Pixel/IBL Raw Data Decoder

Hide Oide (CERN)

Nov 23, 2015

About

- Pixel/IBL RawDataAnalysis tool was developed initiall for commissioning of Pixel/IBL for Run2.
- Standalone analysis tool
- Offers the following tools:
  - **Skimmer**: extract Pixel or IBL ROB fragments and uncompress the data format to the flat encoding format.
  - **Decoder**: convert a skimmed Pixel or IBL raw data format to a ROOT TTree format.
- The tool was developed on-demand rather than a generic decoder package.
- Not fully robust - ad-hoc adaptation is found elsewhere.

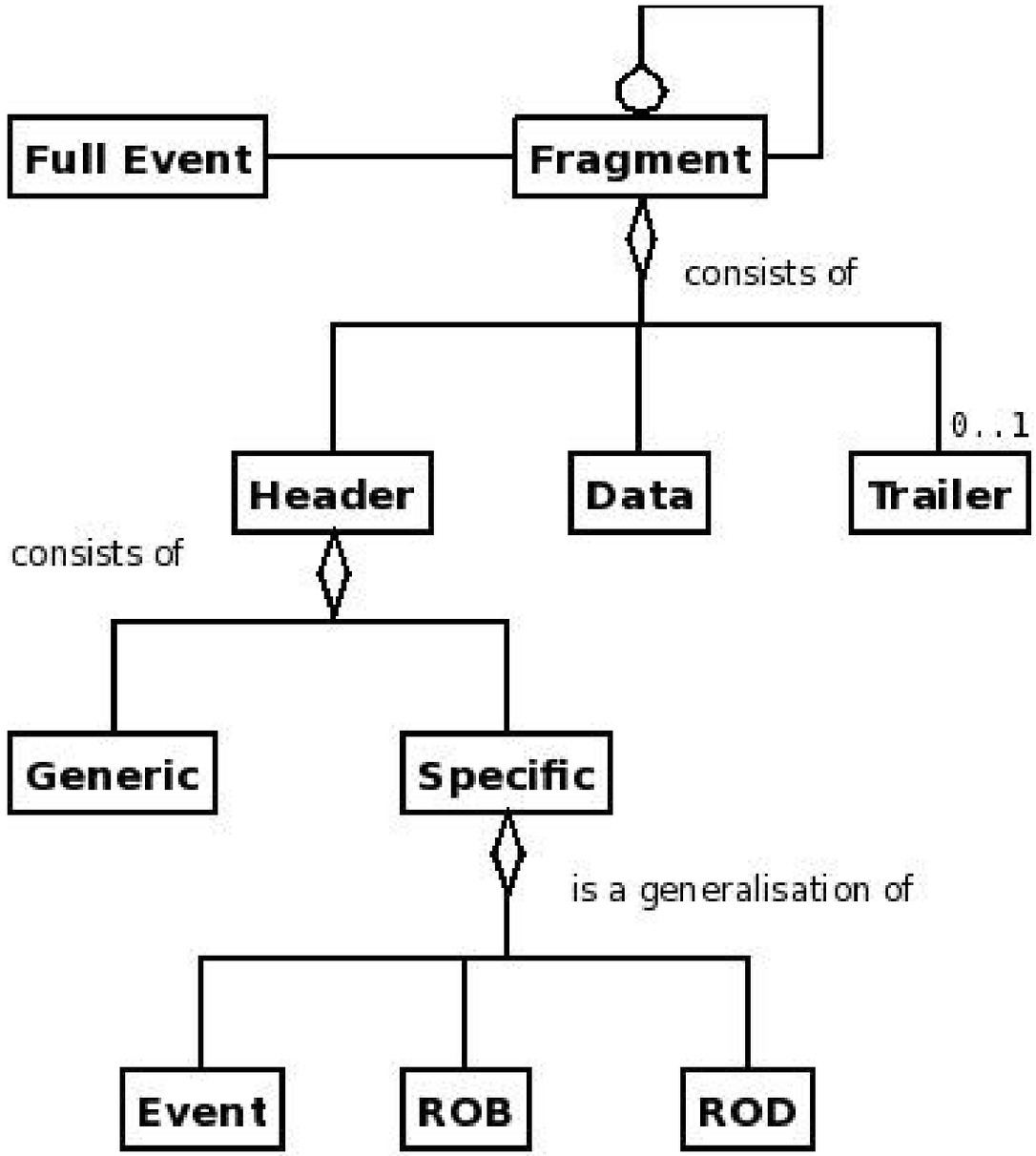
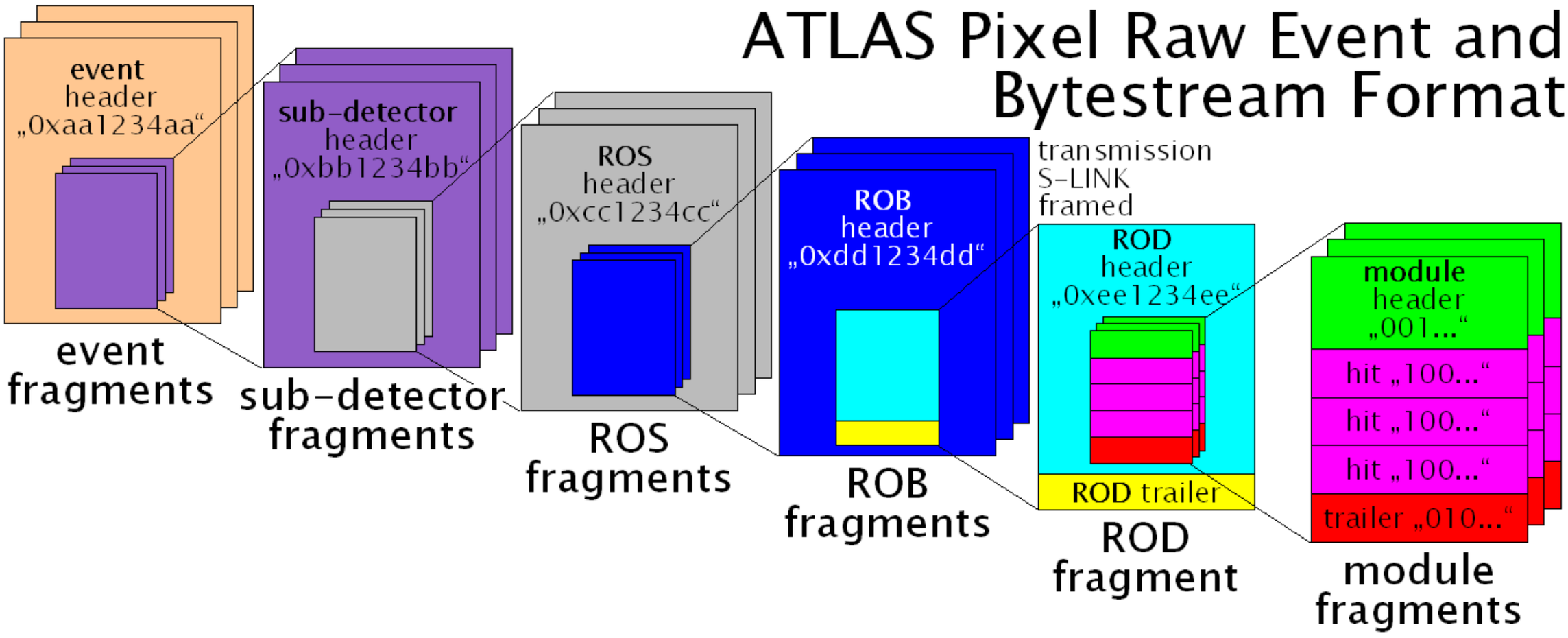


Figure 2: The class diagram of the raw event format.

ATLAS Raw Data Format

- Event format
  - https://edms.cern.ch/document/445840/5.0a
- File format
  - https://edms.cern.ch/document/580290/6



Connectivity Table

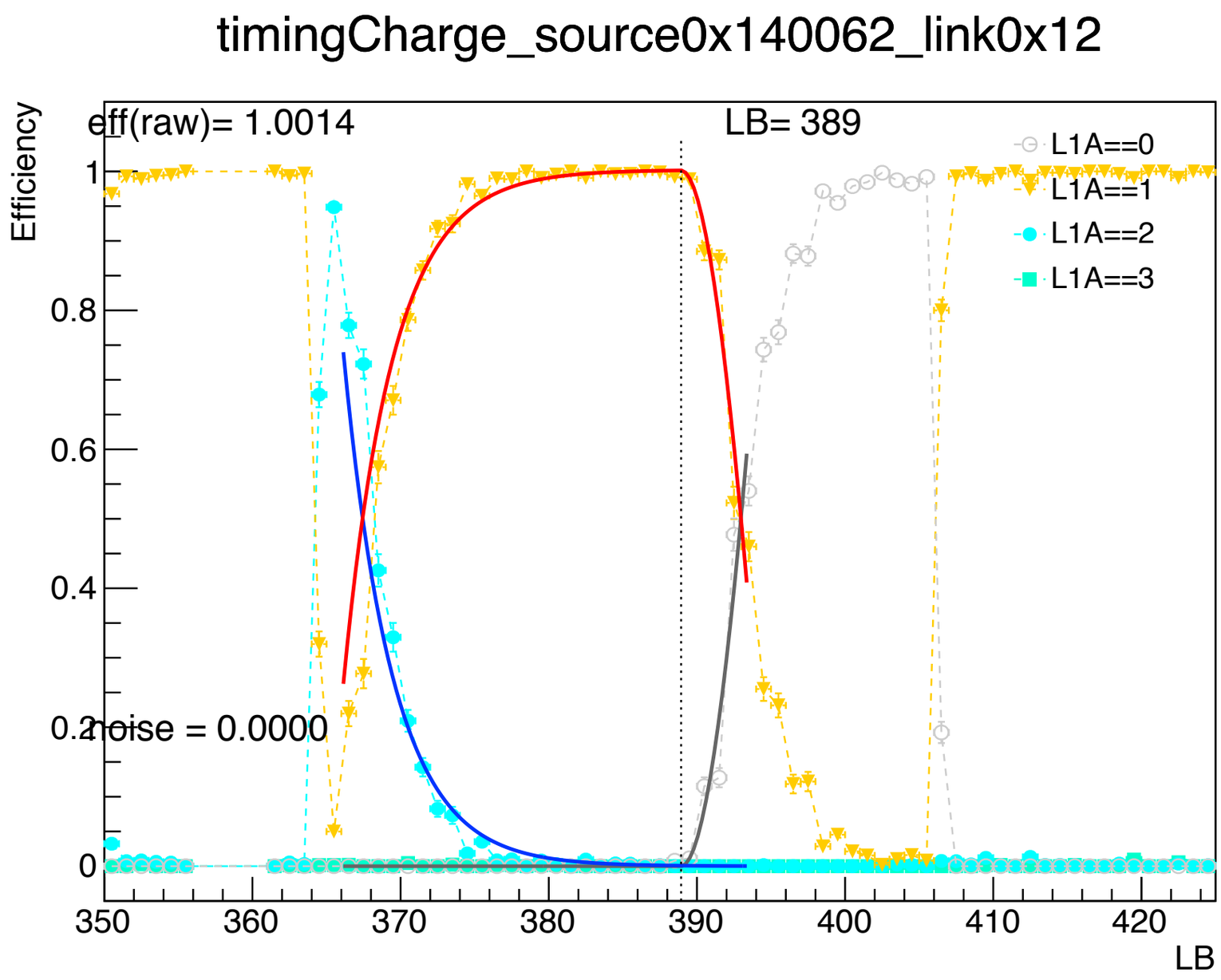
- https://svnweb.cern.ch/trac/atlasoff/browser/InnerDetector/InDetDetDescr/PixelCabling/tags/PixelCabling-00-00-51/share/Pixels_Atlas_IdMapping_M7.dat

What can this tool offer?

- Full information about the RAW data, but no high-level reconstruction
- Low-level validation of DAQ logic and reconstruction chain
- ROD/Module Error analyses
- Hit-level occupancy counting, ToT measurement
- Timing information at pixel level
- Validating DQ monitoring plots
- No clustering, no tracking
- No connectivity (one can feed in the downstream analysis)
  - You can develop your own hit map with the help of connectivity table. https://svnweb.cern.ch/trac/atlasoff/browser/InnerDetector/InDetDetDescr/PixelCabling/tags/PixelCabling-00-00-51/share/Pixels_Atlas_IdMapping_M7.dat

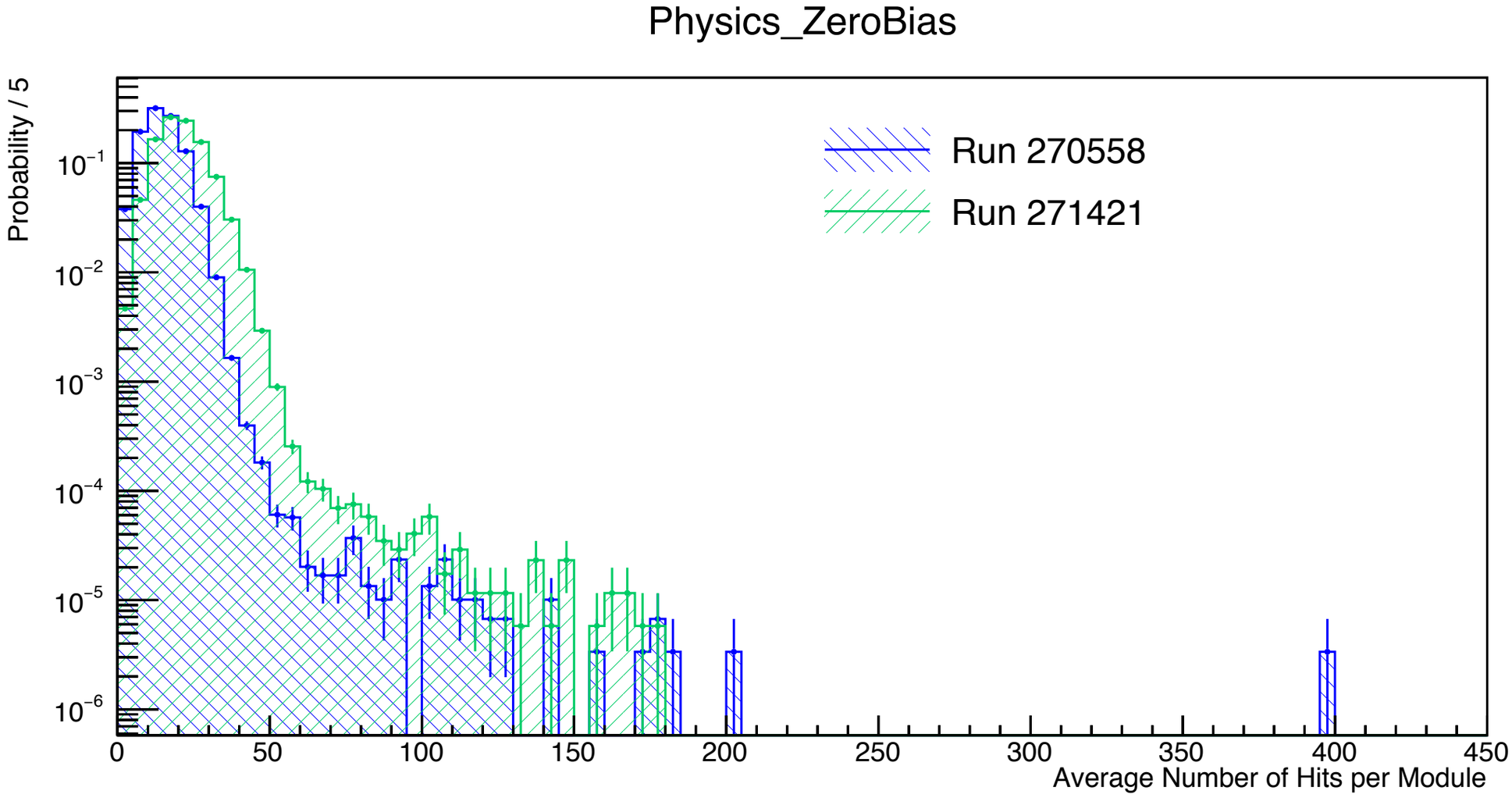
A complementary tool to xAOD analysis at low level

Example: Timing Scan Analysis

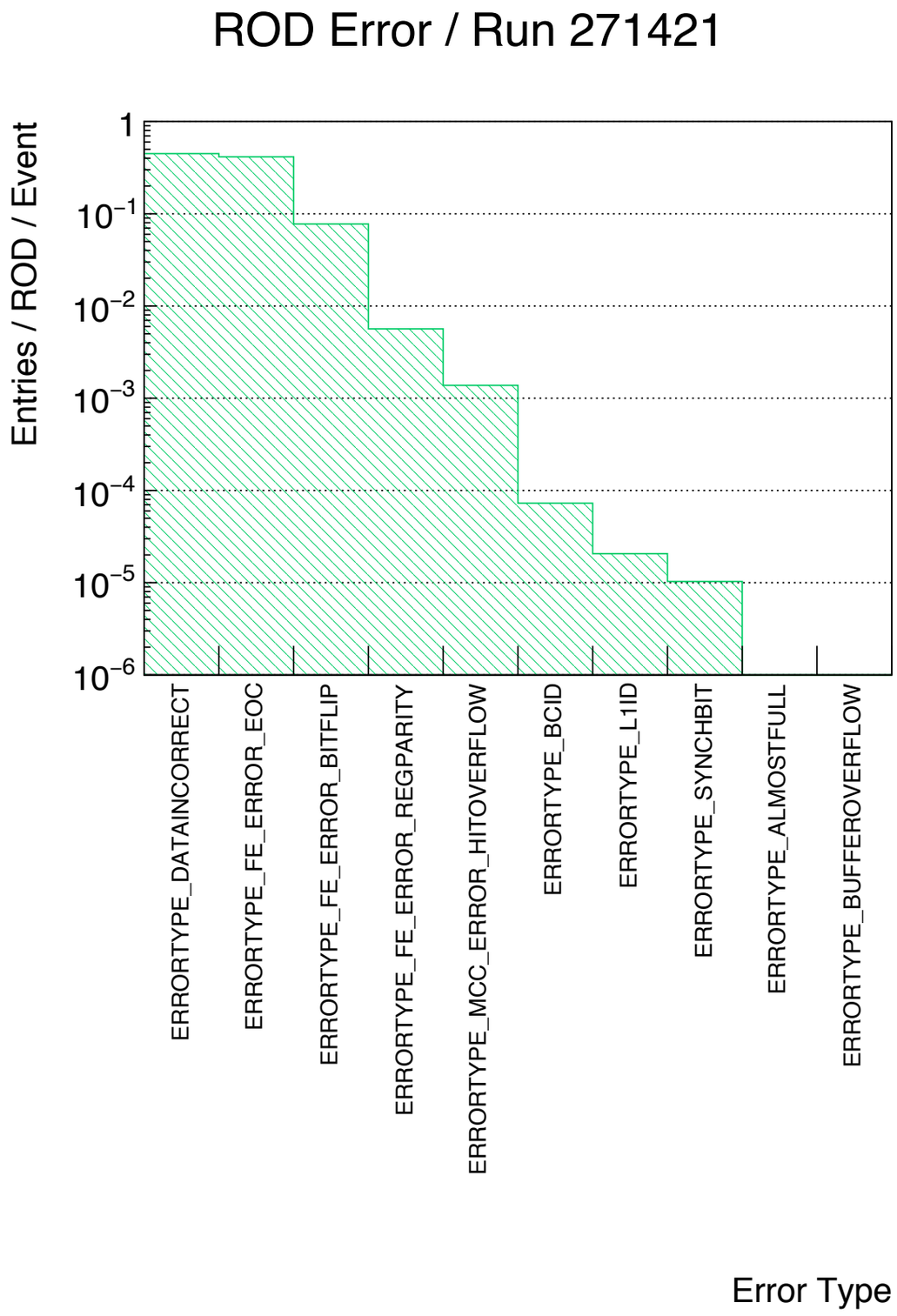


- If you know when it is appropriate to use, this is a simple and powerful tool!

Example Large Occupancy Analysis

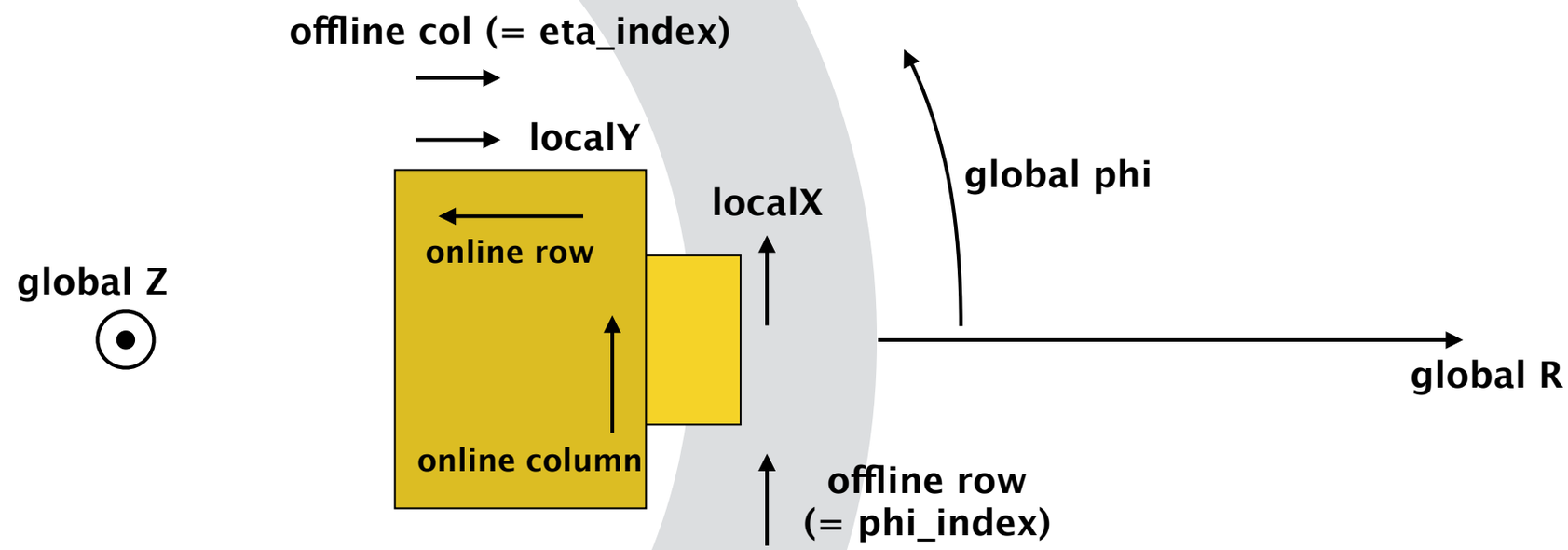


Example: Error Analysis



Example: DBM Connectivity Validation

localX, localY (DBM, A-side) : observation in M8 Data



Conflicting with the PixelCablingSvc!!
PixelCablingSvc is specifying the eta_index should have the same direction with online row.

View from A-side to IP

Setup

- Checkout the repository
  - `$ git clone https://${USER}@gitlab.cern.ch/atlas-pixel/RawDataAnalysis.git`
- Go to the directory and compile
  - `$ cd RawDataAnalysis/`
  - `$ source setup.sh`
  - `$ make`

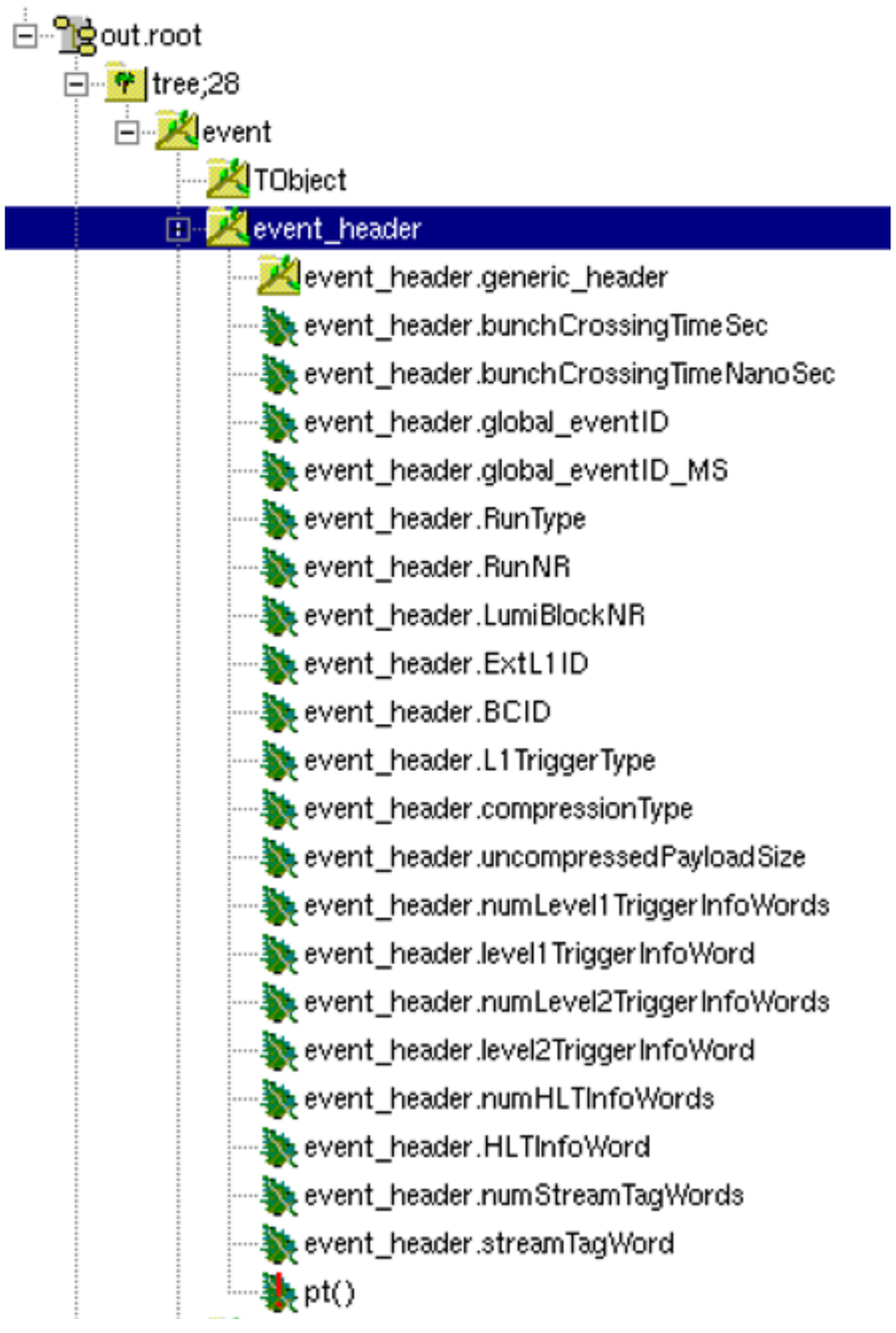
Decoding a sample (for Pixel)

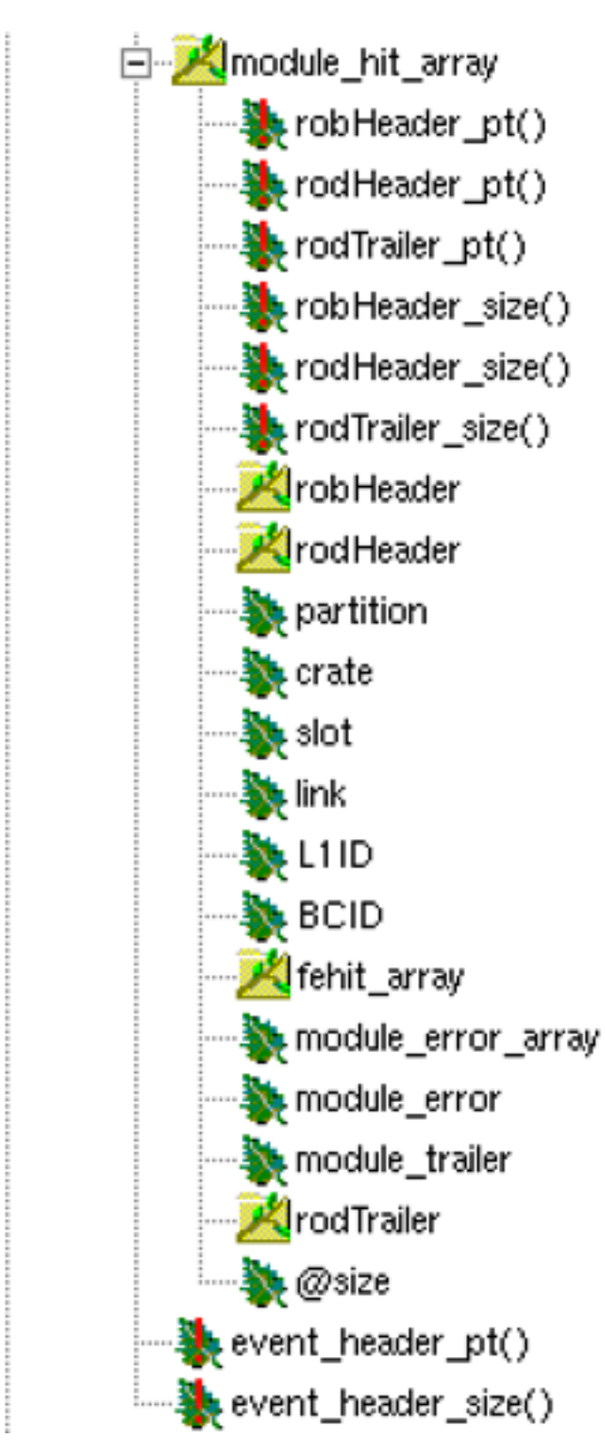
- Copy a data sample:
  - `$ eos cp /eos/atlas/user/h/hoide/pixel_online/data15_13TeV.00282455.express_express.merge.RAW._lb0400._SFO-ALL._0001.1 sample.RAW.dat`
- Skim the data sample for Pixel Detector
  - `$ skim_pixel sample.RAW.dat skimmed_sample.RAW.pixel.dat pixel`
- Decode the skimmed data
  - `$ mkdir skimmed`
  - `$ mv skimmed_sample.RAW.pixel.dat skimmed/`
  - `$ RawDataDecoder_Pixel skimmed pixel_decoded.root`

Decoding a sample (for IBL)

- Skim the data sample for IBL
  - `$ skim_pixel sample.RAW.dat skimmed_sample.RAW.ibl.dat ibl`
- Decode the skimmed data
  - `$ mkdir skimmed`
  - `$ mv skimmed_sample.RAW.ibl.dat skimmed/`
  - `$ RawDataDecoder_IBL skimmed ibl_decoded.root`

Looking into Data





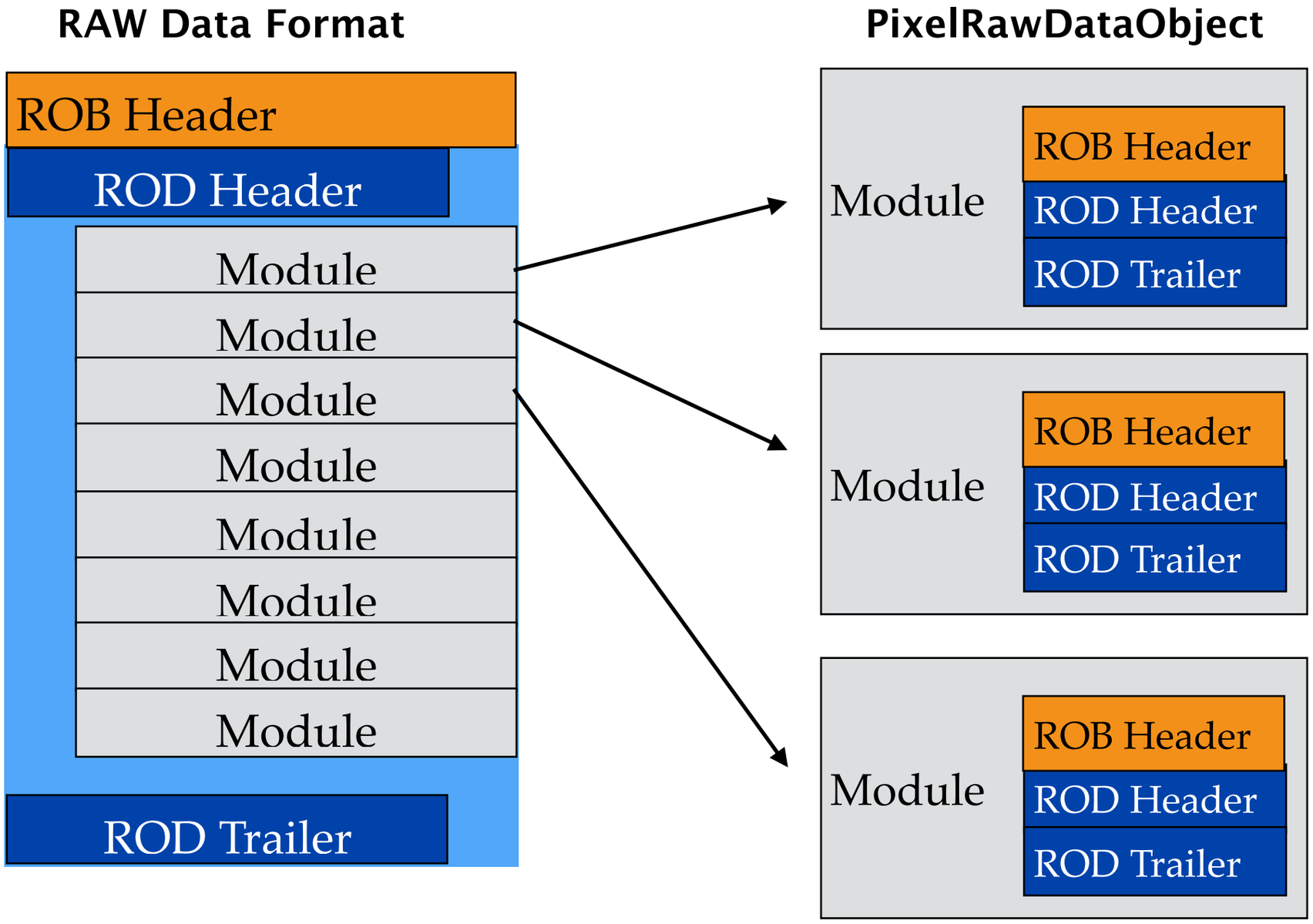
- An event comprises two parts: event_header and module_hit_array

PixelRawDataObject

```
class PixelRawDataObject : public TObject {

public:

    PixelRawDataObject();
    PixelRawDataObject(const PixelRawDataObject &T);
    virtual ~PixelRawDataObject();
    void clear();
    PixelRawDataObject& operator=( PixelRawDataObject &T );

    char* event_header_pt()           const { return (char*)(&(event_header.fragment_size)); }
    const size_t event_header_size() const { return sizeof( event_header ) - sizeof( unsigned int ) ; }

    void print(const int& level=0);

    full_event_header          event_header;
    std::vector<module_hit*>  *module_hit_array;

    ClassDef( PixelRawDataObject, 1 );
};
```

members

PixelRawDataObject



- Instead of nesting inside ROD, the ROB/ROD headers/trailers are duplicated for each module hit array in PixelRawDataObject.
- Shallower nest structure (fast access to actual hits in the loop), at a cost of duplication of data.
- Caution! Do not double count ROB/ROD headers!!

IBLRawDataObject

```
class IBLRawDataObject : public TObject {

public:

    IBLRawDataObject();
    IBLRawDataObject(const IBL::IBLRawDataObject &T);
    virtual ~IBLRawDataObject();
    void clear();
    IBLRawDataObject& operator=( IBL::IBLRawDataObject &T );

    char* event_header_pt()          const { return (char*)(&(event_header.fragment_size)); }
    size_t event_header_size() { return 21*sizeof(unsigned int) ; }

    void print(const int& level=0);

    full_event_header          event_header;
    std::vector<IBL::module_hit*>  *module_hit_array;

    ClassDef( IBL::IBLRawDataObject, 1 );
};
```

members

- Almost the same structure, but having IBL::module_hit array instead of module_hit.

Analysis

- The easiest way of writing analysis macro is just to use TTree::MakeClass() and write Loop() function.
- Need to use libRawDataAnalysis and class header files.
- Typically create a dedicated macro for the purpose to study which produces an output histogram file. Submit a lxbatch job per one or several RAW data, and copy the output histogram files to local directory. Do hadd if needed.
- Examples can be found in macros/ directory.
- Macros in macros/ directory will be compiled (ACLiC) automatically.

Example of batch job file

```
#!/bin/sh

#------------------------------------------------------------
# Preparations
mode=$1
runnum=$2
lumiblock=$3
stream=$4

# You have to rename the path of the package directory
package_dir=$HOME/workdir/pixel_online/RawDataAnalysis

# output name can be arbitrary
outdir=$HOME/workdir/pixel_online/occerr/output_ibl/${stream}_${runnum}
macro=batch_occerr_ibl.C

source ~/.bashrc
export LD_LIBRARY_PATH=${package_dir}/lib:$LD_LIBRARY_PATH

#------------------------------------------------------------
# Download the new data from EOS and skim the data for extracting IBL ROBs.
${package_dir}/batch/download_skim.sh ${mode} ${runnum} ${lumiblock} ${stream} ibl
rm -rf work
mkdir work
mv *.ibl work

#------------------------------------------------------------
# Convert the skimmed RAW data to IBLRawDataObject Tree file.
${package_dir}/bin/RawDataDecoder_IBL work out.root

#------------------------------------------------------------
# Setup the analysis macro.
cp ${package_dir}/batch/${root_macro} .
cp ${package_dir}/batch/rootlogon.C .
ls -ltrh

#------------------------------------------------------------
# Execute the ROOT macro
root -l -b -q ${root_macro}
ls -ltrh

#------------------------------------------------------------
# Copy the final output to the local area
mkdir -p ${outdir}
LB_FORM=`printf %04d ${lumiblock}`
cp out_occerr.root ${outdir}/${stream}_${runnum}_LB${LB_FORM}.occerr.root
```

Example of job submission script

```
#!/bin/sh

source ~/.bashrc
setupATLAS

mode=$1
runnum=$2
stream=$3
runnum_format=`printf %08d ${runnum}`


EOS_DIR=/eos/atlas/atlastier0/rucio/${mode}/${stream}/${runnum_format}/${mode}.${runnum_format}.${stream}.merge.RAW


# Scan the EOS directory and extract the list of lumi block numbers to submit
eos ls ${EOS_DIR} | cut -d_ -f4-4 | cut -d. -f1-1 | sed -e "s/lb//g" | uniq | while read LB
do
    # Remove zeros from the string
    LB=`expr ${LB} + 0`
    bsub -J test -q 1nh -L "/bin/bash" bsub_occerr_ibl.sh ${mode} ${runnum} ${LB} ${stream}
done
```