Taming the beast: Using Python To Control ATLAS Software

David Quarrie
NERSC Division
Lawrence Berkeley National Laboratory
3 July 2006

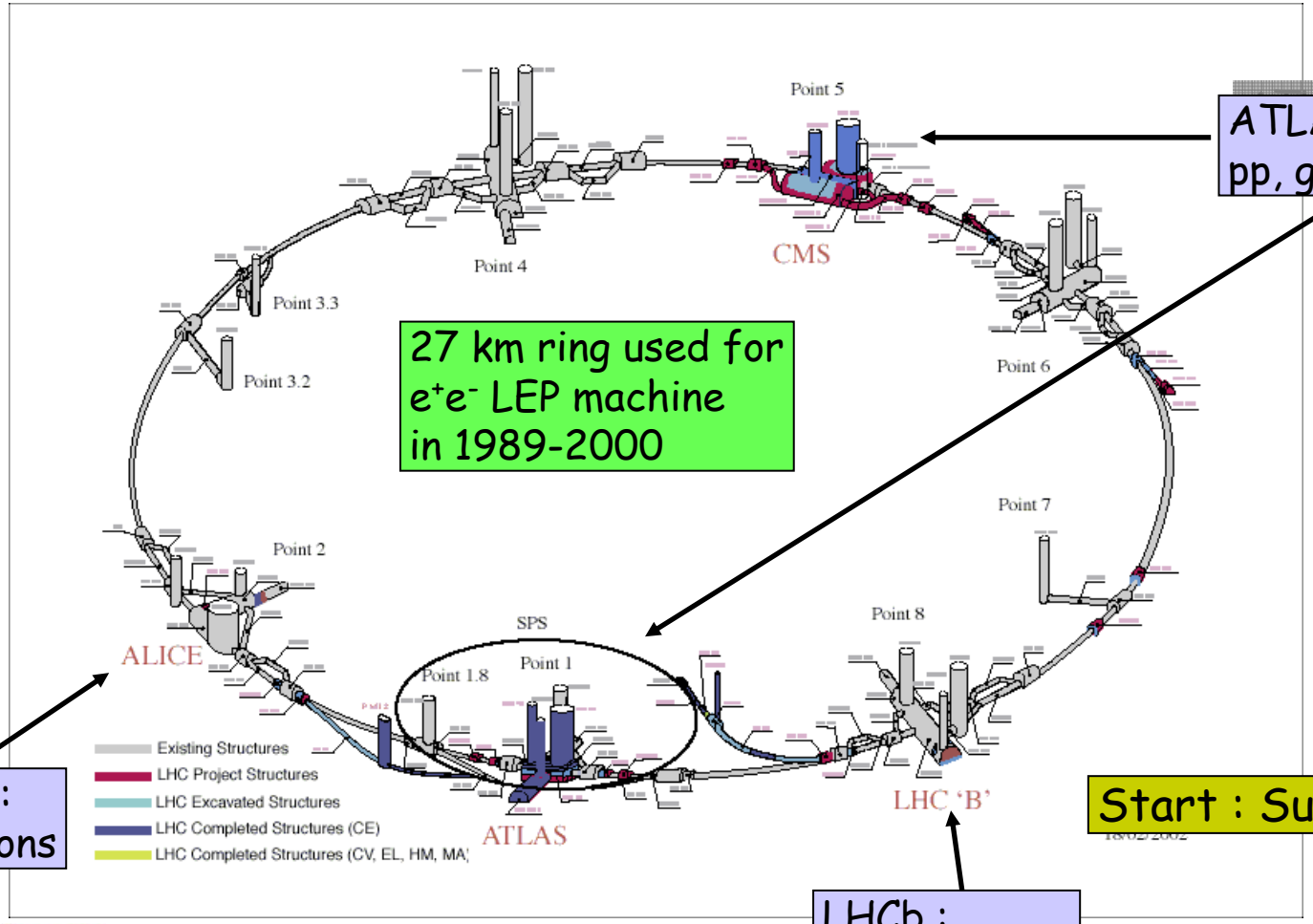
Authors

Paolo Calafiura (LBNL)
Manuel Gallas (CERN)
Wim Lavrijsen (LBNL)
Wolfgang Liebig (CERN)
Tadashi Maeno (Brookhaven National Laboratory)
David Quarrie (LBNL)
David Rousseau (LAL, Orsay)
Andreas Salzburger (Univ. of Innsbruck)
Akira Shibata (QMC, Univ. of London)

LHC

pp

- □s = 14 TeV (7 times higher than Tevatron/Fermilab)
  search for new massive particles up to m ~ 5 TeV
- $L_{design} = 10^{34}\ cm^{-2}\ s^{-1}$ (>10^2 higher than Tevatron/Fermilab)
  search for rare processes with small ∫ (N = L ∫)



27 km ring used for e^+e^- LEP machine in 1989-2000

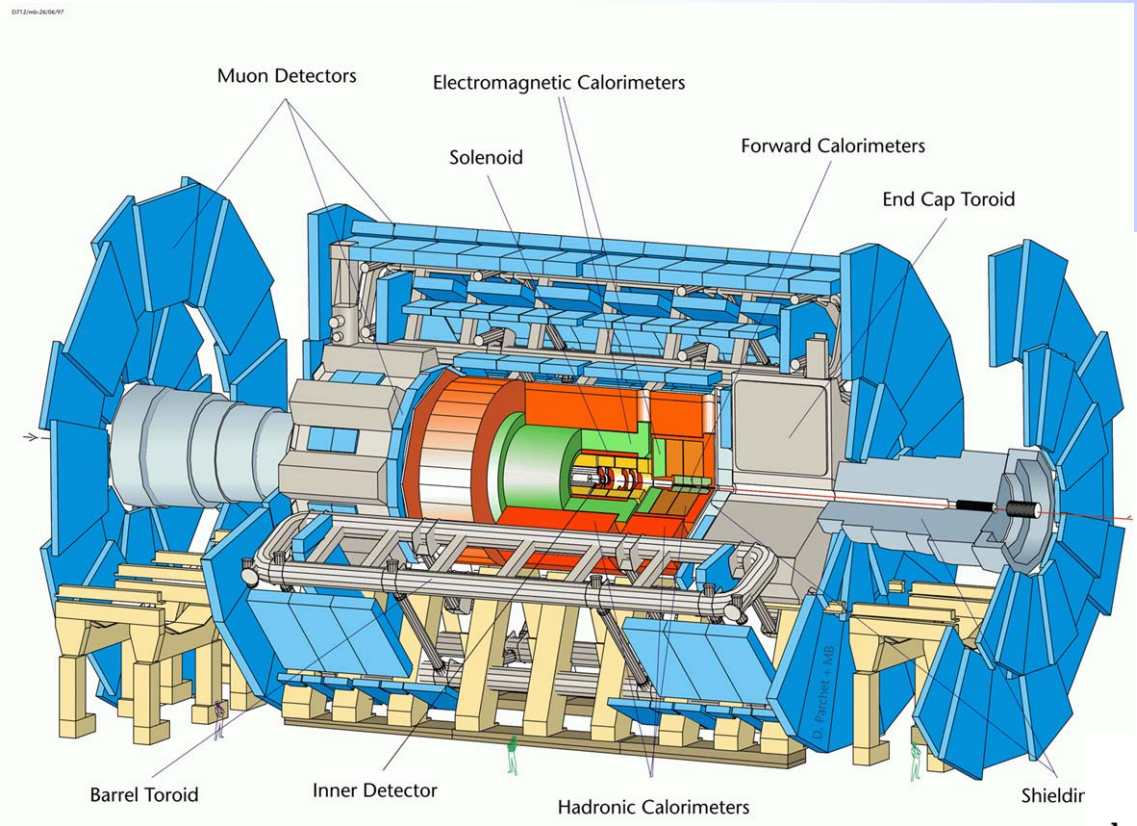
ATLAS and CMS : pp, general purpose

ALICE : heavy ions

LHCb : pp, B-physics

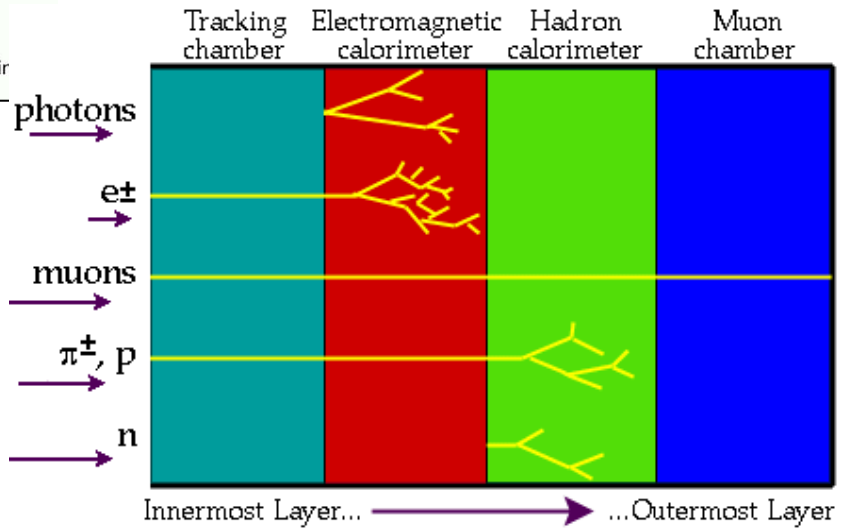
Start : Summer 2007

ATLAS



Length : ~ 46 m
Radius : ~ 12 m
Weight : ~ 7000 tons
~ 10^8 electronic channels
~ 3000 km of cables

- **Tracking (|η|<2.5, B=2T) :**
  - **-- Si pixels and strips**
  - **-- Transition Radiation Detector (e/π separation)**
- **Calorimetry (|η|<5) :**
  - **-- EM : Pb-LAr**
  - **-- HAD: Fe/scintillator (central), Cu/W-LAr (fwd)**
- **Muon Spectrometer (|η|<2.7) :**
  air-core toroids with muon chambers



ATLAS Collaboration

34 Countries
151 Institutions
1770 Scientific Authors

Albany, Alberta, NIKHEF Amsterdam, Ankara, LAPP Annecy, Argonne NL, Arizona, UT Arlington, Athens, NTU Athens, Baku, IFAE Barcelona, Belgrade, Bergen, Berkeley LBL and UC, Bern, Birmingham, Bonn, Boston, Brandeis, Bratislava/SAS Kosice, Brookhaven NL, Bucharest, Cambridge, Carleton, Casablanca/Rabat, CERN, Chinese Cluster, Chicago, Clermont-Ferrand, Columbia, NBI Copenhagen, Cosenza, INP Cracow, FPNT Cracow, Dortmund, JINR Dubna, Duke, Frascati, Freiburg, Geneva, Genoa, Glasgow, LPSC Grenoble, Technion Haifa, Hampton, Harvard, Heidelberg, Hiroshima, Hiroshima IT, Indiana, Innsbruck, Iowa SU, Irvine UC, Istanbul Bogazici, KEK, Kobe, Kyoto, Kyoto UE, Lancaster, Lecce, Lisbon LIP, Liverpool, Ljubljana, QMW London, RHBNC London, UC London, Lund, UA Madrid, Mainz, Manchester, Mannheim, CPPM Marseille, Massachusetts, MIT, Melbourne, Michigan, Michigan SU, Milano, Minsk NAS, Minsk NCPHEP, Montreal, FIAN Moscow, ITEP Moscow, MEPhI Moscow, MSU Moscow, Munich LMU, MPI Munich, Nagasaki IAS, Naples, Naruto UE, New Mexico, Nijmegen, BINP Novosibirsk, Ohio SU, Okayama, Oklahoma, LAL Orsay, Oslo, Oxford, Paris VI and VII, Pavia, Pennsylvania, Pisa, Pittsburgh, CAS Prague, CU Prague, TU Prague, IHEP Protvino, Ritsumeikan, UFRJ Rio de Janeiro, Rochester, Rome I, Rome II, Rome III, Rutherford Appleton Laboratory, DAPNIA Saclay, Santa Cruz UC, Sheffield, Shinshu, Siegen, Simon Fraser Burnaby, Southern Methodist Dallas, NPI Petersburg, Stockholm, KTH Stockholm, Stony Brook, Sydney, AS Taipei, Tbilisi, Tel Aviv, Thessaloniki, Tokyo ICEPP, Tokyo MU, Tokyo UAT, Toronto, TRIUMF, Tsukuba, Tufts, Udine, Uppsala, Urbana UI, Valencia, UBC Vancouver, Victoria, Washington, Weizmann Rehovot, Wisconsin, Wuppertal, Yale, Yerevan

ATLAS Computing Characteristics

- Large, complex detector
  - ~10^8 channels
- Long lifetime
  - Project started in 1992, first data in 2007, last data 2027?
- 320 MB/sec raw data rate
  - ~3 PB/year
- Large, geographically dispersed collaboration
  - 1770 people, 151 institutions, 34 countries
  - Most are, or will become, software developers
- Scale and complexity reflected in software
  - ~1000 packages, ~8000 C++ classes, ~5M lines of code
  - ~70% code is algorithmic (written by physicists)
  - ~30% infrastructure, framework (written by sw engineers)
  - Provide robustness but plan for evolution
  - Requires enabling technologies
  - Requires management & coherency

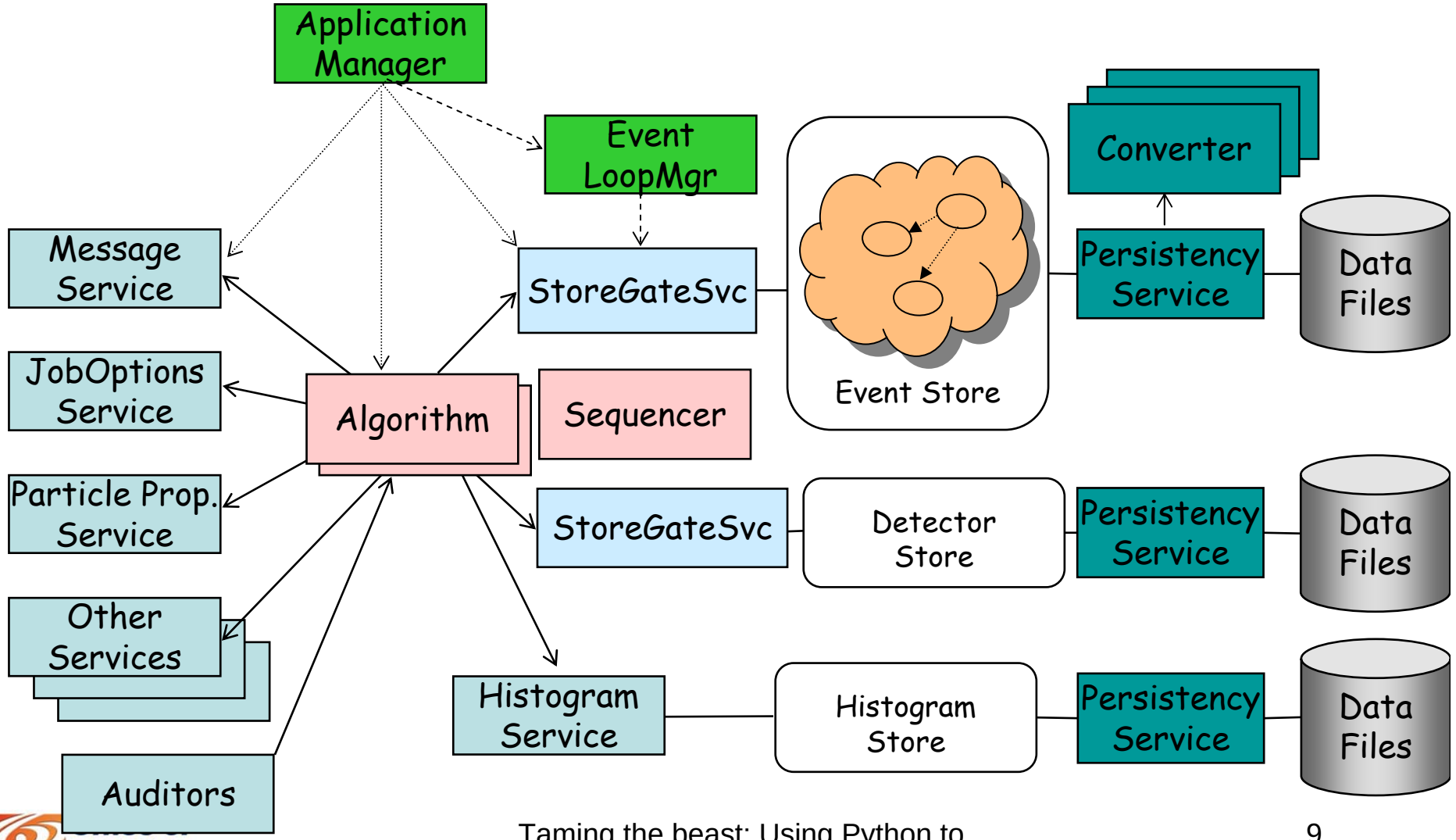
Software Methodology

- Object-Oriented using C++ as programming language
  - Some wrapped FORTRAN and Java
  - Python as interactive & configuration language
- Heavy use of components behind abstract interfaces
  - Support multiple implementations
  - Robustness & evolution
- Lightweight development process
  - Emphasis on automation and feedback rather than very formal process
    - Previous attempt at developing a software system had failed due to a too rigorous software process decoupled from developers
  - Make it easy for developers to do the “right thing”
  - Some requirements/design reviews
  - Sub-system “functionality” reviews
    - 2 weeks each
    - Focus on client viewpoint

Control Framework

- Framework is implementation of an architecture
- Capture common behaviour for HEP processing
  - Processing stages
    - Generation, simulation, digitization, reconstruction, analysis
  - Online (trigger & monitoring) & offline
- Control framework steers series of modules to perform transformations
  - Component based
  - Dynamically reconfigurable
- Although framework captures common behaviour, it's important to make it as flexible and extensible as possible
- Blackboard model
  - Algorithms register and retrieve data on shared blackboard
  - Component decoupling
- Athena/Gaudi Framework common project with LHCb

Athena/Gaudi Component Model



Athena Components

- Algorithms
  - Provide basic per-event processing
  - Share a common interface (state machine)
  - Sequencer is type of Algorithm that sequences/filters other Algorithms
- Tools
  - More specialized but more flexible than Algorithms
- Services
  - E.g. Particle Properties, Random Numbers, Histogramming
- Data Stores (blackboards)
  - Data registered by one Algorithm/Tool can be retrieved by another
  - Multiple stores handle different lifetimes (per event, per job, etc.)
  - Stores accessed via Services (e.g. StoreGateSvc)
- Converters
  - Transform data from one representation to another
    - e.g. transient/persistent
- Properties
  - Adjustable parameters of components
  - Can be modified at run-time to configure job

Python in Athena, a bit of history

- Initially configuration specified using text files and ad-hoc parser
- Extended bindings to most athena components
  - Initial goal to run athena interactively from a python shell
- athena now a python program controlling primarily C++ components

C++/Python Bindings

- Several iterations
  - Manual (C API)
  - boost::python
  - LHC common project C++ Reflex package and its auto-generated python bindings
    - gccxml
- Still need to customize the bindings
  - Not writing a debugger: interaction should respect athena “state machine” architecture
  - Dynamic vs linked language
- see next talk by P. Mato

Job Configuration

- Decided to migrate to python to
  - address scalability, maintainability issues by introducing structure in job configuration
  - Take advantage of a widely used, well documented language
- Two-stage migration strategy
  - Started with 80K lines of text configuration files
  - Use (python) tool to translate text to python

```
HelloWorld.MyDouble = 3.14159
HelloWorld.MyStringVec = [ "Welcome", "to", "Athena",
  "Framework", "Tutorial" ]
```

  - This of course does not solve lack of structure
- Build structure by taking advantage of language features

Building Structure

- **Text-based to python translation left every property setting in global scope**
  - **Long-range side effects**
  - **Could not modularize automatically**
- **Solution is a three step process:**
  - ***Provide smarter, low-level building blocks***
    - **Automatically generated for all components**
  - ***Provide structuring support***
    - **Removes boiler-plate code from configuration**
  - ***Build higher level structures***
    - **Simplify user view, full access for developers**

Example of Structured Configuration

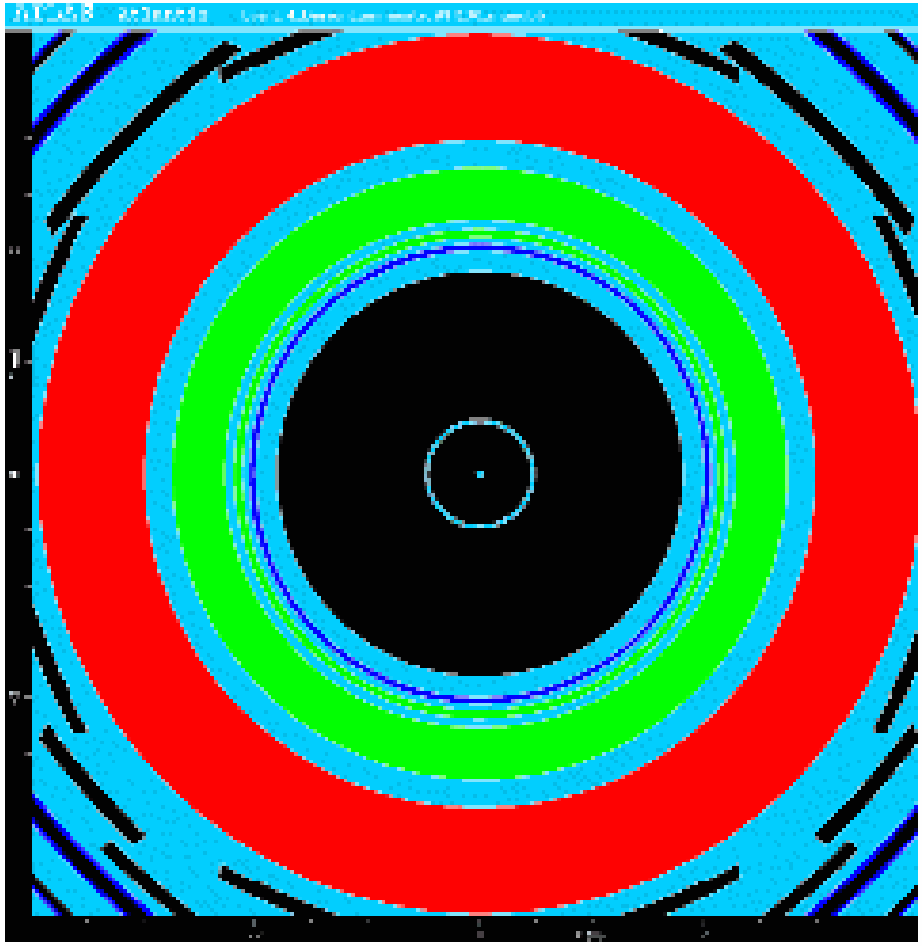
```
# instantiate the top alg sequence for athena
from AthenaCommon.AlgSequence import *
theJob = AlgSequence()

# instantiate one tool looper
from EventViewConfiguration.EVToolLoopers import *
defaultEVLooper = EVMultipleOutputToolLooper("defaultEVLooper")
# the name of the EventView (in StoreGate) created by this tool looper
defaultEVLooper.EventViewCollectionOutputName="defaultEventView"
# schedule EVMultipleOutputToolLoopers to athena top sequencer
theJob += defaultEVLooper

from EventViewConfiguration.DefaultEventView_module import *
# define insertion order, you may want to play with this order
toInsert=["Electron", "Photon", "Muon","TauJet","JetTag”]
# schedule the default inserter module with this insertion order
defaultEVLooper += DefaultEventView("Inserters", toDo=toInsert)

# set up the athena job
theJob.setup()
# print the whole job schedule
print theJob

# Run the job
theJob.run()
```

Python Control Framework

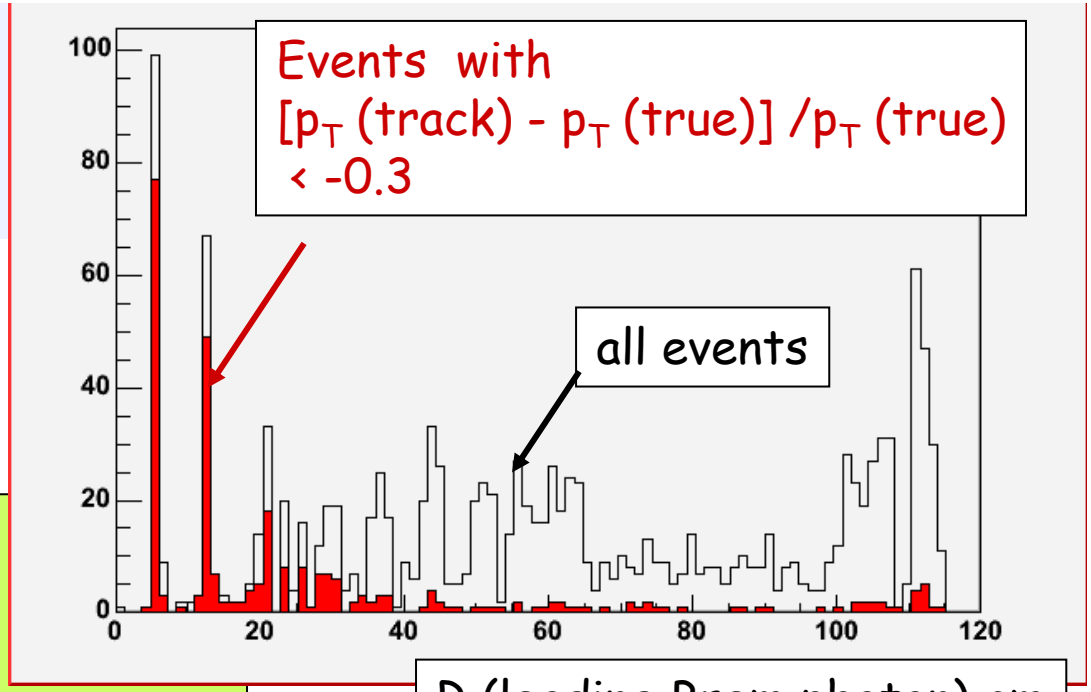
- **Moved Application Control to python**
  - **Support interactivity**
  - **Allow inspection, debugging**
  - **Alternative control flows**
  - **Act as a glue between athena components and 3rd party tools (ROOT, Geant 4 simulation, memory profiler,…)**
  - **String manipulation**
- **“Business” logic still in C++ Algorithms, Tools, Services**

Data Analysis in Python

- Interoperability of athena and analysis tools
  - Atlantis Java Event Display can drive athena in client-server mode using the SimpleXMLRPCServer module
  - Allows interactive (re)processing of athena data



Interactive analysis of $Z \rightarrow ee$ events



PyROOT

```
import ROOT
import math
#
theApp.initialize()

h16 = ROOT.TH1F ("h16", 'R G4 Brem photons',100, 0.,\
120.)
for i in range(5000):
  theApp.nextEvent()
  scon=PyParticleTools.getTruthParticles("SpclMC")

  for p in scon:
   t=p.getGenParticle()
   tver=t.production_vertex()
   if tver :
    R=math.pow(tver.point3d().y(),2)\
+math.pow(tver.point3d().x(),2)
    R=math.sqrt(R)/10.

    if p.pdgId()==22 and math.fabs(t.barcode()) >\
10000.:
      if math.fabs(p.eta())< 1.0:
        h16.Fill(R)
#
```

Access to athena data objects from analysis tools

Some ongoing issues

- Two languages problem
  - Learning curve
    - Not only for users but also for developers
  - Where to place the boundary between the two
  - Language mapping
    - C++ Object ownership
    - C++ templates
- Higher level structures
  - Now have capability but detailed design requires physicist involvement & feedback

Summary and Outlook

- ATLAS software is changing:
  - From development focus to analysis/production
- Configuration building blocks provided
  - Auto-generated, checkable, independent
- Layered structures now possible
  - With layered builders (functions/classes)
  - End-user modifiable, exploration-safe
- Opens up possibilities for new tools
  - Browsers, validators, code generators, etc.


That's all Folks!