RecExTB: Algorithms, Tools, Services and Data Objects important for LAr

Walter LAMPL

LAr Software Tutorial
November, 2nd 2004

What is RecExTB?

- A code-less package, it contains only
  - Top-jobOption files (= python scripts)
    - RecExTB_Combined_2004_jobOptions.py
  - proper cmt-requirements to get a Athena run time environment
- How does the top-jobOption file look like?
  - Sets global flags, run number, input file location, etc…
  - includes many other jobOption fragments for different tasks
    - Typically shared with RecExCommon
- What it does:
  - "Default" reconstruction of **all** Atlas subdetectors in the H8 beamline for real data or MC.
  - Produces either CBNT or ESD file.
- What not does:
  - Specialized tasks like calibration run analysis, quasi-online monitoring, …

Athena Data Objects

important for Monitoring and Reconstruction

Reconstruction Process

- LArDigit (Container)
  - ADC Samples, Gain, Channel Identifier
- LArRawChannel (Container)
  - Energy, Time, Quality factor, Channel Identifier
- LArCell (Container)
  - inherits from CaloCell
  - Energy, Time, Quality factor, Cell Identifier, CaloDetDescElement
- LArCluster (Container)
  - inherits from CaloCluster
  - Energy (per layer), Cluster position, Cluster size (in terms of η,ϕ)

Common code with Tile

Fixed and Free Gain data

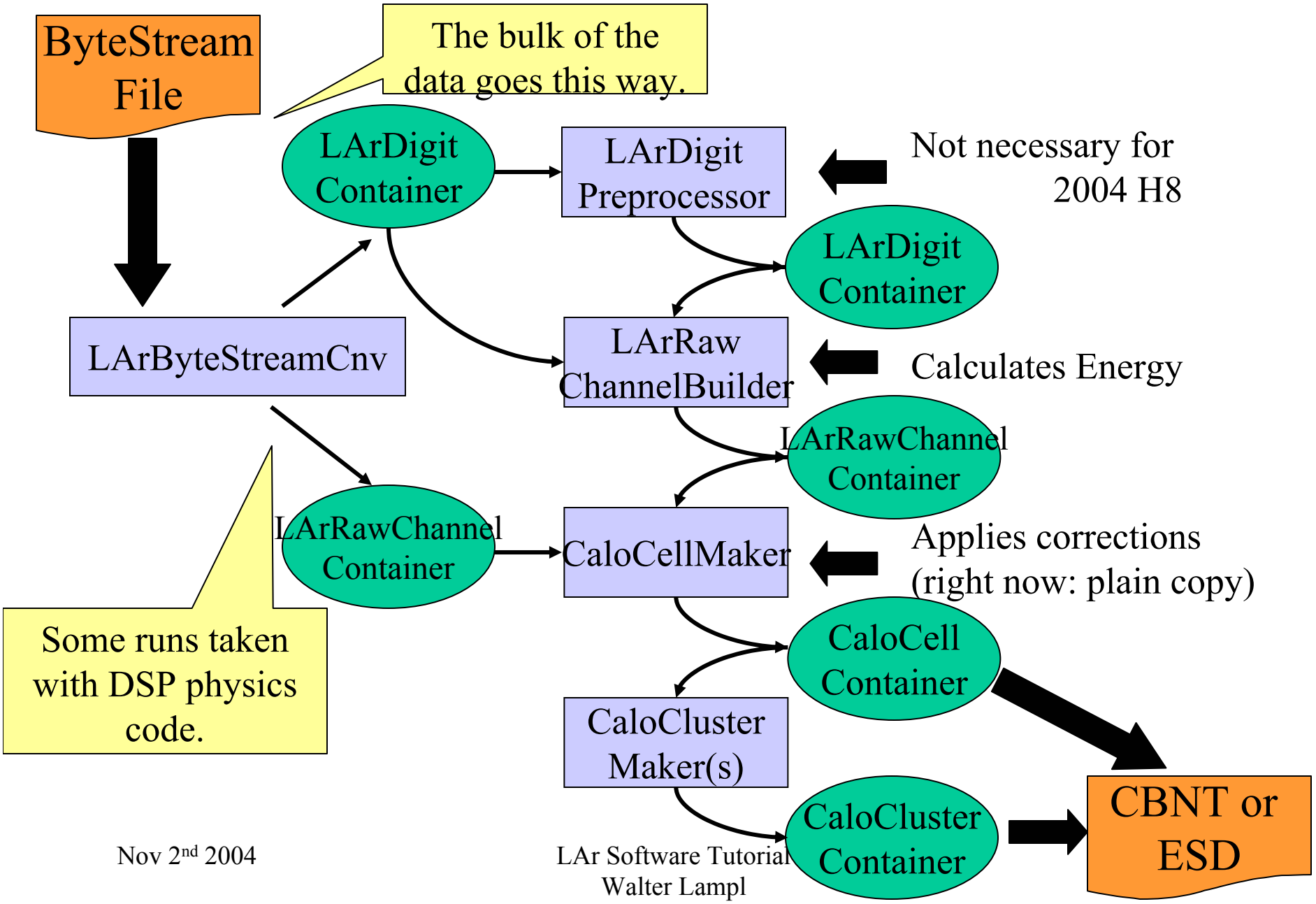
- Multiple instances of the same data object in StoreGate are distinguished by keys.
- Our convention for LArDigit and LArRawChannel:

  Key = Gain

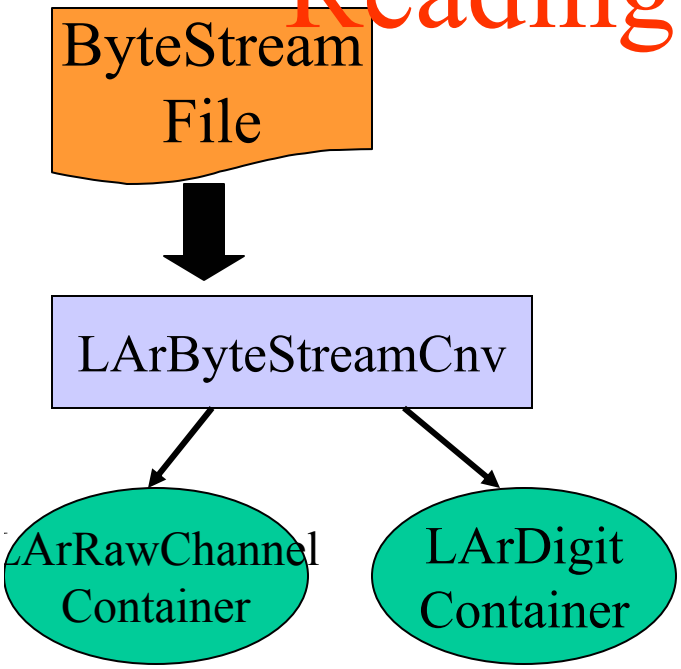
  – FREE, HIGH, MEDIUM, LOW

  – At the conversion level these keys are mandatory to tell the converter what part of the raw data to decode.
- For 2004 H8 Testbeam:
  – All physics data in free gain
  – All calibration data in fixed gain
  – Except of mistakes ….
- H6 Testbeam: Mixture of free and fixed gain (Minirods)

LAr Reconstruction Chain



Reading of ByteStream Files



- Conversion process consists of two parts
  - Generic ByteStreamCnvSvc reads the file and breaks it down into it's fragments (ROS, ROD, ….)
  - LArByteStreamCnv decodes LAr data
- Conversion on-demand: Converter is triggered by StoreGateSvc->retrieve call
- Need to tell ByteStreamAddressProviderSvc what objects (and SG key) can be loaded form BS

Job Option Syntax:

Example: LArCalorimeter/LArTestBeam/LArTBRec/share/LArTBRec_H8_Simple_jobOptions.py)

```
theApp.Dlls+=["LArByteStream"]
ByteStreamAddressProviderSvc=
  Service("ByteStreamAddressProviderSvc")
ByteStreamAddressProviderSvc.TypeNames+=
  ["LArDigitContainer/FREE"]
ToolSvc = Service("ToolSvc")
ToolSvc.LArRodDecoder.FirstSample=2
```

Load library

Data type / StoreGate Key

Add this for Free Gain (Hardware feature …)

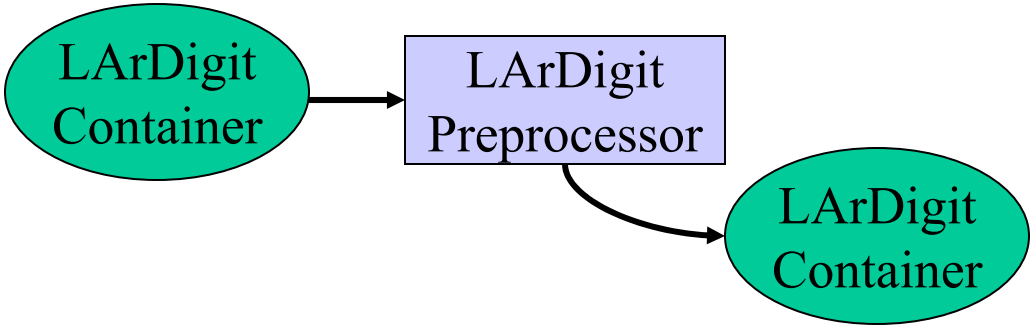
Conditions Data

- The term conditions data includes all our calibration constants like OFCs, Ramp, Pedestal…
- Persistent storage in the NOVA-IOV-Database
  - Subject to the Interval-Of-Validity service
    - You should always get the right constants
  - Retrieved via the DetectorStore (another StoreGate implementation)
  - Browseable per web http://atlobk02.cern.ch/
  - Supports tags (like CVS)
  - Different databases for testbeam, DC2 and development

Job Option Syntax:

```
LArCondCnvDbServer = 'atlobk02.cern.ch'
LArTB04IOVDbTag = "TB04-7"
include("LArCondCnv/LArCondCnv_TB04_jobOptions.py")
```

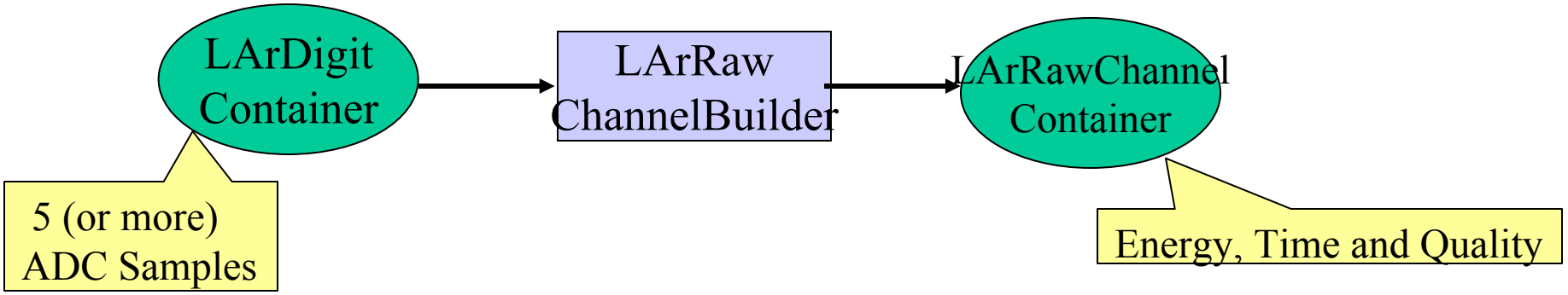
LArDigitPreprocessor



- Does some pre-processing of the raw data coming form the ByteStream
  - Merges HIGH and MEDIUM gain data to FREE gain.
  - Selects subset of samples
    - actually not necessary any more since LArRawChannelBuilder can now deal with any number of samples.
- Used for H6 and 2002 runs to merge fixed gain data
- Not necessary for 2004 H8 runs.

Purpose of the LArRawChannelBuilder

Emulates the functionality of the ROD:
Takes raw ADC samples and calculates energy time and quality.



We have currently two different versions of this algorithm.
Depending on their jobOptions we get several different 'flavors' of the LArRawChannelBuilder.

The LArRawChannelBuilder

- How it works:
  - Retrieves calibration constants from DetectorStore
    - Uses ADC2MeV Tool to get 'total ramp'
  - Selects the OFC subset and time sample subset according to time shift (phase).
  - Subtracts pedestal and applies OF-coefficients.
  - Applies ADC2MeV factors.
  - If energy > threshold
    - Calculate time (using OFC coeffs) and quality (using shape)
- How it handles failures
  - It fails if it can't get basic calibration constants like Ramp, Pedestal or OF-coefficients at all.
  - It skips single channels if these constants are missing.
  - If there is not shape for a channel, the quality factor is set to –1.
  - A error an warning summary is printed at the end of the event and end of run (depending on the output level).

Energy Reconstruction: Using OFCs

- Uses the LArRawChannelBuilder algorithm
- Right now we have only OFCs from 2002 in the Database
  - Does not give good results, mainly because of timing differences (cable length, ….)
- See LArTBRec_H8_OFC_jobOptions.py

Job Option Syntax: (the relevant part)

```
theApp.TopAlg += ["LArRawChannelBuilder"]
LArRawChannelBuilder =
  Algorithm("LArRawChannelBuilder")
LArRawChannelBuilder.DataLocation = "FREE"
LArRawChannelBuilder.UseTDC=True
LArRawChannelBuilder.NOFCTimeBins=25
```

The LArRawChannelSimpleBuilder

- Intended to be used for Monitoring without proper calibration constants.
- Much more configurable
  - Packed with if-then-else clauses to treat special cases or options.
- Special treatment for the FCAL
- Methods to reconstruct the ADC peak:
  - Maximum sample, Fixed sample, Cubic interpolation
- Methods to get the Pedestal
  - Predefined sample (usually the first one)
  - Load from database
- Methods to get the ADC2MeV factor
  - Hardcoded numbers
    - One per sampling for all LArCalorimeters, EMB Middle has two numbers
    - Gain factor hardcoded to 9.8 and 9.8*9.8
  - ADC2MeV tool and database
- Never fails (almost ….)
  - Can fall back to 'simpler' reconstruction methods or hardcode numbers if constants are missing.

Energy Reconstruction: The Simple Version

- The most robust version, does not need any DB access.
- $E = F \cdot (ADC_2 - ADC_0)$
- See: LArTBRec_H8_Simple_jobOptions.py

Job Option Syntax: (the relevant part)

```
theApp.Dlls += ["LArROD"]
theApp.TopAlg += [ "LArRawChannelSimpleBuilder" ]
LArRawChannelSimpleBuilder =
  Algorithm("LArRawChannelSimpleBuilder")
LArRawChannelSimpleBuilder.DataLocation="FREE"
LArRawChannelSimpleBuilder.LArRawChannelContainerName=
 "LArRawChannels"
LArRawChannelSimpleBuilder.PedestalSample=0
LArRawChannelSimpleBuilder.maxSamp=2
LArRawChannelSimpleBuilder.RecoMode="FIXED"
```

Default values anyway..

Energy Reconstruction: The HalfSimple Version

- Currently our "best guess" (until we get OFCs)
- Cubic interpolation to reconstruct the ADC peak, pedestal and ramps from the database.
- See LArTBRec_H8_HalfSimple_jobOptions.py

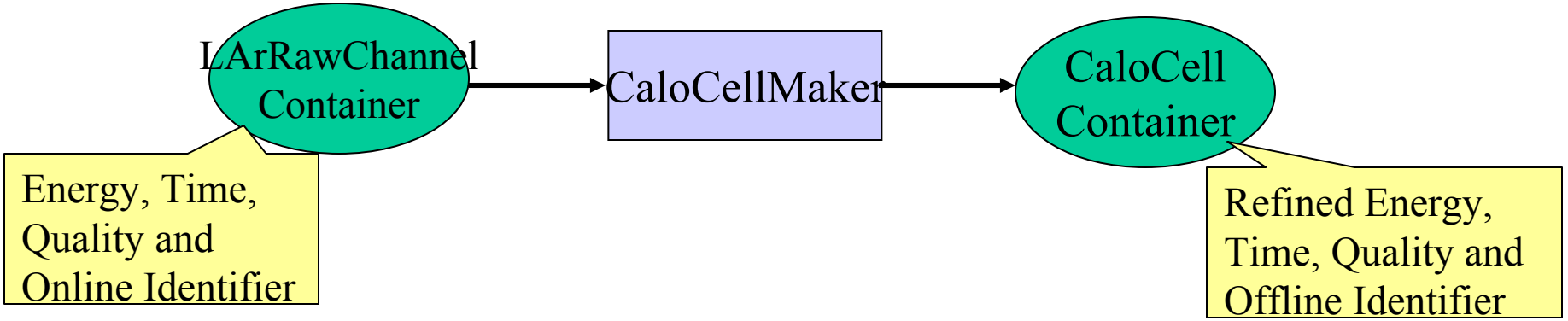
Job Option Syntax: (the relevant part)

```
theApp.TopAlg += [ "LArRawChannelSimpleBuilder" ]
LArRawChannelSimpleBuilder =
  Algorithm("LArRawChannelSimpleBuilder");
LArRawChannelSimpleBuilder.maxSamp = 2
LArRawChannelSimpleBuilder.RecoMode = "CUBIC"
LArRawChannelSimpleBuilder.CubicAdcCut = 50.
LArRawChannelSimpleBuilder.UsePedestalDB=True
LArRawChannelSimpleBuilder.UseRampDB=True
```

Some words about timing….

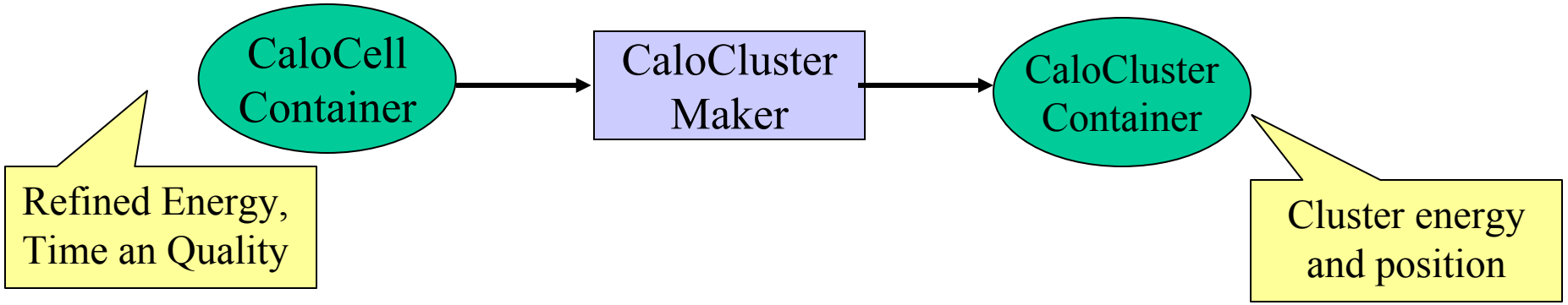
- To apply OFCs, one need to know the peak time of the signal.
  - Time between Master Trigger and 40MHz clock measured by TDC. (At least most of the time)
- One particularity of the Testbeam 2004 was that the trigger timing changed frequently.
  - Corrected by LVL1 accept latency but only in steps of 25 ns.
- To accommodate for this, we have additional conditions data objects holding time offsets
  - Global and Feb-to-Feb
  - Cell-to-cell time offset stored together with OF-Coeffs
- $t_{Total}=t_{Phase}+t_{Global}+t_{FEB}+t_{Cell}$

CaloCellMaker



- In ATLAS, this will be the first step of offline reconstruction.
- Refined calibration, apply corrections
  - In the TB case, right now it just copies LArRawChannels
- Merges data from different Calorimeters (LAr, Tile)
- Architecture: TopAlgo **CaloCellMaker** invokes several AlgTools for different tasks.
- See CaloTBRec_H8_Cell_jobOptions.py

CaloClusterMaker



- Architecture similar to CellMaker
  - TopAlgo CaloClusterMaker that invokes one or more
    - CaloClusterMakerTools
    - CaloClusterCorrectionTools
- Currently we have tree ClusterMakerTools:
  - Topological Clustering
  - Sliding Window
  - EMTB-style Clustering (3 x 3)
- By default, all three are executed in RecExTB
  - See jobOption fragment CaloTBRec_H8_Cluster_jobOptions.py

A few words about Identifiers…

- CaloCells, LArRawChannels, etc, … are uniquely identified by either **Online** or **Offline** identifiers.
- Calibration data can be retrieved using an identifier
- LArCablingSvc provides conversion between these two types of identifiers
- Online Identifiers
  - Encodes: Barrel-EC, Side, FT, Slot, Channel
  - Use LArOnlineID helper class to get individual fields
- Offline Identifier
  - Encodes (EM-case): Barrel-EC, Layer, Region, Eta, Phi
  - Use LArEM_ID, LArHEC_ID, LArFCAL_ID and Tile_ID helper classes to get individual fields.

Practical Info:
Usage of LArCablingSvc and Identifier helpers (1)

JobOptions:

```
#Get get LArDetectorDescription
include( "LArDetMgrDetDescrCnv/LArDetMgrDetDescrCnv_H8_joboptions.py" )
#Got get LArCablingSvc
theApp.Dlls += [ "LArTools"]
```

Class definition (MyAlgo.h)

```
#include "LArTools/LArCablingService.h"

//Member Variables
//Pointer to Services and Helpers
const LArOnlineID* m_onlineHelper;
const LArEM_ID* m_emId;
LArCablingService *m_larCablingSvc;
```

Usage of LArCablingSvc and Identifier Helpers (3)

Class implementation, initialization (MyAlgo.cxx)

```
const CaloIdManager *caloIdMgr=CaloIdManager::instance() ;  //Get CaloIdManager Singleton
m_emId=caloIdMgr->getEM_ID(); //Retrieve LArEM_ID from CaloIdManager
IToolSvc* toolSvc;
sc=service( "ToolSvc",toolSvc );   //Retrieve ToolSvc
if (sc.isFailure()) {
  log << MSG::ERROR << "Unable to retrieve ToolSvc" << endreq;
  return StatusCode::FAILURE;
}
sc=toolSvc->retrieveTool("LArCablingService",m_larCablingSvc); //Retrieve CablingSvc from ToolSvc
if (sc.isFailure()) {
  log << MSG::ERROR << "Unable to retrieve LArCablingService" << endreq;
  return StatusCode::FAILURE;
}
StoreGateSvc* detStore;
sc = service("DetectorStore", detStore); //Retrieve DetectorStore Service
if (sc.isFailure()) {
  log << MSG::FATAL << " Cannot locate DetectorStore " << std::endl;
  return StatusCode::FAILURE;
}
sc = detStore->retrieve(m_onlineHelper, "LArOnlineID");  //Retrieve LArOnlineHelper from DetStore
if (sc.isFailure()) {
  log << MSG::ERROR << "Could not get LArOnlineID helper !" << endreq;
  return StatusCode::FAILURE;
}
```

Usage of LArCablingSvc and Identifier Helpers (3)

Class implementation, execute (MyAlgo.cxx)

```
int eta,phi,layer; // Some fields of the offline ID
int FT, slot, channels; //Some fields of the online ID
LArDigitContainer::const_iterator it=larDigitCont->begin();
LArDigitContainer::const_iterator it_end=larDigitCont->end();
for (;it!=it_end;it++) { //Iterate over LArDigitContainer
  const HWIdentifier chid=(*it)->hardwareID(); //Get OnlineID of LArDigit
  FT = m_onlineHelper->feedthrough(chid);
  slot = m_onlineHelper->slot(chid);
  channel = m_onlineHelper->channel(chid);
  try {
    const Identifier id=m_larCablingSvc->cnvToIdentifier(chid); //Get Offline ID
    if (m_emId->is_lar_em(id)) {
       eta=m_emId->eta(id);
       phi=m_emId->phi(id);
       layer=m_emId->sampling(id);
     }
    else {
      //Not EM, do something else….
     }
    catch (LArID_Exception & except) {
      //Can't get  Offline ID, disconnected channel?
    }
}
```

DetectorDescription : what does it do for you ? (1)

- Reads the famous "primary numbers" from the same DB as simulation
  - all possible dimensions : barrel inner radius, length of the modules
  - all possible information on global positions
    - e.g. EMEC is shifted in z by 4cm
- it gets the information on channels from the Identifier packages
  - e.g how many channels there are in the strips, their width on eta and phi.
- Contains some code using all these numbers to build, for each cell, an object containing all the info needed by reconstruction :
  - Cell position in the Atlas official coordinate system (including the effect of alignment)
  - Cell position in our coordinate system
  - Cell volume (request from combined reconstruction)
  - Anything needed… is either cached or computed on the fly (time/space trade off)

DetectorDescription : what does it do for you ? (2)

- A unique manager provides access to all geometry-related numbers and can answer general questions like:
  - Given eta/phi and a radius, in which cell am I ?
  - Give me the list of cells in a box in eta/phi
  - Give a specific cell with such identifier … or, on the contrary, a fast loop on all of them

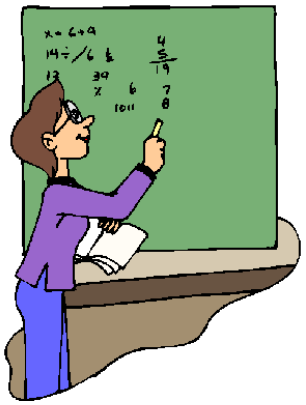
DetectorDescription : How to use it?

See webpage maintained by Claire:

http://atlas.web.cern.ch/Atlas/GROUPS/LIQARGSOFF/DetectorDescription/index.html

Other useful tools

- CaloNoiseTool
  - Provides the sigma and the quadratic sum of electronic and pile-up noise in MeV per cell.
  - Used by algorithms like TopoClusterMaker and MissingEt.
  - Uses pedestal runs for electronic noise (and DC1(2) minimum bias events) and takes the effect of digitization into account.
- LArADC2MeVTool
  - Provides the ADC-Peak to MeV factor per cell by multiplying the ADC to DAC, the DAC to μA and the μA to MeV factors.
  - Value calculated and cashed on begin of each run.
  - Used by LArRawChannelBuilder algorithm.



A lot of theory …..
… now let's try it!

Enjoy the hands-on exercises!