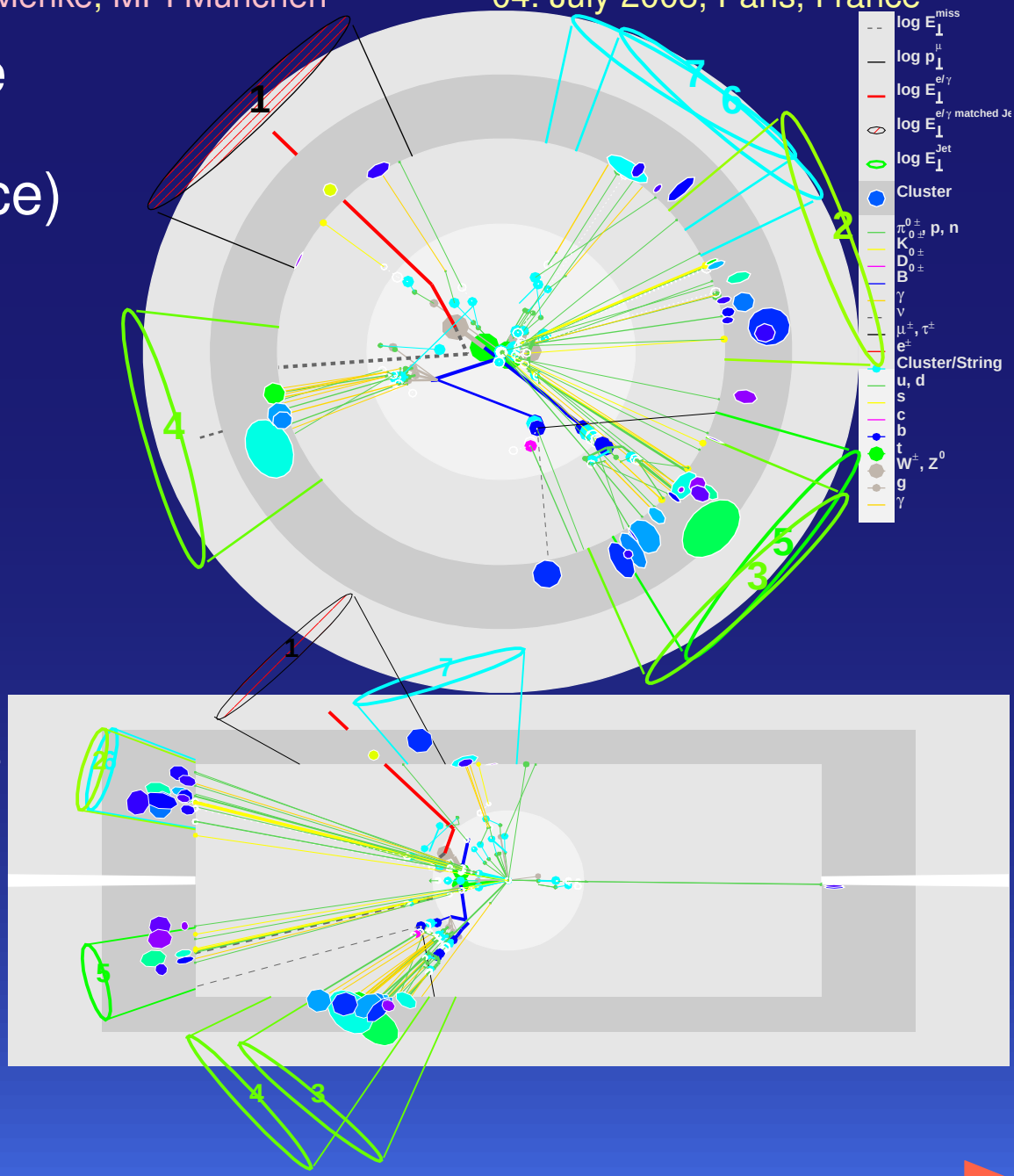
Physics Analysis Tools

2nd Artemis Annual Meeting Sven Menke, MPI München 04. July 2008, Paris, France

- PAT Developments since last anual meeting (September 2007, Greece)
  - Analysis Model and DPDs
  - `UserDataSvc`
  - `PyAthena`
  - `AthenaROOTAccess` Analyses
  - `AtlasAnalysis` release
- EDM Developments
  - `Jet`/`ParticleJet` merger
  - simple `stl::vector<double>` user data
- Introduction to ARA tutorial



- Analysis Model
  - based on the Analysis Model Forums an AMF report (ATL-GEN-INT-2008-001) has been approved
- DPDs
  - Derived Phyiscs Data in form of `POOL/ROOT` files (like small AOD's) will be the what you run your analysis on.
  - Small DPDs are condensed from AODs by
    - `skimming`: filter out unwanted events
    - `thinning`: filter out unwanted collections and objects inside kept collections
    - `slimming`: filter out unwanted properties of kept objects
  - they contain `UserData` (results from complex analyses on AOD level)
  - DPDs replace CBNT/SAN/n-tuple

- AthenaROOTAccess
  - allows to access ESD/AOD/DPD directly from ROOT via python or compiled C++
  - shields high level analyses from athena details
  - still provides you with all the structure of the atlas EDM
    - you get objects like Jet, Electron etc. with rich set of methods
    - https://twiki.cern.ch/twiki/bin/view/Atlas/AthenaROOTAccess by Scott Snyder et al.
- Toolkits and Analysis Frameworks
  - tools need to be modular such that duplicate development is avoided
    - dual use tool base class allows to create tools usable in athena and in AthenaROOTAccess
    - frameworks like EventView, EWPA, AthenaROOTAccess are clients of these common tools
  - tool validation is mandatory
    - RTT tests, best independent of particular framework are an excellent way to perform this
  - input to analysis frameworks
    - selection, cuts, overlap checks etc. need to bw developed in close collaboration with performance groups

- UserDataSvc
  - motivation is to add results of complex analysis or objects/information you can't reproduce from the objects stored on the DPD as additional data objects on the DPD
  - solved in rel. 14 with the UserDataSvc (Paolo Calafiura, Yushu Yao et al.)
  - entire events or SG objects can be "decorated" with the UserDataSvc
  - The decoration consists of a label (e.g. HiggsMass) and an object (virtually anything that can be put into a TTree) – in this case a simple float would do ...
  - The decorated object can be associated with other SG objects
  - decoration info (label, actual decoration object, associations) is written into a TTree in the same file the CollectionTree ends up in
  - reading the file back in with athena or ARA allows access to the decorations
  - initial problems with skimming are being solved (already implemeted; testing is underway)
  - nice documentation and examples here: https://twiki.cern.ch/twiki/bin/view/Atlas/UserDataSvc

- New PyAthena framework by Sebastien Binet et al. in Control/AthenaPython improves the way python algorithms are treated in athena
  - like the C++ algorithms the configuration step and creation step are separate now for python algorithms
  - this is important to be fully Configurables compliant, increases speed and debugging efficiency
- Documentation and Tutorials
  - extensive wiki page: https://twiki.cern.ch/twiki/bin/view/Atlas/PyAthena
  - Please try it and provide feedback to Sebastien!
- Utilities provided by PyAthena
  - class lookup in PyAthena like PyCintex or ROOT
  - retrieve services and tools via PyAthena.py_svc and PyAthena.py_tool
    - this is fantastic for interactive athena!
- StoreGate improvements
  - py_retrieve now is only 50% slower than C++ retrieve from StoreGate
  - py_record about 100% slower
  - both bound by Reflex/PyRoot overhead
  - now both fast enough for full analysis in python!

- AtlasAnalysis release
  - packages in AtlasAnalysis by their nature depend on many EDM and other core areas of athena
  - to provide full functionality stable running conditions in underlying software is needed
  - in the past this was only possible in the final nightlies prior to a full release build
  - that was not sufficient to provide the desired functionality
  - starting now (with rel. 14.3.0) we will get a partial release (14.3.1) based on the latest major release to develop the software under the AtlasAnalysis project.
  - The partial release will have nightlies, and a production Cache that can be installed on the grid
  - interim solution is a migration nightly AtlasAnalysis-14.2.X-MIG1 under tag approval mode by Stathes and me
    - please commit your new tags here for testing

Event Data Model Developments

- `Jet`/`ParticleJet` merger
  - in rel 13 each `JetCollection` from the ESD was converted to a `ParticleJetContainer` on the AOD
  - the `ParticleJet`s held only the 4-vector of the `Jet`, the b-tagging info and a pointer to navigate back to the ESD `Jet`
    - analysis code had to distinguish between ESD and AOD
    - on AOD all info about the jet constituents (`CaloCluster`s, `TruthParticle`s) was lost
    - the b-tagging info was misnamed constituent
  - now the `Jet`s on AOD and ESD are identical
    - analysis code runs unchanged on ESD and AOD
    - constituents of the `Jet` are present on AOD if the underlying constituent container is on the AOD (`LCTopo`, and `Truth` jets)
    - for `H1Topo` jets the underlying cluster collection is not present on the AOD, but the constituent indices point to the corresponding `CaloCalTopoCluster`
  - `JetRec` can now run on AOD's to make any kind of `LCTopo` or `Truth` jet (all stable MC particles are now on the AOD as well)
    - allows for the first time real jet studies on the AOD
- simple user data
  - in addition to the `UserDataSvc` which allows for associations between `StoreGate` objects R.D. Schaffer, Sebastien Binet et al. are currently testing a simple alternative:
    - direct storage of `double`, `int`, `std::string`, `std::vector<double>`, `std::vector<int>`, `std::map<int,double>`, `std::map<std::string,double>`, etc.
  - those objects will be directly on the same `TTree` as other normal `StoreGate` objects and are thus easier to deal with in terms of skimming, and access in ARA

AthenaROOTAccess Tutorial Introduction

- What is `AthenaROOTAccess`?
  - `AthenaROOTAccess` allows you to access the objects in ESD/AOD/DPD directly from `ROOT` without the `athena` framework
  - Many athena classes (most notably the classes describing the transient objects) are available from `ROOT` and `PyROOT` via their dictionaries
  - The `athena` software has to be installed and setup but instead of `athena.py` you run `python -i` or `root`
- How does it work?
  - Minimal `test.py` start script for one AOD:

```
import user
import ROOT
import PyCintex
import AthenaROOTAccess.transientTree
f = ROOT.TFile.Open ('AOD.pool.root')
tt = AthenaROOTAccess.transientTree.makeTree(f)
```

  - sets up a virtual transient tree with branches corresponding to the transient objects identified by their `StoreGate` keys
  - The transient/persistent converters (`TPCnv`) are automatically invoked when a specific entry is requested to convert from the persistent data on the ESD/AOD/DPD to the transient representation
  - for example the branch `CaloClusterContainer_p4_CaloCalTopoCluster` in the persistent `CollectionTree` on the AOD will trigger the creation of the branch `CaloCalTopoCluster` in the transient tree which points to the transient `CaloClusterContainer`

- Minimal `chain.py` start script for many AODs:

```
import user
import ROOT
import PyCintex
import AthenaROOTAccess.transientTree
CollectionTree = ROOT.AthenaROOTAccess.TChainROOTAccess('CollectionTree')
CollectionTree.Add('AOD.*.pool.root')
tt = AthenaROOTAccess.transientTree.makeTree(CollectionTree)
```

► Examples

- `python -i chain.py`

```
tt.Draw('CaloCalTopoCluster.e()')
ce = ROOT.ClusterExample()
ce.plot(tt)


cc = tt.CaloCalTopoCluster

tt.GetEntry(0)
cc.size()
c = cc.at(0)
c.e()
```

- `root`

```
TPython::Exec("execfile('chain.py')");
CollectionTree_trans->Draw("CaloCalTopoCluster.e()");
ClusterExample ce;
ce.plot(CollectionTree_trans);
TBranch * cbr = CollectionTree_trans->
                GetBranch("CaloCalTopoCluster");
const CaloClusterContainer * cc =
   *((CaloClusterContainer **)cbr->GetAddress());
cbr->GetEntry(0);
cc->size();
CaloCluster * c = cc->at(0);
c->e();
```

► Typical development cycle

- Either use `python` or compiled `C++` to develop your analysis
- Use `CINT` only to execute python script and instantiate compiled `C++` classes
- in `python` no compilation needed but typically 2 times slower than compiled `C++` code
- Compiled `C++` code needs to come with a dictionary to be visible from `CINT` and `python`
  - ► typically as easy as adding the class to the `selection.xml` and `XXXDict.h` files in the package
  - ► lookup examples in `PhysicsAnalysis/AthenaROOTAccessExamples`
- To get started check out `PhysicsAnalysis/AthenaROOTAccessExamples` and add a class ...
- Going back and forth between `athena` and `AthenaROOTAccess` is easy since only the retrieval of the containers differs (`StoreGate` vs. `TTree`)
  - ► can develop analysis in `AthenaROOTAccess` and port with almost no effort to `athena` later

► Limits

- need special dual use tools to run in both `athena` and `AthenaROOTAccess`
- databases, detector description, identifiers, services are not available in `AthenaROOTAccess`
  - ► code that needs these has to reside in `athena` and make output available on DPD (maybe as `UserData`) which can be analyzed in `AthenaROOTAccess`