GPU-based Online Tracking for the PANDA Experiment

GPU in High Energy Physics Conference, Pisa

11 September 2014, Andreas Herten

Outline

- PANDA
  - Experiment
  - Online Event Filter
- Algorithms
  - Hough Transform
  - Riemann Track Finder
  - Triplet Finder

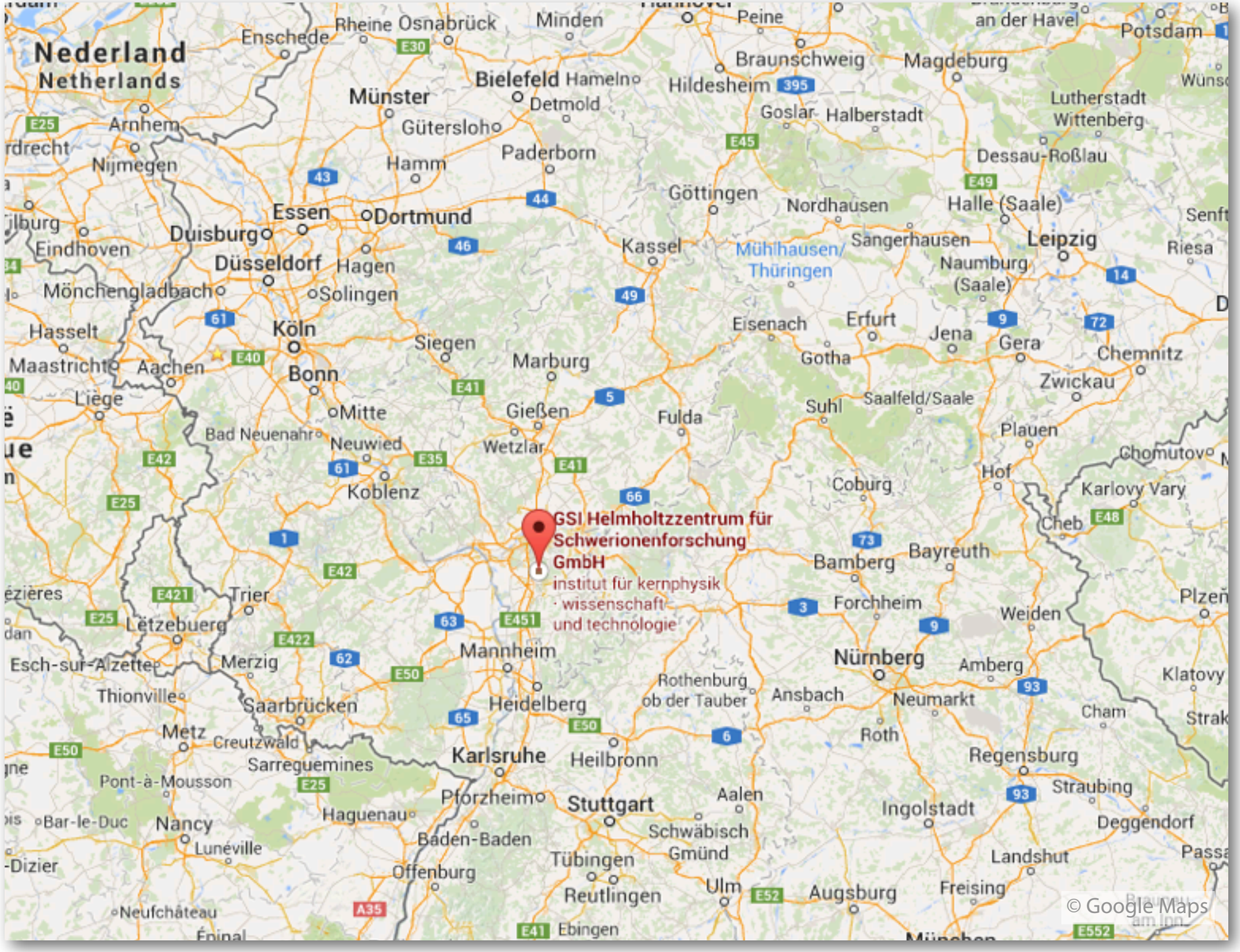
FAIR

- **F**acility for **A**ntiproton and **I**on **R**esearch
  - New accelerator complex (Darmstadt, Germany)
  - Next to GSI laboratory
  - Construction in progress, ending 2018
  - Four pillars of research:

APPA	NUSTAR	CBM	PANDA
Atom & plasma physics	Nuclear structure, astro physics	Hadron physics	Hadron physics

FAIR

- **F**acility for **A**ntiproton and **I**on **R**esearch



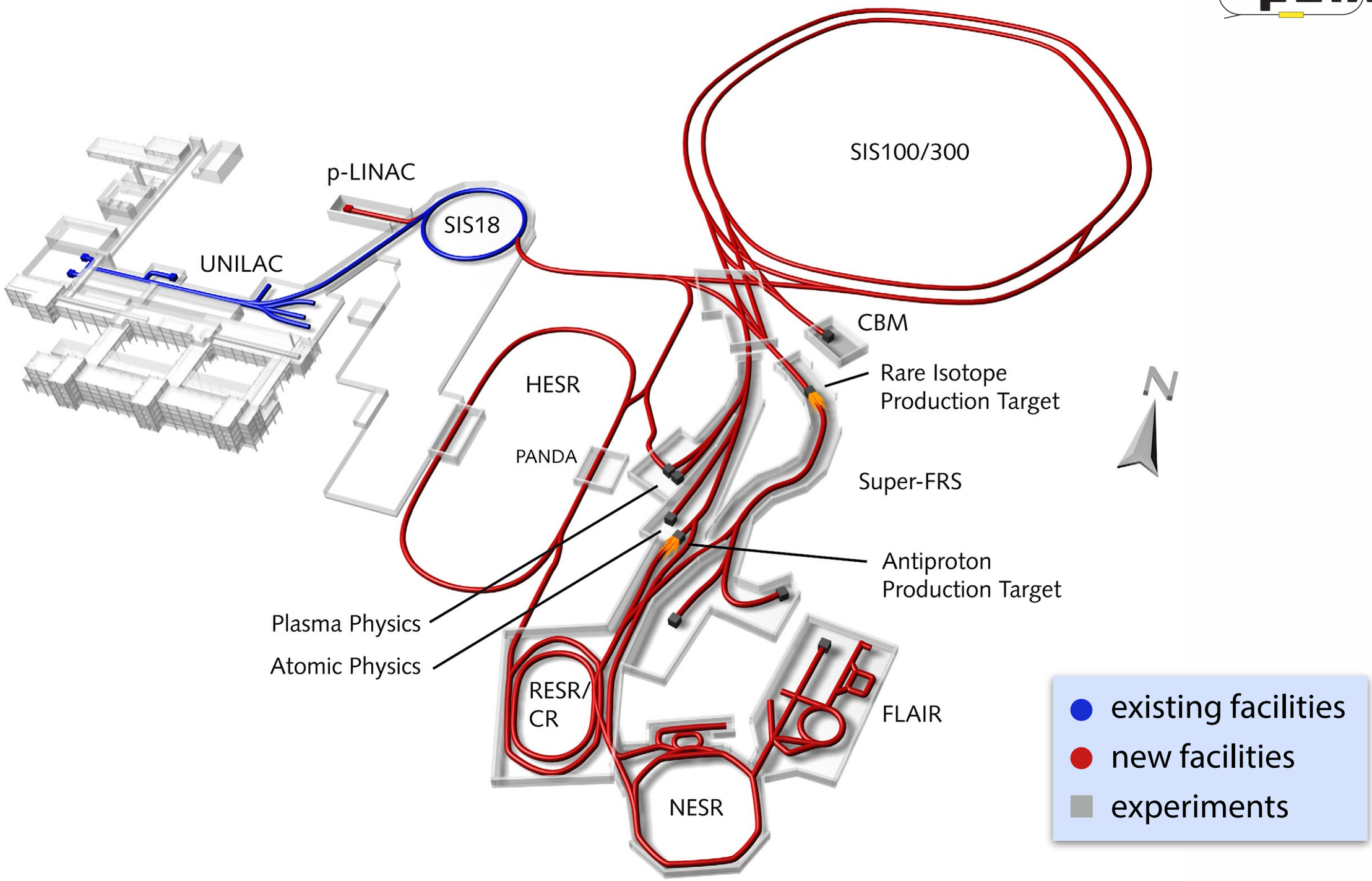
FAIR

- **F**acility for **A**ntiproton and **I**on **R**esearch

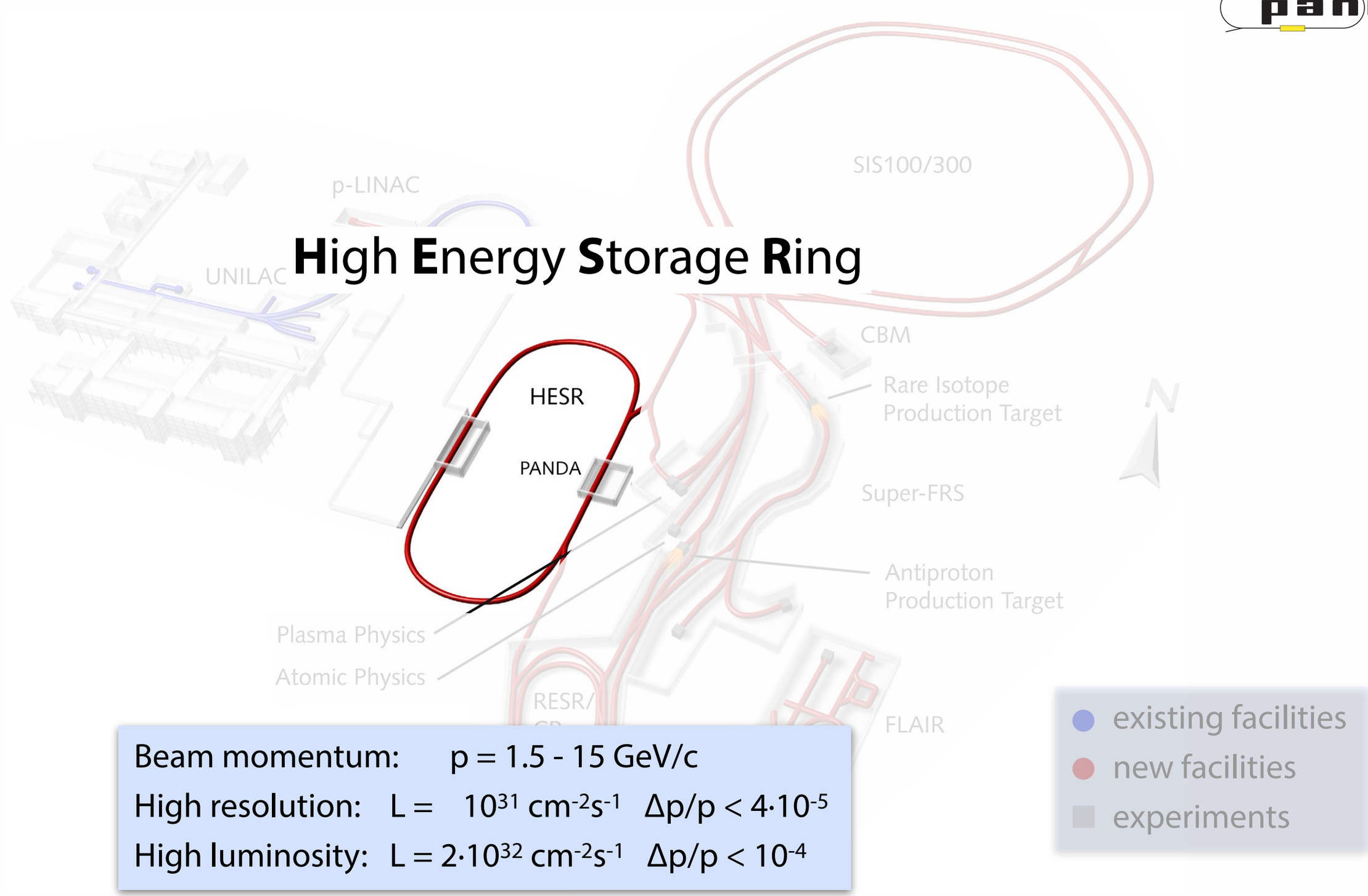
FAIR

- **F**acility for **A**ntiproton and **I**on **R**esearch

FAIR Accelerator Complex



FAIR Accelerator Complex



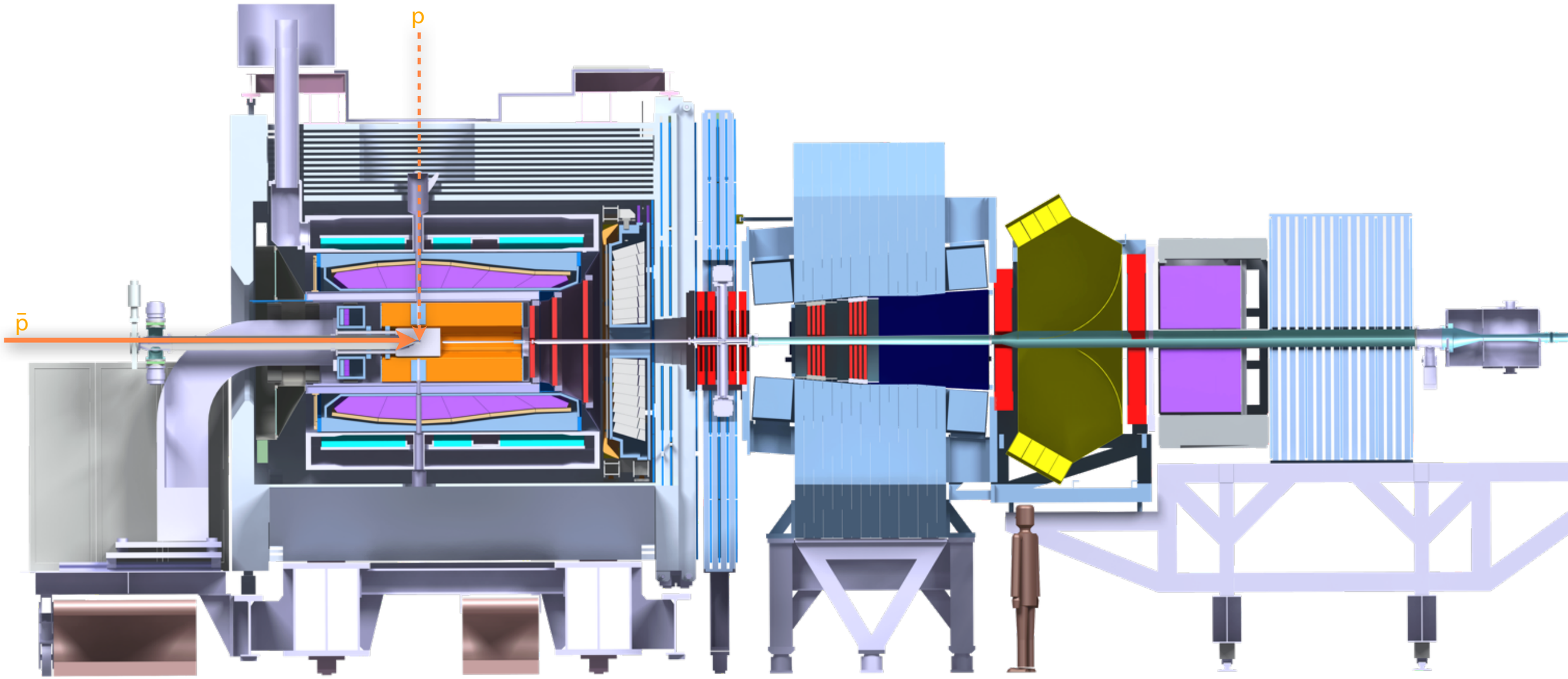
High **E**nergy **S**torage **R**ing

Beam momentum: $p = 1.5 - 15$ GeV/c

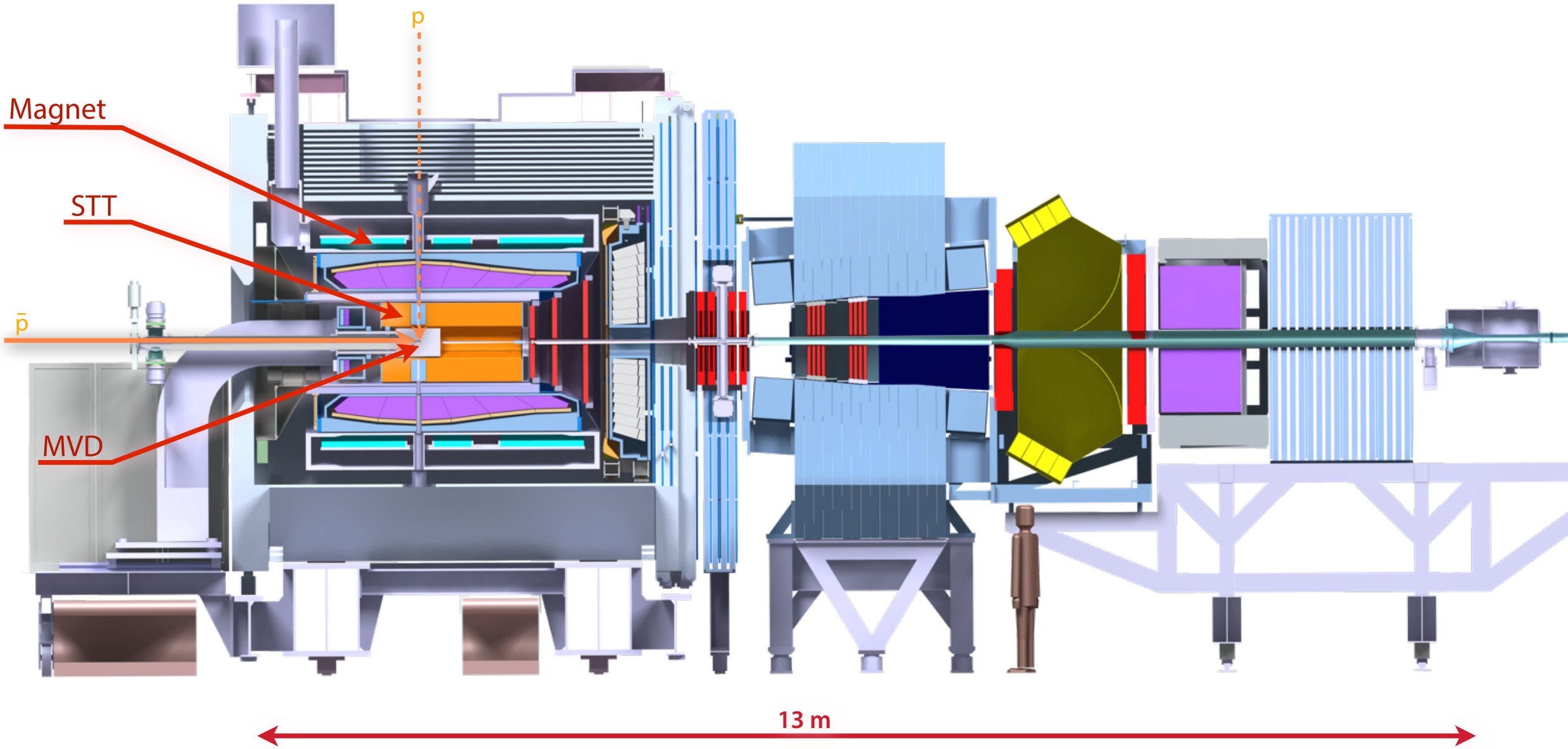
High resolution: $L = 10^{31}\ cm^{-2}s^{-1}$ $\Delta p/p < 4 \cdot 10^{-5}$

High luminosity: $L = 2 \cdot 10^{32}\ cm^{-2}s^{-1}$ $\Delta p/p < 10^{-4}$

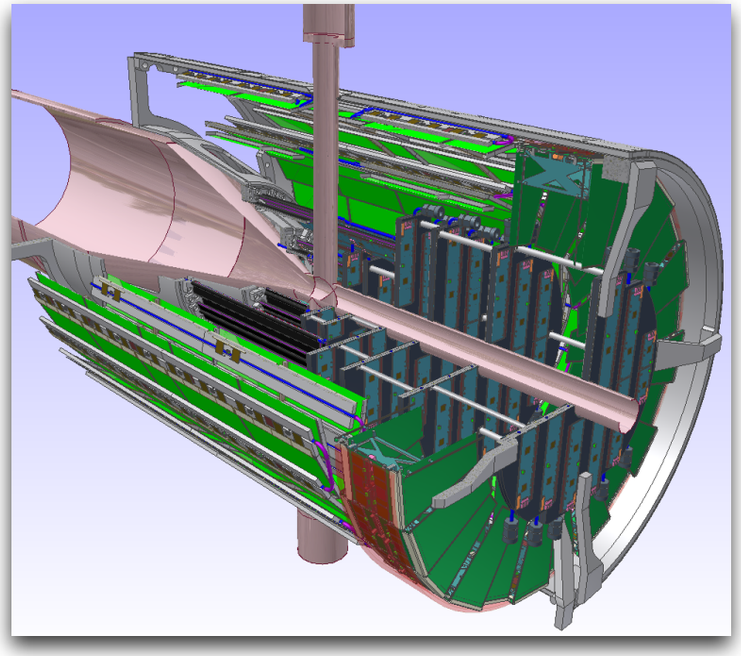
P̄ANDA — The Experiment



P̄ANDA — The Experiment

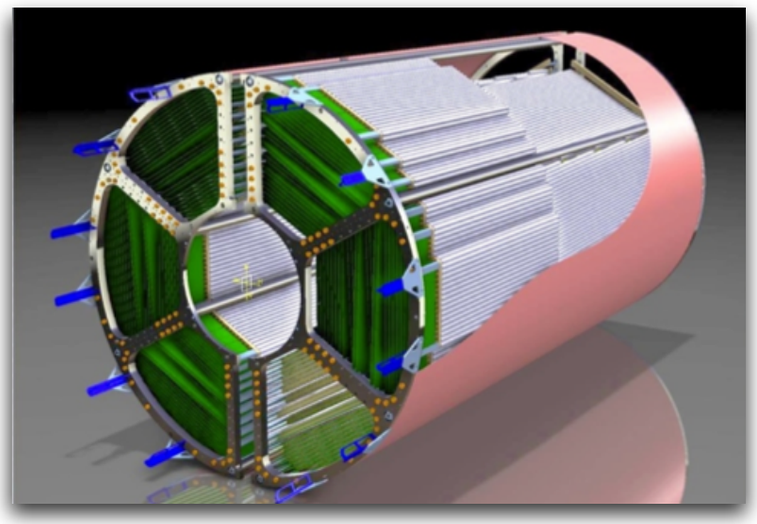


PANDA — MVD & STT



Micro Vertex Detector

- Silicon-based pixel + strip detector
- 10 000 000 + 200 000 channels
- Vertex resolution: < 100 µm

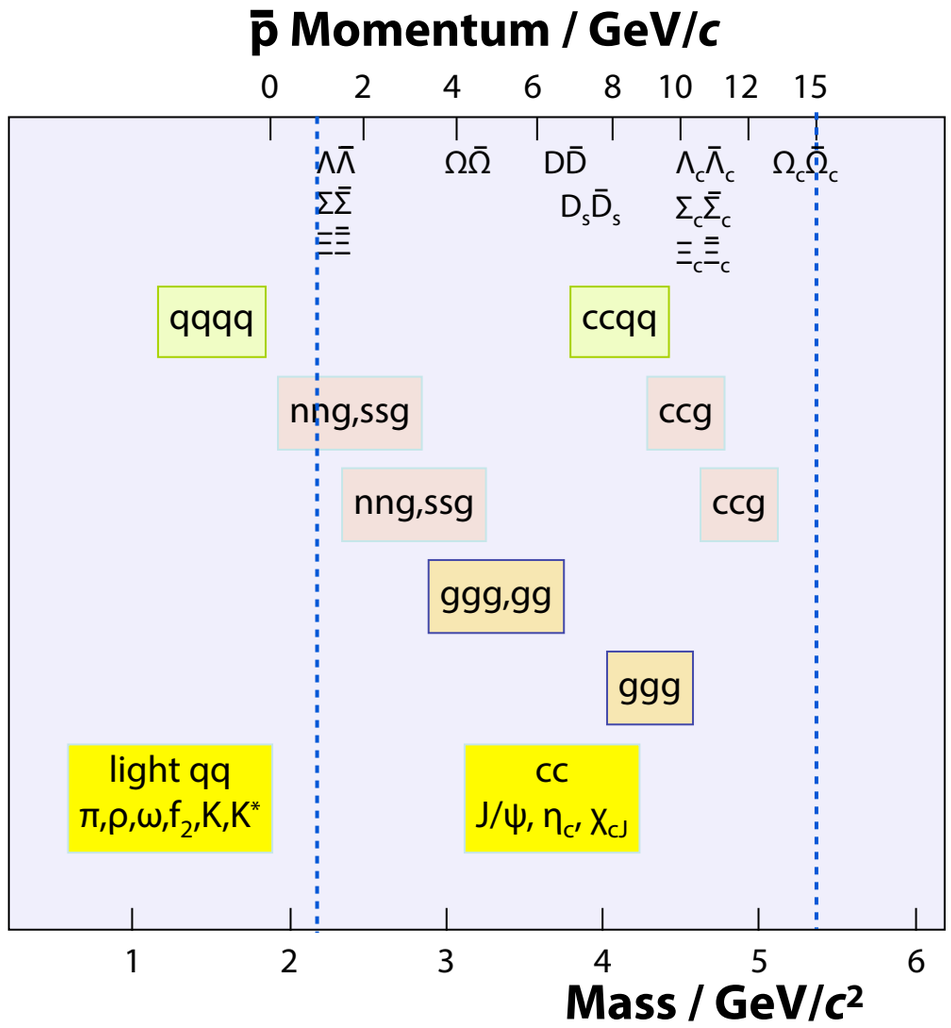


Straw Tube Tracker

- 4636 small drift tubes (ø 1 cm)
  - Drift times: < 250 ns
- 26 layers, 8 skewed
- Material budget: 1.2 % radiation length

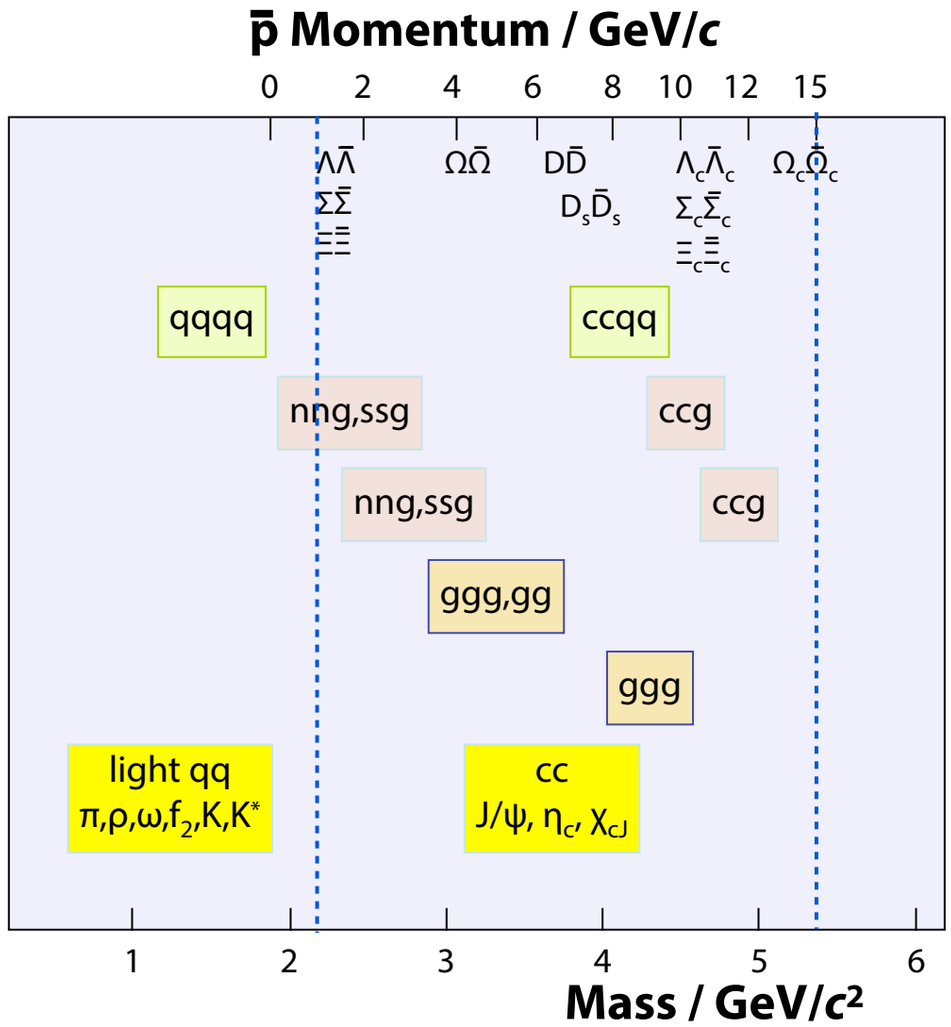
PANDA — Physics

- Meson spectroscopy
  - Light mesons
  - Charmonium
  - Exotic states
    - Glueballs
    - Hybrids
    - Molecules/multiquarks
  - Open charm
- Baryon production
- Nucleon structure, e.m. processes
- Charm in nuclei
- Strangeness physics



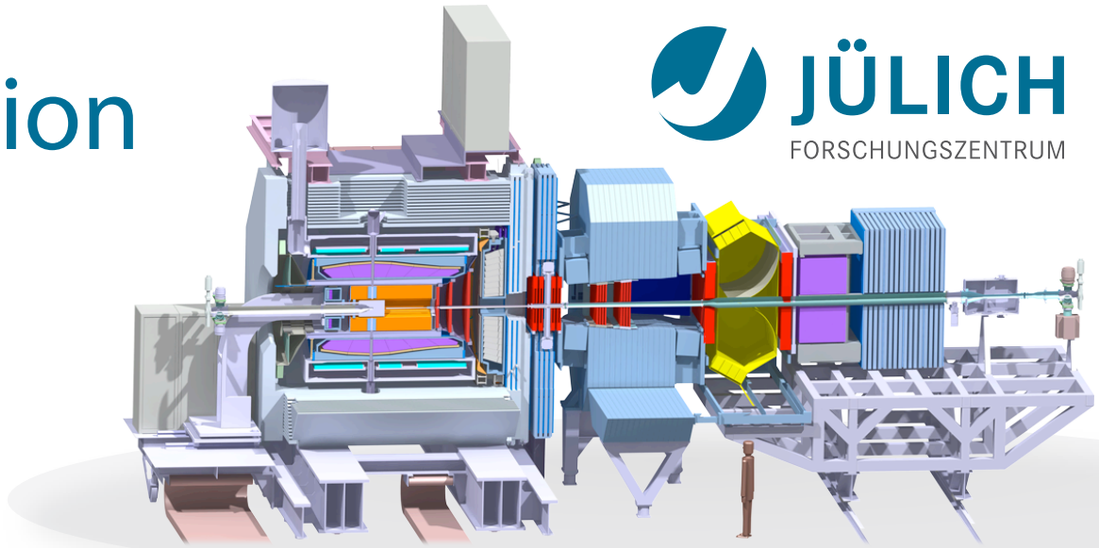
PANDA — Physics

- Meson spectroscopy
  - Light mesons
  - Charmonium
  - Exotic states
    - Glueballs
    - Hybrids
    - Molecules/multiquarks
  - Open charm
- Baryon production
- Nucleon structure, e.m. processes
- Charm in nuclei
- Strangeness physics

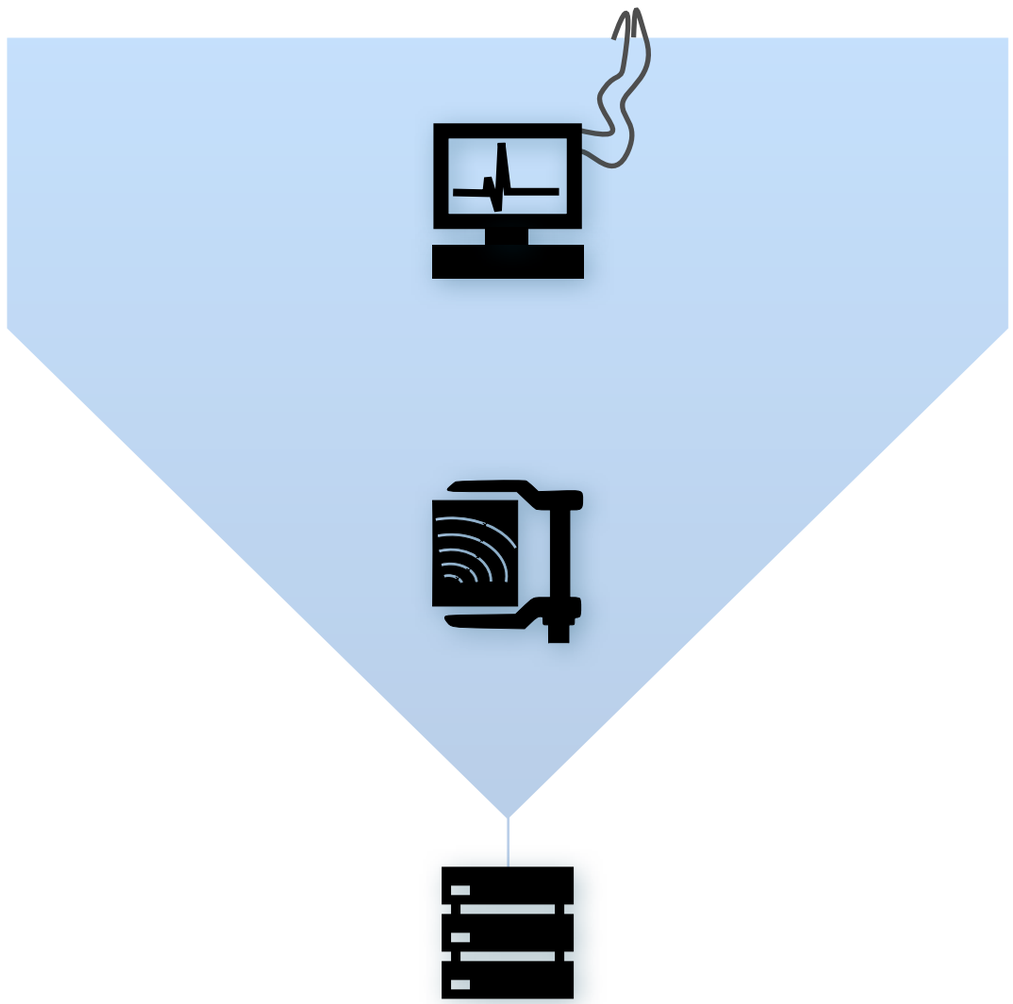


→ Broad **physics** program

P̄ANDA — Event Reconstruction



- **Continuous** read out
  - Background & signal similar
  - *Novel feature*
  - No hardware trigger based on <u>few</u> sub-detectors,
    but online event reconstruction using <u>full</u> detector information

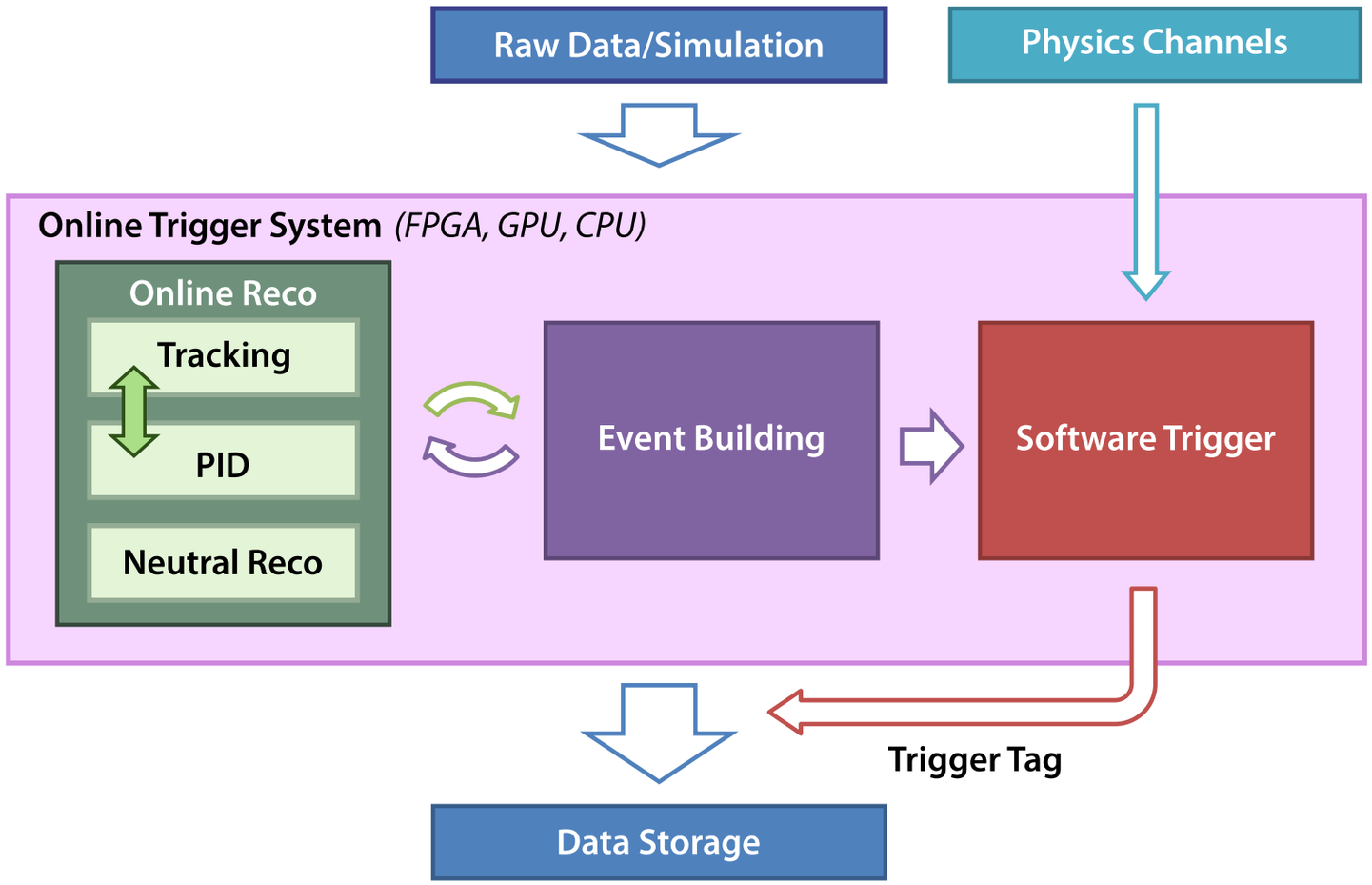


Rate	Event:	**2×10^7/s**
	Raw data:	**200 GB/s**
Reduction	Amount:	**~1/1000**
	Time:	**50 ns/evt**

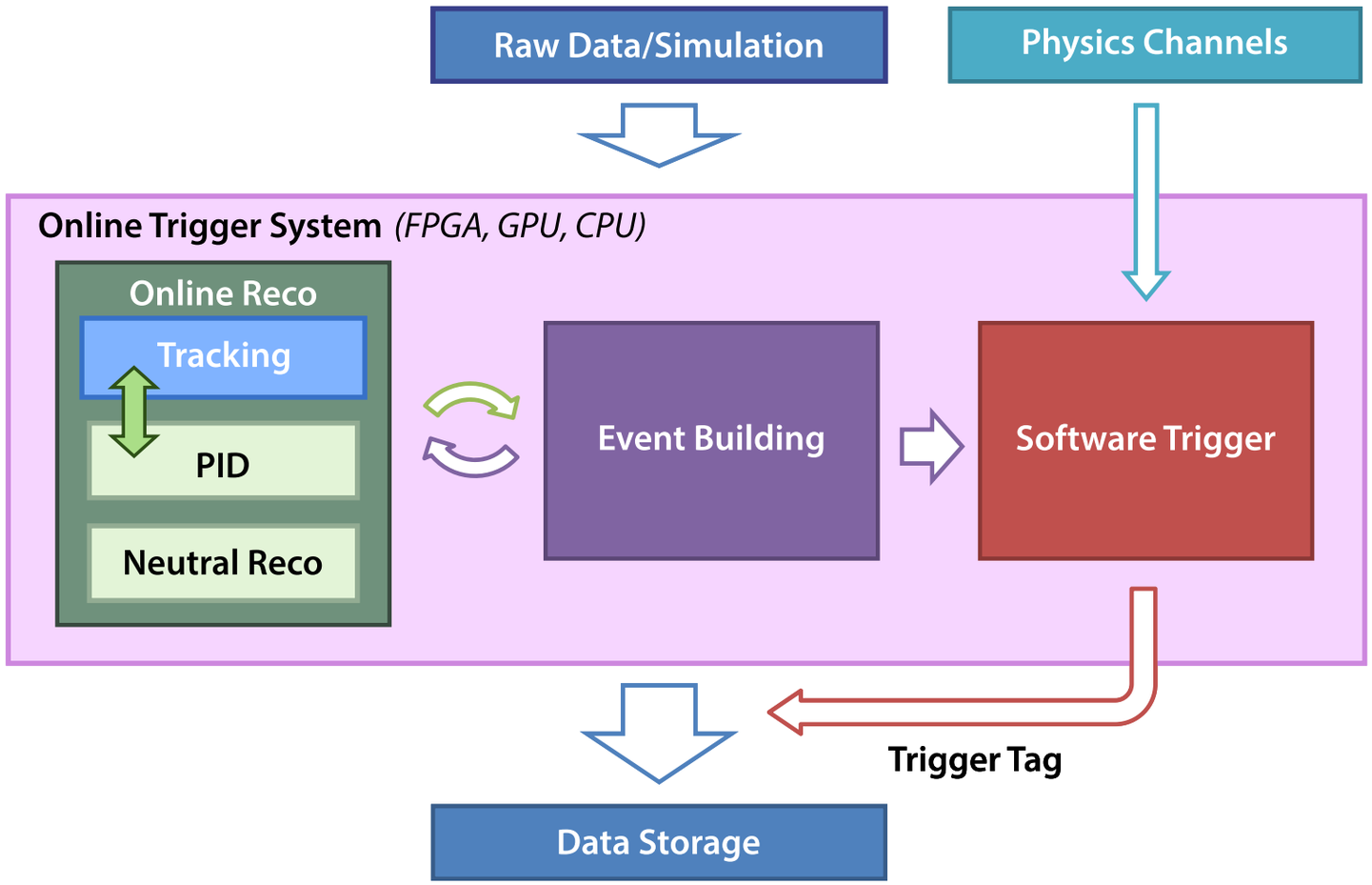
(Reject background events, save interesting events)

Storage space for offline analysis	**3 PB/y**

PANDA — Read Out Scheme

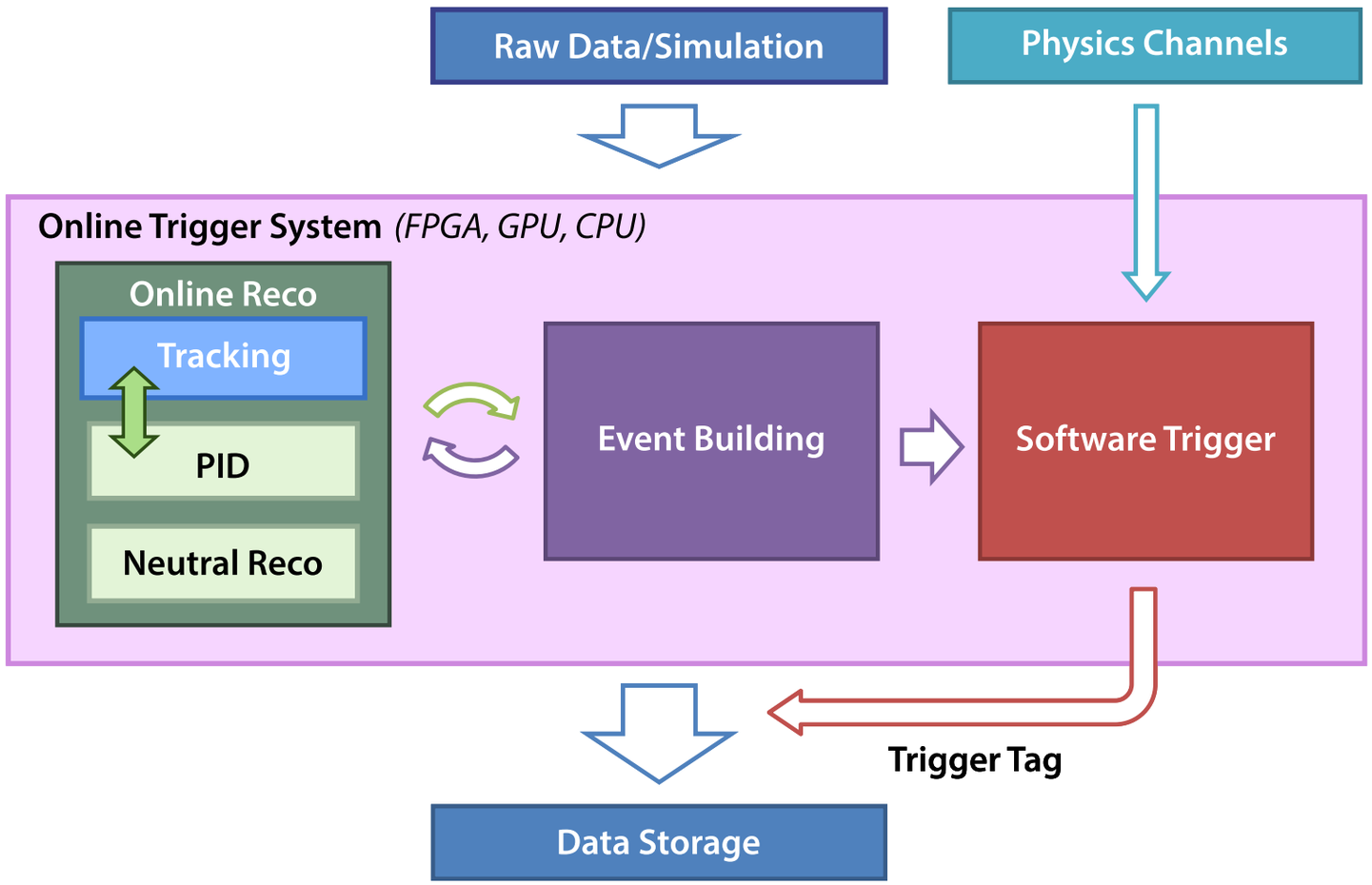


PANDA — Read Out Scheme



- Requirements to Online Tracking
- Fast
- Sophisticated algorithms possible; reprogrammable
- Parallelity beyond single devices
- Fast
- *Limited precision ok*

PANDA — Read Out Scheme



- Requirements to Online Tracking
- Fast
- Sophisticated algorithms possible; reprogrammable
- Parallelity beyond single devices
- Fast
- *Limited precision ok*

GPUs

JÜLICH
FORSCHUNGSZENTRUM

panda

ALGORITHMS

JÜLICH
FORSCHUNGSZENTRUM

panda

ALGORITHMS #1

Hough Transform

Riemann Track Finder

Triplet Finder

Algorithm: Hough Transform

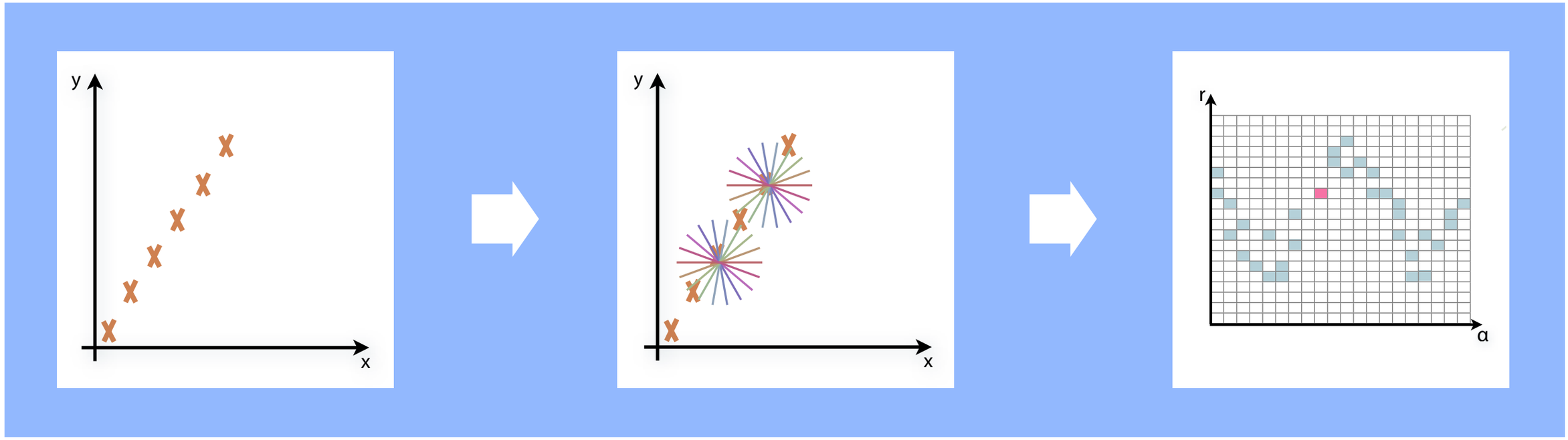
- ***Idea:*** Transform $(x,y)_i \rightarrow (\alpha,r)_{ij}$, find lines via (α,r) space
- Solve r_{ij} line equation for
  - Lots of hits $(x,y,\rho)_i$ and
  - Many $\alpha_j \in [0°,360°)$ each
- Fill histogram

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

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

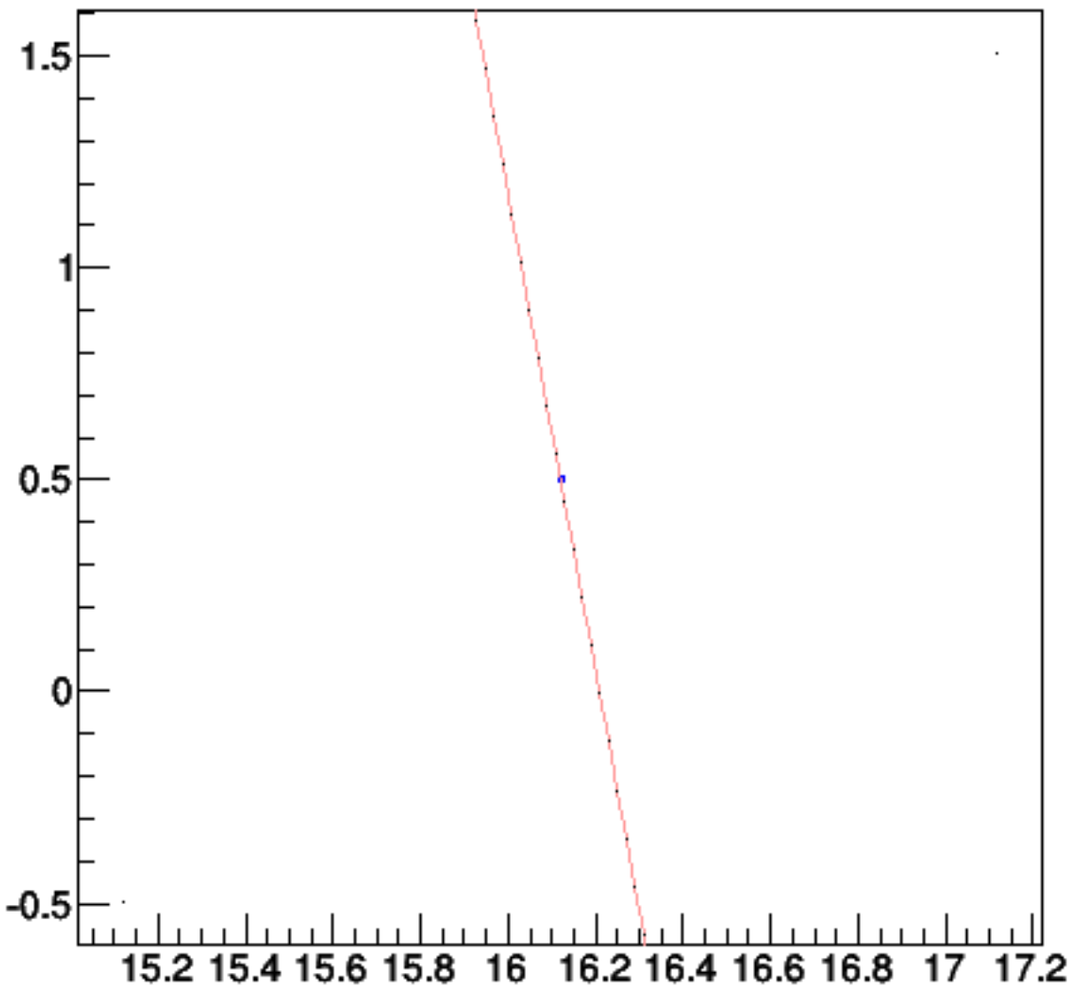
⇨ r_{ij}: 180 000

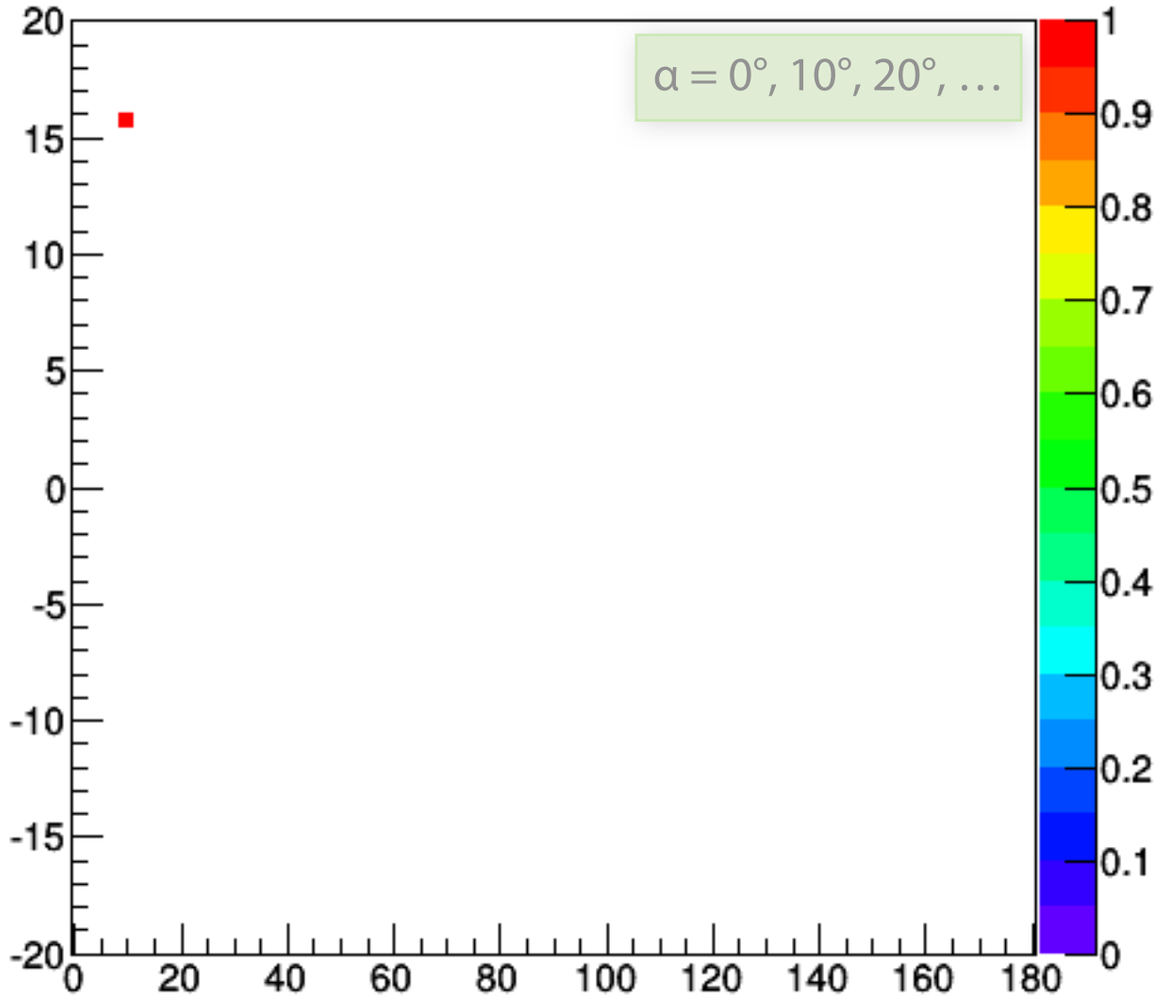
- Extract track parameters



Hough Transform — Visualization Lines

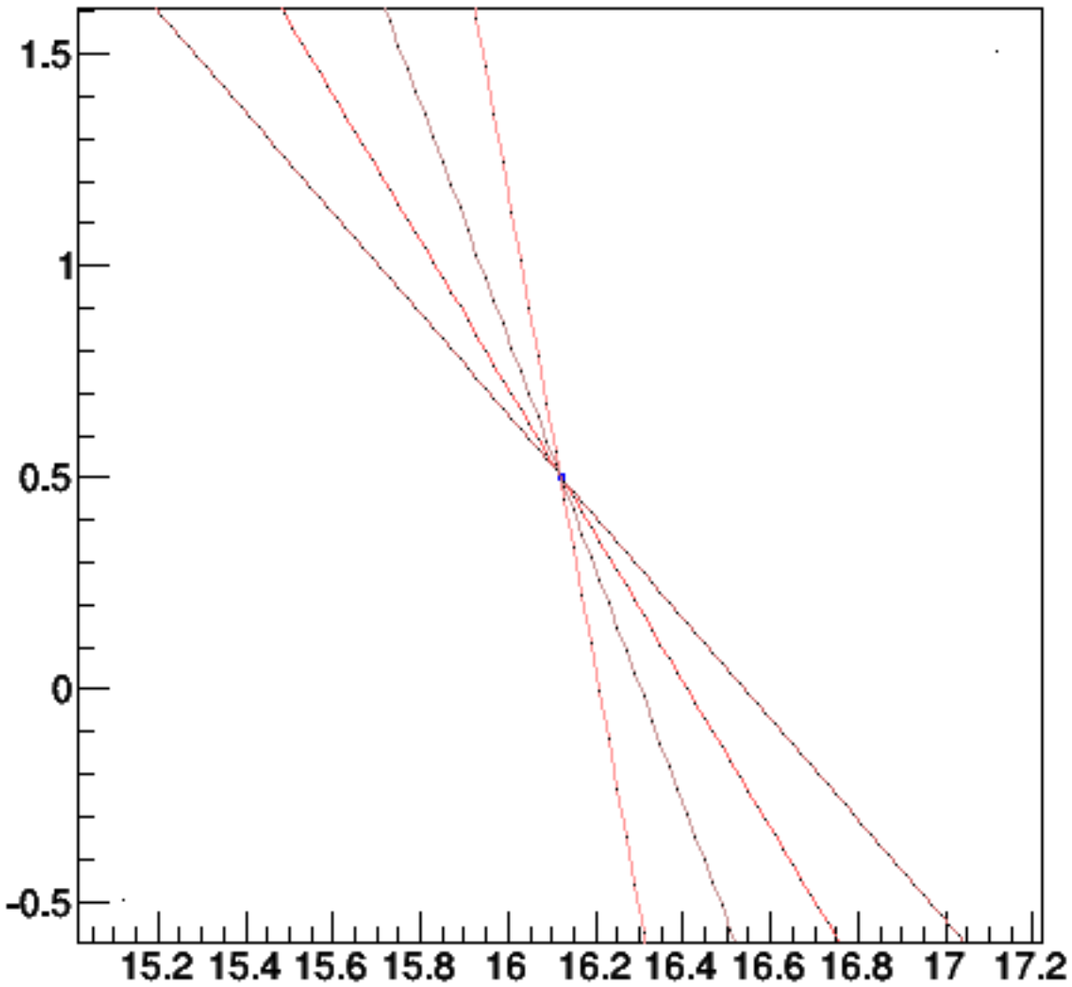
- Create lines going through hit point $(x,y)_i$
  - Line parameterized by $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$
- Fill line parameters $(\alpha,r)_{ij}$ into histogram
  - Rasterize for many $\alpha_j \in [0°,180°)$

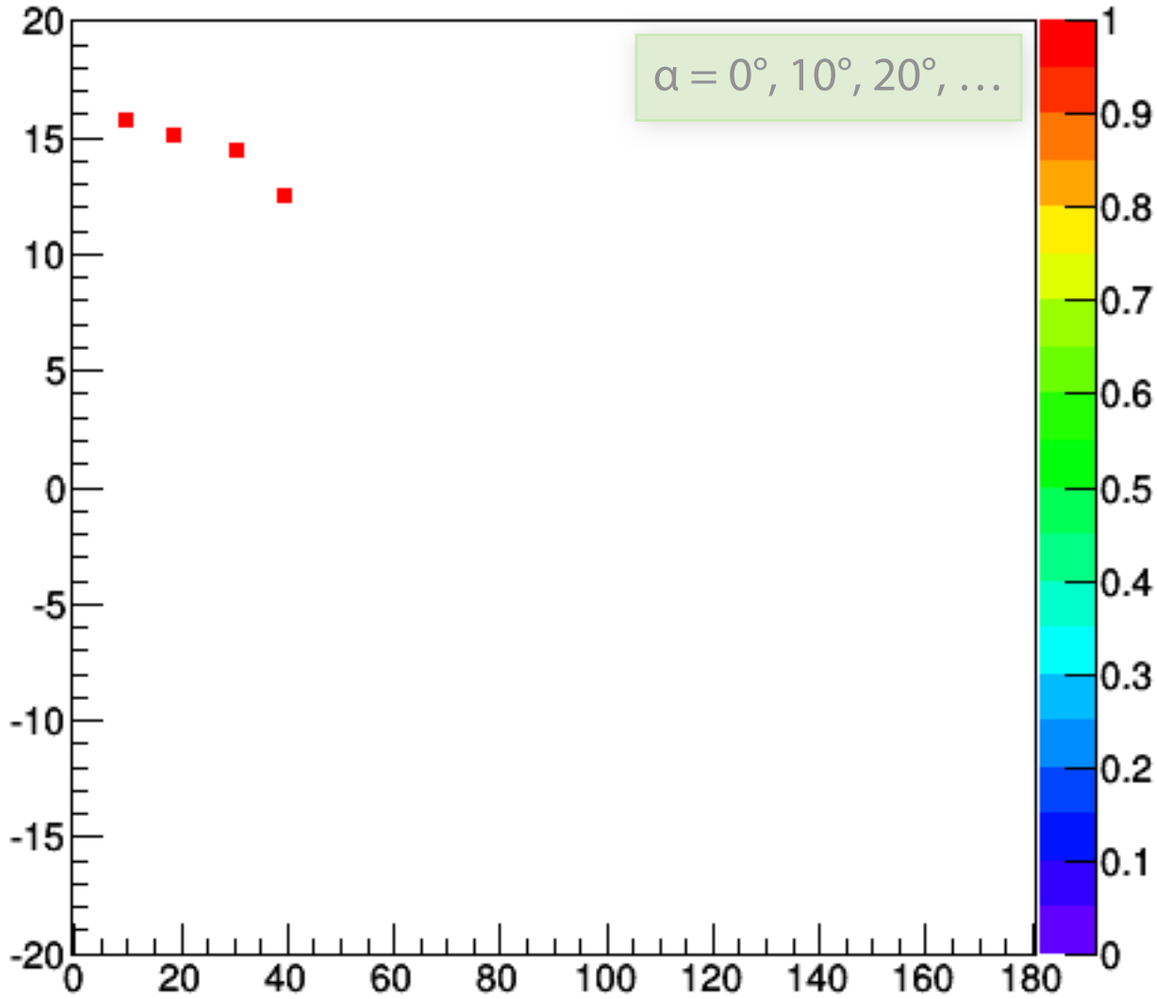




Hough Transform — Visualization Lines

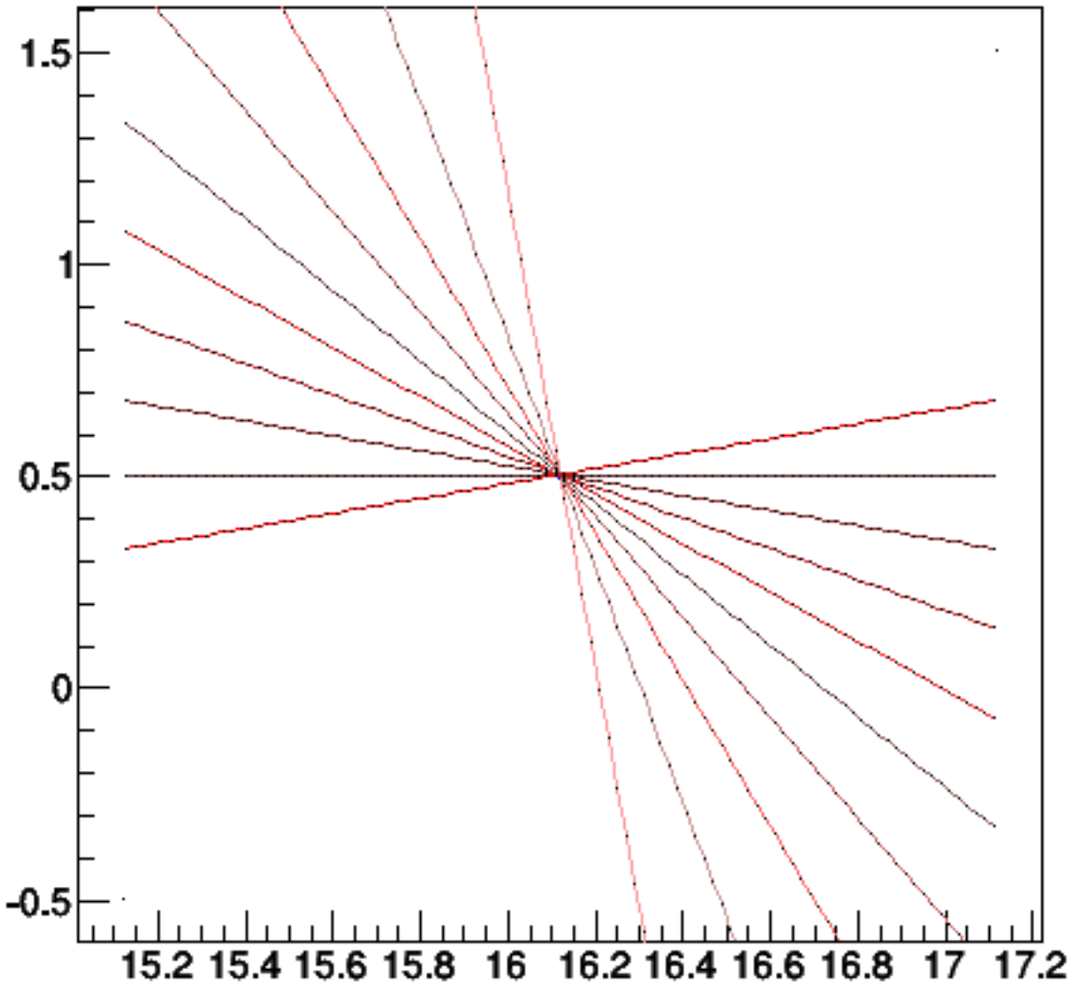
- Create lines going through hit point $(x,y)_i$
  - Line parameterized by $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$
- Fill line parameters $(\alpha,r)_{ij}$ into histogram
  - Rasterize for many $\alpha_j \in [0°,180°)$

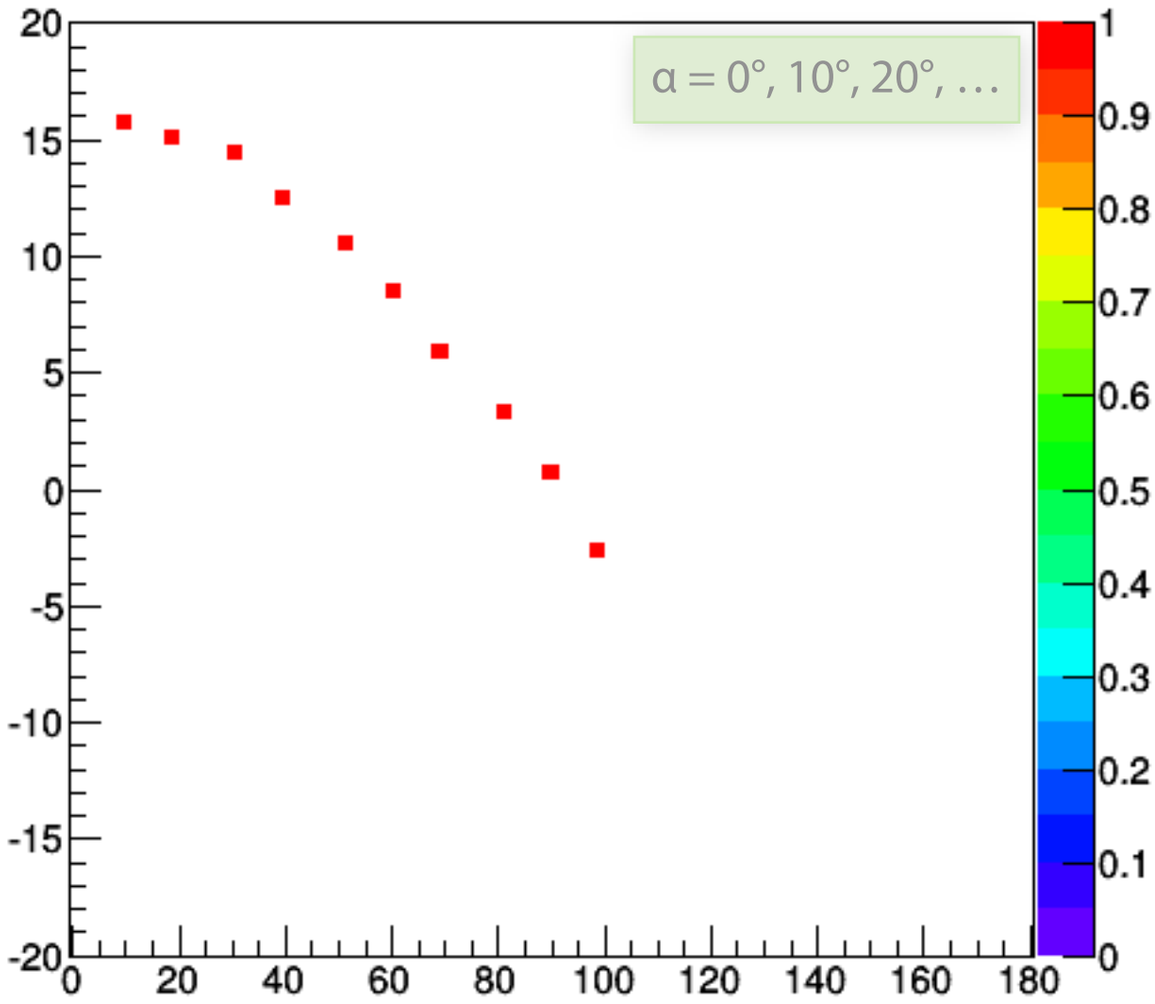




Hough Transform — Visualization Lines

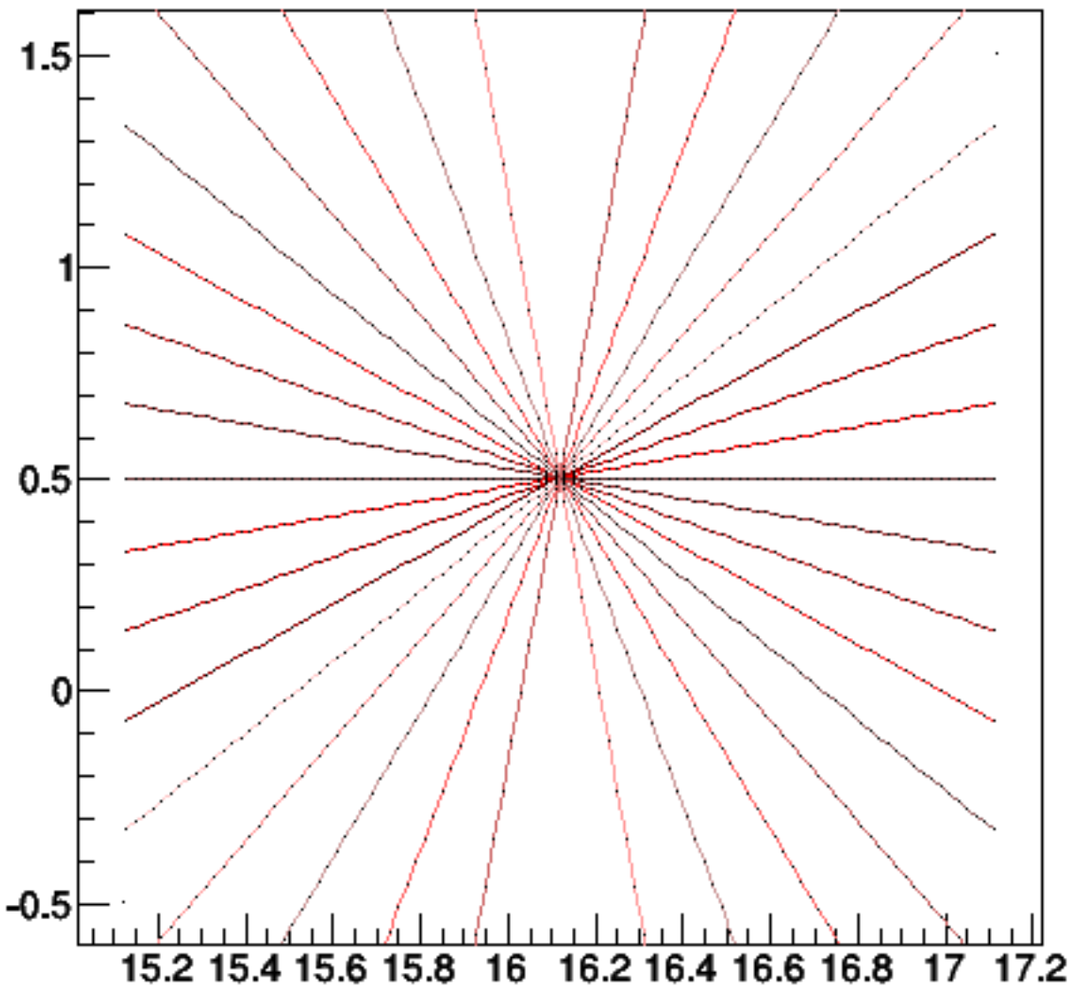
- Create lines going through hit point $(x,y)_i$
  - Line parameterized by $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$
- Fill line parameters $(\alpha,r)_{ij}$ into histogram
  - Rasterize for many $\alpha_j \in [0°,180°)$

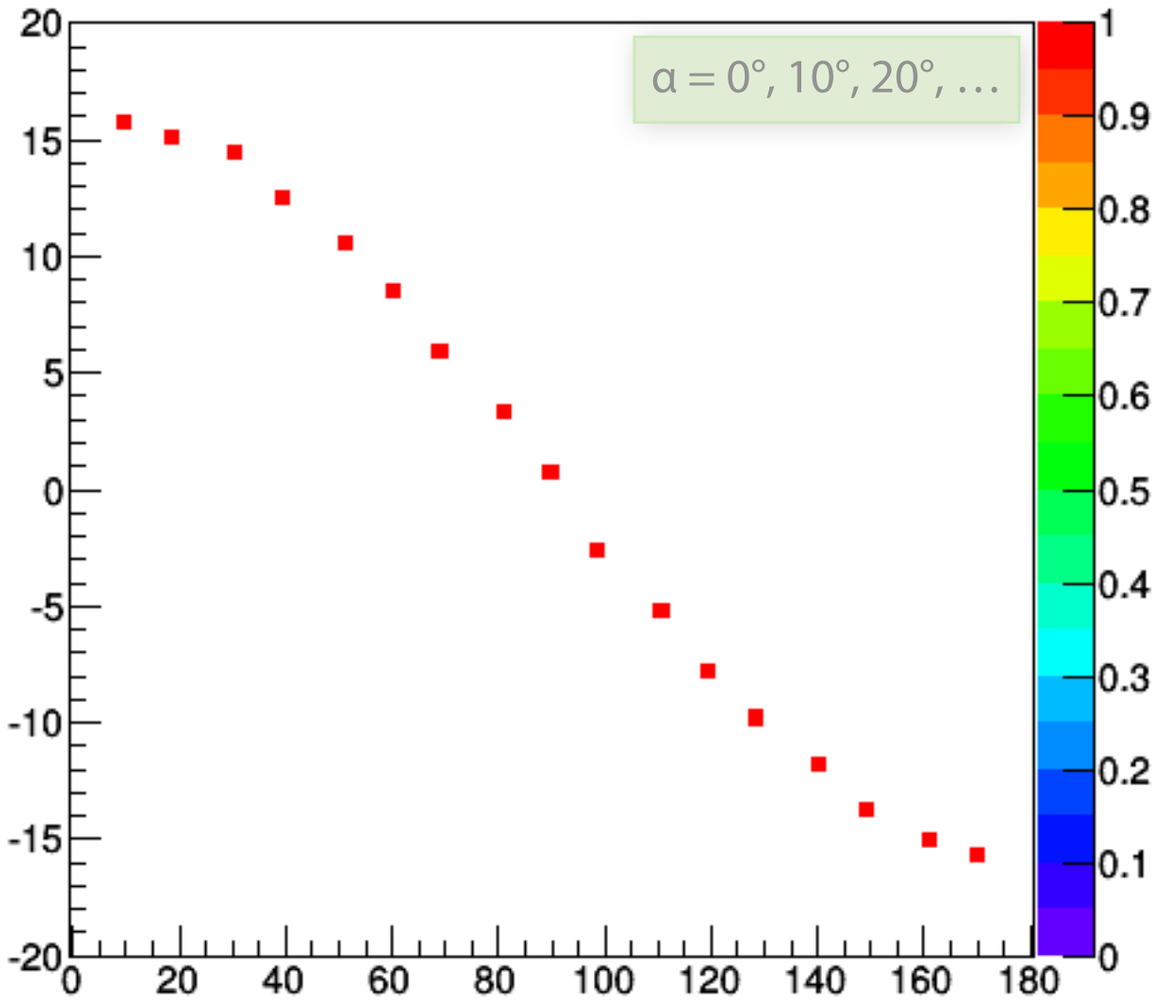




Hough Transform — Visualization Lines

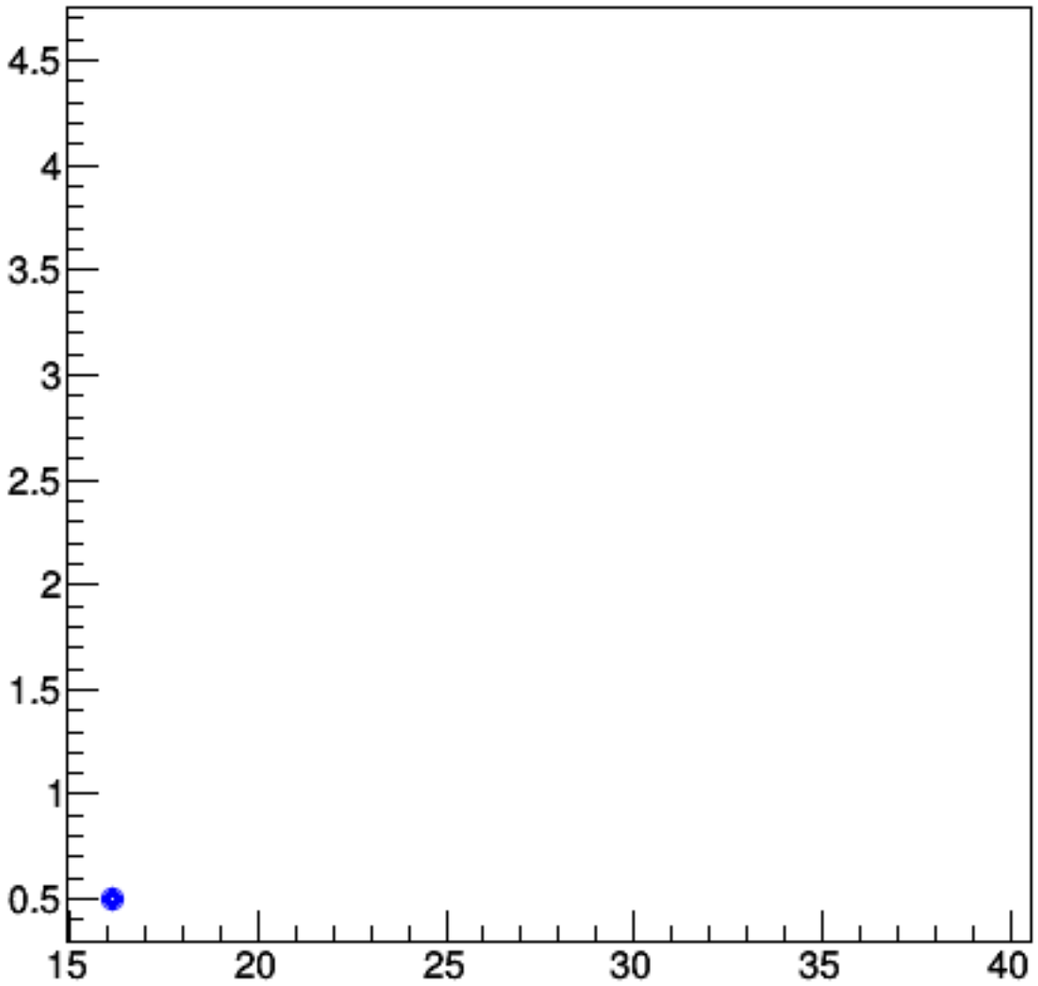
- Create lines going through hit point $(x,y)_i$
  - Line parameterized by $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$
- Fill line parameters $(\alpha,r)_{ij}$ into histogram
  - Rasterize for many $\alpha_j \in [0°,180°)$

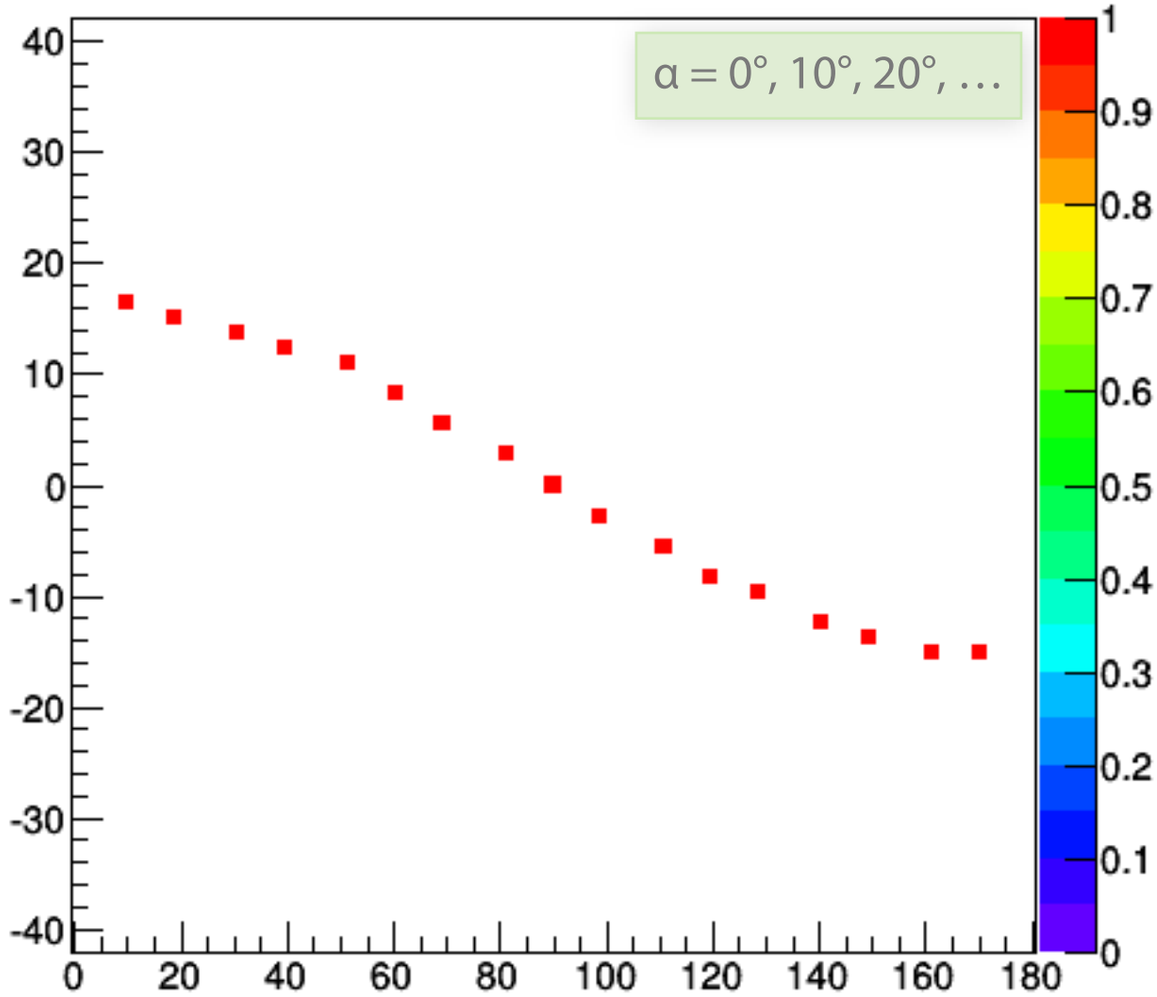




Hough Transform — Visualization Points

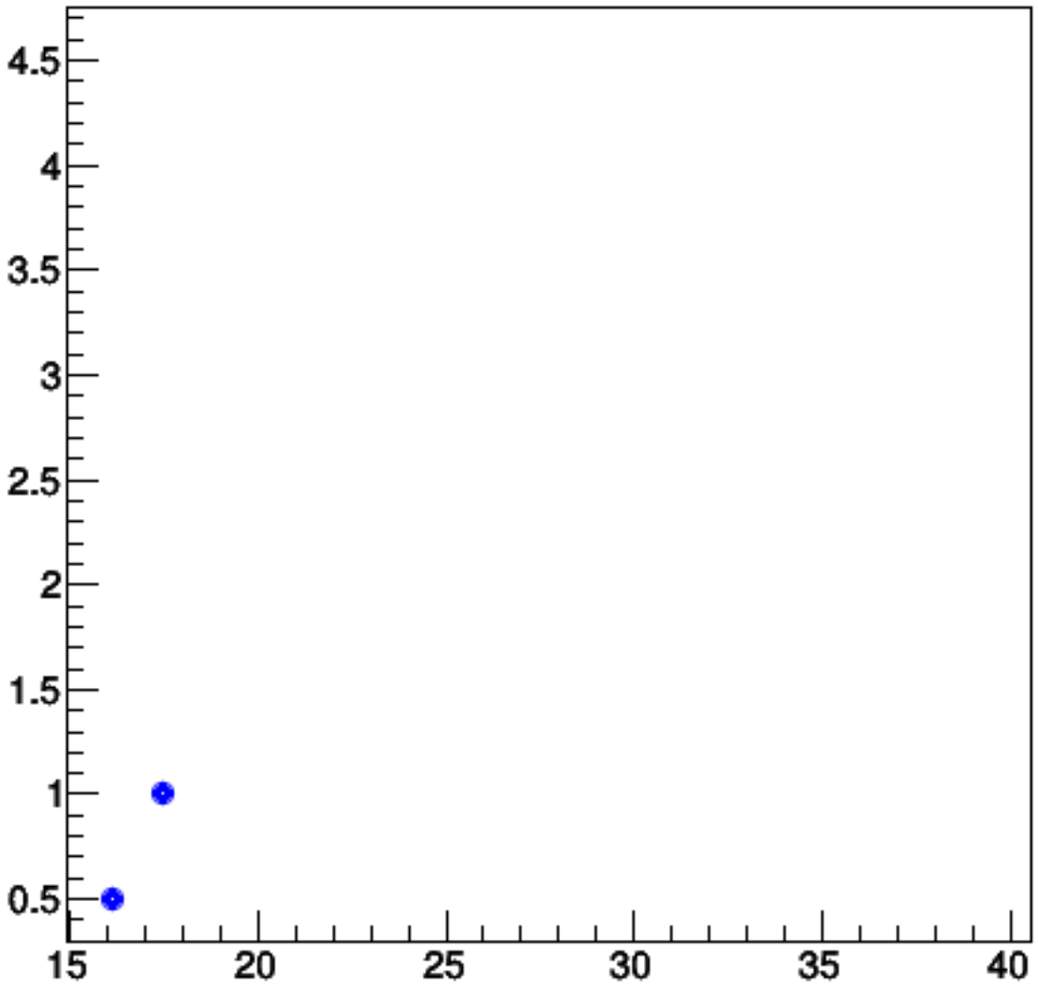
- Create lines going through hit point $(x,y)_i$
- **Repeat** for every hit point i

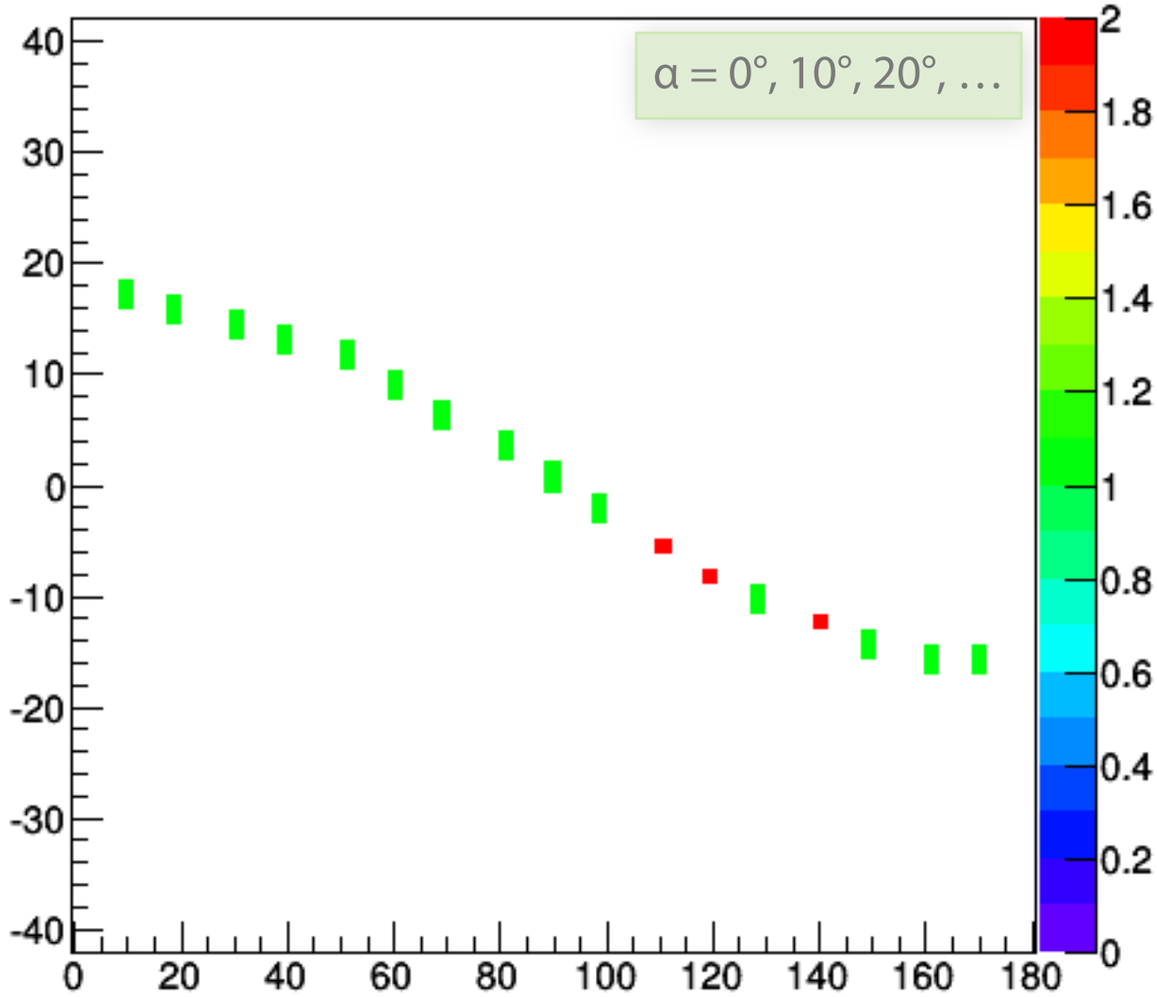




Hough Transform — Visualization Points

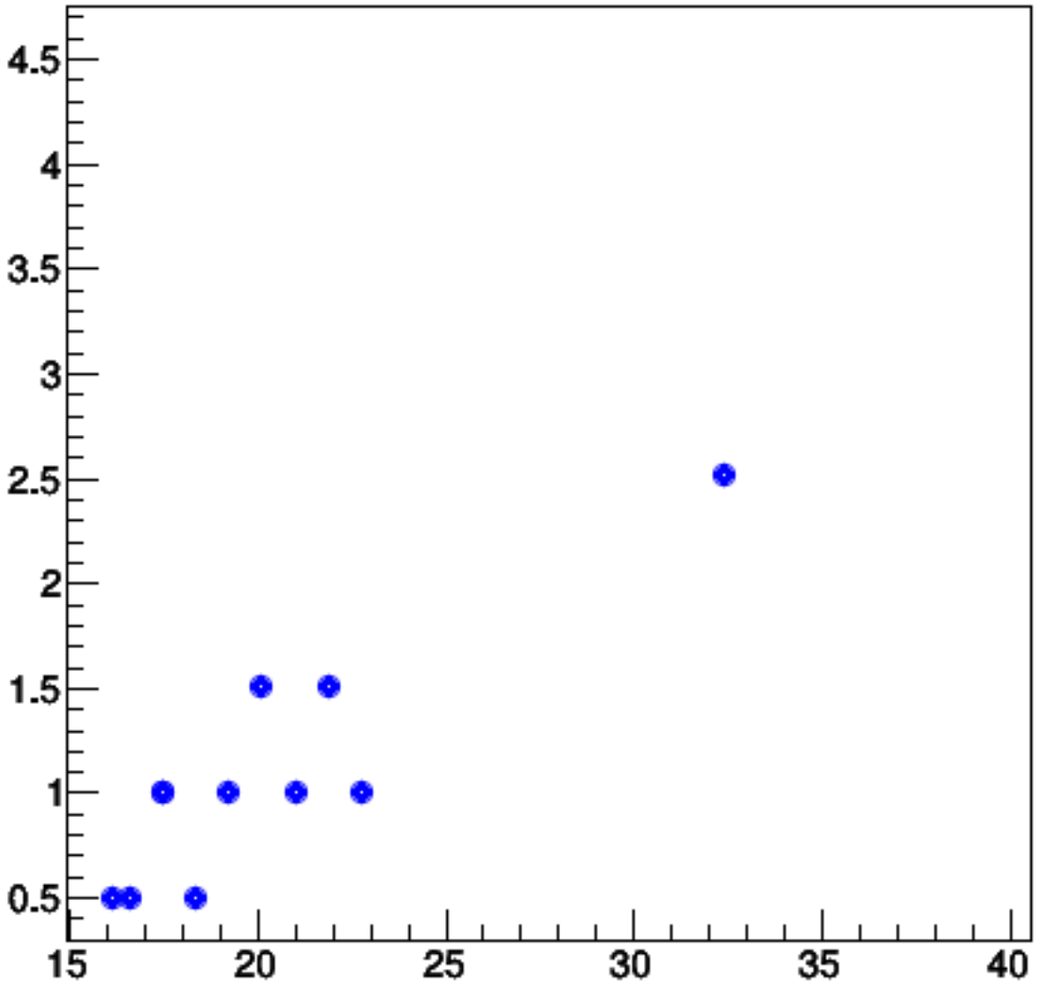
- Create lines going through hit point $(x,y)_i$
- **Repeat** for every hit point i

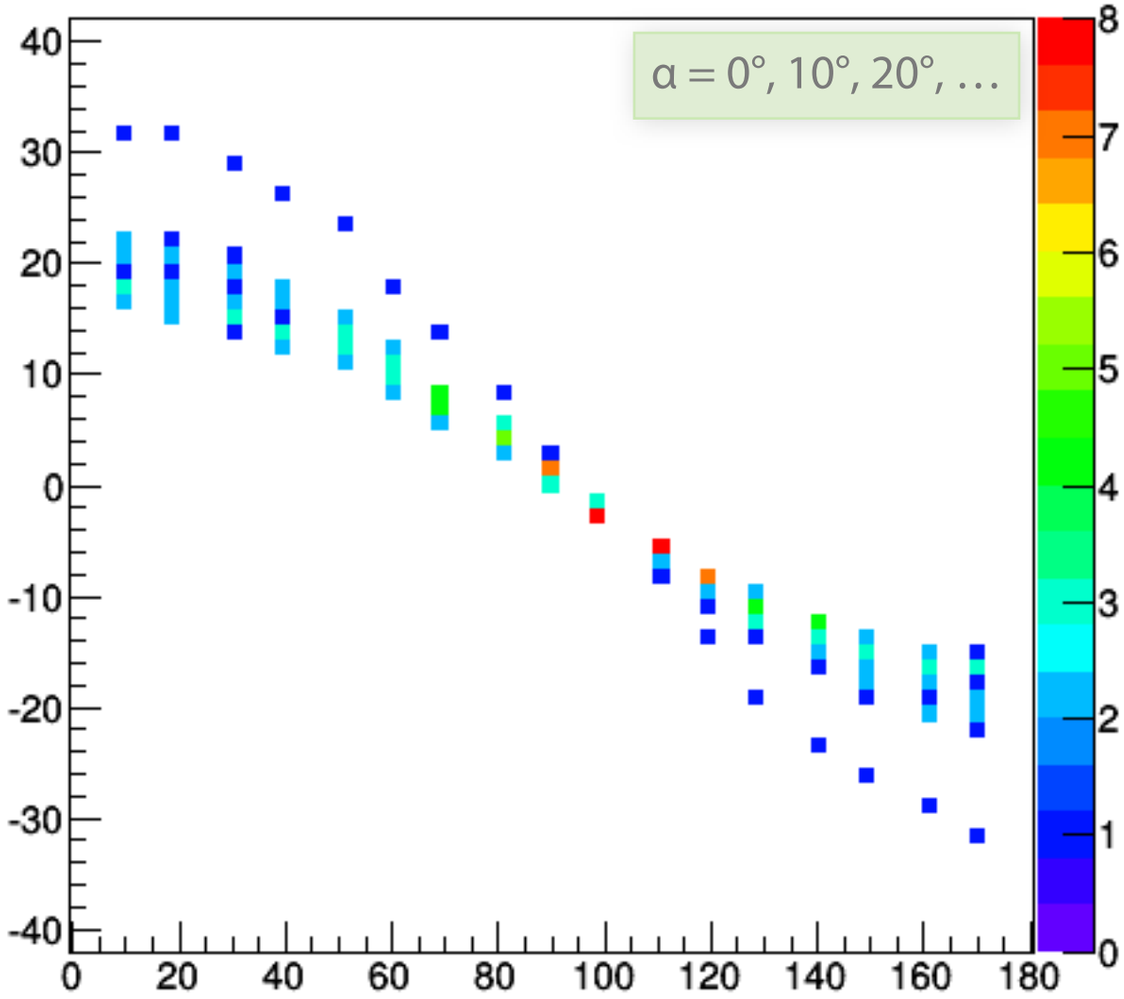




Hough Transform — Visualization Points

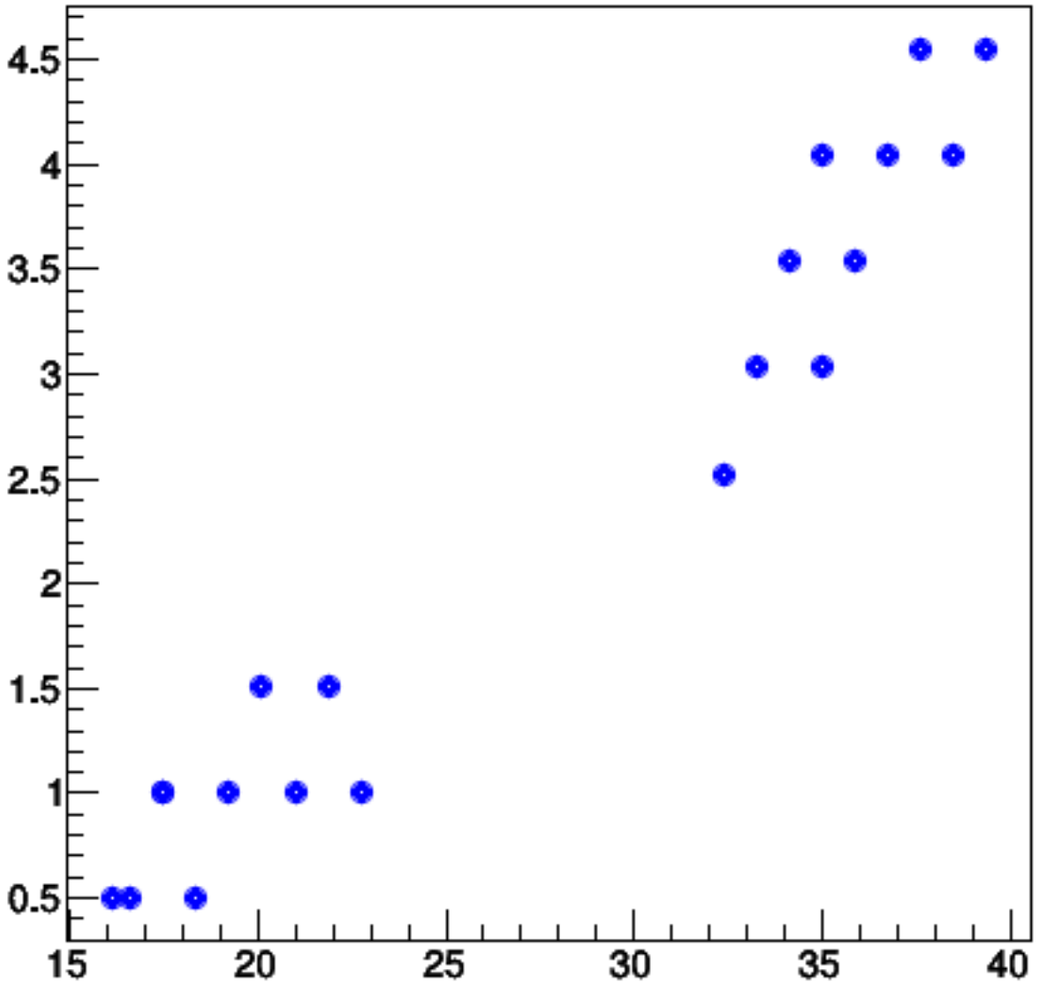
- Create lines going through hit point $(x,y)_i$
- **Repeat** for every hit point i

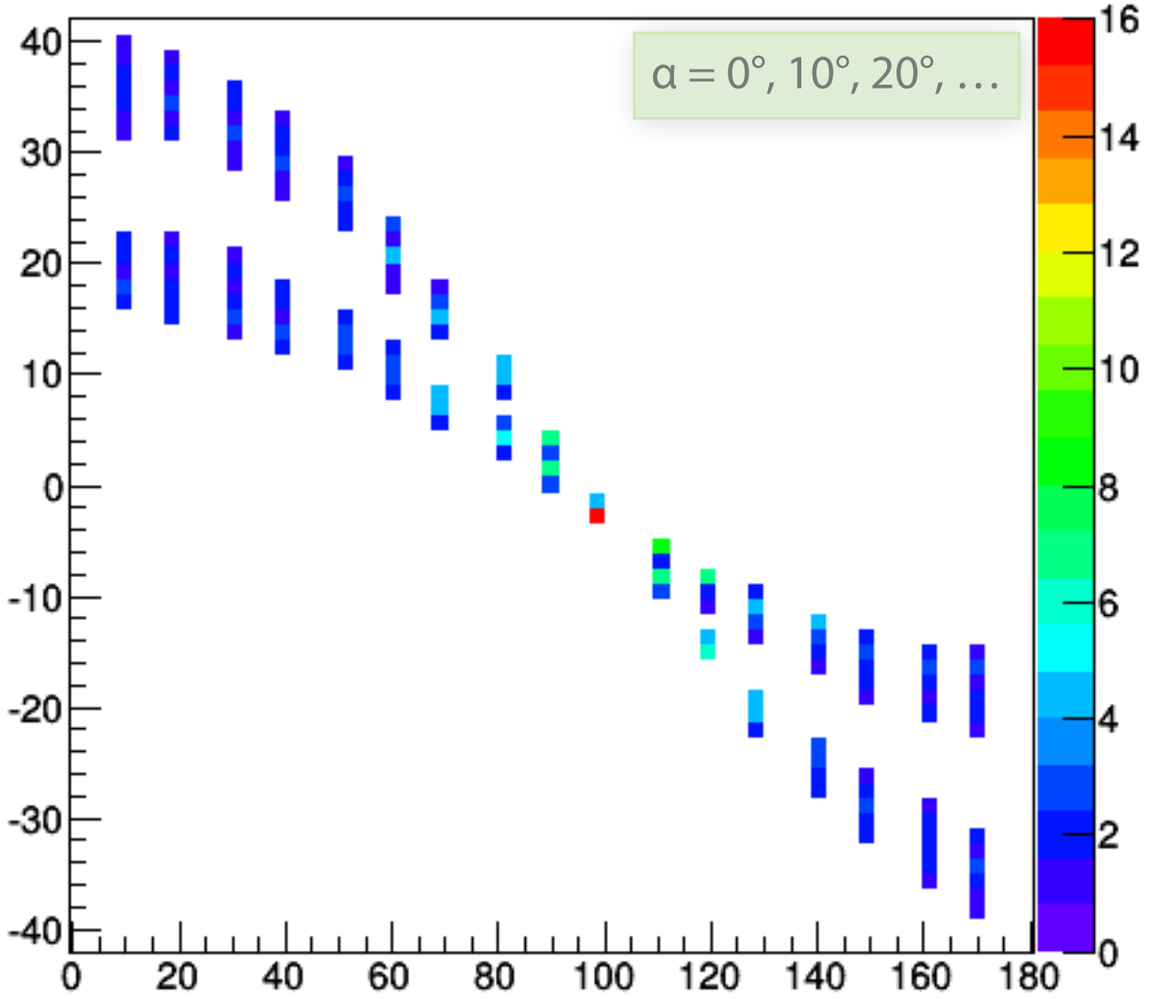




Hough Transform — Visualization Points

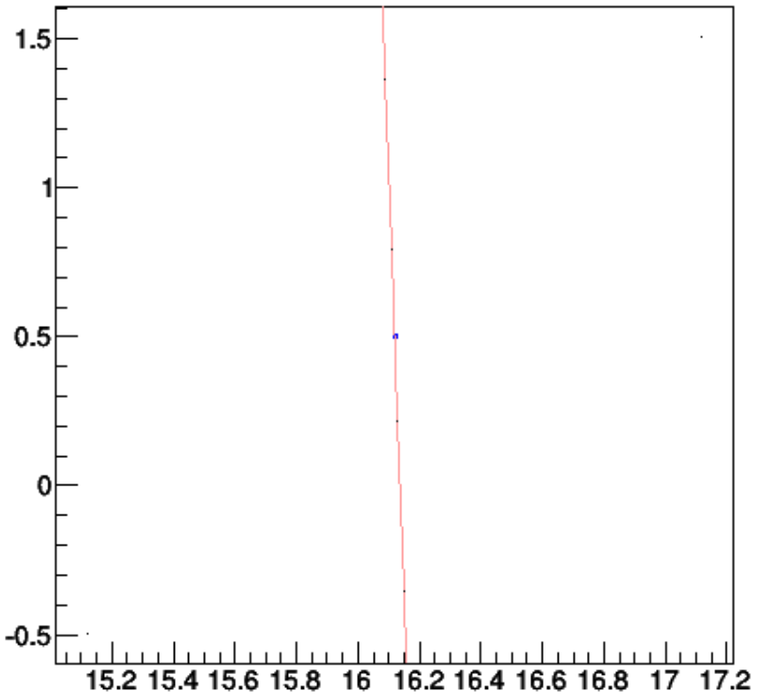
- Create lines going through hit point $(x,y)_i$
- **Repeat** for every hit point i

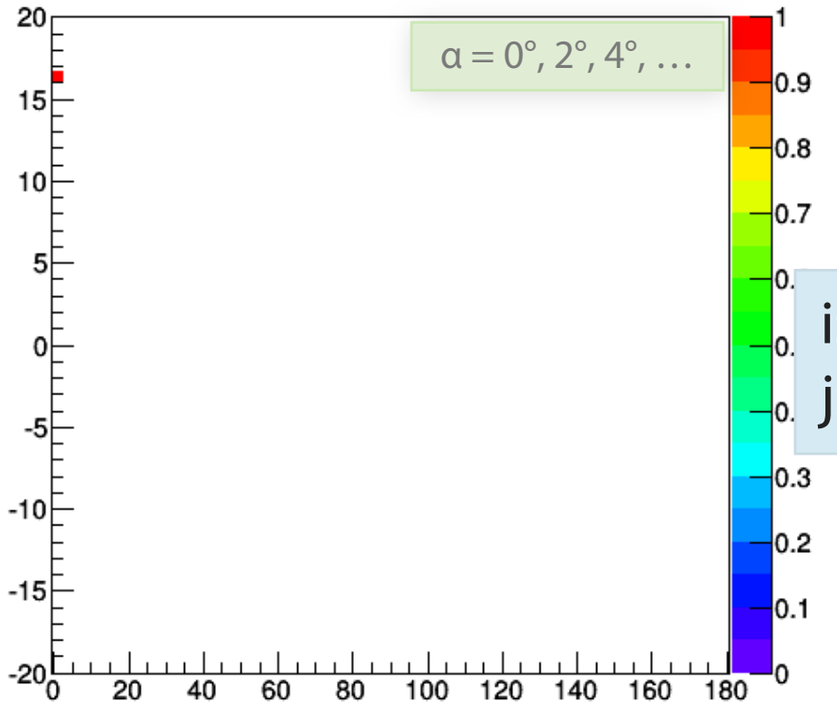




Hough Transform — Granularity

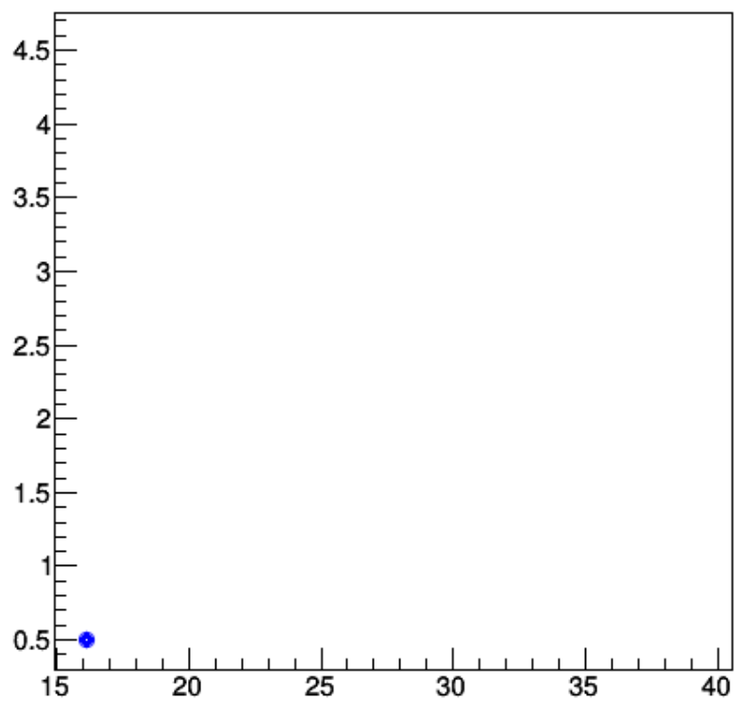
- Choice of α granularity determines resolution

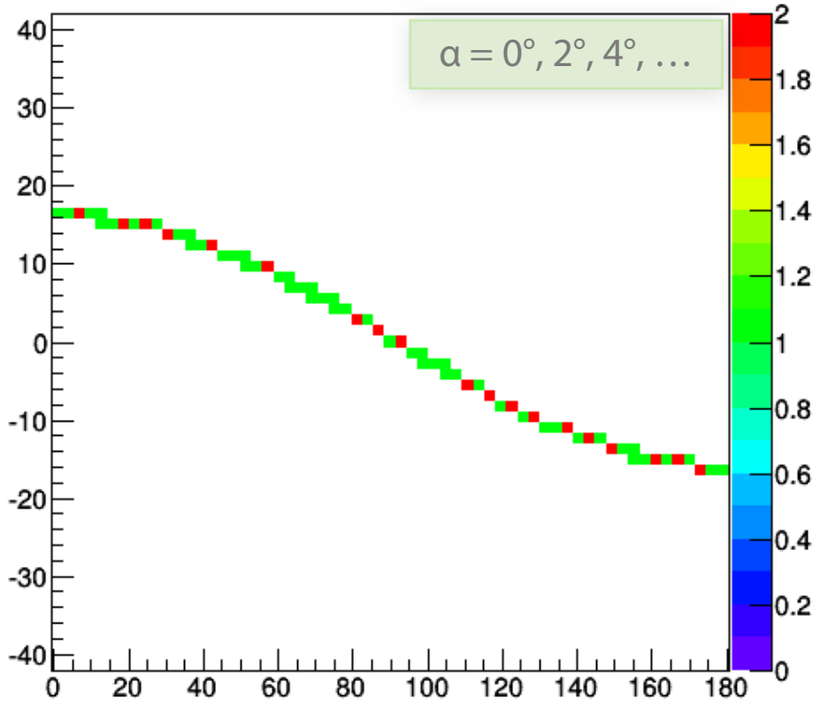




i: ~100 hits/event (STT)
j: every 0.2°

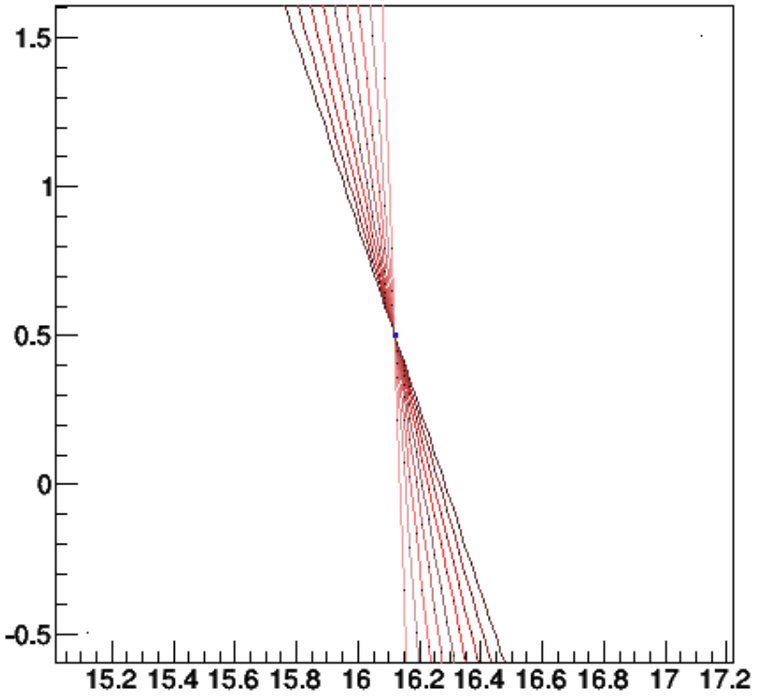
r_{ij}: 180 000

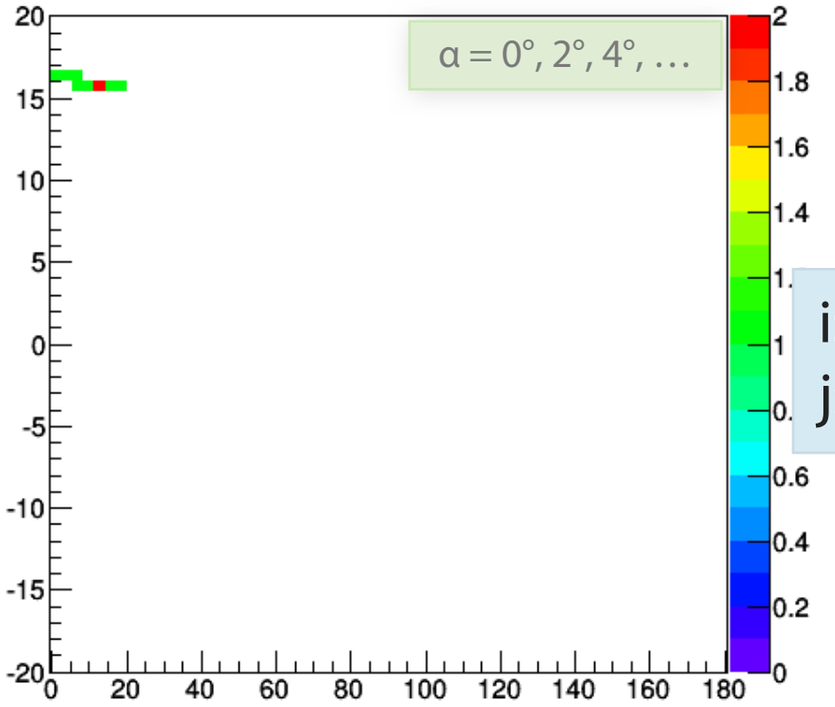




Hough Transform — Granularity

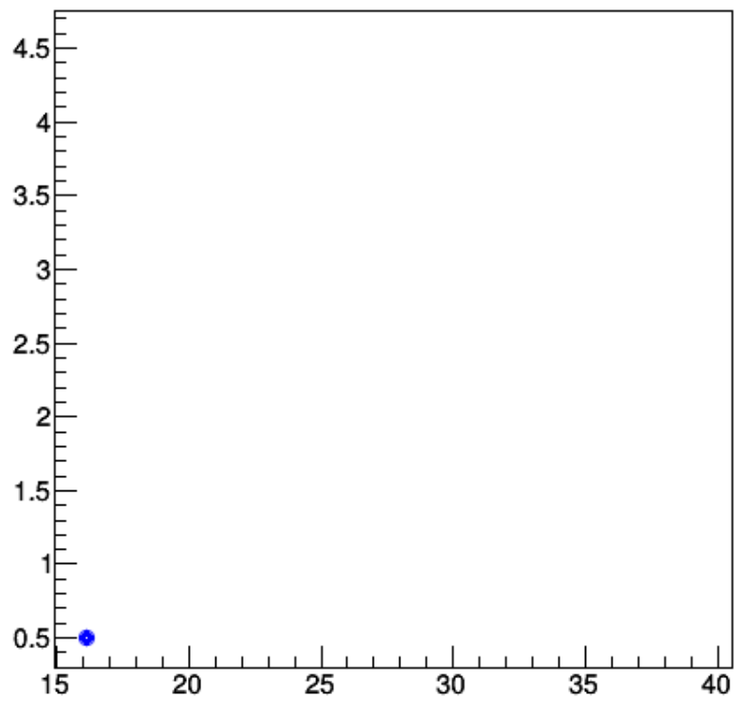
- Choice of α granularity determines resolution

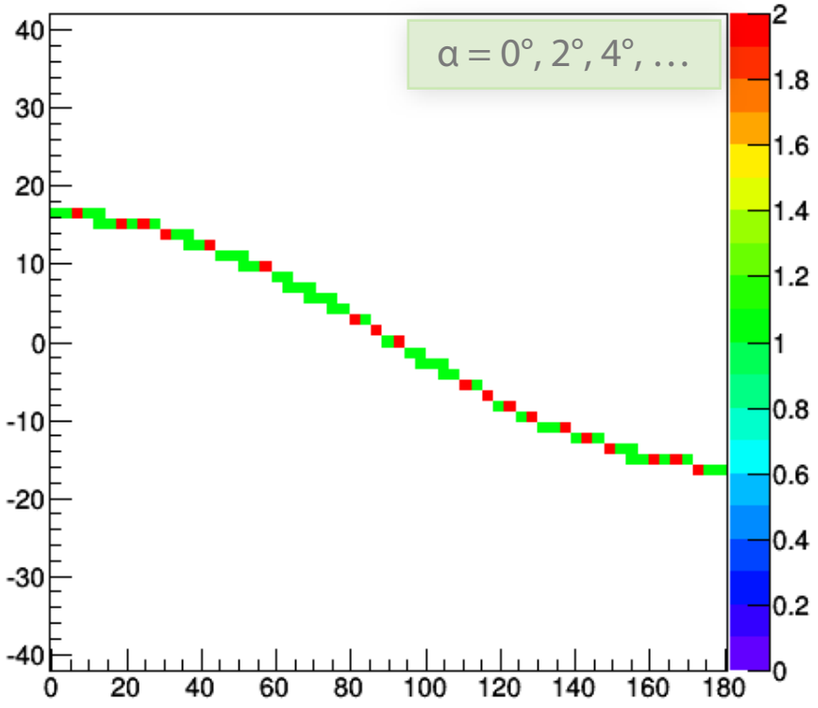




i: ~100 hits/event (STT)
j: every 0.2°

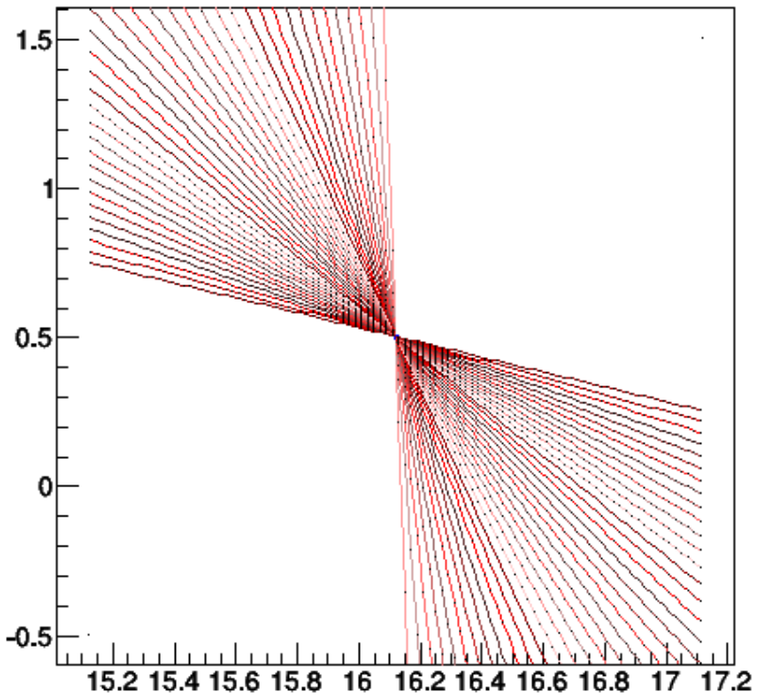
r_{ij}: 180 000

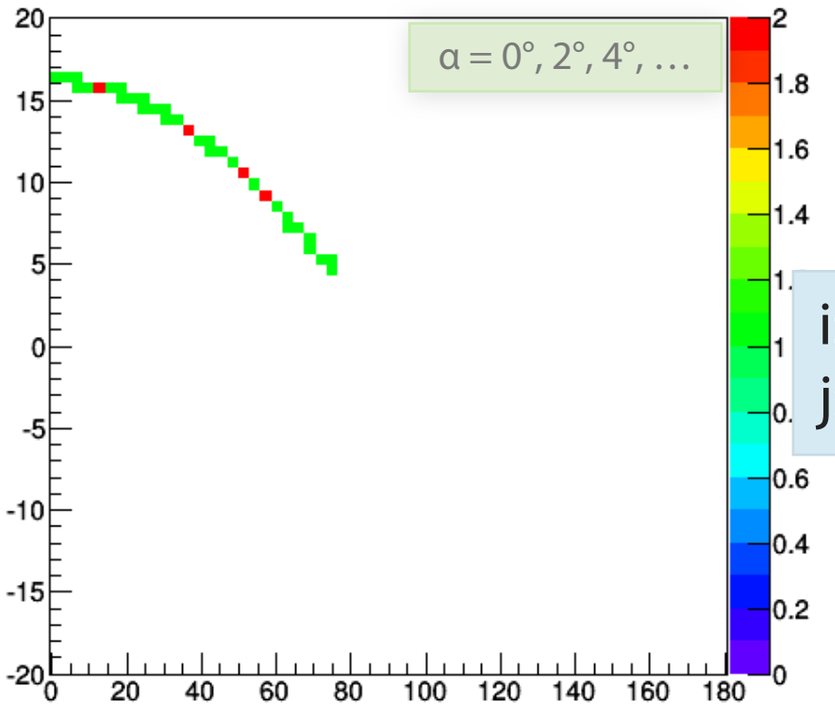




Hough Transform — Granularity

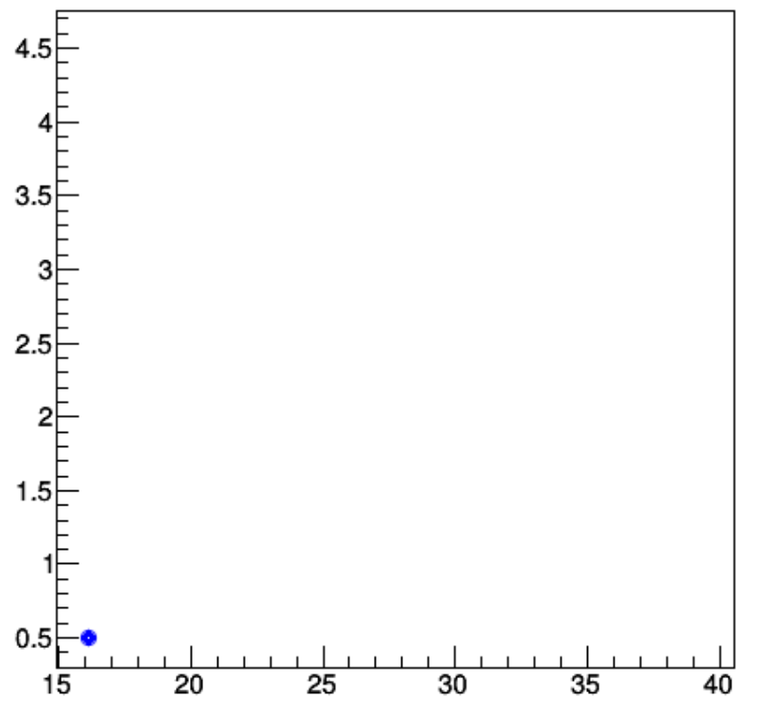
- Choice of α granularity determines resolution

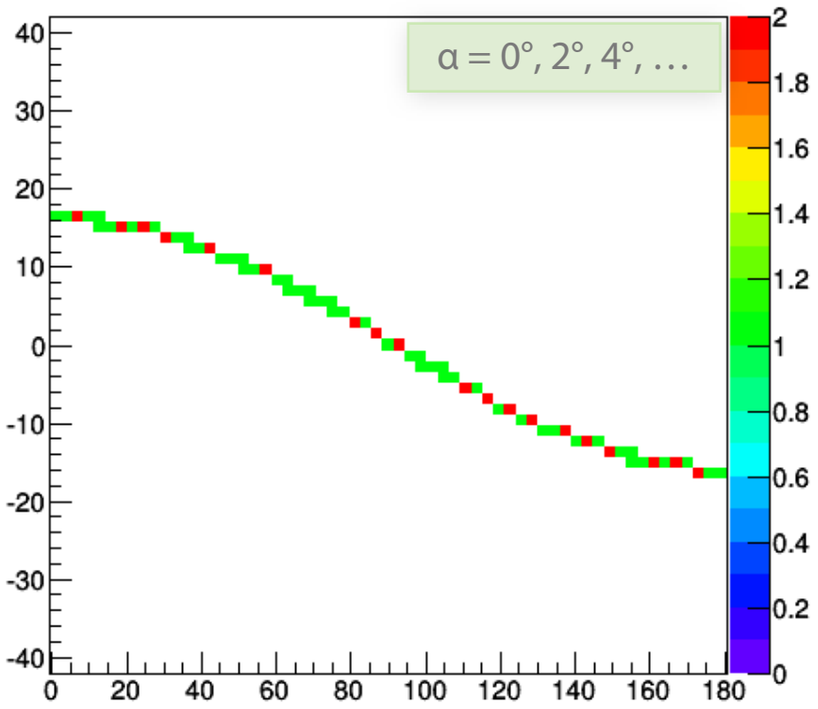




i: ~100 hits/event (STT)
j: every 0.2°

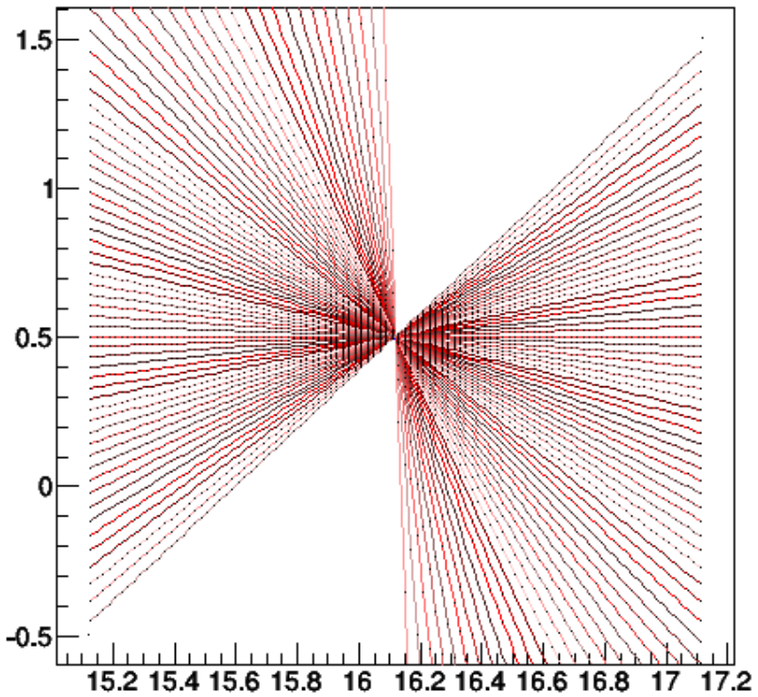
r_{ij}: 180 000

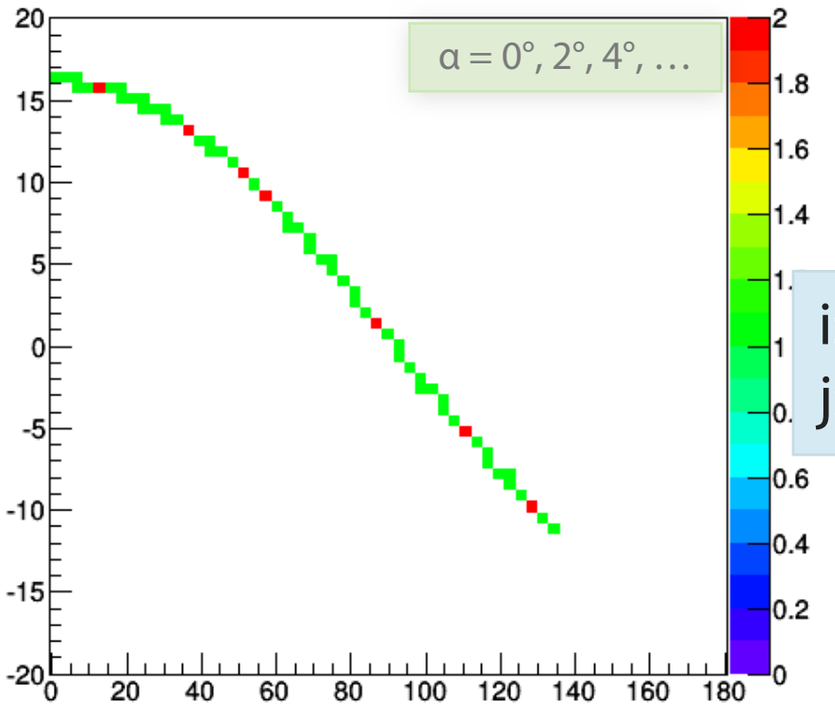




Hough Transform — Granularity

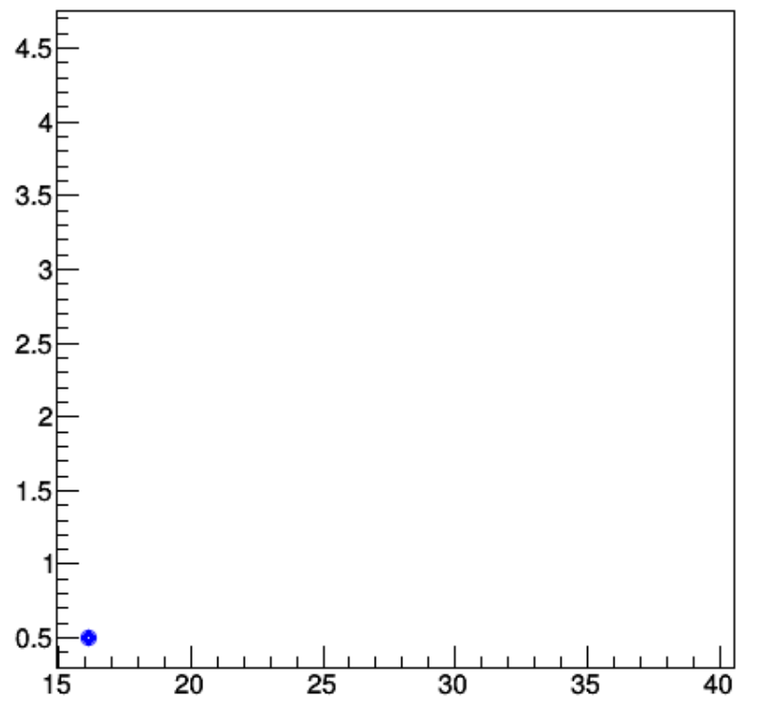
- Choice of α granularity determines resolution

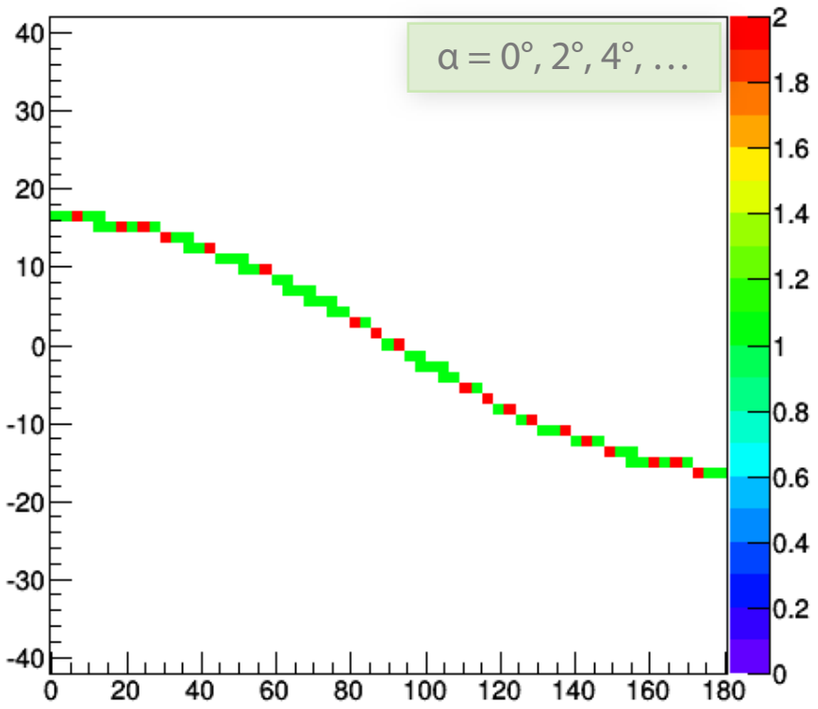




i: ~100 hits/event (STT)
j: every 0.2°

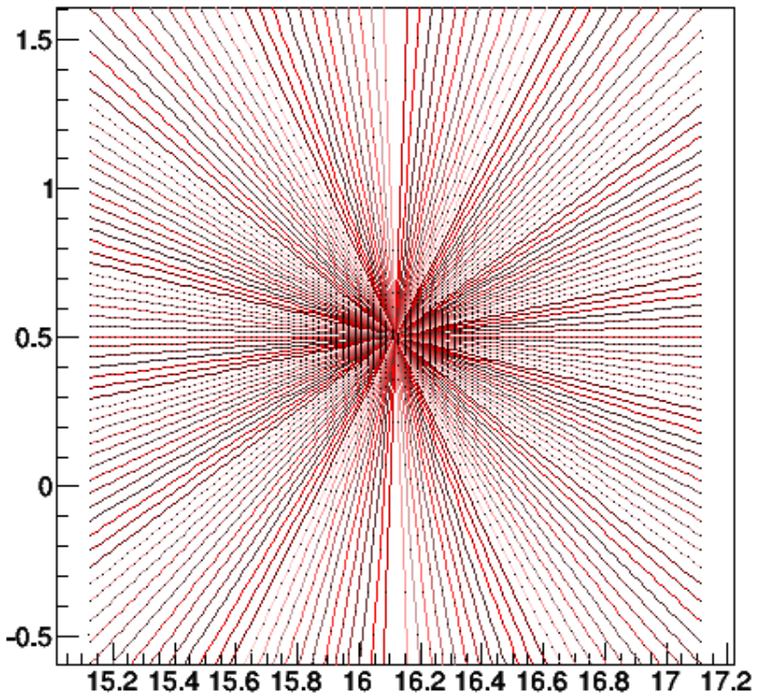
r_{ij}: 180 000

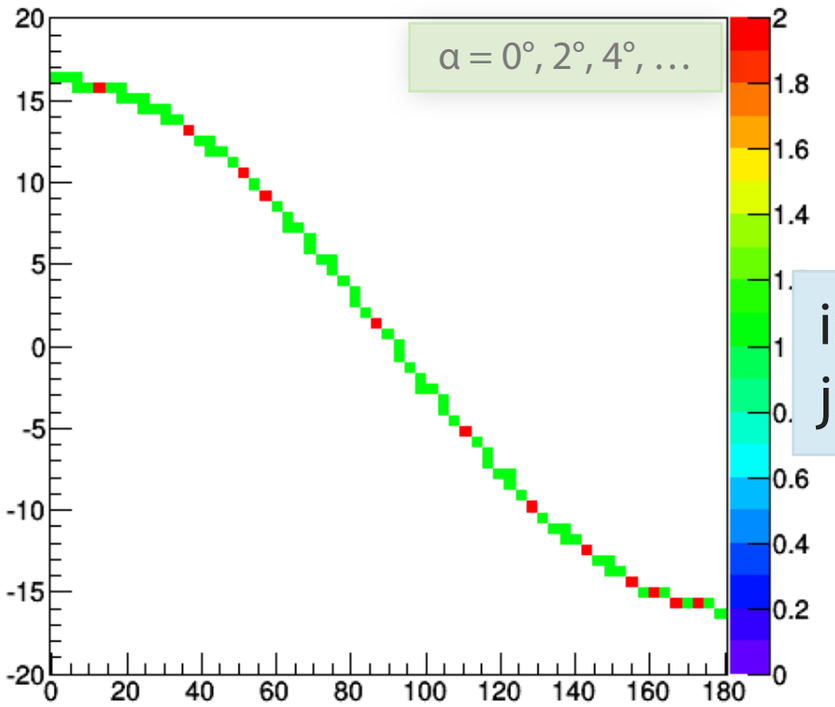




Hough Transform — Granularity

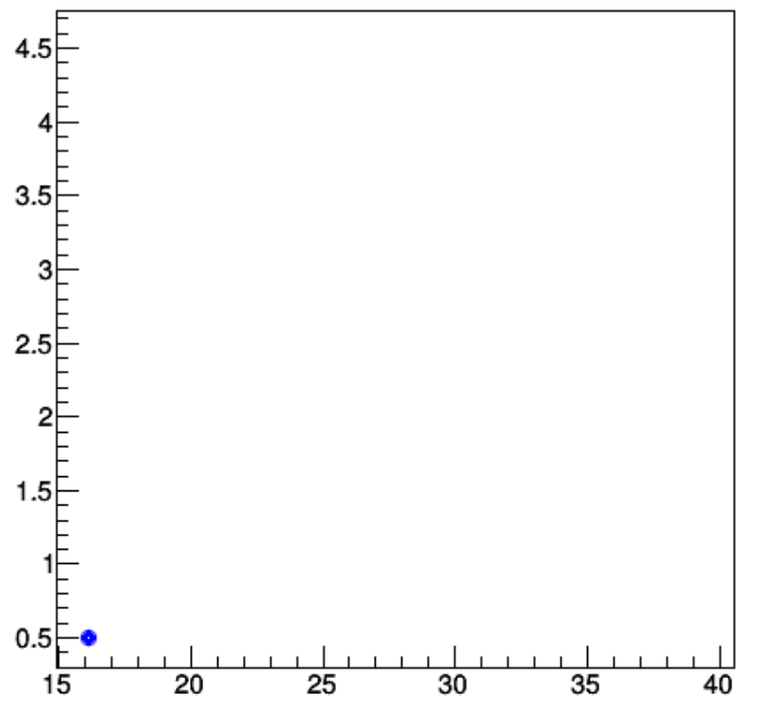
- Choice of α granularity determines resolution

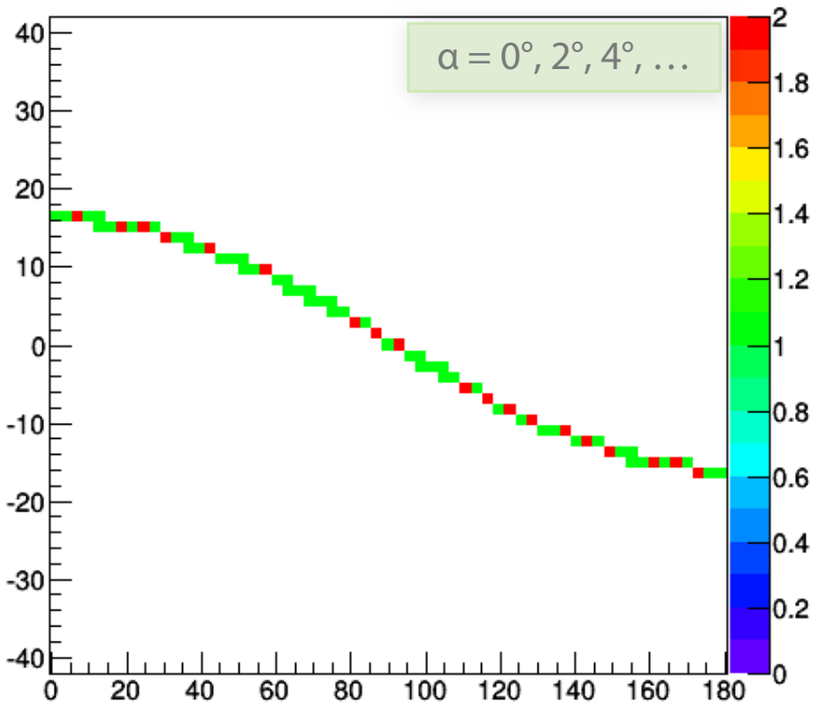




i: ~100 hits/event (STT)
j: every 0.2°

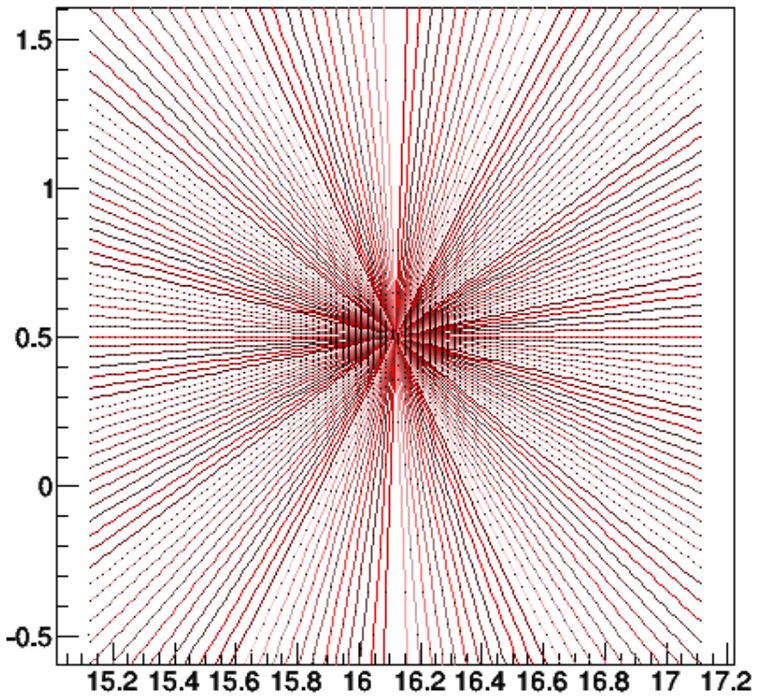
r_{ij}: 180 000

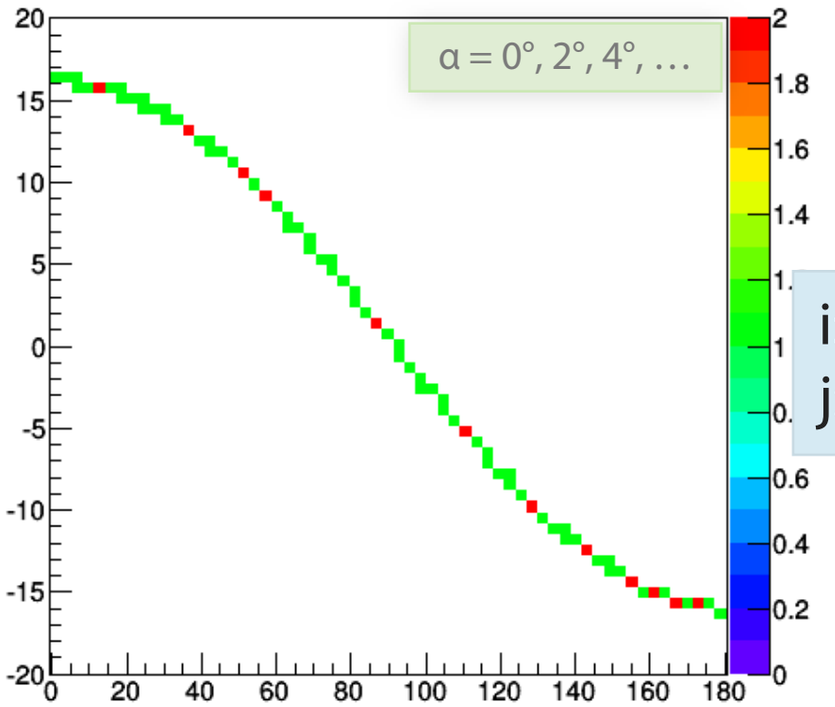




Hough Transform — Granularity

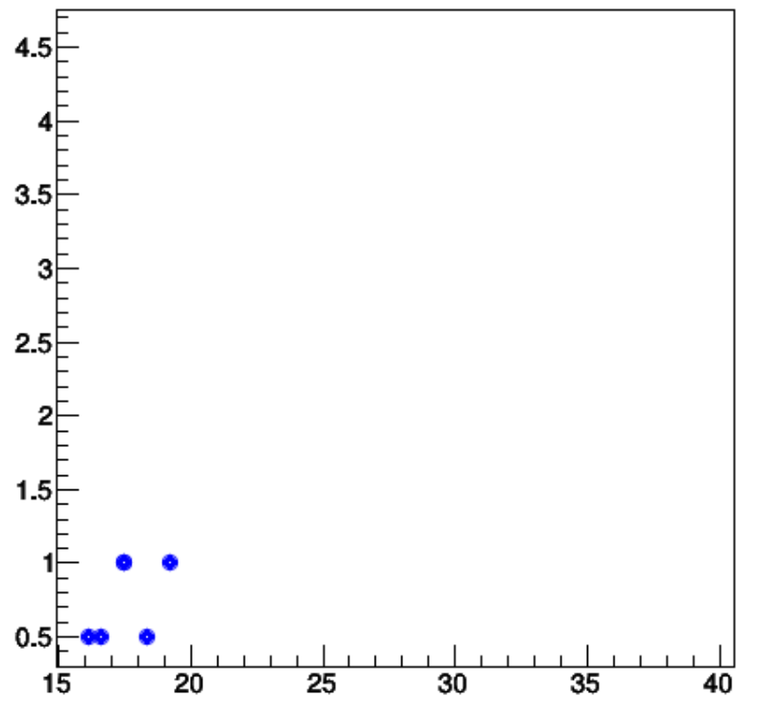
- Choice of α granularity determines resolution

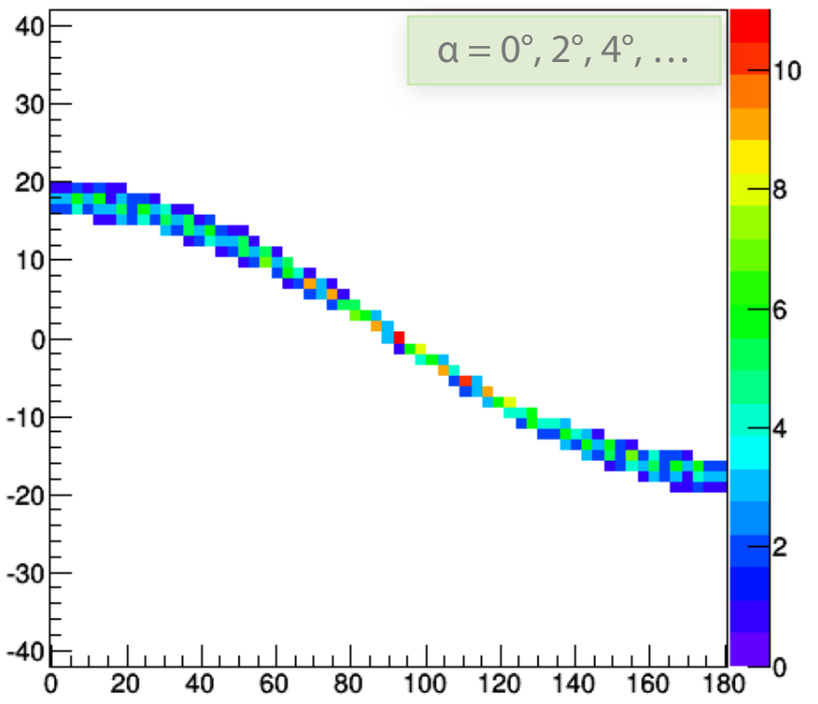




i: ~100 hits/event (STT)
j: every 0.2°

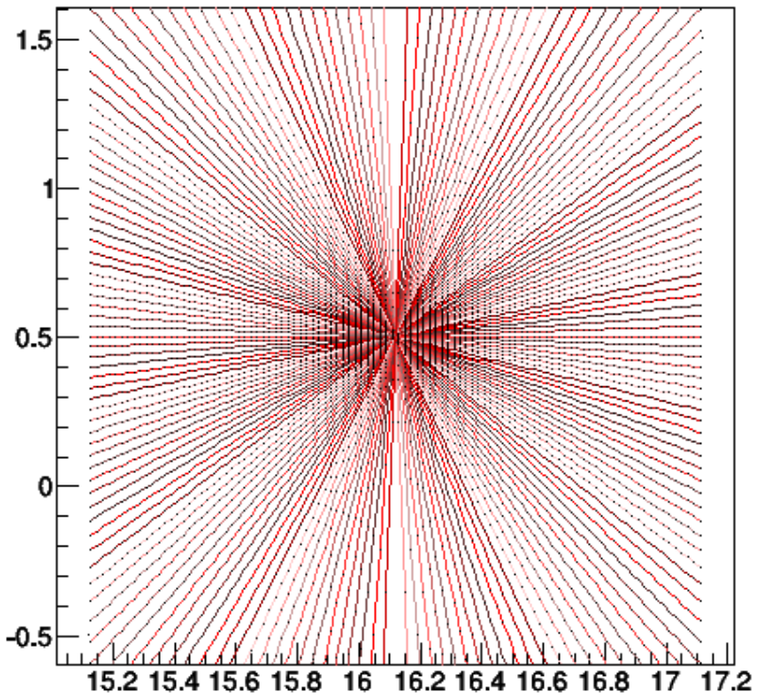
r_{ij}: 180 000

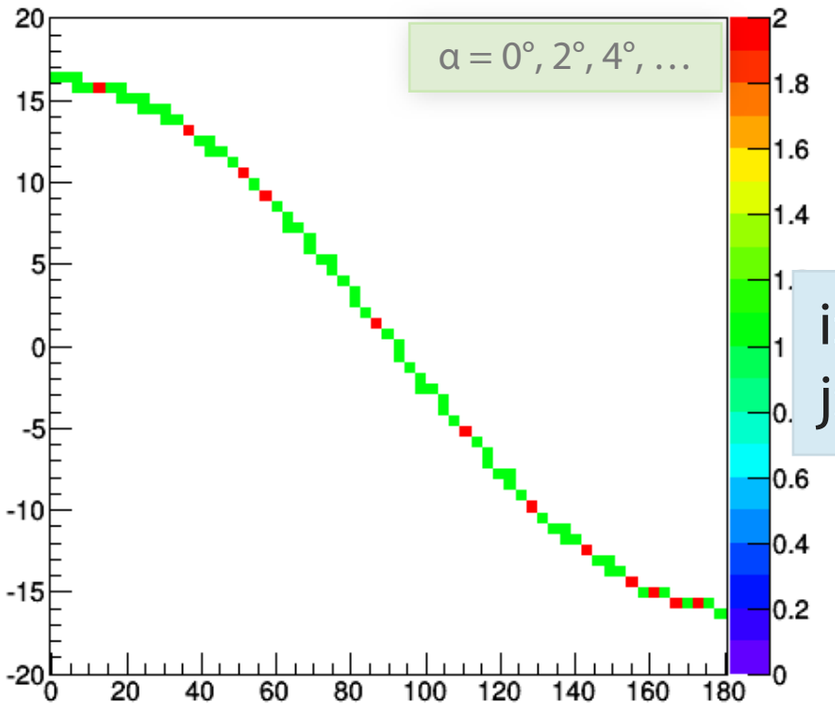




Hough Transform — Granularity

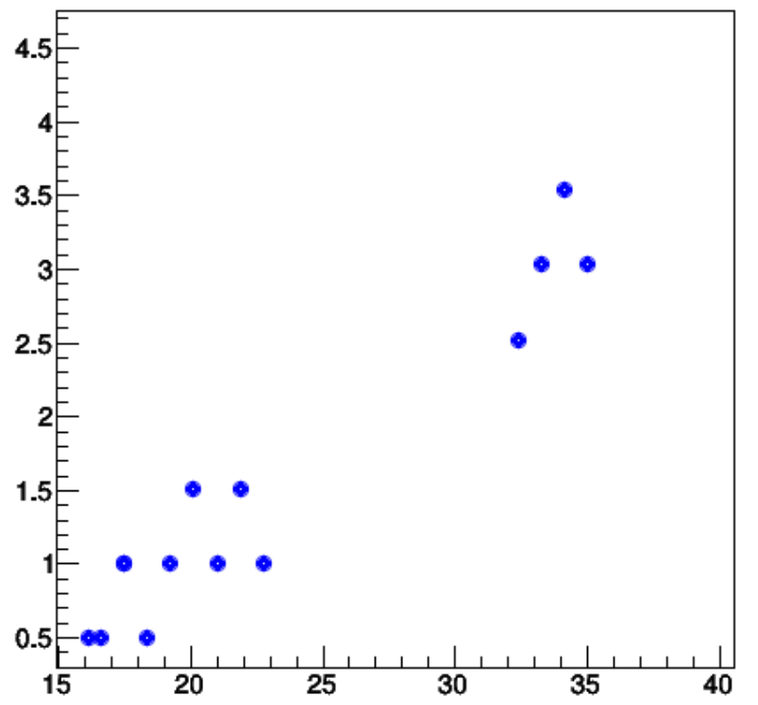
- Choice of α granularity determines resolution

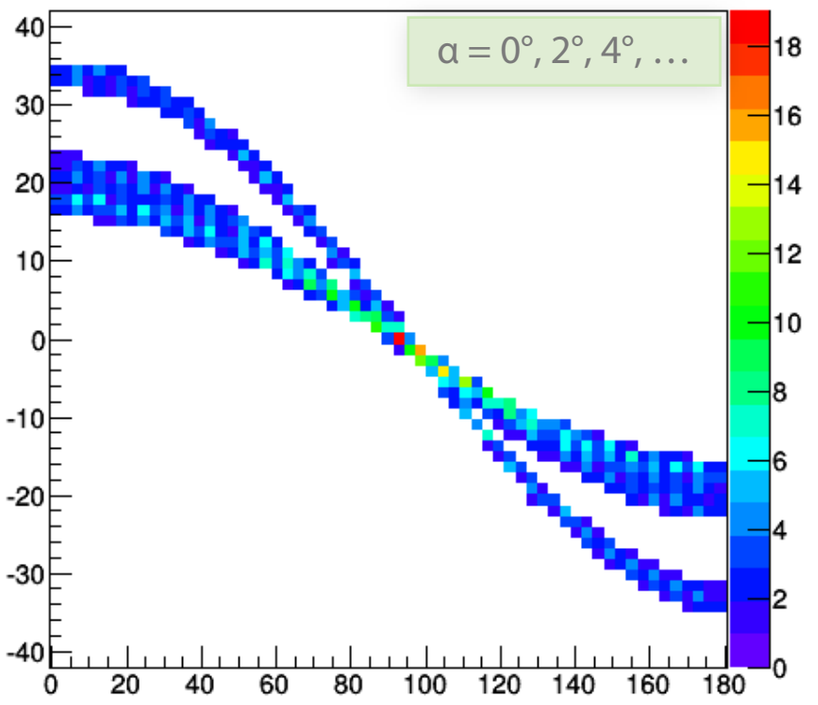




i: ~100 hits/event (STT)
j: every 0.2°

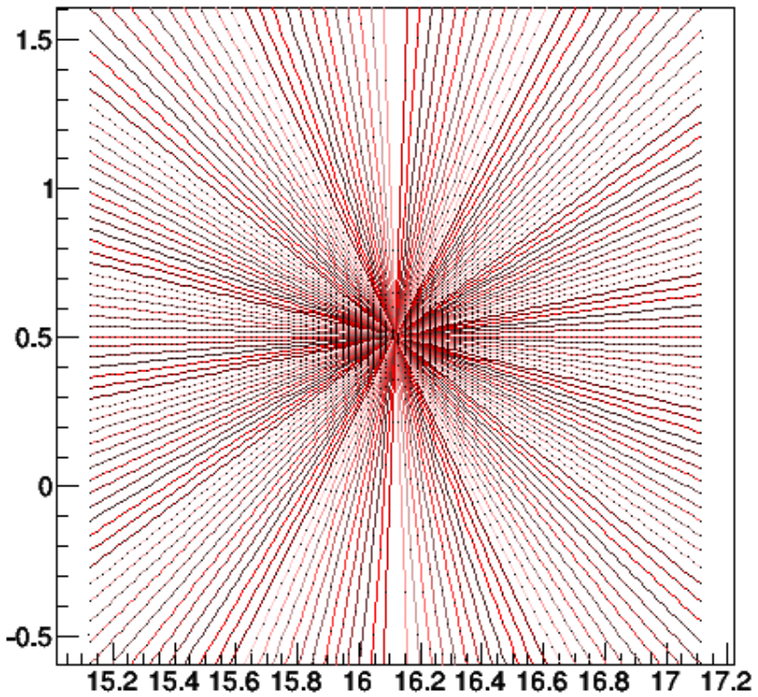
r_{ij}: 180 000

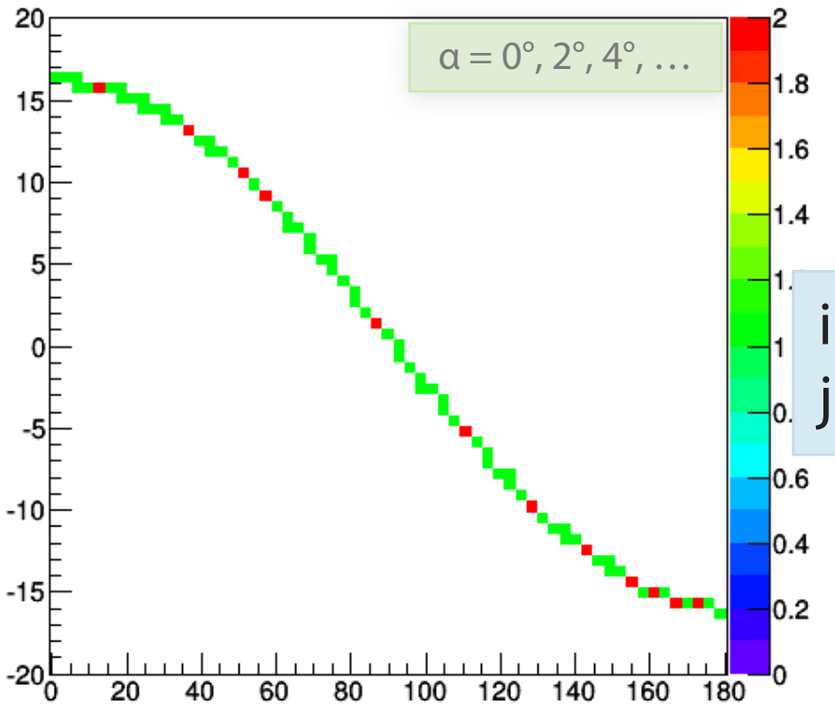




Hough Transform — Granularity

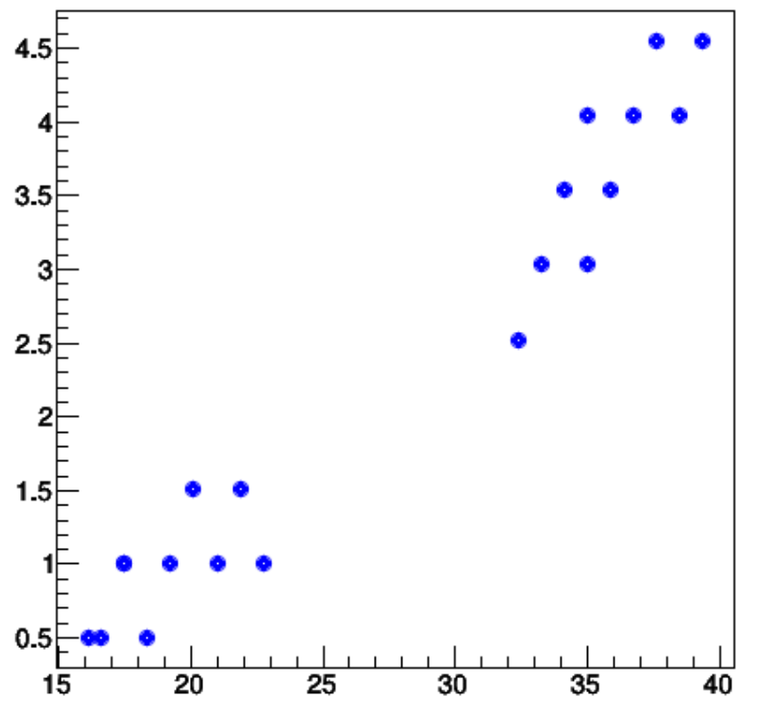
- Choice of α granularity determines resolution

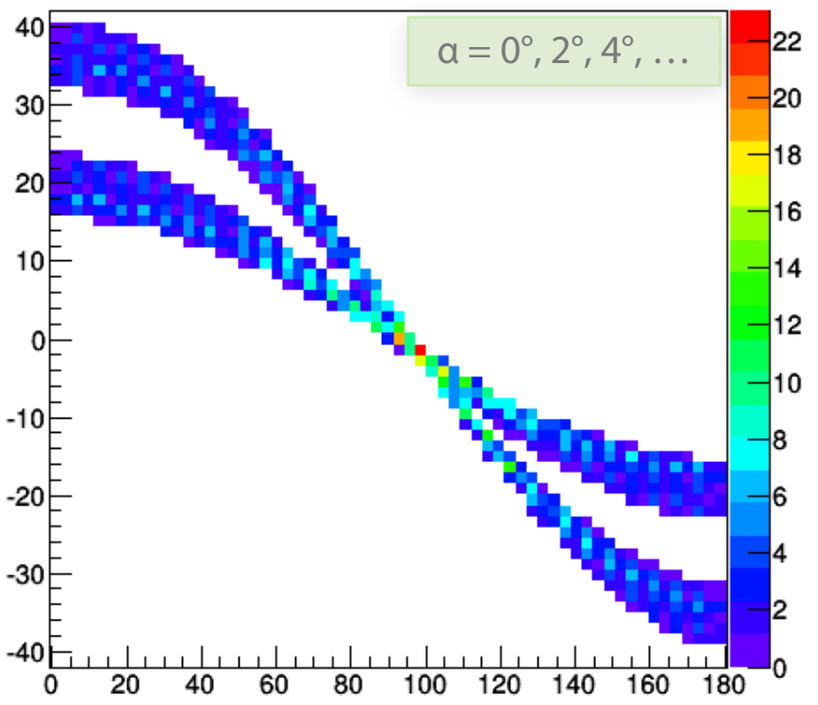




i: ~100 hits/event (STT)
j: every 0.2°

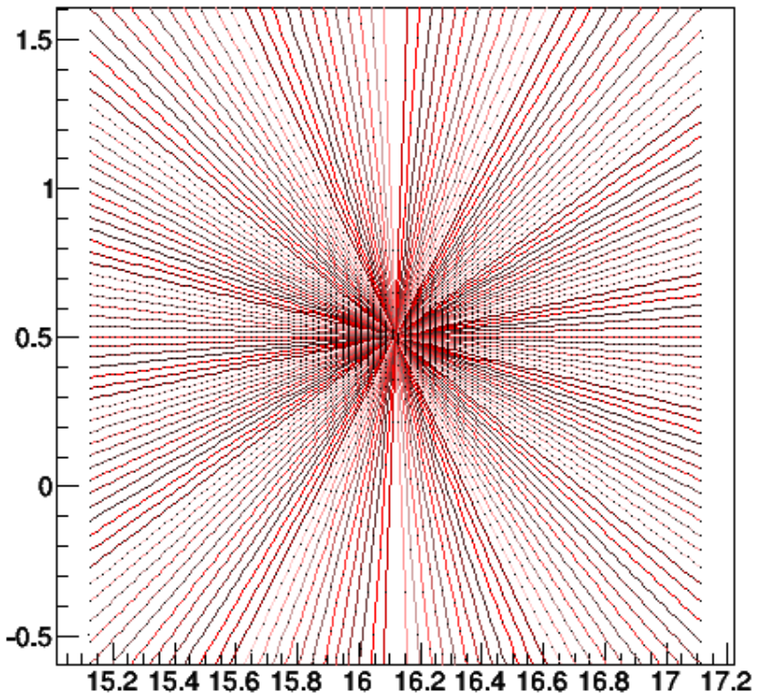
r_{ij}: 180 000

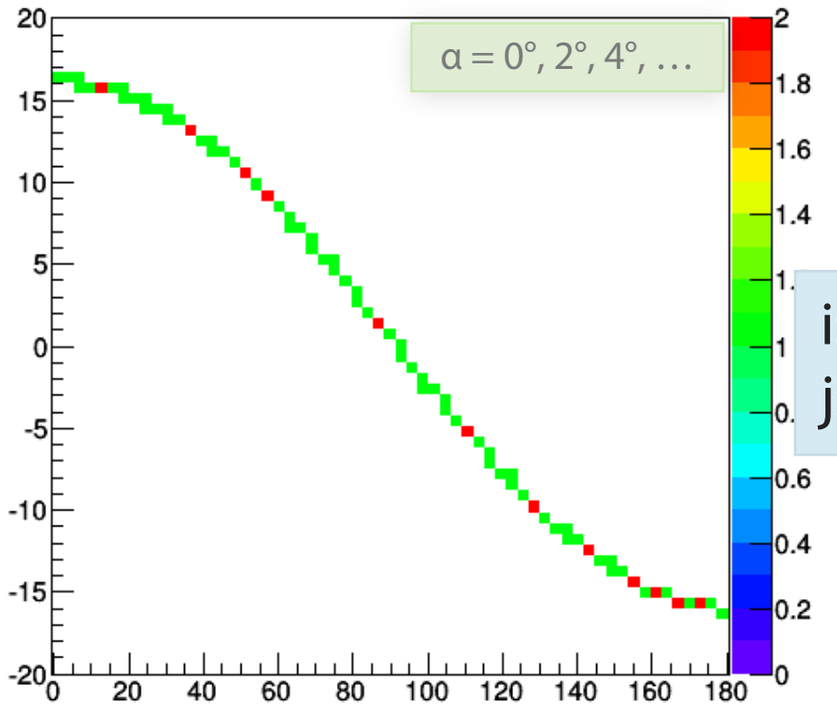




Hough Transform — Granularity

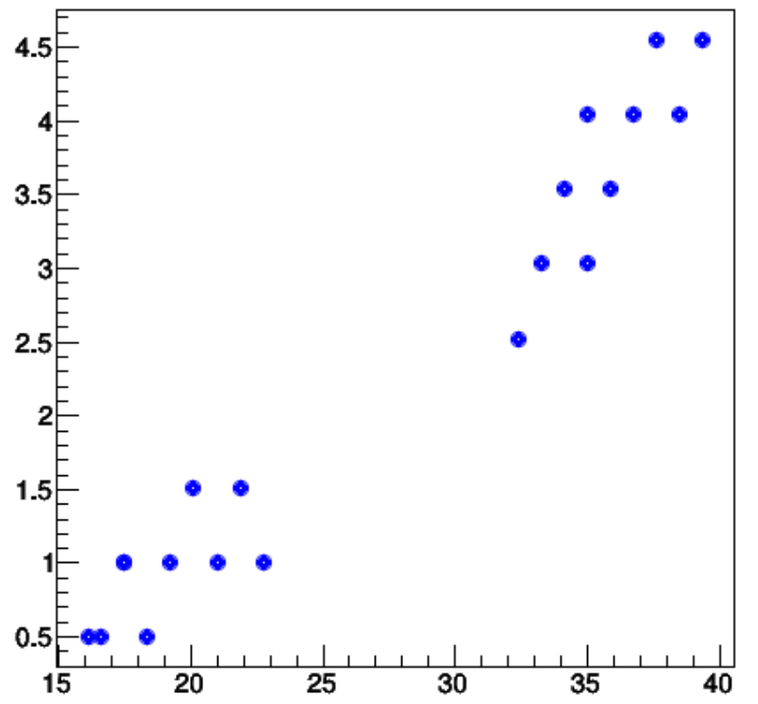
- Choice of α granularity determines resolution

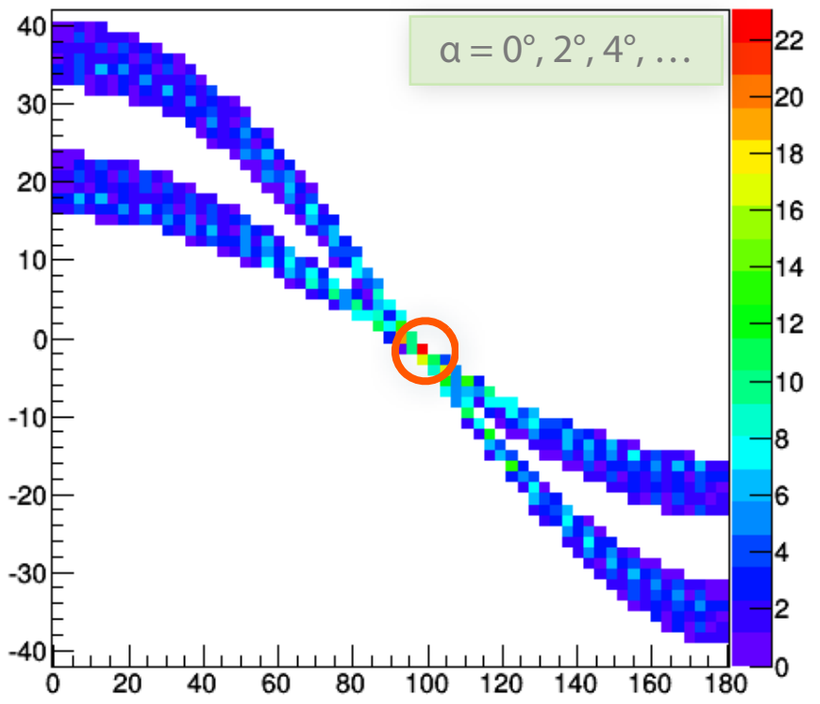




i: ~100 hits/event (STT)
j: every 0.2°

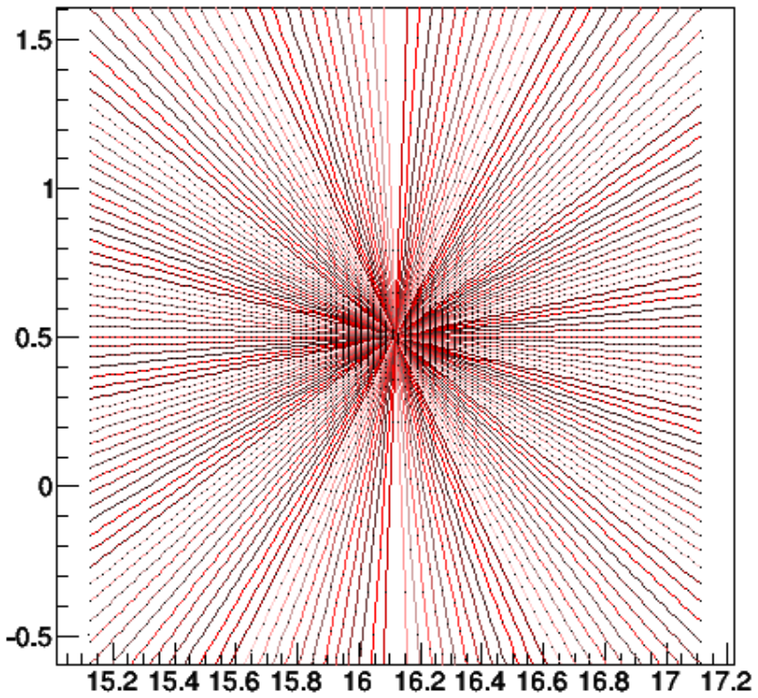
r_{ij}: 180 000

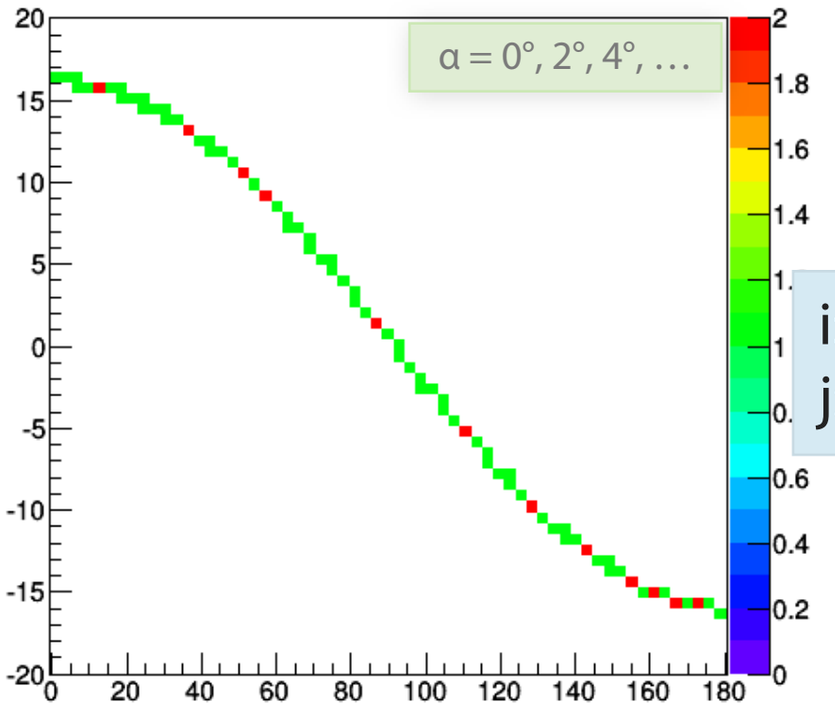




Hough Transform — Granularity

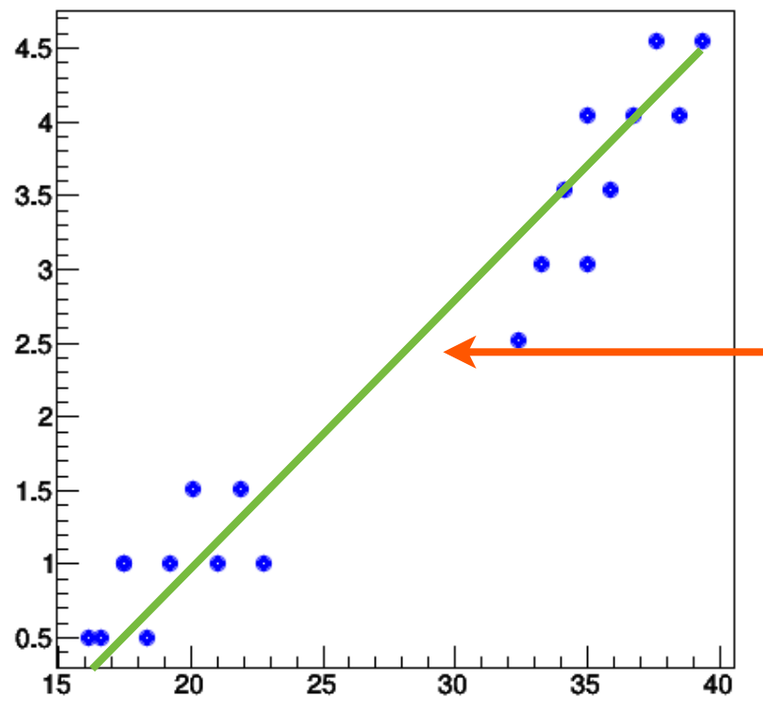
- Choice of α granularity determines resolution

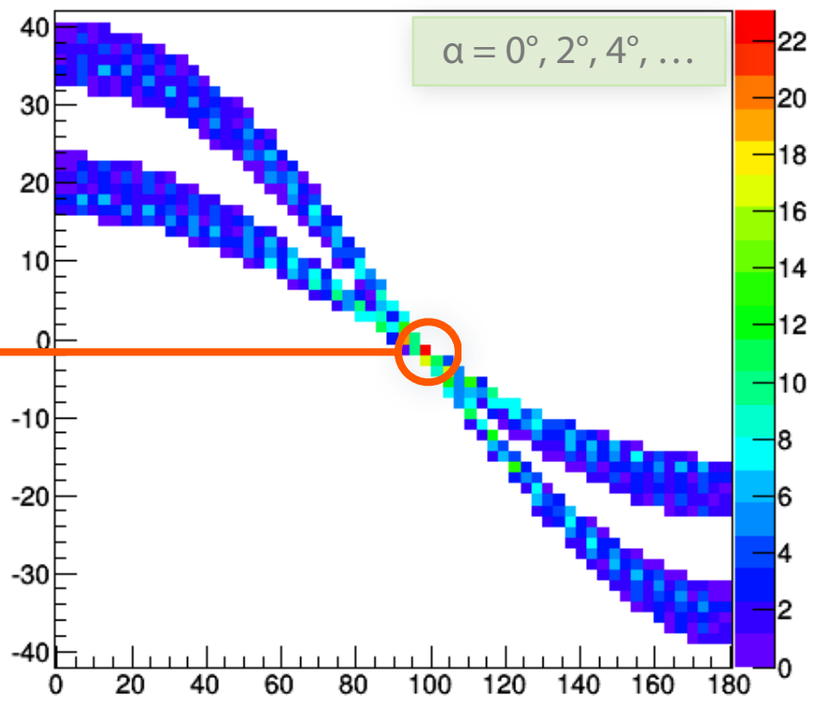




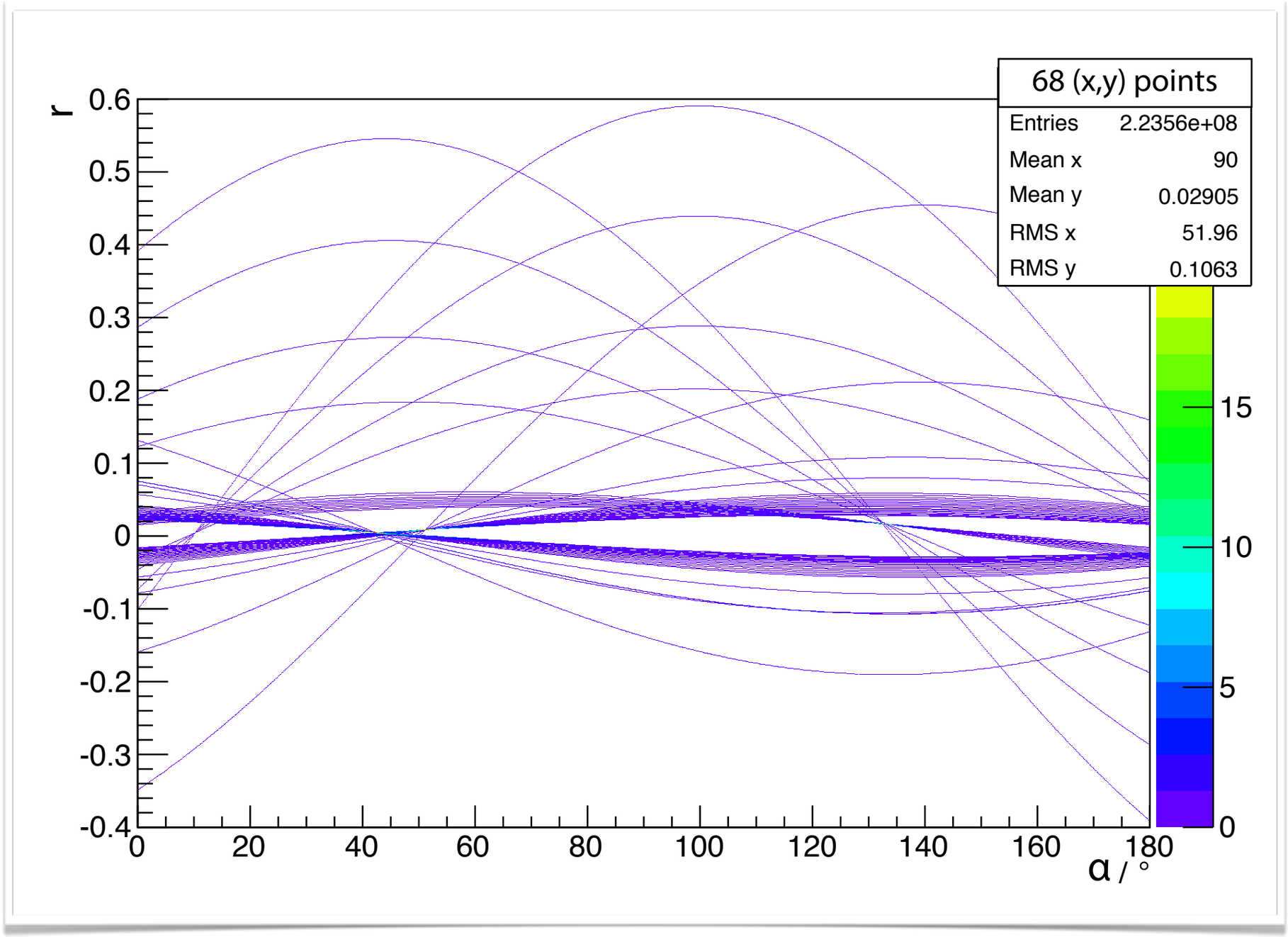
i: ~100 hits/event (STT)
j: every 0.2°

r_{ij}: 180 000





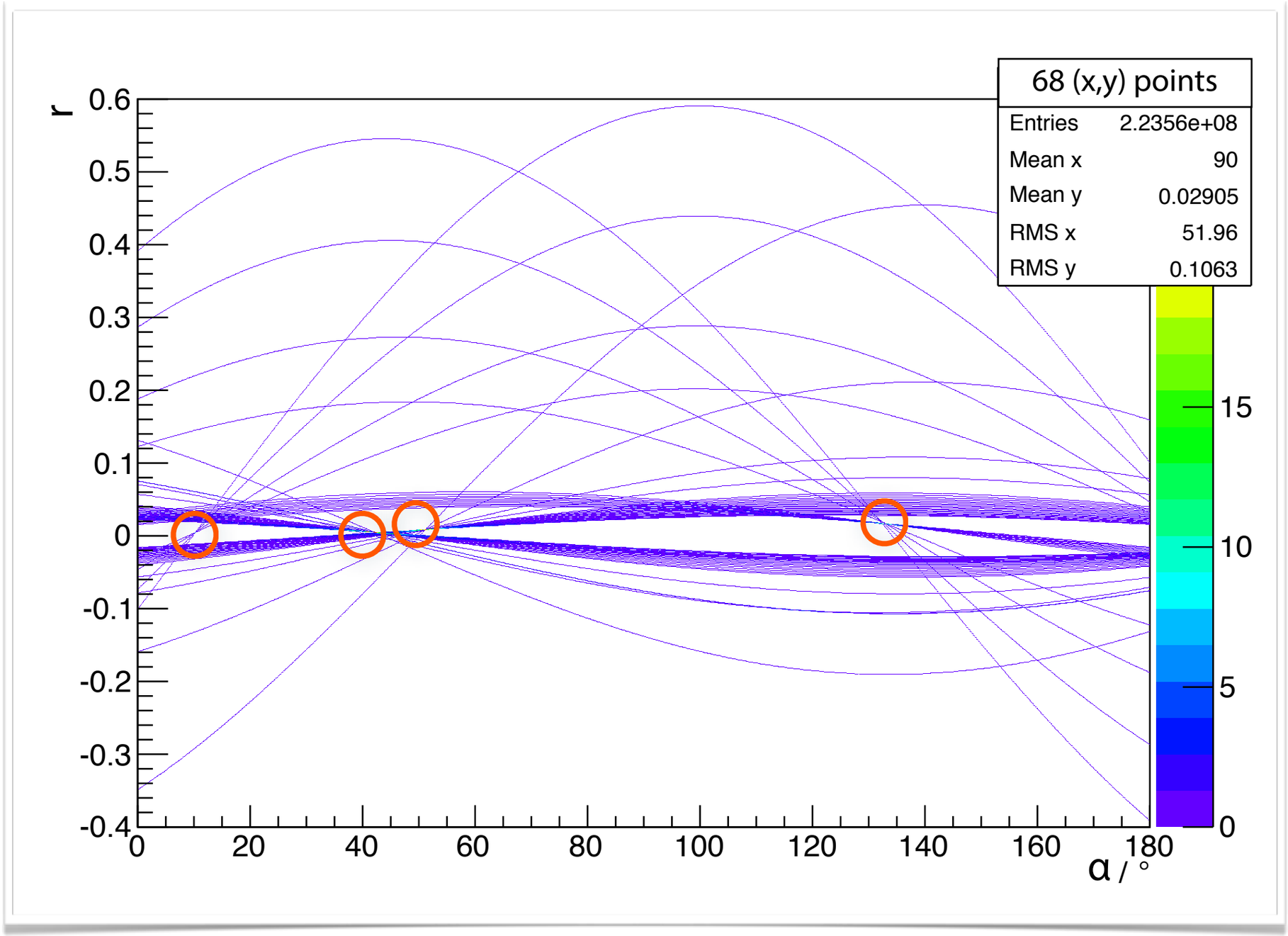
Algorithm: Hough Transform



PANDA STT+MVD

1800 x 1800 Grid

Algorithm: Hough Transform



PANDA STT+MVD

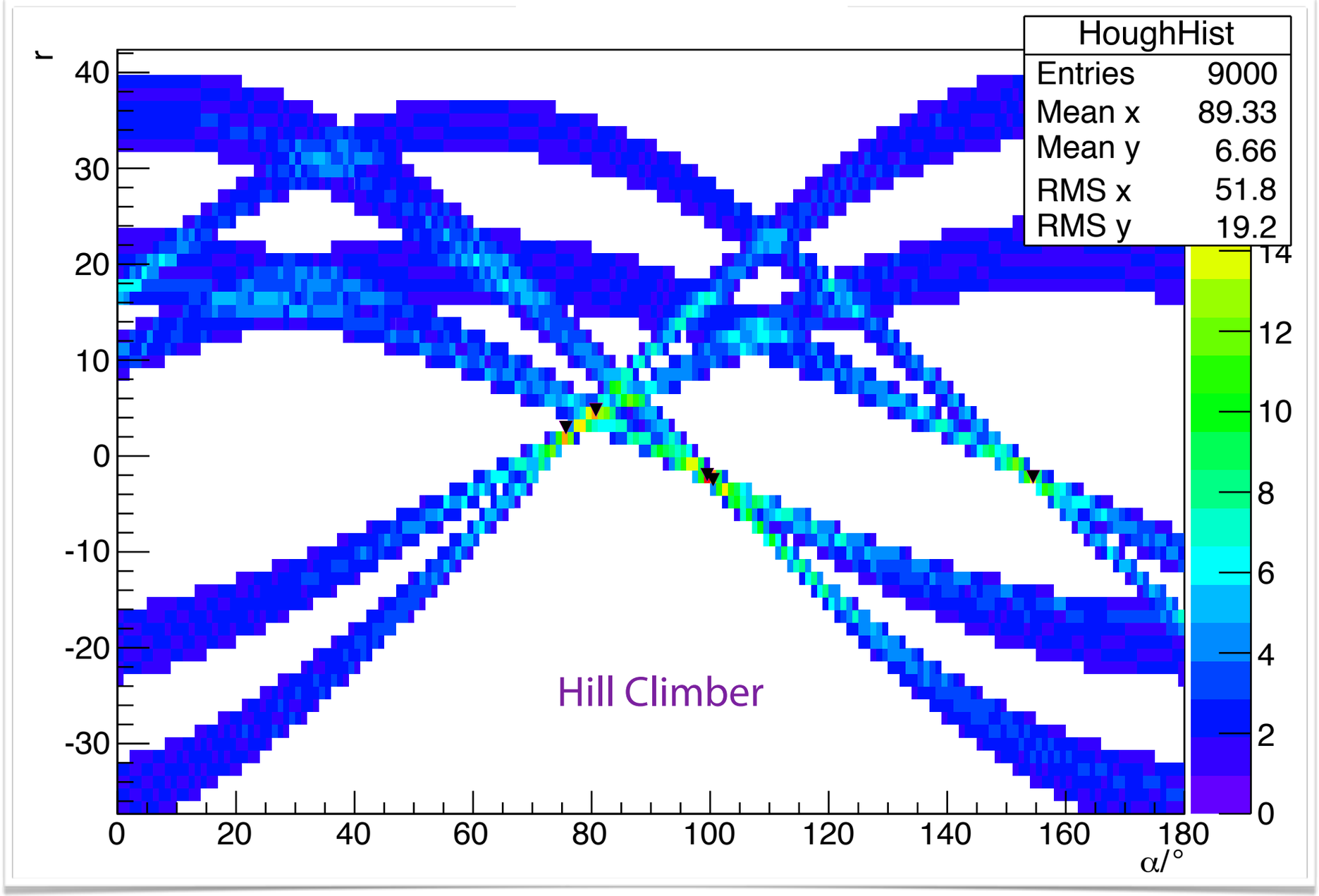
1800 x 1800 Grid

Hough Transform — Remarks

Two Implementations

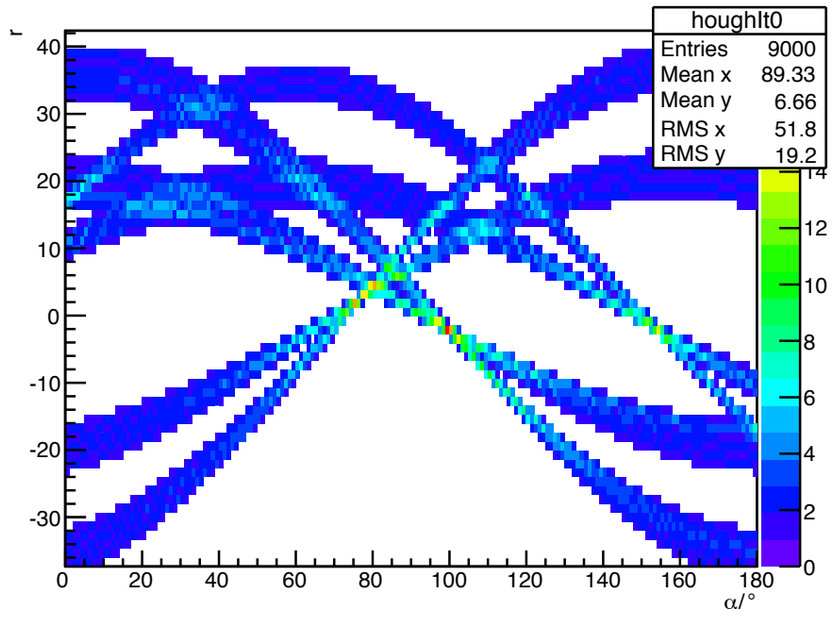
- Thrust *(CUDA's STL)*
  - **Performance**: 3 ms/evt
  - Reduce to set of **standard routines**
    - Fast (uses Thrust's optimized algorithms)
    - Inflexible (hard to customize)
    - Not yet at performance maximum
- Plain CUDA
  - **Performance**: 0.5 ms/evt
  - Build completely for this task
    - Fitting for PANDA; customizable
    - A bit more complicated at parts
- **Peakfinding challenging**

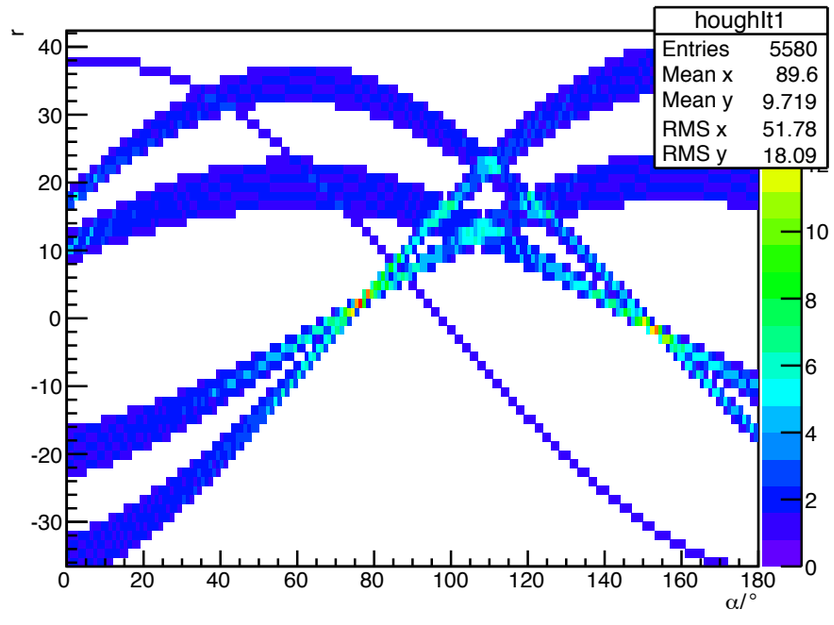
Hough Transform — Remarks



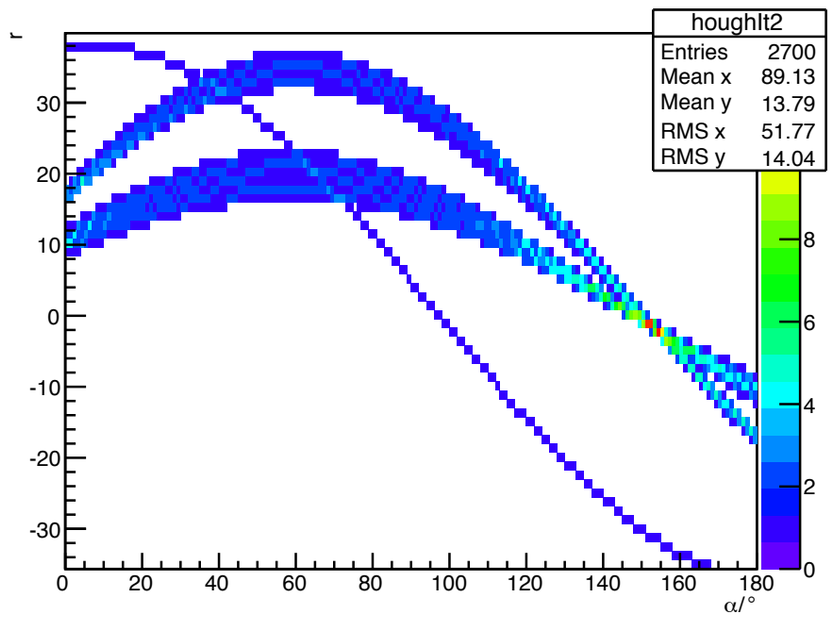
Peakfinding challenging

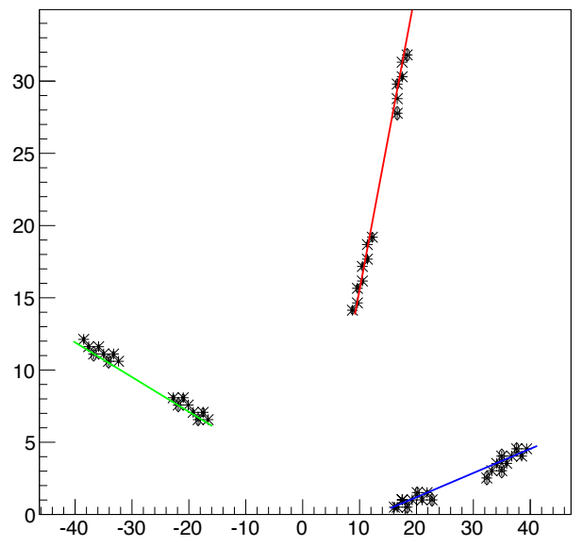
Hough Transform — Remarks





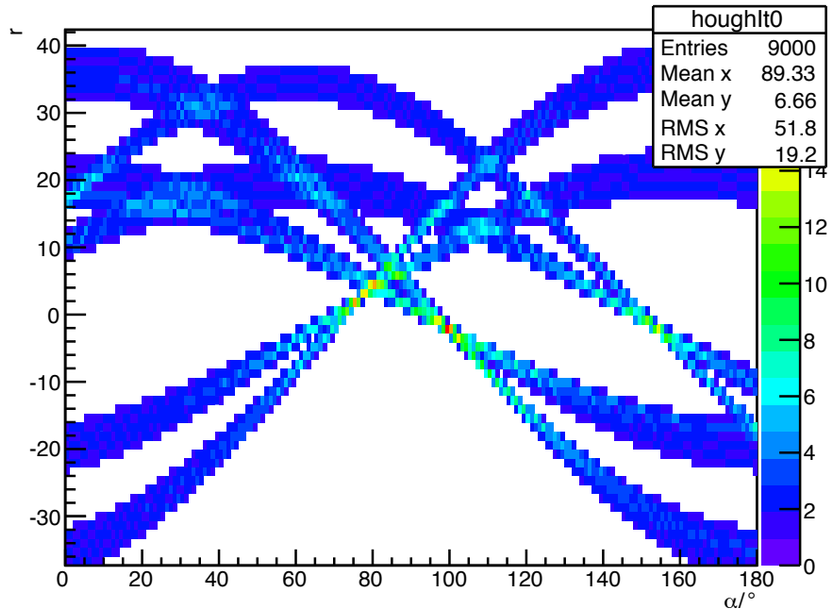
Iterative Maximum Deleter

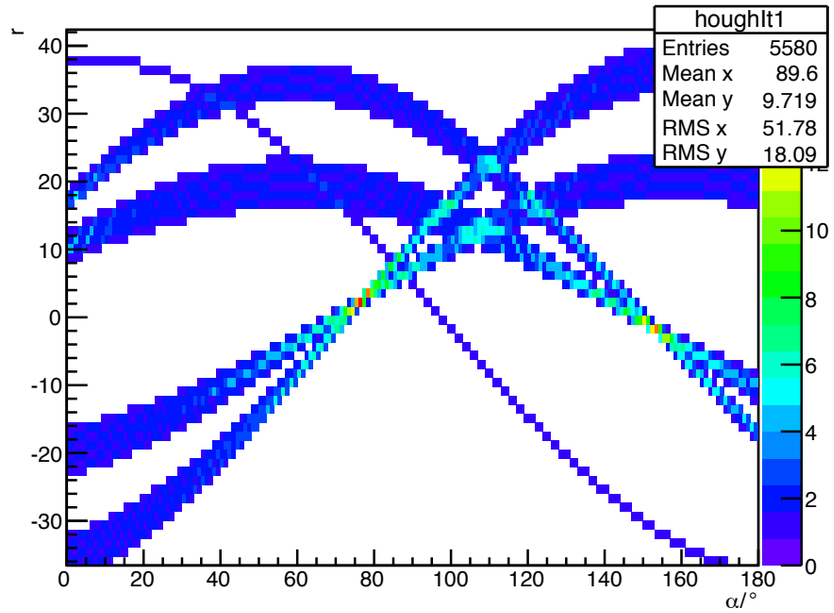




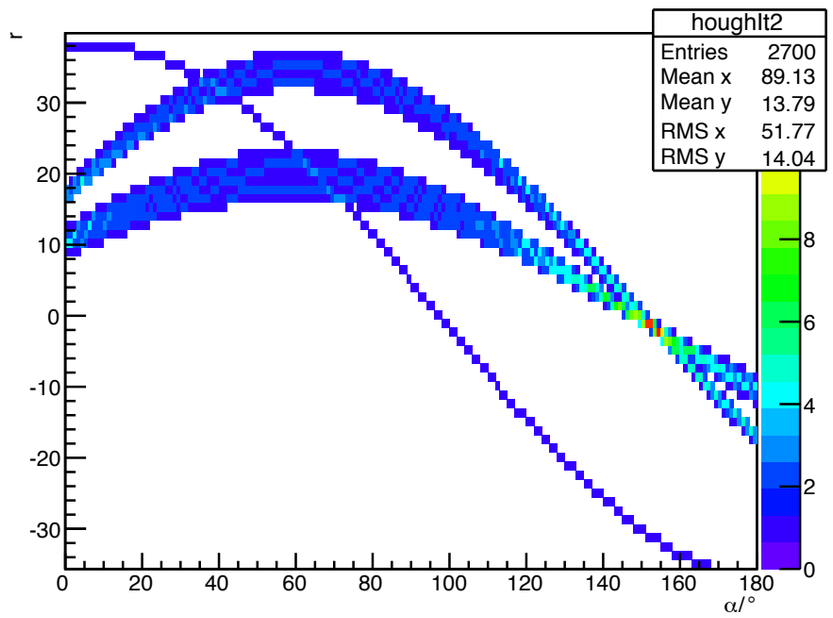
Peakfinding challenging

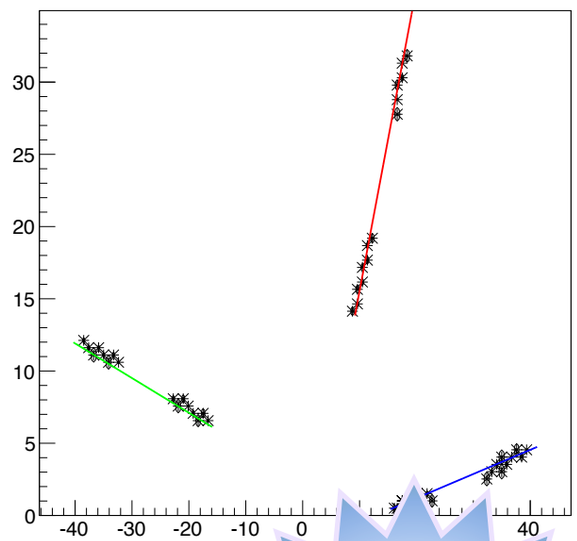
Hough Transform — Remarks





Iterative Maximum Deleter





Peakfinding challenging

current research

JÜLICH
FORSCHUNGSZENTRUM

panda

ALGORITHMS #2

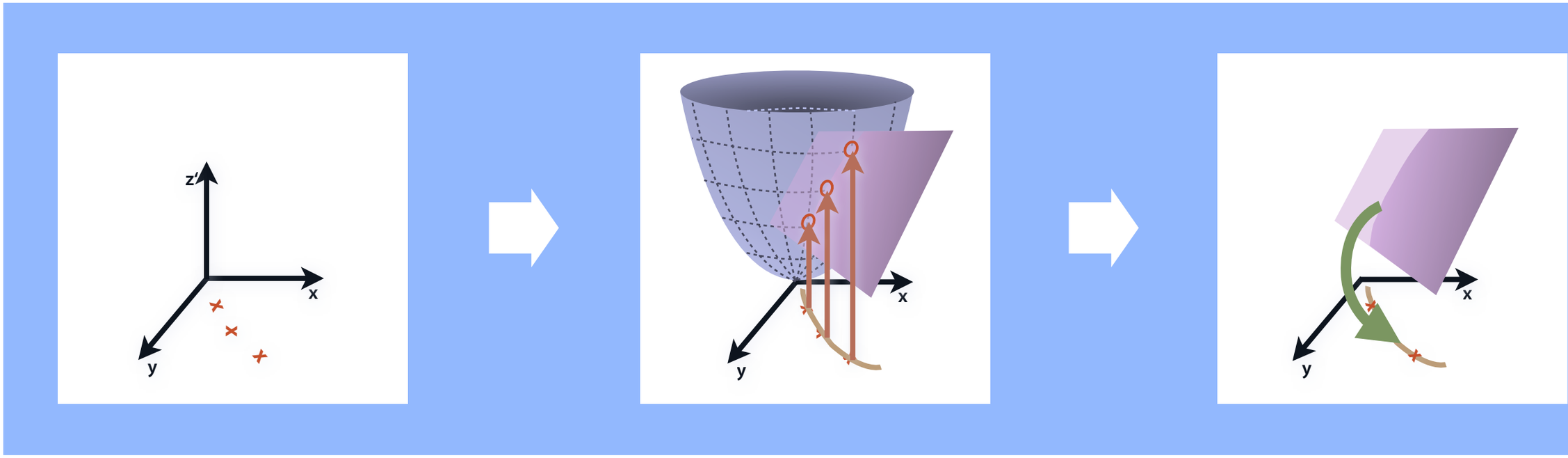
Hough Transform

Riemann Track Finder

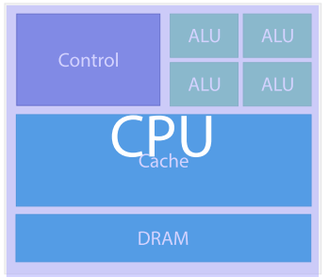
Triplet Finder

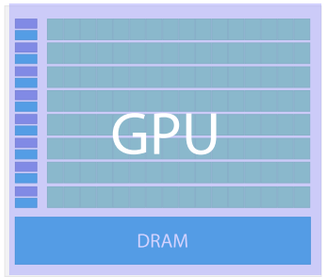
Riemann Track Finder — Method

- ***Idea:*** Don‘t fit lines *(in 2D)*, fit planes *(in 3D)*!
- **Create seeds** 1
  - All possible three hit combinations
- **Grow** seeds to **tracks** 2
  Continuously test next hit if it fits
  - Use mapping to Riemann paraboloid *(+ s-z fit, det. layer)*



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;
```

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

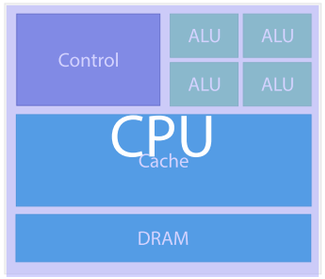
$$pos(n_{Layer_x}) = \frac{\sqrt[3]{\sqrt{3}\sqrt{243x^2-1}+27x}}{3^{2/3}} + \frac{1}{\sqrt[3]{3}\sqrt[3]{\sqrt{3}\sqrt{243x^2-1}+27x}} - 1$$

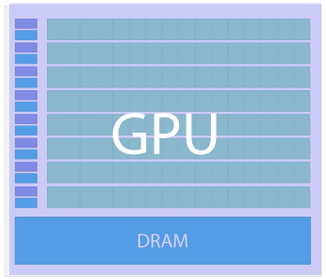
2

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

Port of CPU code; parallelism on seed base

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/event

ALGORITHMS #3

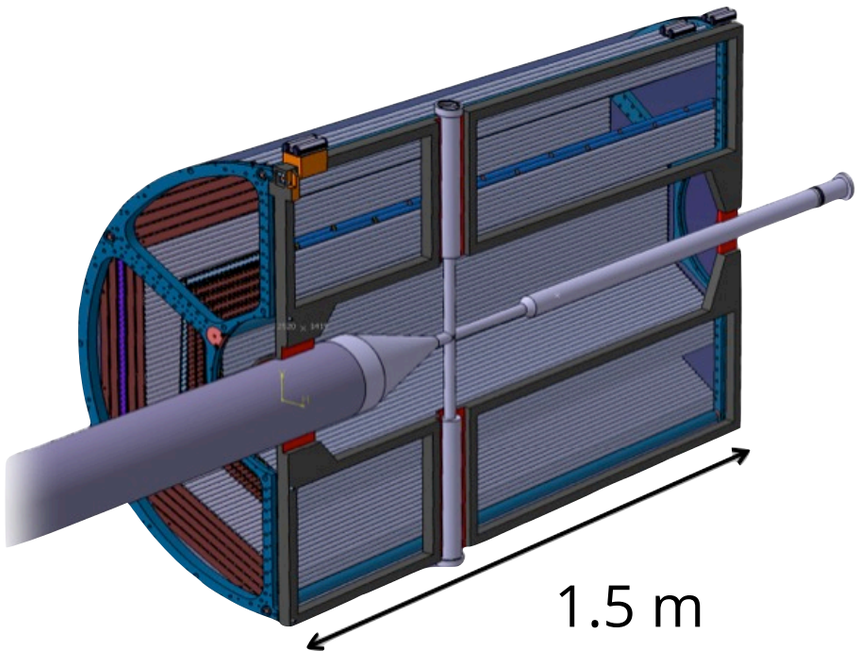
Hough Transform

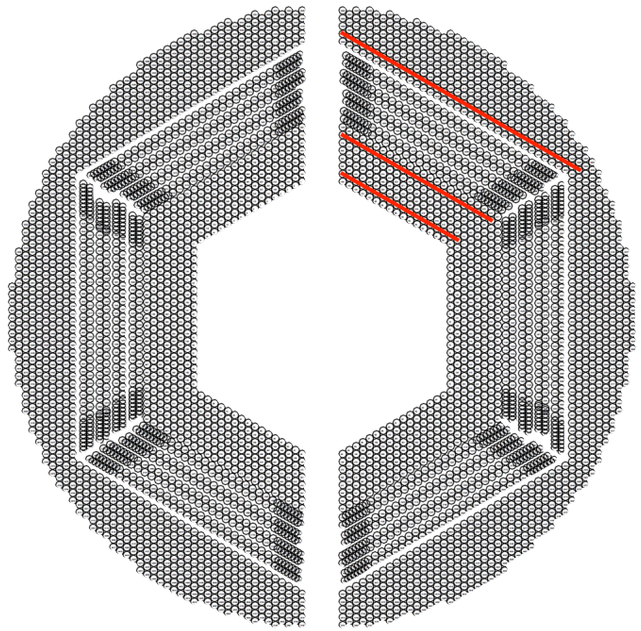
Riemann Track Finder

Triplet Finder

Triplet Finder

- Algorithm specifically designed for the P̄ANDA Straw Tube Tracker *(STT)*





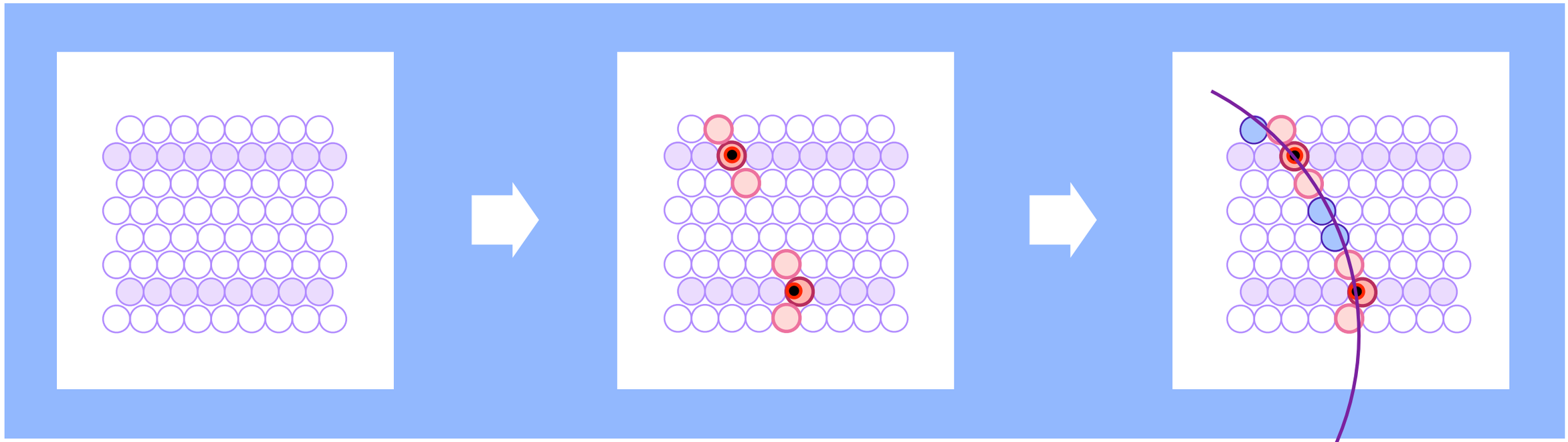
Original algorithm by
Marius Mertens *et al*

- Ported to GPU by **Andrew Adinetz**
  - CUDA, Dynamic Parallelism, Thrust
  - Quality of tracks comparable to CPU

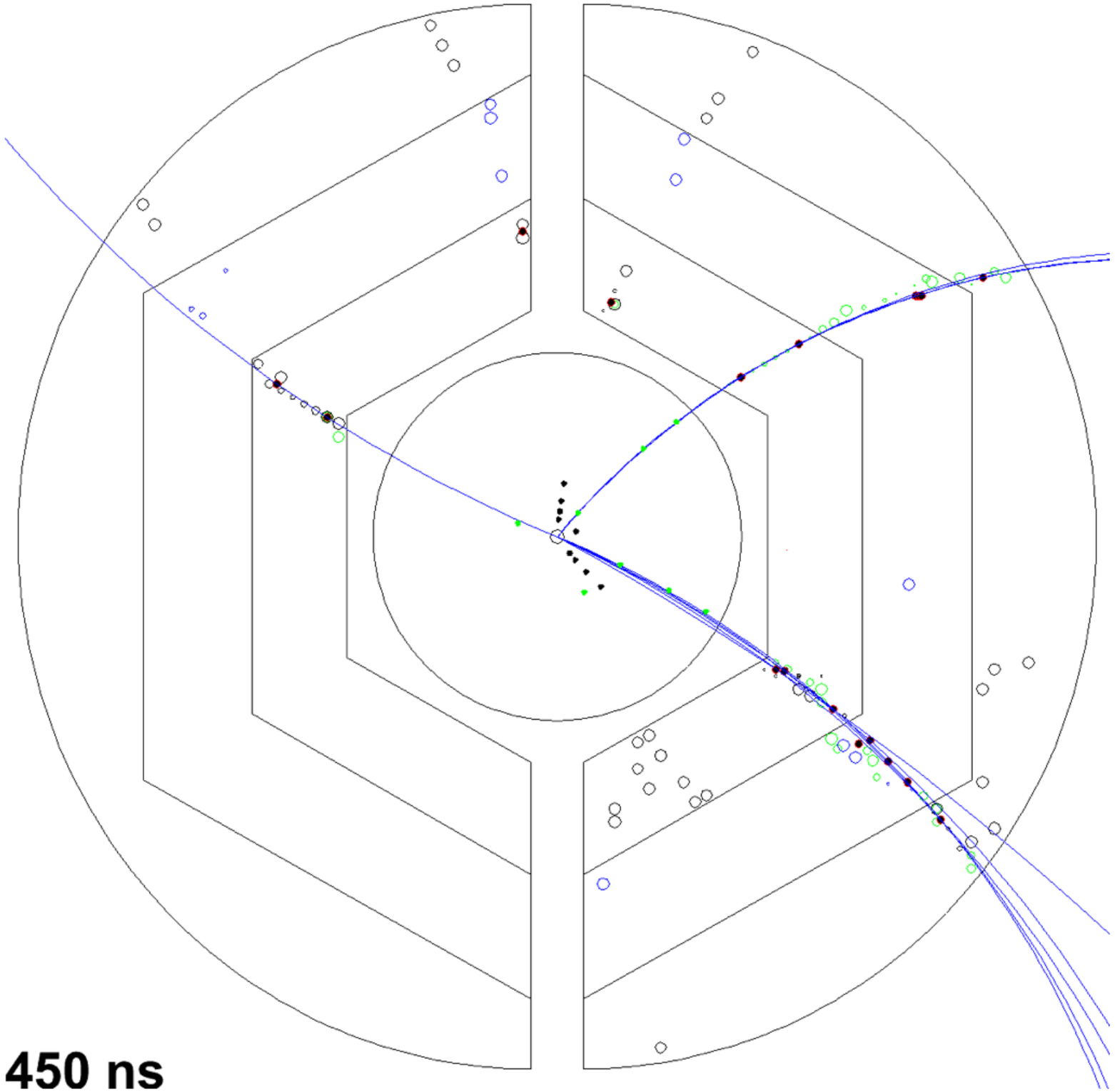
JÜLICH APPLICATION LAB
NVIDIA
http://www.fz-juelich.de/ias/jsc/

Triplet Finder

- ***Idea:*** Use only subset of detector as seed
  - Don‘t use STT isochrones *(drift times)*
  - Calculate circle from 3 points (no fit)
- Features
  - Fast & robust algorithm, no event time needed
  - Many tuning possibilities

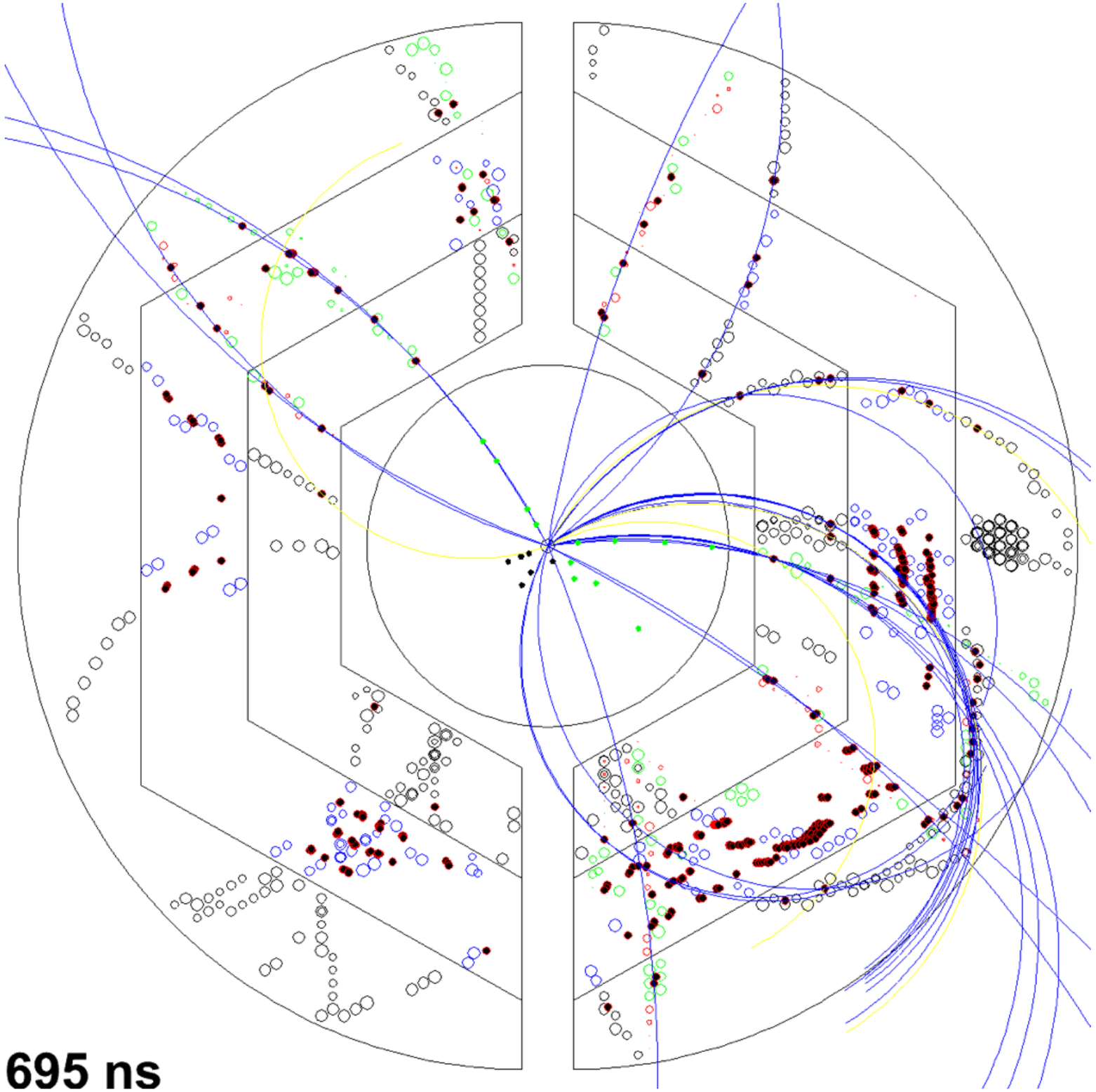


Triplet Finder — Display



- Isochrone *early*
- Isochrone *early* & *skewed*
- Isochrone *close*
- Isochrone *late*
- MVD hit
- **Triplet**
- Track *current*
- Track *timed out*

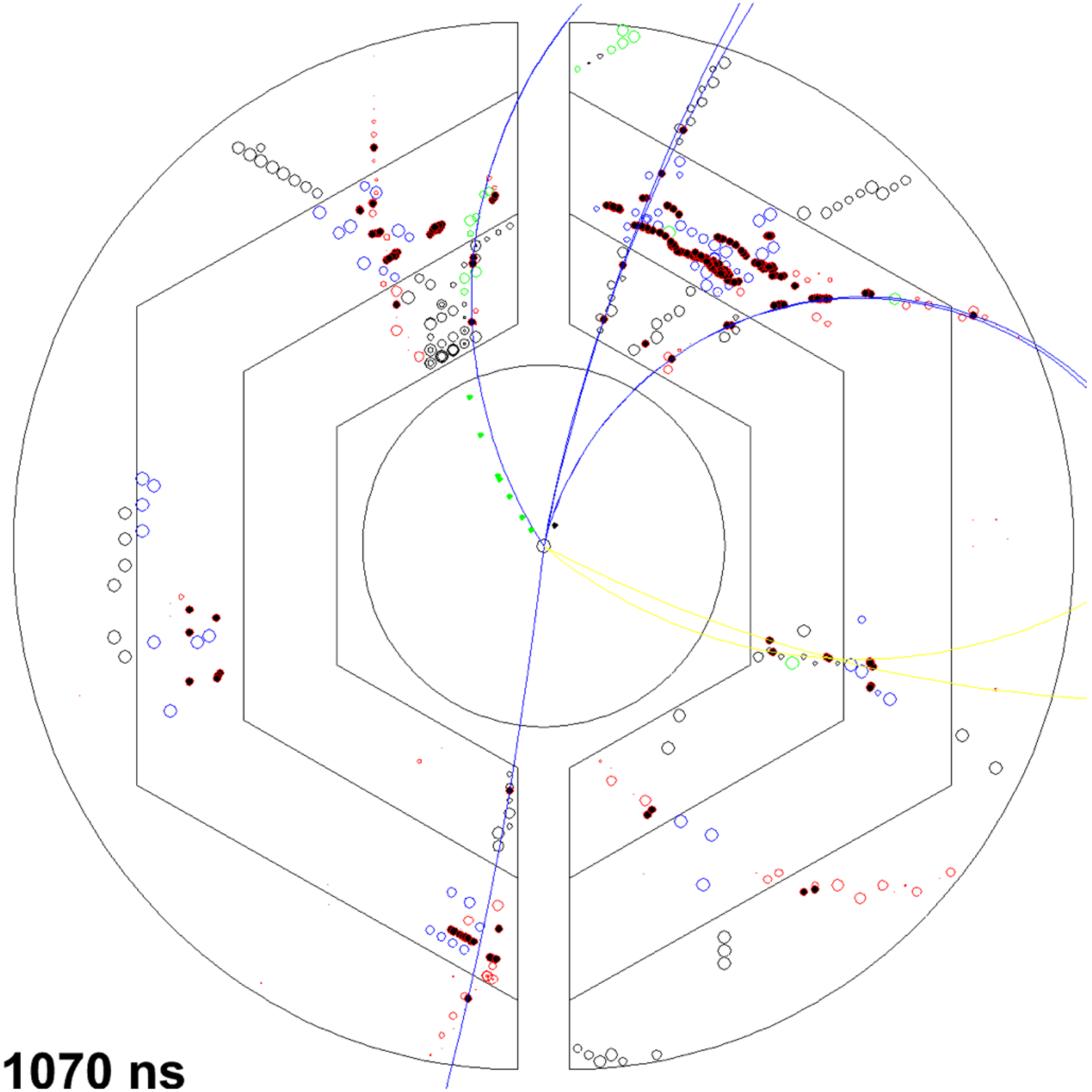
Triplet Finder — Display



695 ns

- Isochrone *early*
- Isochrone *early* & *skewed*
- Isochrone *close*
- Isochrone *late*
- MVD hit
- **Triplet**
- Track *current*
- Track *timed out*

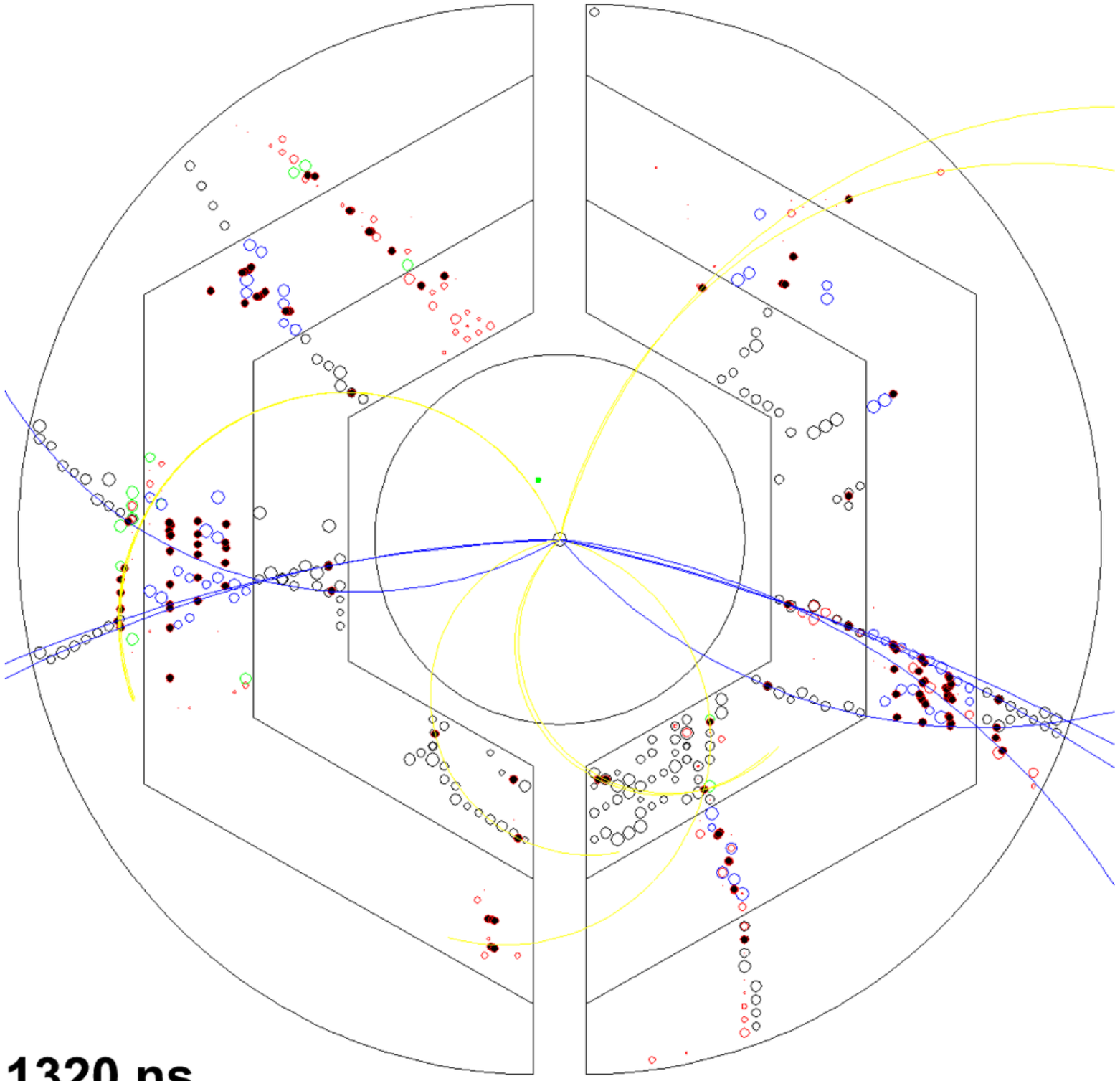
Triplet Finder — Display



1070 ns

- Isochrone *early*
- Isochrone *early* & *skewed*
- Isochrone *close*
- Isochrone *late*
- MVD hit
- **Triplet**
- Track *current*
- Track *timed out*

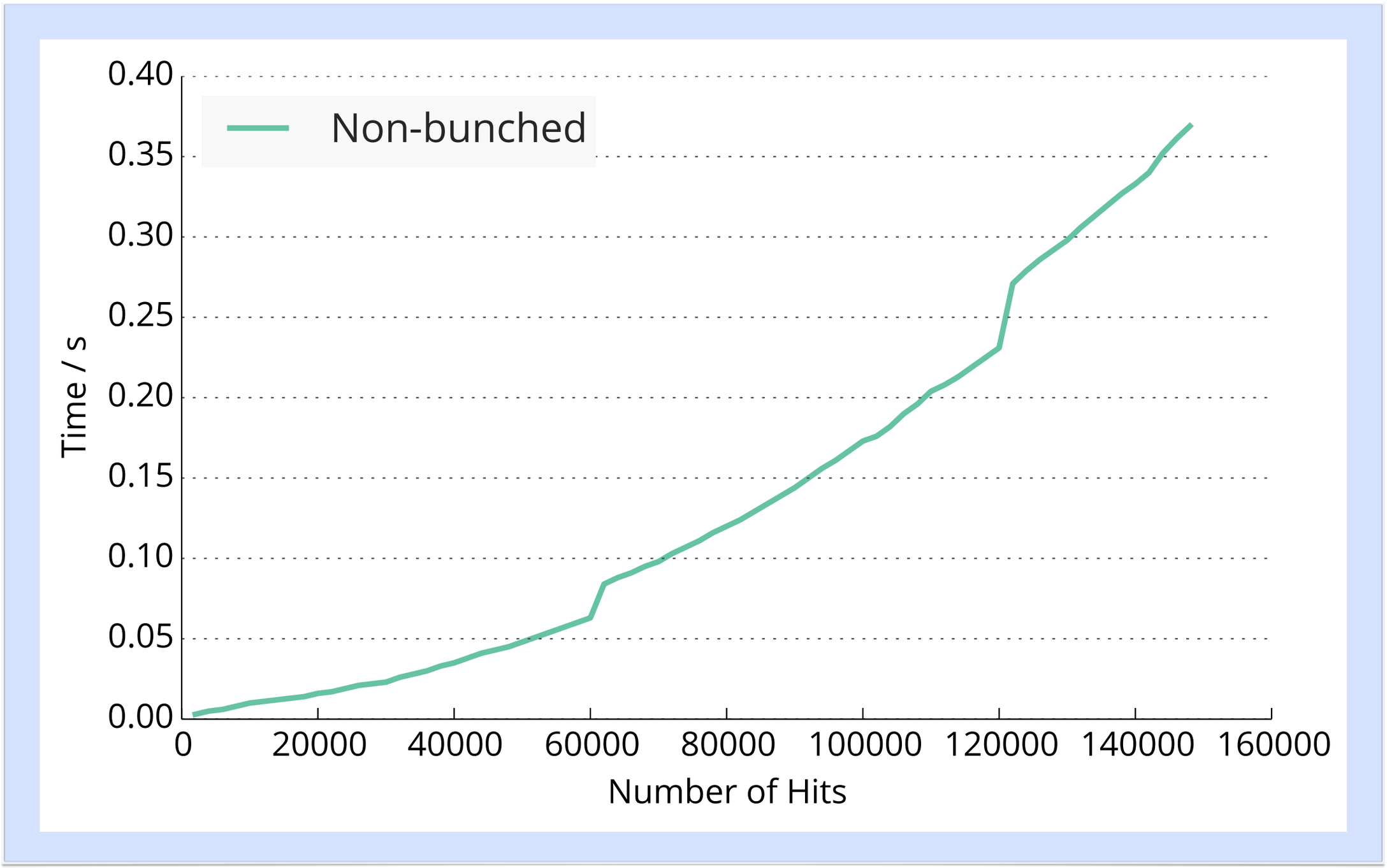
Triplet Finder — Display



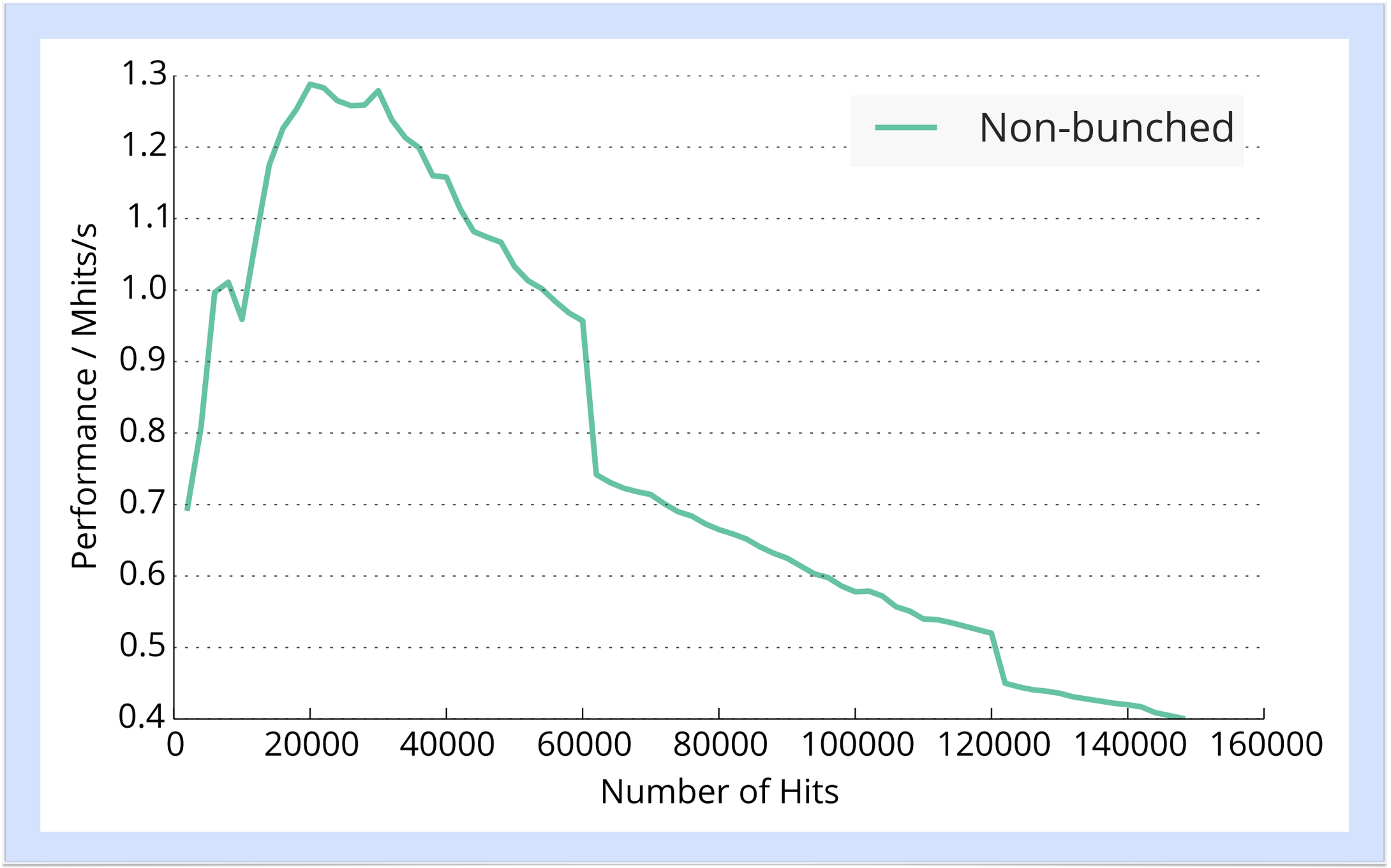
1320 ns

- Isochrone *early*
- Isochrone *early* & *skewed*
- Isochrone *close*
- Isochrone *late*
- MVD hit
- **Triplet**
- Track *current*
- Track *timed out*

Triplet Finder — Times



Triplet Finder — Times



Triplet Finder — Optimizations

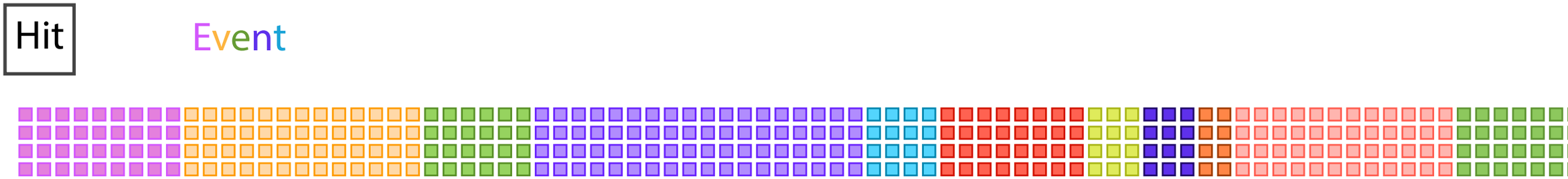
- Bunching Wrapper
  - Hits from one event have similar timestamp
  - Combine hits to sets (bunches) which occupy GPU best

Triplet Finder — Optimizations

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

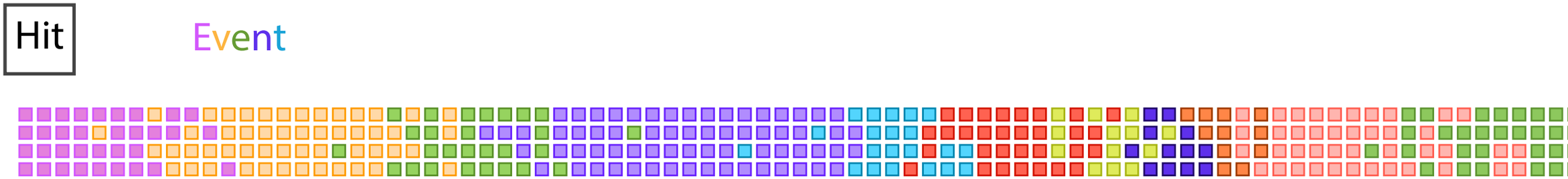
Triplet Finder — Optimizations

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



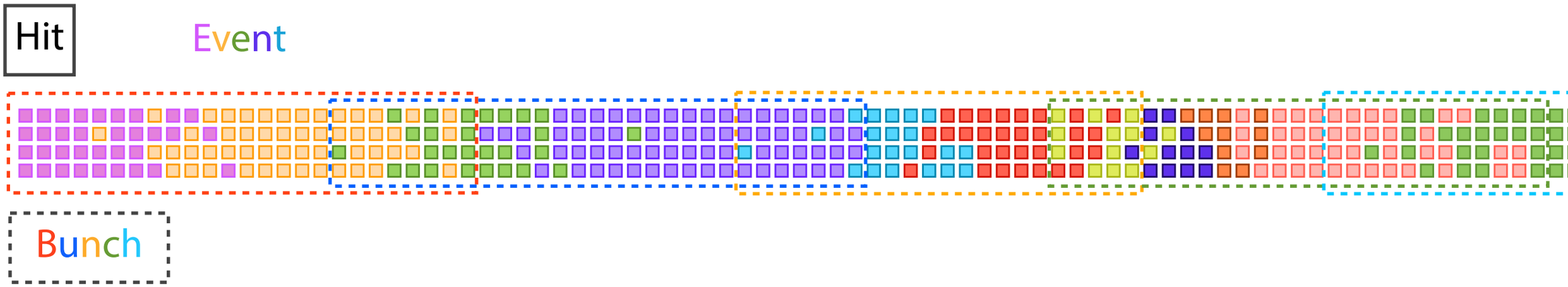
Triplet Finder — Optimizations

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



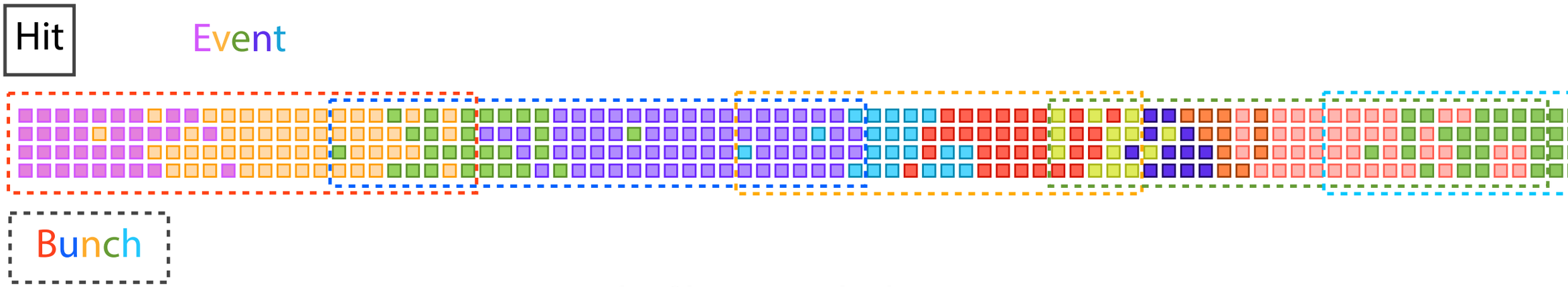
Triplet Finder — Optimizations

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

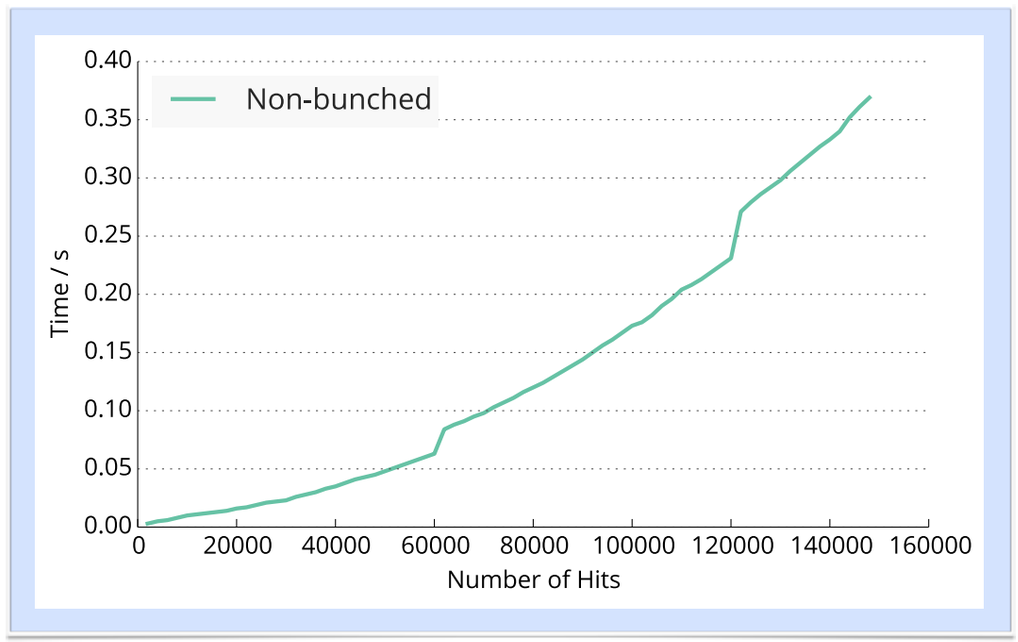


Triplet Finder — Optimizations

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

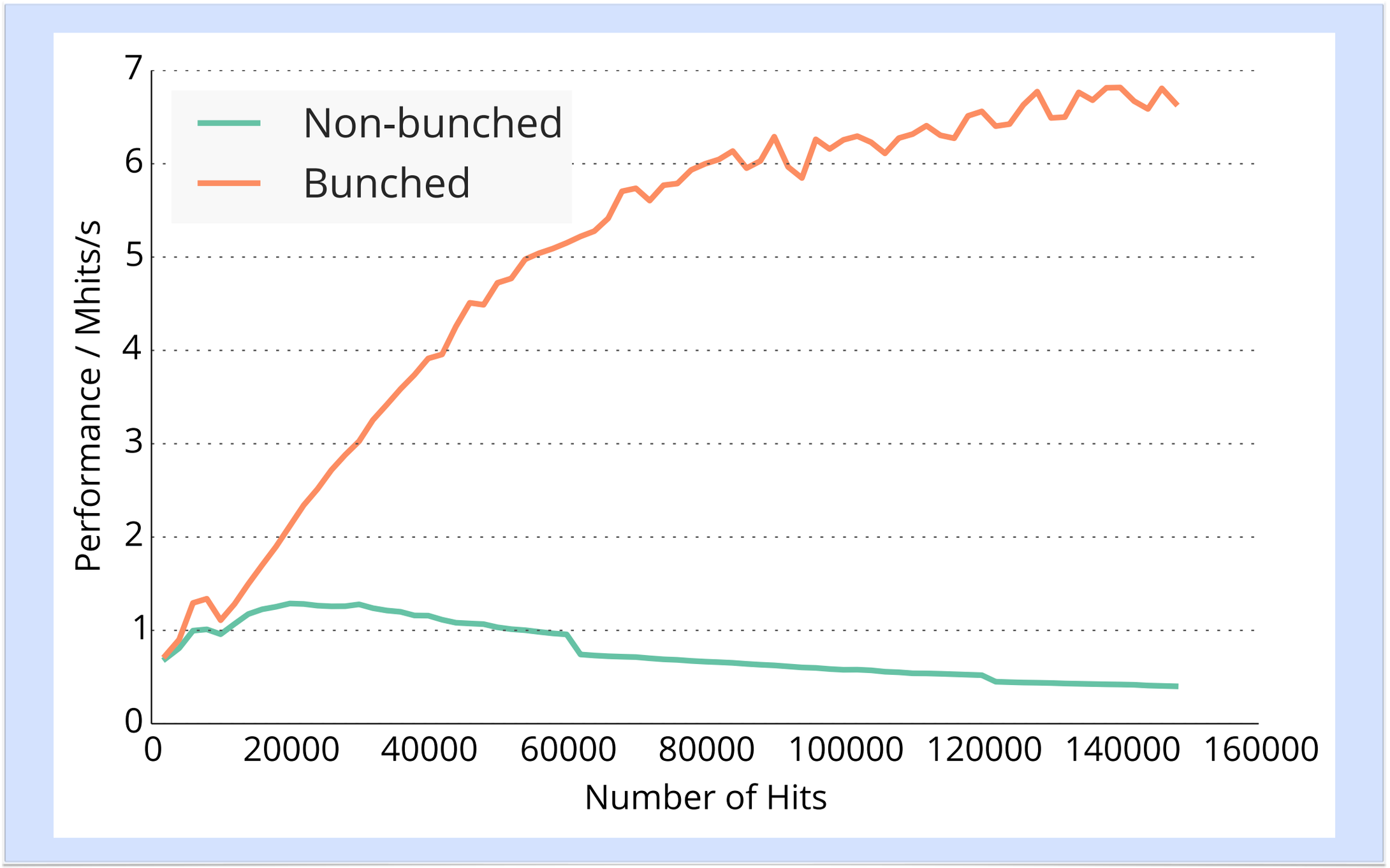


$$\mathcal{O}(N^2) \rightarrow \mathcal{O}(N)$$



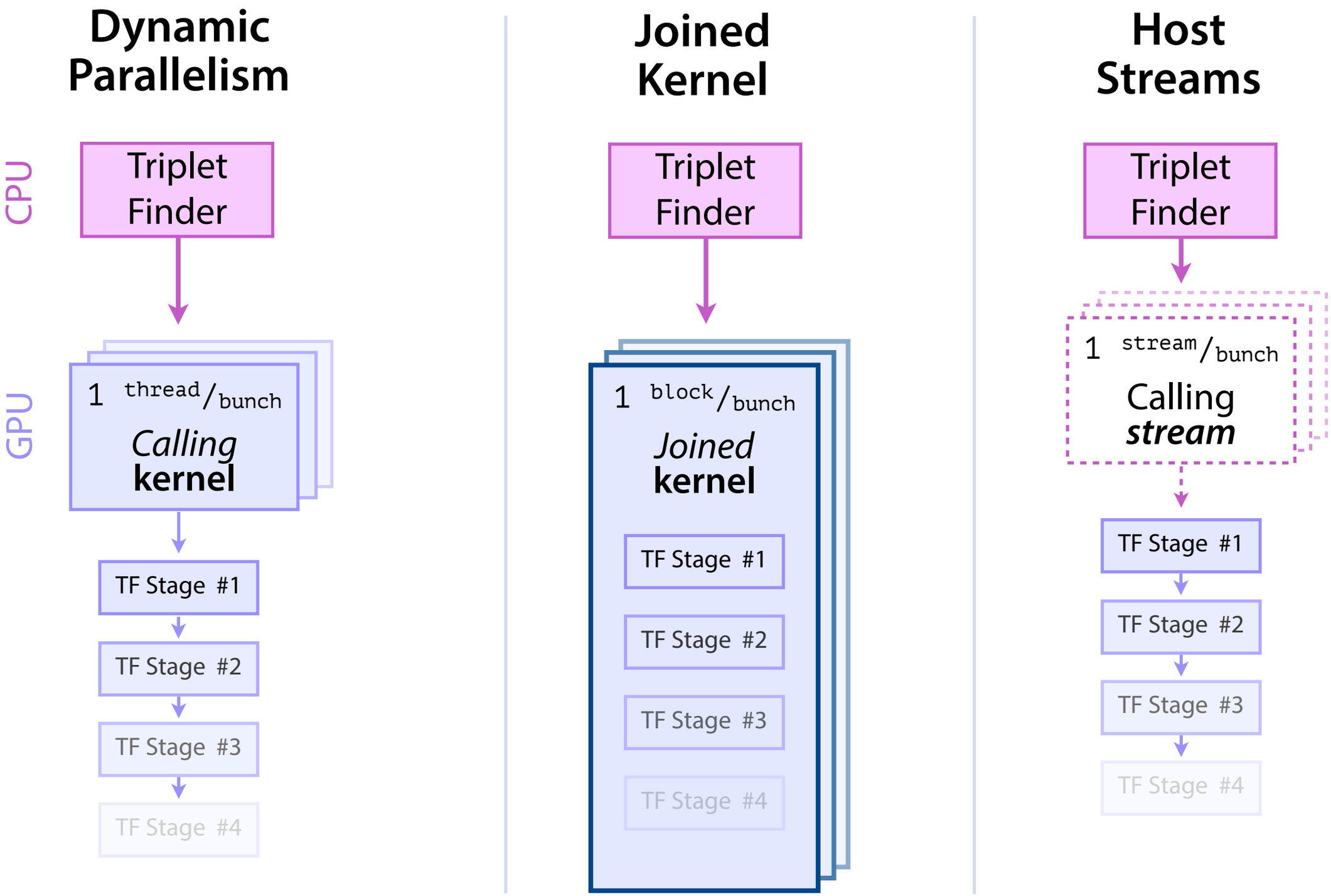
Triplet Finder — Bunching

Performance

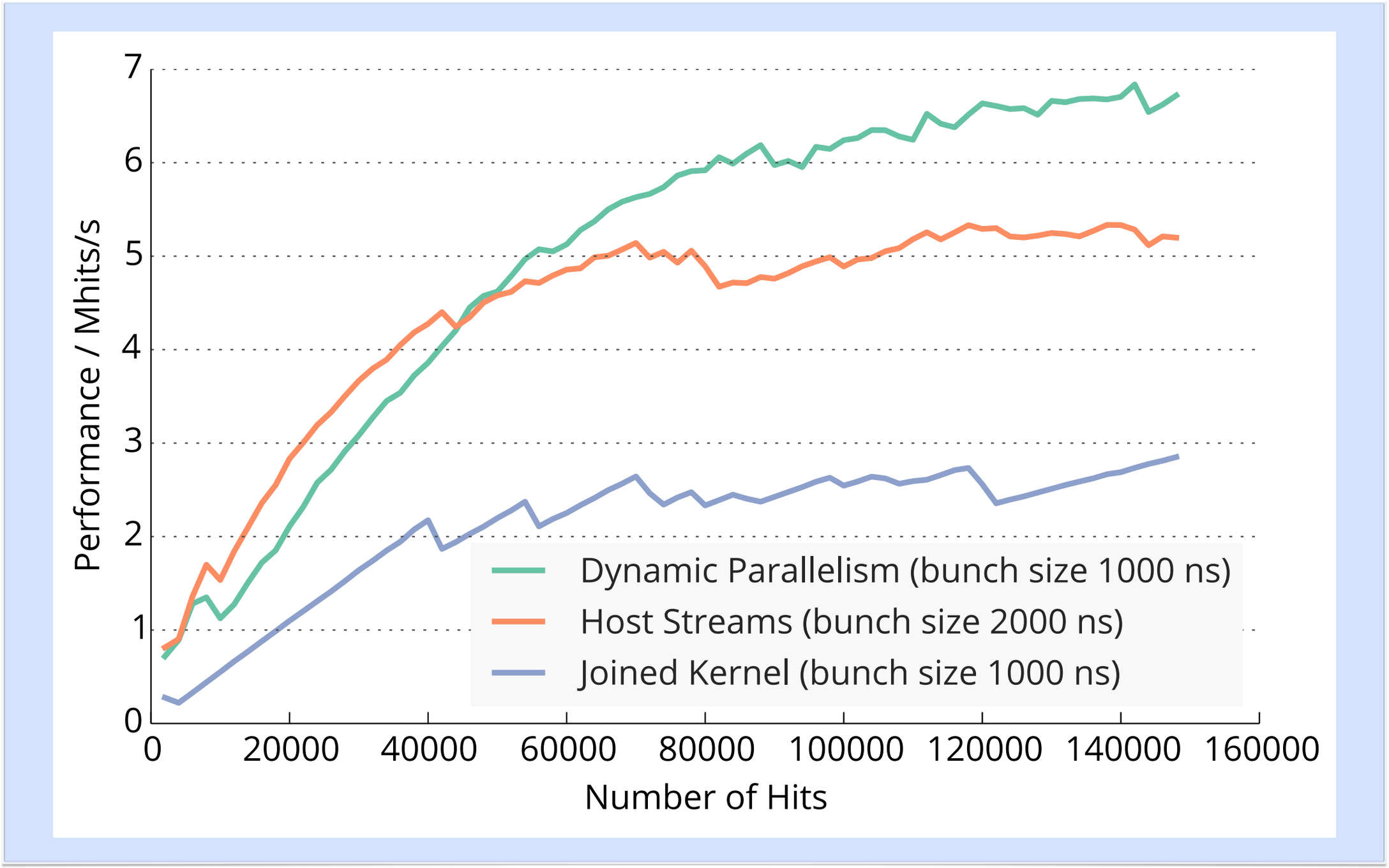


Triplet Finder — Optimizations

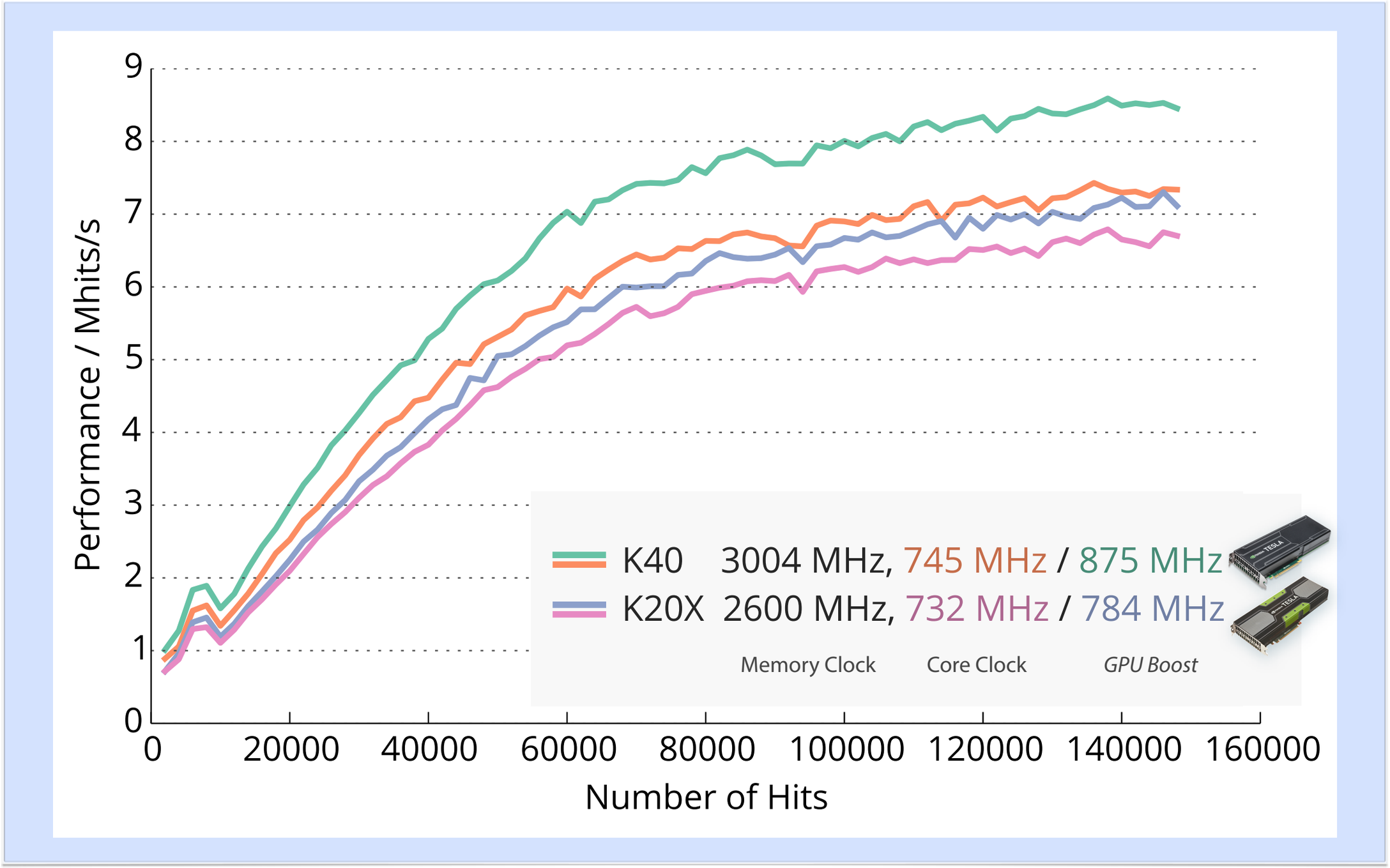
- Compare kernel launch strategies



Triplet Finder — Kernel Launches



Triplet Finder — Clock Speed / GPU



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

- PANDA researches in using GPUs as part of online event reconstruction scheme
- Algorithms in active evaluation and optimization
  - Triplet Finder performance-optimized
- Data transfer to GPU in research: FairMQ

  → Poster by Ludovico Bianchi

Summary

- PANDA researches in using GPUs as part of online event reconstruction scheme
- Algorithms in active evaluation and optimization
  - Triplet Finder performance-optimized
- Data transfer to GPU in research: FairMQ

  → Poster by Ludovico Bianchi

Thank you!

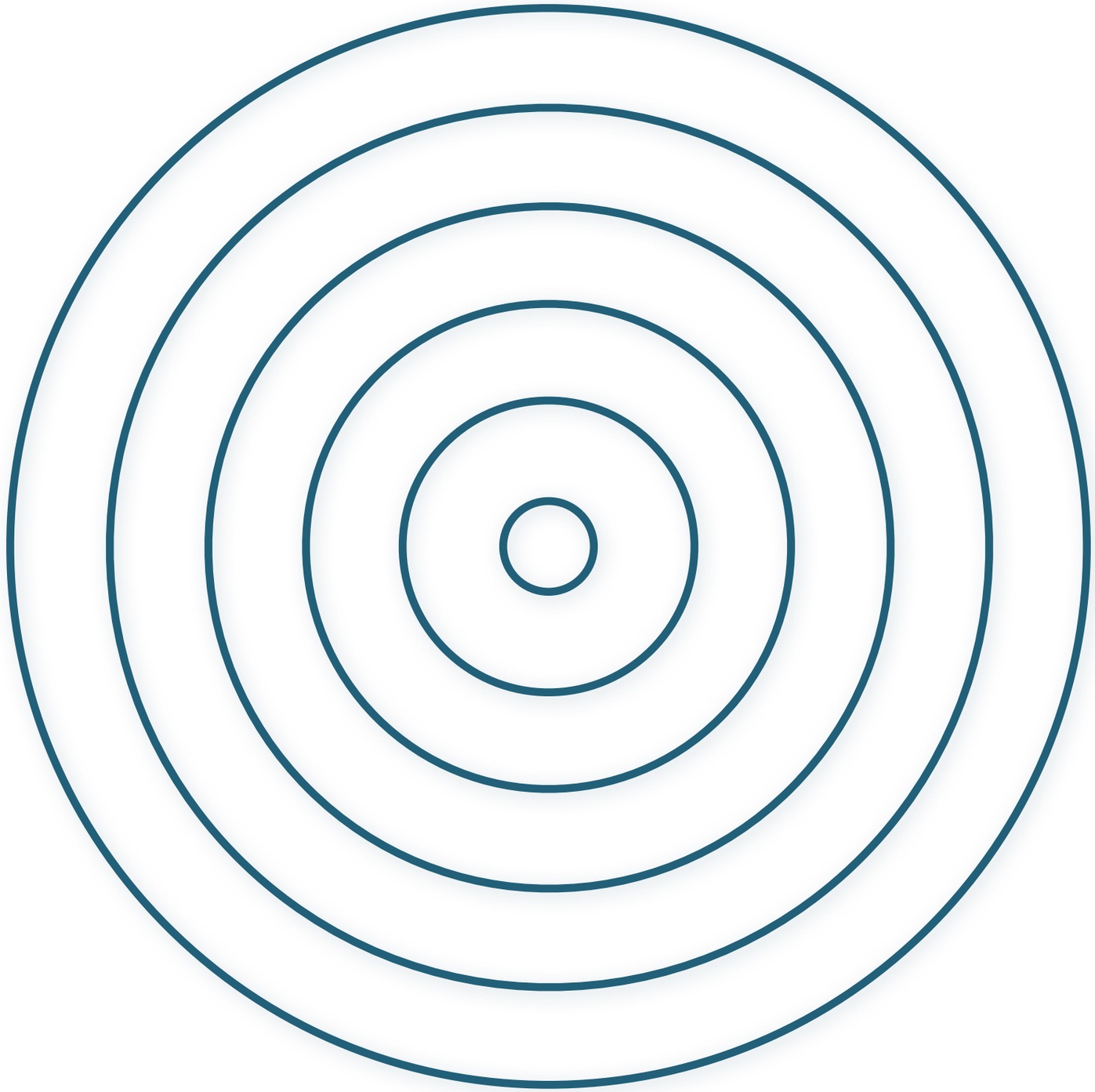
Andreas Herten
a.herten@fz-juelich.de

List of Resources Used

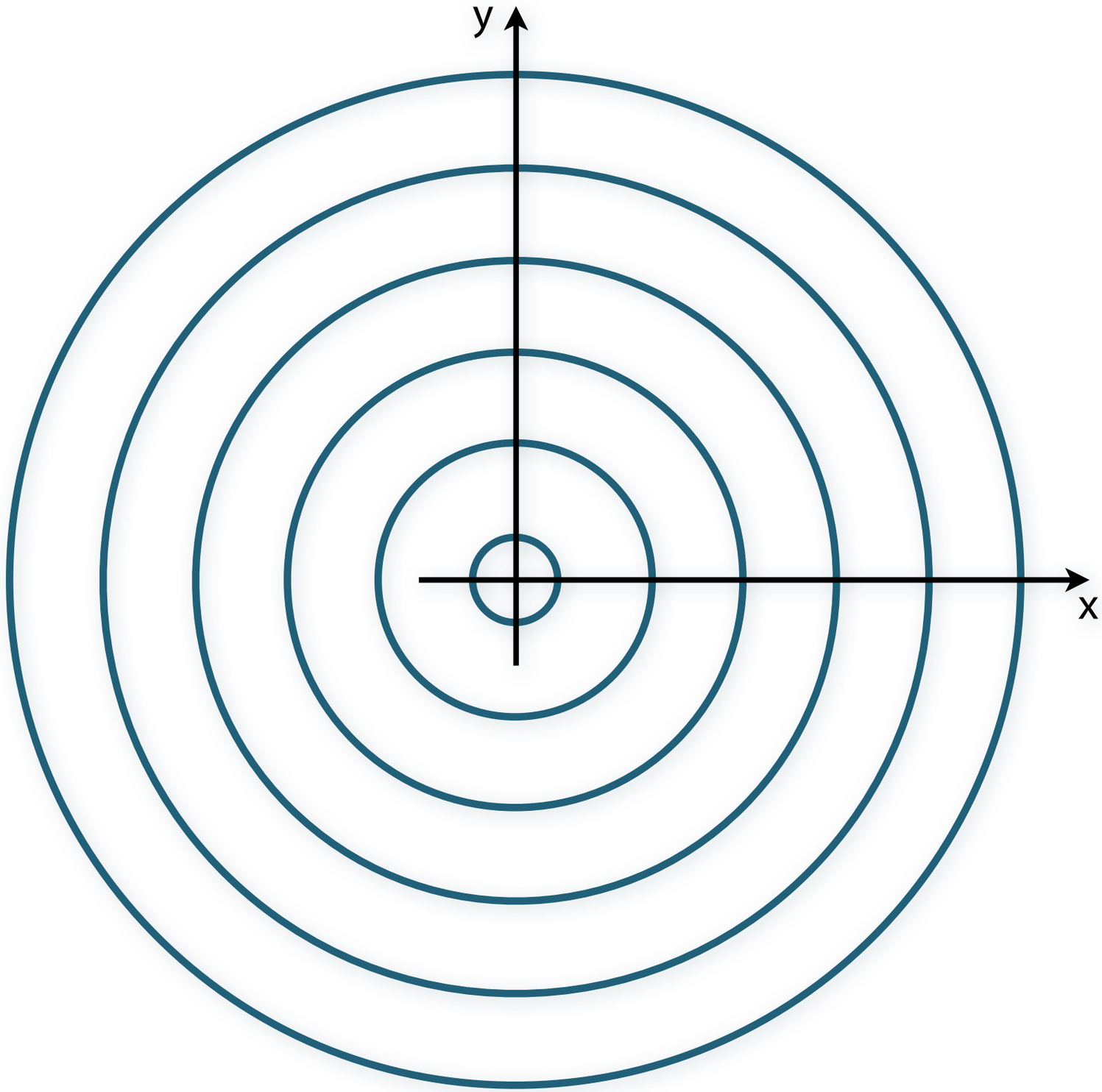
- **#4:** Earth icon by Francesco Paleari from The Noun Project
- **#4:** Einstein icon by Roman Rusinov from The Noun Project
- **#6:** FAIR vector logo from official FAIR website
- **#6:** FAIR rendering from official website
- **#11:** Flare Gun icon by Jop van der Kroef from The Noun Project
- **#27:** 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

BACKUP

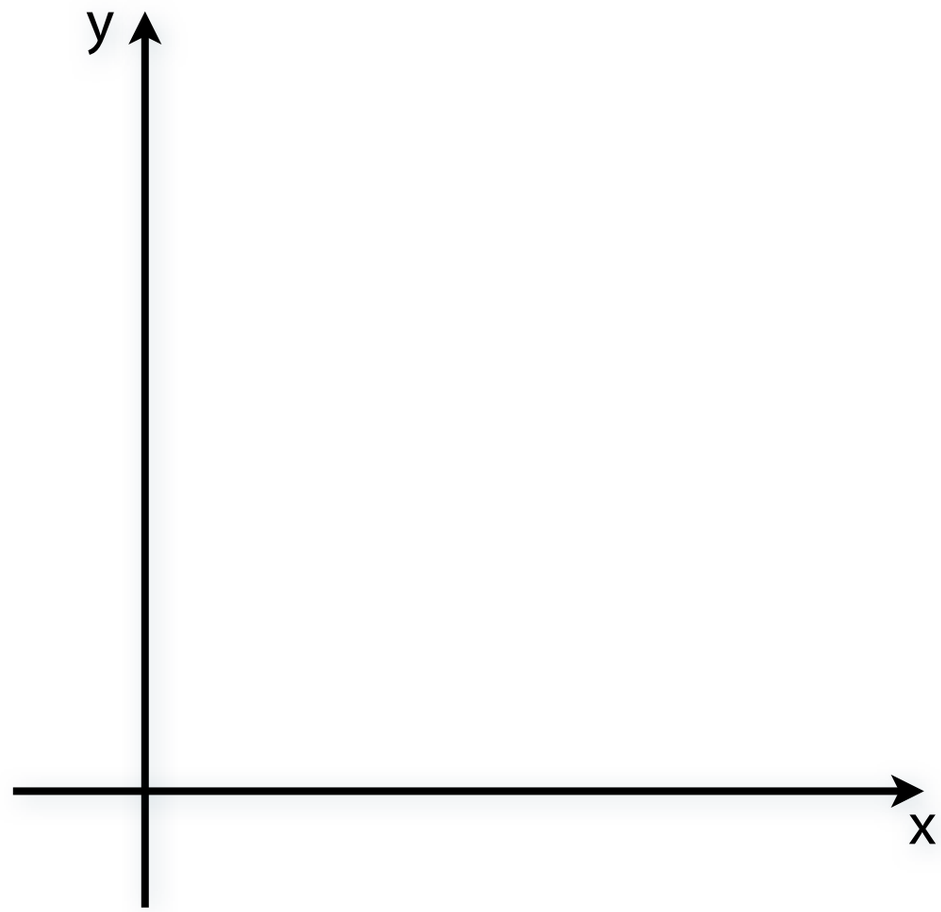
Hough Transform — Principle



Hough Transform — Principle

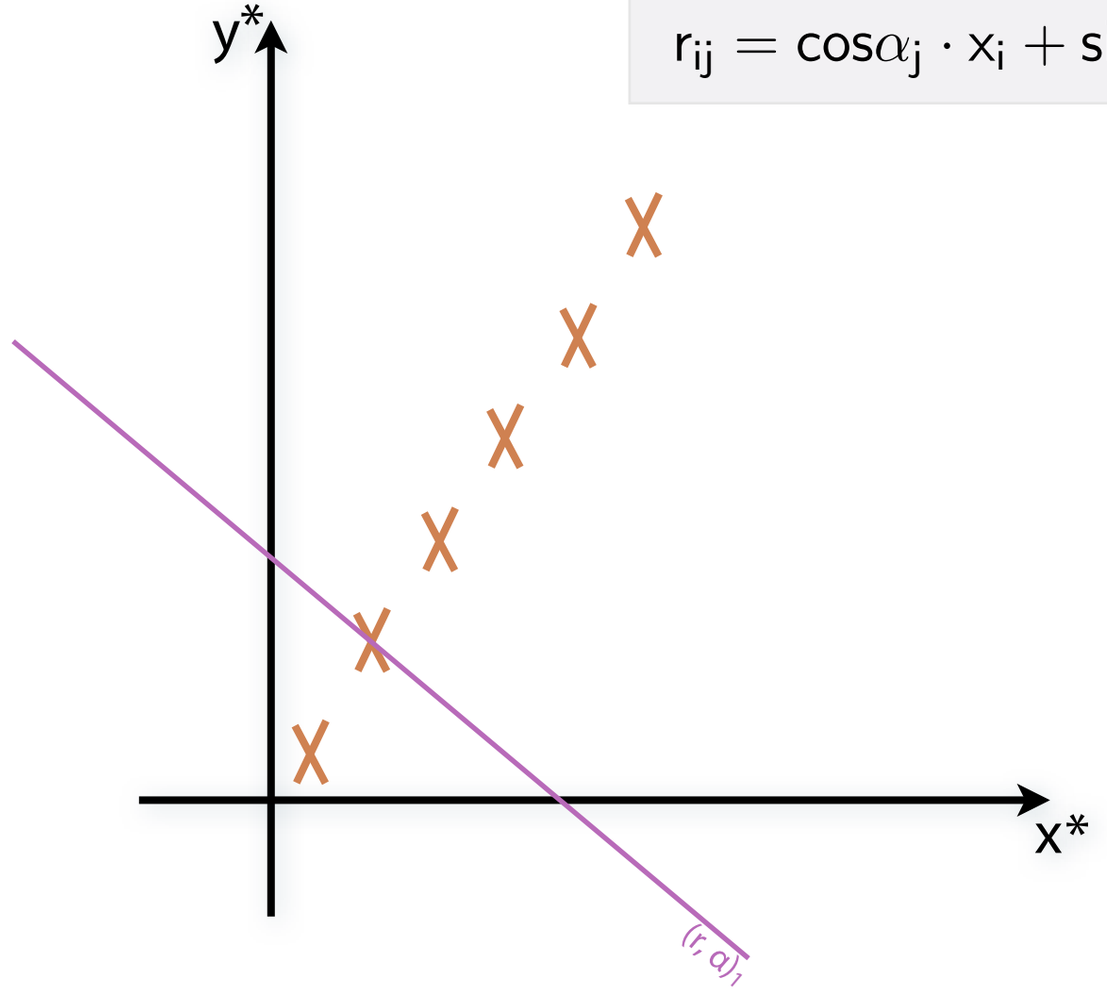


Hough Transform — Principle



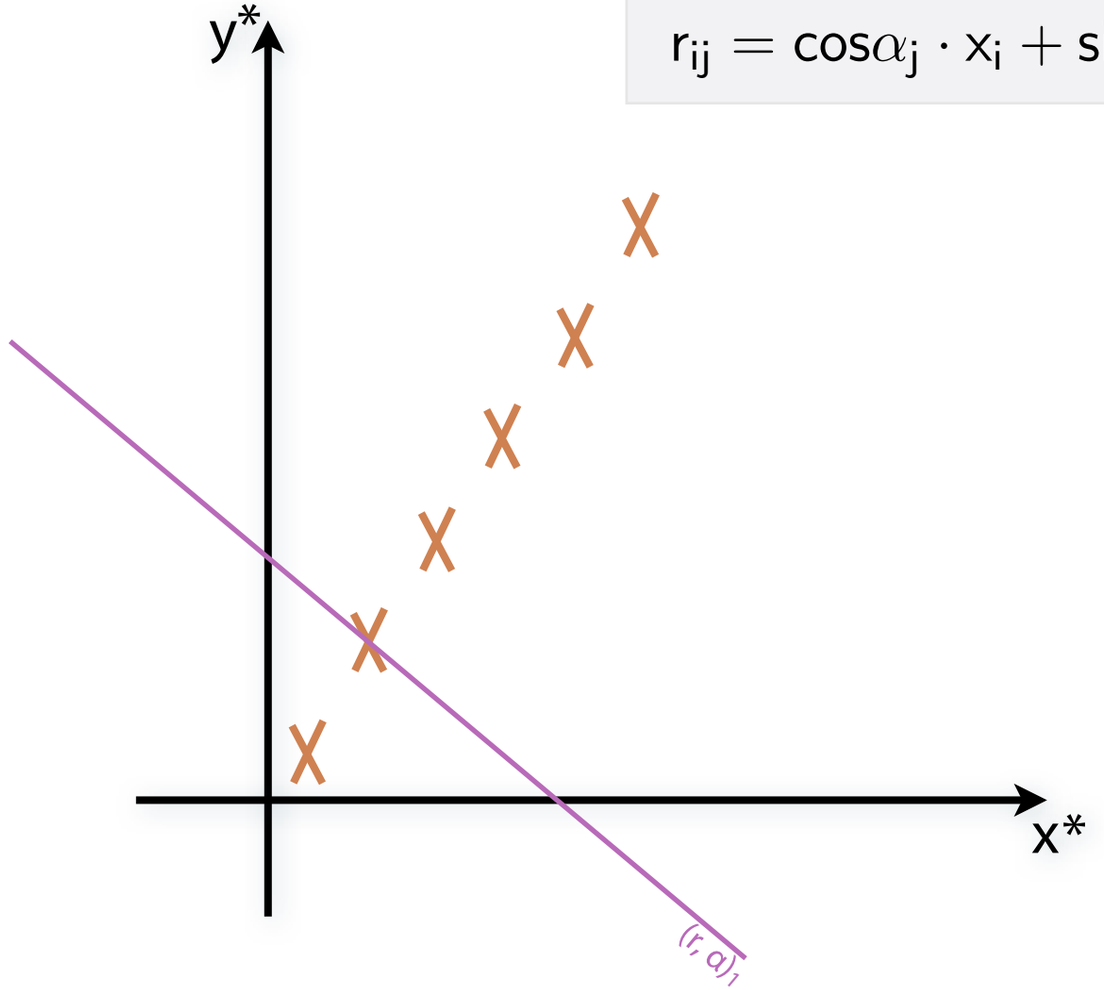
Hough Transform — Principle

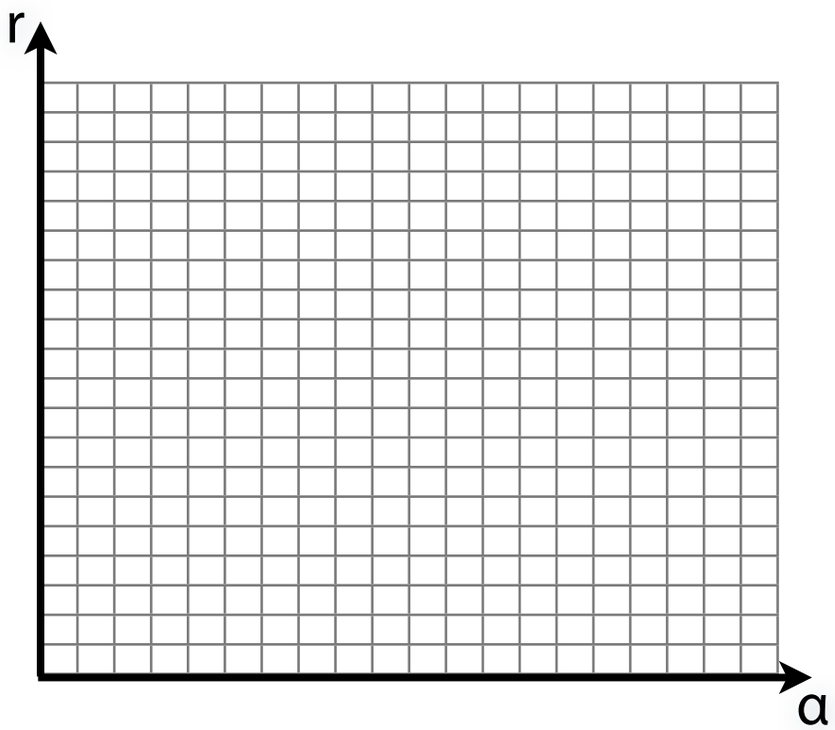
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



Hough Transform — Principle

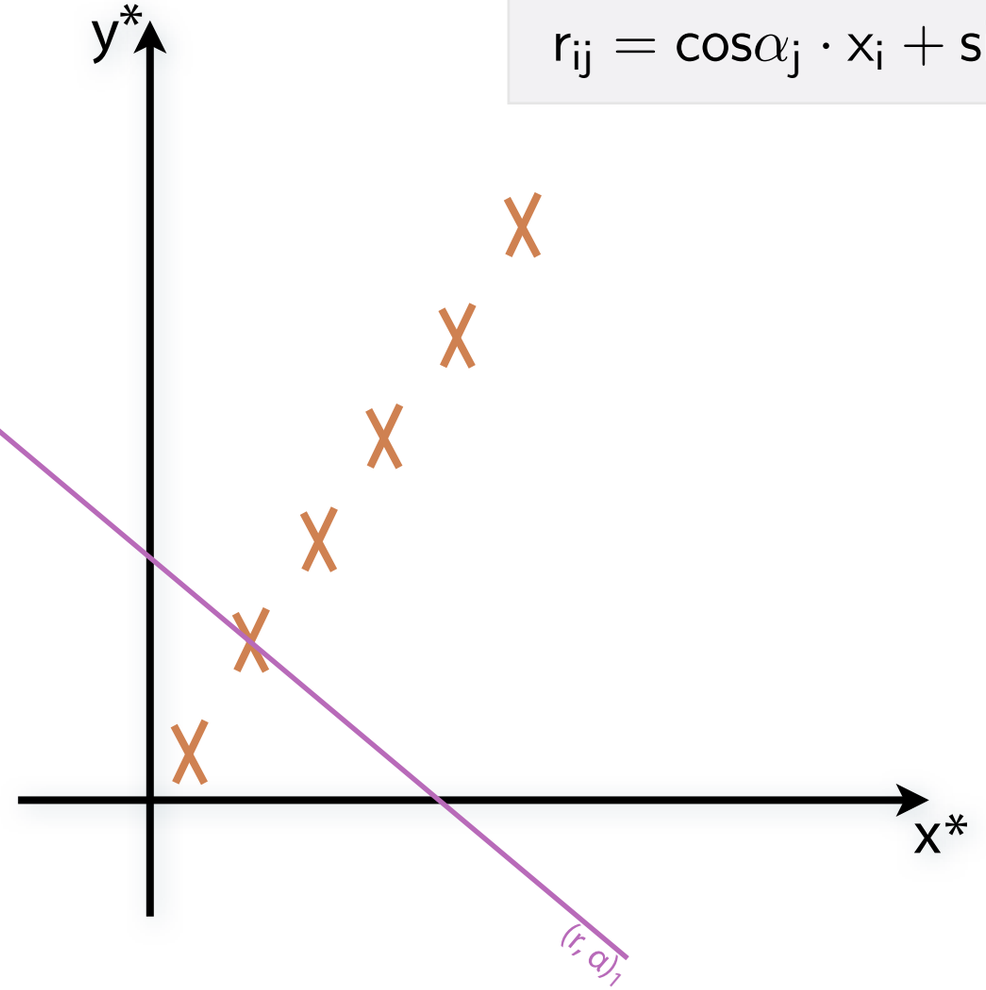
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

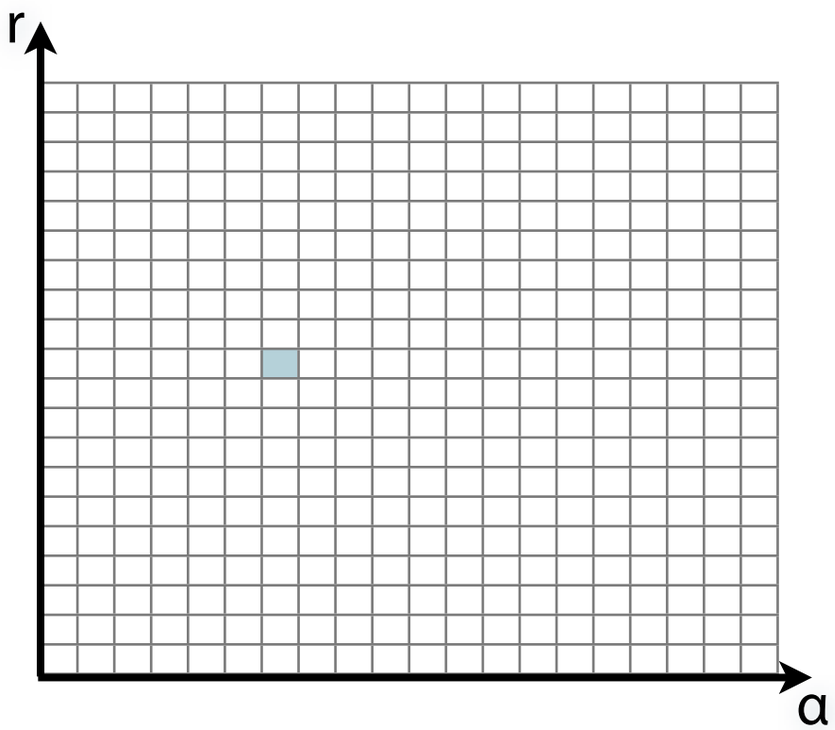




Hough Transform — Principle

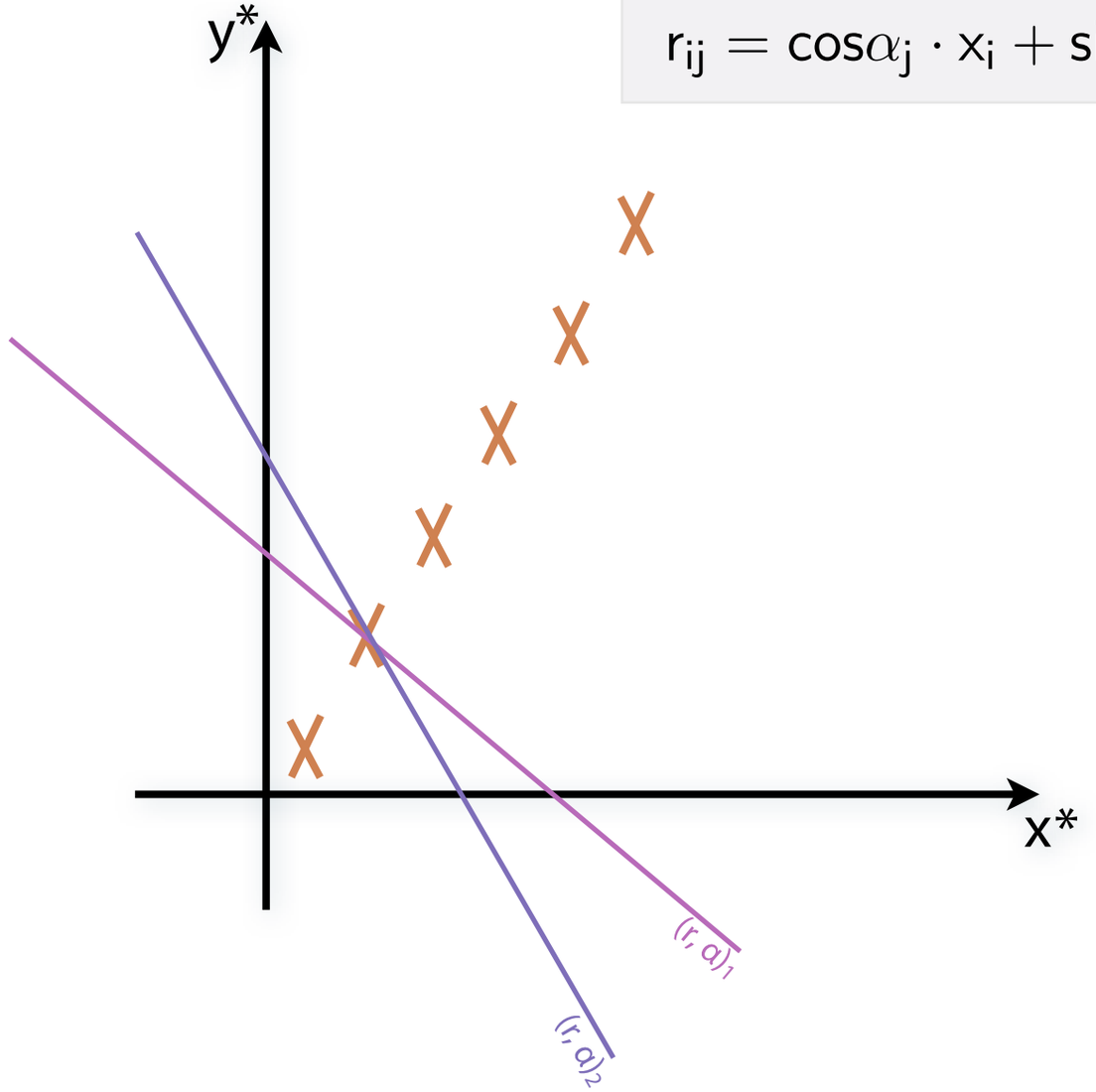
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

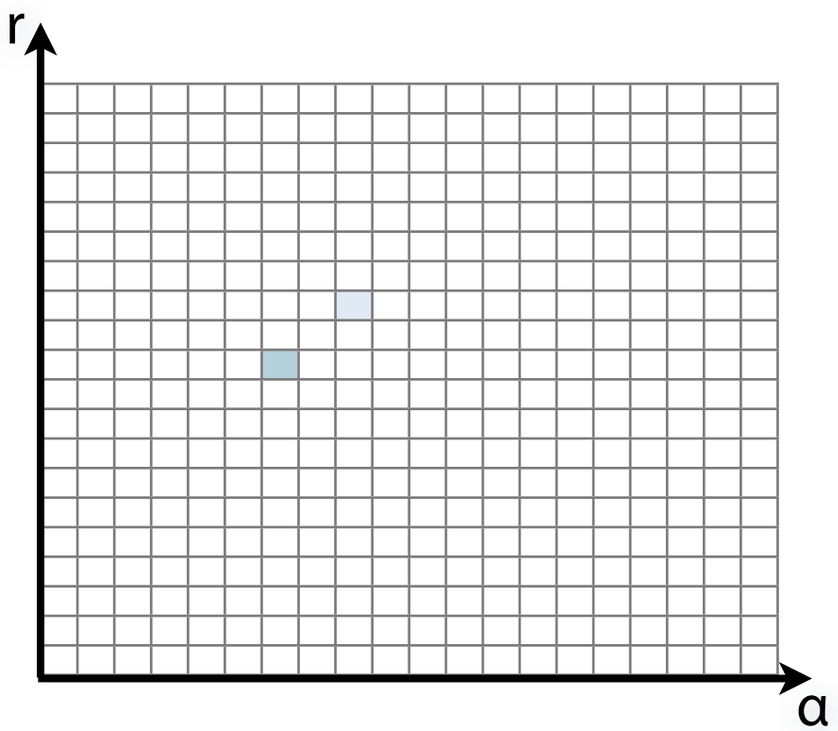




Hough Transform — Principle

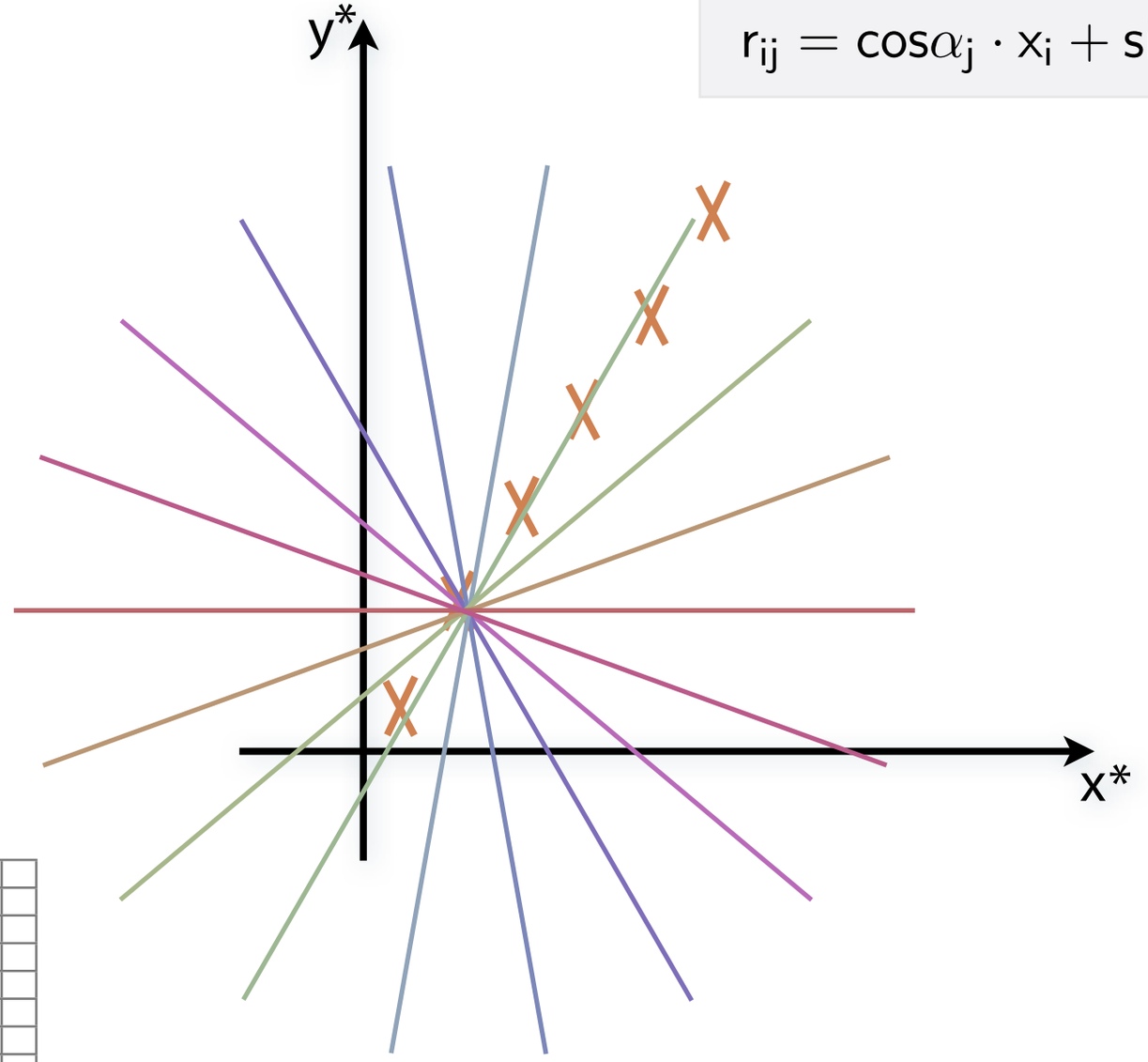
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

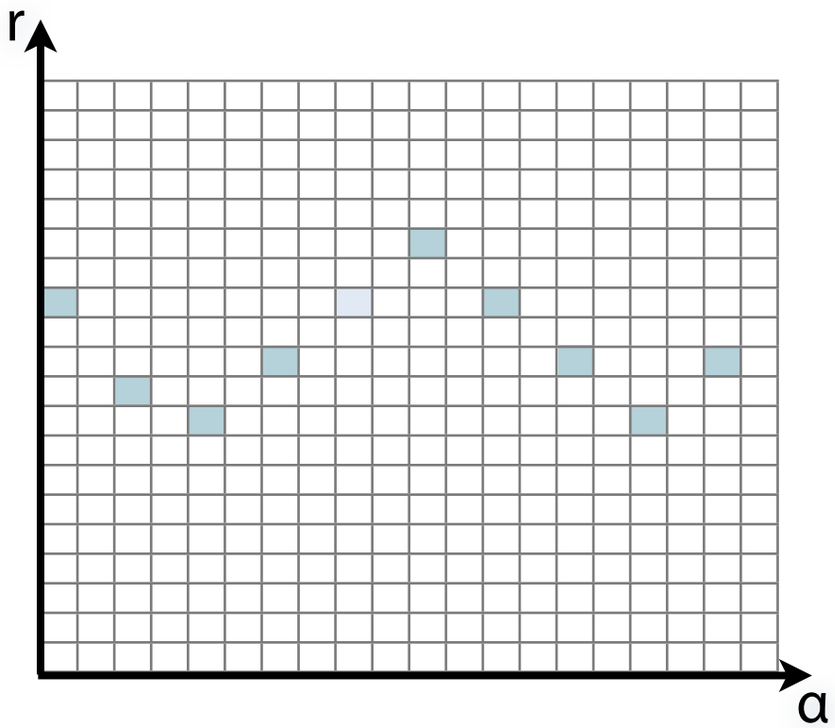




Hough Transform — Principle

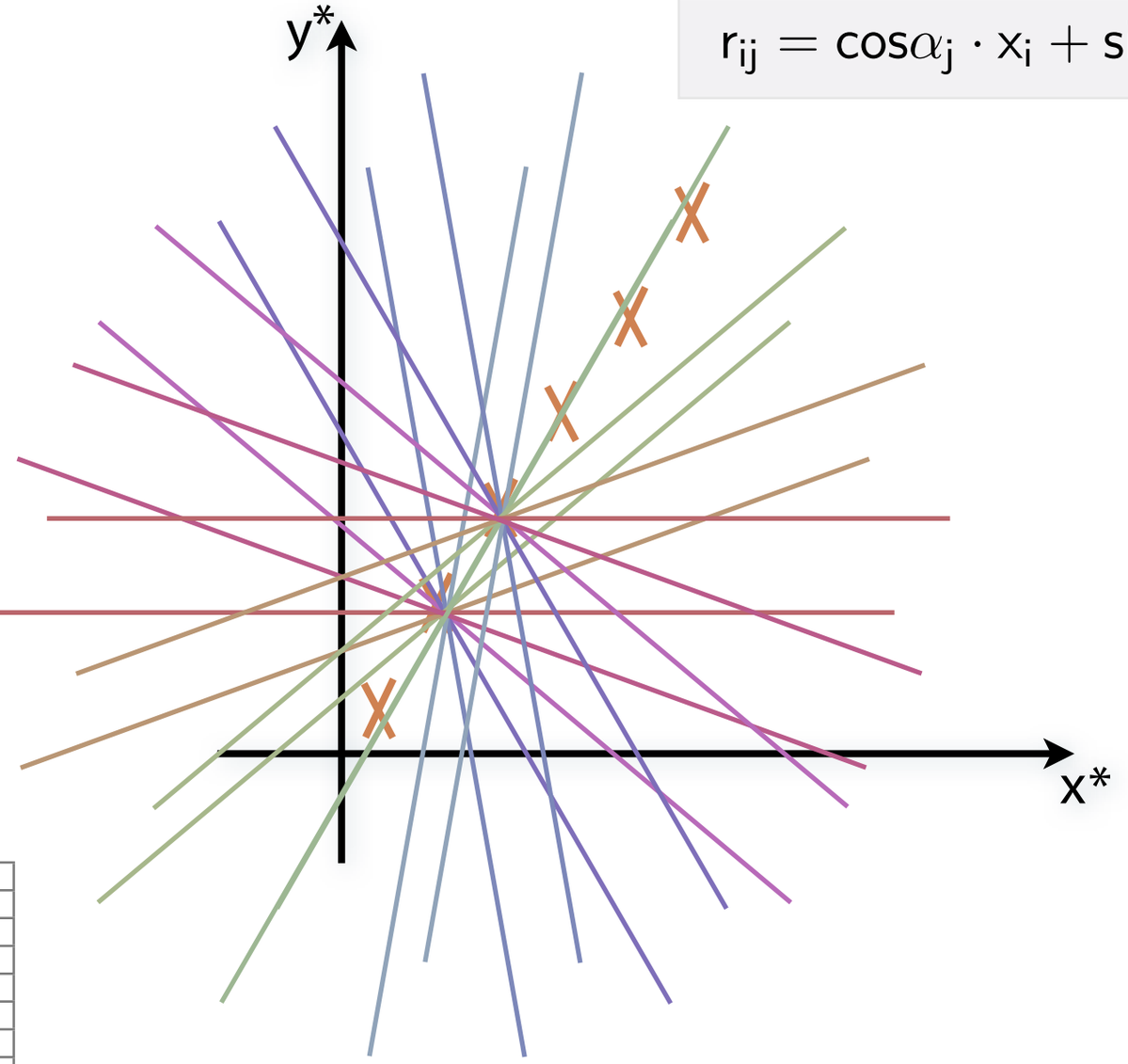
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

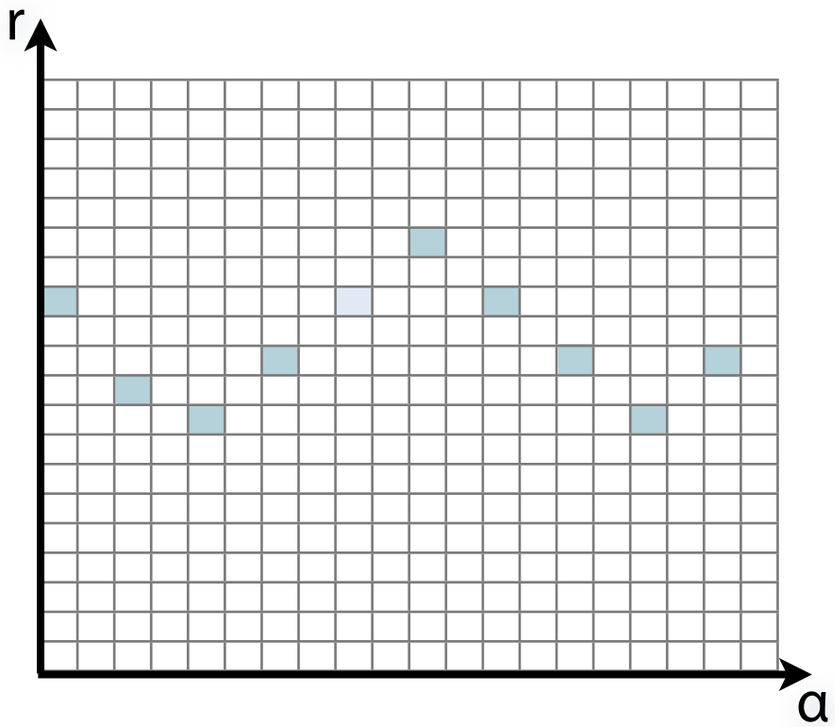




Hough Transform — Principle

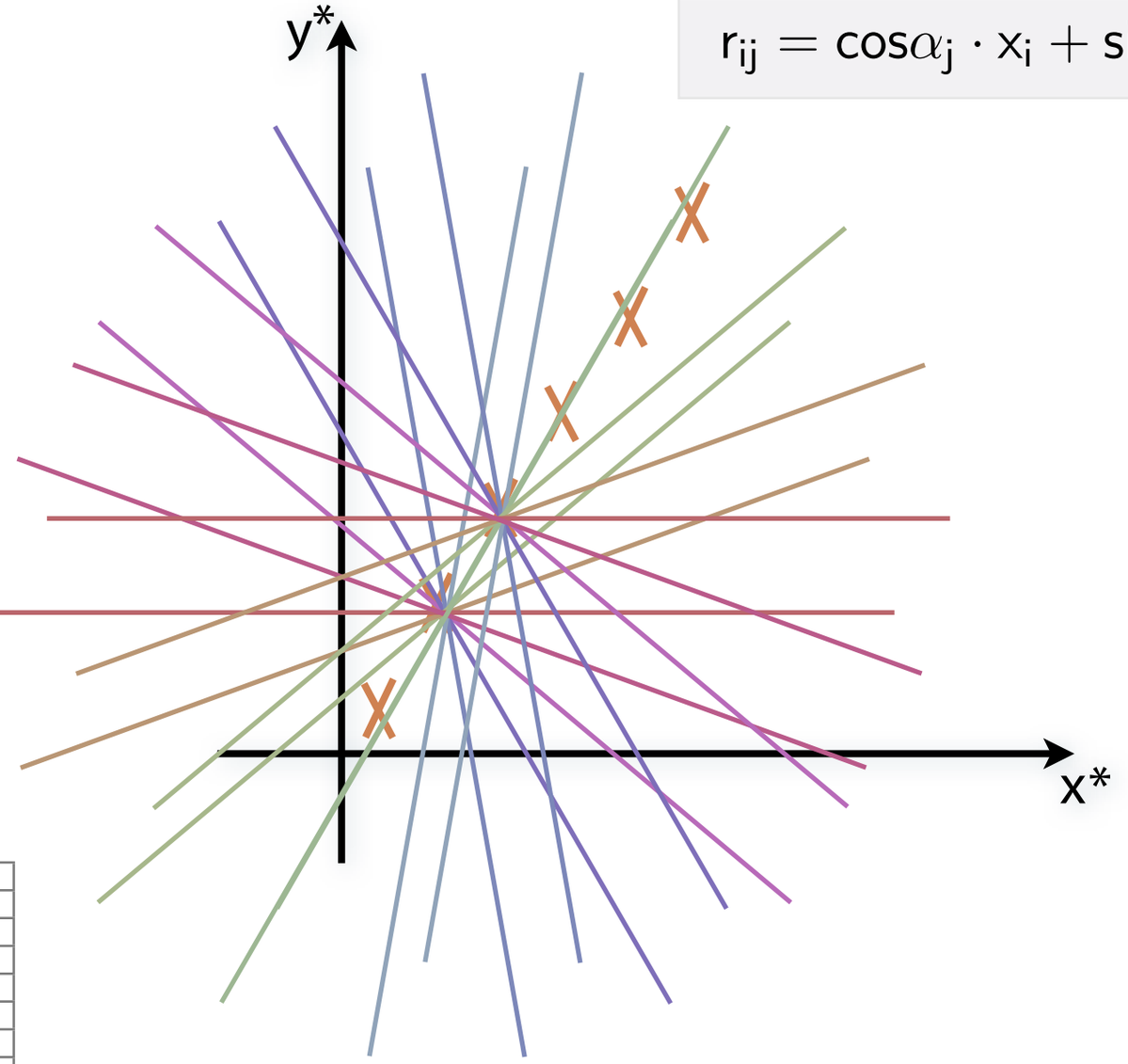
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

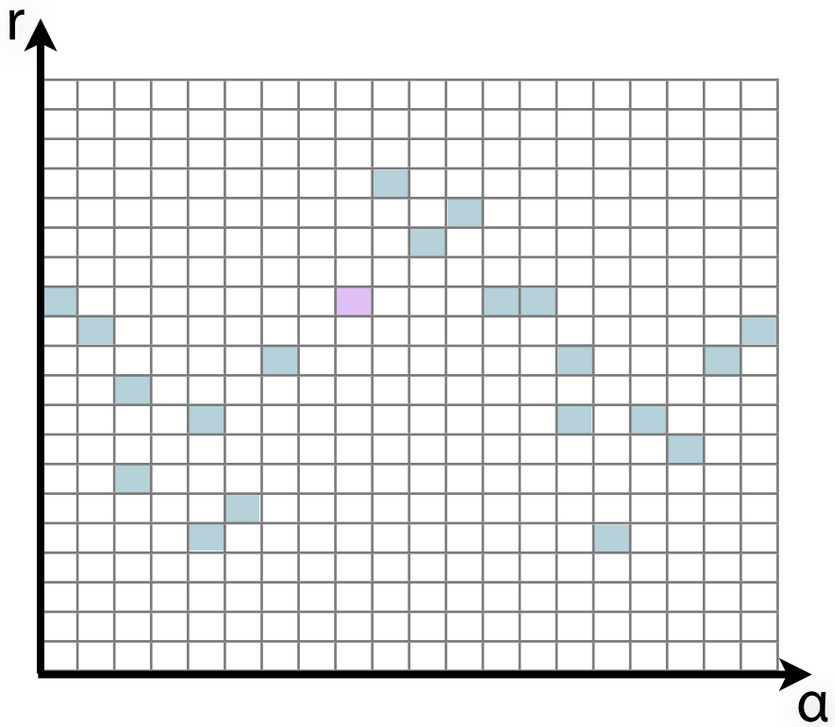




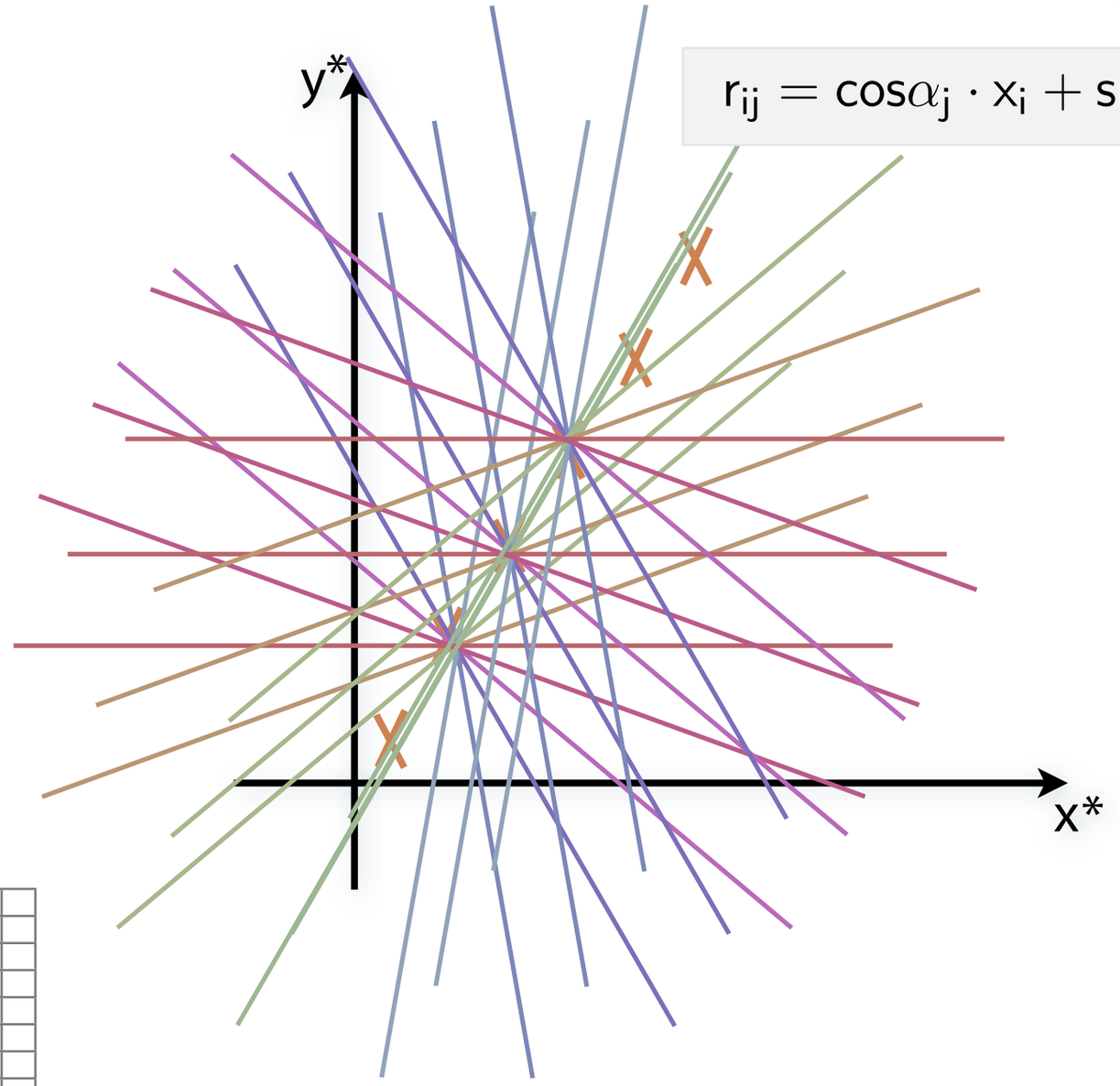
Hough Transform — Principle

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

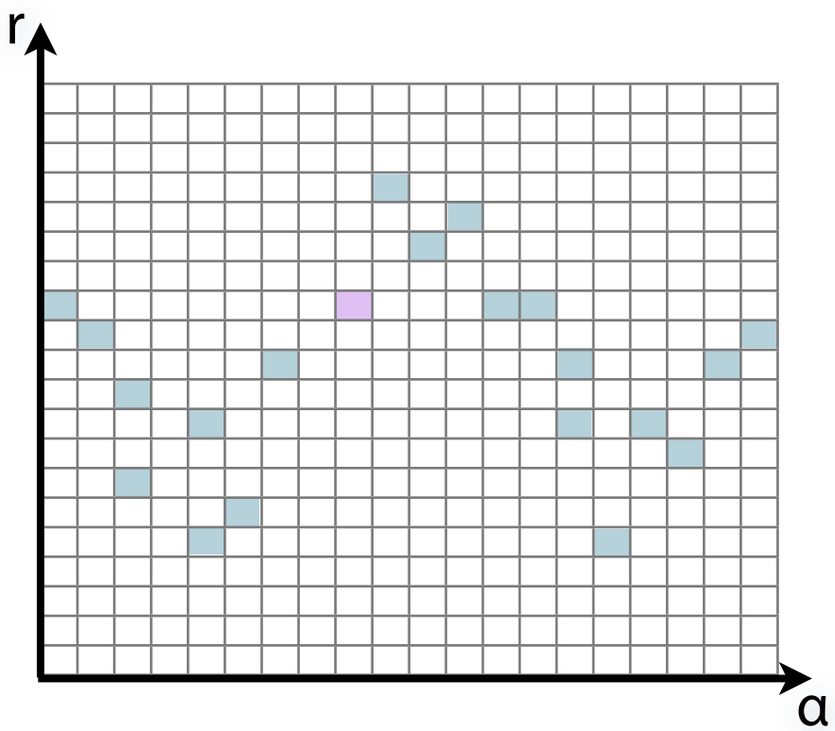




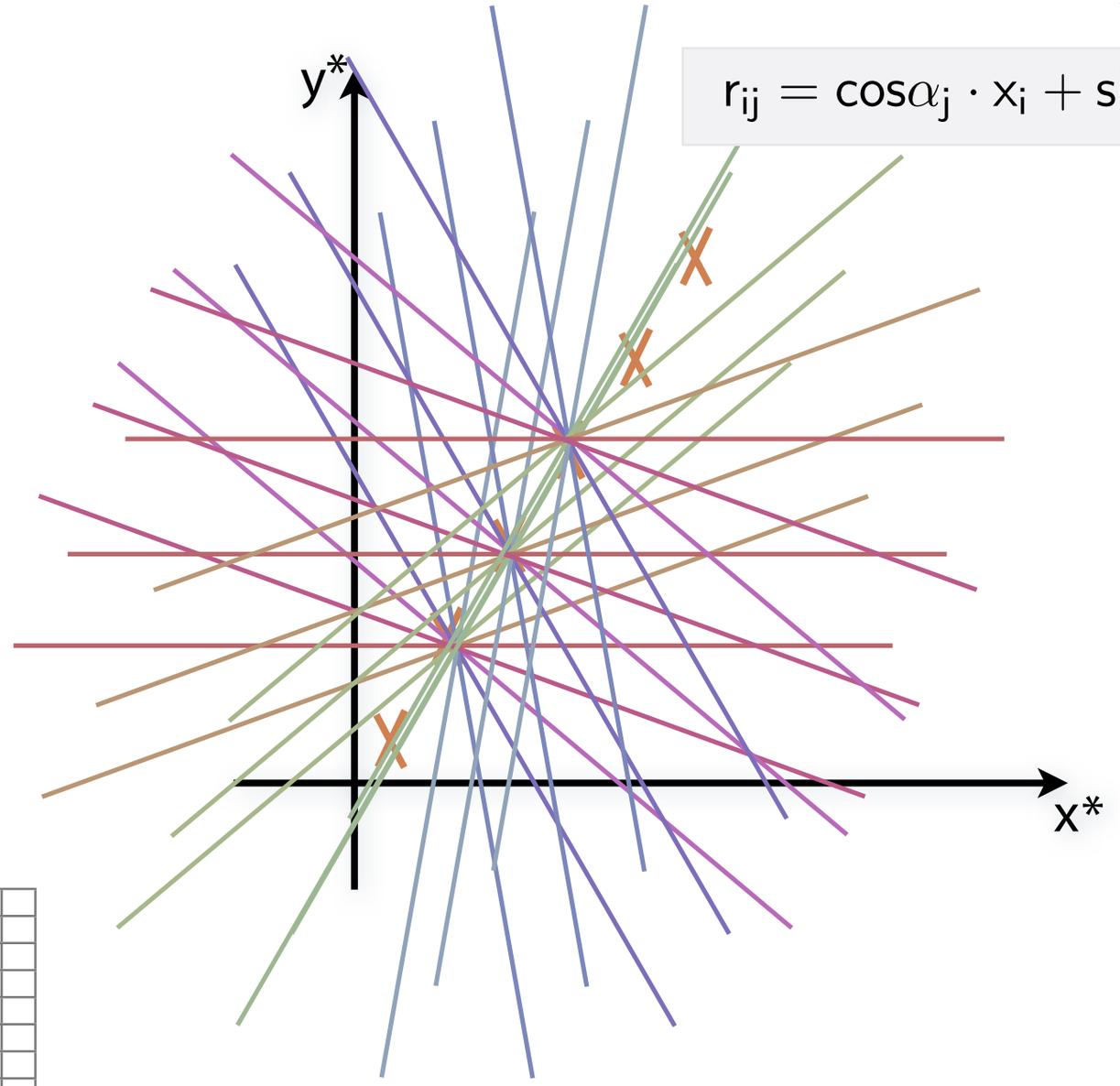
Hough Transform — Principle



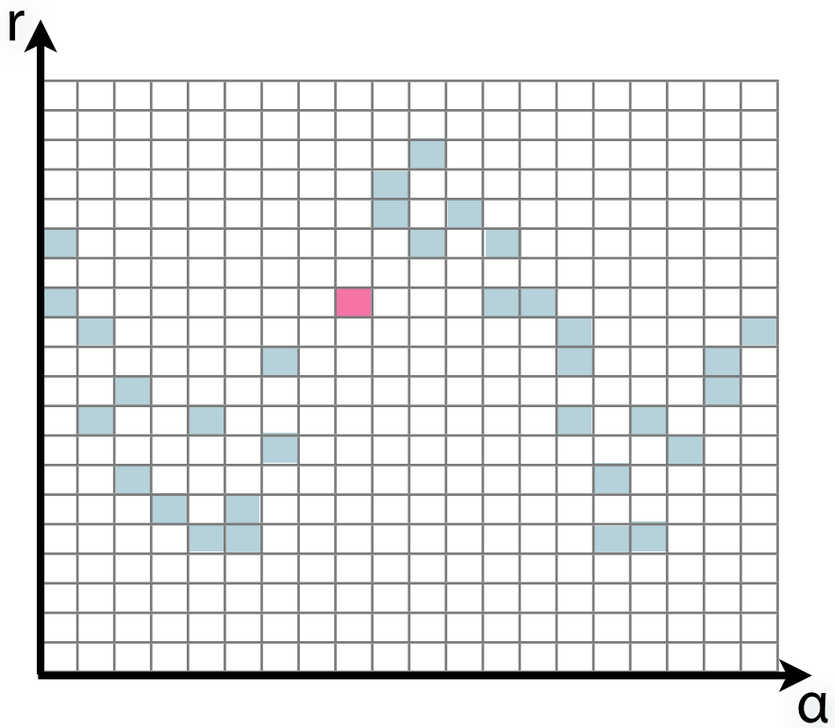
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



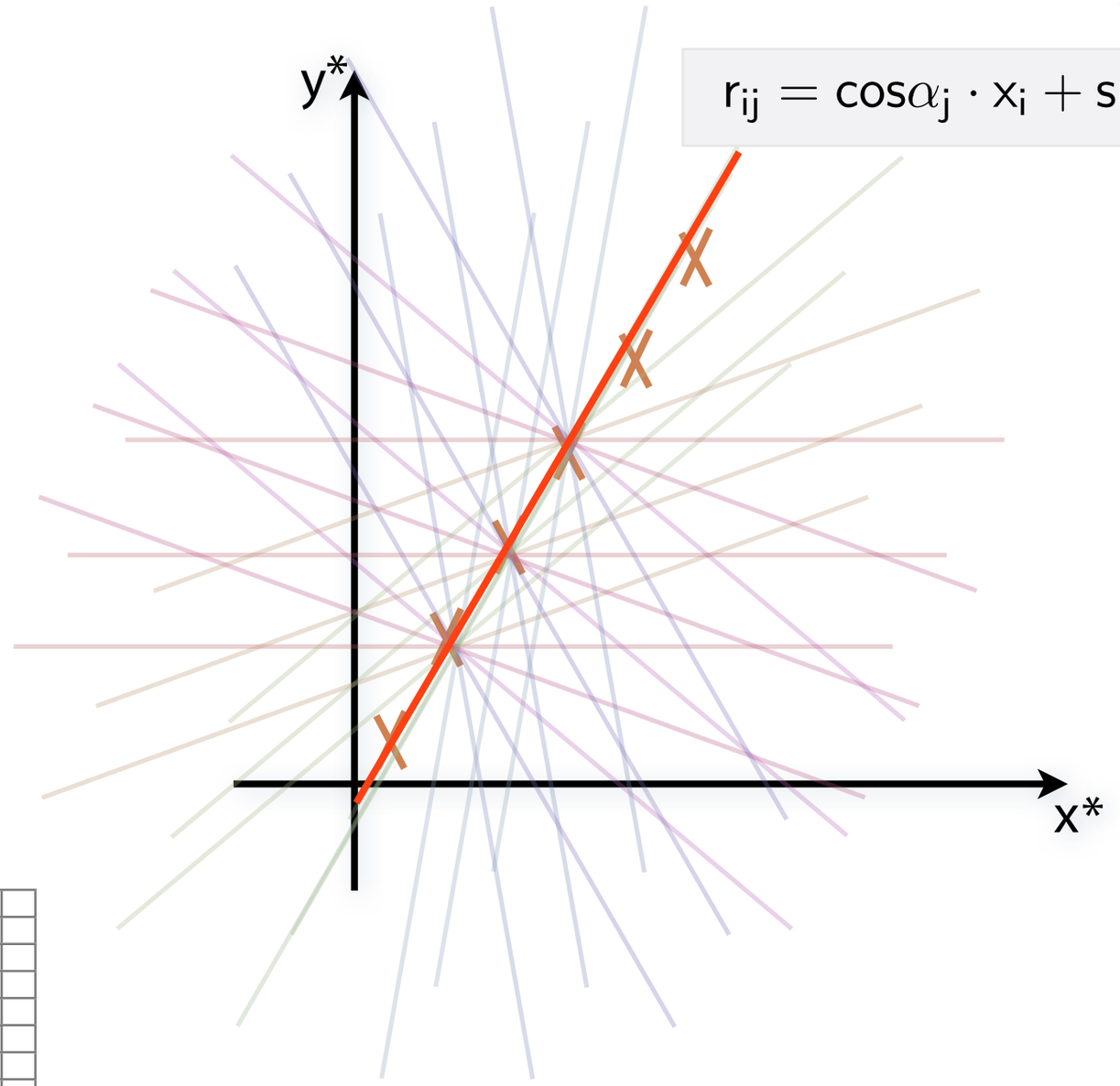
Hough Transform — Principle



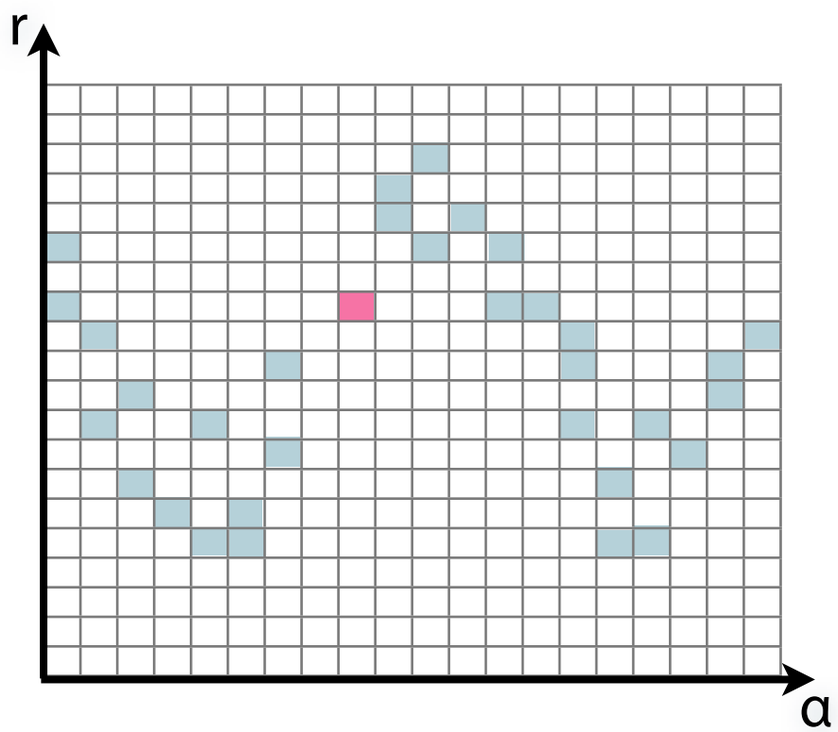
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



Hough Transform — Principle



$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



→ **Bin with highest multiplicity gives track parameters**

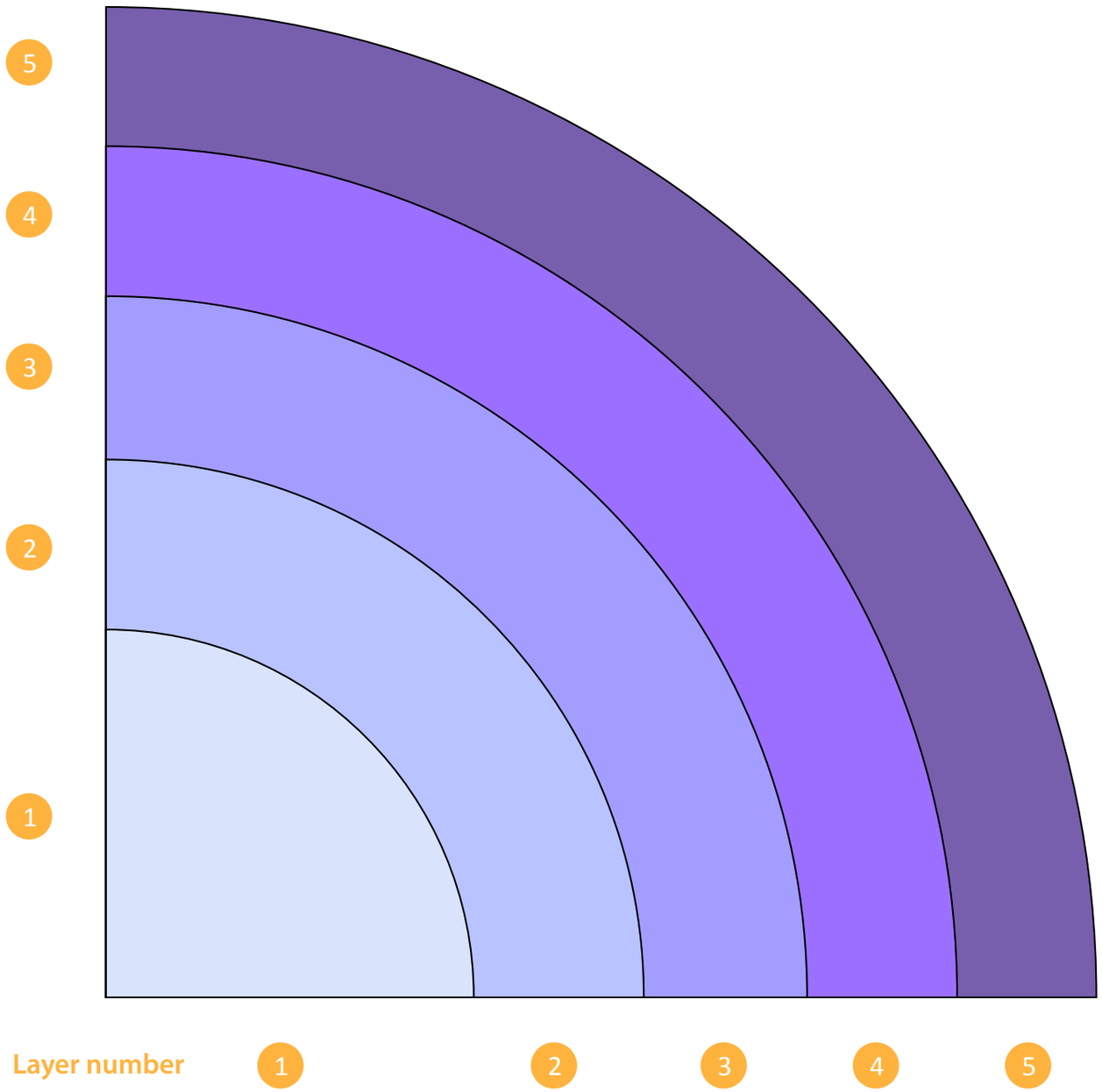
Riemann Algorithm — Procedure

1. **Create triplet** of hit points
   - All possible three hit combinations need to become triplets

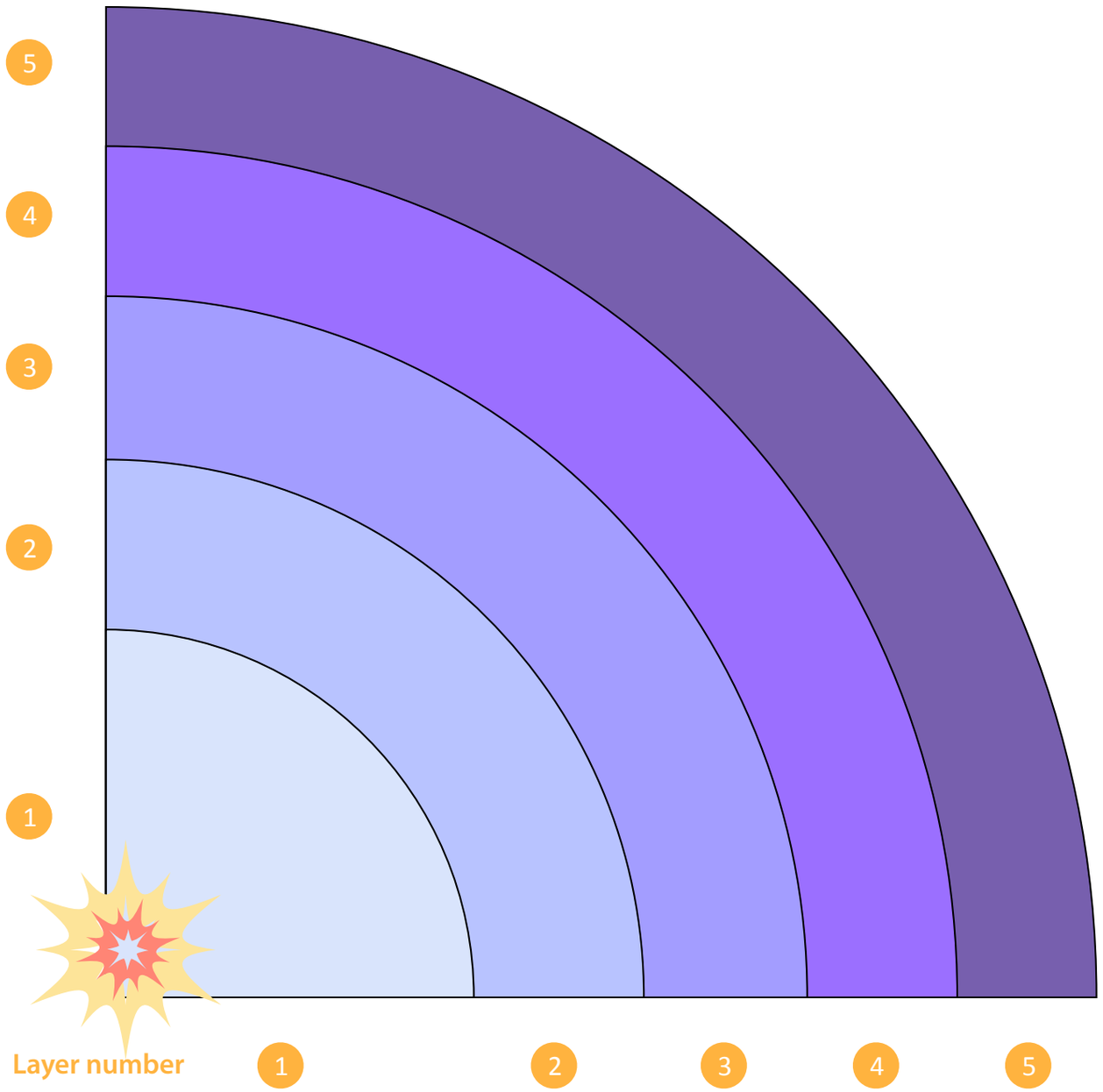
Riemann Algorithm — Procedure

1. **Create triplet** of hit points
   - All possible three hit combinations need to become triplets
2. **Grow** triplets to **tracks**:
   Continuously test next hit if it fits to triplet track
   - Use Riemann paraboloid to circle fit track
     - Test closeness of new hit: good → add hit; bad → dismiss hit
     - Continue with next hit
   - Helix fit: arc length s vs. z position

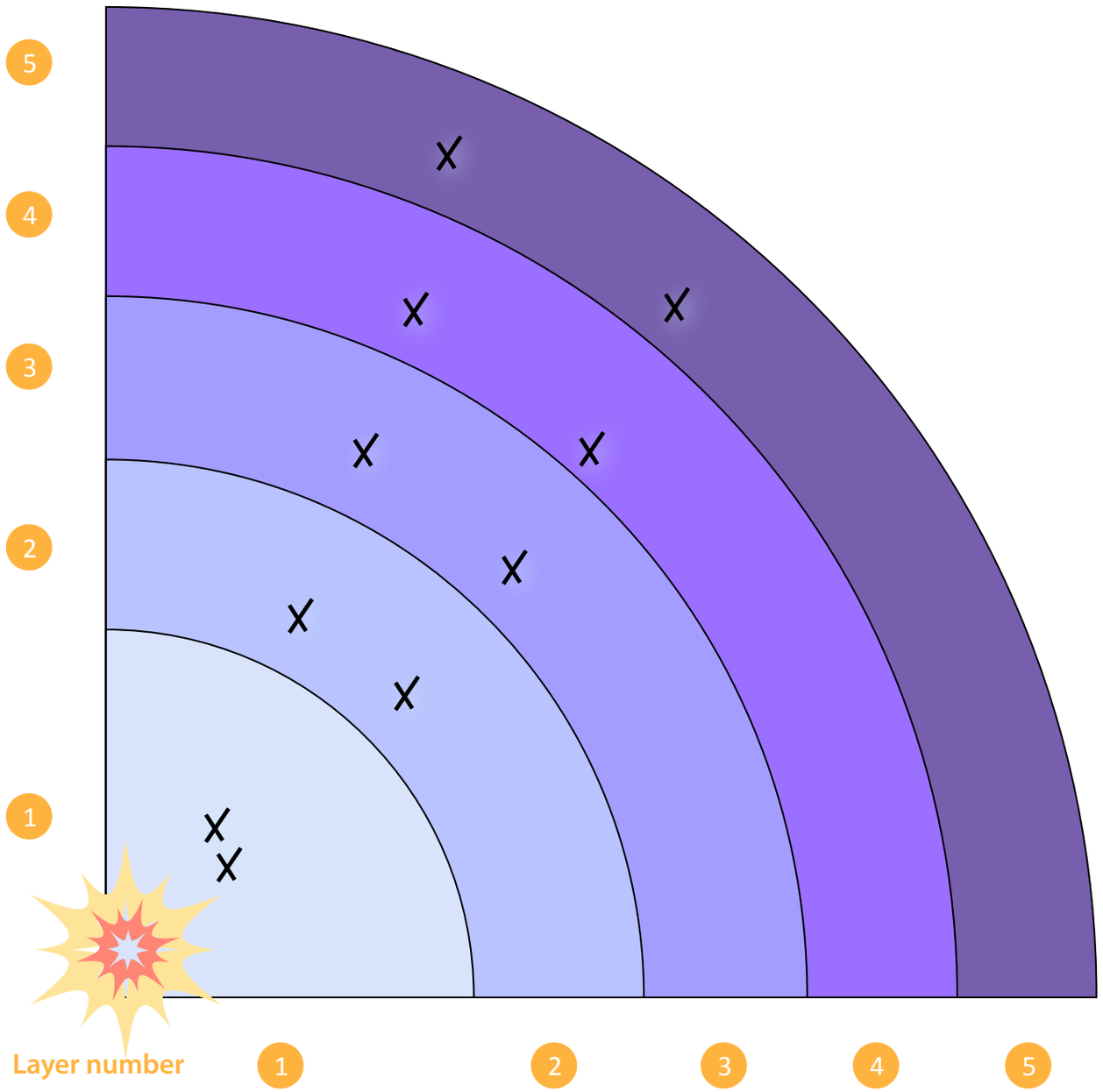
Riemann Track Finder — 1 Seeds



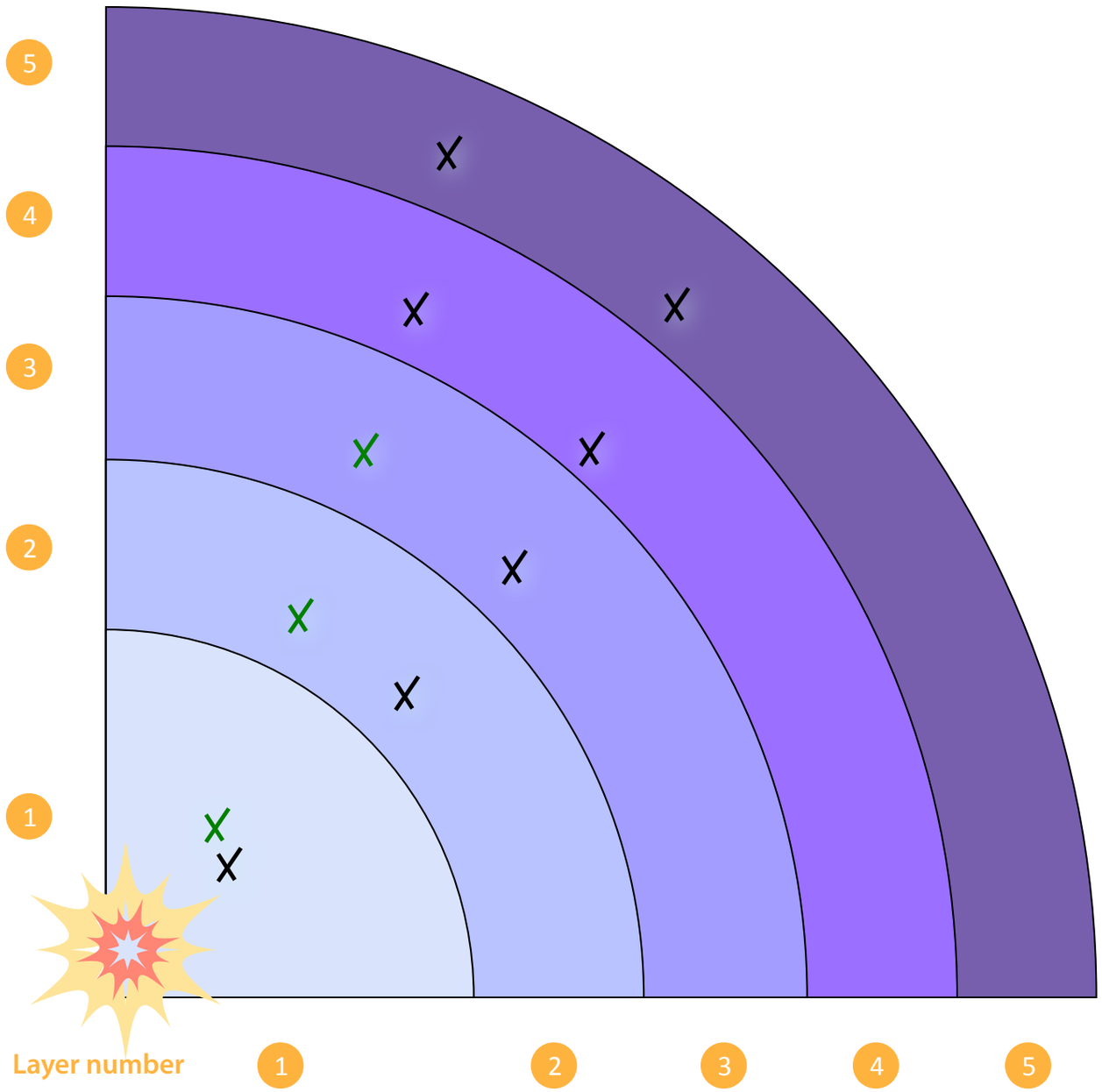
Riemann Track Finder — 1 Seeds



Riemann Track Finder — 1 Seeds

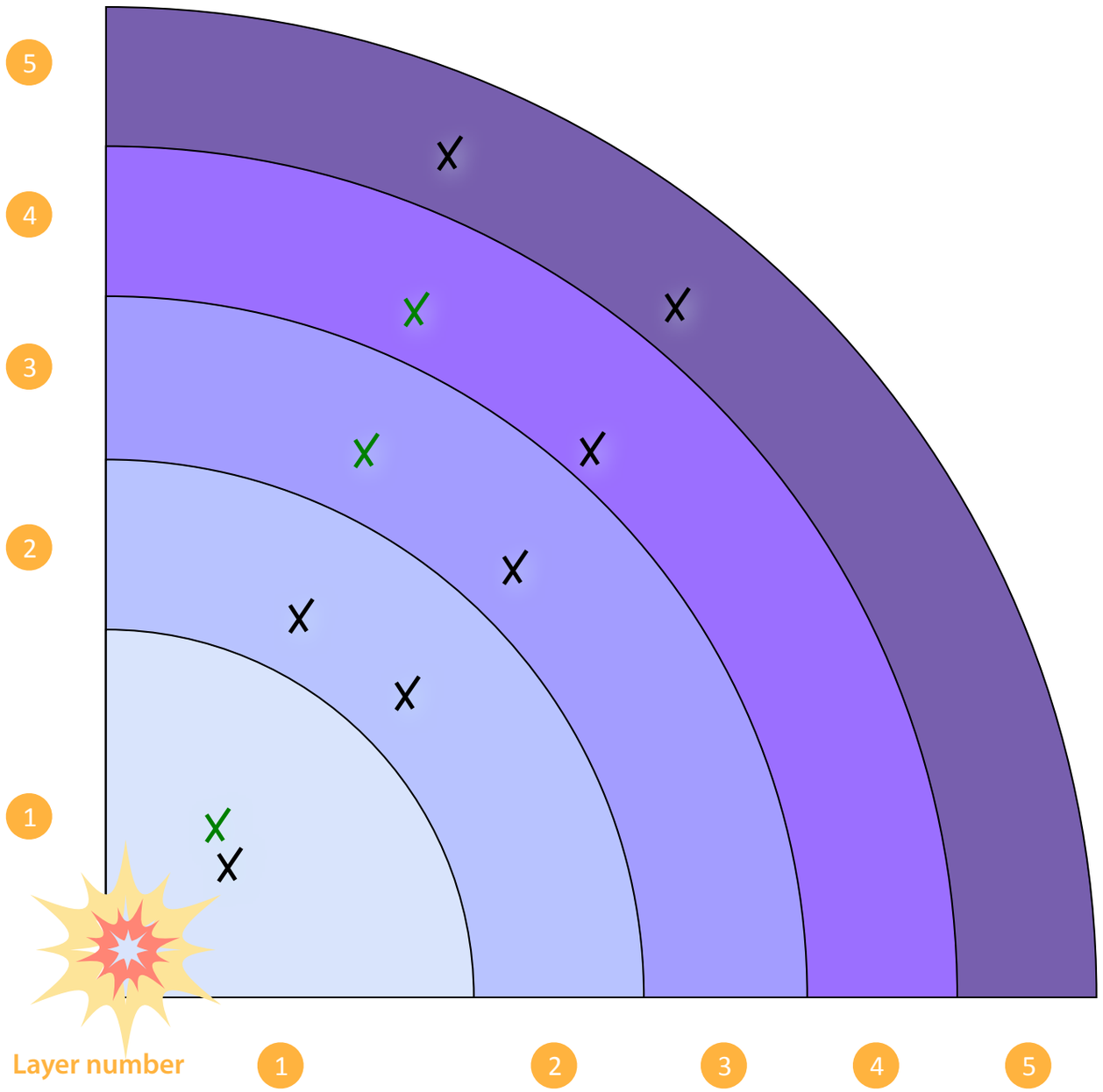


Riemann Track Finder — 1 Seeds



1_1 2_1 3_1

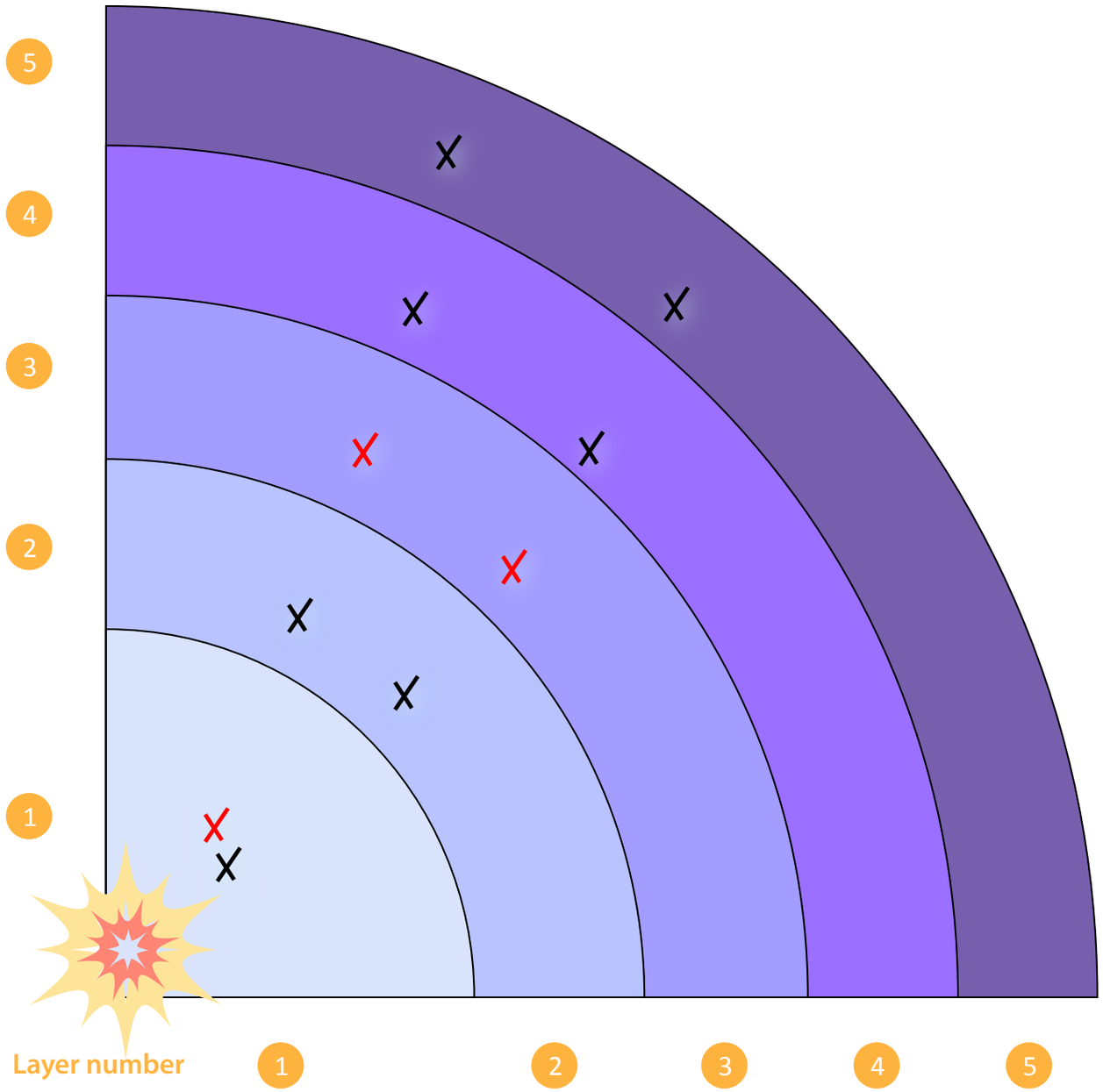
Riemann Track Finder — 1 Seeds



1_1 2_1 3_1

1_1 3_1 4_1

Riemann Track Finder — 1 Seeds

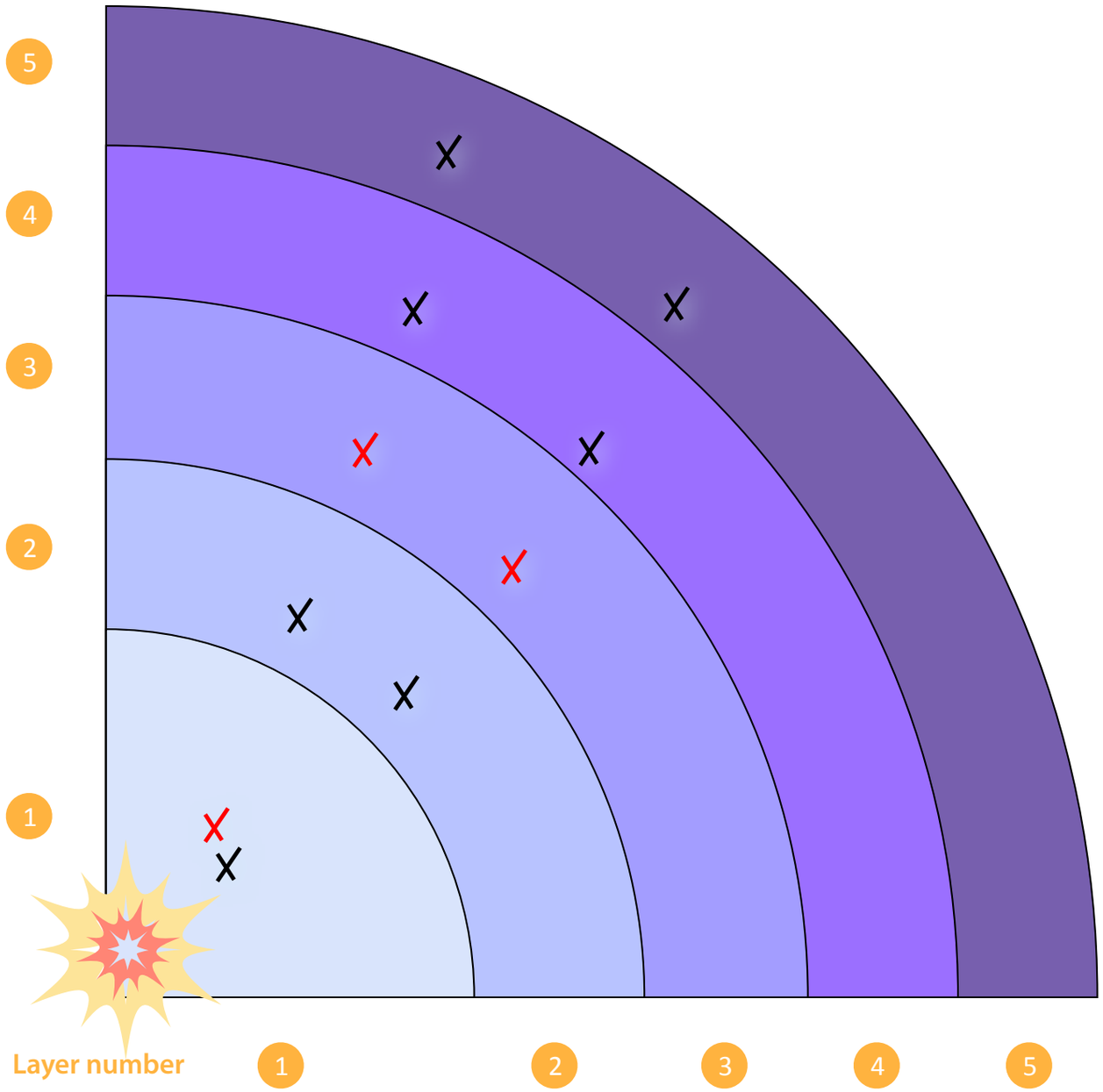


1_1 2_1 3_1

1_1 3_1 4_1

1_1 3_1 3_2

Riemann Track Finder — 1 Seeds



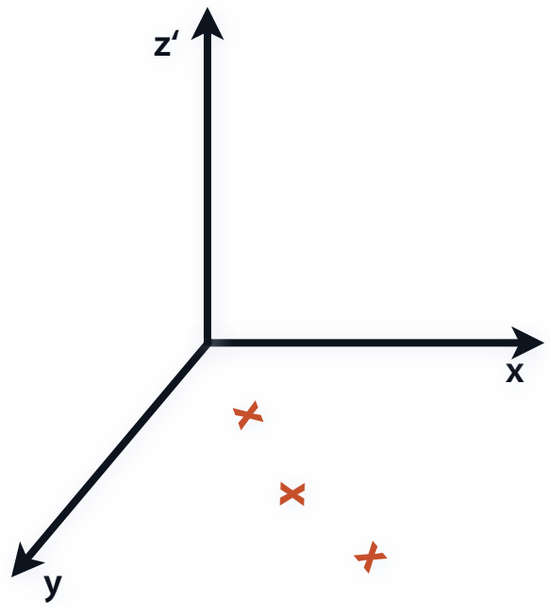
1_1 2_1 3_1

1_1 3_1 4_1

1_1 3_1 3_2

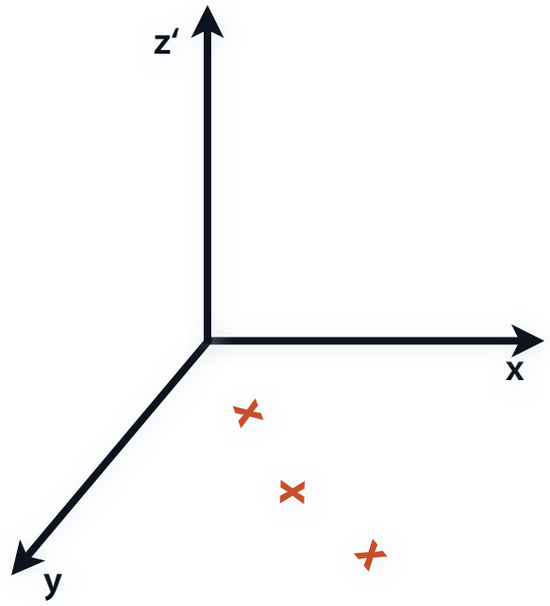
Riemann Algorithm — 2 Expansion

Riemann Algorithm — 2 Expansion

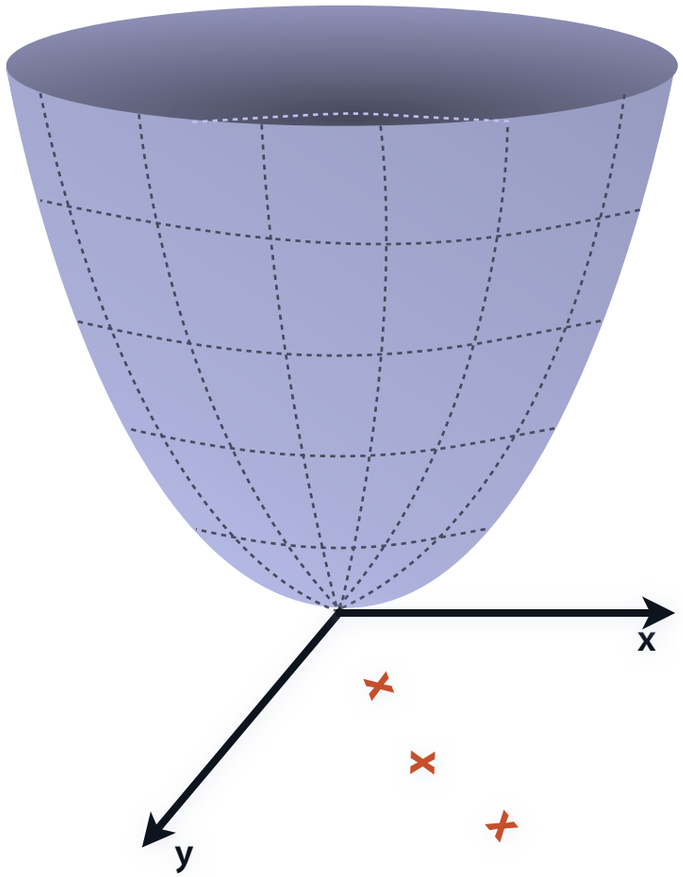


Expand to z‘

Riemann Algorithm — 2 Expansion

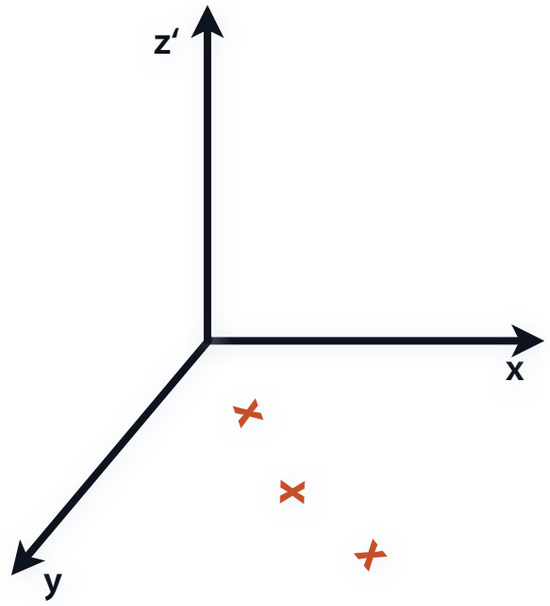


Expand to z‘

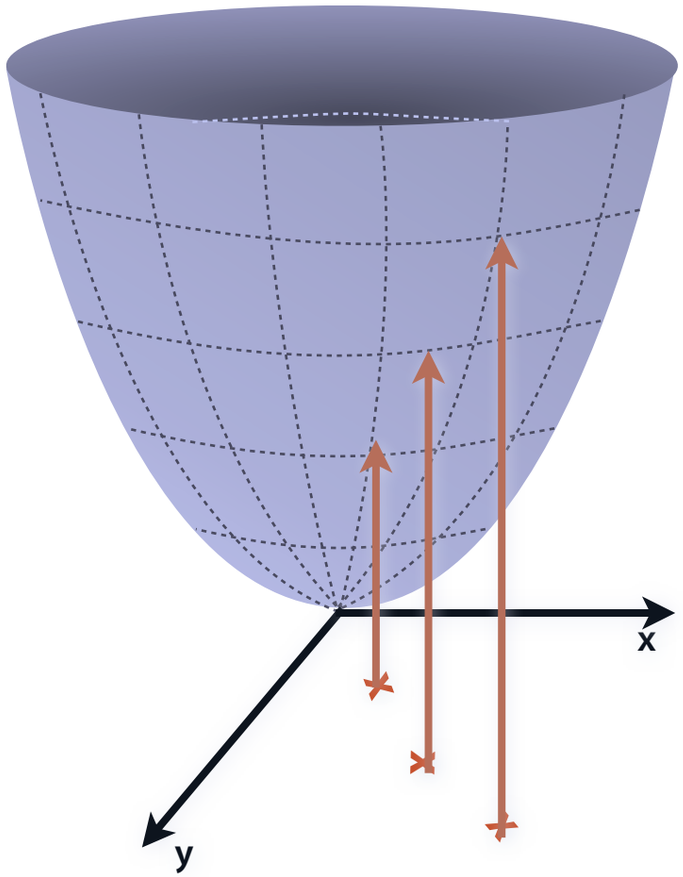


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

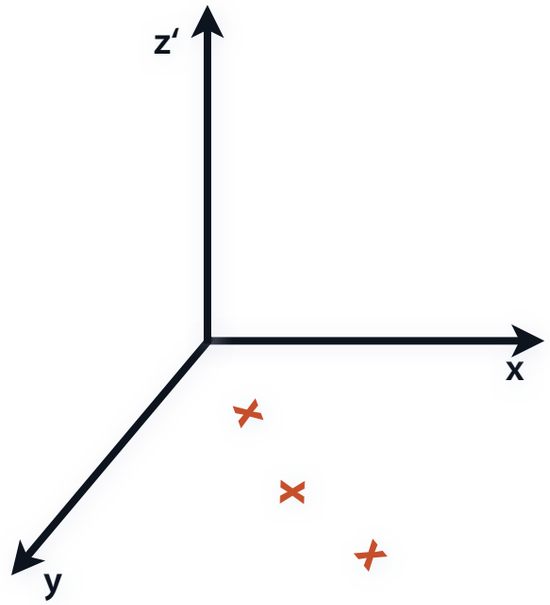


Expand to z‘

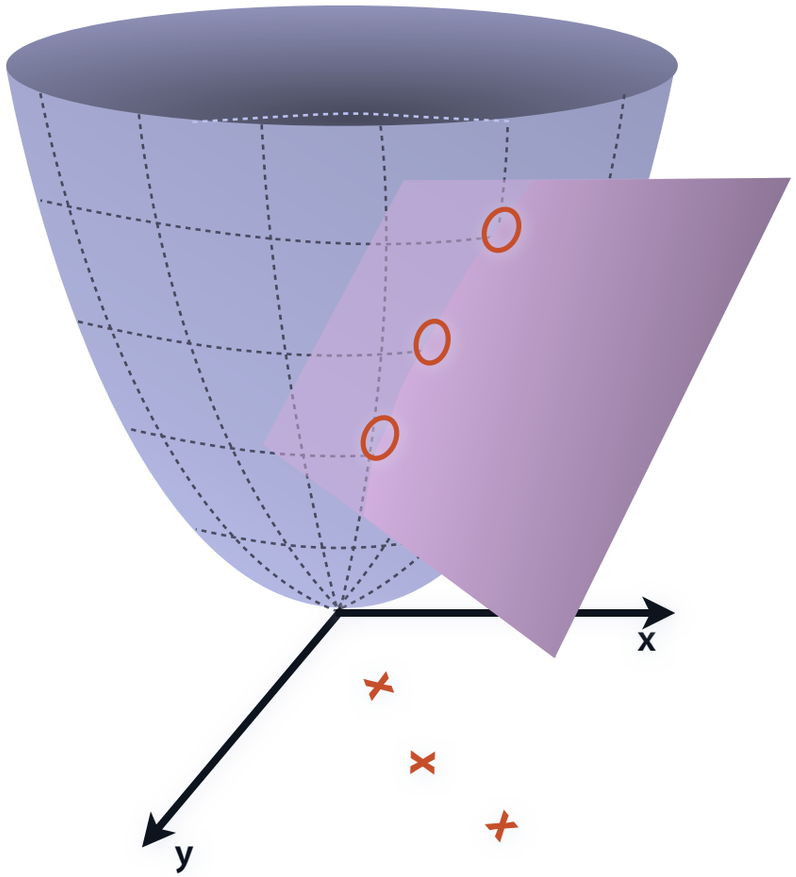


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

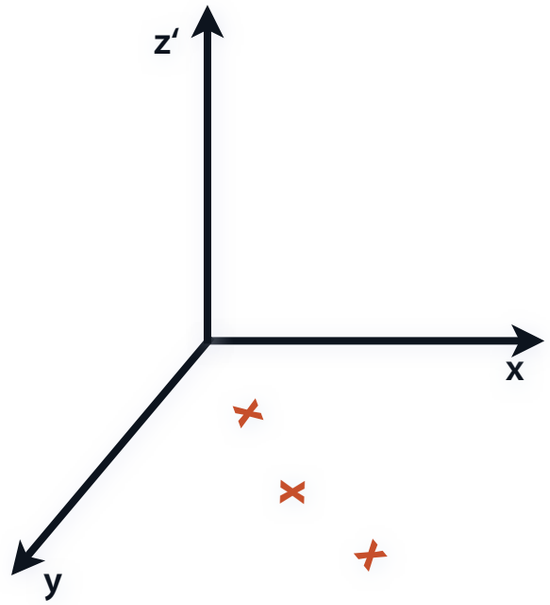


Expand to z‘

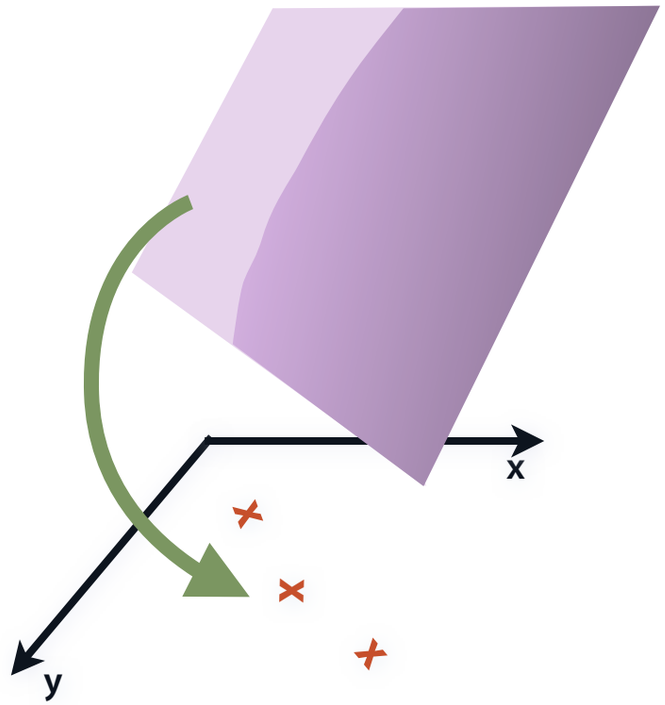


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

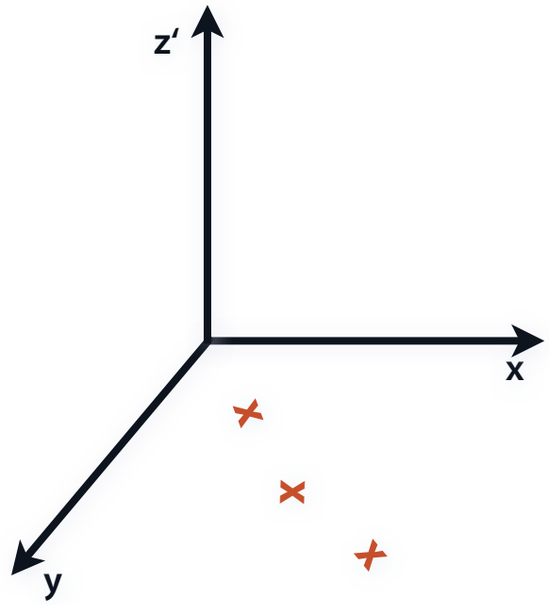


Expand to z‘

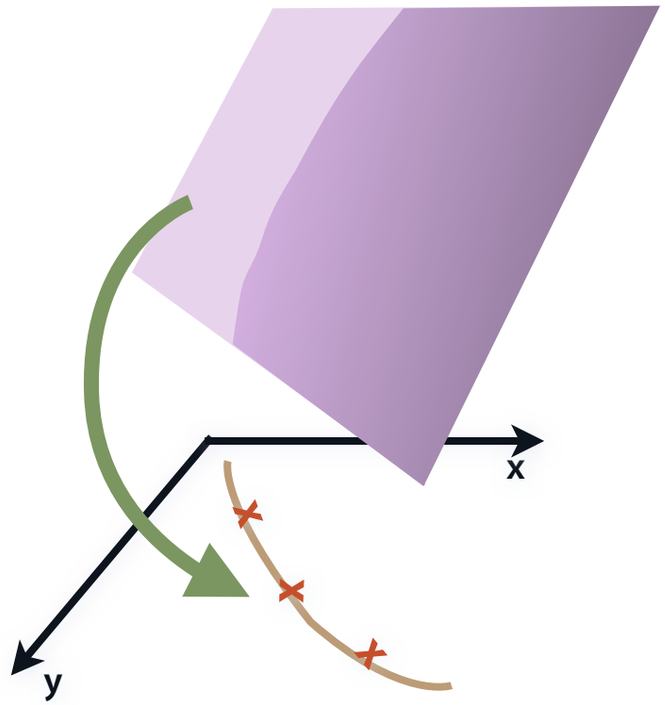


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

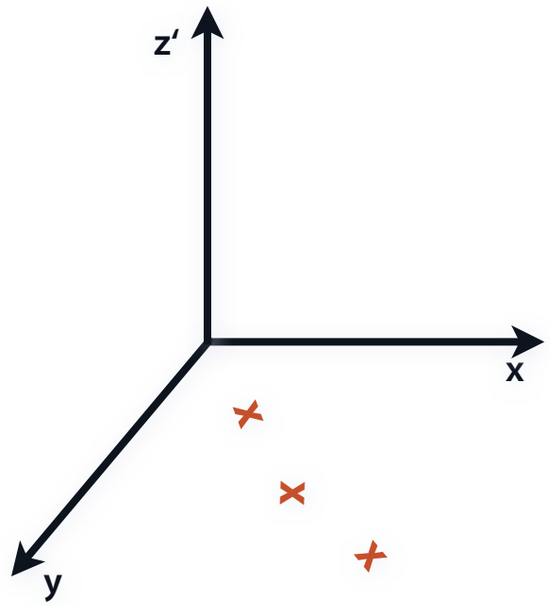


Expand to z‘

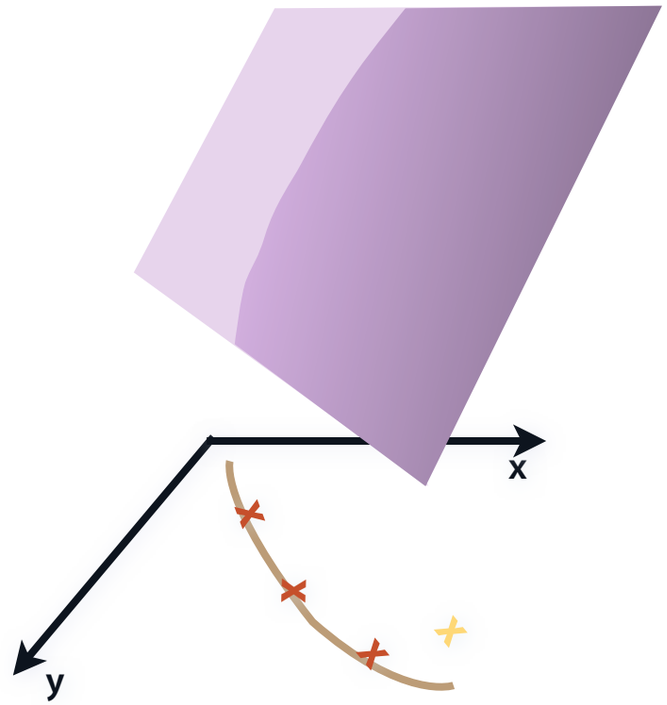


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

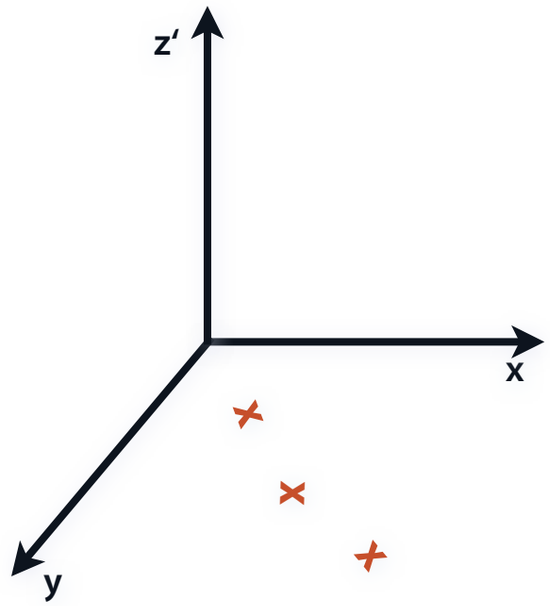


Expand to z‘

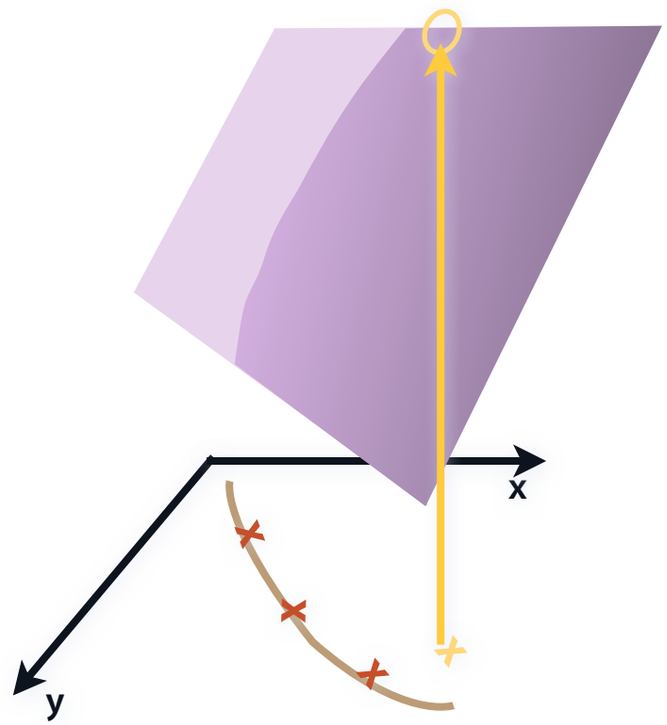


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

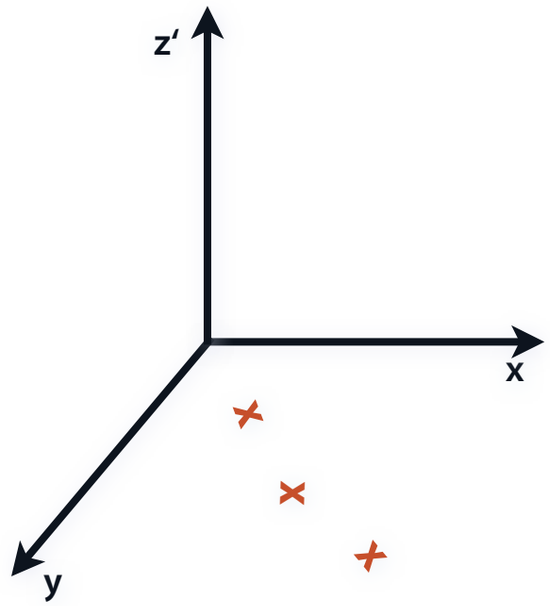


Expand to z‘

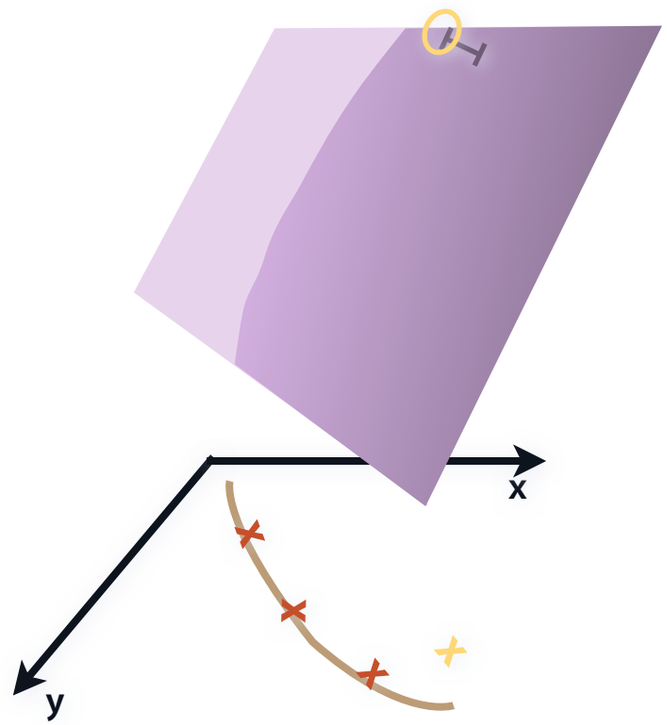


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

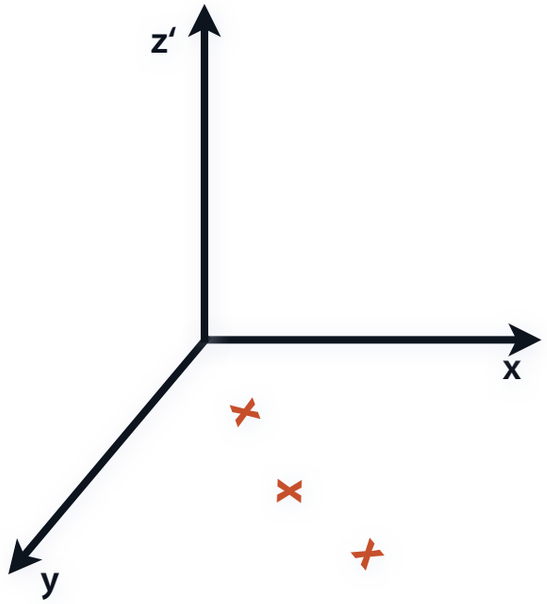


Expand to z‘

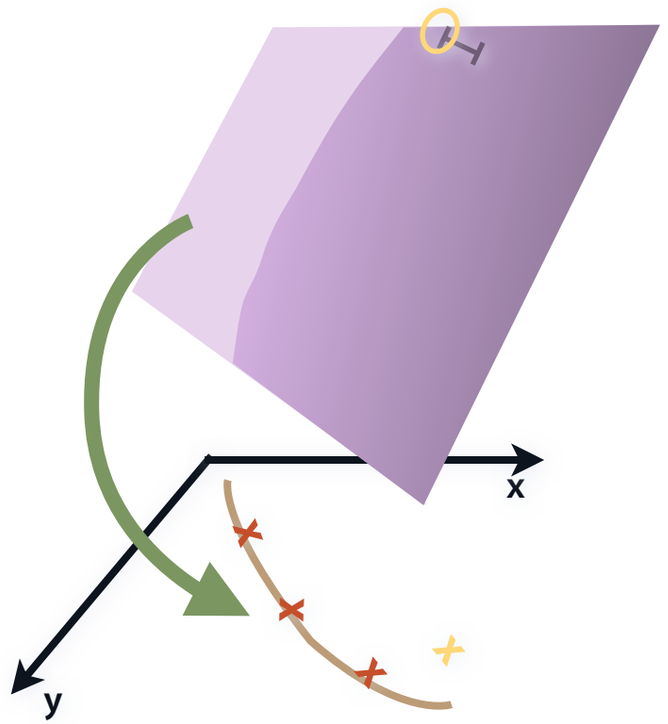


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

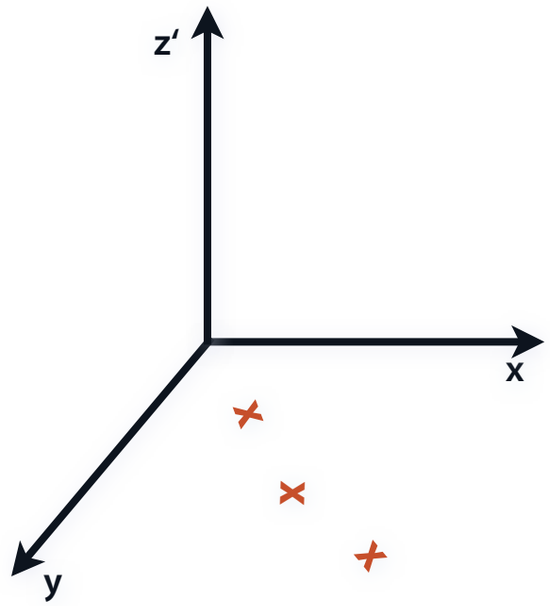


Expand to z‘

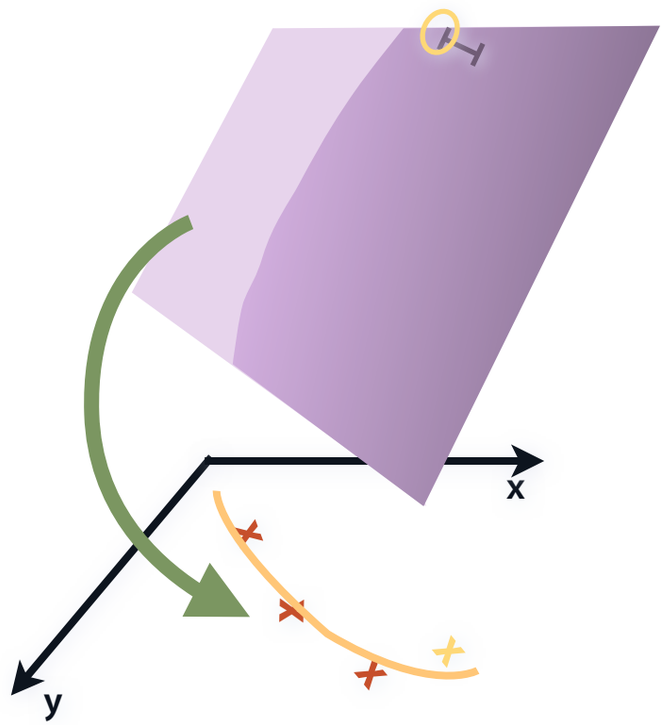


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion



Expand to z‘



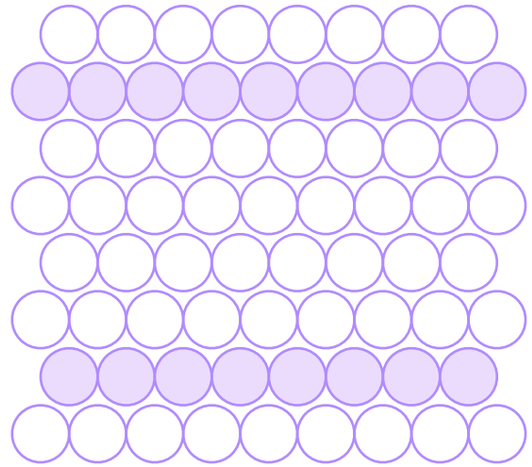
Riemann Surface
(paraboloid)

Triplet Finder — Method

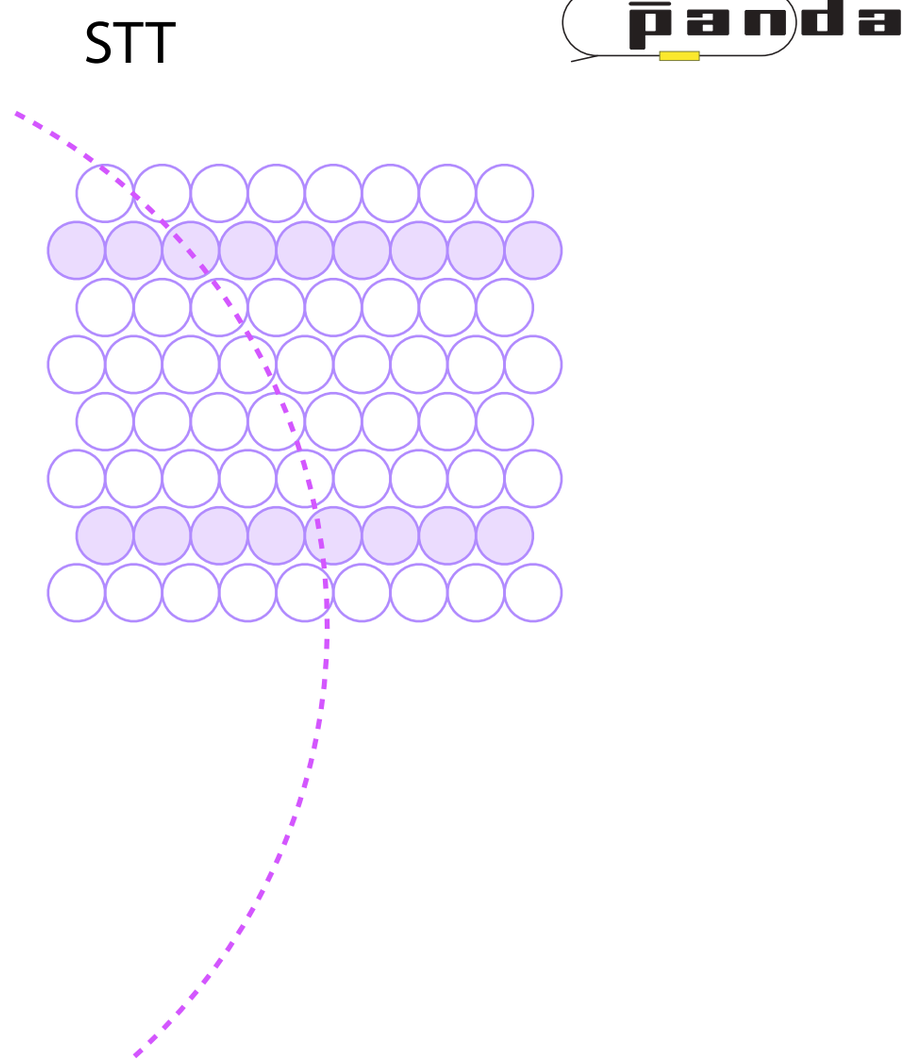
JÜLICH FORSCHUNGSZENTRUM

panda

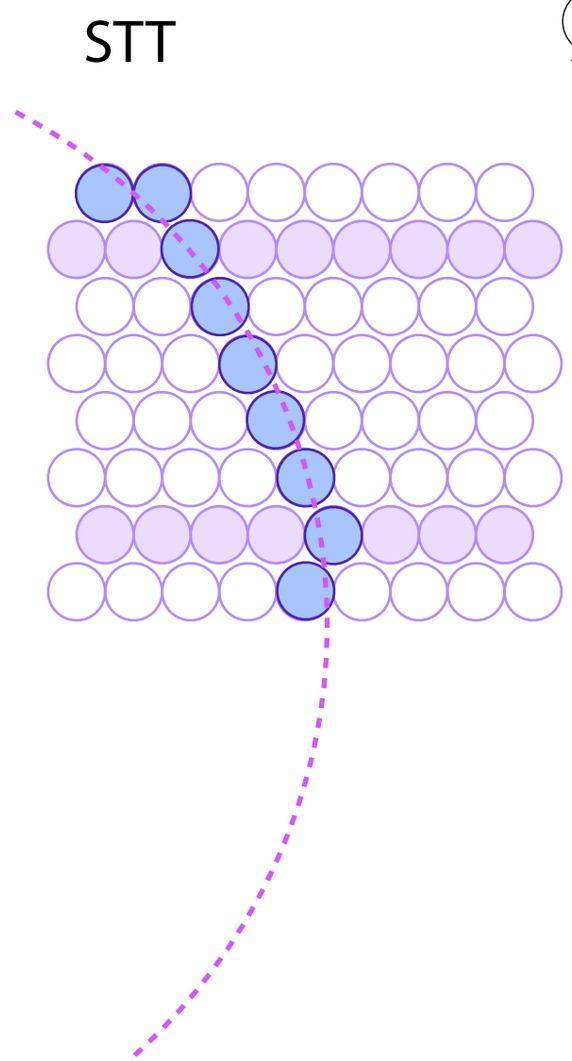
STT



Triplet Finder — Method

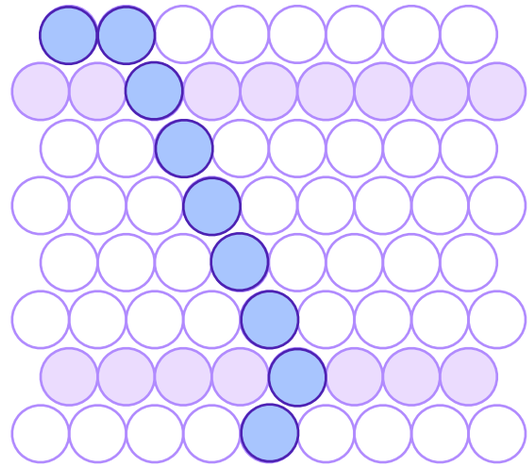


Triplet Finder — Method



Triplet Finder — Method

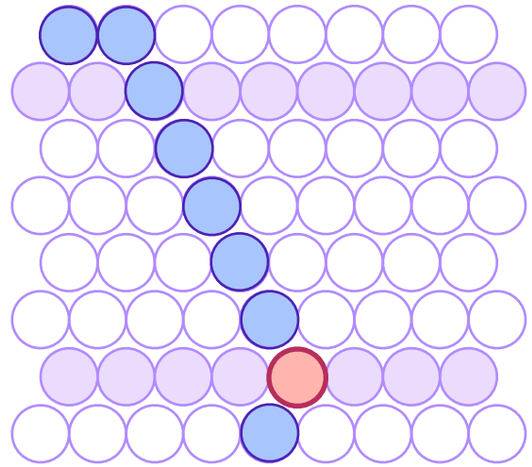
STT



Triplet Finder — Method

- STT hit in **pivot** straw

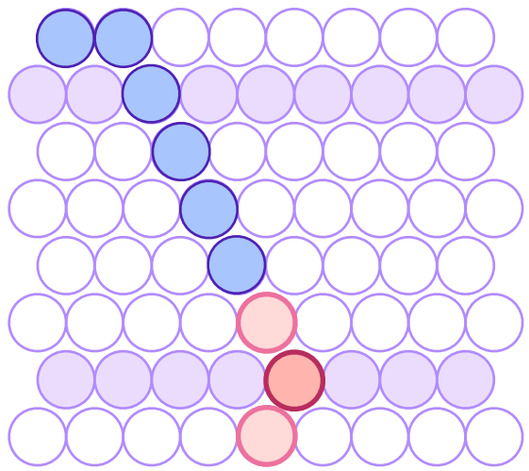
STT



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

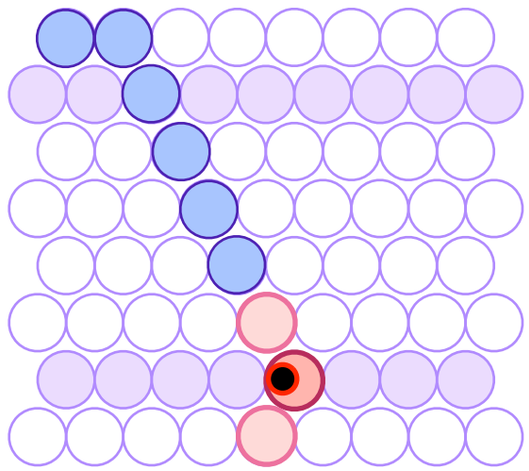
  → Create **virtual hit** *(triplet)* at center of gravity *(**cog**)*



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity *(**cog**)*
- **Combine** with

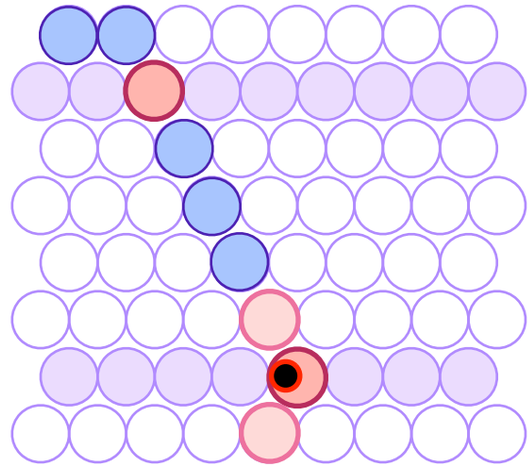


Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



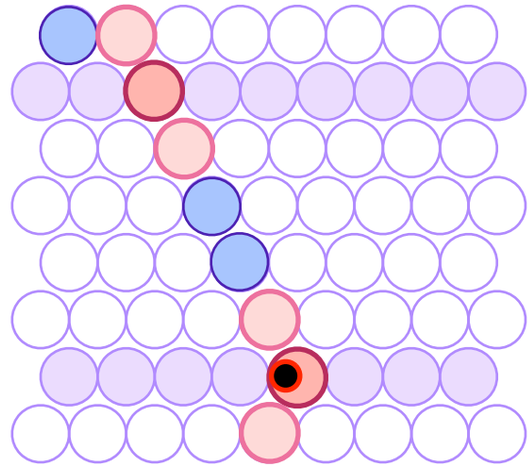
1. Second STT pivot-*cog* virtual hit

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



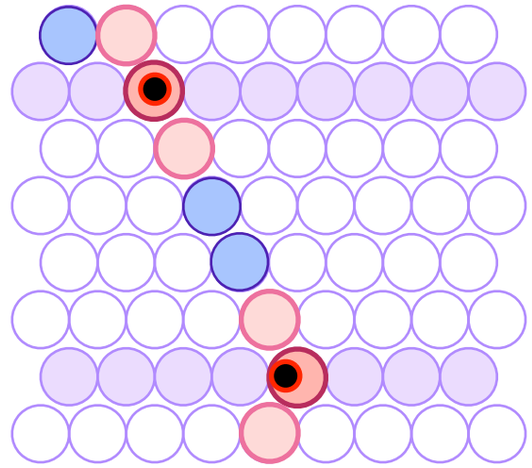
1. Second STT pivot-*cog* virtual hit

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)*
  at center of gravity ***(cog)***
- **Combine** with

STT



1. Second STT pivot-*cog* virtual hit

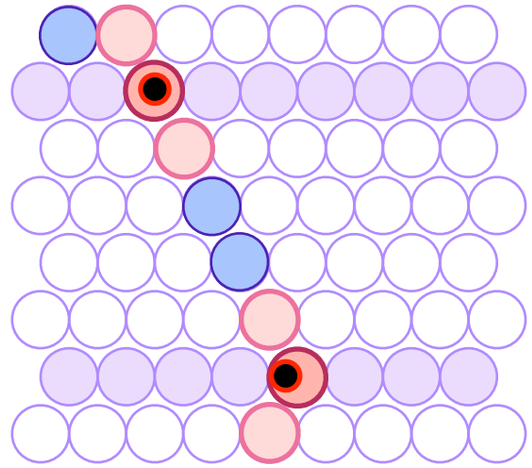
Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

1. Second STT pivot-*cog* virtual hit
2. Interaction point

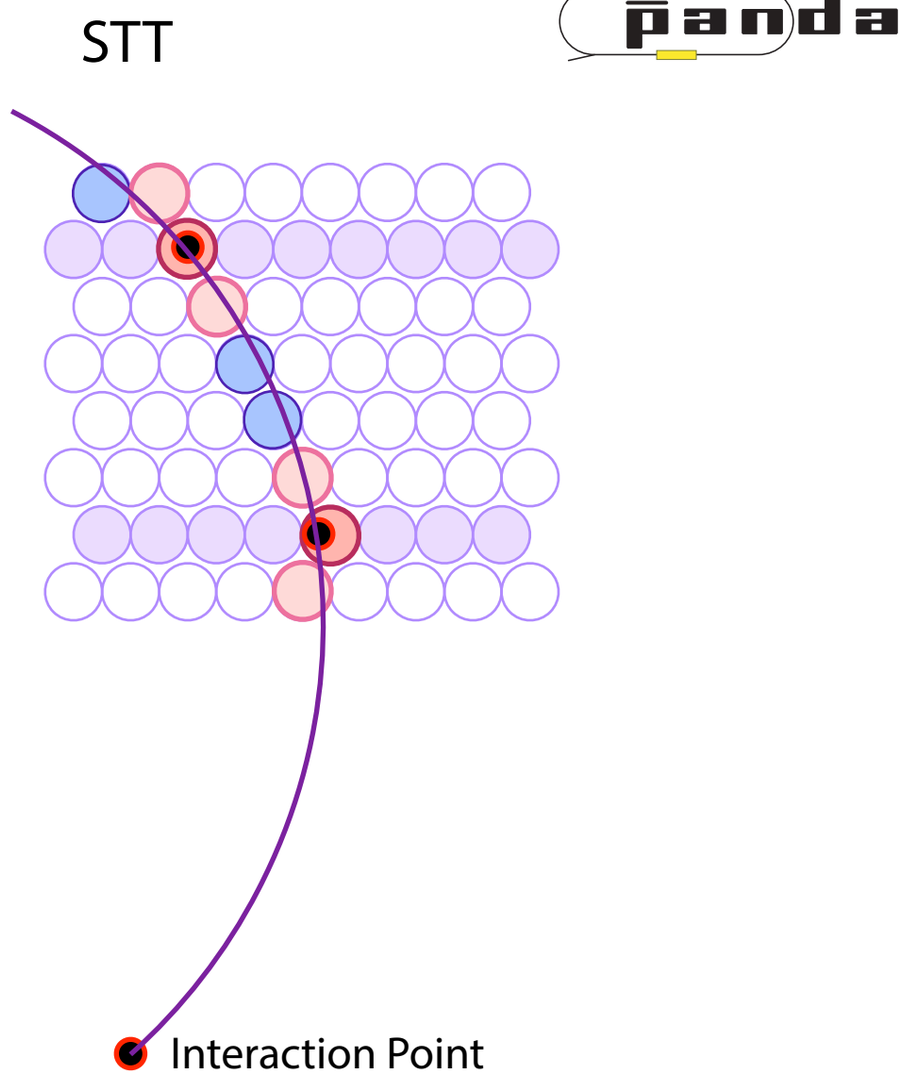
STT



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with
  1. Second STT pivot-*cog* virtual hit
  2. Interaction point
- Calculate **circle** through three points

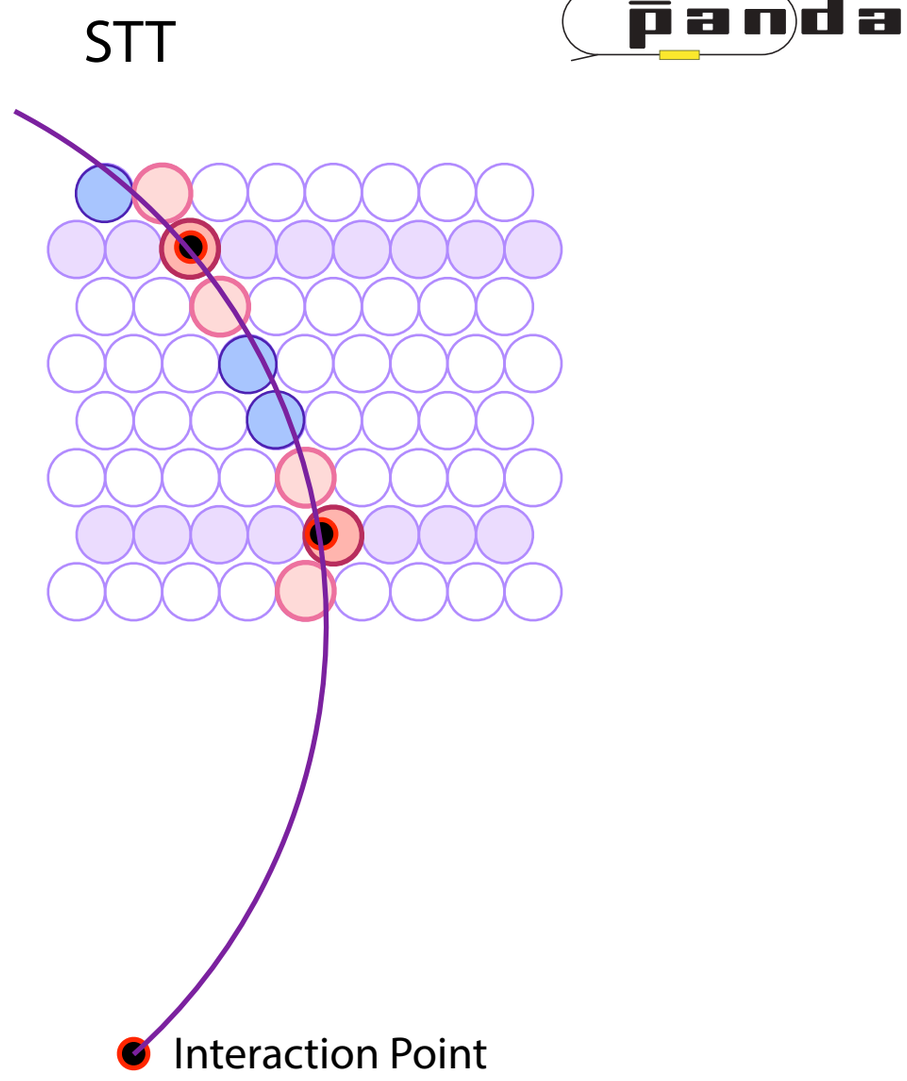


Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

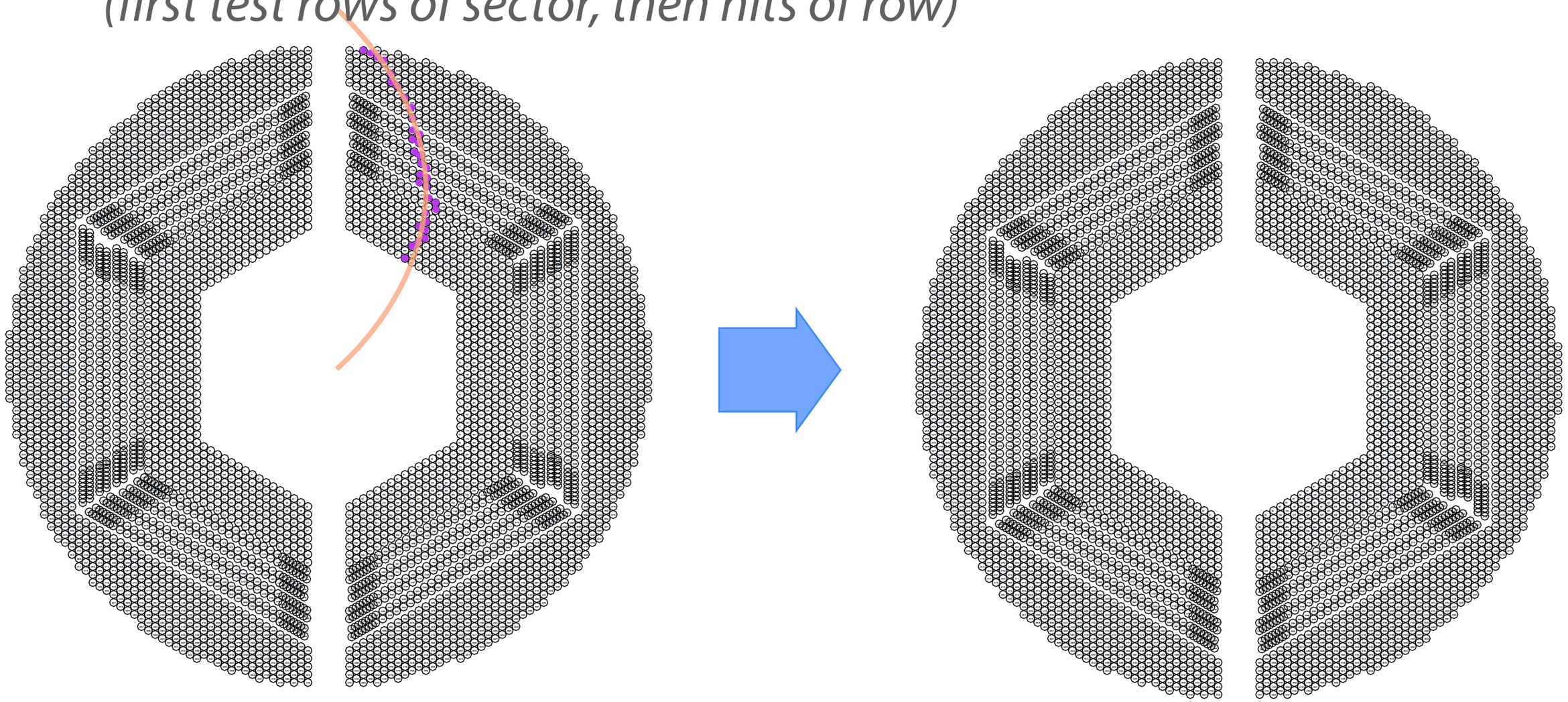
  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with
  1. Second STT pivot-*cog* virtual hit
  2. Interaction point
- Calculate **circle** through three points

→ **Track** Candidate



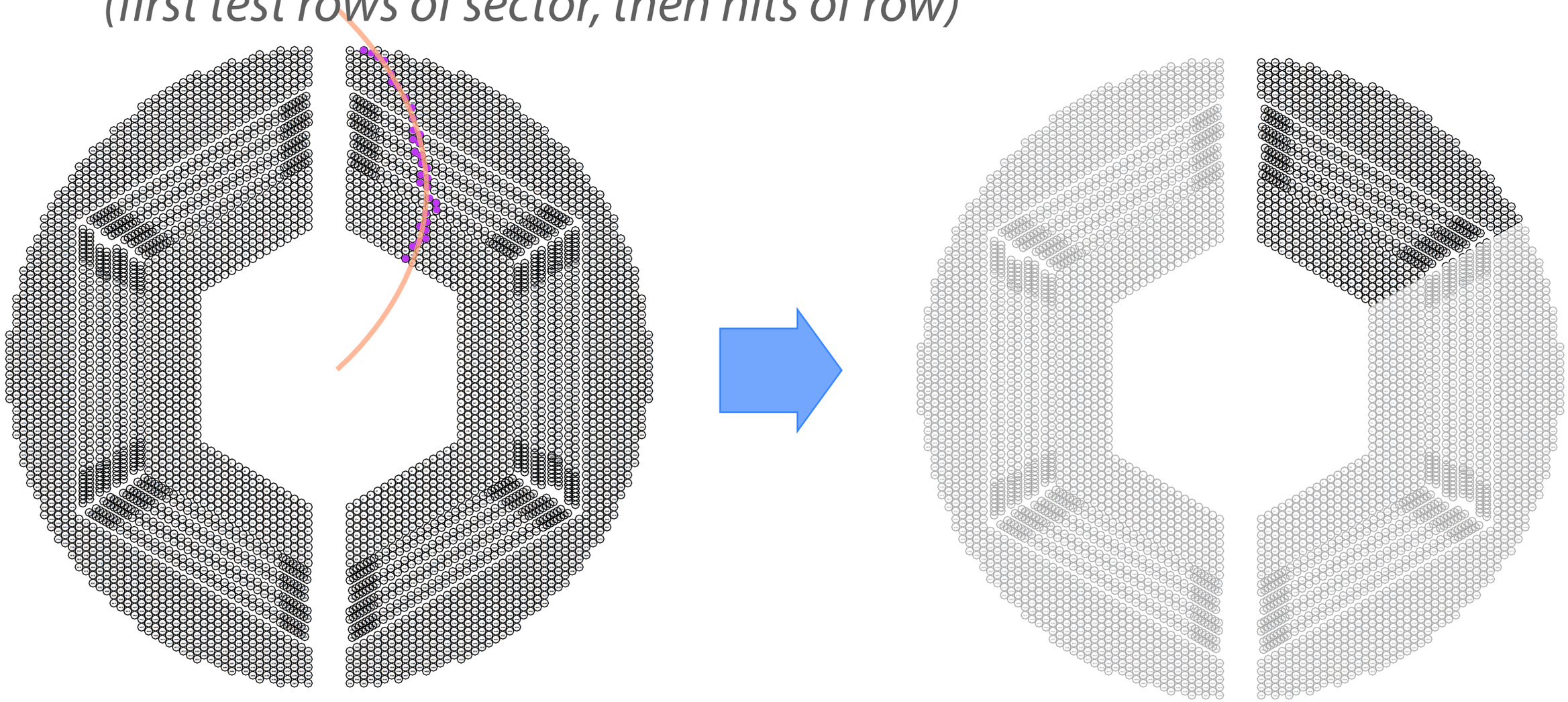
Triplet Finder — Optimizations

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



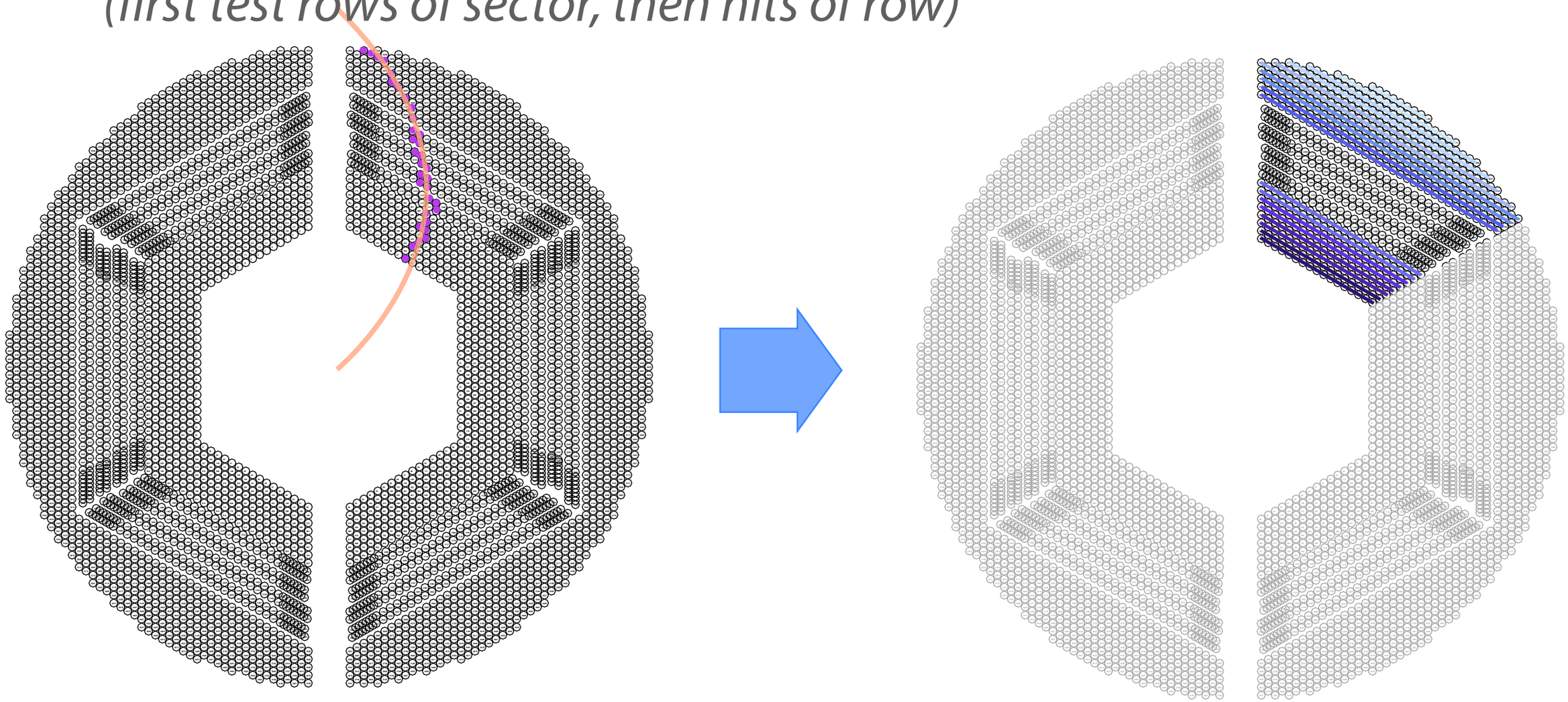
Triplet Finder — Optimizations

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



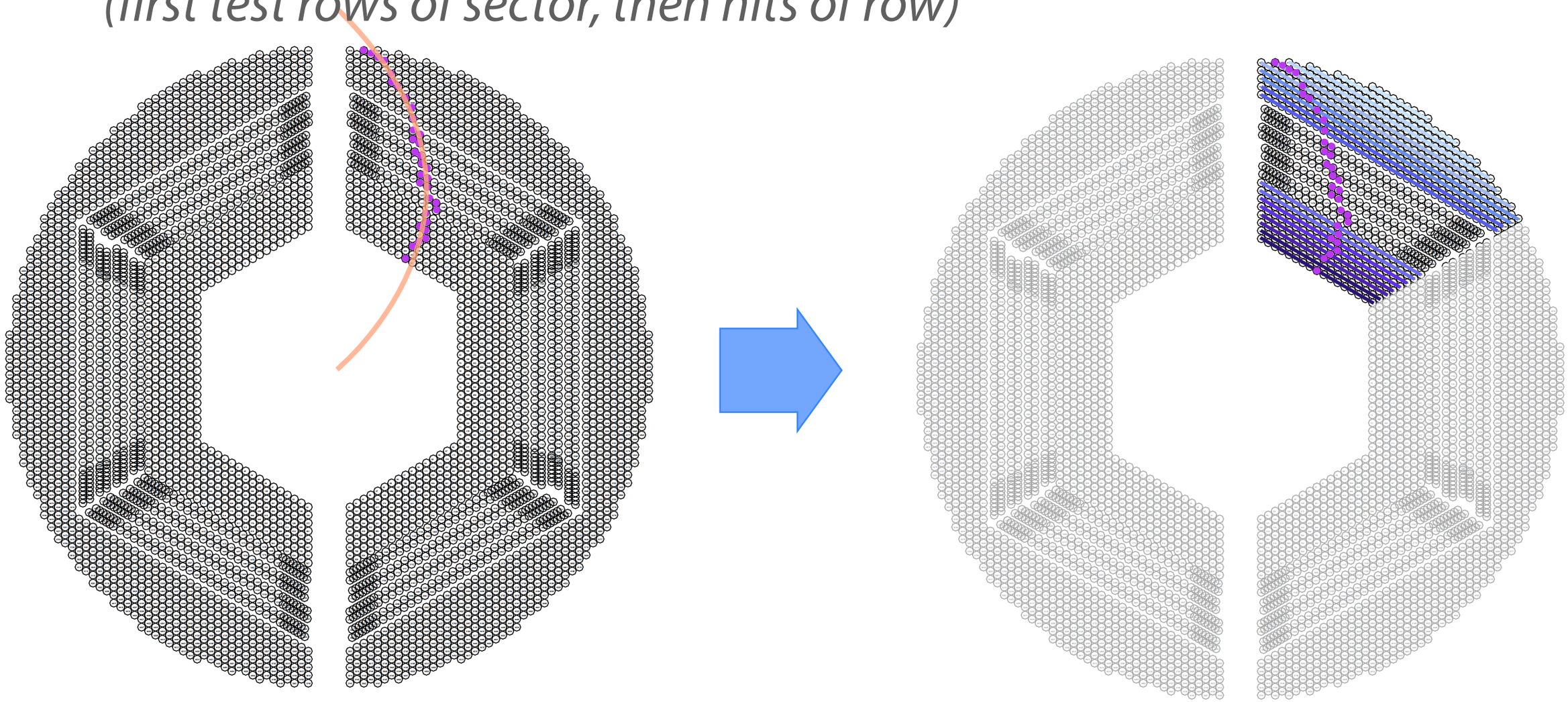
Triplet Finder — Optimizations

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



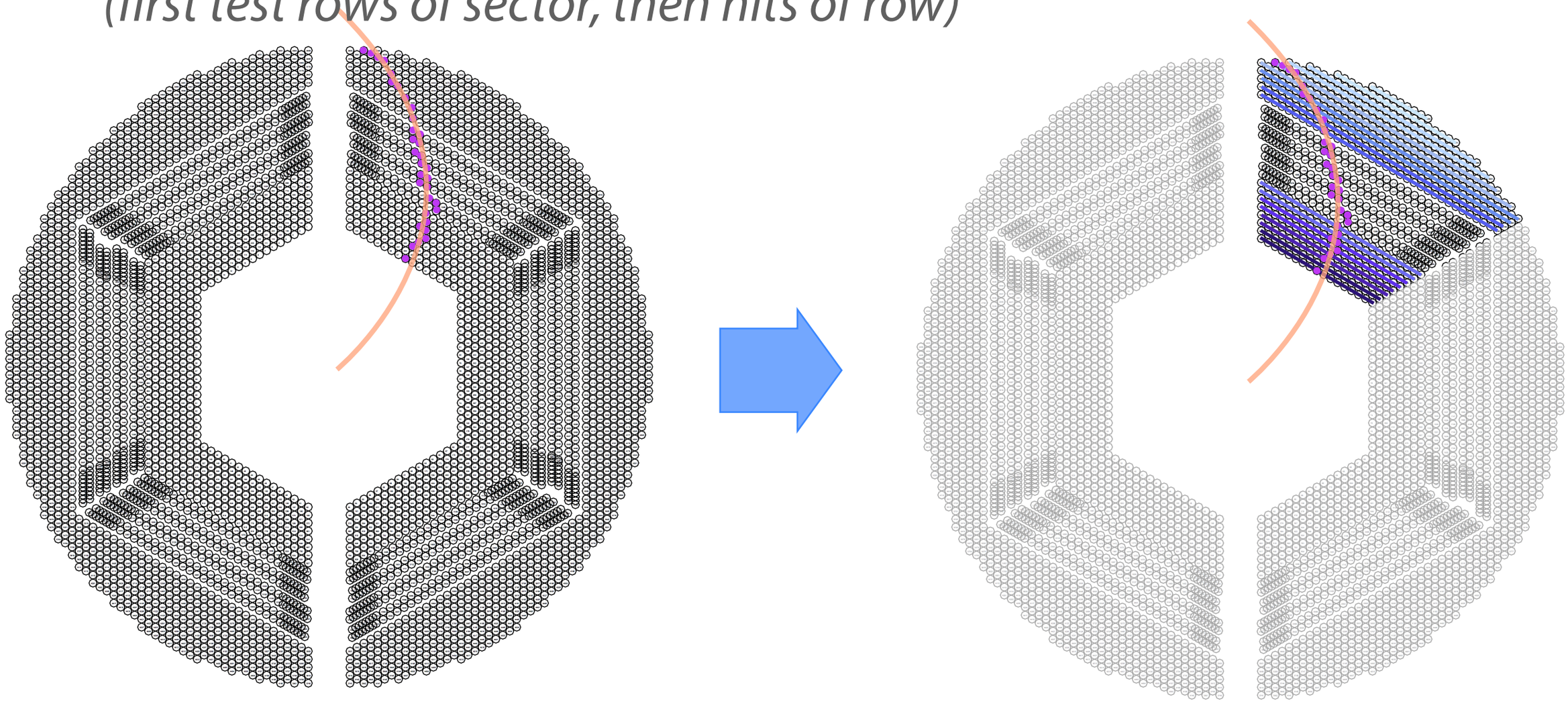
Triplet Finder — Optimizations

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



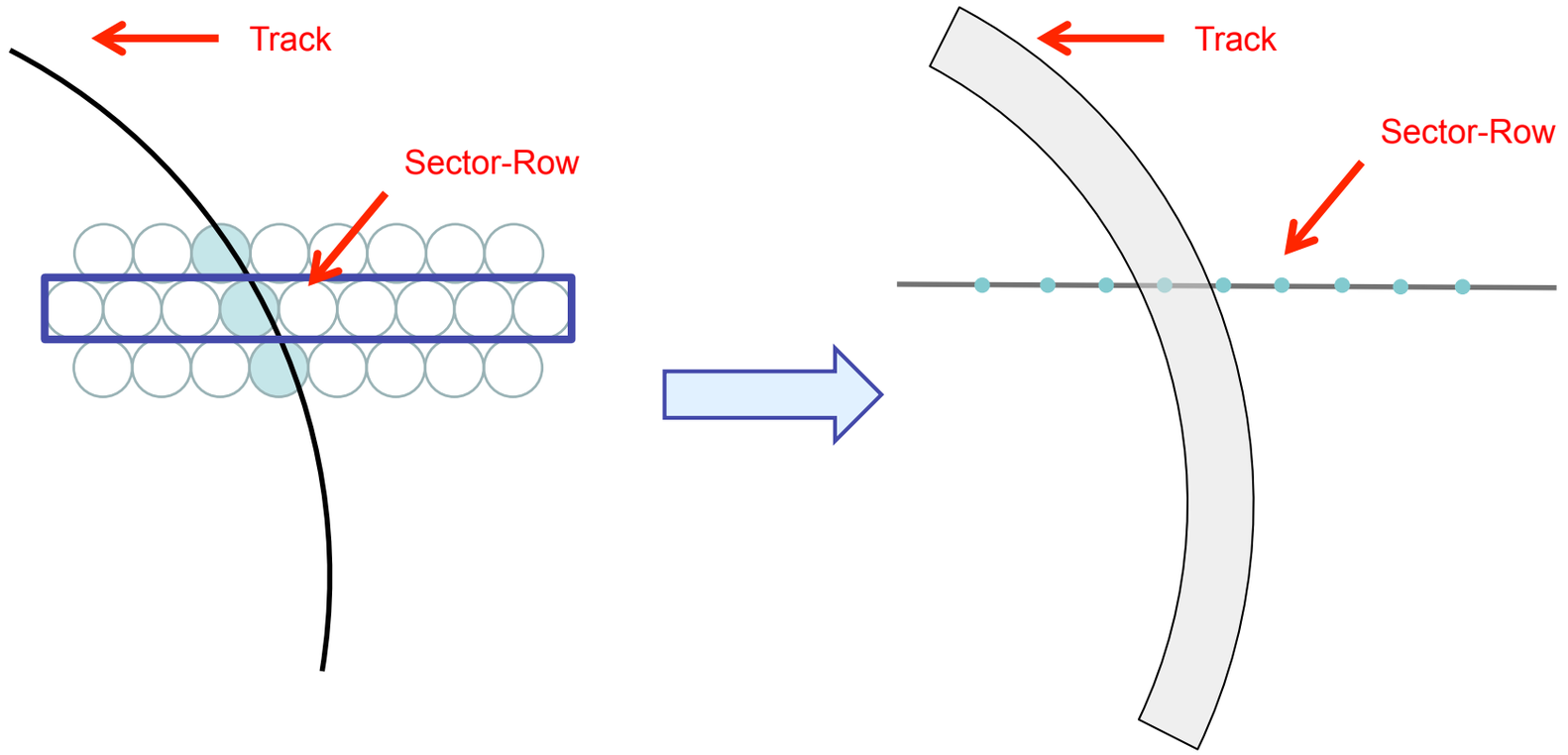
Triplet Finder — Optimizations

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)

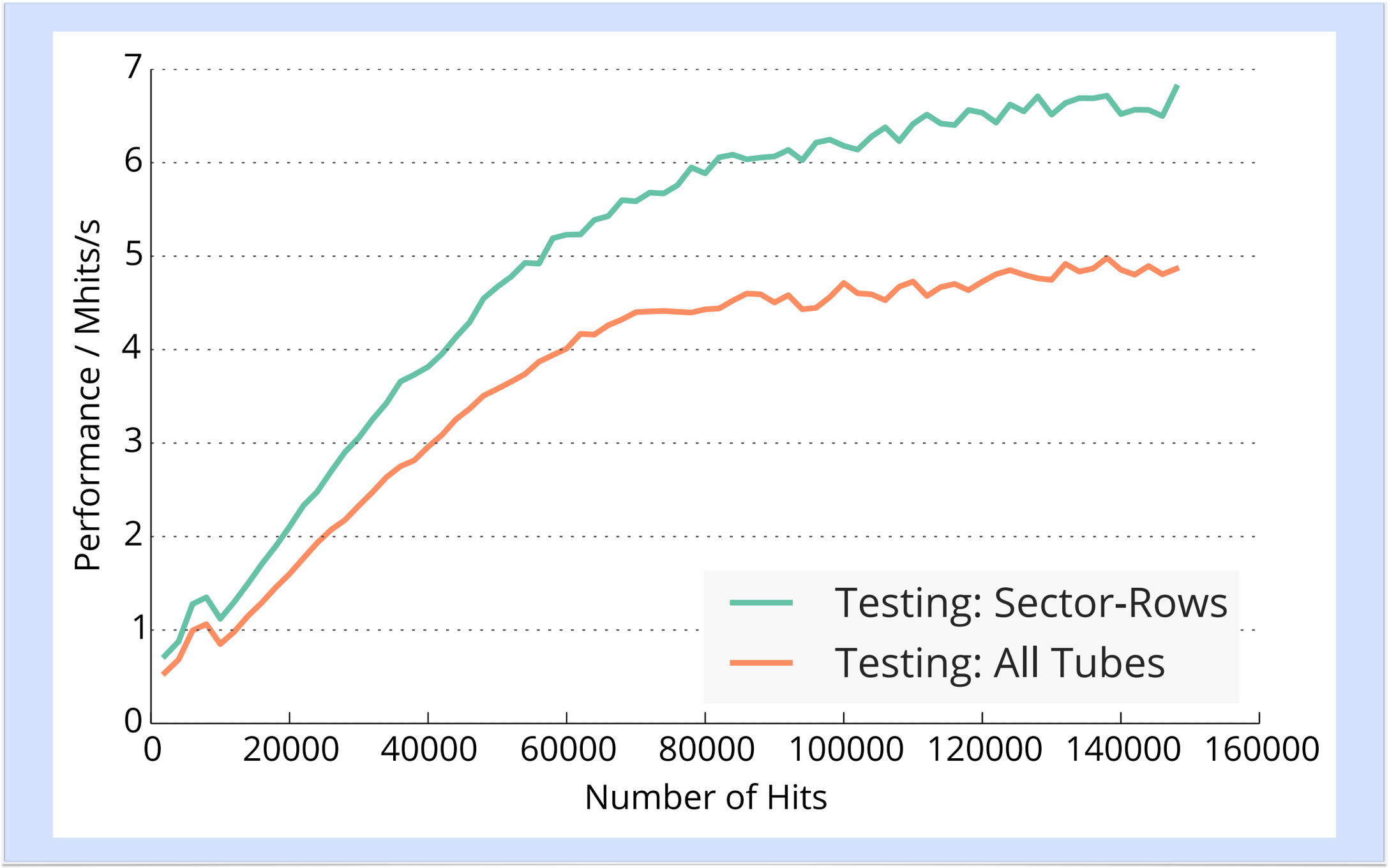


Triplet Finder — Optimizations

- Sector Row testing
  - Thicken track; shrink sector row layer to line
  - Find intersection



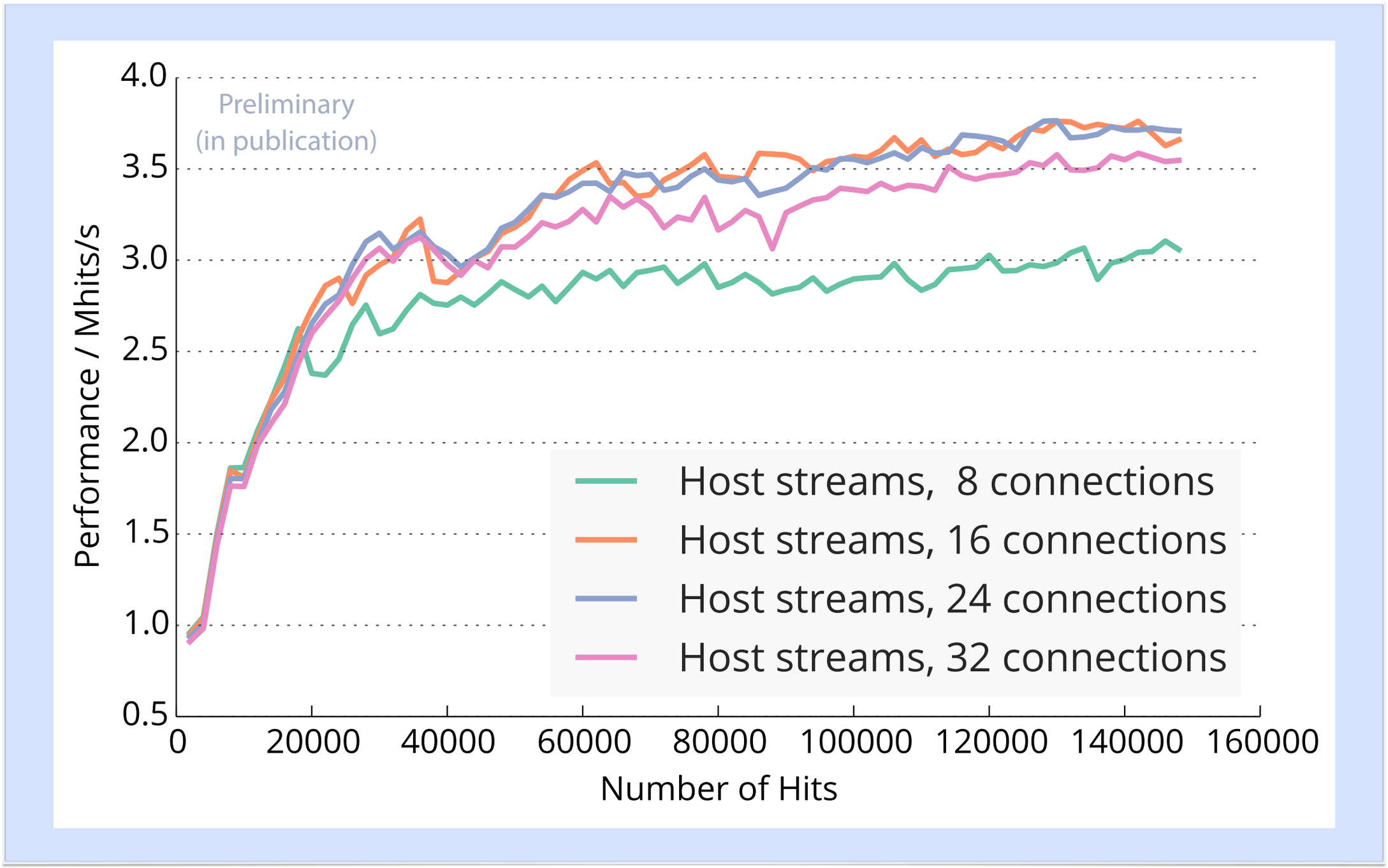
Triplet Finder — Sector Rows



Triplet Finder — Kernel Launch Strategies

- Joined Kernel *(JK)*: slowest
  - High # registers → low occupancy
- Dynamic Parallelism *(DP)* / Host Streams *(HS)*: comparable performance
  - Performance
    - HS faster for small # processed hits, DP faster for > 45000 hits
    - HS stagnates there, while DP continues rising
  - Limiting factor
    - High # of required kernel calls
    - Kernel launch latency
    - Memcopy
  - HS more affected by this, because
    - More PCI-E transfers (launch configurations for kernels)
    - Less launch throughput, kernel launch latency gets more important
    - False dependencies of launched kernels
      - Single CPU thread handles all CUDA streams (Multi-thread possible, but synchronization overhead too high for good performance)
      - Grid scheduling done on hardware (Grid Management Unit) (DP: software)
        - » False dependencies when N(streams) > N(device connections)=32[3.5]

Triplet Finder — Host Stream Connections



Triplet Finder — Bunch Sizes