PFA Framework & PFA Status

Mat Charles (mcharles@slac.stanford.edu)
The University of Iowa

Why a PFA framework?

Writing a PFA is hard -- many non-trivial components

+ Find and identify photons & electrons
+ Find MIPs
+ Find hadronic clusters
+ Find tracks
+ Extrapolate tracks to the calorimeter
+ Match tracks up with clusters
+ Separate charged & neutral hadrons
+ Handle isolated hits/fragments
+ Turn tracks/clusters into particles
+ Energy calibration
+ Make some useful plots at the end

... and even "cheat" versions of these can be non-trivial.

⇒ A lot of overhead, not enough physics.

What is the PFA framework?

First, conventions on how developers write algorithms:

+ Use a modular design, writing algorithms as a series of Drivers.
+ When Drivers need to communicate, they should do it by storing objects in the event with put() and get().
+ Drivers should leave their input unchanged; the output should be a new, separate collection.
+ Store a group of hits as a HitMap.
+ Store a group of clusters as a List<Cluster> (similarly Track, ReconstructedParticle etc.)

What is the PFA framework?

Second, a library of Drivers to let YOU work on PFA.

+ Get started much faster (avoid re-inventing the wheel)
+ Swap individual components in & out -- e.g. see how much difference a new photon-finder makes;
+ Run the same PFA on a new detector design;
+ Try a physics analysis with non-toy simulation.

We are not there yet, but we are getting close...

PFA Framework webpage

lcsim PFA guide - Linear Collider - SLAC Confluence

https://confluence.slac.stanford.edu/display/ilc/lcsim+PFA+guide

BaBar ▾ LC ▾ Physics ▾ Misc SLAC ▾ Computing ▾ Personal ▾ Singly charmed bary... Xic review Omegac review Useful ▾ OPR ▾ Lambdac review ▾

Version 5.10 | lcsim PFA guide - Linear Collider ...

Dashboard > Linear Collider > ... > lcsim Tutorials > lcsim PFA guide

Search

Linear Collider

lcsim PFA guide

Welcome Mat Charles | History | Profile | Log Out

View | Edit | Attachments (0) | Info

Browse Space | Add Page | Add News

Added by Mat Charles, last edited by Mat Charles on Feb 10, 2006 (view change)

Labels: (None) EDIT

Guide for Particle Flow Algorithm developers in org.lcsim

This page documents the framework for developing PFA algorithms, explains the conventions used, and gives example implementations.

- Conventions
- Worked examples
  - HitMap manipulation
  - A very trivial PFA
  - Using DigiSim
  - Reading in and writing out hitmaps
  - How to make things appear in WIRED or the Event Browser in JAS3
- Outline of a complete PFA
- Things that need doing

Conventions

In order to make the PFA components as interchangeable as possible, we have adopted some conventions. These were discussed at the January 2005 Boulder simulation workshop. They will probably evolve slowly over time.

- PFAs should be structured as a series of Drivers. This way, components can be swapped in and out easily.

https://confluence.slac.stanford.edu/display/ilc/lcsim+PFA+guide

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim
    add(new org.lcsim.recon.cluster.util.CalHitMapDriver());
    add(new org.lcsim.digisim.DigiSimDriver());
    add(new org.lcsim.digisim.SimCalorimeterHitsDriver());

    // Convert DigiSim output into a HitMap:
    HitListToHitMapDriver digiHitMap = new HitListToHitMapDriver();
    digiHitMap.addInputList("EcalBarrDigiHits");
    digiHitMap.addInputList("EcalEndcapDigiHits");
    digiHitMap.addInputList("HcalBarrDigiHits");
    digiHitMap.addInputList("HcalEndcapDigiHits");
    digiHitMap.setOutput("digi hitmap");
    add(digiHitMap);

    // Set up MC truth
    add(new CreateFinalStateMCParticleList("Gen"));

    // Cluster the hits (perfect pattern recognition)
    PerfectClusterer clusterer = new PerfectClusterer();
    clusterer.setInputHitMap("digi hitmap");
    clusterer.setOutputHitMap("leftover hits");
    clusterer.setOutputClusterList("perfect clusters");
    clusterer.setMCParticleList(”GenFinalStateParticles”);
    add(clusterer);

    // Find tracks
    add (new org.lcsim.mc.fast.tracking.MCFastTracking());

    // ID the clusters and create reconstructed particles
    PerfectIdentifier id = new PerfectIdentifier();
    id.setInputClusterList("perfect clusters");
    id.setOutputParticleList("perfect particles");
    id.setMCParticleList(”GenFinalStateParticles”);
    id.setInputTrackList(EventHeader.TRACKS);
    add(id);

    // Plot the total energy
    add(new EnergySumPlotter("perfect particles", "perfect.aida"));
    add(new CorrectedEnergySumPlotter("digi hitmap", "perfect particles", mcList, "corrected.aida"));
  }
}
```

(Close to org.lcsim.plugin.web.examples.TrivialPFA)

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

All the code goes in the constructor of your PFA class.

Most of it is just setting the parameters of the Drivers.

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

```
// Run DigiSim
add(new org.lcsim.recon.cluster.util.CalHitMapDriver());
add(new org.lcsim.digisim.DigiSimDriver());
add(new org.lcsim.digisim.SimCalorimeterHitsDriver());
```

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

```
// Convert DigiSim output into a HitMap:
HitListToHitMapDriver digiHitMap = new HitListToHitMapDriver();
digiHitMap.addInputList("EcalBarrDigiHits");
digiHitMap.addInputList("EcalEndcapDigiHits");
digiHitMap.addInputList("HcalBarrDigiHits");
digiHitMap.addInputList("HcalEndcapDigiHits");
digiHitMap.setOutput("digi hitmap");
add(digiHitMap);
```

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

```
// Set up MC truth
add(new CreateFinalStateMCParticleList("Gen"));
```

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

```
// Cluster the hits (perfect pattern recognition)
PerfectClusterer clusterer = new PerfectClusterer();
clusterer.setInputHitMap("digi hitmap");
clusterer.setOutputHitMap("leftover hits");
clusterer.setOutputClusterList("perfect clusters");
clusterer.setMCParticleList(”GenFinalStateParticles”);
add(clusterer);
```

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

```
// Find tracks
add (new org.lcsim.mc.fast.tracking.MCFastTracking());
```

A Trivial PFA

```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

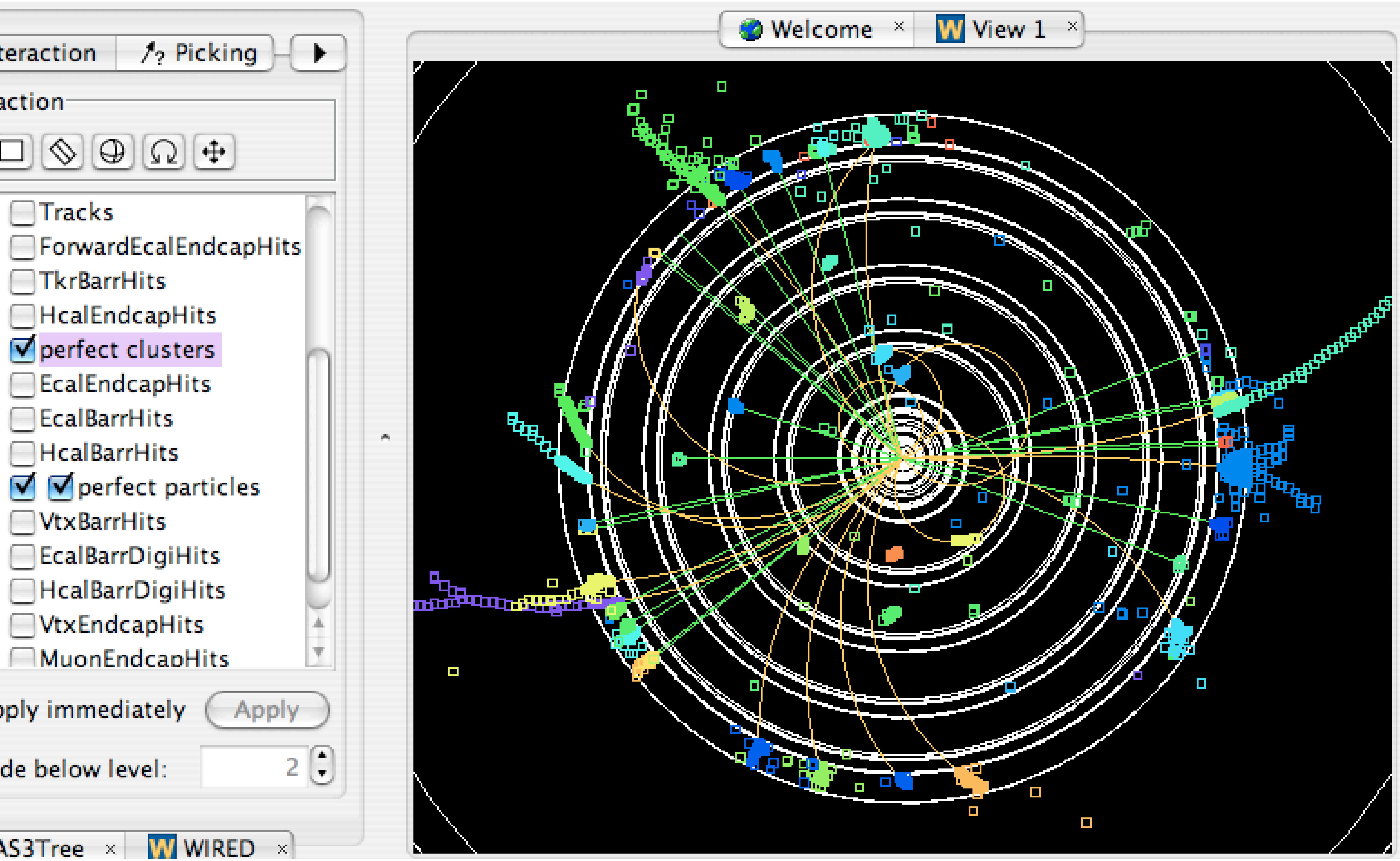
```
// ID the clusters and create reconstructed particles
PerfectIdentifier id = new PerfectIdentifier();
id.setInputClusterList("perfect clusters");
id.setOutputParticleList("perfect particles");
id.setMCParticleList(”GenFinalStateParticles”);
id.setInputTrackList(EventHeader.TRACKS);
add(id);
```

A Trivial PFA

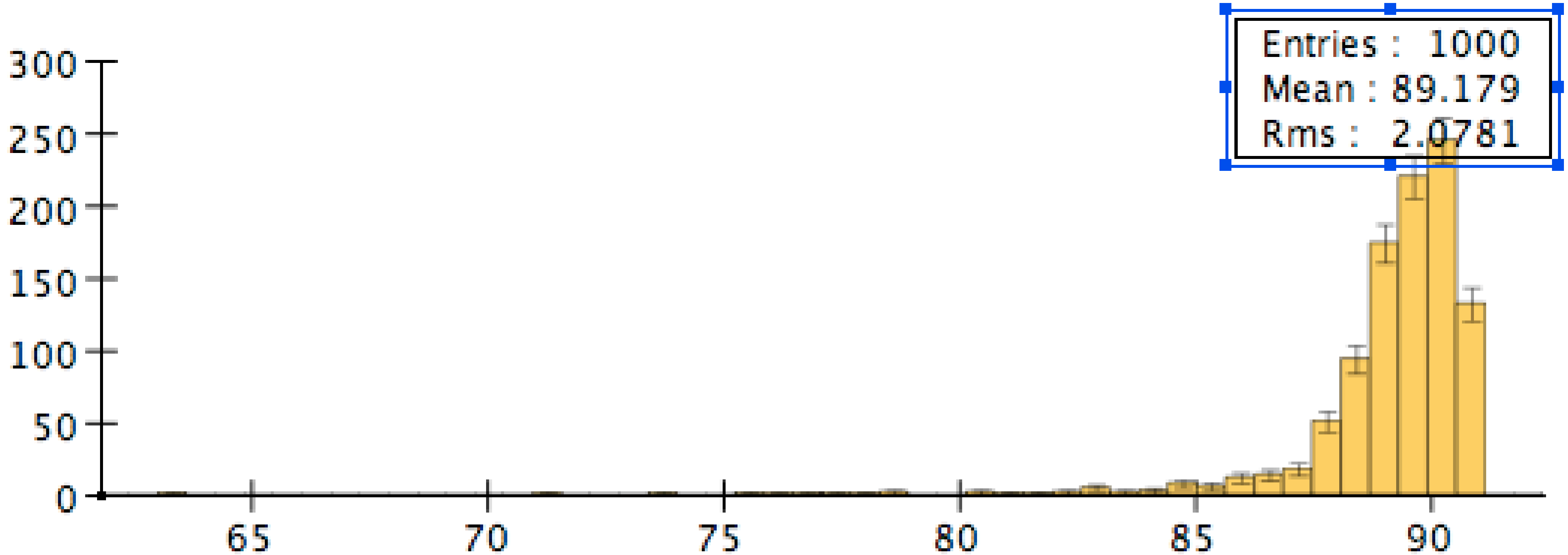
```
public class TrivialPFA extends Driver
{
  public TrivialPFA()
  {
    // Run DigiSim

    // Convert DigiSim output into a HitMap:

    // Set up MC truth

    // Cluster the hits (perfect pattern recognition)

    // Find tracks

    // ID the clusters and create reconstructed particles

    // Plot the total energy
  }
}
```

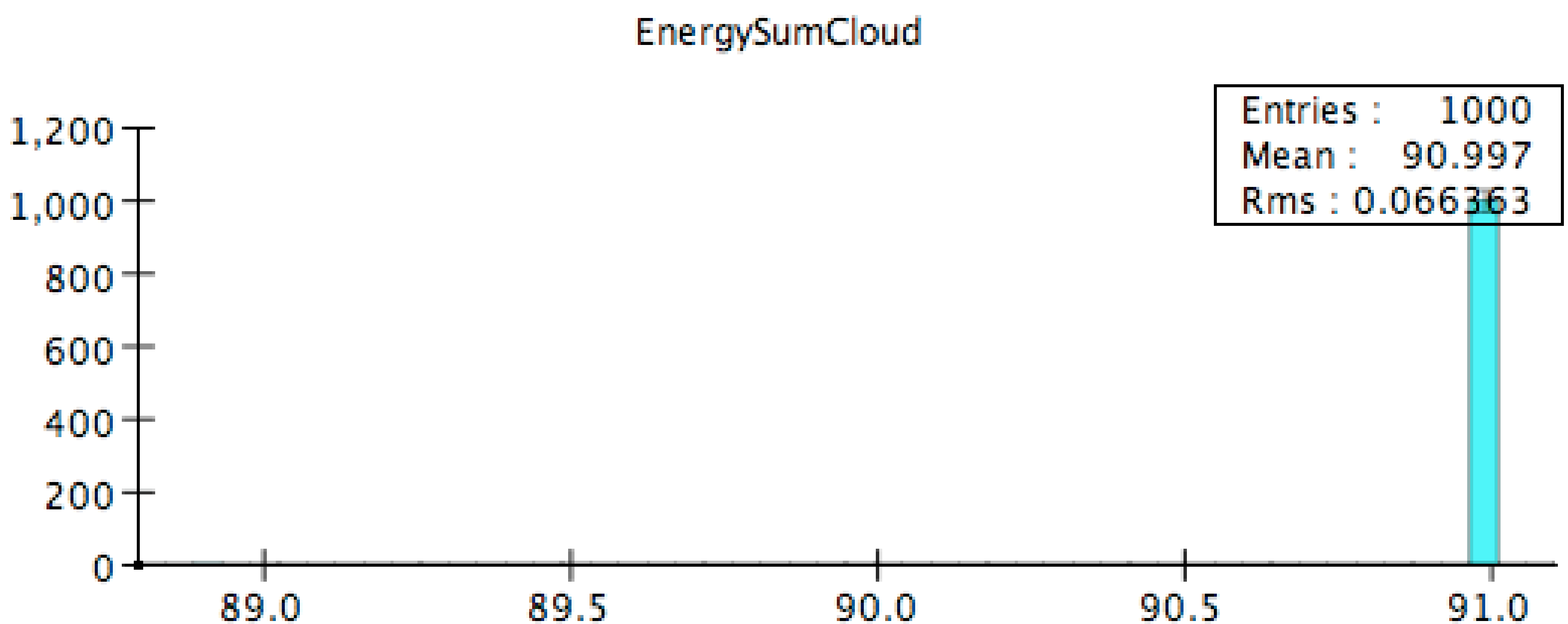
```
// Plot the total energy
add(new EnergySumPlotter("perfect particles", "perfect.aida"));
add(new CorrectedEnergySumPlotter("digi hitmap", "perfect particles",
    “GenFinalStateParticles”, "corrected.aida"));
```

Output of this trivial PFA



Output of this trivial PFA





A more realistic PFA

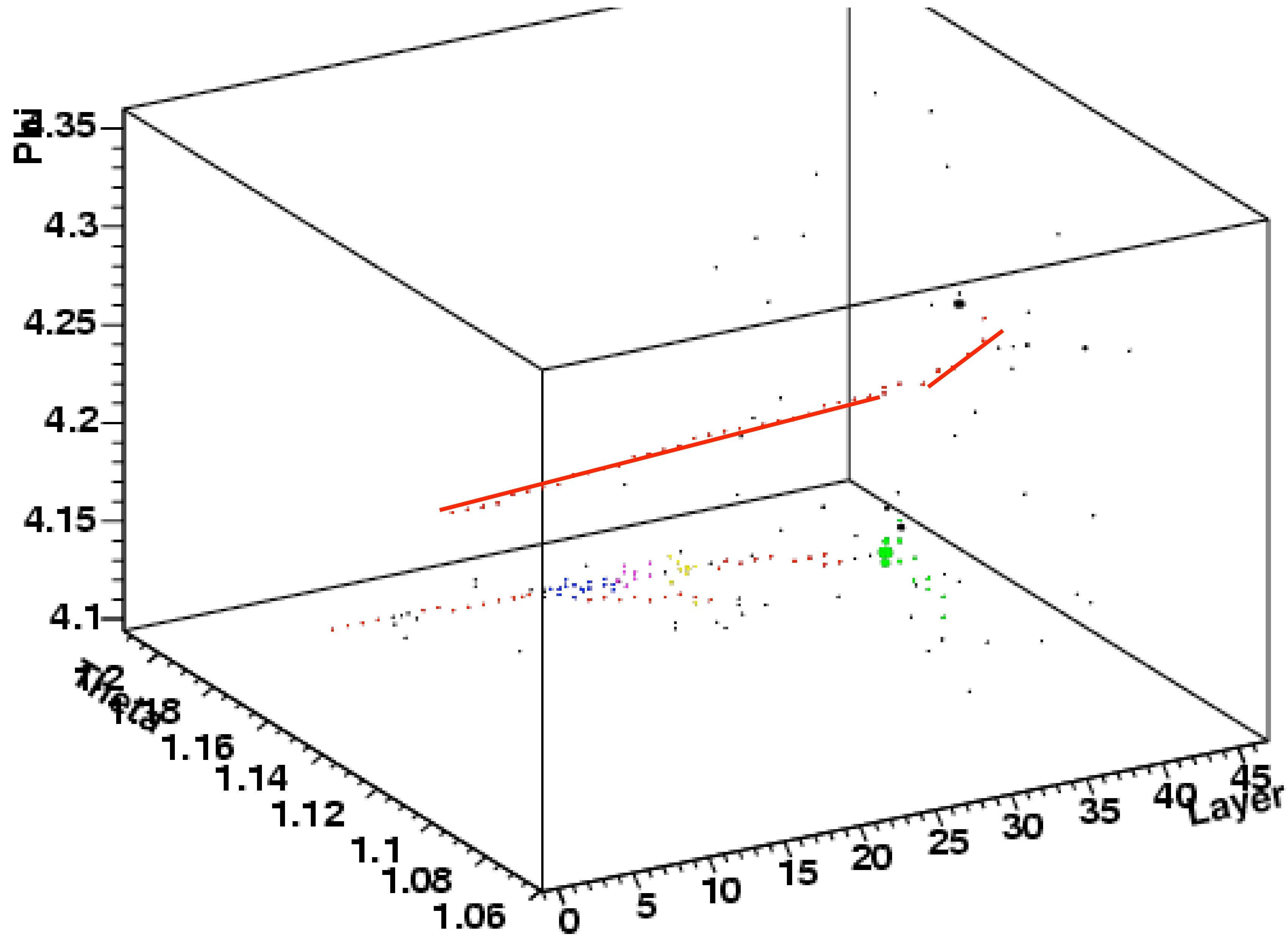
```
public class NonTrivialPFA extends Driver
{
  public NonTrivialPFA()
  {
    // Set up MC truth
    // Find tracks
    // Run DigiSim
    // Convert DigiSim output into a HitMap:
    // Find track segments in the ECAL and HCAL
    // Find photons in the ECAL (cheating) & turn into particles
    // Find clumps of high local hit density in the ECAL and HCAL
    // Build large-scale clusters out of hits, track segs, clumps
    // Link across the ECAL-HCAL boundary
    // For large (>= 10 hit) clusters:
    //   Look at pairs of clumps & track segments inside cluster
    //   Use likelihood selector to decide if they belong together
    //   Add in nearby hits
    //   Make preliminary charged/neutral assignment (from tracks)
    // Identify fragments/secondaries
    // Merge fragments with nearby primaries
    // Make final charged/neutral assignments (track extrapolation)
    // Make ReconstructedParticles out of clusters & tracks
    // Make plots
  }
} Unstable snapshot at org.lcsim.contrib.uiowa.template.NonTrivialPFA
```

[see next slides]

Hadronic showers

- I use geometrical checks to see whether pieces of the skeleton should be linked.
- Separate likelihood selectors for...

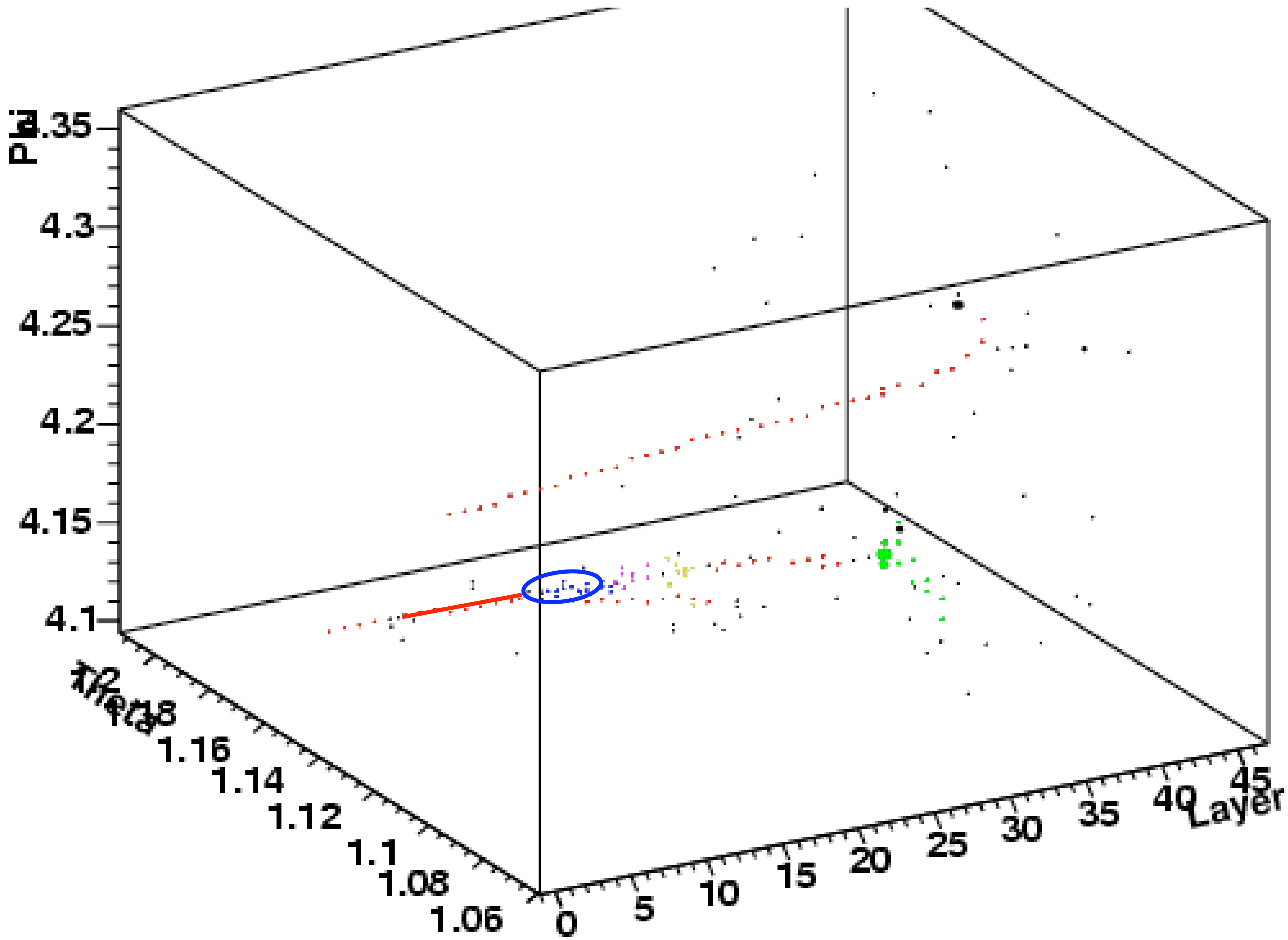
Track-Track links



Hadronic showers

I use geometrical checks to see whether pieces of the skeleton should be linked.
Separate likelihood selectors for...

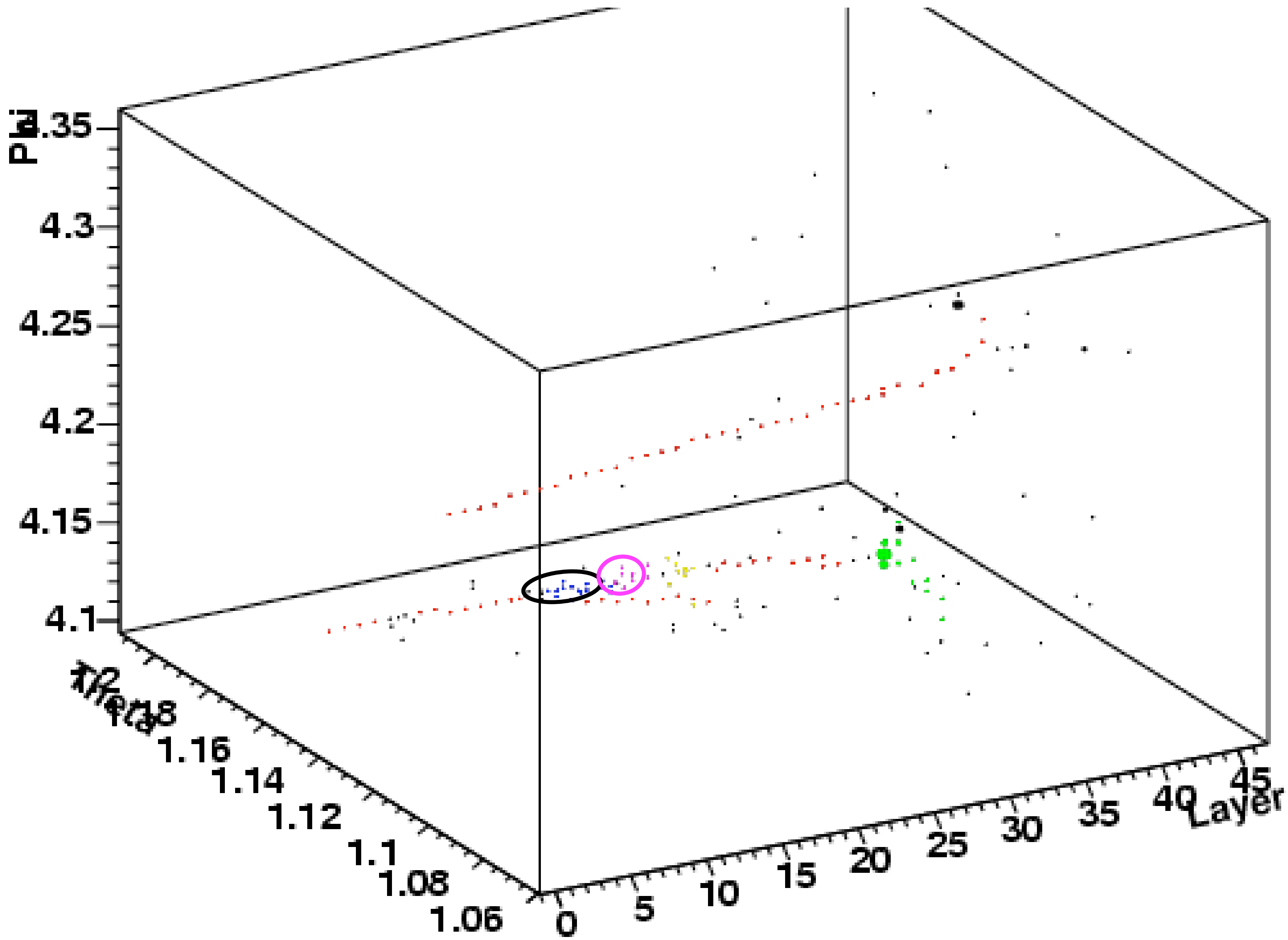
Track-Clump links



Hadronic showers

I use geometrical checks to see whether pieces of the skeleton should be linked.
Separate likelihood selectors for...

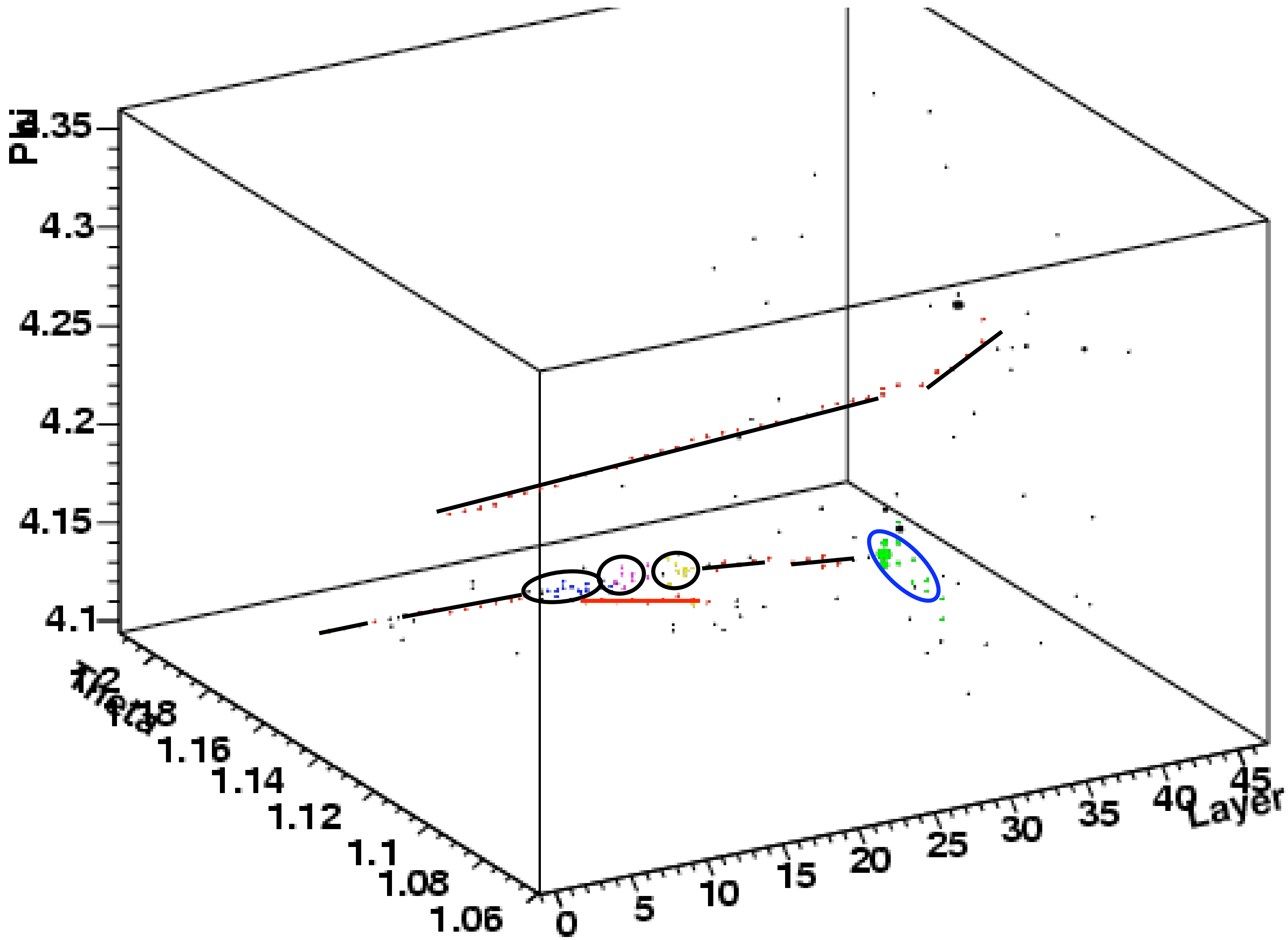
Clump-Clump links



Hadronic showers

I use geometrical checks to see whether pieces of the skeleton should be linked.

Putting it all together:



List of likelihood variables

- Clump-Clump:
  - DOCA
  - Smallest dist from a hit in one cluster to a hit in the other
- Track-Clump:
  - DOCA
  - Smallest dist from a hit in one cluster to a hit in the other
- Track-Track:
  - DOCA
  - Smallest distance from track hit to POCA
  - Whether POCA is inside calorimeter

MISSING

  - Extrapolating track to POCA (or joint CoE for parallel & disjoint tracks)...
    - # Layers where a hit is not found
    - Fraction of layers where a hit is not found (ignoring layers with a hit from cluster itself)

PFA performance

Not really one unique figure of merit.

+ Intrinsic resolution and confusion terms only for evaluating the pattern recognition;

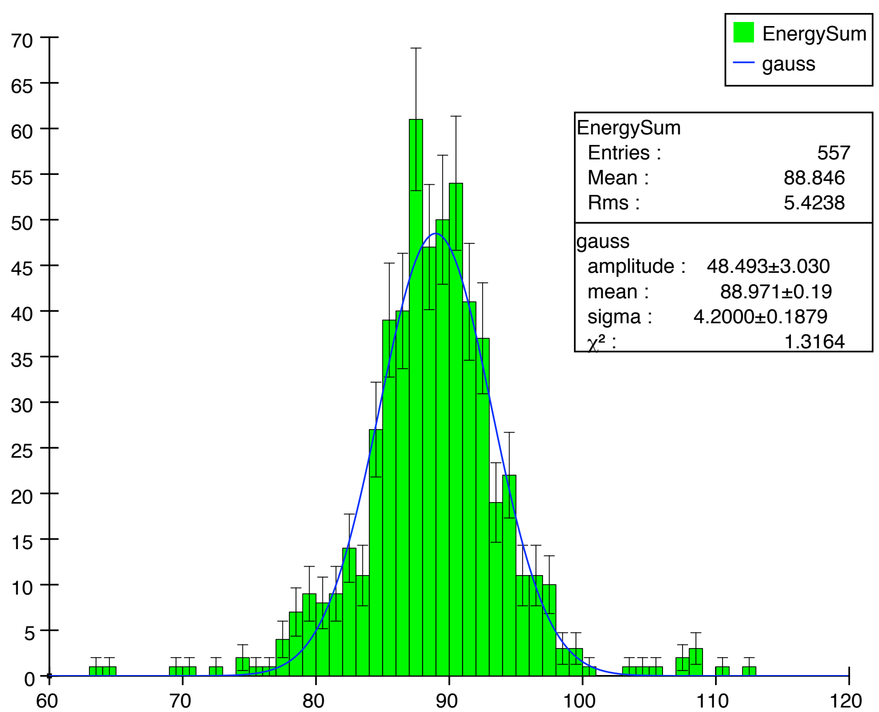
+ Energy sum corrected for missing energy is more realistic but includes things like calibration that aren't easy to get right;

+ Energy sum not corrected for missing energy

Energy sum corrected for missing energy

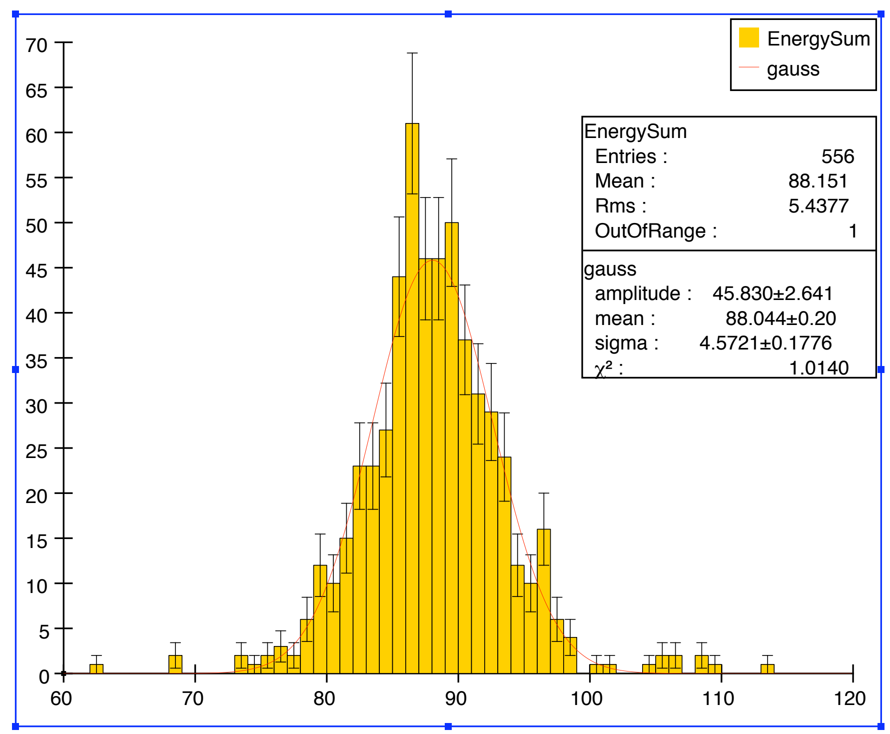
Cheating on...

Likelihood selector
Fragments
Photons



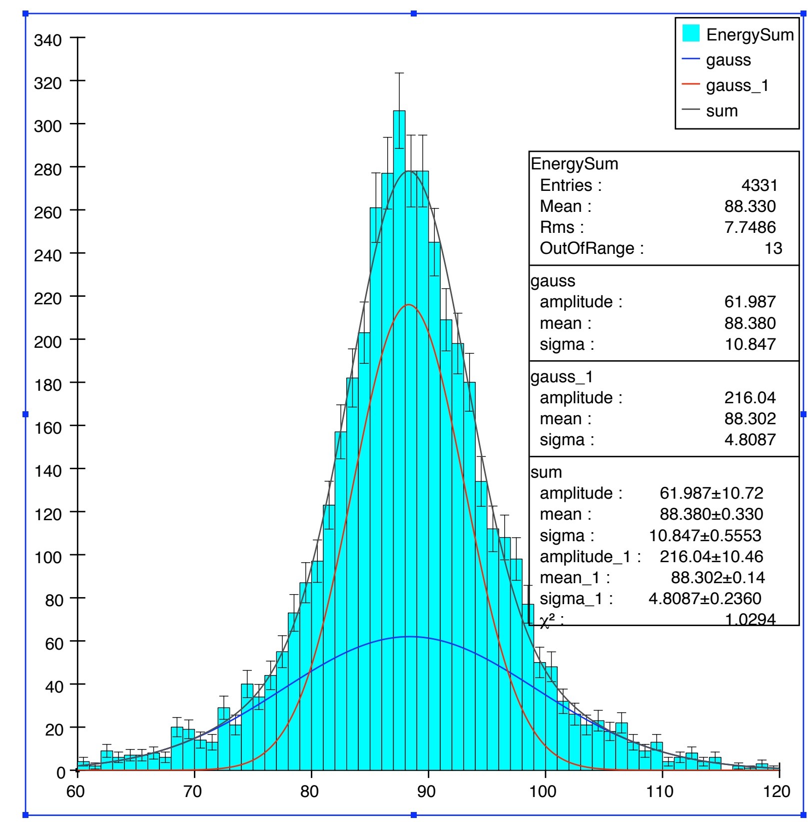
sigma = 4.2 GeV
HWHM = 4.9 GeV
RMS = 5.4 GeV

Likelihood selector
Photons



sigma = 4.5 GeV
HWHM = 5.4 GeV
RMS = 5.4 GeV

Photons



HWHM = 6.5 GeV
RMS = 7.7 GeV

This is ~ an upper bound on the resolution

Intrinsic resolution and confusion terms only

Derived from a toy MC, which takes the confusion distributions from the full simulation as inputs

Cheating on...

Likelihood selector, Fragments, Photons	Likelihood selector, Photons	Photons
sigma = 2.9 GeV	sigma = 4.3 GeV	
RMS = 3.4 GeV	RMS = 5.5 GeV	RMS = 8.6 GeV

This is ~ a lower bound on the resolution

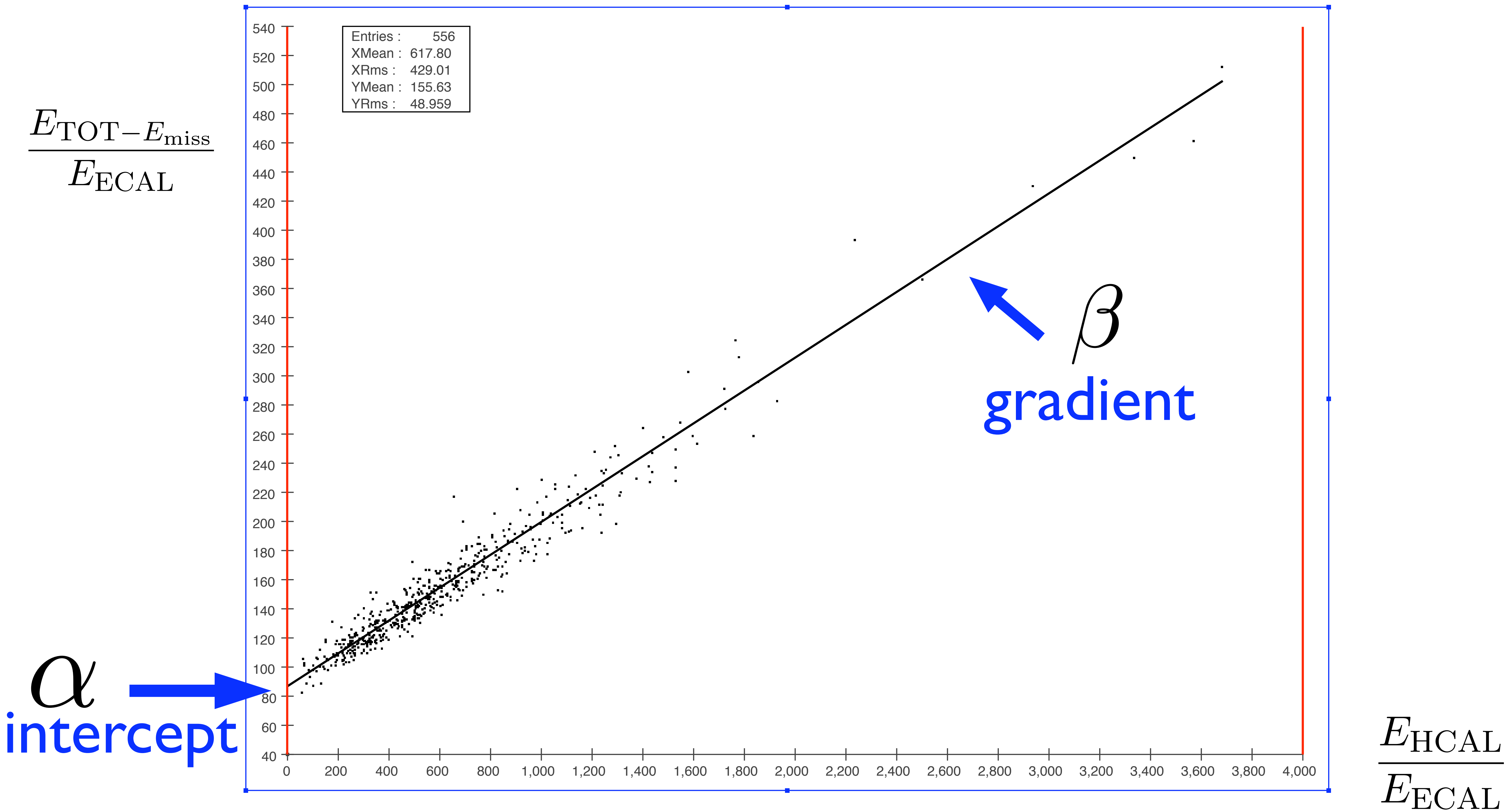
Next steps

- Finally after ICHEP I should have some real time
- PFA framework development
- Likelihood selector breaks clusters up too much
  - Add variables? (missing hits for track extrapolation)
  - Use E/P? Second pass?
- Fragment handling needs to improve
  - Probabilistic/fuzzy hit assignment? [Adam Para]
  - Regional information (nearby charged clusters, neutral clusters, helices) [Usha Mallik]
  - Likelihood selector to tie information together?

Post-script

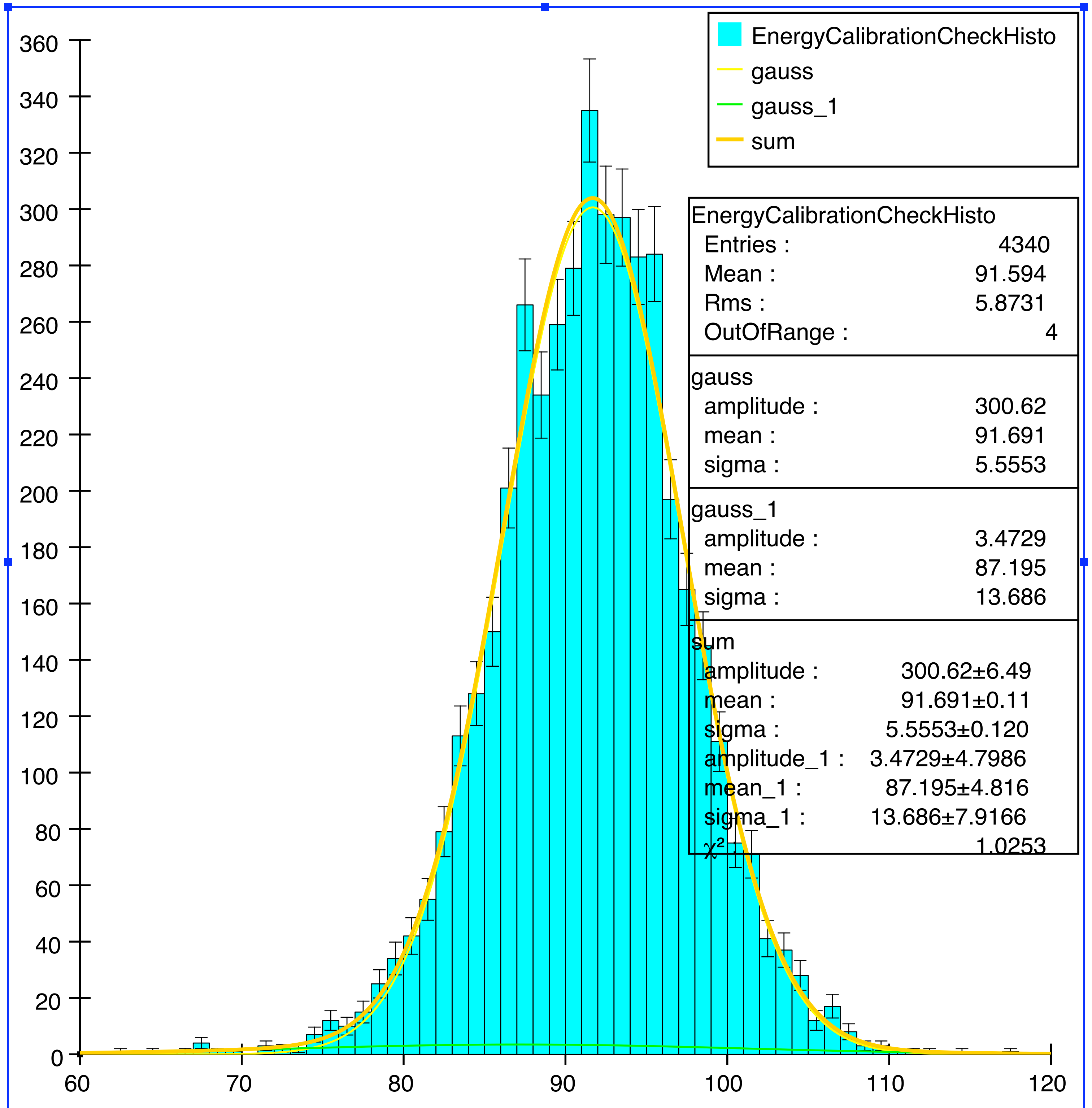
Comparison to a non-PFA straw man algorithm

$$\alpha E_{\mathrm{ECAL}} + \beta E_{\mathrm{HCAL}} + E_{\mathrm{miss}} = E_{\mathrm{TOT}} = 91.0\mathrm{GeV}$$



Post-script

Comparison to a non-PFA straw man algorithm



Double Gaussian fit

Mean	91.7 GeV	87.2 GeV
Sigma	5.6 GeV	13.7 GeV
Area	97%	3%

HWHM: 6.6 GeV