# Streaming readout for an EIC detector

Chris Cuevas – presenting

Graham Heyes – wrote the talk (didn't get any gelato)

Streaming readout workshop, GENOA

May 23rd to 24th 2019

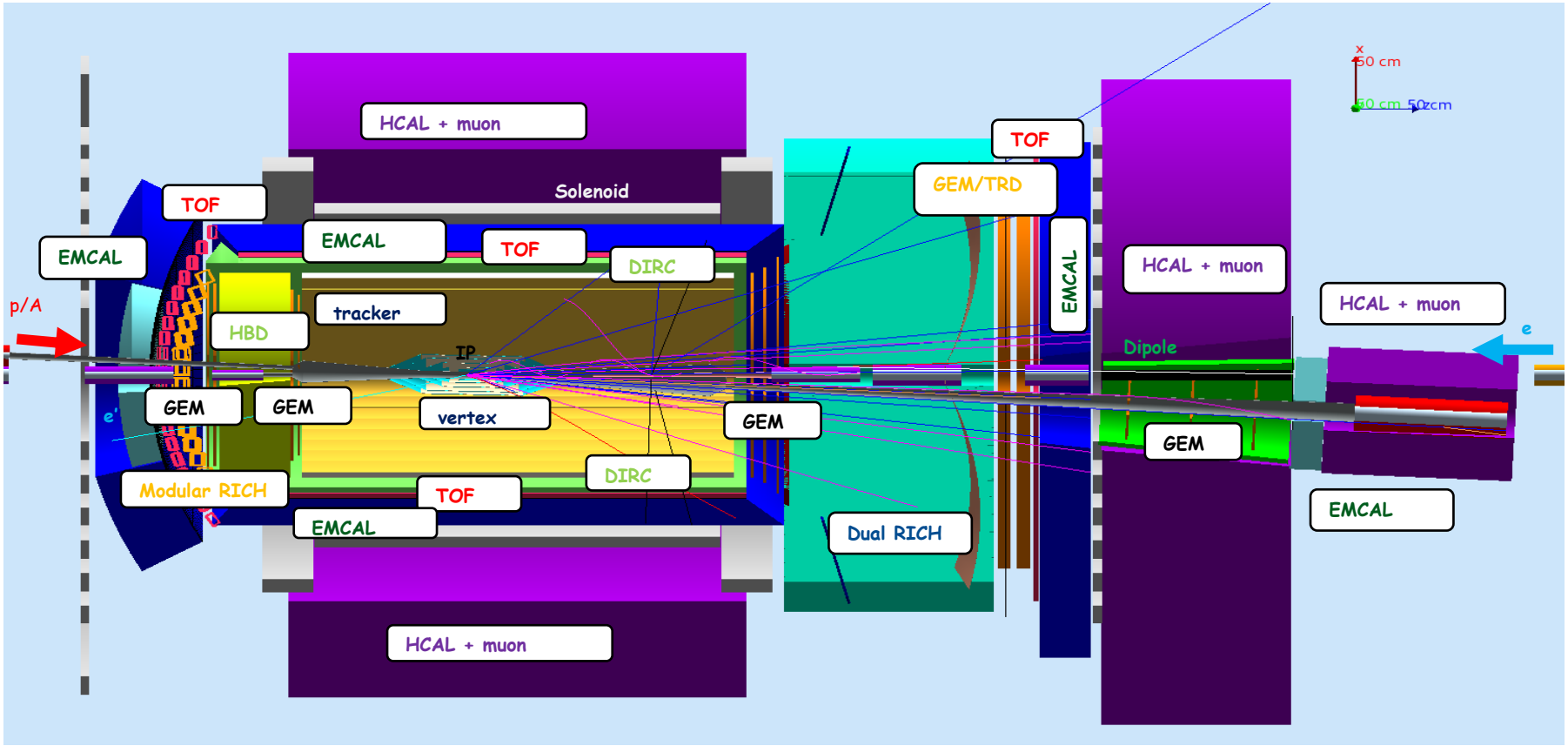
Jefferson Lab

# Introduction

- Example on an EIC detector design.
- Why not use traditional pipelined triggered DAQ?
- Other considerations than just rates.
- How does streaming readout help?
- Example of a DAQ architecture.
- Dealing with the vertex detector
- Won’t it cost a lot?
- Where are we now?
- Summary.

# Example EIC detector design

- Just counting labels on the diagram there are ~25 detector packages.
  - Wide range of response times for the detector types.
- The largest single channel count is the Vertex Detector.



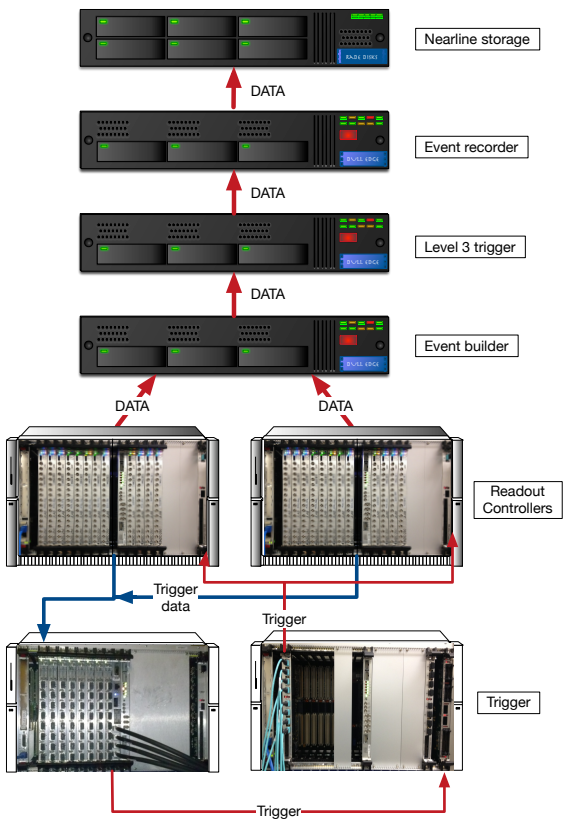

Jefferson Lab

# Channel counts

- Channel count is dominated by Vertex Detector
  - Forward detectors ~120 k ch.
  - PID ~700 k ch.
  - HCAL ~20 k ch.
  - EMCAL ~15 k ch.
  - Tracking ~150 k ch.
  - Vertex 20-50 M ch.
- Non-Vertex Detector channel count ~1M channels.

# What happens if we use traditional DAQ - crates

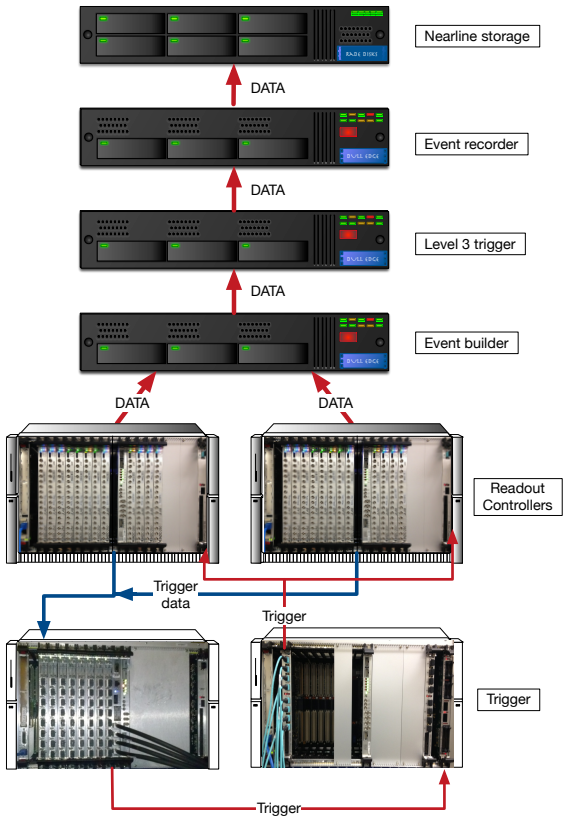
- Back of the envelope calculation ignoring the elephant in the room, the Vertex detector.
- The rest of the detector is ~1M channels.
  - CLAS12 : ~90k channels read by 100 ROCs
  - GLUEX : ~40k channels read by 50 ROCs
  - Average ~1 ROC per 1000 channels, seems like a lot of channels per ROC but is dominated by high channel count detectors.
  - EIC detector would be ~1,000 ROCs.
- Here a “ROC” is abstract, could be a real crate or could be something that interfaces with several detector mounted cards.
  - We need to distribute triggers to 1000 devices.
  - We could have up to 1000 devices contributing signals to the trigger.
- Don’t forget the elephant.




Jefferson Lab

# What happens if we use traditional DAQ - rates

- Beam crossing frequency  500 MHz
  - Interaction rate 20MHz – 50MHz.
  - Assume trigger survival rate 100 kHz (Similar to GLUEX) = factor of 500 cut.
- Assume average 1% occupancy.
  - Vertex detector rate ~240 GB/s. (yes bytes)
  - Rest of the detector ~5 GB/s total.
    - Fair agreement with CLAS12 and GLUEX if we were to scale them up to 100 kHz and 1% of 1M channels.
- 5 GB/s is a lot of data but manageable with four or five event builders running in parallel.
- Don't forget the 240 GB/s elephant though.




Jefferson Lab

# OK, so about the elephant

- How would you read out a detector that generates 250 GB/s ?
- The only way that even remotely makes sense is massive parallelism.
  - Split the detector into small regions and read those out in parallel.
  - No sensible way to sync the different regions in real time without spending a lot on electronics.
- Aim to reduce the rate to storage by using data from the rest of the detector to define regions of interest.
  - Have to hold on to the Vertex Tracker data until R.O.I. can be identified.
- Dealing with the Vertex Tracker dominates the design of the DAQ.

# Other considerations

- In a triggered DAQ we rely on:
  - All subsystems getting the trigger and staying in sync.
  - All subsystems being operational for the entire run.
- Often one of the hardest parts of operating a DAQ is starting and ending data taking gracefully. The difficulty scales with the number of things that have to be ready before you start.
- The frequency of undesirable events grows with the size of the system and trigger rate – long data taking runs end with a crash.
- The mitigating factors that make a large system reliable are frequently ones that also slow it down.
- A large percentage of diagnostic beam time is often labeled “trigger studies”. Triggers at these rates are hard to implement.

| Name | Message | Time | Severity |
|---|---|---|---|
| rcGui-83 | Reset issued. | 19:35:51 02/27 | INFO |
| sms_hd_all.tsg | reseted is started. | 19:35:54 02/27 | INFO |
| sms_hd_all.tsg | CodaRcDownload service failed. | 19:35:54 02/27 | ERROR |
| rcGui-83 | Configure is started. | 19:36:03 02/27 | INFO |
| sms_hd_all.tsg | Download is started. | 19:36:14 02/27 | INFO |
| sms_hd_all.tsg | Done process = hd_all.tsg_PRE_SYNC | 19:36:14 02/27 | INFO |
| sms_hd_all.tsg | Done process = hd_all.tsg_GO_SYNC | 19:36:14 02/27 | INFO |
| ROCTRIG2 | Unable to initialize TI board (client msg) | 19:36:22 02/27 | ERROR |
| ROCTRIG2 | Wrong TI FIBER MEASUREMENT: DELTA T = -3 (client msg) | 19:36:22 02/27 | ERROR |
| ROCTRIG2 | ERROR detected while executing the User Download proceedure (check cMsg log) (client msg) | 19:36:22 02/27 | SEVERE |
| sms_hd_all.tsg | waiting for... ROCFCAL12, ROCFCAL11, ROCFCAL10, ROCSTPSC1, ROCTAGMH, ROCTAGM1, ROCST, ROCFD... | 19:36:27 02/27 | WARN |
| sms_hd_all.tsg | waiting for... ROCTAGMH, ROCFDC14, ROCFDC13, ROCBCAL3, ROCBCAL9, ROCBCAL6, ROCTOF2, ROCBCA... | 19:36:32 02/27 | WARN |
| sms_hd_all.tsg | waiting for... ROCTAGMH, ROCFDC14, ROCFDC13, ROCBCAL3, ROCBCAL9, ROCBCAL6, ROCTOF2, ROCBCA... | 19:36:37 02/27 | WARN |
| sms_hd_all.tsg | waiting for... ROCTAGMH, ROCFDC14, ROCFDC13, ROCBCAL3, ROCBCAL9, ROCBCAL6, ROCTOF2, ROCBCA... | 19:36:42 02/27 | WARN |
| sms_hd_all.tsg | waiting for... ROCTAGMH, ROCFDC14, ROCFDC13, ROCBCAL3, ROCBCAL9, ROCBCAL6, ROCTOF2, ROCBCA... | 19:36:47 02/27 | WARN |
| sms_hd_all.tsg | waiting for... ROCTAGMH, ROCFDC14, ROCFDC13, ROCBCAL6, ROCTOF2, ROCBCAL12, ROCFDC4, ROCFDC... | 19:36:53 02/27 | WARN |

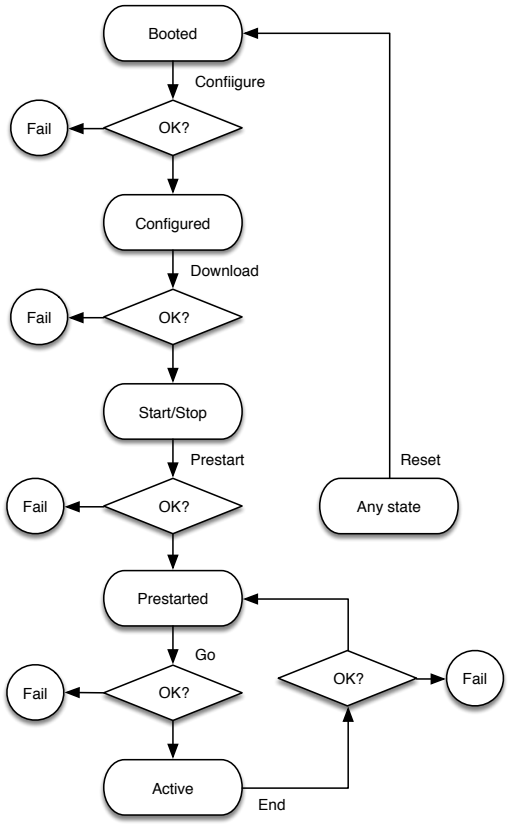
# Before the words Streaming Readout were uttered.

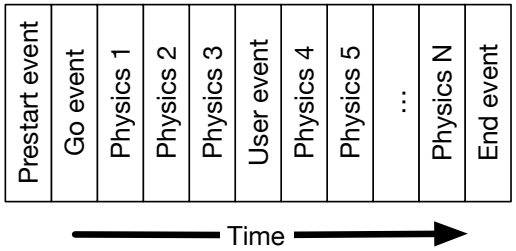
- The CODA DAQ system at JLab is governed by a state machine.
  - Driven by commands issued by Run Control to every software and hardware component.
  - On some state transitions ROCs insert maker events in the data.
  - Downstream components will not complete a transition until they see the matching event.

- Problem - A state change requires both a command from run control and correct marker event from upstream components.

- Solution:
  - Trigger marker events by Trigger Supervisor hardware just like a regular event.
  - Make components virtually stateless, RC no longer issues commands to anything except the trigger.
  - DAQ data flow is controlled by the data itself.
    - i.e. EB performs it's prestart tasks when it receives the prestart event not via a prestart command.
  - This leads to a much simplified system.

- This is natural for a streaming DAQ.



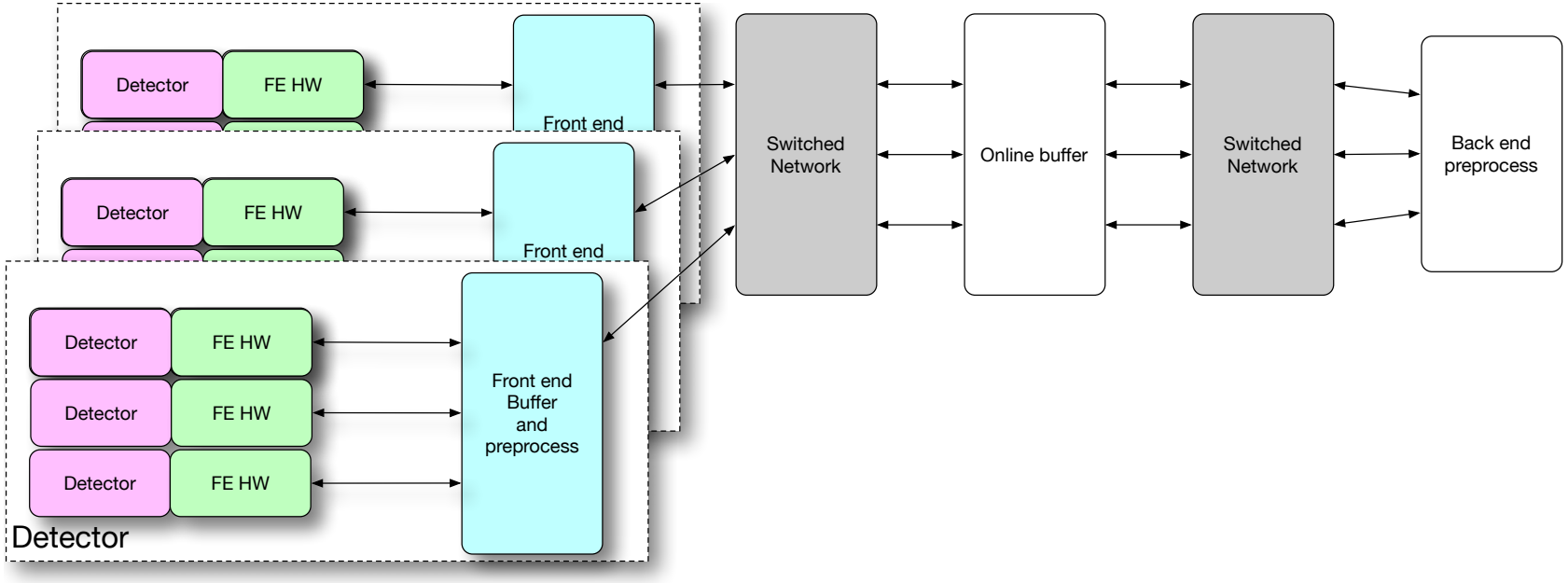




Jefferson Lab

# How does streaming RO help?

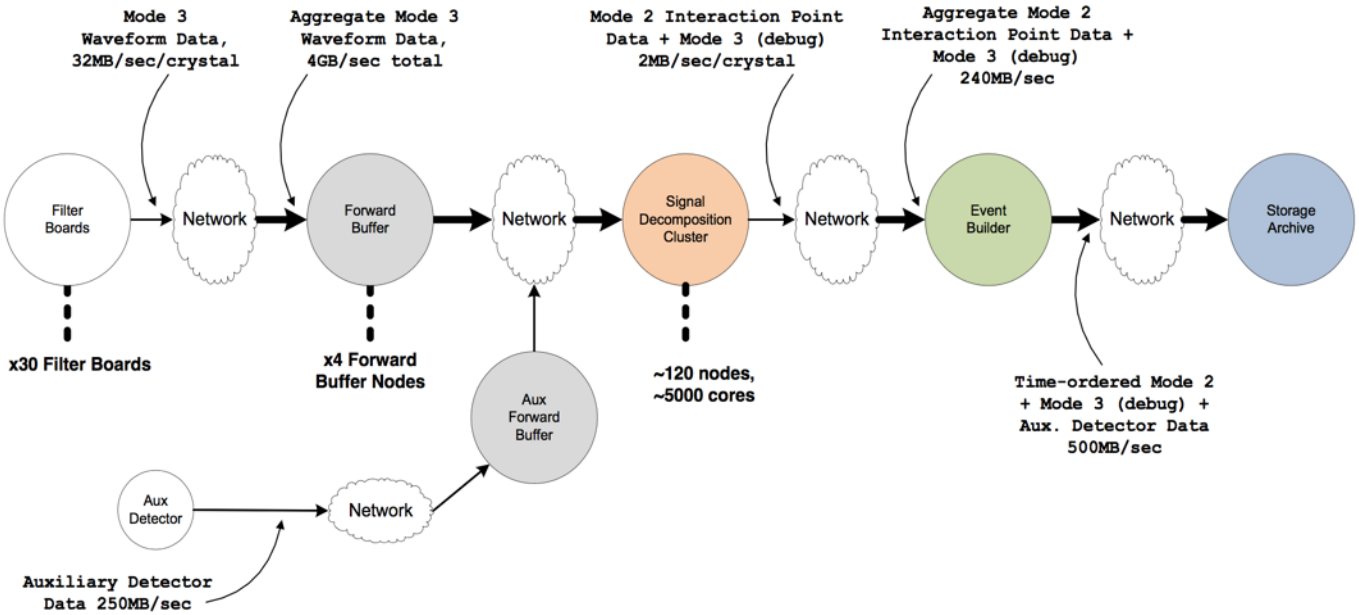
- In a streaming system the trigger is minimal and mostly at the detector level to reduce noise.
  - Many sync problems disappear since we do not demand that all detectors participate in a trigger.
  - We still need timing but this is a simpler problem. One board losing timing does not stop the whole DAQ.
- Since streams are essentially independent…
  - Problems that would otherwise cause us to end a run are confined to one stream. We could recover from problems without ending a run – important in a very large system.
  - New detectors can be added and debugged in parallel without impacting data taking.
- Streaming readout is driven by a clock distribution system this gives hardware control of data flow and leads to a much simpler run control system.

# Putting it all together



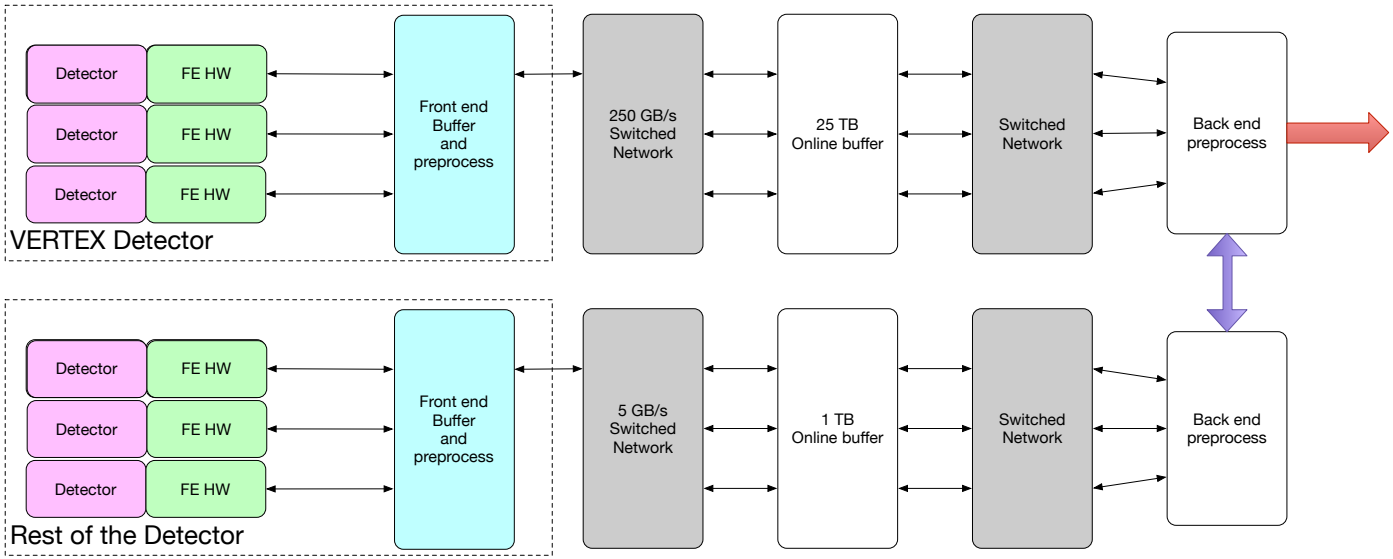

- For a single detector
  - Front end hardware on detector digitizes signals.
  - Point-to-point fiber or copper links to Front End buffer/preprocessor.
    - Fixed latency, prevents data loss.
    - Derandomizes after per channel zero suppression.
    - Further zero suppress, formatting, compression, etc.
- Multiple detectors are connected via switched network.
- Processing on a back end cluster with access to buffered data.

# Greta example




- Greta at FRIB – design using streaming readout well underway.
- 120 UDP streams, each corresponding to a detector crystal,
  - Aggregate (maximum) rate is 4 GB/s.
  - Possibility of load asymmetry of up to 7:1
- Concept is more or less identical.

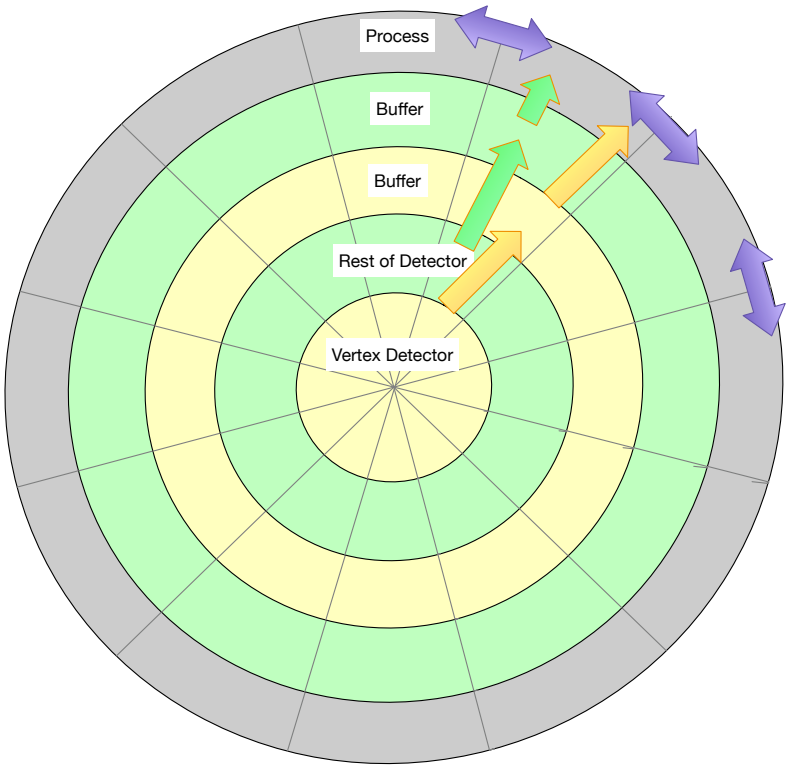
# Dealing with the Vertex detector




- Vertex Detector is read in parallel streams into online buffers.
  - Say 25 front end buffers at 10 GByte/s (Using today's 100 Gbit/s HW).
  - Main Online buffer is 25 nodes with 1TB of memory each ~100s buffer time.
- Rest of detector is read in parallel streams to a smaller online buffer.
  - Say 5 GB/s total – single 1TB buffer ~200s buffer time.
- Process data from rest of detector to identify regions of interest in Vertex Detector.
  - 4 D regions = 3D volume in detector and a timestamp range.
- Send regions and associated data from rest of detector to Vertex Detector back end processors that pull data from Vertex online buffer. Unwanted vertex Detector data is discarded – much reduced rate out.

# Alternative view

- An alternative view of the design is a segmented readout.
  - Data from each segment is confined in its own streams.
  - Each stream is buffered in its own set of buffers.
- The processing layer contains processing node that:
  - Process data from buffers in the same segment.
  - Communicate with processing nodes to the left and right. (For example to follow tracks that curl across several segments.
- Actual topology can vary.



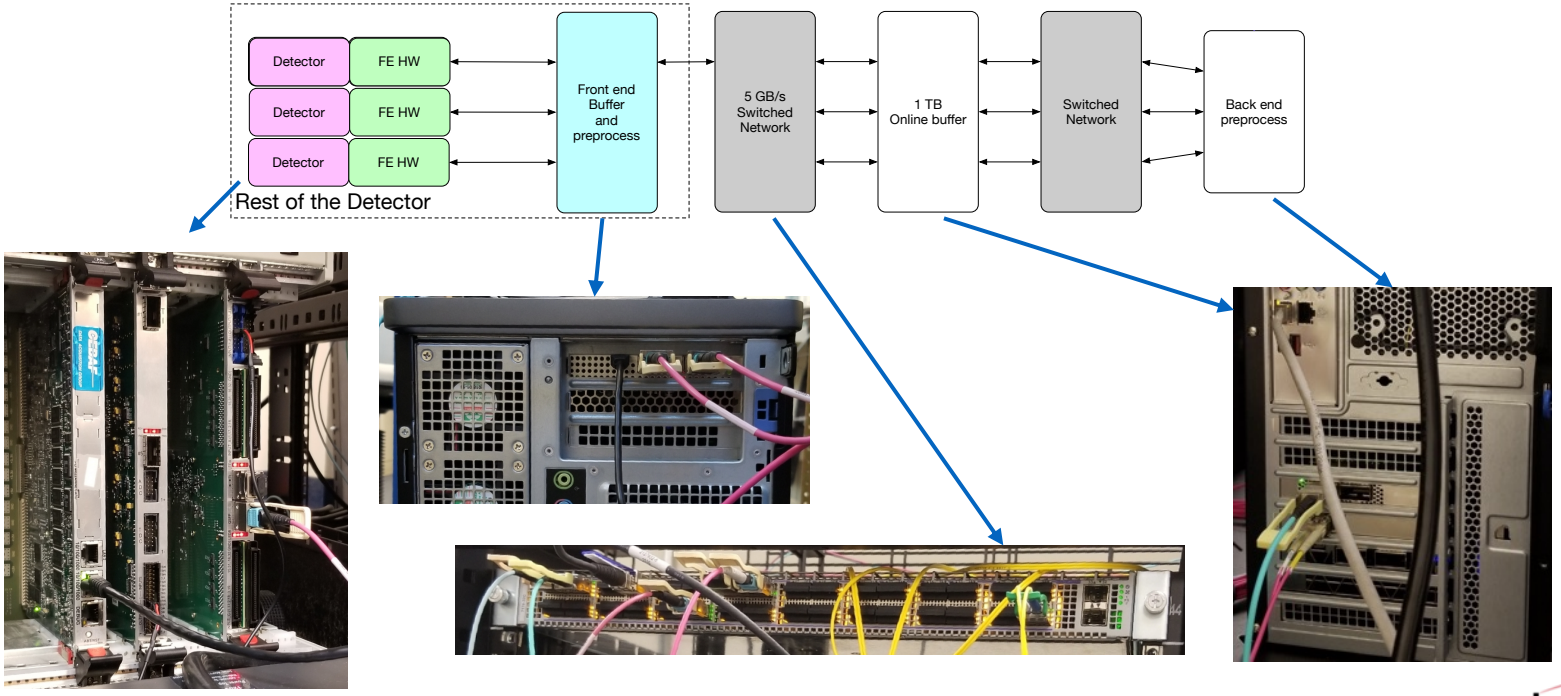
# Isn't it going to cost a lot?

- There is no global trigger hardware.
- The detector front-end hardware is needed no matter what the DAQ architecture.
- What are called front end buffer preprocessor cards are essentially the VTP cards used in the GLUEX and CLAS12 DAQs. These would be needed in a conventional DAQ.
  - We would add more input ports, higher bandwidth and deeper buffering.
  - We are also looking at commercial options.
- Networking switches able to handle EIC rates are commercially available now, cost should come down and performance go up.
- We can buy compute nodes with high bandwidth and 1TB of random access storage for about $15k now.
  - In a regular DAQ we would be buying nodes for event building.
- The processing layer nodes are essentially a Level 3 farm.
  - We would have this no matter what the DAQ architecture.

- The existence of commercially available, and affordable, networking and computing hardware, plus reliable software support, that is making streaming readout attractive. In the past a trigger was required to cut rates to something that affordable hardware could handle.

# So where are we now?

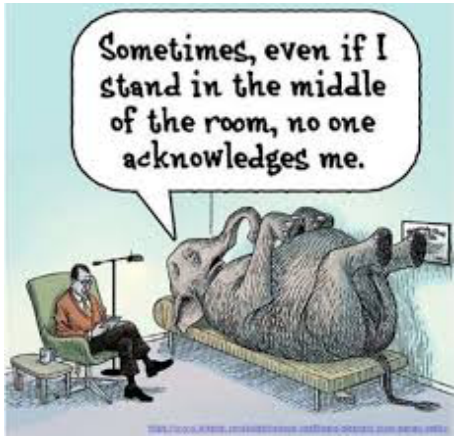
- In the DAQ design presented a few slides ago key elements are.
  - A data source outputting on fiber.
  - A front end buffer with FPGA.
  - A high speed low latency network.
  - An online compute resource to buffer and process data.
- In the INDRA lab at JLab we have put together a test stand using
  - a VETROC TDC to provide a Front End data source
  - A Xilinx Kintex UltraScale FPGA KCU1500 PCI board as a front end buffer and preprocessing device.
  - A Linux PC, with 100 Gbit/s network link as the online buffer and back end processing node.
  - See William Gu's talk.



# Summary

- At the rates that we expect from an EIC detector the DAQ architecture has to be highly parallel. The back end hardware required is similar whether we stream or don't.
- Streaming advantages:
  - No complicated global trigger.
  - Parallel by default so high and scalable bandwidth.
  - Streams are effectively independent
    - Adds fault tolerance.
    - Decreases system complexity.
    - Allows detectors to be run independently.
  - System is data driven – less complex run control.
  - System is defined by software and configuration both of which can easily be changed.

# La fine




grazie per aver ascoltato la mia presentazione

Jefferson Lab