Proposal of new structure

Taikan Suehara
(Kyushu University)

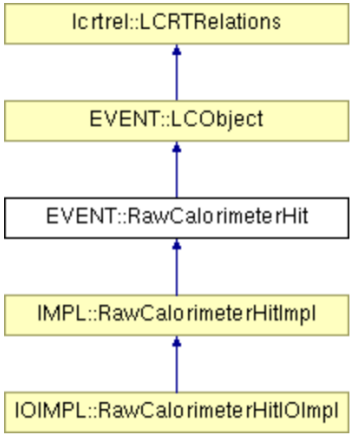
RawCalorimeterHit

EVENT::RawCalorimeterHit Class Reference

The generic calorimeter hit for real data (or simulation thereof). More...

#include <pre-generated/EVENT/RawCalorimeterHit.h>

Inheritance diagram for EVENT::RawCalorimeterHit:



List of all members.

Public Types

typedef RawCalorimeterHit	**lcobject_type** Useful typedef for template programming with **LCIO**.

Public Member Functions

virtual	**~RawCalorimeterHit** () Destructor.
virtual int	**getCellID0** () const =0 Returns the detector specific (geometrical) cell id.
virtual int	**getCellID1** () const =0 Returns the second detector specific (geometrical) cell id.
virtual int	**getAmplitude** () const =0 Returns the amplitude of the hit in ADC counts.
virtual int	**getTimeStamp** () const =0 Returns a time stamp for the hit.

Simple object having 4 integers

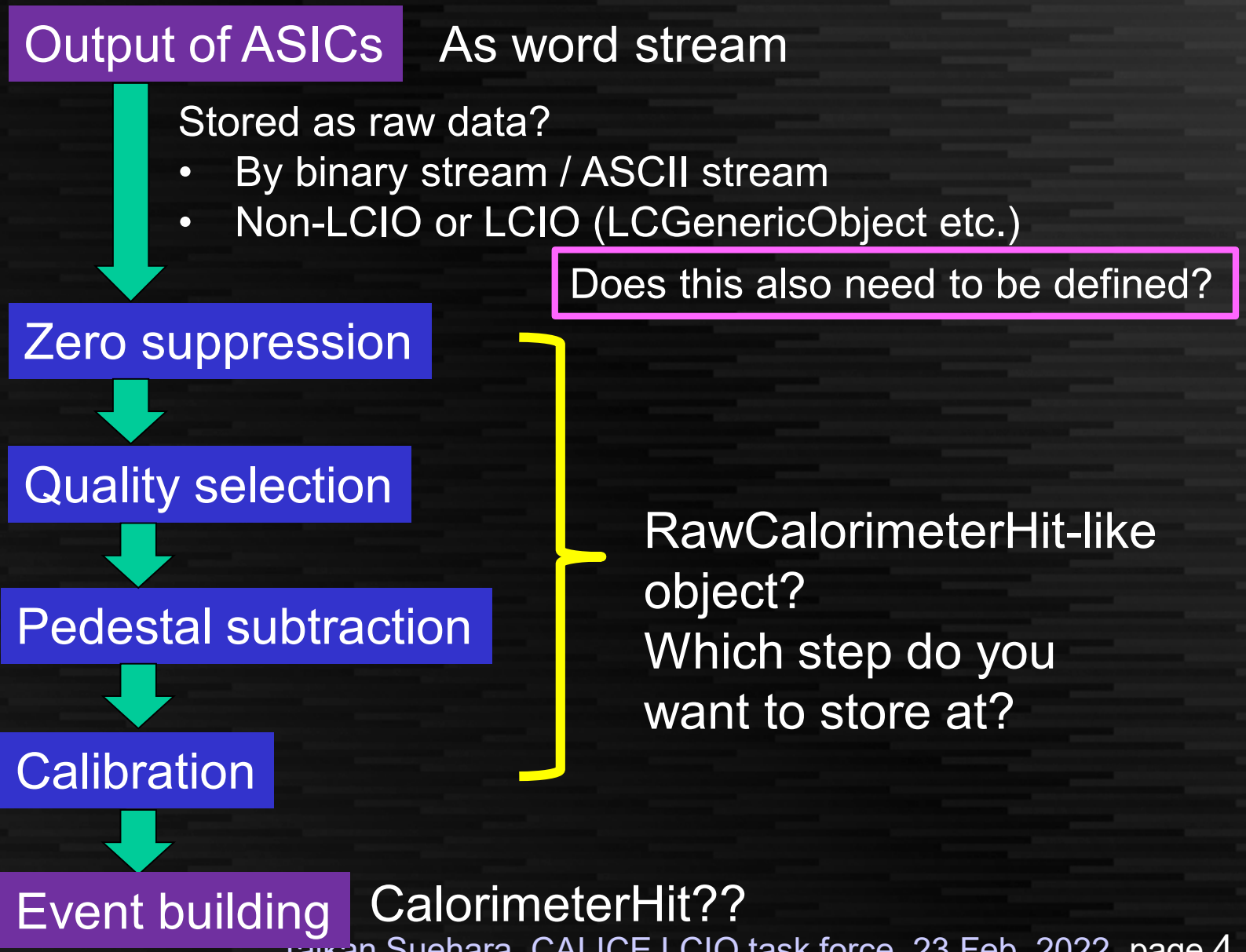
- CellID0
- CellID1
- Amplitude
- TimeStamp

Not enough?

Target of new RawCalorimeterHit

- Currently
  - Only used in SDHCAL
    - Silicon: currently not using LCIO for raw level
    - AHCAL: using LCGenericObject
- Current data structure is not enough to store all necessary data of all subsystems
  - BXID + precise TDC
    - Bits used are different by subsystems
  - ADC high + low gain
  - Trigger bits, etc.

Event data flow (in my understanding)



Extended data structure

Data	Current	Proposed	Comment
Cell ID	4 / 8 bytes	4 / 8 bytes	
Amplitude	4 bytes	4 / 8 bytes	High/low gain
Timestamp	4 bytes	4 / 8 bytes	For both BXID and hi-reso TDC
Flag	-	0 / 4 bytes	Gain, Trig bits

Try to add flexibility to store more data depending on subsystems

- LCIO has CHBIT_xxx flags to enable/disable data fields
  - Defining more CHBIT flags to control the extention
  - May not be kept at new key4hep-based systems
  - One possibility is to fix it maximal and rely on compression
- Accessors to use them as word (2 bytes) arrays are useful

Extended data structure: proposed code

```
int _cellID0 ;
int _cellID1 ; // selective with bit CHBIT_ID1
int _amplitude0 ; // renamed from _amplitude
int _amplitude1 ; // newly introduced; selective with bit eg. CHBIT_RC_AMP1
int _timeStamp0 ; // renamed from _timeStamp
int _timeStamp1 ; // newly introduced; selective with bit eg. CHBIT_RC_TIME1
int _flags ; // newly introduced; selective with bit eg. CHBIT_RC_FLAGS
// minimum 4+4+4=12 bytes, maximum 8+8+8+4=28 bytes
```

--

with selective flags (CHBIT_***) of enabling newly introduced 4+4+4 bytes.

Comments from Frank (1)

- as you know we are working towards the transition to a new EDM package EDM4hep in the context of Key4hep
  - EDM4hep has an EDM that is very much based on LCIO yet there are some subtle differences, e.g. it will not be possible to have bit flags that steer the storing of parts of the classes content
  - as CALICE might want to move EDM4hep at some point in the future, we should at least keep this in mind when designing a new class for CALICE raw data
- Questions
  - No plans to implement dynamic control of data size in EDM4hep?
  - How serious should we think of the transition?

Comments from Frank (2)

- adding an amplitude1 is no problem per se (has been done in the past w/ cellID -> cellID0/cellID1) however moving to 2 ints with the goal of later optionally interpreting it as 4 shorts seems a bit clumsy
  - here one could maybe add a vector<int16_t> or directly int16_t[4] ?
  - adding a timeStamp1 is also of course possible in principle - as w/ cellIDs
  - if I would do it now, I would simply store the 64 bit quantity from the start and rely on compression, so the class could look sth. like:

    int64_t cellID
    int64_t timeStamp
    int64_t amplitude (or int16_t[4])

I think we can live with any types; can think of implementation point of view

Questions from Frank

- I don’t fully understand your proposal for the flags: what is stored here and what is the granularity, i.e. should this be int32_t or bytes and how flexible is the actual length (number) of these objects ?
- are these really flags (bits) ?

Answer:

- I think of a trigger bit, high/low gain bit, data quality bits etc. which can be used freely by each subsystems. 32 bits should be enough.

Interfaces to add

- Accessors for different data sizes (amplitude / timestamp)

eg. for amplitude

virtual int getAmplitude() const ; // return Amplitude0, for backward compatibility
virtual int getAmplitudeShort(int index) const ; // index 0-3 to access 4 x short
virtual int getAmplitudeLong(int index) const ; // index 0-1 to access 2 x long

- Macros for easier readability (options)
  - make macros to identify index eg.
    getAmplitudeShort(SICAL_ADC_LOWGAIN)
  - make macros for accessor eg.
    #define getSicalADCLowGain() getAmplitudeShort(0)
  - (make inheritant classes for individual subsystems with those accessors)

Questions (from me to experts)

- Those can be implemented in LCIO or
  should we prepare eg. CALICE-only header files for them?

Other discussions/questions

- Can the backward compatibility (to read old data with new LCIO library) be kept or should we change the class name to keep it?

- Other questions/comments/suggestions?

Possible schedule

- Kickoff (September)
  - Assign chair (myself)
  - Discussing mandate
- Discussion 1 (November)
  - Collect requirements from each subsystem (Silicon, Scintillator, RPC)
- First proposal (December – January)
  - Communication with software experts
- Discussion 2 (February) – this meeting
  - Discussion of first proposal (with software experts?)
- Discussion 3
  - Final decision?
  - Starting to write a summary document