Systematics Handling at ATLAS

Nils Krumnack (Iowa State University)

Introduction

- ATLAS systematics model was developed during LS1
  - based on run 1 experiences
  - try to standardize+extend past mechanisms

- final corrections, scale factors, etc. are done via "CP tools"
  - encapsulate all of our "standard" recommendations
  - typically run during n-tuple production
  - implement all the systematics internally
  - provide standard interface for systematics handling

- ATLAS has many analysis frameworks
  - all build on the CP tools
  - each has its own systematics handling, etc.
- developed "common CP algorithms" as a common/shared solution
  - trying to implement/develop best practices
  - still in a (very drawn out) rollout

Custom Systematics

- do not have a fixed set of systematics everybody uses:
  - instead of just ±1σ variations can do ±5σ
    - reduces statistical jitter for small systematics
  - CP tools can have multiple systematics sets
    - e.g. simplified vs precision systematics
  - can vary multiple systematics at once
    - allows correlation studies
    - allows special systematics approaches (e.g. toys)
- not going to discuss this here further
  - that would be a talk in itself
  - not necessarily used by a lot of people either

- we don't know the systematics list until all tools are configured
  - plus whatever special systematics handling is needed
- mostly can ignore the details for this talk
  - mainly means that the systematics list can be very custom

CP Tools

- all CP tools have the same interface for systematics (in addition to their regular interface):

```
SystematicSet affectingSystematics () const;
SystematicSet recommendedSystematics () const;
StatusCode applySystematicVariation (const SystematicSet& sys);
```

- have both affecting and recommended systematics
  - some systematics are only for special use cases
  - actual list of systematics depends on the tool configuration
- systematics list is list of abstract systematic names
  - needs to be converted into ±1σ variations (or whatever desired)

- can ignore systematics interface when not using systematics
- with systematics, just set the systematics first:

```
tool->applySystematicVariation (sys);
tool->doSomething (...);
```

  - note that this is not reentrant…

First Approach

- original idea: would have global systematics loop on each event:

```
for (auto& sys : systematicsList) {
  for (auto& tool : toolList)
    tool->applySystematicVariation (sys);

  // do analysis for this event and systematic
}
```

- very simple and robust approach:
  - ensures all tools are configured consistently
  - very hard to get this wrong
- problem: redoes all work for every systematic
  - lose all information on what is affected by which systematic

- usually done in n-tuple maker
  - leads to creating a separate n-tuple for every systematic
  - most variables will be identical between systematics
  - huge waste of disk space (also slower to process)

Scale Factor Improvements

- some analysis frameworks try to optimize this further

- have separate loops just for scale factor (SF) systematics
  - SFs are directly added to the n-tuple
  - SF systematics only affect that one variable
  - make separate variables instead of separate n-tuples for SF sys

- not quite as feasible for momentum systematics
  - momentum changes propagate through downstream code
  - very tedious to do by hand

- some frameworks run scale factors on the n-tuple
  - can be faster than reading all SF variations from disk
  - requires "n-tuples" to be in xAOD format
  - requires special handling/reading code for n-tuple

CP Algorithms

- consolidation effort: common CP algorithms
  - each "CP algorithm" wraps a single CP tool
  - very modular design, multiple possibilities for optimization

- each algorithm contains its own systematics loop
  - allows to run the minimal set of systematics for each tool
  - only run tool for systematics that affect it, i.e.
    - systematics that are implemented by the tool itself
    - systematics that affect any of the tool's inputs
  - all systematics for that algorithm count as affecting all its outputs
  - seems like the most aggressive optimization we can safely do automatically

- systematics bookkeeping and event store access via "data handles"
  - separates systematics handling from algorithmic code
  - allowed for multiple rewrites of the systematics handling

Adding Systematics

```
MuonCalibAlg::execute () {


  xAOD::MuonContainer *muons = nullptr;
  evtStore().retrieve (muons, muonName);
  for (xAOD::Muon *muon : *muons)
    calibTool->applyCorrection (*muon);

}
```

no-sys

```
MuonCalibAlg::execute () {
  for (auto& sys : sysList.systematicsVector()) {
    calibTool->applySystematicVariation (sys);
    xAOD::MuonContainer *muons = nullptr;
    muonHandle.getCopy (muons, sys);
    for (xAOD::Muon *muon : *muons)
      calibTool->applyCorrection (*muon);
  }
}
```

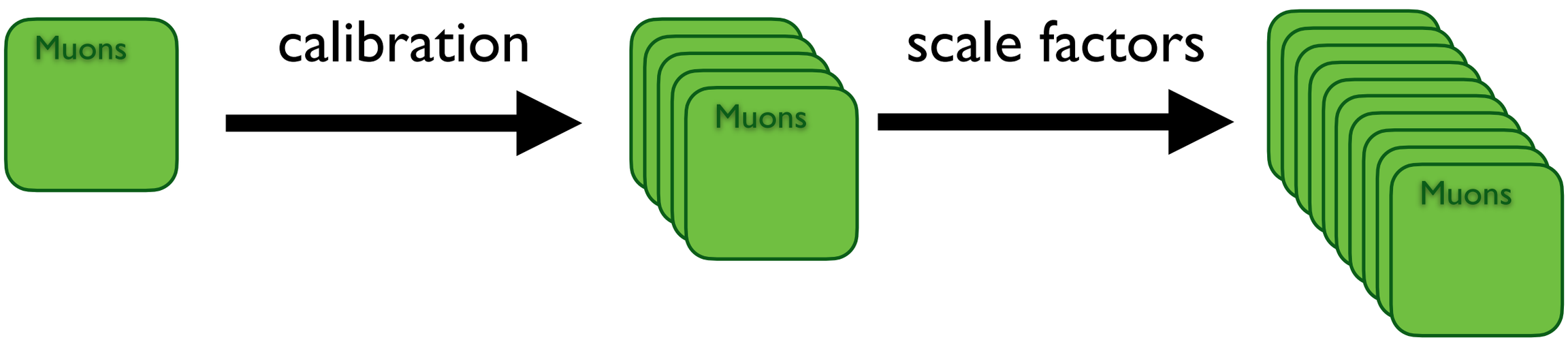
with-sys

- simplified code (no return code checking, etc.)
- main differences:
  - added a loop over systematics
  - access muon container via data handle
- also extra initialization code, extra data members in class
- also extra properties for use during configuration
  - usually set by special configuration handlers
- overall very similar, changes very straightforward

Shallow Copies

- xAOD EDM has shallow copy feature:
  - copies share data members that exist at time of copy
  - new data members get added only to the copy

- original idea: add shallow copies for each systematic
  - each algorithm with systematics add a new set of copies
  - tracking systematics per object
  - algorithms in the middle produce temporary copies



Shallow Copy Problems

- problem: tracking per object is not granular enough
  - most algorithms just add/change a single variable on each object
  - subsequent algorithms may not use that variable
    - per-object tracking still forces them to use all its systematics
  - can address some issues by ordering, but not all

- also: don't know if a given variable was created before or after adding a given systematic
  - can lead to unnecessary duplication of variables in n-tuple
  - generally needs some workarounds

- particular problems:
  - SFs are only added to objects to add them to the n-tuple
  - overlap removal gets affected by all object systematics and propagates them to all objects
  - global information is attached to EventInfo, read in many places

Decoration Systematics

- allow to add systematically varied decorations/variables to objects
  - just distinguished by name of the decoration
  - decoration systematics tracked in addition to object systematics
  - provides extra information to use when creating n-tuples

- still working on incorporating it everywhere
  - easy for decorations that are read directly/stored in n-tuple
  - selection decorations have an entire infrastructure
    ⇒needs more work to change

- caveat: can't do this for all variables
  - some variables are build into the EDM/tools
  - need to still use shallow copies for those
  - not necessarily bad: it's mostly four-momentum, etc. which are anyways used by most/all tools

Towards Run 3/4

- new common data format with all corrections included: PHYSLITE
  - still under development
  - does not include systematics
  - some features like MET depend on user selection, not included

- still need to run CP tools to generate systematics, MET, etc.
  - can run current tool chain on PHYSLITE
  - will typically generate a new n-tuple (like current workflow)

- may include some of the missing information in PHYSLITE
  - at least enough to allow studies without using tool chain

- hope to allow running tool chain on the fly
  - ought to be faster: read 3-4 variables to calculate N systematics
  - need infrastructure to run inside pyroot, etc.
  - need work to ensure this is performant

Summary

- ATLAS has a fairly complex systematics model
  - allows a fair amount of user customization
  - multiple analysis frameworks mean multiple handling strategies
  - (slow) migration to a common framework

- very aggressive on optimizations
  - try to save CPU time and disk space
  - need to track per variable dependencies
  - need to track per variable systematics

- factored out systematics handling into separate subsystem
  - separates it (mostly) from the algorithmic code
  - allowed for massive reworks of the systematics handling