Event Data Model in ATLAS

Edward Moyse,

(University of Massachusetts Amherst)

CHEP 2006,

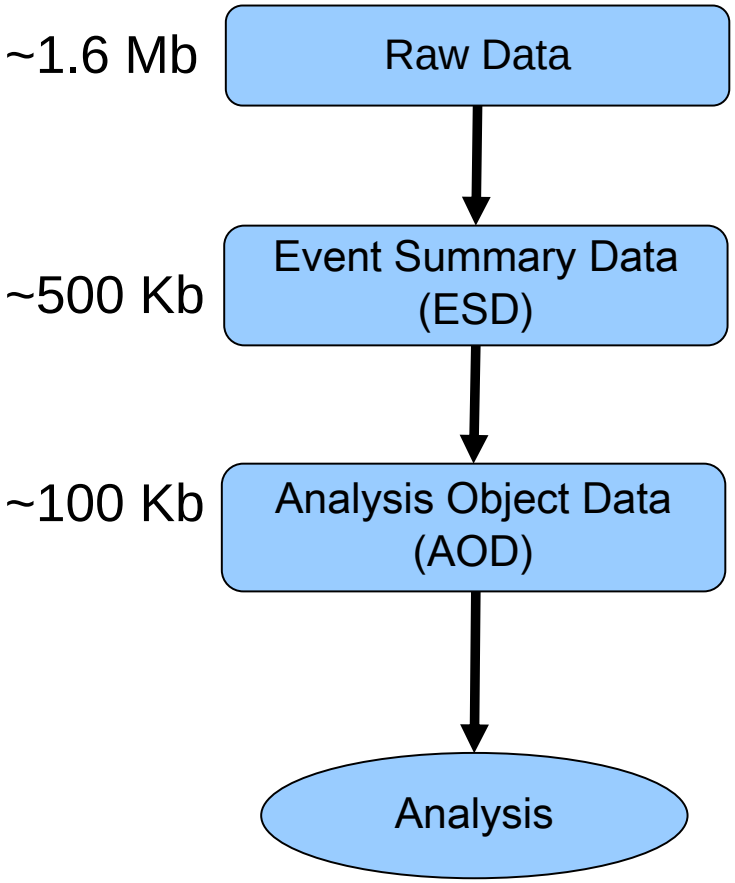
Mumbai

Introduction

- *"For large collaborations like the ATLAS experiment common interfaces and common data objects are a necessity to insure easy maintenance and coherence of the experiments software platform over a long period of time."*

- This talk will give an overview of how the ATLAS EDM is constructed, what the constraints on it were, and what the benefits of this EDM are.
- All sub-detectors are touched on, but I will not have time to go into detail, but will concentrate instead on recent developments.

Computing Model



- Amount of data per year: ~ **1PB**
  - Cost of distribution and storage is a key concern
- Subsets of full data distributed to remote sites.
  - The **E**vent **S**ummary **D**ata (**ESD**) is intended to contain the detailed output of reconstruction, and will contain sufficient information to allow
    - Particle ID, track re-fitting, jet calibration to be re-done.
    - Allows fast turn-around for 'tuning' of algorithms and calibrations
  - **A**nalysis **O**bject **D**ata (**AOD**) contains enough data for common analyses to be performed.
  - In addition, **TAGS** (at **AOD** level) indicate key features of events, allowing the rapid selection of particular event types.

Event Data Model

The ATLAS EDM has been shaped by many considerations:

- Must allow the correct level of modularity (**raw data** / **ESD** / **AOD** etc) to fulfil the computing model.

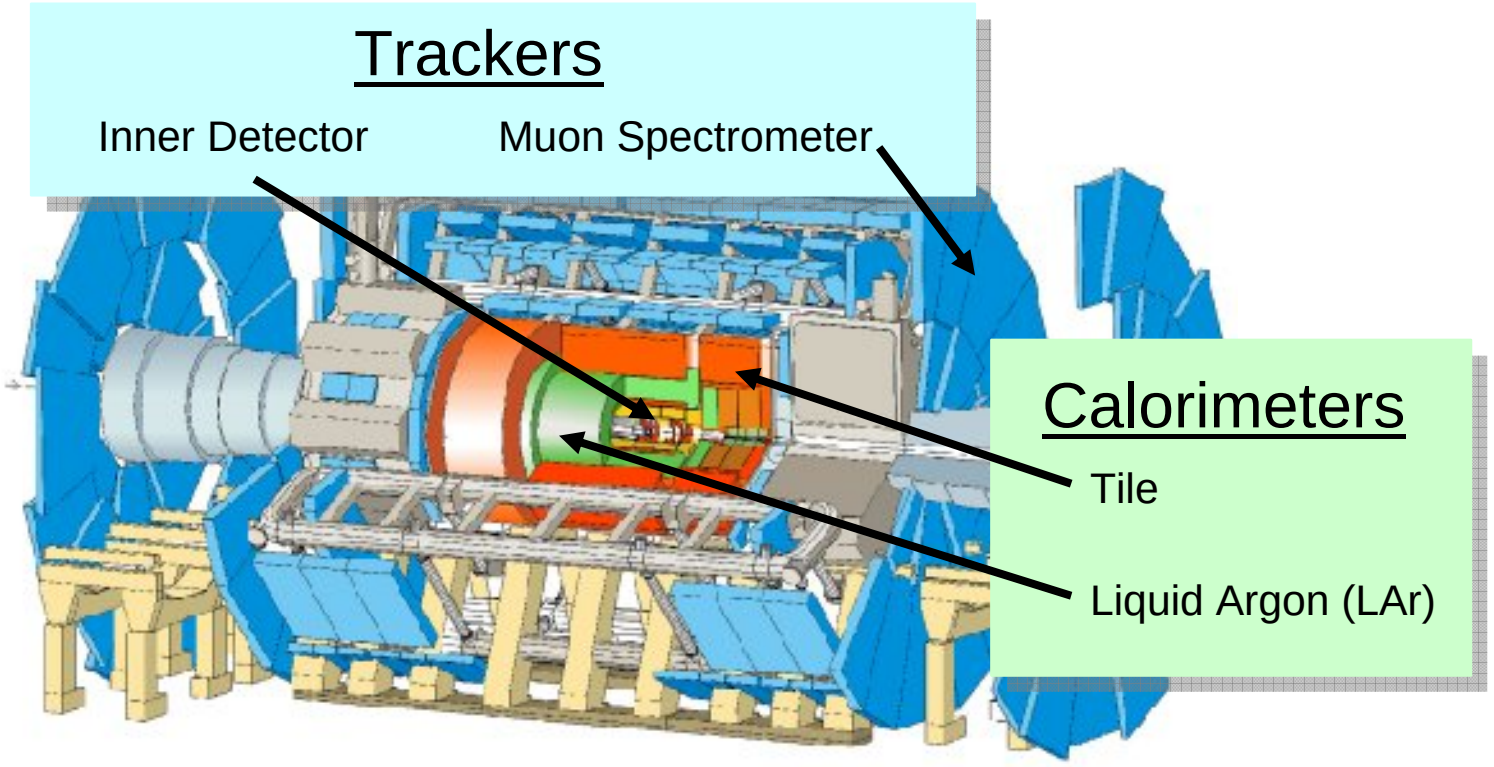
- ***(Obviously) must be able to encapsulate the required event data!***

- Should promote code re-use by:
  - Allowing the factoring out of common tools;
  - Sharing data classes:
    - Between offline reconstruction, and the online trigger
    - Between the sub-detector systems …
    - *…whilst minimising/preventing unnecessary dependencies*

Event Data Model (2)

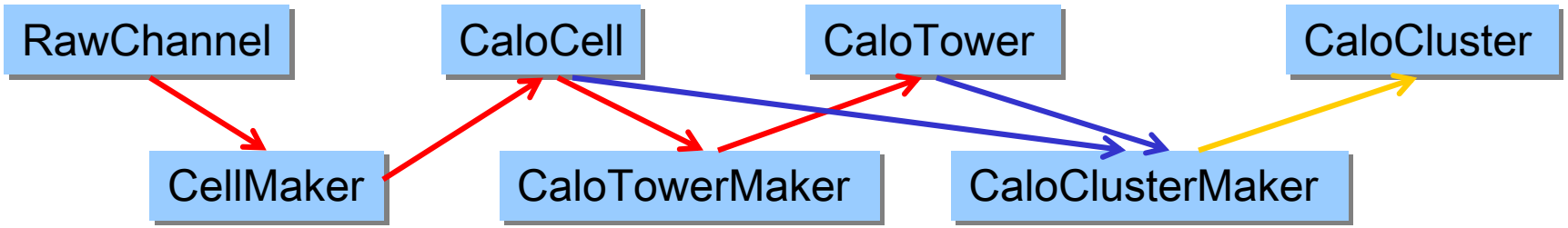
- Separation of Event/Non-event data:
  - For example, try to avoid having detector description (GeoModel) available in event data (normally use "Identifiers" instead)
  - Event and non-event data are even stored differently: ATLAS has StoreGate, (a transient event store), and DetectorStore (exists for whole run)
- The design of the EDM has been strongly shaped by framework requirements
  - The EDM must be "persistifiable" (ATLAS uses POOL to read/write data) which is a non-trivial requirement, and which excludes many possible EDM designs
  - (more on this later)
- Truth
  - Link EDM data objects and the simulated event.

Detector Sub-Systems



- Two types of detectors in ATLAS, trackers and calorimeters.
- The ATLAS EDM is designed to share as much code as possible, within the same sub-system type

Calorimeter

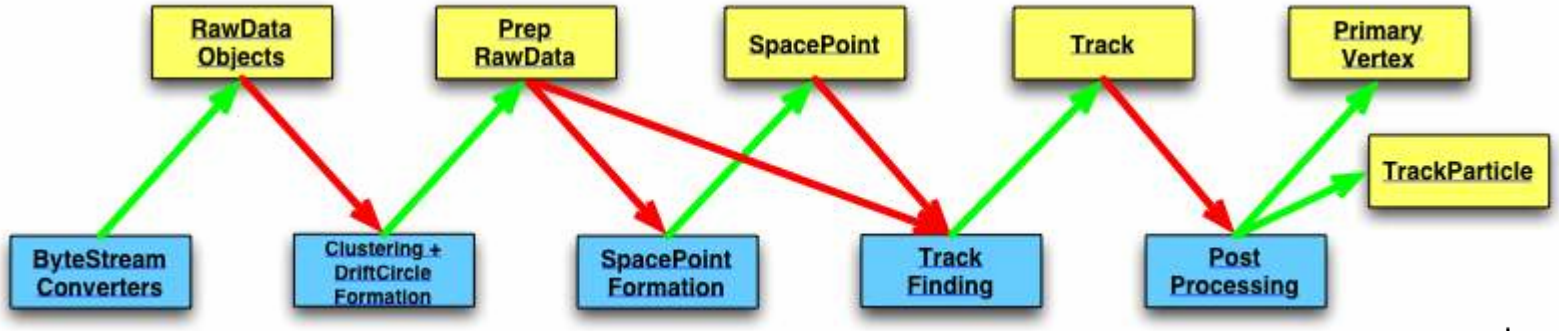


- LAr and Tile retain separate identities at raw data level but LAr and Tile converge at cell level:
  - Calibrated calo cells produced from either Raw Data or simulation
  - Calorimeter '*towers*' then produced from the cells… along with '*clusters*' which are collections of calorimeter elements (i.e. cells, towers, even clusters themselves)
  - Reconstruction Input Object for both calorimeter types are 'CaloCells' and 'CaloClusters', with CaloCells present in ESD, and CaloClusters present in both AOD and ESD.

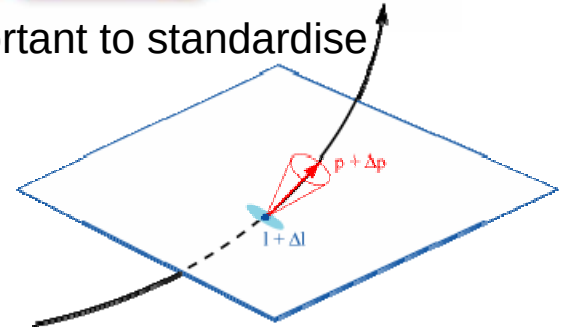
Calorimeter (2)

- Navigation
  - It is possible to retrieve the individual cells used to create any calorimeter object by using the “Navigation Tokens” to retrieve the desired type
  - I.e. using CaloCells as the token will return cell from e.g. an EnergyCluster
- Calorimeter data classes inherit from I4Momentum
  - Allows use of calo objects as input to generic algorithms
  - (Many analysis algorithms will only require a kinematic object to have a 4 momentum interface)
- Compactification
  - To reduce the size of the persistifed data, the calorimeter cells are compressed before being output to disk.
  - For more details, please see Poster : *“142 - The Calorimeter Event Data Model for the ATLAS Experiment at LHC”*

Tracking



- For Tracking to support two different sub-detectors, it is important to standardise more than just with a common Track and 'hits'
- We need common:
  - Track parameter definitions
  - Interfaces to clusters, drift circles etc.
  - Error matrices



- In addition, we need a clean way to handle differing coordinate frames, introduced by the use of many surface types in tracking.

Benefits:

- **Tools and Algorithms do not need to know specifics of detector.**
- **Generalised tools allow Tracking to work equally well on ID and Muons.**

Track

- One of the most important elements in the ATLAS EDM is the common **Track** (an **ESD** level object).
  - It must work in a wide range of applications, from
    - online (where speed is important)
    - alignment studies (which need detailed information)
    - … to reconstruction
  - Tracks at **ESD** level consist of fitted measurements (with Muon and ID concrete implementations, which derive from a common base clase) on multiple surfaces
  - It is the *output* from the fitters, and is the *input* to the combined reconstruction.
  - All reconstruction packages use the same track class.
- For **AOD**, something more lightweight is needed: **TrackParticles** are created from Tracks:
  - Contain summary information about parent track (number of hits on track etc)
  - Are physics analysis objects, with 4-momenta (the class inherits from I4Momentum – in general AOD objects inherit from I4mom, and IParticle etc.)
  - Can be used for vertex finding, but not re-fitting etc.

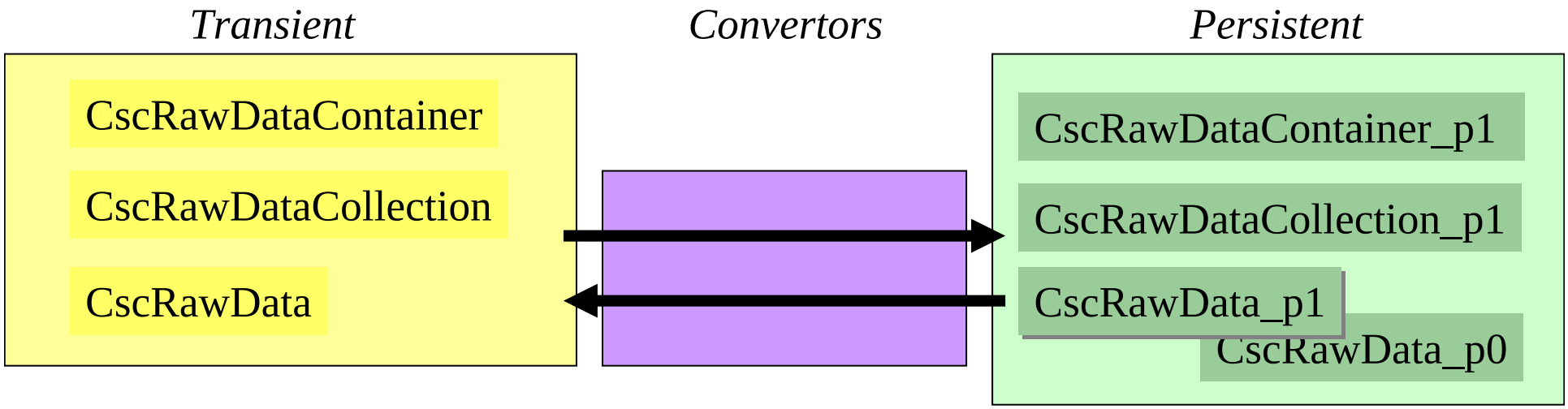
Persistency Issues

- Some problems with ATLAS' persistency mechanisms, namely:
  - 'Schema evolution' (see below)
  - Size (our ESD is too large for our computing requirements)
  - Performance issues reported with e.g. stl maps.
- **Schema Evolution** - a very brief overview follows:
  - When ROOT reads data back in, it first creates an object of the same type (using the default constructor) and then 'streams' data into the object
  - If the object that is created has a different 'shape' from the object that was written out (e.g. an int was removed from the class), then this streaming will not work - at *best* it fails, or crashes. Worst case : subtly corrupted data
  - This can happen very easily, and an accidental 'improvement' of a transient EDM class can make recently produced data unreadable.

Solutions

- Creation of an *'Event Management Board'*, to determine what is to be persistified, and in what format, and in general to give guidance to developers. Will use …
- New tools to automatically detect schema change in our nightly builds
- We (with the help of Marcin Nowak) are currently testing a new design: **two** EDMs
  - one for the transient world (i.e. designed for ease-of-use), and
  - one for the persistent world (i.e. designed for as compact storage as possible)
  - Aim to have raw data classes done by Release 12 (end of March)

An example...



- **To write out** CscRawData the convertors iterate through the container and collections, and produce a CscRawData_p1 for each CscRawData.
  - virtual voidtransToPers(const CscRawData* transObj, CscRawData_p1* persObj, MsgStream &log) const;
- **To read in**, the convertor creates CscRawDatas, and the collections and container which house them
  - virtual voidpersToTrans(const CscRawData_p1* persObj, CscRawData* transObj, MsgStream &log) const

Example (II)

- The trick is that the version on disk, the persistent object, has its version explicitly defined in the name.
  - In the example shown, there are two persistent versions of CscRawData, CscRawData_p0 and CscRawData_p1
- If we try to read old data into Athena, the object that is created is still the old type (CscRawData_p0) meaning we have no schema evolution.
- The convertors can now either pass this data to a specific method which handles the conversion to the current transient EDM, or fail gracefully.
- When we create the persistent models we can do some extra tricks
  - Design persistent classes to enable use of ROOT 'split mode' to optimise perfomance.
  - Compress the data
    - double to float, and
    - enums packed into bits.
  - Smarter compression, such as
    - multi-linear ranges to pack floats
    - Pack several ints into one etc
    - I.e. use knowledge of possible dynamic ranges to reduce ESD size.

Conclusion

- The transient EDM is rapidly stabilising
- Both in the calorimeter, and the tracking, the use of common interfaces has allowed the development of common tools (e.g. fitters which work on ID and Muon data, with little to no tweaking)
- More importantly, it appears to be fulfilling the design requirements (and has been tested on real data: cosmics and Combined Test Beam).
- However, the issue of persistency is obviously a big concern.
  - Being tackled both at the human level, with the creation of the ‘Event Management Board’
  - … and technically with, the proposed solution of having a transient/persistent EDM.