xAOD Analysis Introduction

Nils Krumnack (Iowa State University)

Introduction

- learn today how to set up a basic analysis package
  - set up a basic analysis "Algorithm" (see later slide)
  - run over a single xAOD file
  - do some common operations/tasks

- goal is to teach you best practices
  - often there are alternative ways of doing things
  - today's tools are (mostly) dual-use (see next slide)
  - integrate well into overall ATLAS software environment
  - use (best) ATLAS tools available
    - more robust, more scalable, easier to learn…
  - should work for several years as is

- these are the basics for any kind of ATLAS software work
  - multiple other sessions build on this
  - come back to this whenever you need to set up new package

Analysis Environments

- most ATLAS software based on Athena framework
  - used in the production system, online system, etc.
  - allows access to detector geometry, conditions data, etc.
  - very complete feature set
  - scales up to production/reconstruction level tasks

- early(!) Athena did very poorly for analysis
  - not popular except with Athena/software experts
  - end user analysis is rather different from offline production

- developed set of tools aimed just at analysis: AnalysisBase
  - also called (not all accurate): stand-alone, post-Athena, root-based, EventLoop, RootCore (outdated)
  - evolved into a fairly complete software framework
  - missing some advanced features, e.g. conditions access
  - used by large majority of analysis users

Environment Choice

- which should you use: EventLoop or Athena?
  - not going to tell you to use one or the other
  - little difference for material show-cased today
  - getting ever easier to move code between environments

- often not a technical choice
  - projects you join often require specific framework
  - better to use whatever the person at the next desk uses
    - often your best source of support
  - many people have a strong personal preference

- for today's tutorial should do both
  - first do one, then the other
  - makes you familiar with both environments
  - shows you similarities between both environments

Analysis Releases I

- analysis releases in a nutshell:
  - tool developers tell us whenever there is a new tool version
  - gets reviewed and then added to nightly build
  - once there are enough updates we build a numbered release
  - numbered releases frozen and kept "forever"

- can build a numbered release when you ask for it
  - e.g. before a major production run

- analysis releases distributed via cmvfs (including nightly builds)
  - can use on any compatible machine with cvmfs
  - distributed to all ATLAS sites
- you can build analysis release yourself locally
  - should work on most Linux flavors and MacOS
  - works without cvmfs → downloads calibrations to disk
  - needs network connection on first run, network free afterwards

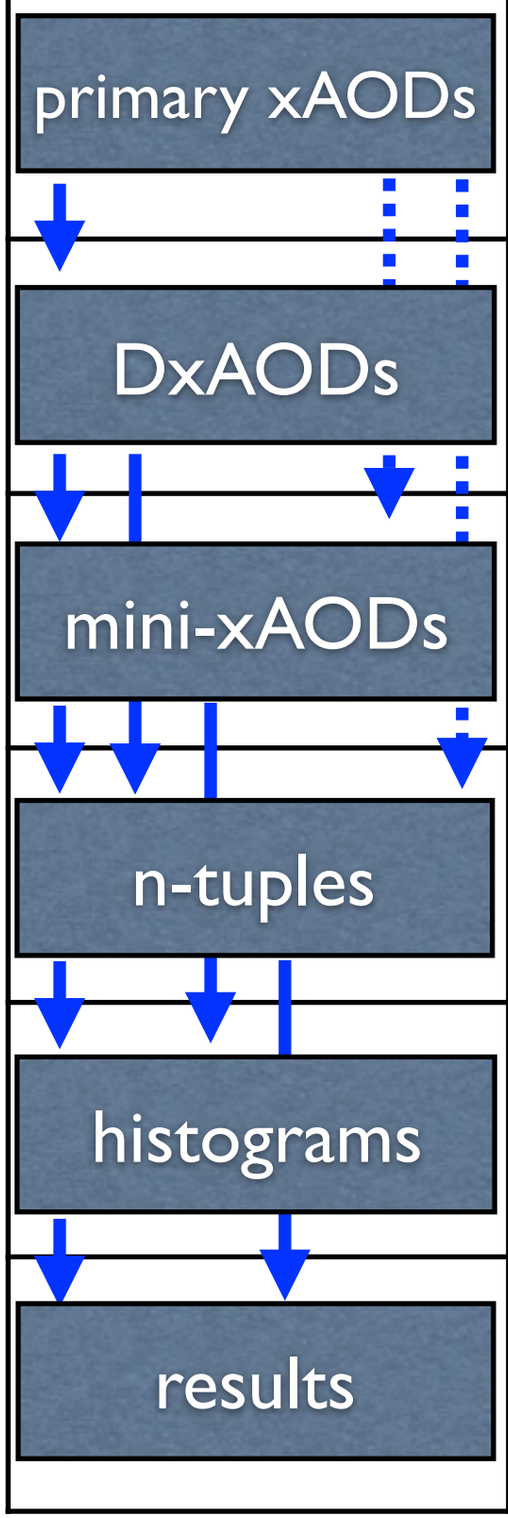
Analysis Releases II

- have a base release everybody can use: AnalysisBase/AthAnalysis
  - contains all the CP tools in their recommended versions
  - contains ASG root analysis tools in their recommended versions
  - saves users from worrying about correct tool versions
  - users have single release number to document tool versions

- several release coordinators (lead: Andy Mehta)
- package authors should request new versions via merge requests
  - see git tutorial: https://atlassoftwaredocs.web.cern.ch/gittutorial/
- new releases are announced at atlas-sw-pat-releaseannounce
  - this is low volume, so everybody should subscribe

- main version used now:
  - 21.2.X:, files produced with release 21, cmake ← shown today
- https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/AnalysisBase

Run 2 Workflow (typical)

<table>
<tr><td rowspan="5">  </td><td>produced centrally, stored on grid</td><td>single definition</td><td rowspan="3">xAOD Format</td><td>no corrections</td></tr>
<tr><td></td><td>defined by physics group</td><td>only AODFix</td></tr>
<tr><td rowspan="2">produced by group/analyzer, stored locally</td><td rowspan="2">defined by group/analyzer</td><td>some/all corrections</td></tr>
<tr><td>TTree</td><td rowspan="3">all corrections</td></tr>
<tr><td>produced by analyzer, stored locally or personal machine</td><td>very analysis dependent</td><td>(mostly) TH1</td></tr>
<tr><td></td><td></td><td>analysis dependent</td></tr>
</table>

Run 2 Workflow (typical)

primary xAODs	produced centrally, stored on grid	single definition	xAOD Format	no corrections
DxAODs		defined by		only AODFix
mini-xAODs	produ group/ stored			some/all corrections
n-tuples				
histograms	produced by analyzer, stored locally or personal machine	very analysis dependent	(mostly) TH1	all corrections
results			analysis dependent	

Your analysis may deviate from this!

This merely reflects how most people do things…

Run 2 Workflow (typical)

primary xAODs	produced centrally, stored on grid	single definition	xAOD Format	no corrections
DxAODs		defined by physics group		only AODFix
mini-xAODs				some/all corrections
n-tuples		by analyzer	TTree	all corrections
histograms	produced by analyzer, stored locally or personal machine	very analysis dependent	(mostly) TH1	
results			analysis dependent	

Aiming at all these stages today

xAOD Format/Classes

- xAODs are covered in almost every session of the tutorial
  - EDM used for Athena and AnalysisBase analysis
  - it's a file format, plus software to access it
    - all your input data is in this format
    - most ATLAS tools run on xAOD objects
  - some nifty extra features (decorations, shallow copies, etc.)

- xAODs should be as fast/small as "flat" n-tuples
  - often faster, if your n-tuple code is not optimal
  - assuming you have the same number of objects, etc.

- you can't modify objects read from the file:
  - safety feature, so you don't confuse yourself
  - need to make (shallow) copy before applying CP tools

Basic xAOD Reading

- can read an xAOD without any framework
  - have TEvent+TStore classes for this
  - still need to use an analysis release

- can also read xAOD without any release at all
  - can open it like a regular root TTree
  - missing out on some functionality (e.g. ElementLink)

- can be better for small/test jobs, i.e.:
  - using just a couple lines of code, not expected to grow
  - only run on a single file or two, no distributed/batch running

- rarely "wrong" to write a "proper" framework job though:
  - EventLoop as fast as a manual loop
  - scales up to more code, more files, etc.
  - need extra "administrative" code though

Basic xAOD Reading Code

```
#include <memory>
#include <iostream>
#include <TFile.h>
#include “AsgTools/MessageCheck.h”
#include “xAODRootAccess/Init.h”
#include “xAODRootAccess/TEvent.h”
#include “xAODEventInfo/EventInfo.h”

int main() {
  // Set up application & “checker code”:
  using namespace asg::msgUserCode;
  ANA_CHECK_SET_TYPE( int );
  ANA_CHECK( xAOD::Init() );

  // Open an input file:
  xAOD::TEvent event;
  std::unique_ptr< TFile > ifile( TFile::Open( “AOD.pool.root”, “READ” ) );
  ANA_CHECK( event.readFrom( ifile.get() ) );

  unsigned count = 0;

  // Loop over the events:
  const Long64_t numEntries = event.getEntries();
  for( Long64_t i = 0; i <numEntries; ++i ) {
    // Load the event:
   event.getEntry( i );

    // Load xAOD::EventInfo:
    const xAOD::EventInfo* ei = nullptr;
    ANA_CHECK( event.retrieve( ei, “EventInfo” ) );
    std::cout << “Processing run #” << ei->runNumber() << “, event #” << ei->eventNumber() << std::endl;
    count += 1;
  }
  std::cout << "processed " << count << " events" << std::endl;
  return 0;
}
```

Basic Framework Jobs

- frameworks help with
  - running distributed/batch jobs and collecting results
  - assembling more complicated jobs
  - integrating code developed separately
  - reconfiguring jobs without changing C++ code
  - plenty of "advanced" features

- use same formalisms as full-scale online/production jobs
  - mostly a configuration & release change
  - can (fairly) easily share code between environments
  - central code has stricter design requirements than analysis
- not focusing on anything here not needed/useful for analysis

- to use a framework you need to
  - provide an algorithm to run (or several algorithms)
  - provide a (basic) configuration file

Algorithm Basics

- with a framework you no longer can/should
  - write your own main() function
  - write a loop over all events
  - create your own output files
    - access them through the framework instead

- instead: need to provide code in form of algorithms
  - provides a standard way for framework to run your code
  - structurally quite similar to standalone macro though

- no need to go fancy:
  - most users have just one algorithm
  - most users don't have a lot of configurable properties
    - configuration is less important when you are the only user
  - don't write code you barely comprehend
    - you'll have trouble debugging it!!

Basic xAOD Reading

```
#include <memory>
#include <iostream>
#include <TFile.h>
#include “AsgTools/MessageCheck.h”
#include “xAODRootAccess/Init.h”
#include “xAODRootAccess/TEvent.h”
#include “xAODEventInfo/EventInfo.h”

int main() {
  // Set up application & “checker code”:
  using namespace asg::msgUserCode;
  ANA_CHECK_SET_TYPE( int );
  ANA_CHECK( xAOD::Init() );

  // Open an input file:
  xAOD::TEvent event;
  std::unique_ptr< TFile > ifile( TFile::Open( “AOD.pool.root”, “READ” ) );
  ANA_CHECK( event.readFrom( ifile.get() ) );

  unsigned count = 0;

  // Loop over the events:
  const Long64_t numEntries = event.getEntries();
  for( Long64_t i = 0; i <numEntries; ++i ) {
    // Load the event:
   event.getEntry( i );

    // Load xAOD::EventInfo:
    const xAOD::EventInfo* ei = nullptr;
    ANA_CHECK( event.retrieve( ei, “EventInfo” ) );
    std::cout << “Processing run #” << ei->runNumber() << “, event #” << ei->eventNumber() << std::endl;
    count += 1;
  }
  std::cout << "processed " << count << " events" << std::endl;
  return 0;
}
```

Basic xAOD Reading

```
#include <memory>
#include <iostream>
#include <TFile.h>
#include “AsgTools/MessageCheck.h”
#include “xAODRootAccess/Init.h”
#include “xAODRootAccess/TEvent.h”
#include “xAODEventInfo/EventInfo.h”

int main() {
  // Set up application & “checker code”:
  using namespace asg::msgUserCode;
  ANA_CHECK_SET_TYPE( int );
  ANA_CHECK( xAOD::Init() );

  // Open an input file:
  xAOD::TEvent event;
  std::unique_ptr< TFile > ifile( TFile::Open( “AOD.pool.root”, “READ” ) );
  ANA_CHECK( event.readFrom( ifile.get() ) );

  unsigned count = 0;

  // Loop over the events:
  const Long64_t numEntries = event.getEntries();
  for( Long64_t i = 0; i <numEntries; ++i ) {
    // Load the event:
   event.getEntry( i );

    // Load xAOD::EventInfo:
    const xAOD::EventInfo* ei = nullptr;
    ANA_CHECK( event.retrieve( ei, “EventInfo” ) );
    std::cout << “Processing run #” << ei->runNumber() << “, event #” << ei->eventNumber() << std::endl;
    count += 1;
  }
  std::cout << "processed " << count << " events" << std::endl;
  return 0;
}
```

done by framework

Basic xAOD Reading

```
#include <memory>
#include <iostream>
#include <TFile.h>
#include “AsgTools/MessageCheck.h”
#include “xAODRootAccess/Init.h”
#include “xAODRootAccess/TEvent.h”
#include “xAODEventInfo/EventInfo.h”

int main() {
  // Set up application & “checker code”:
  using namespace asg::msgUserCode;
  ANA_CHECK_SET_TYPE( int );
  ANA_CHECK( xAOD::Init() );

  // Open an input file:
  xAOD::TEvent event;
  std::unique_ptr< TFile > ifile( TFile::Open( “AOD.pool.root”, “READ” ) );
  ANA_CHECK( event.readFrom( ifile.get() ) );

  unsigned count = 0;

  // Loop over the events:
  const Long64_t numEntries = event.getEntries();
  for( Long64_t i = 0; i <numEntries; ++i ) {
    // Load the event:
   event.getEntry( i );

    // Load xAOD::EventInfo:
    const xAOD::EventInfo* ei = nullptr;
    ANA_CHECK( event.retrieve( ei, “EventInfo” ) );
    std::cout << “Processing run #” << ei->runNumber() << “, event #” << ei->eventNumber() << std::endl;
    count += 1;
  }
  std::cout << "processed " << count << " events" << std::endl;
  return 0;
}
```

initial actions

executed on each event

final actions

Basic Algorithm

```
...
StatusCode MyAlgorithm :: initialize () {
  m_count = 0;
  return StatusCode::SUCCESS;
}
```

initial actions

executed on each event

```
StatusCode MyAlgorithm :: execute () {
   // Load xAOD::EventInfo:
    const xAOD::EventInfo* ei = nullptr;
    ANA_CHECK( evtStore()->retrieve( ei, "EventInfo" ) );
    ANA_MSG_INFO ("Processing run #" << ei->runNumber() << ", event #" << ei->eventNumber());
    m_count += 1;
   return StatusCode::SUCCESS;
}
```

```
StatusCode MyAlgorithm :: finalize () {
  ANA_MSG_INFO ("processed " << m_count << " events");
  return StatusCode::SUCCESS;
}
```

final actions

- basic structure stays the same
  - each code block becomes its own function
  - (some) data stored in class member variables (e.g. m_count)
- some minor changes in conventions (e.g. messaging)
- all functions return status codes (see next slide)
- more example than useful algorithm

Status Codes

- most ATLAS code returns status codes to indicate failures
  - used instead of exceptions
  - need to check all status codes

- helper for status codes: ANA_CHECK() macro
  - it works like CHECK(), ATH_CHECK(), etc.
  - however: ANA_CHECK works with any status code (and some other types as well)
- you can call

```
ANA_CHECK (someFunction(x,y,z));
```

and it will abort the current function if the call fails

- some more advanced features of ANA_CHECK():
  - you can change the return type
  - you can use it in non-member functions

Hands-On Instructions

- only work through first three sections today
  - that's the introductory & dual-use sections
  - future sessions will build on these

- do first two sections in AnalysisBase/EventLoop
  - then log in again, repeat in AthAnalysis/Athena
  - keep the source area, make new build area
- continue to third section in EventLoop or Athena

- have done this tutorial a few times so far
  - still fixing some issues with each iteration
  - keep adding/redesigning material (and breaking things…)
  - hopefully it works reasonably well today
  - hopefully enough time to finish…

- https://atlassoftwaredocs.web.cern.ch/ABtutorial/

backup slides

Algorithms & Tools

- an algorithm is a piece of code that executes on each event
  - most of your analysis code will go into an algorithm
  - makes it easy for EventLoop/Athena to run your code

- a tool is a piece of code you use in your algorithm
  - e.g. CP recommendations are distributed as tools
  - no need to write your own tools (though you can)
  - provide a uniform way of configuring code

- each tool provides a C++ interface to access functionality
  - algorithm needs to call interface methods
    - tool (normally) won't do anything unless called
  - generally pass in the object you want the tool to work on

- there is a bigger component model behind this
  - will hear more about this in framework session (Thursday)

Tool Configuration/Use

- define ToolHandle in algorithm class:

```
ToolHandle<IMyTool> m_tool {"MyTool/name", this};
```

- declare ToolHandle as property in algorithm constructor:

```
declareProperty ("tool", m_tool, "tool for doing xyz");
```

- retrieve tool in initialize (optional, but recommended):

```
ANA_CHECK (m_tool.retrieve());
```

- call tool in execute():

```
m_tool->doSomething (...);
```

- then in your python configuration:

```
addPrivateTool (alg, "tool", "MyTool")
alg.tool.property = value
```

Analysis Releases II

- motivation I: analysis releases provide a fixed version of code
  - means you don't have to worry about whether you are using the right version of each tool (managed centrally)
  - also: you have a single release number to document what you did

- motivation II: grid submission is a lot faster
  - since the releases are available on the grid, there is no need to copy them from the local area
  - can go from half an hour to half a minute(!) submission time

- motivation III: saves compilation time
  - analysis release is over a hundred packages
  - compiling them can take hours (depending on your machine)
  - setting up an analysis release takes only seconds

xAODs in AnalysisBase

- have two classes for accessing/storing xAOD objects:
  - TEvent: for reading/writing objects from/to files
  - TStore: for storing objects you create
- you typically need to create both of these:
  - don't forget to clear the TStore on every event
  - in EventLoop that is taken care of for you

- you can't modify objects read from the file:
  - safety feature, so you don't confuse yourself
  - need to make (shallow) copy before applying CP tools

- in AnalysisBase you need to match releases to file versions:
  - e.g. release 21.2.* reads offline release 21.2 files
- AnalysisBase doesn't apply AODFix (i.e. some basic corrections):
  - to get this, you need to run on derivations (or in Athena)
- files written in AnalysisBase can't be read in Athena

CP Tool Interface Overview

- CP groups distribute their recommendations as tools
  - plan to switch to CP algorithms in the future (Thursday session)

- (most) inputs/outputs via xAOD EDM objects
  - means fewer interface changes
  - makes interfaces simpler
  - utilizes that one can add new fields to xAOD objects

- configuration happens via tool properties
  - follows the mechanism used in Athena
- no real configuration service in AnalysisBase
  - old mechanism: AnaToolHandle for dual-use code instead
  - new mechanism: uniform configuration for tools & algorithms
- most tools do something meaningful with default configuration
  - but not necessarily what you want
  - **always** check the recommendation

Tool-Kit Philosophy

- aim for set of independent tools/packages instead of framework
  - i.e. fairly easy to use one software package without the others
  - lesson from history: using Athena was/is mostly all-or-nothing

- however: please try to use the common tools where possible
  - a lot of experience went into them
  - most are well-tested with large user base
    - fewer bugs, easier to find help
  - often have features that are useful later on
  - easier to exchange code with others

- still, feel free to replace common tools with your own, but:
  - please try the common tools first
  - ask an expert whether the feature you want is already there
    - or whether we can implement it as a new feature
  - consider contributing to the common tools

Dual-Use Philosophy

- put strong emphasis on dual-use code:
  - allow running analysis code inside Athena unchanged
  - only a question of recompiling in the new environment

- original motivation: allow running CP tools in Athena
  - many CP groups only released code for AnalysisBase in run 1
  - some tools are also used/needed for offline/online
  - also: some people huge fans of doing analysis in Athena

- side benefit: learn Athena basics/concepts without using Athena
  - main difference: job configuration/management
  - some things not (yet) dual-use: services, algorithm sequences…

- side benefit: can develop code in preferred environment
  - e.g. even people who know Athena often prefer AnalysisBase

Analysis in Athena

- have a version of Athena for analysis: AthAnalysis
  - becoming more similar to AnalysisBase over time
  - getting close to EventLoop performance (maybe 10-20% slower)
  - has capabilities not in AnalysisBase, e.g. conditions access
  - can definitely do a complete analysis with it

- missing some of the advanced/niche features of AnalysisBase, e.g.:
  - running on MacOS without VM/docker
  - use SampleHandler+PlotMaker for quickly making stack plots
  - dataset discovery with SampleHandler

- Athena/Gaudi more dismissive of analysis feature requests
  - analysis not their main/only use case
  - not a problem if you do things the "Athena way"

- (biased) sales pitch for AthAnalysis tomorrow

SampleHandler

- it is not uncommon to need ~100 datasets for a single analysis
  - can be difficult to track them all
  - also need to track associated metadata, e.g. luminosity, k-factor…
- SampleHandler can do all that bookkeeping for you
  - can store the location and metadata for each dataset
  - can discover available datasets for you
  - can handle different storage systems (including the grid)
  - can group and select data samples in various ways
- some notes:
  - if SH doesn't match your workflow or setup, you can/should ask for an adapter
  - SH doesn't do magic, you still need to define your own metadata
  - most people only use SH to pass file lists to EventLoop

- https://twiki.cern.ch/twiki/bin/view/AtlasProtected/SampleHandler

EventLoop

- a typical analysis often uses a large quantity of data
  - looping over events often slow → parallel processing advised
- EventLoop implements an event loop for you
  - works for local running, batch systems and the grid
  - handles a lot of submission and merging details for you

- easy to use when starting a new analysis
  - if you have an existing analysis it is probably not worth the effort to switch to EventLoop, unless you need its features
- basic EventLoop concepts (examples in the hands-on section):
  - have to write code in the form of algorithms
  - jobs are configured through a job object
  - the location where you run is configured through a driver object

- https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/EventLoop

AnaToolHandle

- allows you to instantiate tools uniformly (old approach)
  - works identically in AnalysisBase and AthAnalysis
  - separates user from implementation details

- basic usage:

```
AnaToolHandle<IMyTool> tool ("MyTool/name");
ANA_CHECK (tool.setProperty ("name", value));
ANA_CHECK (tool.initialize());
tool->call (...);
```

AnaToolHandle

- allows you to instantiate tools uniformly (old approach)
  - works identically in AnalysisBase and AthAnalysis
  - separates user from implementation details

- basic usage:

```
AnaToolHandle<IMyTool> tool ("MyTool/name");
ANA_CHECK (tool.setProperty ("name", value));
ANA_CHECK (tool.initialize());
tool->call (...);
```

behaves like ToolHandle/pointer

behaves like athena/python configurable

AnaToolHandle

- allows you to instantiate tools uniformly (old approach)
  - works identically in AnalysisBase and AthAnalysis
  - separates user from implementation details

- basic usage:

```
AnaToolHandle<IMyTool> tool ("MyTool/name");
ANA_CHECK (tool.setProperty ("name", value));
ANA_CHECK (tool.initialize());
tool->call (...);
```

interface class

implementation class

CP Tool Interface Overview

- (most) inputs/outputs via xAOD EDM objects
  - means fewer interface changes
  - makes interfaces simpler
  - utilizes that one can add new fields to xAOD objects

- configuration happens via tool properties
  - follows the mechanism used in Athena
- no configuration service in AnalysisBase
  - need to set properties manually from C++/python
- most tools do something meaningful with default configuration
  - but not necessarily what you want

- unified systematics interface (see later slides)

- https://cds.cern.ch/record/1639568/files/ATL-COM-SOFT-2013-125.pdf
- https://cds.cern.ch/record/1667206/files/ATL-COM-SOFT-2014-005.pdf

Tools Presented Today

- cmake: build system for building your code
  - replaces previously used RootCore system
- SampleHandler+EventLoop: dataset & job management
- xAOD EDM: format for (input) event data
- analysis releases: collection of tools for analysis
- systematics handling: unified mechanism for all CP tools
- dual-use mechanisms: makes AnalysisBase & Athena more similar

- will also present a couple of CP tools on xAODs:
  - CP tools are how CP groups distribute their code
  - CP tools (should) conform to common interface

- version management not covered today:
  - it's mostly orthogonal to what we do today
  - you should definitely use one
    - any system is fine (git is somewhat preferred)

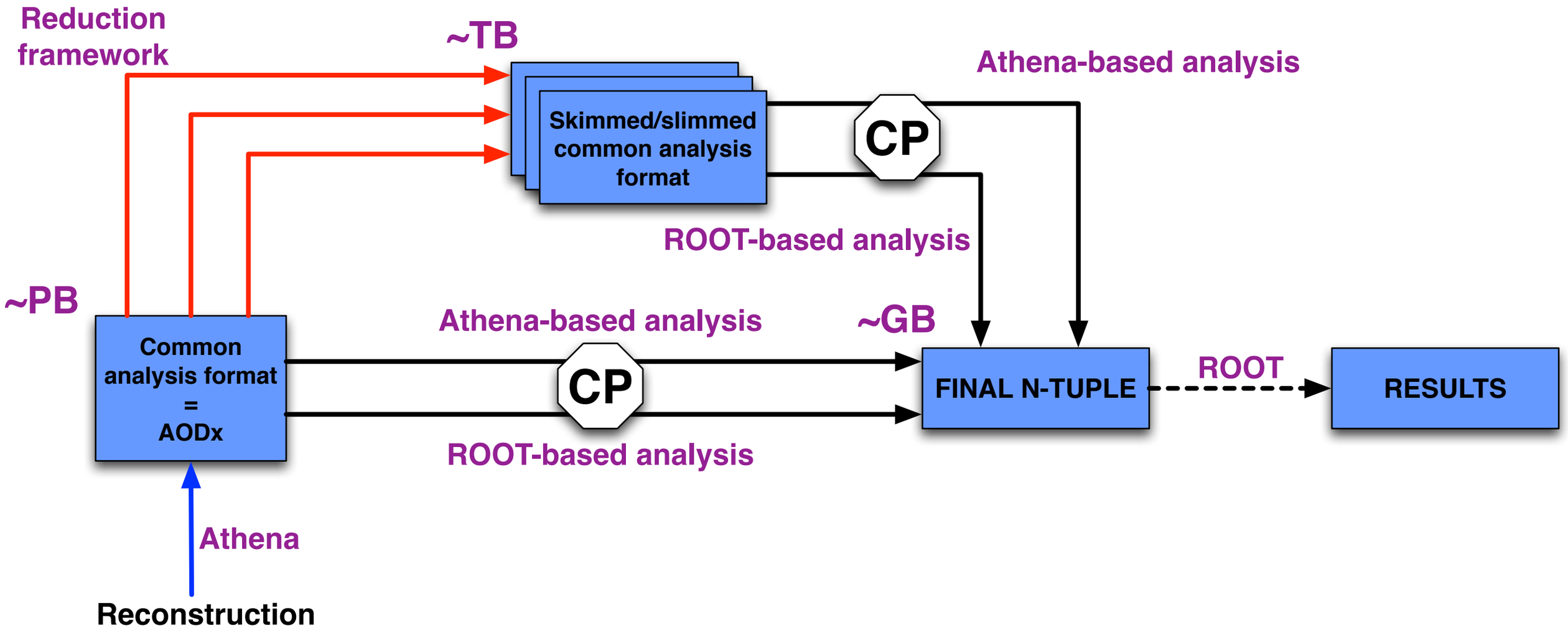
Basic Philosophy

- post-athena packages are meant to work everywhere
  - packages are independent of each other (when feasible)
  - they are meant to work on any machine that runs root
- note: with cmake Athena is more portable as well

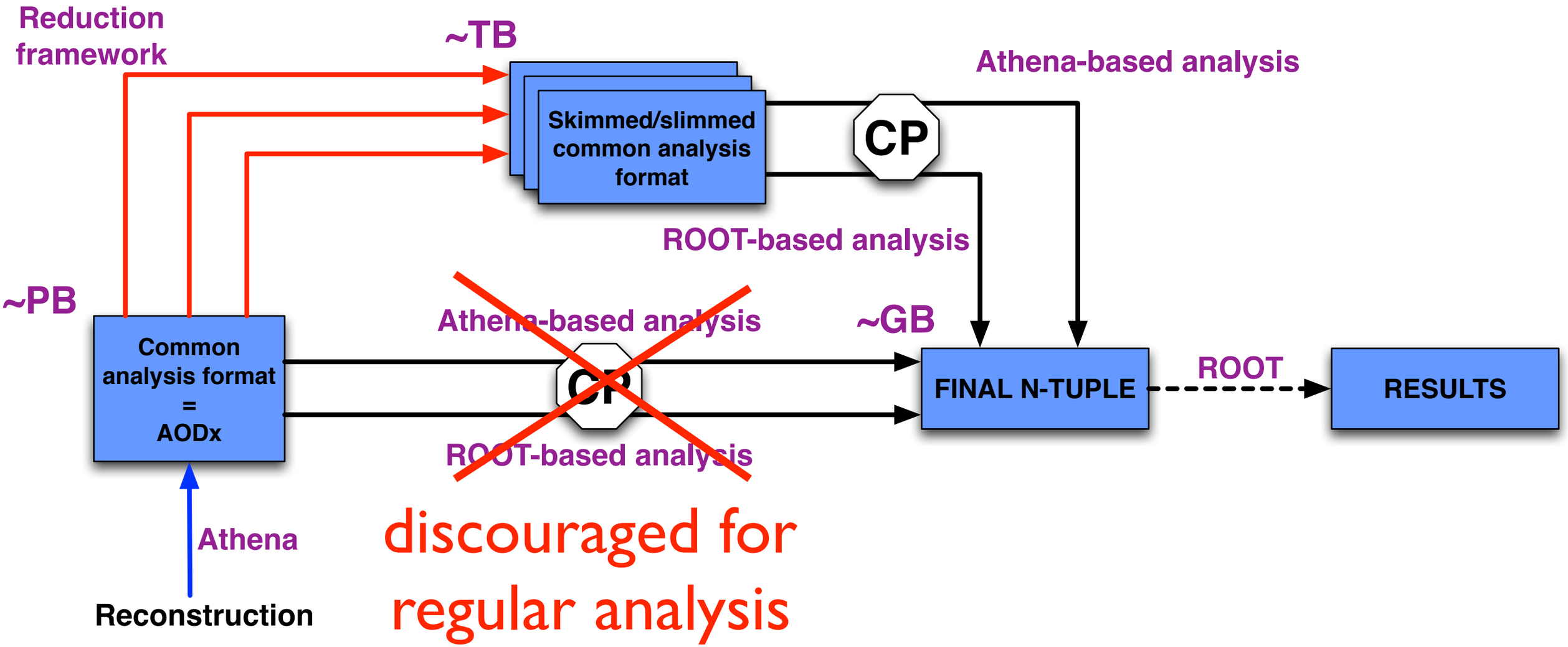
- as a user you get to pick and choose your packages
  - want to reimplement something yourself? go ahead
  - want to implement your own framework? feel free
- however, I strongly urge you to try official solutions first
  - and (almost) always use the official CP tools…
  - and you need analysis releases for xAODs interfaces, etc.
  - and pre-made frameworks save you a lot of time/effort/hassle

- don't like a particular package? tell us why
  - missing features? poor documentation? awkward usage?
- don't see a package addressing your need? tell us
  - many new packages are based on outside requests…

Run 2 Workflow



Run 2 Workflow



Summary

- this was a fairly high level outlook of available analysis tools
  - trying to save time for the hands-on session
- the hands-on part will use most of these tools
  - not using all the advanced features
  - however, it should give you a good feeling of the how-to
  - you can choose to use only a subset of tools for your analysis
- divided into independent sections
  - after you finish the first section, you can work through the other sections in any order
- unlikely you manage to finish it today
  - you can continue asking us questions for the rest of the week or also afterwards by email

- https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/SoftwareTutorialxAODAnalysisInROOT

MultiDraw

- MultiDraw is kind of a specialty package
  - meaning most of you probably will have no use for it
- MultiDraw tries to emulate TTree::Draw inside EventLoop
  - can make multiple plots in parallel (can save time)
  - provides some functionality beyond simple plotting

- could do the same thing using your own EventLoop algorithm
  - main benefit of MD: you can change your output through configuration instead of changing and recompiling algorithms
  - main drawback of MD: can't perform complex operations, i.e. your n-tuple should already include all corrections, etc.
- value of MultiDraw will depend on your personal workflow

- https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/MultiDraw

Release Structure

- analysis releases are based on RootCore:
  - it's what most analyzers are familiar with
  - also what all analysis tools in the release support
  - however: there is a current discussion to support Athena users with an additional cmt based release (lead by Will Buttinger)

- managed like regular Athena releases:
  - package versions managed via Tag Collector
  - nightly builds in NICOS
- no caches: full release comparable in size to Athena cache

- binary releases are distributed via cvmfs & afs:
  - users can set release up via rcSetup (by Shuwei Ye)
  - grid jobs pick up release packages from cvmfs automatically (speeds up submission and reduces resource usage)
- no kits yet, but users can easily check out packages from SVN

Changes To RootCore

- RootCore received a major rewrite for release support:
  - went from shells scripts to a python script
  - expect some more changes when we switch to cmake (unless we completely drop RootCore)

- RootCore version of release support:
  - picks up packages from both release and local area
  - links all files into a local RootCoreBin directory
  - complete out-of-tree build
  - will automatically rebuild release packages if they depend on packages from local area

- for releases root is setup via package Asg_root
  - can still set up root manually if not using a release
  - eliminates need to call ./configure (also when not using release)
- plan to have a separate externals package for RooStats

Dual-Use Tool Interface

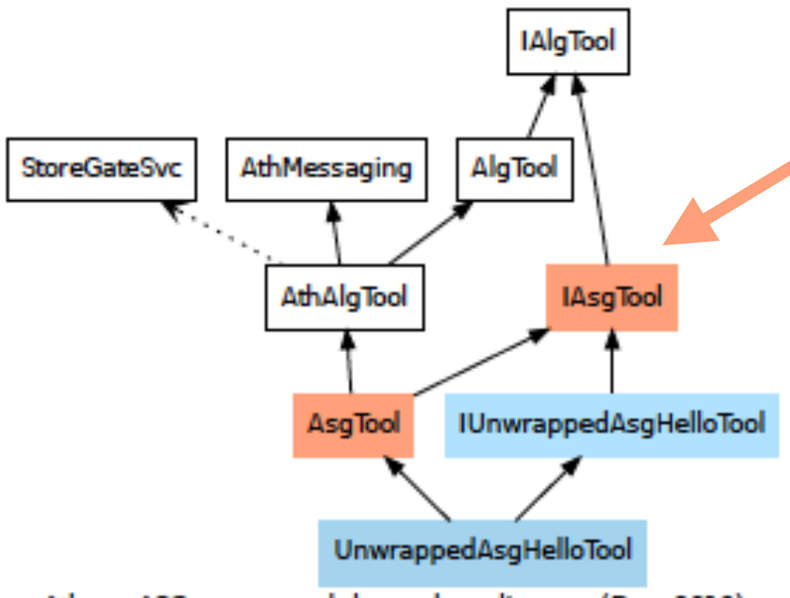
- allows same class to be used as an Athena and a root-only tool
- https://cds.cern.ch/record/1639568?

- AsgTool code in SVN, offline and analysis release
  - code: /Control/AthToolSupport/AsgTools
  - example: /Control/AthToolSupport/AsgExampleTools

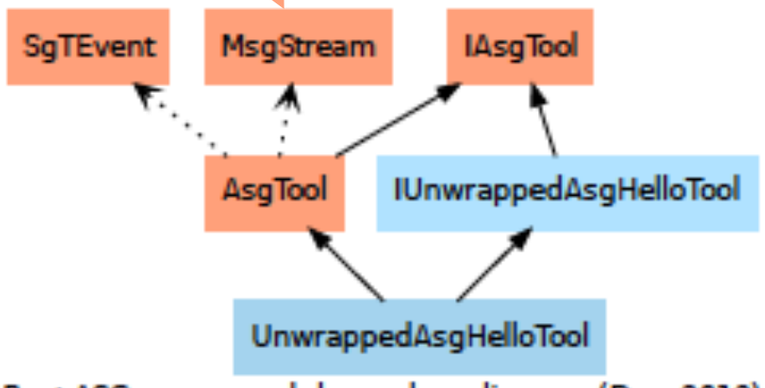
- CP tools should migrate to new interface
  - actually they should be migrated already…

new classes

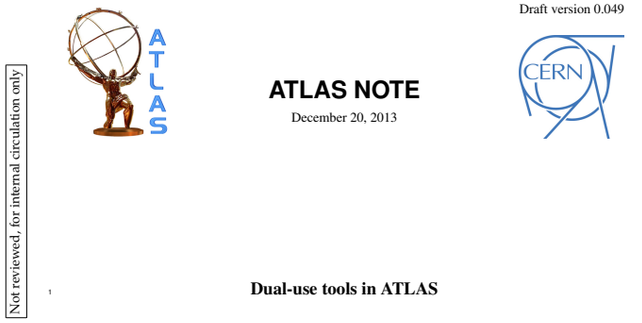
D.Adams et al.



Athena ASG unwrapped demo class diagram (Dec. 2013)



Root ASG unwrapped demo class diagram (Dec. 2013)


Draft version 0.049

Not reviewed, for internal circulation only

ATLAS NOTE

December 20, 2013

Dual-use tools in ATLAS

D. Adams[a], P.A. Delsart[b], M. Elsing[c], K. Köneke[d], E. Lançon[e], W. Lavrijsen[f], S. Strandberg[g], W. Verkerke[h], I. Vivarelli[i], M. Woudstra[j]

[a]*Brookhaven National Laboratory, USA*
[b]*Laboratoire de Physique Subatomique et de Cosmologie, Université Joseph Fourier and CNRS/IN2P3 and Institut National Polytechnique de Grenoble, Grenoble, France*
[c]*CERN, Geneva, Switzerland*
[d]*Fakultät für Mathematik und Physik, Albert-Ludwigs-Universität, Freiburg, Germany*
[e]*DSM/IRFU, CEA Saclay, France*
[f]*Physics Division, Lawrence Berkeley National Laboratory and University of California, Berkeley CA, United States of America*
[g]*Department of Physics, Stockholm University*
[h]*Nikhef National Institute for Subatomic Physics and University of Amsterdam, Amsterdam, Netherlands*
[i]*Department of Physics and Astronomy, University of Sussex, Brighton, United Kingdom*
[j]*University of Manchester, UK*

Abstract

This note provides recommendations and guidelines for the implementation and interfaces of *dual-use* tools. With the event of xAODs, analysis and combined performance software packages should be available through both Athena and ROOT frameworks. Standard interfaces for these *dual-tool* tools have to be defined. Current use cases are listed and requirements for the tools are presented in this note together with recommendations for the implementations. Naming and coding conventions are also proposed. Examples and class diagrams are provided in the appendix.

© Copyright 2013 CERN for the benefit of the ATLAS Collaboration.
Reproduction of this article or parts of it is allowed as specified in the CC-BY-3.0 license.

Tool Design Guidelines

- https://cds.cern.ch/record/1667206/files/ATL-COM-SOFT-2014-005.pdf

AMSG Task Force 3
Draft 0.8, March 24, 2014

Not reviewed, for internal circulation only

Design Guidelines for Combined Performance tools intended to be used in Physics analysis

Introduction

This documents describes a set of design guidelines for analysis tools provided by ATLAS (combined) performance groups that are intended for physics analysis. The goal of this document is to arrive at a homogenous collection of CP analysis tools where tools that perform similar tasks have a similar look-and-feel, and that routine physics analysis tasks, such as the evaluation of systematic uncertainties, are not straightforward to implement by physics users.

This document describes the interface for the implementation of the core functionality analysis tools. The basic interfaces for ROOT/Athena dual-use tools to access configuration data, event store data etc are described in [1].

Derived Releases

- physics groups, subgroups and analysis groups can request their own releases
  - allows to add extra packages the group needs
  - allows to override versions for individual packages
  - allows to create packages on own schedule

- opinions vary greatly on their value:
  - pro: increases flexibility and allows more control by groups
  - con: may make collaboration across groups difficult

- currently have two derived releases
  - AnalysisTop: essentially AnalysisBase + TopRootCore
  - AnalysisSUSY: essentially AnalysisBase + SUSYTools
  - both Top & SUSY already had their own "releases" in run 1
- so far just add packages to base release
- somewhat different release schedule as well

Common Tool Interfaces

- run 1 model: every developer just did whatever he/she wanted

- meant tool interfaces could change last minute
  - typically a few days before a major conference deadline
  - must then be tracked by all users (extra work & stress)
  - trying to have interfaces more stable in run 2

- means every tool had its own recipe for how it is used
  - can spend a lot of time on learning and implementing all of them
  - also increases chances for mistakes
  - try for similar tools to have identical interfaces

- Athena support at times was (very) patchy
  - many developers only work in RootCore
  - now all CP tools should work in Athena out of the box