# JLAB Data Formats for DAQ

(and proposals for Streaming)

**Streaming Readout VII – Virtual Meeting**

**Nov 16-18, 2020**

**David Abbott - FEDAQ Group**

**Jefferson Lab – Physics Division**

Jefferson Lab

# Data Acquisition at JLab

- Historically the DAQ group (in Physics Division) has existed at Jefferson Lab from the very beginning
  - The CODA toolkit has been supported and expanded through 3 generations so far.
  - All 4 Experimental Halls are using some version of CODA for data acquisition.

- One aspect of the CODA toolkit is a data format standard we call EVIO (Event I/O)
  - It defines simple data structures at the Front-End for encapsulating raw data from digitizers/ASICs
  - Also defined is a file format block structure. This structure is also useful for blocking up data for transport over a network or shared memory between CODA software components.

- The CODA tool kit also has useful EVIO libraries and applications
  - C, C++ and JAVA APIs for reading/writing with EVIO Files
  - GUI utility (jeviodmp) for viewing and/or debugging EVIO Files as well as spying on active data streams.

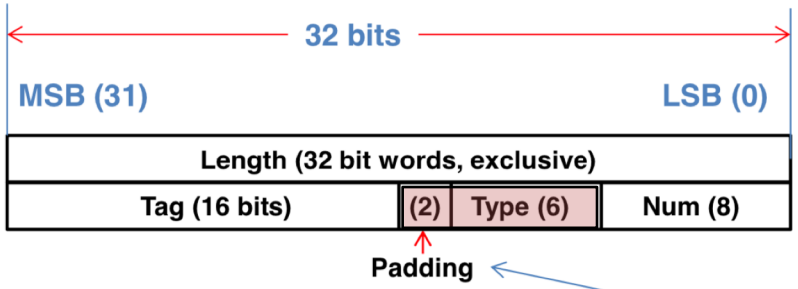
- We are currently looking to adapt EVIO to support Streaming Data architectures.

# EVIO Primitive Data Structures

- EVIO data formats are based on 32 bit words

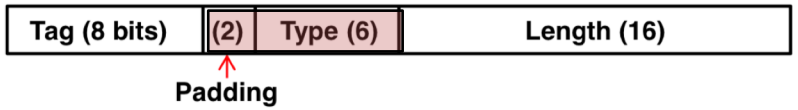
## Evio Header Formats

### Bank :

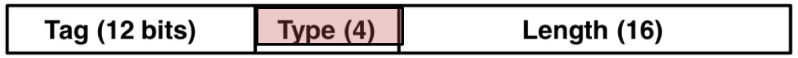



Number of unused bytes at end of following data if not a multiple of 32 bits.
For shorts, it is 0 or 2.
For chars (not strings), it is 0, 1, 2, or 3

### Segment :




### Tag Segment :



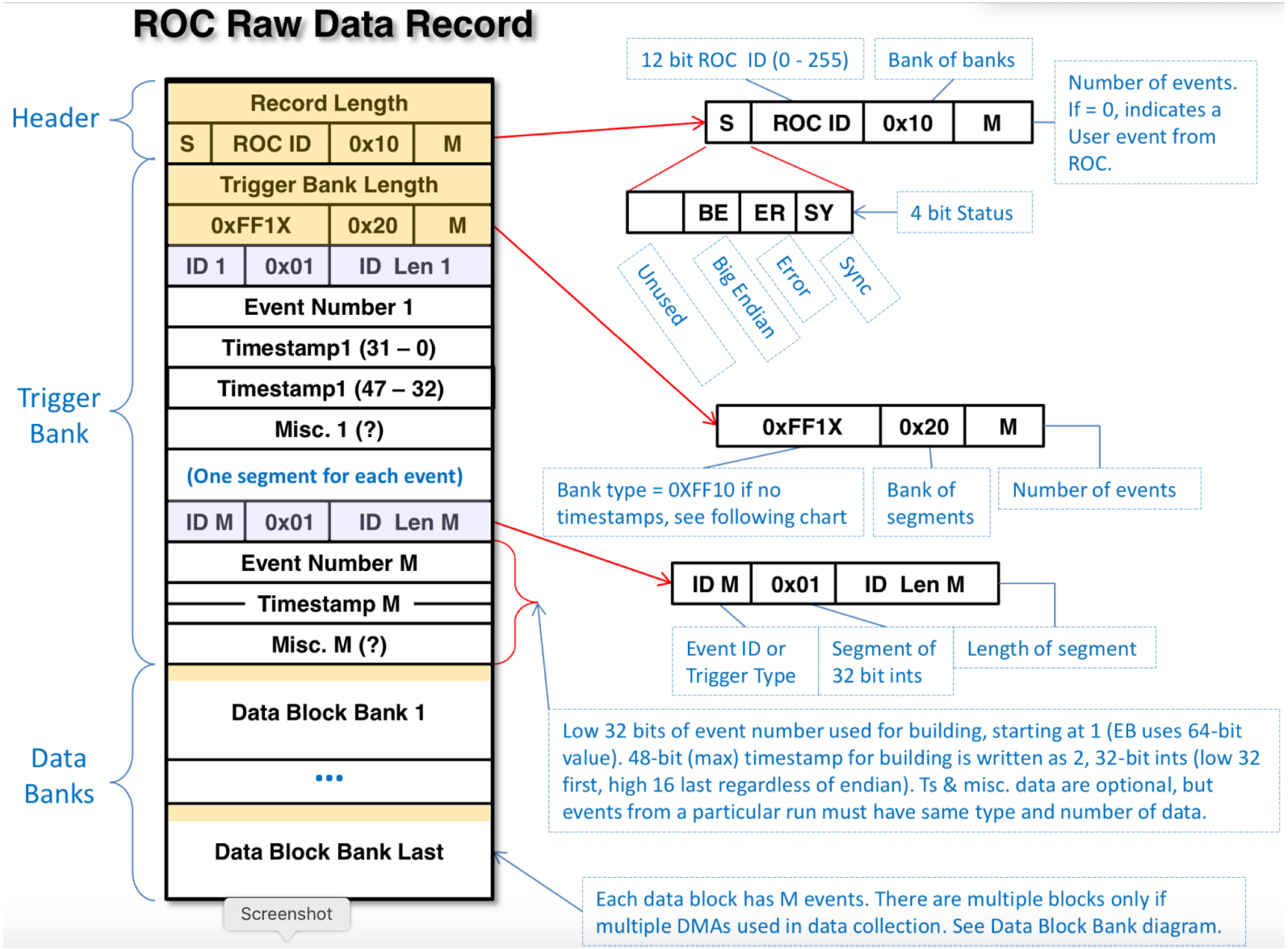

## Evio Content Type Codes

| Content Type | Primitive Data Type |
|---|---|
| 0x0 | 32 bit unknown (not swapped) |
| 0x1 | 32 bit unsigned int |
| 0x2 | 32 bit float |
| 0x3 | 8 bit char* (string) |
| 0x4 | 16 bit signed int |
| 0x5 | 16 bit unsigned int |
| 0x6 | 8 bit signed int |
| 0x7 | 8 bit unsigned int |
| 0x8 | 64 bit double |
| 0x9 | 64 bit signed int |
| 0xa | 64 bit unsigned int |
| 0xb | 32 bit signed int |
| 0xc | Tag Segment |
| 0xd | Segment |
| 0xe | Bank |
| 0xf | Composite |
| 0x10 | Bank |
| 0x20 | Segment |

Jefferson Lab

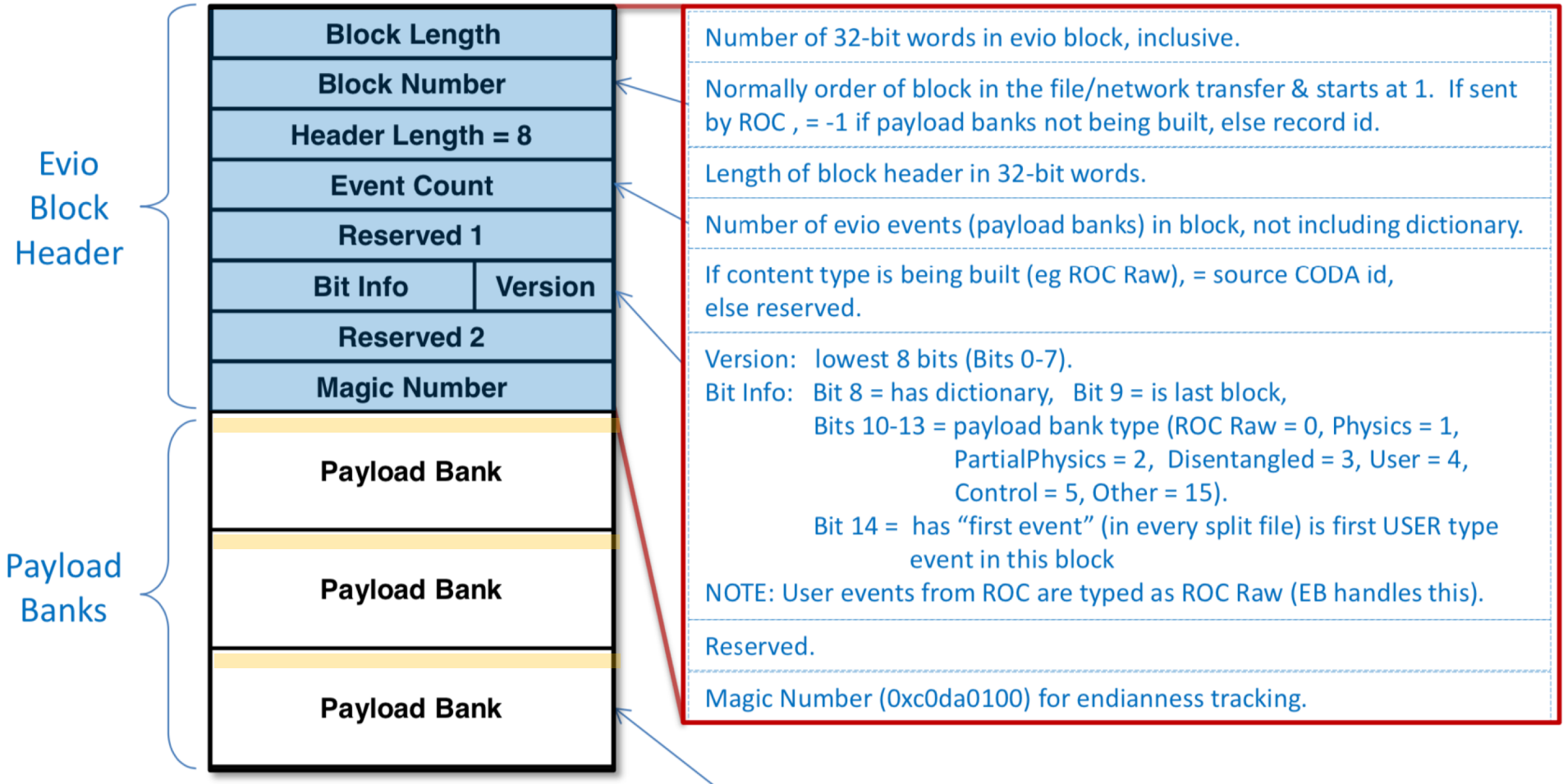
# Front-End (ROC) Data

- The CODA Readout Controller (ROC) is responsible for collecting raw data from digitizers as well as trigger and timestamp information and wrapping it up for transport.
- There is always a Trigger Bank followed by a User definable number of Data Banks.
- Note that Banks can contain Banks or Segments which can further encapsulate information for the User.
- Lengths in the headers facilitate finding the start of the next bank or segment.




Jefferson Lab

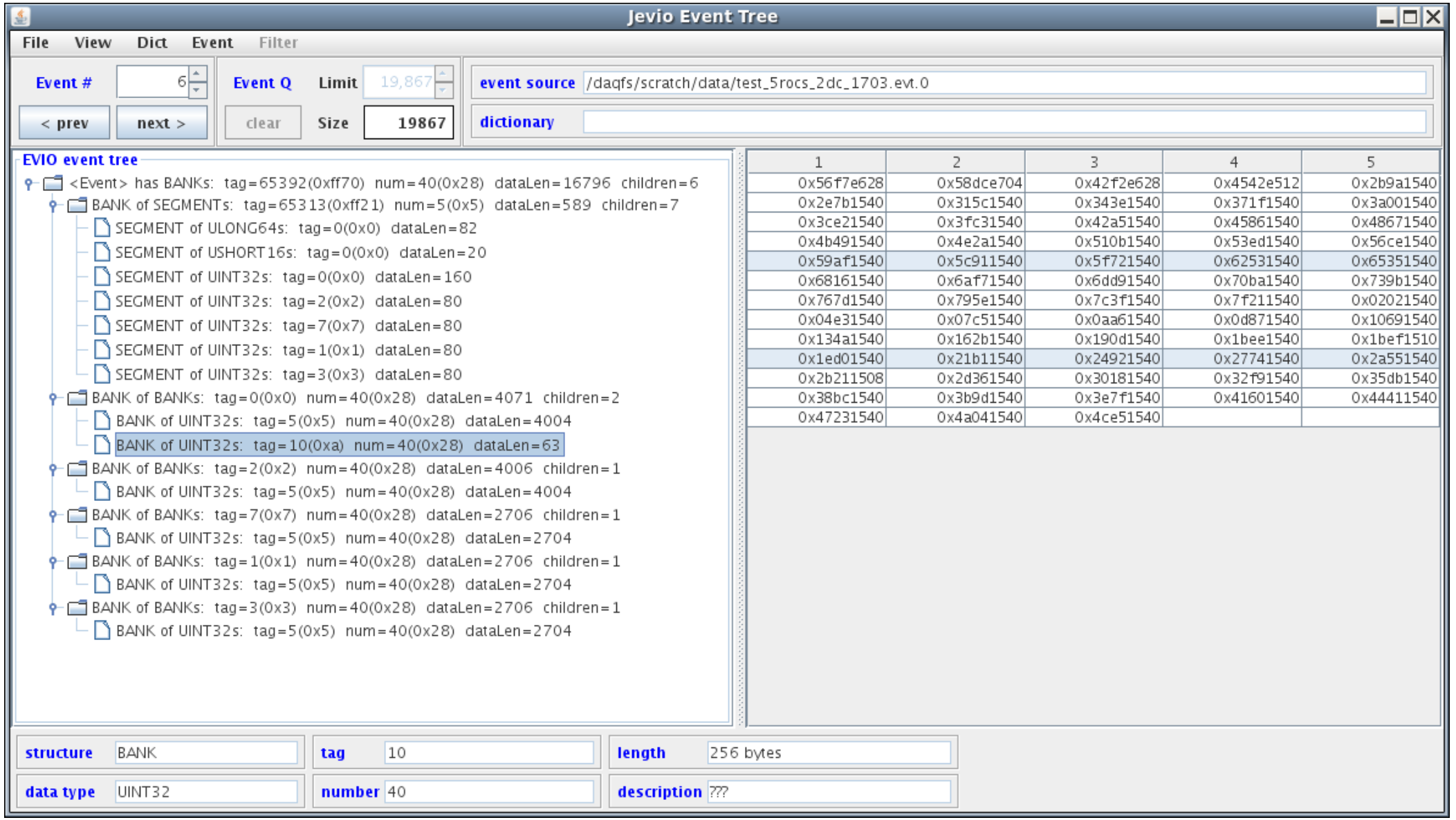
# CODA Data Transport and Files




Format used when sending all types of online CODA data over the network. They are in standard evio buffer/file output format with block headers.

Jefferson Lab

# CODA Tools (jeviodmp)

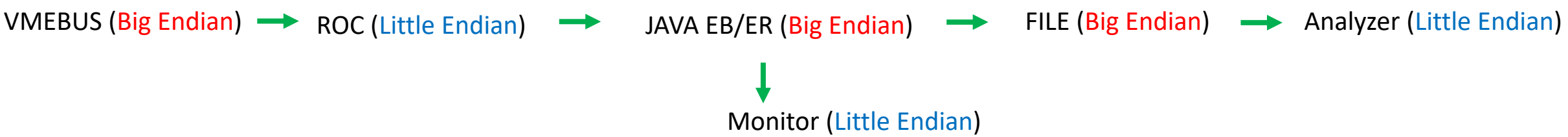



| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 0x56f7e628 | 0x58dce704 | 0x42f2e628 | 0x4542e512 | 0x2b9a1540 |
| 0x2e7b1540 | 0x315c1540 | 0x343e1540 | 0x371f1540 | 0x3a001540 |
| 0x3ce21540 | 0x3fc31540 | 0x42a51540 | 0x45861540 | 0x48671540 |
| 0x4b491540 | 0x4e2a1540 | 0x510b1540 | 0x53ed1540 | 0x56ce1540 |
| 0x59af1540 | 0x5c911540 | 0x5f721540 | 0x62531540 | 0x65351540 |
| 0x68161540 | 0x6af71540 | 0x6dd91540 | 0x70ba1540 | 0x739b1540 |
| 0x767d1540 | 0x795e1540 | 0x7c3f1540 | 0x7f211540 | 0x02021540 |
| 0x04e31540 | 0x07c51540 | 0x0aa61540 | 0x0d871540 | 0x10691540 |
| 0x134a1540 | 0x162b1540 | 0x190d1540 | 0x1bee1540 | 0x1bef1510 |
| 0x1ed01540 | 0x21b11540 | 0x24921540 | 0x27741540 | 0x2a551540 |
| 0x2b211508 | 0x2d361540 | 0x30181540 | 0x32f91540 | 0x35db1540 |
| 0x38bc1540 | 0x3b9d1540 | 0x3e7f1540 | 0x41601540 | 0x44411540 |
| 0x47231540 | 0x4a041540 | 0x4ce51540 | | |

Jefferson Lab

# Some Comments on Endianness

- A typical situation in the current CODA Framework:

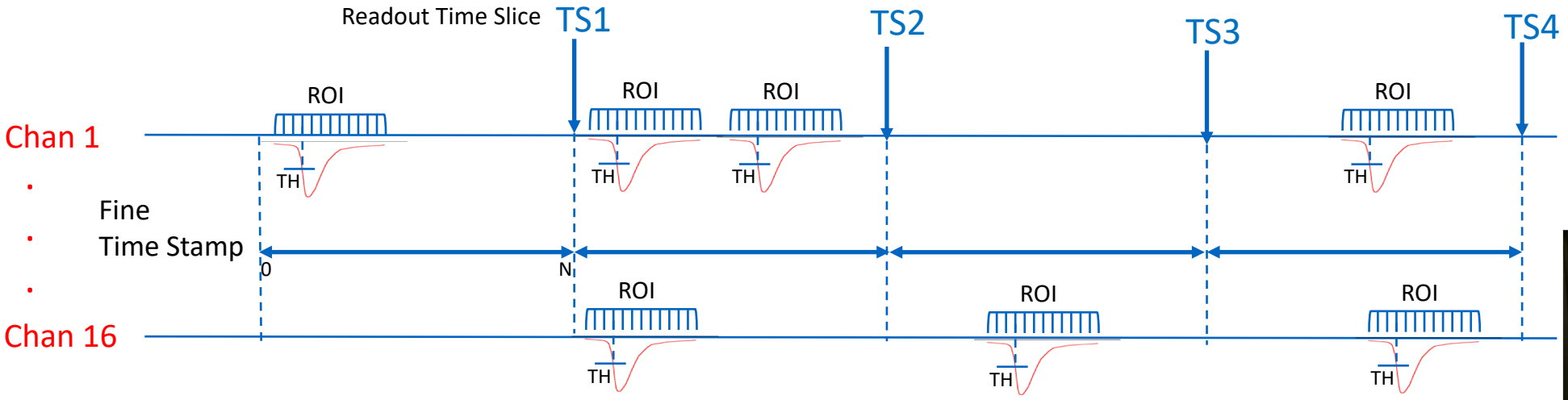


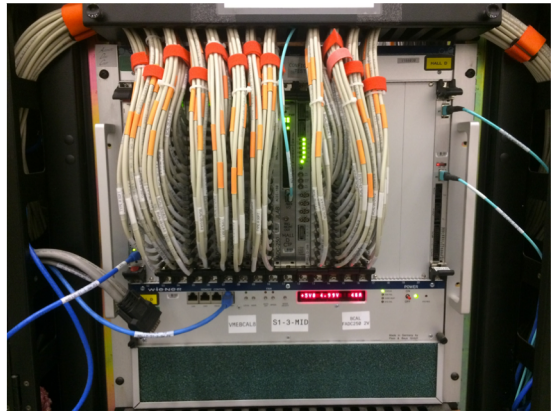

- To try and minimize the issues with Endianness, when we convert a EVIO Data Block BIG to LITTLE or visa–versa we only swap the headers and built banks.
- The Raw User Data Banks are not touched (Except for the Bank and Segment Headers).
- This allows all the CODA EVIO tools to be used, but the User is responsible for knowing what the original Endianness of the raw data was when it was readout and wrapped up in the Bank.

# JLAB VXS FADC in “Streaming Mode”

Streaming data can be thought of as Triggered mode where the trigger is a fixed Frame Rate (internal trigger) and you keep all the data for a single or multiple channels in the current frame (Time Slice).

The JLab 250 MHz FADC generates a 12 bit sample every 4ns. That’s 3 Gb/s for one channel. A 16 channel module is 48 Gb/s. That is over twice the available VXS slot bandwidth. But we don’t need ALL the data.



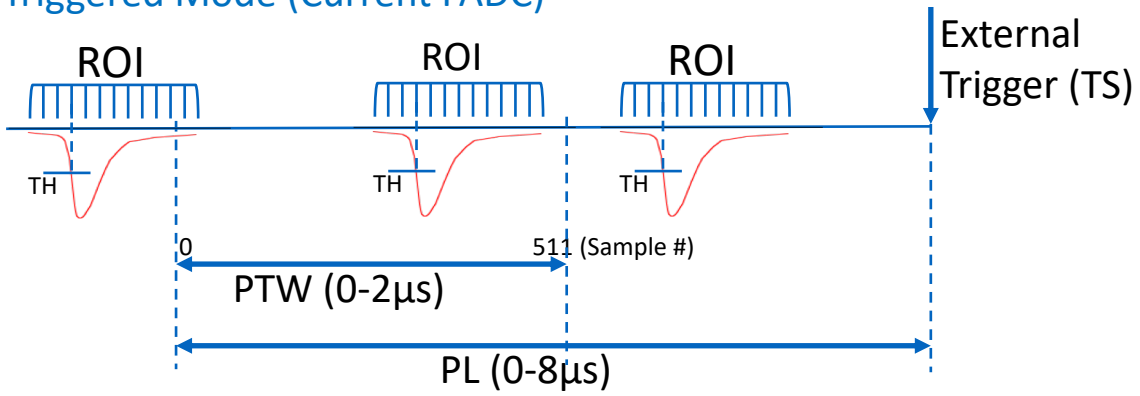
Within the FPGA we keep only the data around a Region of Interest (ROI) from each channel, along with a fine time stamp in each time slice window.

Depending on hit rates and available bandwidth , We can keep the individual samples or just compute a sum.

Jefferson Lab

# Triggered vs Streaming

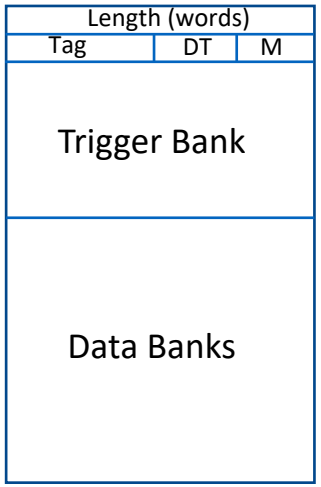
## Triggered Mode (Current FADC)




PL: Programmed Lookback  PTW: Time window
Data we get on a trigger:
ADC Values, Threshold Sample #, Trigger Time Stamp

For a given Block of Triggers we create a bank of data:

ROC Header

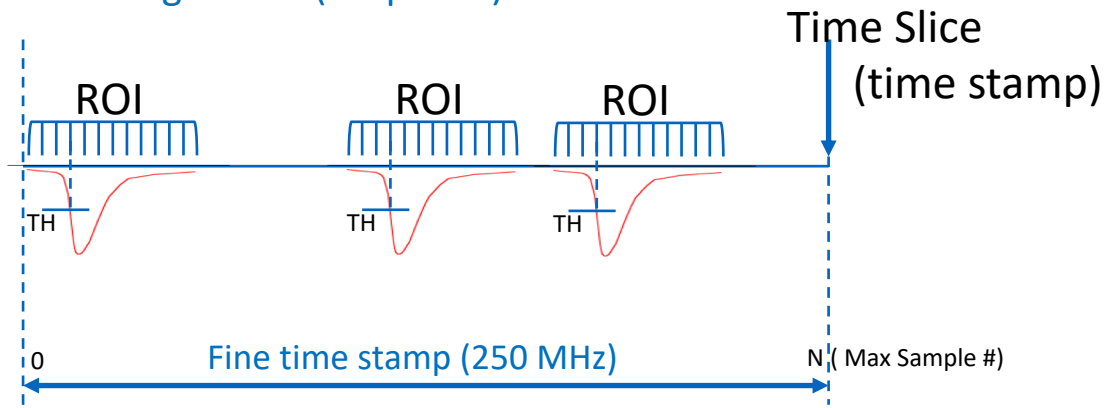



Tag includes: ROC ID/Status
DT: Bank of Banks
M: # of Events in the Block

Trigger Bank includes:
(For M Events)
Trigger Time stamps
Event #s

Data Banks include:
(For M Events)
ADC Values or integrals
Sample #s/Thresh time
For ALL Modules in the Crate

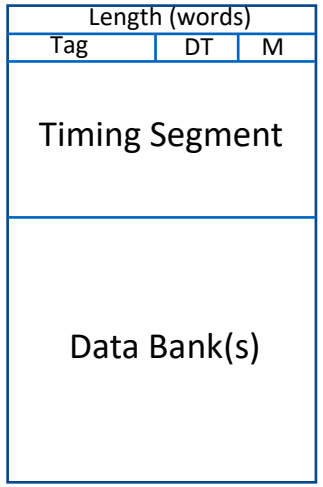
## Streaming Mode  (Proposed)




1 Frame = N Clocks  (16bits -> up to 262 µs for a 4ns clock)
(Each FADC can self trigger at the desired Frame Rate)

For a given Time Slice we can create a  new bank of data:

Stream Header




Tag includes: Module ID/Status
DT: Bank of Banks
M: # of ADC Channels

Timing Bank includes:
Time Slice Value (in # clocks)
Time stamp

Data Banks include:
(For one module - M Channels)
ADC Values or integrals
Sample #s/Thresh time

Jefferson Lab

# Triggered vs Streaming Blocks

- In the GlueX Experiment at JLAB the L1 trigger creates about 100 kHz Event rate ( ~10 μs/trigger). This means that we would need to define a Streaming time slice of 10 μs to, on average, get one good event in the data.
  - How much more junk data will be in this time slice?

- GlueX already blocks 40 events for transport from the ROC.
  - It seems reasonable we should make time slices even larger?
  - A 65 μs time slice corresponds to ~6-7 events and a very managable 15 kHz “Frame” rate.

- The larger the time slice –the more efficient DAQ becomes.
  - Less overhead in the data formatting – e.g. more data with fewer headers
  - Larger blocks of data moved more efficiently in the back-end
  - Making sure there are multiple ”Events” in the slice can make processing more efficient as well (smaller % of boundary events).

# Stream “Building” Considerations

- A Frame has a number (like an Event #) but also an absolute Time stamp based on a system clock.
- Raw streams can have a unique bit/byte structure until they reach the 1st Aggregation point.
  - First opportunity to process and/or filter a Frame and combine with other stream Frames for a given “Time Stamp”.
- Raw stream data can be wrapped in a bank structure (Raw Stream Bank – RSB)
- Multiple streams will be wrapped in an additional Bank (Aggregate stream Bank – ASB)
  - It will be an Bank containing Banks. The Tag will contain the Aggregation ID. This should be unique to the experiment
  - Include Time Stamp Segment-TSS (analogous to the Trigger bank).
  - Then perhaps an Aggregation Info Segment - AIS)
- Subsequent Aggregation of ASBs (secondary aggregation points)
  - Since Aggregation IDs must be unique for the the system, we can now simply build new ASBs that are just a collections of ASB Banks with an Aggregated TSS (just like built trigger banks) as well as an aggregated AIS.
  - Original TSS can be removed.

# Stream Aggregation

## Raw Stream Bank (RSB)

RSB Header

| Length (words) | | |
|---|---|---|
| Tag | DT | SS |
| Raw Stream Data for one Time slice | | |

Tag: Local Stream ID – Unique just to the 1st Aggregation point
Detector/Channel info?

DT: Data Type – User specified

SS: Stream Status – Raw Bank, Did data get dropped? Clock info, Error?

## Aggregate Stream Banks (ASB)

### 1st Aggregation

ASB Header

| Length (words) | | |
|---|---|---|
| Tag | DT | SS |
| TSS | | |
| AIS | | |
| RSB 1 | | |
| RSB 2 | | |
| ⋮ | | |
| RSB N | | |

TSS – Time Stamp Segment
(holds Time slice number and time stamp for first aggregation point ).

AIS – Aggregation Info Segment
(holds number of RSBs that follow).

Tag: Aggregation Point ID – Unique to the whole system
DT: Data Type – Bank of Banks (0x10)
SS: Stream Status – Aggregate Bank, 1st Aggregation
Did data get dumped in any of the streams?

### Subsequent Aggregations

ASB Header

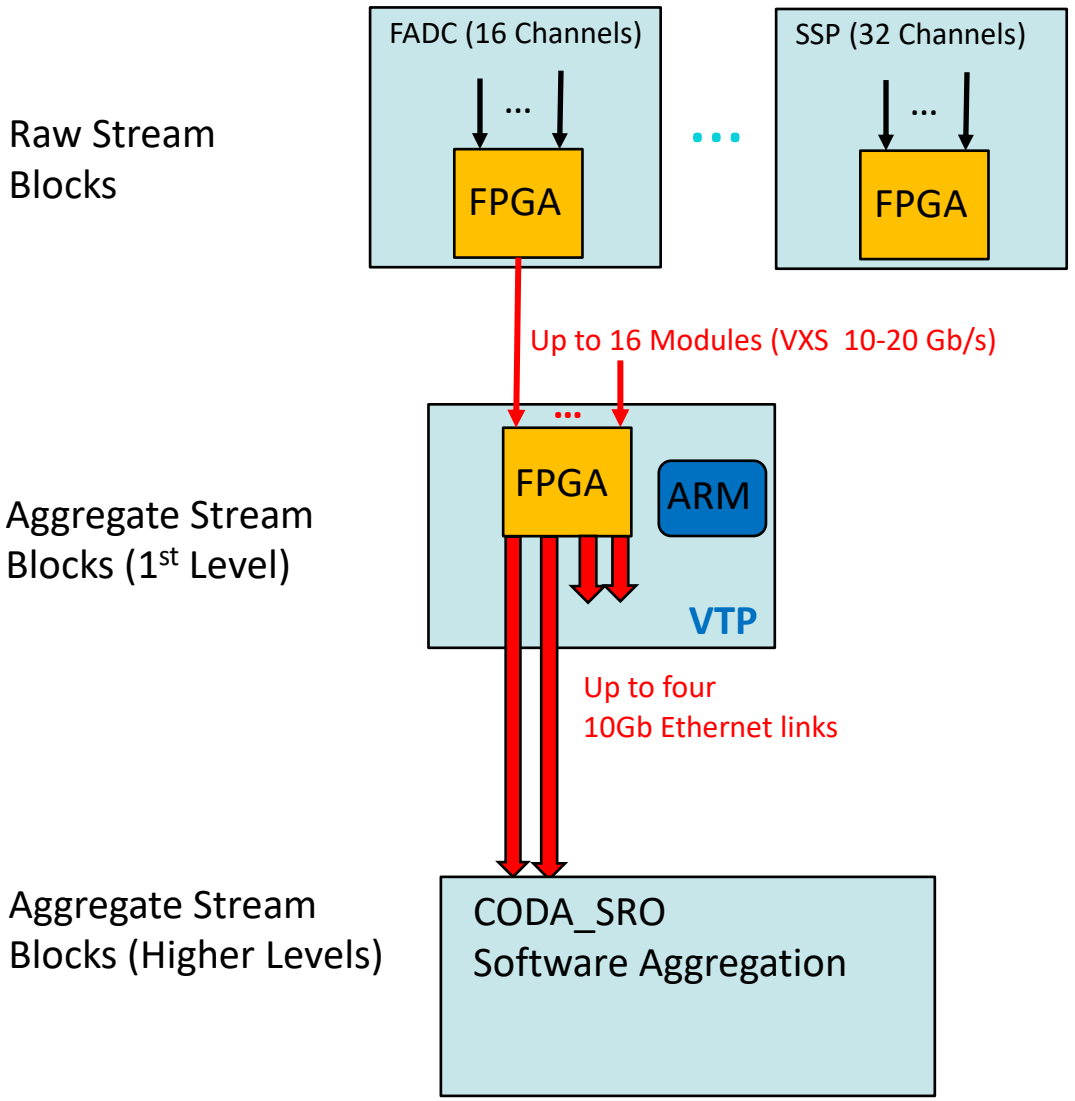
| Length (words) | | |
|---|---|---|
| Tag | DT | SS |
| A-TSS | | |
| A-AIS | | |
| ASB 1 | | |
| ASB 2 | | |
| ⋮ | | |
| ASB M | | |

For subsequent aggregations the ASBs are simply appended and the TSS and AIS are updated to hold all Time stamp info and Total number of ASBs.
**(In principle we could dump the TSS in each of the original ASBs)**

Jefferson Lab

# Aggregation Issues

## JLAB VXS Architecture




- Initial stream aggregation will almost always happen in firmware (FPGAs)
- Stream Aggregation points should have a well defined Data Loss algorithm
  - There is no “trigger” to disable but do we need to disable the whole system at the front-end by a “busy”?
  - Backpressure to an individual stream source may not be practical.
  - Better to drop data from a ”hot” channel before it is aggregated.
- Provide configurable Frame buffering for both inputs and outputs to mitigate network/software latencies.

Jefferson Lab

# Back-End Considerations

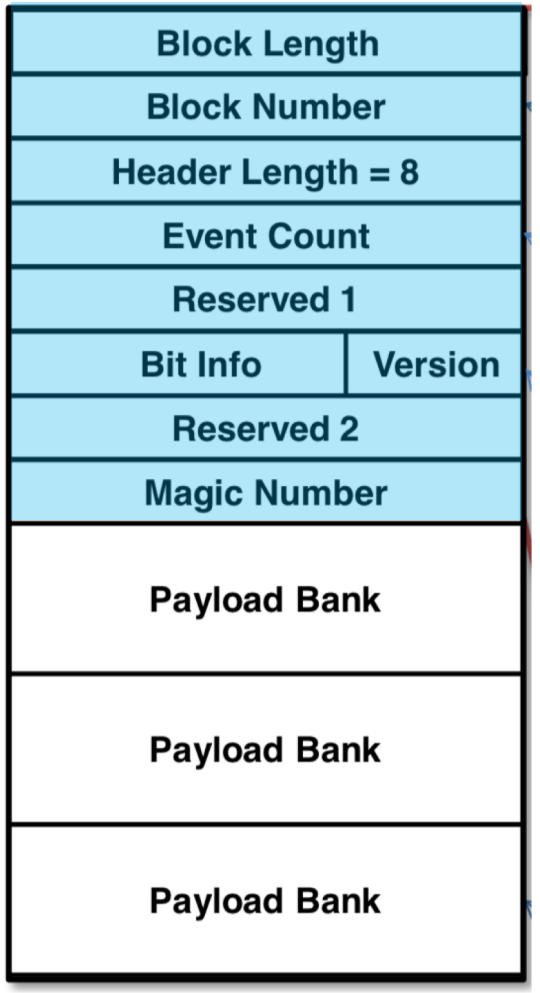
- Implementation of a new data encapsulation standard for CLAS12 offline analysis called HIPO -- High Performance Output prompted efforts to keep CODA EVIO format as consistent as possible with the new standard. Version 4 was too limited.

- New version of EVIO (Version 6) recently developed that provides a compatible File and Data transport header for both EVIO and HIPO.

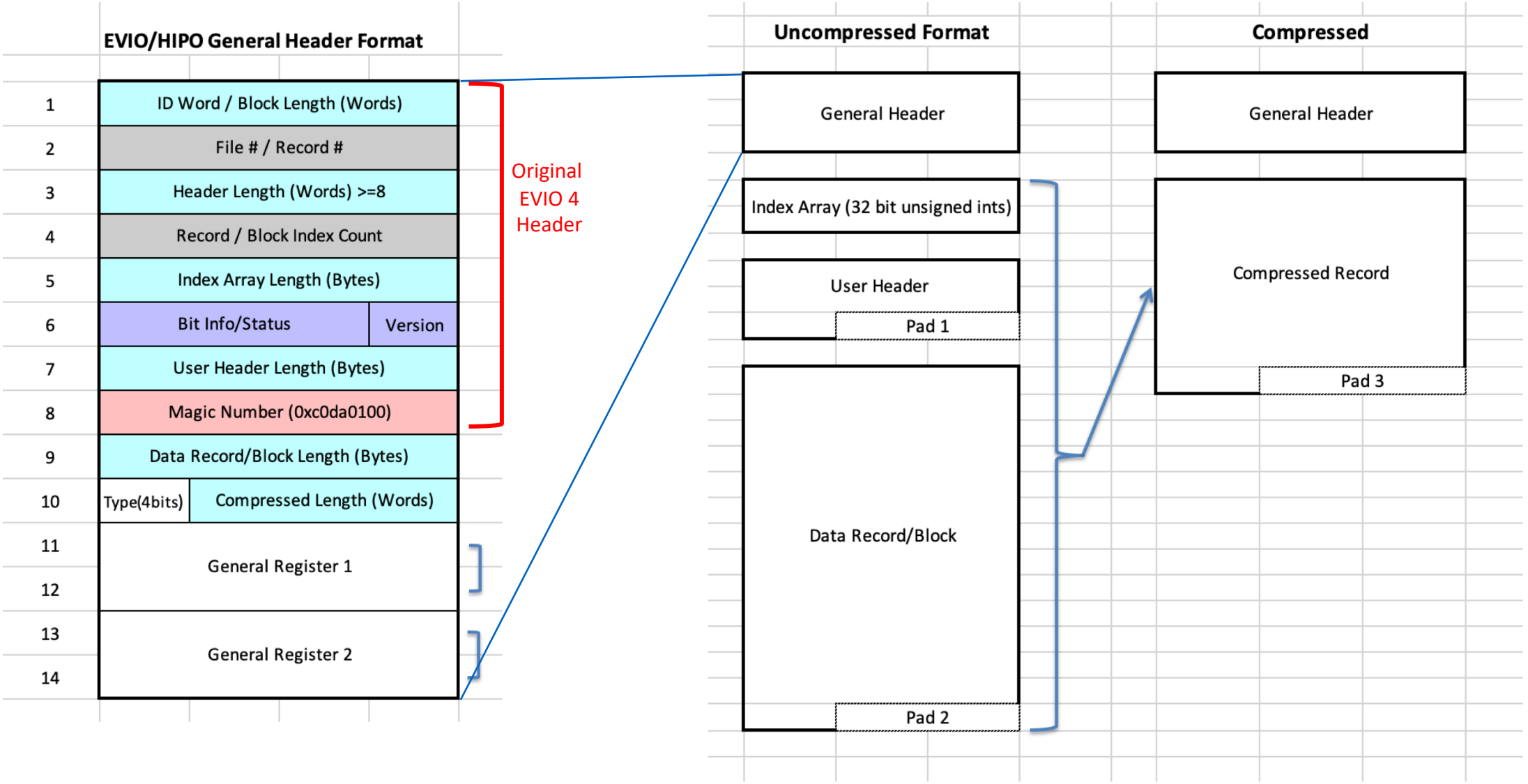
- Support is also included for data compression as well as optional User headers and index array pointers to help provide more efficient file skimming.

- These new features can be useful in the streaming model for the Back-End processing.

EVIO (Version 4)



# EVIO (Version 6) and HIPO Support




Jefferson Lab

# Summary

- Standardizing on a consistent data encapsulation format from the front-end to the back-end is critical to providing common tools for efficient DAQ systems debugging and analysis.

- New consideration for the increased requirements of Back-end processing in the streaming model need to be made to help facilitate processing efficiency.

- For JLab, we see a path forward by adapting the existing CODA EVIO data and file formats to the new streaming readout architecture.

- However, if the community can agree on some common standard, JLab is in the best position now to adopt and continue to develop with this standard in place for SRO.