# CMS analysis @subMIT
# With distributed computing

Simon Rothman,
From Phil Harris group

# Overview

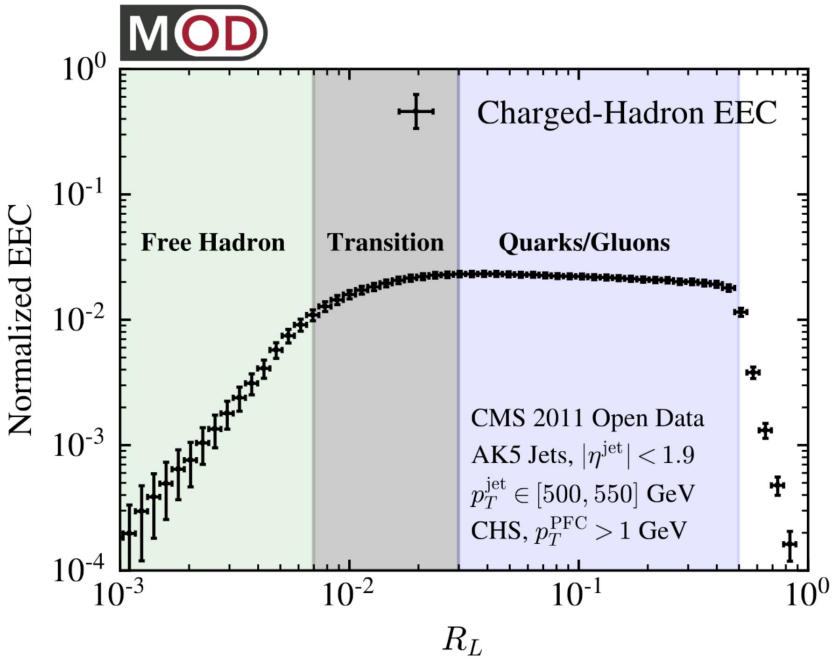
- We are members of CMS experiment
  - ~ 2-3k collaborators across world
  - Petabytes of data

# Overview

- We are members of CMS experiment
  - ~ 2-3k collaborators across world
  - Petabytes of data
  - Free trips to Europe

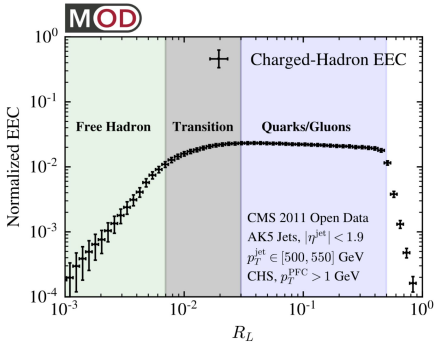
# Overview

- We are members of CMS experiment
  - ~ 2-3k collaborators across world
  - Petabytes of data
  - Free trips to Europe
- My current focus is precision measurement of strong force
  - New observables
  - Strong theory-experiment correspondence



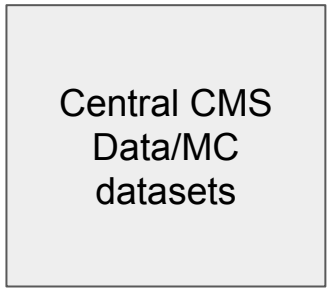
# Overview

- We are members of CMS experiment
  - ~ 2-3k collaborators across world
  - Petabytes of data
  - Free trips to Europe
- My current focus is precision measurement of strong force
  - New observables
  - Strong theory-experiment correspondence
- For today, interesting point is computational requirements
  - Need access to CMS global resources
  - Need substantial local resources (and a way to make use of them!)
  - **Will talk some more about this**



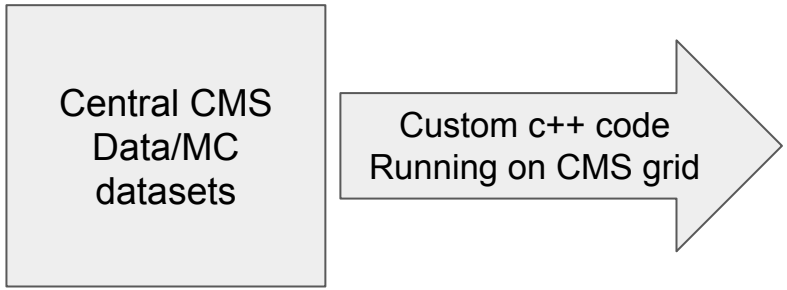

| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 853624 | srothman | 20 | 0 | 2444132 | 562824 | 114348 | S | 103.3 | 0.0 | 0:33.69 | python |
| 853629 | srothman | 20 | 0 | 2210112 | 551460 | 114036 | S | 102.9 | 0.0 | 0:42.95 | python |
| 853633 | srothman | 20 | 0 | 2174712 | 517960 | 113632 | S | 102.9 | 0.0 | 0:30.69 | python |
| 853497 | srothman | 20 | 0 | 2436304 | 547868 | 112460 | S | 102.6 | 0.0 | 0:41.94 | python |
| 853527 | srothman | 20 | 0 | 2208340 | 546480 | 112412 | S | 102.6 | 0.0 | 0:36.87 | python |
| 853695 | srothman | 20 | 0 | 2188856 | 530800 | 114692 | S | 102.6 | 0.0 | 0:26.06 | python |
| 853615 | srothman | 20 | 0 | 2210284 | 553148 | 113652 | S | 102.3 | 0.0 | 0:41.90 | python |
| 853653 | srothman | 20 | 0 | 2213692 | 561100 | 112748 | S | 102.3 | 0.0 | 0:38.88 | python |
| 853665 | srothman | 20 | 0 | 2213700 | 556148 | 113892 | S | 102.3 | 0.0 | 0:42.35 | python |
| 853683 | srothman | 20 | 0 | 2441828 | 570408 | 113468 | S | 101.6 | 0.0 | 0:37.50 | python |
| 853671 | srothman | 20 | 0 | 2215596 | 553720 | 113968 | S | 101.3 | 0.0 | 0:31.86 | python |
| **853596** | **srothman** | **20** | **0** | **2213040** | **554180** | **112932** | **R** | **101.0** | **0.0** | **0:42.87** | **python** |
| 853611 | srothman | 20 | 0 | 2215496 | 555356 | 113780 | S | 101.0 | 0.0 | 0:43.25 | python |
| 853623 | srothman | 20 | 0 | 2219612 | 554552 | 113584 | S | 101.0 | 0.0 | 0:43.20 | python |
| 853677 | srothman | 20 | 0 | 2223816 | 559496 | 113380 | S | 100.3 | 0.0 | 0:40.58 | python |
| 853657 | srothman | 20 | 0 | 2211808 | 554016 | 113700 | S | 99.7 | 0.0 | 0:41.38 | python |
| 853642 | srothman | 20 | 0 | 1576484 | 514304 | 113840 | S | 84.0 | 0.0 | 0:30.37 | python |
| 853641 | srothman | 20 | 0 | 2450916 | 553604 | 114052 | S | 69.9 | 0.0 | 0:33.55 | python |
| 853587 | srothman | 20 | 0 | 2438812 | 550628 | 113692 | S | 69.6 | 0.0 | 0:30.74 | python |
| 853661 | srothman | 20 | 0 | 1604188 | 539536 | 114108 | S | 68.6 | 0.0 | 0:29.62 | python |

# Typical workflow




- 10s of TB
- In distributed storage across the world

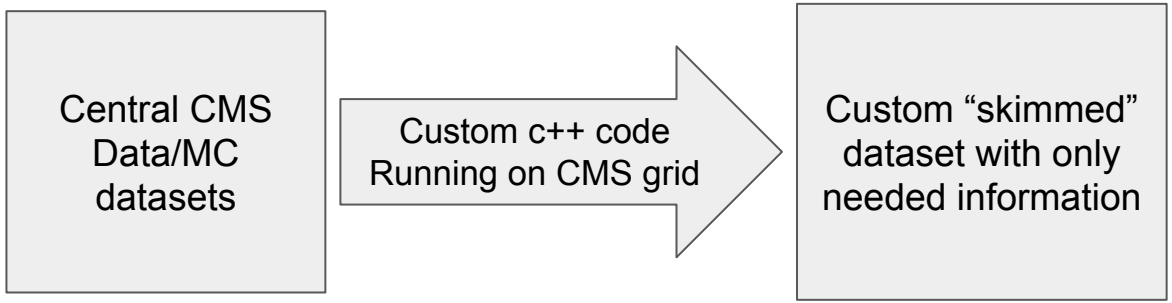
# Typical workflow




- 10s of TB
- In distributed storage across the world

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

# Typical workflow

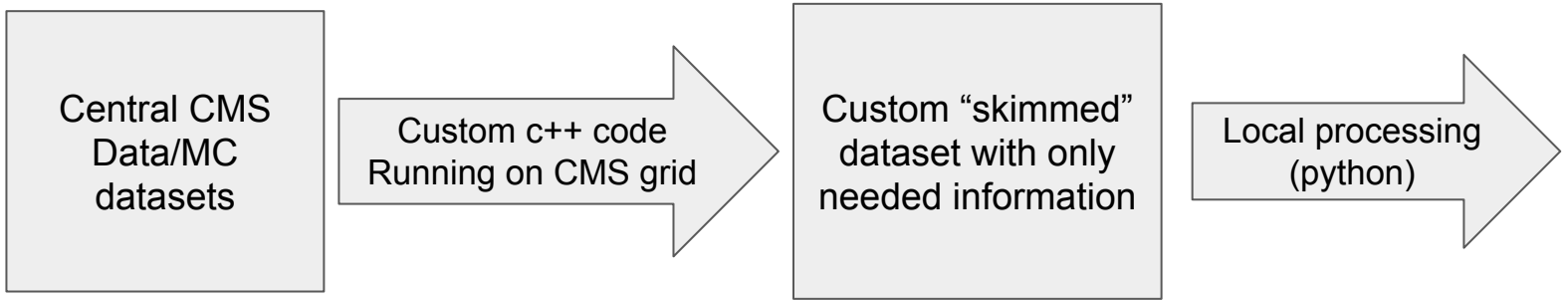



- 10s of TB
- In distributed storage across the world

- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

# Typical workflow



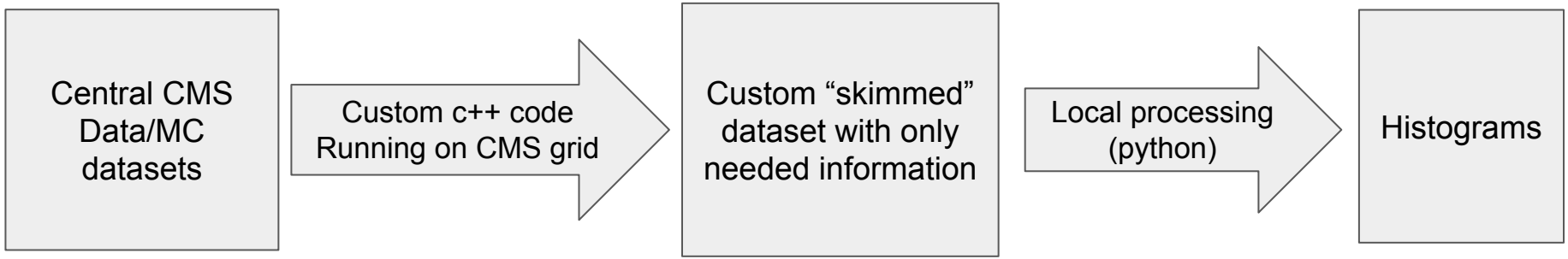

- 10s of TB
- In distributed storage across the world

- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

Requires environment with
- Powerful and fast python tools
- Ability to scale out
- Want to be able to process millions of events per minute

# Typical workflow




- 10s of TB
- In distributed storage across the world

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

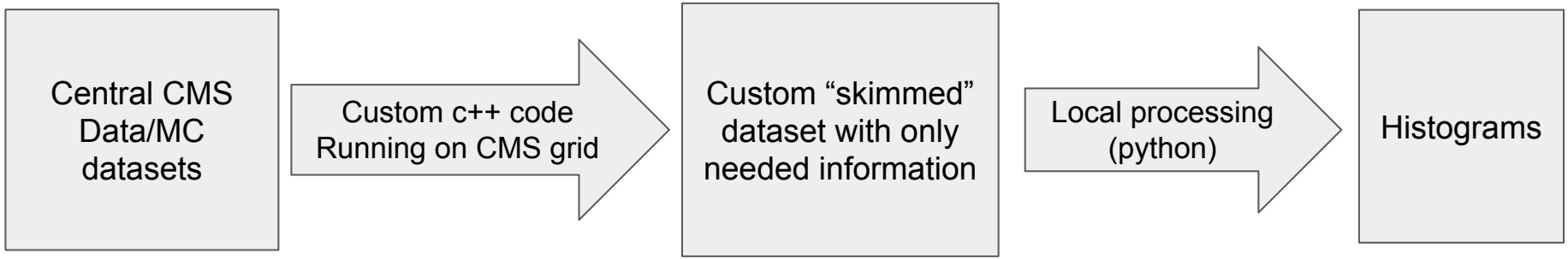
- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

Requires environment with
- Powerful and fast python tools
- Ability to scale out
- Want to be able to process millions of events per minute

- Stored @ subMIT
- Ready for making plots, conclusions

**Need at MIT:**
- **CMS development environments**
- **Access to CMS grid**

# Typical workflow




- 10s of TB
- In distributed storage across the world

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

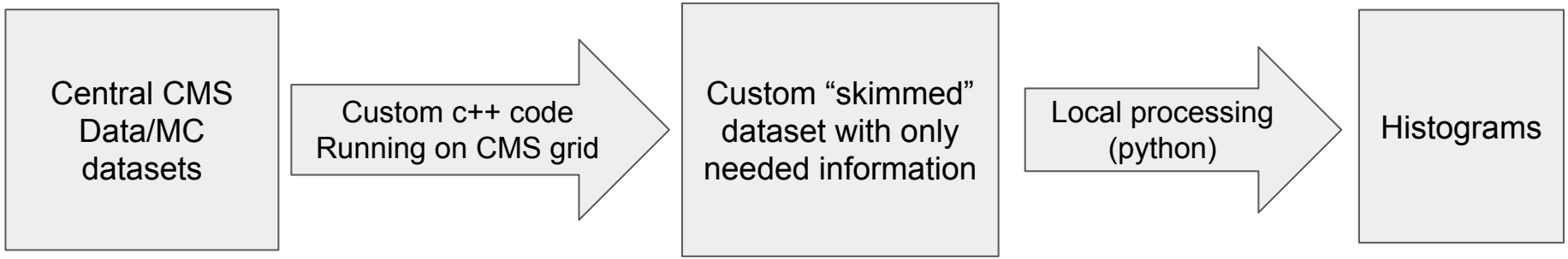
- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

Requires environment with
- Powerful and fast python tools
- Ability to scale out
- Want to be able to process millions of events per minute

- Stored @ subMIT
- Ready for making plots, conclusions

# Typical workflow

**Need at MIT:**
- **CMS development environments**
- **Access to CMS grid**
- **Access to Fermilab storage resources**




- 10s of TB
- In distributed storage across the world

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

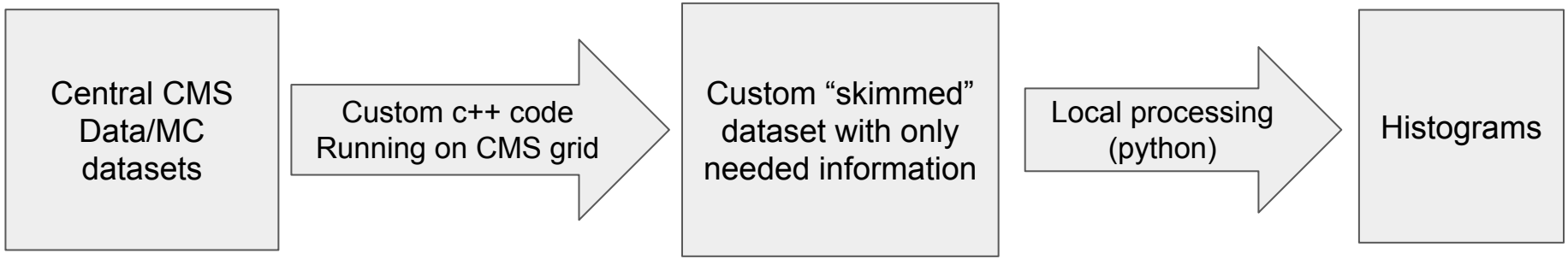
Requires environment with
- Powerful and fast python tools
- Ability to scale out
- Want to be able to process millions of events per minute

- Stored @ subMIT
- Ready for making plots, conclusions

# Typical workflow

**Need at MIT:**

- **CMS development environments**
- **Access to CMS grid**
- **Access to Fermilab storage resources**
- **Ability to scale python analysis tools across many CPUs**
- **Fast networking**




- 10s of TB
- In distributed storage across the world

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

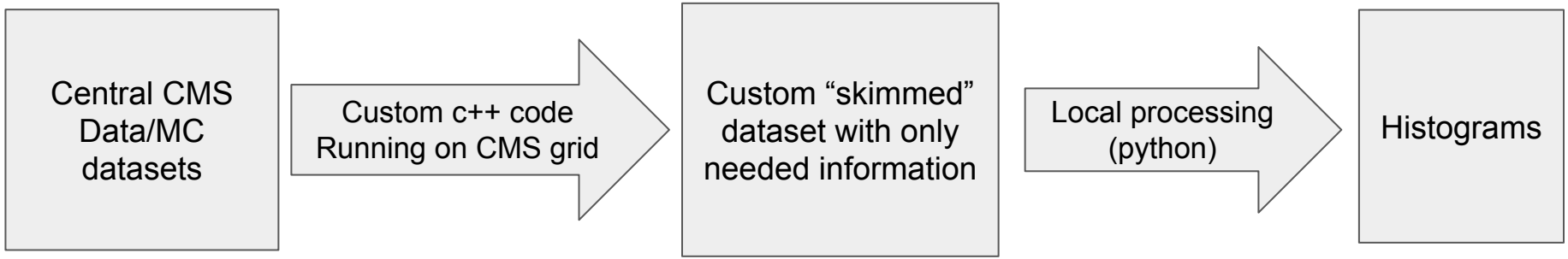
Requires environment with
- Powerful and fast python tools
- Ability to scale out
- Want to be able to process millions of events per minute

- Stored @ subMIT
- Ready for making plots, conclusions

# Typical workflow

**Need at MIT:**

- **CMS development environments**
- **Access to CMS grid**
- **Access to Fermilab storage resources**
- **Ability to scale python analysis tools across many CPUs**
- **Fast networking**
- **Fast local storage resources**




- 10s of TB
- In distributed storage across the world

Requires environment with
- Correctly-configured c++ workspace
- Ability to submit to CMS grid

- Reduced size to O(TB)
- Stored @ FNAL
- Contains only interesting quantities

Requires environment with
- Powerful and fast python tools
- Ability to scale out
- Want to be able to process millions of events per minute

- Stored @ subMIT
- Ready for making plots, conclusions

# CMS computing environment and grid submission

# CMS computing environment

- Development happens in special “CMS-software” (cmssw) environment
- Critical that development environment match deployment on CMS grid
- Use cms-produced **singularity** images
- Available on subMIT via **cvmfs**

**Enterprise linux 8 development environment**

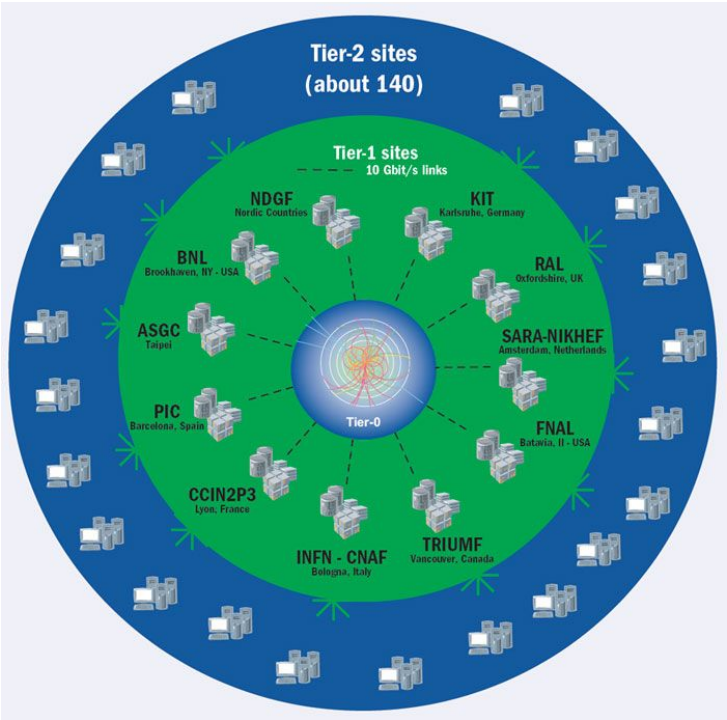
**Centos 7 development environment**

```
20 alias cmssw-el8='cmssource; APPTAINER_BIND="/home/submit,/work/submit,
   /data/submit,/scratch/submit,/cvmfs" cmssw-el8 --command-to-run bash'
21 alias cmssw-cc7='cmssource; APPTAINER_BIND="/home/submit,/work/submit,
   /data/submit,/scratch/submit,/cvmfs" cmssw-cc7 --command-to-run bash'
```

**In .bash_aliases**

# Submission to CMS grid resources

- CERN has huge amounts of cloud computing resources for analysis
- CMS has central tool for staging production to grid, called CRAB: (CMS Remote Analysis Builder)
- Allows you to send jobs to CMS grid
  - Automatically splits up dataset into individual jobs
  - Scales jobs out to all available resources
  - Automagically handles retries for common errors



# Crab from subMIT

- subMIT team has been very helpful and responsive in getting crab to work at subMIT
- Can efficiently run across whole datasets
- Allows rapid development cycles

```
Status on the CRAB server:      SUBMITTED
Task URL to use for HELP:       https://cmsweb.cern.ch/crabserver/ui/task/24
0131_170925%3Asrothman_crab_srothman_Jan31_2024_pythia_highstats_fixed_fixed
_2018_DYJetsToLL
Dashboard monitoring URL:       https://monit-grafana.cern.ch/d/cmsTMDetail/
cms-task-monitoring-task-view?orgId=11&var-user=srothman&var-task=240131_170
925%3Asrothman_crab_srothman_Jan31_2024_pythia_highstats_fixed_fixed_2018_DY
JetsToLL&from=1706717365000&to=now
Status on the scheduler:        FAILED

Jobs status:                    failed                   4.1% ( 38/922)
                                finished                95.9% (884/922)

No publication information (publication has been disabled in the CRAB config
uration file)

Error Summary: (use crab status --verboseErrors for details about the errors
)

 29 jobs failed with exit code 8021

  4 jobs failed with exit code 8002

  3 jobs failed with exit code 8028

  1 jobs failed with exit code 8006

Could not find exit code details for 1 jobs.

Summary of run jobs:
 * Memory: 1357MB min, 3099MB max, 2679MB ave
 * Runtime: 0:08:52 min, 3:26:25 max, 1:29:36 ave
 * CPU eff: 23% min, 95% max, 52% ave
 * Waste: 213:16:40 (13% of total)
```

# Distributed processing on slurm

# Local analysis on subMIT

- CMS grid processing not good enough for rapid development
  - Takes O(1 day)
  - Just reduces data size, doesn’t compute summaries
- Want to be able to run analysis quickly (~minutes)
- Preferably want to be able to do so in python (not c++)
- The problem:
  - Want to fill histograms in python
  - Want to be able to scale analysis to entire dataset (hundreds of large data files)
  - Want whole chain to run in O(minutes)

# Efficiently filling histograms in python

- Industry standard histograms library is boost.histogram
- Problem: it's in c++, and I hate c++
- Enter scikit-hep hist library
  - Python wrapper for boost.histogram
  - Can define (at runtime) arbitrary histograms
  - Efficient threaded fills

```
histogram = Hist(
    Variable(ptbins, name='pt', label = 'Jet $p_{T}$ [GeV]')
    Integer(0, nDR, name='dRbin', label = '$\Delta R$ bin',
            overflow=False, underflow=False),
    Variable(PUbins, name='nPU', label = 'Number of PU vertices',
             overflow=True, underflow=False),
    storage=Double()
)
```

```
    histogram.fill(
        pt     = squash(pt[mask2]),
        dRbin  = squash(dRbin[mask2]),
        nPU    = squash(nPU[mask2]),
        weight = squash(vals[mask2])
    )
```

# Scaling processing to entire dataset: coffea

coffea: magical package that handles:

- Splitting your dataset into discrete jobs
- Scaling out your analysis processing
- Combining resulting histograms into one result

# Scaling processing to entire dataset: coffea

coffea: magical package that handles:

- Splitting your dataset into discrete jobs
- Scaling out your analysis processing
- Combining resulting histograms into one result

```
class EECProcessor(processor.ProcessorABC):
    def __init__(self, config, statsplit=False):
        self.config = config
        self.statsplit = statsplit
```

How it works:

1. Define “processor” that fills analysis histograms

# Scaling processing to entire dataset: coffea

coffea: magical package that handles:

- Splitting your dataset into discrete jobs
- Scaling out your analysis processing
- Combining resulting histograms into one result

How it works:

1. Define “processor” that fills analysis histograms
2. Tell coffea how to scale out
   (eg iterative execution, multiprocessing, etc)

```
class EECProcessor(processor.ProcessorABC):
    def __init__(self, config, statsplit=False):
        self.config = config
        self.statsplit = statsplit
```

```
    runner = Runner(
        executor=FuturesExecutor(workers=4) if args.local_futures else
IterativeExecutor(),
        #executor=FuturesExecutor(workers=10, status=True),
        #chunksize=1000,
        schema=NanoAODSchema
    )
```

# Scaling processing to entire dataset: coffea

coffea: magical package that handles:

- Splitting your dataset into discrete jobs
- Scaling out your analysis processing
- Combining resulting histograms into one result

```
class EECProcessor(processor.ProcessorABC):
    def __init__(self, config, statsplit=False):
        self.config = config
        self.statsplit = statsplit
```

```
    runner = Runner(
        executor=FuturesExecutor(workers=4) if args.local_futures else
IterativeExecutor(),
        #executor=FuturesExecutor(workers=10, status=True),
        #chunksize=1000,
        schema=NanoAODSchema
    )
```

How it works:

1. Define “processor” that fills analysis histograms
2. Tell coffea how to scale out
   (eg iterative execution, multiprocessing, etc)
3. Profit

# Efficient distributed computing: dask

Would be nice if this also worked with HTCondor + MIT Tier2

- Final piece is to scale out across submit slurm cluster
- Want to be able to process ~100 million events in <10 minutes
- Solution: [dask-jobqueue]
  - Lets you point python at the whole slurm cluster
  - Automatically load balances by starting and killing slurm jobs as needed
- Integration with coffea is easy

```
  cluster = SLURMCluster(queue = 'submit-alma9',
                         cores=1,
                         processes=1,
                         memory='4GB',
                         walltime='01:00:00',
                         log_directory=log_directory)
  cluster.adapt(minimum_jobs = minjobs, maximum_jobs = maxjobs)
  client = Client(cluster)
runner = Runner(
    executor=DaskExecutor(client=client, status=True),
    #chunksize=100000,
    schema=NanoAODSchema
)
```

# Conclusions

- Big thanks to subMIT team for
  - Being very responsive
  - Helping me troubleshoot
  - Helping support tools such as CRAB
- subMIT provides useful gateway to global compute resources
- Dask, coffea provide access to scale python analysis across slurm
- It’s really fun to fully load a few hundred slurm jobs all at once