SLAC 2016 data analysis: Documentation and explanations

Kim Siang Khaw
University of Washington

June 24, 2016

1 Introduction

Prerequisites for this tutorial are a basic understanding of what the Muon g-2 experiment and PbF_2 calorimeters [1] are, some knowledge about the electromagnetic (EM) shower and the ROOT data analysis framework [2]. You can follow the exercise sheet step by step which guides you to the data analysis using the reconstructed electron EM shower clusters. Some analyses may require the use of reconstructed crystal hit, which is the basic object of forming a cluster. Advanced users are encouraged to use the FNAL's *art* framework for the data analysis.

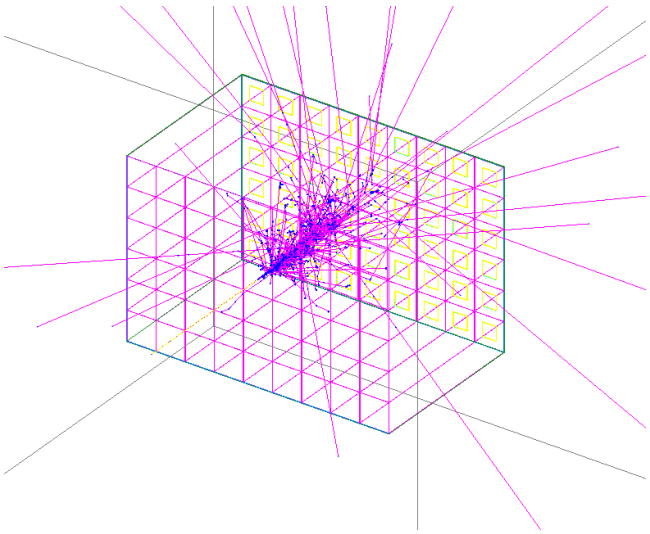

Figure 1: *Illustration of the electromagnetic shower of an electron injected from the left to the right, in a* 9×6 *array* PbF_2 *calorimeter. Cherenkov lights created by the charged shower particles are collected by the Silicon Photomultipliers (SiPMs) glued to the end of the* PbF_2 *crystals.*

Data analysis of this test beam has several components that are common with the Muon g-2 experiment data analysis framework [3]. The physics objects that will be used for most of the analyses are the crystal hits and the hit clusters. These objects are reconstructed from the digitized waveform, after going through steps like pulsing fitting, energy calibration, gain correction, time correction and hit clustering. During the first week of the test beam, all these reconstruction steps

will be refined based on the data taken and these will be taken care of by those who are familiar with the *art* framework. The main focus of this documentation is on the physics analysis using high level physics objects like the crystal hit and the cluster, using a C++/ROOT standalone framework. The user can proceed to more sophisticated analysis once he/she is familiar with the tools. The standalone framework can also be used as a stepping stone to develop analysis algorithm before migrating it to the *art* framework.

Several studies that will be covered for the data analysis (non-exhaustive list) are

- energy resolution of the calorimeter
- position and angular resolution of the calorimeter
- degeneracy of the position and angular information of the calorimeter
- stability of gain monitoring system
- pile up separation for multi-electron events

Further information about how the data will be processed and analyzed are summarized in Appendix A (to be updated).

2 Data samples

This section explains the naming and the locations of the data files (Local location will be obsolete once we move everything to the PNFS disk at FNAL).

2.1 File naming

All the created art/ROOT and ROOT files will have the same run number as the MIDAS they are extracted from.

Table 1: *Data samples and descriptions. The ROOT full data includes raw waveforms, chopped islands and processed data like fit results and crystal hits. The ROOT analysis data has only processed data.*

filename	type
gm2slac_run0xxxx.art	data ready for art-based analysis
gm2slac_run0xxxx_raw.root	full data for standalone C++ analysis
gm2slac_run0xxxx.root	analysis data for standalone C++ analysis

2.2 Location of the files

For the following steps, you have to be connected to the local Wifi network `T536-local`. All the art/ROOT and ROOT files are stored in the 2×3TB HDD in the `g2analysis` machine. Use the following command to copy over the data files that you want to analyze:

```
scp -r g2muon@g2analysis:/data1/slac2016/analysis/gm2slac_run0xxxx.root .
scp -r g2muon@g2analysis:/data2/slac2016/art/gm2slac_run0xxxx.art .
```

Or you can use `rsync` if you have enough space in your hard drive:

```
rsync -avurt g2muon@g2analysis:/data1/slac2016/analysis/ YOUR_LOCAL_DISK/
rsync -avurt g2muon@g2analysis:/data2/slac2016/art/ YOUR_LOCAL_DISK/
```

Or a better way is to use `sshfs` to mount the disk onto your laptop:

```
sshfs g2muon@g2analysis:/data1/slac2016/analysis YOUR_LOCAL_DISK
sshfs g2muon@g2analysis:/data2/slac2016/art YOUR_LOCAL_DISK
```

3 Data format and structure for the ROOT tree

The data samples for this tutorial are stored in ROOT trees. The tree contains a collection of variables (called branches) which are filled once per event (could be 1 or more electrons). The list of variables along with their data type and further explanations are given in the following.

For studies using higher level objects

- `FitResult_EventNum` (`vector<int>`): event number of this fit result
- `FitResult_CaloNum` (`vector<int>`): calorimeter number of this fit result
- `FitResult_XtalNum` (`vector<int>`): crystal number of this fit result
- `FitResult_IslandNum` (`vector<int>`): island number of this fit result
- `FitResult_UtcaSlotNum` (`vector<int>`): utca slot number of this fit result
- `FitResult_ChannelNum` (`vector<int>`): rider channel number of this fit result
- `FitResult_Energy` (`vector<double>`): energy (number of photons) of this fit result
- `FitResult_Time` (`vector<double>`): time (clock tick of 800 MHz) of this fit result within the fill/event
- `FitResult_Pedestal` (`vector<double>`): pedestal of this fit result
- `FitResult_Chi2` (`vector<double>`): Chi squared of this fit result
- `FitResult_ClockCounter` (`vector<long long>`): time stamp (clock tick of 40 MHz) of this fit result from Rider header information
- `XtalHit_EventNum` (`vector<int>`): event number of this crystal hit
- `XtalHit_CaloNum` (`vector<int>`): calorimeter number of this crystal hit
- `XtalHit_XtalNum` (`vector<int>`): crystal number of this crystal hit
- `XtalHit_IslandNum` (`vector<int>`): island number of this crystal hit
- `XtalHit_UtcaSlotNum` (`vector<int>`): utca slot number of this crystal hit
- `XtalHit_ChannelNum` (`vector<int>`): rider channel number of this crystal hit

- `XtalHit_Energy` (`vector<double>`): energy (number of photons) of this crystal hit
- `XtalHit_Time` (`vector<double>`): time (clock tick of 800 MHz) of this crystal hit within the fill/event
- `XtalHit_ClockCounter` (`vector<long long>`): time stamp (clock tick of 40 MHz) of this crystal hit from Rider header information
- `Cluster_EventNum` (`vector<int>`): event number of this cluster
- `Cluster_CaloNum` (`vector<int>`): calorimeter number of this cluster
- `Cluster_IslandNum` (`vector<int>`): island number of this cluster
- `Cluster_Energy` (`vector<double>`): energy (number of photons) of this cluster
- `Cluster_Time` (`vector<double>`): time (clock tick) of this cluster
- `Cluster_X` (`vector<double>`): local x-position of this cluster (logarithmic-weighted)
- `Cluster_Y` (`vector<double>`): local x-position of this cluster (logarithmic-weighted)
- `Italiano_EventNum` (`vector<int>`): event number of this analyzed laser crate waveform
- `Italiano_CaloNum` (`vector<int>`): calorimeter number of this analyzed laser crate waveform
- `Italiano_XtalNum` (`vector<int>`): crystal number of this analyzed laser crate waveform
- `Italiano_IslandNum` (`vector<int>`): island number of this analyzed laser crate waveform
- `Italiano_UtcaSlotNum` (`vector<int>`): utca slot number of this analyzed laser crate waveform
- `Italiano_ChannelNum` (`vector<int>`): rider channel number of this analyzed laser crate waveform
- `Italiano_Amplitude` (`vector<double>`): amplitude (ADC samples) of this analyzed laser crate waveform
- `Italiano_Time` (`vector<double>`): time (clock tick of 800 MHz) of this analyzed laser crate waveform within the fill/event
- `Italiano_Pedestal` (`vector<double>`): pedestal of this analyzed laser crate waveform
- `Italiano_Area` (`vector<double>`): Area of this analyzed laser crate waveform
- `Italiano_ClockCounter` (`vector<long long>`): time stamp (clock tick of 40 MHz) of this analyzed laser crate waveform from Rider header information

For studies using lower level objects

Blow are the Tleaves of the `islandTree`.

- `RunNum` (`int`): run number of this island
- `EventNum` (`int`): event number of this island
- `FillType` (`int`): fill type of this island (1 is muon fill, 2 is laser fill)
- `TriggerNum` (`int>`): trigger number of this island
- `CaloNum` (`int`): calorimeter number of this island
- `XtalNum` (`int`): crystal number of this island
- `IslandNum` (`int`): island number of this island
- `UtcaSlotNum` (`int`): utca slot number of this island
- `ChannelNum` (`int`): rider channel number of this island
- `Length` (`int`): number of sample of this island
- `Time` (`int`): time (clock tick of 800 MHz) of the first sample of this island within the fill/event
- `ClockCounter` (`long`): time stamp (clock tick of 40 MHz) of this island from Rider header information
- `Trace` (`vector<short>`): ADC samples of this island

4 Standalone C++ Analysis framework

The package `SLAC2016Ana` contains an example `C++` framework to help you getting started. You can get it from the github.com by `git clone https://github.com/kimsiang/SLAC2016`. You can build your additional analysis code on top of this example or write a new one based on it. The example is already running but it's not doing much yet. You can compile the analysis code by first sourcing your ROOT environment (e.g. `source /usr/local/bin/thisroot.sh`) and then followed by executing the command `make`.

This will read the necessary ingredients for compilation from the `Makefile` in the same directory. Don't have to worry much about this file at the moment unless you want to add in more classes to the analysis code. The point is that it creates an executable named `ana`. You can then execute the program by the command `./ana input.script` where `input.script` includes a path to the root file that you want to analyze (e.g. `./test.root`).

A description of the individual components of the example are given in the following list. Indicated are also the places where you should start adding your own code:

- `main.cxx`: This is the first starting point. It contains the `main()` function which is necessary for any `C++` program. The first step is to create instances of `MyAna()` class which is implemented in the files `MyAna.h` and `MyAna.C` (explained in the next items). The `TChain` represents the ROOT tree discussed in section 3. The files which should be read from disk

are specified in the function `Add(filename)`. The tree is then read and processed by the `MyAna()` class which takes the `TChain` as argument. The real work is then done in the `Loop()` function of the `MyAna()` class which is discussed in the next two items.

- `MyAna.h`: Definition of the class `MyAna`, which inherits from the `TTree::MakeClass`. It declares variables and ROOT objects that will be used or stored in your analysis. Several basic functions that are common among event-based particle physics analysis like `initialize()`, `clear()`, `execute()` and`finalize()` are declared here.

- `MyAna.C`: The main function which is called automatically which processing the ROOT trees are `Loop()`. The `Loop()` function is called only once per run. In the `Loop()` function, `initialize()` is called at the beginning of the analysis run, `clear()` and `execute()` is called every event, and `finalize()` at the end of the analysis run.

- `t1.h`: Header file for the class `t1` created using `TTree::MakeClass`.

- `t1.C`: Source file for the class `t1` created using `TTree::MakeClass`. The class `Loop()` is used by `MyAna` to loop through each `TBranch`.

- `PlotAll.C`: A ROOT macro which can be used for automatic plotting of a set of histograms which are stored in a ROOT file. Please read the header of the file on how to use it.

5 ROOT-based offline event display

A ROOT-based offline event display is also developed to increase the user friendliness of the data analysis. As shown in Figure 2, this GUI interface allows the user to inspect the fit results of the template fit algorithm by overlaying it with the island samples. More functionality will be added in the coming days.

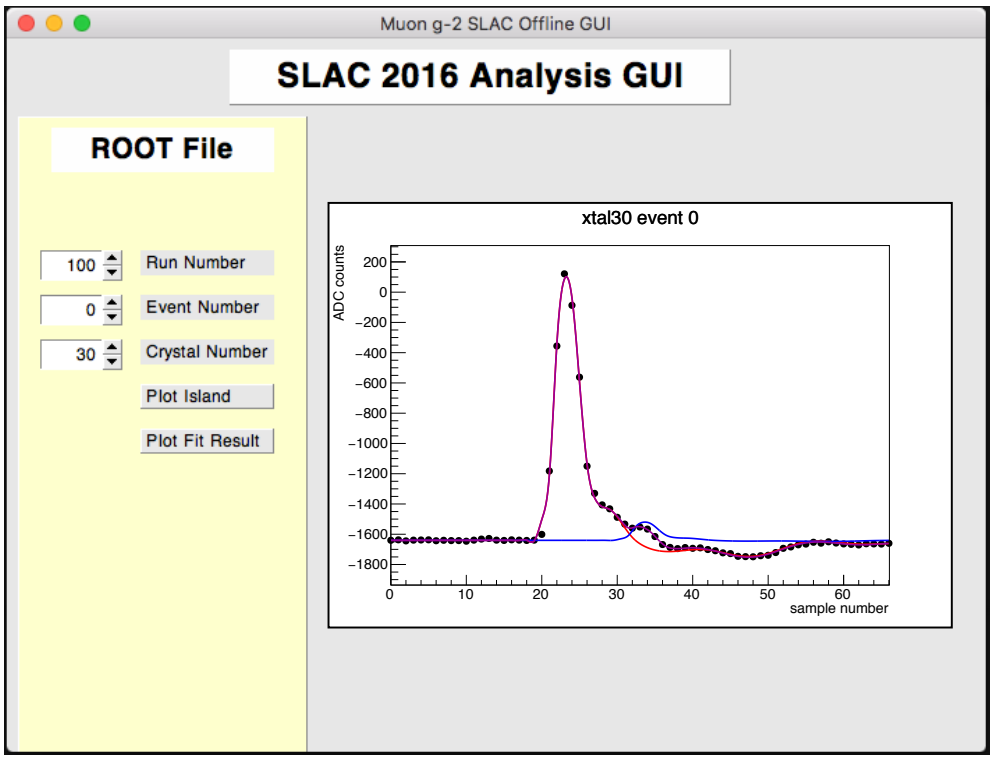

Figure 2: *SLAC Offline analysis GUI. This is a very preminary prototype.*

A Data analysis framework

The data analysis of this test beam has several components. First we need to convert the raw data stored in a MIDAS file (`.mid` or `.mid.gz`) to *art* data products stored in a ROOT file. This is handled using *art* framework's modules and is doing nothing more than storing `16-bit` or `32-bit` word into `vectors`. Next we unpack these `vectors` and give them contexts based on the header information stored within the `vectors`. As this step, all the information are stored as data products you are probably familiar with: `RiderArtRecord`, `IslandArtRecord`, etc.

B Header information

This section provides the information regarding user interface to the event, AMC13 and rider header information. All the information are stored in the art/ROOT files and standalone ROOT files with the TBranch structures in the following sections.

C Header and trailer formats

This section compiles all the available header data formats. Please refer to http://gm2-docdb.fnal.gov:8080/cgi-bin/ShowDocument?docid=3409 for more details.

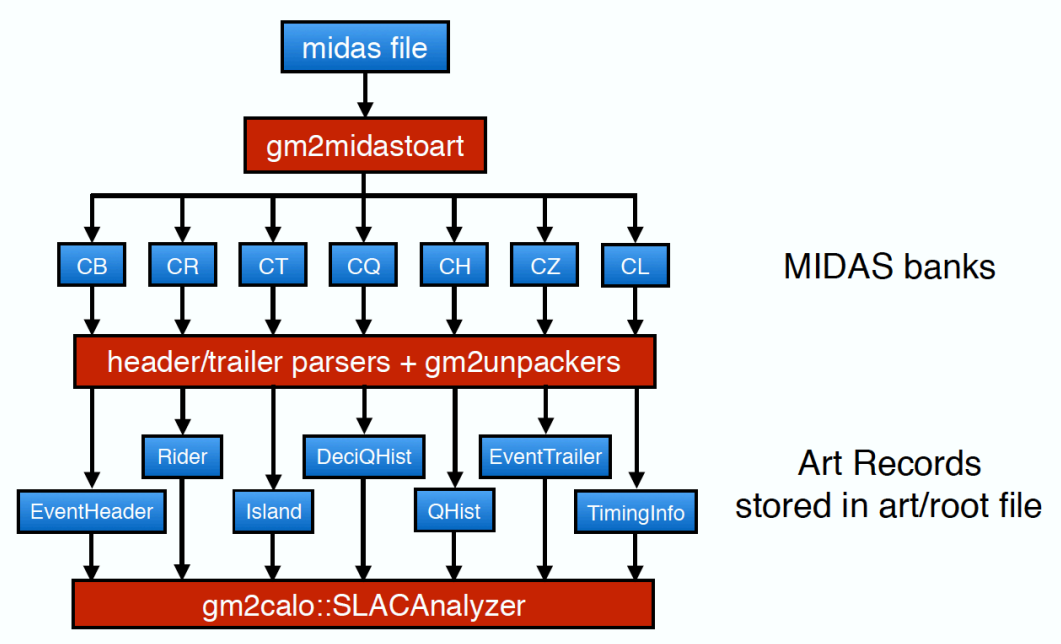

Figure 3: *Offline framework for SLAC experimental data.*

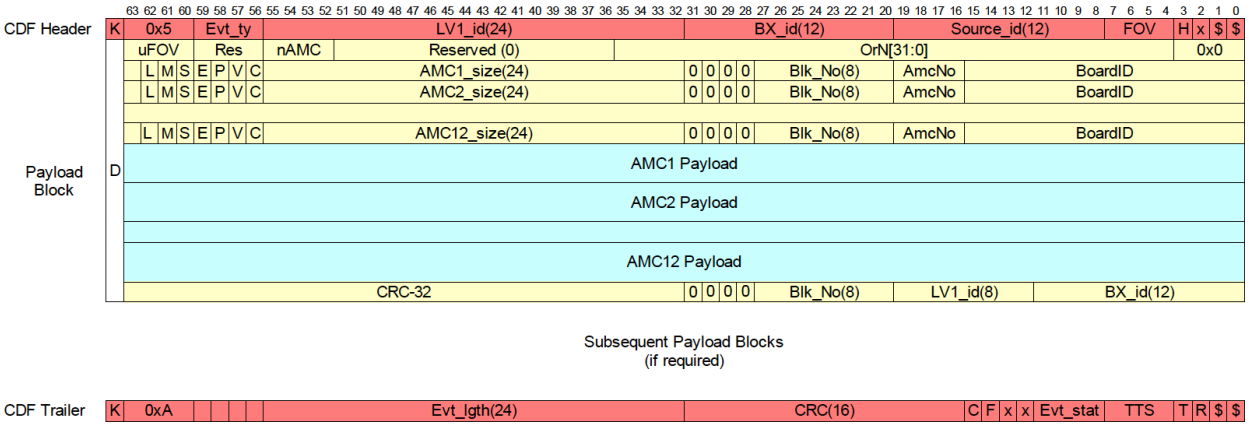

Figure 4: *AMC13 to DAQ data format.*

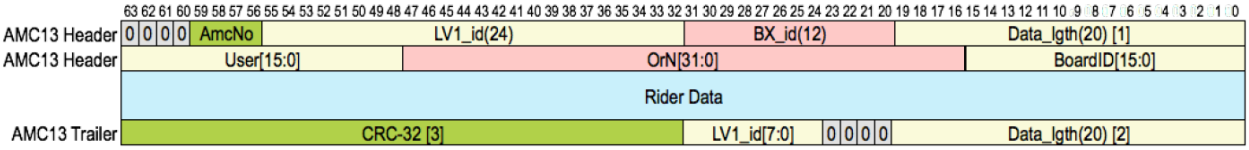

Figure 5: *Rider to AMC13 data format.*

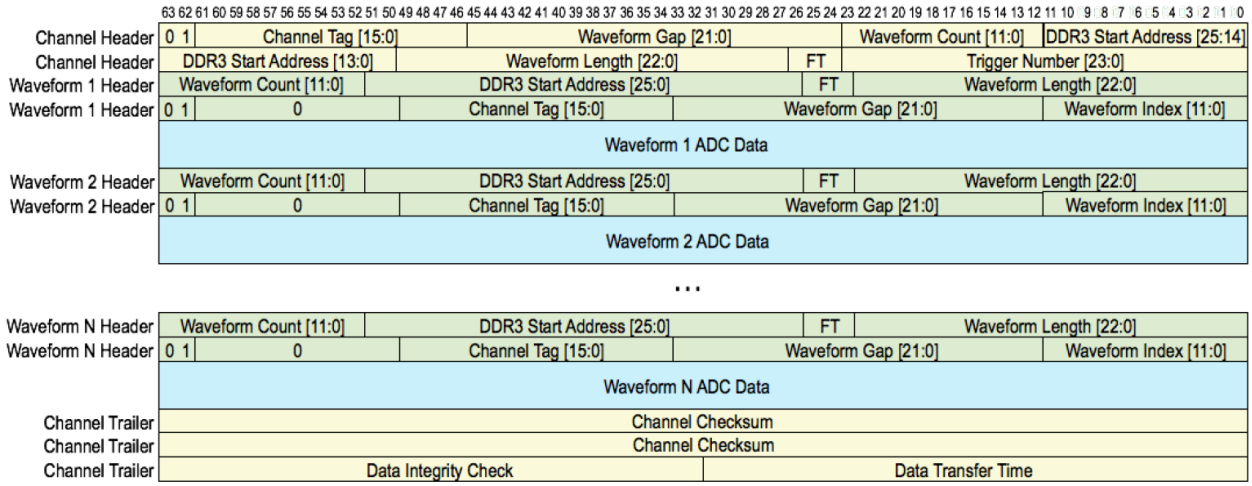

Figure 6: *Rider Channel data format.*

References

[1] A. Fienberg et. al., *Studies of an array of PbF_2 Cherenkvo crystals with large-area SiPM readout*, Nuclear Instruments and Methods in Physics Research Section A, Vol. 783, 12-21 (2015)

[2] Rene Brun and Fons Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Proceedings ALHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch.

[3] K. S. Khaw, *Offline structure group report*, FNAL E989 g-2 Experiment Document, FM2-doc-3781-v3 (2016). http://gm2-docdb-fnal.gov:8080/cgi-bin/ShowDocument?docid=3781

[4] TRIUMF MIDAS homepage, accessed May 13, 2016. https://midas.triumf.ca/MidasWiki/index.php/Main_Page

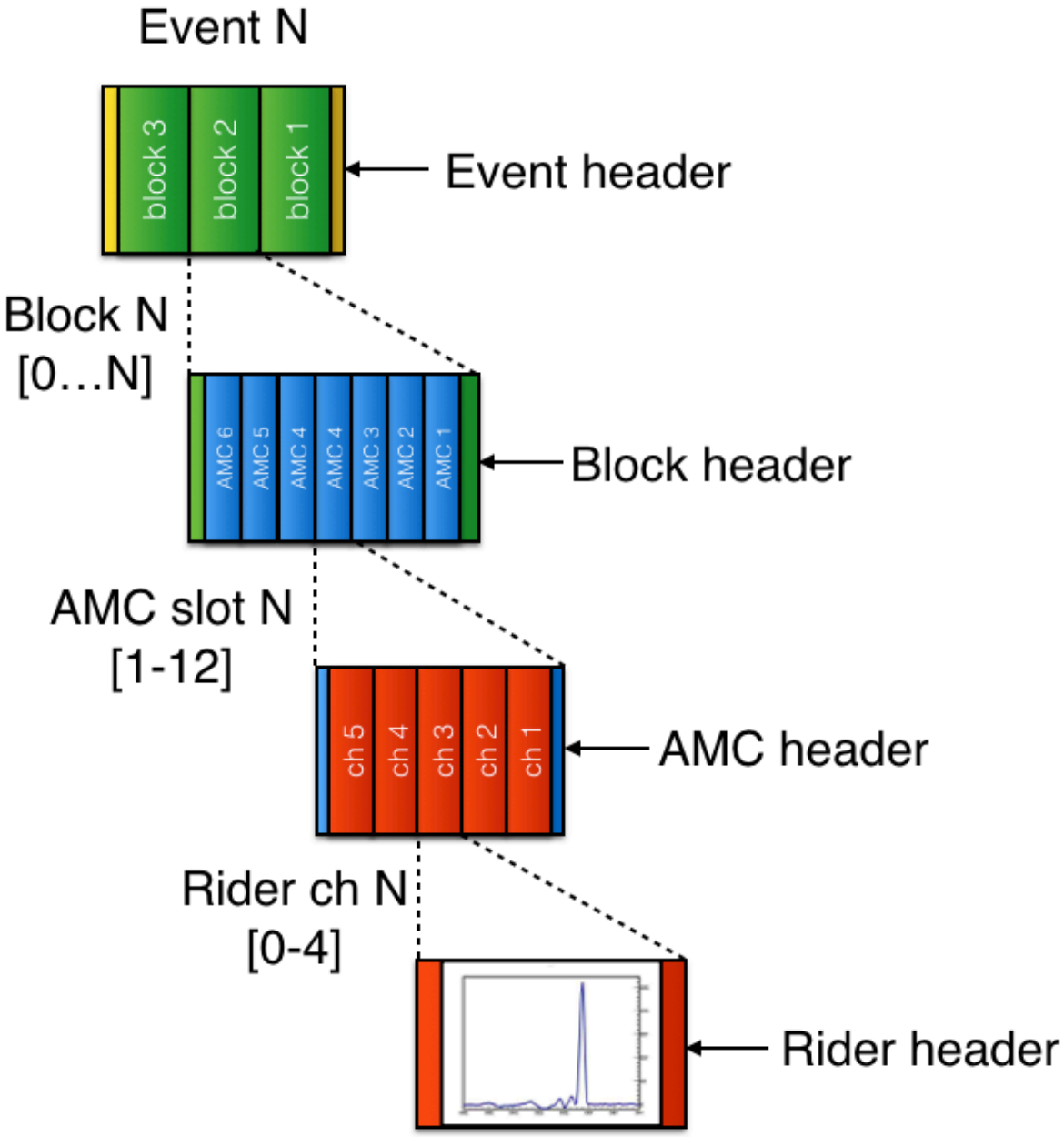

Figure 7: *Per event data format*