Tracking

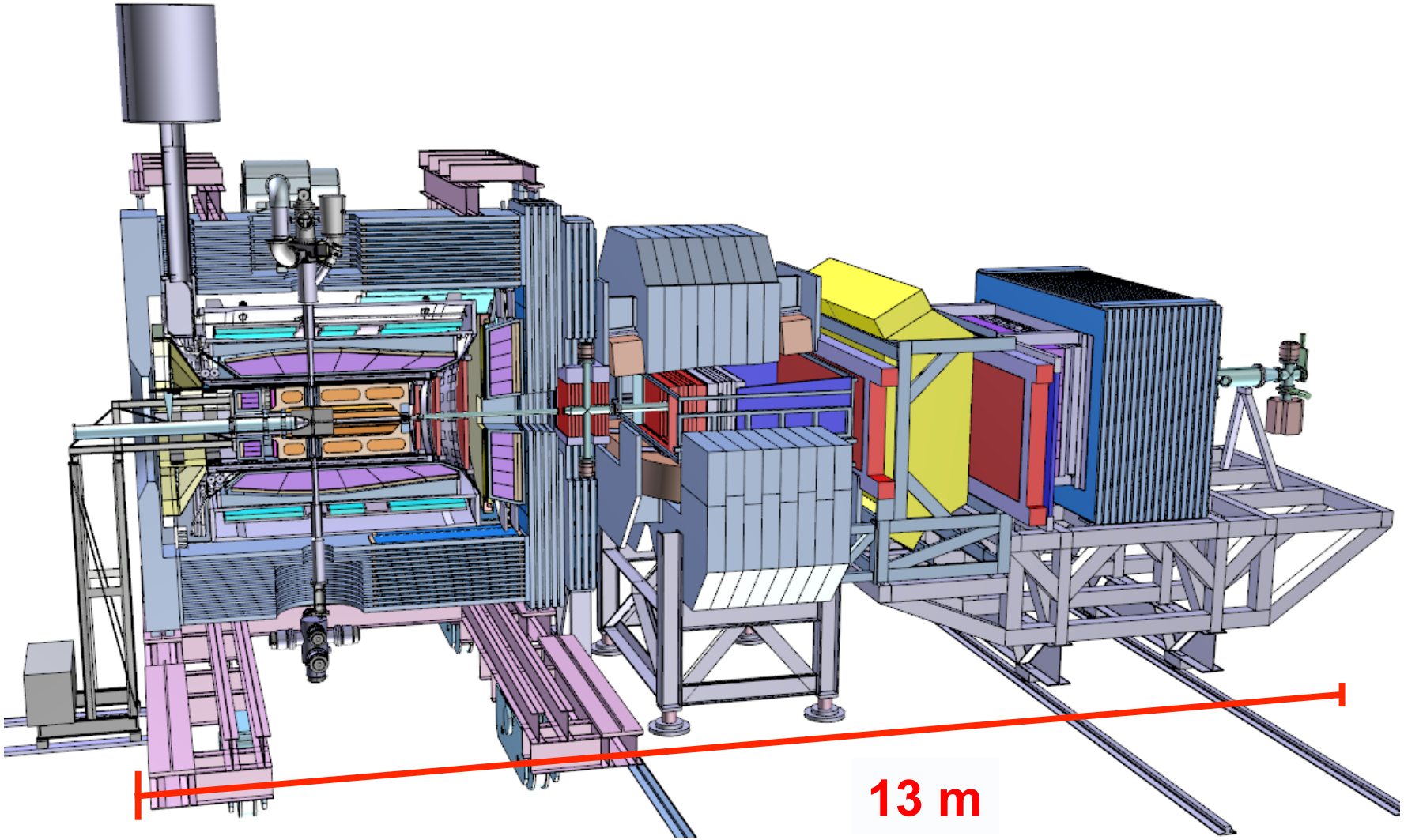
Juli 4, 2017

Motivation

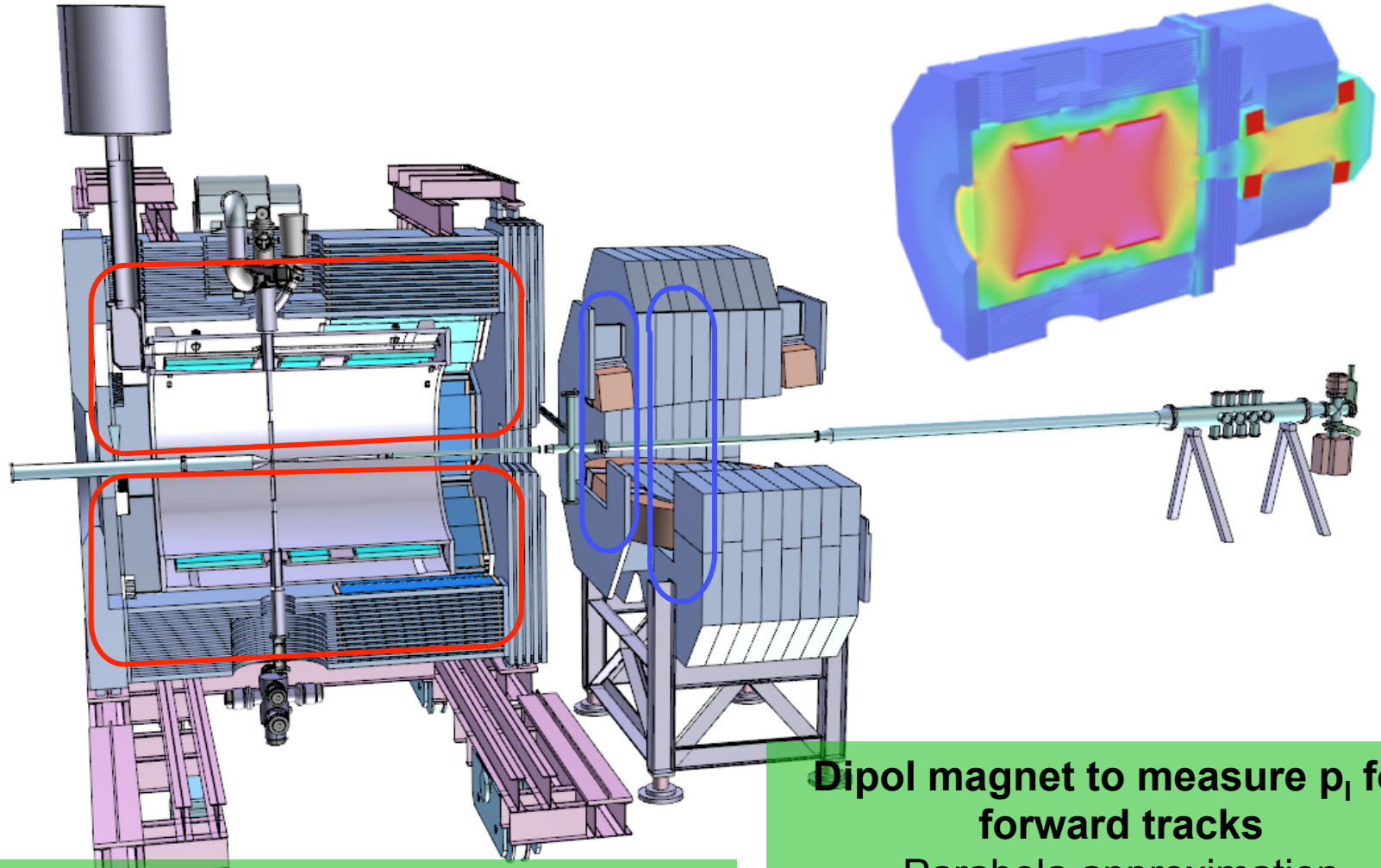
- Why tracking?
  - Distinguish charged from neutral particles
  - Determine the charge of a particle
  - Determine the absolute momentum
  - Determine the direction of momentum
  - Determine the production/decay point
  - See the full story

AntiProton Annihilation at Darmstadt

JÜLICH FORSCHUNGSZENTRUM



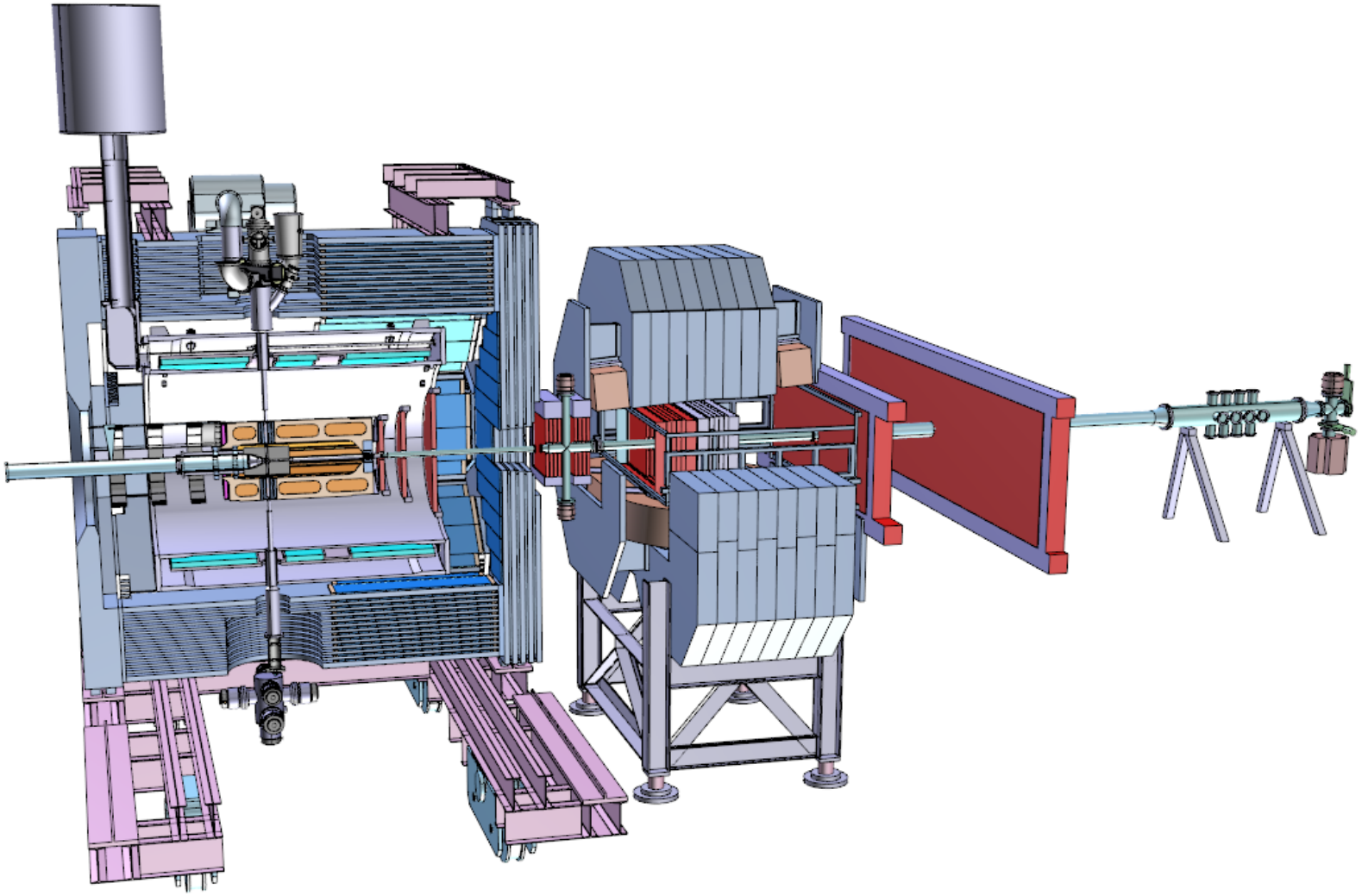
Magnet System



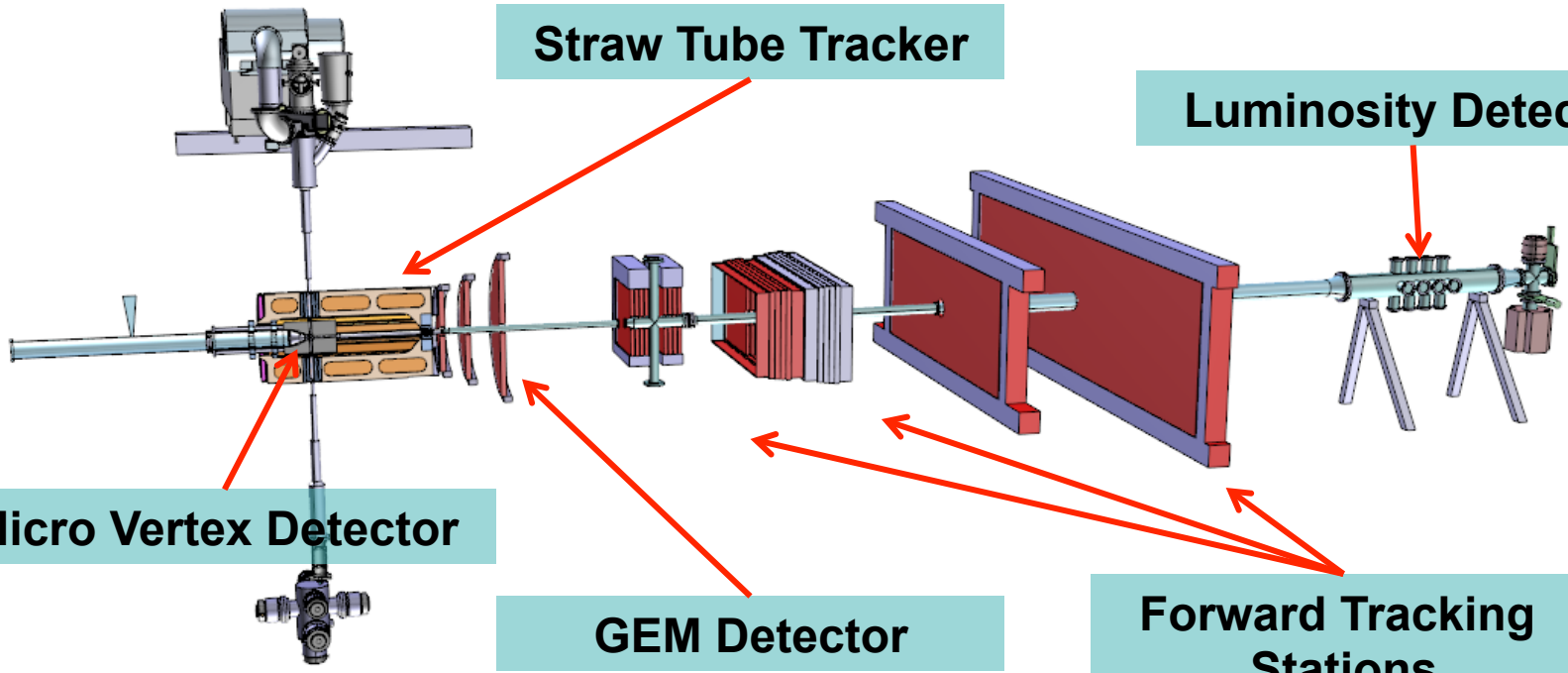
Solenoid magnet to measure p_t
Helix approximation

Dipol magnet to measure p_l for forward tracks
Parabola approximation

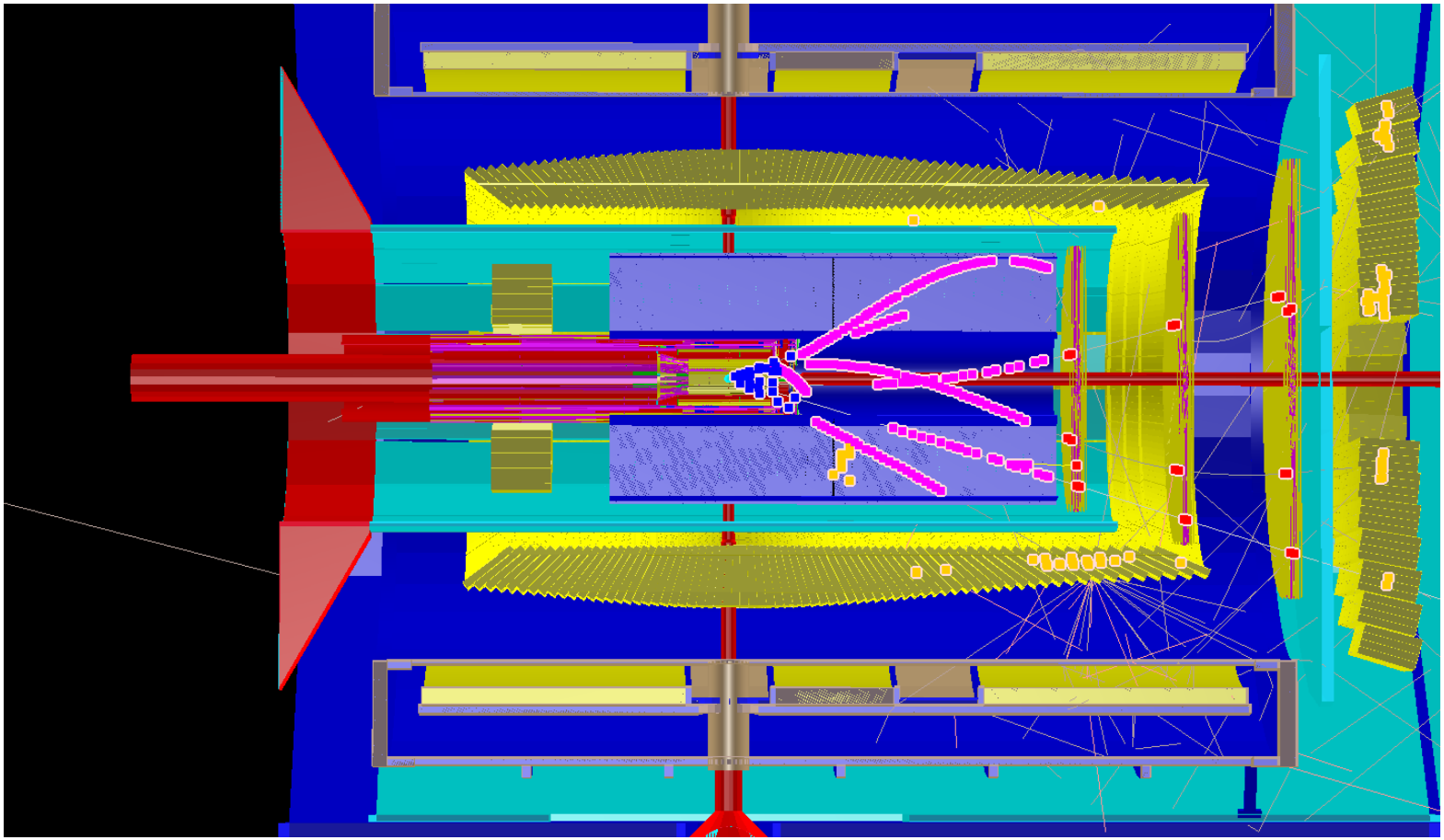
Spurdetektoren



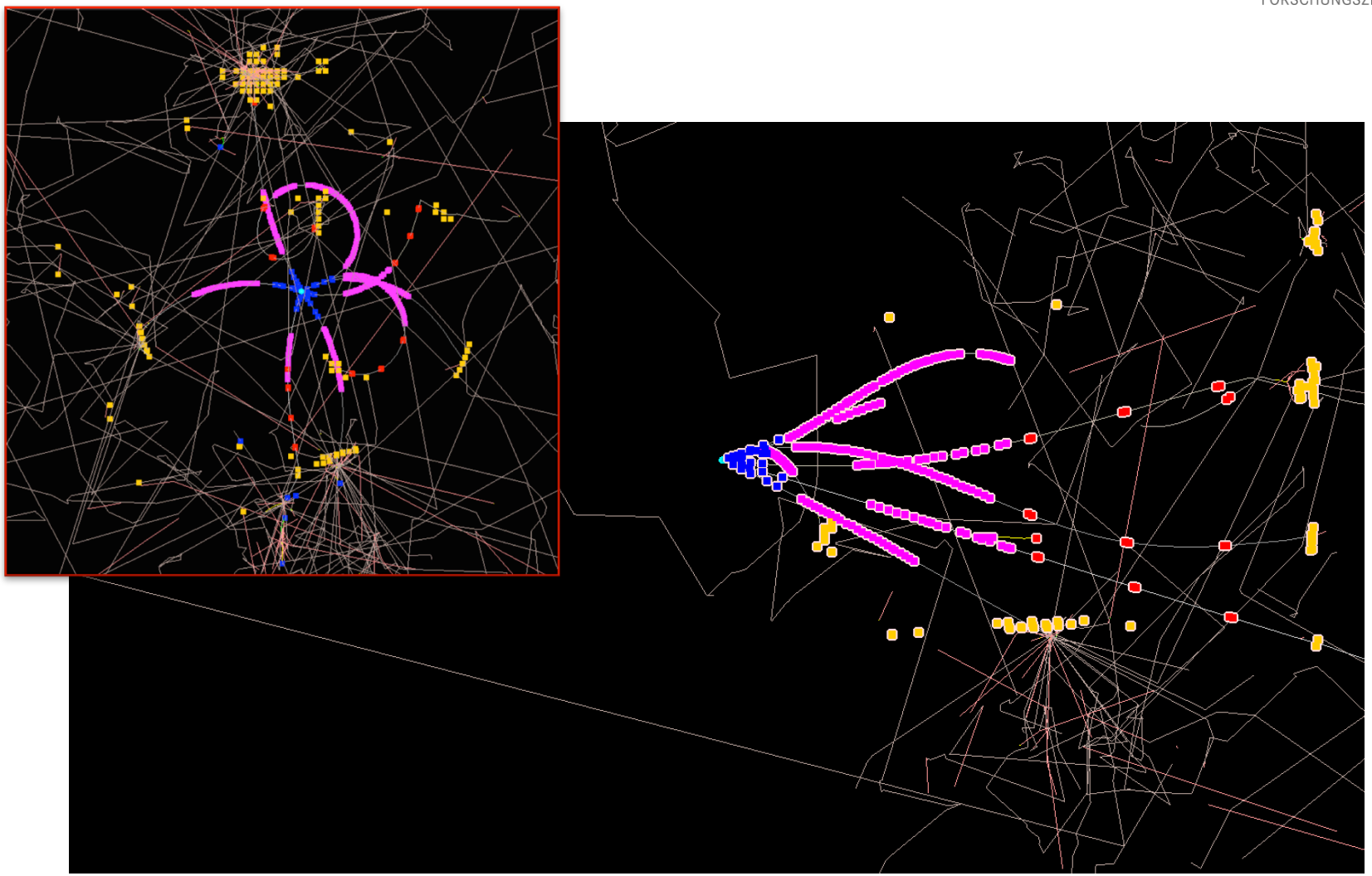
Tracking Detectors



PANDA Event Simulation



PANDA Event Simulation



Procedure

- Full track reconstruction happens in various steps:

1.	Group the hits which belong to the same track	pattern recognition / track finding
2.	Fast evaluation of the track parameters	pre-fit
3.	Precise fitting of the track parameters	Kalman fit
4.	Propagate track parameters to the point where they are needed	propagation / extrapolation

The Data

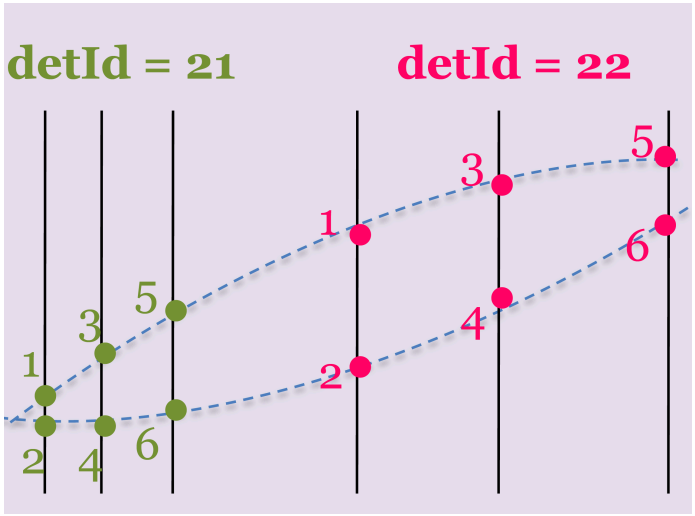
- The following objects contain the tracking information
- They “live” in pnddata/TrackData

  - `PndTrackCand`
    - *Collection of hits inside a track*
    - *Output of track finder*

  - `PndTrack`
    - *Parameters of track*
    - *Output of prefit and Kalman fit*

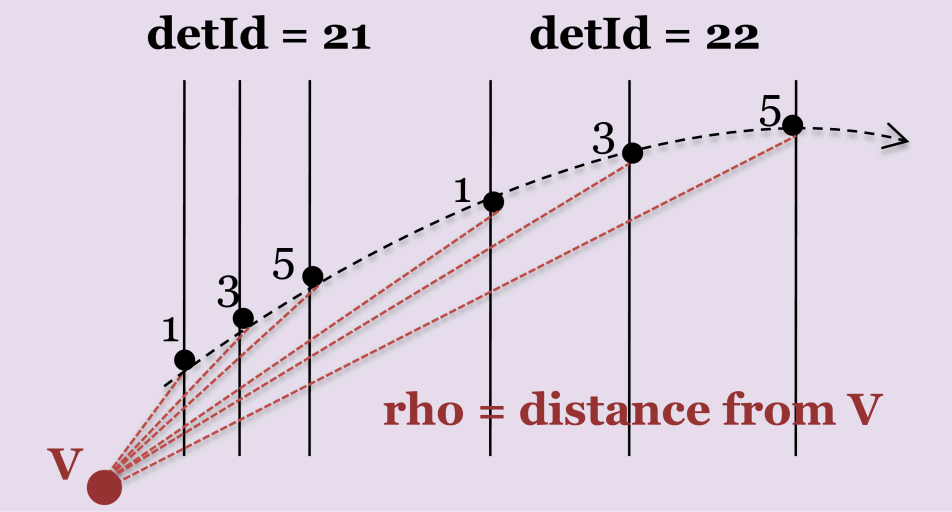
PndTrackCand

- Is a collection of hits
- Each hit is represented by a `PndTrackCandHit` which is a `FairLink(detId, hitId)` plus a sorting parameter rho.
- detId is the branch ID of the detector who generated the hits
- hitId is the position in the TClonesArray



- Rho is an sorting parameter of a hit along the track
- No fixed rule how to calculate rho

PndTrackCand - Rho

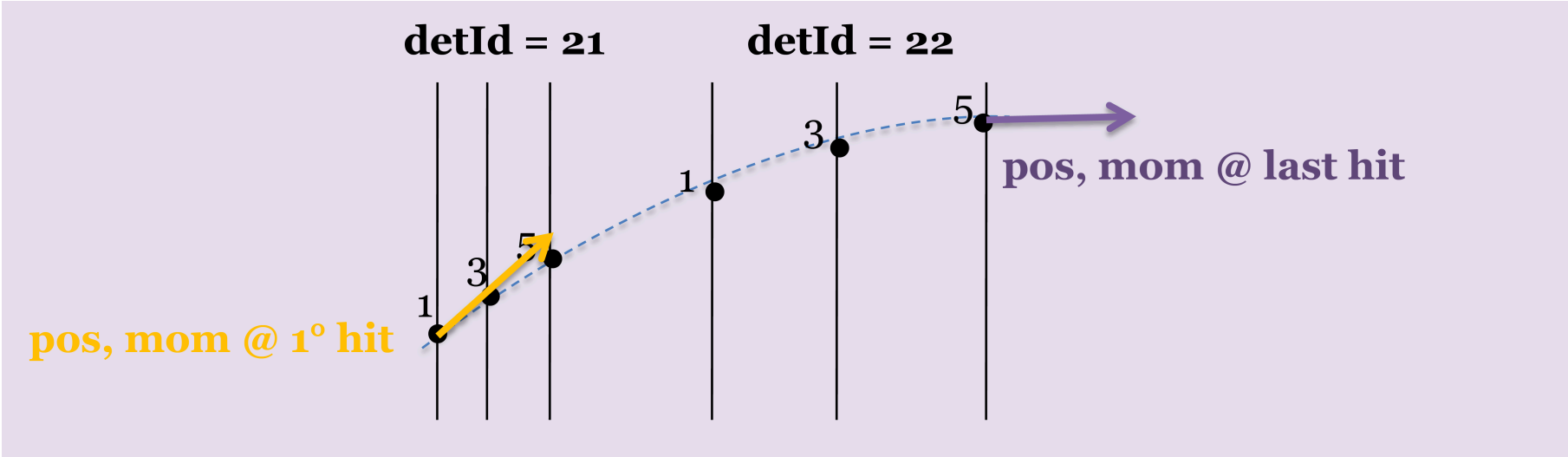


- Possible candidates for Rho
  - Distance from vertex (not a good one)
  - Time (but time resolution usually not sufficient)
  - Length along track (arc length for helix)
  - Secondaries: Where is the start of the track?

PndTrack

- This is the hypothesis of a track
  - Identified by the momentum, charge and position at the first and last hit
  - `PndTrackCand` accessible via `PndTrack`

  - Why two times the track information?



Data Summary

We have two objects to describe a track:

- `PndTrackCand` which contains the list of hits in the track
- `PndTrack` which contains the parameters of the track

- Why two objects instead of one?

Data Summary

We have two objects to describe a track:

- `PndTrackCand` which contains the list of hits in the track
- `PndTrack` which contains the parameters of the track

- Why two objects instead of one?
  - `PndTrackCand` can be used by different algorithms or different particle hypotheses

TRACK FINDING

Track Finding Strategies

- local
  - operates on the data of one sub-detector
  - finds all possible tracks in this sub-detector
  - tracks in different sub-detectors are combined via their track parameters
  - refit of complete track at the end

- global
  - operates on the data of more then one sub-detector at once
  - finds a complete track

Track Finding Strategies

- local vs. global
  - local
    - *optimized algorithms for each sub-detector*
    - *no need to wait until data from all sub-detectors is available*
    - *several different algorithms needed (for each sub-detector and to combine the data)*
    - *how to handle tracks with insufficient hits in one sub-detector?*
  - global
    - *all information at hand from the beginning*
    - *just one (but big) algorithm*
    - *treatment of different kind of data difficult (straws vs. pixel)*

Track Finding Strategies

Additional considerations:

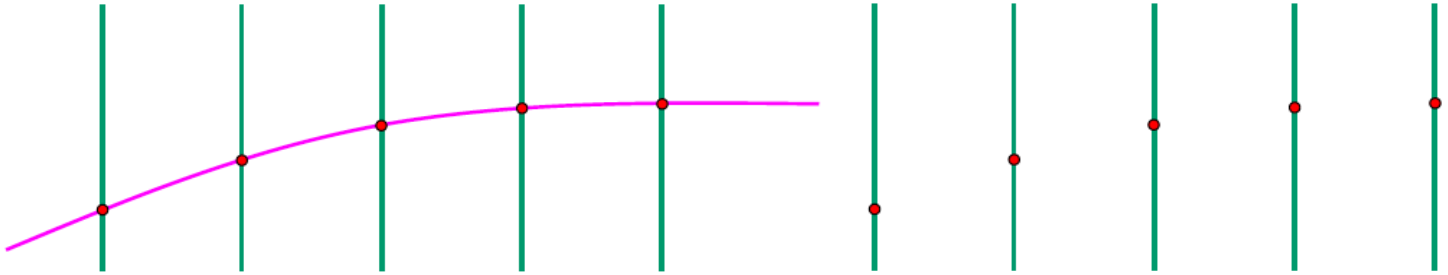
- All at once ←→ Iterative
- Online ←→ Offline
  - Performance vs. speed
- Primary ←→ Secondary
  - Primary: coming from the interaction point
  - Secondary: all the rest
  - Primary tracks more easy to find
  - Secondaries important e.g. for hyperon physics

Track Finding

JÜLICH FORSCHUNGSZENTRUM

Considerations

- Finite number of planes



A track goes through, leaving hits

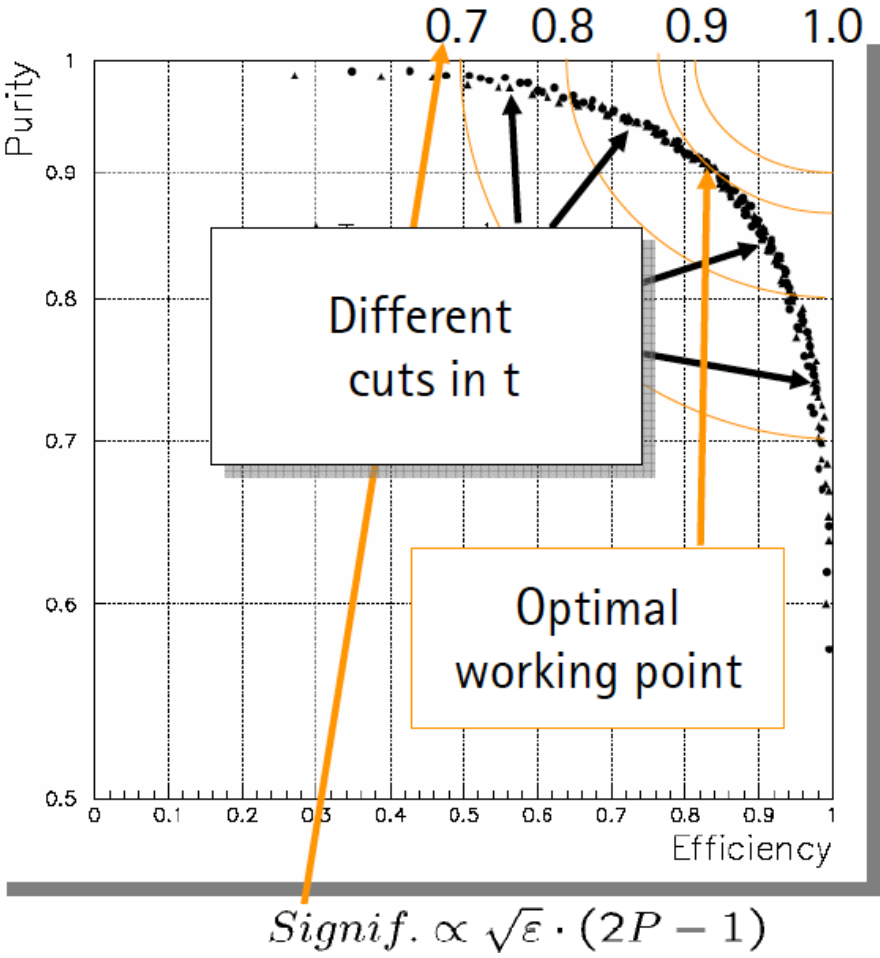
All you really see are the hits, actually

- Often only one coordinate per plane
- Interactions with material
  (multiple scattering, energy loss, Bremsstrahlung for e-)
- Detector imperfections
  (inefficiency, limited spatial resolution, alignment)
- Extra hits from noise, pileup, low momentum particles…

Track Finding

Conflicting Demands

- High efficiency
  find all true tracks
  relevant for the physics
- High purity
  minimize incorrect assign-
  ment of hits to a track,
  that reduces the resolution



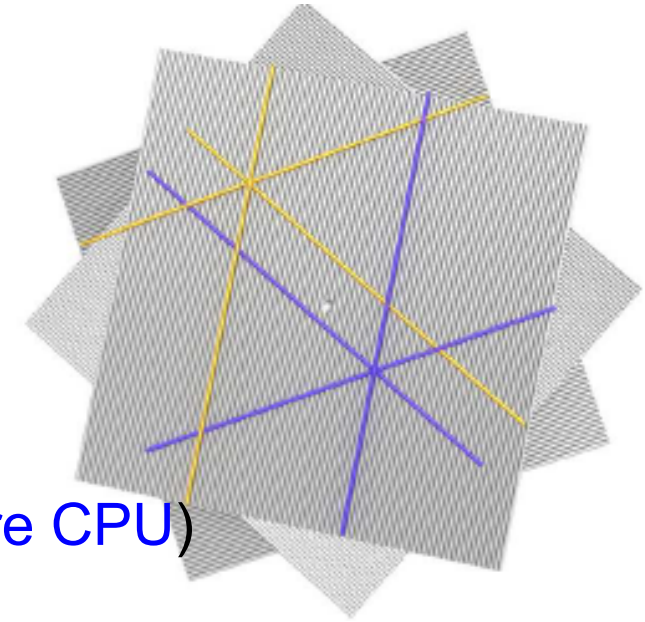
- Low ghost rate
  do not find fake tracks
- Speed
  often online tracking is needed as part of a trigger decision
  (limited time (latency) and resources).

Track Finding with 1D Detectors

JÜLICH FORSCHUNGSZENTRUM

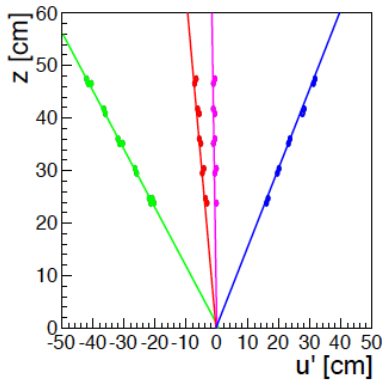
Tracking in space
(generally needs many layers)

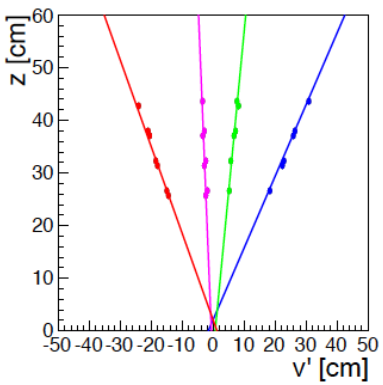
- Combine layers to get a 3D space point,
- do 3D tracking with these points

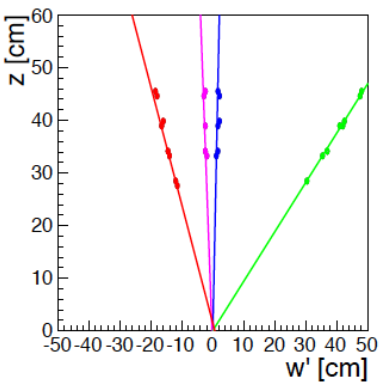


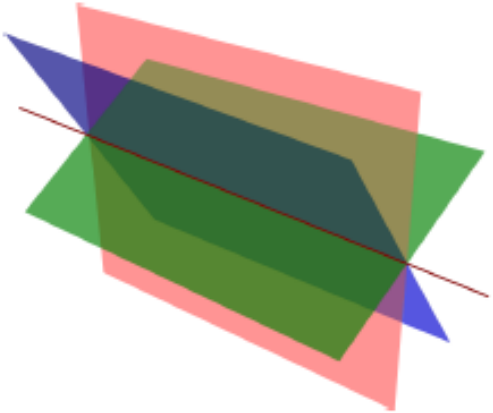
Tracking in projections (generally needs more CPU)

- find 2D track candidates, then
- combine projections (here each line in 2D is a plane in 3D)





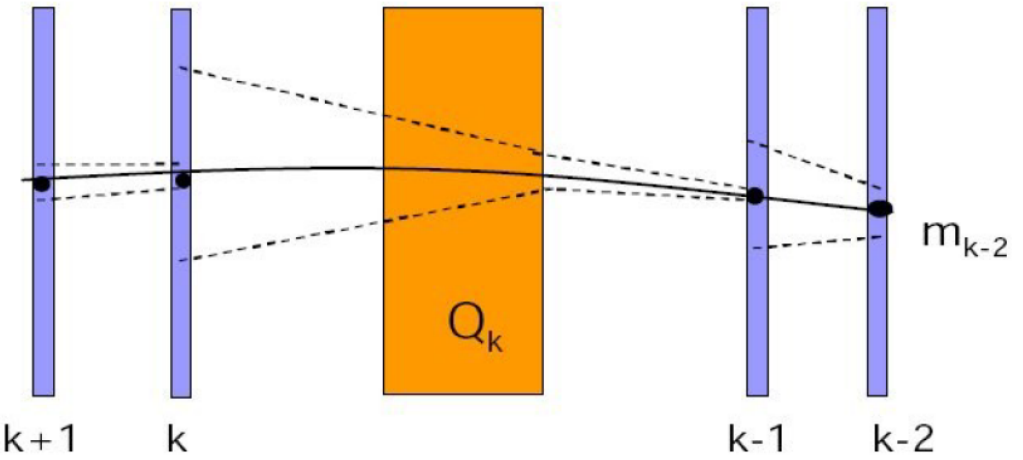




Local Method: Track Following

Follow track through layers

- Define initial track segment (track seed)
- Use track model to extrapolate to next detector layer



- Define search window around expected position, depends on resolution and amount of material
- If hit found, add it to the track candidate and iterate to next layer
  - *Detector inefficiency: if no hit found go to next layer, allow several missing layers*
  - *Allow for combinations, if more than one hit in window follow all paths concurrently, decide later what is correct*

Kalman-filter, will be discussed later

Global Method: Histogramming

For circular tracks (homogeneous B-field (usually solenoid))

Conformal mapping (transformation: circles to lines)

$$u = x/(x^2 + y^2), \quad v = -y/(x^2 + y^2)$$

Compute angle Φ $\Phi = \arctan(v/u)$

Make a histogram of Φ

Find peaks in Φ histogram

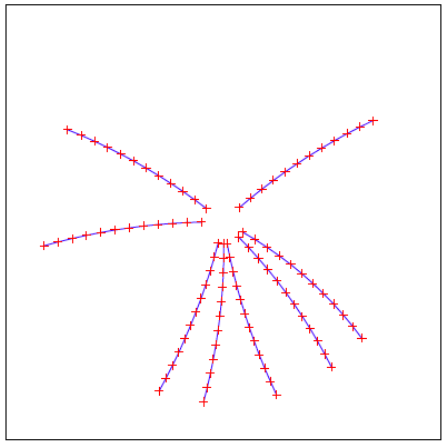
Transforms a 2D problem to 1D

If tracks don't come from (x,y)=(0,0)

- Line is offset, and washed out peak in 1D histogram
- Solution: $x'=x-x_0$ and $y'=y-y_0$ where (x_0,y_0) is e.g. the innermost point on the track

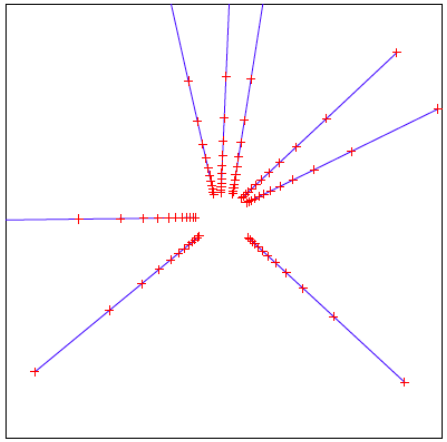
Histogram Method

Coordinate space



Original tracks and hits

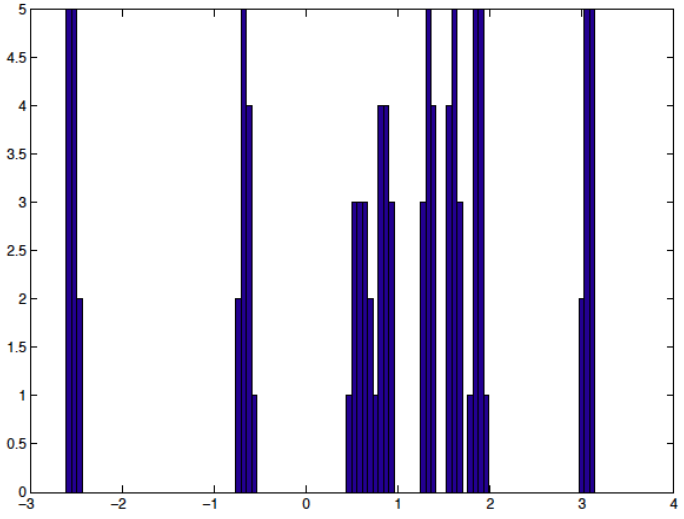
Transformed space



Transformed tracks and hits

As you see, the histogram binning is not arbitrary, it is related to the resolution.

Too fine: then the hits don't make a peak
Too coarse: then the peaks overlap, and the resolution gets worse.



Histogram of the polar angles of all hits

Hough Transformation

- Histogramming technique based on coordinate transformation
- Transform point (image space) to parameter space
- Hit in image space becomes a trajectory in parameter space
- Hits from the same track → trajectories cross in param. space
- Binning of parameter space (histogram)
  (bin size: tradeoff between resolution, efficiency and speed)
- Find peaks in histogram → track candidate

- Very fast for simple track models (straight line, circle)
- Well suited for parallel processing → fast online trigger

OFFLINE PANDA EXAMPLES

Global pattern recognition

`PndSttMvdTrackFinder / PndSttMvdGemTrackFinder`

- default global PR
- Contains: MVD + STT + GEM

- Starts from MVD and STT stand alone track finders
- Extrapolates STT to MVD and vice versa
- New track candidate of MVD and STT is refitted
- Refitted track extrapolated to GEM stations
- Only primary tracks

- No refit with GEM stations because magnetic field in GEMs not homogeneous enough for helix assumption

Global pattern recognition 2

`PndBarrelTrackFinder`

- Alternative global PR

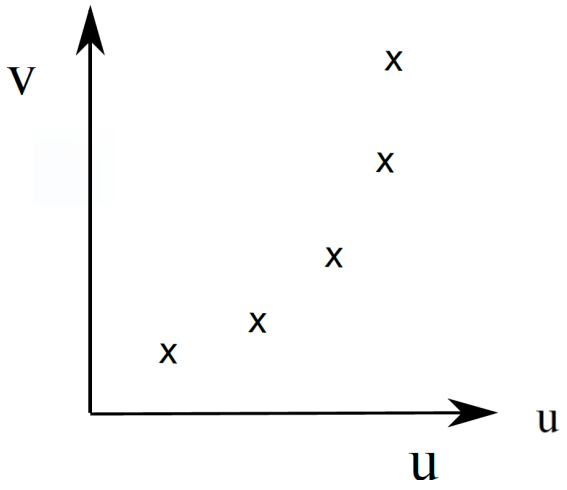
- Does not start at any specific detector
- Treats MVD, STT and GEM hits equal
- Looks for correlation in xy plane
- Tries to gather z information when possible
- Hit mixing to avoid unphysical correlations
- Only primary tracks

ONLINE PANDA EXAMPLES

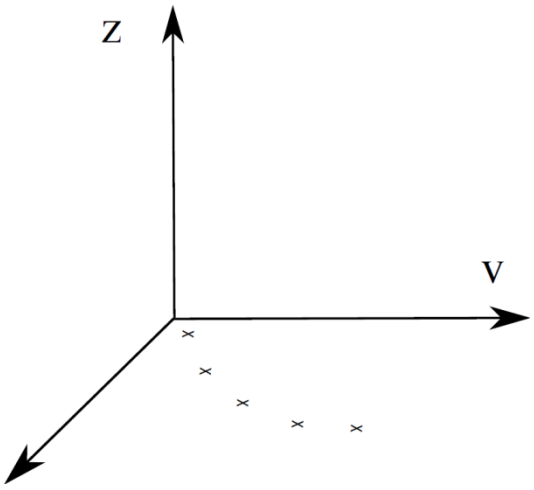
Riemann Track Finder

- The Riemann Track Finder is based on a fast circle fit using a Riemann surface:
  - The 2D points in xy-plane are projected on a paraboloid ($w = x^2 + y^2$)
  - A fast fit of a plane through the new 3D points give you the plane parameters which can be bijectively mapped to the radius and the origin of the circle
  - In a second fit one plots the arc length of the hits on the circle versus their z-position which should be a straight line
- The method is very fast if you neglect energy loss and multiple scattering
- Both can be implemented which leads to very good accuracy but then it is slower than a standard Kalman fit
- Riemann fit was implemented by Sebastian in pandaRoot but without error treatment

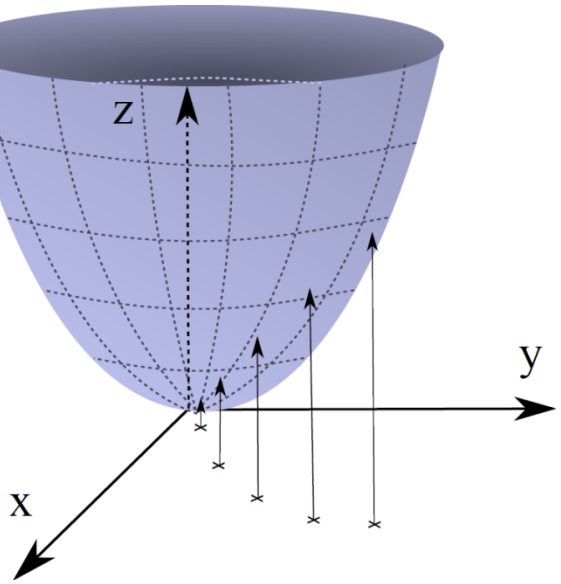
Riemann Track Finder



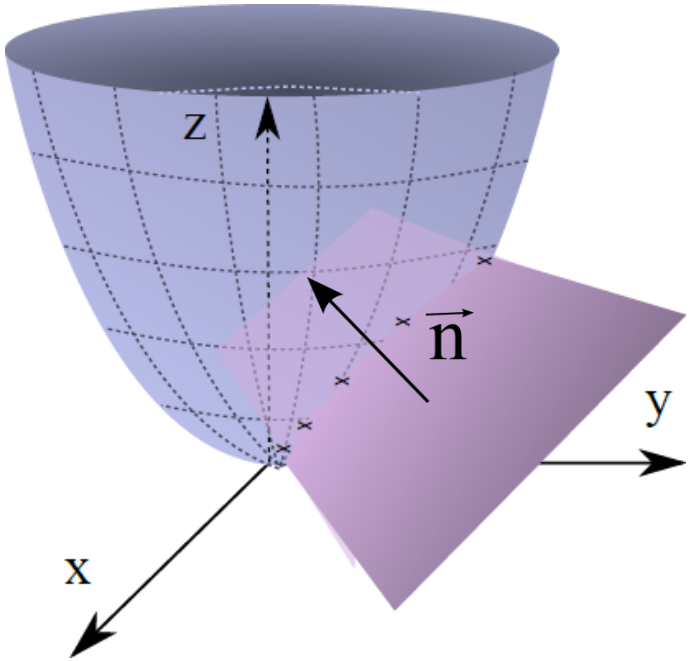
Points to be fitted



Add z-dimension



Map onto paraboloid



Calculation of plane through 3D points

simple eigenvalue determination

Riemann Track Finder

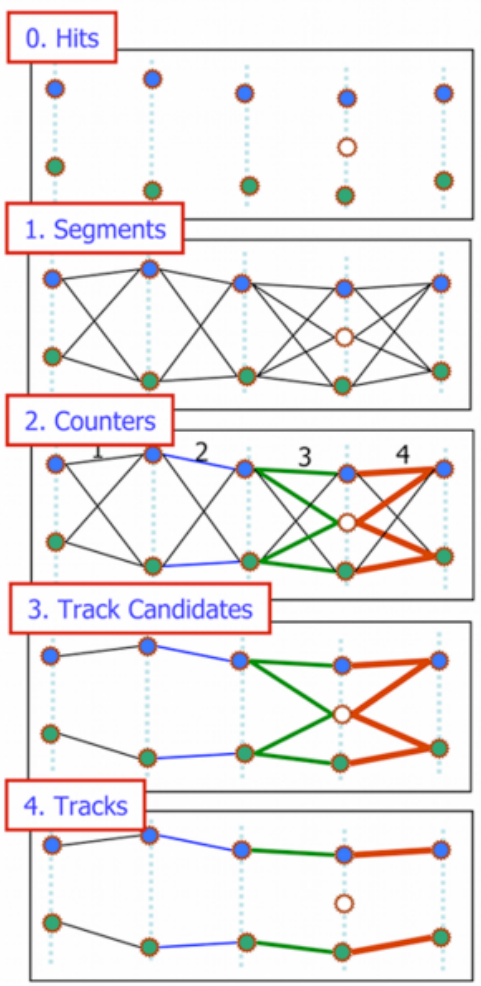
1. Find all (reasonable) combinations of three points
2. Calculate Riemann plane
3. Calculate distance of all hits to plane
4. If below threshold add them to plane
5. Recalculate plane at the end (or in between?)
6. Take only those planes (circles) with more than 3 hits

7. Linear fit or arc length vs. z-coordinates of hits for z-Dimension

CA TRACK FINDER

Cellular Automaton as Track Finder

1. Build short track segments
2. Connect according to track model, estimate a possible position on a track
3. Tree structures appear, collect segments into track candidates
4. Select best track candidate



CELL TRACK FINDER

CellTrackFinder

JÜLICH FORSCHUNGSZENTRUM

1. Generation of tracklets in STT

- Primary tracklets: use of cellular automaton
- Combination of tracklets: use of Riemann Fit
- Faulty: Fit to center of the tubes
- Add remaining hits to best fitting Track

→ TrackletGenerator

2. Correction of STT-Hits based on event time

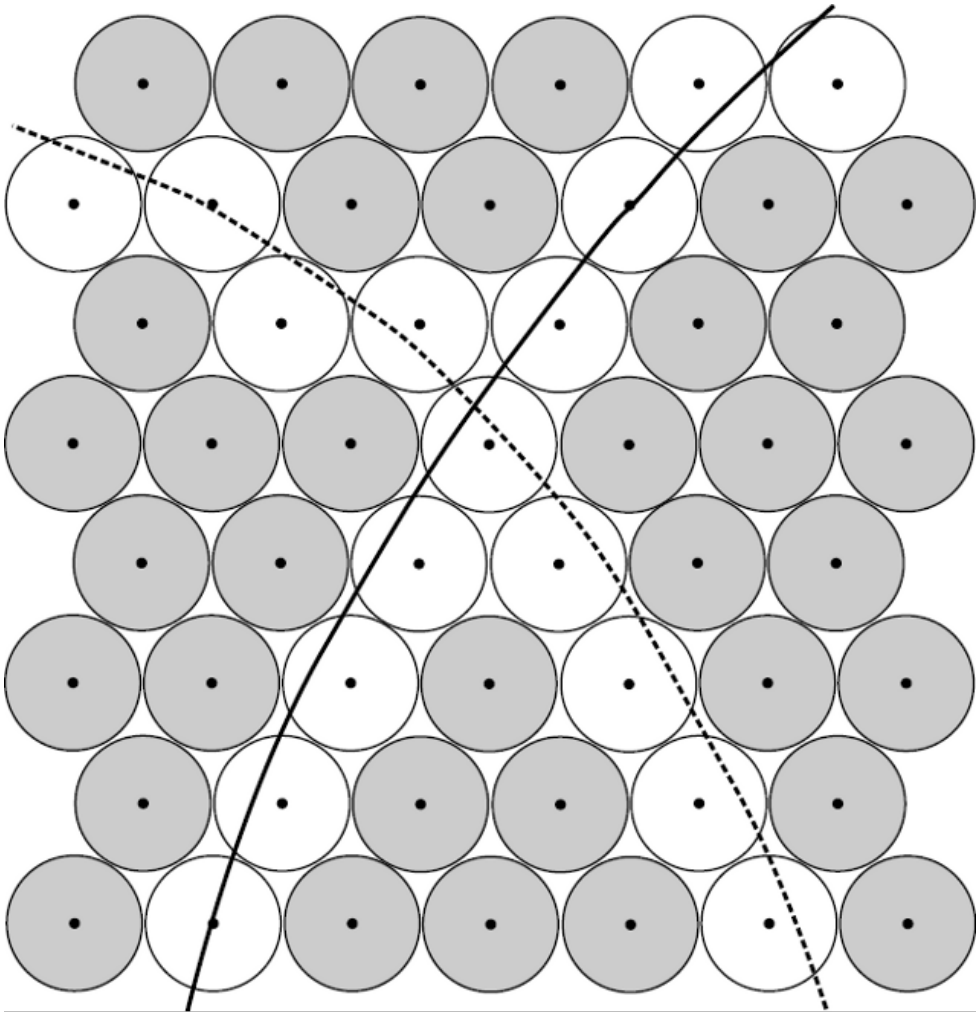
- Correct points for Riemann Fit by calculating tangents to the isochrones

→ HitCorrector

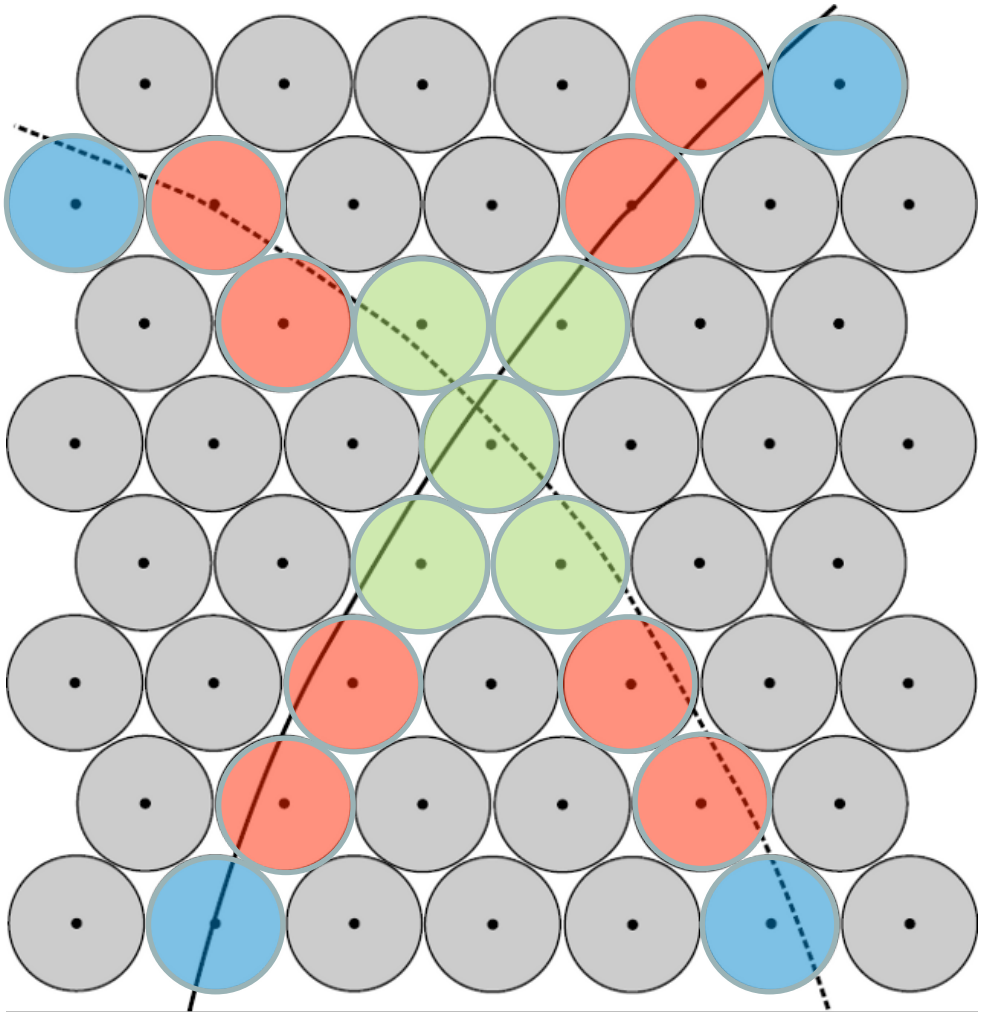
1. TrackletGenerator
Cellular Automaton

- Abstract discrete model for dynamic systems
- Definition by:
  - Cell
  - Neighborhood relations
  - Discrete number of states
  - Transition rules for states
  - Simultaneous changes

1. TrackletGenerator
Cellular Automaton - Apply to STT



1. TrackletGenerator
Cellular Automaton – Apply to STT

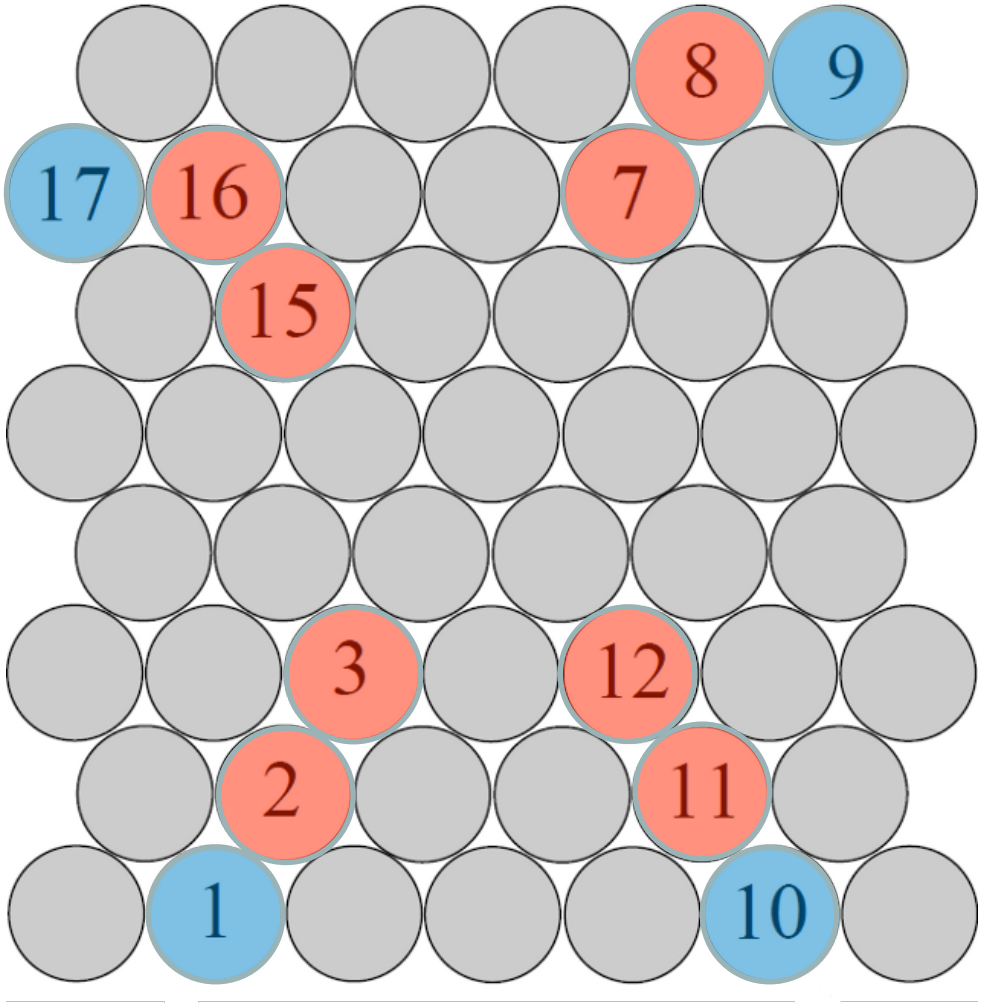


Number of Neighbours:

1
2
unambiguous

> 2 ambiguous

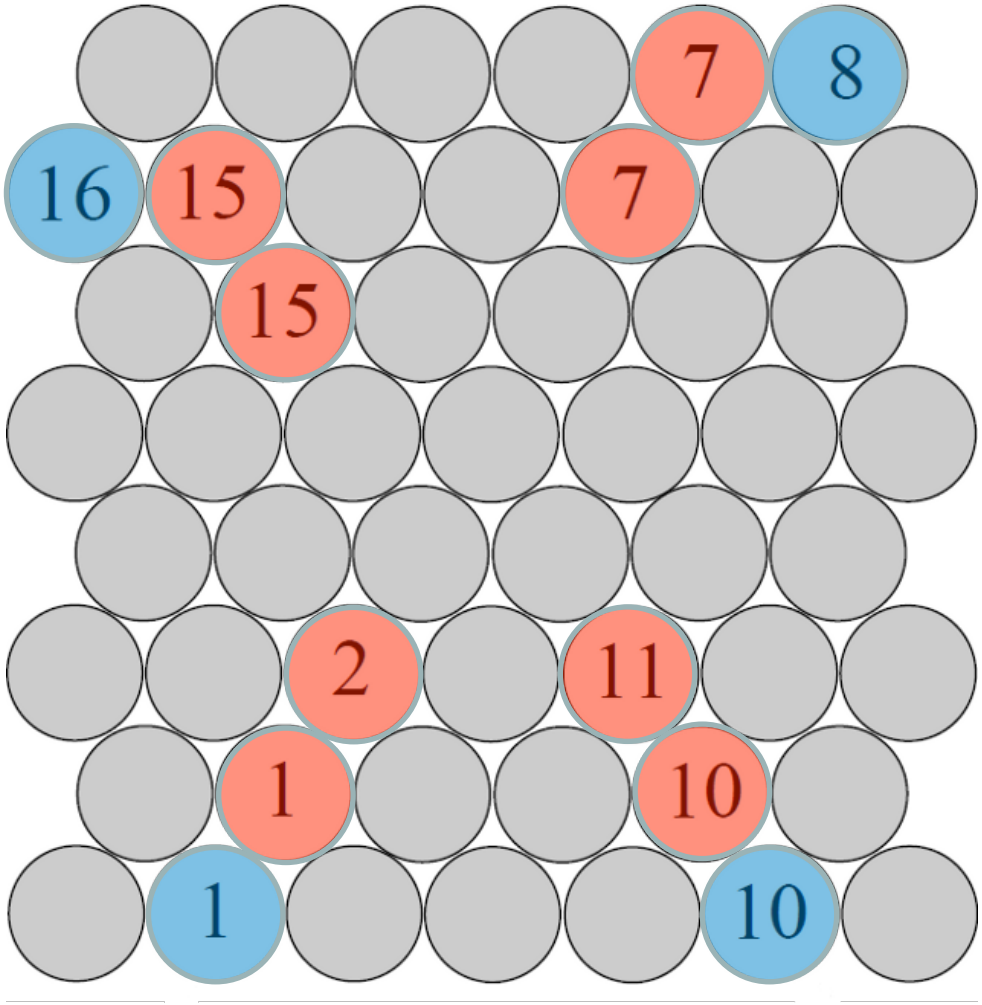
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and one or two neighbours

Rule: Compare your status with the status of your neighbours and take the smallest one

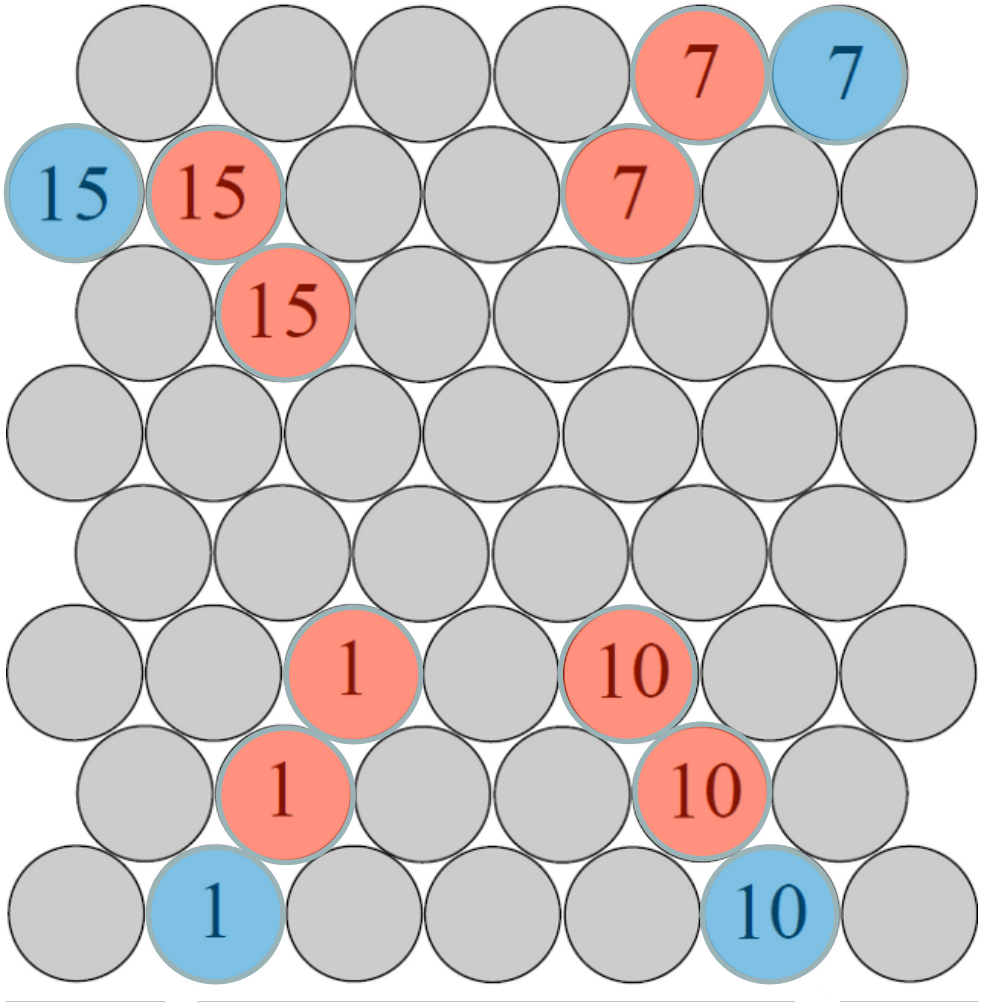
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and one or two neighbours

Rule: Compare your status with the status of your neighbours and take the smallest one

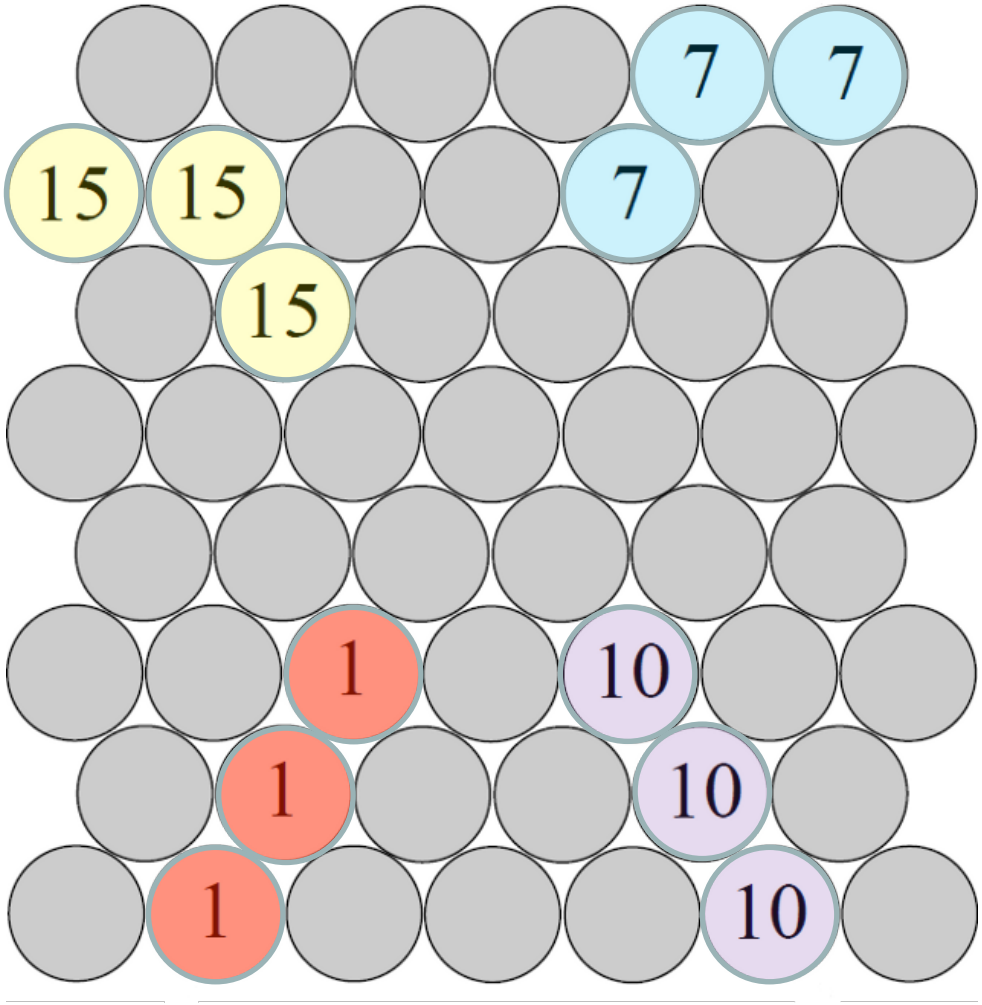
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and one or two neighbours

Rule: Compare your status with the status of your neighbours and take the smallest one

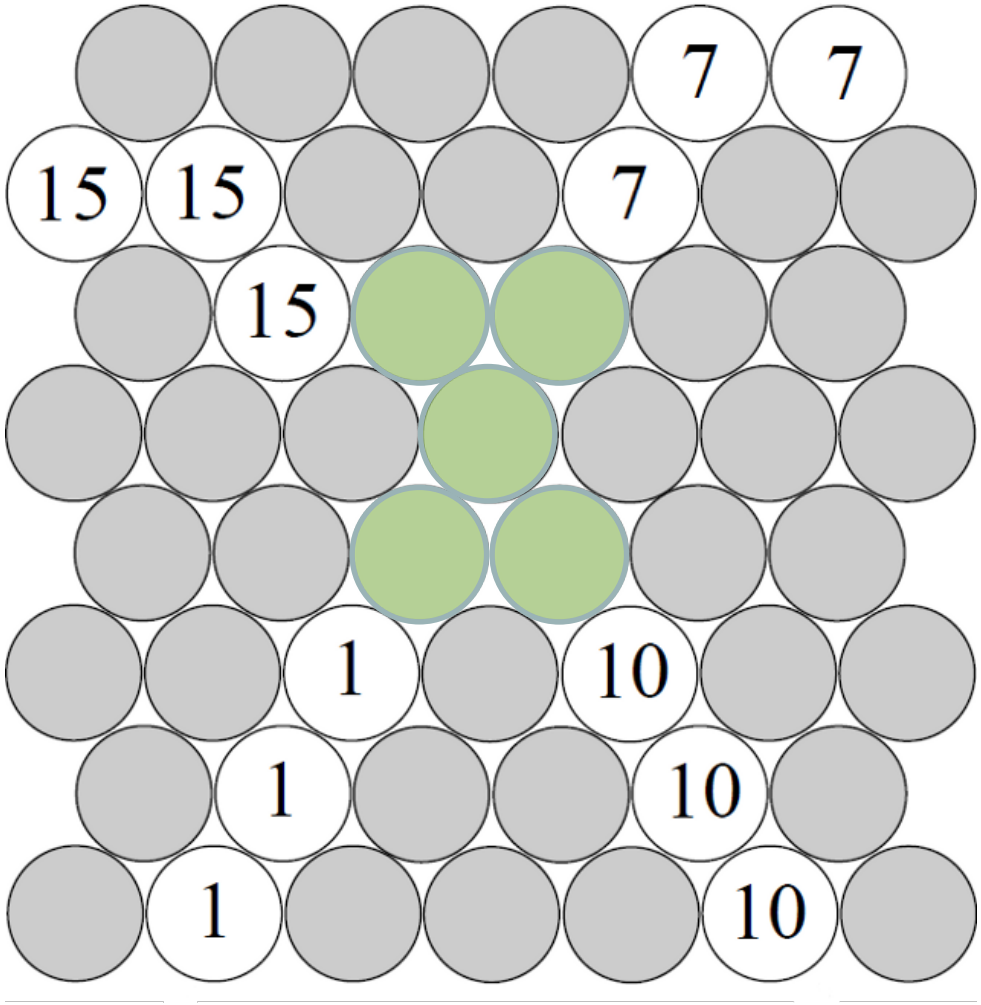
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and one or two neighbours

Rule: Compare your status with the status of your neighbours and take the smallest one

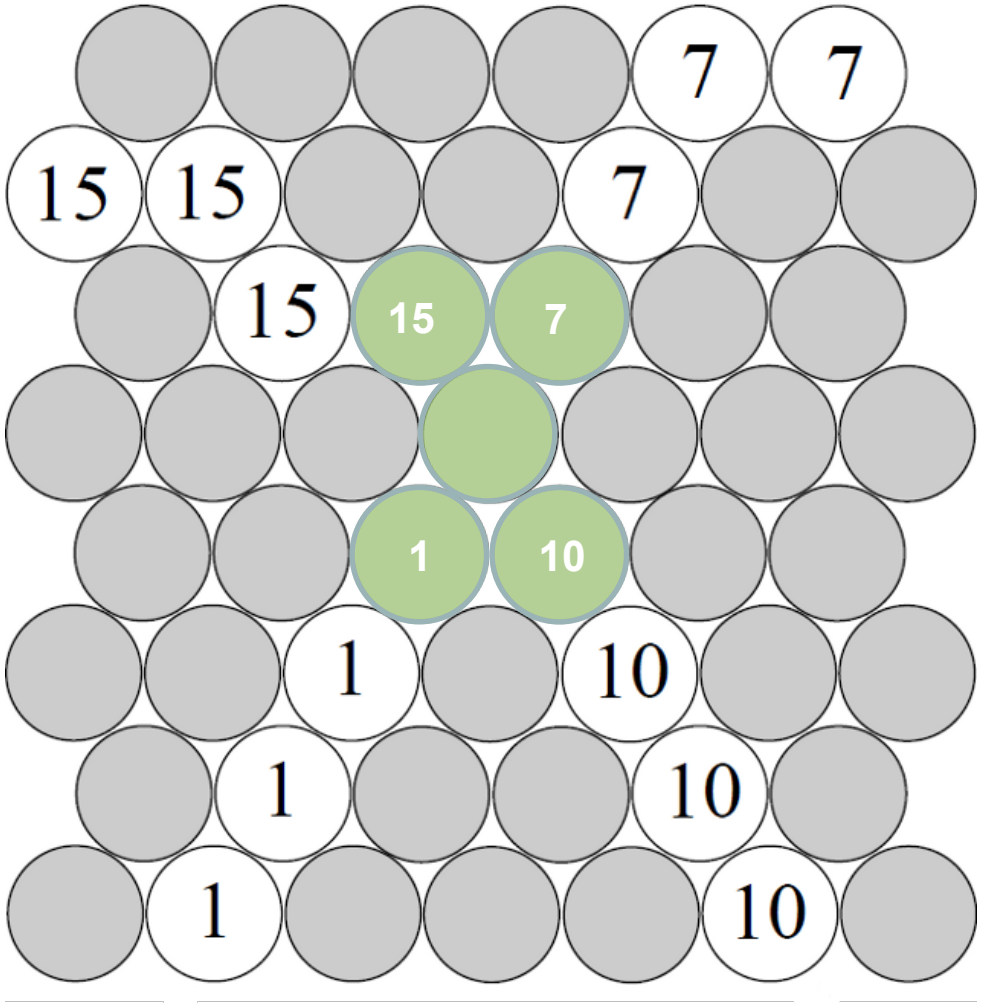
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and more than two neighbours

Rule: Copy the status of all your neighbours into your status

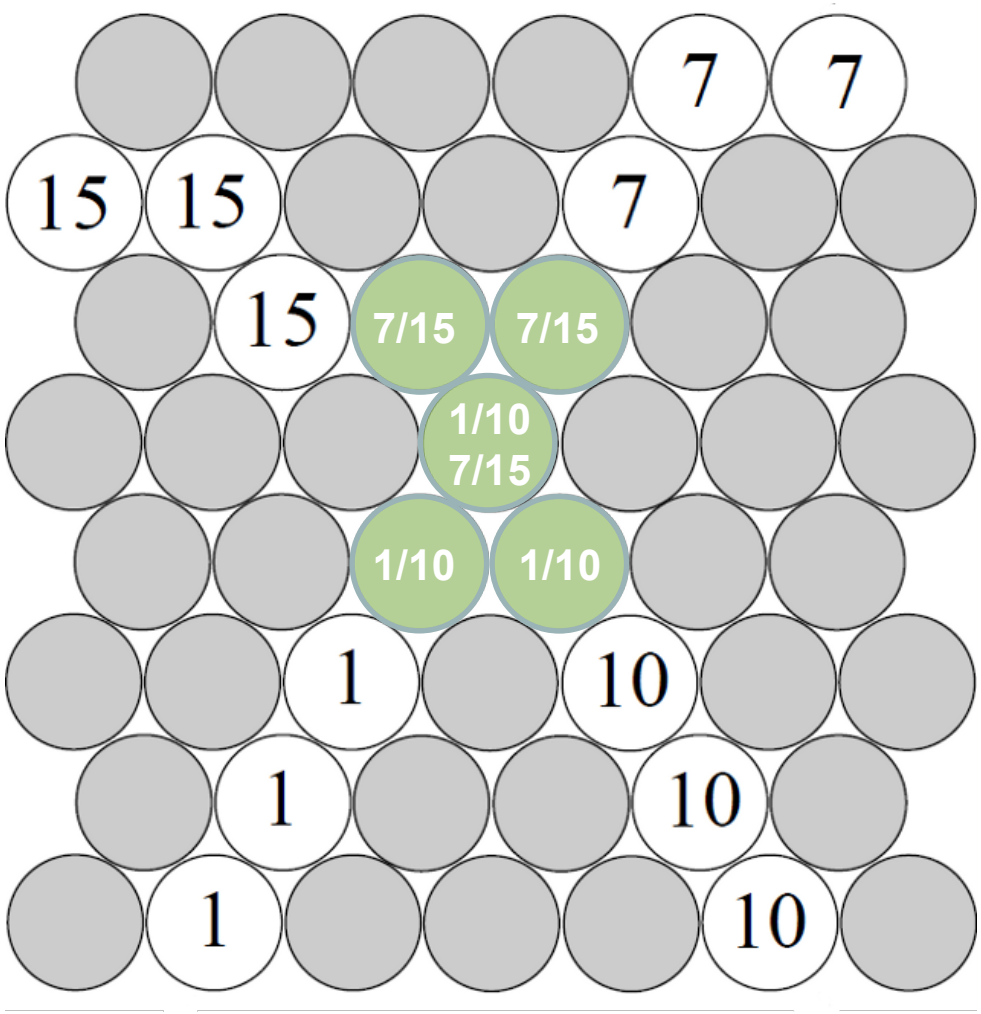
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and more than two neighbours

Rule: Copy the status of all your neighbours into your status

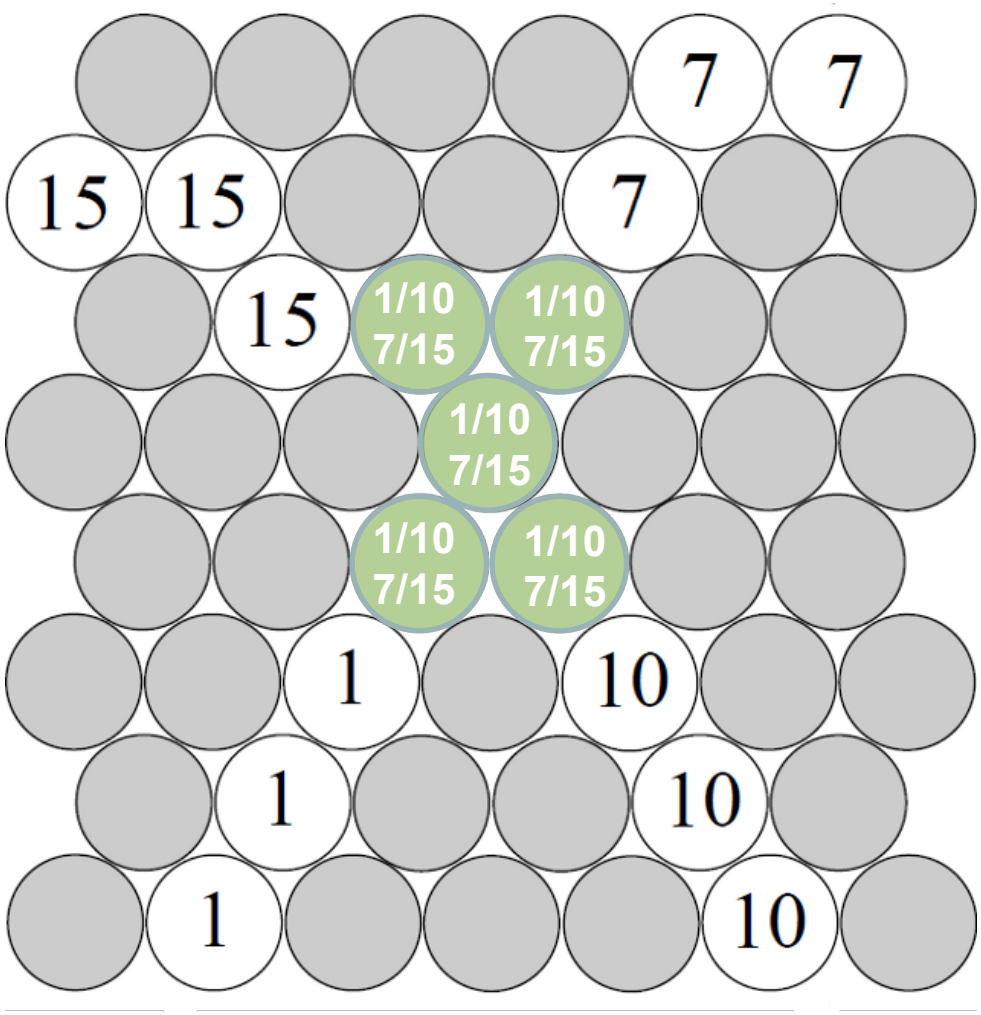
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and more than two neighbours

Rule: Copy the status of all your neighbours into your status

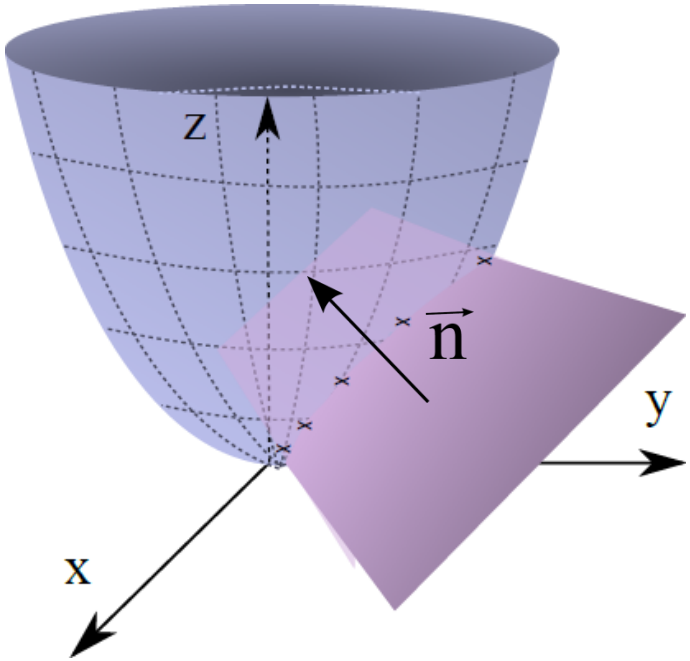
1. TrackletGenerator
Cellular Automaton - Apply to STT



Cell: Straw with hit and more than two neighbours

Rule: Copy the status of all your neighbours into your status

1. TrackletGenerator
Riemann fit

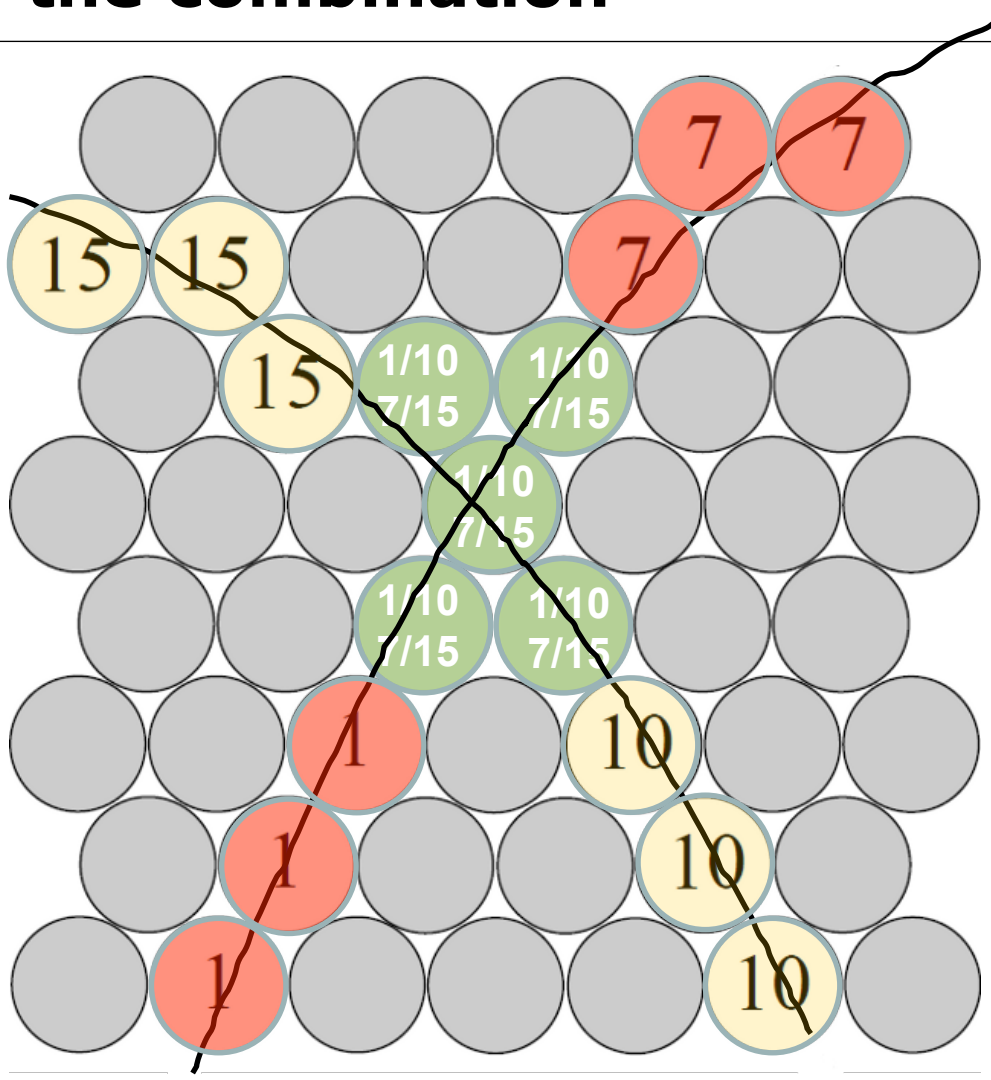


Fit Plane
Normal vector → Circle parameters

Advantages

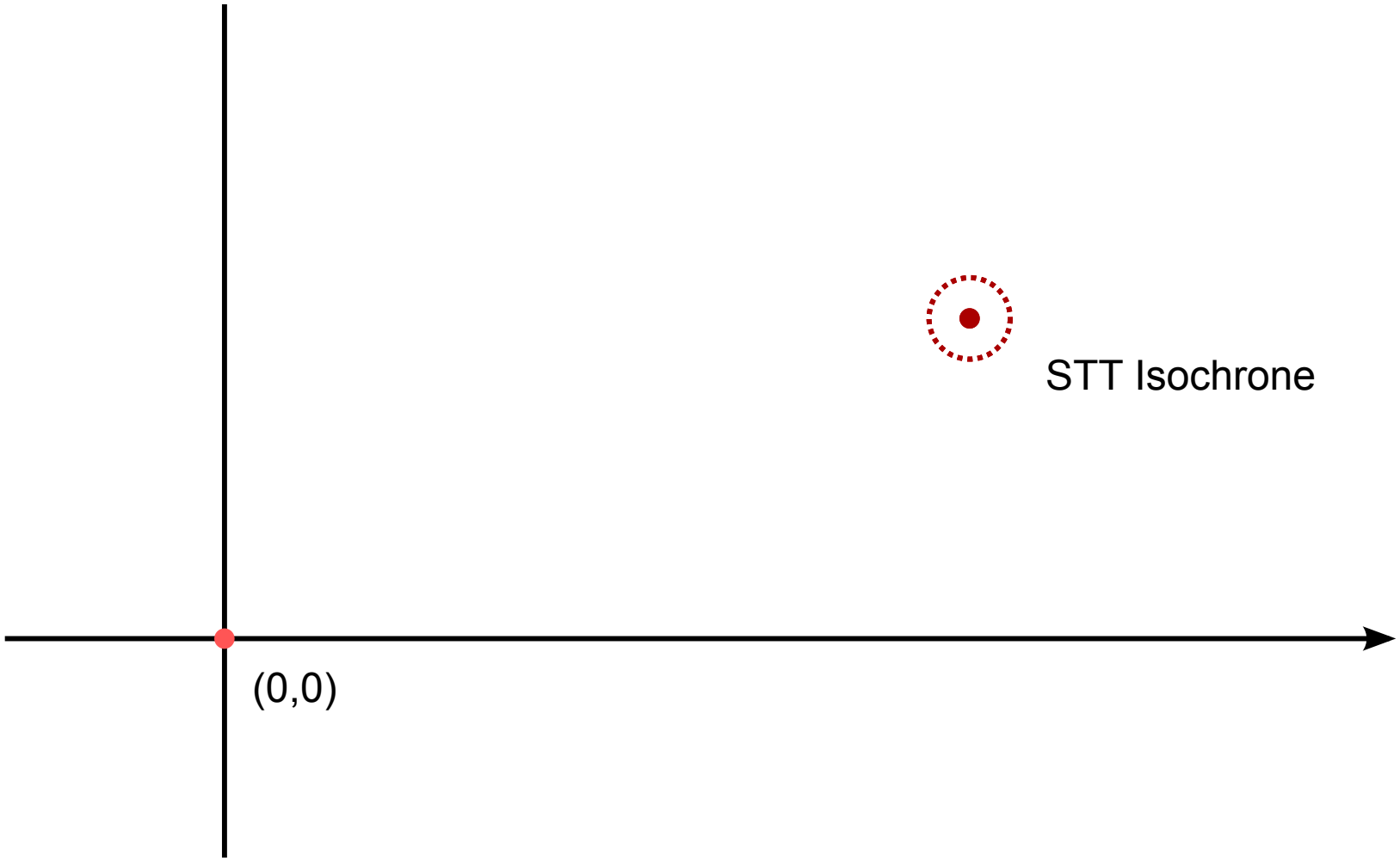
- quadratic → linear
- explicit solution
- already implemented in PandaRoot

1. TrackletGenerator
Result of the combination

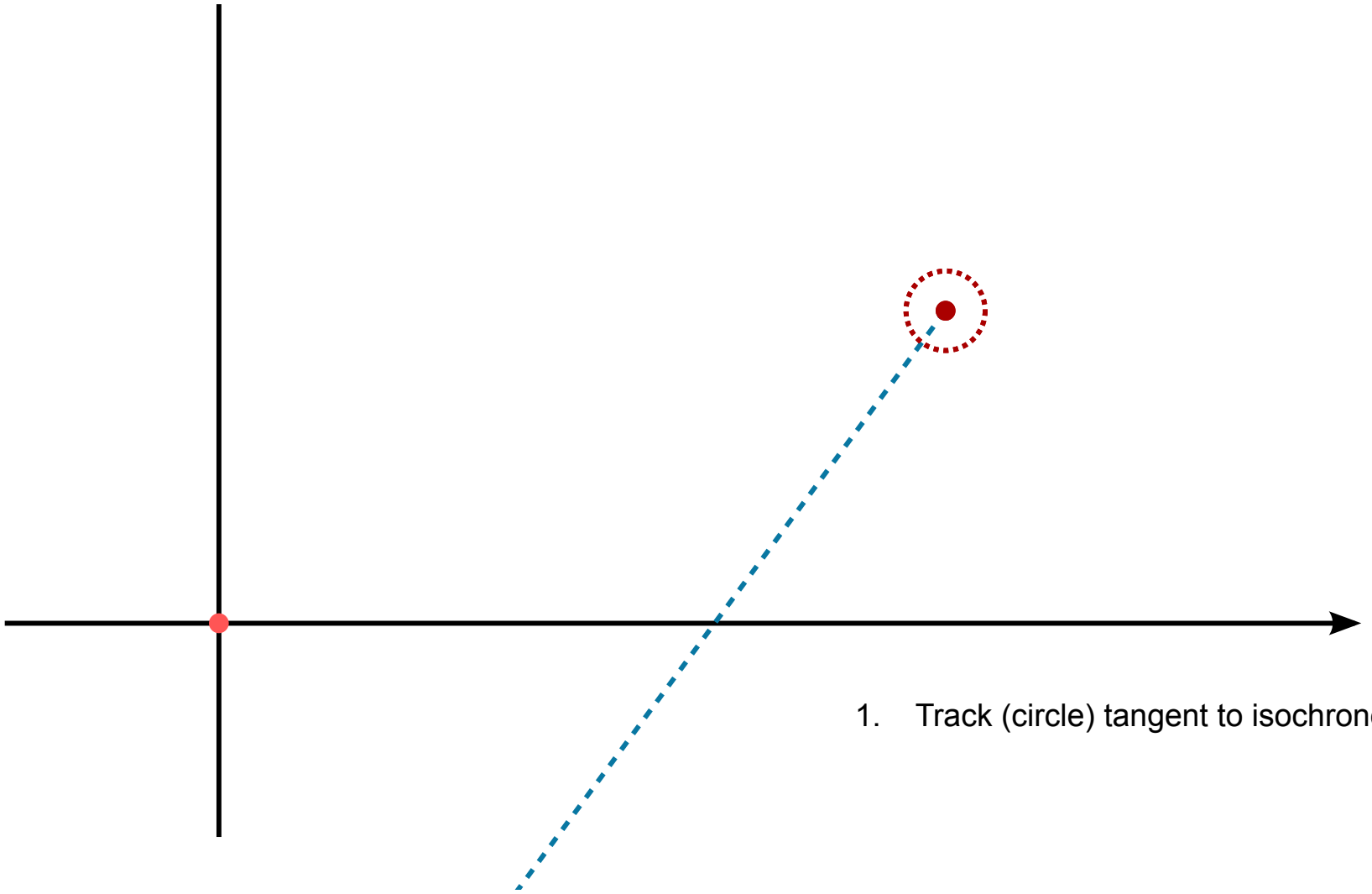


CIRCLE HOUGH

Idea

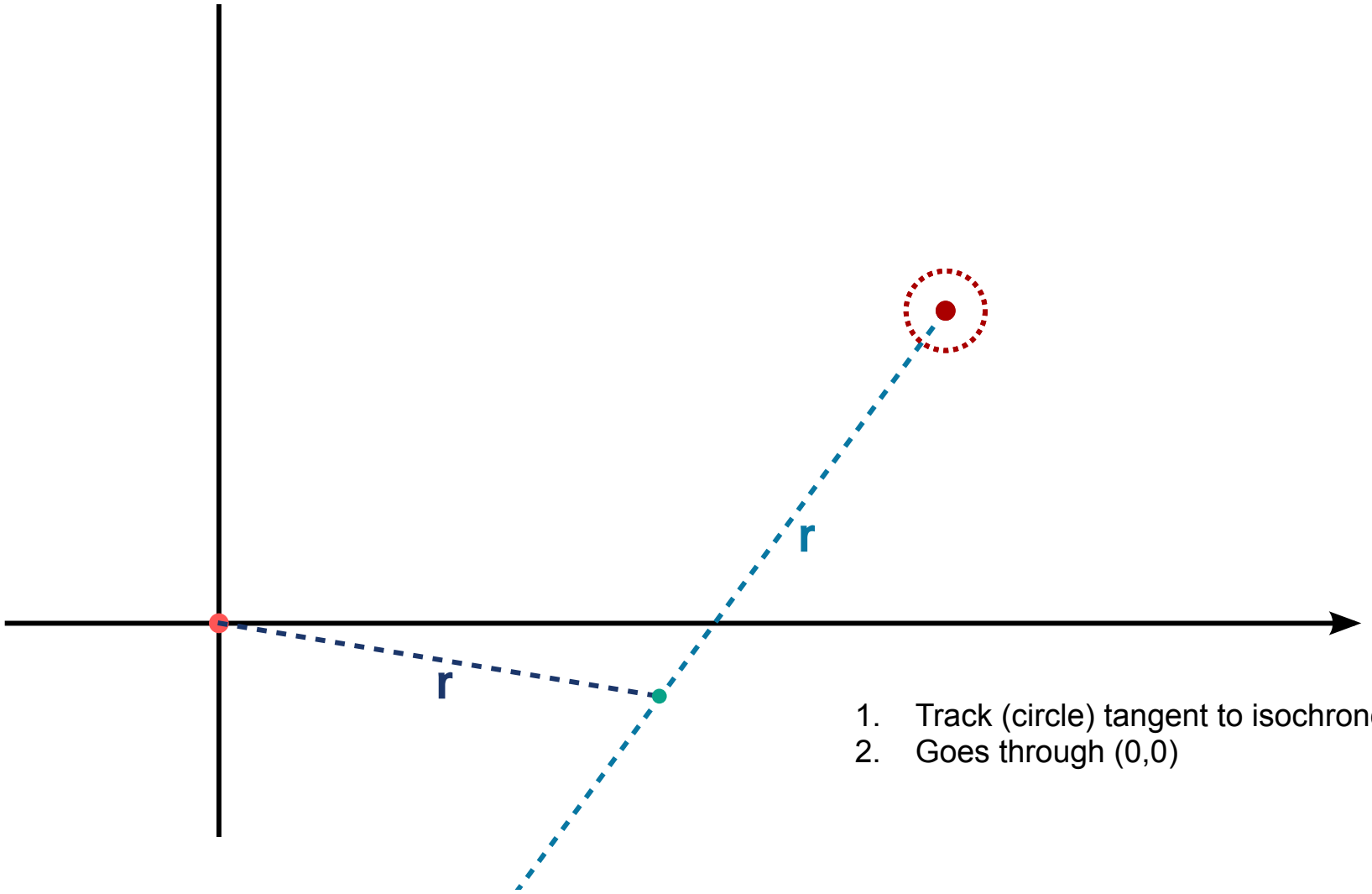


Idea



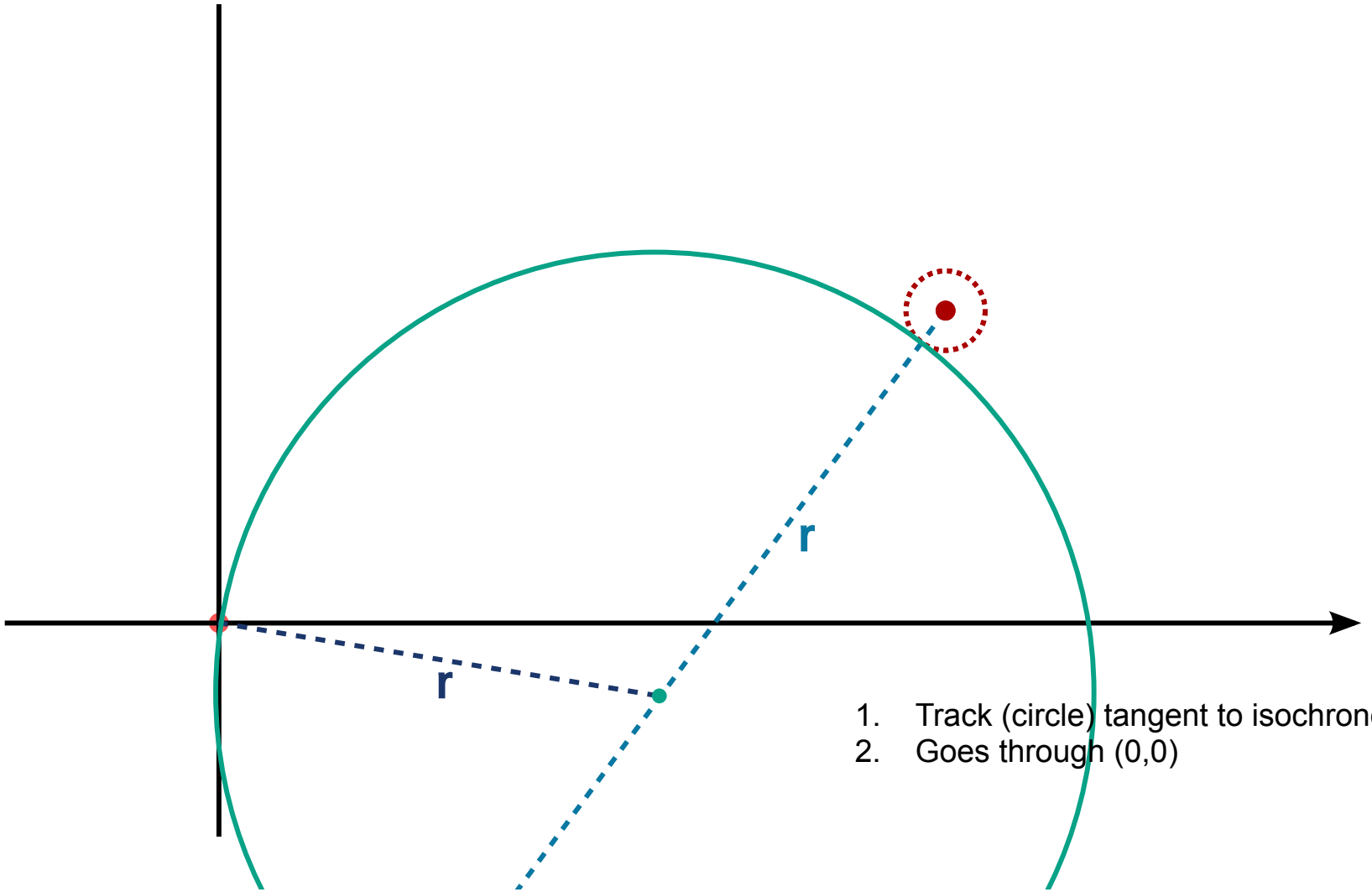
1. Track (circle) tangent to isochrone

Idea



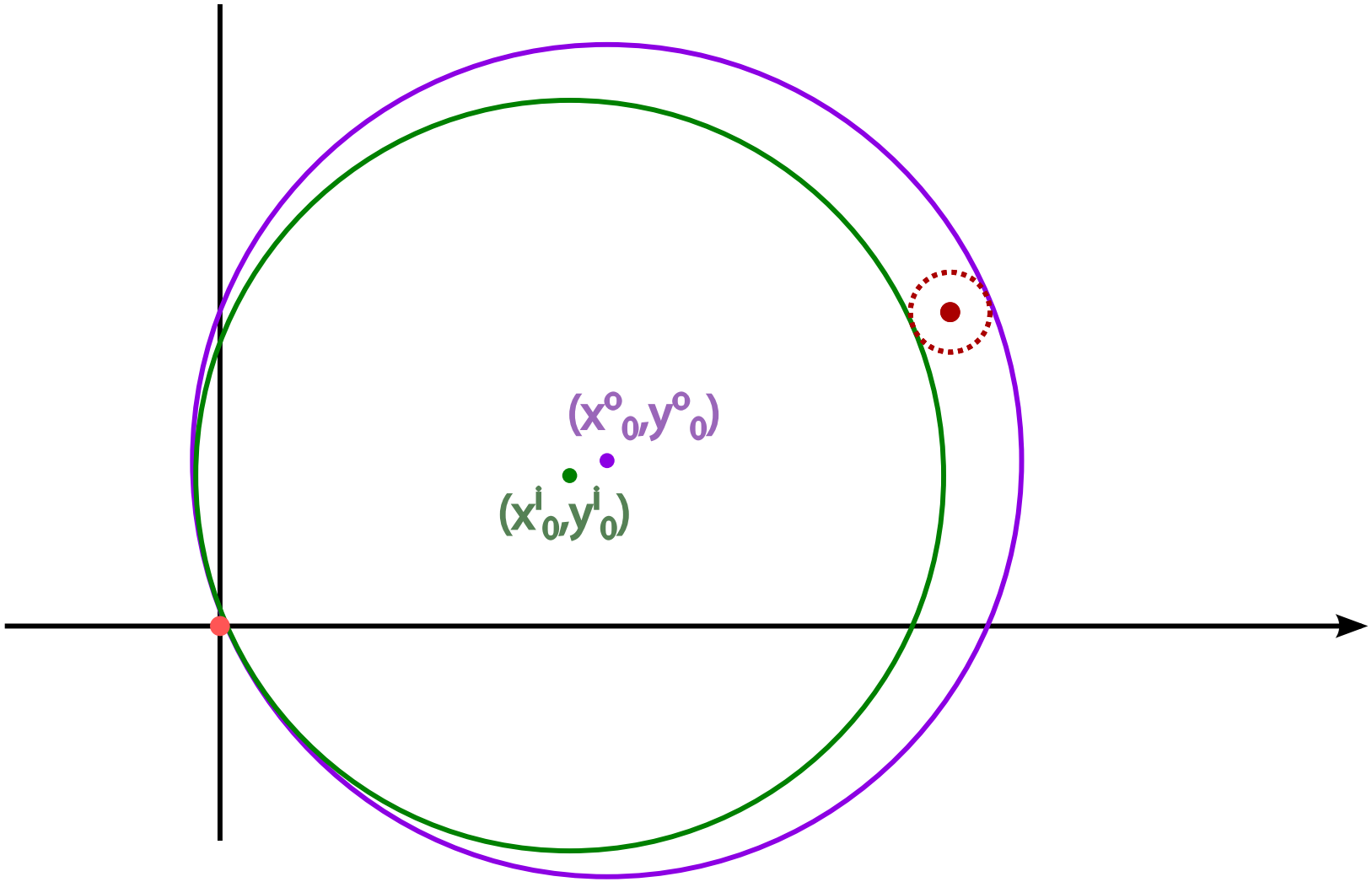
1. Track (circle) tangent to isochrone
2. Goes through (0,0)

Idea

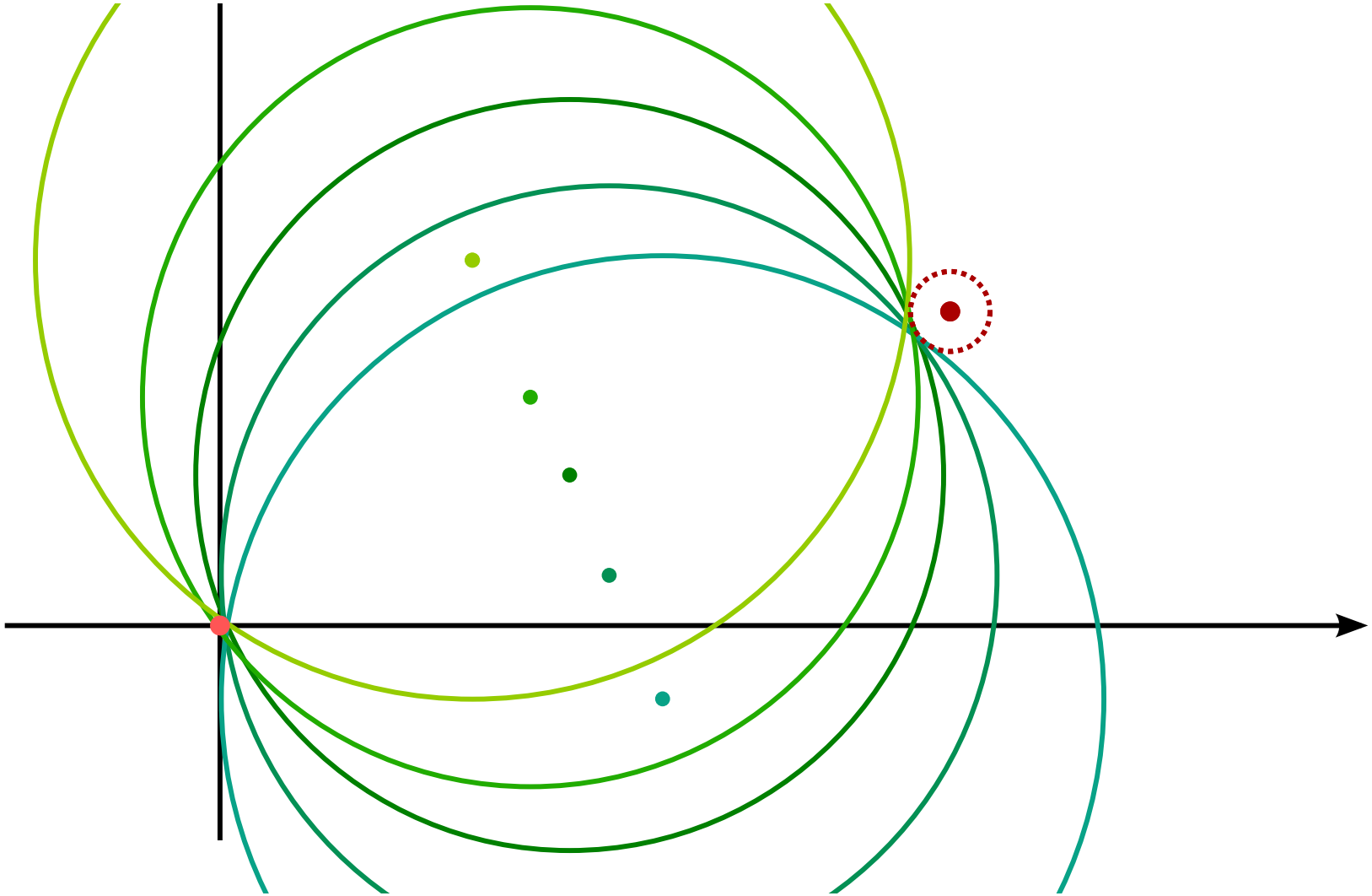


1. Track (circle) tangent to isochrone
2. Goes through (0,0)

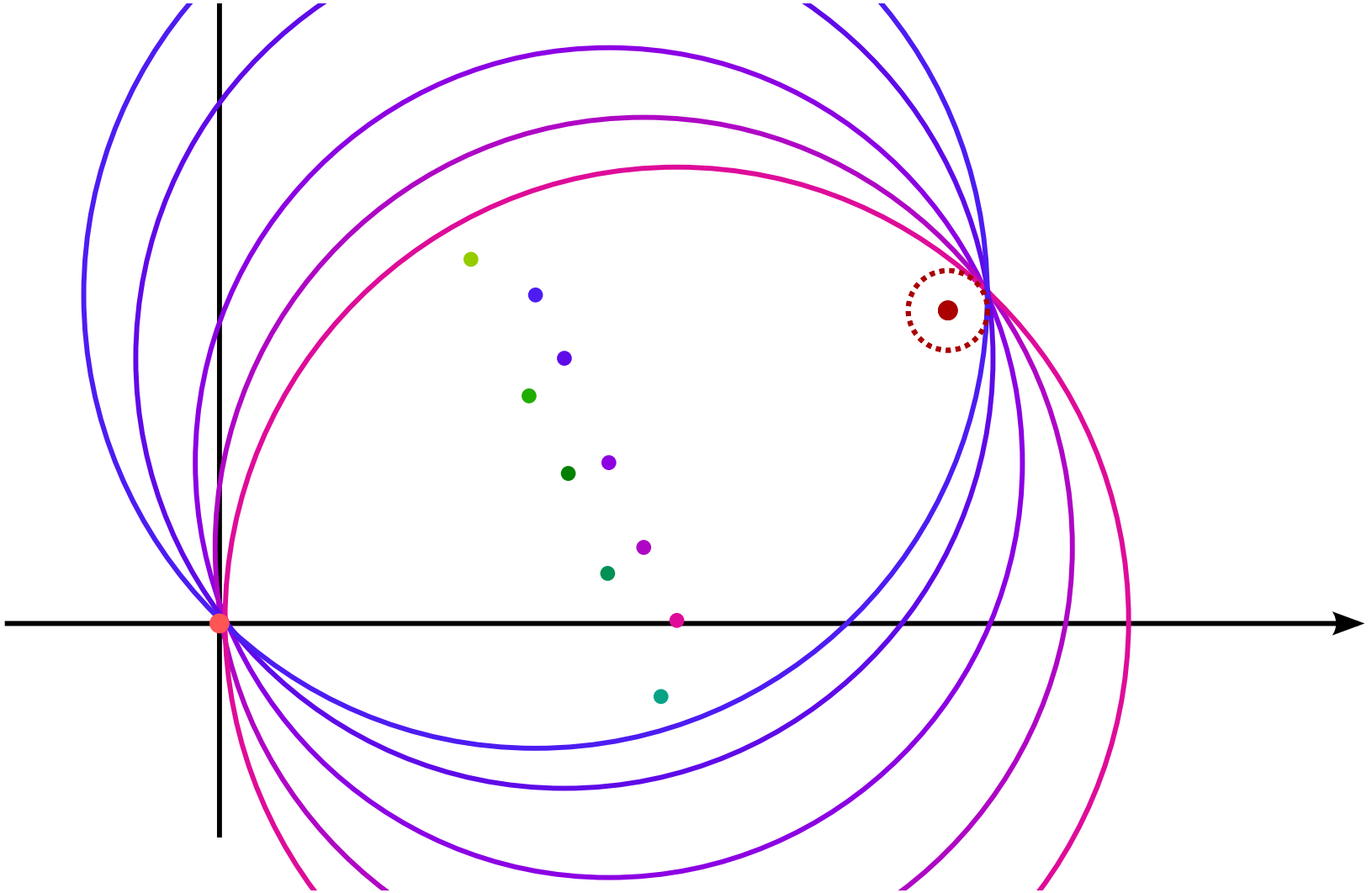
Idea



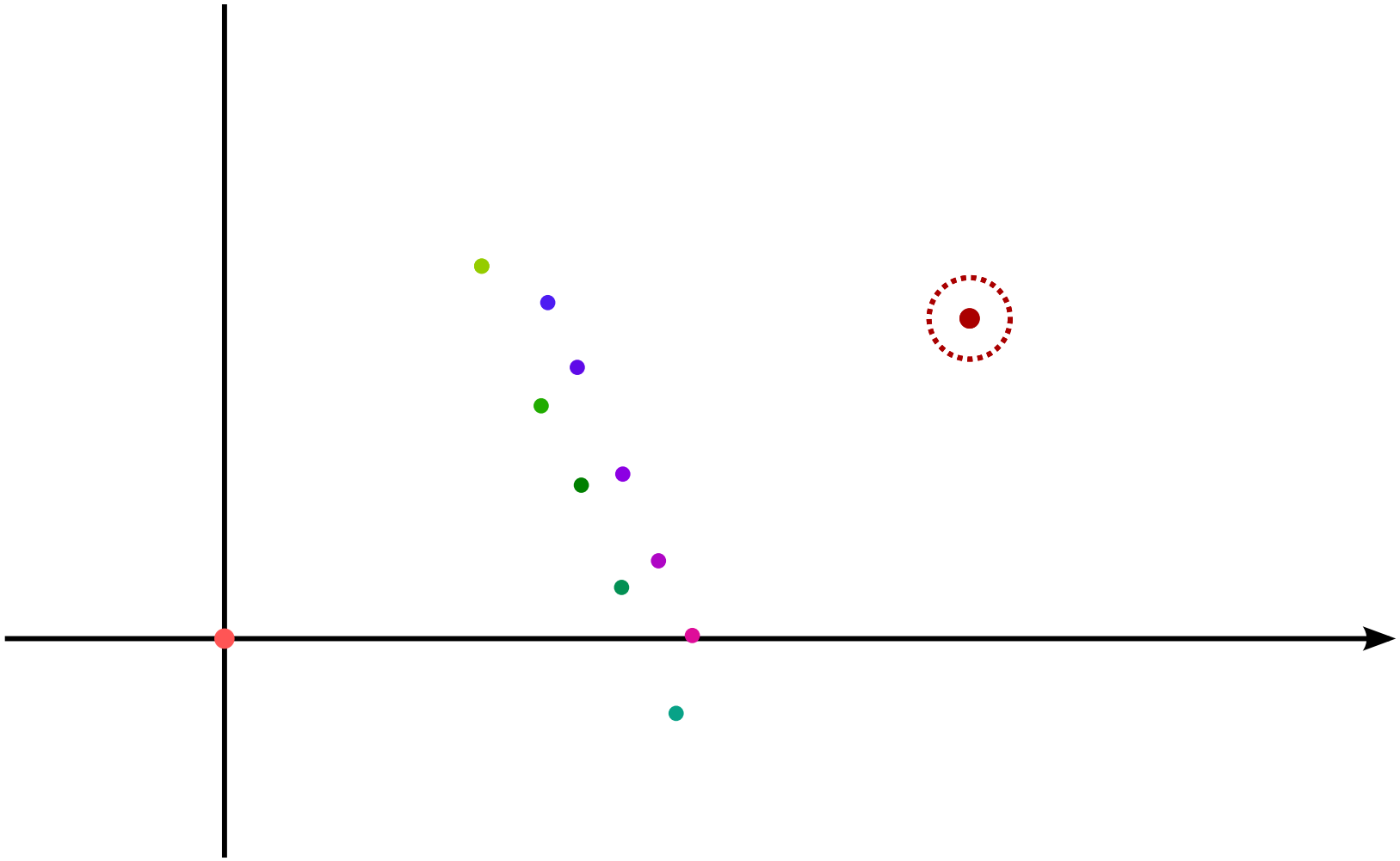
Idea



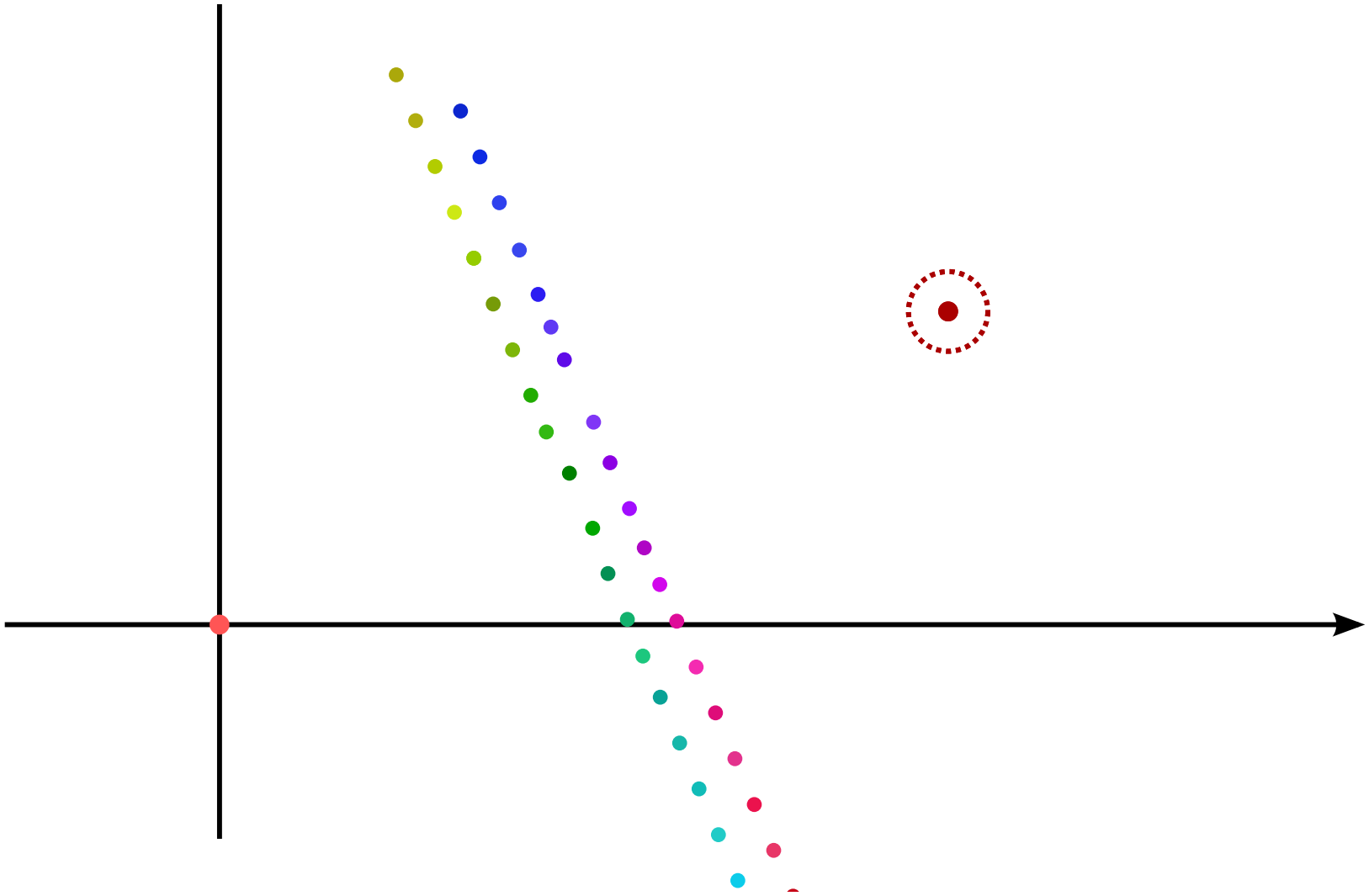
Idea



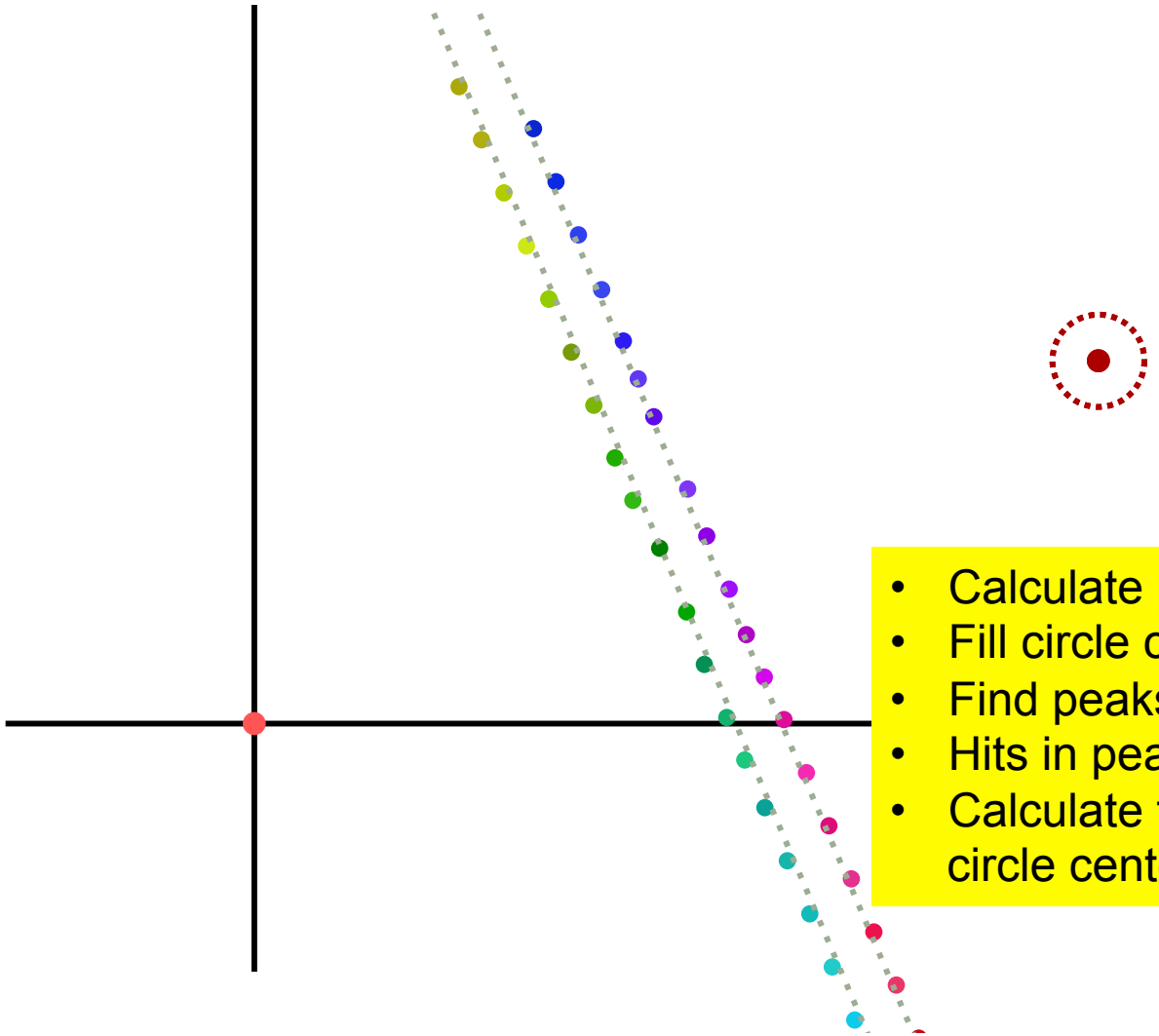
Idea



Idea

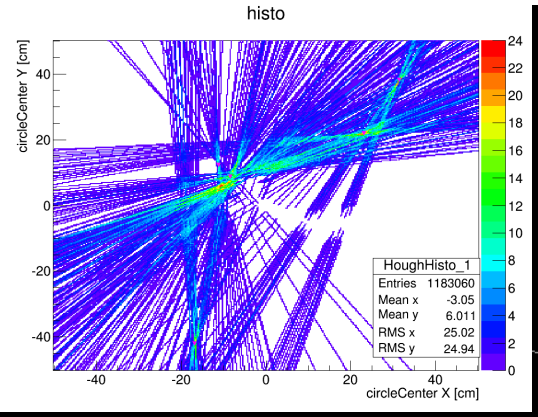


Idea

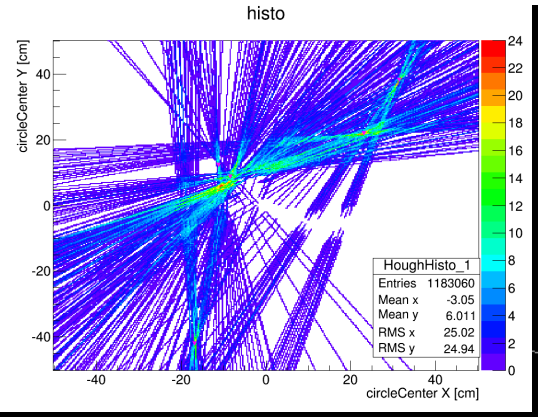


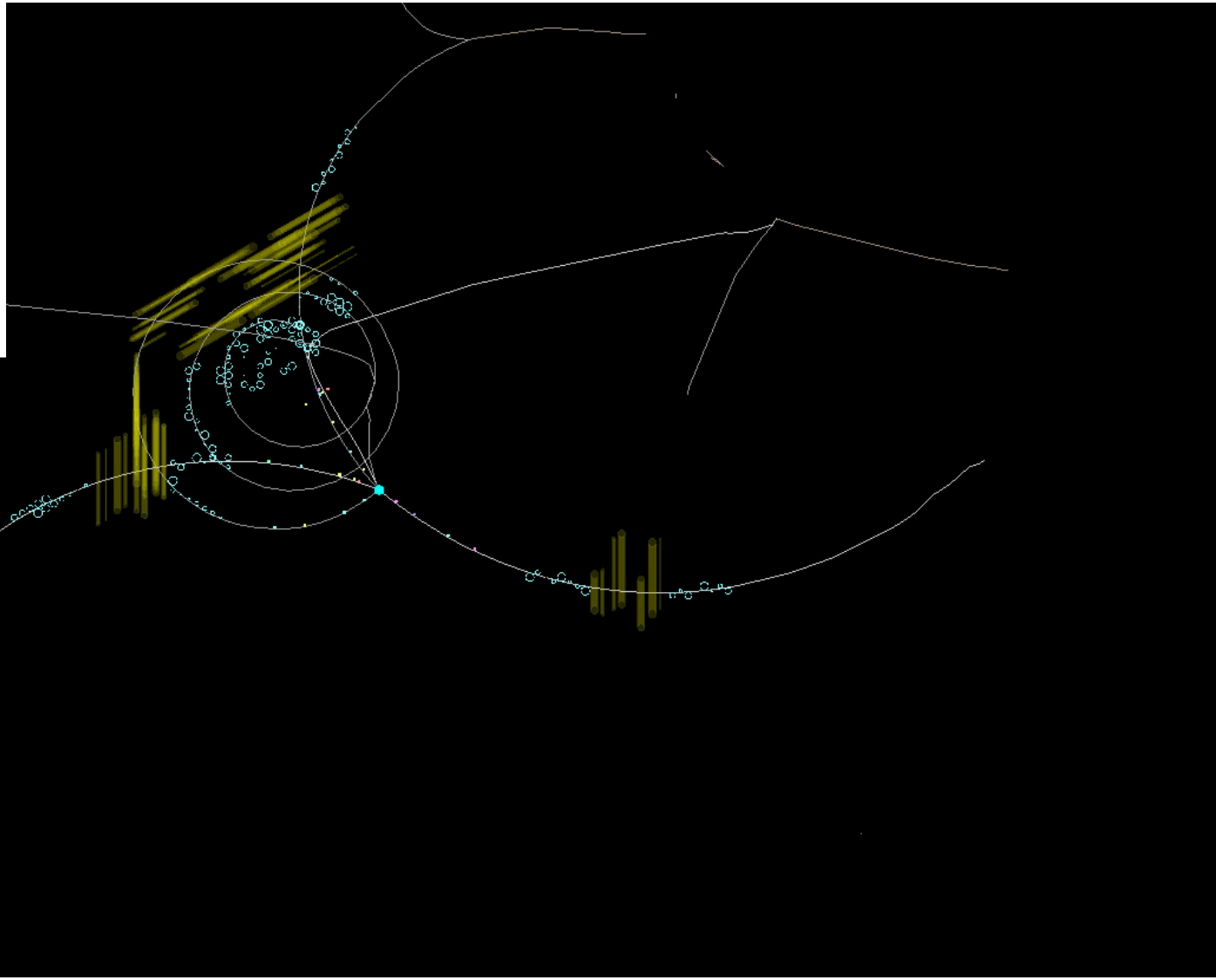
- Calculate possible circles for all hits
- Fill circle centers into xy-histo
- Find peaks
- Hits in peaks are part of a track
- Calculate track parameters from circle centers

DPM Example 2

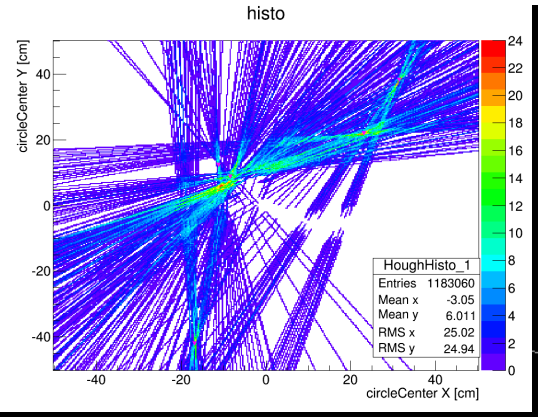


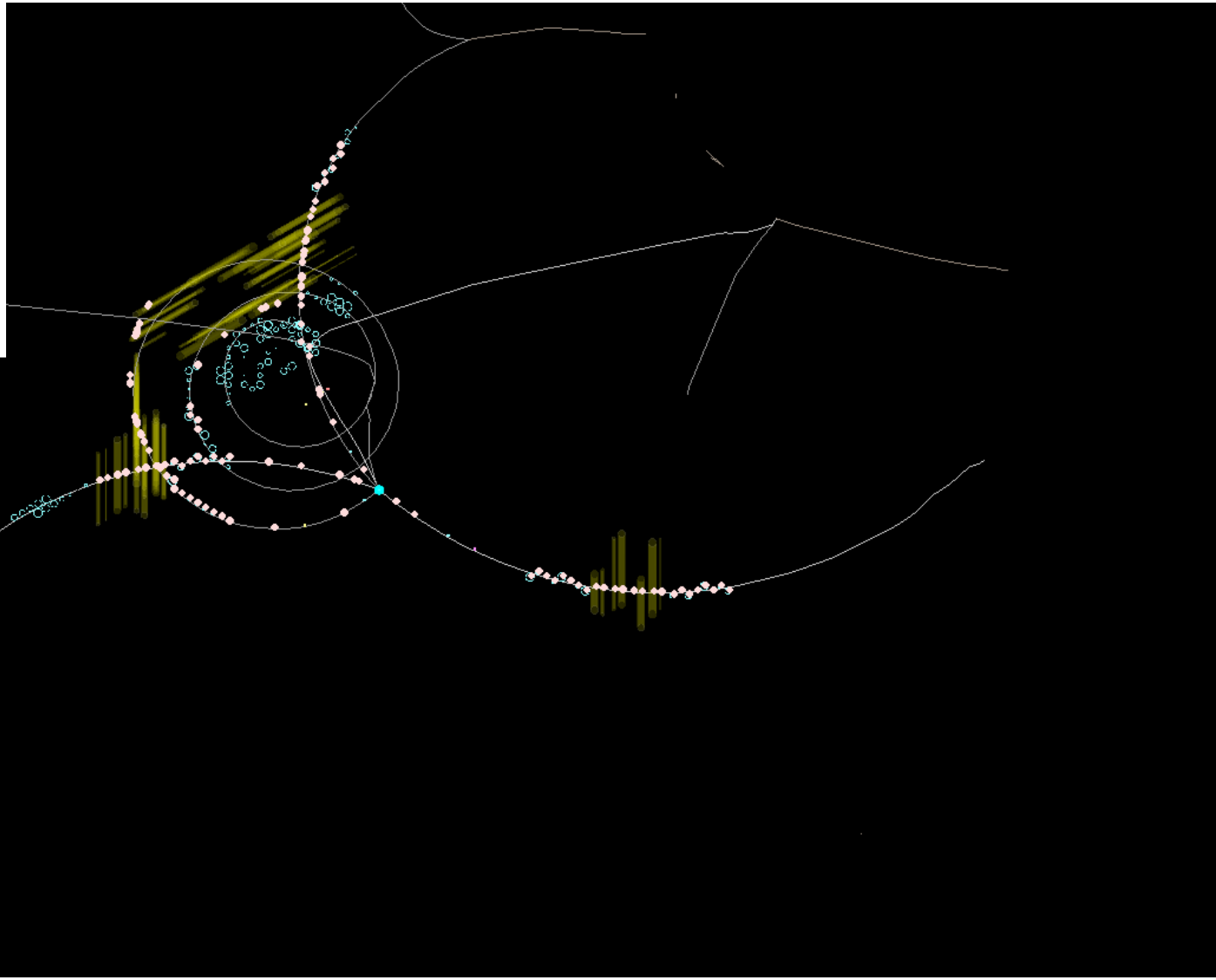
DPM Example 2



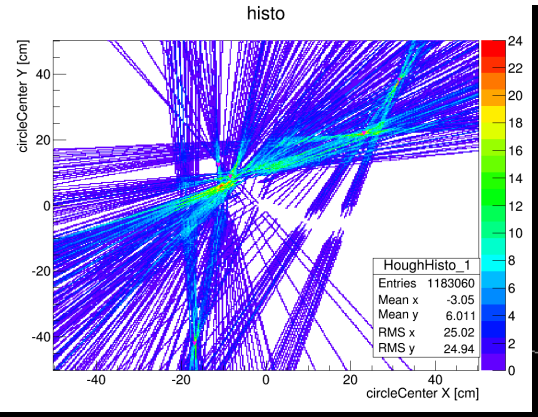


DPM Example 2





DPM Example 2



KALMAN FILTER

Track Parametrization

- In vacuum the trajectory is defined by the initial parameters or state vector.
- 5 parameters are necessary and sufficient
- Each point along the trajectory, describe with a 5-component state-vector (and its 5x5 covariance vector)
- Should be continuous to small changes, approx. Gaussian errors, linear approximation to track propagation
- Best parameterization depends upon B-field and detector geometry: e.g.

detector surface normal to z-axis

surface=beam pipe (collider)

$$\left(x, y, \frac{dx}{dz}, \frac{dy}{dz}, \frac{q}{p}\right)$$

$$\left(r, z, \theta, \phi, \frac{q}{p}\right)$$

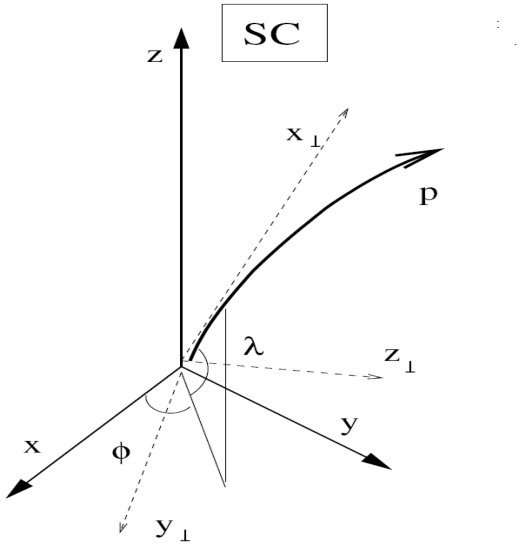
- ✧ DELPHI barrel: x = (Φ, z, θ, β = φ − Φ, 1/R)
- ✧ DELPHI forward: x = (x, y, θ, φ, 1/R)
- ✧ CMS global: position, momentum, charge
- ✧ CMS local: x = (q/p, dx/dz, dy/dz, x, y)

Panda Track Parametrization

`FairTrackPar/FairTrackParP/FairTrackParH`

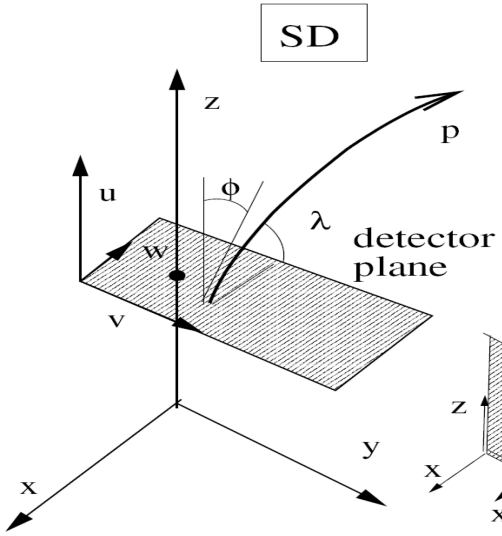
- The physical path of a particle of assigned mass m and momentum p is a six-fold entity of parameters: x, y, z, p_x, p_y, p_z (+ q)
- Track is defined as set of points in detectors, corresponding to the intersection of the physical path of a particle with the detector planes → among six parameters one is fixed by the measurement plane

curvilinear frame



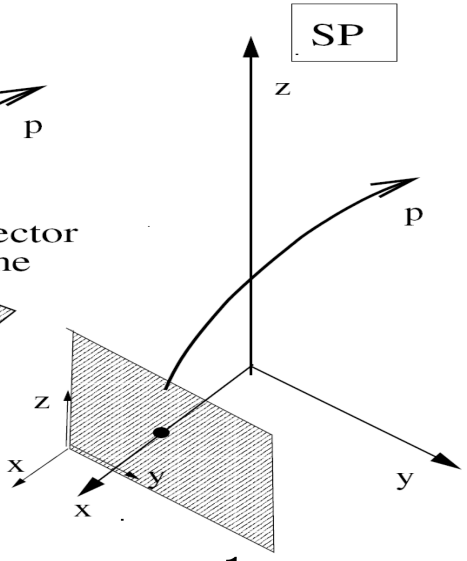
$$\frac{1}{p}, \lambda, \varphi, y_\perp, z_\perp$$

detector frame



$$\frac{1}{p}, v', w', v, w$$

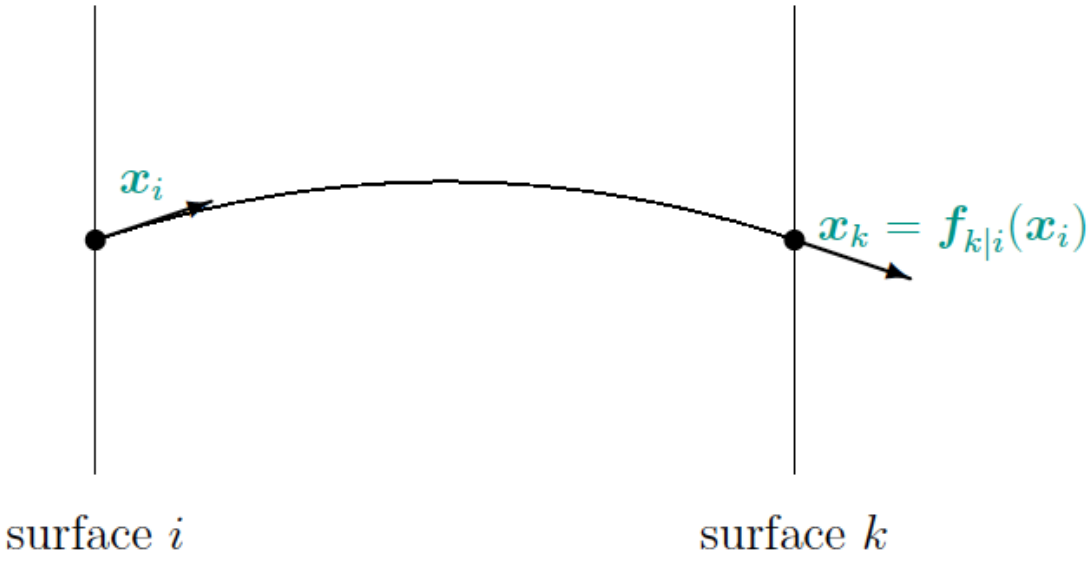
plane frame



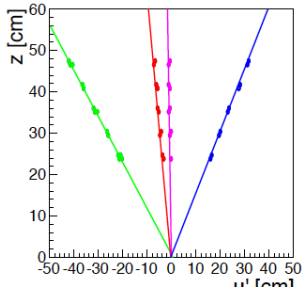
$$\frac{1}{p}, y', z', y, z$$

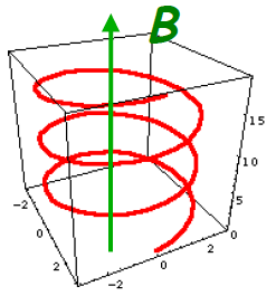
Track Model

- Track Model describes how the state vector at one surface depends on the state vector at a different surface. (describes particle trajectory)
- $$x_k = f_{k|i}(x_i)$$
- $f_{k|i}$ is the track propagator from surface i to surface k.



Track Model





- No magnetic field, no material: straight line
- Homogeneous magnetic field: helix (2D circle)
- Analytic calculation only for simple cases (like above)
- Inhomogeneous B-field, material (energy straggling, small angle scattering): iterative numerical extrapolation of track segments over short distances
  - *Many tracks, many extrapolation steps, need high computing speed*
  - *Simplest: Newton iteration, or describe each segment as a parabola*
  - *Better: numerical integration like “Runge-Kutta”*

Measurement Model

- The measurement model describes the functional dependence of the measured quantities on the state vector at the detector surface:

$$\boldsymbol{m}_k = \boldsymbol{h}_k(\boldsymbol{x}_k)$$

- The vector of measurements m_k usually collects the measured coordinates, but may contain also other quantities, e.g. measurements of direction or even momentum.

- Error on measurement assumed to be Gaussian which is not always true!

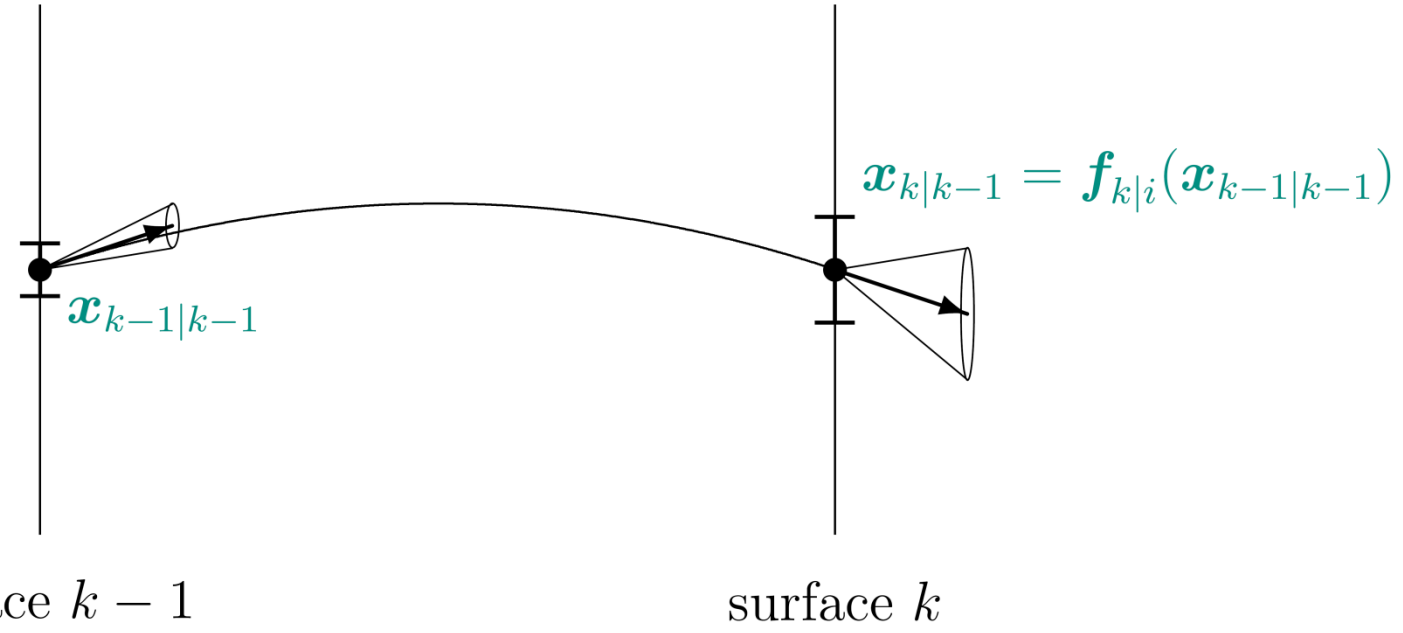
Kalman Filter

- Iterative LS estimation of the state vectors in all measurement layers (or even material layers).

- Two steps are repeated:
  - Prediction: extrapolate the state to the next layer, add up multiple scattering, subtract energy loss
  - Update: combine the predicted state with the current measurement

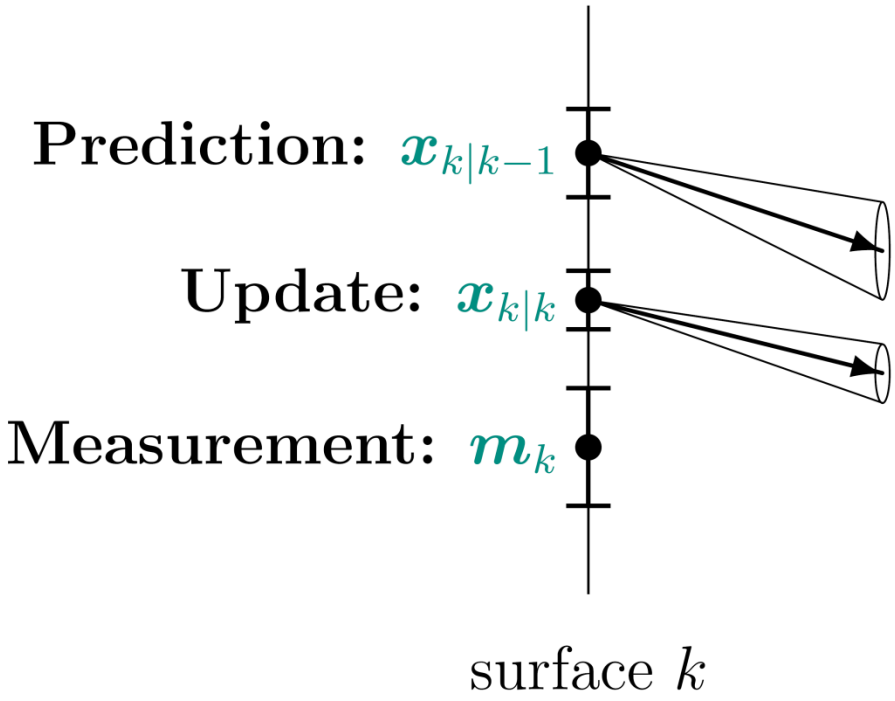
- A good initial state (seed) is important. Its weight should be small.

Prediction stage of Kalman Filter



Prediction step of the Kalman filter

Update stage of Kalman Filter



Update step of the Kalman filter (weighted mean)

Smoother

The last estimate x_n of the filter contains the full information.

The full information can be propagated back to all previous estimates.

This can be done by another iterative procedure, the smoother.

The smoother runs in the opposite direction to the filter.

Smoother

- The smoother can also be realized by combining two filters.
- One filter runs from m_1 to m_n: forward filter .
- One filter runs from m_n to m_1: backward filter .
- The smoothed states are the weighted mean of the predicted states one filter and the updated states of the other filter.
- This is numerically more stable.
- The smoothed state $x_{0|n}$ is the best linear estimate and therefore the same as the usual Least-Squares estimate x

Kalman Filter

The Kalman filter can therefore be used for track finding.

Generate a track seed.

Iterate the following sequence:

- Extrapolate and look for compatible measurements.
- If there is none, go on.
- If there is one, take the most compatible one and make an update.
- If no compatible measurements can be found in several layers, drop the track candidate.

Kalman Filter in PandaRoot

JÜLICH FORSCHUNGSZENTRUM

Genfit

- Originally started as part of PandaRoot as a generic fitting package written by TU Munich
- Now independent external package
- Exists (at the moment) as two implementations: Genfit 1 and Genfit 2

- Genfit 1 is a very old version of Genfit using GEANE as track propagator
- Genfit 2 is a more up-to-date version using a Runge Kutta track propagator → change to recent Genfit 2 planned for next release

- Tests of performance are ongoing in PandaRoot

Genfit

JÜLICH FORSCHUNGSZENTRUM

Offers different track fitters:

Kalman Fitter

- standard track fitting algorithm
- sequential
- equivalent to least squares method

Deterministic Annealing Filter (DAF)

- sequence of Kalman filters
- annealing procedure for outlier rejection, ambiguity resolution

Generalized Broken Lines

- alternative to Kalman filter
- well-suited to Millipede II alignment
- treats track as a whole

Usage of Genfit in PandaRoot

```
PndRecoKalmanTask* recoKalman = new PndRecoKalmanTask();
recoKalman->SetTrackInBranchName("SttMvdGemTrack");
recoKalman->SetTrackOutBranchName("SttMvdGemGenTrack");
recoKalman->SetBusyCut(50); // CHECK to be tuned
//recoKalman->SetIdealHyp(kTRUE);
//recoKalman->SetNumIterations(3);
recoKalman->SetTrackRep(0); // 0 Geane (default), 1 RK
//recoKalman->SetPropagateToIP(kFALSE);
fRun->AddTask(recoKalman);
```

Usage of Genfit2 in PandaRoot

```
PndRecoKalmanTask2* recoKalman = new PndRecoKalmanTask2();
recoKalman->SetTrackInBranchName("SttMvdGemTrack");
recoKalman->SetTrackOutBranchName("SttMvdGemGenTrack");
recoKalman->SetBusyCut(50); // CHECK to be tuned
//recoKalman->SetIdealHyp(kTRUE);
//recoKalman->SetNumIterations(3);
//recoKalman->SetTrackRep(0); // 0 Geane (default), 1 RK
//recoKalman->SetPropagateToIP(kFALSE);
fRun->AddTask(recoKalman);
```

TRACK PROPAGATION - GEANE

GEANE

- Track follower / propagator
- Simplified geant3 tracking algorithm + error propagation
- Original Code is in Fortran
- Distributed with geant3 by CERN
- Part of FairRoot

- It extrapolates both the mean values of the parameters and their covariance matrices
- It uses the MC geometry
- It takes into account:
  - material effects
  - magnetic field
  - physical effects

- Different reference frames
- Different kinds of propagation

Geane Interface

In `geane` you will find:

- `FairGeane`: this is the task reading in the physics cuts and the physics effects to take into account
- This task has to be added to your macro otherwise GEANE will not work!
- `FairGeanePro`: contains the propagation stuff

- In `trackbase` you will find
- `FairTrackPar/ParP/ParH`: track representations of GEANE
- `FairGeneUtil`: frame transformations from to MARS, SC, SD, SP (C++ translation of FORTRAN methods)

How to use it

- First define the start plane:

```
Bool_t PropagateFromPlane(TVector3& v1, TVector3& v2);
```

- Define the end position:

```
Bool_t PropagateToPlane(TVector3& v0, TVector3& v1, TVector3& v2);
Bool_t PropagateToVolume(Tstring VolName, Int_t CopyNo,Int_t option);
Bool_t PropagateToLength(Float_t length);
Bool_t PropagateToPCA(Int_t pca, Int_t dir);
```

- Do the propagation:

```
Bool_t Propagate(FairTrackParH* TStart, FairTrackParH* Tend, Int_t PDG)
Bool_t Propagate(FairTrackParP* TStart, FairTrackParP* TEnd, Int_t PDG)
```

- Back propagation:

```
Bool_t setBackProp()
```

How to use it

- First define the start plane:

```
Bool_t PropagateFromPlane(TVector3& v1, TVector3& v2);
```

- Define the end position:

```
Bool_t PropagateToPlane(TVector3& v0, TVector3& v1, TVector3& v2);
Bool_t PropagateToVolume(Tstring VolName, Int_t CopyNo,Int_t option);
Bool_t PropagateToLength(Float_t length);
Bool_t PropagateToPCA(Int_t pca, Int_t dir);
```

- Do the propagation:

```
Bool_t Propagate(FairTrackParH* TStart, FairTrackParH* Tend, Int_t PDG)
Bool_t Propagate(FairTrackParP* TStart, FairTrackParP* TEnd, Int_t PDG)
```

- Back propagation:

```
Bool_t setBackProp()
```

Track Propagator

- Additional track propagators:
  - `FairRKPropagator`
    - *4th order Runge-Kutta propagator*
    - *not used*
  - `PndHelixPropagator`
    - *much faster than GEANE*
    - *but no energy loss*
    - *no varying magnetic field*

TRACKING QA

Introduction

- Various different tracking algorithms
- Common basis needed to compare and improve tracking algorithms
- Common development of Lia Lavezzi and me

- Idea:
  - Have a set of macros which work on any tracking algorithms which produces PndTrack and PndTrackCand
  - Set of identical simulation and digitization files for all tracking algorithms

Criteria

- Track efficiency
  - How many MC tracks have been found by track finder
- Hit efficiency
  - How many hits of a MC track have been found by the track finder
- Purity
  - Belong all hits of one found track belong to one MC track

- Clones
  - How often was a track found by the track finder
- Ghosts
  - How many hits not belonging to an MC track have been found

Criteria

- Partially found
  - Not all hits belonging to one track have been found but all hits belong to one MC track
- Spurious found
  - > 70% found hits belong to one MC track
- Fully found
  - 100 % of MC hits have been found and no other hits are part of the track
- Clone
  - A MC track was found more than once
- Ghost
  - A reconstructed track not matching to an MC track

Efficiency

- To determine the efficiency of a track finder a basis is needed:
  - All tracks? What about neutral particles or tracks without hits?
  - Only tracks with hits? How many hits?
  - A barrel track finder cannot find tracks in the forward region

- Different criteria defined:
  - At least three hits (minimum for circle determination)
  - Possible Track:
    - *Definable by user via functor (who knows what a functor is?)*
    - *Some predefined in PndTrackers/PndIdealTrackFinder/ PndTrackFunctor.h*

Efficiency

- Predefined functors:
  - StandardTrackFunctor:
    - *MVD > 3 or MVD+Stt+GEM > 5*
  - FtsTrackFunctor
    - *FTS > 5*
  - NoFtsTrackFunctor
    - *FTS == 0 and StandardTrackFunctor*
  - Some more specific for certain track finders
    - *OnlySTTFunctor, RiemannMvdSttGemFunctor, ...*

Implementation

- Macros in `/macro/trackingQA`
- sim, digi and par files provided in folder

- **`reco_complete.C`**`(TString prefix)`
  - Does the tracking. Your tracking code has to go here!
  - In addition runs an ideal track finder to compare with
  - Usage:
    - *Add prefix to identify which data you want to run on*
    - `root reco_complete.C"(\"dpm_G4_7G0_1000\")"`

Implementation

- **`trackingQA_complete.C`**`(TString prefix, TString branch, Int_t nEvents=0):`
  - Compares tracking algorithms with ideal tracking
  - Usage
    - *Add the same prefix as in reco_complete.C*
    - *Add the branch name with the final results of your tacking code (has to be of type PndTrack)*
    - *Optional add the number of events you want to run on*
    - *`root trackingQA_complete.C"(\"dpm_G4_7G0_1000\", \"SttMvdGemTrack\")"`*

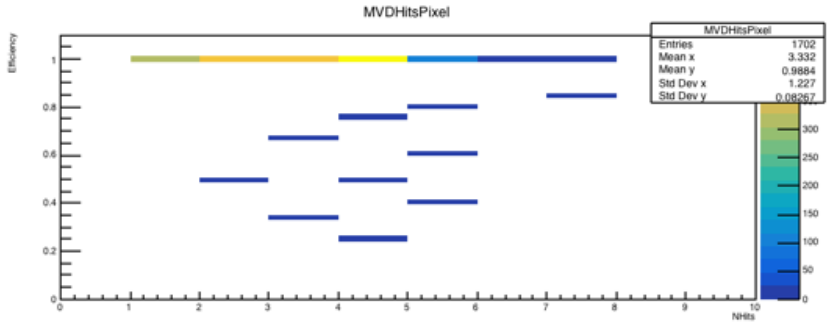
Implementation

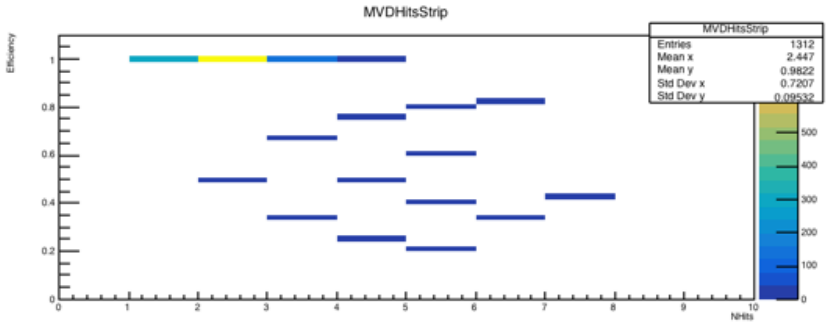
- **`plot_trackingQA`**`(TString fileName)`
  - Plots the generated histograms of trackingQA in a nice way
  - Usage:
    - *Give the complete file name of the trackingQA.root file*

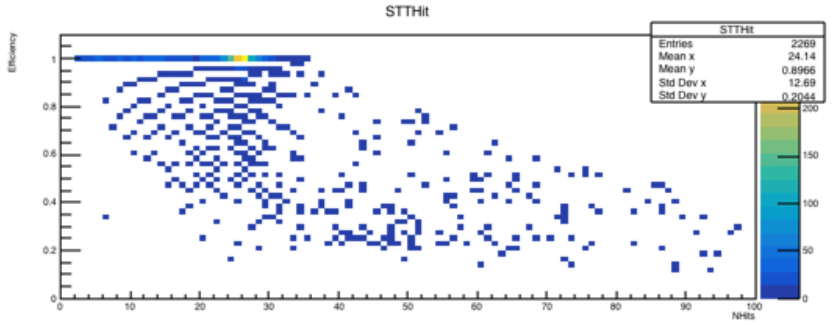
- **`QA_histos`**`(TString prefix, TString trackBranch)`
  - Generates a large set of extracted histograms
  - Usage:
    - *Give prefix and branchName*

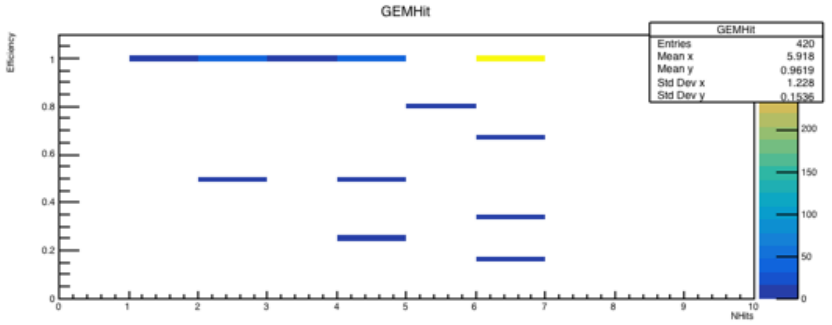
- **`comp_recoqa`**`(TString ,fn TString fn2, int picpercan, …)`
  - Combines the histograms of two different tracking algorithms
  - New feature **pictures per canvas**

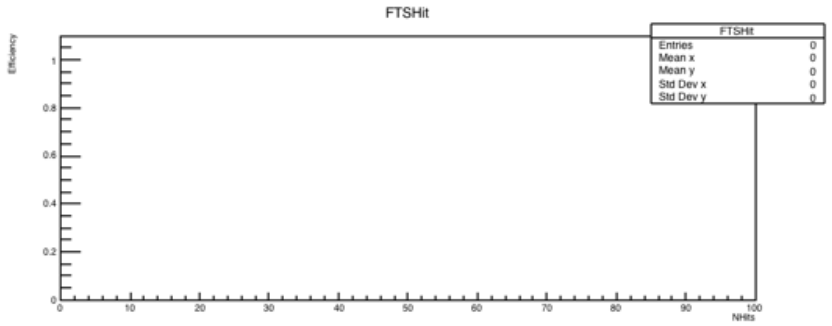
Hit Efficiency



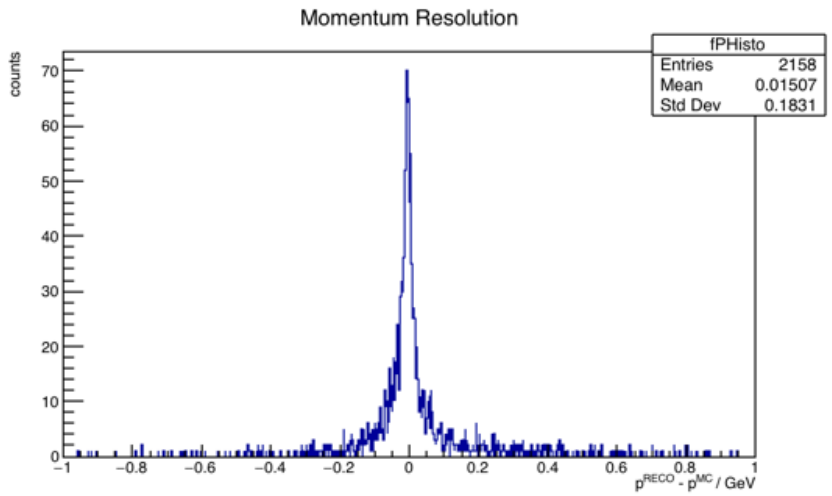


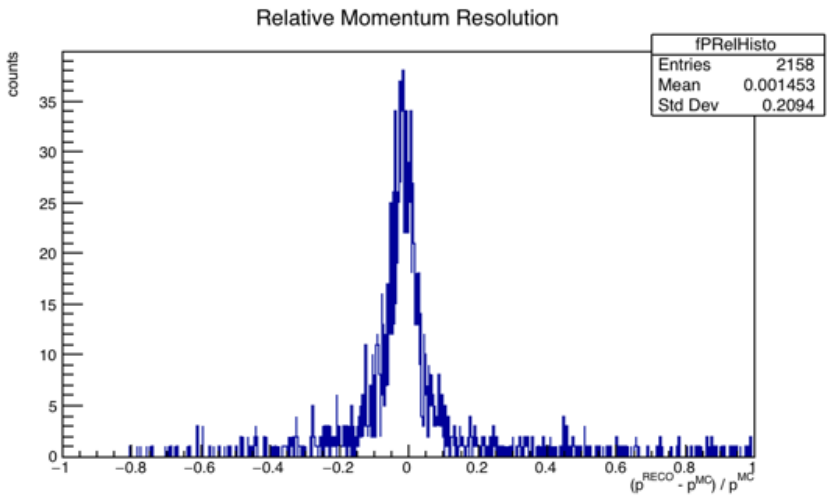


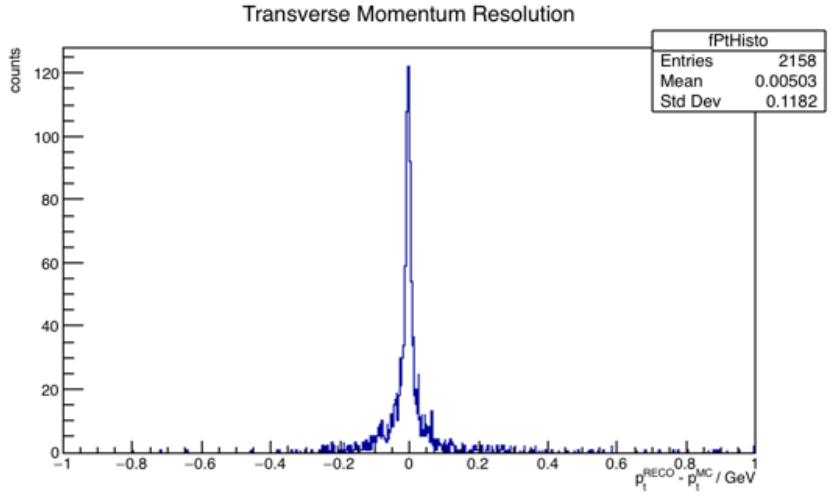


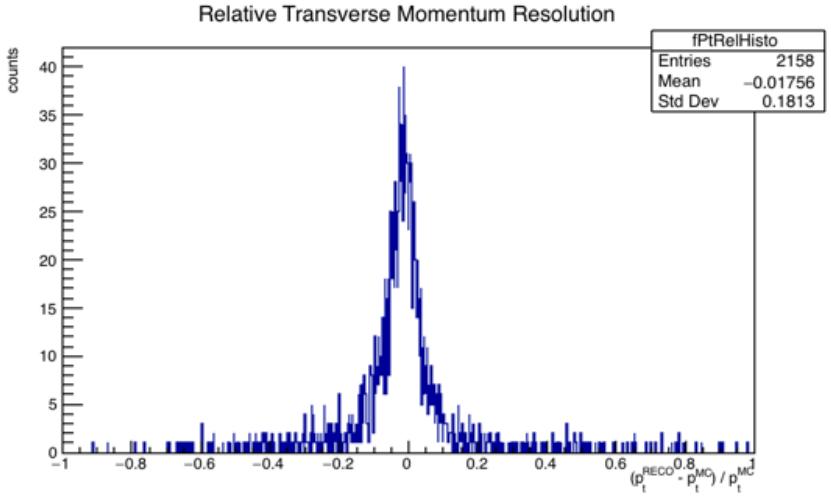


Momentum Resolution

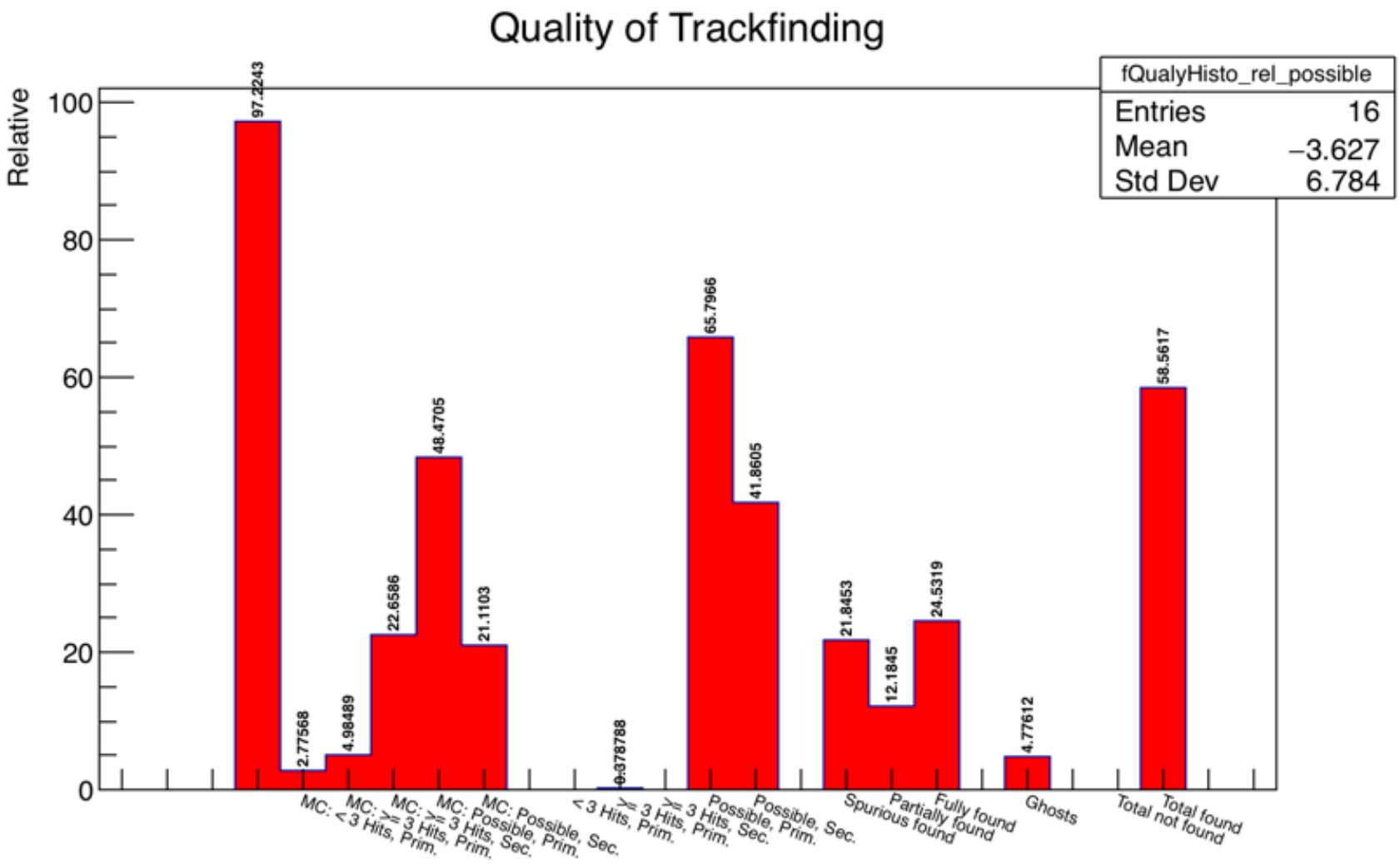




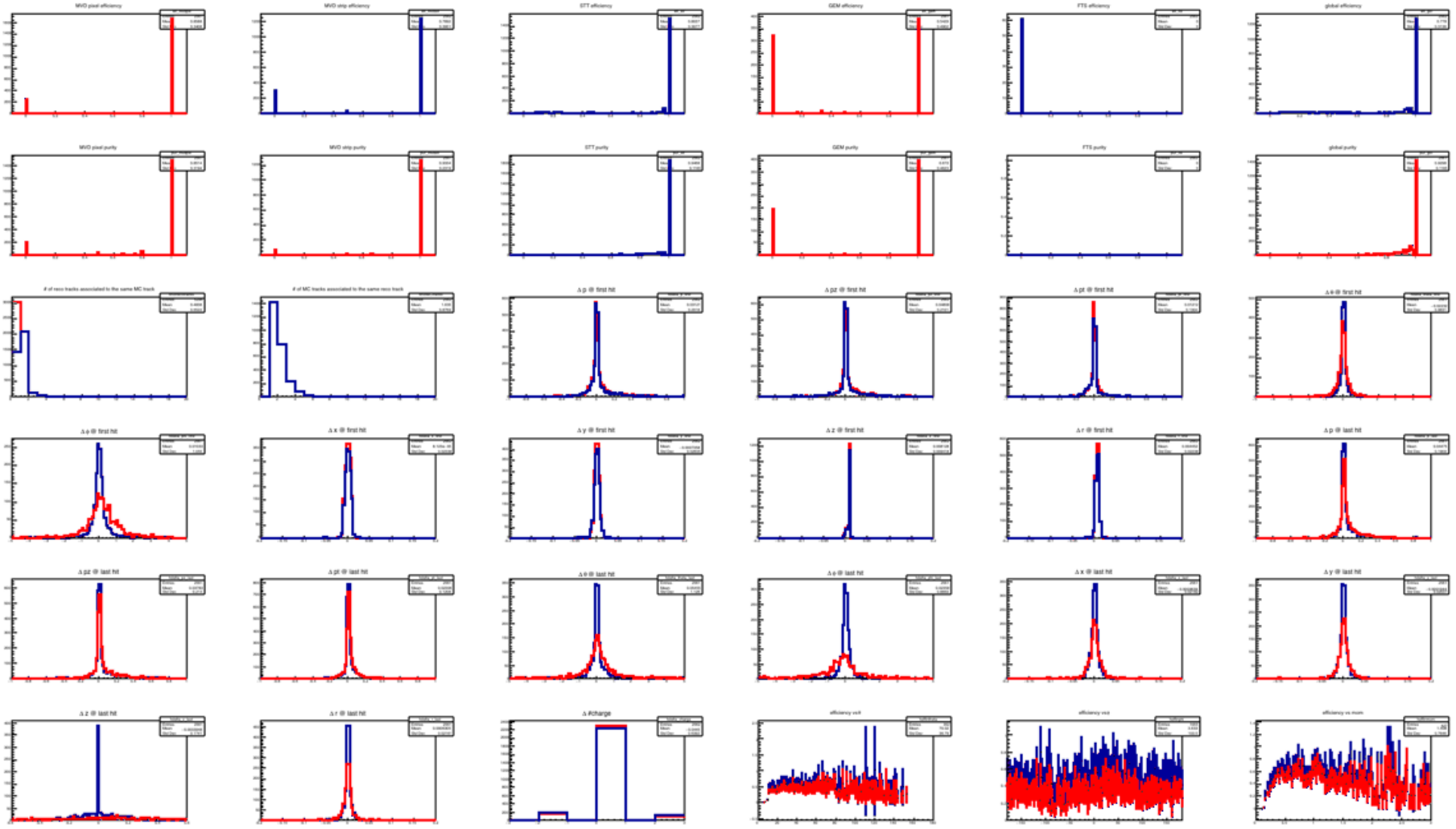




Performance



Overview Performance



Overview of Track Finder

JÜLICH FORSCHUNGSZENTRUM

TRACKING

Algorithm name	Attached Person	Platform	In PandaRoot?	Output PndObjects?	SVN?	Online?	Secondaries?	Barrel/Fwd	Used Detectors	Notes	Links
PndTrkTracking2	Gianluigi	CPU	yes	PndTrack	yes	no	no	Barrel	MVD STT GEM	Main offline track finder	Gianluigimeet8sep15.pdf
BarrelTracker	Radoslaw Karabowicz	CPU	yes	PndTrack	yes	no	no	Barrel	MVD/STT/GEM		
Barrel Cellular Automaton	Ivan Kiesel	CPU	yes	PndTrack	yes			Barrel	MVD(Barrel) STT	Does not use the MVD disks	
Genfit1		CPU	yes	PndTrack	yes	no	yes	Barrel/Forward	all	old version of generic kalman track fitter	
Genfit2		CPU	yes	PndTrack	yes	no	yes	Barrel/Forward	all	current version of generic kalman	