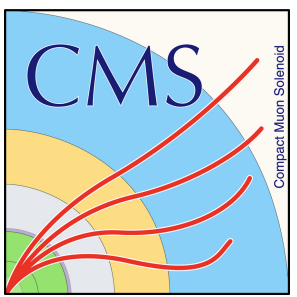
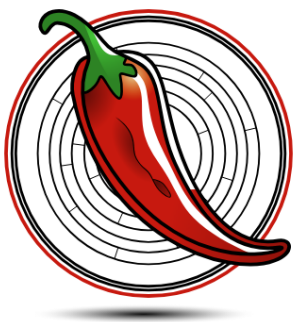
Pepper: a framework for columnar data analysis in CMS

Dominic Stafford, Laurids Jeppe

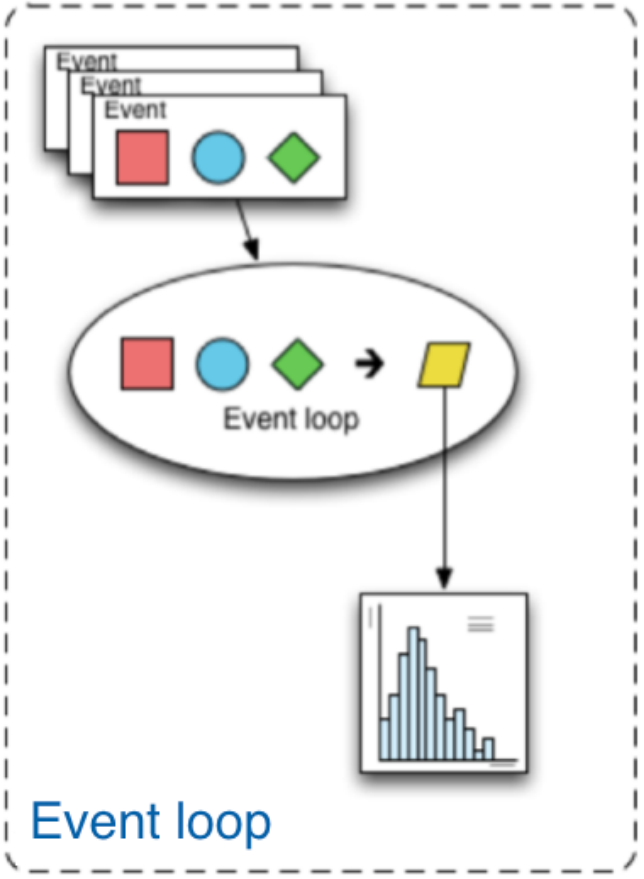


04.07.2025
FH SciComp Workshop



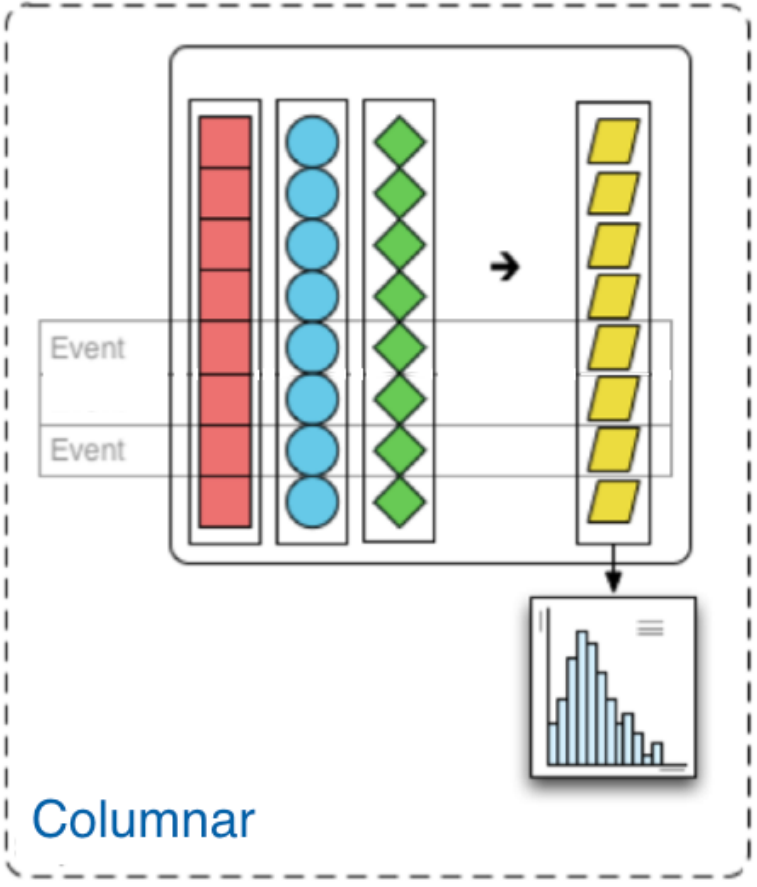
Columnar programming

- HEP analyses consist of many steps
  - Define derived quantities
  - Apply cuts to select events
  - Produce histograms or smaller data skims for further processing
- Traditional analysis frameworks have used event loops
  - Perform all operations on one event, then start on next event
- However for loops in Python are very slow



Columnar programming

- Alternative for Python: columnar processing
  - Perform all operations simultaneously on a chunk of data
- Familiar to users of numpy in python
- Allows for vectorisation of operations
- Allows faster loading of data from ROOT files



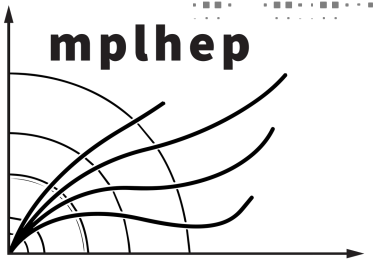
Awkward and Coffea

- HEP events have different numbers of leptons, jets, etc.
  - Not suitable for numpy
- New package Awkward developed to handle these jagged arrays
  - Initially a wrapper around numpy, now dedicated C++ bindings
- Extended by Coffea, which offers a basic analysis framework
  - 4-vector manipulation, scale-out to clusters, CMS object corrections
- Initially developed by CMS members, but designed to be more broad – now explored by all LHC collaborations

Awkward Array

Scikit-hep ecosystem

- Scikit-hep is an ecosystem of tools for HEP analyses, e.g.:
  - ROOT file loading
  - Histogramming
  - Plotting
- Based on scientific python ecosystem, so familiar to most python users
  - Can export to common industry tools
- Dask package is used for scale-out to condor



Pepper - ParticlE Physics ProcEssoR

- Development started in 2019 by Jonas Ruebenach and DS in response to:
  - New lightweight CMS data format “NanoAOD”
  - Scikit-HEP developments/ familiarity with python
- Extends coffea with functionality for full CMS analyses, and scale-out tuned to run on NAF/BIRD
- Developed into a general-purpose framework in response to interest from many DESY groups

Code structure

- Users write their analysis in a Processor class:
  - Inherited from coffea
  - Pre-made implementations of standard CMS cuts and corrections
  - Users can write new cuts/corrections as python functions
- Book-keeping handled by Selector class:
  - Keeps track of cuts, SFs and systematics
  - Calls processor functions which define these
- Automatically fills histograms after each step (configurable)
- Can also produce skims for further processing (e.g. ML)

Main process function

```
selector.add_cut("NoAddLeps",
                 partial(self.no_additional_leptons, is_mc))
selector.set_column("Electron", self.pick_electrons)
selector.set_column("Muon", self.pick_muons)
selector.set_column("Lepton", partial(
    self.build_lepton_column, is_mc, selector.rng))
# Wait with hists filling after channel masks are available
selector.add_cut("AtLeast2Leps", partial(self.lepton_pair, is_mc),
                 no_callback=True)
selector.set_multiple_columns(self.channel_masks)
selector.set_cat("channel", {"is_ee", "is_em", "is_mm"})
selector.set_column("mll", self.mass_lepton_pair)
selector.set_column("dilep_pt", self.dilep_pt, lazy=True)
```

Apply lepton object cuts (config controlled)

Combine into one column

Apply two lepton selection

Split into categories on lepton flavour

Set columns based on dilepton kinematics

Gitlab link

Cut function example

```
def lepton_pair(self, is_mc, data):
    """Select events that contain at least two leptons."""
    accept = np.asarray(ak.num(data["Lepton"]) >= 2)
    if is_mc:
        weight, systematics = self.compute_lepton_sf(data[accept])
        accept = accept.astype(float)
        accept[accept.astype(bool)] *= np.asarray(weight)
        return accept, systematics
    else:
        return accept
```

Basic selection

Compute SFs and systematics for MC

Combine SF weights with selection

Gitlab link

Scale-out

- Can run over files stored either locally (on DCache) or on the LHC computing grid (via xrootd)
- Initially opens files and calculates splitting into chunks (cached)
- Can be run locally (in debug mode) or scaled out via dask:
  - Manager process starts condor jobs, then sends the workers code and indexes of chunks to process via tcp
  - Workers then open appropriate files, and send histograms back via tcp
- Output accumulated on log-in node, and store periodically to allow resuming in case of errors

Code and Installing

- Code stored on CERN gitlab
  - Simple CI for code style and basic running
- Dependencies mostly from lcgenv
- Can then install pepper via pip
- User should then write classes inheriting from pepper classes

Usage

- Currently used by ~12 analyses at DESY, plus 5 publications
  - Largest, but not only, CMS columnar framework on NAF
- Also being used by a few users at other institutes/facilities
- Covers most types of CMS analyses (searches, unfolded measurements, etc.)
- Tutorial at multiple CMS Data Analysis Schools [1], [2], [3]

User experience

- Easy to learn for new users with numpy experience
  - Can start addressing physics problems within a few days
  - Transition for event-loop users can take longer
- Usually flexible with fast time-to-insight
- Large pool of users at DESY allows sharing experience
- Some more complex functions harder to code in a columnar way
- Bottle-necks aren’t always obvious to the user

Performance

- Generally very fast: can typically run a single year of an analysis (without systematics) in < 30 mins
  - Main bottleneck is getting condor workers
- However complex analyses can take notably longer
  - Main bottleneck high memory usage
  - Can be due to either variables or histograms in memory
  - Memory on log-in nodes sometimes an issue
- Initial profiling has already yielded some improvements
  - Scope for more

Issues encountered

- Not very clear to users how much memory workers need, and when jobs exceed memory
  - Considering limiting memory usage in dask
- High memory usage on the log-in node could be avoided by running main process in a condor job
  - But this can then not launch further condor jobs
- IT have reported issues with pepper jobs putting heavy I/O load on afs
  - Unclear what files are being accessed: possibly some dask internal files?

Outlook and Future Plans

- Pepper is a broadly use columnar framework in DESY CMS
- Generally well received by users
- Some issues due to memory usage and file system access encountered
  - Aim to follow up on these with IT
- Plan to do more profiling and work on documentation soon

Backup

Configuration

```
"electron_sf": [
    [
        "$DATADIR/scale_factors/egammaEffi.txt_EGM2D_updatedAll.root",
        "EGamma_SF2D",
        ["eta", "pt"]
    ],
    [
        "$DATADIR/scale_factors/2018_ElectronMVA90.root",
        "EGamma_SF2D",
        ["eta", "pt"]
    ]
],

"ele_cut_transreg": true,
"ele_eta_min": -2.4,
"ele_eta_max": 2.4,
"good_ele_id": "mva:Iso90",
"good_ele_pt_min": 20.0,
"additional_ele_id": "mva:Iso90",
"additional_ele_pt_min": 20.0,
```

Specify SFs from ROOT files

Electron object definition

Gitlab link

Histogramming

```
"leading_electron_pt": {
    "bins": [
        {
            "name": "pt",
            "label": "Electron $p_{\\mathrm{T}}$",
            "n_or_arr": 100,
            "lo": 0,
            "hi": 400,
            "unit": "GeV"
        }
    ],
    "fill": {
        "pt": [
            "Electron",
            "pt",
            {"leading": 1}
        ]
    }
},
```

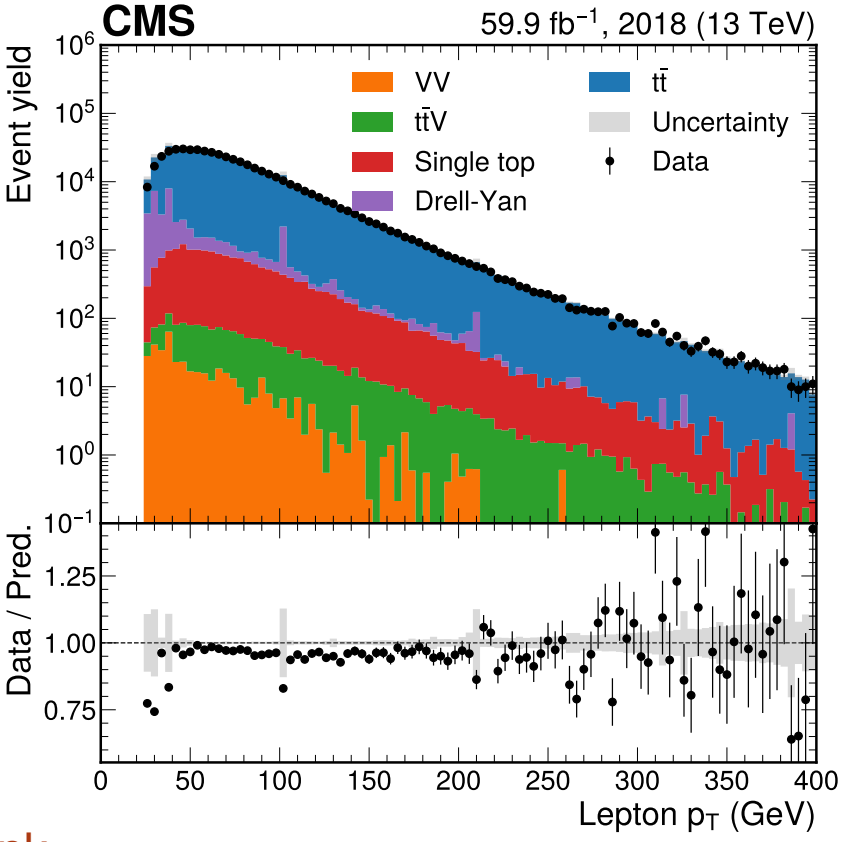
Gitlab link

- By default histograms plotted after each cut
  - Includes selector categories and systematics by default
  - Can restrict cuts, etc. to save memory
- Histograms defined in configuration
  - Helper functions for simple derived quantities, e.g. multiplicities
  - Can also be multi-dimensional
- Can save in root or hist (coffea pickle) format

Plotting

- Configurable plotting script for hist histograms
- Intended as starting point for users

```
"backgrounds": {
    "DY": {
        "label": "Drell-Yan",
        "color": "tab:purple",
        "datasets": [
            "DYJetsToLL_M-10to50_TuneCP5_13TeV-madgraphMLM-pythia8",
            "DYJetsToLL_M-50_TuneCP5_13TeV-madgraphMLM-pythia8"
        ]
    },
    "TT": {
        "label": "$t \\bar{t}$",
        "color": "tab:blue",
        "datasets": [
            "TTTo2L2Nu_TuneCP5_13TeV-powheg-pythia8",
            "TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8"
```



Gitlab link