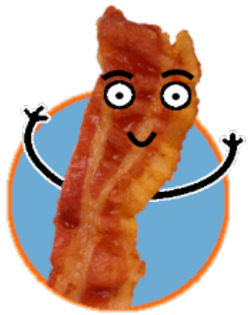
LARDON

LIQUID ARGON RECONSTRUCTION DONE IN PYTHON



Laura Zambelli - LAPP / CNRS

April 11th 2024

Wirecell 2nd Summit - BNL

Goal, Reasons, History

Started in 2019 with ProtoDUNE-DP operation
- At first I wanted have a quick event display and inspect the data
- Written in python as a personal challenge

It was then used to understand, study and filter the noise of ProtoDUNE-DP data
- Noise filtering requires ROI - easily transformed into hit finder
- I re-used some reconstruction code I wrote for QScan (DP own reconstruction/simulation code)
- Being in a COVID lockdown helps a lot to develop a framework

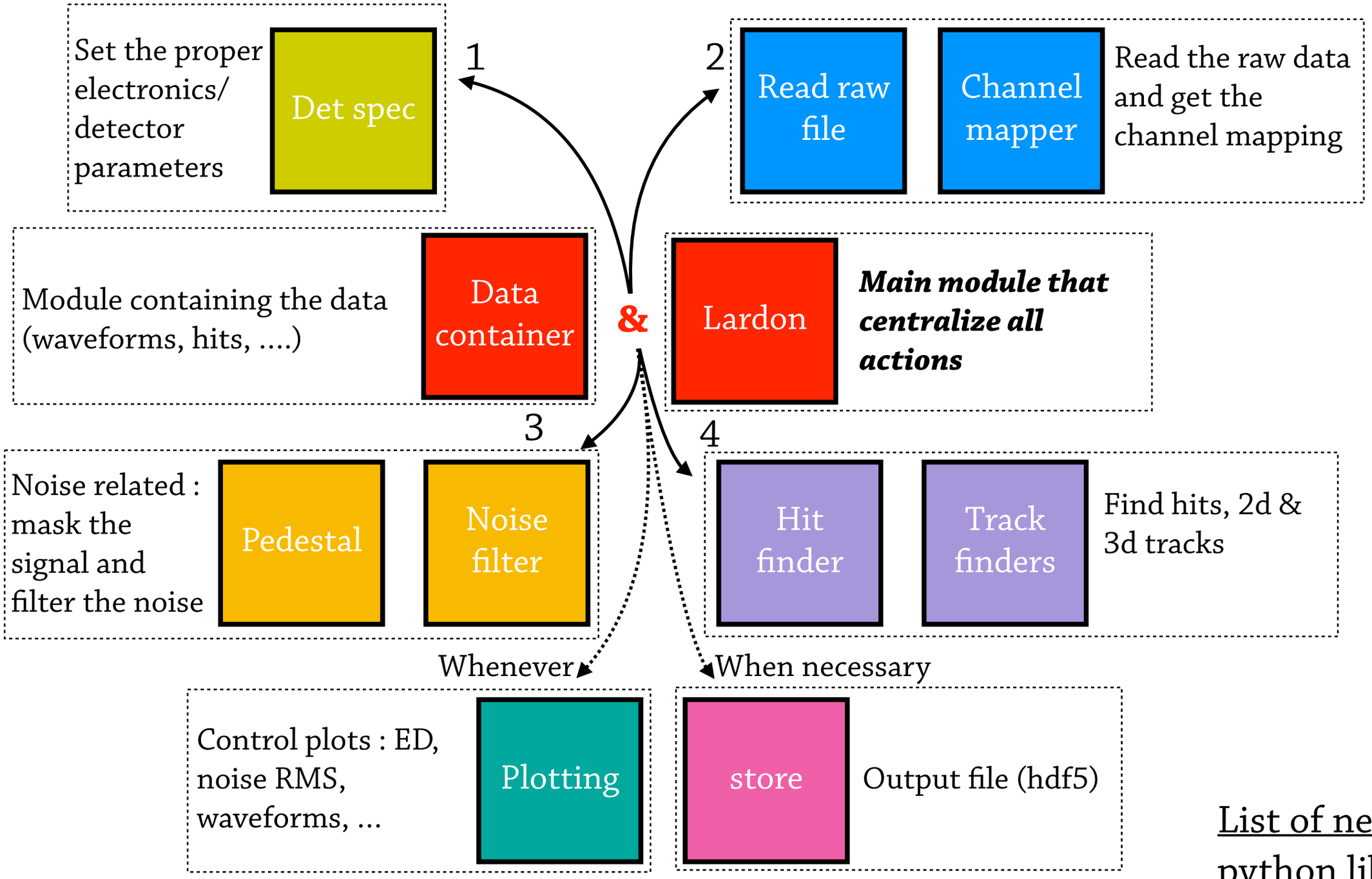
ProtoDUNE-DP data was reconstructed with LARDON together with LArSoft
- Consistent results obtained, DP paper under internal review

LARDON was adapted to the new Vertical-Drift design
- New features: 3-views, induction signal, Bottom-electronics
- First read and analyzed 50L data (VD small R&D setup)
- Successfully reconstructed all VD-Coldbox data campaign for both top and bottom CRPs

LARDON is completely independent from LArSoft, Art and ROOT

LARDON code flow

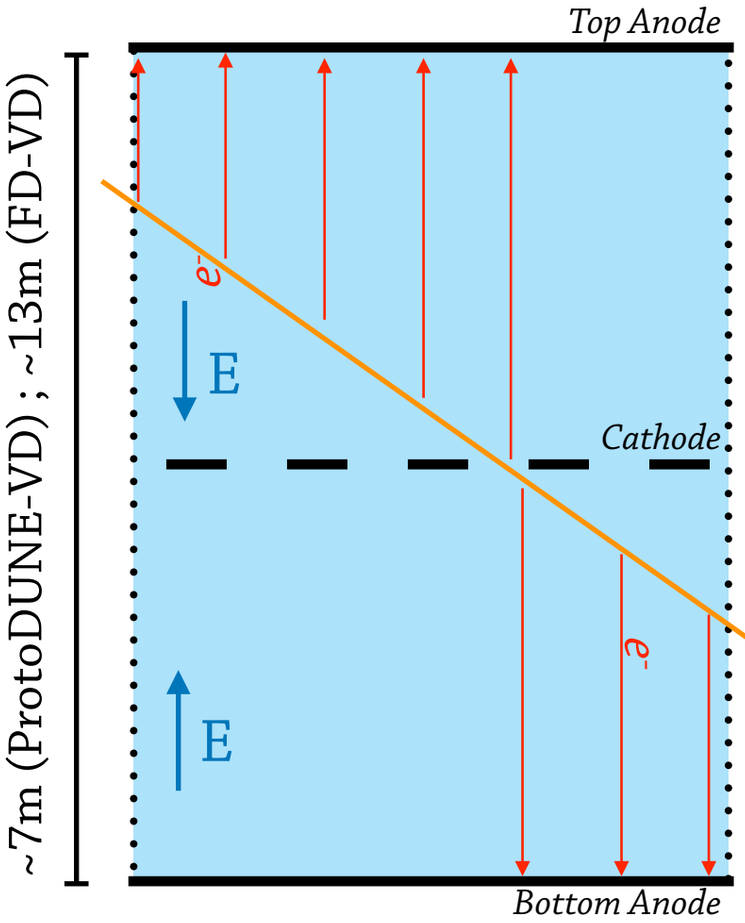
The code is on GitHub at : https://github.com/dune-lardon/lardon

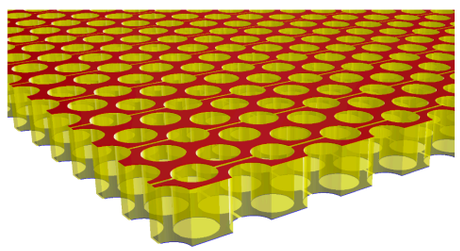


List of needed python libraries

Quick summary on VD-design

The drift is separated into 2 volumes, the cathode is hanged in the middle
Anodes reads the electrons at the top and the bottom of the drift volume
-> Equipped with different electronics ('Top' and 'Bottom' Electronics)



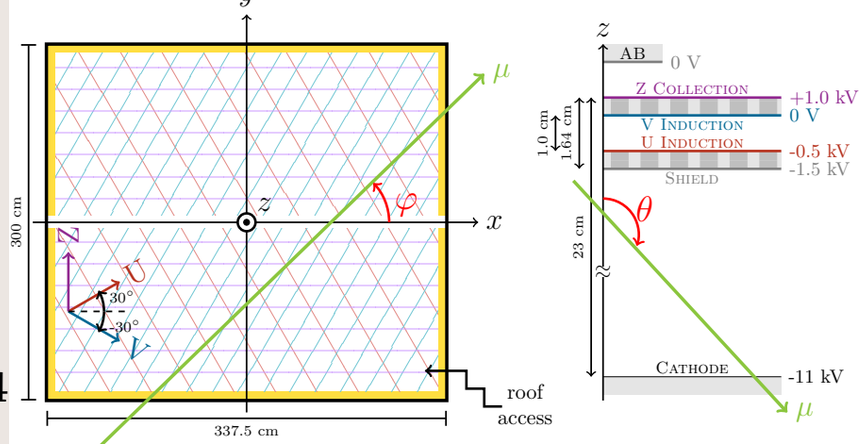


The anodes are made of 2 drilled PCBs:

- The PCB faces (View) are etched to make strips
  -> Three orientations [±30°, +90°]
- Electrons leave an induction signal (on View 0 & 1) and are collected on View 2

PCBs are held together by the 'Charge Readout Plane' : CRP [3×3.4 m²]

ColdBox schematic drawing:



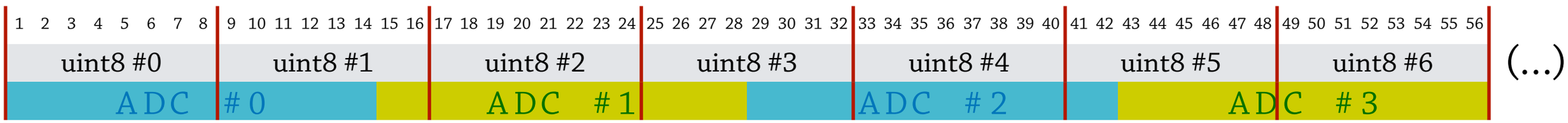
Four CRPs installed in ProtoDUNE-VD
-> Each tested in real conditions in the ColdBox with cosmic data

Decode & store the data

numpy
numba

The raw file format is a HDF5 file, that contains DAQ information and data (from charge and PDS)
-> LARDON directly reads the raw data and headers, and so is Art-free

The 14-bit ADCs are read as multiple 8-bit uint and bit operations are performed to retrieve



↳ These functions are accelerated by `numba`

From the headers, lardon gets the event timestamp and some run informations.
Lardon can now handles the case when one (or more) WIB frames are not written by the DAQ.

Waveforms are stored in a `numpy` array of shape `(crp, channel, tick)`:

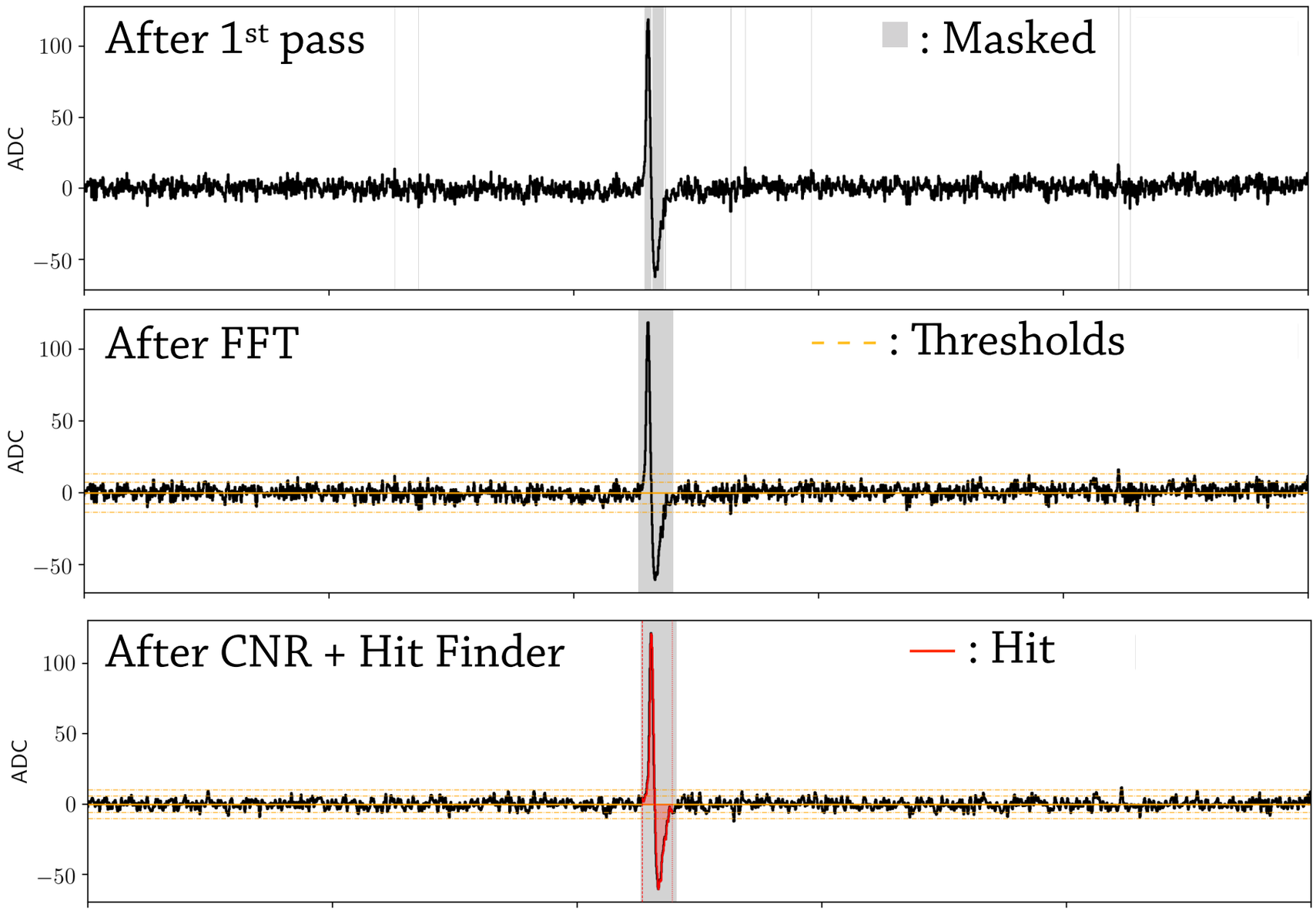
- Channel ordering is in 'DAQ' ordering *ie* in the order of reading the raw data
- Not user-friendly, as consecutive channels can be from different views
- Pedestal, ROI and hit finder are independent operations on waveforms

1 CRP = 3072 channels
Coldbox : 1 CRP
ProtoDUNE-VD: 4 CRPs
FD-VD: 160 CRPs

Corresponding 'shadow' mask array `(crp, channel, tick)` for noise/signal separation
- Filled with Boolean where True/1 = Noise; False/0 = Signal
- Updated during pedestal/ROI calls during the noise filtering

Pedestal and ROI finder

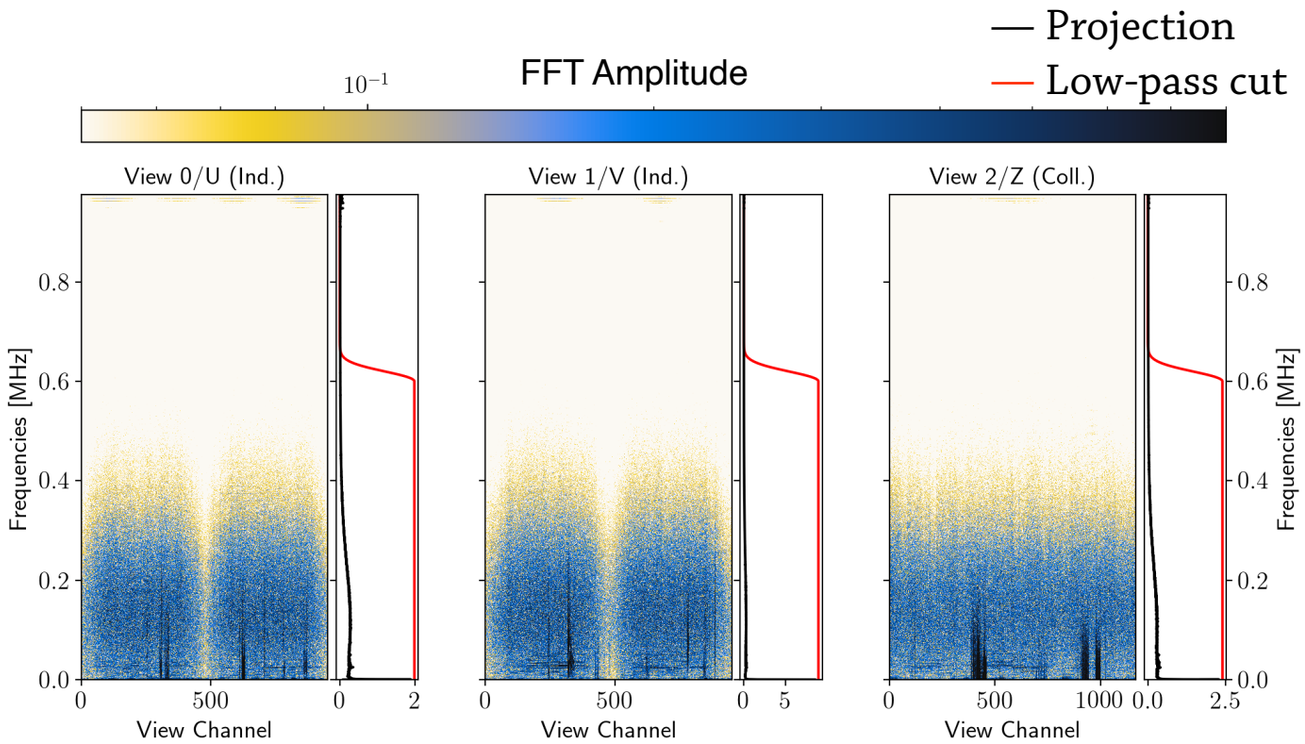
1. Scan each waveform and compute the mean and rms of the un-masked data points
2. Update the mask based on ADC thresholds (1st pass) and ADC thresholds + signal shape (subsequent pass)



The functions for pedestal mean + rms computation and ROI search are accelerated with **numba**

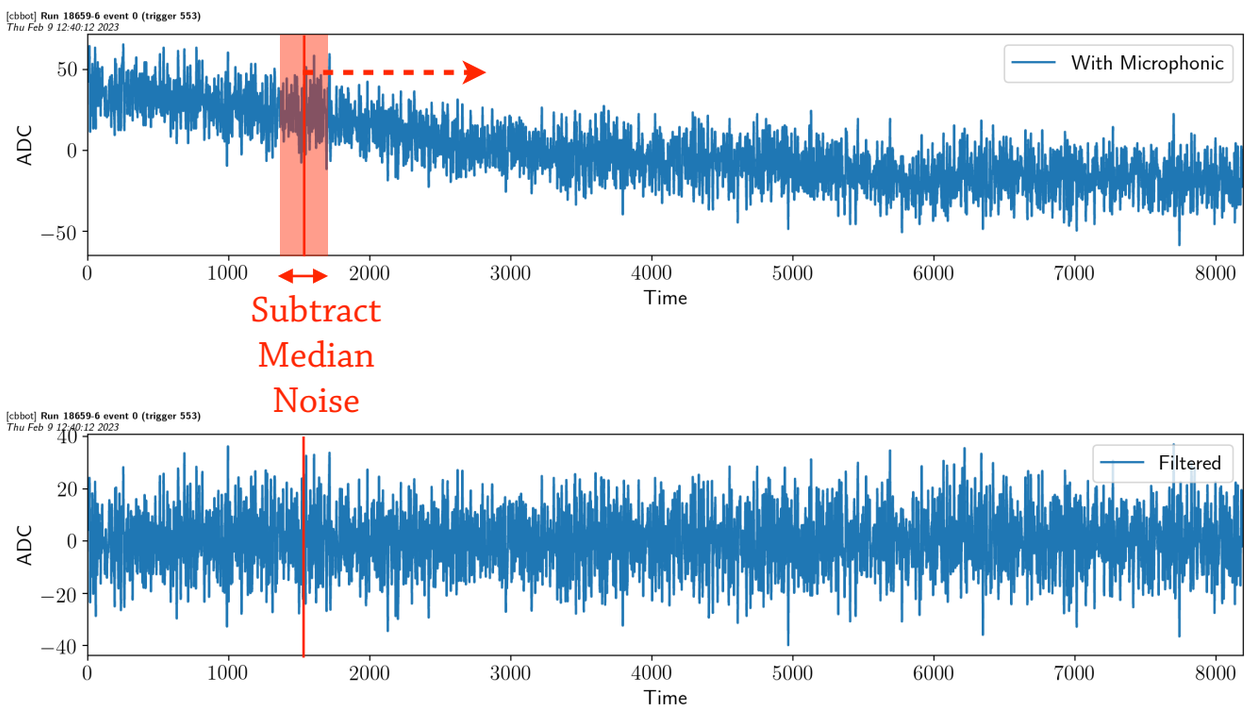
These 2 steps are called twice after each call to a noise filtering algorithm.

FFT low pass and Microphonic noise



numpy

A low-pass FFT filter is applied to the data. There is a smooth gaussian cut after the frequency threshold to avoid ringing artifacts.



bottleneck

The microphonic noise appears like a slow pedestal variation in space and time

-> Unclear origin ; probably from an internal vibration

<u>Microphonic-filter:</u>

At each time tick of each waveform, compute the median noise in a surrounding window

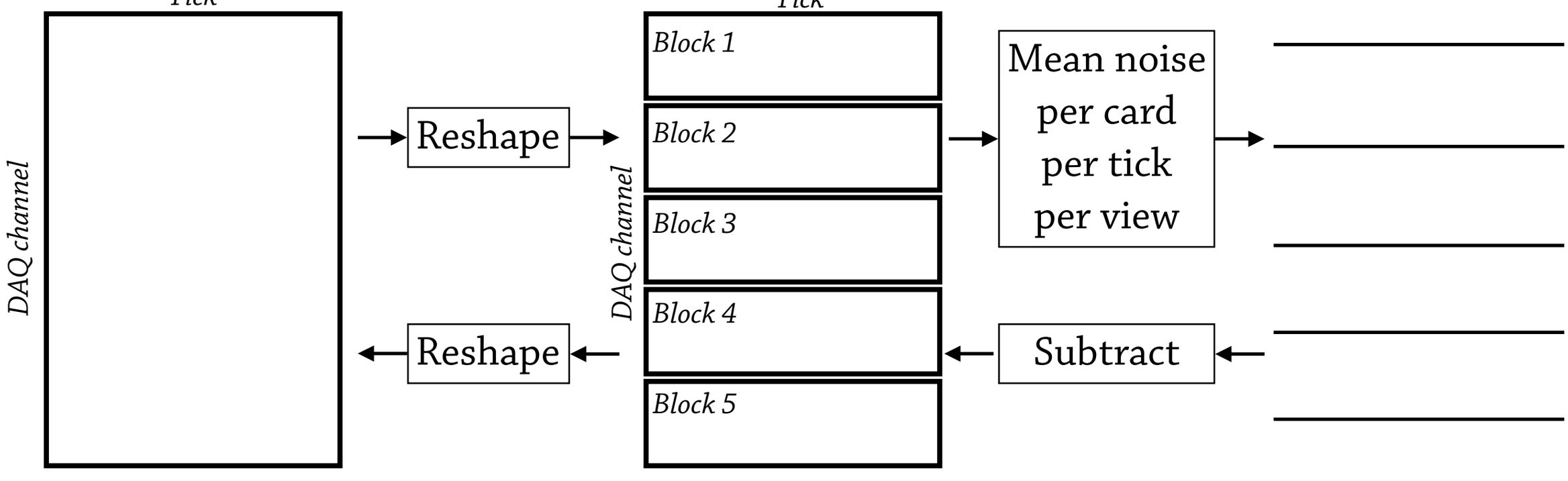
Coherent Noise Filtering

The coherent noise is shared among channels on the same electronic card and view
-> Profit from the data being organized in DAQ channel ordering

numpy

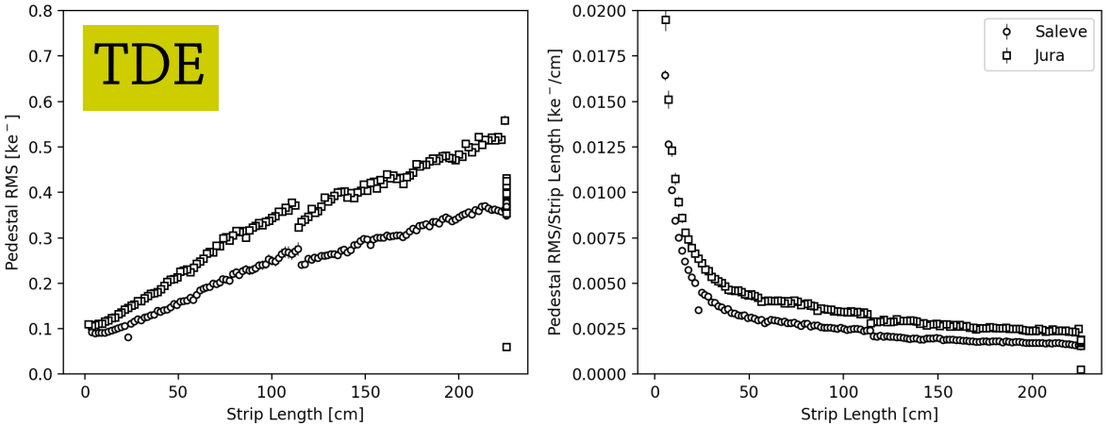
Coherent Noise Filter:

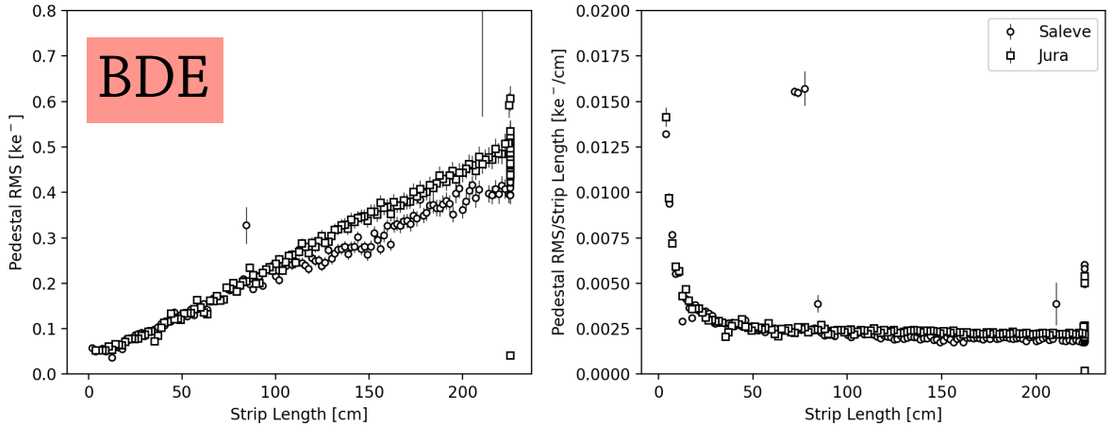
- The data array is reshaped into (sub)-blocks of electronic cards [*e.g* `(3072, 8192)` -> `(24, 128, 8192)`]
- In each view, the mean noise in block is computed and subtracted
- The array is reshaped to its original shape



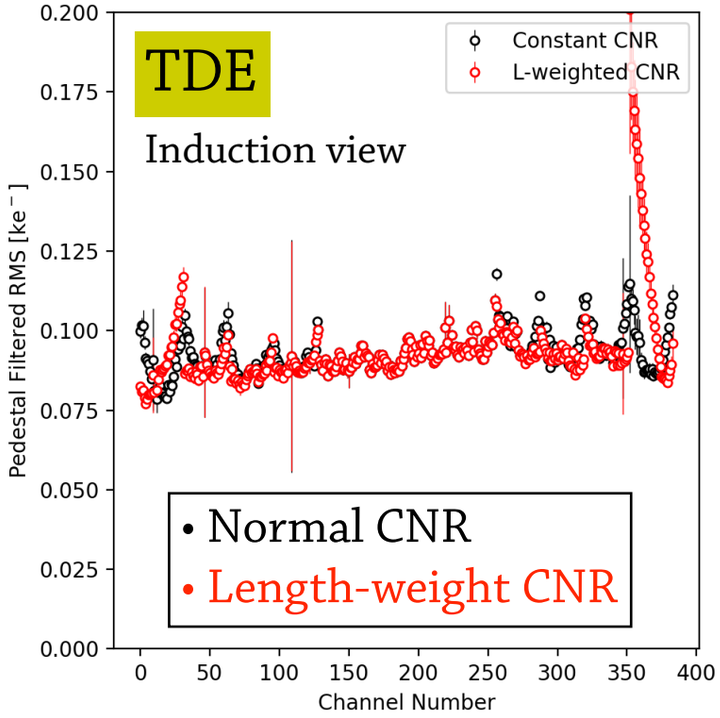
Coherent Noise Filtering

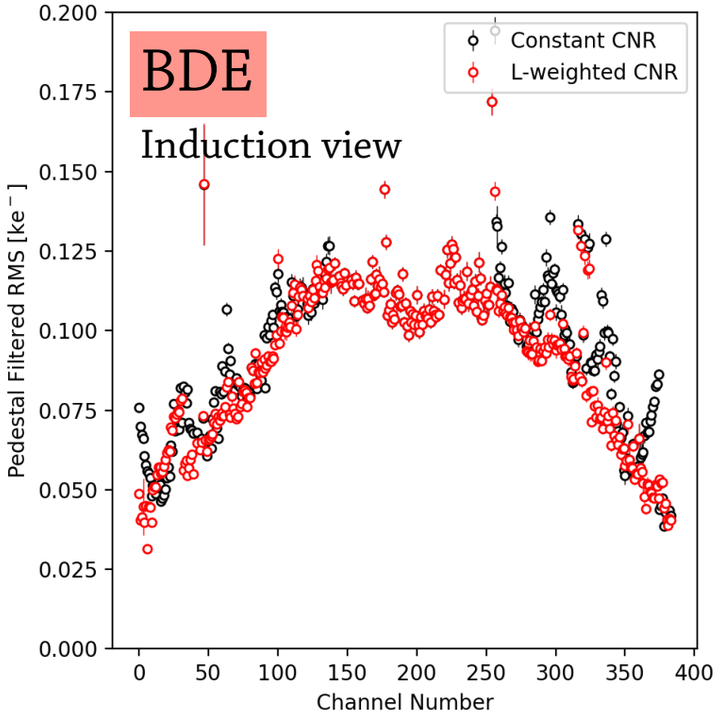
NB : plots made with CRP1 data (different geometry)





The dependance of the noise with the strip length is different for top and bottom electronics





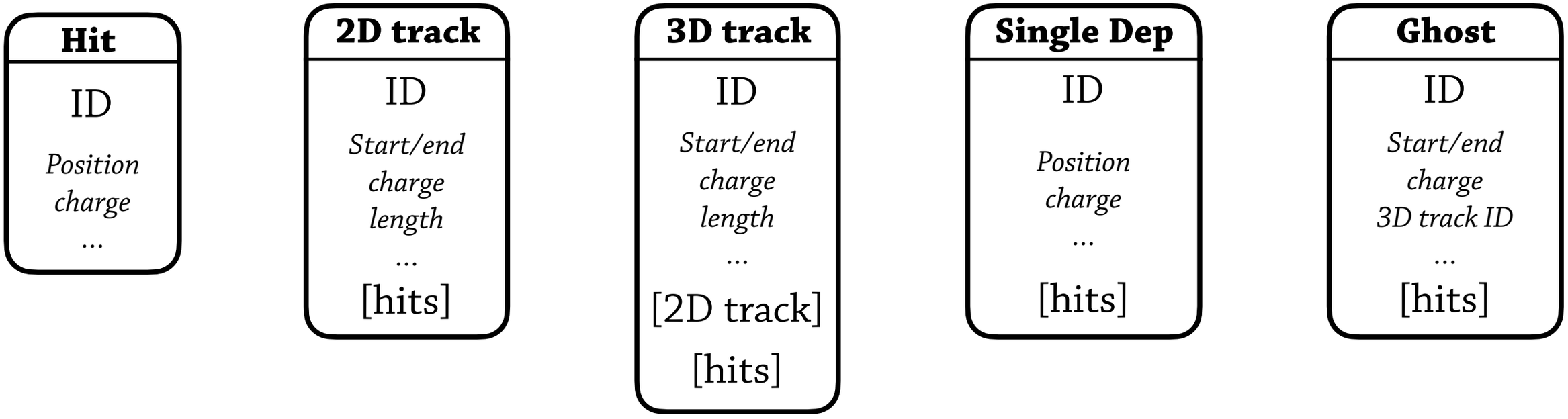
The CNR procedure needs to take this into account to be useful for the small induction strips

-> In lardon the noise is weighted by the strip length in the BDE case

Reconstructed Containers

Reconstructed objects (hits, 2D tracks, 3D tracks, Single deposition, Ghosts) are stored as a list of class objects with :

- General informations (like position, charge, length, angle, …)
- Single ID
- List of sub-containers (informations and ID)



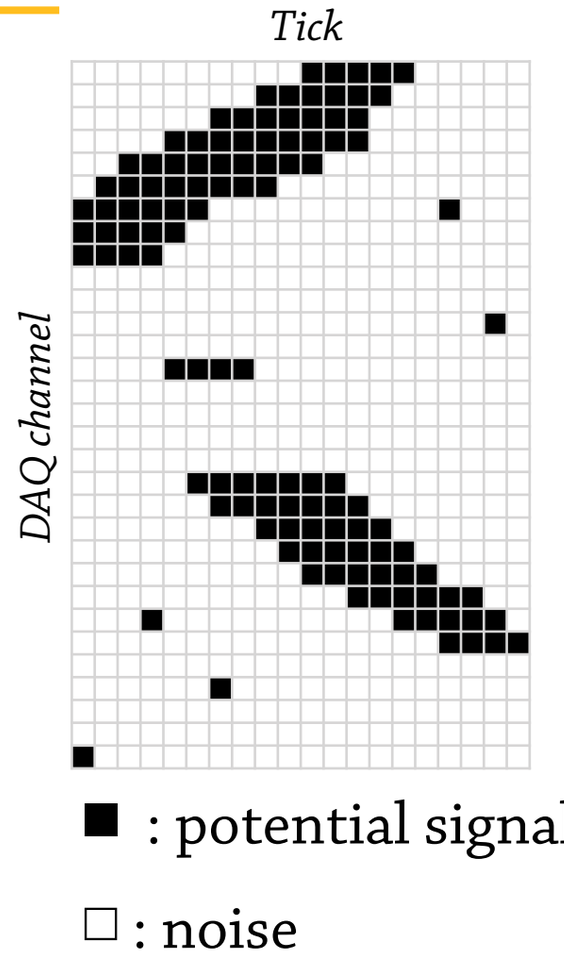
Hit Finder

numpy
numba

The mask array tells wether an ADC point is consider noise or signal
-> Search for long enough consecutive ‘signal’ blocks to make a ‘ROI’

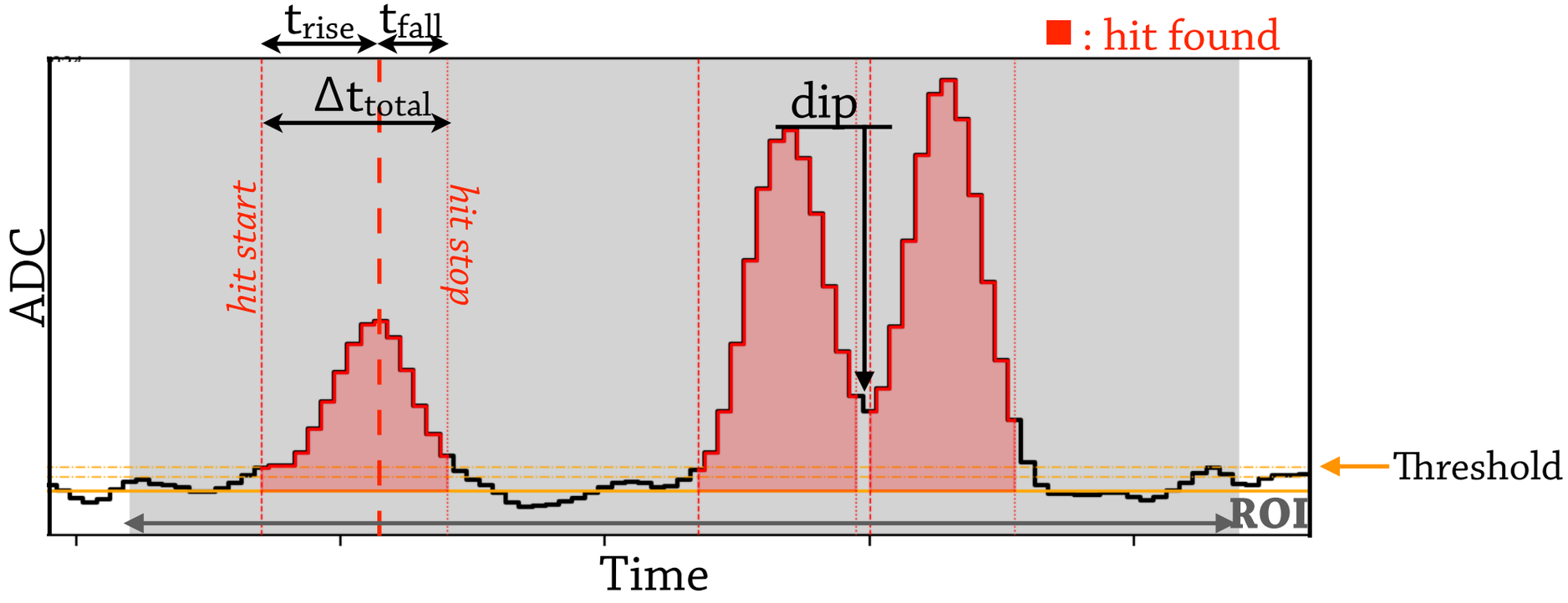
For the ***collection*** view: search for unipolar hit(s)

A hit is defined from rise and fall times above a threshold based on the pedestal RMS
In case of overlapping hits, a dip condition is also required , multiple hits can be found in one ROI



■ : potential signal

□ : noise



Hit Finder

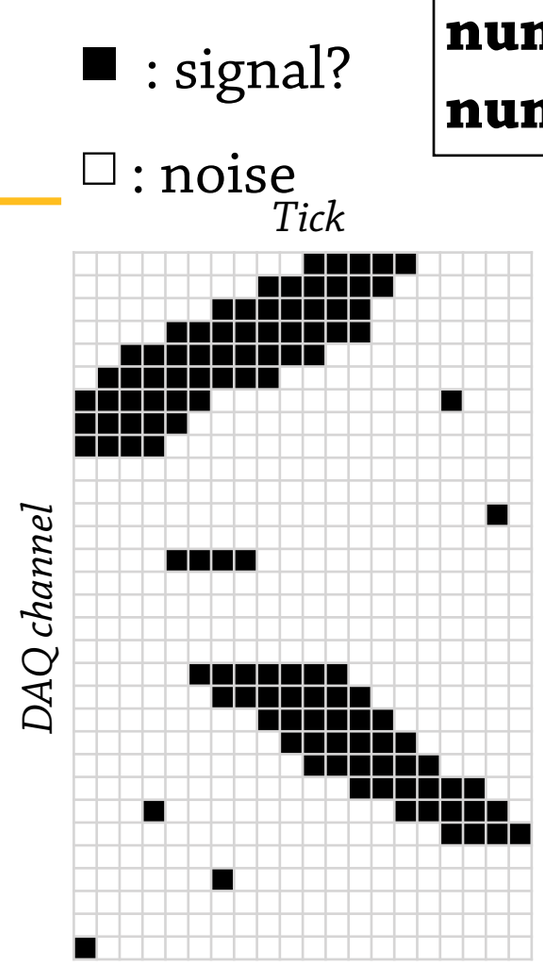
numpy
numba

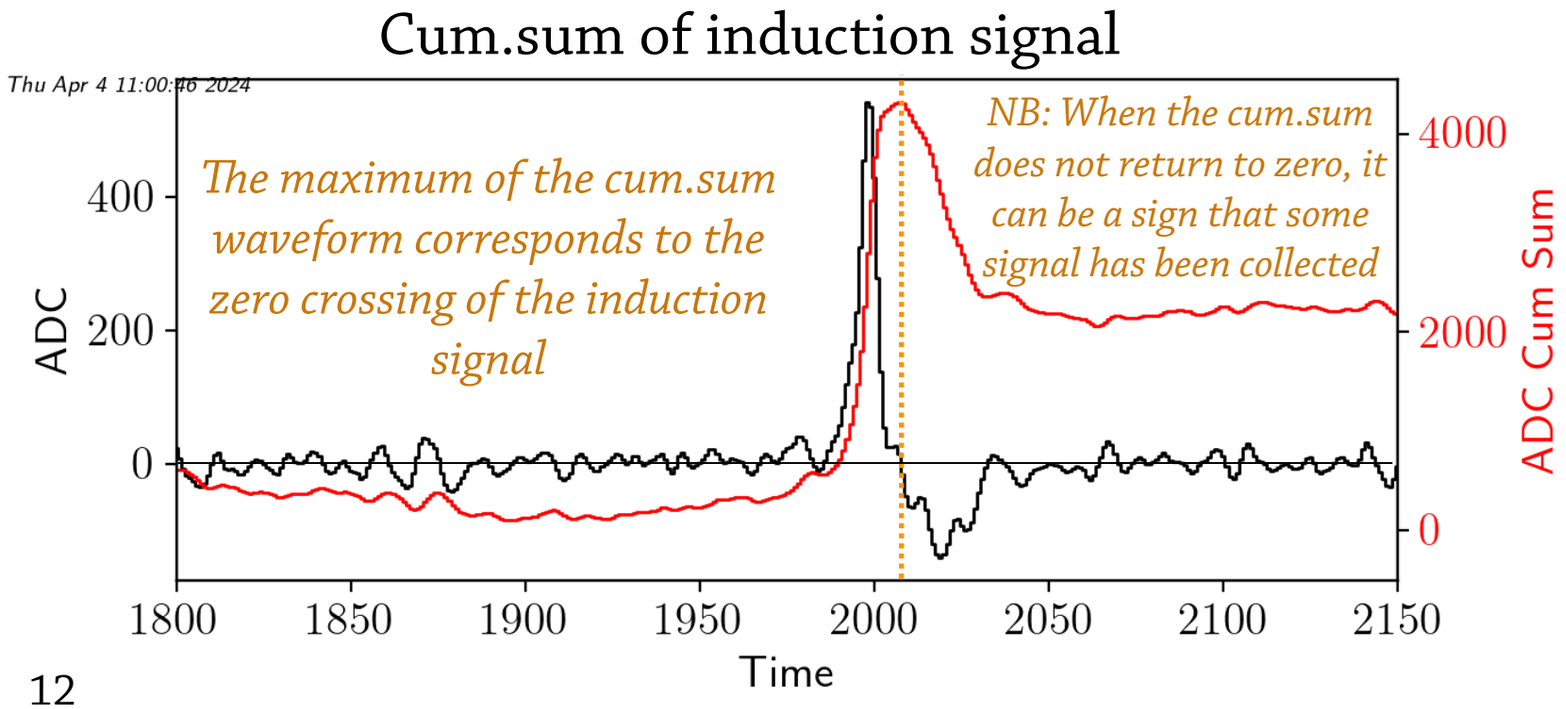
The mask array tells wether an ADC point is consider noise or signal
-> Search for long enough consecutive ‘signal’ blocks to make a ‘ROI’

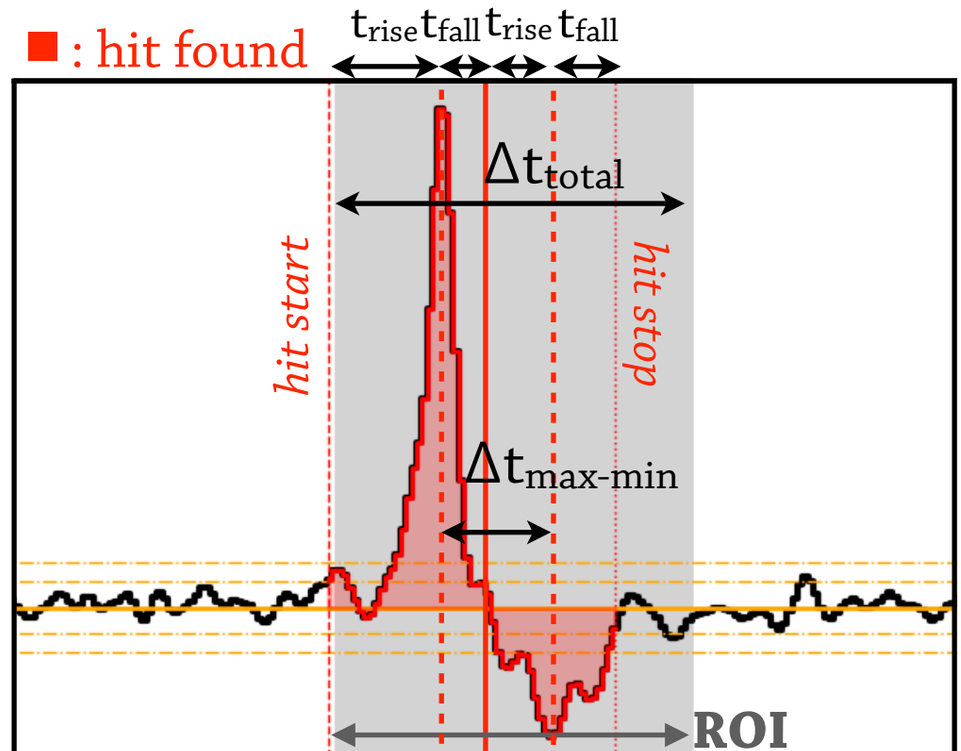
For the ***induction*** views:

1. Make a cumulative sum of the ADC in the ROI
   —> transforms a bipolar signal into a unipolar signal
2. Search for unipolar hit(s) in the cum.sum waveform with similar criteria as for the collection view
   ↳ Gives hit(s) time limits
3. Within the hit time limits, search for bipolar hits

4. In case step 2 and 3 are unsuccessful, search for unipolar hit(s) in the original ROI







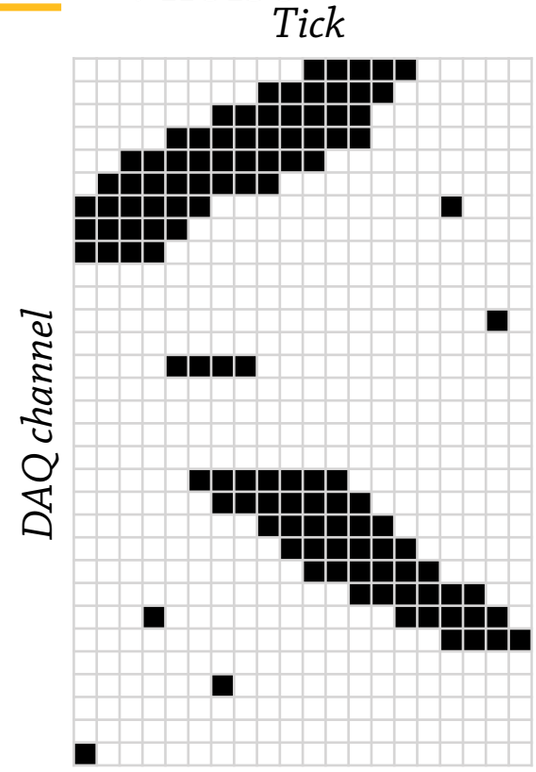
Hit Finder

numpy
numba

The mask array tells wether an ADC point is consider noise or signal
-> Search for long enough consecutive ‘signal’ blocks to make a ‘ROI’

■ : signal?

□ : noise

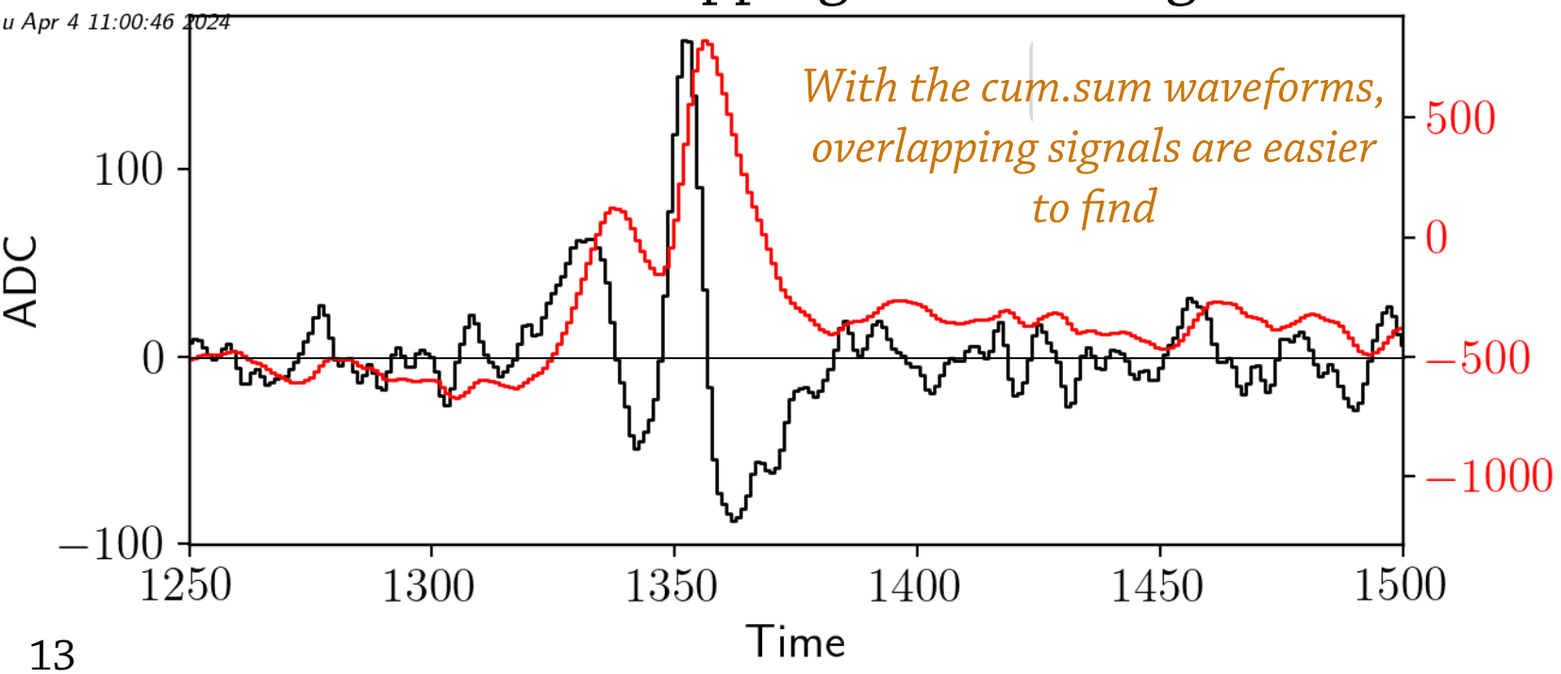


For the ***induction*** views:

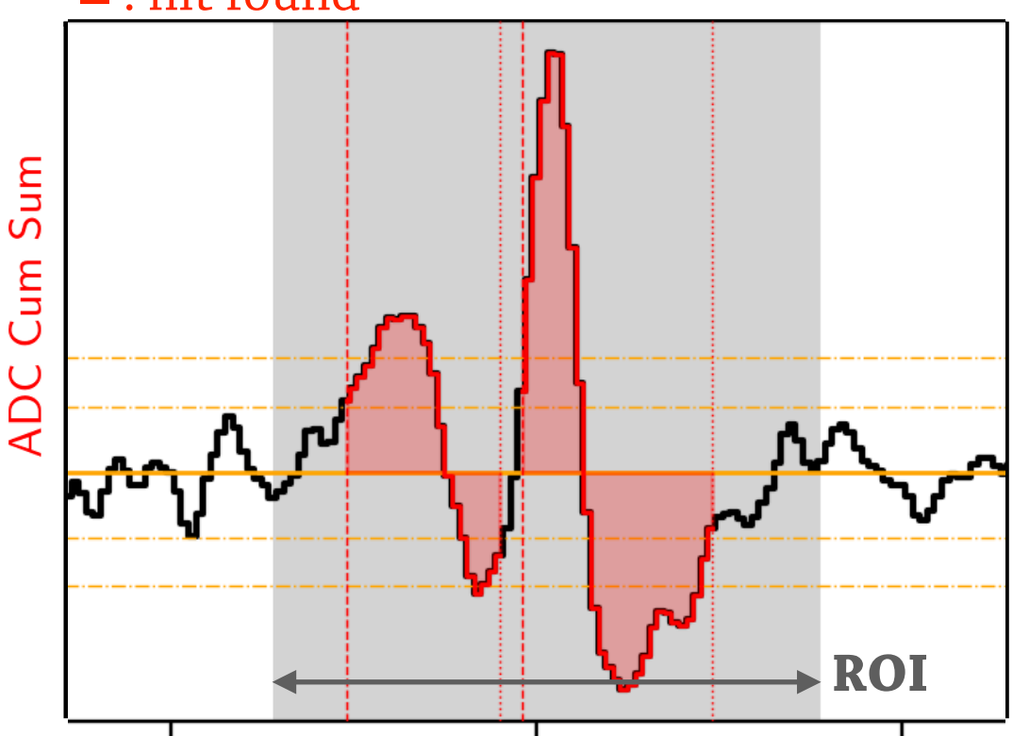
1. Make a cumulative sum of the ADC in the ROI
   —> transforms a bipolar signal into a unipolar signal
2. Search for unipolar hit(s) in the cum.sum waveform with similar criteria as for the collection view
   ↳ Gives hit(s) time limits
3. Within the hit time limits, search for bipolar hits

4. In case step 2 and 3 are unsuccessful, search for unipolar hit(s) in the original ROI

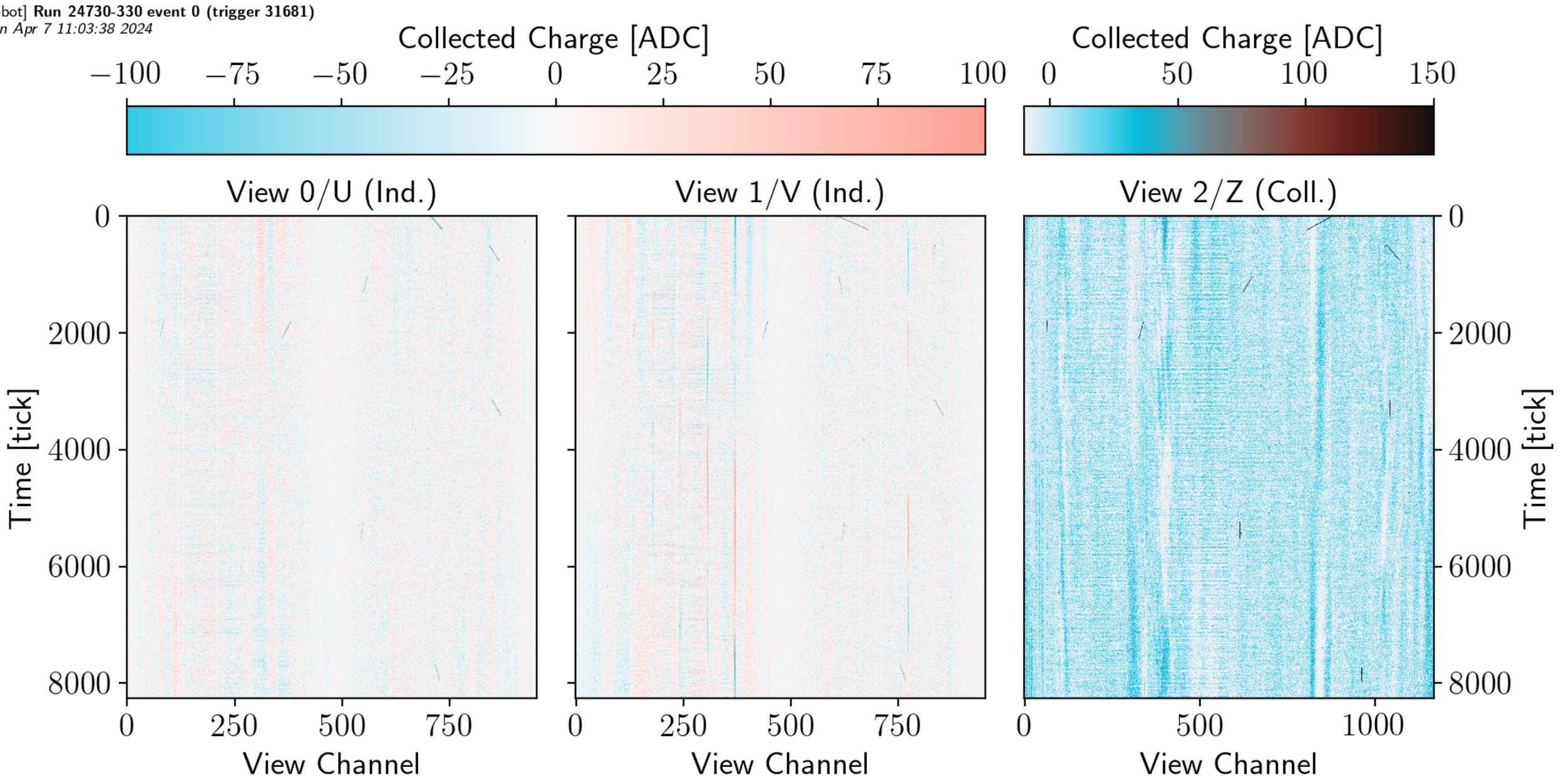
Cum.sum of overlapping induction signals



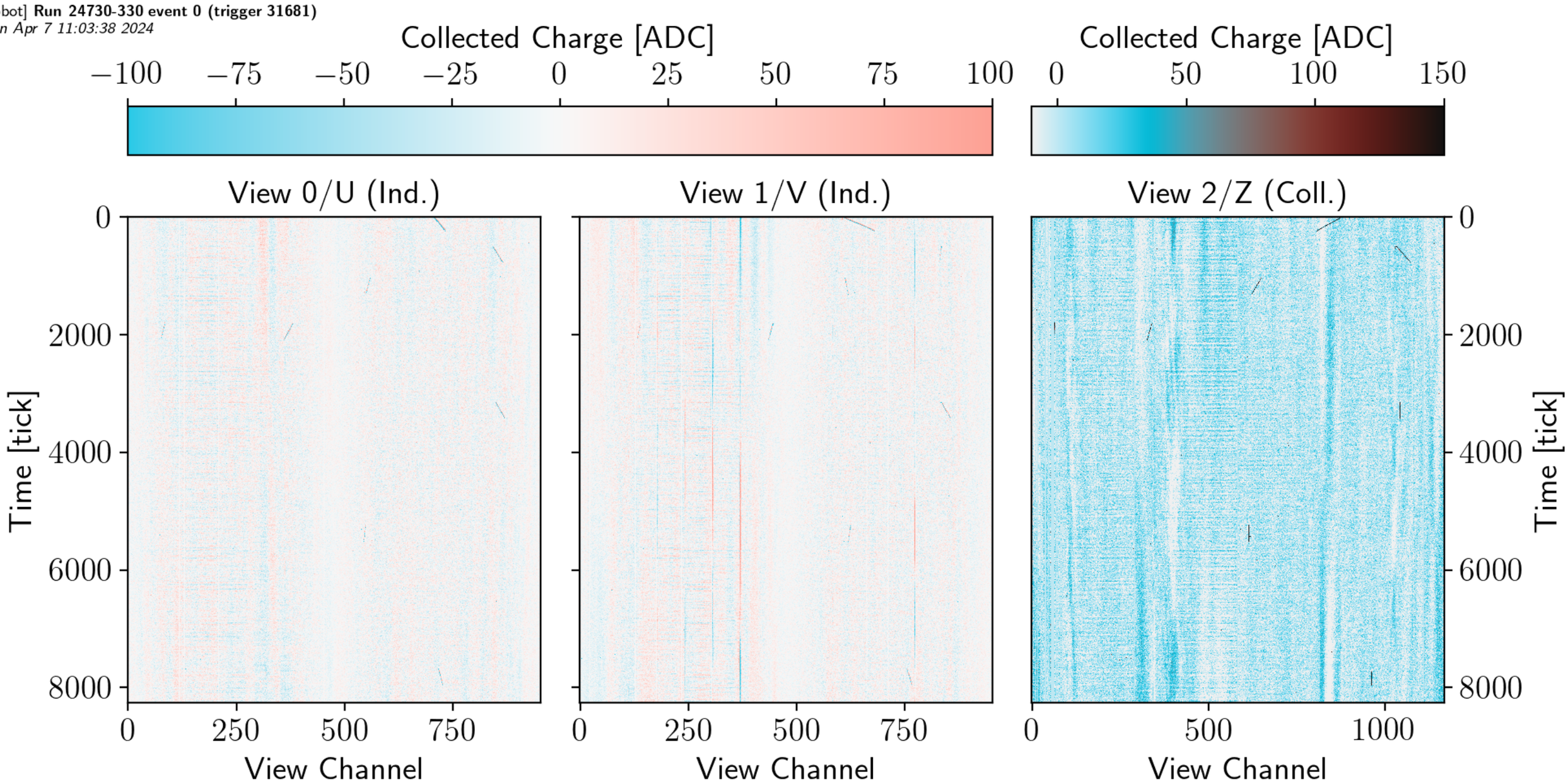
■ : hit found



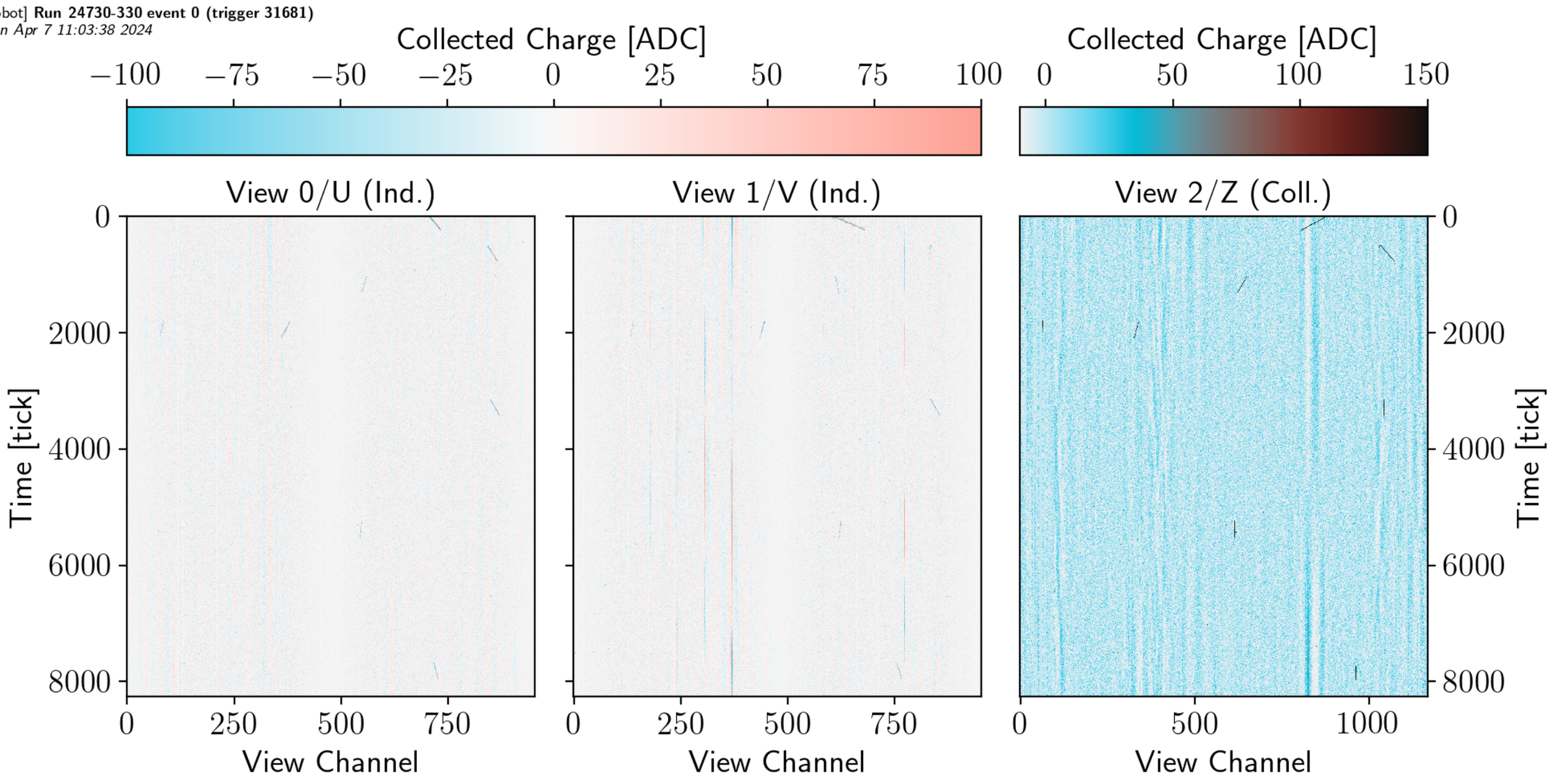
Example - raw event



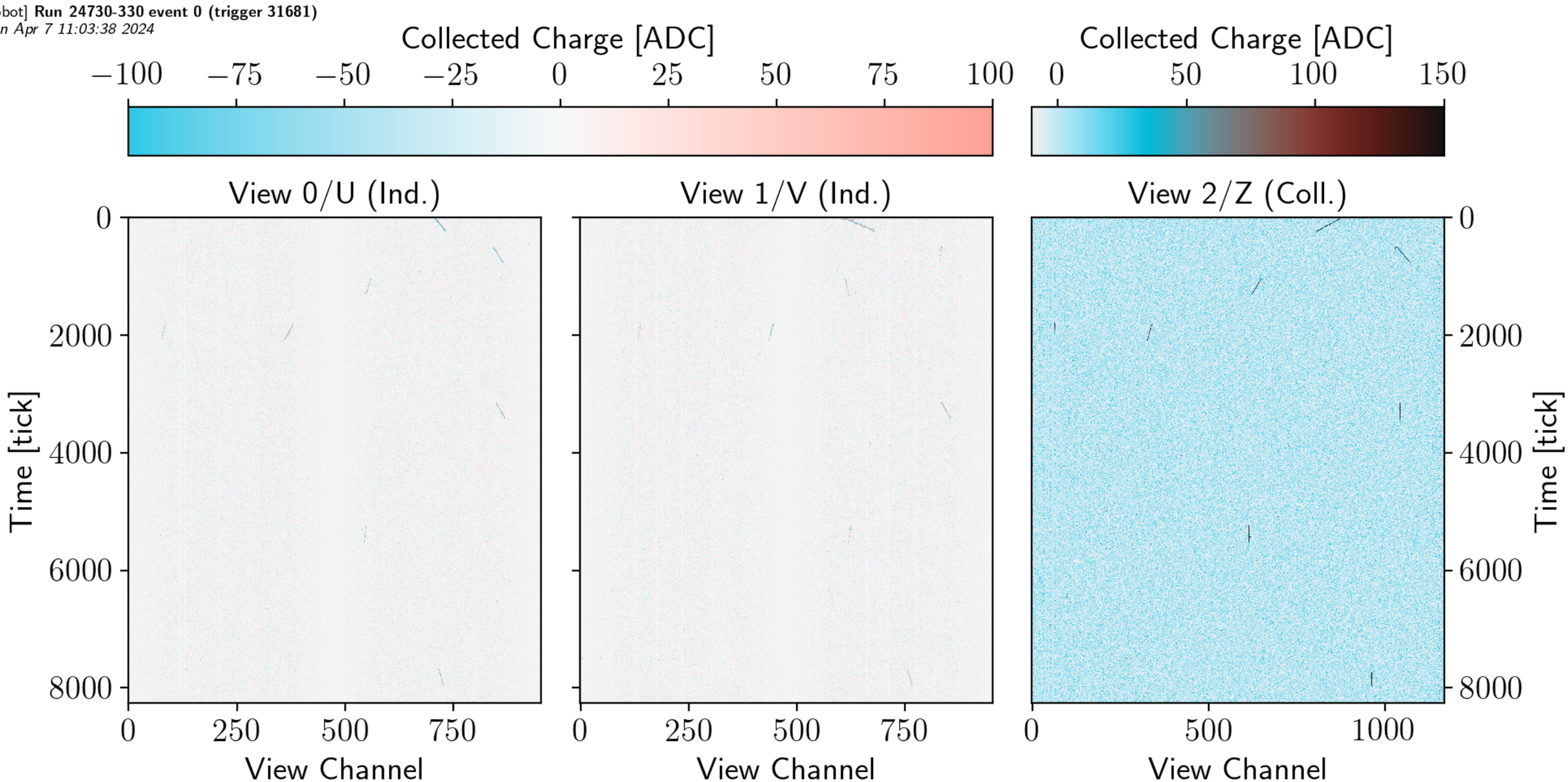
Example - after FFT



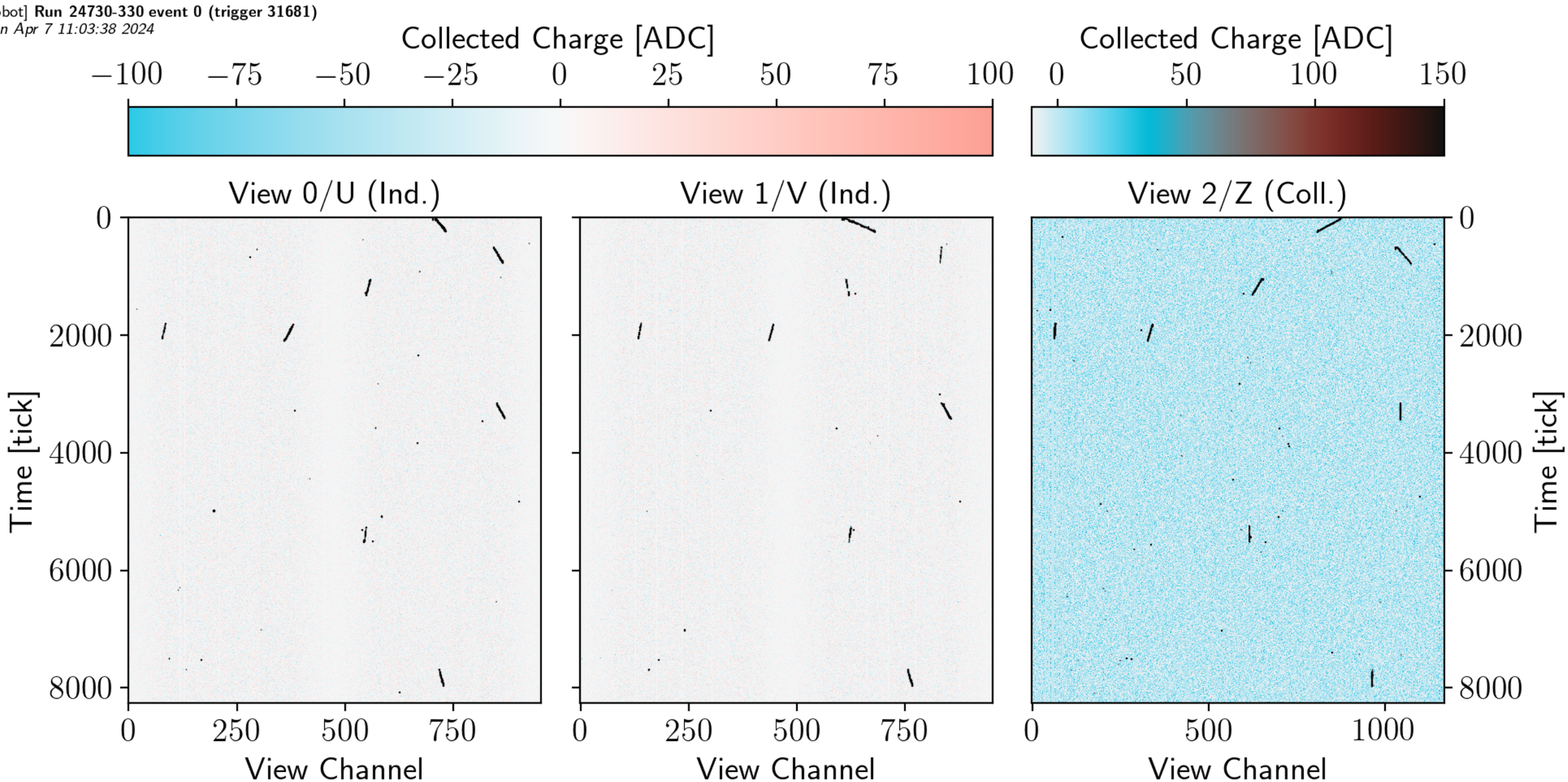
Example - after CNR



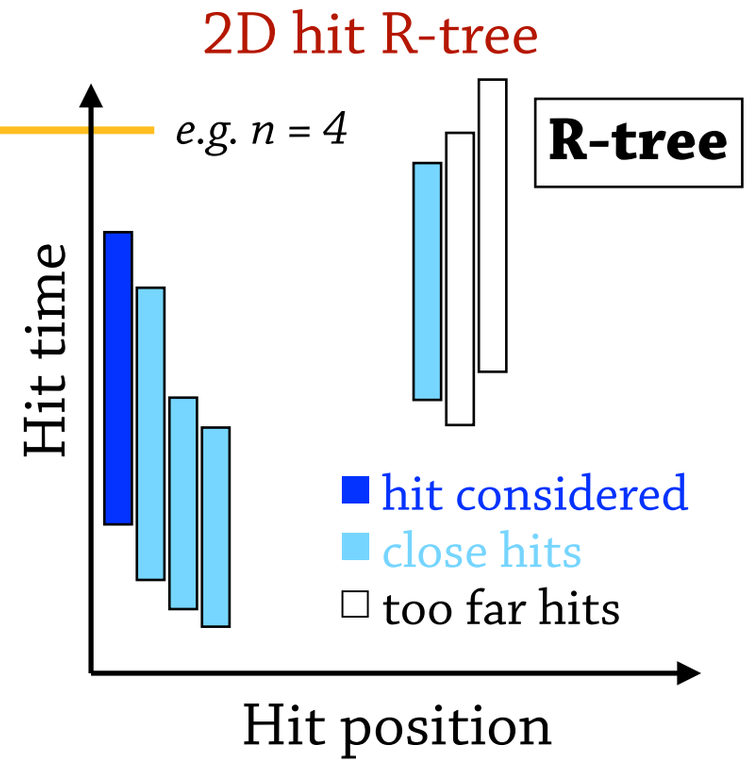
Example - after microphonic



Example - Hit found



Hit indexing and 2D tracks



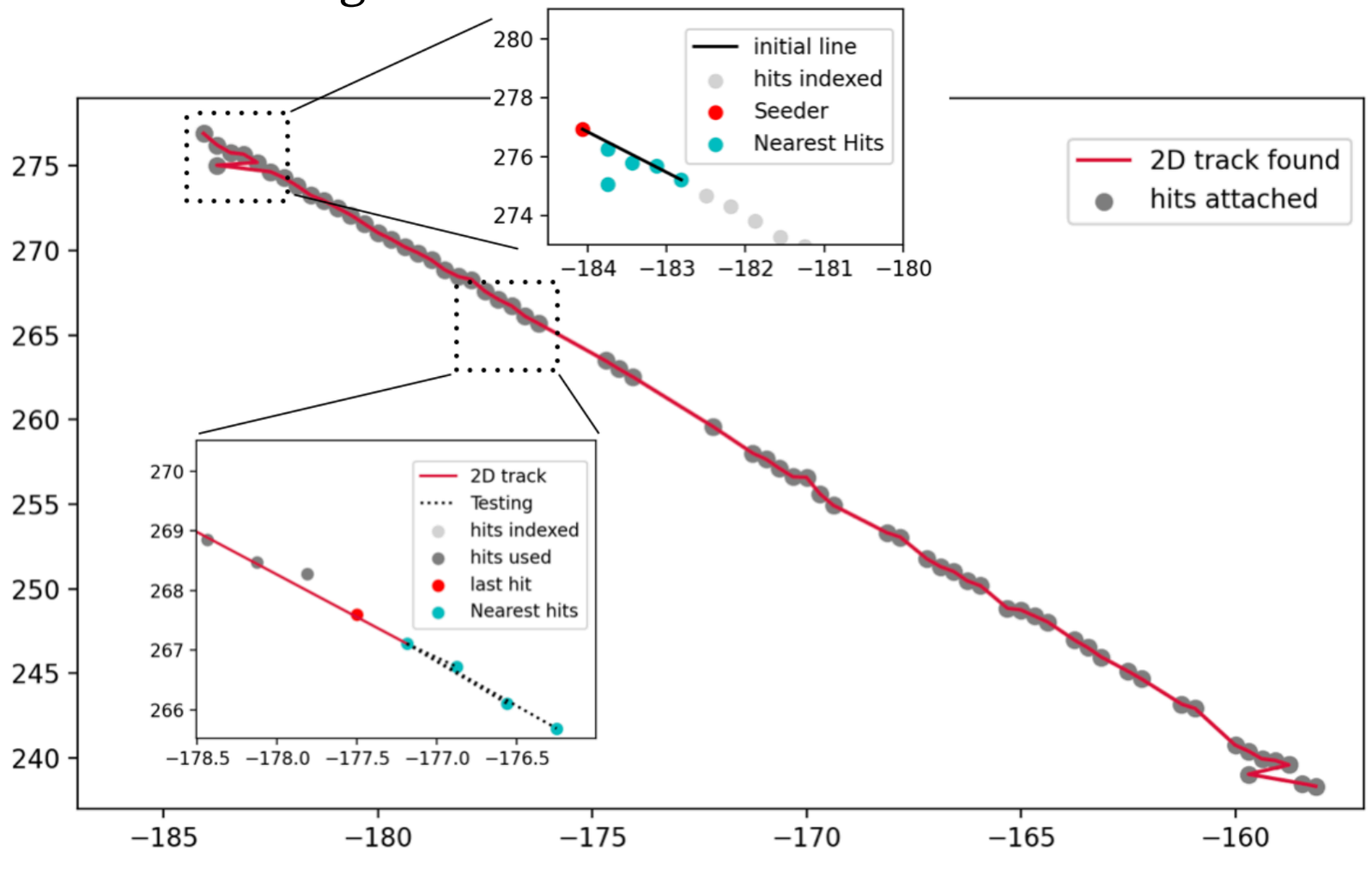
The found hits are indexed in a 2D Rectangular-tree (R-tree)

- One R-tree per view
- Index hits by their position and time

For a given element (hit), the R-tree gives the *n* nearest elements:

More details

From the collection of closed-by hits, 2D tracks are found using a Kalman-like algorithm :



-> The track is built by recursively adding neighboring hits and updating the track parameters

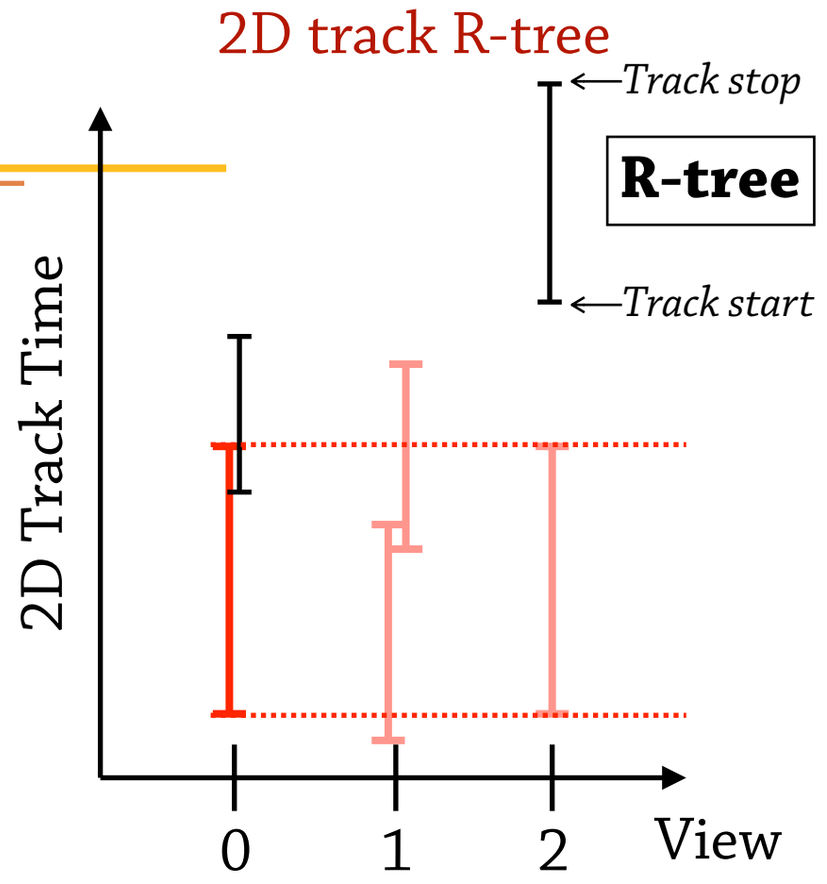
More details

-> Once built, the hits attached to the 2D track are properly re-ordered using an MST graph

More details

Pierre Billoir, « Track fitting with multiple scattering: A new method », NIM 225 (1984), 352-366.

Building 3D tracks



The found 2D tracks are indexed in a 2D R-tree:

1st Dimension : The view

2nd Dimension : The track time span

For a given track in a given view, the R-tree returns all tracks that intersects in time in the other views.

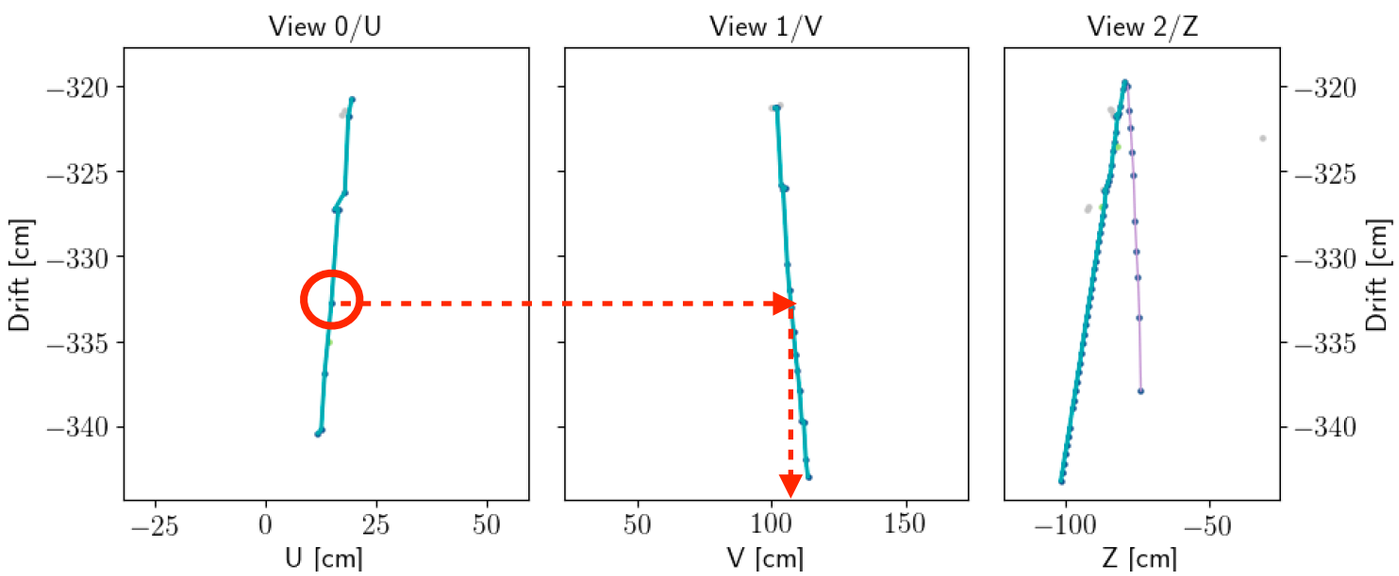
-> In case of ambiguities, the best match is chosen based on total charge balance over the overlapping time

More details

The hit coordinates of the matched 2D tracks are transformed into 3D

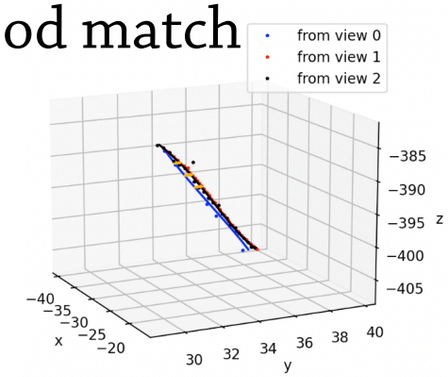
scipy

-> Compute the missing coordinates of each hits of View *i* with the interpolated position of View *j*, based on the time.

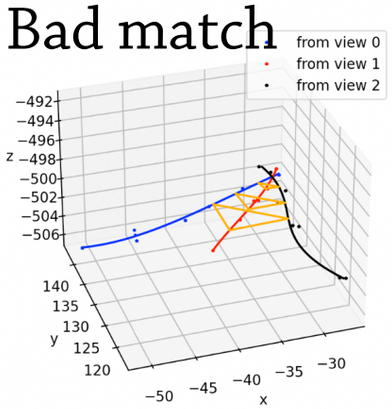


A Matching score is computed to excluded bad association in the analysis

Good match



Bad match

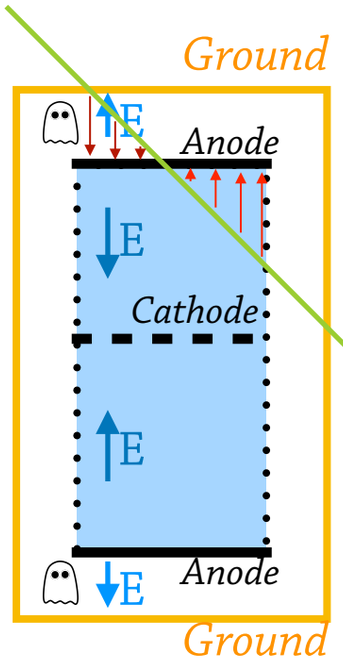


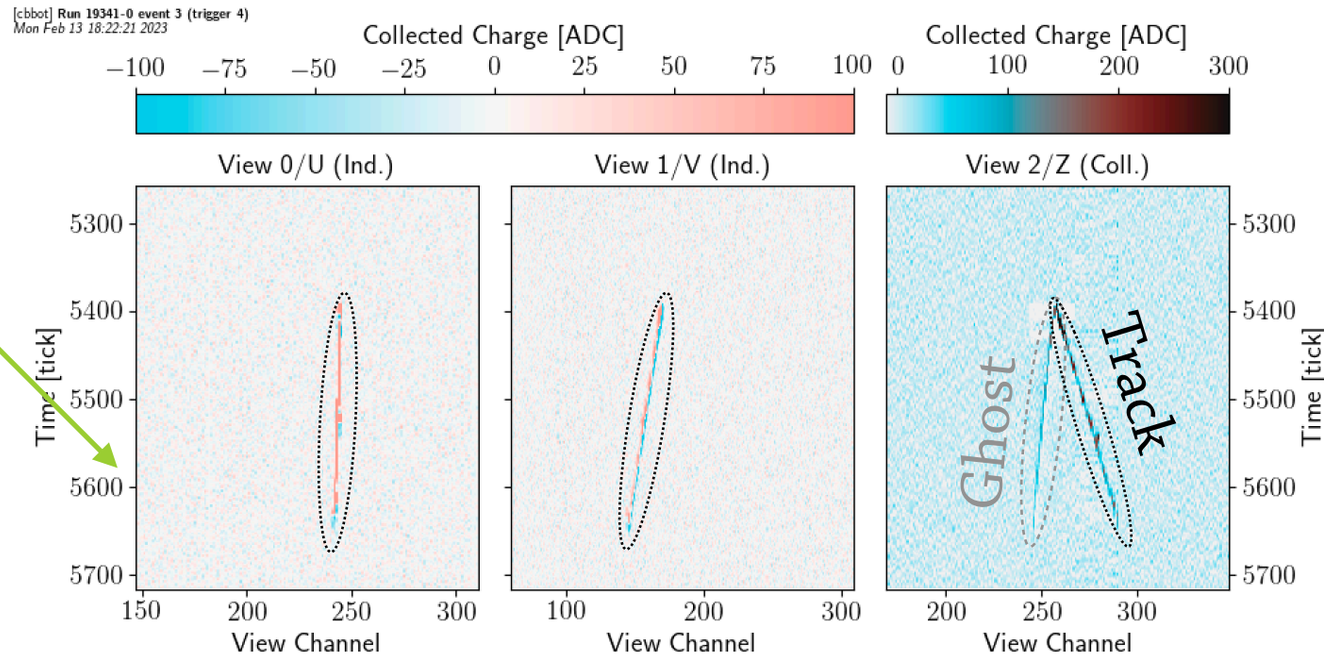
More details

Other reconstructed objects

'Ghost tracks' More details

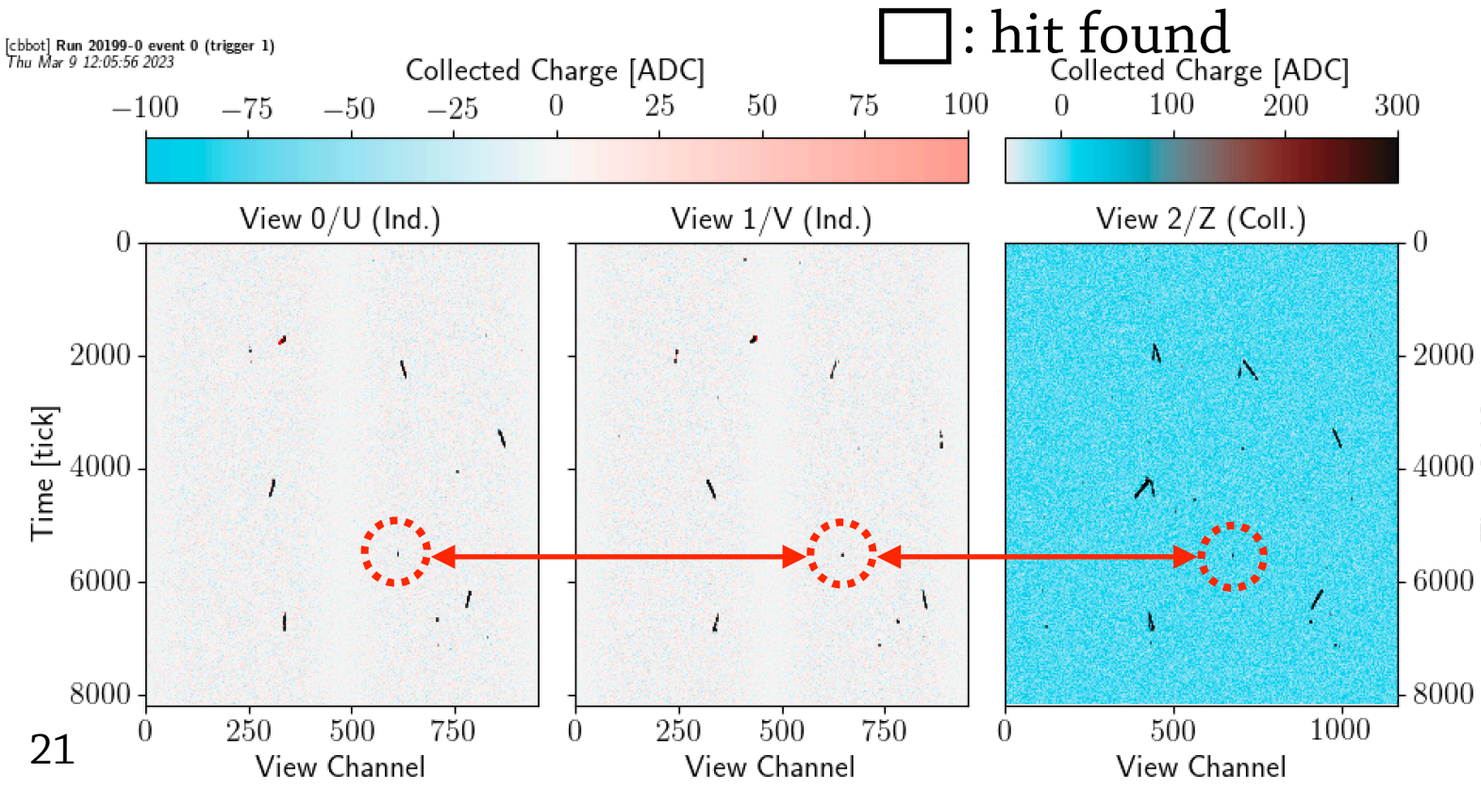
In VD design, the anode plane is powered at +1kV -> There is a 'dummy' drift field region between the anode (View 2) and its closest ground





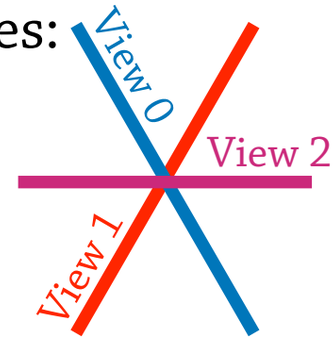
LARDON reconstruct the ghost tracks in 3D using the parameters of its associated physics 3D track

'Single depositions' More details

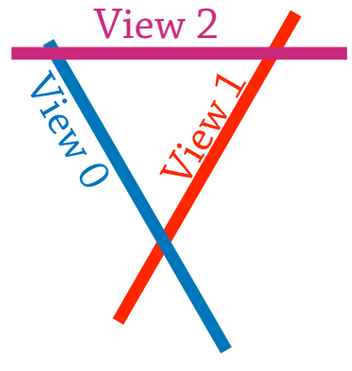


At the end of the reconstruction, test wether free hits have correspondances in the other views, if the strips overlaps and if the free hits are isolated:

Yes:



No:



Reconstruction of light signal

4 ×-ARAPUCAs installed in the ColdBbox

[Work in Progress]

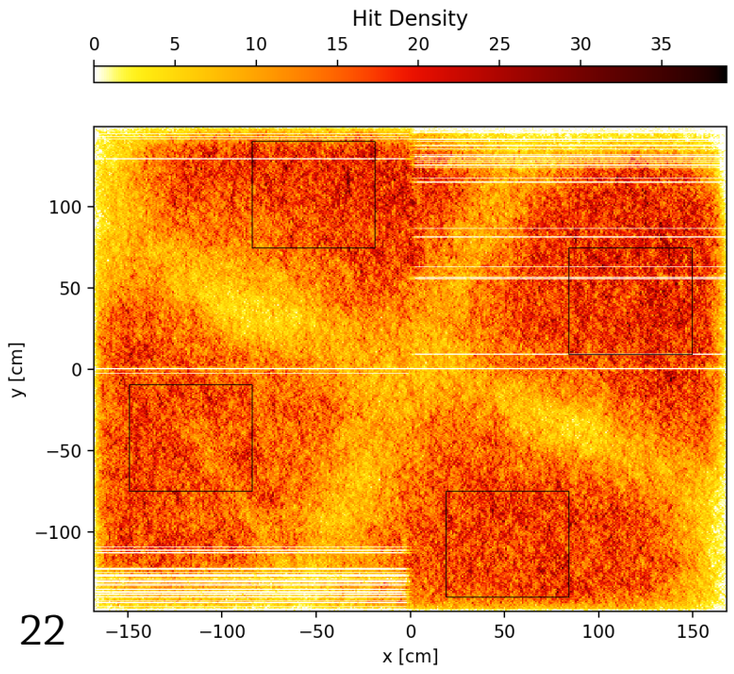
Recently, the PDS readout have been integrated in the DAQ system

-> Adapted LARDON to read both charge and light signals

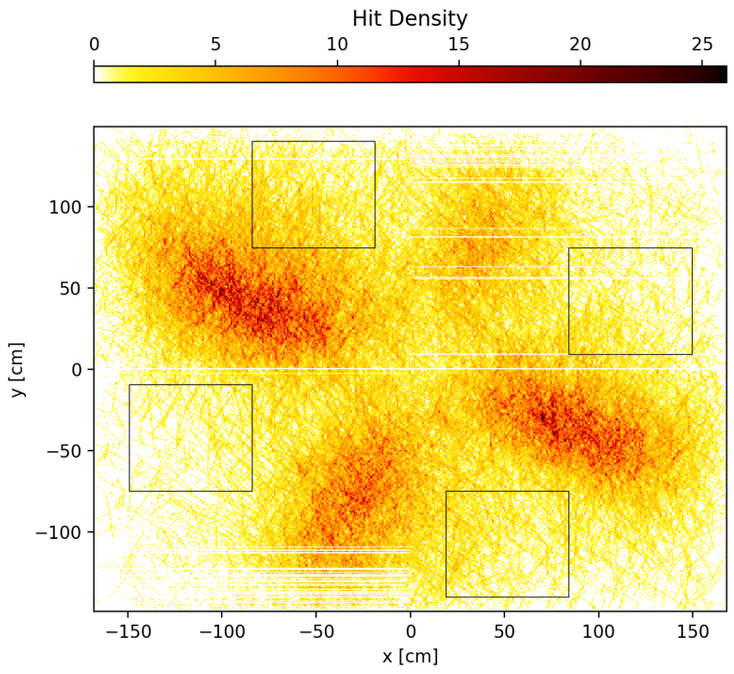
The ×-ARAPUCAs are decoded, pedestal-aligned, and light signals are found with similar algorithm as for the hit finder.

Light flashes are matched with reconstructed 3D tracks if they are compatible in time (accounting for delays between light and charge data frames), and if there are no matching ambiguities

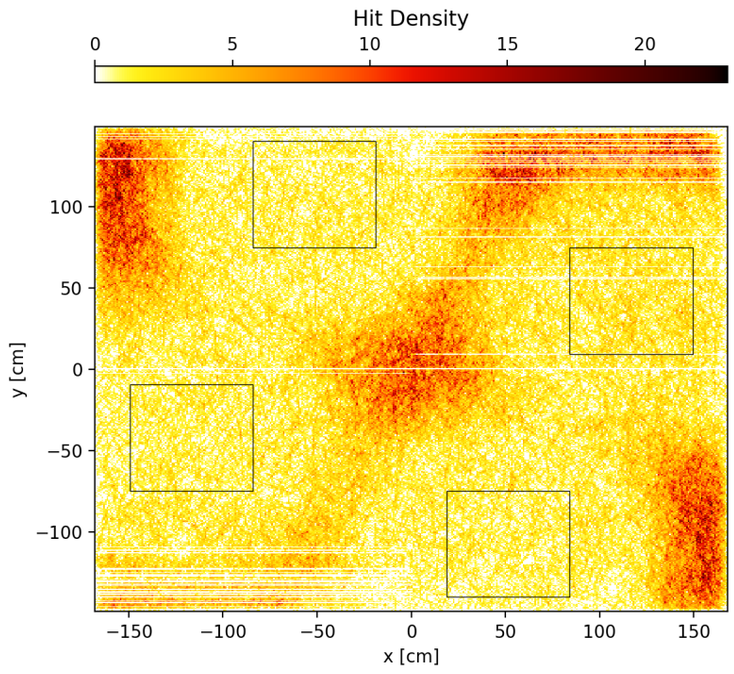
All 3D tracks matched with 1 ×-ARAPUCA



All 3D tracks matched with 2 ×-ARAPUCAs



All 3D tracks not matched with any ×-ARAPUCAs



Output File

All reconstructed objects are stored in a hdf5 file ; lardon is root-free!

pytables

For the VD-CB runs, the output file size is around **~9 MB** (versus ~1.3 GB for LArSoft)

Currently, the output file contains 11 tables and 7 arrays of variable length

General tables :

Infos stores key parameters of the reconstruction
e.g. Drift Field, Run number, File number, ...

Chmap stores the channel mapping
Map DAQ ordering to {View, Channel}

Event gives a summary of each event
e.g. number of hit/track reconstructed

Reco stores the reconstruction parameters used

Dump of one output file

```
Object Tree:
/ (RootGroup) 'Reconstruction Output'
/infos (Table(1,)) 'Infos'
/chmap (Table(1,)) 'ChanMap'
/event (Table(96,)) 'Event'
/ghost (Table(85,)) 'Ghost Tracks'
/pedestals (Table(96,)) 'Pedestals'
/hits (Table(48522,)) 'Hits'
/single_hits (Table(1012,)) 'Single Hits'
/tracks2d (Table(1888,)) 'Tracks2D'
/tracks3d (Table(560,)) 'Tracks3D'
/trk2d_v0 (VLArray(546,)) '2D Path V0 (x, z, q, ID)'
/trk2d_v1 (VLArray(520,)) '2D Path V1 (x, z, q, ID)'
/trk2d_v2 (VLArray(822,)) '2D Path V2 (x, z, q, ID)'
/trk3d_v0 (VLArray(560,)) '3D Path V0 (x, y, z, dq, ds, ID)'
/trk3d_v1 (VLArray(560,)) '3D Path V1 (x, y, z, dq, ds, ID)'
/trk3d_v2 (VLArray(560,)) '3D Path V2 (x, y, z, dq, ds, ID)'
/ghost_tracks (VLArray(85,)) '3D Path (x, y, z, dq, ds, ID)'
/reco (Group) ''
```

Reconstruction tables:

Pedestals contains the raw and filtered pedestal of each event
-> Stored as an array arranged in DAQ ordering

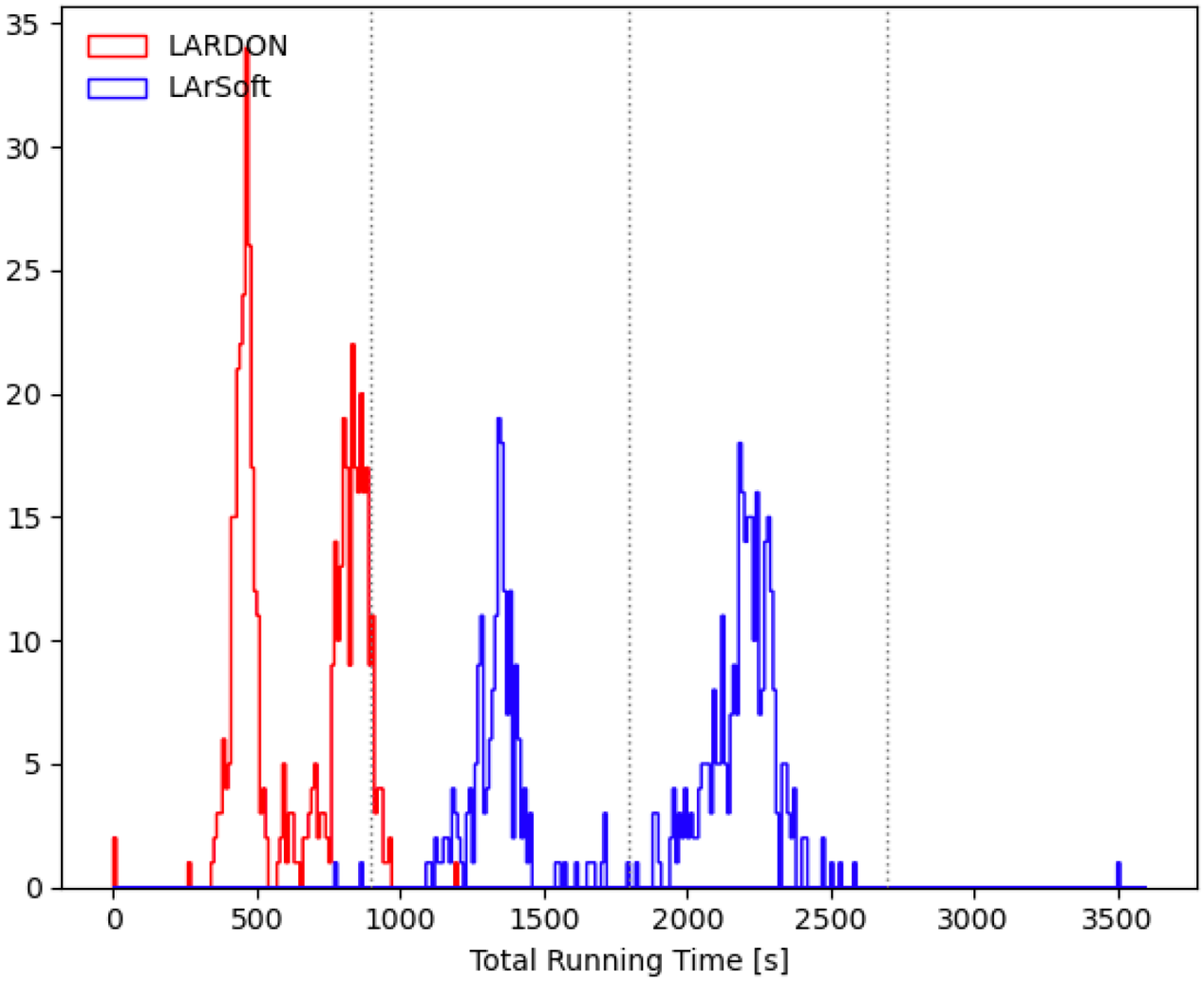
Hits, tracks2d, tracks3d, single_hits, ghost are for high-level objects
-> One entry per object
-> The table contains the general reconstructed information
E.g. position, charge, length ...

More details

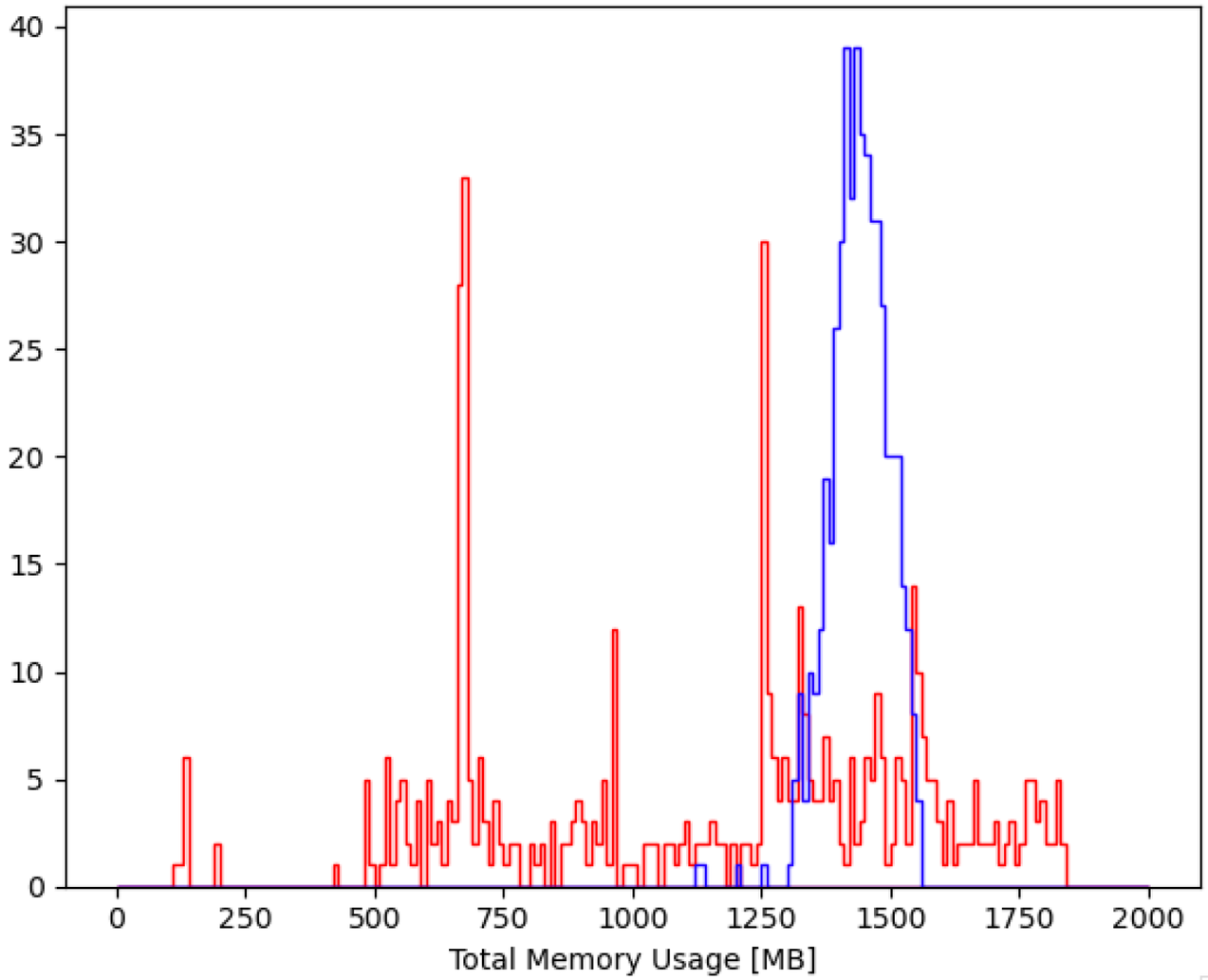
LARDON Resources

Reconstructing the same files through the same batch system (HT CONDOR on lxplus)
- ColdBox run 1727 (top CRP3) : 512 files with 60 events in each file
- LArSoft version v09_72_01d00

Time to reconstruct 1 file

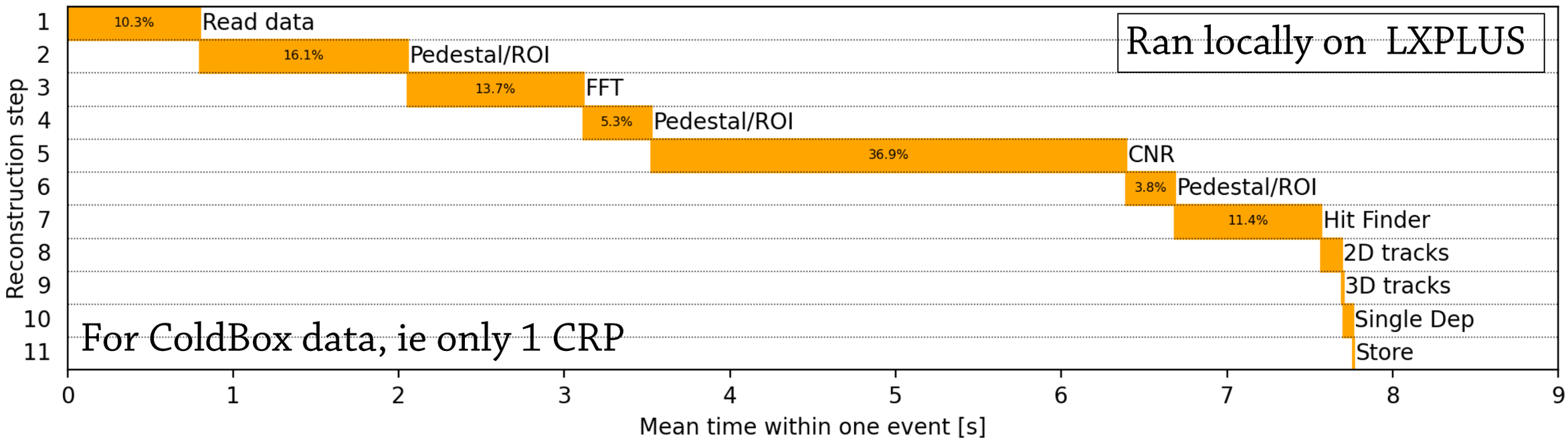


Memory usage for 1 file



LARDON Resources

About 85% of the time is dedicated to the data treatment (decoding, ROI, noise filtering)



Some ideas to speed the data treatment :

- Parallelization of noise filtering part - e.g. CRP per CRP
  -> Can be mandatory for ProtoDUNE-VD data

- Use ML to identify the signal in the raw waveforms
  -> No iterative ROI and pedestal calls needed, with potentially a better signal identification efficiency, this requires a training with a realistic signal simulation and realistic noise

- Selectively filter the noise
  - Signal represents only a small fraction of the event, no need to filter noise-only waveforms

LARDON results for the VD-Coldbox runs

Six CRPs has been tested with cosmic data in the VD-ColdBox

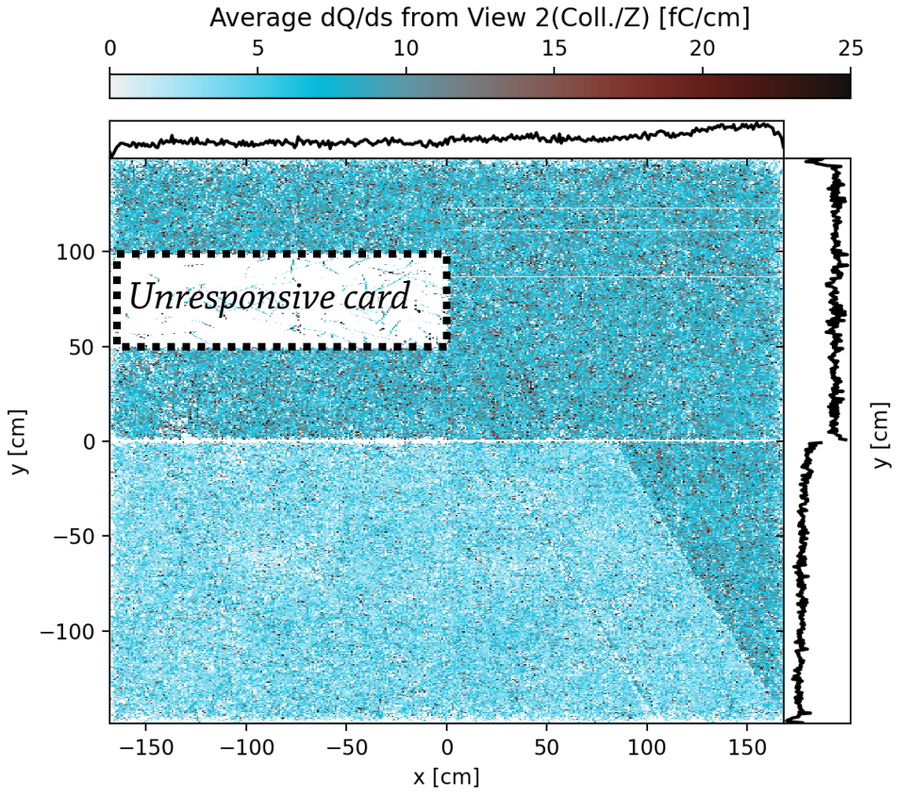
-> All runs has been reconstructed and analyzed through LARDON, ongoing efforts with LArSoft

As LARDON provides fast feedback, a few issues was found while the data was being taken:

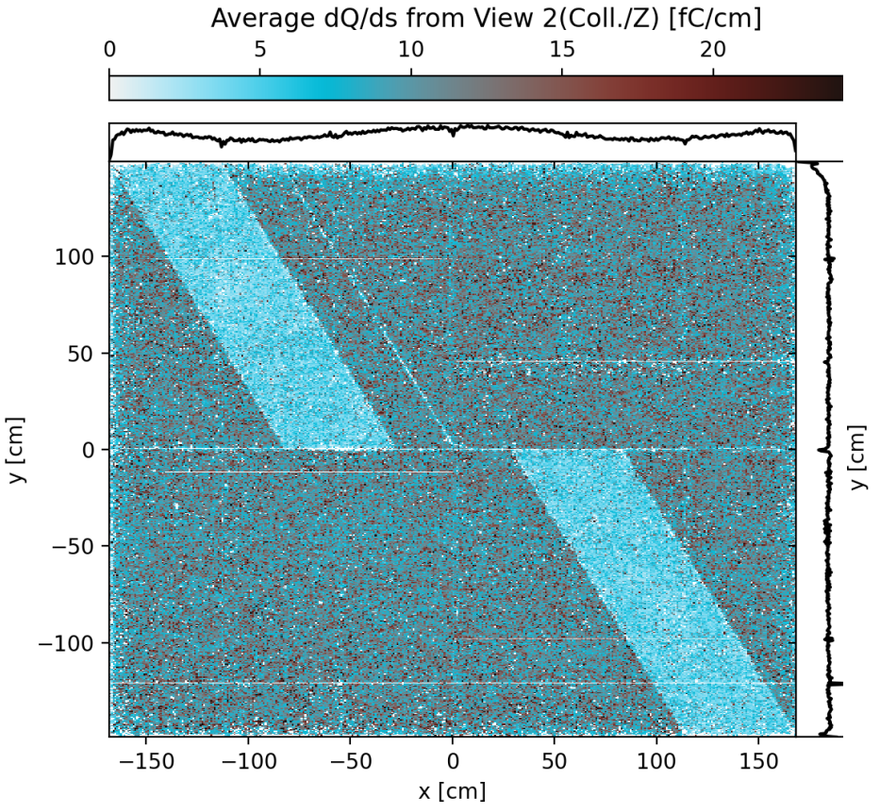
- Missing HV soldering in CRP4&5 affecting the transparency
- Holes misalignments in CRP6 reducing the overall transparency

↳ Track Reconstruction was the only way to see the problem

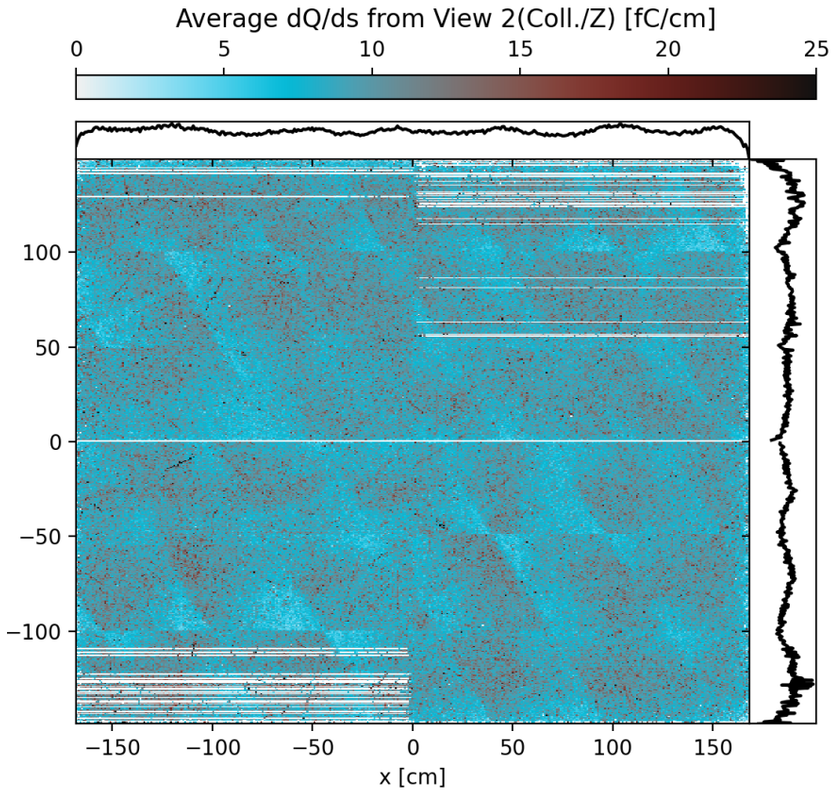
CRP #5



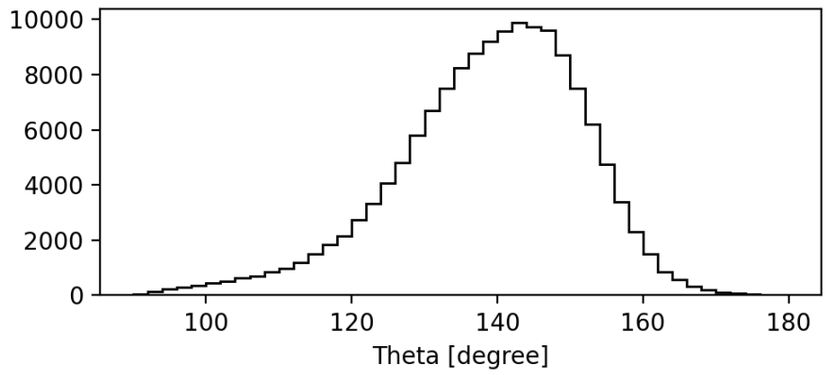
CRP #4

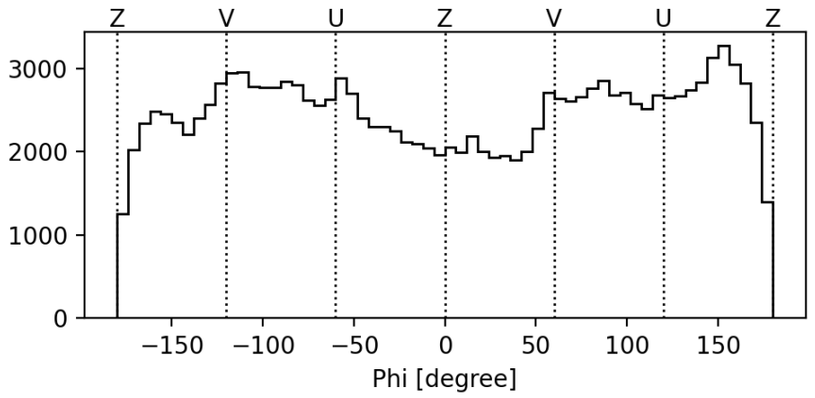


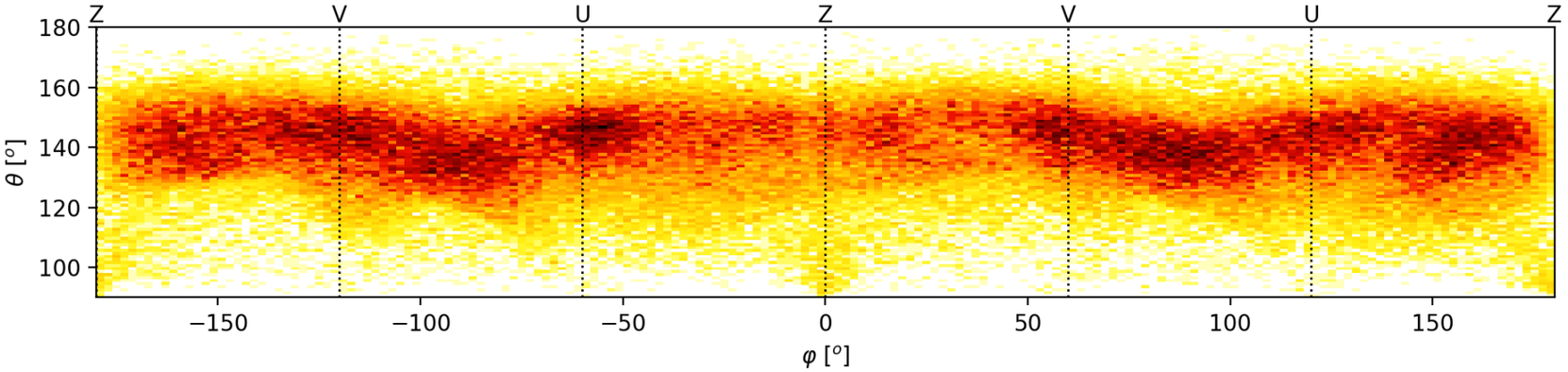
CRP #6



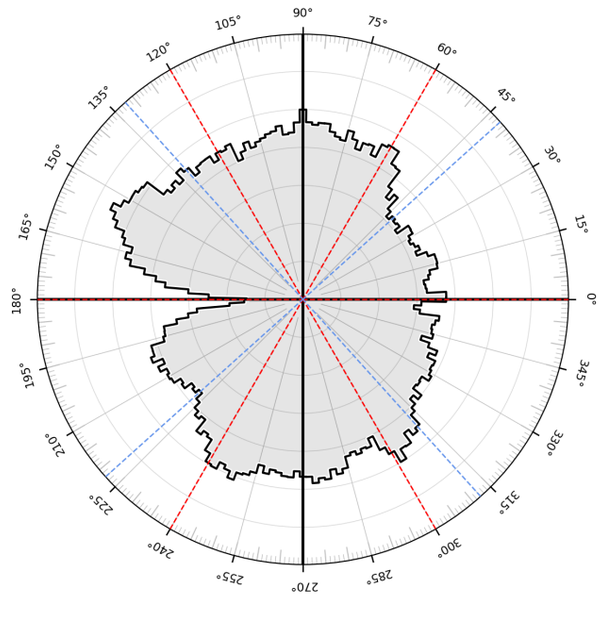
LARDON results for the VD-Coldbox runs







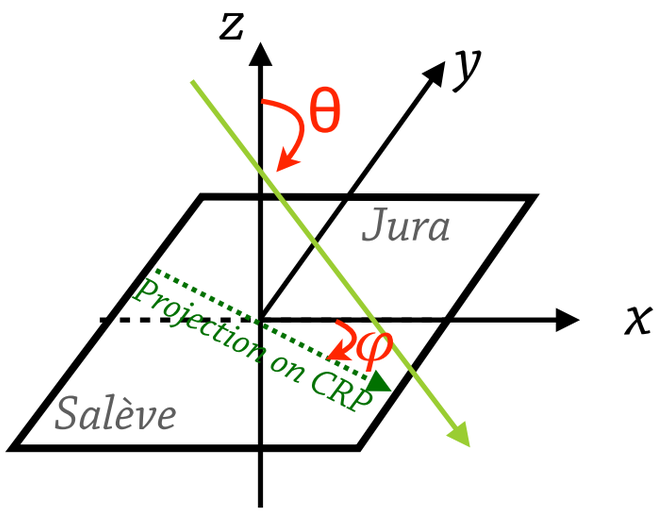
‘Antartica’ plot of reconstructed φ angle



Angle definition :

θ=90° : horizontal

θ=180° : vertical



Lardon is not fully efficient at reconstructing all tracks

- The code has never meant to be so
- Not sure the φ is supposed to be flat as the coldbox is a below ground and surrounded by blocks of concrete
- Lardon cannot run on LArSoft simulations to have a proper estimate of its reconstruction efficiency

(Cannot load the LArSoft libraries into LARDON to ‘decode’ the MC data ; HDF5 output possibility from LArSoft to be investigated)

Conclusions, Perspectives

LARDON is a light, fast, simple reconstruction code in python

- Originally developed for ProtoDUNE-DP, now adapted for the VD technology

- Has been the Data Quality Monitor tool for the VD-ColdBox data campaign
  ↳ Gave fast and reliable results on the CRP performances, for both electronics
  - Has been used to provide feedbacks on the self-trigger DAQ system test campaign VD Coldbox
  - Will be useful for the PNS system test in the coming weeks

- LARDON and LArSoft have similar results
  -> In the analyses of ProtoDUNE-DP and VD-Coldox

- LARDON is a standalone tool, and can be used as an online reconstruction for ProtoDUNE data

- LARDON can also be used to test new ideas on data!
  There is room for improvements:
  - At signal processing : noise filtering, ROI/hit finder
  - At 3D reconstruction : complicated topologies, showers, vertexing, hit->3D
  - Speed up, parallelization, GPU, memory usage, ...

The code is on GitHub at : https://github.com/dune-lardon/lardon

Mandatory Picture

Thank you for the workshop organization !

And a million thanks to Mike !

Developments needed for ProtoDUNE-VD data

- Deal with 12,288 channels in 4ms window events:
  Parallelize the four CRP treatment ?

- Handle two electronics in a single event
  The data from the two drift volume are merged in a single file, it will contain the two electronics

- Cathode at the center of the detector
  Hits and tracks can be below or above the cathode
  Adapt the track t_0 computation

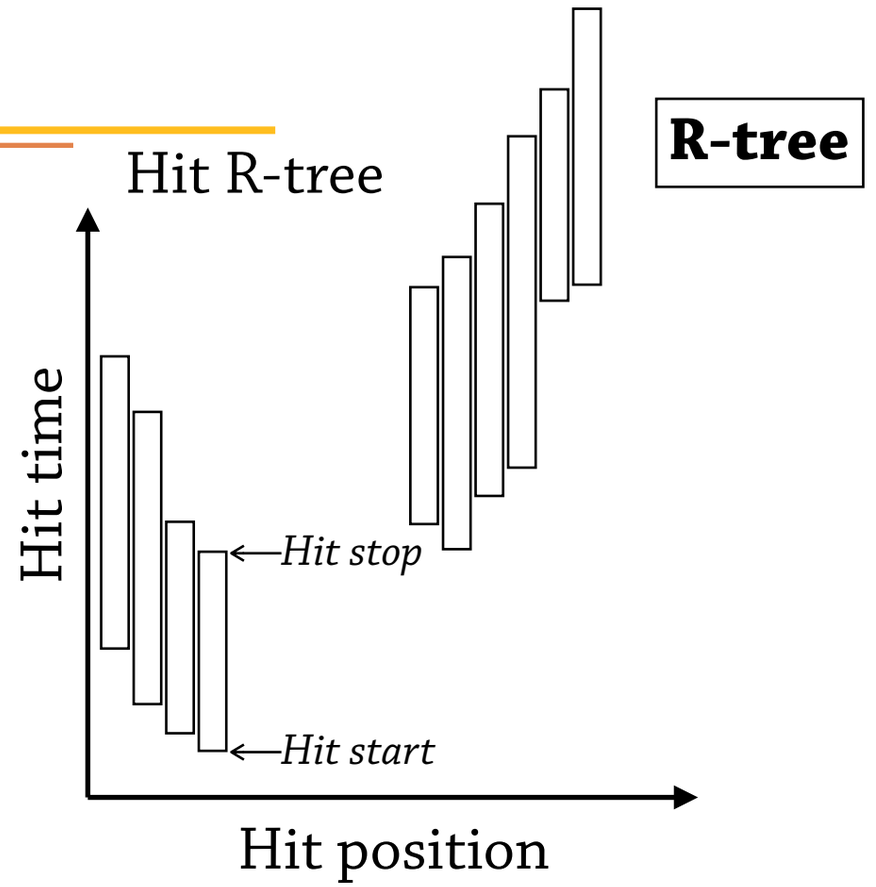
- Track stitching
  Because of the Space-Charge Effect, tracks can have a discontinuity when crossing the cathode and when crossing CRPs

- Ghost Finder
  Improve the algorithm to not be confused with michel electron topology / delta rays

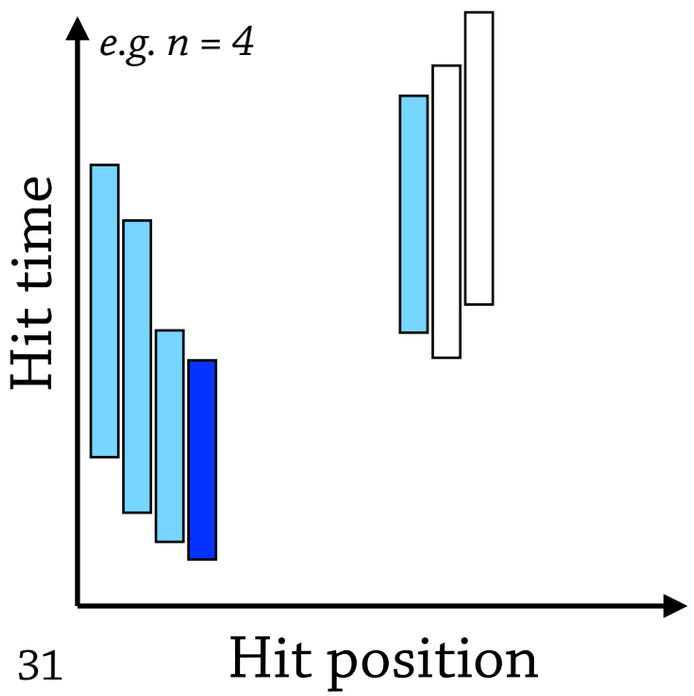
Hit indexing

For each view, all hits are :

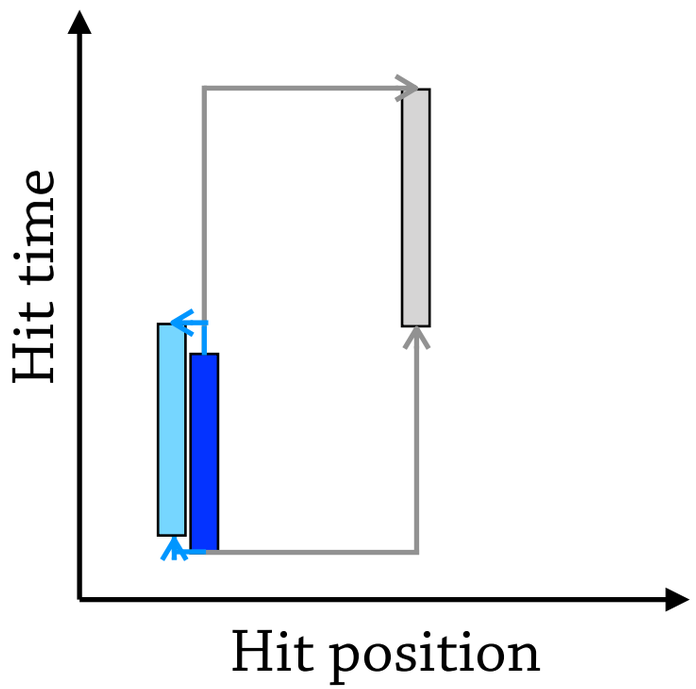
- Order by decreasing drift time and increasing channel
- Indexed in a 2D R-tree (rectangular tree):
  1st dimension : hit position (channel number)
  2nd dimension : hit time extension (start->stop)



For a given element (hit), the R-tree gives the *n* nearest elements:



The distance between two hits is the shortest path between starts and stops
-> Keep only 'close enough' hits (user-defined)

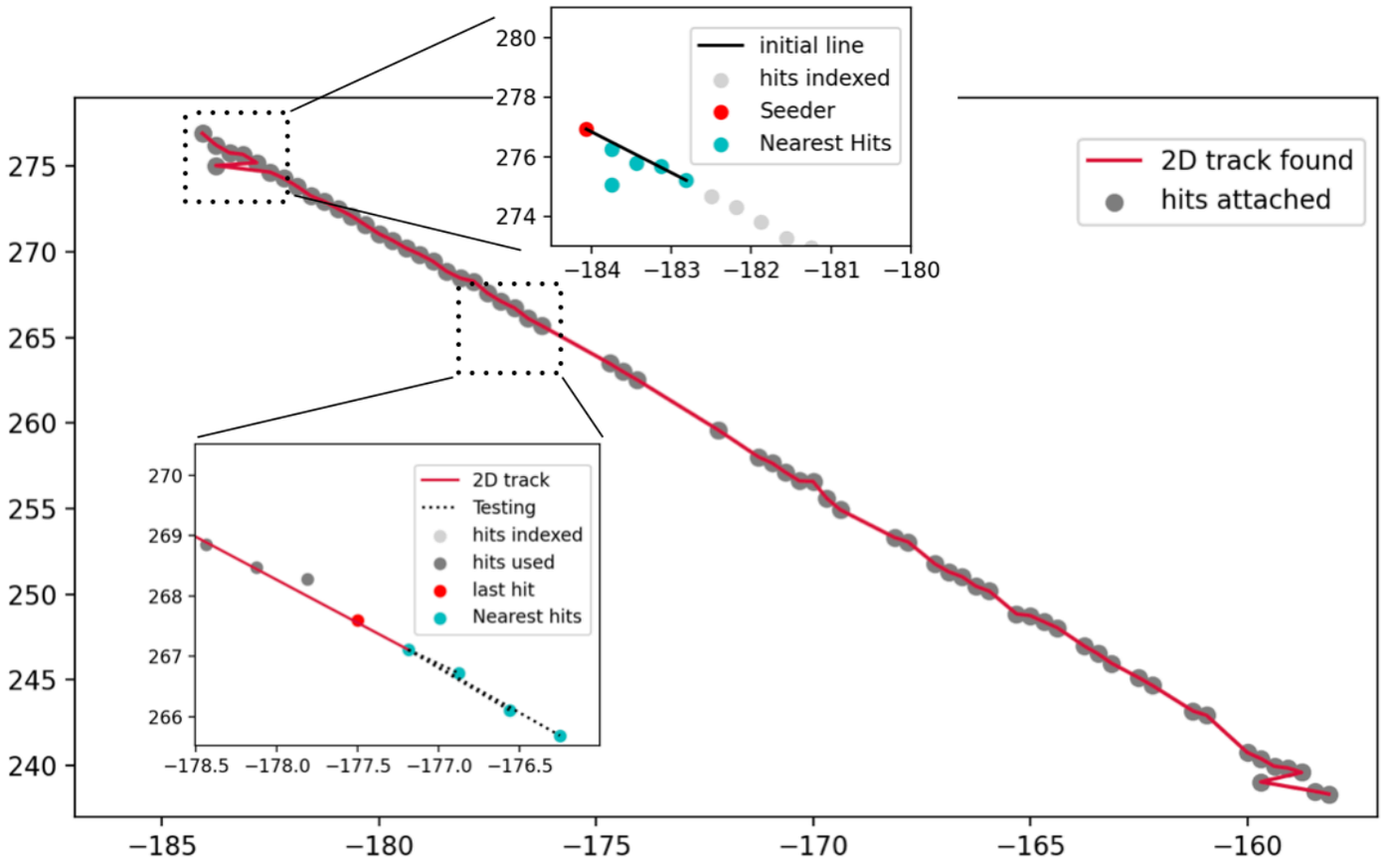


-> Once a hit is tested during the 2D track finding, it is removed from the R-tree

2D tracks : Kalman-like fit

The 2D tracks are found using a Kalman-like algorithm developed by P. Billoir

-> Same principle and mathematically equivalent to Kalman filter, but easier to understand and implement



The algorithm steps :

1. Seed : With first hit indexed + its nearest neighbors, fit a 2D line
2. Test : from last hit attached to the track test wether they belong to the track based on a X^2 evaluation :

$$\chi^2 = \frac{(y_{predicted} - y_{measured})^2}{\sigma^2_{y,data} + \sigma^2_{y,filter}}$$

3. Propagate : Update the track parameters and error matrix based on the new hits, remove the hits from the index
4. Continue until no hits can be added to the track
5. Start again, until the index is empty

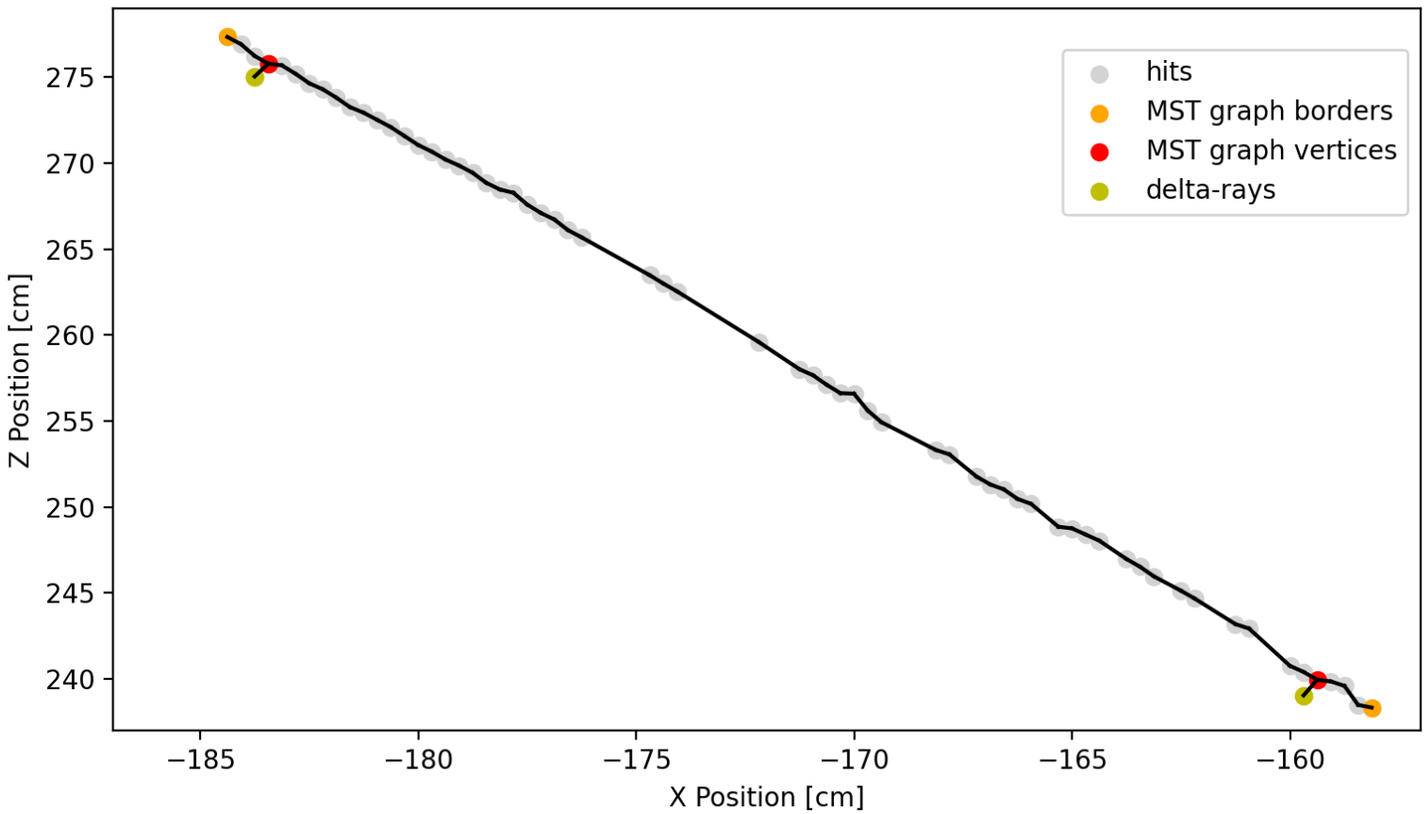
Pierre Billoir, « Track fitting with multiple scattering: A new method », NIM 225 (1984), 352-366.

2D tracks : MST graph

Graph theory can be used to ‘connect the dots’

-> A Minimum Spanning Tree minimize the weight (*distance*) between graph elements (*hits*) without cycles

scipy



The MST graph, the hit ordering within the track are re-organized.

Each element (*hit*) is connected to 1 or more other elements.

In particular, one can identify:

- The track endpoints : elements with only one connection
- Vertices from δ-ray : elements with 3 connections

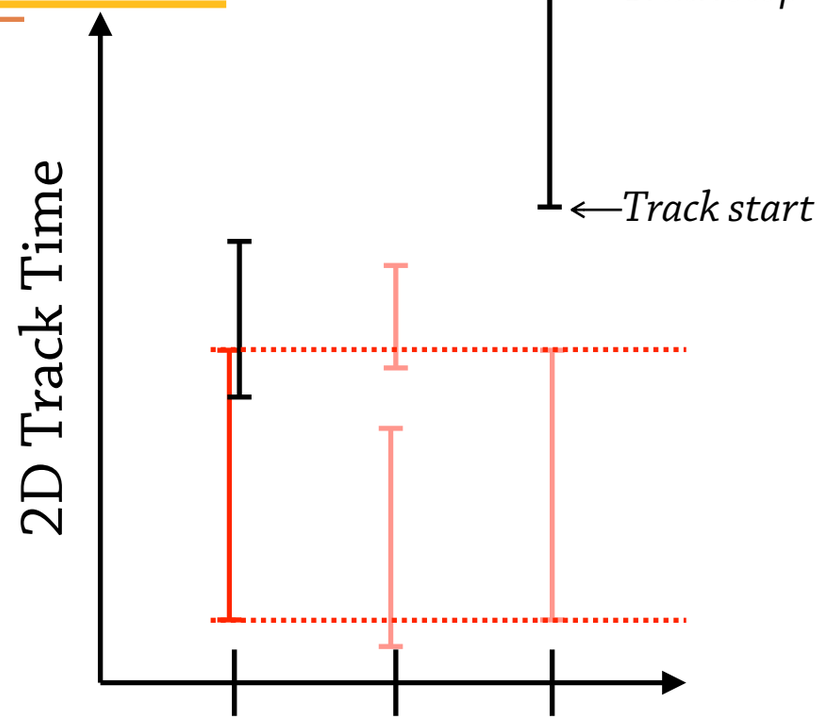
3D tracks : Match

The found 2D tracks are indexed in a new R-tree:

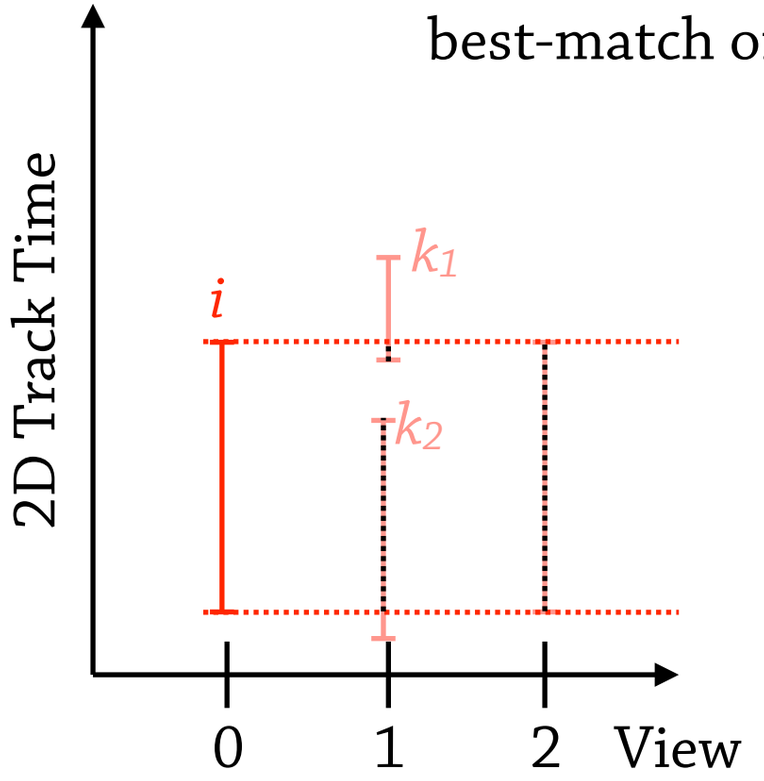
1st Dimension : The view

2nd Dimension : The track time span

For a given track in a given view, the R-tree returns all tracks that intersects in time in the other views:



When multiple overlaps occur in a view, the best-match of track *i* is determined by:



- The smallest charge balance between track *i* and track *k* :

$$\frac{Q_i - Q_k}{Q_i + Q_k}$$

Where Q is the total track charge computed in the overlapping time region

- The shortest time delay between starts and stops

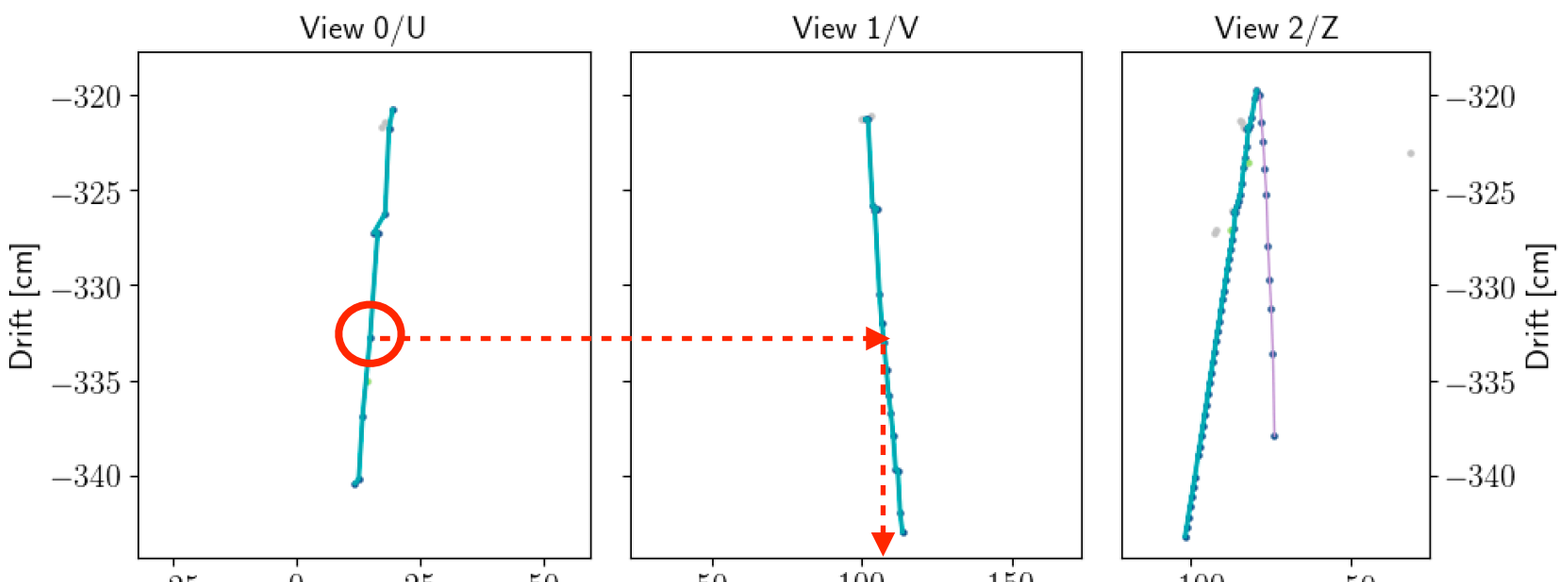
-> Each track gets one (or none) best-match in each view

-> If the best-matches are reciprocal, a 3D track can be built

3D tracks : Trajectory and Calorimetry

The hit coordinates of the matched 2D tracks are transformed into 3D

scipy



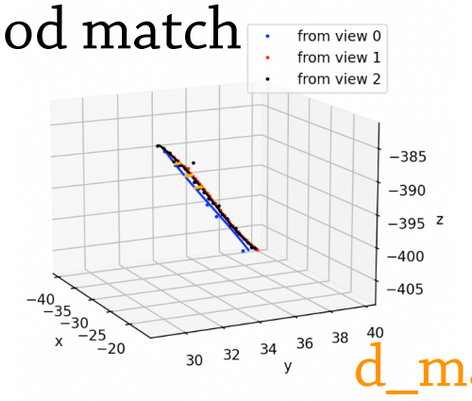
To complete the coordinates of the hits in View 0, the track of View 1 is interpolated (with the Univariate Spline)

-> Can compute the missing coordinates of each hits of View 0 with the interpolated position of View 1, based on the time. The 3D coordinates are then rotated to the orthogonal axis system

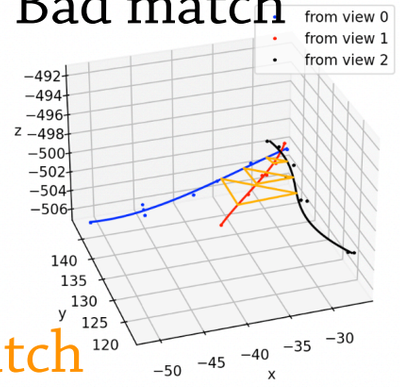
-> With the local track direction, can compute the 3D unit length (ds) of the hit

-> Repeat for the other views

Good match



Bad match



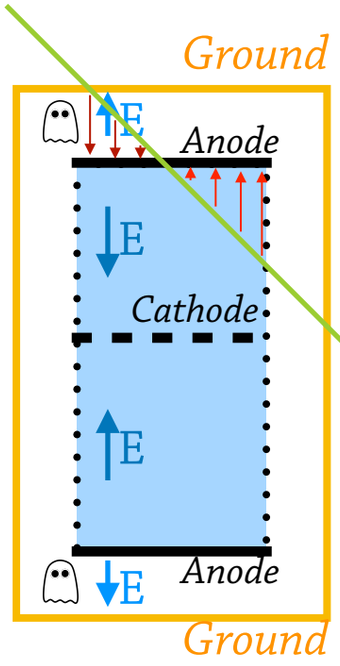
d_match

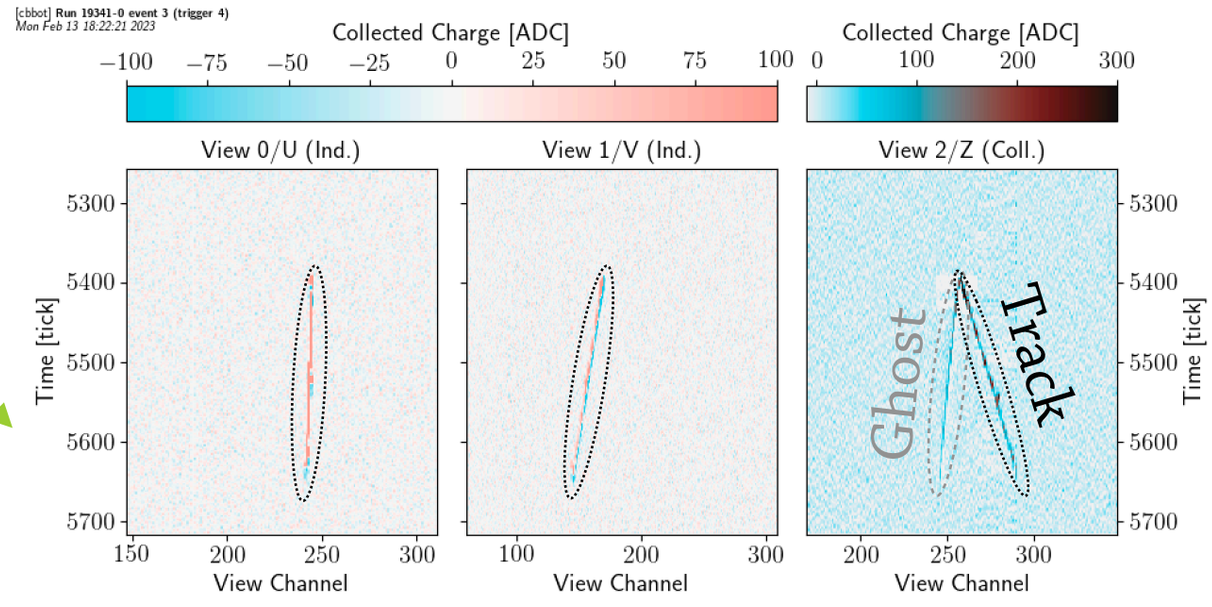
A « Matching score » d_match is computed as the average distance between the hits in the tracks at given drift position

-> Large d_match means a bad 3D matching

Ghost track finder

In VD design, the anode plane is powered at +1kV -> There is a ‘dummy’ drift field region between the anode (View 2) and its closest ground



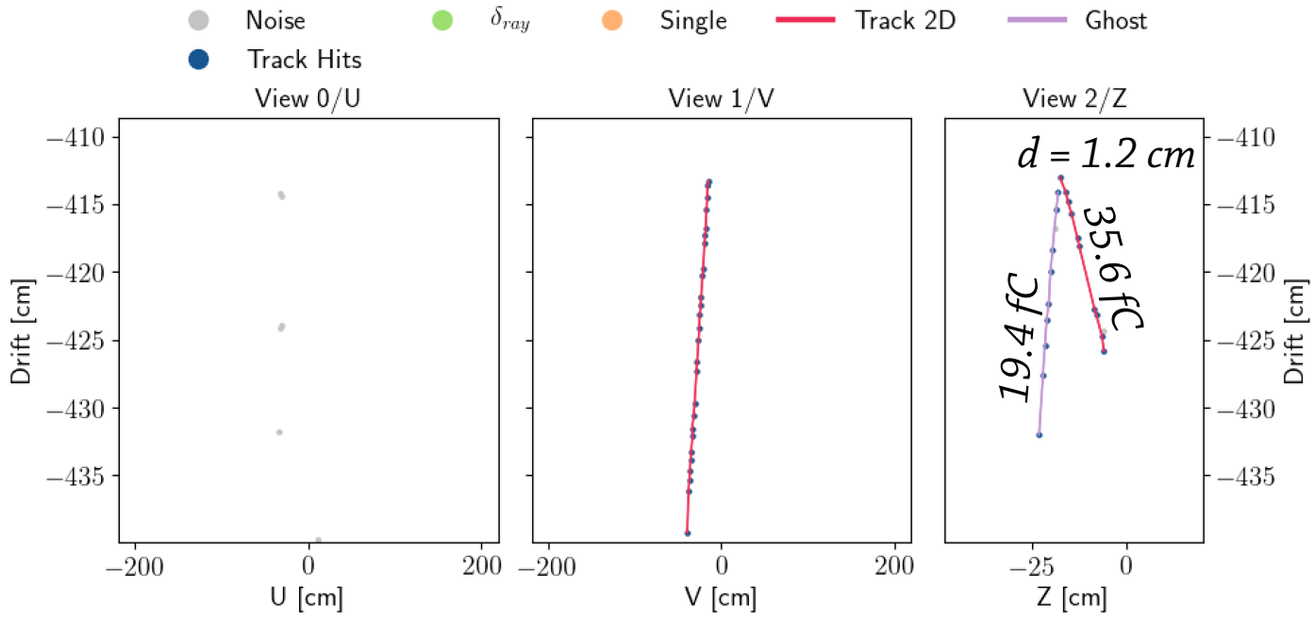


Step 1 - Ghost/Track Separation :

After the 2D reconstruction, only in the Collection View :

If two tracks have their starting point close (<5 cm) and opposite direction ; the one with the smaller total charge is considered the ghost of the other

- Cannot base the ghost separator on the track length
- Works for low multiplicity / small tracks, might need improvements for the ProtoDUNE-VD data

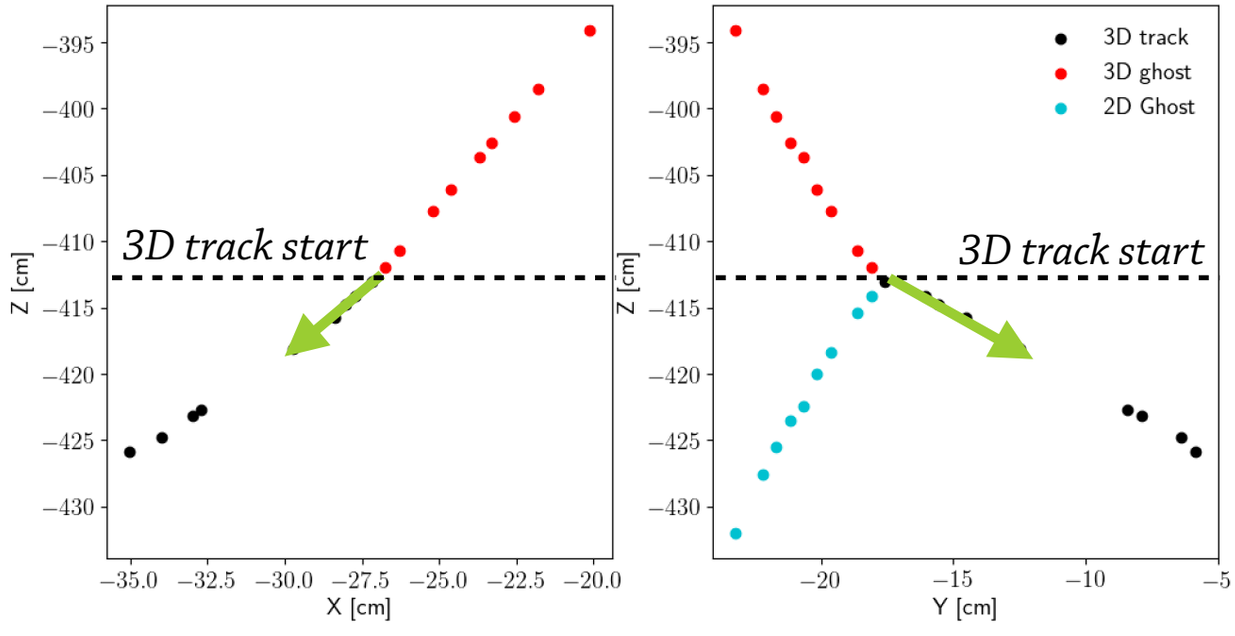


Ghost track finder

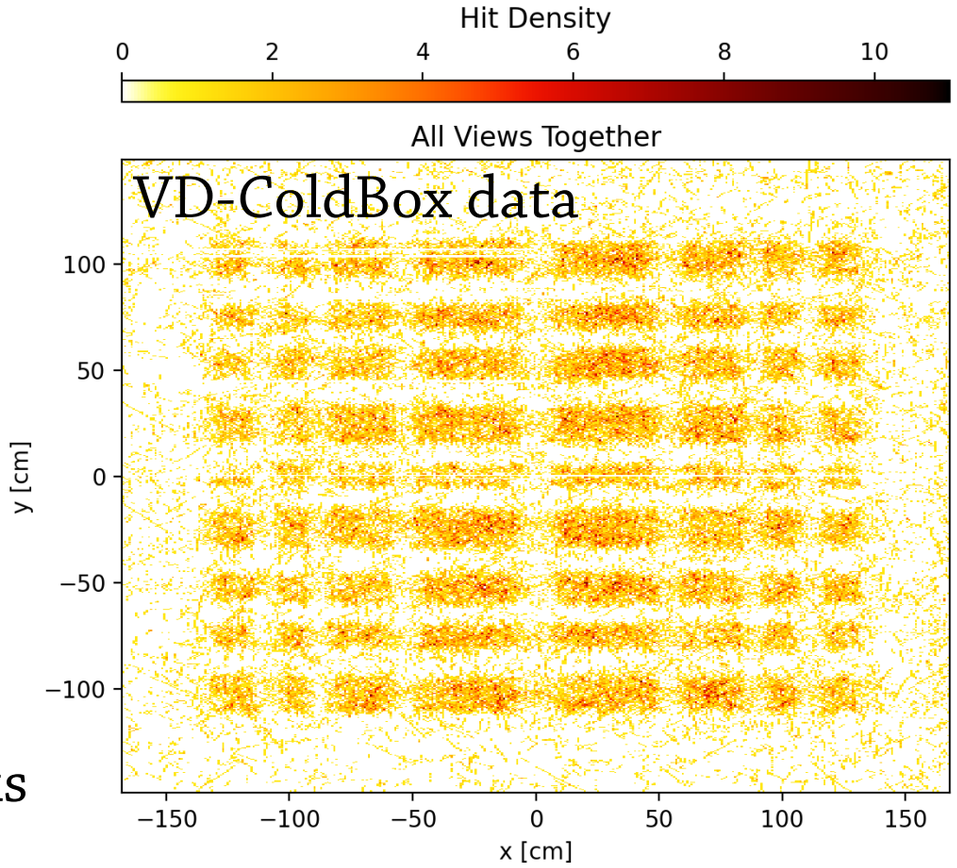
Step 2 - Build the 3D Ghost

After the 3D reconstruction, if the track associated to the ghost is used to build a 3D track:

- ***Ghost z/drift*** position is mirrored wrt to the 3D track start (« x » in LArSoft)
- ***Ghost y*** position is given by the collection strip (« z » in LArSoft)
- Use the 3D (θ, φ) direction at the anode to extrapolate the ghost direction and compute the ***Ghost x position*** (« y » in LArSoft)



Summing all reconstructed ghost allows to muon-scan the structure above the anode in the ColdBox.



In ProtoDUNE-VD and FD-VD, ghost tracks can further tag wether a track crossed the anode plane, e.g. :

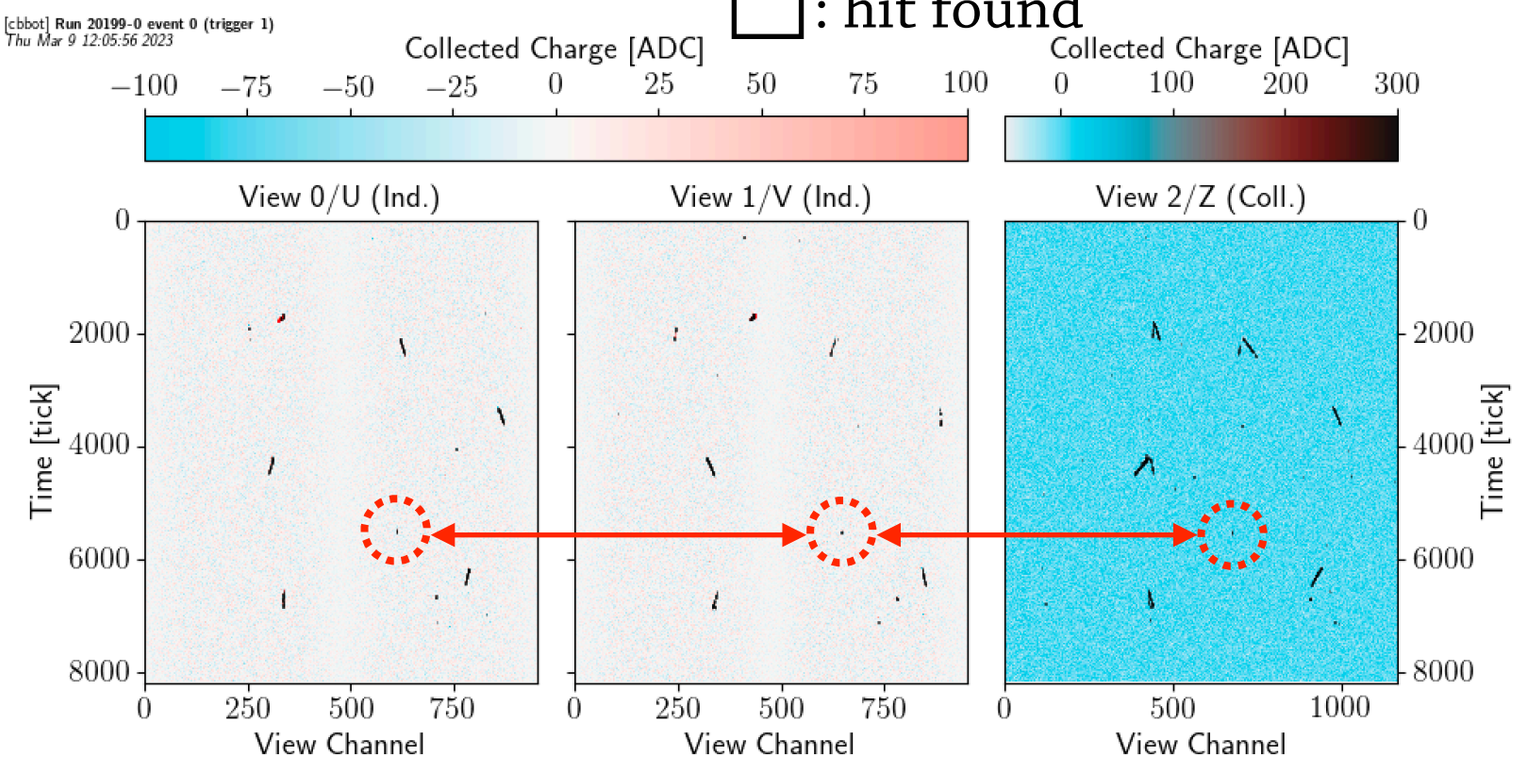
- Cross computation of track t_0
- Space Charge analysis

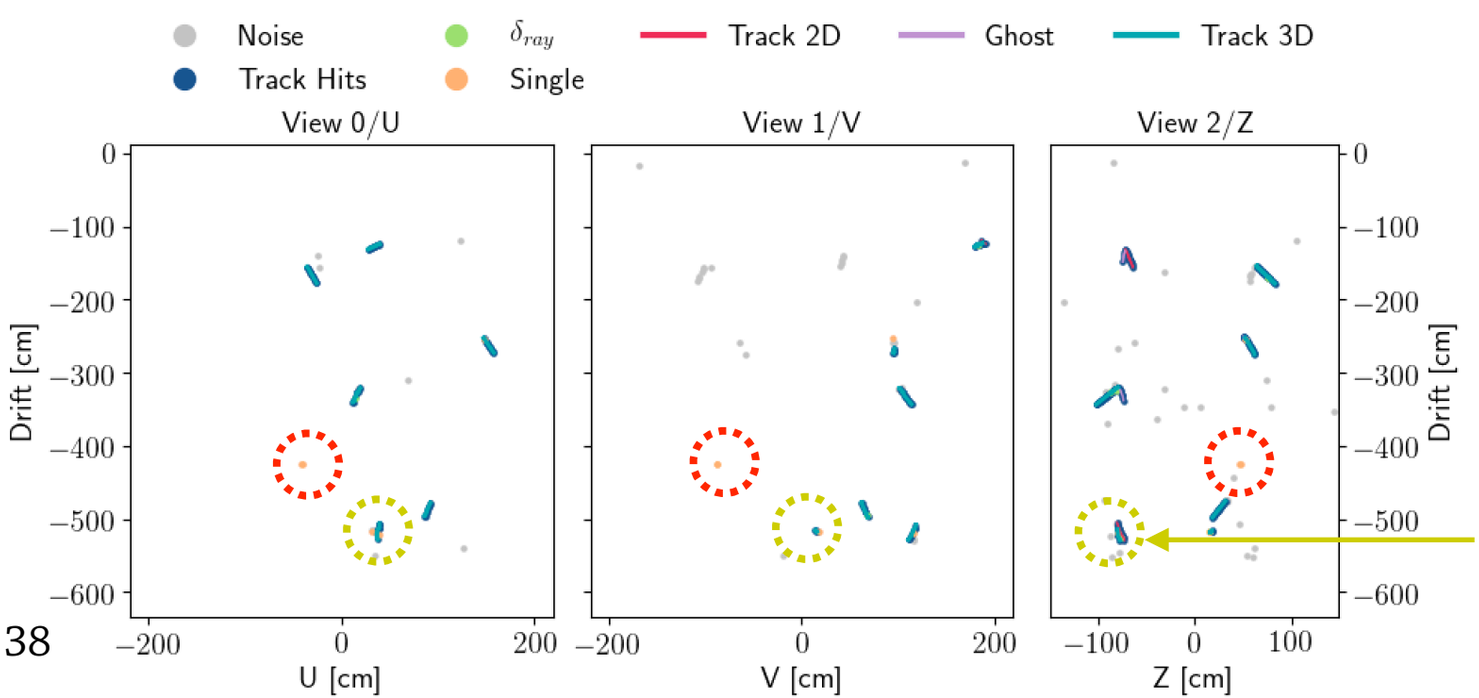
NB : LArSoft does not reconstruct yet the ghost tracks

Single deposition

Once the 3D track reconstruction is done, LARDON looks at un-matched hits and check wether free hits also exists at the same time in other views

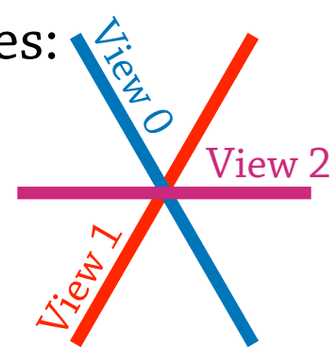
: hit found



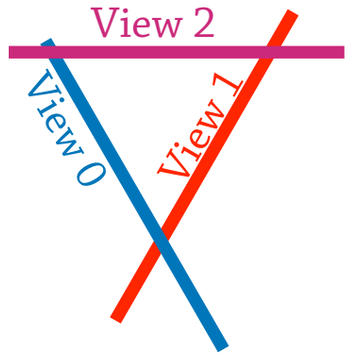


If the single depositions are
- compatible in time among the views
- compatible in space:

Yes:



No:



Then a Single Hit is formed.

LARDON computes:
- -the distance to the closest activity
- -the charge in each view, in an extended region (in space and time)

Algorithm can be improved, some ‘single hits’ identified can belong to a track

Librairies needed to run LARDON

python > 3.6
numpy -> Handles the data, ROI, FFT, ...
numba -> Speeds up some code
scipy -> For interpolation, fits, MST graph
numexpr -> Fast computation of complicated where selection
bottleneck -> To compute moving median for microphonic noise
pytables -> Handles the hdf5 format (input and output)
rtree -> R-tree library for indexing multi-dimensional information
colorcet -> Nice palette of visually distinct colorblind-friendly colormaps
matplotlib -> For the control plots
uproot -> In case of handling ROOT files (MC, output)
jsonc -> To allow comments in the json files

For the analysis of the output files, recommended extra libraries:
iminuit -> python MINUIT fitter
pylandau -> Landau distribution
fast-histogram -> Make fast 1D & 2D histogram of large samples
vitables -> Visualize the content of a HDF5 file

I use conda environments to manage the librairies

Output File

For 2D and 3D tracks, arrays of variable length also exists

-> One for each view

-> Contains the list of hits used to build the track

-> one-to-one row correspondance between the arrays

-> Array of tuples :
(x, z, charge, hit ID) for 2D tracks
(x, y, z, Q, ds, hit ID) for 3D tracks

The tables and the arrays are filled together such that entry *i* corresponds to the same object :

tracks3d Tracks3D

	chi2	d_match	event	len_path	len_straight	matched	n_hits	n_matched	phi_end
0	[0.19961573,... ,0.]	0.08189077	0	[14.264204,1... 0.]	[13.630031,1... 0.]	[True, True,False]	[16, 9, 0]	2	-148.962
1	[0.18111423,0. ,0.5699667]	0.01186099	0	[28.743776, 0. ,30.541424]	[28.559395, 0. ,30.100039]	[True,False, True]	[15, 0,30]	2	80.00791
2	[0.73312956,...	0.7015782	0	[33.860947, 8.375666,26....	[32.481766, 6.703067,26....	[True, True, True]	[14, 5,20]	3	73.81786

Example :
The third 3D track stored has 5 hits from the View 1

trk3d_v1 3D Path V1 (x, y, z, dq, ds, ID)

	0
0	[[131.96373005,-145.51711481,-122.83121619, 4.19279291, 2.15921051,...
1	[[-9.999e+03,-9.999e+03,-9.999e+03, 0.000e+00, 1.000e+00,-1.000e+00],...
2	[[145.0308616 , 59.18582095,-266.92675671, 6.74958705, 1.7718642 ,...
3	[[7.02728088e+01,-8.10089248e+01,-3.2128756...
4	[[89.78797875, 30.8223411 ,-477.12189181, 4.19398242, 0.98259128,...
5	[[89.35574214, -81.61631466,-508.2414864 , 3.90665424, 0.98113675,...
6	[[27.20364116, 18.81308863,-515.09567559, 6.41732528, 5.23016686,...
7	[[-9.999e+03,-9.999e+03,-9.999e+03, 0.000e+00, 1.000e+00,-1.000e+00],...
8	[[54.62566423,147.39942585,-49.64108107, 4.41286271, 1.92930676,...

trk3d_v1: (2,0)

	0	1	2	
0	145.0308616	59.18582095	-266.92675671	6.749
1	144.54722237	59.87813325	-268.58121617	1.184
2	146.07349794	60.99172007	-271.41743239	5.121
3	144.60515273	61.50847157	-272.8355405	0.413
4	145.55248855	61.61930534	-273.15067563	2.178

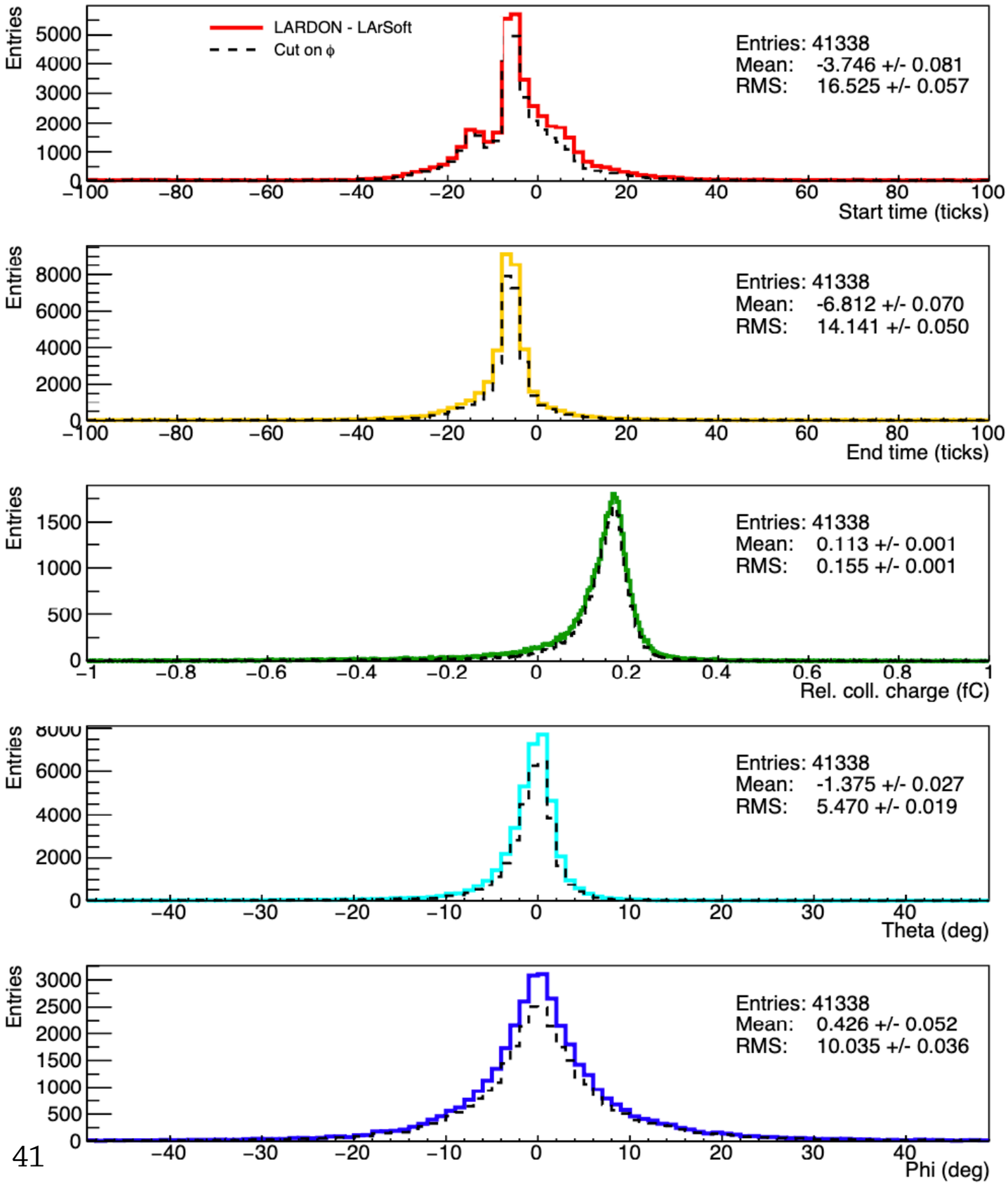
NB : When a track has no hits in a given view, the array is filled with two fake hits (at -9999, -9999, -9999)

-> In the analysis, the 1st and last hits of a track should not be considered anyway

In https://github.com/dune-lardon/cookbook I gave some analysis examples with LARDON's files

-> I plan to add more examples

Comparison with LArSoft



Comparison of LARDON and LArSoft reconstruction on the VD-ColdBox data

-> Difference between LARDON and LArSoft for various reconstruction parameters

Small deviations seen for the reconstructed charge and track times, probably due to the fact that LARDON does not deconvolve the hits