sPHENIX

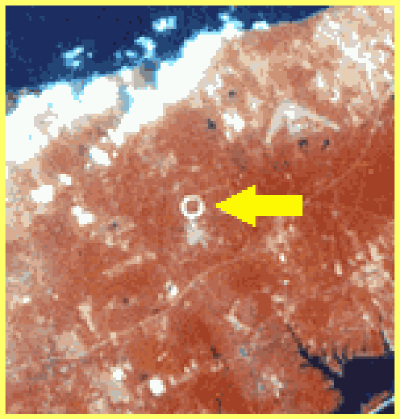
The sPHENIX RCDAQ System – Streaming Readout

Martin L. Purschke

Brookhaven National Laboratory

Manhattan

Long Island, NY



RHIC from space

What I’m going to talk about

A brief overview of sPHENIX

How we “do” SRO

The outsized role SRO played in 2024

sPHENIX data logging

Data compression

Updates for Run 2025

There are 3 more sPHENIX Detector talks

I’ll try to put things into a bit more context for a more useful overview how we ran things

This is how 2023 ended…

During the 2023 RHIC Run, on August 1, the accelerator developed a problem with a “Valve Box” that damaged an important magnet and led to the loss of a substantial amount of Helium

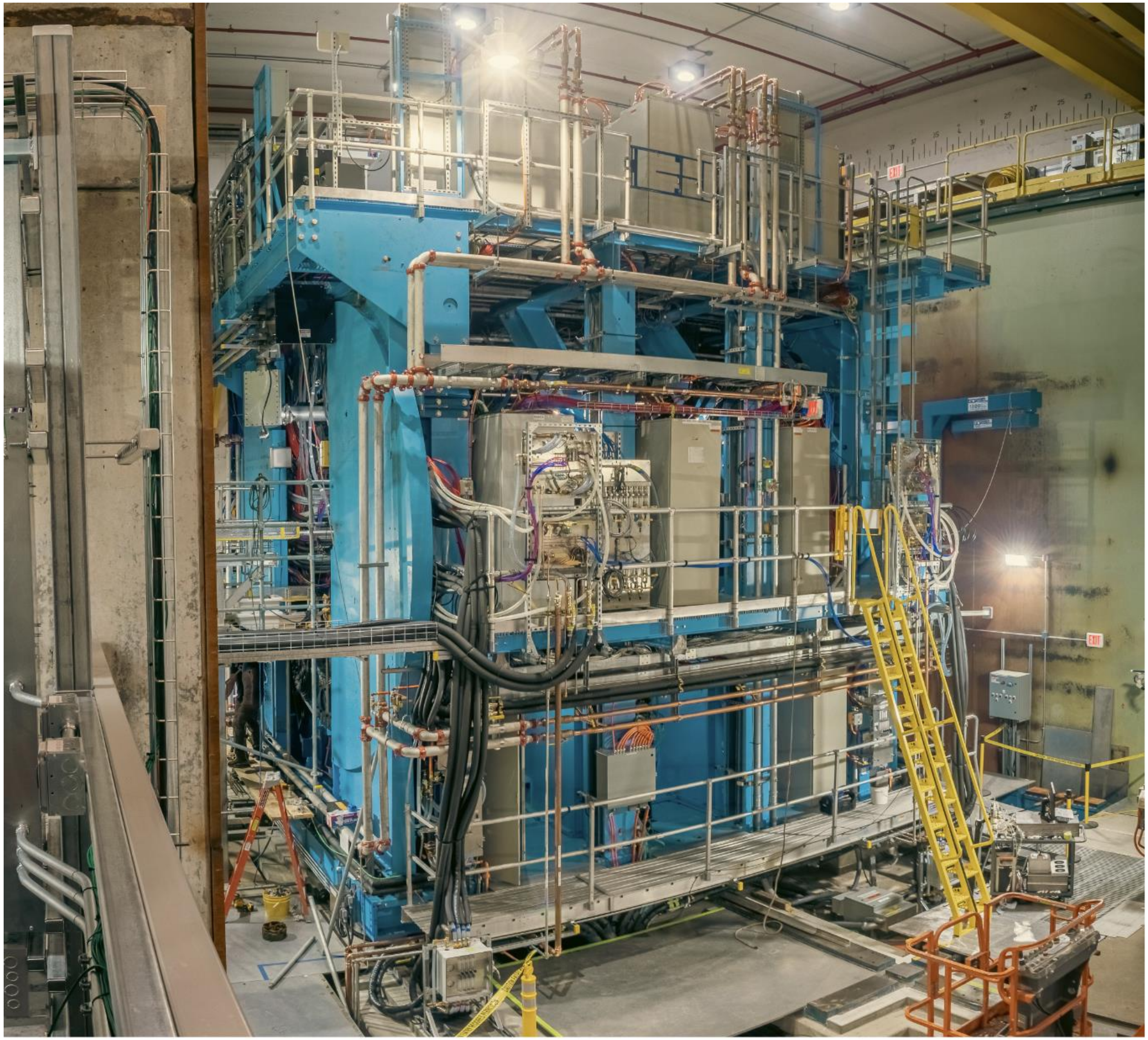
After an initial investigation, we had to terminate the 2023 beam operations, to resume in 2024 (which was a *much* better Run…)

2024 was the polarized p-p run (with 3 weeks of Au+Au at the end)

p-p has a much higher collision rate than Au+Au, requiring more triggers than Au+Au (where we can pretty much get all min-bias triggers to tape)

The higher collision rate is where our streaming-readout really added *a lot* of physics!

The sPHENIX Detector



: Calorimeters

iagnetic Calorimeter

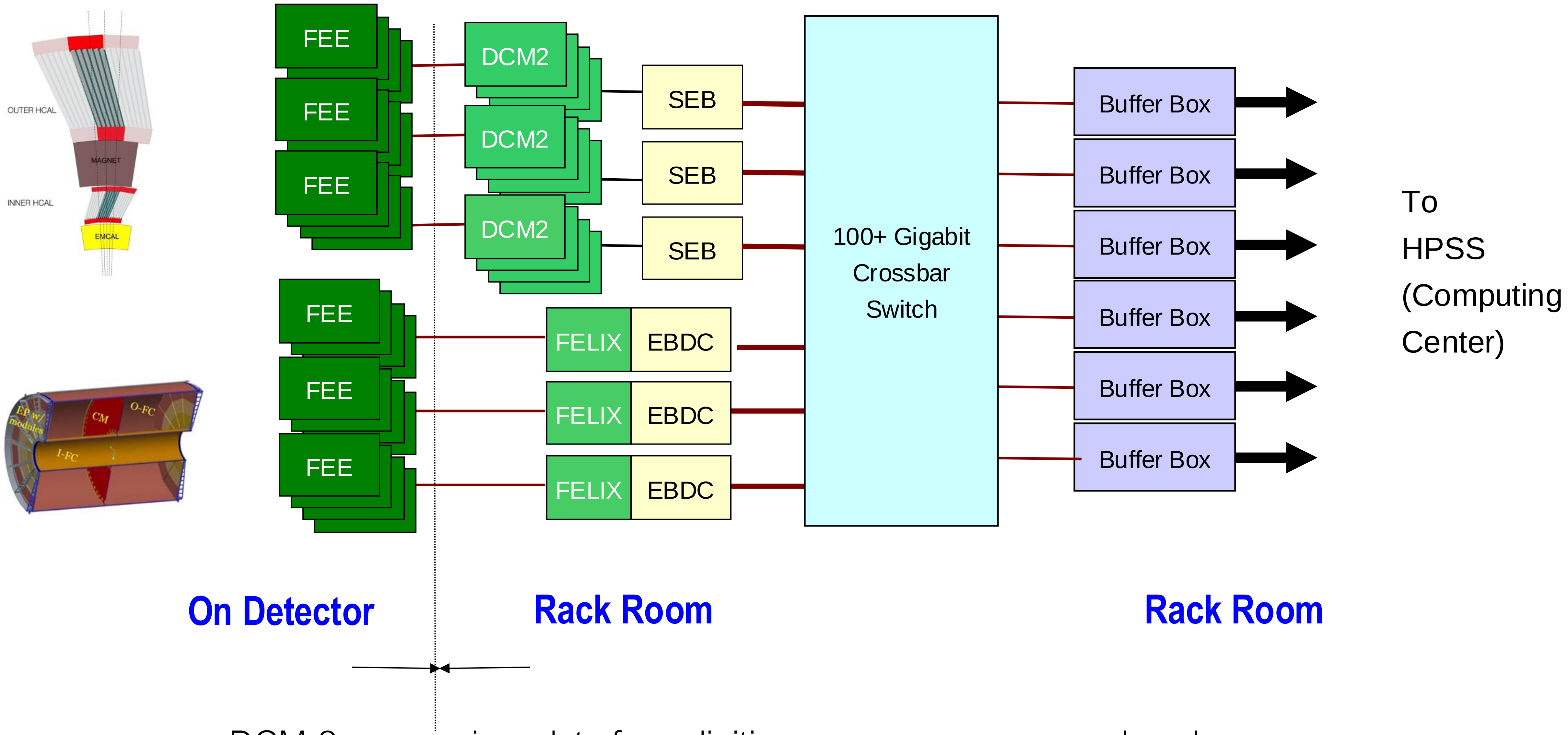
ojection Chamber (TPC)

ı Bias Detector (MBD)

liate Tracker (INTT)

etector (MVTX)

DAQ Overview



FEE

FEE

FEE

DCM2

DCM2

DCM2

SEB

SEB

SEB

FEE

FEE

FEE

FELIX EBDC

FELIX EBDC

FELIX EBDC

100+ Gigabit Crossbar Switch

Buffer Box

Buffer Box

Buffer Box

Buffer Box

Buffer Box

Buffer Box

To HPSS (Computing Center)

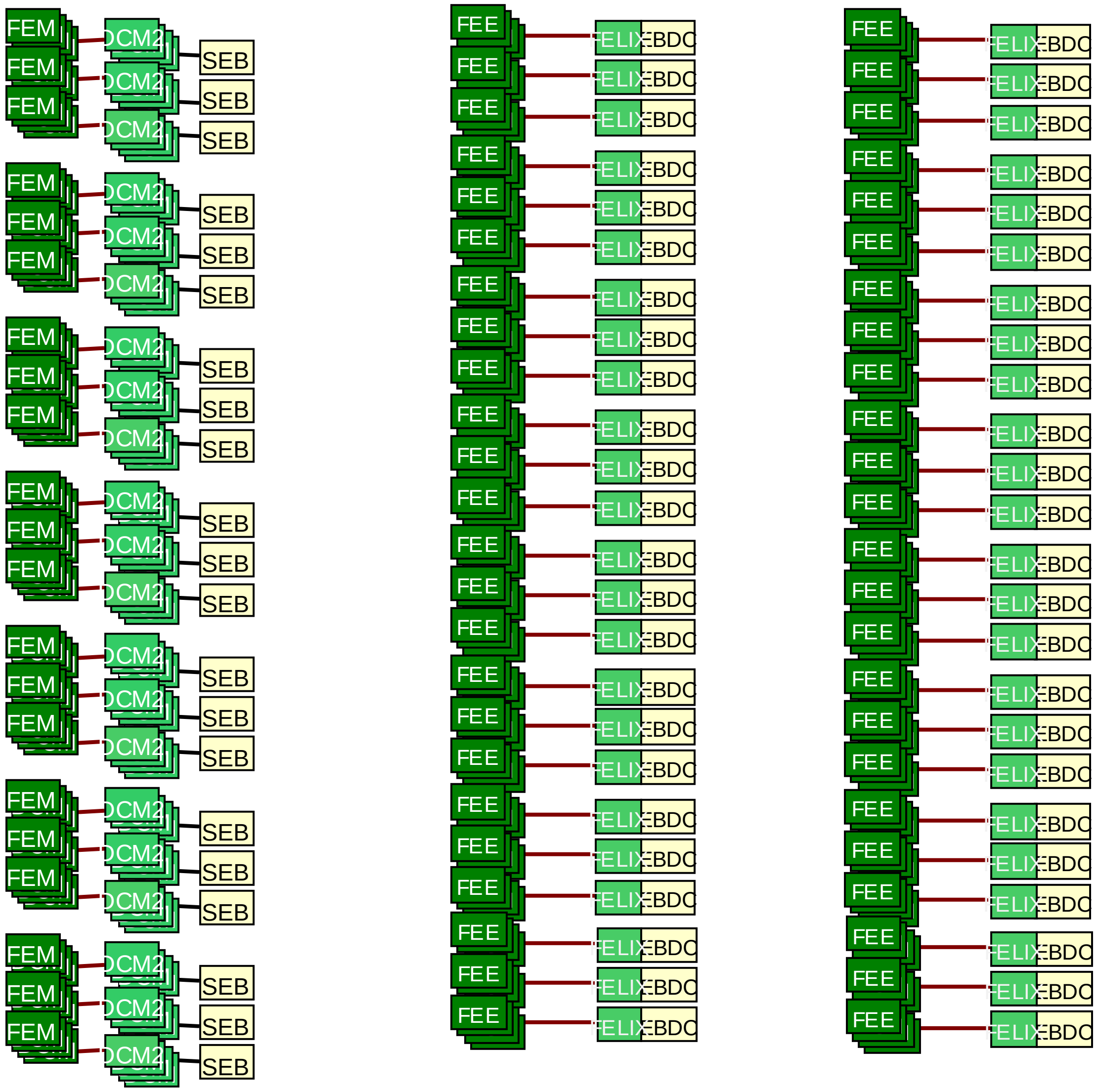
On Detector

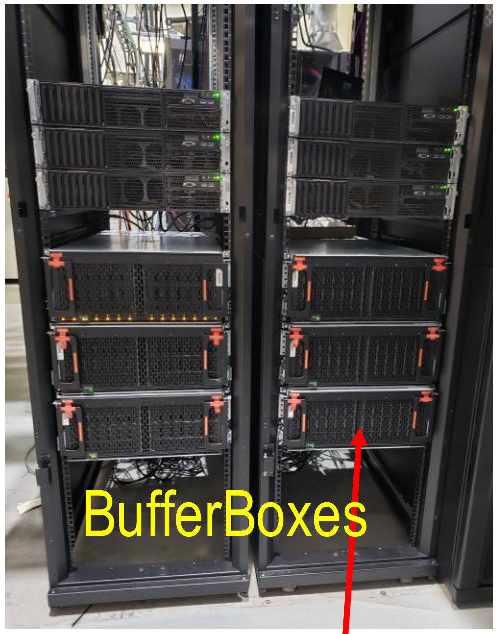
Rack Room

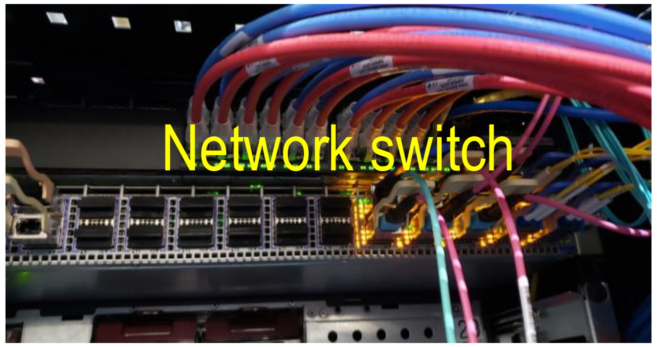
Rack Room

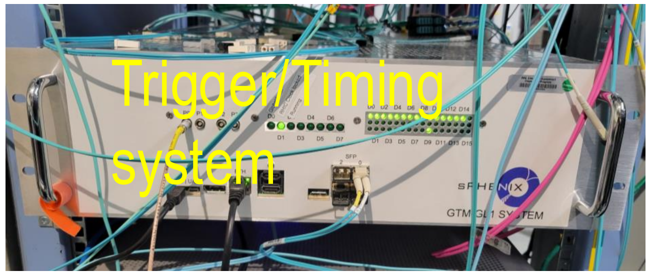
- DCM-2 receives data from digitizer, zero-suppresses and packages
- SEB collects data from a DCM group (~20)
- EBDC Event Buffer and Data Compressor (~40)
- Buffer Box data interim storage before sending to the computing center (6)

Some of our DAQ gear at the experiment









RCDAQ – Some of the High Points

- Each interaction with RCDAQ is a network connection that transmits the action to be taken and a response coming back
- The most-often used implementation is an atomic shell command. There is no "starting an application and issuing internal commands" (think of your interaction with, say, root)
  - That makes everything in RCDAQ inherently scriptable in standard bash or your other favorite shell (or python)
  - We start a sPHENIX DAQ run by pressing one button that fires off a script that takes care of it all
  - In test beams and tests in your lab you can script entire measurement campaigns and run them "on autopilot" – think bias voltage scans, position scans etc
- RCDAQ out of the box doesn't know about any particular hardware. All knowledge how to read out something, say, the FELIX board, comes by way of a **plugin** that teaches RCDAQ how to do that.
- All RCDAQ control interfaces are network-transparent
- There is no practical limit for concurrent control connections for RCDAQ

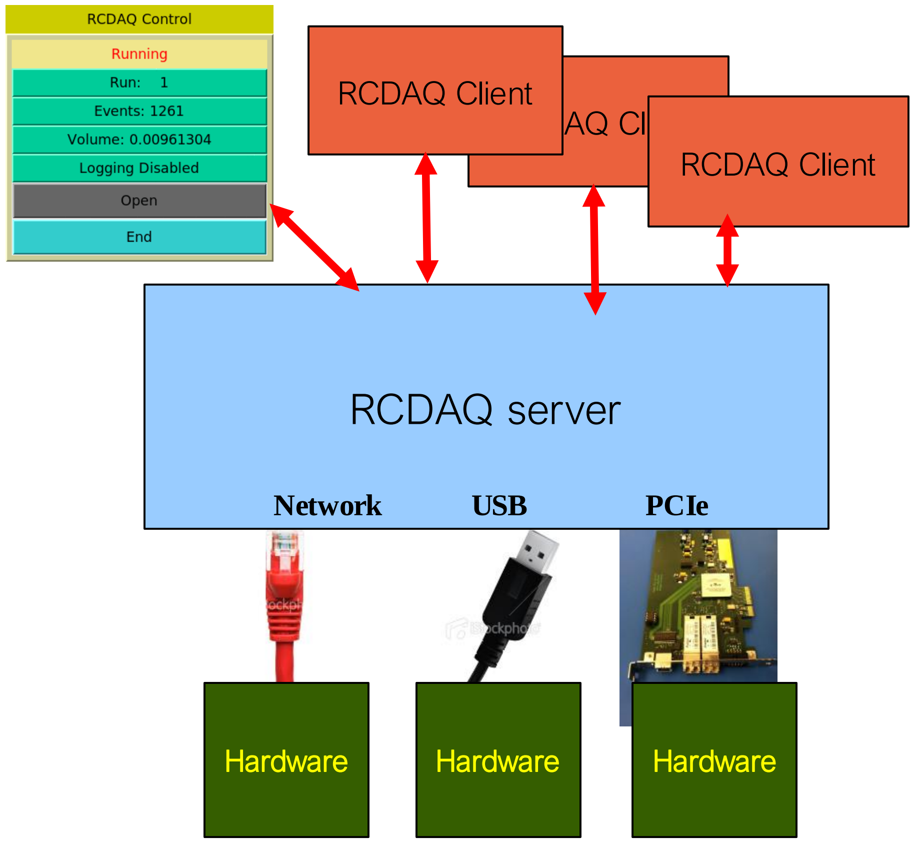
How we read out the detector

At the core of the DAQ is a multitude of individual “RCDAQ” processes on as many PCs that read out a part of the detector

RCDAQ is a versatile DAQ system that can be run standalone or as part of a “cohort”

In the latter case, the RCDAQ instances are controlled by a meta-control process “run control” (rc for short)

Standalone



RCDAQ Control

Running

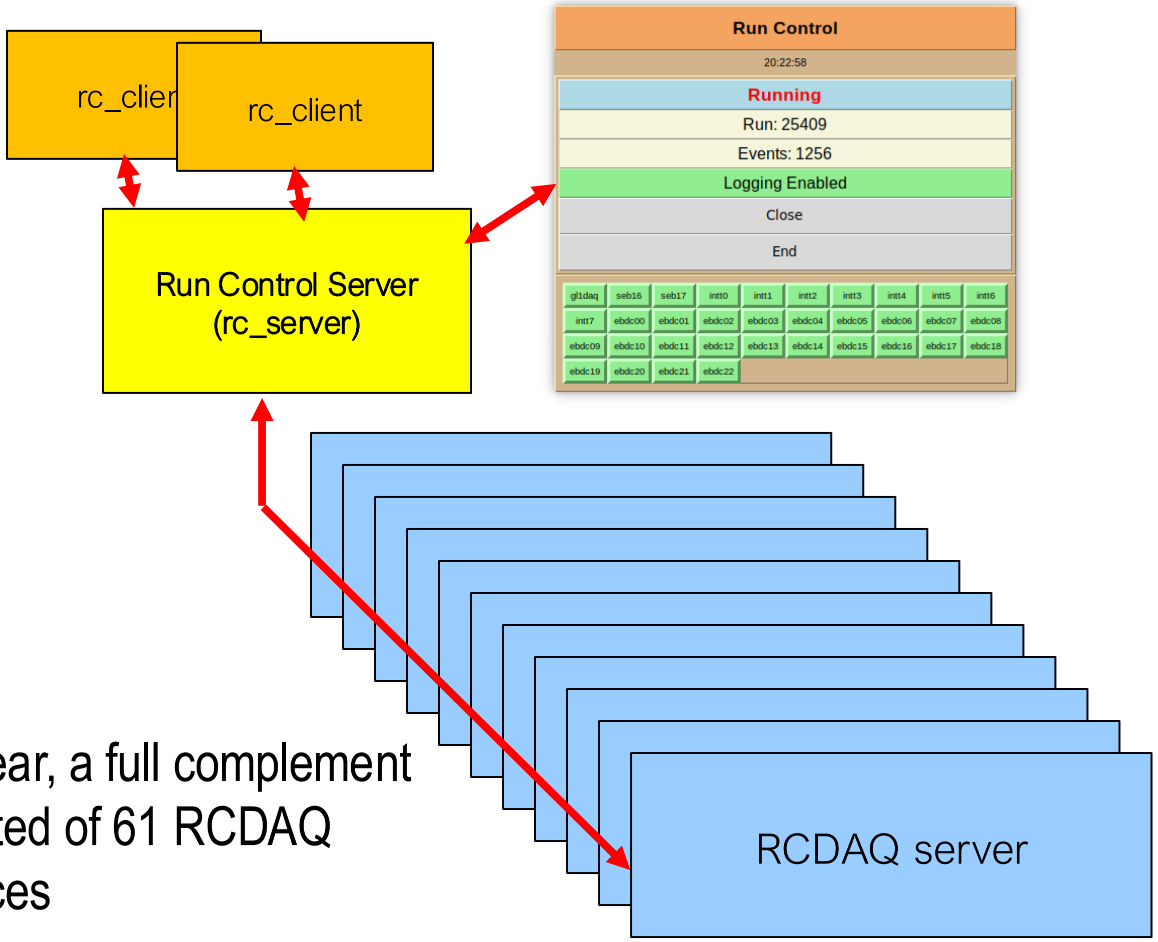
Run: 1

Events: 1261

Volume: 0.00961304

Logging Disabled

As a “cohort”



Run Control

20:22:58

Running

Run: 25409

Events: 1256

Logging Enabled

This year, a full complement consisted of 61 RCDAQ instances

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 it is offered and don’t care what it is”

(It can read out your detector, obviously, but store really any kind of data in its data stream)

In that sense 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

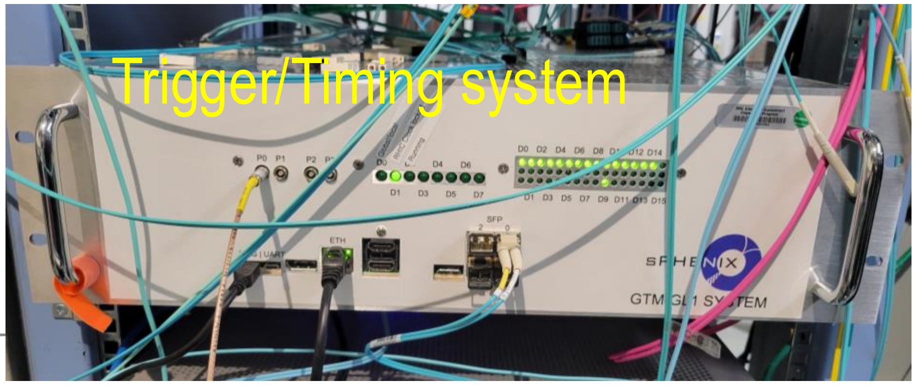
```
#!/bin/bash
rcdaq_client load   librcdaqplugin_dam.so
rcdaq_client create_device device_dam 1 4001 1 128
```

Here:

DAM = “Data Aggregation Module” aka “Felix card”

We have many different plugins that allow RCDAQ to read our different detectors (and many more to support readout hardware you will find in your typical test beam)

The Timing System holds it all together



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

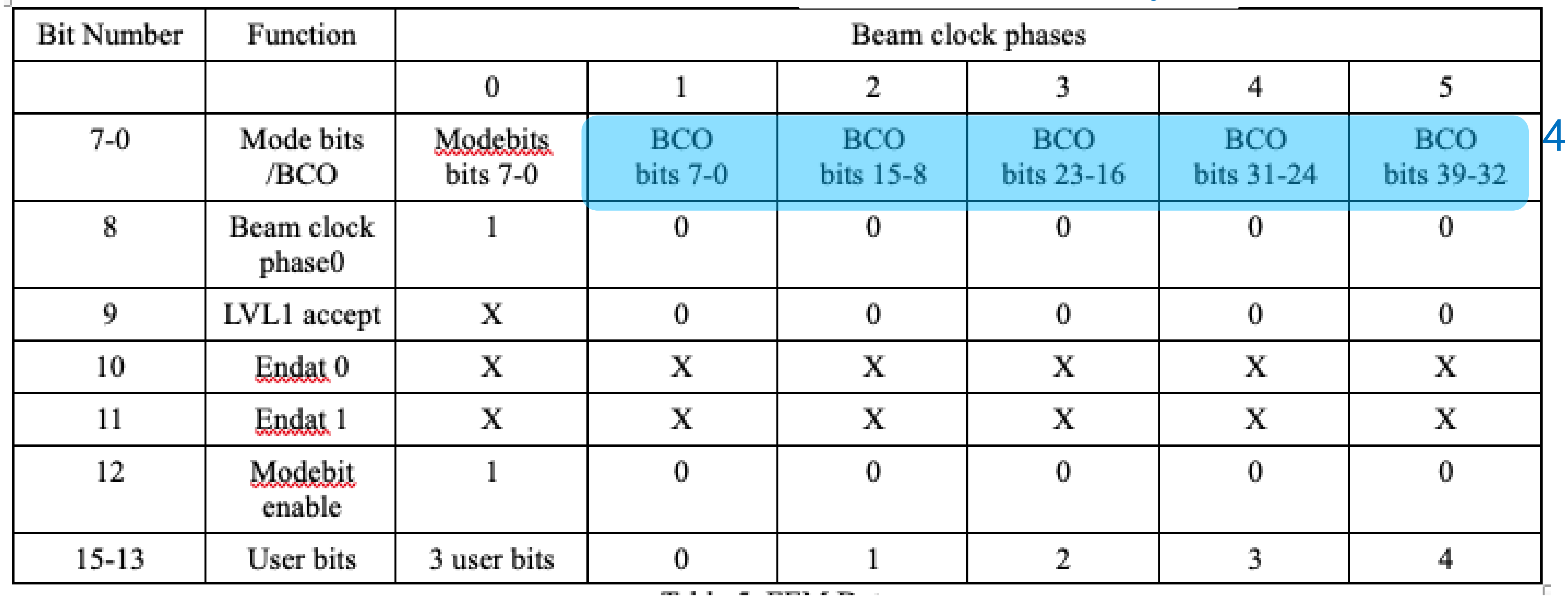
We have a global 64 bit master beam-crossing (BCO) counter.

We transmit 40 of the 64 bits to the FELIX Cards, those 40 bits go into the data stream

The FELIXes 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 ~110ns):

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: (older – 2021) sPHENIX TPC data

Clock values embedded in FEE data

40 bits BCO

```
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
…
```

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

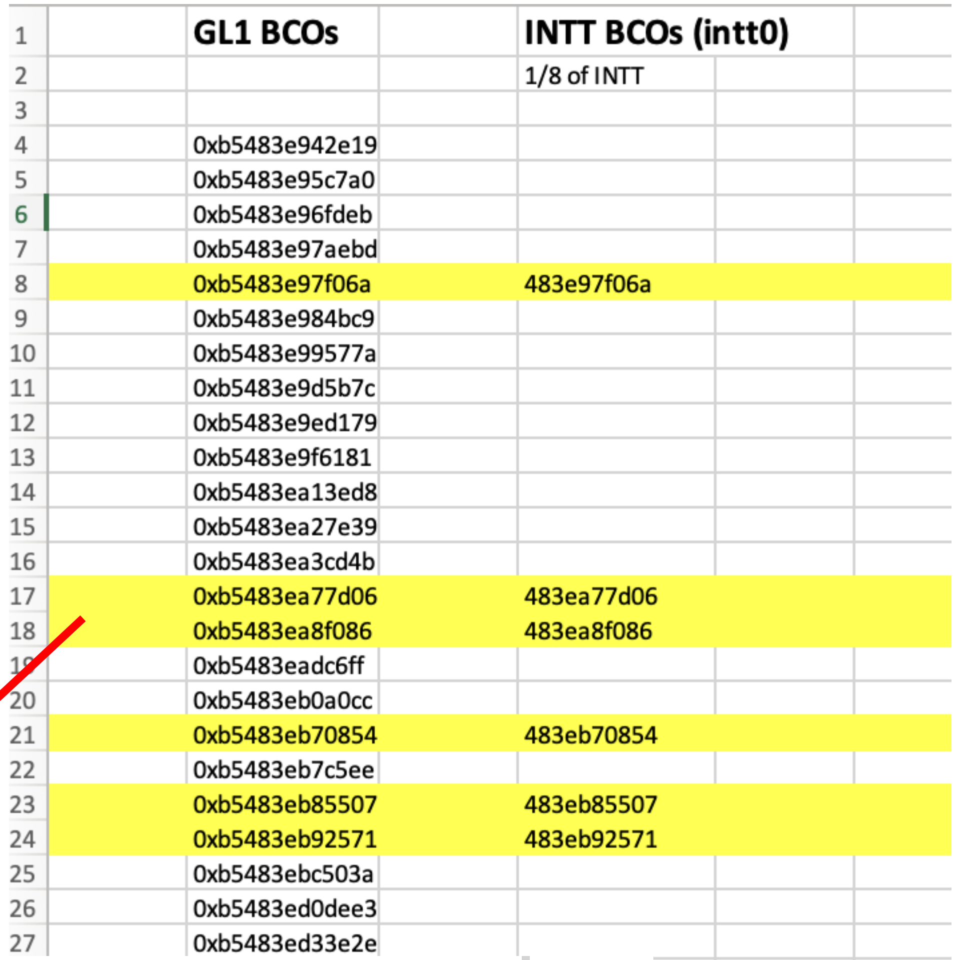
I’m showing an older version here since it’s easier to see

Example – INTT (Intermediate Tracker)

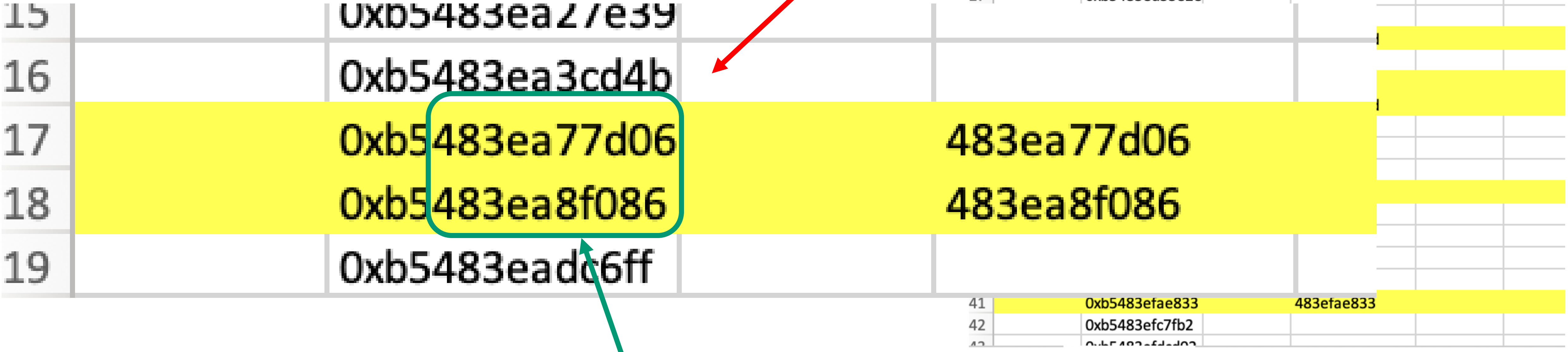
The left column are BCO’s that were triggered on

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		GL1 BCOs		INTT BCOs (intt0)	
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			
41		0xb5483efae833		483efae833	
42		0xb5483efc7fb2			



15		0xb5483ea27e39		
16		0xb5483ea3cd4b		
17		0xb5483ea77d06	483ea77d06	
18		0xb5483ea8f086	483ea8f086	
19		0xb5483eadc6ff		

40 bits

Triggered and Streaming Readout

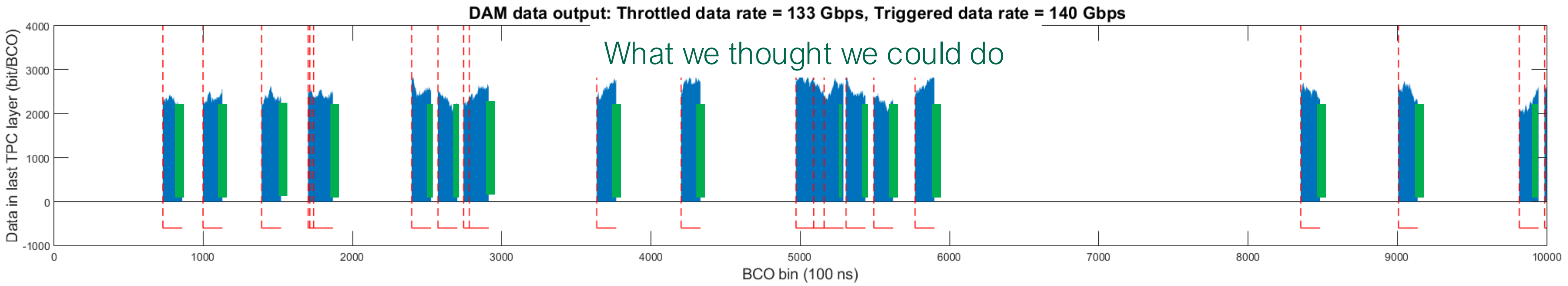
I have talked about sPHENIX’s combined triggered + streaming readout in various places

On a trigger, we always read out *everything*

But then, for the SRO-capable tracking detectors, we don’t say “stop” right away but cover the following beam crossings as well

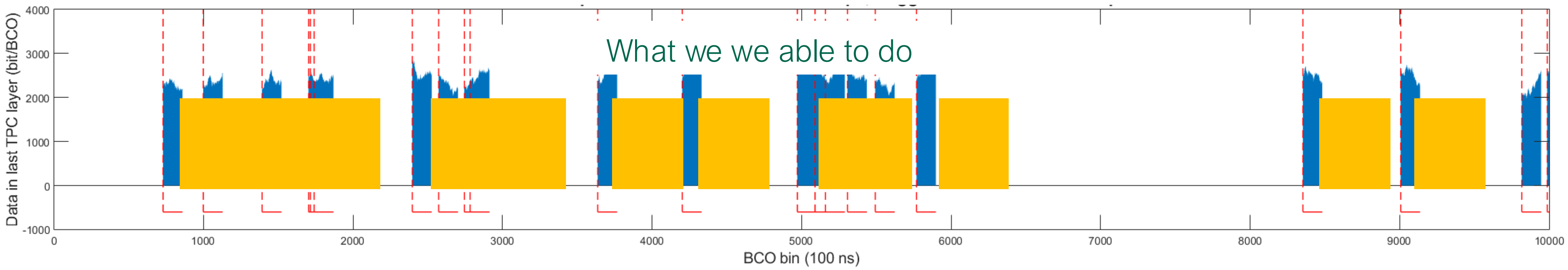
We were able to keep streaming for 50us or ~460 additional beam crossings (much more than we thought we could!)

That added, per original trigger, between 10-25 additional collisions in the data stream



DAM data output: Throttled data rate = 133 Gbps, Triggered data rate = 140 Gbps

What we thought we could do



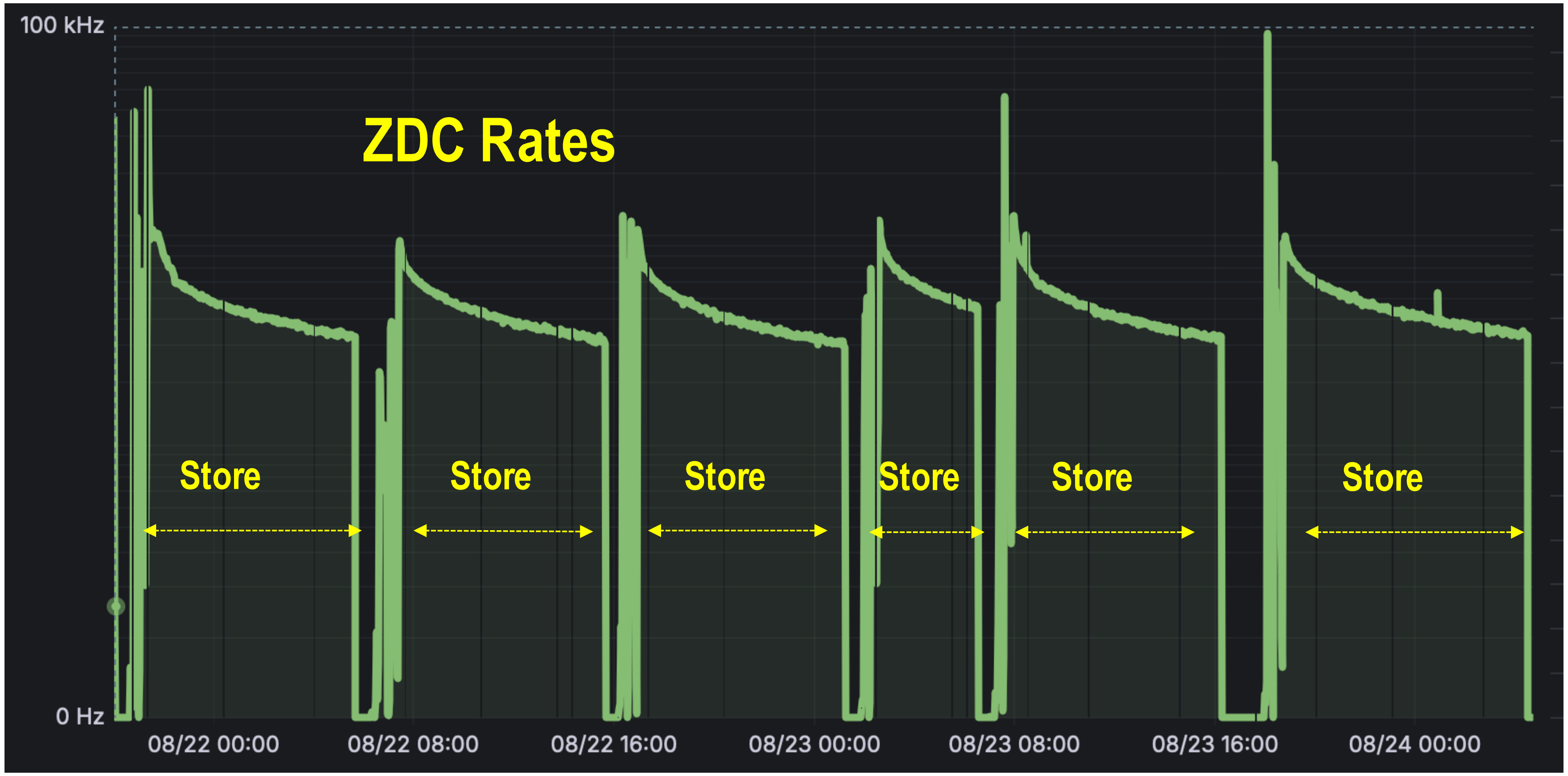
What we we able to do

Data/Trigger rate management

Here is a 3-day timeline of RHIC stores

One can see the high luminosity at the begin of the store, going down over the course of a typical 8-hour store

(the ZDC rates are captured all the time, the min-bias rates I'll show in a moment only when our DAQ is running)



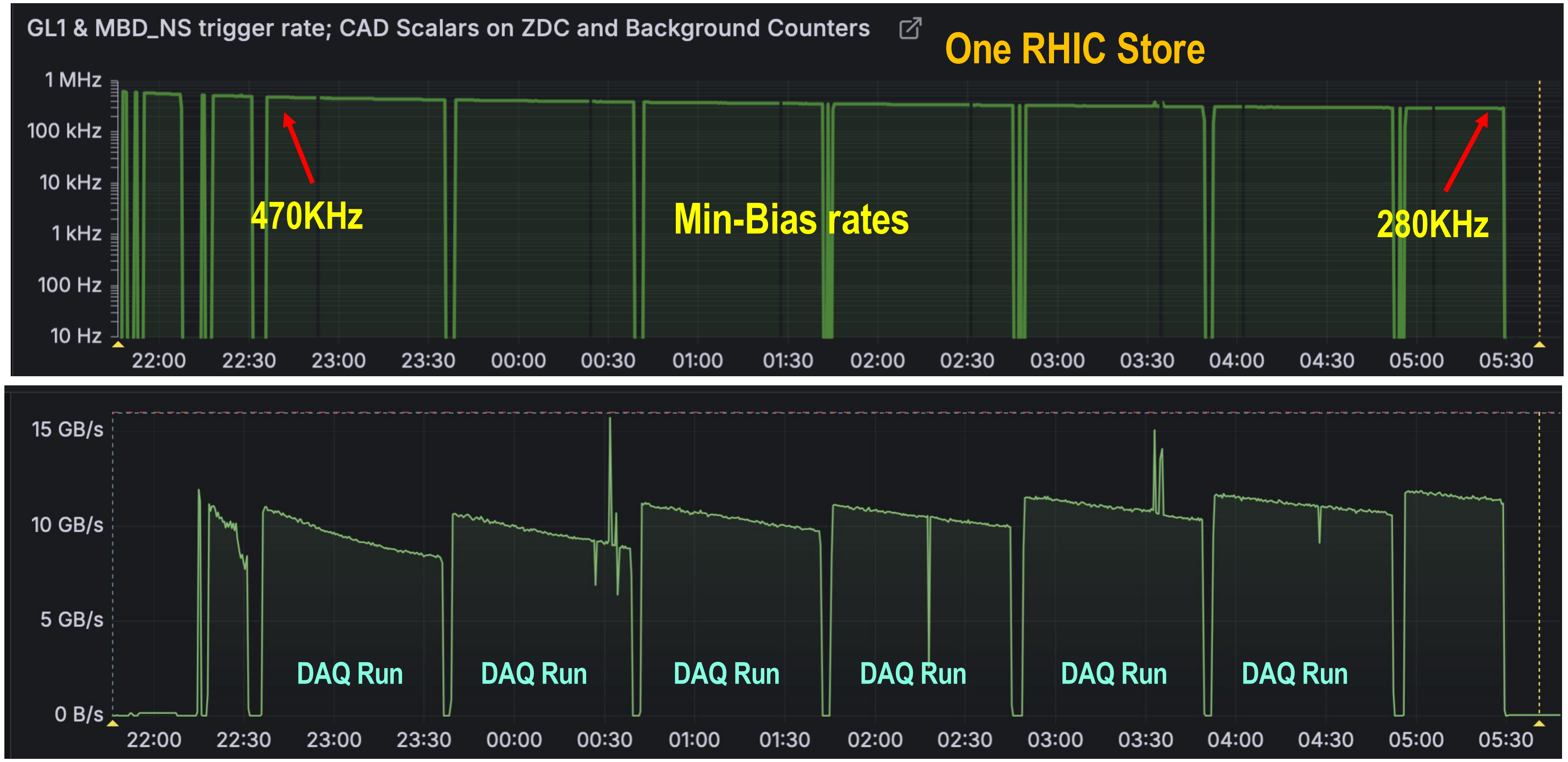
Dynamic data rate management

How many “streaming” collisions you get depends on the current RHIC luminosity

470KHz -> ~24 collisions in any 50 μs

280KHz -> ~14 collisions in any 50 μs

Here we adjusted the triggers for the rates at the begin of each new DAQ run, leading to a “decay” of the data rate over the course of such a run



Dynamic data rate management

We later changed to a “dynamic” rate management

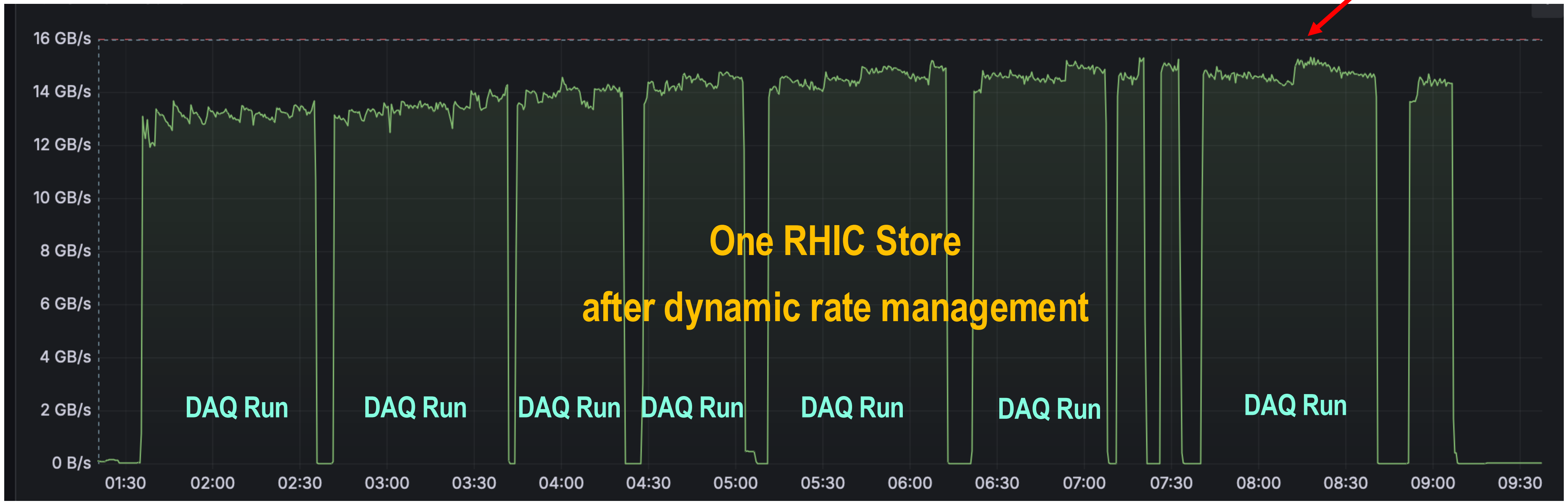
Over the course of a DAQ run we re-calculated and adjusted the min-bias trigger scaledown every 3 minutes

Min-bias in p-p is not a super-valuable trigger, hence the saledown

What this really did is control the number of “50 μs streaming intervals” we would schedule to capture more of streaming collisions at lower collision rates

Now the DAQ rates are staying constant or even increase (coarse adjustment by int. numbers)

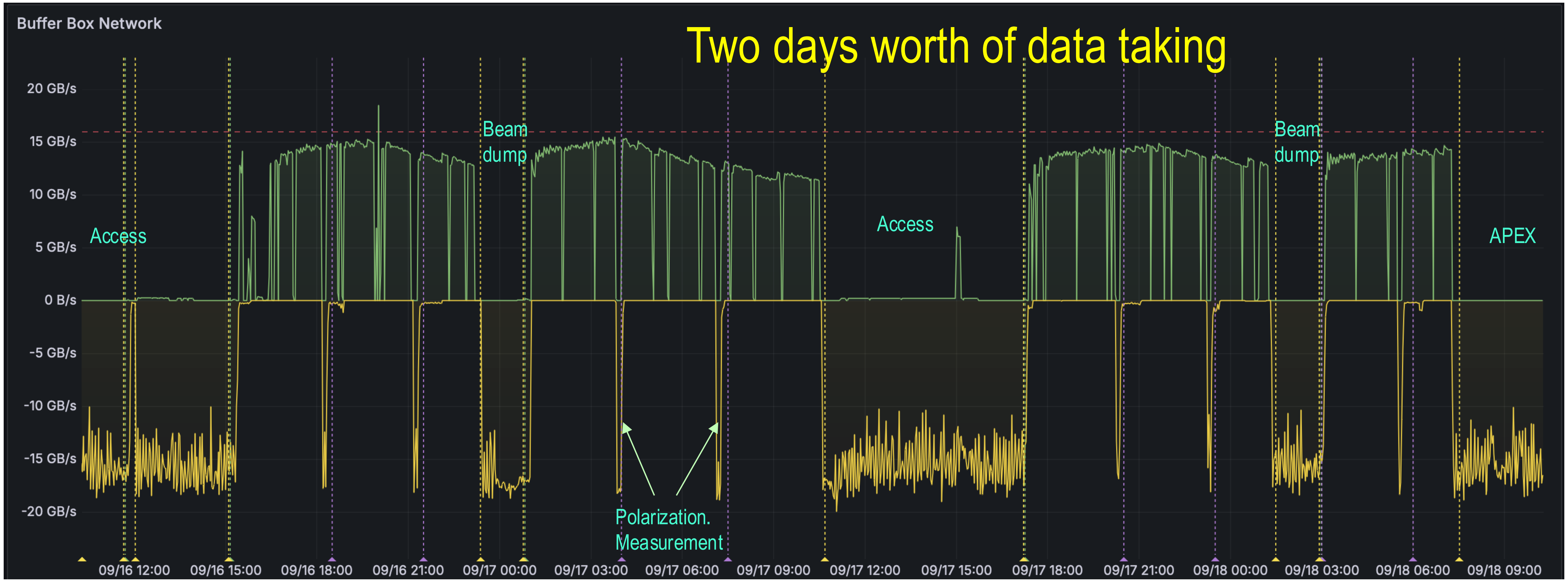
Much better use of the available DAQ bandwidth!



Here is how this looks

The green (positive) is data incoming from the DAQ

The yellow (negative) is data going out to permanent storage



At this point we had to choose between taking data and sending data to storage because we maxed out our disk bandwidth

Not a complaint! The fact that we reached that limit is a measure of how well stuff worked!

But I’ll talk about some upgrades at the end…

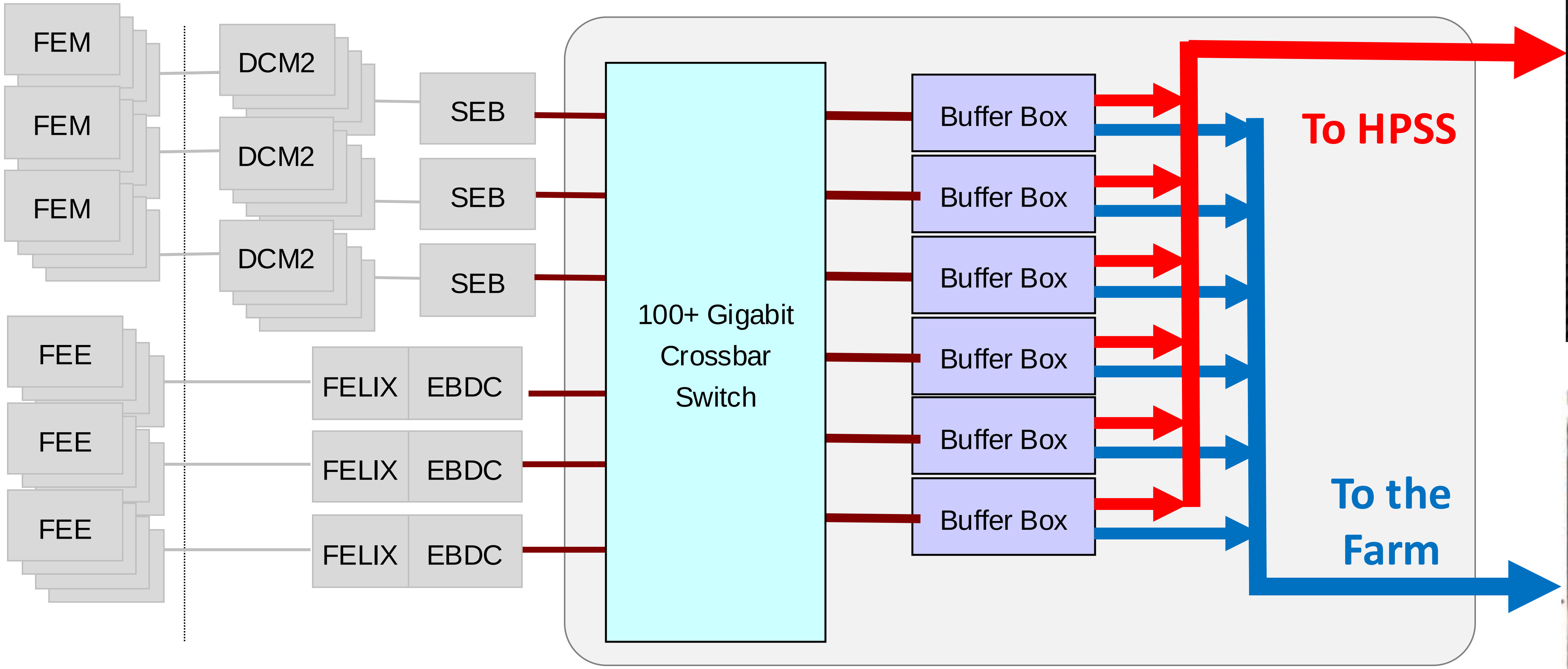
sPHENIX Data Flow to Storage

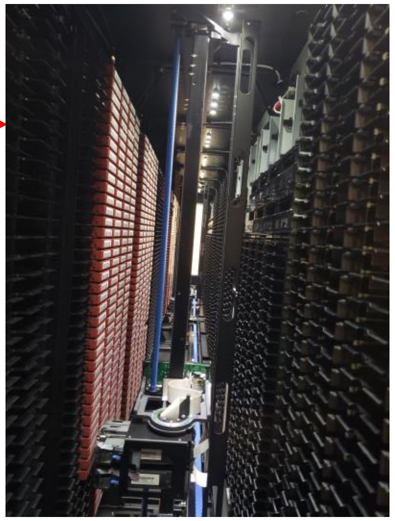
One copy of the raw data goes to the HPSS tape storage system

One copy goes to the computing farm for near-line monitoring, calibration, reconstruction

Much faster turn-around

We can devote more tape drives to writing

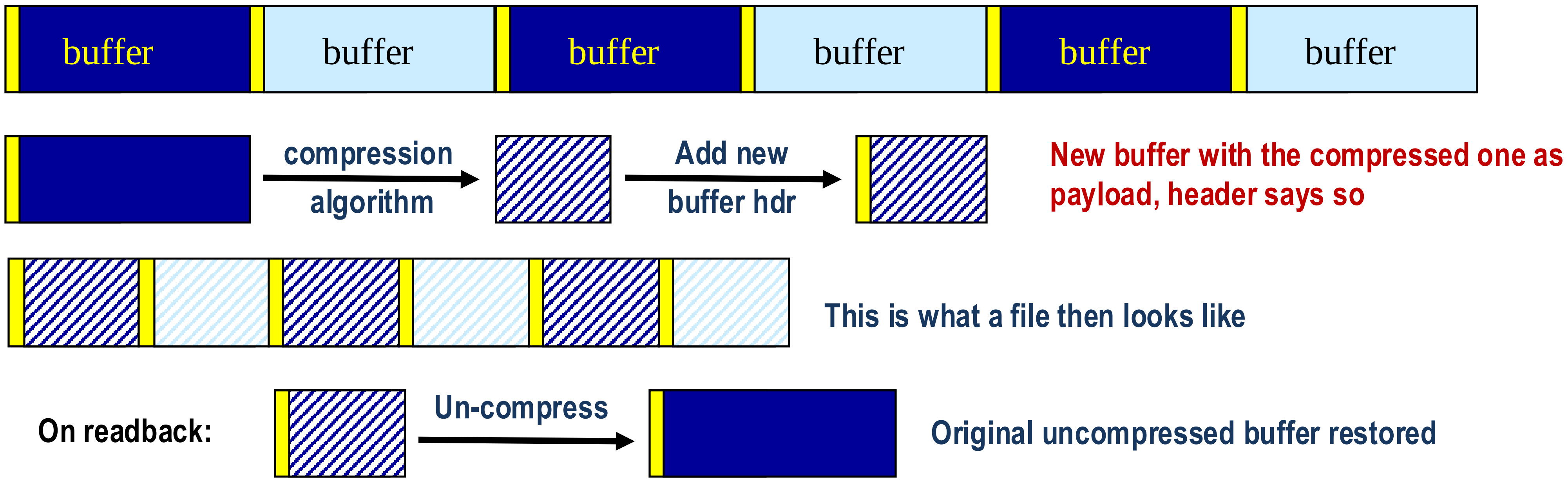




After SRO: Multi-threaded Data compression

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:



New buffer with the compressed one as payload, header says so

This is what a file then looks like

Original uncompressed buffer restored

Our DAQ readout machines all have 96 CPU cores

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

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

Compression yields vary between 30% and 70% - 70% means 100GB become 70GB.

Better use of disk storage and also network throughput.

Compression levels

The original before-compression buffer size needs to be in the new header (so I know how much memory to allocate when uncompressing)

That makes it easy to calculate the per-buffer compression yield

The yields vary a lot by detector. Some samples from a utility that can look at that:

MVTX:

```
buffer at record    0 length = 11411347  1393 marker = ffffbefa BZ2 Marker Or.length: 33680192  33.8815%
buffer at record 1393 length = 11439349  1397 marker = ffffbefa BZ2 Marker Or.length: 33809176  33.835%
buffer at record 2790 length = 11473177  1401 marker = ffffbefa BZ2 Marker Or.length: 34190424  33.5567%
```

INTT:

```
buffer at record 23294 length = 29511868  3603 marker = ffffbefa  BZ2 Marker Or.length: 66846744  44.1485%
buffer at record 26897 length = 29587534  3612 marker = ffffbefa  BZ2 Marker Or.length: 66846968  44.2616%
buffer at record 30509 length = 29735365  3630 marker = ffffbefa  BZ2 Marker Or.length: 66847016  44.4827%
```

TPC:

```
buffer at record     0 length = 254832068  31108 marker = ffffbcfe  LZO Marker Or.length: 369239544  69.0154%
buffer at record 31108 length = 255366094  31173 marker = ffffbcfe  LZO Marker Or.length: 369165720  69.1738%
buffer at record 62281 length = 255258927  31160 marker = ffffbcfe  LZO Marker Or.length: 369129368  69.1516%
```

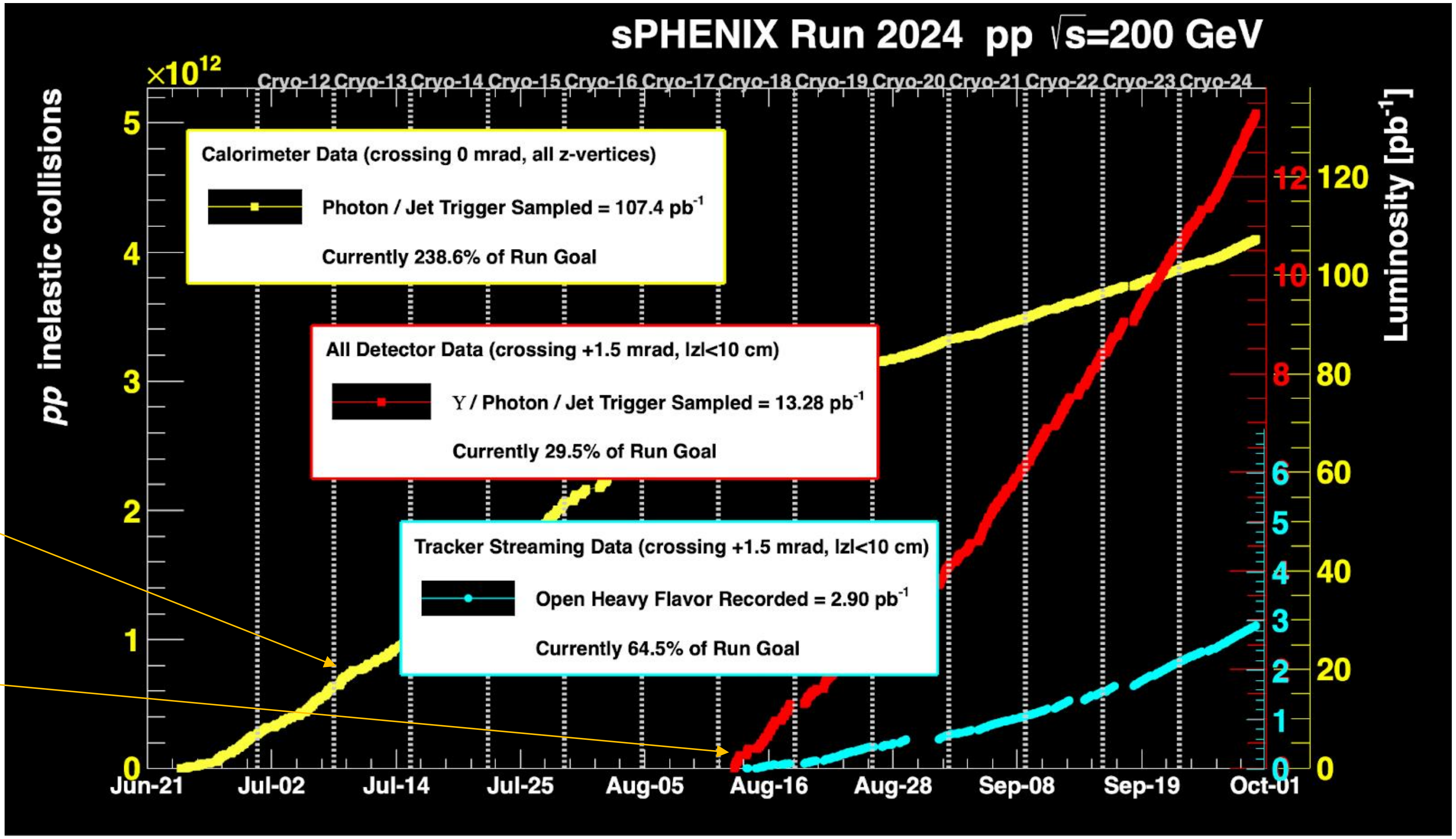
The TPC has too much data for “bz2” compression to be used – too slow

And what did we get?

On average we wrote between 490 and 530TB/day

“Best week” had more than 4PB or 580TB/day

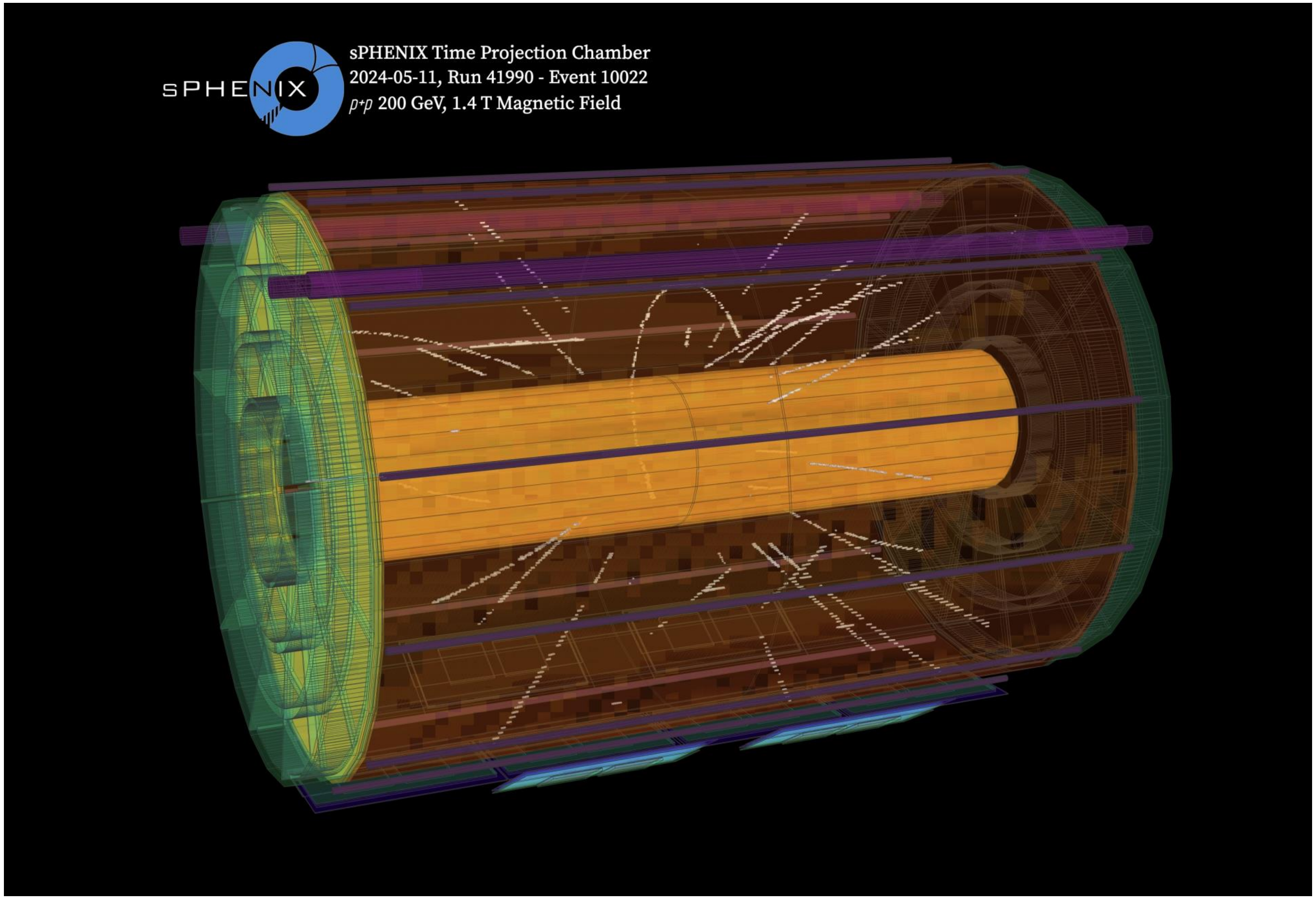
A total of 54PB on tape for Run 2024



The steeper slope at the beginning is from what I called “period 1” where we took high-speed data for the Jet program

We then ramped up the triggered+streaming readout for the Upsilon and Heavy Flavor programs

Let me show a wonderful event display…


SPHENIX
sPHENIX Time Projection Chamber
2024-05-11, Run 41990 - Event 10022
p+p 200 GeV, 1.4 T Magnetic Field

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

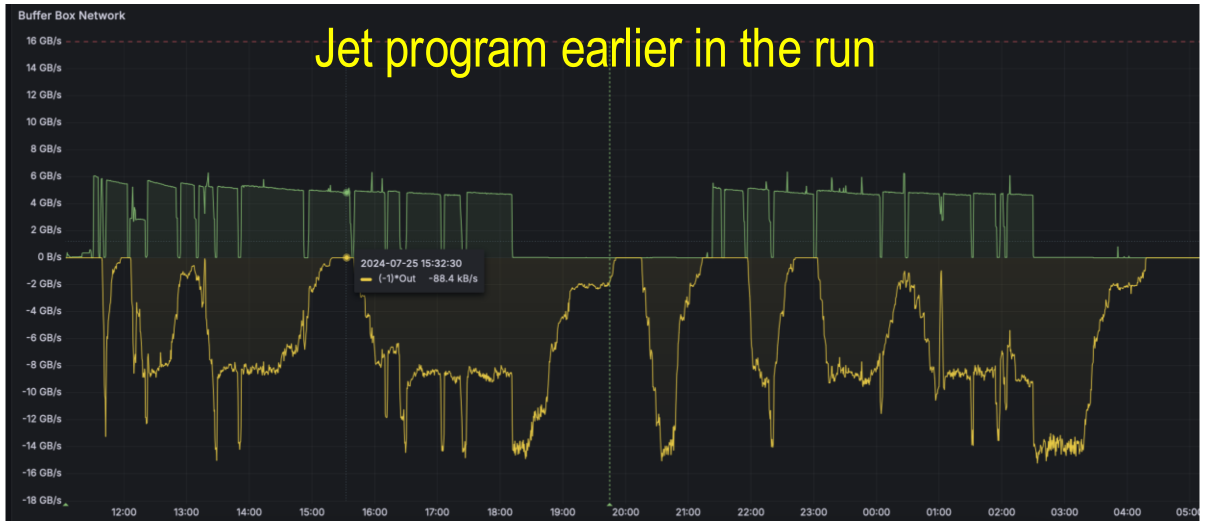
Don’t want to steal the next speaker’s thunder…

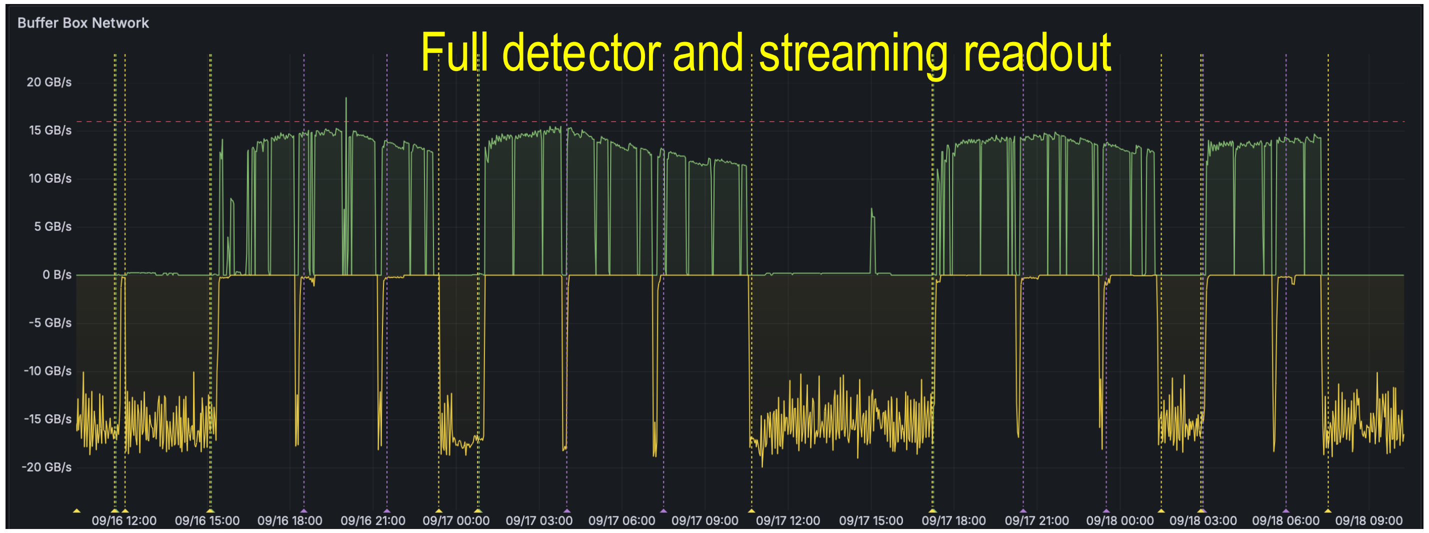
The Future / DAQ Upgrades

I alluded to maxing out the disk bandwidth for data logging – theoretically 22GB/s, 19.5GB/s long-term average in reality

RHIC delivered data for about 16GB/s

I showed before that we had to choose between data taking and data transfer to the computing center later in the run with Streaming Readout


Jet program earlier in the run


Full detector and streaming readout

Across the board, we will double our disk bandwidth by going to 12 instead of 6 bufferboxes

We will also continue to eliminate inefficiencies in the DAQ, failures, etc

Lots of smaller upgrades, such as “fractional scaledowns”

Summary

We had a good run 2024!

Nothing is perfect in year 2 of a new experiment, but the DAQ and the detector worked exceedingly well

The Streaming Readout was a success story beyond our wildest dreams

The multi-threaded compression of the raw data essentially doubled our DAQ logging bandwidth

Buying more hardware to double the DAQ bandwidth again

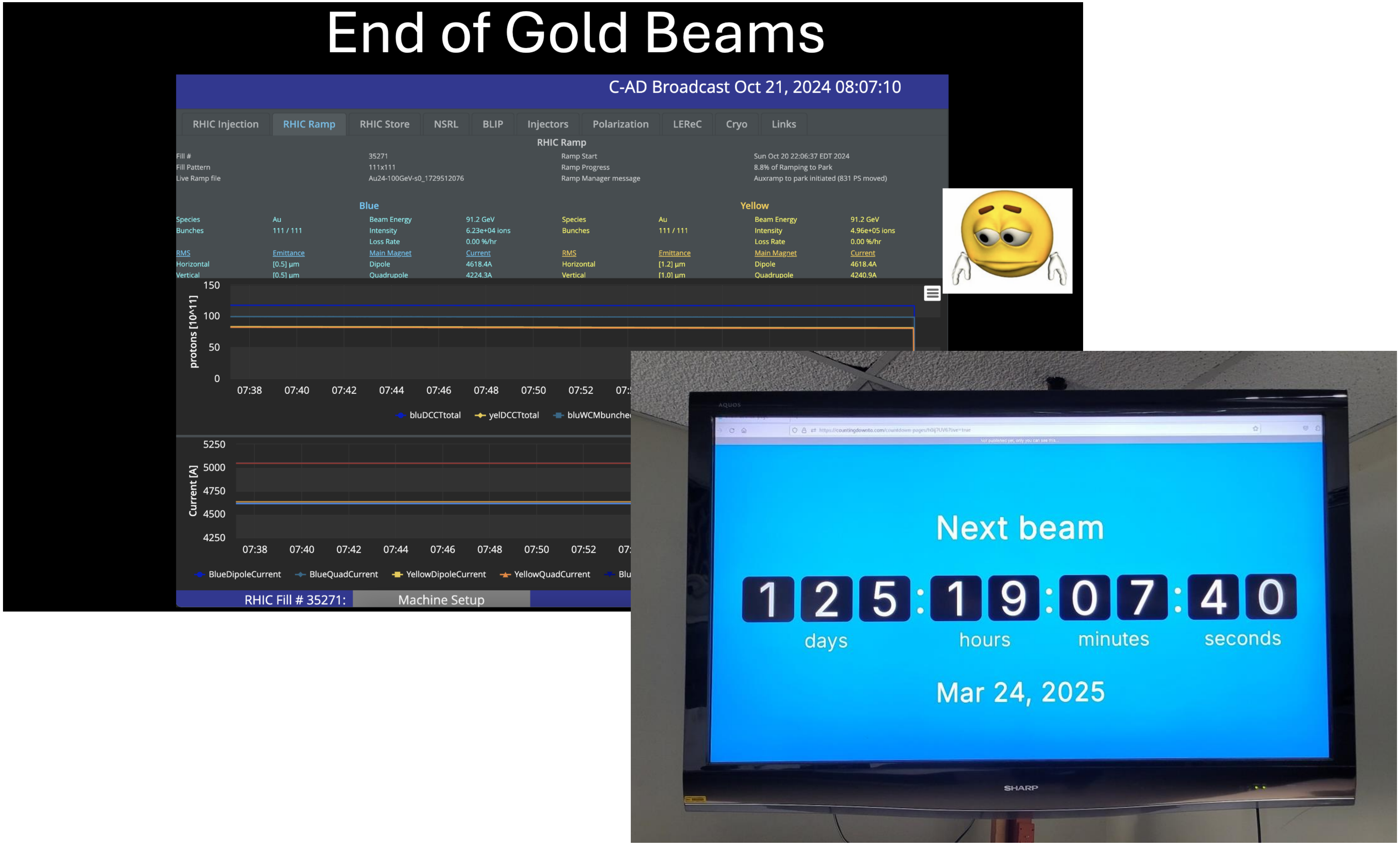
I didn't have time to talk about our trigger system, but that also worked beautifully

We got about 54PB of raw data to tape, with weekly averages of > 500TB/day

After about 45 weeks of being at the experiment pretty much 7 days a week, we are looking forward to a winter break of the 24/7 ops

But mostly we are looking forward to the next 2025 RHIC Run starting in March!

The End on October 21, 2024



End of Gold Beams

C-AD Broadcast Oct 21, 2024 08:07:10

RHIC Ramp

Fill #	35271	Ramp Start	Sun Oct 20 22:06:37 EDT 2024
Fill Pattern	111x111	Ramp Progress	8.8% of Ramping to Park
Live Ramp file	Au24-100GeV-s0_1729512076	Ramp Manager message	Auxramp to park initiated (831 PS moved)

Blue

Species	Au	Beam Energy	91.2 GeV
Bunches	111 / 111	Intensity	6.23e+04 ions
		Loss Rate	0.00 %/hr
RMS	Emittance	Main Magnet	Current
Horizontal	[0.5] µm	Dipole	4618.4A
Vertical	[0.5] µm	Quadrupole	4224.3A

Yellow

Species	Au	Beam Energy	91.2 GeV
Bunches	111 / 111	Intensity	4.96e+05 ions
		Loss Rate	0.00 %/hr
RMS	Emittance	Main Magnet	Current
Horizontal	[1.2] µm	Dipole	4618.4A
Vertical	[1.0] µm	Quadrupole	4240.9A

RHIC Fill # 35271: Machine Setup

Next beam

125 : 19 : 07 : 40

days hours minutes seconds

Mar 24, 2025

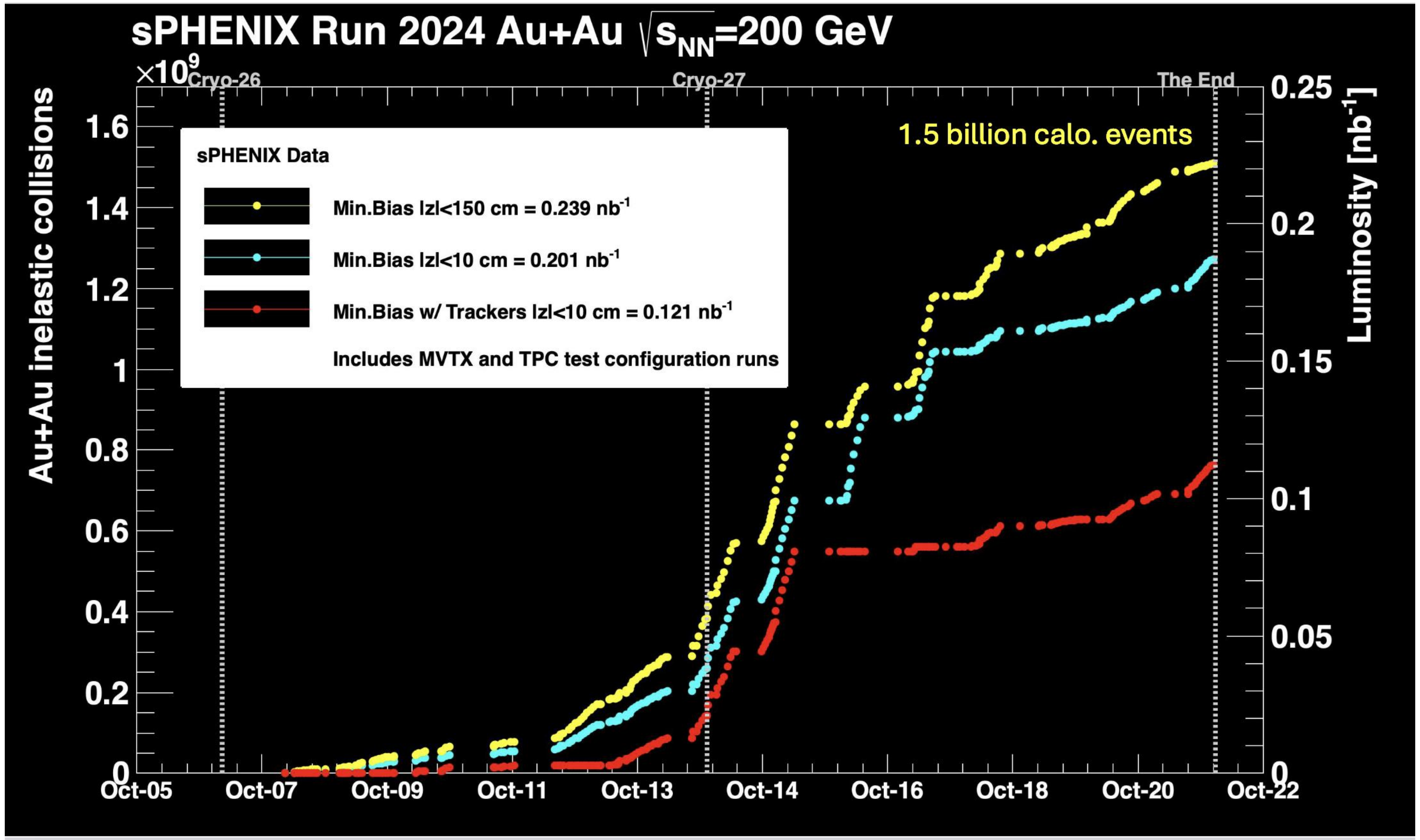
The End

Au-Au statistics

Not primarily goal to be a physics production run

Still got more statistics in some physics programs than in 2023 (ok that wasn't so hard, but still…)

Also gives us data to sink our teeth into Au+Au real data analyses to hit the ground running in 2025



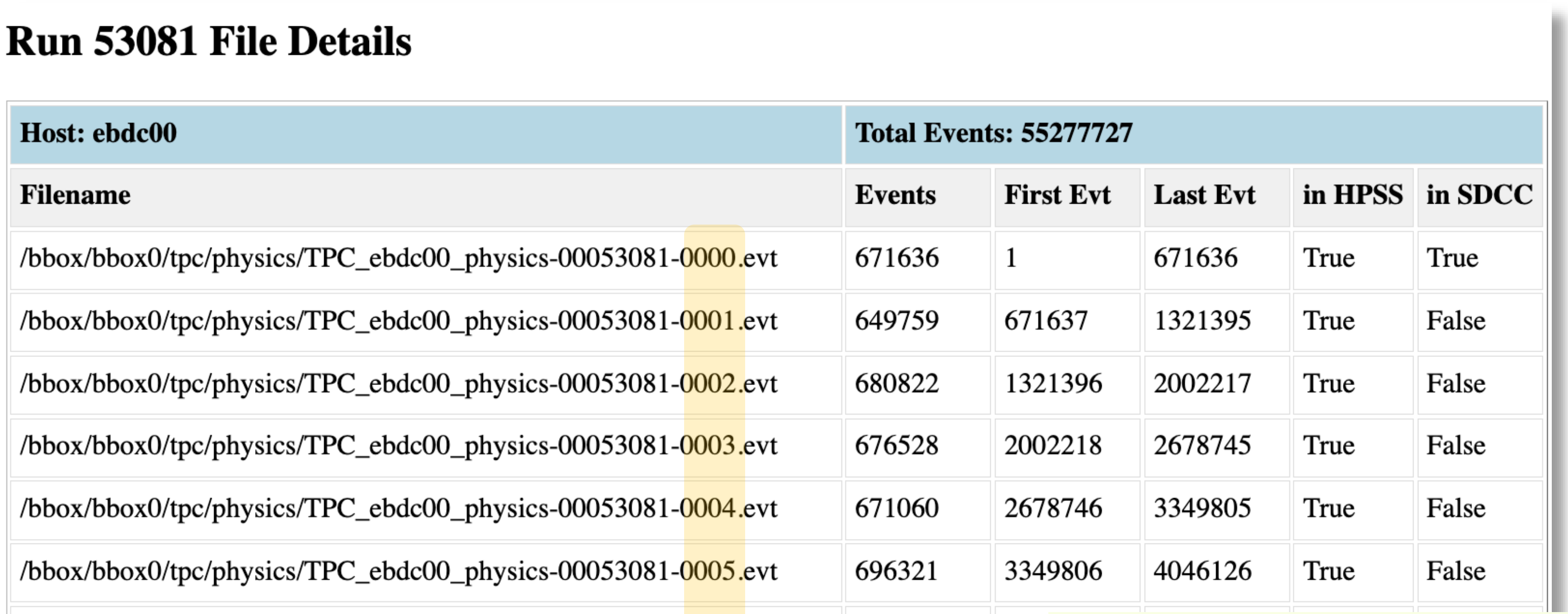
Data logging

Each RCDAQ instance writes one output file at a given time

The files roll over after a prescribed size (typically 20GB) is reached

The data from one DAQ “run” typically consist of about 1500-2500 files

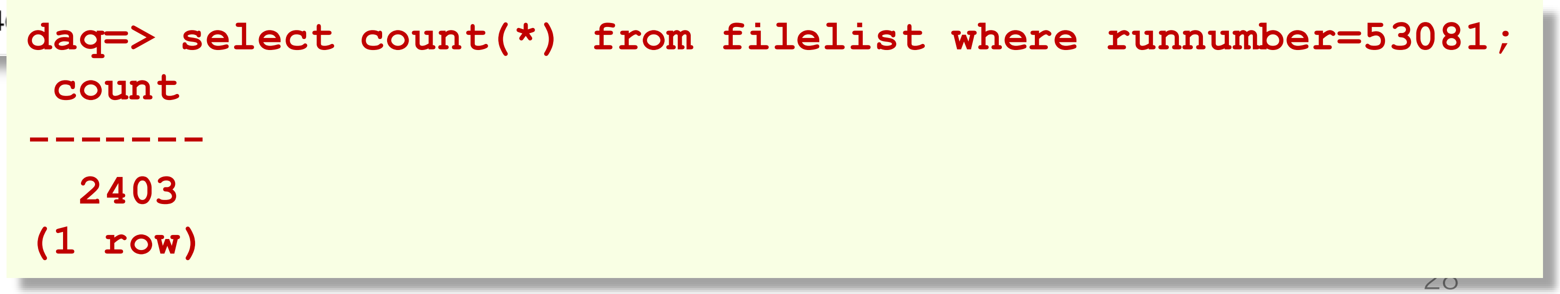
For reconstruction/analysis, those files need to get combined into the full detector response



Run 53081 File Details

Host: ebdc00	Total Events: 55277727				
Filename	**Events**	**First Evt**	**Last Evt**	**in HPSS**	**in SDCC**
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0000.evt	671636	1	671636	True	True
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0001.evt	649759	671637	1321395	True	False
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0002.evt	680822	1321396	2002217	True	False
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0003.evt	676528	2002218	2678745	True	False
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0004.evt	671060	2678746	3349805	True	False
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0005.evt	696321	3349806	4046126	True	False
/bbox/bbox0/tpc/physics/TPC_ebdc00_physics-00053081-0006.evt	706641	404			

The file rollovers from this particular RCDAQ instance

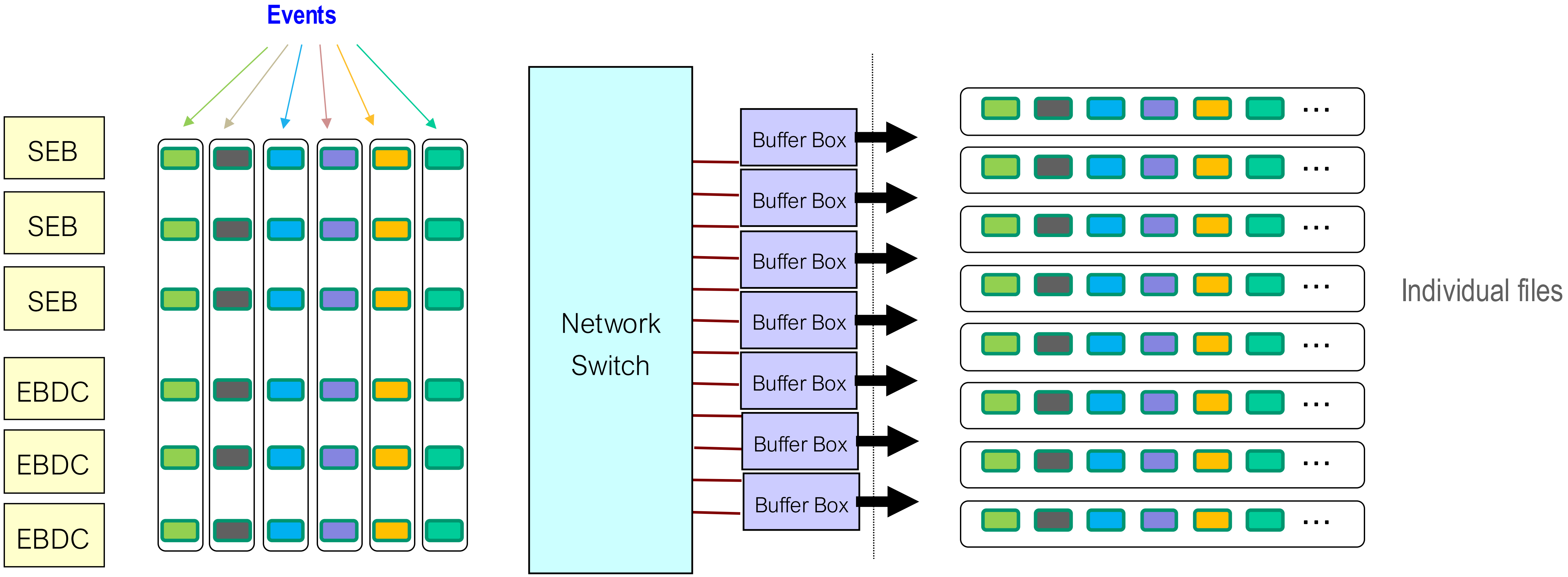


```
daq=> select count(*) from filelist where runnumber=53081;
 count
-------
   2403
(1 row)
```

No Event builder in sPHENIX

We are storing individual files at the SEB/EBDC level on central servers

This makes our operations a lot less risky, less moving parts, simpler software



For the reconstruction, one would need to combine about 60 files with the pieces of a given event

Online, we would do that for a fraction of event (like 10-50Hz worth) for onl. monitoring

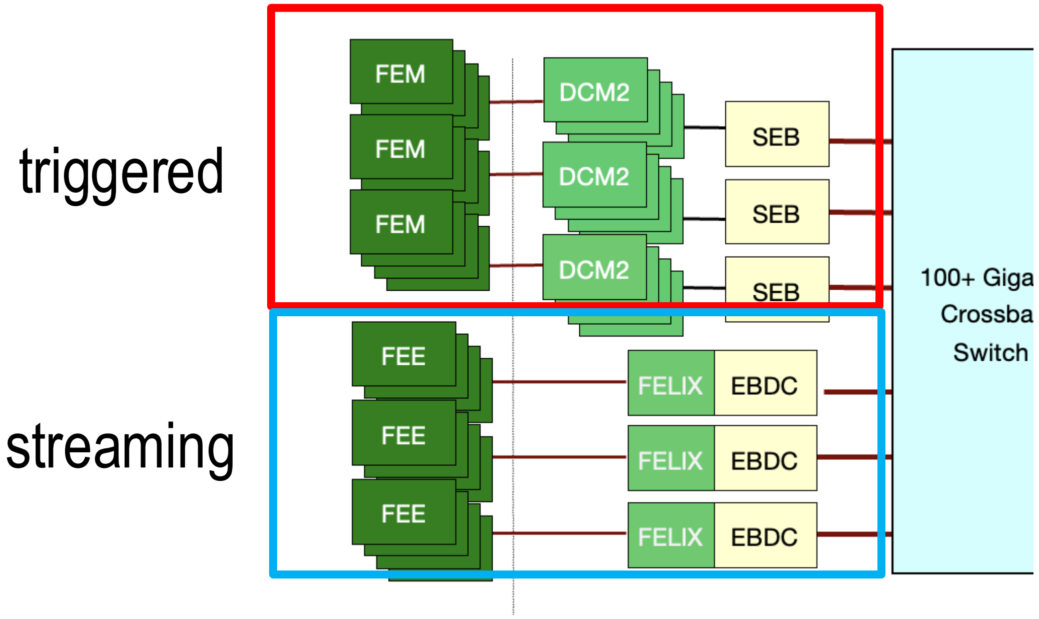
Streaming Readout

sPHENIX has a mix of triggered (Calorimeters, MBD, sEPD) and streaming (tracking system) readout

We distribute the unique Beam Clock Counter (40 bits of the full 64) to the SRO front-ends for each RHIC crossing (@9.4MHz)

The Trigger/Timing system (that acts as a detector in its own right) records ”all we need to know” – scalers, trigger input patterns, and, yes, the Beam Clock Counter (BCO)

This is used to align the SRO data with each other and to correlate the data with the calorimeters



Streaming readout, here we come!

Past the FEE, the readout is completely oblivious to the readout 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!

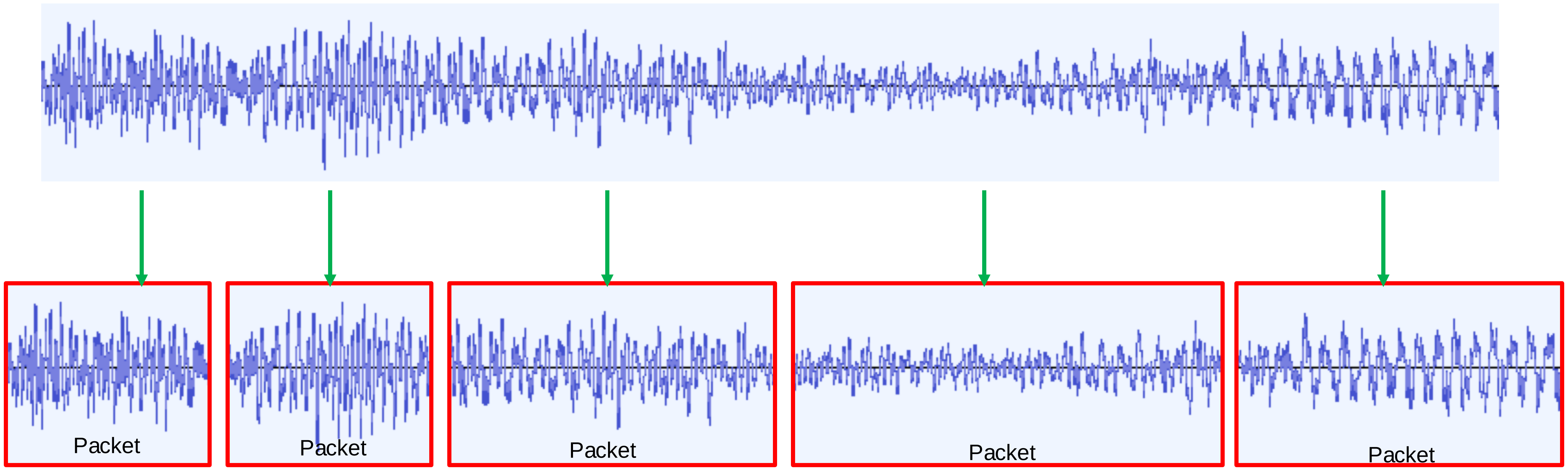
While "event" *n* is streaming, in other places, event *n-1* (or *-2, -3, -4…*) isn't finished yet, and data from different crossings are interleaved

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

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)
$
```

On Autopilot - Scripts at work

Very often – especially in your R&D days – you want to step through a range of values of a configuration parameter and see what your detector prototype has to say

- Bias voltage scans (we characterized gazillions of SiPMs)
- Position scans
- Temperature scans
- And on and on

Such a measurement is best done in a script that reads predetermined positions / voltage settings / what have you and performs the measurement

I picked an example: What is the response uniformity of a calorimeter module when a shower develops in different places? (We were very worried about this)

We were simulating different shower positions by “writing light with a light fiber” on the module front face

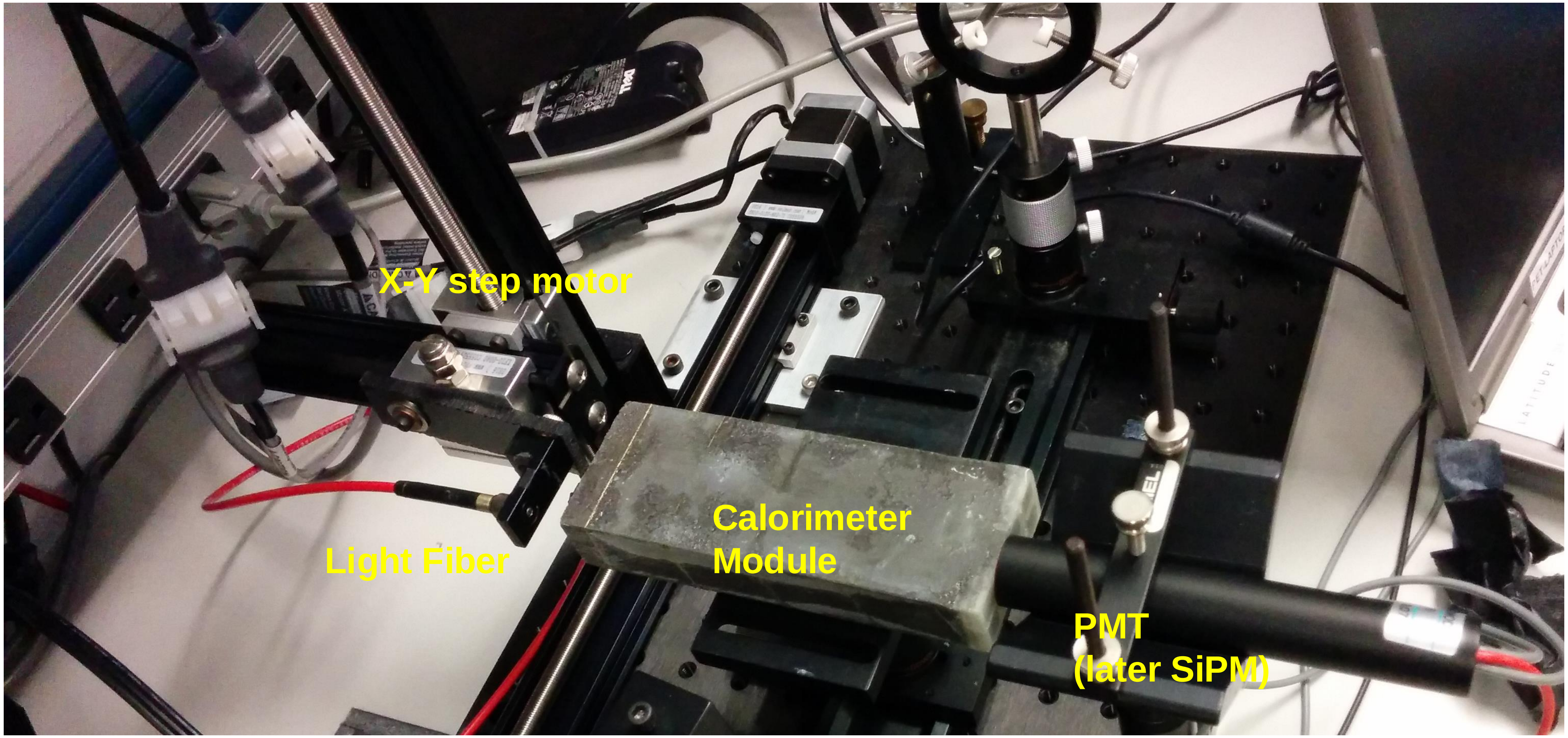
Measurements on autopilot through scripting

Simulate shower incidence positions by moving a light fiber in x and y

take a run for each position w/ 4000 events

50 x 25 = 1250 positions (later we had 60x60, 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

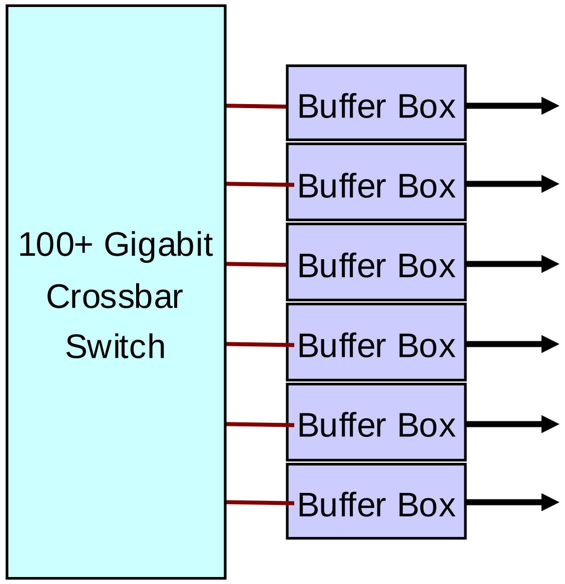
move the x motor → quickmove.sh $CURRENTPOSX 1

start the DAQ → rcdaq_client daq_begin

next x → done

next y → done

Why do we call those “BufferBoxes”?



100+ Gigabit Crossbar Switch

Buffer Box

Buffer Box

Buffer Box

Buffer Box

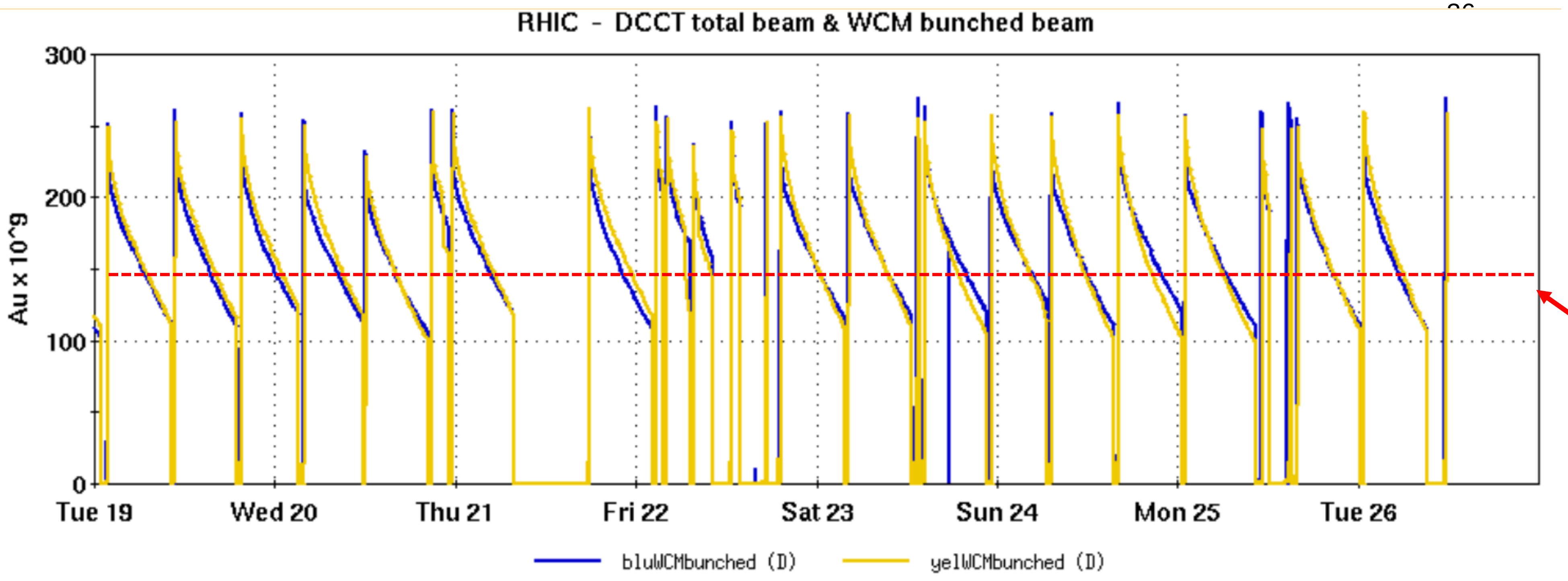
Buffer Box

Buffer Box

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

A typical RCDAQ Setup Script

```
#! /bin/sh
# this sets up the DRS4 readout with 5GS/s, a negative
# slope trigger in channel 1 with a delay of 140
```

We comment a lot as a way of documentation

```
if ! rcdaq_client daq_status > /dev/null 2>&1 ; then
     echo "No rcdaq_server running, starting..."
     rcdaq_server > $HOME/rcdaq.log 2>&1 &
     sleep 2
fi
```

If no server is running, we start one here.

```
MYSELF=$(readlink -f $0)
```

We convert the script filename into a full path

```
rcdaq_client daq_clear_readlist
```

We clear all existing definitions

```
rcdaq_client create_device device_file 9 900 "$MYSELF”

rcdaq_client load librcdaqplugin_drs.so
rcdaq_client create_device device_drs -- 1 1001 0x21 -150 negative 140 3
```

We load the plugin(s) and define the device(s)

Here is the actual setup script for our TPC (FELIX)

Abridged version, just the essentials

```
#!/bin/bash

RunType=beam
H=$RCDAQHOST
[ -z "$H" ] &&  H=$(hostname)

MYSELF=$(readlink -f $0)
rcdaq_client daq_clear_readlist
rcdaq_client create_device device_file 9 900 "$MYSELF"

rcdaq_client load   librcdaqplugin_dam.so
rcdaq_client create_device device_dam 1 4${H:4:2}1 1 128

rcdaq_client daq_set_runcontrolmode 1
```