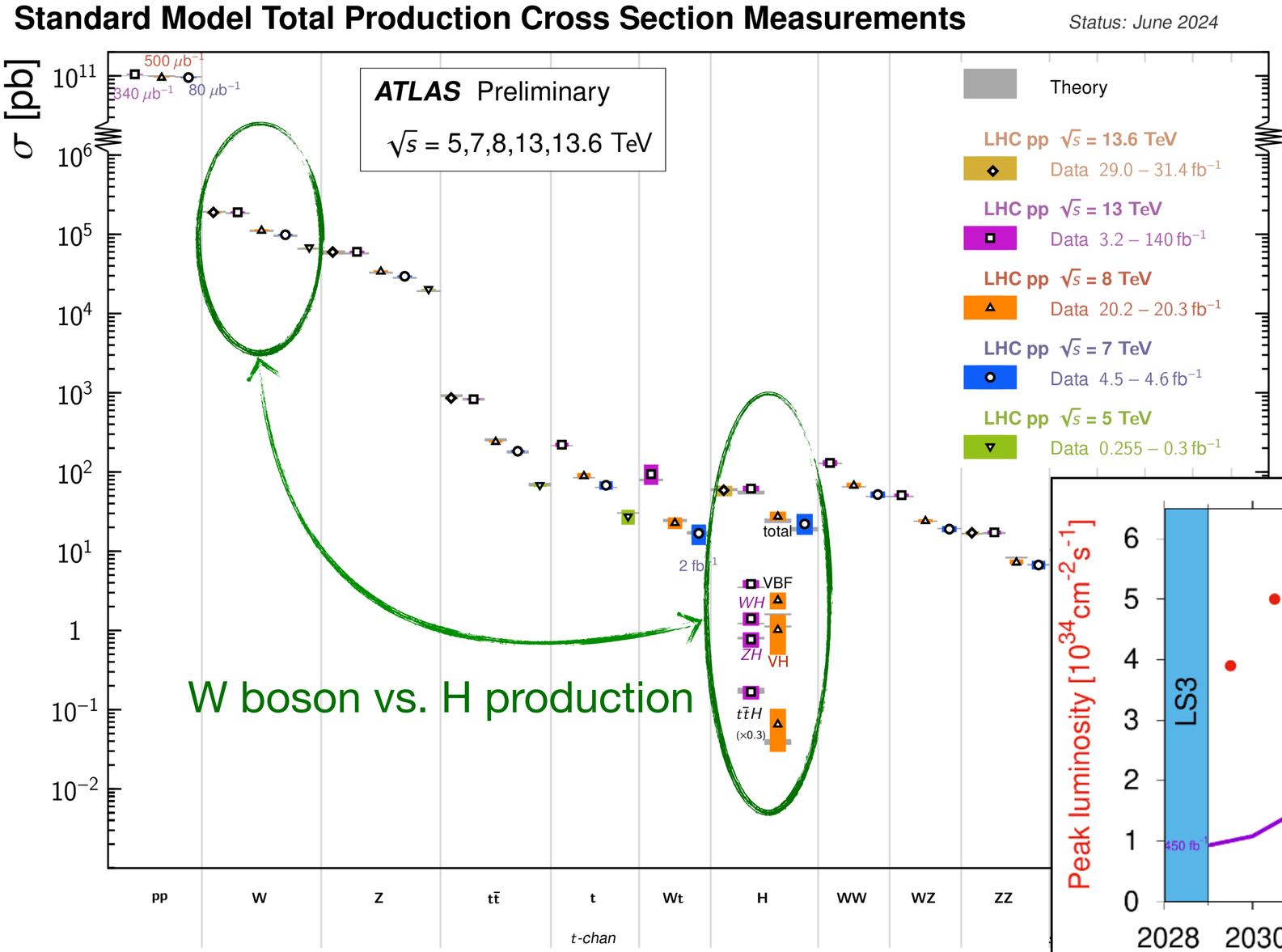
The CMS W mass analysis workflow and framework

a blueprint for analysis at the HL-LHC

EPS-HEP 2025

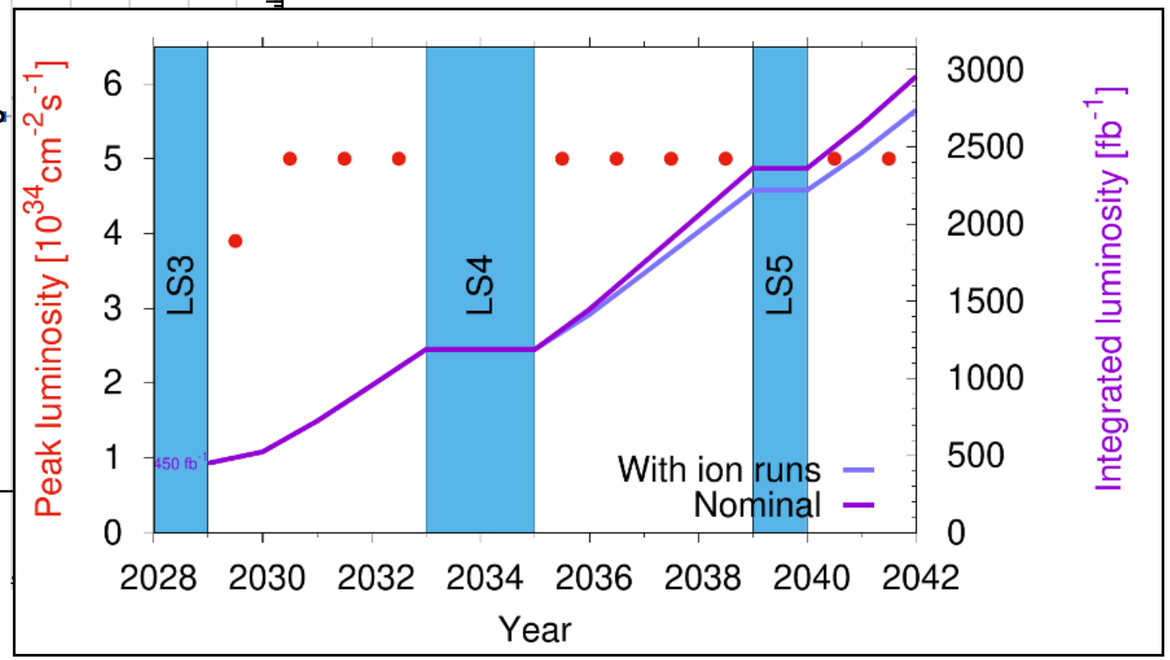
Kenneth Long

Introduction



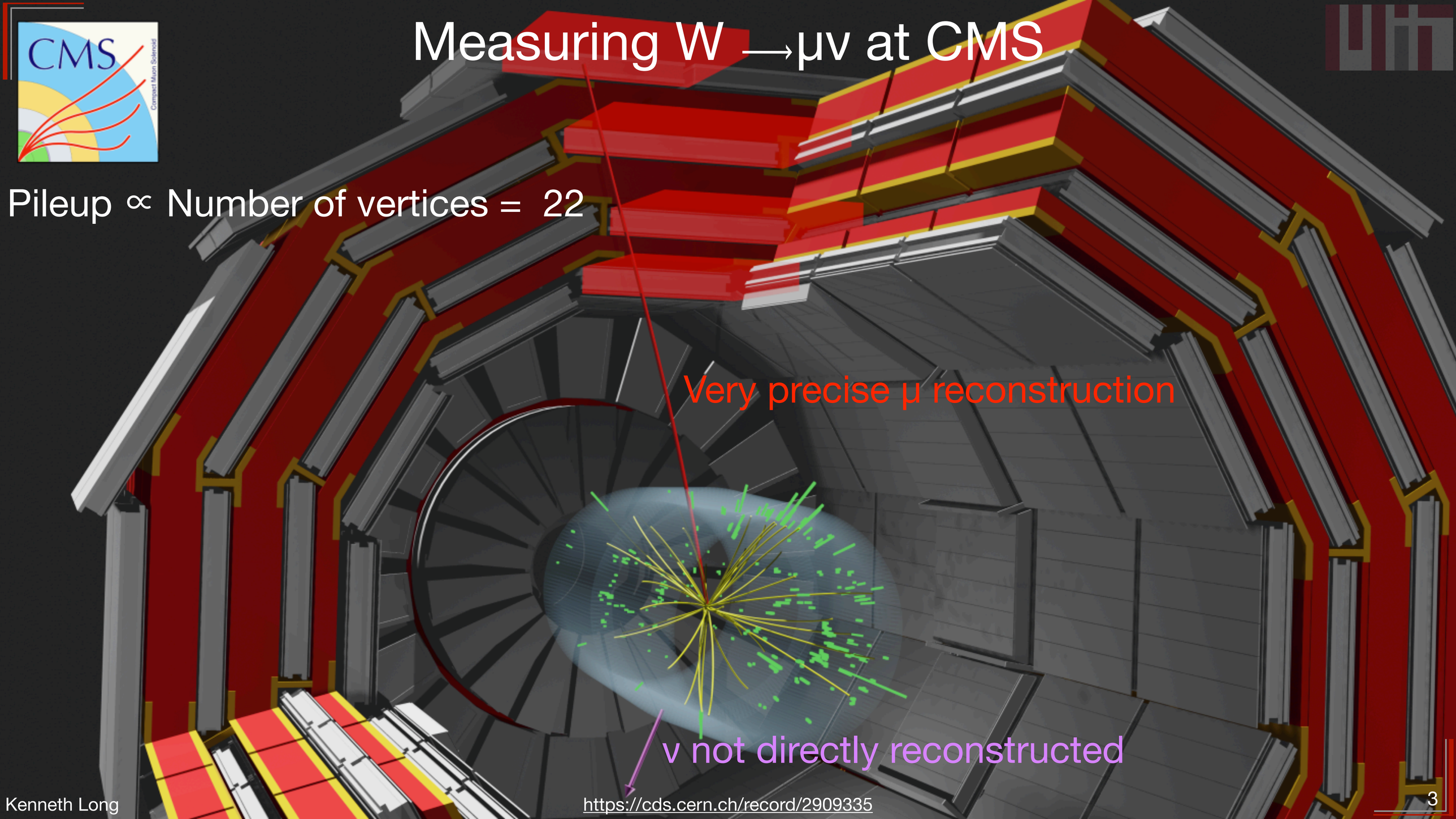
- Run 2-3 datasets are unprecedented
  - Rate of accumulation will only increase with HL-LHC
- W boson prod. is ~O(1000)x greater than Higgs production
  - Need to cope with huge datasets

➡HL-LHC is now!



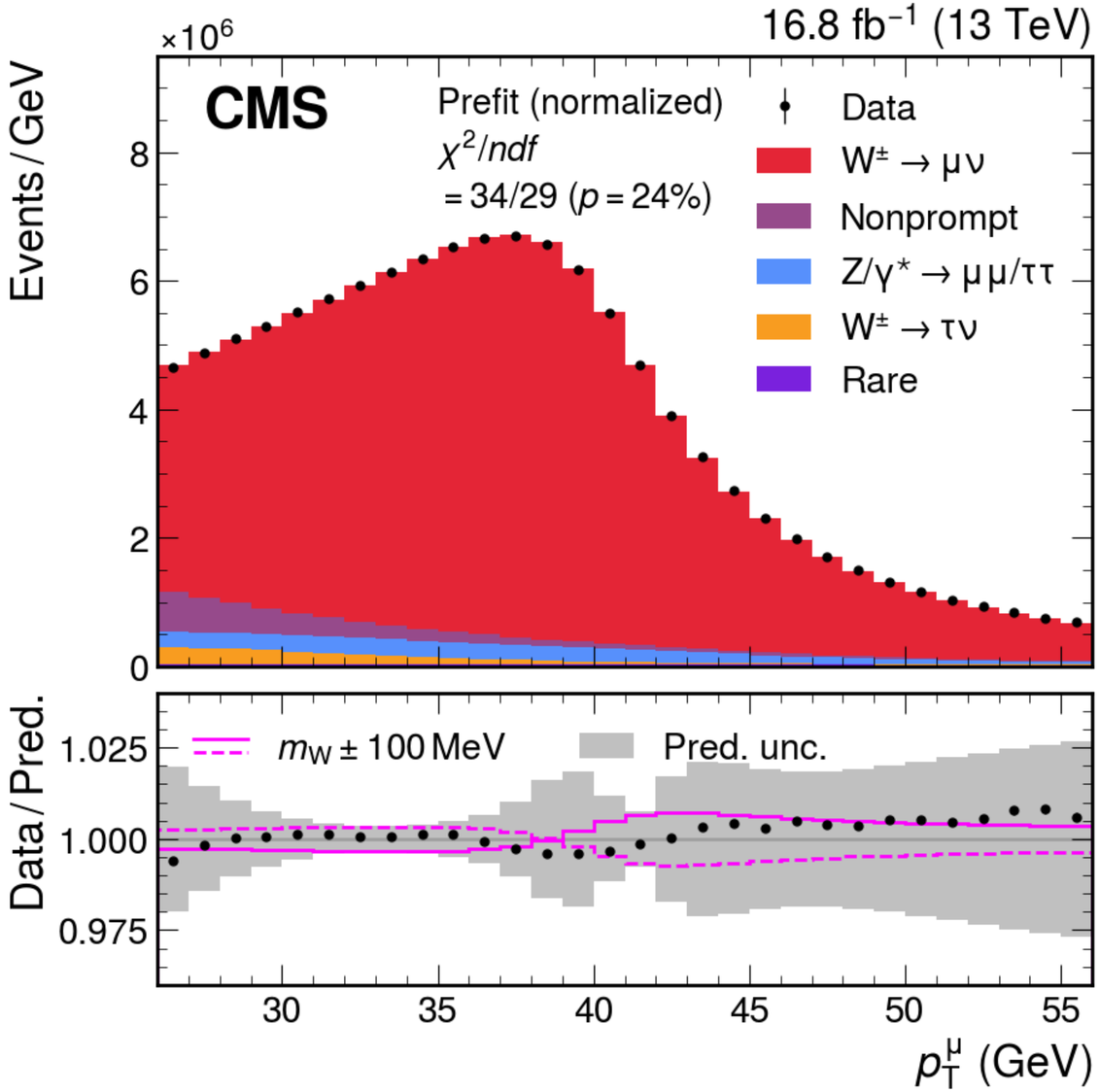
Measuring W $\longrightarrow \mu\nu$ at CMS

Pileup $\propto$ Number of vertices = 22

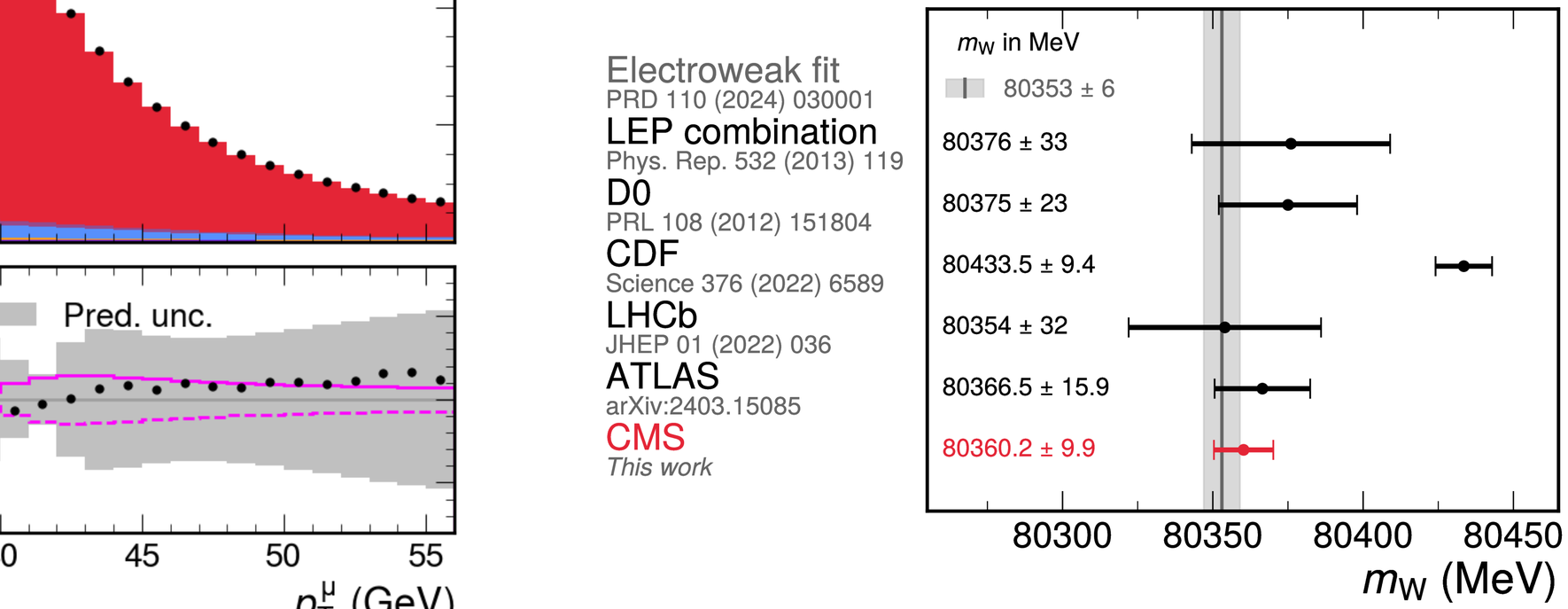


https://cds.cern.ch/record/2909335

m_W measurement at a glance



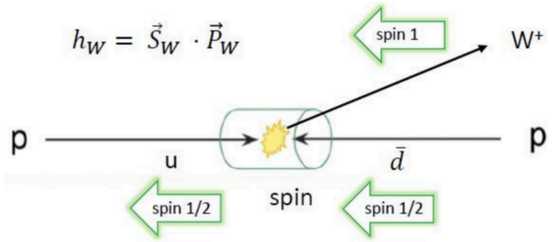
https://arxiv.org/abs/2412.13872

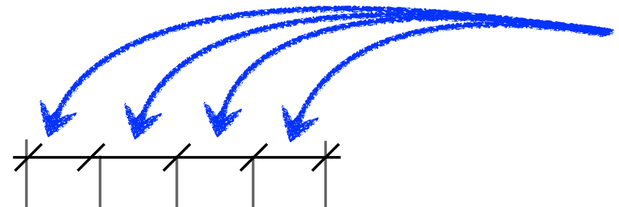


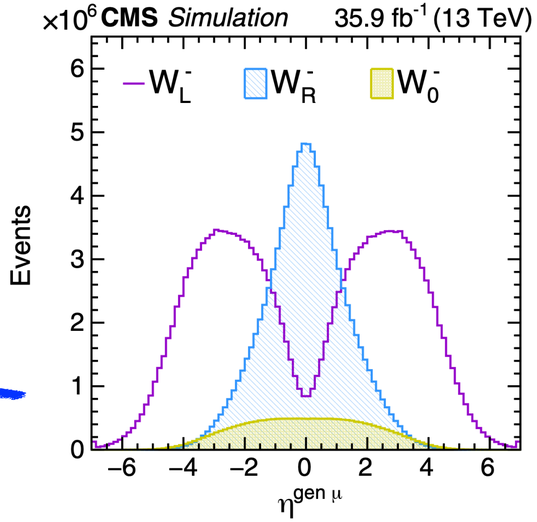
- Binned maximum likelihood fit: test consistency of data with different m_W hypotheses

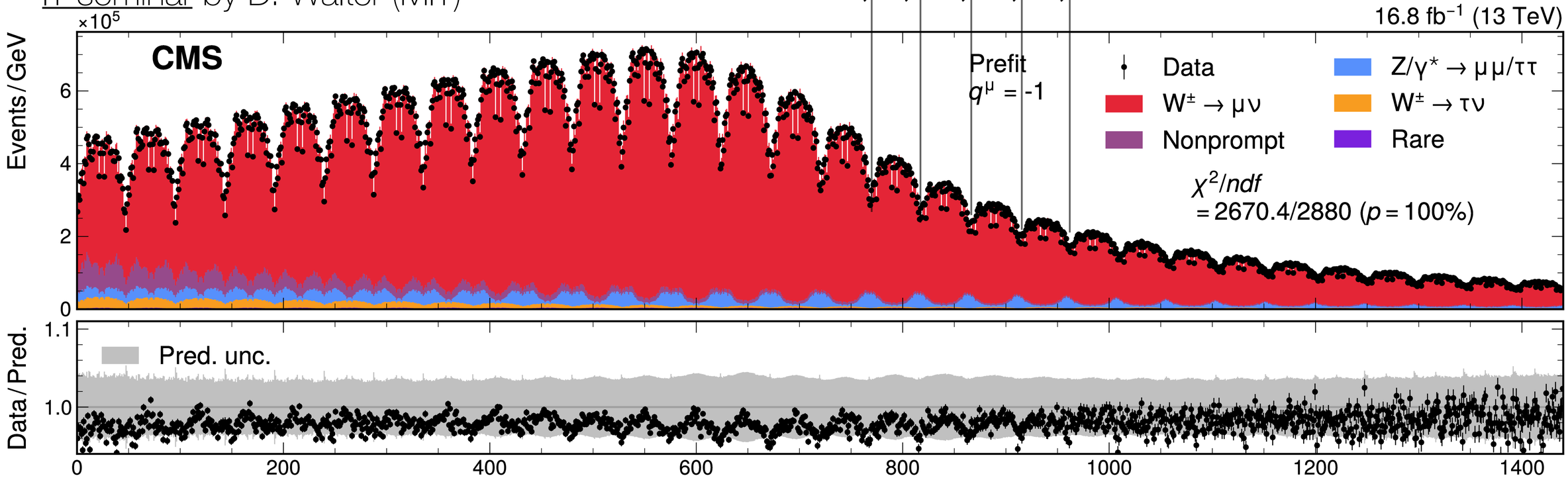
The m_W measurement at CMS

- y^W (η^μ), is dependent on W helicity, driven by PDFs
  - Sensitivity to PDF from η^μ
  - ➡**Extract mass from fit to (q^μ, η^μ, p_T^μ) distribution**
  - ~2000 bins and 5000 nuisance parameters
  - Major computational challenge!
    - This talk is a condensed version of a detailed CERN IT seminar by D. Walter (MIT)





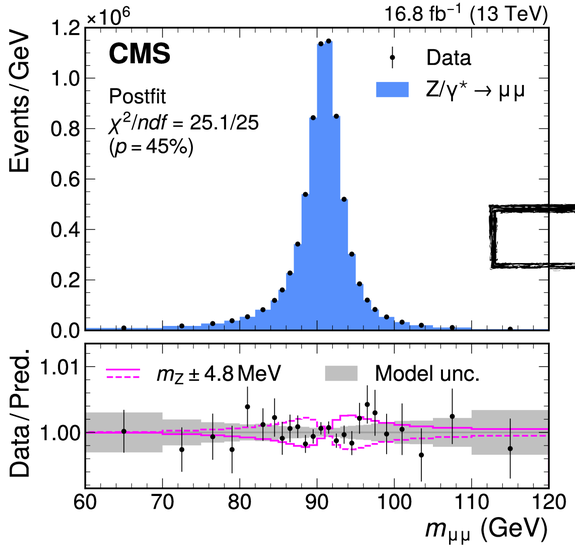




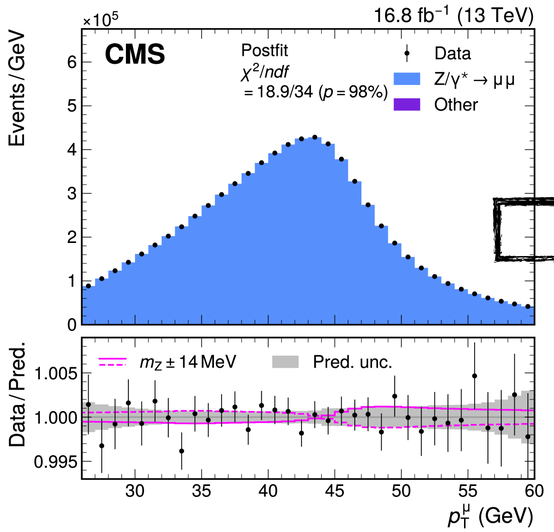
1D visualisation of 2D distribution: η^μ in 1 GeV bins of p_T^μ from 26-56 GeV

More than a single measurement!

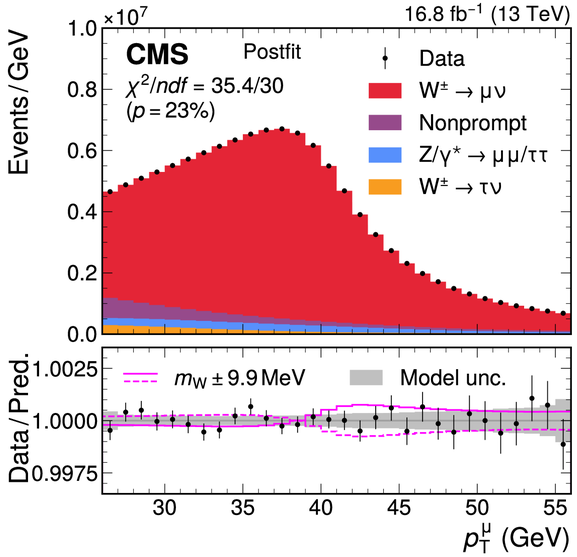
★ m_Z measurement from $m_{\mu\mu}$

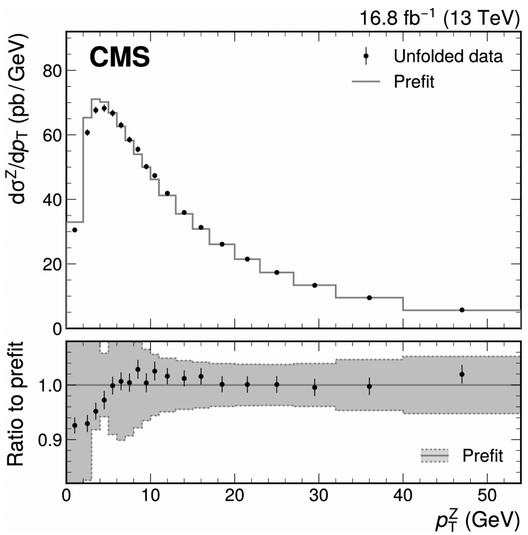


★ m_Z measurement from p_T^μ

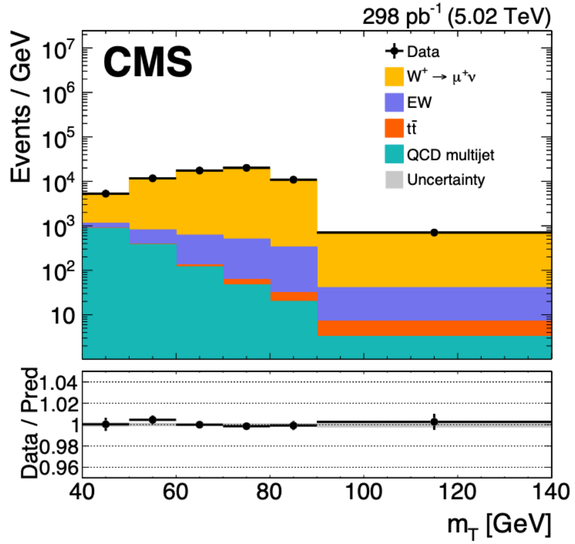


★ m_W measurement

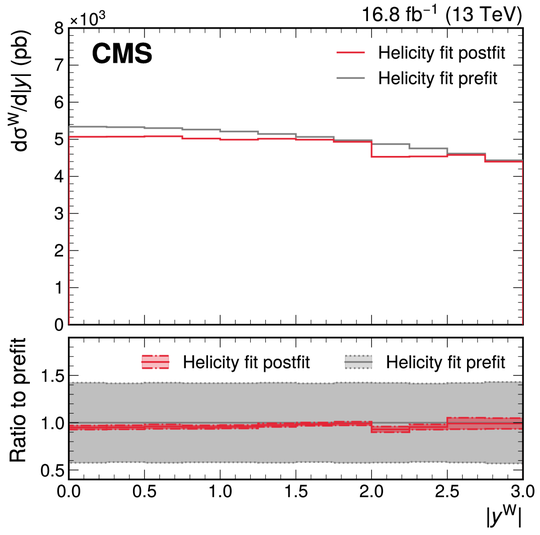




Unfolding, simulation studies



Analysis of low pileup data



Helicity cross section fit

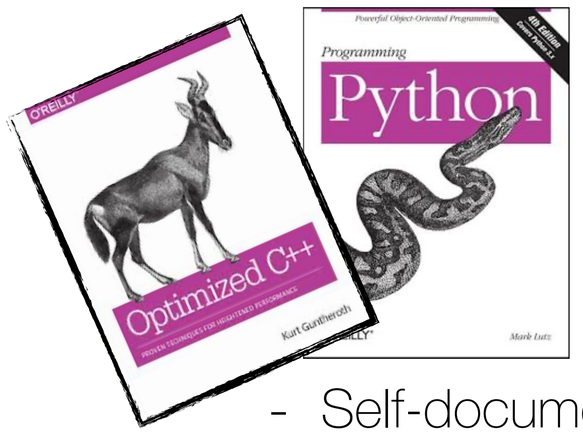
+ … ?

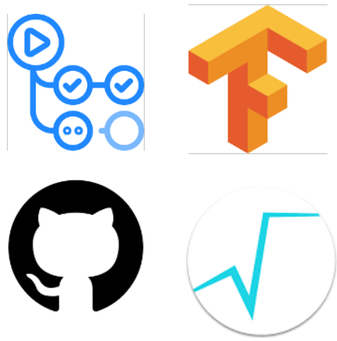
Design considerations and choices

- Performant!
  - Fast time-to-insight while analysing large dataset
  - Full exploit computational resources
    - Highly parallelised
  - Low level code in C++
  - Avoid thread locking actions (e.g., unnecessary memory allocations)

- Robust and transparent
  - Well validated
  - Well documented

Design choices



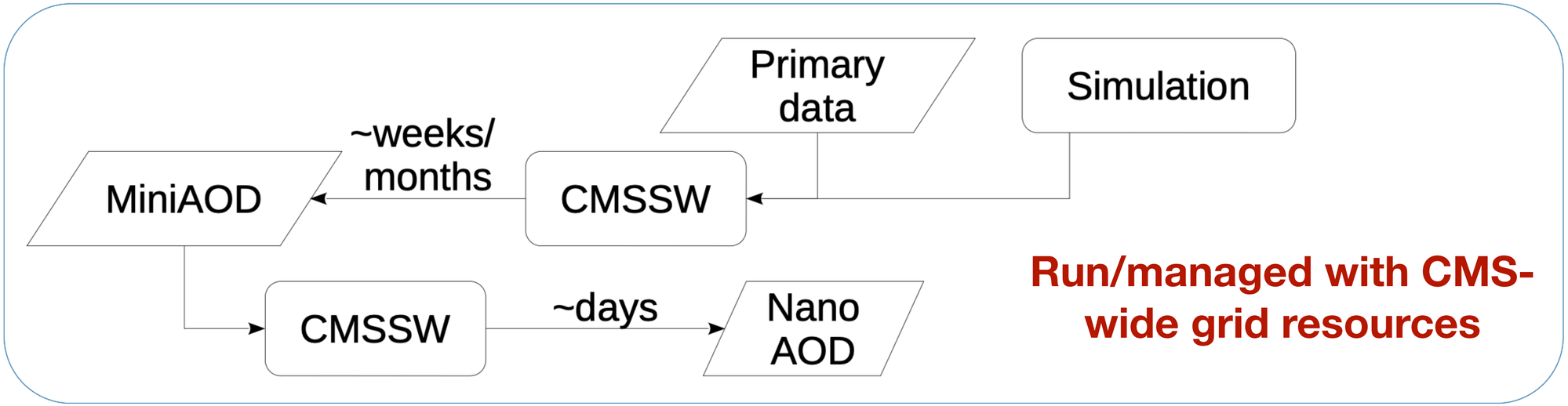


- Self-documenting
- Meticulous logging
- CI/CD with Github actions

- Low barrier to entry
  - Flexible/general design
  - Separation of tasks
  - Easily extensible
  - Steer and postprocess with python
  - Use libraries, favour general implementations
  - Examples and user support

Ideals can be in conflict
⟹ a balancing act!

Outline of the data processing workflow



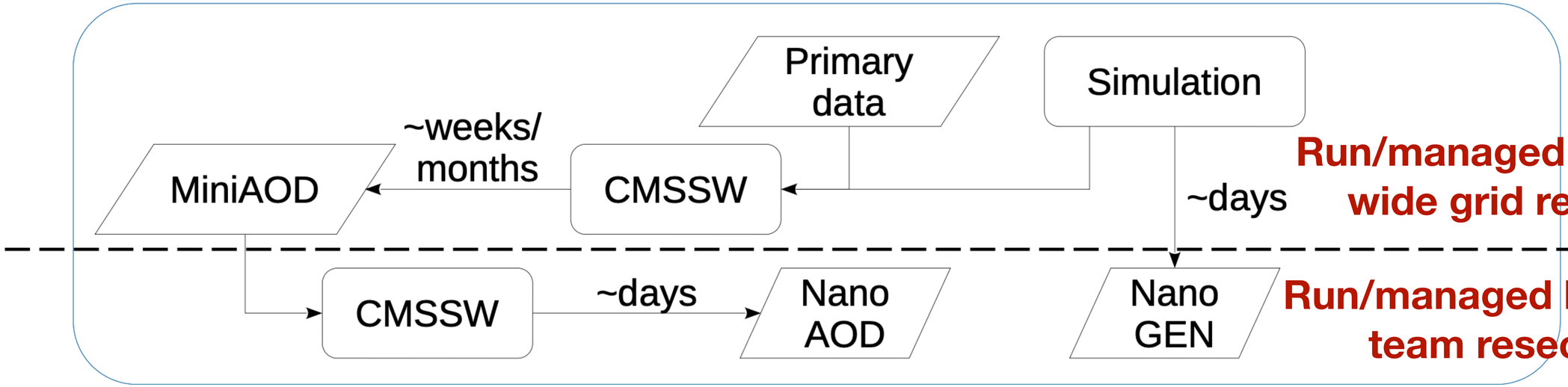
- Raw data (and simulation) processed with the standard CMS reconstruction chain (EDM format instantiates C++ objects)

➡Final, lightweight **NanoAOD** produced with collaboration wide resources in standard processing chain[0,1]

  - Flat ROOT TTree with only data primitive types (or arrays of primitives)
  - Independent of experiment specific software (e.g., no custom C++ objects)
  - High level physics objects (p_T, η, ϕ, ID, ... of muons, electrons, jets, …)
  - ~2kB per event
  - Good for ~50% of analyses

Data tier	Size (kB)
RAW	1000
Gen	<50
SIM	1000
DIGI	3000
RECO(SIM)	3000
AOD(SIM)	400
MiniAOD(SIM)	50
NanoAOD(SIM)	2

Outline of the data processing workflow



➡Final, lightweight **NanoAOD** produced with collaboration wide resources in standard processing chain[0,1]

- Flat ROOT TTree with only data primitive types
- Independent of experiment specific software (e.g., no custom C++ objects)
- High level physics objects (p_T, η, ϕ, ID, … of muons, electrons, jets, …)
- ~2kB per event

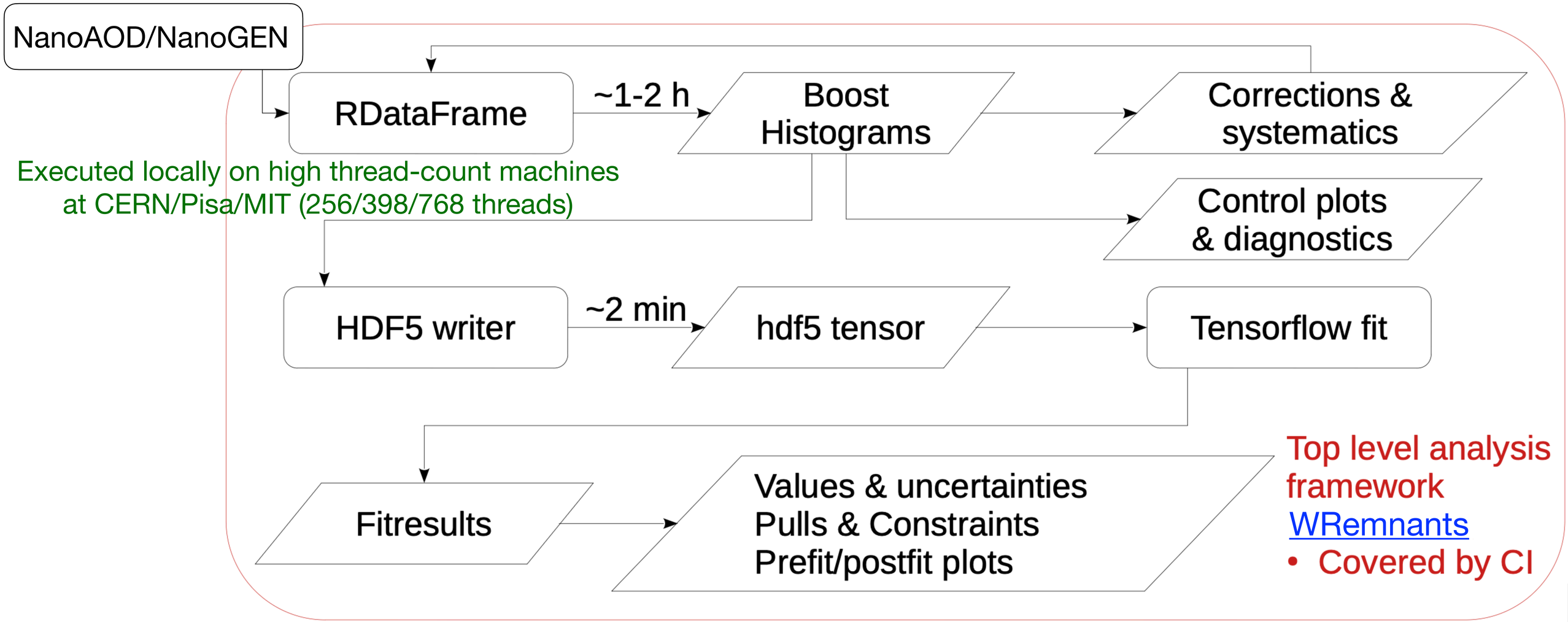
- **Easily customisable, important for this analysis**
  - Refit muon tracks, store low-level fit information, additional generator information (e.g., more PDF sets…)

(Ran ~10 times over 3 years)

Data tier	Size (kB)
RAW	1000
Gen	<50
SIM	1000
DIGI	3000
RECO(SIM)	3000
AOD(SIM)	400
MiniAOD(SIM)	50
NanoAOD(SIM)	2

The “analysis” steps of the processing workflow

- **Process ~1 B data events and 4B simulation events** in NanoAOD format (every time, no pre-filtering!)
  - Output high dimensional Boost histograms
  - Store in custom hdf5 format, further processing for statistical analysis, publication-level plots…



Ran on daily basis (>1000 times)

Data processing with RDataframe

- Select objects, filter events, fill histograms
- Pythonic, declarative, graph-style analysis
- Lazy execution: perform all operations in single (parallelised) event loop
- Code JIT compiled
  - From short strings in df.Define()
  - From C++ code, possibly with objects holding user data

```
from ROOT import RDataFrame
df  = RDataFrame(dataset);
df2 = df.Filter("x > 0")
        .Define("r2", "x*x + y*y");

rHist = df2.Histo1D("r2");

g = df2.Graph("x","y")
```

Python access

```
helper = ROOT.wrem.pileup_helper(puweights)
```

```
df = df.Define("weight_pu", pileup_helper, ["Pileup_nTrueInt"])
```

Histogram with corrections as member data

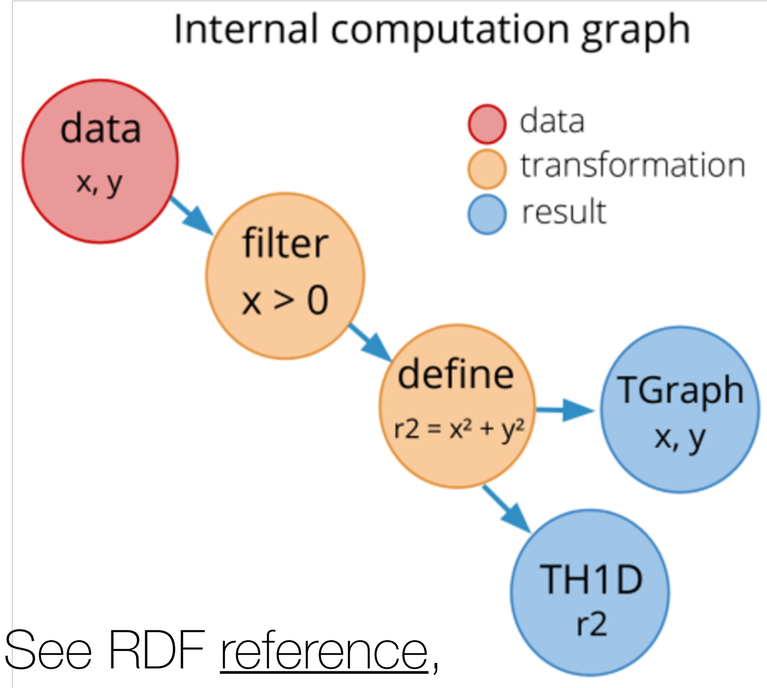
C++ definition

```
class pileup_helper {
public:

  pileup_helper(const TH1D &puweights) :
    puweights_(make_shared_TH1<const TH1D>(puweights)) {}

  // returns the pileup weight
  double operator() (float nTrueInt) const {
    return puweights_->GetBinContent(puweights_->FindFixBin(nTrueInt));
  }


private:
  std::shared_ptr<const TH1D> puweights_;
```



- See RDF reference, documentation, CERN seminar

High-dimensional Boost histograms

- Multi-dimensional histograms are the basic unit of the analysis
  - Results from fit to 3D distribution
  - Additional dimensions define control/signal regions
  - Systematic variations axis (e.g., axis of length 100 for 100 PDF eigenvector variations)
    - Avoids multiple bin lookups with filling histogram with variations defined by weights

★ Largest variation histogram is 8D, total ~20 M bins, ~10 processes = 2.5 GB

- By default RDF paralyses with 1 copy of histograms per thread ⟹ infeasible memory footprint!

➡ Solution: use Boost histogram with std::atomic<double> storage type

  - One copy of histogram shared by all threads

400M (W→μν) events, 10 copies of pdf variation histograms, 256 threads (2xEPYC 7702)

	Hist Type	Hist Config	Evt. Loop	Total	CPUEff	RSS
256 copies (1/thread)	ROOT THnD	10 x 103 x 5D	59m39s	74m05s	0.74	400GB
	ROOT THnD	10 x 6D	7m54s	25m09s	0.27	405GB
Bin lookup per syst entry	Boost ("sta")	10 x 6D	7m07s	7m17s	0.90	9GB
	Boost ("sta")	10 x (5D + 1-tensor)	1m54s	2m04s	0.81	9GB
Minimal graph complexity	Boost ("sta")	1 x (5D + 2-tensor)	1m32s	1m42s	0.77	9GB

Statistical analysis

- Analysis is based on determining the value of m_W that maximizes Likelihood (minimises -2ln L)

$$-\ln L = \sum_{ibin} \left(-n^{obs}_{ibin} \ln n^{exp}_{ibin} + n^{exp}_{ibin} \right) + \frac{1}{2} \sum_{ksyst} \left(\theta_{ksyst} - \theta^0_{ksyst} \right)^2 \quad \text{where} \quad n^{exp}_{ibin} = \sum_{jproc} \mu_{jproc}\, n^{exp}_{ibin,jproc} \prod_{ksyst} \kappa^{\theta_{ksyst}}_{ibin,jproc,ksyst}$$

~2000 Gaussian constraint nuisance parameters ~10 ~5000

- RooFit+Minuit workflow found to be insufficient for minimisation
  - Limited numerical precision/efficiency/run time

- Built custom implementation of likelihood and minimisation in tensorflow:

➡Combinetf (PyHEP talk)

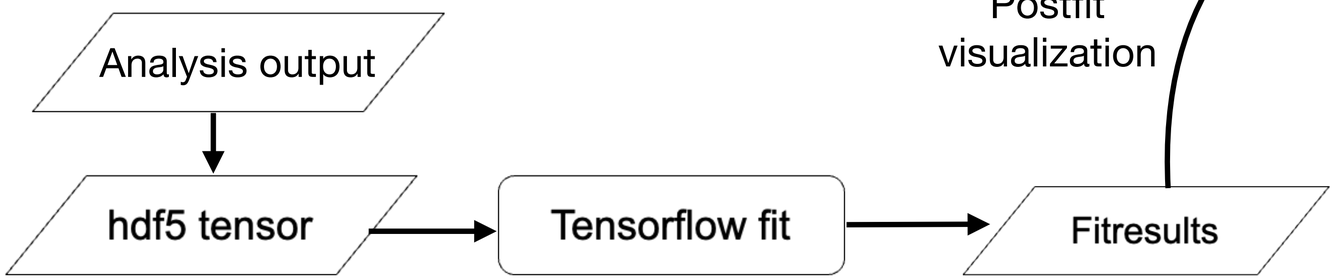
  - Automatic differentiation for exact gradient calculation
  - Custom minimizer to reliably find global minimum in high dimensions based on arXiv:1506.07222
  - Fast—O(10s), numerically accurate, stable
    - Extensively validated against CMS Combine package

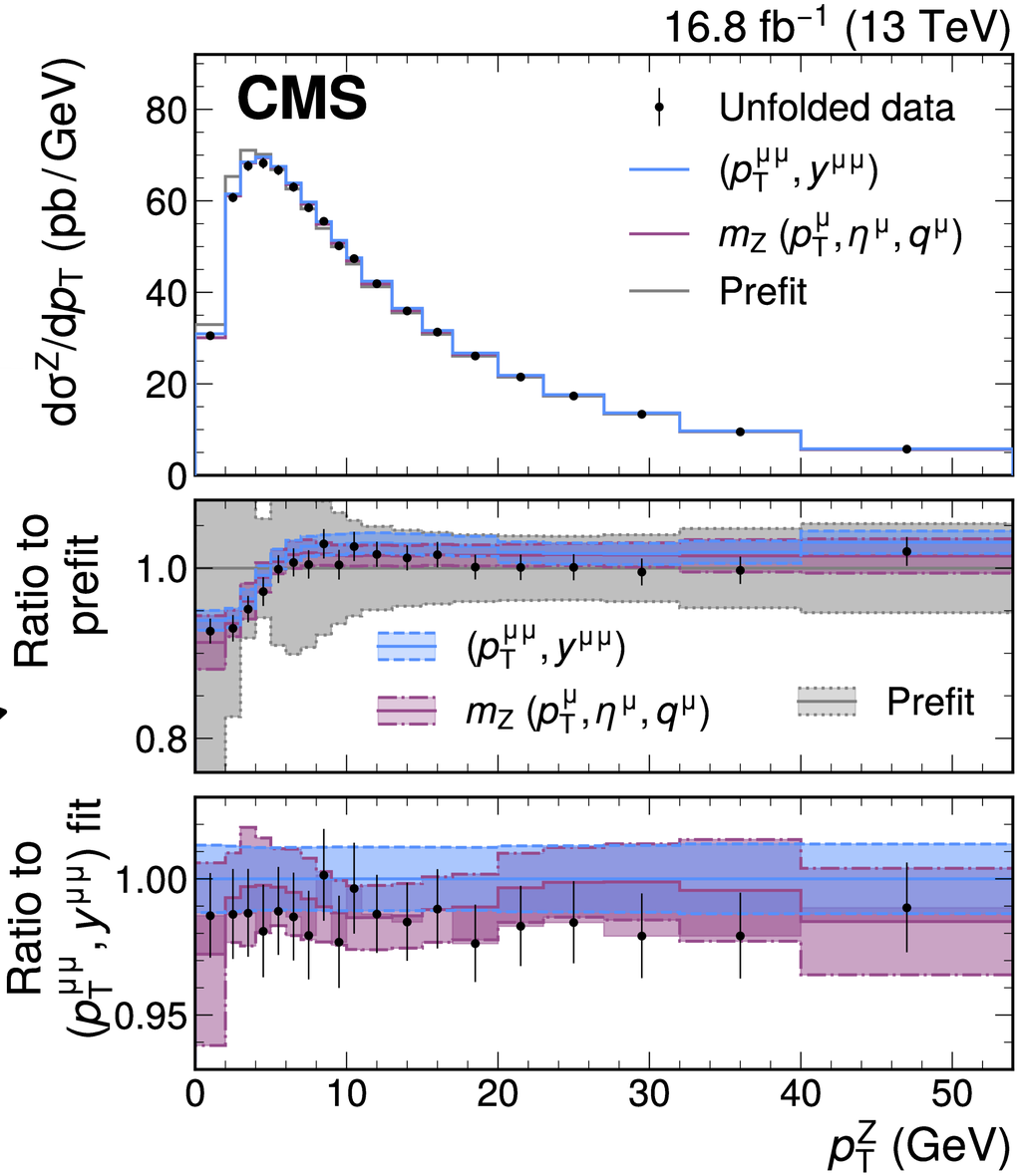
Statistical analysis

- Rewrite in tensorflow2 recently completed

➡**Rapid Automatic Bin Based Inference Tool**

- New UI + More developer friendly
- More efficient computation of hessian and hessian vector products
- Trust-krylov minimizer from scipy
- Native/improved support for plotting post-fit distributions
  - Including applying postfit nuisance pulls/constraints from separate fit
  - Ex: postfit generator-level distribution from fit to reconstructed variables





https://github.com/WMass/rabbit

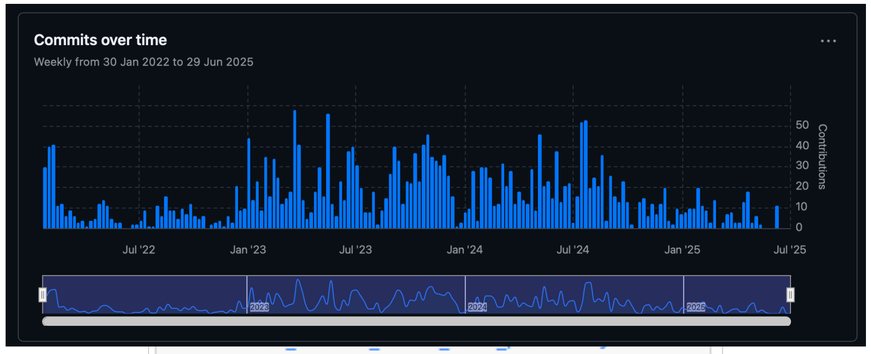
Continuous integration

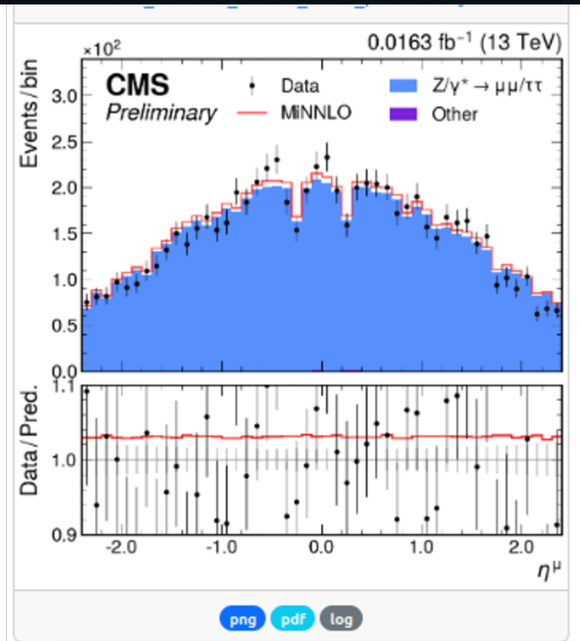
- Common framework for multiple analysis interpretations
  - Reuse existing code, find/avoid bugs, save time
  - Rapid developement with O(10) contributors
    - >600 pull requests (PRs)

- Updates often unintentionally affected (or break) other parts
  - Not noticed immediately, difficult to trace down source
  - Harder to fix after the fact

- Solution → GitHub actions: platform for automate developer workflows
  - Use continuous integration and deployment (CI/CD) pipeline
  - Slim and easily to set up and manage (compared to e.g. Jenkins)

- Locally hosted on dedicated machine at CERN
  - Executed on subset of data for each PR
  - Full stats run 3/week over night

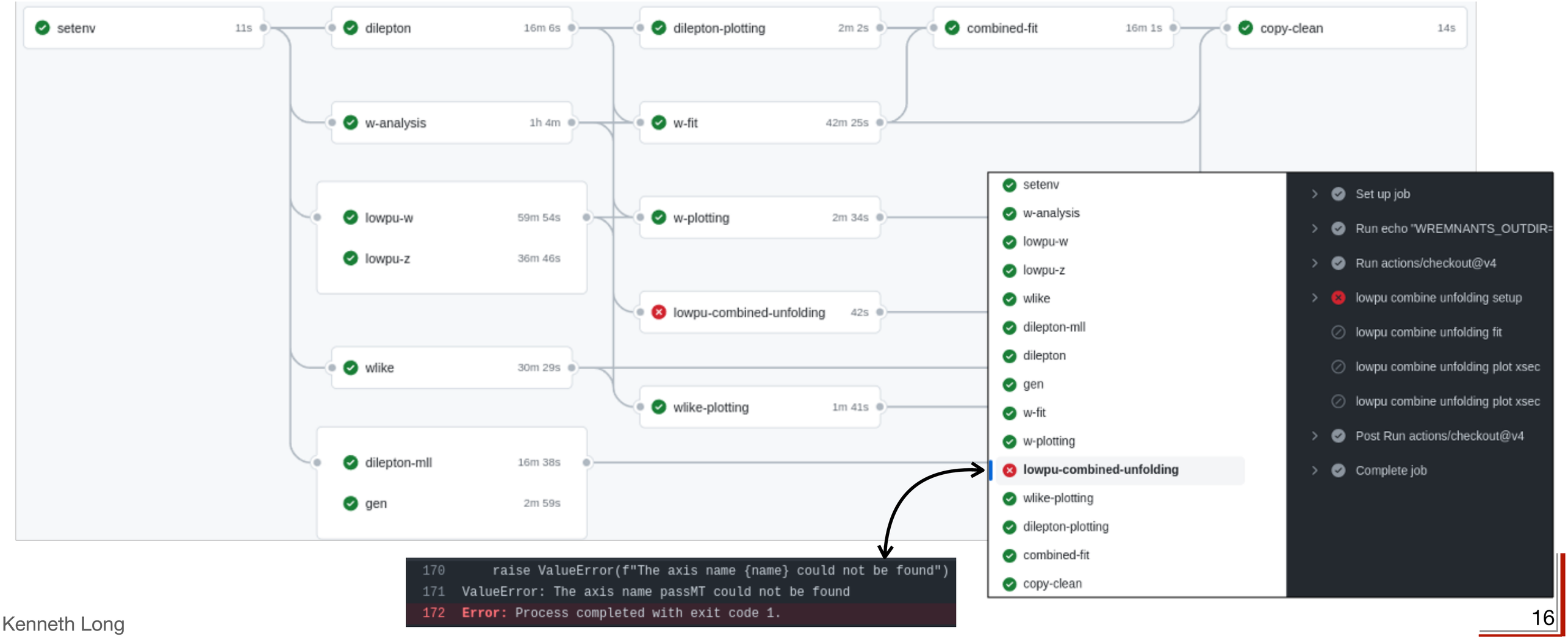




```yaml
on:
  pull_request:
    branches: [ main ]
  schedule:
    - cron: '30 5 * * 1-5'
    - cron: '0 1 * * 2,4,6'
  workflow_dispatch:
```

Continuous integration pileline

- Execute full graph of analysis workflows and dependencies for every PR
- independent steps run in parallel, error stop further processing
- Linters (Black, Flake, isort) check code quality and basic errors in first step



Continuous integration and self-documentation: summary pages

- Full result produced on interactive web pages
  - Plots for all analysis
  - Log files with yields
  - Uncertainty Impacts, pulls/constraints
- Allows precise validation of changes to physics results

/eos/home-i04/c/cmsmwbot/www/WMassAnalysis/PRValidation/PR598/2025_06_20

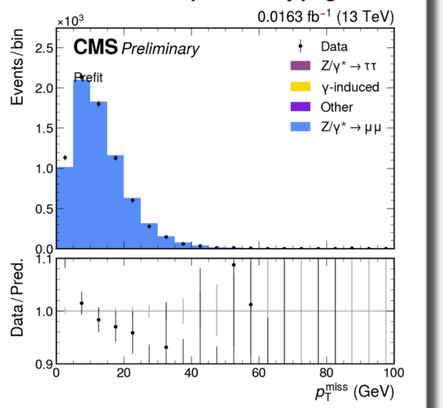
Directories

[parent] [lowPU] [unfolding_dilepton] [unfolding_mw]

Plots

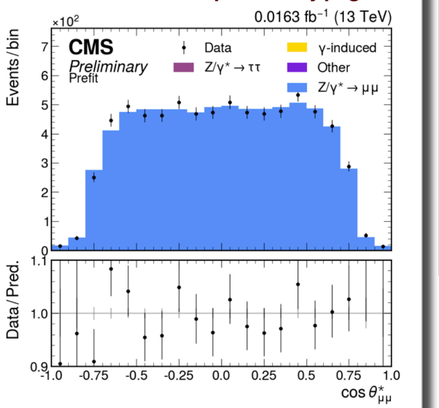
Filter: Go RegExp

MET_wlike_preliminary.png



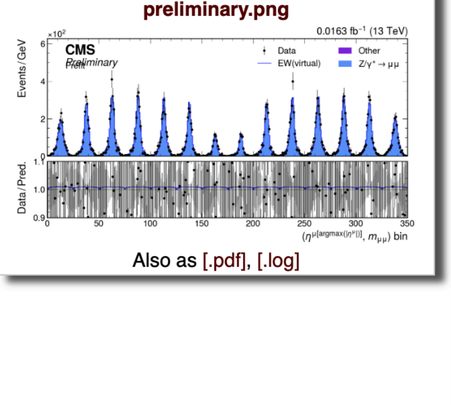
Also as [.pdf], [.log]

cosThetaStarll_z_preliminary.png



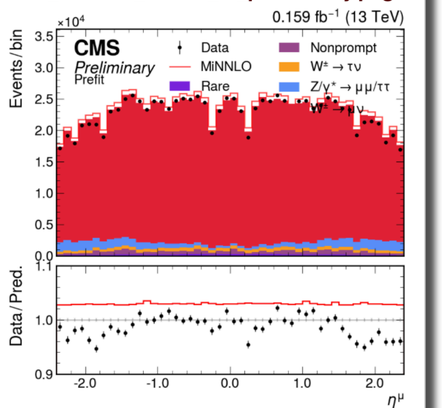
Also as [.pdf], [.log]

etaAbsEta_mll_weak_default_z_-preliminary.png

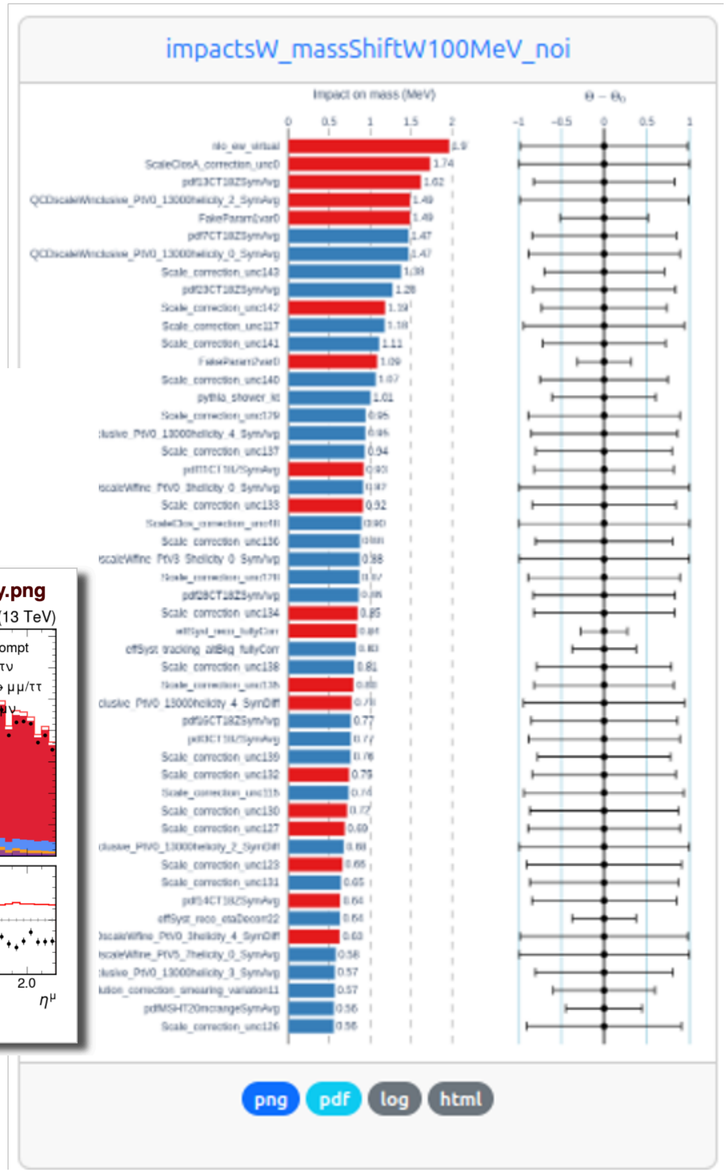


Also as [.pdf], [.log]

eta_nominal_uncorr_W_preliminary.png



Also as [.pdf], [.log]



Continuous integration and self-documentation: logging

- Every output file and plot contains all meta data needed to reproduce it
  - Command used to produce it
  - Git hash/diff of repo when it was created
  - Same information about input file(s) needed
- Additional useful summary information for plots

```
--------------------------------------------------------------------------------
Script called at 2024-10-01 15:56:53.608388
The command was: scripts/plotting/postfitPlots.py
'/scratch/dwalter/CombineStudies/test/ZMassDilepton_ptll_yll/fitresults_123456789.root' --
legCols 1 --eoscp -f '241001_test' --yscale '1.25'
```

```
Yield information for Stacked processes
--------------------------------------------------------------------------------
                          Process       Yield  Uncertainty
0               $\gamma$-induced     1796.97        94.01
1  Z/$\gamma^{\star}\to\tau\tau$     1582.57        77.61
2                          Other     1630.27        14.59
3    Z/$\gamma^{\star}\to\mu\mu$  1931915.83       336.74

Yield information for Unstacked processes
--------------------------------------------------------------------------------
     Process       Yield  Uncertainty
0       Data  1936925.64      2547.78
1  Inclusive  1936925.64       313.63

===> Sum unstacked to data is 100.00%
```

```
Meta info from input file AnalysisOutput

--------------------------------------------------------------------------------
{
     "time": "2024-10-01 15:39:52.107792",
     "command": "scripts/combine/setupCombine.py -i
'/scratch/dwalter/results_histmaker/test/mz_dilepton.hdf5' --fitvar 'ptll-yll' --lumiScale
100 --realData -o '/scratch/dwalter/CombineStudies/test'",
     "args": {
          "outfolder": "/scratch/dwalter/CombineStudies/test",
          "inputFile": [
               "/scratch/dwalter/results_histmaker/test/mz_dilepton.hdf5"
          ],
          "postfix": null,
          "verbose": 3,
```

[...]

```
     "git_hash": "\"4713c27278391e1df49f86834c6d122cff8beba5\"
",
     "git_diff": "diff --git a/scripts/combine/saturatedGOF.py
b/scripts/combine/saturatedGOF.py
index 1b9fb2e9..1df140a8 100644
--- a/scripts/combine/saturatedGOF.py
+++ b/scripts/combine/saturatedGOF.py
@@ -15,7 +15,7 @@ tree.GetEntry(0)

 fitresult_h5py = combinetf_input.get_fitresult(args.infile.replace(\".root\",\".hdf5\"))
 meta = ioutils.pickle_load_h5py(fitresult_h5py[\"meta\"])
-nbins = sum([np.product([len(a) for a in info[\"axes\"]]) for info in
meta[\"channel_info\"].values()])
+nbins = sum([np.prod([len(a) for a in info[\"axes\"]]) for info in
meta[\"channel_info\"].values()])
 ndf = nbins - tree.ndofpartial
```

Conclusions and future perspectives

- High-performance software, computing played a major role in the CMS m_W measurement
  - Software designing for the task was a significant collaborative effort
  - Largely successful! But trade-offs necessary, and improvements continue
  - Many more interesting details of the analysis, see CERN IT seminar by D. Walter (MIT) for more

- Developments being leveraged more widely within CMS
  - Some optimisations integrated upstream into ROOT (e.g., xrootd file reading) or planned for future development (atomic histogram filling)
  - Libraries or code can be used by other analyses (extended NanoAOD, luminosity counter)

- Performance and design considerations are highly relevant for the HL-LHC era
  - Exact software solutions may change over time
  - Core design considerations and principles that form the foundation of successful analysis today will likely stay relevant

Backup

Hardware and resources

- The “analysis” (data processed into histrograms) step is executed locally
  - No resubmission of failed jobs/ merging of jobs etc.
  - Direct feedback on progress
  - Heavily multithreaded

	CERN	MIT/Pisa	MIT
CPU	2 EPYC 7702	2 EPYC 7702	2 EPYC 9965
cores	128	192	384
threads	256	384	768
memory	1TB	1.5/2TB	1.5 TB

- Necessitates high performance machine with high availability
  - High performance, high thread count machines (256/398/768 threads) at CERN, Pisa, MIT
  - Reading/writing on fast NVMe SSDs
  - Local or via network interface 100Gbit/s (e.g., from CERN eos via xrootd)

- Sounds like a luxury that cannot be widely adopted but…
  - Price/core is increasingly competitive with several low-core machines
  - Can be seamlessly integrated into condor/slurm/etc cluster
  - For future: DistRDF allows interactive-like running

https://arxiv.org/abs/2506.01958