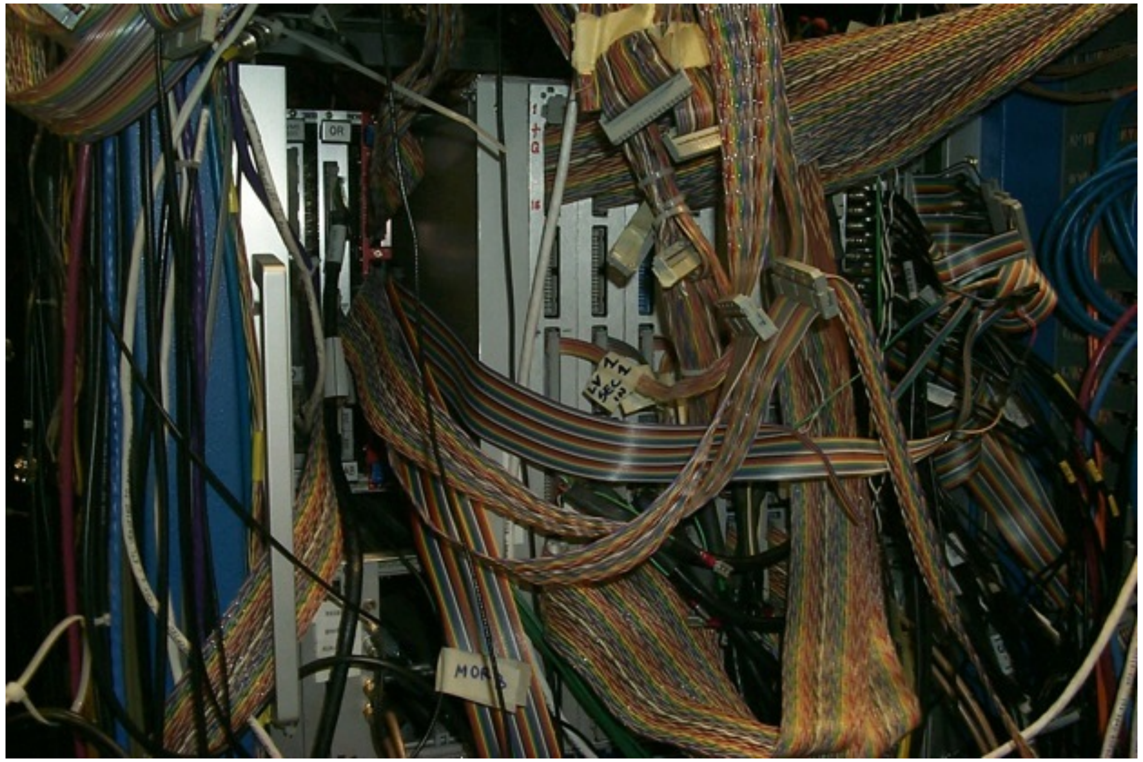
Data Acquisition at Jefferson Lab



David Abbott

Data Acquisition Support

Experimental Nuclear Physics

(June 2020)

Data Acquisition

- **data acquisition**

  Verb - Data acquisition is defined as the process of collecting and organizing information.

- Two data acquisition support groups at Jlab help experimenters to take the data from their detectors and store it for future analysis.
  - **FEDAQ Group** – Physics Division – primarily front-end hardware and software (close to the detectors).
  - **EPSCI Group** – CST Division – focus on back-end software tools for data transport, processing and storage.

- Anatomy of this DAQ Talk :
  - What are all the pieces of a typical DAQ system?
  - How we have implemented them here at Jlab?
  - What are we working on for the future?
  - Hopefully I can make it interesting…

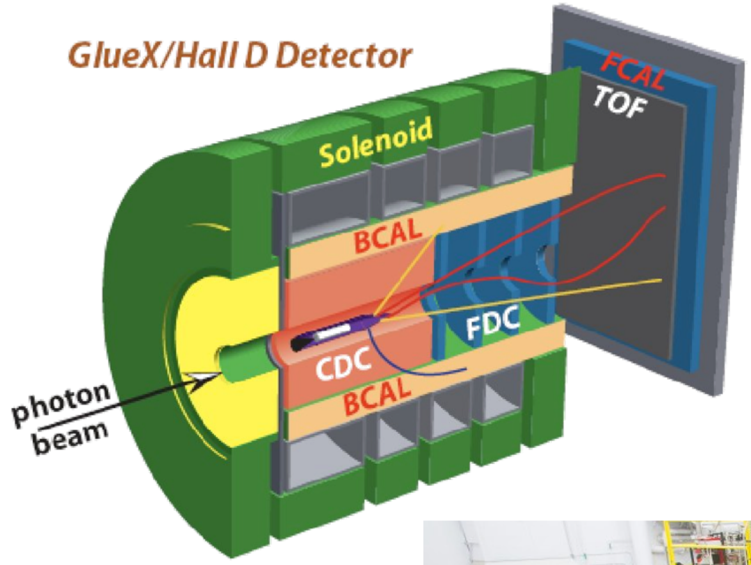
We all can usually be found here -
on the 2nd Floor of F-Wing in Cebaf Center
(when there is not a pandemic going on)

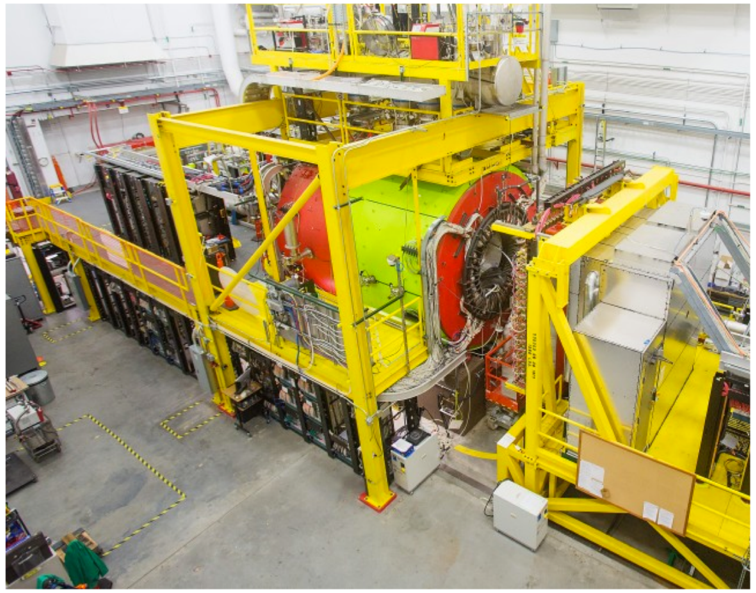
FEDAQ (Fast Electronics and Data Acquisition)
EPSCI (Experimental Physics Scientific Computing Infrastructure)

Detection and data acquisition

- When a particle interacts with a target, resultant particles from the interaction enter a detector.
- Various sub-detectors measure properties of these particles (type, energy, trajectory) as electrical signals.
- Three basic types of measurements are - charge, time and count.
- Electronics and software convert electrical signals into digital data that can be stored and later analyzed.

  ADC = charge, TDC = time, Scaler = count
- All of the data generated from one "interaction" is called an Event.



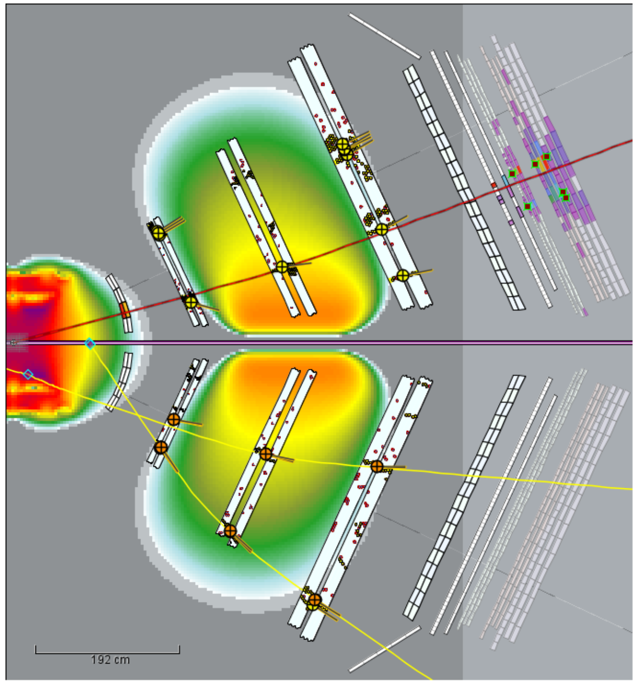


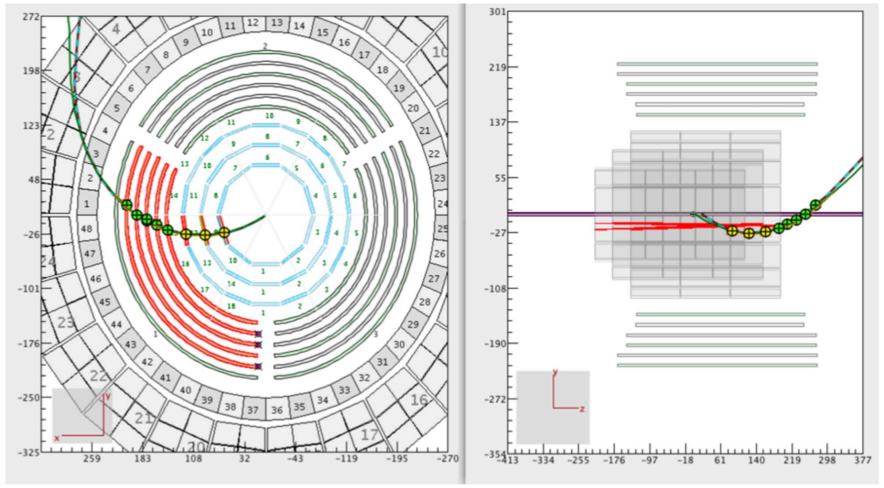
The Event as a unit of data

- Data from one event has no history. It doesn't depend upon events that went before and doesn't influence later events.

- Events occur with random timing.
  - Average rates from a few Hz to 100s of kHz
  - Hardware may not be ready for new data.
    - Dead time when data is lost.
  - Events may overlap in time - event pileup.
  - Peak event rates can be much more than the average.

- Total event size depends upon the physics.
  - Accidental hits unconnected with event.
  - Electronic noise.
  - Distribution of event sizes.
    - Some very large events.





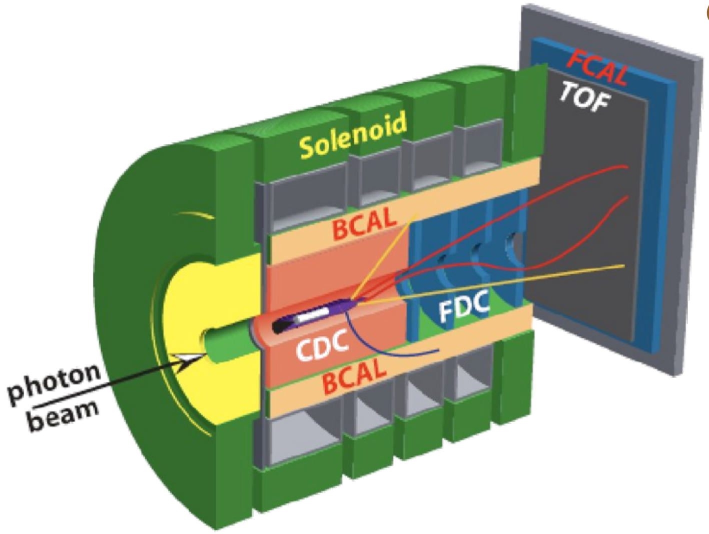
Jefferson Lab

Data acquisition for big experiments

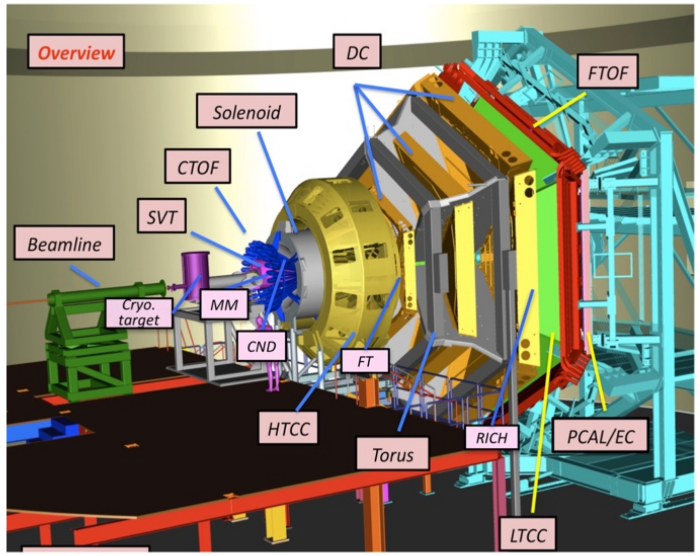
- Detecting hardware is large (many thousands of channels) and physically distributed within the detector. So we need to:
  - Tag where the data came from (and when).
  - Gather all data "fragments" for one Event together.
  - Do it as efficiently as possible (no back-end dead time)

- Experiments typically run for weeks or months. So we need:
  - Stability.
  - Control - to start and stop the whole system.
  - Monitor all experimental conditions under which data was taken.

GlueX/Hall D Detector



Clas12/Hall B Detector

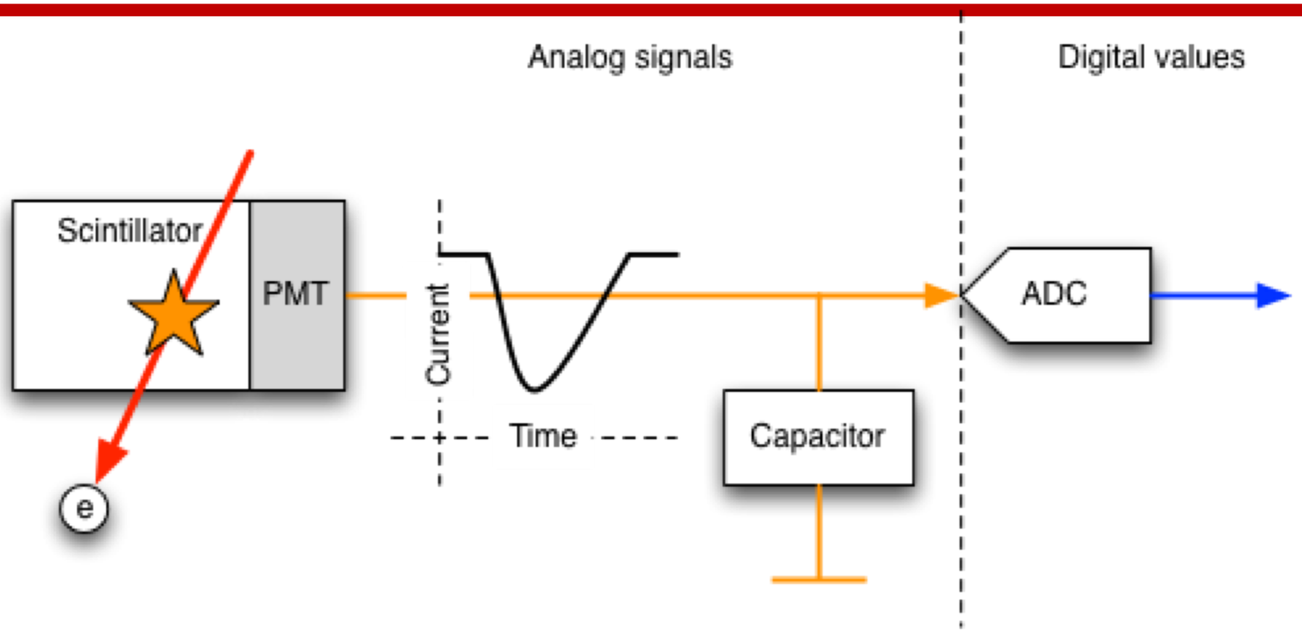


Jefferson Lab

Anatomy of a DAQ System

- **Readout** – digitizing detector signals
- **Triggering** – choosing the data we want to keep
- **Data formatting** - standardize the data we are saving
- **Event building** - putting all the event fragments together
- **Event transport -** make events available to all
- **Event storage -** save events for later analysis
- **Run Control** - configure, start and stop experiments
- **Monitoring** - tell the experimenter what's going on

Detector readout example, a scintillator

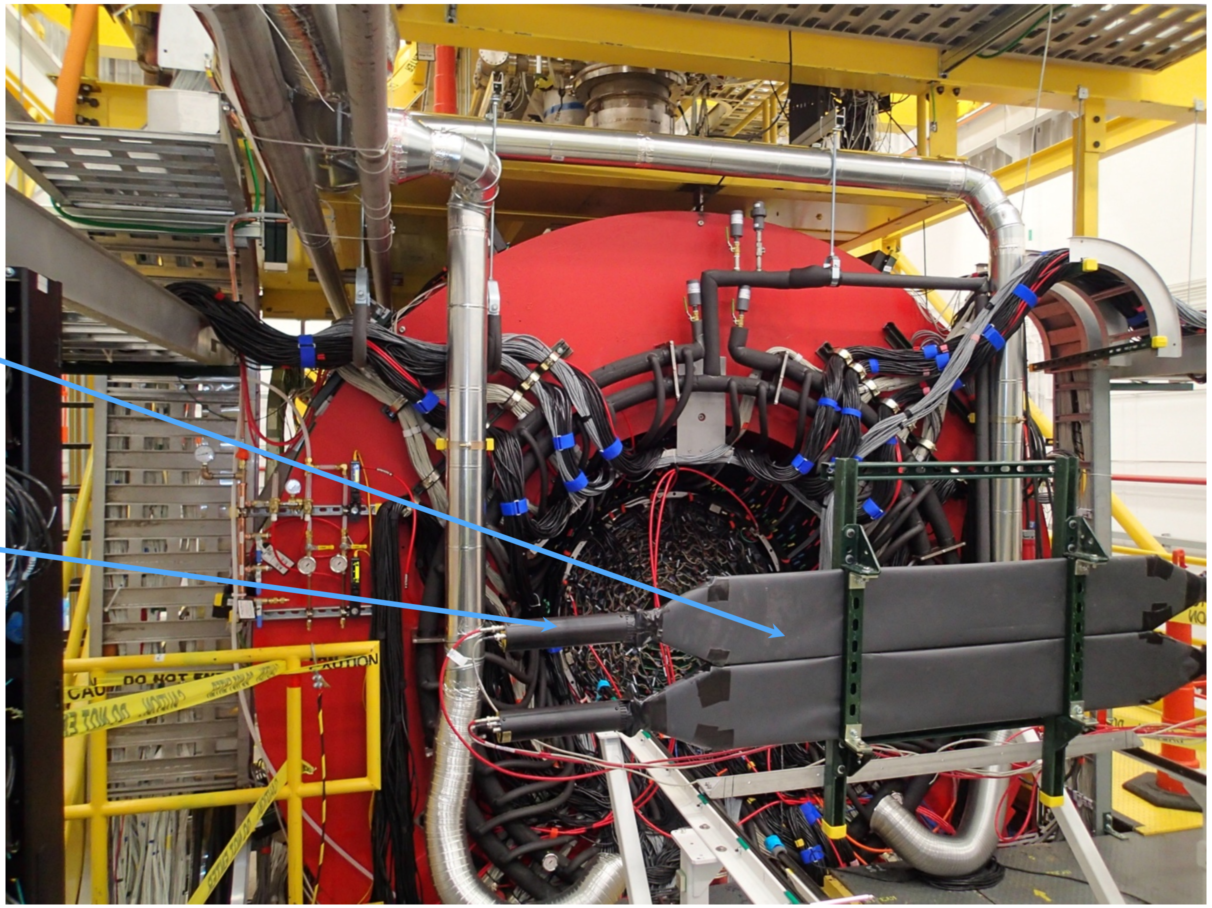


- A particle deposits energy in a scintillating material that converts it into light.
- A Photo Multiplier Tube (PMT) converts the light into a pulse of electricity.
- The charge is captured to generate a voltage.
- These pulses are typically fast (~ 10s of ns wide)
- An Analog to Digital Converter (ADC) measures the voltage as a digital value.

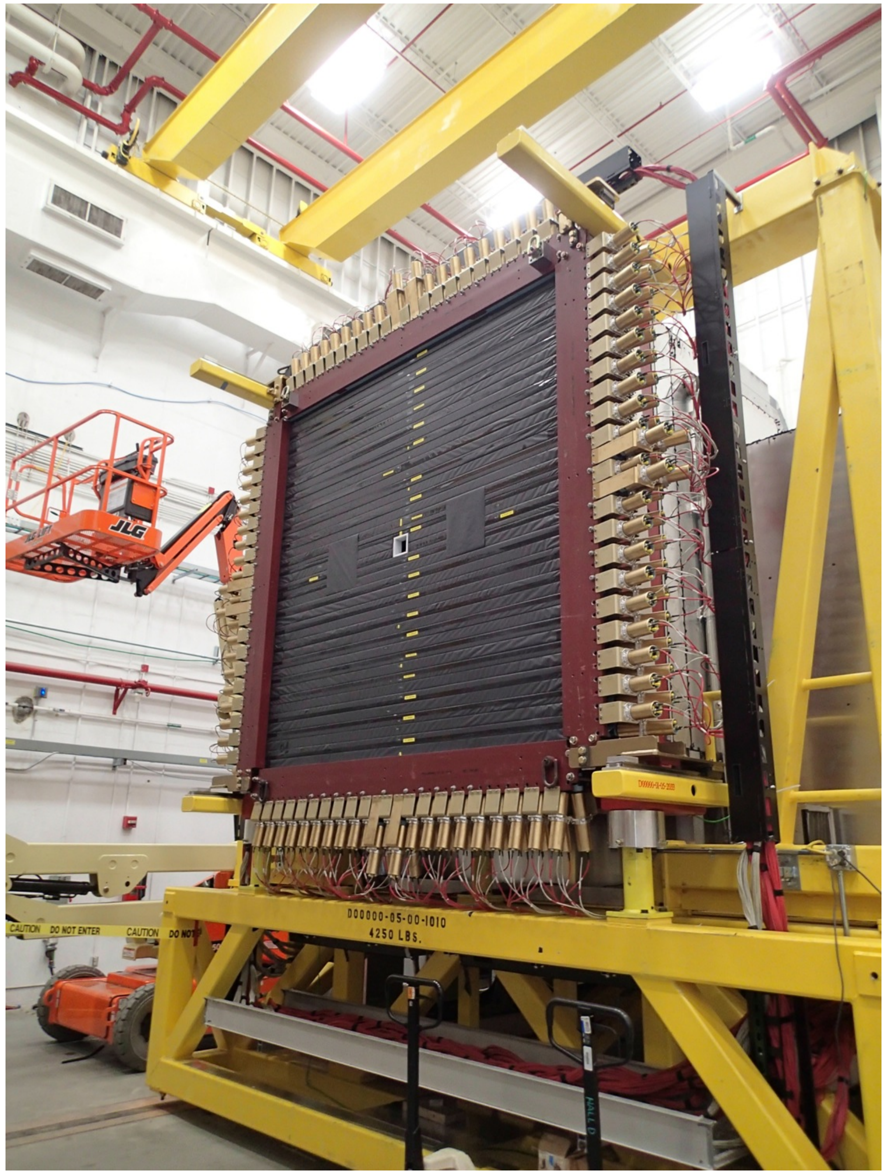
Picture of test scintillators in Hall-D

Scintillator

Photomultiplier

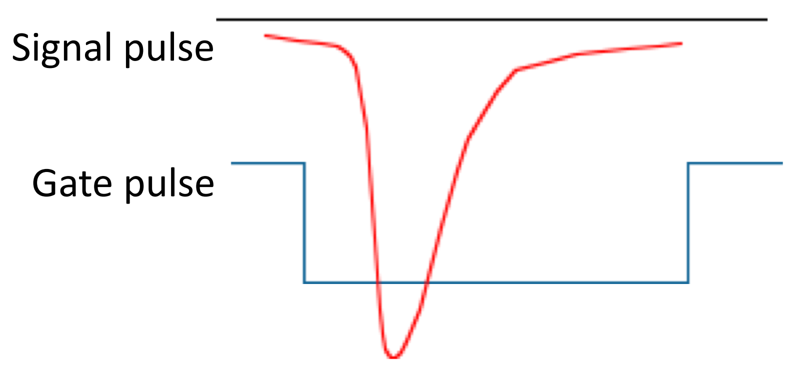


Array of scintillators

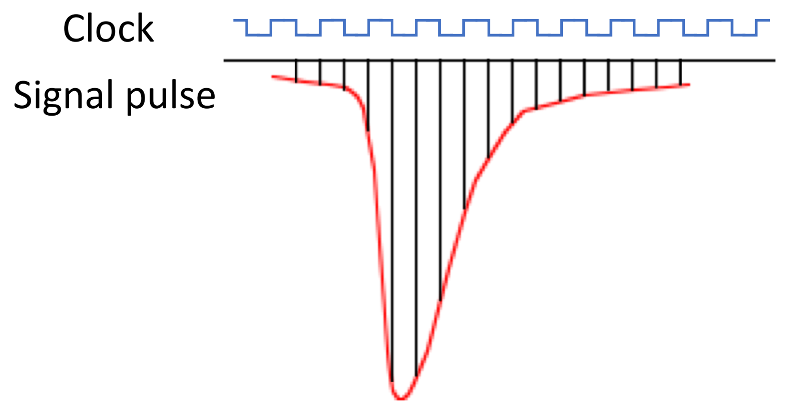


Sampling vs Integration - ADCs

- A traditional "integrating" ADC can take many microseconds to digitize a pulse. A gate (logic) pulse, generated by trigger electronics marks the region of interest for the signal and enables integration of the charge from the signal pulse.
- This type of ADC generates a single measurement representing the charge sum during the gate.
- During the digitization period after the gate any later pulses from the signal are lost (readout dead time).



- A Flash ADC samples continuously at a fixed rate based on an input clock.
- For example, a 250 MHz ADC samples every 4 ns and generates ~5-15 measurements during a typical gate.
- These samples describe the pulse shape as well as the total charge.
- There is no dead time.

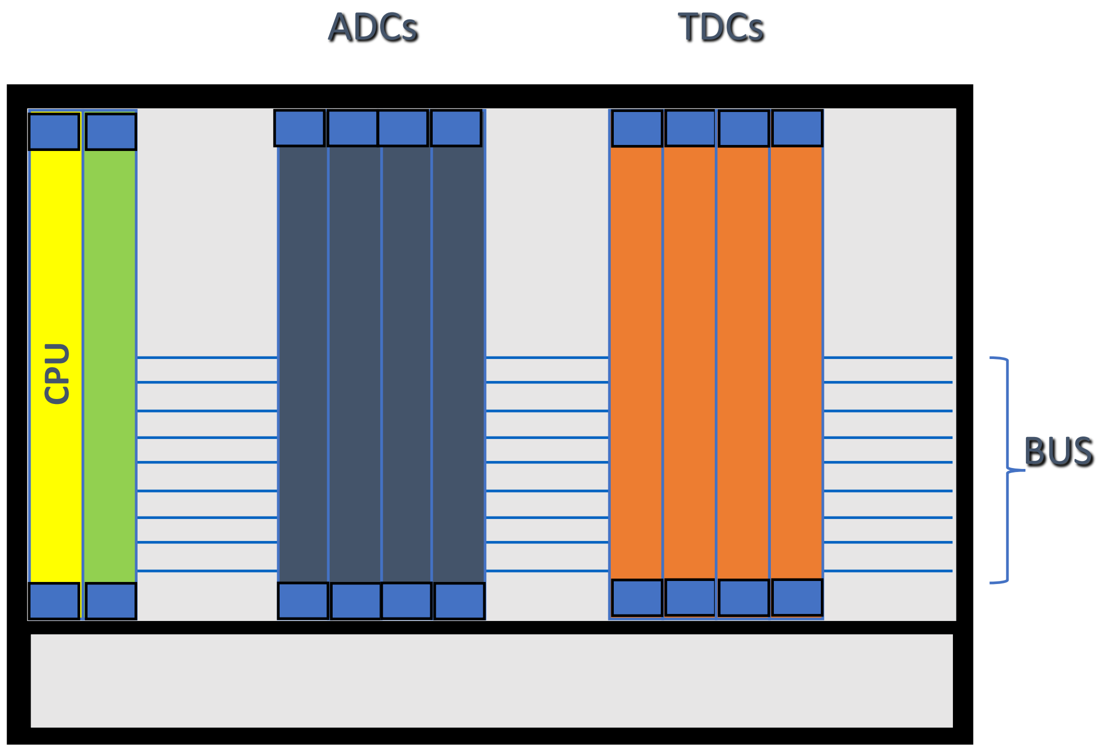


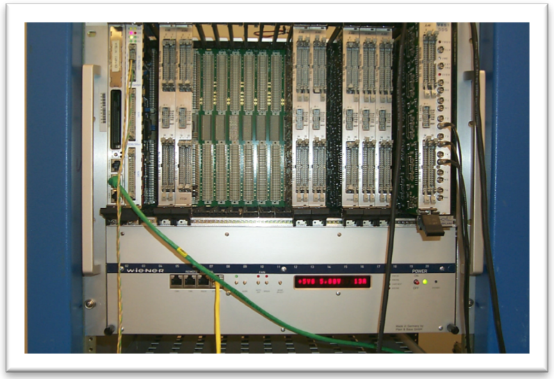
Front-End Electronics (Modules / Buses)

- Big Detectors in the experimental halls can have many 1000s of channels. All must be digitized (ADCs TDCs).
  - Pack these circuits onto modules
  - Pack modules into Crates
  - Place crates into Racks
  - Use a standard bus to connect everything to a CPU.

- Common parallel bus standards still found in experimental nuclear physics include:
  - CAMAC (24 bit, 3MB/s)
  - FASTBUS (32 bit, 40MB/s)
  - VME (32/64 bit, 40 MB/s – 320 MB/s)
  - PCI (64 bit, 500 MB/s)

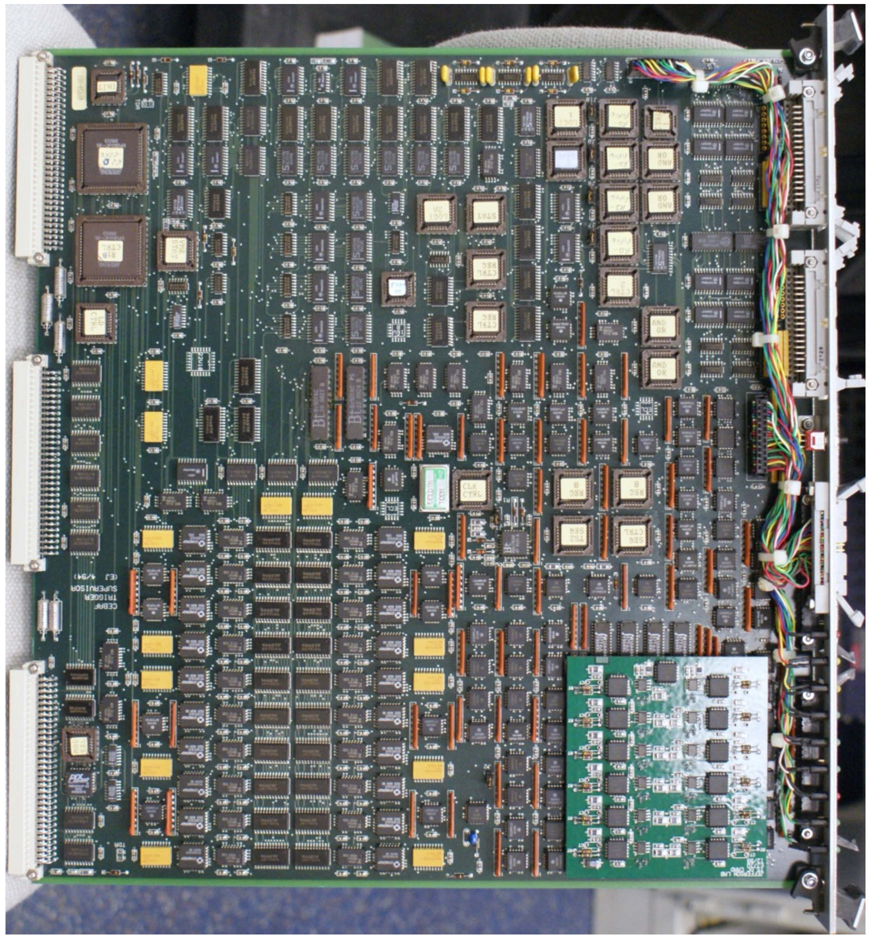
- In the last ~10 years, parallel buses have been mostly replaced or extended using serialized buses.
  - eg USB, PCIe, VME/VXS
  - We will talk more about this later…



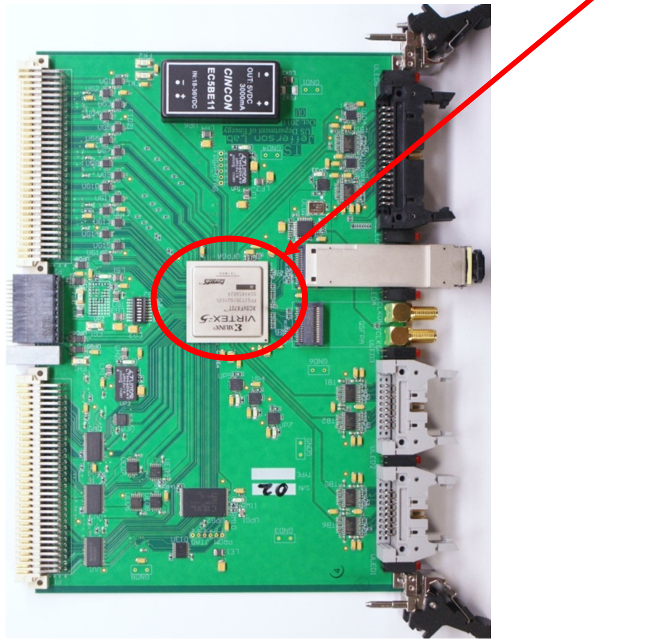


Electronics evolve

Trigger Supervisor – Circa 1995



Circa 2011



FPGA – Field Programmable Gate Array

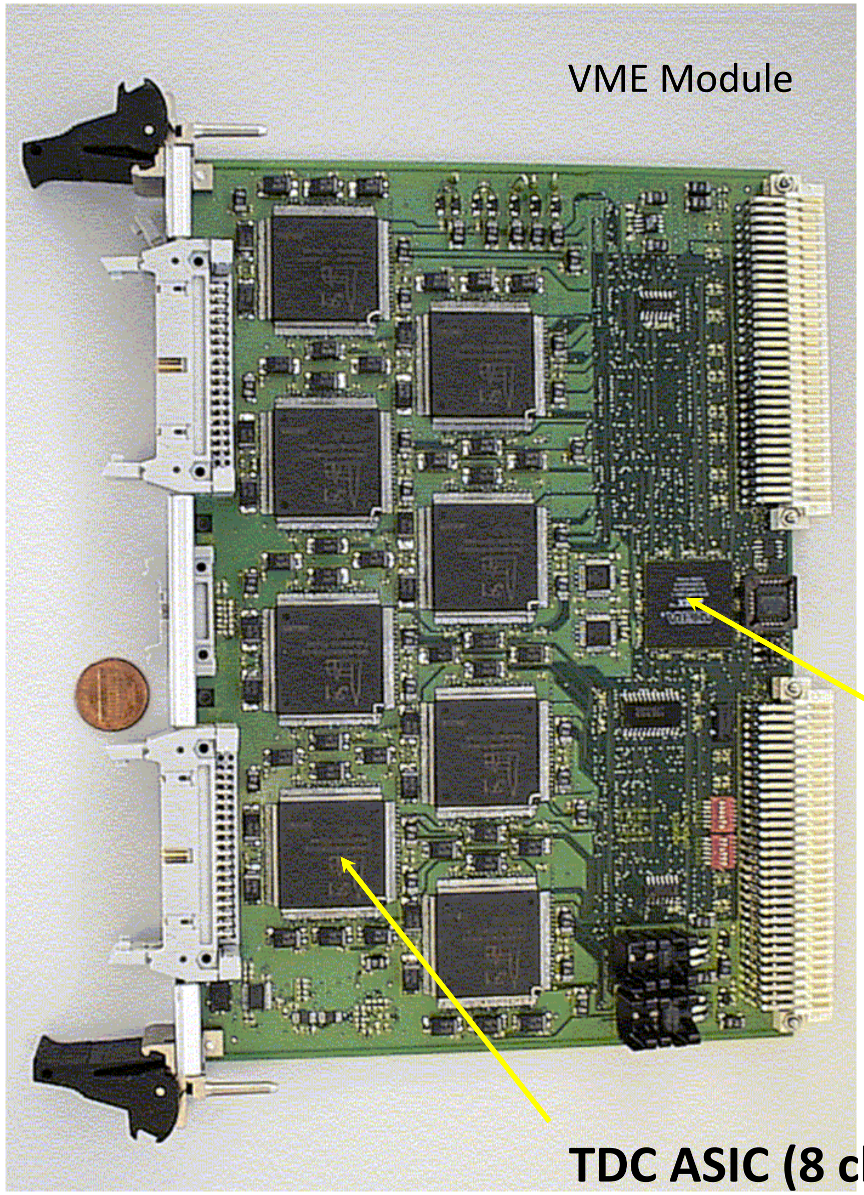
We can now buy programmable logic arrays that allow us to implement complex algorithms in the firmware on a single chip.

Virtually all the hardware trigger processing can be done in these very powerful chips. We just have to get all the relevant detector signals to them.

Another important development in electronics is the ASIC or Application Specific Integrated Circuit.

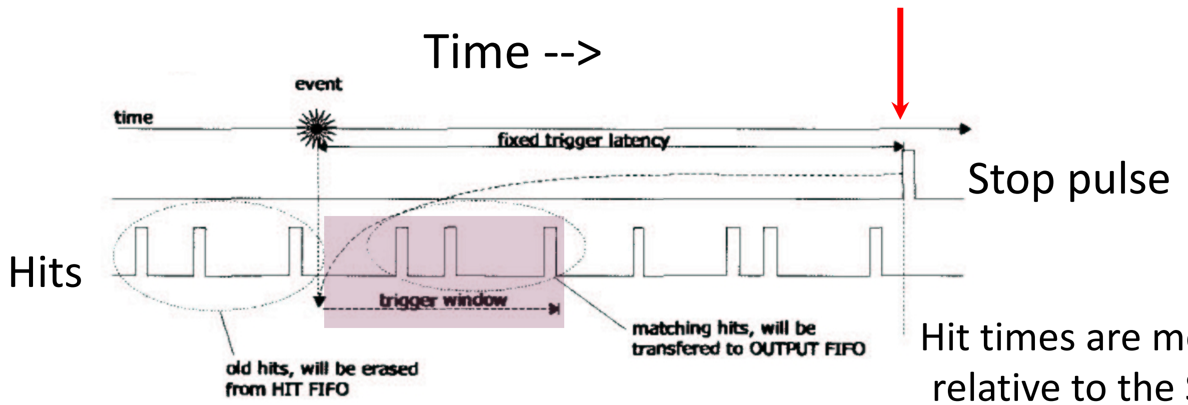
It is similar to an FPGA but is generally designed to a much more specific task. It is being used increasingly as an interface to the analog signals from the detectors.

JLAB – 64 channel Pipeline TDC

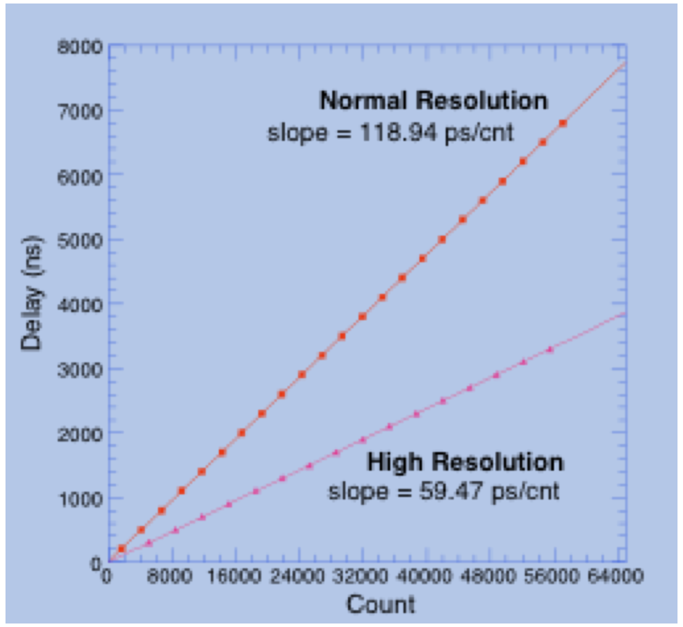


FPGA

TDC ASIC (8 channels)

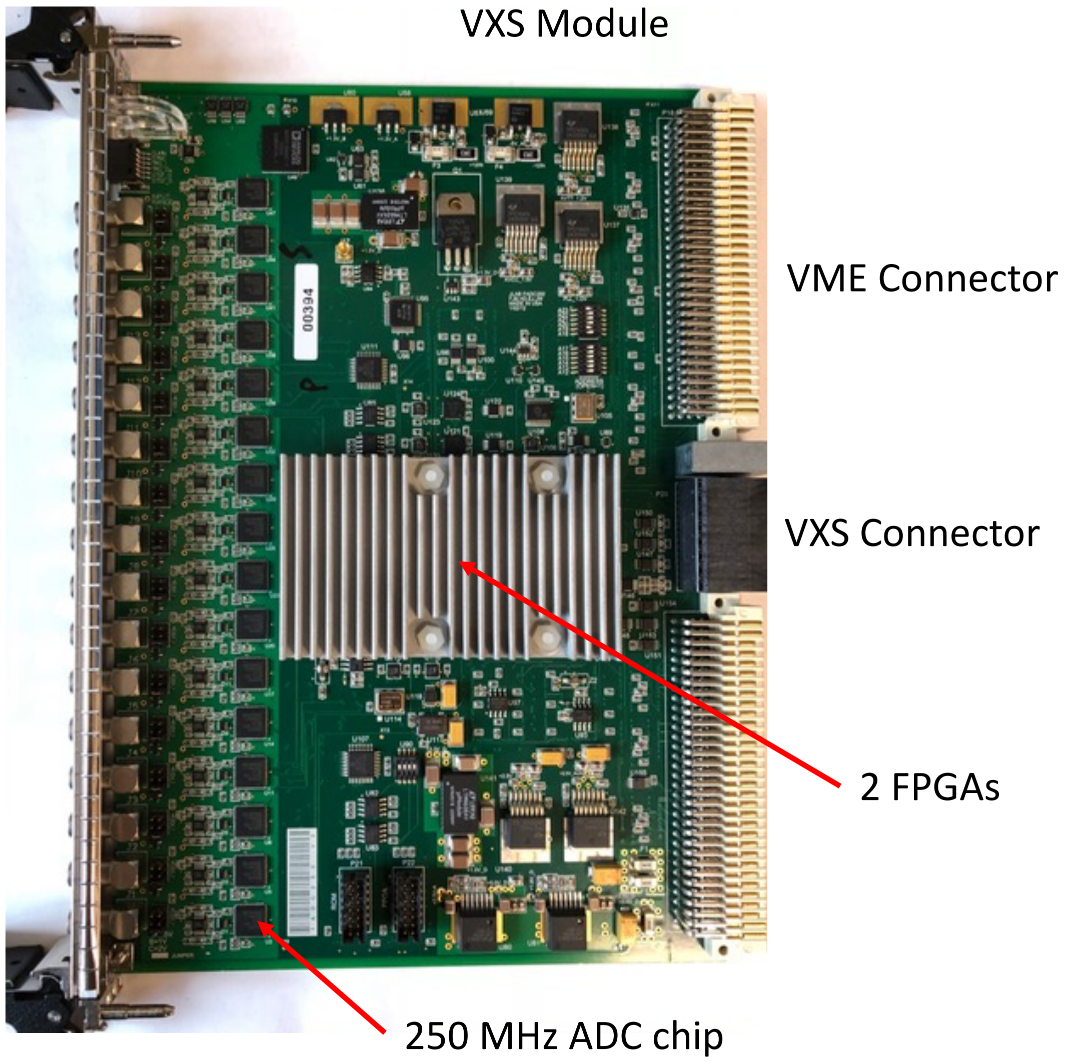

Time -->
event
time
fixed trigger latency
Stop pulse
Hits
trigger window
matching hits, will be transfered to OUTPUT FIFO
old hits, will be erased from HIT FIFO

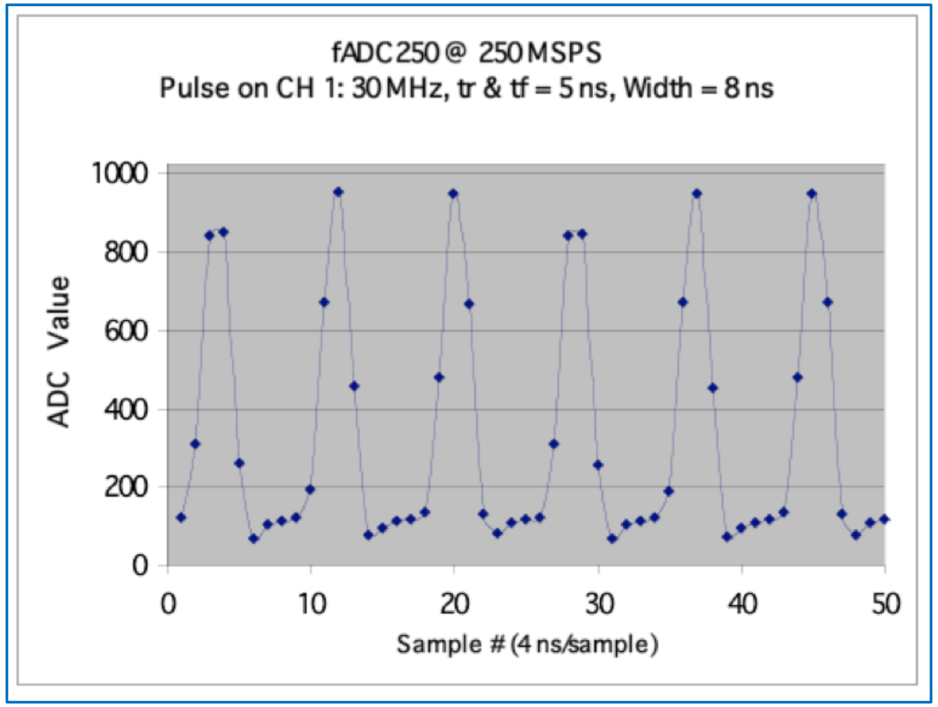
Hit times are measured relative to the Stop pulse (which can also be a Trigger)



Jefferson Lab

JLAB – 16 channel 250 MHz Flash ADC

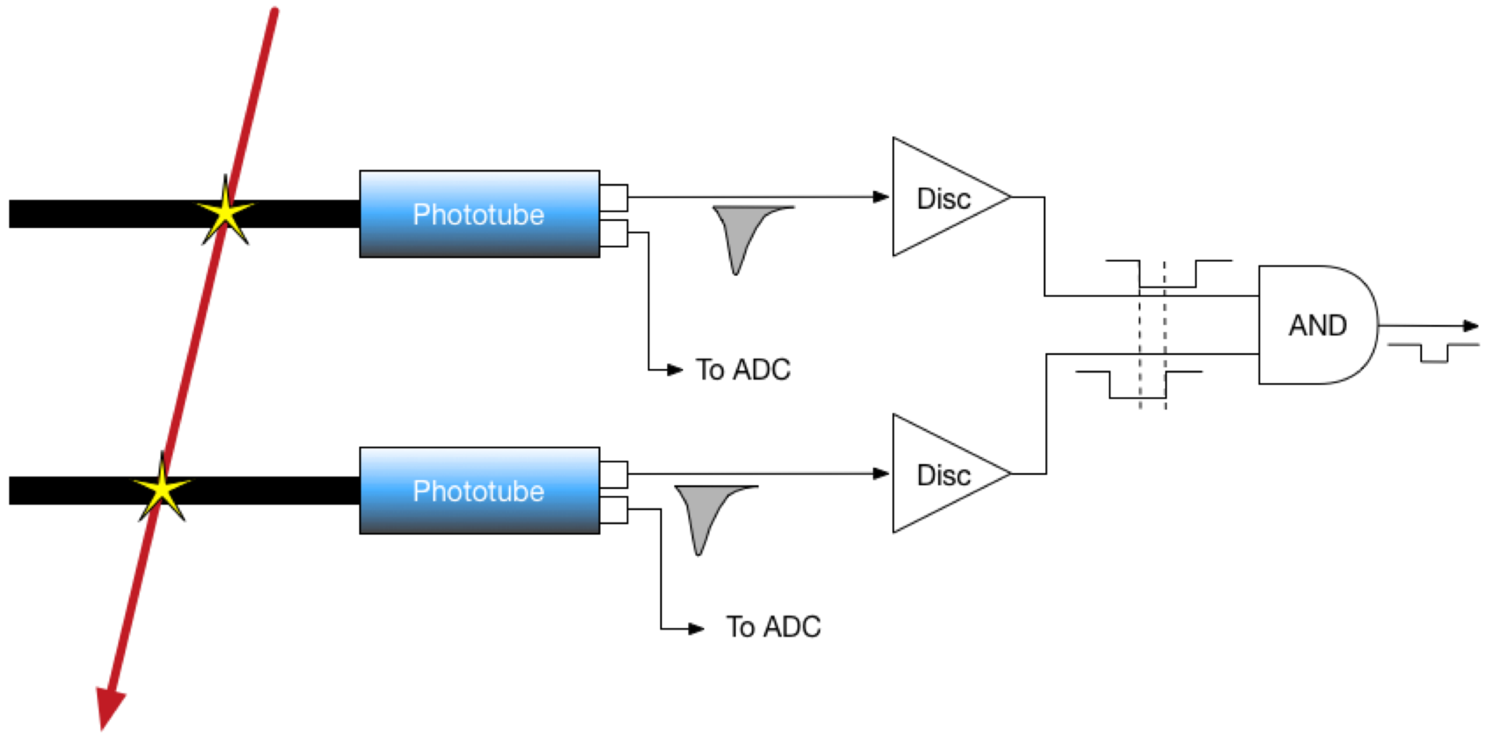




This module is currently used in all 4 Experimental Halls at JLAB. It is a core component to establish both Trigger Processing and Data Readout from the detectors.

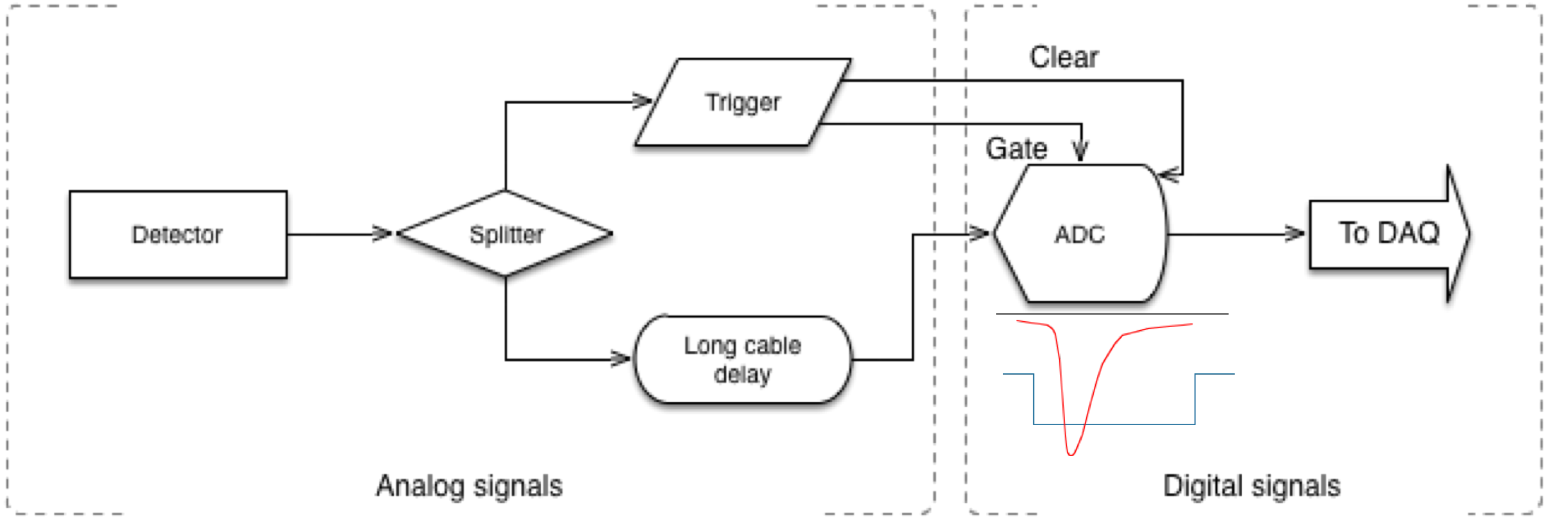
A Simple Trigger

- How do we know the detector signal came from an event?
  - Fortunately we have more than one detector.
  - Combine data from different detectors to characterize events.
  - Determine which events are interesting.
- Example - Coincidence trigger – two detectors have data within a time window.



This "trigger" can now be used to form a gate for the ADCs.

An analog trigger

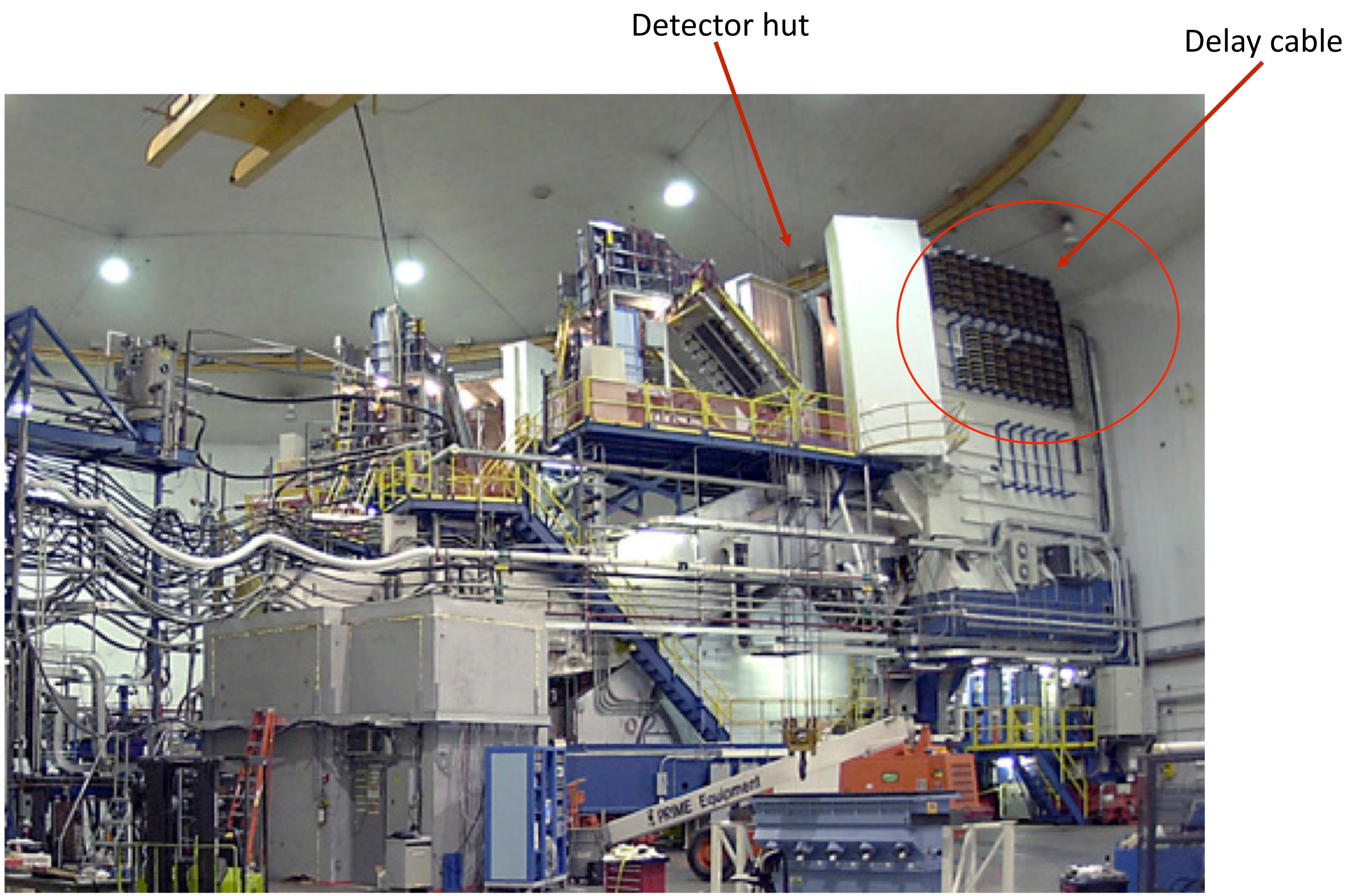


- It takes some time for the trigger logic to decide if a signal should be digitized.
- The analog signal must be delayed so that the gate and signal arrive at the ADC at the same time. Typical coax cables ~1 ns/ft so you could simply delay the signals using long cables.
  - Matching cable lengths is very important.
  - The ADC cannot process a new signal until it is read or cleared.
  - This limits the accepted trigger rate.

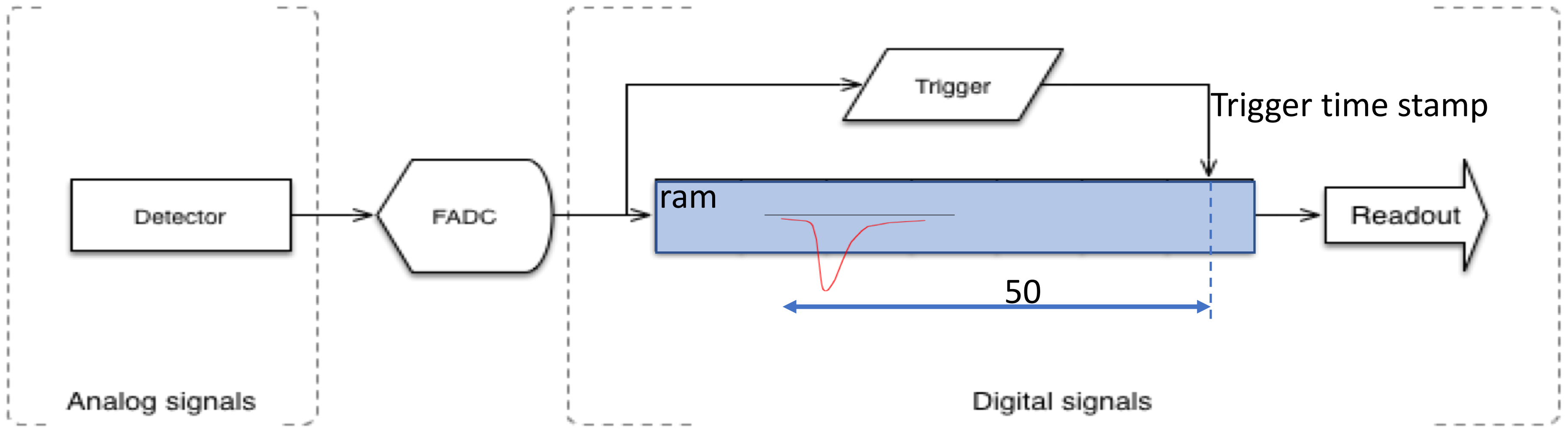
Here's the long cable in Hall-A.

When you have a 1000 channels and multiple logic stages for the trigger. You need A LOT of cable delay for the signals.

Where do you put it…



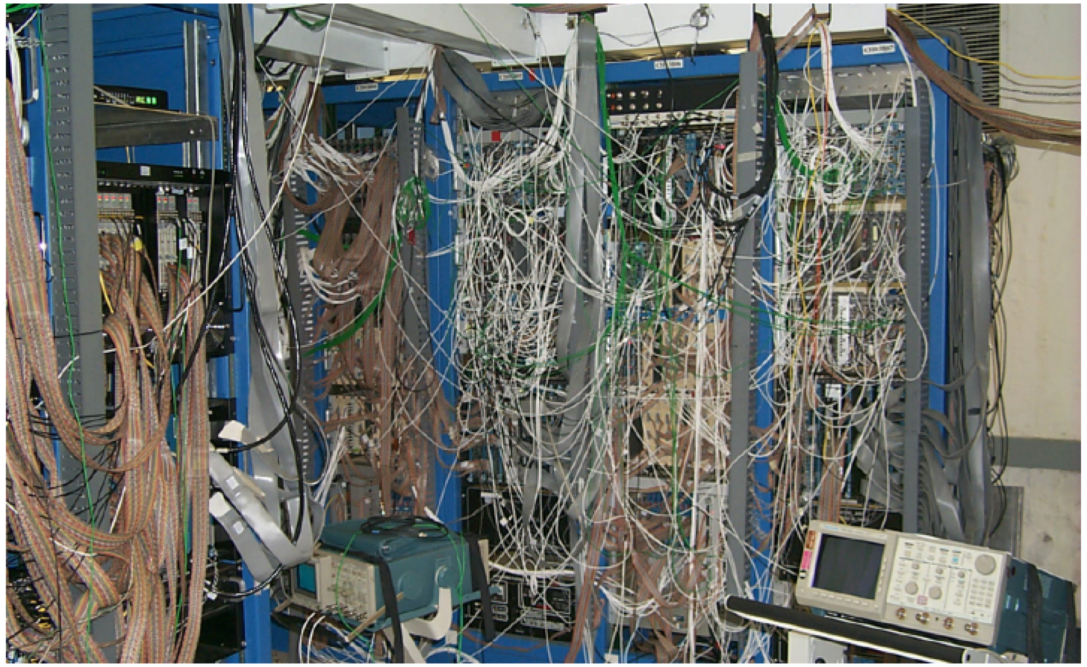
Pipeline trigger



- Replace all that cable with digital memory.
- In a pipelined system a Flash ADC digitizes at a constant rate and stores the values in a memory. Values are clocked into memory at the same rate as the Flash ADC clock which, in the case of the JLAB FADC, is 250MHz (4 nS).
- For example, if the trigger logic takes 200 nS we know that trigger corresponds to measurements 200/4 = 50 clocks down the memory pipeline.
- The readout software can Read all the samples or calculate the integrated charge (sum the samples) + save time of arrival.

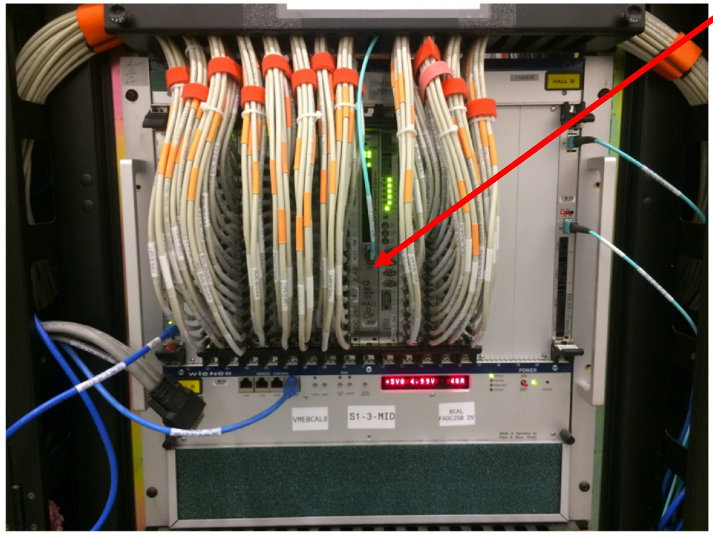
Trigger logic evolves

- Triggers used to use a lot of electronics wired together. We can't do that now:
  - Propagation times down cables limit trigger rates.
  - Modern experiments require very complex trigger decisions.

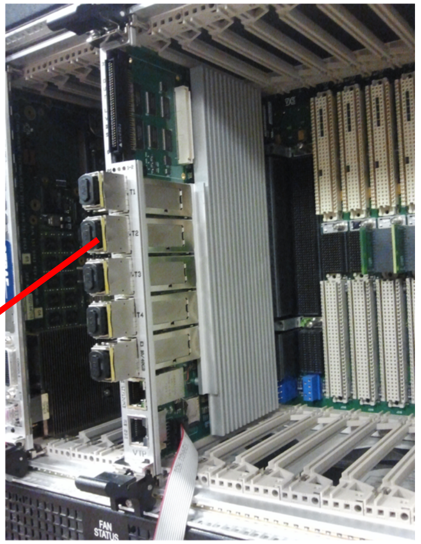

Hall C – Trigger processing - Circa 1994

Flash ADCs and FPGAs have replaced all the spaghetti cabling

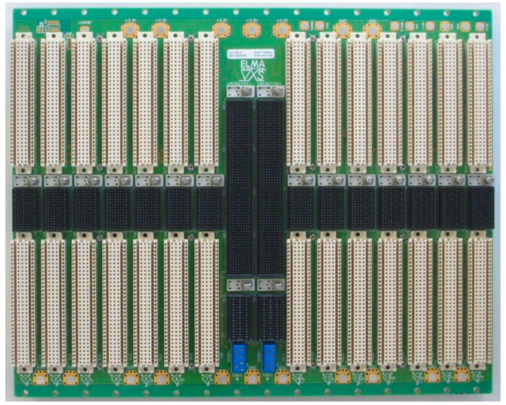
VXS supports both a VME parallel bus backplane as well as a high speed serial mesh connecting all payload slots to 2 switch slots


Hall D – FADC Crate with VXS Trigger Processor (VTP)

VTP



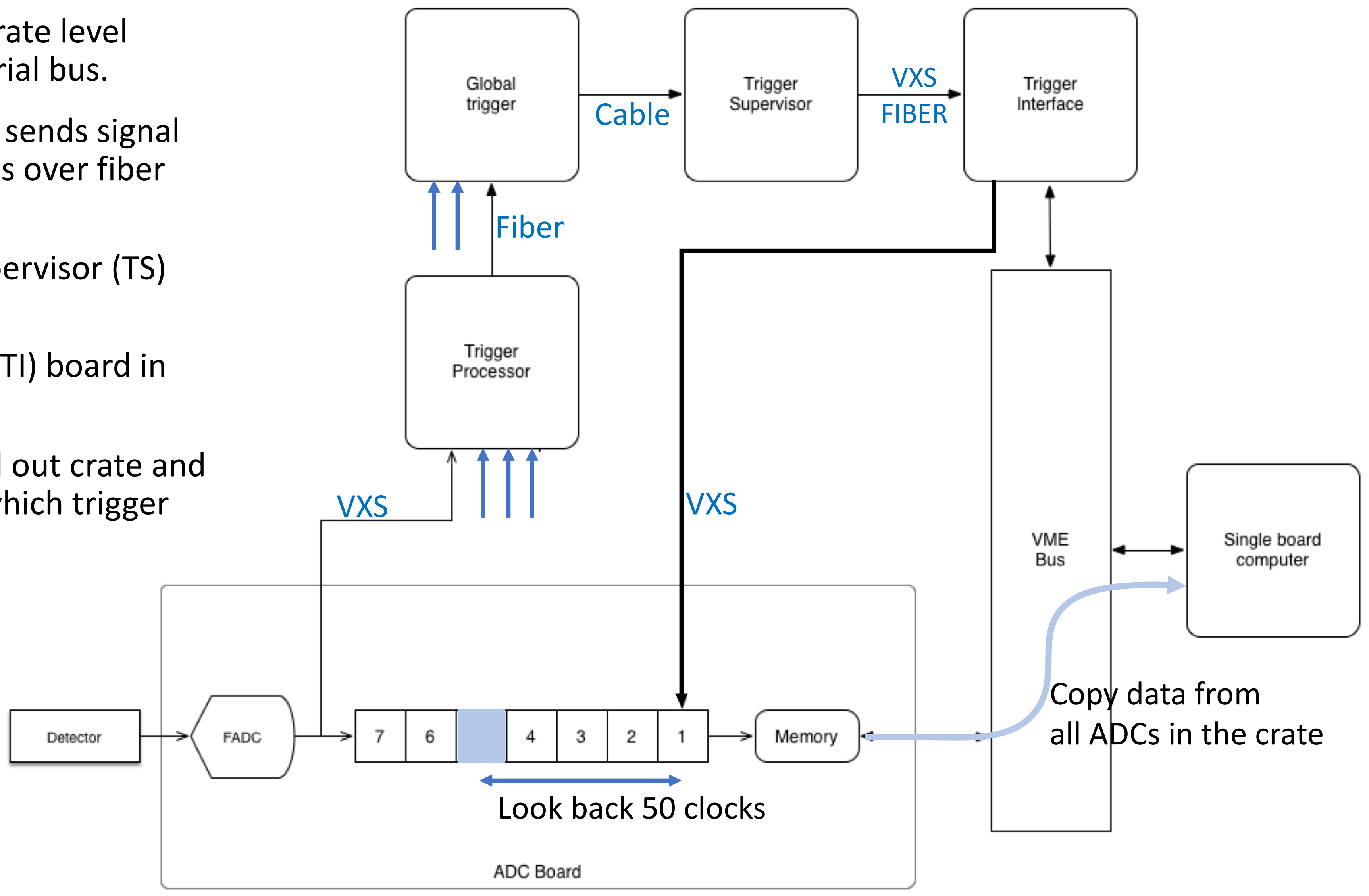
VXS Backplane



Jefferson Lab

Example – The GLUEX trigger

- Each ADC sends signals to a crate level trigger processor over VXS serial bus.
- Each CRATE Trigger processor sends signal to a global trigger for all crates over fiber optic cable.
- Global trigger tells trigger supervisor (TS) which events are good.
- The TS tells Trigger Interface (TI) board in each crate.
- The TI signals the CPU to read out crate and provides information about which trigger the data belongs to.



Jefferson Lab

GlueX trigger, starts at ADC crate

16 channels per board

15 boards = 240 channels per crate

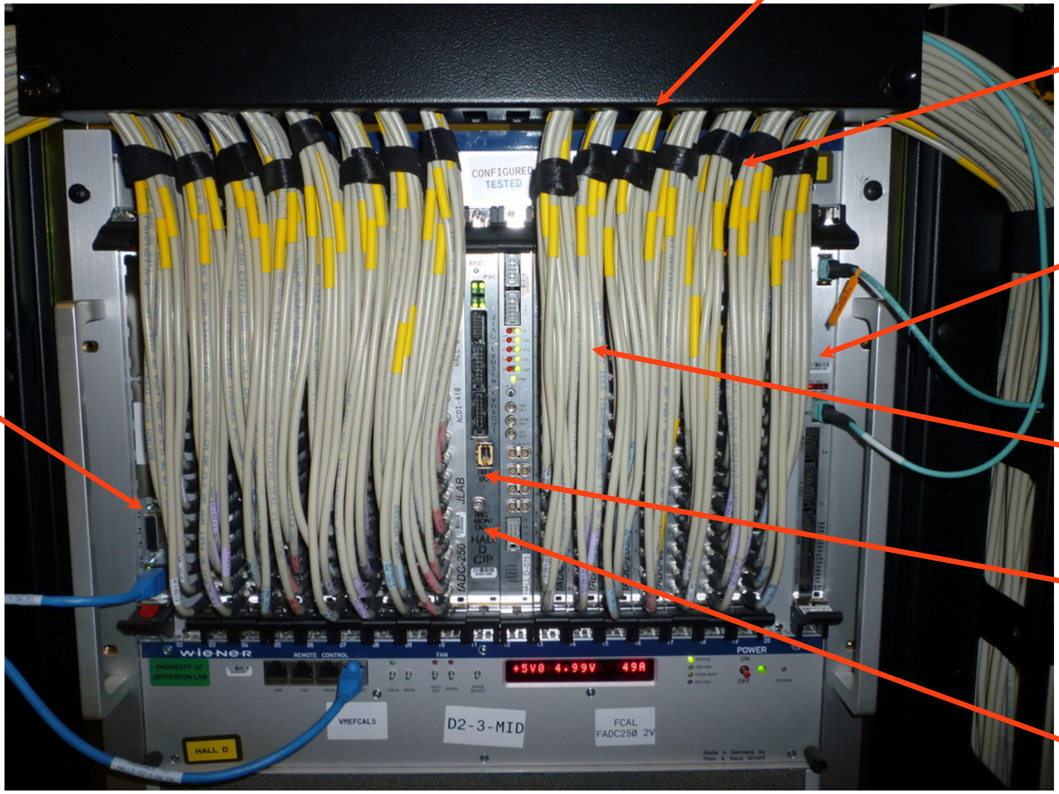
Trigger interface

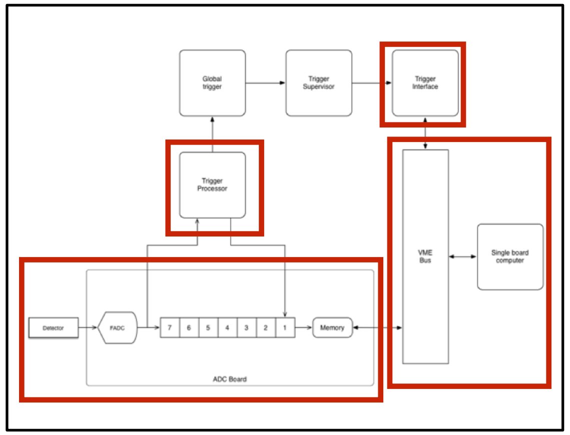
Intel CPU Read Out Controller (ROC) running Linux

ADC Boards are connected to CPU via the VME bus.

ADC trigger data over VXS serial bus

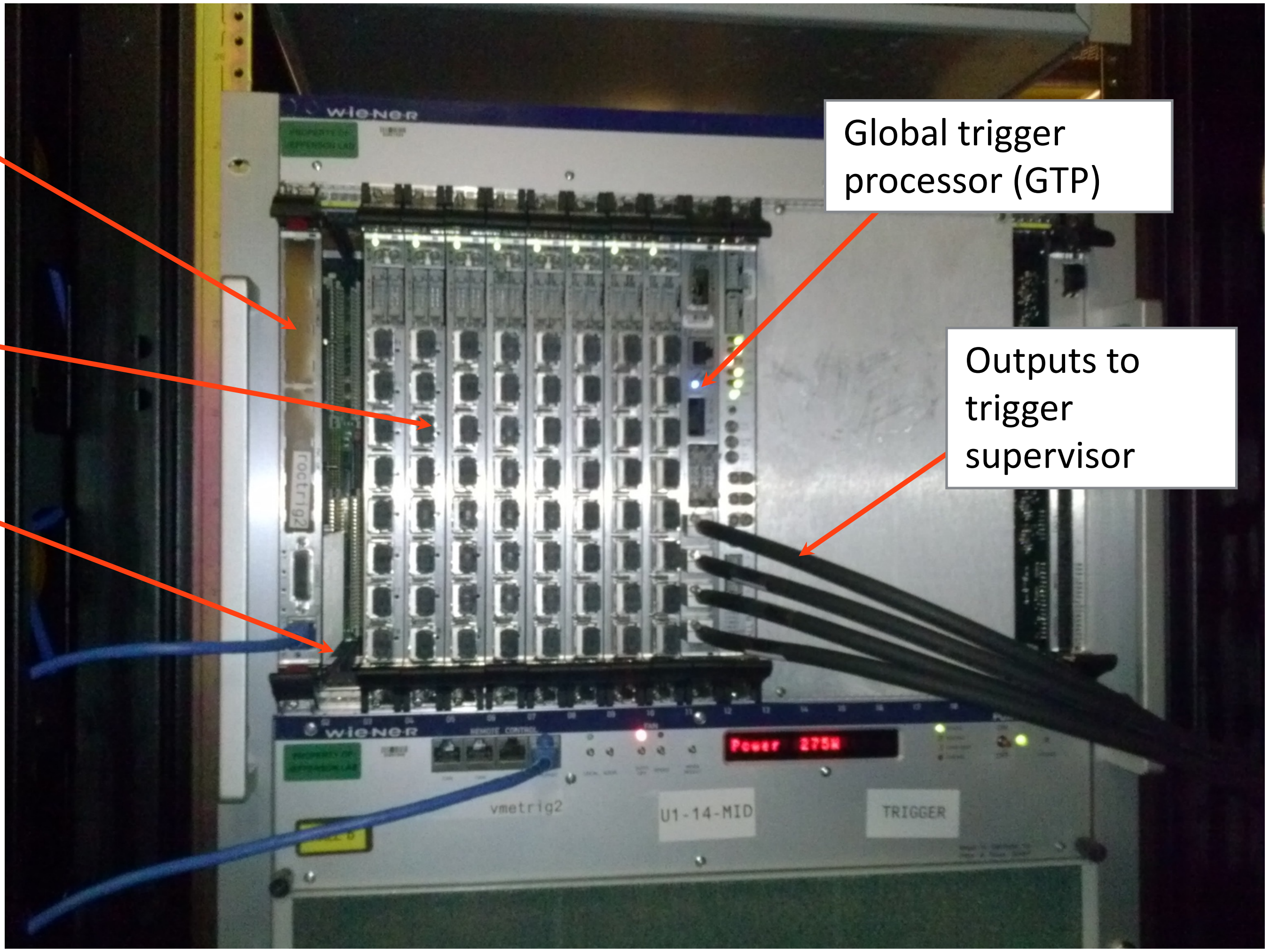
CTP Board sends local trigger info to the global trigger over fiber

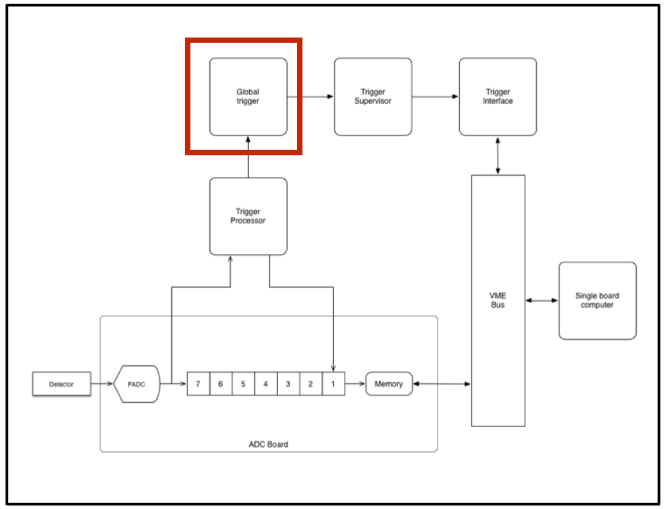




Jefferson Lab

All crates connect to the Global Trigger Crate


Intel CPU controller
Sub-system processor board (SSP)
Eight boards with eight connectors each so up to 64 crates.
Global trigger processor (GTP)
Outputs to trigger supervisor
wiener
Power 275W
vmetrig2
U1-14-MID
TRIGGER



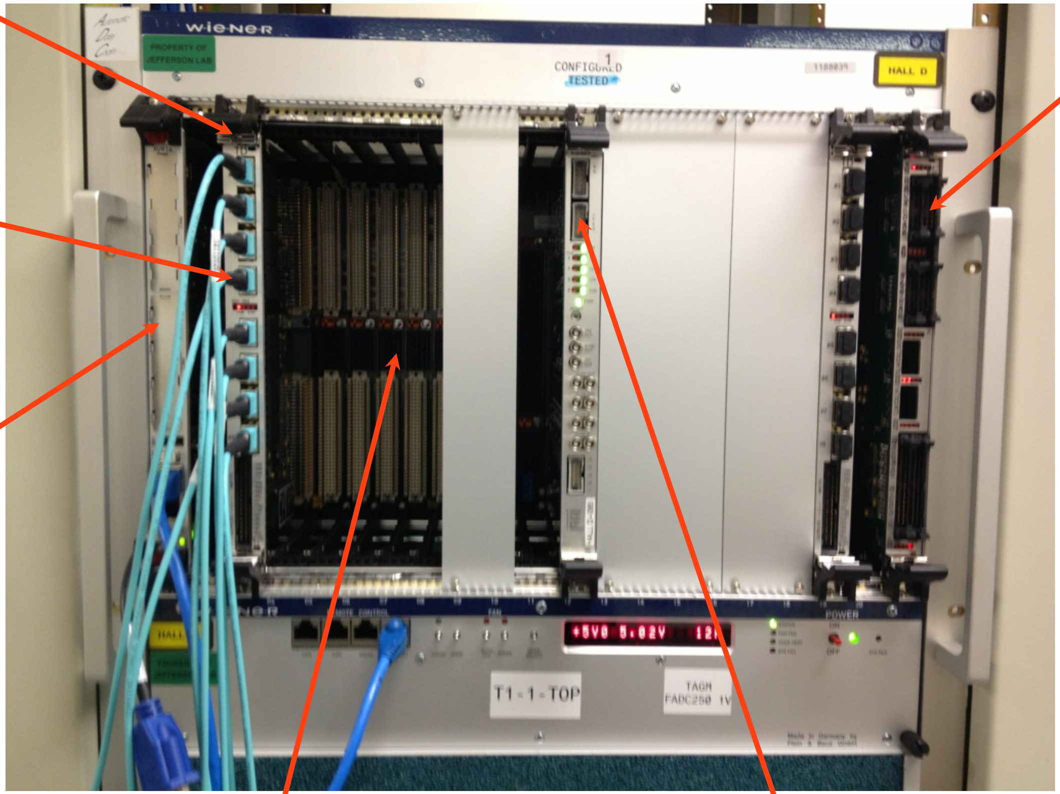
Final trigger goes to Trigger Supervisor crate

Trigger Distribution board (TD).

Optical trigger link back to crates.

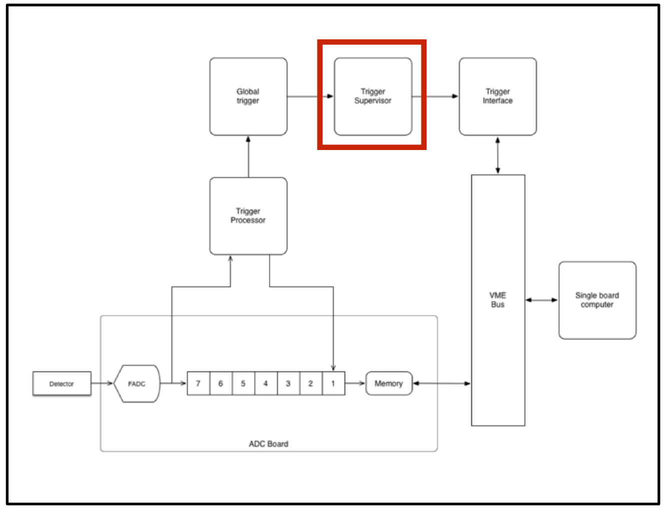
Intel CPU for control and configuration of the TD and TS.

Trigger Supervisor (TS)

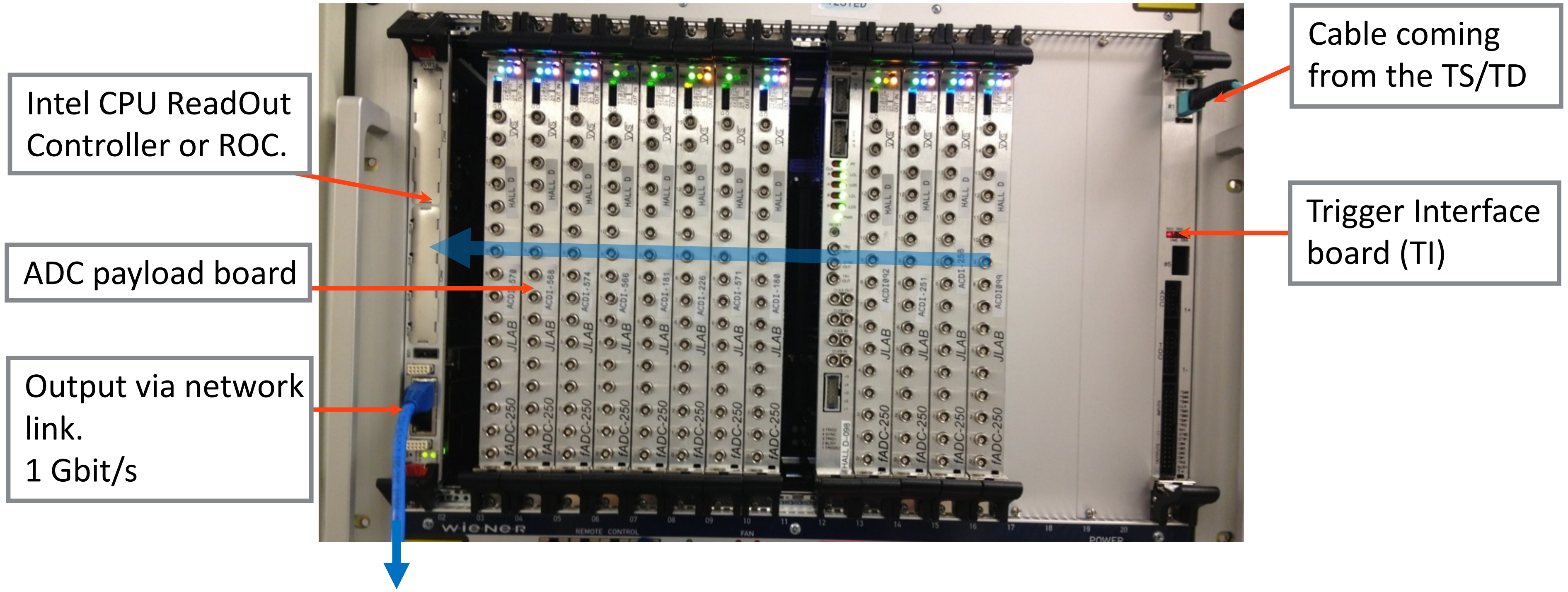


VXS serial backplane

Signal Distribution board (SD)



Then back to the start to trigger VME readout


Intel CPU ReadOut Controller or ROC.
ADC payload board
Output via network link.
1 Gbit/s
Cable coming from the TS/TD
Trigger Interface board (TI)

- The TI board gets from the trigger supervisor:
  - Signal to CPU to read the memory of the ADC boards.
  - TI is read to get data about which events the ADC data belong to.
- The CPU:
  - Copies ADC and TI data into memory via VME backplane,
  - CPU encodes data in EVIO data format.
  - Sends the data over the network to the rest of the data acquisition system.

CODA

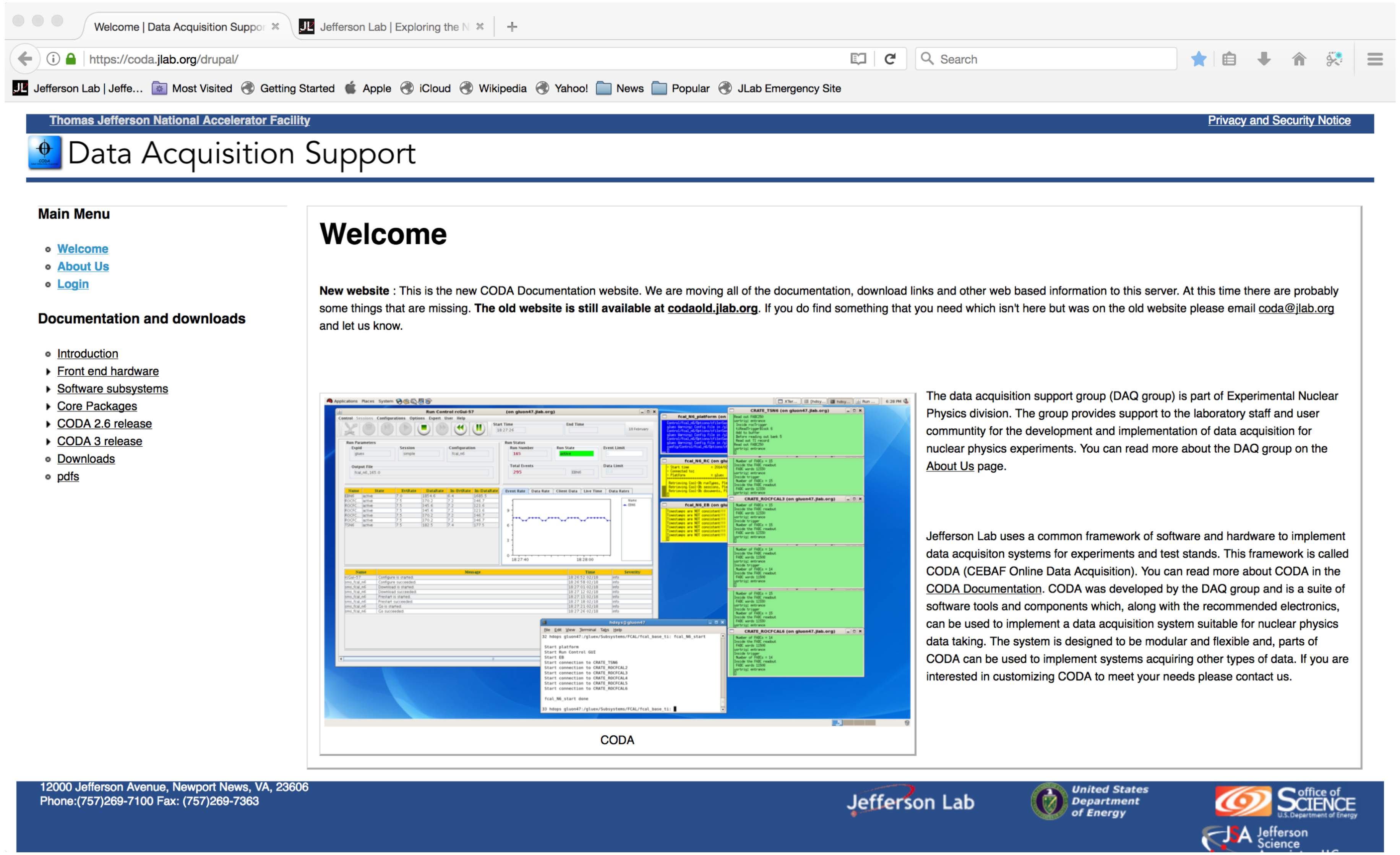
- What is CODA (also see coda.jlab.org)
  - Software toolkit for implementing data acquisition systems.
  - Hardware/Electronics
    - Custom boards like trigger, TDCs and ADCs.
    - Support for commercial hardware.
  - Software includes :
    - Interface with electronics (libraries/drivers).
    - Readout Front End and format data (ROC)
    - Inter-process communication - Control and Data (cMsg)
    - Merge data streams (DC, PEB, SEB)
    - Give users access to data for analysis and monitoring (ET System)
    - Write data to files (EVIO, ER)
    - Manage and control the data acquisition system (AFECS)

- CODA is modular. Build a single crate DAQ or a full Experimental Hall system

CODA – Support/Documentation

Web: https://coda.jlab.org

Fileserver: /u/site/coda



Thomas Jefferson National Accelerator Facility

Privacy and Security Notice

Data Acquisition Support

Main Menu

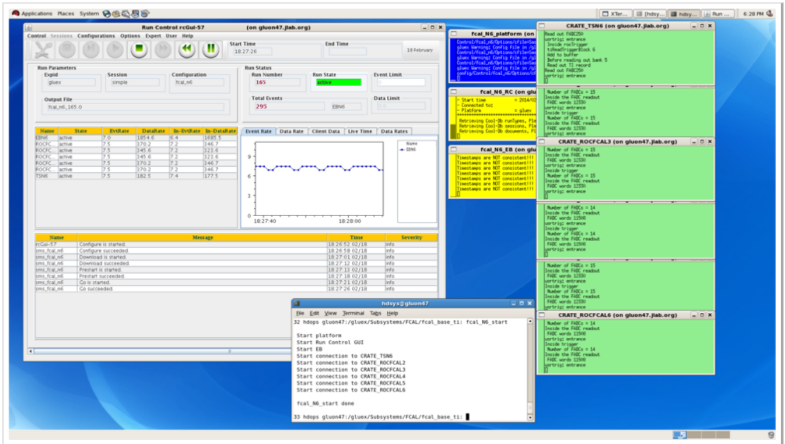
- Welcome
- About Us
- Login

Documentation and downloads

- Introduction
- Front end hardware
- Software subsystems
- Core Packages
- CODA 2.6 release
- CODA 3 release
- Downloads
- pdfs

Welcome

New website : This is the new CODA Documentation website. We are moving all of the documentation, download links and other web based information to this server. At this time there are probably some things that are missing. **The old website is still available at codaold.jlab.org**. If you do find something that you need which isn't here but was on the old website please email coda@jlab.org and let us know.



CODA

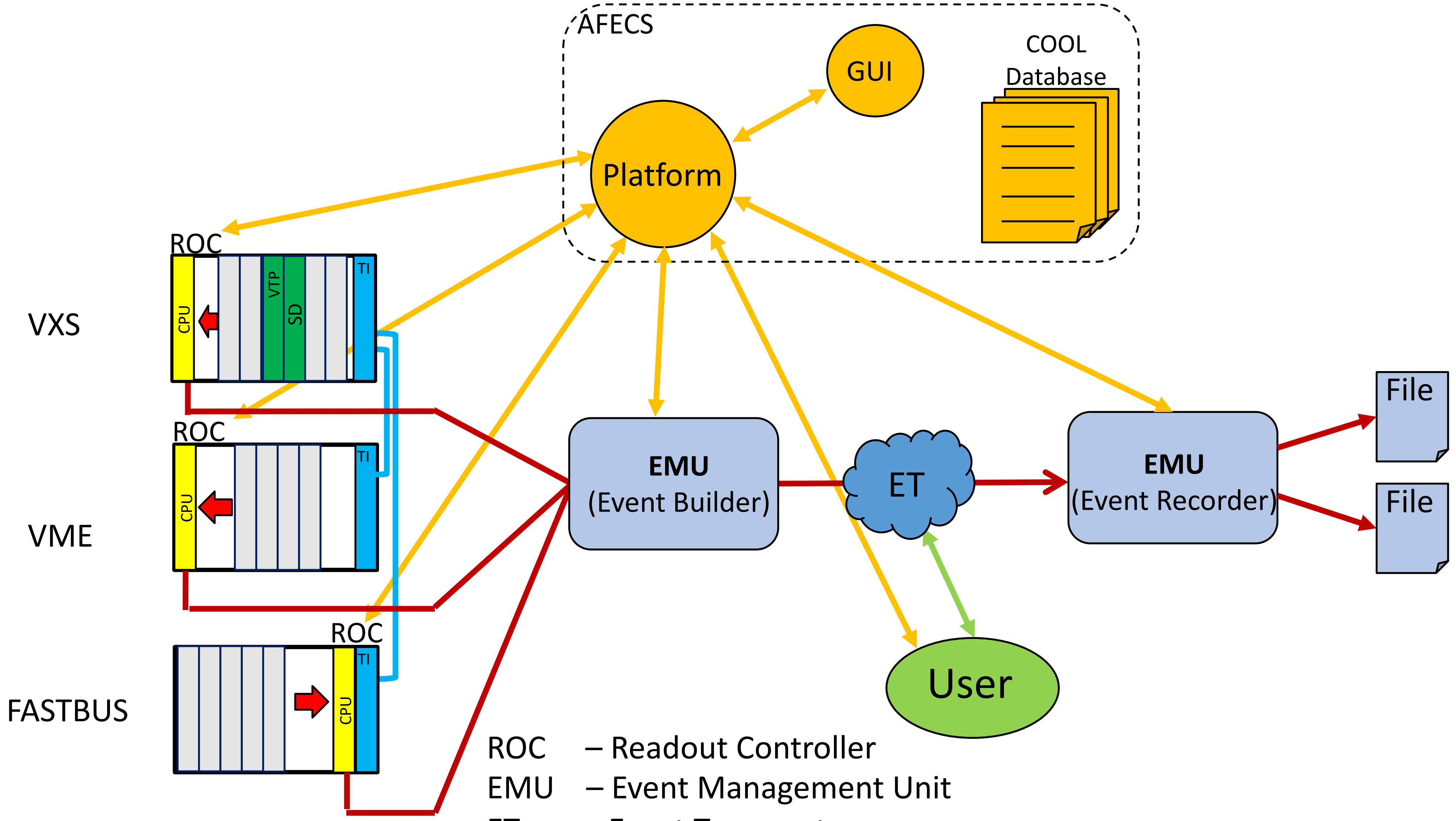
The data acquisition support group (DAQ group) is part of Experimental Nuclear Physics division. The group provides support to the laboratory staff and user community for the development and implementation of data acquisition for nuclear physics experiments. You can read more about the DAQ group on the About Us page.

Jefferson Lab uses a common framework of software and hardware to implement data acquisiton systems for experiments and test stands. This framework is called CODA (CEBAF Online Data Acquisition). You can read more about CODA in the CODA Documentation. CODA was developed by the DAQ group and is a suite of software tools and components which, along with the recommended electronics, can be used to implement a data acquisition system suitable for nuclear physics data taking. The system is designed to be modular and flexible and, parts of CODA can be used to implement systems acquiring other types of data. If you are interested in customizing CODA to meet your needs please contact us.

12000 Jefferson Avenue, Newport News, VA, 23606
Phone:(757)269-7100 Fax: (757)269-7363

Jefferson Lab
United States Department of Energy
Office of Science U.S. Department of Energy
JSA Jefferson Science

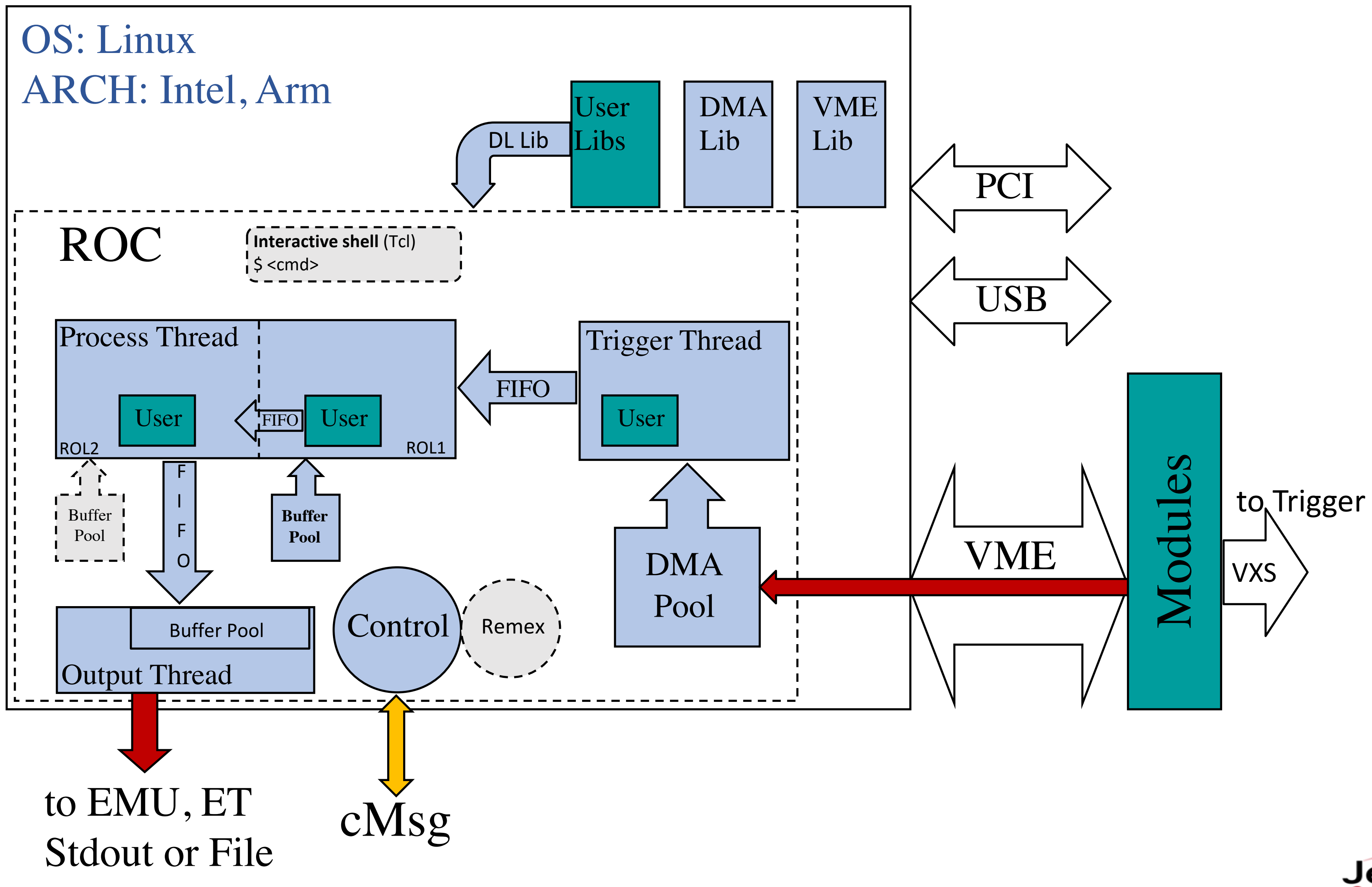
CODA 3



ROC – Readout Controller
EMU – Event Management Unit
ET – Event Transport
AFECS – Agent Framework Experiment Control System

Jefferson Lab

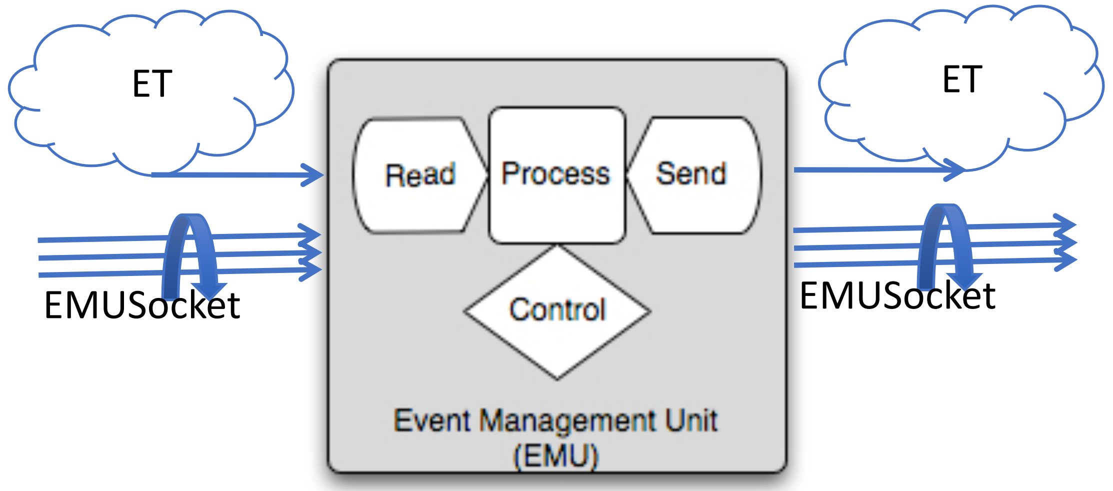
CODA Front-End – Readout Controller (ROC)



Jefferson Lab

CODA3 – The EMU Component

- EMU – Event Management Unit – is a JAVA-based general processing application for DAQ. It comes in many flavors:
    - DC – Data Concentrator
    - PEB – Primary Event Builder
    - SEB – Secondary Event Builder
    - ER – Event Recorder
    - FCS – Farm Control Supervisor
- Input/Output Connections made via an ET system or by EMUSocket protocol which is part of the CODA cMsg library (it allows for “fat” pipes on high bandwidth networks)



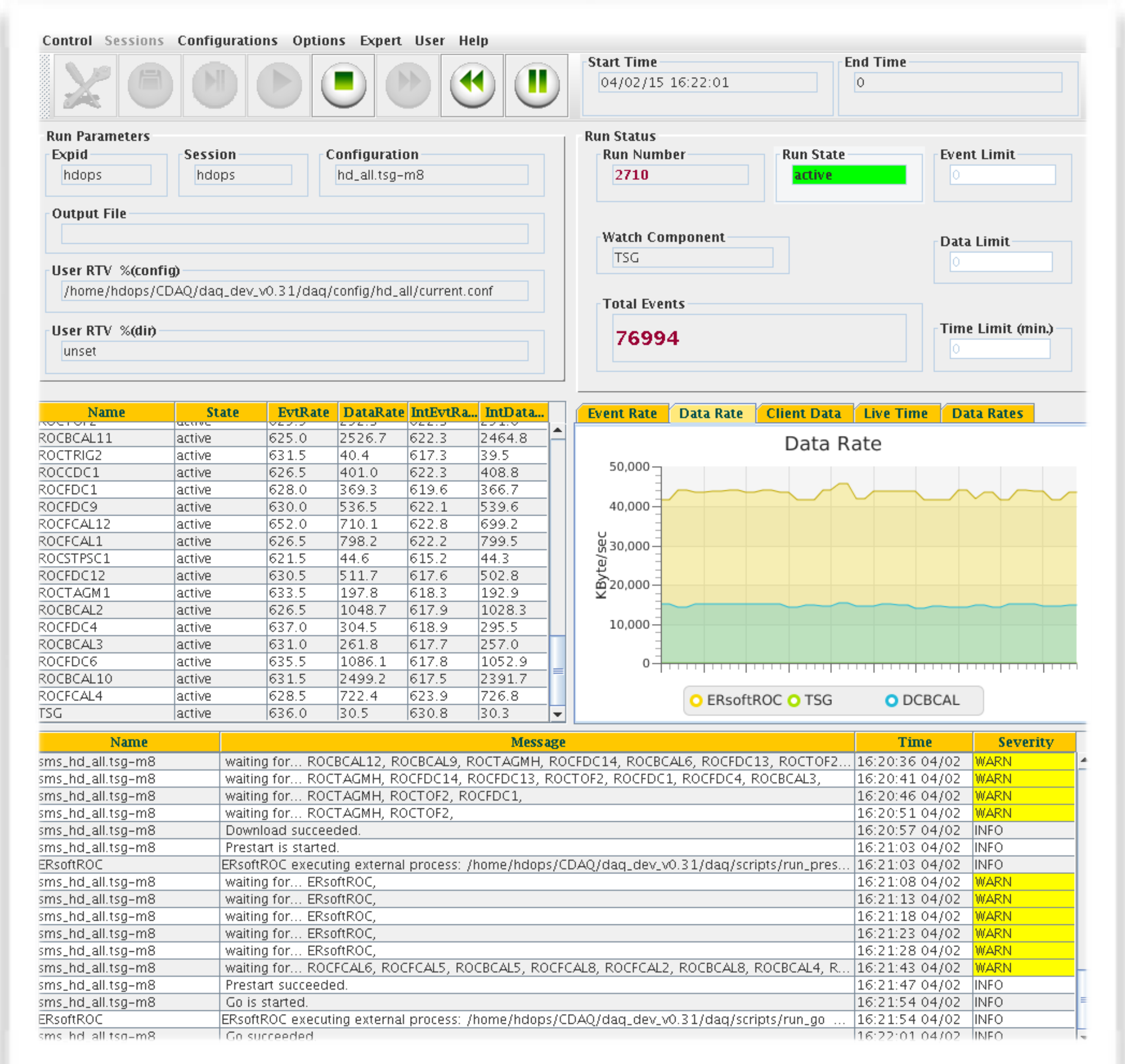
Jefferson Lab

Run Control - AFECS

The “platform” is a JAVA-based application running multiple “agents” that monitor and control external CODA client components (ROC, PEB, ER…) or internal processes (scripts). Multiple run configurations can also be running simultaneously.

Many “rcgui” processes can communicate with a single “platform” that is defined by a COOL Database with a Name = env(EXPID).

External commands can be used in scripts to communicate directly with the platform.



Start Time: 04/02/15 16:22:01
End Time: 0

Run Parameters
Expid: hdops
Session: hdops
Configuration: hd_all.tsg-m8
Output File:
User RTV %(config): /home/hdops/CDAQ/daq_dev_v0.31/daq/config/hd_all/current.conf
User RTV %(dir): unset

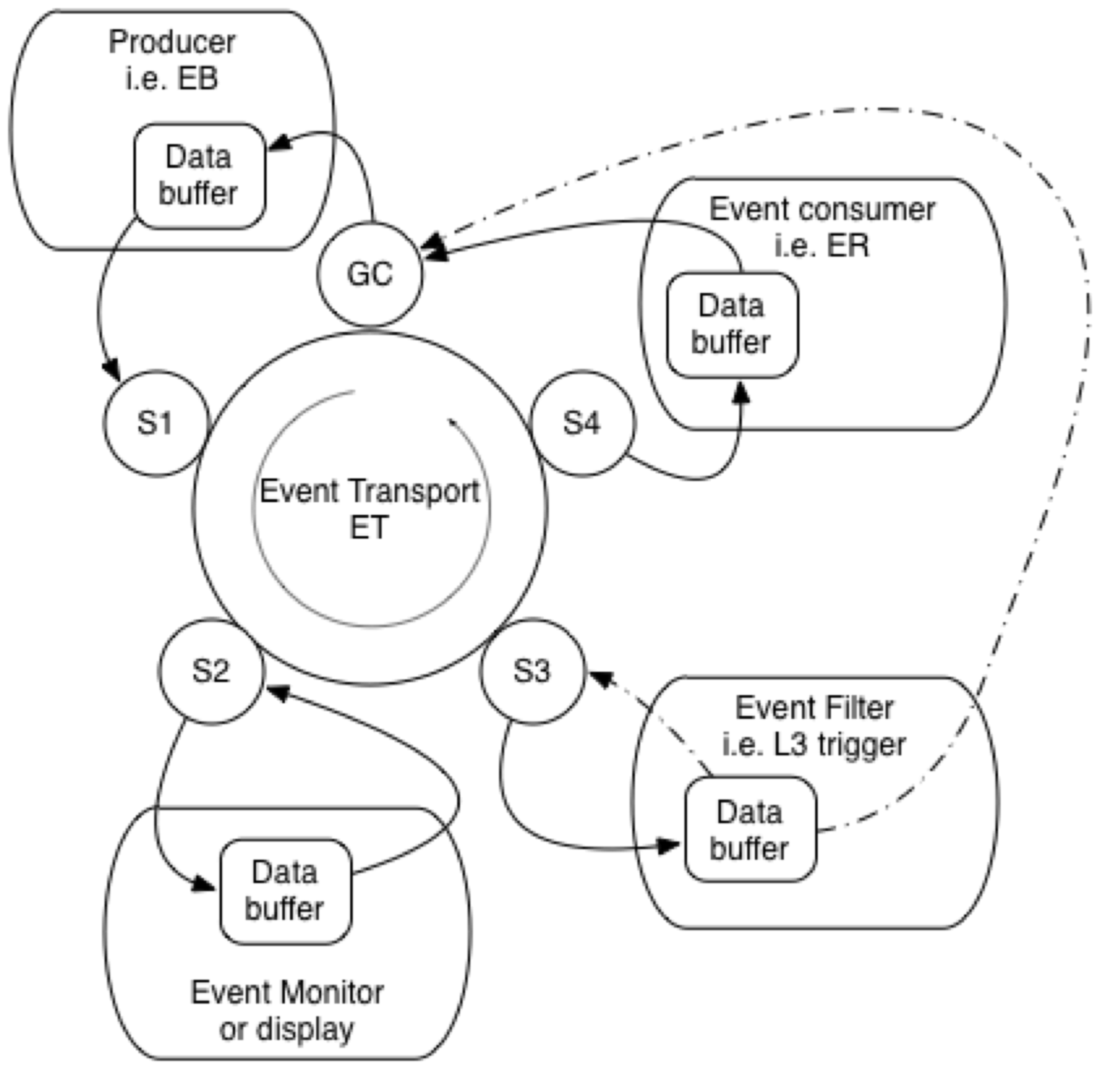
Run Status
Run Number: 2710
Run State: active
Event Limit: 0
Watch Component: TSG
Data Limit: 0
Total Events: 76994
Time Limit (min.): 0

Name	State	EvtRate	DataRate	IntEvtRa...	IntData...
ROCBCAL11	active	625.0	2526.7	622.3	2464.8
ROCTRIG2	active	631.5	40.4	617.3	39.5
ROCCDC1	active	626.5	401.0	622.3	408.8
ROCFDC1	active	628.0	369.3	619.6	366.7
ROCFDC9	active	630.0	536.5	622.1	539.6
ROCFCAL12	active	652.0	710.1	622.8	699.2
ROCFCAL1	active	626.5	798.2	622.2	799.5
ROCSTPSC1	active	621.5	44.6	615.2	44.3
ROCFDC12	active	630.5	511.7	617.6	502.8
ROCTAGM1	active	633.5	197.8	618.3	192.9
ROCBCAL2	active	626.5	1048.7	617.9	1028.3
ROCFDC4	active	637.0	304.5	618.9	295.5
ROCBCAL3	active	631.0	261.8	617.7	257.0
ROCFDC6	active	635.5	1086.1	617.8	1052.9
ROCBCAL10	active	631.5	2499.2	617.5	2391.7
ROCFCAL4	active	628.5	722.4	623.9	726.8
TSG	active	636.0	30.5	630.8	30.3

Name	Message	Time	Severity
sms_hd_all.tsg-m8	waiting for... ROCBCAL12, ROCBCAL9, ROCTAGMH, ROCFDC14, ROCBCAL6, ROCFDC13, ROCTOF2...	16:20:36 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ROCTAGMH, ROCFDC14, ROCFDC13, ROCTOF2, ROCFDC1, ROCFDC4, ROCBCAL3,	16:20:41 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ROCTAGMH, ROCTOF2, ROCFDC1,	16:20:46 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ROCTAGMH, ROCTOF2,	16:20:51 04/02	WARN
sms_hd_all.tsg-m8	Download succeeded.	16:20:57 04/02	INFO
sms_hd_all.tsg-m8	Prestart is started.	16:21:03 04/02	INFO
ERsoftROC	ERsoftROC executing external process: /home/hdops/CDAQ/daq_dev_v0.31/daq/scripts/run_pres...	16:21:03 04/02	INFO
sms_hd_all.tsg-m8	waiting for... ERsoftROC,	16:21:08 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ERsoftROC,	16:21:13 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ERsoftROC,	16:21:18 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ERsoftROC,	16:21:23 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ERsoftROC,	16:21:28 04/02	WARN
sms_hd_all.tsg-m8	waiting for... ROCFCAL6, ROCFCAL5, ROCBCAL5, ROCFCAL8, ROCFCAL2, ROCBCAL8, ROCBCAL4, R...	16:21:43 04/02	WARN
sms_hd_all.tsg-m8	Prestart succeeded.	16:21:47 04/02	INFO
sms_hd_all.tsg-m8	Go is started.	16:21:54 04/02	INFO
ERsoftROC	ERsoftROC executing external process: /home/hdops/CDAQ/daq_dev_v0.31/daq/scripts/run_go ...	16:21:54 04/02	INFO
sms_hd_all.tsg-m8	Go succeeded.	16:22:01 04/02	INFO

Event Transport, ET

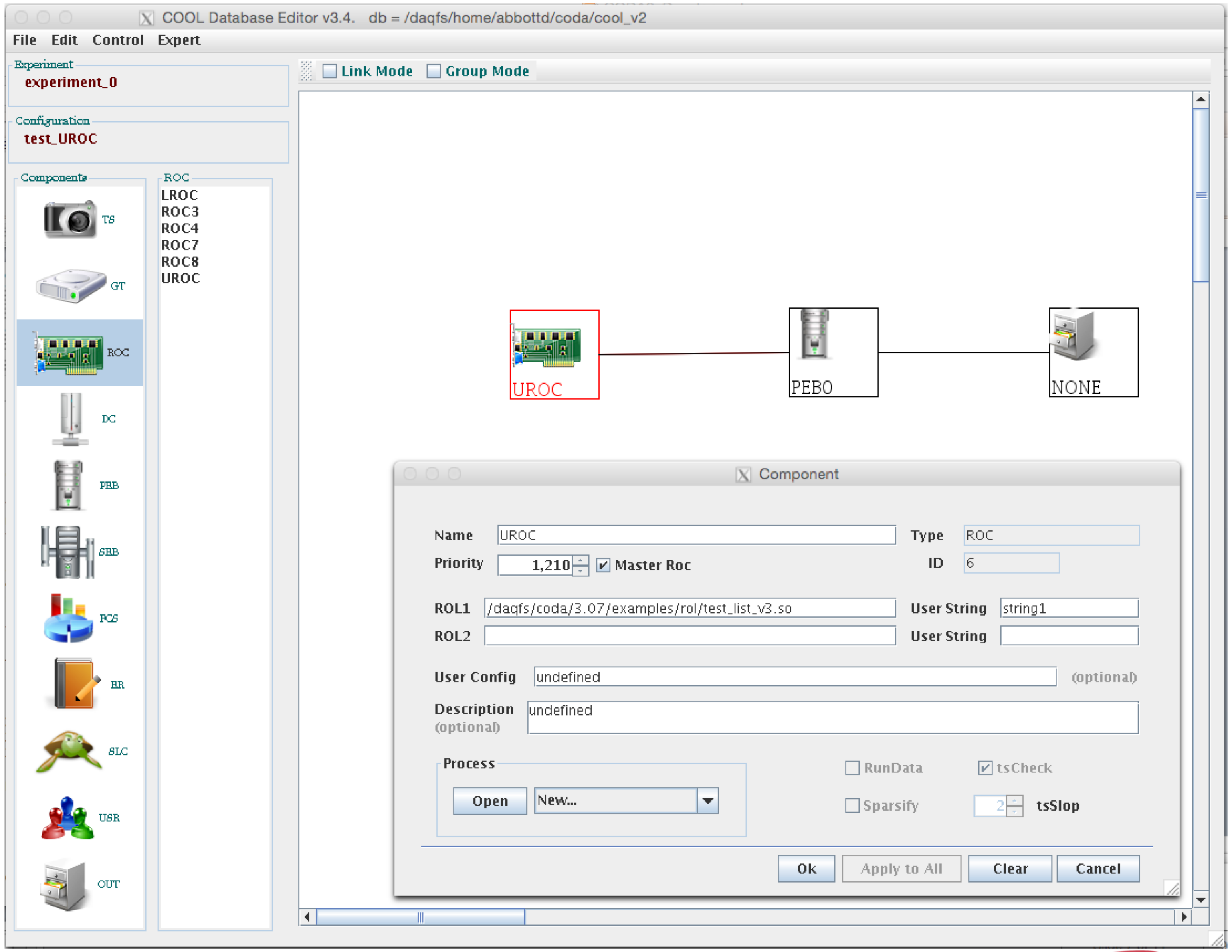
- Allocating and freeing buffers is time consuming.
- The ET system gives programs access to data via preallocated shared buffers.
- The system uses a railroad metaphor. Empty data buffers originate at Grand Central. They are filled by data producers and tagged to describe the content.
- The buffers “move” around a circular track and at each station the tag is checked to see if the buffer should stop at the station.
- An event monitor could set up station, S2, to take 1% of the events.
- An event filter could set up S3 to take all events. Discarded events are sent back to GC good ones move on.
- An event recorder takes all events and, after the data is written to a file sends the buffer back to GC.



Configuration editor

The program jcedit allows the User to graphically create different DAQ configurations, defining the components, data links, data files and other details.

Here is a simple example for one ROC connected to an Event Builder and the Event Builder is then "throwing the data away"



COOL Database Editor v3.4. db = /daqfs/home/abbottd/coda/cool_v2

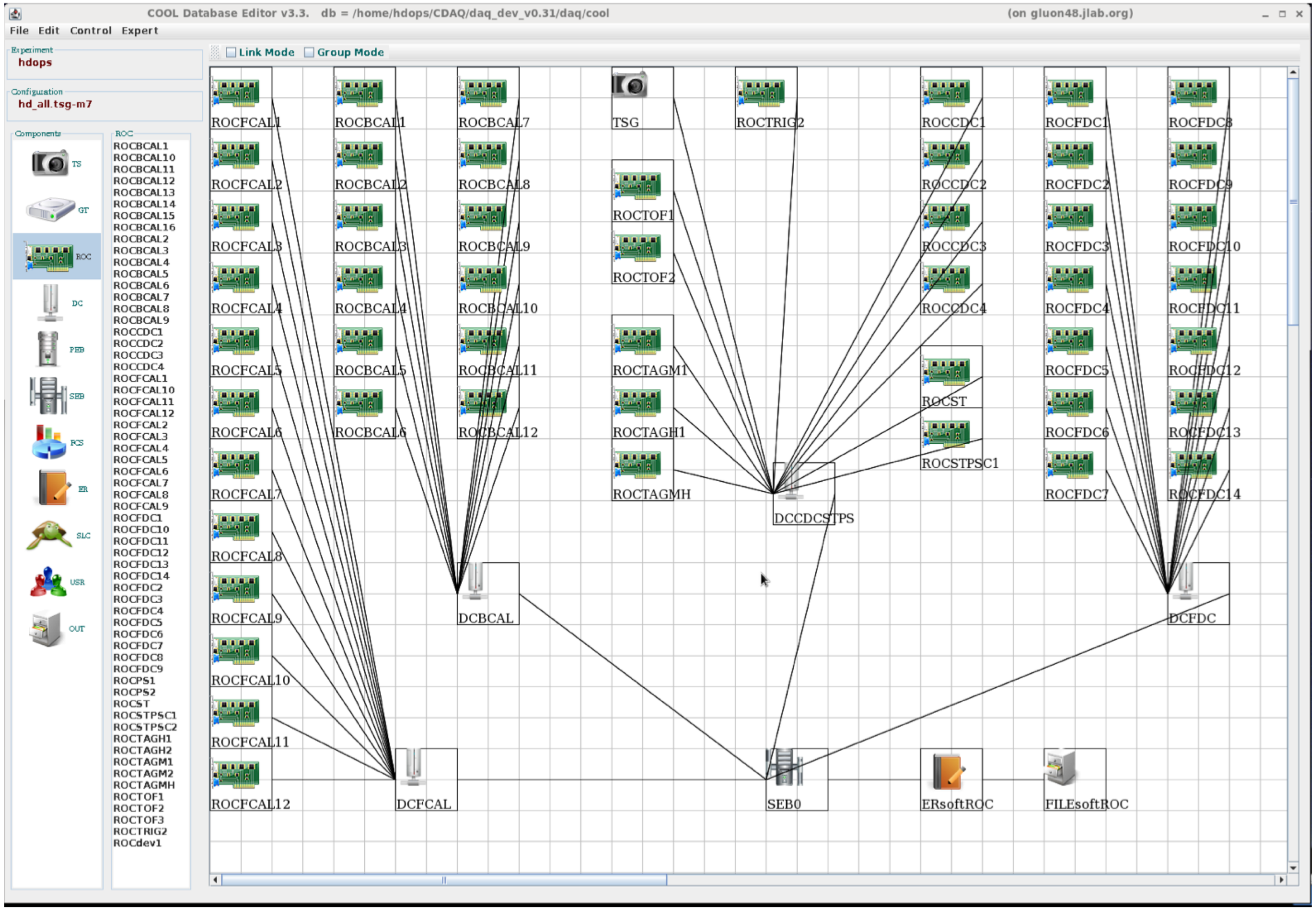
Experiment: experiment_0

Configuration: test_UROC

ROC:
LROC
ROC3
ROC4
ROC7
ROC8
UROC

Name: UROC
Type: ROC
Priority: 1,210 [✓] Master Roc
ID: 6
ROL1: /daqfs/coda/3.07/examples/rol/test_list_v3.so
User String: string1
ROL2:
User String:
User Config: undefined (optional)
Description (optional): undefined
Process: New...
[ ] RunData [✓] tsCheck
[ ] Sparsify 2 tsSlop

GLUEX configuration:
50 ROCS, 4 Data Concentrators, 1 Secondary EB, 1 Event Recorder.



COOL Database Editor v3.3. db = /home/hdops/CDAQ/daq_dev_v0.31/daq/cool (on gluon48.jlab.org)

File Edit Control Expert

Experiment: hdops

Configuration: hd_all.tsg-m7

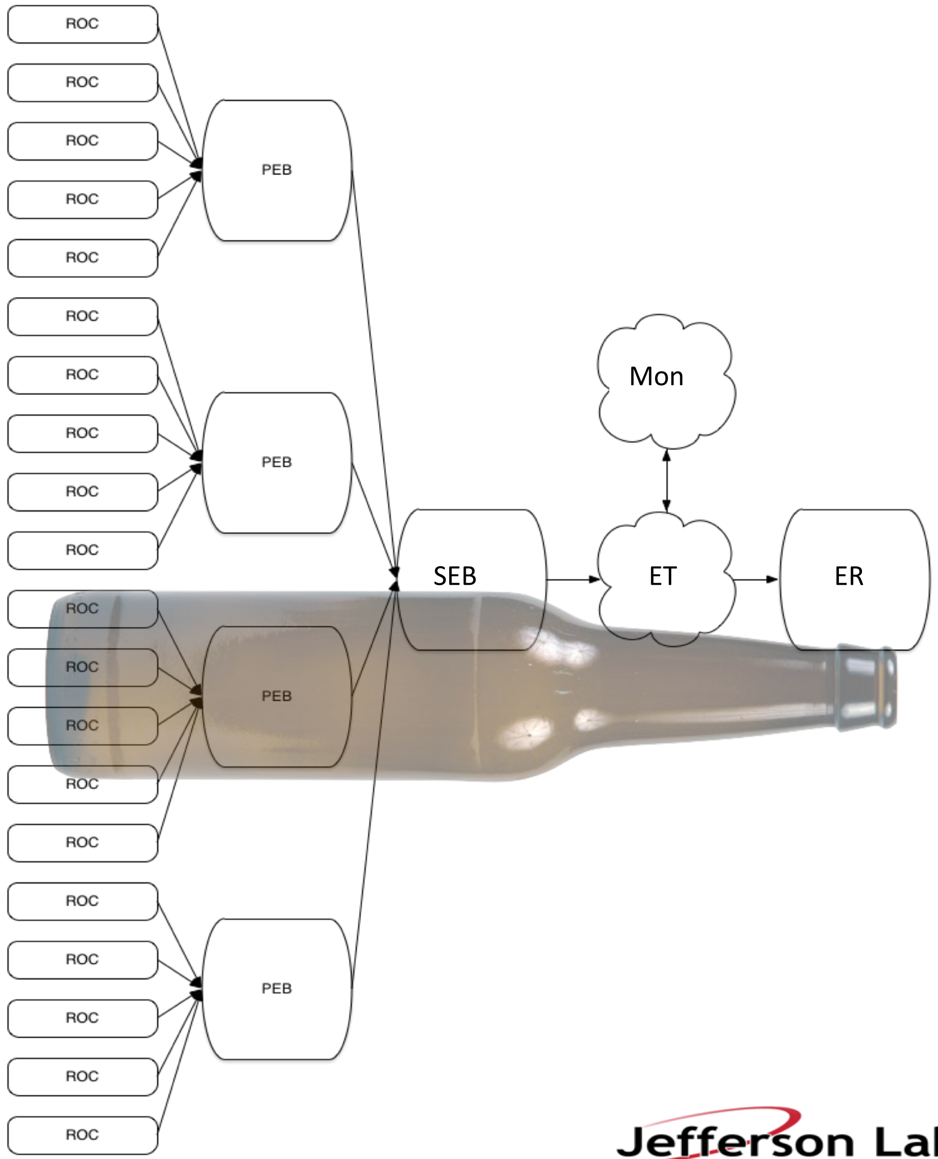
ROC

```
ROCBCAL1
ROCBCAL10
ROCBCAL11
ROCBCAL12
ROCBCAL13
ROCBCAL14
ROCBCAL15
ROCBCAL16
ROCBCAL2
ROCBCAL3
ROCBCAL4
ROCBCAL5
ROCBCAL6
ROCBCAL7
ROCBCAL8
ROCBCAL9
ROCCDC1
ROCCDC2
ROCCDC3
ROCCDC4
ROCFCAL1
ROCFCAL10
ROCFCAL11
ROCFCAL12
ROCFCAL2
ROCFCAL3
ROCFCAL4
ROCFCAL5
ROCFCAL6
ROCFCAL7
ROCFCAL8
ROCFCAL9
ROCFDC1
ROCFDC10
ROCFDC11
ROCFDC12
ROCFDC13
ROCFDC14
ROCFDC2
ROCFDC3
ROCFDC4
ROCFDC5
ROCFDC6
ROCFDC7
ROCFDC8
ROCFDC9
ROCPS1
ROCPS2
ROCST
ROCSTPSC1
ROCSTPSC2
ROCTAGH1
ROCTAGH2
ROCTAGM1
ROCTAGM2
ROCTAGMH
ROCTOF1
ROCTOF2
ROCTOF3
ROCTRIG2
ROCdev1
```

Jefferson Lab

Event Building challenges

- The GlueX event rate is up to 90 kHz and there are 50+ crates.
- The GLUEX goal data rate is 1.5 GByte/s.
  - 30 MByte/s average per each of the 50 incoming links.
  - 1.5 GByte/s through the EB, ET and ER.
  - Since data is copied several times the data rate inside machine running EB is several times 1.5 GByte/s.
- This is what is commonly called the bottleneck effect. Even if the network bandwidth is sufficient. It could be a lot for one machine to handle.
  - Solution: multi stage parallel event builder.



CODA3 Example Parallel Config

If all the Front-End (ROCs) generate too much data in aggregate - make the event building parallel.

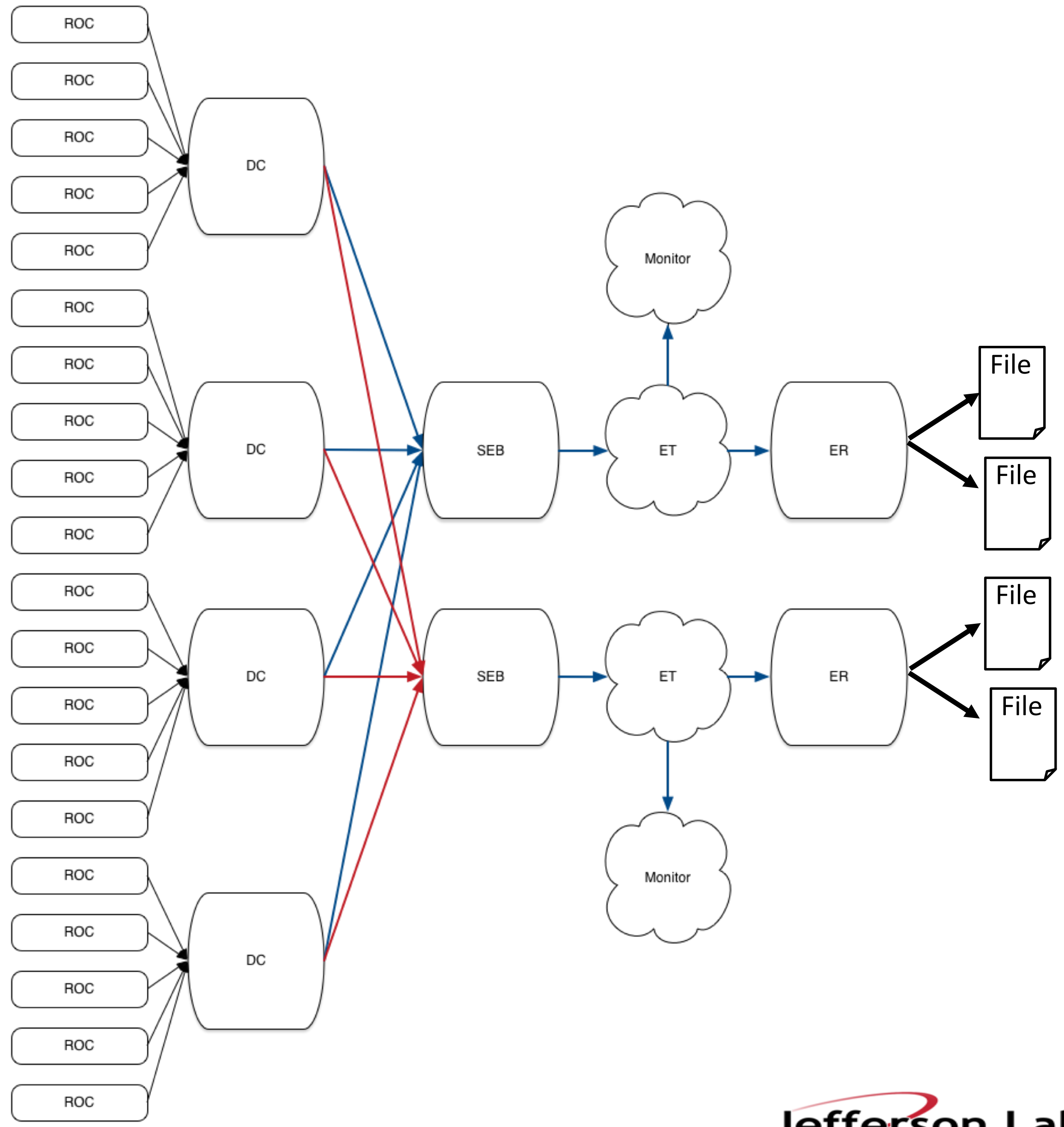
Two Stage Event Building
2 Parallel Data Streams
(each writes half the data)

Front-End ROCs send to
4 Data Concentrators
2 Secondary Event Builders
2 Event Recorders
(with ET systems for Inputs)

By default data is routed from DCs to SEBs equally with alternating event blocks.

Control Events go to both streams.

User events will only go to the first stream.

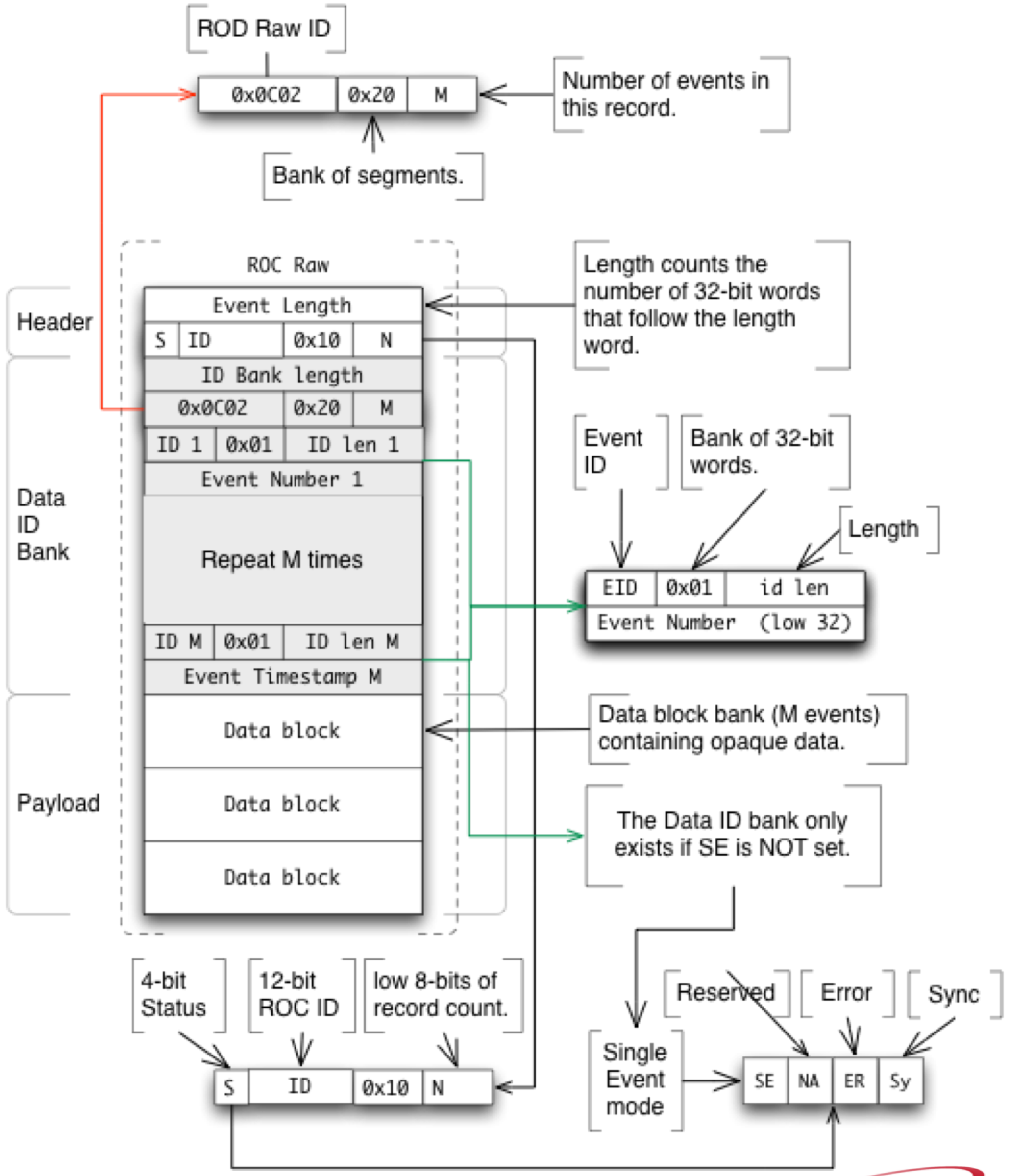


Jefferson Lab

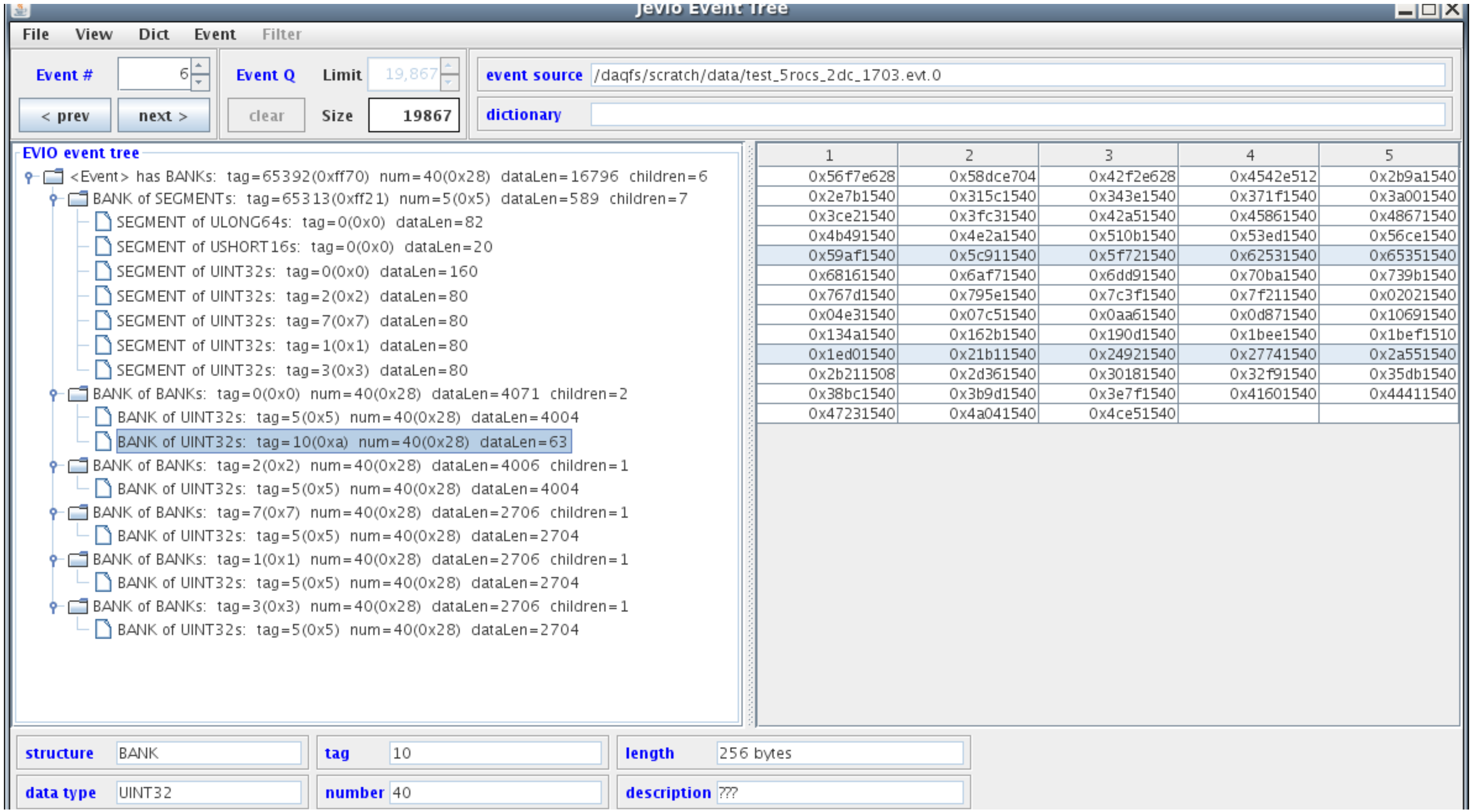
EVIO Data format, what does it look like

- EVIO data are are blocks of 32 bit words
- Bank - container for other data
- Each bank starts with a header (2 words)
  - Length (in words).
  - Description of content.

- The outer header tells us this bank contains more banks.
- The first bank is a list of trigger information for all the events in the block.
- The following "payload banks" contain blocks of raw data read from ADCs or TDCs.



EVIO File Viewer - jeviodmp



```
<Event> has BANKs:  tag=65392(0xff70)  num=40(0x28)  dataLen=16796  children=6
  BANK of SEGMENTs:  tag=65313(0xff21)  num=5(0x5)  dataLen=589  children=7
    SEGMENT of ULONG64s:  tag=0(0x0)  dataLen=82
    SEGMENT of USHORT16s:  tag=0(0x0)  dataLen=20
    SEGMENT of UINT32s:  tag=0(0x0)  dataLen=160
    SEGMENT of UINT32s:  tag=2(0x2)  dataLen=80
    SEGMENT of UINT32s:  tag=7(0x7)  dataLen=80
    SEGMENT of UINT32s:  tag=1(0x1)  dataLen=80
    SEGMENT of UINT32s:  tag=3(0x3)  dataLen=80
  BANK of BANKs:  tag=0(0x0)  num=40(0x28)  dataLen=4071  children=2
    BANK of UINT32s:  tag=5(0x5)  num=40(0x28)  dataLen=4004
    BANK of UINT32s:  tag=10(0xa)  num=40(0x28)  dataLen=63
  BANK of BANKs:  tag=2(0x2)  num=40(0x28)  dataLen=4006  children=1
    BANK of UINT32s:  tag=5(0x5)  num=40(0x28)  dataLen=4004
  BANK of BANKs:  tag=7(0x7)  num=40(0x28)  dataLen=2706  children=1
    BANK of UINT32s:  tag=5(0x5)  num=40(0x28)  dataLen=2704
  BANK of BANKs:  tag=1(0x1)  num=40(0x28)  dataLen=2706  children=1
    BANK of UINT32s:  tag=5(0x5)  num=40(0x28)  dataLen=2704
  BANK of BANKs:  tag=3(0x3)  num=40(0x28)  dataLen=2706  children=1
    BANK of UINT32s:  tag=5(0x5)  num=40(0x28)  dataLen=2704
```

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		

structure	BANK	**tag**	10	**length**	256 bytes
data type	UINT32	**number**	40	**description**	???

Jefferson Lab

Data storage

- 6 GeV experiments ran at tens of MB/s.
- 12 GeV experiments, hundreds of MB/s.
- Generate tens of petabytes per year.
- Tape is cheap but disk is faster.
- We write data to disk then copy from disk to tape later.
  - Tape speed only needs to handle average rate over a 24 hour period.
  - Tape drives and library robots are expensive and fragile. Writing to disk allows data taking to continue if the tape system breaks.
  - We typically have enough disk to hold three days of raw data.

JLAB Tape Library – CEBAF Center, F-Wing

Jefferson Lab

Looking forward

- New experiments are being proposed (eg MOLLER, TDIS, SoLID)
  - Detectors that do not play well together due to timing.
    - Traditional trigger and event builder strategies are not ideal.
  - Detectors with peculiar topologies.
    - Detectors split or segmented in a way that makes forming a trigger hard.
  - High event and/or data rates.
    - Particles from more than one event in a detector at the same time – need to disentangle.
- The data acquisition from these experiments does not fit well with current techniques.
- In the DAQ business we are always looking at ways to take advantage of the trends in electronics and computing.

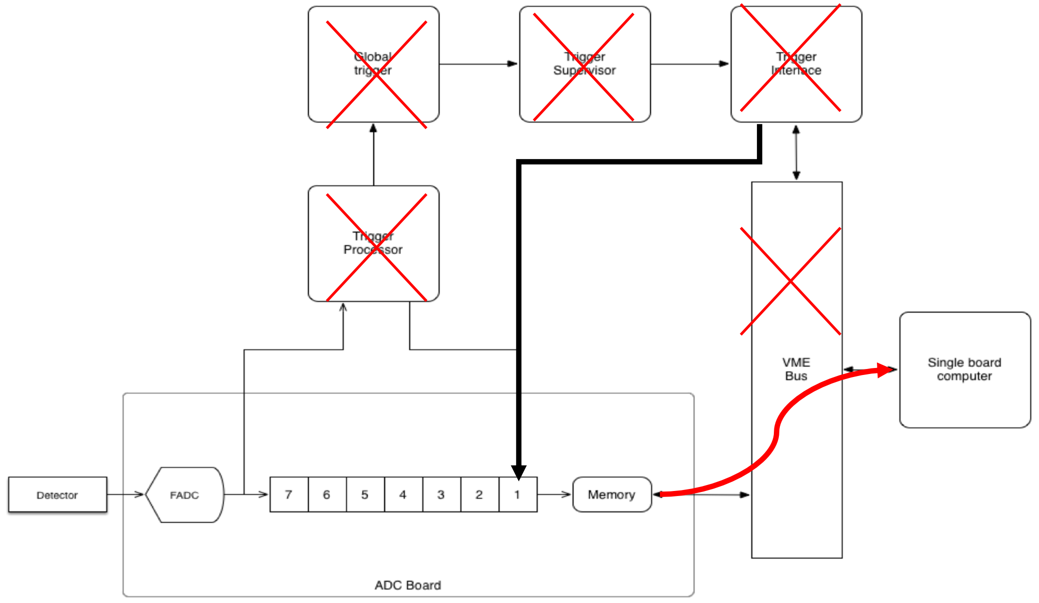
What would we expect to be happening?

- If we extrapolate current trends:
  - CPUs are becoming more powerful but the performance that matters for online systems is achieved mainly through doing more in parallel rather than improvements in per-core performance.
  - FPGA performance, affordability and usability are still improving.
  - IO and serial network bandwidth still seem to be growing exponentially.
- It is time to revisit the ideas that have dominated nuclear physics detector readout for the past twenty or thirty years.
  - Move things that we moved from hardware to software 25 years ago back into hardware (or at least firmware).
  - Reevaluate the use of busses – serial links are faster and more cost effective.
  - Reevaluate data flow.

Streaming mode

- In traditional triggered readout:
  - Data is digitized into buffers and a trigger, per event, starts readout.
  - Parts of events are transported through the DAQ to an event builder where they are assembled into events.
  - At each stage the flow of data is controlled by "back pressure".
  - Data is organized independently by event.

What if we can get rid of the the whole Front-end Trigger Processing path?



- In a Streaming readout:
  - Data is read continuously from all channels.
  - Validation checks at source reject noise and suppress empty channels.
  - The data then flows unimpeded in parallel channels to storage or a local compute resource.
  - Data flow is controlled at source.
  - Data is organized in multiple dimensions by channel and time.

Jefferson Lab

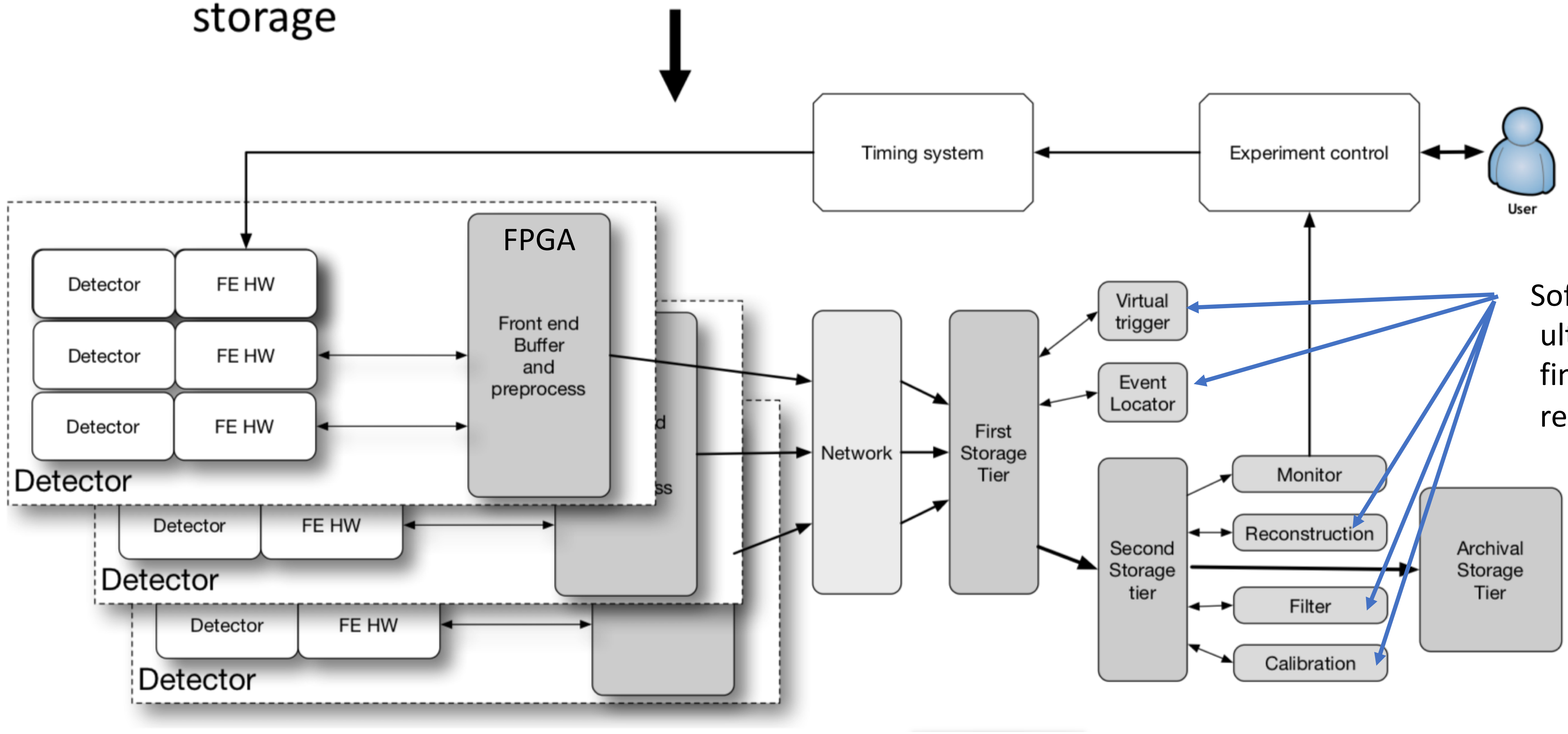
Streaming advantages

- The lack of a trigger means that:
  - Potentially useful physics is not discarded.
  - Run groups of experiments in parallel.
  - The system is simplified.
  - Readout speed is independent of detector response time.
  - Flow control at data source not via backpressure.
- Parallel timestamped streams mean:
  - System is robust against minor hardware or firmware glitches.
  - Can use different analysis techniques such as looking for hit patterns rather than reconstructing events.
- Requires robust and accurate time stamp generation and distribution.
  - Is still a simpler task than an online trigger.

Jefferson Lab

In practical terms, what does it look like?

- A timing system synchronizes streams at source.
- Front-end outputs data in streaming format on a net.
- DAQ consists of tiers processing separated by tiers of temporary storage

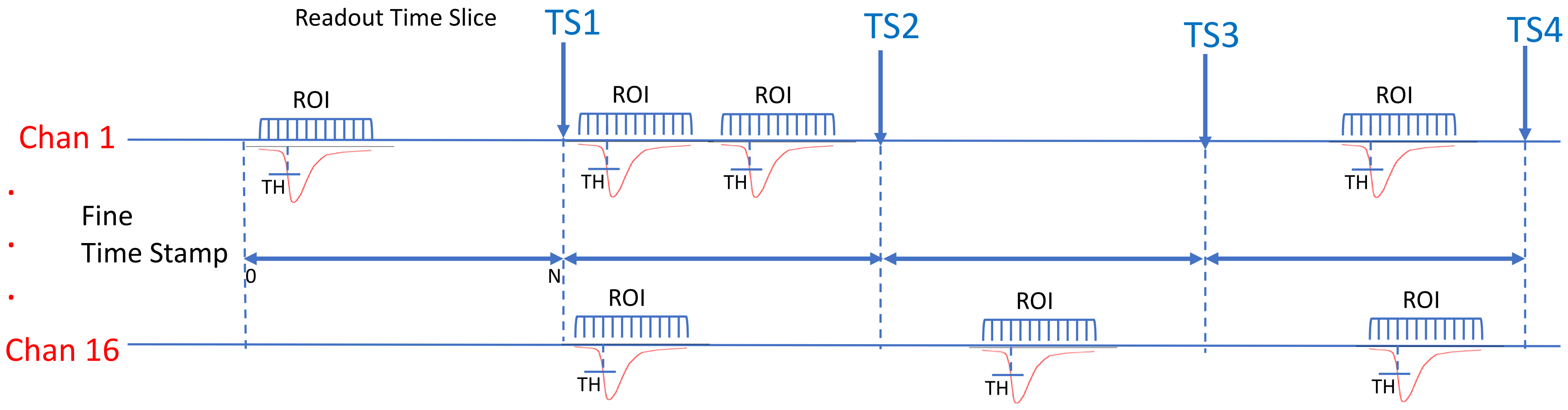


Software processing ultimately reduces the final data storage requirements

JLAB FADC in "Streaming Mode"

Streaming data can be thought of as Triggered mode where the trigger is a fixed pulser and you keep all the data for a single channel generated from the last pulse.

A 250 MHz FADC generates a 12 bit sample every 4ns. That corresponds to 3 Gb/s for one channel!
We can't deal with all that data (particularly if we have 1000s of channels) and we really do not want to.



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.

We can keep the individual samples or just compute a sum.

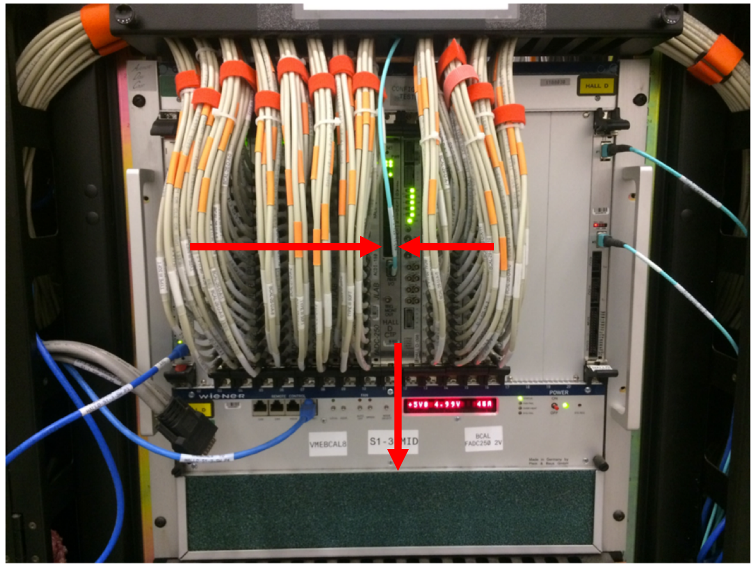
CODA Front-end for Streaming

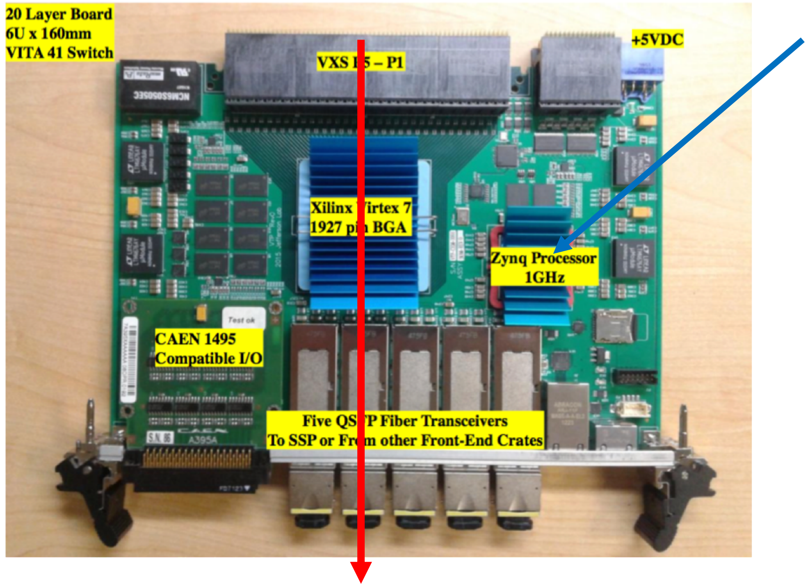
- VME Readout is a bottleneck for Streaming
- The VTP board is able to read 250 MHz FADC via the VXS serial backplane and stream the data out over the front panel transceivers.

a) Read a crate in this mode.
   - Zero suppression logic
   - Flow control
   - What comes out on the front panel fiber?
   - How to interface with DAQ?

b) The VXS interface allows read out for some detectors in streaming mode!

c) Can we repackage to remove need for expensive VXS crates etc. for all the detectors?
   - Cheaper systems for university groups?

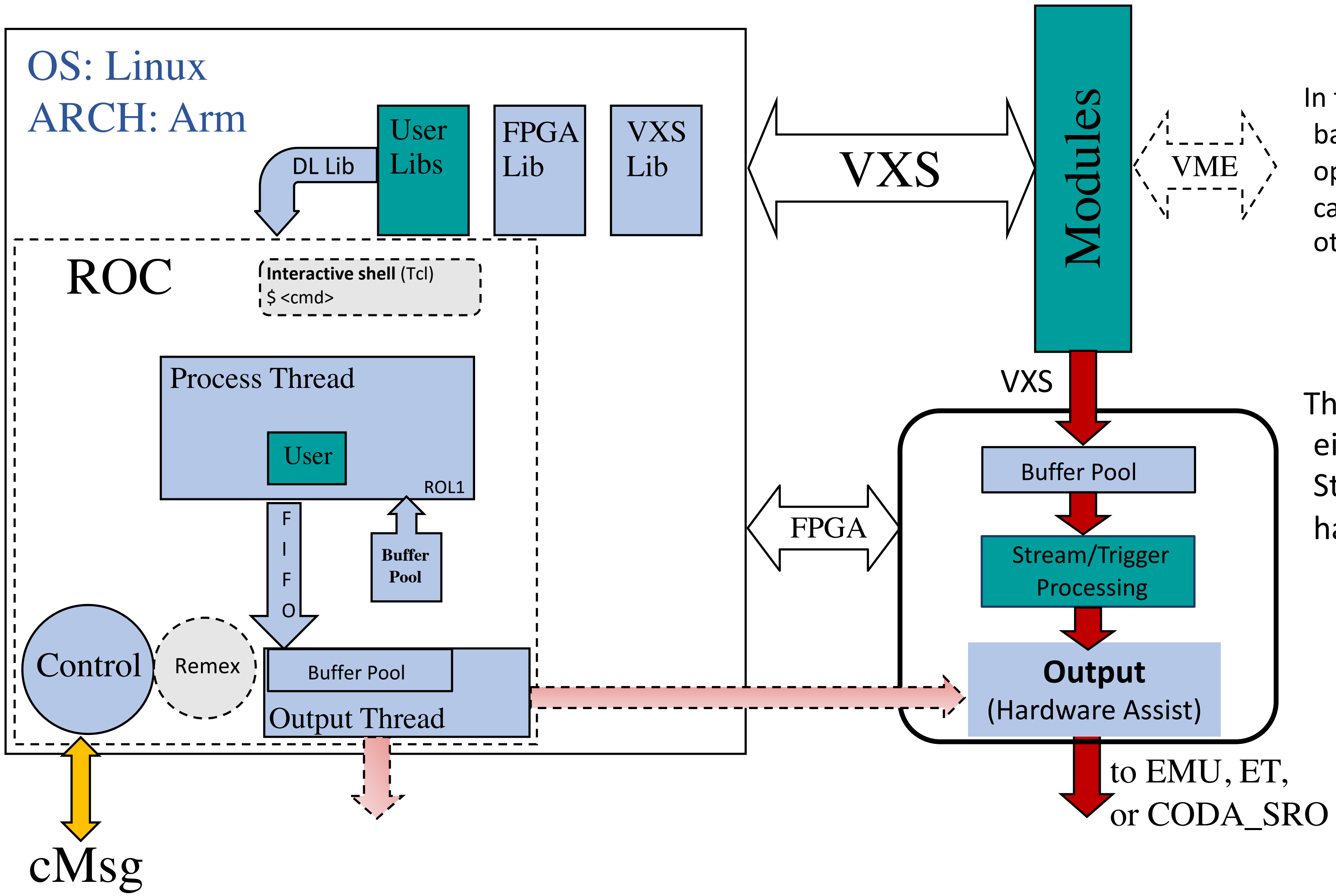




The CODA ROC can run on the ARM processor chip primarily to configure the hardware.

Data can be processed and streamed directly through the FPGA

CODA ROC on the VTP

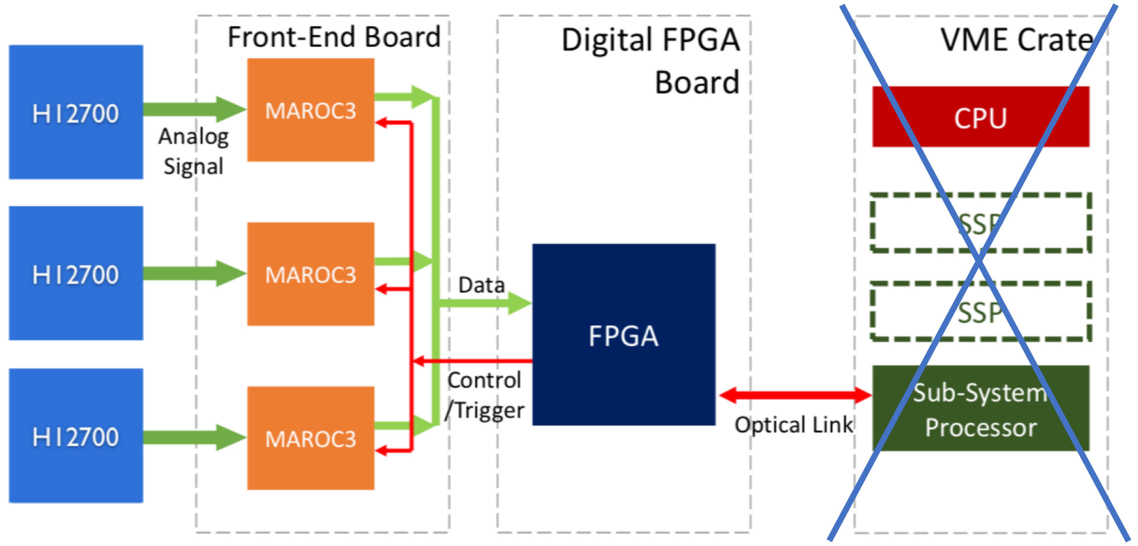


In this case the VME backplane becomes optional. The VME CPU can process scalars or other monitoring tasks.

The primary Data path either Triggered or Streamed stays in hardware.

Front-End Electronics – Sans crate!

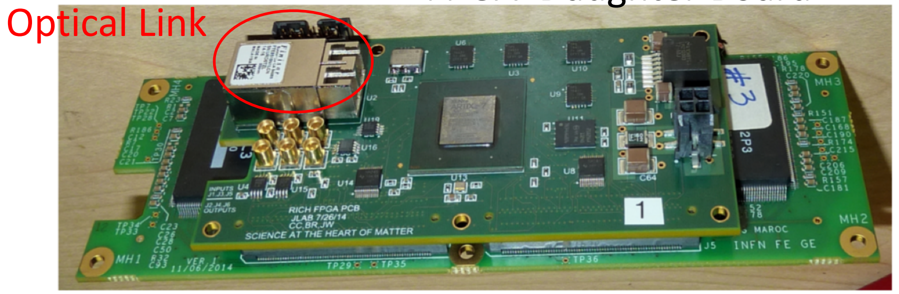
- Recent Front-End Electronics developed by the FEDAQ group have been implemented in several detectors (CLAS12 RICH and GlueX DIRC)
- Currently the detector data is sent to another FPGA board in a VXS crate, but in practice the optical link off the FEE FPGA can be streamed directly to an ethernet port on a PC.
- This can facilitate development by collaborators without big system overhead, but it is also a peek into the future streaming model for Big experiments.



MAROC ASIC "Front-End" Board

ASIC board produced for the RICH detector for CLAS12 with MAROC chips installed

FPGA Daughter Board



FPGA ~~b~~ Screenshot for the RICH detector for CLAS12 with ASIC board attached

Summary

- Data acquisition is constantly challenging.
  - Technology changes all the time.
  - Physicists think up experiments with tougher requirements.
  - The boundary between hardware and software is fluid and depends on what is available when a system is implemented.
  - There is always some R&D time to discover new algorithms and techniques.
  - To be honest, we do it because it’s fun and we get to play with all of the cool toys!
  - Thank you for your attention…