The RCDAC Data Acquisition System

Martin L. Purschke

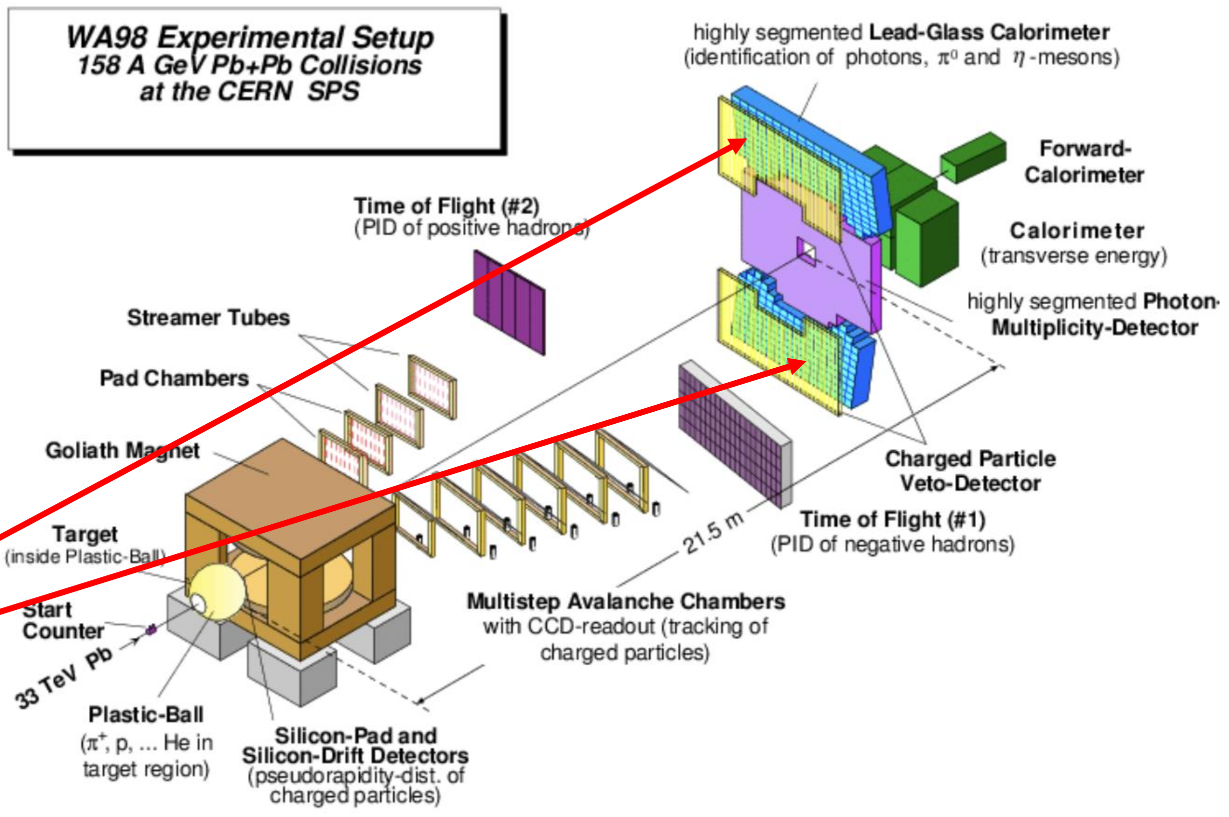
Brookhaven National Laboratory

DAQ manager of sPHENIX

Previous DAQ/Electronics WG co-convener for what is now ePIC

I have been a DAQ (and notably, a calorimeter) guy since my early days at CERN (1985-1996)

Why do I have a soft spot for calorimeters? I helped design and build the original lead glass "SAPHIR" SF5 lead glass calo for WA80… played a big role in building the WA98 calos

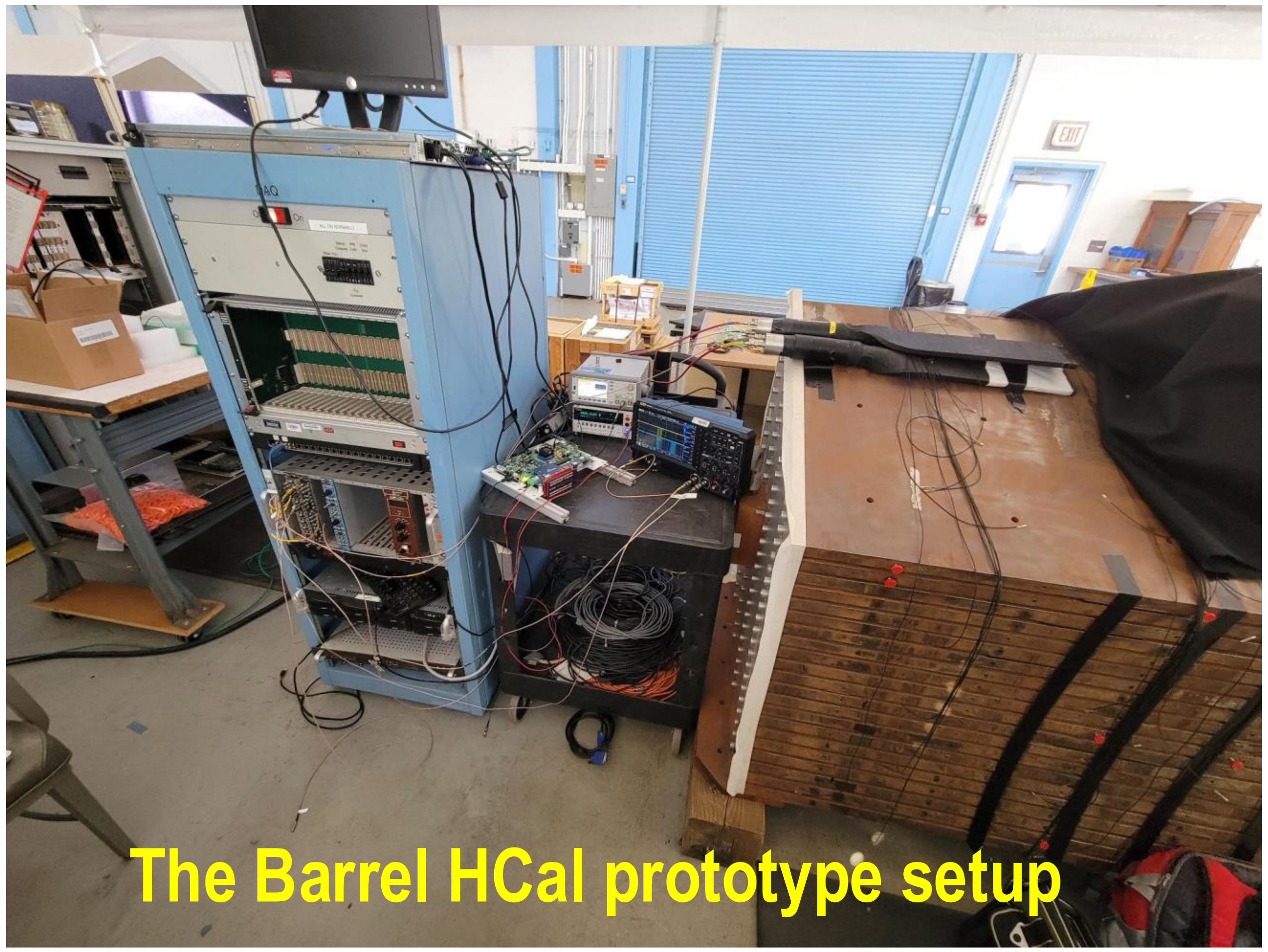

WA98 Experimental Setup
158 A GeV Pb+Pb Collisions
at the CERN SPS
highly segmented Lead-Glass Calorimeter
(identification of photons, π^0 and η-mesons)
Forward-Calorimeter
Calorimeter
(transverse energy)
highly segmented Photon-Multiplicity-Detector
Time of Flight (#2)
(PID of positive hadrons)
Streamer Tubes
Pad Chambers
Goliath Magnet
Target
(inside Plastic-Ball)
Start Counter
33 TeV Pb
Plastic-Ball
(π^+, p, ... He in target region)
Silicon-Pad and Silicon-Drift Detectors
(pseudorapidity-dist. of charged particles)
Multistep Avalanche Chambers
with CCD-readout (tracking of charged particles)
21.5 m
Time of Flight (#1)
(PID of negative hadrons)
Charged Particle Veto-Detector

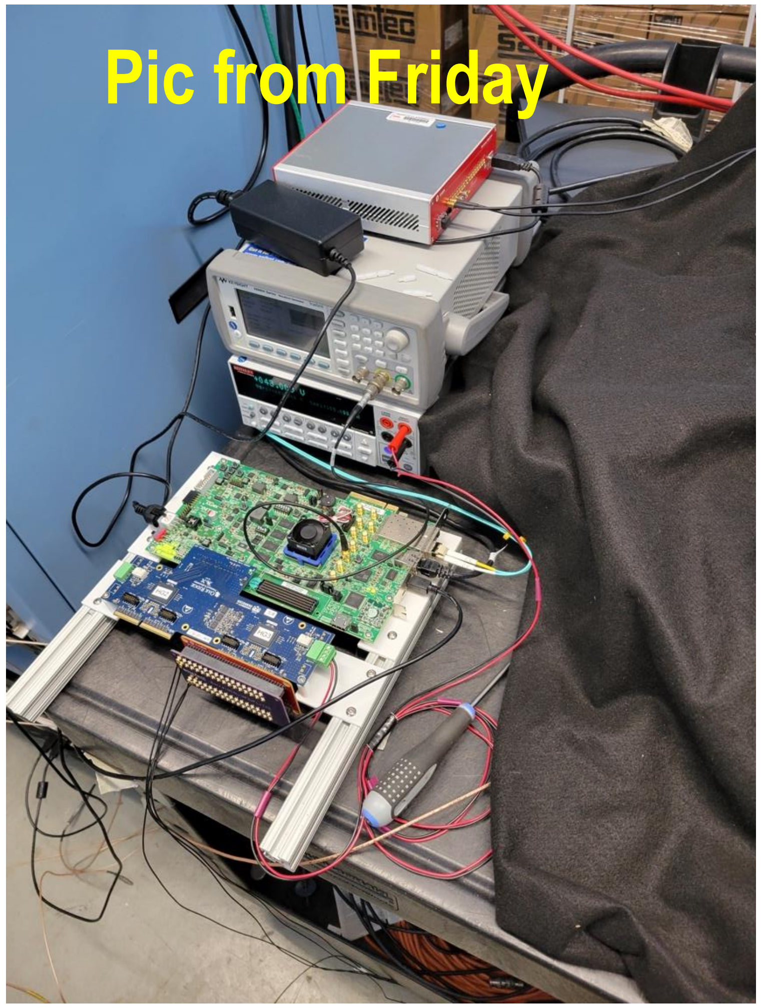
What I'll be talking about today

First, a more generic overview of RCDAQ / monitoring / analysis

Then, some demo targeted towards what we will do

Please interrupt me any time for questions. A lot of it is an actual live demo.





RCDAQ

RCDAQ is DAQ system that has been around since about 2012

It started out as your swiss army knife-type DAQ system to quickly read out whatever you need for your R&D project

It was used in pretty much all R&D campaigns for sPHENIX, but already much earlier in several EIC-themed test beams and other measurements, typically at the Fermilab Test Beam Facility

To the best of my knowledge (some I know, some I learn about when I get questions), it is in use in about 25-30 places around the world

I have step-by-step manuals with examples that seem to work for groups that set up completely autonomously. Just the other day I got a question from an outside group that I didn't know about. They had been using RCDAQ for years and finally had a question.

RCDAQ was chosen to be the main DAQ system of sPHENIX

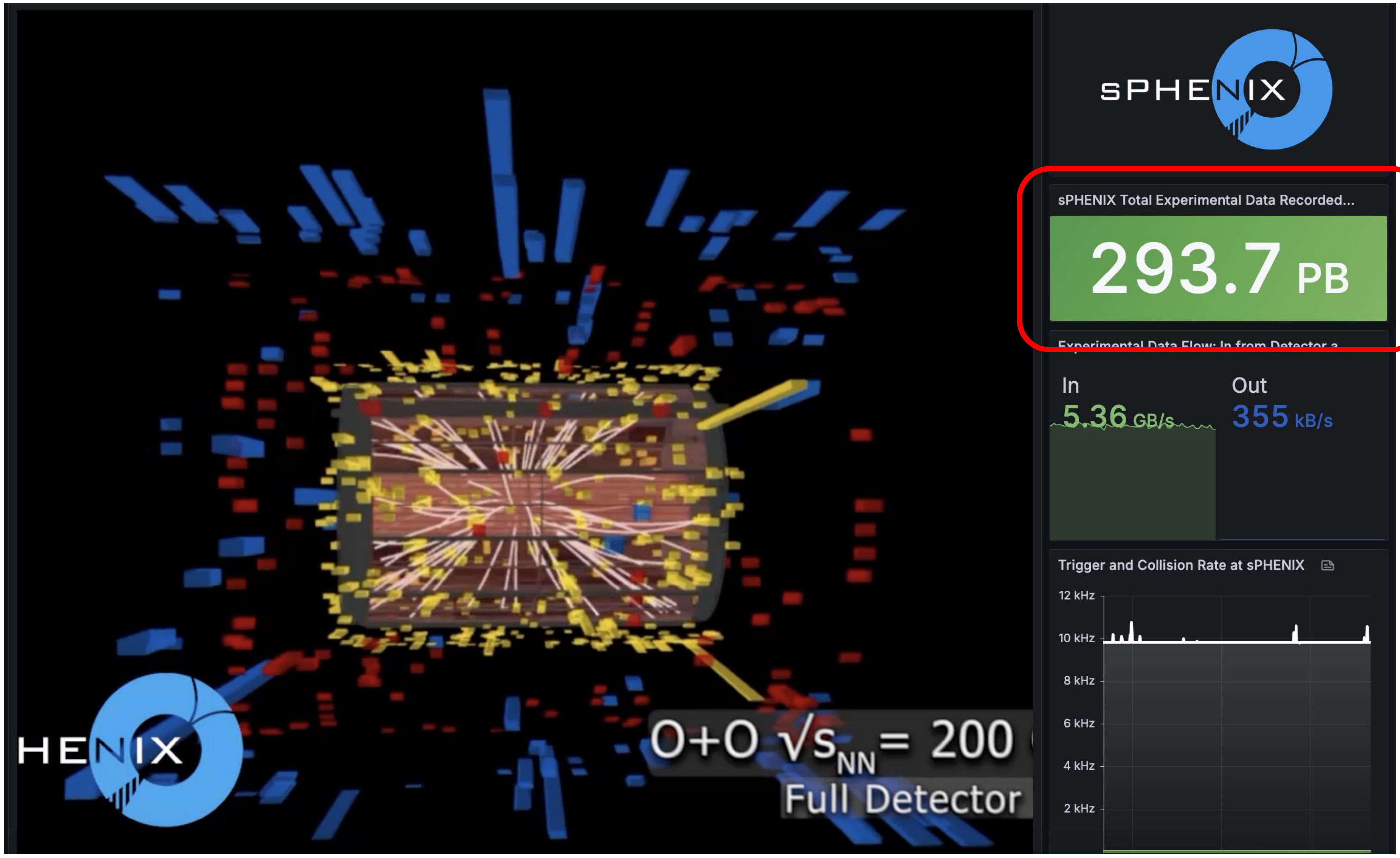
We achieved a sustained data rate of a bit more than 42 Gbyte/s (330GBit/s), about 3x what ePIC intends to do. We ended up taking 293PB of data with sPHENIX and RCDAQ.

And most of it in streaming mode.

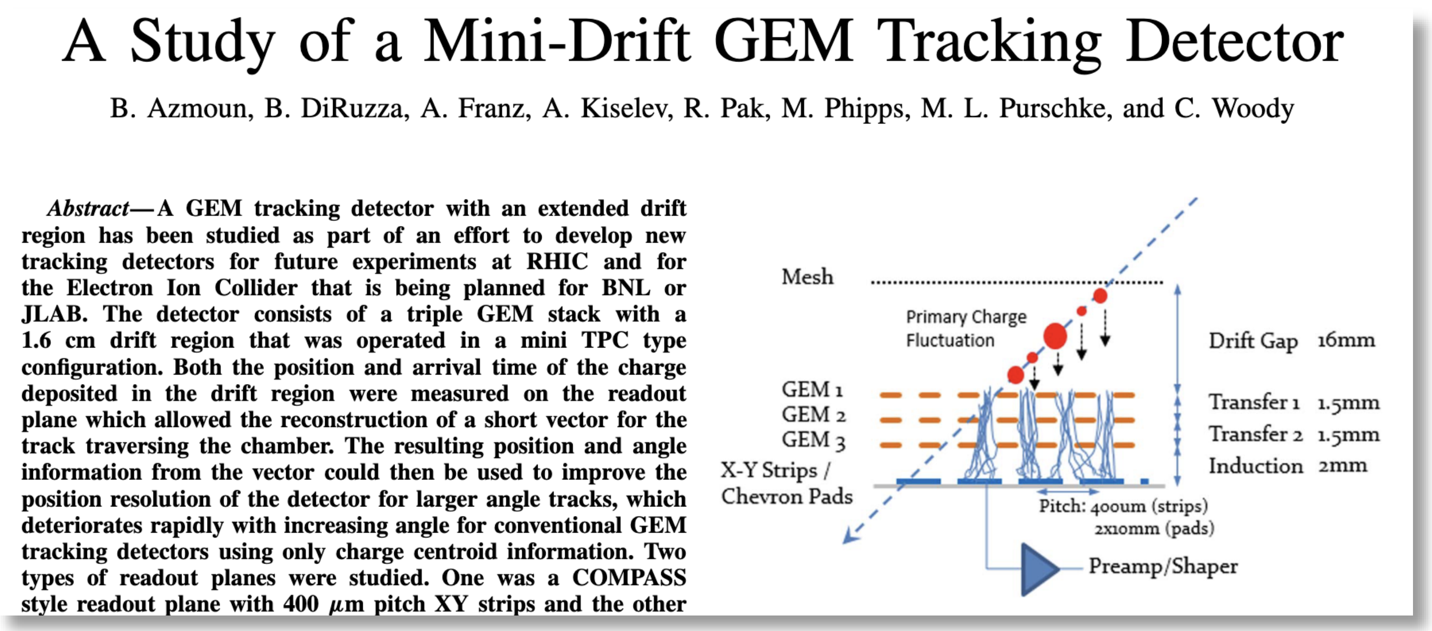
Go to https://www.phenix.bnl.gov/~purschke/rcdaq for the manuals and sample data files etc

A lot of data

Can't stop myself showing this -


HENIX
O+O $\sqrt{s_{NN}}$ = 200
Full Detector
sPHENIX
sPHENIX Total Experimental Data Recorded...
293.7 PB
Experimental Data Flow: In from Detector a
In
5.36 GB/s
Out
355 kB/s
Trigger and Collision Rate at sPHENIX
12 kHz
10 kHz
8 kHz
6 kHz
4 kHz
2 kHz

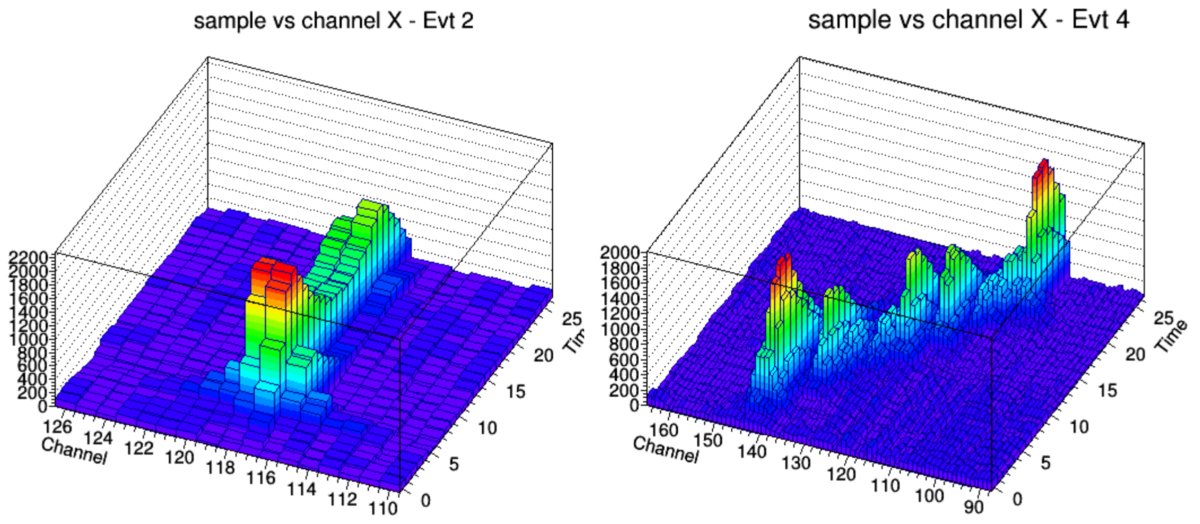
One of the early EIC test beam campaigns with RCDAQ - The Minidrift GEM tracking detector (2014)



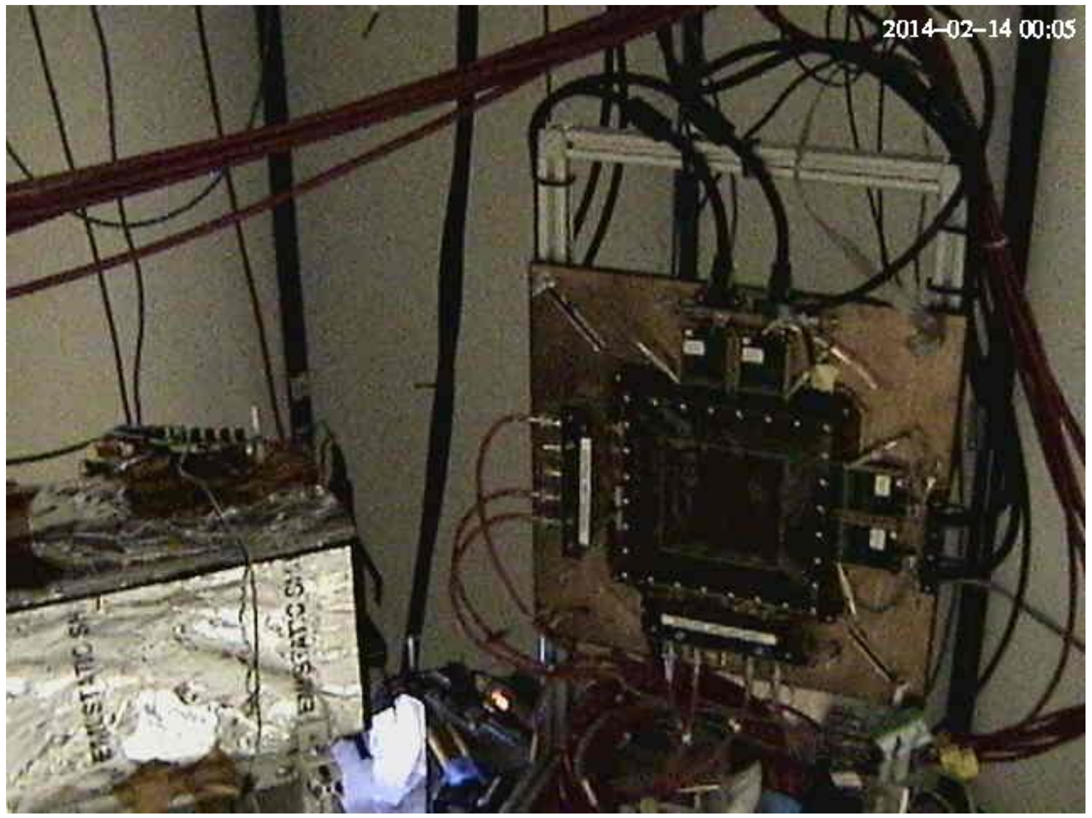
A Study of a Mini-Drift GEM Tracking Detector

B. Azmoun, B. DiRuzza, A. Franz, A. Kiselev, R. Pak, M. Phipps, M. L. Purschke, and C. Woody

Abstract—A GEM tracking detector with an extended drift region has been studied as part of an effort to develop new tracking detectors for future experiments at RHIC and for the Electron Ion Collider that is being planned for BNL or JLAB. The detector consists of a triple GEM stack with a 1.6 cm drift region that was operated in a mini TPC type configuration. Both the position and arrival time of the charge deposited in the drift region were measured on the readout plane which allowed the reconstruction of a short vector for the track traversing the chamber. The resulting position and angle information from the vector could then be used to improve the position resolution of the detector for larger angle tracks, which deteriorates rapidly with increasing angle for conventional GEM tracking detectors using only charge centroid information. Two types of readout planes were studied. One was a COMPASS style readout plane with 400 μm pitch XY strips and the other

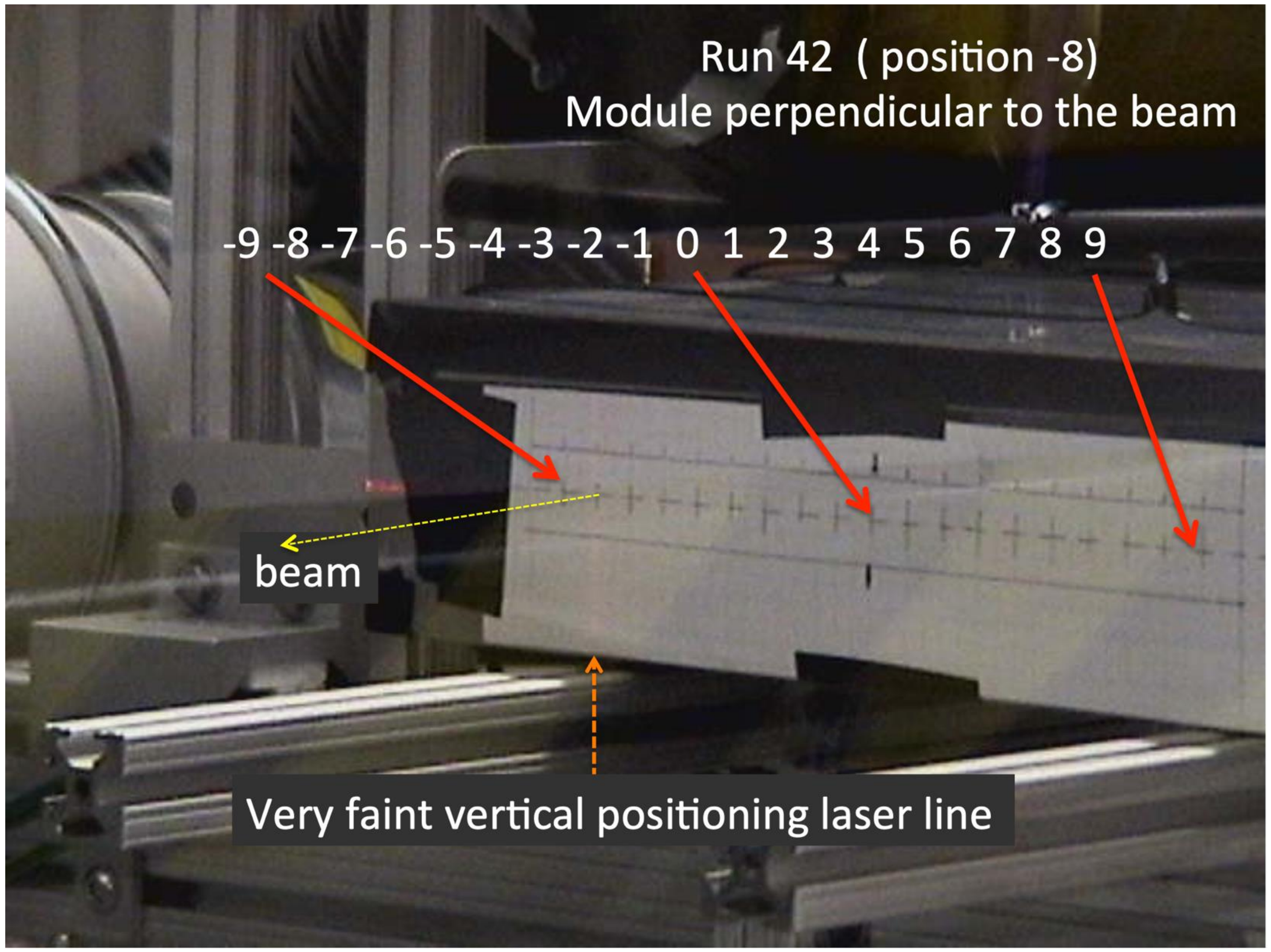


IEEE Transactions on Nuclear Science, vol. 63, no. 3, pp. 1768-1776, June 2016

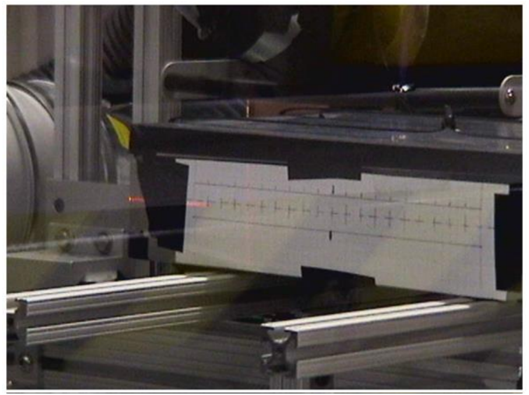


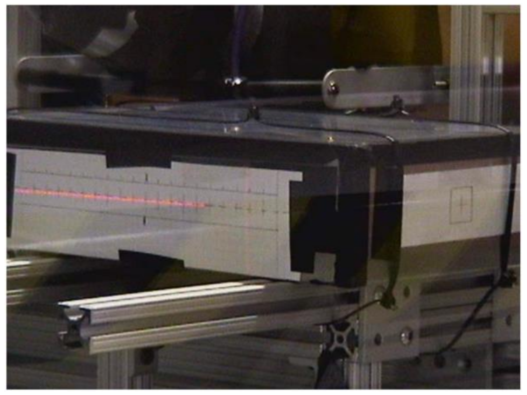
Another one from the EIC calo orbit – Oleg, Craig, myself

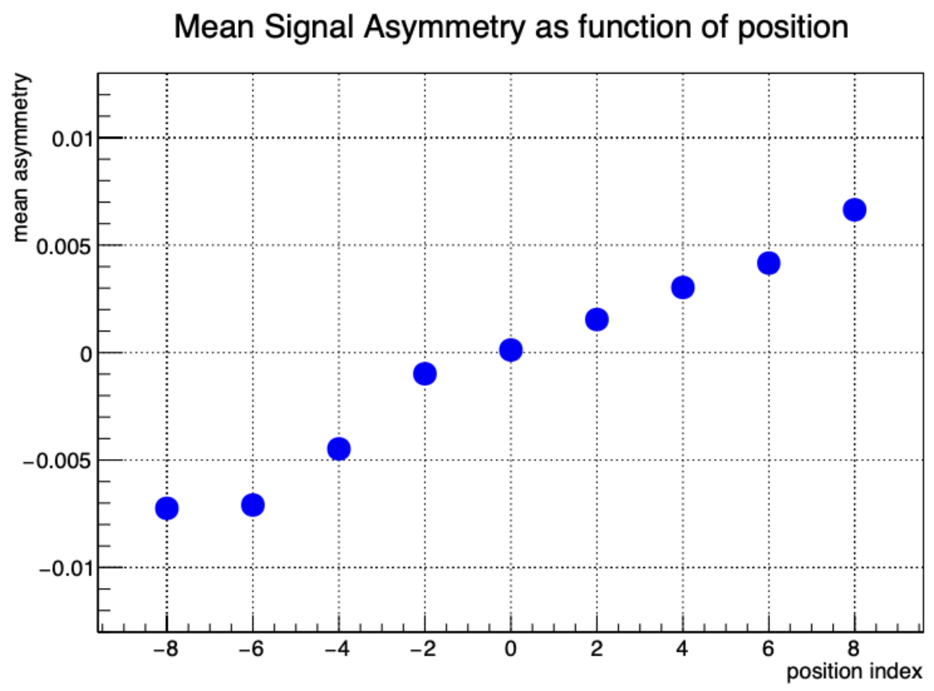
”Can we get more calorimeter position information by adding a dual readout and measure time?”


Run 42 (position -8)
Module perpendicular to the beam
-9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9
beam
Very faint vertical positioning laser line

Moving the calorimeter in the beam for “more left” or “more right” incidence







In the end we found that it’s too small an effect to pursue, but it was worth checking!

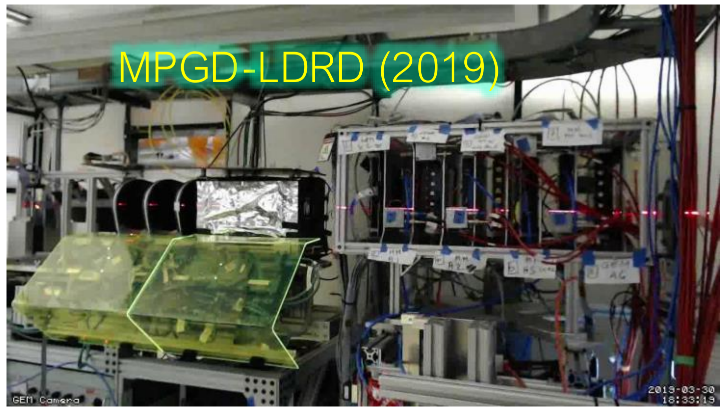
Long History of EIC-themed test beams with RCDAQ

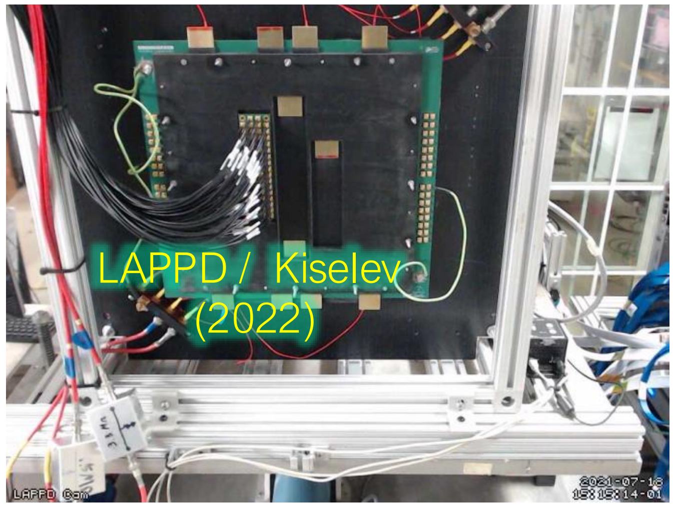
The RCDAQ system has been a pillar of EIC-themed data taking for R&D, test beams etc since 2013 – eRD1, eRD6, LDRDs, …

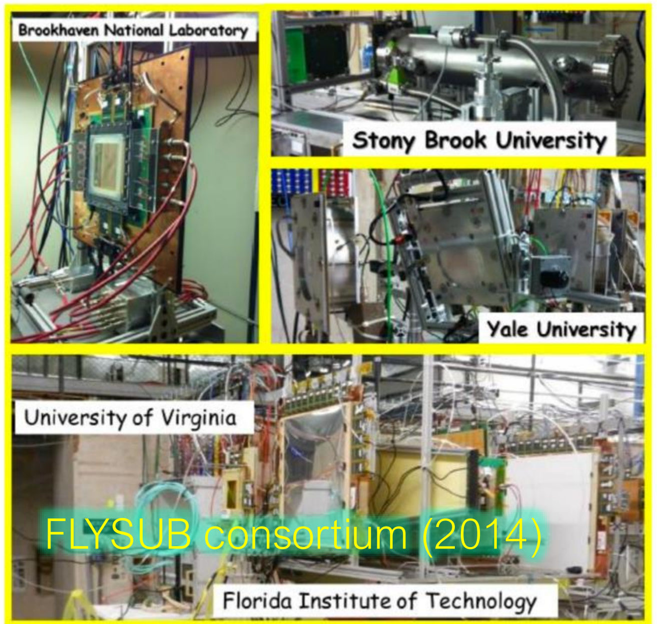
Many active RCDAQ installations in the ePIC orbit + ~30 elsewhere

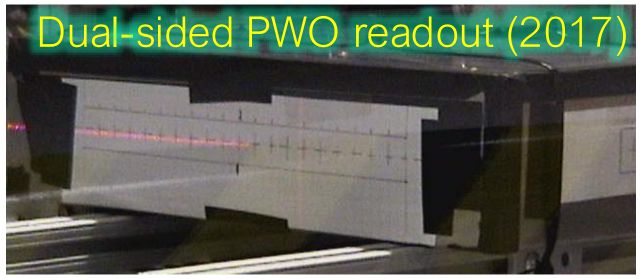
Usual entry by ease-of-use for standard devices (DRS, SRS, CAEN, …) and support for fully automated measurement campaigns

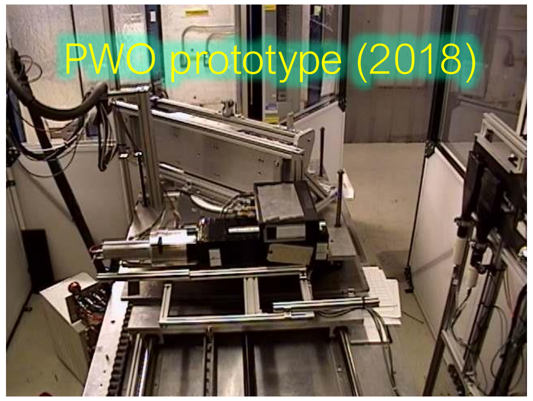
Manuals and “application notes” at https://www.phenix.bnl.gov/~purschke/rcdaq











Thanks to Jin Huang for this collection

What does one need?

Pretty much any Linux machine and distro will do. (I do lots of development on a Linux VM on my M2 Mac here)

I'm striving to be distro-agnostic:

- RedHat and derivatives (RHEL, Fedora, Alma… all good)
- Debian and similar (Ubuntu, Mint, …)
- Arch Linux

And yes, it does run on a **Raspberry Pi** – sometimes you want to take a few weeks worth of cosmics with some detector module without tying up a more expensive PC

The main platforms currently at work in sPHENIX are 96-core AMD EPYC PCs.

For practicing/going through the manuals, you can run everything **without any actual readout hardware** – RCDAQ provides "pretend-devices" that behave like an ADC, for you to be able to kick the tires…

Data Formats in general…

One of the trickiest parts when developing a new application is defining a data format

It can take up easily half of the overall effort – think of Microsoft dreaming up the format to store this very PowerPoint presentation you are seeing in a file. We used to have ppt, now we have pptx – mostly due to limitations in the original format design

A good data format takes design skills, experience, but also the test of time

The tested format usually comes with an already existing toolset to deal with data in the format, and examples – nothing is better than a working example

Case in point: We could easily accommodate the sPHENIX Streaming Readout data in this format, even though no one had ever heard the term when I designed this

Modularity and Extensibility

No one can foresee and predict requirements of a data format 20 years into the future.

Must be able to grow, and be extensible

The way I like to look at this:

FedEx (and UPS) cannot possibly know how to ship every possible item under the sun

But they know how to ship a limited set of box formats and types, and assorted weight parameters and limits

Whatever fits into those boxes can be shipped

During transport, they only look at the label on the box, not at what's inside

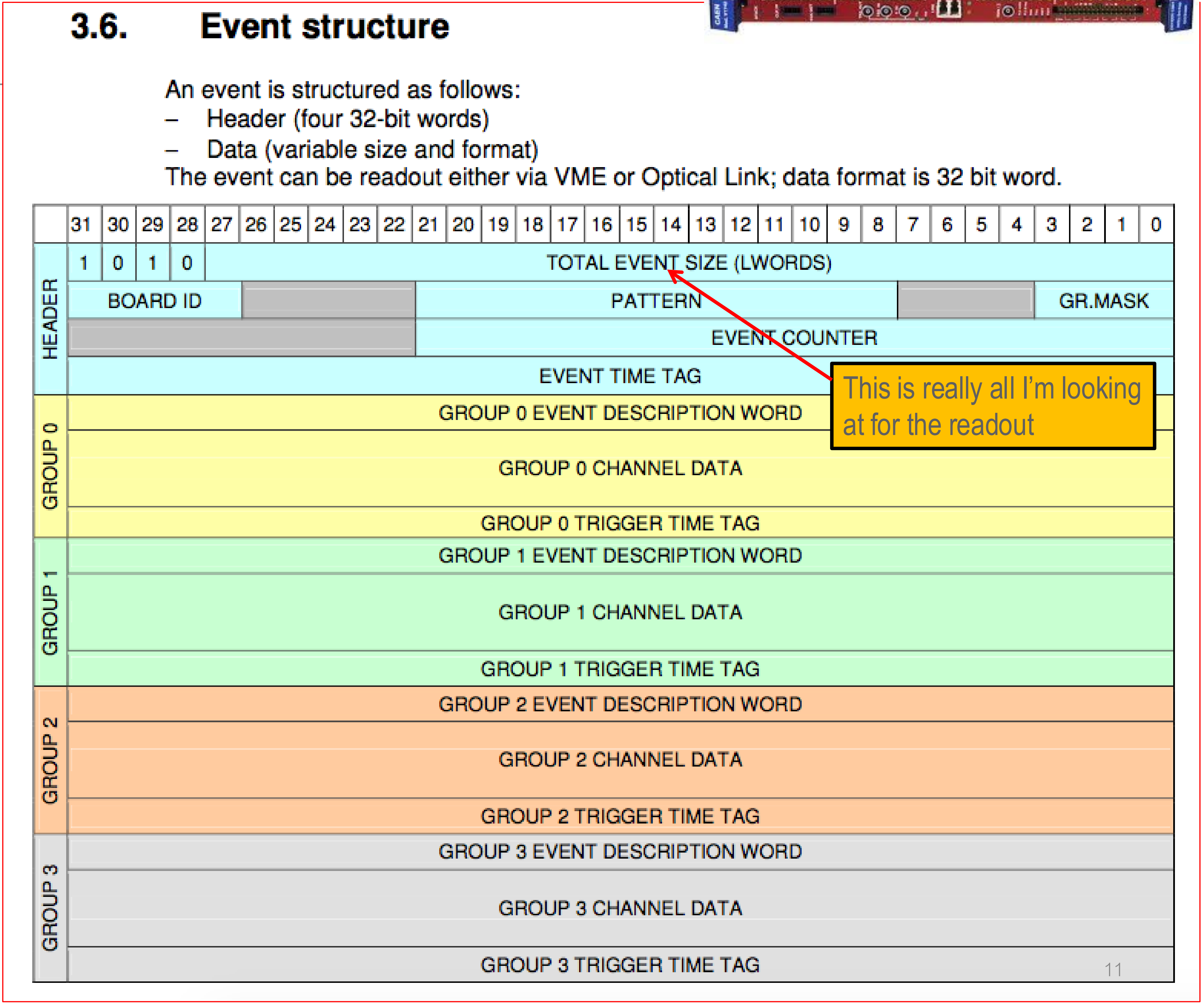
We will see a surprisingly large number of similarities with that approach in a minute

Example: CAEN's V1742/DT5743 format

We just take that blob of memory, "put it in a box", done.

The analysis software takes care of the unpacking and interpretation later

Just grab it. Don't waste time here.



3.6. Event structure

An event is structured as follows:

– Header (four 32-bit words)
– Data (variable size and format)

The event can be readout either via VME or Optical Link; data format is 32 bit word.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HEADER	1	0	1	0	TOTAL EVENT SIZE (LWORDS)																											
	BOARD ID										PATTERN																		GR.MASK			
											EVENT COUNTER																					
	EVENT TIME TAG																															
GROUP 0	GROUP 0 EVENT DESCRIPTION WORD																															
GROUP 0	GROUP 0 CHANNEL DATA																															
GROUP 0	GROUP 0 TRIGGER TIME TAG																															
GROUP 1	GROUP 1 EVENT DESCRIPTION WORD																															
GROUP 1	GROUP 1 CHANNEL DATA																															
GROUP 1	GROUP 1 TRIGGER TIME TAG																															
GROUP 2	GROUP 2 EVENT DESCRIPTION WORD																															
GROUP 2	GROUP 2 CHANNEL DATA																															
GROUP 2	GROUP 2 TRIGGER TIME TAG																															
GROUP 3	GROUP 3 EVENT DESCRIPTION WORD																															
GROUP 3	GROUP 3 CHANNEL DATA																															
GROUP 3	GROUP 3 TRIGGER TIME TAG																															

This is really all I'm looking at for the readout

RCDAQ - The High Points

Each interaction with RCDAQ is a **shell command**. There is no such thing as “starting an application and issuing internal commands” (think of your interaction with, say, root)

RCDAQ out of the box doesn’t know about any particular hardware. All knowledge how to read out something, say, a FELIX card, comes by way of a **plugin** that teaches RCDAQ how to do that.

That makes RCDAQ highly portable and also **distributable** – some sPHENIX FEMs use commercial drivers for the readout; I cannot re-distribute CAEN software, etc etc

RCDAQ has **no proprietary-format configuration** files. (huh? In a minute).

Support for different **event types**

Built-in support for standard **online monitoring**

Built-in support for **electronic logbooks** (Stefan Ritt’s Elog)

Network-transparent control interfaces

Everything is a shell command…

One of the most important features. Any command is no different from “ls –l“ or “cat”

Everything is inherently scriptable

You have the full use of the shell’s capabilities for if-then constructs, error handling, loops, automation, cron scheduling, and a myriad of other ways to interact with the system

In that sense, there are no proprietary configuration files – only configuration *scripts.*

This is quite different from “my DAQ supports scripts”!

I do not want to be trapped within the limited command set of any application!

With shell commands, the DAQ is fully integrated into your existing work environment

(And yes there are GUIs – many GUIs. They usually just trigger the appropriate command)

On Autopilot - Scripts at work

Very often – especially in your R&D days – you want to step through a range of values of a configuration parameter and see what your detector prototype has to say

- Bias voltage scans (we characterized gazillions of SiPMs)
- Position scans
- Temperature scans
- And on and on

Such a measurement is best done in a script that reads predetermined positions / voltage settings / what have you and performs the measurement

I picked an example: What is the response uniformity of a calorimeter module when a shower develops in different places? (We were very worried about this)

We were simulating different shower positions by “writing light with a light fiber” on the module front face

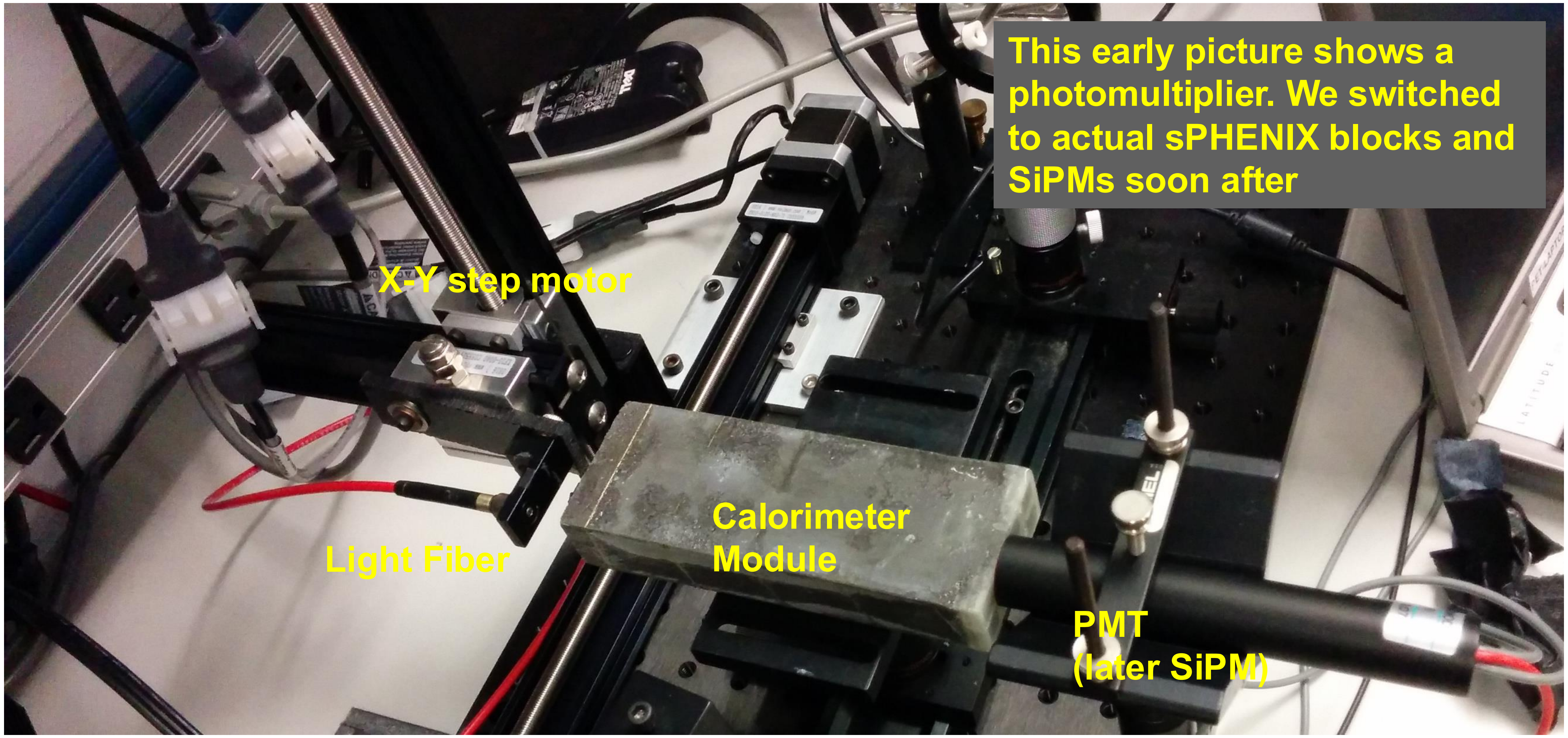
Measurements on autopilot through scripting

Simulate shower incidence positions by moving a light fiber in x and y

take a run for each position w/ 4000 events = 4000 LED pulses

50 x 25 = 1250 positions (later we had 70x70, you really want to automate that)

Let it run overnight, come back in the morning, look at the data


This early picture shows a photomultiplier. We switched to actual sPHENIX blocks and SiPMs soon after
X-Y step motor
Calorimeter Module
Light Fiber
PMT (later SiPM)

The Script

The DAQ operation becomes an integral part of your shell environment

Automatic end after 4000 events

25 positions in y

move the Y motor

50 positions in x

move the x motor

start the DAQ

next x

next y

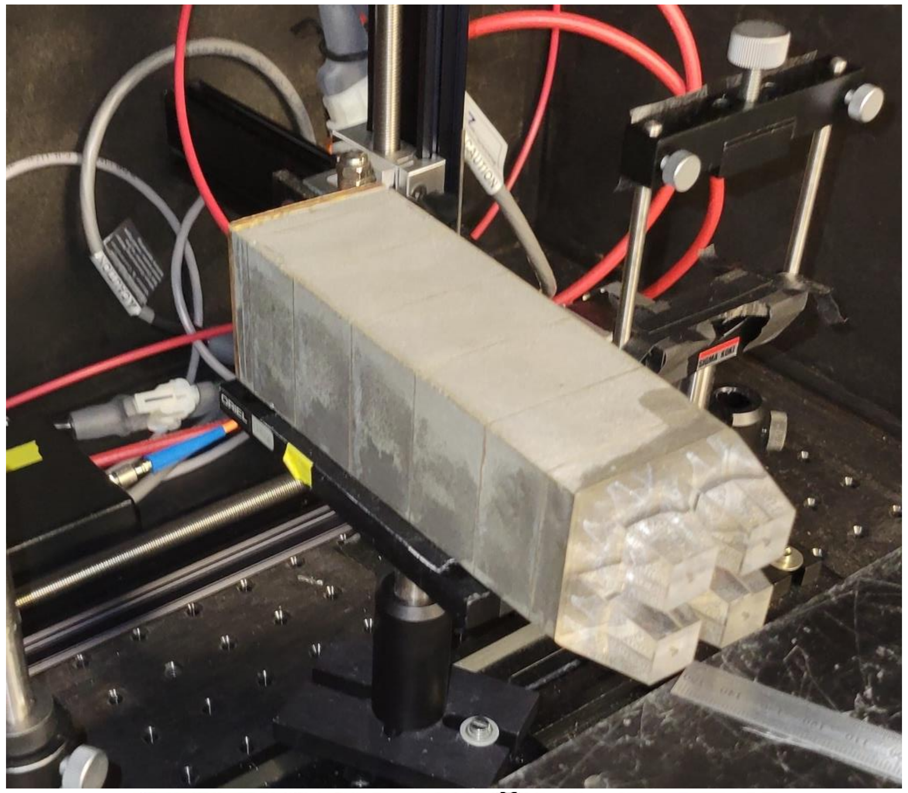
```
#! /bin/sh
STARTPOSX=0
STARTPOSY=9900
INCREMENTX=200
INCREMENTY=-200

CURRENTPOSY=$STARTPOSY
rcdaq_client daq_set_maxevents 4000

for posy in $(seq 25) ; do

    quickmove.sh $CURRENTPOSY 2
    sleep 5
    CURRENTPOSY=$( expr $CURRENTPOSY + $INCREMENTY)
    CURRENTPOSX=$STARTPOSX

    for posx in $(seq 50) ; do

      echo "moving to $CURRENTPOSX"
      quickmove.sh $CURRENTPOSX 1
      sleep 5

      rcdaq_client daq_begin
      wait_for_run_end.sh

       CURRENTPOSX=$( expr $CURRENTPOSX + $INCREMENTX)
    done
done
```

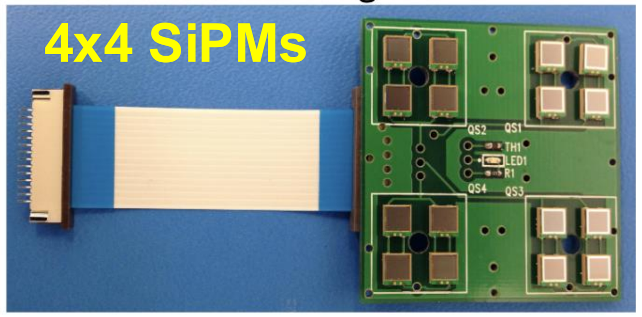
There’s a lot more to this…

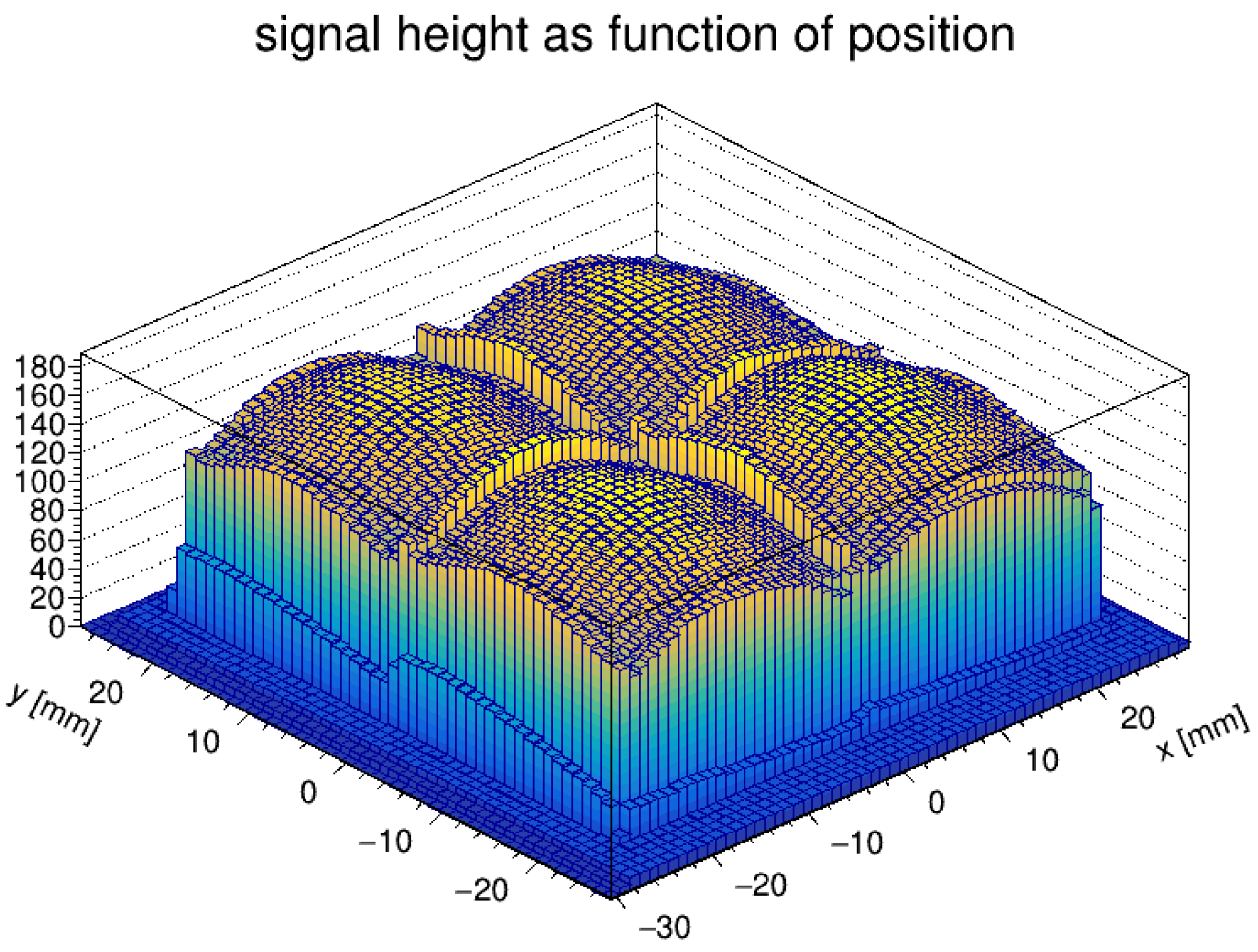
We also added more things to this RCDAQ setup to make the next-day analysis super-easy.

I first need to explain a few things; we get back to this later.

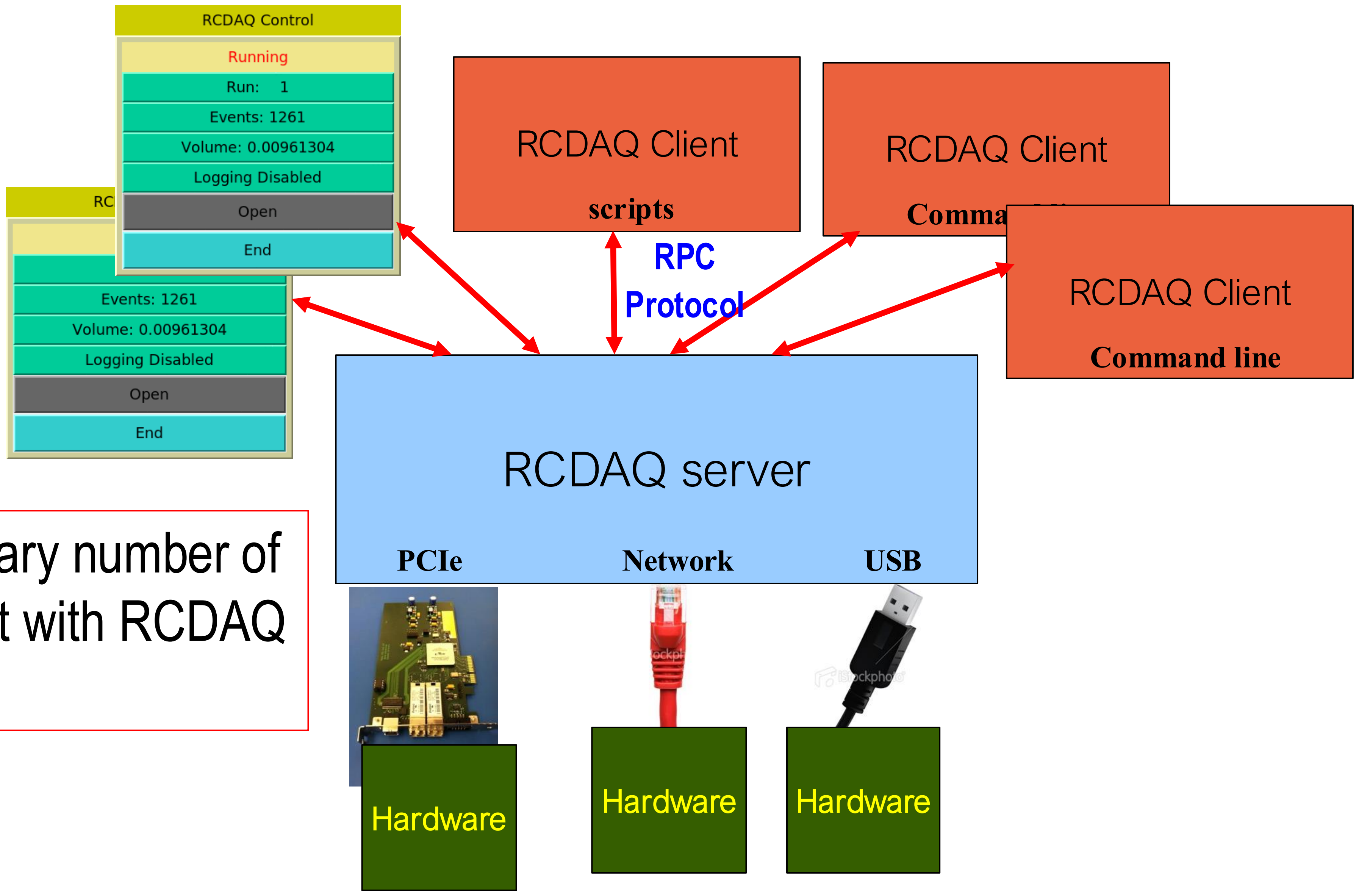
This is the outcome of a later 70x70 position scan of a “real” calorimeter module







The RCDAQ client-server concept



This allows an arbitrary number of processes to interact with RCDAQ concurrently

These are the 3 fundamental ways for data to get into a PC –

- PCIExpress
- Network
- USB

Multiple RCDAQs – almost always with streaming readout

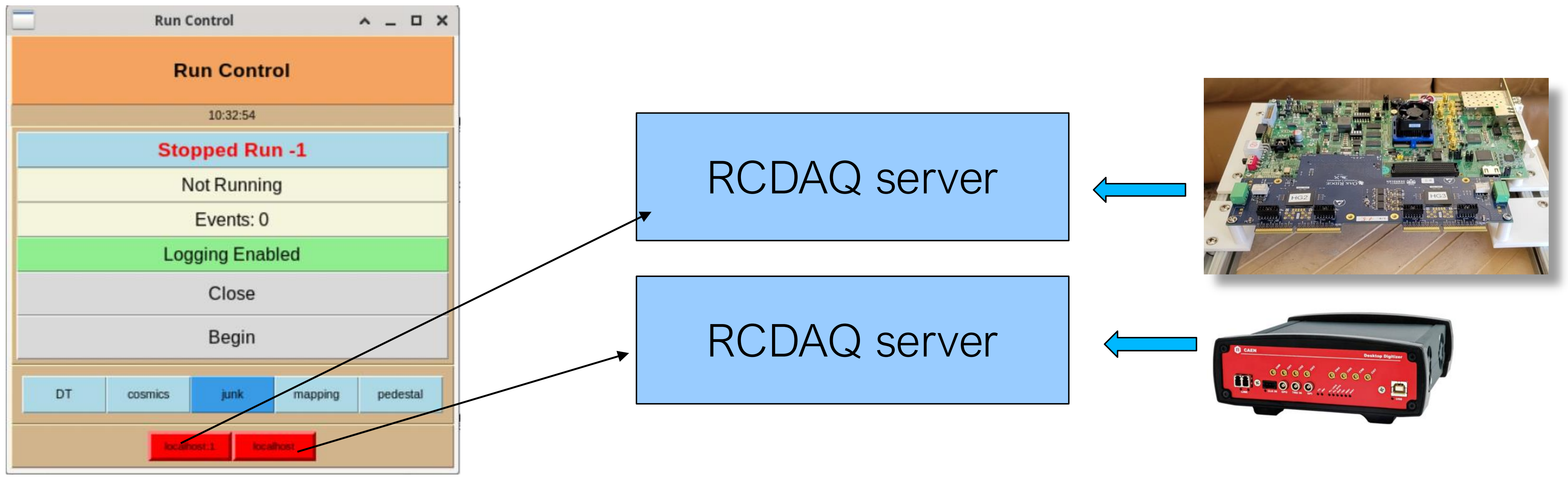
Here you have multiple RCDAQ Servers, each of them reading out their part of your streaming detector(s)

Instead of individually controlling one RCDAQ instance, we have “Run Control” that controls your “cohort” of rcdaq servers.

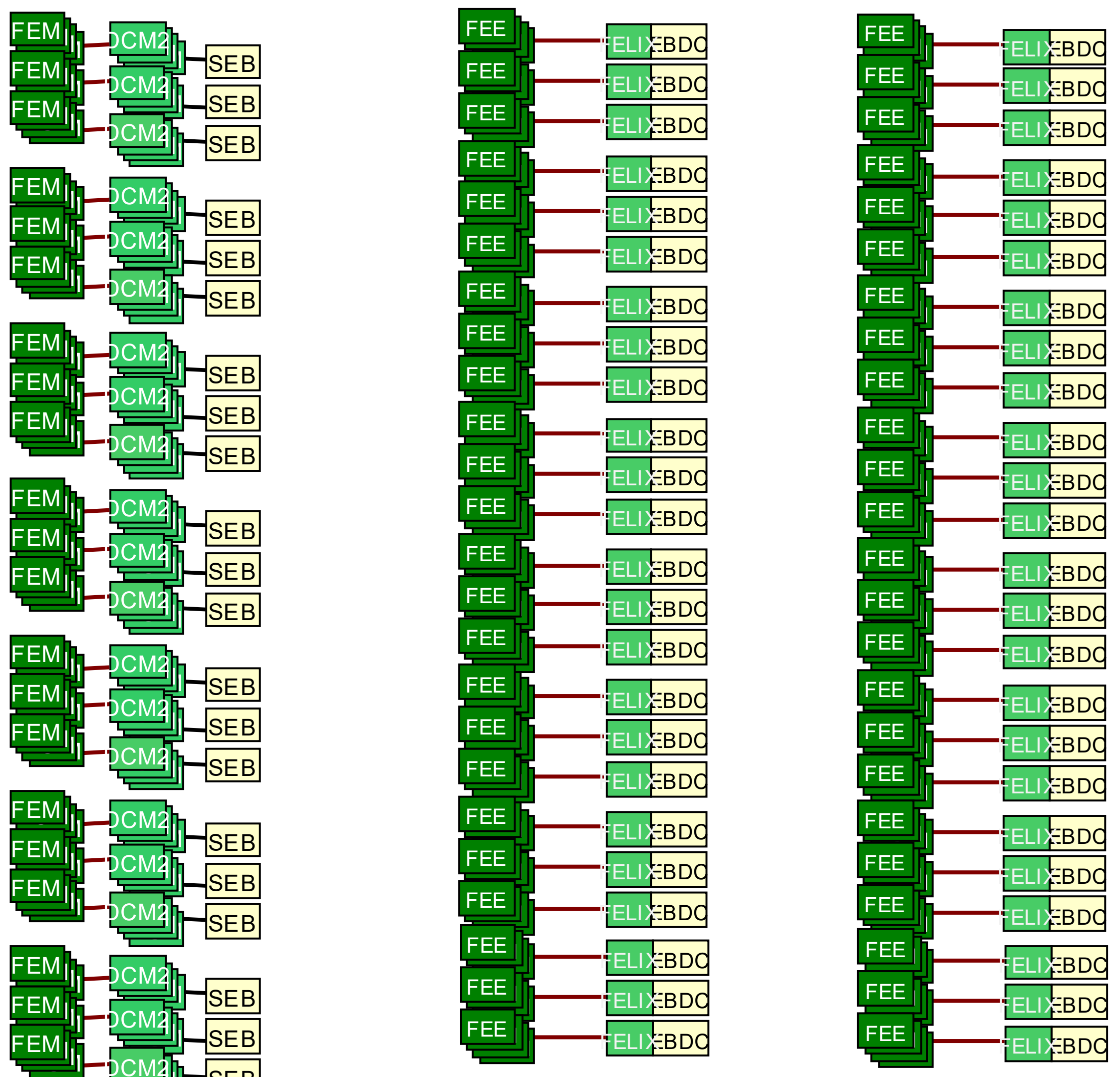
Here I’m showing 2 RCDAQ instances from the BHCal setup

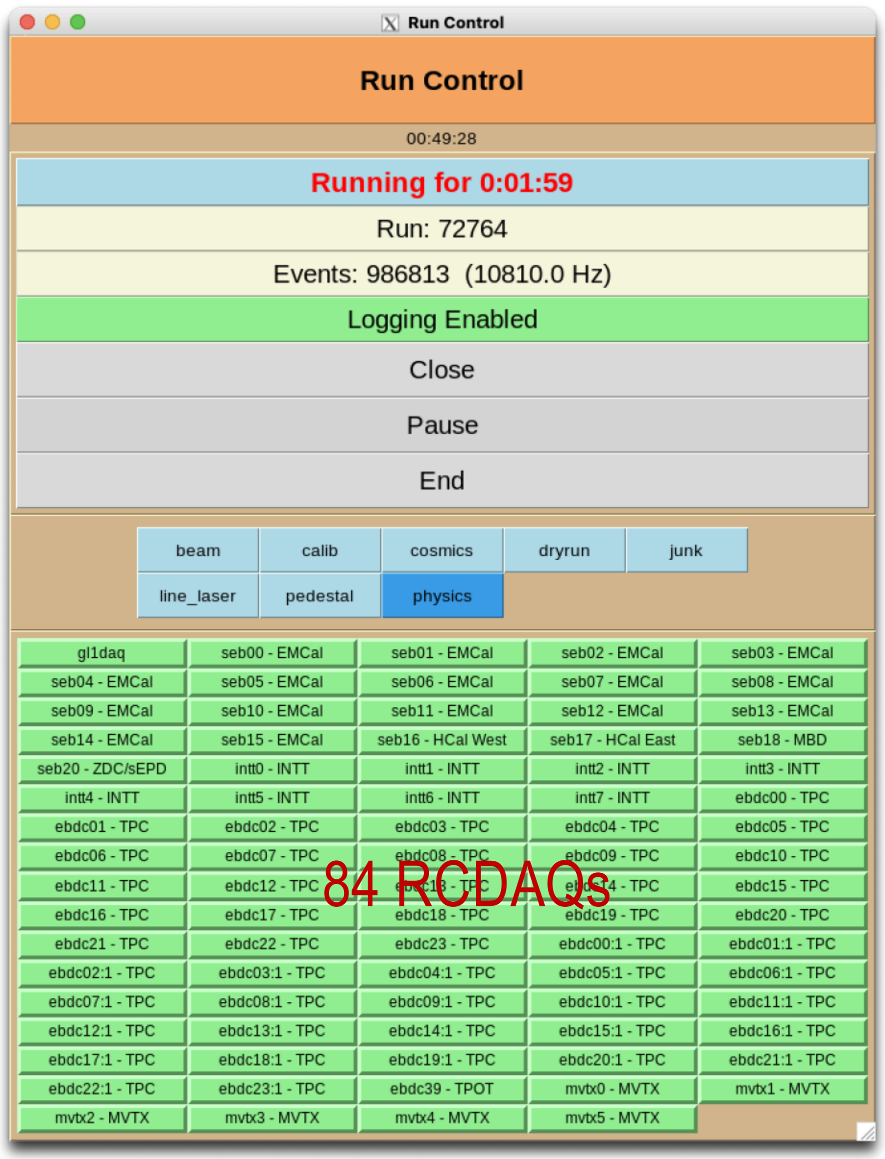
One is reading out the H2GCROC

One is reading a CAEN DT5743 waveform sampler (which we will use at Jlab – yes?)



Here is the sPHENIX setup





00:49:28

Running for 0:01:59

Run: 72764

Events: 986813 (10810.0 Hz)

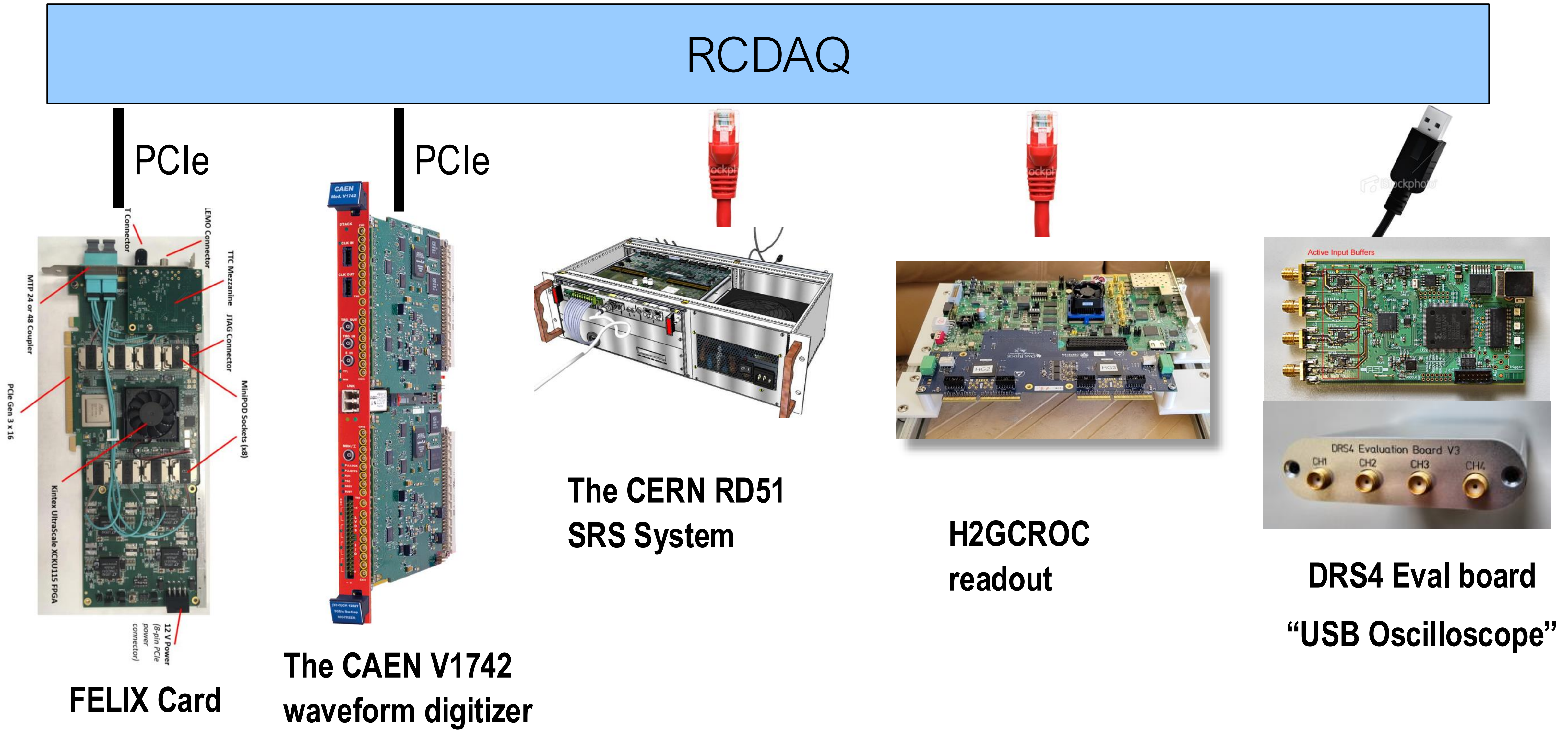
Logging Enabled

gl1daq	seb00 - EMCal	seb01 - EMCal	seb02 - EMCal	seb03 - EMCal
seb04 - EMCal	seb05 - EMCal	seb06 - EMCal	seb07 - EMCal	seb08 - EMCal
seb09 - EMCal	seb10 - EMCal	seb11 - EMCal	seb12 - EMCal	seb13 - EMCal
seb14 - EMCal	seb15 - EMCal	seb16 - HCal West	seb17 - HCal East	seb18 - MBD
seb20 - ZDC/sEPD	intt0 - INTT	intt1 - INTT	intt2 - INTT	intt3 - INTT
intt4 - INTT	intt5 - INTT	intt6 - INTT	intt7 - INTT	ebdc00 - TPC
ebdc01 - TPC	ebdc02 - TPC	ebdc03 - TPC	ebdc04 - TPC	ebdc05 - TPC
ebdc06 - TPC	ebdc07 - TPC	ebdc08 - TPC	ebdc09 - TPC	ebdc10 - TPC
ebdc11 - TPC	ebdc12 - TPC	ebdc13 - TPC	ebdc14 - TPC	ebdc15 - TPC
ebdc16 - TPC	ebdc17 - TPC	ebdc18 - TPC	ebdc19 - TPC	ebdc20 - TPC
ebdc21 - TPC	ebdc22 - TPC	ebdc23 - TPC	ebdc00:1 - TPC	ebdc01:1 - TPC
ebdc02:1 - TPC	ebdc03:1 - TPC	ebdc04:1 - TPC	ebdc05:1 - TPC	ebdc06:1 - TPC
ebdc07:1 - TPC	ebdc08:1 - TPC	ebdc09:1 - TPC	ebdc10:1 - TPC	ebdc11:1 - TPC
ebdc12:1 - TPC	ebdc13:1 - TPC	ebdc14:1 - TPC	ebdc15:1 - TPC	ebdc16:1 - TPC
ebdc17:1 - TPC	ebdc18:1 - TPC	ebdc19:1 - TPC	ebdc20:1 - TPC	ebdc21:1 - TPC
ebdc22:1 - TPC	ebdc23:1 - TPC	ebdc39 - TPOT	mvtx0 - MVTX	mvtx1 - MVTX
mvtx2 - MVTX	mvtx3 - MVTX	mvtx4 - MVTX	mvtx5 - MVTX	

It uses a synchronized "cohort" of 84 individual RCDAQ instances, each reading a particular part of the detector

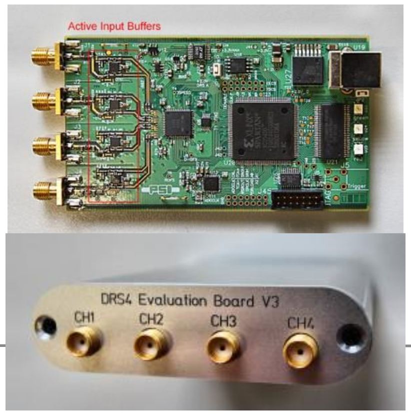
Held together by "Run Control", like a conductor keeping everyone in line

Some workhorse devices implemented in RCDAQ



There are *many* more not shown (all told, there are plugins for about 60)
Many devices that you can often find in your institute already, or in the CAEN catalog

Setting up and reading out a DRS4 Eval board



Do you remember me saying that each RCDAQ interaction is a shell command?

```
$ rcdaq_client load librcdaqplugin_drs.so

$ rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 120 3

$ daq_open

$ daq_begin

  # wait a while…

$ daq_end
```

BTW, this double-dash is shell standard for "stop processing command line options"

Else the later "-150" would be interpreted as -1 -5 -0 options like "ls –ltr".

Try to delete a file named "-l"…

This is a super-simple device, so all the setup can be done on the "create" command line

Not normally the case. In short: we are reding out a data event (1) with the DRS going to packet 1001, triggering (like a scope) on ch 1, -150 threshold, negative slope, move the pulse on the x-axis by 120 samples "right", and setup 5GS/s

What’s the significance of that “1001”?

It is literally just a packet ID number that you are free to pick.

As a convention (nothing more) we pick numbers above 1000

Here we are just playing around, so 1001 is as good as any

That packet number identifies a particular readout unit (say, a FELIX card/ ROC card, what have you) and so it identifies a particular piece of real estate of your detector

Once assigned, one should never change that assignment

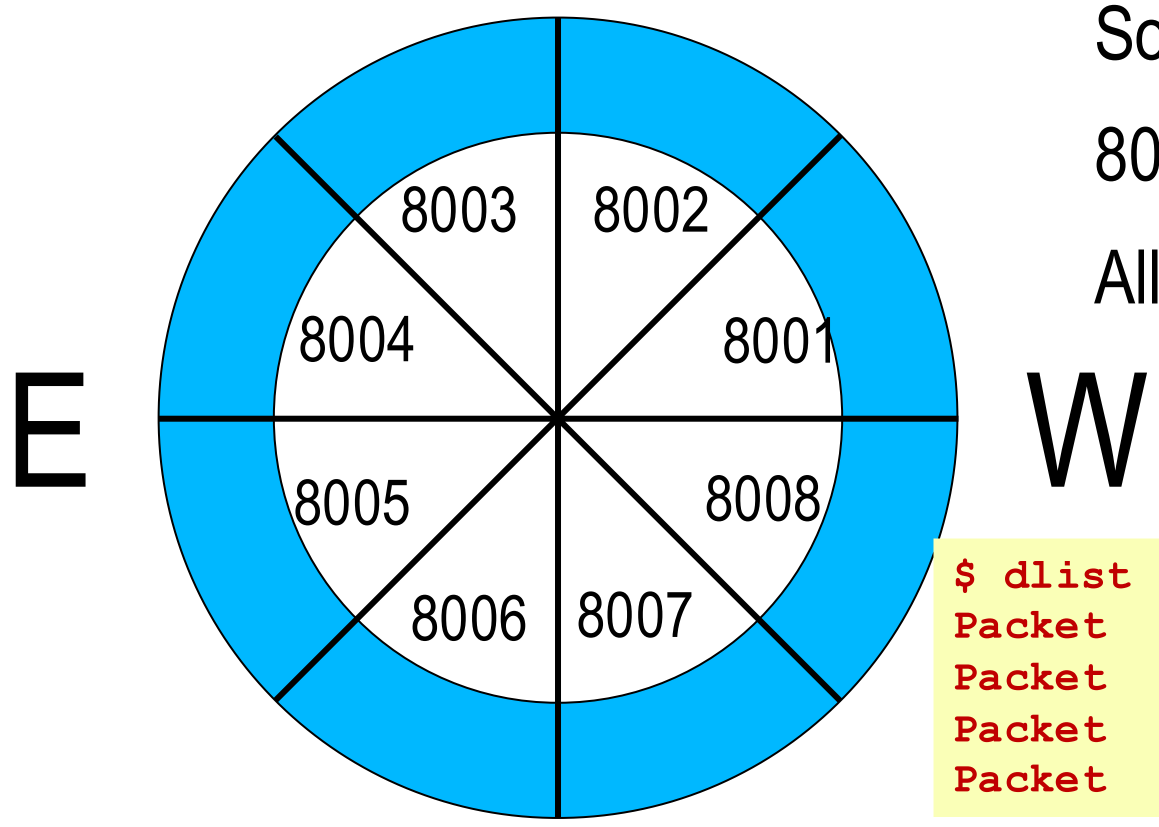
They are still just numbers… in both PHENIX and sPHENIX we imposed a numbering scheme where each detector system was enumerated (MBD 1, MVTX 2, INTT 3, TPC 4, …. and so on… outer HCal = 8)

sPHENIX has o(250) packets in total

PacketID assignment Example (BHCal)

Each detector then can assign packet IDs in the detector_nr * 1000 +1 …. d_nr * 1000 +999 range

8001… 8999 for the Hcal (the “even 1000” is reserved). Much more than we need



So we have 8001, 8002, 8007 and 8008 on the west side.

8003, 8004, 8005, 8006 on the east side

All 8 together constitute the data of the entire outer HCal

```
$ dlist /bbox/bbox6/W/HCal/cosmics/cosmics_seb16-064906-0000.prdf
Packet    8001    788   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
Packet    8002    676   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
Packet    8007    753   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
Packet    8008    753   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
```

```
$ dlist  /bbox/bbox7/W/HCal/cosmics/cosmics_seb17-064906-0000.prdf
Packet    8003    746   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
Packet    8004    732   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
Packet    8005    662   0 (Unformatted)    172  (IDDIGITIZERV3_12S)
Packet    8006    690   0 (Unformatted)    172  (IDDIGITIZERV3_12S))
```

The channel mapping for one device /packet repeats for each packet – super-easy to deal with

Meta Data Capturing

In the “real” experiment that’s running for a few years (think sPHENIX, ATLAS, what have you) you are embedded in an environment that supports all sorts of record keeping

At a test beam or you in your lab needs a different kind of “record keeping support”

What was the temperature? Was the light on? What was the HV? What was the position of that X-Y positioning table?

We capture this information in the raw data file itself and **the data cannot get lost**

I often add a webcam picture to the data so we have a visual confirmation that the detector is in the right place, or something

A picture captures everything…

Let me show you how we always capture the RCDAQ setup itself

For that I need to introduce “Event Types”…

Reading different things with different Event Types

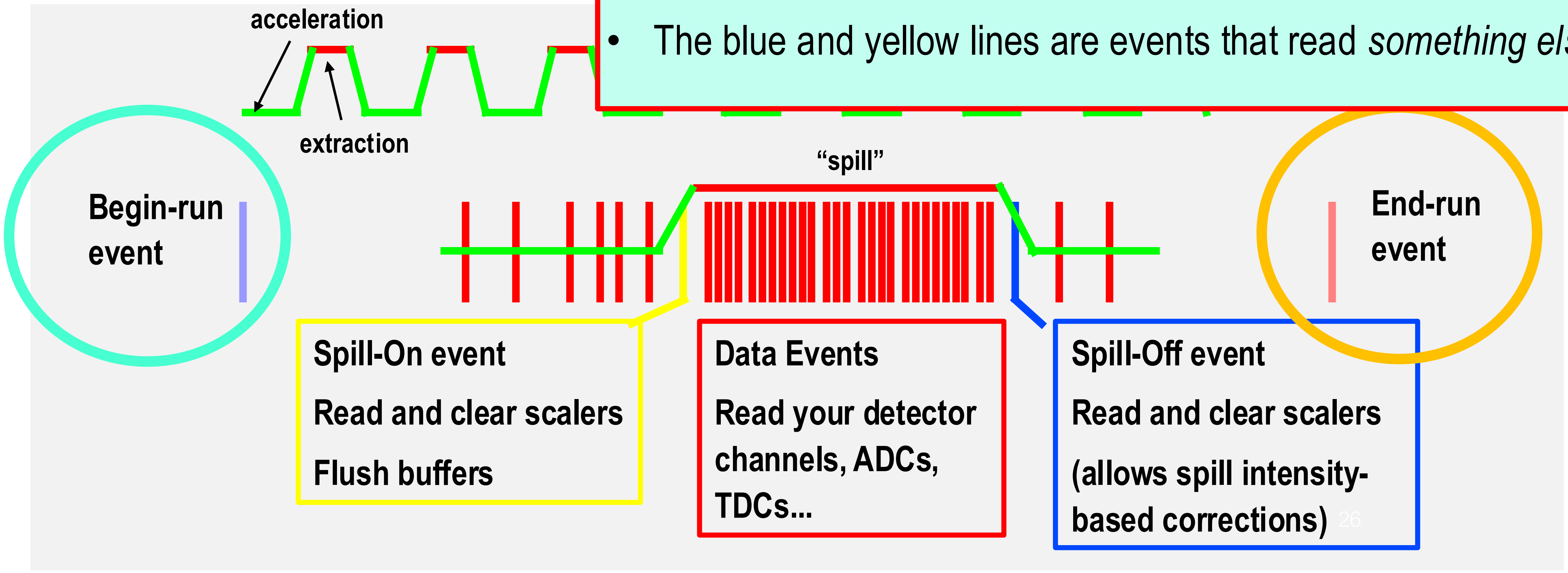
You would think of the DAQ as "reading your detector"

Almost always, it is necessary to r

Let's go to the CERN-SPS (or the

Let's recap here:

- The red lines are events that you would expect (read your detector)
- The blue and yellow lines are events that read *something else*



Spill-On event
Read and clear scalers
Flush buffers

Data Events
Read your detector channels, ADCs, TDCs...

Spill-Off event
Read and clear scalers
(allows spill intensity-based corrections)

In addition to your data, you need information about the spill itself – each one is different

You need to make intensity-dependent corrections on a spill-by-spill basis

So you put some signals on scalers and get an idea about the intensity, dead times, microstructures, etc

Why are begin- and endrun events so important?

In each run, they are generated exactly once

This is the place where you can keep information that is needed/valid for the entire run

Like the position information of the x/y motors that you saw before - they would get stored in the begin run.

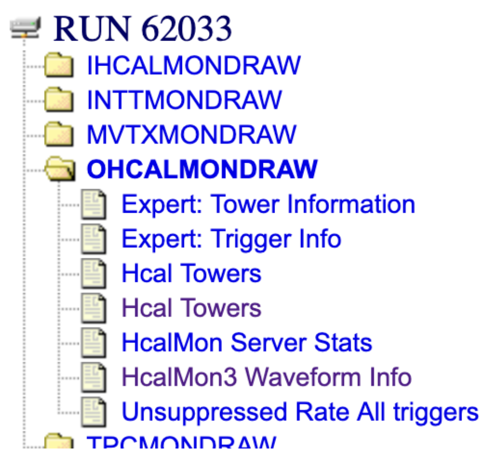
Ditto for configuration information – we always make it so that we store it in the begin-run.

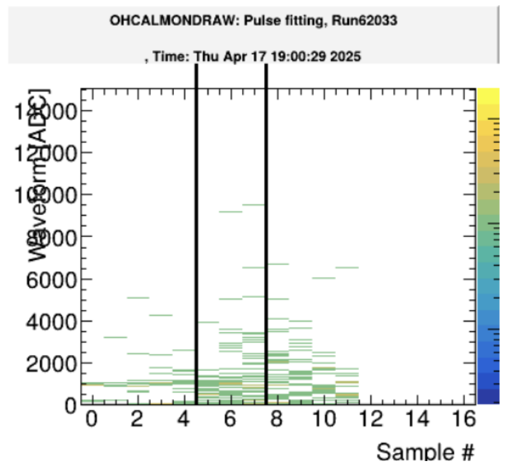
RCDAQ guarantees that the begin-run is the first, and the end-run is the last event in that run

If nothing else, if you treat your data as a stream (think online monitoring – doesn't stop, sees many runs), these serve as convenient markers that a new run has begun, or a run has ended.

- On receipt of the begin-run event, you typically clear all your monitoring histograms
- On receipt of the end-run event, you save your histograms to a file

This is actually how all this online monitoring history is generated!





Let’s capture meta-data now!

This was our typed-in example from before

```
$ rcdaq_client load librcdaqplugin_drs.so

$ rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 120 3
```

Now you put this into a script so you always get the same setup:

```
#! /bin/sh

rcdaq_client load librcdaqplugin_drs.so

rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 120 3
```

Capturing the setup script itself for posterity

We add this very setup script file into our begin-run event for posterity

This "device" captures a file as text into a packet

This "9" is the event type of the beg-run

And this refers to the name of the file itself

```
#! /bin/sh
rcdaq_client create_device device_file 9 900 "$0"
rcdaq_client load librcdaqplugin_drs.so
rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 120 3
```

So this gets added as packet with id 900 in the begin-run

It's not quite right yet - $0 is usually just "setup.sh", so the server may not be able to find it.

Let me show the "end product":

A typical RCDAQ Setup Script

```
#! /bin/sh
# this sets up the DRS4 readout with 5GS/s, a negative
# slope trigger in channel 1 with a delay of 140

if ! rcdaq_client daq_status > /dev/null 2>&1 ; then
    echo "No rcdaq_server running, starting..."
    rcdaq_server > $HOME/rcdaq.log 2>&1 &
    sleep 2
fi
MYSELF=$(readlink -f $0)
rcdaq_client daq_clear_readlist

rcdaq_client create_device device_file 9 900 "$MYSELF”

rcdaq_client load librcdaqplugin_drs.so
rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 120 3
```

We comment a lot as a way of documentation

If no server is running, we start one here.

We convert the script filename into a full path

We clear all existing definitions

We load the plugin(s) and define the device(s)

Output file management

The output file names are governed by what is called a “file rule”

It acts as the format portion of a c-program “printf” statement

The startup default is “rcdaq-%08d-%04d.evt” – called with 2 int parameters, one the run number, and one a “file sequence number” if we roll over files

You can try this even on the command line (I added the \n to have it by itself on a line)

```
$ printf "rcdaq-%08d-%04d.evt\n" 100 0
rcdaq-00000100-0000.evt
```

The %08d and %04d parts make an 8- or 4-digit number with leading zeroes – we want to avoid spaces in filenames.

We usually give a file rule with a full and complete path, such as

```
/data/phnxrc/BHCal/cosmics/cosmics_ROC-%08d-%04d.evt
```

You can have “Run Types” that quickly switch between different pre-made file rules for beam/cosmics/calibration/pedestal/junk etc

Here is the actual setup script for our H2GCROC readout

In 2 parts, and still a bit abridged…

```
#! /bin/sh
MYSELF=$(readlink -f $0)
if ! rcdaq_client daq_status > /dev/null 2>&1 ; then

    echo "No rcdaq_server running, starting... log goes to $HOME/rcdaq.log"
    rcdaq_server > $HOME/rcdaq.log 2>&1 &
    sleep 2

    # see if we have an elog command (ok, this is a weak test but
    # at least it tells us that elog is installed.)
    ELOG=$(which elog 2>/dev/null)
    [ -n "$ELOG" ]  && rcdaq_client elog localhost 666 RCDAQLog

fi
rcdaq_client daq_setrunnumberfile $HOME/.rcdaq_run
```

Here we set up our elog, so RCDAQ can make automated entries

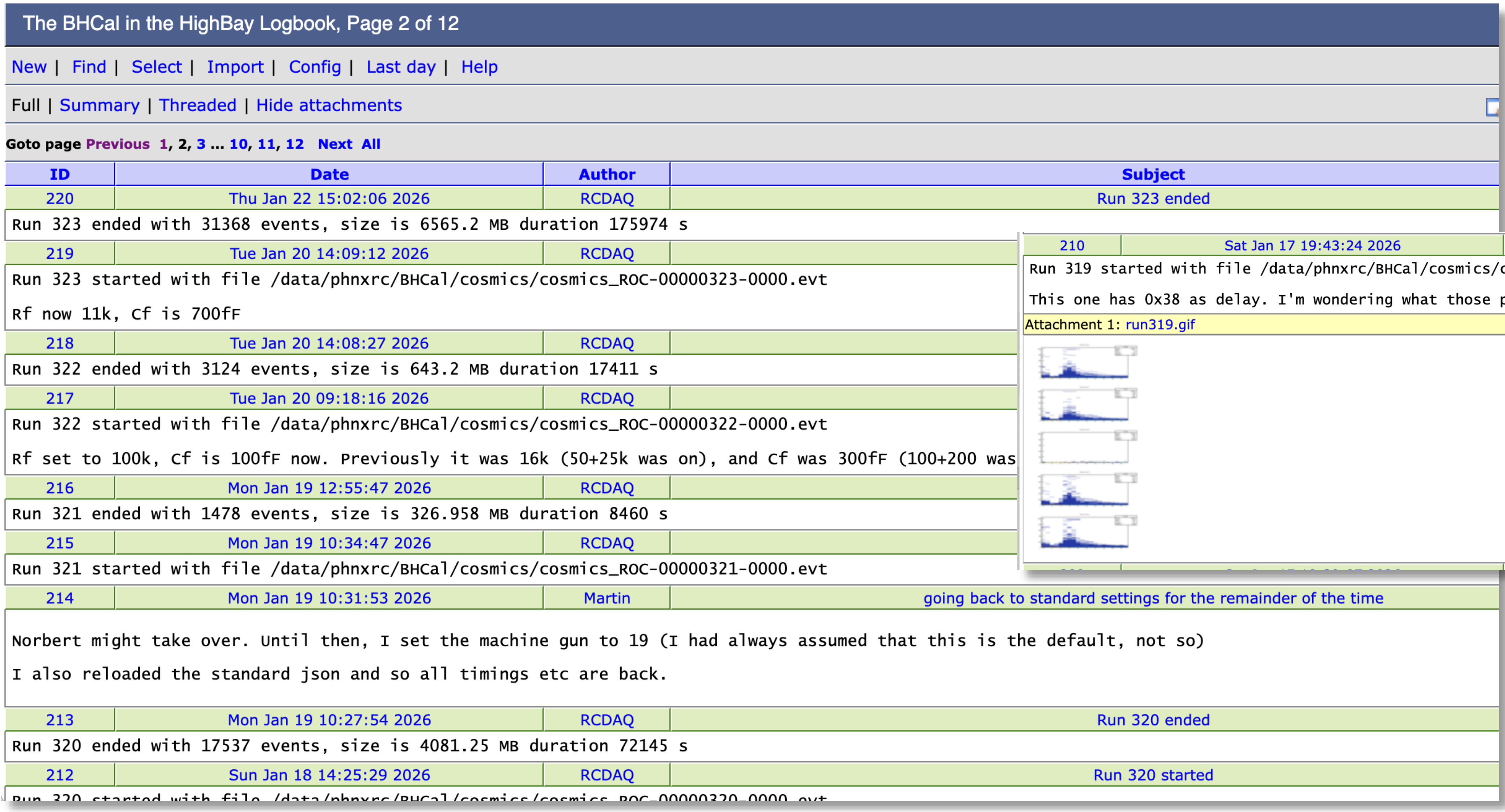
We make run numbers persistent across restarts

Automated Elog Entries

RCDAQ can make automated entries in your Elog

Of course you can make your own entries, document stuff, edit entries, add plots…

Gives a nice timeline and log



The BHCal in the HighBay Logbook, Page 2 of 12

New | Find | Select | Import | Config | Last day | Help

Full | Summary | Threaded | Hide attachments

Goto page Previous 1, 2, 3 ... 10, 11, 12 Next All

ID	Date	Author	Subject
220	Thu Jan 22 15:02:06 2026	RCDAQ	Run 323 ended

```
Run 323 ended with 31368 events, size is 6565.2 MB duration 175974 s
```

219	Tue Jan 20 14:09:12 2026	RCDAQ	

```
Run 323 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000323-0000.evt

Rf now 11k, Cf is 700fF
```

218	Tue Jan 20 14:08:27 2026	RCDAQ	

```
Run 322 ended with 3124 events, size is 643.2 MB duration 17411 s
```

217	Tue Jan 20 09:18:16 2026	RCDAQ	

```
Run 322 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000322-0000.evt

Rf set to 100k, Cf is 100fF now. Previously it was 16k (50+25k was on), and Cf was 300fF (100+200 was
```

216	Mon Jan 19 12:55:47 2026	RCDAQ	

```
Run 321 ended with 1478 events, size is 326.958 MB duration 8460 s
```

215	Mon Jan 19 10:34:47 2026	RCDAQ	

```
Run 321 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000321-0000.evt
```

214	Mon Jan 19 10:31:53 2026	Martin	going back to standard settings for the remainder of the time

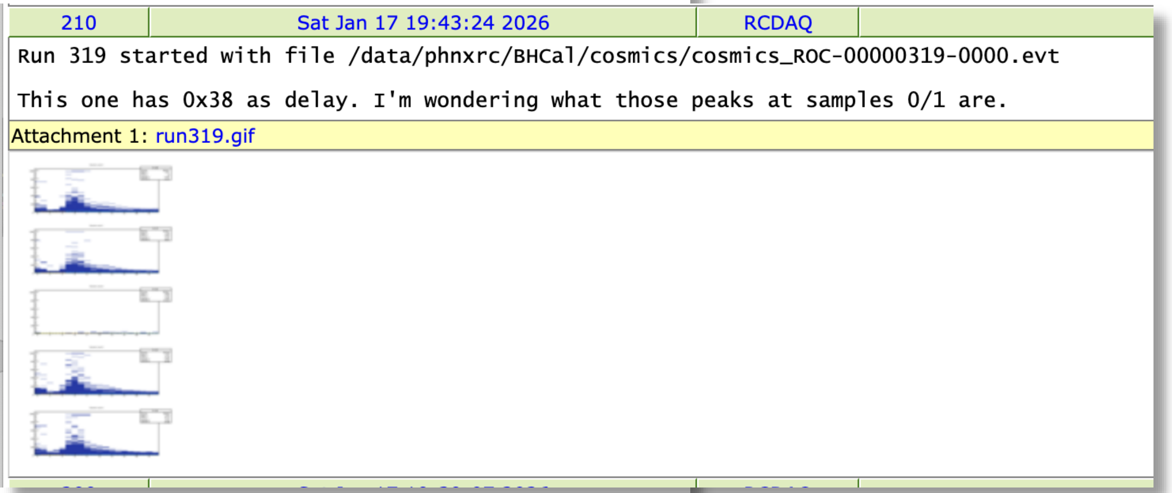
```
Norbert might take over. Until then, I set the machine gun to 19 (I had always assumed that this is the default, not so)

I also reloaded the standard json and so all timings etc are back.
```

213	Mon Jan 19 10:27:54 2026	RCDAQ	Run 320 ended

```
Run 320 ended with 17537 events, size is 4081.25 MB duration 72145 s
```

212	Sun Jan 18 14:25:29 2026	RCDAQ	Run 320 started

```
Run 320 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000320-0000.evt
```



210	Sat Jan 17 19:43:24 2026	RCDAQ	

```
Run 319 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000319-0000.evt

This one has 0x38 as delay. I'm wondering what those peaks at samples 0/1 are.
```

Attachment 1: run319.gif

Actual setup script for our H2GCROC readout, part 2

2nd part…

```
STEM=/data/phnxrc/BHCal
rcdaq_client daq_define_runtype junk      $STEM/junk/junk_ROC-%08d-%04d.evt
rcdaq_client daq_define_runtype mapping   $STEM/mapping/mapping_ROC-%08d-%04d.evt
rcdaq_client daq_define_runtype cosmics   $STEM/cosmics/cosmics_ROC-%08d-%04d.evt
rcdaq_client daq_define_runtype pedestal  $STEM/pedestal/pdestal_ROC-%08d-%04d.evt
rcdaq_client daq_set_runtype junk

rcdaq_client daq_clear_readlist

# we add this script itself to the begin-run event
rcdaq_client create_device device_file 9 900 "$MYSELF"

rcdaq_client load librcdaqplugin_h2gcroc3.so
rcdaq_client create_device device_h2gcroc3 1 12001 10.1.2.208 1 200

rcdaq_client daq_open
```

We pre-make 5 run types to switch quickly (file rules)

And this is the business end of this entire setup…

More about capturing your environment

Sometimes you add meta-info to make the analysis easier, especially in the aforementioned "scanning something" setups - example later

Many times you capture things only "just in case"

I usually add a camera picture to the begin-run, especially when the detector moves in the beam for some position scan

You don't routinely look at them in your analysis, but it's good to have that info

If you have some inexplicable feature, you can use the meta-data to do "forensics"

Find out what, if anything, went wrong

The more data you capture, the better this gets

Think of it as "black box" on a plane…

Forensics (FermiLab test beam for the future ePIC HCal)

“It appears that the distributions change for Cherenkov1 at 1,8,12,and 16 GeV compared to the other energies. It seems that the Cherenkov pressures are changed. […] Any help on understanding this would be appreciated.”

Martin: “Look at the info in the data files:”

```
$ ddump -t 9 -p 923 beam_00002298-0000.prdf
S:MTNRG   = -1     GeV
F:MT6SC1 =  5790       Cnt
F:MT6SC2 =  3533       Cnt
F:MT6SC3 =  1780       Cnt
F:MT6SC4 =  0          Cnt
F:MT6SC5 =  73316      Cnt
E:2CH     =  1058  mm
E:2CV     =  133.1 mm
E:2CMT6T =  73.84 F
E:2CMT6H =  32.86 %Hum
F:MT5CP2 =  .4589 Psia
F:MT6CP2 =  .6794 Psia
```

```
$ ddump -t 9 -p 923 beam_00002268-0000.prdf
S:MTNRG   = -2     GeV
F:MT6SC1 =  11846      Cnts
F:MT6SC2 =  7069       Cnts
F:MT6SC3 =  3883       Cnts
F:MT6SC4 =  0          Cnts
F:MT6SC5 =  283048     Cnts
E:2CH     =  1058  mm
E:2CV     =  133   mm
E:2CMT6T =  74.13 F
E:2CMT6H =  37.26 %Hum
F:MT5CP2 =  12.95 Psia
F:MT6CP2 =  14.03 Psia
```

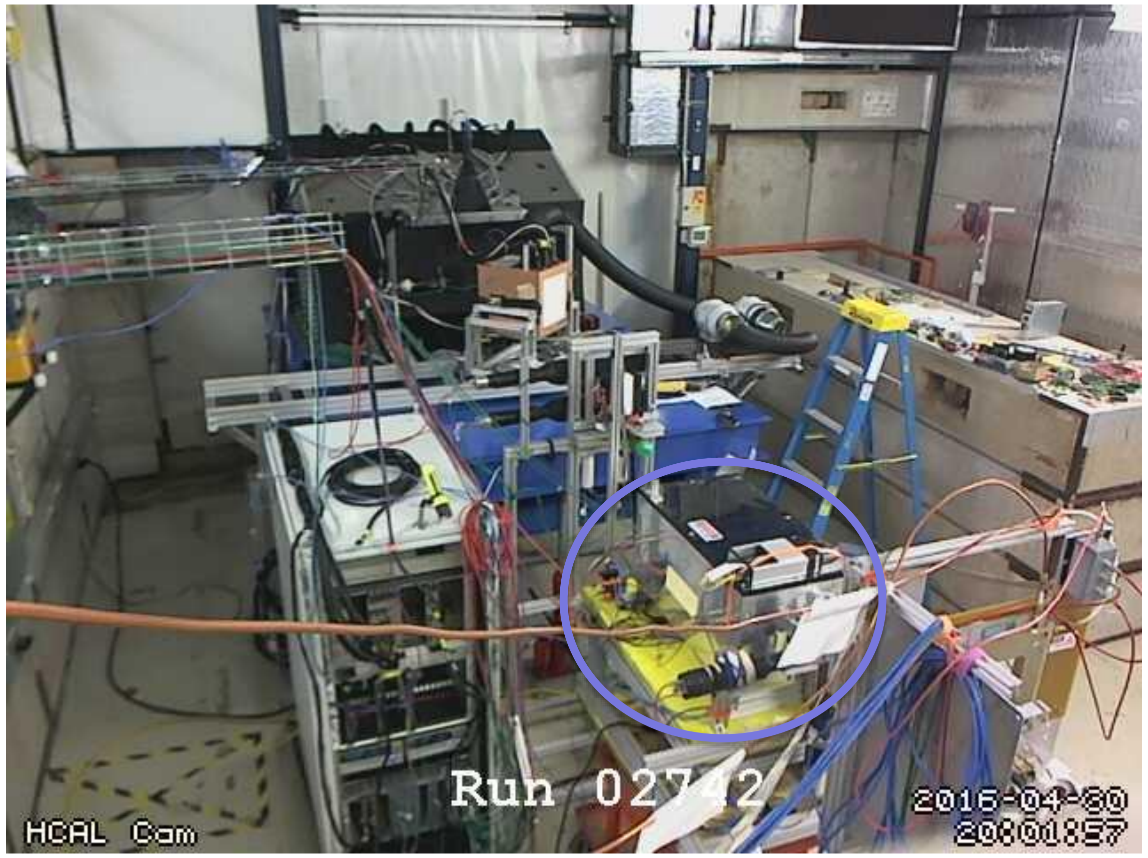
Among many other things, we capture the most relevant beamline parameters

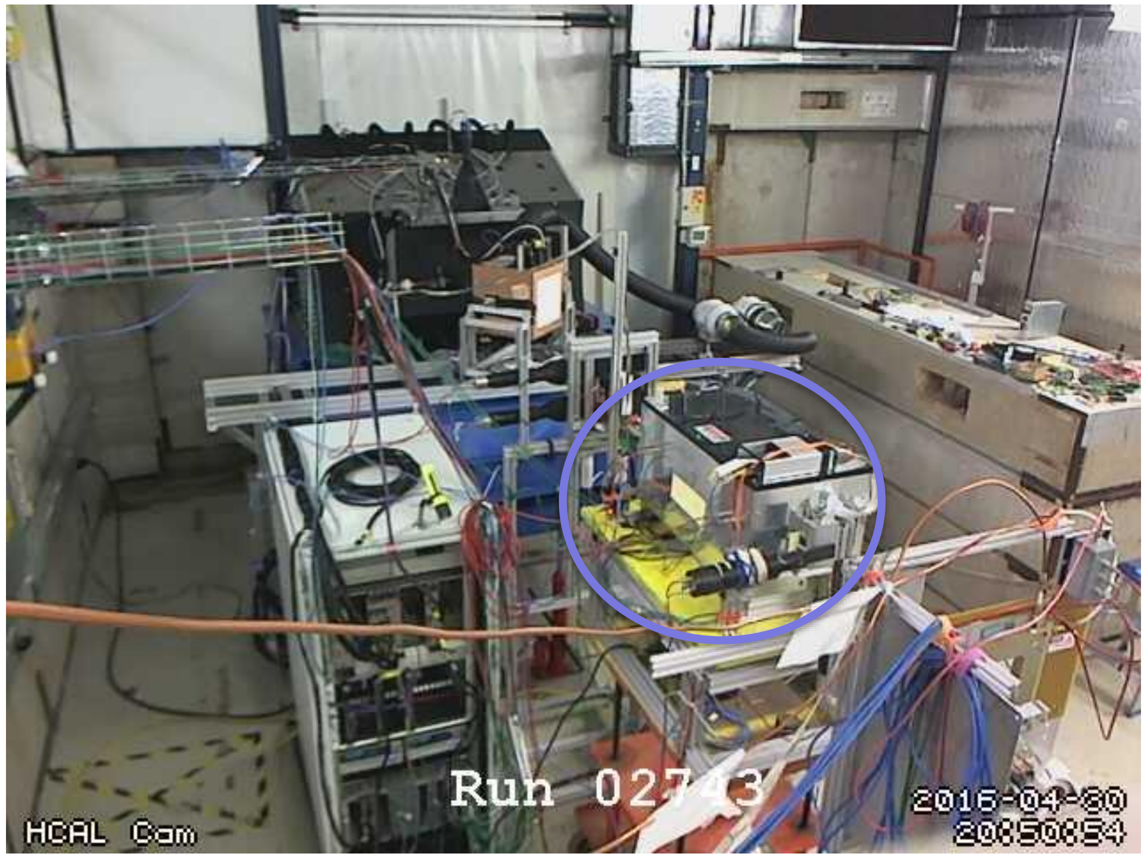
More Forensics (HCal at the Fermilab test beam again…)

“There is a strange effect starting in run 2743. There is a higher fraction of showering than before. I cannot see anything changed in the elog.”

Look at the cam pictures we automatically captured for each run:

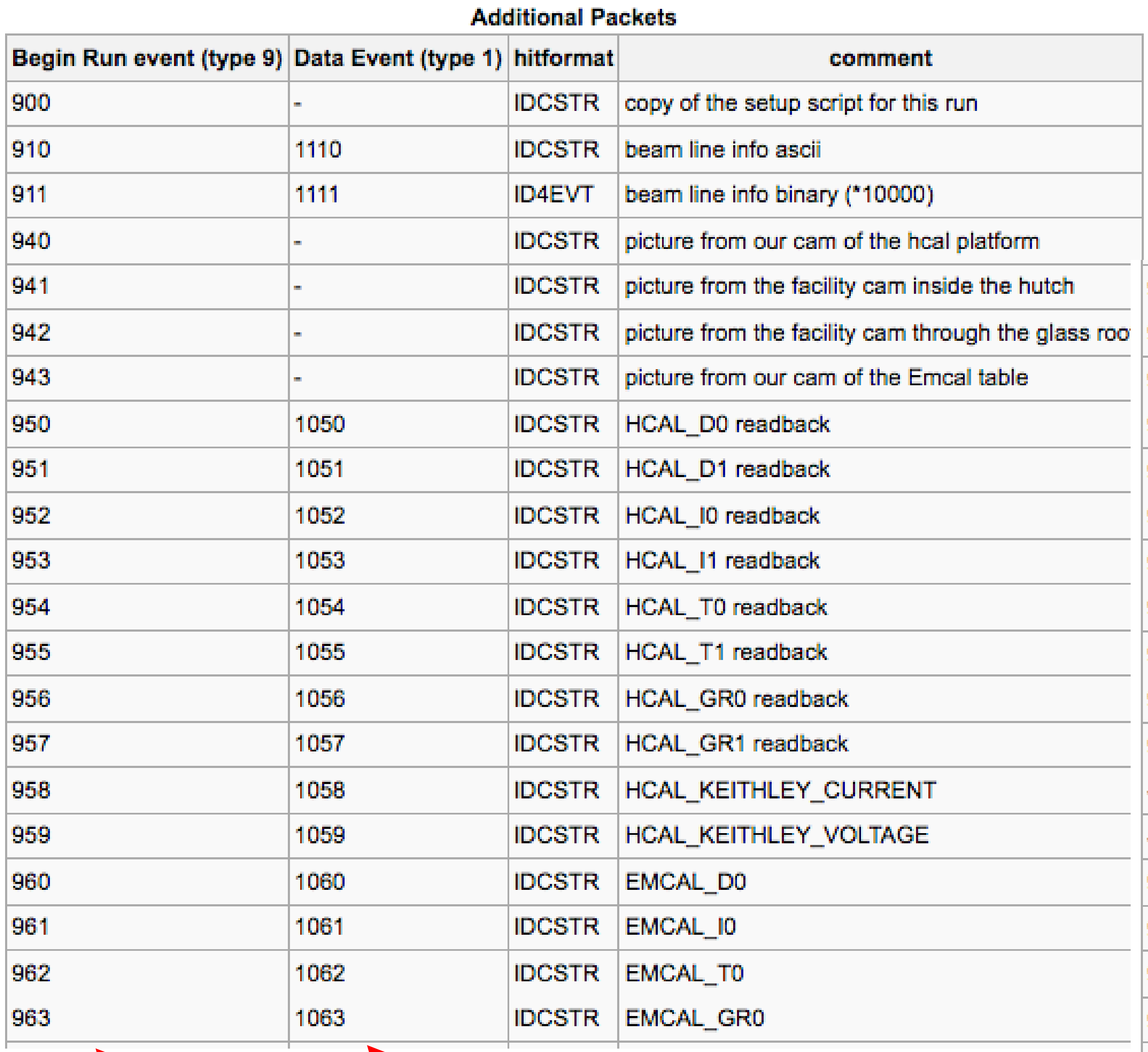
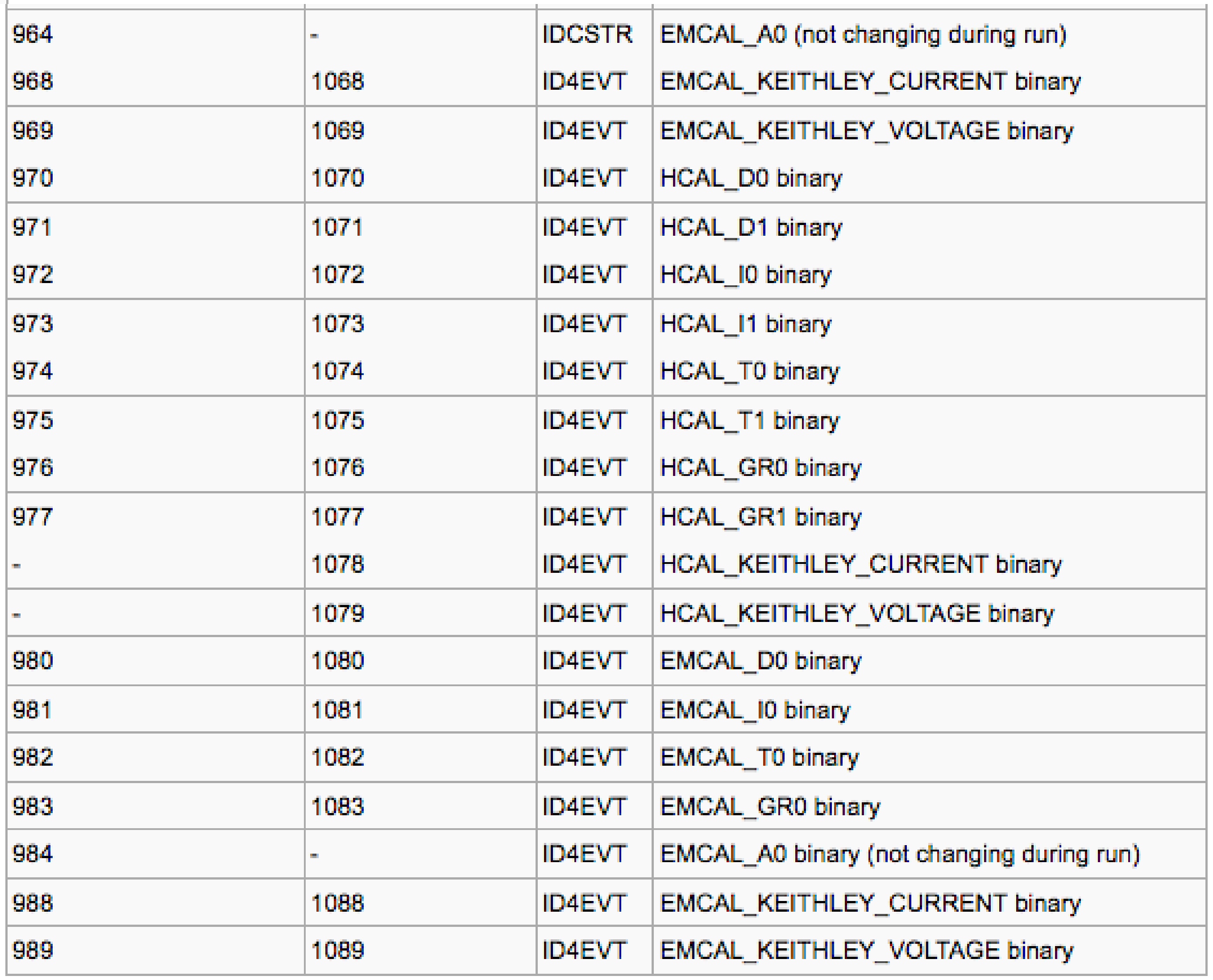
```
$ ddump -t 9 -p 940 beam_00002742-0000.prdf > 2742.jpg
$ ddump -t 9 -p 940 beam_00002743-0000.prdf > 2743.jpg
```





“Meta Data” Packet list from that test beam

More than 72 environment-capturing packets (accelerator params, voltages, currents, temperatures, pictures, …)

Additional Packets

Begin Run event (type 9)	Data Event (type 1)	hitformat	comment
900	-	IDCSTR	copy of the setup script for this run
910	1110	IDCSTR	beam line info ascii
911	1111	ID4EVT	beam line info binary (*10000)
940	-	IDCSTR	picture from our cam of the hcal platform
941	-	IDCSTR	picture from the facility cam inside the hutch
942	-	IDCSTR	picture from the facility cam through the glass roo
943	-	IDCSTR	picture from our cam of the Emcal table
950	1050	IDCSTR	HCAL_D0 readback
951	1051	IDCSTR	HCAL_D1 readback
952	1052	IDCSTR	HCAL_I0 readback
953	1053	IDCSTR	HCAL_I1 readback
954	1054	IDCSTR	HCAL_T0 readback
955	1055	IDCSTR	HCAL_T1 readback
956	1056	IDCSTR	HCAL_GR0 readback
957	1057	IDCSTR	HCAL_GR1 readback
958	1058	IDCSTR	HCAL_KEITHLEY_CURRENT
959	1059	IDCSTR	HCAL_KEITHLEY_VOLTAGE
960	1060	IDCSTR	EMCAL_D0
961	1061	IDCSTR	EMCAL_I0
962	1062	IDCSTR	EMCAL_T0
963	1063	IDCSTR	EMCAL_GR0
964	-	IDCSTR	EMCAL_A0 (not changing during run)
968	1068	ID4EVT	EMCAL_KEITHLEY_CURRENT binary
969	1069	ID4EVT	EMCAL_KEITHLEY_VOLTAGE binary
970	1070	ID4EVT	HCAL_D0 binary
971	1071	ID4EVT	HCAL_D1 binary
972	1072	ID4EVT	HCAL_I0 binary
973	1073	ID4EVT	HCAL_I1 binary
974	1074	ID4EVT	HCAL_T0 binary
975	1075	ID4EVT	HCAL_T1 binary
976	1076	ID4EVT	HCAL_GR0 binary
977	1077	ID4EVT	HCAL_GR1 binary
-	1078	ID4EVT	HCAL_KEITHLEY_CURRENT binary
-	1079	ID4EVT	HCAL_KEITHLEY_VOLTAGE binary
980	1080	ID4EVT	EMCAL_D0 binary
981	1081	ID4EVT	EMCAL_I0 binary
982	1082	ID4EVT	EMCAL_T0 binary
983	1083	ID4EVT	EMCAL_GR0 binary
984	-	ID4EVT	EMCAL_A0 binary (not changing during run)
988	1088	ID4EVT	EMCAL_KEITHLEY_CURRENT binary
989	1089	ID4EVT	EMCAL_KEITHLEY_VOLTAGE binary

Captured at begin-run

Captured again at spill-off

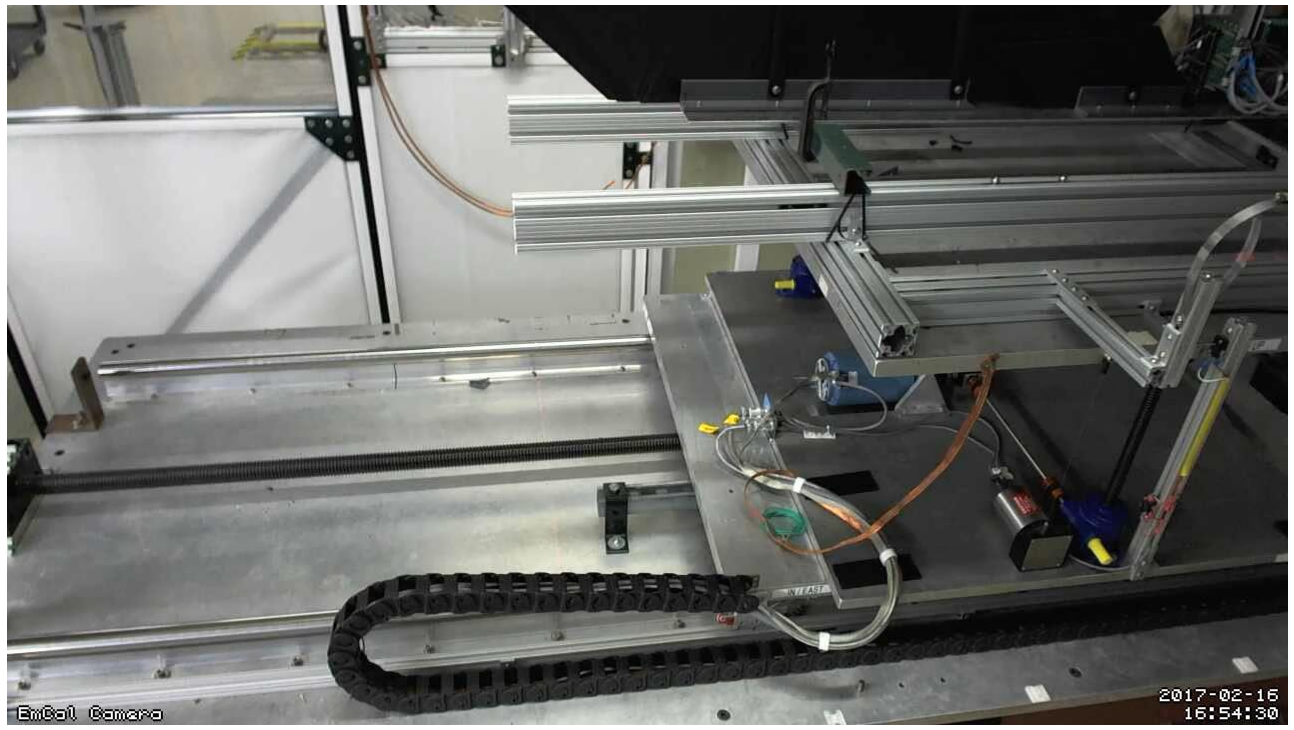
Moving Detector Example: “HCal Tile Mapping” at the Fermi Test Beam Facility

“Tile mapping” refers to mapping the position-dependent response of a hadronic calorimeter tile. We were moving a stack of Hcal tiles so the beam hits particular positions (In the picture, beam is coming at you)

About 200 individual positions of the tile relative to the beam – you’d go nuts doing all that manually, and you are bound to make mistakes

And all on camera for ach run for feel-good value – does that thing move right when we think it should?

This setup exercises many of the aforementioned features: scripting and reacting to the FTBF spill, network transparency (we cannot access the table positioning from our DAQ machine, but a FTBF-owned machine can control our DAQ)



Wait a minute – how to I get “fresh” begin-run information??

I have shown you how you swallow an existing file (like you setup file) into the begin-run event

But this was different – for each run we got a “fresh” picture from the cam… how?

Throw in the ”daq_device_command”

It doesn’t read out anything – just when the event is triggered, it executes the assorted command

```
# get the cameras
rcdaq_client create_device device_command 9 0 "$HOME/mlp/cam_capture.sh"
rcdaq_client create_device device_file_delete 9 940 "$HOME/mlp/hcals.jpg" 110
rcdaq_client create_device device_file_delete 9 941 "$HOME/mlp/ftbf_hut.jpg" 70
rcdaq_client create_device device_file_delete 9 942 "$HOME/mlp/ftbf_hut_birdseye.jpg" 70
rcdaq_client create_device device_file_delete 9 943 "$HOME/mlp/emcal.jpg" 110
rcdaq_client create_device device_file_delete 9 944 "$HOME/mlp/ftbf_moving_cam.jpg" 70
```

The subsequent commands grab the just-made jpg files *and delete them after*

So it is impossible to store stale info in case the picture-grabbing from the cam fails for some reason

Meta-data:I had called our H2GCROC setup script “abridged” – two more lines

```
# we add this script itself to the begin-run event
rcdaq_client create_device device_file 9 900 "$MYSELF”

rcdaq_client create_device device_command 9 0 "RequestScript.py > /tmp/roc_setup.json"
rcdaq_client create_device device_file_delete 9 910 /tmp/roc_setup.json

rcdaq_client load librcdaqplugin_h2gcroc3.so
rcdaq_client create_device device_h2gcroc3 1 12001 10.1.2.208 1 200

rcdaq_client daq_open
```

Analogous to what I showed you with the cameras before, the first command asks the KCU to dump every configuration setting into that /tmp/roc_setup.json file

The 2nd command absorbs the file into packet 910. So for each data file, you will be able to figure out how the setup was configured.

Config capture

```
$ ddump -t 9 -p 910  /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000299-0000.evt | more

['0xa0', '0x0', '0x10', '0x0', '0x0', '0x95', '0x5', '0xa0', '0x0', '0x0', '0x0', '0x0',
   '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
   '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
   '0x0', '0x0', '0x0', '0x0']

ASIC0 Register Top:  0xa0 0x00 0x10 0x00 0x00 0x95 0x05 0xb4 0x0b 0x0f 0x40 0x7f 0x00 0x07
   0x85 0x00 0xff 0x00 0xff 0x00 0xff 0x00 0x7f 0x00 0x22 0x02 0x04 0x00 0x00 0x00 0x00
   0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

['0xa0', '0x0', '0x10', '0x0', '0x0', '0x8f', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
   '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
   '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
   '0x0', '0x0', '0x0', '0x0’]

. . . And on and on…
```

Online monitoring in 3 slides

In an offline analysis, you run through a dataset. At the end, you write out or look at {histograms, ntuples, what have you}

No good for online monitoring!

You want to look at (and more generally manipulate, such as fit, reset, …) at any time ***while*** the data are coming in

What I see from other packages often is a root loop that refreshes some pre-made displays periodically. I call this “billboard monitoring”, and it has its applications.

However: I want total interactivity! Bottom line: I want my root prompt “active” at all times to be able to do stuff while the data are flowing!

And yes, I sometimes also do billboard monitoring where appropriate (like on a completely passive X display mounted on the wall).

But the interactivity is great when you are tuning things.

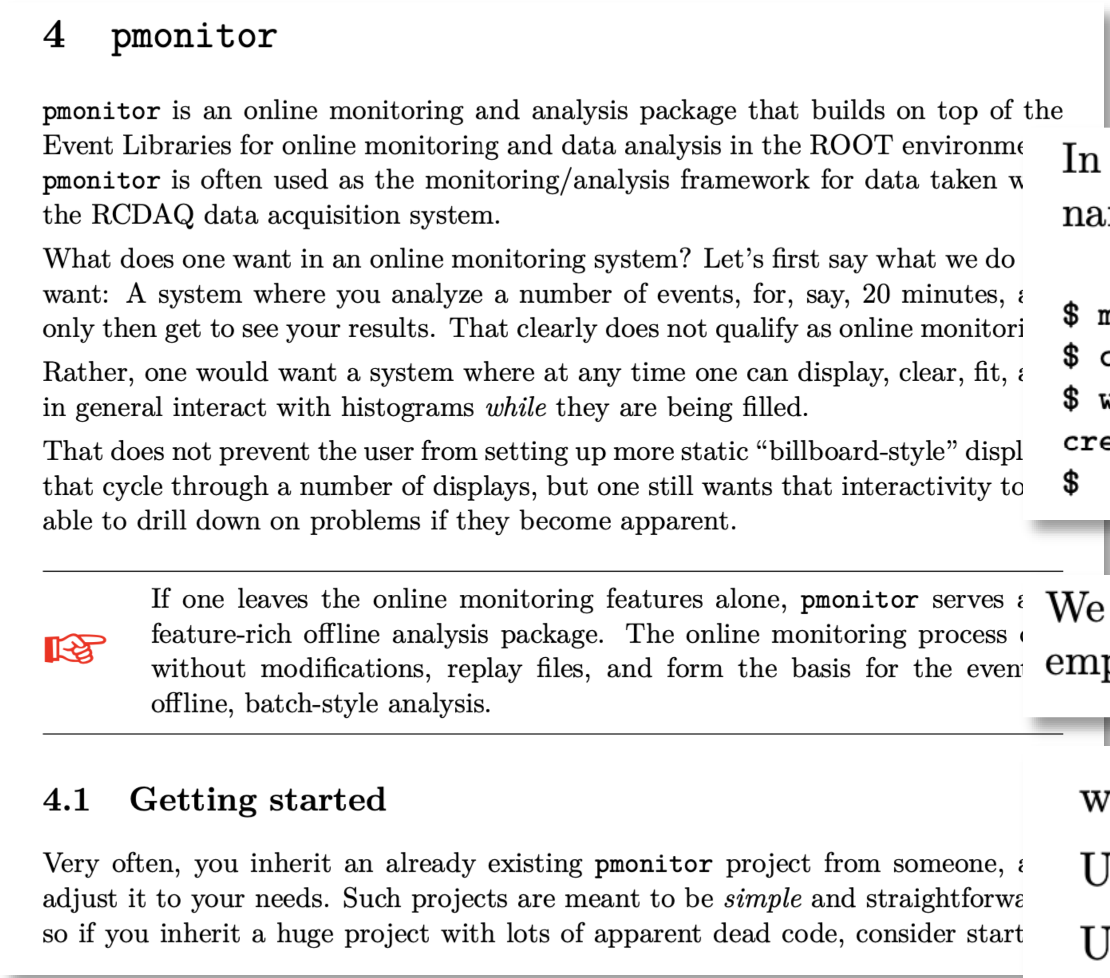
My package is called “pmonitor”. It works by pushing the processing to a background thread, so you always have the root prompt

You have the option not to use this feature – then you have a swiss-army-knife type offline analysis.

Let me show you the example from the manual:

In an offline analysis, you run through a dataset. At the end, you write out or look at {histograms, ntuples, what have you}

I have all theROC monitoring done, but this is how it started! Straight from the manual:



4 pmonitor

pmonitor is an online monitoring and analysis package that builds on top of the Event Libraries for online monitoring and data analysis in the ROOT environme pmonitor is often used as the monitoring/analysis framework for data taken w the RCDAQ data acquisition system.

What does one want in an online monitoring system? Let's first say what we do want: A system where you analyze a number of events, for, say, 20 minutes, a only then get to see your results. That clearly does not qualify as online monitori

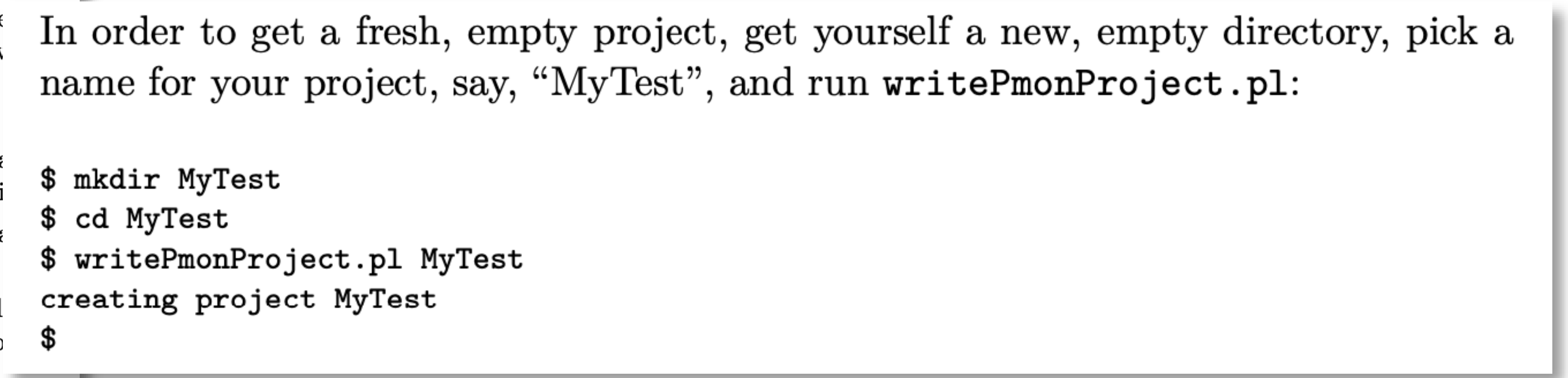
Rather, one would want a system where at any time one can display, clear, fit, a in general interact with histograms *while* they are being filled.

That does not prevent the user from setting up more static "billboard-style" displ that cycle through a number of displays, but one still wants that interactivity to able to drill down on problems if they become apparent.

☞ If one leaves the online monitoring features alone, pmonitor serves a feature-rich offline analysis package. The online monitoring process c without modifications, replay files, and form the basis for the even offline, batch-style analysis.

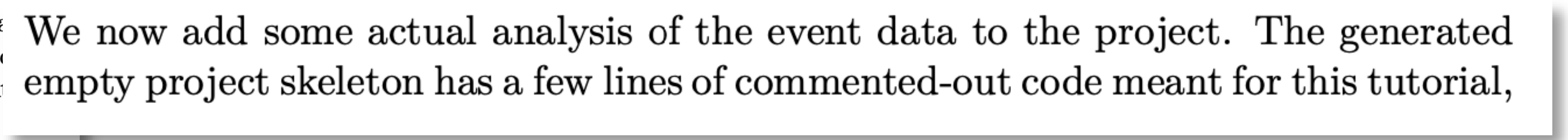
4.1 Getting started

Very often, you inherit an already existing pmonitor project from someone, a adjust it to your needs. Such projects are meant to be *simple* and straightforwa so if you inherit a huge project with lots of apparent dead code, consider start

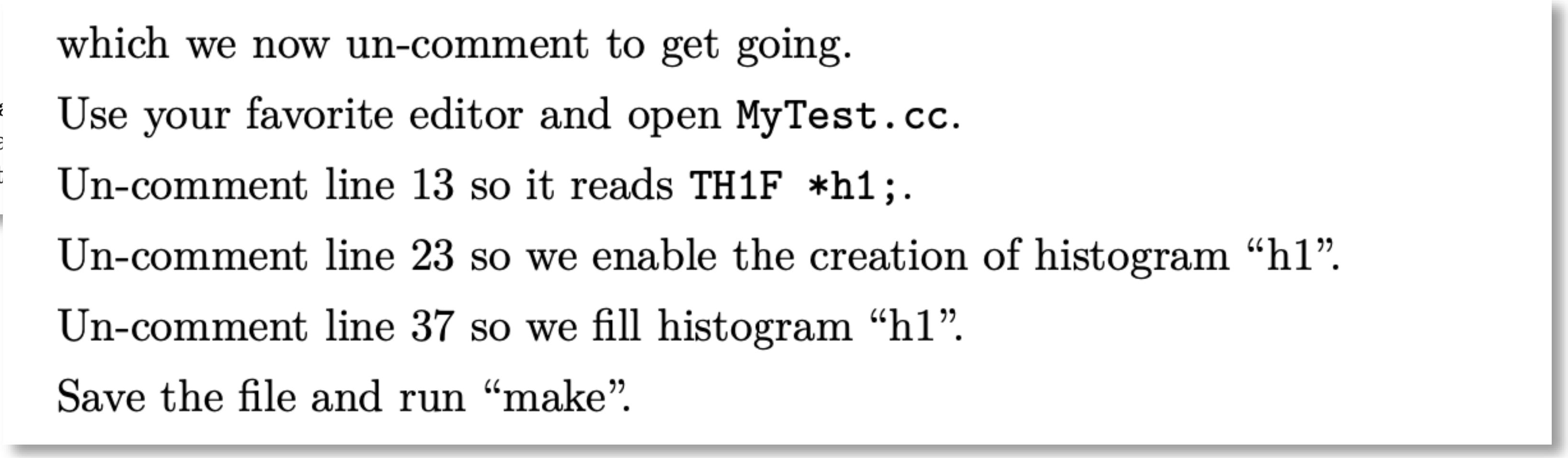


In order to get a fresh, empty project, get yourself a new, empty directory, pick a name for your project, say, "MyTest", and run writePmonProject.pl:

```
$ mkdir MyTest
$ cd MyTest
$ writePmonProject.pl MyTest
creating project MyTest
$
```



We now add some actual analysis of the event data to the project. The generated empty project skeleton has a few lines of commented-out code meant for this tutorial,



which we now un-comment to get going.

Use your favorite editor and open MyTest.cc.

Un-comment line 13 so it reads TH1F *h1;.

Un-comment line 23 so we enable the creation of histogram "h1".

Un-comment line 37 so we fill histogram "h1".

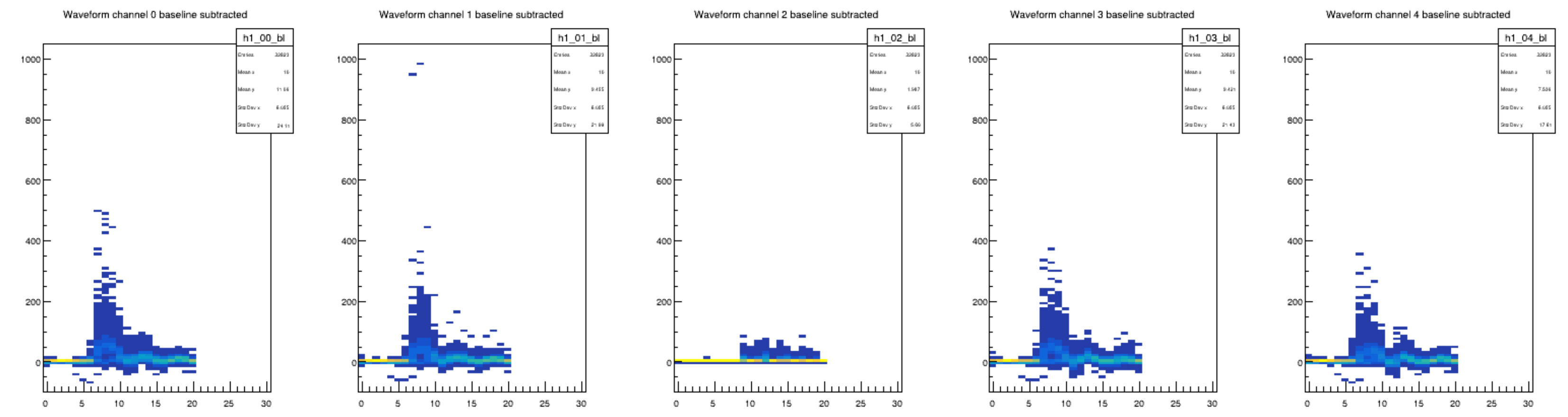
Save the file and run "make".

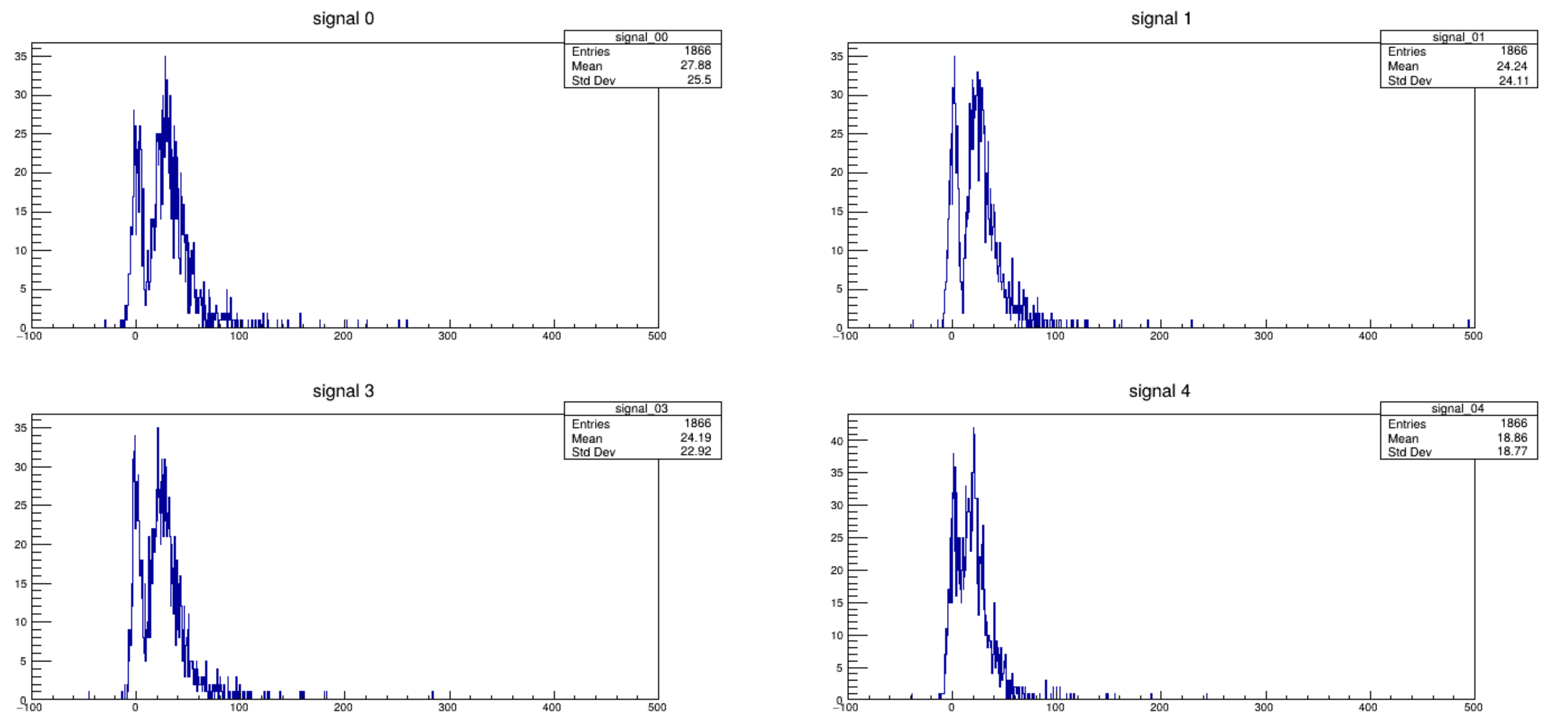
This is our online monitoring:

It makes many monitoring histograms, and has some configurable Canvas setups it can generate

What you see here are the persistency plots that plot many waveforms on top of each other (gleaned from a digital scope "persistency mode" where the same happens)

We see that one of our scintillators has a weak signal…





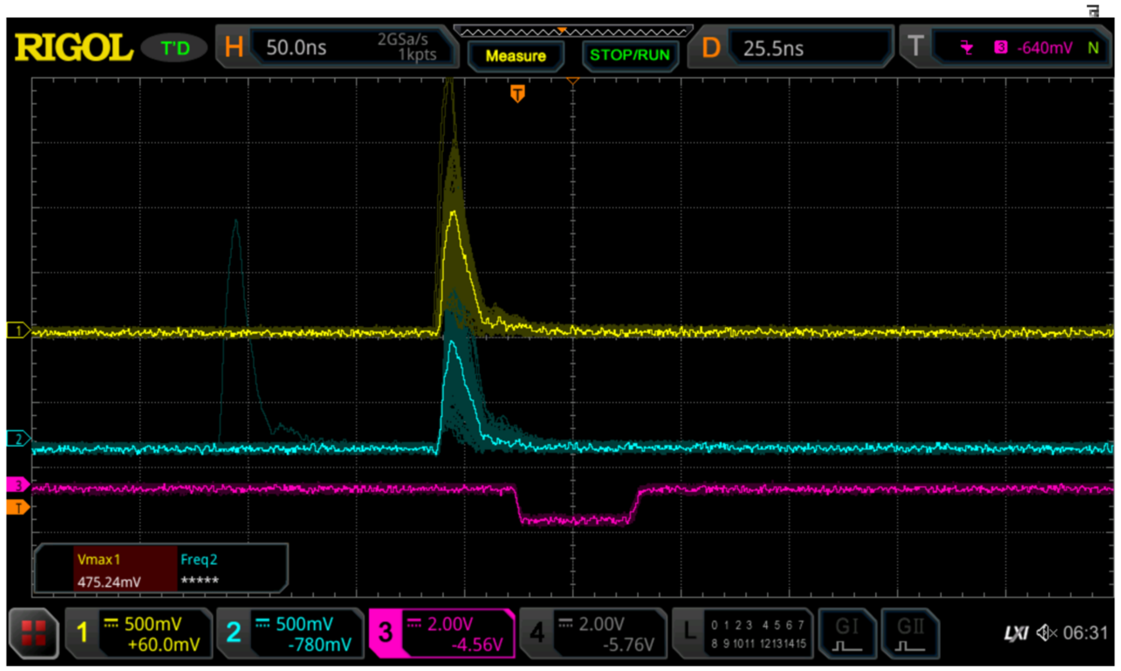
This is a crude waveform analysis to get a signal, histogrammed here

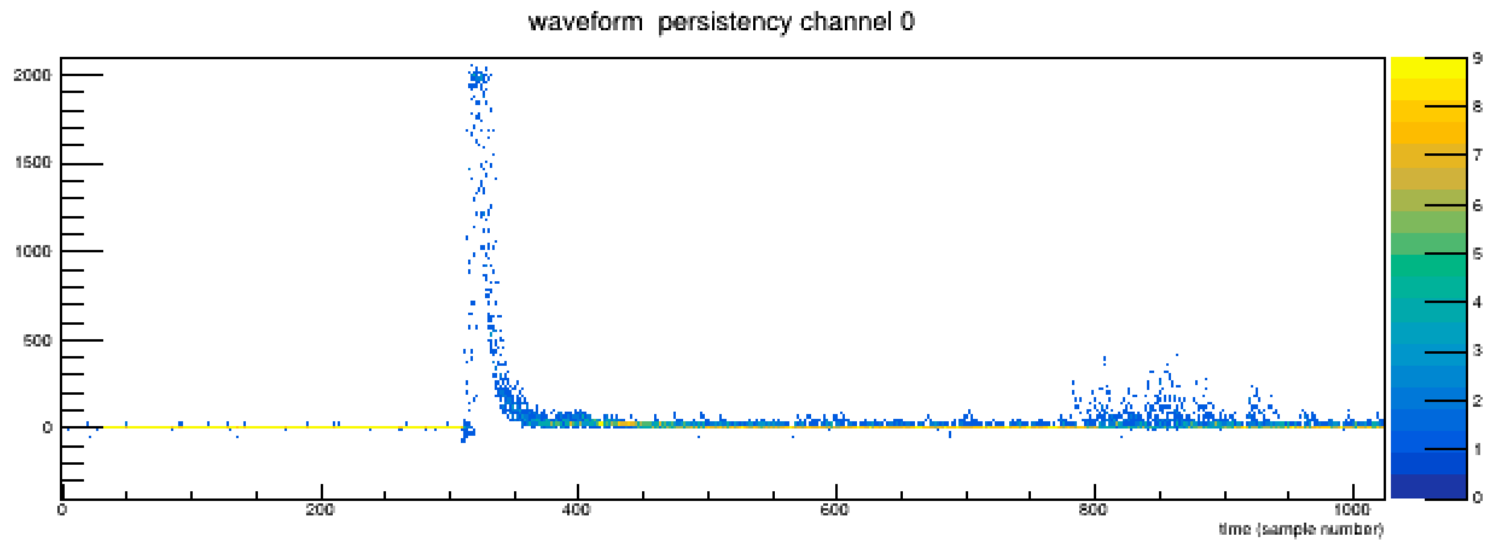
And for our eventual trigger setup:

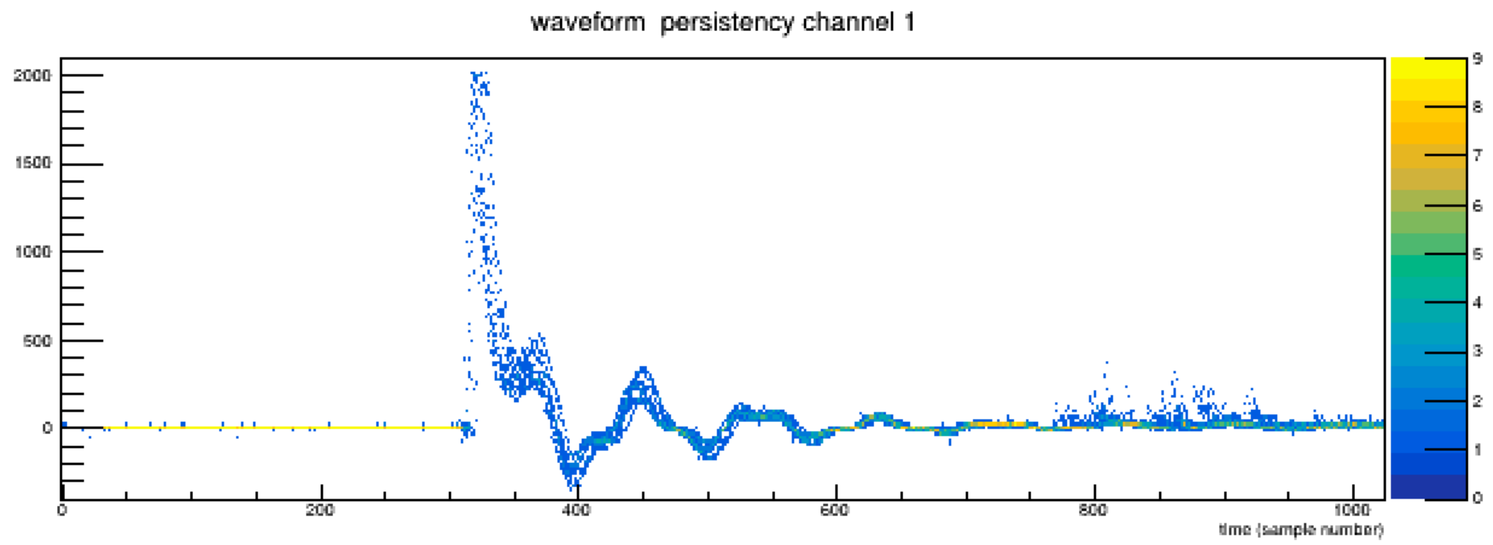
I added the CAEN DT5743 waveform sampler to the setup.

In order to have something to show, I added out trigger scintillators' last-dynode output to ch 0 and 1

Here shown on a scopeshot and from onl. monitoring





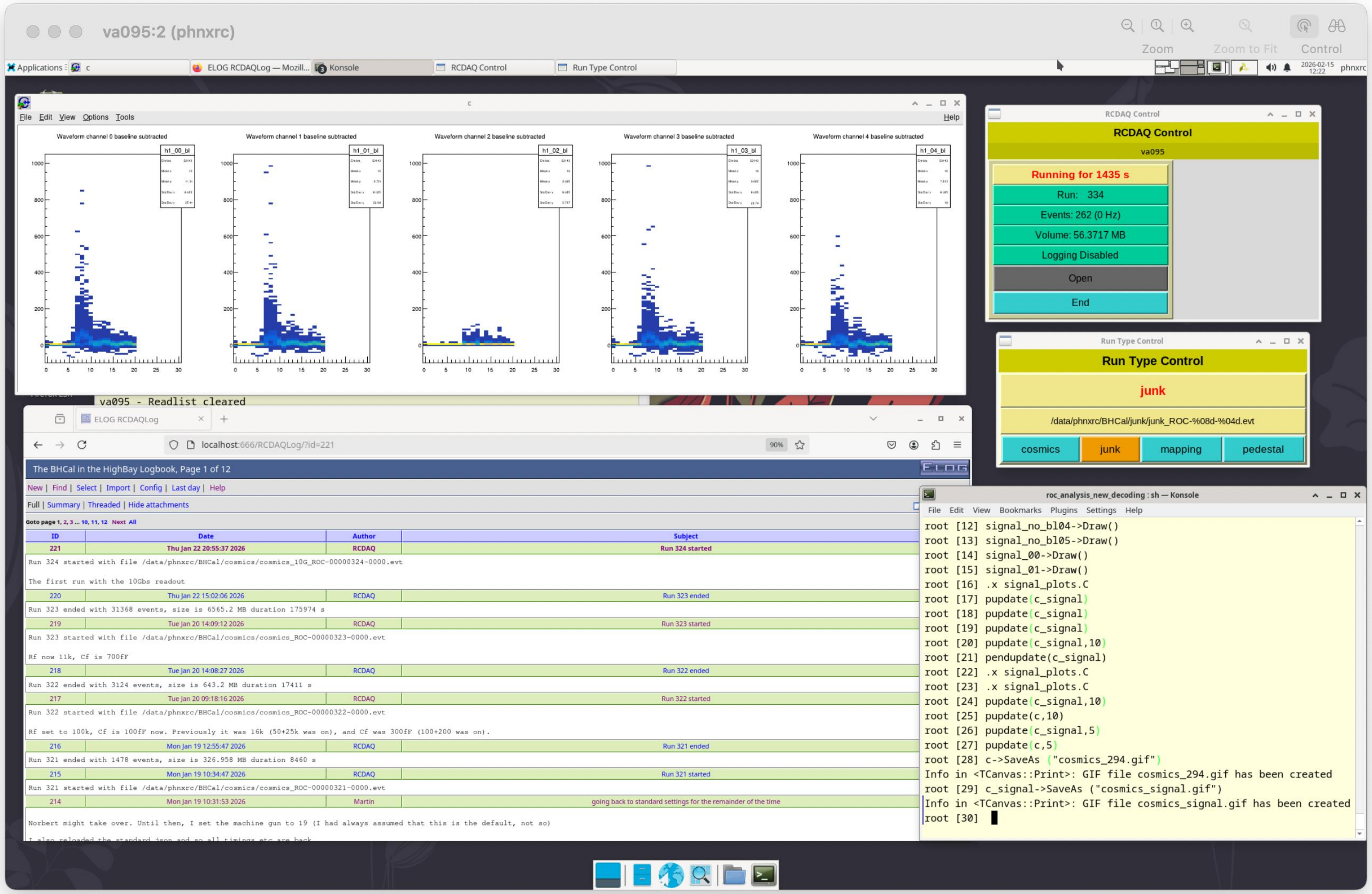


This is our VNC “Control Station”

We run our entire show in a VNC desktop (the machine itself is headless, what’s the point…)

You may have used NoMachine – VNC is similar but way better

It allows us to share the same desktop among different people. So experts can intervene in the same environment/Desktop (”let me quickly show you…”)



```
va095 - Readlist cleared
```

ELOG RCDAQLog — localhost:666/RCDAQLog/?id=221

The BHCal in the HighBay Logbook, Page 1 of 12

New | Find | Select | Import | Config | Last day | Help

Full | Summary | Threaded | Hide attachments

Goto page 1, 2, 3 ... 10, 11, 12 Next All

ID	Date	Author	Subject
221	Thu Jan 22 20:55:37 2026	RCDAQ	Run 324 started
Run 324 started with file /data/phnxrc/BHCal/cosmics/cosmics_10G_ROC-00000324-0000.evt The first run with the 10Gbs readout			
220	Thu Jan 22 15:02:06 2026	RCDAQ	Run 323 ended
Run 323 ended with 31368 events, size is 6565.2 MB duration 175974 s			
219	Tue Jan 20 14:09:12 2026	RCDAQ	Run 323 started
Run 323 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000323-0000.evt Rf now 11k, Cf is 700fF			
218	Tue Jan 20 14:08:27 2026	RCDAQ	Run 322 ended
Run 322 ended with 3124 events, size is 643.2 MB duration 17411 s			
217	Tue Jan 20 09:18:16 2026	RCDAQ	Run 322 started
Run 322 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000322-0000.evt Rf set to 100k, Cf is 100fF now. Previously it was 16k (50+25k was on), and Cf was 300fF (100+200 was on).			
216	Mon Jan 19 12:55:47 2026	RCDAQ	Run 321 ended
Run 321 ended with 1478 events, size is 326.958 MB duration 8460 s			
215	Mon Jan 19 10:34:47 2026	RCDAQ	Run 321 started
Run 321 started with file /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000321-0000.evt			
214	Mon Jan 19 10:31:53 2026	Martin	going back to standard settings for the remainder of the time
Norbert might take over. Until then, I set the machine gun to 19 (I had always assumed that this is the default, not so)			

roc_analysis_new_decoding : sh — Konsole

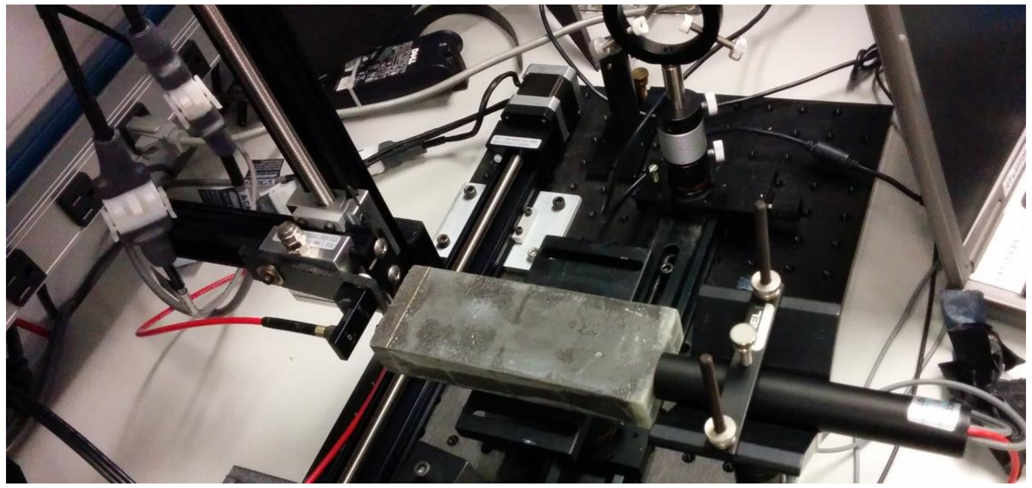
```
root [12] signal_no_bl04->Draw()
root [13] signal_no_bl05->Draw()
root [14] signal_00->Draw()
root [15] signal_01->Draw()
root [16] .x signal_plots.C
root [17] pupdate(c_signal)
root [18] pupdate(c_signal)
root [19] pupdate(c_signal)
root [20] pupdate(c_signal,10)
root [21] pendupdate(c_signal)
root [22] .x signal_plots.C
root [23] .x signal_plots.C
root [24] pupdate(c_signal,10)
root [25] pupdate(c,10)
root [26] pupdate(c_signal,5)
root [27] pupdate(c,5)
root [28] c->SaveAs ("cosmics_294.gif")
Info in <TCanvas::Print>: GIF file cosmics_294.gif has been created
root [29] c_signal->SaveAs ("cosmics_signal.gif")
Info in <TCanvas::Print>: GIF file cosmics_signal.gif has been created
root [30]
```

Using RCDAC features to streamline your analysis

Going back to the “calo module mapping” for a moment

You are ending up with several thousand individual data files, one per position

In each begin-run event we capture additional information, especially the x/y position where this data point is from



RCDAQ has a “command device” that doesn’t read out anything but executes a command when the event is triggered – here we execute “getmotorpositions.sh” at each begin-run

This script reaches out to the positioning system and writes a file “positions.txt” with the two numbers

We then absorb the file into the begin-run event

```
# we add a command to capture the positions to a file
rcdaq_client create_device device_command 9 0 "$HOME/getmotorpositions.sh"
rcdaq_client create_device device_file 9 920 "$HOME/positions.txt"
rcdaq_client create_device device_filenumbers_delete 9 921 "$HOME/positions.txt"
```

And we can retrieve this in the analysis (here shown with a command-line utility):

```
 $ ddump -t 9 -p 920 scan10_0000201800-0000.evt
4600
-4400
```

Using RCDAC features to streamline your analysis

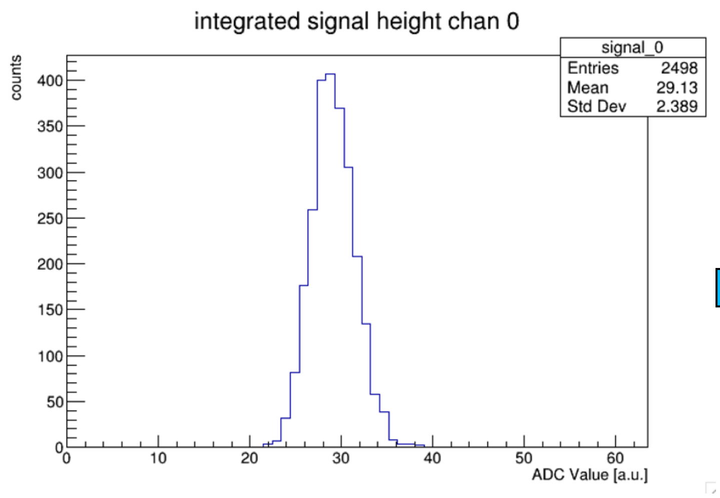
Then we make a list of all files that belong to this scan, and throw them all at the at the analysis process

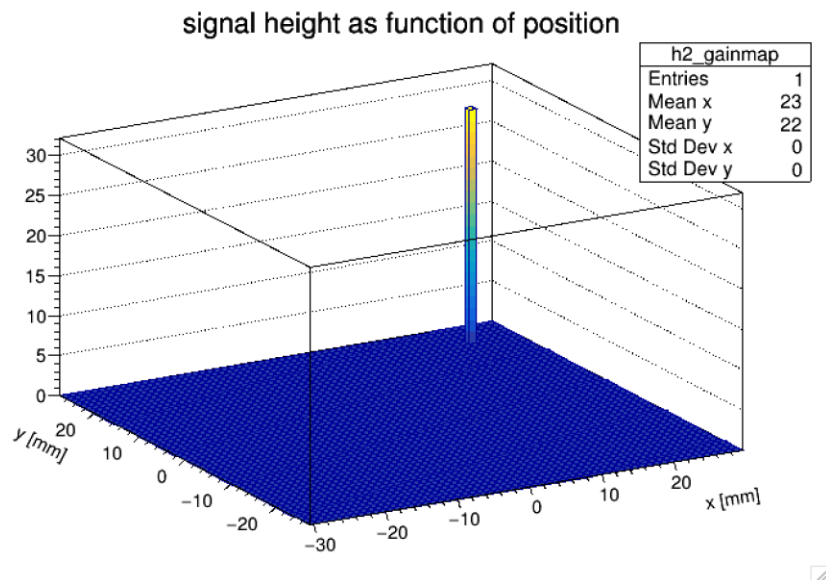
We get the begin-run event and extract and remember the x-y positions

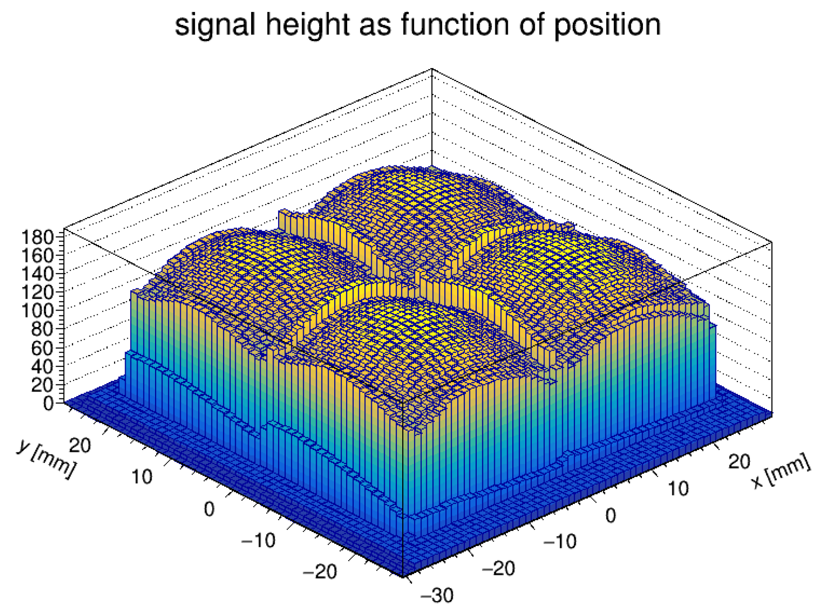
We then histogram the signal with the actual data events

Eventually we hit end end-run event, know that this position is done. Get the mean from the histogram, and fill it in the map at the right position

Run through all files in the set, have a coffee, and see the results:







In this way you can run the analysis without burdensome additional bookkeeping

All you need to analyze the data in one fell swoop is contained in the data themselves

Some code snippets

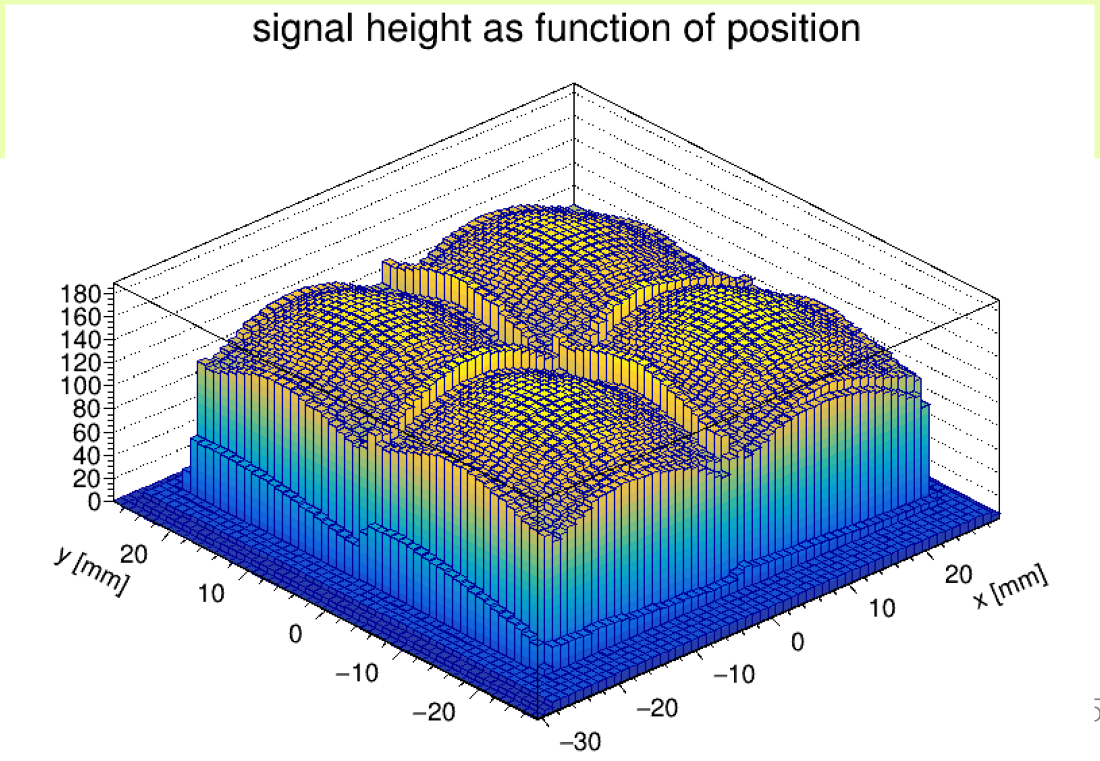
Here I’m extracting the x and y position from packet 921:

```
if ( e->getEvtType() == BEGINRUN )
    {
      reset_histograms();
      old_runnr = e->getRunNumber();
      Packet *p921 = e->getPacket(921);
      if ( p921)
        {
          xpos = p921->iValue(0);
          ypos = p921->iValue(1);
          cout << "Postions " << xpos << "   " << ypos << endl;
          delete p921;
        }
      return 0;
    }
```

Some code snippets

Here I recognize the end-run event and analyze the signal distribution for this run:

```
if ( e->getEvtType() == 12 )    // end run
   {

     h2_gainmap->Fill( (xpos+400)/200., (ypos-10000)/-200., h_signal->GetMean() );

     if (open_flag)
       {
         xlsfile << xpos  << " " << ypos << "   " << h_signal->GetMean() << endl;
       }

     return 0;
   }
```



Internal data format changes

We have transitioned from the old KCU firmware to the new 10G-capable one, which brought some internal format changes

Q: Do I have to re-write/modify all my analysis/monitoring/user code now?

A: Nope. The user code never accesses the raw data directly, but uses APIs to get at the information it needs

The APIs deliver the data in the same way, independent of internal format

This is done with the (historically named) “hitformat”

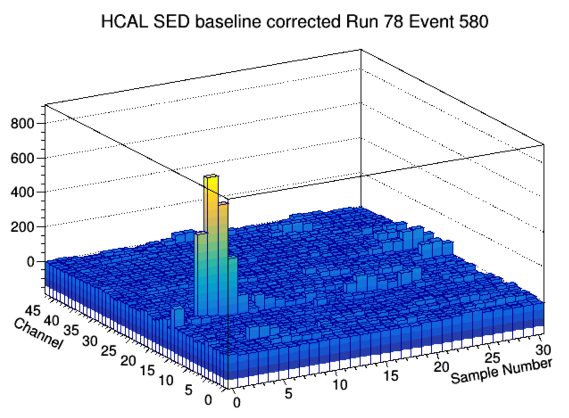
```
$ dlist cosmics_ROC-00000295-0000.evt
Packet 12001 73004 -1 (ePIC Packet) 301 (IDH2GCROC3)

$ dlist cosmics_10G_ROC-00000324-0000.evt
Packet 12001 31422 -1 (ePIC Packet) 302 (IDH2GCROC3_10G)
```

Some sPHENIX readout units (like the calorimeters) have seen 5,6,7 format changes, mostly to improve the “density” of the data storage

Like instead of first storing 2x10bits in a 32bit field, now store 3x10bits in same

No changes to any of the user/monitoring/reconstruction code!

A word about APIs



The packets provide their info via a number of APIs (about 10 different ones)

You need to know *what kind* of questions to ask

you need to know, in this example, that you are dealing with a waveform sampler (as opposed to, say, a pixel detector)

The APIs then hand you theit information in a generic way.

For example, **all non-streaming waveform samplers** (DRS4, CAEN 1742, the sPHENIX calorimeters, etc etc) can use the following code that asks

- Hey packet, how many channels do you have?
- Hey channel, how many samples do you have?
- Ok then, please give me sample s of channel c

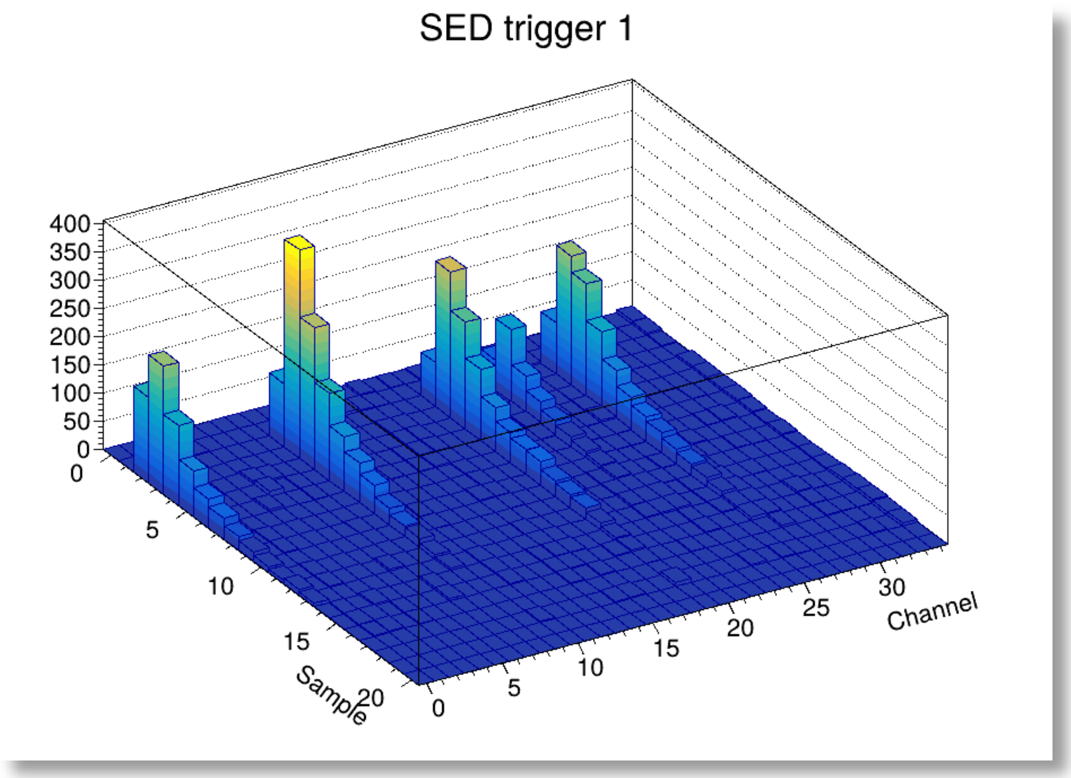
```
for ( int c = 0; c <  p->iValue(0,"CHANNELS"); c++) // the channels
 {
   for ( int s = 0;  s < p->iValue(c,"SAMPLES"); s++) // the samples
     {
       h2->Fill(s, c, p->iValue(s,c) );
     }
 }
```

Our Streamers have one more dimension

A SRO packet usually contains more than one waveform in a “TimeFrame”

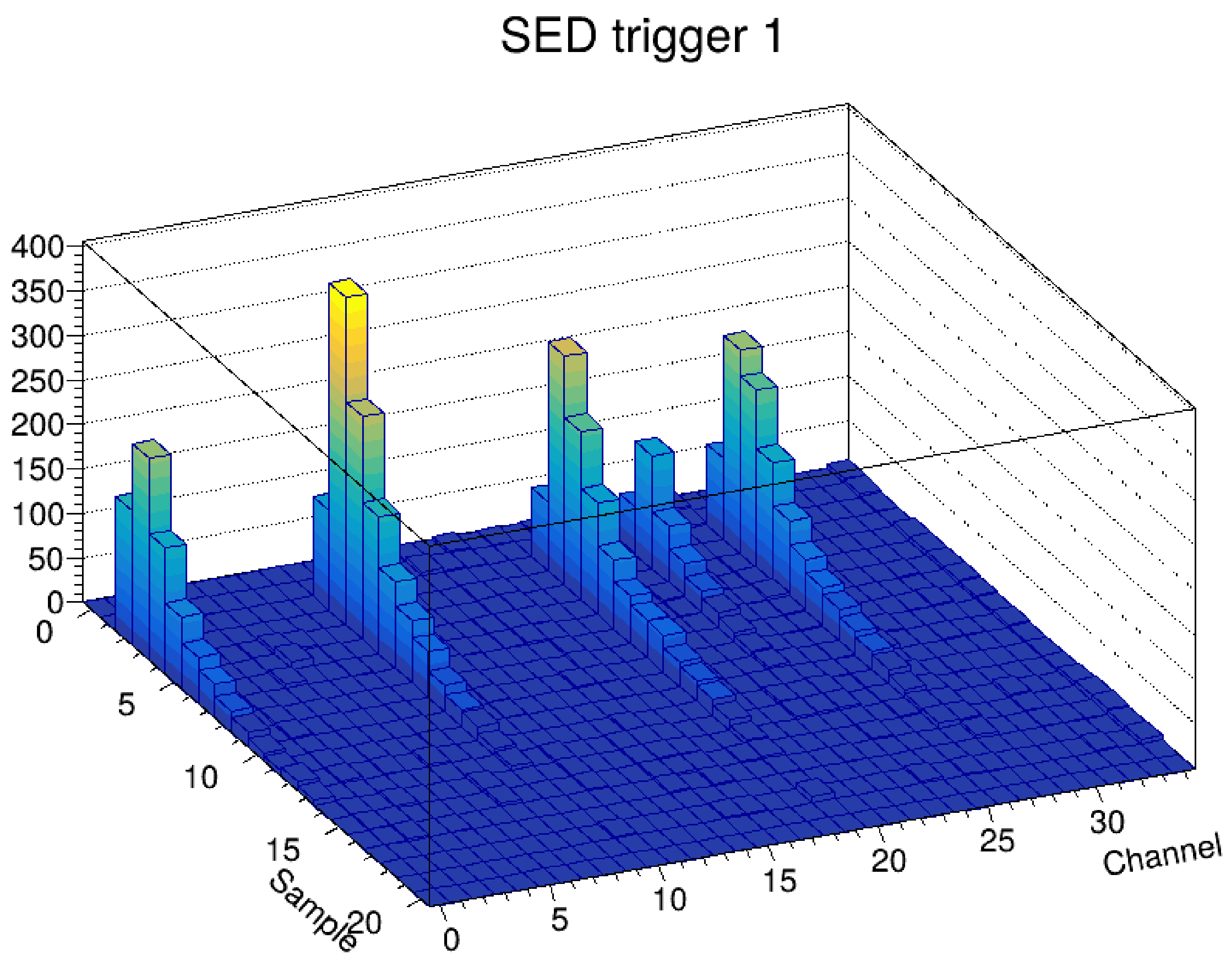
Here you ask:

- Hey packet, how many waveforms do you have?
- Hey packet, how many channels do you have for waveform n?
- Hey channel, how many samples do you have for waveform n ?
- Ok then, please give me sample s of channel c of waveform n

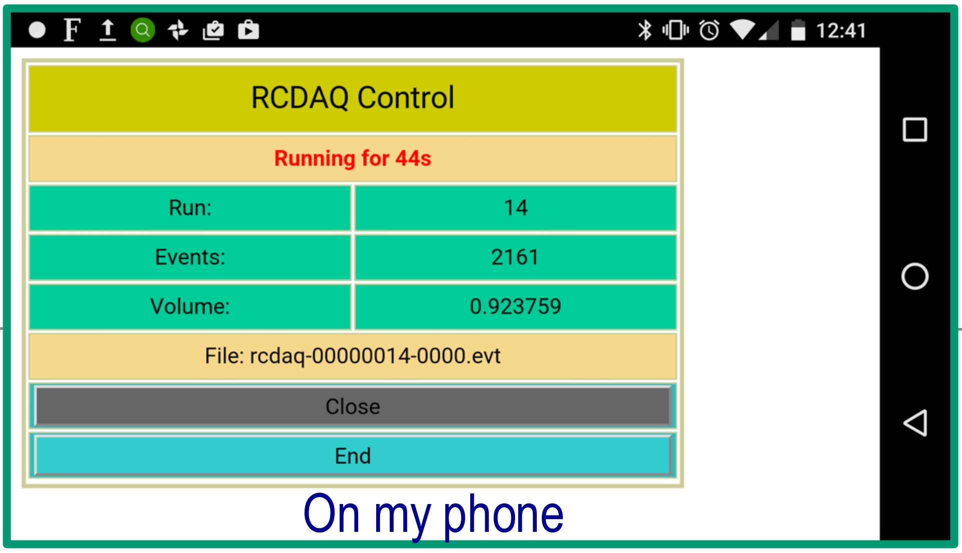


This is the end…

Let me end this with a bit of eye-candy

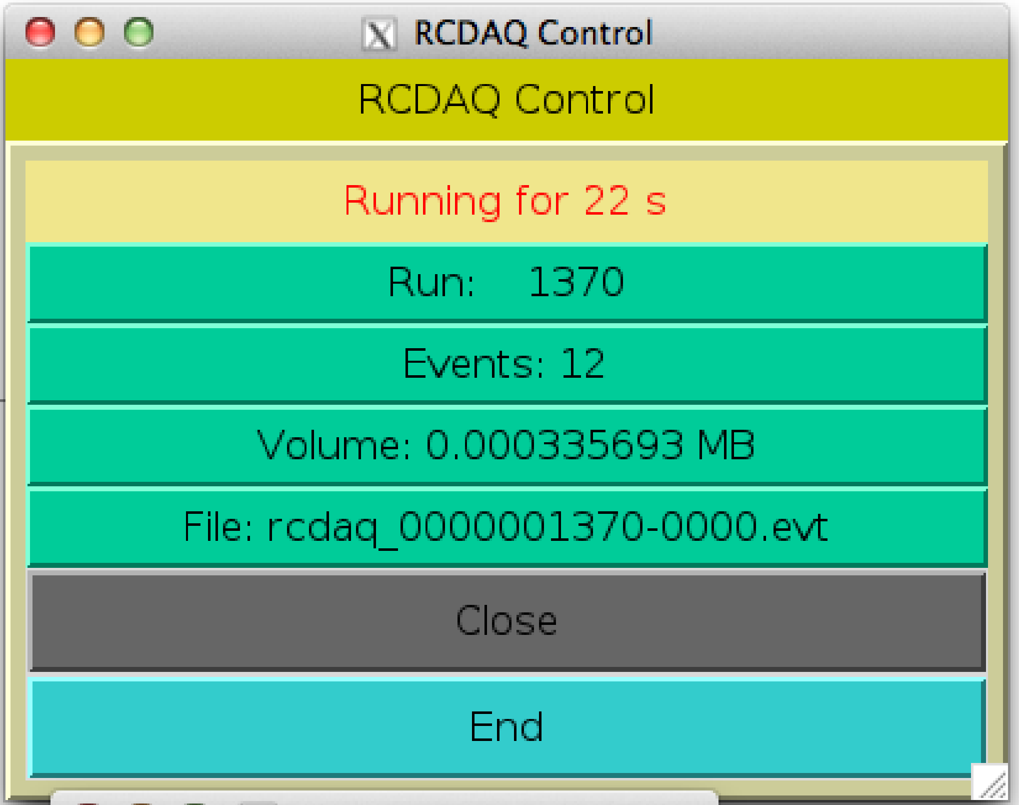


GUIs



RCDAQ Control	
Running for 44s	
Run:	14
Events:	2161
Volume:	0.923759
File: rcdaq-00000014-0000.evt	
Close	
End	

On my phone



RCDAQ Control

RCDAQ Control

Running for 22 s

Run: 1370

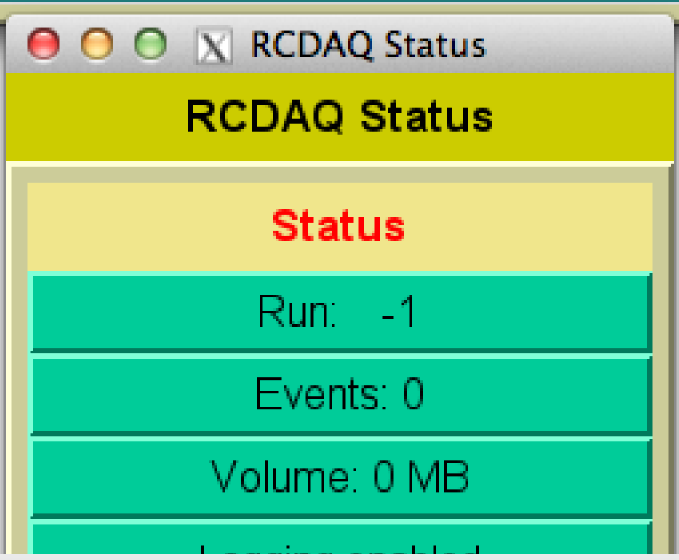
Events: 12

Volume: 0.000335693 MB

File: rcdaq_0000001370-0000.evt

Close

End



RCDAQ Status

RCDAQ Status

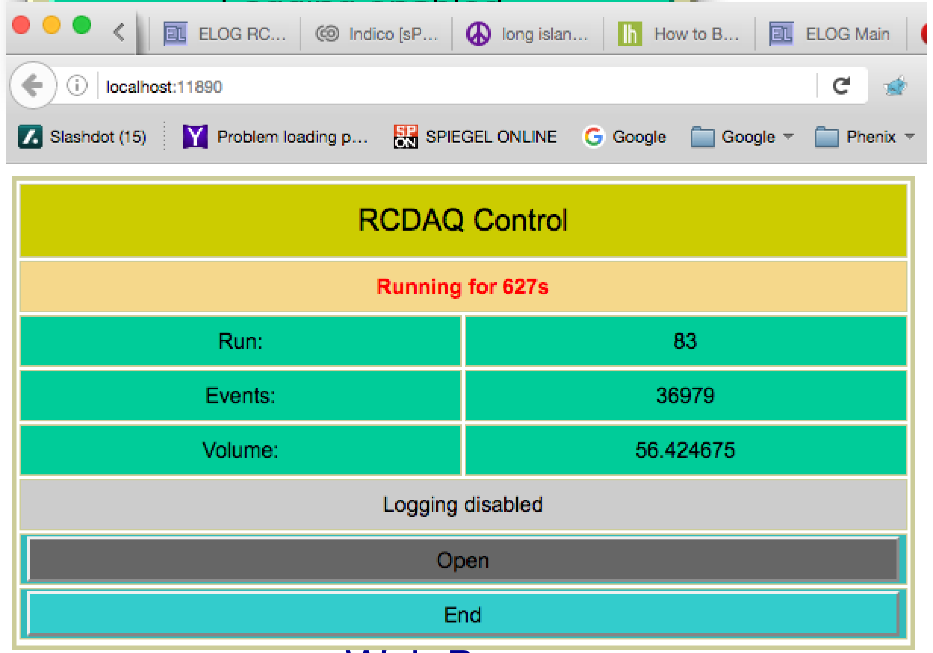
Status

Run: -1

Events: 0

Volume: 0 MB

Perl-TK



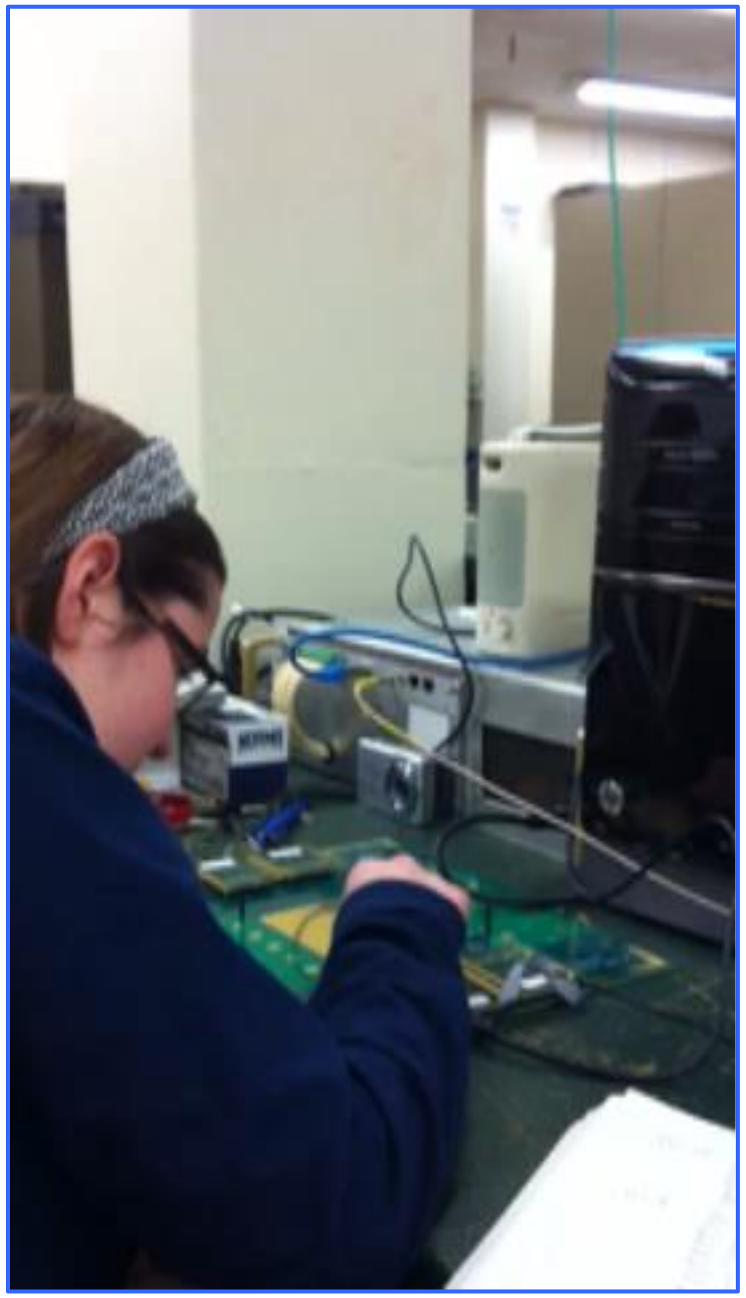
localhost:11890

RCDAQ Control	
Running for 627s	
Run:	83
Events:	36979
Volume:	56.424675
Logging disabled	
Open	
End	

Web Browser

- **GUIs must not be stateful!**
- Statelessness allows to have multiple GUIs at the same time
- And allows to mix GUIs with commands (think scripts)
- (all state information is kept in the rcdaq server)
- My GUI approach is to have perl-TK issue standard commands, parse the output
- Slowly transitioning to Web-based controls (web sockets + Javascript)

Shell integration



THE SPEAKING DAQ

```
#! /bin/sh

rcdaq_client daq_setfilerule /home/sbeic/calibfiles/srs-%010d-%02d.evt

for column in $(seq $1 $2) ; do

   for row in $(seq 0 20) ; do
      echo "$column and row $row" | festival --tts
      sleep 2

      echo "Go" | festival --tts

      echo rcdaq_client daq_begin  ${column}555${row}
      rcdaq_client daq_begin  ${column}555${row}

      sleep 3
      echo "End" | festival --tts

      echo rcdaq_client daq_end
      rcdaq_client daq_end


   done
done

rcdaq_client daq_setfilerule /home/sbeic/datafiles/srs-%04d-%02d.evt
```