supernemo collaboration

Hitchhiker’s guide to Cimrman module

A Mostly Harmless Guide to Track Reconstruction

Tomáš Křižák

Collaboration meeting | July 2026


"Space is big.
It's really, really big.
You just won't believe
how vastly, hugely,
mind-bogglingly big it is."
– The Guide
ββ candidate
SUPERnEMO
HITCHHIKER'S GUIDE TO CIMRMAN MODULE
A MOSTLY HARMLESS GUIDE TO TRACK RECONSTRUCTION
CIMRMAN
HOW TO RECONSTRUCT A TRACK (MOSTLY HARMLESS)
1. Collect hits
2. Build candidates
3. Fit in Legendre space
ρ
z
4. Select the best one
z
y
y
x
DON'T PANIC.
JUST RECONSTRUCT.
42
Vertex
Track candidate
Likelihood
ℒ(θ)
Time
t
Mostly Harmless
Drift hits
Reconstructed track
Calorimeter cluster
DON'T PANIC

Topics

1. Full algorithm overview + code implementation

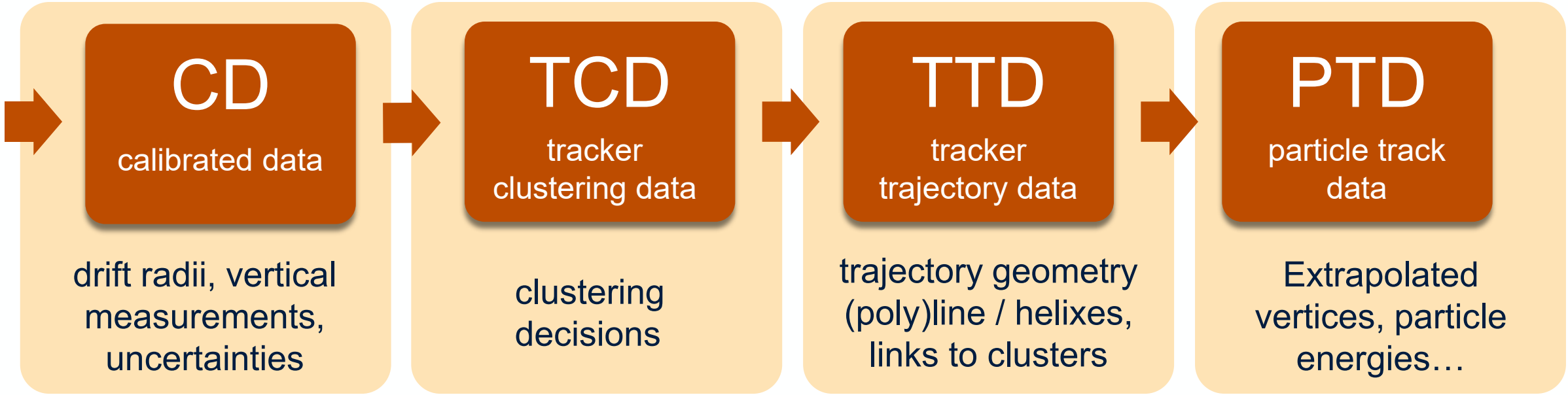
Topics

1. Full algorithm overview + code implementation
2. New polyline reconstruction framework

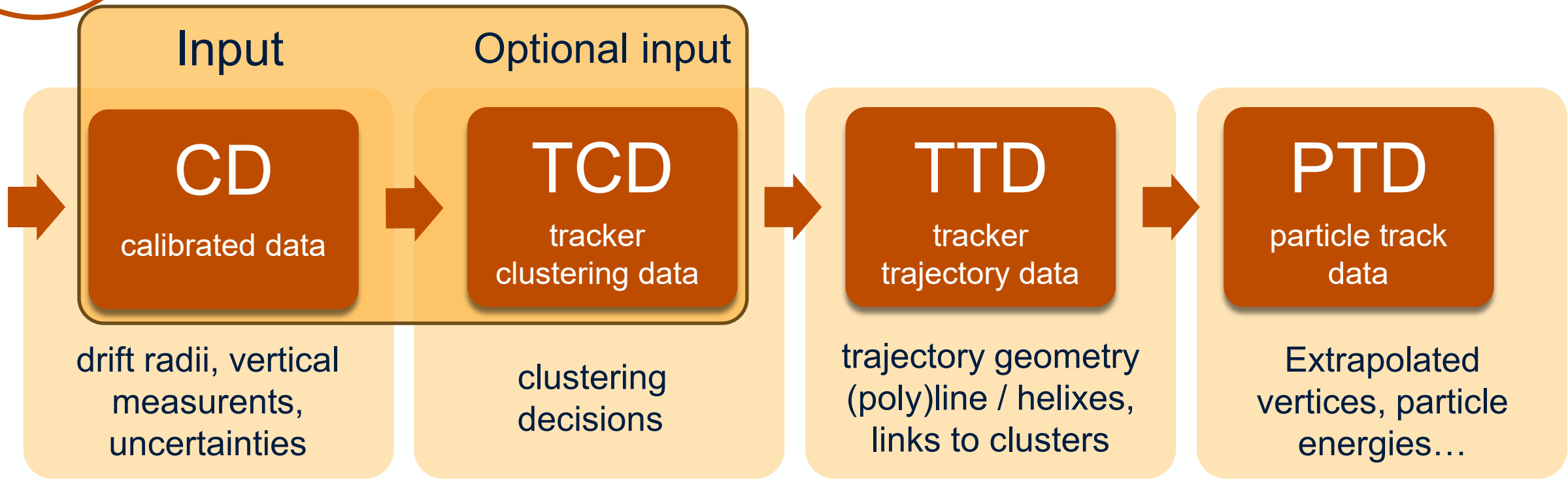
supernemo collaboration

1. Algorithm overview

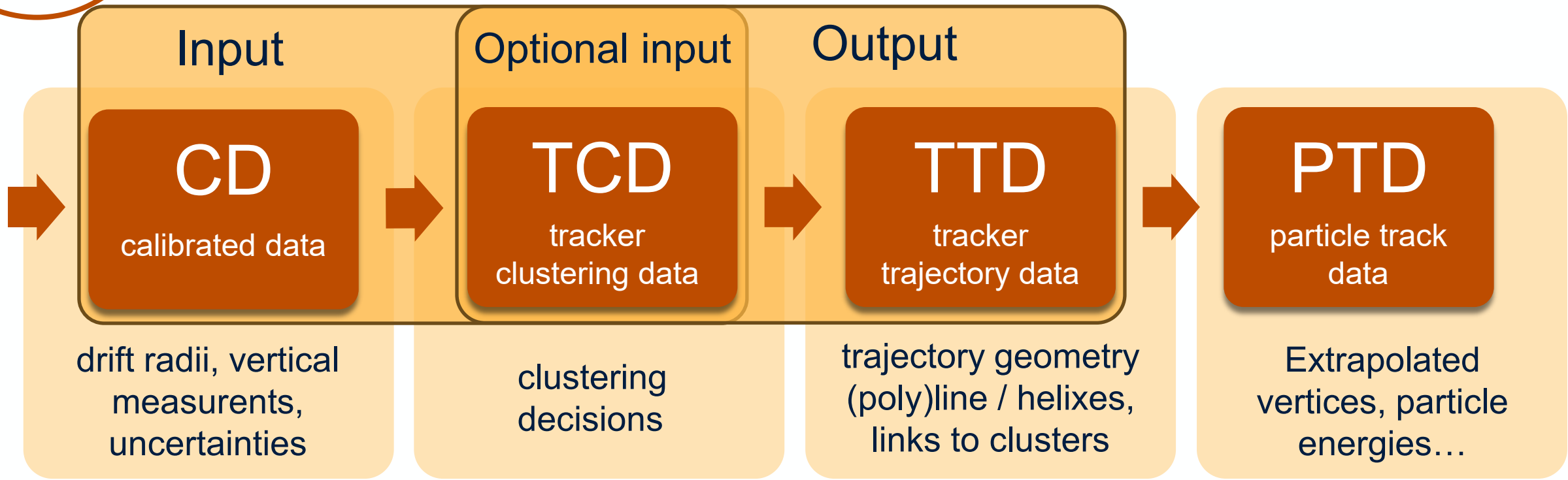
Cimrman's place in the pipeline



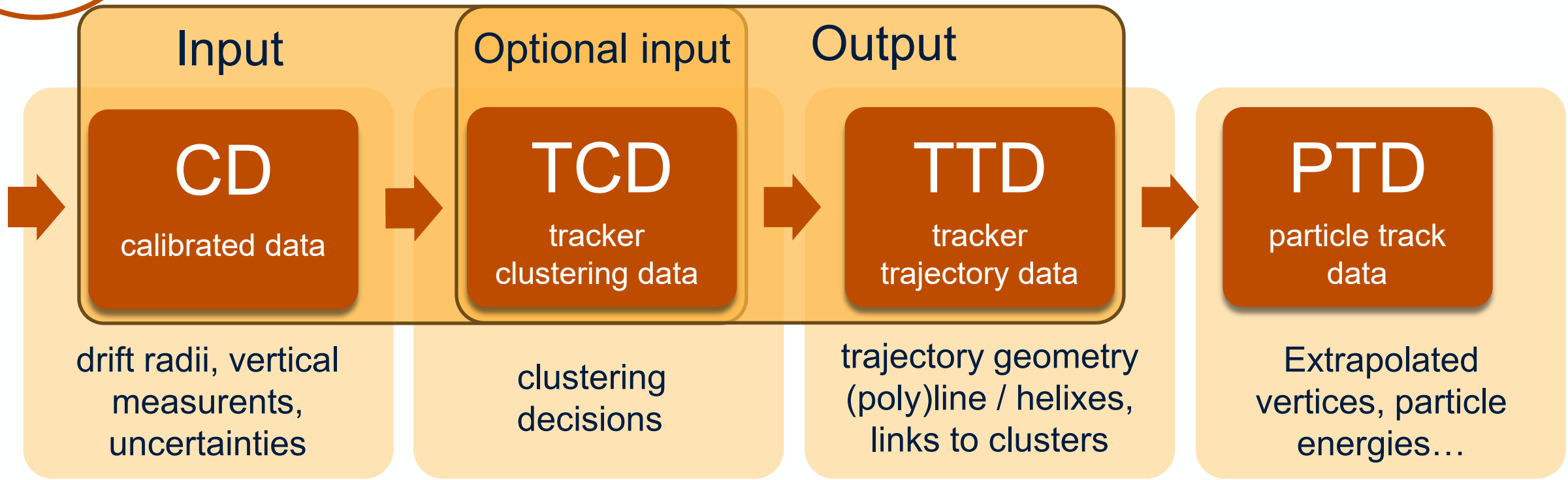
Cimrman's place in the pipeline



Cimrman's place in the pipeline



Cimrman's place in the pipeline



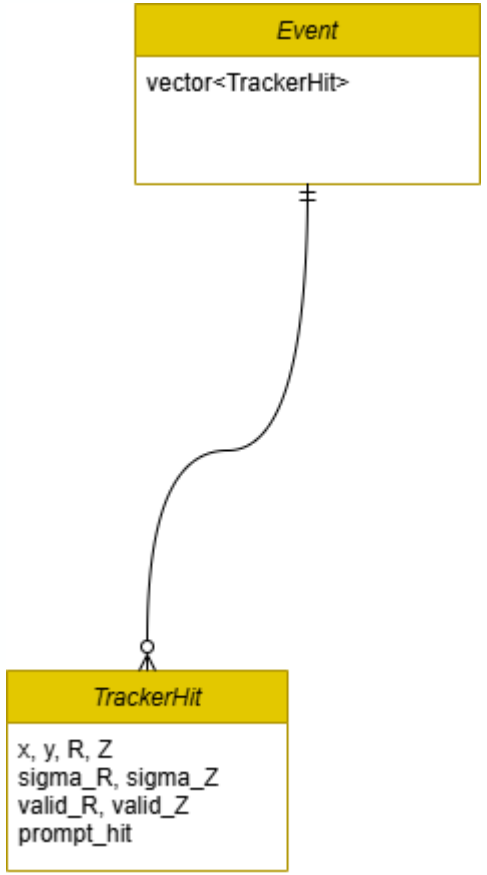
Warning: Cimrman currently changes existing TCD bank. Should create a new bank instead.

Cimrman.cpp

- What happens when you run flreconstruct with Cimrman?
- Once at the beginning:
  Cimrman::initialize(…)

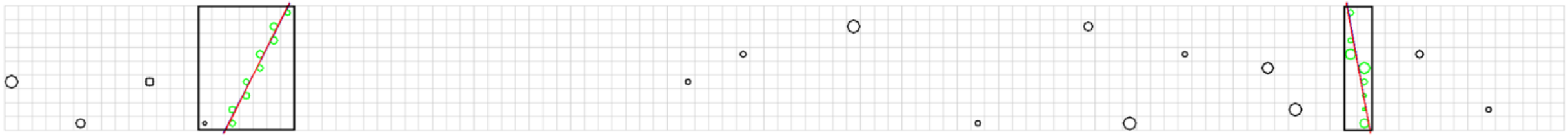
Cimrman.cpp

- What happens when you run flreconstruct with Cimrman?
- Once at the beginning:
  Cimrman::initialize(…)
- Algos: Cimrman's internal worker class
- Per event:
  Cimrman::process(…) {
    CD, (TCD) → Cimrman data structure
    Algos::process(…)
    Cimrman data structure → TCD, TTD
  }


Event
vector<TrackerHit>
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit

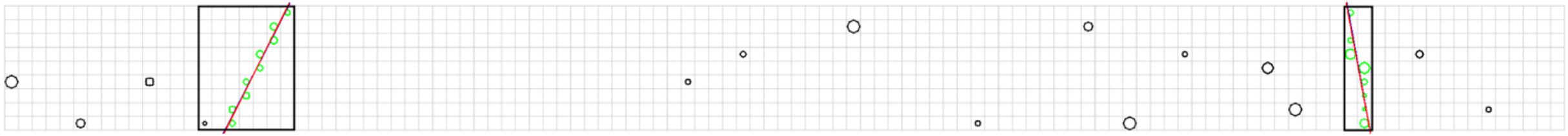
Step 1: preclustering

- Division of event into subevents = ***preclusters***
- Based on:
  - (horizontal 2D) distance



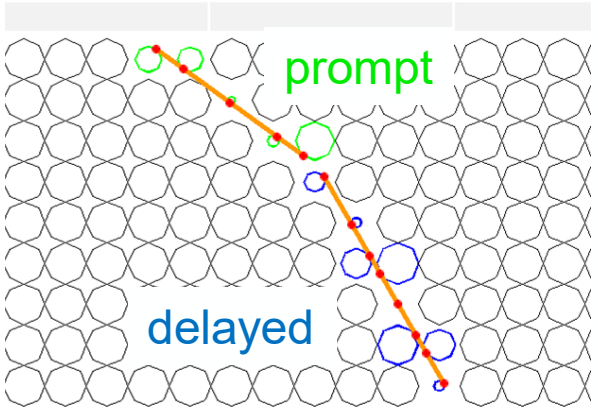
Step 1: preclustering

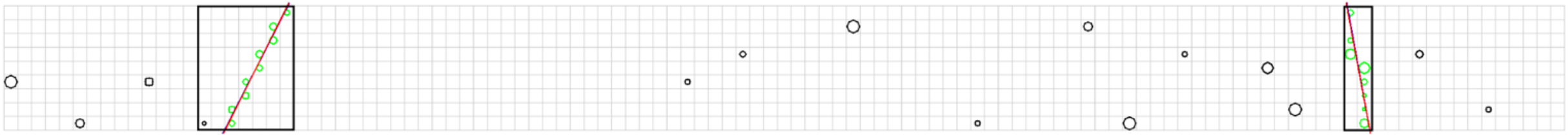
- Division of event into subevents = ***preclusters***
- Based on:
  - (horizontal 2D) distance
  - trigger time (different ref OM = different tracks)



Step 1: preclustering

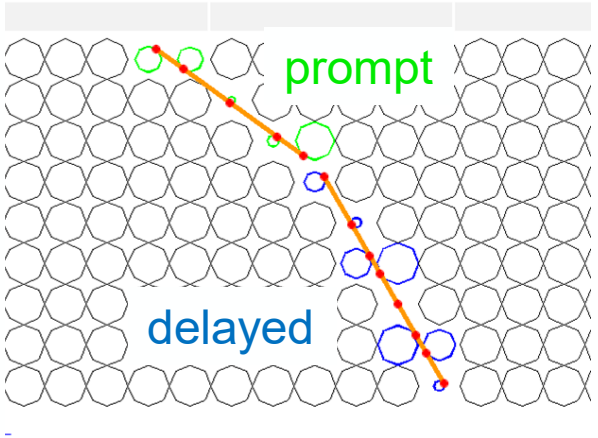
- Division of event into subevents = ***preclusters***
- Based on:
  - (horizontal 2D) distance
  - trigger time (different ref OM = different tracks)
  - delayed/prompt

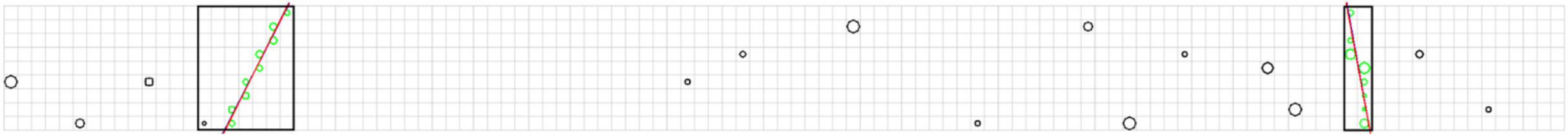




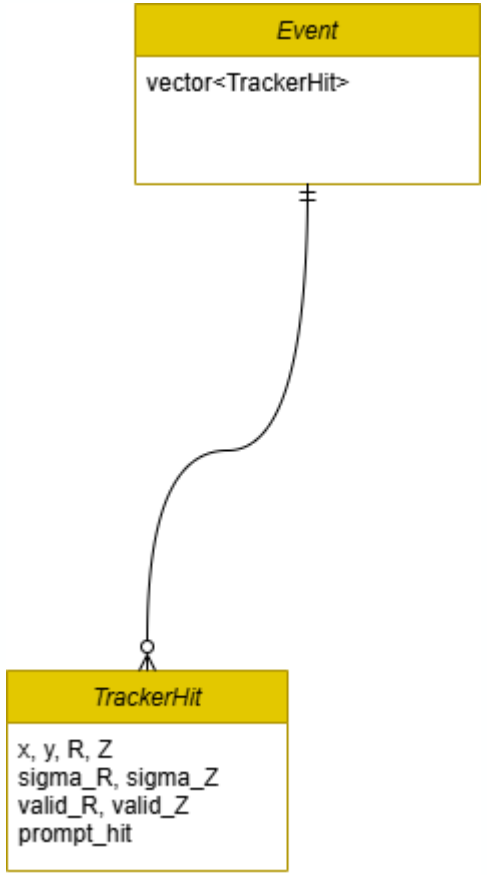
Step 1: preclustering

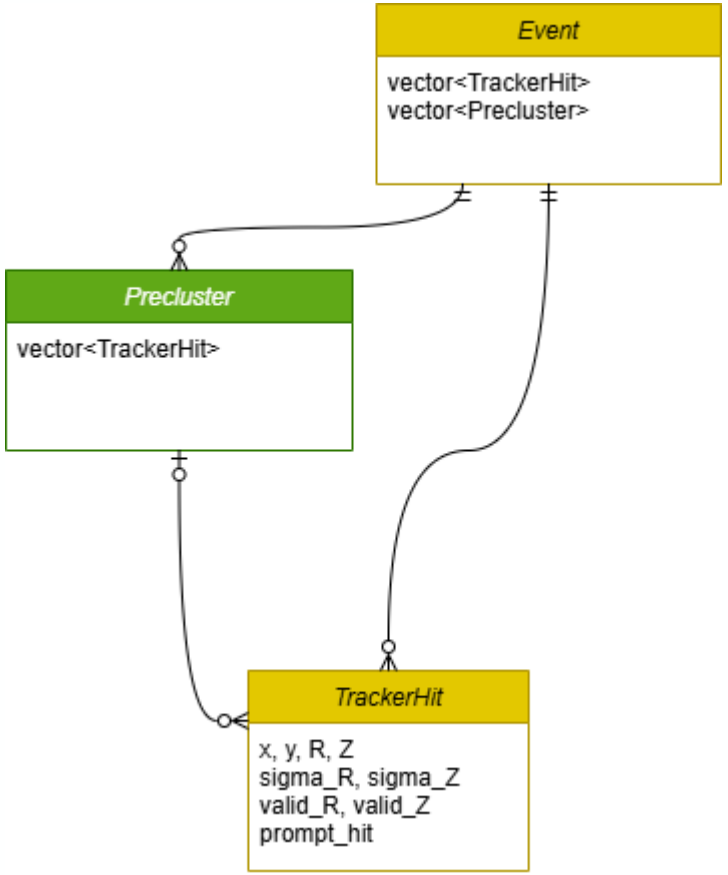
- Division of event into subevents = ***preclusters***
- Based on:
  - (horizontal 2D) distance
  - trigger time (different ref OM = different tracks)
  - delayed/prompt





- **Rest of reconstruction done for each *precluster* separately**
  (saves time, avoids exponential blowup due to ambiguities)


Event
vector<TrackerHit>
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit

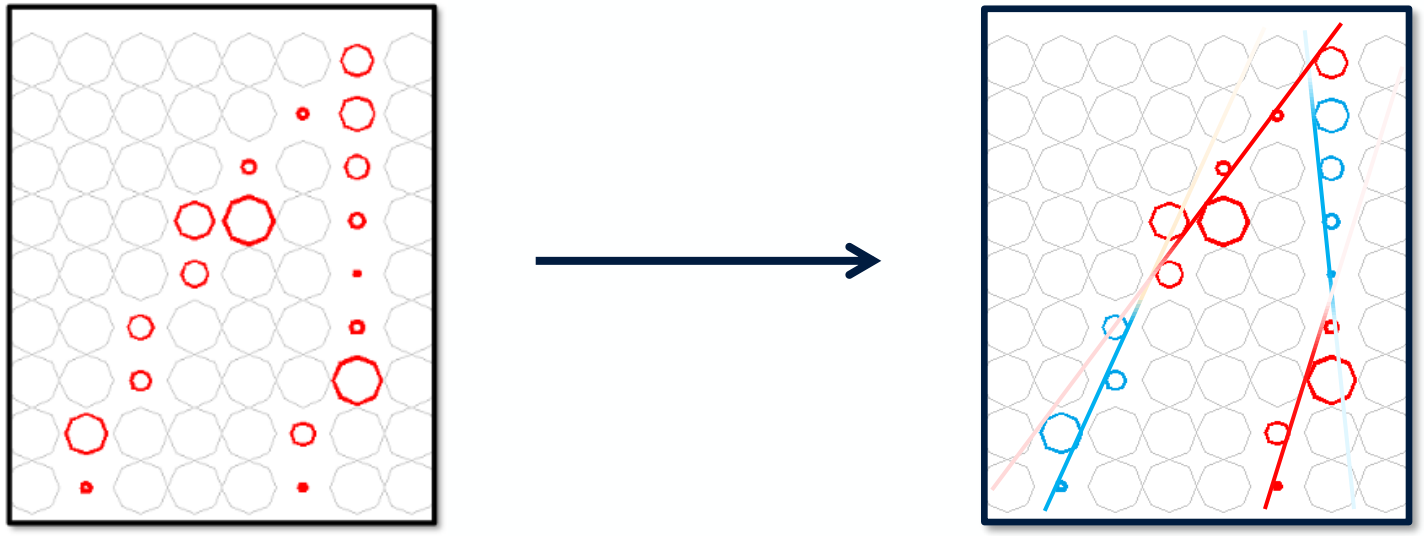

Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit

Step 2: clustering and line estimation

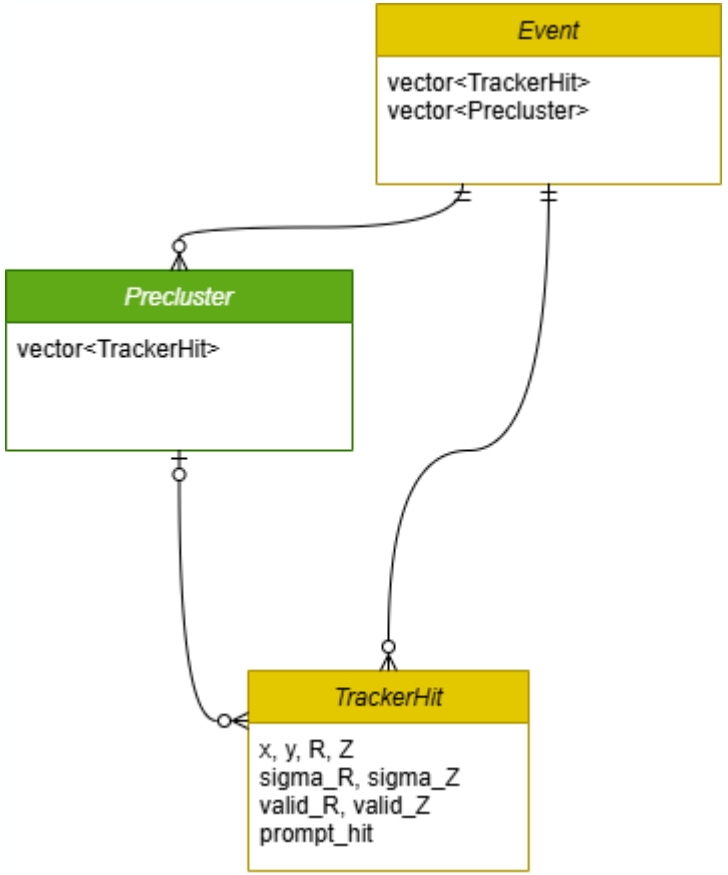
- Legendre transform approach (detailed explanation in article 1)
  - Dedicated worker class for clustering (*Sinogram* class)

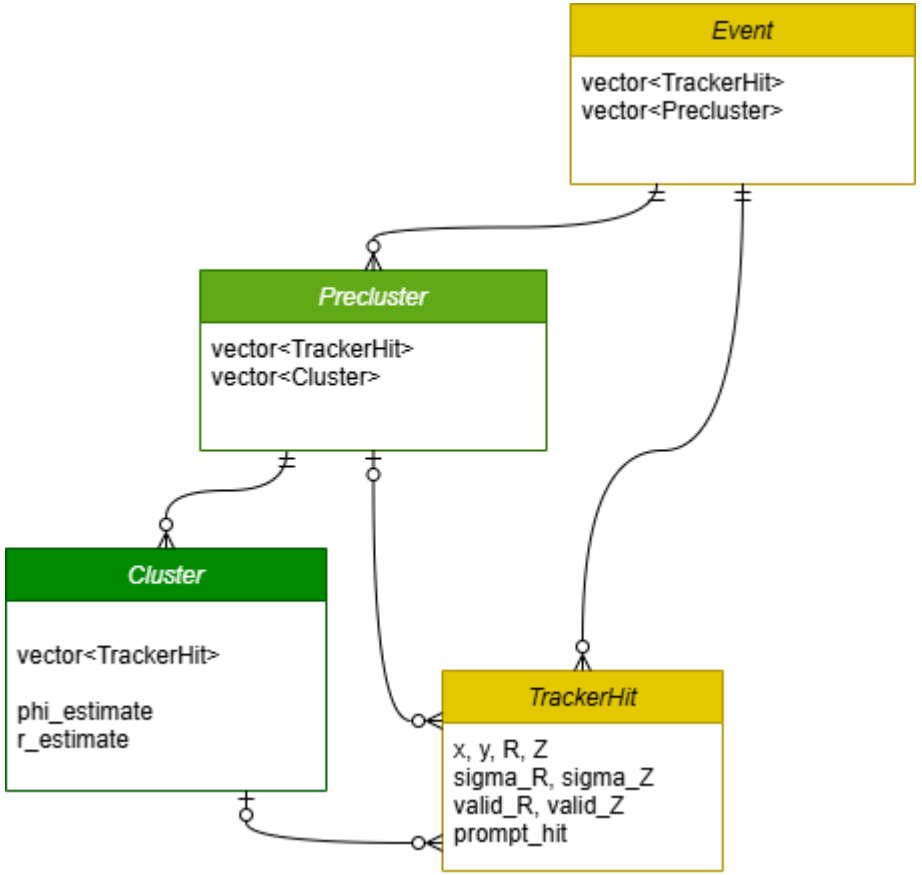
Step 2: clustering and line estimation

- Legendre transform approach (detailed explanation in article 1)
  - Dedicated worker class for clustering (*Sinogram* class)
- Two outputs:
  - **Line fit estimates** (only horizontal part of the fits)
  - their pairing to tracker hits (***clusters***)



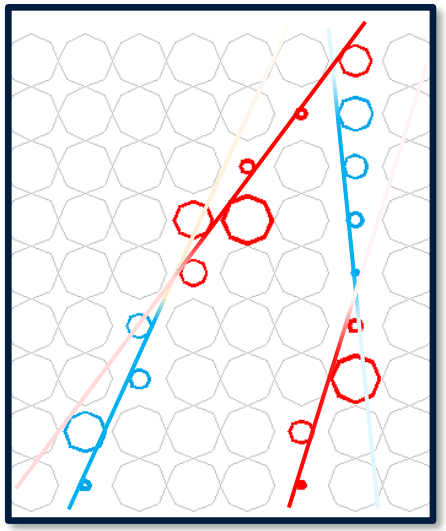
(4 clusters, 4 lines estimated)


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
Cluster
vector<TrackerHit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit

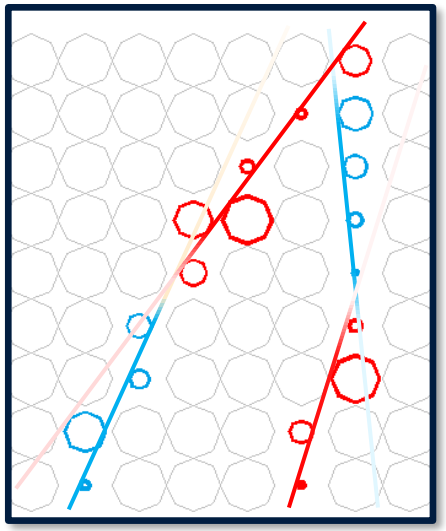
Step 3: ambiguity checking and fitting

- Checks each cluster for **symmetry** (ambiguities)



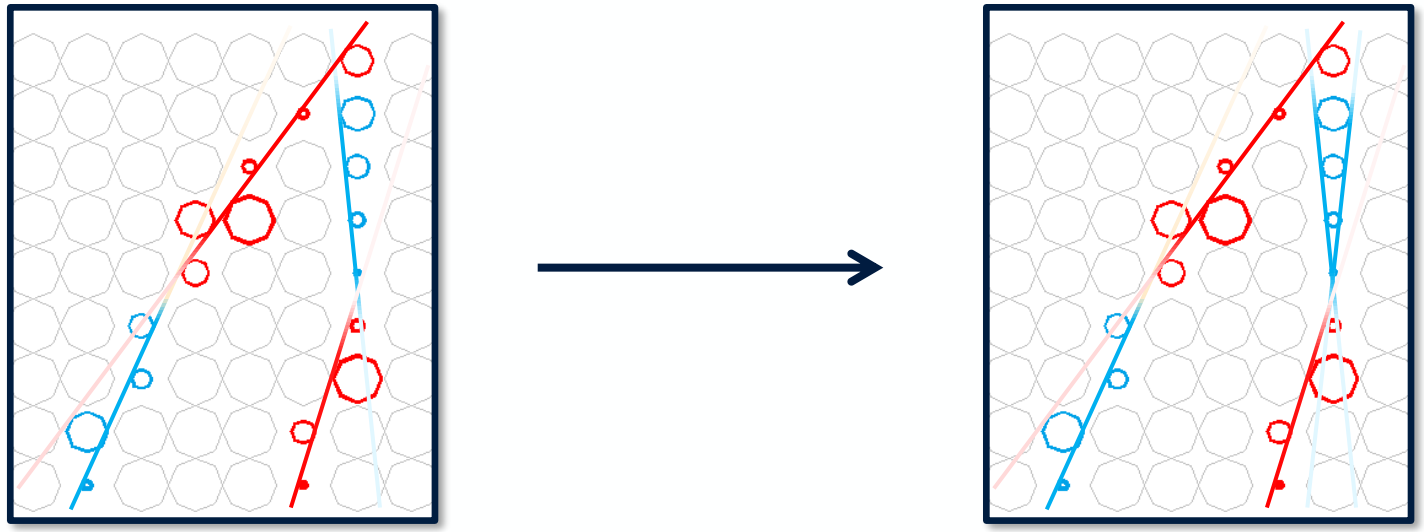
Step 3: ambiguity checking and fitting

- Checks each cluster for **symmetry** (ambiguities)
- **Maximum likelihood method** for fitting → ***LinearFit***
  - Refines existing horizontal estimates
  - Adds vertical components of the fits



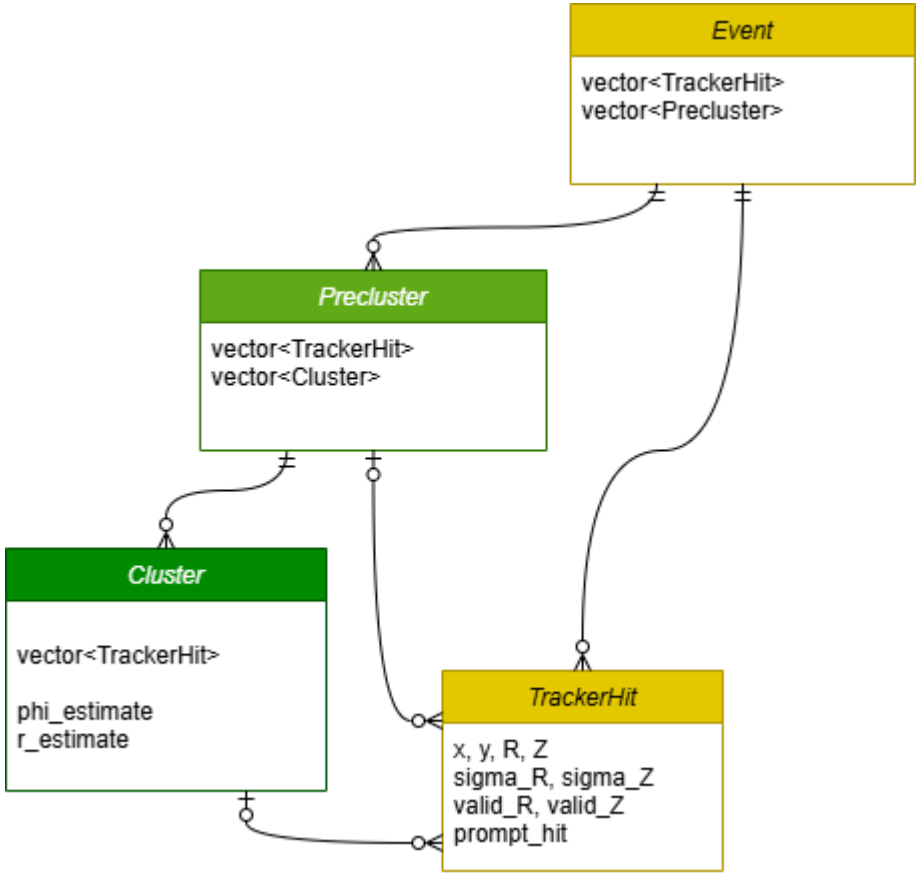
Step 3: ambiguity checking and fitting

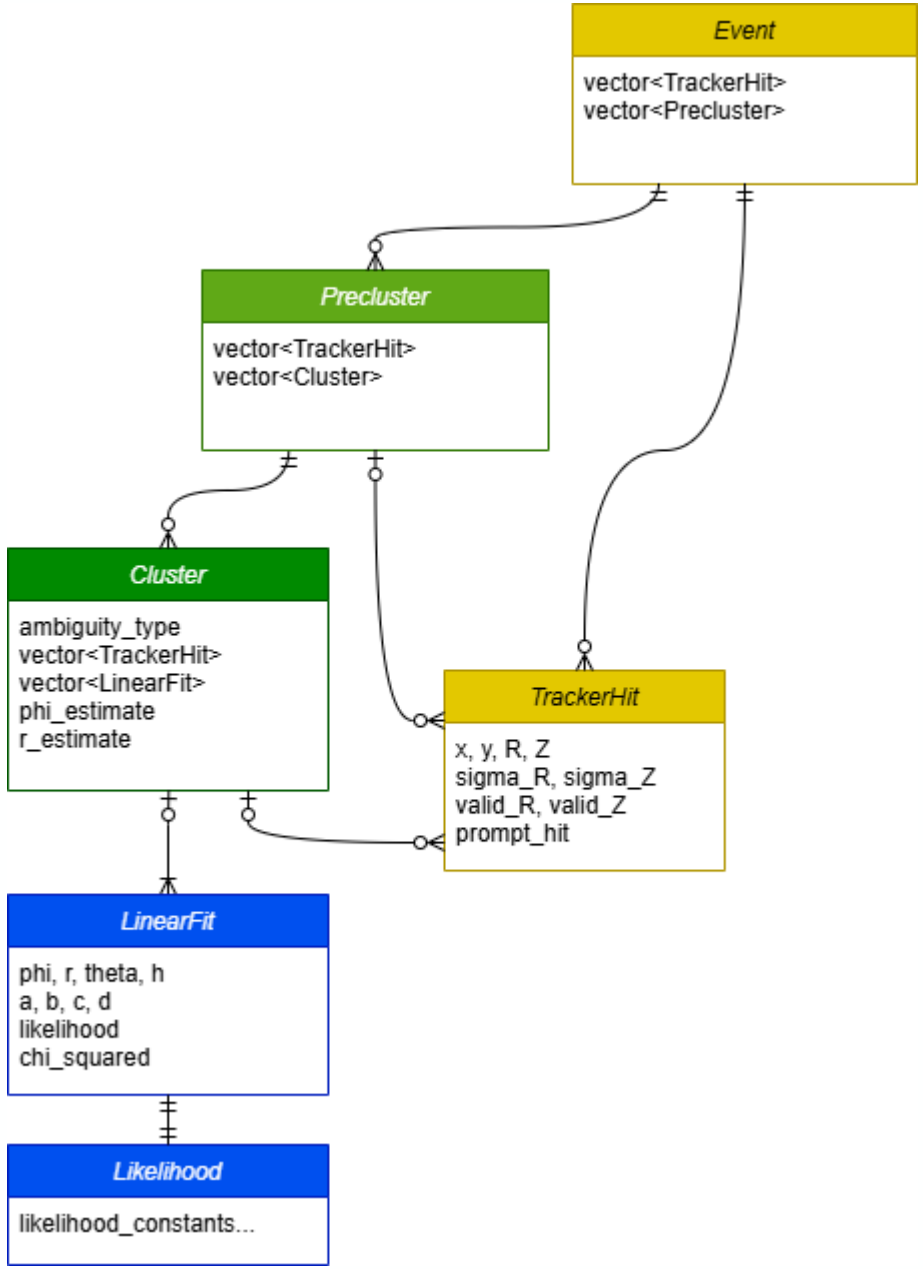
- Checks each cluster for **symmetry** (ambiguities)
- **Maximum likelihood method** for fitting → ***LinearFit***
  - Refines existing horizontal estimates
  - Adds vertical components of the fits



4 lines estimated

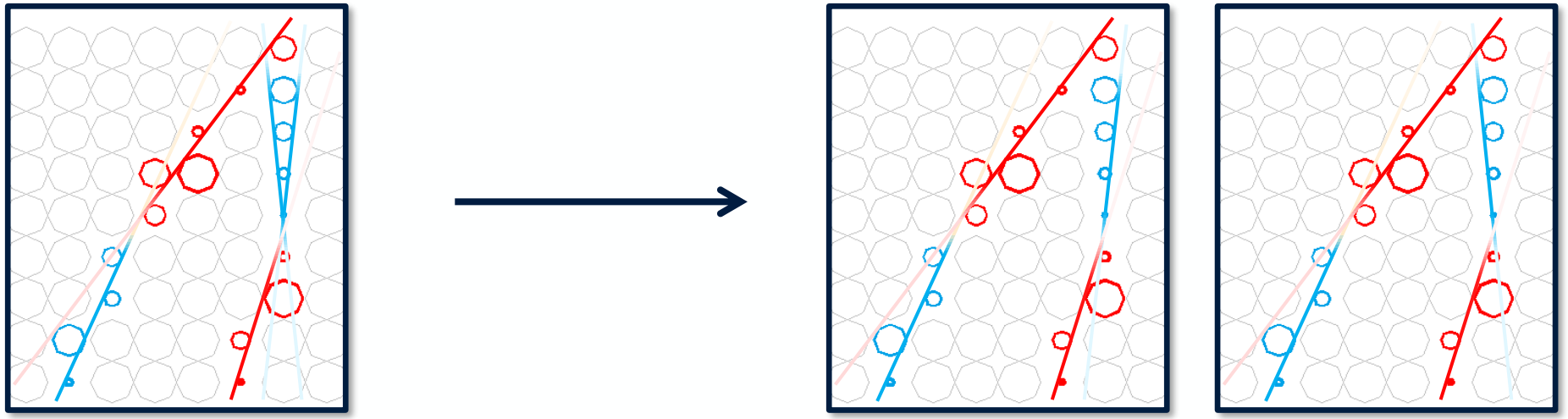
5 lines fitted


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
Cluster
vector<TrackerHit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Likelihood
likelihood_constants...

Step 4: Precluster solutions

- Creates "***precluster solutions"*** = unique collection of fits explaining the precluster
- Rest of the process done separately for each *precluster solution*



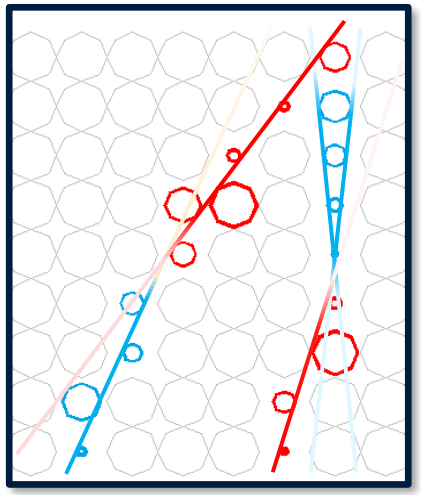
4 clusters, 5 fits

Precluster solution 1

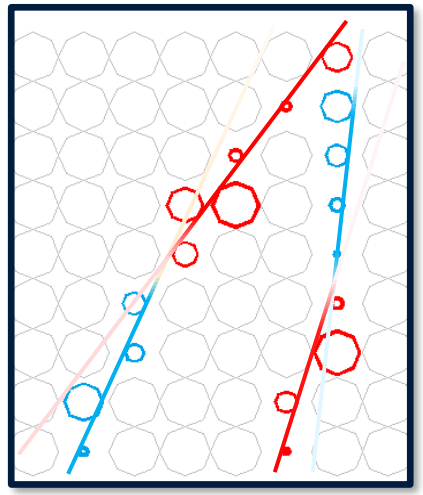
Precluster solution 2

Step 4: Precluster solutions

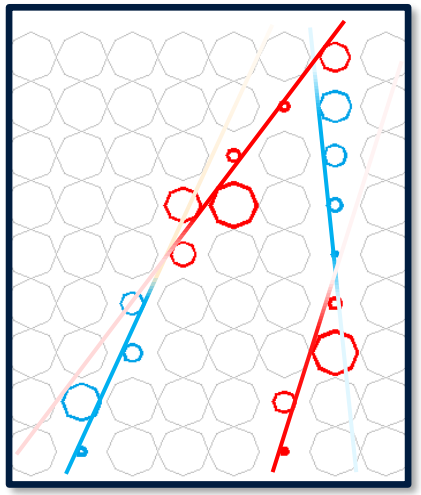
- Creates "***precluster solutions"*** = unique collection of fits explaining the precluster
- Rest of the process done separately for each *precluster solution*
- ***LinearFits*** → ***Tracks*** (*LinearFit* + tracker hit association points)



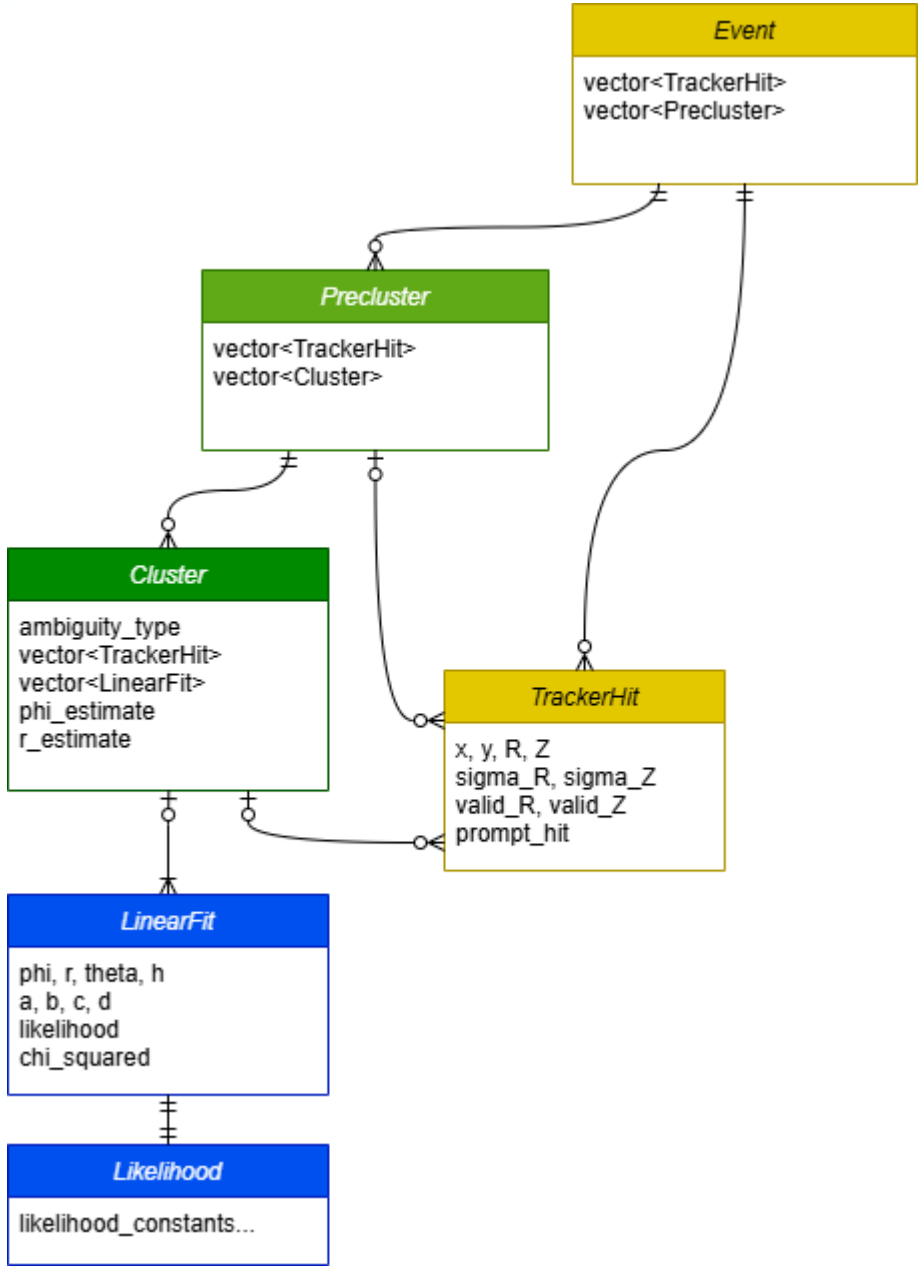
4 clusters, 5 fits

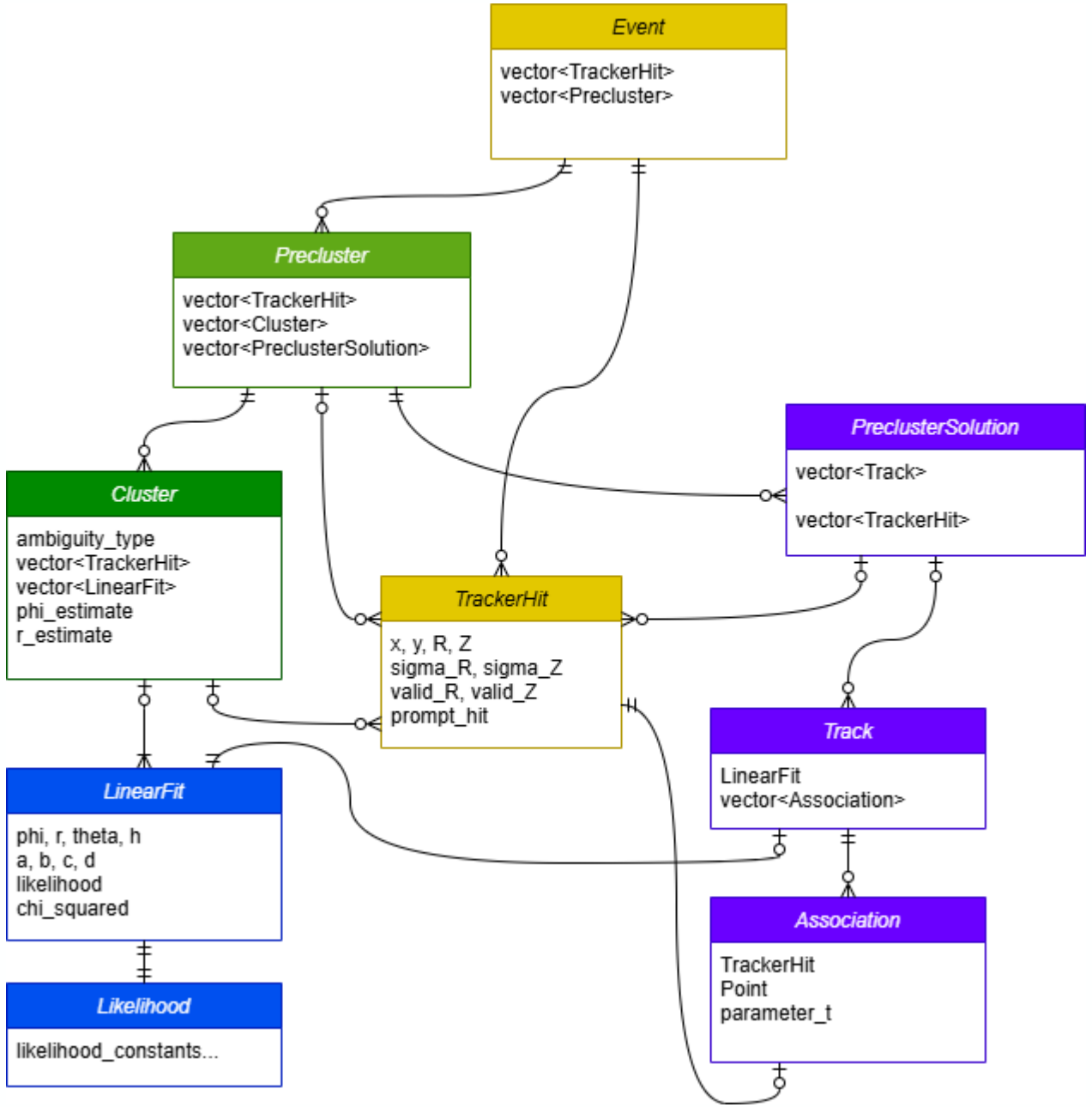


Precluster solution 1



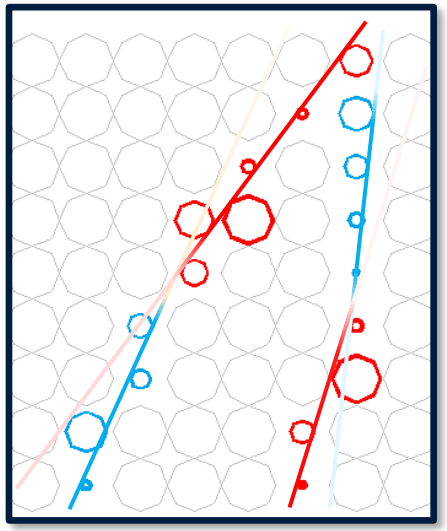
Precluster solution 2


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Likelihood
likelihood_constants...

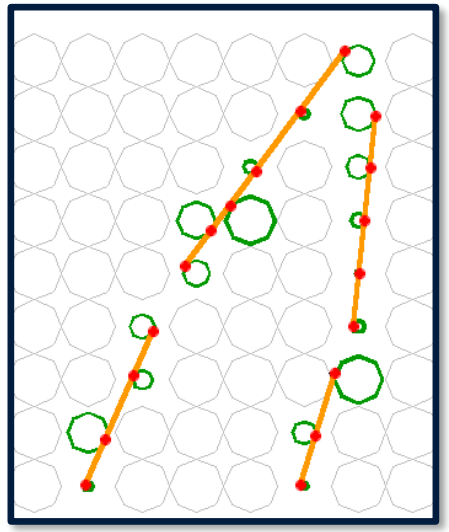

Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...

Step 5: Line trajectories

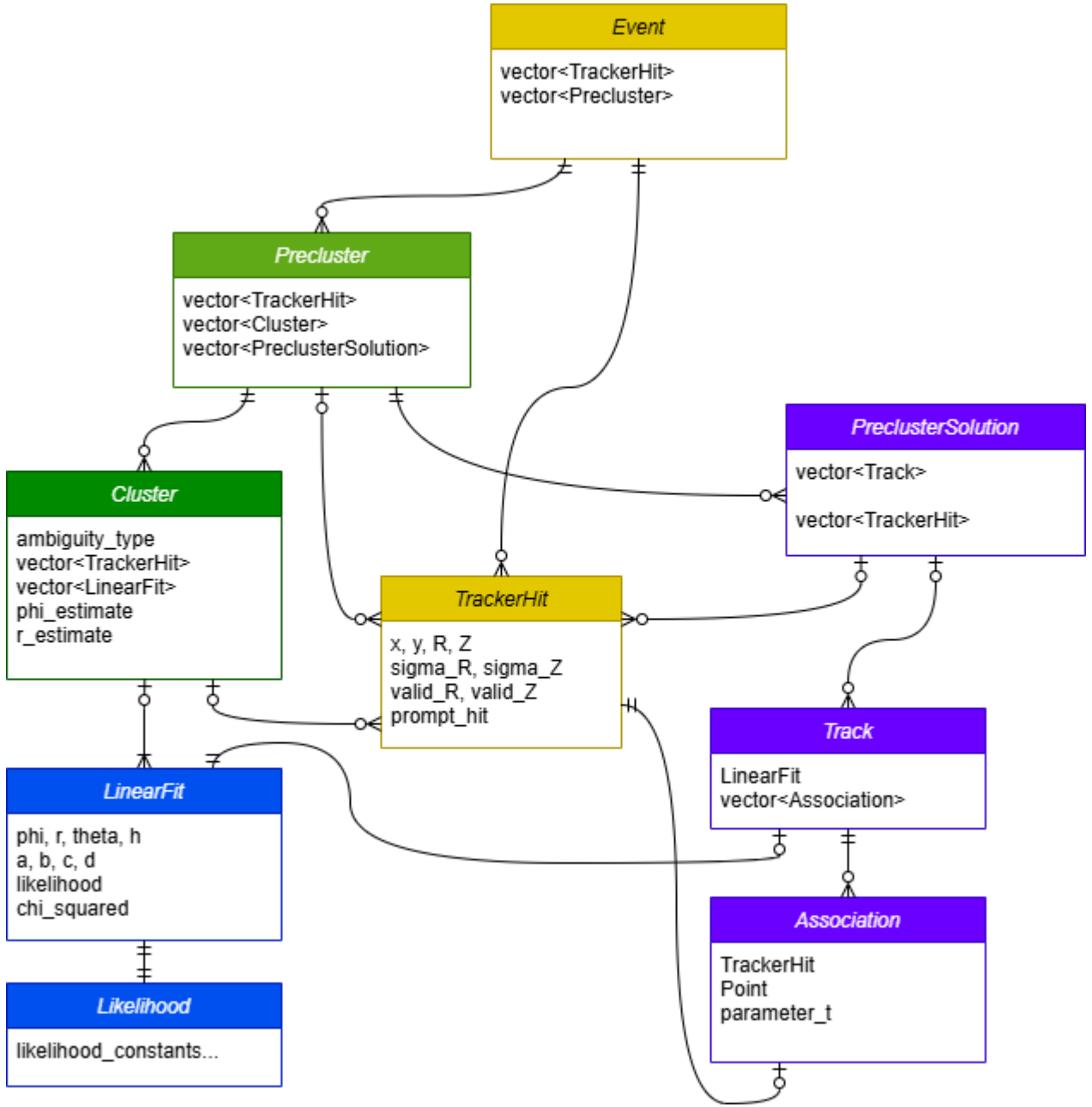
- ***Track*** → ***Trajectory*** (*Tracks* + endpoints)
- **Endpoints** = furthest association points

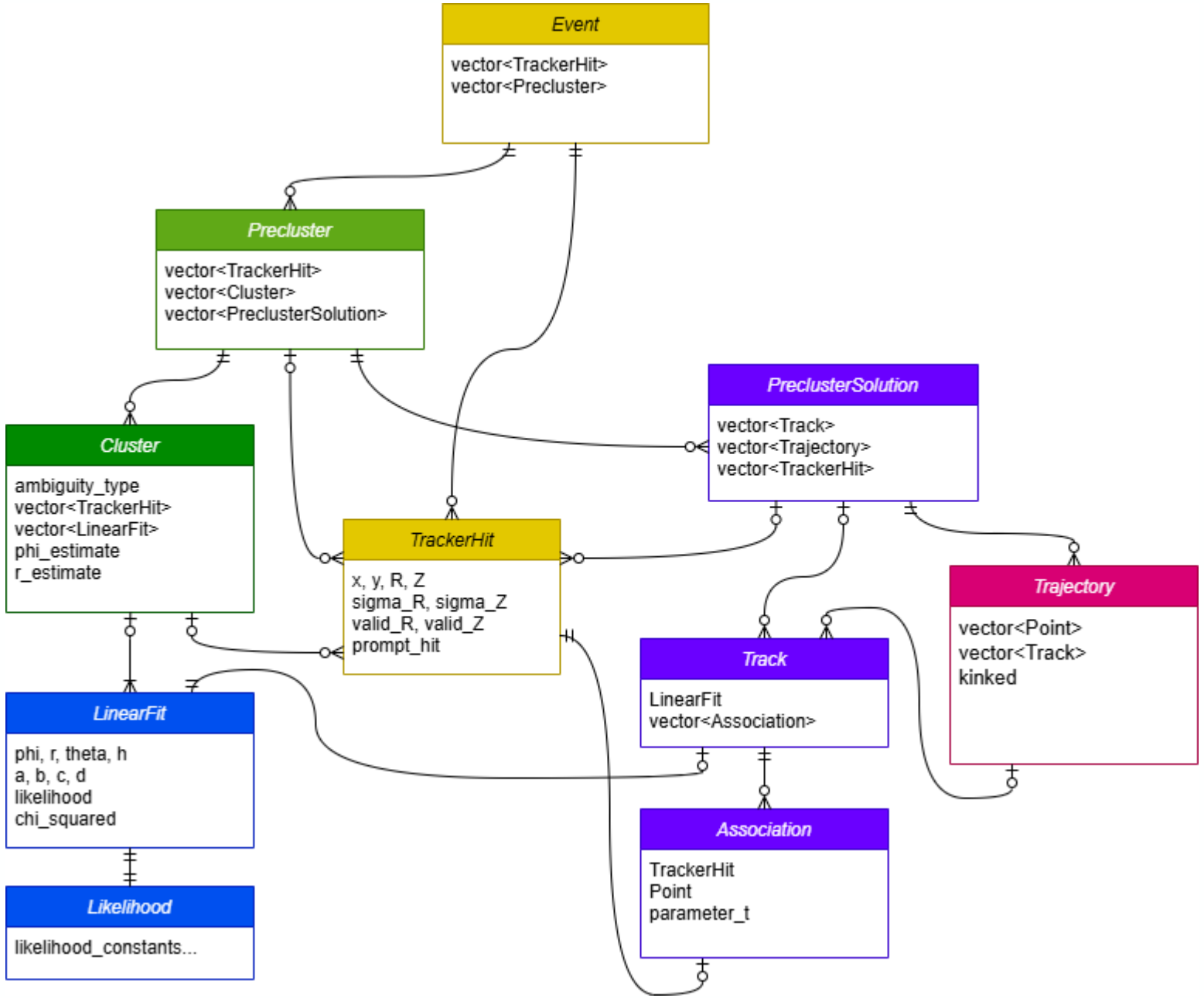


Precluster solution 1



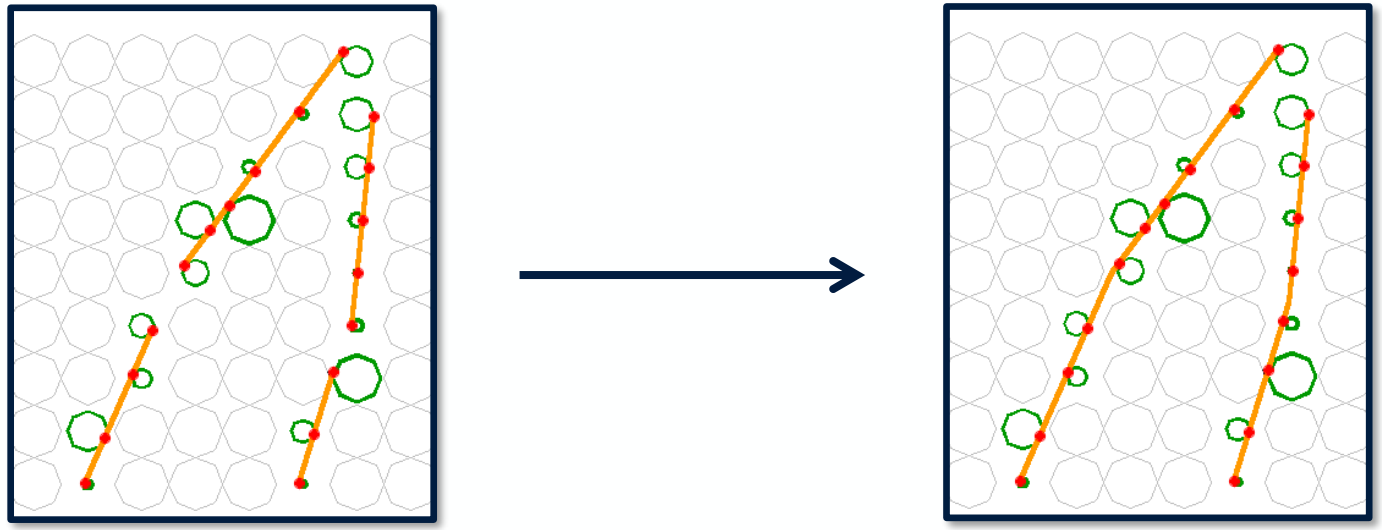
Precluster solution 1


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<Trajectory>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Trajectory
vector<Point>
vector<Track>
kinked
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...

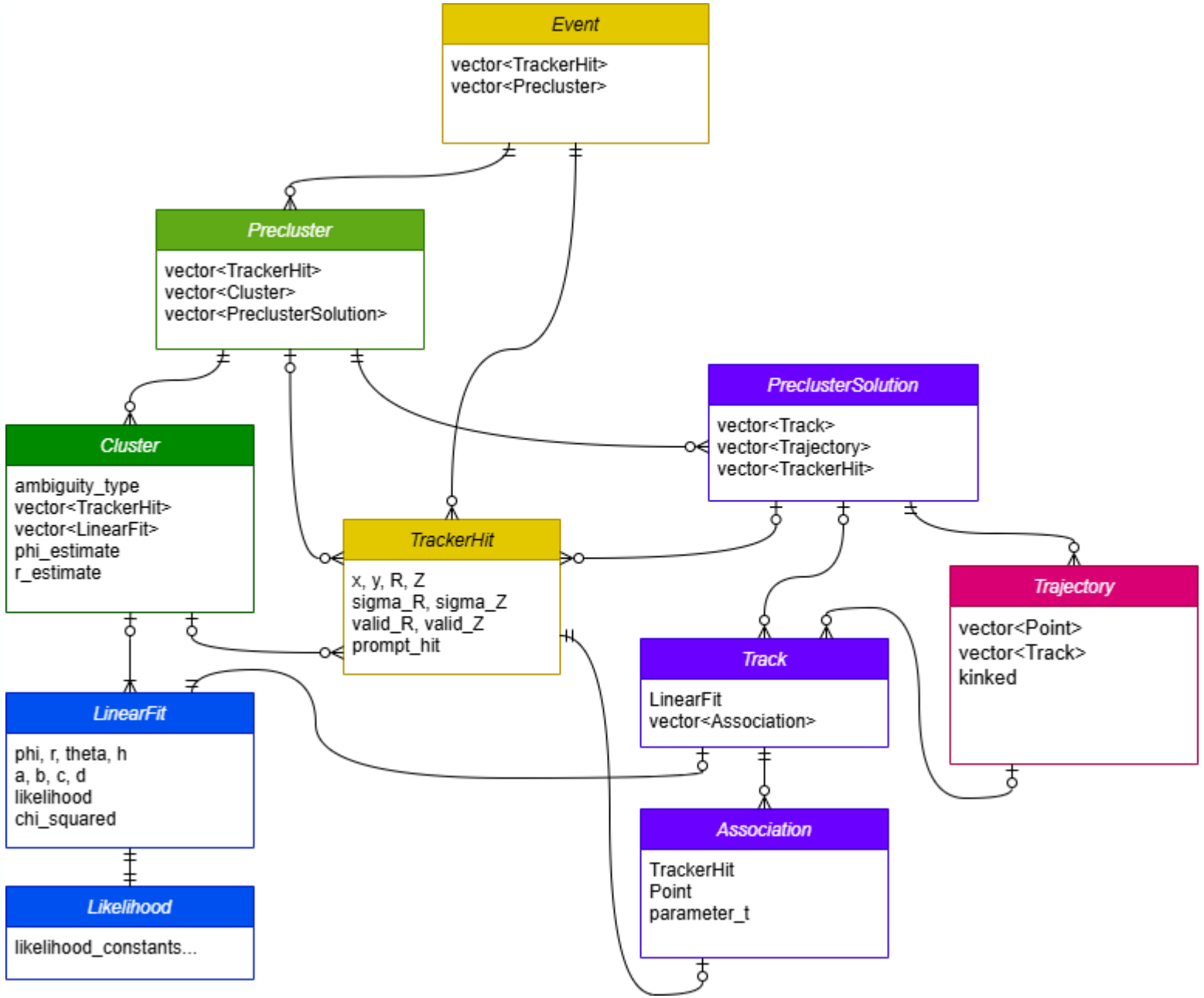
Step 6: Building polylines

- Line ***Trajectories*** → (poly)line ***Trajectories***
- **Merging trajectories together**
- More details in the second part of the presentation



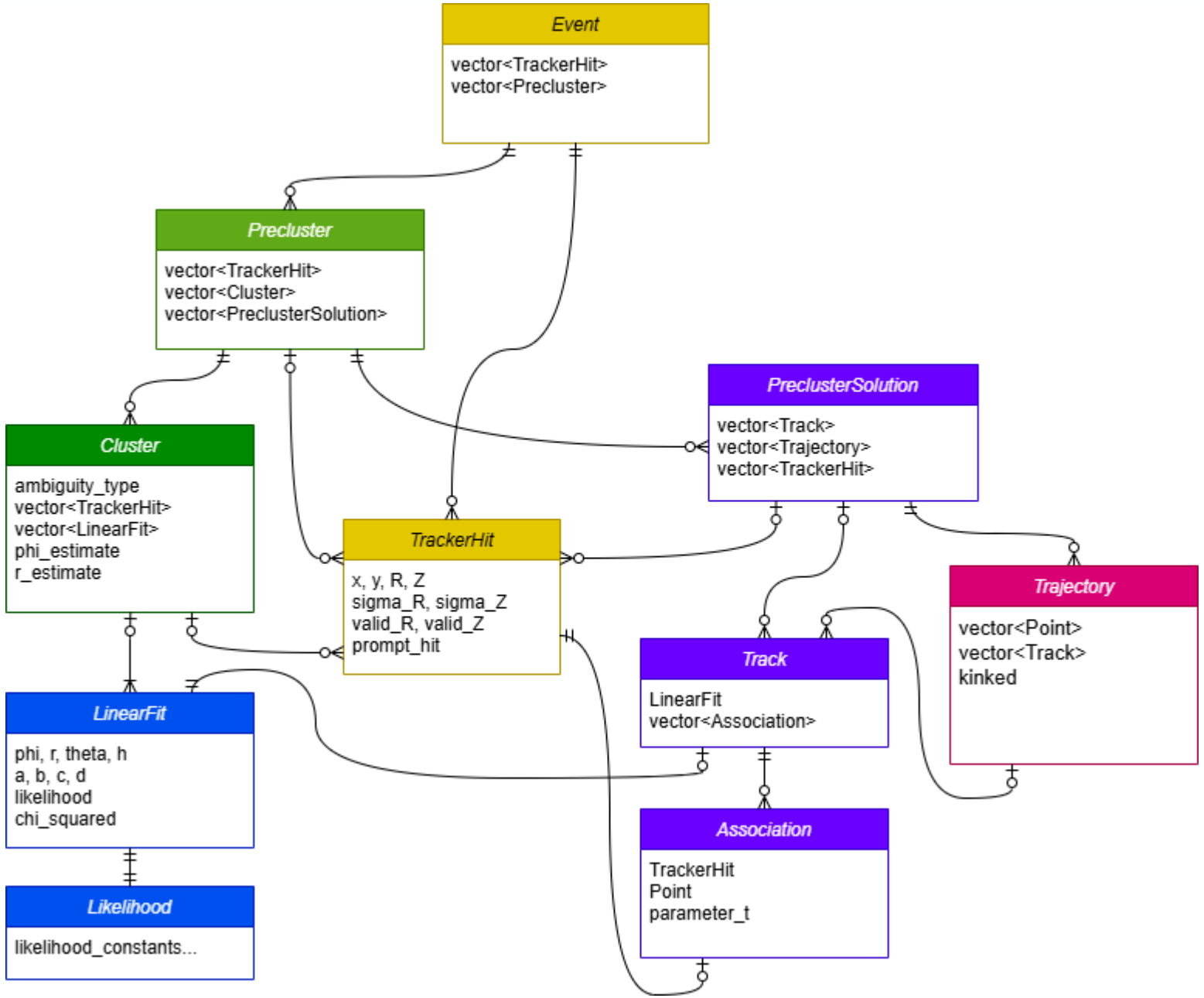
Precluster solution 1

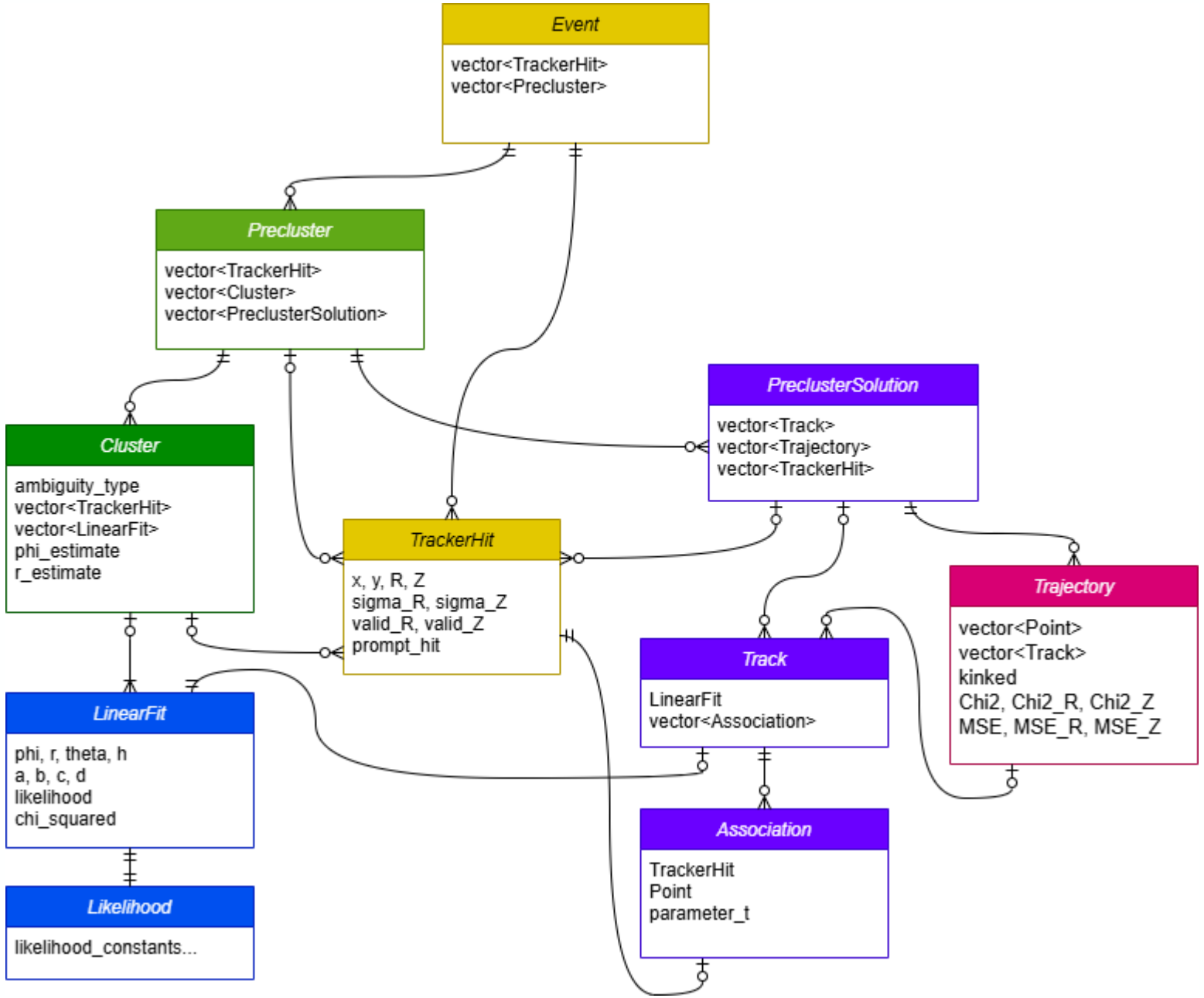
Precluster solution 1


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<Trajectory>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Trajectory
vector<Point>
vector<Track>
kinked
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...

Step 7: Trajectory evaluation

- Total χ^2 of trajectories
  - requires tracker hit uncertainties
- RMSE of trajectories (root mean square error)
  - no uncertainties required


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<Trajectory>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Trajectory
vector<Point>
vector<Track>
kinked
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<Trajectory>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Trajectory
vector<Point>
vector<Track>
kinked
Chi2, Chi2_R, Chi2_Z
MSE, MSE_R, MSE_Z
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...

Step 8: Merging into solutions

- Combining *precluster solutions* into full event ***Solutions***

Precluster A:

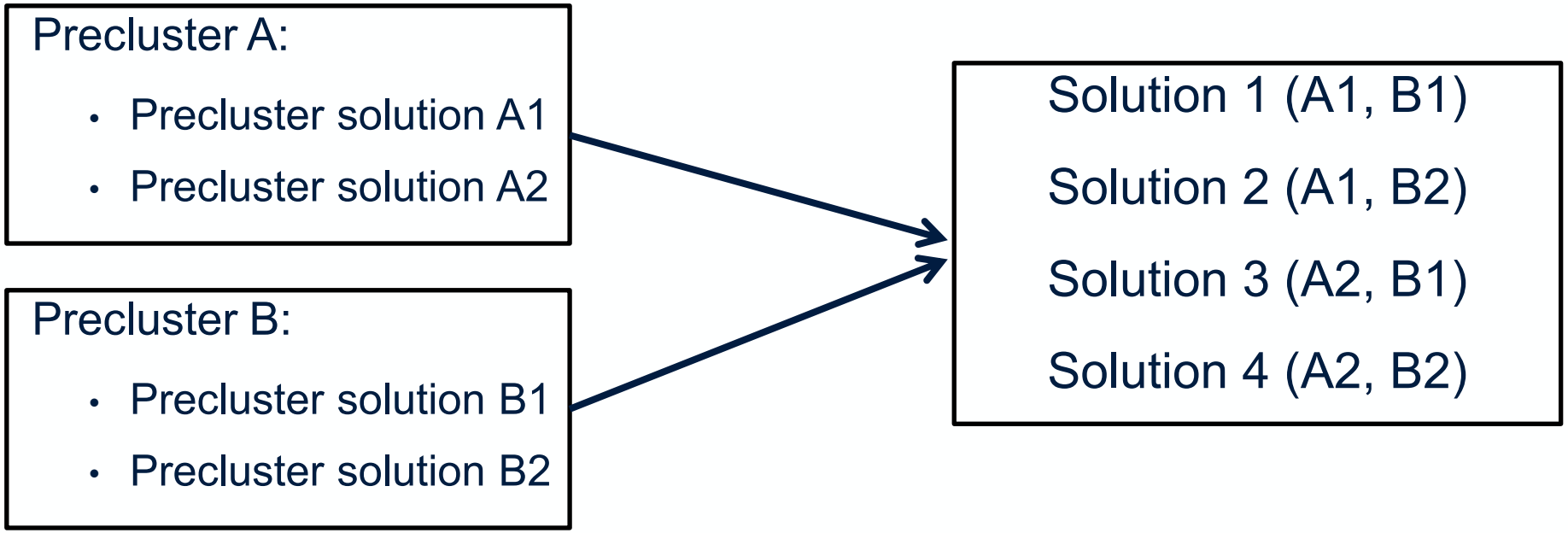
- Precluster solution A1
- Precluster solution A2

Precluster B:

- Precluster solution B1
- Precluster solution B2

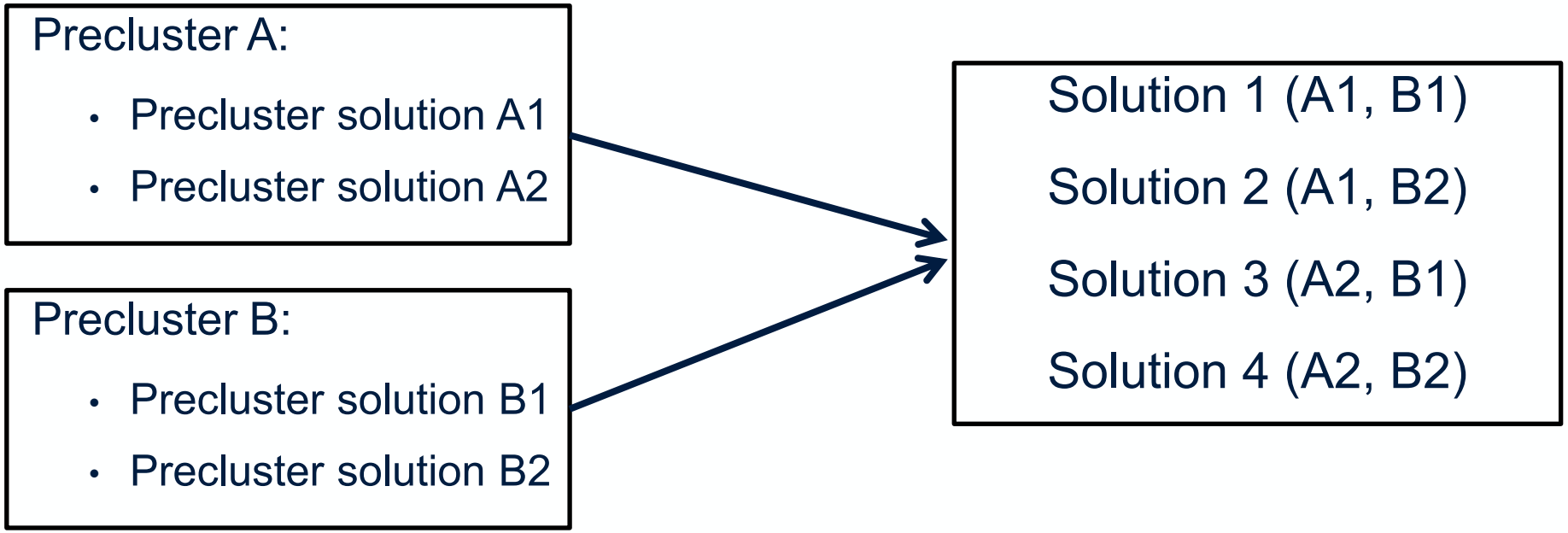
Step 8: Merging into solutions

- Combining *precluster solutions* into full event ***Solutions***

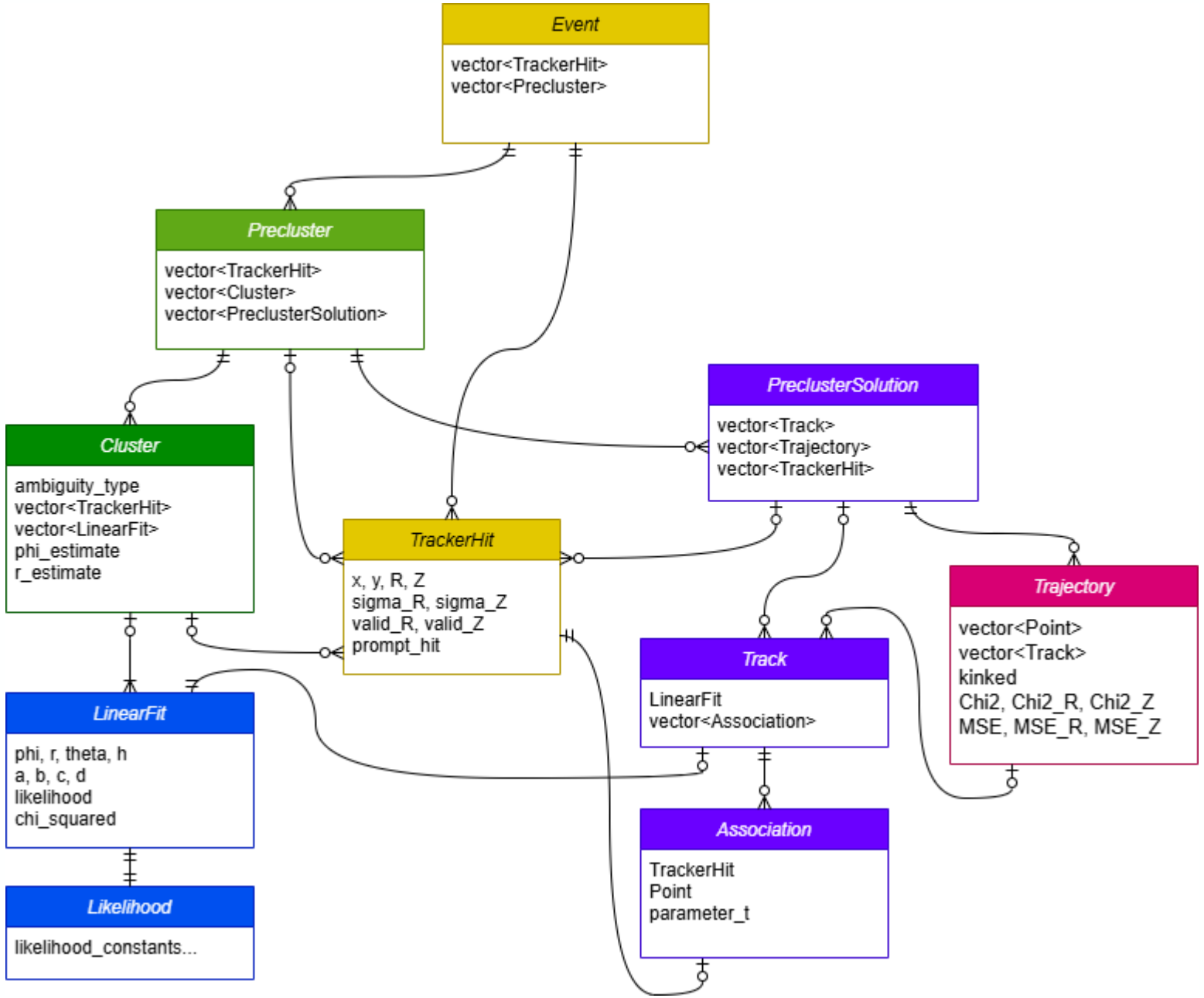


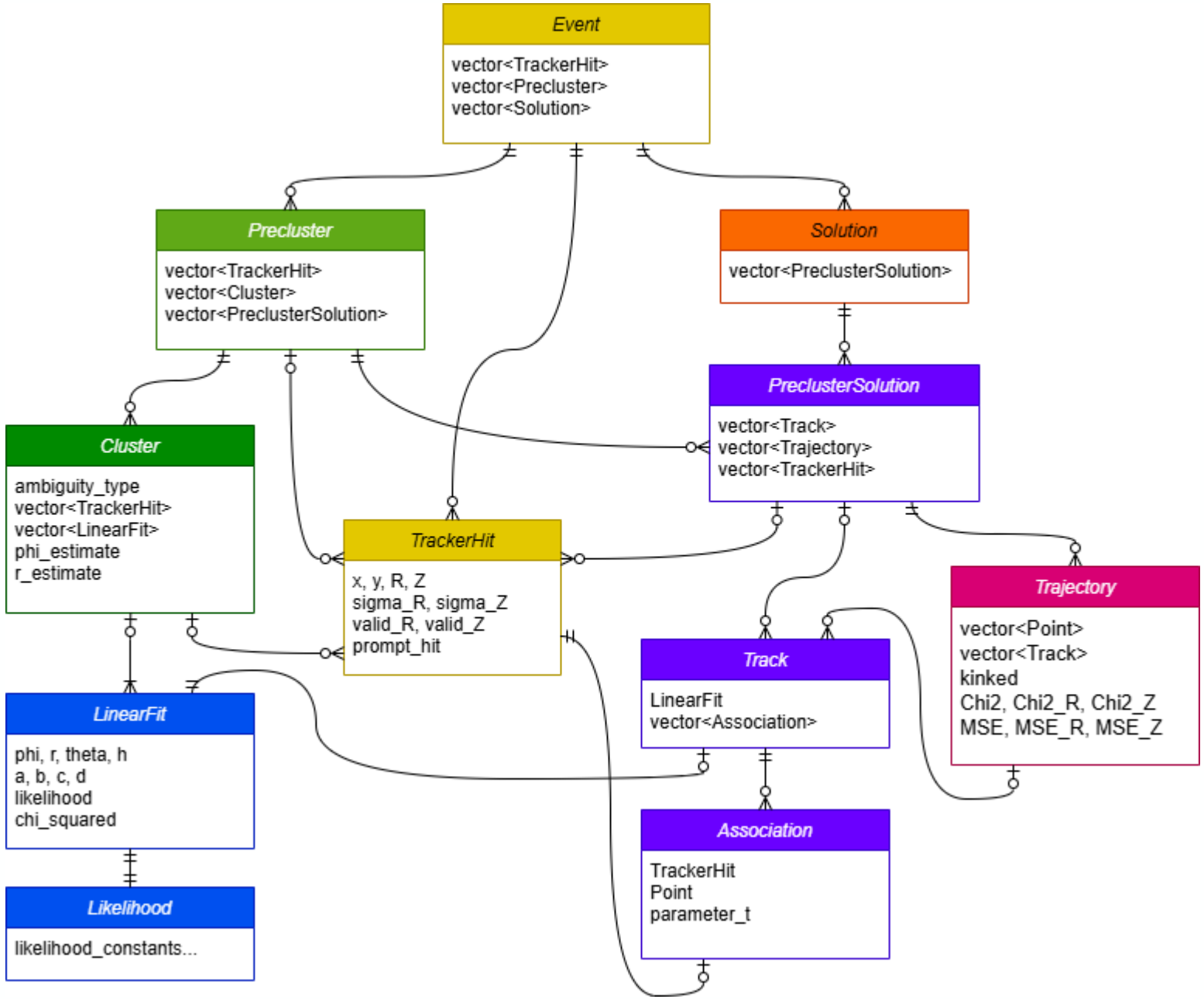
Step 8: Merging into solutions

- Combining *precluster solutions* into full event ***Solutions***



- Ordering solutions (# linear fits, # trajectories, # total χ^2)
- Optionally removing the worst solutions


Event
vector<TrackerHit>
vector<Precluster>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<Trajectory>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Trajectory
vector<Point>
vector<Track>
kinked
Chi2, Chi2_R, Chi2_Z
MSE, MSE_R, MSE_Z
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...

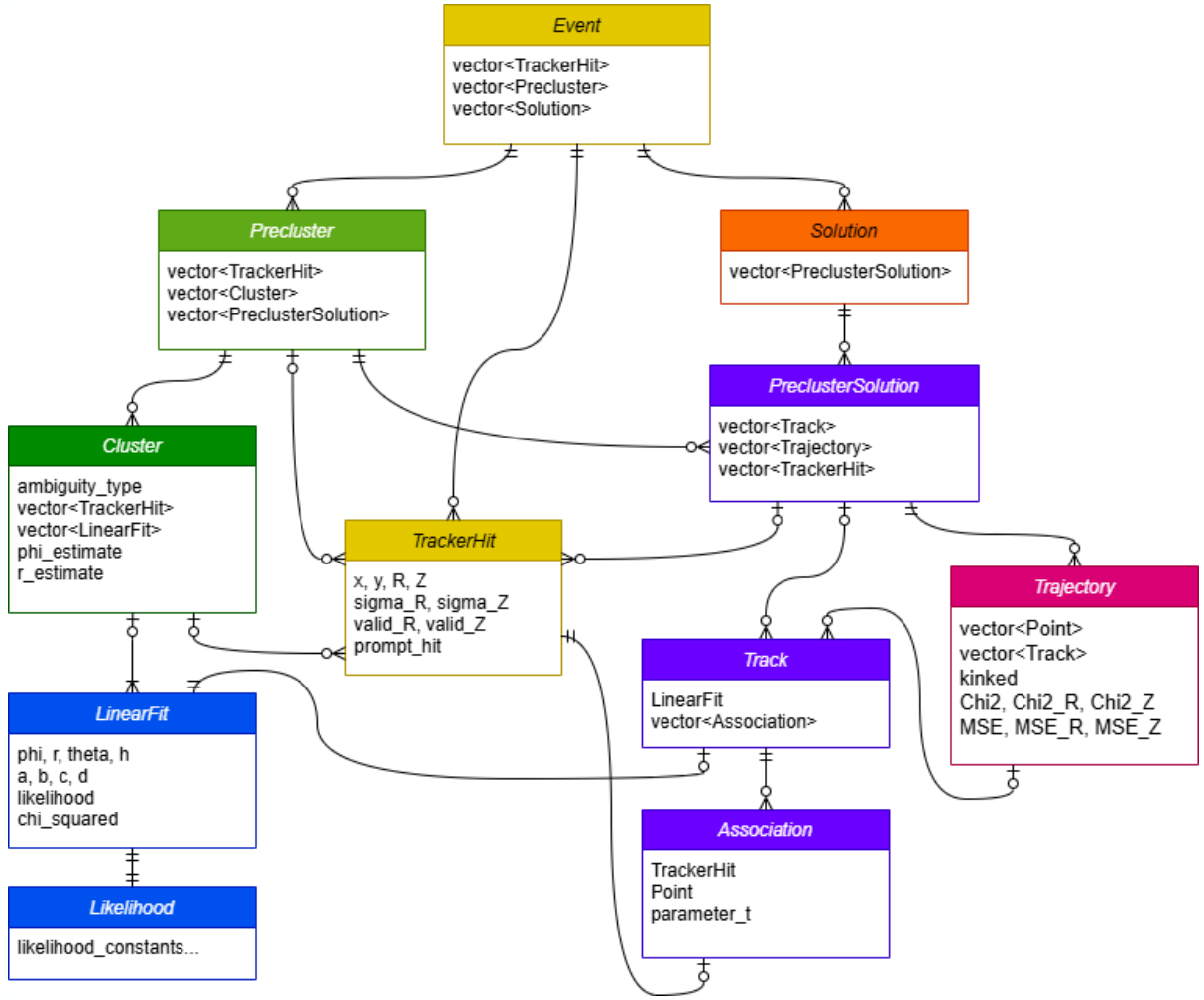

Event
vector<TrackerHit>
vector<Precluster>
vector<Solution>
Precluster
vector<TrackerHit>
vector<Cluster>
vector<PreclusterSolution>
Solution
vector<PreclusterSolution>
PreclusterSolution
vector<Track>
vector<Trajectory>
vector<TrackerHit>
Cluster
ambiguity_type
vector<TrackerHit>
vector<LinearFit>
phi_estimate
r_estimate
TrackerHit
x, y, R, Z
sigma_R, sigma_Z
valid_R, valid_Z
prompt_hit
Trajectory
vector<Point>
vector<Track>
kinked
Chi2, Chi2_R, Chi2_Z
MSE, MSE_R, MSE_Z
Track
LinearFit
vector<Association>
LinearFit
phi, r, theta, h
a, b, c, d
likelihood
chi_squared
Association
TrackerHit
Point
parameter_t
Likelihood
likelihood_constants...

Algorithm summary

```
Algos::process(Event&) {
    precluster();

    for(precluster) {
        clustering();
        make_MLM_fits();
        combine_into_precluster_solutions();

        for(precluster_solution) {
            create_line_trajectories();
            build_polylines();
            evaluate_trajectories();
        }
    }
    create_solutions();
}
```



supernemo collaboration

2. Polyline reconstruction framework

Polyline reconstruction framework

Cimrman 1.1.0 (current official version):

- Chaotic unmanageable implementation

Polyline reconstruction framework

Cimrman 1.1.0 (current official version):

- Chaotic unmanageable implementation

Cimrman 1.2.0? (development version):

- Functionally (almost) the same
- Clean structured implementation

Polyline reconstruction framework

Cimrman 1.1.0 (current official version):

- Chaotic unmanageable implementation

Cimrman 1.2.0? (development version):

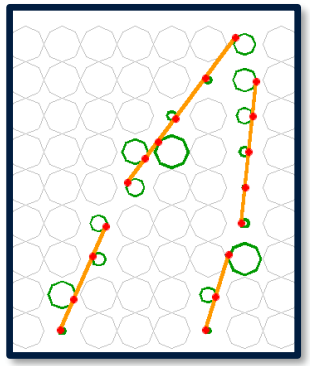
- Functionally (almost) the same
- Clean structured implementation
- Dedicated worker class ***TrajectoryBuilder***
- Could be reused for connecting tracks outside tracker volume (crossing tracks, BiPo events, double beta events…)

TrajectoryBuilder

- Dedicated class for connecting trajectories
- Modifies one *precluster solution* at a time (no ambiguities involved)

TrajectoryBuilder

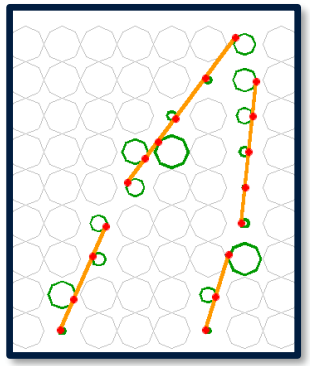
- Dedicated class for connecting trajectories
- Modifies one *precluster solution* at a time (no ambiguities involved)
- Input: collection of (line) trajectories (with associated hits)



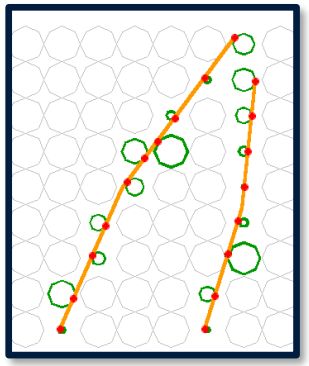
(T_1, T_2, T_3, T_4)

TrajectoryBuilder

- Dedicated class for connecting trajectories
- Modifies one *precluster solution* at a time (no ambiguities involved)
- Input: collection of (line) trajectories (with associated hits)
- Output: collection of (polyline) trajectories



$$(T_1, T_2, T_3, T_4) \rightarrow (T_1, T_2, T_3) \rightarrow (T_1, T_2)$$



TrajectoryBuilder

Process():

- For each pair of trajectories (T_i, T_j):

TrajectoryBuilder

Process():

- For each pair of trajectories (T_i, T_j):
  - For each pair of their endpoints (E_k, E_l):

TrajectoryBuilder

Process():

- For each pair of trajectories (T_i, T_j):
  - For each pair of their endpoints (E_k, E_l):
    - Can T_i and T_j be connected? (*KinkFinder*)
    - If yes, merge T_j into T_i

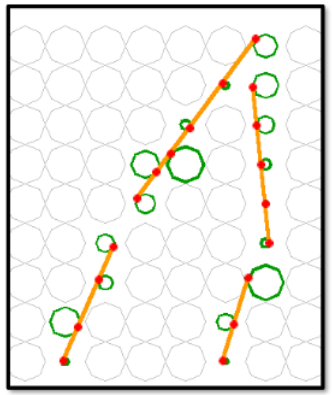
TrajectoryBuilder

Process():

- For each pair of trajectories (T_i, T_j):
  - For each pair of their endpoints (E_k, E_l):
    - Can T_i and T_j be connected? (*KinkFinder*)
    - If yes, merge T_j into T_i

4 line trajectories = 6 pairs

$\rightarrow$ 6 x 4 = 24 endpoint pairs



Solution 1

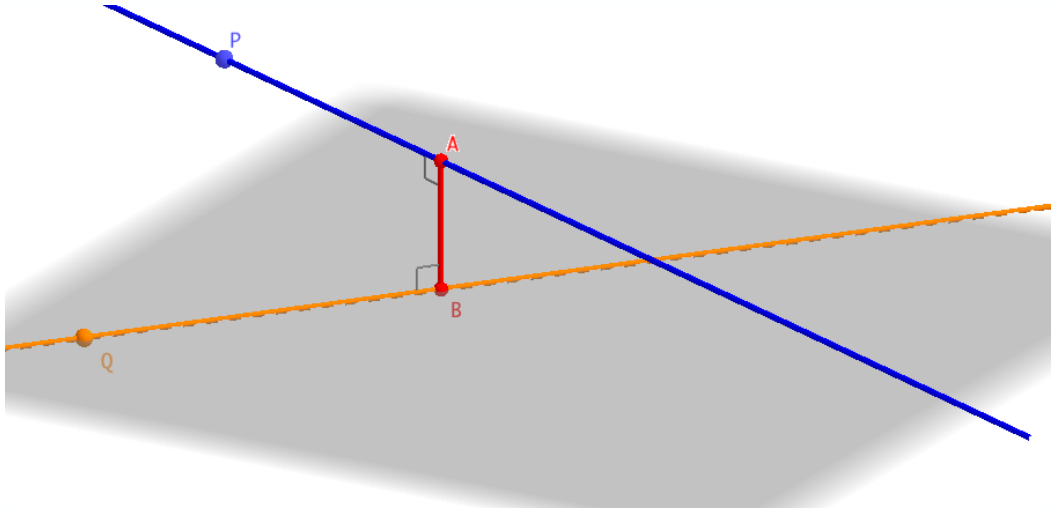
KinkFinder

- Dedicated class for connection decisions
- **Can two trajectories be connected? If yes, how?**

KinkFinder

- Dedicated class for connection decisions
- **Can two trajectories be connected? If yes, how?**
- Two lines in 3D have (almost certainly) no intersection!

  → Fits must be changed to enforce continuity



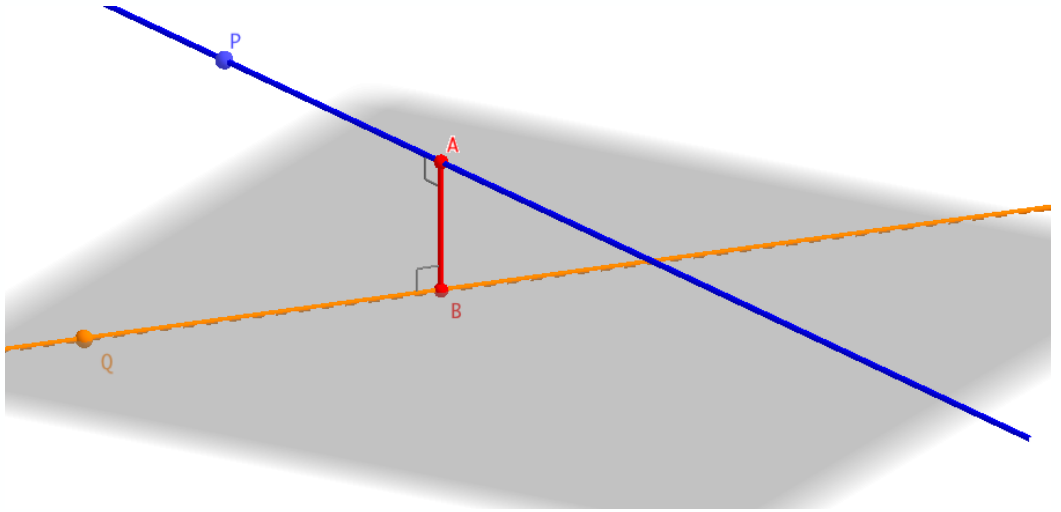
KinkFinder

- Dedicated class for connection decisions
- **Can two trajectories be connected? If yes, how?**
- Two lines in 3D have (almost certainly) no intersection!

  → Fits must be changed to enforce continuity
- Two ingredients:

  1. Kink point construction strategy

  - How do you construct the point?



KinkFinder

- Dedicated class for connection decisions
- **Can two trajectories be connected? If yes, how?**
- Two lines in 3D have (almost certainly) no intersection!

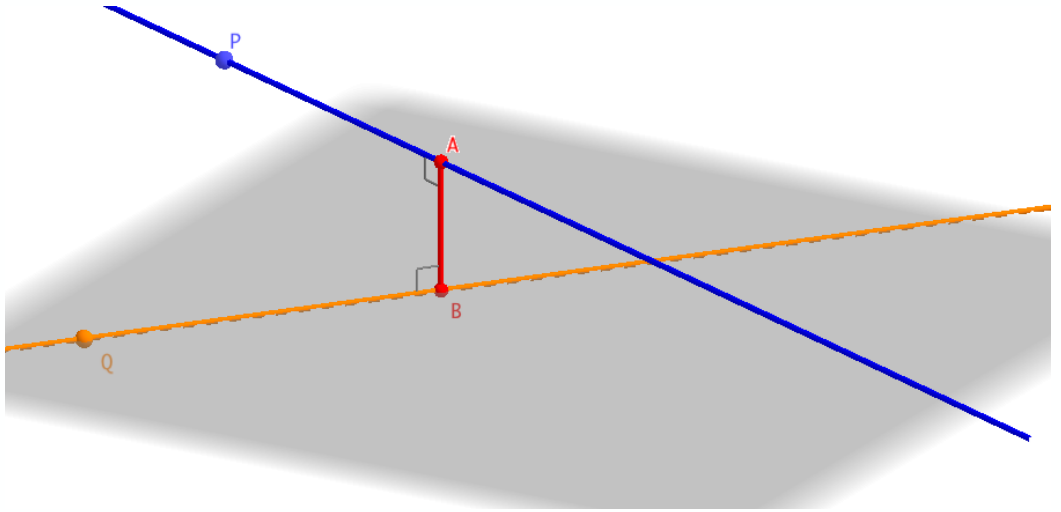
  → Fits must be changed to enforce continuity
- Two ingredients:

  1. Kink point construction strategy

  - How do you construct the point?

  2. List of validation criteria

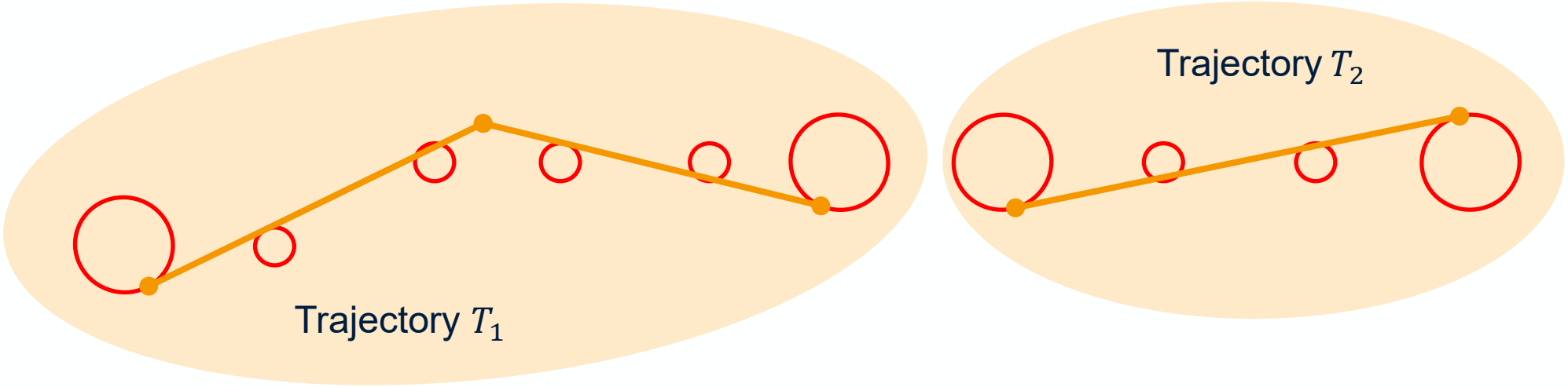
  - How do I check that it is valid?



KinkFinder

Inputs:

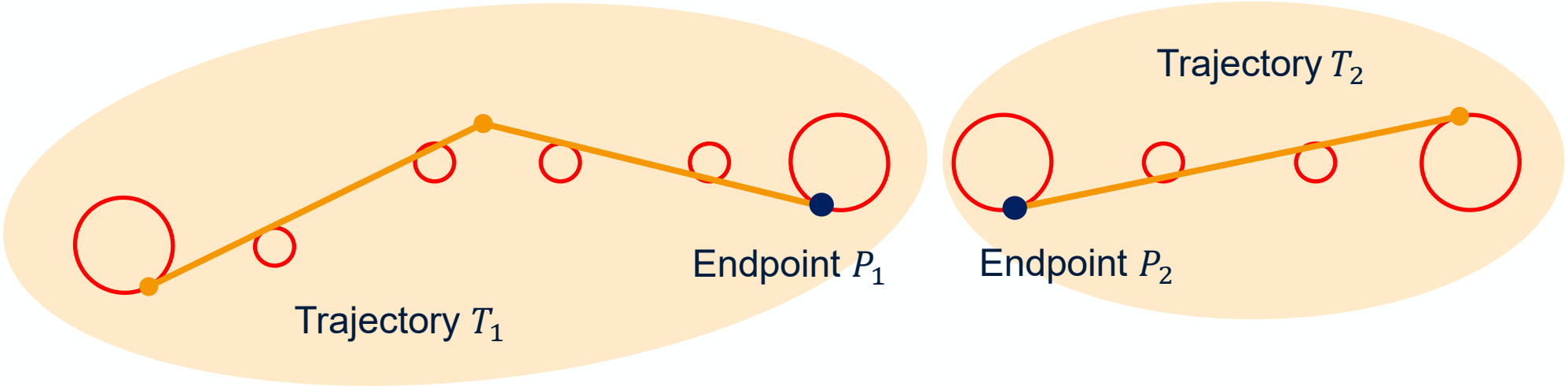
- Trajectories T_1, T_2



KinkFinder

Inputs:

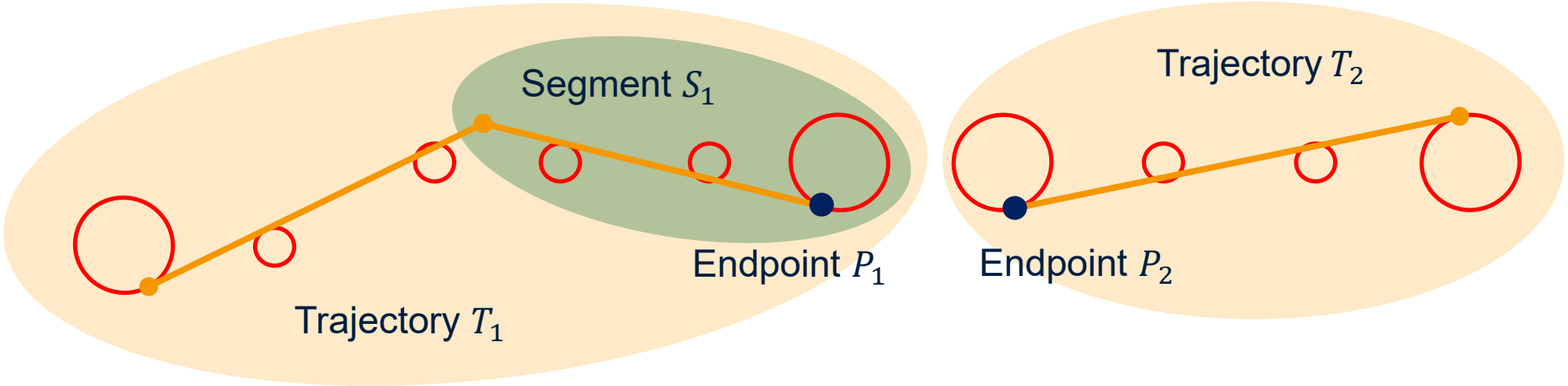
- Trajectories T_1, T_2
- Selected endpoints P_1, P_2



KinkFinder

Inputs:

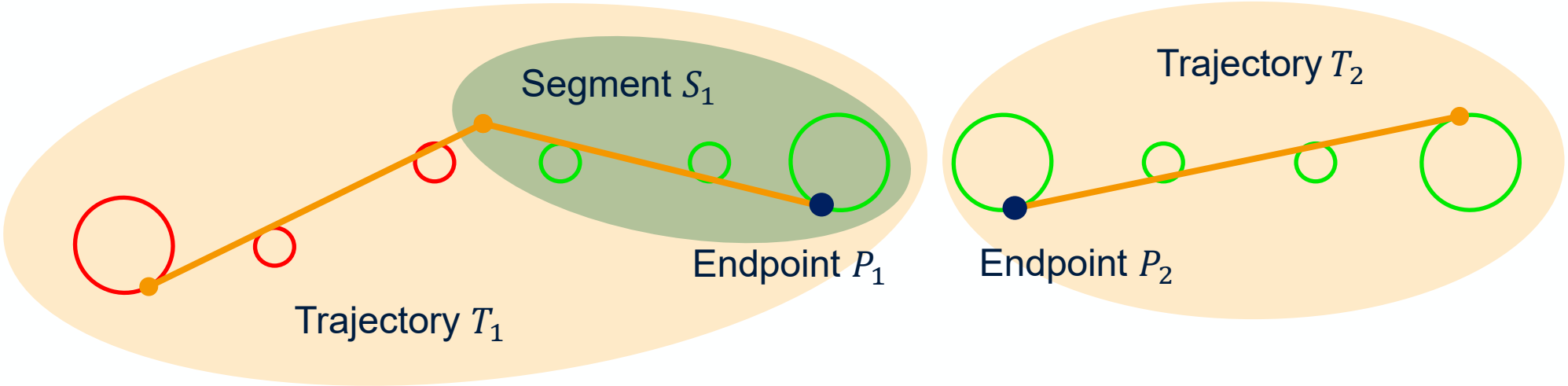
- Trajectories T_1, T_2
- Selected endpoints P_1, P_2
- Selected segments of the (poly)lines



KinkFinder

Inputs:

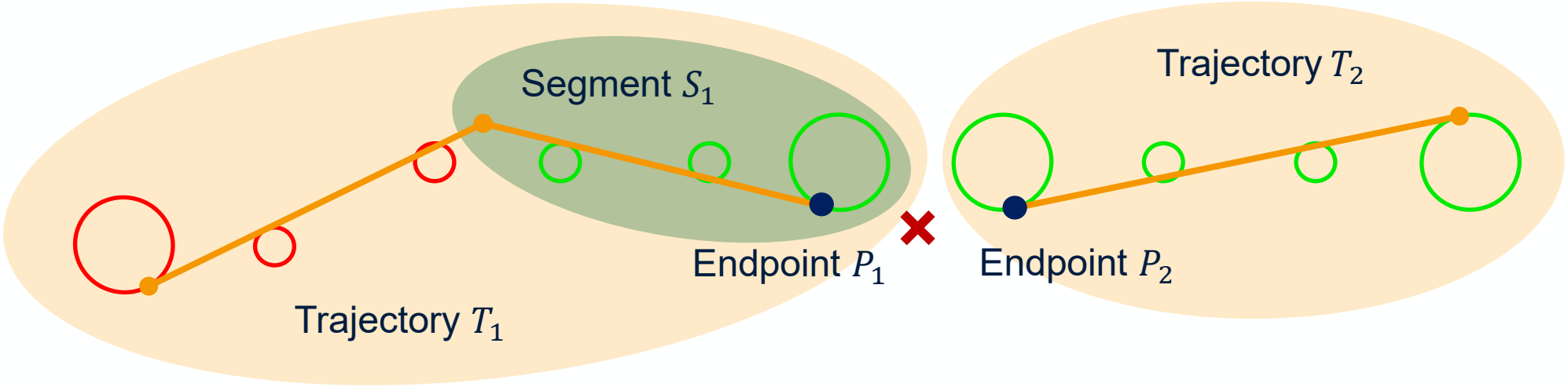
- Trajectories T_1, T_2
- Selected endpoints P_1, P_2
- Selected segments of the (poly)lines
- Associated hits



KinkFinder

Outputs:

- Yes/No connection decision
- kink point position



Connection strategies

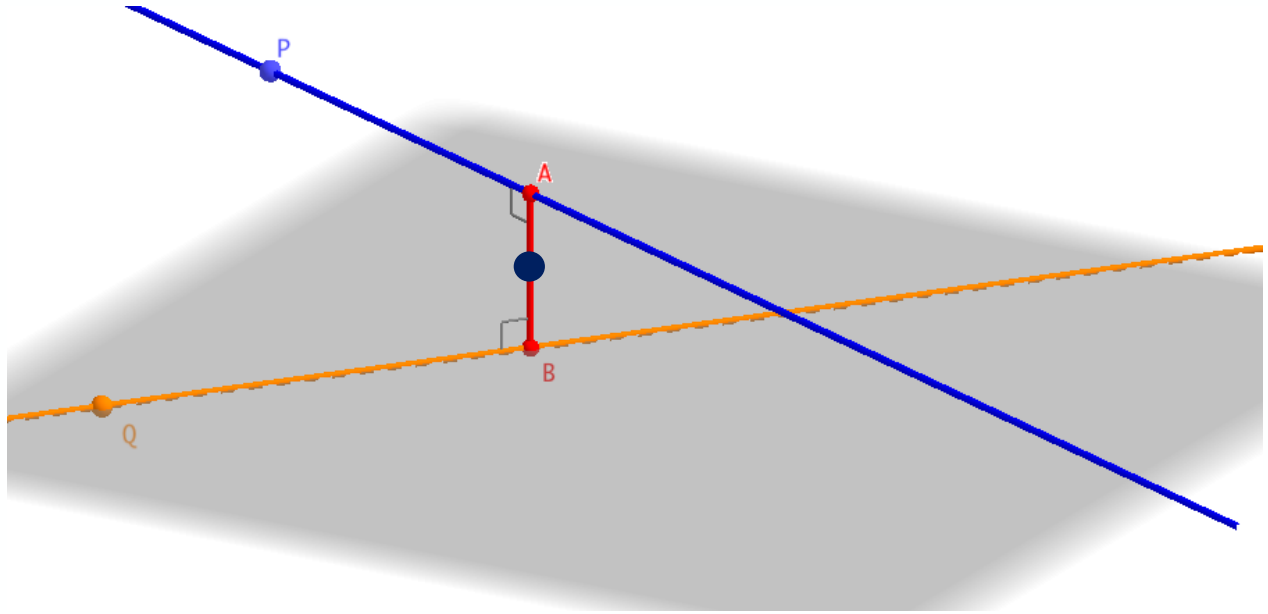
1. How to construct kink points?

Strategy 1: “Vertical alignment”

- **We enforce continuity by changing only the vertical parts**
- $\boldsymbol{\sigma_R \ll \sigma_Z}$ → horizontal parts of the fits are more trustworthy

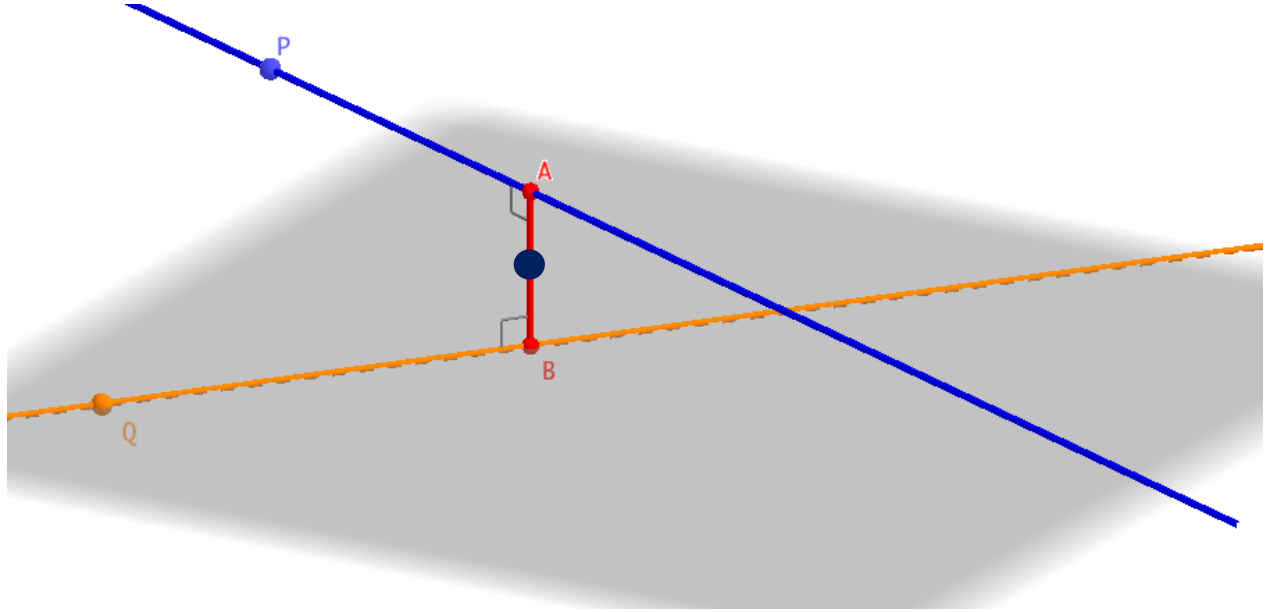
Strategy 1: “Vertical alignment”

- **We enforce continuity by changing only the vertical parts**
- $\boldsymbol{\sigma_R \ll \sigma_Z}$ → horizontal parts of the fits are more trustworthy
- Line trajectories T_1, T_2 have intersection (x_0, y_0) in 2D



Strategy 1: “Vertical alignment”

- **We enforce continuity by changing only the vertical parts**
- $\boldsymbol{\sigma_R \ll \sigma_Z}$ → horizontal parts of the fits are more trustworthy
- Line trajectories T_1, T_2 have intersection (x_0, y_0) in 2D
- Z coordinates of T_1 and T_2 in (x_0, y_0) are z_1 and z_2
- Kink point = $(x_0, y_0, (z_1 + z_2)/2)$



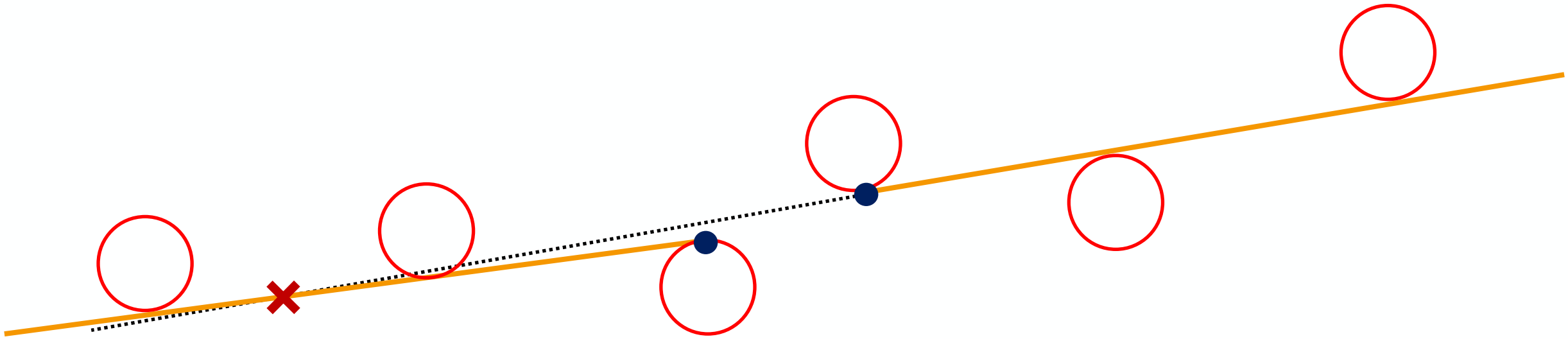
Strategy 1: “Vertical alignment”

Pros: easy to calculate, works most of the time

Strategy 1: “Vertical alignment”

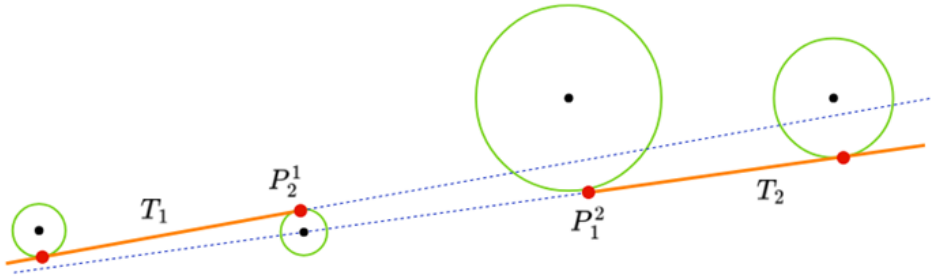
Pros: easy to calculate, works most of the time

Cons: unstable for smaller angles!

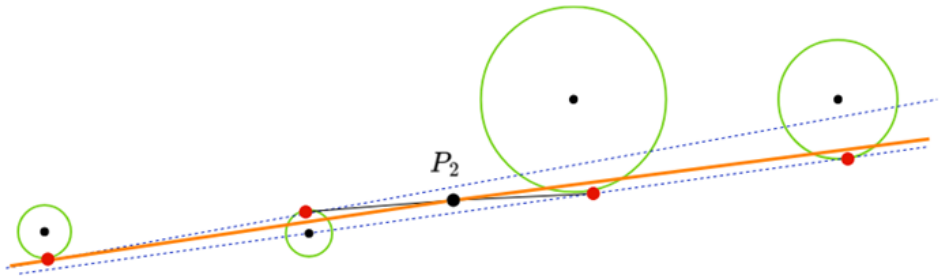


Strategy 2: “Endpoints middle”

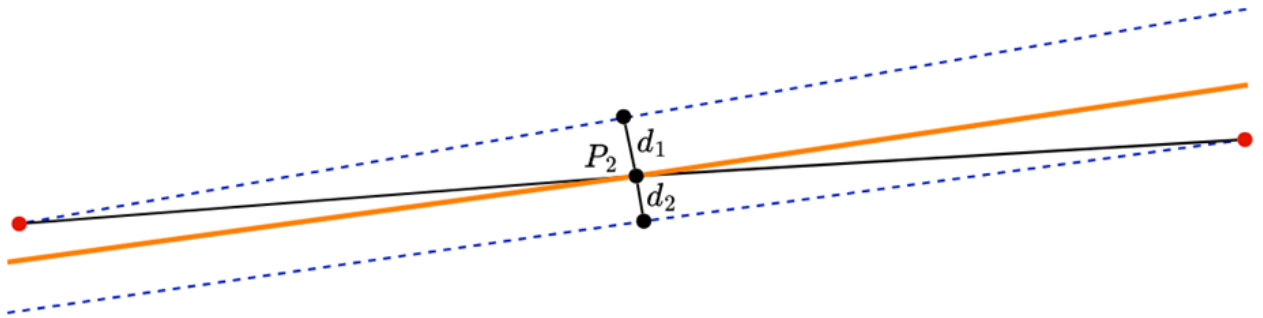
- Made as a correction for small angles
- Kink Point = middle point between the two ends



(a) Disconnected trajectories



(b) Connected trajectories



(c) Distances of new trajectory point to the original tracks

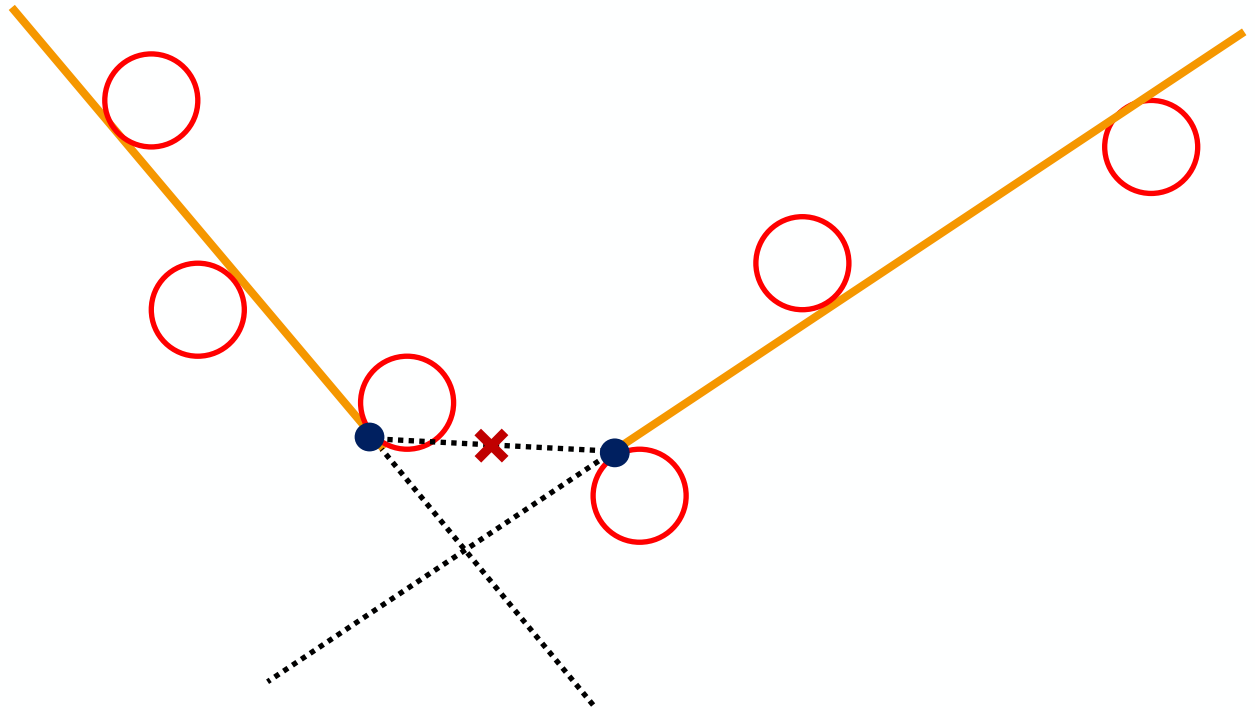
Strategy 2: “Endpoints middle”

Pros: great for small angles

Strategy 2: “Endpoints middle”

Pros: great for small angles

Cons: terrible for larger angles



Strategy 3: “closest approach”

- Point of closest approach between the two lines
- No obvious benefits
- Might be superior to “vertical alignment” strategy (needs testing)

Strategy 3: “closest approach”

- Point of closest approach between the two lines
- No obvious benefits
- Might be superior to “vertical alignment” strategy (needs testing)

Pros: similar as “vertical alignment”

Cons: unstable for small angles

More possible strategies

- "*Point on wire*" strategy for Bi-Po events?
- "*Point on foil*" strategy for $\beta\beta$ events?

Validation criteria

2. How do we check validity?

Validation criteria

- Each strategy has its own set
- Modular reusable implementation

```
// angular criteria
bool check_angle_2D_after( const double min_angle, const double max_angle ) const;
bool check_angle_3D_after( const double min_angle, const double max_angle ) const;
bool check_angle_2D_before( const double min_angle, const double max_angle ) const;
bool check_angle_3D_before( const double min_angle, const double max_angle ) const;

// position criteria
bool check_is_inside_tracker() const;
bool check_distance_to_MW( const double min_distance ) const;
bool check_distance_to_XW( const double min_distance ) const;
bool check_distance_to_SF( const double min_distance ) const;

// associated tracker hit criteria
bool check_close_hits_2D( const double max_angle ) const;

// endpoints criteria
bool check_are_ends_close_2D( const double max_distance ) const;
bool check_are_ends_close_3D( const double max_distance ) const;

// distance criteria specific to the connection strategies
bool check_vertical_distance( const double max_distance) const;
bool check_middle_point_distance( const double max_distance) const;
```

Validation criteria

1. “Vertical alignment” criteria set:

```
// constructing a list of criteria
std::initializer_list<Criterion> criteria = {
  [&]{ return check_are_ends_close_2D( config.max_trajectory_endpoints_distance ); },
  [&]{ return check_close_hits_2D( config.max_tracker_hits_distance ); },
  [&]{ return check_angle_3D_after( config.min_kink_angle, config.max_kink_angle ); },
  [&]{ return check_angle_3D_before( config.min_kink_angle, config.max_kink_angle ); },
  [&]{ return check_kink_point_distance( config.max_vertical_distance / 2.0 ); },
  [&]{ return check_is_inside_tracker(); },
  [&]{ return check_distance_to_SF( config.min_distance_from_foil ); },
  [&]{ return check_distance_to_MW( config.min_distance_from_main_walls ); },
  [&]{ return check_distance_to_XW( config.min_distance_from_X_walls ); }
};
```

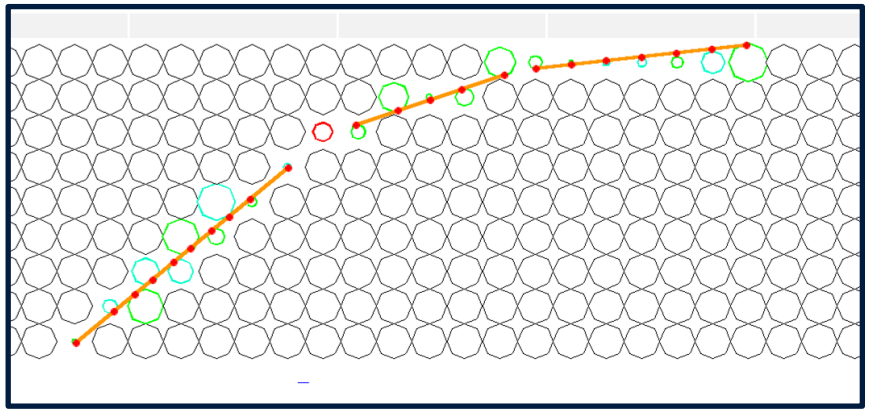
2. “Endpoints middle” criteria set:

```
// constructing a list of criteria
std::initializer_list<Criterion> criteria = {
  [&]{ return check_are_ends_close_3D( config.max_trajectory_endpoints_distance ); },
  [&]{ return check_close_hits_2D( config.max_tracker_hits_distance ); },
  [&]{ return check_middle_point_distance( config.max_trajectories_middlepoint_distance ); },
  [&]{ return check_angle_3D_after( config.min_trajectory_connection_angle, config.max_trajectory_connection_angle ); },
};
```

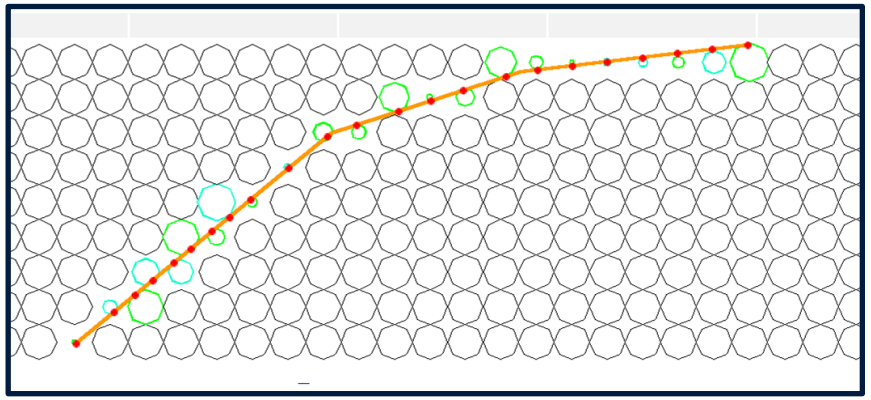
Refinements

We changed the fits. What broke?

Refinements – wrong associations

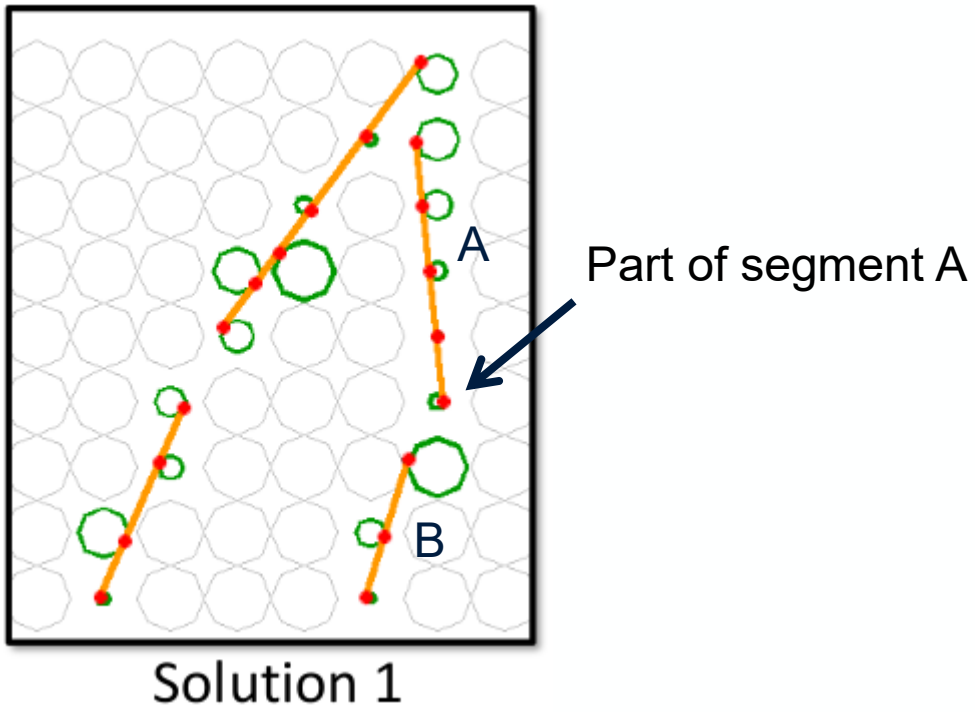


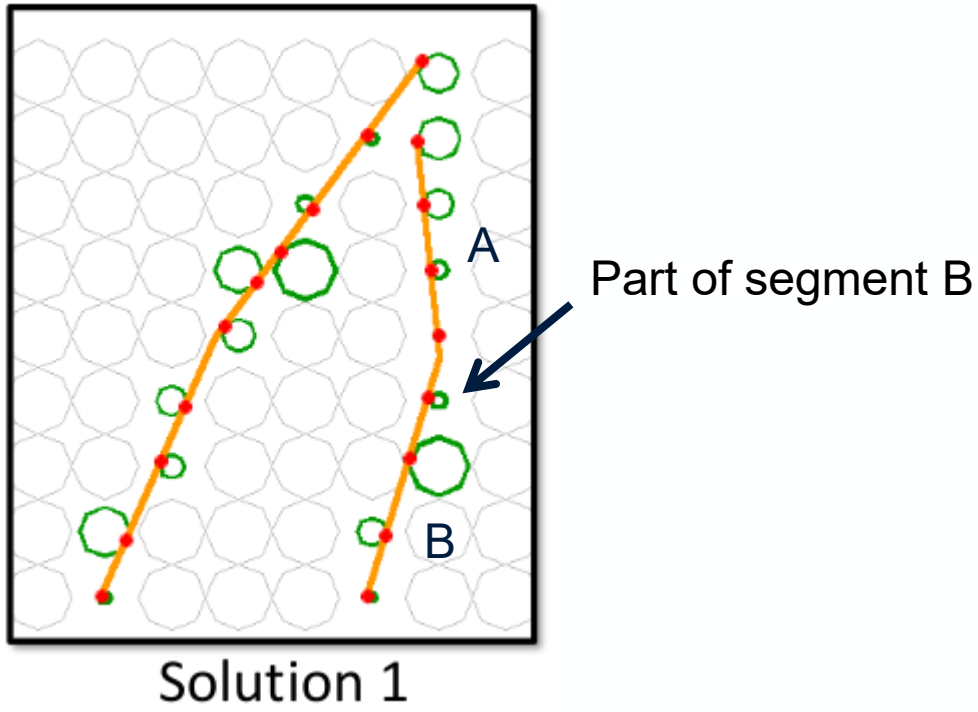
Missed completely



Can be added after connection (the fits changed)

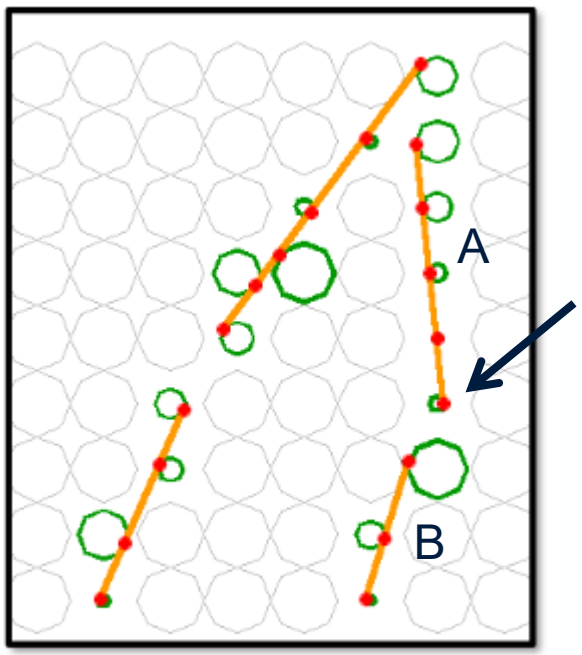
Refinements – wrong associations





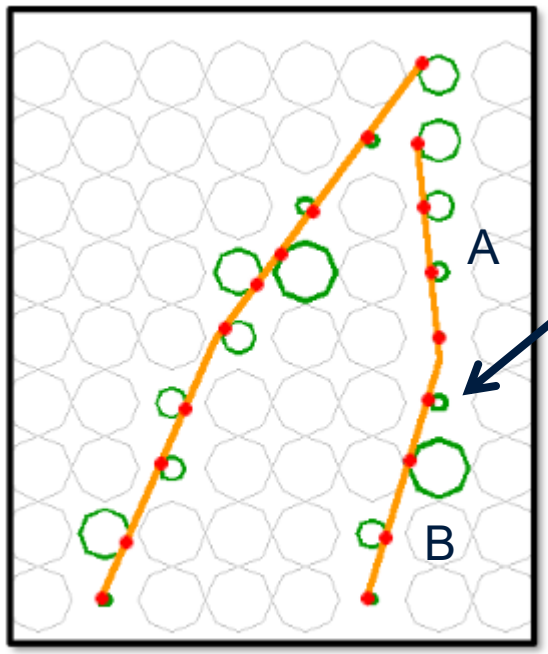
Refinements – wrong associations

- Reevaluating all associations to segments
- Updating all association points (changes χ^2…)



Part of segment A

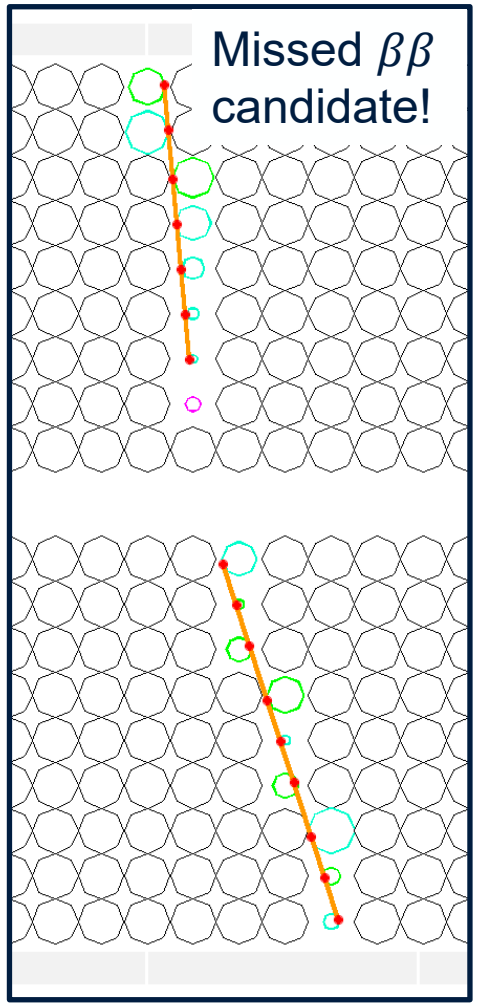
Solution 1

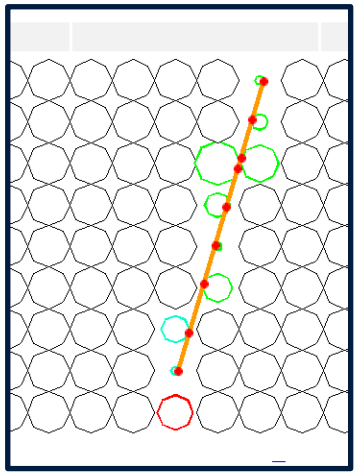


Part of segment B

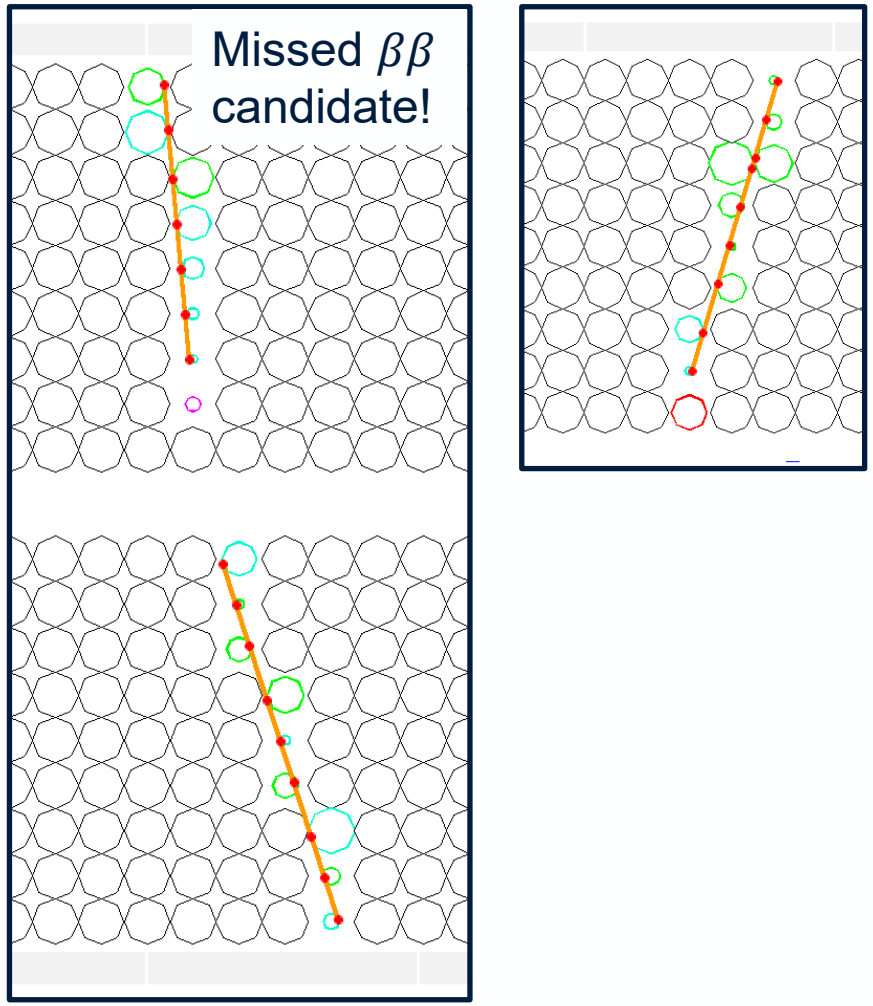
Solution 1

Refinements – trajectory extension



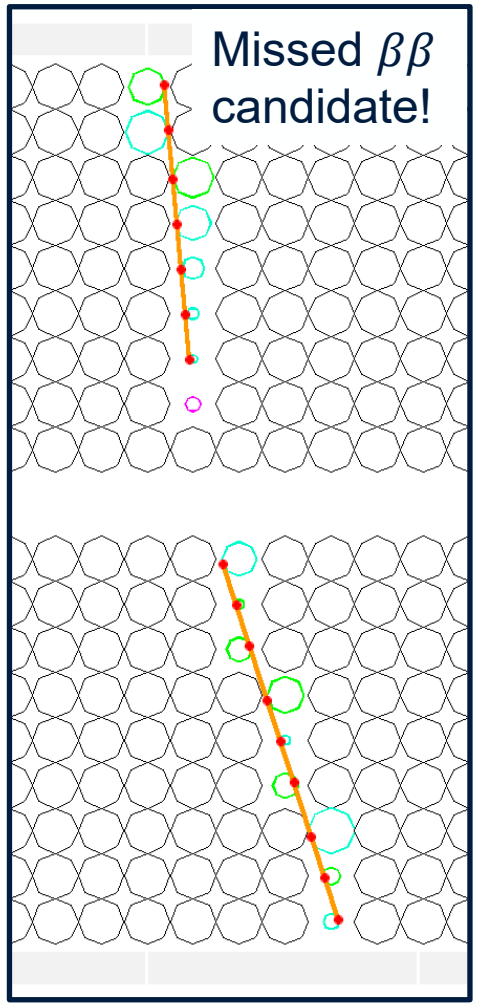


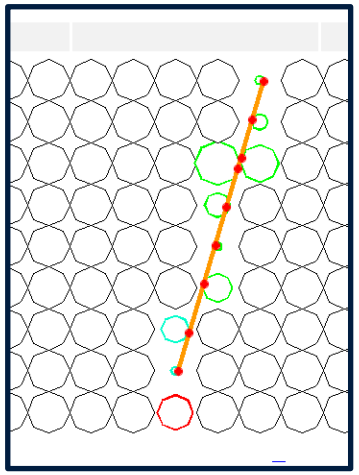
Refinements – trajectory extension



- Extending the trajectories (up to certain length)
- Recovers missed hits at the ends
- Higher mismatch tolerance

Refinements – trajectory extension





- Extending the trajectories (up to certain length)
- Recovers missed hits at the ends
- Higher mismatch tolerance

```
########################
# Trajectory extension #
########################

# Maximum allowed length to prolong the track to include previously unassociated or incorrectly associated hit
polylines.max_extention_distance : real as length = 120.0 mm

# Maximum distance to associate tracker hit to a track during extension
polylines.hit_association_distance : real as length = 12.0 mm
```

What's missing?

1. **TCD input/output conflict**

What's missing?

1. **TCD input/output conflict**
2. **TrajectoryBuilder:**
   - Better control through config
     (on/off switch for individual strategies, better option names…)
   - Make official release (Cimrman 1.2.0)
   - Do we want to reuse the code?

What's missing?

1. **TCD input/output conflict**
2. **TrajectoryBuilder:**
   - Better control through config
     (on/off switch for individual strategies, better option names…)
   - Make official release (Cimrman 1.2.0)
   - Do we want to reuse the code?
3. **Alpha tracking:**
   - No upgrade since last Collaboration meeting
   - Lack of feedback (Antoine is using TrackFit)
   - More testing needed
   - Communication with official drift model

What's missing?

4. **Solutions in PTD bank?**
   - At least a module dedicated for decision making


SUPER NEMO
Source = ββ Isotope = ^{82}Se
Drift cells (Geiger mode)
e⁻
Calorimeter energy
Drift hits
Reconstructed track
Vertex (ββ decay)
DON'T PANIC
It's not just a detector. It's a trip.
CIMRMAN RECONSTRUCTION MODULE
CIMRMAN
Turning hits into understanding.
Mostly harmless.

Thank you