

# Track Seeding Example for ACTS (A Common Tracking Software)

**Peter Chatain**

**Mentors: Dr. Rocky Bala Garg & Prof. Lauren Tompkins**

August 28<sup>th</sup>, 2020



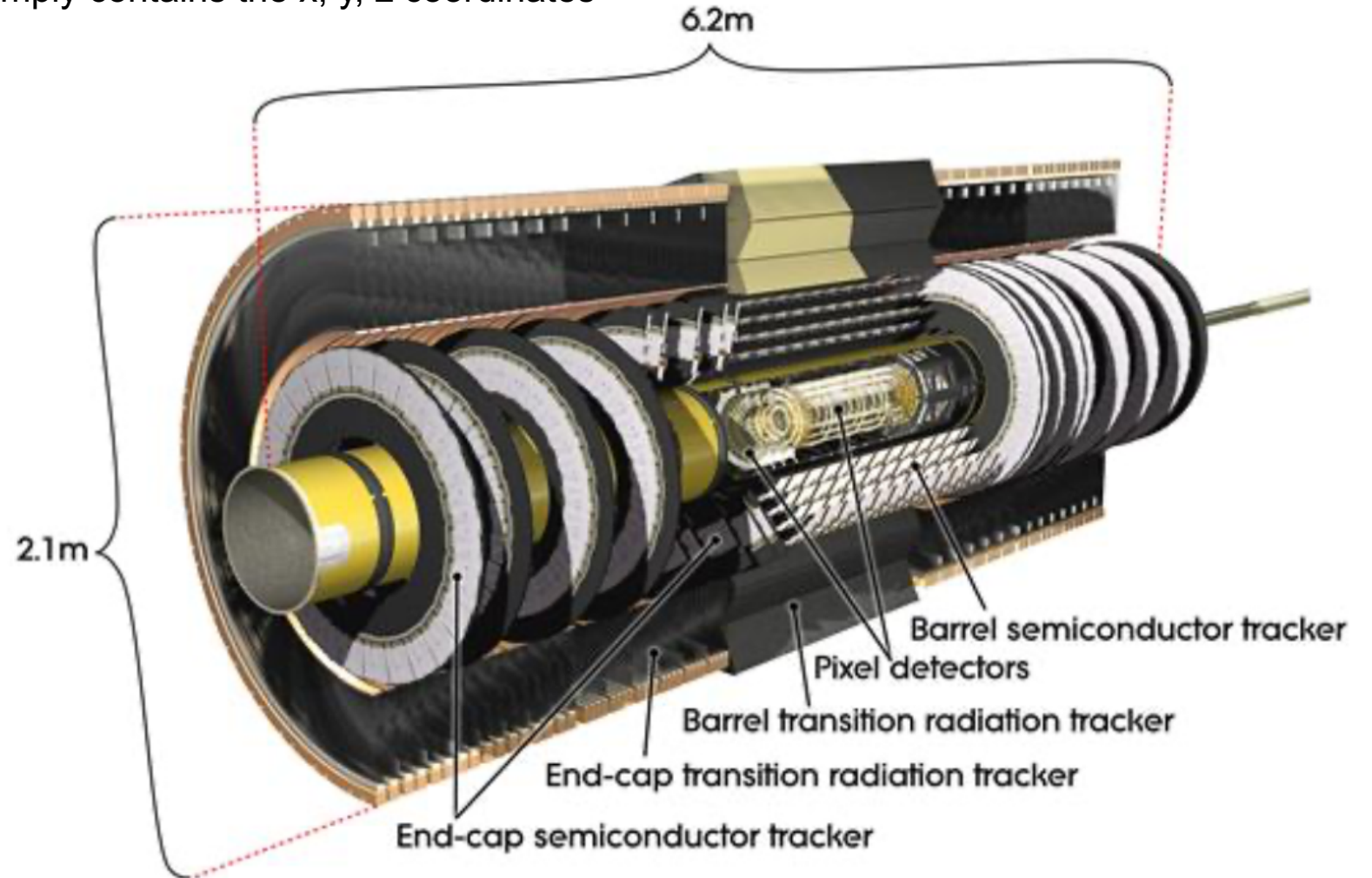
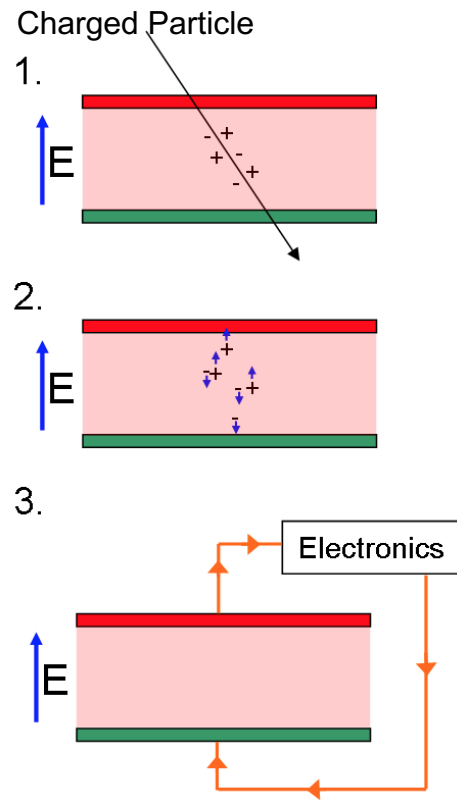
**My Project was to code a Track Seeding Example  
for the open source project ACTS**

- **To best explain, I will briefly review tracking and  
the goals of ACTS**



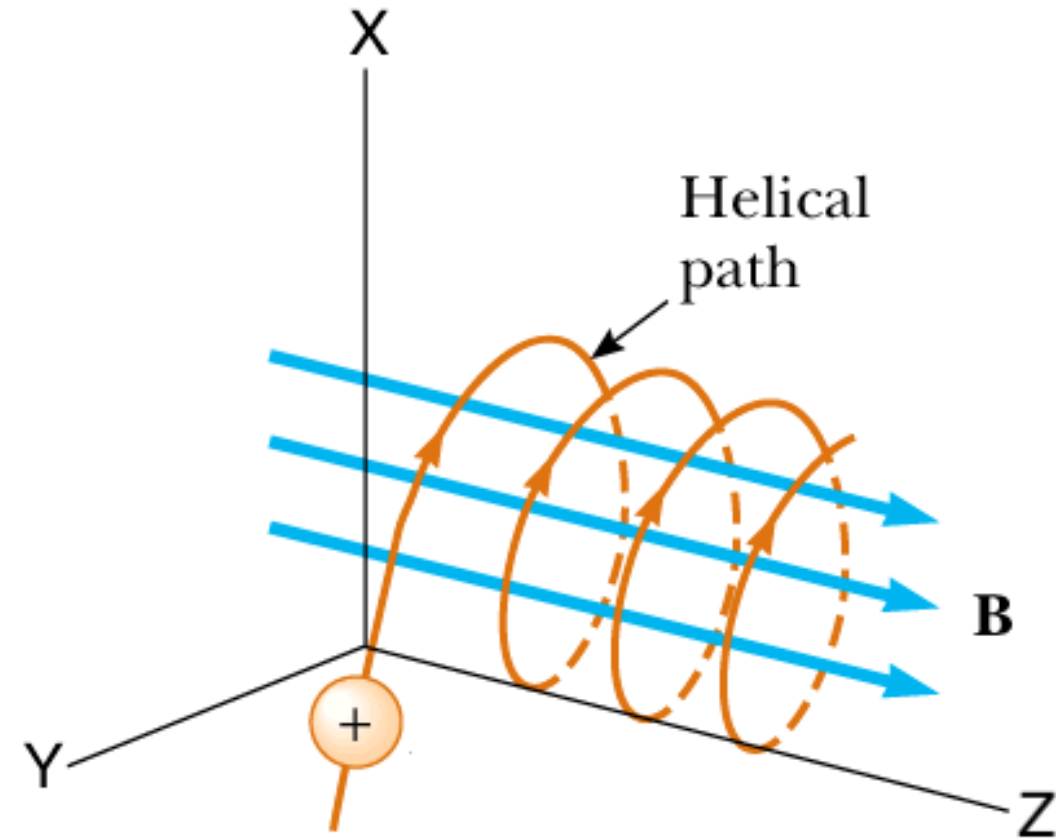
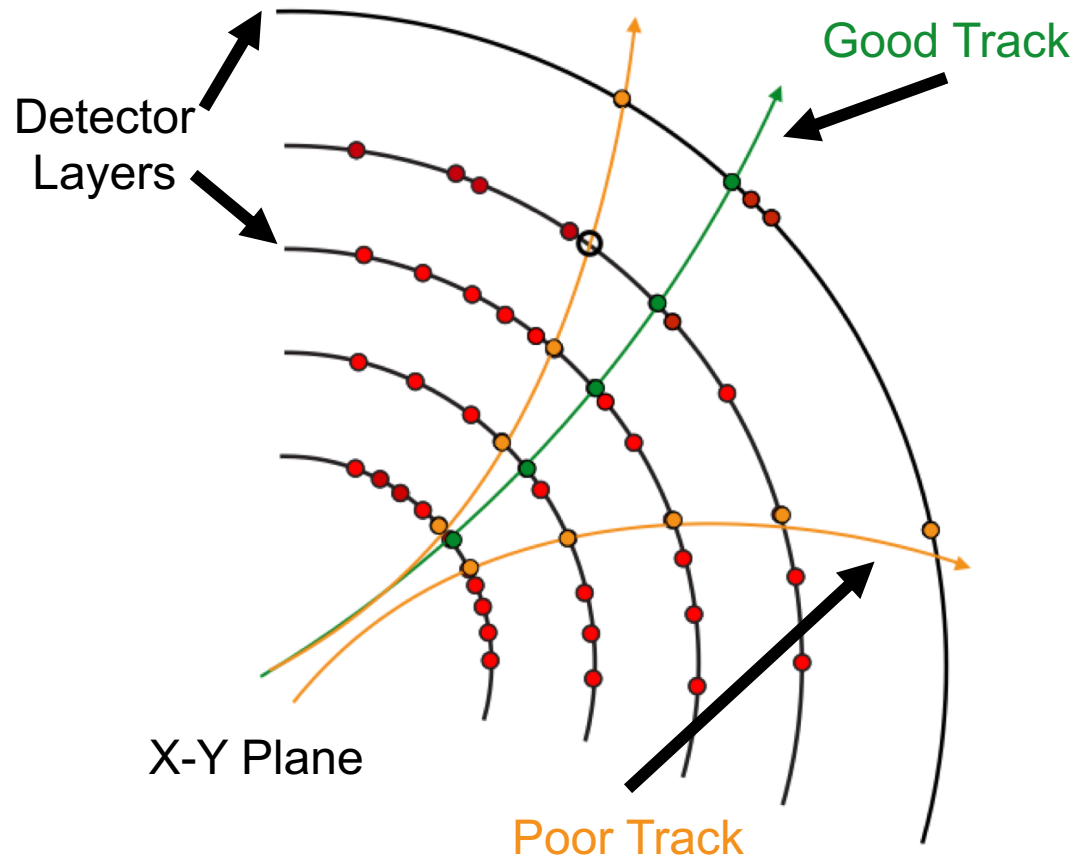
# ATLAS Detector

- **Particles interact with the detector, creating a “hit” or “Space Point”**
  - For the innermost detector layers, a hit simply contains the x, y, z coordinates



# What is Tracking?

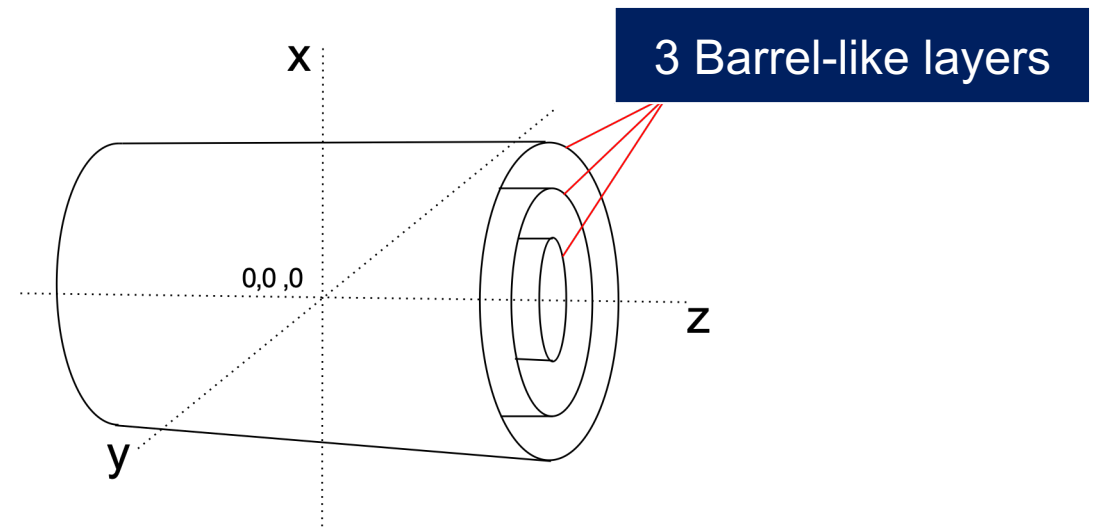
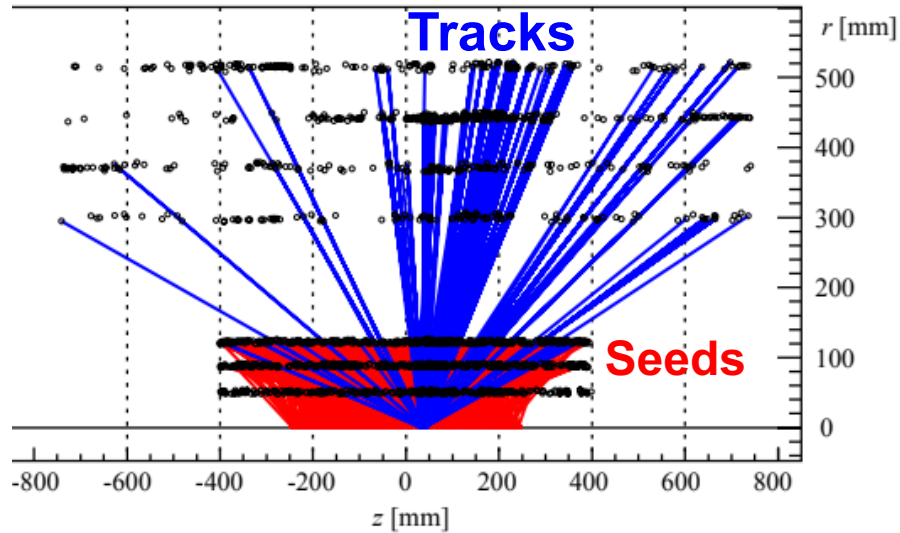
- Tracking is the process taking hits to reconstructed particle trajectories





# What is Track Seeding?

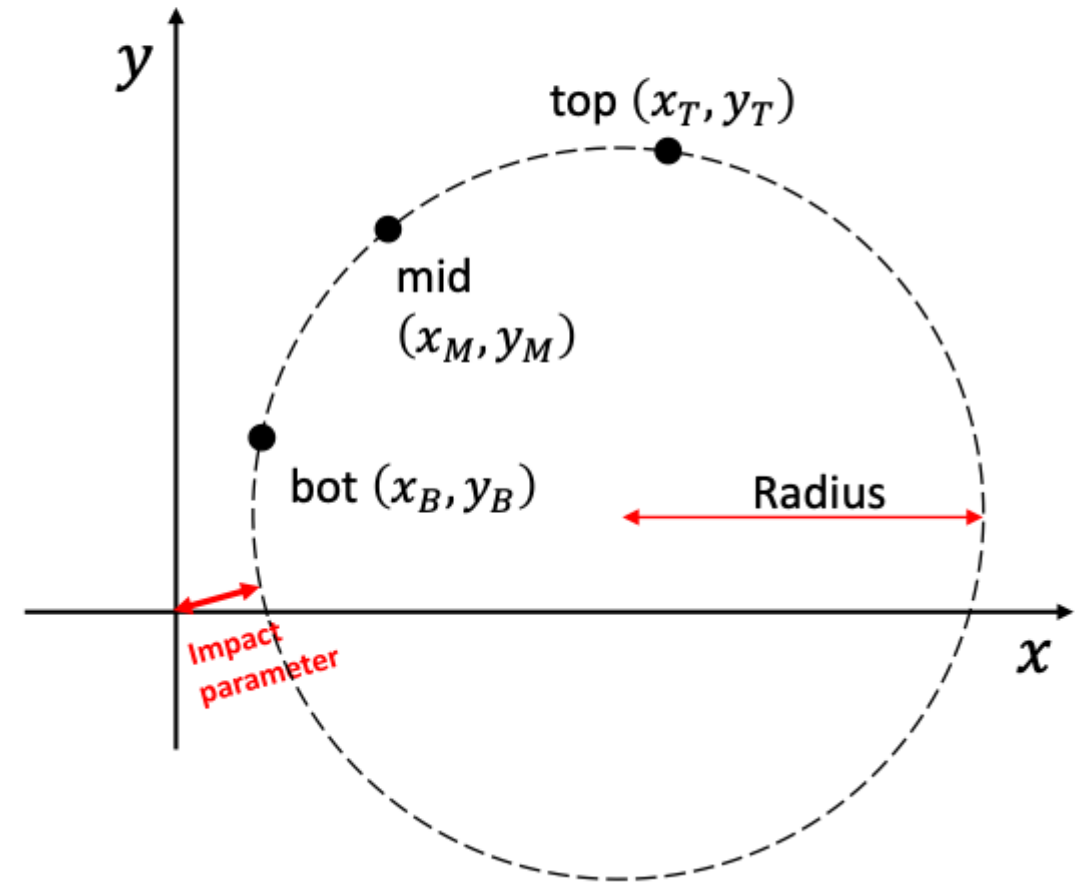
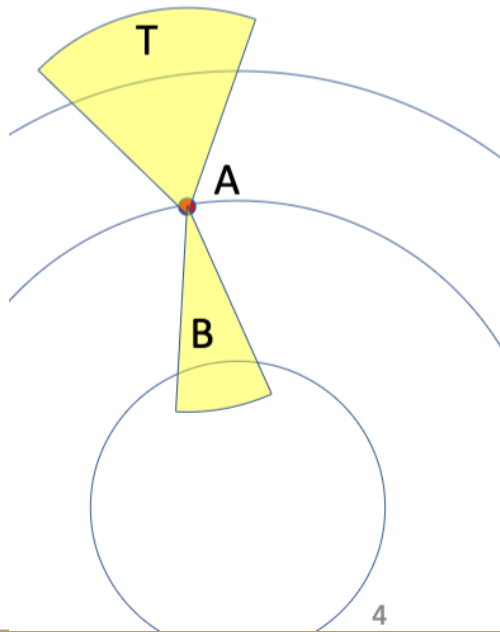
- One strategy for tracking is to break it into two stages: Track Seeding and Track following
- Seeds are tracks with only 3 hits
  - Initial estimates of tracks
  - Can be many fake seeds



# How Does the Seed Finding Algorithm Work?

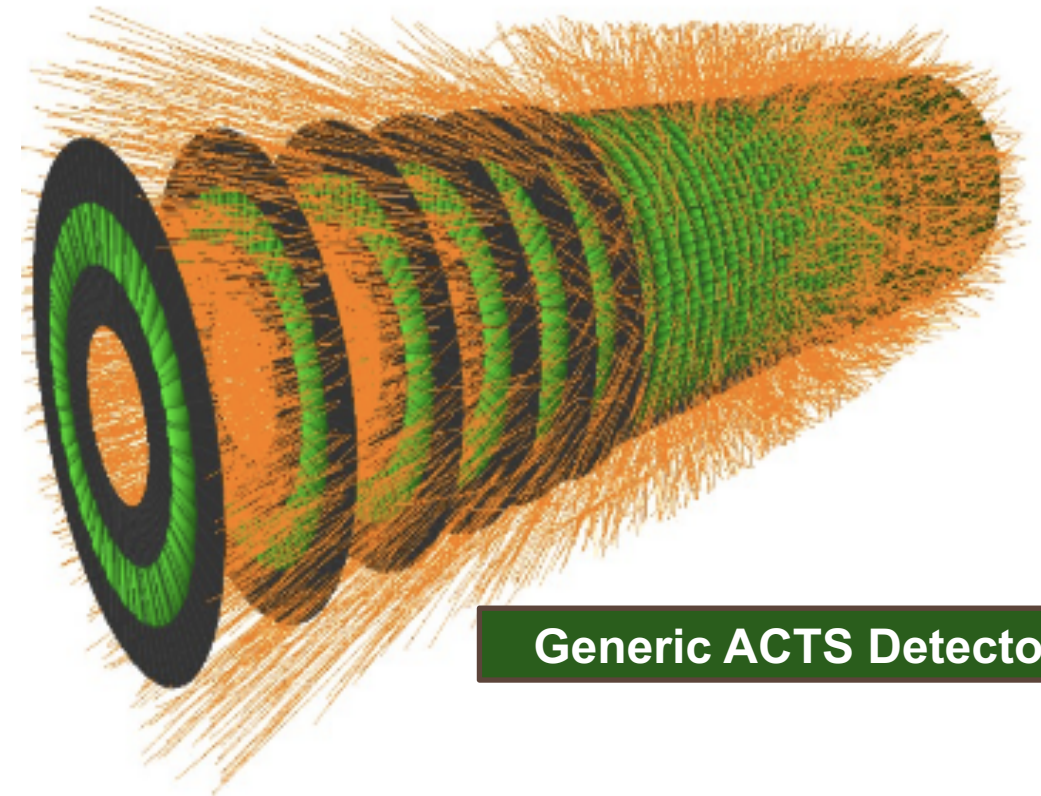
- Groups hits into 3 bins – Top, Middle, and Bottom
- Create seeds within search window
- Search complexity grows exponentially with more data points
  - Check for compatibility with a helix trajectory
  - Filter Seeds according to parameters (e.g. Impact Parameter)
  - Seeds ranked according to a score

Search window per middle space point “A”



# Why Do We Need Faster Algorithms?

- The HL-LHC is planning an increase in data production per event by a factor of 5-7
- This creates a problem:
  - Compute time explodes exponentially with more data.



Generic ACTS Detector

This is an example of 1 event, each orange line represents the trajectory of a charged particle

# Goals of ACTS (A Common Tracking Software)

- **Clear, maintainable code to help develop faster algorithms**
  - Thread safe code, parallelization and modern C++ advantages such as const correctness



# Goals of ACTS (A Common Tracking Software)

- **Clear, maintainable code to help develop faster algorithms**
  - Thread safe code, parallelization and modern C++ advantages such as const correctness
- **Allow for many detectors around the world to share these advances**
  - Challenging, because Algorithms often depend on specific detector geometry



# Goals of ACTS (A Common Tracking Software)

- **Clear, maintainable code to help develop faster algorithms**
  - Thread safe code, parallelization and modern C++ advantages such as const correctness
- **Allow for many detectors around the world to share these advances**
  - Challenging, because Algorithms often depend on specific detector geometry
- **Examples allow for testing and optimization of individual components of tracking**
  - E.g. track seeding



# Goals of ACTS (A Common Tracking Software)

- **Clear, maintainable code to help develop faster algorithms**
  - Thread safe code, parallelization and modern C++ advantages such as const correctness
- **Allow for many detectors around the world to share these advances**
  - Challenging, because Algorithms often depend on specific detector geometry
- **Examples allow for testing and optimization of individual components of tracking**
  - E.g. track seeding
- **Previously, the Tracking Example did not use the seed finding algorithm**
  - Used truth information
  - My project allows generating seeds using the seed finding algorithm





# Track Seeding Example

- **My project was to create the track seeding example**
  - Command line tools to test track seeding algorithms and write performance plots

```
bash-4.2$ ./ActsExampleTestSeedAlgorithm --input-dir sim-pythia8/ --output-dir atlasSeedPerf/
```



# Track Seeding Example

- **My project was to create the track seeding example**
  - Command line tools to test track seeding algorithms and write performance plots

```
bash-4.2$ ./ActsExampleTestSeedAlgorithm --input-dir sim-pythia8/ --output-dir atlasSeedPerf/
TrackSeeding INFO Fake rate (nSeeds - nTrueSeeds) / nSeeds --- 44%
TrackSeeding INFO Duplicate rate (nDuplicateSeeds / nSeeds) --- 31%
TrackSeeding INFO Efficiency 1659/1691 = 98%
TrackSeeding INFO Wrote seed finder performance trees to /gpfs/slac/
```



# Track Seeding Example

- My project was to create the track seeding example
  - Command line tools to test track seeding algorithms and write performance plots

```
bash-4.2$ ./ActsExampleTestSeedAlgorithm --input-dir sim-pythia8/ --output-dir atlasSeedPerf/
TrackSeeding INFO Fake rate (nSeeds - nTrueSeeds) / nSeeds --- 44%
TrackSeeding INFO Duplicate rate (nDuplicateSeeds / nSeeds) --- 31%
TrackSeeding INFO Efficiency 1659/1691 = 98%
TrackSeeding INFO Wrote seed finder performance trees to /gpfs/slac/
```

- **Efficiency = fraction of particles found by at least one seed**
  - Very important for track seeding



# Track Seeding Example

- **My project was to create the track seeding example**
  - Command line tools to test track seeding algorithms and write performance plots

```
bash-4.2$ ./ActsExampleTestSeedAlgorithm --input-dir sim-pythia8/ --output-dir atlasSeedPerf/
TrackSeeding INFO Fake rate (nSeeds - nTrueSeeds) / nSeeds --- 44%
TrackSeeding INFO Duplicate rate (nDuplicateSeeds / nSeeds) --- 31%
TrackSeeding INFO Efficiency 1659/1691 = 98%
TrackSeeding INFO Wrote seed finder performance trees to /gpfs/slac/
```

- **Efficiency = fraction of particles found by at least one seed**
  - Very important for track seeding
- **Fake rate = fraction of seeds that don't correspond to a real particle trajectory**
  - Important to keep low for computational efficiency



# Track Seeding Example

- **My project was to create the track seeding example**
  - Command line tools to test track seeding algorithms and write performance plots

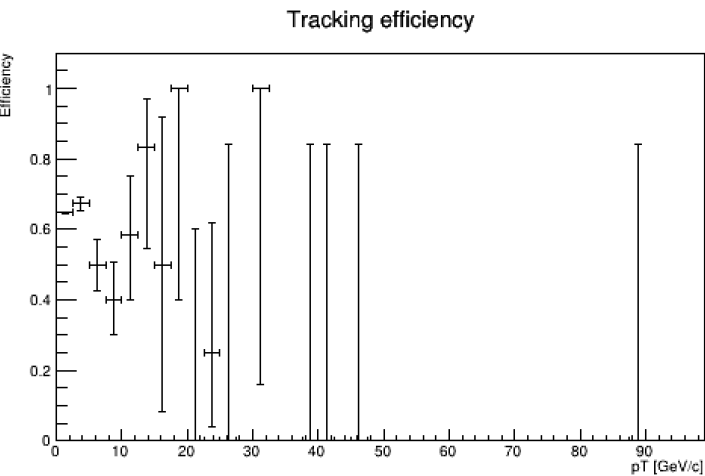
```
bash-4.2$ ./ActsExampleTestSeedAlgorithm --input-dir sim-pythia8/ --output-dir atlasSeedPerf/
TrackSeeding INFO Fake rate (nSeeds - nTrueSeeds) / nSeeds --- 44%
TrackSeeding INFO Duplicate rate (nDuplicateSeeds / nSeeds) --- 31%
TrackSeeding INFO Efficiency 1659/1691 = 98%
TrackSeeding INFO Wrote seed finder performance trees to /gpfs/slac/
```

- **Efficiency = fraction of particles found by at least one seed**
  - Very important for track seeding
- **Fake rate = fraction of seeds that don't correspond to a real particle trajectory**
  - Important to keep low for computational efficiency
- **Duplicate rate fraction of true seeds that re-identify a particle**
  - Important to keep low for computational efficiency

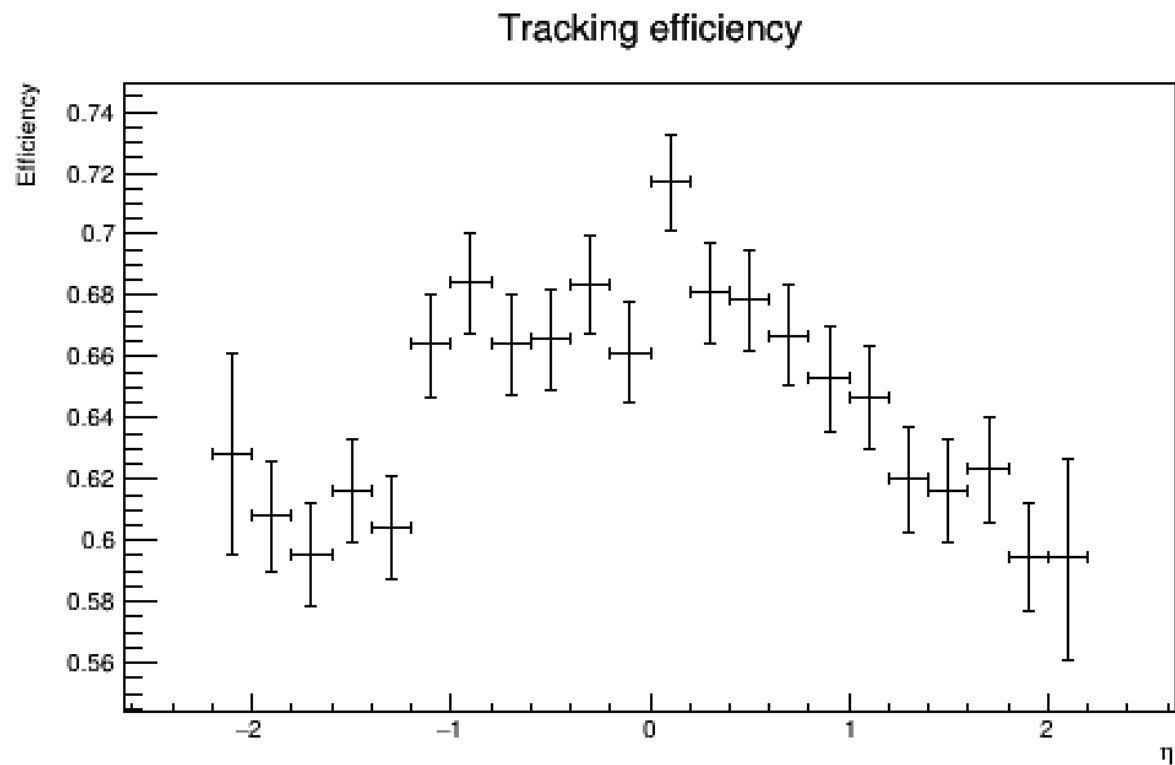
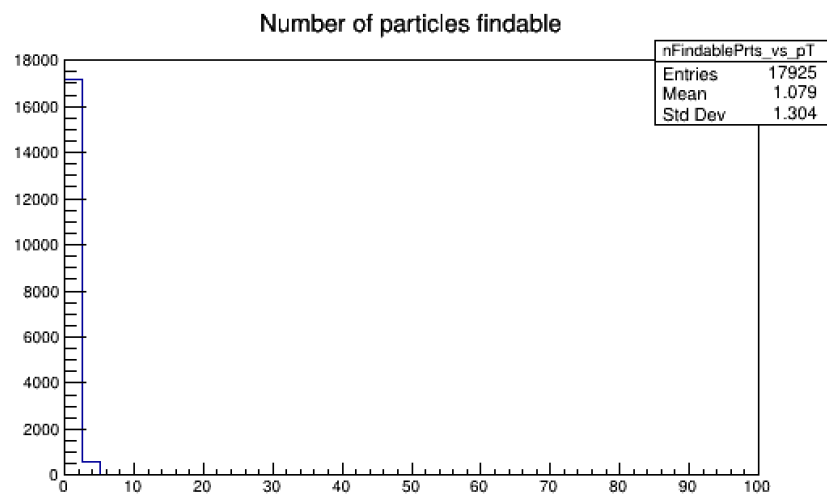


# Performance Without Optimizations

- On ttbar sample with pileup 200 and pT range from 0.1 to 100 &  $|\eta| < 2.5$

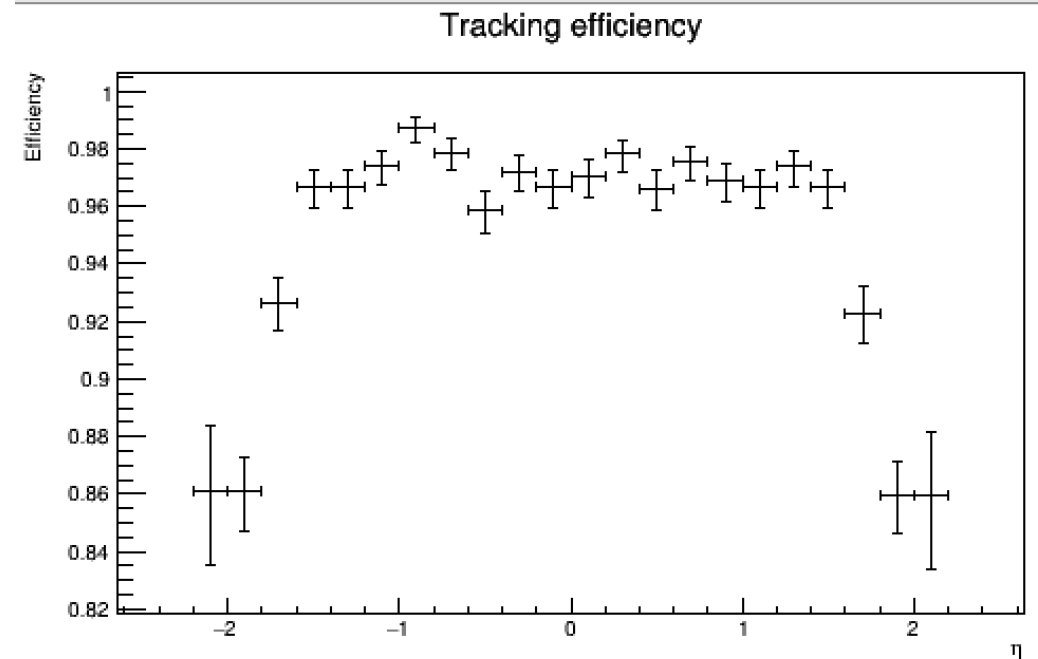
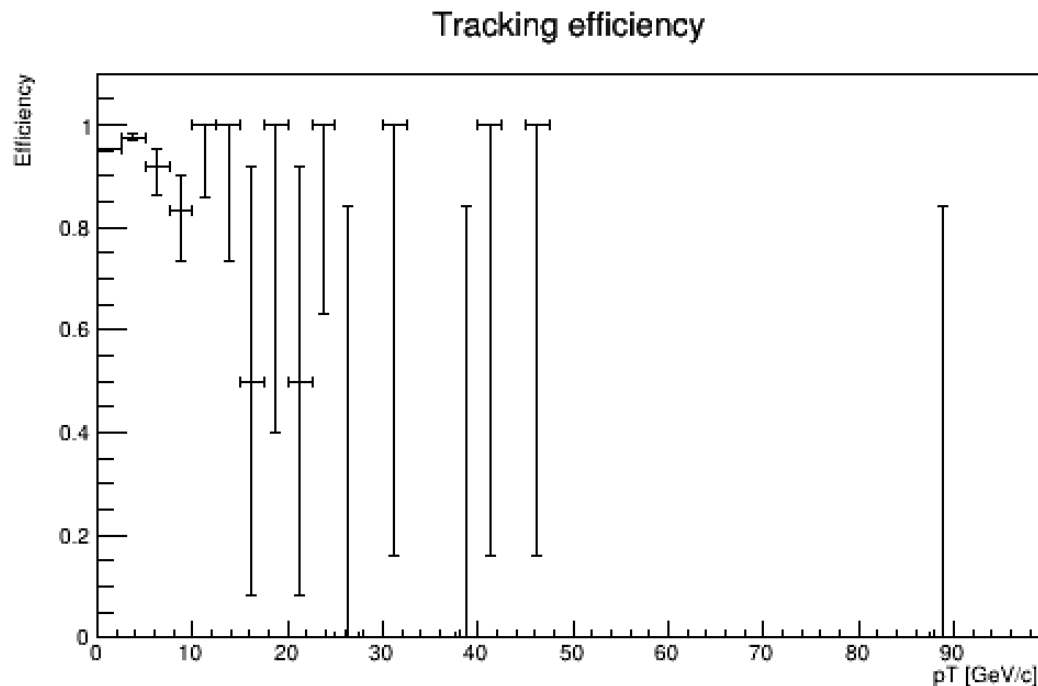


Large error bars  
due to very few  
high pT particles



# Performance With Optimizations

- On ttbar sample with pileup 200 and pT range from 0.1 to 100 &  $|\eta| < 2.5$
- Beam position moved from (-.5, -.5) to (0,0)
- Max radius adjusted to utilize pixel detector
- Max Impact parameter reduced
- Sigma Scattering increased to include 2.25 sigma
- Delta R Min & Delta R Max reduced

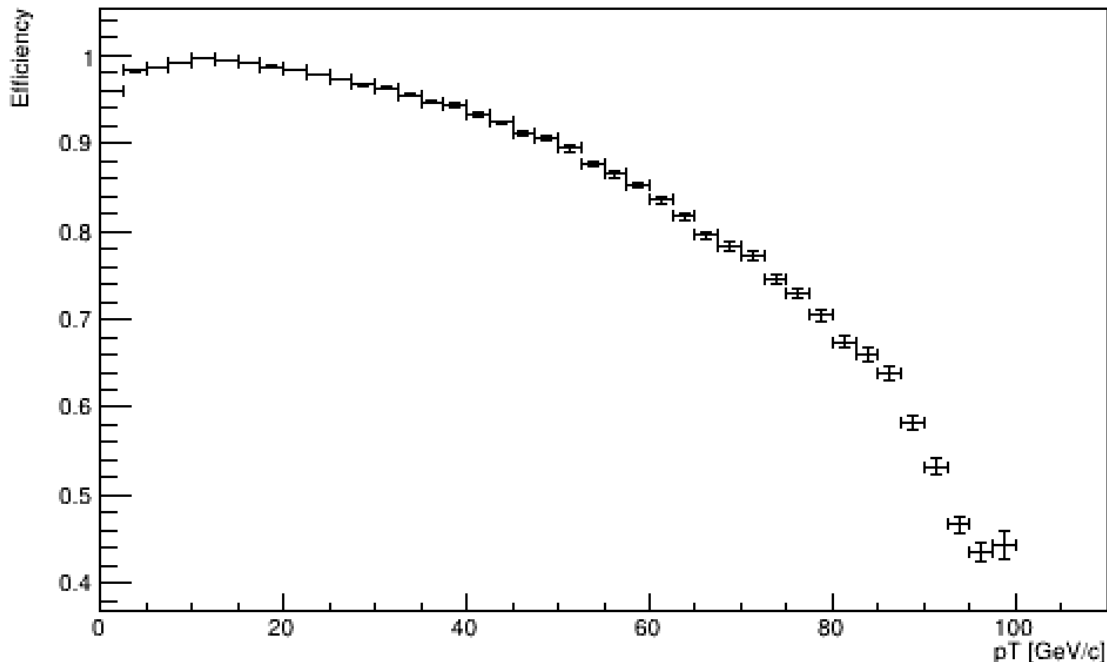




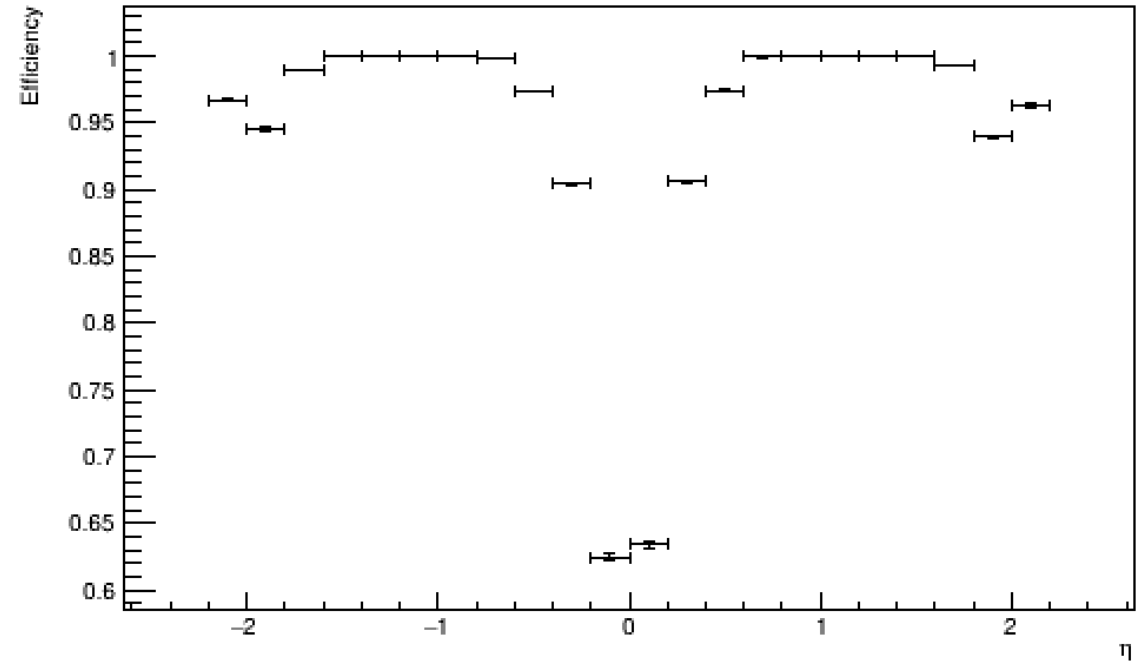
# Performance With optimizations

- **Single Muon 100 events with 10,000 particles per event**
  - A lot of data since error bars were large – not enough higher pT particles
- **We expect better performance for higher pT and close to  $\eta = 0$** 
  - Opposite trend initially found
  - We also expect better overall performance on single muon than ttbar

Tracking efficiency



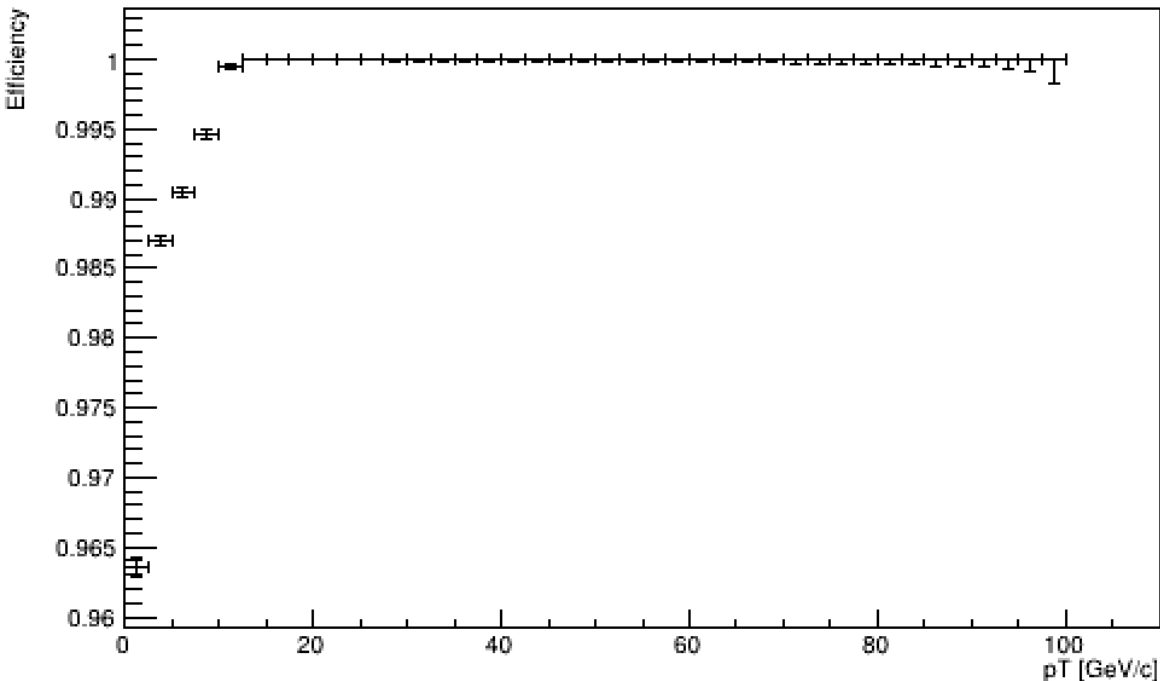
Tracking efficiency



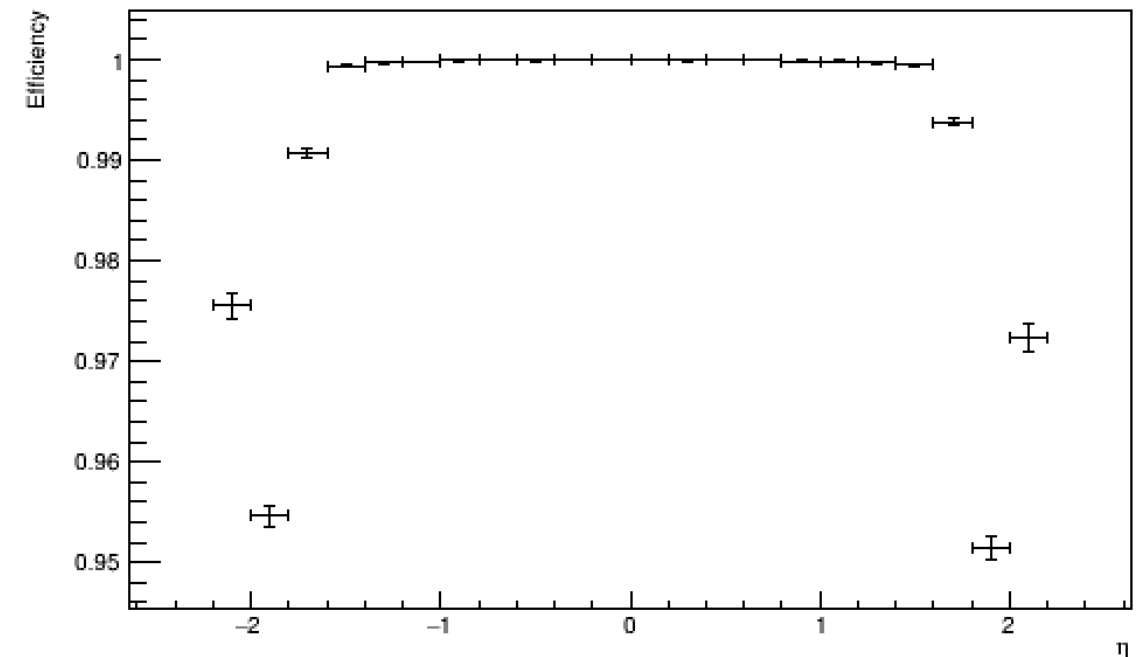
# Performance After Removing a Particular Seed Quality Cut

- Removed multiple scattering angle requirement as it was not optimized for this geometry
- We get expected efficiency, but too many seeds generated (~40,000 seeds from ~40,000 hits)
  - Multiple scattering angle requirement needs tuning
  - Configurable parameters will need more tuning

Tracking efficiency



Tracking efficiency



# Specific Work on Seeding Algorithm

- **Wrote a function to convert hit data into Space Points**
  - [readSP](#)
- **Ran seeding algorithm on Space Points**
- **Converted seeds into Proto tracks**
  - [seedToProtoTrack](#)
  - Performance tools for tracking and track fitting utilize proto tracks
  - Stored for later analysis
- **Code located in**
  - <https://github.com/Pchatain/acts/tree/master/Examples/Run/TestSeeding>



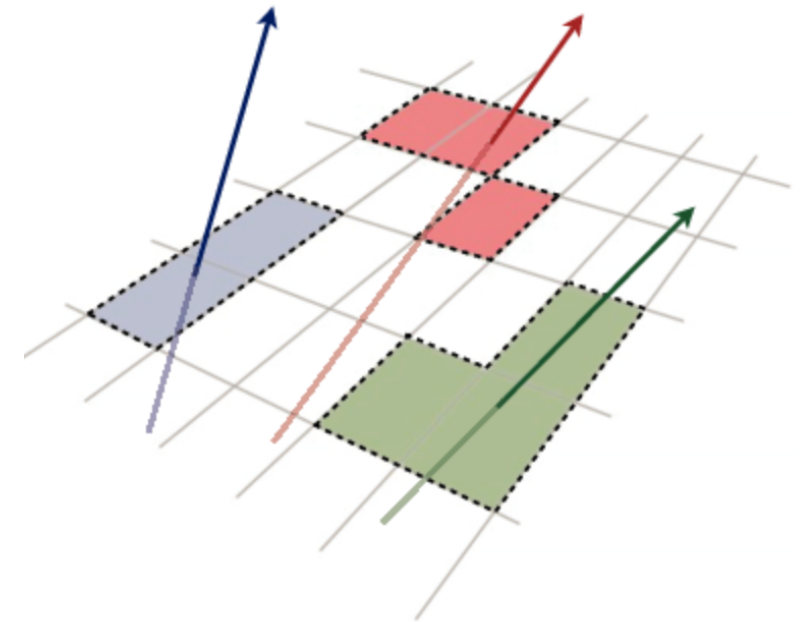
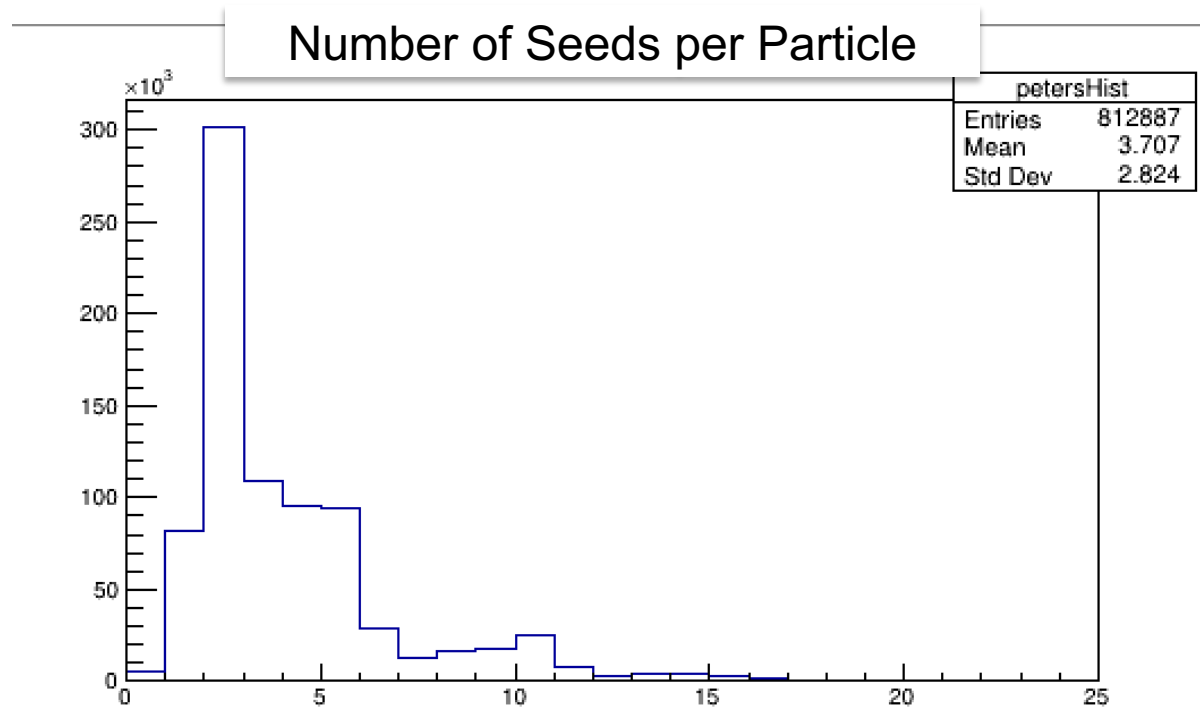
# Specific Work on Performance Analysis

- **Wrote a seeding performance writer**
  - [identifySharedParticles](#) categorizes seeds as true seeds or fake
  - [prtFindable](#) filters the particles
    - We don't expect seeding to find all particles
  - [analyzeSeed](#) records metrics for the seed
  - Outputs performance plots for efficiency
  - Prints out metrics such as fake rate, duplicate rate, efficiency before and after the filter
- **Code located at**
  - <https://github.com/Pchatain/acts/blob/master/Examples/lo/Performance/ACTFW/lo/Performance/TrackSeedingPerformanceWriter.cpp> and the [header](#)
- **Currently undergoing code review with Ai, Xiaocong to add my example to ACTS on GitHub**
  - Written with const correctness
  - Working for 2 more weeks



# Further Work

- **Clustering hits before reading into space points**
  - Will reduce duplicate seeds
- **Integration for histogram plotting of fake rate**
- **Optimization pipeline development**
  - Get configurable parameters outside so that they're read into the program instead of hardcoded magic numbers
  - Specific to detector geometry



# Acknowledgements

- My amazing summer mentors: Rocky Bala Garg and Lauren Tompkins
- Helpful ACTS Contributors
  - Ai Xiaocong and Robert Langenberg
- Questions?

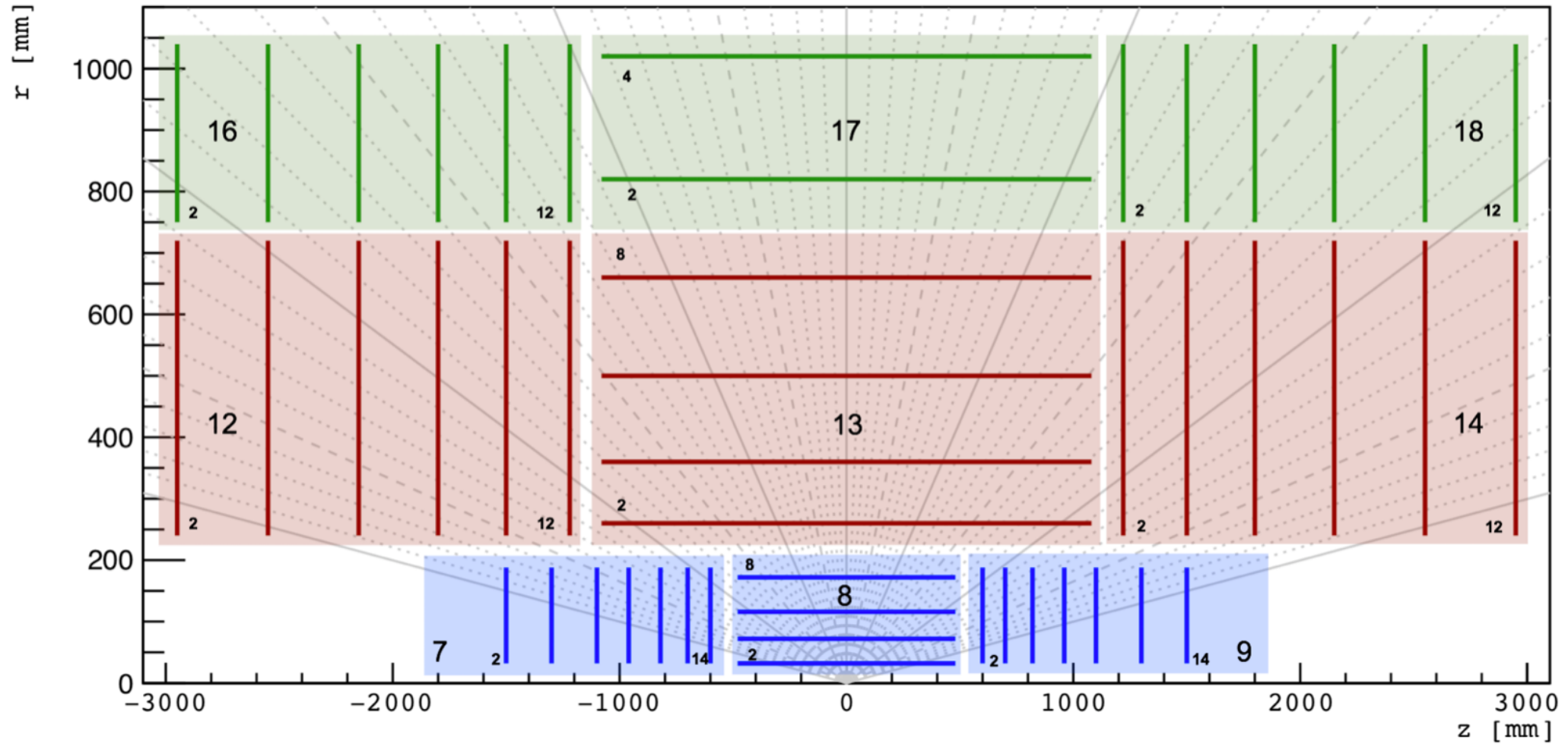


iris  
hep

Institute for Research & Innovation  
in Software for High Energy Physics

# Generic ACTS Detector

- Seeding plots generated by using all pixel layers





# Configurable Parameters

- **Parameters before:**
  - `config.rMax = 160.;`
  - `config.deltaRMin = 5.;`
  - `config.deltaRMax = 160.;`
  - `config.collisionRegionMin = -250.;`
  - `config.collisionRegionMax = 250.;`
  - `config.zMin = -2800.;`
  - `config.zMax = 2800.;`
  - `config.maxSeedsPerSpM = 5;`
  - `// 2.7 eta`
  - `config.cotThetaMax = 7.40627;`
  - `config.sigmaScattering = 1;`  
`config.minPt = 500.;`
  - `config.bFieldInZ = 0.00199724;`  
`config.beamPos = {-0.5, -0.5};`
  - `config.impactMax = 10.;`
- **Parameters after:**
  - `config.rMax = 200.;` // utilized entire inner pixel volume
  - `config.deltaRMin = 1.;` // increased duplicate seeds, and increased efficiency
  - `config.deltaRMax = 80.;` // reduced duplicates with less than 0.5% reduction in efficiency
  - `config.collisionRegionMin = -250.;`
  - `config.collisionRegionMax = 250.;`
  - `config.zMin = -2000.;` // Hits only range -2000 to 2000 for pixel layers
  - `config.zMax = 2000.;`
  - `config.maxSeedsPerSpM = 3;` // seedFilter.ipp adds 1
  - `// 2.7 eta`
  - `config.cotThetaMax = 7.40627;`
  - `config.sigmaScattering = 2.25;` // higher efficiency but more duplicates and compute time
  - `config.minPt = 500.;`
  - `config.bFieldInZ = 0.00199724;`
  - `config.beamPos = {0, 0};` // better performance all around than {-0.5, -0.5}
  - `config.impactMax = 3.;` reduced duplicates and fakes

# Performance on ttbar sample

- **Before optimization**

- Time to create seeds: 0.81739s
- Number of regions: 260
- Number of hits used is: 20443 --- 37% usage
- Number of seeds generated: 4423
- Number of true seeds generated: 1434
- Fake rate  $(n\text{Seeds} - n\text{TrueSeeds}) / n\text{Seeds}$  --- 67%
- Number of duplicate seeds generated: 203
- Technical Efficiency  $(n\text{TrueSeeds} - n\text{DuplicateSeeds} / n\text{Seeds})$  --- 27%
- Duplicate rate  $(n\text{DuplicateSeeds} / n\text{Seeds})$  --- 4%
- Raw Efficiency: (particles matched to truth) / nParticles =  $1231 / 4740 = 25\%$
- Efficiency (taking into account whether we expect the particle to be found)  $1156/1691 = 68\%$

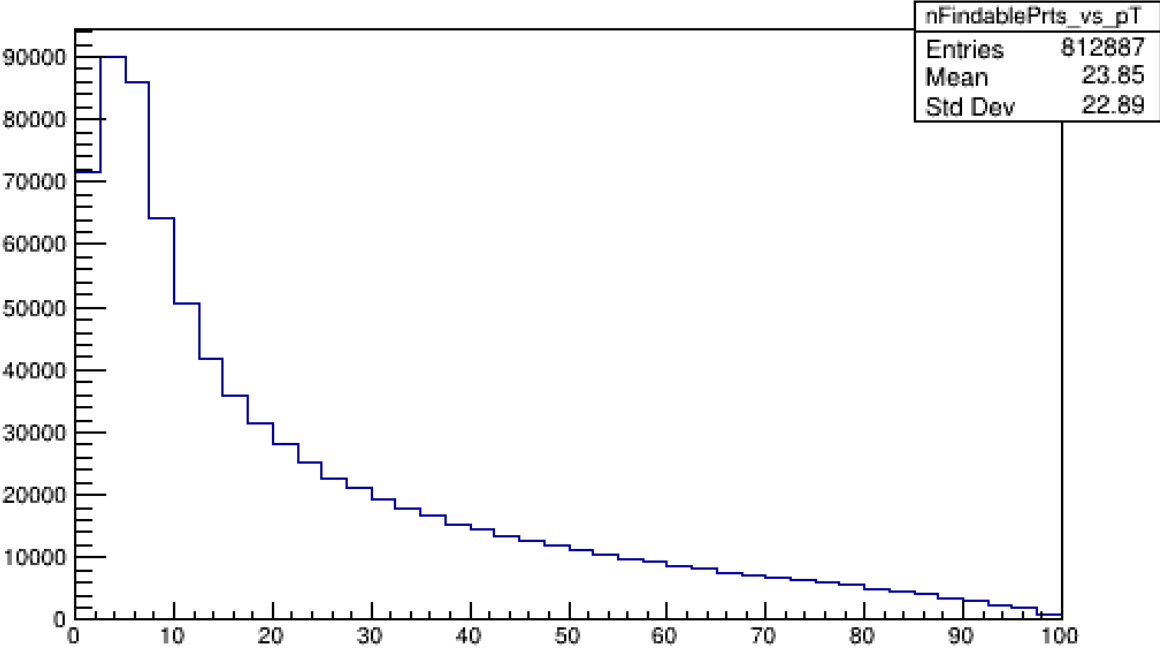
Still has seed cut on error not factoring in pT problem

- **After my preliminary Optimization**

- Time to create seeds: 0.277339s
- Number of regions: 1557
- Number of hits used is: 20443 --- 37% usage out of whole detector
- Number of seeds generated: 8108
- Number of true seeds generated: 4501
- Fake rate  $(n\text{Seeds} - n\text{TrueSeeds}) / n\text{Seeds}$  --- 44%
  - Mainly due to more duplicate seeds
- Number of duplicate seeds generated: 2528
- Technical Efficiency  $(n\text{TrueSeeds} - n\text{DuplicateSeeds} / n\text{Seeds})$  --- 24%
- Duplicate rate  $(n\text{DuplicateSeeds} / n\text{Seeds})$  --- 31%
- Raw Efficiency: (particles matched to truth) / nParticles =  $1973 / 4740 = 41\%$
- Efficiency (taking into account whether we expect the particle to be found)  $1659/1691 = 98\%$

# Denominator Plots for Seeding Efficiency on Single Muon

Number of particles findable



Number of particles findable

