pySiDR: Python Event Reconstruction for SiD

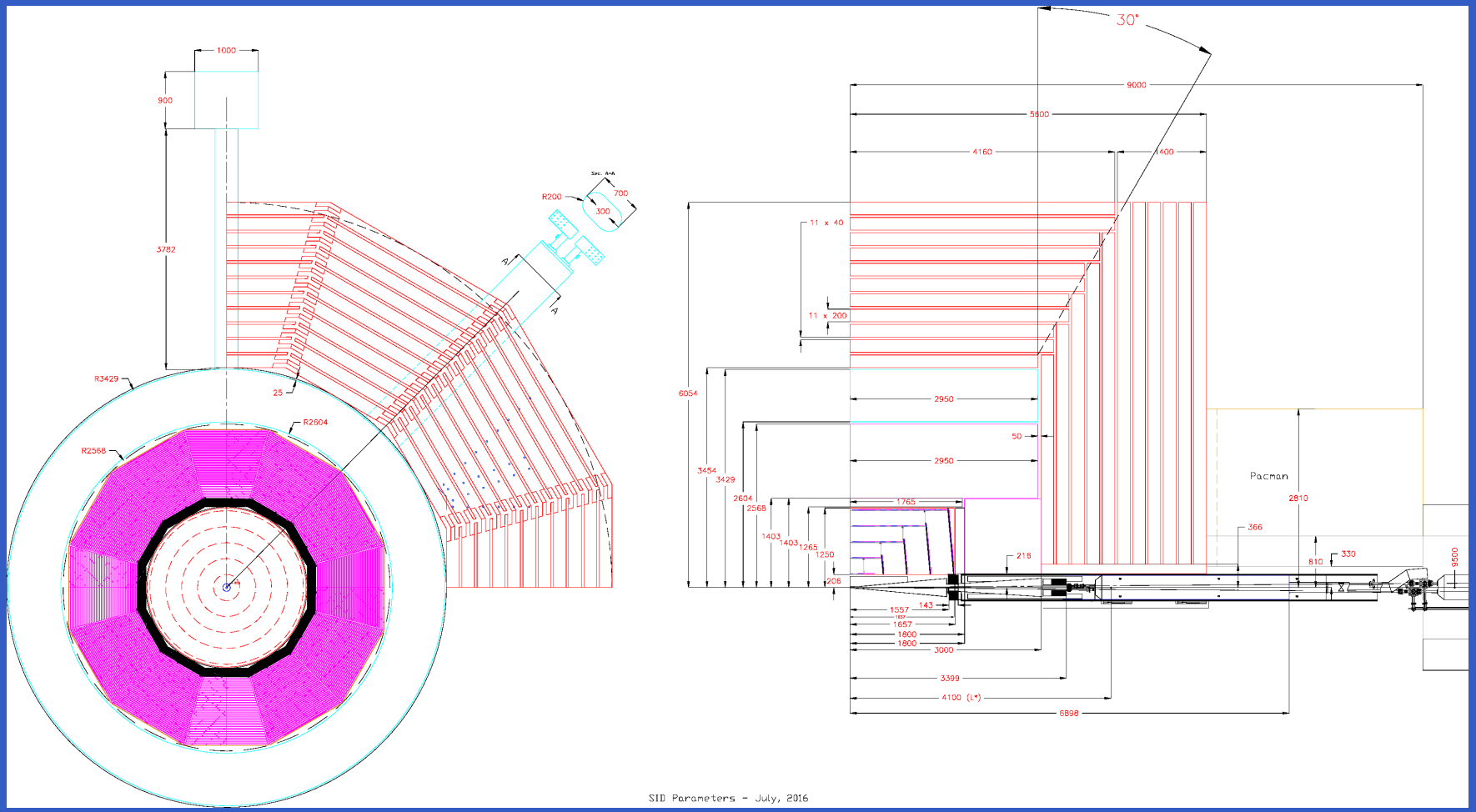
Chris Potter

University of Oregon

Introduction

- Elementary Observations of a Nonexpert
  - Event reconstruction in ILC/CLIC has relied on algorithms implemented in C++.
  - Python package pyLCIO allows complete access to tracker/calorimeter hits in LCIO files.
- Drawbacks of ILCsoft C++ Reconstruction
  - Code is compiled, challenging to develop on CVMFS.
  - Code can be sprawling over many thousands of lines in many files.
  - Code is high level C++, inaccessible to the nonexpert.
- Benefits of Python Reconstruction
  - Code is interpreted, giving immediate feedback to developers and users.
  - Code is minimal, just a few hundred lines of code, centralized in a few files.
  - Code is self documenting, easily understood by a wide range of users.
  - Timing penalty is minimal, since bytecode is compiled (to be established rigorously).
  - Interface with Tensorflow, Scikit-Learn and other machine learning tools is smooth.
- In this talk I discuss pySiDR, a Python package for SiD event reconstruction.

Nominal (TDR) SiD Design



VxD/Tracker: $\frac{\Delta p_T}{p_T^2} = 0.002 \oplus \frac{0.00002}{\sin\theta}$

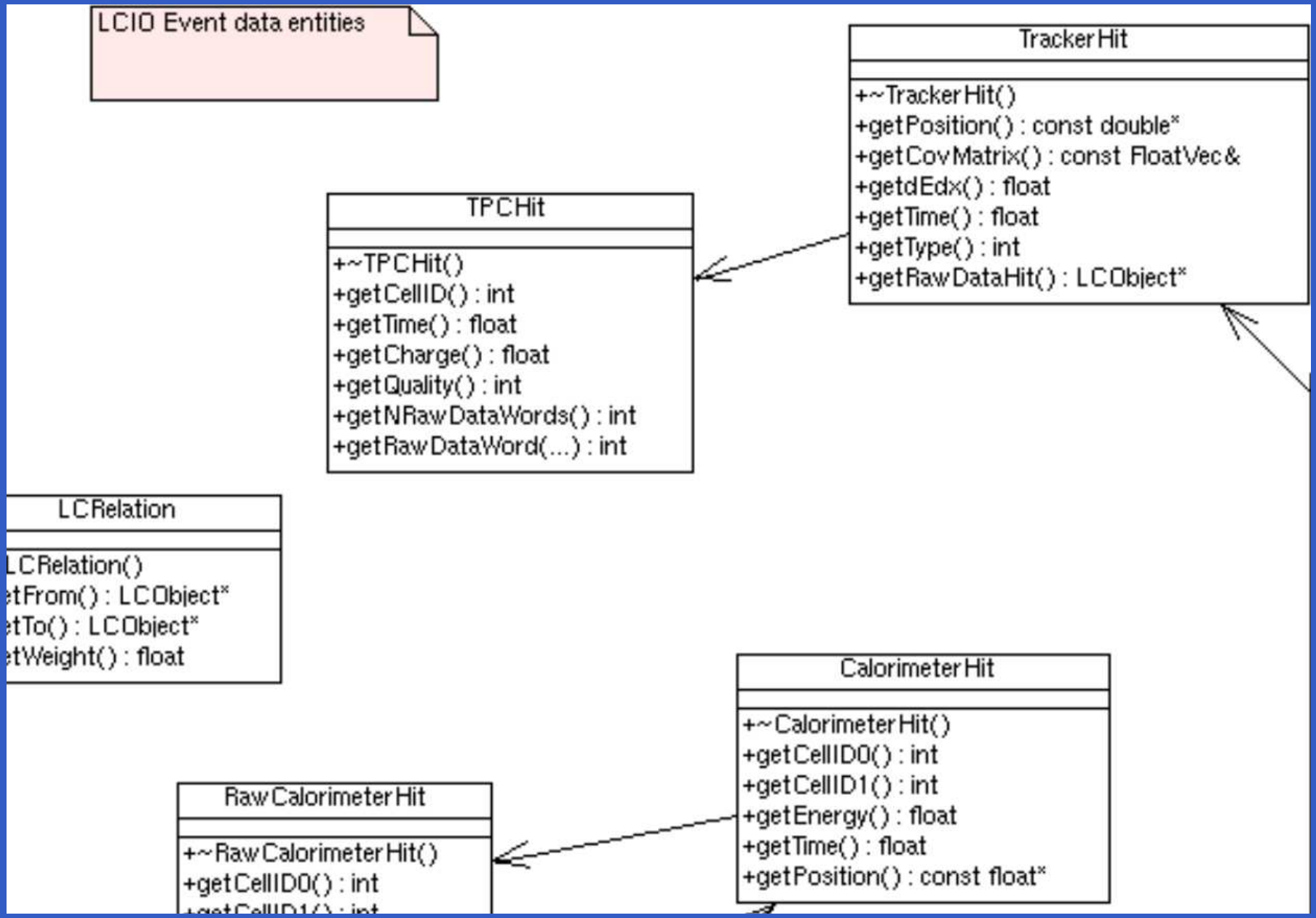
ECal: $\frac{\Delta E}{E} = 0.01 \oplus \frac{0.17}{\sqrt{E}}$

HCal: $\frac{\Delta E}{E} = 0.094 \oplus \frac{0.56}{\sqrt{E}}$

Technical Details

- Single Particle Event Generation
  - Single particles with lcgeo/example/lcio_particle_gun.py (ILCSoft v02-00-02)
  - Flat distribution of single $e, \gamma, \pi^{\pm}, n, \mu$ with $1 < E < 125$ GeV
- Higgstrahlung Event Generation
  - Staged ILC250: $\sqrt{s} = 250$ GeV
  - Whizard 2.6.3 including ISR, beamstrahlung, polarization (80/30).
  - Beamstrahlung with Guineapig 1.4.4 using staged ILC250 beam parameters.
- SiD Simulation
  - Full simulation with ILCSoft v02-00-02
  - Compact SiD_o2_v03 detector description
- Python package pySiDR in SiDPerformance (GitHub)
  - Python 2.7.10, which ships with ILCsoft v02-00-02
  - Contents, compiled into bytecode *.pyc
    - pySiDR.py: track fitting , track and cluster finding, particle flow
    - utilities.py: often used mathematical functions, linear regression, circular regression
    - sid_o2_v3.py: SiD detector geometry file
  - Geometry file included for SiD, but any other detector geometry file is easily adapted.

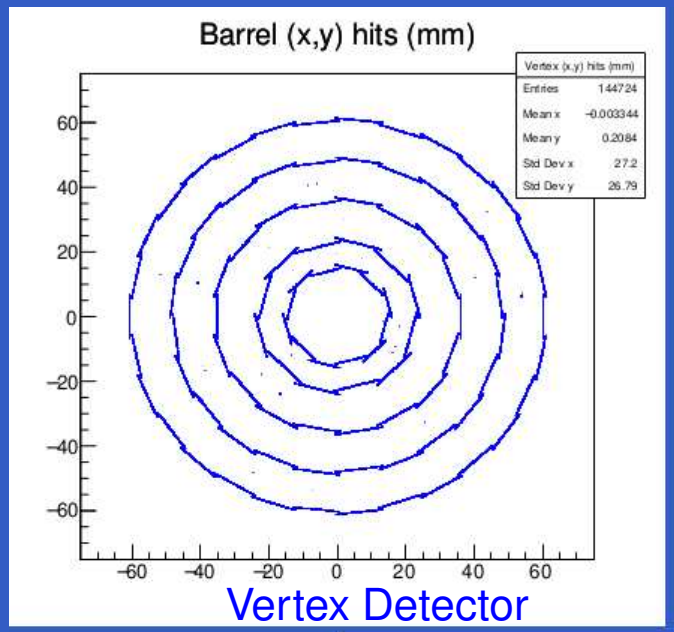
Tracker and Calorimeter Hits in LCIO

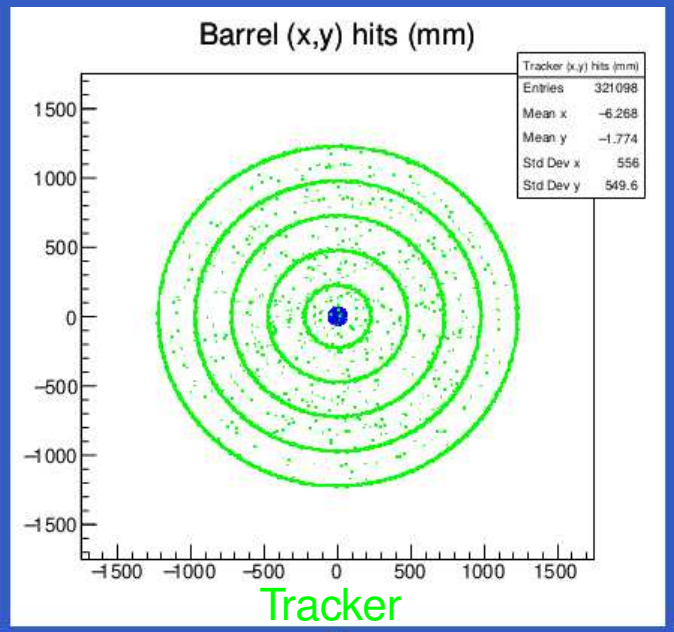


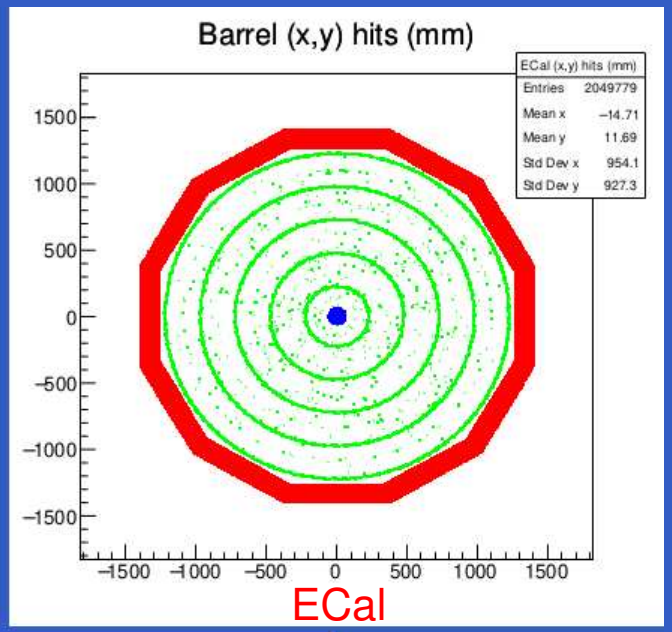
Obtaining Python Hit Lists with pyLCIO

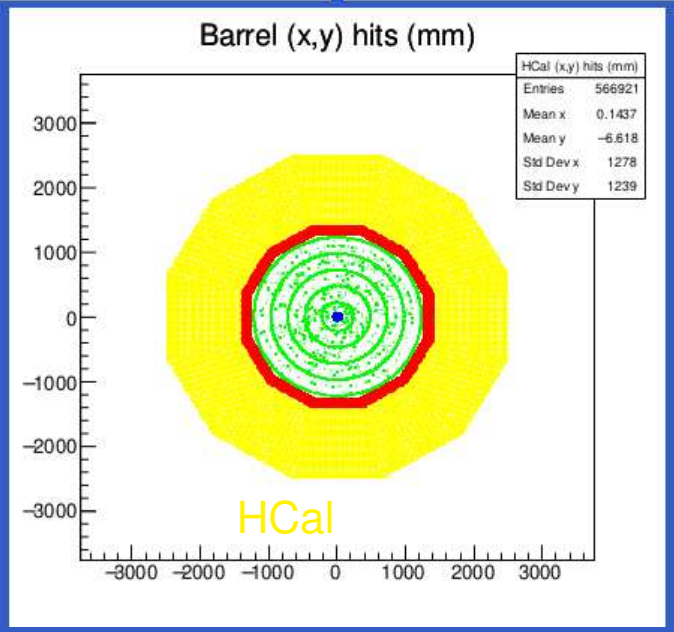
```
from pyLCIO import IOIMPL

reader=IOIMPL.LCFactory.getInstance().createLCReader()
reader.open('projects_ilcsoft/slcio/hzmumu250ub.hepmc.sim.digi.slcio')

for event in reader:

    vxd_hit_collection=event.getCollection('SiVertexBarrelHits')
    vxd_hits=[[hit.getPosition()[0], hit.getPosition()[1], \
        hit.getPosition()[2]] for hit in vxd_hit_collection]

    ecal_hit_collection=event.getCollection('ECalBarrelReco')
    ecal_hits=[[hit.getPosition()[0], hit.getPosition()[1], \
        hit.getPosition()[2],hit.getEnergy()] for hit in ecal_hit_collection]

#   now do something with those hits!
```

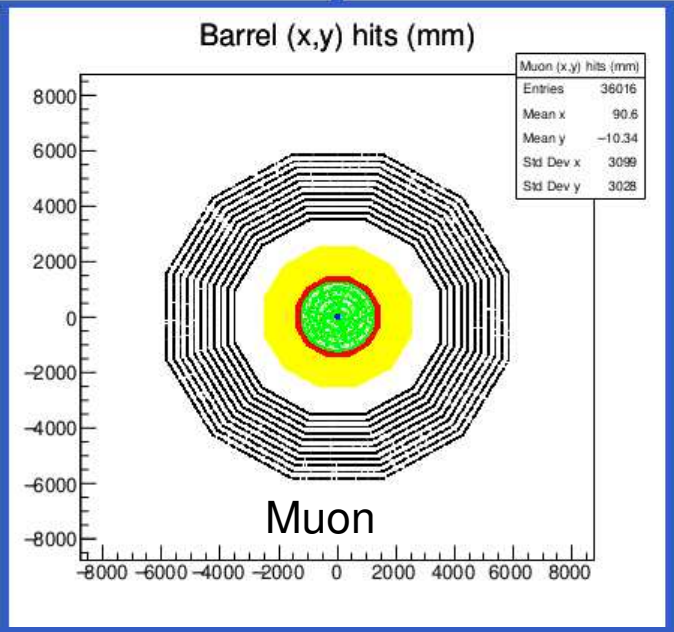
Subdetector Hitmaps for 1K Higgstrahlung Events





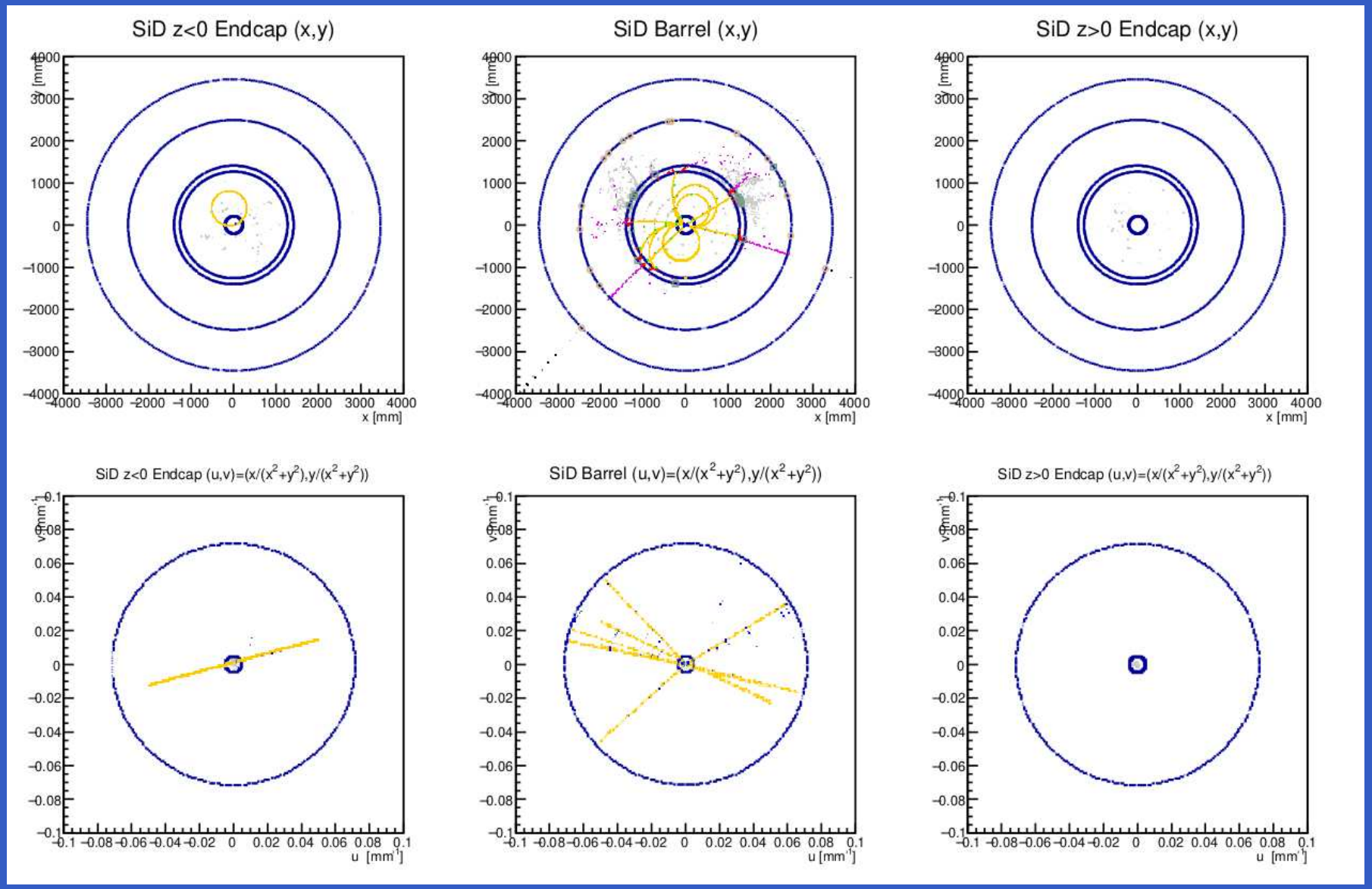






Hits only, no detector components. Offlayer hits misassigned from endcap to barrel hit collections.

Hits and Fits for 1 Higgstrahlung Event ($Z \to \mu^+\mu^-$)



Conformal track fits with $p_T > 0.1$ GeV (orange) and hits fitted by subdetector: VXD (blue), Tracker (green), ECal (red), HCal (magenta), Muon (black).

Functions in pySiDR.py (211 Lines)

from pySiDR.py

```
def get_track_parameters(track):
def get_track_chi2(track, resolution_xy=0.01):
def get_track_4v(track, mass_hypothesis=0.140):
def extrap_to_barrel(track, barrel_r):
def extrap_to_endcap(track, endcap_z):
def vxd_layer(hit):
def get_vxd_seeds(seed_layer1, seed_layer2, vxd_hits, pt_min=1.):
def clone_removal(tracks, nhit_common_max=1):
def find_tracks(seeds, hits, dist_max, pt_min=1., nhit_min=5, \
    nhit_common_max=1, chi2_max=10):
```

```
def ecal_layer(hit):
def build_one_ecal_topocluster(seed,hits,t_neighbor=0,t_cells=0):
def build_all_ecal_topoclusters(original_hits,t_seed=0.1, energy_min=1.):
def hcal_layer(hit):
def build_one_hcal_topocluster(seed,hits,t_neighbor=0,t_cells=0):
def build_all_hcal_topoclusters(original_hits,t_seed=0.1, energy_min=1.):
def cluster_xyz(cluster):
def cluster_energy(cluster):
def get_neutral_4v(cluster, mass_hypothesis=0):
def get_pfos(tracks, ecal_clusters, hcal_clusters, dist_max):
```

Functions in utilities.py (99 Lines), Fragment of sid_o2_v3.py

from utilities.py

```
def dist1(x,y):
def dist2(x,y):
def dist3(x,y):
def dist(x,y):
def dot_product(x,y):
def radialr(xyz):
def radialr_theta_phi(xyz):
def linear_solutions(a, b, c, d, e, f):
def quadratic_solutions(a, b, c):
def linear_regression_2d(xy_list):
def circular_regression_2d(xy_list):
def mean_stdev(x_list):
```

from sid_o2_v3.py

```
hcal_layer_width=27.18
hcal_symmetry=12
hcal_rmin=1406.
hcal_rmax=2487.
hcal_nrlayer=40
```

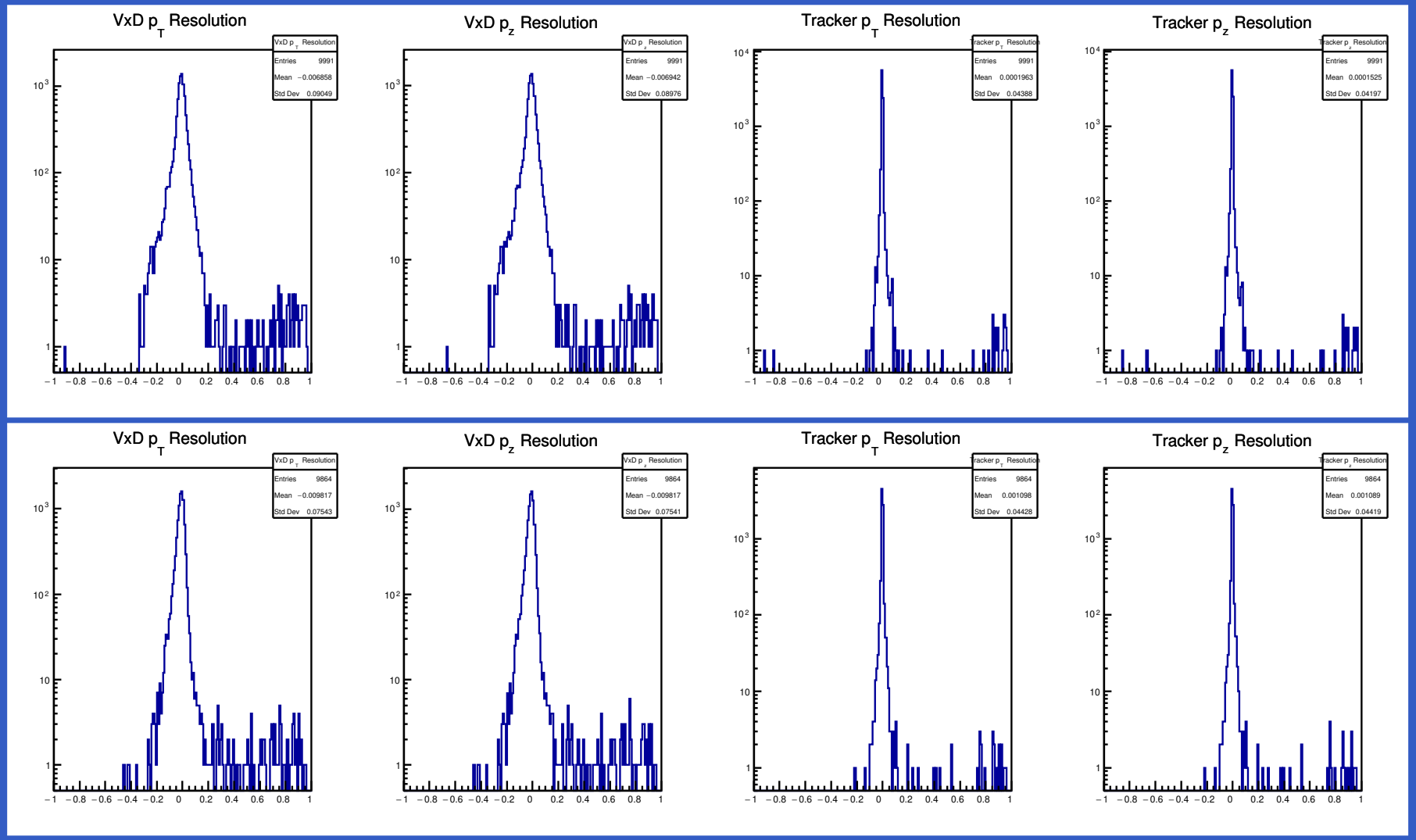
Track Finding in pySiDR

In pySiDR, a track is a list of hits [x,y,z]

```
def find_tracks(seeds, hits, dist_max, pt_min=1, nhit_min=5, \
    nhit_common_max=1, chi2_max=10):

    track_candidates=[]
    for seed in seeds:
        track=list(seed)
        track.extend([hit for hit in hits if \
            dist3(hit, extrap_to_barrel(seed, radialr(hit)))<dist_max])
        track.extend([hit for hit in hits if \
            dist3(hit, extrap_to_endcap(seed, abs(hit[2])))<dist_max])
        if len(track)<nhit_min or get_track_chi2(track)>chi2_max:
            continue
        if radialr(get_track_4v(track))>=pt_min:
            track_condidates.append(track)
    tracks=clone_removal(track_candidates, nhit_common_max)
    return tracks
```

Track p_T, p_z Resolution: Default Constraints

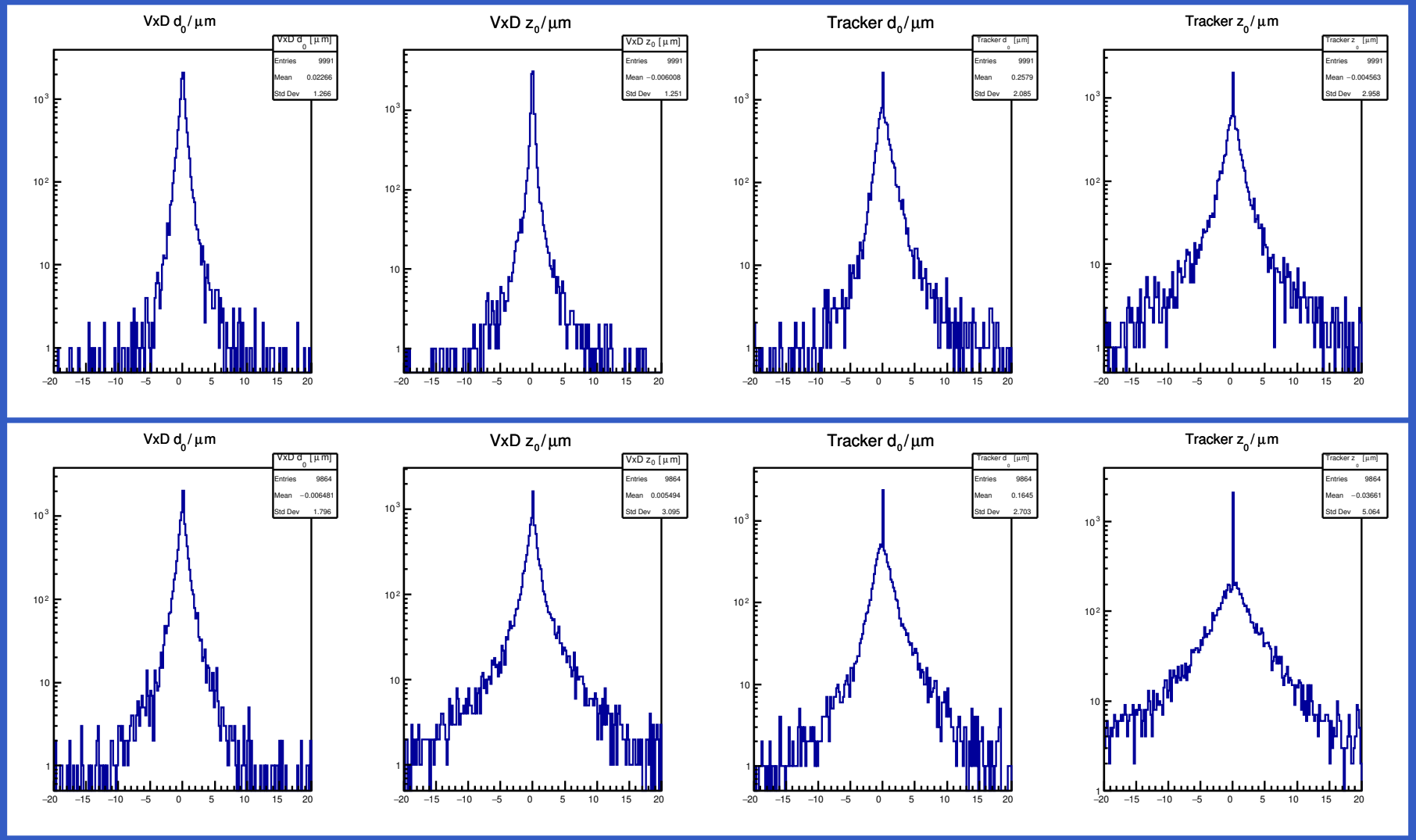
Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



Left to right, vertex detector p_T and p_z resolution, tracker p_T and p_z resolution. Results: 8-9% for VxD, 4% for Tracker ($1 < E < 125$ GeV).

Track d_0, z_0 Resolution: Default Constraints

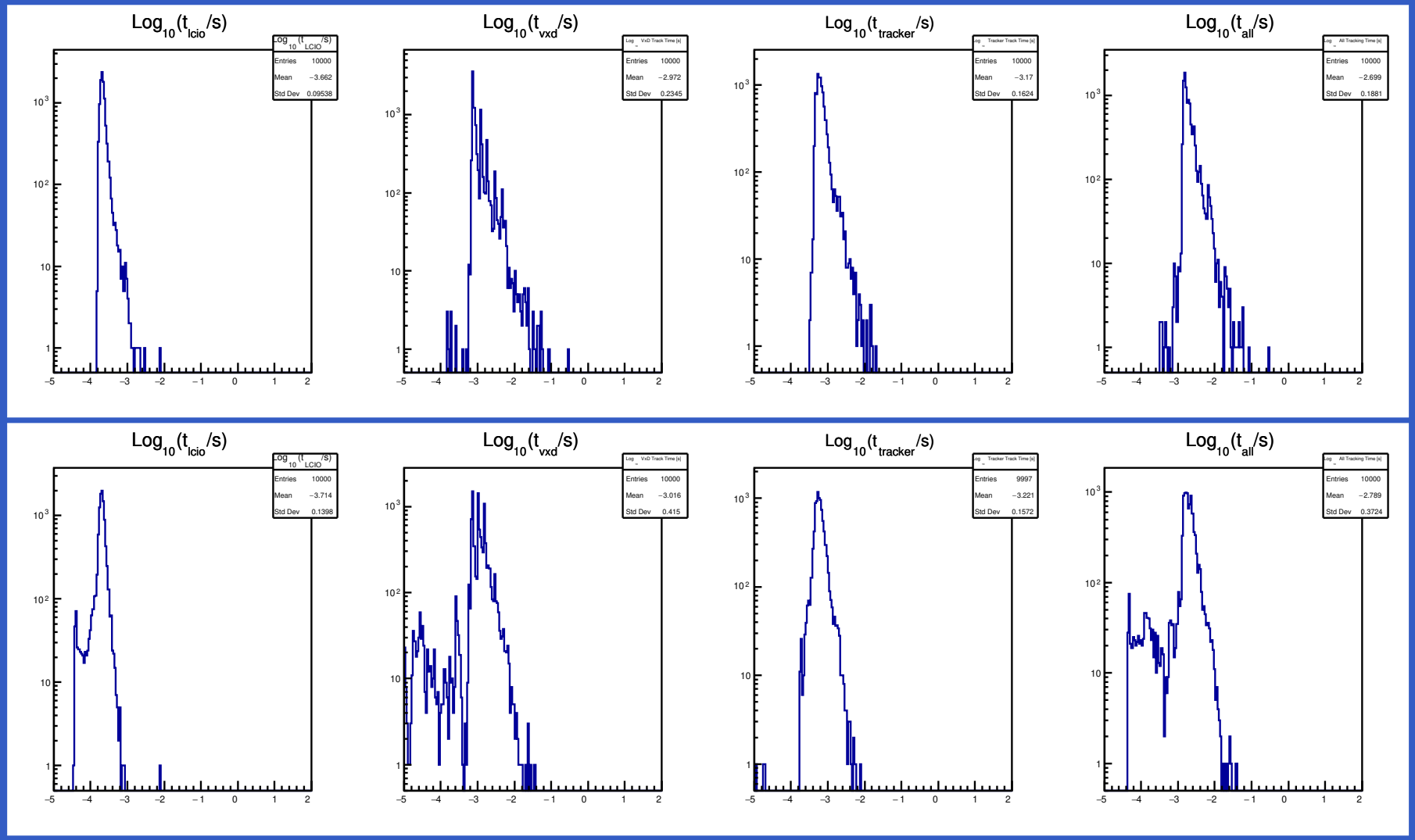
Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



Left to right, vertex detector d_0 and z_0 resolution, tracker d_0 and z_0 resolution. Results: 1-5 μm for VxD and Tracker ($1 < E < 125$ GeV).

Track Finding Timing: Default Constraints

Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



Left to right, time/event for LCIO access, vertex detector track finding, tracker track finding, all three. Timing performed with Python `time` module. Results: total timing around 2ms.

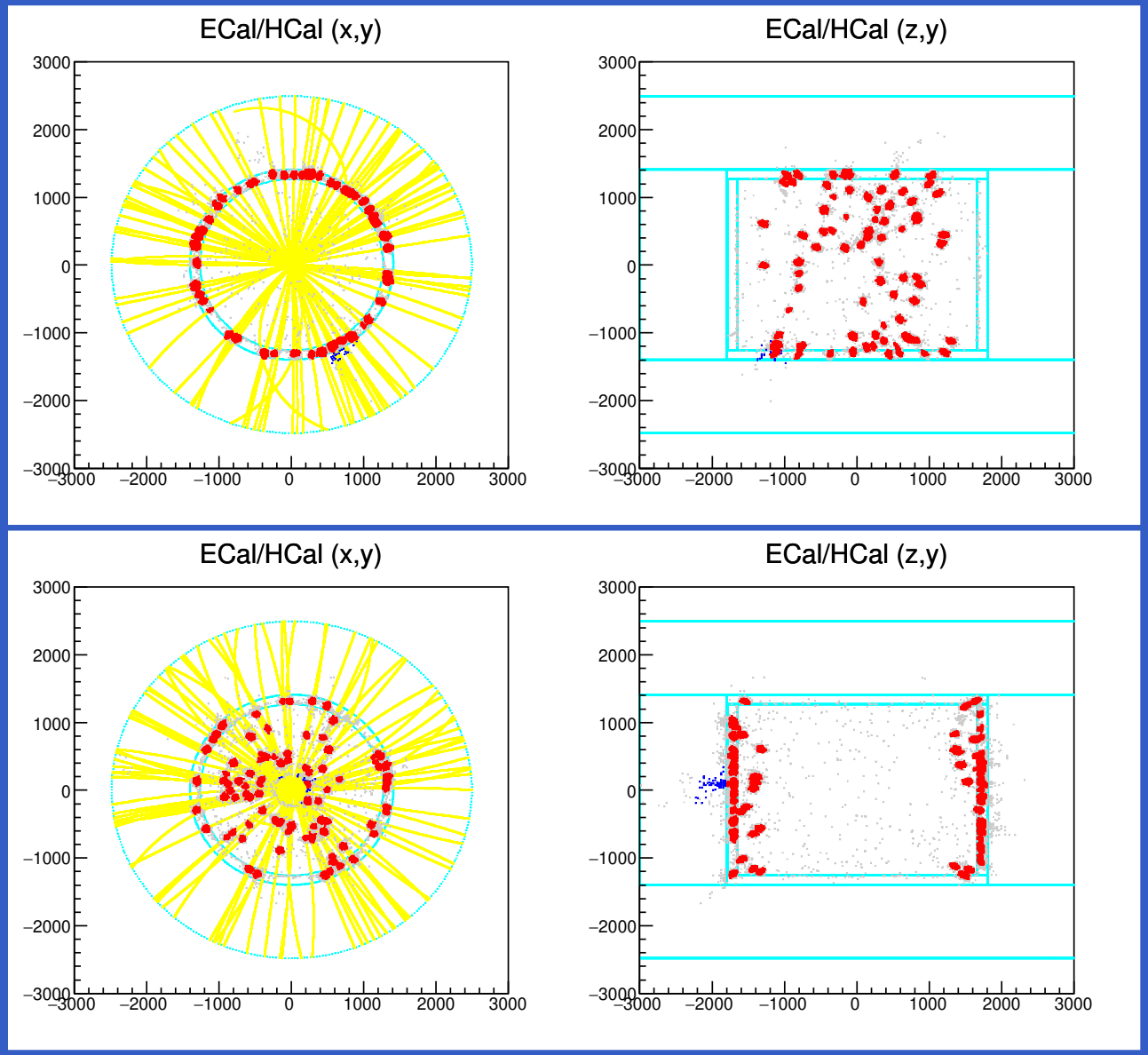
Topocluster Finding in pySiDR

In pySiDR, a cluster is a list of hits [x,y,z,E]

```
def build_one_ecal_topocluster(seed,hits,t_neighbor=0,t_cells=0):
    adjacent_candidates=[hit for hit in hits \
        if dist3(seed,hit)<5.*ecal_cell_size \
        and ecal_layer(seed)==ecal_layer(hit)]
    offlayer_candidates=[hit for hit in hits \
        if dist3(seed,hit)<5.*ecal_thin_layer_width \
        and abs(ecal_layer(seed)-ecal_layer(hit))==1]
    adjacent_candidates.extend(offlayer_candidates)
    addcells=[hit for hit in adjacent_candidates if hit[3]>t_cells]
    for addcell in addcells:
        hits.remove(addcell)
    neighbors=[hit for hit in addcells if hit[3]>t_neighbor]
    for neighbor in neighbors:
        addcells.extend(build_one_ecal_topocluster(neighbor,hits))
    return addcells
```

100 Single Electron Clusters

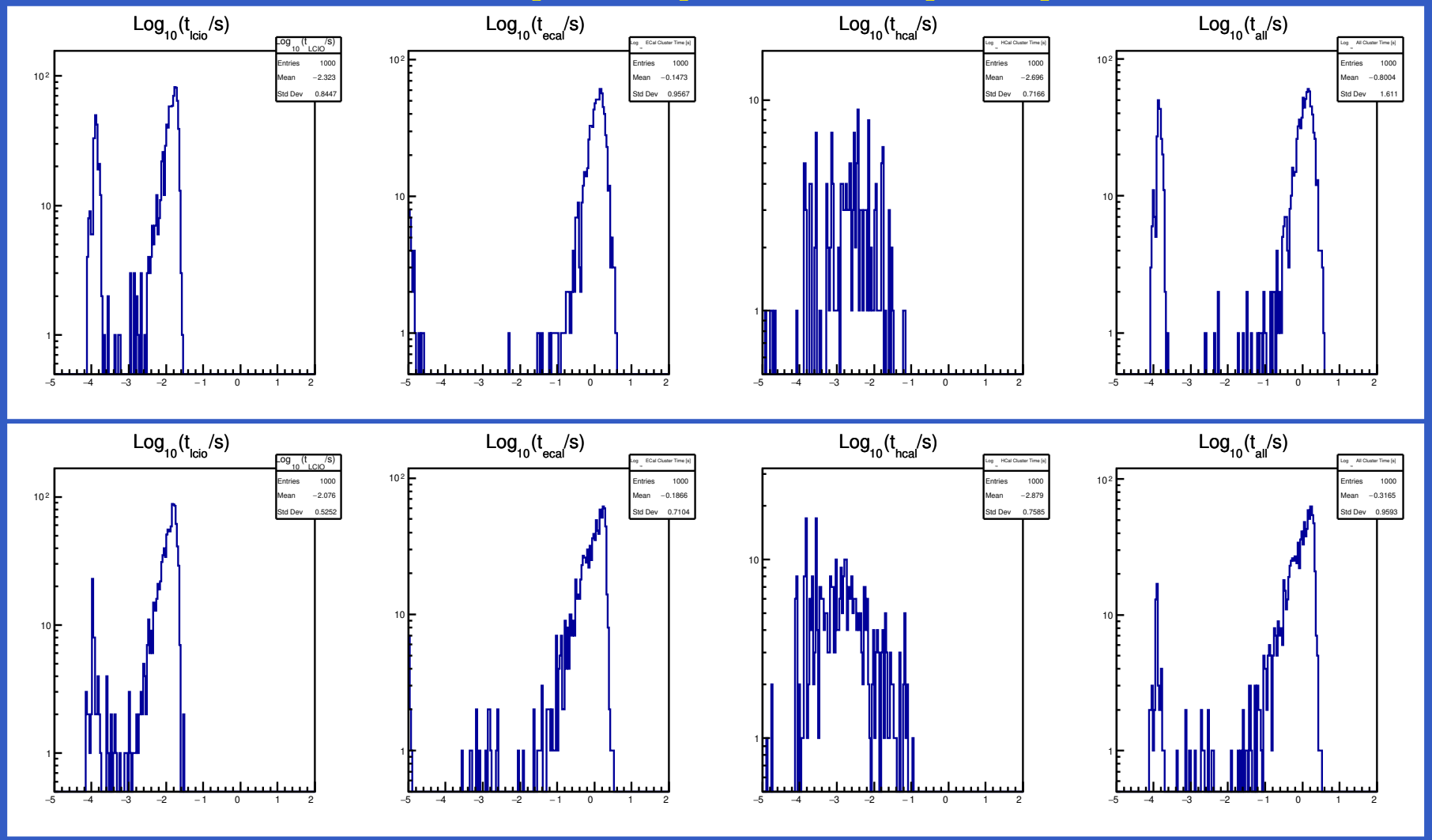
Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



ECal and HCal hits, ECal hits in clusters, HCal hits in cluster, detector, MC Truth.

Timing: 1K Single Electron Clusters

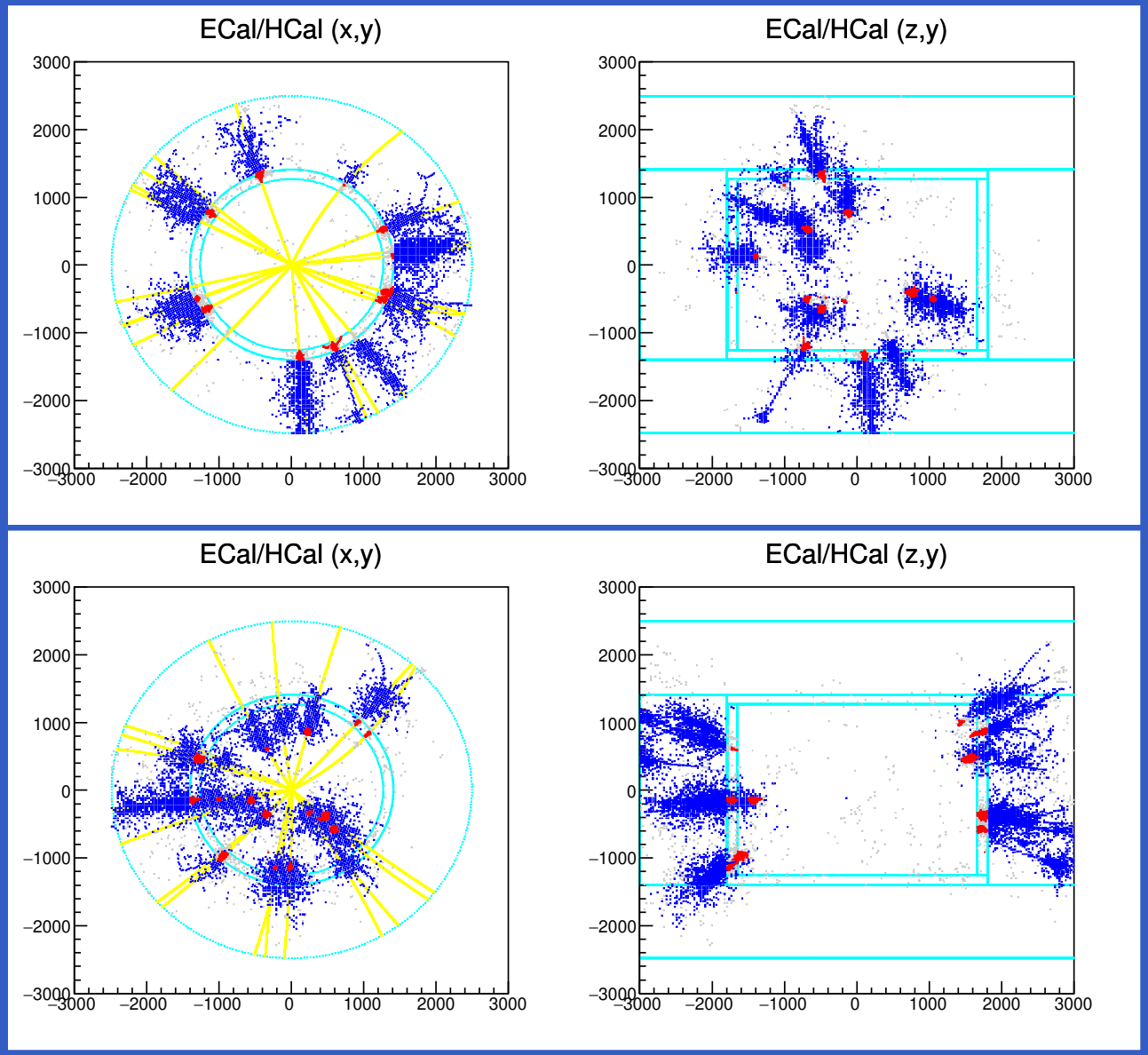
Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



Left to right, time/event for LCIO access, ECal clustering, HCal clustering and for all three. Timing performed with Python `time` module. Results: total timing around 0.2s-0.5s.

10 Single Pion Clusters

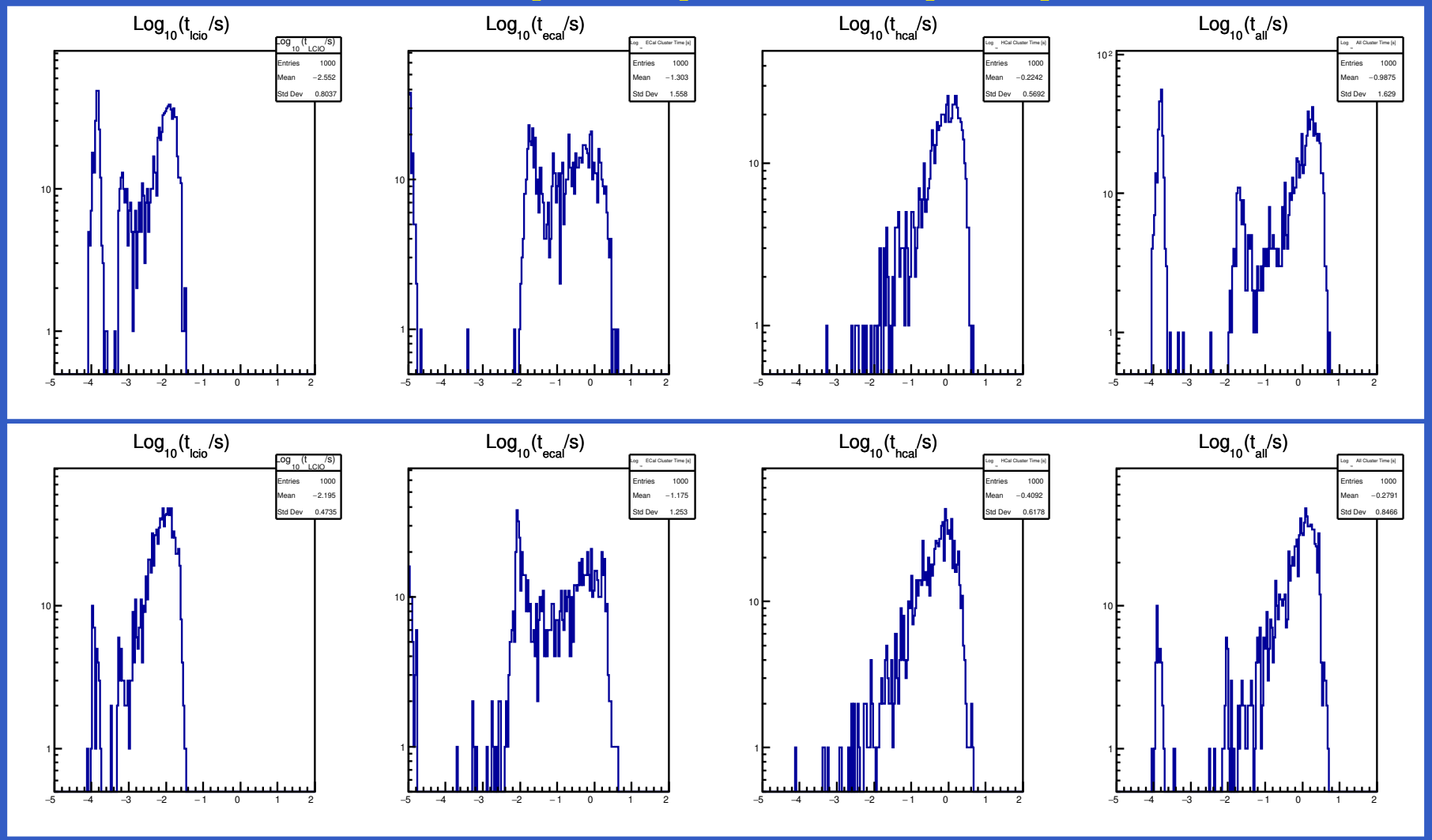
Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



ECal and HCal hits, ECal hits in clusters, HCal hits in cluster, detector, MC Truth.

Timing: 1K Single Pion Clusters

Above, $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, below $\theta < -\frac{\pi}{4}, \theta > \frac{\pi}{4}$



Left to right, time/event for LCIO access, ECal clustering, HCal clustering and for all three. Timing performed with Python `time` module. Results: total timing around 0.2s-0.5s.

Marlin Reconstruction Time/Event

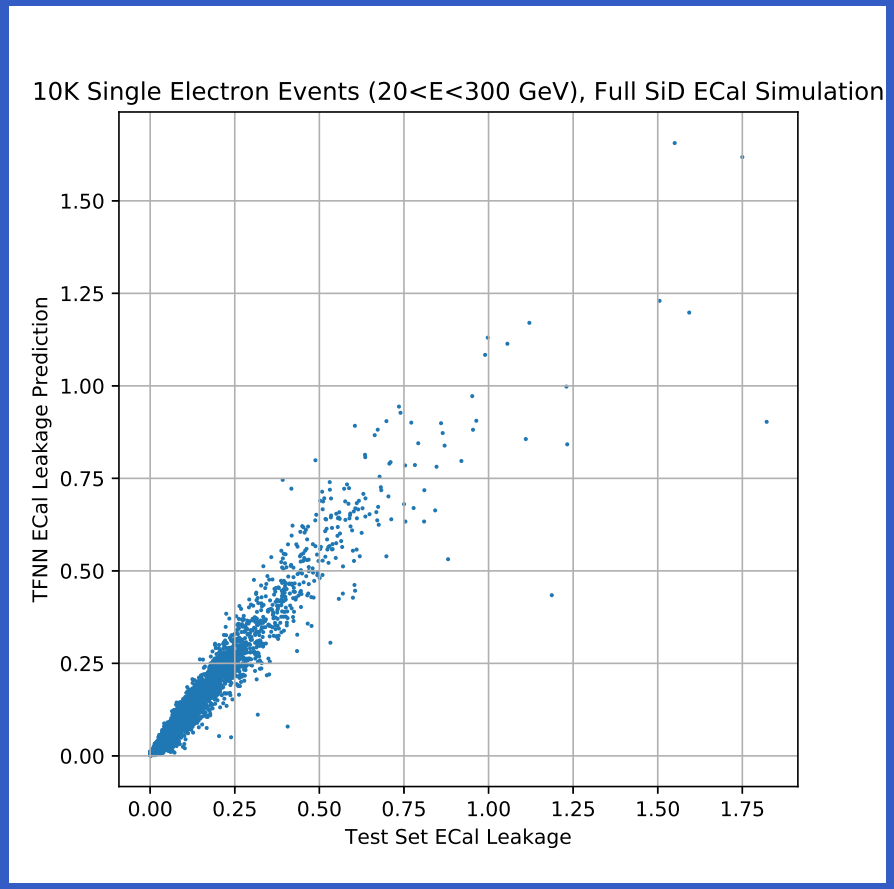
Processor	Single Muon Time	Fraction	Higgstrahlung Time	Fraction
Digitization Only	0.0017s	6%	0.078s	2%
And Conformal Tracking	0.028s	88%	2.9s	91%
And PandoraPFA	0.030s	6%	3.2s	7%

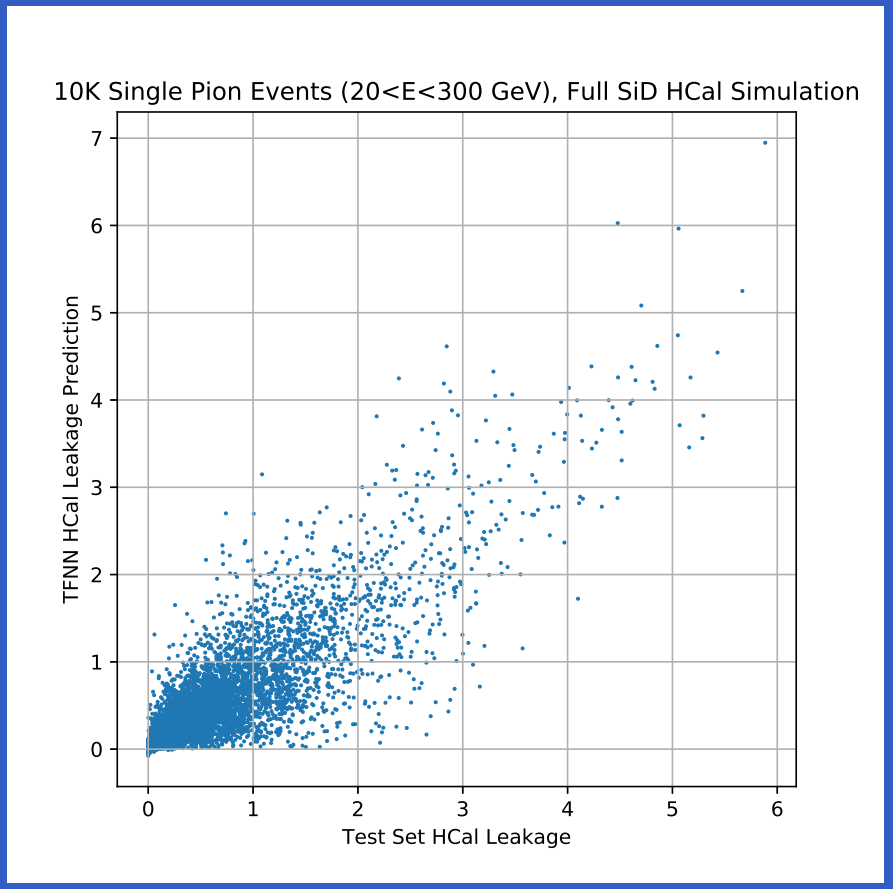
Time/event for Marlin. Processors invoked in execute tag of SiDReconstruction.xml.

- Times are not definitive, but suggestive, and definitive timing studies should be performed.
- Reported times are obtained with Linux time command with the Marlin executable from CVMFS.
- Compare to pySiDR trackfinding (2ms) and clusterfinding (0.2-0.5s) for single muon events.

TensorFlow ECal and HCal Leakage Recovery

Predicted vs actual leakage in the ECal (left) and HCal (right).





- Neural Network fit to energy deposit by layer yields a prediction for leakage.
- Leakage recovery is not only important on its own: it can also improve energy resolution.
- See also "Application of the Machine Learning to the collider experiments", M. Iwasaki, this session.

Conclusions

- Simulation with ddsim is compute intensive, but digitization with Marlin is NOT.
- Reconstruction with Marlin is compute intensive, particularly for tracking.
- Reconstruction can instead be performed with Python and pyLCIO using LCIO digitization files made with Marlin.
- Status of Trackfinding and Clusterfinding in pySiDR
  - Trackfinding is fast and reconstruction performant.
  - Topoclusterfinding is slow and performance needs evaluation.
  - Clusterfinding issues to be addressed: merging of ECal/HCal, Barrel/Endcap
  - Preliminary particle flow is promising but needs to wait on clusterfinding solutions.
  - Performance in complex events must still be evaluated.
- Status of pySiDR Package
  - The package will soon be found in SiDPerformance in ILCsoft on GitHub
  - pySiDR is a work on progress, not a polished final product, and can easily be improved.
  - Expected future development includes particle flow, leakage recovery, bremstrahlung recovery, jetfinding and vertexing.
  - Despite its name, pySiDR is geometry agnostic and may become geometry atheistic.