Improving the LCIO/SIO file format

A Proposal

Frank Gaede
DESY, March 2007

SIO/LCIO shortcomings

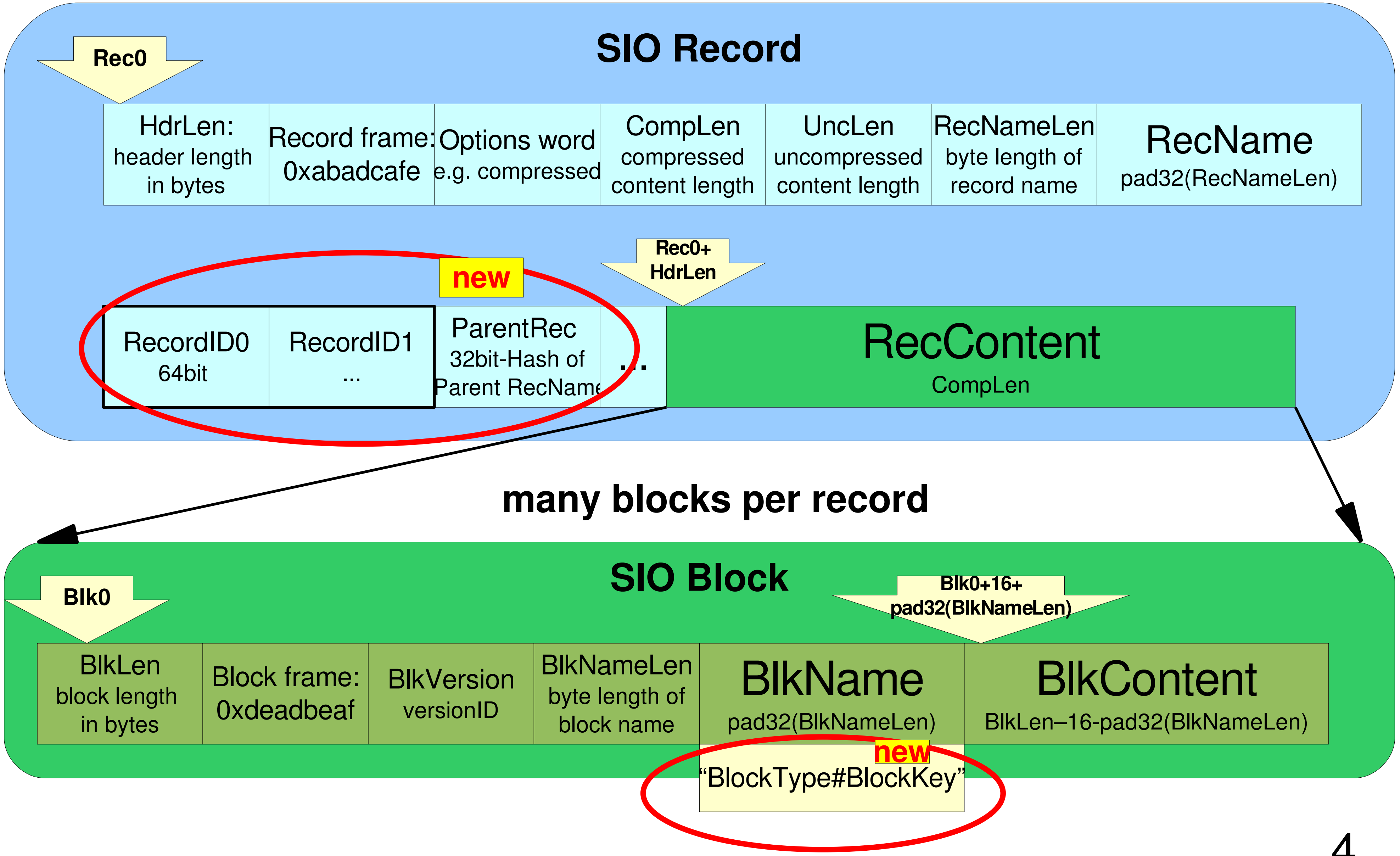
- LCIO/SIO fairly successful – however with growing user community some shortcomings start to become relevant:
  - no direct access
  - event records can't be split:
    - large file (event) sizes
    - poor I/O performance when only subset of data is needed
  - event data can't be distributed over several files
  - storing user defined data:
    - somewhat inconvenient with LCGenericObjects
    - imposes performance penalty (LCGenericObjects)
  - using LCIO in testbeam DAQ systems requires (very) fast I/O (as little overhead as possible)

Proposed SIO modifications

- all of the described shortcomings can be addressed fairly easily with a few minor modifications to the SIO file format:
  - as suggested by Tony Johnson direct access can be provided through an additional directory record stored at the end of every file with file pointers (long64 for >2GByte files) –
  - such a directory record can easily be (re)created by reading the record headers only (no data unpacking/interpreting) provided that we introduce a recordID (preferably 64 bit) in the header
  - by adding a 32bit key for a parent record one can group records into subrecords (e.g. EventHeader + several data records)
  - using the existing block names to store a BlockType and BlockKey e.g. separated by '#' on can read every block independent of type information from a parent record (LCEventHeader)
  - introducing a FileHeader record with a UUID (128bit) allows to distribute data among several files
    - the 'parent' file can be found through lookup via the stored parent UUID

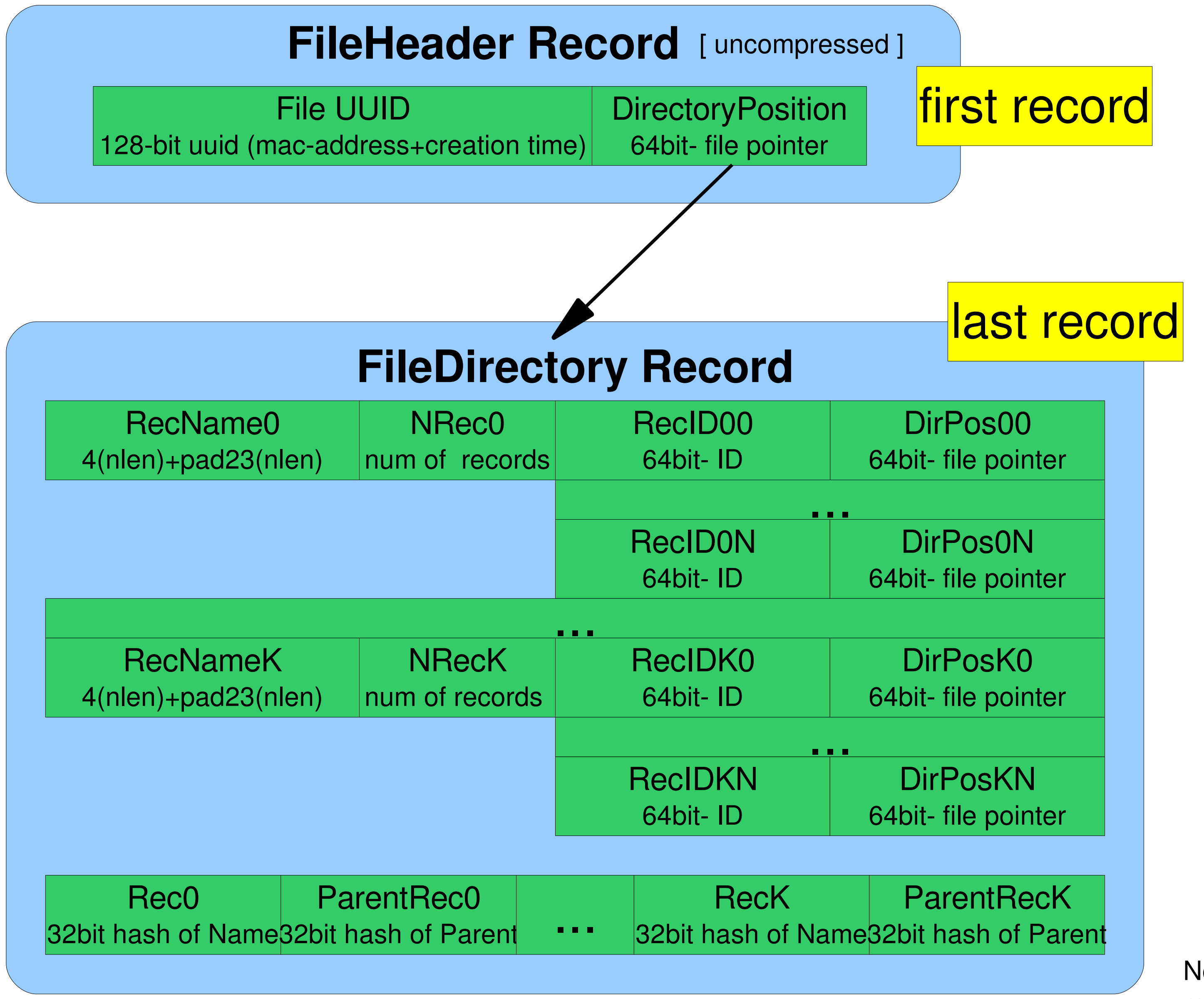
see next slides for implementation suggestions

Modified SIO file format



SIO Record
Rec0
HdrLen: header length in bytes
Record frame: 0xabadcafe
Options word e.g. compressed
CompLen compressed content length
UncLen uncompressed content length
RecNameLen byte length of record name
RecName pad32(RecNameLen)
new
RecordID0 64bit
RecordID1 ...
ParentRec 32bit-Hash of Parent RecName
...
Rec0+ HdrLen
RecContent CompLen
many blocks per record
SIO Block
Blk0
BlkLen block length in bytes
Block frame: 0xdeadbeaf
BlkVersion versionID
BlkNameLen byte length of block name
Blk0+16+ pad32(BlkNameLen)
BlkName pad32(BlkNameLen)
BlkContent BlkLen–16-pad32(BlkNameLen)
new
"BlockType#BlockKey"

Two new SIO records



allows hierarchy
of SIO/LCIO
files

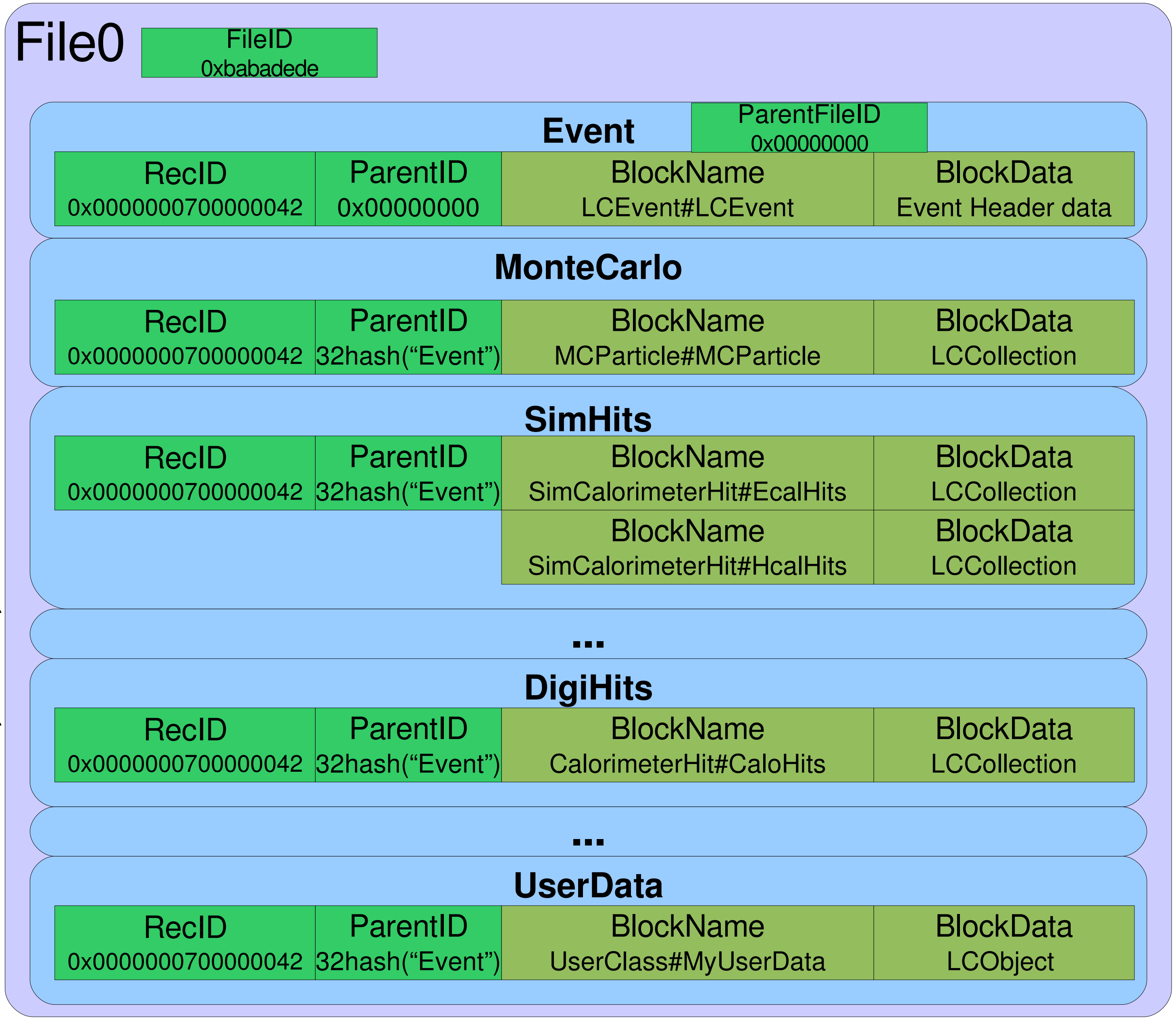
map of record
positions per
record type (name)
and ID
-> direct access
to every record !
-> allow extension
of event at the end !

allows hierarchy
record types, i.e.
subrecords

Note: FileDirectory record will be
rewritten at end of file with every close
after write access !

example: LCIO event split into subrecords

Frank Gaede, DESY, 2007



File0

FileID
0xbabadede

Event

ParentFileID
0x00000000

RecID	ParentID	BlockName	BlockData
0x0000000700000042	0x00000000	LCEvent#LCEvent	Event Header data

MonteCarlo

RecID	ParentID	BlockName	BlockData
0x0000000700000042	32hash(“Event”)	MCParticle#MCParticle	LCCollection

SimHits

RecID	ParentID	BlockName	BlockData
0x0000000700000042	32hash(“Event”)	SimCalorimeterHit#EcalHits	LCCollection
		SimCalorimeterHit#HcalHits	LCCollection

...

DigiHits

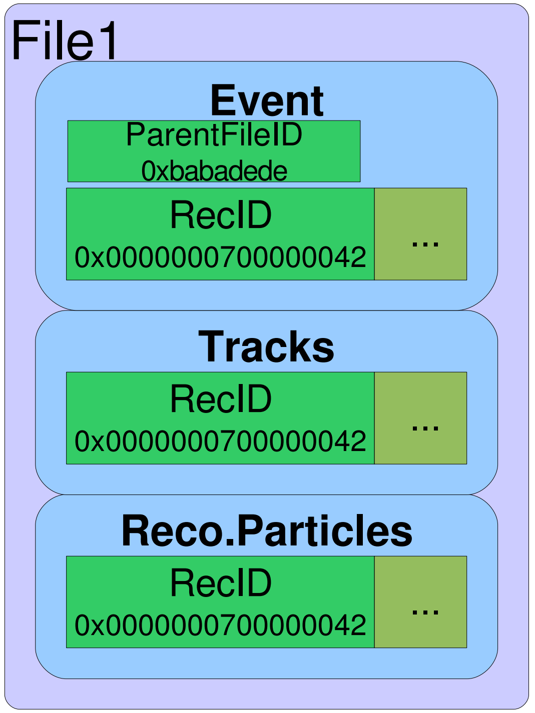
RecID	ParentID	BlockName	BlockData
0x0000000700000042	32hash(“Event”)	CalorimeterHit#CaloHits	LCCollection

...

UserData

RecID	ParentID	BlockName	BlockData
0x0000000700000042	32hash(“Event”)	UserClass#MyUserData	LCObject

- LCEvent subrecords can be
- anywhere in the file
  - identified through ParentID and RecID
- or even in another file
  - linked through the ParentFileID (stored in Event)



File1

Event

ParentFileID
0xbabadede

RecID	
0x0000000700000042	...

Tracks

RecID	
0x0000000700000042	...

Reco.Particles

RecID	
0x0000000700000042	...

example: run 0007 event 042

long64 RecID = (evt->RunNum() << 32 | evt->EvtNum())

Features of new I/O format I

- file directory provides direct access to records
  - direct access to specific events w/o need of fast skip
  - could store non event data (conditions data, histograms etc.) in the same file
- event data can be split into an arbitrary number of records
  - new type#key in block name allows to read records independent of type information in LCEventHeader
  - only requested records need to be read and uncompressed
  - the event can be extended with new collections that are added in new records at the end of the file
  - non-LCIO/user records can be read and attached to the event (access to LCObject pointer from key)

Features of new I/O format II

- event can be distributed among more than on file
  - can have classical HEP model DST like data hierarchy,e.g.
    - full simulated data (hits)
    - digitized hits
    - reconstructed (PFA) objects
    - high level event summary data
  - could store raw data from testbeam DAQ systems in SIO/LCIO records (I/O performance !) and combine with real LCIO later
    - important requirements by EUDET testbeam groups
- for splitting the event a new mechanism for storing pointers is needed -> see next page

New pointer mechanism in LCIO

- Current mechanism depends on SIO feature of pointer relocation within one record
  - store pointedAt and pointerTo integer tags (32bits each)
- proposal to use only pointerTo in SIO independent way:
  - store 64bit LCIOPointer type containing collectionID and index:
    (hash32(colName) << 32 | colIndex)
  - should be set by LCEvent before writting into LCObject::id
  - streamers simply store long64(obj->ptr->id())
  - need functionality in LCEvent class that replaces these links with proper pointers (C++) / references (Java)
  - collisions in hash32 avoided by using hash32 values in event's collection map
  - no increase in data volume for 'normal case' where every objects is only pointed at by one other object
    - slight increase of data volume if objects are pointed at by several other objects
- can use same streamers/data block in any I/O format

SIO Implementation/API changes

- SIO needs to provide a method for reading a record of a given type (name), e.g:
  - SIOReader.readRecord(string name, long64 id=0)
    - if id==0 the next record of the given type is read
  - SIO should provide a method to read all subrecords of a given type, e.g.
  - SIOReader.readSubRecords(string name, long64 id)
    - read all subrecords with the given id for which a suitable streamer/handler is registered
    - will be used to read the next (a specific) event
  - the FileHeader record should be created automatically when a new file is opened for writing
  - the FileDirectory records needs to be created and written automatically at the close()
  - the open("file","read") needs to read the FileDirectory record and optionally (re)create it in case it doesn't exist (e.g.I/O error on a previous close())

LCIO Implementation/API changes I

- LCWriter needs to provide an (optional) way of configuring how the event is going to be split into records, e.g.
  - LCWriter.addSubrecord(“MonteCarlo”,LCIO::MCPARTICLE) ;
    - all MCParticle collections (typically one) will be written to the record MonteCarlo
  - LCWriter.addSubrecord(“HcalSim”,LCIO::SIMCALORIMTERHIT, “HcalBarrel”) ;
  - LCWriter.addSubrecord(“HcalSim”,LCIO::SIMCALORIMTERHIT, “HcalEndcap”) ;
    - Only the specified collections will be written to the record HcalSim
- the logic for distributing the collections to records is then:
  - if a records has been registered for
    - a collection type / name combination use the corresponding record
    - a collection type use the corresponding record
  - else
    - use default record 'Event' ?
- should provide a reasonable default setting, e.g. one record per type
- users can customize the splitting according to their needs
  - mapping could be stored in a dedicated record at the beginning of the file
  - should have a print method in LCWriter and LCReader for the mapping

LCIO Implementation/API changes II

- LCEventHeader needs a field for the ParentFileID and no longer needs the collection types and names
- LCWriter needs a way to know the parent file if any,e.g
  - LCReader rdr ; // ...
  - LCWriter(“ouputfile.slcio”, rdr) ;
  - will set ParentFileID in all written event headers (typically n-1 relation between parent and daughter files, data reduction)
- LCReader needs some mechanism to automatically load parent files if requested (postpone this for now !?) ...
- LCReader needs to provide an (optional) way of specifying which event records are going to be read, e.g.
  - LCReader.readOnly(“MonteCarlo”) ;
  - LCReader.readOnly(“HcalSim”) ;
  - ...
  - default will be to read all (sub)records – the first call to readOnly() will reset the reader to reading only specified records
  - pointers to objects not read will be 0/NULL

Comments ?

- all of the above is a proposal suggested for further discussion
- any feedback, improvement, criticism is welcome