ECAL analyis User Experience

Cristina Cârloganu

Clermont Ferrand

How did we start in Clermont ...

We started end of 2006 to work on testbeam data analysis

Since

- no grid experience
- new in the collaboration
- rec data in the familiar LCIO format
- marlin environment available locally
- the reconstruction code seemed quite complicated
  mostly our fault - lack of time or experience
  partly lack of documentation

work locally on reconstructed data

... seems we are quite representative !

According to answers to Paul's questionnaire:

- ✦ only ONE analyser out of EIGHT uses the grid for job submission ... and with mitigated results.

- ✦ people work mainly on reconstructed files and in some cases eventually plan to move towards raw files

What 's needed in order to use the data

What 's needed in order to use the data

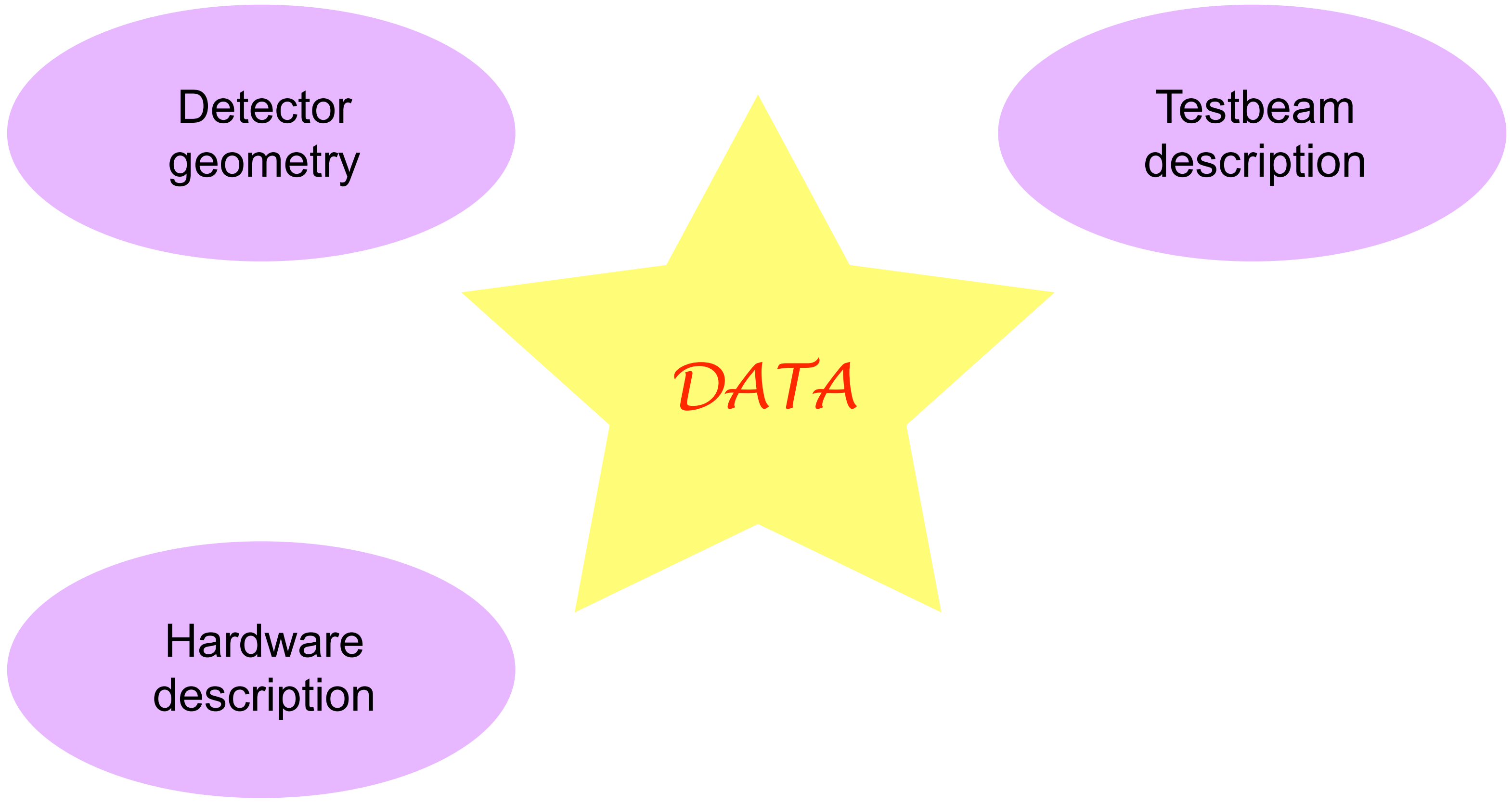
Detector geometry

DATA

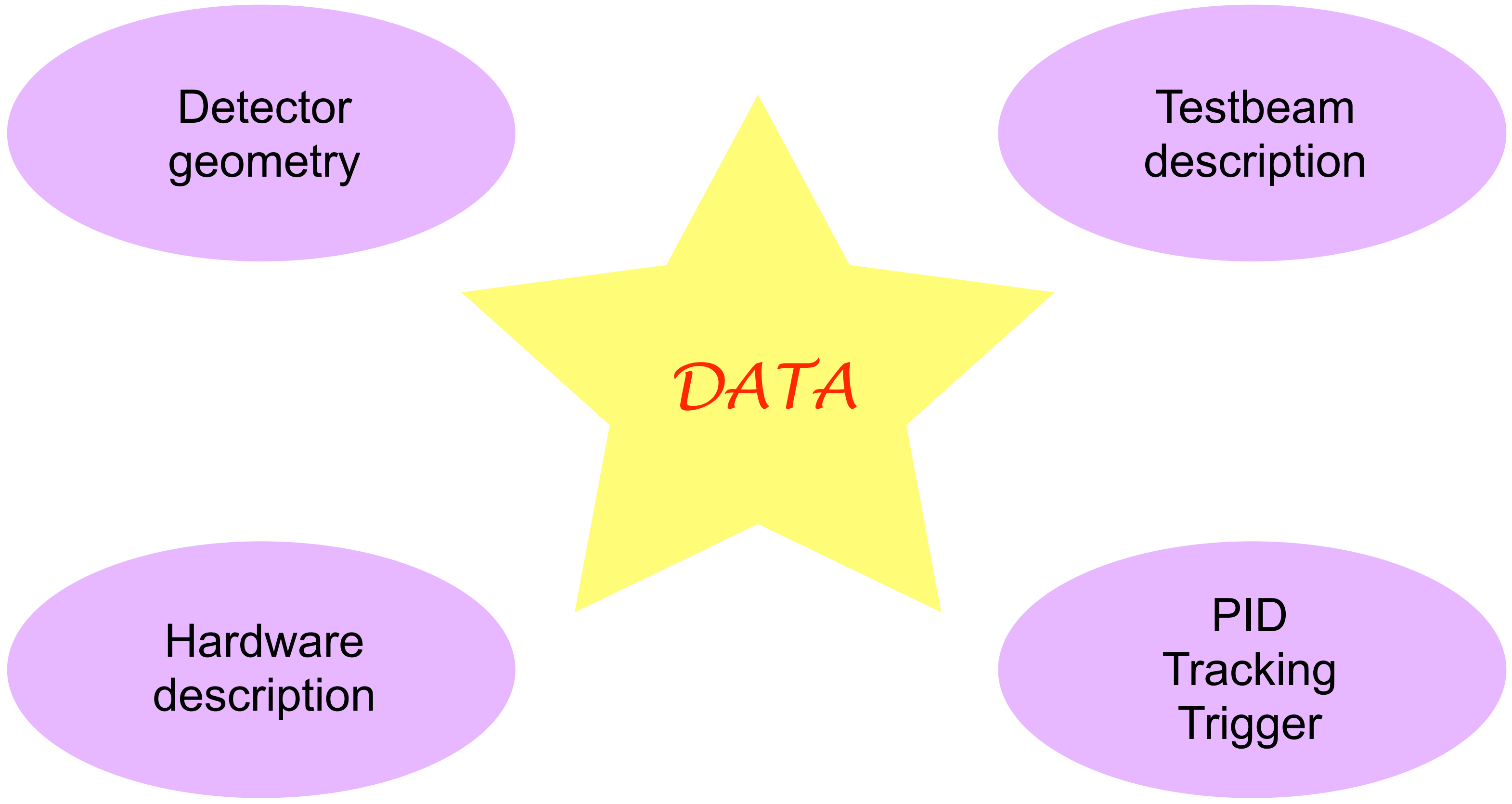
What 's needed in order to use the data

Detector geometry
Testbeam description
DATA

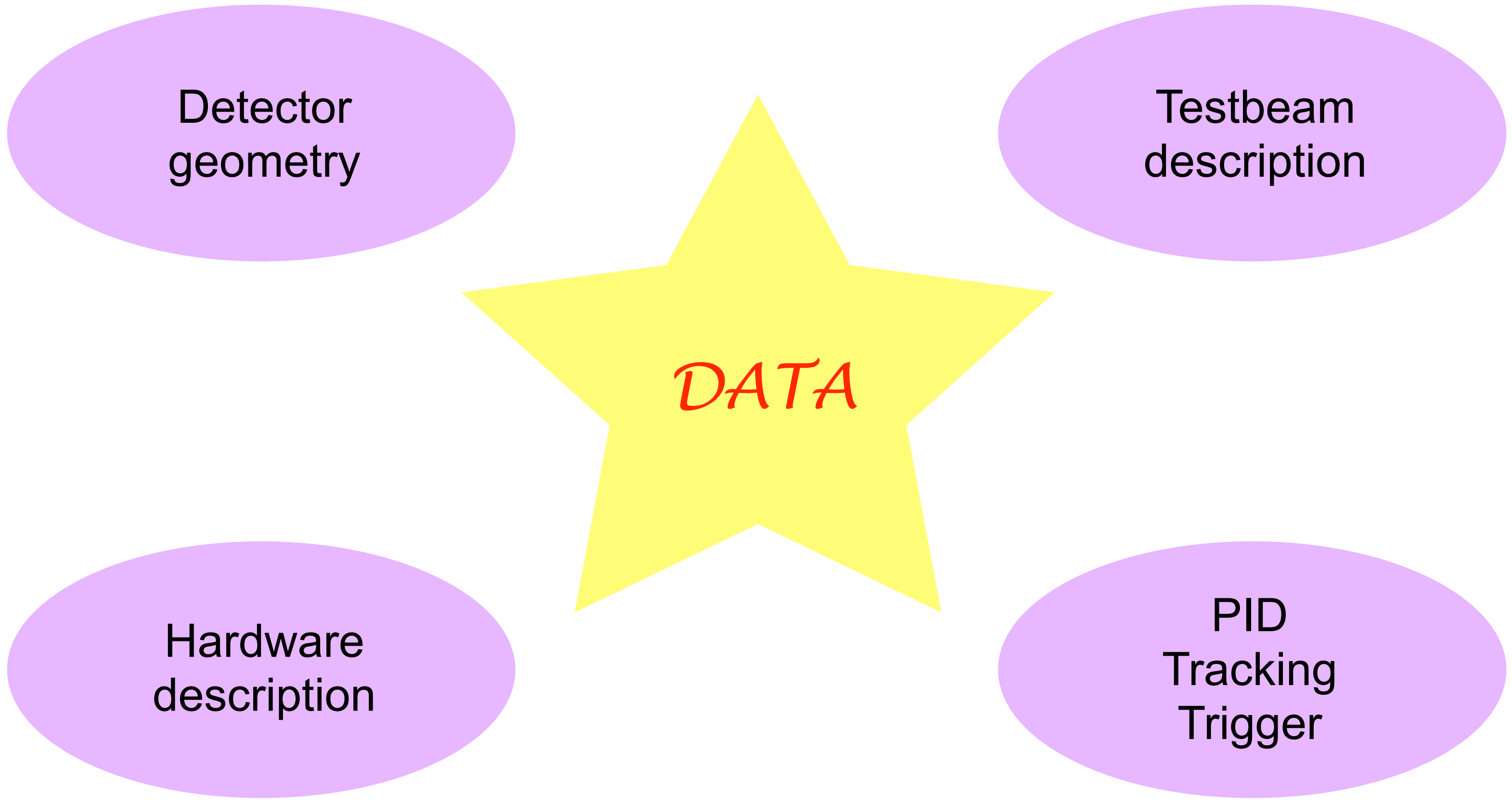
What 's needed in order to use the data



What 's needed in order to use the data



What 's needed in order to use the data



two analysis working
&
some interesting results in a relatively short time

WORKING ENVIRONMENT

Data structure and access (ECAL)

LCIO $\longrightarrow$ CalorimeterHit (*x, energy)

fine for MC, for data I eventually want to link it to the detection cell.

Our solution: a new object ...

CaloHit : public HepLorentzVector : public CaloCell

CaloCell :: public CALICE::CellIndex

Data structure and access (ECAL)

LCIO $\longrightarrow$ CalorimeterHit (*x, energy)

fine for MC, for data I eventually want to link it to the detection cell.

Our solution: a new object ...

CaloHit : public HepLorentzVector : public CaloCell

CaloCell :: public CALICE::CellIndex

calice_userlib

Data structure and access (ECAL)

LCIO $\longrightarrow$ CalorimeterHit (*x, energy)

fine for MC, for data I eventually want to link it to the detection cell.

Our solution: a new object ...

CaloHit : public HepLorentzVector : public CaloCell

CaloCell :: public CALICE::CellIndex

```
CaloCell ( protected:
               unsigned int _chip;
               unsigned int _rocable;
               double       _calibct;
               bool         _noisy;
               bool         _dead;
             )
```

calice_userlib

Data structure and access (ECAL)

Does everybody want to know about about the hits?

... not really

Data structure and access (ECAL)

Does everybody want to know about about the hits?

... not really

CaloEvent

```
CaloEvent   protected:
                        std::vector< std::vector<CaloHit> > _hits;
                        double                               _en_total;
                        std::vector<double>                  _en_layers;
                        unsigned int                         _layer_maxen;
                        unsigned int                         _nhits;

                        double                               _xb;
                        double                               _yb;
                        double                               _zb;
```

Data structure and access (ECAL)

```
void MyTBProcessor::processRunHeader( LCRunHeader* run) {
        RunInformation RunInfo = RunInformation(run);
        _beam_energy = RunInfo.beamEnergyMeV()*0.001;
        _date        = RunInfo.runMonth();
        std::cout << " Run info " << std::endl;
        std::cout << "          beam energy :    " << _beam_energy << std::endl;
        std::cout << "          taken at " << RunInfo.location() << " on " << _date << std::endl;
        _nRun++ ;
}

void MyTBProcessor::processEvent( LCEvent * evt ) {
  LCCollection*  ecalHitsCol= 0 ;
  try{ ecalHitsCol = evt->getCollection( _ecalMCHitsName ); }
  catch(DataNotAvailableException &e){}

   _event = CaloEvent(ecalHitsCol,  _hit_en_thresh,  _hit_calib);
    if(_verbose) std::cout << "CaloEvent filled with original hits" << std::endl;

  std::vector< std::vector<CaloHit> > Hits = _event.hits();
  for(unsigned int ilayer=0; ilayer<30; ++ilayer){
      for(unsigned int ih=0; ih<Hits[ilayer].size(); ++ih) {
        std::cout << Hits[ilayer][ih].getWaferRow() <<" "<< Hits[ilayer][ih].getWaferColumn() << " "
                     << Hits[ilayer][ih].getPadRow() <<" "<< Hits[ilayer][ih].getPadColumn()<< std::endl;
      }
    }
}
```

Data structure and access (ECAL)

```
void MyTBProcessor::processRunHeader( LCRunHeader* run) {
        RunInformation RunInfo = RunInformation(run);
        _beam_energy = RunInfo.beamEnergyMeV()*0.001;
        _date        = RunInfo.runMonth();
        std::cout << " Run info " << std::endl;
        std::cout << "          beam energy :    " << _beam_energy << std::endl;
        std::cout << "          taken at " << RunInfo.location() << " on " << _date << std::endl;
        _nRun++ ;
}

void MyTBProcessor::processEvent( LCEvent * evt ) {
  LCCollection*  ecalHitsCol= 0 ;
  try{ ecalHitsCol = evt->getCollection( _ecalMCHitsName ); }
  catch(DataNotAvailableException &e){}

   _event = CaloEvent(ecalHitsCol, _hit_en_thresh, _hit_calib);
    if(_verbose) std::cout << "CaloEvent filled with original hits" << std::endl;

  std::vector< std::vector<CaloHit> > Hits = _event.hits();
  for(unsigned int ilayer=0; ilayer<30; ++ilayer){
     for(unsigned int ih=0; ih<Hits[ilayer].size(); ++ih) {
       std::cout << Hits[ilayer][ih].getWaferRow() <<" "<< Hits[ilayer][ih].getWaferColumn() << " "
                     << Hits[ilayer][ih].getPadRow() <<" "<< Hits[ilayer][ih].getPadColumn()<< std::endl;
     }
    }
}
```

Data structure and access (ECAL)

```
void MyTBProcessor::processRunHeader( LCRunHeader* run) {
        RunInformation RunInfo = RunInformation(run);
        _beam_energy = RunInfo.beamEnergyMeV()*0.001;
        _date        = RunInfo.runMonth();
        std::cout << " Run info " << std::endl;
        std::cout << "         beam energy :    " << _beam_energy << std::endl;
        std::cout << "         taken at " << RunInfo.location() << " on " << _date << std::endl;
        _nRun++ ;
}

void MyTBProcessor::processEvent( LCEvent * evt ) {
  LCCollection*  ecalHitsCol= 0 ;
  try{ ecalHitsCol = evt->getCollection( _ecalMCHitsName ); }
  catch(DataNotAvailableException &e){}

   _event = CaloEvent(ecalHitsCol, _hit_en_thresh, _hit_calib);
    if(_verbose) std::cout << "CaloEvent filled with original hits" << std::endl;

  std::vector< std::vector<CaloHit> > Hits = _event.hits();
  for(unsigned int ilayer=0; ilayer<30; ++ilayer){
     for(unsigned int ih=0; ih<Hits[ilayer].size(); ++ih) {
       std::cout << Hits[ilayer][ih].getWaferRow() <<" "<< Hits[ilayer][ih].getWaferColumn() << " "
                     << Hits[ilayer][ih].getPadRow() <<" "<< Hits[ilayer][ih].getPadColumn()<< std::endl;
     }
   }
}
```

Data structure and access (tracking)

What about the other subsystems?

- [✓] Francois needed one day to get the tracking info
- [ ] it's necessary to extract two classes describing the tracks from a full package performing the tracking

Please, try to separate in the software structure
ObjectDescription - Algorithms (processors) - Tools

Data structure and access (tracking)

What about the other subsystems?

- ☑ Francois needed one day to get the tracking info
- ☐ it's necessary to extract two classes describing the tracks from a full package performing the tracking

Please, try to separate in the software structure
ObjectDescription - Algorithms (processors) - Tools

CaliceEvent
- ✦ CaloEvent
  - *CaloHit*
  - *CaloEvent*
  - *CaloCluster*
- ✦ TrackEvent
- ✦ TriggerInfo
- ✦ CerenkovInfo

Data structure and access (tracking)

What about the other subsystems?

☑ Francois needed one day to get the tracking info
☐ it's necessary to extract two classes describing the tracks from a full package performing the tracking

Please, try to separate in the software structure
ObjectDescription - Algorithms (processors) - Tools

CaliceEvent
- ✦ CaloEvent
  - *CaloHit*
  - *CaloEvent*
  - *CaloCluster*
- ✦ TrackEvent
- ✦ TriggerInfo
- ✦ CerenkovInfo

Processors
- ✦ Calibration
- ✦ HitSearch
- ✦ Clusterisation
- ✦ Tracking

Data structure and access (tracking)

What about the other subsystems?

☑ Francois needed one day to get the tracking info
☐ it's necessary to extract two classes describing the tracks from a full package performing the tracking

Please, try to separate in the software structure
ObjectDescription - Algorithms (processors) - Tools

CaliceEvent
- ✦ CaloEvent
  - *CaloHit*
  - *CaloEvent*
  - *CaloCluster*
- ✦ TrackEvent
- ✦ TriggerInfo
- ✦ CerenkovInfo

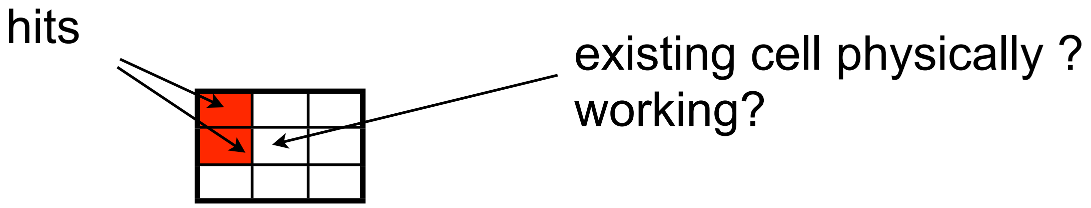
Processors
- ✦ Calibration
- ✦ HitSearch
- ✦ Clusterisation
- ✦ Tracking

Tools
- ✦ Track extrapolation
- ✦ Neighbour search

Detector geometry description

There is/was not a lot of information around. Our most efficient way of getting it : phoning/ mailing experts. What about a common web page centralising it and where everybody can contribute?

All bare geometry information lost in the reconstructed files



The information can be re-constructed ... *I know there are 2x3 wafers and I kow the row and column of each pad/wafer* ... but should each user do it?

What about a geometry package that reads the Database, defines Mokka geometry model and provides some basic user tools (eg: which are the neighboring cells for a given cell?) ?

TestBeam description

Originally, for 2006 some scattered info about the runs (mails, some presentations).

Now an extensive list of characteristics for lots of the runs, but again scattered information . What about a web page accesible to all of us (write mode)?

Excellent developement:

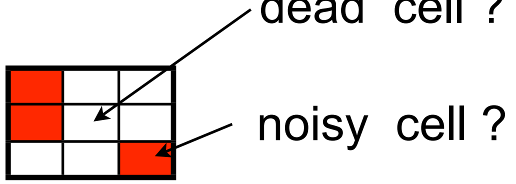
Run info available in the reconstructed files

still some lacking info: particles, incident angle

Pending problem: beam knowledge (coming later)

Hardware description

Do we need to access the database for all the hardware configurations?



What about some cell information in the RunHeader?

How do I know for a reconstructed file which are the set of constants used for calibration/digitisation?

the format should be the same for data and MC and I want to be able to play with them for MC

a tag in the run header less error prone than an external "user database"

What about some alignment info ? Is it already available in the database?

PID, particle selection

Some processors provided to the collaboration, which should add some info to the reco files.

David already has some proposal for the electrons.

If approved, they could be run together with the official reconstruction

Some summary on systematics

All the systematics I could think of for a general "energy" analysis can be addressed in the current software framework, since they imply varying MC simulation parameters. Just try to keep track of the parameters in the rec files ...

The only ones directly "connected" to the data :

✓ zero-suppression

- ✦ not very clear : cuts on the hit energy (min 0.5 MIP), signal to noise and even pedestal level.
- ✦ above 6 GeV 0.5MIP threshold low enough for systematics study, should be the same down to 1GeV
- ✦ for crosstalk studies (square events) necessary to re-process the data

✓ missing/noisy cells information (especially for clusterisation studies) - necessary to access the database.

Actually, the trickiest source of systematics for me is the beam description (momenta, geometry, uncertainty ...)

Conclusion

Working environment, most of the ingredients are there

Suggested software changes:
- re-organise (re-baptize) the code
- add an analysis package
- add a geometry package (unify MOKKA+Calice geometry description)

Suggested organisation changes:
- unify the documentation on a single/interactive(?) web page
- add documentation

We are actually quite pleased with the current software environment ...