalisation
- Expect to set upper limits on the branching fraction for all channels
- All code in the analysis framework has been written by me
- Try to treat everything the same way to reduce the workload
  - Electrons emit bremsstrahlung radiation making the fit shapes very
  - Some channels contain resonances which have to be removed
  - Different backgrounds are present in different decays

Analysis without ROOT

- Stopped using ROOT except for: `pandas.save_hdf(root_pandas.read_root(…))`
  - Now `uproot` can be used instead
- Why?
  - Conda provides an great Python environment but including ROOT was tedious
  - Lack of interoperability with standard Python components like numpy, matplotlib
  - Didn't always interact well **(order of imports suddenly matter, segfaults, JupyROOT crashing Jupyter)**
- I ended up needing to use ROOT, I'll come back to this

NOTE: This has improved since this analysis started

- Almost entirely used Jupyter notebooks
  - Created GitHub Gists, sent to supervisor, used markdown to explain what was going on
- Tried unsuccessfully run notebooks in a pipeline
  - This might have improved in the last ~3 years
- Now I develop code using Jupyter or IPython then copy it to a Python script
  - Use argparse to make it configurable

```python
import argparse

def run_something(channel, year, input_fn, output_fn):
    pass

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--channel', choices=config.channels, required=True)
    parser.add_argument('--year', choices=config.years, required=True)
    parser.add_argument('--input_fn', required=True)
    parser.add_argument('--output_fn', required=True)
    args = parser.parse_args()
    run_something(args.channel, args.year, args.input_fn, args.output_fn)

if __name__ == '__main__':
    parse_args()
```

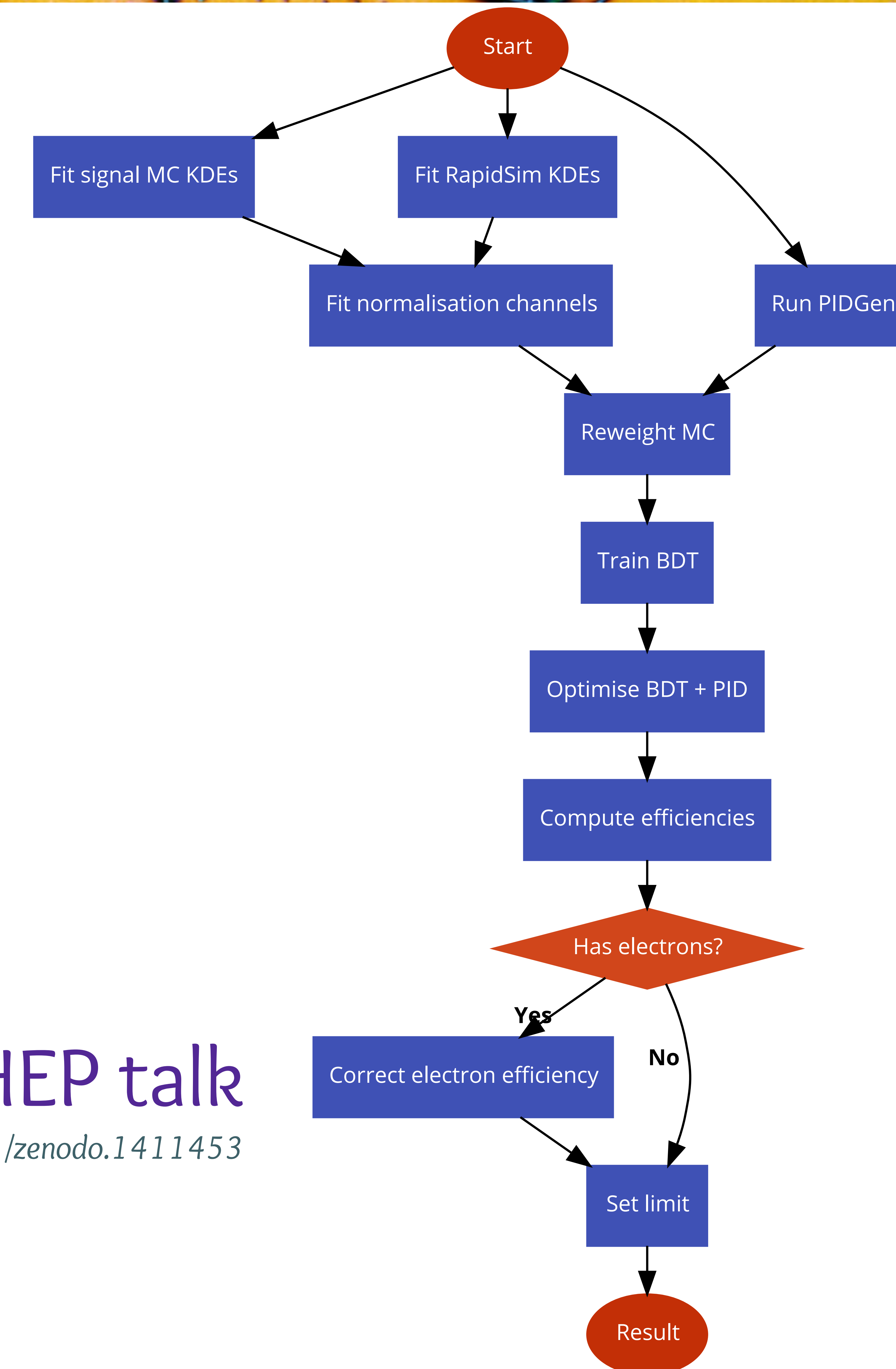
- Use Snakemake to write pipelines using Python 3 (+syntactic sugar)
  - Developed for bioinformatics, cited by a large number of publications
  - Integrates with: conda, singularity, cluster/batch systems, XRootD, GridFTP
  - Rapidly growing user base in LHCb but the initial learning curve is quite steep
- Input data is on the grid (~1,800 files and ~5TB)
  - Using XRootD to access data at CERN from my institute is slow
  - Prone to random failures causing errors, or even segfaults within XRootD itself
  - Can't use the fallback mechanisms to use other sites instead
  - I've seen other people have issues with firewalls blocking XRootD
- Apply preprocessing and download 492 ROOT files (~40GB)
  - Loose cuts and avoid unneeded variables (makes everything so much faster)
- Almost every step is single threaded
  - Snakemake handles running many steps in parallel

What does the pipeline now do

- Overview of stages
  - Generate toy MC using RapidSim
  - Calibrate MC using PIDGen **(internal LHCb tool)**
  - Perform maximum likelihood fits using RooFit
  - Compute sWeights using hep_ml
  - Reweight MC with a BDT using hep_ml
  - Train a BDT using the scikit-learn interface of XGBoost
  - Use CLs to compute a limit with RooStats

- More details about tools I use can be found in my PyHEP talk

https://doi.org/10.5281/zenodo.1411453



Storing output

- Similar system to that used for the cross section analyses
- Output is stored in `cloned_repository/output`
- Analysis note uses this directory so plots and values are always up to date
  - `ln -s ~/analysis-code-repository/output ~/analysis-note–repository/output`

```
output
    D0ToKpi
        2011
            DVntuple_bkg.root           _ntuple of real sideband events for BDT training_
            DVntuple_sig.root           _ntuple of MC signal events for BDT training_
            MagDown
                DVntuple_Cheat.root         _ntuple of MagDown MC cheated events_
                DVntuple_Gen.root           _ntuple of MagDown MC generator level events_
                DVntuple_MC.root            _ntuple of MagDown MC signal events_
                DVntuple_Real.root          _ntuple of real MagDown events_
            MagUp
                DVntuple_Cheat.root         _ntuple of MagUp MC cheated events_
                DVntuple_Gen.root           _ntuple of MagUp MC generator level events_
                DVntuple_MC.root            _ntuple of MagUp MC signal events_
                DVntuple_Real.root          _ntuple of real MagUp events_
```

What does the pipeline now do

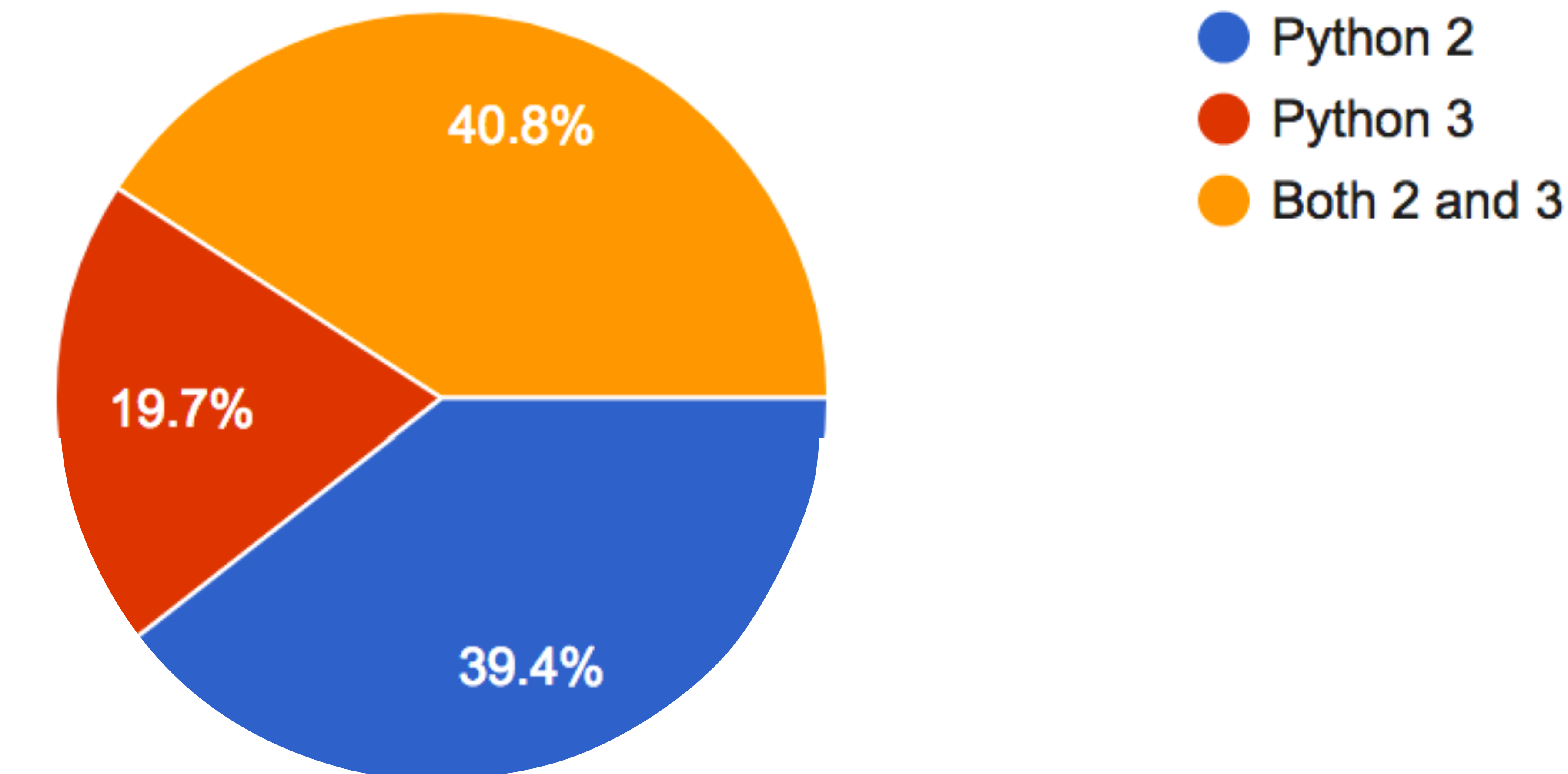
- Now contains over 11,000 steps
  - Takes ~36 hours on a 16 core machine (excluding initial data download)
- When finishing the analysis and computing systematics
  - Could easily rerun everything when issues were found
  - Rerun large portions of the analysis with data stored to `output/systematics/alternative{1..4}/...`

```
2   Count   Job name
3   8624    limit_run_partial
4    428    download_file
5    312    limit_merge_results
6    272    add_mc_weight_column
7    166    generate_rapidsim_sample
8    156    latex_limit_table
9    128    truth_match
10   128    run_pidgen
11   128    apply_triggers_to_mc
12   116    compute_efficiencies_v3
13   110    prepare_model_for_limit
14    72    blind_data
15    64    prefit_signal
16    64    download_mc_without_DaughtersInLHCb_cut
17    64    check_trigger_efficiencies
18    56    plot_signal_prefit
19    56    background_channels
20    56    add_classifier_columns_mc_for_systematics
21    54    add_classifier_columns_mc
22    36    real_data
23    36    optimise_selection_plot
24    36    optimise_selection_grid
25    28    train_classifier
26    28    prefit_background_rapidsim
27    28    prefit_background
28    28    plot_classifier_inputs_comparision
29    28    plot_background_prefit
30    28    make_classifier_plots
31    28    add_classifier_columns_backgrounds
32    18    add_classifier_columns_unblinded
33    10    make_summary_limits_plot_no_systematics
34    10    make_summary_limits_plot
```

```
2   Count   Job name
3      8    train_mc_reweighter
4      8    plot_normalisation_fit
5      8    plot_data_mc_differences_v2_one_norm
6      8    norm_channels
7      8    fit_normalisation_channels
8      4    prefit_background_rapidsim_kdes
9      4    plot_data_mc_differences_v2_all_norms
10     2    make_systematic_summary_tables
11     2    make_summary_limits_plot_with_alternatives
12     2    make_summary_limits_plot_with_all_years
13     2    make_selection_summary_table
14     2    make_estimated_limit_table
15     2    make_efficiency_table
16     2    make_classifier_training_yields_table
17     2    fit_dimuon_norm_with_template
18     2    download_generator_stats_fake
19     2    download_generator_stats
20     2    compute_signal_fit_shape_systematic
21     2    compute_data_mc_systematic
22     2    calculate_electron_corrections
23     1    plot_loose_pid_cut_effect
24     1    mc_mass_plots
25     1    make_electron_corrections_table
26     1    make_crosscheck_table
27     1    compute_norm_systematics
28     1    compare_ntracks
29     1    calculate_tracking_corrections
```

Frustrations and dreams

Python 3



- Snakemake is Python 3 only
  - $D^+_{(s)} \to h^\pm l^+ l'^\mp$ is also using Python 3 only features **(f-strings)**
- Using Python 3 within LHCb quite painful
  - ROOT is missing from conda **(well it was until last week)**
  - Using an LCG view
    - Causes weird issues, especially once they are nested
    - Pip and virtualenv don't work well
  - Replacing `#!/usr/bin/env python` with `#!/usr/bin/env python2` when needed can really help
- Despite this, Python 3 is now widely used in LHCb

Setting up from environments

- Compiling from source is unreliable and slow
- Ideally `something install awesome_package` should just work for anything
- Different stages can have conflicting dependencies
  - Need to be able to manage multiple environments
  - Switching should be easy not create conflicts
- Should be able to share or preserve an environment

See the HSF Packaging WG

- Eventually RooFit and RooStats became necessary for $D_{(s)}^{+} \to h^{\pm} l^{+} l'^{\mp}$
  - Nothing is as mature and flexible
- Even after switching, this was still the most time consuming part
  - I find the API is difficult to use, especially from PyROOT
  - Often hard to see why a fit is failing or what is actually being fitted
- It also doesn't scale for existing datasets
  - And this will only get worse in the upgrade
- It's immature, but I think the idea of zfit is the way forward *https://github.com/zfit/zfit*
  - Build on top of a symbolic math library like tensorflow
  - Lots of features come for "free":
    - CPU, GPU and multi GPU support
    - Underlying graph can be visualised or manipulated
    - Profiling to find why a fit is slow

Efficiency

- It's often easier to make the problem easier than make the tools faster
  - Binned vs unbinned fits
  - Avoiding applying cuts that remove events that are never going to be used
  - Including every possible variable in TTrees
  - Choosing functions that are faster to compute
- Doing both is even better

External dependencies

- ➤ Using the cross-section code as an example ->
- ➤ External dependencies are bad for analysis preservation
- ➤ Also has a handful of data dependencies in user's home areas

Running on lxplus

The `charmproduction` Python module depends on LuaTeX and a few Python modules, and has a couple of C++ components that need building before it can be used. On `lxplus`, the environment can be set up with

```
$ git clone git@github.com:alexpearce/CharmProduction.git   ← Repository moved
$ cd CharmProduction
$ SetupDaVinci   ← No version and LHCb setup scripts have changed
$ wget https://bootstrap.pypa.io/get-pip.py
$ python get-pip.py --user
$ rm get-pip.py
$ export PATH=~/.local/bin/:/afs/cern.ch/sw/XML/texlive/latest/bin/x86_64-linux:$PATH
$ export PYTHONPATH=$PYTHONPATH:~/.local/lib/python2.7/site-packages
$ pip install --user uncertainties   ← No version
$ cd analysis/BDTTools && make && cd -
$ cd analysis/charmproduction/fitting/shape_classes && make && cd -
$ cd analysis
```

The `export` commands needs to be run in every new shell.

To calculate the PID efficiency a customised version of PIDCalib must be present `cmtuser`:

```
                               User area
ln -s /afs/cern.ch/user/d/dmuller/cmtuser/Urania_v2r4 ~/cmtuser
```

To download some data, for example, do

```
$ python run_analysis_framework.py -m D0ToKpi -y 2011 -p MagDown --setup-only
```

Clusters and batch systems

- Takes too long to set up each time
  - Especially problematic as everyone has access to different systems
- Working locally or on a single machine is always more convent
- I often see laptops running scripts for a whole weekend

- Securely authenticating to storage is hard
  - Generate token for GitLab CI that only has access to one directory
  - Currently have to expose full CERN password or have a service account
- Pipelines can result in a lot of files
  - My $D^+_{(s)} \to h^\pm l^+ l'^\mp$ folder has over 80,000 plot, data and log files (~100 GB)
  - My testbeam alignment folder has 1,368,681 (849 GB)
- I end up using local disk storage most of the time

- Everyone has a 128 cores, 1TB ram and 16TB of SSD on their laptop

My dream for an analysis environment

- Everyone has a 128 cores, 1TB ram and 16TB of SSD on their laptop
  - That's not going to happen any time soon…
  - Try and be ***slightly*** more realistic
- I often wish I could request 1 big VM or container
  - Tens of cores and ~1GB of RAM per core
  - Optionally include a GPU
- Mount a ~1TB volume of POSIX-like storage
  - Doesn't need to have shared read/write access
  - Snapshots and cloning would be nice
- Everything is contained and easier to preserve **(Presumably CVMFS and EOS won't last forever)**
- Only resort to batch/cluster/grid resources for very rare cases

Miscellaneous remarks

Archived analyses as documentation

- I frequently refer back to the charm cross-section analysis
- I often wish I could see the code when replicating from analysis notes

Assertions

- When building pipelines it's errors can propagate a long way
- Much easier to debug if sanity checks are constantly being performed
- I tend to do this with assertions in Python
- Choosing some randomly examples from $D_{(s)}^{+} \to h^{\pm}l^{+}l'^{\mp}$ with grep:

```
1  assert '2016' not in split_fn
2  assert 'pi' in channel, channel
3  assert 1e-5 < eff_1 and eff_1 < 1-1e-5, eff_1
4  assert len(_raw_df) > 30, _raw_df
5  assert len(x_match) == 1 and x_match[0]+1 < len(X[0, :]), x_match
6  assert np.abs(1-dp_track_corr[0]) < 0.01, dp_track_corr
7  assert xs[0] - x2 <= 0, (xs[-1], x2, xs[0] - x2)
8  assert y_match_index+2 < len(Y[:, 0])
```

Questions?