My thoughts on ATLAS analysis software and frameworks

Rustem Ospanov

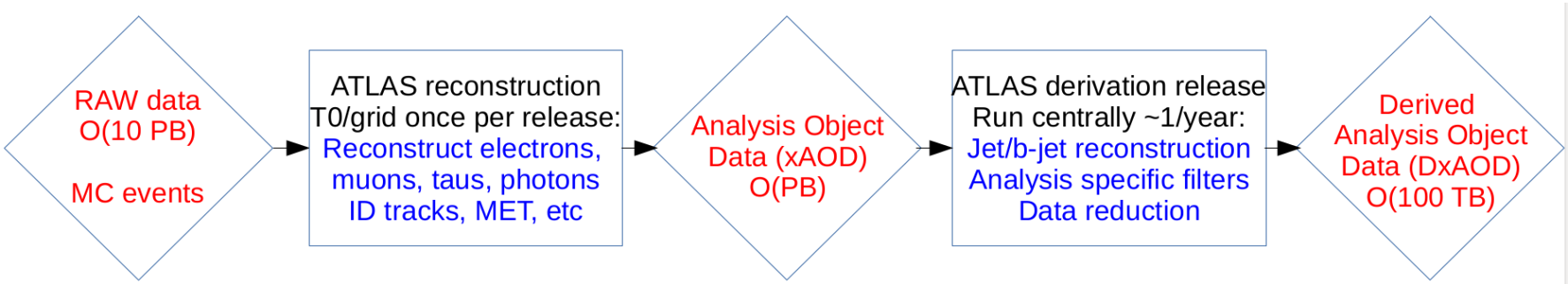
USTC

January 23, 2019

Introduction

- Share my own personal experiences with analysis software and frameworks
- Highlight approaches used by recent analyses with which I was involved:
  - $t\bar{t}H$ and same sign WW electroweak production
- Try to answer the questions posed and also share my own personal observations
  - ATLAS is large diverse collaboration that published more than 700 papers
  - There are many different approaches/views that are equally valid or better
  - By my estimate, there are $O(\mathrm{dozen})$ frameworks on ATLAS
- All opinions and mistakes in these slides are all my own

ATLAS reconstruction workflow

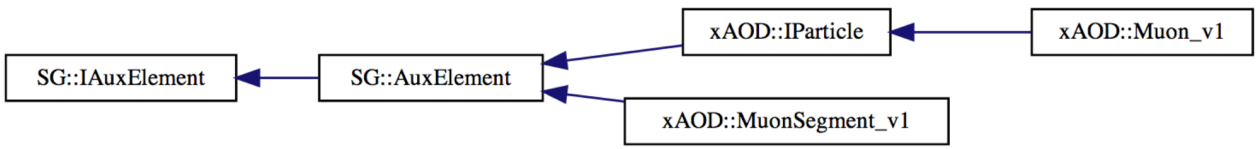


- Very simplified view of ATLAS reconstruction and derivation workflows
  - $O(300k)$ production jobs for simulation, reconstruction and derivations
  - 10-15 GB/s continuous traffic for data transfers
- Reconstruction xAOD and derived DxAOD use same object and event classes
  - xAOD and DxAOD each take of order 60 PB of grid disk - 120 PB total
- xAOD $\rightarrow$ DxAOD step is performed by central production system
  - Jet and b-jet reconstruction executed at derivations step to save disk
  - Process $O(1000)$ MC datasets - non-trivial bookkeeping

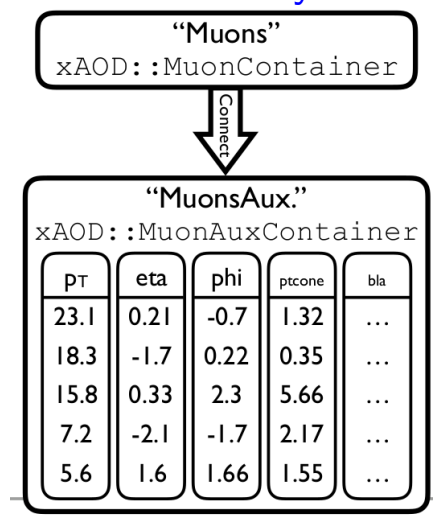
ATLAS Analysis Object Data (xAOD)

- Reconstruction xAOD and derived DxAOD use same object and event classes:
  - Major effort for Run 2 - C++ code for xAOD data classes
  - *Common interface for calibration and selection tools that operate on xAOD objects*
  - Variables stored as auxiliary data - cacheable string to value maps

Structure of xAOD::Muon class



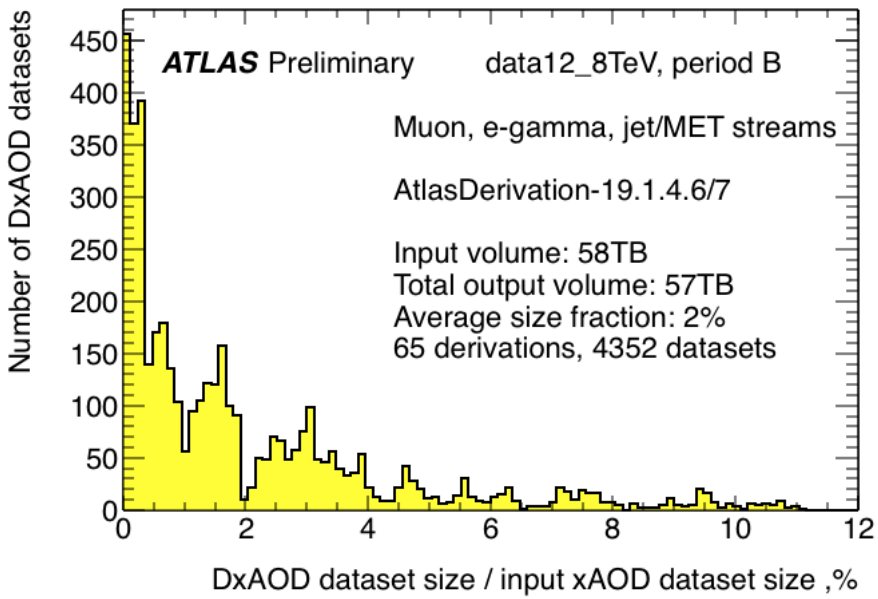
Muon auxiliary data

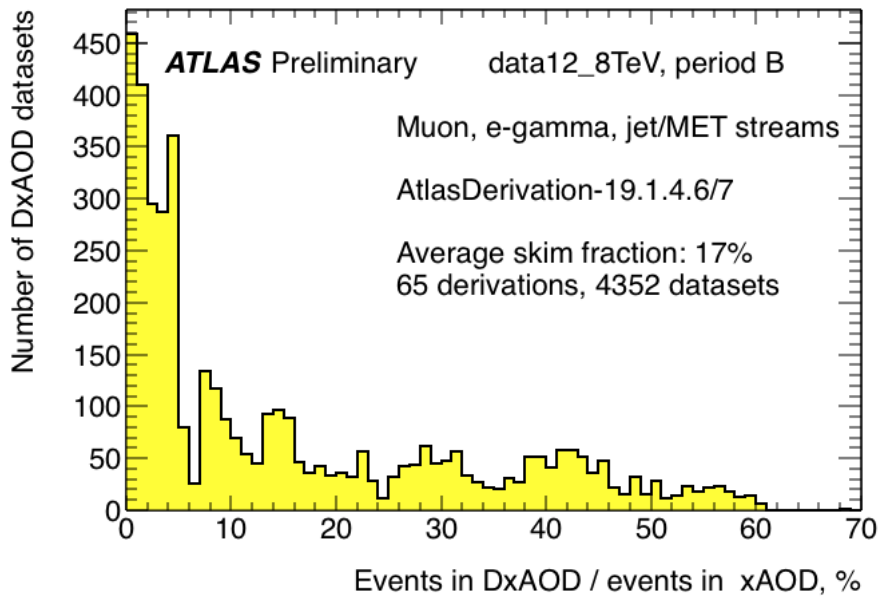


- User analysis starts from derived DxAOD datasets
  - Most analysis groups define own DxAOD filters - $O(\text{dozens})$ DxAOD formats
  - Example derivation di-lepton filter for SM group: STDM3.py

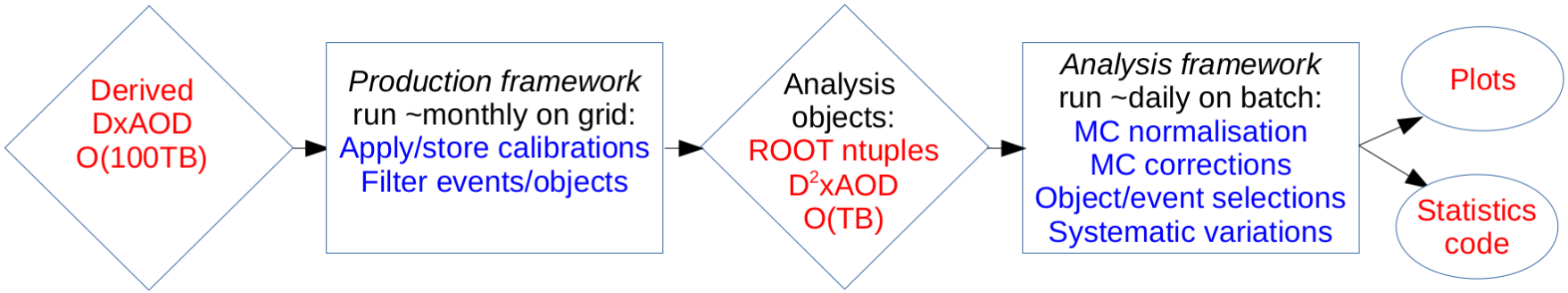
ATLAS DxAOD derivation sizes

- xAOD $\rightarrow$ DxAOD step is essentially $1 \rightarrow 1$
- Main advantage is faster processing times for reading DxAOD by analysis users





Example analysis workflow



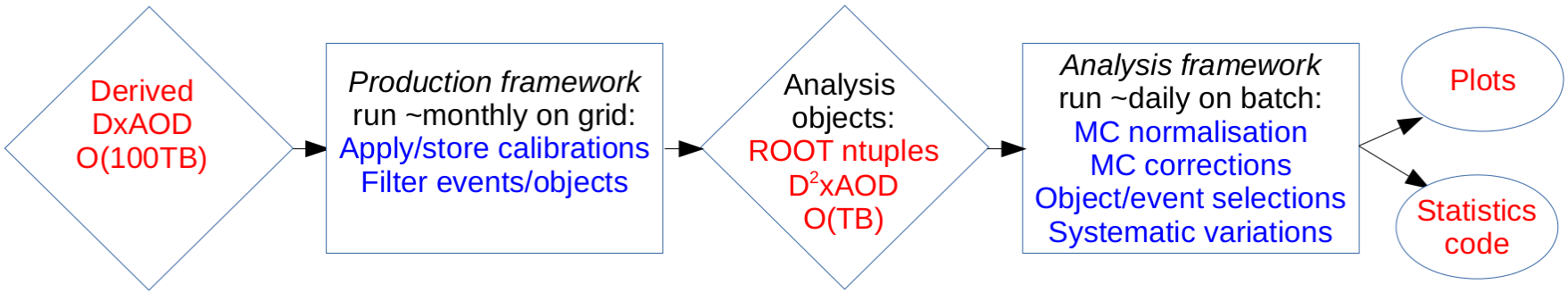
Use *production framework* to create intermediate data format:

- Athena based or standalone frameworks to process DxAOD objects
- Running over complete set of DxAOD datasets takes $O(\mathrm{days})$ on grid
- Output are usually ROOT ntuples or re-derived D^{2}xAOD
- *Dedicated software releases that fully specify calibration versions*

Use *analysis framework* for analysis work:

- Frameworks are usually C++ based using ATLAS analysis software releases
- Optimise analysis, make plots, perform studies, etc
- Process $O(100\mathrm{s})$ MC and data datasets - non-trivial bookkeeping
- Usually run on batch systems using order $O(100)$ processes
- Typical turnaround time of $O(\mathrm{hours})$ when using batch

Example analysis workflow: typical challenges



Typical challenges:

- Four vectors are are rarely if ever sufficient
- Correctly implementing object calibrations and systematic uncertainty - they change
- Correctly implementing object and event selections
- Bookkeeping for $O(100s)$ MC samples both for production and analysis runs
- Handling special cases, exceptions, crashes, unexpected results
- Analysers may require $O(\mathrm{months})$ to converge on stable/validated production+analysis setup for a new analysis

Analysis steps and evolution

Which kind of main tasks and operations do you perform on data and MC?

- Calibrate objects and compute correction factors with best guess selections
  - Often depend on optimised object/event selections - iterative process
- Estimate sources of background events
  - Data driven backgrounds require multiple passes over full data/MC - adds complexity
- Optimise selections to maximise signal sensitivity
  - Often depends on background estimation uncertainty - iterative process
- Validate/cross-check full analysis
  - Check data/MC agreements for many distributions in several control regions
  - Cross-check against similar analysis and/or with independent framework
- Produce inputs for statistical analysis
  - ROOT histograms or trees
  - Requires evaluation of full systematic uncertainty - usually done once
  - Statistical analysis is usually performed with another framework/scripts
- Produce publication quality plots
  - Plotting code is often included with the framework

How this change from day zero of preliminary analysis studies to last day before publication?

- I think procedures do not change much - just shift of focus/priority
  - One exception is systematic uncertainty and validation against another analysis

How do you deal with systematics?

- Evaluation of detector systematic uncertainty usually requires special production
  - Large set of variations for detector calibrations and corrections
  - Often sufficient to perform once when analysis matures and ready for approval
  - Can apply full analysis selections and reduce number of plots to speed up processing
  - Search-like analyses are usually limited by statistical, background or theory uncertainty
  - Precision analyses like Higgs/W/top mass are more sensitive to detector effects
- Evaluation of theory systematic uncertainty usually not as difficult
  - Run over dedicated systematic samples and/or apply additional weights
- Evaluation of systematic uncertainty for data driven backgrounds can be difficult
  - Typically requires good physics understanding of other background sources
  - MC can be unreliable or not have enough events
  - Flexible frameworks can be useful to perform dedicated checks and studies

Analysis interfaces, scaling, reusability, preservation and sharing

- Analysis interfaces
  - Personally, I prefer to use ATLAS *athena* framework with python configuration
  - Significant improvements in speed and usability for reading xAOD/DxAOD in Run 2
  - *EventLoop* is very popular with C++ executable - Analysis Software Tutorial
- Scaling
  - I find that batch resources are essential for analysis work
  - LXBatch is often very useful but limited by amount of user writable storage
  - Our university has T3 with 2000 slots and O(700 TB) storage
  - Allows fast turnaround: $O(\mathrm{day})$ from DxAOD to final plots - clear benefits
  - I tend to use interactive machines mostly for plotting, development and tests
- Reusability
  - Similar to CMS, heavy workload to maintain full featured framework
  - Frameworks (that I am familiar with) are usually used for many analyses
- Preservation and sharing
  - Personally, I feel that it is almost hopeless to reproduce full analysis much later
  - Requires detailed knowledge held only by people doing actual work - people move on

Missing functionality $\rightarrow$ my thoughts on desired framework criteria

My thoughts on desired framework criteria

Prioritised features for my ideal framework:

1. Results must be correct
2. Reliable, sensible and responsive developers
3. Transparency and readability
4. Ease of use and modularity
5. Design choices, speed and efficiency

Personal observations

- I find it extremely difficult to write code without bugs
  - It is never a question whether there are bugs or not
  - Main question seems to be how many and whether they are important
- Rely on usual approaches to code debugging and validation:
  - *Looking at plots and numbers catches most problems* - requires experience
  - Compare different frameworks - can be difficult and does not always converge
  - Use well maintained framework used by many groups - more eyes help
  - Run data/MC comparisons for standard processes like $t\bar{t}$ and $Z + jets$
- Experiments have robust review and validation processes
  - Serious bugs that significantly change results seem rare?
  - In my experience, code analysers for analysis code seem to be rare - too many hits
  - When happen, mistakes tend to be logic flaws rather than coding bugs
  - *Code transparency and readability always help*

Education and tutorials

- ATLAS has excellent software tutorial session
  - One week overview of ATLAS software with hands on sessions
  - Also introductions to popular frameworks and much more...
- How long does it take to learn C++?
  - https://isocpp.org/wiki/faq/big-picture#time-to-learn:
    "It takes 6-12 months to become broadly proficient in C++, especially if you havent done OO or generic programming before. It takes less time for developers who have easy access to a local body of experts, more if there isnt a good general purpose C++ class library available. To become one of these experts who can mentor others takes around 3 years."
- I think that reference framework(s) supported by the experiment are very useful
  - Clear design that follows best practices with tutorials and documentation
  - Start up kit that implements full analysis workflow (then also useful for cross checks)
  - Analysis framework developed by analysis team members should be allowed to get messy - no choice as this happens when a student hears "We need this plot tomorrow"
  - Making framework difficult to hack I think just makes hacks uglier
- Personally, I feel that we should make it higher priority to teach software skills
  - Requires resources and strong support by the management and community

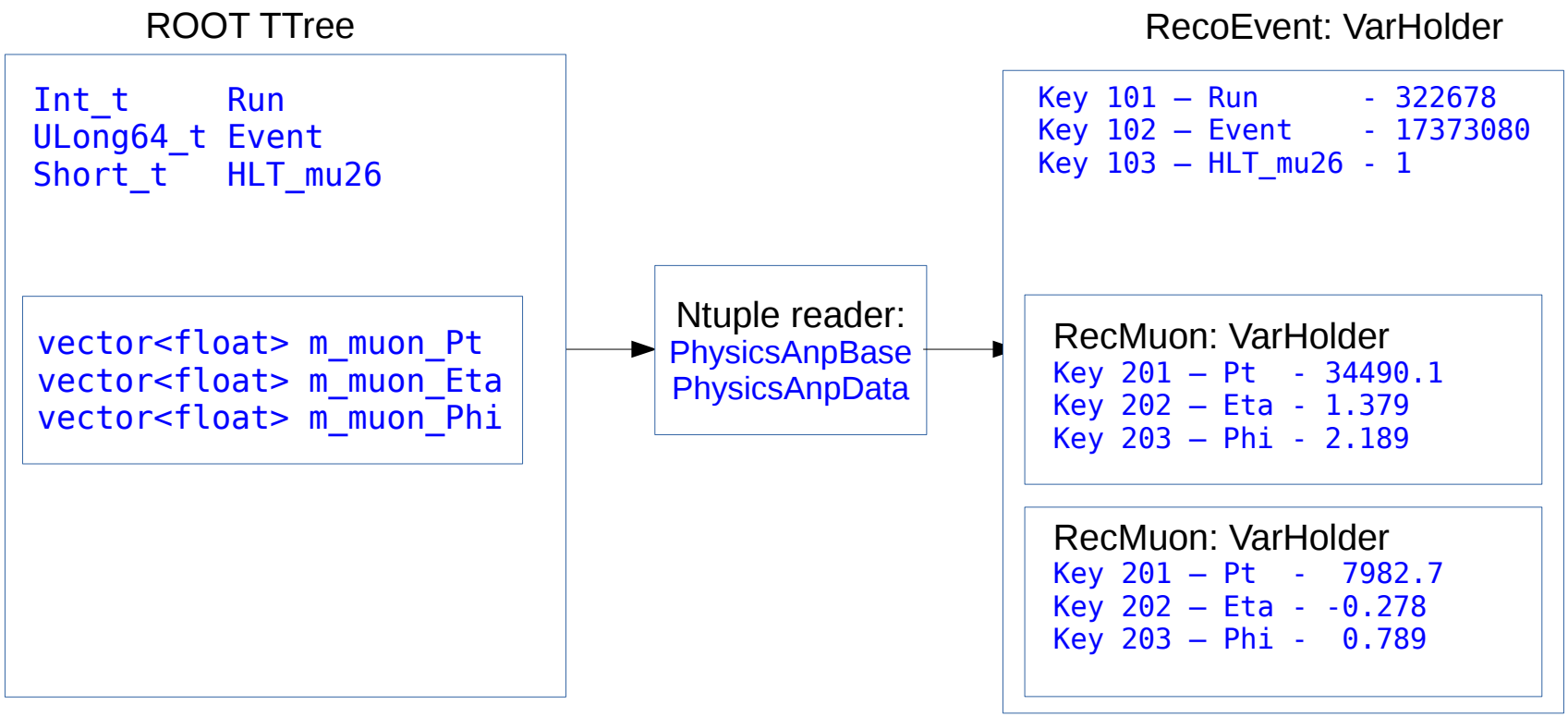
Analysis framework examples

ATLAS Common Analysis Framework (CAF)

- Full featured framework with dedicated core developers
  - C++ framework implements full analysis workflow from DxAOD to publication plots
  - Athena based design for Run 2 with python executables
  - Central Concepts of the CAF and code base
  - Developed and maintained by the Freiburg ATLAS group
- Used successfully for $H \to WW$ and many other ATLAS analyses
- In my opinion, key feature is use of text file configuration for cuts and plots
  - Allows arbitrary expressions for cuts and plots
  - Can combine dynamically stored variables with quantities computed by user code
  - Adding new cuts and plots can be done through configuration files
  - Use TTreeFormula to parse string variable names to computations
  - Plan to migrate to RDataFrame and CLING

My own framework for studies and development

- Read TTree with flat/vector branches to objects in memory
- Map variable names to integers at initialisation step
- Variables stored as (integer, double) pairs using VarHolder base class
- Use TFormula to parse cut/plot expressions and maps them to integers
- Use python to define cuts and xml to define histograms



Example of python configuration:

```
alg = AlgConfig('prepCand', 'PrepCand')
alg.SetKey('Debug',    'no')
alg.SetKey('SaveHist', 'yes')
alg.SetKey('HistKey',  'PlotMuon')

muon_cuts = [CutItem('CutPt',  'Pt  > 6.0e3'),
             CutItem('CutPID', 'Medium == 1')]

addCuts(alg, 'CutMuon', muon_cuts)
```

Corresponding C++ class definition:

```
namespace Anp
{
  class PrepCand: public virtual AlgEvent,
                  public virtual AlgBase
  {
  public:
     bool         fDebug;
     bool         fSaveHist;

     Ptr<CutFlow> fCutMuon;

     Ptr<HistKey> fHistMuon;
  };
}
```

C++ code that reads configuration:

```
void Anp::PrepCand::Config(const Registry &reg)
{
  reg.Get("Debug",         fDebug    = false);
  reg.Get("SaveHist",      fSaveHist = false);

  fCutMuon = BookCut("CutMuon", reg);

  fHistMuon = BookHist(reg);
}
```

C++ code that applies cuts and fills histograms:

```
for(Ptr<RecMuon> &muon: event.GetVec<RecMuon>()) {
  if(fCutMuon->PassCut(muon.ref())) {
    fHistMuon->FillHists(muon.ref(), 1.0);
  }
}
```

Example of python configuration:

```
alg = AlgConfig('prepCand', 'PrepCand')
alg.SetKey('Debug',    'no')
alg.SetKey('SaveHist', 'yes')
alg.SetKey('HistKey',  'PlotMuon')

muon_cuts = [CutItem('CutPt',  'Pt  > 6.0e3'),
             CutItem('CutPID', 'Medium == 1')]

addCuts(alg, 'CutMuon', muon_cuts)
```

Histogram definitions in PlotMuon.xml:

```
<histograms>

    <dirbase dirs="PlotMuon" />

    <hist key="Pt" name="" title="">
        <Xaxis bin="50" min="0.0" max="250.0e3" title="Muon p_{T} [MeV]" />
    </hist>

    <hist key="Eta" name="" title="">
        <Xaxis bin="60" min="-3.0" max="3.0" title="Muon #eta" />
    </hist>

</histograms>
```

The End

BACKUP

- Single athena based package built with AthAnalysisBase release
- Python code to configure tools and select output variables
- C++ readers to apply calibrations/pre-selections, writer to save ntuples

Configuration example for muon reader:

```
alg = CfgMgr.Ath__ReadMuon('readMuons')
alg.inputContainerName = 'Muons'
alg.outputVectorName   = 'm_muon_'
alg.configMuonCuts     = ['Pt > 5.0e3', 'Loose > 0']
alg.triggersHLT        = ['HLT_mu26_ivarmedium', 'HLT_mu50']
alg.auxVars            = ['MuonSpectrometerPt', 'InnerDetectorPt']
```

Configuration example for writer and defining output muon variables:

```
muon_vars = ['Pt                  :type=Float',
             'Eta                 :type=Float',
             'Phi                 :type=Float',
             'HLT_mu26_ivarmedium :type=Short',
             'HLT_mu50            :type=Short',
             'MuonSpectrometerPt  :type=Float',
             'InnerDetectorPt     :type=Float']

write_tool = CfgMgr.Ath__WriteEvent(name)
write_tool.branches = ['m_muon_|%s'  %(','.join(muon_vars))]
```