GPUs for Track Trigger in High Energy Physics

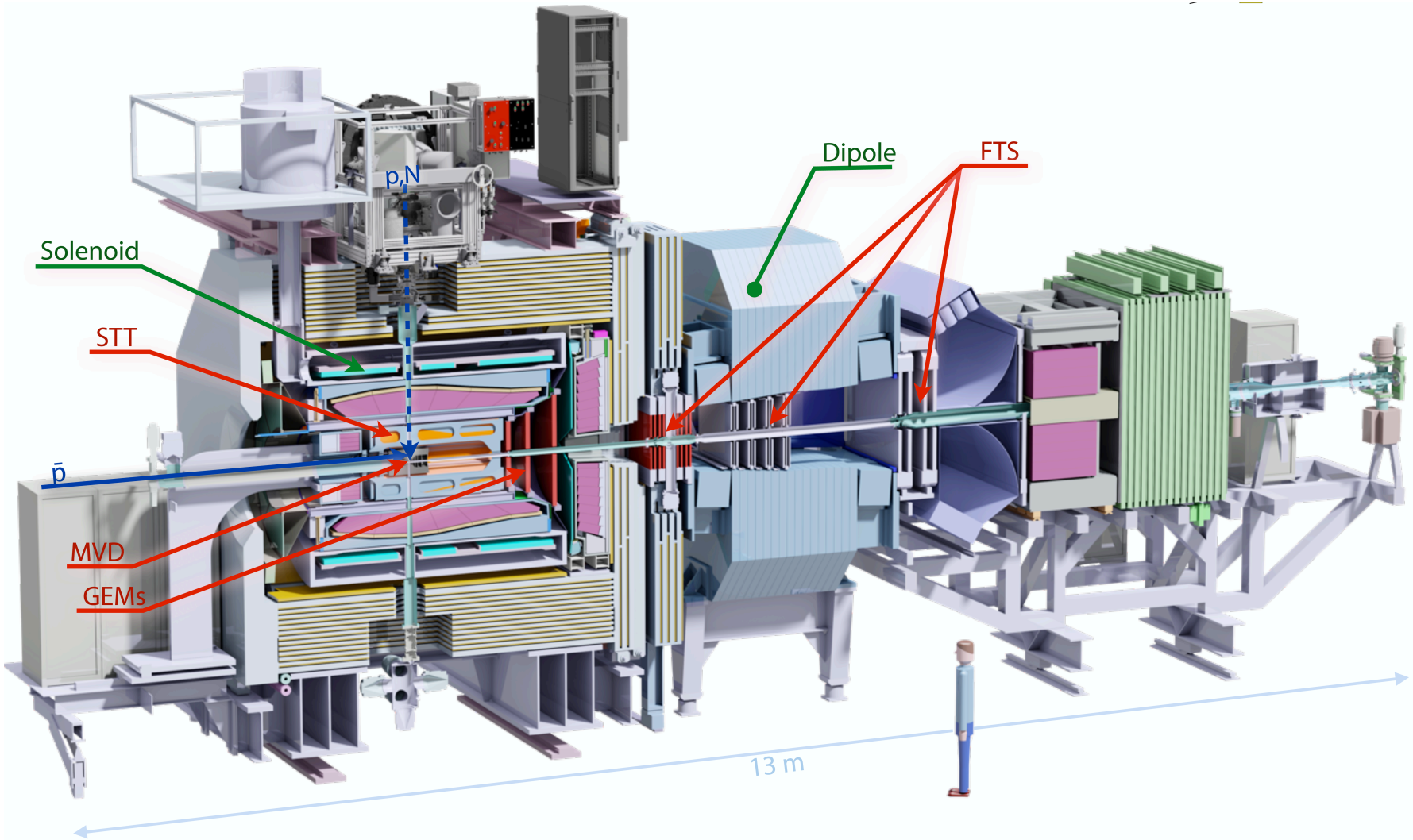
on the example of the PANDA experiment

März 9, 2016 | Tobias Stockmanns

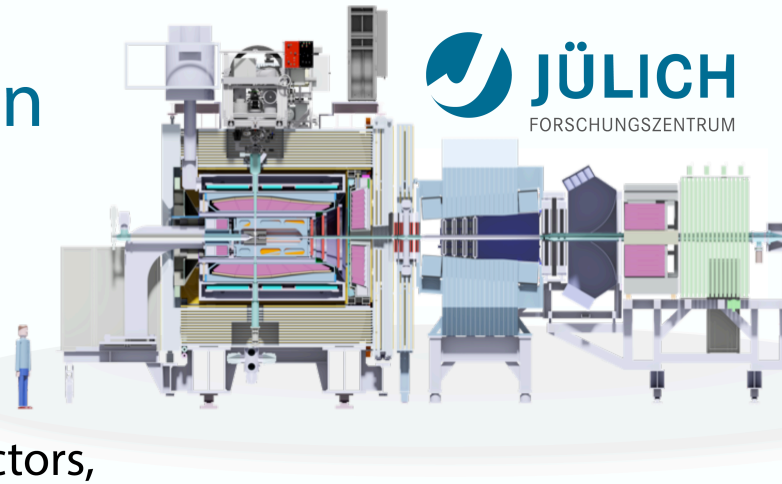
Outline

- Particle physics experiments (example PANDA)
  - Experimental setup
  - Data selection
- Tracking algorithms
  - Hough transformations
  - Cellular automaton
- Summary

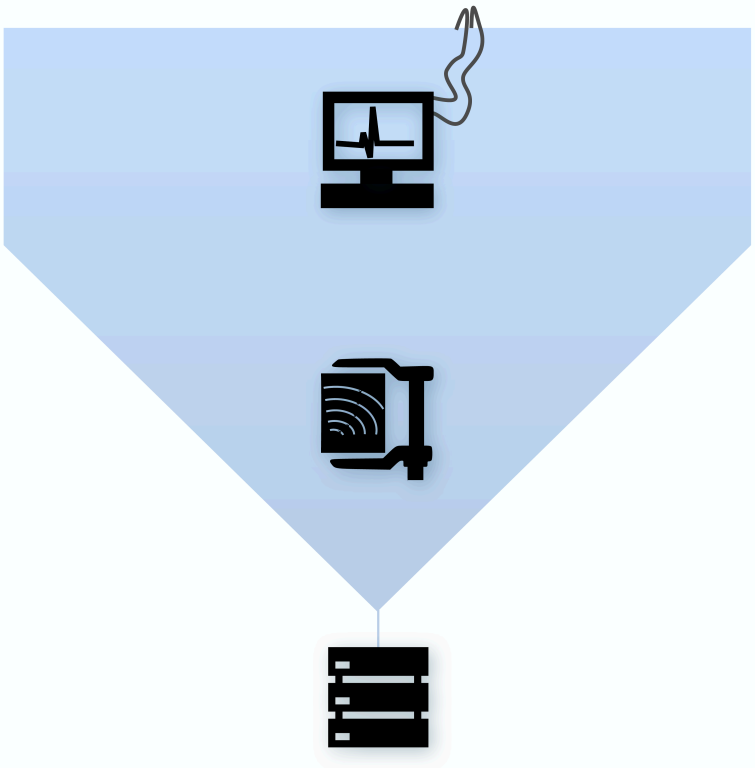
PANDA



P̄ANDA — Event Reconstruction



- **Continuous** read out
  - *Unique & novel feature*
  - Background & signal similar, diverse physics
  - No hardware trigger based on few sub-detectors, but online event reconstruction using full detector information

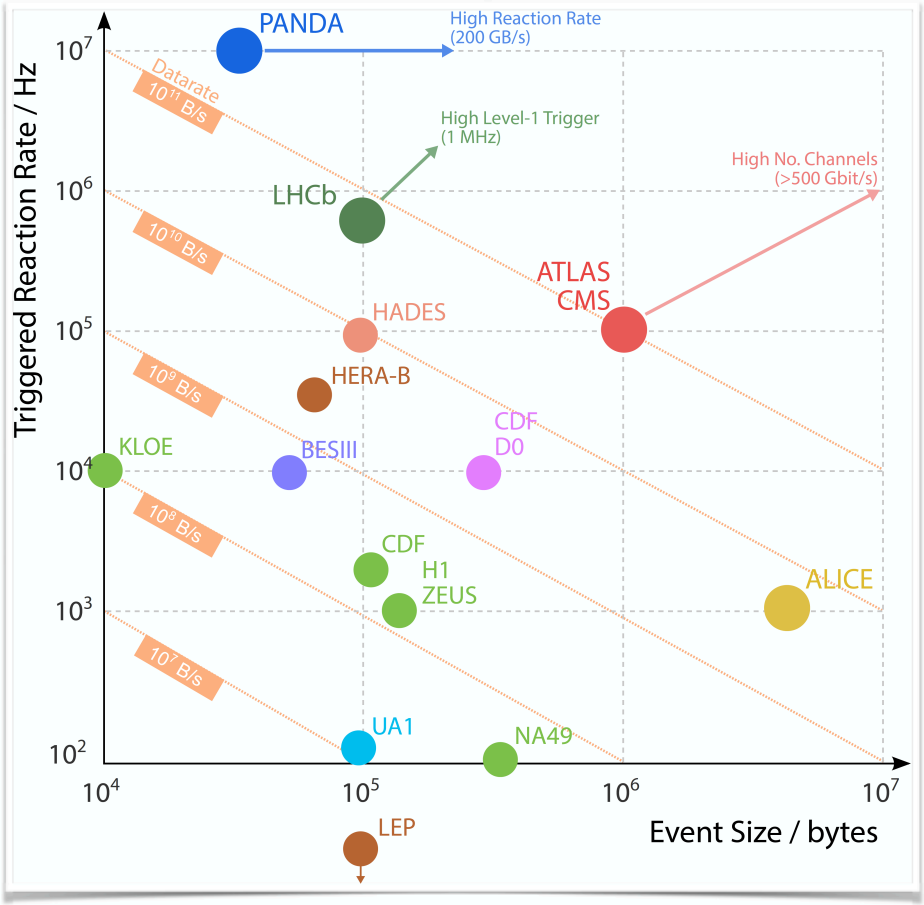


Rate	Event:	2×10^7/s *Full version*
	Raw data:	**200 GB/s**
Reduction	Amount:	**~1/1000**
	Time:	**50 ns/evt**

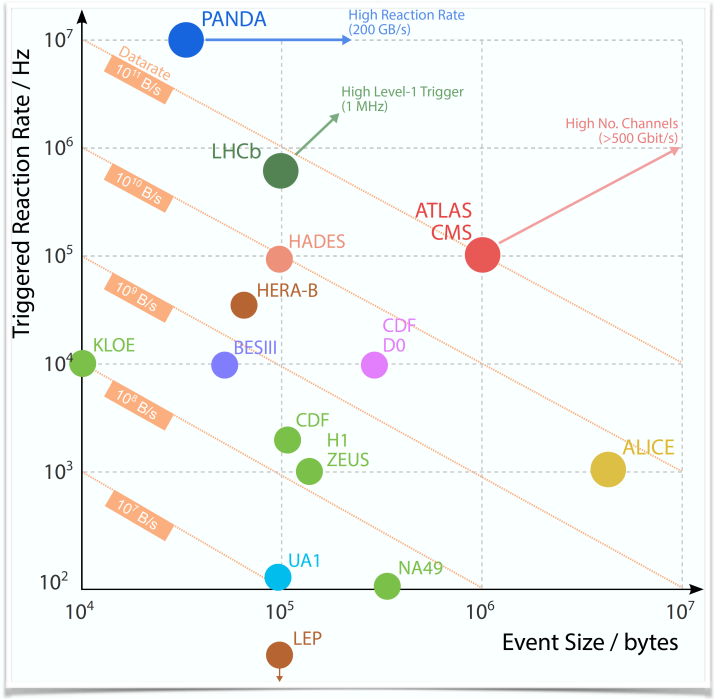
(Reject background events, save interesting events)

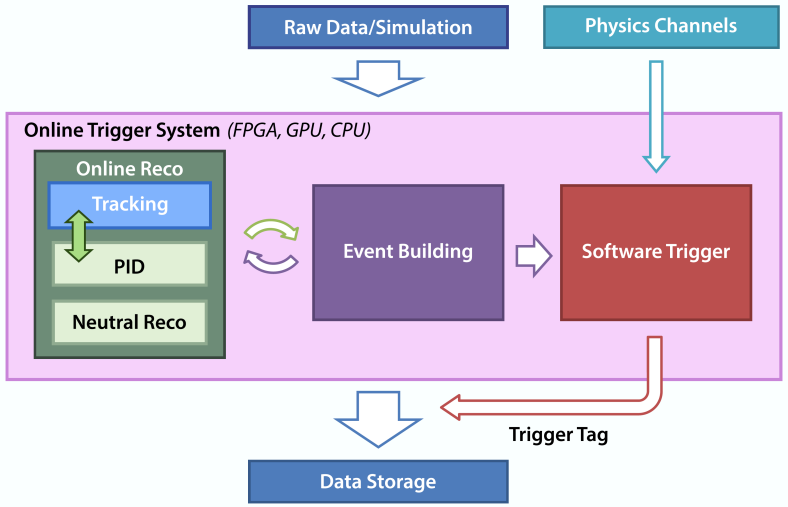
Storage space for offline analysis	**3 PB/y**

P̄ANDA — Online Trigger

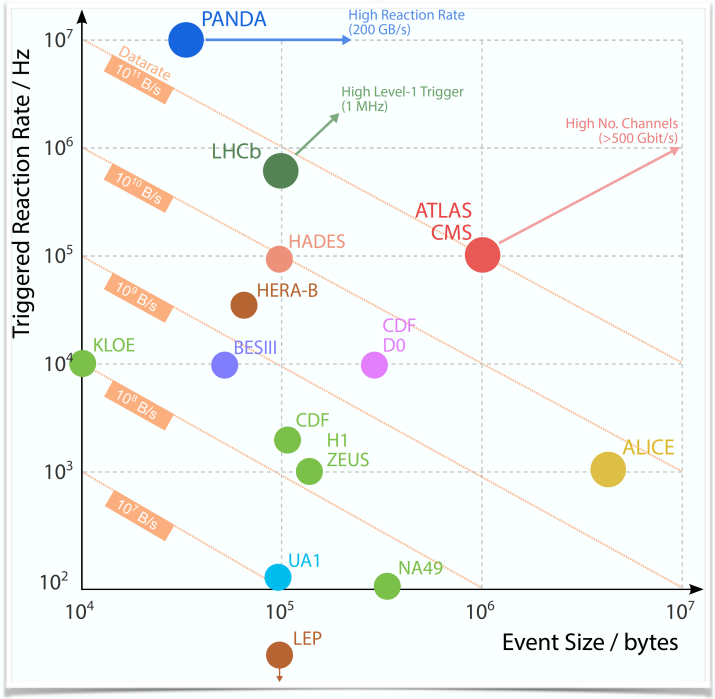


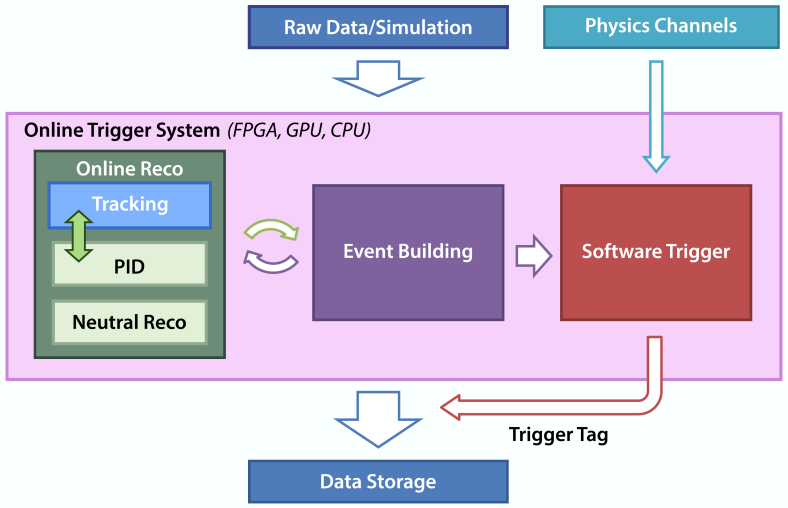
P̄ANDA — Online Trigger





PANDA — Online Trigger



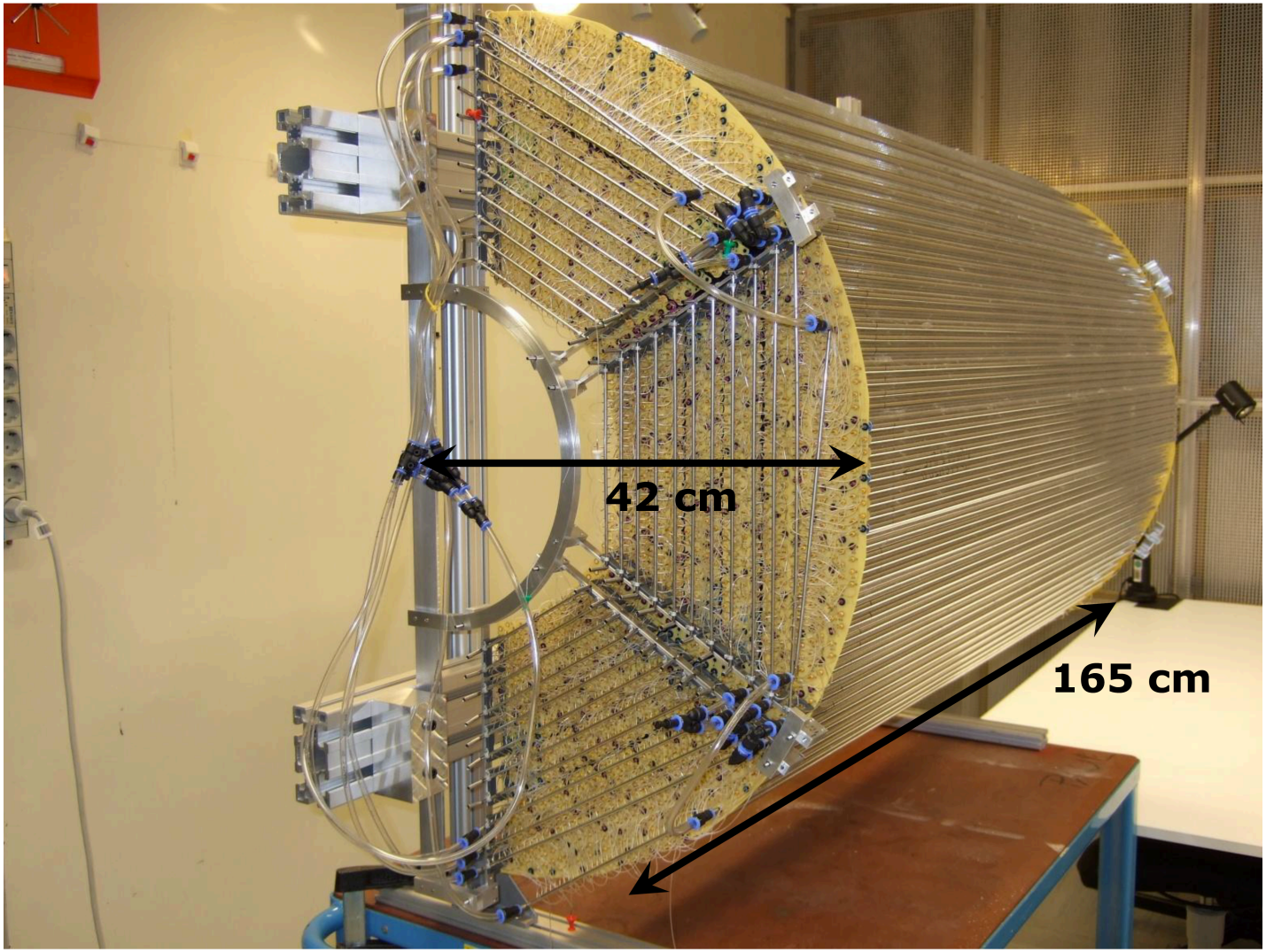


Requirements to Online Tracking

- Fast
- Sophisticated algorithms possible; reprogrammable
- Speed ↔ precision

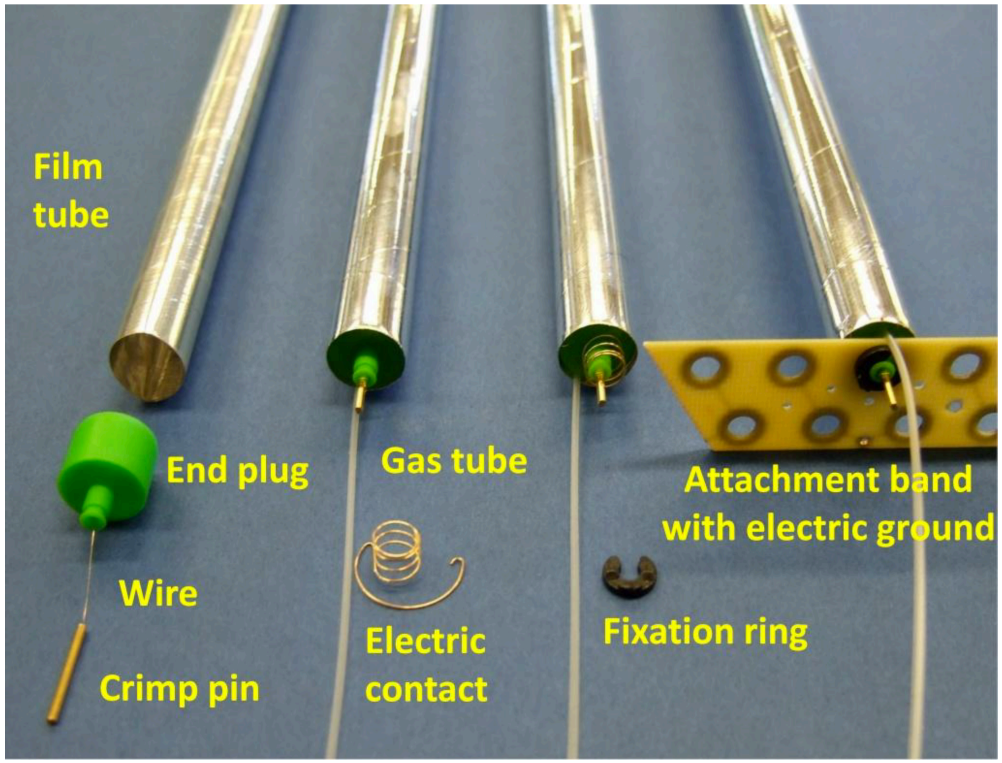
GPUs

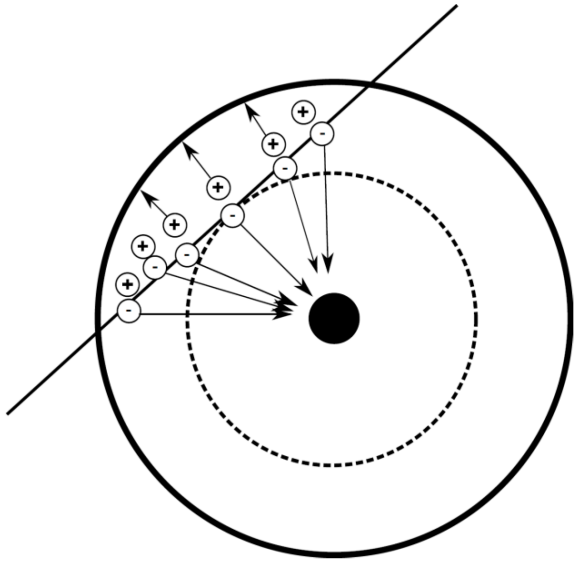
Straw Tube Tracker (STT)



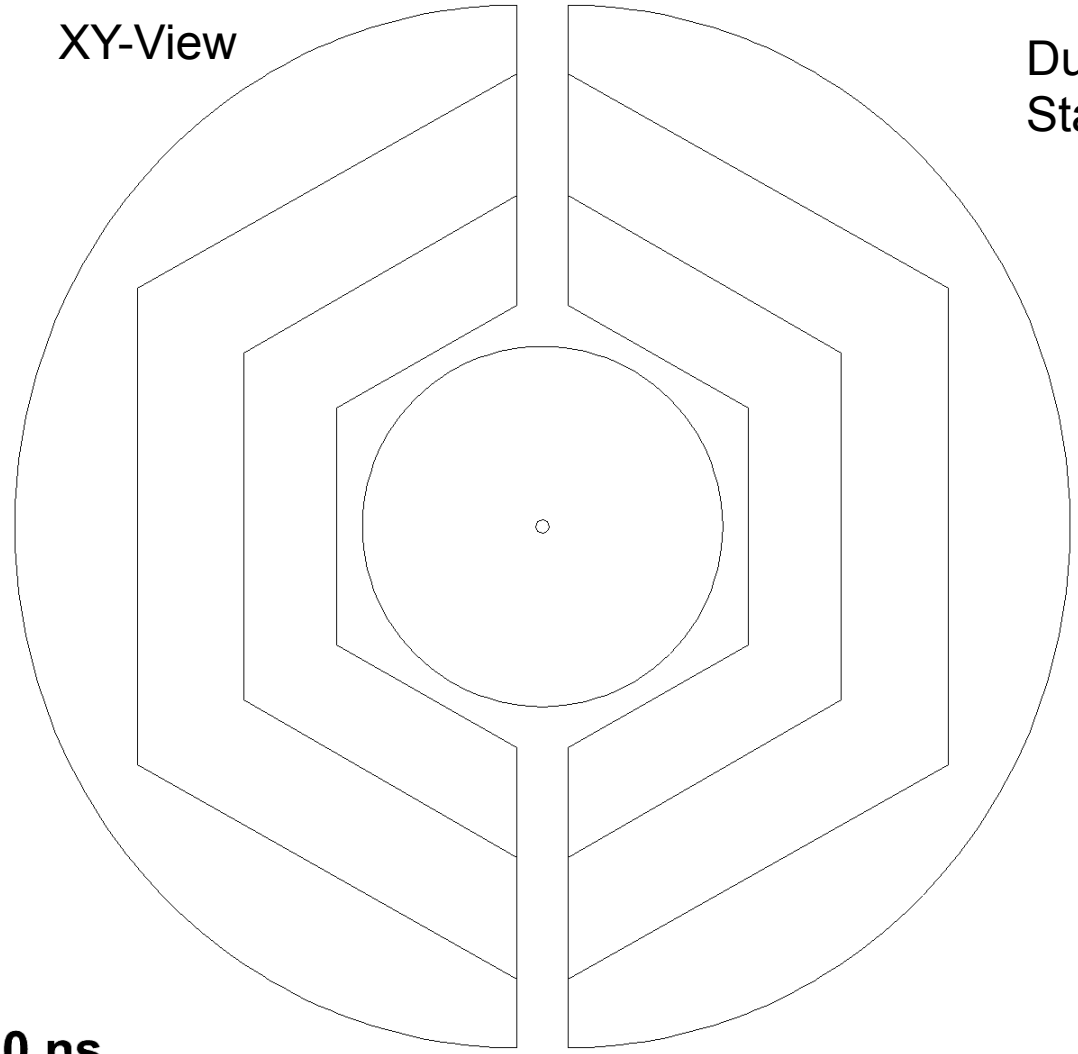
Straw Tube Tracker STT

- 4636 Straw Tubes
- Length: 150 cm, Diameter: 1 cm





Hitstream Display: 15 GeV/c DPM, 50 ns Mean Time



Dual Parton Model (DPM):
Standard p̄p background generator

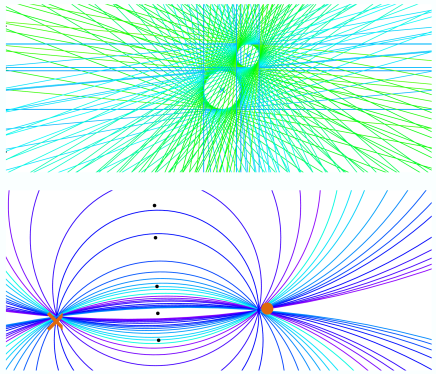
Black circles: Early isochrone
Blue circles: Early skewed isochrone
Green circles: Close isochrone
Red circles: Late isochrone
Black dots: MVD hits
Green dots: MVD hits r/z > 0.3
Black+Red dots: Triplets/Skewlets
Yellow tracks: Timed out track
Blue tracks: Current track

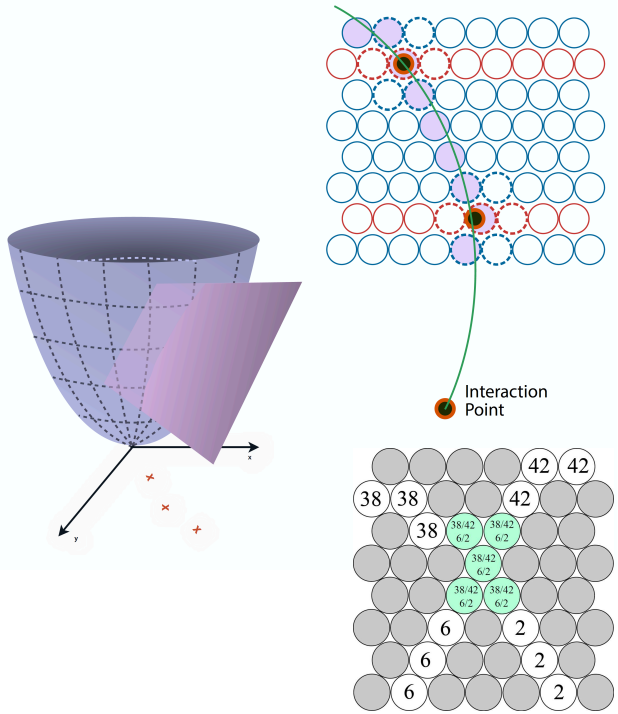
DPM Benchmark: Realistic event rate and structure, continuous operation

0 ns

Online Track Reconstruction Algorithms

- Hough-based Algorithms
  - Line Hough Transform
    - *CUDA*
    - *Thrust*
  - Circle Hough
  - Locus Circle Hough
- Triplet Finder
- Riemann Track Finder
- Cellular Automaton





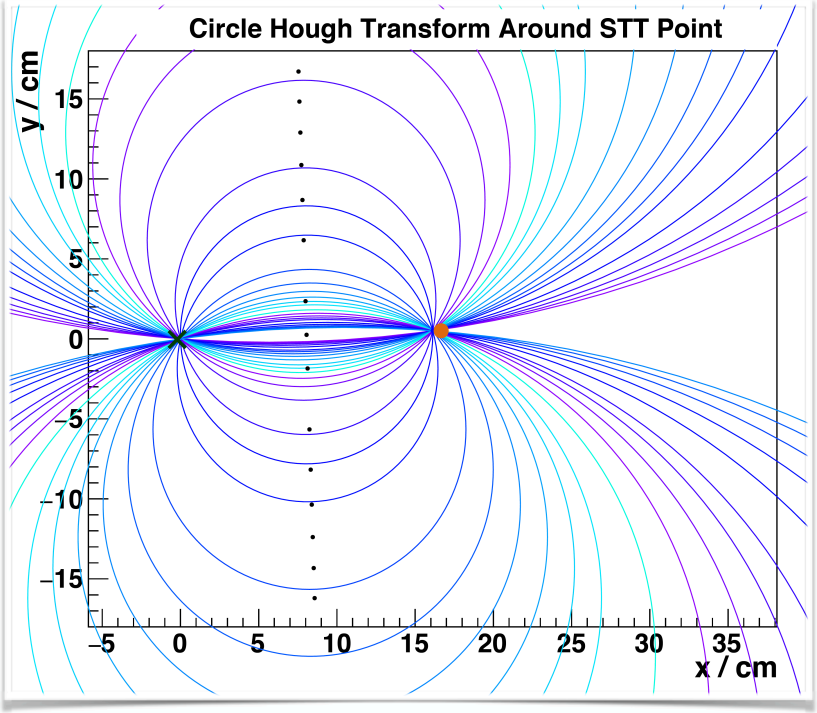
Circle Hough Algorithm

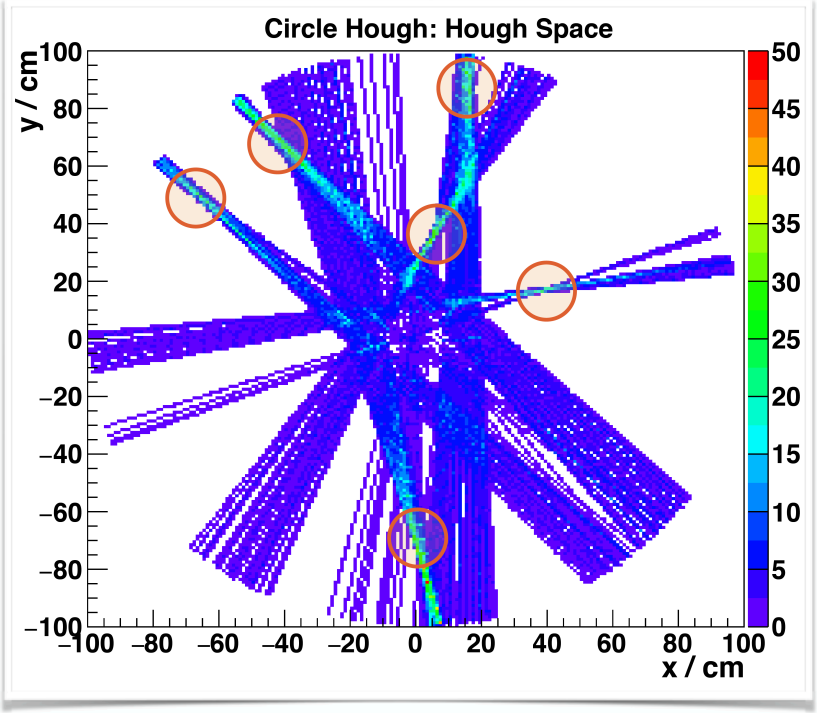
General principle

- Based on the Hough transform algorithm for detection of a shape S within a data set D
  - Parametrize target shape S with set of parameters
  - For each point in D, calculate all possible instances of S
  - Collect votes in parameter space P (Hough space)
  - Most voted sets of parameters in $P \Longrightarrow$ instances of S in D
- For trackfinding in $\bar{\text{P}}$ANDA
  - Calculate all possible tracks compatible with one hit
  - Repeat for many hits
  - Most voted track parameters $\Longrightarrow$ real tracks

Circle Hough Transform

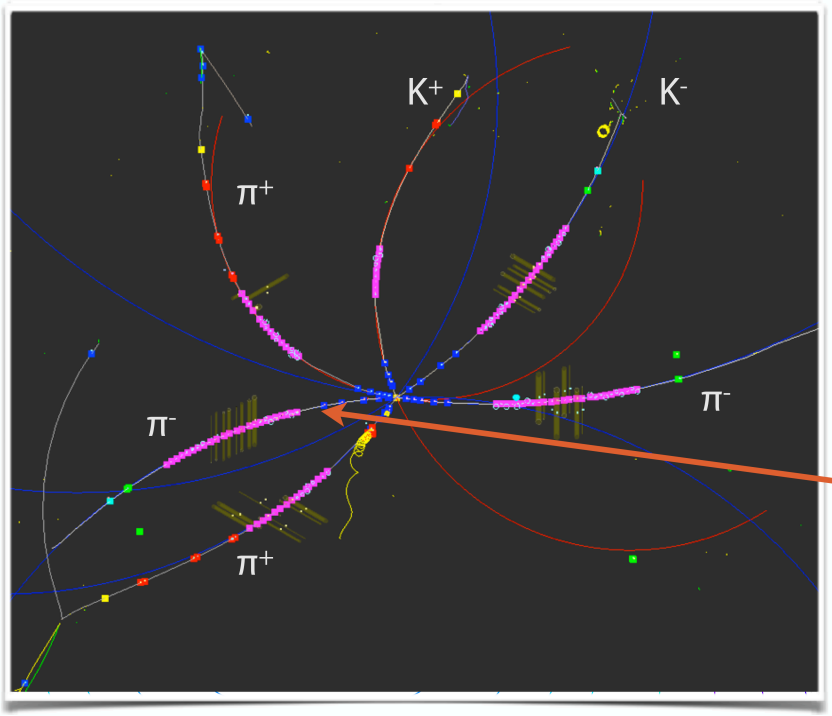
- Basic idea:
  - For every hit point, sample all possible circles going through coordinate origin in (x, y)-plane
  - Collect circle parameters in accumulator array
  - Extract peaks → tracks!

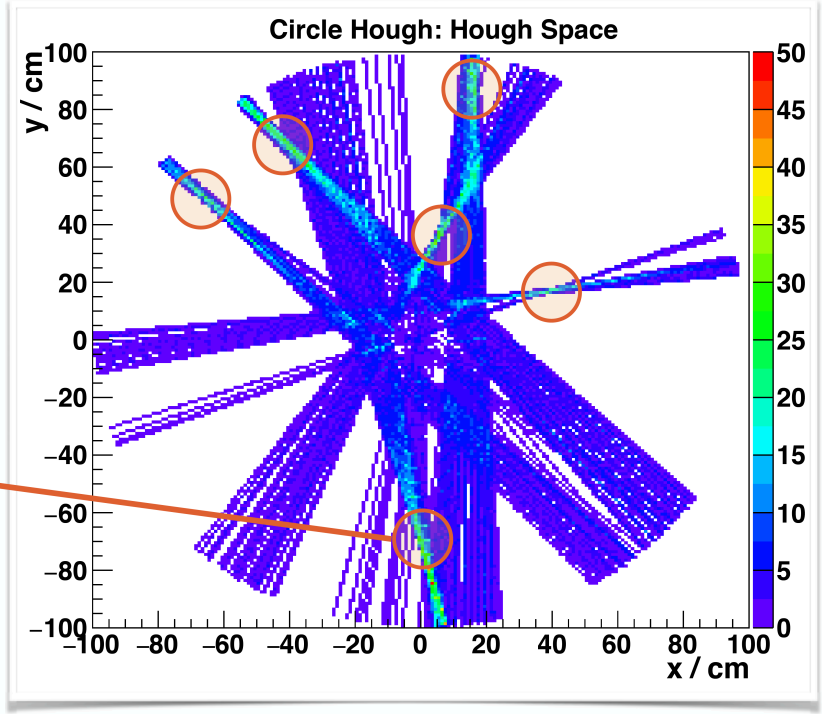




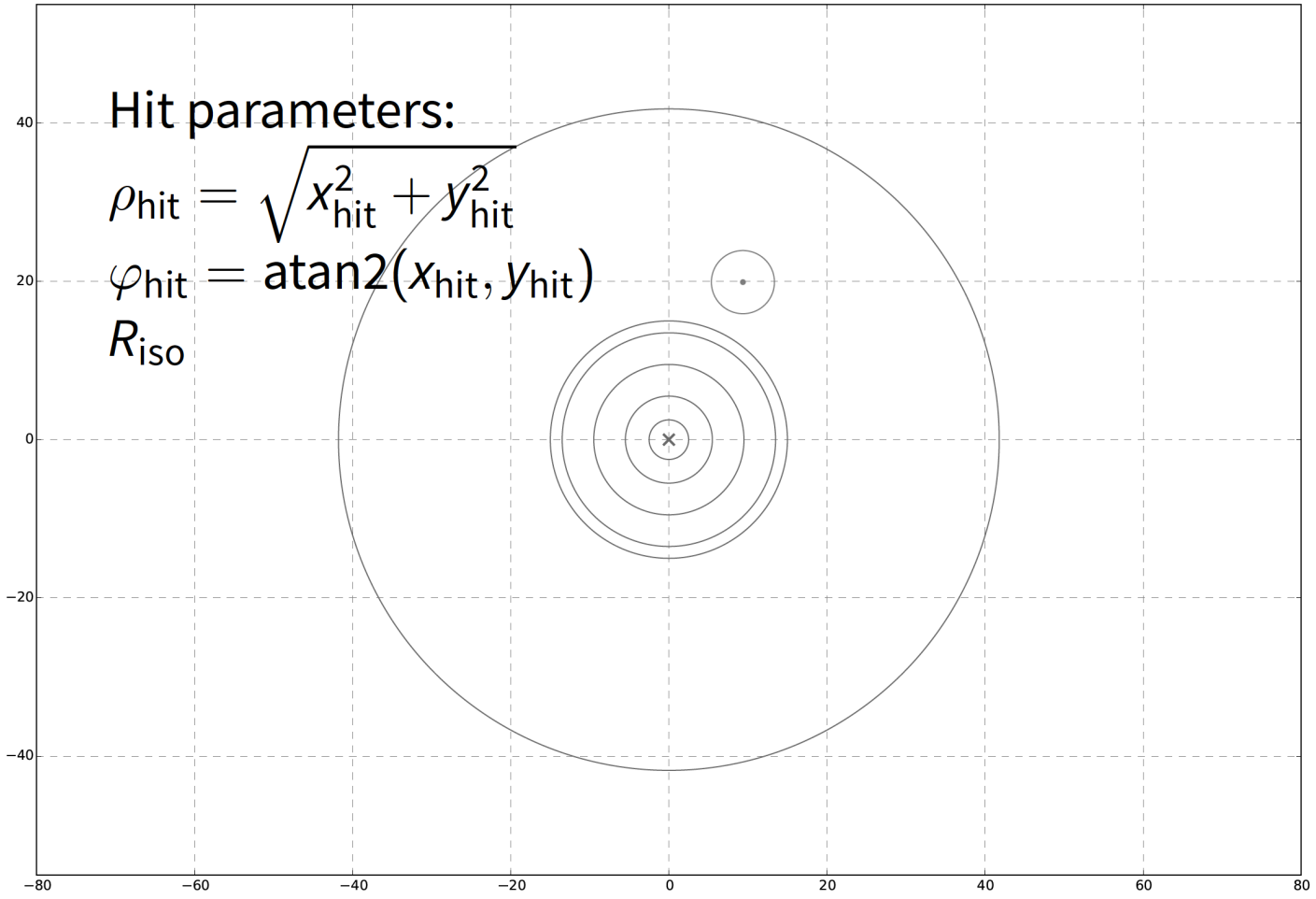
Circle Hough Transform

- Basic idea:
  - For every hit point, sample all possible circles going through coordinate origin in (x, y)-plane
  - Collect circle parameters in accumulator array
  - Extract peaks → tracks!



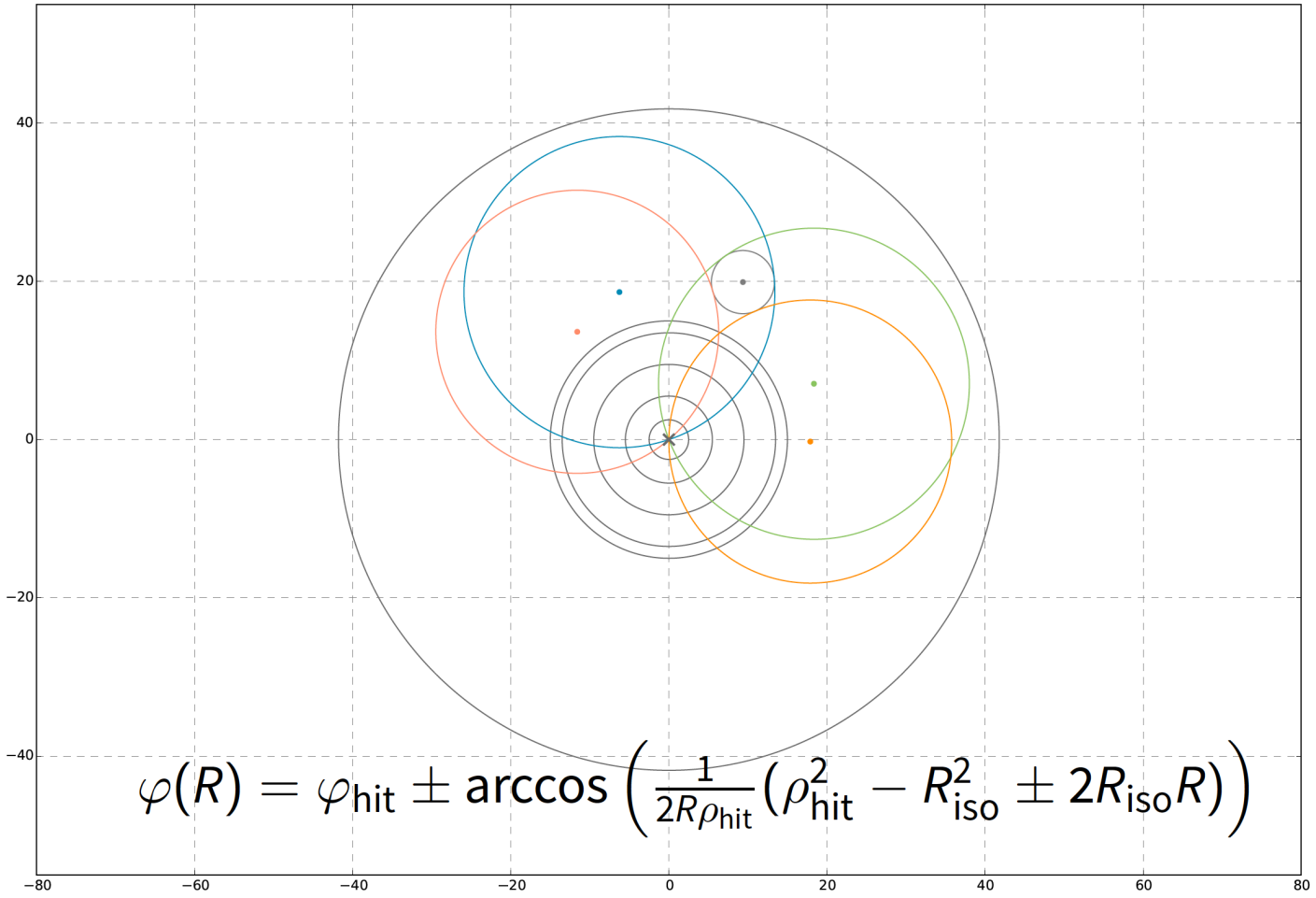


Hough Circles

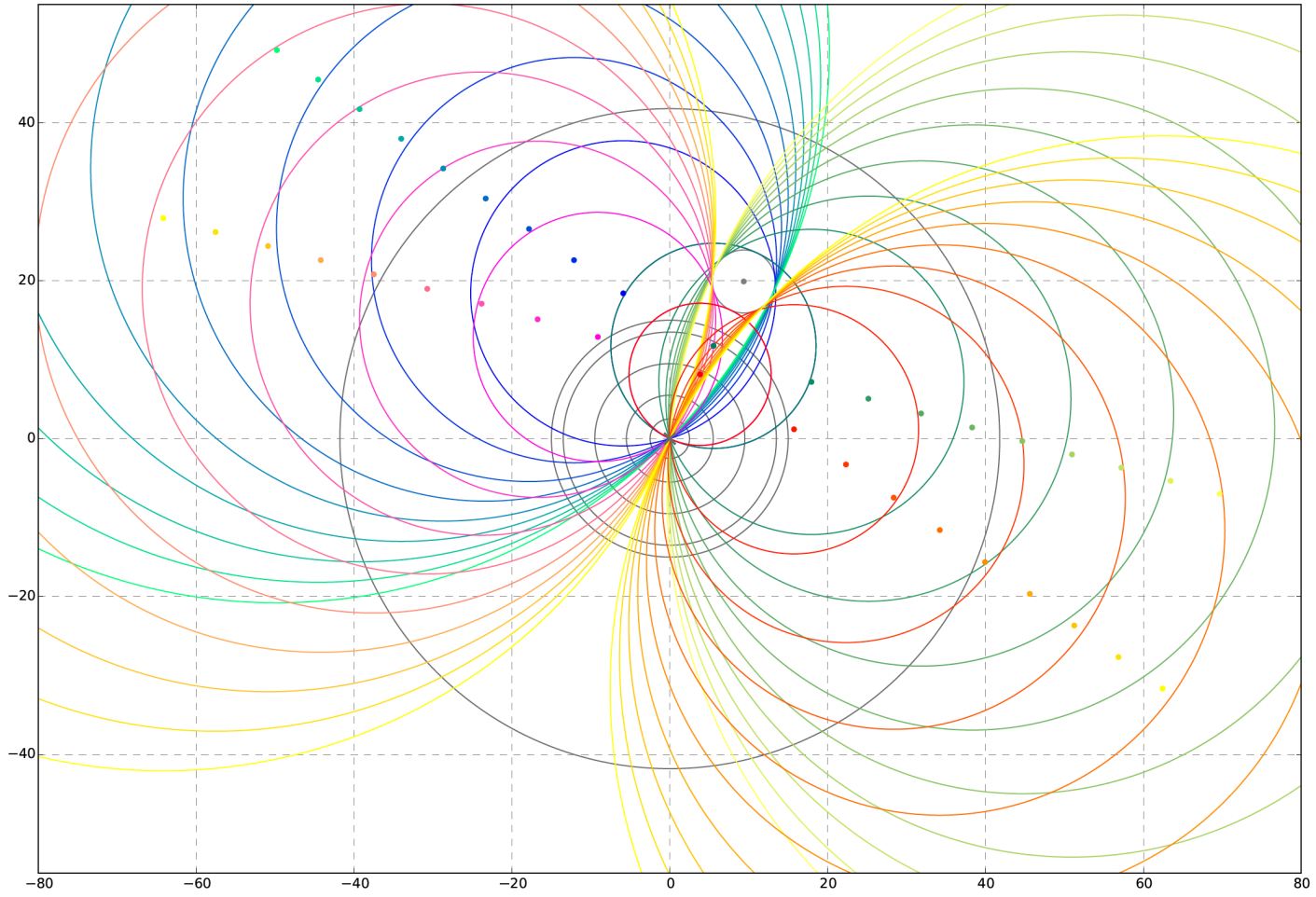


Hough Circles

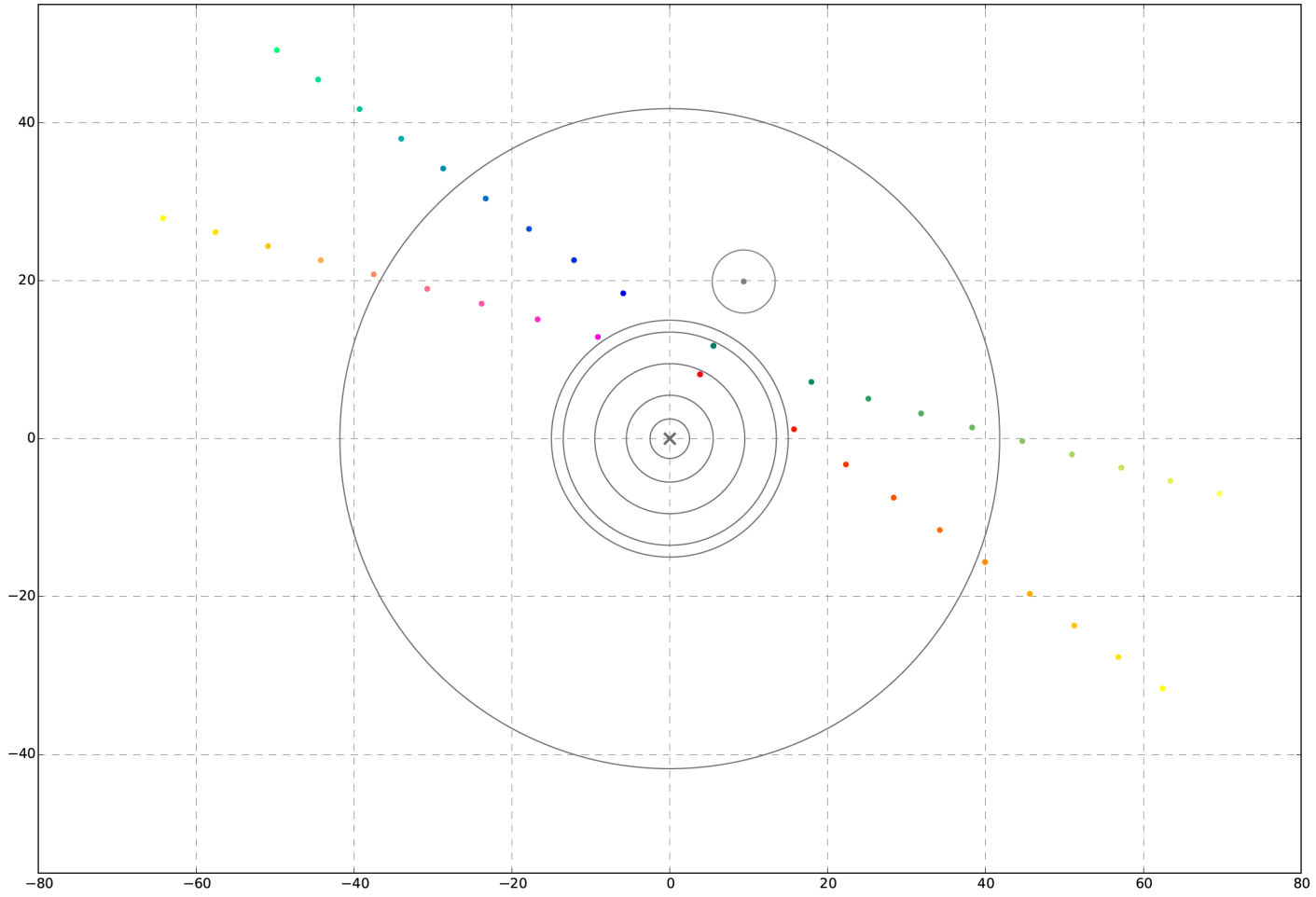
For each hit, 4 possible circles



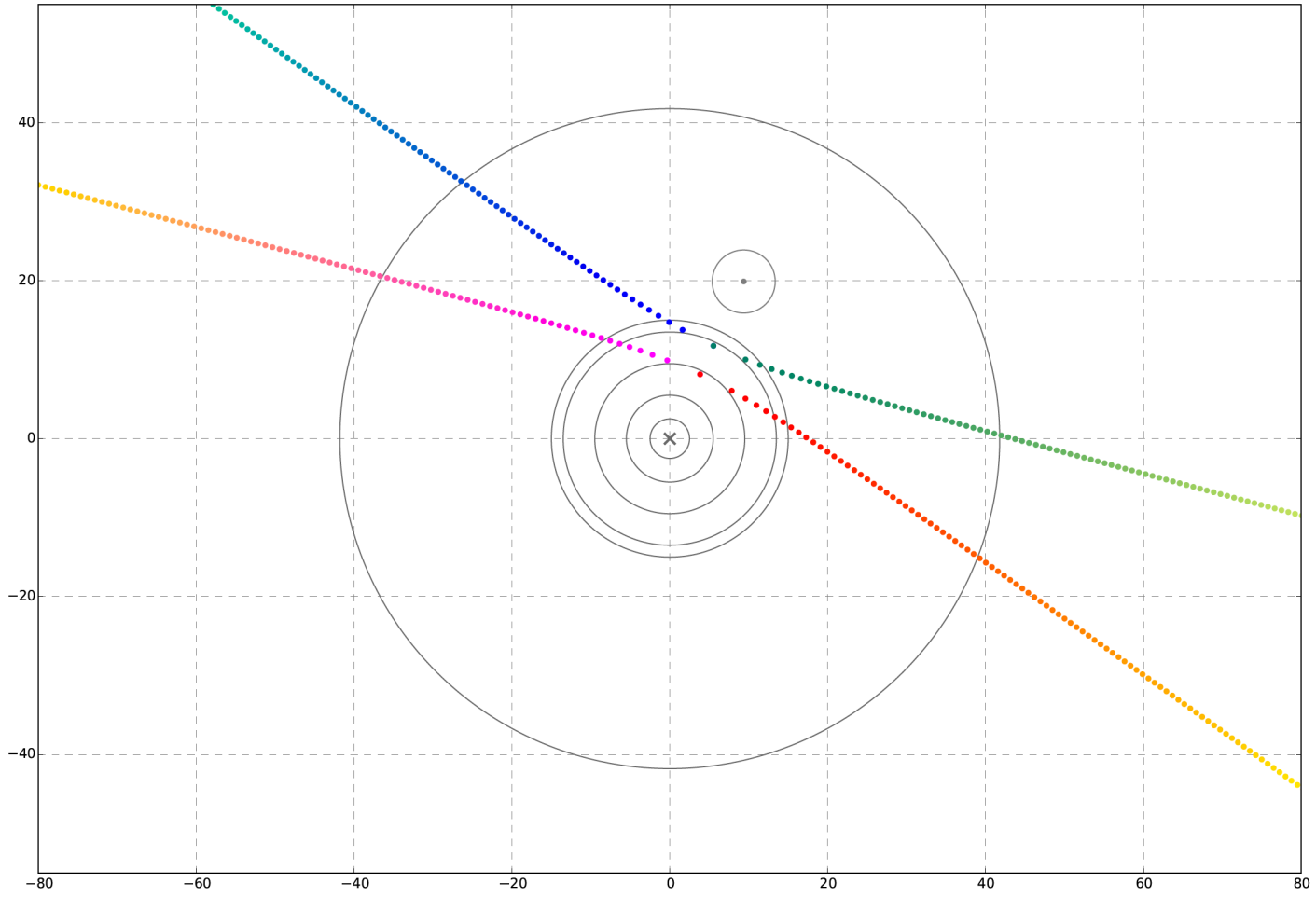
Hough Circles, Both Series



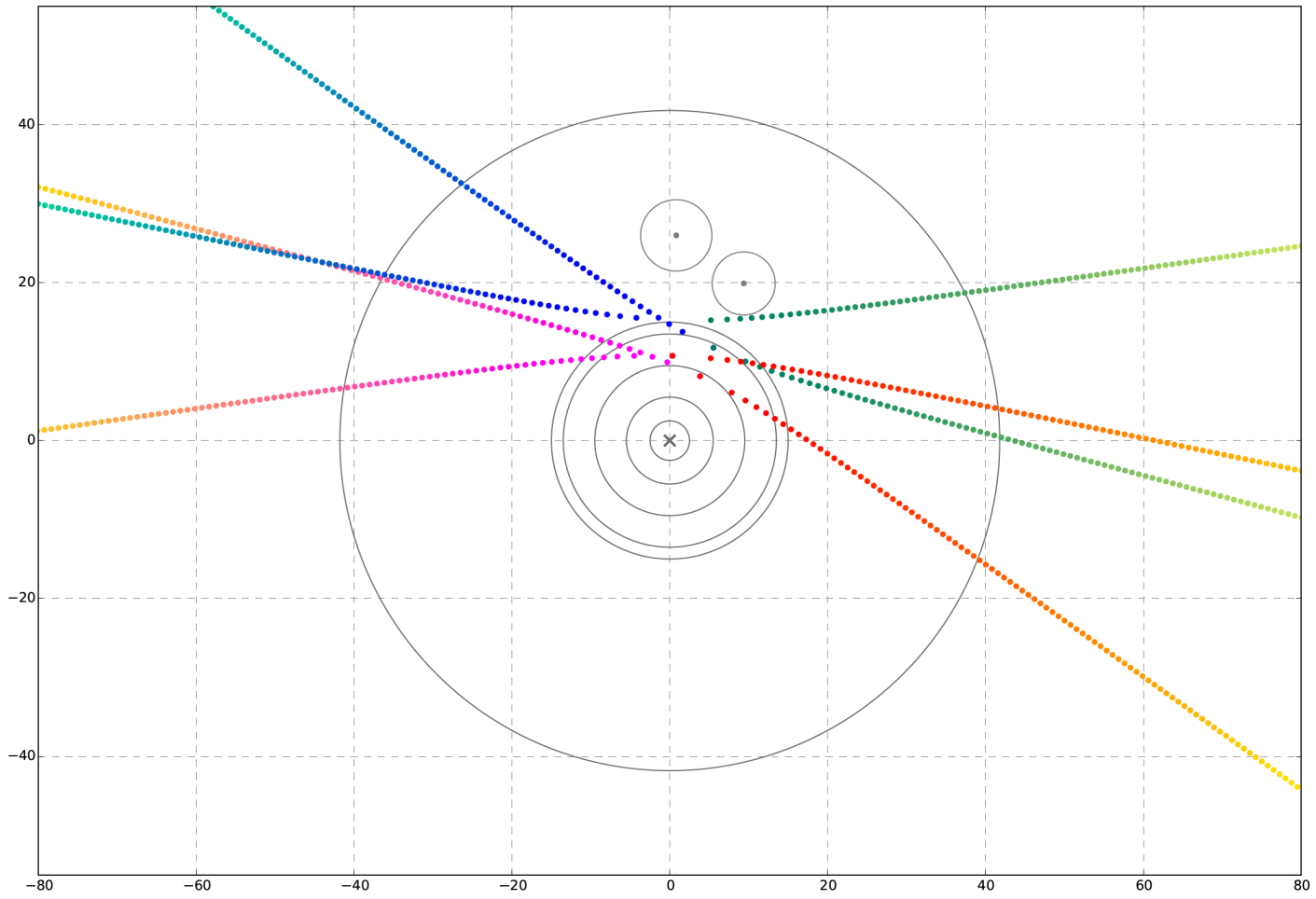
Hough Circles, Multiple Hits



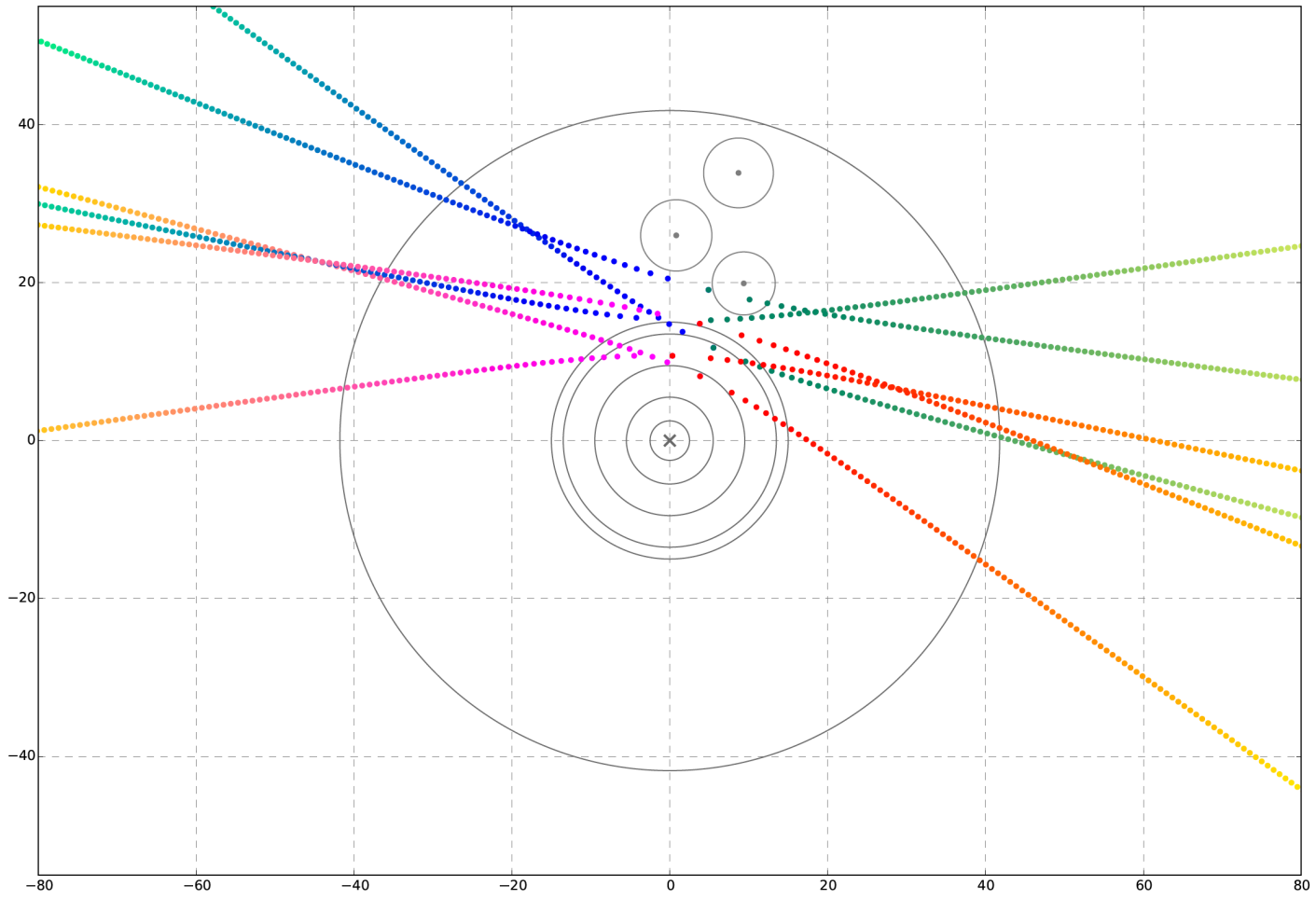
Hough Circles, Multiple Hits



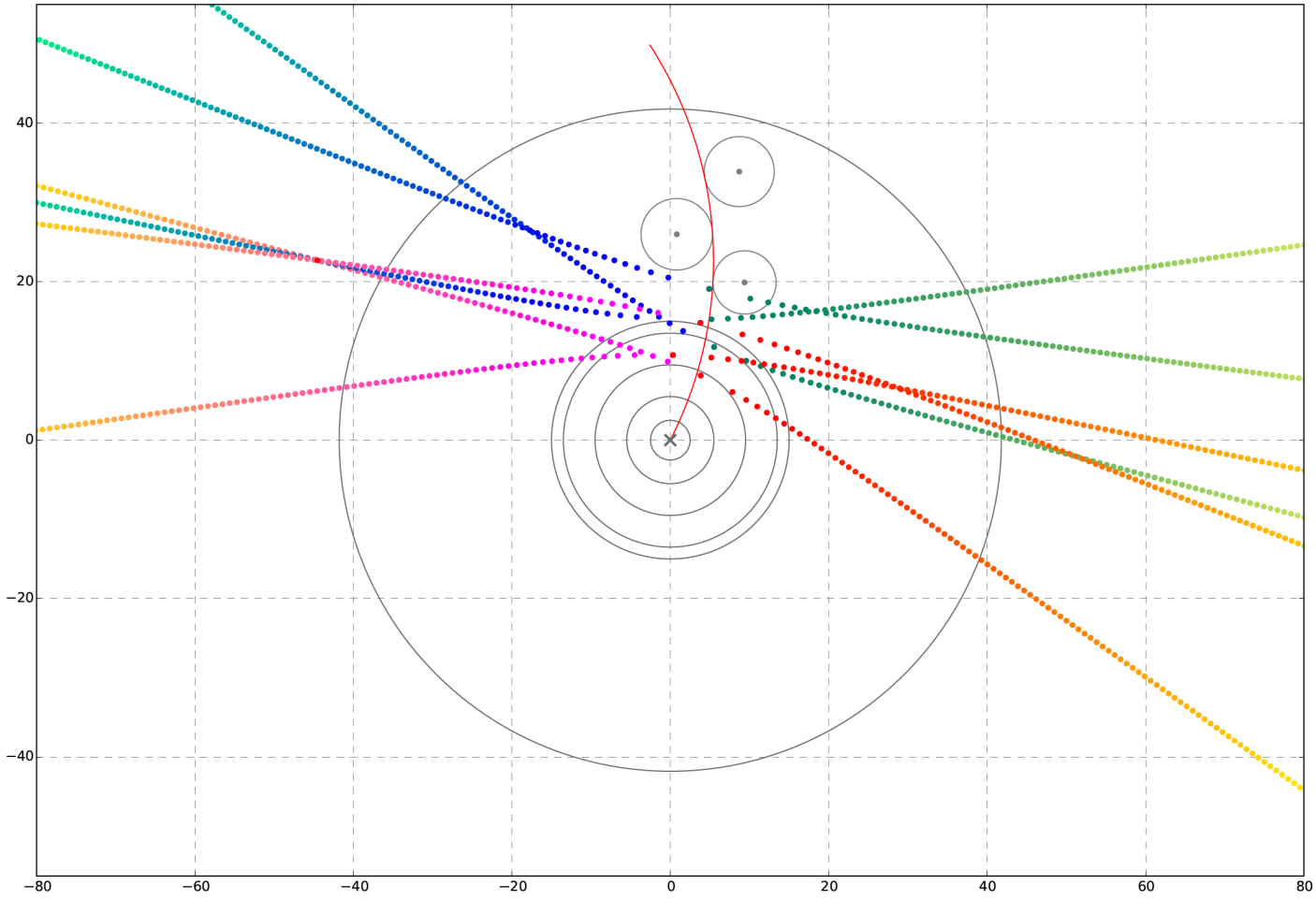
Hough Circles, Multiple Hits



Hough Circles, Multiple Hits



Hough Circles, Multiple Hits

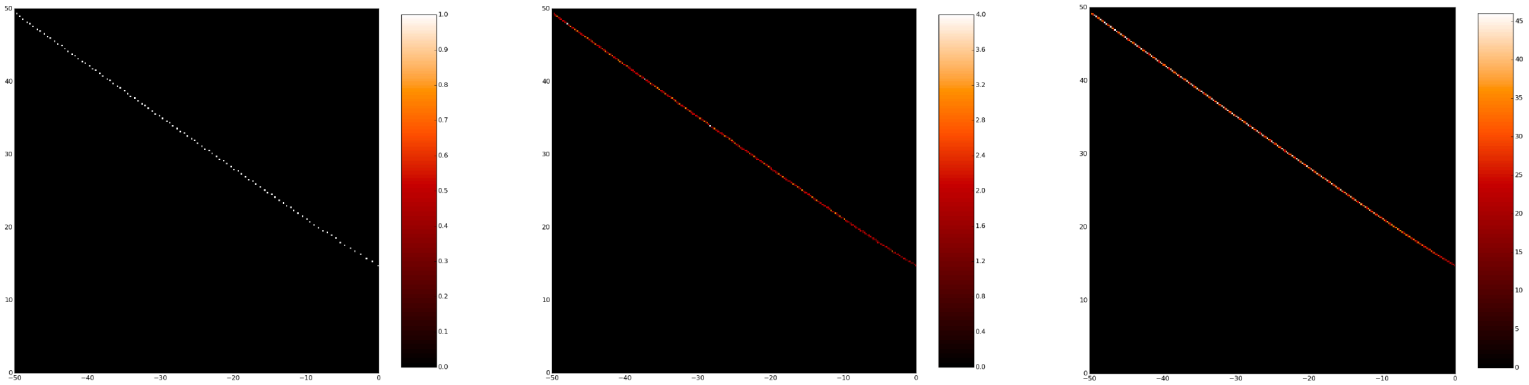


Accumulator Array

- Collect parameters in accumulator array
- Discrete 2D domain

$\Longrightarrow$ Fine tune necessary between radius sampling and array size

$\Longrightarrow$ Dishomogeneous structure makes peakfinding more difficult



- Each coordinate pair must fill the array exactly once

$\Longrightarrow$ Current solution:

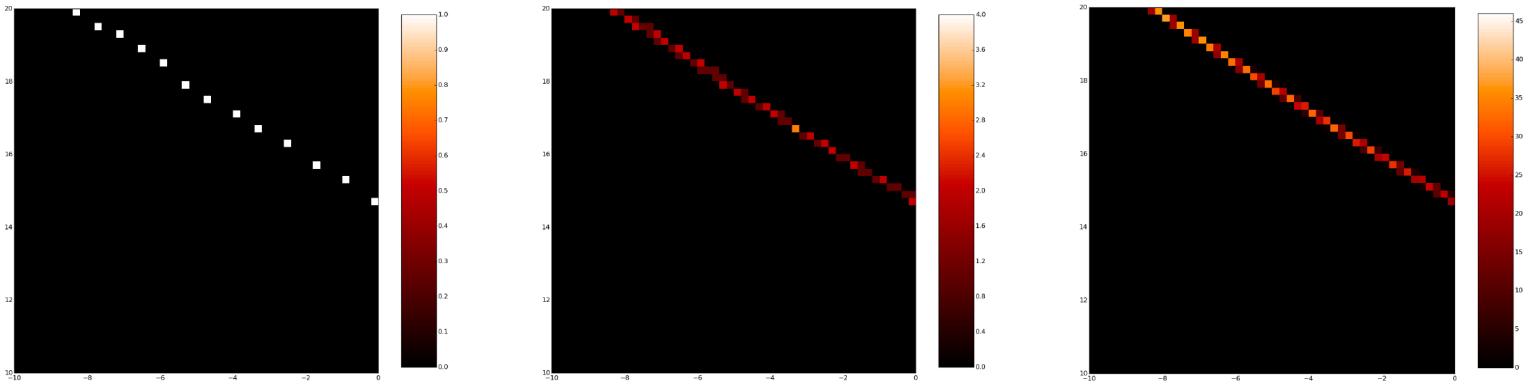
  - Reject values ending up in the same bin
  - Fill empty values by interpolation

Accumulator Array

- Collect parameters in accumulator array
- Discrete 2D domain

$\Longrightarrow$ Fine tune necessary between radius sampling and array size

$\Longrightarrow$ Dishomogeneous structure makes peakfinding more difficult



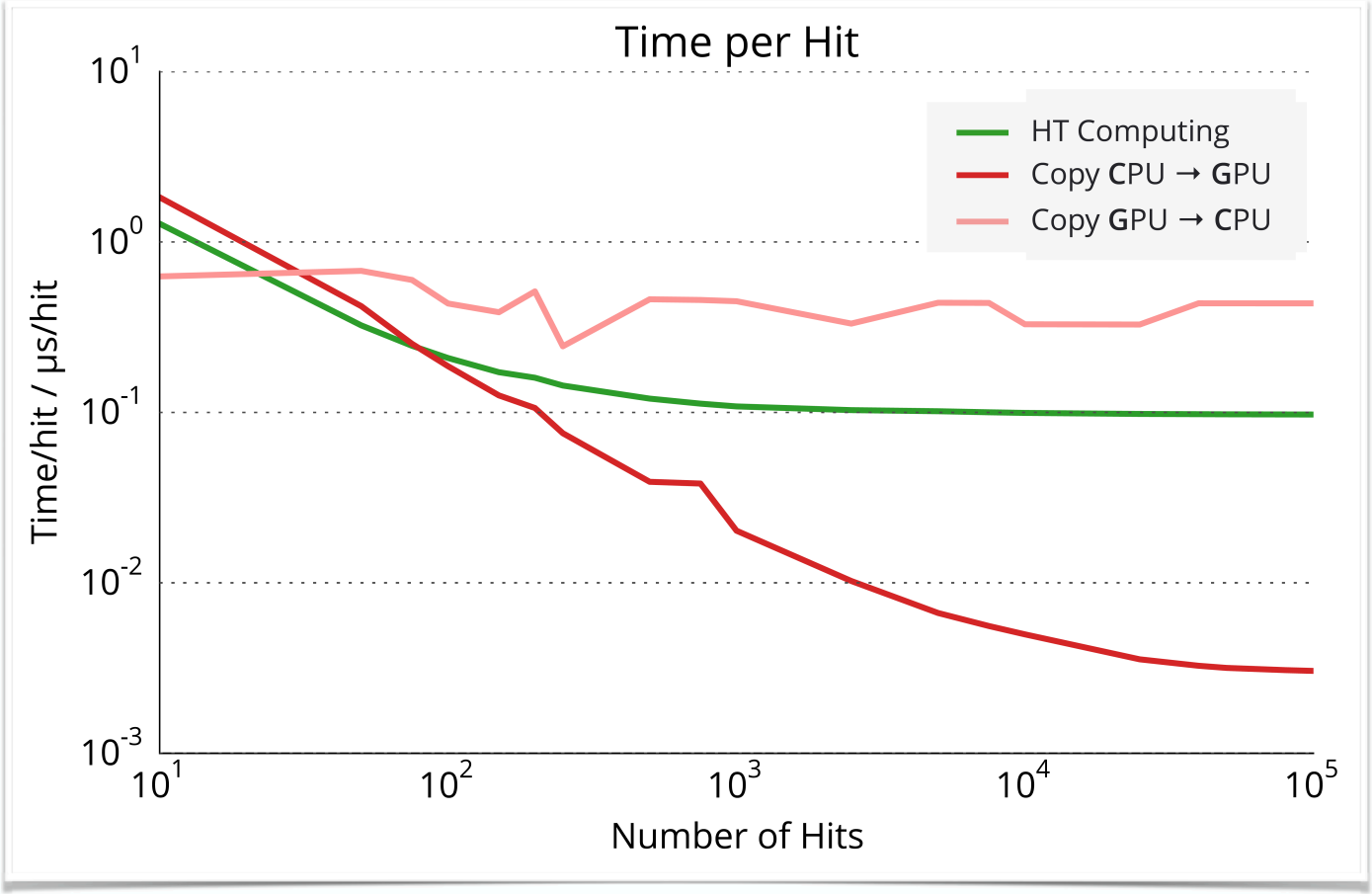
- Each coordinate pair must fill the array exactly once

$\Longrightarrow$ Current solution:

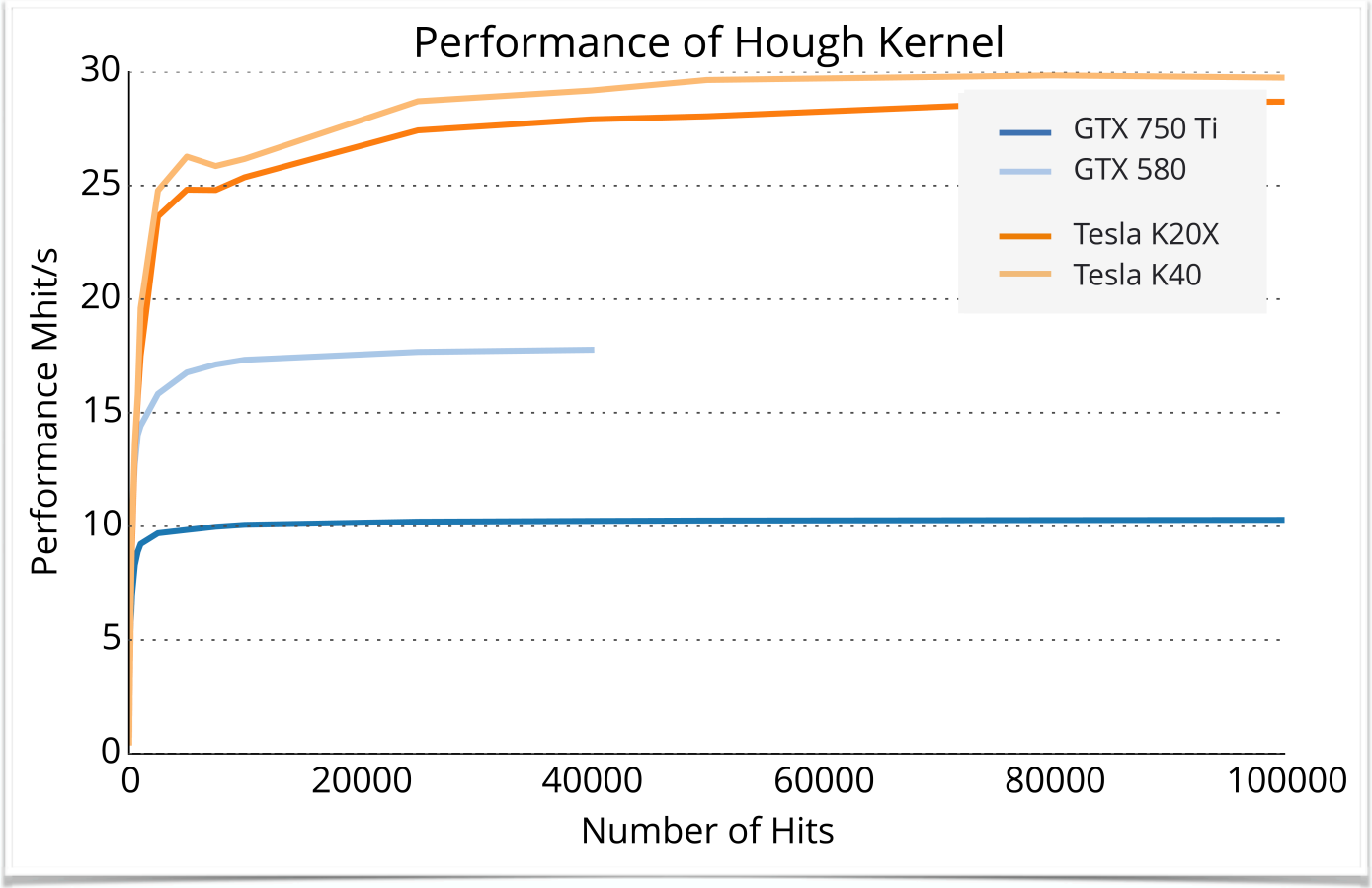
  — Reject values ending up in the same bin

  — Fill empty values by interpolation

Circle Hough — GPU Performance



Circle Hough — GPU Performance



Circle Hough — Implementation

- GPU standalone version
  For raw performance measurements
- PandaRoot version
  For physics measurements
  - With interface to GPU version *(CMake)*
  - Automatically selects GPU-accelerated methods, if GPU device is found; CPU methods, if not

```
PndCircleHoughTask * chTracker = new PndCircleHoughTask();
chTracker->AddHitBranch("MVDHitsStrip");
chTracker->AddHitBranch("STTHit");
chTracker->SetUseGpu(false);  // can be turned off
```

Alternative Approach

JÜLICH FORSCHUNGSZENTRUM

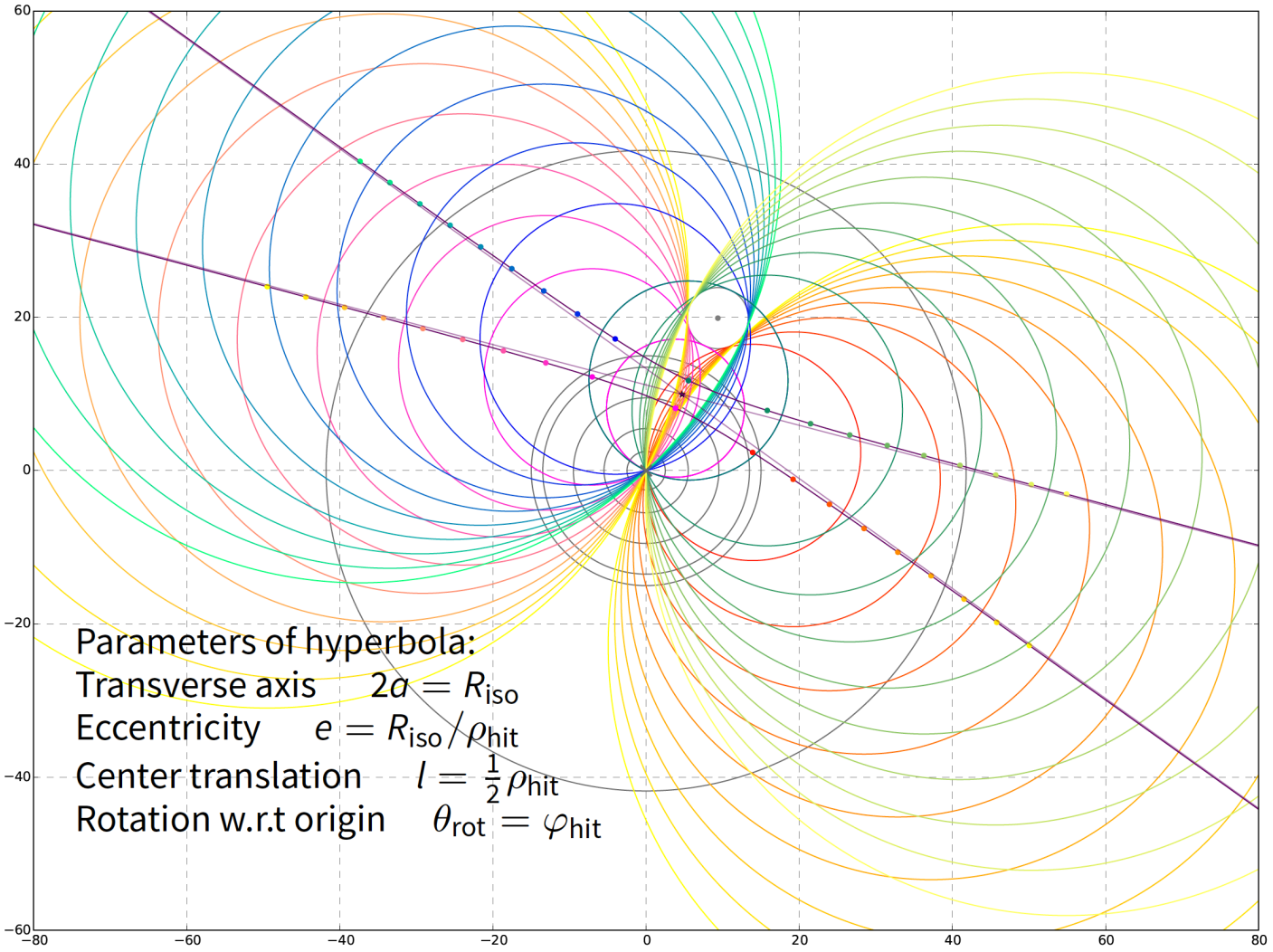
panda

Based on two properties of our problem:

- Geometric locus of circle centers is known
  - Hyperbola, whose parameters are easily computed from hit data
- Analogy between Hough space and 2D image
  - Bin in 2D accumulator array $\Longrightarrow$ pixel in bitmap
  - Take advantage of image processing algorithms
  - Especially important for implementation on parallel architectures

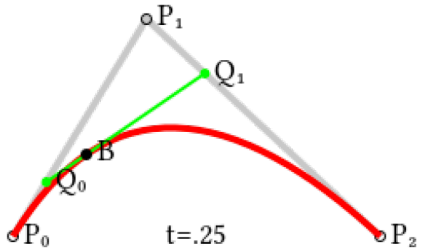
$\Longrightarrow$ Operate directly on locus rather than on individual circles

Locus of Hough circle centers



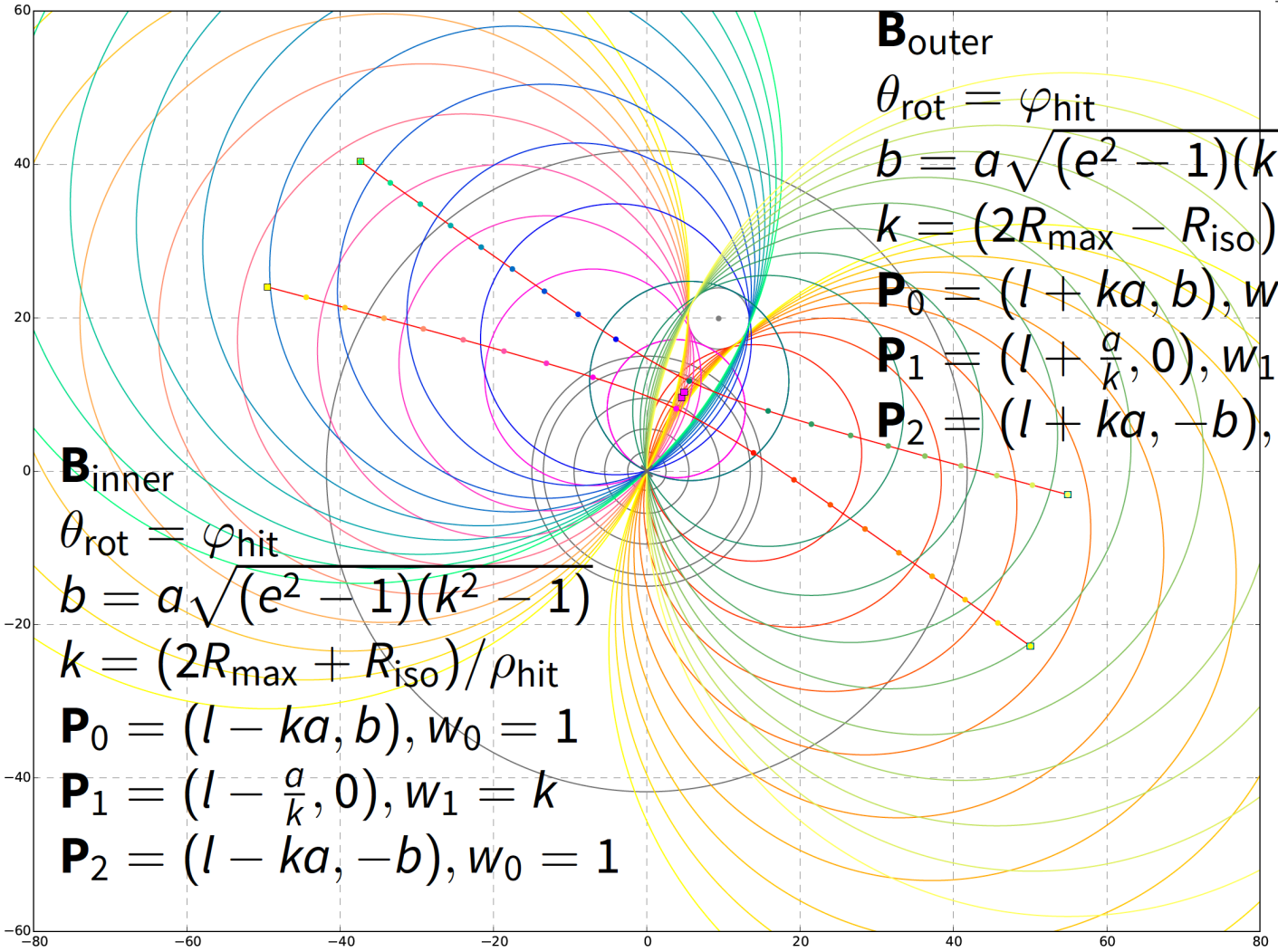
Bezier Curves

- Bezier curves: parametric polynomial curves of degree N, defined by their $N + 1$ control points $\mathbf{P}_0 \ldots \mathbf{P}_N$
- Rational Bezier curves of degree 2 represent conic sections exactly $\Longrightarrow$ “lossless” description of hyperbola
  - Convenient parametrization
  - Easy linearization
  - Widely used in computer graphics

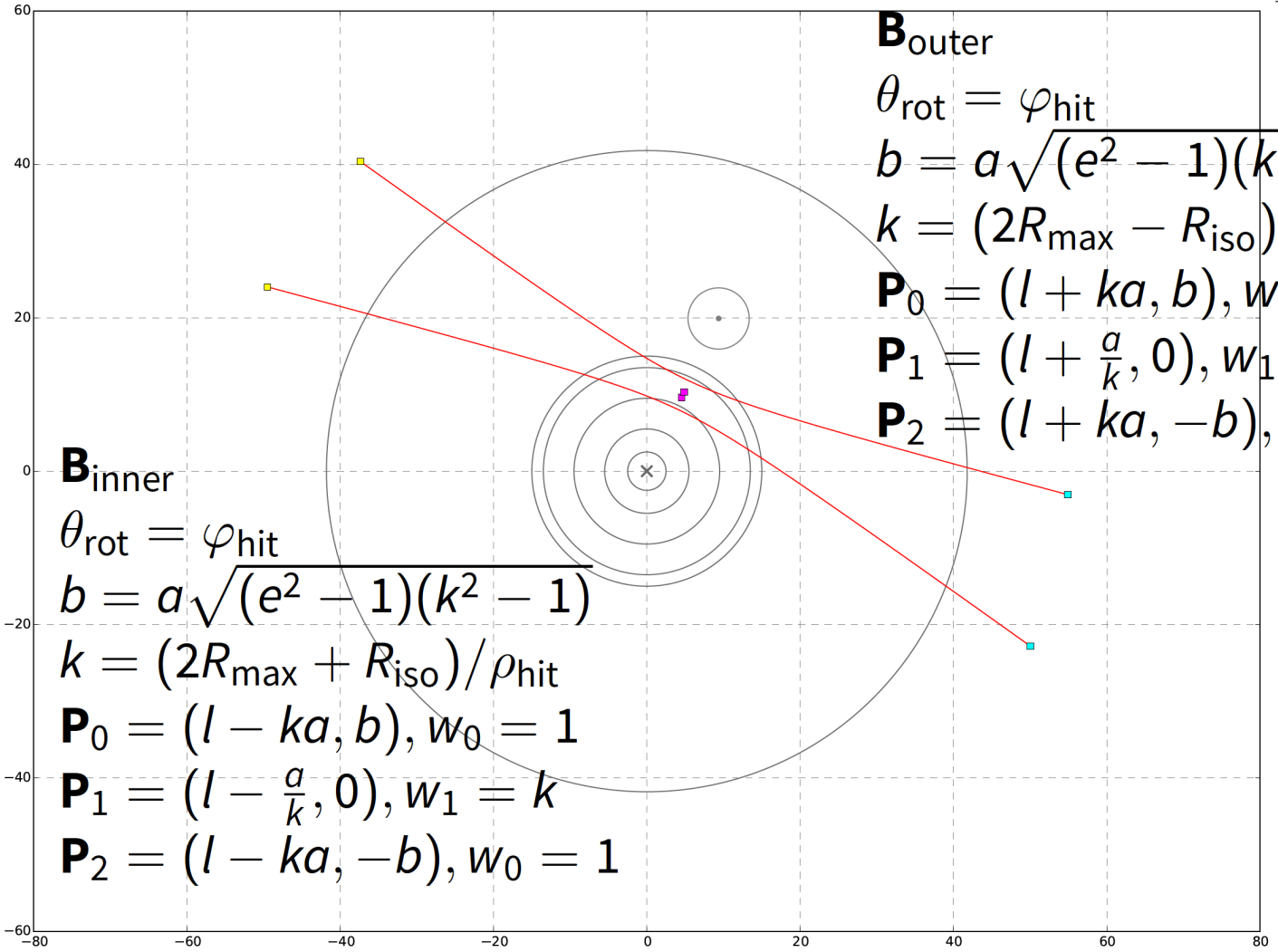


$$\mathbf{B}(t) = (1-t)^2 w_0 \mathbf{P}_0 + 2(1-t)t w_1 \mathbf{P}_1 + t^2 w_2 \mathbf{P}_2, \qquad 0 \leq t \leq 1$$

Parametrization with Bezier Curves



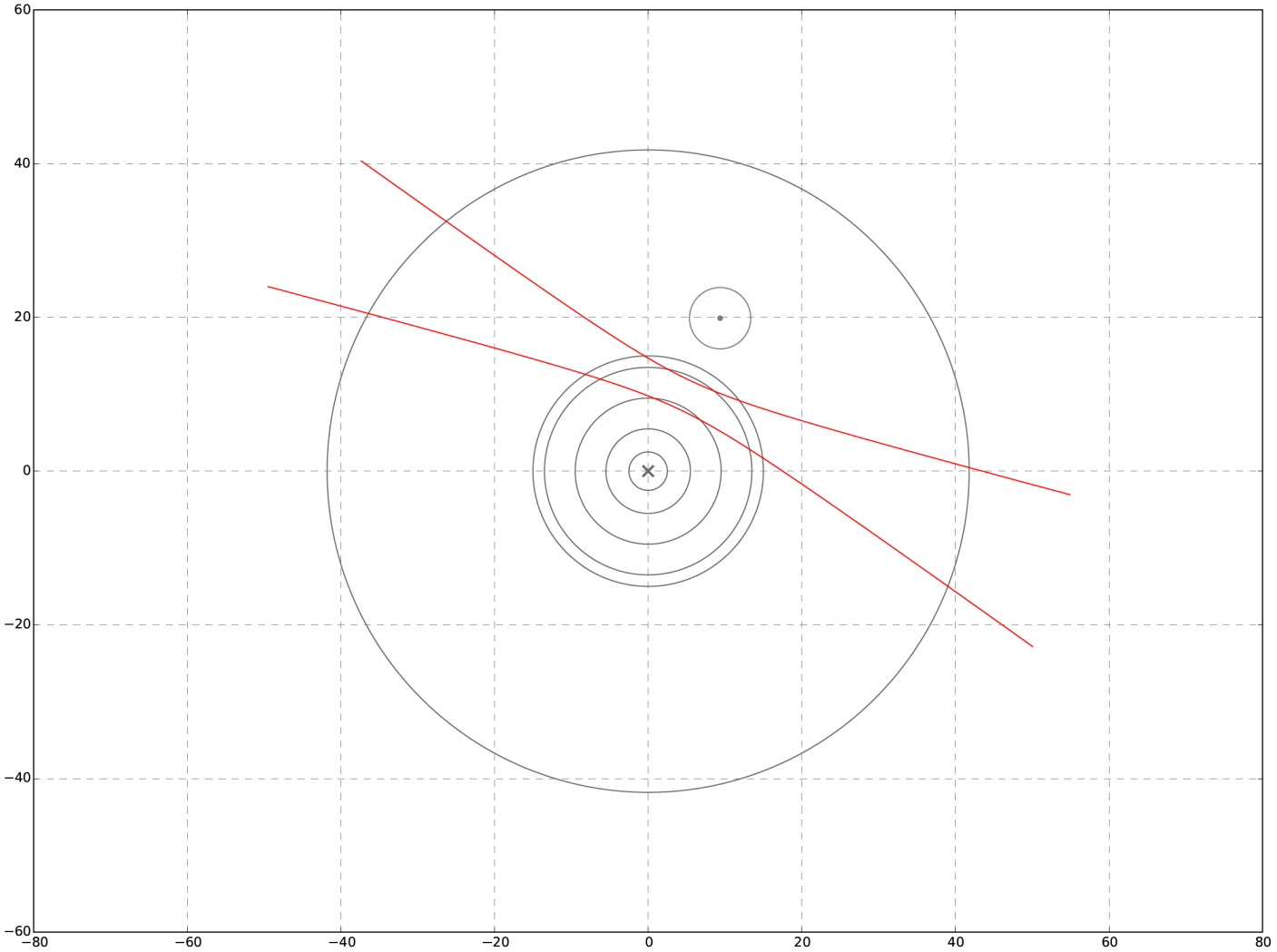
Parametrization with Bezier Curves



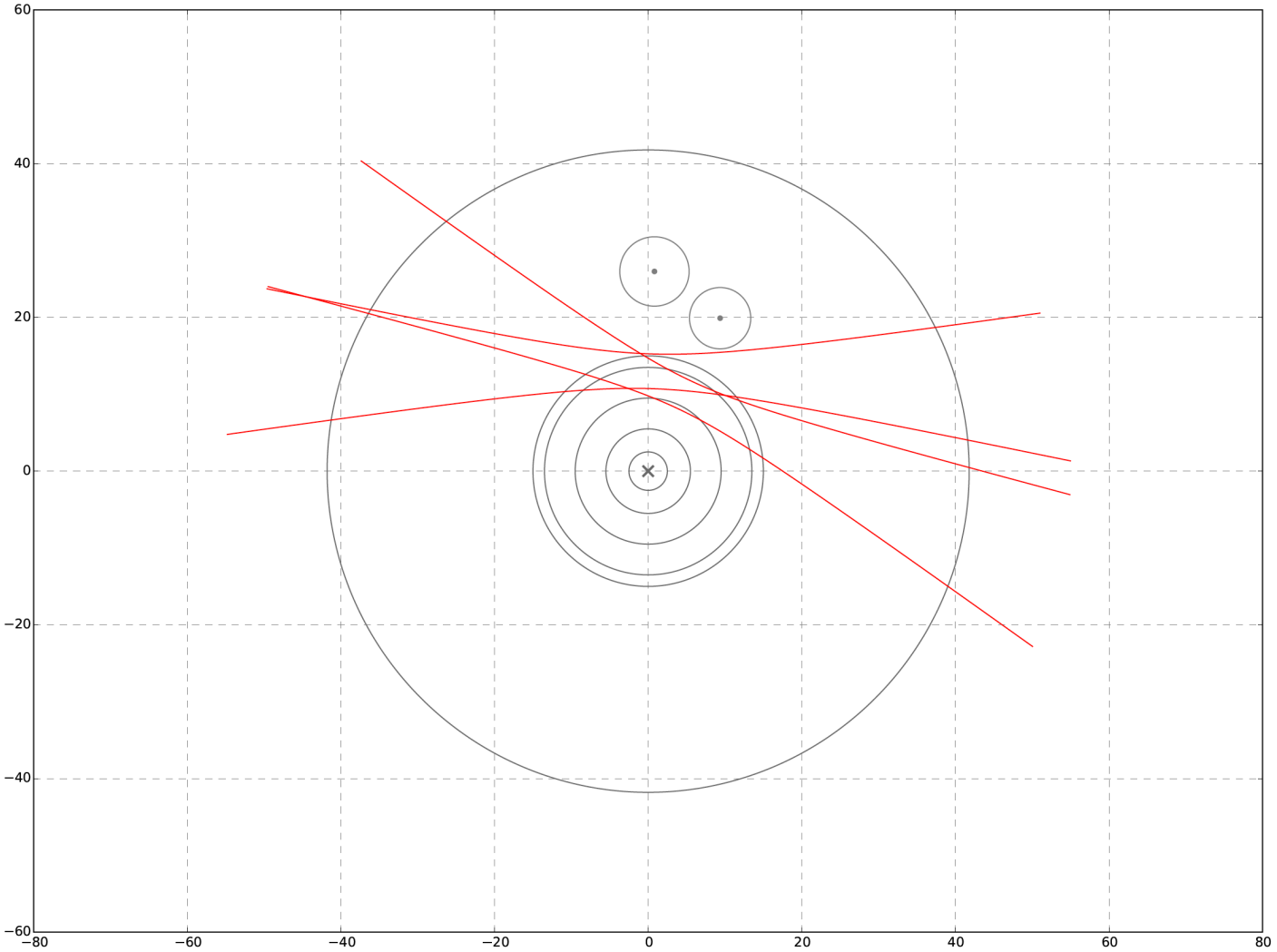
$$\mathbf{B}_{\text{inner}}$$
$$\theta_{\text{rot}} = \varphi_{\text{hit}}$$
$$b = a\sqrt{(e^2-1)(k^2-1)}$$
$$k = (2R_{\text{max}} + R_{\text{iso}})/\rho_{\text{hit}}$$
$$\mathbf{P}_0 = (l - ka, b), w_0 = 1$$
$$\mathbf{P}_1 = (l - \tfrac{a}{k}, 0), w_1 = k$$
$$\mathbf{P}_2 = (l - ka, -b), w_0 = 1$$

$$\mathbf{B}_{\text{outer}}$$
$$\theta_{\text{rot}} = \varphi_{\text{hit}}$$
$$b = a\sqrt{(e^2-1)(k^2-1)}$$
$$k = (2R_{\text{max}} - R_{\text{iso}})/\rho_{\text{hit}}$$
$$\mathbf{P}_0 = (l + ka, b), w_0 = 1$$
$$\mathbf{P}_1 = (l + \tfrac{a}{k}, 0), w_1 = k$$
$$\mathbf{P}_2 = (l + ka, -b), w_0 = 1$$

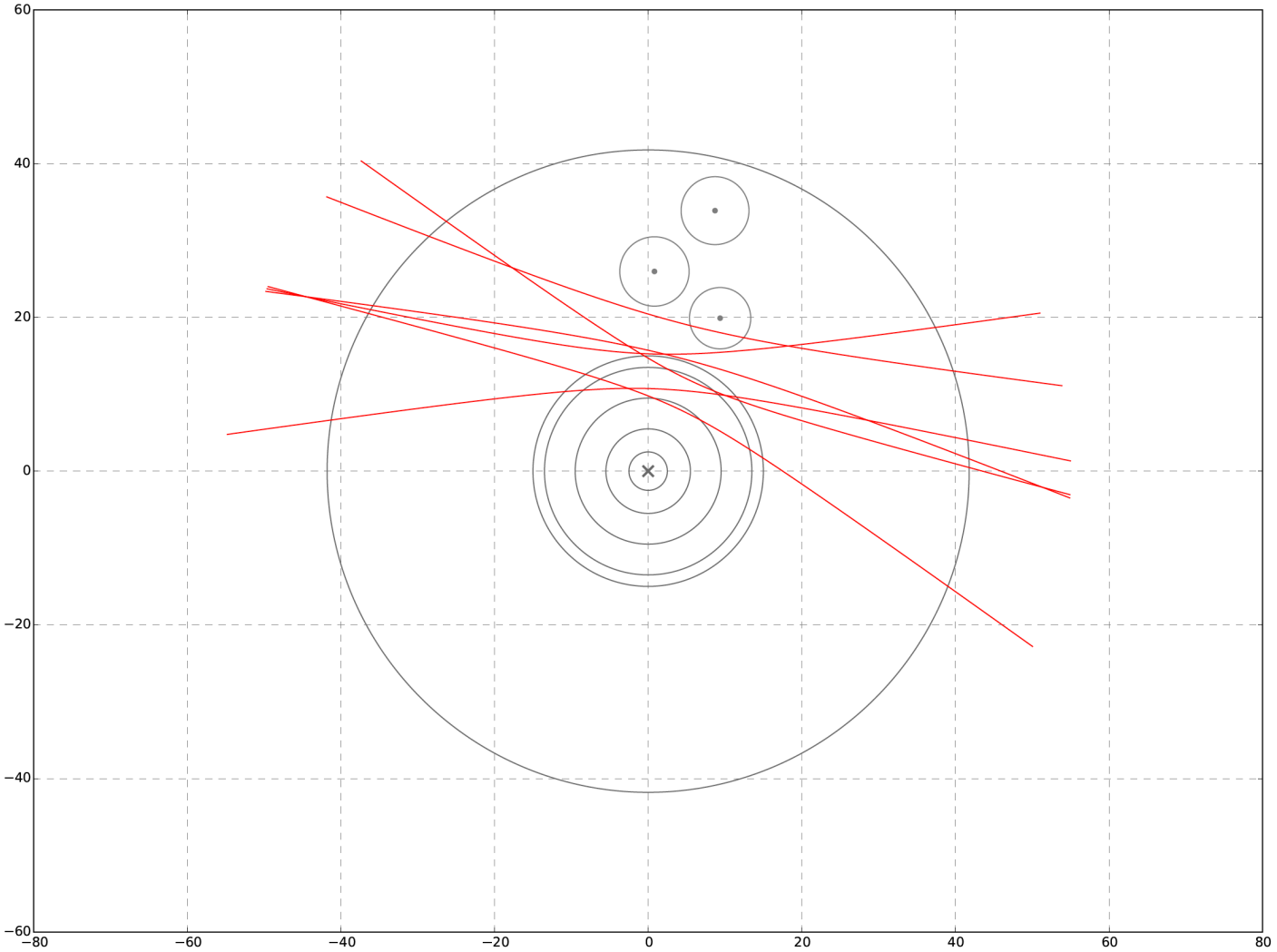
Bezier Curves for Multiple Hits



Bezier Curves for Multiple Hits



Bezier Curves for Multiple Hits



Rasterization

- Draw a figure to a bitmap
- Bresenham algorithm: simple, fast algorithm for rasterization of straight lines
- Adaptable to Bezier curves (directly or via polyline)

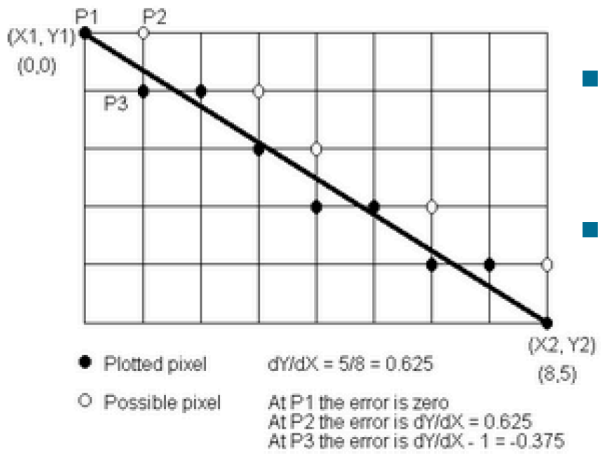


Image credit: http://kobi.nat.uni-magdeburg.de/patrick/pmwiki.php?n=BEng.TheLCDController

- Calculate slope $(y_1 - y_0)/(x_1 - x_0)$
- At each step calculate deviation from ideal line ε
- Only two possible choices:
  - $\varepsilon < 0.5$: increment x, y unchanged
  - $\varepsilon > 0.5$: decrement y instead

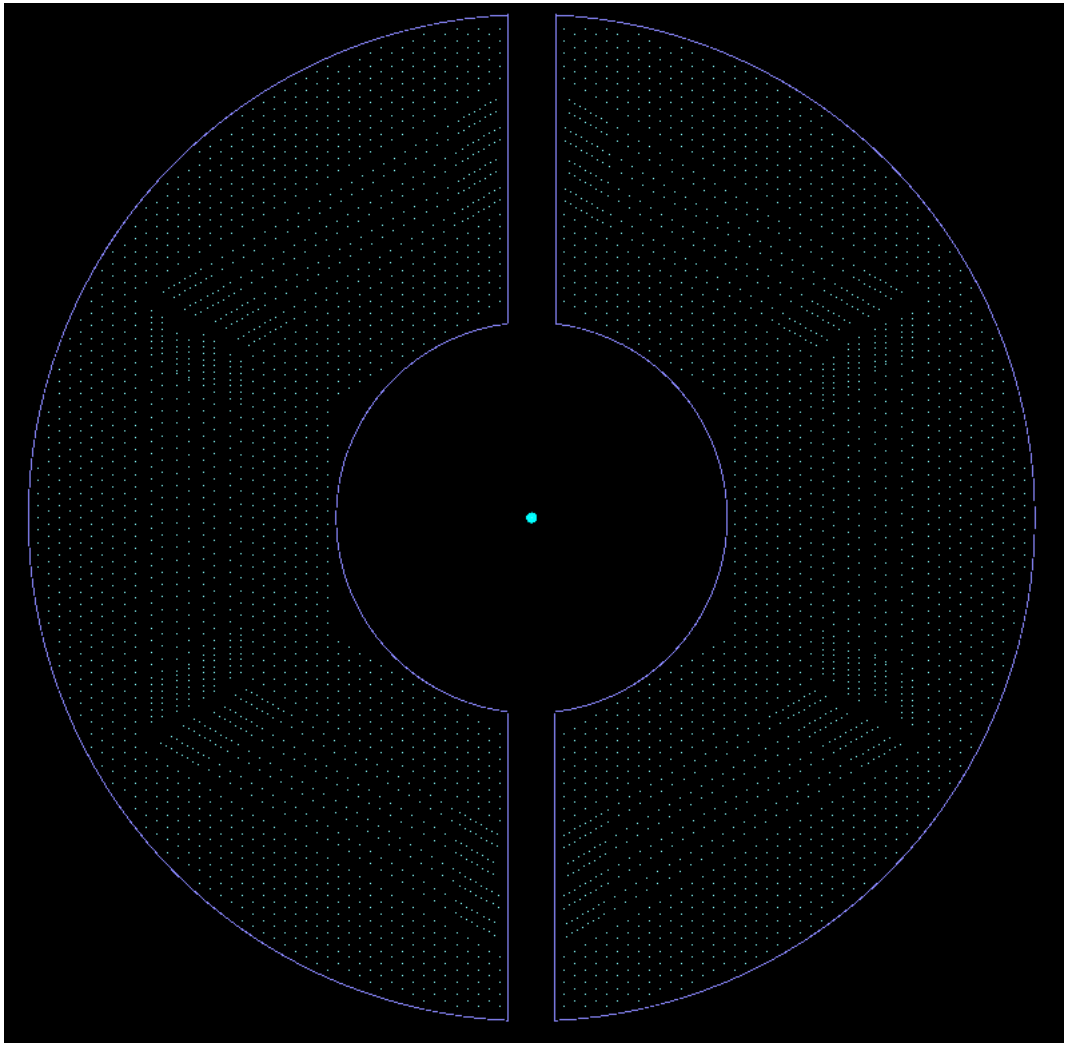
Conclusion & Outlook

JÜLICH FORSCHUNGSZENTRUM

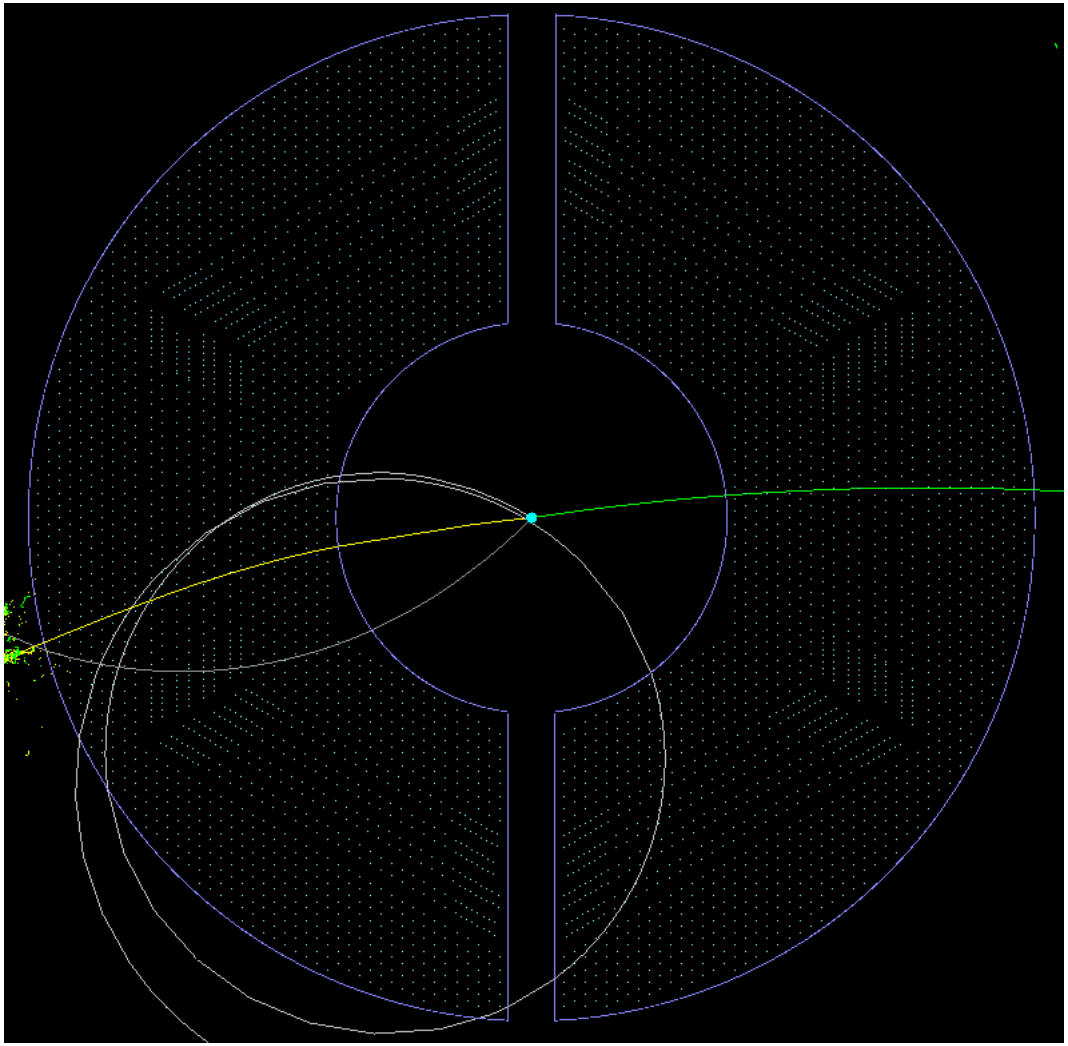
panda

- Circle Hough algorithm for trackfinding
  - Compute all tracks compatible with one hit
  - Combine results from *N* hits
  - Find accumulation of track parameters
- Developed alternative approach based on geometric properties of track centers
  - Parametrize locus (hyperbola) using Bezier curves calculated from hit data
  - Rasterize Bezier curves on accumulator array
  - Perform peakfinding on accumulator array
- Work in progress:
  - ☐ Finalize peakfinding
  - ☐ Write optimized implementation CPU, GPU
  - ☐ Physics studies

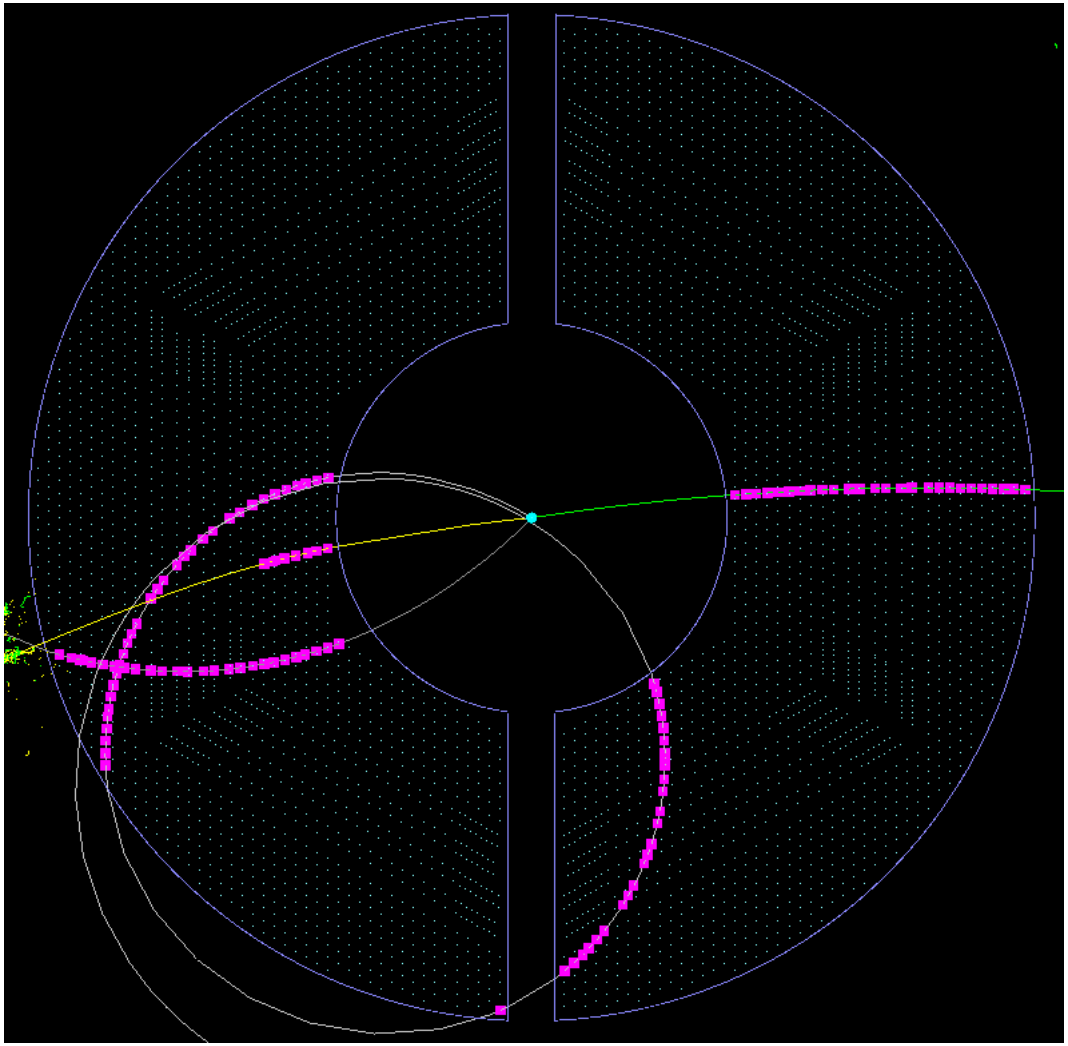
Cellular Automaton



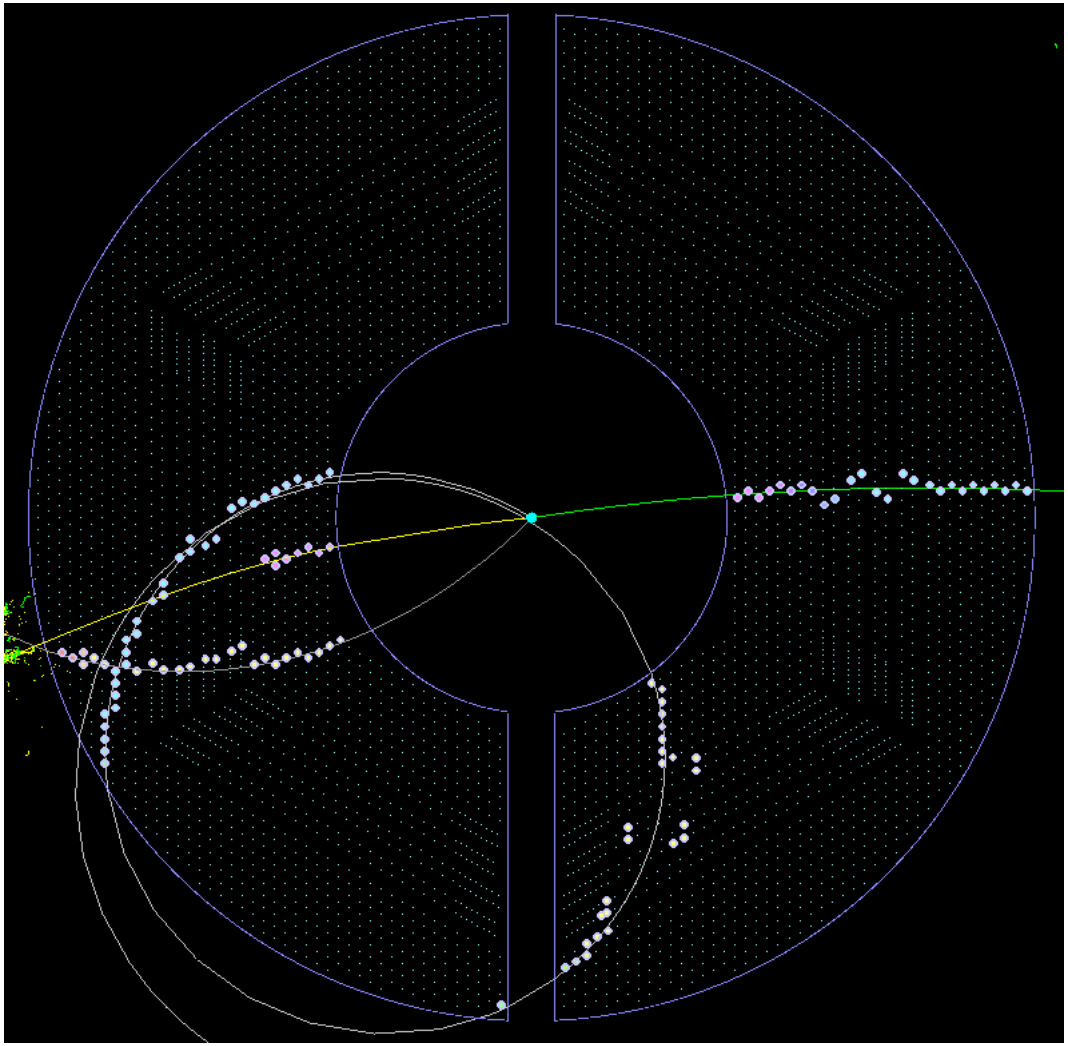
Cellular Automaton



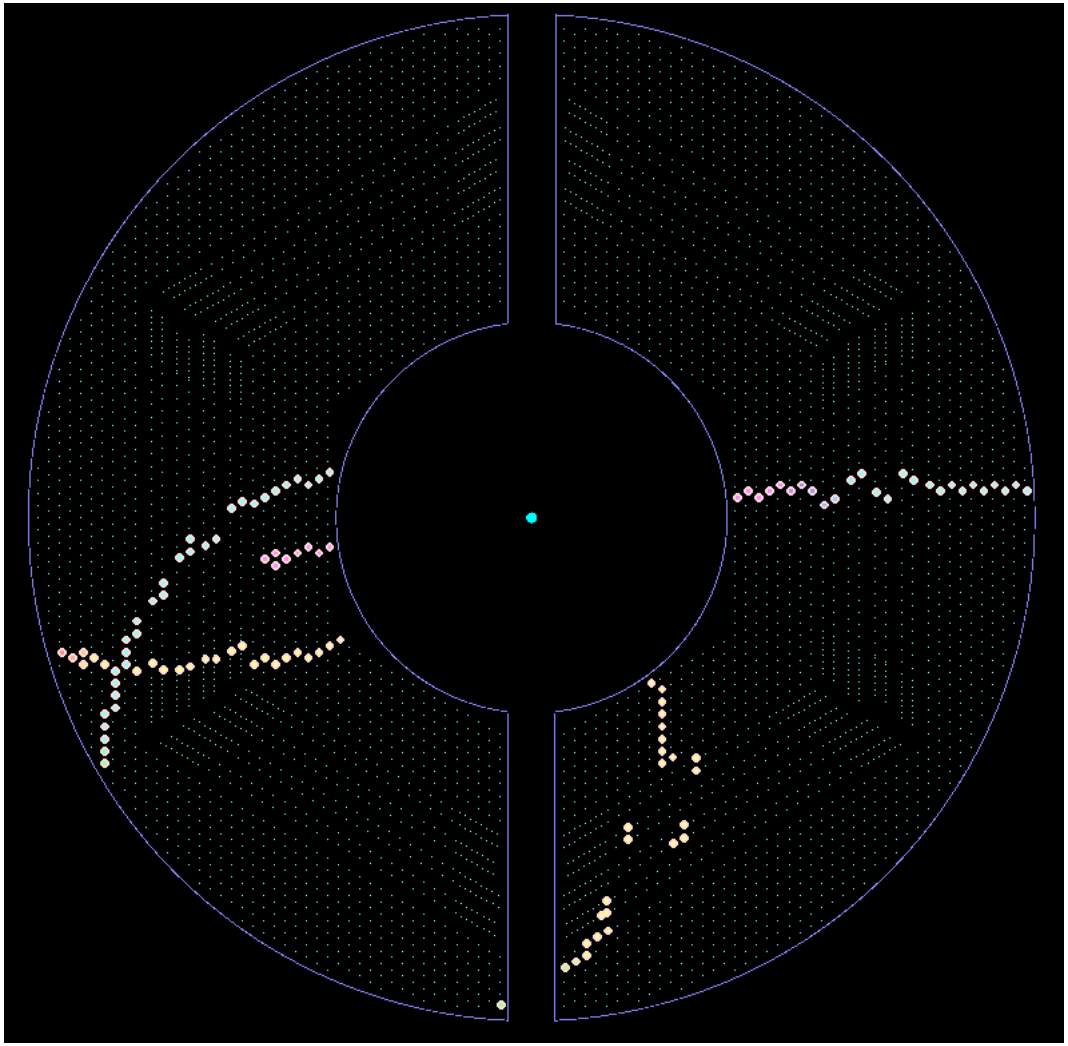
Cellular Automaton



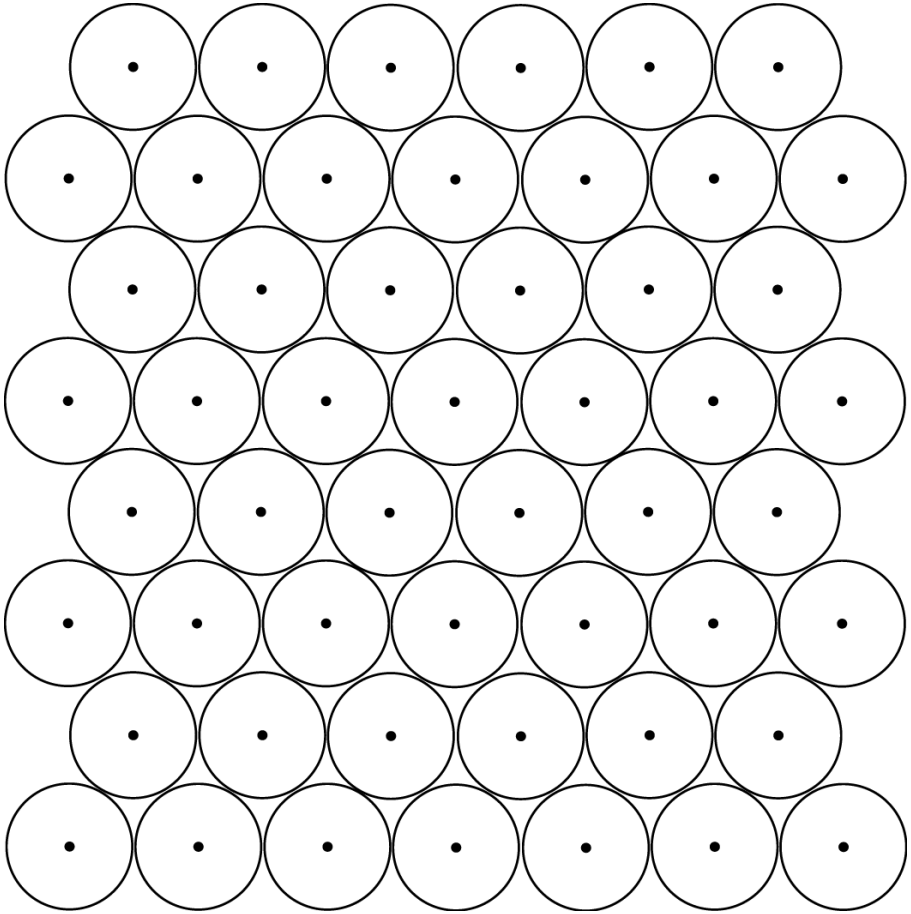
Cellular Automaton



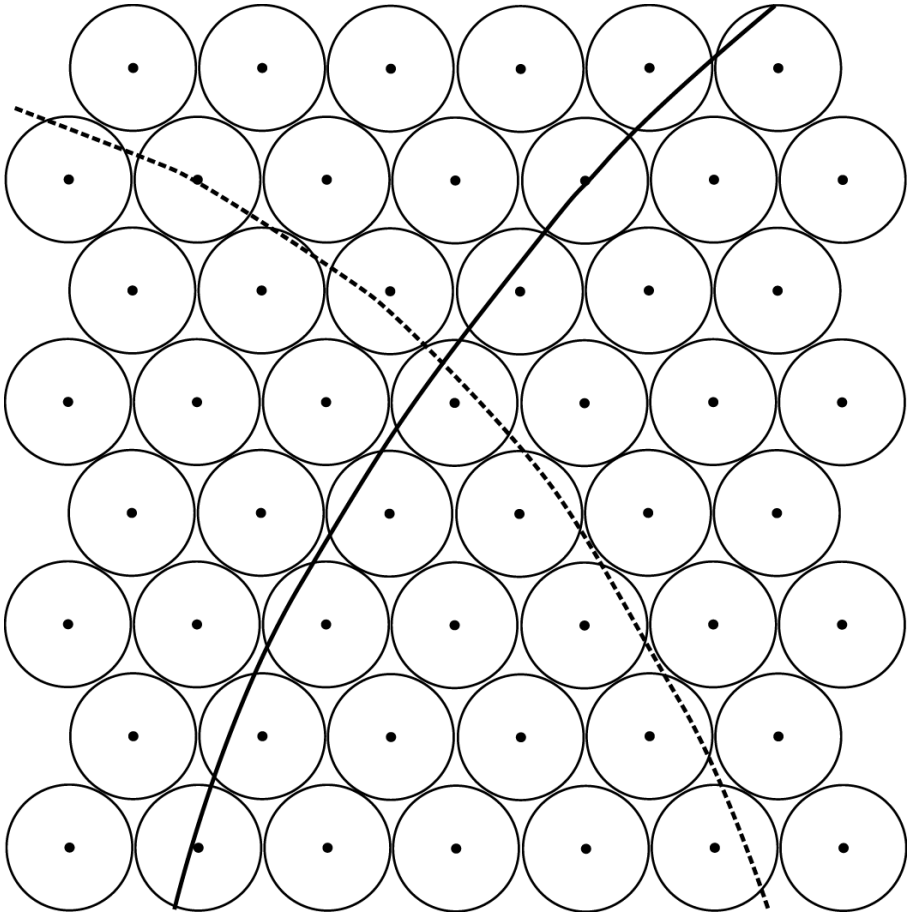
Cellular Automaton



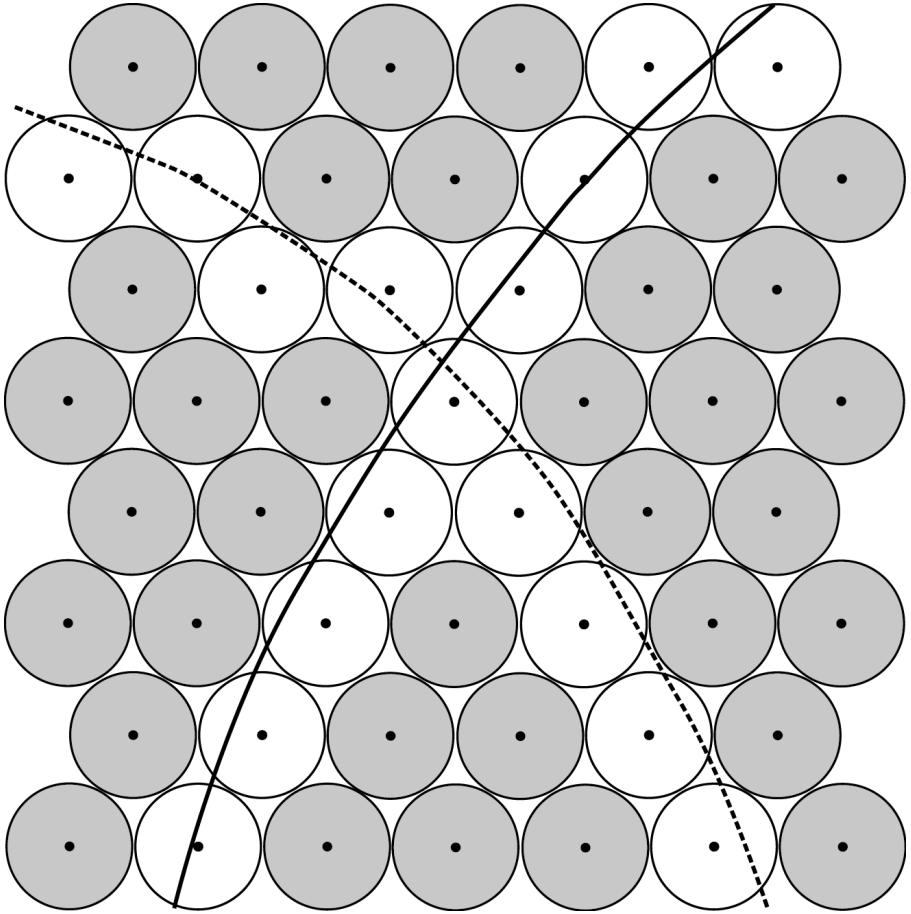
Cellular Automaton



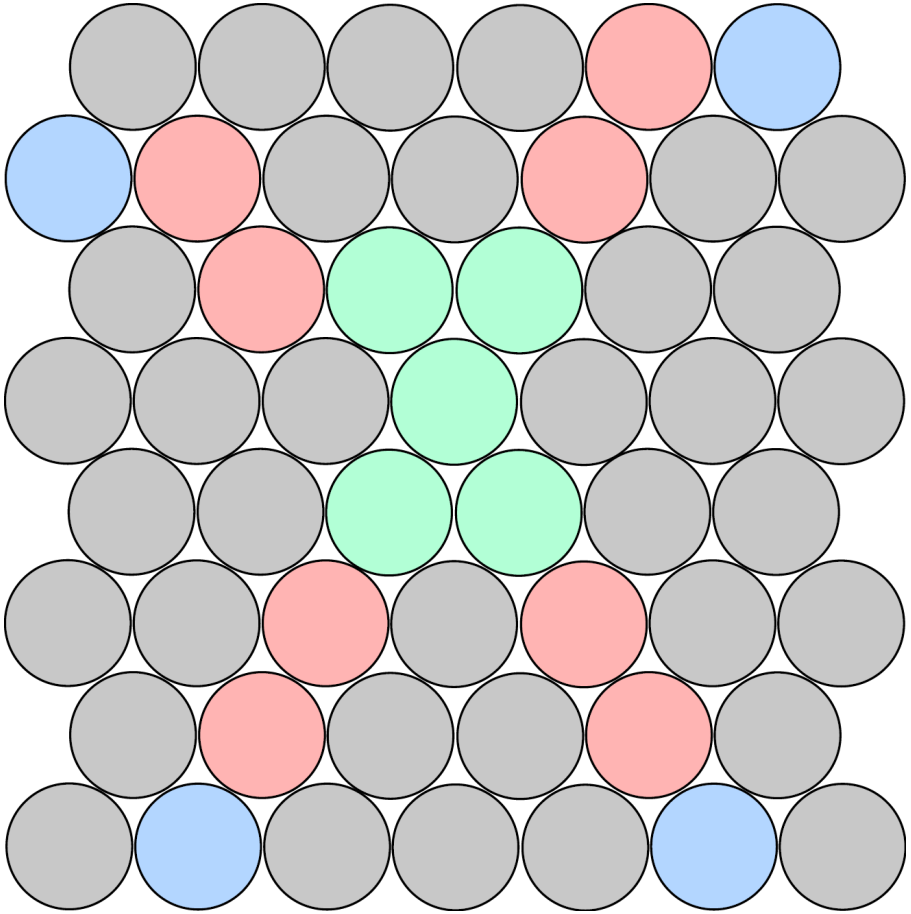
Cellular Automaton



Cellular Automaton



Cellular Automaton



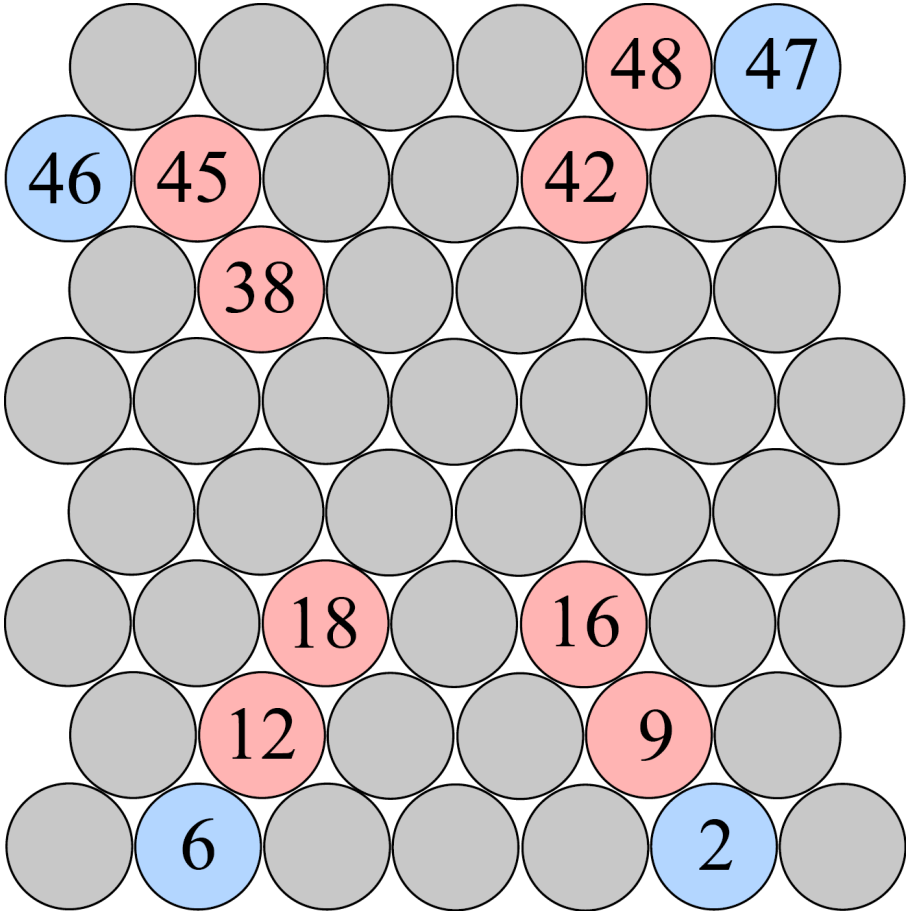
Number active neighbors:

1 unique

2

> 2 ambig.

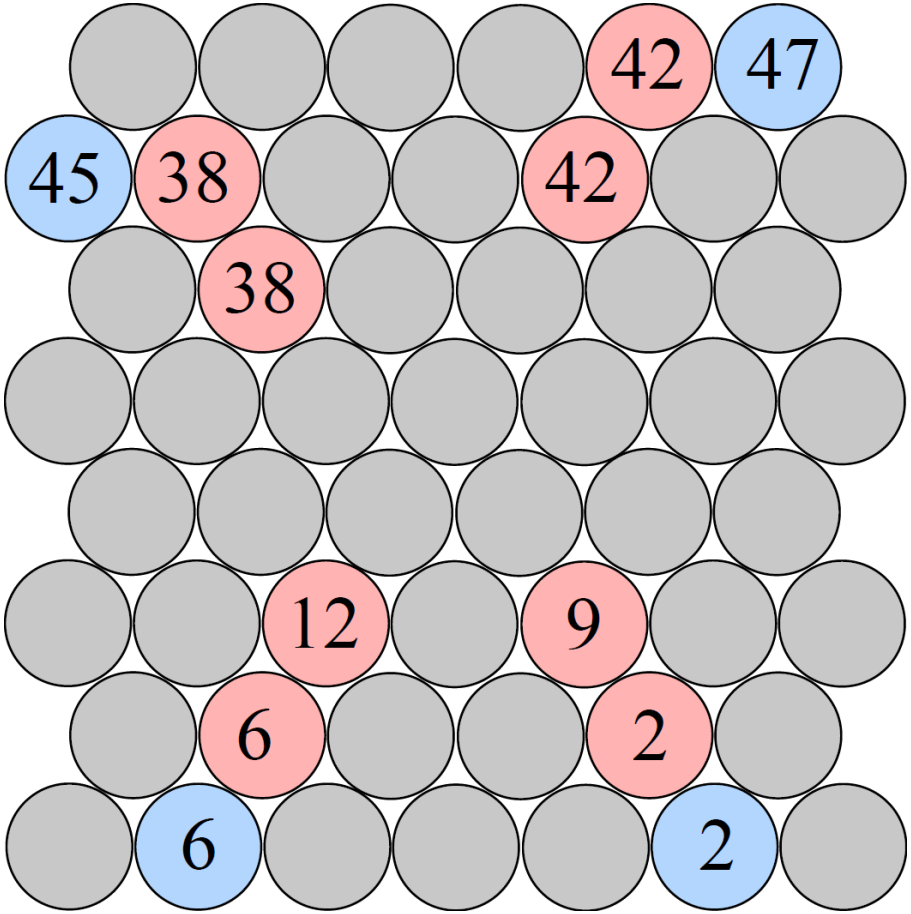
Cellular Automaton



Cell: Tube with one or two active neighbors

Transition rule: New state = minimum of own state and neighbors state

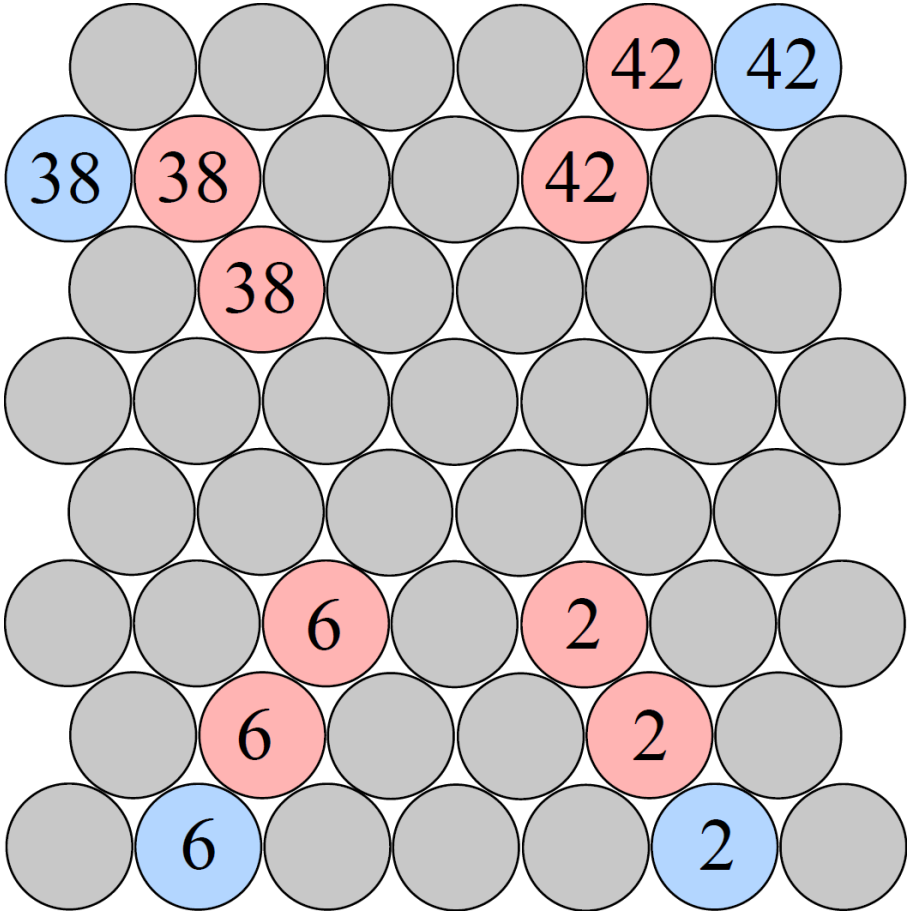
Cellular Automaton



Cell: Tube with one or two active neighbors

Transition rule: New state = minimum of own state and neighbors state

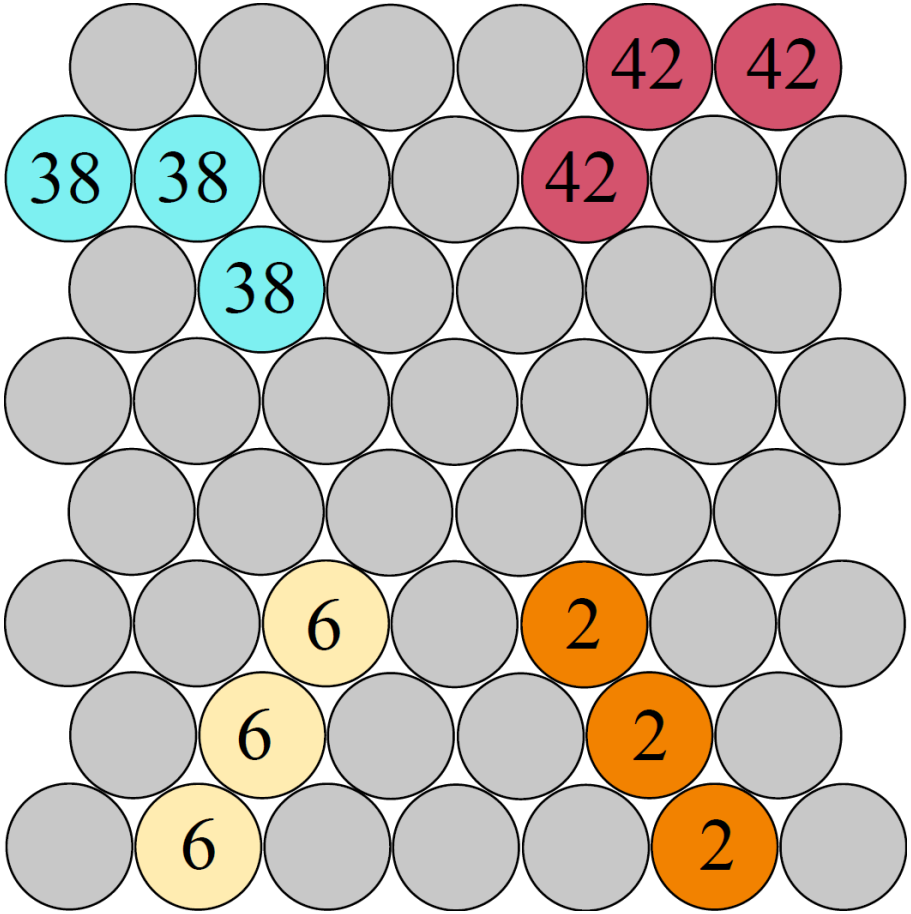
Cellular Automaton



Cell: Tube with one or two active neighbors

Transition rule: New state = minimum of own state and neighbors state

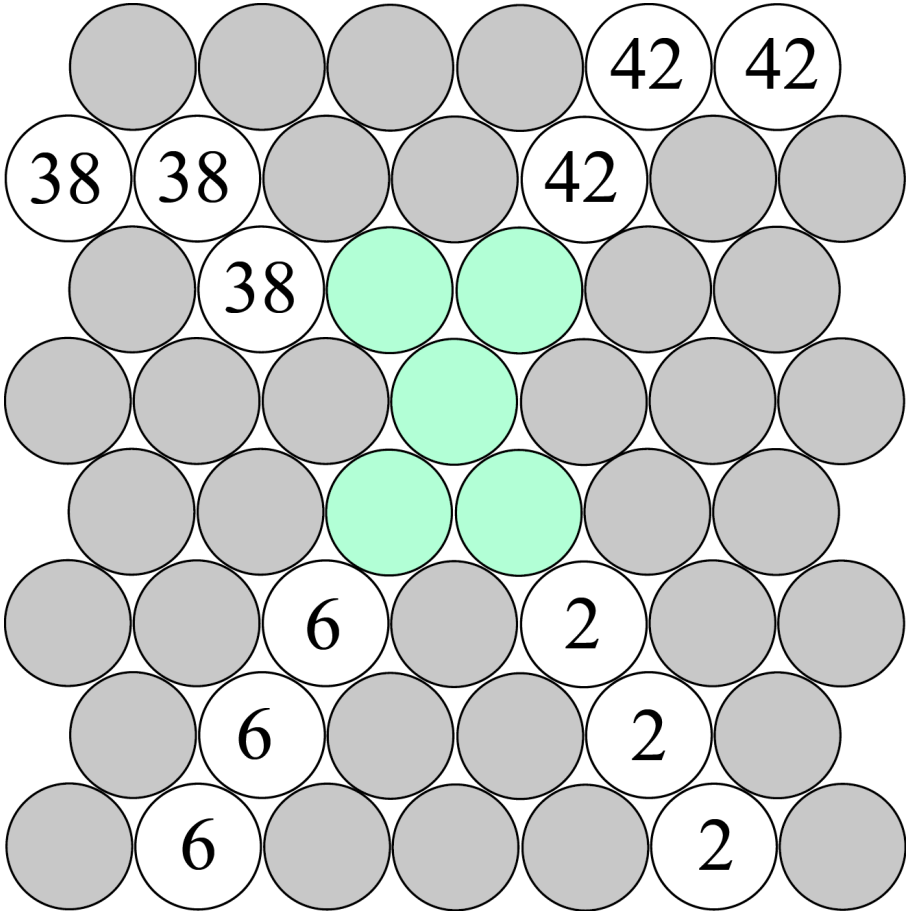
Cellular Automaton



Cell: Tube with one or two active neighbors

Transition rule: New state = minimum of own state and neighbors state

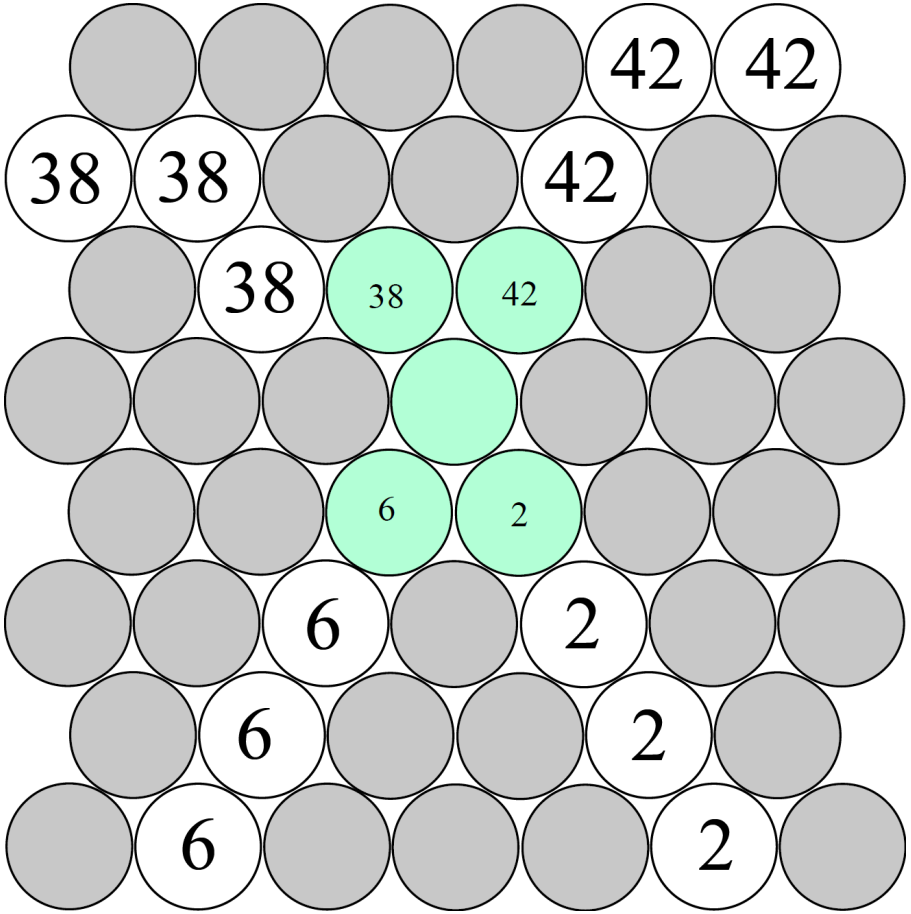
Cellular Automaton



Cell: Tube with more than two active neighbors

Transition rule: New state = All neighbor states

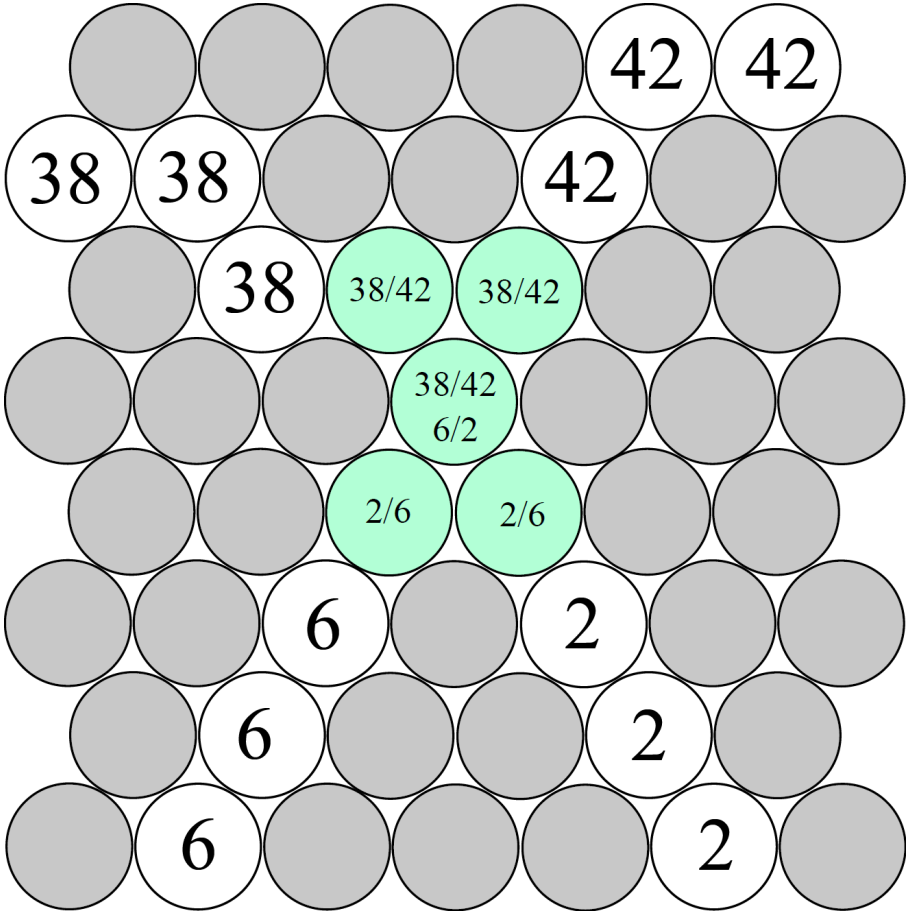
Cellular Automaton



Cell: Tube with more than two active neighbors

Transition rule: New state = All neighbor states

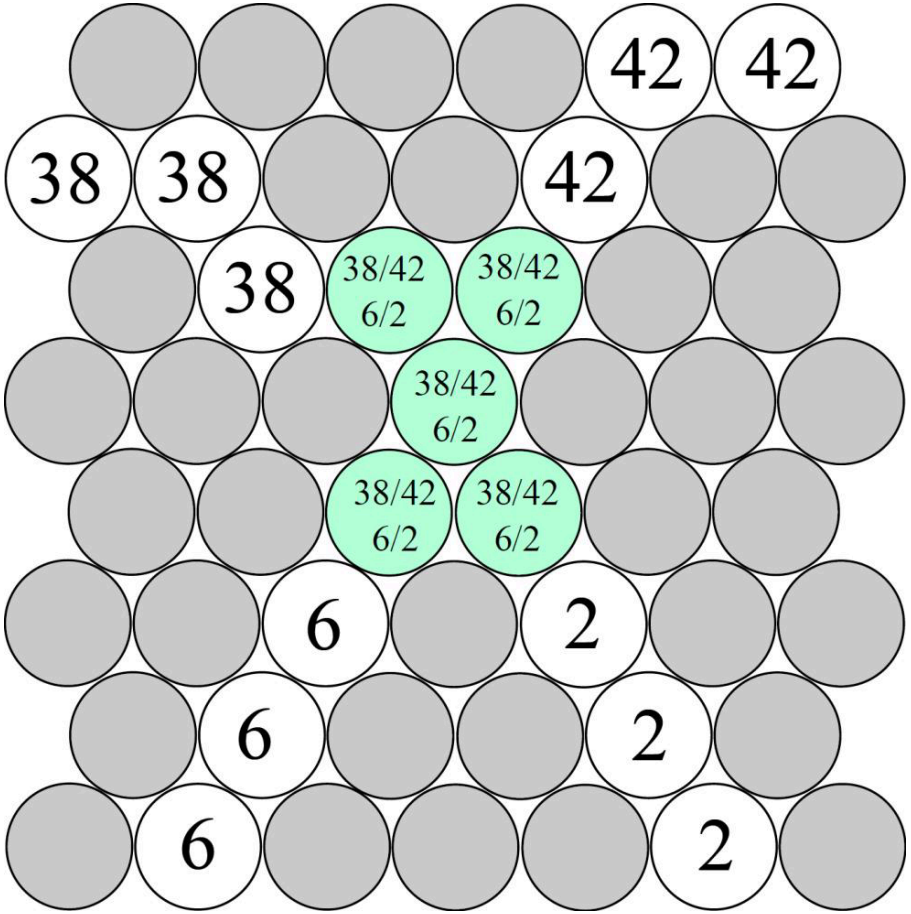
Cellular Automaton



Cell: Tube with more than two active neighbors

Transition rule: New state = All neighbor states

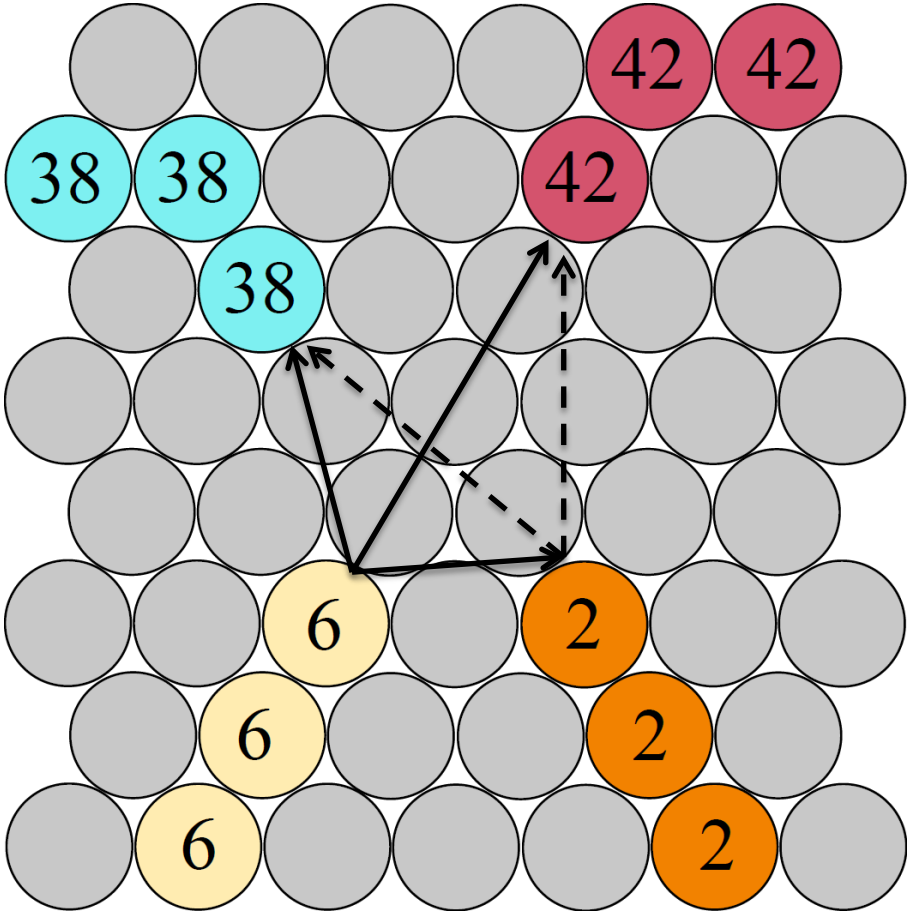
Cellular Automaton



Cell: Tube with more than two active neighbors

Transition rule: New state = All neighbor states

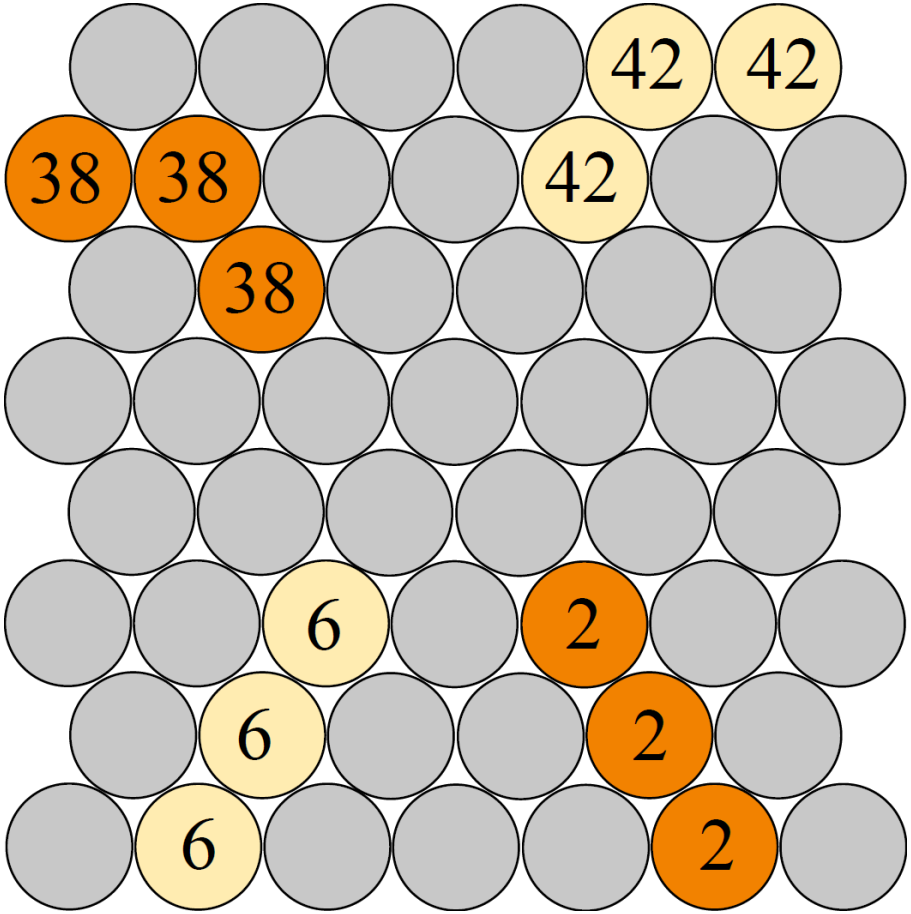
Cellular Automaton



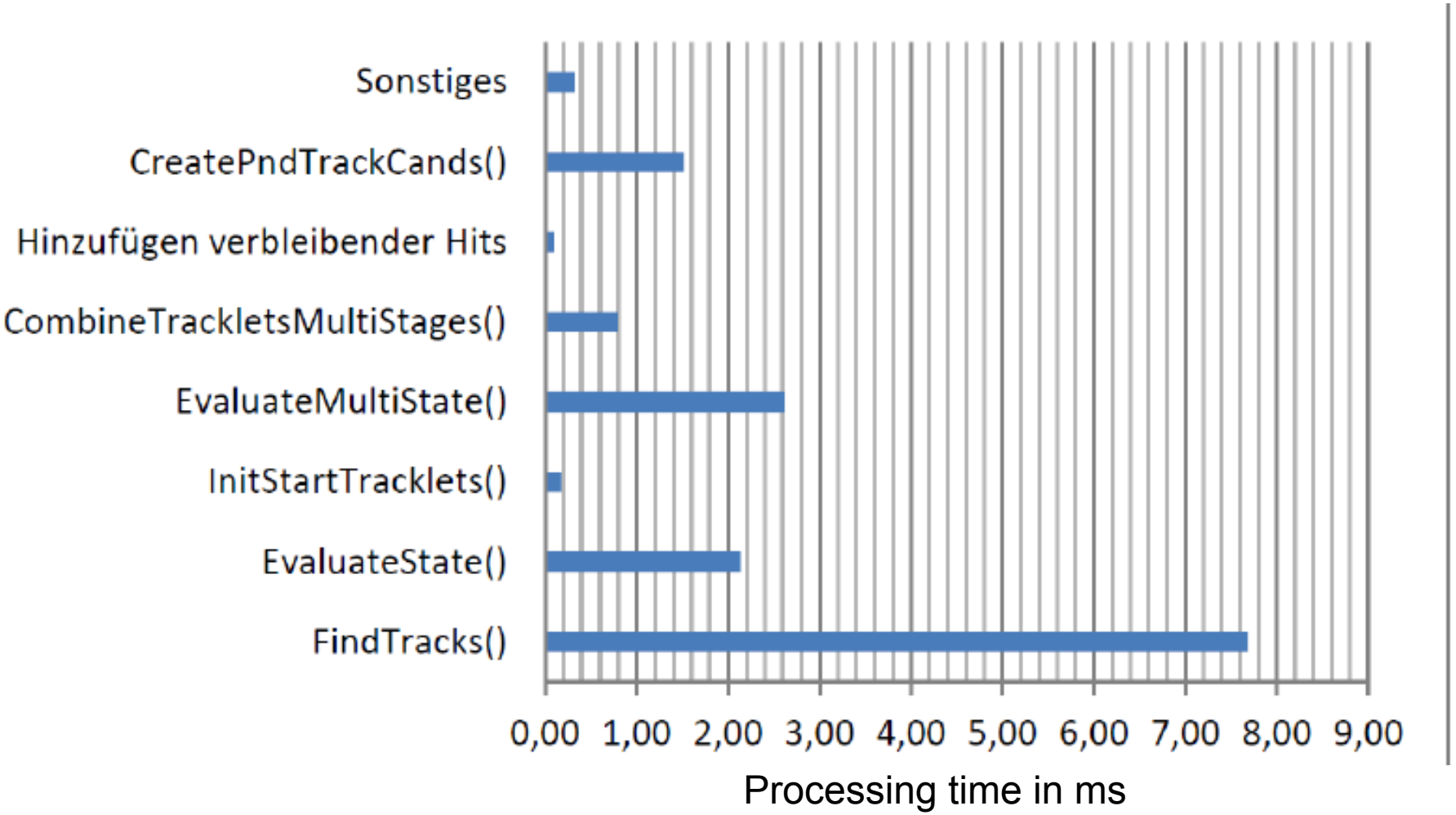
Test all combinations of tracklets with circle fit

→ Choose the best

Cellular Automaton



Average Processing Time per Event



Level of Parallelism

Algorithm Layer

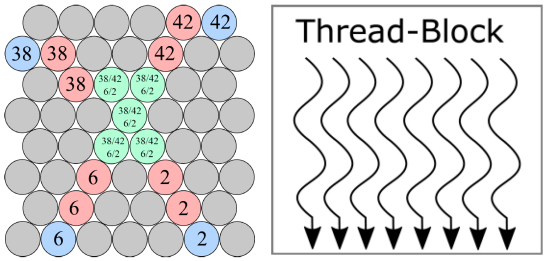
One track finder is processed by several threads

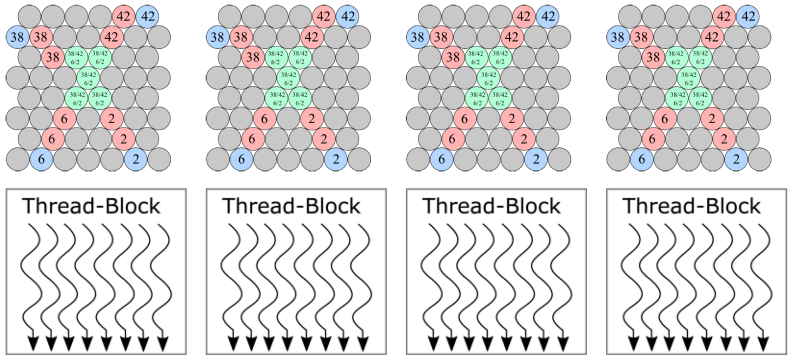
- Parallelize computational intensive part ➔ Cellular Automaton

Event Layer

Several track finders operate in parallel on thousands of events

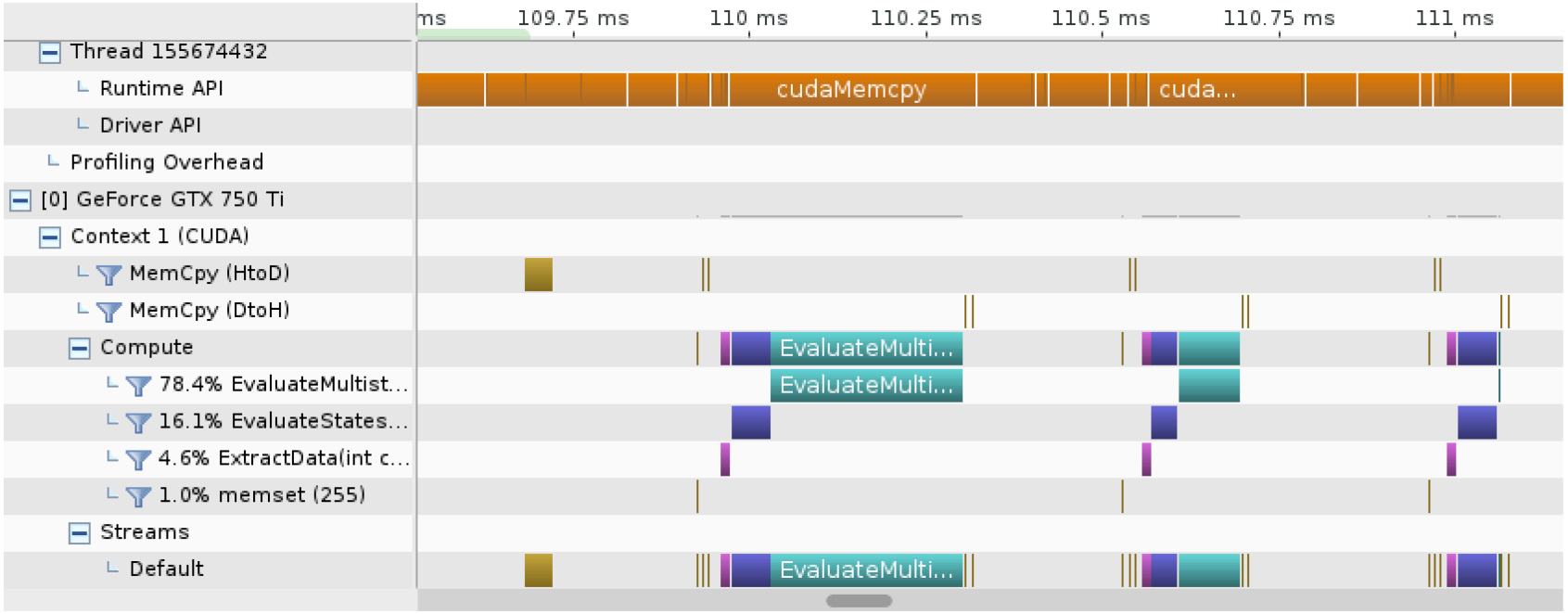
- Large acceleration possible





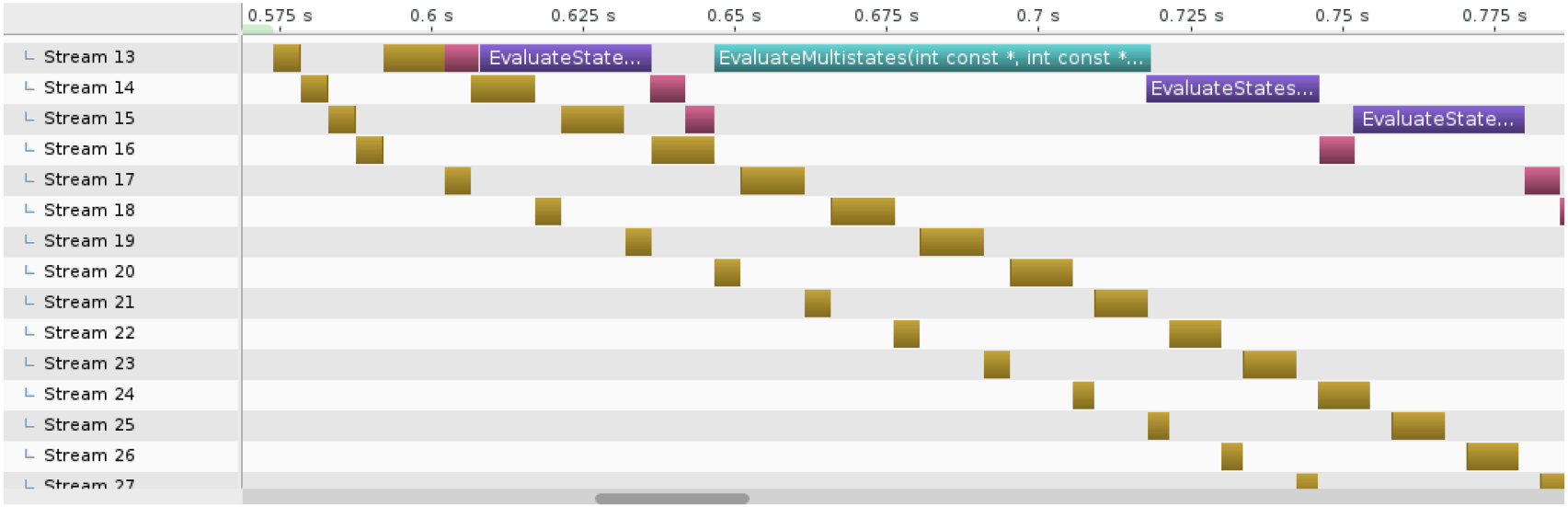
Performance Algorithm Layer

- Start-up configuration: 1 Block, max. 1024 Threads
- Copied data packages are to small to use the maximum bandwidth
- Workload of kernels low



Performance Event Layer

- Start-up configuration: 3000 blocks, 1024 threads
- Efficient copying
- Complex kernels
- Limitation to 60,000 events due to memory capacitance of used GPU



Summary Performance Analysis

- Parallelization on algorithm layer is not sufficient
- Parallelization on event layer uses the GPU efficient
- Usage of 1000 GPUs (GeForce GTX 750 ti)) would speed up the processing to cope with event rate

Track finder	Run time to generate states	Speedup
CPU	4.76 ms	-
GPU Algorithm	0.575 ms	8
GPU + event layer	0.045 ms	105

Summary

- Huge raw data sets generated in HEP experiments
- Amount of data way too big for permanent storage and most of the data not interesting ➔ only store physically interesting data
- New approach: Trigger less readout
- Online processing of data in real time necessary
- Tracking of charged particles key requirement

- GPUs one possible hardware solution to process data
- Optimization of algorithms for GPUs not trivial
- Needed:
  - Efficient data flow handling
  - Scalability