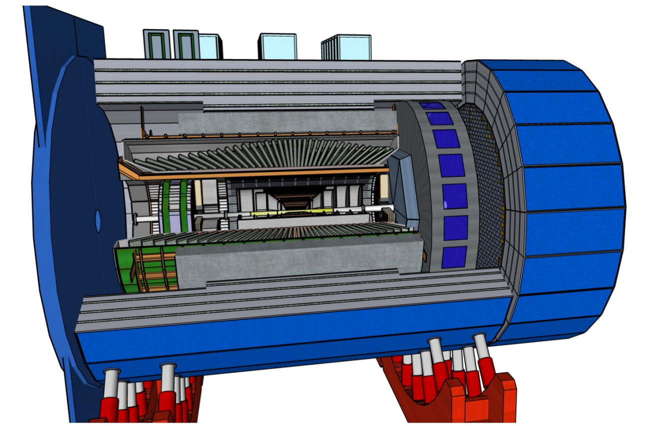


The DAQ and Online System for the ECCE Proposal at EIC

Martin L. Purschke

Manhattan

Long Island, NY

RHIC/EIC from space

The ECCE DAQ

Conveners: Chris Cuevas (Jlab) and Martin Purschke (BNL)

Chris and I have been the conveners during the proposal phase.

Chris continues, and Jo Schambach (ORNL) has taken over from me (I got fired ☺)

(My real day job is the sPHENIX DAQ manager)

The ECCE DAQ and Timing system is heavily influenced by the corresponding sPHENIX systems

Some personnel overlap, but also

- Key concepts (Streaming Readout, the use of ASICs, use of a “DAM” (FELIX in 2022)) are very similar
- Low-jitter clock distribution to a FELIX successor is a key ingredient
- Concept is designed with a distributed calibration/reconstruction paradigm (Grid) in mind
- It’s scalable, and has well-defined hand-off points where common technologies (transports, storage, monitoring and other APIs) take over

Today

- Detectors to be read out
- The DAQ plan
- Brief overview of what we have in terms of DAQ
- A few notes about SRO (this is the SRO workshop, after all)
- What we have done so far with existing DAQ technology for our R&D
- A few slides about the concept behind RCDAQ
- A feature list what we can do already today and during the R&D phase

From the Proposal

The current sPHENIX carriage with the outer HCal and the magnet

ECCE

oHCAL

iHCAL

cryostat

μRwell

Backward discs

Forward discs

μRwell

hpDIRC

EEMC

mRICH

AC-LGAD

Si barrel

AC-LGAD

AC-LGAD

BEMC

dRICH

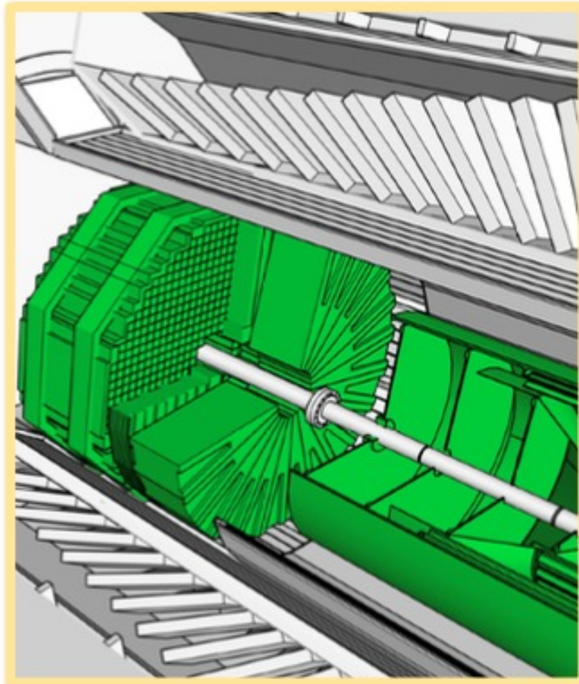
FEMC

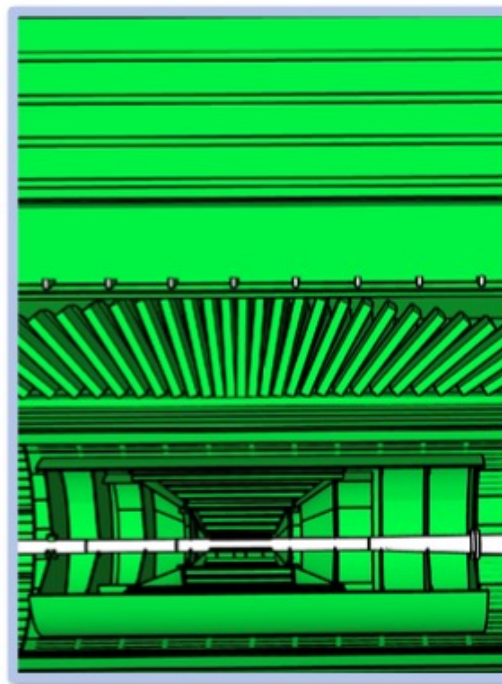
LFHCAL

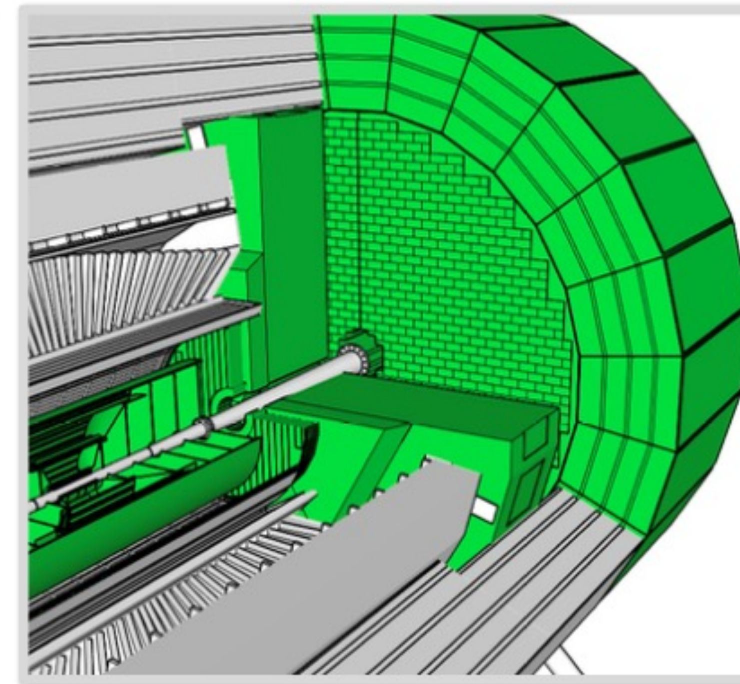
Hadron Beam

Electron Beam

From the Proposal







Backward Endcap

Tracking:
- ITS3 MAPS Si discs (x4)
- AC-LGAD

PID:
- mRICH
- AC-LGAD TOF
- $PbWO_4$ EM Calorimeter (EEMC)

Barrel

Tracking:
- ITS3 MAPS Si (vertex x3; sagitta x2)
- μRWell outer layer (x2)
- AC-LGAD (before hpDIRC)
- μRWell (after hpDIRC)

h-PID:
- AC-LGAD TOF
- hpDIRC

Electron ID:
- SciGlass EM Cal (BEMC)

Hadron calorimetry:
- Outer Fe/Sc Calorimeter (oHCAL)
- Instrumented frame (iHCAL)

Forward Endcap

Tracking:
- ITS3 MAPS Si discs (x5)
- AC-LGAD

PID:
- dRICH
- AC-LGAD TOF

Calorimetry:
- Pb/ScFi shashlik (FEMC)
- Longitudinally separated hadronic calorimeter (LHFCAL)

Rough Subsytem Count

- ~ 20 different detector components combined in the 3 parts (backward/forward/central barrel)
- Just a quick overview for reference here (backward/forward), read all about it in the proposal

Topic	Challenge	ECCE solution	Comment
Far-Backward – Low-Q^2 Tagger	Measure low-Q^2 photo-production with as minimal a Q^2-gap as possible.	Spectrometer with AC-LGAD tracking and $PbWO_4$ calorimetry	
Far-Backward – Luminosity Detector	e-ion collision luminosity to better than 1% and relative Luminosity for spin asymmetries to 10^{-4}	Zero Degree Calorimeter with x-ray absorber and e^+/e^- pair spectrometer with AC-LGAD tracking and $PbWO_4$ calorimetry	two complementary detection systems
Far-Forward – B0 Spectrometer	$\eta > 4$ charged particle tracking and γ measurement	Four Si trackers with 10 cm $PbWO_4$ calorimeter	
Far-Forward – Off-momentum Detectors	forward particles (Δ, Λ, Σ, etc) decay product measurement	AC-LGAD detectors	Sensors on one side detect p, on other side p^- from Λ decay; sensors outside beam pipe
Far-Forward – Roman Pots	Detect low-p_T forward-going particles	AC-LGAD detectors	fast timing (~35 ps) removes vertex smearing effects from crab rotation; 10σ from beam
Far-Forward – Zero-degree Calorimeter	Measure forward-going neutrons γ and heavy-ion fission product	FOCAL-type calorimeter with high-precision EM and Hadron Calorimetry	Upgrade option: AC-LGAD layer to capture very high rapidity charged tracks

An idea of the channel count

PID WBS Name	Detector	ASIC	Channels
Barrel PID	hpDIRC	High Density SoC	69,632
	TOF	eRD112 development	8,600,000
Electron Endcap	mRICH	High Density SoC	65,536
	TOF	eRD112 development	920,000
Hadron Endcap	dRICH	MAROC3	19,200
	TOF	eRD112 development	1,840,000
Far-Forward Detectors	Roman Pots	eRD112 development	524,288
	B0 Detector	eRD112 development	2.6M
	Off-Momentum Detectors	eRD112 development	1.8M
Far-Backward Detectors	Low-Q^2 Tagger	eRD112 development	4.6M
	Luminosity Monitor	eRD112 development	268,441

Channels
69,632
8,600,000
65,536
920,000
19,200
1,840,000
524,288
2.6M
1.8M
4.6M
268,441

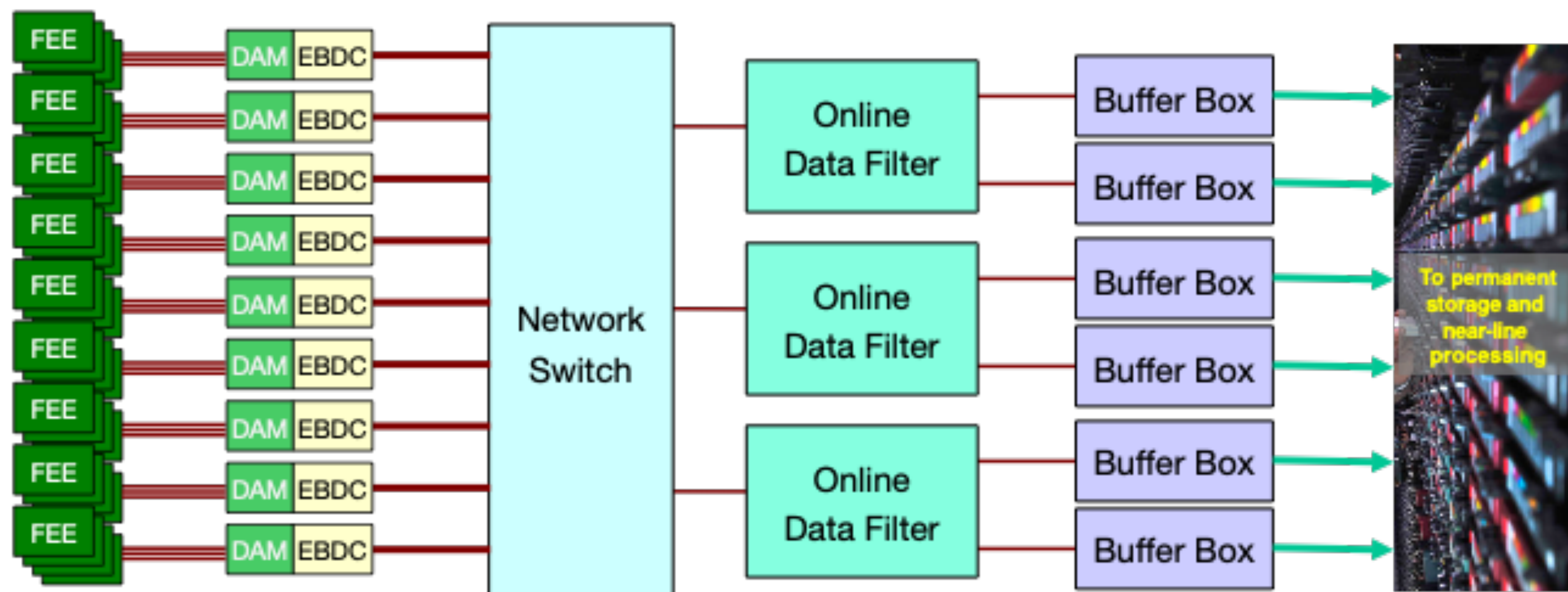
Makes about 21 million channels in round numbers

Data statistics

	ECCE Runs		
	year-1	year-2	year-3
Luminosity	$10^{33}\mathrm{cm}^{-2}\mathrm{s}^{-1}$	$2 \times 10^{33}\mathrm{cm}^{-2}\mathrm{s}^{-1}$	$10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$
Weeks of Running	10	20	30
Operational efficiency	40%	50%	60%
Disk (temporary)	1.2 PB	3.0 PB	18.1 PB
Disk (permanent)	0.4 PB	2.4 PB	20.6 PB
Data Rate to Storage	6.7 Gbps	16.7 Gbps	100 Gbps
Raw Data Storage (no duplicates)	4 PB	20 PB	181 PB
Recon process time/core	5.4 s/ev	5.4 s/ev	5.4 s/ev
Streaming-unpacked event size	33kB	33kB	33kB
Number of events produced	121 billion	605 billion	5,443 billion
Recon Storage	0.4 PB	2 PB	18 PB
CPU-core hours (recon+calib)	191M core-hours	953M core-hours	8,573M core-hours
2020-cores needed to process in 30 weeks	38k	189k	1,701k

To put the red box into context – sPHENIX will write 1.5PB/day, 9PB a week – compare to 6PB/week here, in 2033 or so

DAQ Bird’s eye view

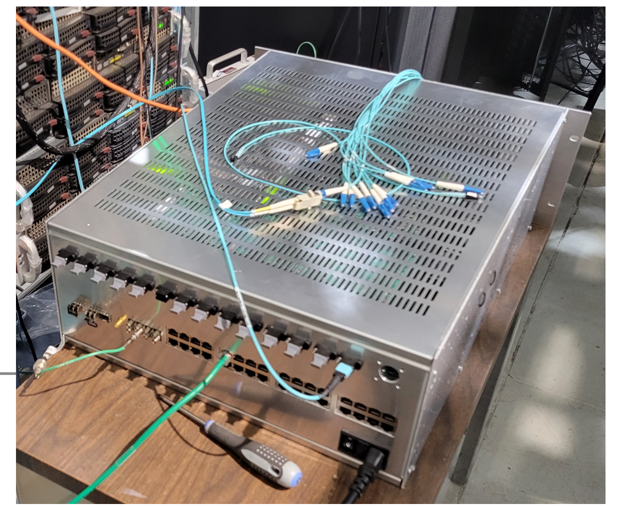


Only a few elements shown

FEEs vary a lot, complexity varies a lot, data volume varies a lot

Common denominator is that there is a uniform data structure at the output of the DAM

Timing System



Pick a convenient multiple of the beam clock frequency

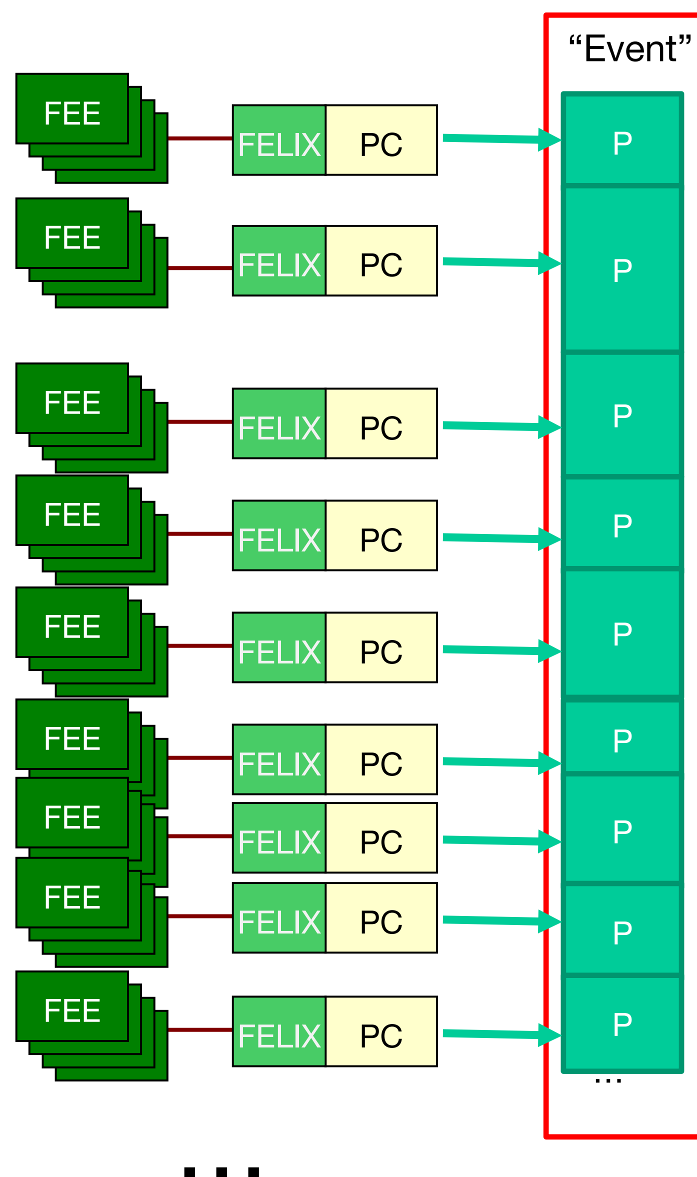
Have a global, never-reverting master BCO counter – 64 bit, transmit BCO LSBs to front-ends (40 bits)

Front-ends embed a number of those bits in their lower-level data structures (Felix - 40, FEE - 20)

The **only way** to send information to the FEE on a per-crossing basis (like, have the FEE do something different in the abort gap)

Bit Number	Function	Beam clock phases					
		0	1	2	3	4	5
7-0	Mode bits /BCO	Modebits bits 7-0	BCO bits 7-0	BCO bits 15-8	BCO bits 23-16	BCO bits 31-24	BCO bits 39-32
8	Beam clock phase0	1	0	0	0	0	0
9	LVL1 accept	X	0	0	0	0	0
10	Endat 0	X	X	X	X	X	X
11	Endat 1	X	X	X	X	X	X
12	Modebit enable	1	0	0	0	0	0
15-13	User bits	3 user bits	0	1	2	3	4

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

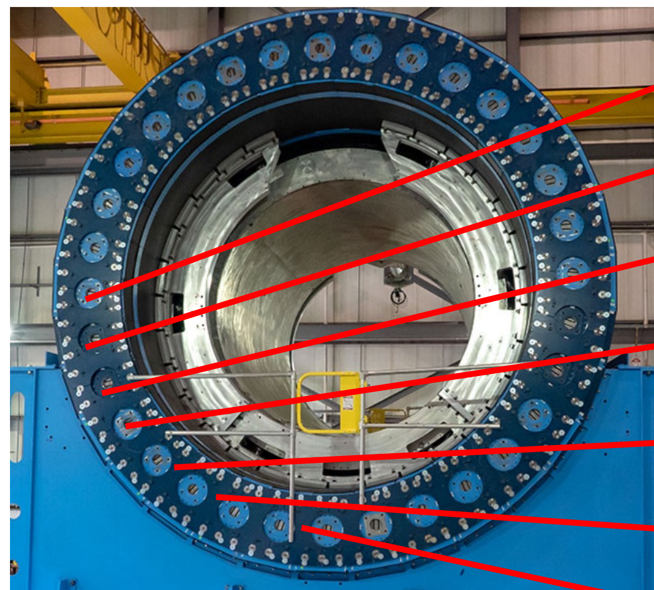
Very rough number: 250-300 packets collectively

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

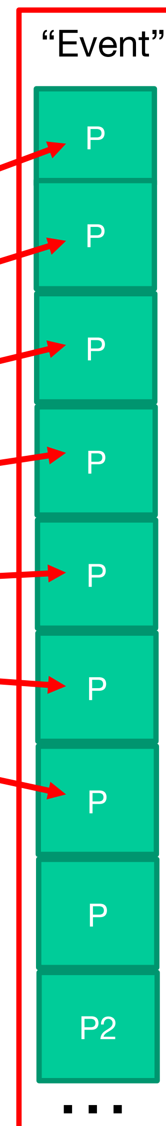
Example: Full Outer HCal Real Events

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



...



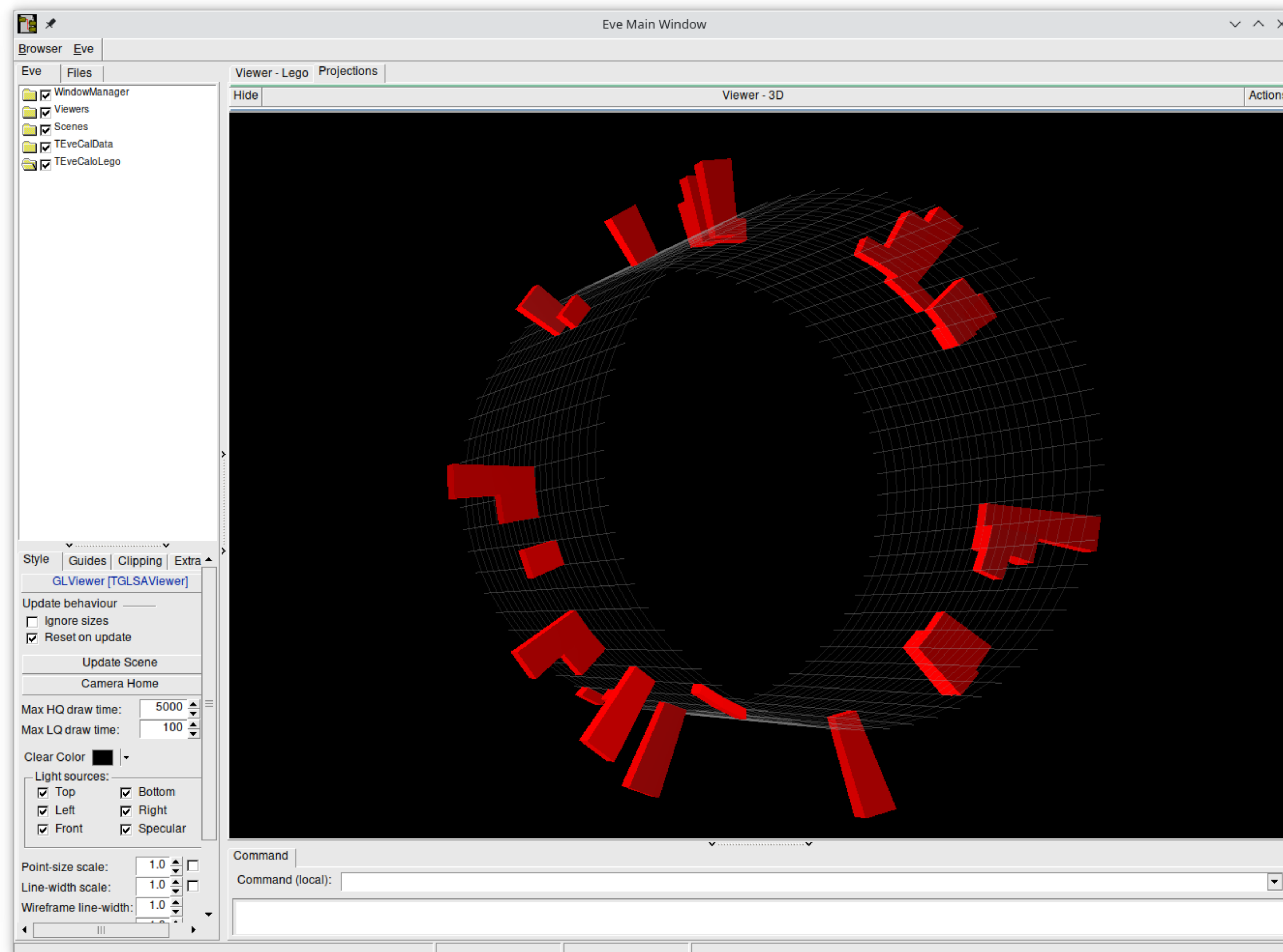
Hitformat

```
$ 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)
```

oHCal Data

As a quick exercise, I took our oHCal events and made an Event Display

- Final packet access API
- What we are doing here (with cosmics) will look (structure-wise) exactly like our data next year
- People can work with the data for calibration procedures, verify channel mappings, interface F4A with the real data structure, and on on and on
- (and yes, make Event Displays…)

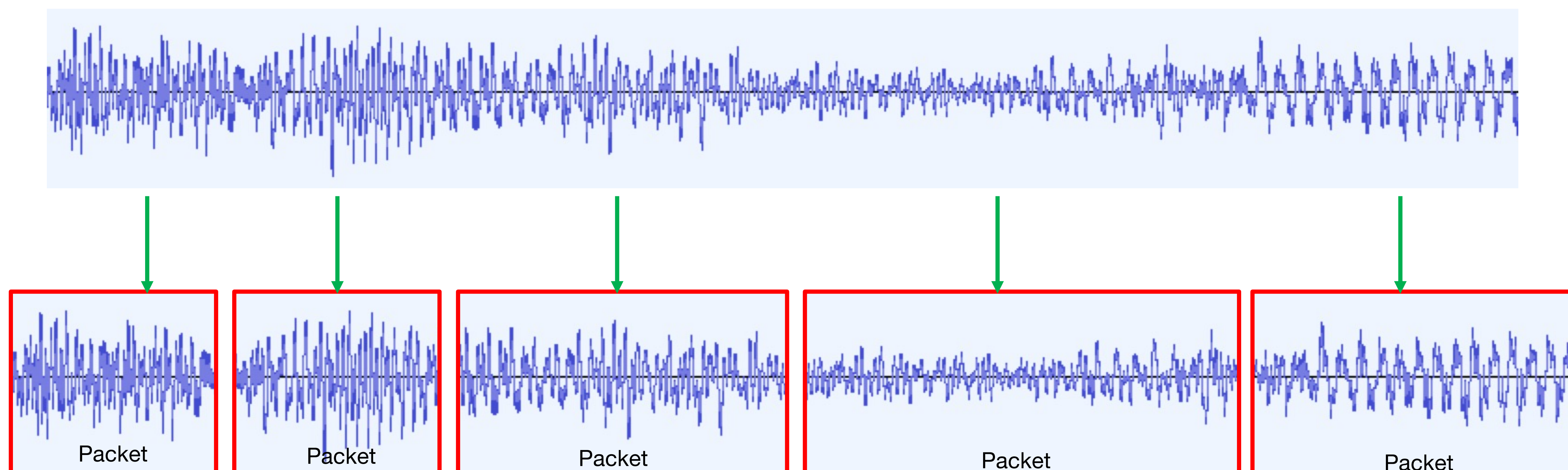


Different Event

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



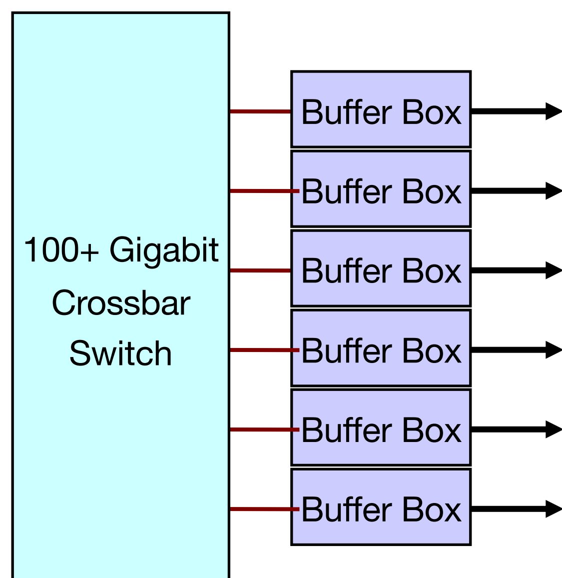
We are chopping the streaming detector data into conveniently-sized packets for storage

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

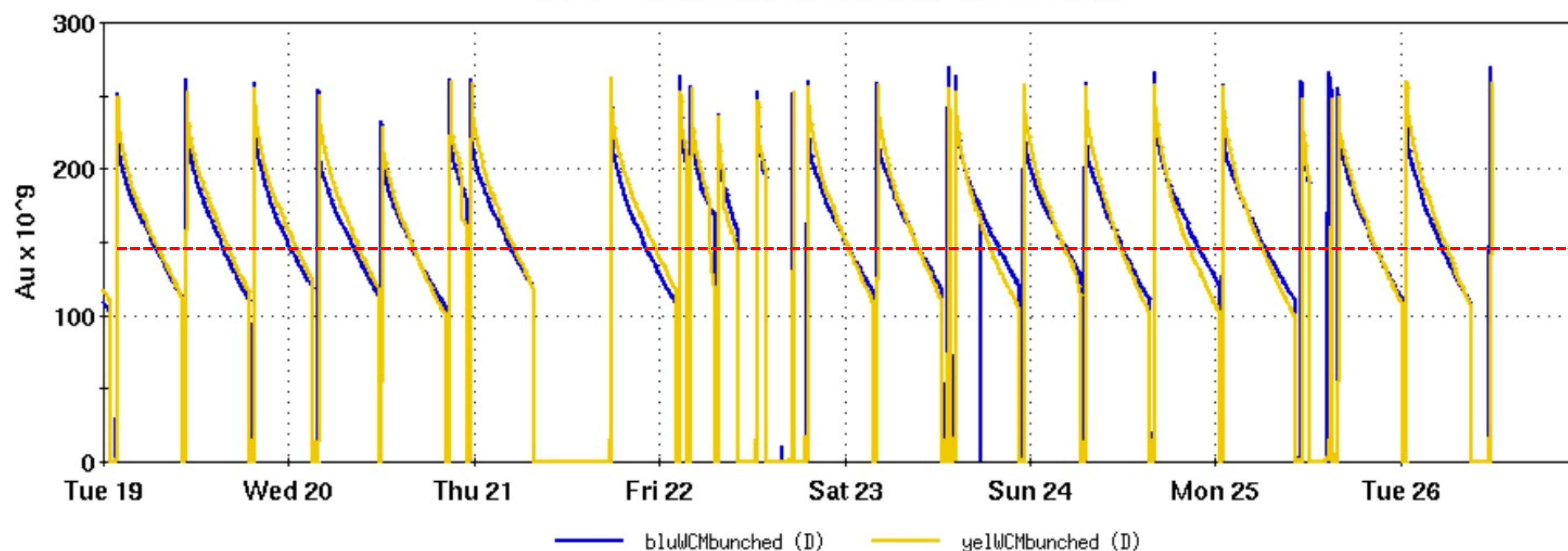
Why do we call those “BufferBoxes”?



The data rate at a collider is “bursty” – high luminosity at the begin of a RHIC store, then ”burning off” – change of a factor of 2

Also gaps in data flowing with collider dump/fill, access, APEX, MD

This Buffer boxes allows us to deal with the average, rather than the peak rate at the SDCC-facing “end”



2016 (last PHENIX run) beam intensity over a week

Average

Data Reduction/Compression

Every detector obviously wants to minimize the data volume without losing physics information

Lossy: zero-suppression (threshold), clustering w/ threshold, etc

Zero-suppression is a must to avoid clogging up the front-end pipes.

However:

We can apply loss-less compression as a catch-all to offset compromises in threshold settings

Also, the early data are not as ”dense-packed” – development/learning curve requiring actual data

Set thresholds as low as the front-end bandwidth allows, let late(r)-stage compression do the rest

For 2 decades we have always applied late-stage, distributed compression to our raw data (PHENIX/sPHENIX)

Distributed: happens at the EBDC stage

Rock solid, and even saves significant time at the reconstruction stage

(reading less data vastly over-compensates the small penalty for the decompression step)

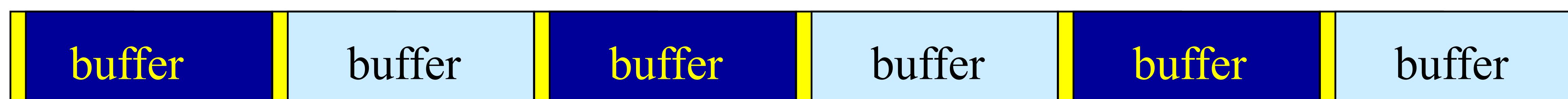
Conceptually similar to compressed root files, but different/much faster compression algorithm

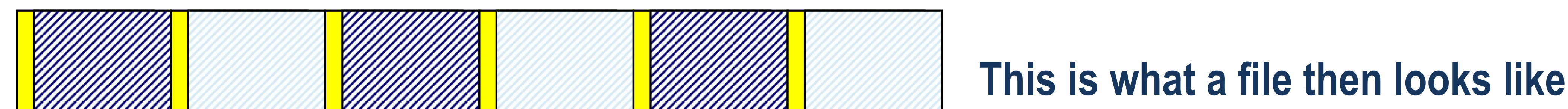
Data Compression

After all data *reduction* techniques (zero-suppression, bit-packing, etc) are applied, you typically find that your raw data are still compressible to a significant amount

Our compressed raw data format supports a late-stage data compression:

This is what a raw data file would normally look like (a buffer typically holds 10-500 events, e.g. 64MB)





All this is handled completely in the I/O layer, the higher-level routines just receive a buffer as before.

Compressed data

The current ECCE test bench/test beam/etc data that we take are super-compressible

no or super-low occupancy, not zero-suppressed, etc

compression **down to** ~5% of the original, not typical for the “real” running

```
$ prdfcheck /data/phnxrc/1008/junk/hcal_lzo_00000100_0000.evt | more
buffer at record    0 length =  201799  25 marker = ffffbbfe  LZO Marker  Or.length: 4194208  4.81137%
buffer at record   25 length =  201304  25 marker = ffffbbfe  LZO Marker  Or.length: 4194264  4.79951%
buffer at record   50 length =  201424  25 marker = ffffbbfe  LZO Marker  Or.length: 4194264  4.80237%
```

Expect a 40% value (compression by 60%) in the early going, going up to low 70%s

(One can think of this as a metric for the actual “information content per bit”)

Late-generation PHENIX raw data (2016, last run):

```
$ prdfcheck  EVENTDATA_P00-0000443135-0001.PRDFF  | more
buffer at record    0 length = 3285885  402 marker = ffffbbfe  LZO Marker  Or.length: 4357160  75.4135%
buffer at record  422 length = 3064576  375 marker = ffffbbfe  LZO Marker  Or.length: 4349976  70.4504%
buffer at record  797 length = 3204863  392 marker = ffffbbfe  LZO Marker  Or.length: 4250952  75.3917%
```

Compression speed

“dpipe” is a swiss-army-knife utility to work with raw data. Take a file, uncompress, re-compress, manipulate, etc etc. The file here contains 77524 events.

```
$ time dpipe -sf -df -l EVENTDATA_P00-0000443135-0001.PRDFF EVENTDATA_P00_LZO-0000443135-0001.PRDFF

real     0m2.866s
user     0m2.354s
sys      0m0.507s
```

Asks for LZO-compression

```
$ bc -lq
2.866 / 77524 * 10^6
36.96919663588050000000

77524 / 2.866
27049.54640614096301465457
```

So 37μs per event and 3.5GBytes/s compression rate per thread

BTW – I keep a gzip-compression format around as a benchmark – this shows just how much faster LZO is compared to gzip, for only a 10% additional improvement

```
$ time dpipe -sf -df -z EVENTDATA_P00-0000443135-0001.PRDFF EVENTDATA_P00_gz_-0000443135-0001.PRDFF

real     6m58.935s
user     6m55.183s
sys      0m3.659s

$ ls -l   /mnt/ramdisk/*
-rwxr--r-- 1 phnxrc phnxrc 10739654656 May  9 08:15 /mnt/ramdisk/EVENTDATA_P00_LZO-0000443135-0001.PRDFF
-rwxr--r-- 1 phnxrc phnxrc  9161973760 May  9 08:22 /mnt/ramdisk/EVENTDATA_P00_gz_-0000443135-0001.PRDFF
```

Asks for gzip-compression

A bit about my personal beliefs w.r.t. DAQ…

- During R&D/Tests beams/etc you should have a DAQ that is on the evolutionary path towards the eventual system
- You want to be able to re-use (of course, still refine) the code and experience you have gained with your detector in test beams and lab tests
- This code very often evolves into the early online monitoring code and later into the reco code

Case in point with sPHENIX:

- We have been using RCDAQ for all measurements, test beams, detector calibrations, muon measurements, burn-in’s, etc etc
- Same access APIs, code base, applied algorithms
- Junior folk gain experience with what becomes the “real thing”

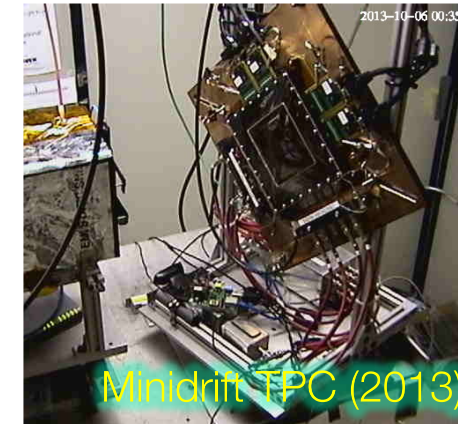
Why has the sPHENIX DAQ been important for the EIC and ECCE R&D?

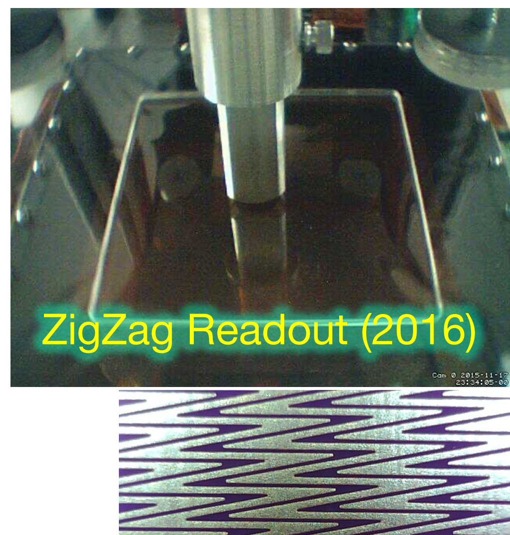
sPHENIX is one of the experiments paving the way for streaming readout in our community

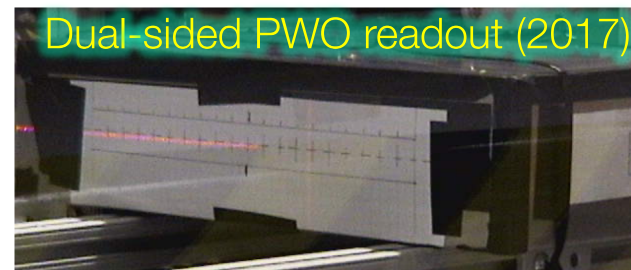
sPHENIX's RCDAQ system has been a pillar of EIC-themed data taking for R&D, test beams etc since 2013 – eRD1, eRD6, LDRDs, …

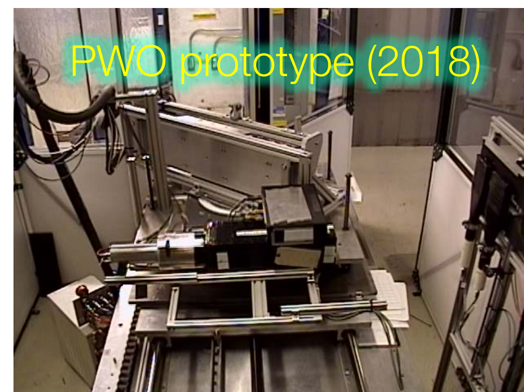
Estimated 25 active RCDAQ installations in the EIC orbit + ~30 elsewhere

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

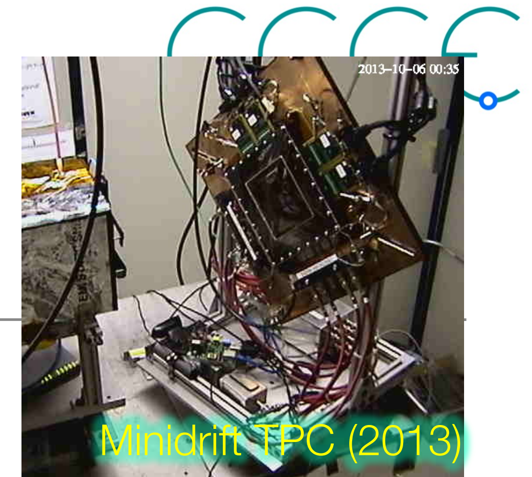








My personal first EIC R&D campaign (Fermilab test beam, 2013)

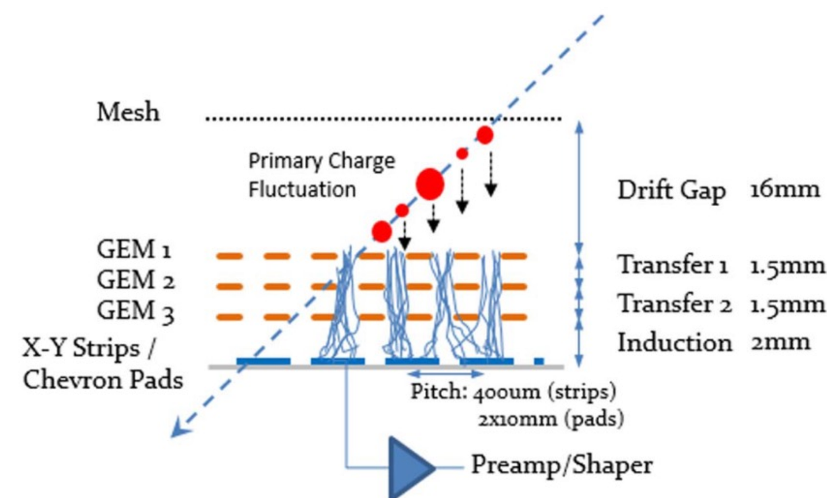


1768 IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 63, NO. 3, JUNE 2016

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**



You can guess what DAQ system we were using…

We can analyze the data today exactly as we did then

Great training ground for students

Many more more EIC-themed R&D efforts since then, all using RCDAQ

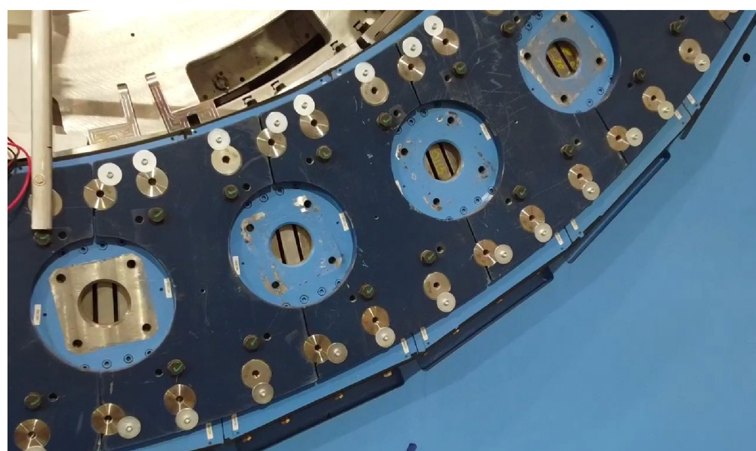
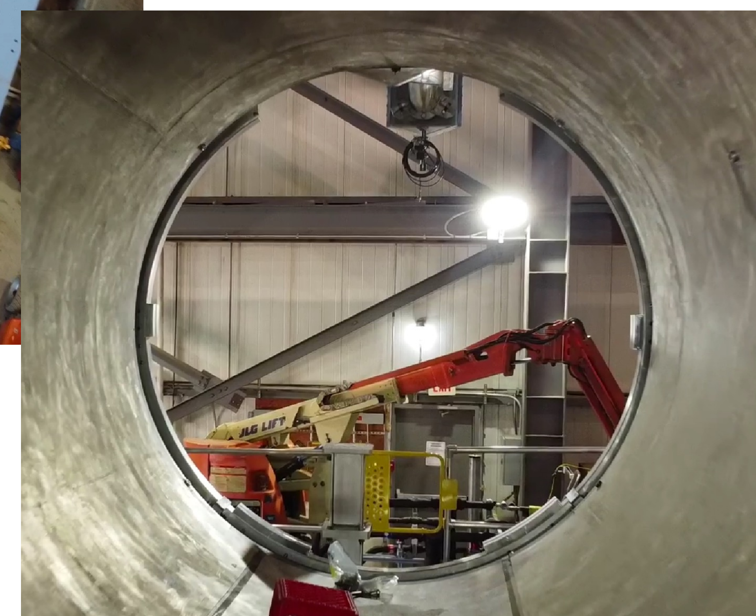
ECCE

Fly through the Detector!

We had the opportunity to fly a drone through the current installation with the outer Hcal and the magnet…

Take a flight through the detector! Go to

https://www.phenix.bnl.gov/~purschke/Drone/cut01.mp4

Summary

Solid proposal that revolves around the concept of a Data Aggregation Module (DAM) and the existing and rock-solid RCDAQ system

Today: DAM==FELIX (that cannot be built any longer)

Several projects to bring the “next FELIX” into the next decade under way or on the horizon

A modest amount of new ASICs for the front-ends (didn’t have time to talk about that)

Envisioned data rates/volumes manageable even by today’s standards

(off the cuff: that usually leads to great new ideas what to do with that bandwidth!)

Lots of support available for R&D-level DAQs (old eRDxes, 1,6,23,…) and new eRD108, eRD110, …

Backup

Streaming or not, here we come!

What I mean here is that past the FEE, the readout is completely oblivious to the readout out mode – it doesn't care how the front-end arrived at the decision to send up the data.

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

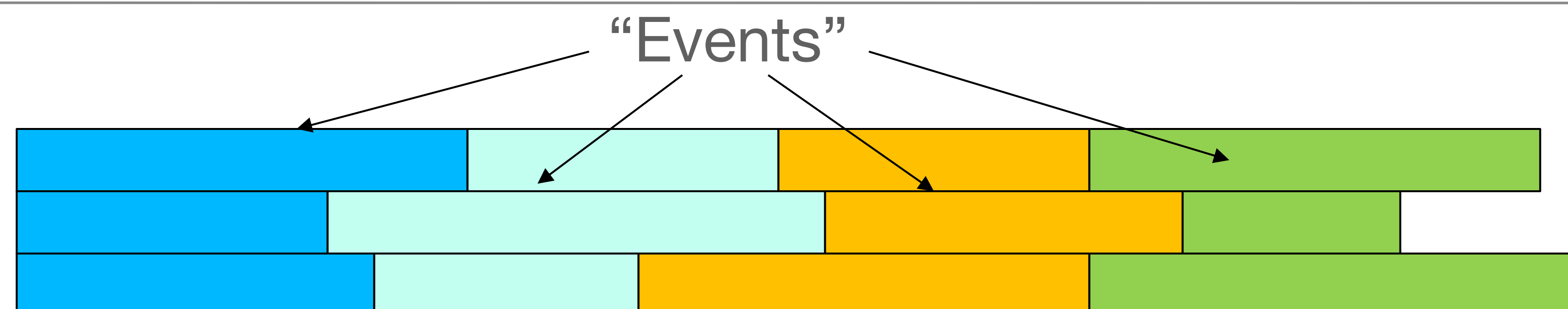
I have come to regard a particular feature of SRO as the defining property, even if you ultimately trigger your front-end:

There is no synchronized end to a given event!

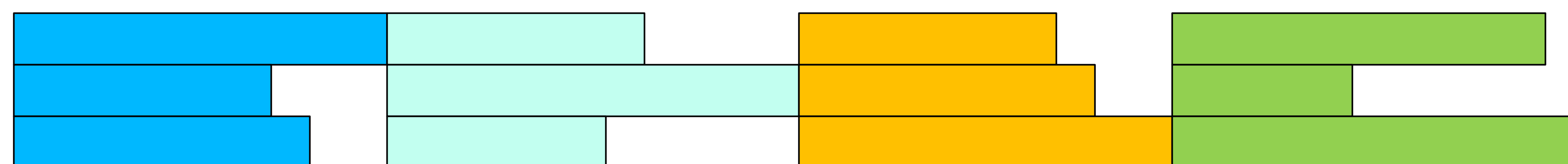
While "event" *n* is streaming, in other places, event *n-1* (or *-2, -3, -4…*) isn't finished yet, and data from different crossings are interleaved

And that's where the speed increase can be significant even for "classic" systems

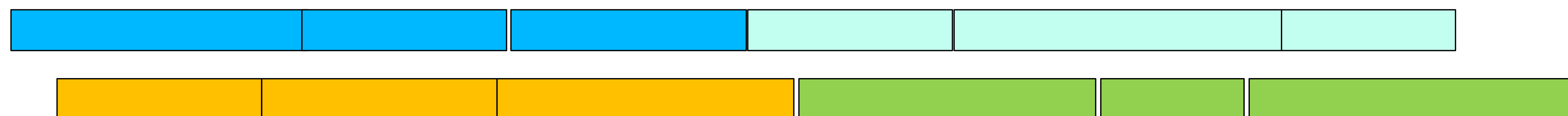
How do you deal with that?



You could throttle your event rate with busies to not let that happen:



Or, if you insist on "event boundaries" in your data, you could buffer those event fragments in DAQ memory, assemble them, then write out



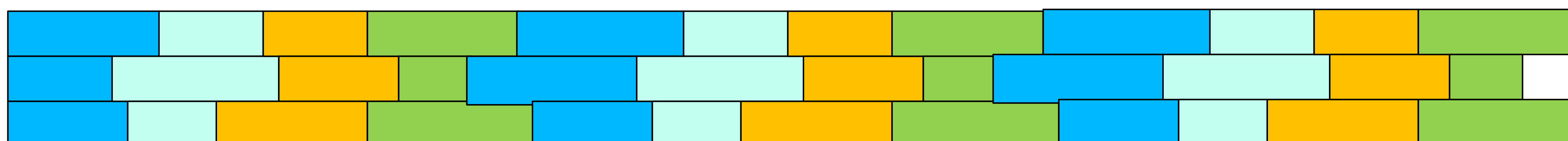
Remember that you can often not "ask" a device to give you its data when it's convenient for you, you need to be ready to catch them as they come (e.g. network)

You have brought the event builder back, have to do it online, have to do it right the first time…

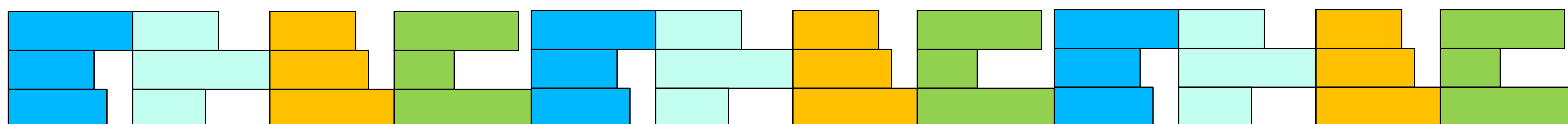
ECCE

Offline sorting streaming data into events/crossings

A “streaming data” offline **pool** holds a number of crossings worth of data (like 1000) and sorts them by crossing number (beam clock value)



It then hands out per-crossing data:



As processed data (oldest crossings) get discarded, new data are inserted (high-and low-water marks)

By doing that you relieve the DAQ from a lot of bookkeeping tasks. Faster! Safer!

Think of a test beam setup (or your Lab setup) for a moment

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

We have the PHENIX run database as an example – we log "everything", AND there's infrastructure and support so most people know how to get at it.

I'm not disputing the need for a database, I'm saying that a test beam or your test 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?

A database allows you to search for runs with certain properties. But capturing this information in the raw data file is more flexible 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…

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 shown before)

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…

I don’t have time to go through this in detail today; I have added a few backup slides

the short version:

You can capture “meta data” at the begin of a run, at the end, or periodically (say, read temperatures at he start, and the end, and every 120s)

“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

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
```

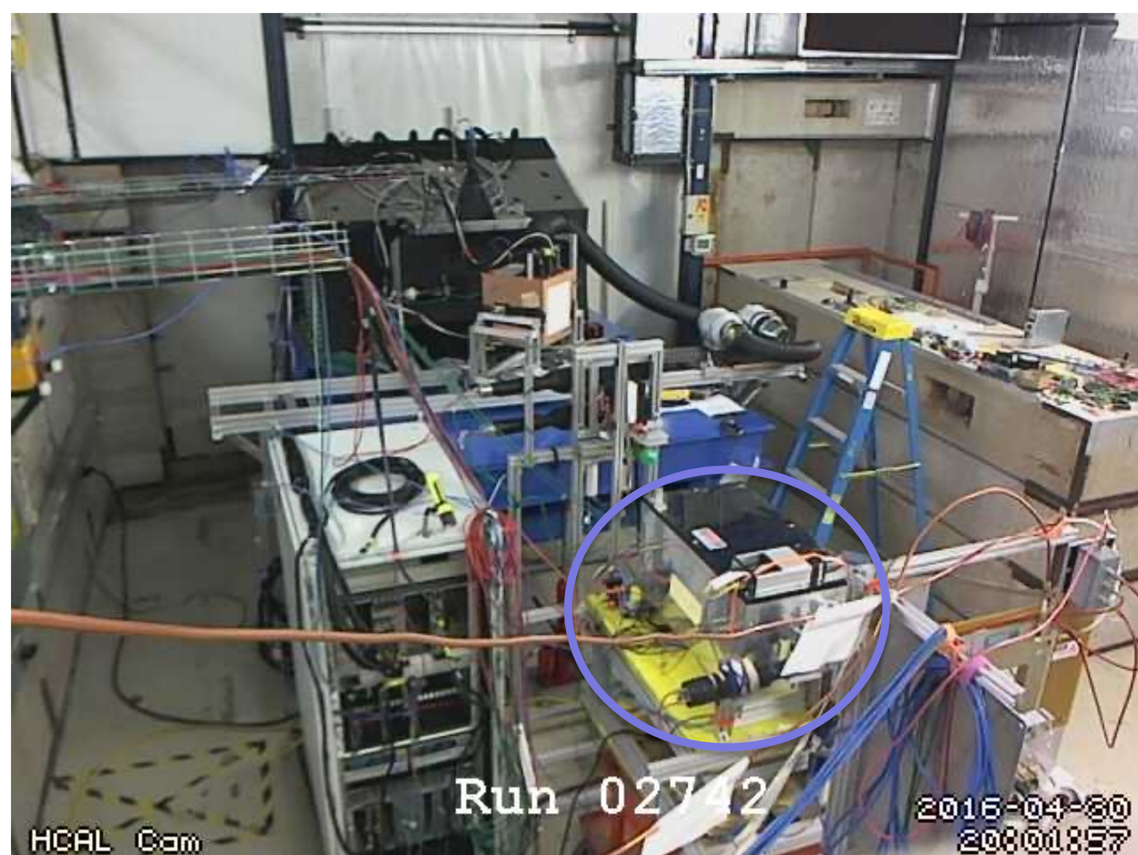
```
$ ddump -t 9 -p 923 beam_00002268-0000.prdf
S:MTNRG  = -2     GeV
F:MT6SC1 =  11846       Cnts
F:MT6SC2 =  7069        Cnts
F:MT6SC3 =  3883        Cnts
F:MT6SC4 =  0           Cnts
F:MT6SC5 =  283048      Cnts
E:2CH    =  1058  mm
E:2CV    =  133   mm
E:2CMT6T =  74.13 F
E:2CMT6H =  37.26 %Hum
F:MT5CP2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

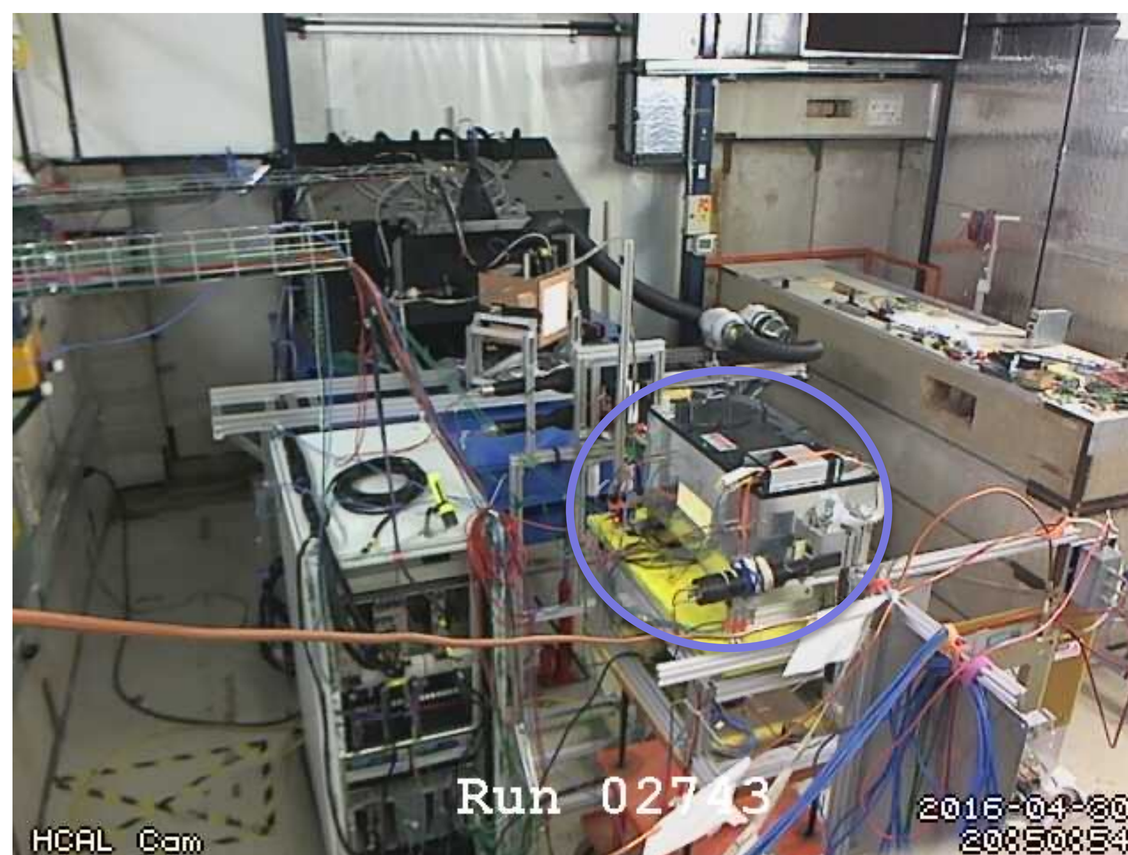
More Forensics (my poster child why this is so useful…)

“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
```





ECCE

All about automation

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 usually want to write a script to do that)

In that sense, there are no 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

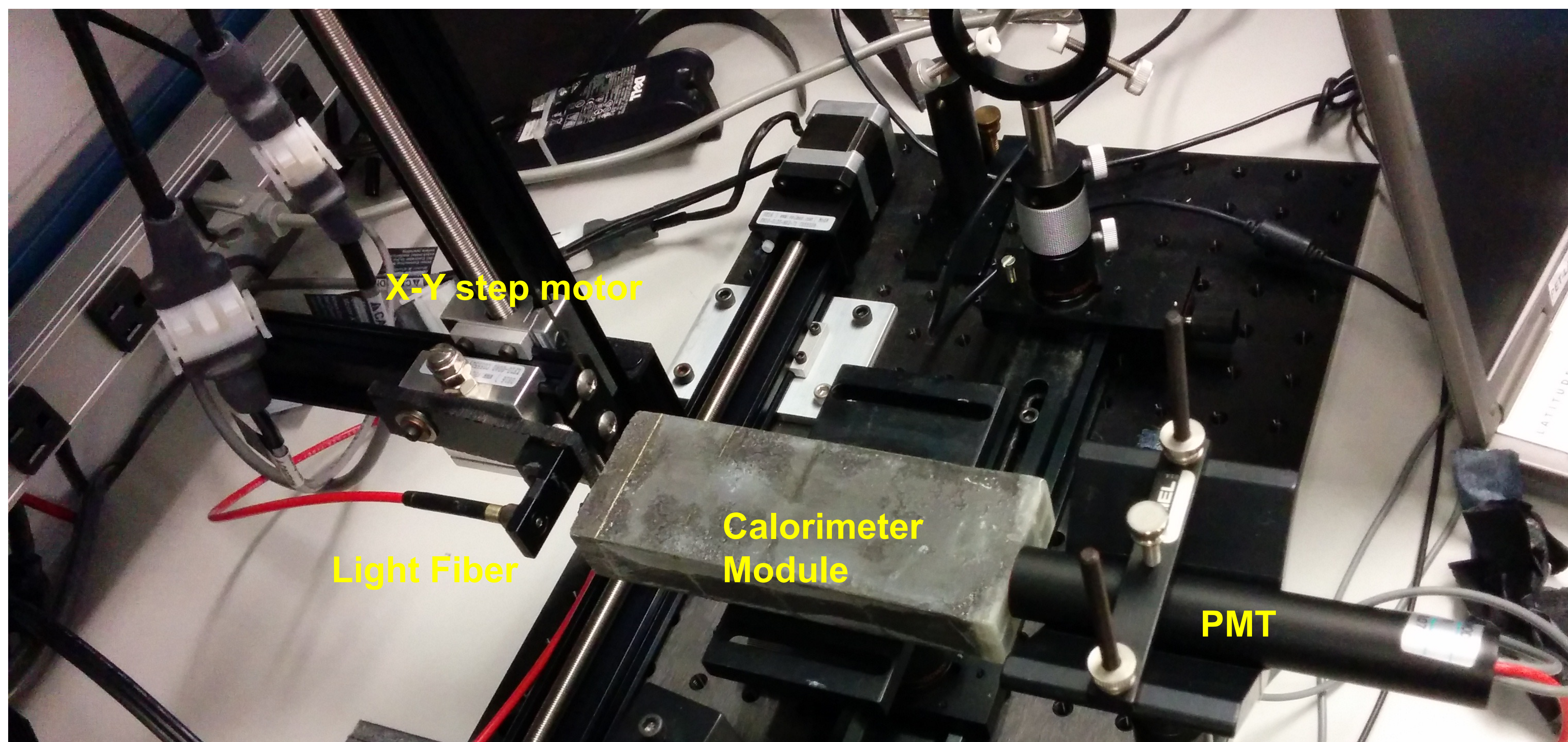
Measurements on autopilot through scripting

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

The DAQ operation becomes an integral part of your shell environment

```
#! /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
```

Automatic end after 4000 events → `rcdaq_client daq_set_maxevents 4000`

25 positions in y → `for posy in $(seq 25) ; do`

move the Y motor → `quickmove.sh $CURRENTPOSY 2`

50 positions in x → `for posx in $(seq 50) ; do`

move the x motor → `quickmove.sh $CURRENTPOSX 1`

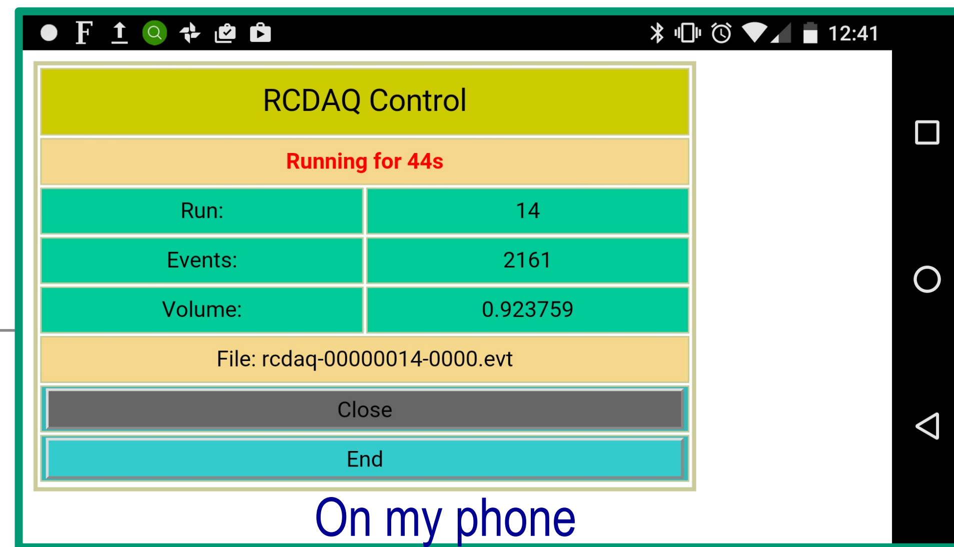
start the DAQ → `rcdaq_client daq_begin`

next x → `done`

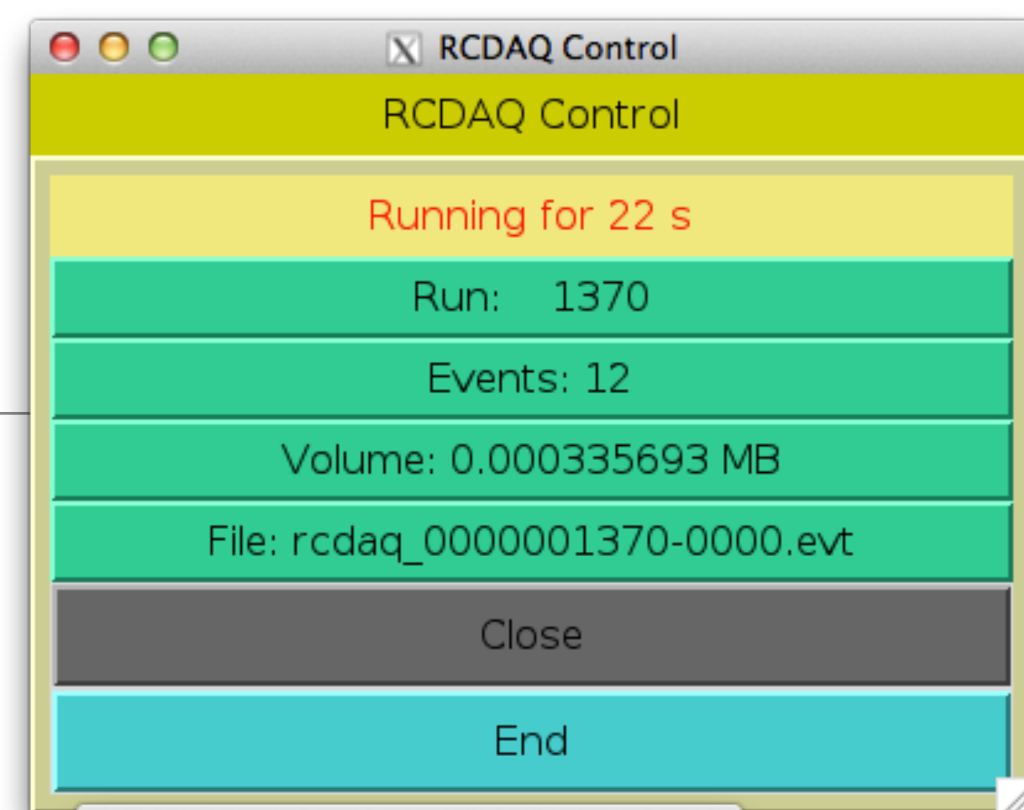
next y → `done`

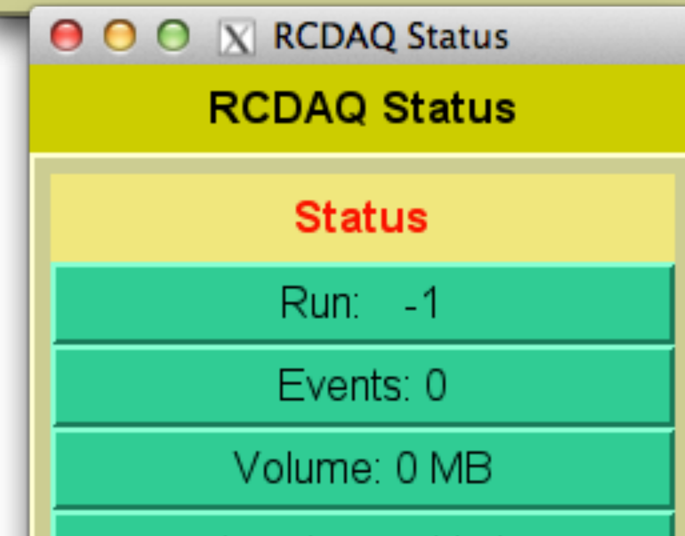
GUIs

GUIs

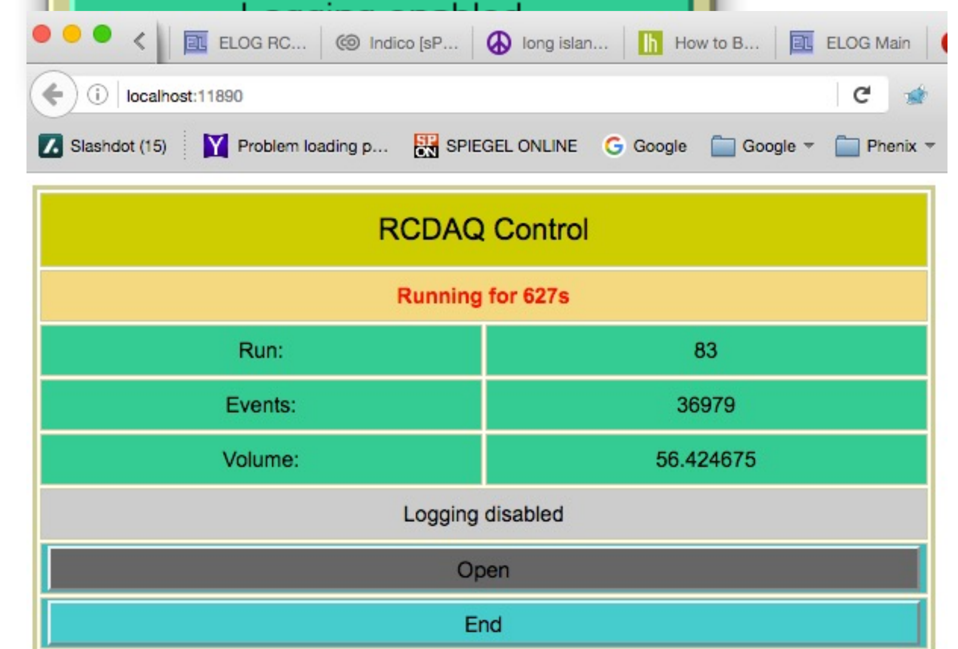


On my phone





Perl-TK



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 the GUI issue standard commands, parse the output
- Slowly transitioning to Web-based controls (web sockets + Javascript)

ECCE

Run Control (all currently available front-ends)

Meet the Run Control Setup that can deal with our > 60 components

Run Control

Run Control

23:55:32

Stopped

seb01 seb02 seb03 seb05
ebdc02 ebdc03 ebdc04 ebdc05
ebdc12 ebdc13 ebdc14 ebdc15
ebdc22 ebdc23 ebdc24 ebdc25
ebdc32 ebdc33 ebdc34 ebdc35

Run Control

Run Control

23:56:55

Stopped

Run: -1

Events: 0

Logging Enabled

Close

Begin

seb01 seb02 seb03 seb05 seb06 seb07 seb08
ebdc02 ebdc03 ebdc04 ebdc05 ebdc06 ebdc07 ebdc08
ebdc12 ebdc13 ebdc14 ebdc15 ebdc16 ebdc17 ebdc18
ebdc22 ebdc23 ebdc24 ebdc25 ebdc26 ebdc27 ebdc28
ebdc32 ebdc33 ebdc34 ebdc35 ebdc36 ebdc37 ebdc38

Run Control

Run Control

23:56:17

Running

Run: 2

Events: 13758

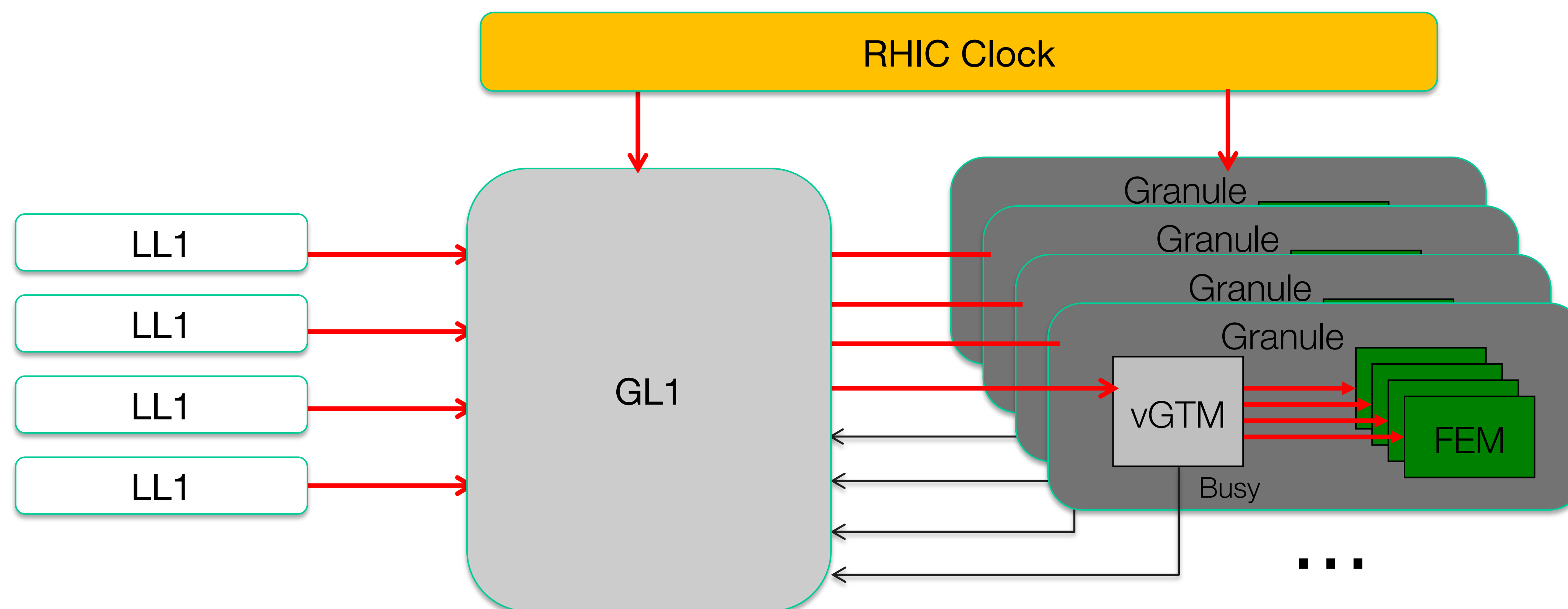
Logging Disabled

Open

End

seb01 seb02 seb03 seb05 seb06 seb07 seb08 seb09 ebdc00 ebdc01
ebdc02 ebdc03 ebdc04 ebdc05 ebdc06 ebdc07 ebdc08 ebdc09 ebdc10 ebdc11
ebdc12 ebdc13 ebdc14 ebdc15 ebdc16 ebdc17 ebdc18 ebdc19 ebdc20 ebdc21
ebdc22 ebdc23 ebdc24 ebdc25 ebdc26 ebdc27 ebdc28 ebdc29 ebdc30 ebdc31
ebdc32 ebdc33 ebdc34 ebdc35 ebdc36 ebdc37 ebdc38 ebdc39

Timing System



The Local-Level 1's provide input to the GL1

The GL1 transmits clock and trigger decision to the vGTMs

The vGTMs transmit clock and trigger info to their front-ends

They are aware of their granule's busy state

What's in the raw data

Each readout electronics flavor (Digitizer, TPC-FEE/FELIX, INTT, MVTX) has a unique way to embedding the BCLK

Example TPC:

The 20 bits FEE clock data (57K crossings, 17ms) allow us to check the internal alignment of the 26 FEEs per FELIX, also delta within FELIX structure

The 40 bits (good for ~5 hrs) correlate the structure with the BCLK

FELIX-type readout

The vGTM sends this structure to the FELIXes:

6x BCLK == 6 words per crossing transmitted

Bit Number	Function	Beam clock phases					
		0	1	2	3	4	5
7-0	Mode bits /BCO	Modebits bits 7-0	BCO bits 7-0	BCO bits 15-8	BCO bits 23-16	BCO bits 31-24	BCO bits 39-32
8	Beam clock phase0	1	0	0	0	0	0
9	LVL1 accept	X	0	0	0	0	0
10	Endat 0	X	X	X	X	X	X
11	Endat 1	X	X	X	X	X	X
12	Modebit enable	1	0	0	0	0	0
15-13	User bits	3 user bits	0	1	2	3	4

40 bits BCLK (BCO)

The MVTX runs its on 40MHz clock, 2 clock values reported

The TPC embeds 20 bits, as explained

INTT – remains to be seen

TPC data

Clock values embedded in FEE data

```
0000000 feee ba5e 0ff1 0001 7229 f7a0 0088 0004
0000020 002f 8782 0004 ffff 0081 0000 0050 0050
...
0001020 d72c 0081 feed 0000 0088 3e2b 0004 feed
0001040 000f 0088 9f7a 0000 0000 0007 ffff 58af
...
0002100 0088 ad79 0004 feed 0017 0088 9f7a 0000
0002120 0000 000f ffff 58af 0081 0008 0000 ffff
...
0004740 0004 feed 0027 0047 0088 9f7a 0000 8782
0004760 0000 0004 001f ffff ffff 58af 0000 0000
```

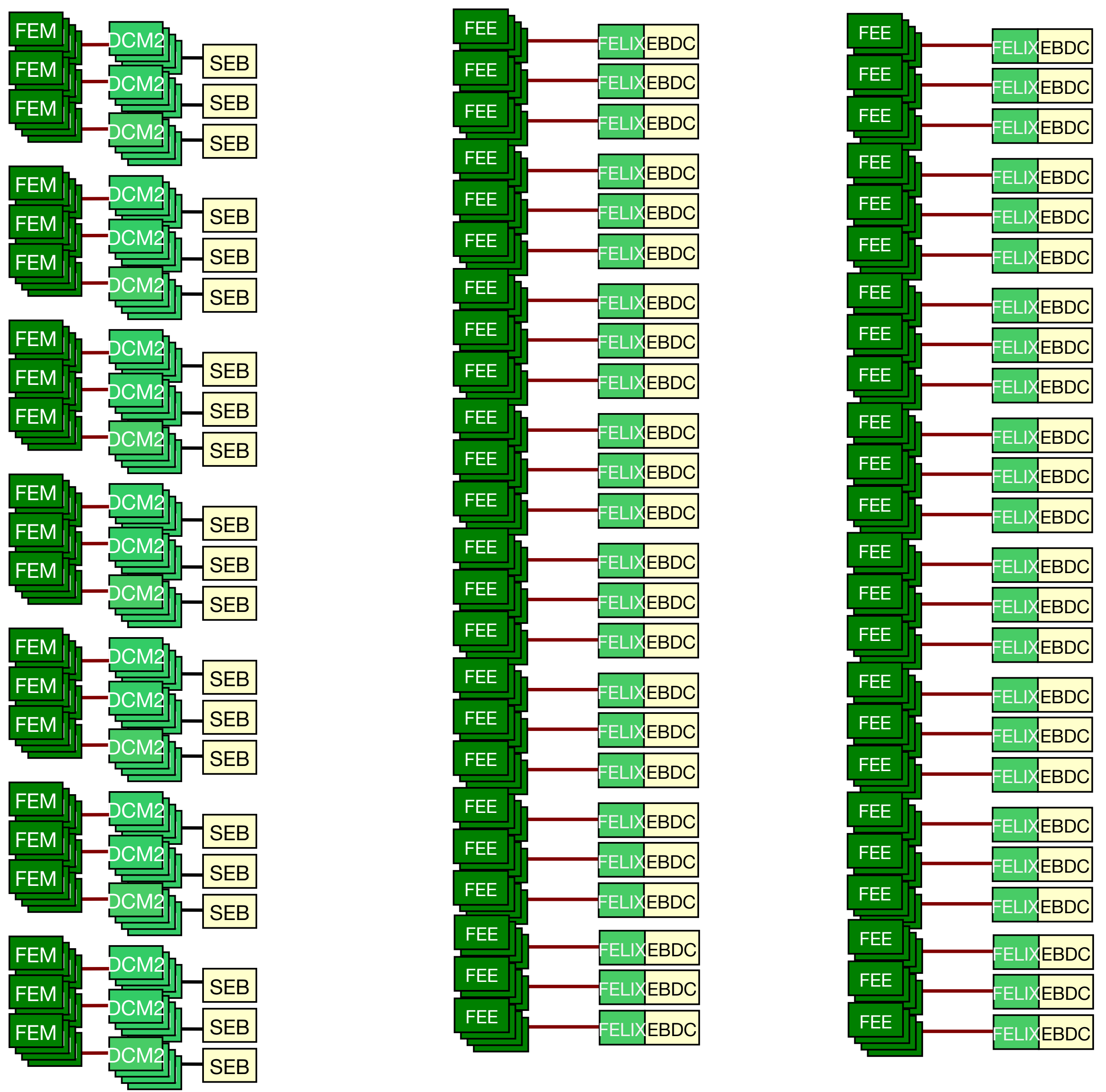
FELIX Hdr

FEE structures

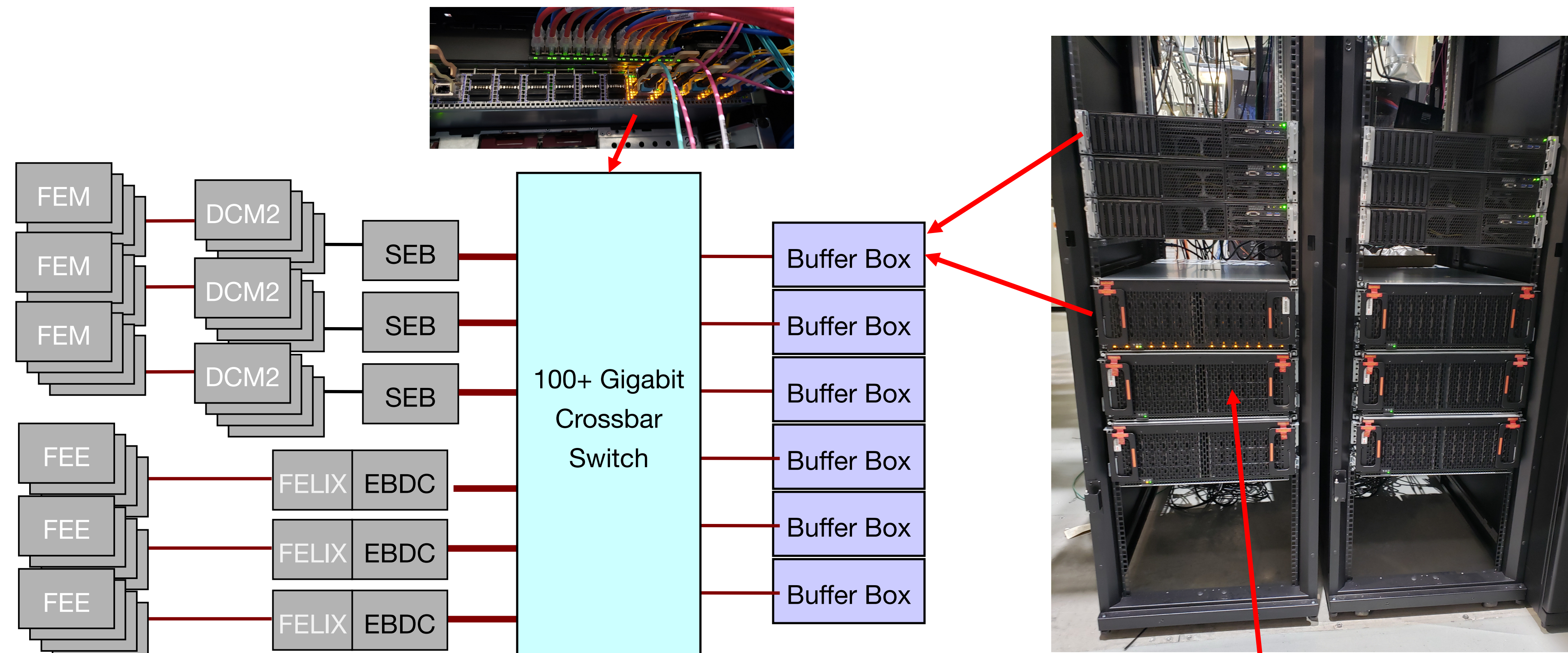
Clock values

```
bx 9f7a0
bx 9f7a0
bx 9f7a0
bx 9f7a0
...
```

Current hardware



… and the switch and the “Buffer Boxes”



JBOD = “Just a Bunch Of Disks”

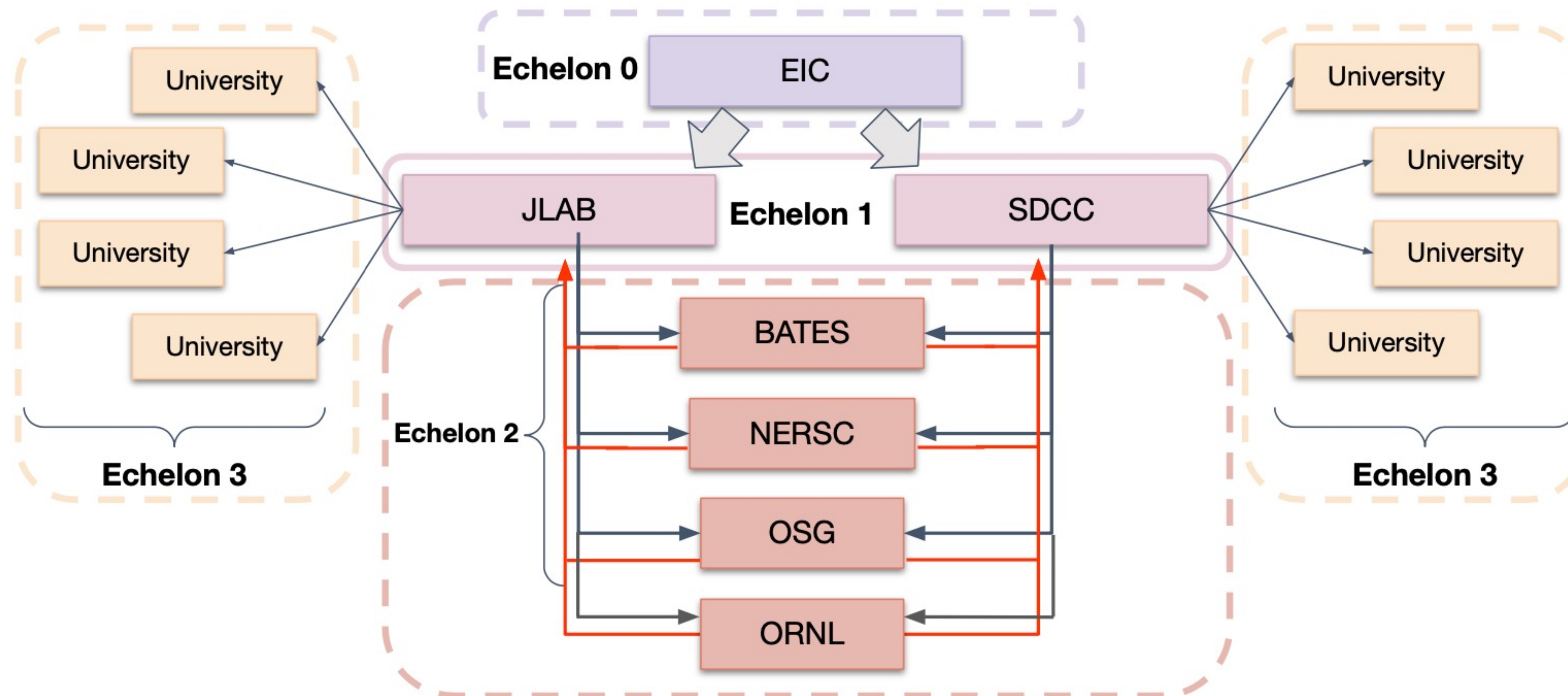
102 14TB disks, ~1.5PB raw disk space

~ 1.2PB usable after RAID etc

6 BBs -> 7PB disk space

JBOD enclosure

Connection to offline computing



Pretty standard GRID/distributed computing paradigm