ATLAS Tile Calorimeter Testbeam Pilot Project

ATLAS Software Workshop

Geneva

2 September 1999

David Malon
malon@anl.gov

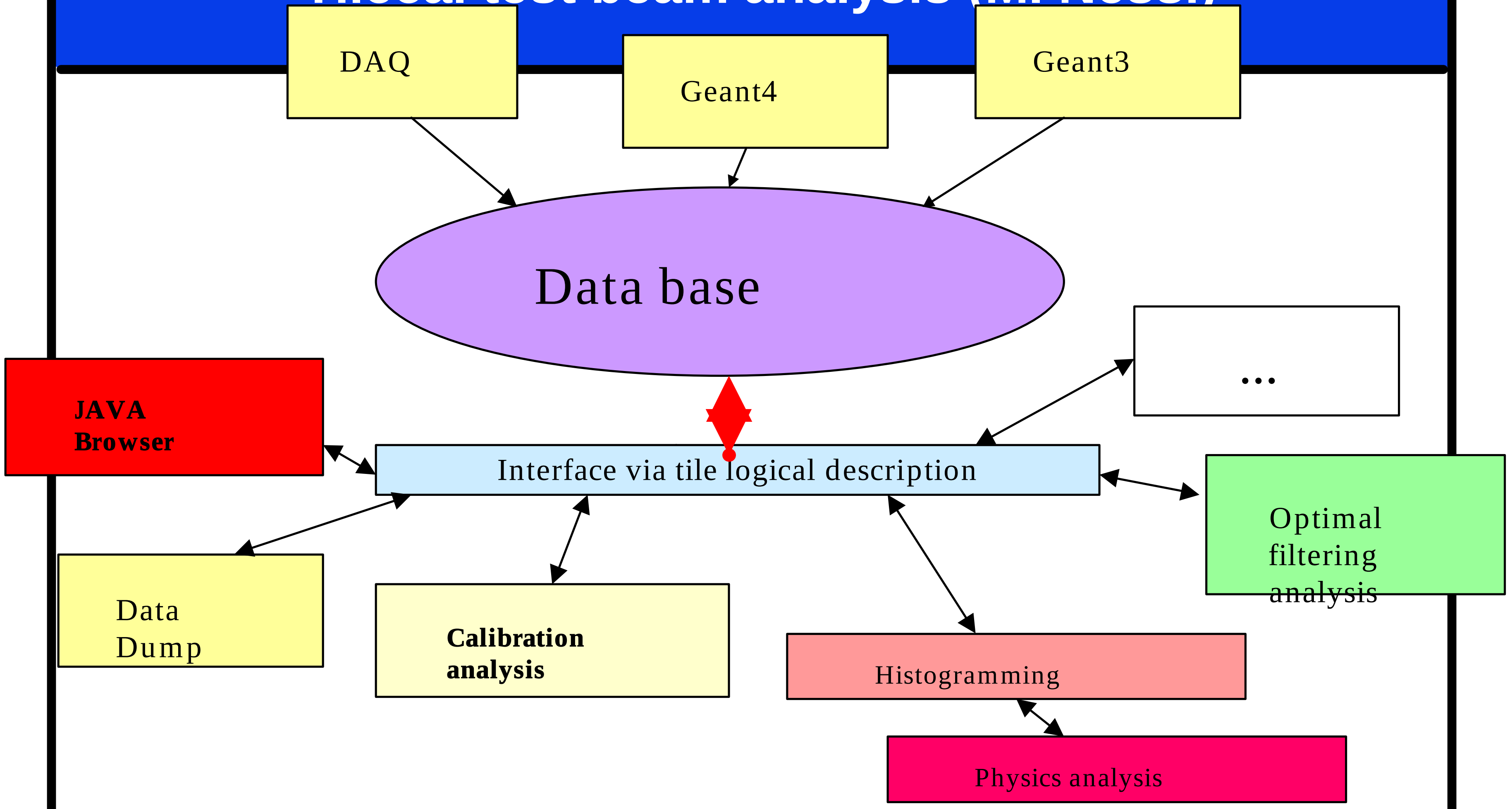
Goals and Background

- Support 1999 ATLAS tile calorimeter testbeam data analysis using candidate ATLAS object-oriented technologies
- Database foundation is the extensive work done by Sasha Solodhkov (Protvino) to put last year's raw testbeam data into Objectivity
- Serve as a testbed for ATLAS software strategies and technologies
- Planned since March 1999; begun late May 1999
- Testbeam period: 7-21 July 1999

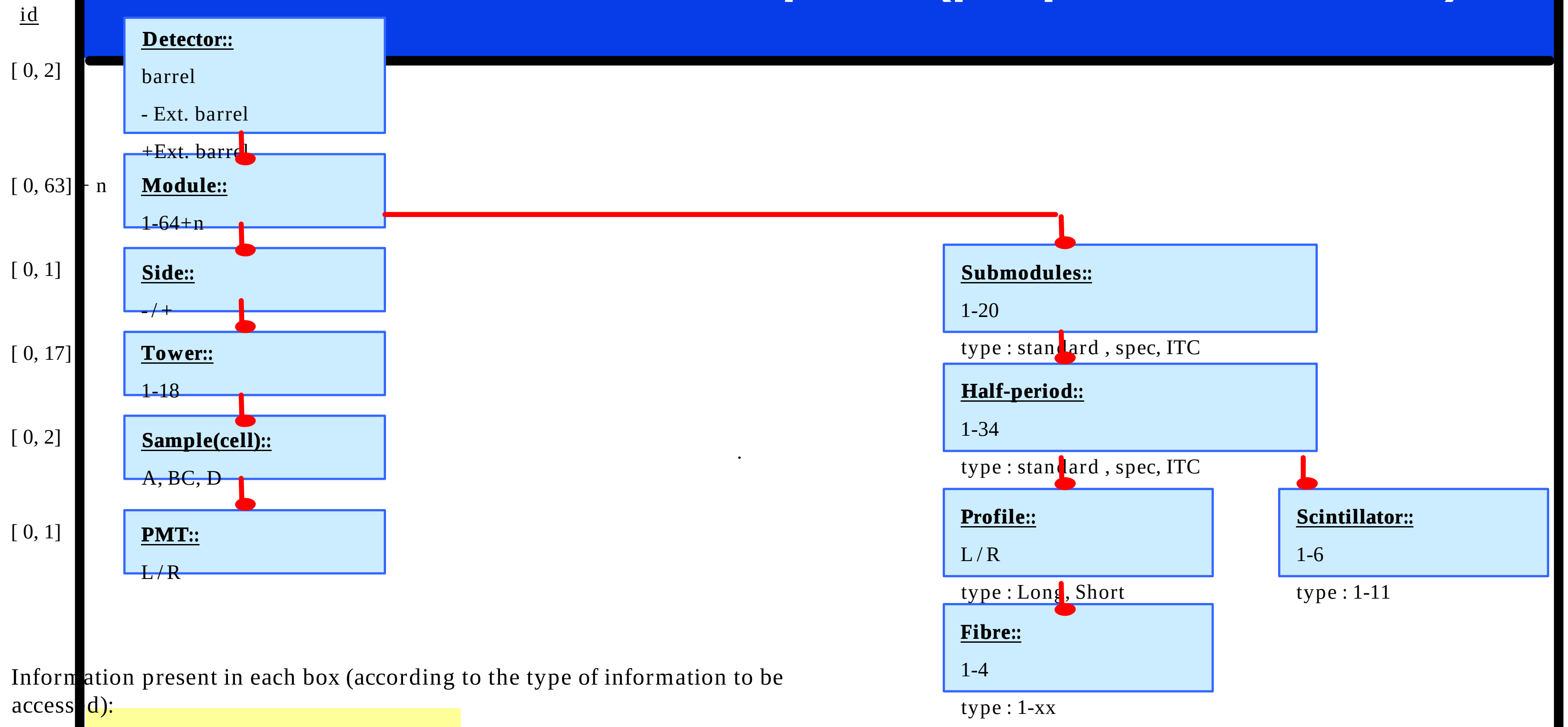
Status

- object-oriented implementation of a logical model of the tile calorimeter
- detector-centric data access architecture
- access to 1998 and 1999 raw testbeam data from object database
- support for hot-swappable custom calibration strategies
- support for 1998 default calibration for main module, and for provisional 1999 default calibration
- reconstructed runs and event collections written to object database
- for non-C++ folks (PAW/KUIP users), support (with examples) creation of HBOOK ntuples

Tilecal test beam analysis (M. Nessi)



Tilecal detector description (proposal, M. Nessi)



Information present in each box (according to the type of information to be accessed):

Position *(R,Z)*

Size

Geometry

Energy

Timing

Trigger

Calibration (data, type)

Raw data

QC information

Design Notions

- Detector-centric view
  - The detector is primary. Events are stimuli to which we want to understand the detector's response.
  - Contrast with event-centric: The event is primary. The role of the detector is to help describe and understand the event.
  - These are duals of one another. The difference in emphasis has implications.
- Keep event as opaque as possible
  - Physicists navigate through the detector, not through the event
  - less chance of significant conflict with ongoing ATLAS event definition efforts
- [Tile]Detector, Module, Side, Tower, Cell, PMT, and ADC are TileElements.
- TileCalorimeter is a TileElement factory.

Design Notions

- TileElements are created only on demand.
  - Instantiating the calorimeter does not mean instantiating every child element (detector, module, side, tower, cell, PMT, and ADC).
  - This has performance implications (positive and negative).
- TileElements are identified using ATLAS Identifier model.
  - Hierarchical naming/numbering
- TileElement states are implicitly updated when a new event is associated with the TileCalorimeter.

Transient/Persistent Separation

- Physicists see only transient model of calorimeter, event, and other data.
- Consonant with ATLAS Computing Review recommendations
- LITMUS TEST: user code must compile without Objectivity or other datastore header files
  - no HepRefs, d_Refs, ooStatus in user code
  - a TransientEvent cannot even hold a d_Ref<PersistentEvent> as a data member (though use of Objectivity directly by an implementation of TransientEvent is not strictly precluded).
- CONSTRAINT: No modification to existing persistent classes for raw data
- Even with these constraints, countless strategies are possible
- Exploring several three-layer strategies
  - middle layer can be light or heavy, smart or dumb

Middle Layer Examples

Some Objectivity mapping examples

- Simple forwarding: wrap a d_Ref
- Intermediate forwarding: wrap a few d_Refs
  - adapter may decide whether datum comes from raw or reconstructed event, or know that charge injection calibration constants and cesium constants are stored separately
- Shared adapters:
  - all TileADCs share a common TileADCAdapter object; provides certain collective optimizations

Many, many other strategies are possible.

STATUS: Here is what is in the Examples directory today.

- ShowEnergiesAndAFewADCSamples
  - illustrates how to navigate through the logical calorimeter model in C++ to get energies and access to raw data. Energies are read from the database if the run you name has been reconstructed; otherwise, they're computed from the raw data using a default calibration strategy.
- CompareCalibrations
  - illustrates how to supply your own calibration strategy, so that YOUR favorite methods are invoked by the system software to compute energies and timings, and how to alternate easily among multiple calibration strategies during program execution

STATUS: Here is what is in the Examples directory today.

- ReconstructRun
  - shows how to apply your favorite calibration (or the default calibration) to every calorimeter PMT for all the events in a run, and how to save those results in the database
- ReconstructPartialRun
  - shows how to accomplish the same task for a selected subset of events
- CreateSimpleNtuple (contributed by Tom LeCompte)
  - shows how to fill a simple PAW ntuple from the logical calorimeter model

The basic event loop (detector-centric view)

```
 TileEventIterator iter;
  for (iter = myRun->begin(); iter!= myRun->end(); ++iter)
  {
  myCal.associate_event(*iter);
  cout<<"PMT "<<(pmt1->id())<<" energy (default calibration) is ";
  cout<<pmt1->energy()<<" , timing is "<<pmt1->timing()<<endl;
  cout<<"Cell "<<(cell1->id())<<" energy is "<<cell1->energy()
    <<endl;
// or adc2->cis_calib() or adc2->samples(…) or ...
}
```

The basic event loop (comparing calibrations)

```
 TileEventIterator iter;
  for (iter = myRun->begin(); iter!= myRun->end(); ++iter) {
  myCal.associate_event(*iter);
myCal.associate_calib_strategy(&strategy1);
  cout<<"PMT "<<(pmt1->id())<<" energy (default calibration) is ";
  cout<<pmt1->energy()<<" , timing is "<<pmt1->timing()<<endl;
  cout<<"Cell "<<(cell1->id())<<" energy is "<<cell1->energy()
    <<endl;
myCal.associate_calib_strategy(&strategy2);
  cout<<"PMT "<<(pmt1->id())<<" energy (custom calibration) is ";
  cout<<pmt1->energy()<<" , timing is "<<pmt1->timing()<<endl;
  cout<<"Cell "<<(cell1->id())<<" energy is "<<cell1->energy()
    <<endl;
}
```

What’s Next?

- Use it as the testbed for core technologies (especially database) that it was intended to be
  - This was the PURPOSE, from the point of view of core software.
- Fill in the pieces needed to make it useful for tile calorimeter data analysis
  - It handles Module 0 data and calibration well--that was the July goal--but what about beam data and high voltage info and the muon walls and …?
  - Pass the baton to tile calorimeter personnel.

These are related. It does not remain an interesting testbed for long if it does not have clients trying to do real work.

- Make it part of the ATLAS offline software suite.

This will also force us to address database infrastructure issues.

Generalizations?

- Geant4 tile calorimeter simulations as alternative data sources and generalization to other calorimeter testbeams have been proposed
- Both of these require revisiting the raw data model (which for this summer was fixed and inherited from 1998 tilecal work)
- The raw model needs to be revisited in any case:
  - Some testbed work requires it (e.g., evaluation of certain components of HepODBMS)
  - Makes technical sense given what was learned in implementing the new software