RCDAQ and EIC and all of that

Martin Purschke, BNL

A bit of the RCDAQ history

Dedicated EIC test beams and other data-taking campaigns

High points of RCDAQ

Meta data

Actual EPIC detector readout

History of RCDAQ

- From 2006 or thereabouts we had quite a number of smaller-scale data taking campaigns, R&D for future detectors, but also our then-medical imaging R&D… RCDAQ was "born"
- A large number of commercial or otherwise widely used devices were implemented (today about 120 "active" ones).
- In 2012 we ramped up several EIC-themed measurements of various detectors, GEMs, calorimeters, TOF, RICH, … all of which used RCDAQ
- RCDAC's first test beam at FermiLab's FTBF was in 2013 (some calorimeter modules)
- 2014 – the EIC FLYSUB consortium at the FTBF, sort of the Super Bowl of data taking with 5 setups – BNL (Woody/Purschke), SBU (Hemmick/Dehmelt), UVa (Kondo), Yale (Majka/Smirnov), FIT(Hohlmann)
- 2015 - sPHENIX adopts RCDAQ as its DAQ technology, first sPHENIX test beam w/ RCDAQ in 2016 (outer HCal and Emcal, different W calos)
- Also a parade of "field-unrelated" users, UTexas, MPI Munich, NIKHEF, Weizmann, … ~15 users and also contributors (usually plugins, in a minute)

EIC-themed test beams and other datasets

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

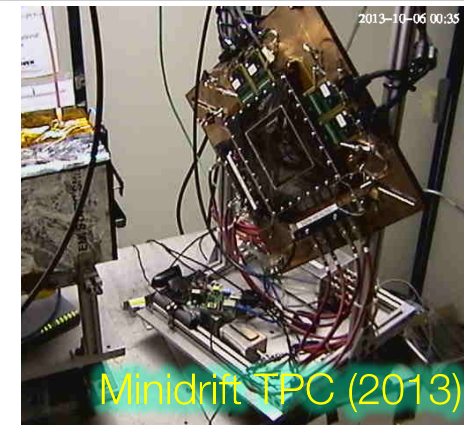
Estimated 25 active RCDAQ installations in the EIC orbit + ~30 elsewhere

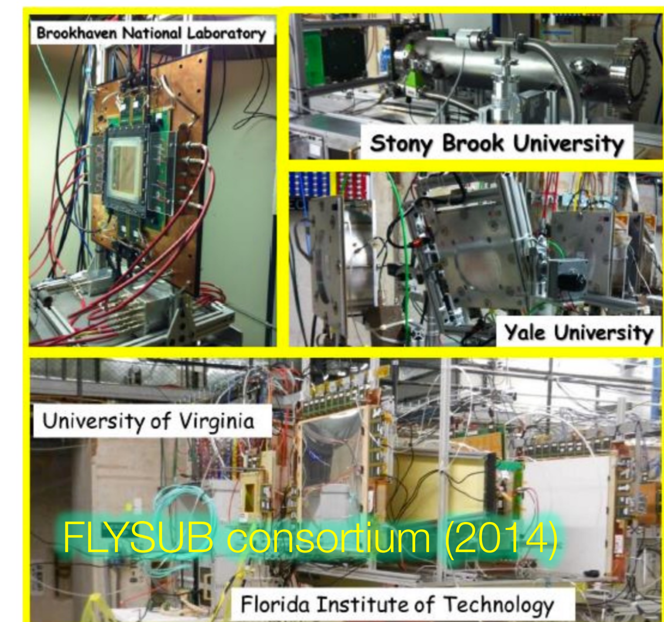
Usual entry by ease-of-use for standard devices (DRS, SRS, CAEN, …) and support for fully automated measurement campaigns

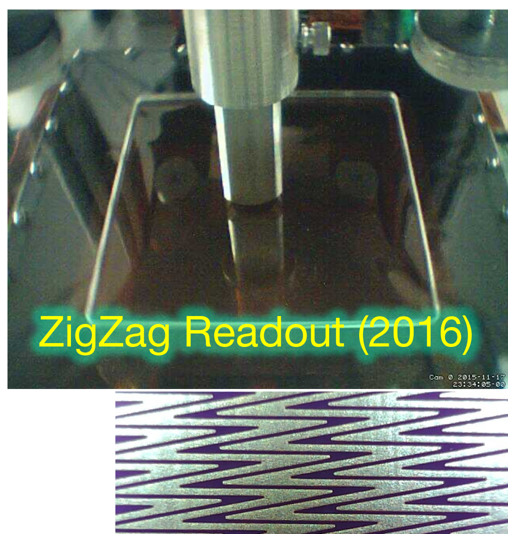
From Markus Diefenthaler at the Users meeting:

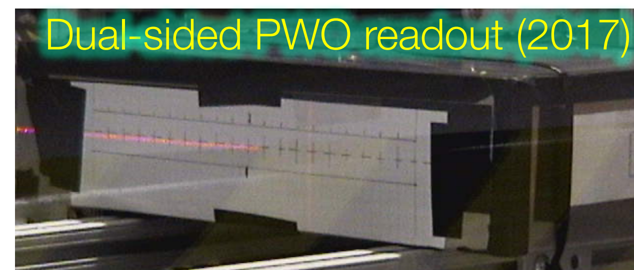
- **Detector Simulations**
  - Validation of Geant4: Make test-beam setup and results available.

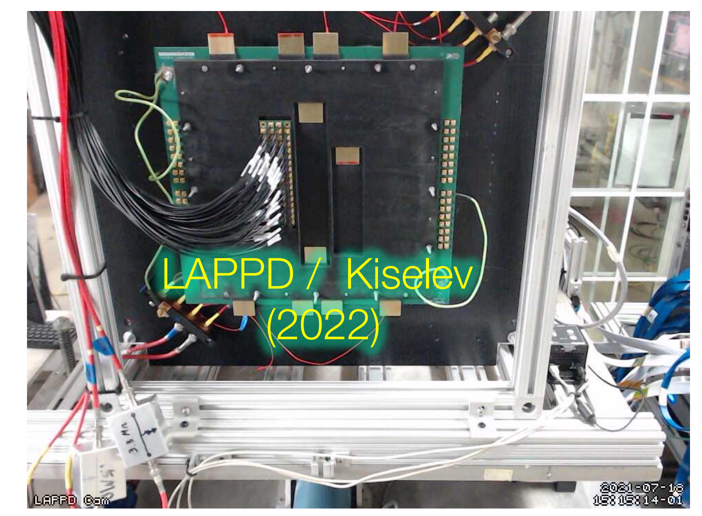
Not sure what prompted that remark… I cannot see how this could be a problem











Some RCDAQ Concepts

- Modularity
- Binary payload format agnostic
- Different event types
- Set of tools to inspect / display / manipulate files
- Online monitoring support
- Electronic Logbook support
- OS integration
- Interface to community analysis tools (these days: root and 3rd-party frameworks)
- **It's gotta be fun to use!**

That's quite a list…

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: Parts of the RCDAQ’s native format have their roots at the CERN-SPS, and the Bevalac Plastic Ball experiment in the 80’s – that’s a solid “test of 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

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

Events and Packets

Deliberately lightweight and minimalistic packet and event header structures

No assumption on what “each structure” has in common – you are almost always wrong

```
typedef struct evt_data
 {
   unsigned int evt_length;
   int evt_type;
   int evt_sequence;
   int run_number;
   int date;
   int time;
   int reserved[2];
   int data[];
} *evtdata_ptr;
```

8 32bit fields
=32 bytes

```
typedef struct packet_data
 {
   unsigned int packet_length;
   short packet_id;
   short packet_type;
   short packet_hitformat;
   short packet_padding;
   short reserved[2];
   int   data;
 } *packetdata_ptr;
```

4 32bit fields
=16 bytes

Max length for both Event and Packet 16GBytes (length is in units of int’s)

“Binary payload agnostic” – what is that?

Most of the “devices” we read out provide their data in some pre-made (and usually quite good) compact binary format already. Usually done in some FPGA.

(Think DAM/FELIX.)

All you want to do is to grab the blob of data, stick it into a packet, put a label (packet header) on that says what’s in it, done.

That is literally all we do to the data

From that point forward, the DAQ does not care. The “FedEx” approach – they ship boxes, we ship ***packets***.

More generally: Usually we store data from our readout devices, but we must be able to store literally *anything* in our data stream.

Want to store an Excel spreadsheet? A text file? A jpeg image? No problem.

If you think ”why would one want to do that!”, just wait a few minutes.

Everything in RCDAQ is a shell command

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

That makes everything inherently scriptable, and 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

Nothing beats the shell in flexibility and parsing capabilities (of course you also have GUIs)

You can type in a full RCDAQ configuration on your terminal interactively, command by command (although you virtually always want to write a script to do that)

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.

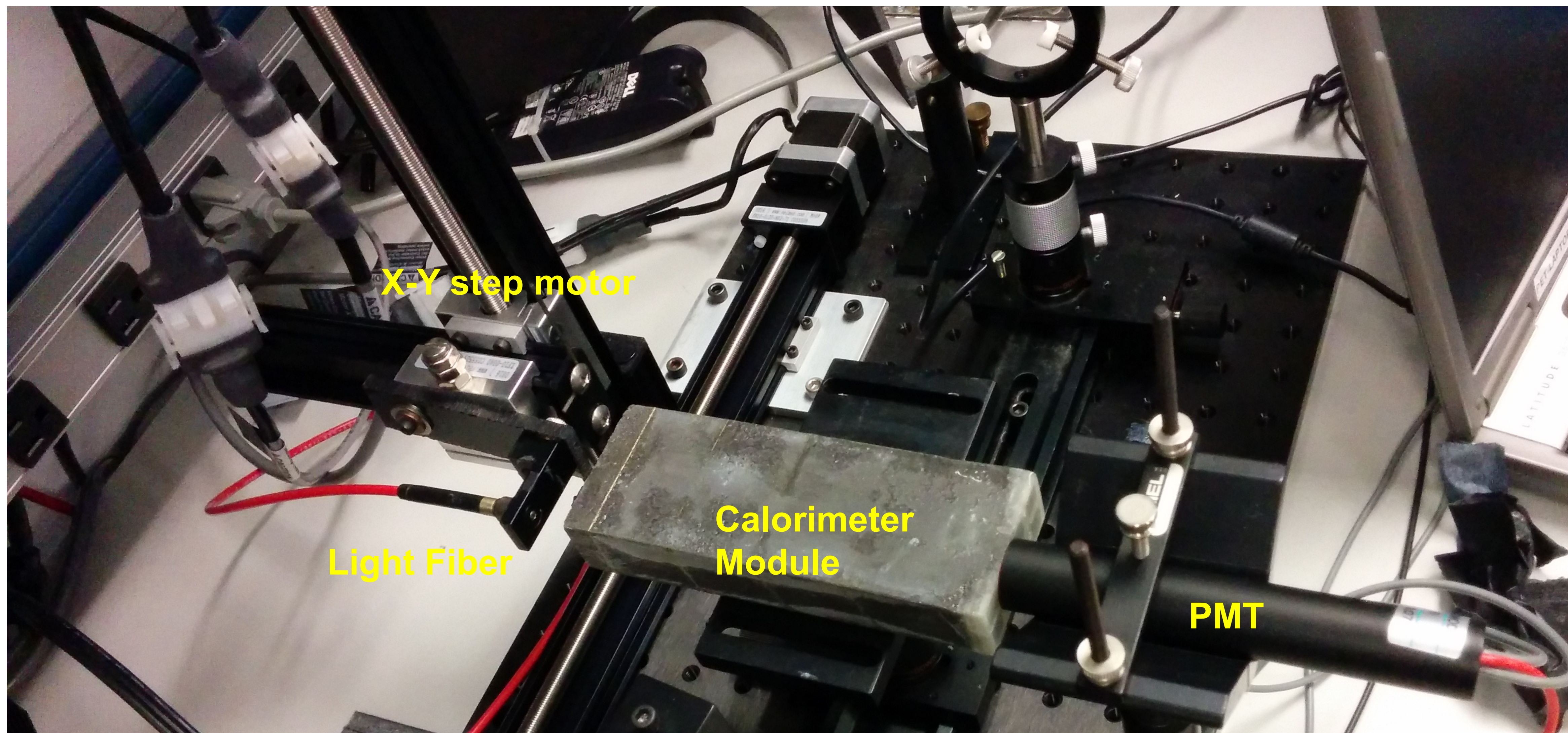
Measurements on autopilot

You want to run measurements where you step through some values of a parameter completely on autopilot

Here: Move a light fiber with 2 step motors, 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

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

The DAQ operation becomes an integral part of your shell environment

Automatic end after 40

25 positions in y

move the Y motor

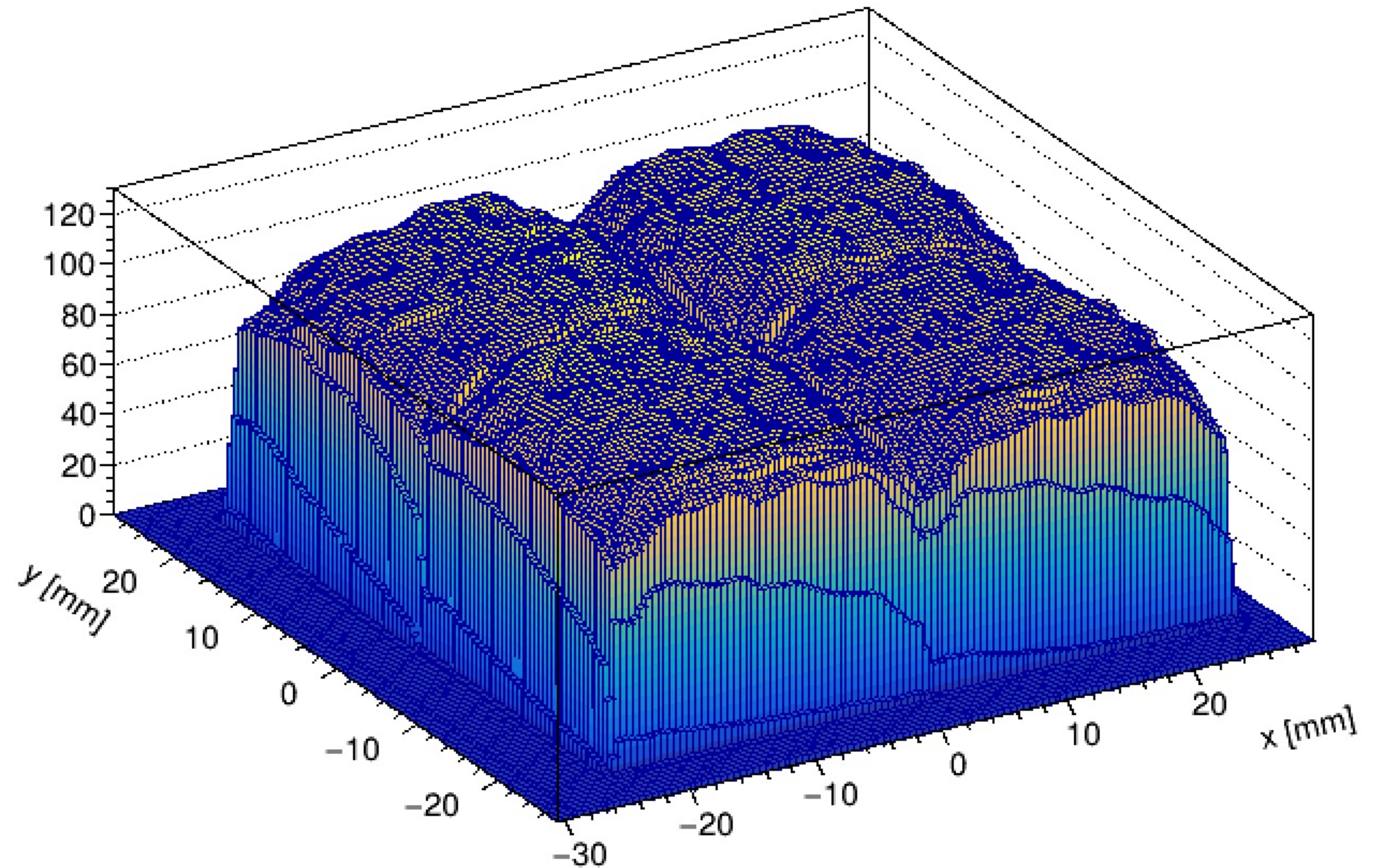
50 positions in x

move the x motor

start the DAQ

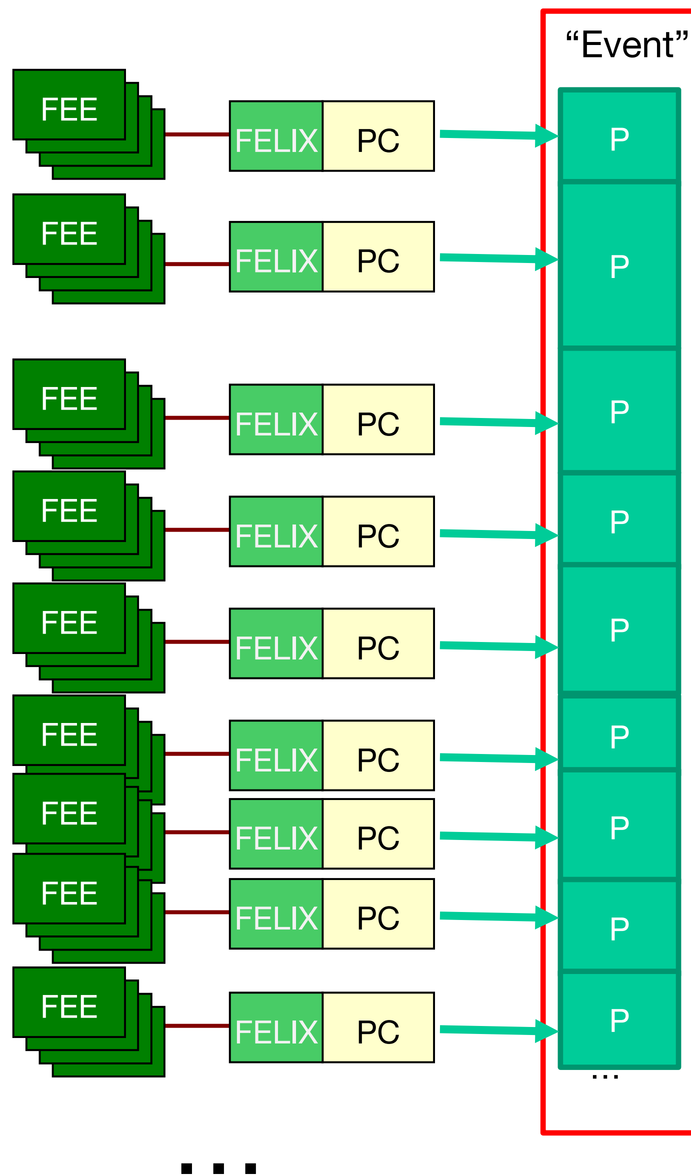
next x

next y




```
NTY)
MENTX)
done
```

Event / Streaming Data Structures



Each Front-End Card generally contributes what we call a “Packet” to the overall event structures

A Packet ID uniquely identifies the detector component / front-end card where it comes from

A hitformat field identifies the format of the data, und ultimately selects the decoding algorithm

You interact with a standard set of APIs to access the data

We can change/improve the binary format and assign a new hitformat for a packet at any time

Insulation of offline software from changes in the online system

API delivers the data independent of internal encoding

Very rough number: 1200-2500 packets collectively

In case of a triggered DAQ, such an event structure and the packets therein would correspond to the data from one crossing

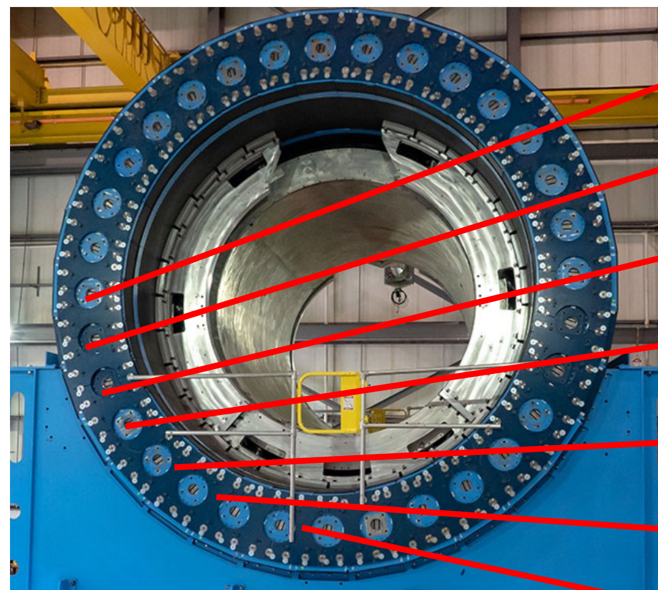
Example: Full EPIC Outer HCal, Real Events

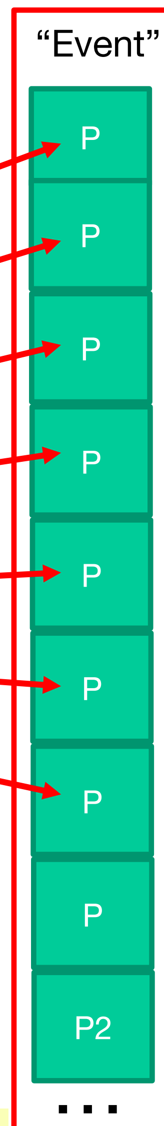
That's one of the detectors that will survive into Detector 1

For us it's subsystem #8, makes 32 Packets with IDs 8001 - 8032

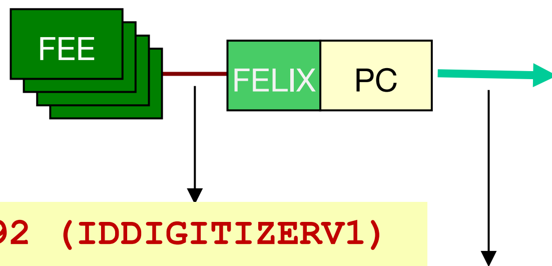
Hitformat

```
$ dlist oHCal-00000100-0000.evt
Packet   8001  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8002  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8003  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8004  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8005  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8006  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8007  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8008  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8009  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8010  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8011  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8012  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8013  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8014  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8015  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8016  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8017  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8018  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8019  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8020  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8021  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8022  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8023  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8024  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8025  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8026  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8027  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8028  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8029  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8030  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8031  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
Packet   8032  1000 -1 (sPHENIX Packet)   92 (IDDIGITIZERV1)
```





...

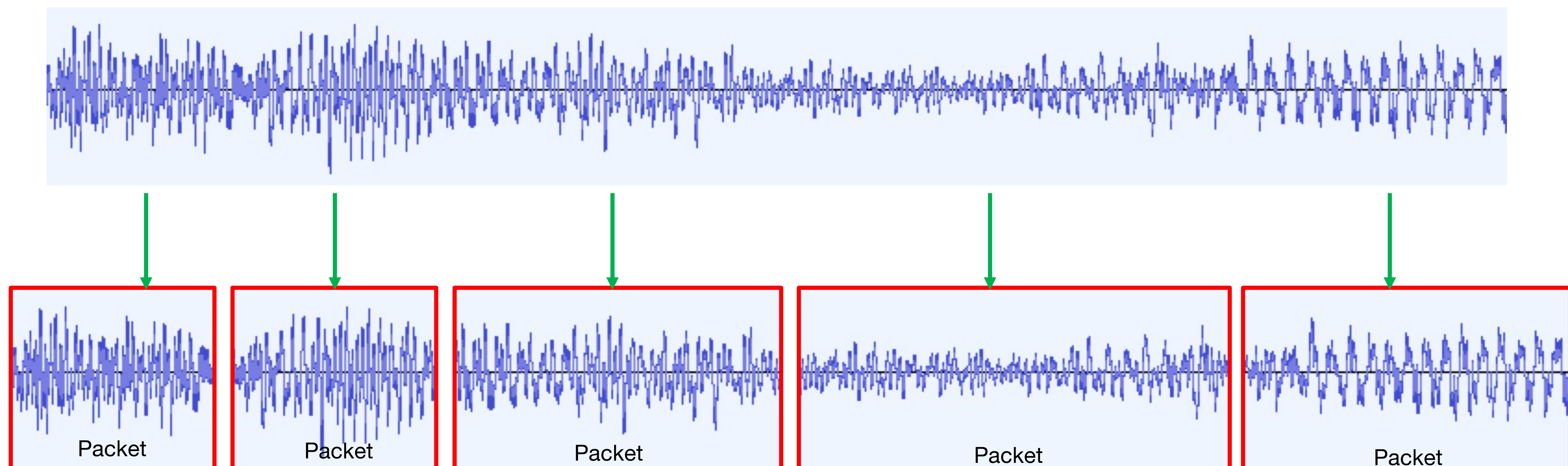


Different formats from the front-end or after the full chain – no change in analysis code!

Streaming Readout and Packets

For streaming data, the "Packet" paradigm changes its meaning a bit

It becomes like a packet in the Voice-Over-IP sense - VoIP is chopping an audio waveform into conveniently-sized chunks to transfer through a network



We are chopping the streaming detector data into conveniently-sized packets for storage

Here: Streaming sPHENIX TPC data (entire sPHENIX tracking system streams!)

```
$ dlist  rcdaq-00002343-0000.evt -i
 -- Event      2 Run:  2343 length: 5242872 type:  2 (Streaming Data)  1550500750
Packet  3001 5242864 -1 (sPHENIX Packet)  99 (IDTPCFEEV2)
$
```


14

Streaming readout, here we come!

Past the Front-end, the readout is **completely oblivious to the readout mode**

It doesn't care how the front-end + FELIX arrived at the decision to send up the data.

Triggered or streaming, from the readout perspective they look 100% the same

Our FELIX drivers adhere to POSIX standards (here: our INTT):

```
FD_SET(_intt_fd, &read_flags);

sel = select(_intt_fd+1, &read_flags, NULL, NULL, &timeout);

if (FD_ISSET(_intt_fd, &read_flags))
    {
        // data available . . .
```

```
sevt->packet_length =  SEVTHEADERLENGTH;
sevt->packet_id = ipacket;
sevt->packet_type=2;
sevt->packet_decoding = IDINTT_v1;
sevt->reserved[0] = 0;
sevt->reserved[1] = 0;
uint16_t *dest = (uint16_t *) &sevt->data;

int ret = read(_intt_fd, dest, _length);

sevt->packet_padding = ret%2 ;
sevt->packet_length += (ret + sevt->packet_padding);
return sevt->packet_length;
```

So we grab the data when they come

RCDAQ doesn't know "readout"…

Huh?

Out of the box, RCDAQ doesn't know how to read out any kind of hardware. Nada.

RCDAQ is "taught" about any hardware device by way of a plugin

Many readout routines wouldn't co-exist to begin with, no monolithic binary possible

```
$ daq_status -ll
  Stopped
  Logging disabled
  Filerule:     rcdaq-%08d-%04d.evt
  Buffer Sizes:     32832 KB adaptive buffering: 15 s
  Web control Port:  8899
  Elog: not defined
  -- defined Run Types:   (none)
   No Plugins loaded
```

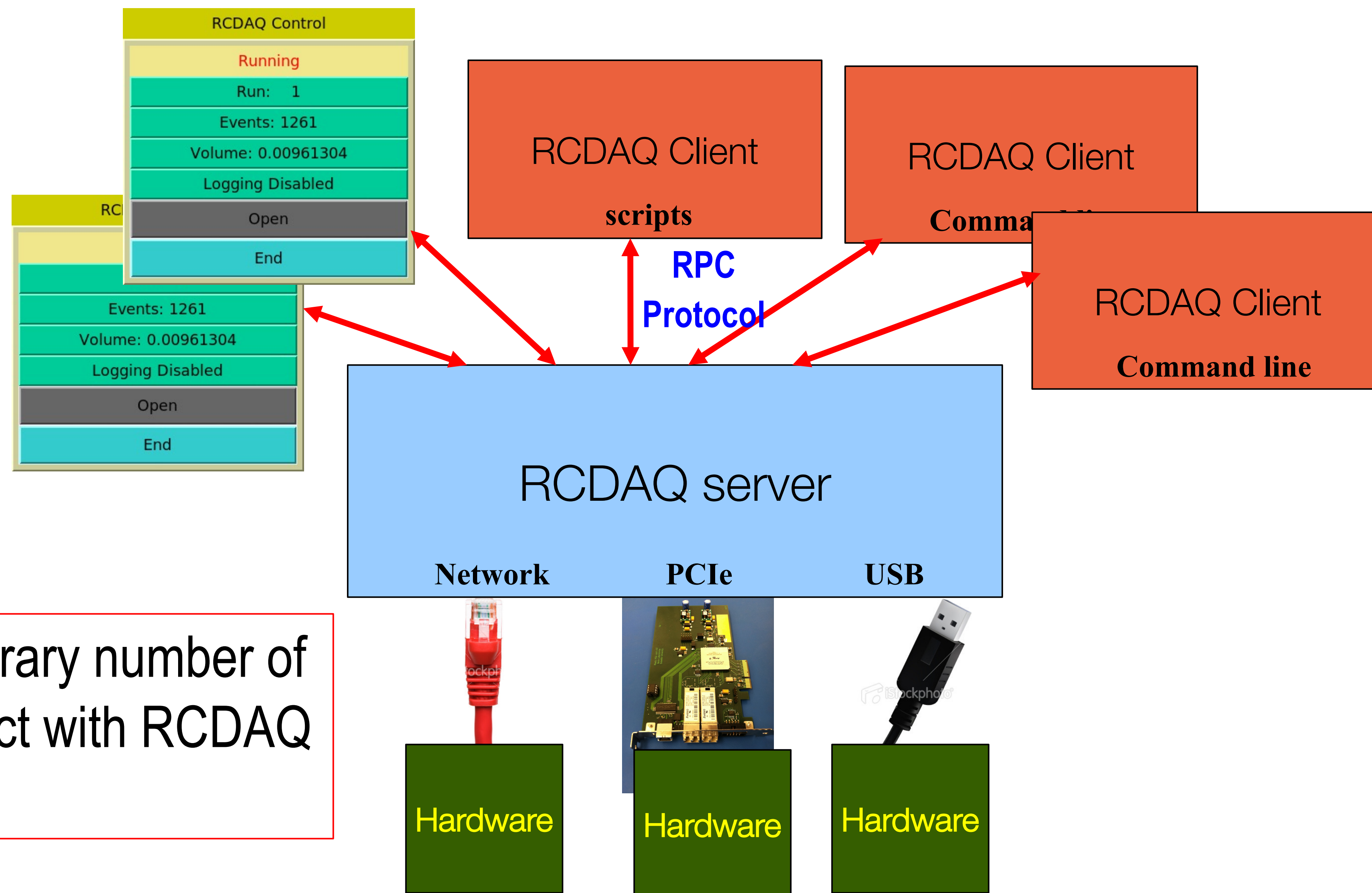
This makes RCDAQ modular and also distributable

Several plugins contain licensed (all FELIX firmware) or EULA-restricted code (CAEN), etc

But we can make RCDAQ and all other plugins available!

```
$ rcdaq_client load librcdaqplugin_intt.so

$ daq_status -ll
  Stopped
  Logging disabled
  Filerule:     rcdaq-%08d-%04d.evt
  Buffer Sizes:     32832 KB adaptive buffering: 15 s
  Web control Port:  8899
  Elog: not defined
  -- defined Run Types:   (none)
  List of loaded Plugins:
   - INTT Plugin, provides -
   -     device_intt (evttype, subid [, npackets, trigger] ) - INTT FELIX Board
```

The RCDAQ client-server concept

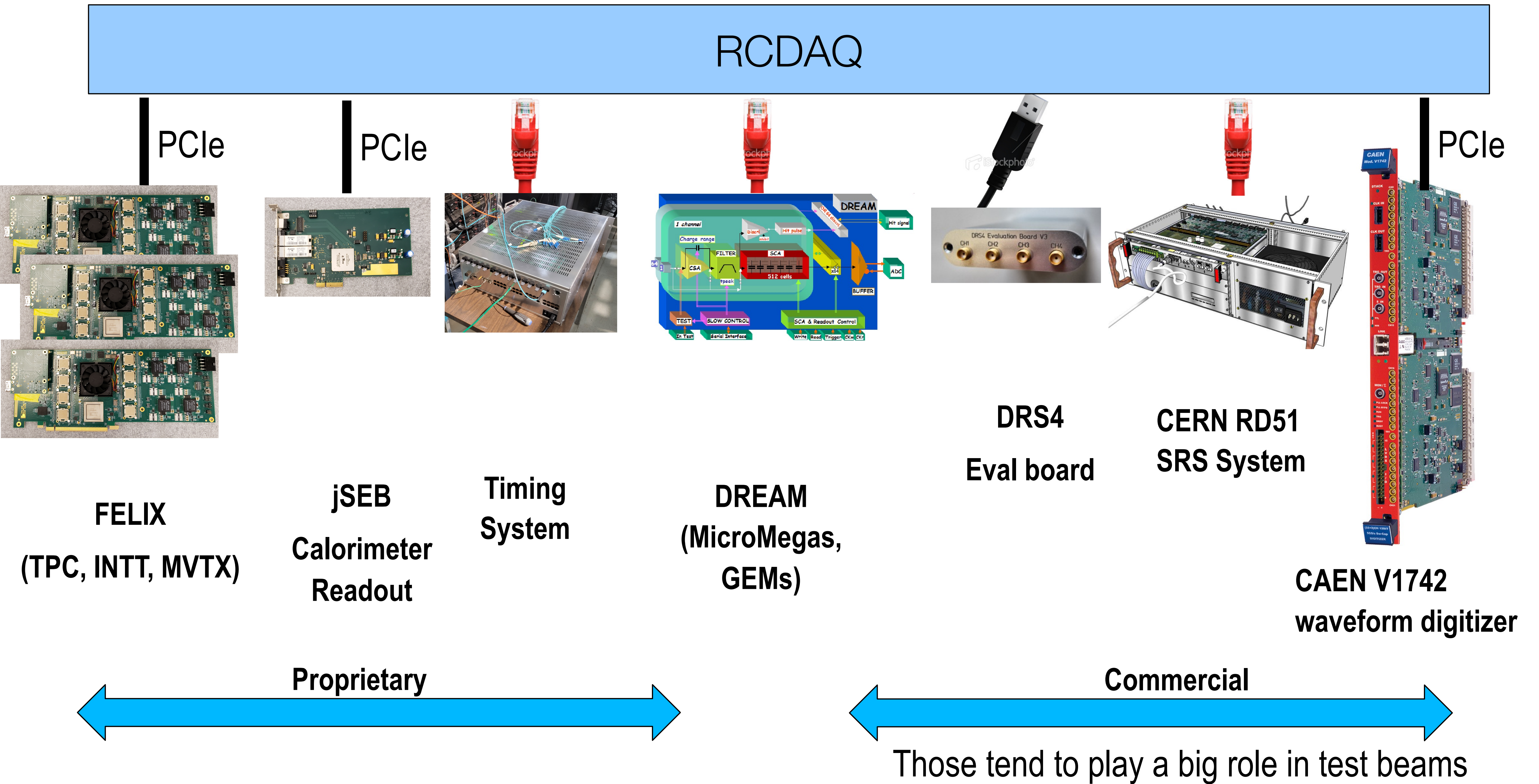


This allows an arbitrary number of processes to interact with RCDAQ concurrently

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

“The 3 main pathways into a PC”

A selection of devices implemented in RCDAQ

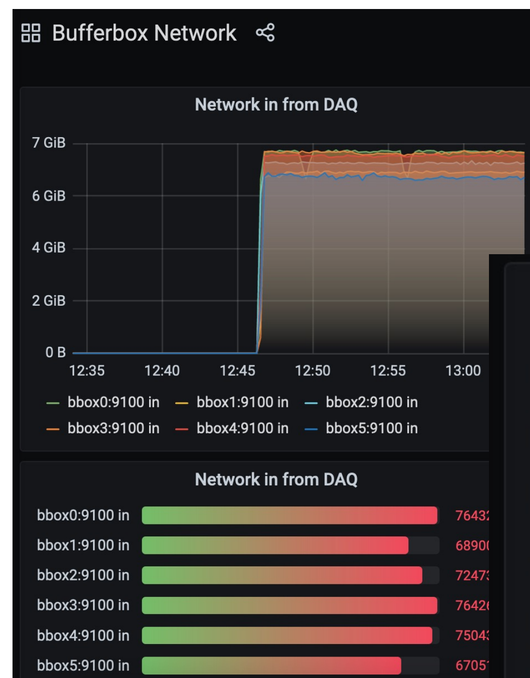


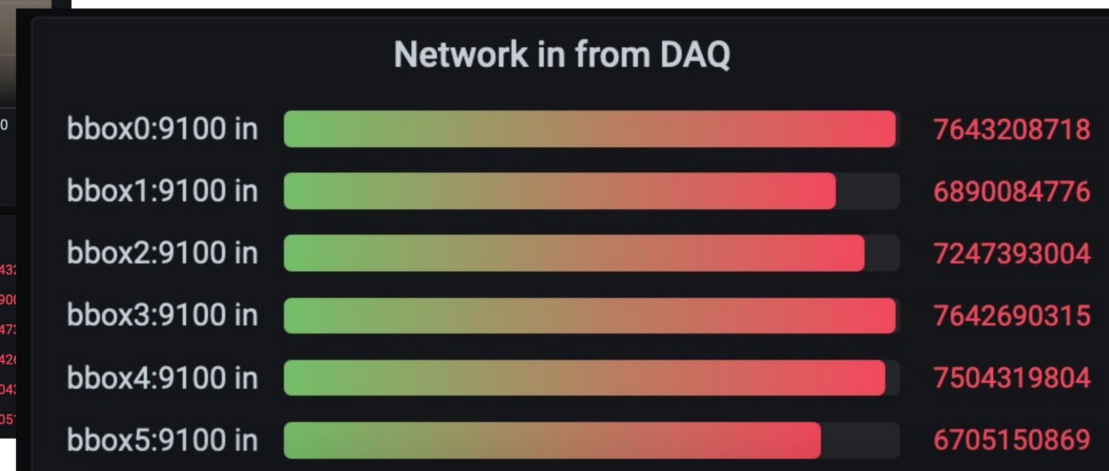
Those tend to play a big role in test beams

There are many more not shown…

Where does RCDAC run?

- Pretty much any Linux flavor. RHEL, CentOS, Debian, Ubuntu, Arch, … and Raspian!
- Yes, it can run on a Raspberry Pi (no PCIe devices, of course)
- Not really just a gimmick, we have run weeks-long cosmics measurements where you don’t want to tie up a higher-end PC, and you get like 250Hz event rate with the DRS4
- Runs on PCs, laptop, and of course what sPHENIX has right now in Bldg 1008…





That’s more than we need in EPIC’s year-3

Different Event Types for Meta- and other data

Jeff last week:

- How do we incorporate Slow Controls/Collider information with the DAQ data?

- Time Granularity of data files
  - Standard Frames
  - Detector by Detector Frames
  - BX

- "Scalers"

Table 1: The currently defined event types.

Event type	meaning	comment
1	Data Event	Readout of detector hardware
2	Streaming Data Event	Streaming Readout of detector hardware
3 . . . 7	Data Events	reserved for future use
8	Spill-On Event	
9	Begin-Run Event	Automatically generated
12	End-Run Event	Automatically generated
14	Scaler Event	Scaler information
15	Lumi Event	Denotes the start of a new Luminosity Block
16	Spill-Off Event	

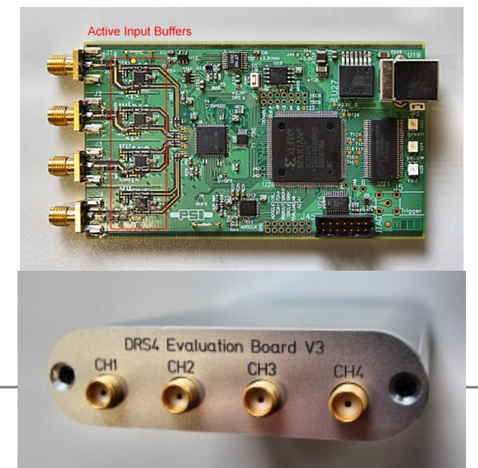
I'll concentrate on this one

Capturing your environment

- The most-often used event is the begin-run event
- Automatically generated (same as end-run)
- Guaranteed to be first and last, respectively (super-useful in continuous online monitoring)
- Each event type has its own “list of devices to read”
- data events typically read your detectors, begin-run or other types different things
- (some info in the backup, gets too long here)

Let me show you the application of beg-run that is pretty much always used…

Setting up and 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 can, but of course you wouldn't do it like that

A Setup Script

Now you got yourself a setup script as I advertised before, call it, say,

“setup.sh”

```
#! /bin/sh

rcdaq_client load librcdaqplugin_drs.so

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

Make it executable and you can re-initialize your DAQ each time the same way

Capturing the setup script 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.

We need the name with a full path

Expanding the $0 to a full filename

The readlink expands the file to a full filename

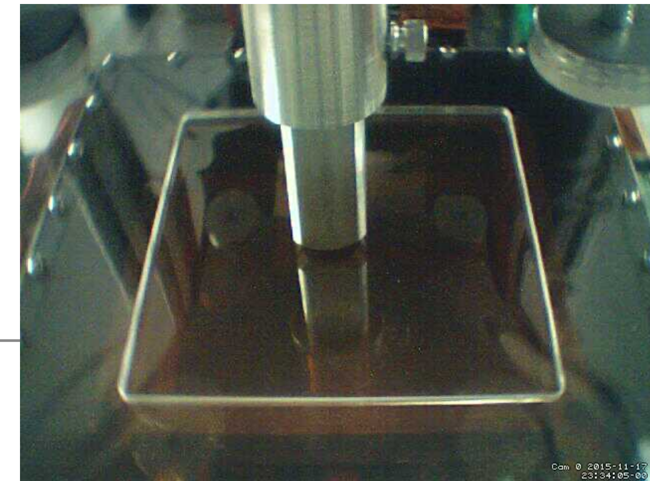
```
#! /bin/sh

MYSELF=$(readlink –f $0)

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
```

In the same spirit, you can capture anything else as well

Some “pseudo devices”: device_file, device_filenumbers, device_file_delete, device_filenumbers_delete, device_command

This one is super-powerful

Example: Characterizing GEM gain uniformity



GEM under a moving X-ray gun with step motors

Outer script lays a 20x20 or so grid as I've already shown

In begin-run we add a "device_command" that executes another script

That reaches out to the step motors, gets the x/y positions into a file like this

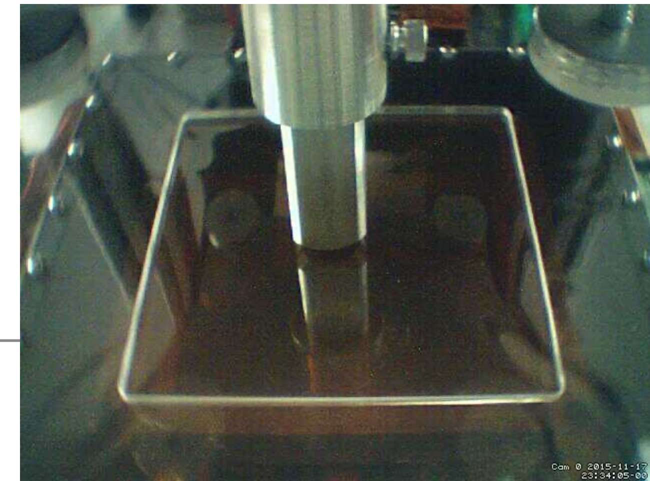
```
8031
8377
```

device_file captures as ASCII text, device_filenumbers binary like ADC values (easy programmatic access)

We also capture a webcam image for feel-good value (does it move right when it should?)

```
#add the position information

rcdaq_client create_device device_command 9 0 /home/eic/struck/getmotorpositions.sh

rcdaq_client create_device device_file 9 910 /home/eic/struck/positions.txt

rcdaq_client create_device device_filenumbers_delete 9 911 /home/eic/struck/positions.txt

# add the camera picture

rcdaq_client create_device device_command 9 0 "/home/eic/capture_picture.sh
   /home/eic/struck/cam_picture.jpg"

rcdaq_client create_device device_file_delete 9 940 /home/eic/struck/cam_picture.jpg
```

Super-easy analysis!



GEM under a moving X-ray gun with step motors

Your throw all 400 files at your analysis

In each data file, the analysis recognizes the begin-run, extracts x, y

Goes through the actual data, determines “gain”

Hits end-run, fills “gain” in the right x/y slot (like a 2D-histogram)

Next file until done. Have a coffee, come back and enjoy the result.

No database access needed! All info contained in the data files.

```
eicdaq2 ~ $ ddump -p 910 -t 9  ZZ48_0000001600-0000.evt
8031
8377 <-----
eicdaq2 ~ $ ddump -p 910 -t 9  ZZ48_0000001601-0000.evt
8031
8393 <-----
eicdaq2 ~ $ ddump -p 910 -t 9  ZZ48_0000001602-0000.evt
8031
8409 <-----
eicdaq2 ~ $ ddump -p 910 -t 9  ZZ48_0000001603-0000.evt
8031
8425 <-----
```

We are scanning in y direction here

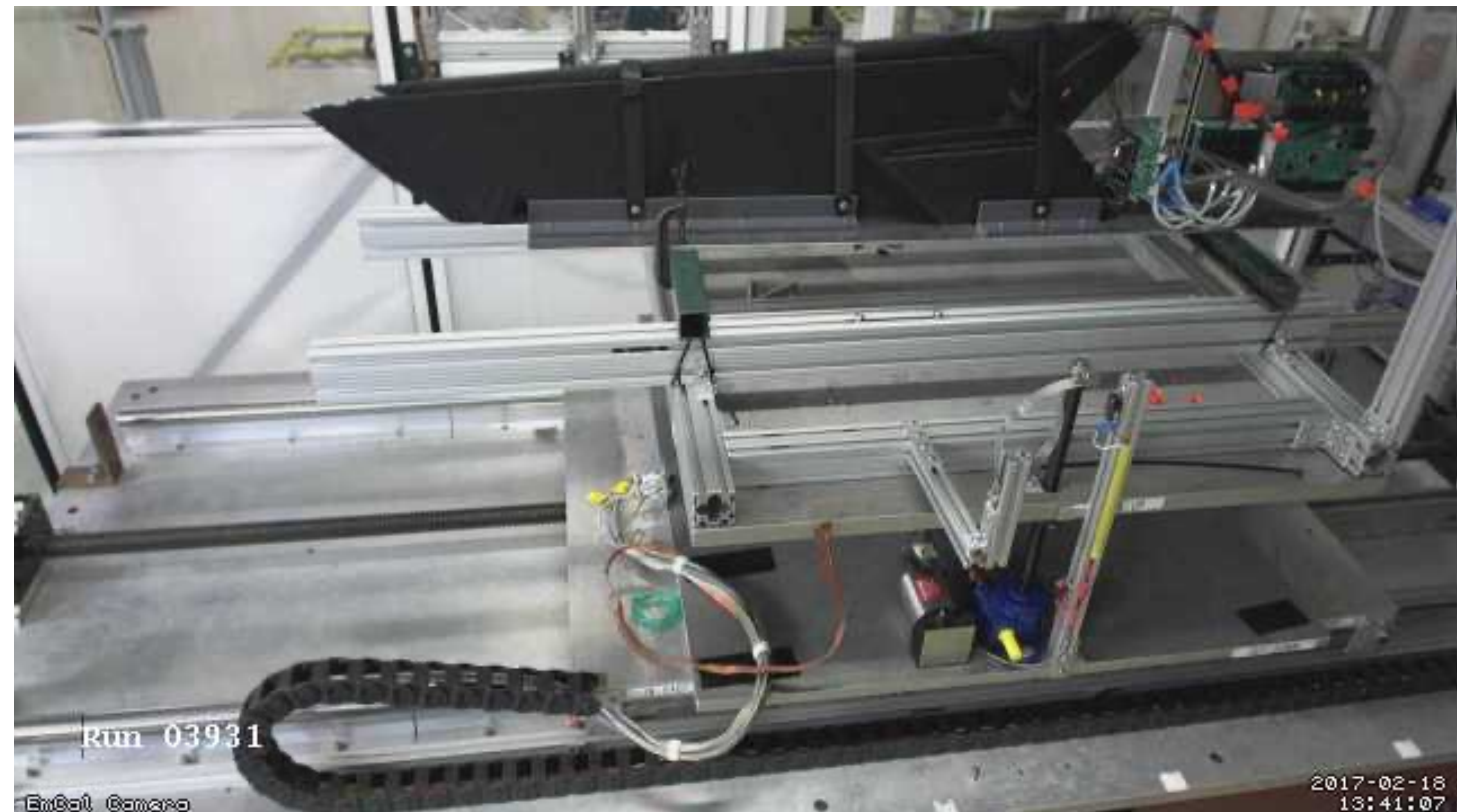
Autopilot example: “Tile Mapping” at the Fermi Test Beam Facility

“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 a script that drives to predetermined positions

Same deal as before, positions in the data files, analysis is a snap



Let's look at the recent LAPPD FTBF test setup

If in 3 year's time we don't remember how Alexander and I set this up, here we can find out

(ok, not with this font size – here are some excerpts:)

```
$ ddump -p900  -t9 junk_lappd-00023020-0000.evt
#! /bin/bash

. . .
# get the beam and other parameters (this makes beam.txt and beam_values.txt)
rcdaq_client create_device device_command 9 0 "$HOME/mlp/get_beamparameters.sh"
rcdaq_client create_device device_file_delete 9 910 "$ANC_PACKETS/beam.txt"
rcdaq_client create_device device_filenumbers_delete 9 911 "$ANC_PACKETS/beam_values.txt"
```

```
NODE=0
for LINK in {0..6} ; do
let PACKET=2000+10*$LINK+$NODE
  if [ $LINK -eq 0  ] ; then
    # Found last board/link, which should have TRIGGER set to 1
      echo rcdaq_client create_device device_CAENdrs 1 $PACKET $LINK $NODE 1
           rcdaq_client create_device device_CAENdrs 1 $PACKET $LINK $NODE 1
  else
      echo rcdaq_client create_device device_CAENdrs 1 $PACKET $LINK $NODE
           rcdaq_client create_device device_CAENdrs 1 $PACKET $LINK $NODE
  fi
```

Overall we read out 10 CAEN V1742's, 1 DRS4, 1 DREAM

```
$ ddump -p900  -t9 /gpfs02/eic/TEST.RUNS/2022-FNAL/junk/junk_lappd-00023020-0000.evt
#! /bin/bash

B=`basename "$0"`

MYSELF=$(readlink -f $0)
HERE=$(dirname "$MYSELF")

# we figure out if a server is already running
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

    # FIXME: uncomment once elog is functional;
    ELOG=$(which elog 2>/dev/null)
    [ -n "$ELOG" ]  && rcdaq_client elog 192.168.60.1 7815 LAPPD
fi

# FIXME: move this file to /home/eic/FERMILAB or /home/eic/FERMILAB/mpgd4eic/fnal_ops_2022 - done mlp
rcdaq_client daq_setrunnumberfile $OPS/.runnumber.txt

# and the run types
rcdaq_client daq_define_runtype beam     /data/eic/fnal/beam/beam_lappd-%08d-%04d.evt
rcdaq_client daq_define_runtype calib    /data/eic/fnal/calibration/calibration_lappd-%08d-%04d.evt
rcdaq_client daq_define_runtype junk     /data/eic/fnal/junk/junk_lappd-%08d-%04d.evt

# preset to nowhere
#rcdaq_client daq_set_runtype nowhere
rcdaq_client daq_set_runtype junk

# clears the list of devices to read out
rcdaq_client daq_clear_readlist

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

# get the beam and other parameters (this makes beam.txt and beam_values.txt)
rcdaq_client create_device device_command 9 0 "$HOME/mlp/get_beamparameters.sh"

. . .

  # create CAEN devices (1 $PACKET $LINK $TRIGGER)

  # FIXME: this is not clean (as the first entry in LINK {} can be a non-zero number);
  if [ $LINK -eq 0  ] ; then

. . .

# FIXME: 'junk' may need to be adjusted for running;
#rcdaq_client daq_set_runtype junk
#rcdaq_client daq_set_maxevents 1000000
#rcdaq_client daq_open

exit
```

Forensics

We tend to capture "everything" we can, like a plane's black box… Example:

"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
```

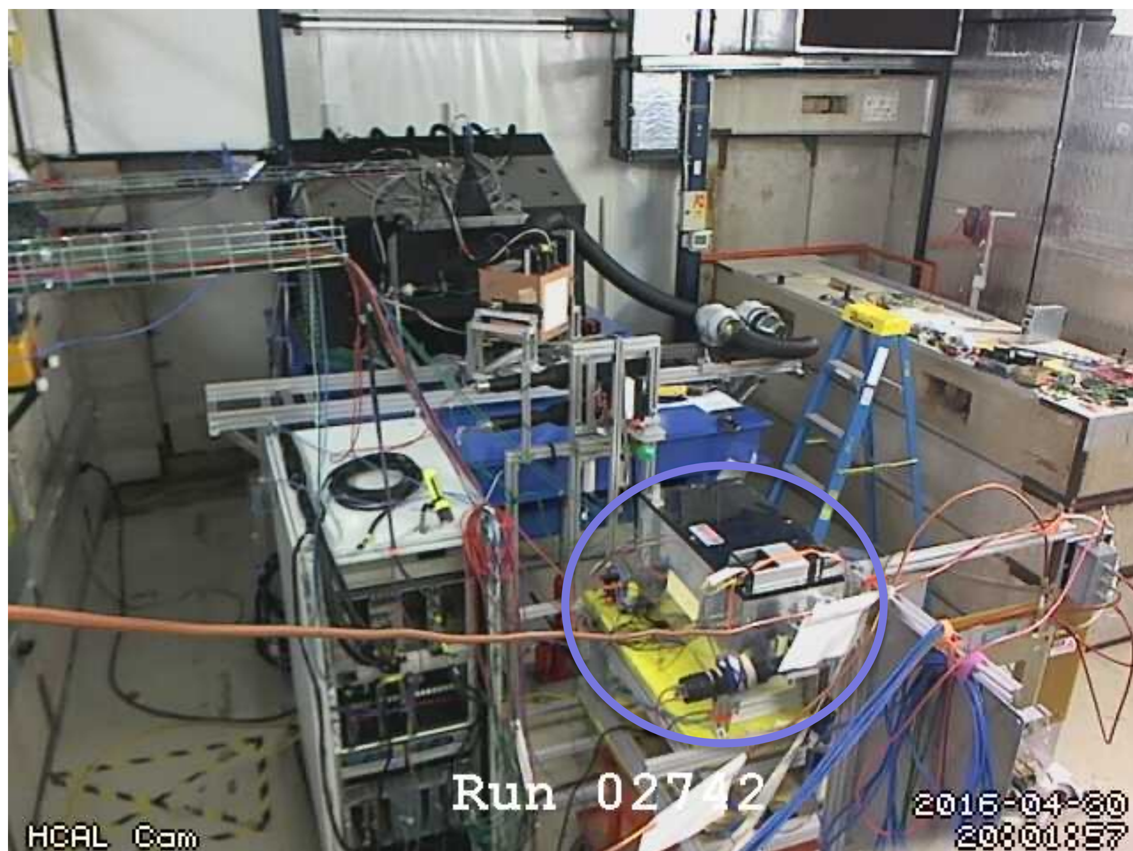
```
$ ddump -t 9 -p 923 beam_00002268-0000.prdf
S:MTNRG   = -2      GeV
F:MT6SC1 =  11846      Cnts
F:MT6SC2 =  7069       Cnts
F:MT6SC3 =  3883       Cnts
F:MT6SC4 =  0          Cnts
F:MT6SC5 =  283048     Cnts
E:2CH     =  1058   mm
E:2CV     =  133    mm
E:2CMT6T =  74.13  F
E:2CMT6H =  37.26  %Hum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

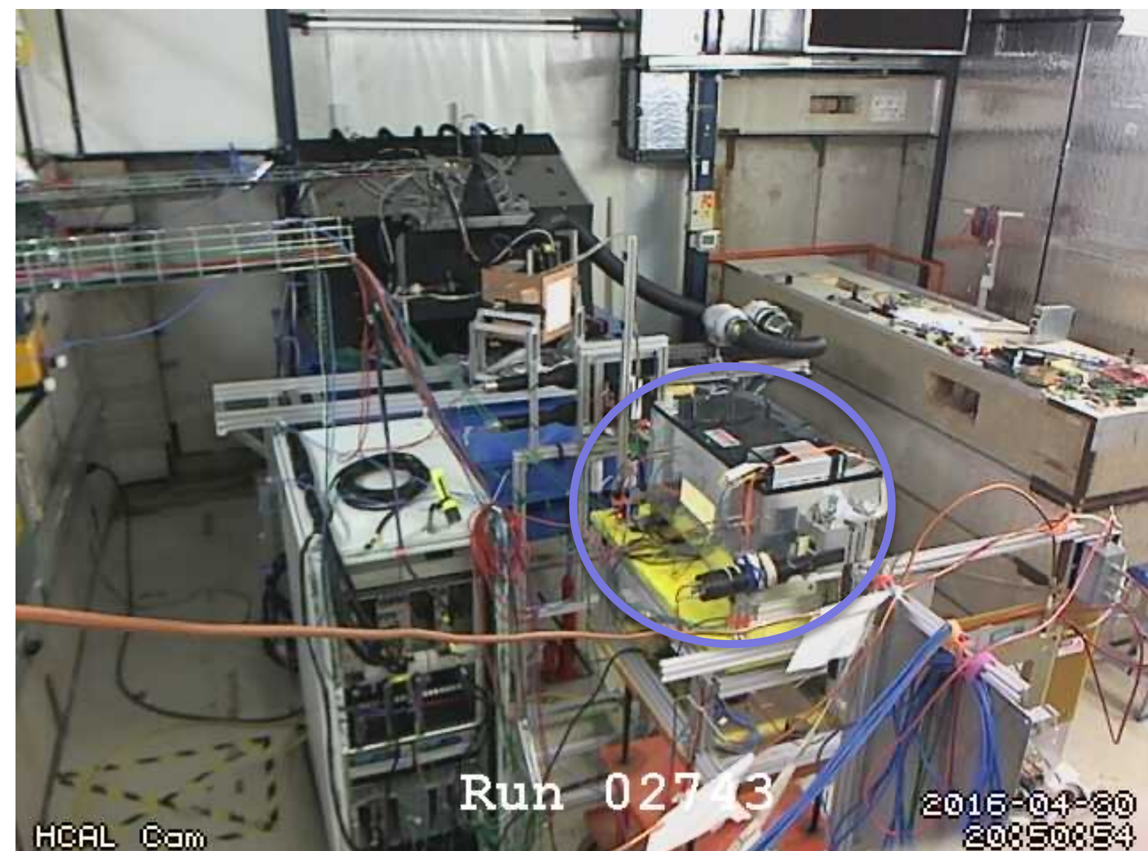
More Forensics (my poster child why this is so useful…)

“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
```





“Meta Data” Packet list from a recent 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

Summary

Rock-solid, scalable, versatile DAQ system

Large user base in the EIC community and outside

As well as many “power users” – Kiselev, Azmoun, Stoll, Hemmick, Hohlmann, Tsai, …

Superb analysis and online monitoring support (another time)

Tons of operational experience in sPHENIX, test beams (permanent setup @ the FTBF), dozens of lab setups

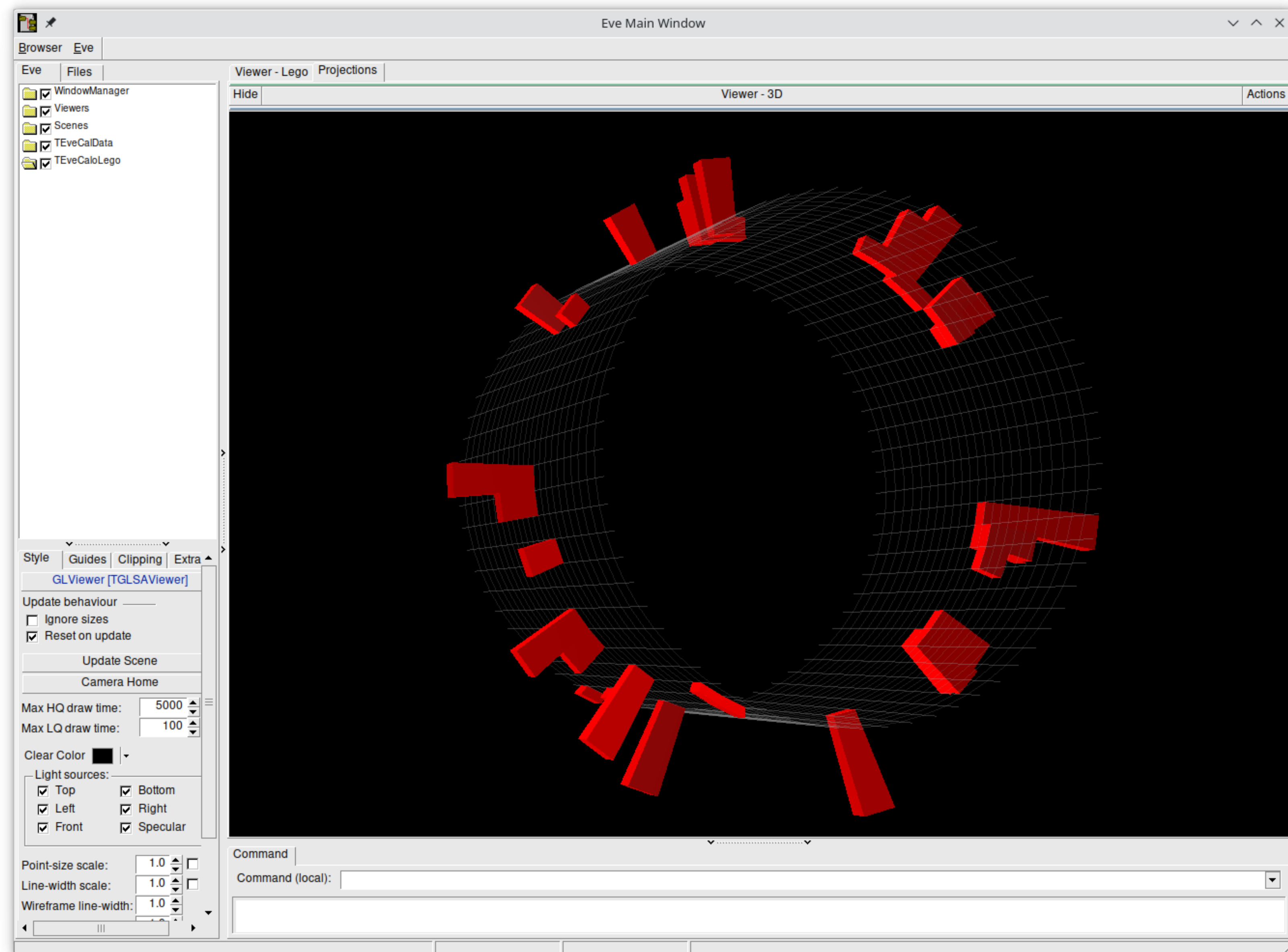
Hard to overstate to importance of having the DAQ + format in hand from day-1 - student training, experience

Early test beam online monitoring and analysis code often grows into the eventual reco code

That’s all for today

Plenty more EPIC-relevant things to talk about maybe at a future meeting

- Current benchmarks
- Timing system integration
- Analysis support / APIs
- Getting Beam Crossing info
- Data inspection tools
- Online monitoring
- CD-2/3a Experience



Thank you!

(I leave you here with a Single-Event Display of an EPIC outer HCal event, real data, cosmics)