Details of CALICE Software Model

Roman Pöschl
LAL Orsay

Part I: Calice Dataprocessing

- Calice Testbeam Data Taking
- Data Management
- Event Building and Reconstruction Software
- Summary

Part II: Conditions Data Handling

- Conditions Data, LCCD, Database and all that
- Discussion of critical items
- Summary and Outlook

CALICE Software Review 18/12/07

Part I

Calice Dataprocessing

The Three Pillars of Calice Software

ILC Software

GRID

Database

See talk this afternoon

Objectives explicitly:

- Application of general ILC Software tools where possible and therefore benefiting from general developments of the ILC Software. At the same time the application of these tools allow for the identification of the needs of the ILC Software for real data already at an early stage of the R&D phase

- Since test beam data are taken at different locations they have to be independent of the experimental site This leads to the employment of grid tools

- High data integrity which demands the employment of database mechanism

- As many users as possible as possible are to get involved in the analysis effort therefore the entry points for an easy start-up of the analysis have to be provided

In the following I will outline how these tools are employed and work together and how the objectives are met

CALICE Testbeam Data Taking

CALICE collaboration is preparing/performing large scale testbeam

Data taking in Summer 2006/2007

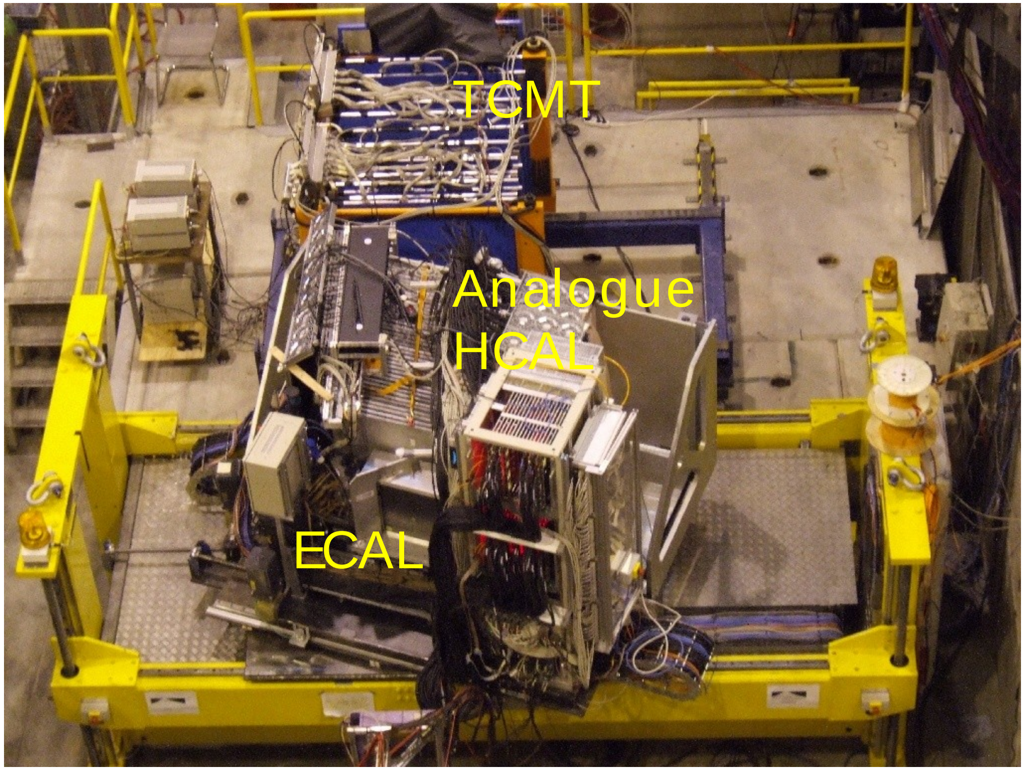
Testbeam program poses software/computing " challenges"

- Data processing from Raw Data to final Clusters in a coherent way

- Handling of Conditions Data Detector Configuration Calibration, Alignment etc.

-Comparison with simulated data 'Physics' Output

Testbeam Setup at CERN 2007



O(15000) calorimeter cells readout by Calice DAQ
No Zero Suppression

CALICE “TIER 0” – Infrastructure in the Control Room

Picture courtesy of C. Rosemann DESY

Gigabit Uplink
- High Speed Connection to the outside world
- Serves all Calice Control Room Computers

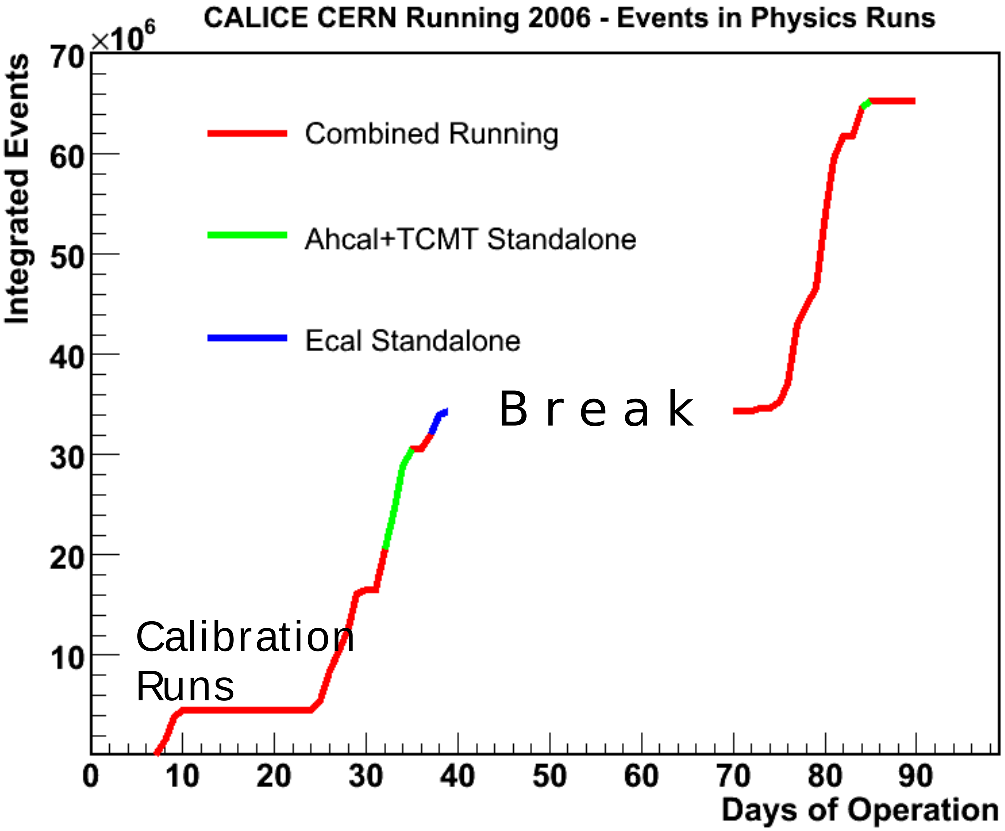
caliceserv.cern.ch
- Online Monitoring
- Grid Transfers

Disk Array

DAQ Computer

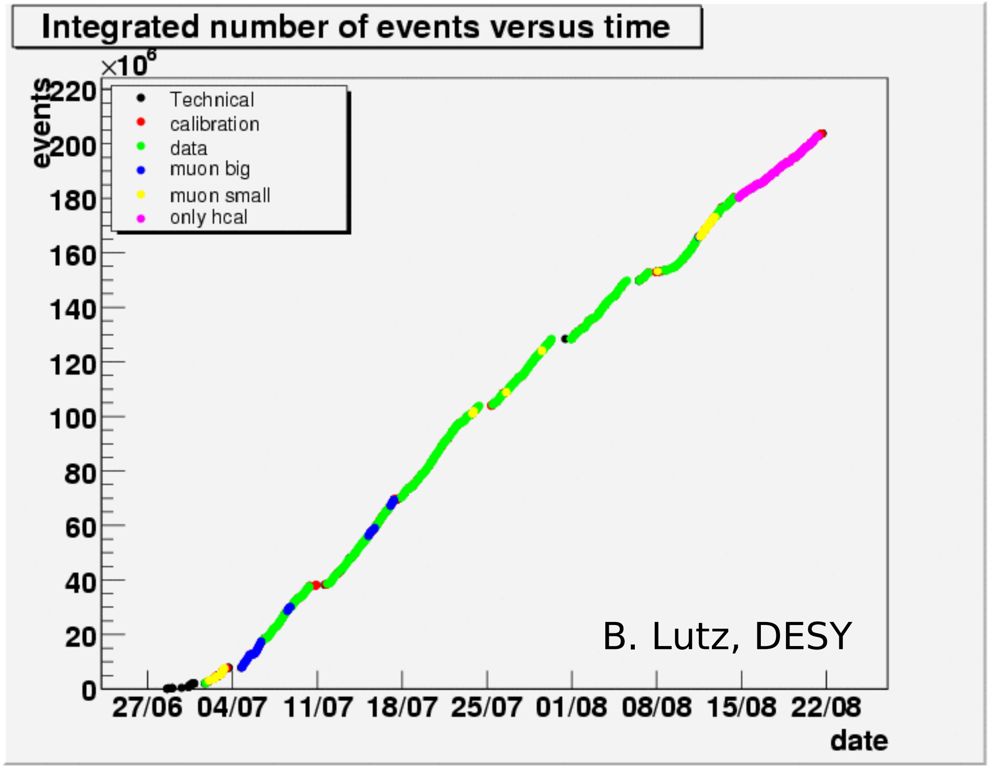
Well organized setup of computing
Thanks to B. Lutz

CALICE - CERN Data taking 2006/2007



Break

Calibration Runs



B. Lutz, DESY

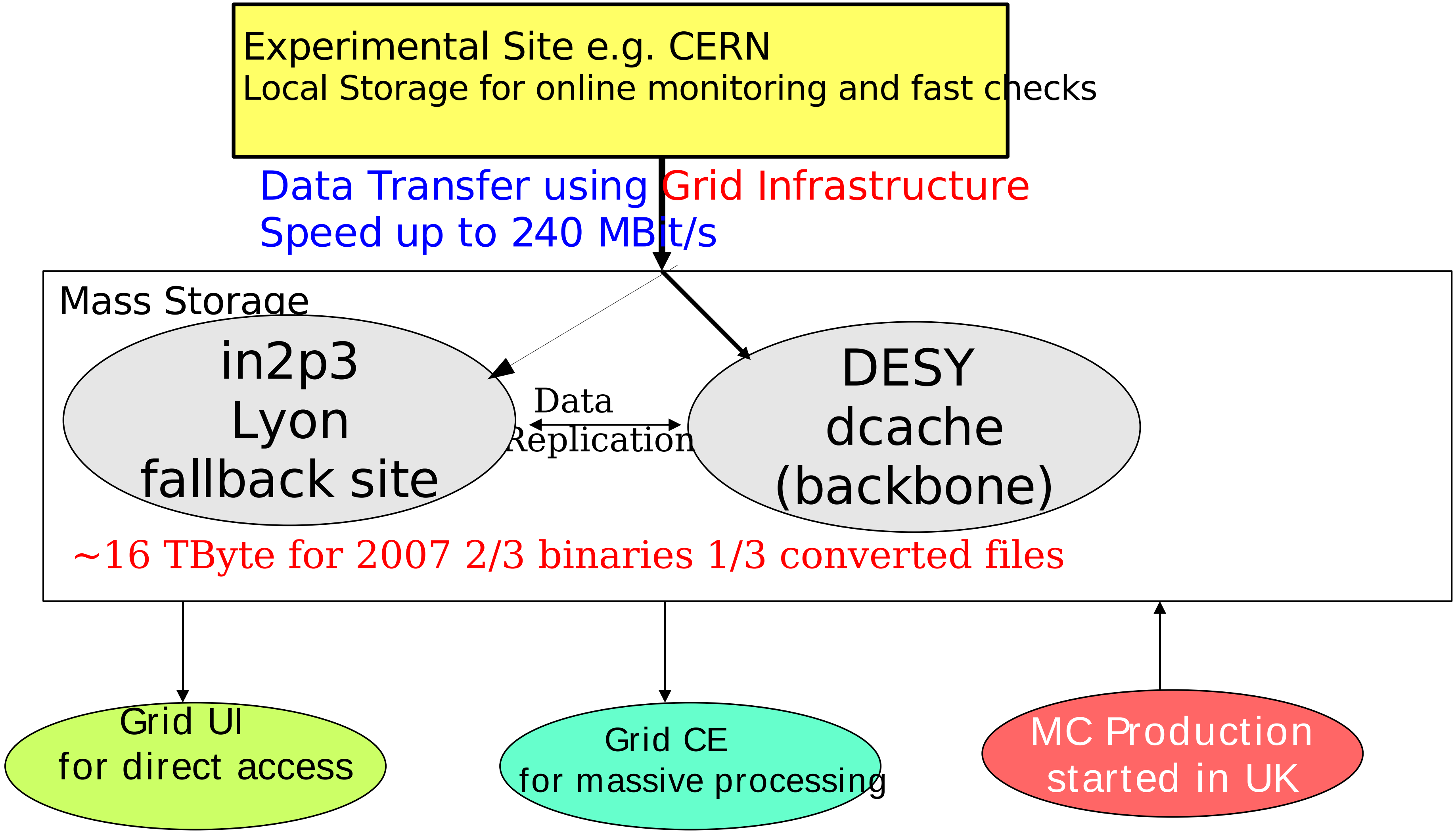
~200 Mio Events
in 'Physics' Runs

+

O(50 Mio). Muon
Calibration Events)

Efficient and fast
way of data distribution
and processing ?

Data Handling and Processing



Experimental Site e.g. CERN
Local Storage for online monitoring and fast checks

Data Transfer using Grid Infrastructure
Speed up to 240 MBit/s

Mass Storage

in2p3
Lyon
fallback site

Data
Replication

DESY
dcache
(backbone)

~16 TByte for 2007 2/3 binaries 1/3 converted files

Grid UI
for direct access

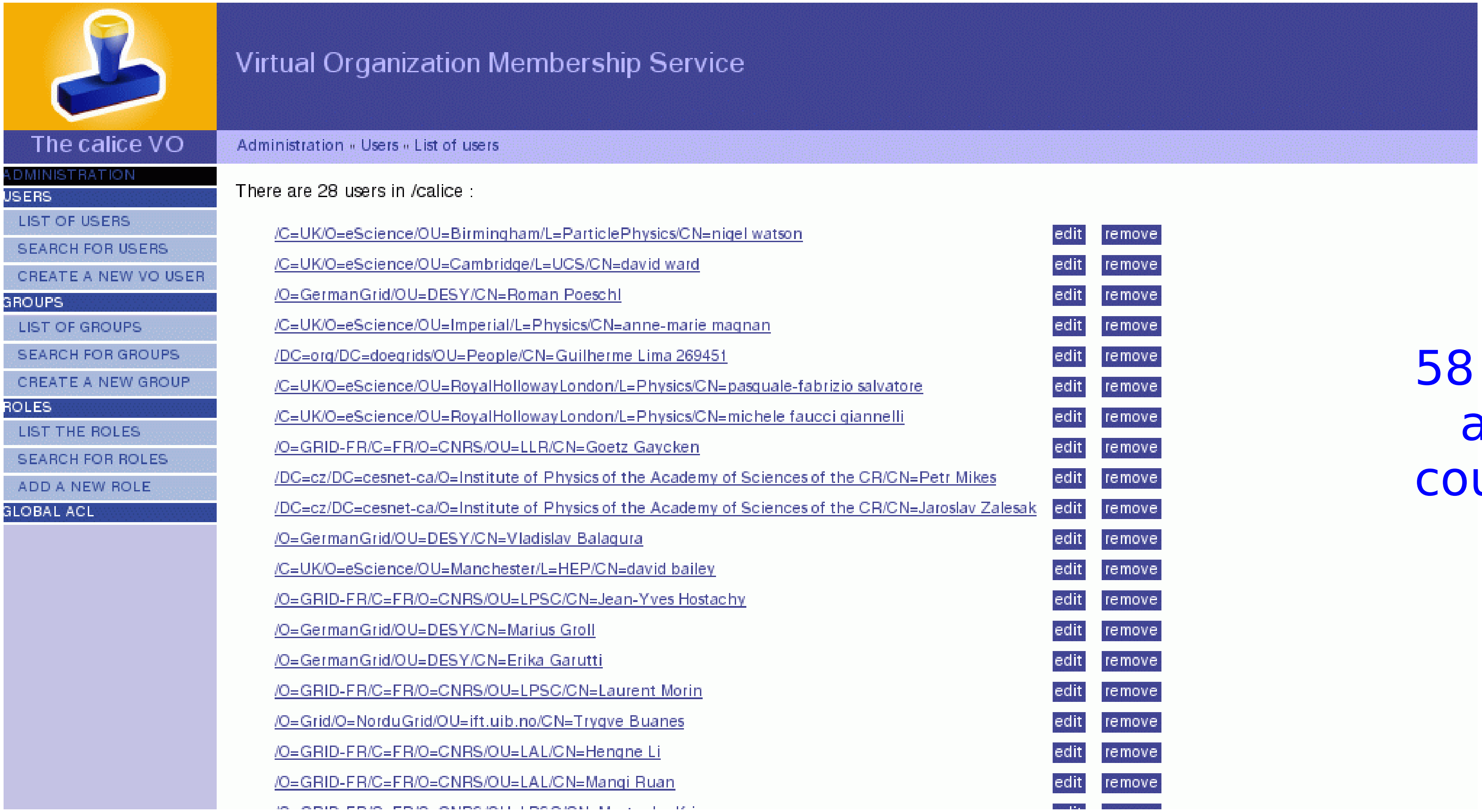
Grid CE
for massive processing

MC Production
started in UK

- Raw Data are (usually) available ~20 Min. after Run End
- Delay of Converted Files (usually) < 1 day

The Virtual Organisation - vo calice

Hosted by DESY:
Page for registration is https://grid-voms.desy.de:8443/voms/calice



There are 28 users in /calice :

/C=UK/O=eScience/OU=Birmingham/L=ParticlePhysics/CN=nigel watson	edit	remove
/C=UK/O=eScience/OU=Cambridge/L=UCS/CN=david ward	edit	remove
/O=GermanGrid/OU=DESY/CN=Roman Poeschl	edit	remove
/C=UK/O=eScience/OU=Imperial/L=Physics/CN=anne-marie magnan	edit	remove
/DC=org/DC=doegrids/OU=People/CN=Guilherme Lima 269451	edit	remove
/C=UK/O=eScience/OU=RoyalHollowayLondon/L=Physics/CN=pasquale-fabrizio salvatore	edit	remove
/C=UK/O=eScience/OU=RoyalHollowayLondon/L=Physics/CN=michele faucci giannelli	edit	remove
/O=GRID-FR/C=FR/O=CNRS/OU=LLR/CN=Goetz Gaycken	edit	remove
/DC=cz/DC=cesnet-ca/O=Institute of Physics of the Academy of Sciences of the CR/CN=Petr Mikes	edit	remove
/DC=cz/DC=cesnet-ca/O=Institute of Physics of the Academy of Sciences of the CR/CN=Jaroslav Zalesak	edit	remove
/O=GermanGrid/OU=DESY/CN=Vladislav Balagura	edit	remove
/C=UK/O=eScience/OU=Manchester/L=HEP/CN=david bailey	edit	remove
/O=GRID-FR/C=FR/O=CNRS/OU=LPSC/CN=Jean-Yves Hostachy	edit	remove
/O=GermanGrid/OU=DESY/CN=Marius Groll	edit	remove
/O=GermanGrid/OU=DESY/CN=Erika Garutti	edit	remove
/O=GRID-FR/C=FR/O=CNRS/OU=LPSC/CN=Laurent Morin	edit	remove
/O=Grid/O=NorduGrid/OU=ift.uib.no/CN=Trygve Buanes	edit	remove
/O=GRID-FR/C=FR/O=CNRS/OU=LAL/CN=Hengne Li	edit	remove
/O=GRID-FR/C=FR/O=CNRS/OU=LAL/CN=Manqi Ruan	edit	remove

58 Members
and
counting ...

VO Manager: R.P./ LAL, Deputy: A. Gellrich/ DESY

Institutes which provide Grid support for Calice

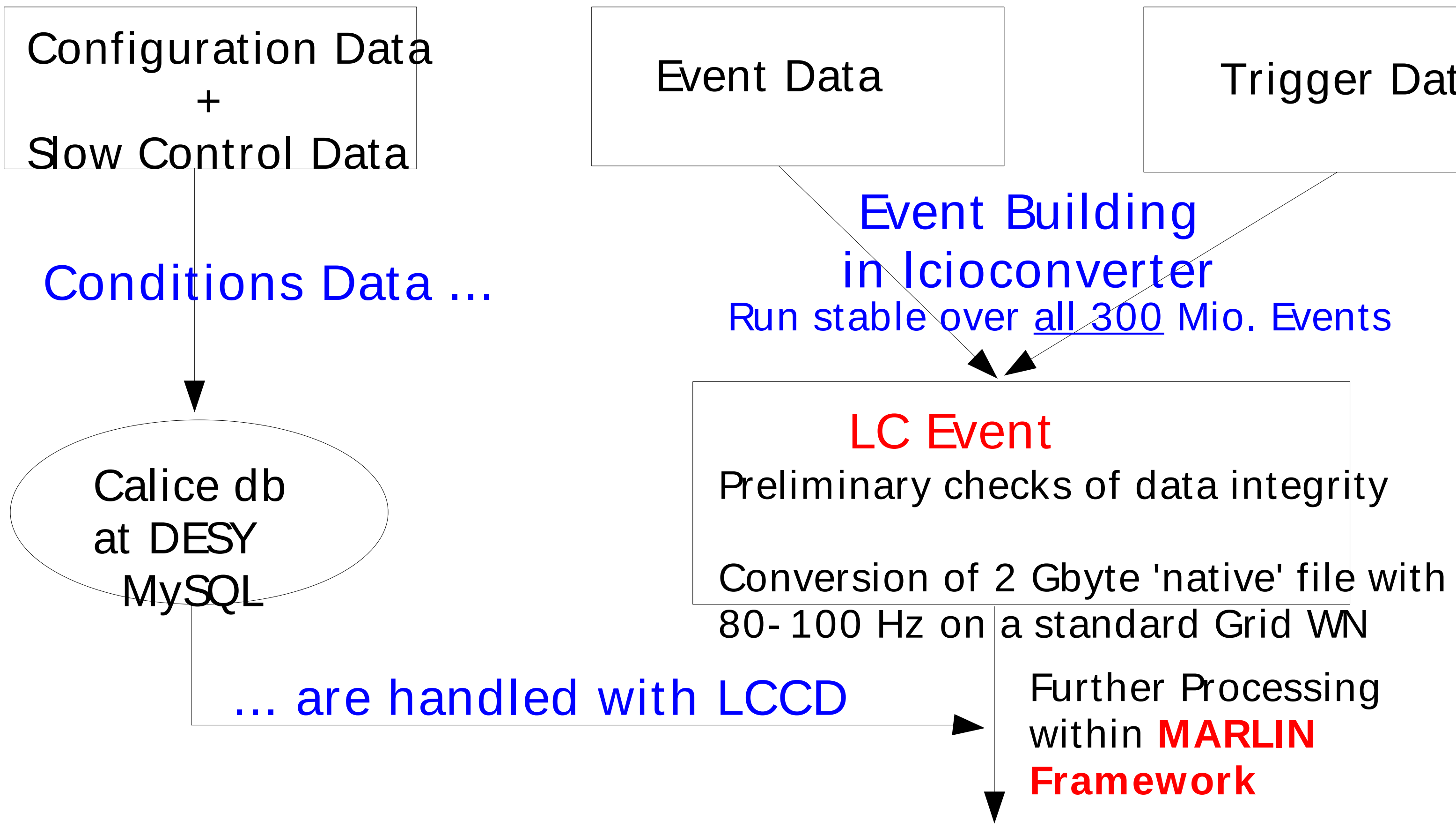
Supported by: DESY Hamburg	Hosting, Computing and Storage
LAL	Computing and Storage
LLR	Computing and Storage
DESY Zeuthen	Computing and Storage
Imperial College	Computing and Storage
Birmingham	Computing and Storage
cc in2p3 Lyon	Computing and Storage
Cambridge	Computing and Storage
Institute of Physics Prague	Computing and Storage (in preparation)
University College	Computing and Storage
KEK	Computing and Storage
Manchester	Computing and Storage
CIEMAT Madrid	Computing and Storage
Fermilab	Computing and Storage Exploit started between Fermilab and NIU Colleagues
Univ. Liverpool	Resources Provided (not yet exploited)
Univ. Regina	Offer Received

- Most of the sites have been involved in recent data and MC processing
  Smaller Problems at Manchester and KEK (about to be solved)

Conversion to LCIO

DAQ data types are converted/wrapped into LCIO on the basis of LCGenericObjects

DAQ Data Files/Types



Configuration Data
+
Slow Control Data

Event Data

Trigger Data

Conditions Data ...

Event Building
in lcioconverter
Run stable over all 300 Mio. Events

Calice db
at DESY
MySQL

LC Event
Preliminary checks of data integrity

Conversion of 2 Gbyte 'native' file with
80-100 Hz on a standard Grid WN

... are handled with LCCD

Further Processing
within **MARLIN Framework**

Remark: LCIO and ILC software framework is not needed to analyze calice data but using it delivers important input for future ILC s/w development
-> General ILC Concept for low level data handling

Intermezzo: Important Definitions

Expert:

Person who by position or charge (i.e. In a task force)
is entrusted/responsible with preparation and
running of the reconstruction jobs

More general, person who is able to run the reco job

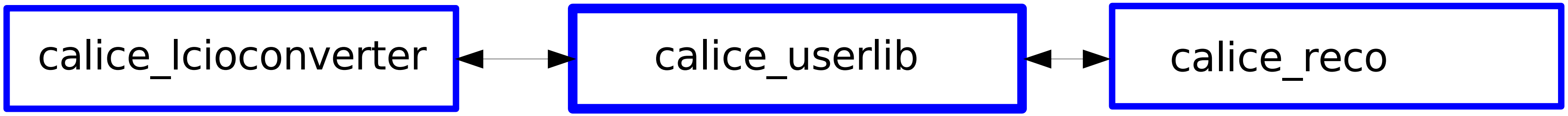
User:

Person which starts his/her analysis on the
reconstruction files

Calice Software

Three main packages

Contributions by groups from
DESY, Imperial, LAL, LLR, NIU, RHUL



calice_lcioconverter

calice_userlib

calice_reco

Current version
v04-02-06

converts
calice DAQ format
into LCIO (LCGenericObjects)

needs DAQ software
expert work

MARLIN processors

Current version
v04-10

Interface classes to
LCGenericObjects

utililty functions e.g.
For TriggerHandling

Current version
v04-06

RawData into
CalorimeterHits
(standard LCIO)
TrackerHits

First stages of
higher level analysis
MARLIN processors

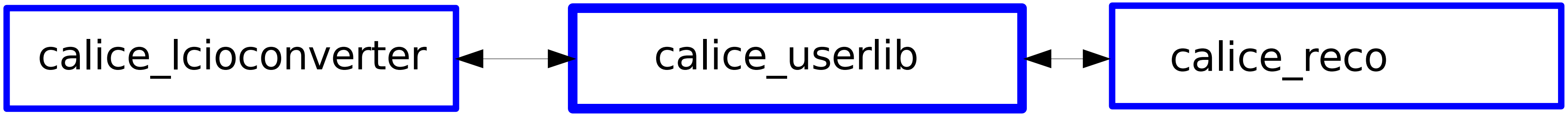
225 classes or functions

Data of four different Calorimeter Prototypes are
available in LCIO format

Calice Software

Three main packages

More details


calice_lcioconverter

calice_userlib

calice_reco

Clear Expert work
No user should link
against it

Central Library for calice

Should be free in
dependency of third
party packages such as root

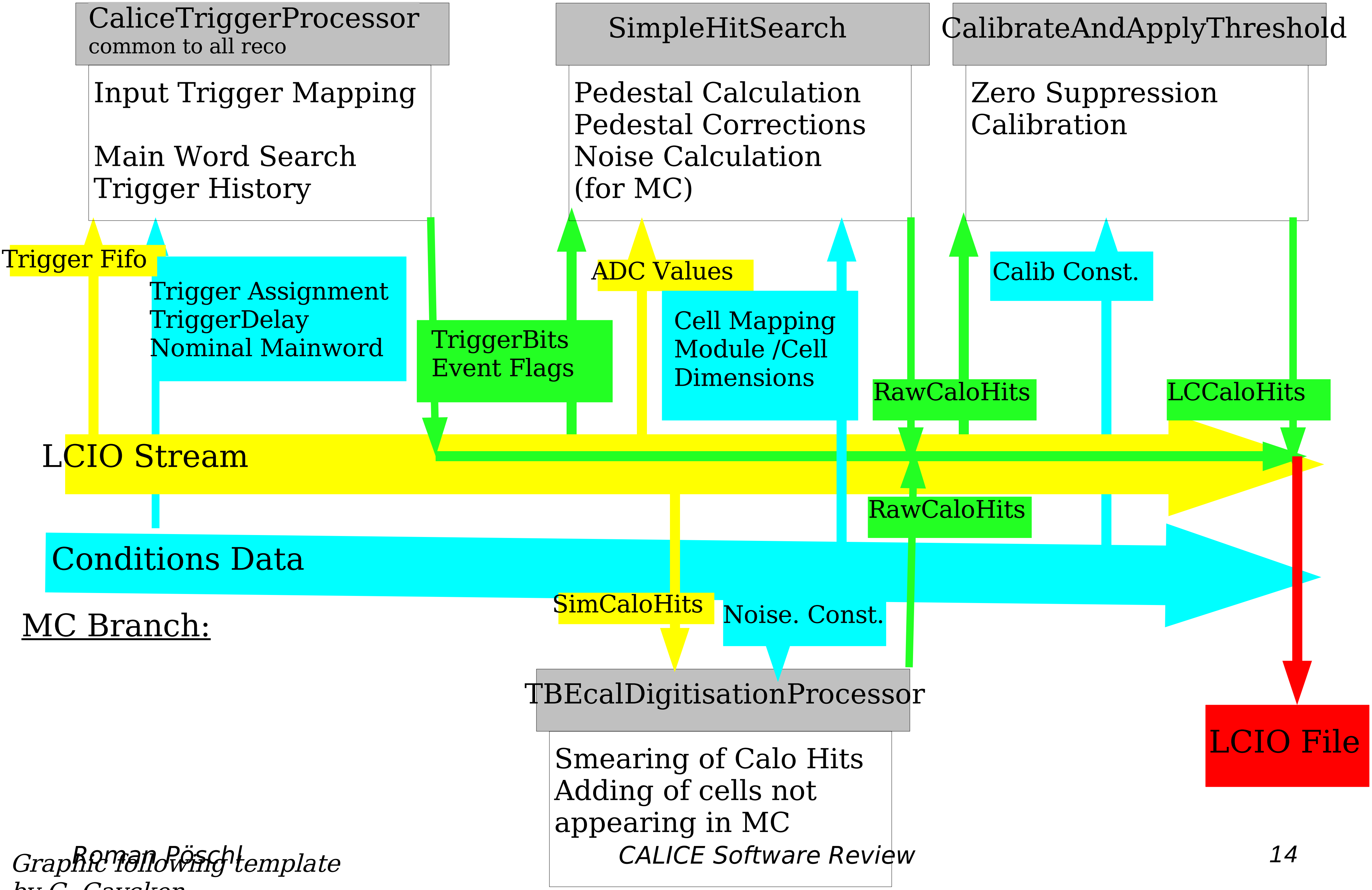
To be linked against .
user applications

Expert work
No user should link
against it

Systematic Studies
which require
a re-running of the
reconstruction are to be
performed by dedicated
task forces

These packages might be completed by a forth package
calice_analysis which should contain algorithms
needed for analysis (and may depend on third party packages)

Example for Data Processing - SiW Ecal



CaliceTriggerProcessor
common to all reco

Input Trigger Mapping

Main Word Search
Trigger History

SimpleHitSearch

Pedestal Calculation
Pedestal Corrections
Noise Calculation
(for MC)

CalibrateAndApplyThreshold

Zero Suppression
Calibration

Trigger Fifo

Trigger Assignment
TriggerDelay
Nominal Mainword

TriggerBits
Event Flags

ADC Values

Cell Mapping
Module /Cell
Dimensions

RawCaloHits

Calib Const.

LCCaloHits

LCIO Stream

RawCaloHits

Conditions Data

MC Branch:

SimCaloHits

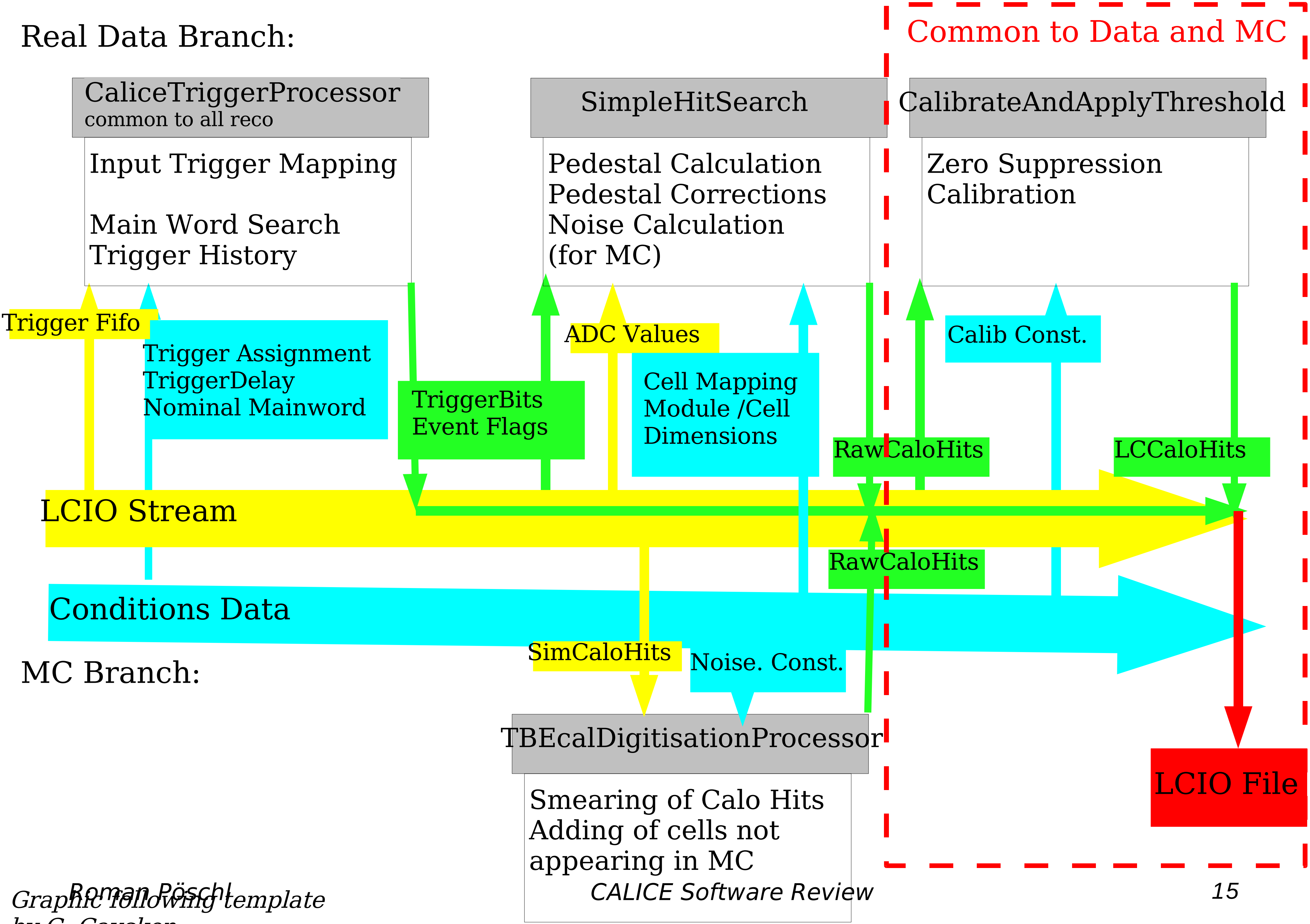
Noise. Const.

TBEcalDigitisationProcessor

Smearing of Calo Hits
Adding of cells not
appearing in MC

LCIO File

Graphic following template
by G. Gaycken



Real Data Branch:

Common to Data and MC

CaliceTriggerProcessor
common to all reco

Input Trigger Mapping

Main Word Search
Trigger History

SimpleHitSearch

Pedestal Calculation
Pedestal Corrections
Noise Calculation
(for MC)

CalibrateAndApplyThreshold

Zero Suppression
Calibration

Trigger Fifo

Trigger Assignment
TriggerDelay
Nominal Mainword

TriggerBits
Event Flags

ADC Values

Cell Mapping
Module /Cell
Dimensions

Calib Const.

RawCaloHits

LCCaloHits

LCIO Stream

RawCaloHits

Conditions Data

SimCaloHits

Noise. Const.

MC Branch:

TBEcalDigitisationProcessor

Smearing of Calo Hits
Adding of cells not
appearing in MC

LCIO File

Graphic following template
by G. Gaycken

Reconstructed LCIO files are entry point for newcomers

... and starting point of high level analysis

Main Line:

Should contain only objects as defined by the LCIO data model
Contain e.g. 'familiar' CalorimeterHits
Principle violated for testbeam tracking (can/should be changed)
Unavoidable that the reco files contain 'calice specific'
data which can however be accessed by the userlib functions
No additional information e,g, from database is needed
(as the analyses become sophisticated it looks as if this principle
cannot be maintained anymore, see later)

Future:

Reco files are to be more exploited by an calice_analysis package
which is under discussion
e.g. Different clustering algorithms, PFAs (?)
Track Extrapolation

Rule: Classes needed to interpret the 'calice specific' data types go into userlib
Classes needed for analysis and go beyond interpretation go to analysis package

Reconstruction Input

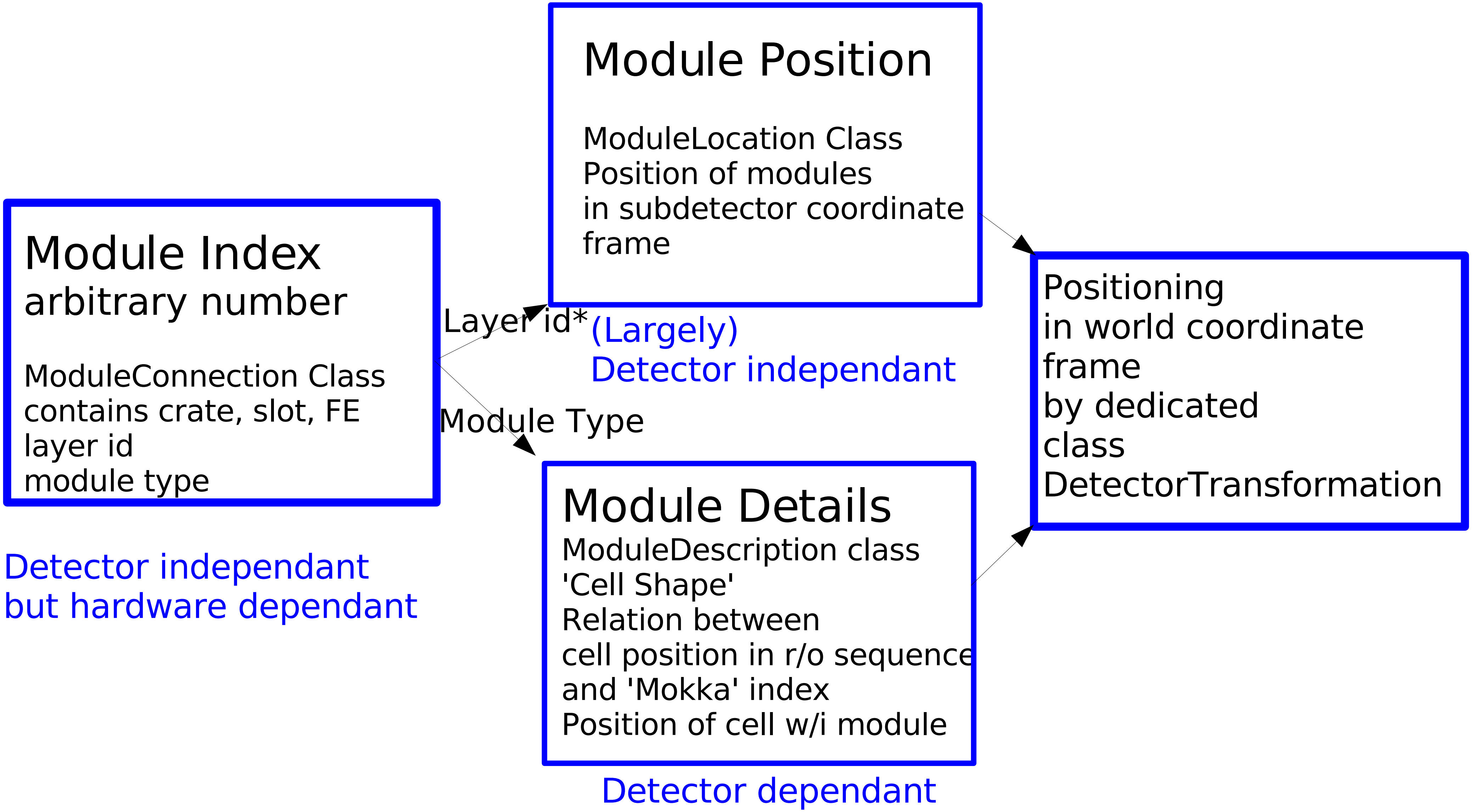
- Triggerinformation

TriggerMapping (ie. Bits to meaning) defined in database
One common class to handle triggers
Evaluation of trigger info is tightly coupled to datatypes delivered by calice DAQ

- Mapping and Alignment

Common classes structure for (currently) three calice detectors
Detector specific information is stored in database (e.g. Relation hardware index 'Mokka' index)
Access functions are the same for all detectors

Mapping and Alignment - Details



Module Index

arbitrary number

ModuleConnection Class
contains crate, slot, FE
layer id
module type

Detector independant
but hardware dependant

Layer id*

Module Type

Module Position

ModuleLocation Class
Position of modules
in subdetector coordinate
frame

(Largely)
Detector independant

Module Details

ModuleDescription class
'Cell Shape'
Relation between
cell position in r/o sequence
and 'Mokka' index
Position of cell w/i module

Detector dependant

Positioning
in world coordinate
frame
by dedicated
class
DetectorTransformation

Scheme invented for SiW Ecal by G. Gaycken and adopted by AHCal and TCMT

*+ indicator to account for vertical subdivision of Ecal

Additional Complications

- Calice

takes data at different Locations CERN, DESY and FNAL (in 2008)
sometimes even parallel
There could have been in principle parallel datataking
of several detectors at the same location

- Marlin

Program execution is piloted by a steering file

⇒different steering files for the reconstruction job
Mainly due to different database folders
Details on database see this afternoon

Steering Files - Details

- DESY Ecal Running 2006

1 Steering File

- CERN Running 2006

3 steering files for three running modes
ecal, hcal only combined running
In practice 4 to account four periods with missing Hcal Constants

Missing calibration constants lead to a significant slow down of job execution time due to large number of thrown exceptions due to missing database entries (can be improved by redesign of Hcal calibration folders in database)

- CERN Running 2007

Three steering files (same as 2006)

- FNAL Running 2008

DHCAL and ScintEcal to be integrated
Additional steering files needed depending on 'detector' permutations

Steering Files – Details cont'd

- Different steering files (for DESY and CERN Running) created from two (three) template steerings during submission of grid jobs

One for desy and one(two) for cern, the latter again due to missing calibration constants for parts of the running
At FNAL we should able to work with one template steering

- Automatization?

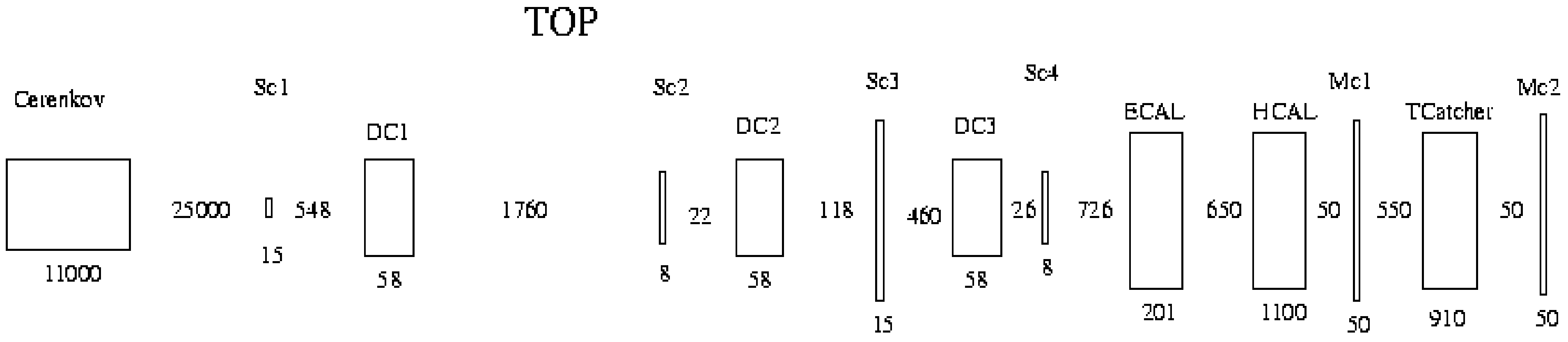
i.e. Automatic recognition of experimental setup and fetching of database constants w/o steering

Running the reconstruction is expert work (i.e. Composition of the steering)
Considerable Effort to automize the correct fetching of db constants
Propose to stick to the current concept

Users can nevertheless fetch a working steering file from grid
The used steering is copied together with the reconstructed run

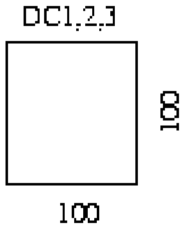
A view to the Monte Carlo Branch

- Model for the simulation of the CERN (and DESY)test beam is available (in release 06-04-p03 of Mokka)



FRONT

Sc1 is 30x30
Sc2 and Sc4 are 100x100
Sc3 is 200x200



Mc1 and Mc2 are 1000x1000

All distances are in mm

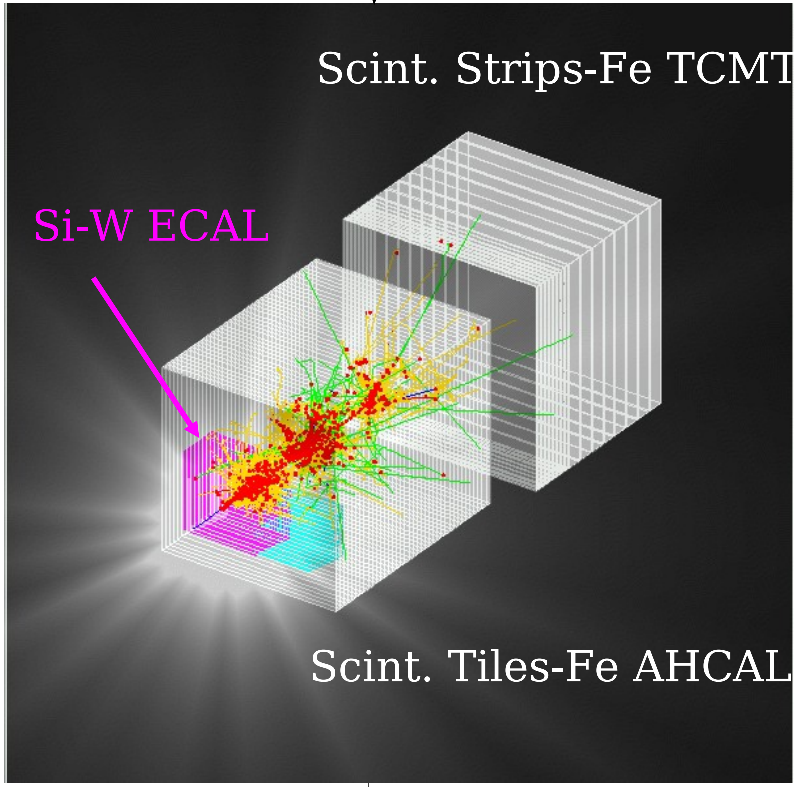
DESY, RAL, LLR

Common effort of groups at RHUL, DESY, LLR, NIU

Do use grid for MC production

Geometry Definitions: Mokka database <-> Calice database

Mokka db for Simulation



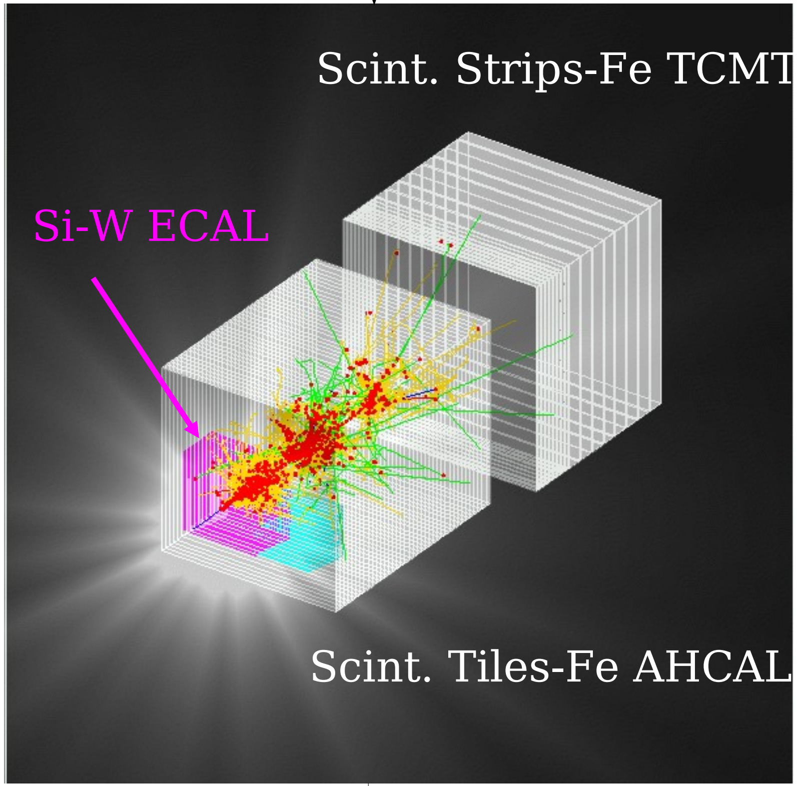
Might lead to conflicts if simulated setup different from reality

Needed to simulated run conditions beam energy etc.

Calice db for Reconstruction

Geometry Definitions: Mokka database <-> Calice database

Mokka db for Simulation



Might lead to conflicts if simulated setup different from reality

Envisaged step
Unification
i.e. Feed mokka drivers from calice db

Calice db for Reconstruction

Needed to simulated
run conditions
beam energy etc.

On Digitisation and Strategy to Produce MC

- Currently two different ways to digitize SiW Ecal and Ahcal, i.e. Noise overlay

SiW Ecal stores (average) noise in database
Ahcal intends to work with noise event overlay

Honest: I don't enough insight in the details to say more on this at this stage

- Since experimental conditions may vary on a run by run basis (e.g. Dead Channels) simulation has to be done according to specific run conditions

i.e. MC and data files for each run using the same reco version

Summary and Conclusion

- Calice has a working software chain based on

  ILC Software

  Grid Infrastructure

- Approach allows for analysis of e.g. Ecal data at > 6 different institutes without major startup problems and expert knowledge

- Common classes to define e.g. geometries for different detectors In principle every calice detector can use these classes

- General Deficits in design due to the following reasons
  - Common approach not always in mind of collaborators
  - Reconstruction software started out from Ecal (therefore strong Ecal Legacy but other detectors were kept in mind) At a given point we had to take what was there Lead to the fact that badly tailored s/w survive in the software
  - Design phase basically in parallel to data taking

This is true for many parts of the calice software

Growing (user) community which understands the advantage of a common approach ask for a sharpening of rules and underlying concepts

That's why we are here

Annex A

Software packages needed by

Experts:

calice_reco (+further dependencies), calice_userlib,
(calice_analysis)
LCIO, Marlin, LCCD, CondDBMySQL

Users:

LCIO, calice_userlib (indispensable)

(calice_analysis (recommended))
Marlin (highly recommended)
LCCD, CondDBMySQL
(not needed in first stages of analysis
but cannot be excluded, see later)

Installation of software will be facilitated by application of cmake building environment which become the standard in calice!!!!

Annex B: Main Coding Rules or guidelines

Formulated for the first time here

- Code has to be written in c++ (Extension to java can be considered for the analysis package)

- Write code platform independent !!!!

- Put comments into the code and prepare for doxygen documentation

- Avoid global variables, follow the principle of data encapsulation

- Always Initialize variables

- No hardcoded parameters, everything has to be steerable

- Use stdlib container classes when dealing with arrays
  Even on the expense of performance penalty

- In general, use stdlib methods whereever possible

- use c++ methods and not c like methods

- Don't work with pointers (or justify why you cannot avoid it)

- Don't use 'new' operator (or justify why you cannot avoid it)

- Avoid termination of a program in case you enter an odd situation in your package, i.e. No 'assert' (can be used for debugging)
  Rather use std::exception mechanism and 'throw try catch'

- Avoid complicated inheritance structures or templates (or justify why you need them)

Part II:
Conditions Data Handling

Based on a talk I gave at the Calice Collaboration Meeting
NIU March 2005

which in turn makes largely use of talks
by F. Gaede on the topic

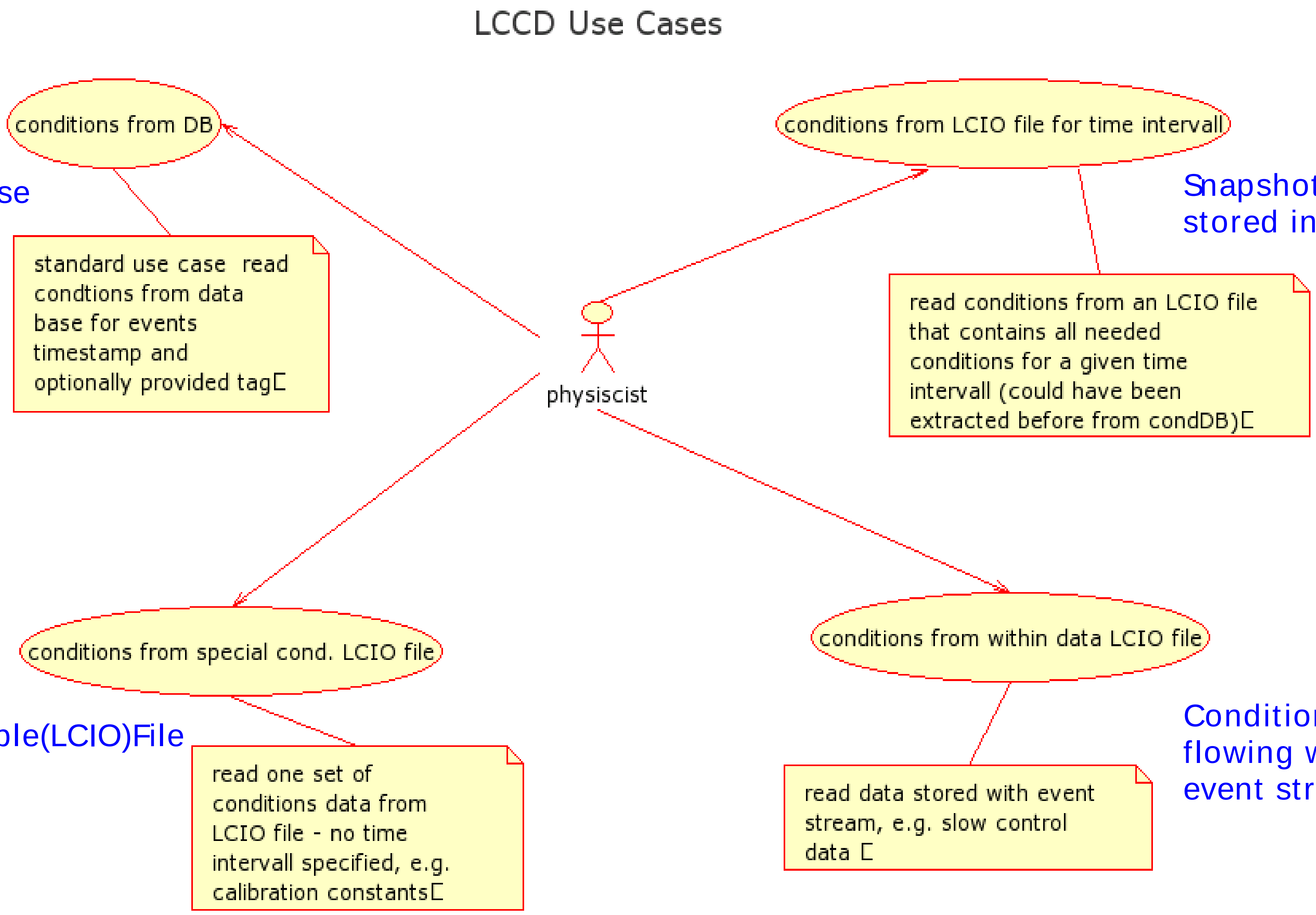
Introduction to LCCD

- LCCD – Linear Collider Conditions Data Framework:
  - Software package providing an Interface to conditions data
    - database
    - LCIO files

    Author Frank Gaede, DESY

- Conditions Data:
  - all data that is needed for analysis/ reconstruction besides the actual event data
  - typically has lifetime (validity range) longer than one event
    - can change on various timescales, e.g. seconds to years
    - need for tagging mechanism, e.g. for calibration constants

Sources of Conditions Data – Use Cases



LCCD Use Cases

conditions from DB

Database

standard use case read condtions from data base for events timestamp and optionally provided tag□

conditions from LCIO file for time intervall

Snapshot of DB stored in LCIO File

read conditions from an LCIO file that contains all needed conditions for a given time intervall (could have been extracted before from condDB)□

physiscist

conditions from special cond. LCIO file

Simple(LCIO)File

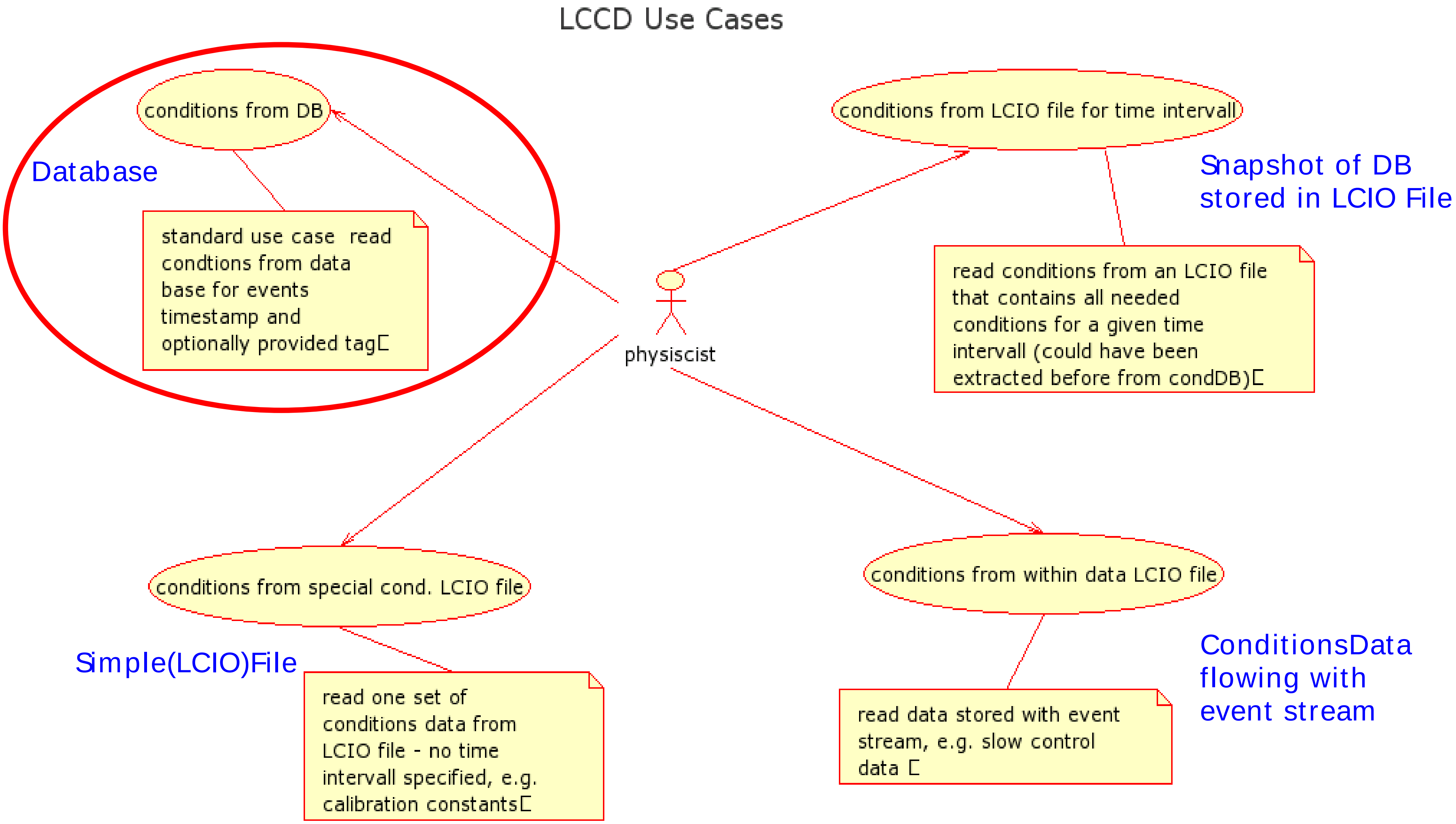
read one set of conditions data from LCIO file - no time intervall specified, e.g. calibration constants□

conditions from within data LCIO file

ConditionsData flowing with event stream

read data stored with event stream, e.g. slow control data □

Sources of Conditions Data – Use Cases



LCCD Use Cases

conditions from DB

Database

standard use case read condtions from data base for events timestamp and optionally provided tag

conditions from LCIO file for time intervall

Snapshot of DB stored in LCIO File

read conditions from an LCIO file that contains all needed conditions for a given time intervall (could have been extracted before from condDB)

physiscist

conditions from special cond. LCIO file

Simple(LCIO)File

read one set of conditions data from LCIO file - no time intervall specified, e.g. calibration constants

conditions from within data LCIO file

ConditionsData flowing with event stream

read data stored with event stream, e.g. slow control data

ConditionsDBMySQL – Overview

Digged out and explored out by Frank Gaede for us
Interfaced to LCCD by Frank

- Open source implementation of CondDB API
  - Conditions data interface for ATLAS (Cern IT)

- developed by Lisbon Atlas group
- features
  - C++ interface to conditions database in MySQL
  - data organized in folder/foldersets
  - objects stored as BLOBs (binary large objects)

    e.g. LCIO objects or std::vector .....
  - tagging mechanism similar to CVS
  - scalability through partitioning options
  - outperforms implementation based on Oracle

ConditionsDBMySQL – Folder Organization

```
+--------+---------+---------------------+-------------------------+----------------------------------
+-------+----------+---------+----------+
| fld_id | fparent | insert_t        | fpath                | fdesc                         | fattr  | ddtype   | db_id  |  is_se   |
+--------+---------+---------------------+-------------------------+----------------------------------
+-------+----------+---------+----------+
|      1 |       0 | 20050210202708 | /                        | romans new folder           |       |        0  |      1 |       1 |
|      2 |      1  | 20050210202708 | /roman                   | romans new folder           |       |       0  |      1 |       1 |
|      3 |       2 | 20050210202708 | /roman/mapfolder        | romans new folder           |       |        0  |       1 |       0 |
|      4 |       2 | 20050210202943 | /roman/calibfolder      | romans new calib folder     |       |        0 |       1 |       0 |
|      5 |       1 | 20050214183955 | /lccd                    |                             |       |       0 |       1 |       1 |
|      6 |       5 | 20050214183955 | /lccd/myhcal             |                             |       |        0 |      1 |        0 |
|      7 |       1 | 20050301151849 | /lccd_calice             |                             |       |       0 |      1 |       1 |
|      8 |       7 | 20050301151849 | /lccd_calice/CellMap |                             |       |       0 |      1 |       0 |
+--------+---------+---------------------+-------------------------+-------------------------
+------------------------------------+-------+---------+----------+----------+
```

- UNIX-Like Tree Structure
- Each Folder Contains one set of ConditionsData

  LCCD provides Streamer Methods to read LCIO data types back from DB
- Access via Folder Name
- Folders have to be filled by us (templates do exist)

ConditionsDBMySQL – Versioning of Conditions Data

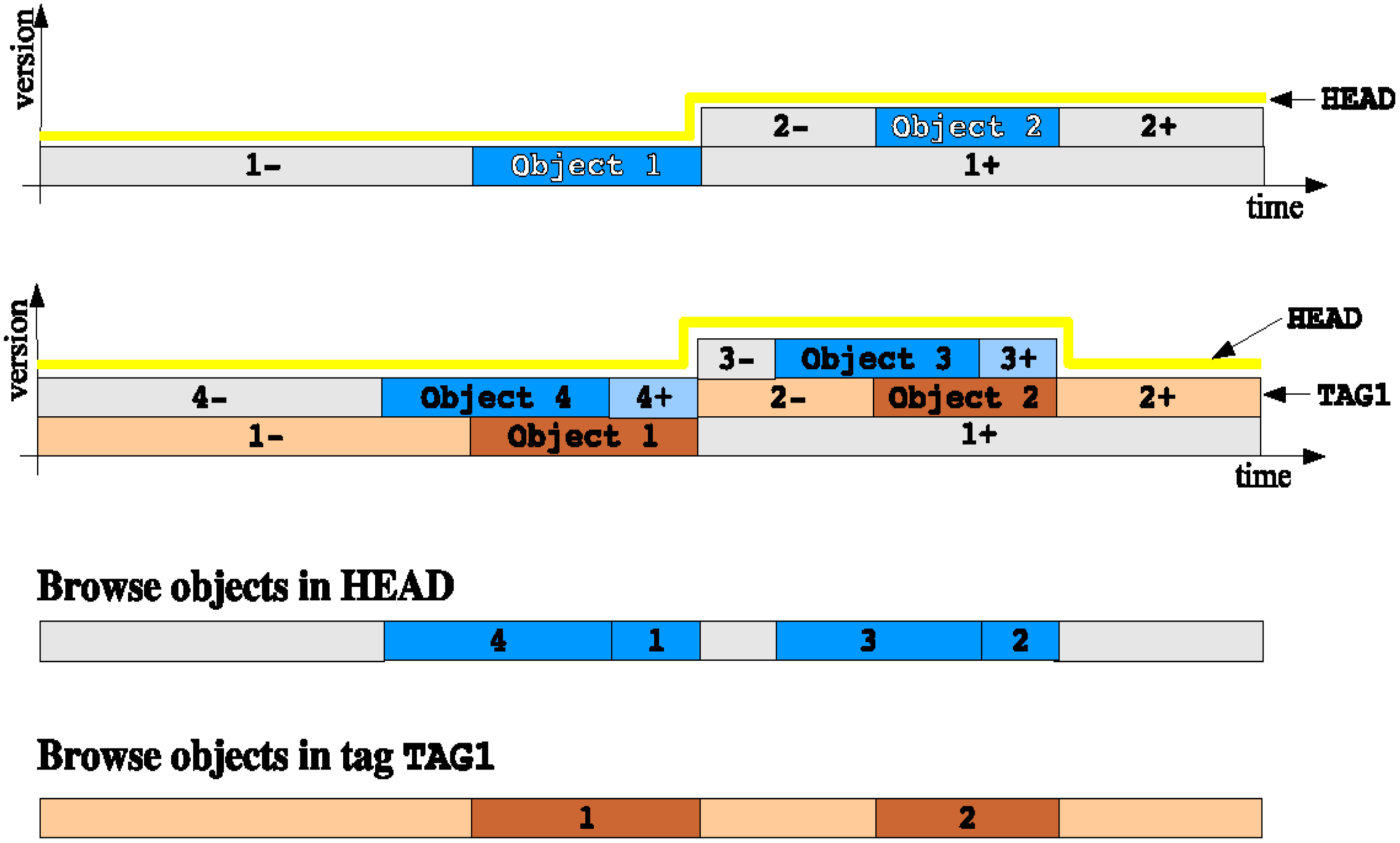


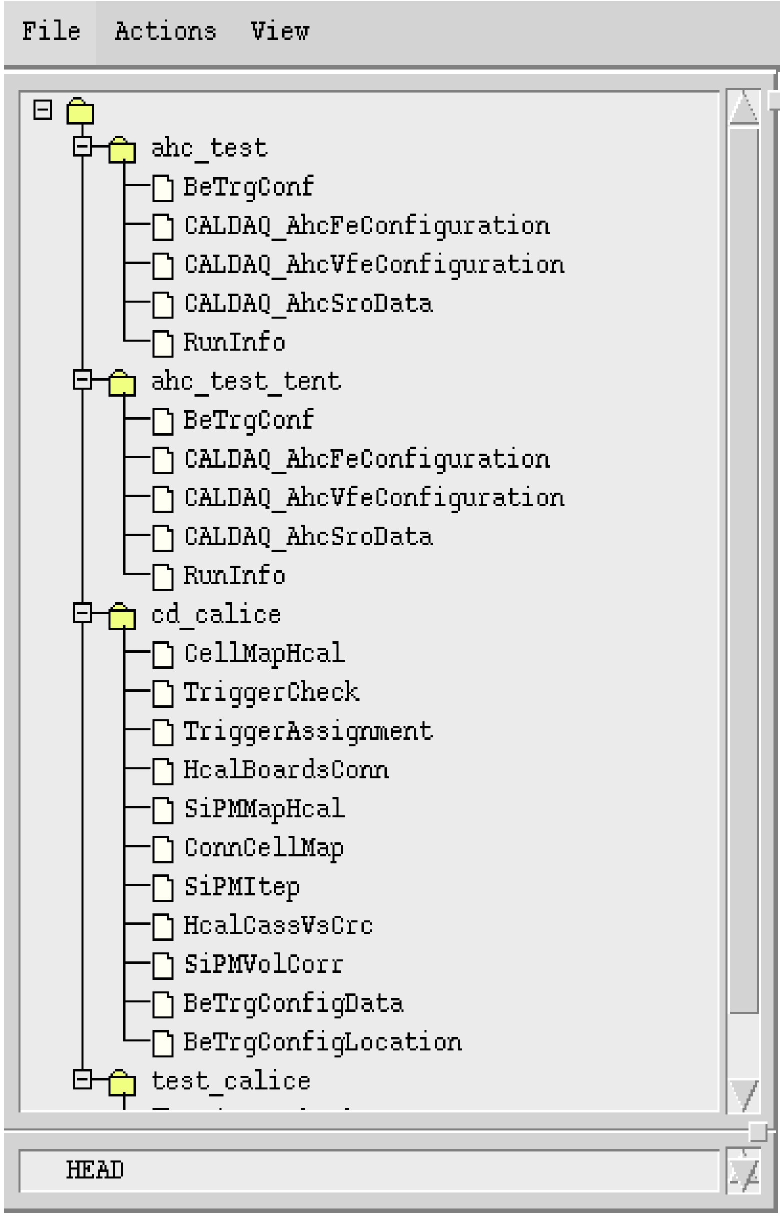
Figure 3: tagging and browsing example in the ConditionsDB mySQL's implementation.

CVS- like management system

'Horizontal' and vertical browsing in time possible

Time Stamp (by LCCD) in units of nanoseconds

CALICE Database Hosted by DESY



```
⊟ 
  ⊟ ahc_test
      BeTrgConf
      CALDAQ_AhcFeConfiguration
      CALDAQ_AhcVfeConfiguration
      CALDAQ_AhcSroData
      RunInfo
  ⊟ ahc_test_tent
      BeTrgConf
      CALDAQ_AhcFeConfiguration
      CALDAQ_AhcVfeConfiguration
      CALDAQ_AhcSroData
      RunInfo
  ⊟ cd_calice
      CellMapHcal
      TriggerCheck
      TriggerAssignment
      HcalBoardsConn
      SiPMMapHcal
      ConnCellMap
      SiPMItep
      HcalCassVsCrc
      SiPMVolCorr
      BeTrgConfigData
      BeTrgConfigLocation
  ⊟ test_calice
```

Trigger Info: Assignment of triggerbits
Trigger Configuration
Info to validate Trigger information

Calibration Data

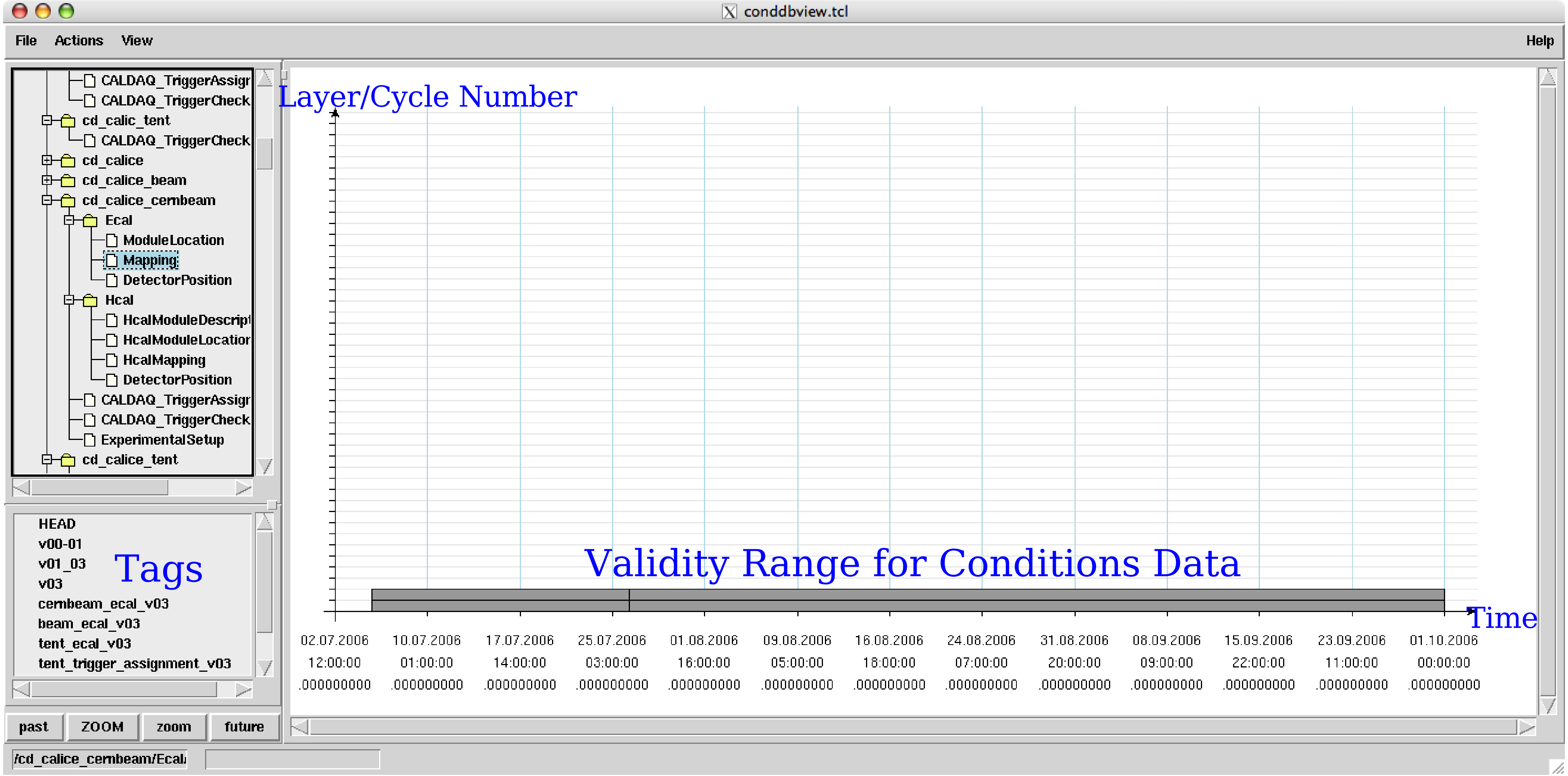
Cell Mappings: Relation electronic channel
and
geometrical channel
i.e. Cabling of devices

Hardware configuration during data taking.

Attempt to visualize Conditions Data
(S.Schmidt, M.Schenk, R.P.)

Largely used by R.P. To inspect data
Might need to be advocated to larger community

Conditions Data in CALICE Database



Layer/Cycle Number

Tags

Validity Range for Conditions Data

Time

Organization of Conditions Data in terms of time stamps requires the creation of different folders in case of parallel data taking!!!

Acessing ConditionsData Using LCCD – Users Point of View

MARLIN is prepared to deal with Conditions Data
(Note: LCCD does not depend on MARLIN and vice versa)

- Source of ConditionsData defined in MARLIN steering File

  e.g. ConditionsData for Cell Mapping from DB

  ```
  DBCondHandler channelmap /lccd_calice/CellMap V00-01
  ```

- Handling of Conditions Data (updating etc.) within a ConditionsProcessor (provided by MARLIN)

  User has to provide ChangeListeners which will be notified if Conditions Data Change

- Code is completely transparent to Conditions Data Source

Access to Conditions Data should always happen via the LCCD Interfaces!!!!

And yes, this implies that users have to learn LCCD!!!
Stronger user support by (calice) experts clearly needed

Conditions Data and Systematic Studies

- The running of a database is indispensable for and experiment as big as calice

  allows e.g. for quick reproduction of running conditions

- The interface to the database (or better conditions data) may lack convenience

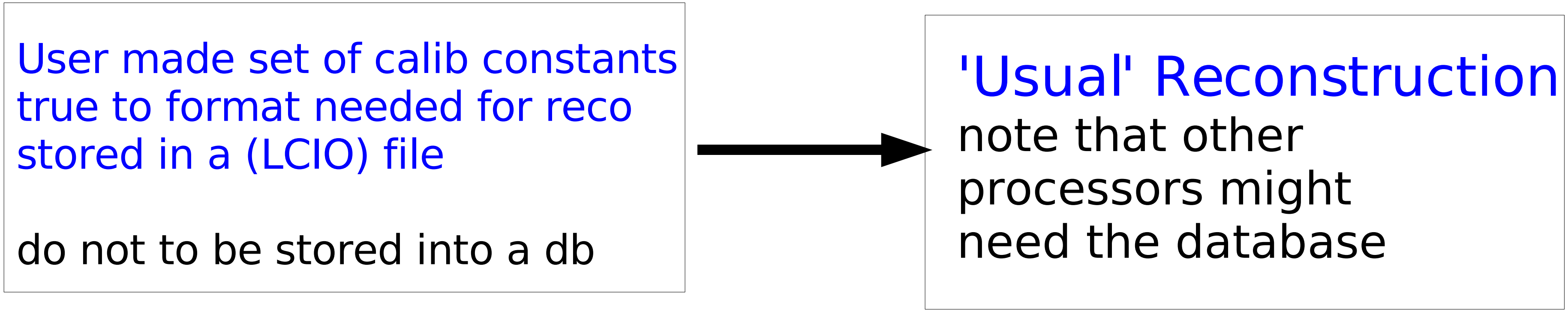
  However, all studies needed can be done with the available software

- As of today systematic studies as e.g. varying the calibration constants do need a re-running of the reconstruction (different collections in raw and reco files)

  In principle no database access is needed for these
  LCCD allows for other sources (see above)
  => New set of calibration constants can be put into (LCIO) files

  The following use cases can be imagined =>

Systematic Studies – Use Cases

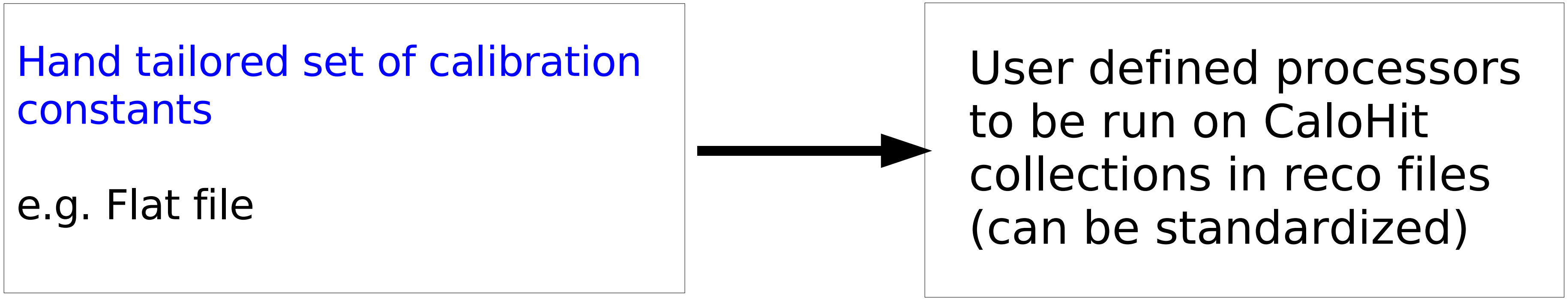


User made set of calib constants true to format needed for reco stored in a (LCIO) file

do not to be stored into a db

'Usual' Reconstruction
note that other processors might need the database

Please note that conditions data does not mean automatically literal database access, can e.g. work with a snapshot of the db
It means however using the LCCD interfaces!!!



Hand tailored set of calibration constants

e.g. Flat file

User defined processors to be run on CaloHit collections in reco files (can be standardized)

The latter scenario however might lead to conflicts with other cuts applied earlier in reco (e.g. noise cuts)

Conditions Data – Critical Issues

- Starting Point: Nothing else but LCIO needed to work with reco files (+userlib to interpret calice specific data stored in LCGenericObjects), i.e. no LCCD

- ~95% of the conditions data are handled currently in the reco job and hidden from the user (i.e. Non expert)

(As experience and sophistication of analysis grows)

Analysis might require to access conditions data not yet handled

Users have to be ready to use LCCD also during analysis since it is difficult to predict what might be needed

- There are sets of conditions data useful for analysis, e.g. Dead Cell Map

a) Only Storage in database

b) Identify these data and attach them to the (first) event in addition to the storage in the database

a) is the cleanest solution (and my proposal)!!!
In any case access to be performed via LCCD interface
Easy benefit from updates of this map

- Studies of different Alignments

Looks like we have to store the difference to the default alignment into the file – Details to be discussed

Conditions Data Handling – General Comment, Guidelines and Improvements(?)

General Comment:

- The handling of ~95% of the conditions data
  is already handled and hidden from the user
  e.g. Details of the trigger handling
  (Default) Alignment
  Application of Calibration Constants

Guideline:

- (Again) If conditions data apart from these are needed
  use the Lister Pattern

Improvements(?):

- Trust the handling of 'identified as being important'
  conditions data to manager/handler
  classes which may act as Singleton
  User can (even outside processors) use the interface
  classes to access important conditions data
  Done already for the trigger handling

The latter is a still a bit doubtful since it has no correspondenc in ILC Software
and will remain naturally calice tailored/calice specific

Conditions Data - Summary

- Conditions Data are an integral part of the calice data processing

  First experiment withing ILC which produce these data extensively

- LCCD allows to handle conditions data in a clean way

  Common interfaces for different sources

- Users have to be prepared to use LCCD

  Still a major step which users like to circumvent

  Better training, examples needed

- Need to define a strategy to handle conditions data which are intimately needed for analysis or which might need to be changed for systematic studies at the 'core' of the data processing, i.e. During MC simulation