Designing calorimeter digitization software for the test beam

Dhiman, Guilherme

NORTHERN ILLINOIS UNIVERSITY

DHCal Meeting at NIU
November 24, 2004

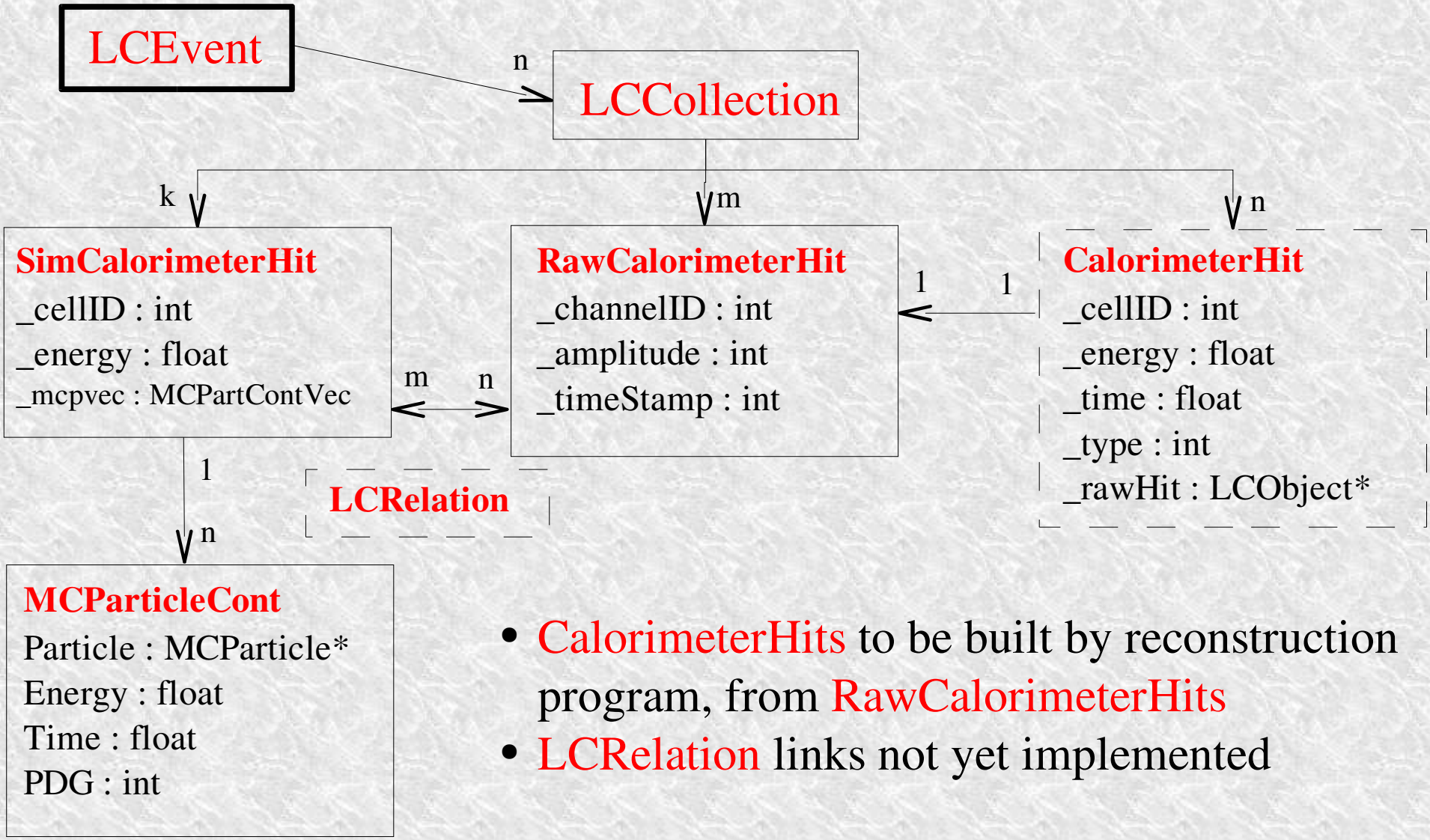
Basic requirements

- Goal: develop a program to simulate the digitization system, to be used for the test beam. Resulting modules may be used as a basis for the digitization of the full detector simulations.
- All test beam code is based on C++.
- Use LCIO for persistency, and Marlin as the framework.
- Object oriented design to simplify maintenance and implementation of new functionality.

Digitizer role

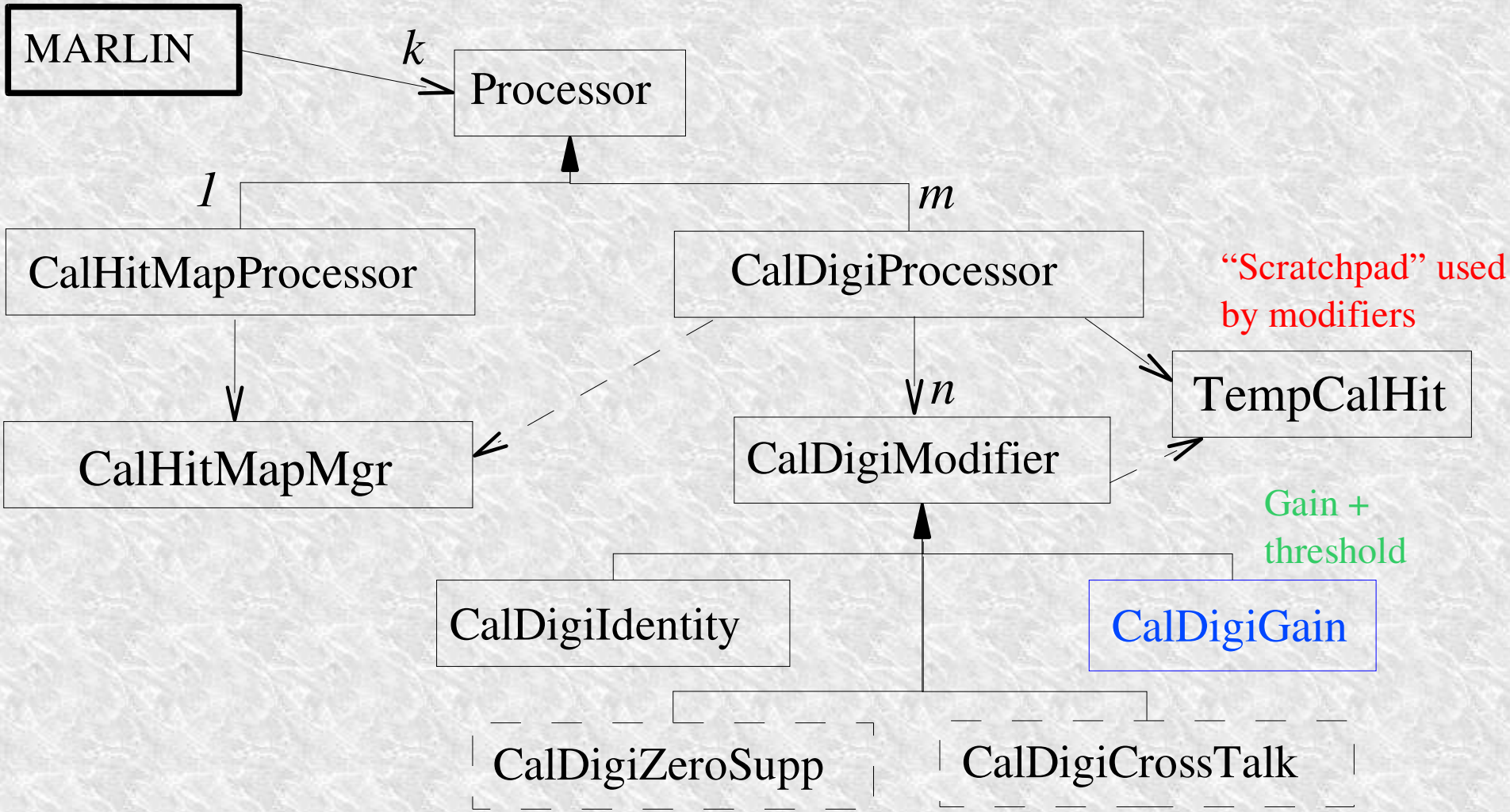
- Detector simulation (Geant4) produces ideal energy depositions in detector *cells*
- Real detector produces real hits, with ADC counts and time stamps in readout *channels*
- Basic role of digitizer is convert the energy depositions and timings into electronics readout, so that simulation output can be as close as possible to the detector output. Same reco / analysis code can be used for both.
- *As close as possible* means that all known data acquisition effects should be taken into account, if possible (inefficiencies, noise, crosstalks, non-uniformities, etc.)

LCIO event model



- CalorimeterHits to be built by reconstruction program, from RawCalorimeterHits
- LCRelation links not yet implemented

CalDigi class diagrams



Steering file example (1/3)

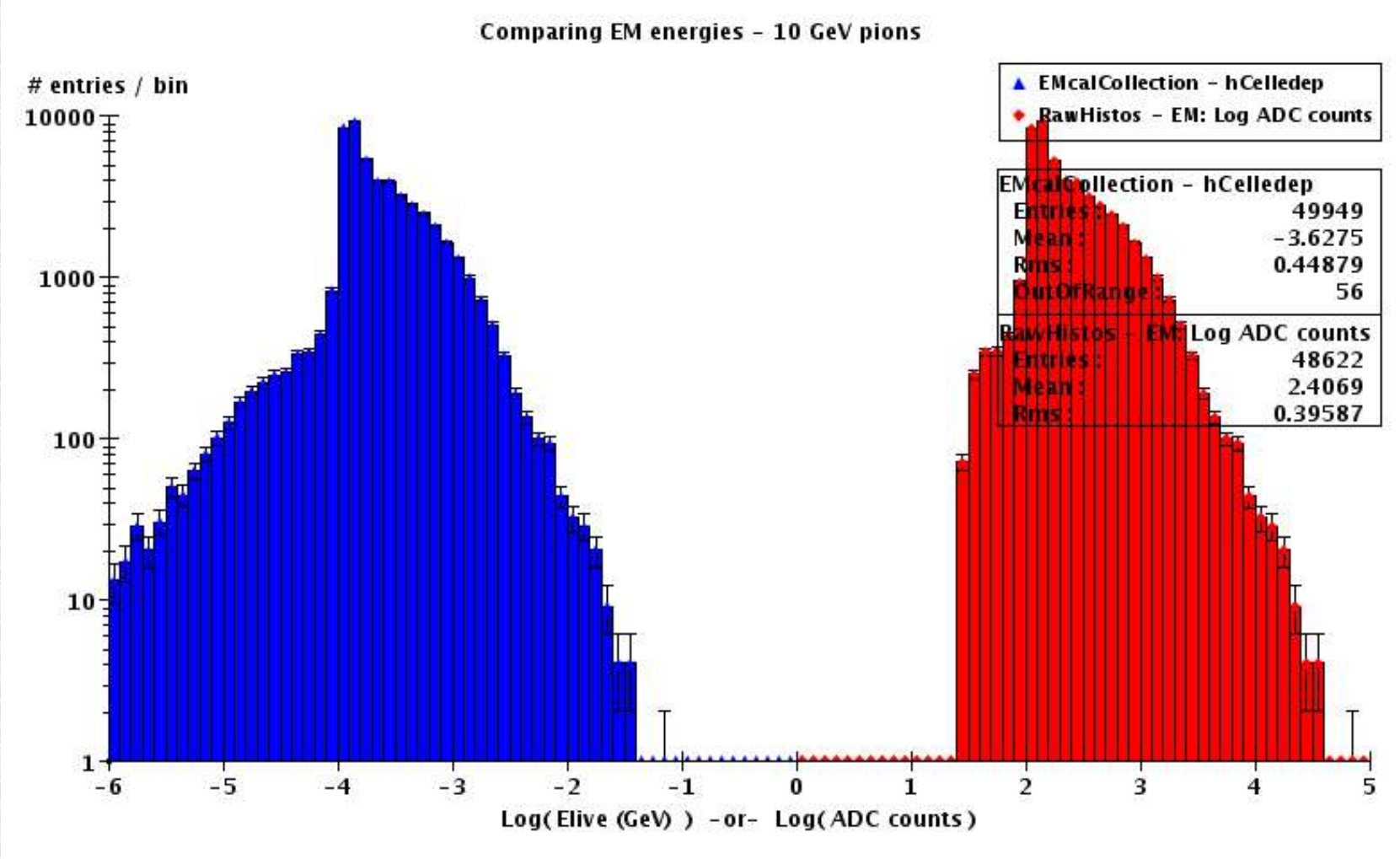
```
##############################################
#  Example steering file for Marlin
##############################################

.begin Global  ------------------------------------------
# specify one ore more input files (in one ore more lines)
LCIOInputFiles inputfile

# the active processors that are called in the given order
ActiveProcessors CalHitMapProcessor
ActiveProcessors EMCalDigiProcessor
ActiveProcessors HADCalDigiProcessor
ActiveProcessors OutputProcessor

# limit the number of processed records (run+evt):
MaxRecordNumber 500
.end Global  ------------------------------------------
```

Steering file example (cont.)

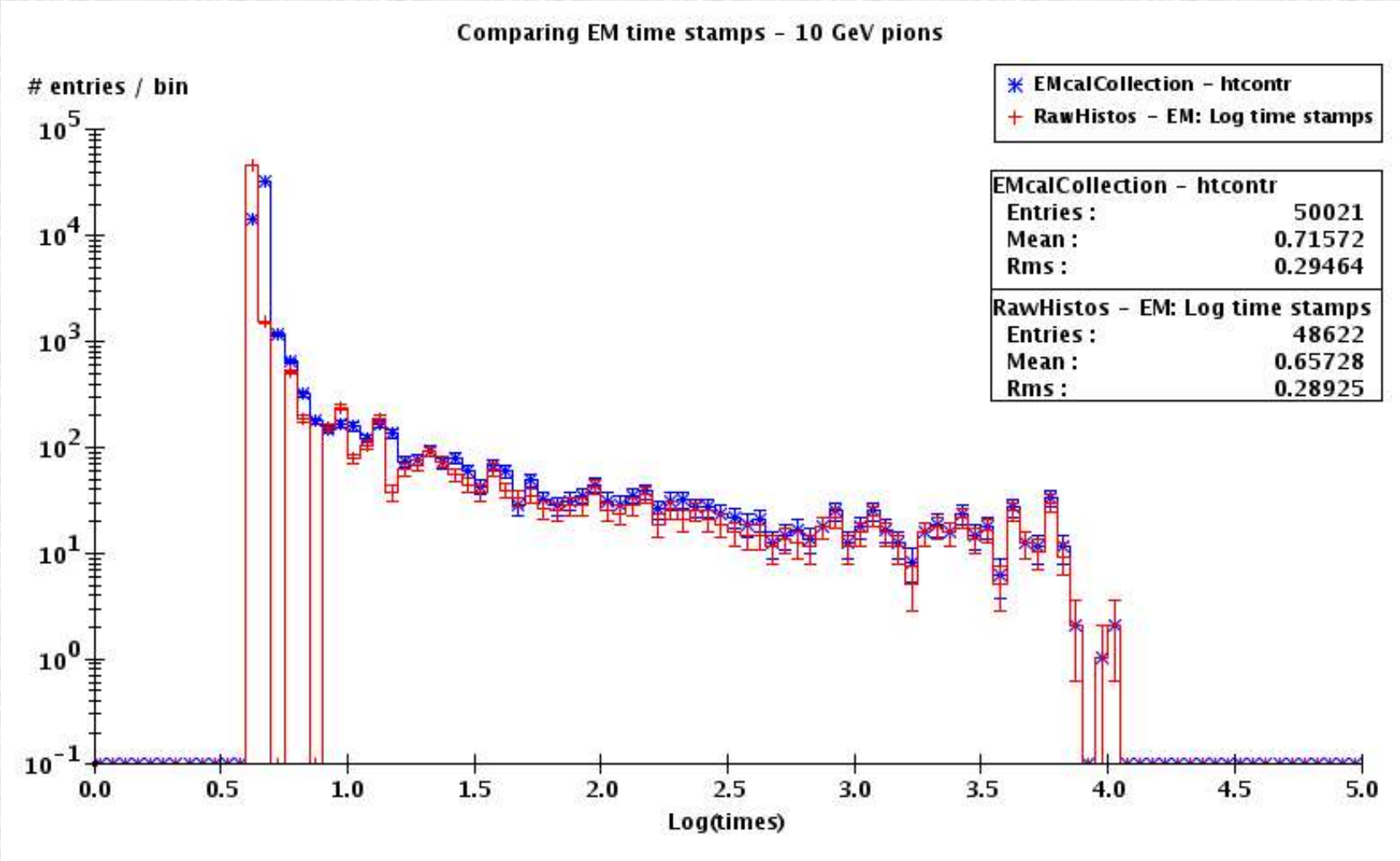
```
#################################################################
# Cal digitizer processor.  Instantiates one or more calorimeter hit
#  "modifiers", which together represent the full digitization process.
.begin EMCalDigiProcessor ----------------------------------------------
ProcessorType CalDigiProcessor

InputCollection  EMcalCollection
OutputCollection EMrawCollection
CalorimeterType  EM

ModifierNames    EMFixedGain EMThreshOnly

# Parameters are: gainNom  gainSigma  thresh  threshSigma
EMFixedGain       1000000          0       0        0
EMThreshOnly            1          0      30        0

EMSinglePass      1000000          0      30        0

EMGaussianGain    1000000      50000       0        0
EMGainThresh      1000000      50000      30      1.5
.end -------------------------------------------------------------------
```

(As many as needed)

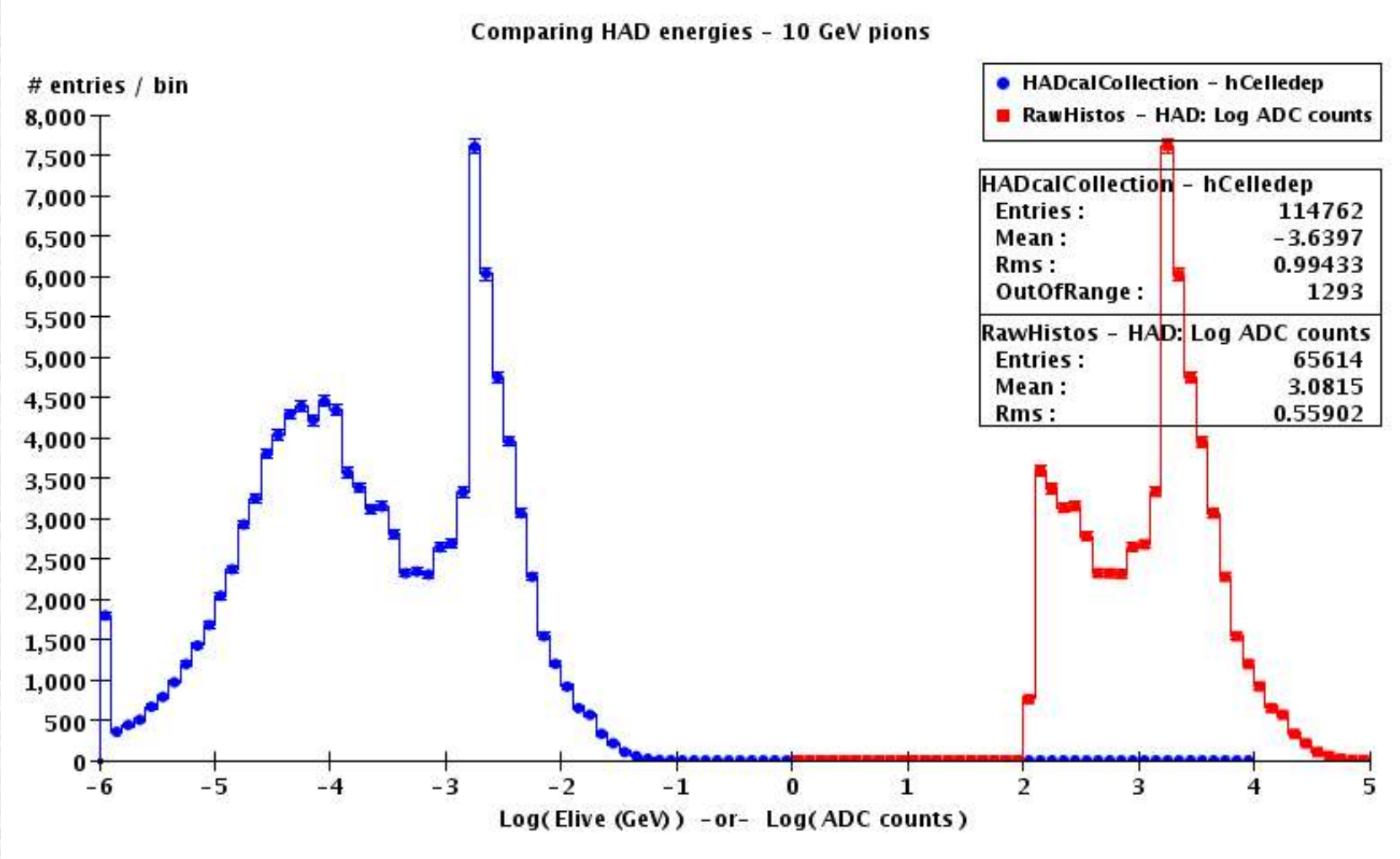
10 GeV $\pi^{\pm}$ on EM: gain + threshold



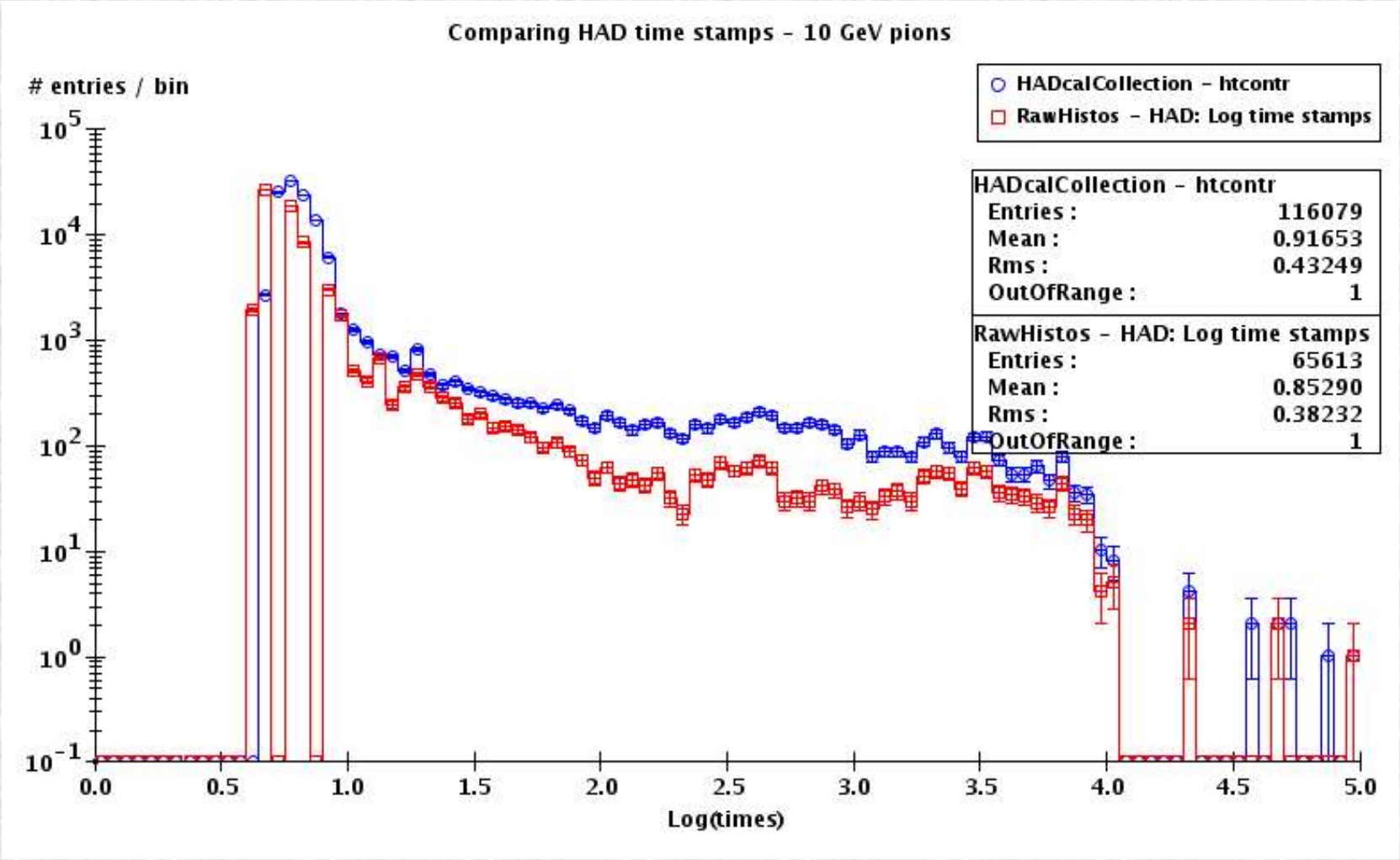
10 GeV $\pi^{\pm}$ on EM: time stamps



10 GeV $\pi^{\pm}$ on HAD: gain + threshold



10 GeV $\pi^{\pm}$ on HAD: time stamps



Status

- A first version (proof of concept) is implemented
  - First real tool implemented: gain + threshold. No “saturation” effect for now.
  - LCRelation: to be implemented (soon?), example code at hand (thanks to F.Gaede)
- Output LCIO files contain EM and HAD RawCalorimeterHit collections, while keeping simulation collections untouched.
- Analysis code for raw hits, plots confirm expected behavior.
- Creation of new modifiers is quite easy, by just copying the existing modifier class (*CalDigiGain*) and implementing the desired transformation.
- Crosstalk requires cell-neighborhood. NonProj code exists (java and C++), but projective geometry is available only in java.

Some open questions

- Requirements on ordering of modifiers execution (like crosstalks before hot channels)
- Variables the modifiers can depend on?
  - Now available: cellID, energy deposition and timing
  - Not available: space points (x,y,z) and cell neighborhood
    Need a geometry-aware class to provide missing information