Interactive Analysis in ATLAS
(thoughts on Analysis Model)

Amir Farbin
CERN

Overview

- Definitions/Disclaimers
- Motivation: Concrete example of issues we are trying to address: a *BaBar* analysis
- Analysis in ATLAS Today:
  - Event Data Model
  - Framework (Athena)
  - Interactive Athena (w/ some examples)
  - Analysis Object Data: features and problems
  - Present Analysis Model
- Some Recent Developments (EventViews/UserData)
- A vision of the Analysis Model with EventViews/UserData as basic elements and Interactive Analysis as a goal.

Definitions/Disclaimers

- Some of what I call *"interactive"* you wouldn't do interactively. I really mean python based analysis in analogy to Cint/ROOT analysis. In other words interpreted code + prompt.
- *Interactive Athena* (framework) means interactive access to Athena data, algorithms, services, etc… This is available now.
- *Interactive Analysis* adds basic analysis concepts to interactive Athena to help w/ analysis. We are working towards this…
- There is no official ATLAS Analysis Model, yet… some of us have some ideas. But not everyone is convinced so we are listening and accomodating requirements.
- Fundamental questions:
  - Is Interactive Athena just a "code proto-typing" environment or can "online validation" ➔ high level analysis be done on Athena prompt?
  - Is "Analysis Object Data (AOD) + Framework + Interactive Analysis + Distributed analysis" better than ntuplizing all data (like experiments before us)?

A BaBar Analysis (2001-2004)

To put you in the mind frame of the analyzer (ie *the* consumer of the analysis model), here's an example of a "modern" analysis:

- The measurement of "sin 2α" in B->$\pi\pi$ is (was?) a BaBar "flag-ship" analysis. It was very complicated (and technically challenging): rare decay, large bkg, PID (B->πK), flavor tag, tag vertex, ~100K event fit extracting 119 parameters.
- Often results were targeted for both summer and winter conferences every year. This was fueled by competition w/ other experiments.
- Reprocessed (reconstructed w/ updated algorithms/alignments) data available in bulk ~ 2-3 months before conference. Last data intended for conference ~ 1 month. Analyzers must understand differences due to reprocessing.
- Very rigorous 8 week internal review process… lots of required docs.
- Analysis was performed blind (to avoid bias): don't look at the data until permitted… don't look at the fit results until permitted.
- Typically the 2-3 analyzers spent 2-3 months preparing w/ MC, 2 months performing the analysis, and 1 month recovering.
- LHC analysis style/issues will be somewhat different and hopefully less stressful… nonetheless it will be demanding.
- My point: As consumers of Analysis Model, analyzers struggle w/ technical details everyday. Better model, better physics.

A BaBar Analysis (technical steps)

"Ntuple Production"

- **Job submission** (sh + perl scripts)
  - Query DB for event collections
  - Properly configure/run framework jobs
  - Keep batch queues populated w/ jobs
  - Resubmit failed jobs (can't miss 1 event)
  - Perform book keeping
  - Issues: unreliable services/resources, slow access to data, competition w/ in batch queues, disk space, …
  - In best case: 2-3 weeks to run over all data.
  - At least 1 FTE.
- **Analysis in Framework** (C++ & tcl)
  - Complicated algorithms: Particle ID, combinatorics, vertexing, kinematic fitting, flavor tagging, observable calculation
  - Well designed analysis tools meant that code rarely had to be modified…
  - But slow access to data meant that analyzers Ntuplized almost everything
  - Issues: BaBar could not store numerous ntuplized copies of data. Hard to repeat this step.

"Interactive Analysis"

- **Ntuple analysis** (PAW or ROOT)
  - Ntuple format dependent analysis framework.
  - 1st level Interactive analysis
  - Make final selections, plots, etc…
  - Export "flat files" of subset of observables.
  - Issues: different groups had different suites of analysis "macros". Could not go back to previous step of analysis (change algorithm/parameter to study effect).
- **Maximum likelihood fit** (LISP+PAW or RooFit)
  - High-level interactive analysis environment.
  - Build model of observables based on detector and physics ideas and try to extract parameters/measurements.
  - Produce toy experiments (in batch queues)
  - Fit data, control samples, Toy MCs
  - Make likelihood projection plots, tables
  - Issues: No connection to previous steps.

There are lots of issues… which ATLAS addresses in different ways. But note:

- Every step is in a completely different environment.
- Some reasonable questions could not be answered in simple ways.

Interactive Framework

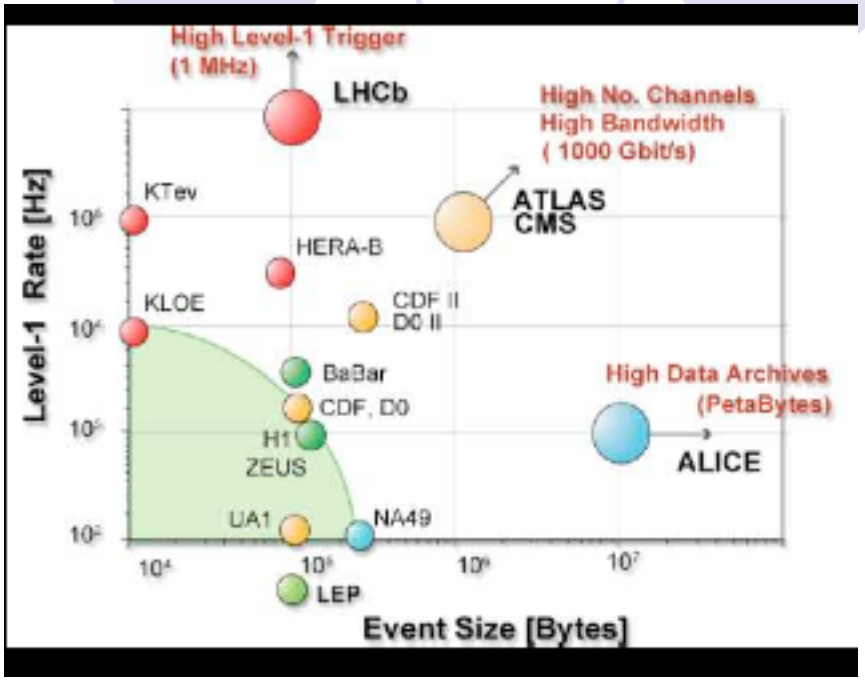
- General Idea: All of the steps shown in previous slide (and more) should be controlled/monitored from one environment.
  - And more: generation, simulation, digitization, reconstruction, analysis, event display, database interactions, online/offline monitoring, GRID services,…
- For the most part, this means that everything should use the same framework.
- But it also means that components which fall outside of framework (fitters, job submission, …) are pulled into a common environment as the framework.
- Finally, since interactivity is essential to analysis, this environment must provide a “prompt” and simple interfaces for accessing various components.

Goal: An Ideal Analysis Environment

- Physicists should be able to easily ask a “physics” question from the software.
  - Simple Ex: Compare the P_T distribution of leading jet for events w/ > 200 GeV of missing-E_T between Data, MC SUSY (signal), MC QCD (bkg), and MC top (bkg) samples… properly normalized.
  - Complicated Ex: I am updating my Higgs mass measurement. What part of the change in my fitted Higgs mass is due to new algorithms, new calibrations, new data, new analysis?
- We’re not talking natural language processing and AI…
- But such question should easily translate into 1 to O(10) statements… which in principle should be entered at a prompt.
- Next, I’ll tell you about analysis in ATLAS today…
- Then, I’ll present some ideas of how try to get from today to the ideal.

The ATLAS Data

- ATLAS will record ~3 PB of RAW data a year.
- Physics requirements and computing realities are balanced to determine data distribution and detail available for analysis.
- Identified data types (Event Data Model):
  - **RAW**: 1.6 MB/event.
  - **RDO**: Raw Data Object. Object representation of RAW. Persistified for simulation, transient only for data.
  - **ESD**: Event Summary Data. Collection of output of the various reco algs. Includes hits/cells. Intended for calibration. Limited access (tier 1). 500 KB/event.



- **AOD**: Analysis Summary Data. Analysis specific objects. Derived/Slimmed from ESD. Available to all (tier 2). 100 KB/event.
- **TAG**: Event summary data for fast selection. 1 KB/event.
- **DPD**: Derived Physics Data. Ie ntuples, histograms, UserData.

Athena Framework

- Enhanced version of Gaudi.
- Relevant Features:
  - Abstract Interfaces for Algorithms, Tools, Transient Data Stores, Services, Selectors, Converters, Properties, Utilities.
  - “athena” is a python script which loads the application manager using py bindings to core Athena abstractions.
  - Everything is loaded dynamically.
  - Data and algorithms are separated as design principle.
  - Transient data store: StoreGate. Persistified through POOL. Back-navigation allows referencing objects in different files (eg AOD -> ESD).
  - Allows access to python prompt.

Typical (“non-interactive”) Athena Job

- Algorithms, tools, etc written in C++, configured in py.
- Any athena job (eg: generation, simulation, digitization, reco, analysis) is defined by some python ***jobOption*** file which says:
  - Load *these* libraries
  - Read in *this* data
  - Run *these* algorithms in *this* order
  - Configure each algorithm in *this way*
  - Output *this* data
- Algorithm execution order, configuration, input/output done at run-time.

Interactive Athena

- So far, we have a limited set of tools which augment the athena environment to make make basic interactive analysis possible
  - Reflection dictionaries generated for data persistified in POOL. So we have python bindings to data through **PyIcgdict**. (Move to new py bindings when available)
  - **PyRoot** for histogramming/ntupling
  - "**PyGate**" interface to retrieve objects from StoreGate
  - **PyPoolSeek** for random access of events
  - **PyKernel** for interface to AIDA histograms.
  - **PyParticleTools**, **PyTriggerTools**, **PyAnalysisCore** for convenient access to AOD, Trigger, and Truth objects
  - Algorithms can be written purely in python.

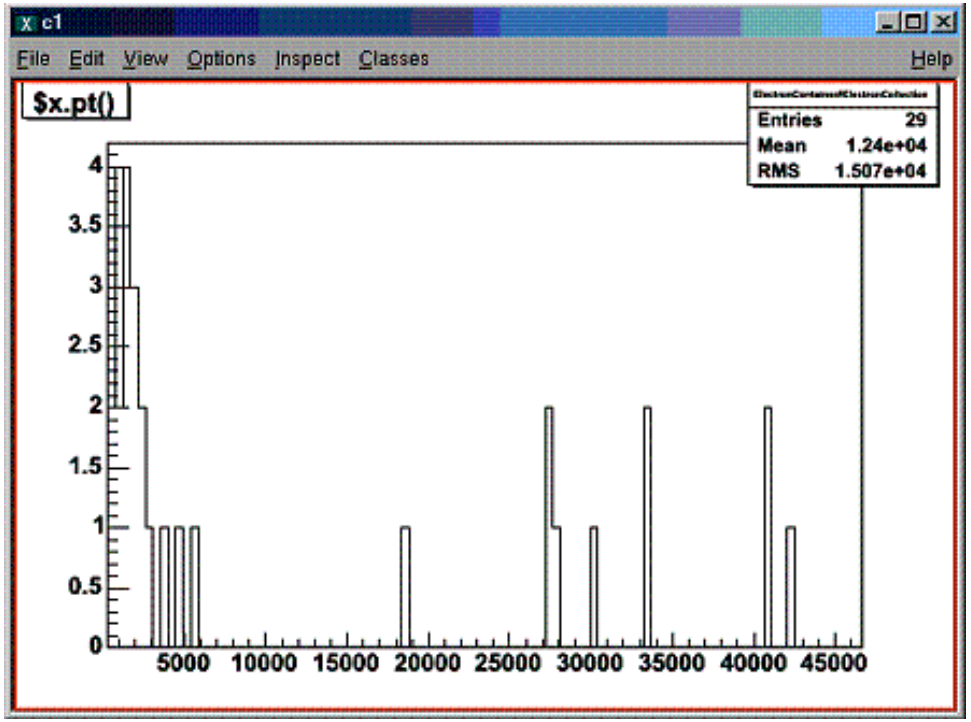
Interactive Athena Examples

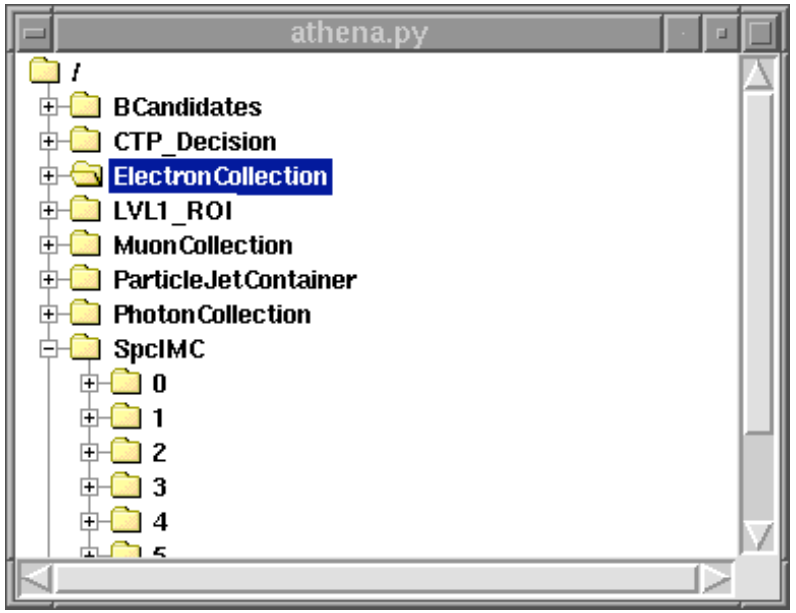
We can do basic stuff:

```
> athena -i Interactive_topO.py
athena> theApp.initialize()
athena> theApp.nextEvent()
athena> econ = PyParticleTools.getElectrons("ElectronCollection")
athena> len(econ)
4
athena> e = econ[0]
athena> dir(e)
['Electron', 'Navigable', 'P4EEtaPhiM', …]
athena> e.pt()
1978.4287625828167
athena> tp = e.track()   # Accessing track object
athena> dir(tp)
['=', 'NavigableTerminalNode', 'P4PxPyPzE', …]
athena> tp.pt()
2216.3881545625945
athena> eg = e.eg()  # Navigating Back from AOD -> ESD
athena> eg.e()
11949.0
```

Interactive Athena Examples

```
athena> plot("ElectronContainer#ElectronCollection","$x.pt()",nEvent=5)
```





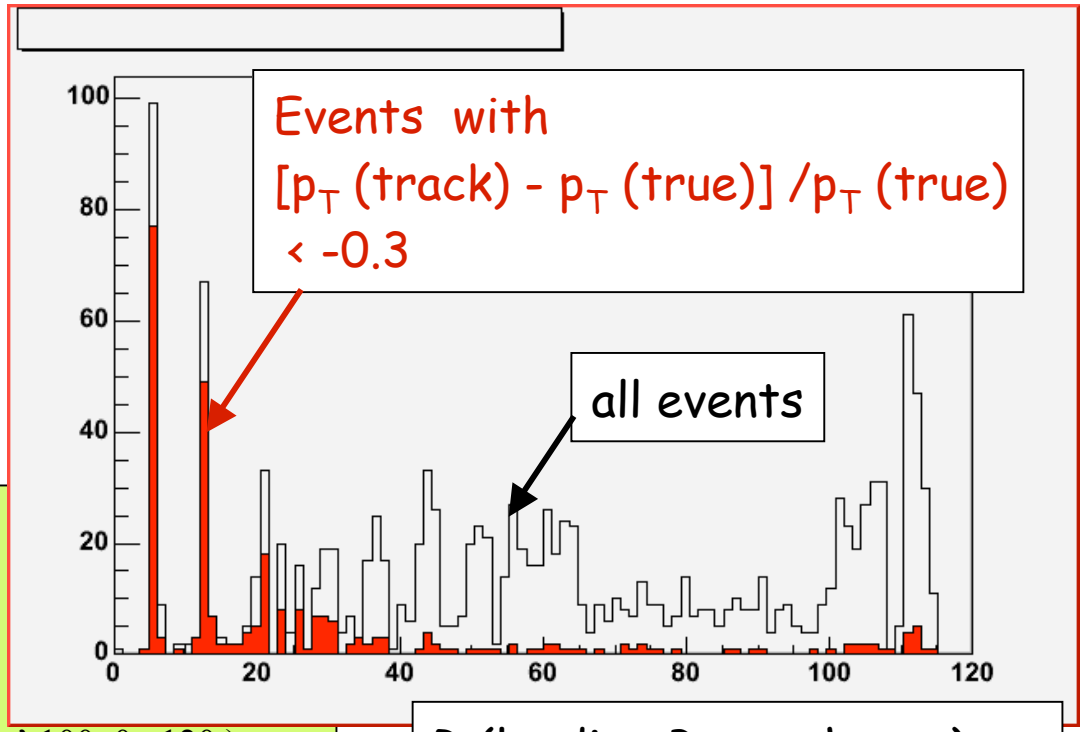

```
athena> include ("PyAnalysisExamples/PyPoolBrowser.py")
```

- So we can now interactively examine the contents of our POOL files!
- There is an effort to interactively QA output of generation, simulation, digitization, reco, …

Interactive Athena Examples

Python code

```
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)
#
```



Z → ee events (AOD)

Note: this can be done in an algorithm also…

Experience w/ Interactive Athena

- Several non-experts have tried interactive athena.
- One user (S. Gadomski) has compared execution time of C++ and python analysis code.

Variant of the analysis code	CPU time [ms/event]	overhead [ms/event]
SUSY group analysis in C++ without the "user" part	33.74	0.00
with user part in C++ (plug-in to SUSYPlot Algorithm)	34.15	0.41
with user part in a Python algorithm ala Scott	35.87	2.13
with empty Python algorithm ala Scott	33.84	0.10
with user part in a Python class, user event loop in athena -i	38.60	4.86

- User impressions:
  - Though interpreted python is slower… difference negligible compared to AOD object retrieval time.
  - Python analysis is “simple” and “powerful”
- Interactive Athena today is good for analysis end point histogramming and fast algorithm prototyping.

Analysis Model (Today)

- We have the basic ingredients for analysis:
  - Framework where physicists can write analysis algorithms/tools, make ntuples/histograms, etc…
  - EDM: AOD and ESD w/ back-navigation
  - Particle objects in AOD (more on this next slide)
  - Some analysis tools for combinatorics, neutrino reco, …
- We would like physicists to perform as much of their analysis as possible in Athena, rather than dumping ntuplize copies of the AOD. Besides being able to do better analyses in Athena, we cannot afford to store the ntuples.
- The AOD was very successfully used by almost all analysis groups to present results at our Rome Physics Workshop last month.
- The general trend was to write all analysis code into one C++ algorithm which dumped histograms or an ntuple. Many analyses could benefit from some more modularity.
- Some analysis packages created by groups, but they were either too complicated or failed to benefit from some basic features of our framework.
- There are some problems which must be addressed before we have the “ideal” interactive analysis environment…

A bit about the AOD

- AOD mostly stores collections of IParticles
  - Have 4-momentum representation, origin, PDG ID
  - Have constituents (reco obj in ESD or other particles)
  - Different inherited classes for Electron, Photon, Muon, TauJet, ParticleJet, BJet, Composite, Neutrino each with customized content.
- AOD particles are built by copying information from and linking to reco'ed obj in ESD.
- AOD versions of tracks and calo clusters (coming soon) are also kept.
- Possible operations on AOD: Now: Vertexing; Soon: B-Tagging, Missing E_T, Jet Clustering
  - Important principle: Apply complicated algorithms which analyzers might need to tweak on AOD.

Present Problems w/ AOD Analyses

- It's too slow! 50k events in 1 hour.
  - An effort has begun to optimize. Factor 10 is not inconceivable.
  - Distributed analysis might help.
  - But if AOD is the input to an interactive analysis, this needs to be at ntuple speed (1000 X faster).
  - Difficult to read data, create objects, at ntuple speeds.
- Overlaps between particles are not removed:
  - Ex: Since the tau, electron, and jet reco algs all start from calo cells, the same object in the detector will be reco'ed as a tau, electron, and jet.
  - This is actually considered a feature, since different analyses will have different requirements.
  - But every analysis starts w/ removing overlaps.
- Once you write ntuples, you loose your connection to AOD.
- Different AOD particle types are stored in different containers which don't inherit from each other.
  - Difficult to write generalized algorithms for modular analyses
  - Ex: Cannot write an algorithm which find all combinations of N particles consistent w/ a certain mass.
  - Symlink solves this problem, but causes other.

Problems w/ Current Analysis Model (restated)

- Can't access the AOD fast enough for interactive plotting.

➔Must store ntuples to store observables.

- No connection from ntuple to AOD.
- AOD particles overlap

➔Must remove overlaps and keep track of results.

- Difficult to write generalized tools which work for any particle type.

A solution: EventViews

- EventView (EV) is a collection of reconstructed objects that are coherent, mutually exclusive, exhaustive, i.e. the particles that don't overlap.
- Keep true to Athena principles, we have separated EV EDM and EV building. Building EV can be very complicated, so we allow users to decide how to fill their EV.
- EV Features (so far):
  - Keeps links to AOD particles (FinalStateObjects). EVs back-navigate to AOD. Keeps links to particles created during analysis (InferredObjects).

  ➔*Solves overlap problem.*
  - Have a parent EV… users can build EV trees.
  - Serves as proxy to StoreGate (eases retrieval, looping, and symlink).

  ➔*Solves generalized algorithm problem.*
  - Store for UserData: ability to dynamically add templated user-defined data which fast accessible (ntuple replacement). Simple interface:

    `"EV->setUserData<double>("Blah",5.)"`

  ➔*Addresses fast access/ntuple->AOD connection problems.*
  - EV is persistified in POOL.
  - UserData can be persistified into POOL, Tag, and/or AIDA (ROOT ntuple).
  - We have one implementation of EV builder…
- Upcoming EV Features
  - Parallel/child EV creation
  - Particle labeling

Hierarchical EventView Building.

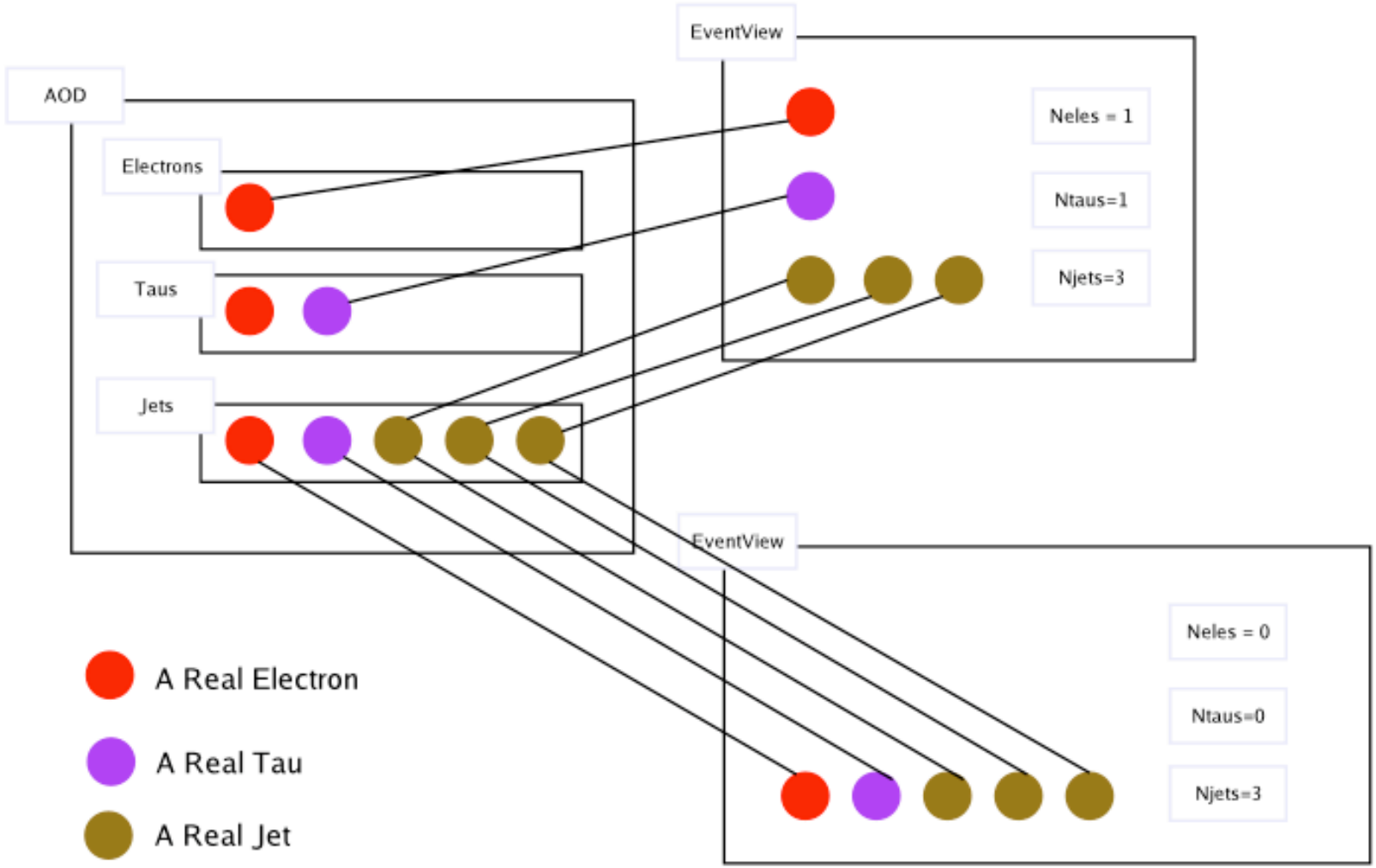
One implementation of EV Building:

```
defaultEV.EventViewTools=[
    "EVElectronInserter",
    "EVPhotonInserter",
    "EVMuonInserter",
    "EVTauJetInserter",
    "EVParticleJetInserter",
    "EVMissingEtUserData"
    ]
```

```
alternateEV.EventViewTools=[
    "EVParticleJetInserter",
    "EVTauJetInserter",
    "EVElectronInserter",
    "EVPhotonInserter",
    "EVMuonInserter",
    "EVMissingEtUserData"
    ]
```

```
toolSvc.EVElectronInserter.ContainerKey="ElectronCollection"
toolSvc.EVElectronInserter.etCut=10*GeV
toolSvc.EVElectronInserter.deltaRCut=.1

toolSvc.EVPhotonInserter.ContainerKey="PhotonCollection"
toolSvc.EVPhotonInserter.etCut=10*GeV

toolSvc.EVMuonInserter.ContainerKey="MuonCollection"
toolSvc.EVMuonInserter.etCut=10*GeV

toolSvc.EVTauJetInserter.ContainerKey="TauJetCollection"
toolSvc.EVTauJetInserter.etCut=10*GeV
toolSvc.EVTauJetInserter.likelihoodCut=4.

toolSvc.EVParticleJetInserter.ContainerKey="ConeTowerParticleJets"
toolSvc.EVParticleJetInserter.etCut=10*GeV
```

EventView Builder Example



EventView for Analysis Bookkeeping

- Building EV is likely the first step to any analysis.
- Users build hierarchy of EV as they “refine” their view:
  - Create Parallel EV when deferring decisions. Ex: I can’t decide this is a tau or a jet, so I consider both.
  - Create Child EV to reflect a different combinatoric choice
- UserData
  - Association of UserData w/ EV is natural, since in analyses observables are usually only valid w/ in a view.
  - UserData keep the hierarchical structure of EV. If I don’t have a variable, maybe my parent does.
  - UserData provides a mechanism of communicating information between algorithms.

EventView as Basic Analysis Object

- Example Analysis Flow (within an Event)
  1. Build fundamental and/or parallel EVs
  2. Calculate Event variables (N Jets > 100 GeV, event shape, etc …) for each EV, attach as UserData
  3. 1st Selection on EV using UserData
  4. Perform Combinatorics to reco W's, Z's, H, etc … until you build specific signature.
  5. Calculate/store observables (masses, angles, kinematics, detector info, etc…) into UserData.
  6. Make final selections, optimize cuts, perform studies, make plots using UserData.
  7. Navigate back EV -> AOD -> ESD for interesting events.
- Note that User may write out EV at any point and continue the analysis in another session. In fact, 1 and 2 could be performed by the analysis group (Ex: SUSY EventView).

Developing A New Analysis

- We have plans of standard interfaces and generalized implementations of tools for basic EV operations: build views, selections, combinatorics, observable calculation, …
- We imagine 3 different possible implementations of the same tools based on same abstract interface:
  1. Pure C++, configurable in py through jobOptions
  2. Pure python… jO configurable
  3. C++ w/ python call backs… jO configurable
     - Ex: Tool which performs combinations in C++, but interprets some py (passed as jO) to determine whether or not to keep the candidate.
- Analyses will be comprised of a combination of code written in 1, 2, and 3.
- Ideally, generalized tools are either type 1 and 3 in order to be fast… and when the generalized tool does not meet a users specific needs, he/she will only need to write type 2.
- This is the “Analysis in Framework” step in BaBar Example.

Developing Analysis in the Framework

- General procedure:
  - Use standard tools when possible.
  - If tools doesn't exist, write your own.
  - Put them together in right order. Properly configure them.
- Some basic analysis tools users will use/develop in course of analysis:
  - EV Builders: If necessary specialized EV particle adder or EV builder.
  - Composite particle builders
  - Observable Calculators (UserData): Cache AOD/ESD info, masses, kinematics, Neural Network, …
  - Selections: Based on UserData?
- With standard interfaces, the physics community will hopefully develop libraries which can be shared.

Performing Analysis On UserData

- Persistency technology not decided (ask me for details, if you like)
  - POOL is too slow.
  - Tag (AttributeList) cannot save arrays.
  - Ntuples do not keep connection to framework.

  We have a plan for how to proceed.
- UserData is really an ntuple, which may be read directly in ROOT… we would like to support the “laptop ROOT analysis on plane” scenario.
- Typical ROOT based analysis start w/ a framework for keeping track of different chains of ntuples, looping (TTree::MakeClass()), making plots, ...
- We plan to provide a python analysis environment which
  - Keeps track of UserData datasets in a matter consistent w/ EventSelector.
  - Provides “EventLoop” and “Algorithm” similar to Athena Framework.
  - Provides convenient interfaces. Imagine: Samples(“signal”,”bkg”).Draw(“JetpT[0]”,”MET>200”)
  - In addition, users should be able to back-navigate, entering the standard Athena event loop while examining UserData. They should be able to recalculate UserData.
- This is the replacement for “Ntuple Analysis Framework” in BaBar Example.

Basic Vision of Analysis Model

- Recalling the steps in BaBar Example
  - Job Submission, data management, etc: Python is a scripting language. ATLAS is providing services for this (Distributed Analysis?). Much of this is python based already.
  - Framework Analysis: EV can be basic object passed between modular parts of the analysis. Users will mix standard tools with their own python and C++. They can develop/configure their analysis and write code “interactively” in python.
  - “Ntuple” Analysis: UserData will serve as ntuple with links back to EV->AOD->ESD. A lightweight python framework will mimic EventLoop and Algorithms, provide convenient tools, and allow easy back-navigation to Framework Analysis and recalculation of UserData. User may also perform ROOT based analyses.
  - High Level Analysis: Access to such packages (such as fitters) provided through python bindings.
- So everything can be done in a single python session.
- Ideally, users can write py scripts which mix these steps to answer complicated questions.

Final Words

- Most fundamental elements for Analysis Model is available.
- Much of the other elements are either already available, or soon will be.
- Key element to model: We imagine most interactions w/ software will occur in python… “Interactive Analysis”
- No matter how smart we think we are, we cannot think of every conceivable problem/requirement.
- “We” here are actually users. We are developing this so we can perform our own analyses…
- Therefore, we plan on a process of
  - Delivering a certain set of features to other users
  - Providing instructive examples of real analyses (our own analyses)
  - Providing documentation and tutorials which teach how to perform analysis w/ in the model.
  - Identify individuals w/ in each analysis group who will use, test, and comment. Regular meetings to discuss issues/ideas from User community.
  - Iterate…

What needs to be done

- Finish EventViews
- Figure out UserData persistency
- Put clusters in AOD, build jets in AOD
- Develop overlap tools
- Bring in higher level analysis tools into python. Ex: RooFit (easy, since it's in ROOT).
- Make EventSelector keep track of multiple samples.
- Create the UserData analysis environment
- Give lots of tutorials.