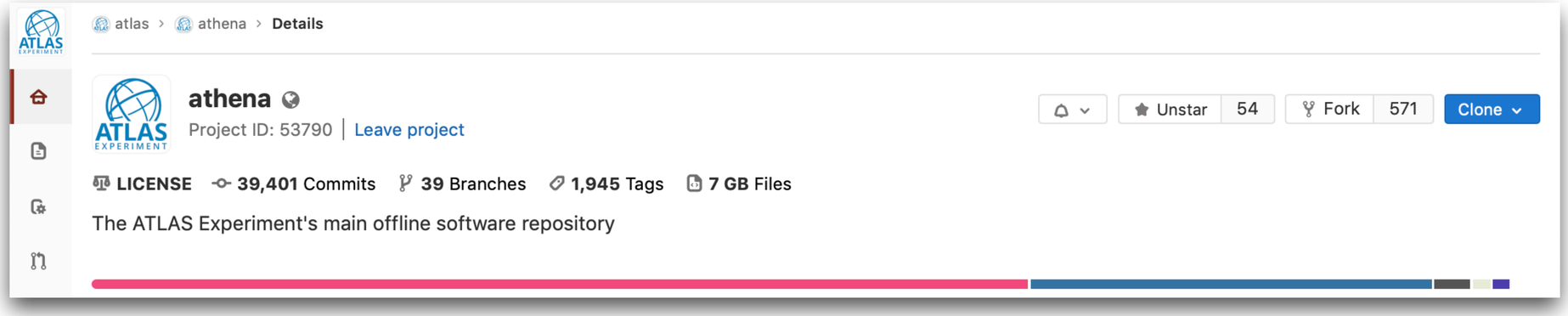
ATLAS EXPERIMENT

xAOD Analysis (in AthAnalysis)

Attila Krasznahorkay

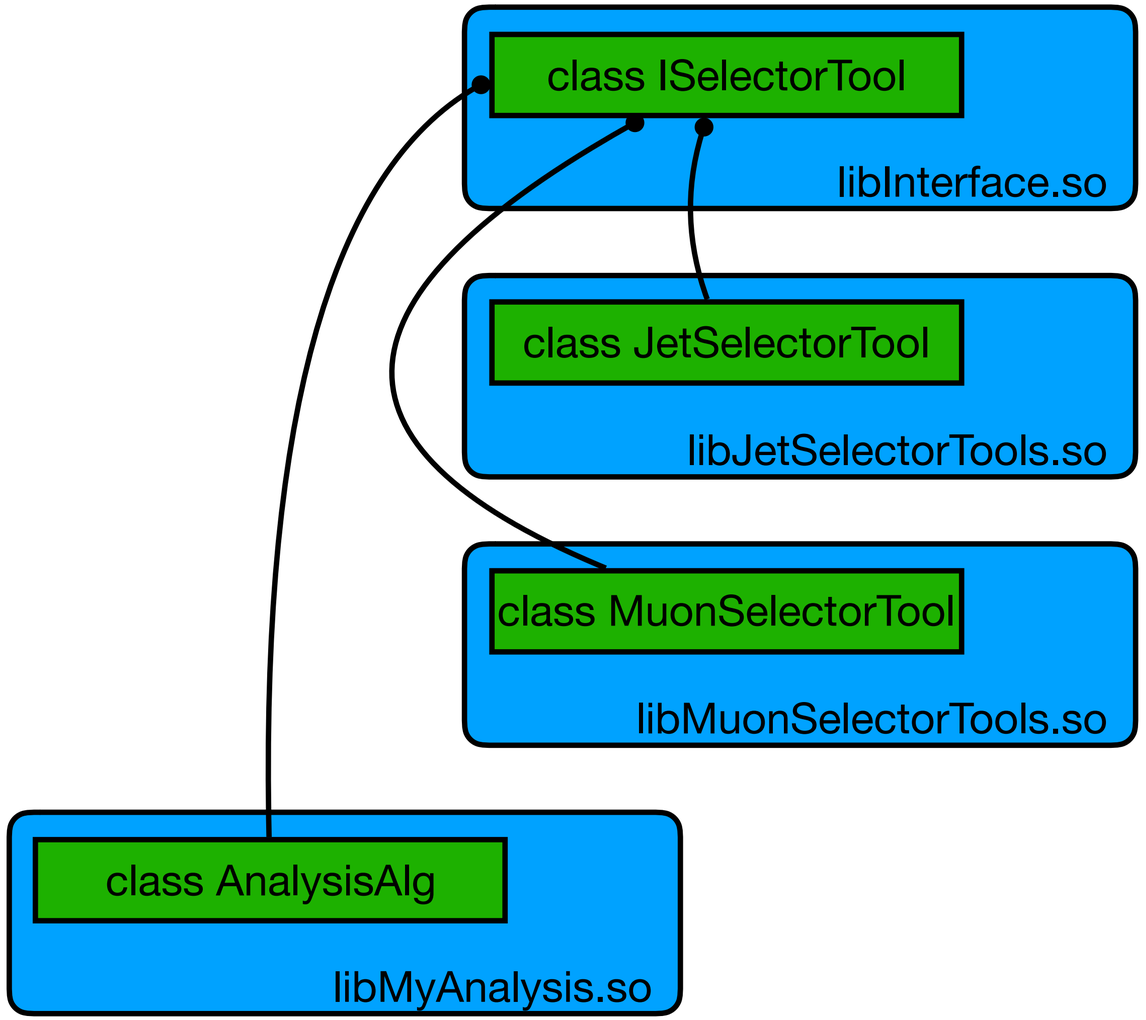
Athena

- As was mentioned many times already, Athena is the name of the software framework that we use to run our simulation, reconstruction and HLT trigger
  - Allowing us to (re)use code written for one, in another one as well
- It is based on the Gaudi framework
  - Building on top of it to provide an ATLAS specific software environment
- By “Athena” we usually mean a job started by `athena.py`, and the software environment that supports running that job
  - But the software releases also contain a **lot** of code that is not run through `athena.py`…



Component Model

- In all of ATLAS software we make a lot of use of the “component model” software design
  - You already encountered it in yesterday’s hands’ on session
- The basic concept is that code compiled in separate “compilation units” (libraries/executables) should only directly depend on each other when necessary
  - Whenever possible, pieces of code should interact with each other through well defined (mostly virtual) interfaces
- This allows us to hide complexities of implementation from code using those interfaces, switching out which concrete implementation would be used by a piece of code without code changes, etc.
- This is also how algorithms work in EventLoop and Athena…



Factories

- Another concept that we use all over HEP code is that of “factories”
  - In general this refers to a way in which code can generate object instances implementing some interface, in some generic way
  - This concept is always used when some program has the ability to use plugins for instance
- ROOT provides a very nice way of constructing objects in generic ways
  - So much of our core software just relies on ROOT for creating new objects in memory
- This is the background to why you had to set up dictionaries and Gaudi factories for your algorithms in yesterday’s example

```
#include <GaudiKernel/DeclareFactoryEntries.h>

#include <MyAnalysis/MyxAODAnalysis.h>

DECLARE_ALGORITHM_FACTORY (MyxAODAnalysis)
```

  - By compiling that piece of code into your library, you taught the EventLoop and Athena frameworks how to create your algorithms, without having to compile the core EventLoop / Athena software against your personal code

Algorithms / Tools / Services

- There are 3 component types defined in Gaudi (that you should know about…)
  - Technically there's a 4th type as well, but you don't need to know about that one…
- Algorithms, as you saw yesterday, have a simple/fixed interface
  - They can be initialised, executed once per event, and finalised
  - They can be scheduled into job
- Tools are objects that are usually private to one owner, and implement a "tool interface"
  - They can access the event store conveniently
  - Are meant to help an algorithm/tool/service to perform one specific task
- Services are accessible by all other components of the job, and implement a "service interface"
  - They are not meant to interact with the event store directly
  - They are usually meant to provide some functionality to the job that's independent of the individual events. Like the magnetic field, detector conditions, etc.

Input / Output

- From a user's perspective input file reading behaves the same way in Athena as it does in EventLoop
  - You need to define in your job configuration which files you want to process, and the framework puts all available objects from those files into the event store event by event
  - The exact syntax is different from EventLoop, but the overall design is exactly the same
  - Athena can even be told to use the same xAOD::TEvent code for file reading in the background that EventLoop uses
- Output files are handled a bit differently however
  - In Athena you generally have a different service for each output file type that you want to write
    - To write a POOL (xAOD) output file, you have to use AthenaPoolCnvSvc. You don't use it directly however, you define output stream writing algorithms, that write files using that service.
    - To write a ROOT (histogram/ntuple) output file, you have to use THistSvc. Though it's easiest to just rely on the helper functions provided by EL::AnaAlgorithm to create histograms and ntuples in a uniform way between Athena and EventLoop
      - Note that EL::AnaAlgorithm does use THistSvc behind the scenes in Athena

JobOptions

- Athena is truly a framework in the sense that you don’t write a top level executable / script yourself
  - You write a python configuration that by itself would not run anything, feed it to the `athena.py` executable, and that sets up and runs a job based on that configuration
- Let’s see what the simplest jobOption configuration looks like:

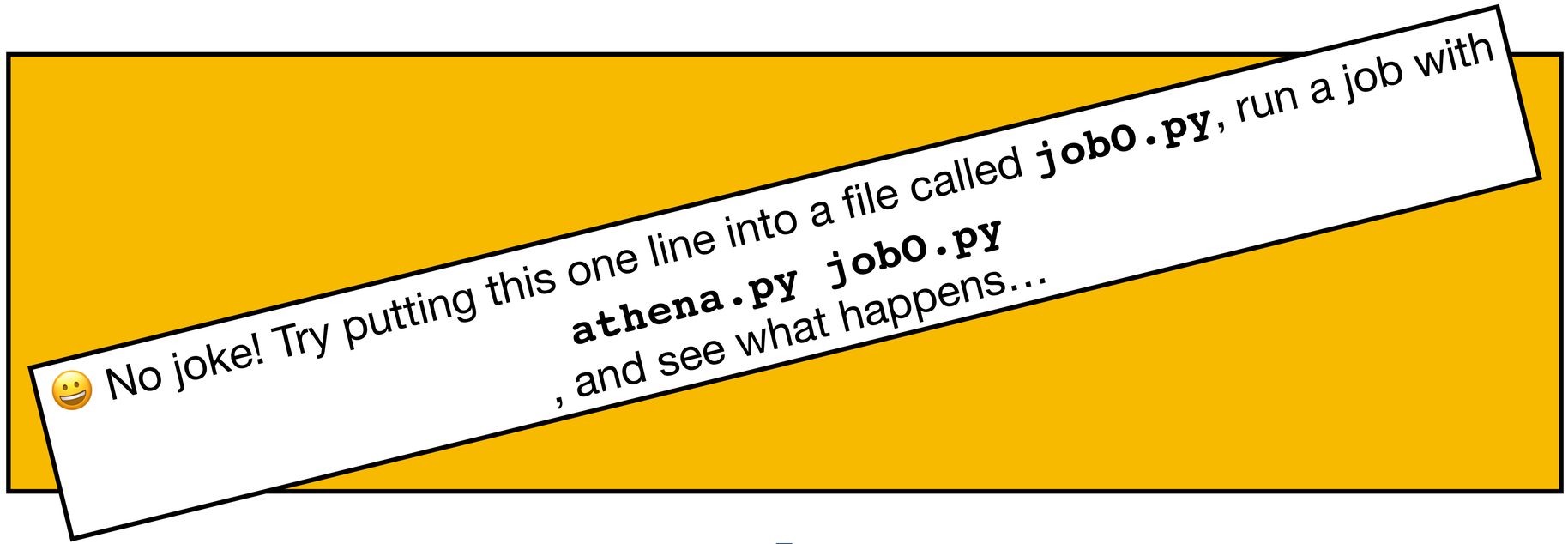
JobOptions

- Athena is truly a framework in the sense that you don't write a top level executable / script yourself
  - You write a python configuration that by itself would not run anything, feed it to the `athena.py` executable, and that sets up and runs a job based on that configuration
- Let's see what the simplest jobOption configuration looks like:

```
theApp.EvtMax = 10
```

JobOptions

- Athena is truly a framework in the sense that you don't write a top level executable / script yourself
  - You write a python configuration that by itself would not run anything, feed it to the `athena.py` executable, and that sets up and runs a job based on that configuration
- Let's see what the simplest jobOption configuration looks like:



Job With an Input File

```
# Set up the xAOD::TEvent based xAOD reading infrastructure.
import AthenaRootComps.ReadAthenaxAODHybrid
# Or uncomment the following line instead to use "normal" xAOD reading.
#import AthenaPoolCnvSvc.ReadAthenaPool

# Set the name(s) of the input file(s).
svcMgr.EventSelector.InputCollections = [ ‘xAOD.pool.root’ ]

# Set up / access the main algorithm sequence of the job.
from AthenaCommon.AlgSequence import AlgSequence
topSequence = AlgSequence()

# Add a “random” algorithm.
topSequence += CfgMgr.GRLSelectorAlg( 'GRLSelector' )
topSequence.GRLSelector.Tool.GoodRunsListVec = [
   'GoodRunsLists/data16_13TeV/20180129/physics_25ns_21.0.19.xml'
   ]

# Run on just 10 events of the input file.
theApp.EvtMax = 10
```

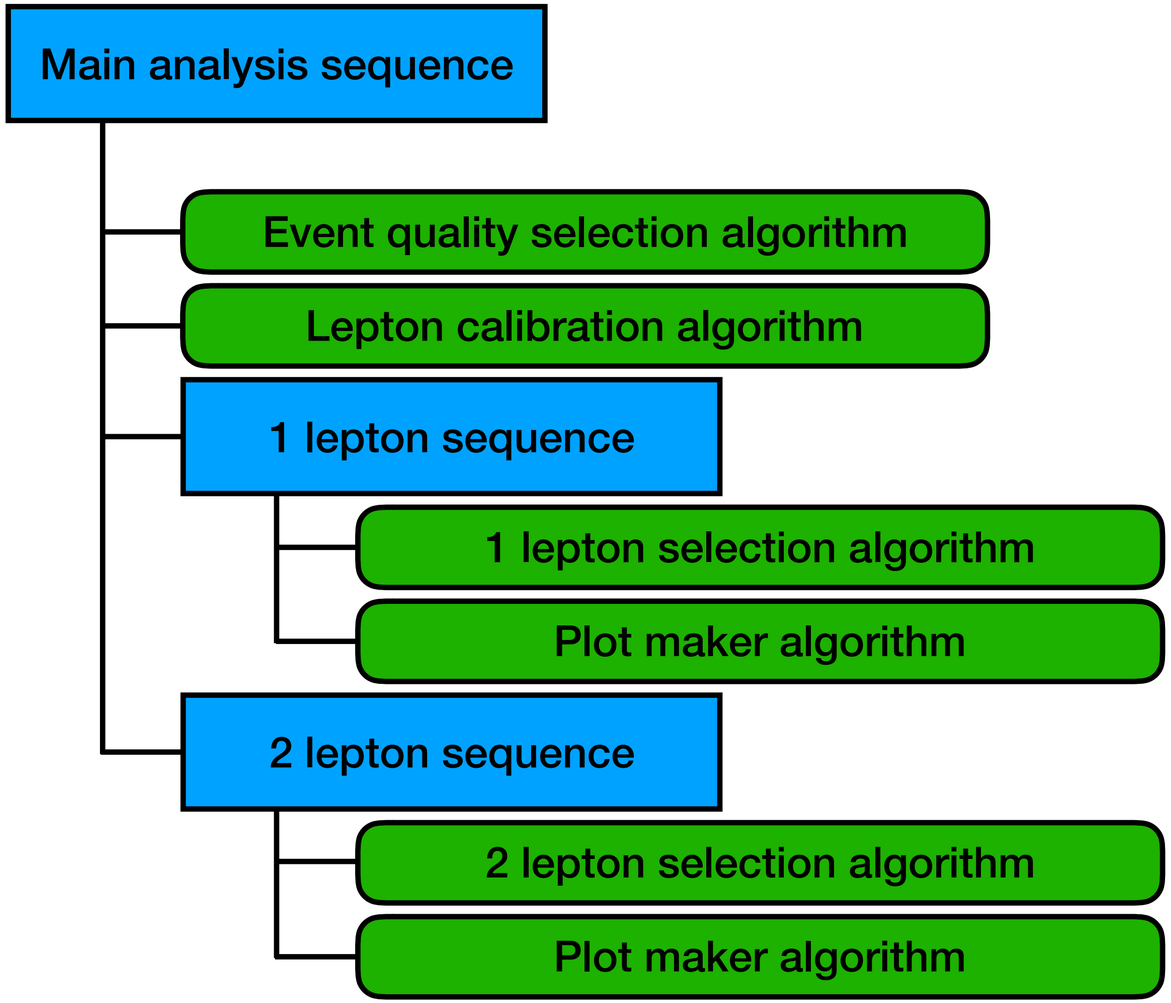
Configurables

- When we set up a “factory” for a component with Gaudi/Athena, during compilation Gaudi inspects the object that we created the factory for, and generates a “Python representation” of that component
- This representation is then used to set up instances of that component in (analysis) Athena jobs

```
alg = CfgMgr.MySuperAlgorithm( ‘MyAlgorithm’, JetContainer = ‘AntiKt4EMTopoJets’ )
alg.ElectronContainer = ‘Electrons’
alg.Tool = CfgMgr.MySuperTool( ‘MyTool’, OutputLevel = DEBUG )
alg.Tool.AnalysisQuality = ‘HIGH’
```

- It is very important to remember that the algorithm/tool/service python objects that you interact with in a jobOption are not the C++ objects
  - They are just a representation of the configuration of the C++ objects, but they themselves are pure Python objects

Algorithm Sequences



- Athena allows us to have multiple (nested) algorithm sequences
- This allows for a number of neat tricks
  - Though usually you could just implement the same branching behaviour inside your own algorithms as well
- Still, if you decide to use AthAnalysis, you should consider if using algorithm sequences in a smart way could help your code's organisation

xAOD Outputs

- As I already said, writing histograms and ntuples in Athena is not drastically different from doing the same in EventLoop
- But (mini-)xAOD file writing is…
- To set up a custom mini-xAOD output file from your job, you need to add the following kind of configuration to it:

```
from OutputStreamAthenaPool.MultipleStreamManager import MSMgr
minixAOD = MSMgr.NewPoolRootStream( ‘DAOD_MYANALYSIS',
               FileName = ‘test.DAOD_MYANALYSIS.pool.root’ )
minixAOD.AddItem( [
   'xAOD::EventInfo#EventInfo',
   'xAOD::EventAuxInfo#EventInfoAux.-',
   'xAOD::MuonContainer#AnalysisMuons_NOSYS',
   'xAOD::AuxContainerBase#AnalysisMuons_NOSYSAux.eta.phi.pt' ] )
```

xAOD Outputs

- As I already said, writing histograms and ntuples in Athena is not drastically different from doing the same in EventLoop
- But (mini-)xAOD file writing is…
- To set up a custom mini-xAOD output file from your job, you need to add the following kind of configuration to it:

```
from OutputStreamAthenaPool.MultipleStreamManager import MSMgr
minixAOD = MSMgr.NewPoolRootStream( 'DAOD_MYANALYSIS',
                FileName = 'test.DAOD_MYANALYSIS.pool.root' )
minixAOD.AddItem( [
    'xAOD::EventInfo#EventInfo',
    'xAOD::EventAuxInfo#EventInfoAux.-',
    'xAOD::MuonContainer#AnalysisMuons_NOSYS',
    'xAOD::AuxContainerBase#AnalysisMuons_NOSYSAux.eta.phi.pt' ] )
```

Sets up an output stream/file

xAOD Outputs

- As I already said, writing histograms and ntuples in Athena is not drastically different from doing the same in EventLoop
- But (mini-)xAOD file writing is…
- To set up a custom mini-xAOD output file from your job, you need to add the following kind of configuration to it:

```
from OutputStreamAthenaPool.MultipleStreamManager import MSMgr
minixAOD = MSMgr.NewPoolRootStream( 'DAOD_MYANALYSIS',
                FileName = 'test.DAOD_MYANALYSIS.pool.root' )
minixAOD.AddItem( [
    'xAOD::EventInfo#EventInfo',
    'xAOD::EventAuxInfo#EventInfoAux.-',
    'xAOD::MuonContainer#AnalysisMuons_NOSYS',
    'xAOD::AuxContainerBase#AnalysisMuons_NOSYSAux.eta.phi.pt' ] )
```

Sets up an output stream/file

Specifies what to write to the file

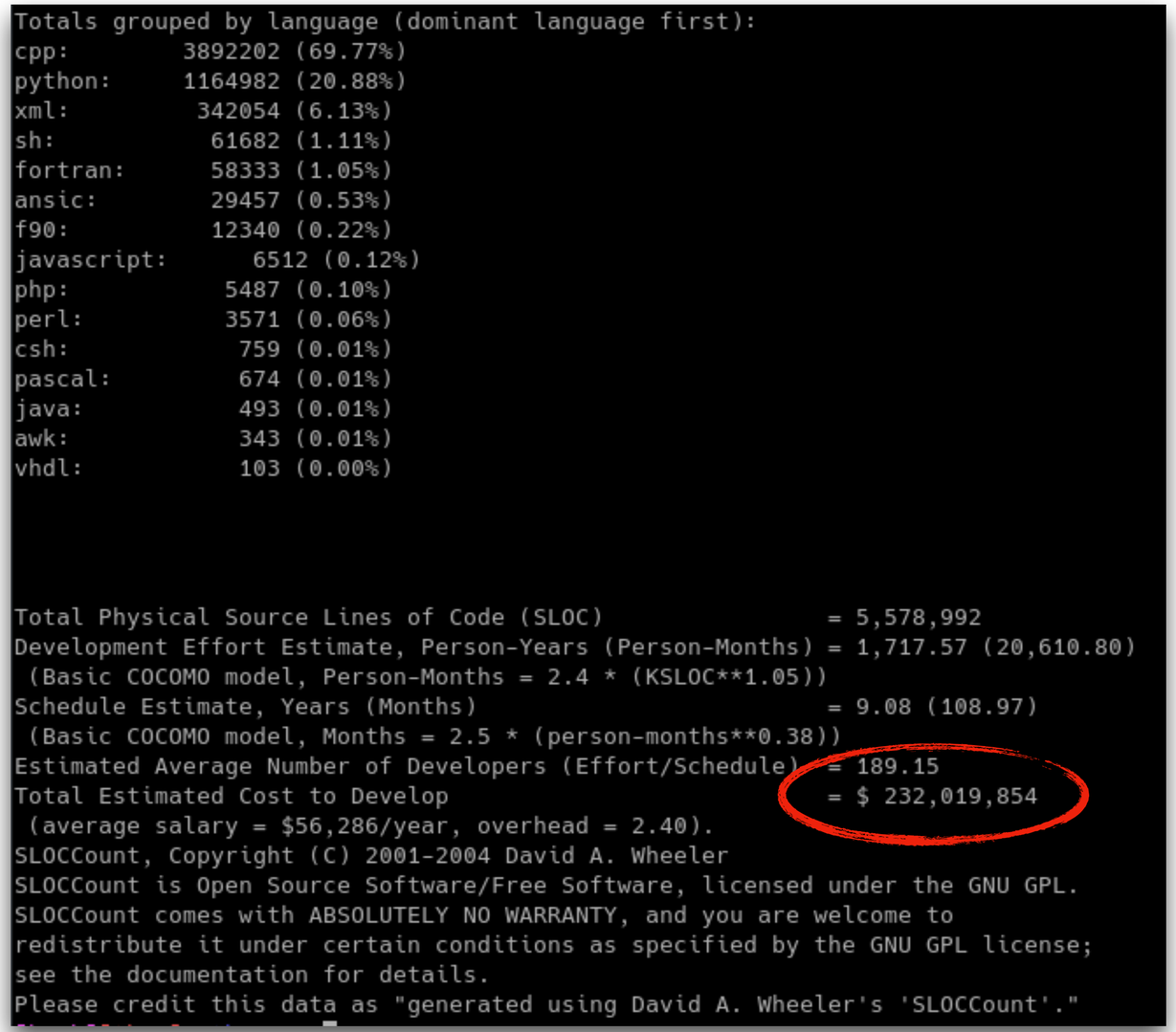
Large Configurations

- Analysis jobs are usually reasonably easy to configure
  - Though Nils will show you later that even for the centrally provided analysis algorithms the full configuration ends up being not completely trivial…
- What usually gives Athena a bad name is how really large configurations / jobs are handled in it
  - The full reconstruction uses O(500) algorithms, O(1000) tools, O(100) services
  - Since that configuration grew pretty organically for ~10 years, by now it became pretty hard to handle
  - For Run-3 there is an effort ongoing to reform how we handle large configurations like that, but this is still under heavy development
- All in all, if you need to work on offline/trigger code, you will encounter a lot of additional python code that we use to manage such configurations
  - JobProperty flags;
  - Customised configurables;
  - etc.
- You will see a little bit of these details in the “derivation session”

Project Size

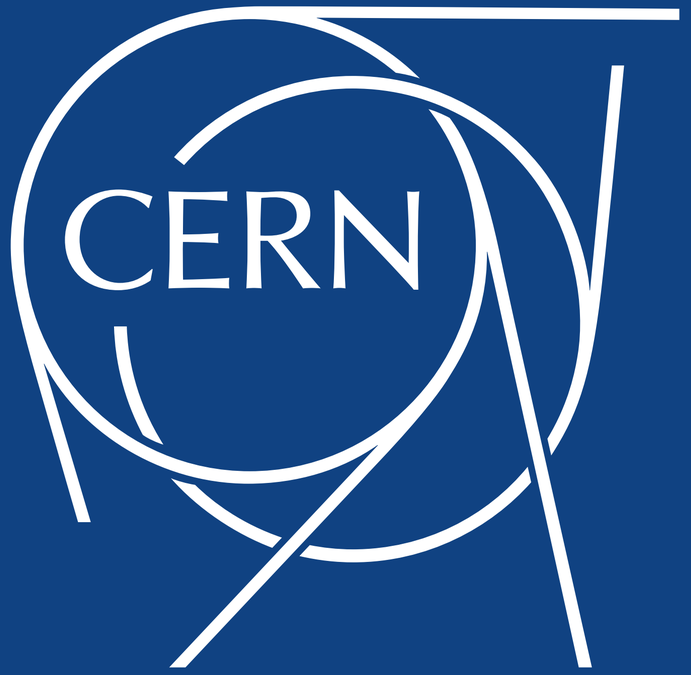
```
Totals grouped by language (dominant language first):
cpp:         3892202 (69.77%)
python:      1164982 (20.88%)
xml:          342054 (6.13%)
sh:            61682 (1.11%)
fortran:       58333 (1.05%)
ansic:         29457 (0.53%)
f90:           12340 (0.22%)
javascript:        6512 (0.12%)
php:            5487 (0.10%)
perl:           3571 (0.06%)
csh:             759 (0.01%)
pascal:          674 (0.01%)
java:            493 (0.01%)
awk:             343 (0.01%)
vhdl:            103 (0.00%)



Total Physical Source Lines of Code (SLOC)                = 5,578,992
Development Effort Estimate, Person-Years (Person-Months) = 1,717.57 (20,610.80)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 9.08 (108.97)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 189.15
Total Estimated Cost to Develop                           = $ 232,019,854
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Project Size



Summary

- Athena provides a **very** powerful framework for ATLAS's simulation/reconstruction/trigger code
- With all the latest improvements its "analysis performance" can be on par with EventLoop jobs
  - Specific jobs may be slightly faster in one or the other, but in general they should be able to run with about the same speed
- Setting up Athena jobs "nicely" requires learning a bit more than for small EventLoop jobs
  - Though when you try to set up a small Athena job, that is not fundamentally different from setting up a small EventLoop job
  - You will see a bit more about "job configuration" in the common analysis algorithm session
- Learning how to use Athena can have benefits for your career
  - Whether you stay in HEP, or move on to something else after your PhD



http://home.cern