# SLAC 2016 data analysis: Documentation and explanations

Kim Siang Khaw
University of Washington

May 13, 2016

## 1 Introduction

Prerequisites for this tutorial are a basic understand 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 may use the gm2 *art* framework for the data analysis.

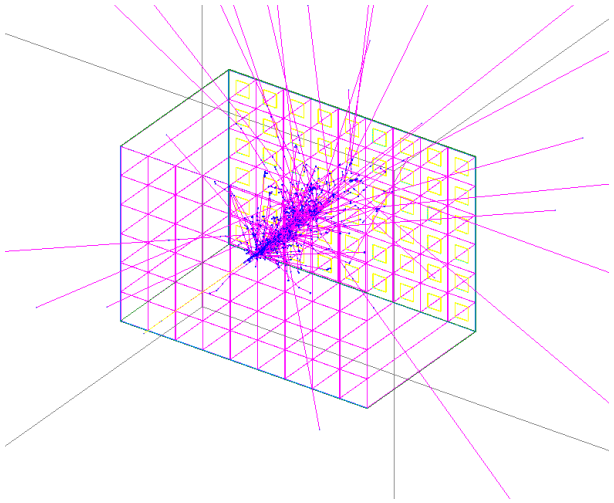

**Figure 1:** *Illustration of the electromagnetic shower of an electron injected from the left to the right, in a* $9 \times 6$ *array* $PbF_2$ *calorimeter. Cherenkov lights created by the charged shower particles are collected by the Silicon Photomultipliers (SiPMs) glued to the end of the* $PbF_2$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

Data analysis for 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 the steps like pulse fitting, energy calibration, gain 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 analysis using high level physics objects like the crystal hit and the hit cluster, using a C++/ROOT standalone framework. The user can proceed to more sophisticated analysis once he/she is familiar with the tools.

Several studies that will be done during/after the SLAC run (non-exhaustive list) are the following:

- 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 location of the data files.

## 2.1 File naming

The midas run number is preserved for each art/ROOT and ROOT files created from it.

**Table 1:** *Data samples and descriptions.*

| filename | type |
|---|---|
| run00xxx_unpacked.root | data ready for art-based analysis |
| run00xxx_analysis.root | data ready for standalone C++ analysis |

## 2.2 Location of the files

For the following steps, you have to be connected to the local Wifi network `g2SLAC`. (See Appendix B for the control room network map.) All the art/ROOT and ROOT files are stored in the $2 \times 3$TB HDD in the `g2analyzer` machine. Use the following command to copy over the data files that you want to analyze:

```
scp -r g2muon@g2analyzer:/data/root_analysis/run00xxx_analysis.root
```

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

```
rsync -avurt g2muon@g2analyzer:/data/root_analysis/ /your/folder/
```

# 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.

## Common parameters

- `RunNum` (int): run number of MIDAS DAQ system
- `EventNum` (int): event number with a run
- `EventType` (int): event type, 1 for muon fill, 2 for laser fill, 3 for pedestal

## For studies using higher level objects

- `NCluster` (int): number of clusters in the event.
- `Cluster_X` (double): local x-position of the cluster in the event.
- `Cluster_Y` (double): local y-position of the cluster in the event.
- `Cluster_Z` (double): local z-position of the cluster in the event.
- `Cluster_E` (double): energy (MeV) of the cluster in the event.
- `Cluster_T` (double): time (ns) of the cluster in the event.
- `NXtalHit` (int): number of crystal hits in the event.
- `XtalHit_X` (double): local x-position of the crystal hit in the event.
- `XtalHit_Y` (double): local y-position of the crystal hit in the event.
- `XtalHit_Z` (double): local z-position of the crystal hit in the event.
- `XtalHit_E` (double): energy (MeV) of the crystal hit in the event.
- `XtalHit_T` (double): time (ns) of the crystal hit in the event.
- `XtalHit_XtalNum` (int): crystal number in a calorimeter of the crystal hit in the event.

## For studies using lower level objects

- `Island_IslandNum` (int): island number of the island in the event.
- `Island_XtalNum` (int): crystal number of the island in the event.
- `Island_Time` (int): index of the first sample of the island in the event.
- `Island_Length` (int): length of the island in the event.
- `Island_Trace` (vector<short>): vector of the island ADC samples in the event.
- `FitResult_XtalNum` (int): crystal number of the fit result in the event.
- `FitResult_Time` (double): fitted time of the fit result in the event.
- `FitResult_Energy` (double): fitted energy (uncalibrated) of the fit result in the event.
- `FitResult_Chi2` (double): chi squared of the fit result in the event.

## For studies related to DAQ and digitizers

Please refer to Appendix C for more details about the header and trailer information.

**For each event:**

- `AMC13TrigNum` (long): trigger number of the AMC13 (From CDF and payload header)
- `AMC13ClockCounter` (long): clock counter [BX_id and OrN] of the AMC13 (From CDF and payload header)

**For each rider:**

- `AMCSlotNum` (int): AMC slot number (1-12) that rider resides
- `RiderTrigNum` (int): rider trigger number
- `RiderBoardID` (int): rider board ID
- `ClockCounter` (long): rider clock counter (should be synced with `AMC13ClockCounter`)

**For each channel:**

- `ChannelTag` (short): channel tag of the channel
- `WaveformGap` (int): waveform gap of the channel
- `WaveformCount` (short): waveform count of the channel
- `DDR3StartAddress` (int): DDR3 start address of the channel
- `WaveformLength` (int): waveform length of the channel
- `TriggerNum` (int): trigger number of the channel
- `FillType` (char): fill type of the channel

**For each waveform:**

- `ChannelTag` (short): channel tag of the waveform
- `WaveformGap` (int): waveform gap of the between waveforms
- `WaveformCount` (short): waveform count of the waveforms
- `DDR3StartAddress` (int): DDR3 start address of the waveform
- `WaveformLength` (int): waveform length
- `WaveformIndex` (int): waveform index
- `FillType` (char): fill type of the waveform

This list will be updated according to the needs of the on-the-spot data analysis.

# 4 Standalone C++ Analysis framework

The package `SLAC2016Ana.tar.gz` contains an example C++ framework to help you getting started. Unpack the example with `tar xvzf SLAC2016Ana.tar.gz` to your work folder. 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. The GUI is built using `TGuiBuilder` and it calls a nice class written by Aaron to display the results from the template fitter. More functionality will be added in the coming days.

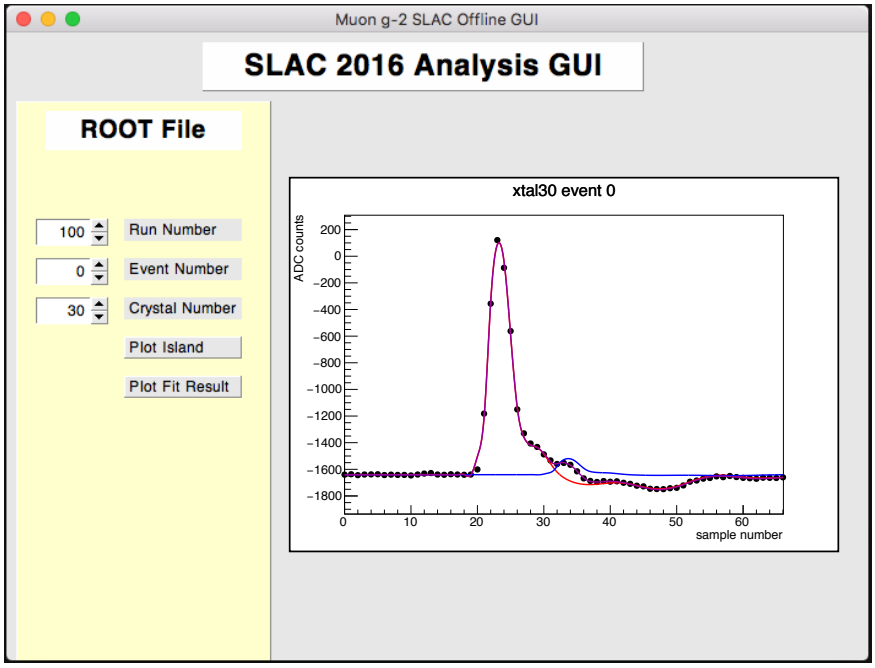


**Figure 2:** *SLAC Offline analysis GUI. This is a very preliminary prototype. The histogram/graph will be shown in a separate canvas so that the user can interact with it and can export the figure to any format like the TCanvas.*

# A Data analysis framework

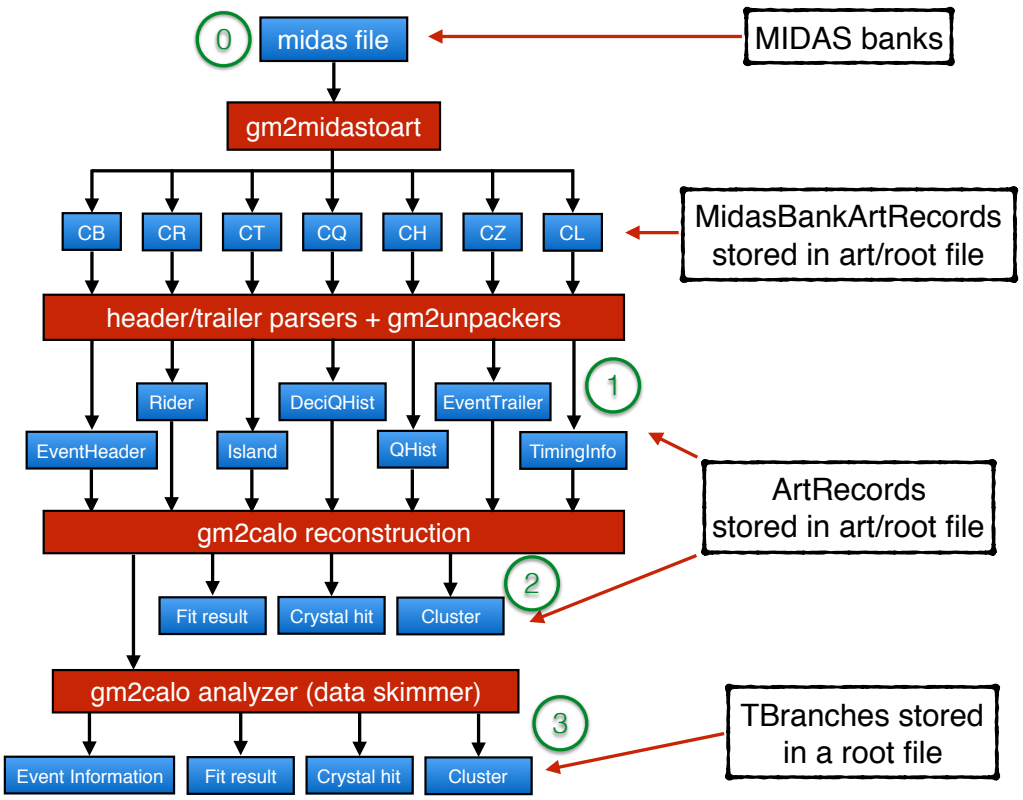


**Figure 3:** *Offline framework for SLAC experimental data. See text for the explanation of the flow.*

The data analysis of this test beam has several components and are depicted in the Figure 3. At step 0, we receive the raw MIDAS files from the Midas DAQ system [4]. First we need to convert the data banks (CB, CR, etc) 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`. Then we unpack these `vectors` and give them contexts based on the header information stored within the `vectors`. As this step 1, all the information are stored as data products you are probably familiar with: `RiderArtRecord`, `IslandArtRecord`, etc. At step number 2, we reconstruct the physics objects using `IslandArtRecord` and produced higher level objects such as `CrystalHitArtRecord` and `ClusterArtRecord`. For the standalone C++ analysis, an additional step (3) is taken to skim high level objects to a ROOT file.

# B SLAC network diagram for offline analysis

The local and external network connection are summarized in Figure 4. Note that you need to switch between the local-only and the internet connection if you want to download the data files from the g2analyzer machine.

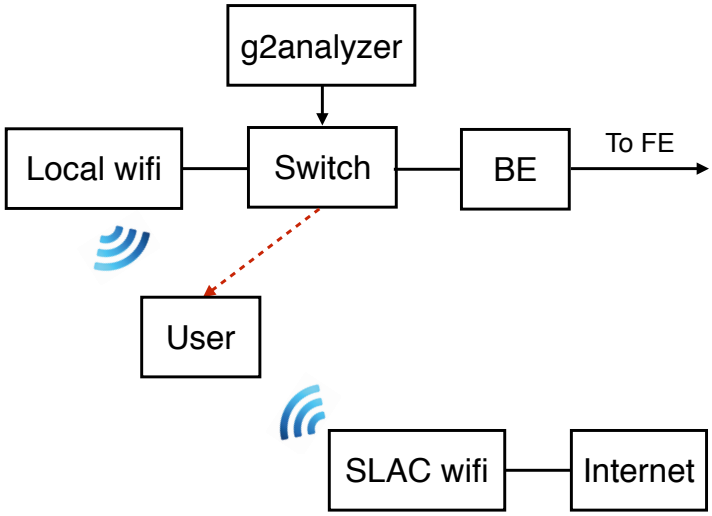


**Figure 4:** *Network connection for the SLAC control room. A user can download the analysis root files by connecting to the switch or through the local wifi network. Internet is accesible only through the SLAC wifi.*

# 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.

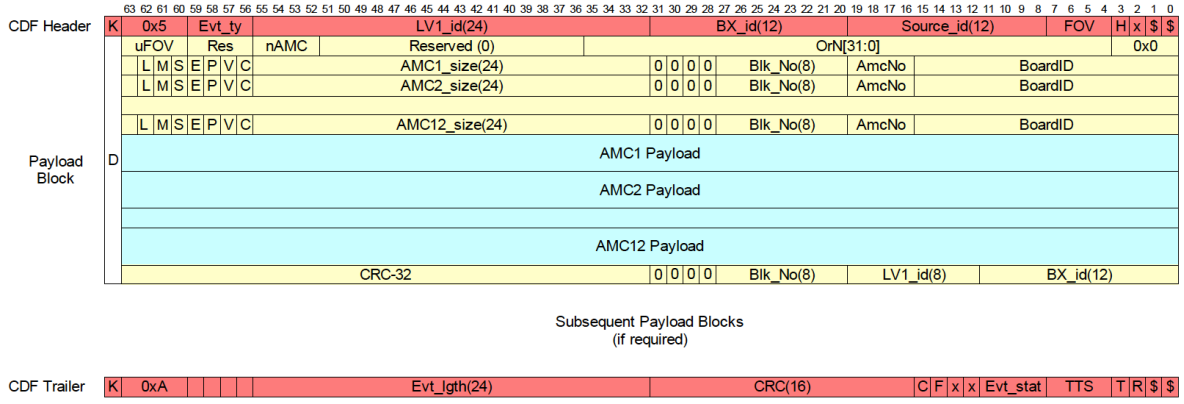


**Figure 5:** *AMC13 to DAQ data format.*

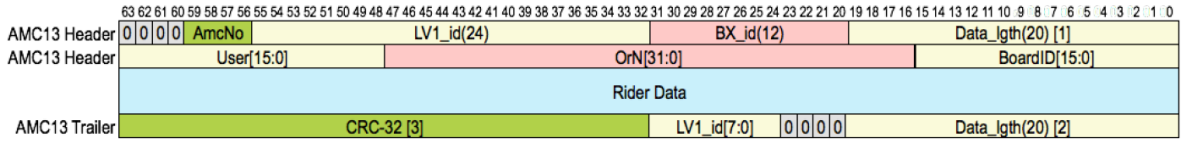


**Figure 6:** *Rider to AMC13 data format.*

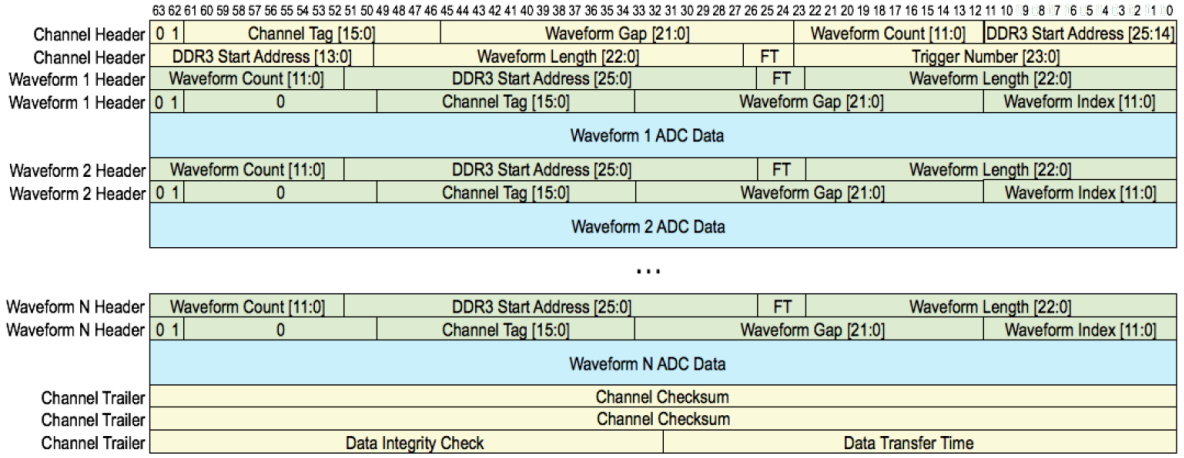


**Figure 7:** *Rider Channel data format.*

# References


[1] A.T. Fienberg et. al., *Studies of an array of $PbF_2$ Cherenkov 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 AIHENP'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, GM2-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

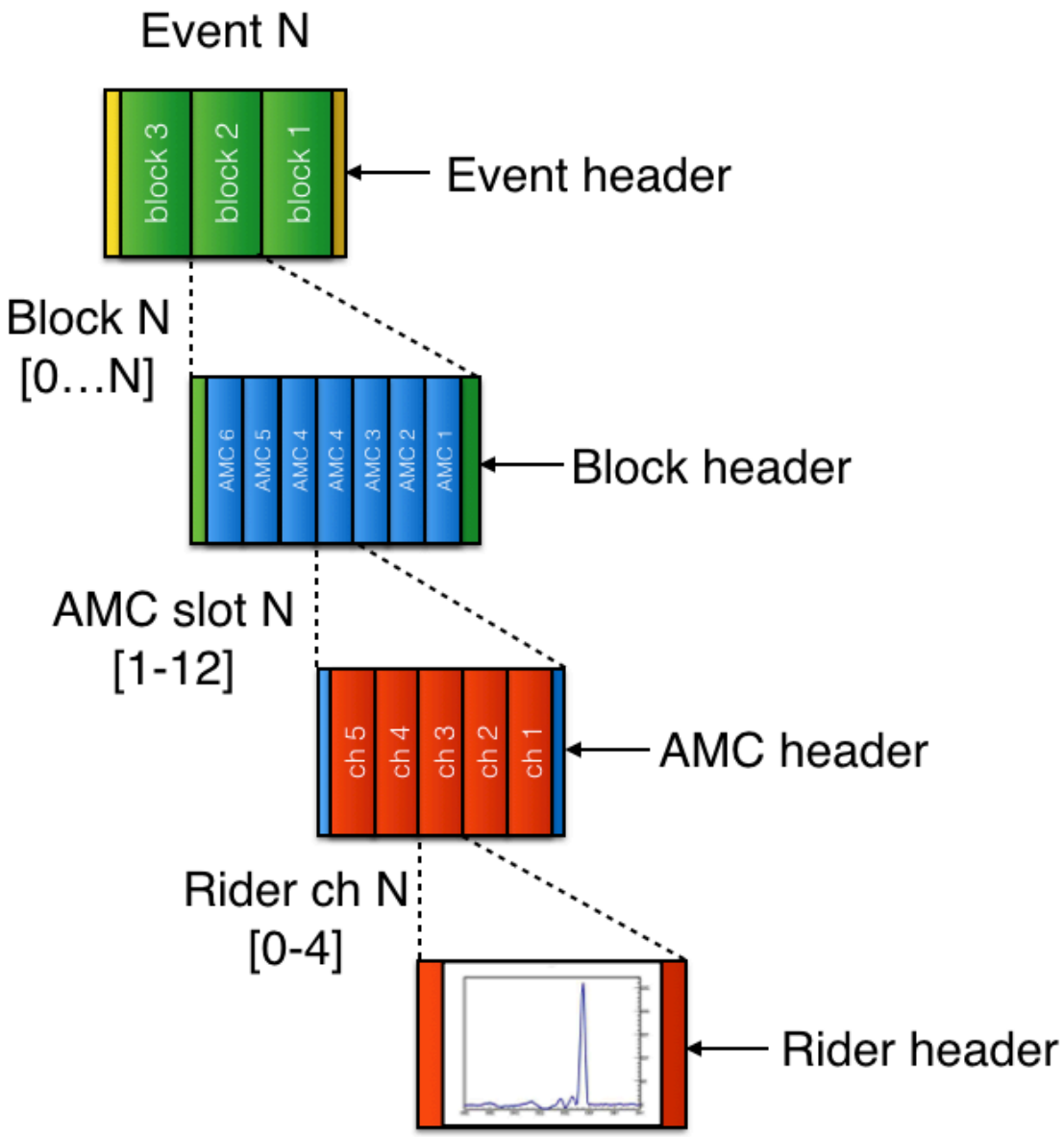


**Figure 8:** *Per event data format*