Karlsruhe Institute of Technology

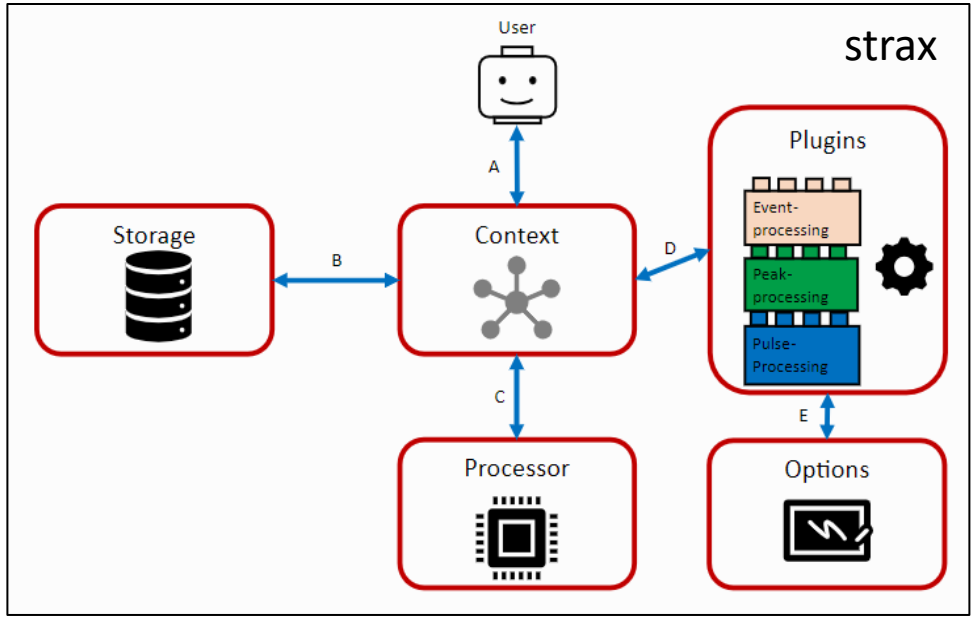
Helix:
DELight data processing framework

Alexander (Sasha) Zaytsev

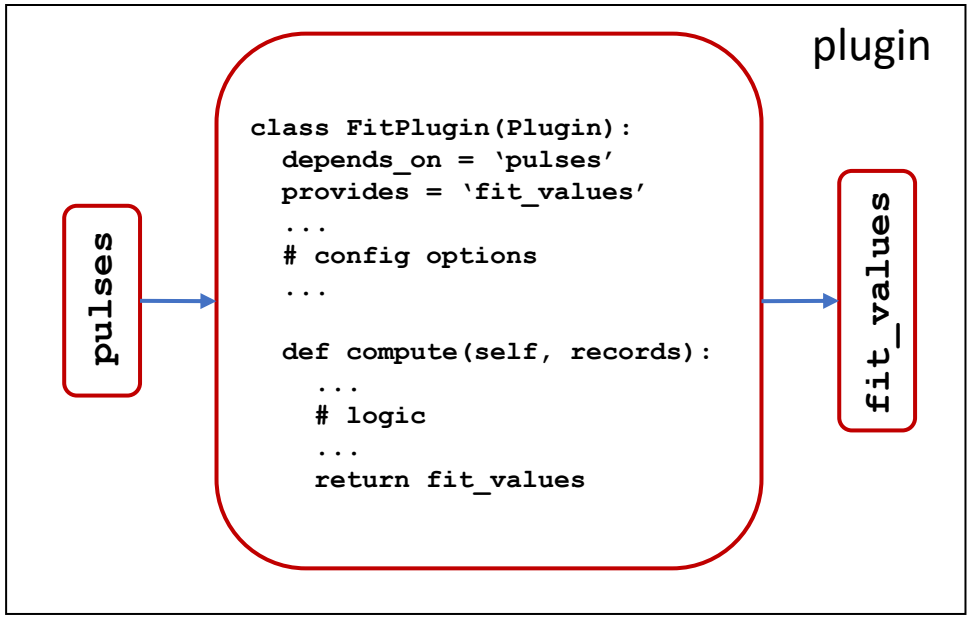
23.04.2024

Strax

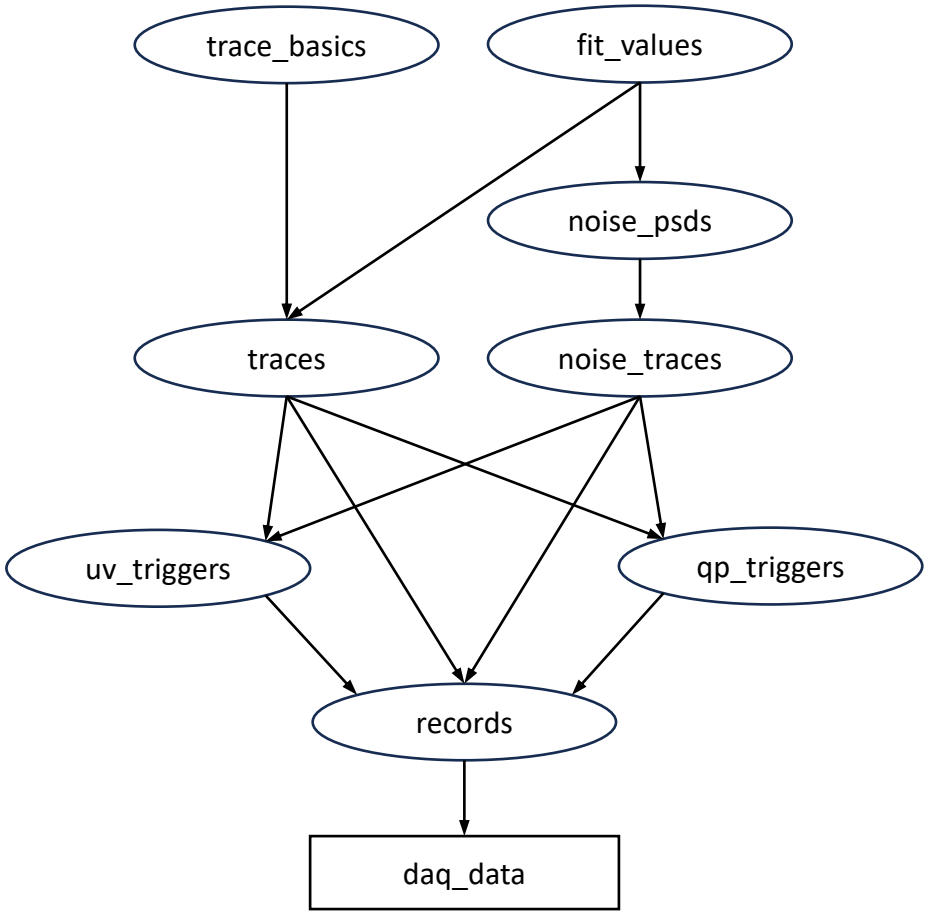
- Strax: data processing framework



- Processing algorithms are contained within plugins

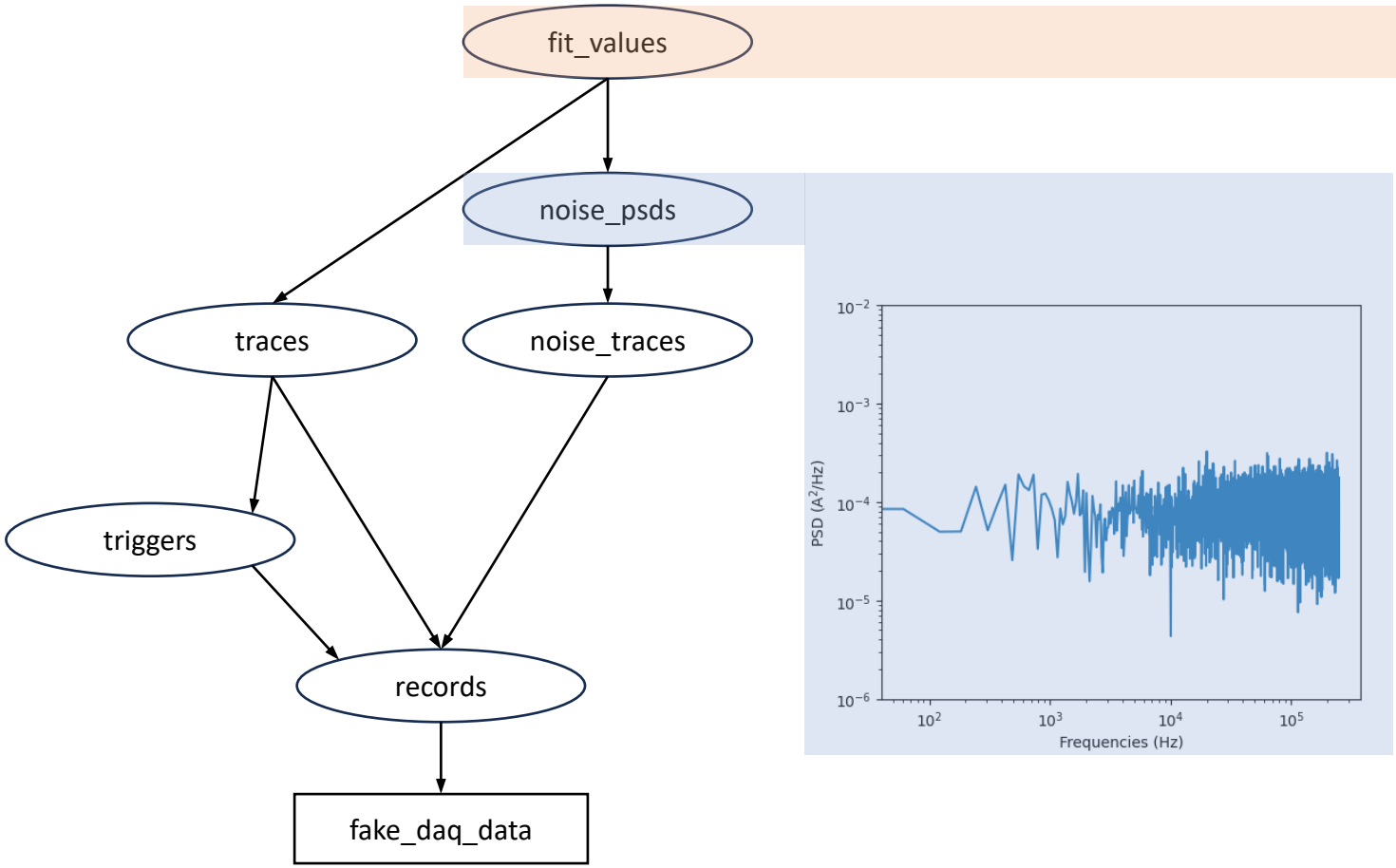


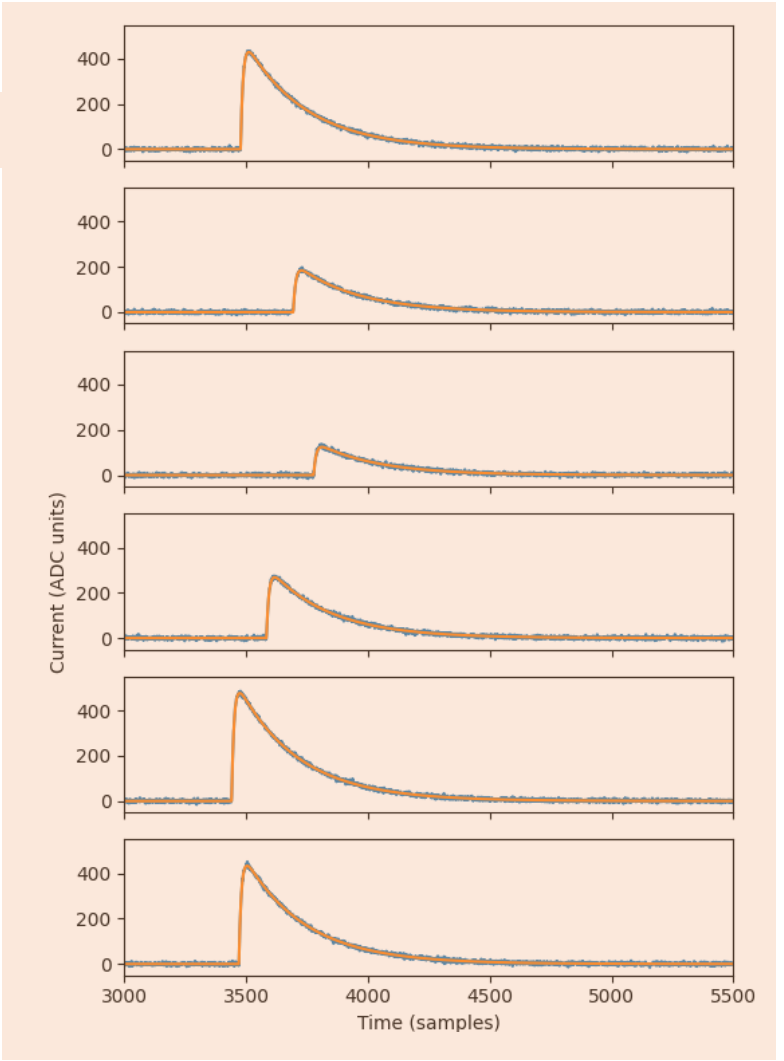
Helix data structure



- basics, fit_values: fit results, integrals, baselines, etc
- noise_psds: noise Power Spectrum Densities for optimum filter
- traces: pieces of records containing events
- triggers: trigger locations with some trigger info
- records: digitized waveforms in strax format
- daq_data: DAQ output data files

Helix skeleton

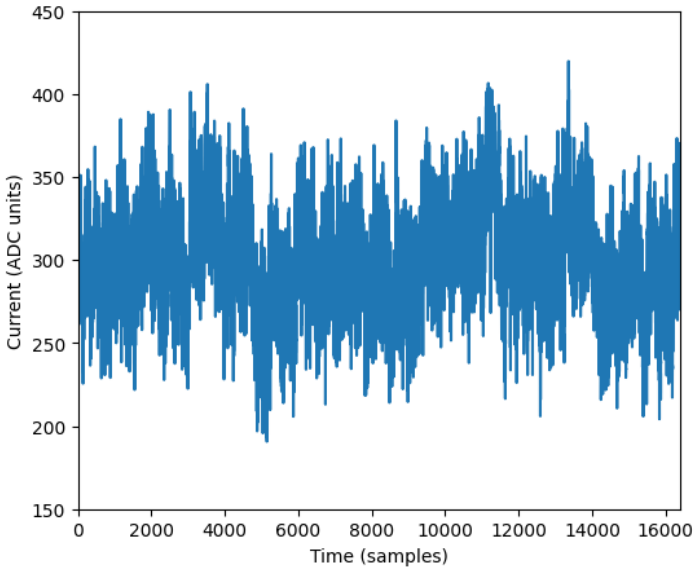




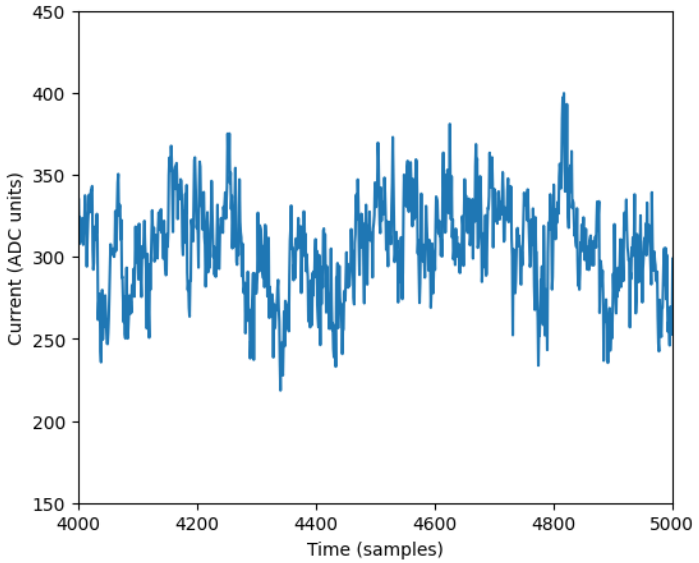
Helix toy data: noise

- 50 channels – 15 vacuum, 35 submerged
- Pink 1/f noise with one additional correlated 5 kHz line
- 250 kHz sampling frequency

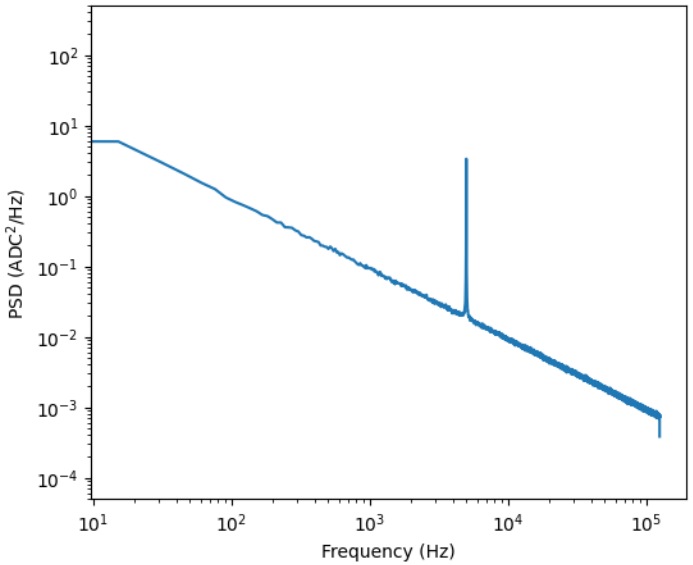
Noise trace



Noise trace (zoomed in)



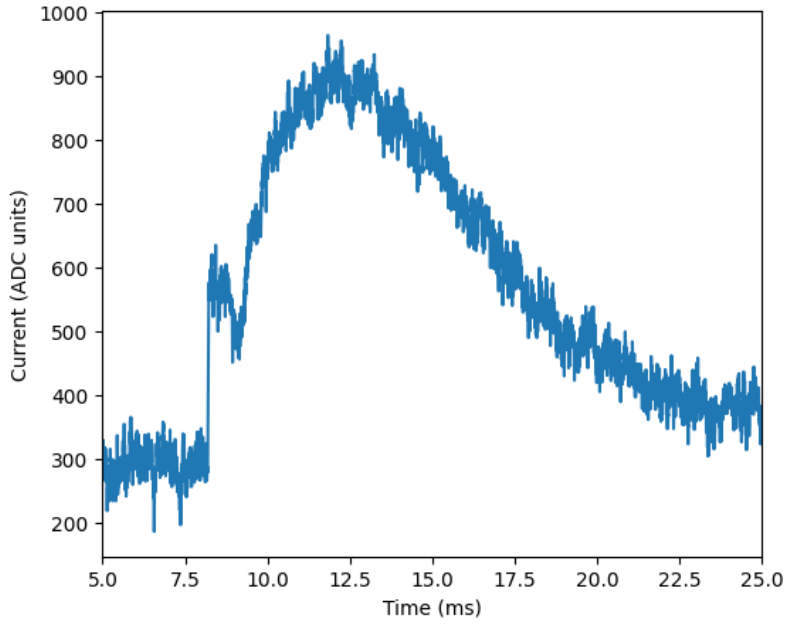
PSD

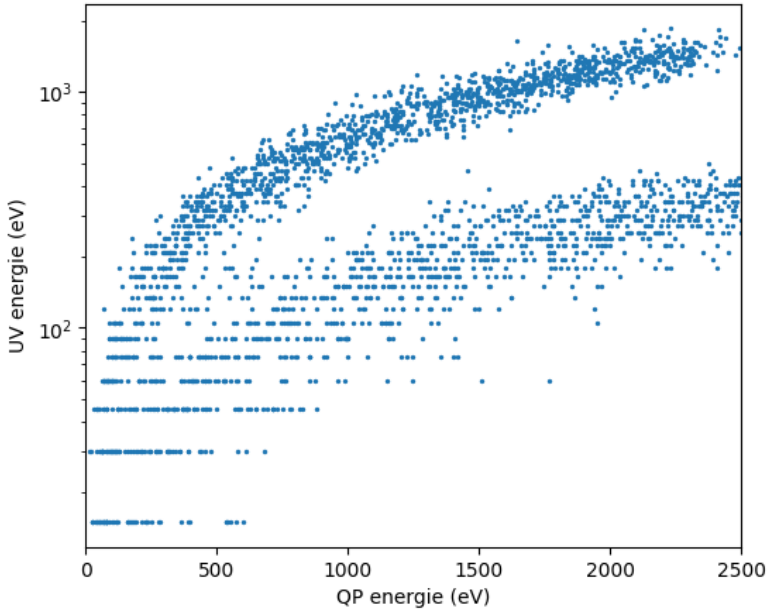


Helix toy data: signals

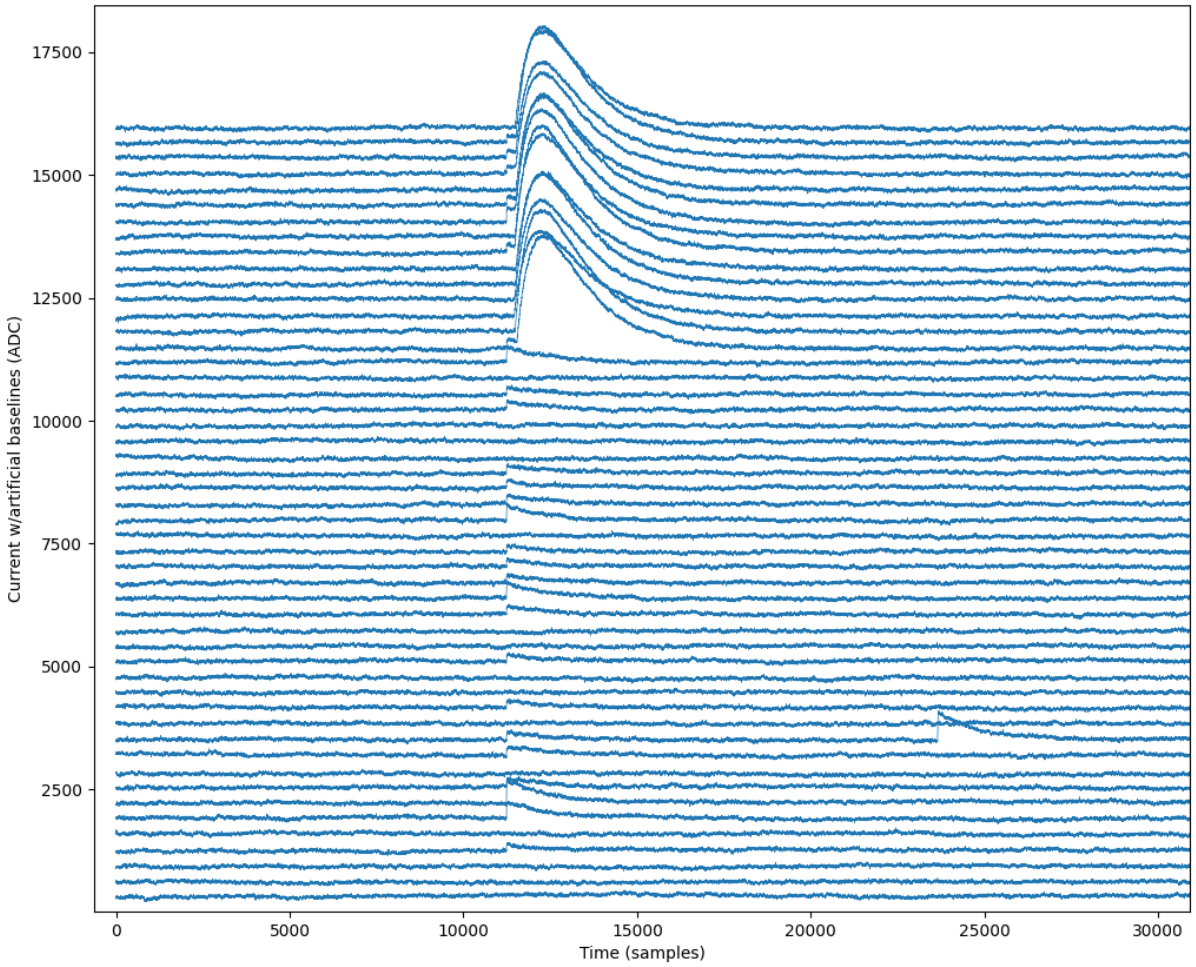
- Analytical 2-exponential templates with different rise times (20 µs for UV, 2 ms for QP)

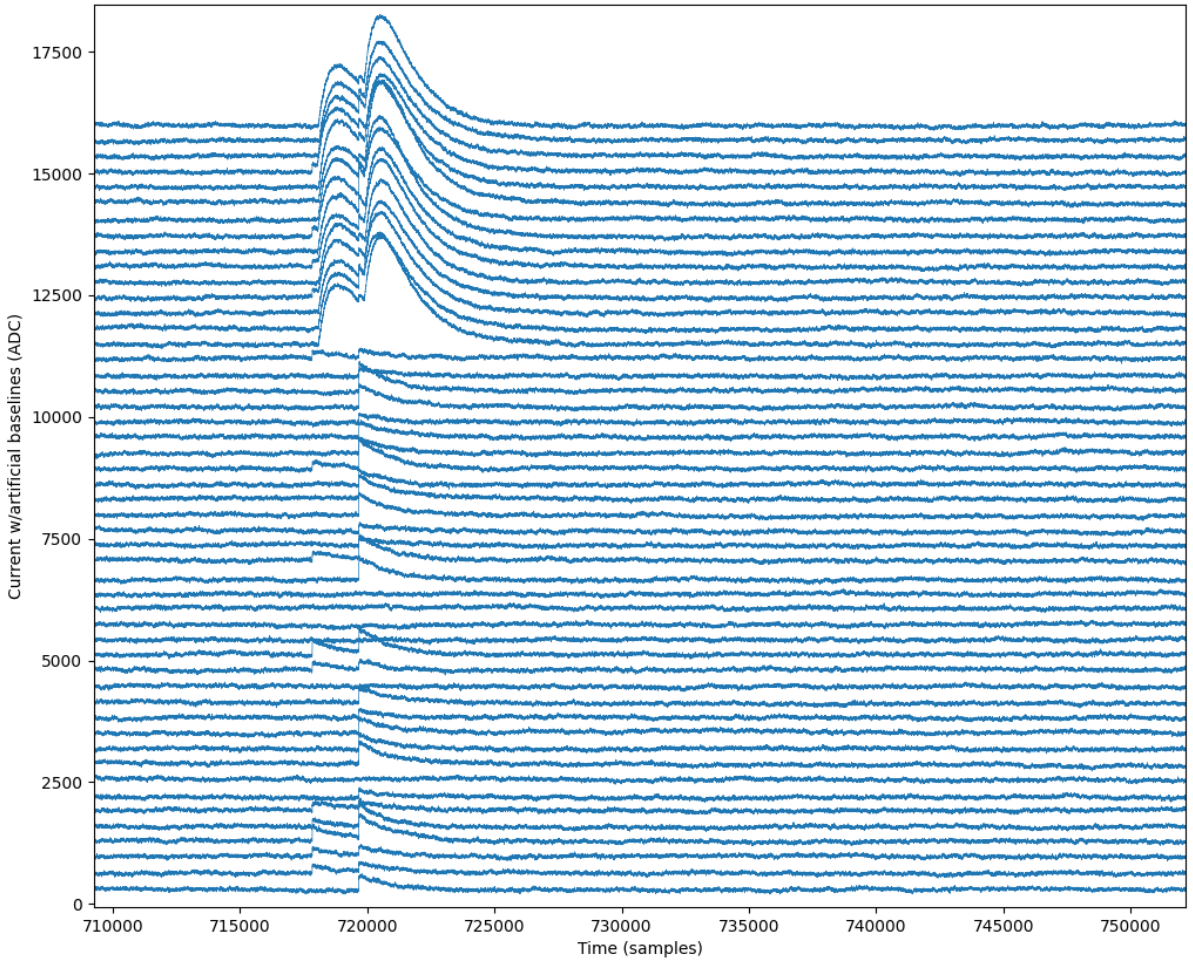
- Event energy split between QP and UV signals, UV signal is quantized, QP is randomly distributed among vacuum channels
- QP signal has a varying delay
- Backgrounds: lone hits with UV template, high-energy muons
- Saturation cutoff
- Events on record edges (split into two records)



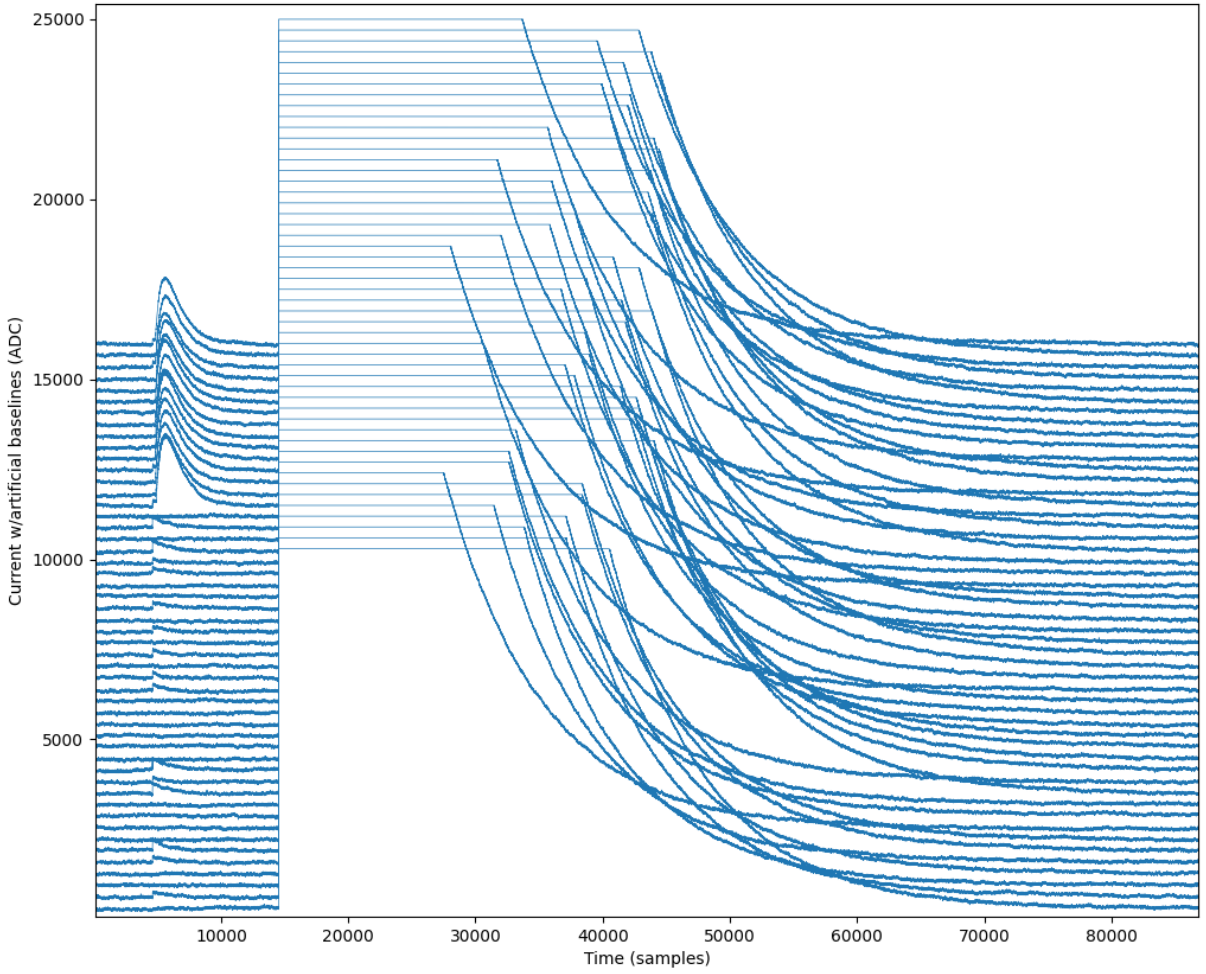


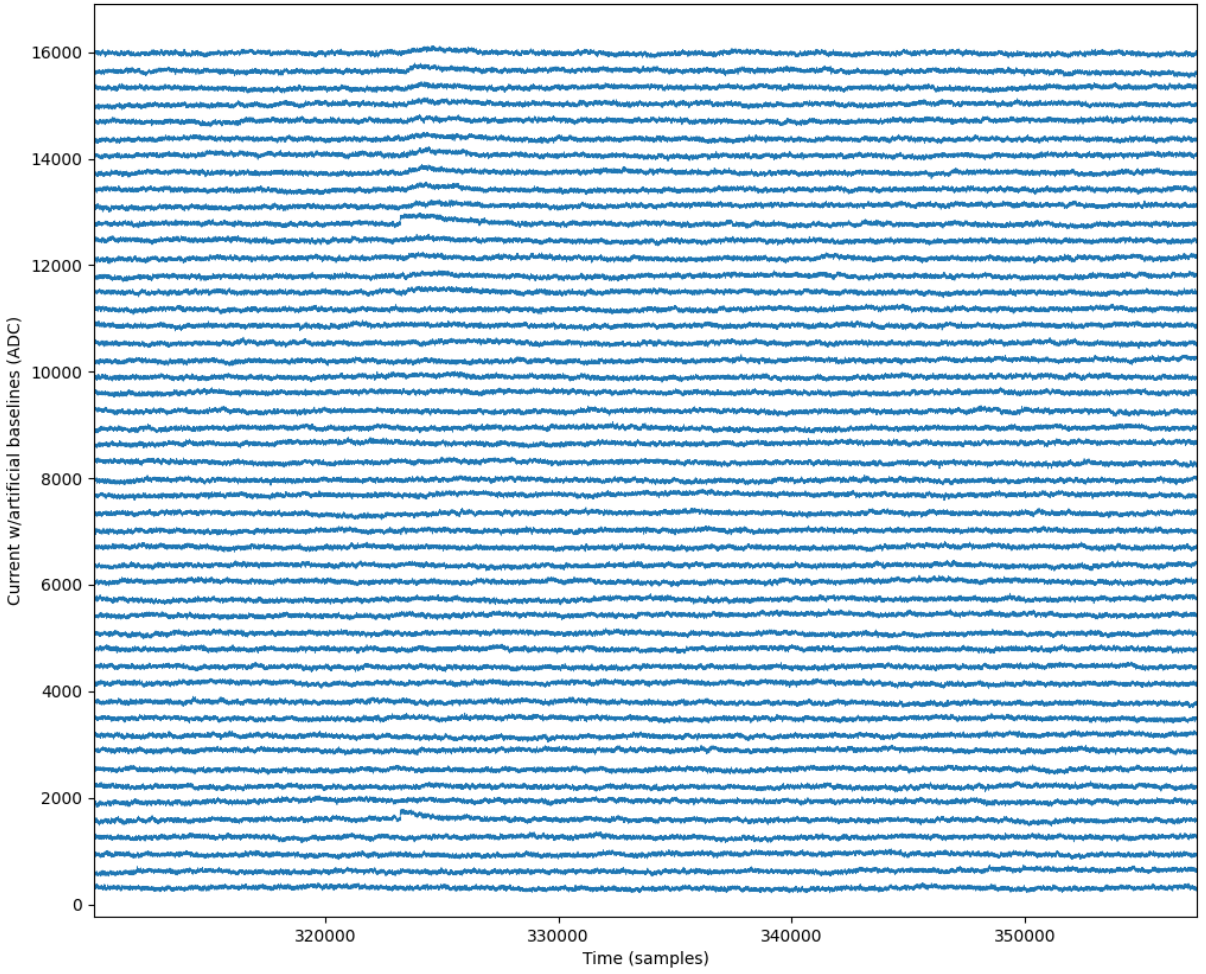
Helix toy data





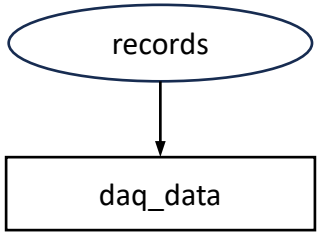
Helix toy data





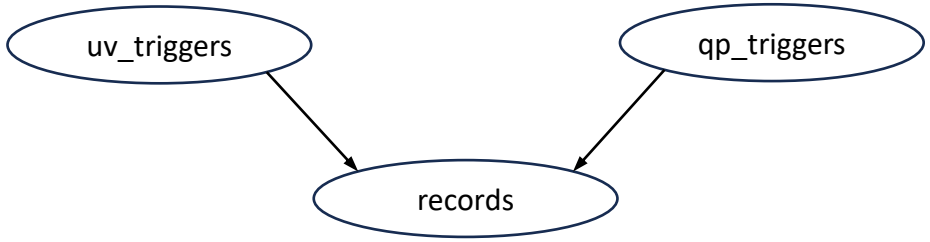
Helix toy data

- 50 channels, 250 kHz sampling, int16 arrays, lz4 compression = 20 MB/sec of raw data
- 1 TB for every 14 hours of raw data, 50 TB for a month-long run
- Doubles when the data is converted to strax format (can do work-arounds)



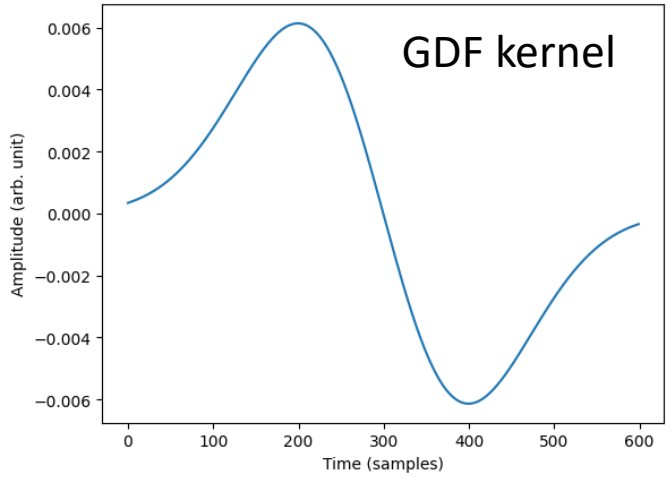
Triggers

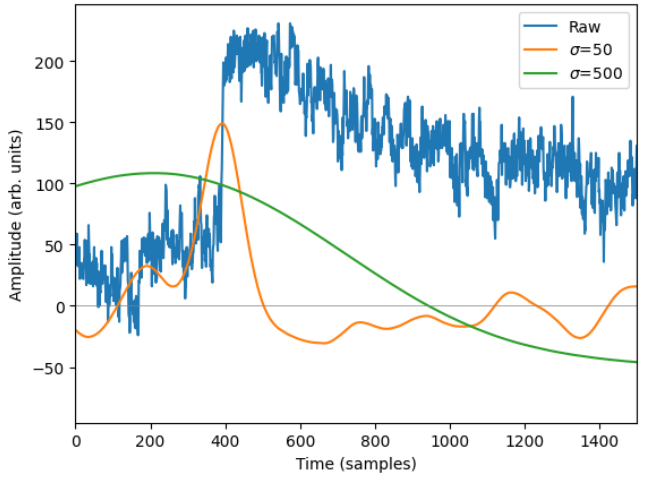
- Base trigger plugin: takes a filter kernel as a config, convolve records with the kernel, applies activation/deactivation triggering with an optional holdoff
- Custom triggers are “child plugins” that inherit the base trigger plugin
- Multiple triggers can be used simultaneously

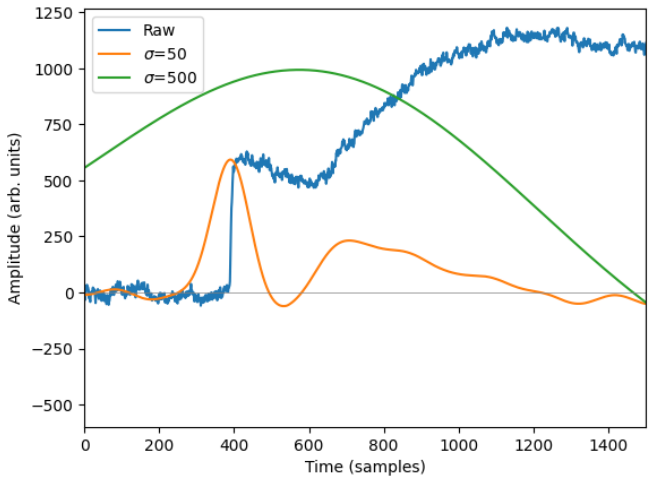


Triggers

- Gaussian derivative filter (GDF) – fast recovery, no echoes, no false triggers on fall edges of muons
- Two GDF triggers: one with narrow kernel for UV signals, one with wide kernel for QP signals





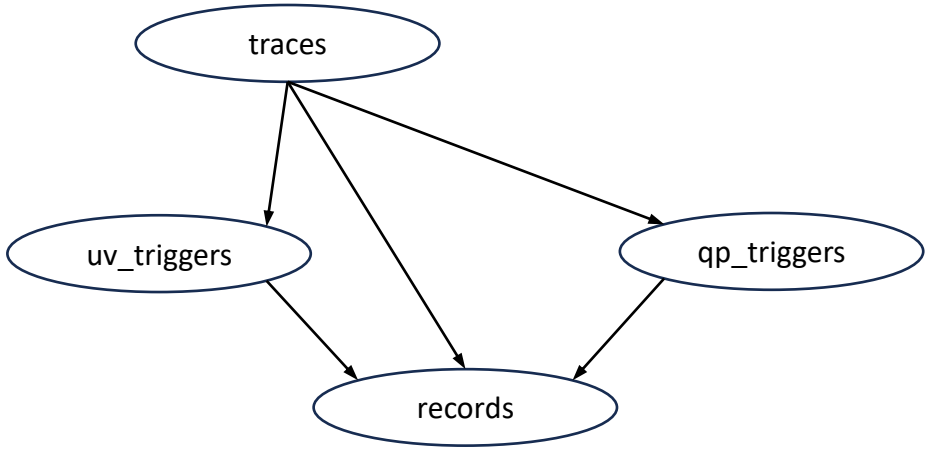


- Slow! 5 min for 5 min of raw data

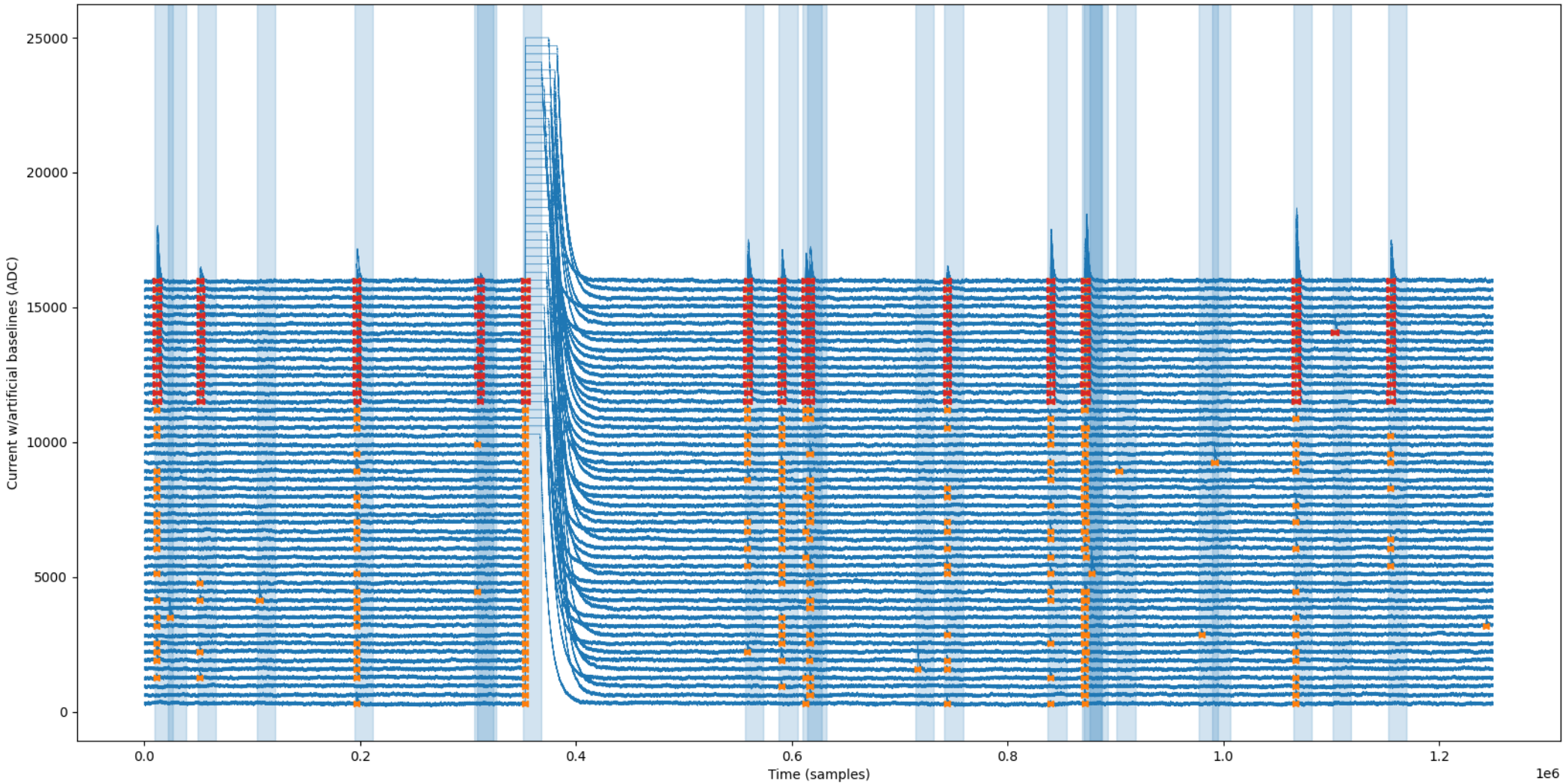
Traces (event building)

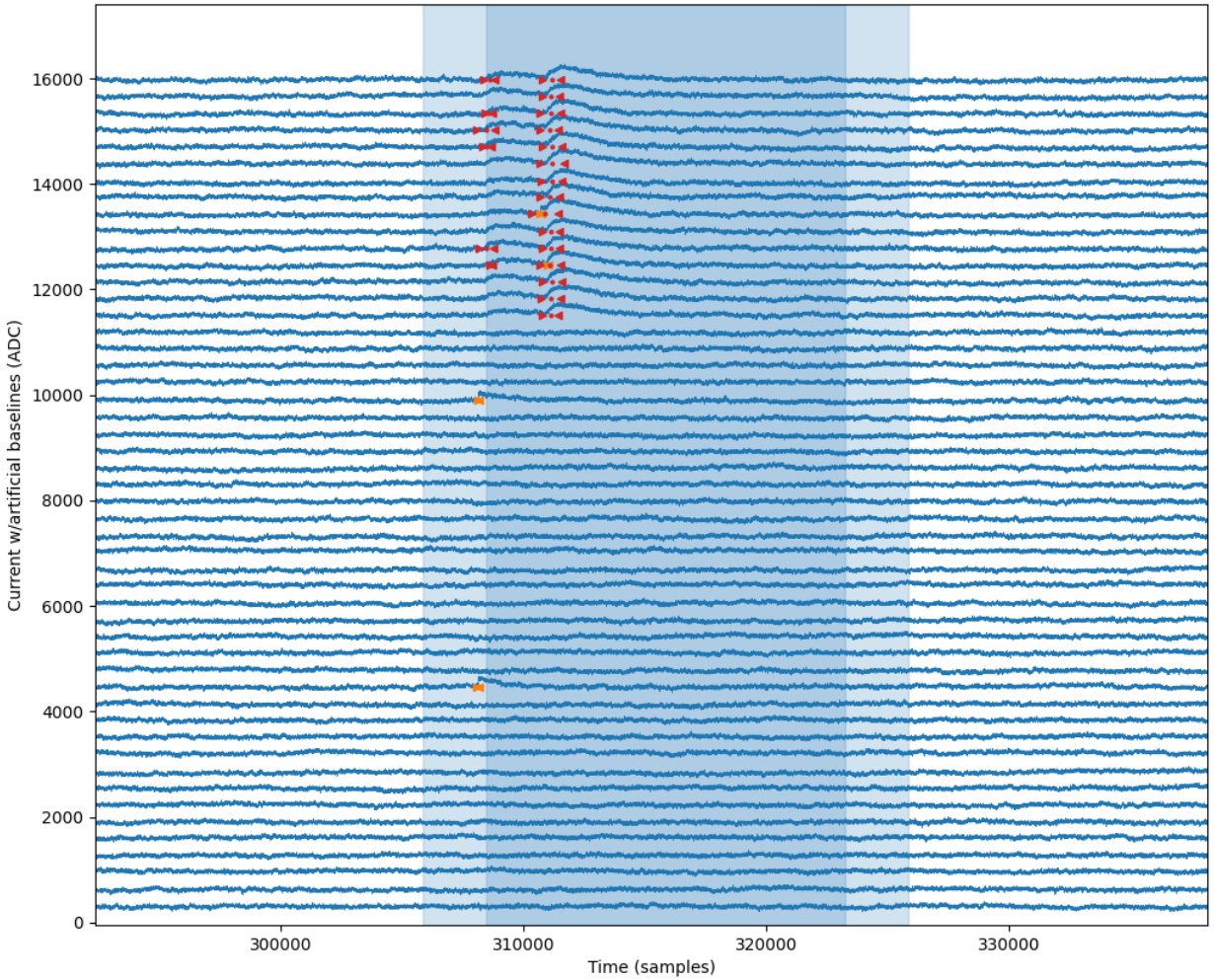
- Triggers within a configurable window are assigned to 1 event
- Allowed fit shifts should be the same as this window to prevent double-counting

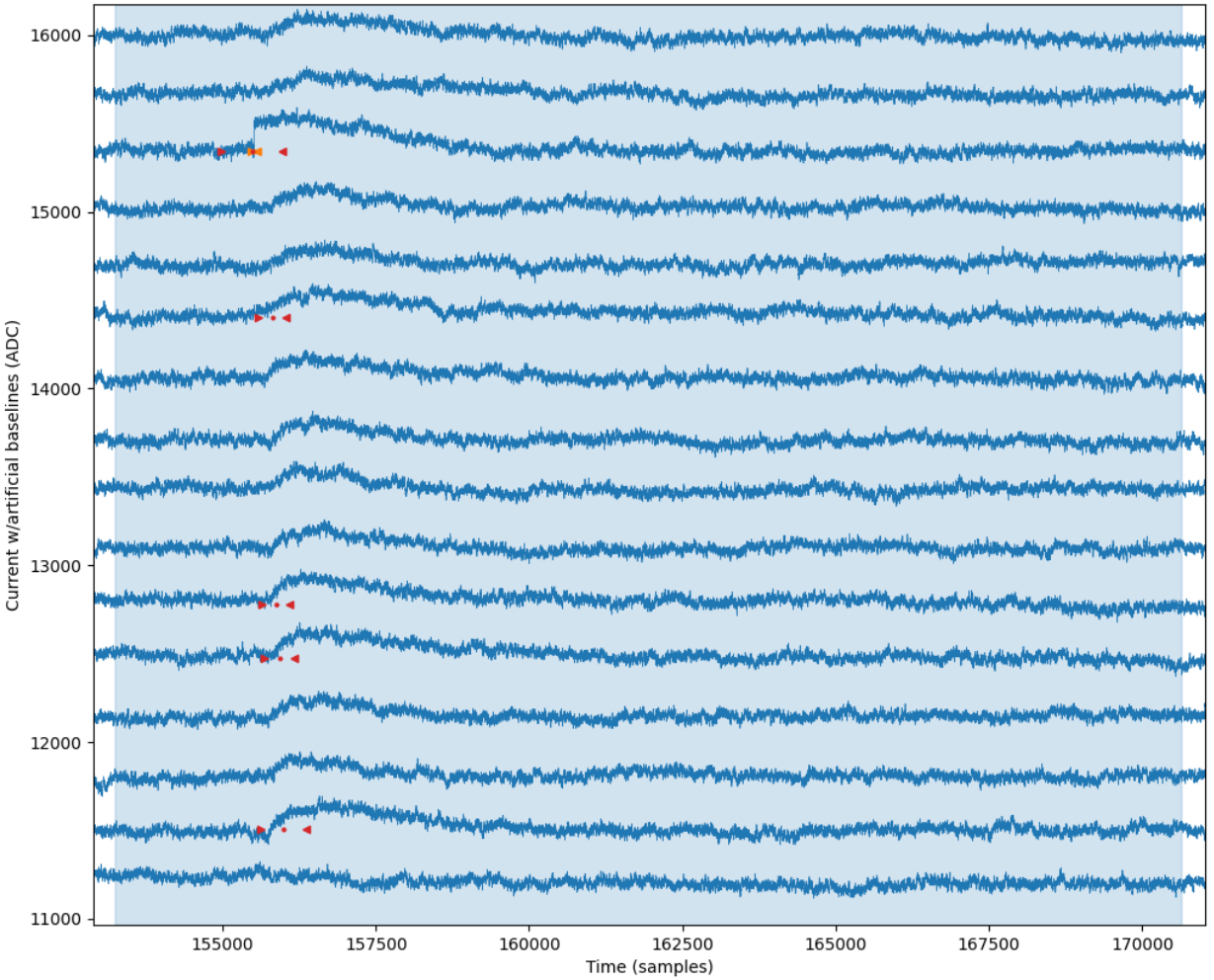
- For each event, 56 traces are saved:
  - 50 for each channel
  - Sum of all channels
  - Sum of all submerged channels
  - Sum of all vacuum channels
  - Sum of all channels with triggers
  - Sum of all submerged channels with triggers
  - Sum of all vacuum channels with triggers



- Traces a float64 arrays

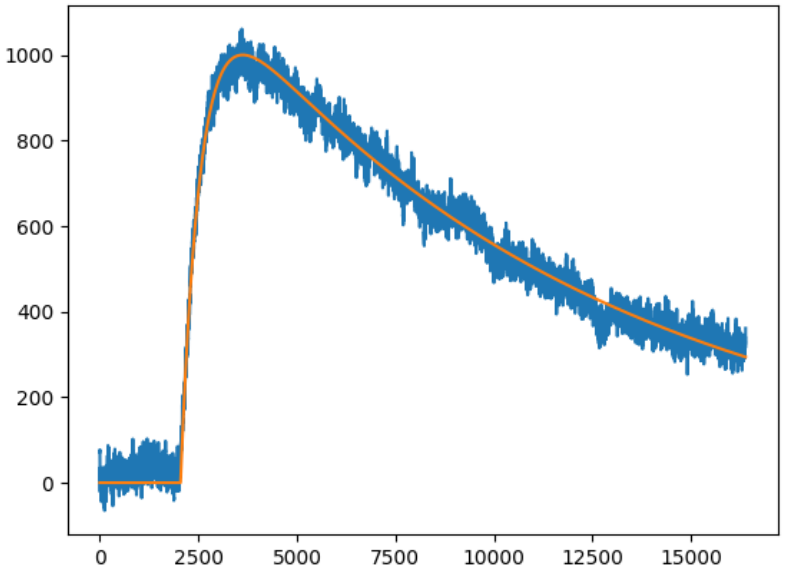

25000
20000
15000
10000
5000
0
Current w/artificial baselines (ADC)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e6
Time (samples)


Current w/artificial baselines (ADC)
Time (samples)
0
2000
4000
6000
8000
10000
12000
14000
16000
300000
310000
320000
330000


Current w/artificial baselines (ADC)
Time (samples)
11000
12000
13000
14000
15000
16000
155000
157500
160000
162500
165000
167500
170000

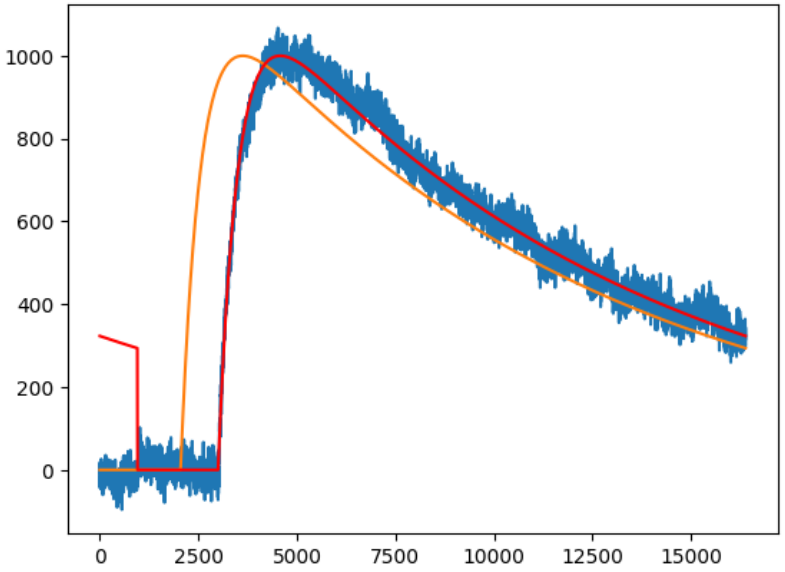
OF: rolling vs sliding

- QP template can have a very long tail
- But you don’t need the tail to do a fit



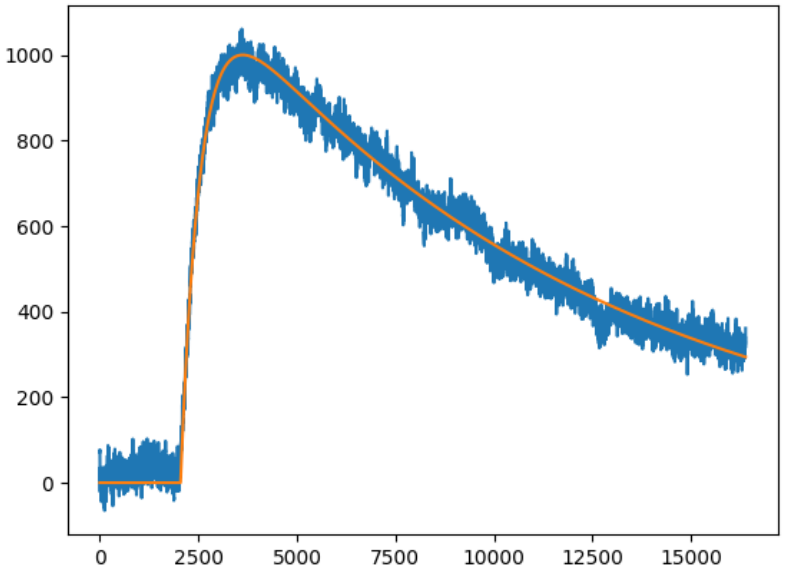
- Problem: our current OF definition assumes a rolling template. This does not work if the template does not go to 0

$$\chi^2 = \int_{-\infty}^{\infty} \frac{|v(f) - Ae^{-i\omega t_0} s(f)|^2}{J(f)} df$$



OF: rolling vs sliding

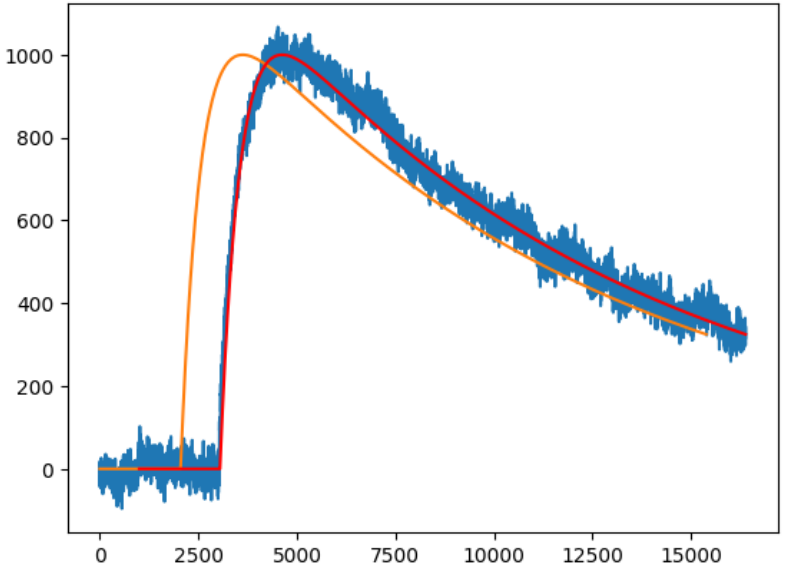
- QP template can have a very long tail
- But you don’t need the tail to do a fit



- Problem: our current OF definition assumes a rolling template. This does not work if the template does not go to 0

$$\chi^2 = \int_{-\infty}^{\infty} \frac{|v(f) - Ae^{-i\omega t_0} s(f)|^2}{J(f)} df \quad \Rightarrow \quad \chi^2 = \int \frac{|v(f) - As(f|t_0)|^2}{J(f)} df$$

scipy.signal.ShortTimeFFT



Strax bug-fix

- Found a small bug in strax – time.time() was used to calculate the time elapsed from the last update of a progress bar. However, on some systems time.time() resolution is low, it can return the exact same value if called twice within a short time interval. Caused a “division by zero” crash
- Attempted to fix it, submitted a pull request. Strax maintainers (Dacheng and Yue) came up with their own solution and merged it within the same day