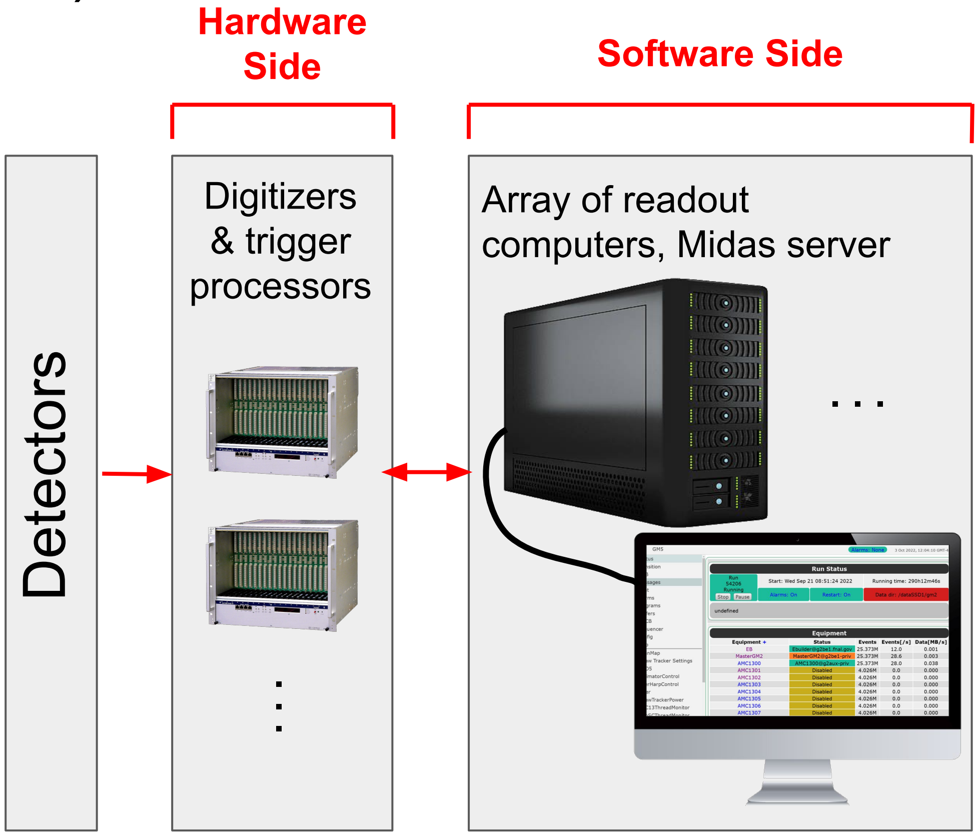
# DAQ Introduction

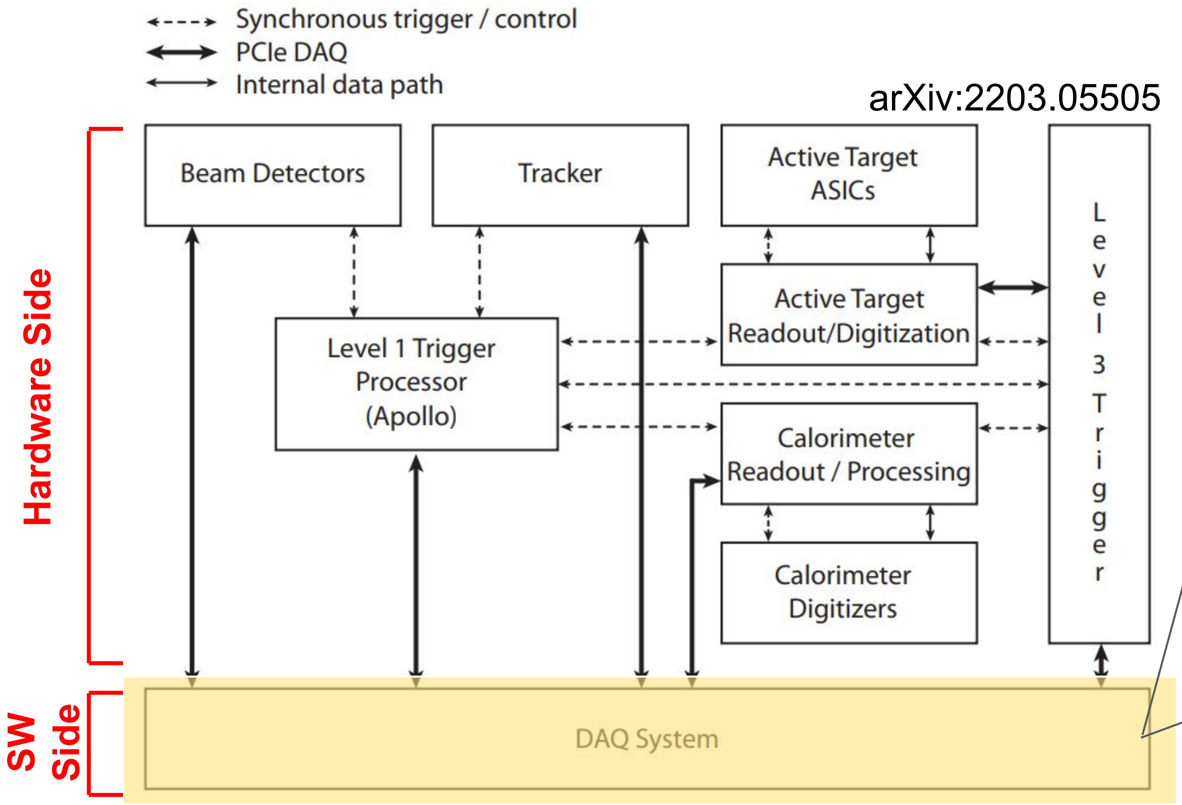
Jack Carlton
University of Kentucky

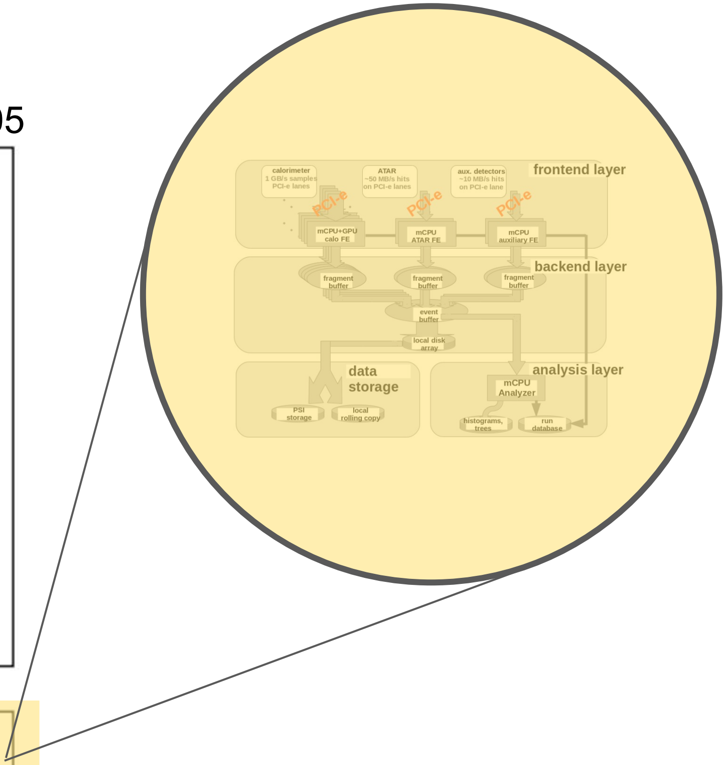
## What is Data Acquisition (DAQ)?

- **"DAQ" refers to the system of electronics used to convert analog signals from an experiment and package them into digital "events"**
  - Usually "DAQ" refers to the "software side", but sometimes refers to hardware as well
  - Hardware side also called "electronics"
- I like to differentiate between the software and hardware sides


Hardware Side
Software Side
Detectors
Digitizers & trigger processors
Array of readout computers, Midas server

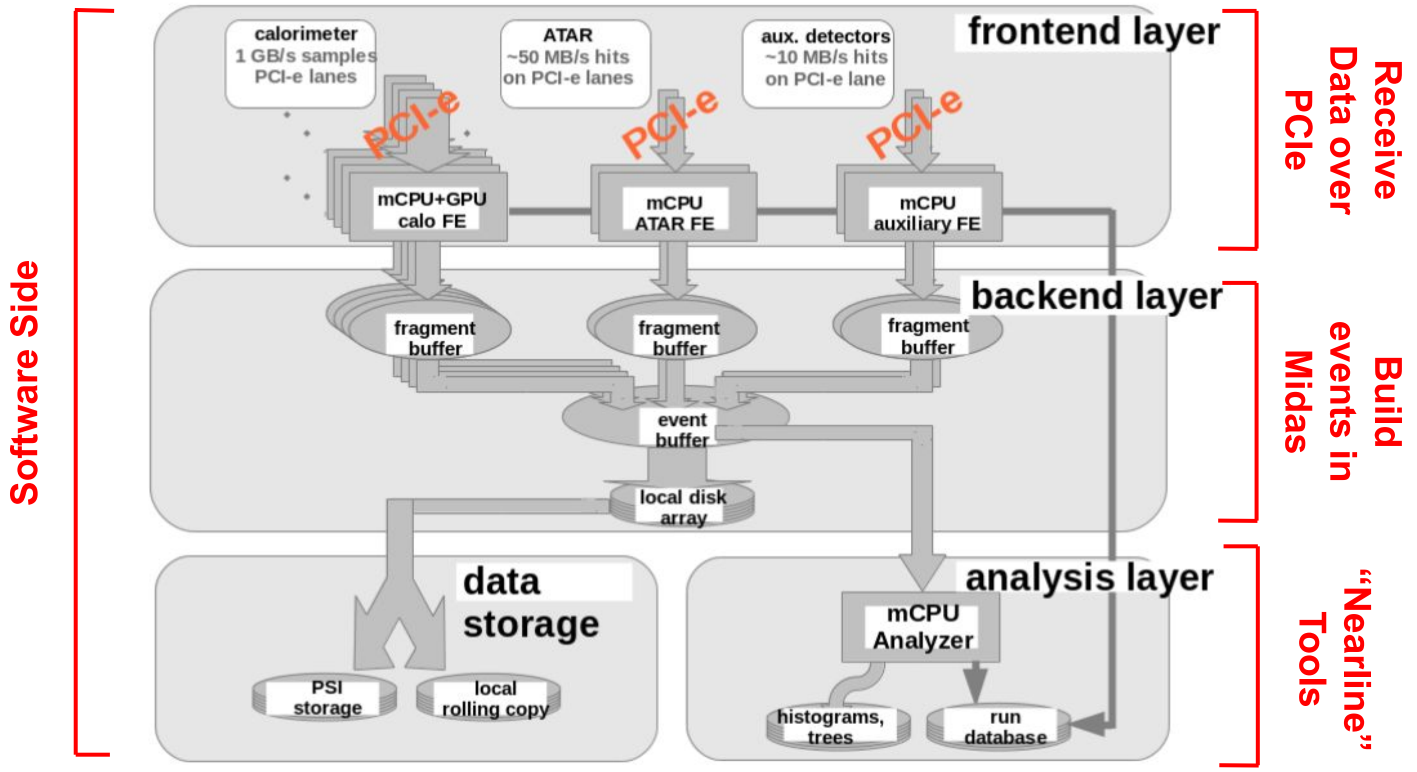
## Proposed Data Acquisition (DAQ) Framework





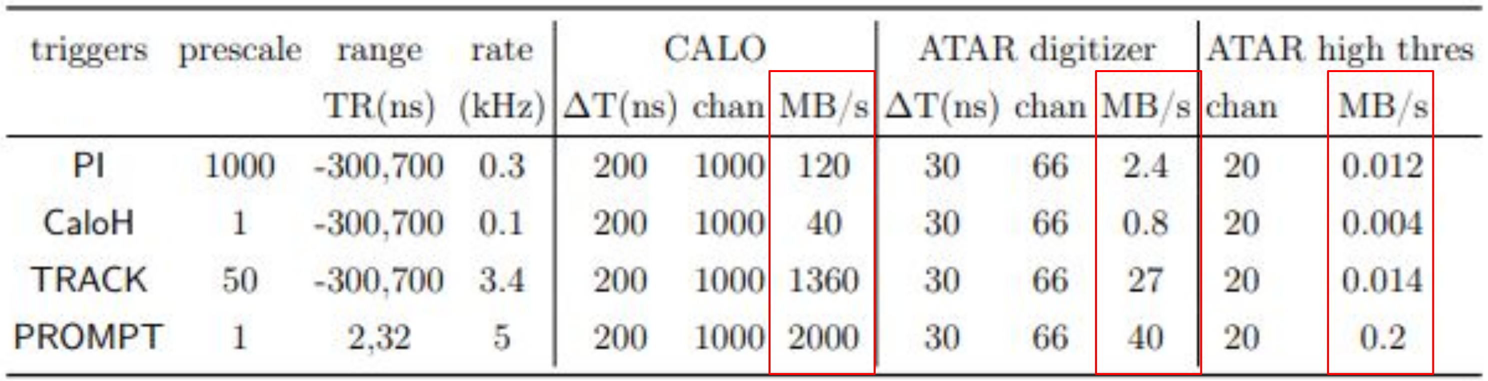
# Proposed Data Acquisition (DAQ) Framework

arXiv:2203.05505



## Data Rates

arXiv:2203.01981

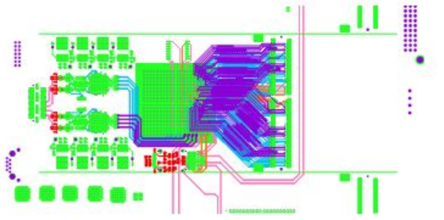


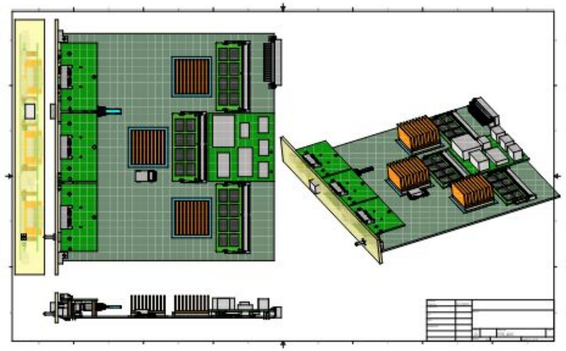
| triggers | prescale | range | rate | CALO | | | ATAR digitizer | | | ATAR high thres | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TR(ns) | (kHz) | ΔT(ns) | chan | MB/s | ΔT(ns) | chan | MB/s | chan | MB/s |
| PI | 1000 | -300,700 | 0.3 | 200 | 1000 | 120 | 30 | 66 | 2.4 | 20 | 0.012 |
| CaloH | 1 | -300,700 | 0.1 | 200 | 1000 | 40 | 30 | 66 | 0.8 | 20 | 0.004 |
| TRACK | 50 | -300,700 | 3.4 | 200 | 1000 | 1360 | 30 | 66 | 27 | 20 | 0.014 |
| PROMPT | 1 | 2,32 | 5 | 200 | 1000 | 2000 | 30 | 66 | 40 | 20 | 0.2 |

- PIONEER DAQ expects data rate of **~3.5GB/s**
- This is **~100,000 TB/year**
- How do we compress this in real time? (Not in this talk)
  - Fit data, store fit parameters
  - Compress and store residuals, throw some out
  - Graphics Processing Units (GPUs) used for this operation

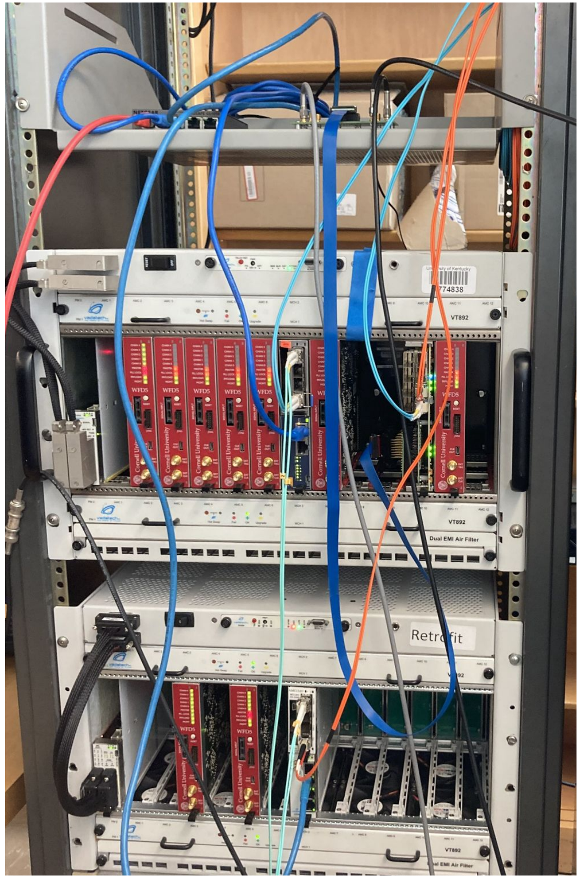
## Our Two DAQs

- g-2 modified DAQ
  - Modified for various experiments across the collaboration (test beam, LXe testing, LYSO testing, …)
- PIONEER DAQ
  - In nascent development state
  - Design catered to PIONEER full experiment necessities





**PIONEER ADC schematic drawings**



**UKY test stand MicroTCA crates**

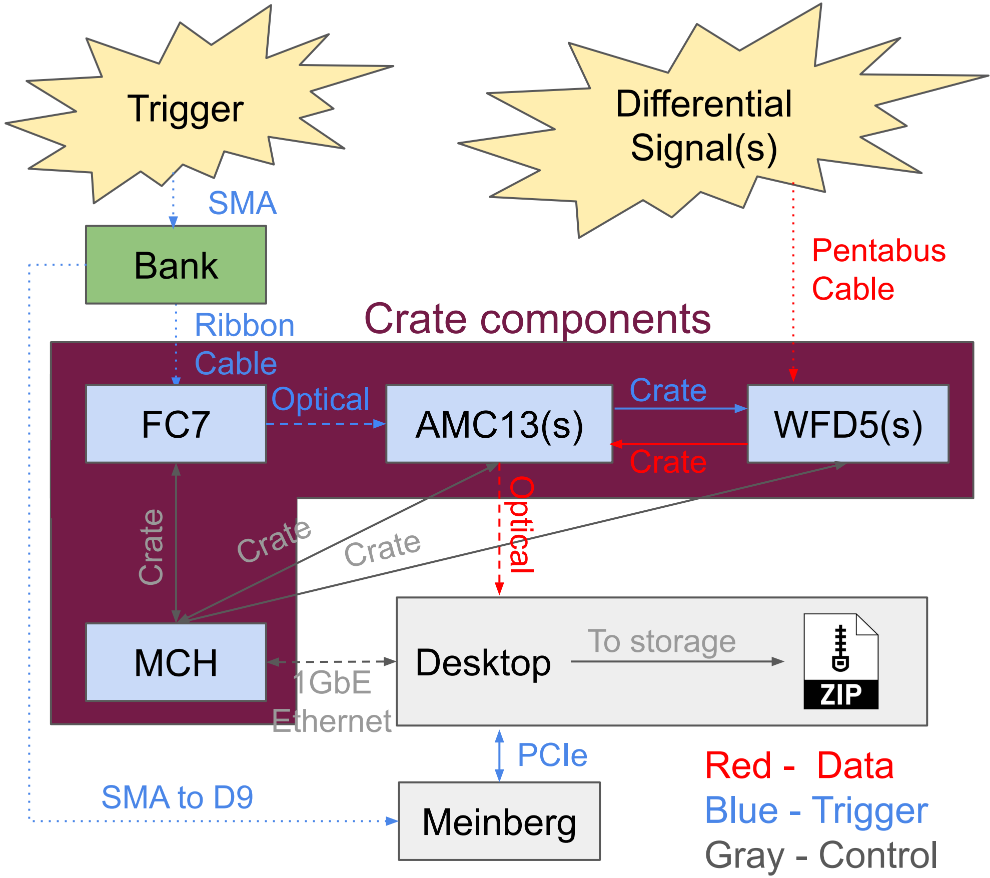
## What is a Field Programmable Gate Array (FPGA)?

- Commonly used for real time data processing
- Programmable
  - Typically use a software tool called Vivado
  - Typically programmed using Verilog or VHDL
  - Use low-level software called “firmware”
- Allows for fast, flexible control of logic signals to board components
- Building block in almost all of our hardware (WFD5s, FC7s, AMC13s)

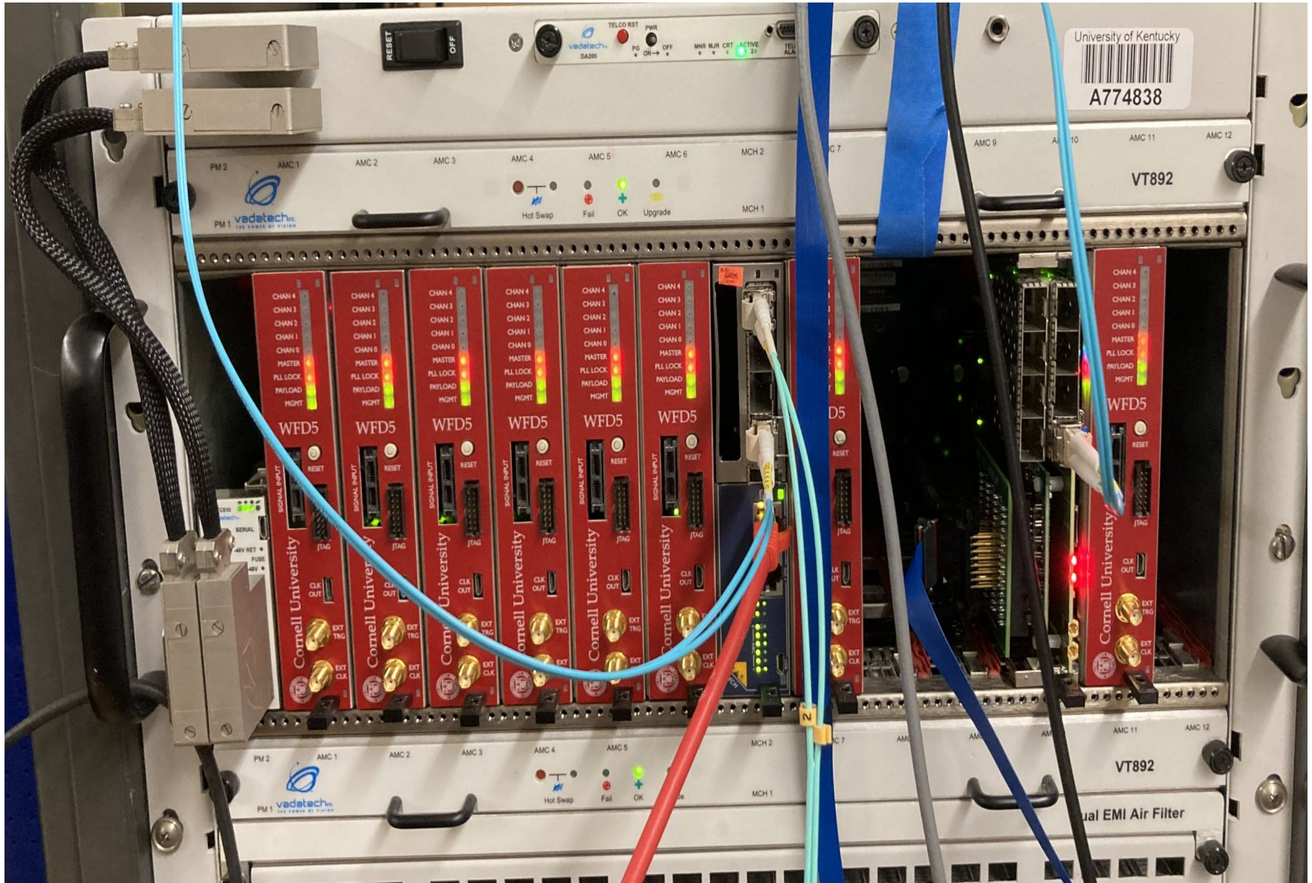
**A Xilinx Development Board with a XC6LX45T FPGA (Spartan-6)**

# Teststand DAQ Hardware (Modified g-2 DAQ)

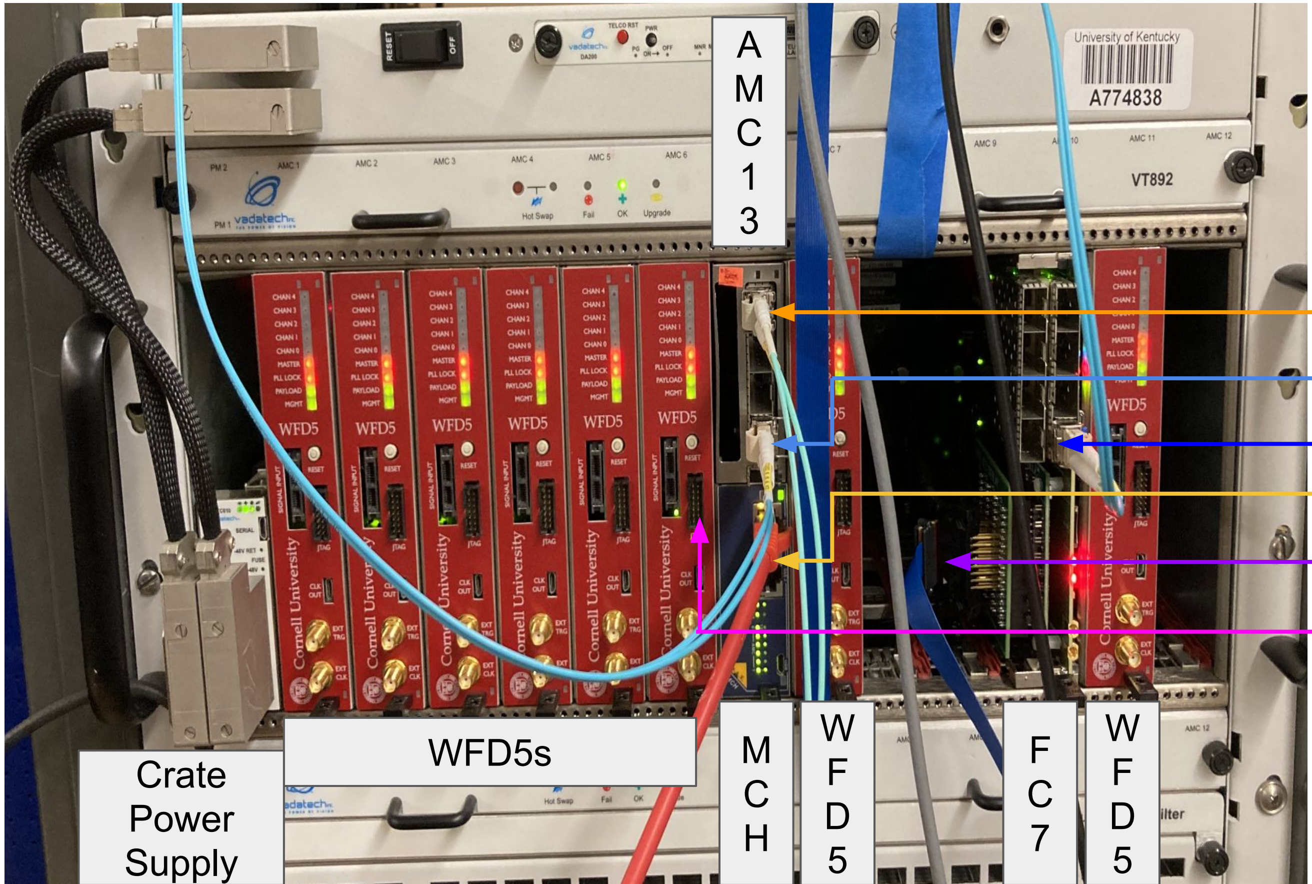
- Differential signal into WFD5 (Waveform Digitizer)
- Trigger signal into FC7 (Flexible Controller)
- AMC13 (Advanced Mezzanine Card) gathers data, sends over 10GbE (10 Gigabit Ethernet) to desktop
- MCH (MicroTCA Carrier Hub) facilitates Desktop↔Crate communication over 1GbE
- Desktop CPU handles event processing
- Meinberg gives trigger timestamp to computer



## Teststand DAQ Hardware (Modified g-2 DAQ)



# Teststand DAQ Hardware (Modified g-2 DAQ)



Note: AMC13 and MCH are half slot modules

10GbE out (data) AMC13→desktop

Trigger in AMC13

Trigger out FC7

1GbE MCH in/out (comm.)

FC7 Trigger in

WFD5 5-channel, differential signal in (no connection in this picture)

## PIONEER DAQ Hardware (In a Nascent State)

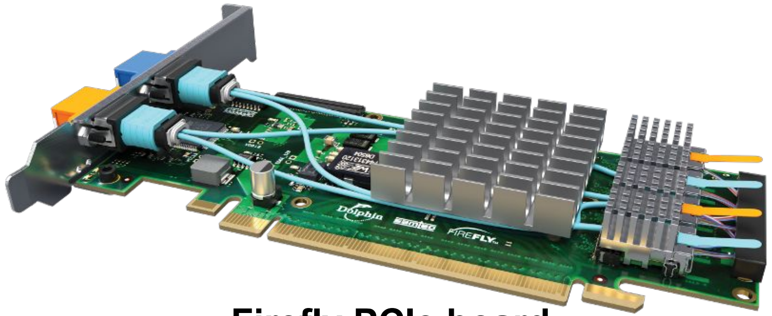
- Using APOLLO system (no more µTCA crates)
- Data is moved using “Firefly” optical flyover system
  - 25 gb/s > 10gb/s links from g-2
- Data received by desktop through Firefly PCIe cards

Command Module (CU)

+

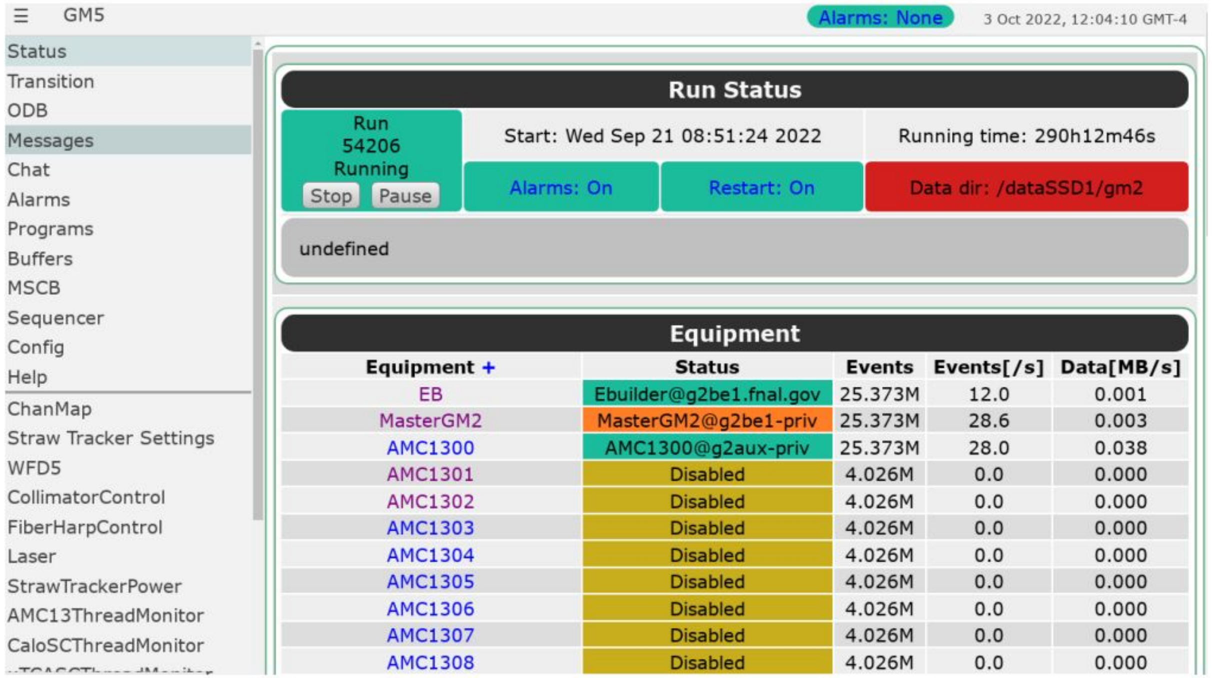
Service Module (BU)

=


**Firefly PCIe board**

## Midas Framework

- C/C++ (mostly) package of modules for
  - run control,
  - expt. configuration
  - data readout
  - event building
  - data storage
  - slow control
  - alarm systems
  - Etc.
- Can link with custom software



**Run Status**

| | | |
|---|---|---|
| Run 54206 Running | Start: Wed Sep 21 08:51:24 2022 | Running time: 290h12m46s |
| | Alarms: On / Restart: On | Data dir: /dataSSD1/gm2 |

undefined

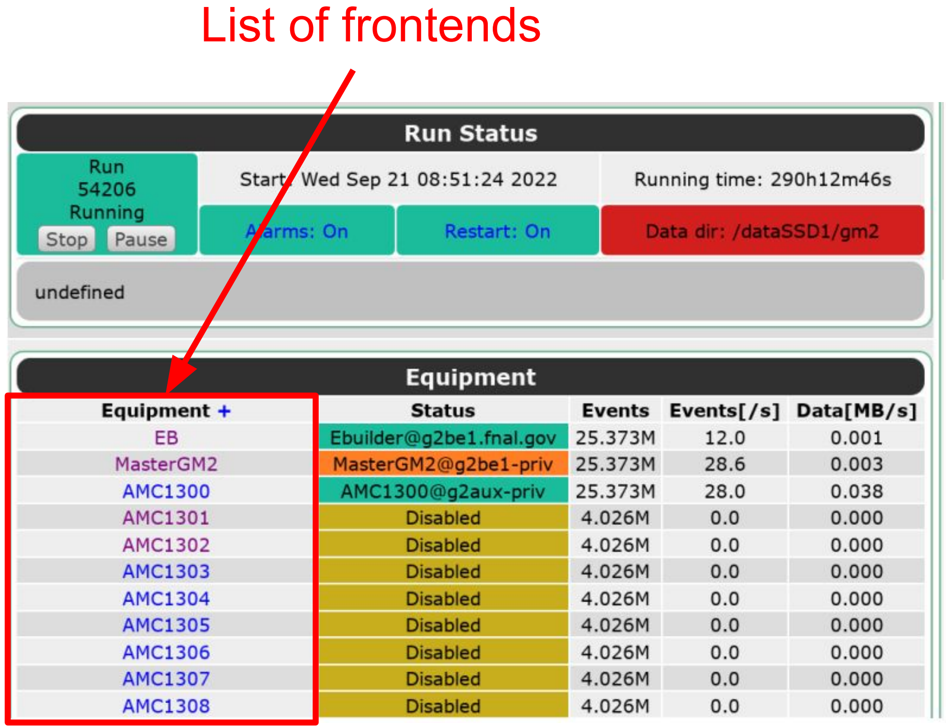
**Equipment**

| Equipment + | Status | Events | Events[/s] | Data[MB/s] |
|---|---|---|---|---|
| EB | Ebuilder@g2be1.fnal.gov | 25.373M | 12.0 | 0.001 |
| MasterGM2 | MasterGM2@g2be1-priv | 25.373M | 28.6 | 0.003 |
| AMC1300 | AMC1300@g2aux-priv | 25.373M | 28.0 | 0.038 |
| AMC1301 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1302 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1303 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1304 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1305 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1306 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1307 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1308 | Disabled | 4.026M | 0.0 | 0.000 |

**Example g-2 Midas Webpage**

# Midas Frontends

- C++ programs operating in the midas framework
- Typically handle receiving, processing, and packing data into midas events
- Simple example frontend



Run
54206
Running
Stop Pause

Start: Wed Sep 21 08:51:24 2022

Running time: 290h12m46s

Alarms: On

Restart: On

Data dir: /dataSSD1/gm2

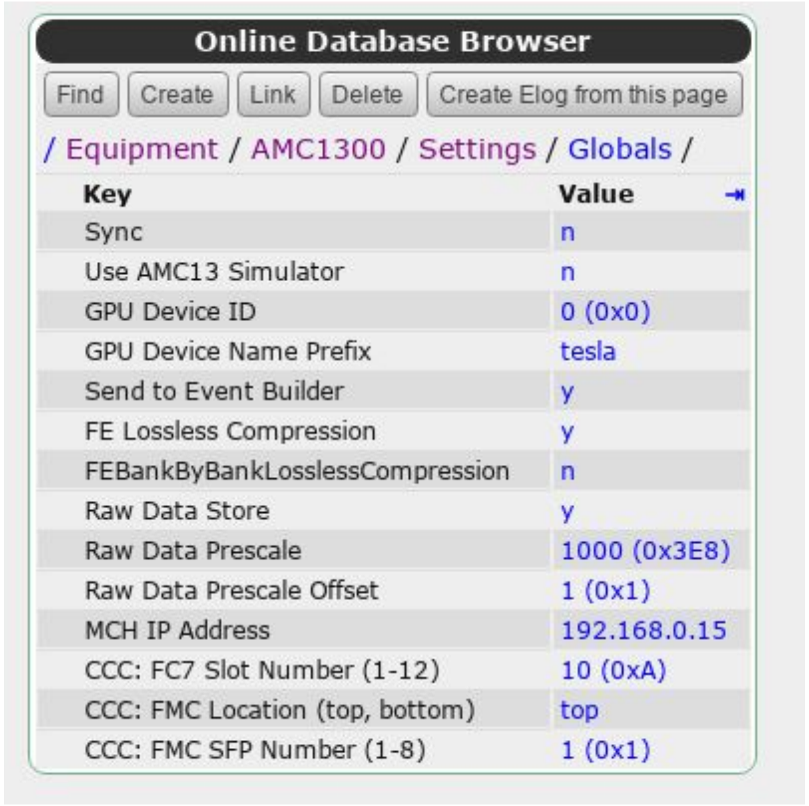
undefined

| Equipment + | Status | Events | Events[/s] | Data[MB/s] |
|---|---|---|---|---|
| EB | Ebuilder@g2be1.fnal.gov | 25.373M | 12.0 | 0.001 |
| MasterGM2 | MasterGM2@g2be1-priv | 25.373M | 28.6 | 0.003 |
| AMC1300 | AMC1300@g2aux-priv | 25.373M | 28.0 | 0.038 |
| AMC1301 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1302 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1303 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1304 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1305 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1306 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1307 | Disabled | 4.026M | 0.0 | 0.000 |
| AMC1308 | Disabled | 4.026M | 0.0 | 0.000 |

**Example g-2 Midas Webpage**

## Online Database (ODB)

- GUI on midas webpage
  - Also available command line
- Allows for “on the fly” adjustments between runs
- Built in configurations:
  - Midas webpage
  - Logger write location
  - Webpage update rate
  - Etc.
- Custom configurations
  - Configure hardware
  - etc.



/ Equipment / AMC1300 / Settings / Globals /

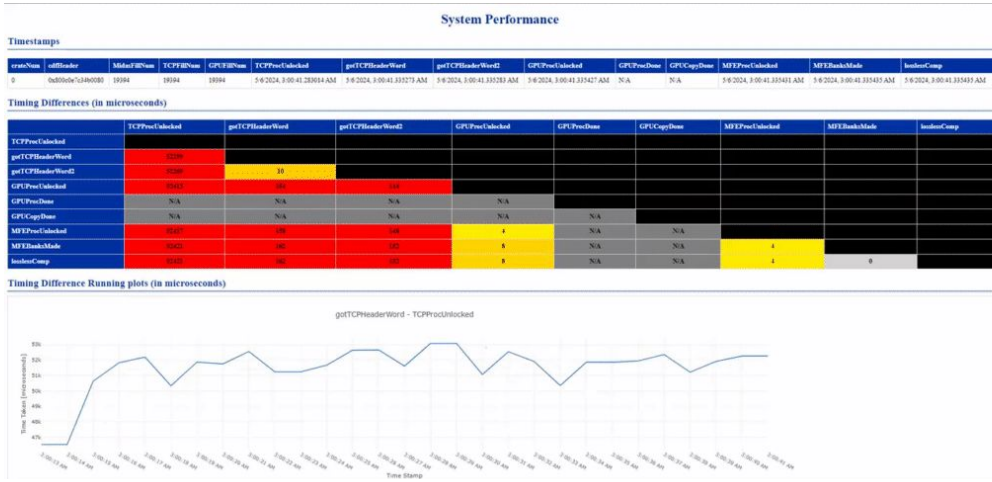
| Key | Value |
|---|---|
| Sync | n |
| Use AMC13 Simulator | n |
| GPU Device ID | 0 (0x0) |
| GPU Device Name Prefix | tesla |
| Send to Event Builder | y |
| FE Lossless Compression | y |
| FEBankByBankLosslessCompression | n |
| Raw Data Store | y |
| Raw Data Prescale | 1000 (0x3E8) |
| Raw Data Prescale Offset | 1 (0x1) |
| MCH IP Address | 192.168.0.15 |
| CCC: FC7 Slot Number (1-12) | 10 (0xA) |
| CCC: FMC Location (top, bottom) | top |
| CCC: FMC SFP Number (1-8) | 1 (0x1) |

**Example ODB Page on Midas Webpage**

# Custom Software

- Can write “clients” that connect to midas experiment
  - Python
  - C++

- Allows for user to write software to fit their needs, for example:
  - Data Quality Monitor
  - Offline analysis scripts
  - Automatic ODB management



System Performance

Timestamps

| crateNum | cdfHeader | MidasFillNum | TCPFillNum | GPUFillNum | TCPProcUnlocked | gotTCPHeaderWord | gotTCPHeaderWord2 | GPUProcUnlocked | GPUProcDone | GPUCopyDone | MFEProcUnlocked | MFEBanksMade | losslessComp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0x800c0e7c34b0080 | 19394 | 19394 | 19394 | 5/6/2024, 3:00:41.283014 AM | 5/6/2024, 3:00:41.335273 AM | 5/6/2024, 3:00:41.335283 AM | 5/6/2024, 3:00:41.335427 AM | N/A | N/A | 5/6/2024, 3:00:41.335431 AM | 5/6/2024, 3:00:41.335435 AM | 5/6/2024, 3:00:41.335435 AM |

Timing Differences (in microseconds)

| | TCPProcUnlocked | gotTCPHeaderWord | gotTCPHeaderWord2 | GPUProcUnlocked | GPUProcDone | GPUCopyDone | MFEProcUnlocked | MFEBanksMade | losslessComp |
|---|---|---|---|---|---|---|---|---|---|
| TCPProcUnlocked | | | | | | | | | |
| gotTCPHeaderWord | 52259 | | | | | | | | |
| gotTCPHeaderWord2 | 52269 | 10 | | | | | | | |
| GPUProcUnlocked | 52413 | 154 | 144 | | | | | | |
| GPUProcDone | N/A | N/A | N/A | N/A | | | | | |
| GPUCopyDone | N/A | N/A | N/A | N/A | N/A | | | | |
| MFEProcUnlocked | 52417 | 158 | 148 | 4 | N/A | N/A | | | |
| MFEBanksMade | 52421 | 162 | 152 | 8 | N/A | N/A | 4 | | |
| losslessComp | 52421 | 162 | 152 | 8 | N/A | N/A | 4 | 0 | |

Timing Difference Running plots (in microseconds)

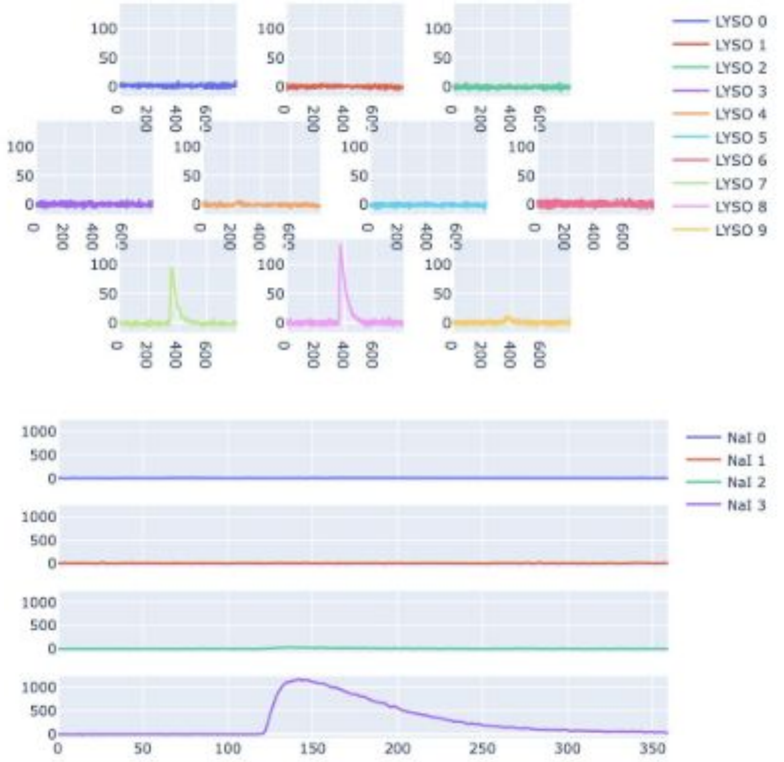
Example System Performance Webpage that Links with Midas

## Nearline Processing

- Any preliminary processing on the data before moving to permanent storage

**Examples:**

- Data quality monitors (DQM) that effectively sample and display data
- Building ROOT trees from midas files (Unpacker, by Sean Foster)
- Moving/Mirroring files



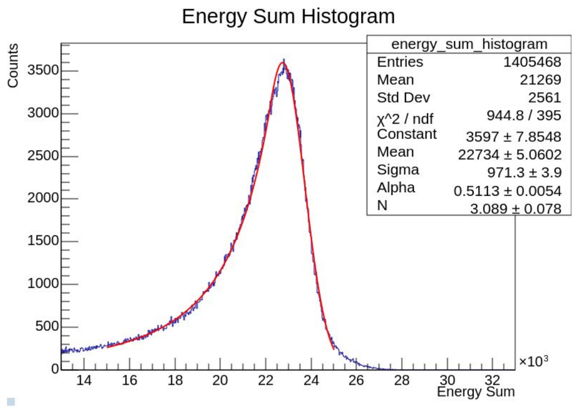
**Josh LaBounty's 2023 testbeam DQM page**

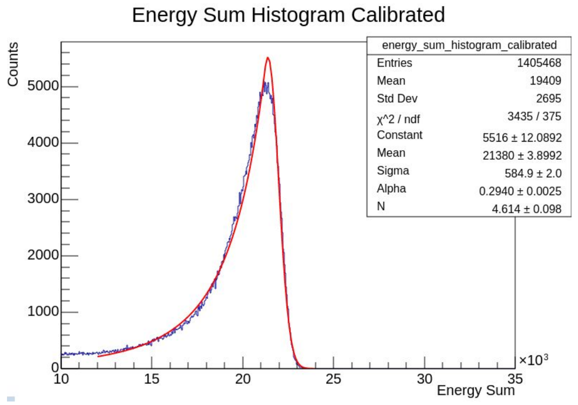
## Offline Processing

- Any processing on the data after it has been moved to permanent storage

**Examples:**

- Creating deposited energy histograms
- Chaining runs together
- Pretty much any rigorous analysis





**Preliminary Energy Sum Histograms from the 2023 Testbeam**

## Auxiliary Slides

## Outline

I. Introduction and Motivation
   A. What is DAQ?
   B. Proposed PIONEER DAQ Framework
   C. Why do all this? - Data Rates
   D. Two DAQs - Why?

II. The Hardware Side
   A. What is an FPGA?
   B. g-2 DAQ Hardware
   C. PIONEER DAQ proposed hardware

III. The Software Side
   A. Midas
   B. Frontends
   C. “Nearline” Processing
   D. “Offline” Processing

# Hardware Initialism Cheatsheet

| Initialism | Meaning | Example (if applicable) |
|---|---|---|
| DAQ | **D**ata **A**c**q**uisition | |
| ADC | **A**nalog-to-**D**igital **C**onverter | |
| 10GbE | **10 G**iga**b**it **E**thernet | |
| AFE | **A**nalog **F**ront **E**nd | |
| FPGA | **F**ield **P**rogrammable **G**ate **A**rray | |
| CPU | **C**entral **P**rocessing **U**nit | Intel Core i7-12700K |
| GPU | **G**raphics **P**rocessing **U**nit | NVIDIA A5000 |
| uTCA (or µTCA) | **M**icro **T**elecommunications **C**omputing **A**rchitecture | |
| WFD | **W**ave**f**orm **D**igitizer | WFD5 |
| FC | **F**lexible **C**ontroller | FC7 |
| AMC | **A**dvanced **M**ezzanine **C**ard | AMC13 (confusingly, also FC7 and WFD5) |
| MCH | **M**icroTCA **C**arrier **H**ub | |
| DDR | **D**ouble **D**ata **R**ate | DDR3, DDR4 (RAM) |
| PCIe | **P**eripheral **C**omponent **I**nterconnect **E**xpress | PCIe2, PCIe3, … |

# FPGA Types

Rough example name breakdown:

**XCVU190+1**:

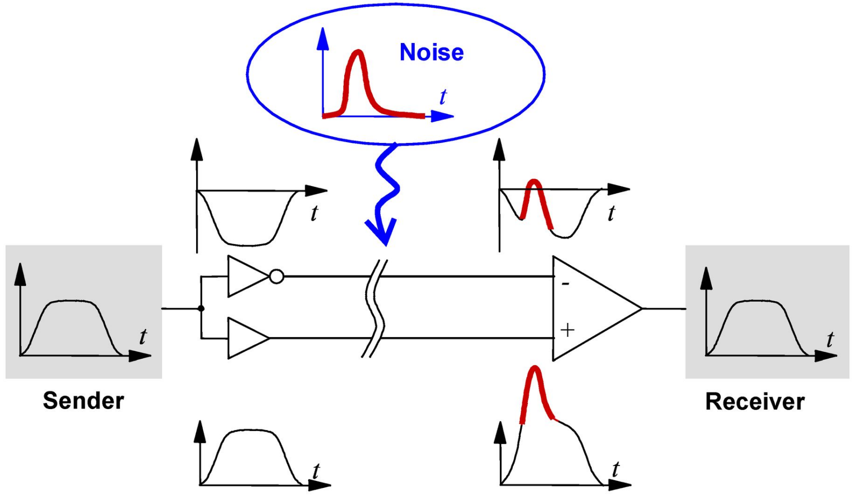
- **X**: Xilinx
- **C:** Some family indicator (?)
- **VU**: FPGA Family. "VU" → Virtex UltraScale family.
- **9**: Device capacity or size
- **+1,+2,+3:** A speed grade for the FPGA

| Series | Example FPGA |
|---|---|
| Virtex UltraScale+ | XCVU9P |
| Virtex UltraScale | XCVU190 |
| Kintex UltraScale+ | XCKU15P |
| Kintex UltraScale | XCKU040 |
| Artix UltraScale+ | XA7A50T |
| Artix-7 | XC7A200T |
| Zynq UltraScale+ MPSoC | XCZU9EG |
| Zynq-7000 SoC | XC7Z045 |
| Spartan-7 | XC7S100 |
| Spartan-6 | XC6SLX75 |

# Why a Differential Signal?

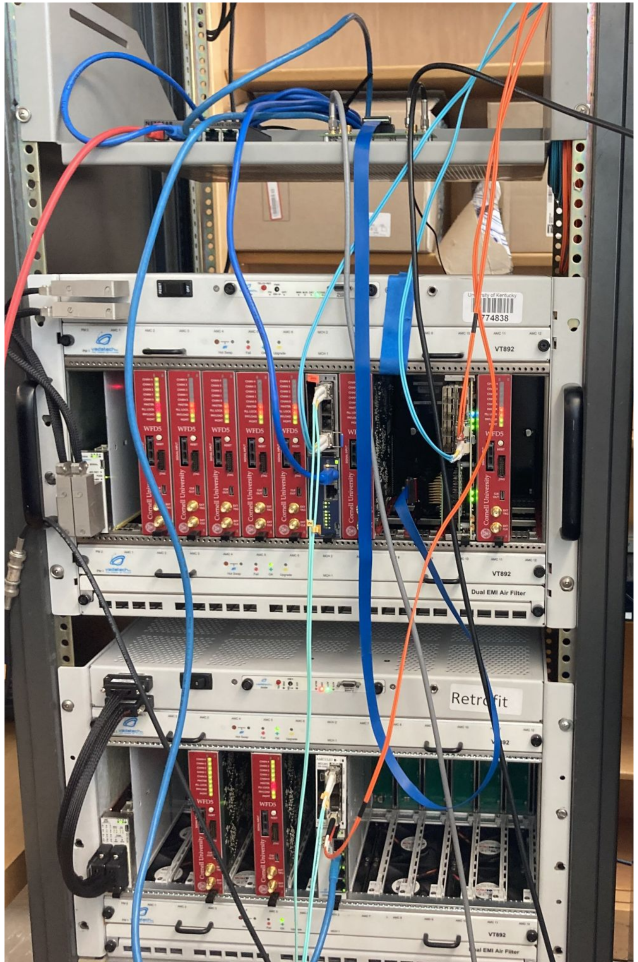
- More resistant to noise → cleaner signal

- Lower supply voltages can be used
    - reduce power consumption, and allow for higher operating frequencies.
    - Low Voltage CMOS (LVCMOS) is 3.0–3.3 V



## Multiple Crate g-2 DAQ Hardware

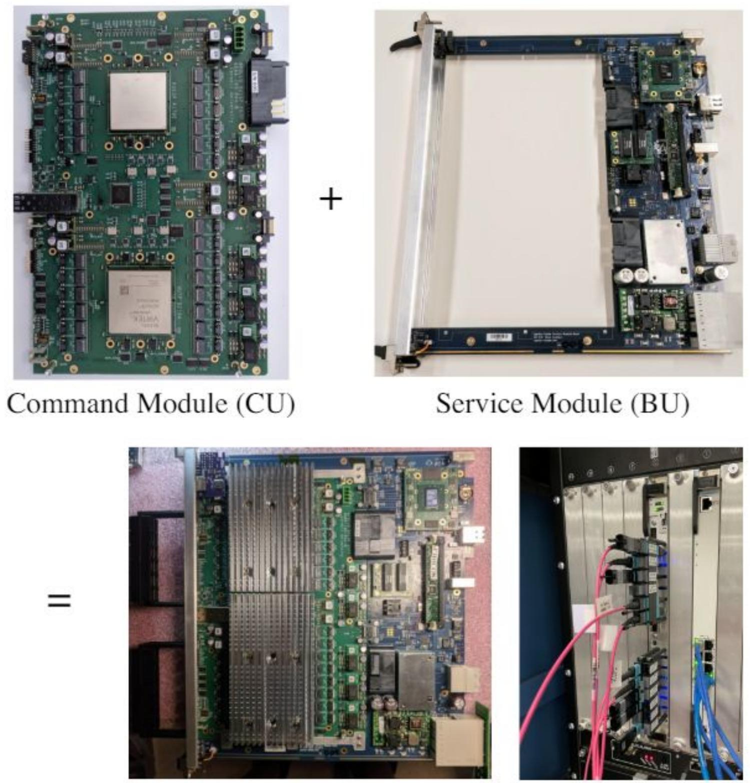
- Each crate needs an MCH to communicate with desktop
  - Another 1GbE link required, ethernet splitter introduced (see blue 1GbE cables)
- Each crate needs an AMC13
  - Another 10GbE data link to desktop introduced (see bottom mint cable)
  - Trigger signal fed from FC7 in first crate to AMC13 in bottom crate via optical cable (see orange cable)
- Note: There are two mint optical cables running towards a desktop rather than 1 mint cable connecting both AMC13s



Citation: DAQ backbone exploration, Lawrence Gibbons
https://pioneer.npl.washington.edu/docdb/0000/000023/001/apollo.pdf

# Why the Apollo System?

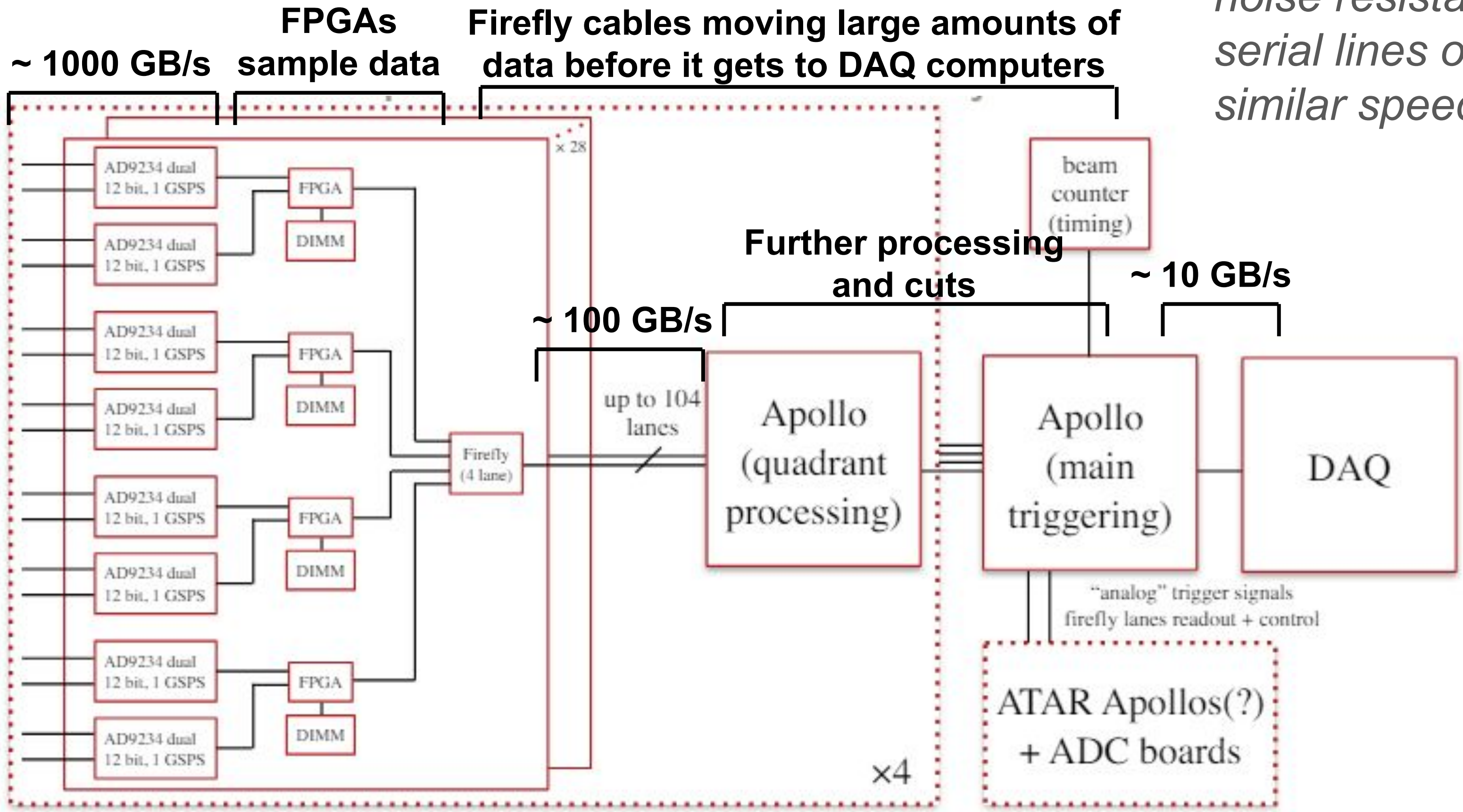
- CERN + CMS/ATLAS → APOLLO platform
  - Cornell already had a hand in designing boards for APOLLO system
- Unlike µTCA, the actual data handling does not need to move through the backplane
  - More user control
- APOLLO system handles more channels per optical link → fewer desktops needed
  - APOLLO System ~ 3000 channels/(400 chan/board * 2 boards/computer) ~ **4 computers**
  - µTCA System ~ 3000 channels/(60 chan/crate * 2 crates/computer) ~ **30 computers**



Citation: DAQ backbone exploration, Lawrence Gibbons
https://pioneer.npl.washington.edu/docdb/0000/000023/001/apollo.pdf

## Why Firefly Cables?

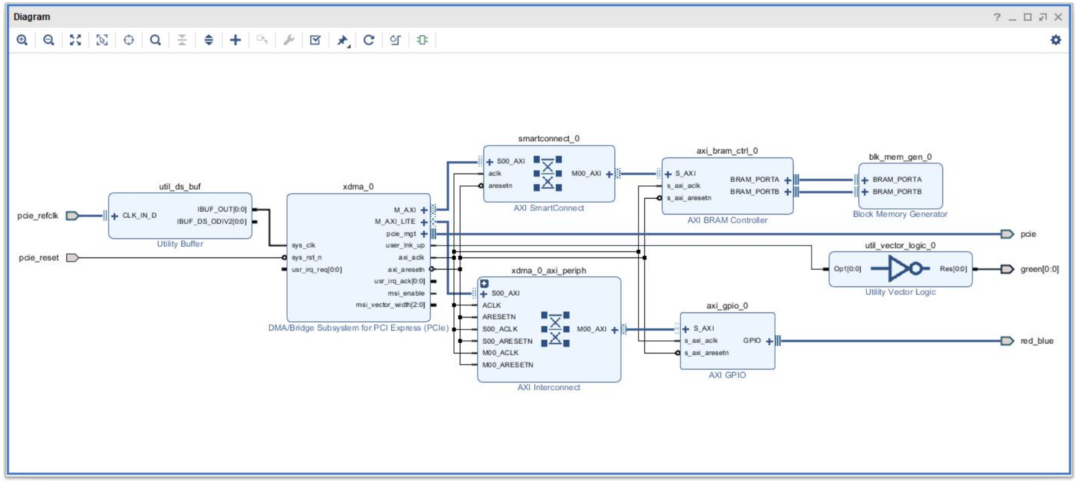
*Optical → More noise resistant than serial lines of similar speeds*


~ 1000 GB/s
FPGAs sample data
Firefly cables moving large amounts of data before it gets to DAQ computers
AD9234 dual 12 bit, 1 GSPS
FPGA
DIMM
Firefly (4 lane)
× 28
~ 100 GB/s
up to 104 lanes
Further processing and cuts
Apollo (quadrant processing)
beam counter (timing)
~ 10 GB/s
Apollo (main triggering)
DAQ
"analog" trigger signals
firefly lanes readout + control
ATAR Apollos(?) + ADC boards
×4

# Communication with FPGA over PCIe

- Want a midas frontend that communicates with an FPGA over PCIe

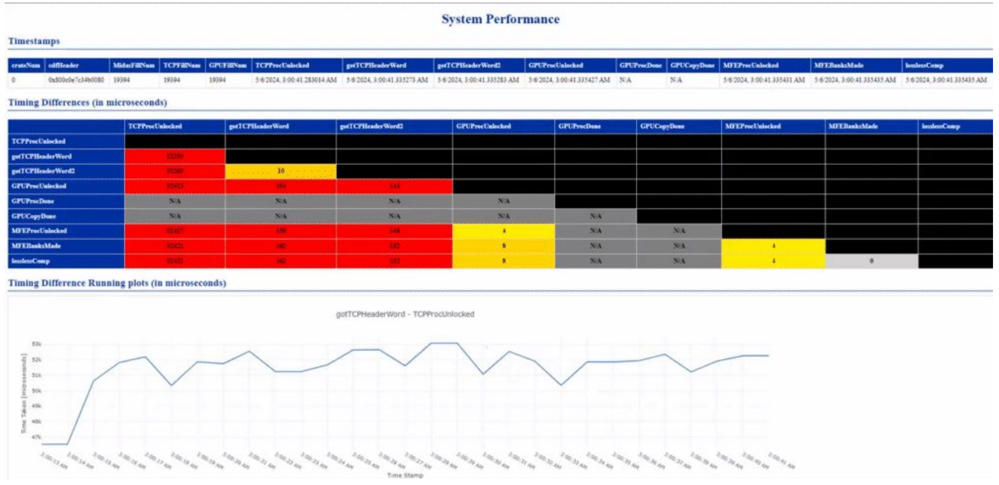
- This should streamline implementation when Cornell finalizes hardware



**Example block diagram (made in Vivado) for a PCIe FPGA**

## Adding More Debugging Diagnostics to g-2 modified DAQ

- Created a more general DQM page (no assumption on number of channels/channel mapping)
- Rate limitations were an issue during 2023 test beam
  - Could only run at ~300Hz
- Added timing diagnostics to identify bottleneck
- Plan to add CPU, RAM, and FC7 diagnostic pages as well



**System Performance**

**Timestamps**

| crateNum | cdfHeader | MidasFillNum | TCPFillNum | GPUFillNum | TCPProcUnlocked | gotTCPHeaderWord | gotTCPHeaderWord2 | GPUProcUnlocked | GPUProcDone | GPUCopyDone | MFEProcUnlocked | MFEBanksMade | losslessComp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0x800c0e7c34b0080 | 19394 | 19394 | 19394 | 5/6/2024, 3:00:41.283014 AM | 5/6/2024, 3:00:41.335273 AM | 5/6/2024, 3:00:41.335283 AM | 5/6/2024, 3:00:41.335427 AM | N/A | N/A | 5/6/2024, 3:00:41.335431 AM | 5/6/2024, 3:00:41.335435 AM | 5/6/2024, 3:00:41.335435 AM |

**Timing Differences (in microseconds)**

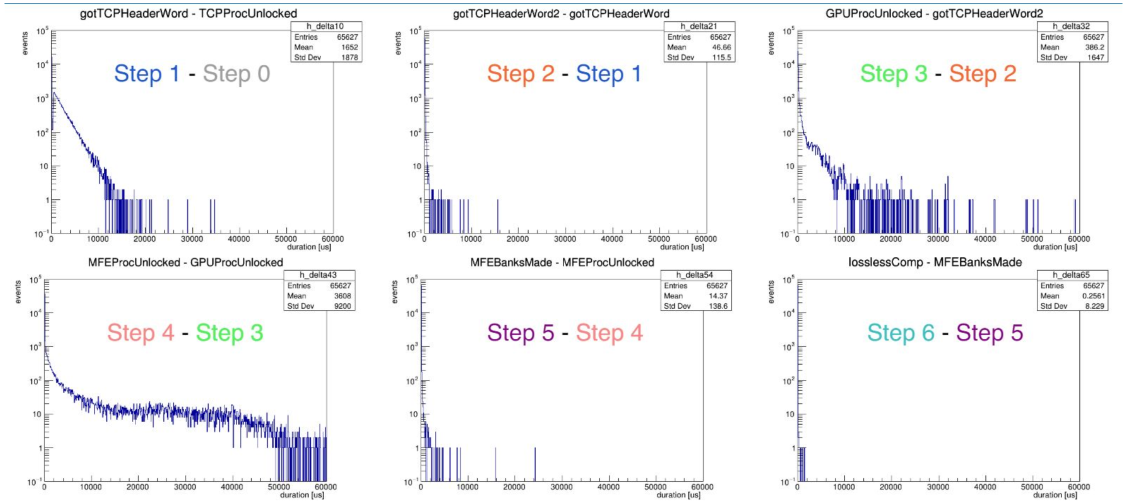
| | TCPProcUnlocked | gotTCPHeaderWord | gotTCPHeaderWord2 | GPUProcUnlocked | GPUProcDone | GPUCopyDone | MFEProcUnlocked | MFEBanksMade | losslessComp |
|---|---|---|---|---|---|---|---|---|---|
| TCPProcUnlocked | | | | | | | | | |
| gotTCPHeaderWord | 52259 | | | | | | | | |
| gotTCPHeaderWord2 | 52269 | 10 | | | | | | | |
| GPUProcUnlocked | 52413 | 154 | 144 | | | | | | |
| GPUProcDone | N/A | N/A | N/A | N/A | | | | | |
| GPUCopyDone | N/A | N/A | N/A | N/A | N/A | | | | |
| MFEProcUnlocked | 52417 | 158 | 148 | 4 | N/A | N/A | | | |
| MFEBanksMade | 52421 | 162 | 152 | 8 | N/A | N/A | 4 | | |
| losslessComp | 52421 | 162 | 152 | 8 | N/A | N/A | 4 | 0 | |

**Timing Difference Running plots (in microseconds)**

**Example System Performance Webpage that Links with Midas**

# Rate Testing/Improving g-2 modified DAQ

- Analyzed test beam and UKY teststand performance data
  - Bottlenecks are due to rare, long pauses between events
  - Yet to determine exact reason for pauses
- Plan to remove Meinberg card from system, replace with parallel port system
  - Should be faster and more straightforward

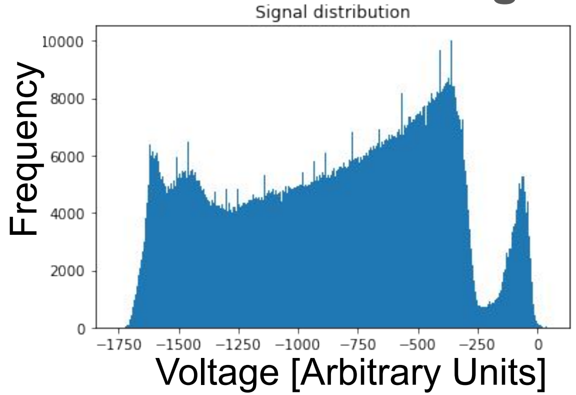


**Timings of various stages of the data readout midas frontend**

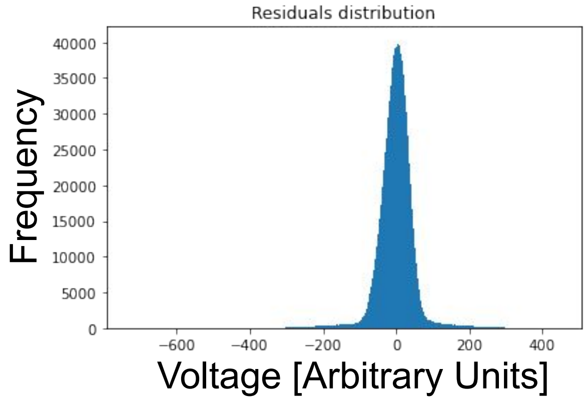
## Signal Conditioning

- Want a narrow distribution for compression. Let $r_i$ be the numbers we compress
- Methods tried:
  - No conditioning
  - Delta encoding:
    $r_i = y_{i+1} - y_i$
  - Twice Delta Encoding:
    $r_i = y_{i+2} - 2y_{i+1} + y_i$
  - Double Exponential Fit:
    $r_i = y_i - (A \cdot \exp(at_i) + B \cdot \exp(bt_i))$
  - **Shape Fit**:
    $\mathbf{r_i = y_i - (A \cdot T(t_i - t_0) + B)}$

**No Conditioning**



**Shape Fit**



## Shape Fitting Algorithm

1. Construct a discrete template from sample pulses
2. Interpolate template to form a continuous Template, *T(t)*
3. “Stretch” and “shift” template to match signal:

$$X[i] = a(t_0)T(t[i] - t_0) + b(t_0)$$

[Note: a and b can be calculated explicitly given $t_0$]

4. Compute $\chi^2$ (assuming equal uncertainty on each channel i)

$$\chi^2 \propto \sum_i \{X[i] - a(t_0)T(t[i] - t_0) + b(t_0)\}^2$$

5. Use Euler’s method to minimize $\chi^2$

## Lossless Compression Algorithm

- Rice-Golomb Encoding
  - Let x be number to encode
    y = “s”+“q”+”r”
    - q = x/M (unary)
    - r = x%M (binary)
    - s = sign(x)
  - Any distribution
  - Close to optimal for valid choice of M
  - One extra bit to encode negative sign
  - Self-delimiting
  - If quotient too large, we “give up” and write x in binary with a “give up” signal in front

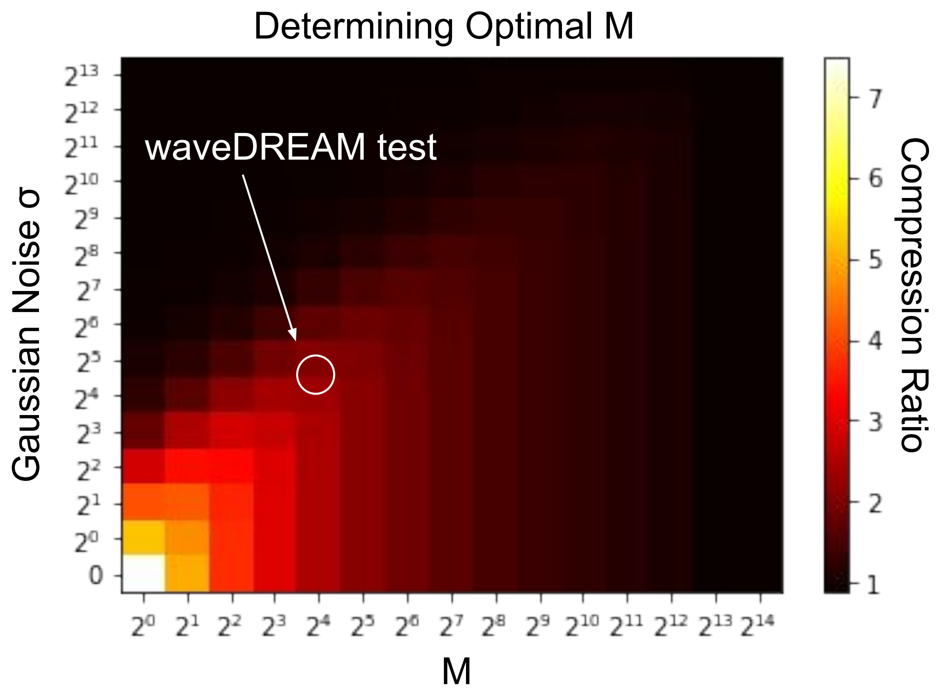
**Rice-Golomb Encoding (M=2)**

| Value | Encoding |
|---|---|
| -1 | 011 |
| 0 | 000 |
| 1 | 001 |
| 2 | 1000 |

Red = sign bit
Blue = quotient bit(s) (Unary)
Yellow = remainder bit (binary)

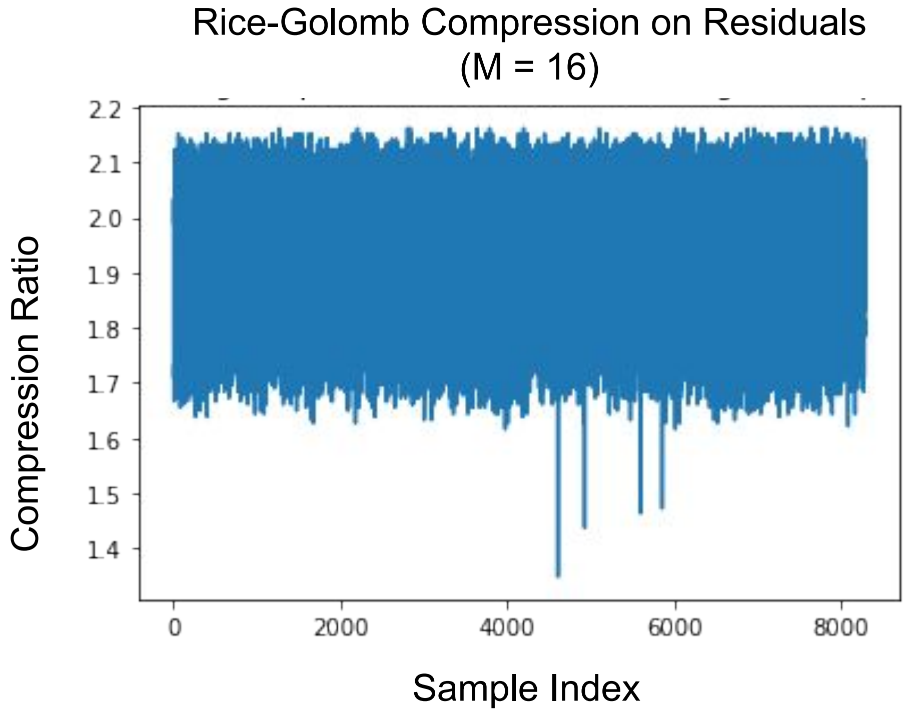
# How to choose Rice-Golomb parameter M

- Generated fake Gaussian data (centered at zero) with variance $\sigma^2$
- For random variable X, M ≈ median(|X|)/2 is a good choice
  - This is the close to the diagonal on the plot
- $\sigma \approx 32$ for residuals of shape on wavedream data → M = 16 is a good choice



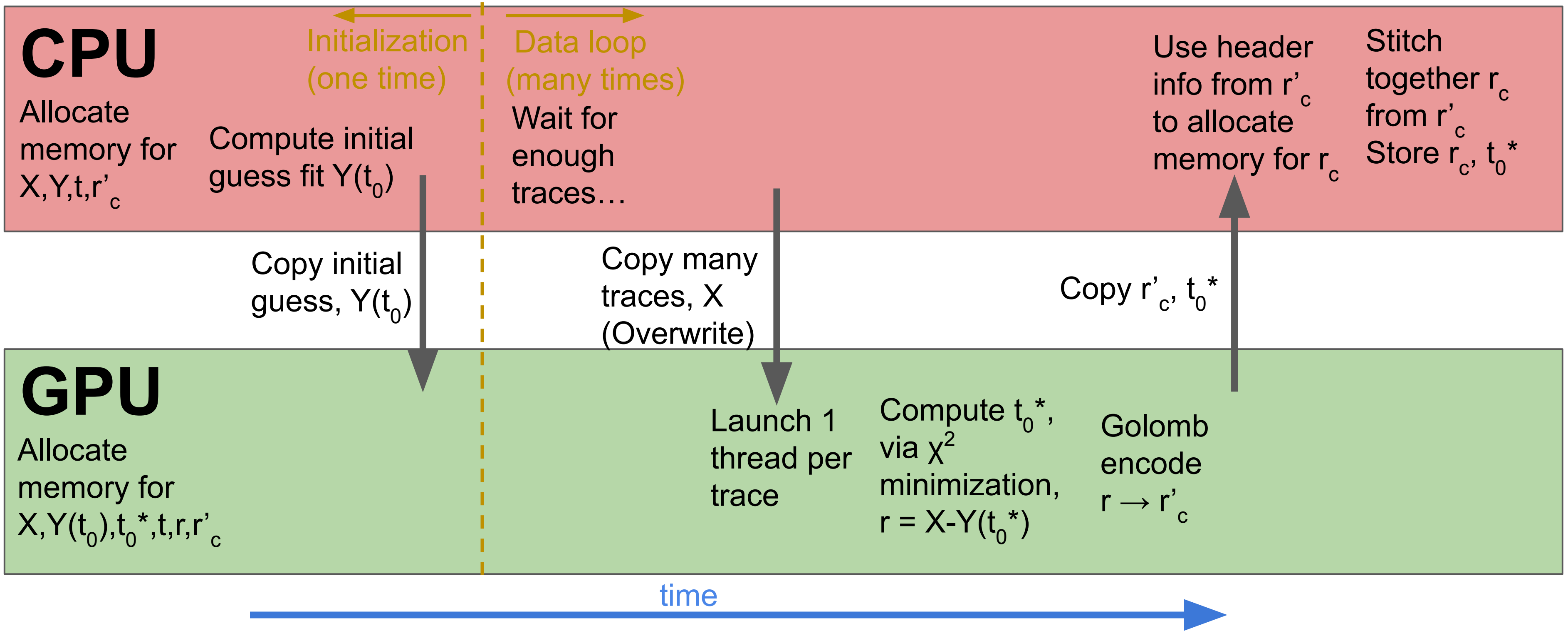
## Compression Ratio from Rice-Golomb Encoding

- Lossless compression factor of ~2
- In agreement with plot from simulated data on last slide
- Best compression ratio we achieved



## Real Time Compression Algorithm

- We choose to let the FE's GPU and CPU handle compression for flexibility

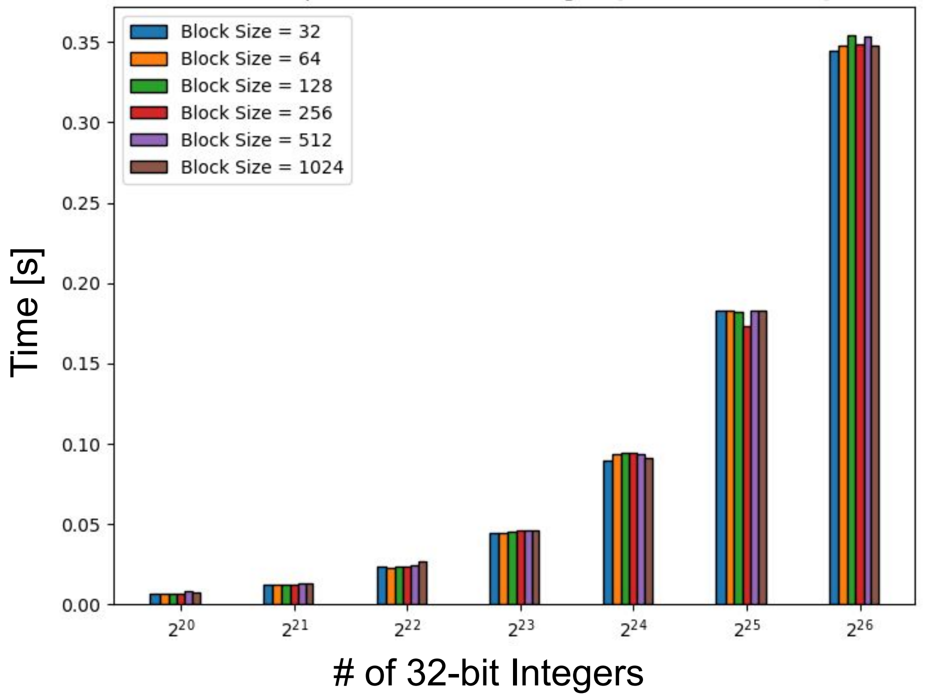

CPU
Initialization (one time)
Data loop (many times)
Allocate memory for X,Y,t,$r'_c$
Compute initial guess fit $Y(t_0)$
Wait for enough traces…
Use header info from $r'_c$ to allocate memory for $r_c$
Stitch together $r_c$ from $r'_c$ Store $r_c$, $t_0^*$
Copy initial guess, $Y(t_0)$
Copy many traces, X (Overwrite)
Copy $r'_c$, $t_0^*$
GPU
Allocate memory for X,$Y(t_0)$,$t_0^*$,t,r,$r'_c$
Launch 1 thread per trace
Compute $t_0^*$, via $\chi^2$ minimization, $r = X-Y(t_0^*)$
Golomb encode $r \rightarrow r'_c$
time

## GPU Benchmarking (Timings)

- Block Size:
  - A GPU parameter, number of threads per multiprocessor

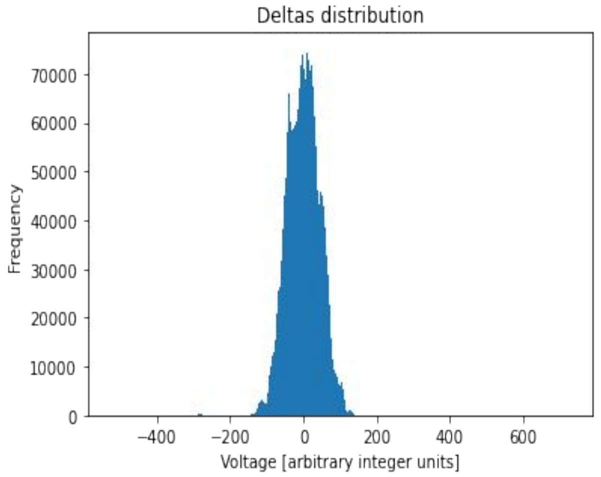
- Can compress $2^{26}$ integers (32-bit) in roughly ⅓ of a second. → ~ **0.8 GB/s** compression rate

Fit + Compression Time using A5000 in PCIe4
(Batch Size = 1024)

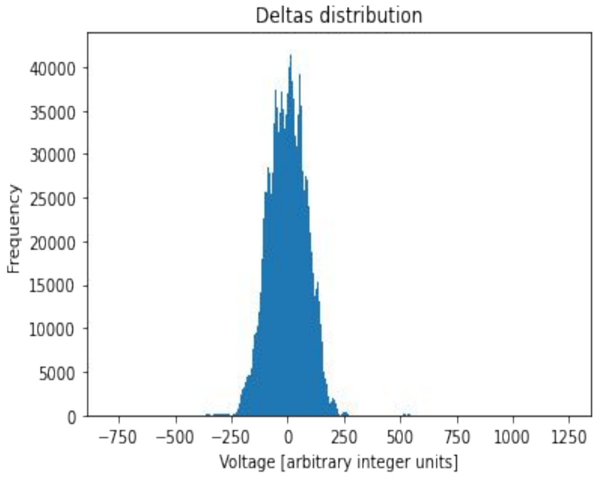


## Other Conditioning Distributions

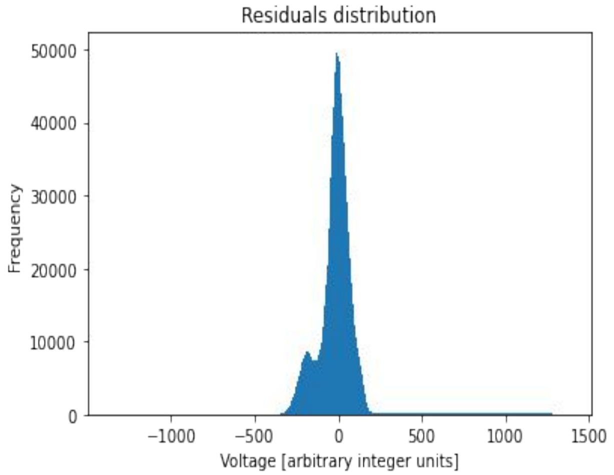
### Delta Encoding



### Twice Delta Encoding



### Double Exponential Fit



## Shape Fitting Details

Fit Function

$$X[i] = aT(t[i] - t_0) + b$$

Explicit a($t_0$) calc

$$a(t_0) = \frac{\sum_i^N X[i] \sum_i^N T(t[i] - t_0)^2 - \sum_i^N T(t[i] - t_0) \sum_i^N T(t[i] - t_0) X[i]}{N \sum_i^N T(t[i] - t_0)^2 - (\sum_i^N T(t[i] - t_0))^2}$$

Explicit b($t_0$) calc

$$b(t_0) = \frac{N \sum_i^N T(t[i] - t_0) X[i] - \sum_i^N T(t[i] - t_0) \sum_i^N X[i]}{N \sum_i^N T(t[i] - t_0)^2 - (\sum_i^N T(t[i] - t_0))^2}$$

Explicit $\chi^2$ calc

$$f(t_0) \equiv \chi^2 \propto \sum_i \{X[i] - a(t_0) T(t[i] - t_0) + b(t_0)\}^2$$

Newton's method

$$(t_0)_{n+1} = (t_0)_n - \frac{f'((t_0)_n)}{f''((t_0)_n)}$$

Threshold requirement

$$|(t_0)_{n+1} - (t_0)_n| < \epsilon \equiv \text{"Threshold"}$$

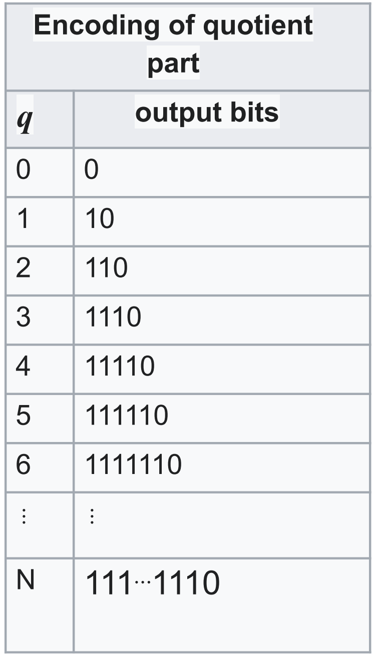
Citation: Wikipedia
(https://en.wikipedia.org/wiki/Golomb_coding)

## Golomb Encoding

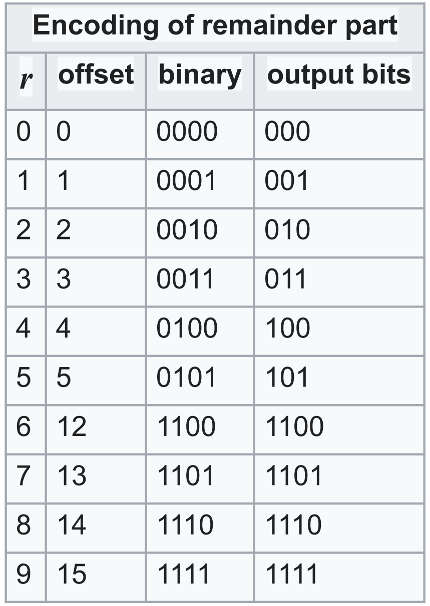
- In general, M is an arbitrary choice
- Since computers work with binary, $M = 2^x$ such that x is an integer is a “fast” choice
  - This is called Rice-Golomb Encoding
- Self delimiting so long as the information M is provided

### Golomb Encoding Example

Choose $M = 10$, $b = \log_2(M) = 3$
$2^{b+1} - M = 16 - 10 = 6$
$r < 6 \rightarrow$ r encoded in b=3 bits
$r \geq 6 \rightarrow$ r encoded in b+1=4 bits



| Encoding of quotient part | |
|---|---|
| $q$ | output bits |
| 0 | 0 |
| 1 | 10 |
| 2 | 110 |
| 3 | 1110 |
| 4 | 11110 |
| 5 | 111110 |
| 6 | 1111110 |
| ⋮ | ⋮ |
| N | 111⋯1110 |



| Encoding of remainder part | | | |
|---|---|---|---|
| $r$ | offset | binary | output bits |
| 0 | 0 | 0000 | 000 |
| 1 | 1 | 0001 | 001 |
| 2 | 2 | 0010 | 010 |
| 3 | 3 | 0011 | 011 |
| 4 | 4 | 0100 | 100 |
| 5 | 5 | 0101 | 101 |
| 6 | 12 | 1100 | 1100 |
| 7 | 13 | 1101 | 1101 |
| 8 | 14 | 1110 | 1110 |
| 9 | 15 | 1111 | 1111 |

Citation: Wikipedia
(https://en.wikipedia.org/wiki/Huffman_coding)

## Huffman Encoding

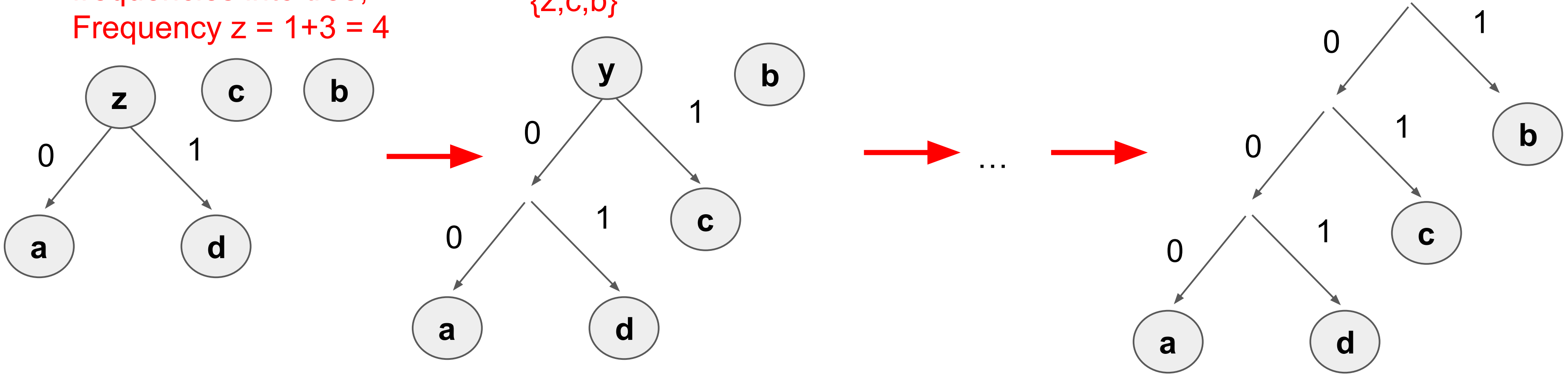
- Requires finite distribution
- Values treated as “symbols”
- Self-delimiting (sometimes called “greedy”)

### Huffman Encoding Example

| Value | Frequency | Encoding |
|---|---|---|
| -1 ≡ a | 1 | 000 |
| 0 ≡ b | 10 | 1 |
| 1 ≡ c | 5 | 01 |
| 2 ≡ d | 3 | 001 |

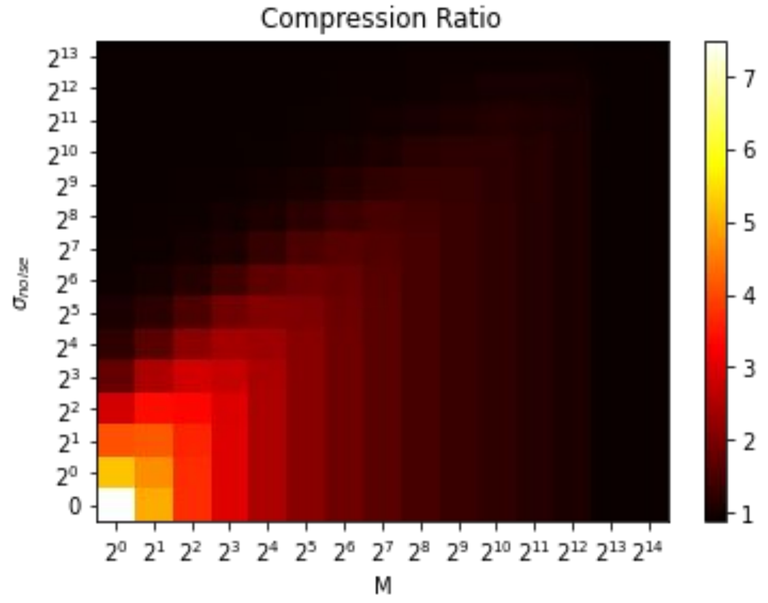
“Combine” two lowest frequencies into tree, Frequency z = 1+3 = 4

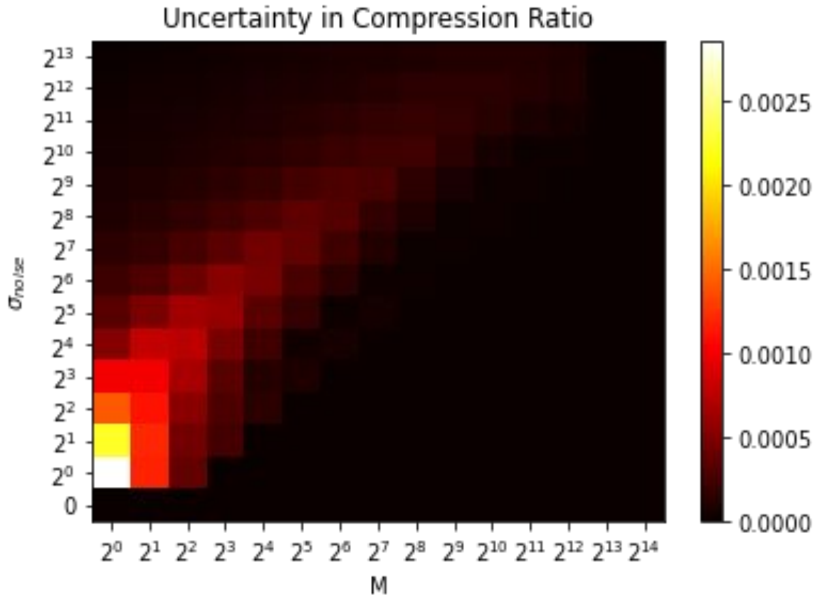
Repeat for set {z,c,b}



# Theoretical Uncertainty in Compression Ratio from Gaussian Noise

- ~ 0.1% relative error

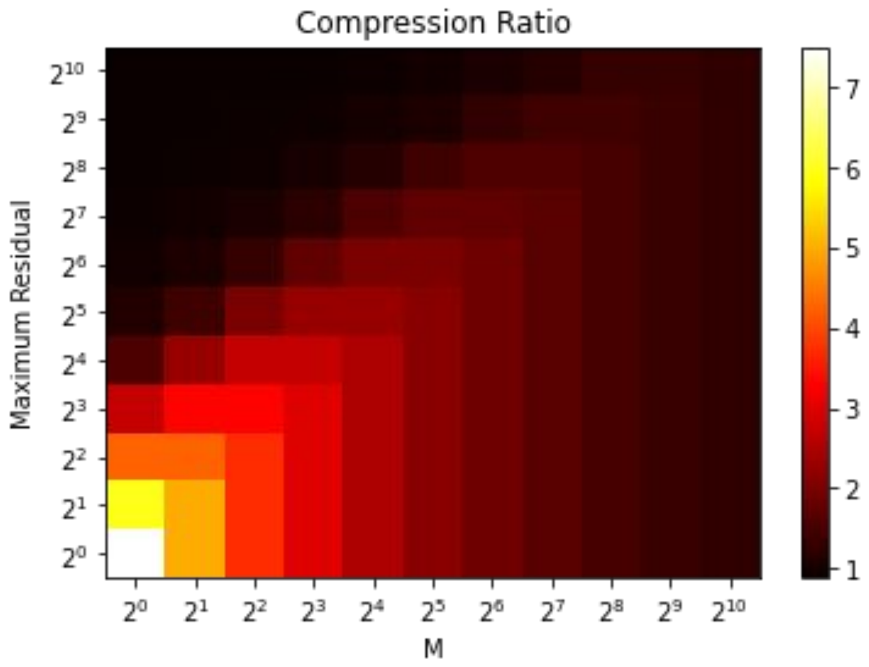




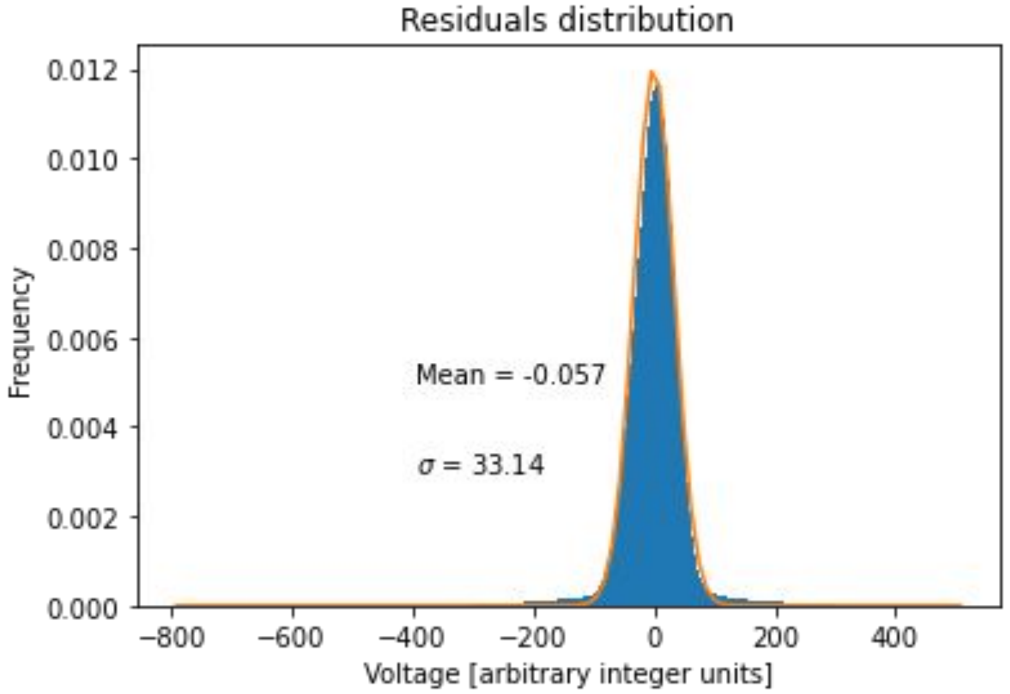
# Uniform Distribution of Noise effect on Compression Ratio

- Here instead we use a uniform distribution to generate the noise

- Not much different than gaussian noise, same conclusions really



## Residuals Distribution and Optimal M



| **M** | **Compression Ratio** |
|---|---|
| 1 | 1.04721105 |
| 2 | 1.21287474 |
| 4 | 1.53114598 |
| 8 | 1.92616642 |
| **16** | **2.09307249** |
| 32 | 2.02975311 |
| 64 | 1.86037914 |
| 128 | 1.66627451 |
| . . . | . . . |

# Lossy Compression Idea

- In lossless compression, Rice-Golomb encodes:
  1. Fit parameters
  2. Residuals

- If the residuals meet some criteria, we may choose to threw them out just keeping our fit of the signal.

  Example Criteria: $$\sum_i r[i] < \epsilon \equiv \text{"Threshold"}$$