MLP’s Status and to-do’s for the Jlab test

What I have, and almost have

- RCDAQ is working for the H2GCROC readout (one KCU)
- CAEN readout is a standard feature
- Viable Online monitoring (IMHO the “barebones” project I made covers a lot of what we need)
- Multiple RCDAQ servers (standard feature) (next slides)
- Run Control to coordinate multiple servers together
- Plugin support for >1 KCU (we had some discussions already). I had to make an emulator as I have only one card. Work in progress. [alternatives available]
- Astropix readout – I demonstrated a crude version, mostly reverse-engineering existing python code
- The decoding for astropix in alpha at best
- We need to think of a common busy latch.

Multiple RCDAQs

The H2CCROC readout is inherently streaming (never mind that we have a trigger). Streaming == “there is no well-defined end to an event”.

The CAEN readout is a genuine non-streaming device.

Streaming and not-streaming usually requires different RCDAQs, as there is no one-on-one correlation with events.

And that opens the question of how to synchronize the different streams.

1) Can the CAEN and the ROCs operate off a common clock source? (seems a long shot, has not really been tried. I was in contact with the CAEN engineers. Should be possible but seems like another R&D project that I’d rather avoid)
2) RCDAQ has a built-in “real-time clock” device. Its primary purpose is to record the time distance to the previous event. It records the RTCLOCK_MONOTONIC_RAW (usually raw ns ticks). Since multiple RCDAQs run on the same PC == same clock, it should be possible. (I tried that, later slides)
3) I learned that the 2 KCUs make point-to-point connections to individual 10G network interfaces. This opens the possibility to beat the “multiple KCU” issue with 2 RCDAQs, too.

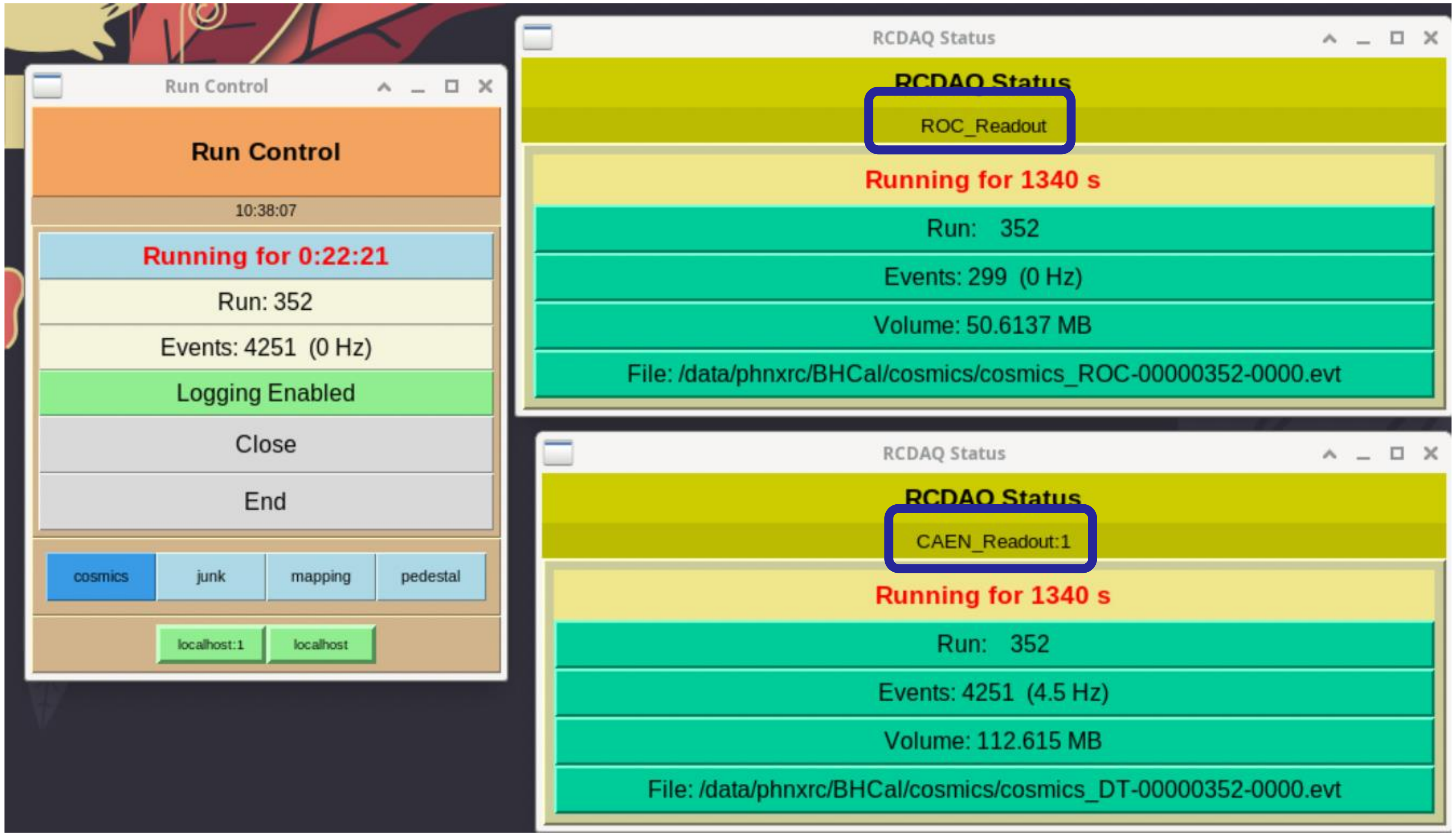
Multiple RCDAQs

The operation of multiple RCDAQs is synchronized by “Run Control”

Here are the 2 RCDAQs (one for the KCU, one for the CAEN) running together

We will likely need to run the AstroPix readout in a dedicated RCDAQ, too

In the same spirit, we could run 4 (2 KCUs + AP + CAEN). Remains to be seen which way is better.



Run Control

10:38:07

Running for 0:22:21

Run: 352

Events: 4251 (0 Hz)

Logging Enabled

RCDAQ Status

ROC_Readout

Running for 1340 s

Run: 352

Events: 299 (0 Hz)

Volume: 50.6137 MB

File: /data/phnxrc/BHCal/cosmics/cosmics_ROC-00000352-0000.evt

RCDAQ Status

CAEN_Readout:1

Running for 1340 s

Run: 352

Events: 4251 (4.5 Hz)

Volume: 112.615 MB

File: /data/phnxrc/BHCal/cosmics/cosmics_DT-00000352-0000.evt

“Trigger” Synchronization

I’m hoping that we can get by with a ”sync” object (the RT-Clock I alluded to earlier)

Works in this special case where everything runs on the same PC

Would never work between different PCs

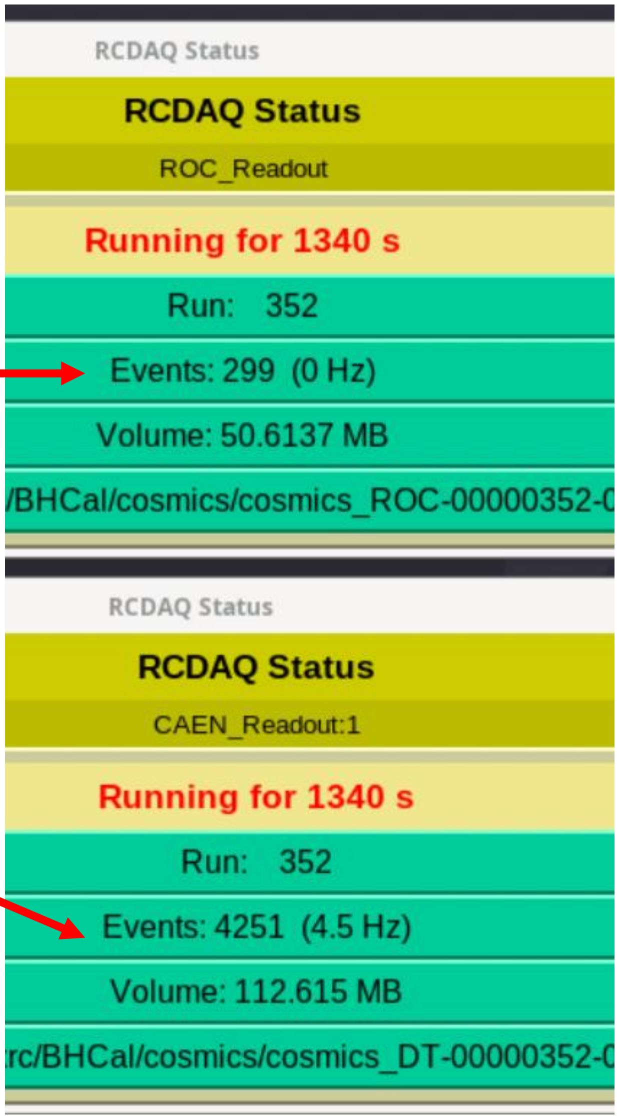
Maybe you noticed this before –

That’s because the CAEN makes one ”real” event for each trigger

For the SRO-ROC this is more like a timeframe, multiple triggers are in one ”event”
- I should start calling those “timeframes”

But on top of this, we don’t have a busy latch here, so there is no guarantee that both RCDAQs take the same triggers

But once we have one, the CAEN will give you a true representation of beam triggers taken



RCDAQ Status

RCDAQ Status

ROC_Readout

Running for 1340 s

Run: 352

Events: 299 (0 Hz)

Volume: 50.6137 MB

/BHCal/cosmics/cosmics_ROC-00000352-0

RCDAQ Status

RCDAQ Status

CAEN_Readout:1

Running for 1340 s

Run: 352

Events: 4251 (4.5 Hz)

Volume: 112.615 MB

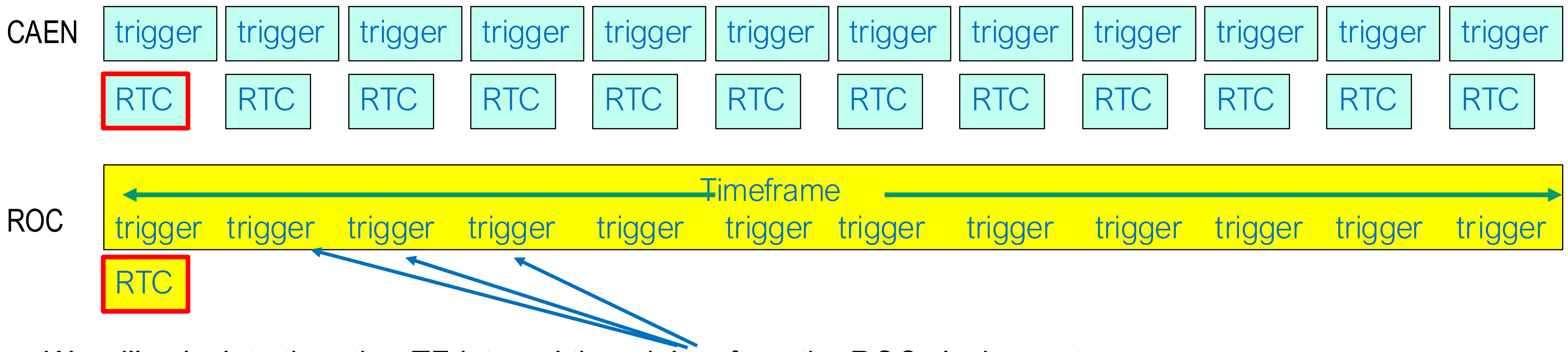
rc/BHCal/cosmics/cosmics_DT-00000352-0

Correlating Triggers across data streams

I took the 2 files/streams shown before and looked at the real-time clock packets

```
$ dlist -r
Packet  1012     14 -1 (ePIC Packet)   9 (IDRTCLOCK)
Packet 12001 12298 -1 (ePIC Packet) 302 (IDH2GCROC3_10G)
$ dlist -r localhost:1
Packet  1011     14 -1 (ePIC Packet)   9 (IDRTCLOCK)
Packet  2000   6924 -1 (ePIC Packet)  85 (IDCAEN V1742)
```

The RTClock info is present once per event/timeframe – in each CAEN trigger, but only once per ROC TF


CAEN
trigger trigger trigger trigger trigger trigger trigger trigger trigger trigger trigger trigger
RTC RTC RTC RTC RTC RTC RTC RTC RTC RTC RTC RTC
ROC
Timeframe
trigger trigger trigger trigger trigger trigger trigger trigger trigger trigger trigger trigger
RTC

We will calculate the other TF-internal time deltas from the ROC clock counters.

Note that this inherent to all SRO data, so we might as well get used to it. (And why we'd want common clocks)

Next I'm looking for the "red" RTC data

RTClock timeline

I made a “bash analysis” (just using command-line tools) of the time stamps of both streams (1 = CAEN, 0 = ROC), sorted by time

Here you see “timeframes”, that is, one TF contains multiple triggers [cosmics here]

Timeframe

```
856198213866453 0
856198242553041 1
856198283073371 1
856198427944369 1
856198705036789 1
856199795958738 0
856199796268809 1
856200559837603 1
856200916002711 1
856201458347717 1
856201494807778 1
856202291870586 1
856202315680536 1
856202854446462 1
856203311052628 1
856203465666554 1
856203581007788 1
856203613067219 1
856203972615312 1
856204157822250 1
856204583222847 1
856204737259871 1
856204947966373 1
856205153842447 1
856205205248728 1
856205953378046 1
856206819174517 0
856206819589467 1
```

Timeframe

Meanwhile, the CAEN makes one event for each trigger

RTClock timeline

Same list, I just added the difference to the previous time (and shuffled the columns around)

note the green-colored items (units are ns)

```
0 856198213866453 51184649
1 856198242553041 28686588
1 856198283073371 40520330
1 856198427944369 144870998
1 856198705036789 277092420
0 856199795958738 1090921949
1 856199796268809 310071
1 856200559837603 763568794
1 856200916002711 356165108
1 856201458347717 542345006
1 856201494807778 36460061
1 856202291870586 797062808
1 856202315680536 23809950
1 856202854446462 538765926
1 856203311052628 456606166
1 856203465666554 154613926
1 856203581007788 115341234
1 856203613067219 32059431
1 856203972615312 359548093
1 856204157822250 185206938
1 856204583222847 425400597
1 856204737259871 154037024
1 856204947966373 210706502
1 856205153842447 205876074
1 856205205248728 51406281
1 856205953378046 748129318
0 856206819174517 865796471
1 856206819589467 414950
```

These values look like the same trigger

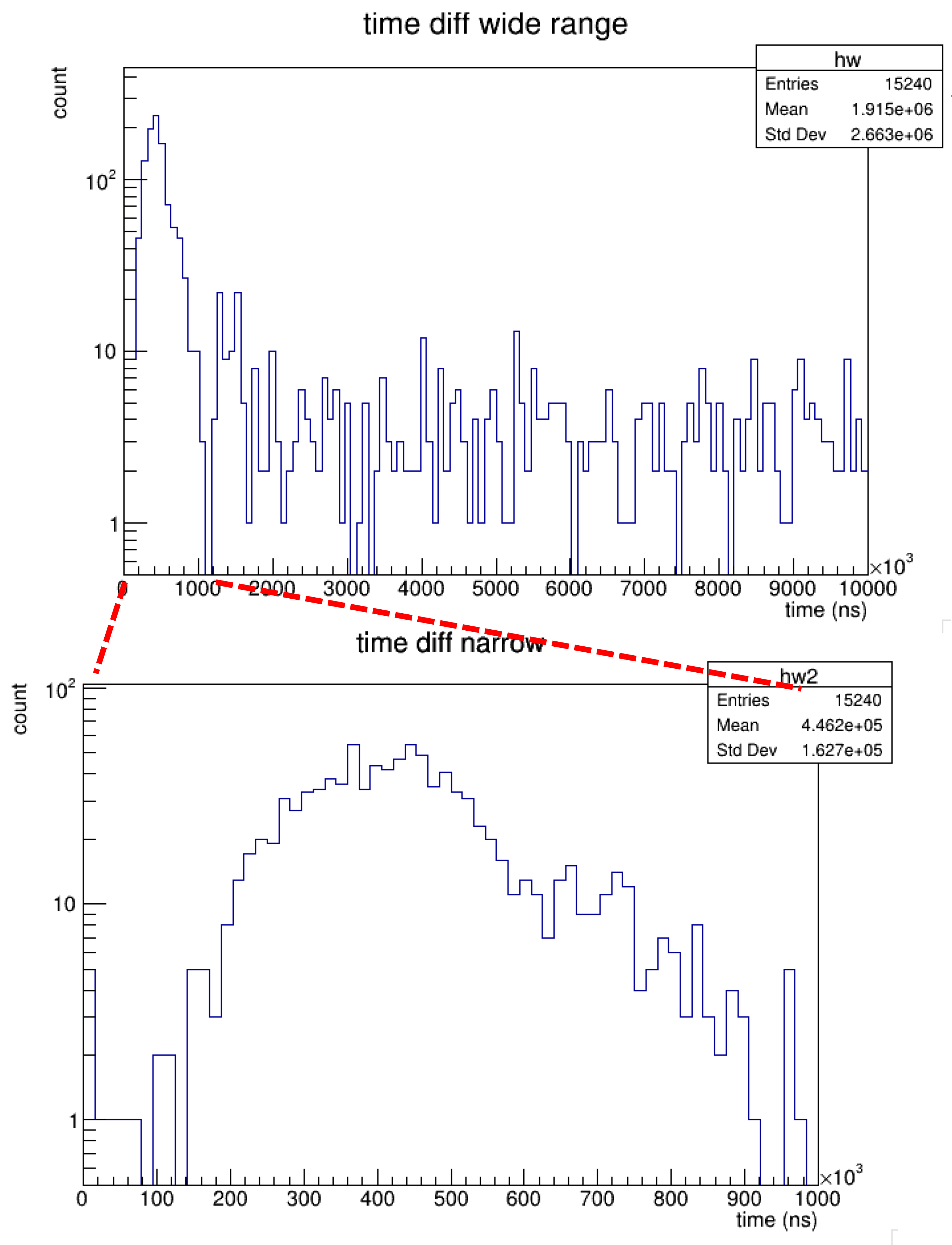
There is still a large time jitter of < 0.5 milliseconds

RCDAQ generates the RTClock packet when it recognizes a trigger, hence the jitter

(ROC trigger when a NW packet is coming, CAEN issues a trigger through CAEN's libraries)

Also, the very first line here appears to be a not-correlated trigger (we have no busy latch)

Coarse results



Here is what I get (coarse analysis)

So, the trigger correlation is < 500μs

Remains to be see if that's good enough

What's the alternative?

Both CAEN and H2GCROC provide a clock tick counter, both last for decades (60 bits/ 64 bits)

Never mind that the CAEN runs on 50MHz and the ROC on 40MHz…

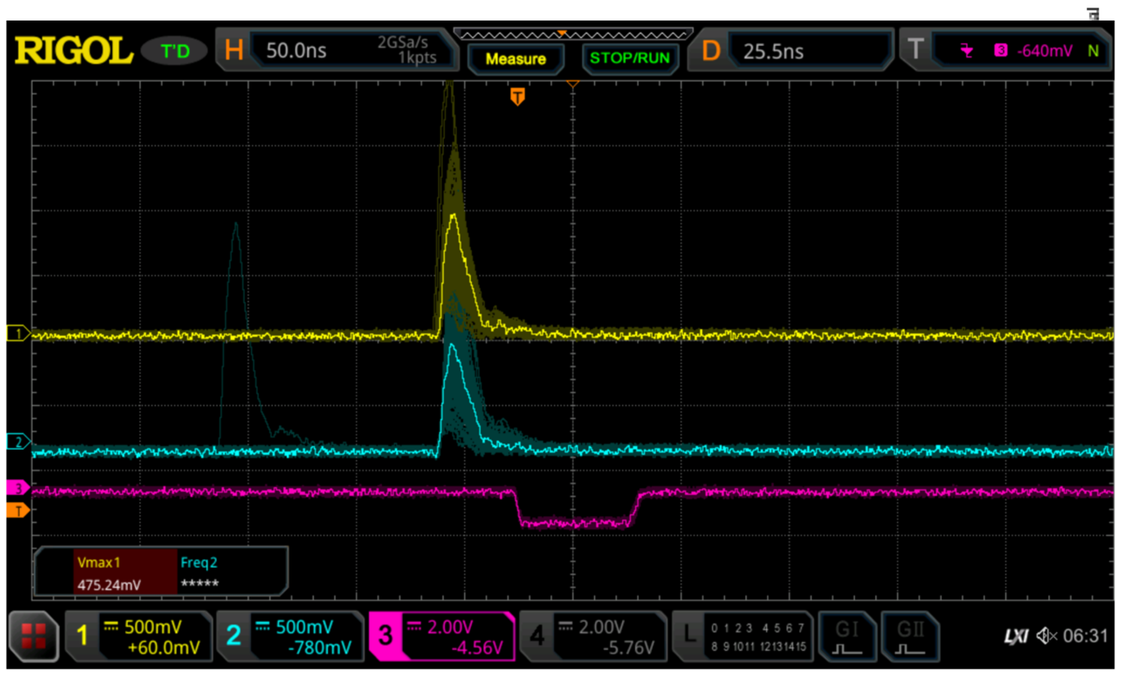
If the clocks could come from the same source, they would be strictly in sync

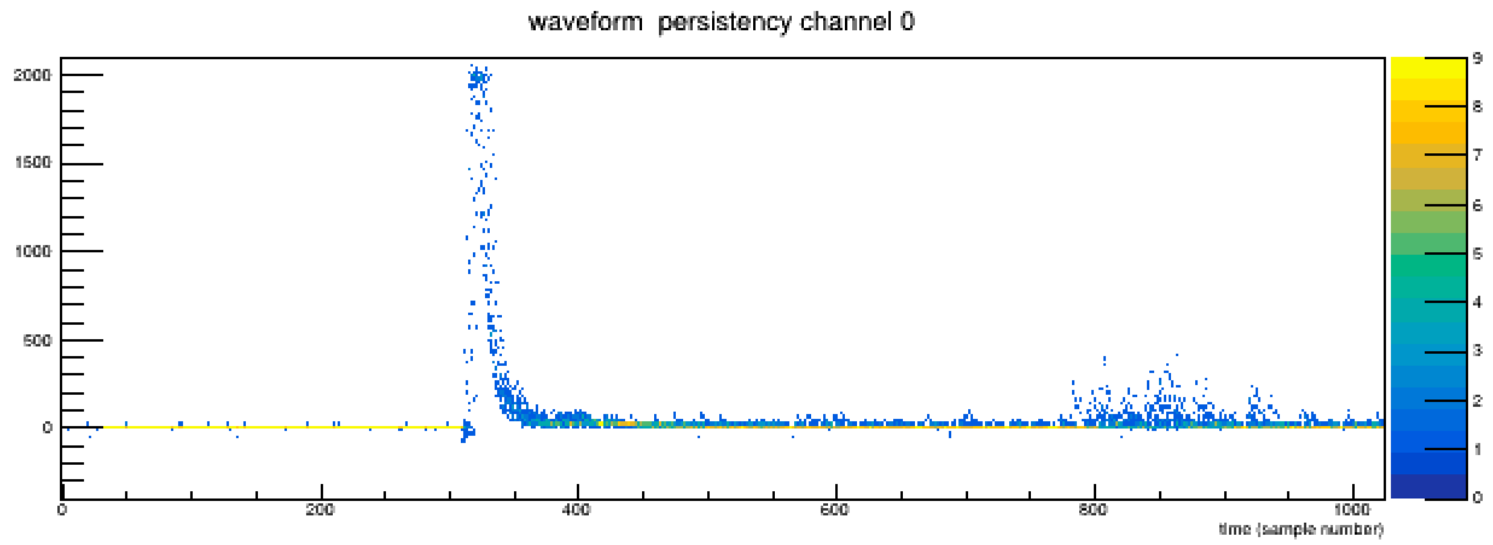
We could equally strictly correlate our events.

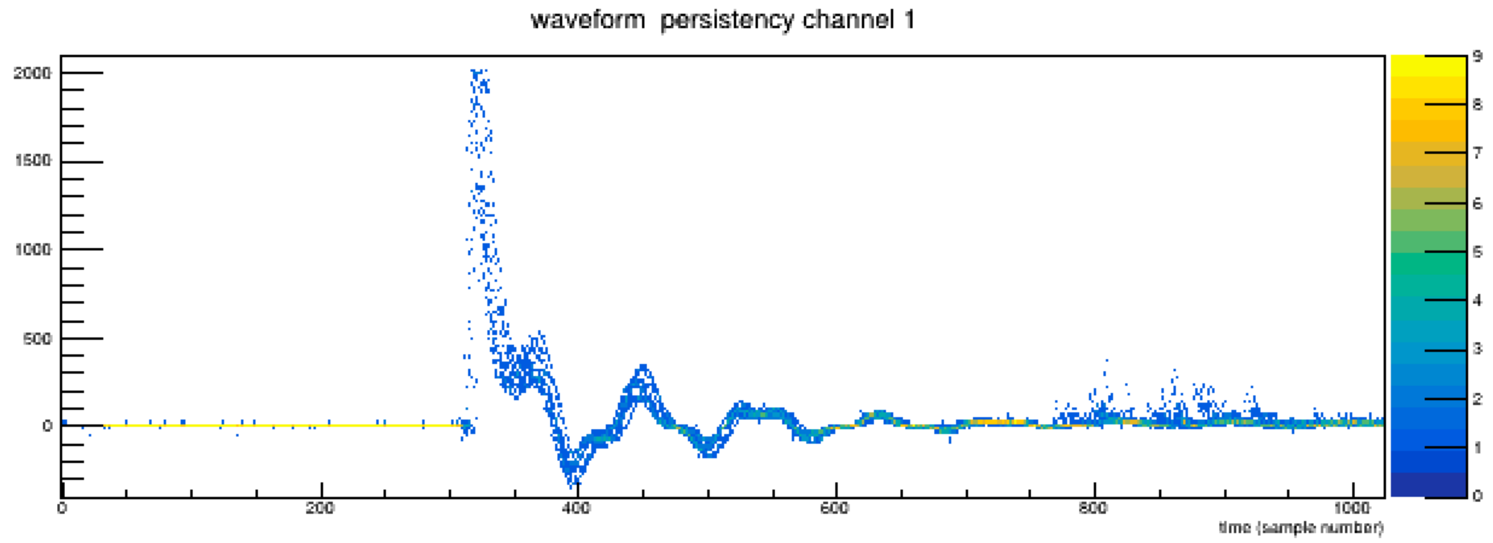
BTW, what does the CAEN show?

My two trigger scintillators have their last-dynode signals brought out.

I connected them to the CAEN to have something on there.







Reading out the AstroPix

This is the part that still needs the most work

I can read the hardware with Bobae’s/Adrien’s python code

And I have the RCDAQ plugin that reads the data *once the board is sending them*

That latter part is where the work still is.

I’m planning to go about it like we do for pretty much every device we read (and the ROC is no exception):

- Use external code to configure the hardware just so
- Then have RCDAQ issue the magic incantation to tell the front-end “please start sending data”
- Read data (that part is done)
- And at the end, issue “please stop sending data”.

Here is the ROC implementation of the start/stop

https://git.racf.bnl.gov/gitea/EIC/EIC_H2GCROC3/src/commit/f8362ed8021a24e2c2a192e4a8d9f93cef89926f/daq_device_h2gcroc3_10G.cc#L76

```
 unsigned char Start[]   =
{0xa0,0x00,0x09,0x00,0x00,0x00,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

return sendData( Start, sizeof(Start));
```

```
 unsigned char Stop[]   =
{0xa0,0x00,0x09,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

  ret =  sendData( Stop, sizeof(Stop));
```

I have worked on reverse-engineering the python code to do the equivalent, not quite there yet

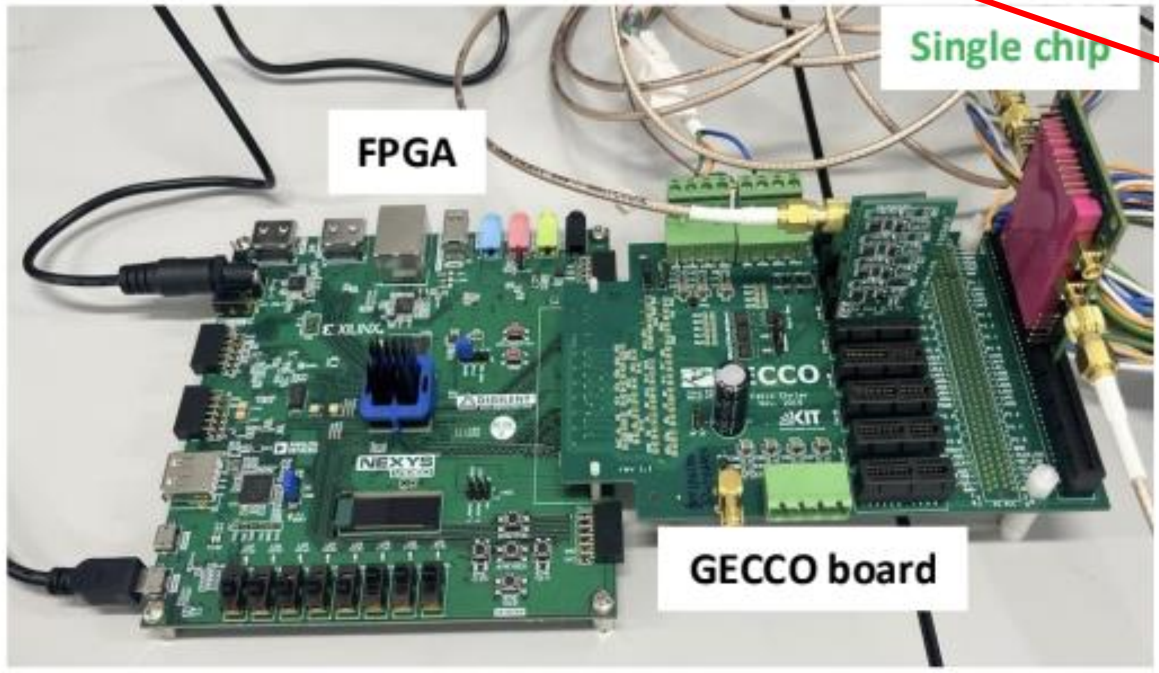
Let me show you where I am

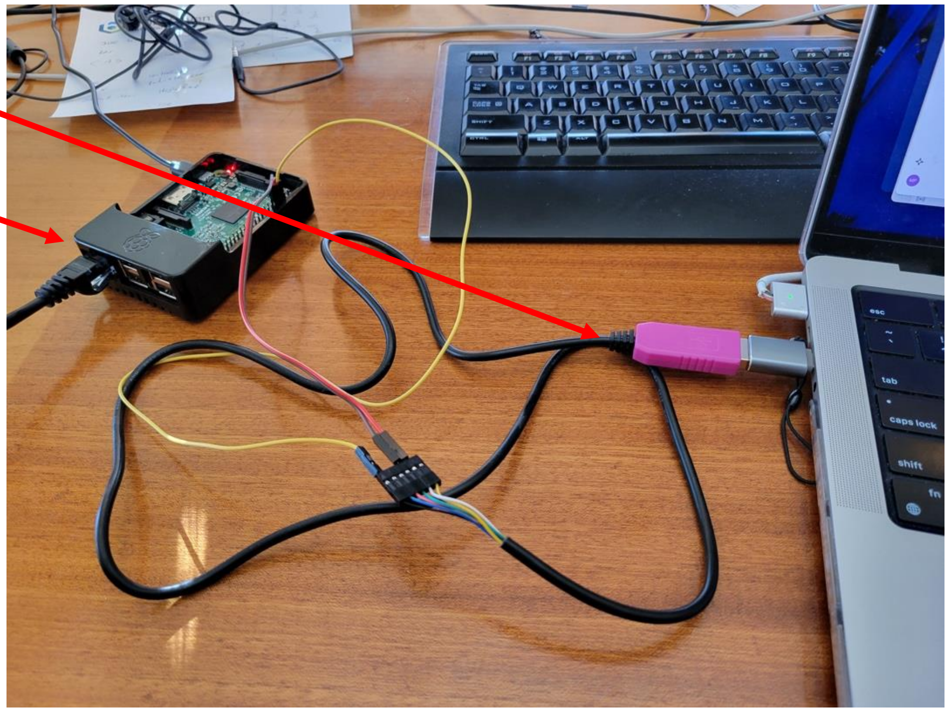
From a few weeks ago: my AstroPix “emulator”

Before I had the hardware, I used the data file that Bobae had made available to play (I can get my own through the python-DAQ now)

I made a utility that fishes out the data sequences from the file

I put this on a Raspberry Pi with a serial line that now acts as a stand-in for the FPGA board





This looks no different than what the FPGA board is sending, I just need to convince the board to do so

This is from a few weeks ago -

I took data files using my emulator-Pi. Here we see that packet 13001. It identifies itself as containing data from our AstroPix (I'm sure there will be more versions beyond V1…)

```
$ dlist    /mac_home/data/AstroPix/tests/test_Astro-00000008-0000.evt
Packet 13001   202 -1 (ePIC Packet) 350 (IDASTROV1)
```

Each such Packet type has a specific dump function that presents the data in a human-readable format:

```
$ ddump   /mac_home/data/AstroPix/tests/test_Astro-00000008-0000.evt
Packet 13001   202 -1 (ePIC Packet) 350 (IDASTROV1)
Nr of units: 72
    # |    layer    id payload location iscol    ts    ToT fpga_ts
    0 |       3      2       4       10      0    73    191 1242370936
    1 |       2      3       4       10      0    72    571 1242370961
    2 |       3      2       4       32      1    72    173 1242370978
    3 |       2      3       4       11      0    72    433 1242371003
    4 |       1      0       4       10      0    12   3329 1242371022
    5 |       2      3       4       12      0    71   1076 1242371045
    6 |       2      3       4       13      0    71   2171 1242371087
 . . .
```

We can discuss better ways to present this, that's what I came up with…

Idiot checks

Once I had the emulator code that reads from Bobae's file, nothing stops me from making a "fake" emulator that sends units with known properties and amounts of data

I took a good unit (that then repeats over and over), but I overwrite the FPGA_TS value to be a counter, 0,1,2,3,4,5…

That allows me to see that all units make it alright (they do). Checked like 20k events.

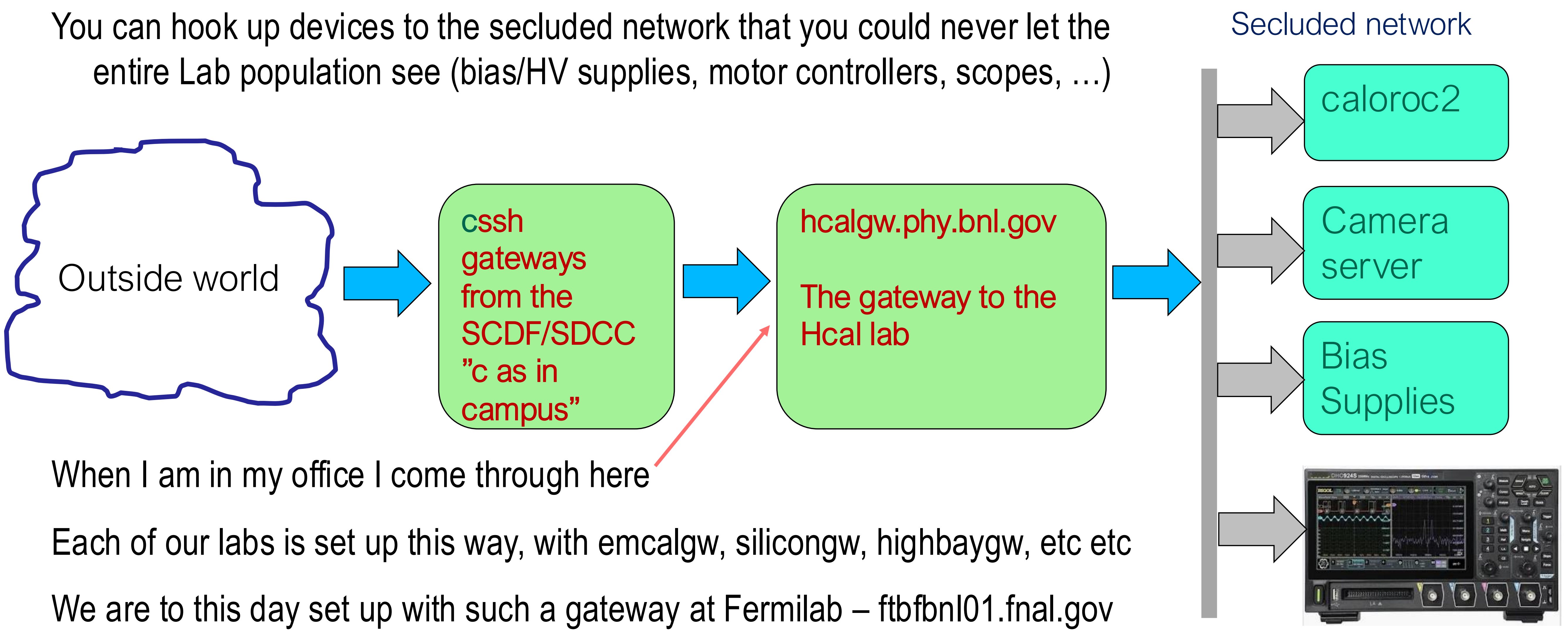
```
$ ddump    /mac_home/data/AstroPix/tests/test_Astro-00000003-0000.evt
Packet 13001   280 -1 (ePIC Packet) 350 (IDASTROV1)
Nr of units: 100
    # |    layer     id payload location iscol    ts     ToT fpga_ts
    0 |        3      2       4       10      0      73     191       0
    1 |        3      2       4       10      0      73     191       1
    2 |        3      2       4       10      0      73     191       2
    3 |        3      2       4       10      0      73     191       3
    4 |        3      2       4       10      0      73     191       4
    5 |        3      2       4       10      0      73     191       5
    6 |        3      2       4       10      0      73     191       6
    7 |        3      2       4       10      0      73     191       7
    8 |        3      2       4       10      0      73     191       8
    9 |        3      2       4       10      0      73     191       9
   10 |        3      2       4       10      0      73     191      10
. . .
```

Changing gears – what I think the network should be

I have been running every test beam like this for ages, and I set up all my labs up like this

Here is the BNL “Hcal lab” setup where the AstroPix hardware currently resides

You can hook up devices to the secluded network that you could never let the entire Lab population see (bias/HV supplies, motor controllers, scopes, …)



When I am in my office I come through here

Each of our labs is set up this way, with emcalgw, silicongw, highbaygw, etc etc

We are to this day set up with such a gateway at Fermilab – ftbfbnl01.fnal.gov

We have perfected the ssh setup on our laptops/desktops so we just say “ssh caloroc2”

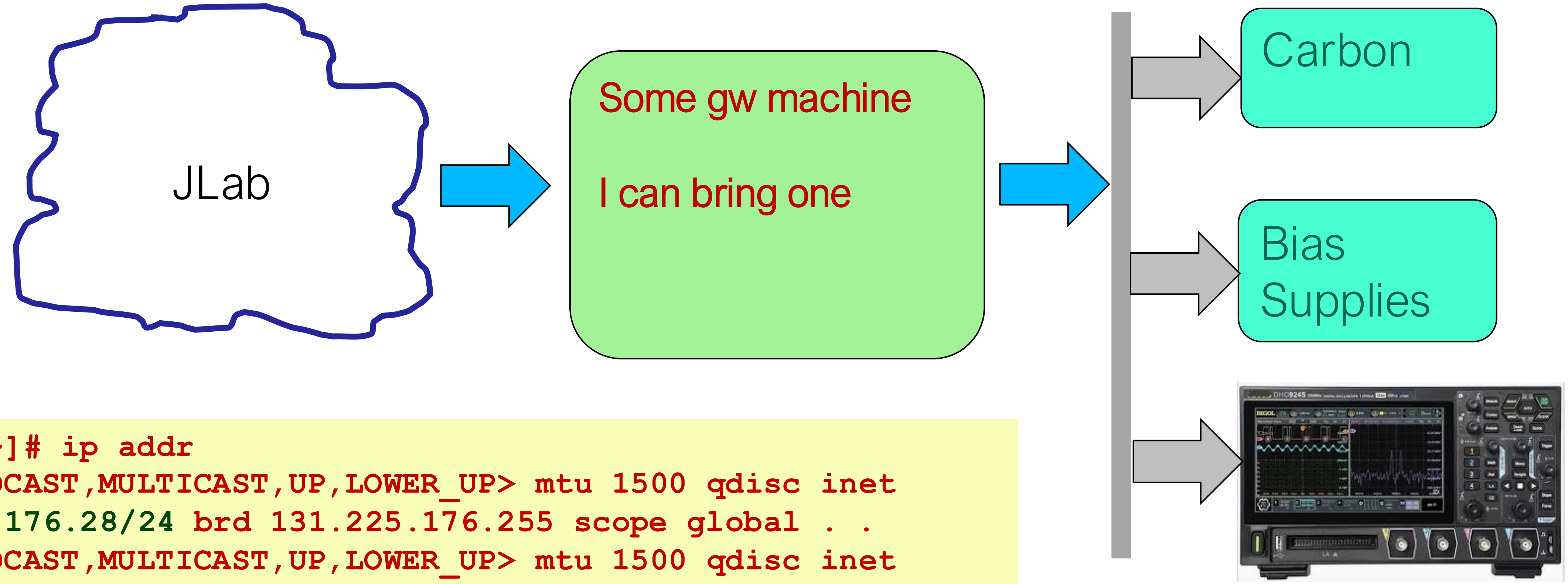
The proposed equivalent at JLab

This would sidestep all kinds of JLab security requirements for the actual DAQ machine(s)

For example, the aforementioned ftbfbnl01.fnal gov complies with all FermiLab requirements (that actually conflict with BNL's…). So FermiLab is happy.

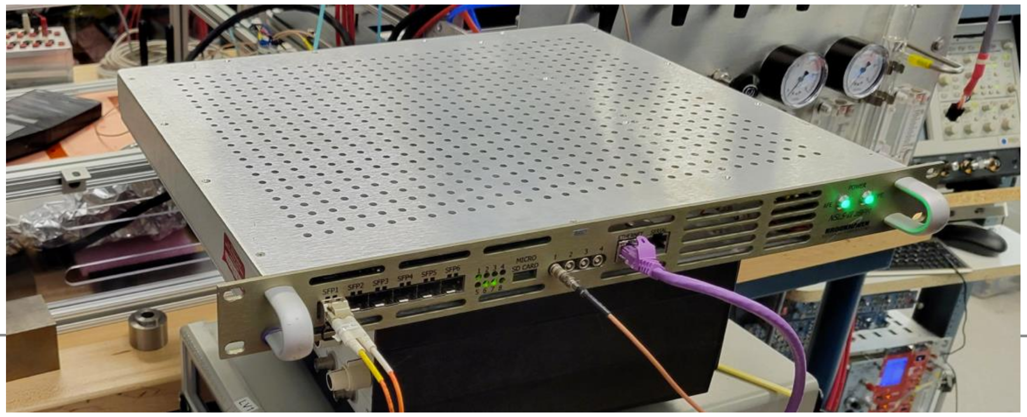
Behind that on that secluded network we can have what we need.

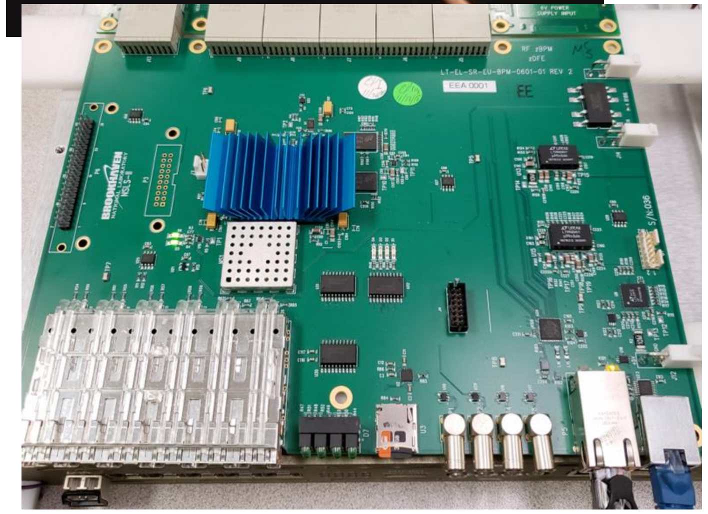
Since the secluded net can have the same setup as we have at BNL, we don't need to change any network setup/scripts/whatever - "take your network with you"



```
[root@ftbfbnl01 ~]# ip addr
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc inet
    inet 131.225.176.28/24 brd 131.225.176.255 scope global . .
3: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc inet
    inet 192.168.60.1/24 brd 192.168.60.255 scope global . . .
```

Busy latch





I have a number of sPHENIX “GL1/GTM” prototypes, BNL-grown 1U FPGA boards

Those look like the best and easiest path to getting a meaningful busy latch

They have 2 built-in TTL ins and outs, and many (>50) 3.3V-grade headers.

They are Zynq-based, meaning that the FPGA has built-in ARM CPUs that run Linux

The CPU has access to registers in the FPGA fabric to configure / take action

So far I have only found time to “warm up” my SDK again so I can get started (re-built one of my earlier projects)

The idea is that a signal sets a busy latch, busy latch going high is sending our trigger, latch is getting reset by RCDAQ when readout is complete

Let me stress that this is vaporware at this point in time (vapor-firmware, that is – I have the hardware)

But I should be able to code this up quickly (you would not be the only customers – pfRICH wants this, too)

There *might* be (but I have not done this so far) a way for it to make the 40 and 50MHz clocks