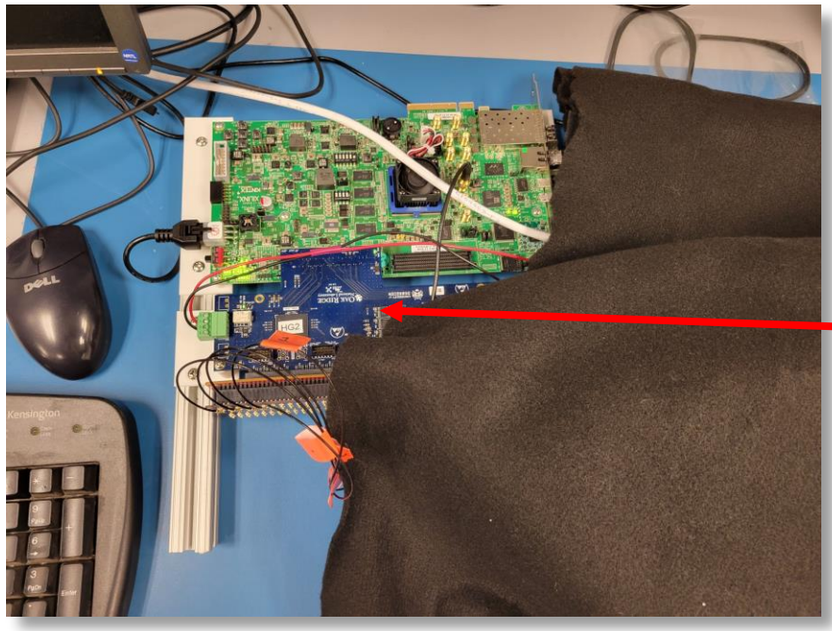
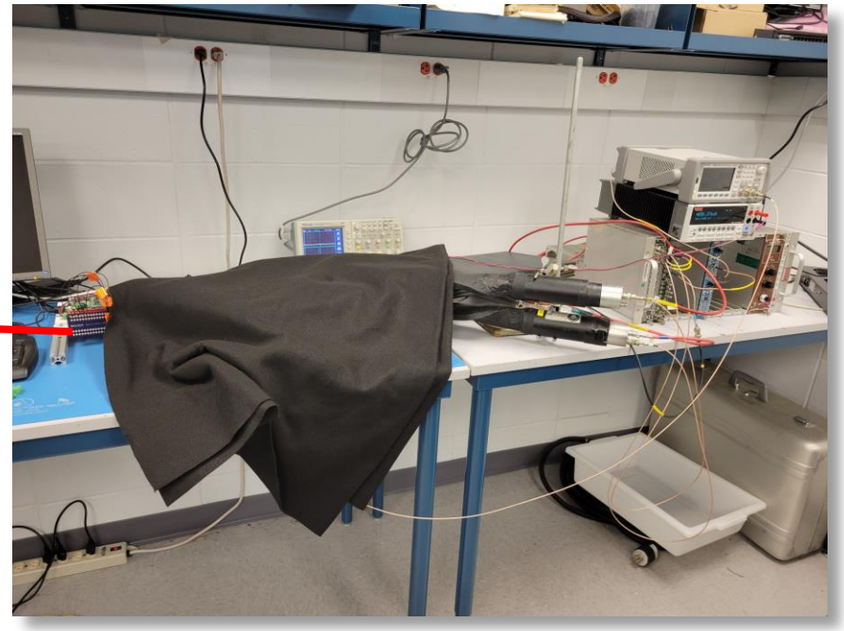
ROC readout in the Hcal Lab



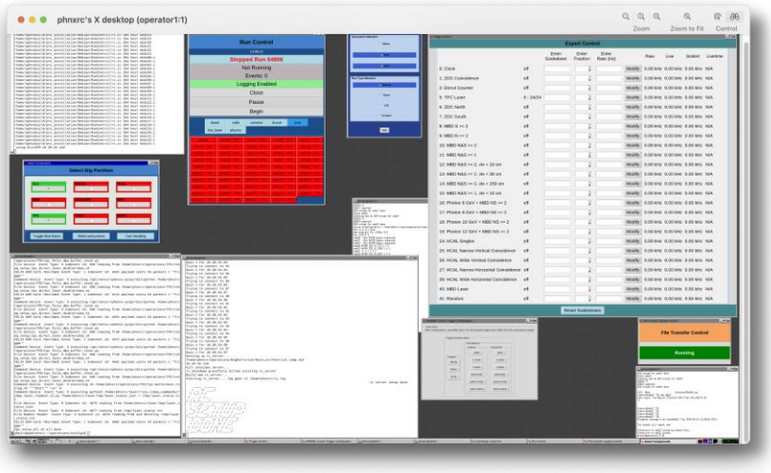


The “VNC lifestyle” – making a shared-by-many desktop

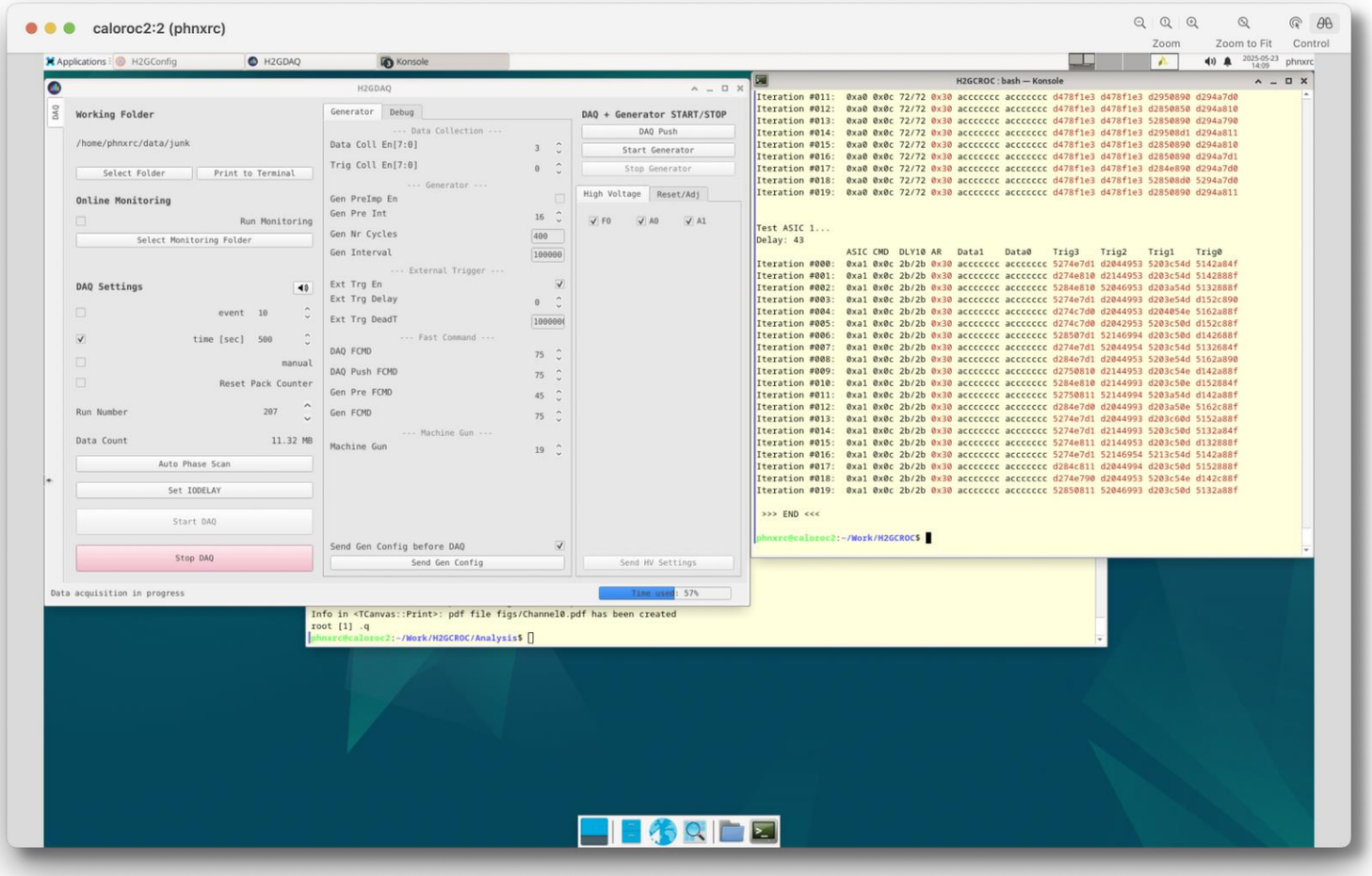
The entire sPHENIX operation lives on shared VNC desktops

This allows us experts to a) see the exact same that the shift crew sees, and b) we can take over as needed

JohnH coined the phrase “VNC lifestyle” – if you tried NoMachine at the SDCC (a 1958 Chevy), this does the same and more, and is a 2025 Porsche.

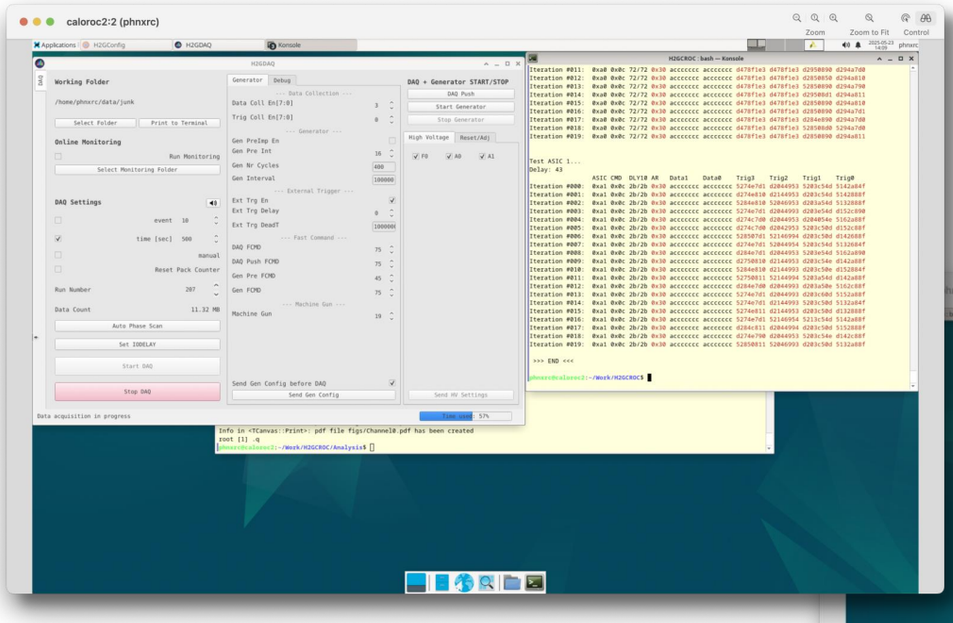


The sPHENIX DAQ station VNC



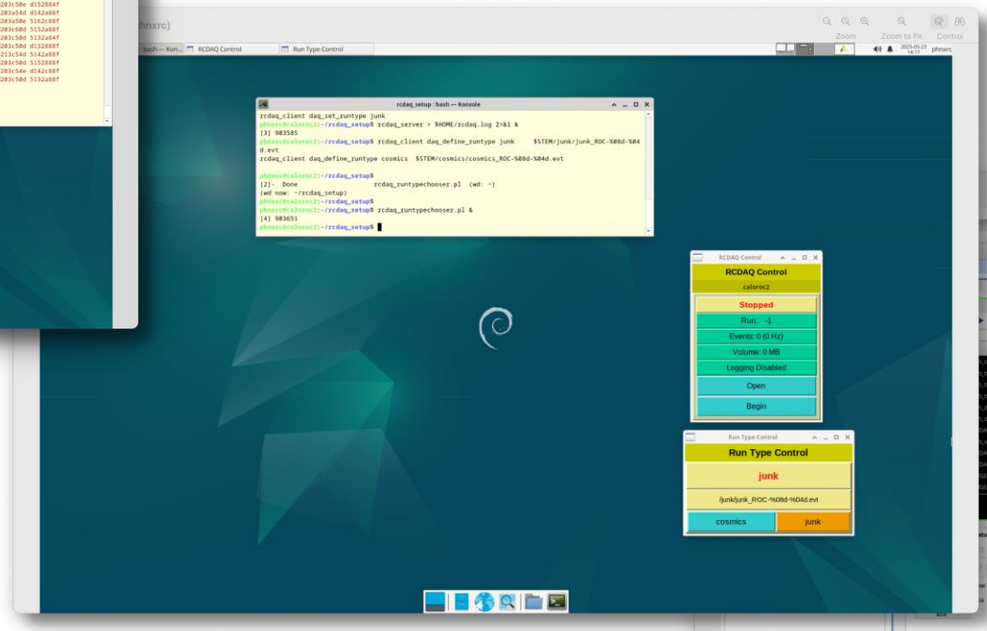
We do the same for the caloroc2 desktop – let’s run everything from here

All is a mouseclick away

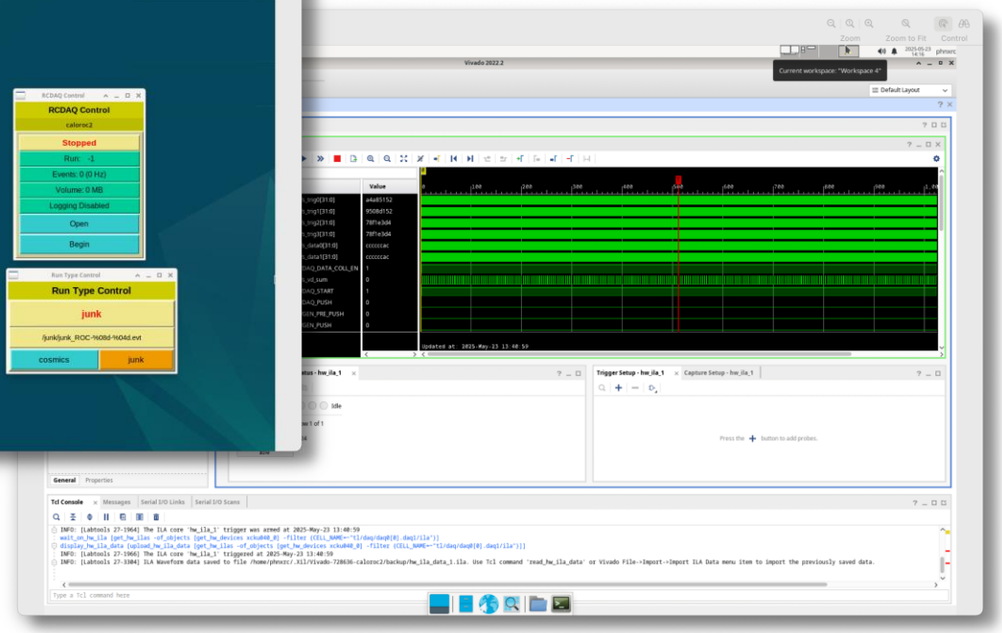


The currently active stuff

Finishing up the cable length test the way we started it



The RCDAQ Desktop



Vivado on yet another desktop

Back to RCDAQ for a moment…

This shows the relevant portions of the RCDAQ setup file

It loads the plugin that teaches RCDAQ how to read the H2GCROC3 front-end

And then we create

```
rcdaq_client load librcdaqplugin_h2gcroc3.so

rcdaq_client create_device device_h2gcroc3 1 12001 10.1.2.208 1 128
```

This is all that the plugin needs to know

```
$ daq_status -ll
caloroc2 -  Stopped
  Filerule:     /home/phnxrc/data/junk/junk_ROC-%08d-%04d.evt
  Logging enabled

. . .  Lines deleted

List of loaded Plugins:
   - H2GCROC3 Plugin, provides -
 -     device_h2gcroc3 (evttype, subid, IP addr, trigger, nr_packets) - readout an H2GCROC3
```

Analysis API

The device generates a packet with hitformat 301 (mnemonic “IDH2GCROC3”)

Remember these numbers are a simple enumeration, and numbers are plentiful

```
$ dlist /home/phnxrc/data/cosmics/cosmics_ROC-00000012-0000.evt
Packet 12001  7304 -1 (ePIC Packet) 301 (IDH2GCROC3)
```

The data access APIs are exactly like those for any waveform sampler we support (CAEN V7142, DRS4, the (sPHENIX) Sampa (== ePIC’s Salsa) chips…).

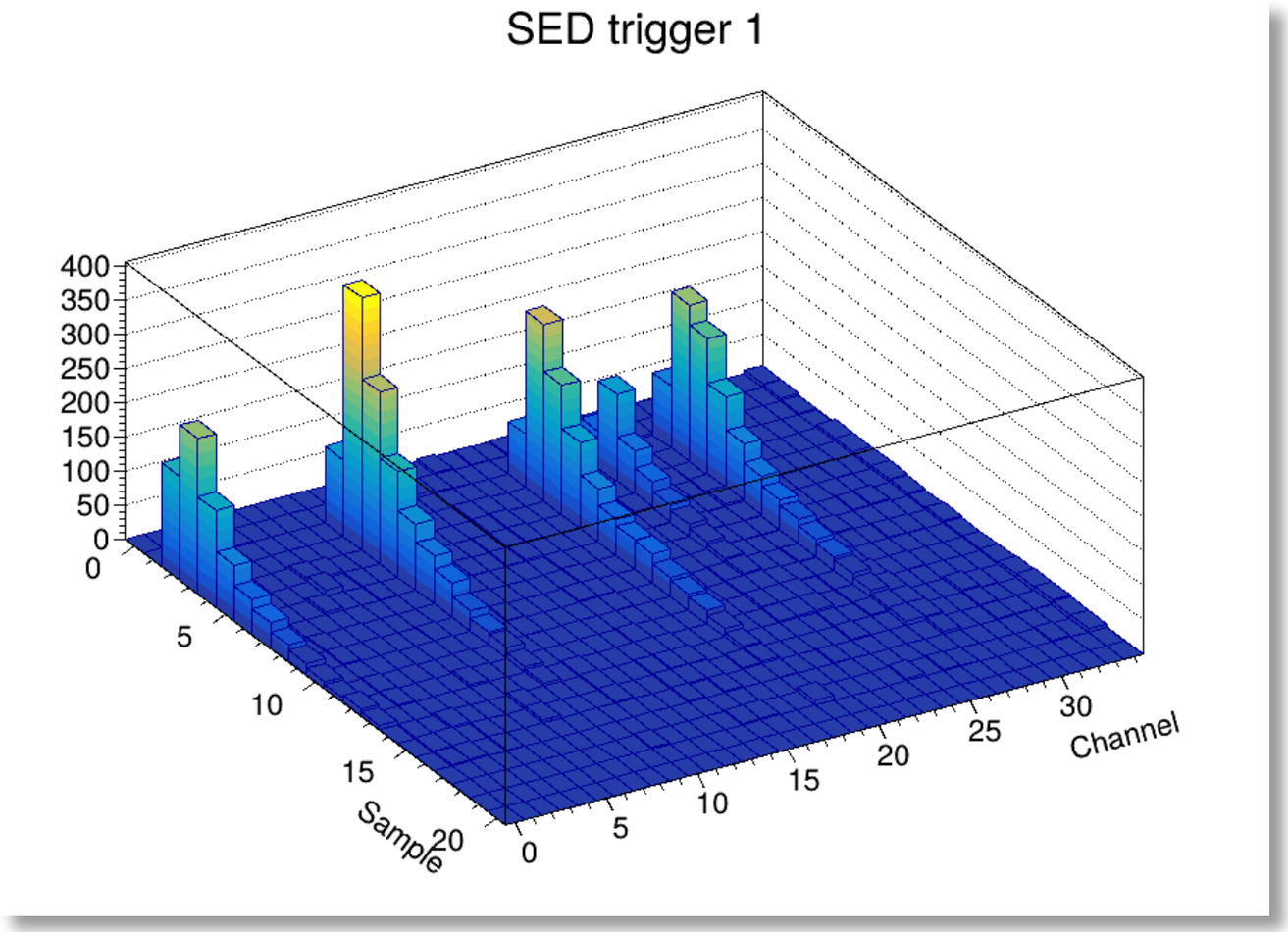
```
p->iValue(0,"SAMPLES")  tells you how many samples are present

p->iValue(0,”CHANNELS") tells you how many channels that device has  (here:144)

p->iValue(c,s)          gives you sample s from channel c  (there are faster interfaces available)
```

We we get away with (0,”SAMPLES”) b/c it’s the same for all channels. It supports different numbers of samples per channel

There are more APIs to ask the packet all kinds of other questions, but those are the most-often used ones here

The “mlp standard plot”

Whenever I have data from a waveform sampler (that is, a device with a number of channels, and each channel provides a waveform) I make a Lego plot with Channel vs Sample, with z the sample value:



Connected channels

We see what we expect: Signals from 5 tiles, and with the right channel numbers

The waveforms are baseline-corrected. And we saw the first of those plots from online monitoring on Thursday

The next step - Norbert is working on that

Norbert is working on a utility that retrieves *all* setup parameters form the front-end and saves them in a json file (I don't know the utility's name yet, I gave it the fictitious name "**`roc_save_json`**")

This json file can then be used to restore the settings exactly like they were at a given time

We (will) use this to store the entire setup in each data file's begin-run event for documentation purposes (and yes, you can easily extract and re-use it)

The first "device_command" device executes the command that saves the configuration to the json file

The second "device_file_delete" absorbs that file into begin-run packet 910, and deletes it after (so it's impossible to get a stale file).

```
rcdaq_client create_device device_command 9 0 "roc_save_json > /tmp/roc_parameters.json"
rcdaq_client create_device device_file_delete 9 910 "/tmp/roc_parameters.json"

rcdaq_client load librcdaqplugin_h2gcroc3.so

rcdaq_client create_device device_h2gcroc3 1 12001 10.1.2.208 1 128
```

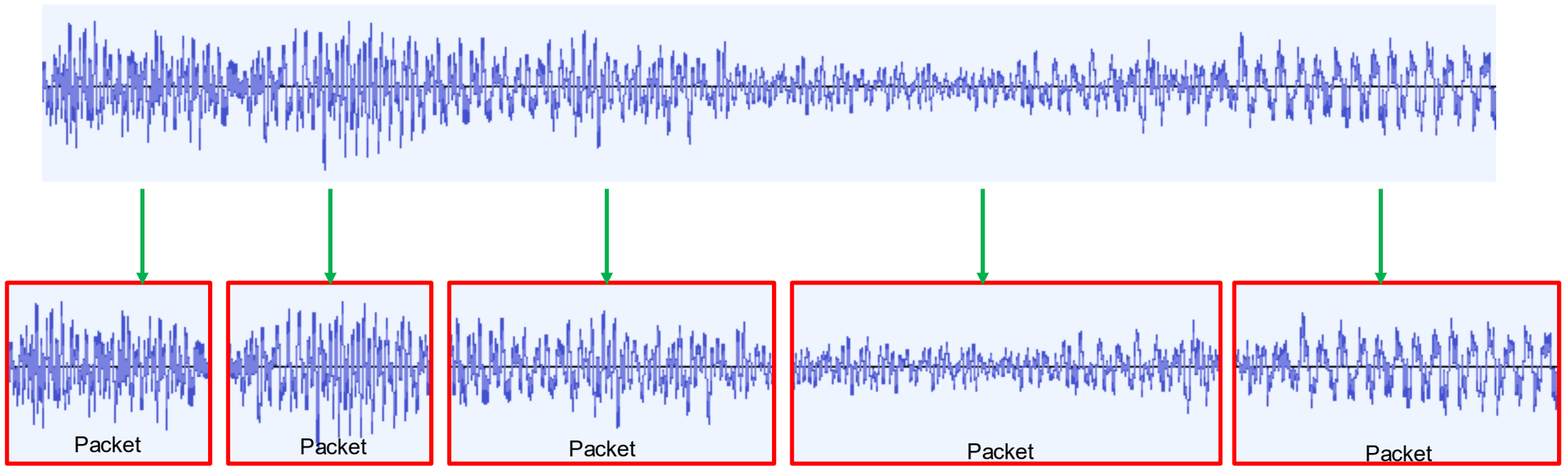
Analysis of existing rcdaq runs

With the current firmware we have no choice but to treat the data as “fully streaming”

Maybe not a bad thing – we might as well get used to the future

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



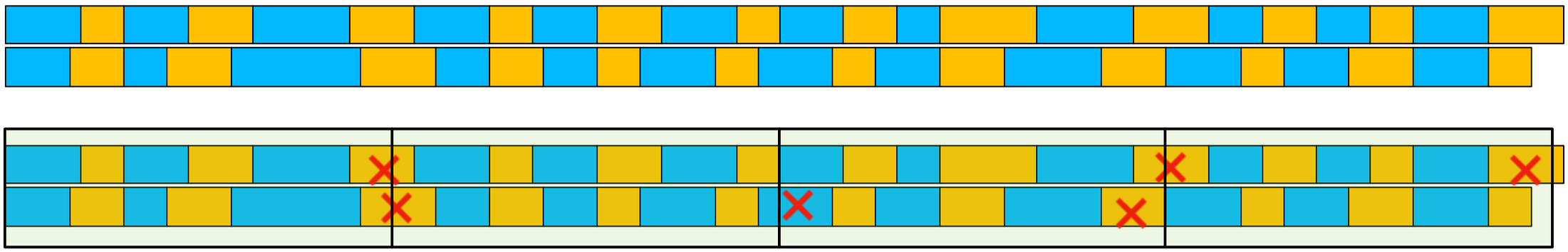
SRO packets chop where they chop

Streaming data have their individual channels' hit data interleaved

There is no place where a crossing's data (or even a hit) of multiple channels end

We simply say "stop" when a size limit is reached.

Some hits are cut off in the middle then



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

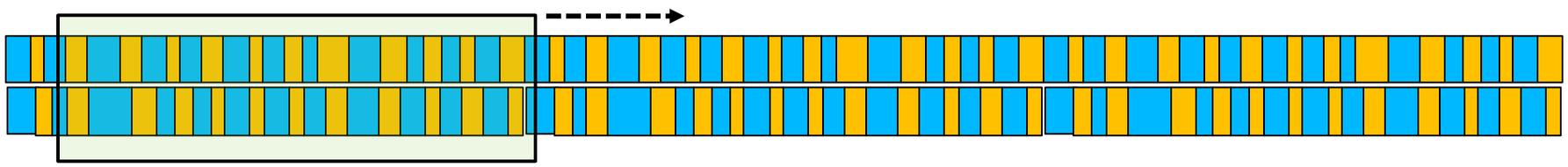
If you only look in a "packet purview", you will lose a few hits ("keyhole view")

And this is unavoidable no matter what you do! If the boundary is not a (however large) RCDAQ packet as it is here, then it's a file boundary, disk boundary…

Enter the h2roc2 pool

The pool is putting together again the pieces that were cut into packets, as if that packetizing had never happened

Like in my VOIP example, the chopped-into-packets waveform snippets are put back together into a longer timeline



The pool provides a sliding window across the data timeline large enough that all data from one crossing are in that window

Drop data that have been processed, fill in new data

Go through the streaming data without excessive memory usage

Instead of the “keyhole” actual packet, you use the pool instead

Keep throwing in packets until prescribed depth is reached

APIs are (by design) identical to the packet itself, so no API/programming change

Some eye candy

A quick movie of the lego display of consecutive events…