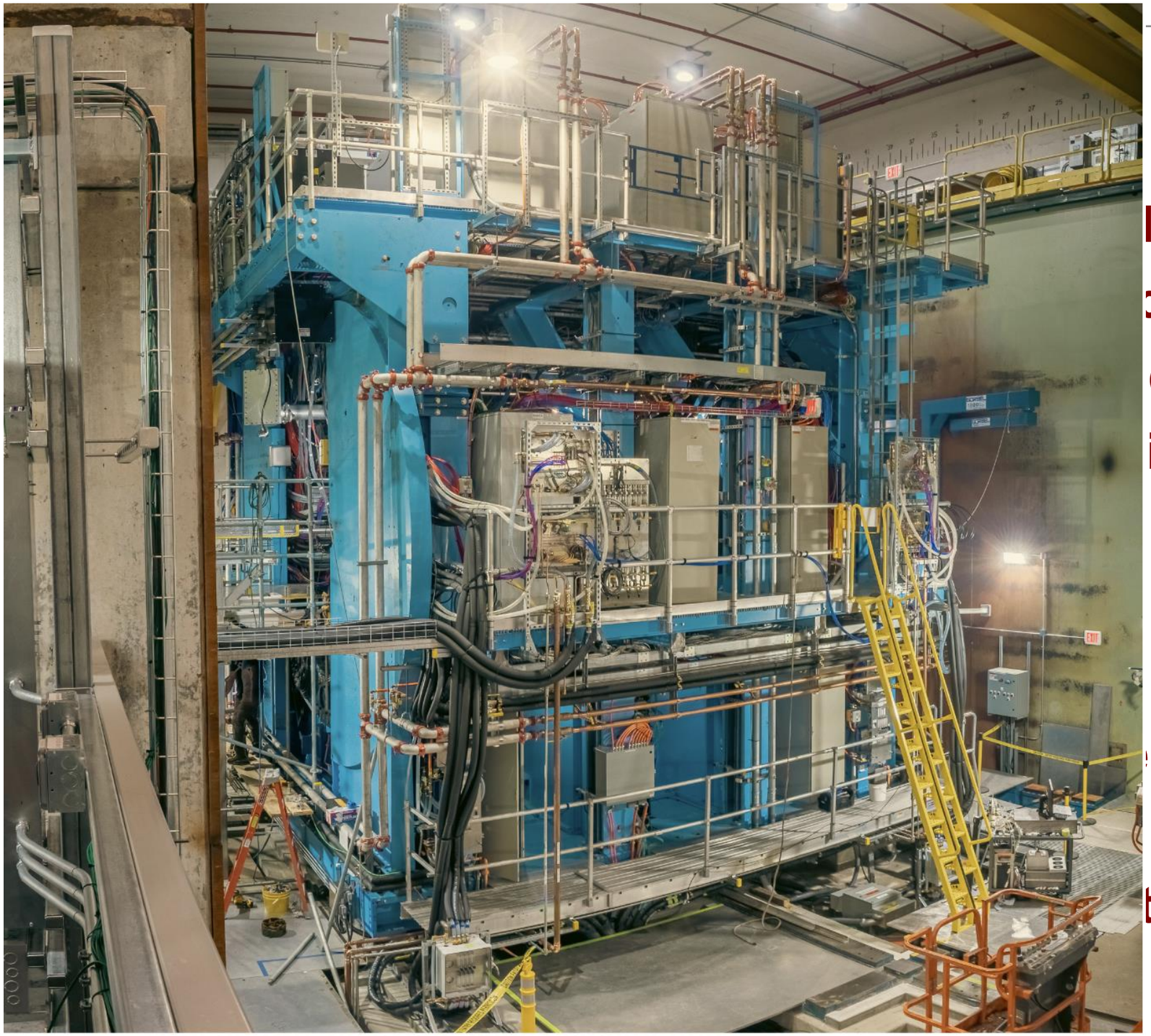
The Final Run of sPHENIX

Martin L. Purschke

Long Island, NY

RHIC from space

The sPHENIX Detector



lronic Calorimeters

ctromagnetic Calorimeter

e Projection Chamber (TPC)

imum Bias Detector (MBD)

rmediate Tracker (INTT)

tex Detector (MVTX)

Not shown: an event plane detector and a small timing detector

sPHENIX Runs

The plan had been for 3 years of running (2023, 2024, 2025)

- During the 2023 RHIC Run, on August 1, the accelerator developed a Helium leak that ended the 2023 run prematurely before we ever got to “physics” running
- 2024 was the polarized p-p run (with 3 weeks of Au+Au at the end, mostly commissioning)
- 2025/26 has been the long Au+Au run, p+p, and a few days of O+O

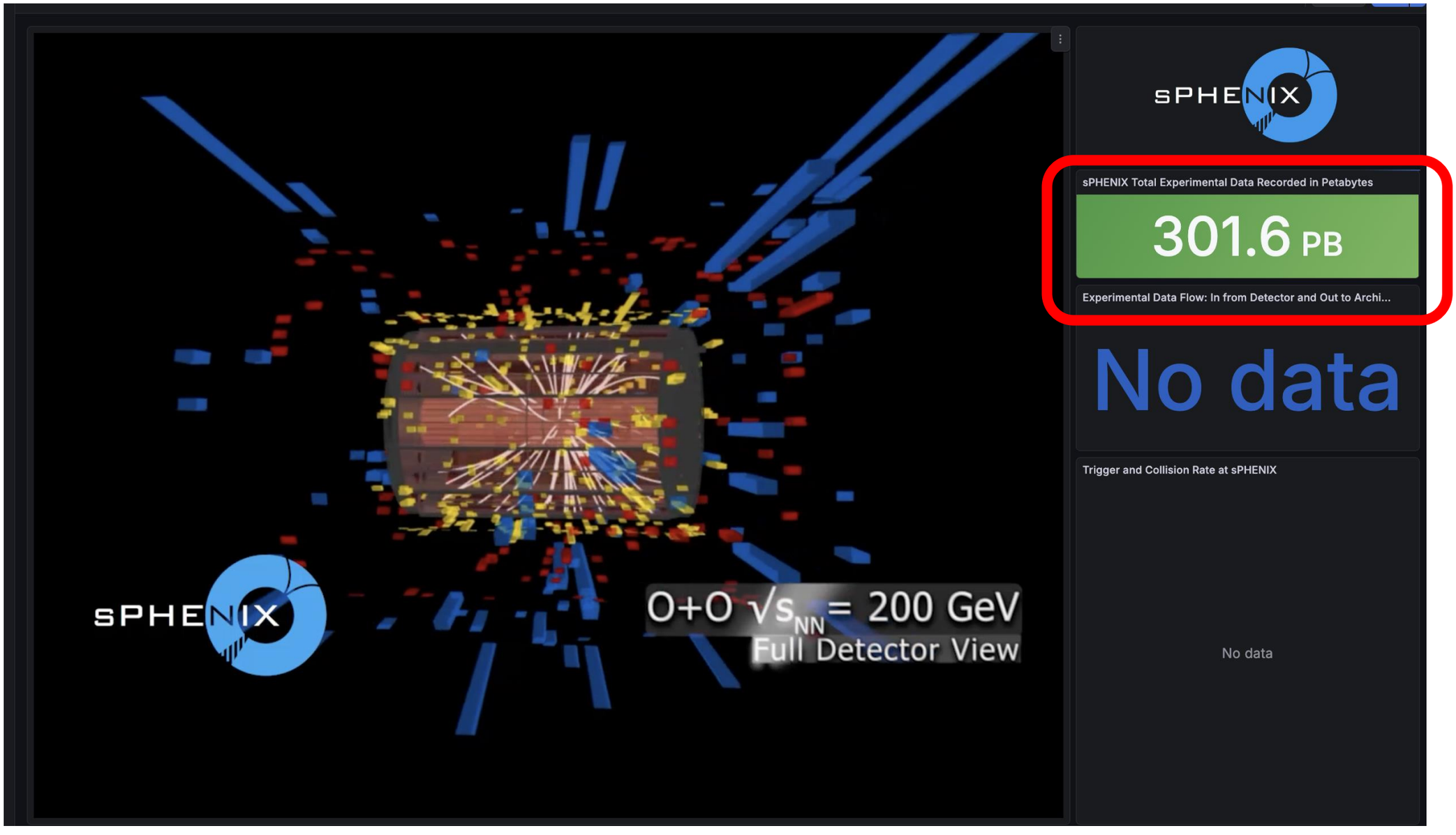
Both 2024 and 2025 runs have been a great success, despite some challenges

Our Streaming Readout played an outsized role in achieving our luminosity goals.

To jump right into it…

This is how we ended sPHENIX… Beam operations ended with ~297PB

We crossed the 300PB threshold with the post-beam calibration runs



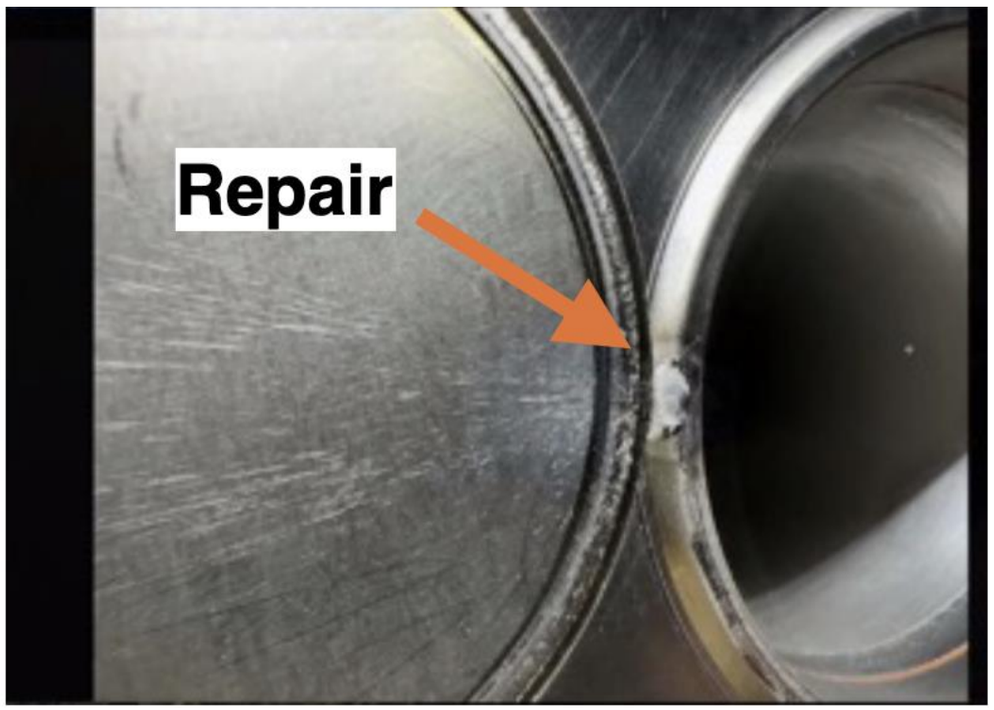
2025 challenges (many smaller ones, too)

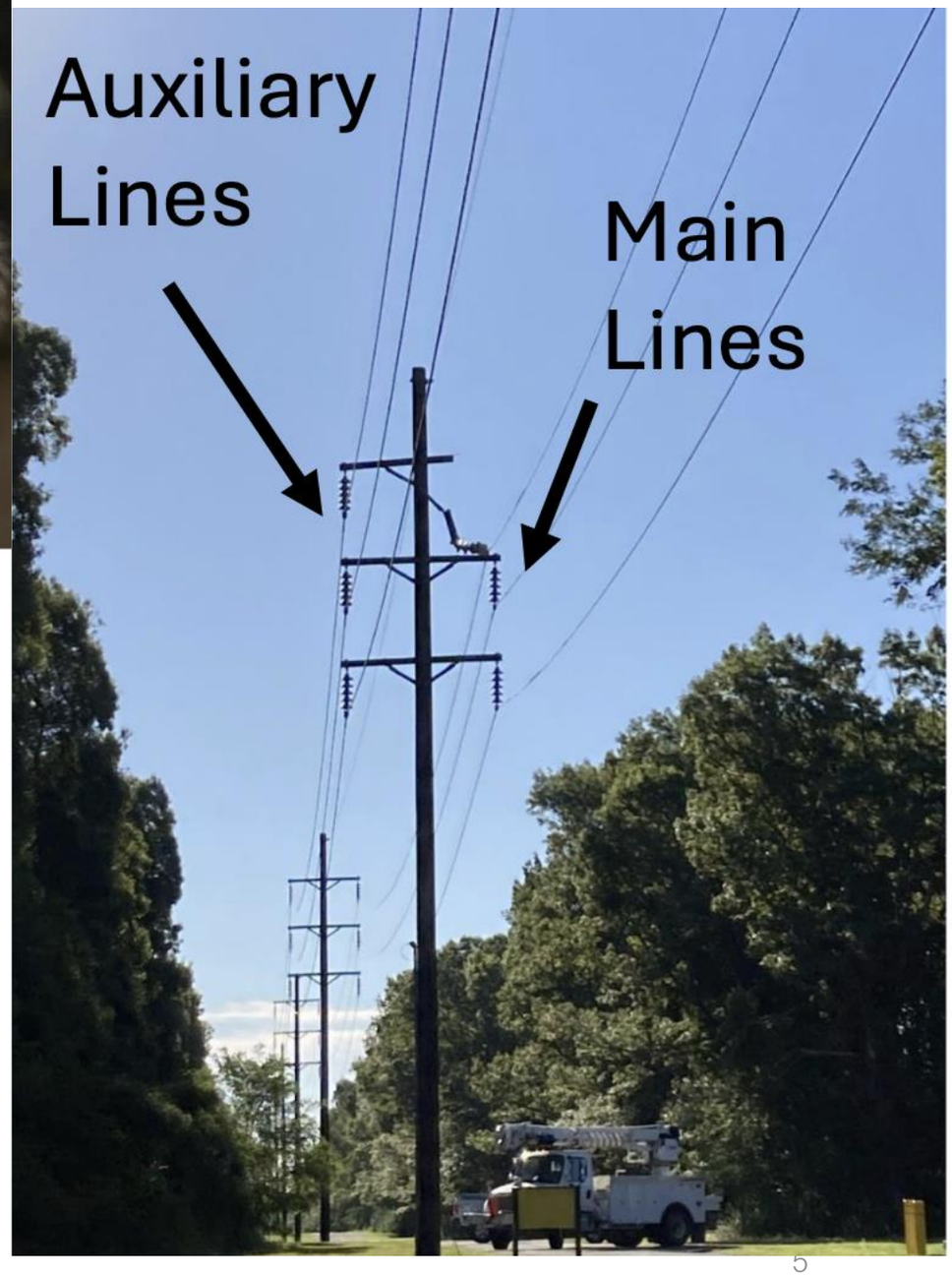
We had a few tense moments

A squirrel “visiting” a power bus

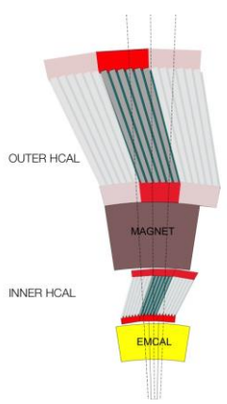
The main RHIC power feed broke an arm

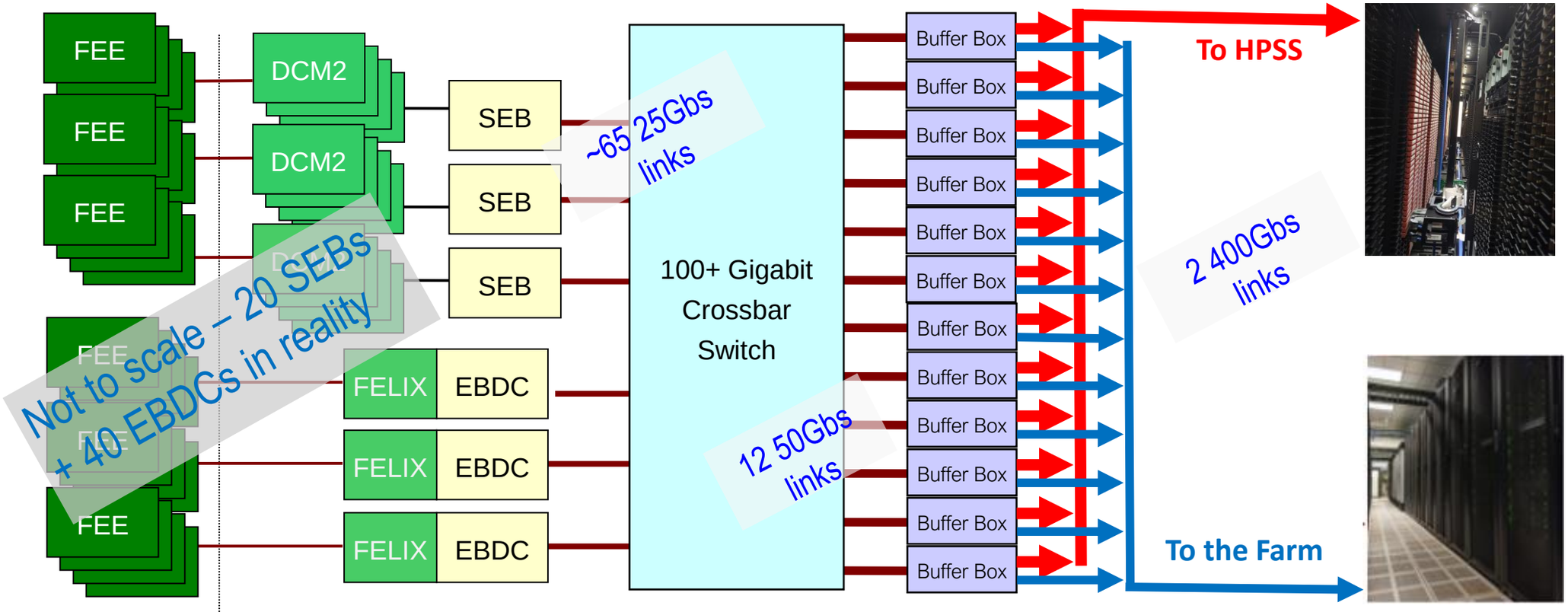
An abort kicker pre-fire sent the beam to the vacuum pipe wall and drilled a hole, exposed to air

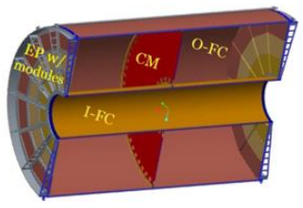




sPHENIX DAQ Birds-Eye View (2025)







On Detector **Rack Room** **Rack Room** **SCDF Computing Center**

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

For the Au+Au run, we upgraded from 6 to 12 bufferboxes to more than double our DAQ bandwidth

RCDAQ – our core DAQ system

RCDAQ has been around for about 13 years

It has been the workhorse DAQ system for many, many R&D efforts (sPHENIX ones, and also EIC-themed ones, also Medical Imaging, lots more)

I'm aware of about 30 test beam campaigns and countless lab setups using it

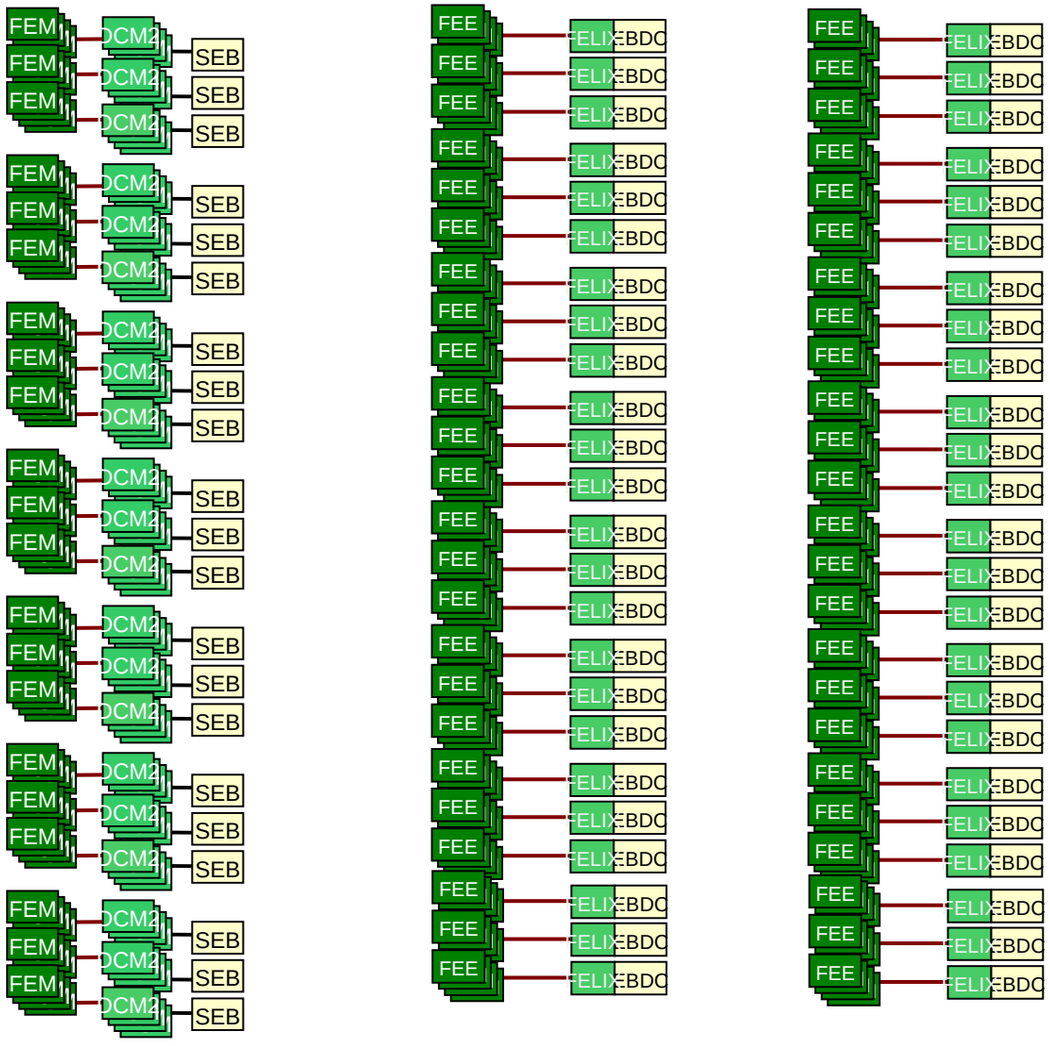
Support for the sPHENIX hardware (digitizers, but notably 3 different flavors of the ATLAS FELIX card)

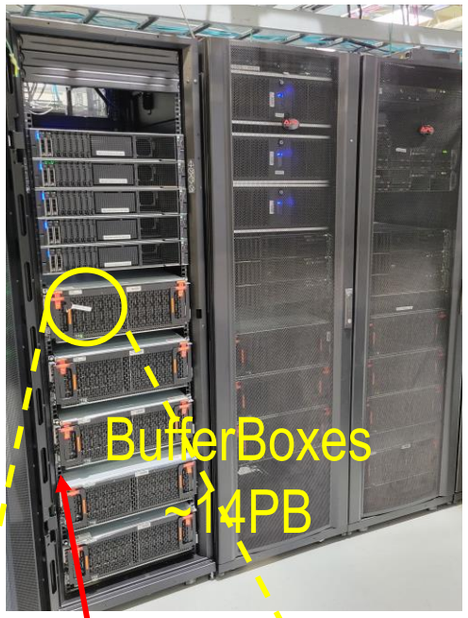
All knowledge how to read out a particular hardware is contained in RCDAQ plugins

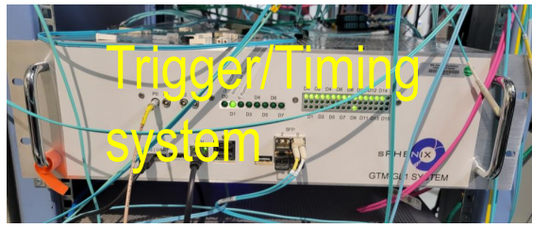
We are using many (84) concurrent RCDAQ instances ("cohort"), each reading out a particular part of the sPHENIX detector in a synchronized way

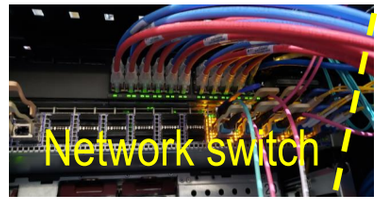
This "cohort" of RCDAQs is controlled by a super-process called "Run Control", gives consistent and coherent operations

Some of our DAQ gear at the experiment









It uses a synchronized “cohort” of 84 individual RCDAQ instances, each reading a particular part of the detector

Held together by “Run Control”, like a conductor keeping everyone in line

The Timing System keeps it all synchronized

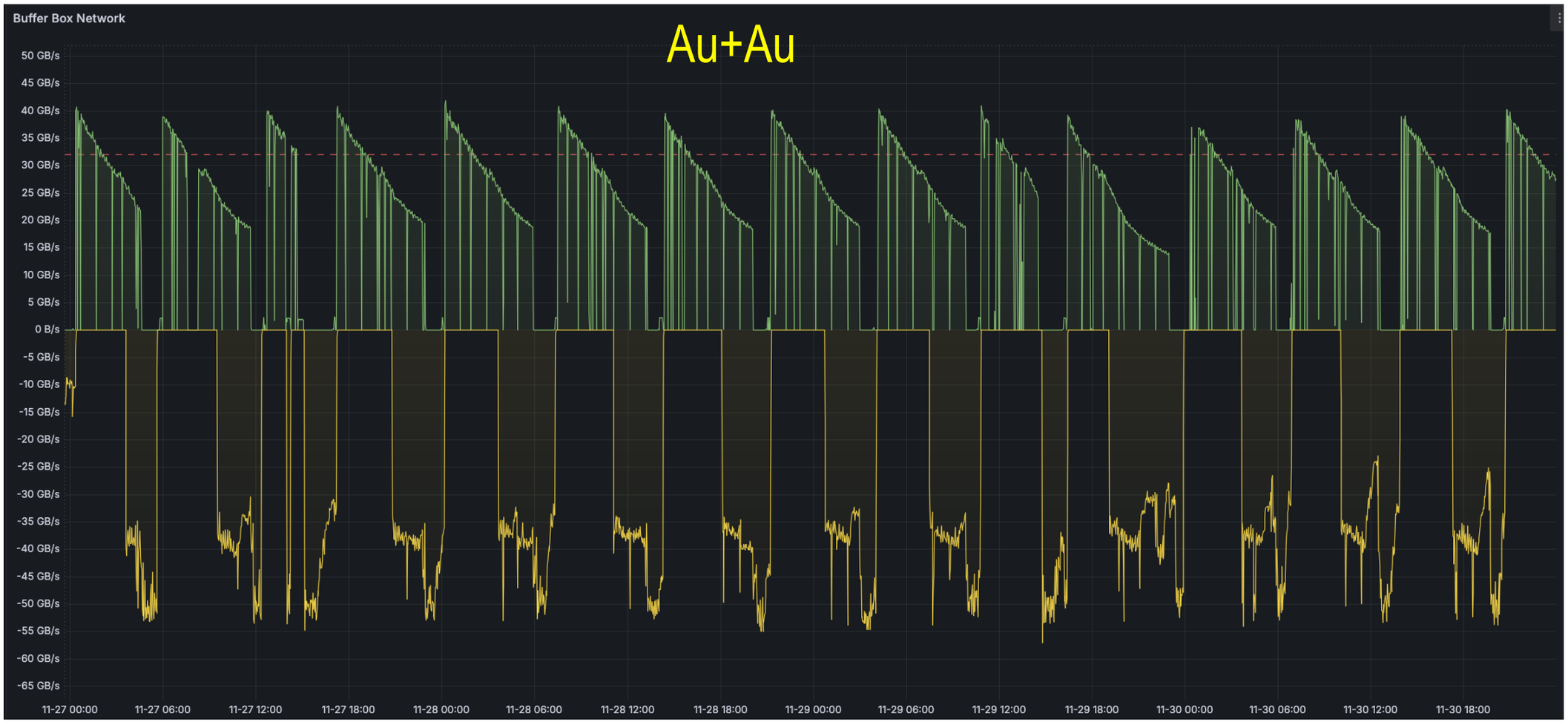
And here is what our investment of $339,278.22 got us

sPHENIX Data Taking

For Au+Au, 12PB in a week (~ 1.7PB/day) is a good yield, although not every week was like that

In Au+Au (different from p+p), we pretty much take all collisions we get

As the luminosity decays, there aren't more collisions to be had, so the data rate goes down



Incoming from the DAQ

Data sent to the SDCC/HPSS

We turned the transfers off during the highest-lumi periods

p+p is different

Here we get MHz worth of collisions and can always fill the bandwidth

We run the most selective triggers at high luminosity, and then increase the streaming fraction

Compromise to get data to the computing center fast for near-line monitoring

Streaming Readout in sPHENIX

Let me say this up front:

It is hard to overstate just how much easier everything is with streaming readout!

You don’t need to worry as much about event alignment

It just runs…

The Beam Crossing number (“BCO”) is what replaces an “event number”

In a distributed system as everyone has now, not as much concern about “missing a beat”

Streaming Readout in general

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 collision *n* is streaming in some places, in other places, data from collision *n-1* (or *-2*, *-3*…) isn't finished yet

Data are interleaved in this way. No need to wait for an event stream to be finished – do it offline. Just stream!

Bottom line: Streaming Readout is independent of how the front-end arrives at its decision to send up the data. Can be triggered, can be self-determined from the data… it's still streaming data.

How does RCDAQ support “Streaming Readout”?

RCDAQ itself is pretty much unaware of what kind of data it reads

It has a concept of “read the data that is offered and don’t care what it is”

It doesn’t really care (or even know) how the front-ends arrived at the decision to send data up

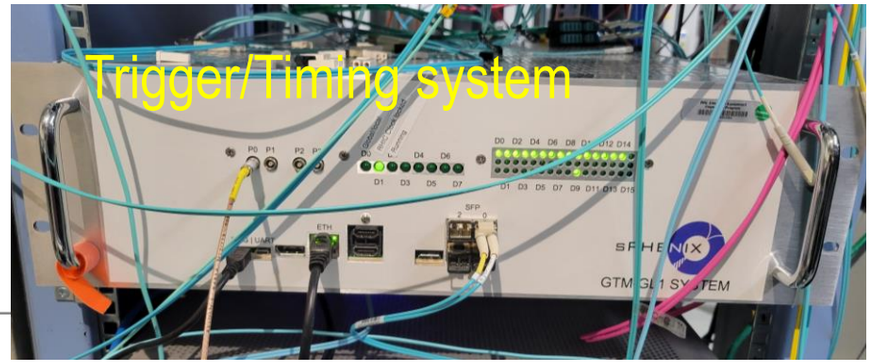
Triggered or streaming, no matter - when data arrive, they are getting stored

All the magic lies in the RCDAQ plugins that teach RCDAQ how to read out a given kind of front-end electronics

We use about 10 different plugins to read the different sPHENIX detectors

(There are *many* more to support readout hardware that you will find in your typical test beam – while I‘m here, I remotely support two test beam setups, one at CERN, one at Jlab)

The Timing System holds it all together



We picked a convenient multiple of the beam crossing frequency (x6)

We have a global 64 bit master beam-crossing (BCO) counter, lasts > 60,000 years

We transmit 40 of the 64 bits to the front-ends, those 40 bits go into the data stream

Some front-ends again pass 20 bits on to the FEEs for “micro-alignment” between FEEs

This data block (96 bits) is sent out for each RHIC beam crossing (every ~106 ns):

One beam crossing

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 BCO

Example – INTT (Intermediate Tracker)

The left column are BCO's that were triggered on

The right column shows the SRO data from (1/8th) of the INTT (not all triggers have data in every portion)

You can see the matching BCO numbers

		GL1 BCOs		INTT BCOs (intt0)		
1						
2				1/8 of INTT		
3						
4		0xb5483e942e19				
5		0xb5483e95c7a0				
6		0xb5483e96fdeb				
7		0xb5483e97aebd				
8		0xb5483e97f06a		483e97f06a		
9		0xb5483e984bc9				
10		0xb5483e99577a				
11		0xb5483e9d5b7c				
12		0xb5483e9ed179				
13		0xb5483e9f6181				
14		0xb5483ea13ed8				
15		0xb5483ea27e39				
16		0xb5483ea3cd4b				
17		0xb5483ea77d06		483ea77d06		
18		0xb5483ea8f086		483ea8f086		
19		0xb5483eadc6ff				
20		0xb5483eb0a0cc				
21		0xb5483eb70854		483eb70854		
22		0xb5483eb7c5ee				
23		0xb5483eb85507		483eb85507		
24		0xb5483eb92571		483eb92571		
25		0xb5483ebc503a				
26		0xb5483ed0dee3				
27		0xb5483ed33e2e				

40 bits

What's in it for us?

Our calorimeter systems are triggered. Full stop. Can't be changed.

But: There's *a lot* of physics just with the tracking system! Get whatever we can get...

When we trigger the calorimeters, we need to make sure that the tracking systems covers that triggered-on crossing.

Our TPC, for example, has a drift time of about 13 μs, call it 120 RHIC beam crossings

If you read out just those 120 crossings, then you have covered exactly that trigger crossing.

But add just one more (121), then you have also covered the one after the trigger

Add 2, cover 2 more crossings… you get the picture

We were able to stream *1000* crossings after a trigger – cover ~880 more crossings

If a new trigger arrives during that period, reset the count and start the 1000 again

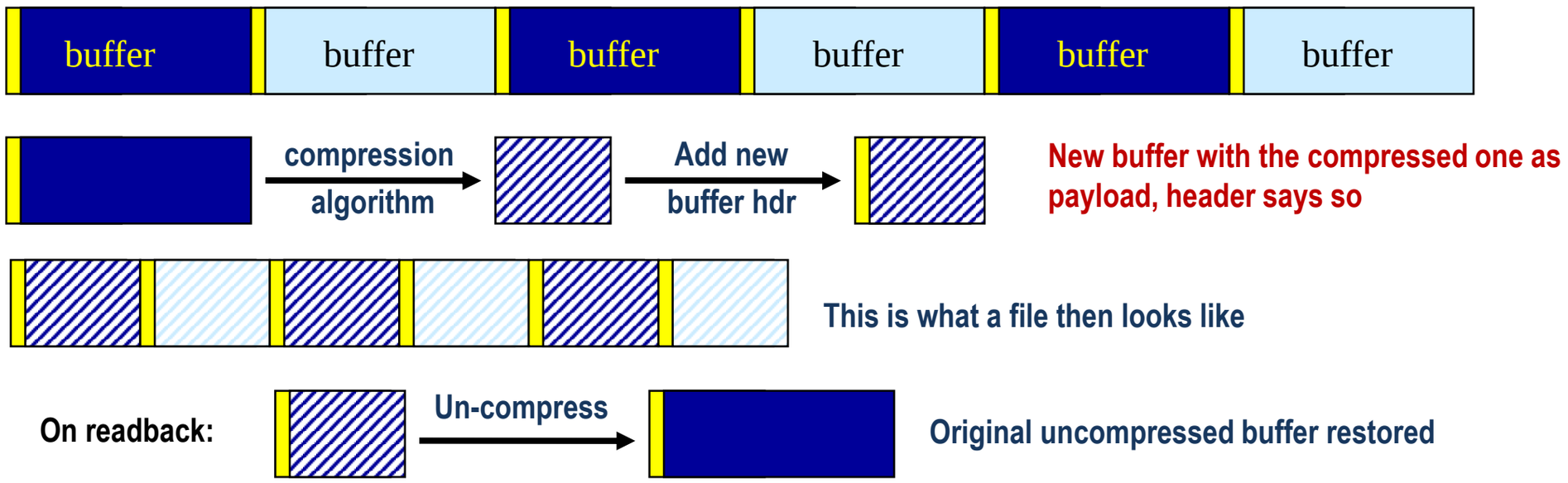
With high luminosity and a high rate of collisions, you get to *continuous streaming*.

Adjust the coverage window to what your bandwidth allows.

Multi-threaded Data compression

After all data *reduction* methods are applied, the data are still compressible (try gzip on your data file… you will be surprised…)

Our raw data format supports a late-stage data compression that works on an I/O buffer:



Our DAQ readout machines all have 96 CPU cores

We run a multi-threaded compression on the output buffers before writing

4 different compression levels to choose from - 3 LZO algorithms, and ”bz2”

We achieve compression yields around 60% - meaning 100GB become 60GB.

Better use of disk storage and also network throughput.

Compression Yields

For everything but the TPC, we have the CPU muscle to run the “bz2” compression, sort of the Holy Grail of compression

The original size of the buffer is in the header, to allocate enough memory at uncompression time… that makes it easy to see the compression yields:

```
$ prdfcheck physics_seb10-00068999-0000.prdf | more
buffer     1 at record     0 length = 32575780  3977 marker = ffffbefa  BZ2 Marker  Or.length: 67058744   48.578%
buffer     2 at record  3977 length = 32606651  3981 marker = ffffbefa  BZ2 Marker  Or.length: 67066472   48.6184%
buffer     3 at record  7958 length = 32602595  3980 marker = ffffbefa  BZ2 Marker  Or.length: 67066084   48.6126%
buffer     4 at record 11938 length = 32591621  3979 marker = ffffbefa  BZ2 Marker  Or.length: 67065432   48.5968%
buffer     5 at record 15917 length = 32596554  3980 marker = ffffbefa  BZ2 Marker  Or.length: 67063228   48.6057%
buffer     6 at record 19897 length = 32603688  3980 marker = ffffbefa  BZ2 Marker  Or.length: 67070792   48.6109%
```

The TPC has too much data to keep up; we can “only” run “LZO” here. It is also denser to begin with:

```
$ prdfcheck TPC_ebdc16_0_physics-00077980-0000.evt | more
buffer     1 at record     0 length =     7997   1 marker = ffffbcfe  LZO Marker  Or.length: 17640   45.3345%
buffer     2 at record     1 length = 361533375  44133 marker = ffffbcfe  LZO Marker  Or.length: 520119368   69.5097%
buffer     3 at record 44134 length = 363089288  44323 marker = ffffbcfe  LZO Marker  Or.length: 520112984   69.8097%
buffer     4 at record 88457 length = 363165726  44332 marker = ffffbcfe  LZO Marker  Or.length: 520110952   69.8247%
buffer     5 at record 132789 length = 362751108  44282 marker = ffffbcfe  LZO Marker  Or.length: 520100040   69.7464%
buffer     6 at record 177071 length = 362783445  44286 marker = ffffbcfe  LZO Marker  Or.length: 520112680   69.7509%
```

Overall we get a reduction to about 60% of the original size.

But what can the DAQ *really* do?

The numbers I showed so far are constrained what the accelerator and the front-ends deliver

I wanted to know what the DAQ can really do without those constraints…

Post-beam, with all detectors in top shape, I took some “break-the-DAQ” type runs

Switched off or turned down zero-suppression and generated the highest data rate the front-end can deliver (DAQ goes 100% busy, but we don’t care here)

70 Gbyte/s

72GB/s or ~580Gbs average over a day or so

Remember, that’s to disk

That gives almost 6PB/day

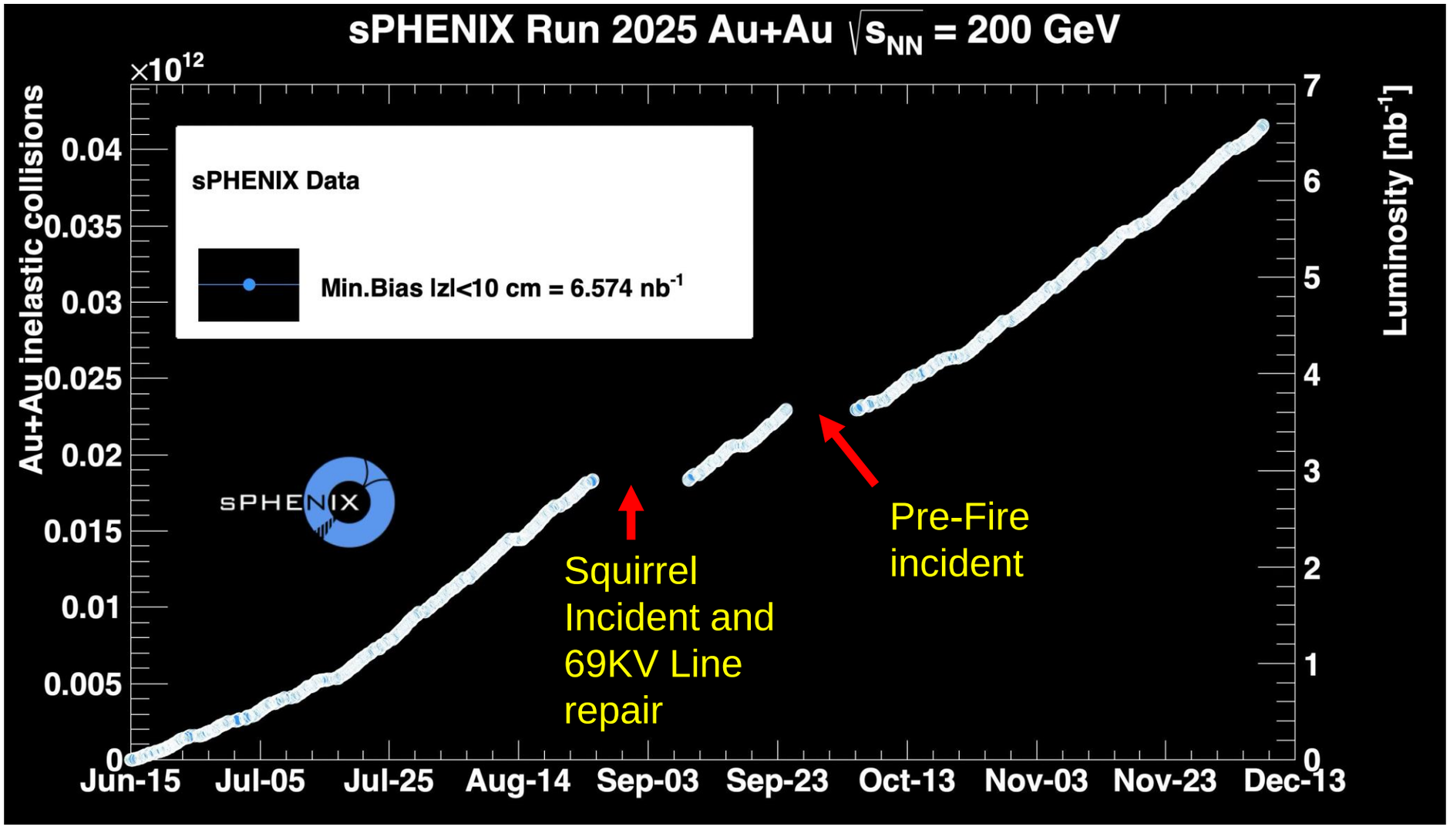
No issues, it kept running and running…

How we did in terms of our goals

182 days of Au+Au, 6.6/nb, 94% of goal

41 days of p+p, 16.7/pb, 130% of goal

5.3 days of O+O, 23.6/nb, 182% of goal



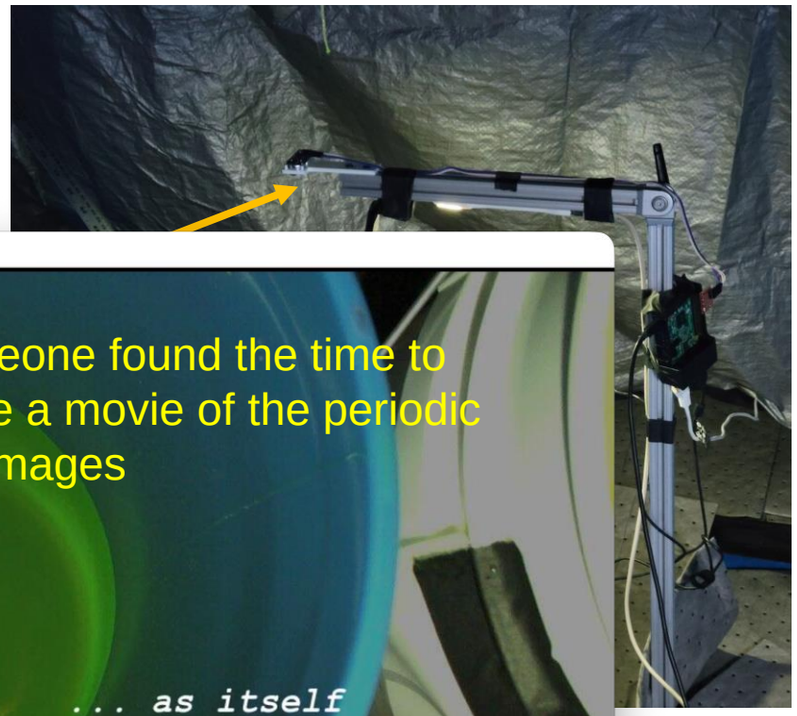
A bit of sPHENIX fun (with a serious background…)

My personal metrics of when an experiment has reached maturity: The experts have a bit of time on their hands for some playfulness…

In the last weeks of the run, the EMCal glycol cooling loop developed a small leak, inaccessible…

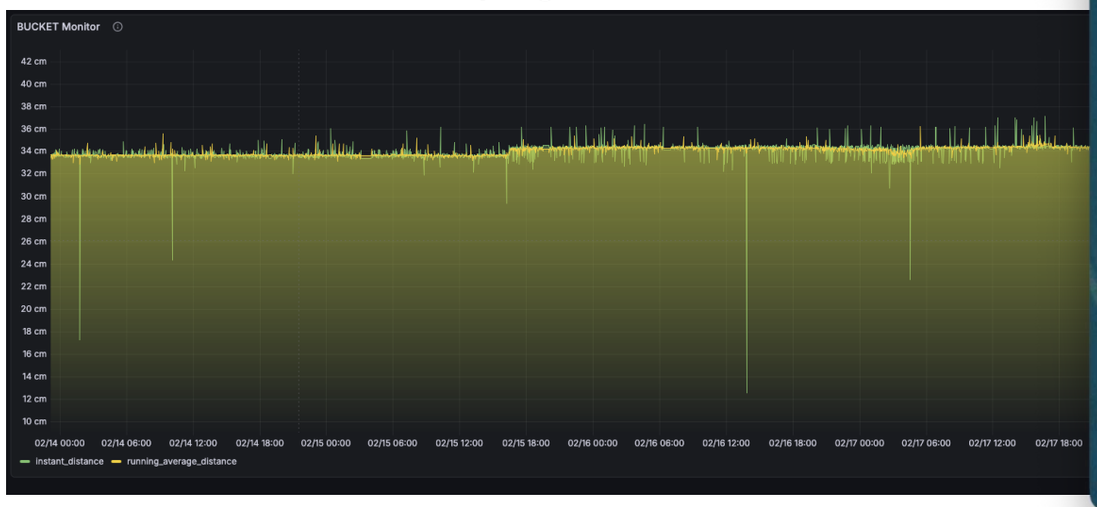
We used a rather low-tech bucket to capture the glycol seeping out from the low point of the HCal barrel

Rather than making a daily access to check on the level (leak might get worse), we promoted the BUCKET to the "13th sPHENIX subsystem"



I devised an ultrasound system to continuously monitor the glycol level in the bucket, also a came

We made a Grafana page…



… that allowed us to skip the level-check accesses

sPHENIX Approaching the End

Last collisions ever on Friday, February 6, 9:08

Post-collisions work until Feb 21 Midnight

Last shift crew

I was the one to turn the lights off…

Summary

We had good runs 2024 and 2025/26!

We got 300PB of data

The DAQ as designed and built can sustain 72GB/s (that's G**Byte**/s) of data

The Streaming Readout was a success story beyond our wildest dreams

Buying more hardware to double the DAQ bandwidth for 2025 was a great investment

I personally saw the first RHIC collisions in 2000, and was there for the last

And was the very last shift leader and turned the lights off…

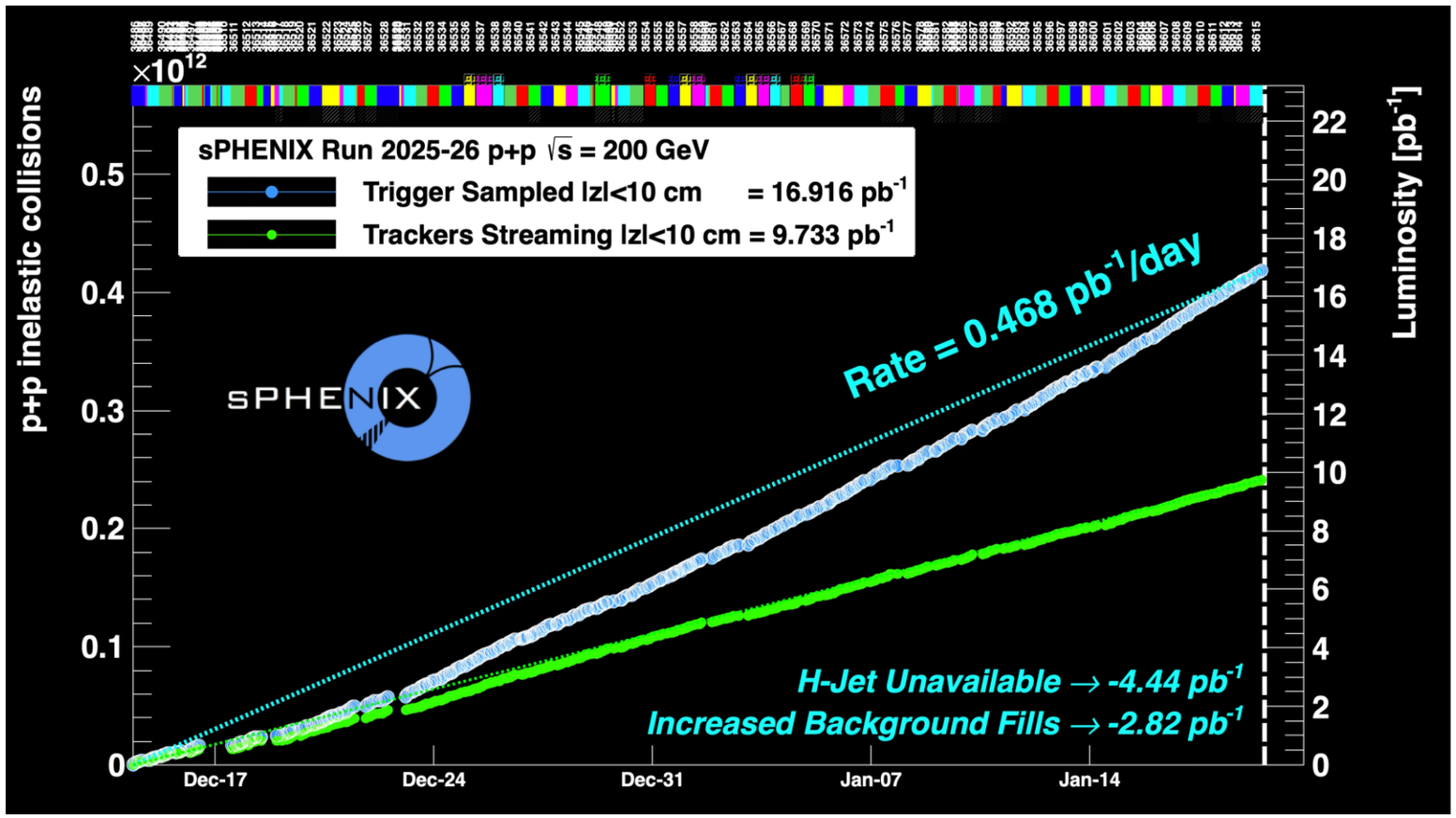
A bit of melancholy, but…

Looking forward to the Electron-Ion Collider!

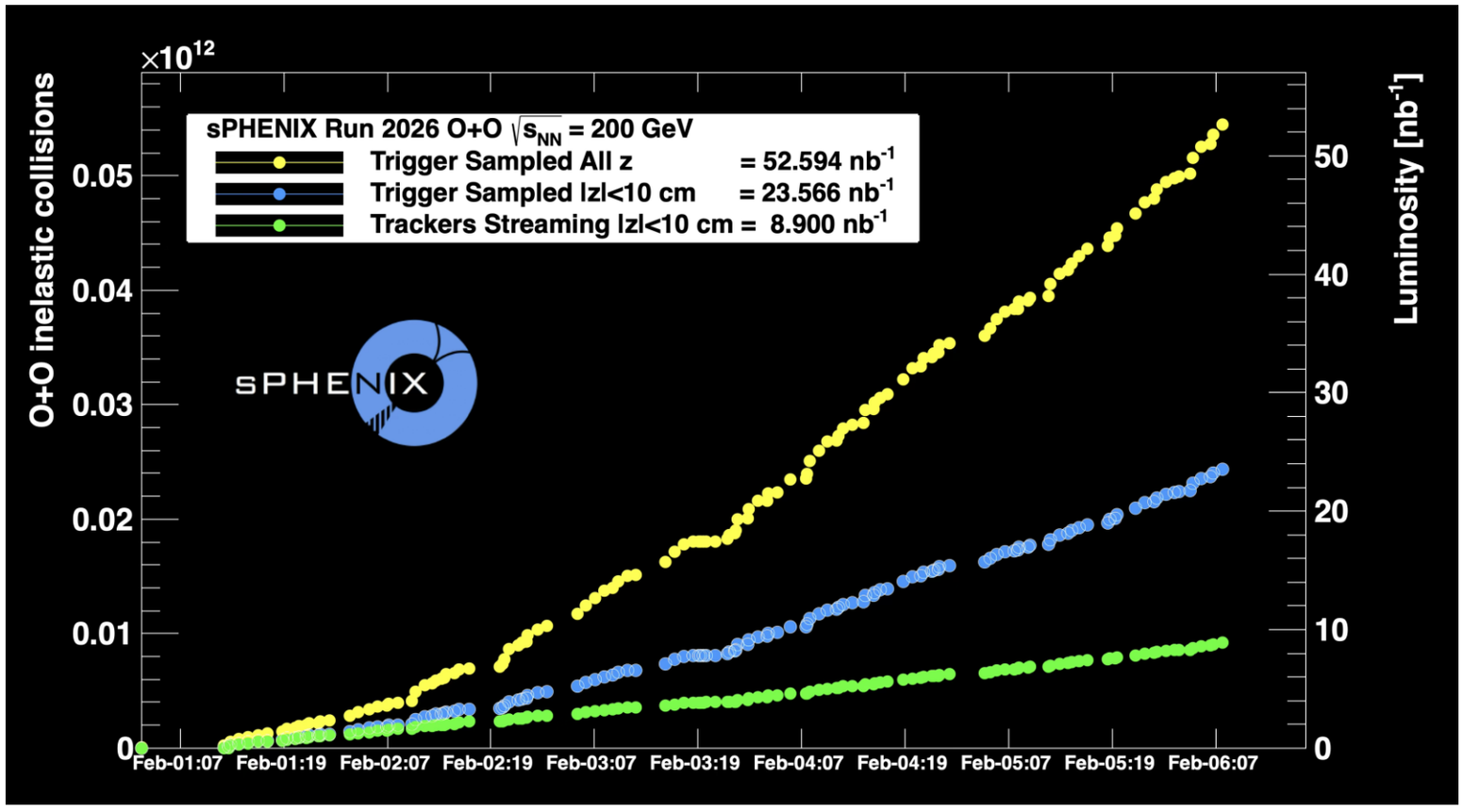
sPHENIX sign-off

The End

Performance p-p



Performance O-O



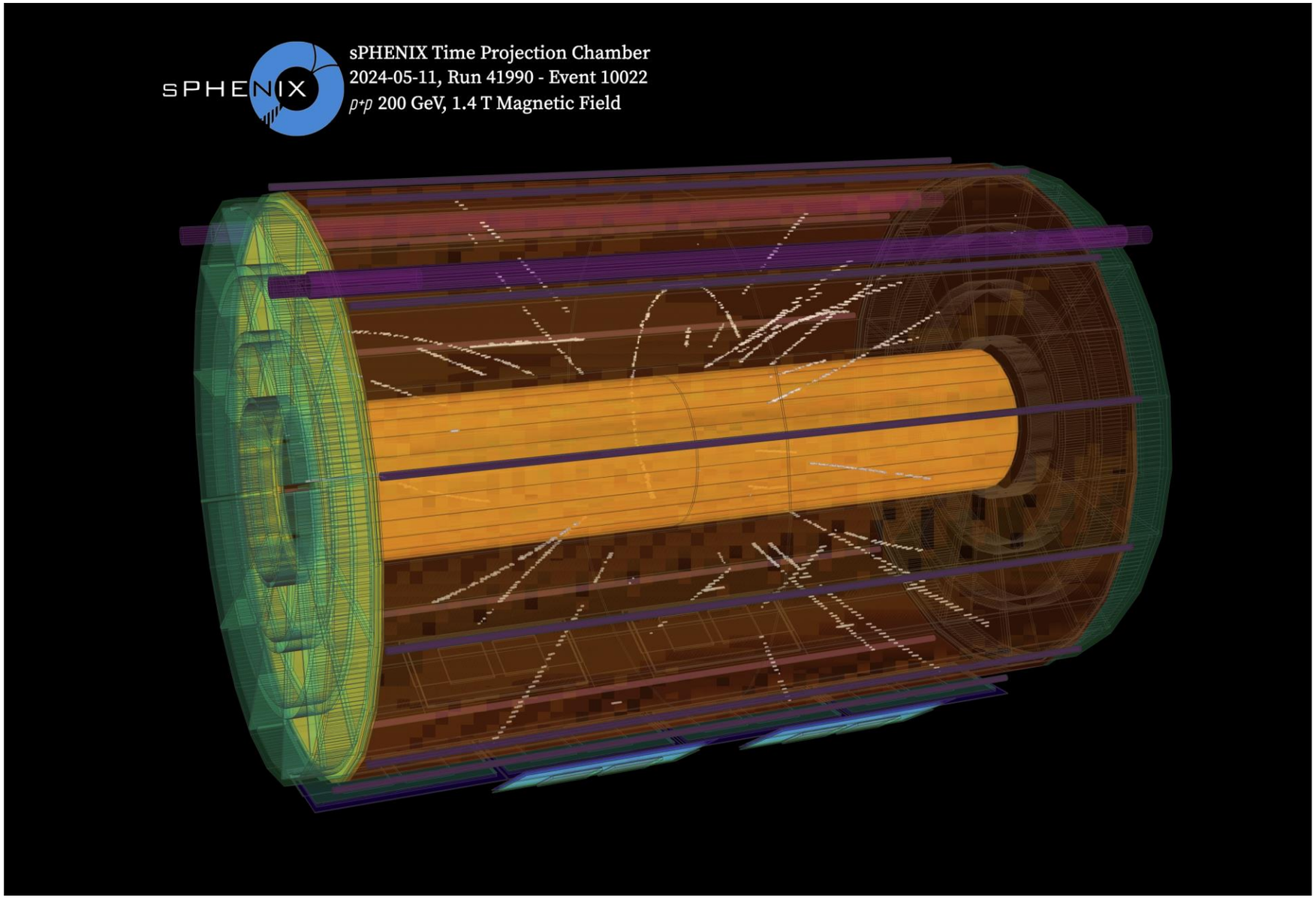
Generic Setup script example (TPC)

This is a heavily abridged setup script for our TPC (for one of 24 sectors) read out with ann ATLAS FELIX card

```
#!/bin/bash

H=$RCDAQHOST
[ -z "$H" ] &&  H=$(hostname)
MYSELF=$(readlink -f $0)

rcdaq_client daq_clear_readlist
rcdaq_client daq_set_eventformat 0


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

rcdaq_client load librcdaqplugin_dam.so

rcdaq_client create_device device_dam 1 4${H:4:2}0 1 $NUnit 1 /dev/dam0
rcdaq_client create_device device_dam 1 4${H:4:2}1 1 $NUnit 1 /dev/dam1
```

Many more settings, rules to create file names, DB whereabouts, compression and file rollover setting, and more

Let me show a wonderful event display…



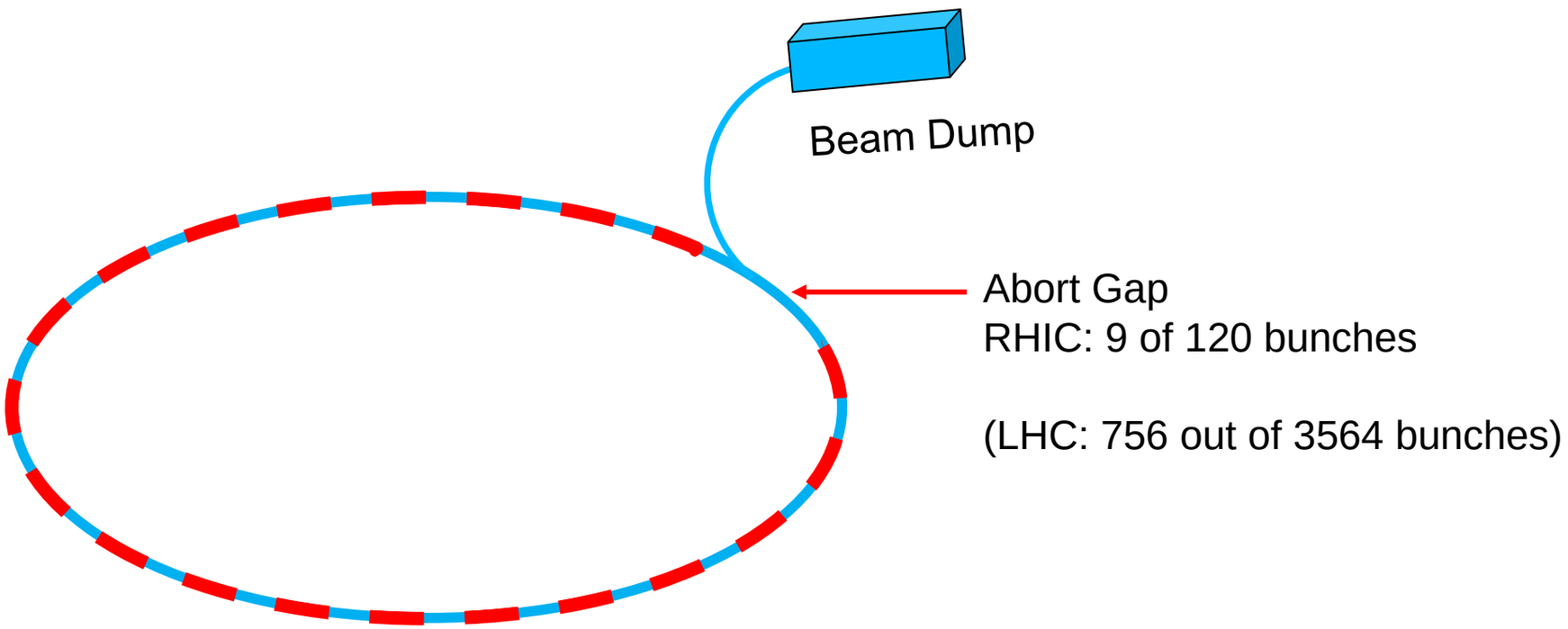
There are many more at https://www.sphenix.bnl.gov/EventDisplays

Bunch number

Our Trigger/Timing system needs to be aware of the “RHIC rotation”

Each collider needs to have an ”abort gap”, empty bunches where we can flip a switch and “derail” the beam to a beam dump

The abort gap defines the “front of the bunch train” where we can flip a switch and divert the beam to the beam dump when done



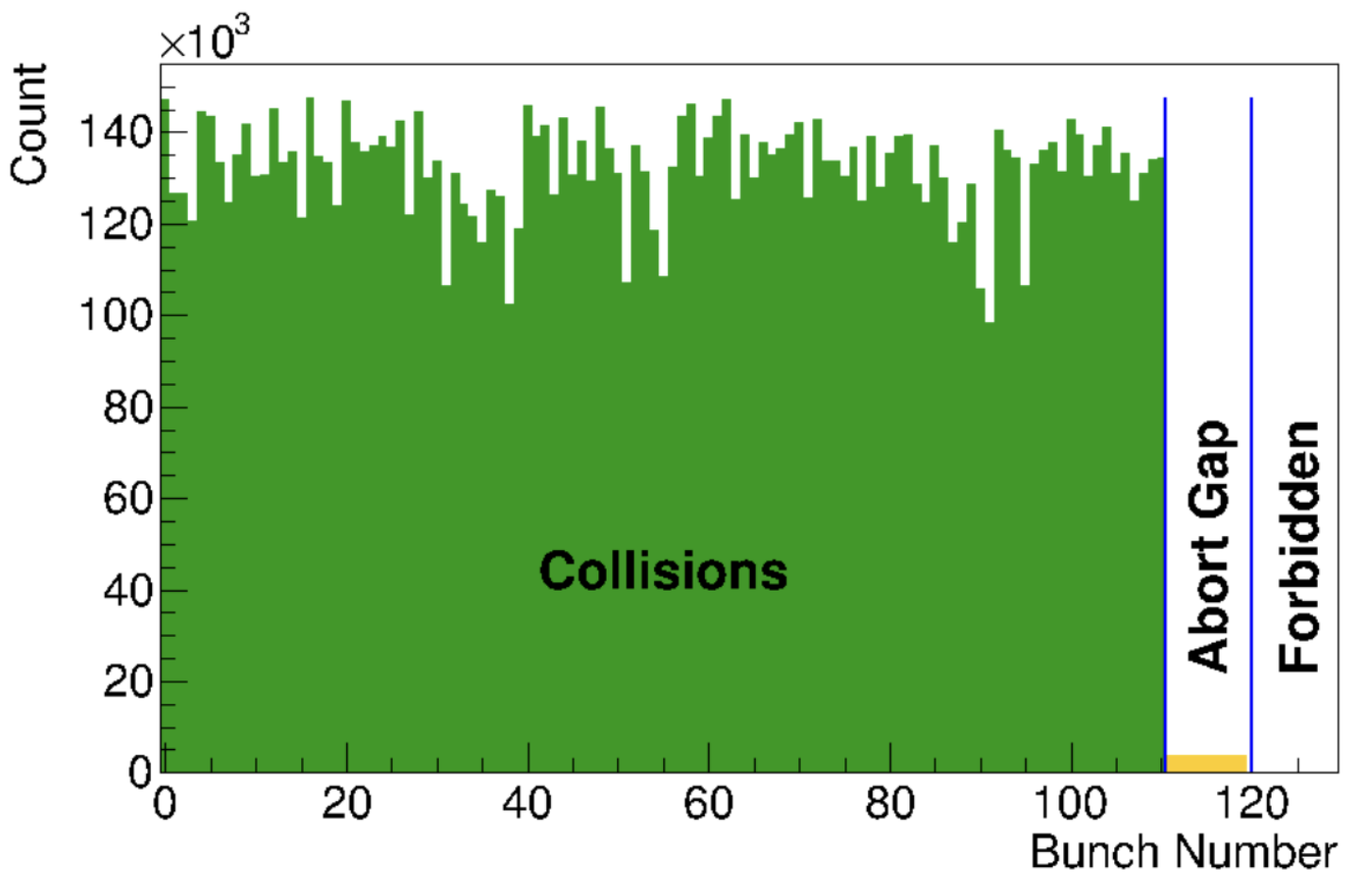
Bunch number

For us, the abort gap provides the opportunity to schedule calibration events where no collisions happen

For next year's polarized-proton running we *must* have the bunch number right – each bunch is special with polarization and number of protons

This shows that we are sync'ed up with the rotation!

We meticulously check (online monitoring plot here) that the abort gap is in the right place, meaning that the accelerator interface told us the truth



Accelerator rotation / “modebits”

RHIC has 120 bunch “buckets” – one full rotation of about 12us

The vGTMs know (from the v124) when “bunch 0” is passing buy

Each vGTM has 32 “banks” with information for 120 individual bunch crossings each

We can store detector instructions (“modebits”) for any crossing and per vGTM (so you can send different instructions to, say, the TPC and the EmCal)

That is the way to make a frond-end do things exactly when we need it to

They are detector-specific special instructions such as “reset all your internal counters”, “prime yourself for taking data”, “start”, etc

All modebits but a tiny handful are empty - “just the clock and nothing else”

Better explained with an example…

Modebits

Here is the (really simple) setup for our calorimeters using 5 such 120-slot banks

(remember each bank represents one RHIC turn, about 12us)

When we start, we issue a “run” modebit, more accurately “prime yourself”

Later we issue a “reset” modebit, clearing all internal counters - now we are ready to go

```
# modegrp repeat jump target values
 0         100    0    0
 1           0    0    0       9:RUN;
 2         100    0    0
 3           0    0    0       9:RESET;
 4      120000    1    4
```

0 – we make 100 turns doing nothing

1 – in bucket 9 we issue “run”

2 – 100 turns / 1.2ms nothing

3 – bucket 9 sends “reset”

4 – 120,000 turns of nothing, followed by a jump to bank 4 again, going on forever

What you see here is a one-shot init section in banks 0..3, followed by sending ”nothing” indefinitely (until we say “stop”)

LED etc calibs

Now it's easy to see how calibration events are triggered, and the readout is started.

We place the LED firing into the abort gap, where the detector is quiet

From there it's a calibrated propagation latency until we need to trigger the ADC readout (here: 29 buckets)

```
# modegrp repeat jump target values
  0          100     0     0
  1            0     0     0       9:RUN
  2          100     0     0
  3            0     0     0       0:RESET
#  757 is about 10ms
  4          757       0       0
  5            0       0       0    119:PULSE
  6            0       1       4     28:FA
```

0…3 you already know

4 – we wait 757 turns - ~10ms

5 – "PULSE" fires the LEDs in bucket 119 (last bucket in abort gap)

6 – 29 beam crossings after we start the readout (FA = "Forced Accept")

… and we jump back to 4, so we get ~100Hz of LED events.

… again until we say "stop"

And you can see what this does

```
# modegrp repeat jump targe
  0          100     0     0
  1            0     0     0
  2          100     0     0
  3            0     0     0
#  757 is about 10ms
  4          757       0       0
  5            0       0       0     119:PULSE
  6            0       1       4      28:FA
```

```
# modegrp repeat jump target values
  0          100     0     0
  1            0     0     0       9:RUN
  2          100     0     0
  3            0     0     0       0:RESET
#  757 is about 10ms
  4          757       0       0
  5            0       0       0
  6            0       1       4      28:FA
```

This is the setup for a pedestal run, everything as before, just no LED firing

Summary

The GL1 core, and by extension the vGTMs know the “bunch number” that is currently flying by

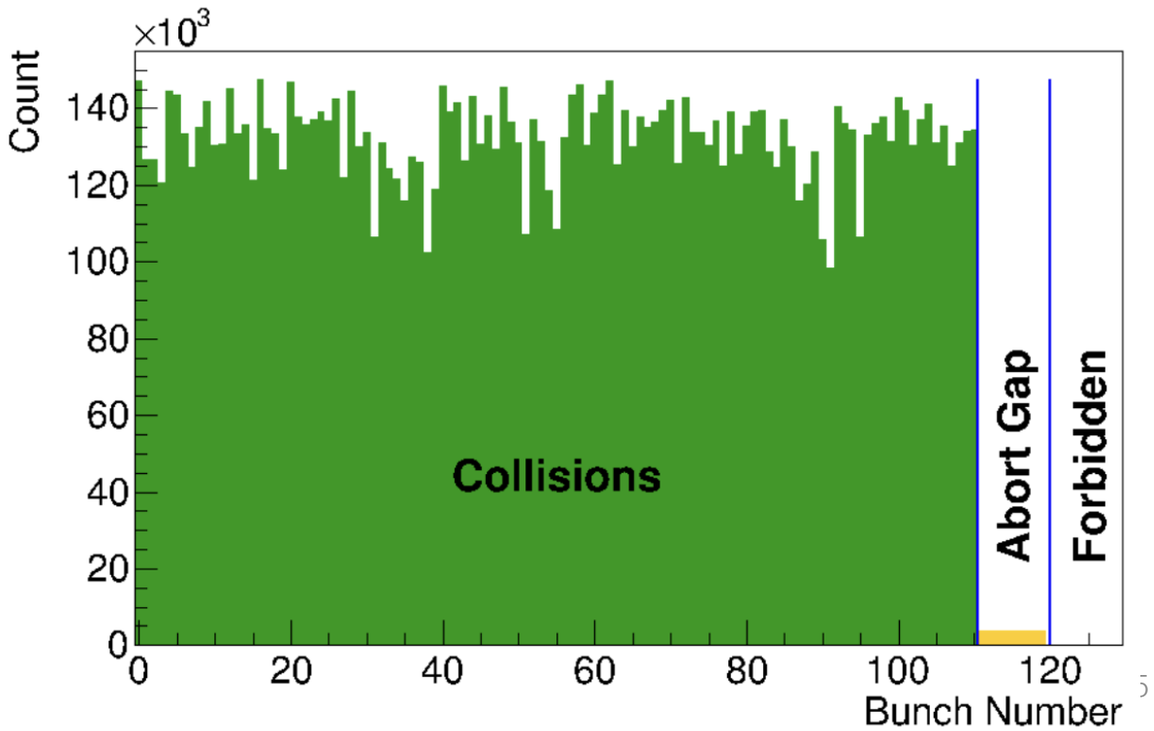
They count bunches 0…119, 0…119, and so on

That allows us to place special actions such as the LEDs firing into the abort gap when the detector is quiet, and schedule the detector readout at the right moment after

We can do that on a per-detector basis (latency of the HCal LED->signal is different from the emcal)

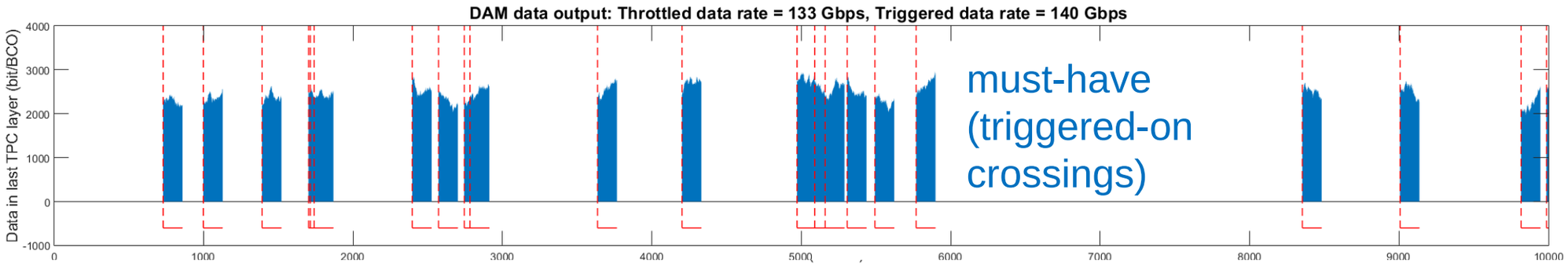
Important b/c even though we take dedicated LED runs, we need to do this with beam in the machine.

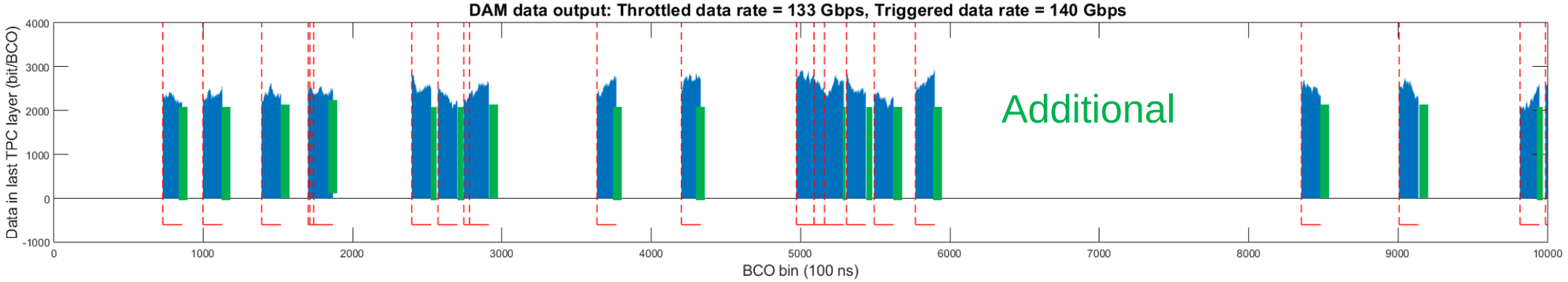
We meticulously check (online monitoring plot here) that the abort gap is in the right place, meaning that the V124 told us the truth

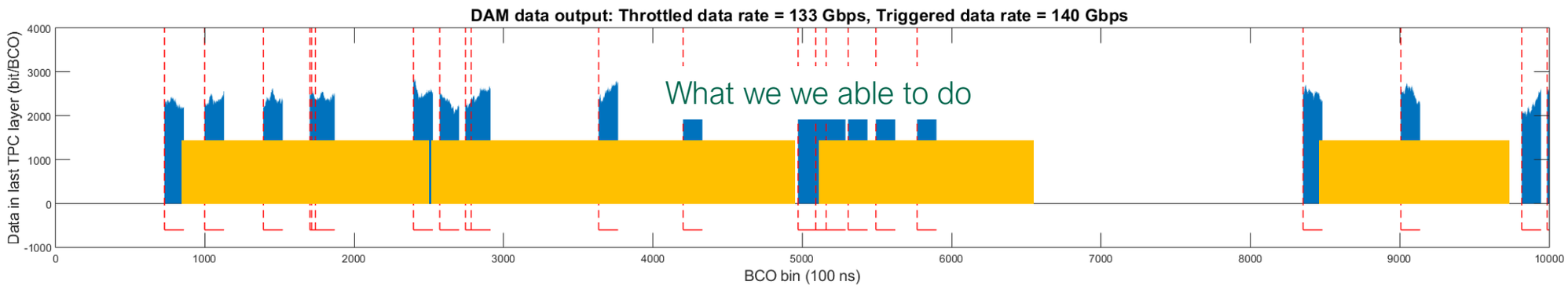


Let me show you

We initially assumed that we could add another 50-100 crossings that way







Example: (older – 2021) sPHENIX TPC data

Clock values embedded in FEE data

40 bits BCO

```
0000000 feee ba5e 0ff1 0001 7229 f7a0 0088 0004   ← FELIX Hdr
0000020 002f 8782 0004 ffff 0081 0000 0050 0050
…
0001020 d72c 0081 feed 0000 0088 3e2b 0004 feed   ← FEE structures
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
```

Clock values

```
bx 9f7a0
bx 9f7a0
bx 9f7a0
bx 9f7a0
…
```

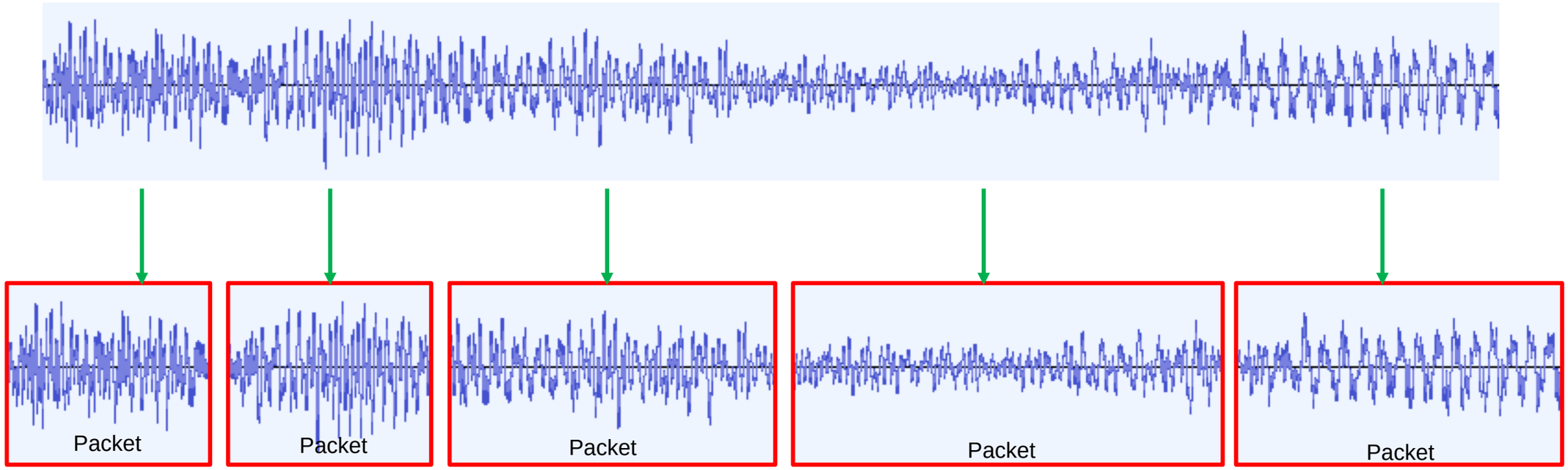
In this way you can verify the integrity of the internal data structures, and sort the data by “time”

I’m showing an older version here since it’s easier to see

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



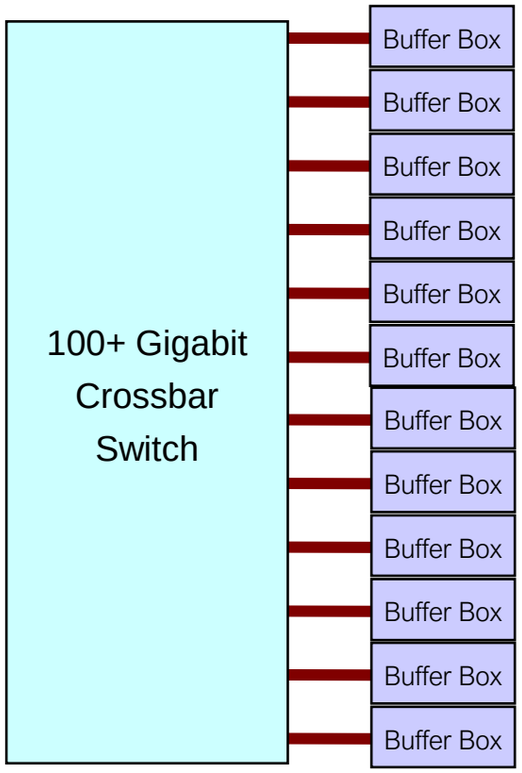
We are chopping the streaming detector data into conveniently-sized packets for storage

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)
$
```


38

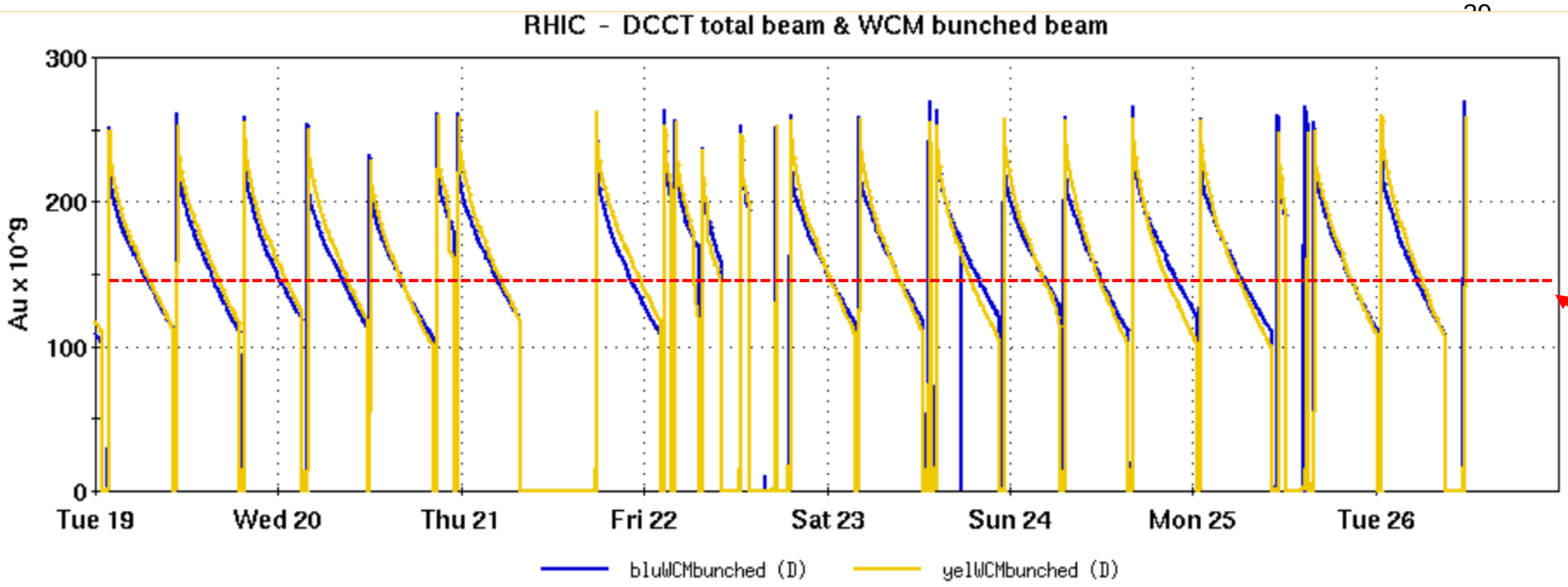
Why do we call those “BufferBoxes”?



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

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

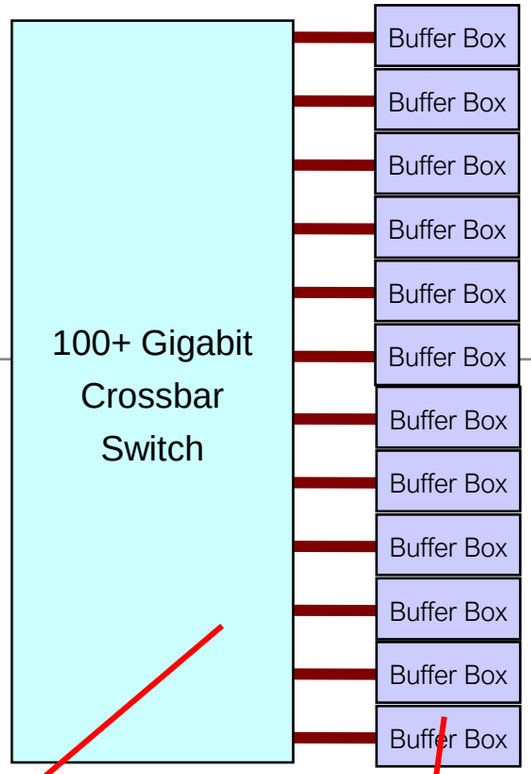
This Buffer boxes allow us to send the average, rather than the peak rate through the WAN

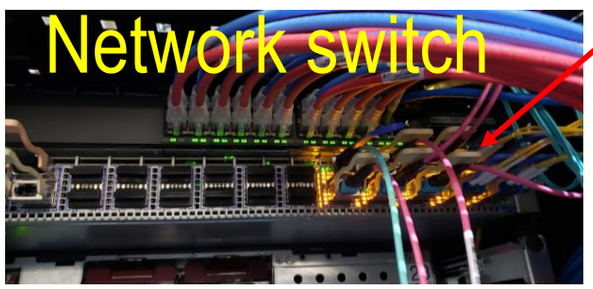


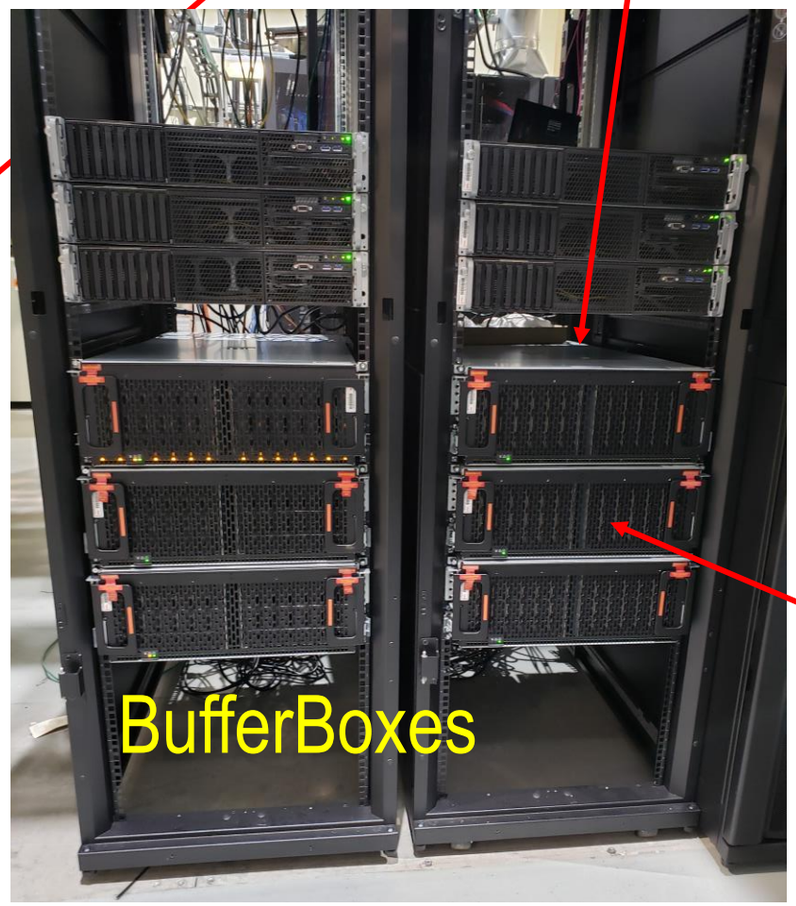
2016 (last PHENIX run) beam intensity over a week

Average

Some Pictures







Lustre (and Zeph)

Remember, RAID allows you to “stripe” your I/O across multiple disks

You gain throughput (and data protection).

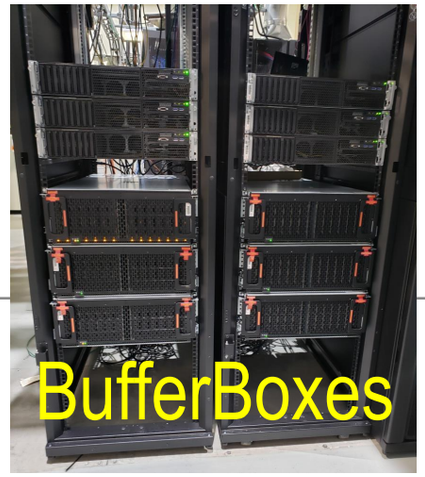
Lustre takes this same concept to multiple file servers

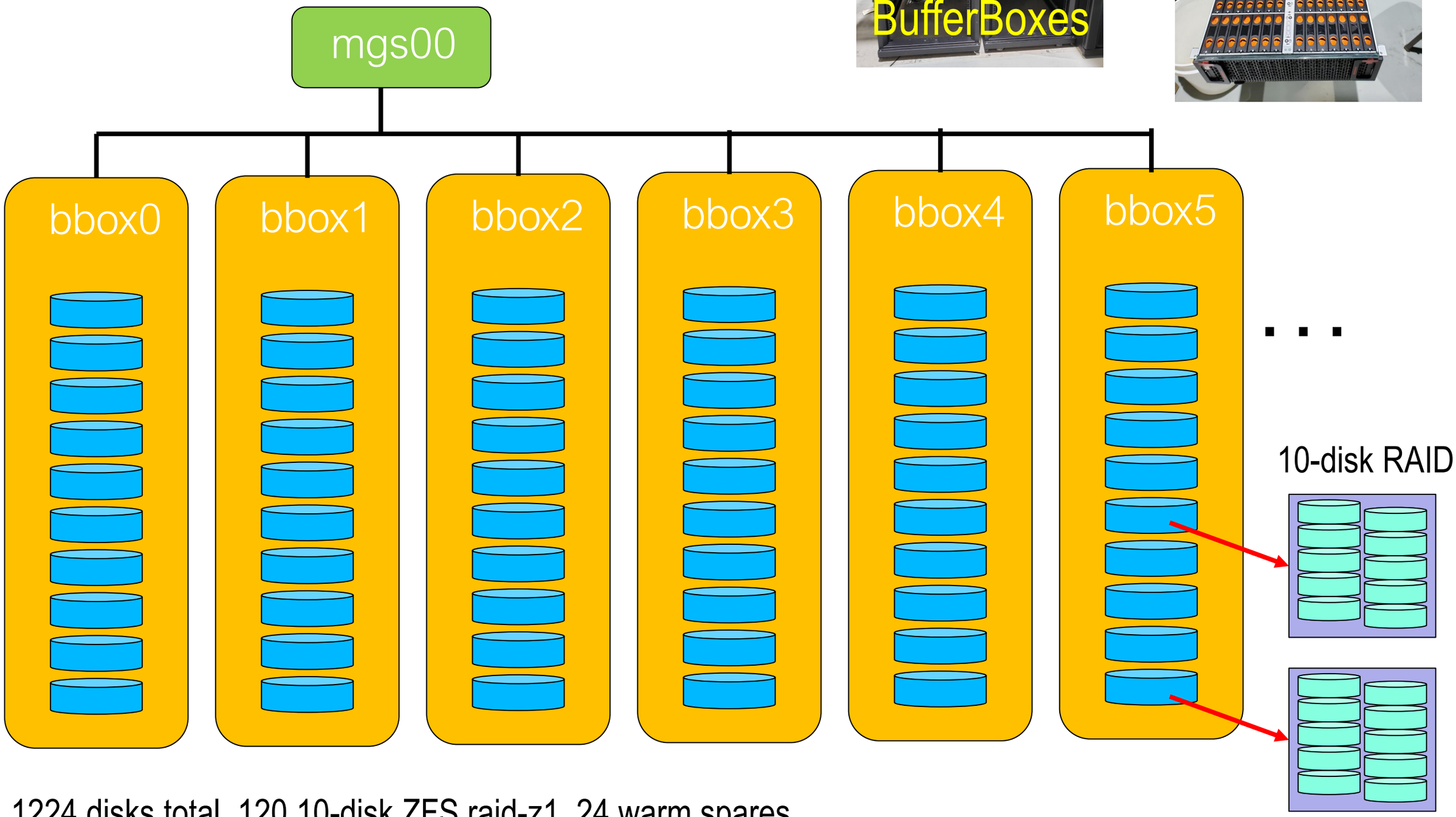
Same idea, distribute I/O across multiple servers, gain I/O capacity and (additionally) network capacity

Else this works pretty much like RAID, break up files into chunks, distribute across file servers.

Lustre is open-source, Zeph is fee-based from RedHat (pretty much the same concept)

Lustre

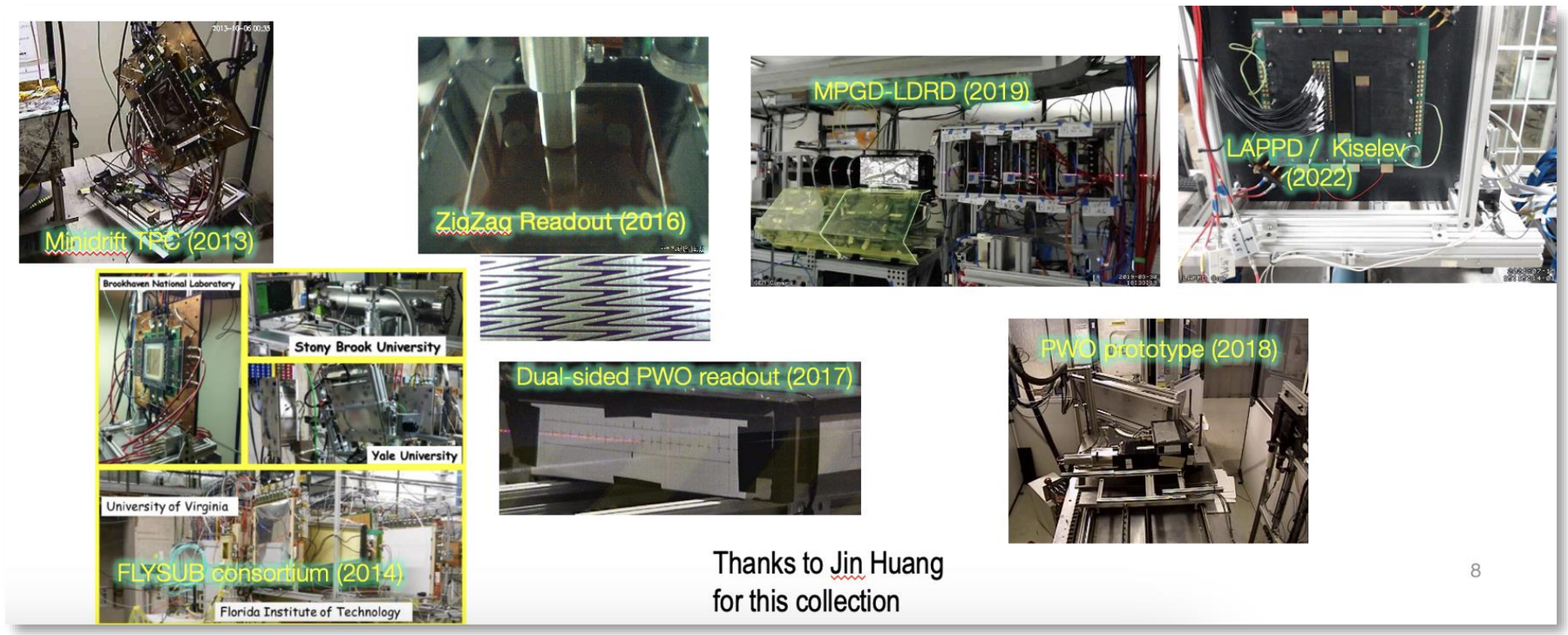




1224 disks total, 120 10-disk ZFS raid-z1, 24 warm spares

Test beams

Remember, RAID allows you to “stripe” your I/O across multiple disks



A small collage of the most important test beam setups over the years