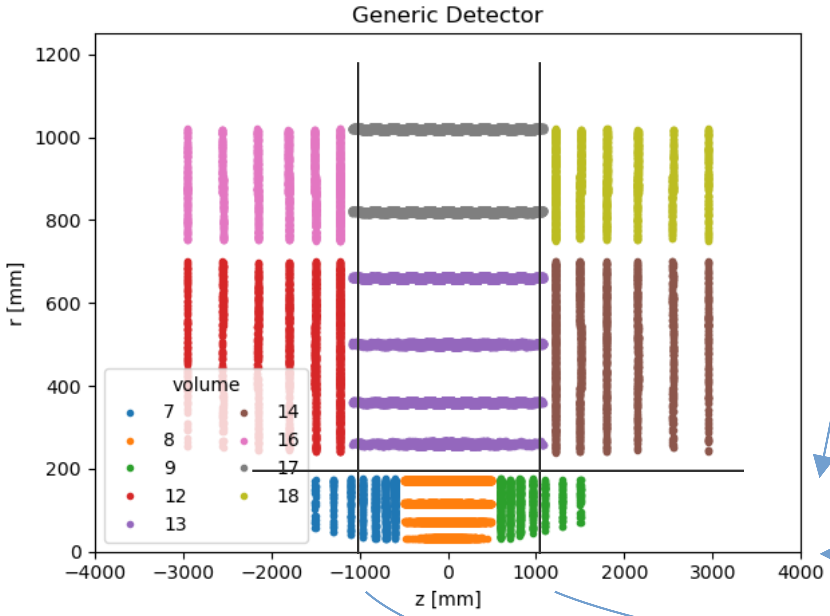
# Tracking with Hashing Overview

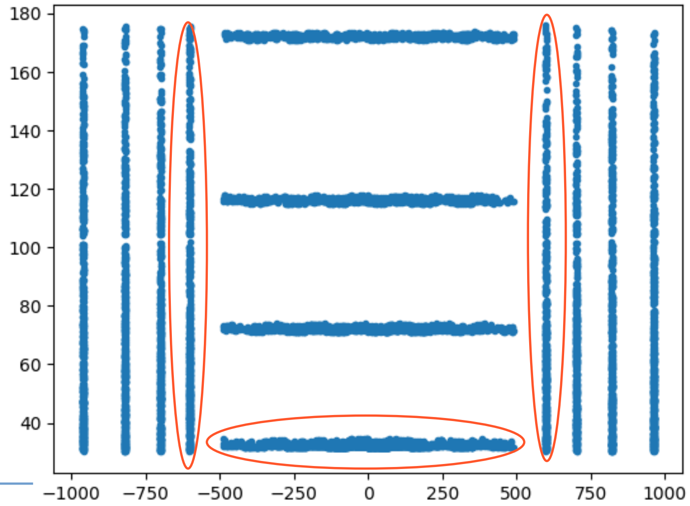
Jeremy Couthures

# Setup

# Generic Detector: Space Points




Space points position:



|η| <= 4

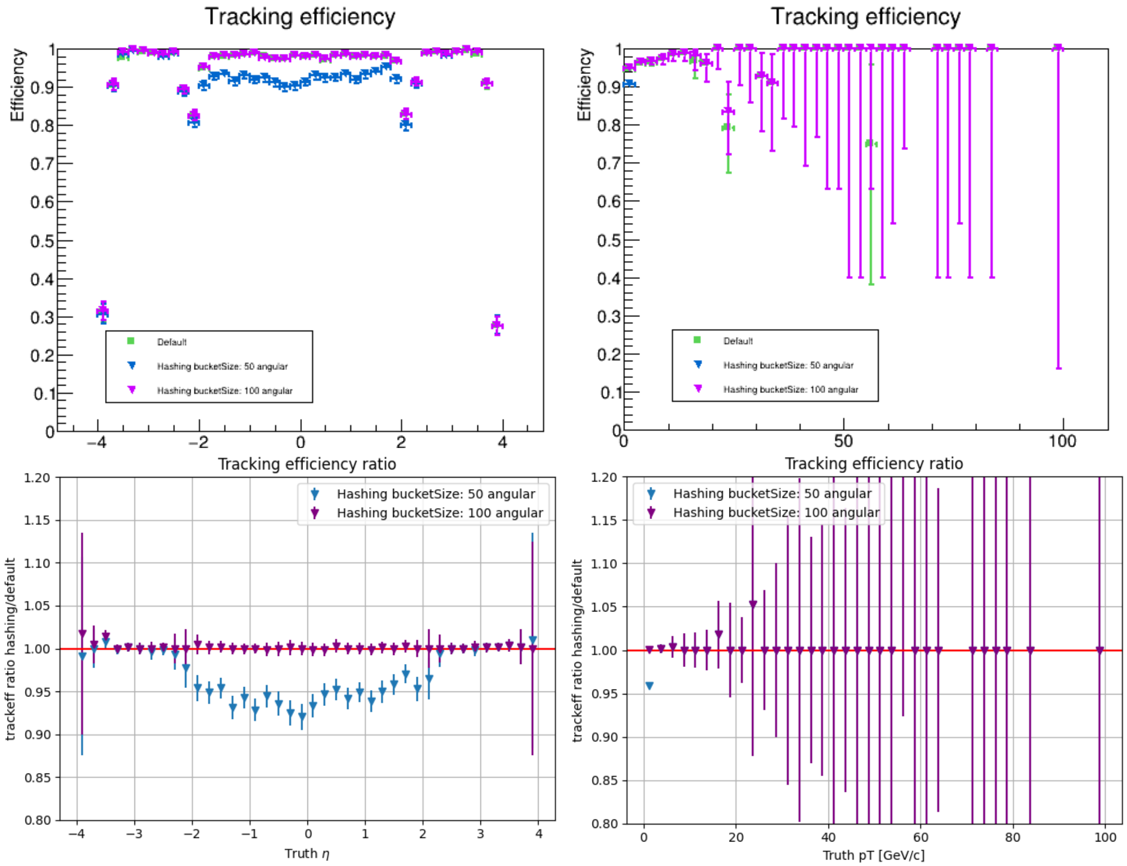
➡ Hashing currently uses only Pixel Space Points
Buckets are built from Space Points of layers 0

# Seed finder configuration

```
SeedfinderConfigArg = SeedfinderConfigArg(
        r=(None, 200 * u.mm),  # rMin=default, 33mm
        deltaR=(1 * u.mm, 60 * u.mm),
        collisionRegion=(-250 * u.mm, 250 * u.mm),
        z=(-2000 * u.mm, 2000 * u.mm),
        maxSeedsPerSpM=1,
        sigmaScattering=5,
        radLengthPerSeed=0.1,
        minPt=500 * u.MeV,
        bFieldInZ=1.99724 * u.T,
        impactMax=3 * u.mm,
        cotThetaMax=cotThetaMax # =1/tan(2×atan(e^(-eta)))
    )
```

# Bucket Size

# Performance μ = 50 Δφ metric



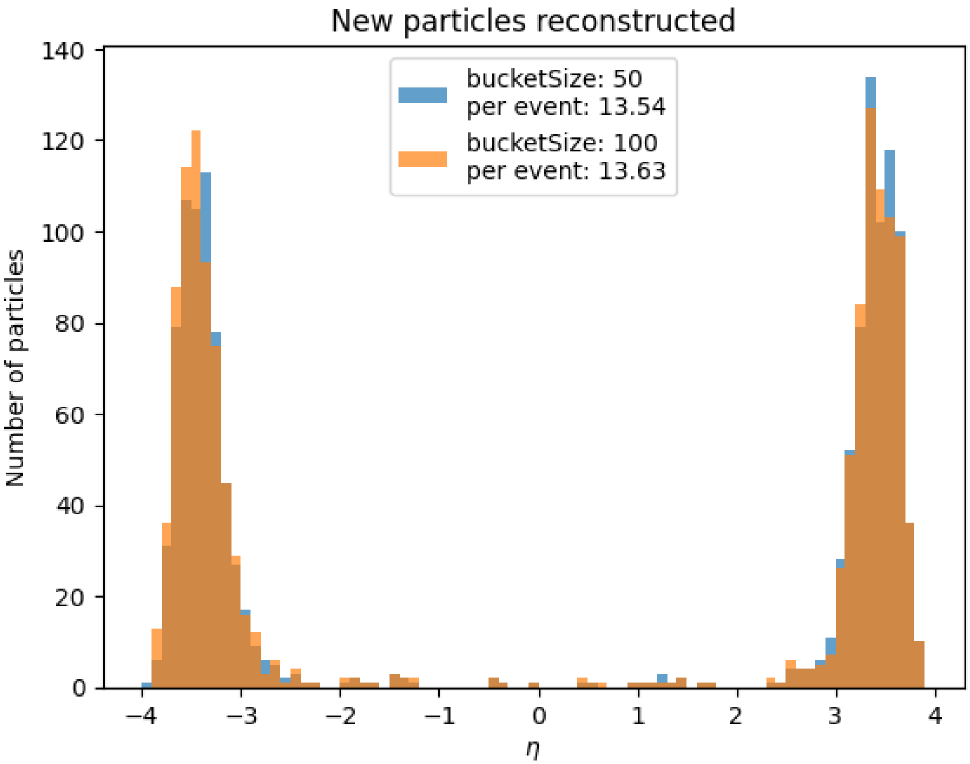

Bucket size 50:
low pT are not well reconstructed

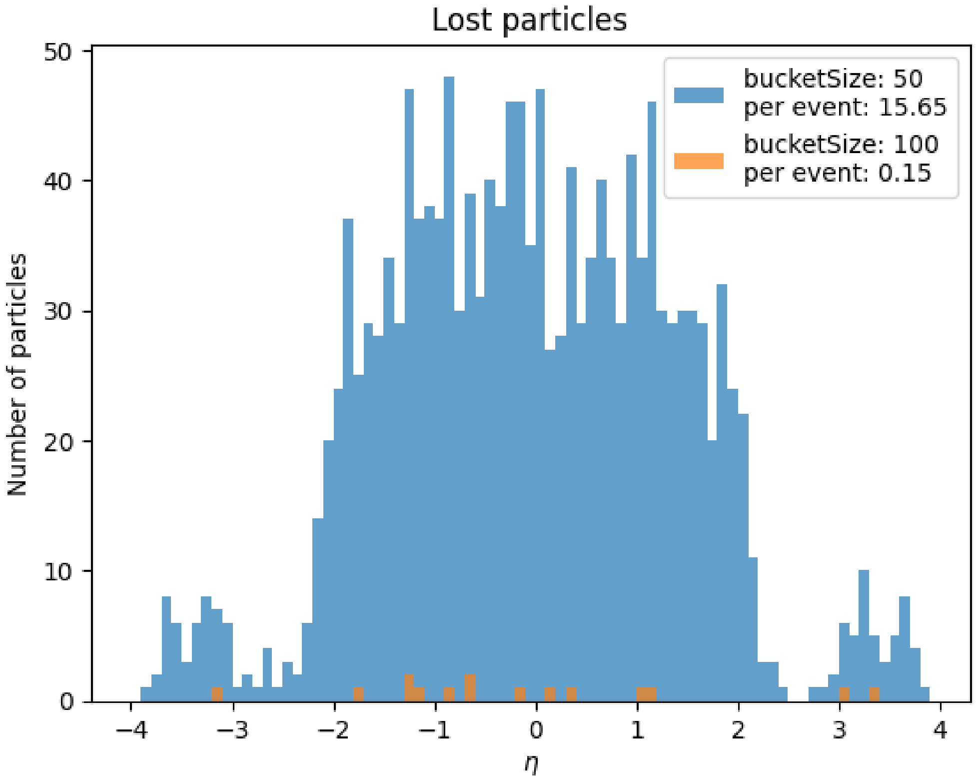
Loss of efficiency in the central region
Better efficiency in the forward region

Bucket size 100:
low pT are well reconstructed

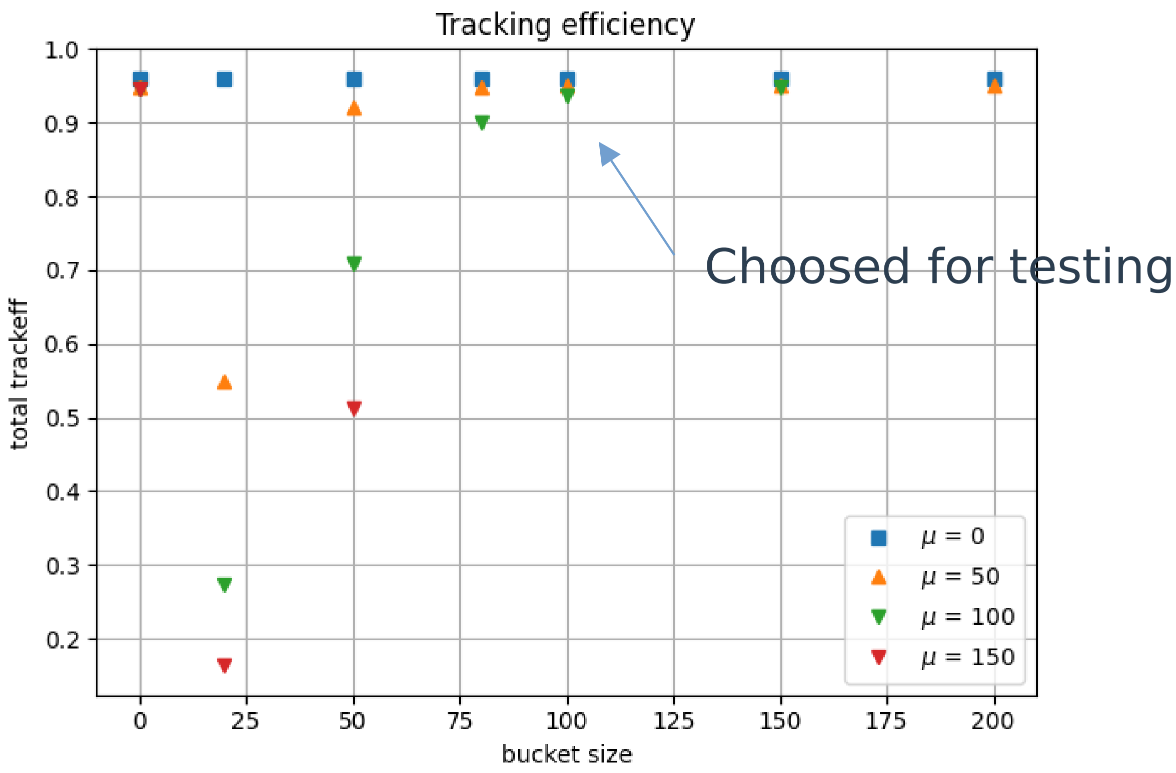
Better efficiency in the central region
Better efficiency in the forward region

# Bucket size effect → Where?



# Optimal bucket size



# Bucket Size Conclusion

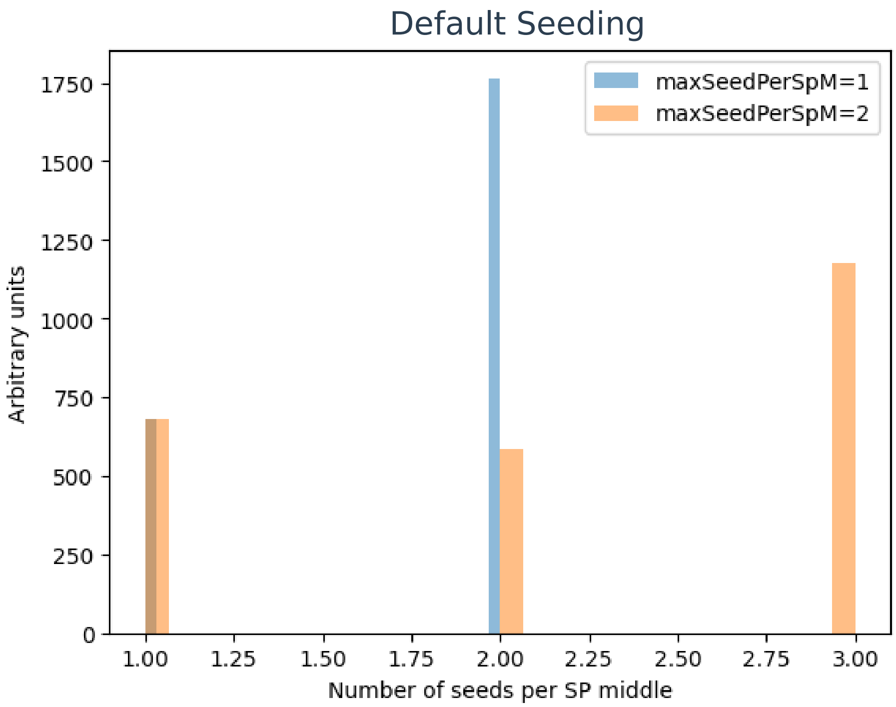
- **Hashing can improve physics performance**
- **The bucket size has a high impact on the performance**
- **New particles are in the forward region**
- **A bucket size of 100 is used for testing**

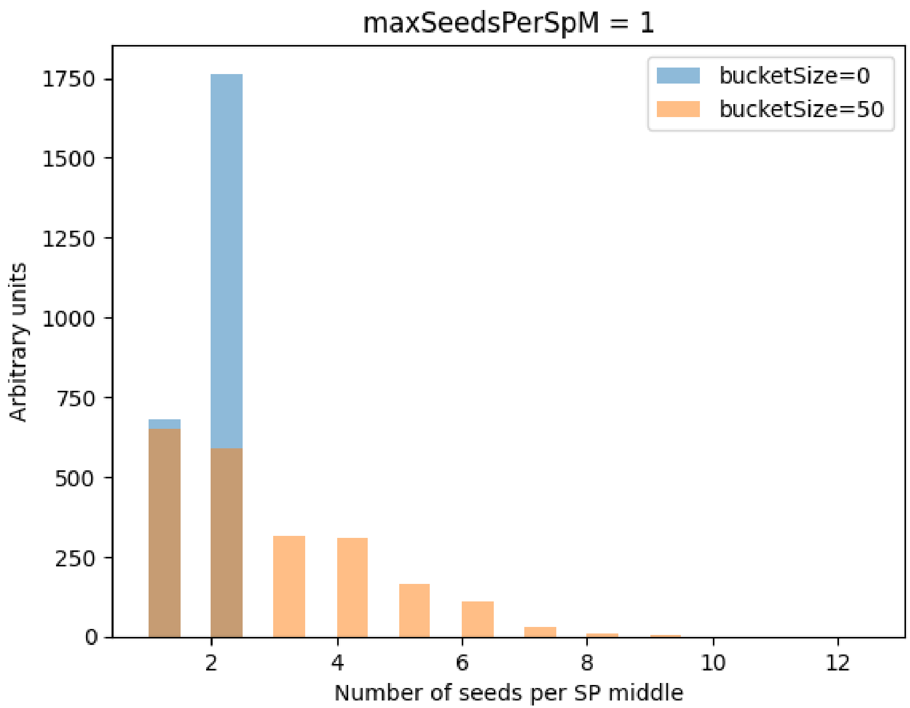
# Origin of improvement

# MaxSeedsPerSpM cut

- **Purpose:**
  - Reduce the number of seeds to expand to speedup the track finding
- **Idea:**
  - Only keep at most MaxSeedsPerSpM+1 seeds sharing the same middle space point
- **Implementation:**
  - Uses a score to compare the seeds
  - The score is related to how close the impact parameter is to 0
- **Benefit:**
  - speedup and less memory used
- **Consequence:**
  - Loss of efficiency

# MaxSeedsPerSpM cut vs Hashing




→ Hashing get through the cut

# Seeding: Skipping triplets check with sets

- **Use 5 sets:**
  - Bad bottom: stores incompatible (middle, bottom) space points
  - Good bottom: stores compatible (middle, bottom) space points
  - Bad top: stores incompatible (middle, top) space points
  - Good top: stores compatible (middle, top) space points
  - Triplets: stores every checked (middle, bottom, top) space points

  **+ Set seed container from before: stores compatible triplets**
- **Skip if already in the set**

# Skipping triplets check with sets (results)

- **Event 98: Hashing mu=50 bucketSize=100**
- **9860 Space Points → ~100.000.000 possible doublets**

Overlap indicator

| Set name | Set size | nSkipped | Ratio |
|---|---|---|---|
| **Bad bottom** | **24.433.199** | 322.132.498 | 13,18 |
| **Good bottom** | **3.592.664** | 63.294.324 | 17,62 |
| **Bad top** | **30.363.102** | 392.248.454 | 12,92 |
| **Good top** | **4.973.975** | 91.166.619 | 18,33 |
| **Triplets** | **18.204.058** | 269.635.750 | 14,81 |
| **Seeds** | **5.623** | x | x |

MaxSeedsPerSpM = 1 → Seeds

Total running time x1.5

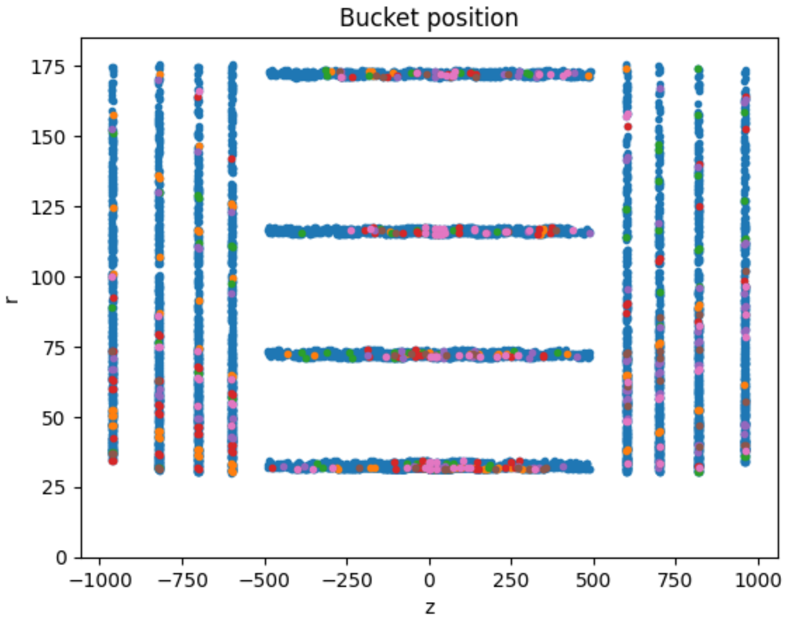
# Origin of improvement Conclusion

- **MaxSeedsPerSpM cut reduces physics performance**
- **Hashing get through the MaxSeedsPerSpM cut**
- **There is overlap between the buckets:**
  - The seeds are reconstructed several times

# The metric

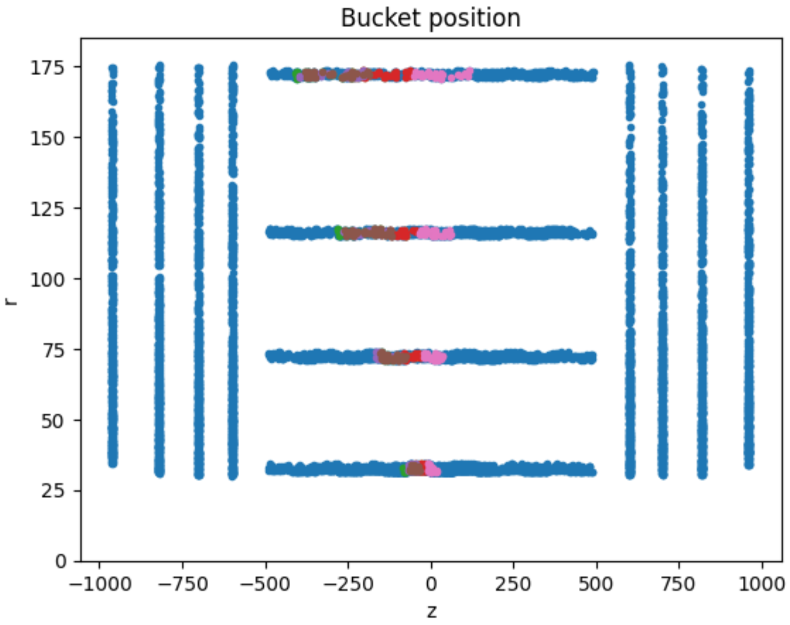
# Other metric: ΔR

Angular: Δφ

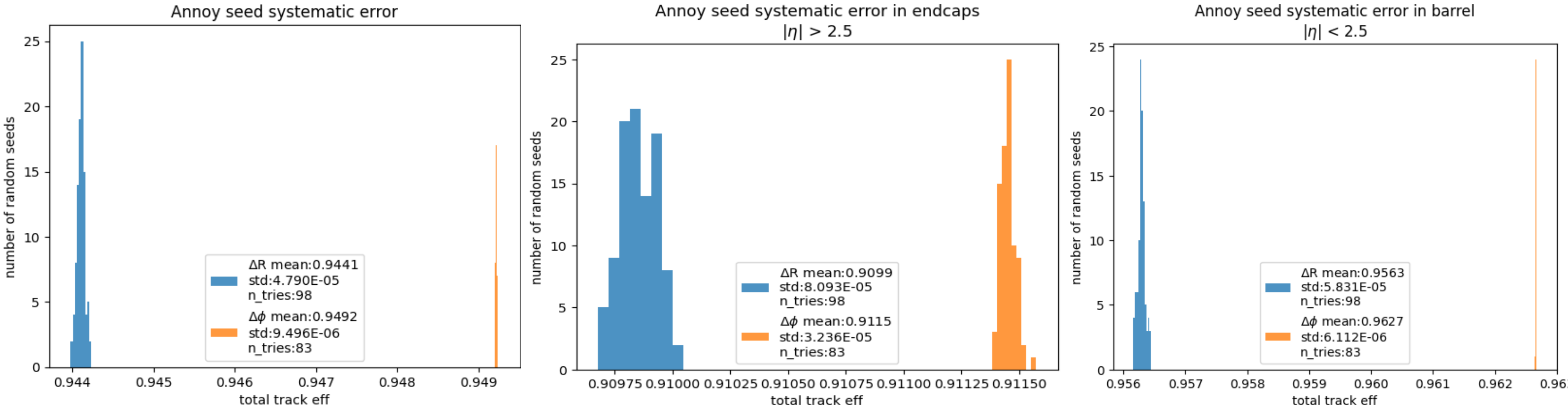



$\Delta R = \sqrt{\Delta\varphi^2 + \Delta\eta^2}$

If $\Delta\varphi > \pi$:
$\Delta\varphi = 2*\pi - \Delta\varphi$



# Annoy random seed systematic error



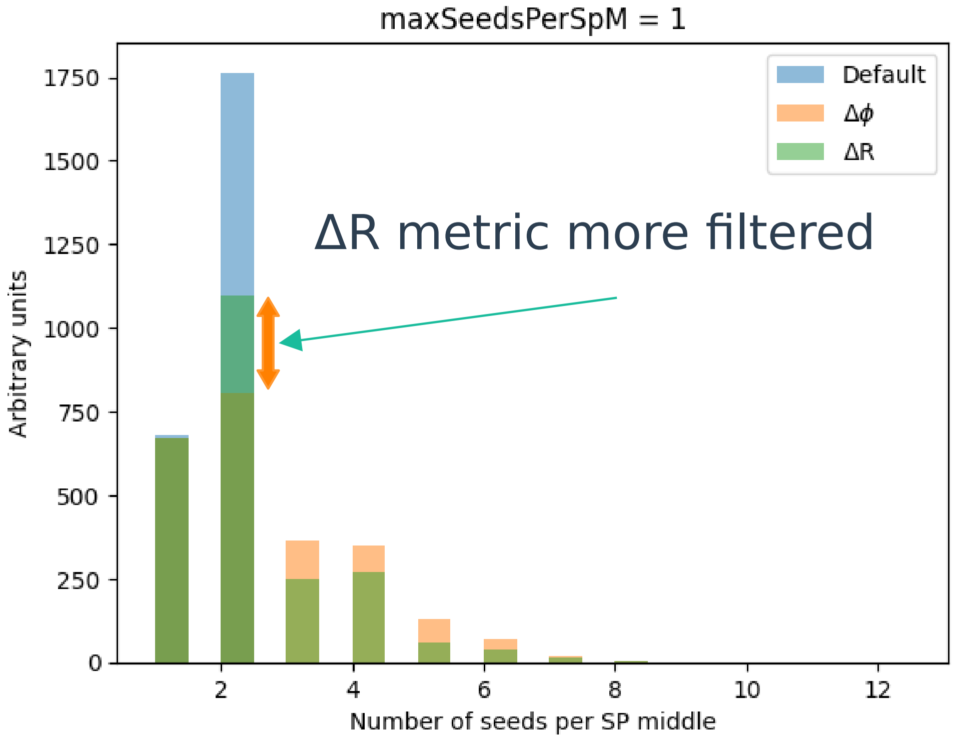

1000 events
in each try

BucketSize: 100
Mu: 50

Δφ is better

# MaxSeedsPerSpM and ΔR metric

On 1 event:




Filtered Middle Space points are on the maxSeedsPerSpM bin

Some of the “Buckets shared Middle Space points” are on the bins after the maxSeedsPerSpM bin

Differences in the bins before maxSeedsPerSpM correspond to lost seeds

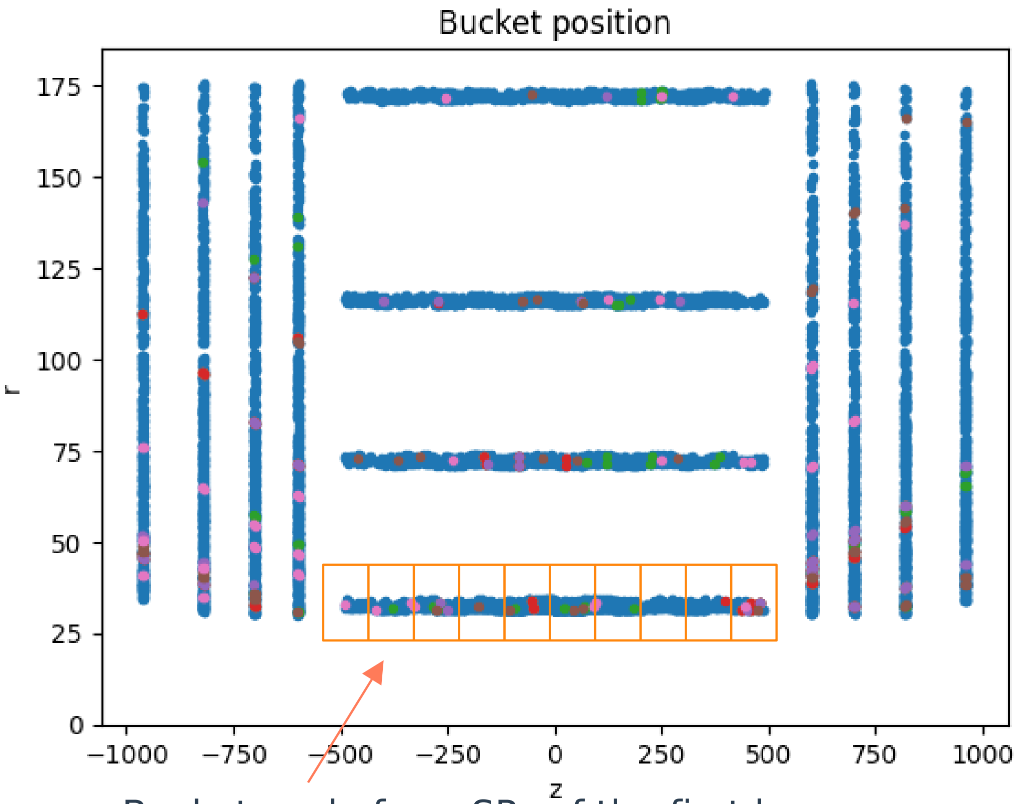
Default nSeeds: 4208
Δφ nSeeds: 6053
ΔR nSeeds: 5300

# Metric Conclusion

- **Different metrics lead to different performance**
- **MaxSeedsPerSpM cut favors buckets with unrelated tracks**
- **The random seed of Annoy has a smaller impact**
- **The number of reconstructed seeds depends of the metric**

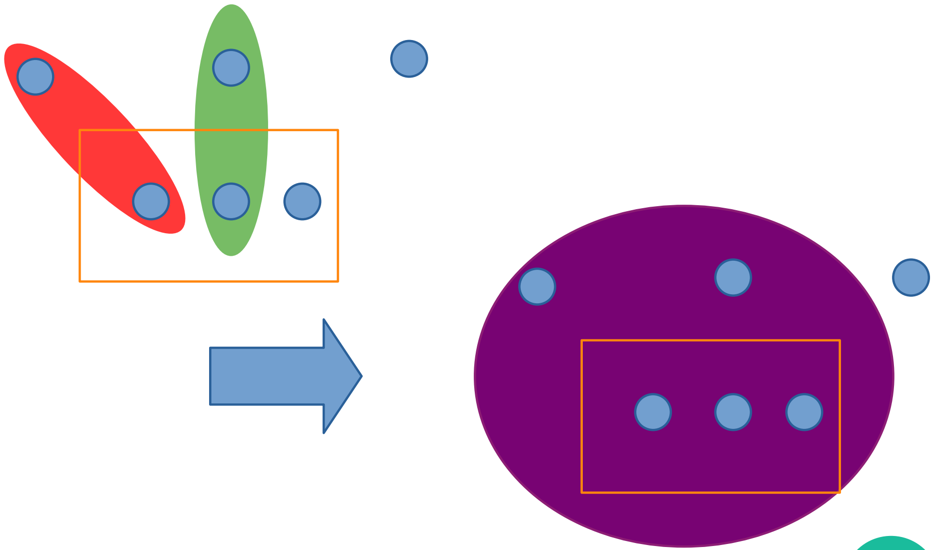
# Binning and super buckets

# Binning

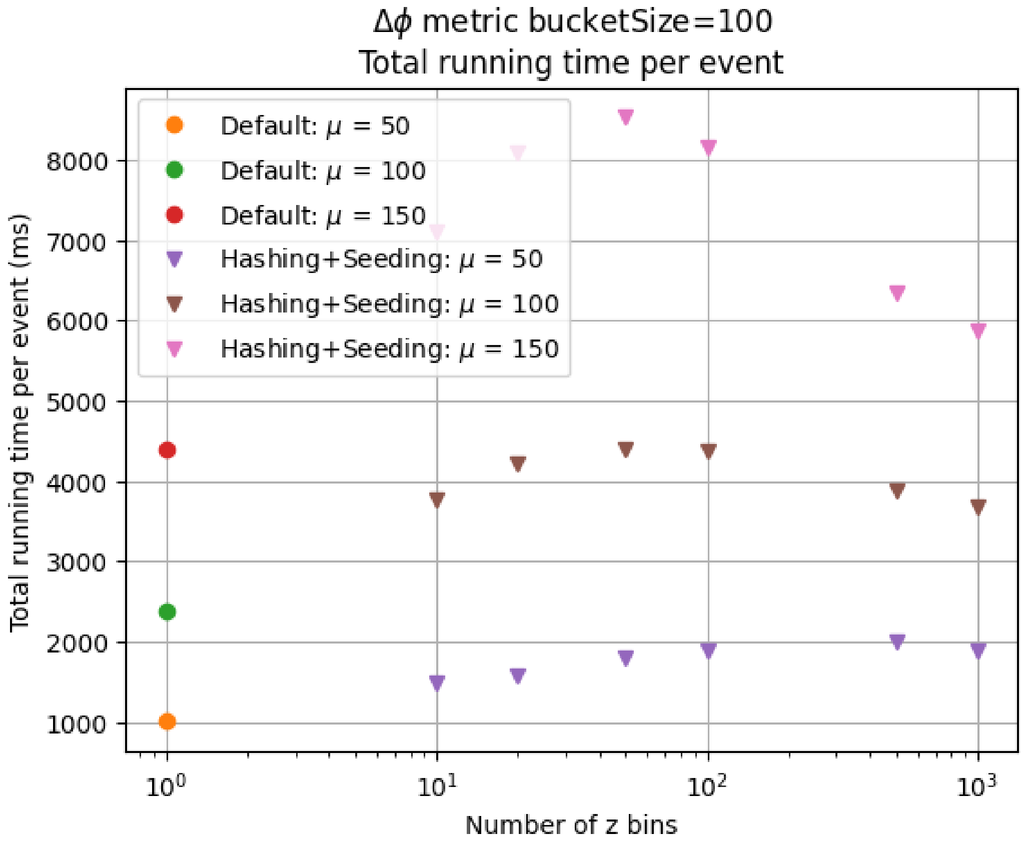


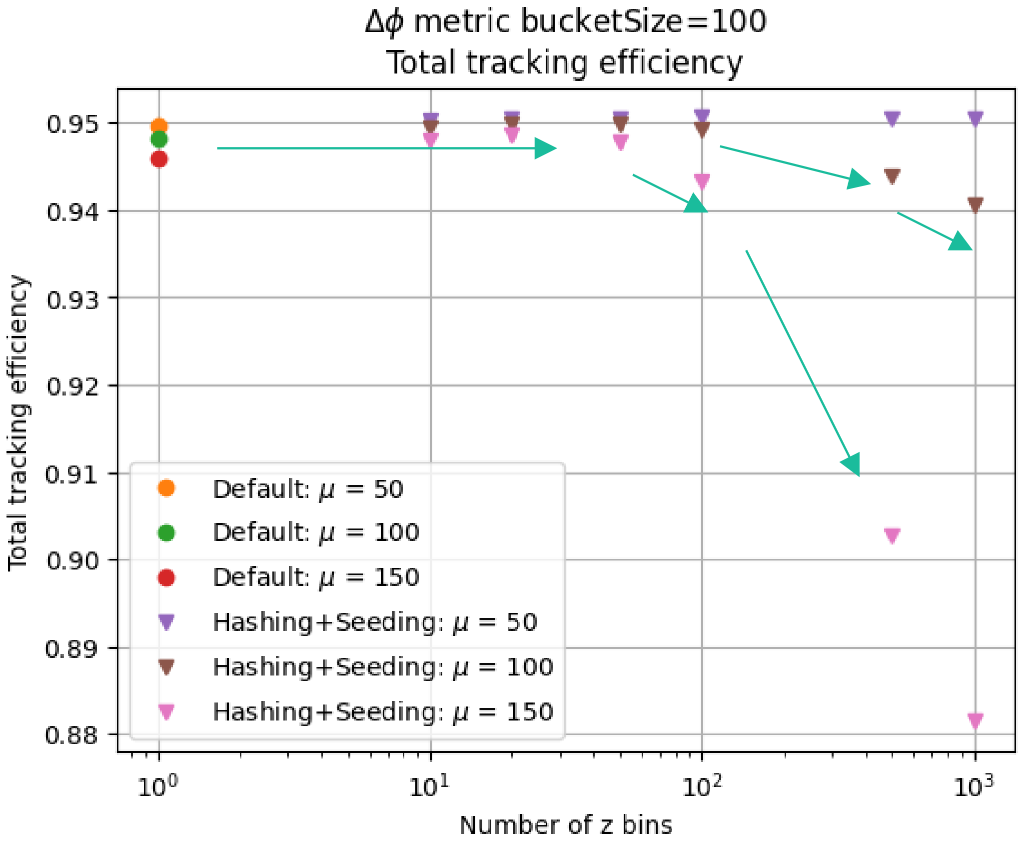

Buckets only from SPs of the first layer

Super bucket:
Merging of the buckets created from the space points inside the bin

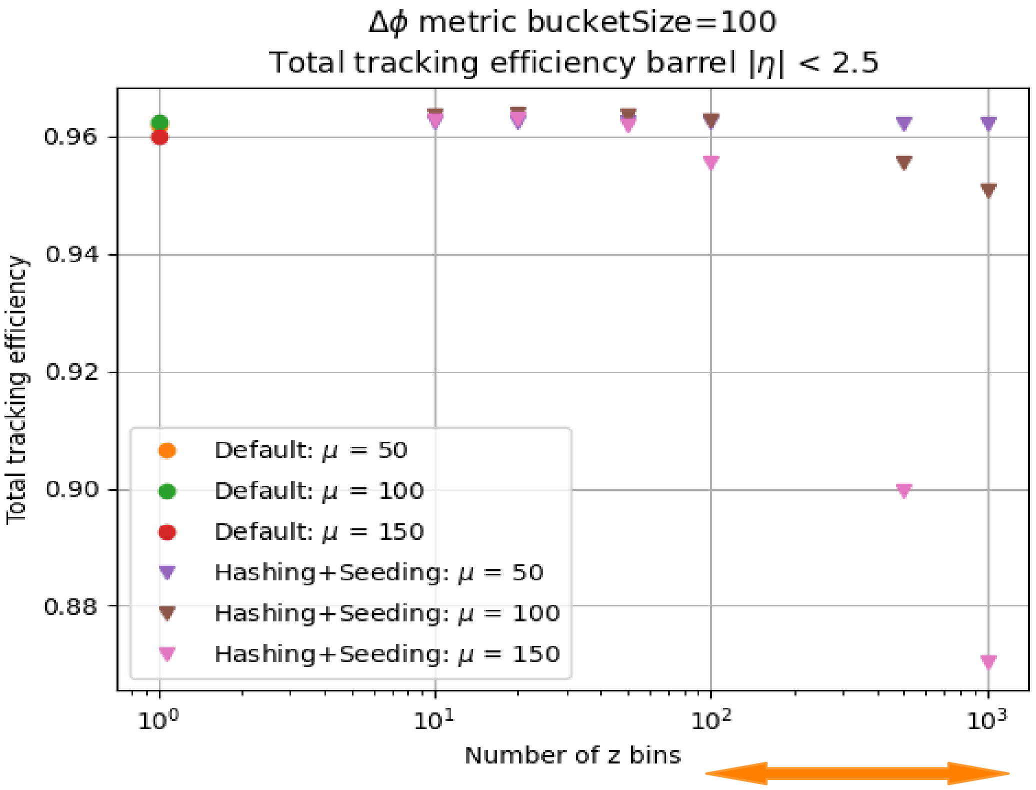


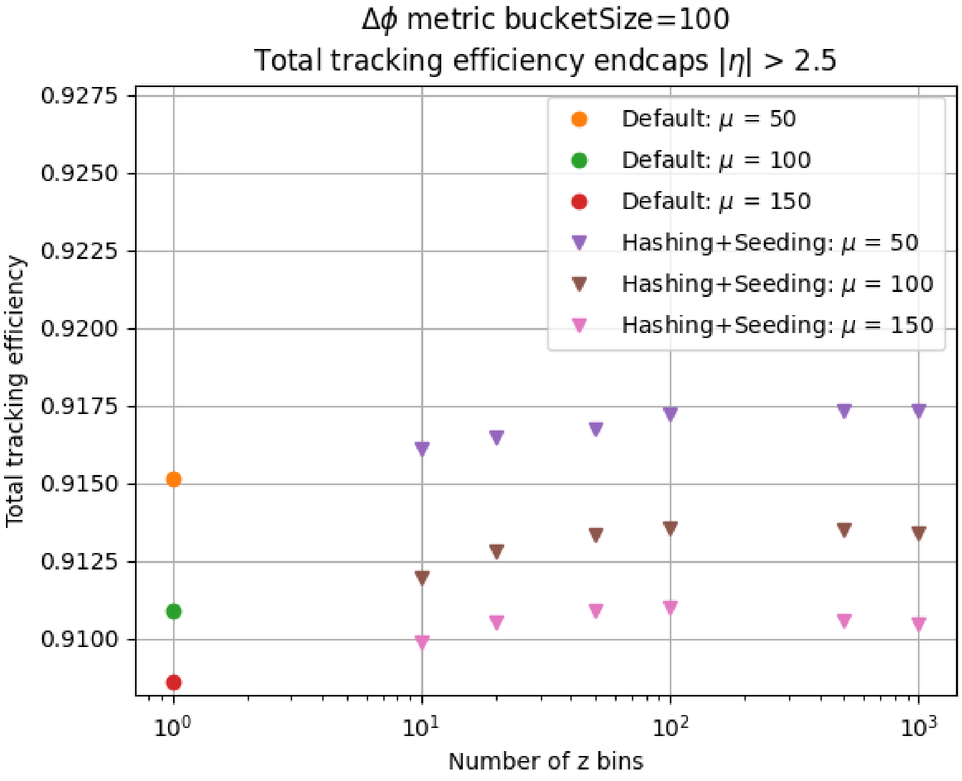
# Perfomances vs zBins



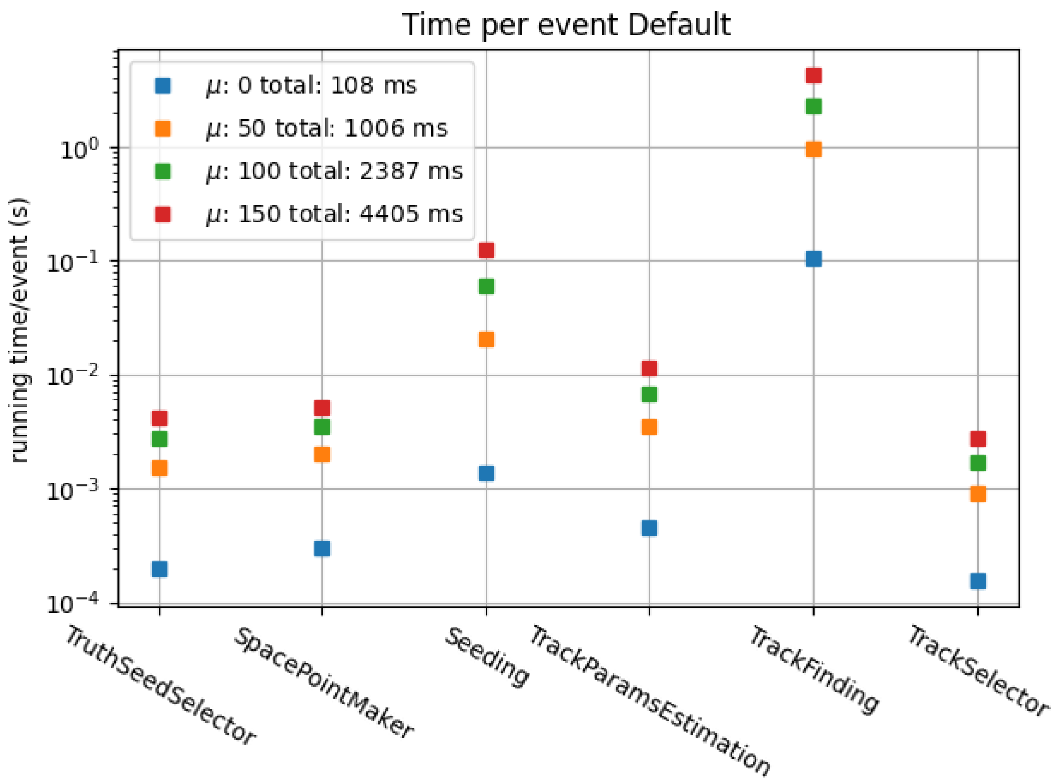
# Efficiency vs zBins (detailed)

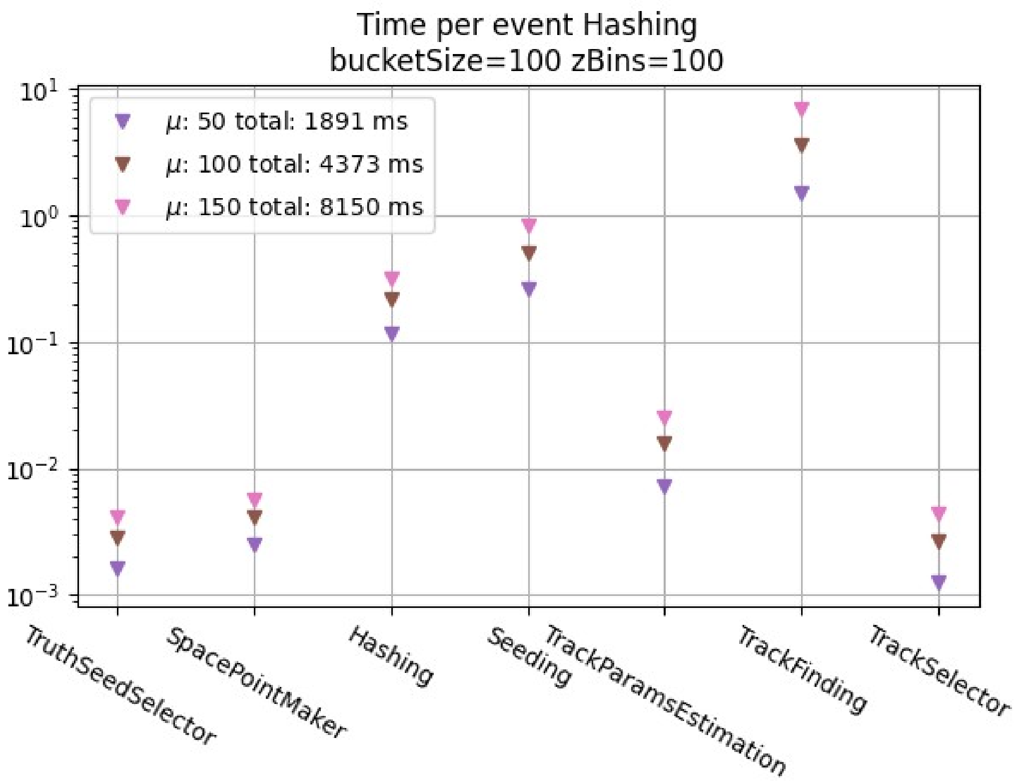



MaxSeedsPerSpM cut dominates

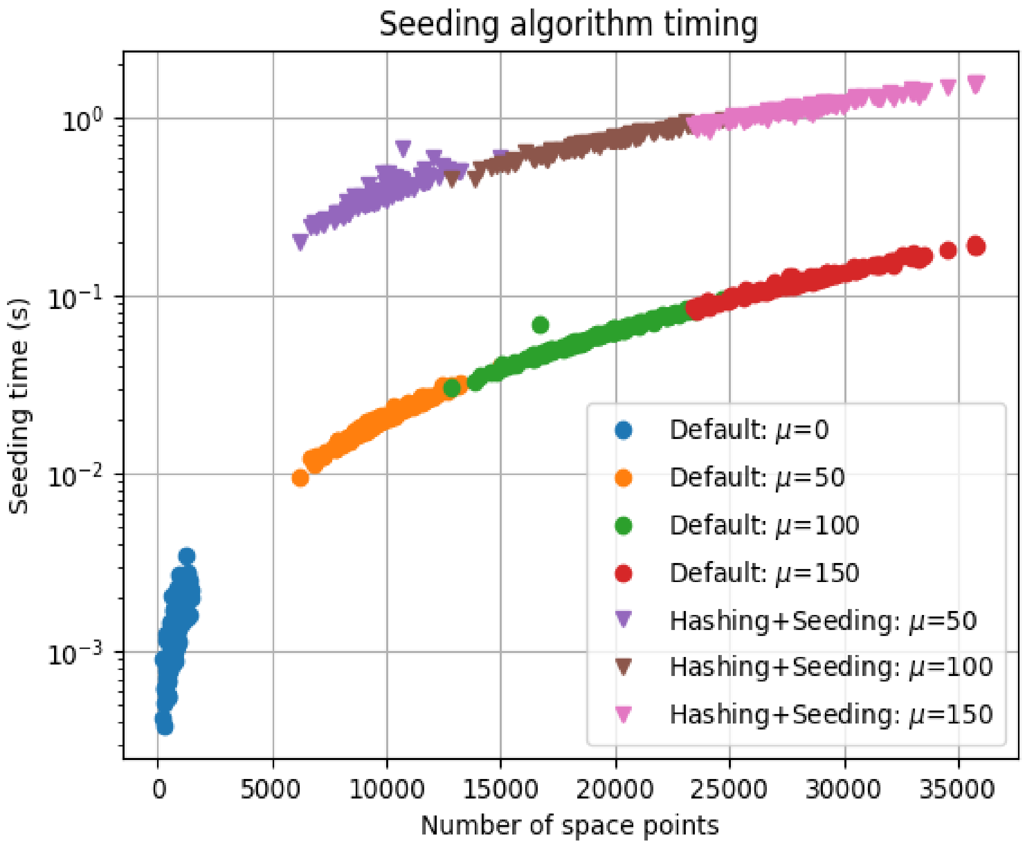
# Time per event (detailed)



# Timing vs number of space points

BucketSize = 100 ; zBins = 100

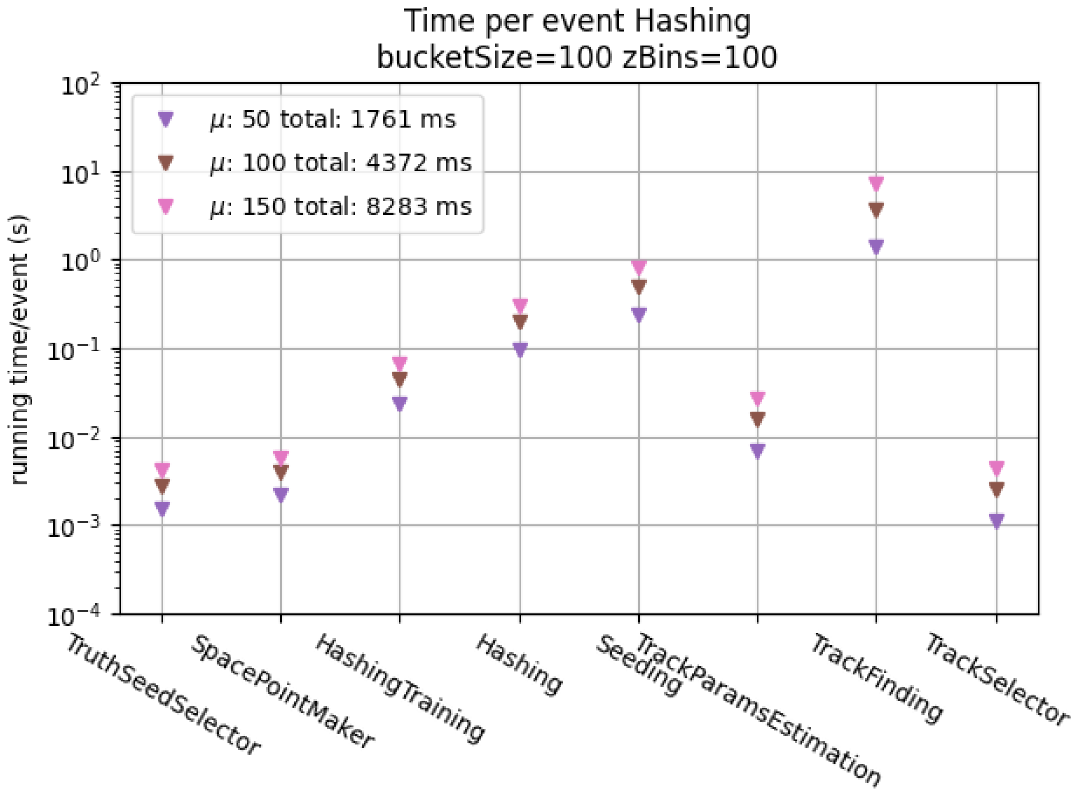


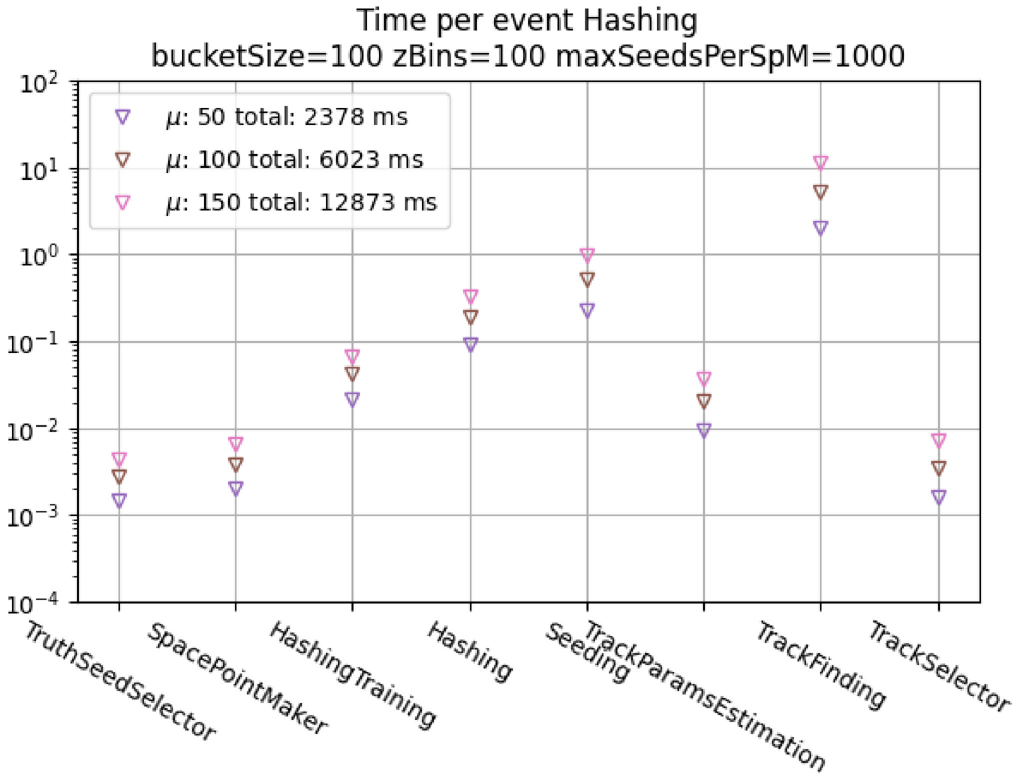
# Binning Conclusion

- Binning impacts physics and timing performances
- Hashing takes more time than default
- Total running time ~ x2 default

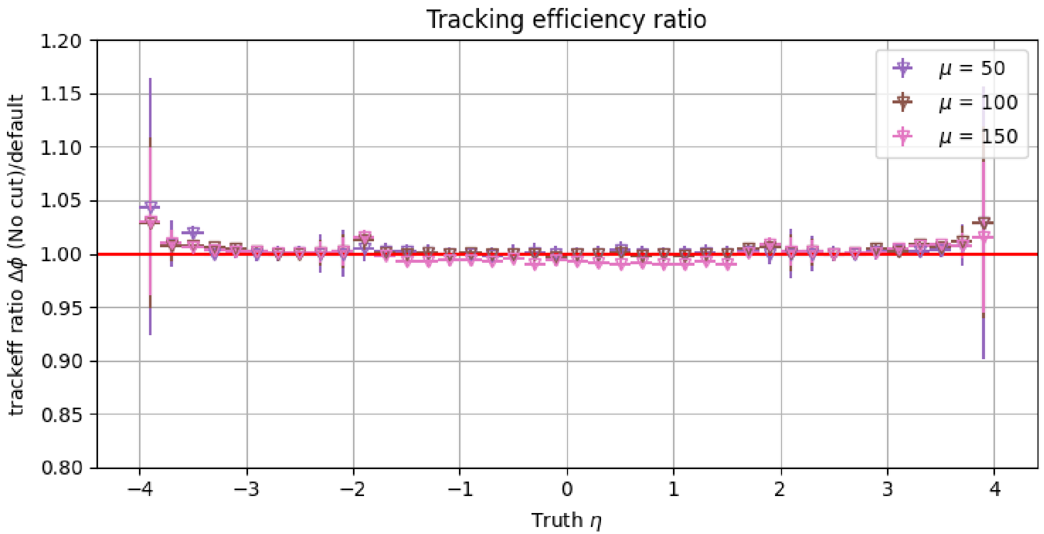
# Removing the cut

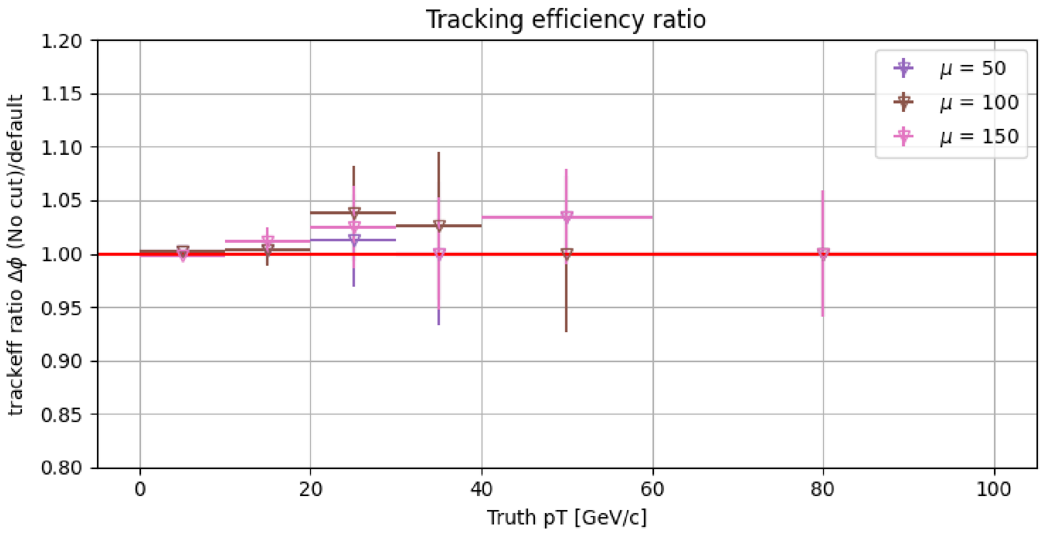
# Running time no cut



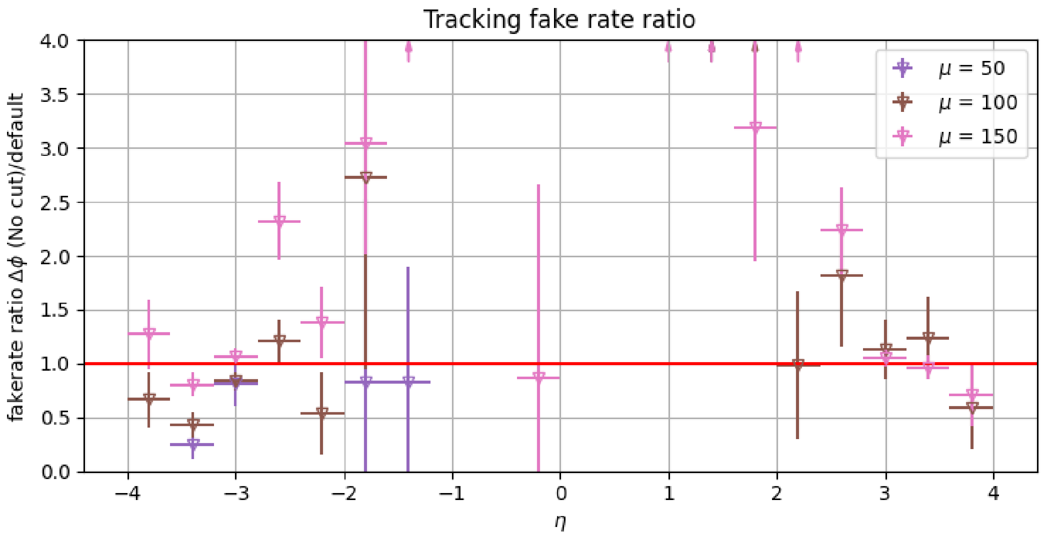



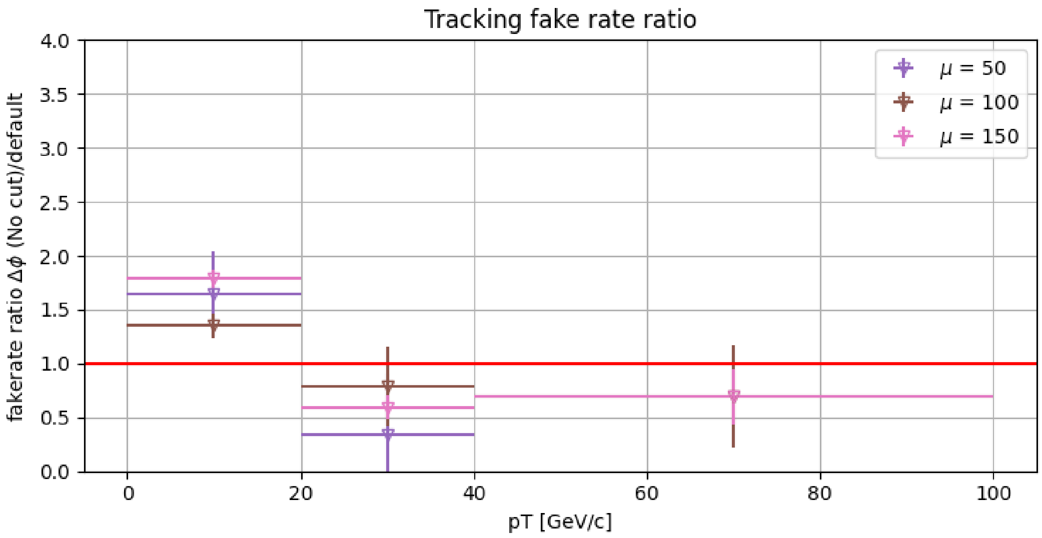
# Ratio no cut vs Default: efficiency



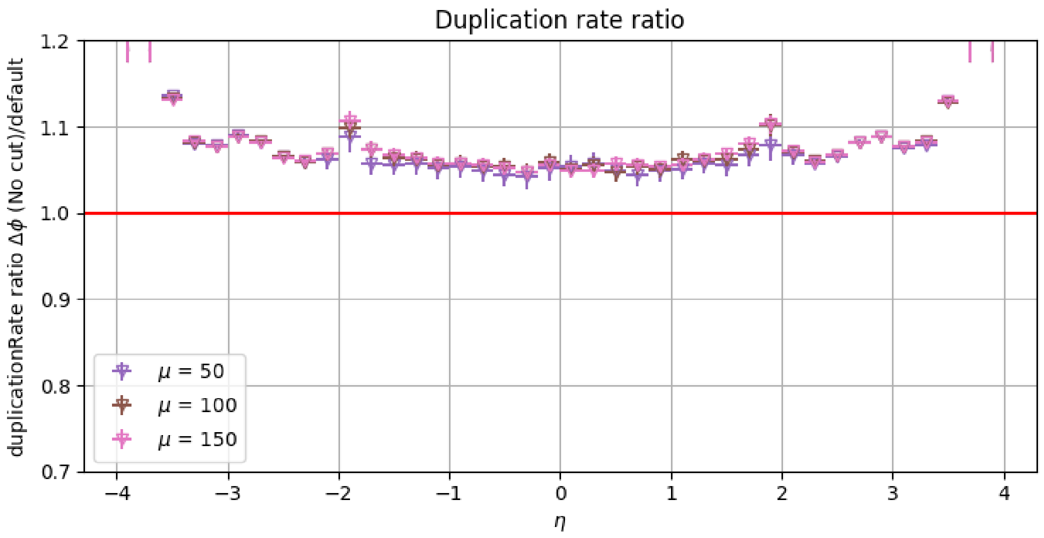



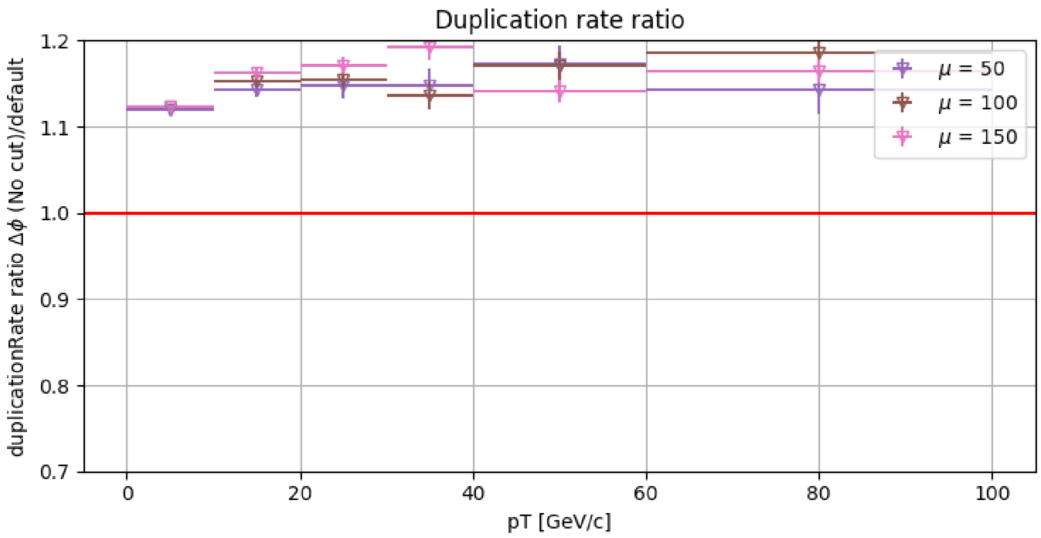
# Ratio no cut vs Default: fake rate



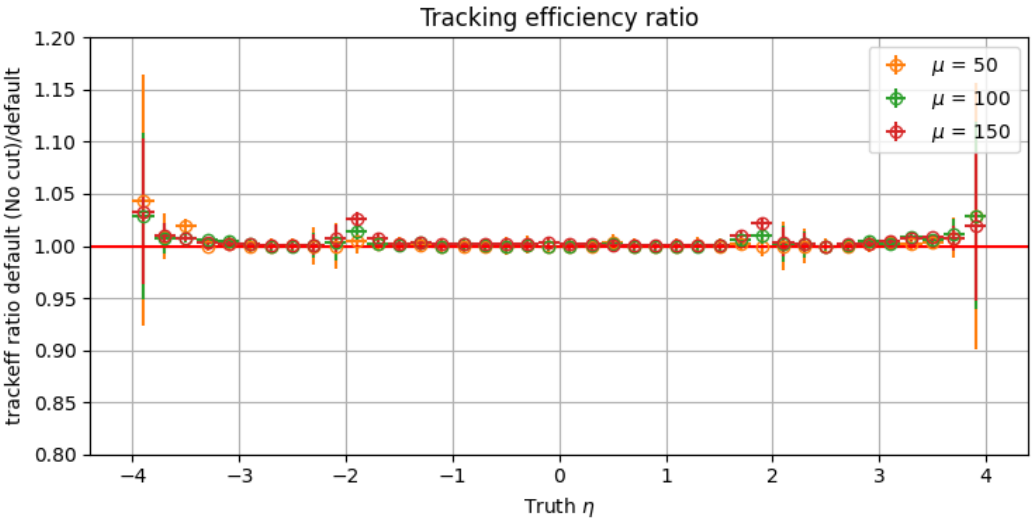



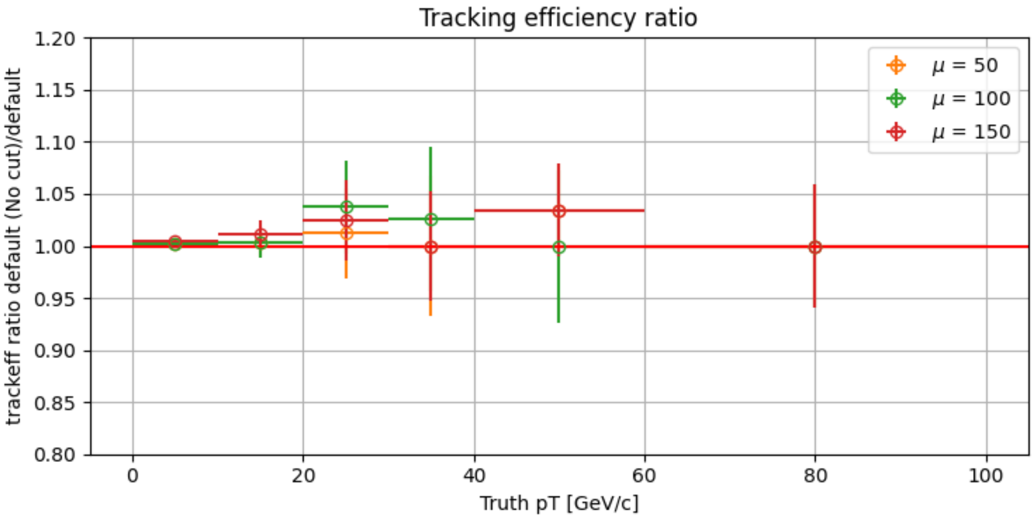
# Ratio no cut vs Default: duplicates



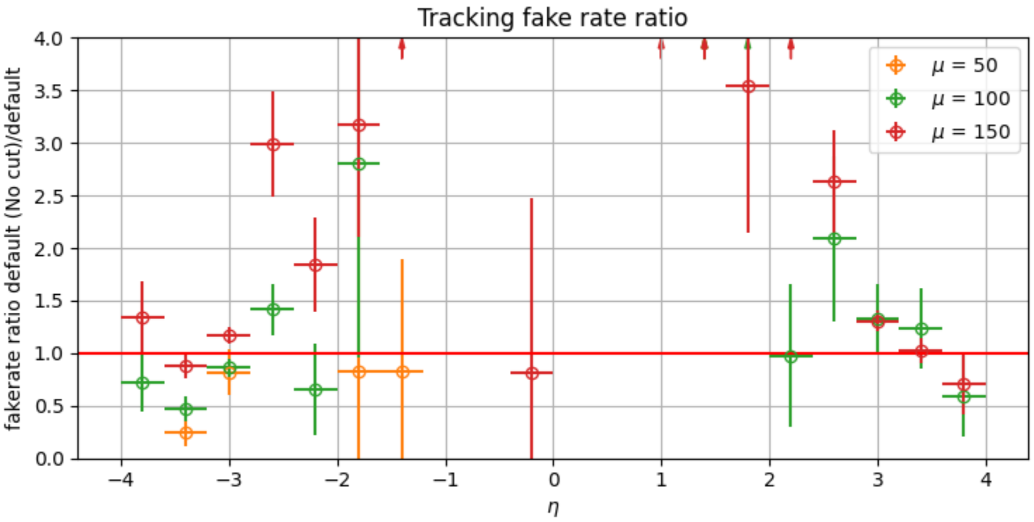
# Ratio Default no cut vs Default: efficiency

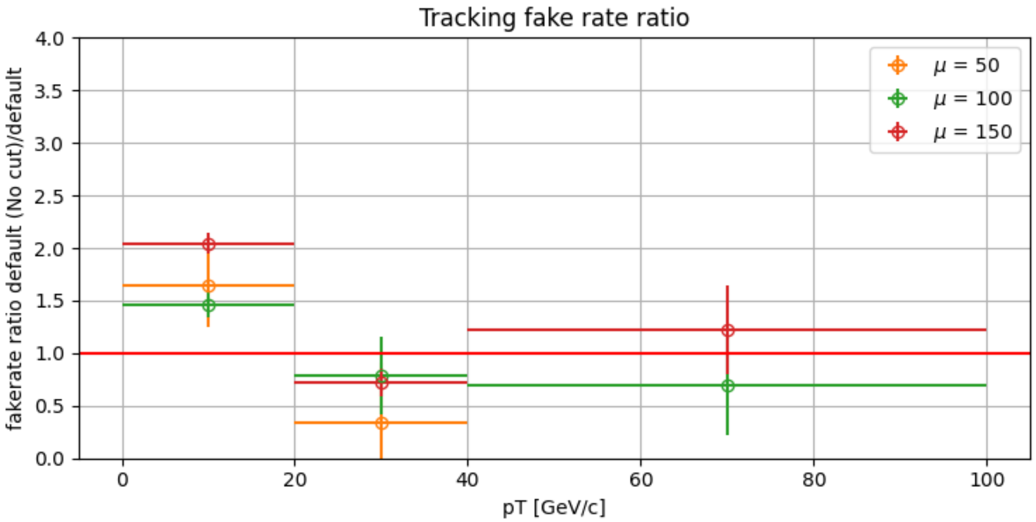



Cutted tracks are in forward region and around |η| = 2

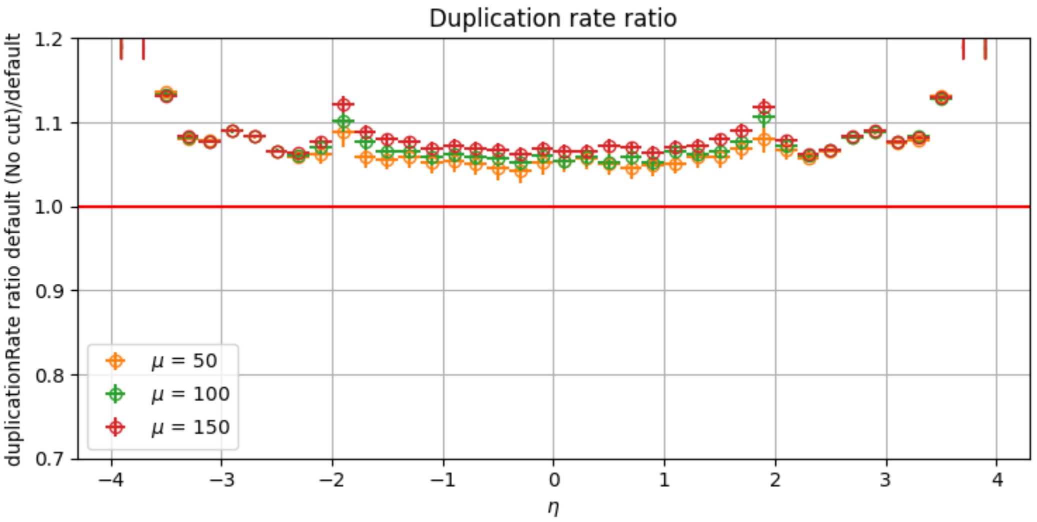
# Ratio Default no cut vs Default: fake rate

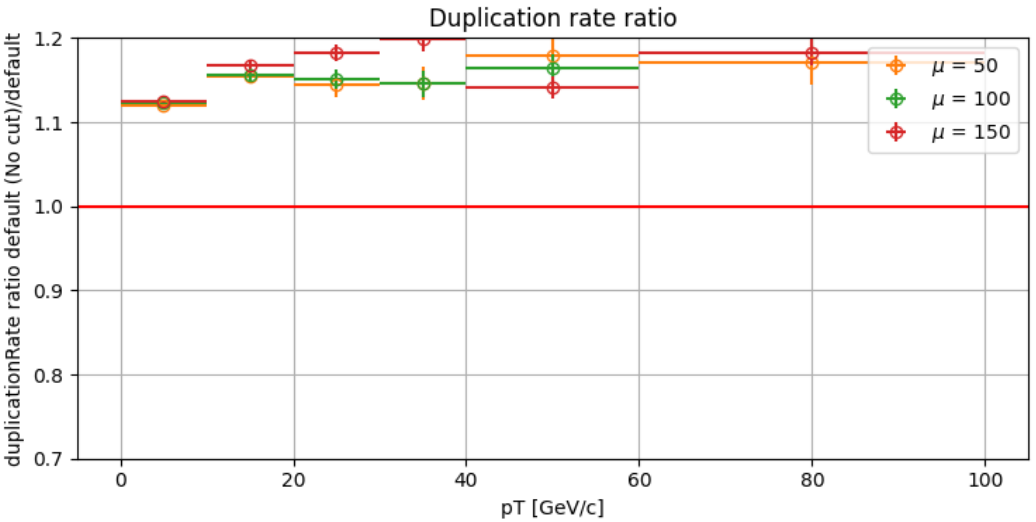



Higher fake rate in central region and low pT

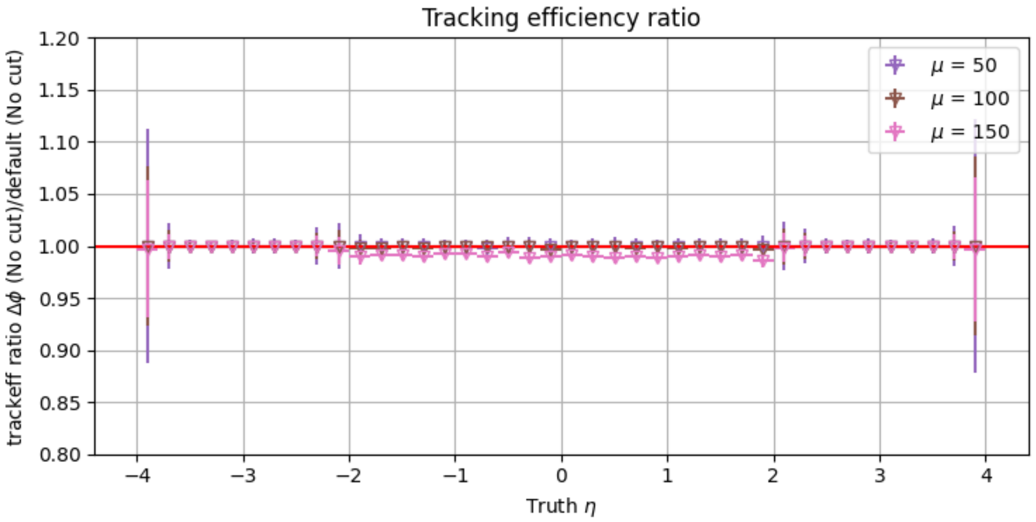
# Ratio Default no cut vs Default: duplicates

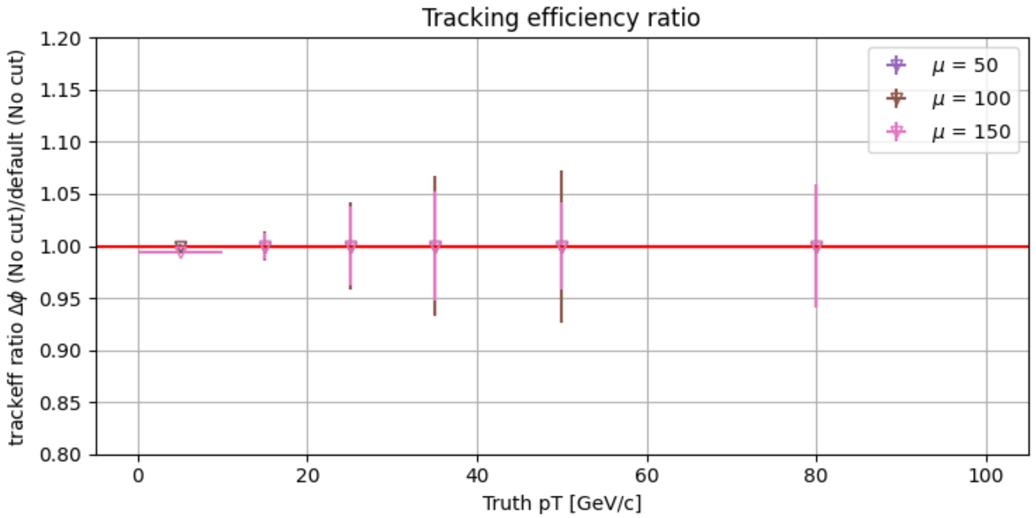



Higher duplication rate and shape similar to efficiency

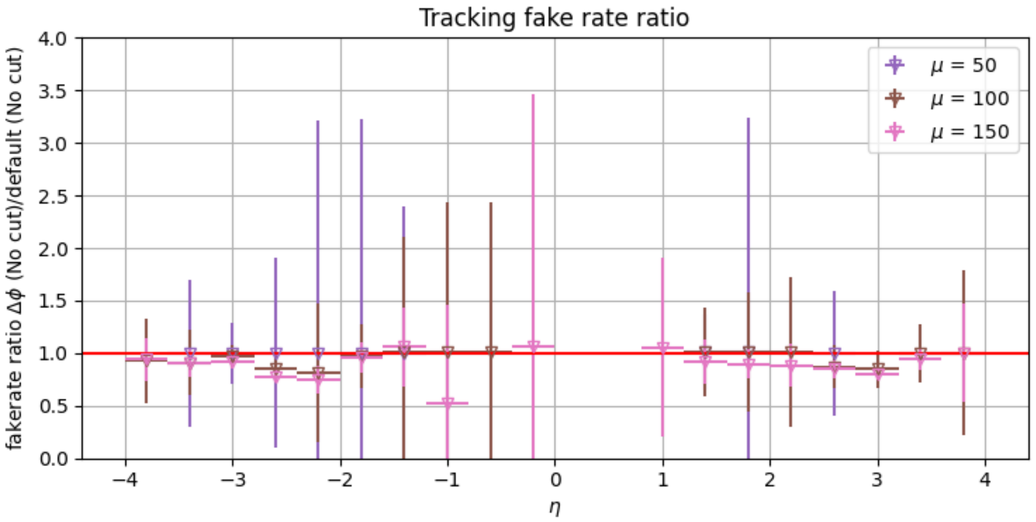
# Ratio Hashing no cut vs Default no cut: efficiency

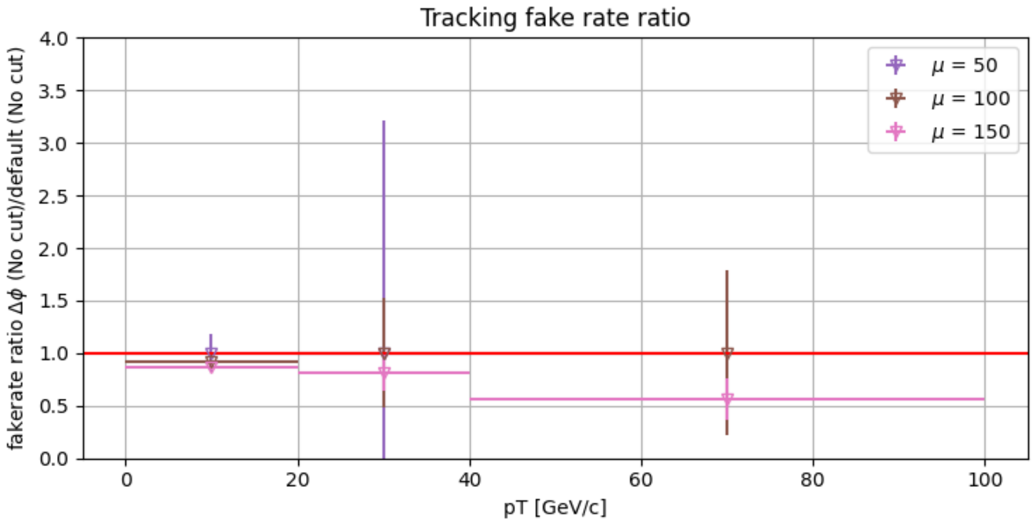



Same efficiency with and without hashing except in central region (low pT)

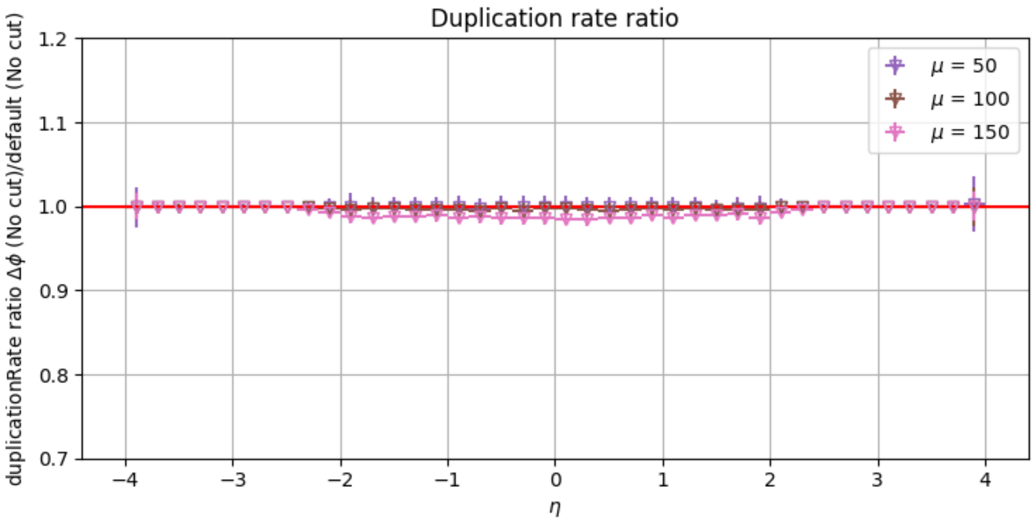
# Ratio Hashing no cut vs Default no cut: fake rate

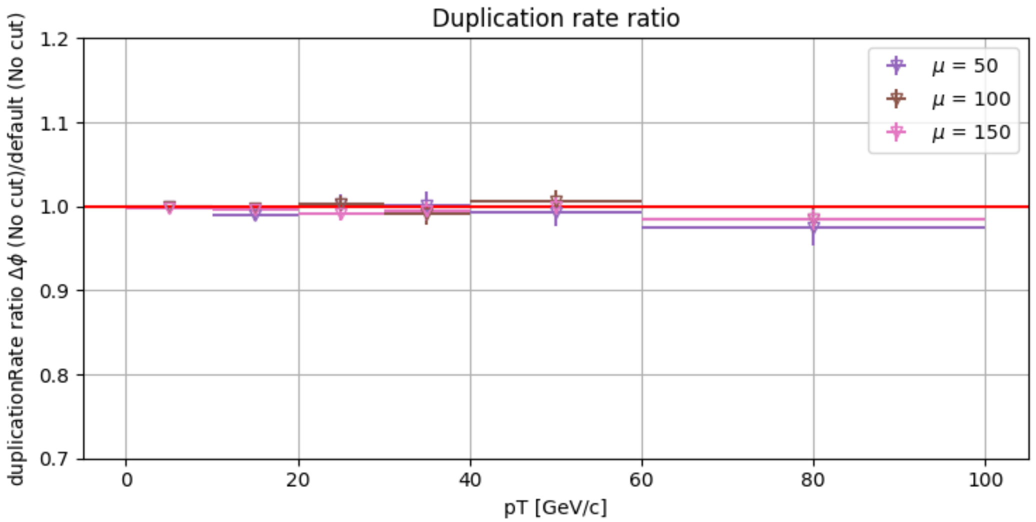



Similar fake rates

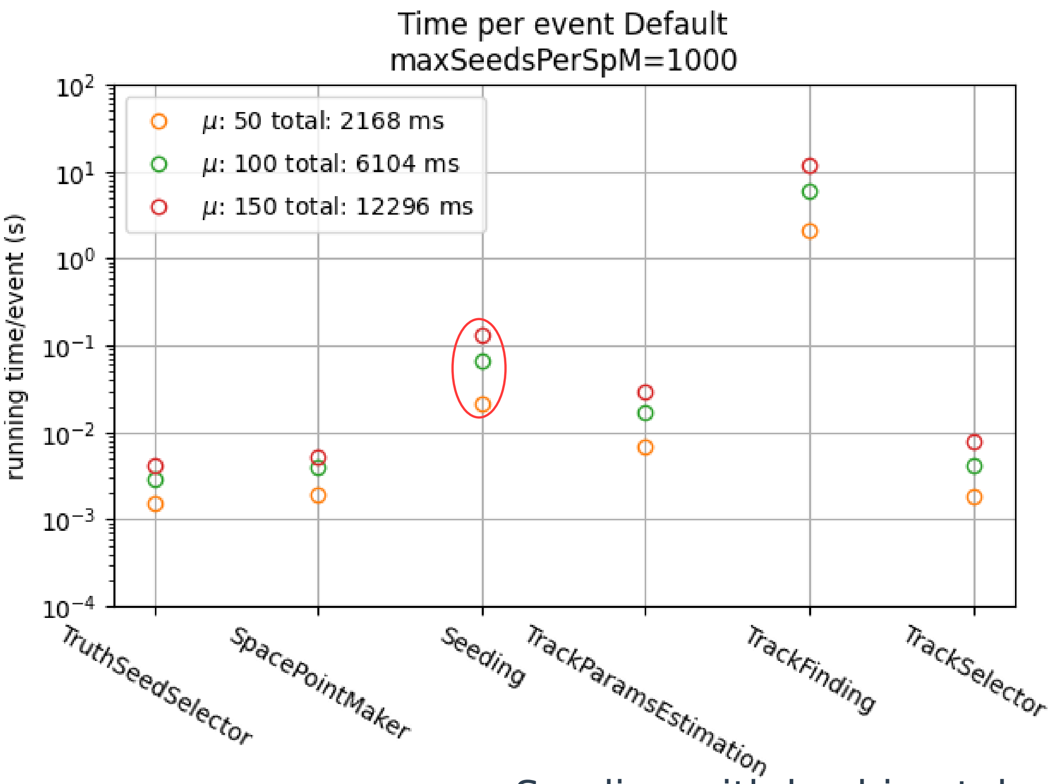
# Ratio Hashing no cut vs Default no cut: duplicates

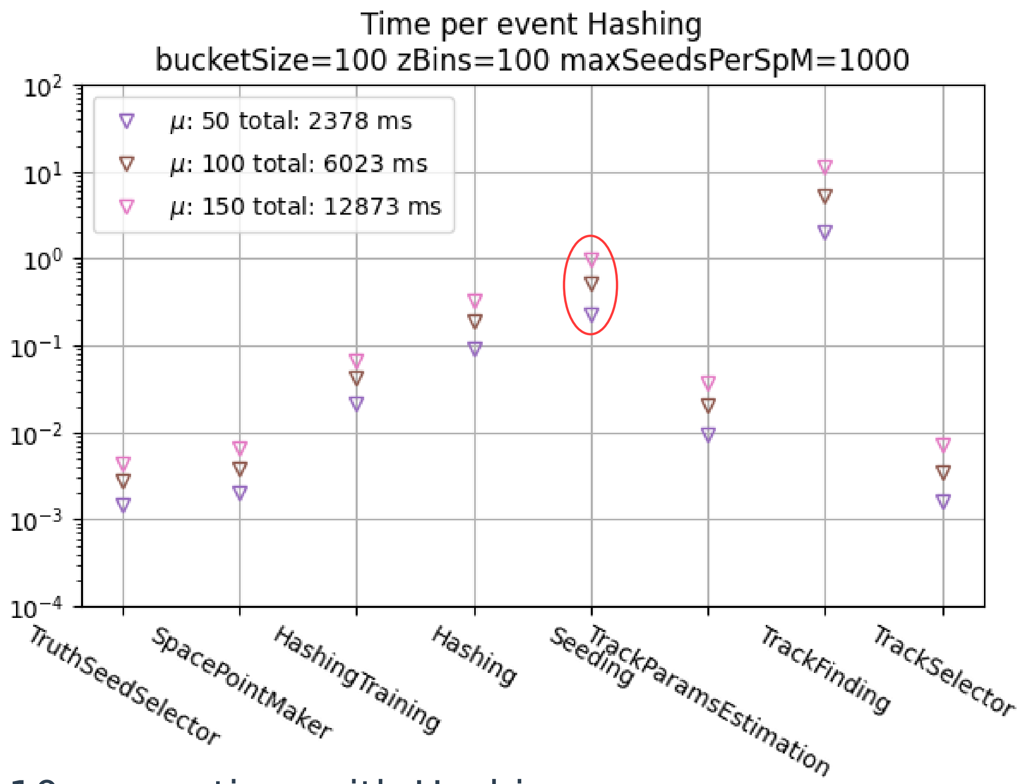



Same duplication rate except in central region

# Running time no cut



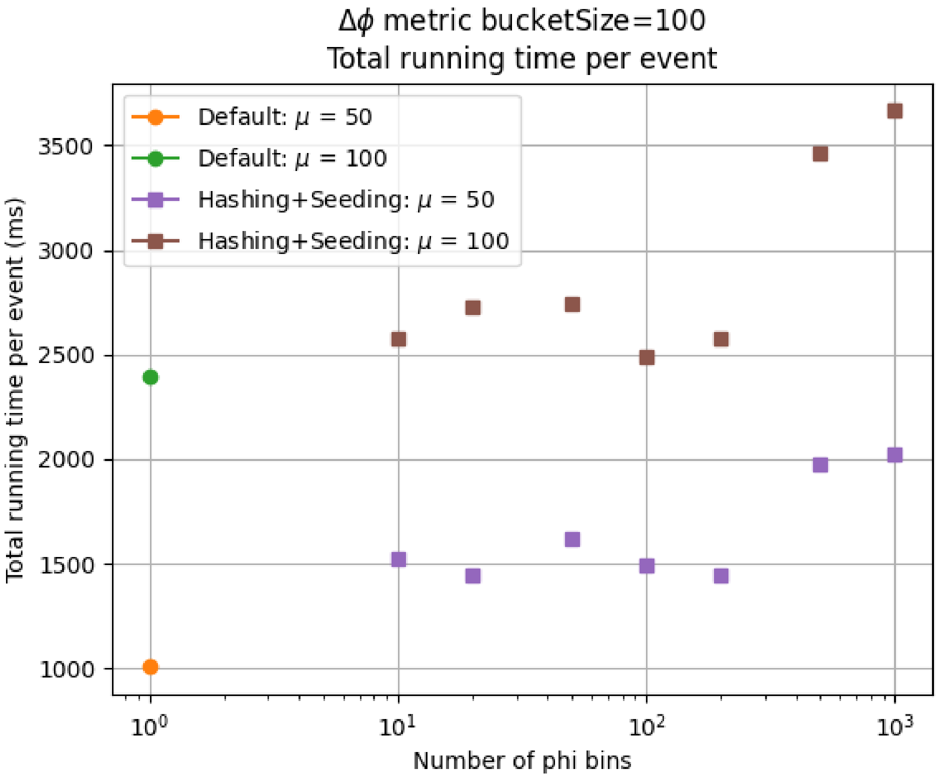

Seeding with hashing takes 10x more time with Hashing
~ 900 ms for $\mu$ = 150

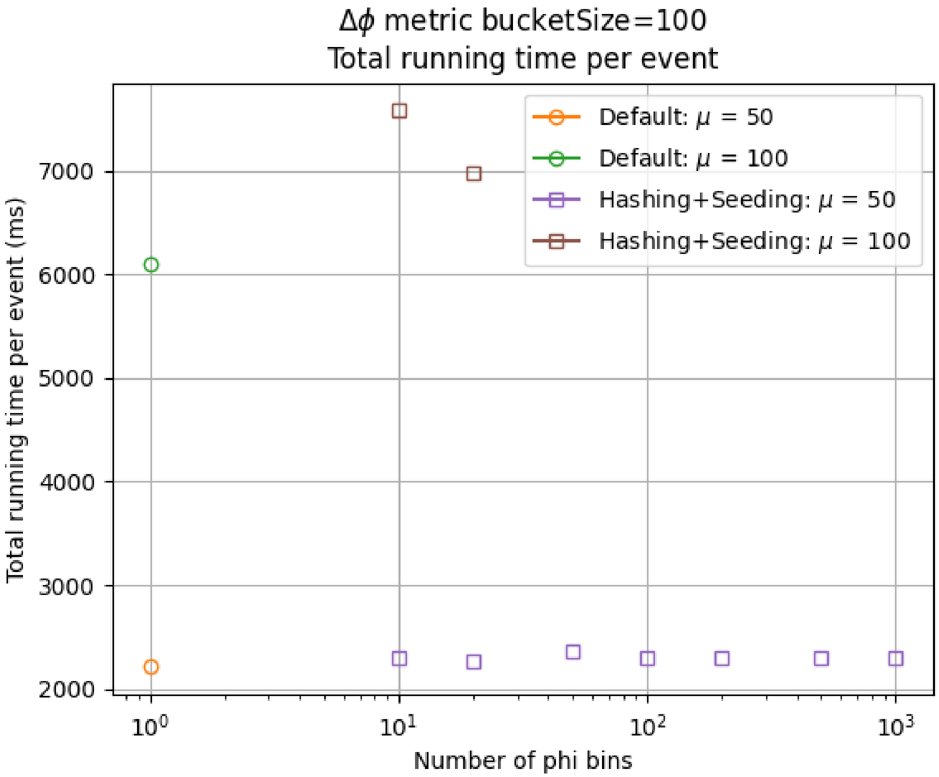
# Removing cut Conclusion

- **Without cut Hashing performances are similar to Default without cut**
  - No gain using hashing without the cut
- **Improving seeding running time is not enough**
  - Need to reduce the number of seeds for track finding

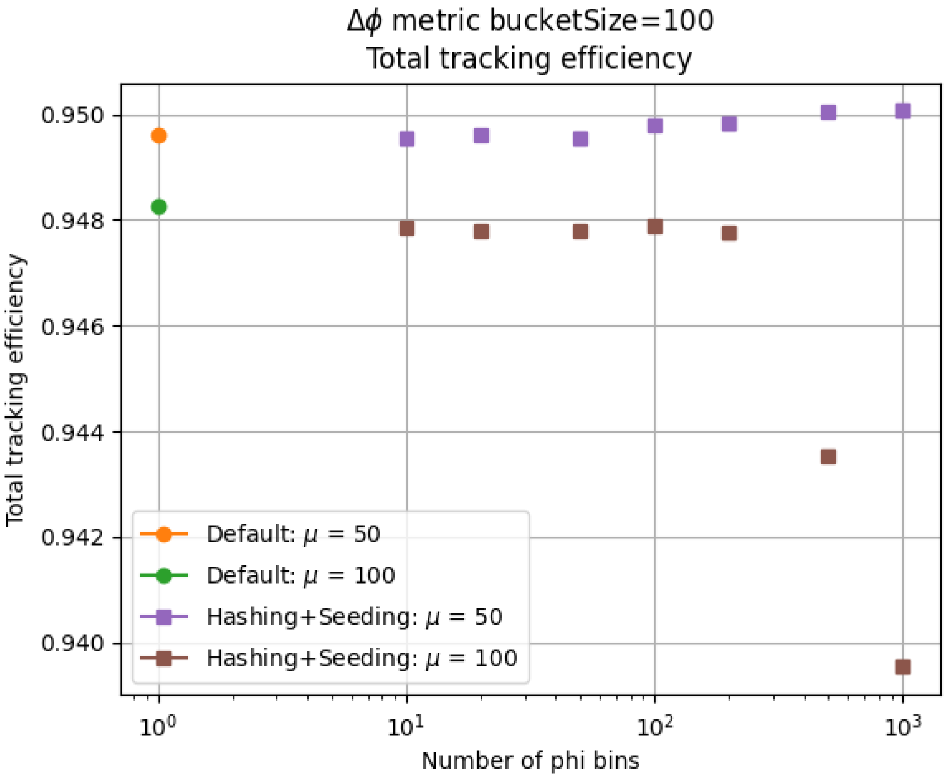
# Superbuckets in Phi

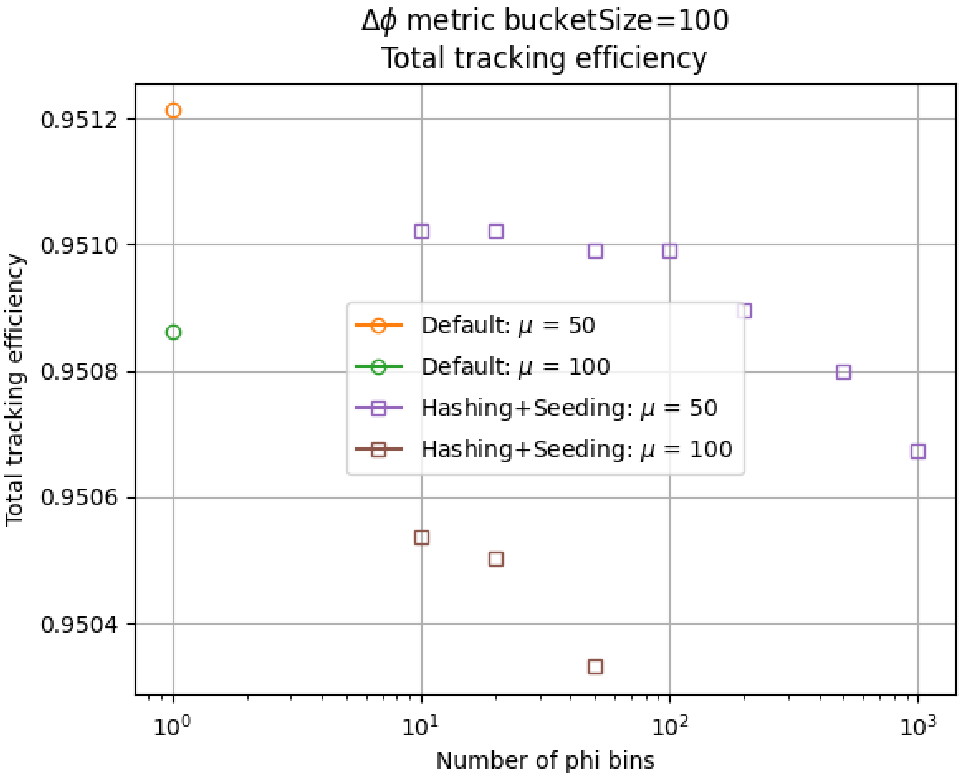
# Phi bins (no cut): Timing



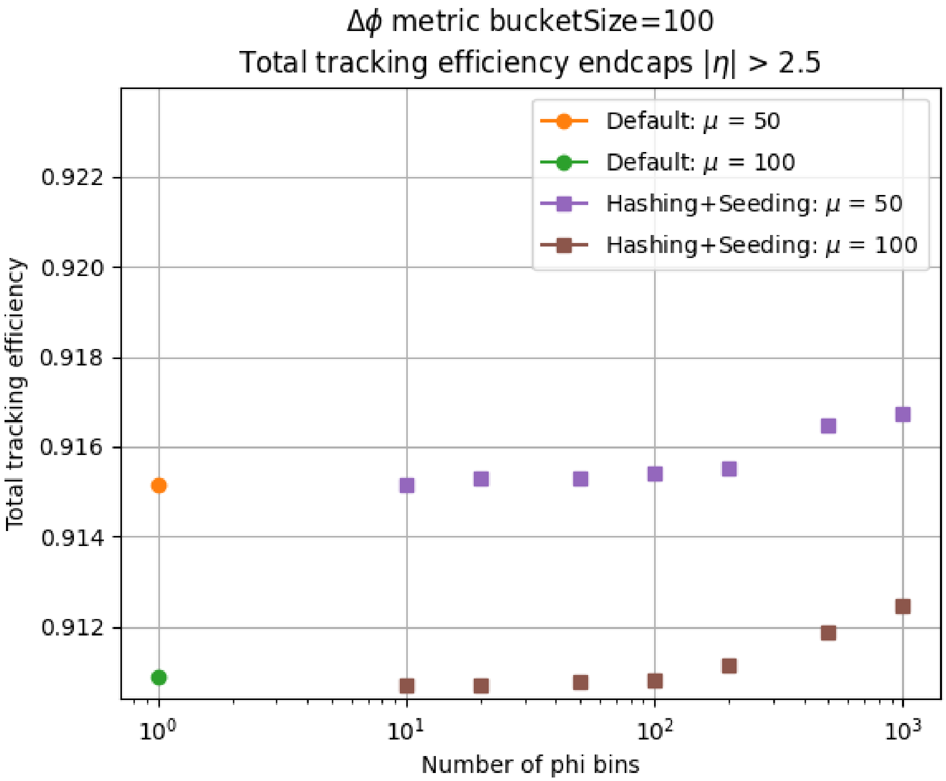



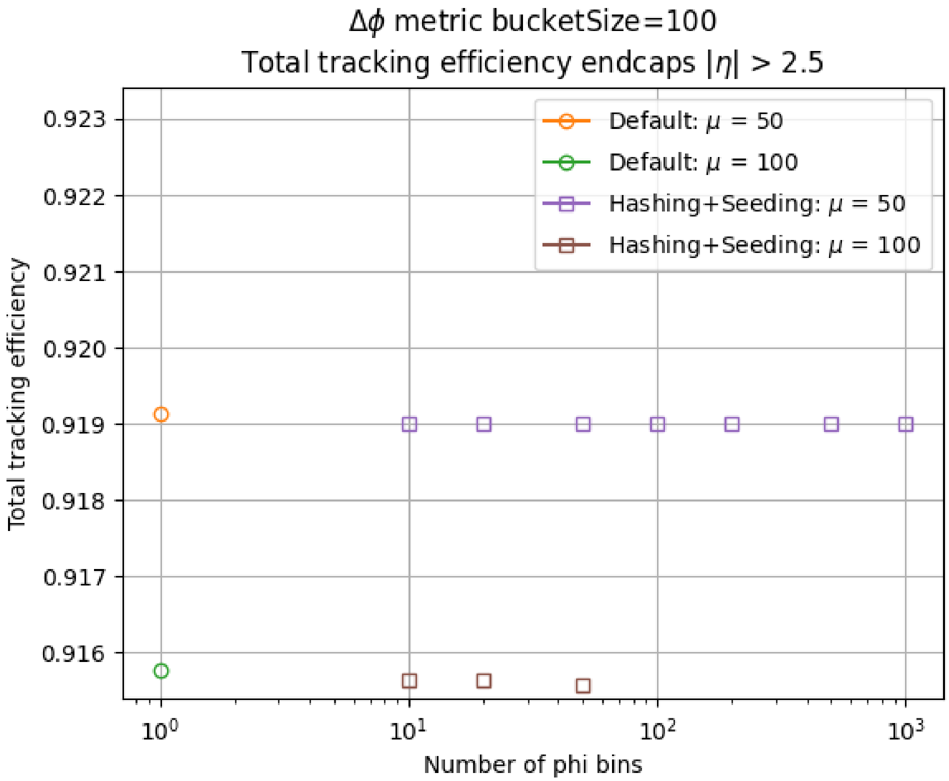
# Phi bins (no cut): Tracking efficiency



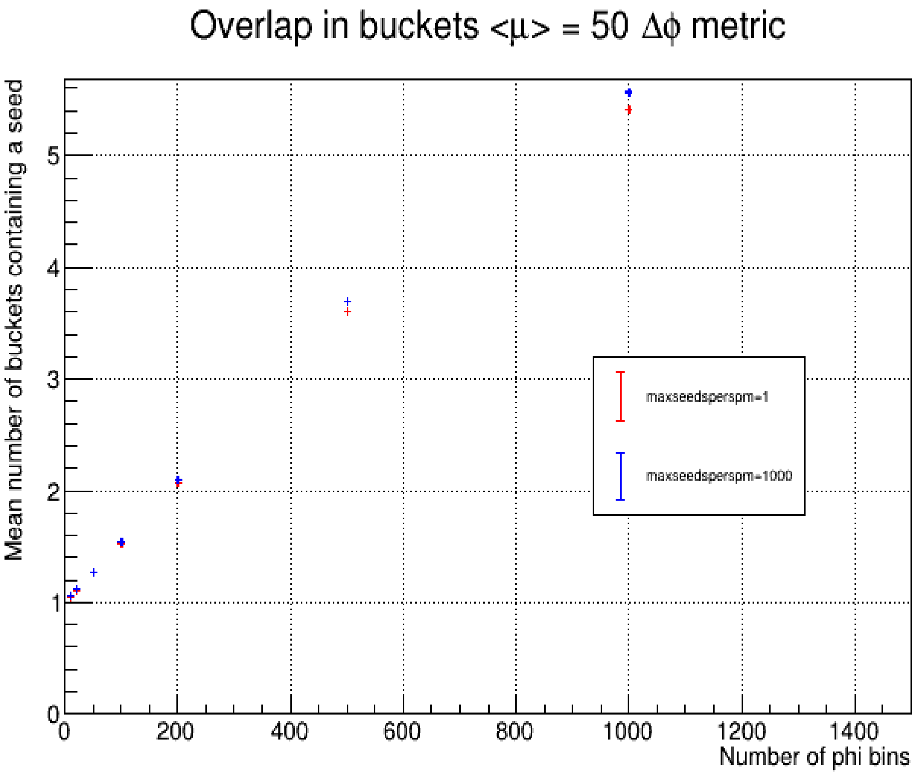



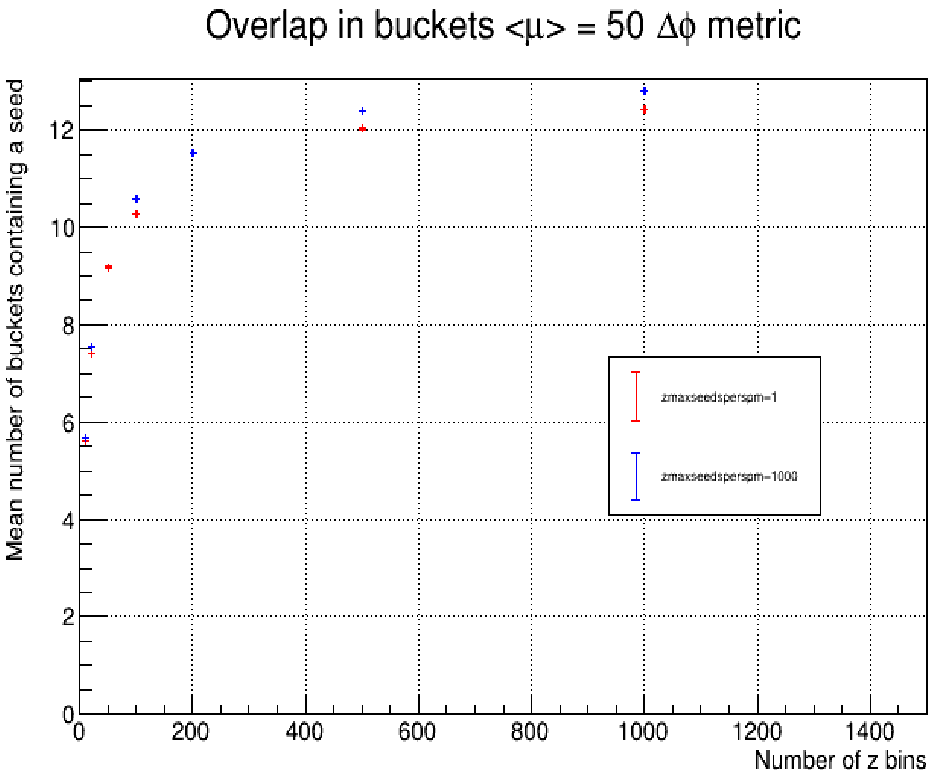
# Phi bins (no cut): Tracking efficiency



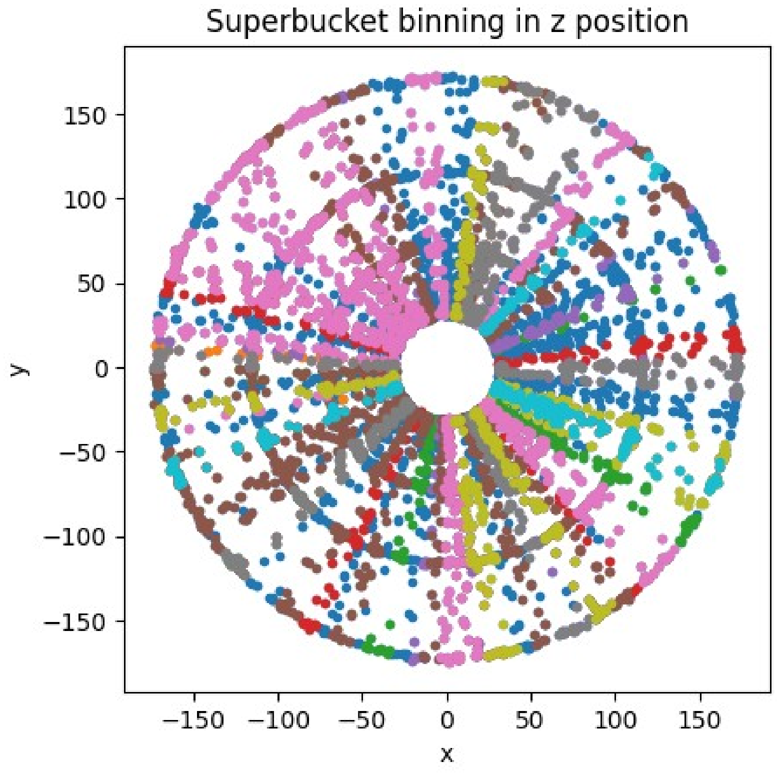



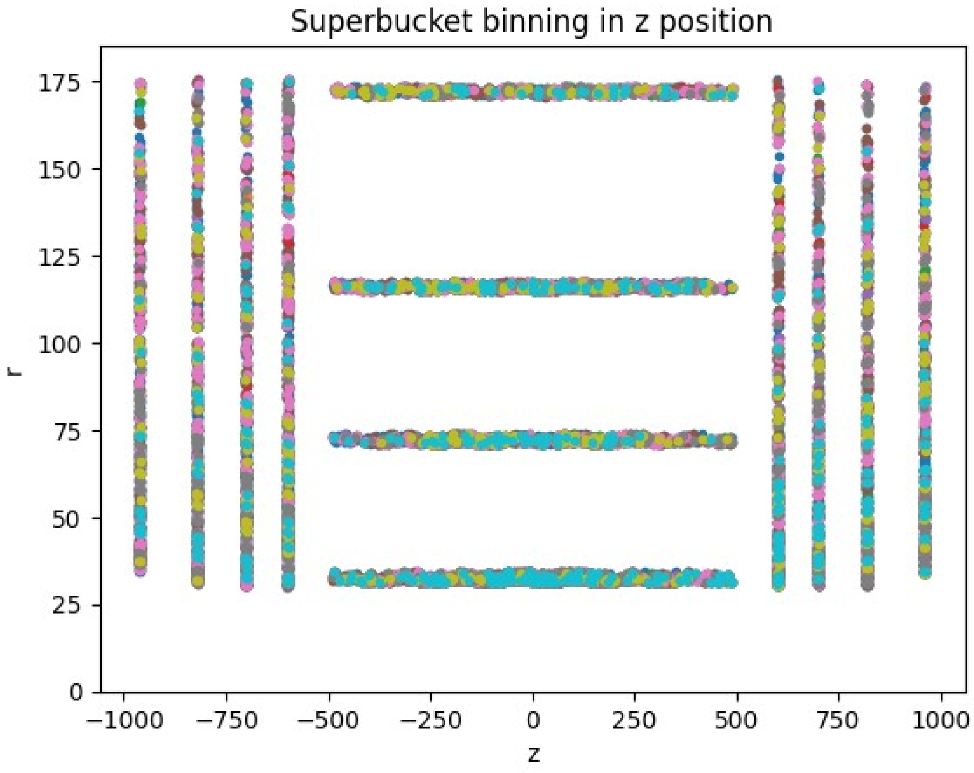
# Phi bins: overlap in buckets



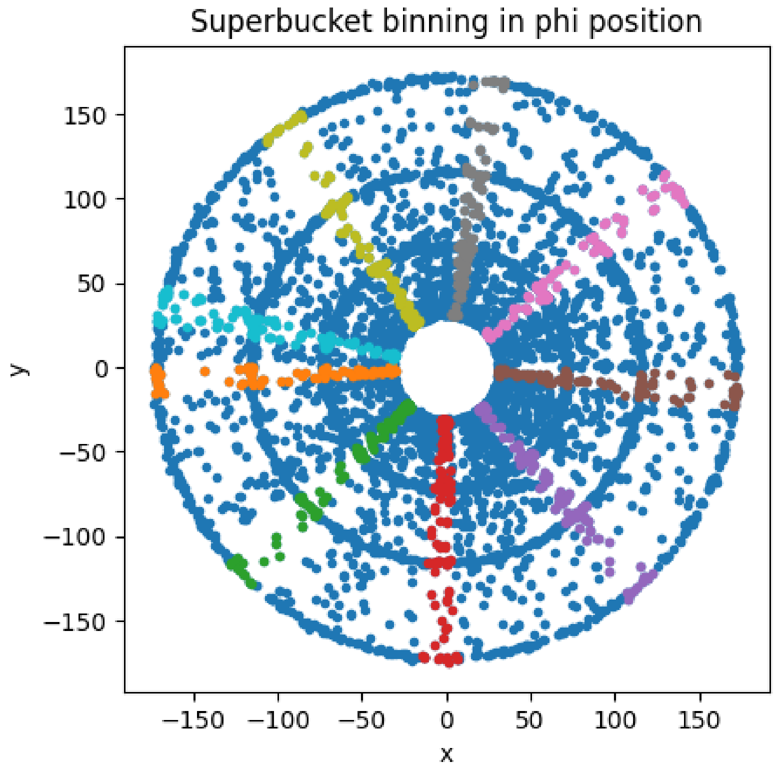



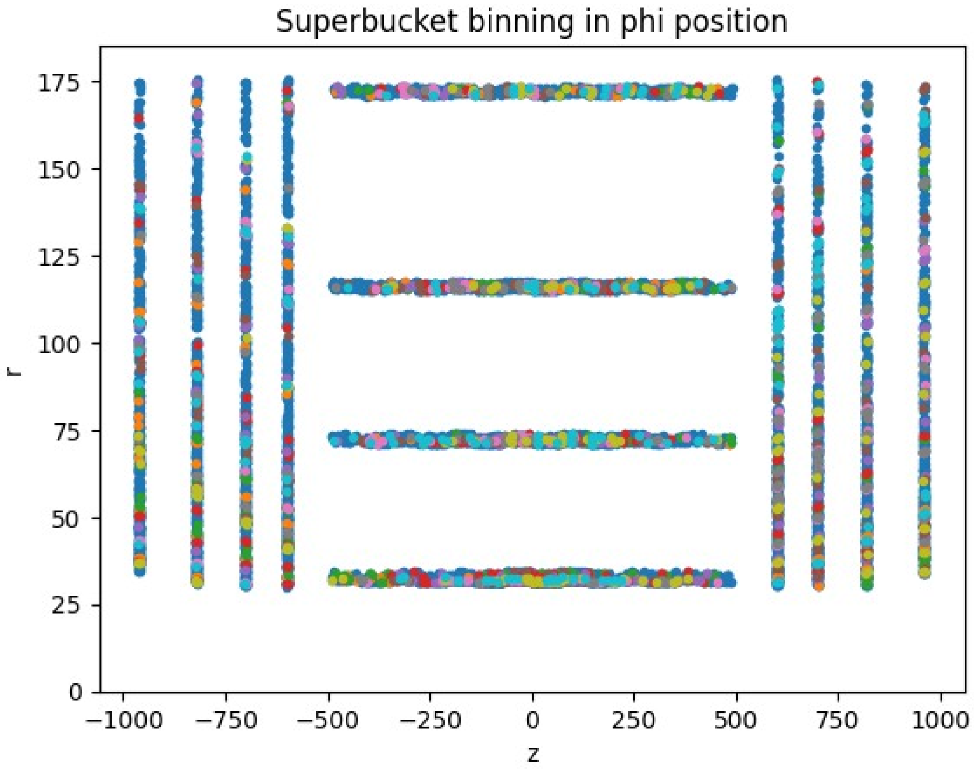
# Superbucket binning in Z position



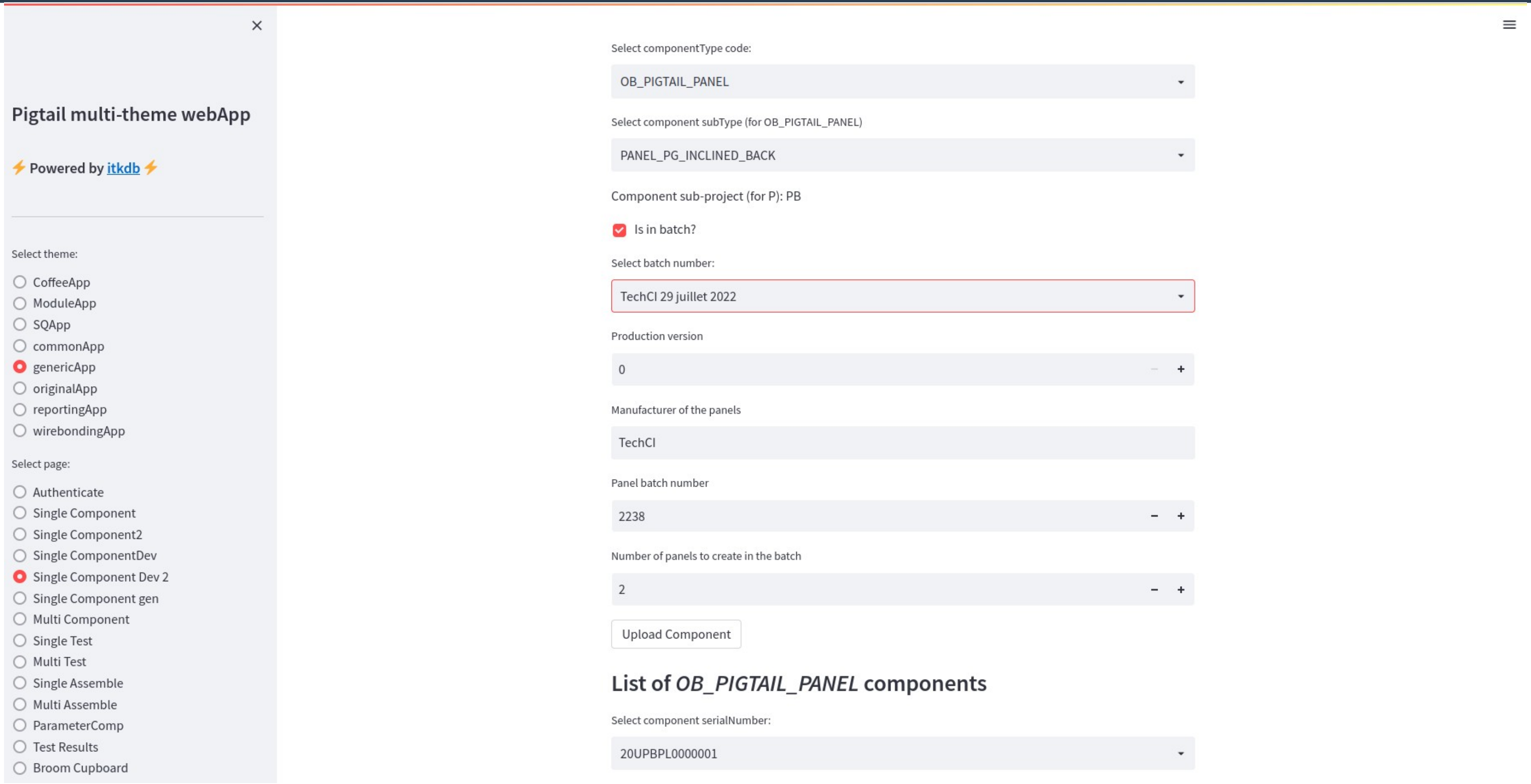



# Superbucket binning in Phi position



# ITk Production Database webapp



# Training hours

# Improving tracking performances with machine learning

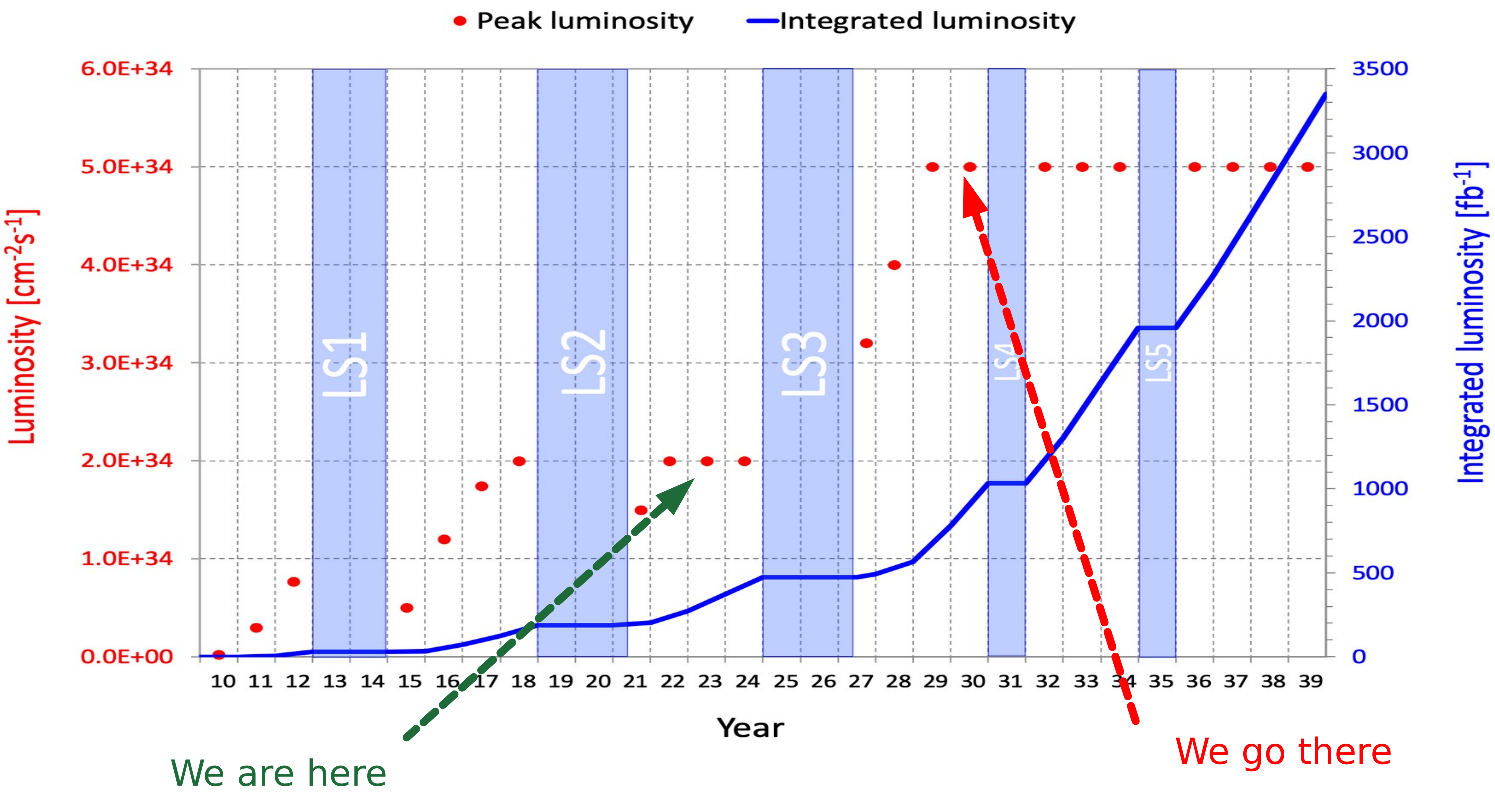
Jeremy Couthures

PhD at LAPP, Annecy
Supervised by Jessica Levêque and Sabine Elles

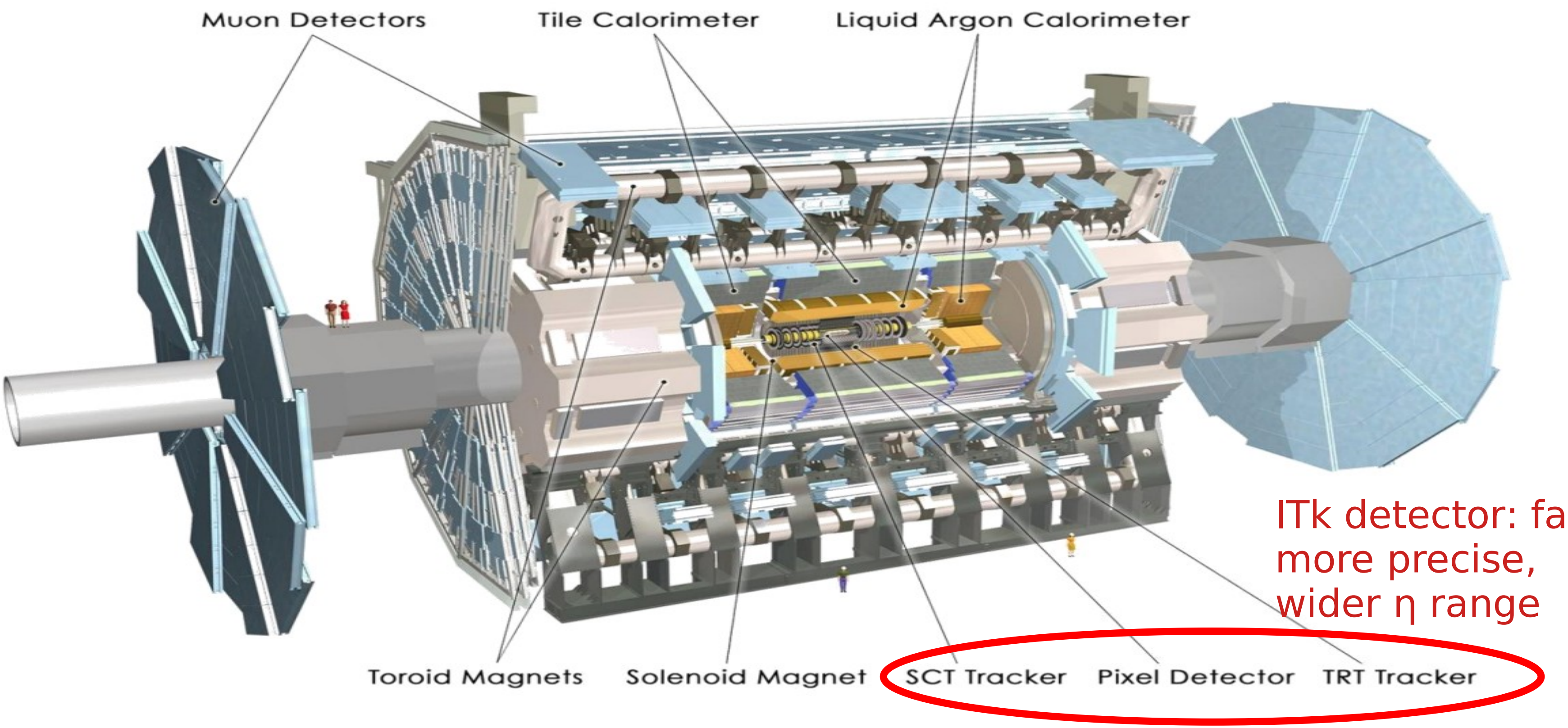
# Overview

- **HL-LHC context**
  - Increase of luminosity
  - Inner Tracker detector
  - Consequences
- **Tracking context**
  - Steps of track reconstruction
  - Combinatorial problem
- **The Hashing step**
  - Annoy
  - Results & discussion

# From LHC to HL-LHC: increase of the luminosity

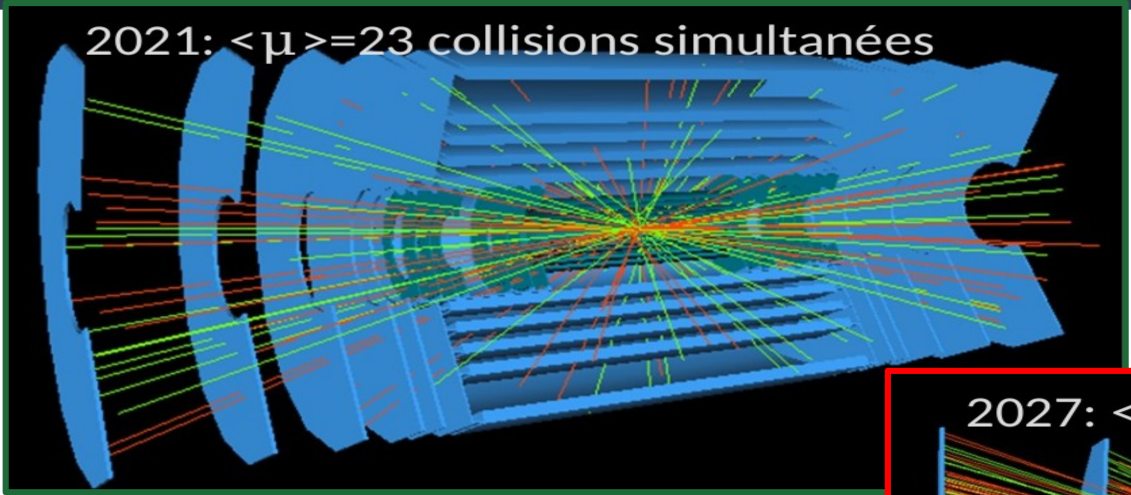


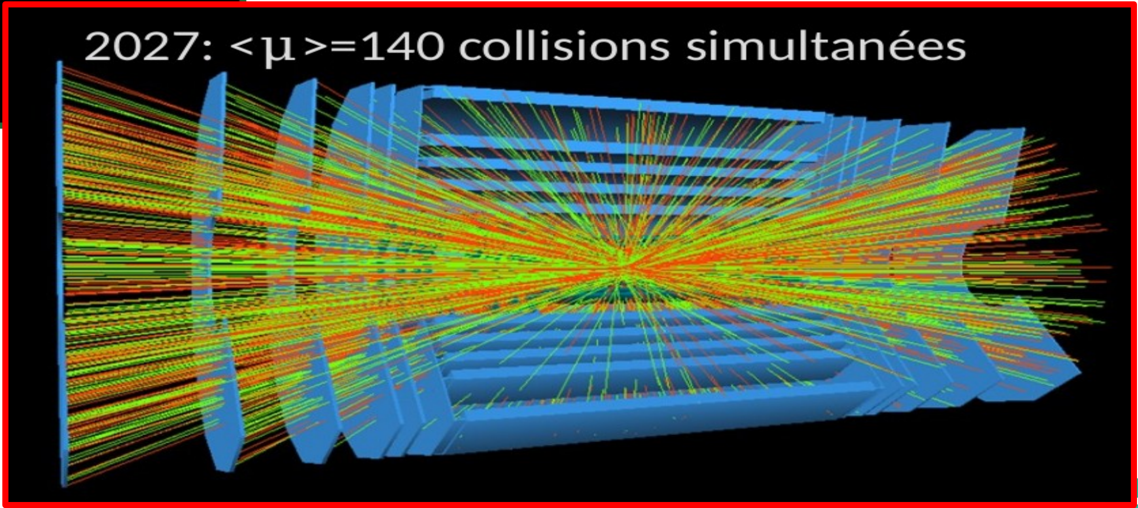
# Inner Tracker detector



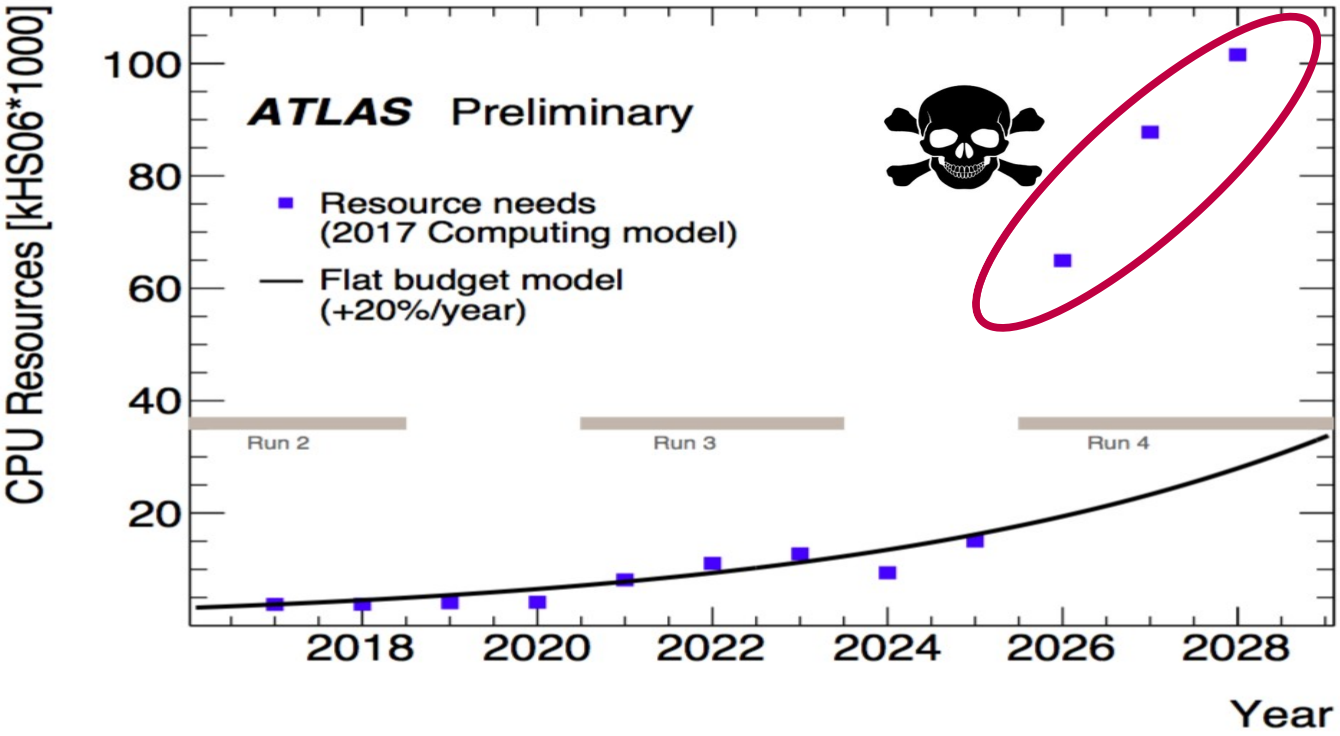

ITk detector: faster, more precise, wider η range

# Consequence 1: problem complexity



# Consequence 2: computing time and budget requirements



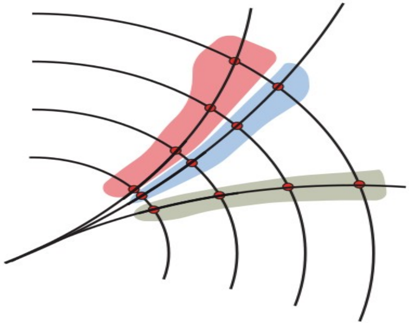

⇒ Improve track reconstruction algorithms

# Overview

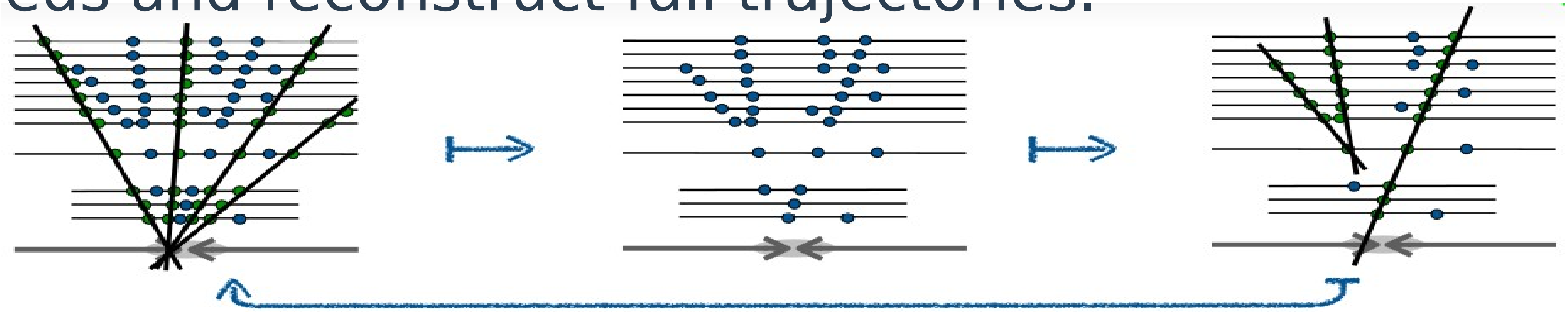
- **HL-LHC context**
  - Increase of luminosity
  - Inner Tracker detector
  - Consequences
- **Tracking context**
  - Steps of track reconstruction
  - Combinatorial problem
- **The Hashing step**
  - Annoy
  - Results & discussion

# The 3 steps of track reconstruction

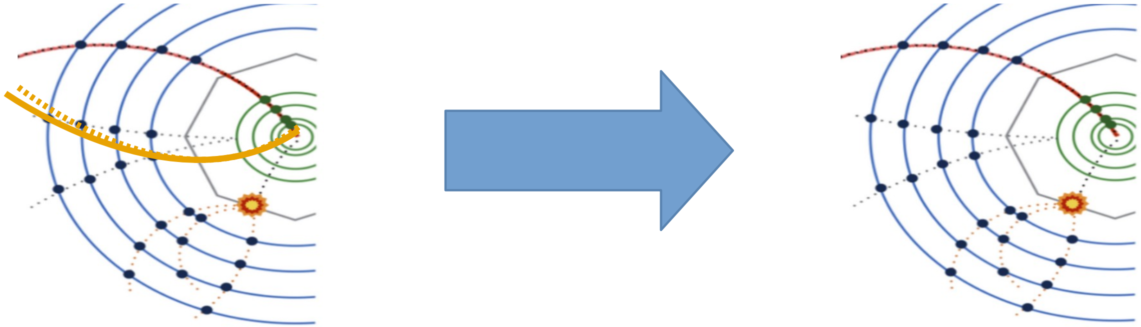
**1. Seeding:** find triplets of compatible points to make a proto-track ("seed").



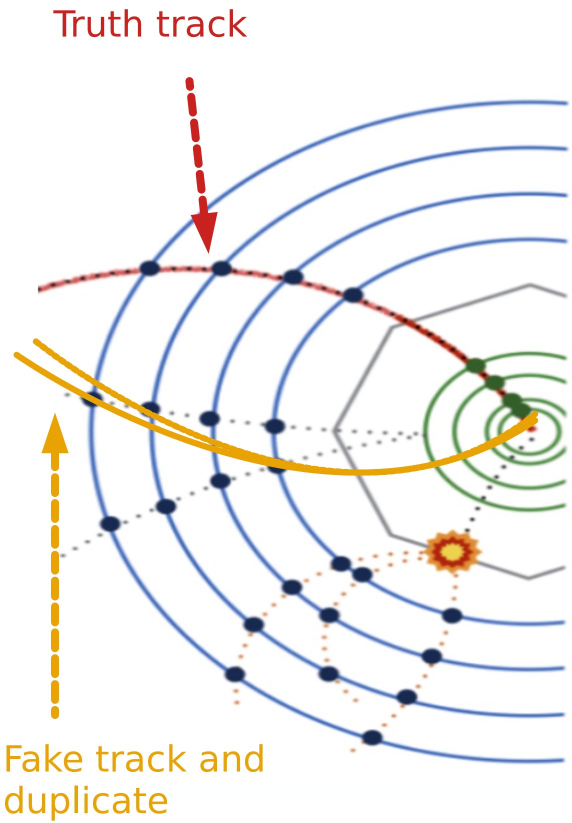
**2. Kalmann Filter:** Iterative process to propagate the seeds and reconstruct full trajectories.



**3. Ambiguity resolver:** remove bad quality tracks and duplicates

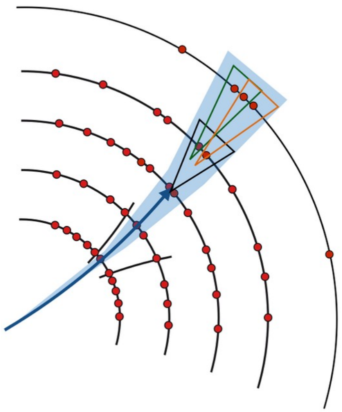


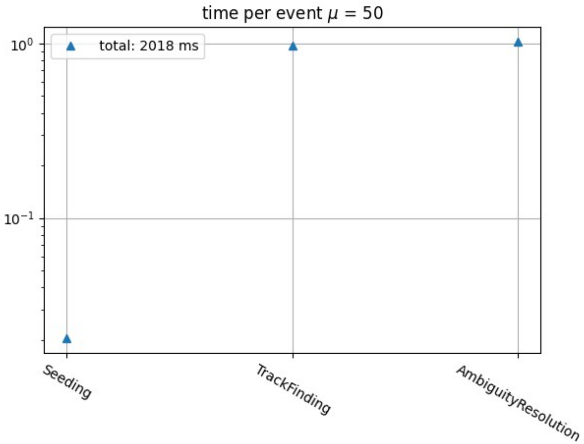
# Improving track reconstruction algorithms




- **Improving?**
  1. Reconstruct highest number of “truth” tracks…
  2. ..while reconstructing lowest number of “fake” tracks (noise)
  3. While avoiding duplicates
  4. And… going faster.

# Combinatorial problem






## Combinatorial Kalman Filter:

- Several possibilities of expanding the seeds at each layer → need to test them all
- Number of combinations increases exponentially with the number of layers

- **Every seed is expanded:**
  - Less seeds → less tracks → less bad quality and duplicated tracks

ACTS Poor man's Ambiguity resolver

## How to get less seeds?

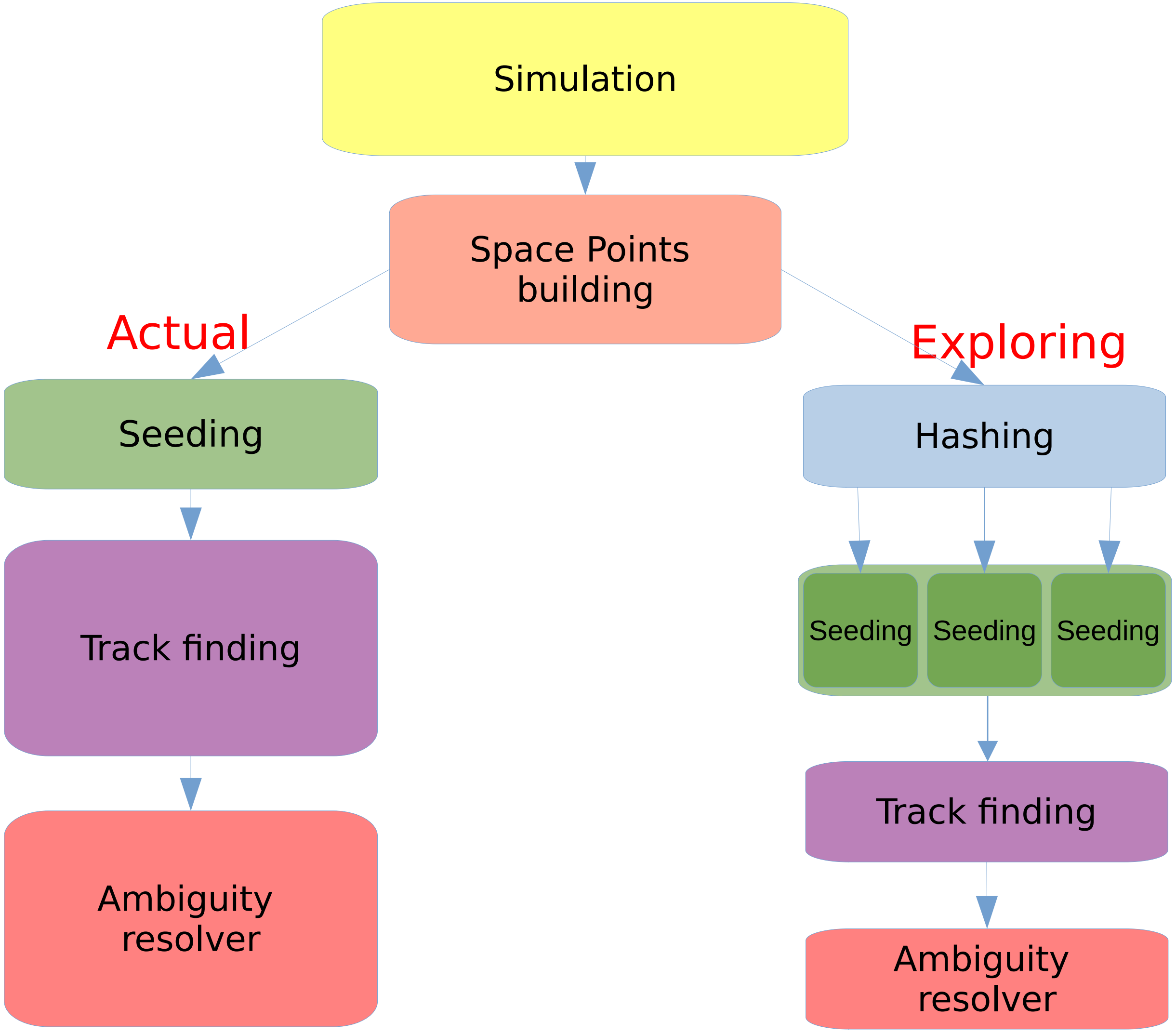
→ Remove the bad ones!

- How?
  - Current: Filter the seeds + detailed optimisation
  - My work: Build the seeds differently

# Overview

- **HL-LHC context**
  - Increase of luminosity
  - Inner Tracker detector
  - Consequences
- **Tracking context**
  - Steps of track reconstruction
  - Combinatorial problem
- **The Hashing step**
  - Annoy
  - Results & discussion

# Approaches




- Seeding parallelization
- Hashing groups space points into buckets
- Hashing reduces the number of space points at a time (focus on relevant space points) → less seeds per bucket

# The Hashing Step

**Hashing:**

1. Group space points into buckets
2. Do the seeding on each bucket to reduce the number of seeds given to the Track Finding algorithm

**Algorithm used:**

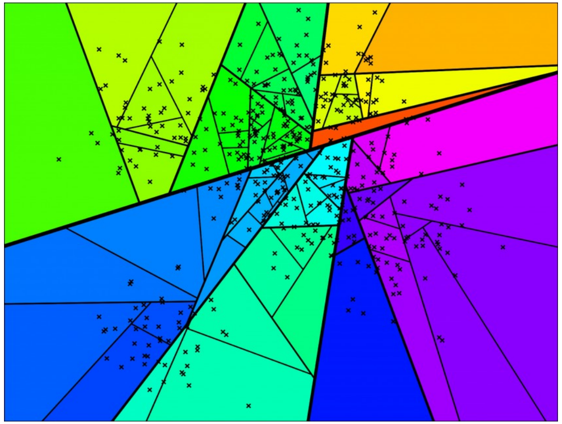
Approximate Nearest Neighbors Oh Yeah (**Annoy**)
→ Used by Spotify

- Machine Learning algorithm type:
  - k Nearest Neighbors (unsupervised)
  - Random based

- Number of Neighbors (bucket size)

- Use the distance between the points
  → need to define a (relevant) metric

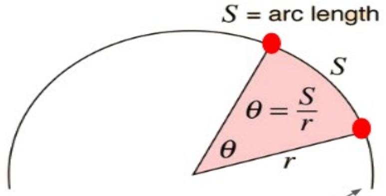
Space separation by Annoy



# Metric

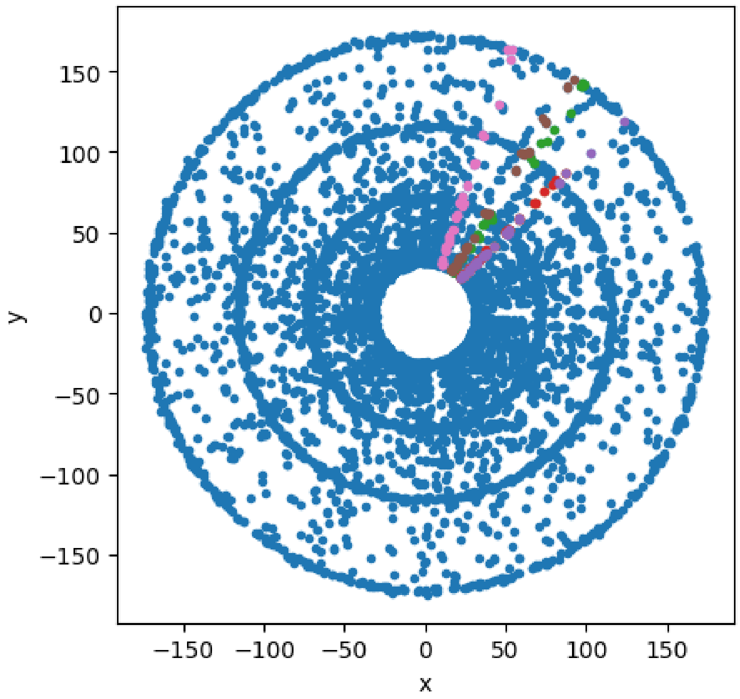
Metric used : <u>angular distance</u>

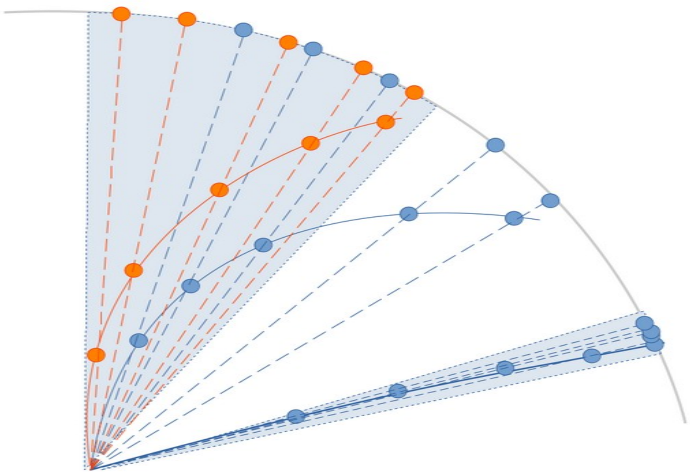
$$\theta = S / R$$




where S = distance travelled and R = radius of the circle

Δφ



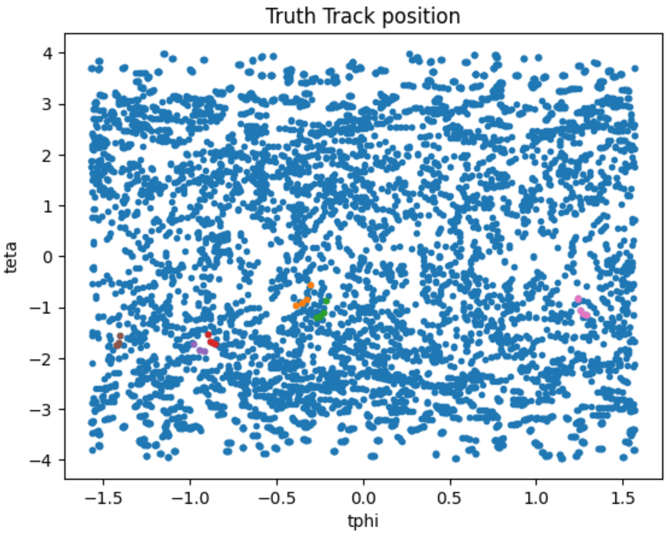



High pT track ~ linear track: all the hits are expected to fall in the same bucket

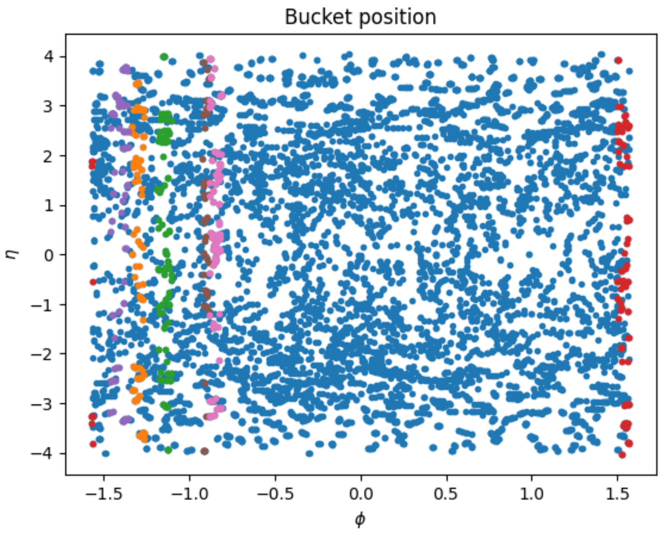
Low pT tracks at high μ: Buckets may contain mixed hits from several tracks → efficiency loss

# Using other metrics?

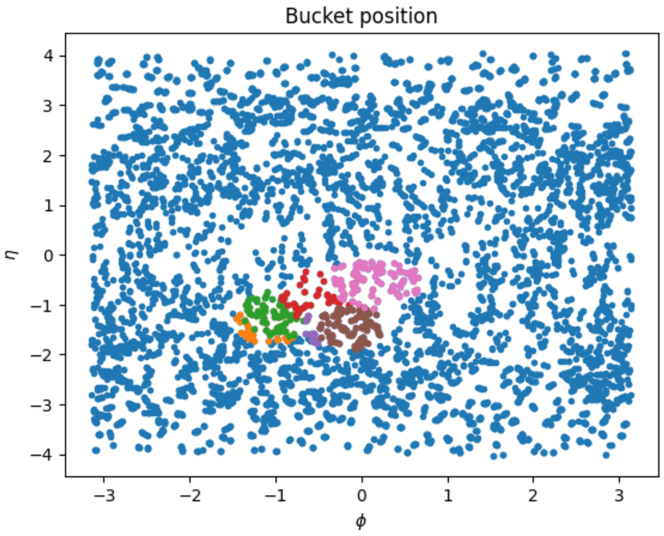
Truth Tracks (hits)




Angular: Δφ




ΔR



# Testing setup

Generic detector: (toy detector)

acts github

100 tt events

$|\eta| \leq 4$

pT > 1GeV

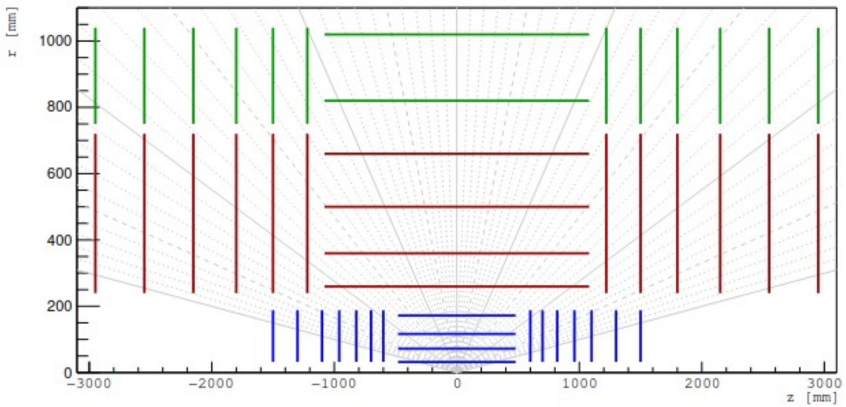
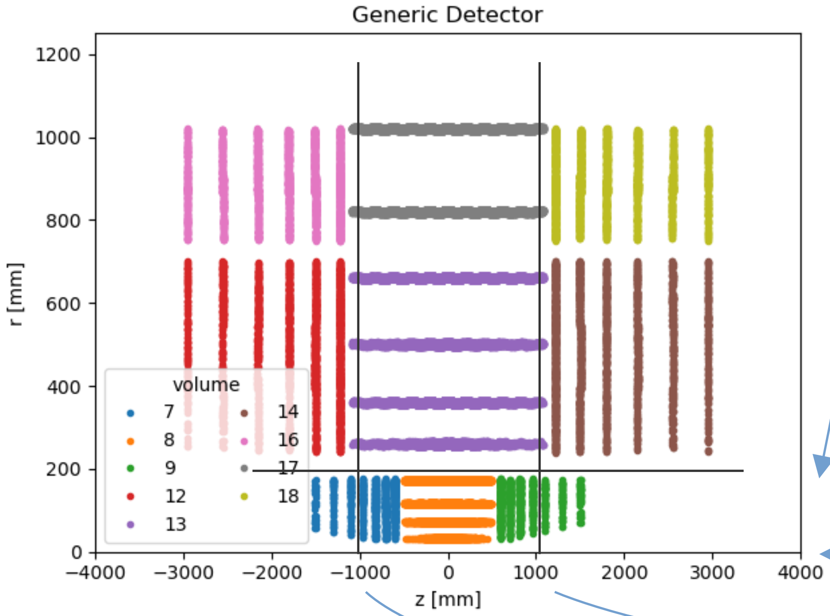


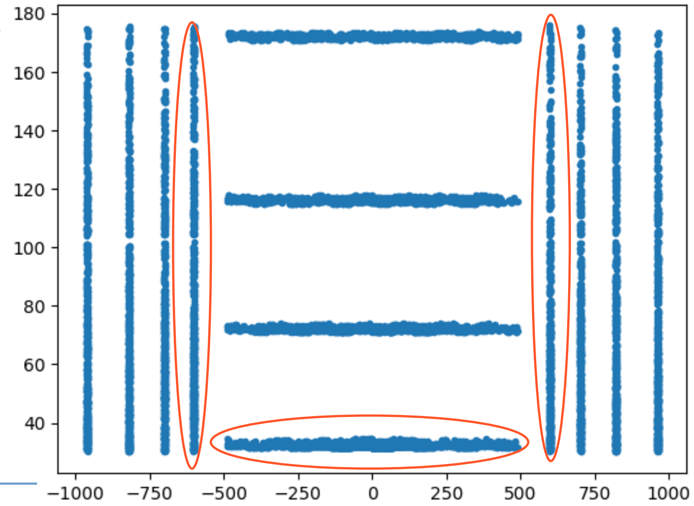

**Fig. 1** Sketch of the TrackML detector as used in both the "Accuracy" and "Throughput" phase. Vertical lines indicate disks while horizontal lines indicate cylinders, all with the *z* axis as axis of revolution. Three different sub detectors build the overall detector setup: a central pixel system (blue), enclosed by first a short strip (red) and then a long strip detector (green).

https://arxiv.org/pdf/2105.01160.pdf

# Generic Detector: Space Points

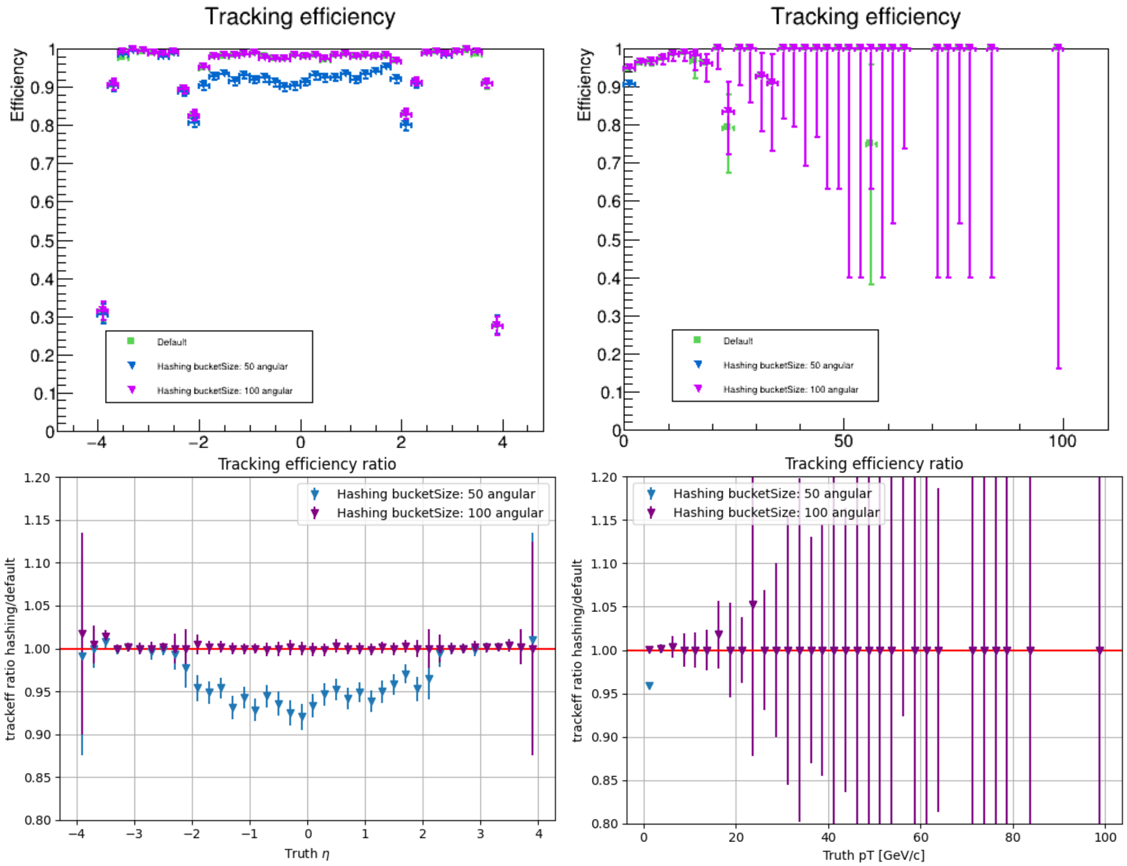



Space points position:



➡ Hashing currently uses only Pixel Space Points
Buckets are built from Space Points of layers 0

# Performance μ = 50 Δφ metric



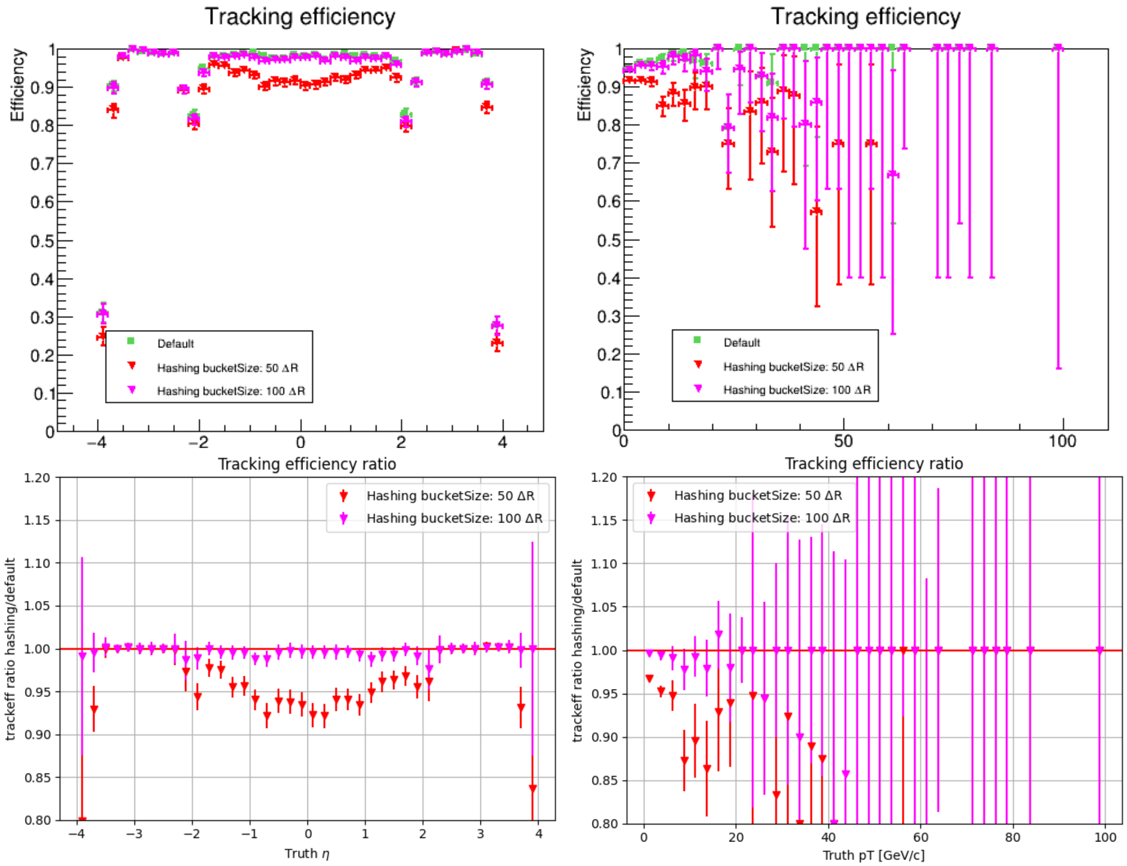

Bucket size 50:
low pT are not well reconstructed

Loss of efficiency in the central region
Better efficiency in the forward region

Bucket size 100:
low pT are well reconstructed

Better efficiency in the central region
Better efficiency in the forward region

# Performance μ = 50 ΔR metric



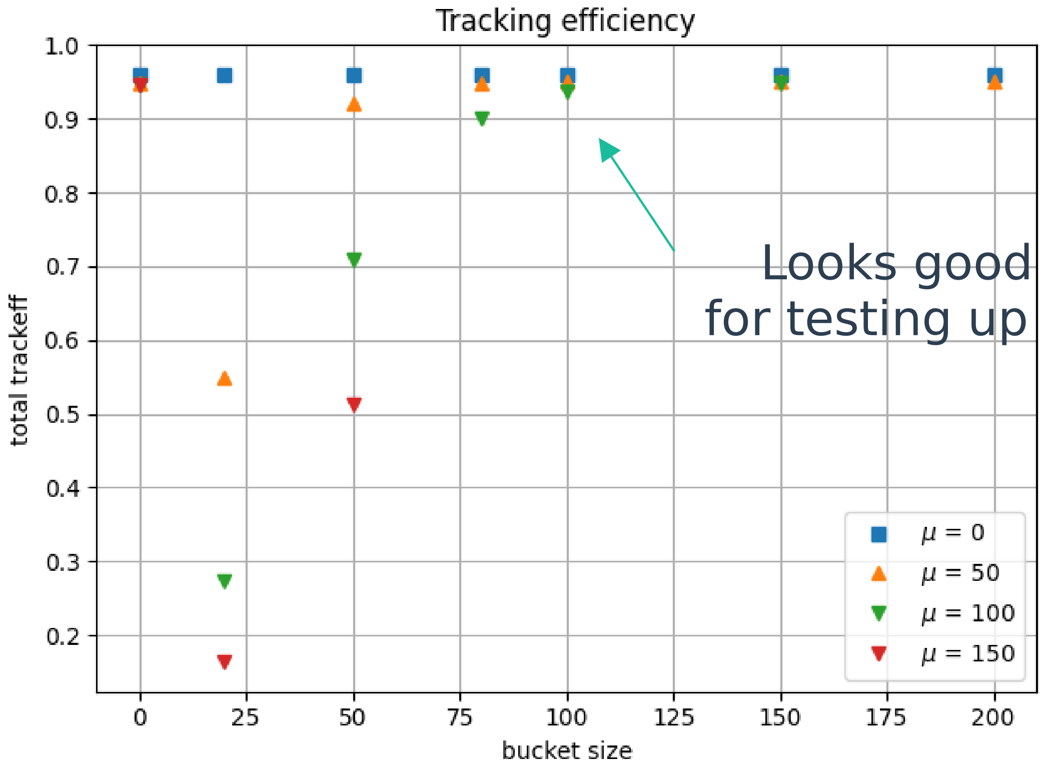

Bucket size 50:
low and high pT are not well reconstructed

Loss of efficiency in the central region
Better efficiency in the forward region

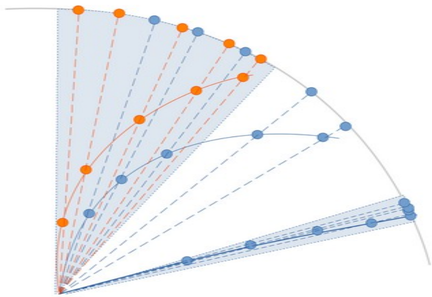
Bucket size 100:
low and high pT are reconstructed

Loss of efficiency in the central region
Better efficiency in the forward region

# Optimal bucket size

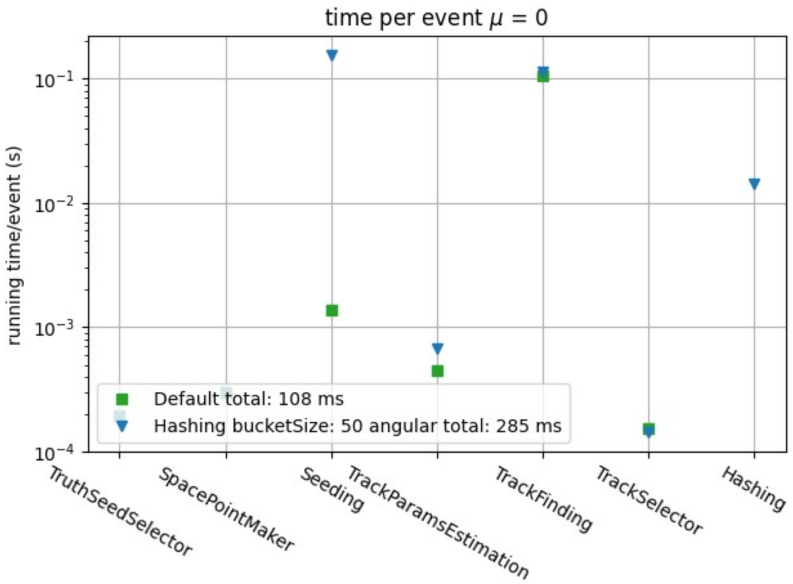


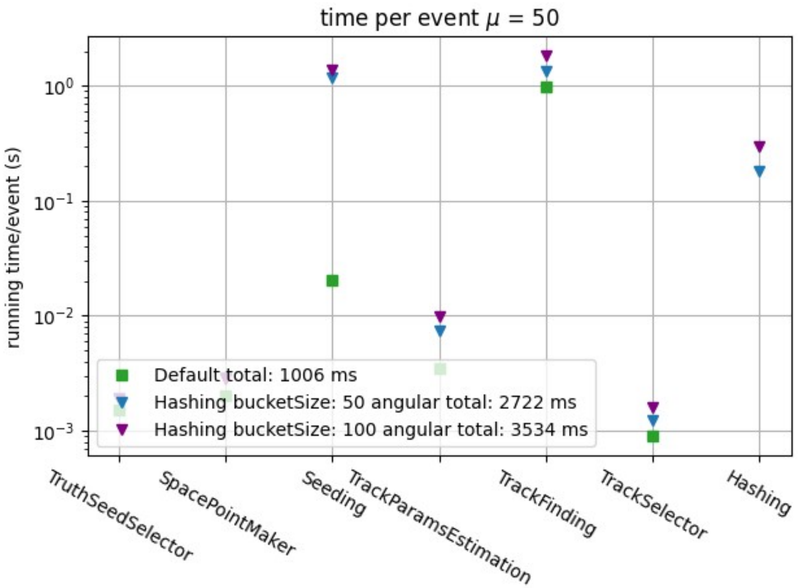

Looks good enough
for testing up to μ = 100



# CPU time

1 Space Point $\Rightarrow$ 1 Bucket



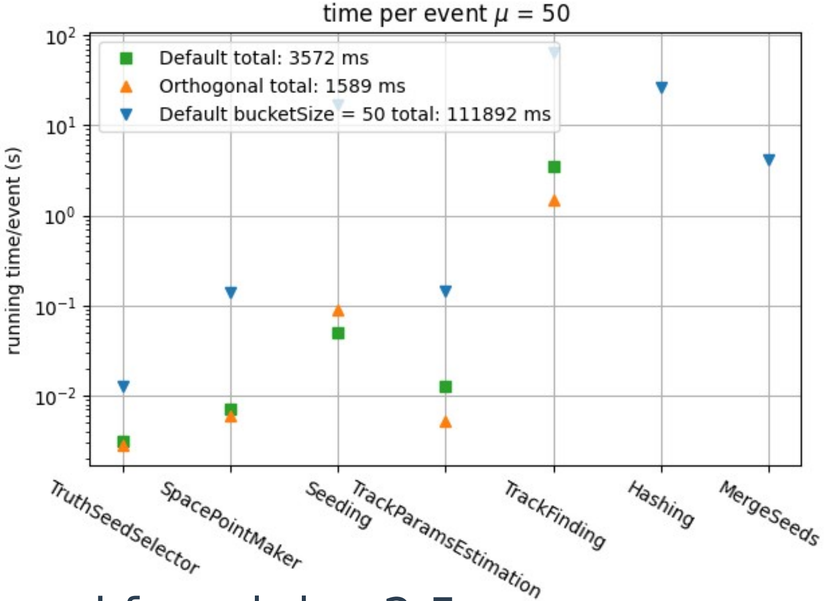




Mean seeding time per bucket (size = 20): 0.9 ms (constant with μ)

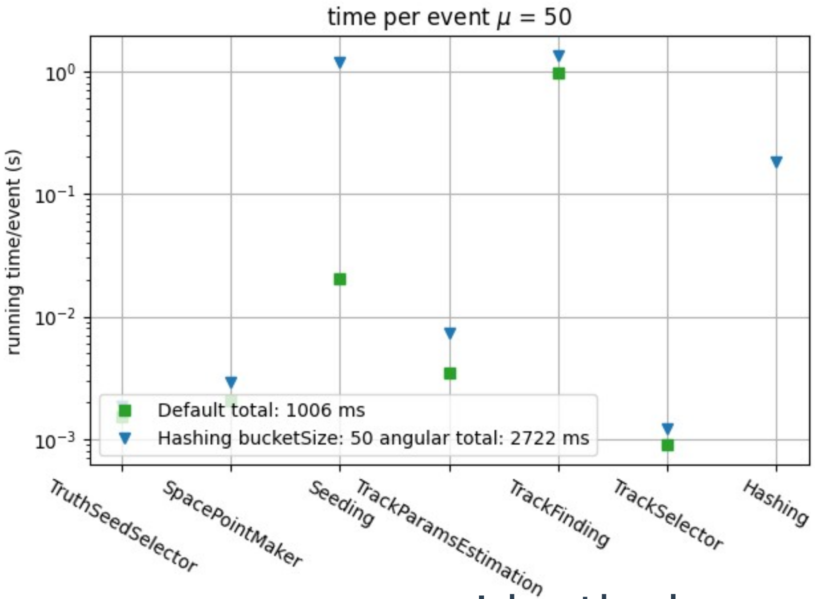
Lot of overlap between buckets $\Rightarrow$ reduce number of buckets to reduce total time

# Recent speedups

Before




After




Changed from $|\eta| \leq 2.5$ to $|\eta| \leq 4$

High speedup from removing Identical buckets

Use only SPs from layer 0 to create the buckets

Identical seeds are removed using a set

# What's Next?

- Test other metrics
- Make the seeding algorithm create seeds only once
- Reduce the number of buckets, and their overlap
- Use different bucket size in center and forward regions? Different metrics?
- Focus on QT tasks!

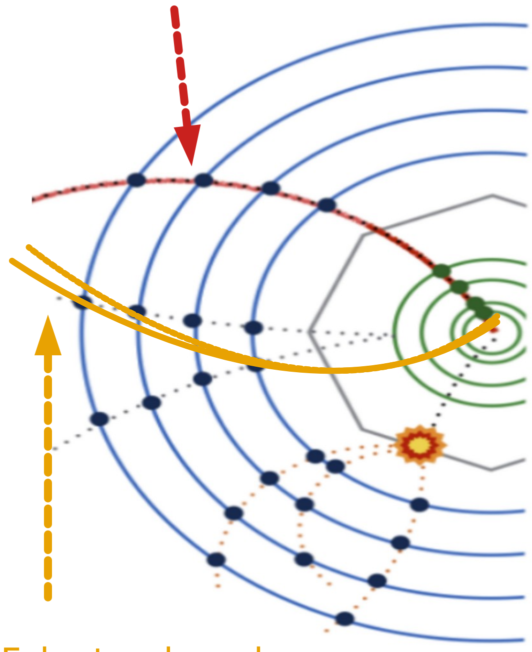
# Qualification Task(s)

- **ITk production Database:**
  - Build dedicated web applications for easy registration of components and tests in the database

- **Athena seeding related QT. Under discussion.**

# Backup

# Evaluation of performance




**Physics:** (Kalman Filter performance)

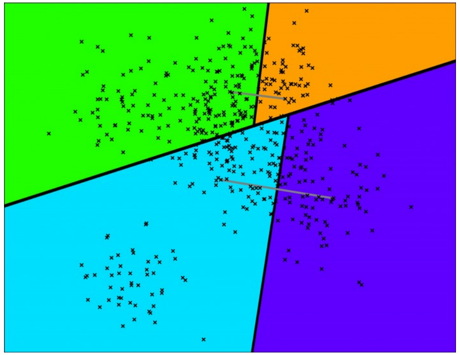
- Efficiency = $\frac{\#\ tracks\ matched\ to\ a\ truth\ particle}{\#\ reconstructible\ particles\ (>1GeV, >9\ hits)}$
- Fake rate = $\frac{\#\ tracks\ not\ matched}{\#\ reconstructed\ tracks}$
- Duplicate rate = $\frac{\#\ reconstructible\ particles\ with > 1\ track\ match}{\#reconstructible\ particles\ (>1GeV, >9\ hits)}$

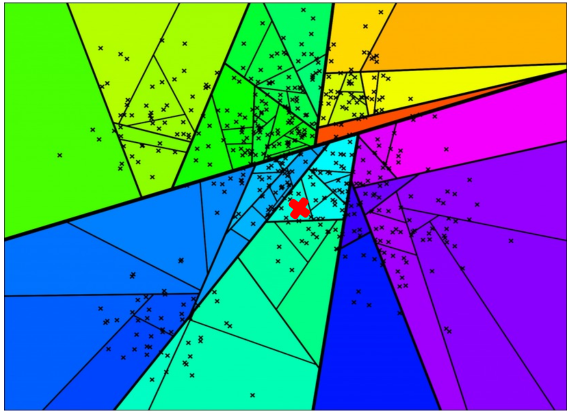
**Computing:**

- Monitoring of CPU time

# Annoy training

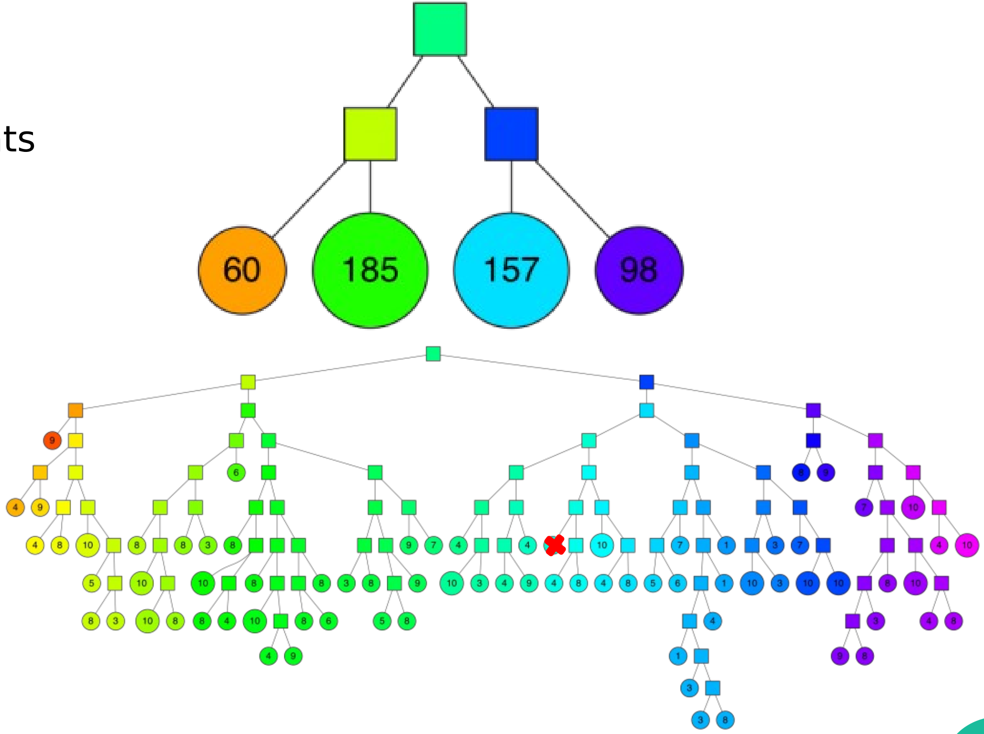
Space separation



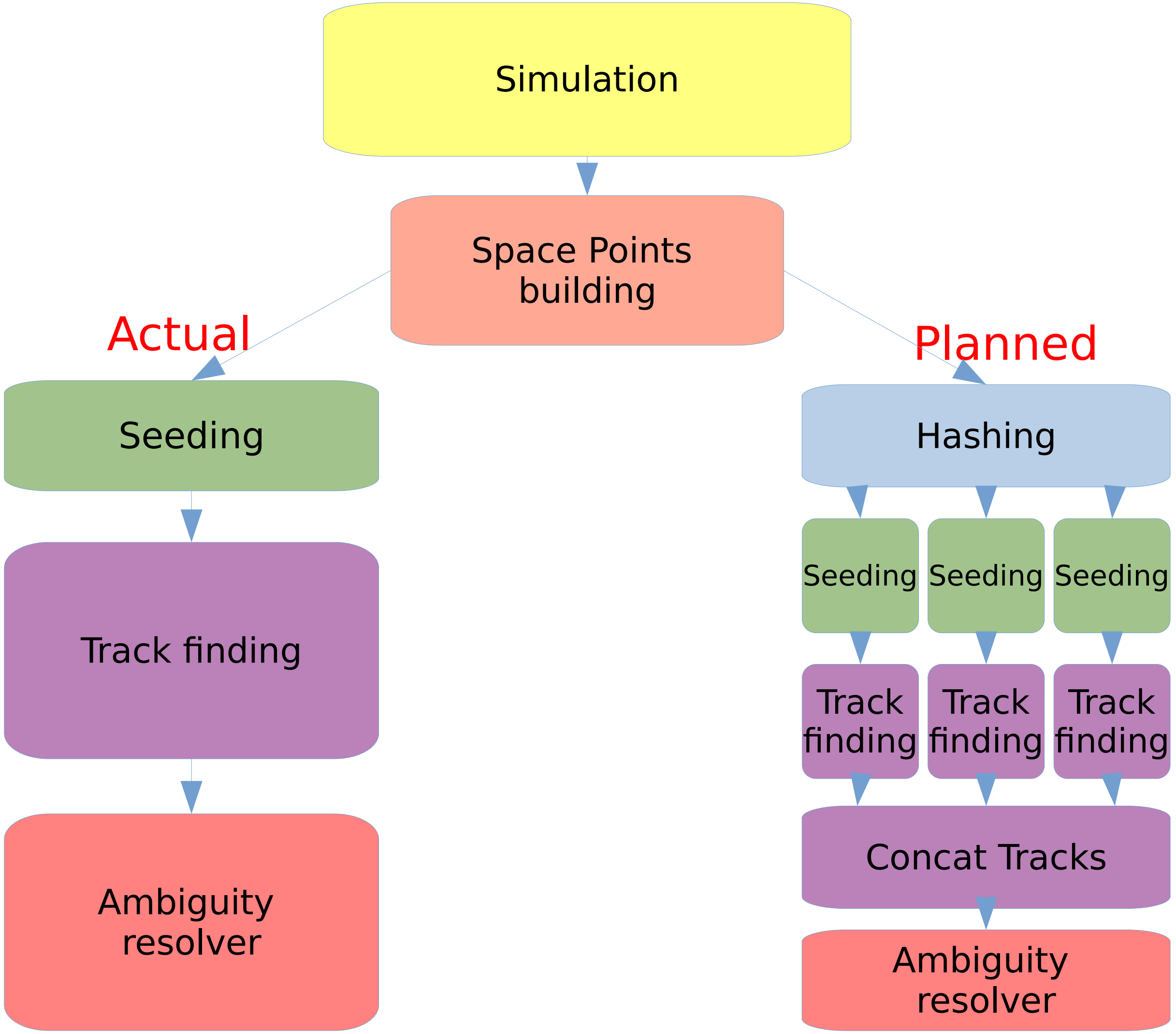


Takes two random points iteratively

Corresponding binary tree

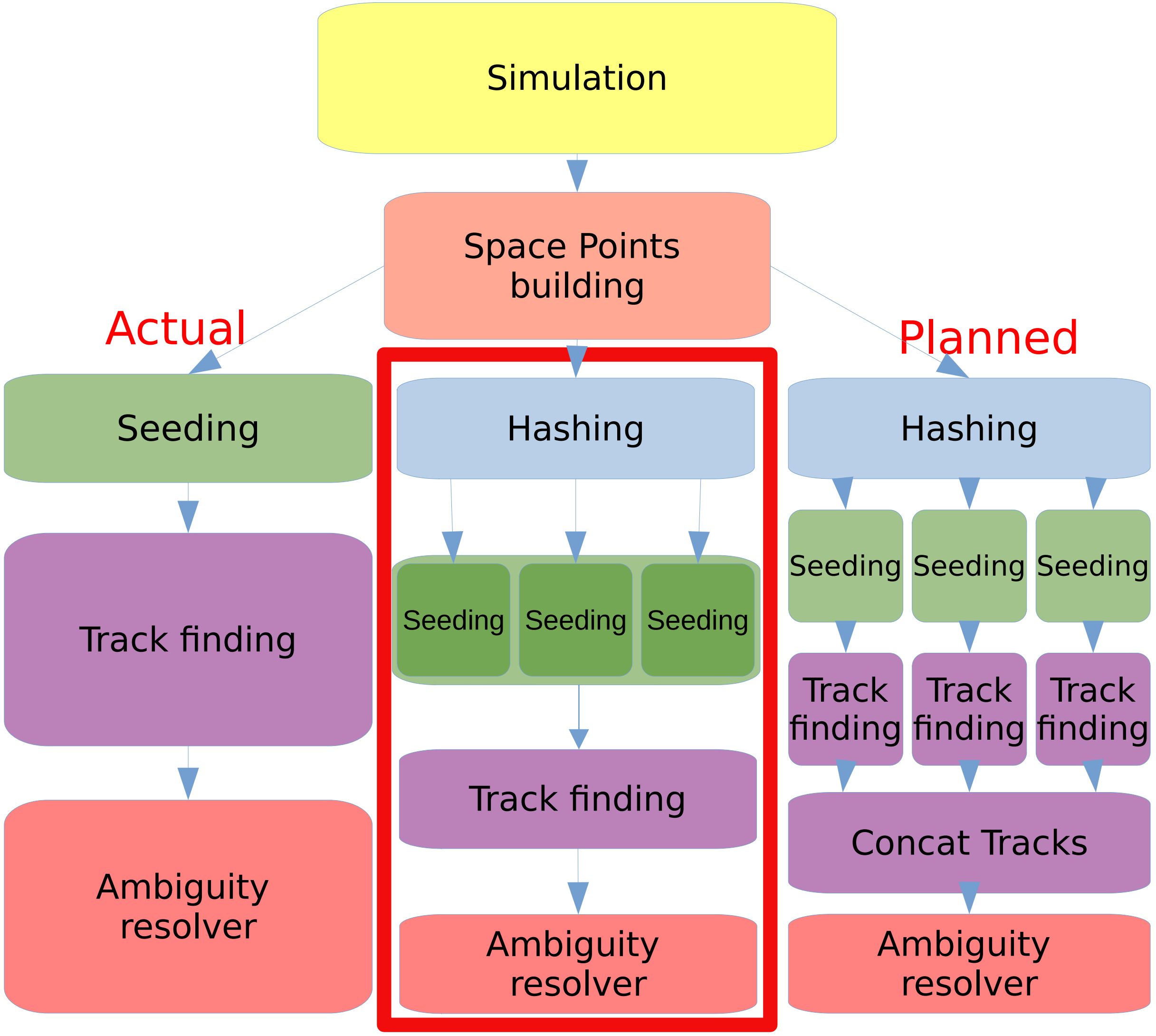


# Approaches




- Full parallelization
- Hashing reduces the number of space points at a time (focus on relevant space points)
  → less seeds per bucket and less possible expand combinations

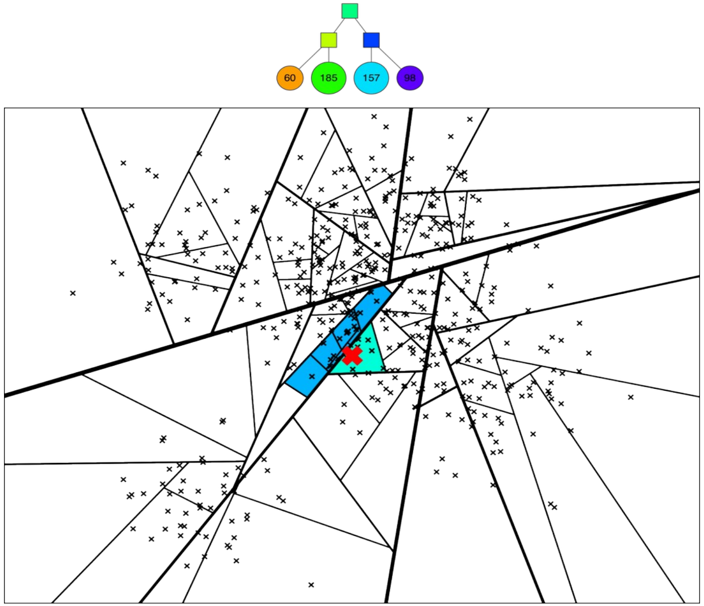
# Approaches



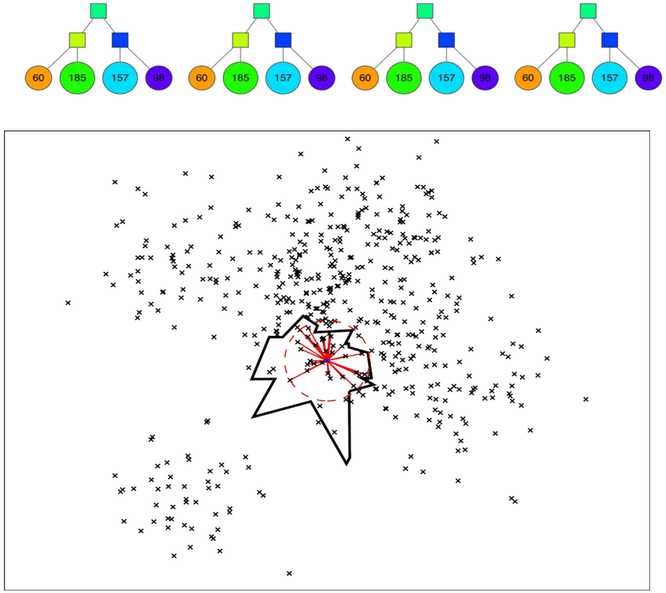

- Seeding parallelization
- Hashing reduces the number of space points at a time (focus on relevant space points) → less seeds per bucket

# Annoy query

Approximation




Merge neighbor subspaces




Union of trees' subspace

- Annoy tuning parameters: number of neighbors, number of trees, metric used, features used, number of subspace to look at