The RCDAC Data Acquisition System

Martin L. Purschke

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

Brookhaven® National Laboratory

Manhattan

Long Island, NY

RHIC/EIC from space

What I'll be talking about today

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 20-25 places around the world

RCDAQ was chosen to be the main DAQ system of sPHENIX, with several higher-level additions ("Run Control") that I don't have time to talk about today

I will mostly focus on lab-test / R&D-style / test beam setups today

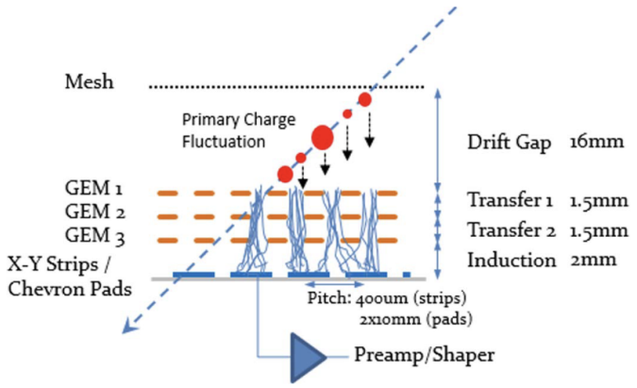
Go to https://www.phenix.bnl.gov/~purschke/rcdaq for the manuals and sample data files etc

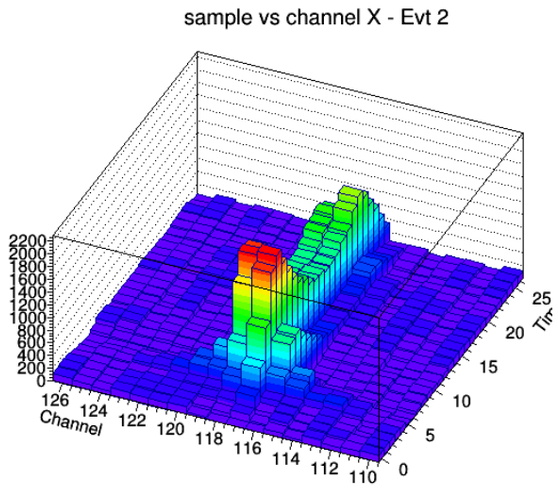
One of the early EIC test beam campaigns with RCDAQ - The Minidrift GEM tracking detector (2014)

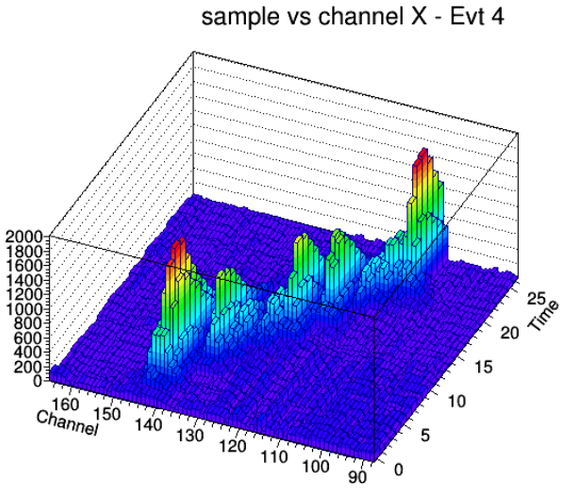
A Study of a Mini-Drift GEM Tracking Detector

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

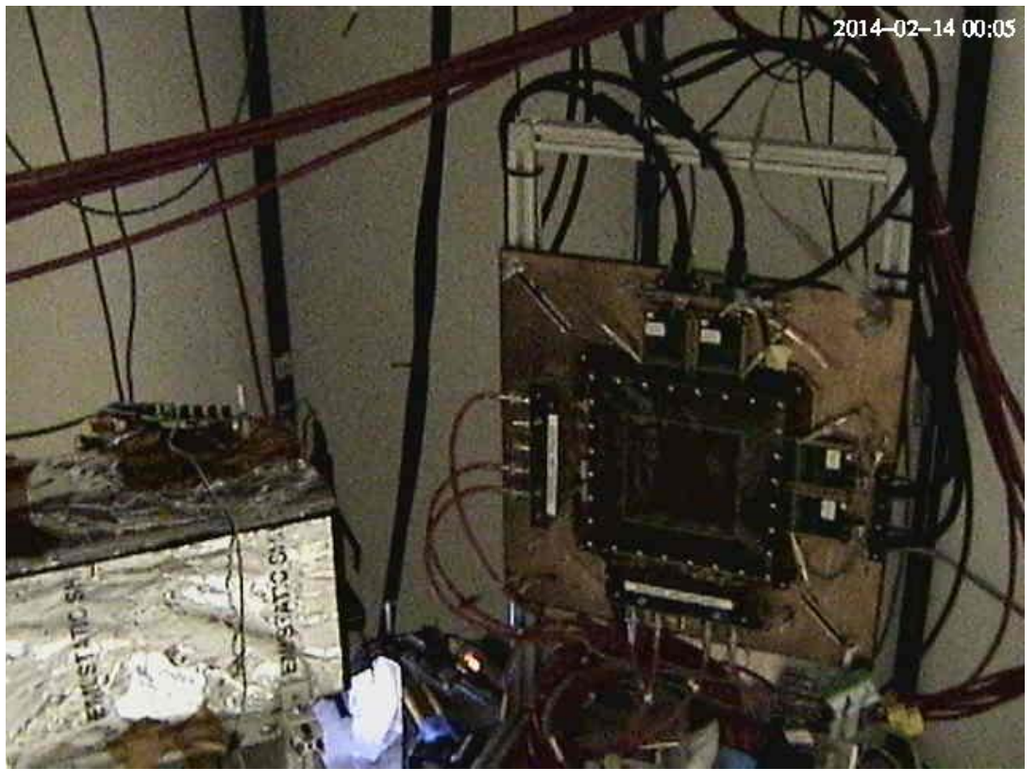
***Abstract*— A GEM tracking detector with an extended drift region has been studied as part of an effort to develop new tracking detectors for future experiments at RHIC and for the Electron Ion Collider that is being planned for BNL or JLAB. The detector consists of a triple GEM stack with a 1.6 cm 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**





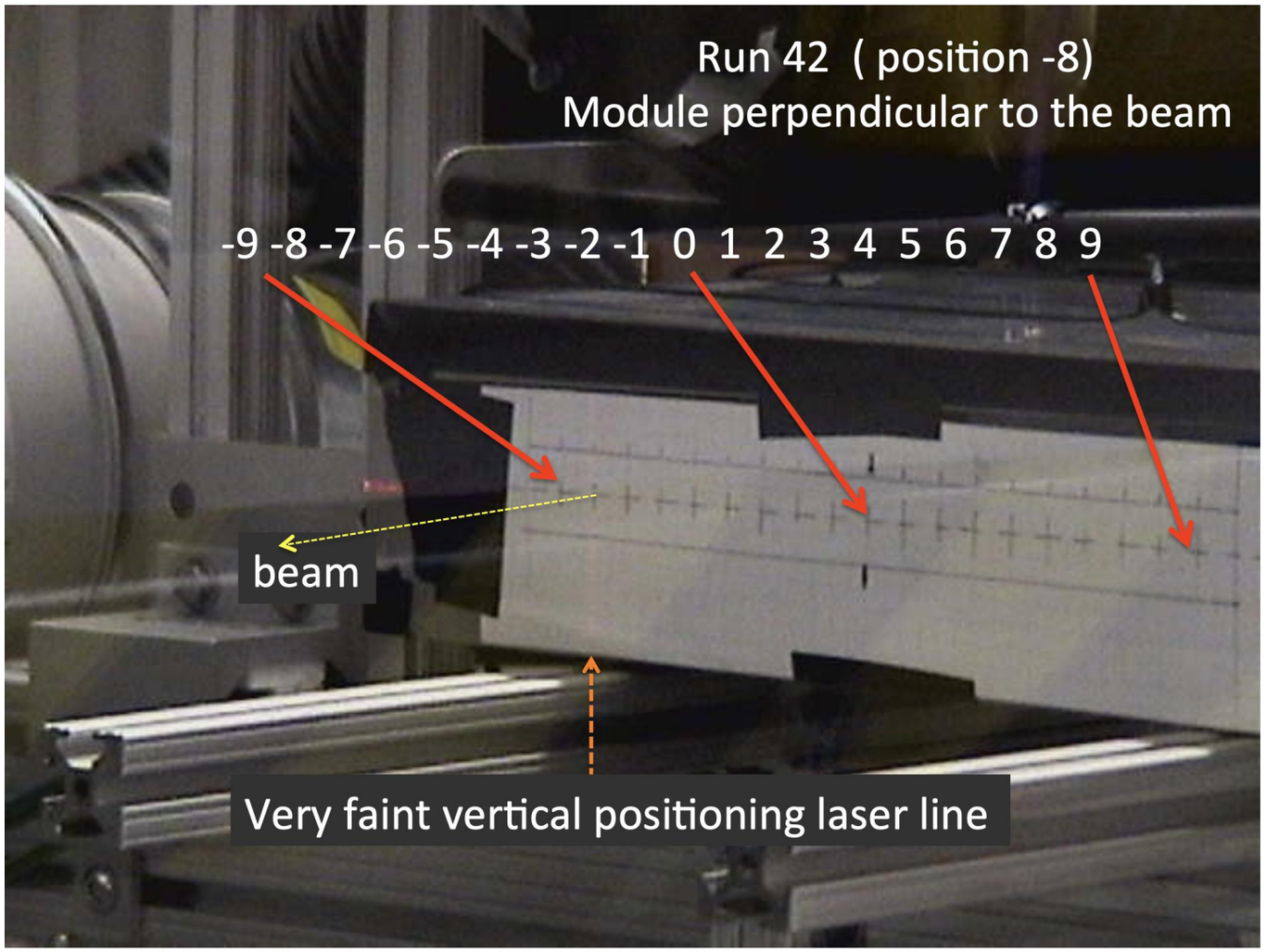


IEEE Transactions on Nuclear Science, vol. 63, no. 3, pp. 1768-1776, June 2016

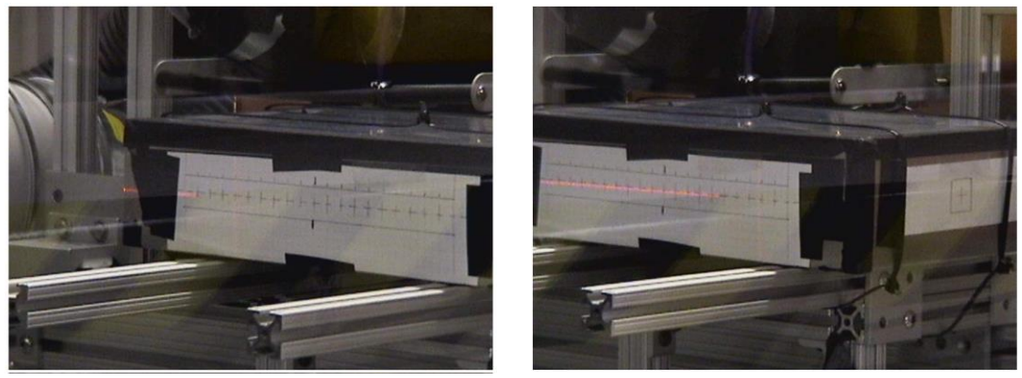


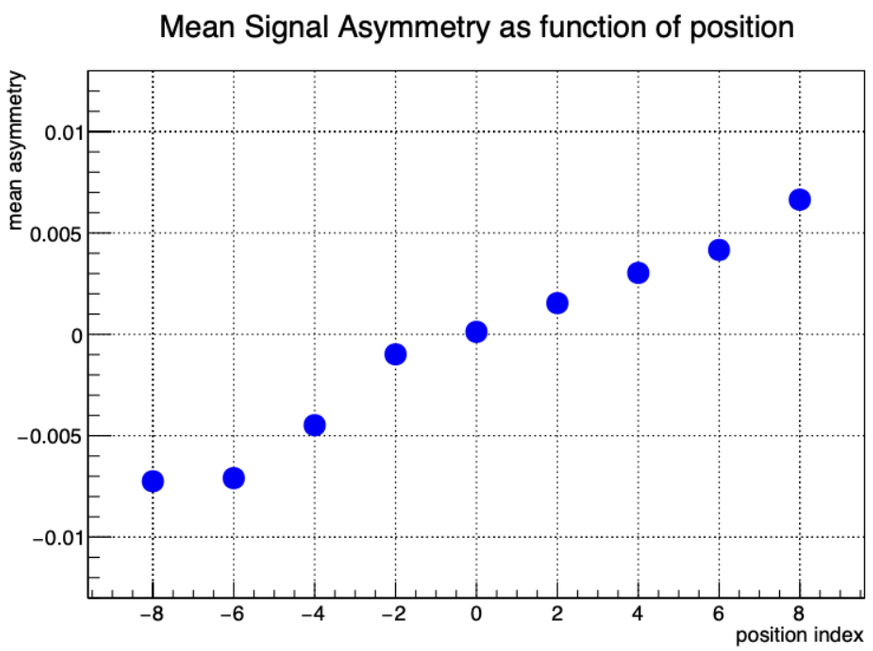
Another one from the EIC calo orbit – Oleg, Craig, myself

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



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!

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, event though no one had ever heard the term when I designed this

I have no time today to talk about the analysis end / online monitoring, etc of this, maybe another time

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 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 CHANNEL DATA																															
	GROUP 0 TRIGGER TIME TAG																															
GROUP 1	GROUP 1 EVENT DESCRIPTION WORD																															
	GROUP 1 CHANNEL DATA																															
	GROUP 1 TRIGGER TIME TAG																															
GROUP 2	GROUP 2 EVENT DESCRIPTION WORD																															
	GROUP 2 CHANNEL DATA																															
	GROUP 2 TRIGGER TIME TAG																															
GROUP 3	GROUP 3 EVENT DESCRIPTION WORD																															
	GROUP 3 CHANNEL DATA																															
	GROUP 3 TRIGGER TIME TAG																															

RCDAQ - The High Points

Each interaction with RCDAQ is a **shell command**. There is no “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 couldn’t re-distribute CAEN software, etc etc

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

Support for different **event types** (one of the more important features)

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!

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

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

On the next slide: what is the response uniformity of a calorimeter module when a shower develops in different places?

Since all collider exps are always fighting for space, you will likely need to make compromises…

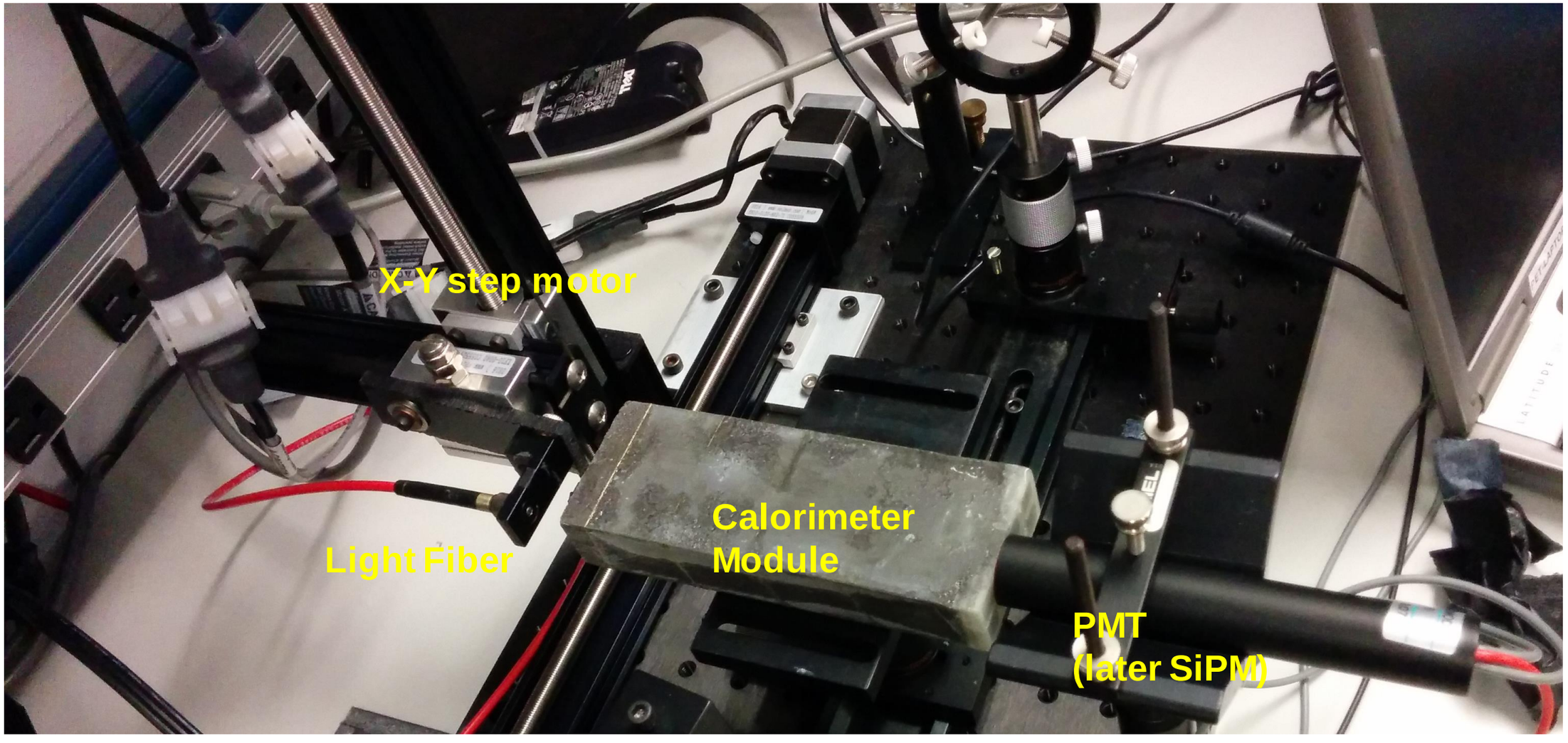
Measurements on autopilot through scripting

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

take a run for each position w/ 4000 events

50 x 25 = 1250 positions (you really want to automate that)

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



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

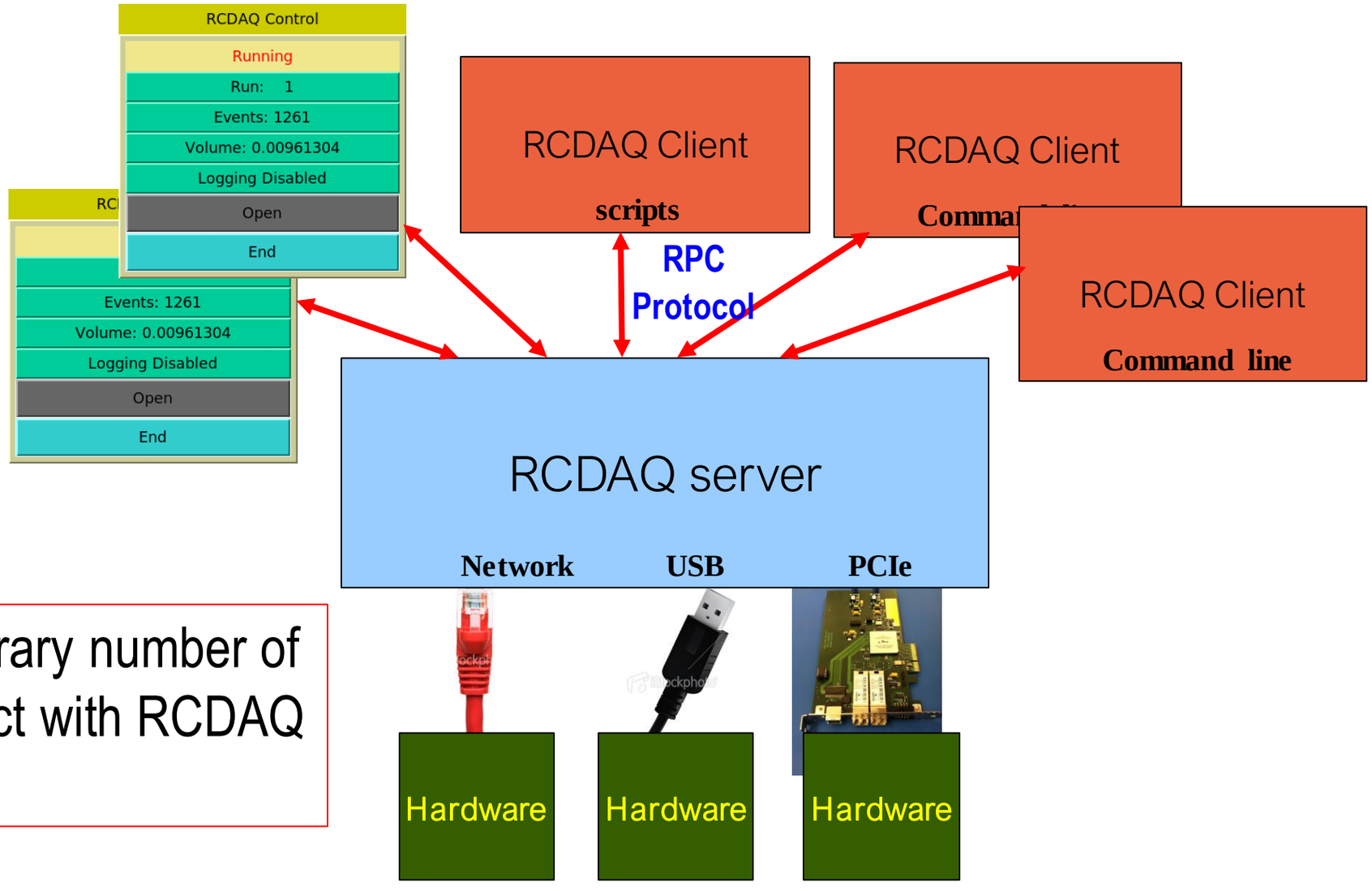
next x

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
```

As an aside: I use this example as the basis of a whole CERN School lecture, there's a lot more good stuff hiding here – happy to talk about it one day

The RCDAQ client-server concept



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

The RCDAQ server does not accept *any* input from the terminal. All interaction is through the clients.

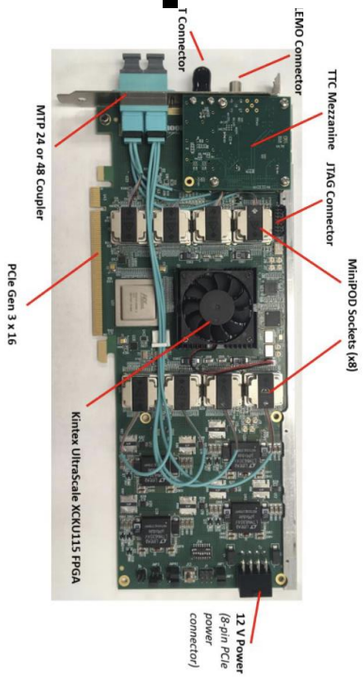
Example:

```
$ rcdaq_client daq_begin
```

Some standard devices implemented in RCDAQ

RCDAQ

PCIe



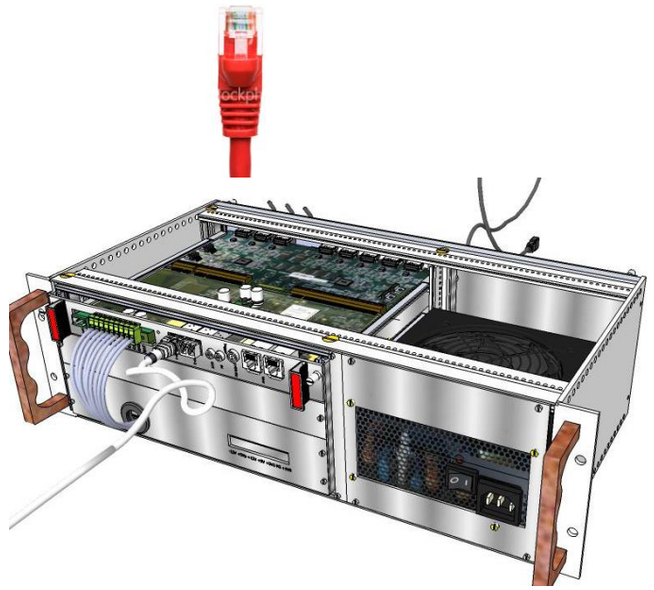
FELIX Card

PCIe

sPHENIX digitizer

DRS4 Eval board

“USB Oscilloscope”



The CERN RD51
SRS System

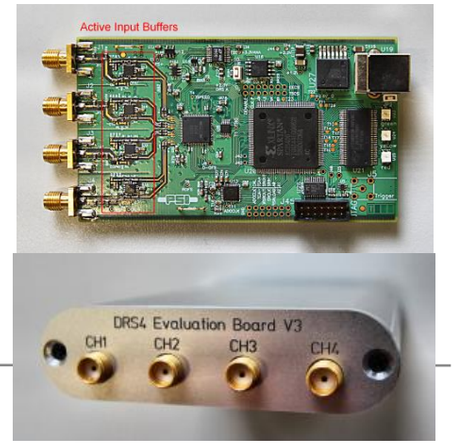
PCIe

The CAEN V1742
waveform digitizer

There are *many* more not shown

Many devices that you can often find in your institute already, or in the CAEN catalog…

Example: reading out a DRS4 Eval board




```
$ rcdaq_client load librcdaqplugin_drs.so

$ rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 140 3

$ daq_open

$ daq_begin
  # wait a while…

$ daq_end
```

You see, each interaction is a separate shell command!

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…

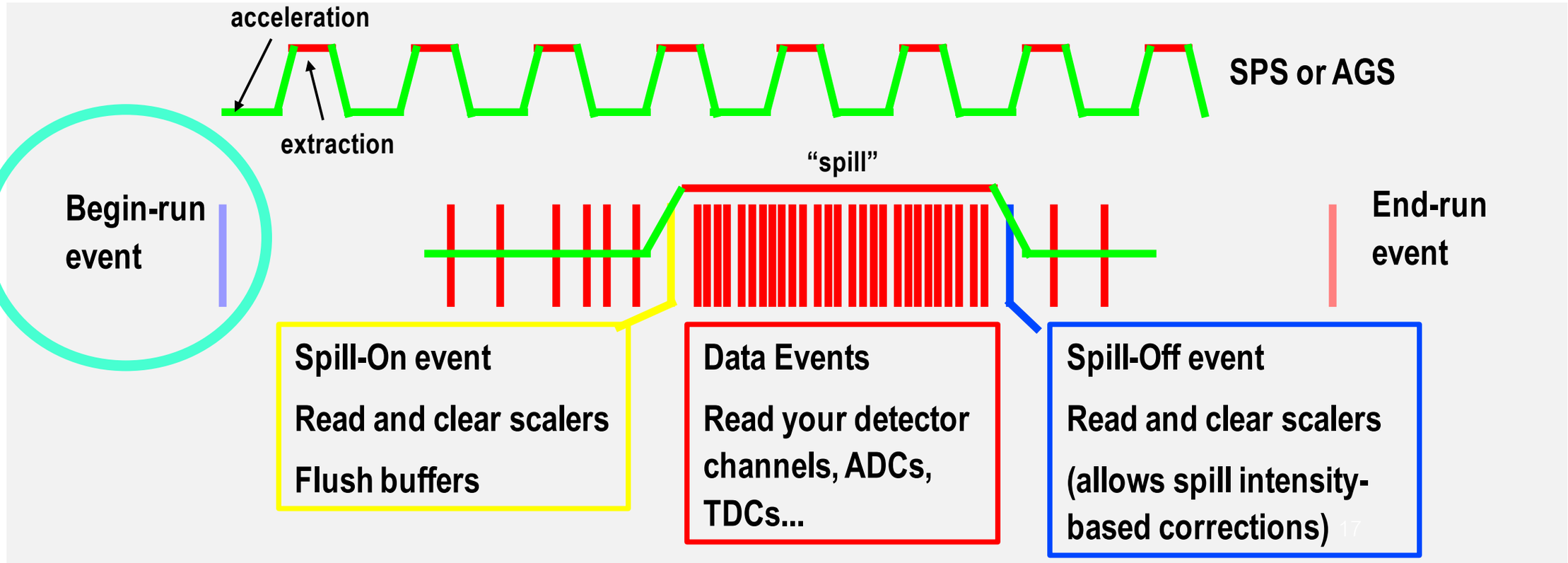
Let me show you how we always capture the RCDAQ setup itself

Reading different things with different Event Types

You would think of the DAQ as “reading out your detector”

Very often, it is necessary to read out different things at different times.

Let’s go to the CERN-SPS (or the BNL AGS) for an example:



Remember this?

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 140 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 140 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 140 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 140 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)

Here is the actual setup script for our TPC (FELIX)

Abridged version, just the essentials

```
#!/bin/bash


RunType=beam
H=$RCDAQHOST
[ -z "$H" ] &&  H=$(hostname)


MYSELF=$(readlink -f $0)
rcdaq_client daq_clear_readlist
rcdaq_client create_device device_file 9 900 "$MYSELF"

rcdaq_client load   librcdaqplugin_dam.so
rcdaq_client create_device device_dam 1 4${H:4:2}1 1 128

rcdaq_client daq_set_runcontrolmode 1
```

More about capturing your environment

Many times you capture things only “just in case”

You don’t routinely look at them in your analysis (such as cam pictures)

But if you have some inexplicable feature, you can use the 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

"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
```

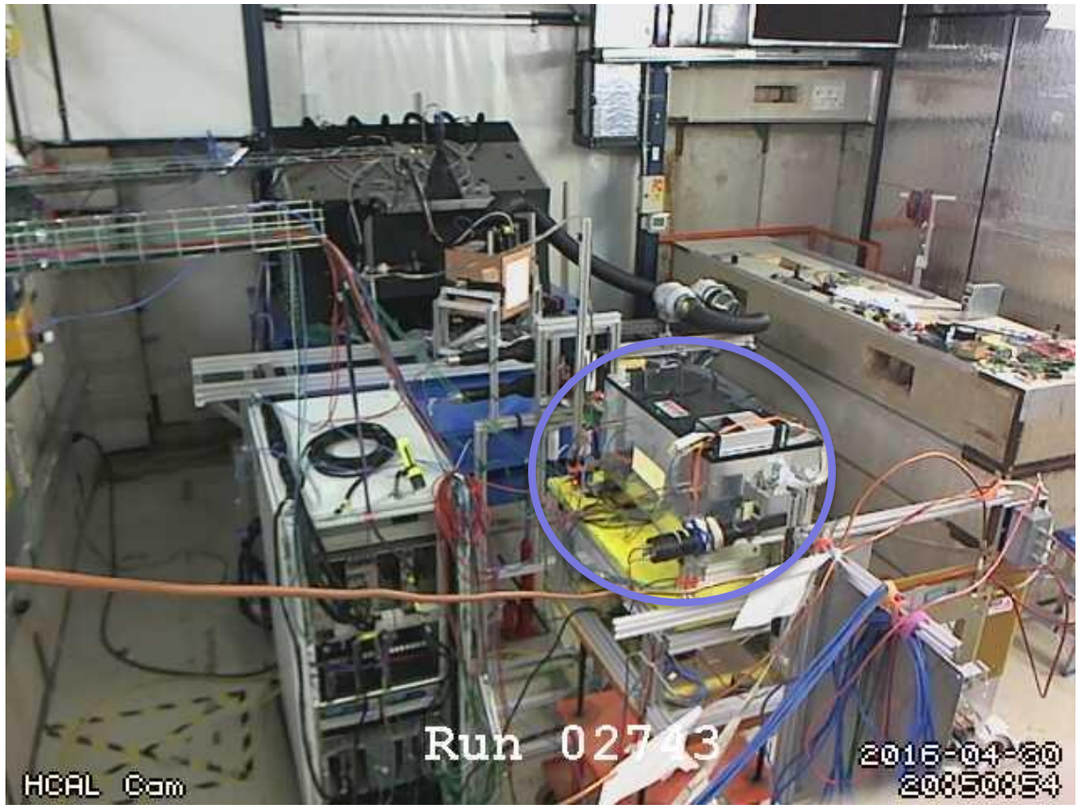
```
$ ddump -t 9 -p 923 beam_00002268-0000.prdf
S:MTNRG   = -2      GeV
F:MT6SC1  =  11846      Cnts
F:MT6SC2  =  7069       Cnts
F:MT6SC3  =  3883       Cnts
F:MT6SC4  =  0          Cnts
F:MT6SC5  =  283048     Cnts
E:2CH     =  1058   mm
E:2CV     =  133    mm
E:2CMT6T  =  74.13  F
E:2CMT6H  =  37.26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

More Forensics (my poster child why this is so useful…)

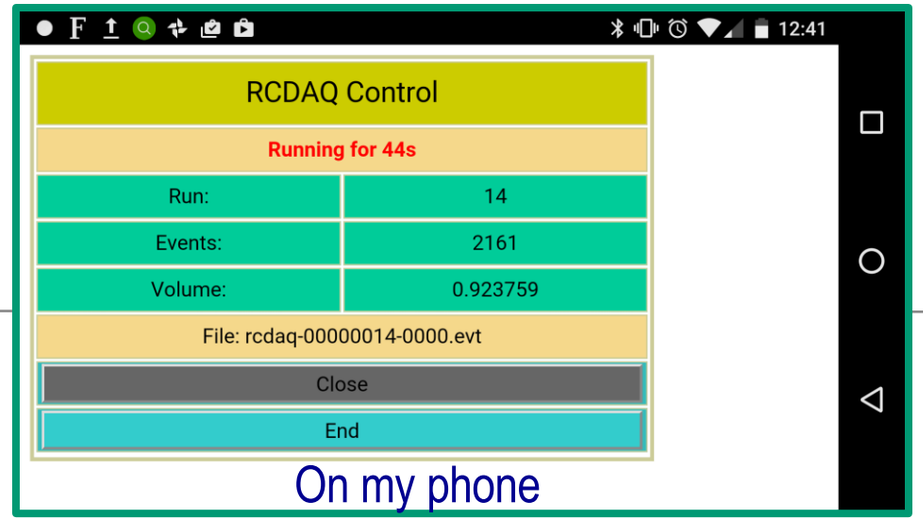
"There is a strange effect starting in run 2743. There is a higher fraction of showering than before. I cannot see anything changed in the elog."

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

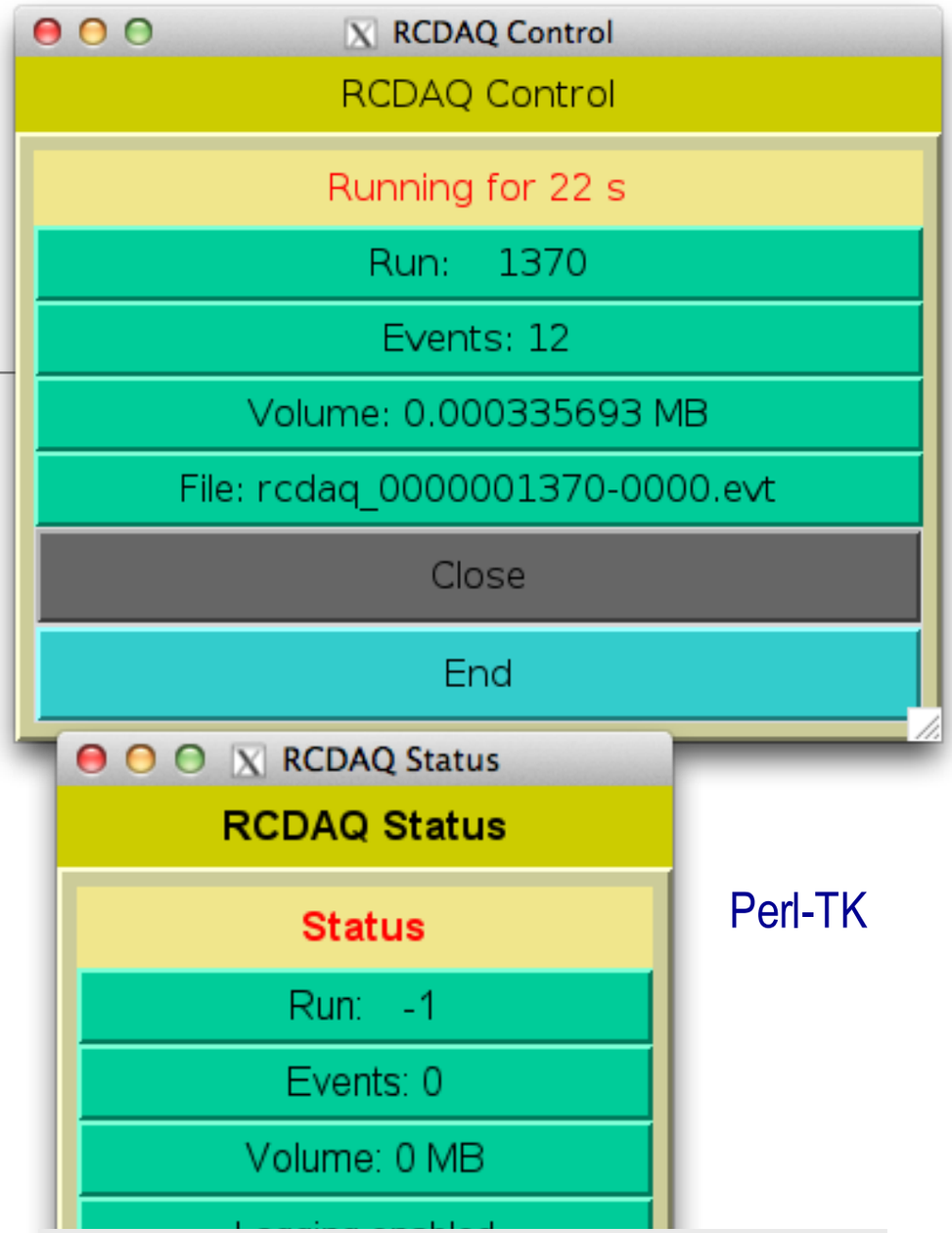
```
$ ddump -t 9 -p 940 beam_00002742-0000.prdf > 2742.jpg
$ ddump -t 9 -p 940 beam_00002743-0000.prdf > 2743.jpg
```



GUIs

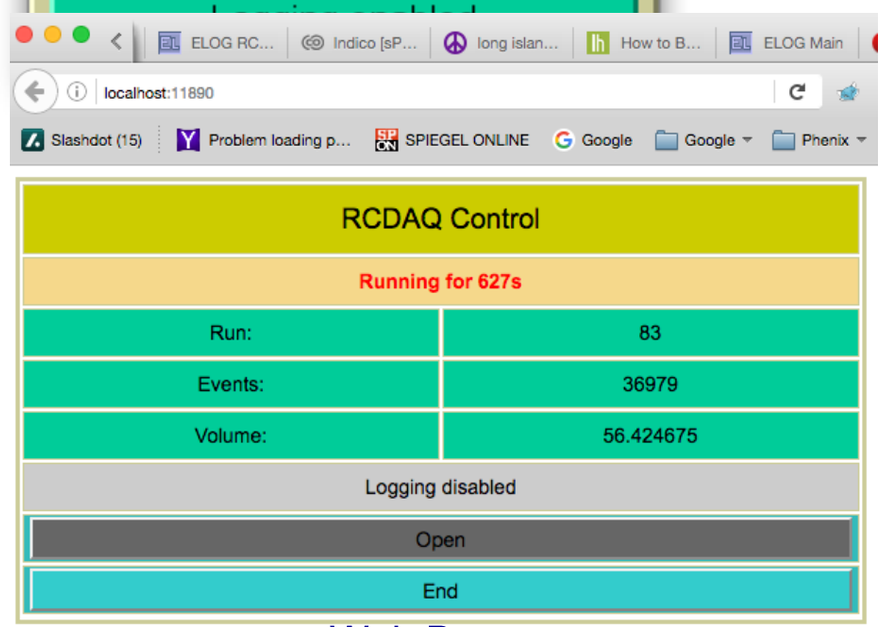


On my phone



Perl-TK

- **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)



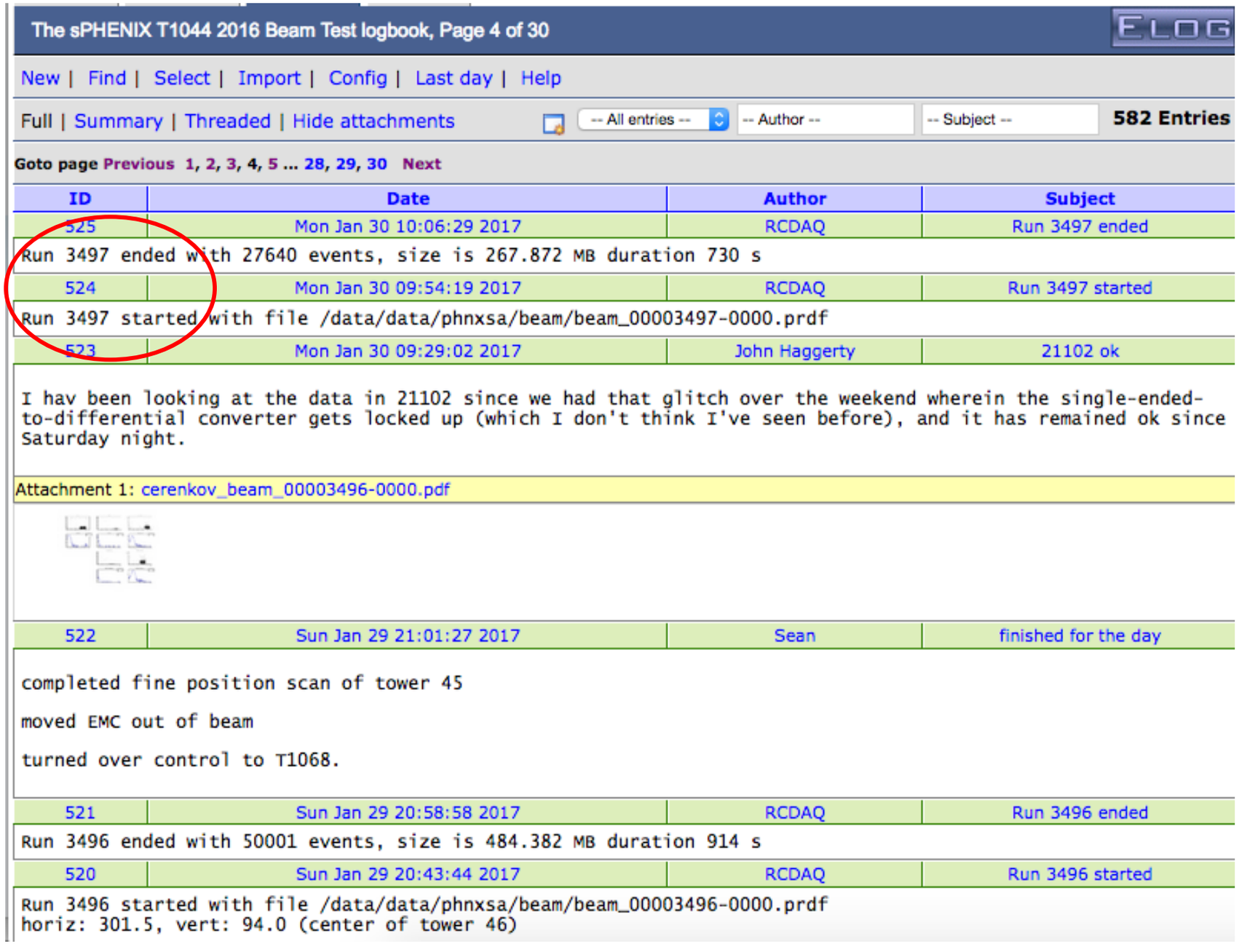
Web Browser

Automated Elog Entries

RCDAQ can make automated entries in your Elog

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

Gives a nice timeline
and log



What's in it for ePIC

I take pride in having one of the easiest-to-use and most versatile DAQ systems out there (for a "known" device from scratch to seeing a histogram: one hour)

RCDAQ can read out our detector-specific devices (think FELIX), and many commercial devices that are often used in lab tests (like CAEN V1742, DRS4, …)

The latest addition is Nalu's ASocV3 (in progress), and I have an ePIC ROC and collaborate with Norbert

We have been (and are, all BNL/YaleSBU/FIT test beams) using RCDAQ for our R&D, ample of operational experience (Alexander, Bob, Craig, myself, …)

Superb support for automated measurements that we will need for many tests

Support for analysis and online monitoring (not enough time today, maybe another time)

Jana2 support for the data format… planned, effort with DL was started, but life interfered

BTW: To this day we maintain a "bridgehead" at Fermi and can spin up a readout system in an hour after you are through the front gate

There is one thing I haven't told you yet…

What does "RCDAQ" stand for?

The "Really Cool Data Acquisition"

Thank you!

This page is intentionally left blank

Ok, that was a lot about test beams… what about sPHENIX

I harped on lab tests / test beams etc a bit because that's what ePIC will be busy with for some time

What do we do in sPHENIX?

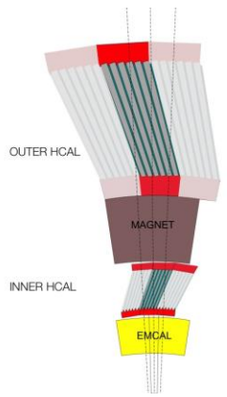
In short, each detector element connects to a PC that runs its own instance of RCDAQ

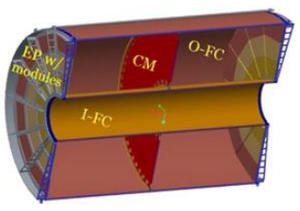
It's all "glued" together with the timing system

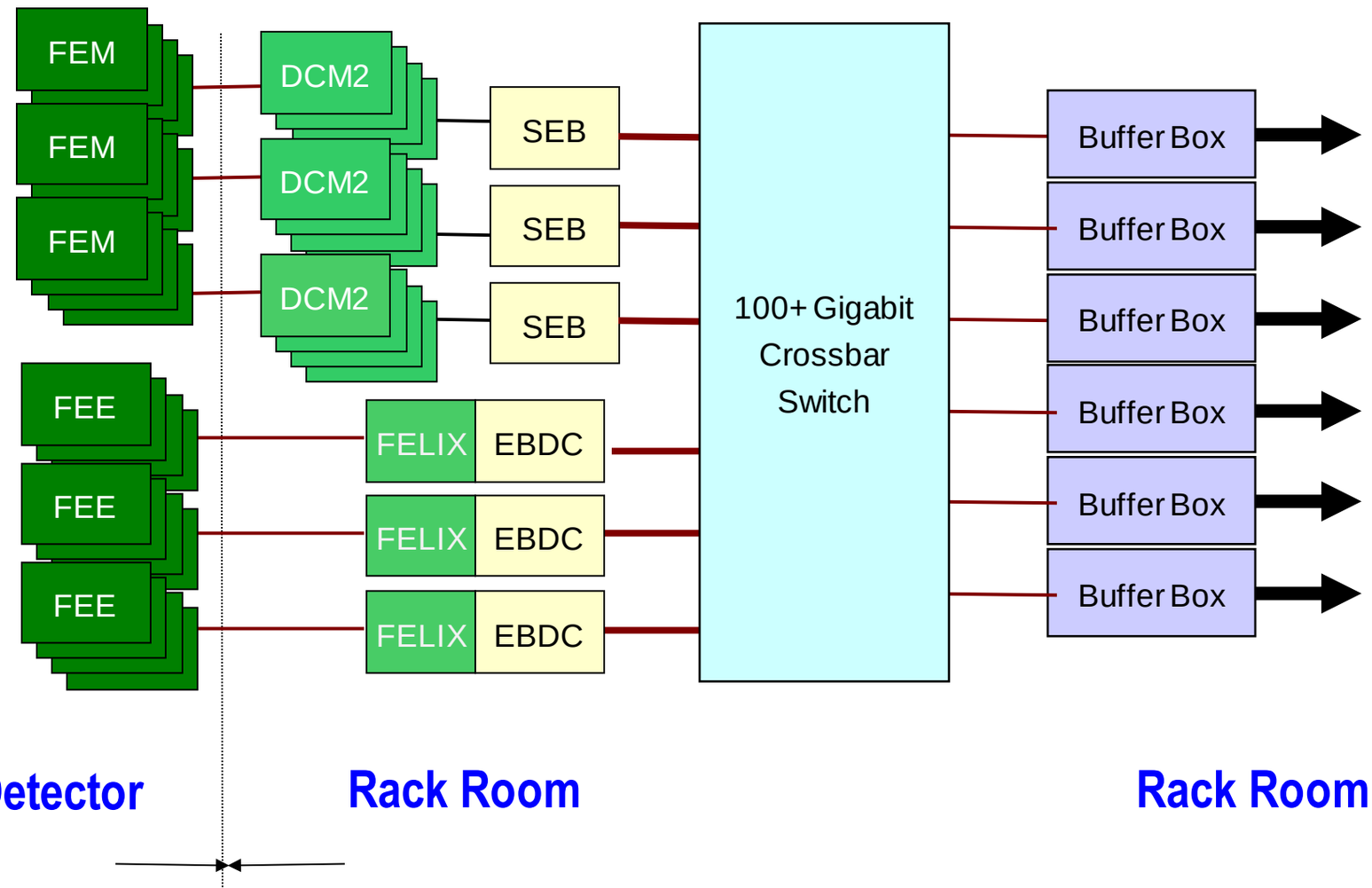
And our (in 2023) 52 RCDAQ instances are controlled by a meta-process "RunControl"

The RCDAQs run in a "run control mode" that reduces some of their autonomy so that Run Control is, well, in control

sPHENIX DAQ Bird's eye view

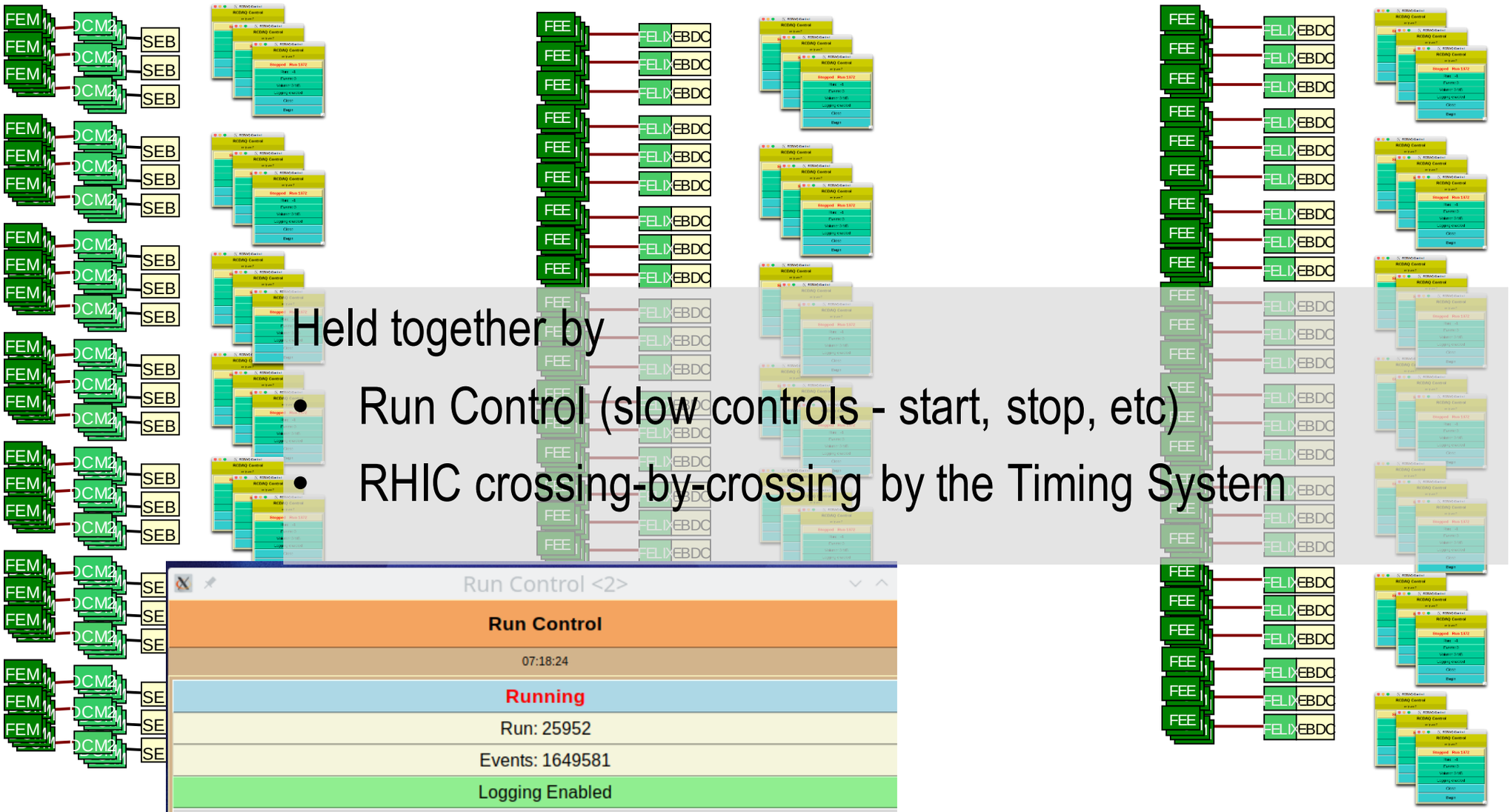




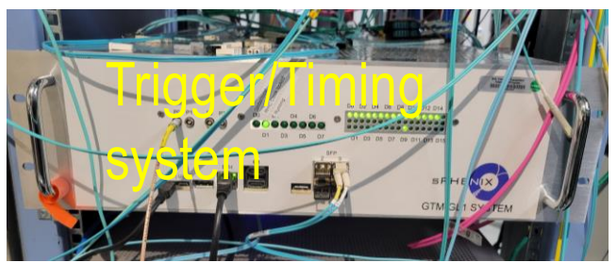


- DCM-2 receives data from digitizer, zero-suppresses and packages
- SEB collects data from a DCM group (~20)
- EBDC Event Buffer and Data Compressor (~40)
- Buffer Box data interim storage before sending to the computing center (6)

many, many RCDAQs in sPHENIX



- Held together by
  - Run Control (slow controls - start, stop, etc)
  - RHIC crossing-by-crossing by the Timing System



“Meta Data” Packet list from a recent test beam

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

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

Captured at begin-run

Captured again at spill-off

Coming back to the “shell command” feature

For the last 3 minutes, I want to harp some more on the superiority of that “everything is a shell command” approach

Often I’m learning of a new ingenious way to use this aspect for something cool

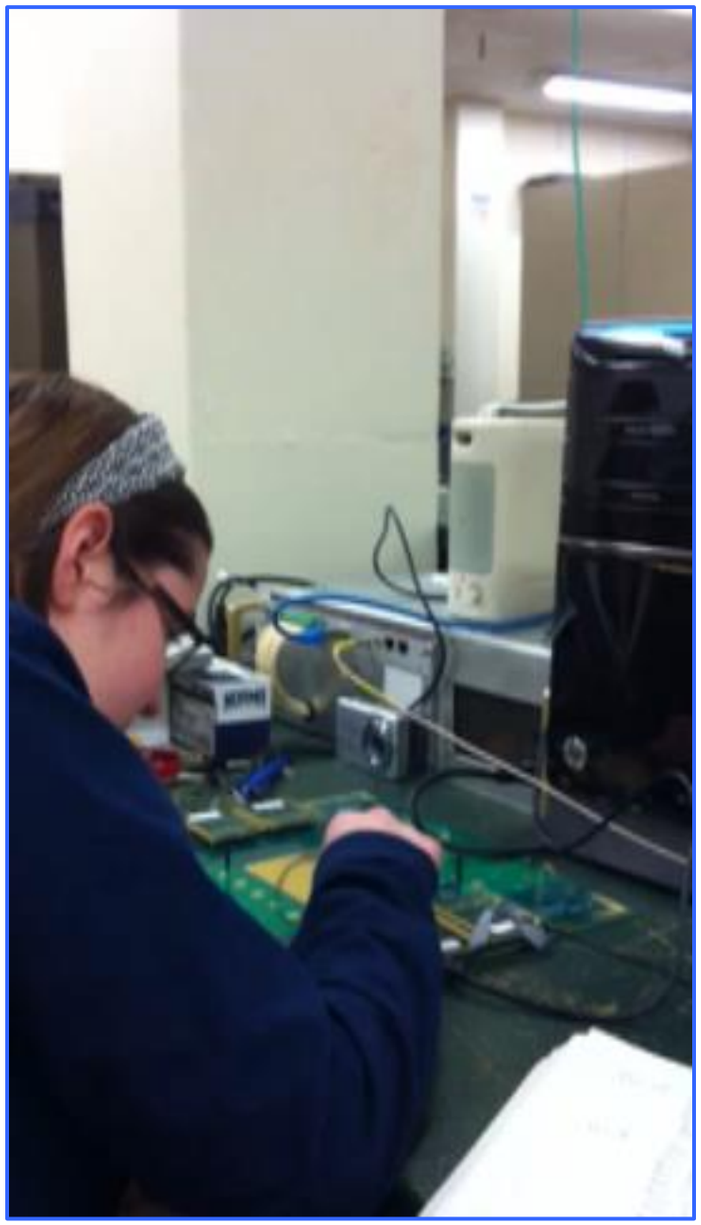
A real good tool gets used in ways that the designer did not envision… but it works!

A group needed to test a few thousand pads on a plane if they a) work and b) are connected right.

Inject charge into the pads one by one... **but you can't take your eyes (or the probe) off the pad plane or you lose your position**

They came up with…

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
```

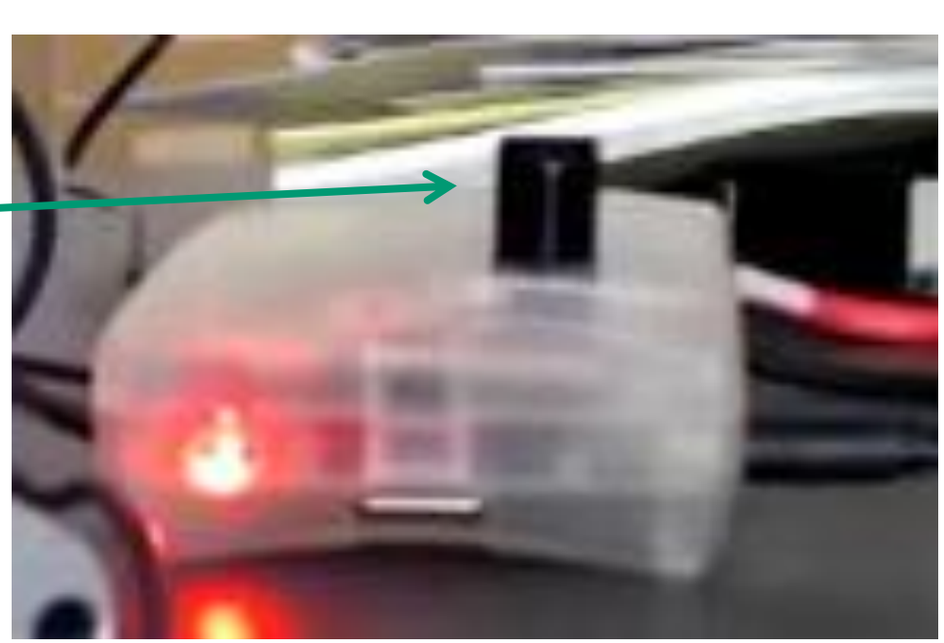
One more cool thing

Anything that's capable of issuing a shell command can control the DAQ

I have said (but not shown you yet) that the DAQ can be controlled remotely, through the network

I have a Raspberry Pi connected here that I have set up so it controls RCDAQ running on my Laptop

And you see it has developed some kind of growth on its head… That's an infrared receiver



We know we can assign arbitrary commands to buttons pressed on virtually any IR remote

I guess you see where this is going…

Autopilot example: “Tile Mapping” at the Fermi Test Beam Facility

“Tile mapping” refers to mapping the position-dependent response of a hadronic calorimeter tile.

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

The FTBF M2.6 table is controlled via the accelerator controls (ACNET) – some caution required

This setup exercises most of the aforementioned features: scripting and reacting to the FTBF spill, network transparency (we cannot access ACNET from the DAQ machine, but an ACNET-enabled machine can control our DAQ)

I told you about my cameras, right?