What we can do today – the RCDAQ perspective

Martin, BNL

I want to lay out the abilities that we already have and have been using for a long time.

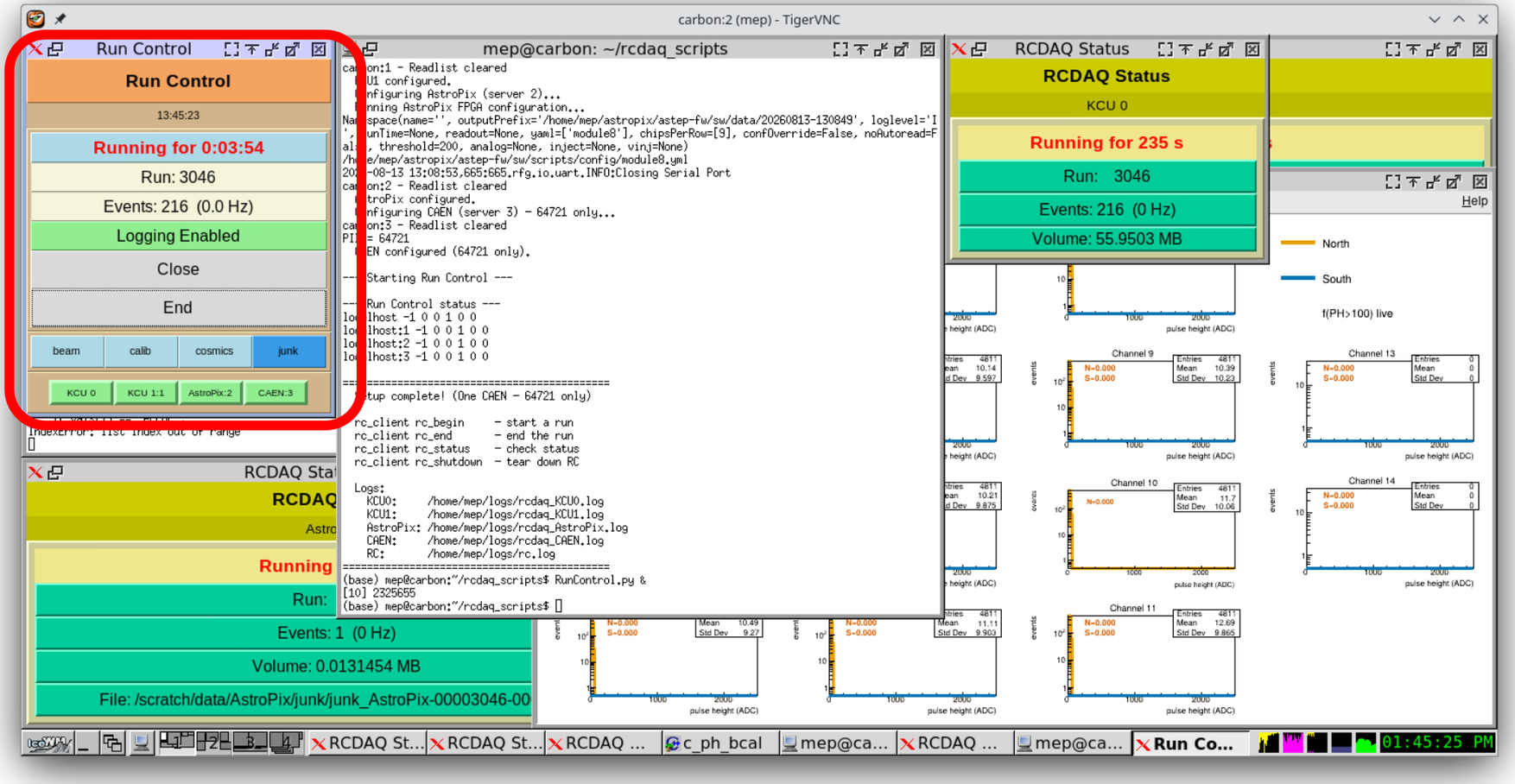
- We can take data with the available ePIC front-end prototypes, chiefly the H2GCROC and the Astropix (we are also supporting lots of non-EPIC specific stuff, such as CAEN hardware, the SRS, DRS4, what have you)
- We can build Timeframes from multiple streams, and build Super-Timeframes, too
- We have a “Run Control” that manages an arbitrary number of components for synchronized starts/stops, etc etc
- We can transport the data in many different ways, to disk, through the LAN, and through the WAN
- We have a widely used online monitoring framework
- All data banks/containers/packets have well-defined data access APIs
- Well-defined APIs to obtain data from the front-ends
- We can interface the data (TFs/STFs) to EicRecon.

It exists!

The list of abilities is not just what we intend to do.

We have used all of the above, and I can demonstrate that all of it works

The flagship application in ePIC-land was the recently completed Barrel Imaging Calorimeter test beam in Jlab's Hall D that featured 4 individual data streams, with 3 of them in streaming mode



TimeFrames = containers of data for a given time span

The RCDAQ readout/data format natively supports streaming data

Not a new development - the sPHENIX tracking system was read out with FELIXes in streaming mode

In RCDAQ parlance: a “Packet” = “Timeframe”

It contains data from multiple clocks (in the real data, these correspond to different beam crossings)

Here is an example of H2GCROC data spanning 10 different clocks

```
$ ddump -p 12001 -e 8 /data/purschke/BIC/data/KCU0/beam/beam_ROC_0-
00002784-0000.evt | grep Wave
 Number of Waveforms: 10
 ----- Waveform  0 Samples:  2  Timestamp: 0x30a4c6dd  816105181
 ----- Waveform  1 Samples: 20  Timestamp: 0x30a54121  816136481
 ----- Waveform  2 Samples: 20  Timestamp: 0x30a5e401  816178177
 ----- Waveform  3 Samples: 20  Timestamp: 0x30a641c1  816202177
 ----- Waveform  4 Samples: 20  Timestamp: 0x30a6ce75  816238197
 ----- Waveform  5 Samples: 20  Timestamp: 0x30a7de49  816307785
 ----- Waveform  6 Samples: 20  Timestamp: 0x30a8c3c5  816366533
 ----- Waveform  7 Samples: 20  Timestamp: 0x30a9456d  816399725
 ----- Waveform  8 Samples: 20  Timestamp: 0x30aa1d19  816454937
 ----- Waveform  9 Samples: 13  Timestamp: 0x30aabd75  816495989
```

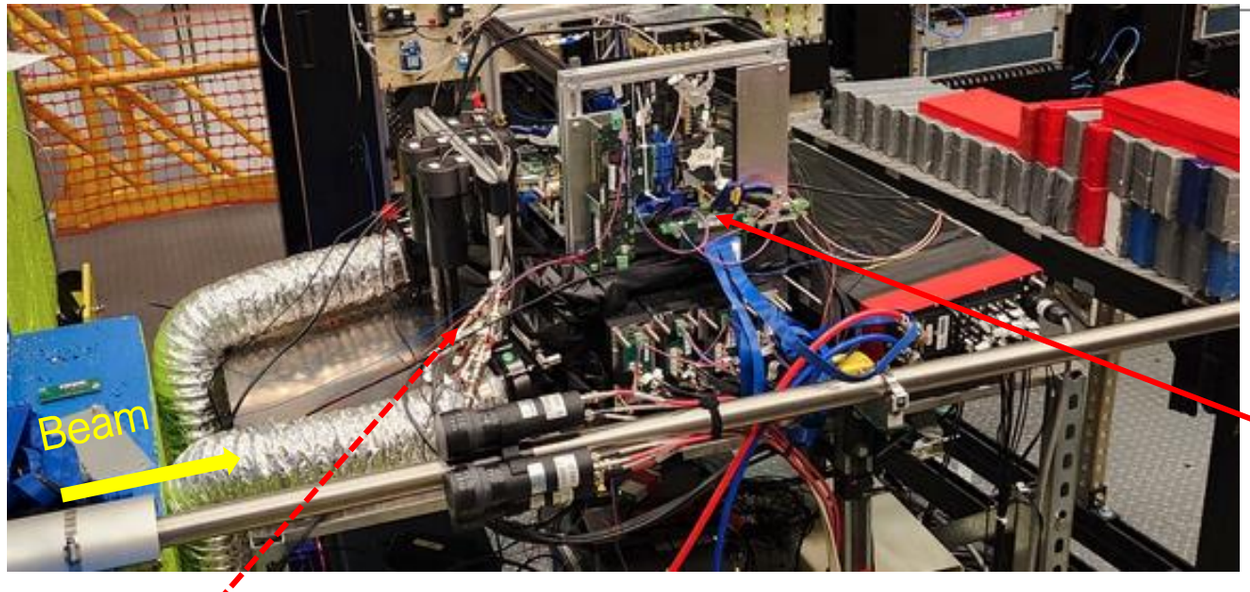
Each “crossing” reads out 288 individual channels.

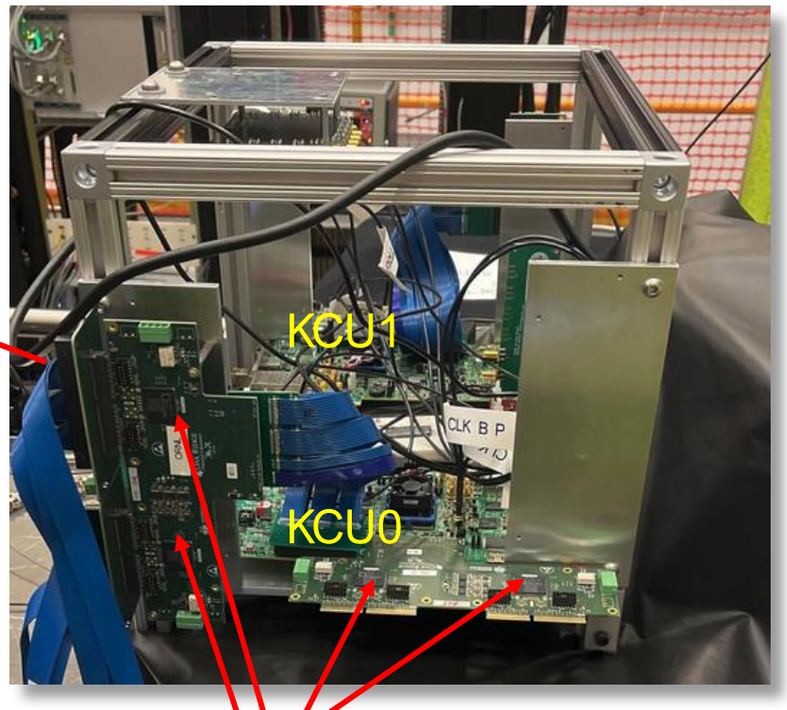
So you have 20x288 individual waveforms here, sorted by clock.

We had low-intensity beam here; we close a TF when there is a “gap” so data don’t get stuck

There is no practical size/time limit to a TF (max size 16GB) to accommodate the envisioned 600ms.

Some context: What did we read out

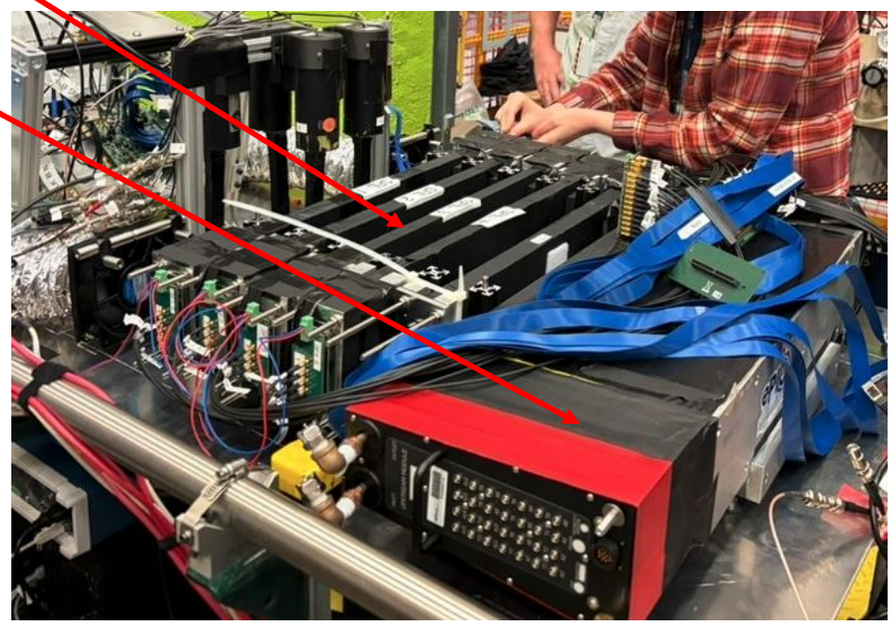




Astropix (not visible)

5 SFILs = Scint. Fiber Inter Layer (192 SiPMs each side)

Baby BCal



4 H2GCROC ASICs per KCU = 288 channels/KCU

Each KCU is read out in its own stream.

Now we need to combine all these data!

First, some other data

Our KCUs had a common clock, but there is no GTU to keep the *counters* the same across streams

The KCU clocks count in sync (same source) but there is no common reset mechanism

So the clocks have an offset that we need to determine on a run-by-run basis for our data

It remains constant for the duration of a run

BTW, the Astropix has its own clock, which makes things more complicated. We are working on sync'ing that but let's concentrate on the KCUs.

Let me show you some ideal data first that did have the benfit of a GTU (from the sPHENIX INTT).

Same idea, multiple crossings in a Timeframe, but this time it's all really synchronized.

And each of the 8 FELIXes/DAMs was a separate stream, held together by the GTU

INTT Data

The INTT has 8 individual streams. Each stream has one data packet

```
$ ls -1  /sphenix/lustre01/sphnxpro/physics/INTT/physics/*00080279*0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt0-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt1-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt2-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt3-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt4-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt5-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt6-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt7-00080279-0000.evt
```

The packets are “named” by an ID that uniquely identifies the stream. The INTT had 3001…3008.

```
$ dlist /sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt0-00080279-0000.evt
Packet  3001 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
$ dlist /sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt1-00080279-0000.evt
Packet  3002 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
...
$ dlist /sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt7-00080279-0000.evt
Packet  3008 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
```

Clocks/Crossing numbers

The sPHENIX “GTU” provided each stream (not just the INTT) with its master-count of the global crossing number (BCO). Each stream gets 40bits of the GTU’s 64bit number.

Here I fish out the first few BCOs in that first timeframe (154 total here):

```
$ ddump  /sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt0-00080279-0000.evt
| grep BCO
 Number of unique BCOs: 154
 BCO    0:  0x22cba8872e    number of FEEs for this BCO  14
 BCO    1:  0x22cba887a6    number of FEEs for this BCO  14
 BCO    2:  0x22cba8881e    number of FEEs for this BCO  14
 BCO    3:  0x22cba88896    number of FEEs for this BCO  14
 BCO    4:  0x22cba8890e    number of FEEs for this BCO  14
 BCO    5:  0x22cba88986    number of FEEs for this BCO  14
. . .
```

And here is the next stream (here all 8 happen to have this same list. Not guaranteed in SRO!)

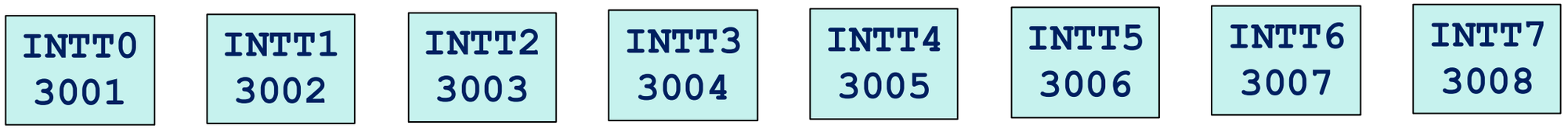
```
$ ddump  /sphenix/lustre01/sphnxpro/physics/INTT/physics/physics_intt1-00080279-0000.evt
| grep BCO
 Number of unique BCOs: 154
 BCO    0:  0x22cba8872e    number of FEEs for this BCO  14
 BCO    1:  0x22cba887a6    number of FEEs for this BCO  14
 BCO    2:  0x22cba8881e    number of FEEs for this BCO  14
 BCO    3:  0x22cba88896    number of FEEs for this BCO  14
 BCO    4:  0x22cba8890e    number of FEEs for this BCO  14
 BCO    5:  0x22cba88986    number of FEEs for this BCO  14
```

Combining all streams

Now we take those 8 streams and combine them into what the Timeframe is meant to look like, all 8 packets 3001…3008 in one place, sync'ed by BCO

(In the real reconstruction we combine all 84 streams that make up the full sPHENIX detector complement)

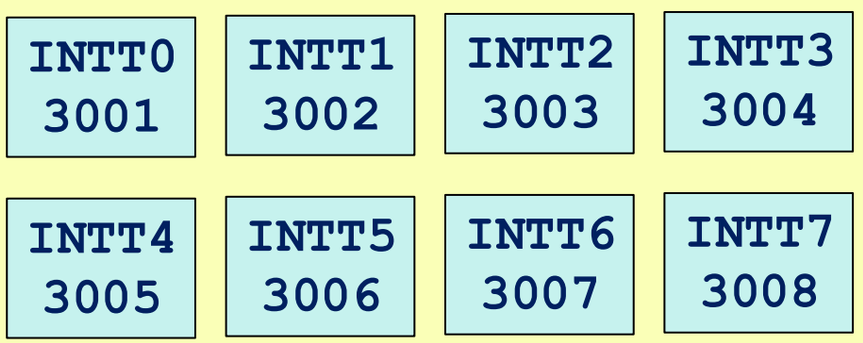
Here I'm doing this for 500 time frames only:




```
$ eventcombiner -n 500 /sphenix/data/data01/purschke/intt-00080279-0000.evt
/sphenix/lustre01/sphnxpro/physics/INTT/physics/*00080279*0000.evt
```

And here we are. An all-detector timeframe.

```
$ dlist  /sphenix/data/data01/purschke/intt-00080279-0000.evt
Packet  3001 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3002 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3003 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3004 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3005 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3006 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3007 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
Packet  3008 16388 -1 (sPHENIX Packet) 110 (IDINTTV0)
```



Ok, that's the *Timeframe*. Where is the Super-Timeframe?

Behind the scenes, it is there. The "cash and carry" of the data are "buffers"

Buffers are there to make larger-sized entities of the often too small Timeframes

Buffer = SuperTimeframe = many Timeframes

In day-to-day operations best seen in online monitoring

The online monitoring receives buffers with many contiguous TFs (needed in many monitoring applications)

Each buffer/STF is a self-contained contiguous large "time section" of data

The buffers/STFs are also the ones that "move" – to the disk, through the network, to a remote "combiner"

With my measly 500 TFs, I got 4 such STFs. The sizes are completely tunable to what we need/want.

```
$ prdfcheck  /sphenix/data/data01/purschke/intt-00080279-0000.evt
buffer   128 at record    0 length = 66647704  8136 marker = ffffc0c0  sPHENIX Marker
buffer   127 at record 8136 length = 66604920  8131 marker = ffffc0c0  sPHENIX Marker
buffer   127 at record 16267 length = 66604920  8131 marker = ffffc0c0  sPHENIX Marker
buffer   118 at record 24398 length = 61884888  7555 marker = ffffc0c0  sPHENIX Marker
```

Back to our KCUs

Here we need to determine the clock offset first before diving into the TF/STF building

```
$ ddump -e 3 /data/purschke/BIC/data/KCU0/beam/beam_ROC_0-00000723-0000.evt | grep Wave
 Number of Waveforms: 10
 ----- Waveform  0 Samples:  8  Timestamp: 0x30bf57b7  817846199
 ----- Waveform  1 Samples: 20  Timestamp: 0x30c00497  817890455
```

```
$ ddump -e 3 /data/purschke/BIC/data/KCU1/beam/beam_ROC_1-00000723-0000.evt | grep Wave
 Number of Waveforms: 30
 ----- Waveform  0 Samples:  7  Timestamp: 0x30b77a28  817330728
 ----- Waveform  1 Samples:  2  Timestamp: 0x30b7fe67  817364583
```

There is no guarantee (no GTU!) that the first timestamp in both streams is actually the same event

But naively using the offset 817846199 – 817330728 = 515471, we can get “close”

But after that we need to rely on the actual data to determine the exact offset (beyond the scope here)

Same recipe:

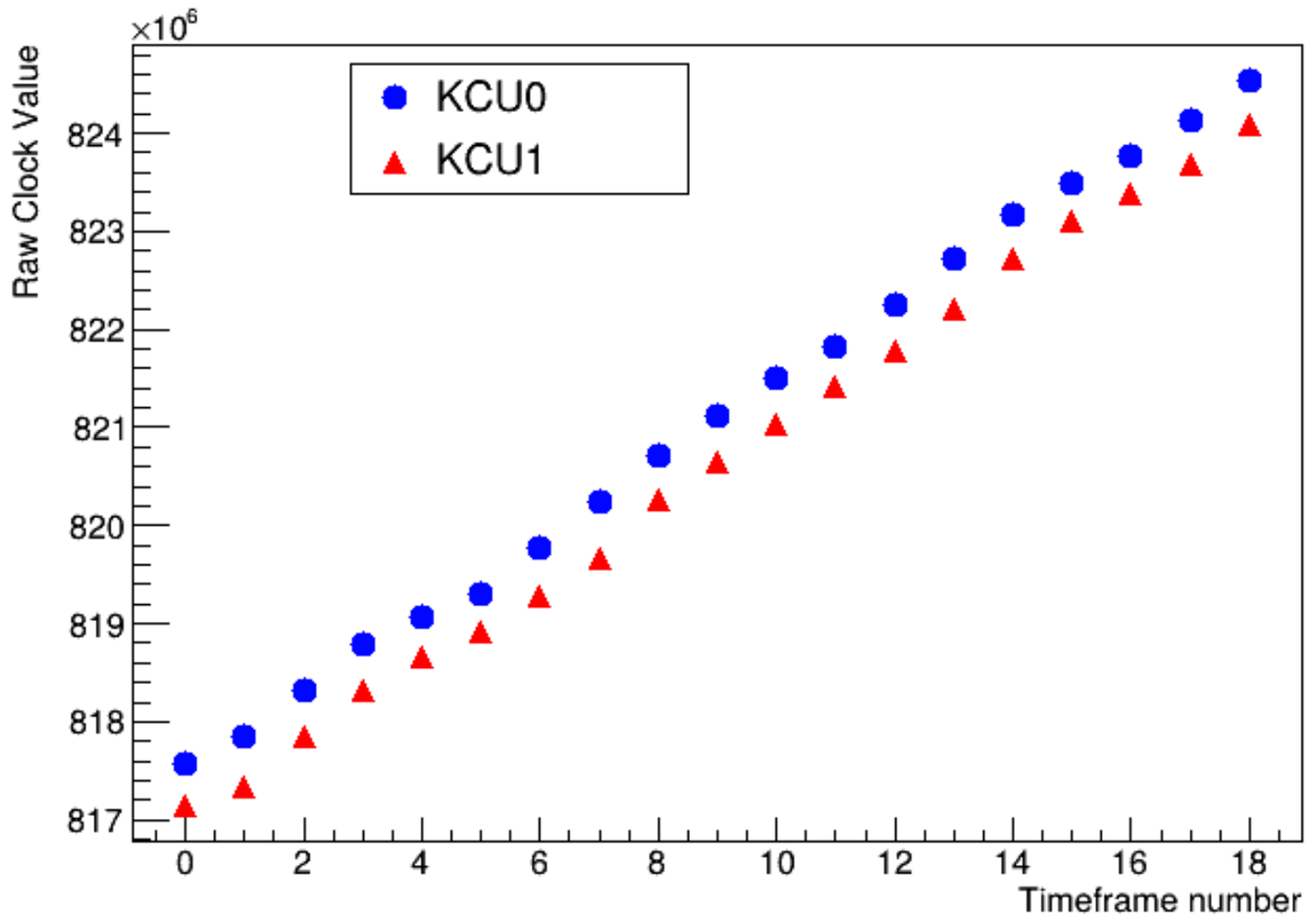
KCU0 12001 + KCU1 12002 ➡ beam_combined-00000723-0000.evt

```
$ dlist beam_combined-00000723-0000.evt
Packet 12001 68304 -1 (ePIC Packet) 302 (IDH2GCROC3_10G)
Packet 12002 68304 -1 (ePIC Packet) 302 (IDH2GCROC3_10G)
```

Clock correlation

Here are the original clock values, just using the offset to make a crude correlation.

It's not 100% right yet, but that's a different issue - all that goes away with a proper GTU.



Ok, that's a different WG ☺

Online STF building

So far I have shown the STF building from existing files.

We can do the same online by receiving the streams from multiple RCDAQs.

Unfortunately I have nothing in hand where I could generate *two* or more clock-sync'ed streams "live"

So I can show only one. We are taking just one TF stream and "combine" it into, well, a cohort of one.

But: I can do this through the LAN, and I can do this through the **WAN**

Just to show off a bit with a future Echelon1-type logging from BNL directly to JLab, I finessed this:

```
purschke@mlp6: ~
purschke@mlp6:~$
purschke@mlp6:~$ daq_status -l
mlp6 -  Stopped
  Filerule:     rcdaq-%08d-%04d.evt
  Logging enabled via remote server localhost Port 5001
  have a trigger object
  Buffer Sizes:     65536 KB adaptive buffering: 15 s
  Web control Port:  8899
  Elog: not defined
  -- defined Run Types:   (none)
   No Plugins loaded
purschke@mlp6:~$ daq_begin
mlp6 - Run 11 started
purschke@mlp6:~$ daq_end
mlp6 - Run 11 ended
purschke@mlp6:~$
```

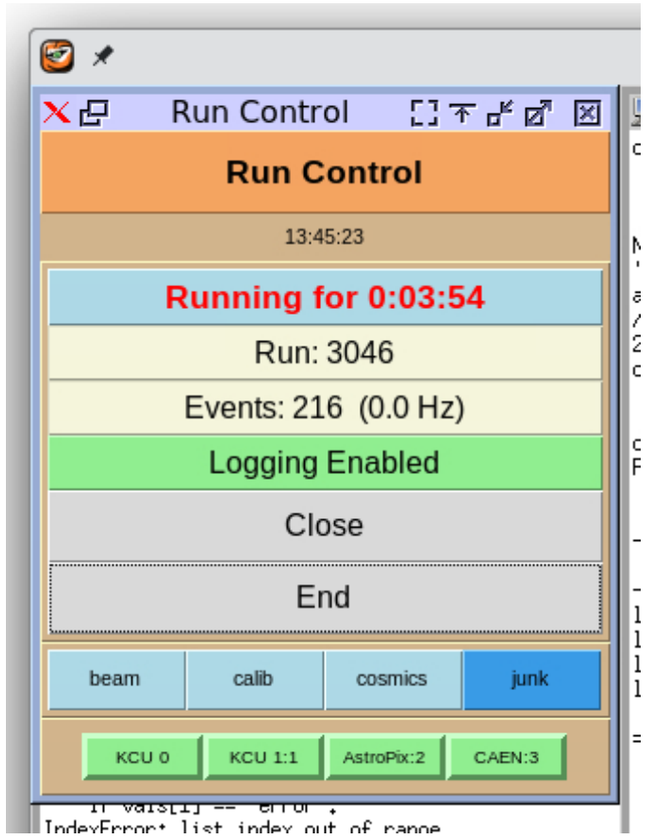
BNL

```
purschke@ifarm2401:~                                                    17
[purschke@ifarm2401 ~]$
[purschke@ifarm2401 ~]$ aml2 5001 test_to_jlab-%08d-%04d.evt
aml version 1.02 Built on Aug 26 2026  00:55:09
pid_key = 840
Semid = 3
write thread created
waitforendrun thread created, ID=140523388003904
Wed Aug 26 01:17:04 2026
  /tmp/aml_  waitforendrun thread started
bind complete on port 5001
Wed Aug 26 01:17:32 2026
  /tmp/aml_  new connection accepted from localhost
New thread 7fce261fd640 started for host localhost
 new run = 11 sent from localhost
Wed Aug 26 01:17:48 2026
  /tmp/aml_  waitforendrun recognized start of run
Opened file: test_to_jlab-00000011-0000.evt
Wed Aug 26 01:17:58 2026
  /tmp/aml_ Run end signal from localhost   Number of buffers: 2
Wed Aug 26 01:17:58 2026
  /tmp/aml_  waitforendrun recognized end of run, buffers= 2
Closing file test_to_jlab-00000011-0000.evt
md5sum: d41d8cd98f00b204e9800998ecf8427e  test_to_jlab-00000011-0000.evt
Wed Aug 26 01:17:58 2026
  /tmp/aml_ Closing connection from localhost   Number of buffers: 2
```

ifarm@jlab

Switching gears – Run Control

You have seen this screenshot from the BIC data taking already – cropped and zoomed here



It is a meta-control for any number of RCDAQ instances that then operate as a coherent “cohort”

Synchronized starts/stops (and other controls, like the “Run Type” selection, such as beam/calib/cosmics/junk)

We wouldn’t have been able to operate the 4 streams otherwise

I had mentioned the 84 instances we had in sPHENIX – that’s how it looked there.

It has hooks for arbitrary actions at begin/end run (used for systems requiring special treatment at begin/end run)

It also controlled the GTU (de-fanged for the BIC since there is none)

Online monitoring

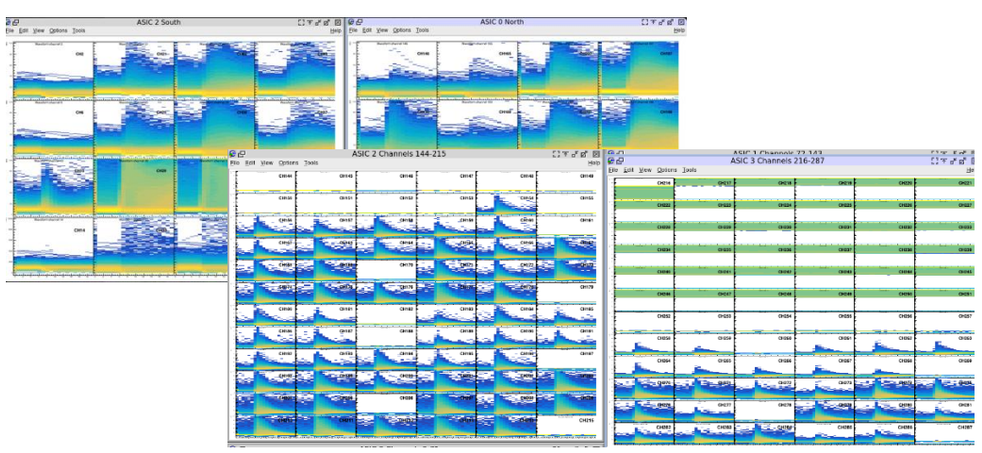
RCDAQ supports real-time online monitoring.

An arbitrary number of clients (usually just one) can opportunistically request STFs (buffers) from RCDAQ and fill monitoring histograms

There is a framework called pmonitor that makes that a breeze

pmonitor (and any client, really, dlist, ddump, …) can open any type of data stream and monitor “live” data, or replay data from a file or a combined file

We had one “central” online monitoring setup that made 6 or so root canvases and several “specialty” ones



We could see in seconds what, say, a change in some parameter did

Provisions to automatically reset all histos for a new run, and save everything for posterity at end-run

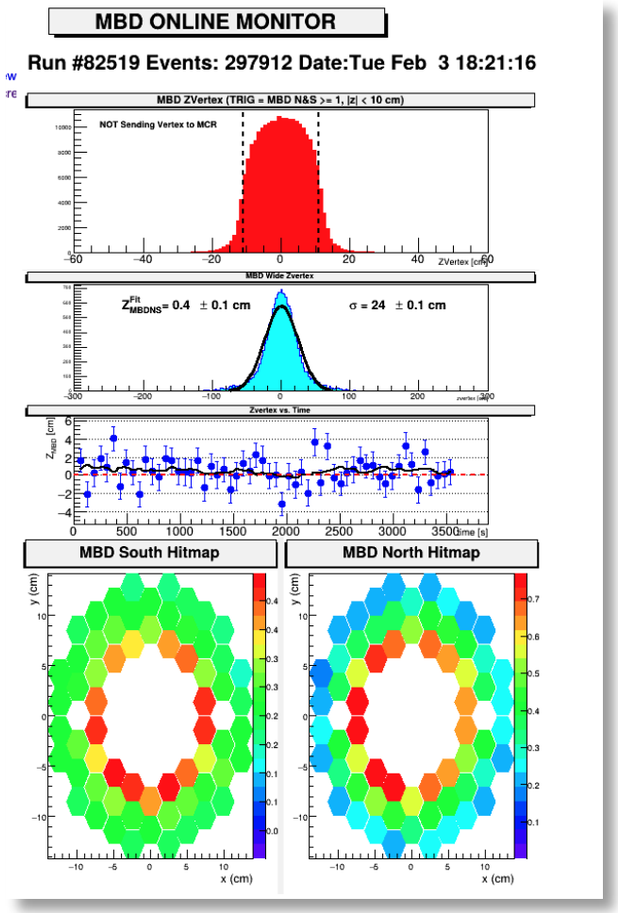
POMS – more formalized online monitoring

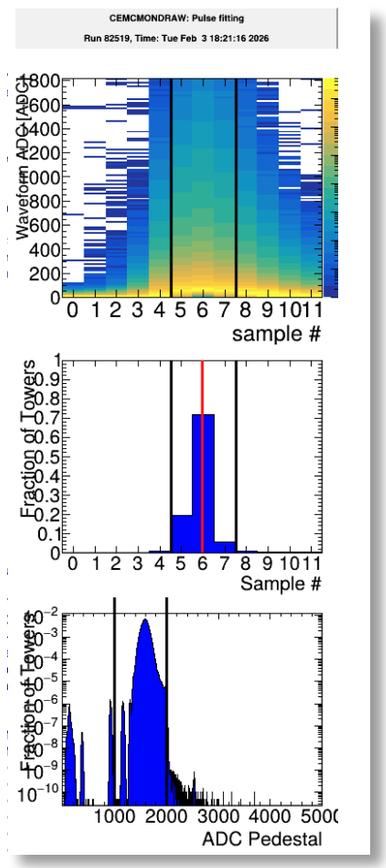
In sPHENIX we had formalized the online monitoring with a class/wrapper called POMS - the ePIC Online Monitoring System" ☺

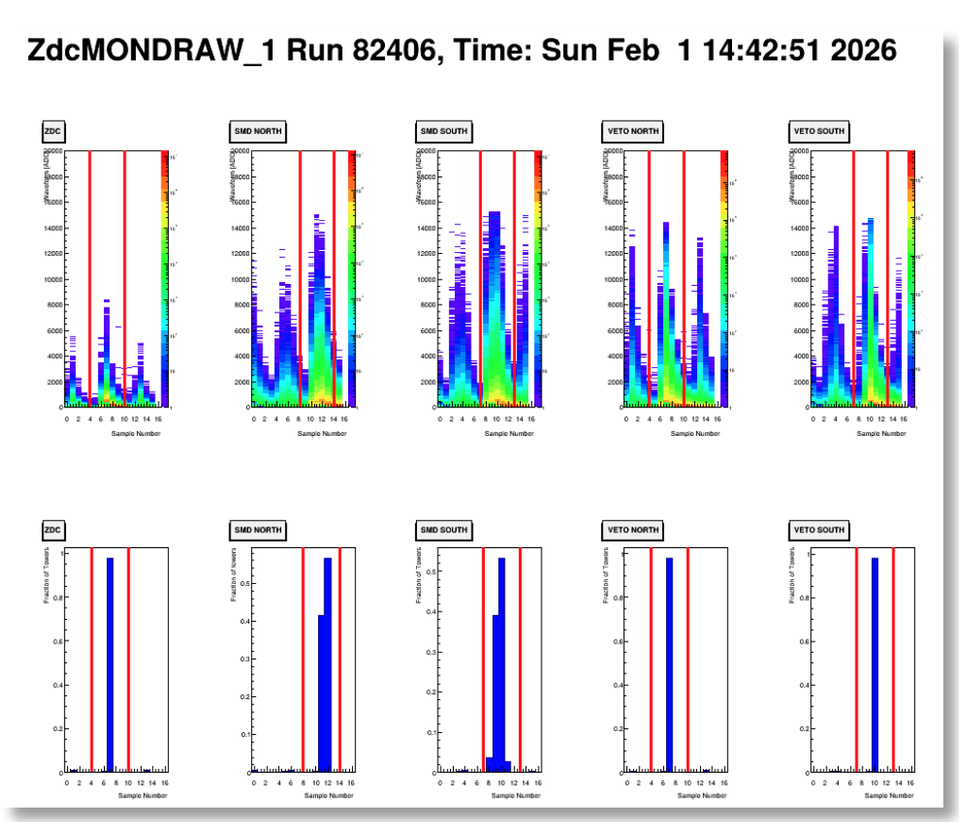
The end product was a stylish root Canvas with annotations, explaining what you see and how stuff should look like

This is just a framework that the detector experts used to set up various levels of monitoring (for shift crew, detector experts, power users…)

We have each monitoring canvas captured for every run ever taken by sPHENIX







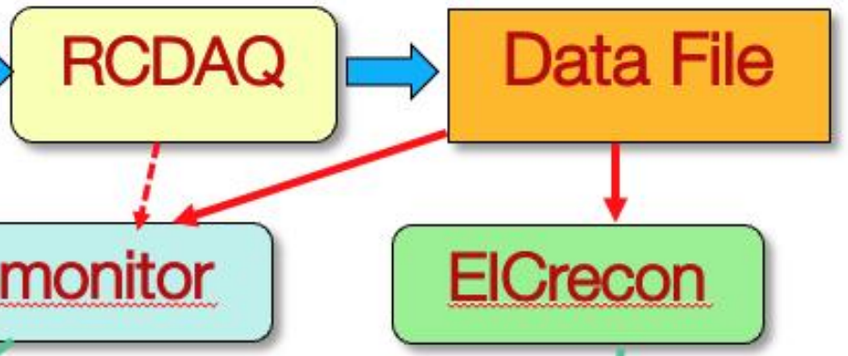
Final thing – eicrecon support

We should get into the habit of using the actual analysis framework to analyze our data

Last December I demonstrated a full “in-house” analysis chain for RCDAQ data with eicrecon

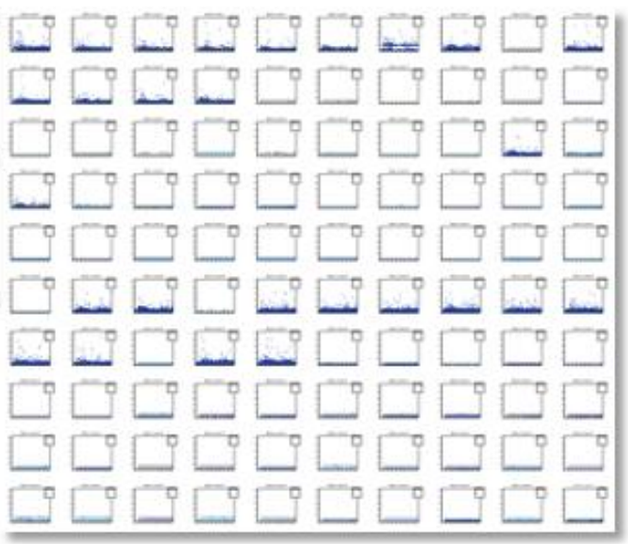
I used the at the time latest ZDC test beam data (also a H2GCROC3, but any front-end will work)

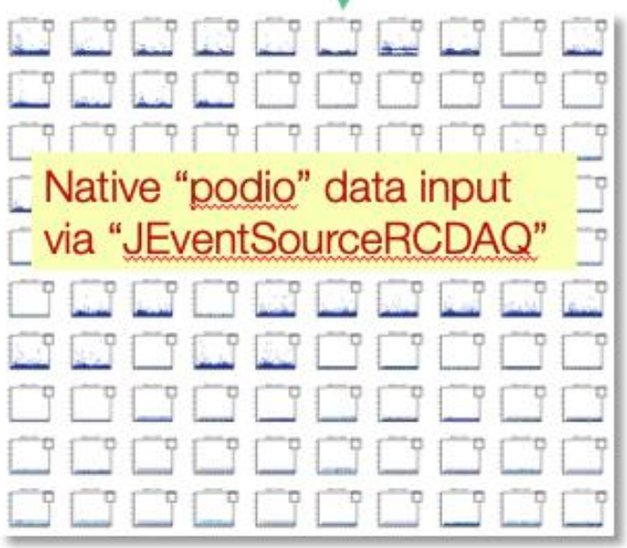
https://indico.bnl.gov/event/30440/



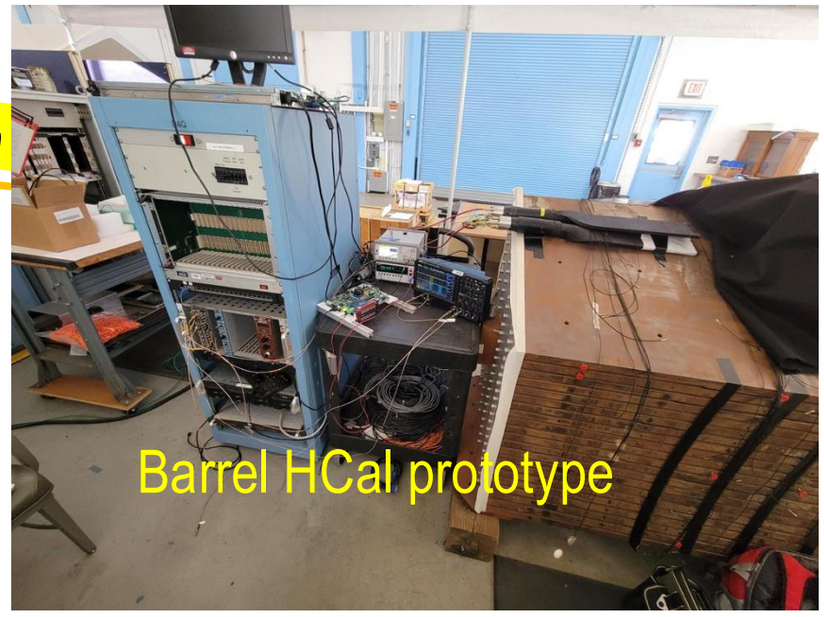
I re-tooled the existing pmomitor-based analysis with EICrecon

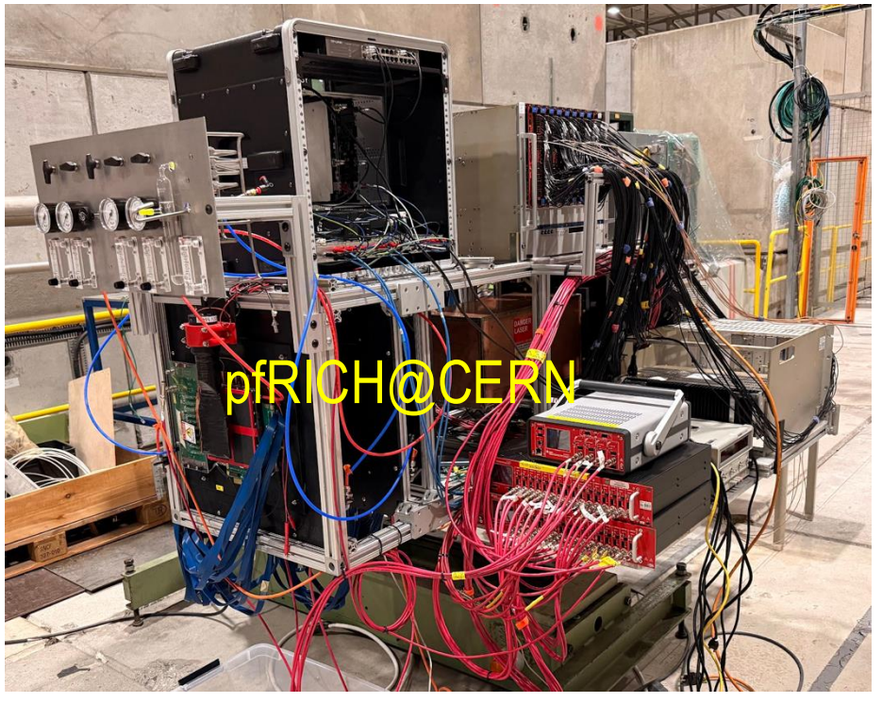
This gives identical output/results from both

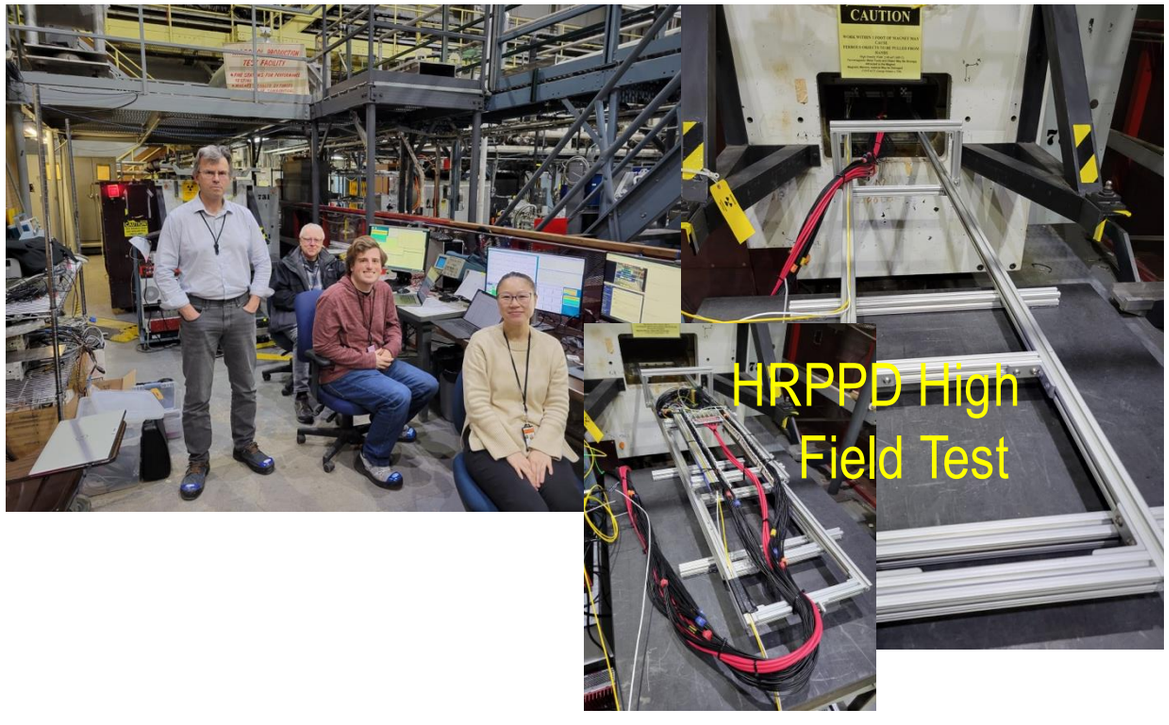


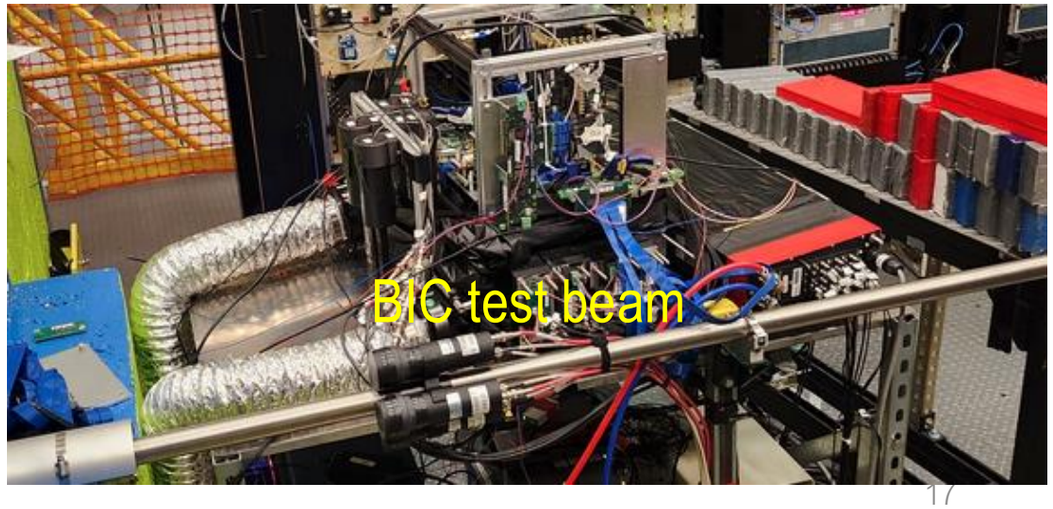


Select recent ePIC Data Taking Campaigns









We have plugins for many common devices

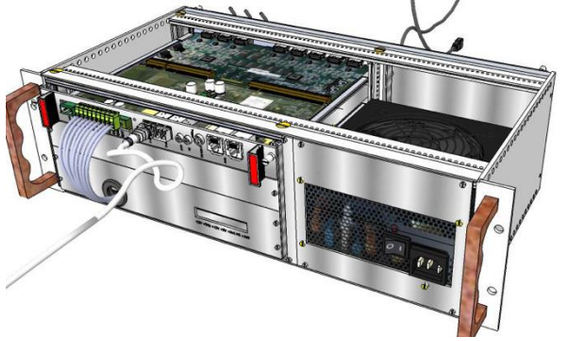
RCDAQ

PCIe

PCIe

DRS4 Eval board

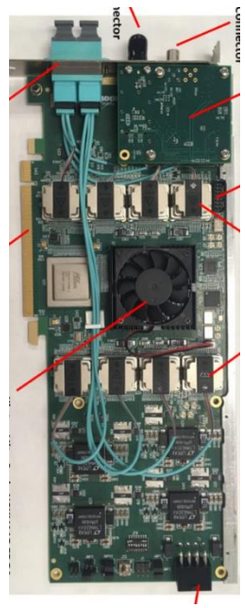
“USB Oscilloscope”



The CERN RD51 SRS System

The CAEN V1742 waveform digitizer

Dream



FELIX Cards

The SRS used to be **the** workhorse readout system (still used a lot in ePIC R&D with MPGDs)
The support here is what got RCDAQ on the map in many external places

And many more!
I know of about 35 or so

Summary

The structures-by-a-different-name in RCDAQ map seamlessly on to Timeframes and Super-Timeframes

We can combine individual streams from a cohort of RCDAQs into a clock-matched all-detector STF

The underlying protocols are network-transparent and work on the LAN and the WAN

The “buffer” == STF is already used as a standalone/self-contained structure, just the way we envision it

We have Run Control as a meta-controller to seamlessly manage a cohort of RCDAQs, each reading out a particular FELIX/DAM (or what have you)

We have support for real-time, “live” online monitoring (or to replay existing data) – also network-transparent

Data can be analyzed in eicrecon

We used most of the above features routinely in 5 different measurement campaigns, but most comprehensively in the BIC test beam.

I will get one grad student next week to work with me, and may get another 0.5 PD, to be seen

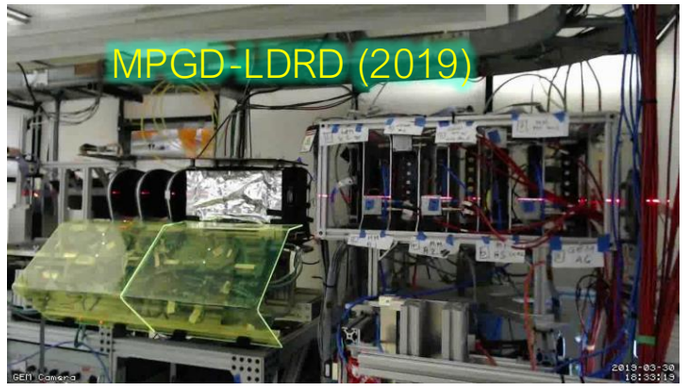
I would very much appreciate and welcome collaboration on this. The various test beams (and, in case of pfRICH, years-long buy-in have helped to grow some experts already)

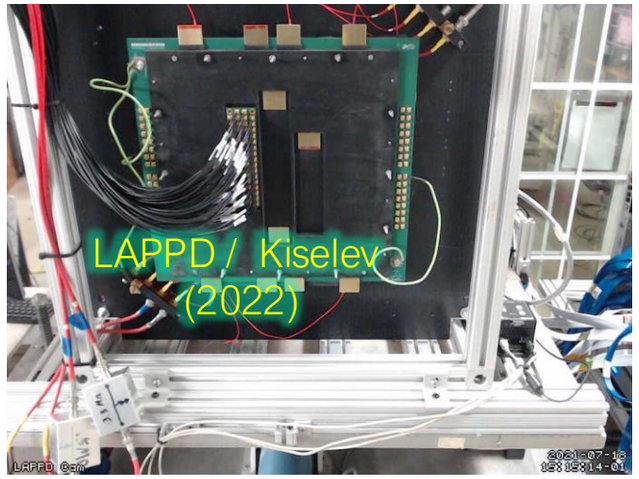
Long History of EIC-themed test beams with RCDAQ

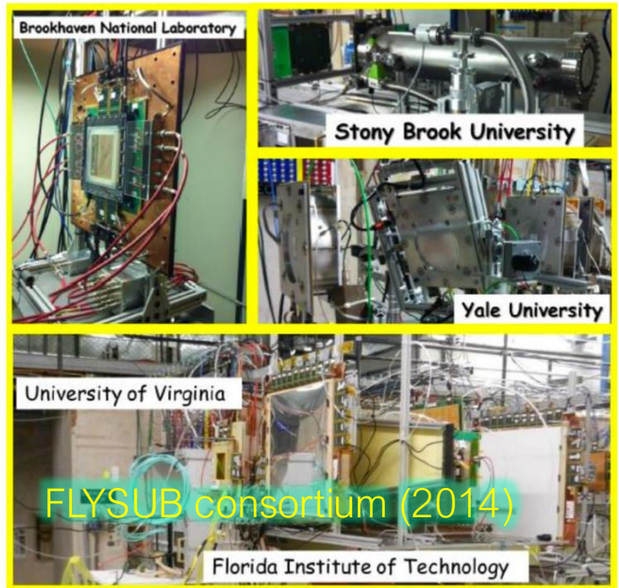
The RCDAQ system has been a pillar of EIC-themed data taking for R&D, test beams etc since 2013 – eRD1, eRD6, LDRDs, …

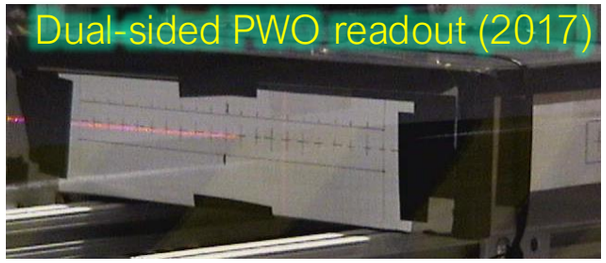
Many active RCDAQ installations in the ePIC orbit + ~30 elsewhere

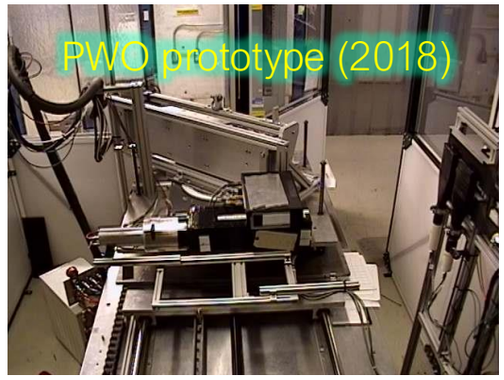
Usual entry by ease-of-use for standard devices (DRS, SRS, CAEN, …) and support for fully automated measurement campaigns











Thanks to Jin Huang for this collection