ONLINE TRACKING IN PANDA

PANDA TRACKING WORKSHOP 2018

18.9.2018 | TOBIAS STOCKMANNS

Hough Transformation

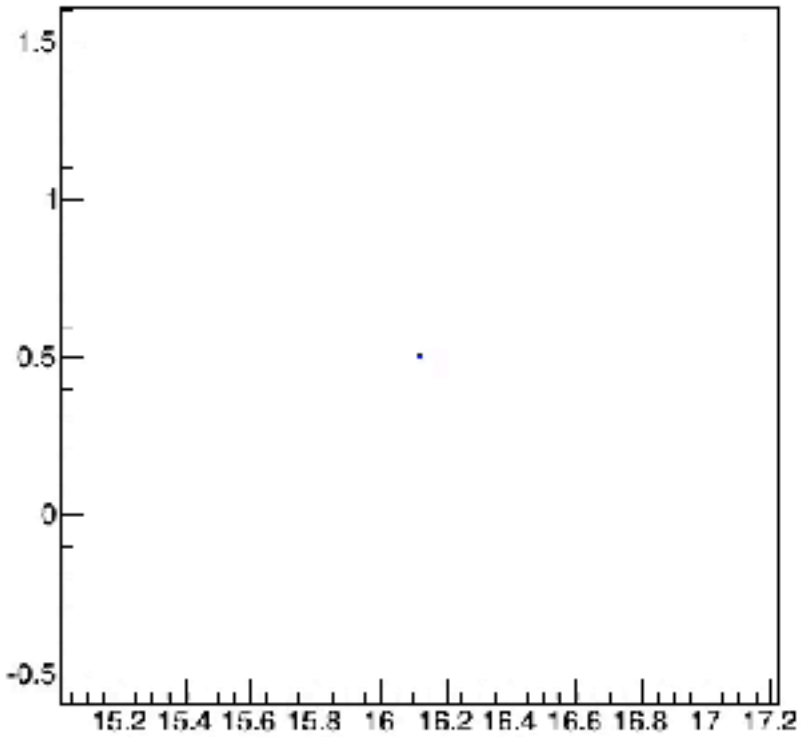
Hough Transform — Visualization Lines

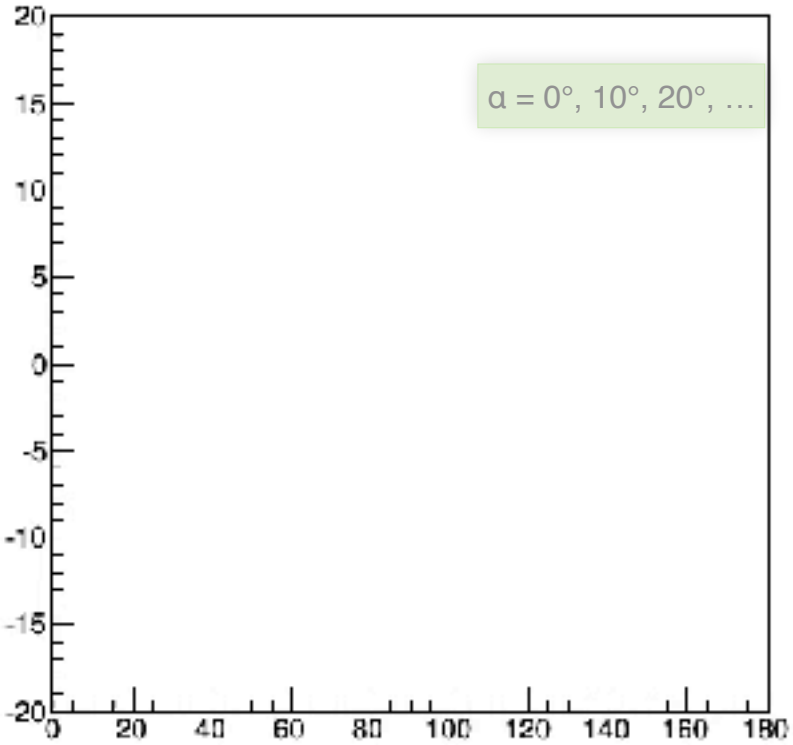
Create lines going through hit point (x,y)i

- **Line parameterized by rij = cos(αj) · xi + sin(αj) · yi**

Fill line parameters (α,r)ij into histogram

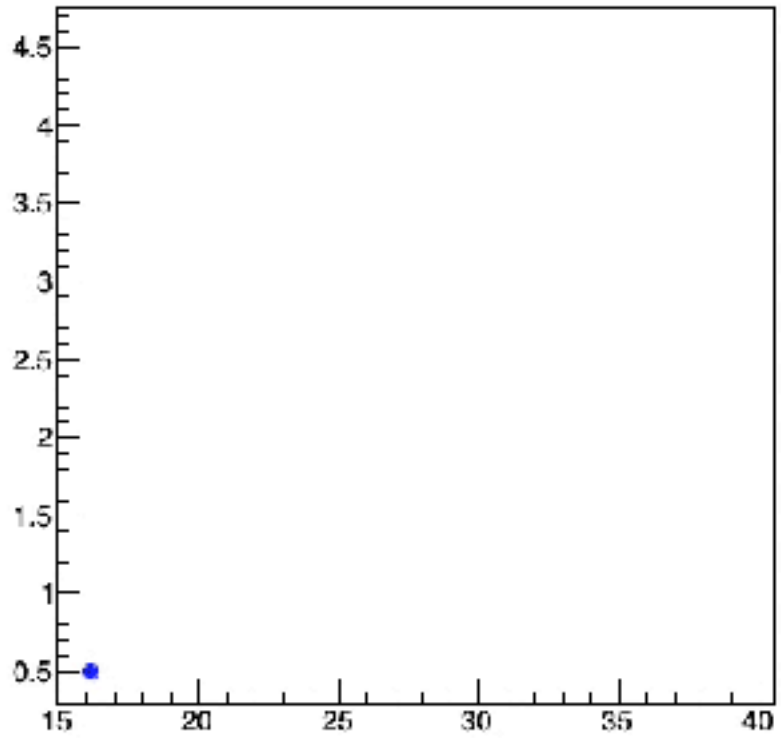
- **Rasterize for many αj ∈ [0°,180°)**

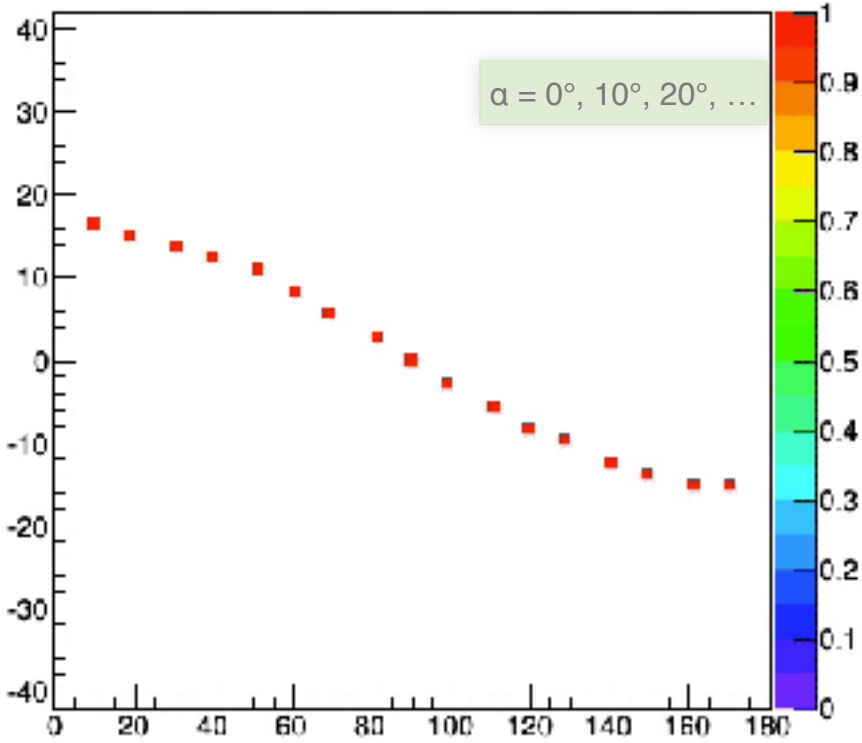




Hough Transform — Visualization Points

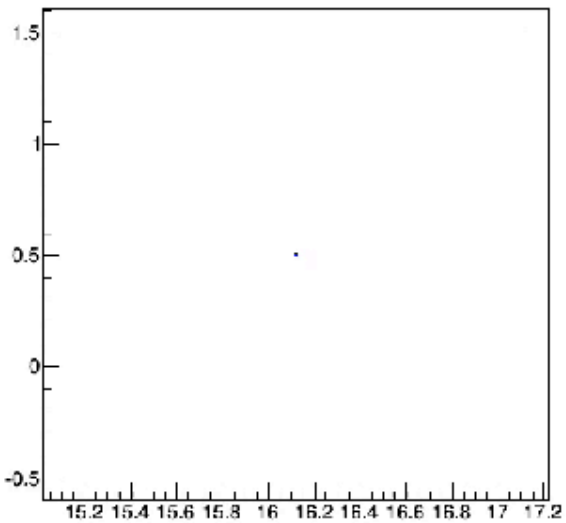
Create lines going through hit point $(x,y)_i$
Repeat for every hit point i

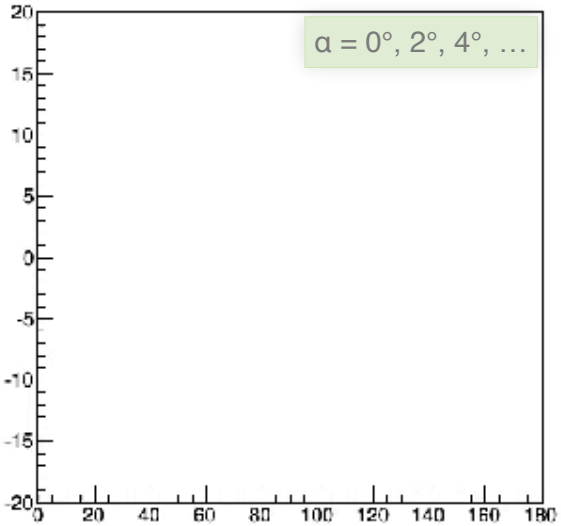




Hough Transform — Granularity

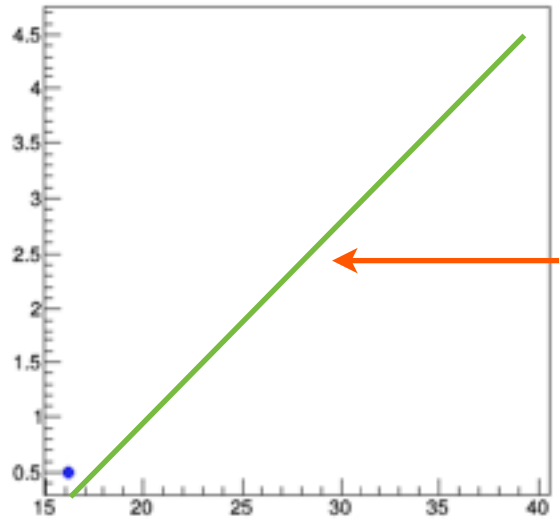
Choice of α granularity determines resolution

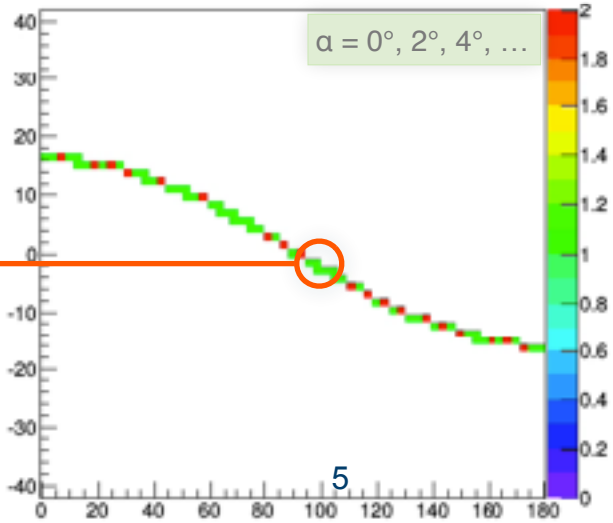




i: ~100 hits/event (STT)
j: every 0.2°

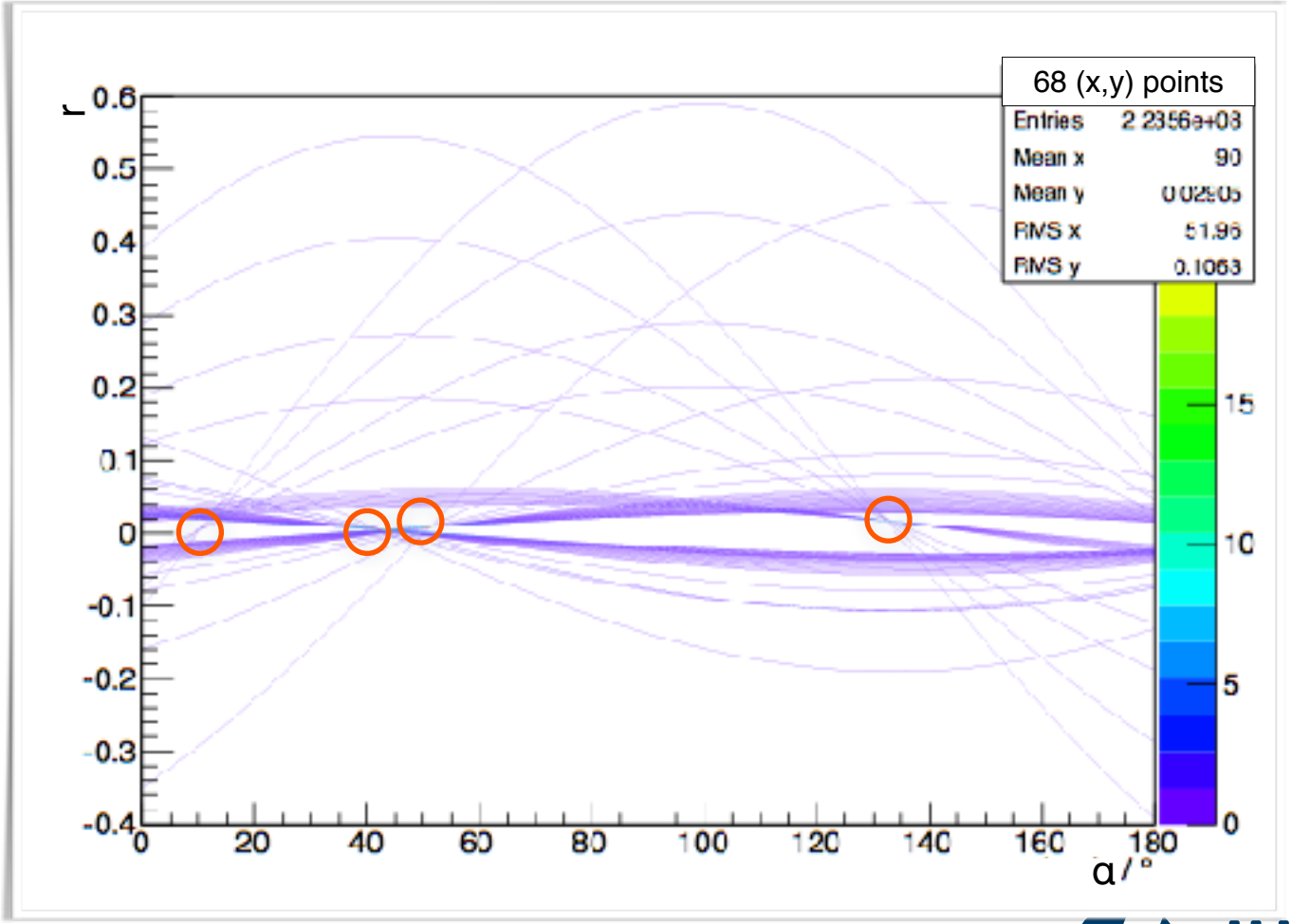
r_{ij}: 180 000





JÜLICH Forschungszentrum

Algorithm: Hough Transform



Hough Transform — Remarks

Two Implementations

Thrust *(CUDA's STL)*

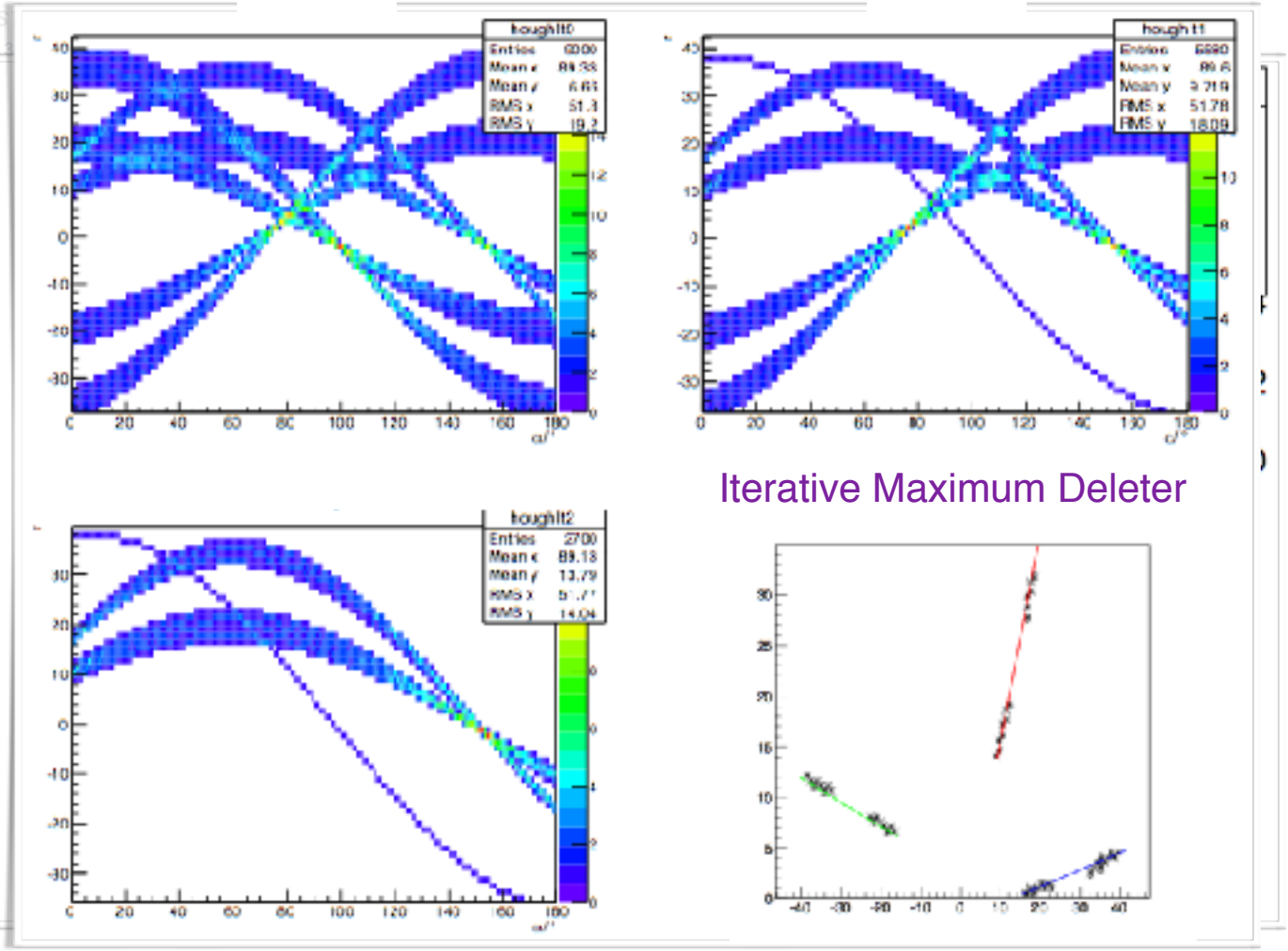
- Performance: 3 ms/evt
- Reduce to set of standard routines
  - Fast (uses Thrust's optimized algorithms)
  - Inflexible (hard to customize)
  - Not yet at performance maximum

Plain CUDA

- Performance: 0.5 ms/evt
- Build completely for this task
  - Fitting for PANDA; customizable
  - A bit more complicated at part

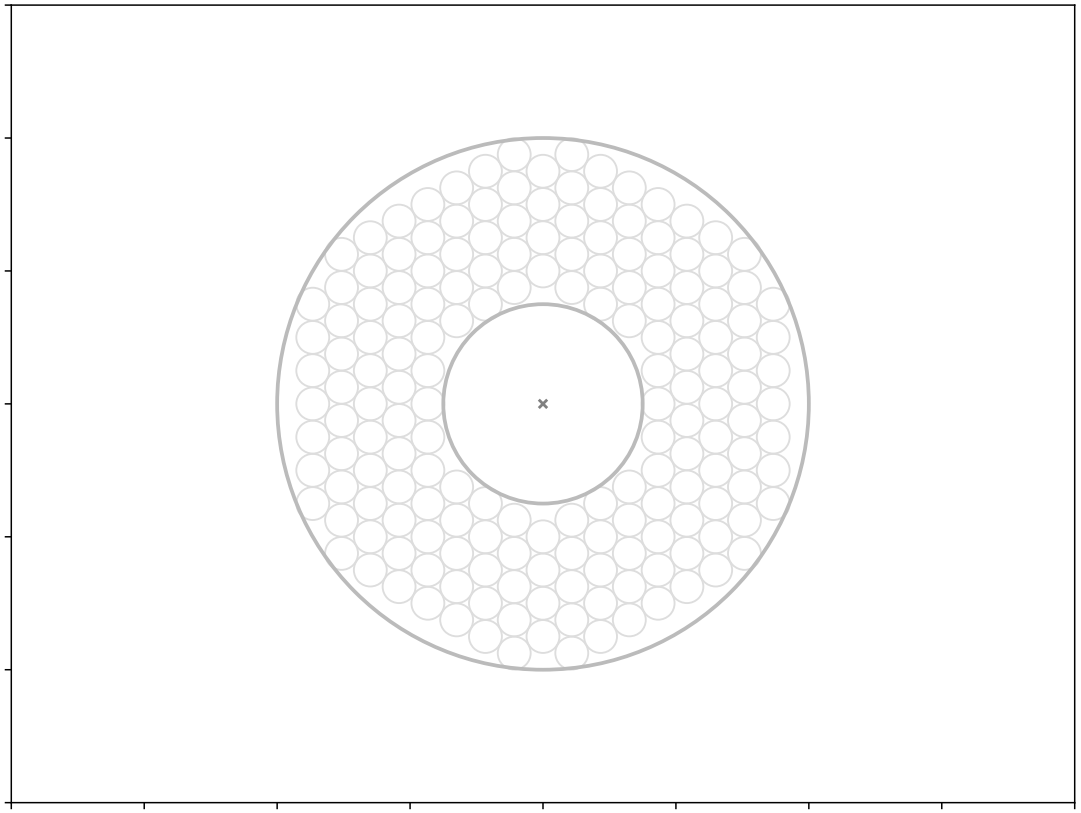
Peakfinding challenging

Hough Transform — Remarks

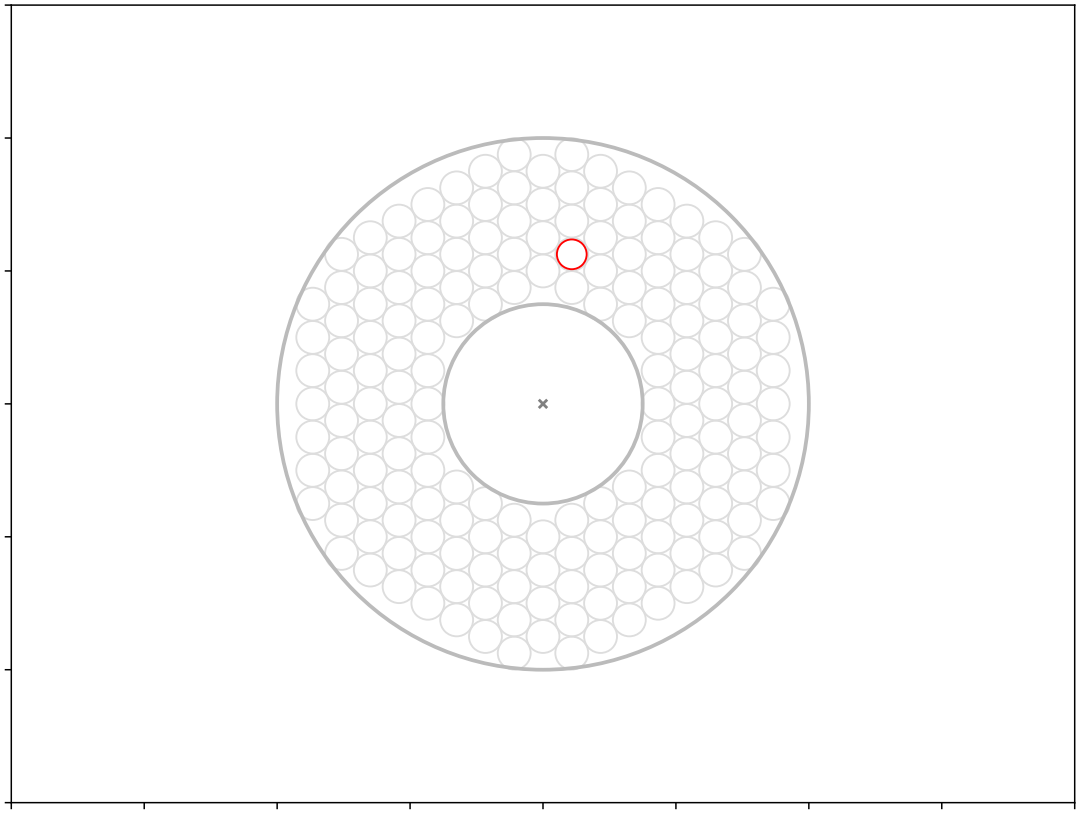


Peakfinding challenging

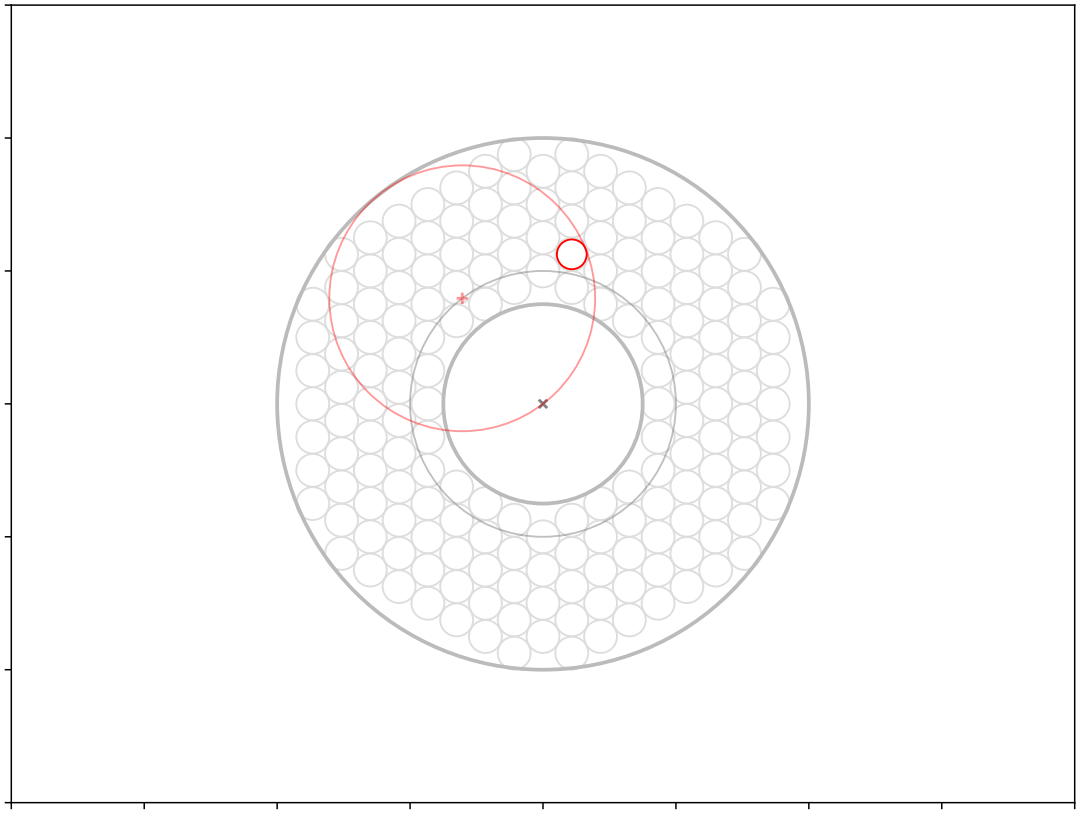
CIRCLE HOUGH



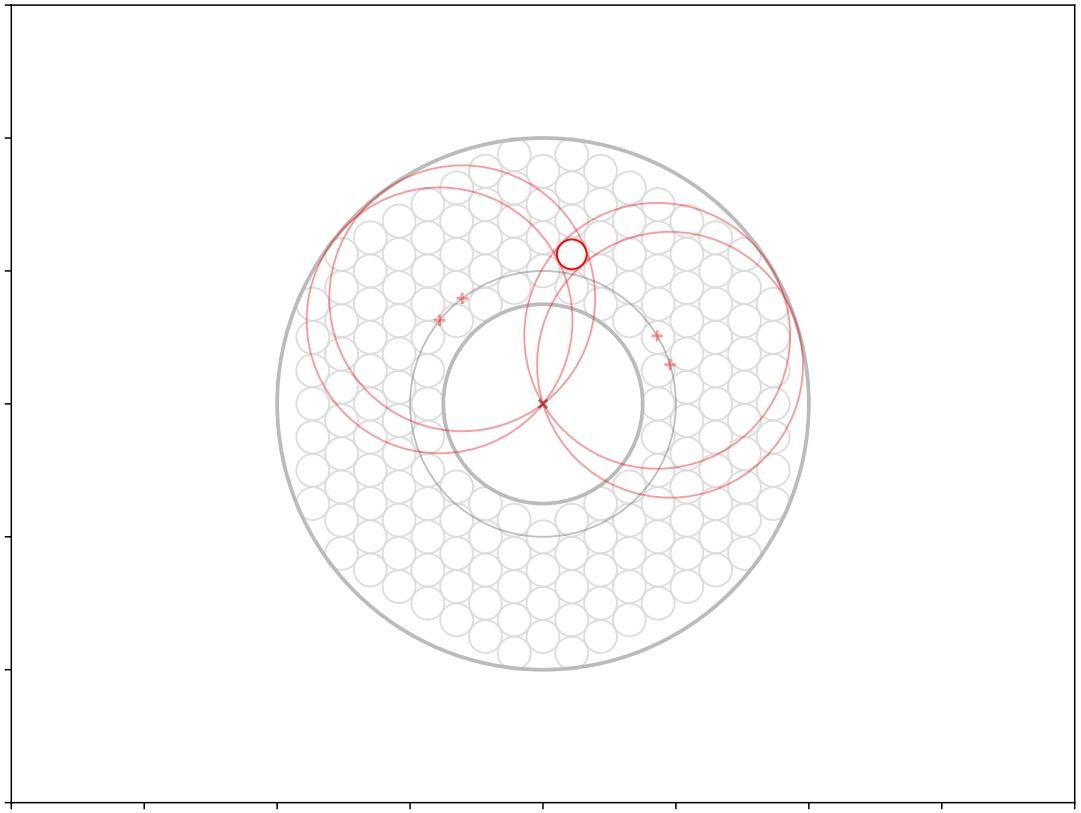
CIRCLE HOUGH



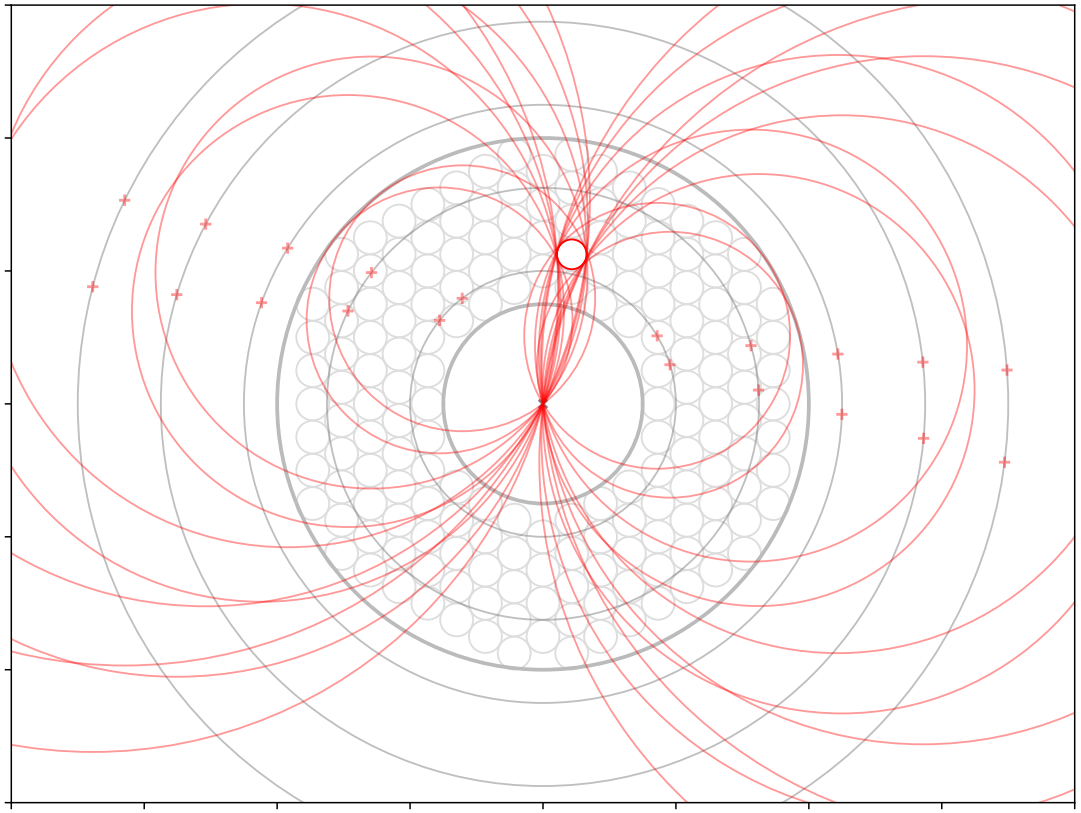
CIRCLE HOUGH



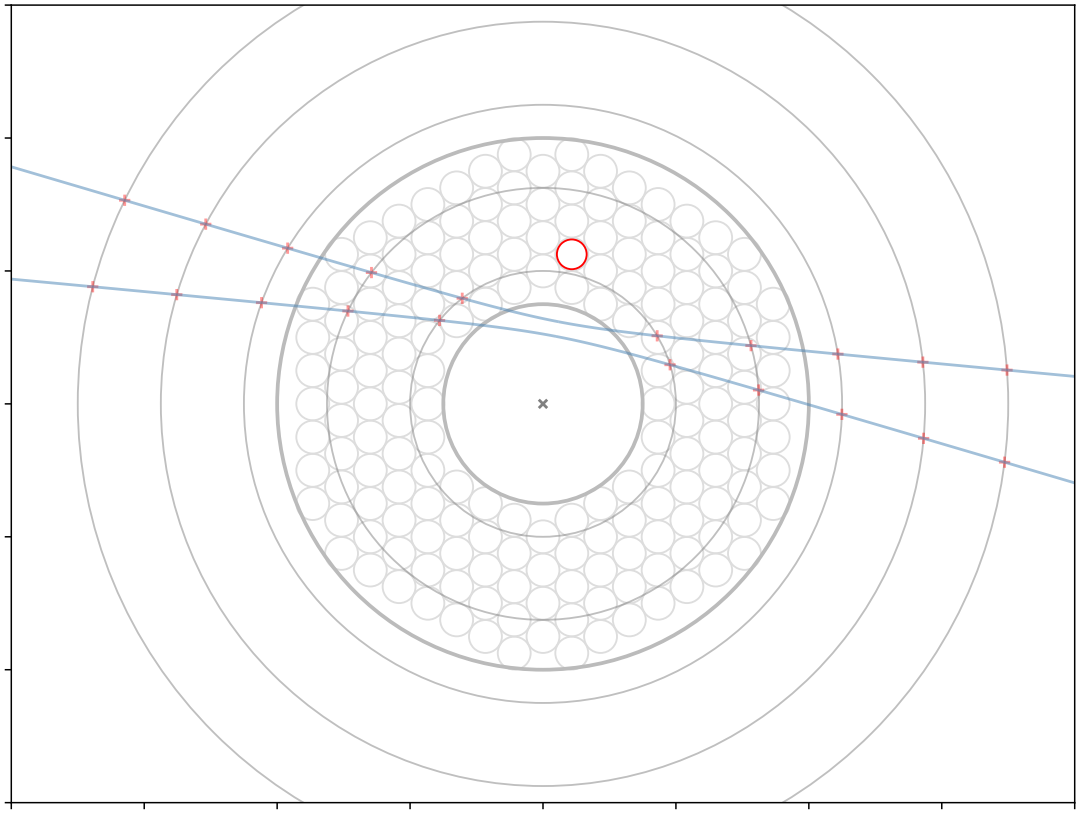
CIRCLE HOUGH



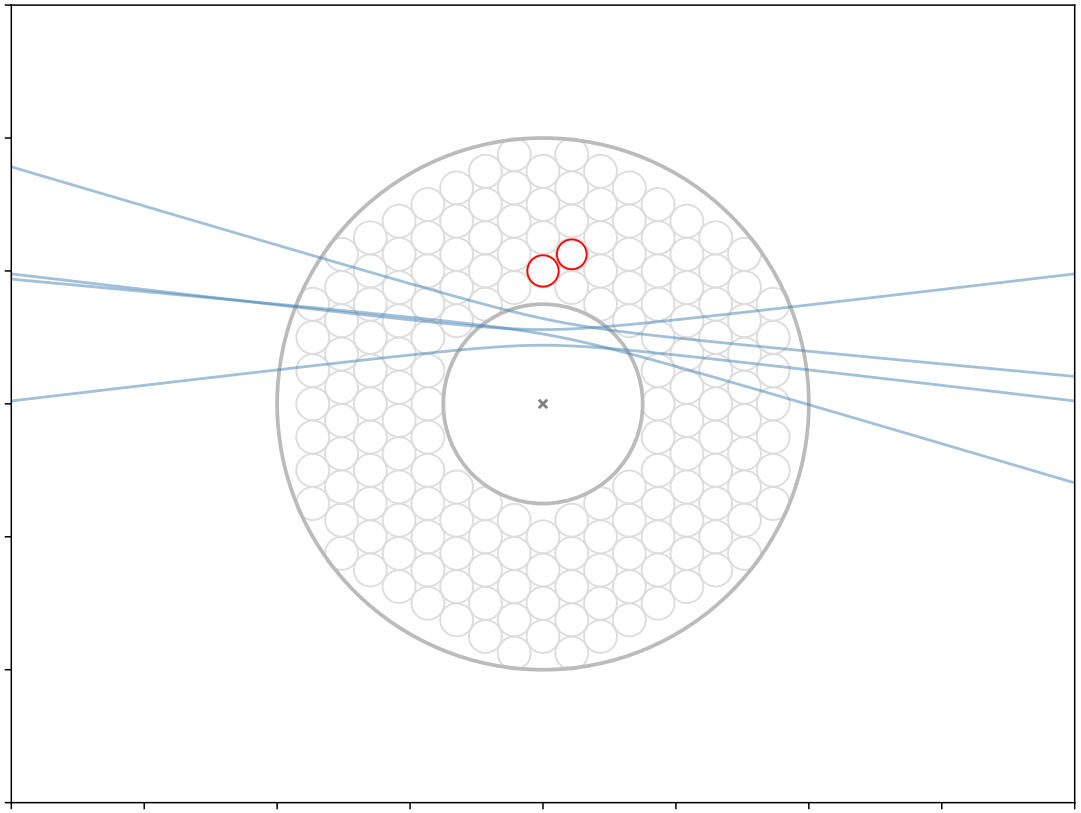
CIRCLE HOUGH



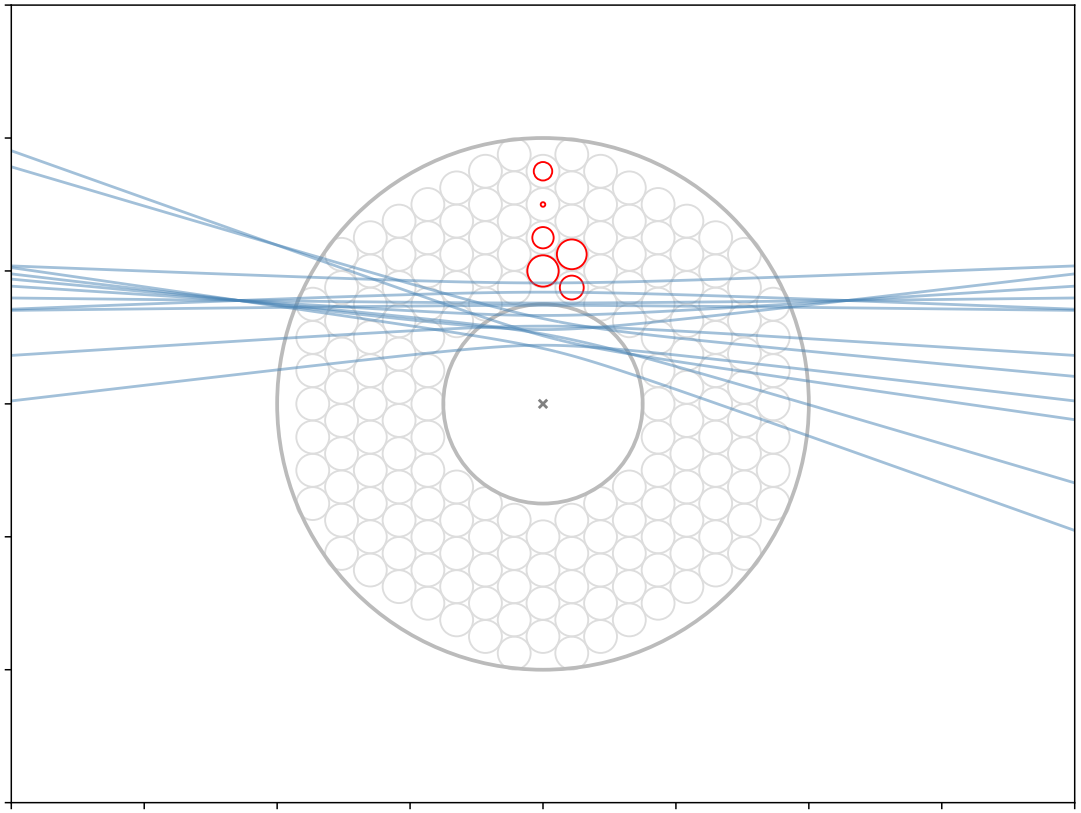
CIRCLE HOUGH



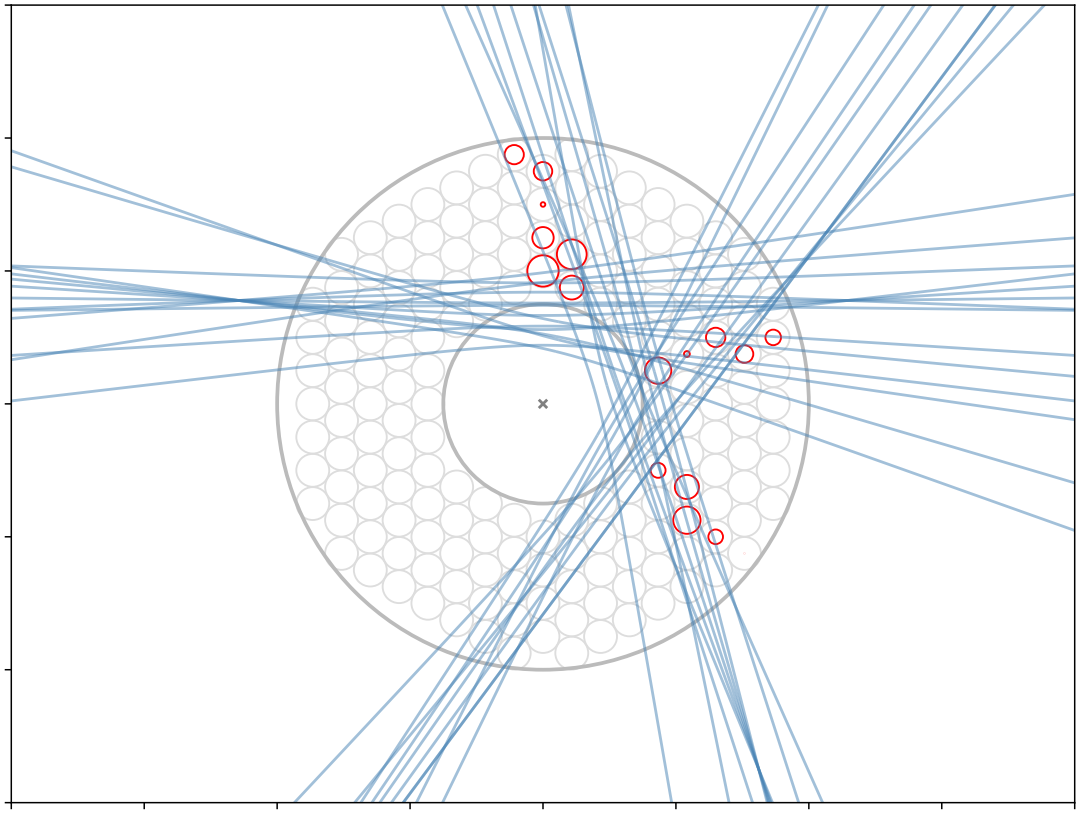
CIRCLE HOUGH



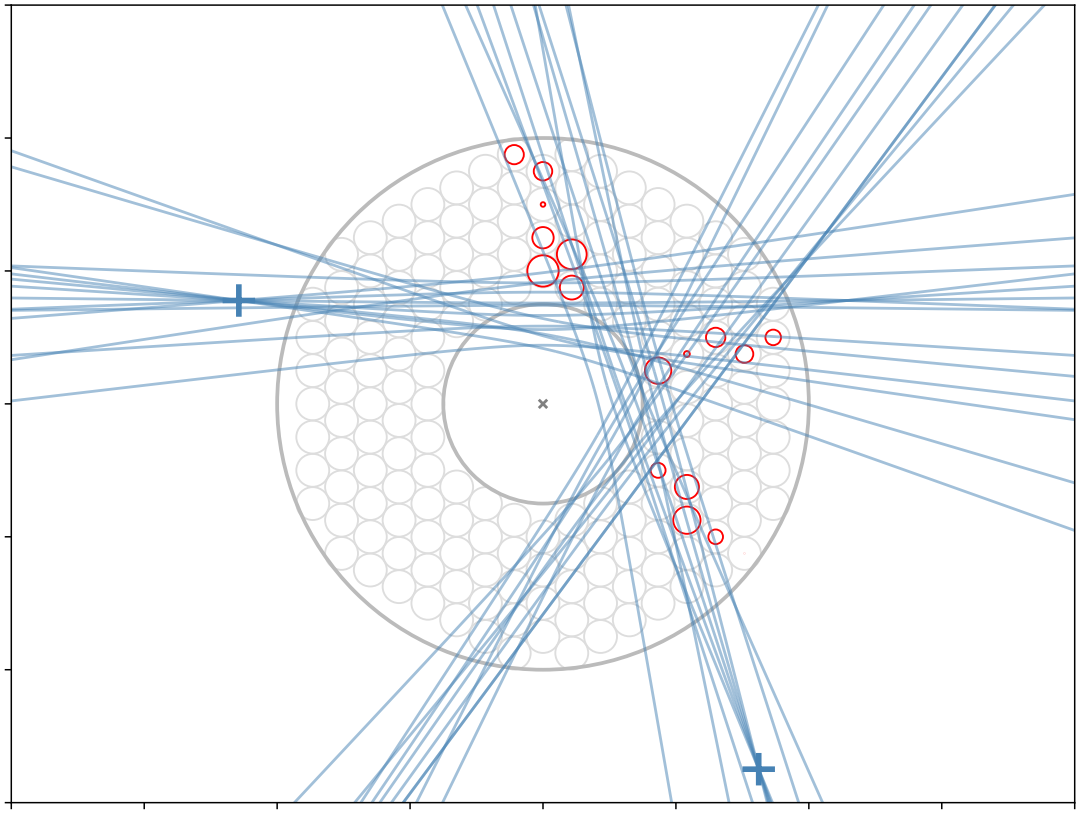
CIRCLE HOUGH



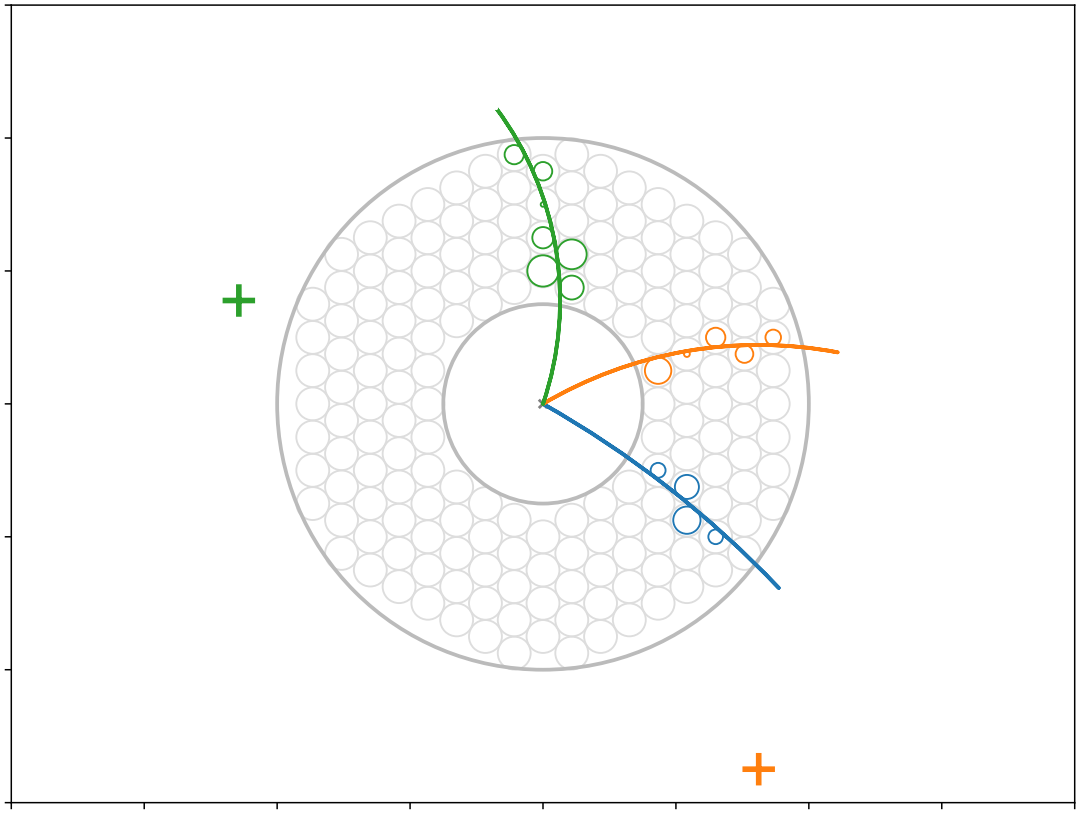
CIRCLE HOUGH



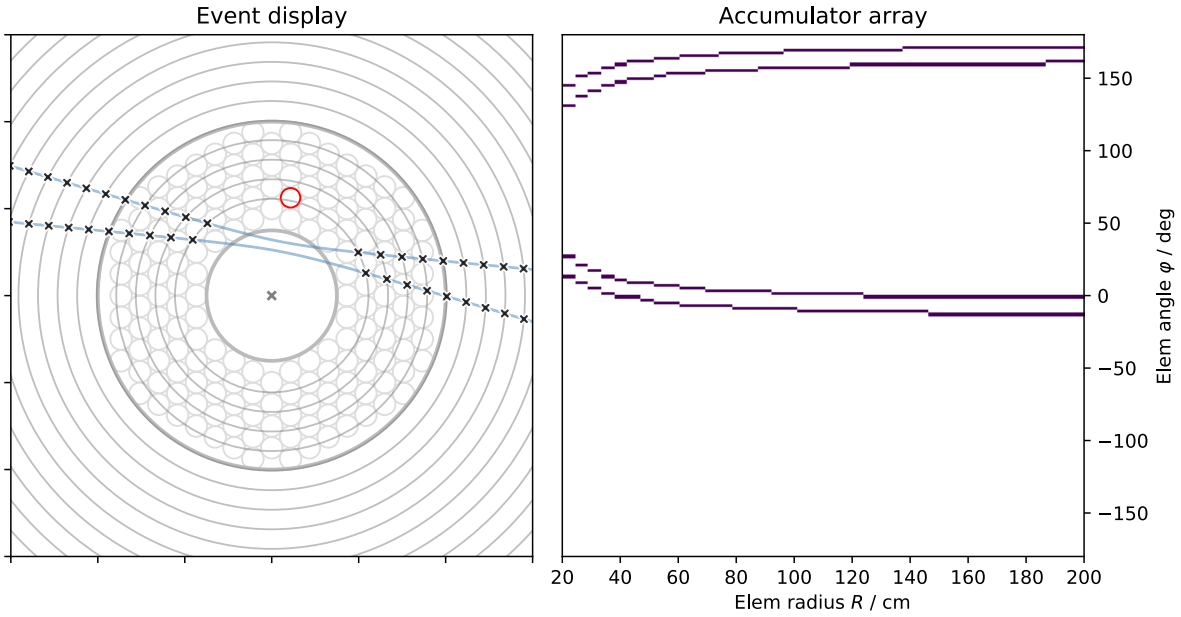
CIRCLE HOUGH



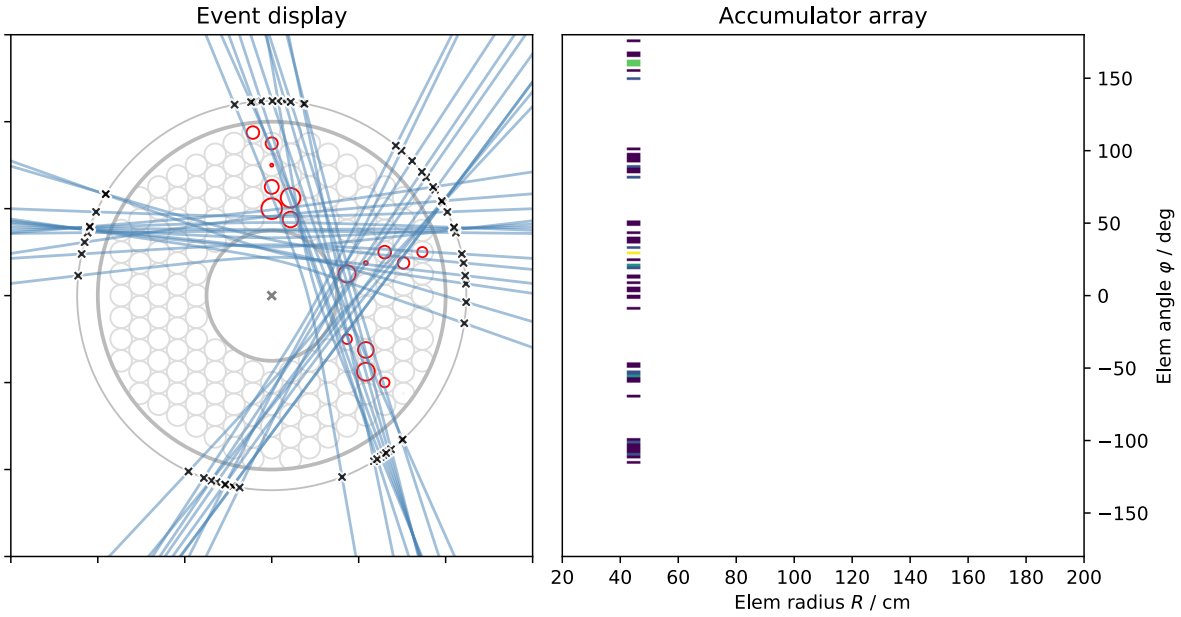
CIRCLE HOUGH



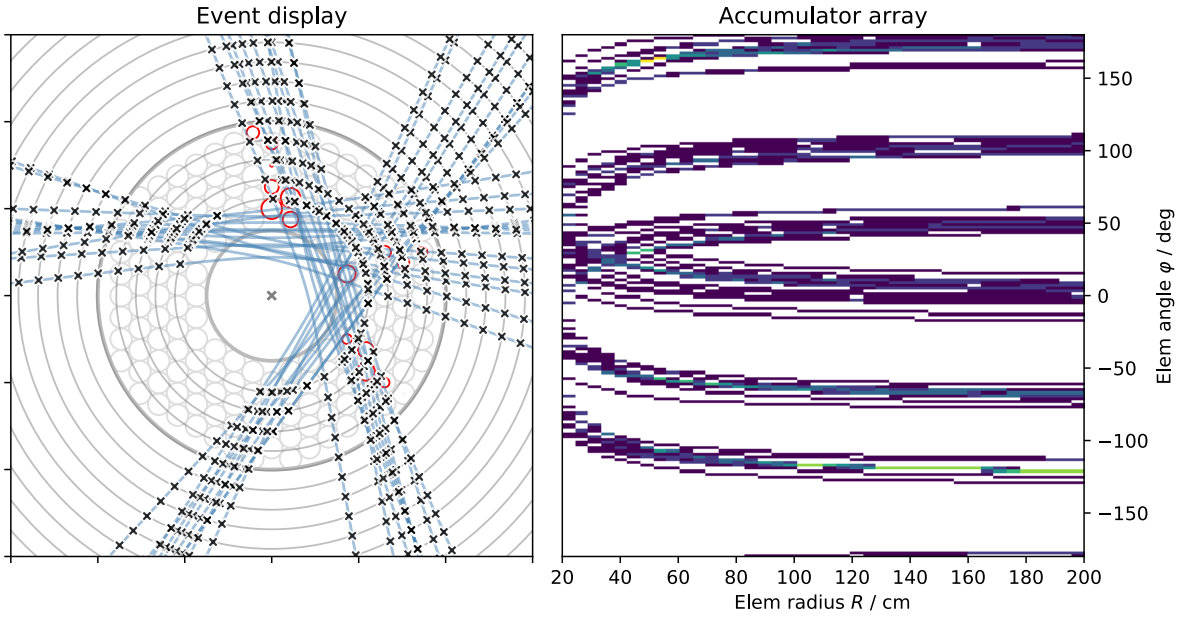
CIRCLE HOUGH



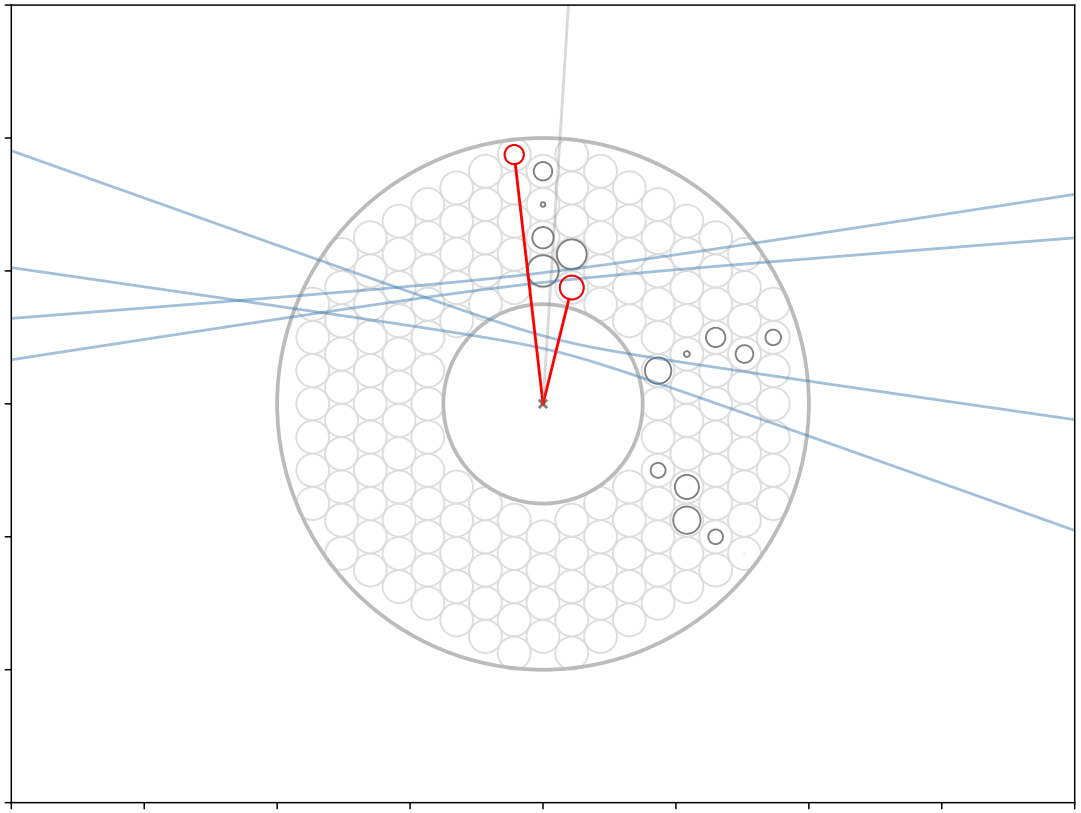
CIRCLE HOUGH



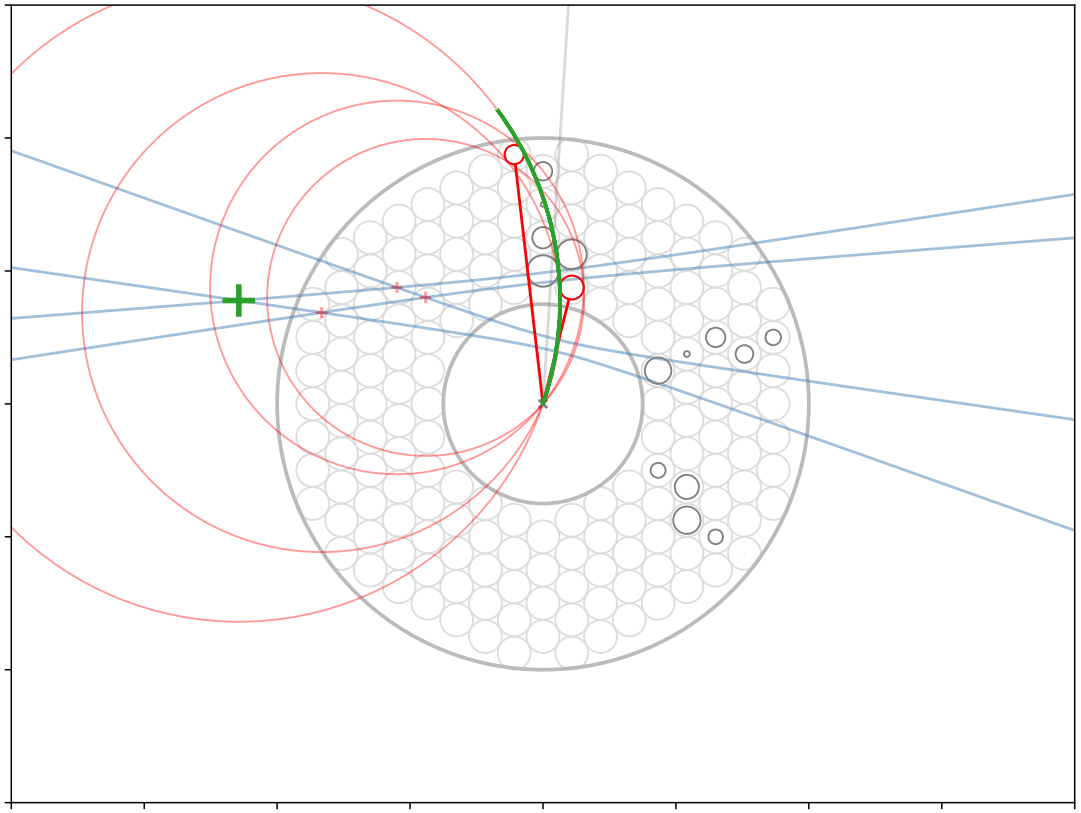
CIRCLE HOUGH



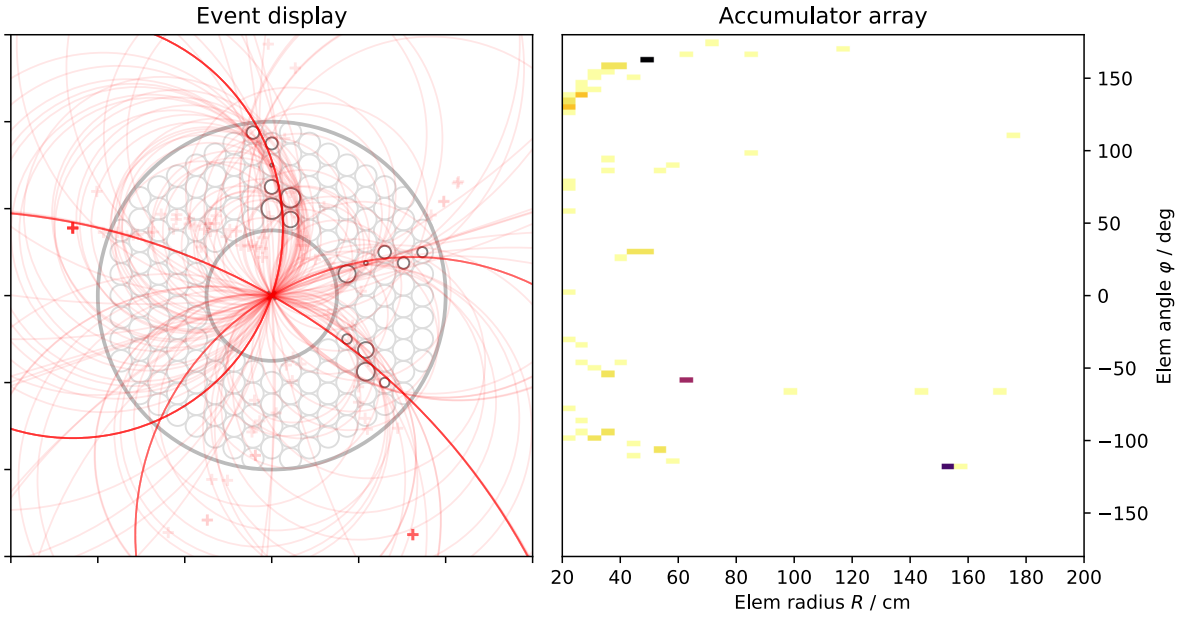
HIT PAIRS HOUGH



HIT PAIRS HOUGH



HIT PAIRS HOUGH



Riemann Track Finder

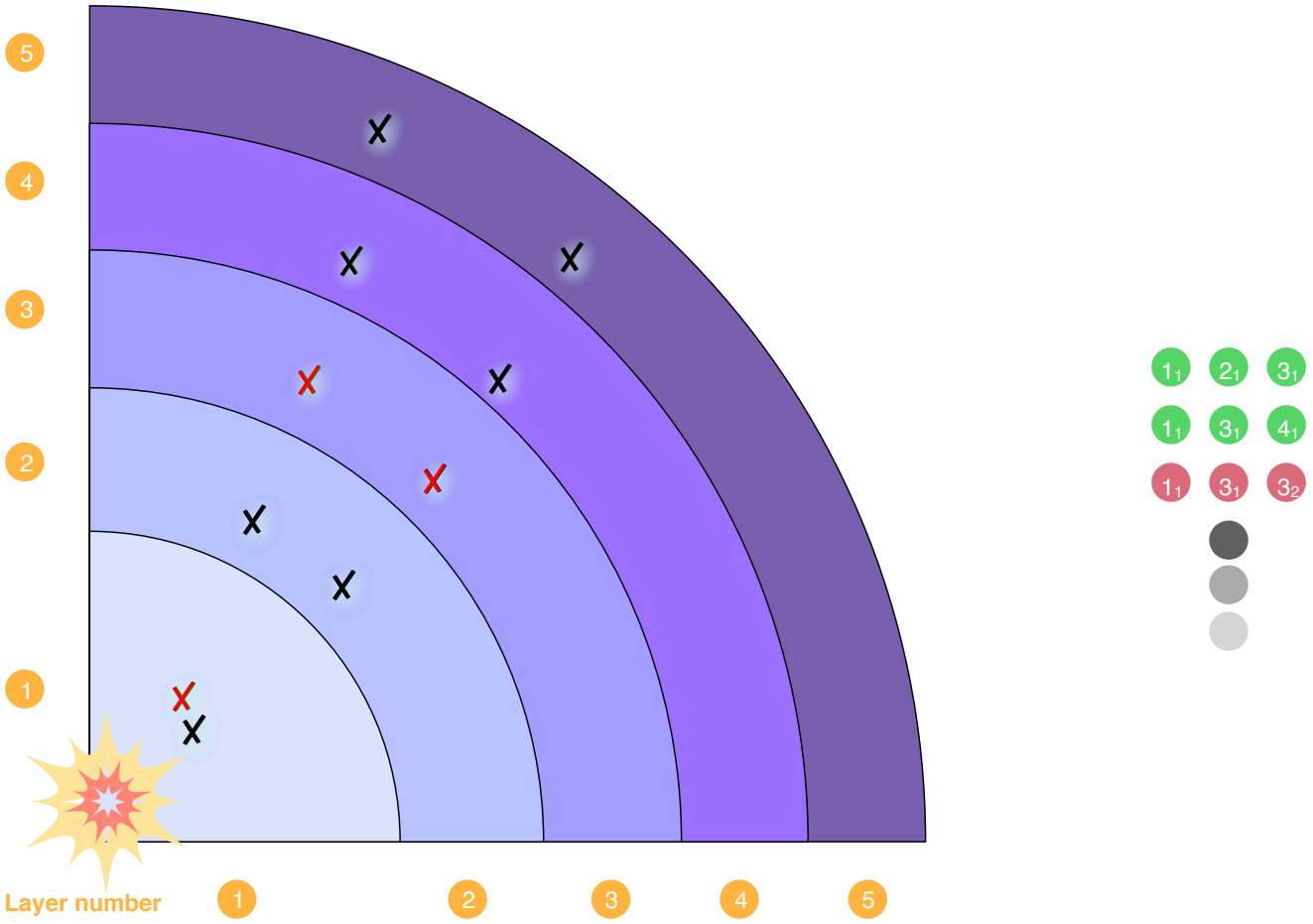
Riemann Algorithm — Procedure

Create triplet of hit points

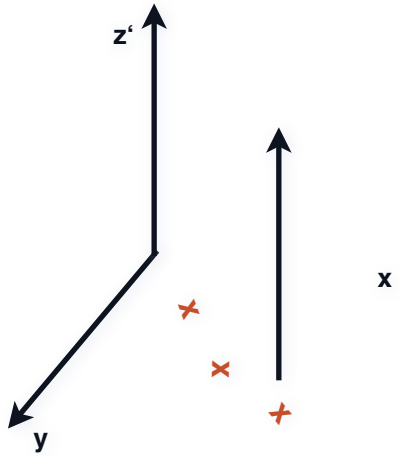
- **All possible three hit combinations need to become triplets**

Grow triplets to tracks:

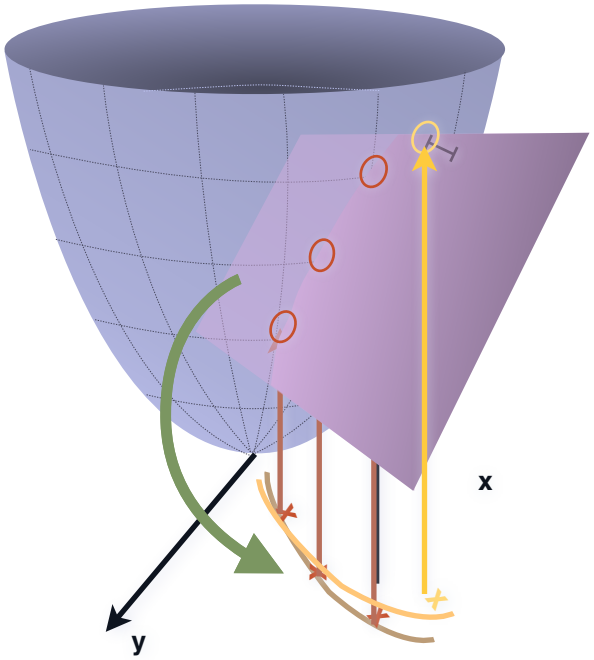
Riemann Track Finder — Seeds



Riemann Algorithm — Expansion

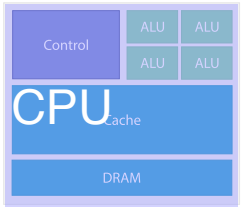


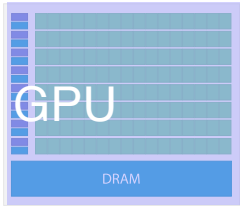
Expand to z‘



Riemann Surface
(paraboloid)

Riemann Track Finder — GPU Adaptations





1 **3 loops to generate seeds serially**

```
for (int i = 0; i < hitsInLayerOne.size(); i++) {
  for (int j = 0; j < hitsInLayerTwo.size(); j++) {
    for (int k = 0; k < hitsInLayerThree.size(); k++) {
      /* Triplet Generation */
    }
  }
}
```

Needed: Mapping of inherent GPU indexing variable to triplet index

```
int ijk = threadIdx.x + blockIdx.x * blockDim.x;
```

$$n_{Layer_x} = \frac{1}{2}\left(\sqrt{8x+1}-1\right)$$

$$pos(n_{Layer_x}) = \frac{\sqrt[3]{\sqrt{3}\sqrt{243x^2-1}+27x}}{3^{2/3}} + \frac{1}{\sqrt[3]{3}\sqrt[3]{\sqrt{3}\sqrt{243x^2-1}+27x}} - 1$$

2 Only *easy* computations; e.g. 3x3 matrices

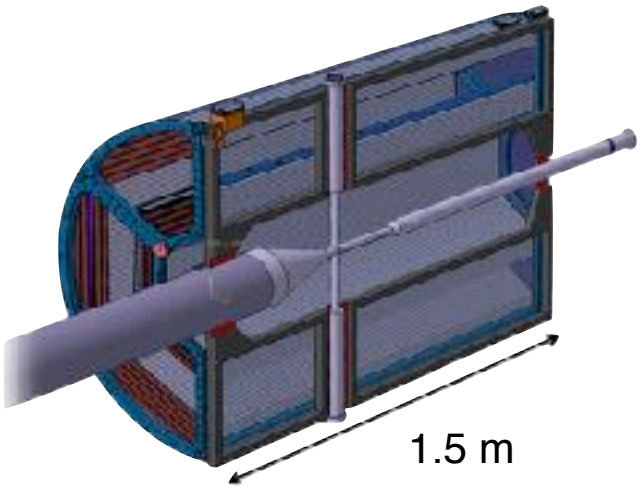
Port of CPU code; parallelism on seed base

→ 100 × faster than CPU version: ~0.6 ms/

Triplet Finder

Triplet Finder

Algorithm specifically designed for the PANDA Straw Tube Tracker (STT)



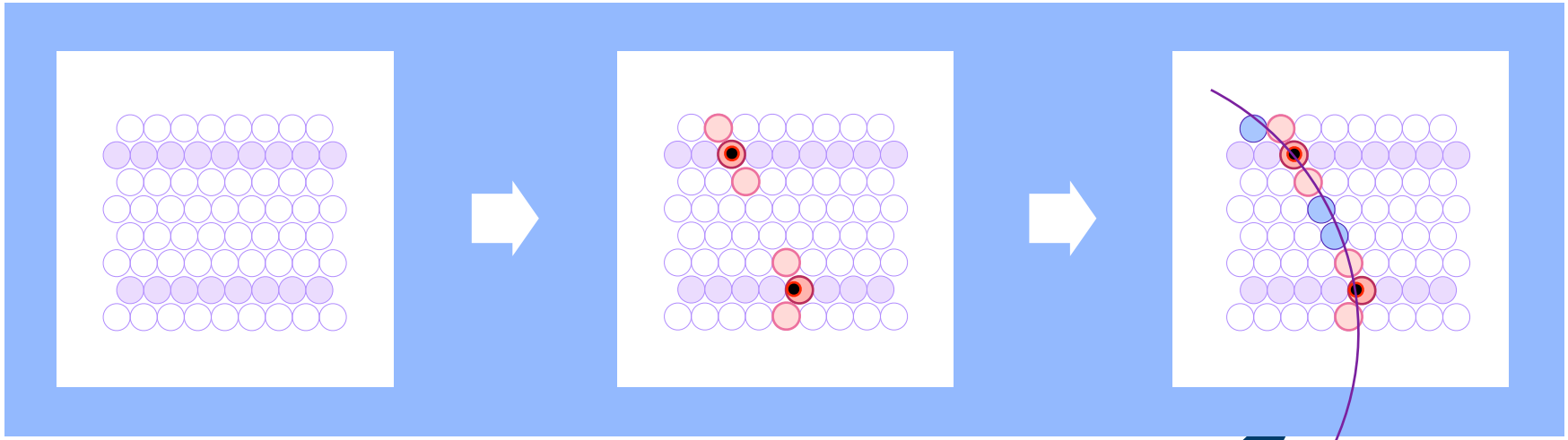
Original algorithm by
Marius Mertens *et al*

JÜLICH
APPLICATION LAB
NVIDIA
http://www.fz-juelich.de/jas/jsc/

Triplet Finder

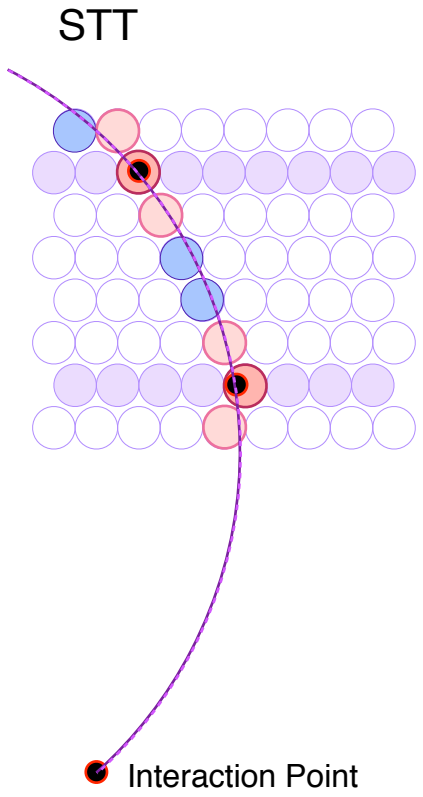
Idea: **Use only subset of detector as seed**

- **Don‘t use STT isochrones** ***(drift times)***
- **Calculate circle from 3 points (no fit)**

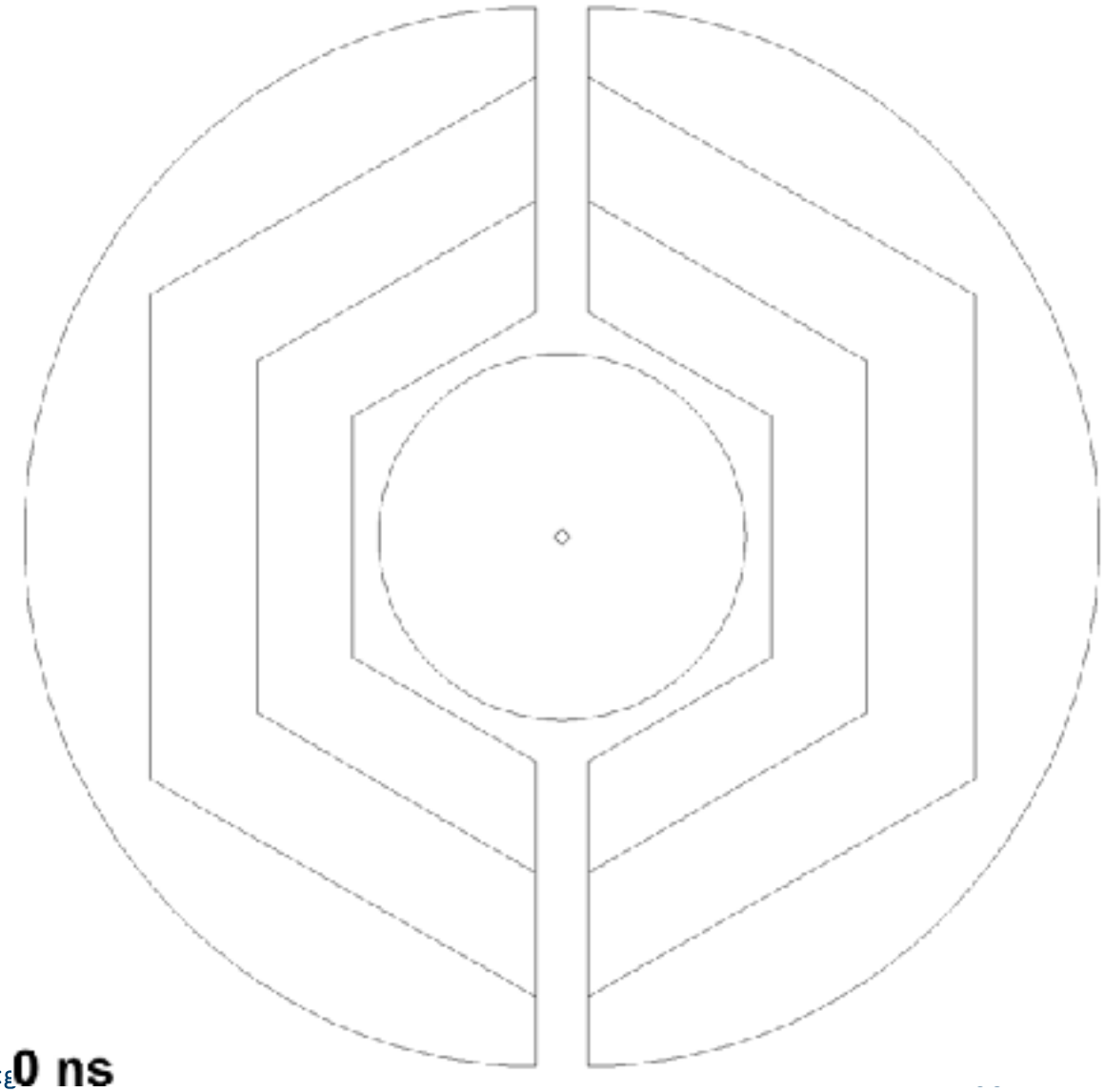


Triplet Finder — Method

STT hit in pivot straw
Find surrounding hits
→ Create virtual hit *(triplet)*



Triplet Finder — Animation



Isochrone *early*

Isochrone *early* & *skewed*

Isochrone *close*

Isochrone *late*

MVD hit

Triplet

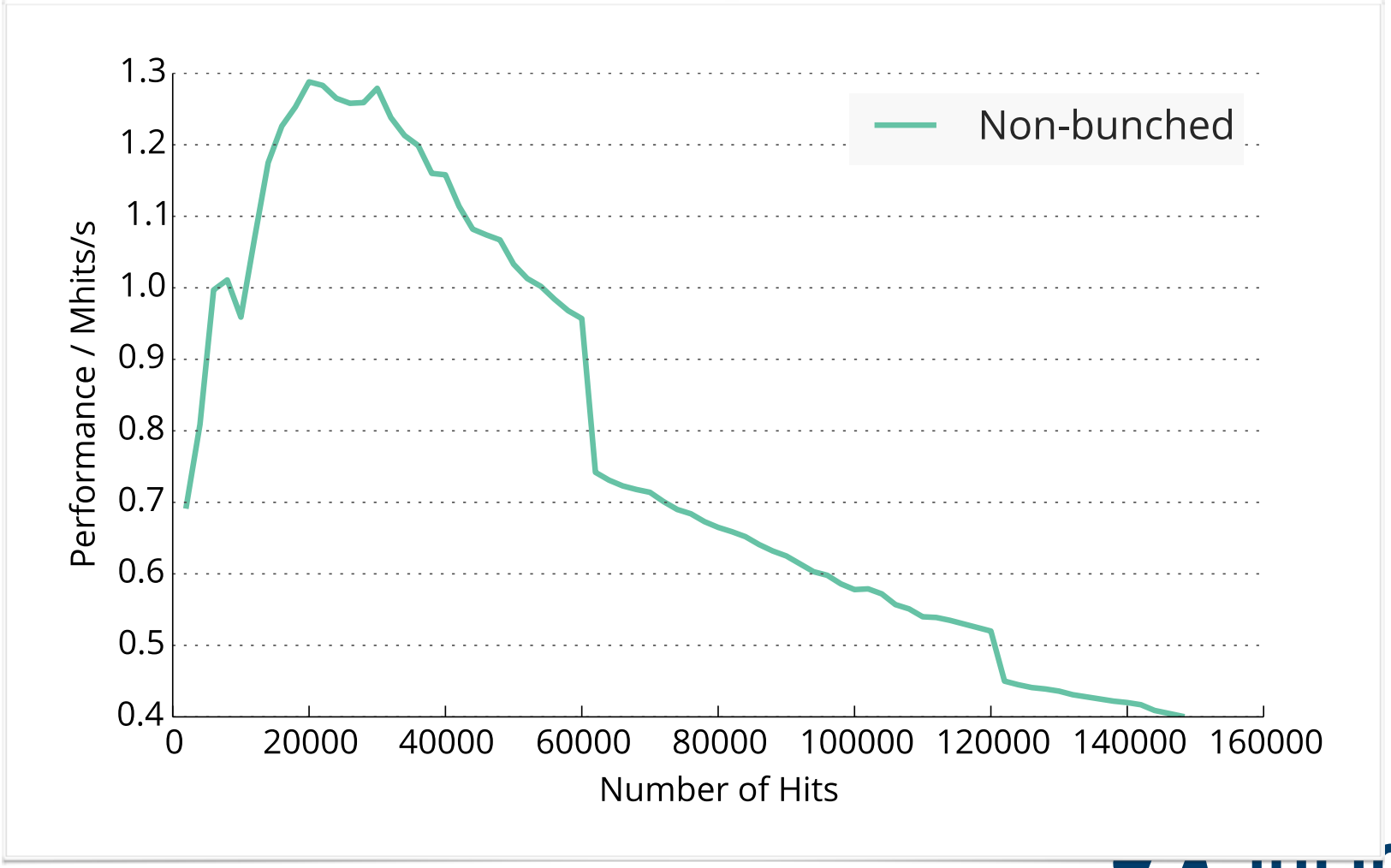
Track *current*

Track *timed out*

Mitg 0 ns

JÜLICH
Forschungszentrum

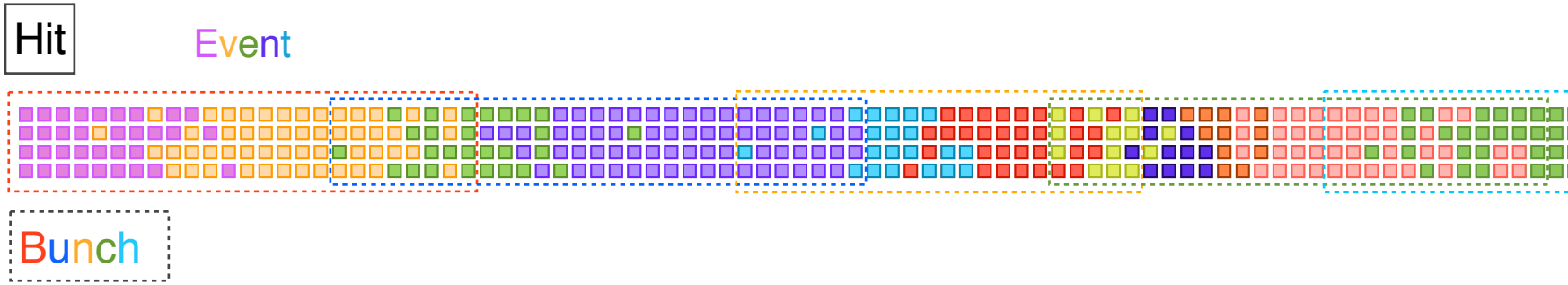
Triplet Finder — Times



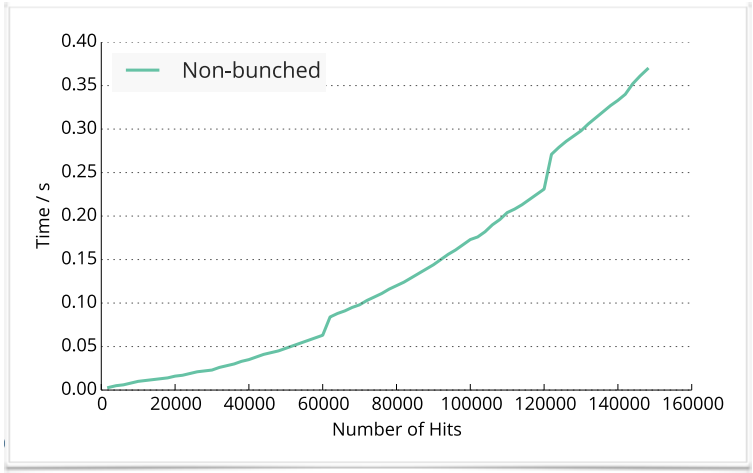
Triplet Finder — Optimizations

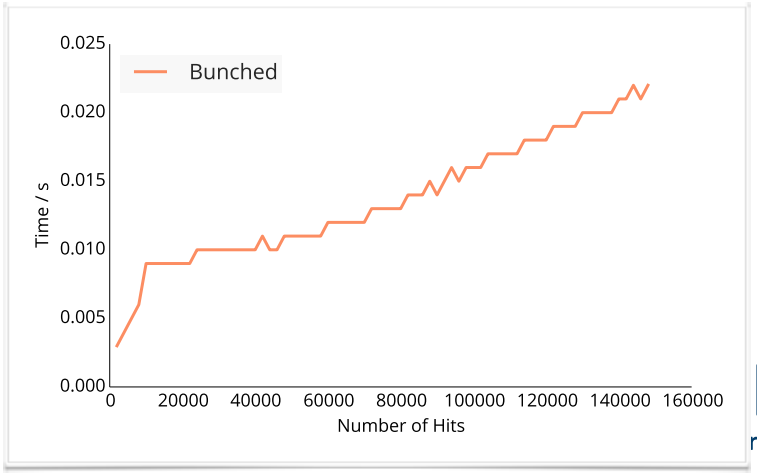
Bunching Wrapper

- **Hits from one event have similar timestamp**
- **Combine hits to sets (bunches) which occupy GPU best**



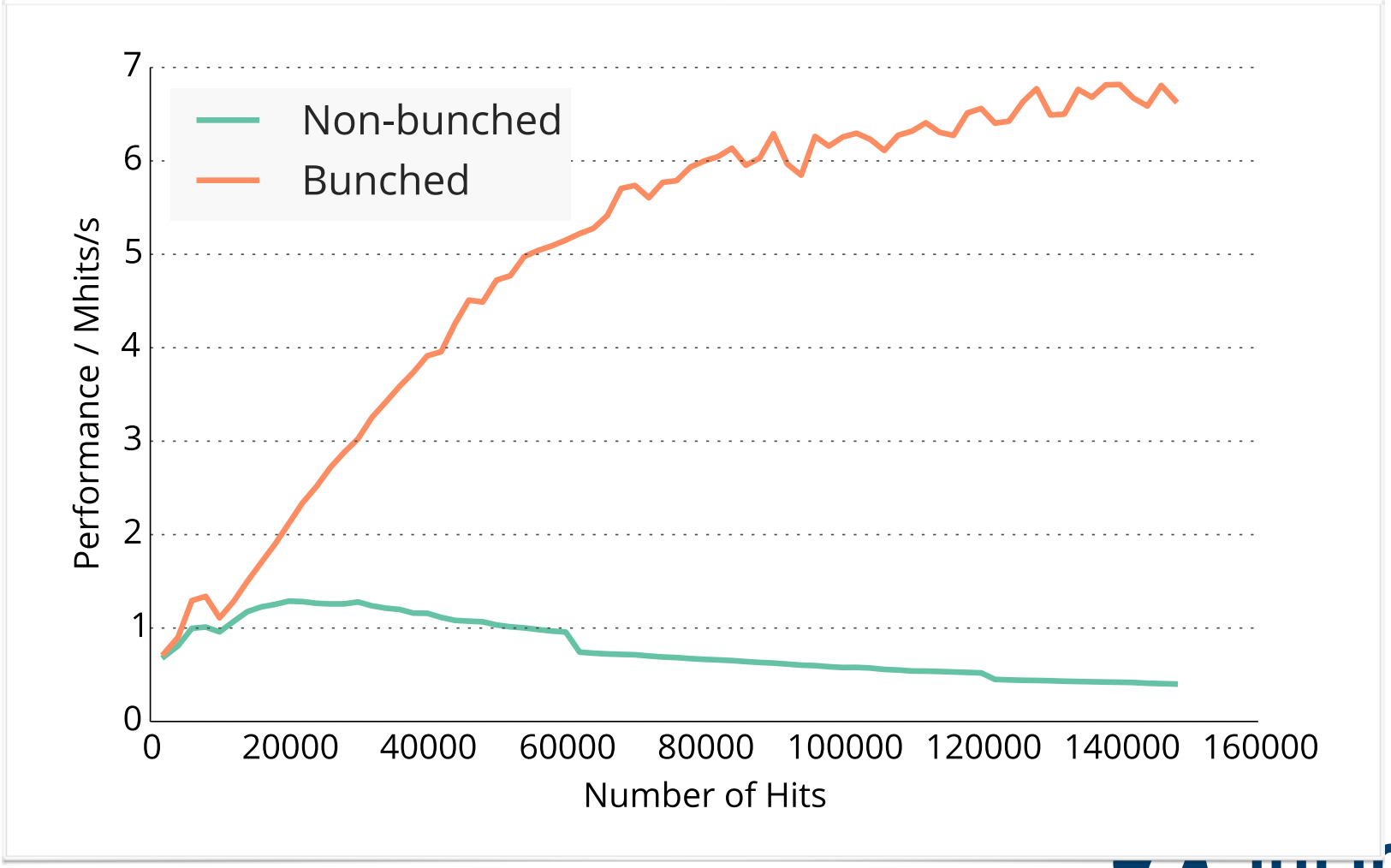
$$\mathcal{O}(N^2) \rightarrow \mathcal{O}(N)$$





Triplet Finder — Bunching

Performance

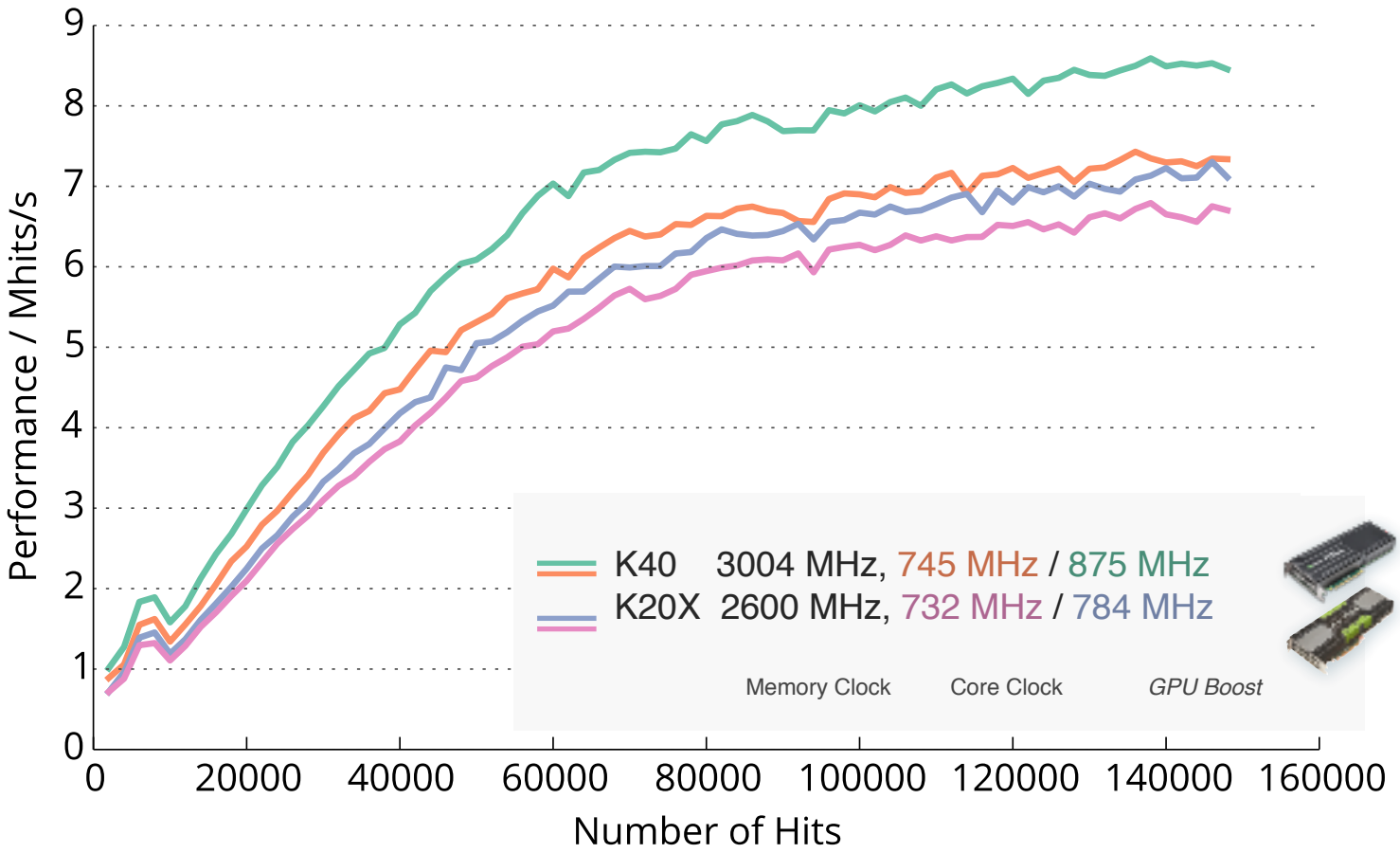


Triplet Finder — Clock Speed / GPU

Original algorithm by Marius C. Mertens *et al.*

Ported to GPU by Andrew A. Adinetz (NVIDIA Application Lab)

- **Thrust, CUDA, some dynamic parallelism**



Triplet Finder — Summary

Best performance: 20 *µs*/event

- **$20 \cdot 10^{-6}$ s/event * $2 \cdot 10^{7}$ event/s ⇒ 400 GPUs[2014]**
- **PANDA[2019]: Multi GPU system – $\mathcal{O}$(100) GPUs**

Optimizations possible & needed

- **ε needs to be improved**
- **Speed, €:**
  - ***More float less double*-cards a la K10**
  - **Consumer-grade cards a la GTX**

Summary

Many ideas how to do fast online tracking

But:

- **not a single version fully running in PandaRoot**
- **all have their strengths and weaknesses**

Still a lot of works!

List of Resources Used

#4: Earth icon by Francesco Paleari from The Noun Project

#4: Einstein icon by Roman Rusinov from The Noun Project

#6: FAIR vector logo from official FAIR website

#6: FAIR rendering from official website

#11: Flare Gun icon by Jop van der Kroef from The Noun Project

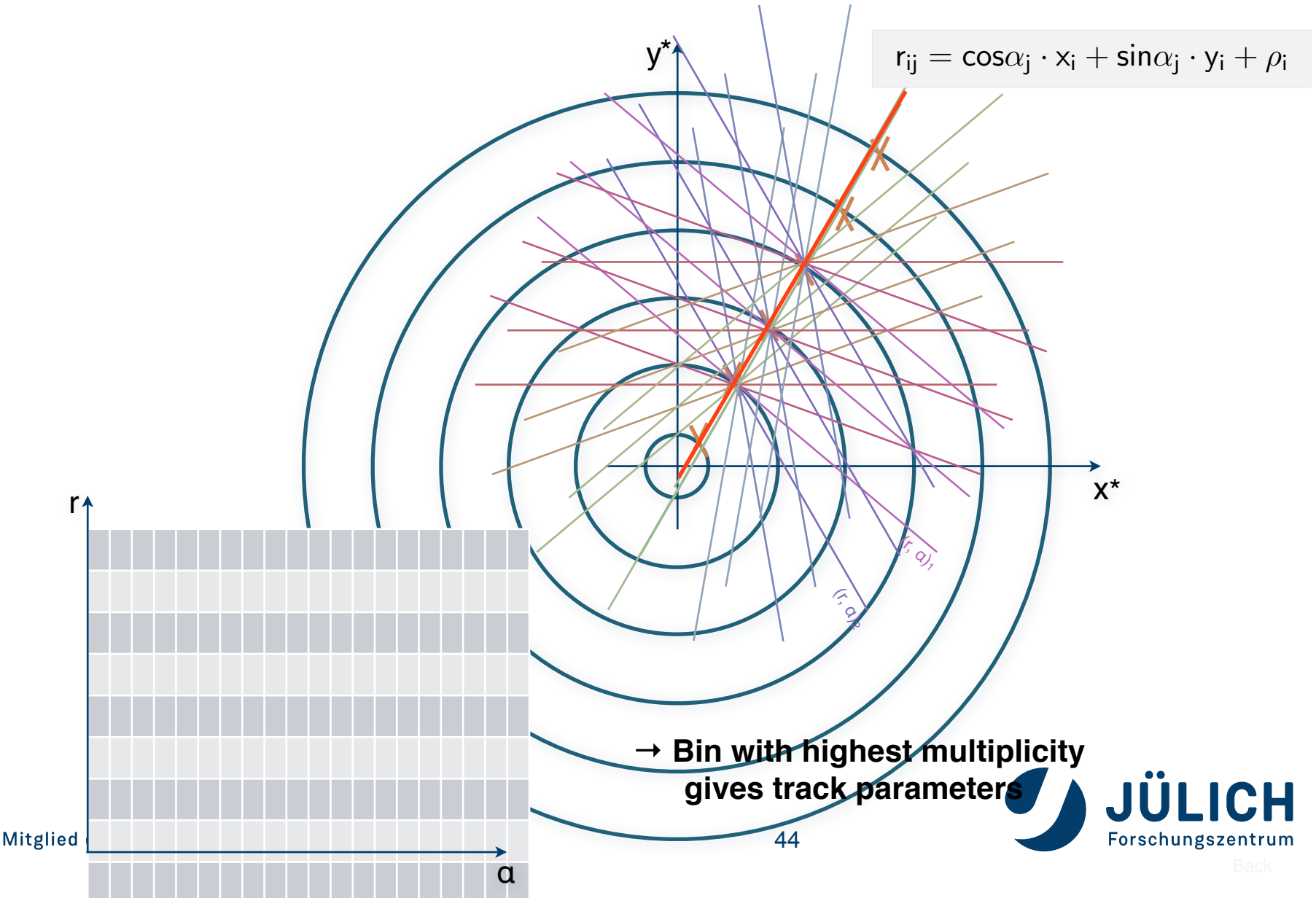
#27: STT event animation by Marius C. Mertens

#35: Graphics cards images by NVIDIA promotion

#35: GPU Specifications

- **Tesla K20X Specifications: http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf**
- **Tesla K40 Specifications: http://www.nvidia.com/content/PDF/kepler/Tesla-K40-Active-Board-Spec-BD-06949-001_v03.pdf**
- **Tesla Familiy Overview: http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf**

Hough Transform — Principle



JÜLICH Forschungszentrum

Back

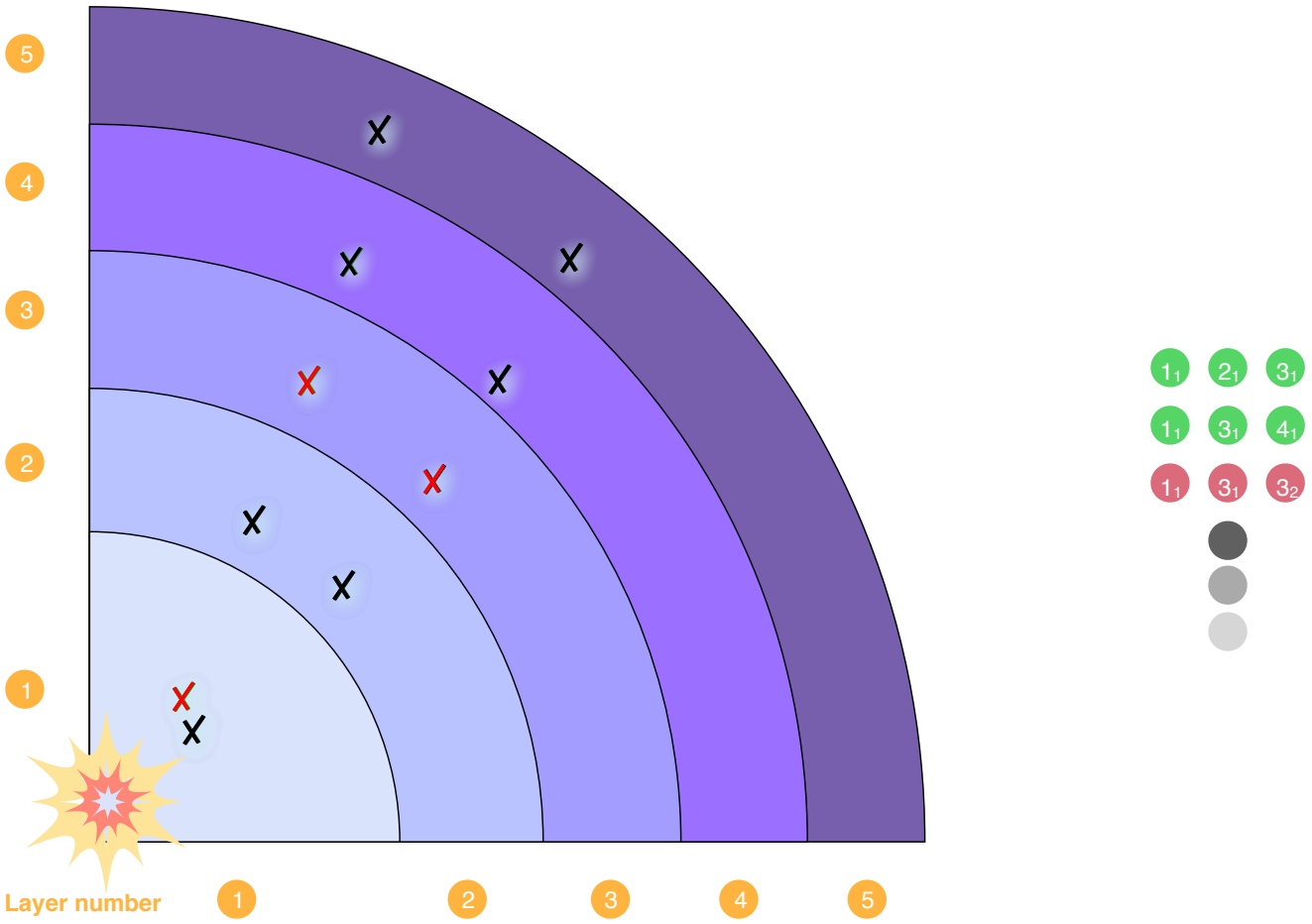
Riemann Algorithm — Procedure

1

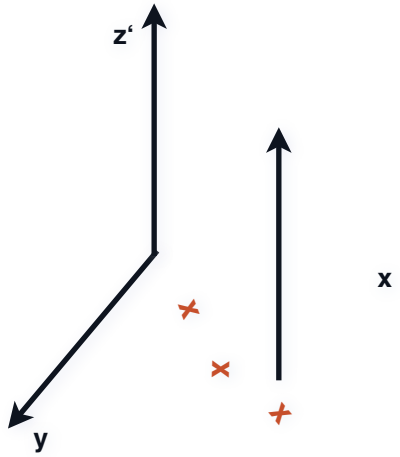
C triplet of hit points
l possible three hit combinations need to become triplets
G triplets to tracks:

2

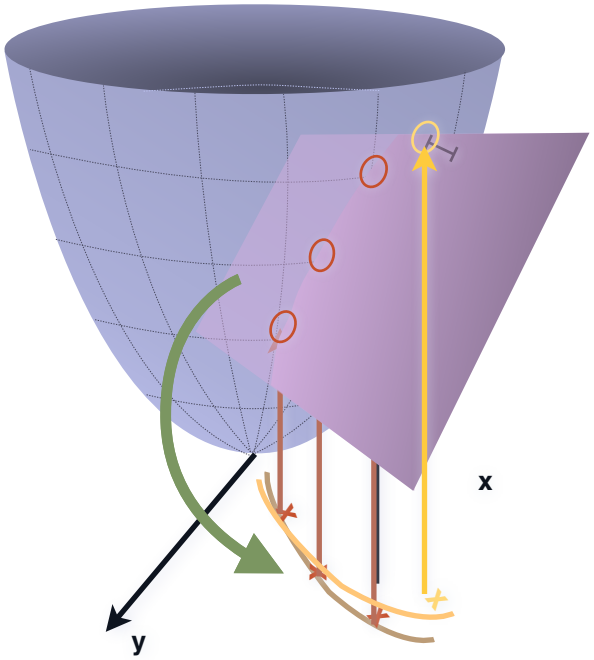
Riemann Track Finder — 1 Seeds



Riemann Algorithm — 2 Expansion



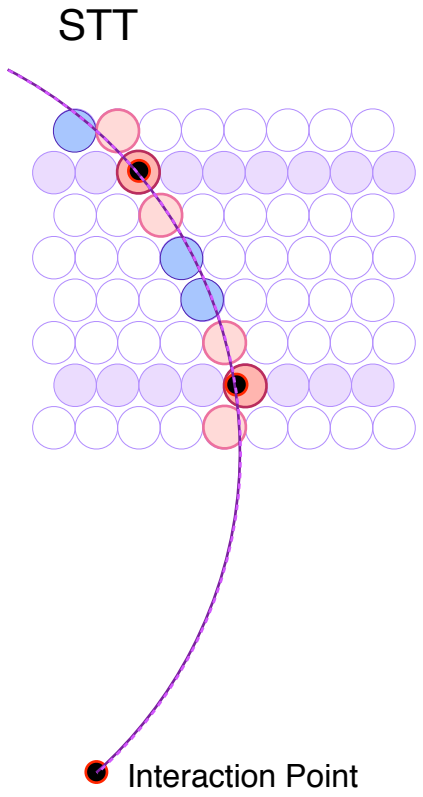
Expand to z‘



Riemann Surface
(paraboloid)

Triplet Finder — Method

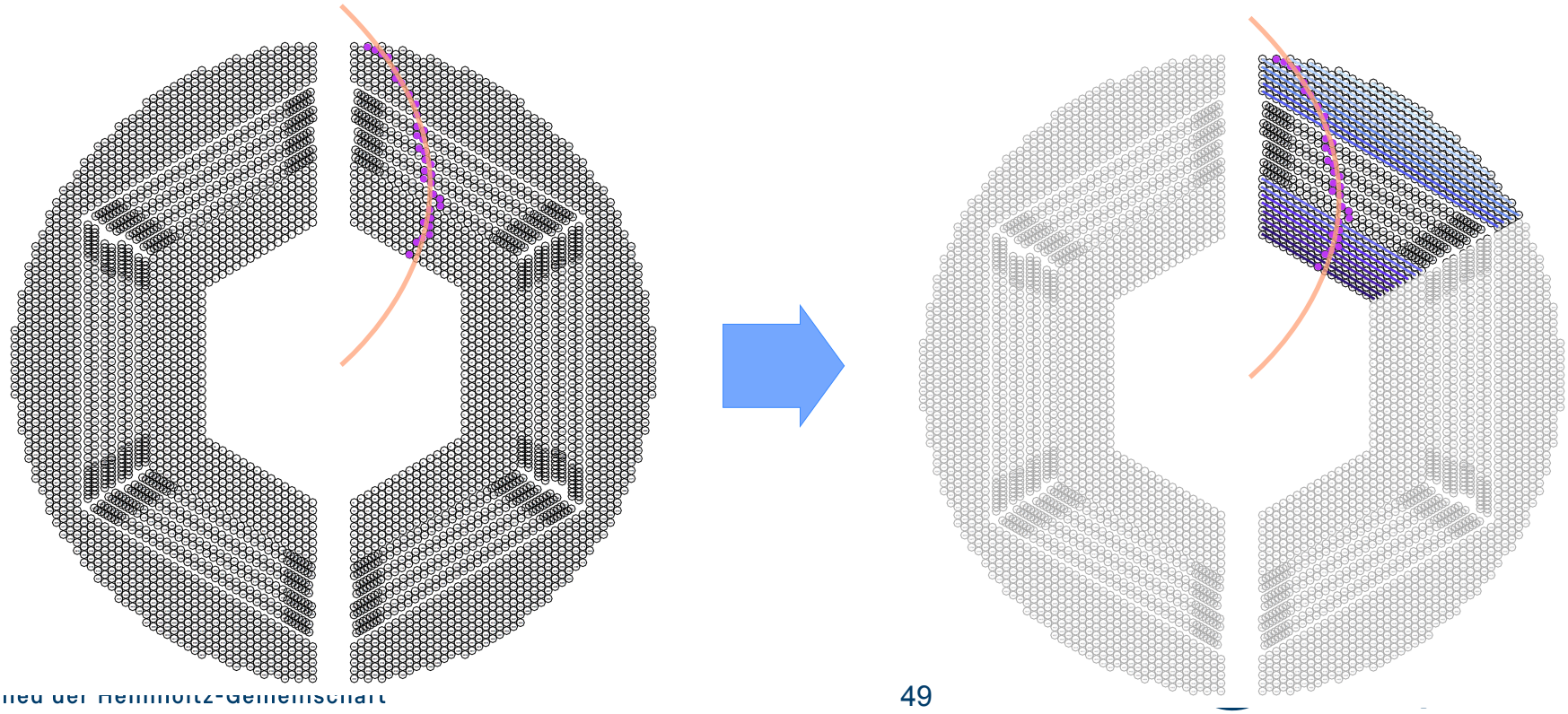
STT hit in pivot straw
Find surrounding hits
→ Create virtual hit *(triplet)*



Triplet Finder — Optimizations

Sector Row testing

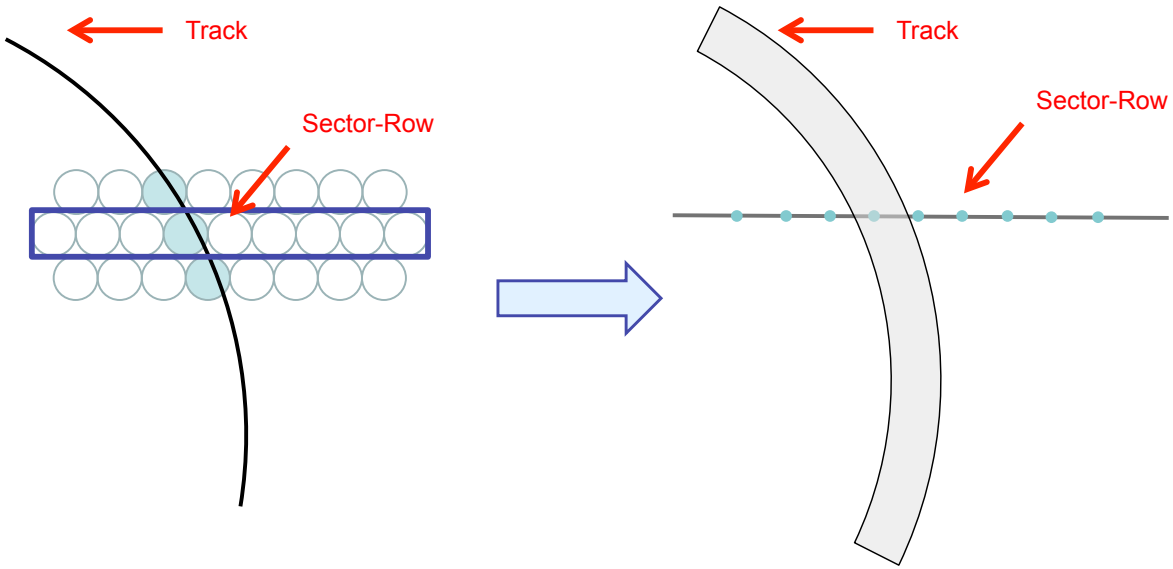
- **After found track:**
  Hit association not with *all* hits of current window,



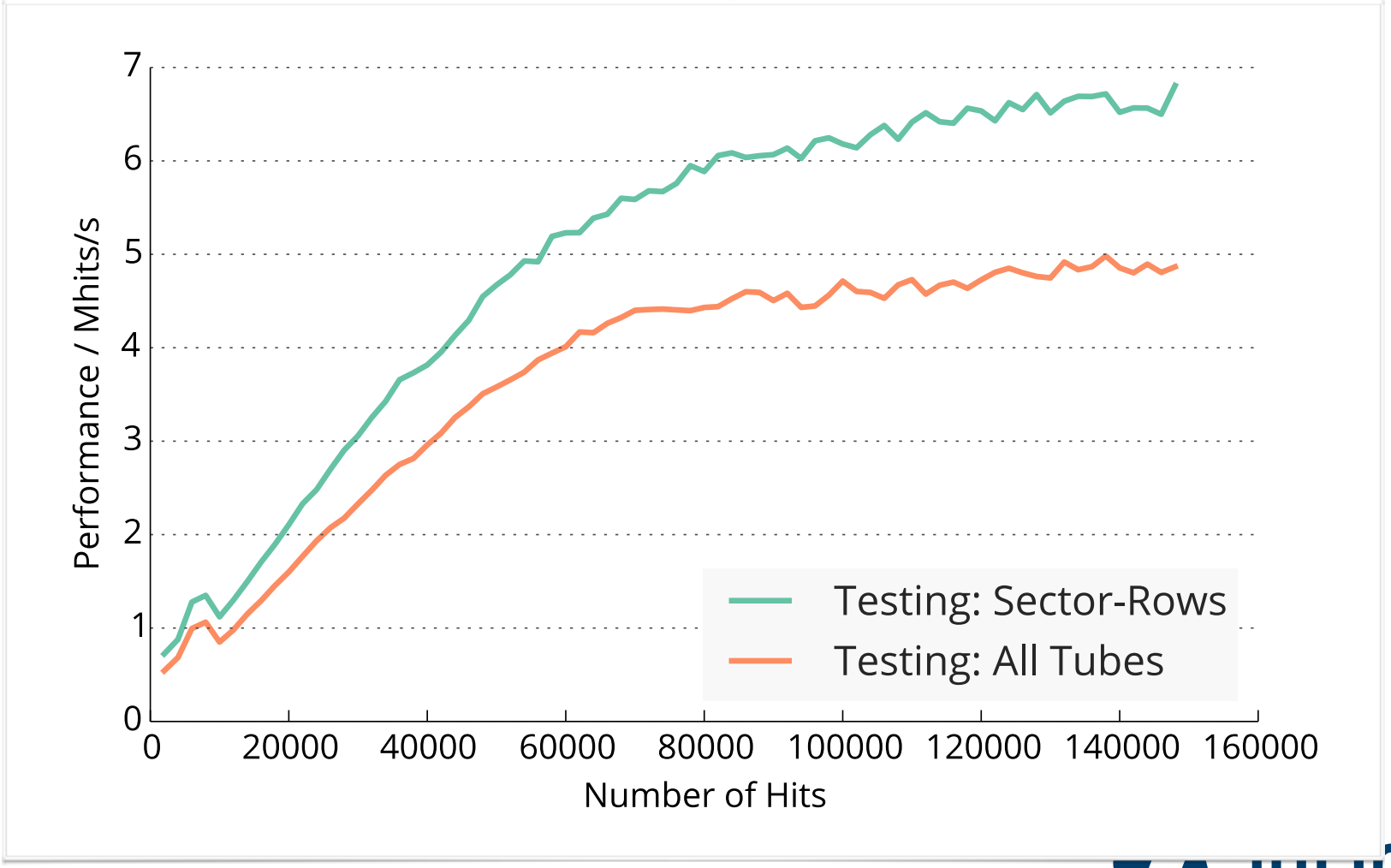
Triplet Finder — Optimizations

Sector Row testing

- **Thicken track; shrink sector row layer to line**
- **Find intersection**



Triplet Finder — Sector Rows



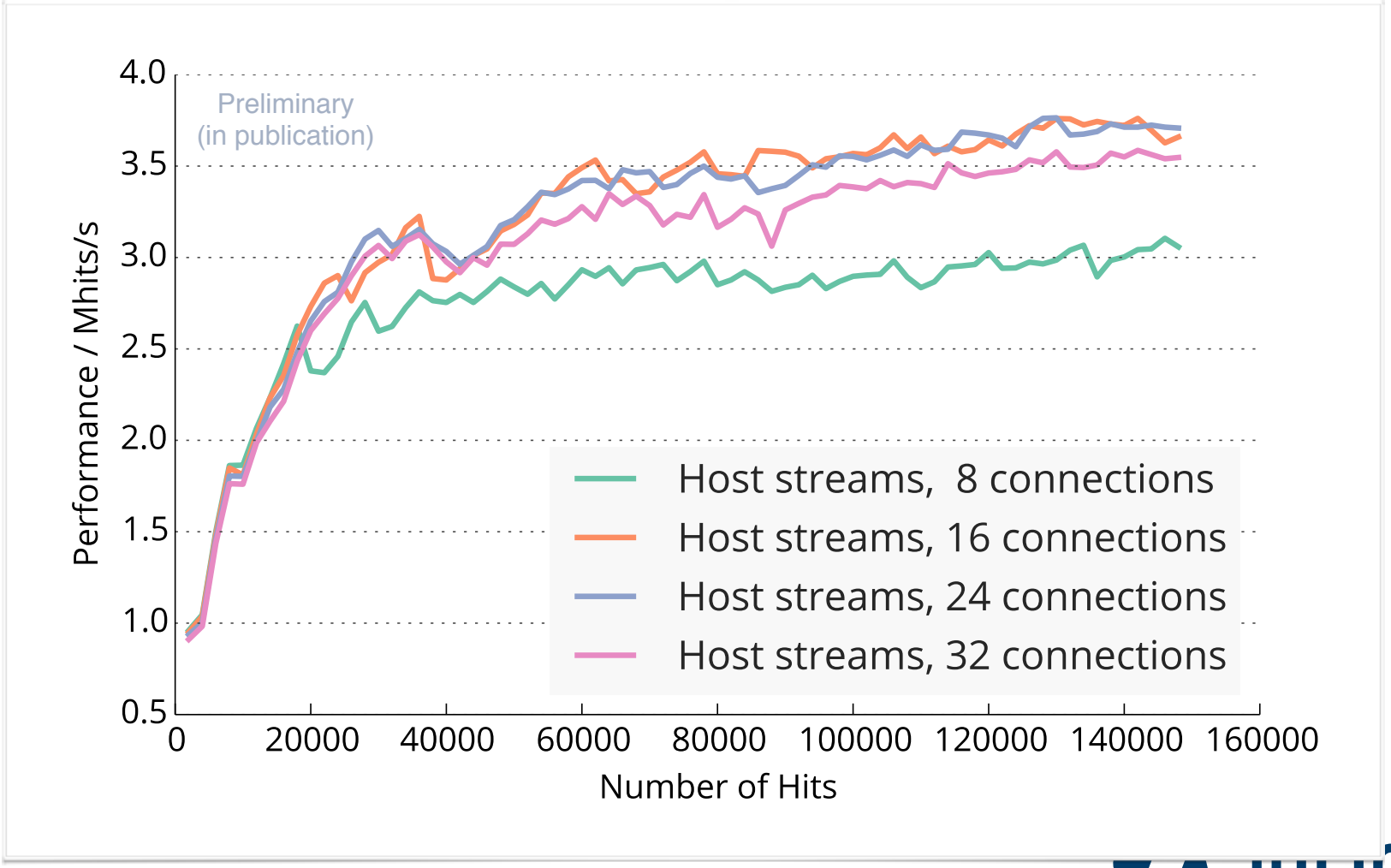
Triplet Finder — Kernel Launch Strategies

Joined Kernel *(JK)*: slowest

- **High # registers → low occupancy**

Dynamic Parallelism *(DP)* / Host Streams *(HS)*: comparable performance

Triplet Finder — Host Stream Connections



Triplet Finder — Bunch Sizes