
Using transformer for trajectory reconstruction at the HL-LHC

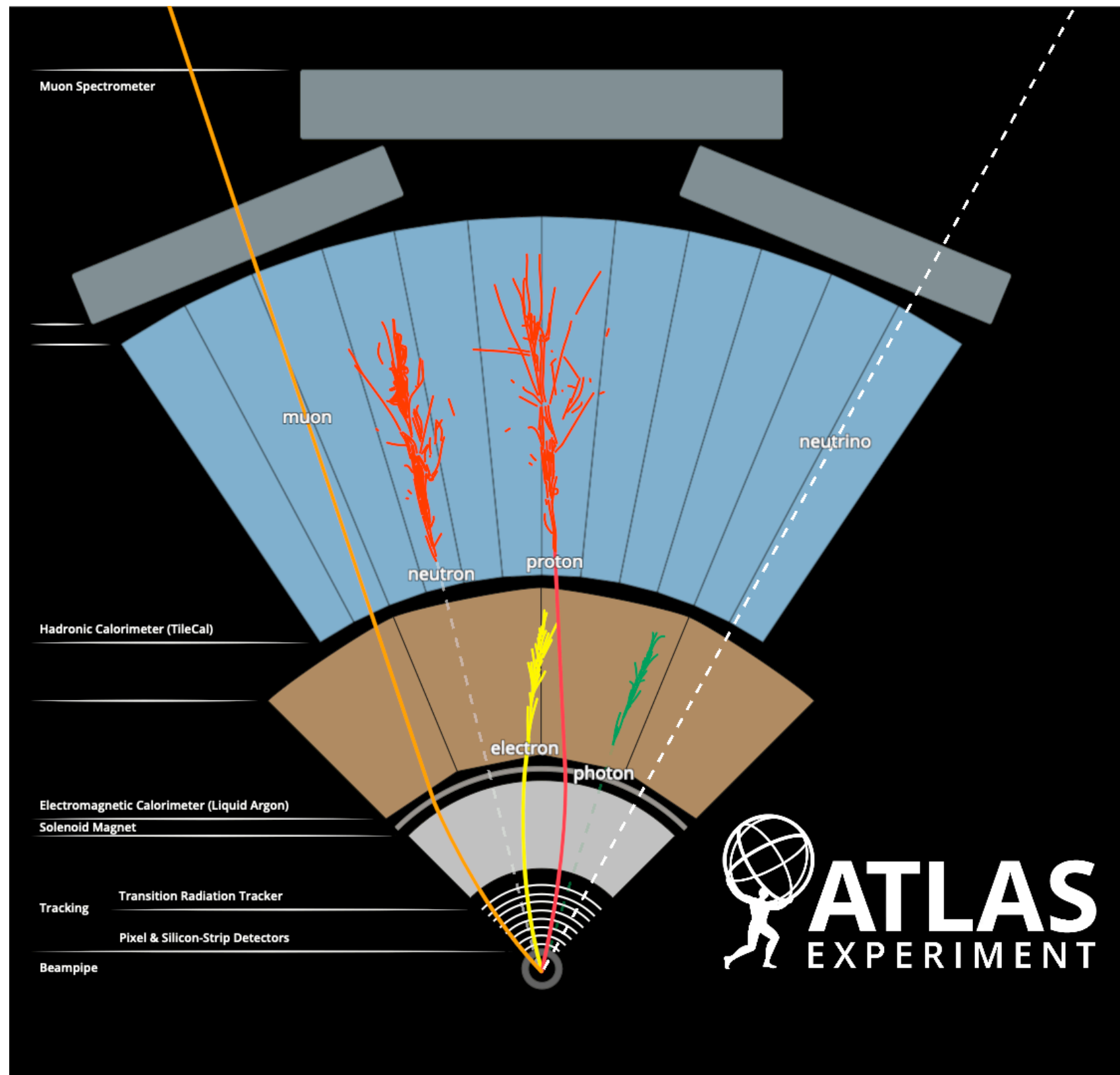
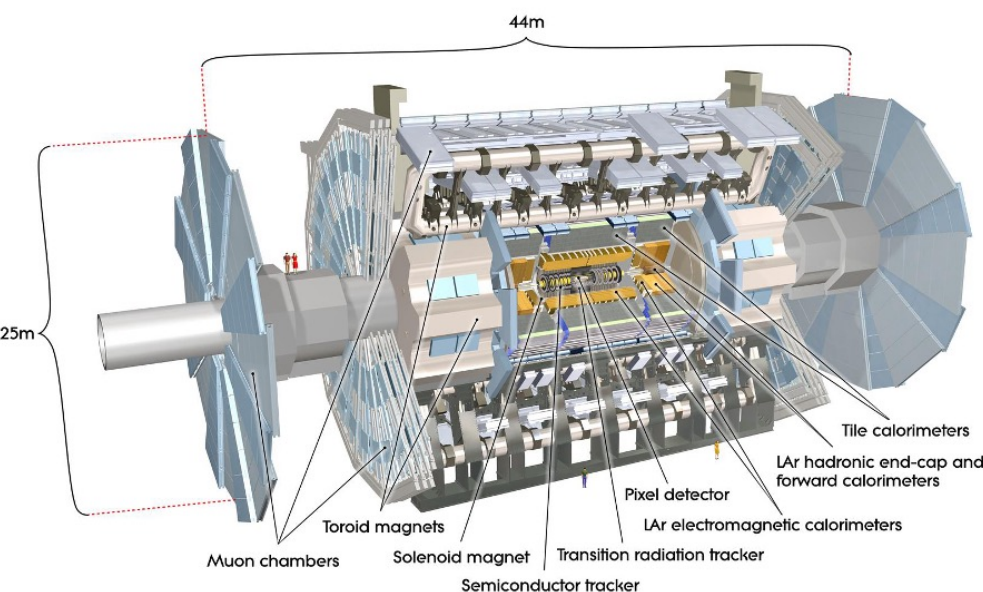


 Corentin Allaire



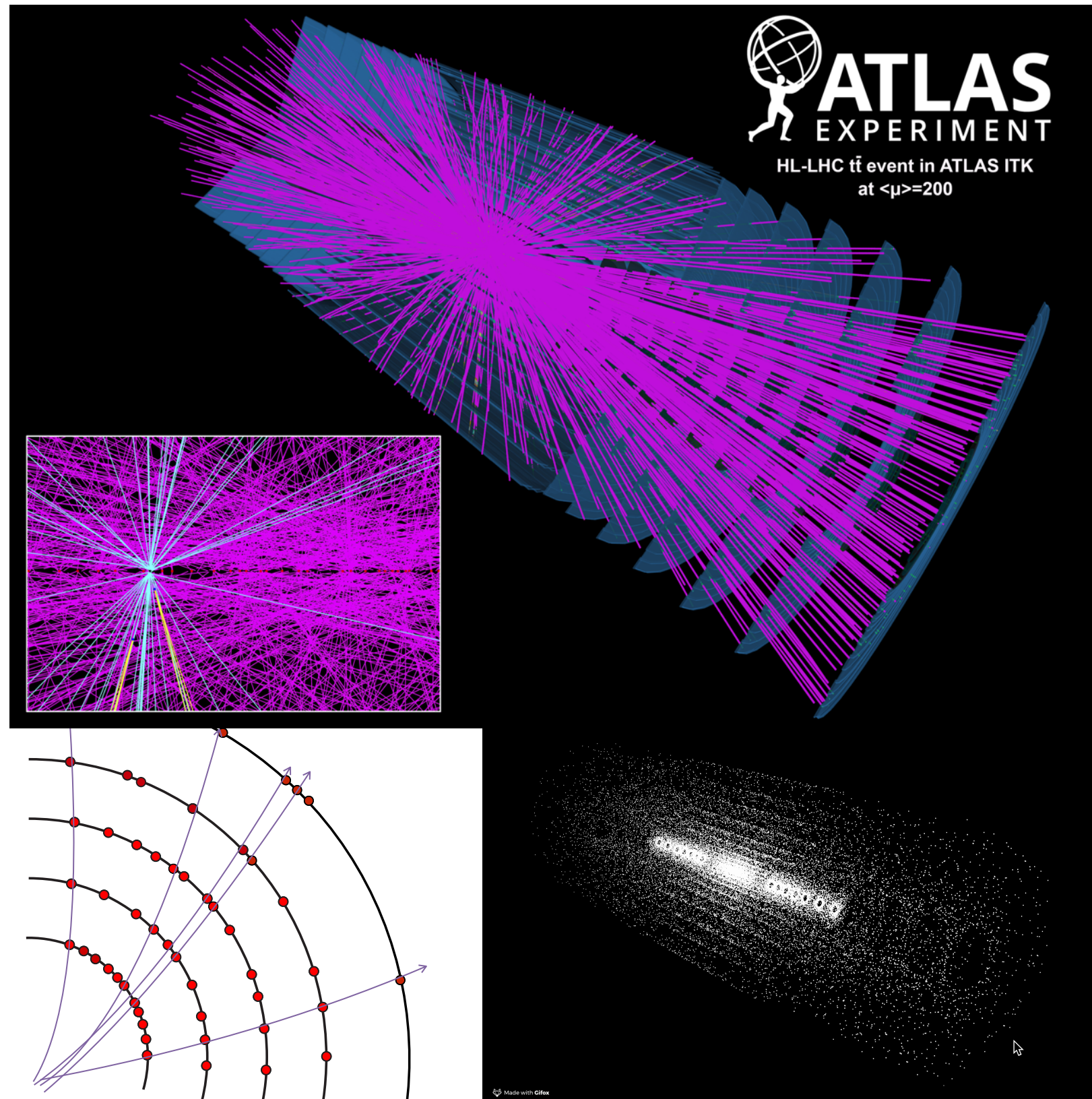
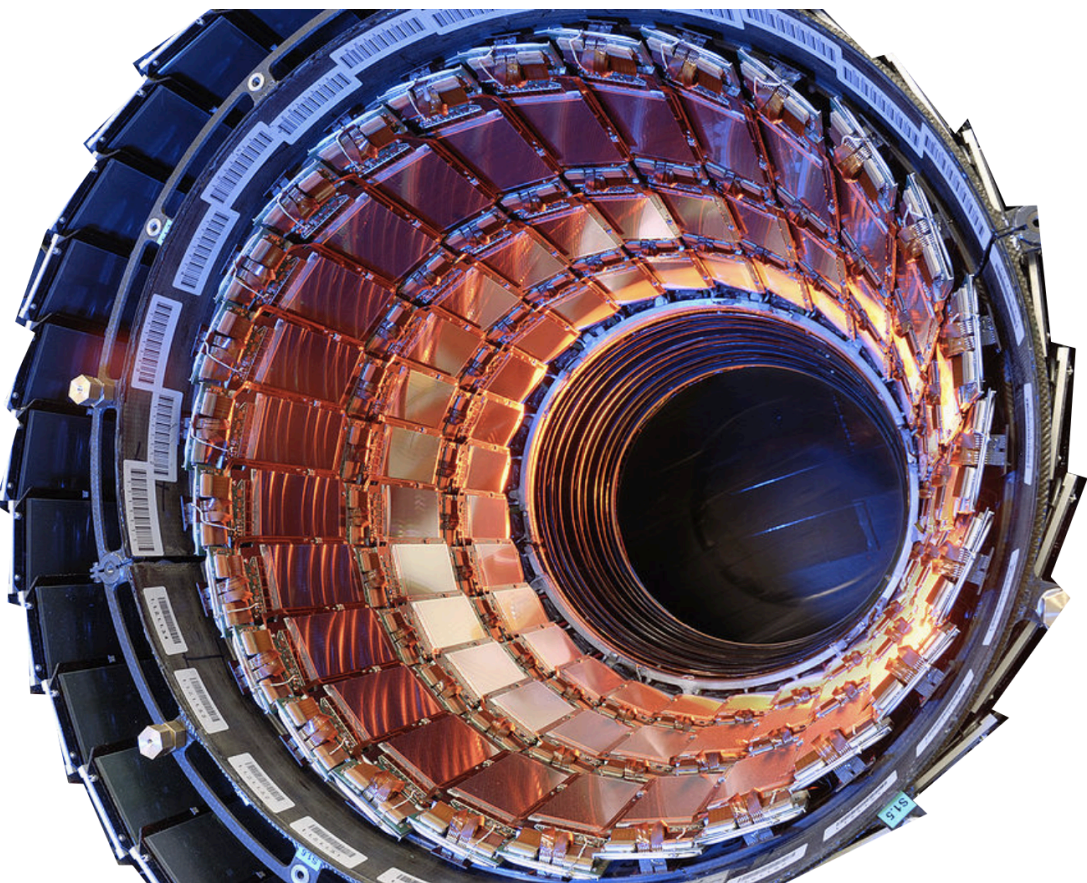
Principle of the particle detector

- Different layers for different types of particles
- Centre: **Tracker** reconstructs charged particles trajectories
- Outer Layer: **Calorimeter** reconstructs particles energies



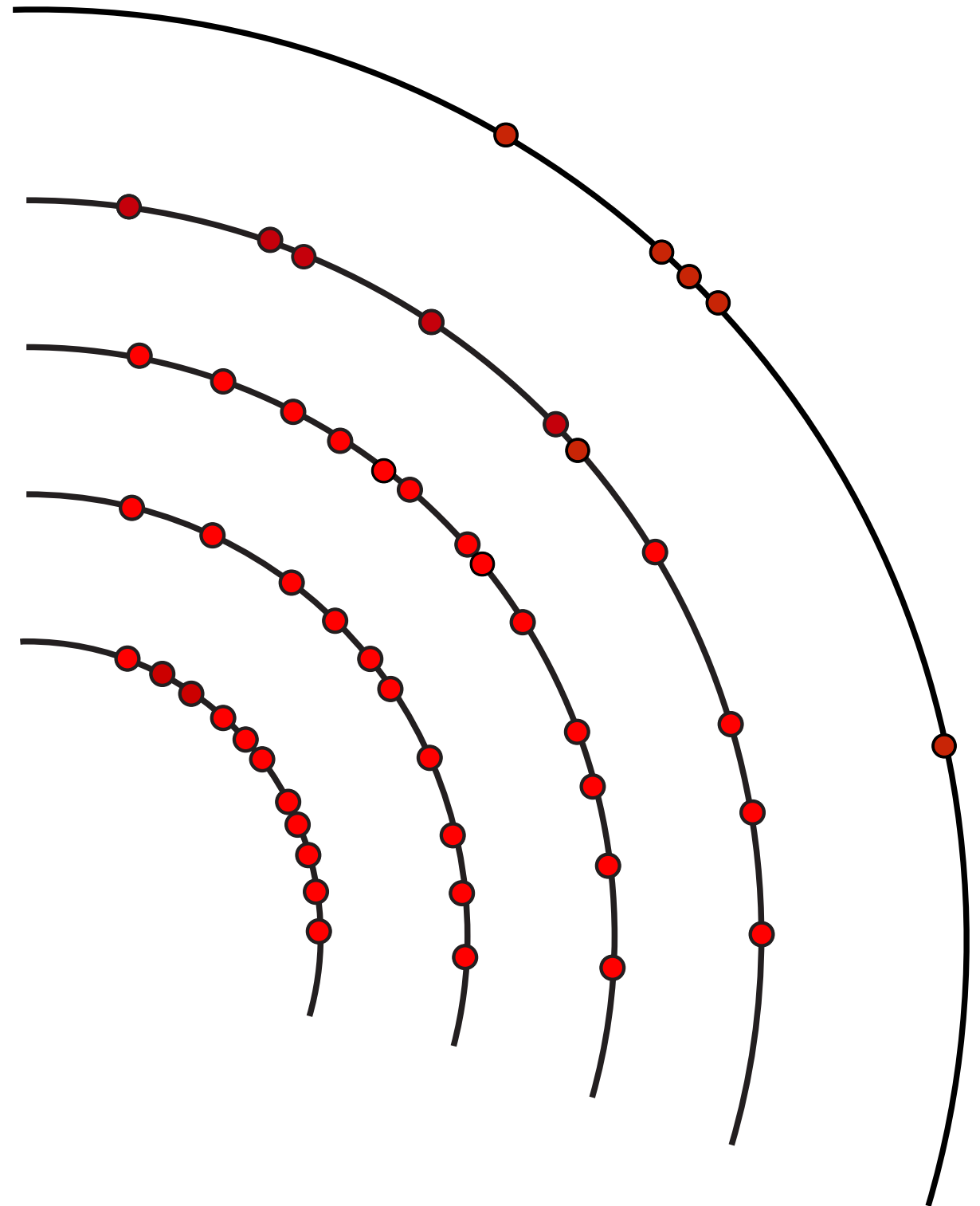
Tracker

- Use **silicon** or **gas-based** sensors
- Particle ➡ series of **3D points**
- Only detect **charged** particles



Track Reconstruction

Space point formation
Clustering



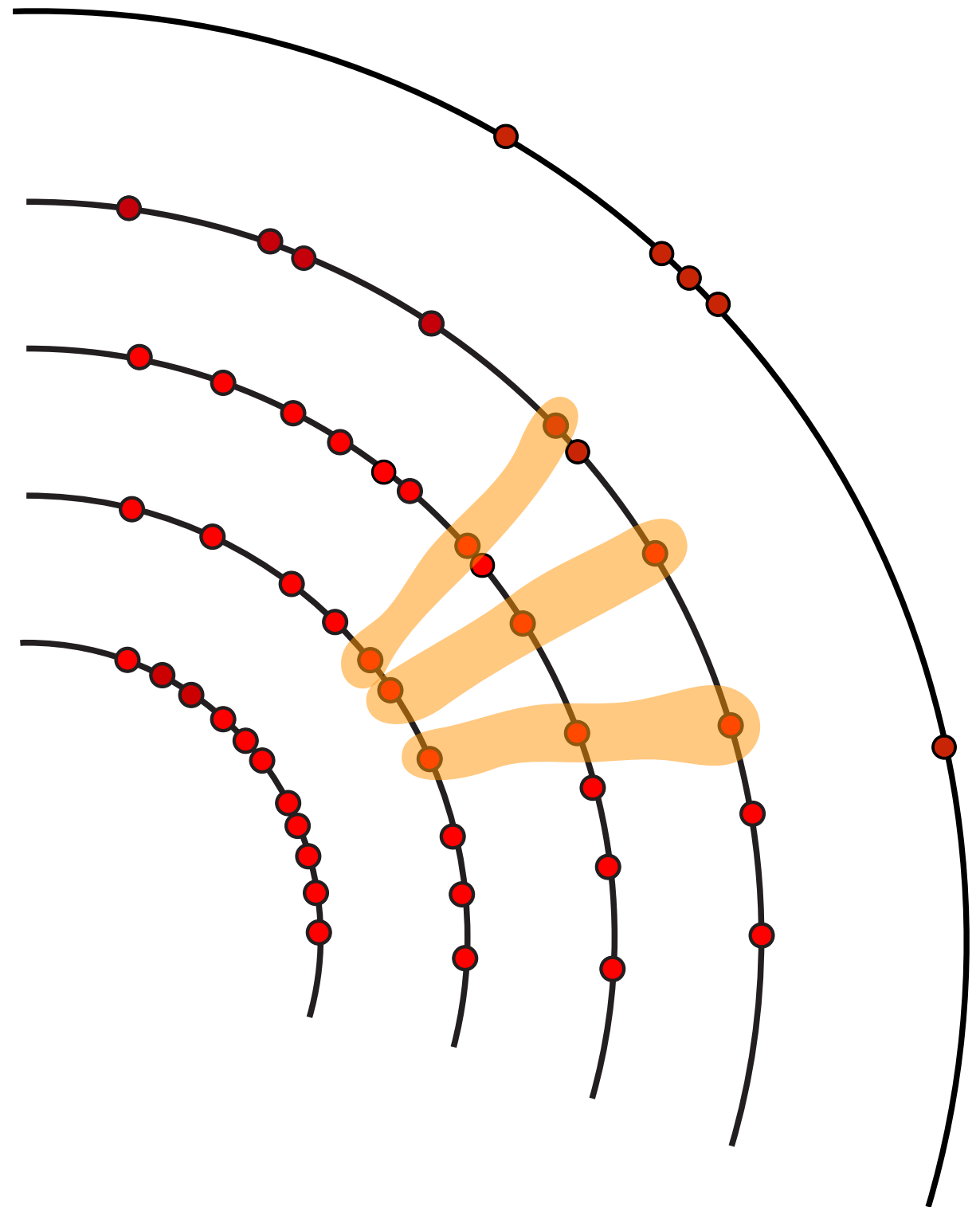
- Hits in the detector are collected to create **measurement points**

Track Reconstruction

Space point formation
Clustering

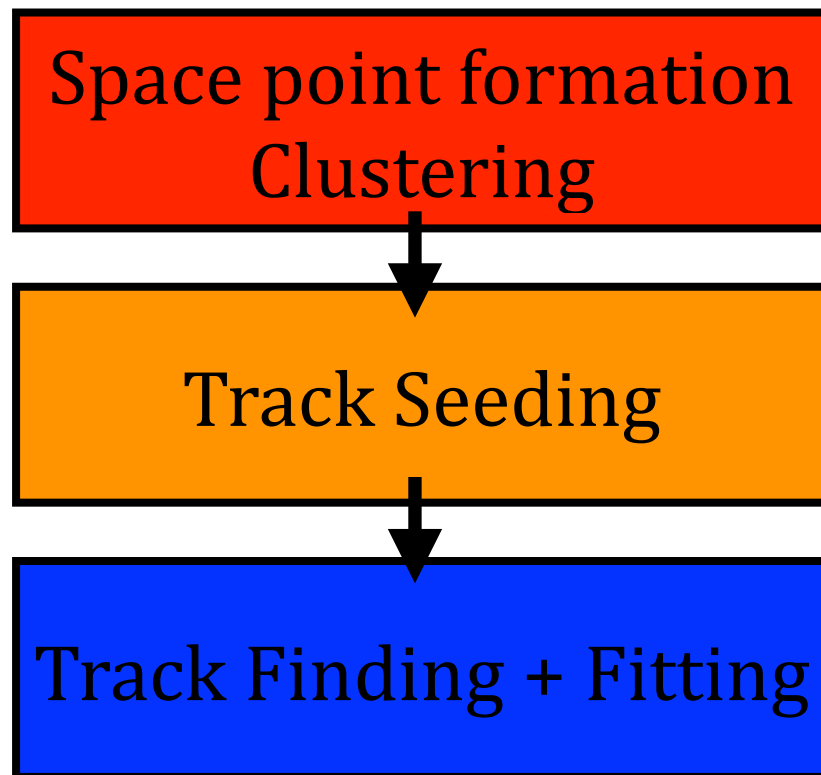


Track Seeding

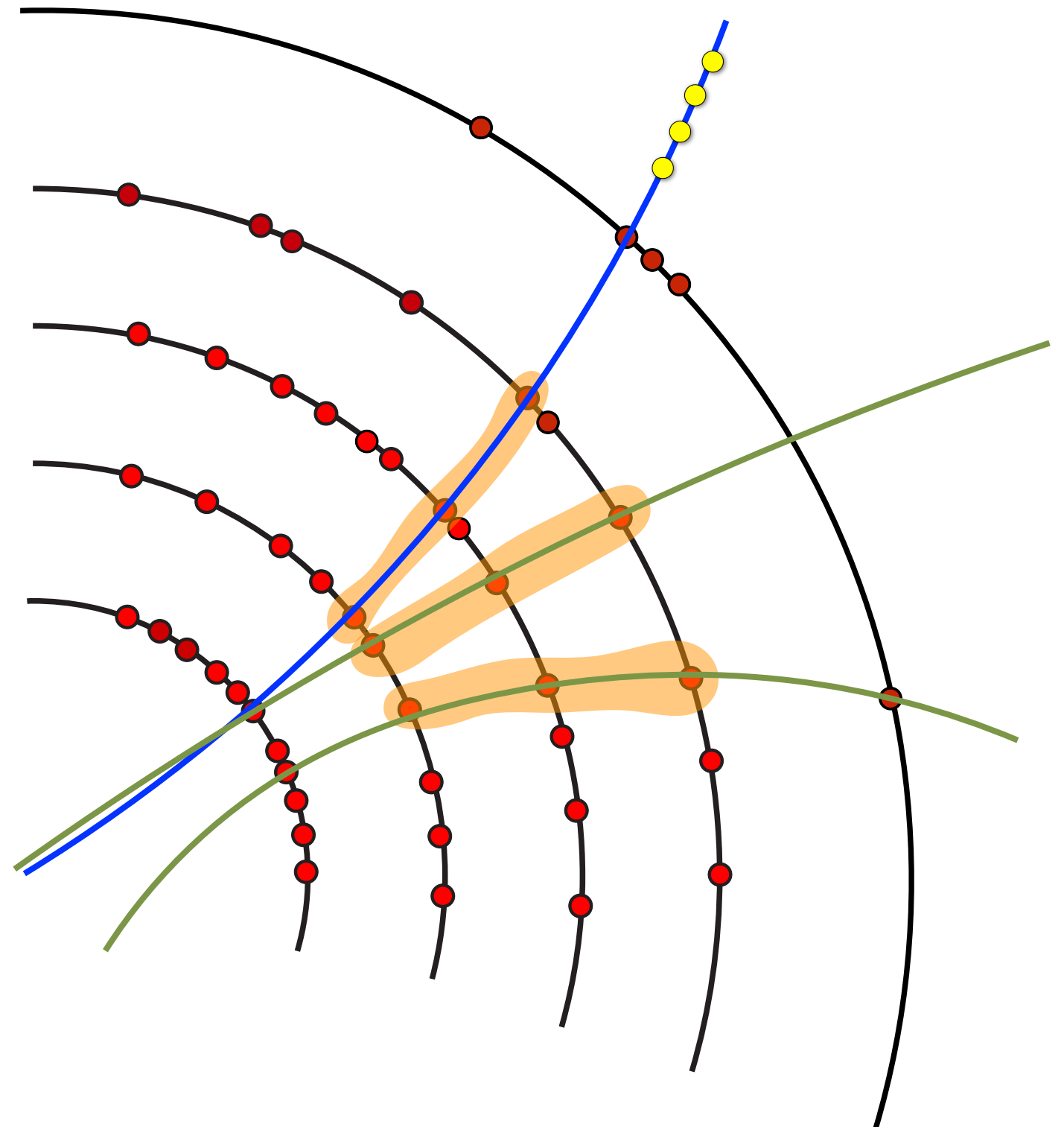


- **Seeds**: groups of **three measurements** compatible with **track's hypothesis**
- Extended to create the track candidates

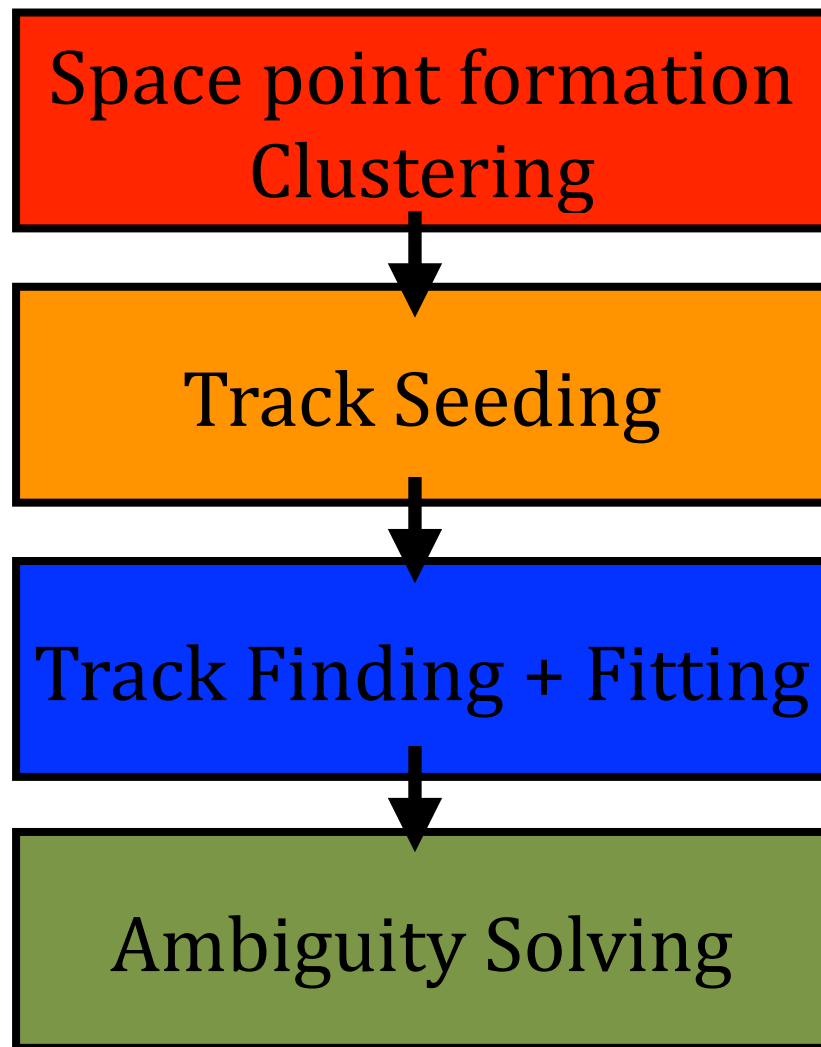
Track Reconstruction



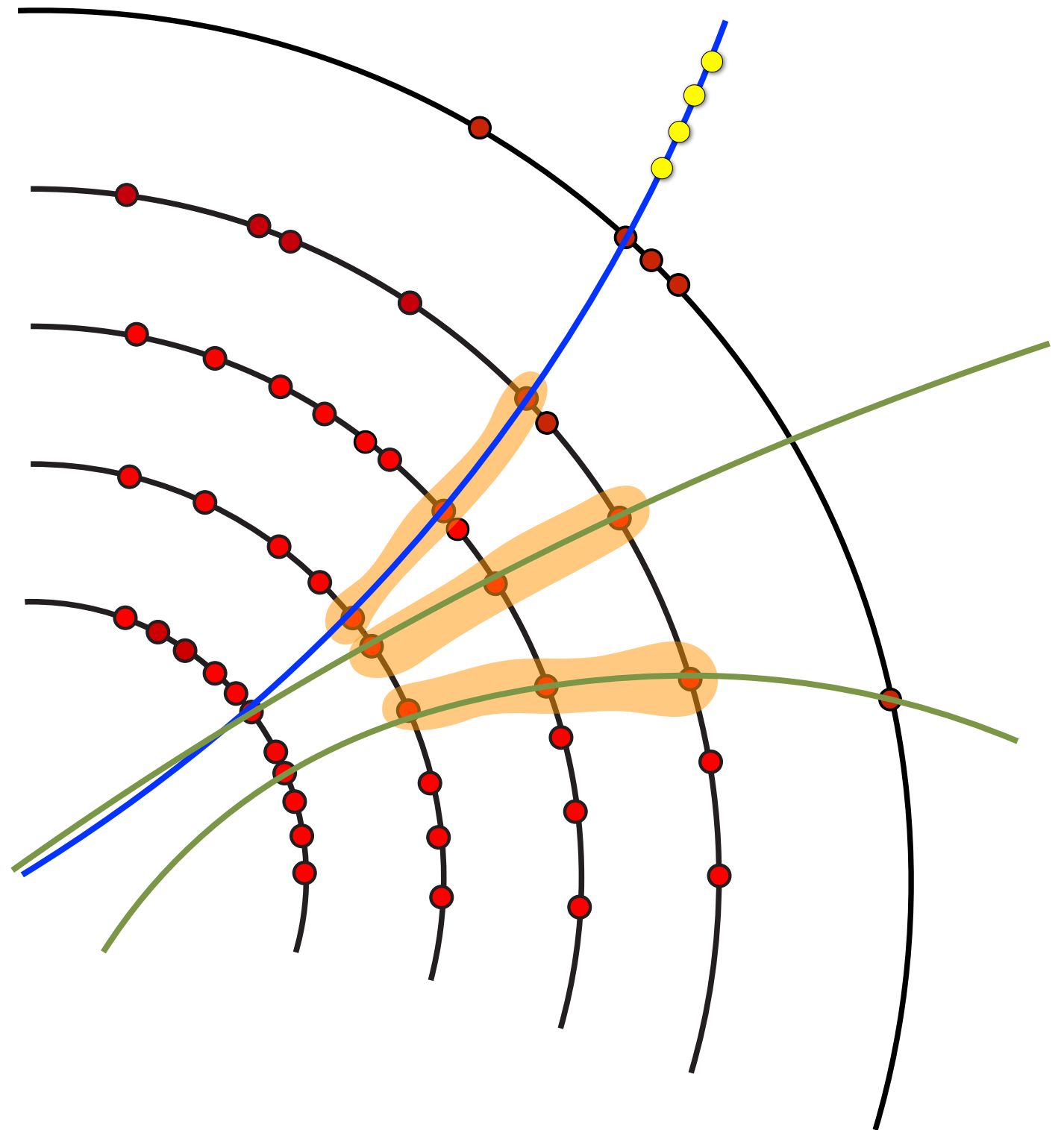
- **Combinatorial Kalman filter** used to build track candidates
- **Track Candidate** = Seed + compatible measurements
- **Tracks** are fitted on the fly



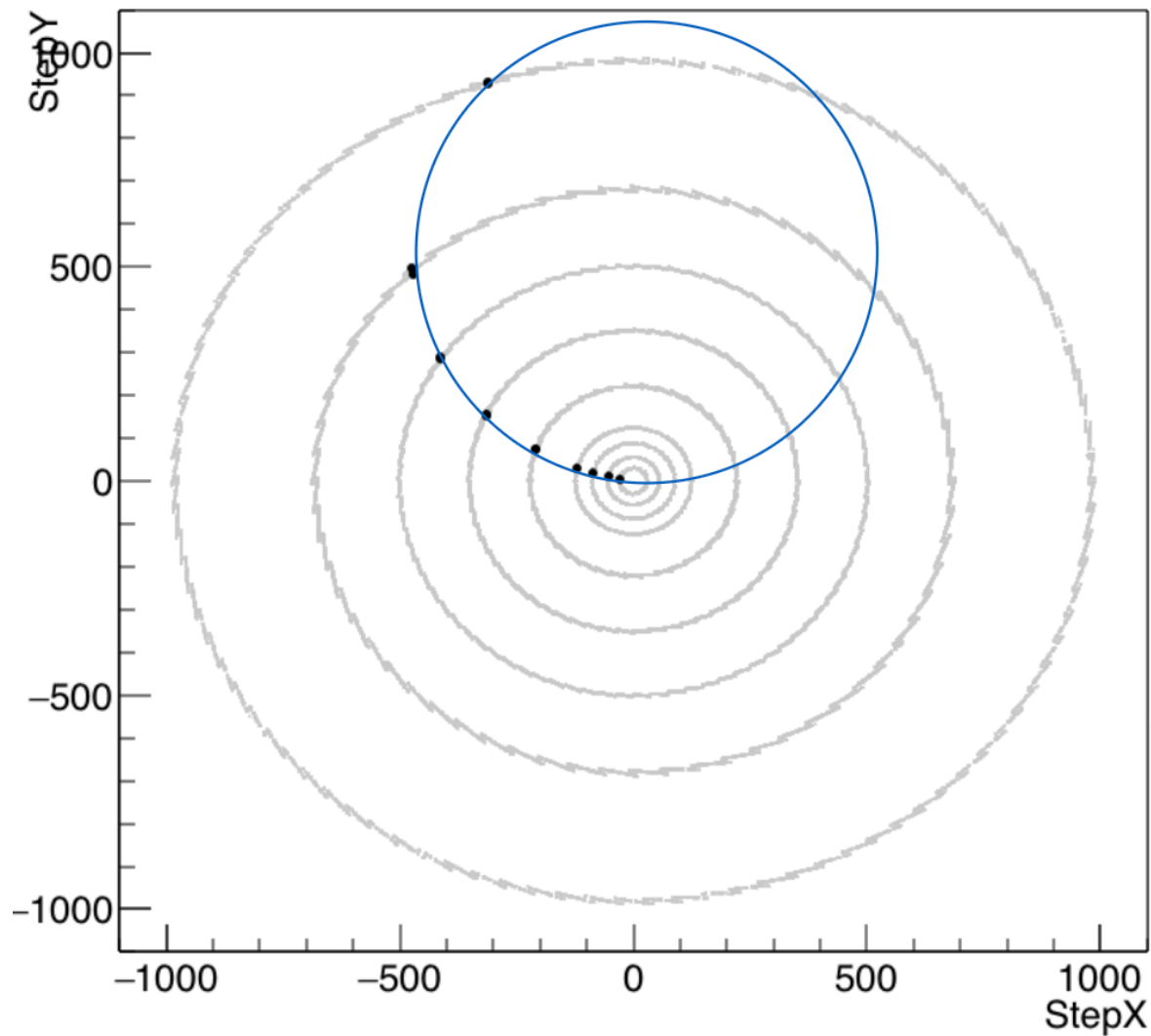
Track Reconstruction



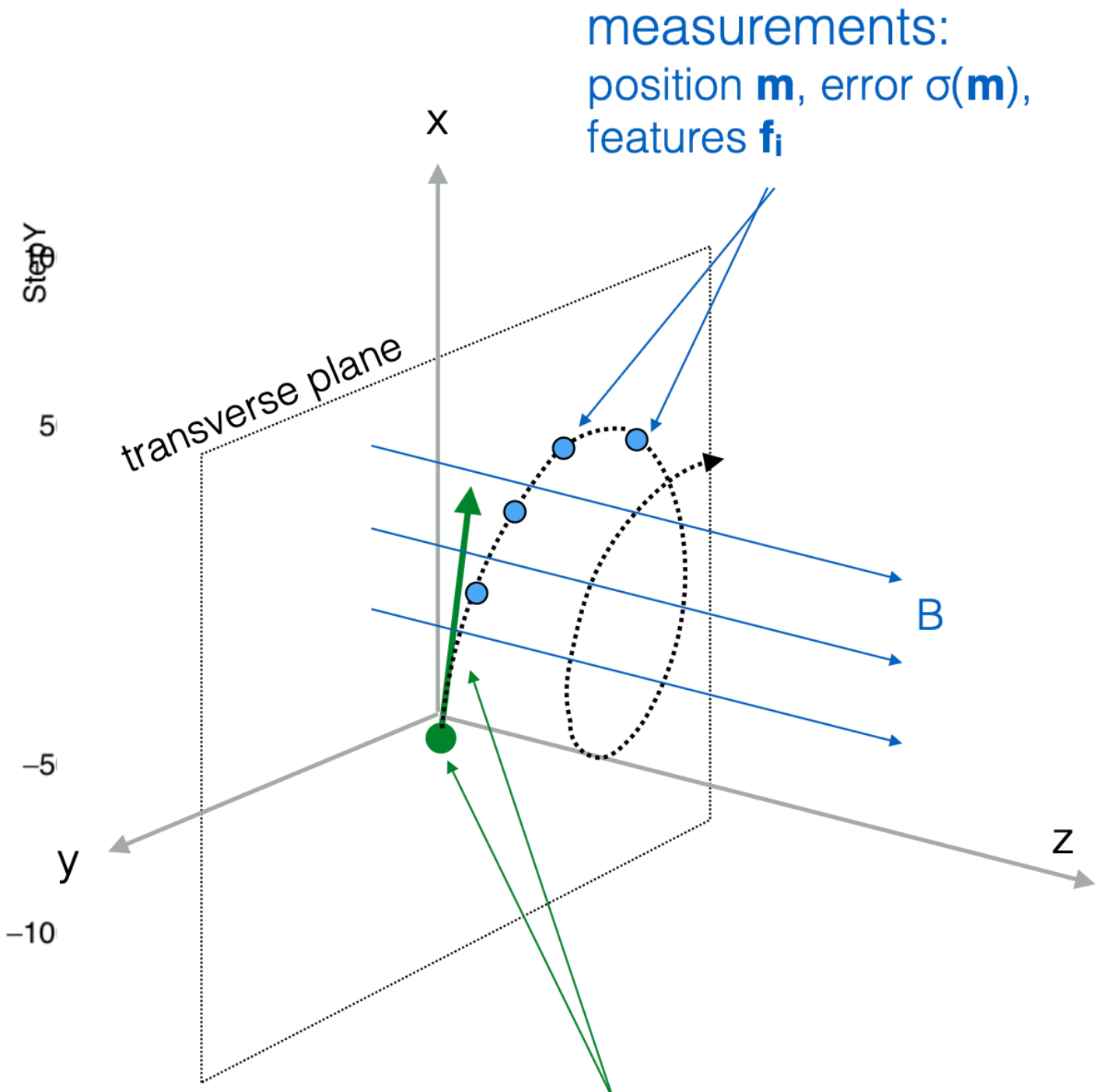
- Score associated with each track candidate
- **Resolve ambiguity:** reconstruct tracks in descending order of a track score



Magnetic field



hits from 1 particle

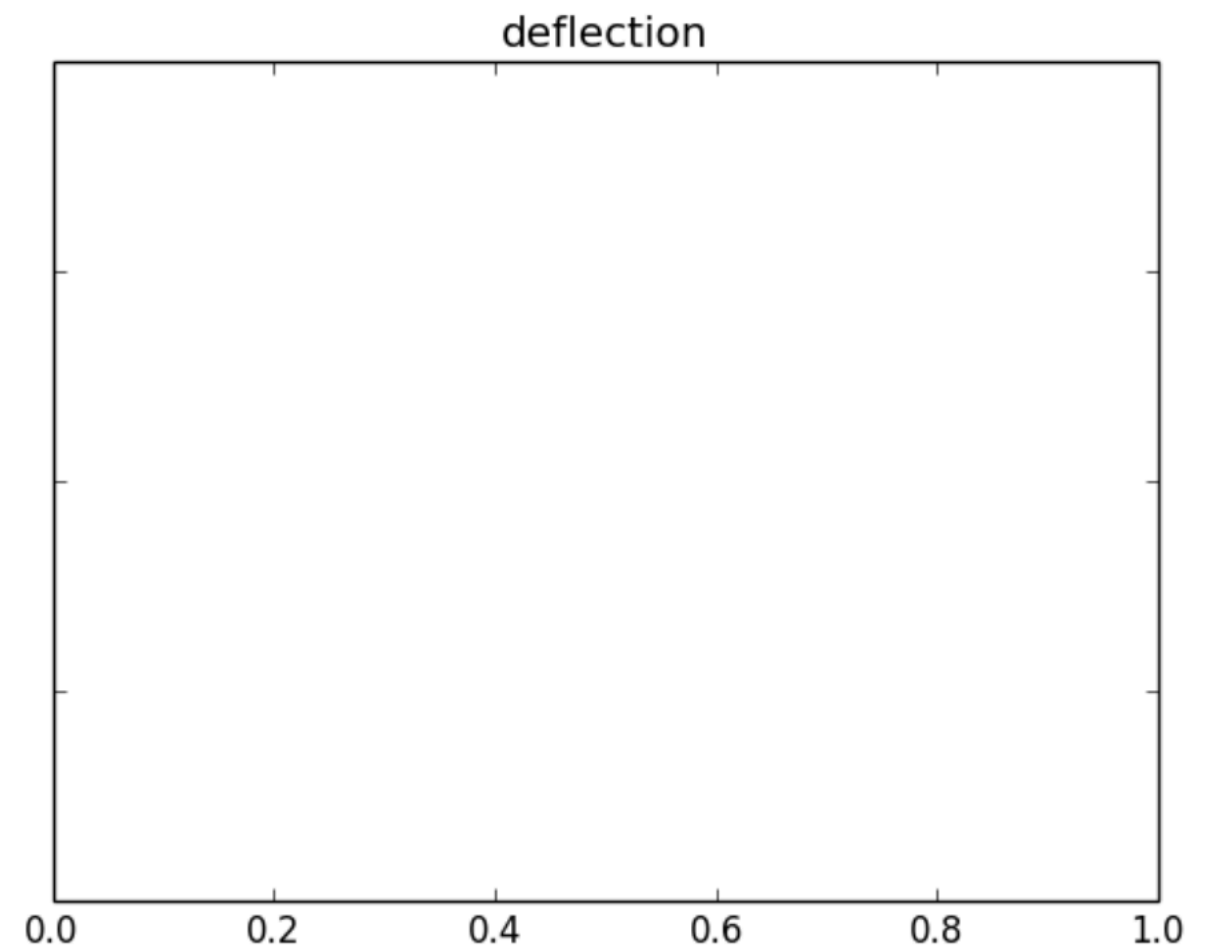
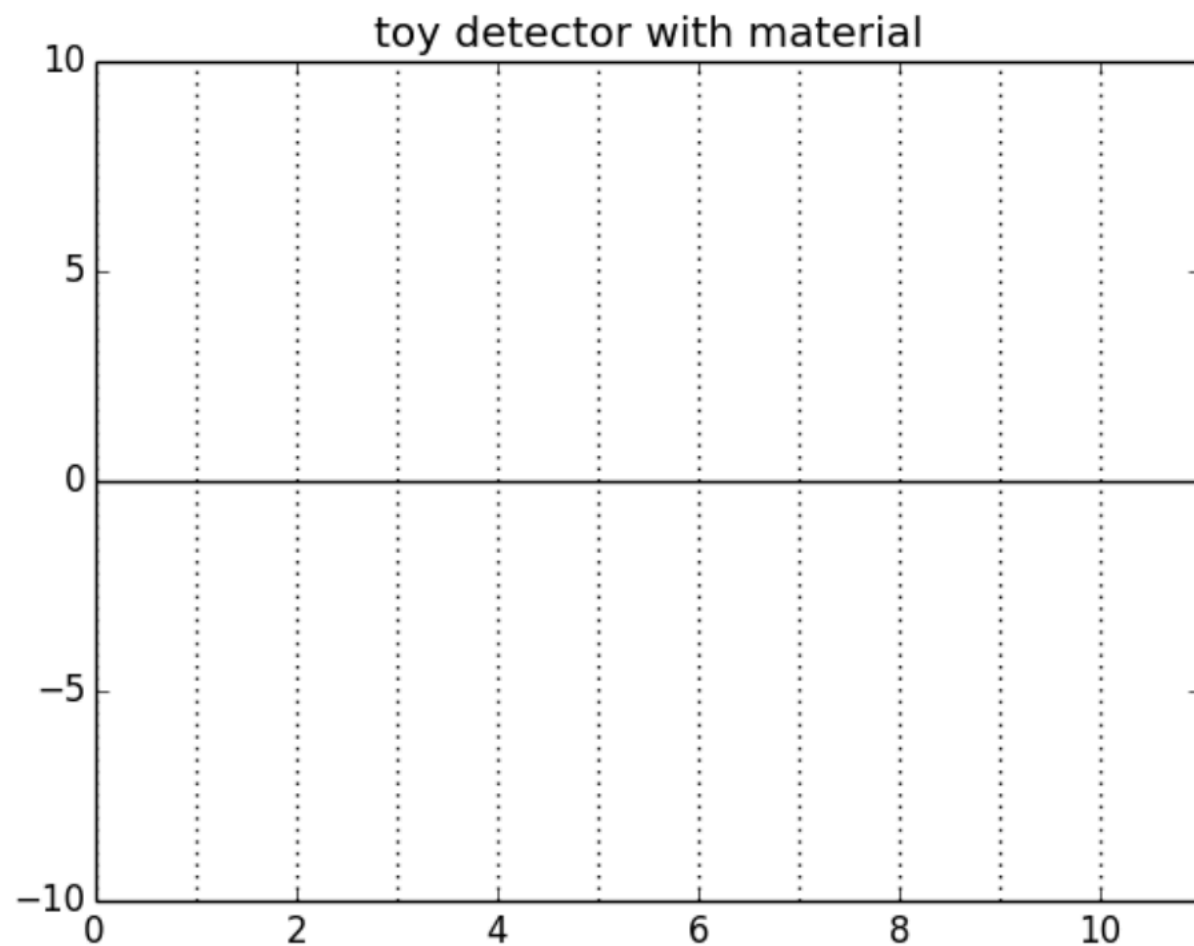


Initial particle parameters:
position $\mathbf{x}$, momentum $\mathbf{p}$, charge q

Material interactions

Particle interaction through matter:

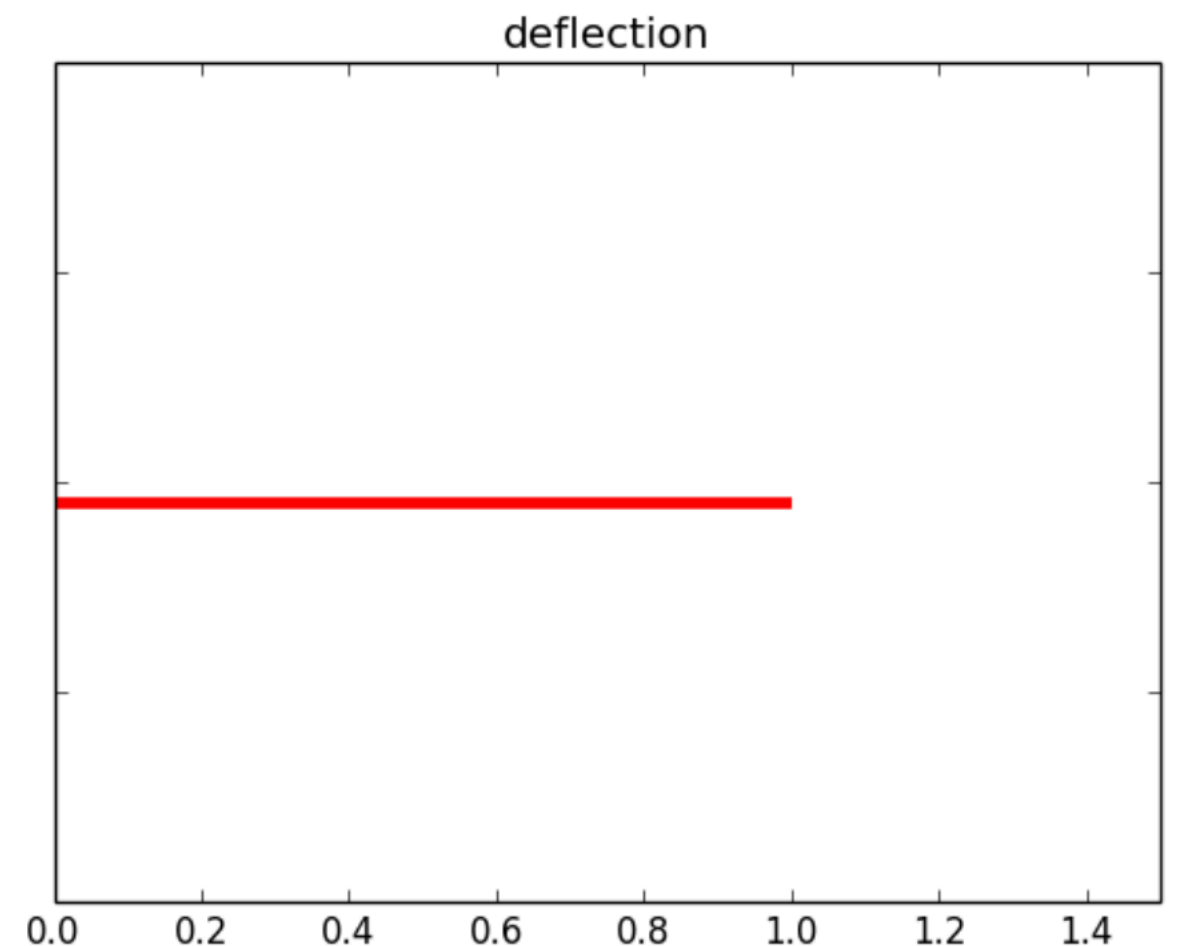
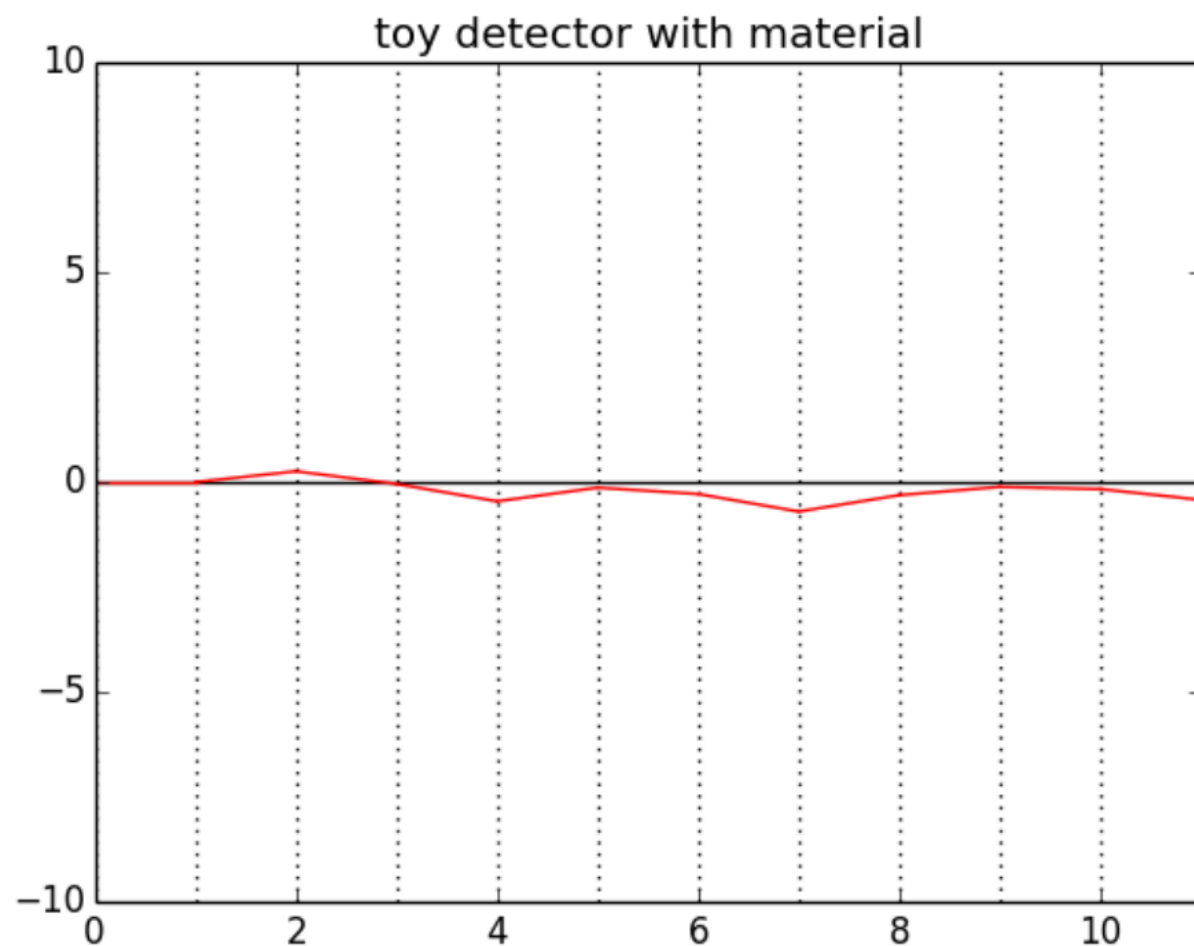
- Deviate the particle trajectory
- Effects inertly proportional to the momentum
- Almost Gaussian noise



Material interactions

Particle interaction through matter:

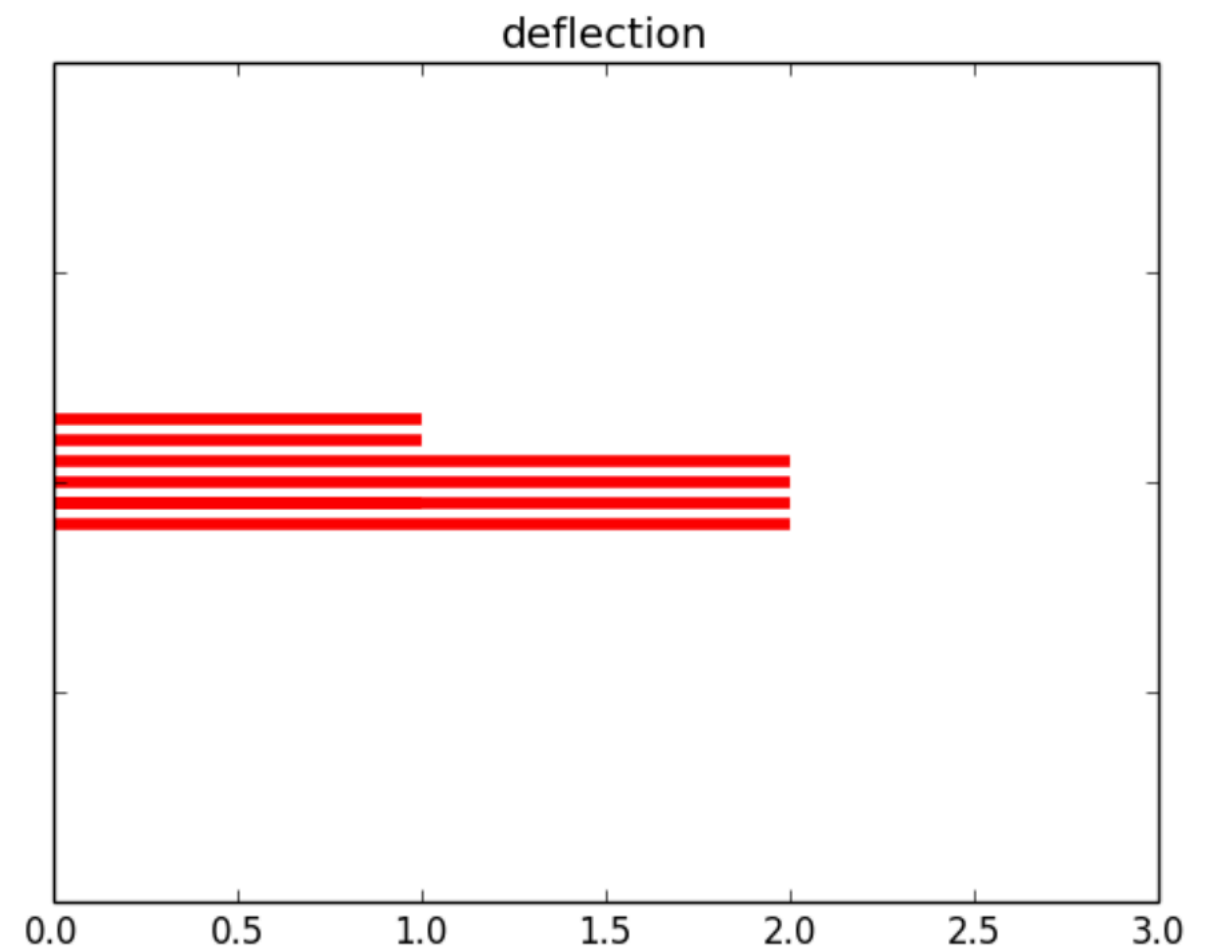
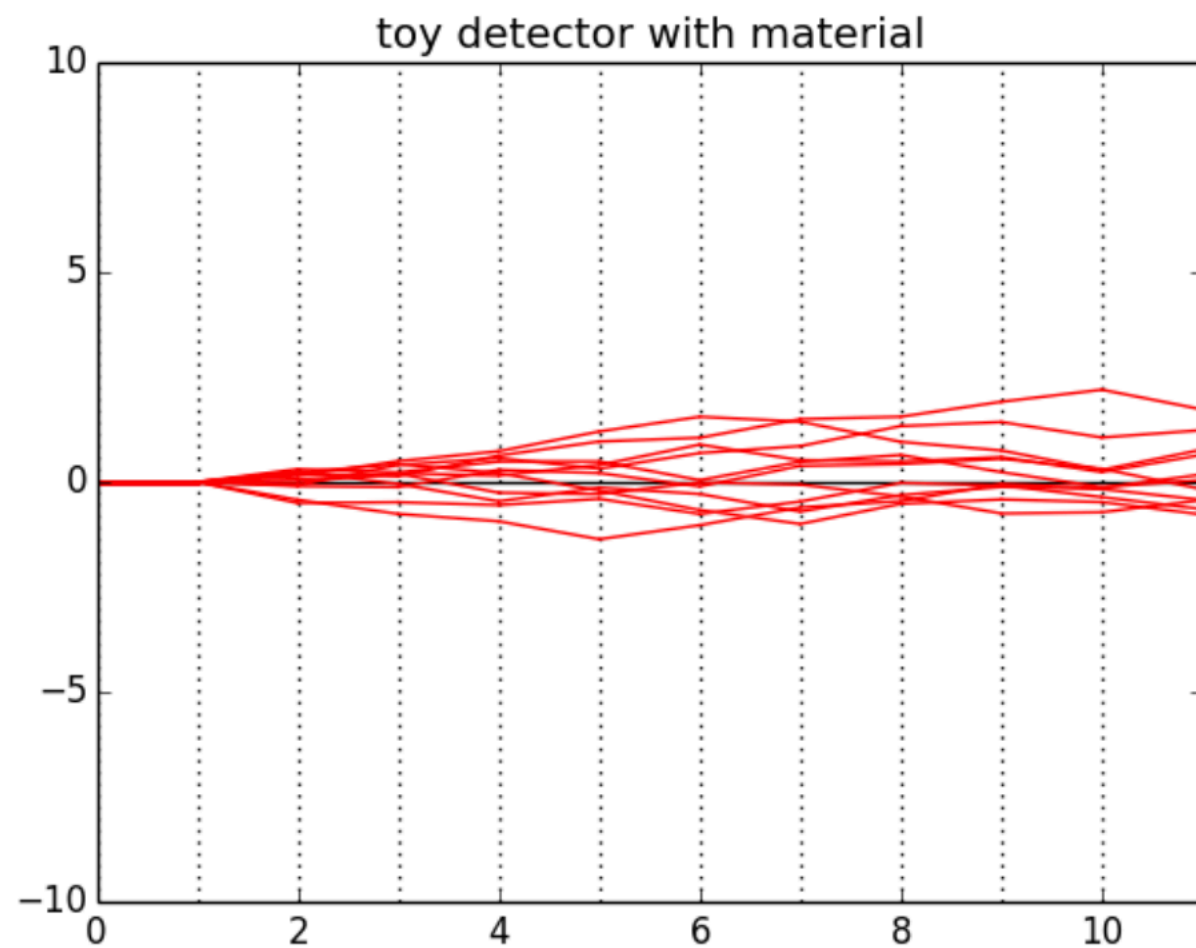
- Deviate the particle trajectory
- Effects inertly proportional to the momentum
- Almost Gaussian noise



Material interactions

Particle interaction through matter:

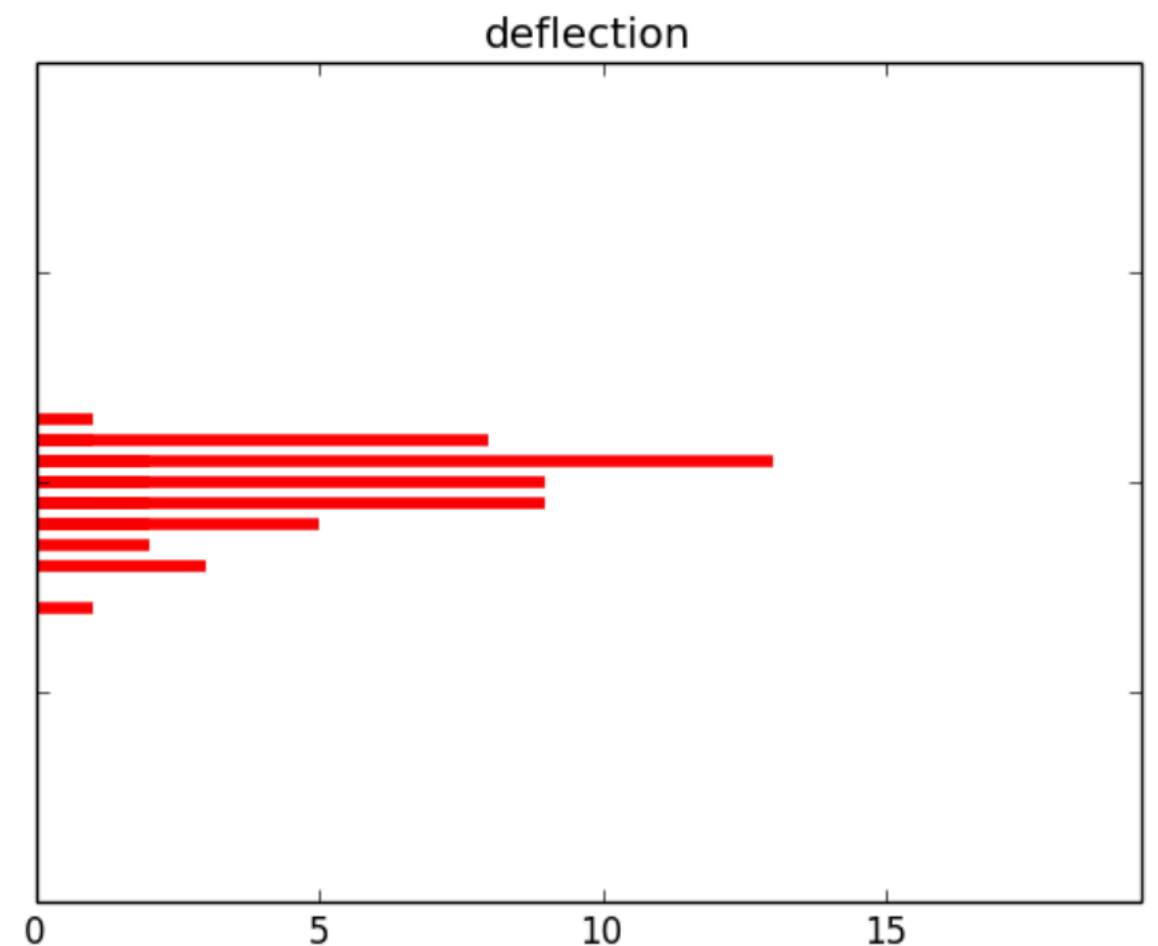
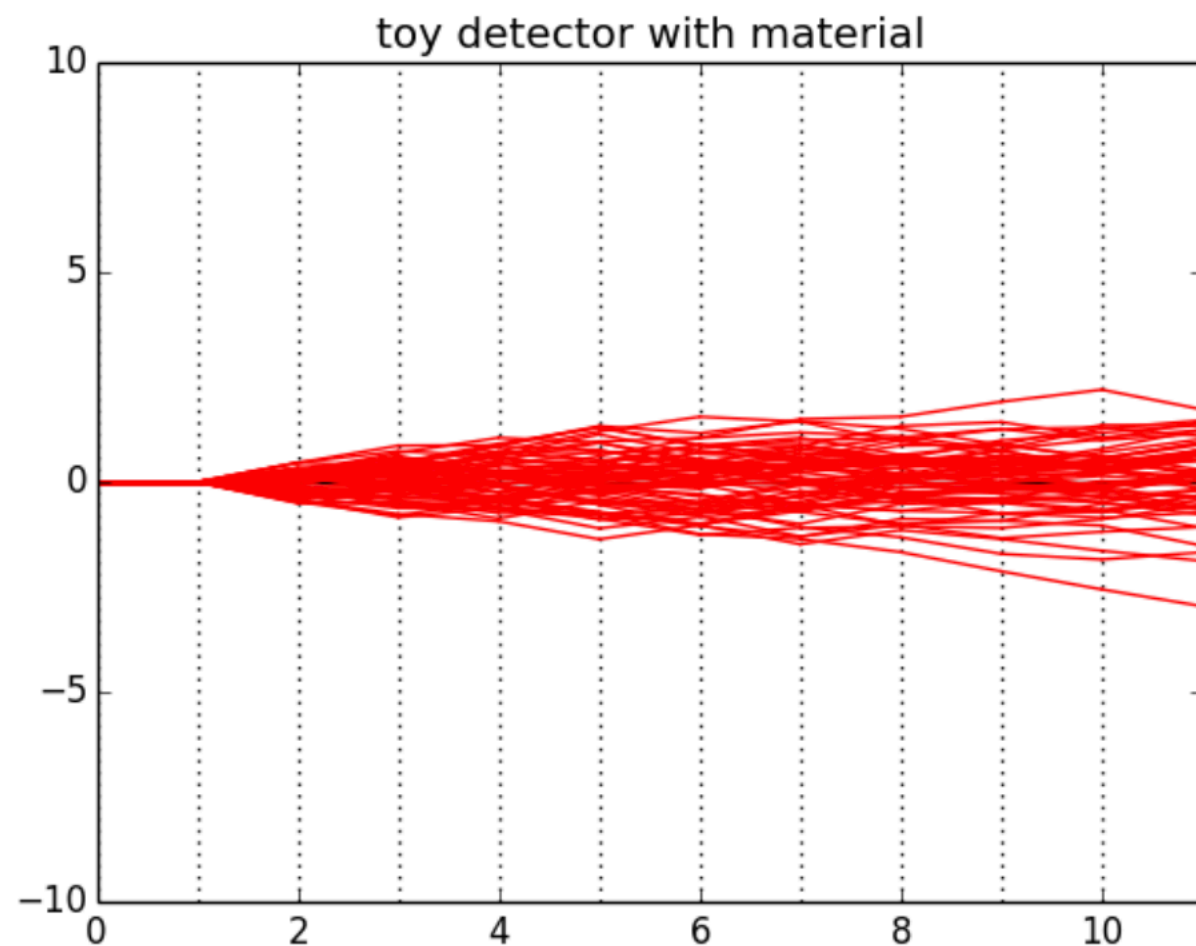
- Deviate the particle trajectory
- Effects inertly proportional to the momentum
- Almost Gaussian noise



Material interactions

Particle interaction through matter:

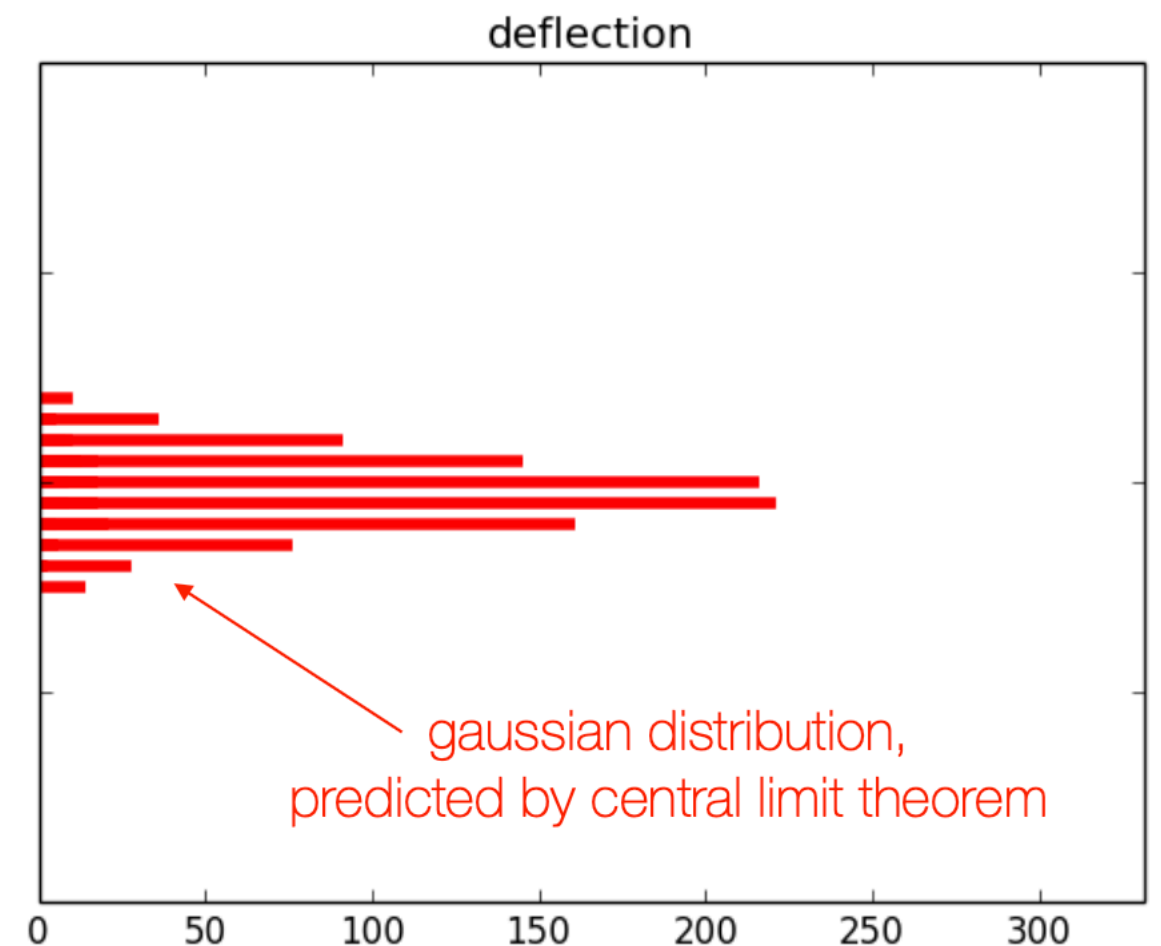
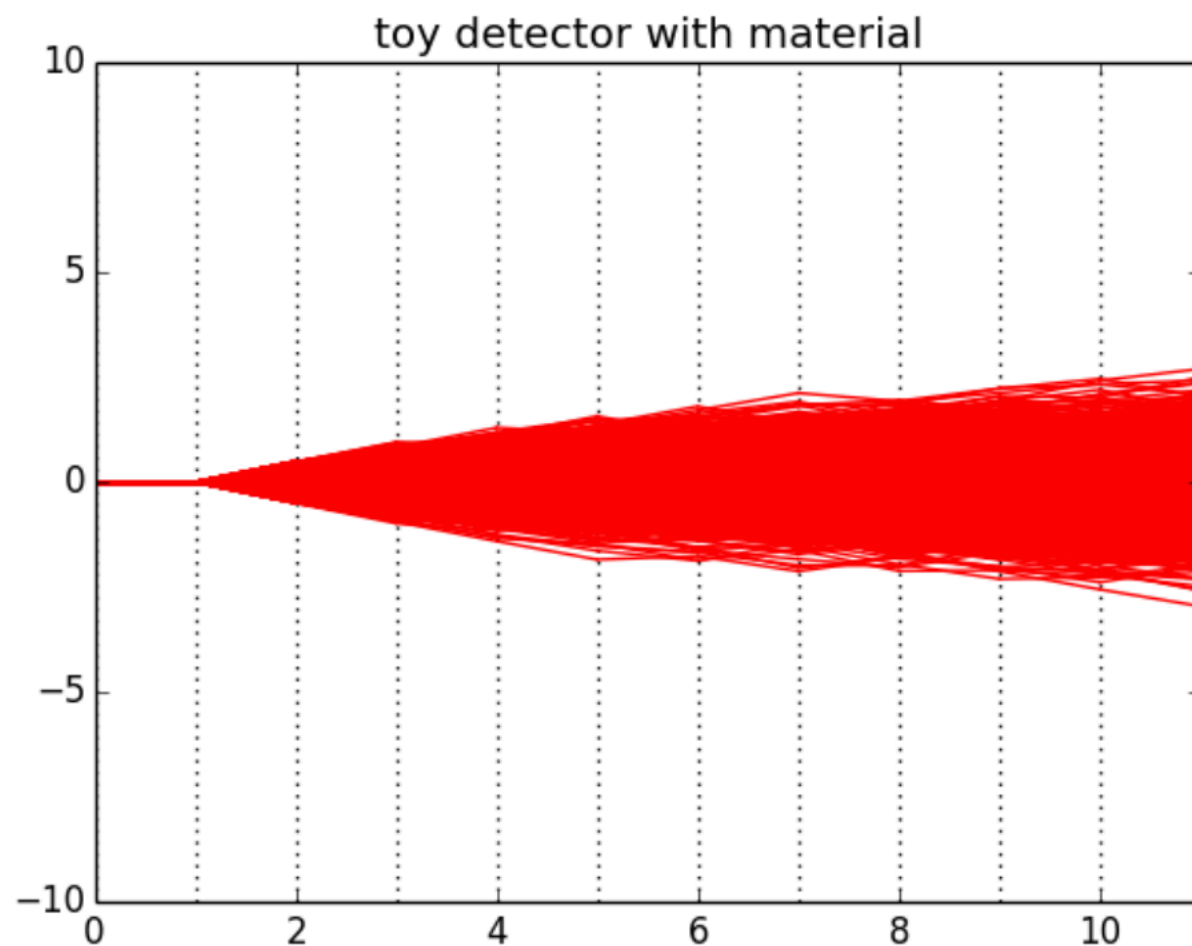
- Deviate the particle trajectory
- Effects inertly proportional to the momentum
- Almost Gaussian noise



Material interactions

Particle interaction through matter:

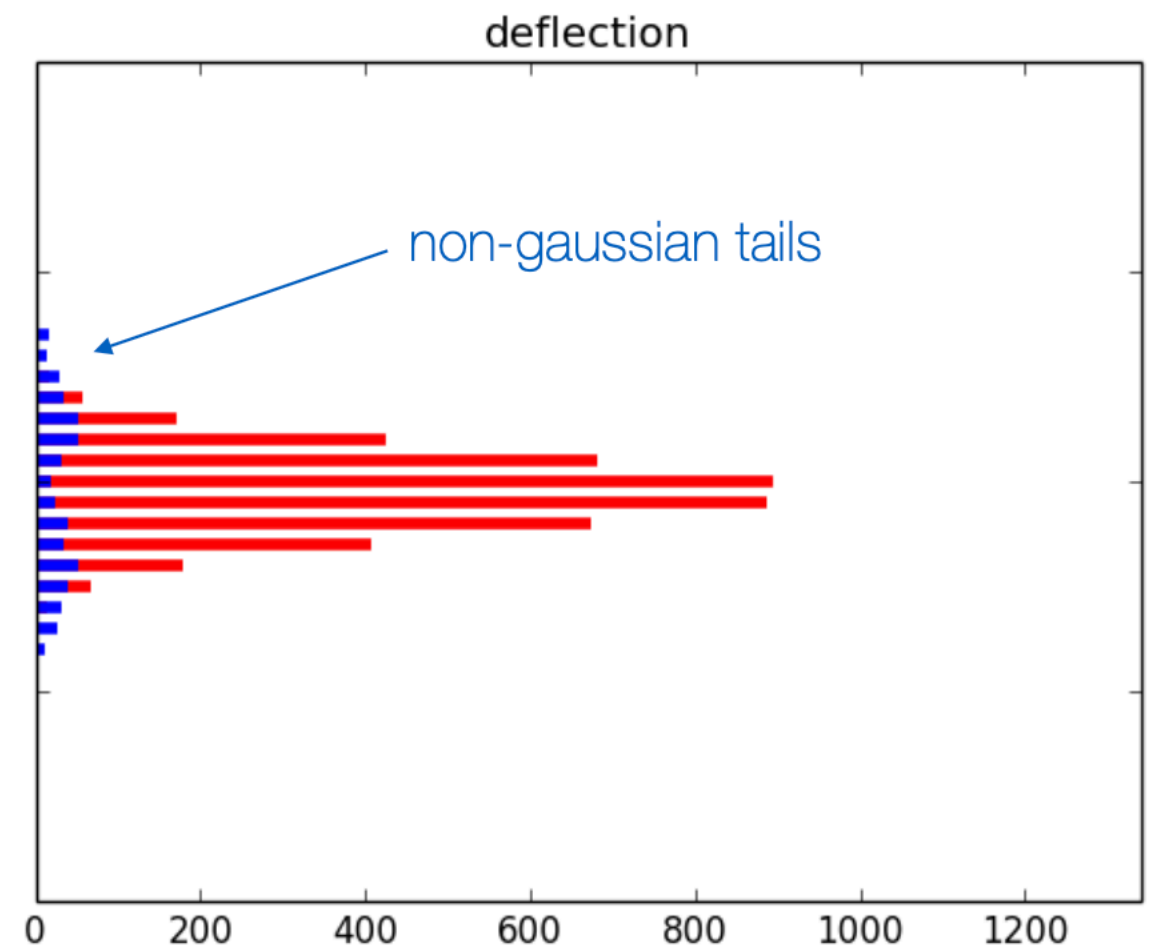
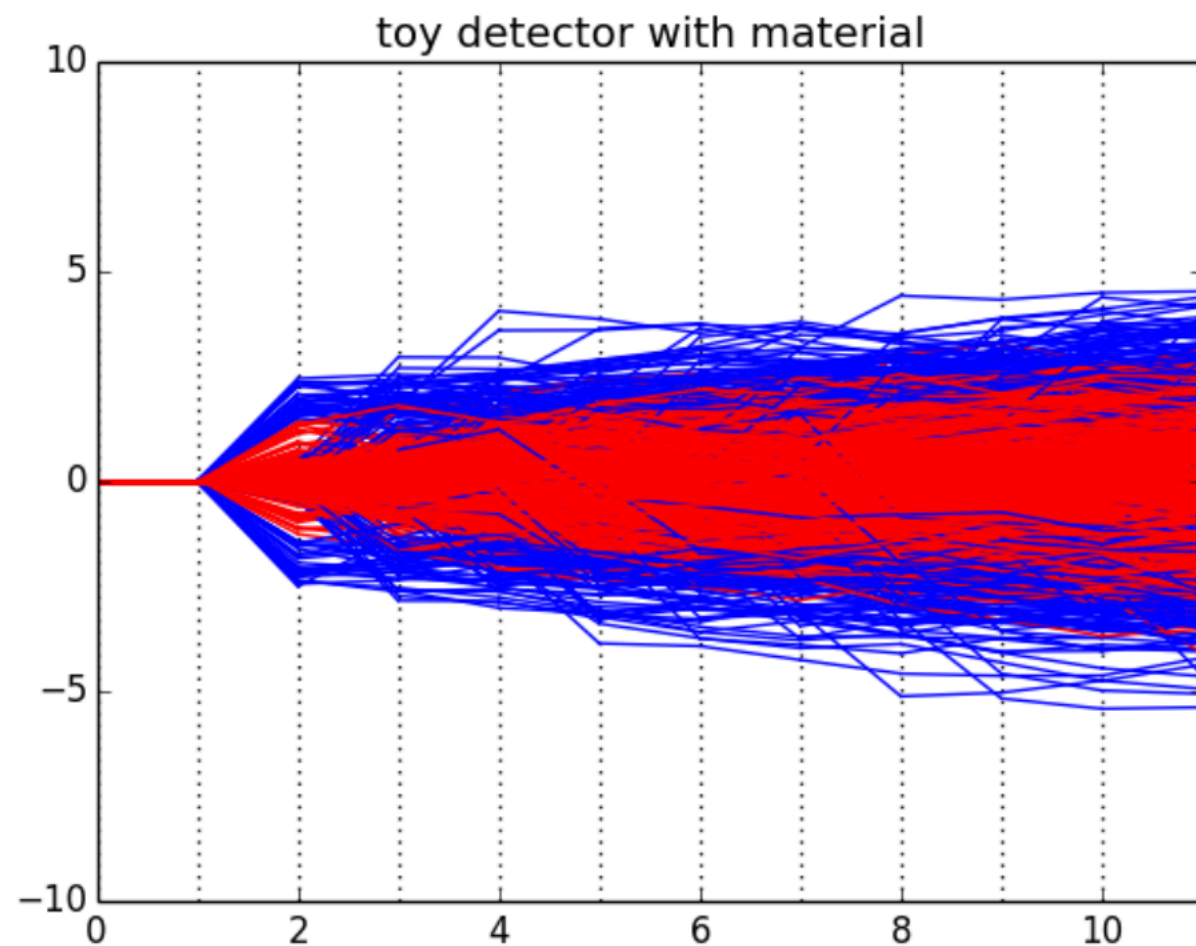
- Deviate the particle trajectory
- Effects inertly proportional to the momentum
- Almost Gaussian noise



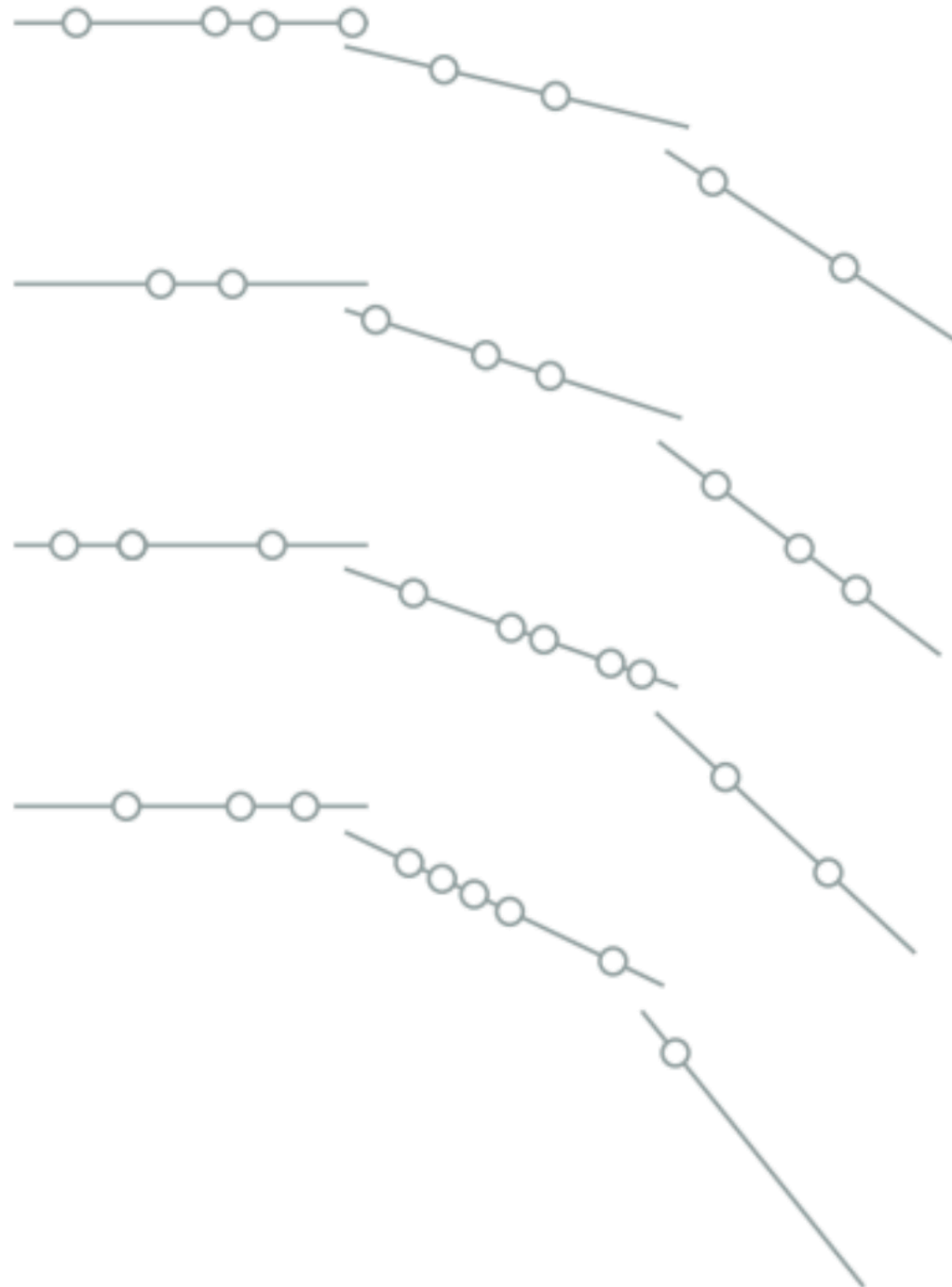
Material interactions

Particle interaction through matter:

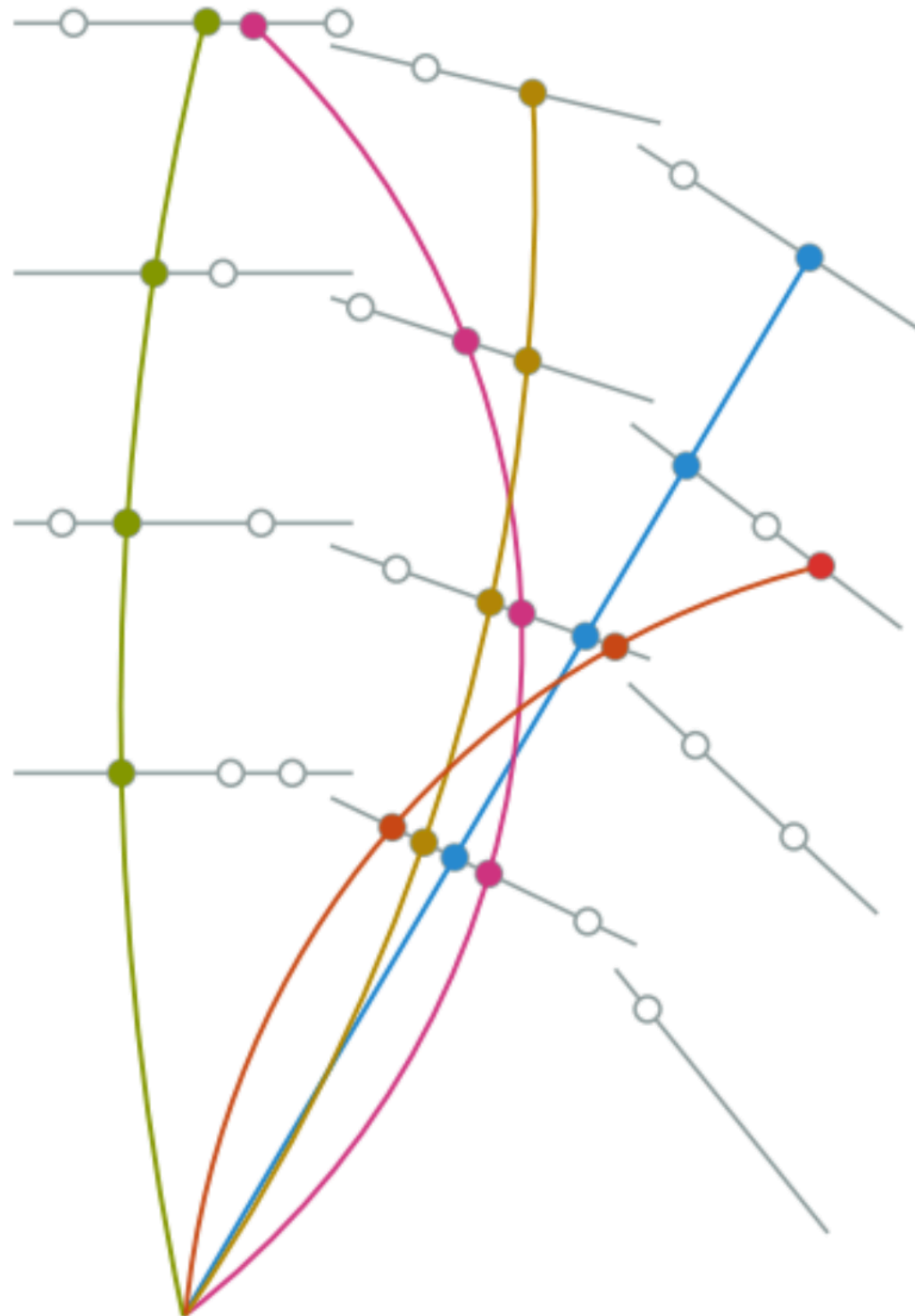
- Deviate the particle trajectory
- Effects inertly proportional to the momentum
- Almost Gaussian noise



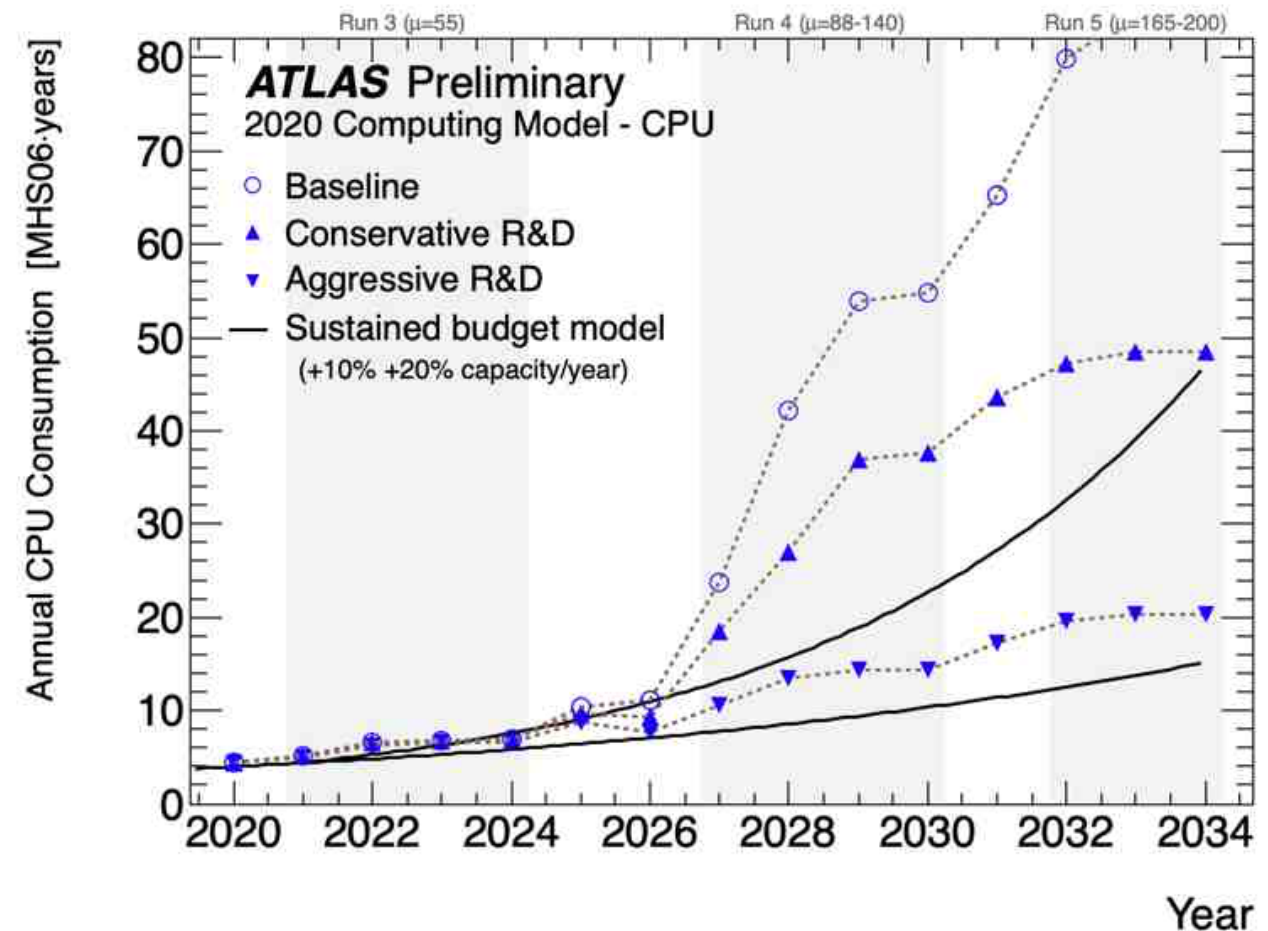
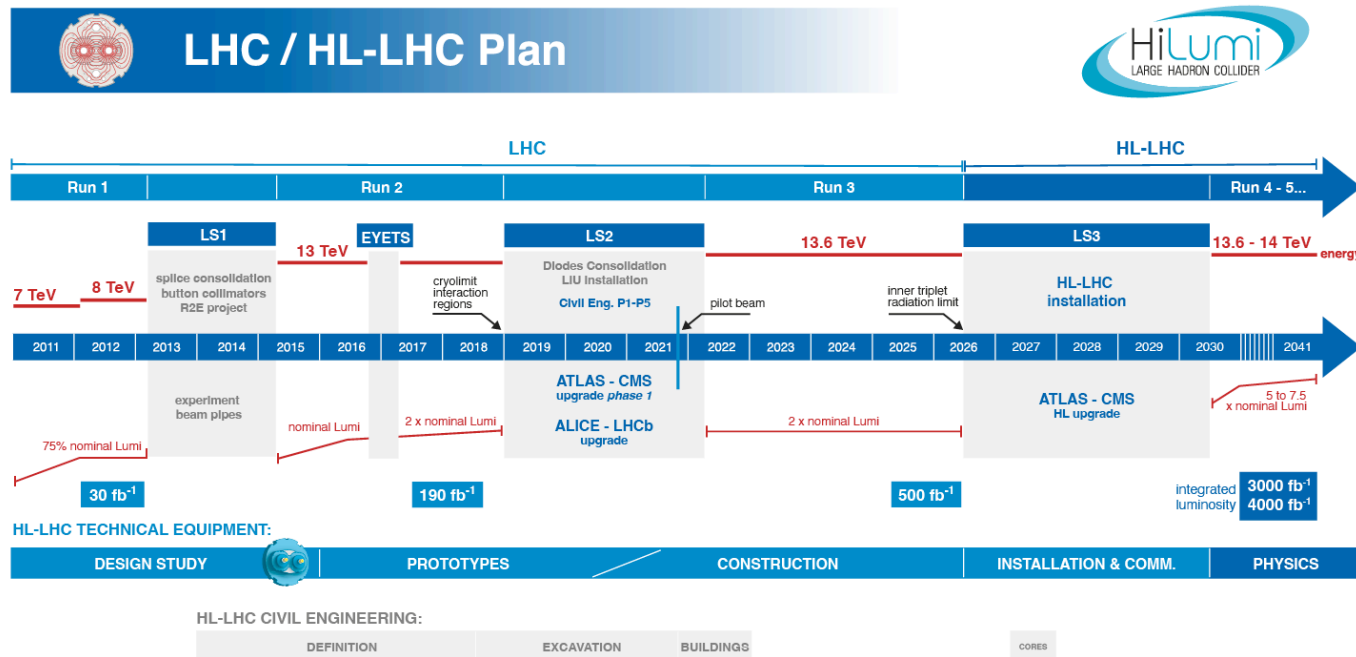
In Summary



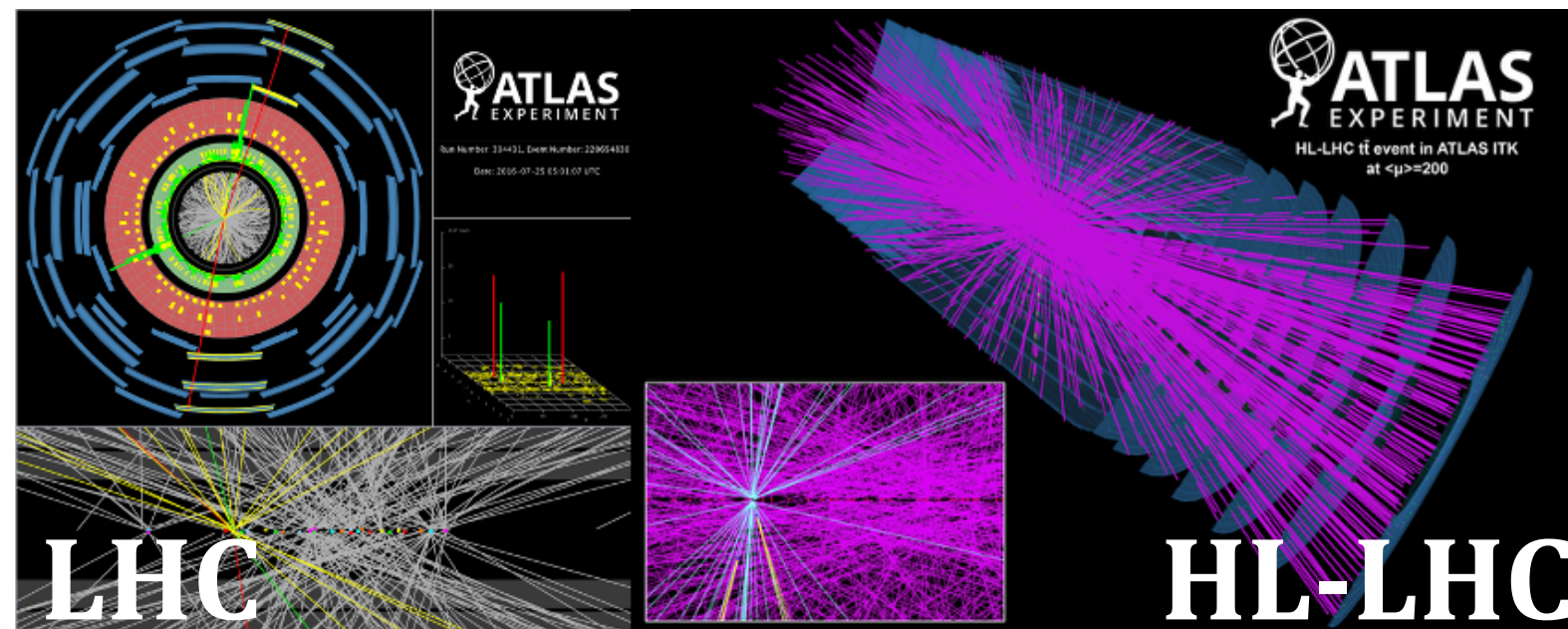
In Summary: Obviously !



Why use Deep Learning ?

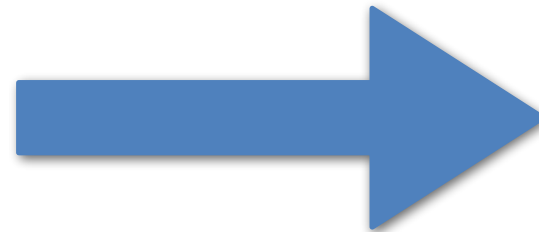
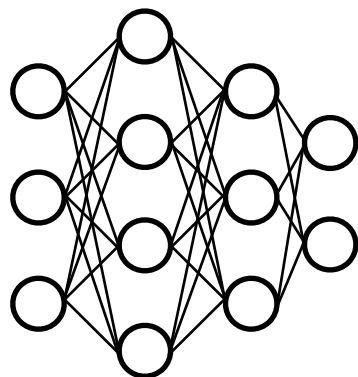
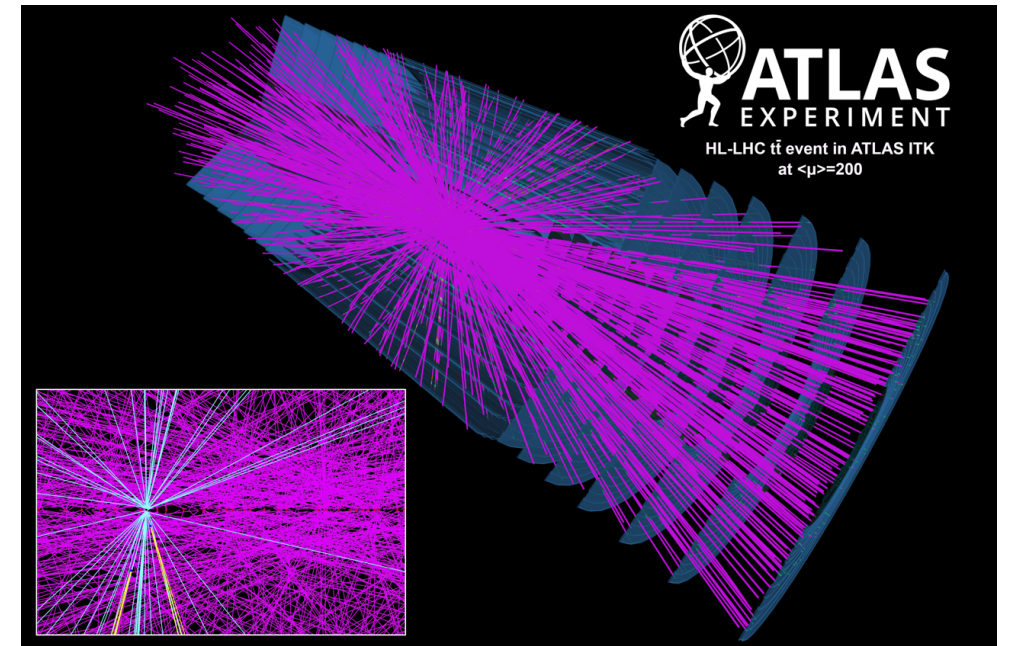


- Upgrade of the Large Hadron Collider: **HL-LHC**
- ~ **5x** more collisions per bunch crossing ➡ Reconstruction much more complicated
- Use DL for **more efficient** reconstruction



How to use DL for Tracking

- ~500k 3D points (x, y, z)
- Variable number of points
- Points only on sensor (non-contiguous)

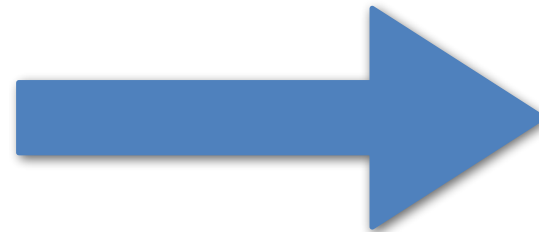
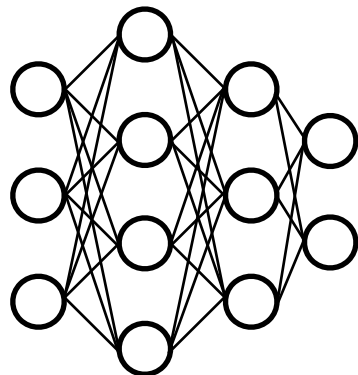
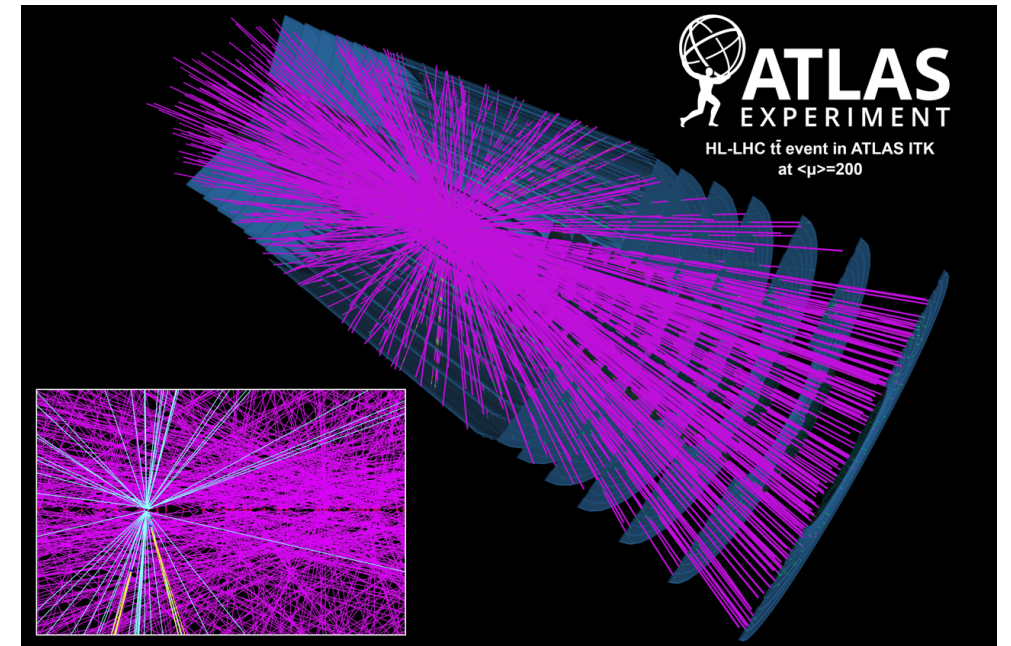


Dense Neural Network?

- Only work for fix input size
- Dependent on the input order

How to use DL for Tracking

- ~500k 3D points (x, y, z)
- Variable number of points
- Points only on sensor (non-contiguous)



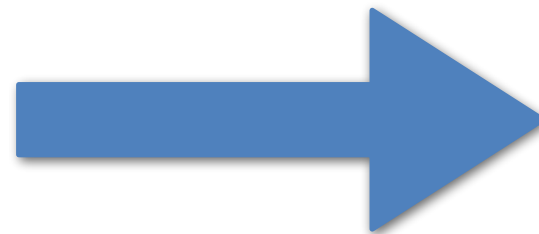
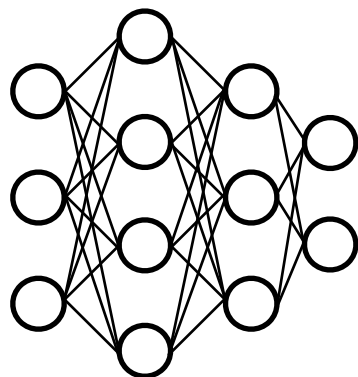
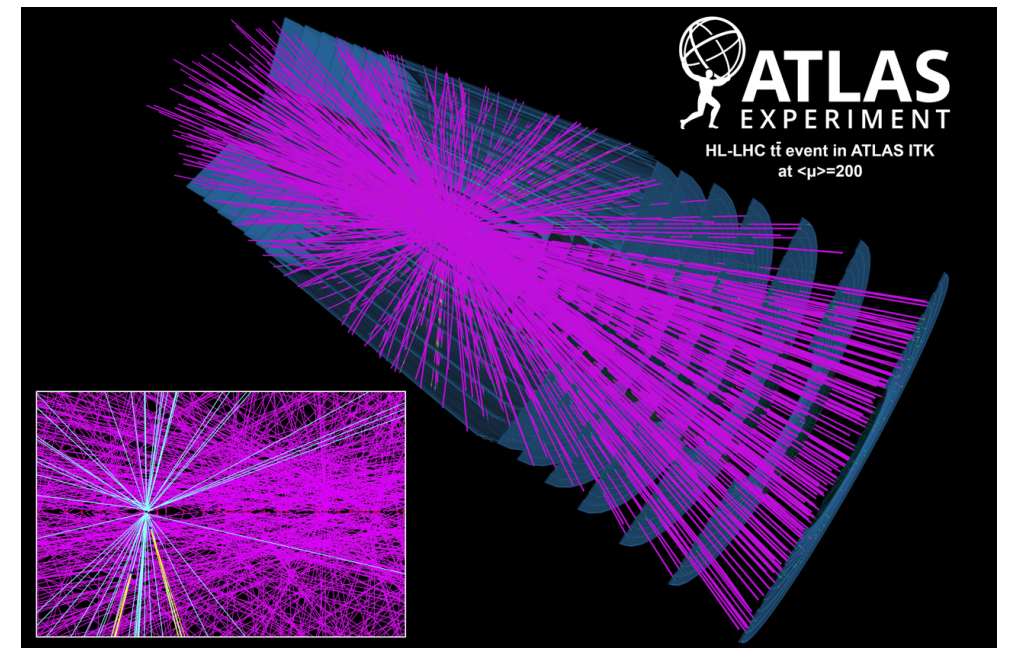
Dense Neural Network?

- Only work for fix input size
- Dependent on the input order



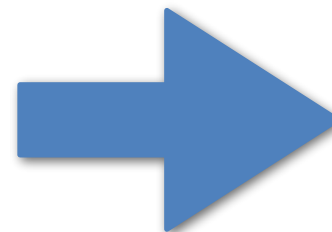
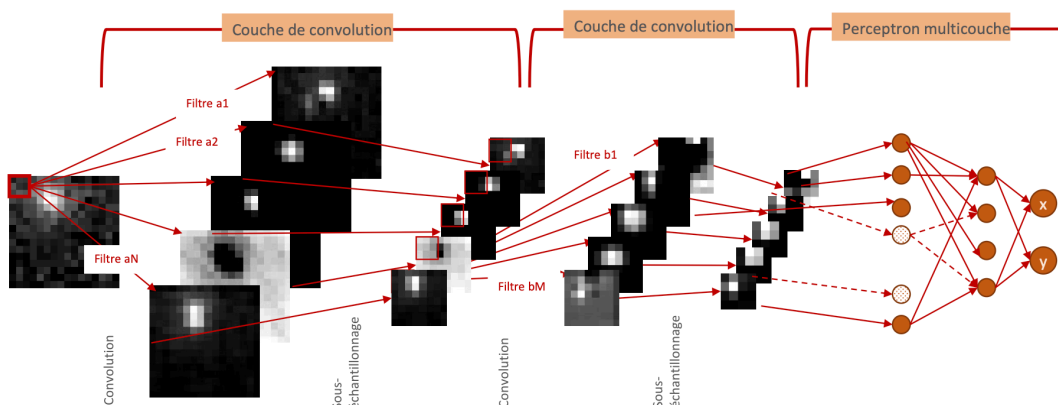
How to use DL for Tracking

- ~500k 3D points (x, y, z)
- Variable number of points
- Points only on sensor (non-contiguous)



Dense Neural Network?

- Only work for fix input size
- Dependent on the input order

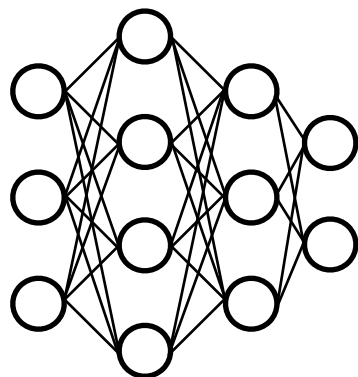
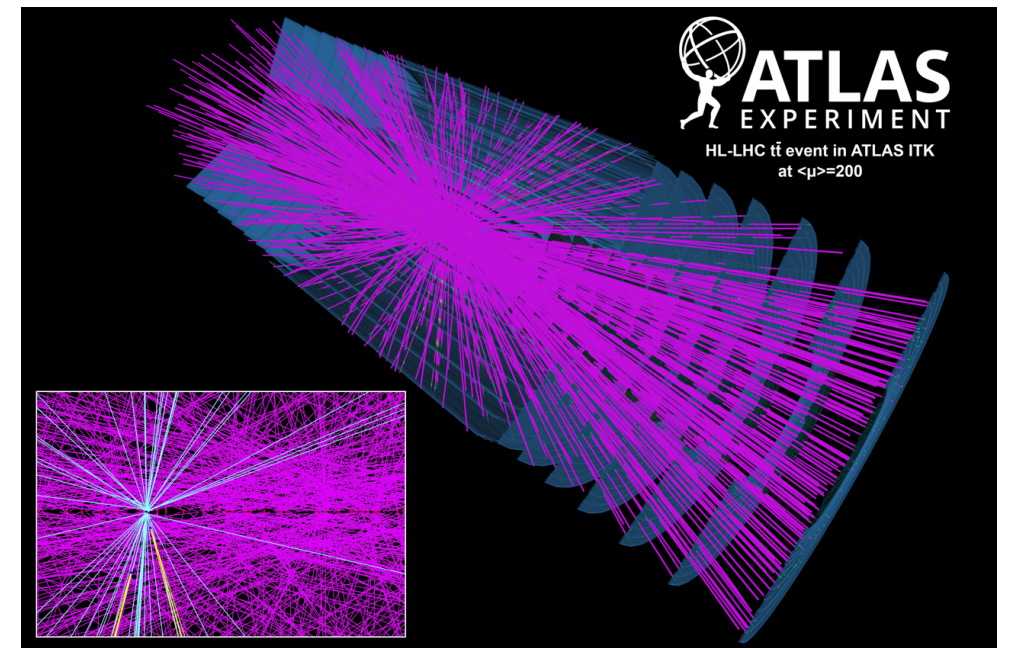


Convolutional NN?

- Tracker data $\neq$ Images
- Doesn't work well on sparse images

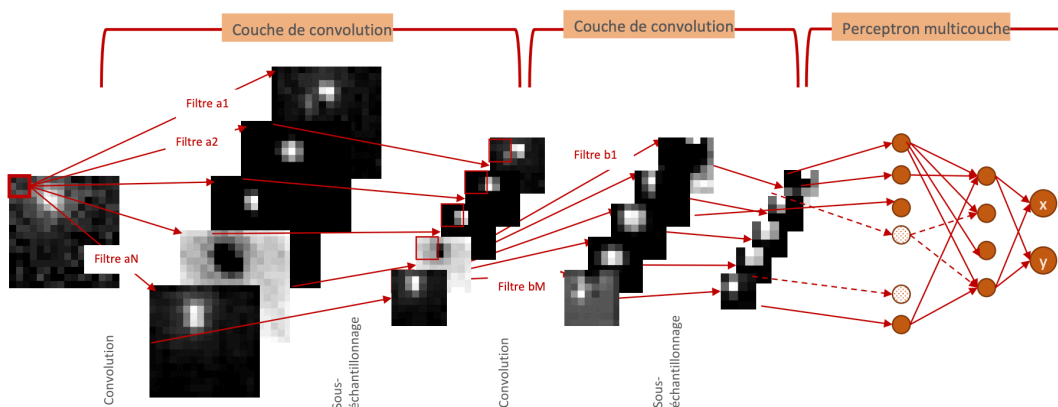
How to use DL for Tracking

- ~500k 3D points (x, y, z)
- Variable number of points
- Points only on sensor (non-contiguous)



Dense Neural Network?

- Only work for fix input size
- Dependent on the input order



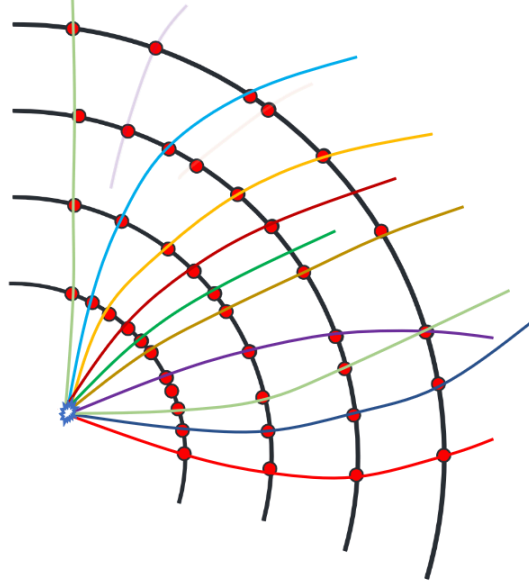
Convolutional NN?

- Tracker data $\neq$ Images
- Doesn't work well on sparse images

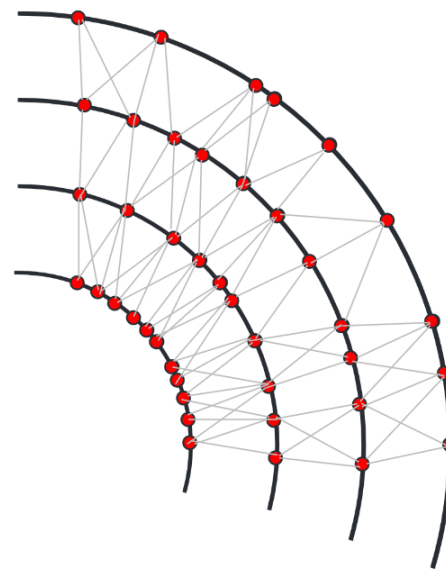


The GNN Revolution

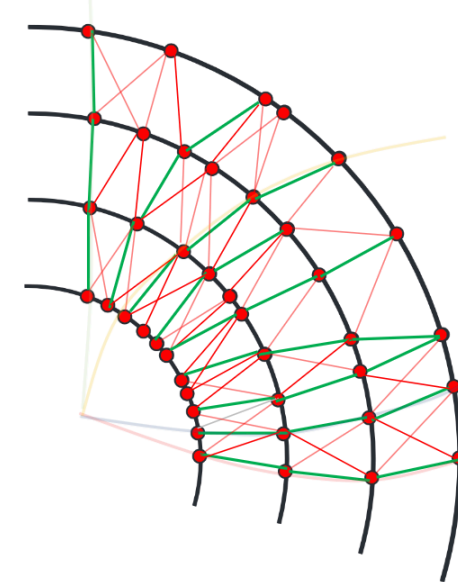
Charged particles leave hits in the detector



Represent the data using a graph



Goal:
classify the edges of the graph



High classification score

=> high probability that the edge is part of a track

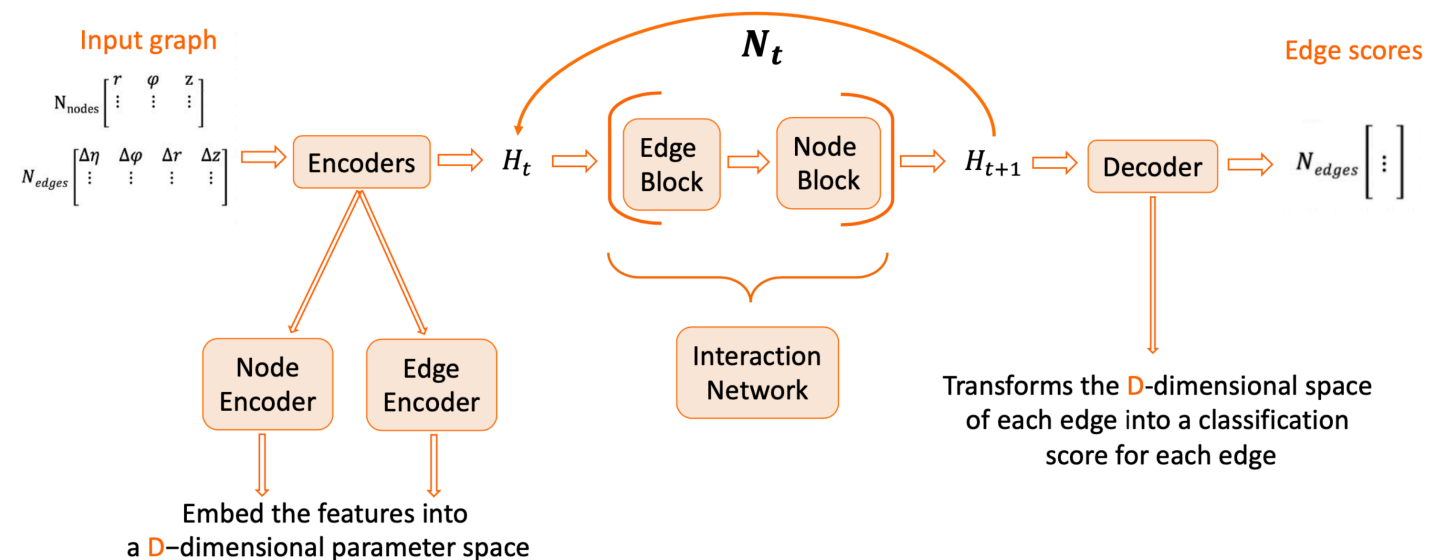
Low classification score
=> low probability that the edge is part of a track

One node of the graph = one hit in the detector

Connect two nodes using an edge if "it seems possible" that the two hits are two (consecutive) hits on a track

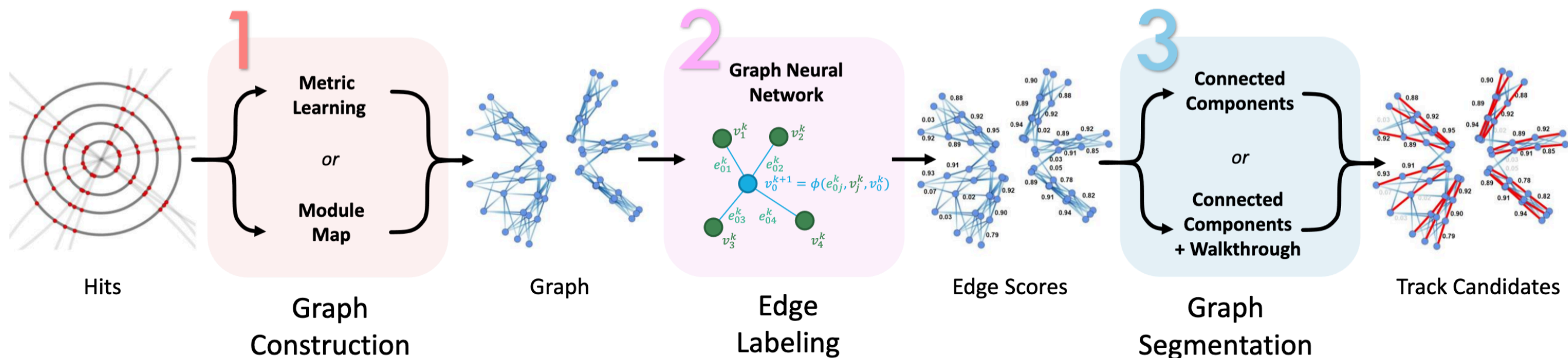
[Jan Stark](#)

- Graph representation natural for tracking problem
- Development started around 2020



GNN4ITk

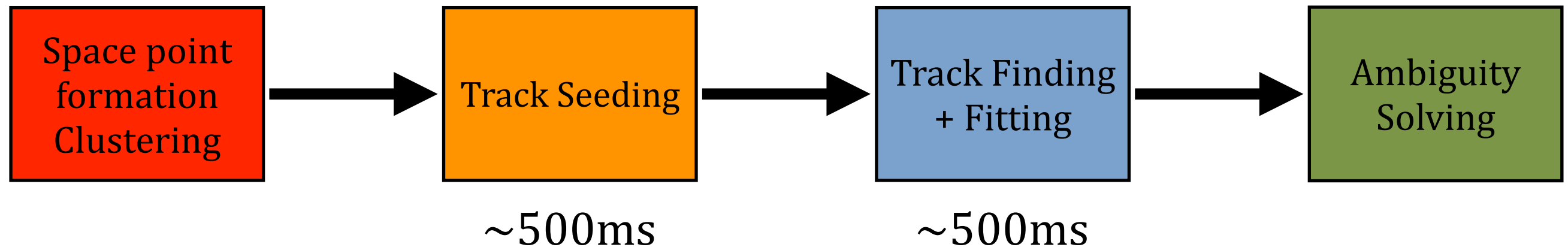
- Already exists for full tracking: [GNN4ITk](#)
- **Constructing** the graph can be complex
- Interactions between **distant nodes** might not be well captured
- Physics performance is almost on par with traditional methods, but too **compute intensive**
- The network needs to learn things that physicists **easily understand**



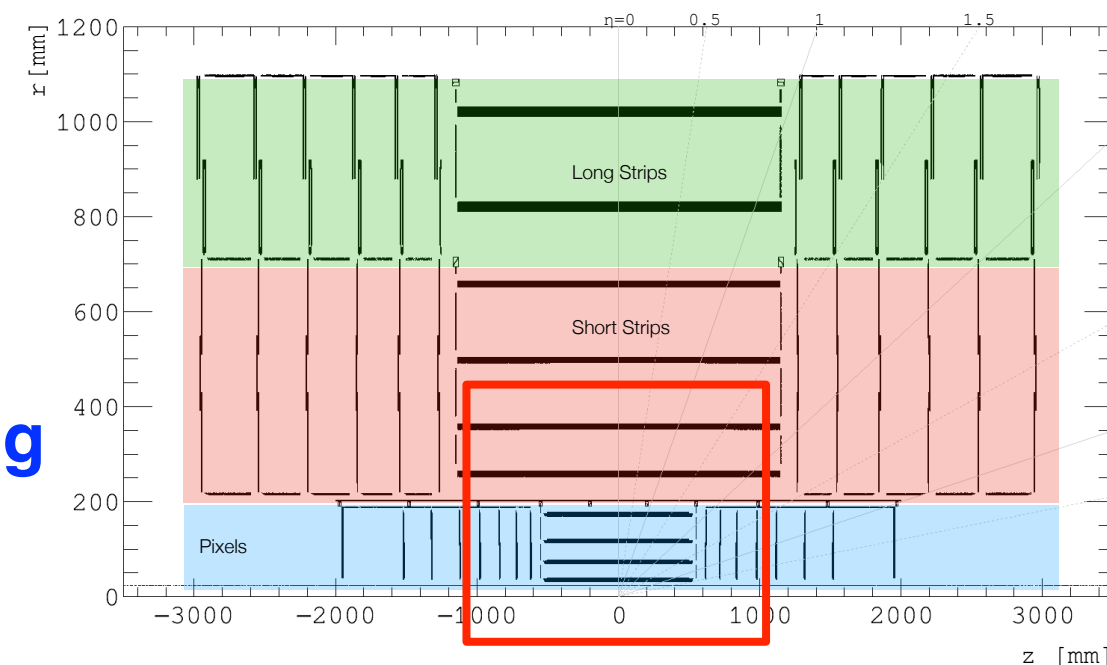
A transformer based SeedFinding algorithm

Generic
Unilateral
Neural
Track
Approximator
from **M**easurement

Why seed reconstruction?



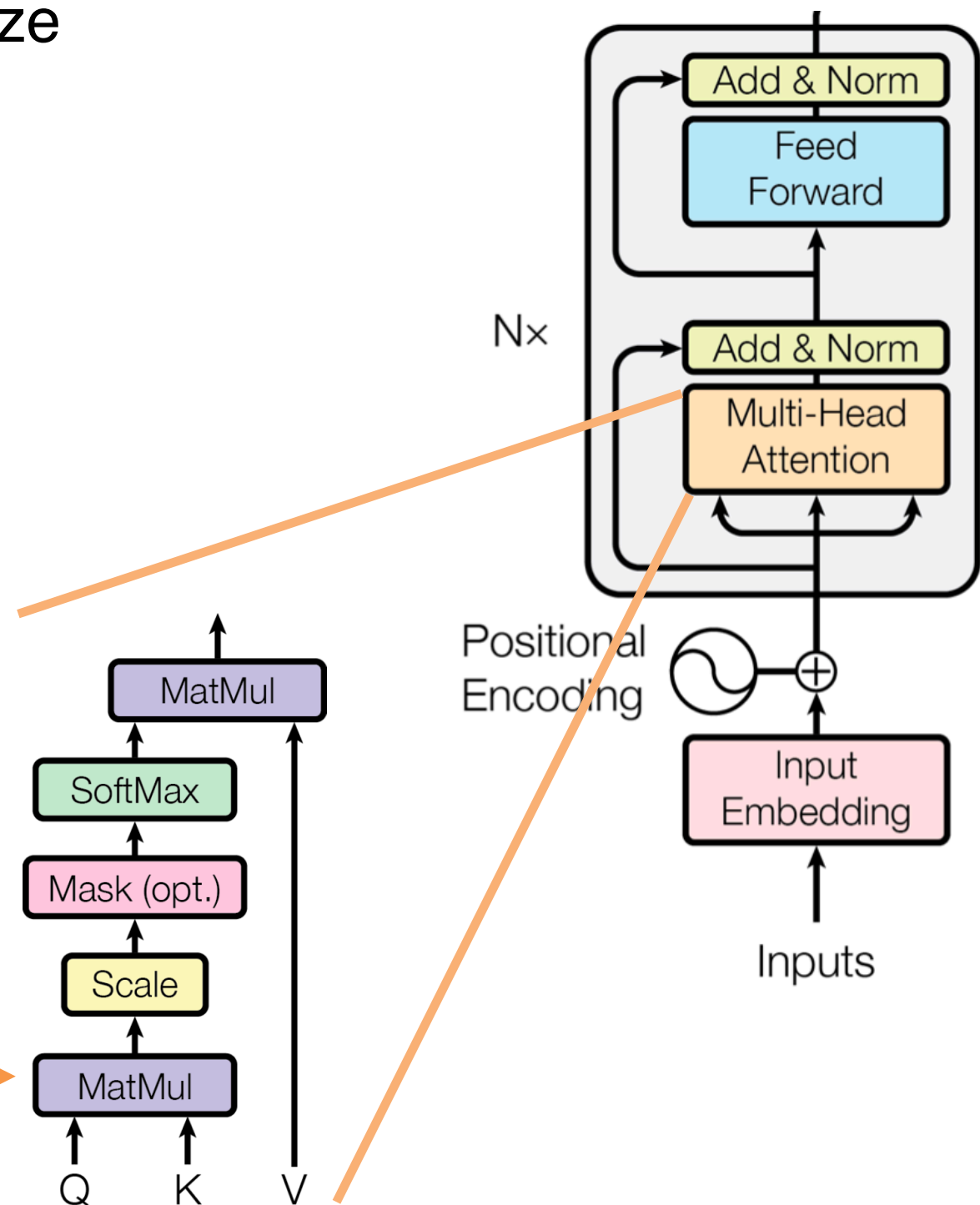
- **Seeding** and **Tracking** speed equivalent
- **Tracking** speed is highly correlated with **Seeding** performance:
 - More **Seed** (fake/duplicate) $\Rightarrow$ Slower **Tracking**
 - Missed **Seed** $\Rightarrow$ Missed **Track**
- CPU **Tracking** very efficient
- Can we replace the **Seeding** with a **Deep Learning alternative**?



Transformer

- **Transformer**: process set or arbitrary size
- Each element is encoded as a vector through the **embedding**
- **Attention mechanism**: compute input correlations
- **MLP** for updates of each entry

$$q_i = x_i \times W^Q$$
$$k_i = x_i \times W^K$$
$$v_i = x_i \times W^V$$
$$A = \begin{bmatrix} att_{11} & att_{12} & att_{13} & \dots & att_{1n} \\ att_{21} & att_{22} & att_{23} & \dots & att_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ att_{n1} & att_{n2} & att_{n3} & \dots & att_{nn} \end{bmatrix}$$
$$A = \frac{QK^T}{\sqrt{d_k}}$$



The Seeding Transformer

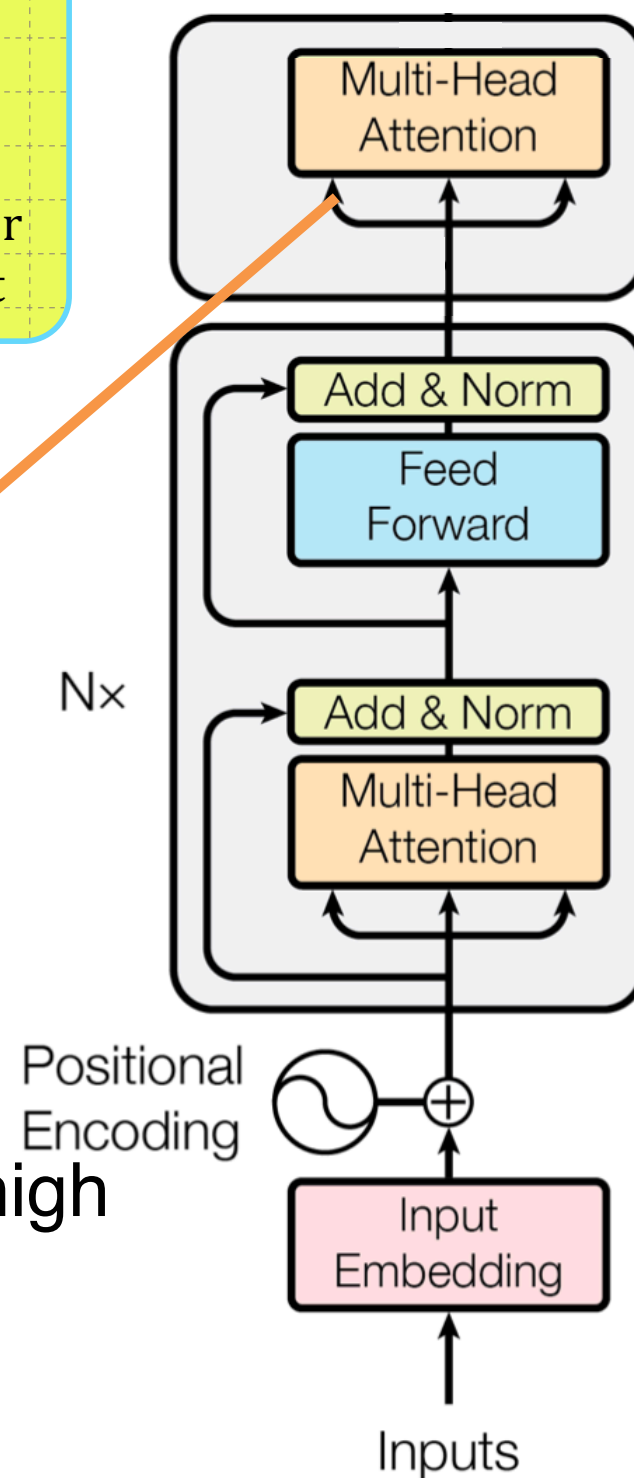
Transformer-based seed reconstruction library: **GUNTAM Seed**

- **Input:** Sets of hits
- Exploit the transformer **attention matrix** ➔ build a hit **adjacency matrix**
- Use a reconstruction algorithm to turn the **graph** into **seeds**

Generic
Unilateral
Neural
Track
Approximator
from **M**easurement

$$A = \begin{bmatrix} att_{11} & att_{12} & att_{13} & \dots & att_{1n} \\ att_{21} & att_{22} & att_{23} & \dots & att_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ att_{n1} & att_{n2} & att_{n3} & \dots & att_{nn} \end{bmatrix}$$

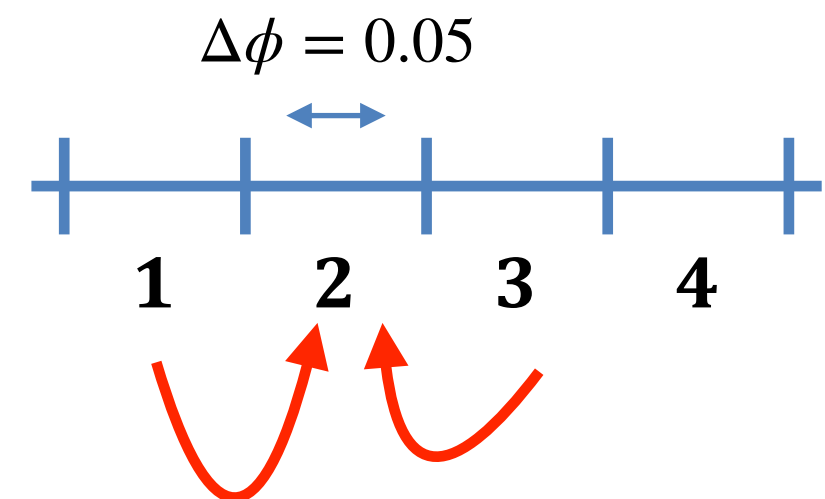
Uses specific loss to force high att_{ij} for **same particle hits**



Input Preprocessing

Transformers cannot handle **> 50000 hits** set:

- **Bin** the detector hits in phi:
 - $R_{max} = 200 \rightarrow \Delta\phi < 0.06$
 - $R_{max} = 500 \rightarrow \Delta\phi < 0.075$
 - Uses bin of $\Delta\phi = 0.05$
 - **Add neighbouring bin hits**
($\sim\Delta\phi = 0.15$ with $\Delta\phi = 0.05$ **overlap**)
 - More duplicates, but better efficiency
- The Transformer will run on **all bin in parallel**



$$1 \leftarrow 4, 1, 2 \quad 3 \leftarrow 2, 3, 4$$

$$2 \leftarrow 1, 2, 3 \quad 4 \leftarrow 3, 4, 1$$

$$\Delta\phi = 0.05 \Rightarrow 400 \times 3 \text{ hits}$$

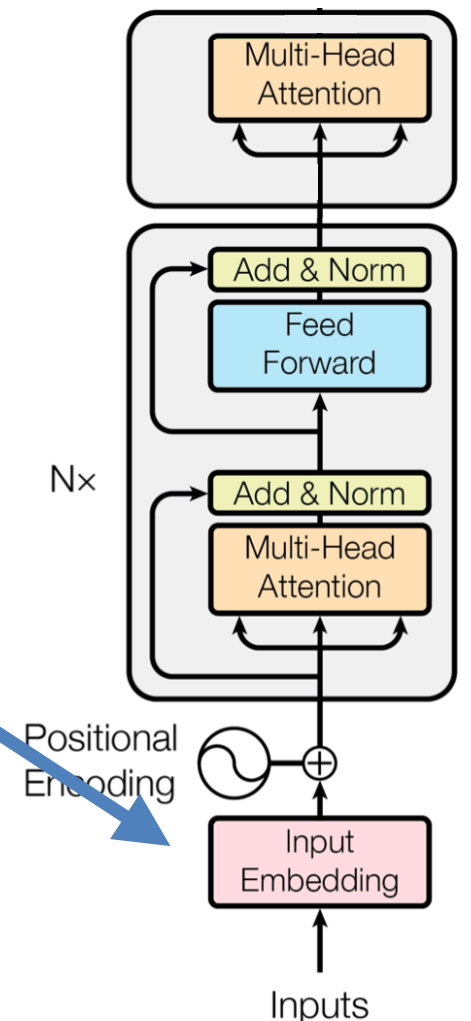
Hits Embedding

Our hits need to be transformed into expressive vectors $\mathbf{R}^{256}$:

- Input: (x, y, z, r, phi, eta)
- **Default solution: 2-layer MLP**
 - $(N_{hits}, 6) \rightarrow (N_{hits}, 256)$
 - Never worked for me
 - Max efficiency $\sim 20\%$ on simple cases
- Our Solution: **Fourier Embedding**

$$Emb(pos) = (\cos(2\pi \times pos), \sin(2\pi \times pos), \\ \cos(4\pi \times pos), \sin(4\pi \times pos), \\ \dots \\ \cos(2^k \pi \times pos), \sin(2^k \pi \times pos))$$

$$pos = \{x, y, z, r, \phi, \eta\}$$
$$k = 20$$

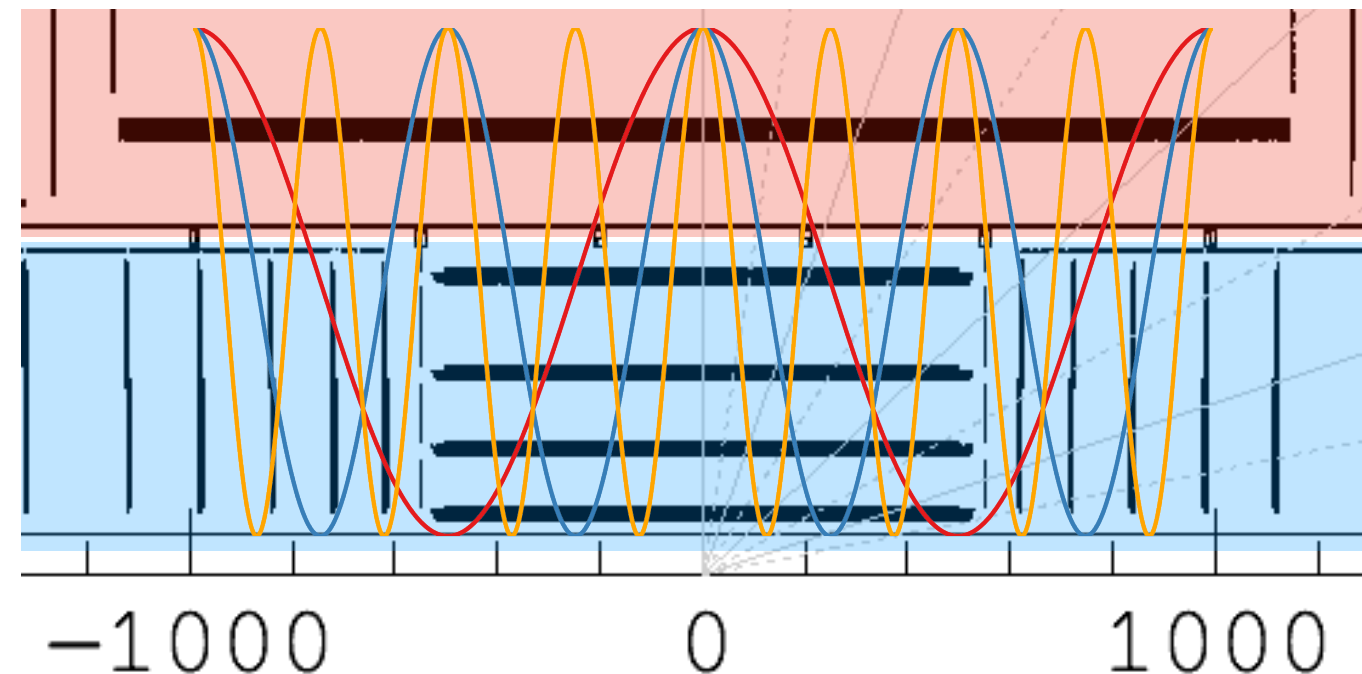


Hits Embedding

$$\text{Emb}(\text{pos}) = (\cos(2\pi \times \text{pos}), \sin(2\pi \times \text{pos}), \\ \cos(4\pi \times \text{pos}), \sin(4\pi \times \text{pos}), \\ \dots \\ \cos(2^k \pi \times \text{pos}), \sin(2^k \pi \times \text{pos}))$$

Fourier Embedding:

- Applied to the 3D position
- Encode the position with an **increasingly fine granularity**
- **Doubling** the frequency down to the pixel size



Fourier Embedding?

Fourier Embedding space:

- Attending to a **neighbour** can be achieved by a matrix multiplication
$$\begin{bmatrix} \cos(2^k \pi \times (pos - \Delta)) \\ \sin(2^k \pi \times (pos - \Delta)) \end{bmatrix} = \begin{bmatrix} \cos(2^k \pi \times \Delta) & \sin(2^k \pi \times \Delta) \\ -\sin(2^k \pi \times \Delta) & \cos(2^k \pi \times \Delta) \end{bmatrix} \cdot \begin{bmatrix} \cos(2^k \pi \times pos) \\ \sin(2^k \pi \times pos) \end{bmatrix}$$
- The product of two vectors only depend on their **distance** in real space
$$K \cdot Q^T = \sum_k [\cos(2^k \pi \times \Delta)]$$

$$\Delta < \frac{1}{2^k}; K \cdot Q^T \approx k$$

$$\Delta > \frac{1}{2^k}; K \cdot Q^T \approx 0$$

$$K \cdot Q^T = [1 + 0.99 + 0.97 + 0.88 + 0.53 - 0.42 - 0.64 - 0.15 - 0.93]$$

$$\Delta = 0.01 \rightarrow \Delta Z = 10mm$$

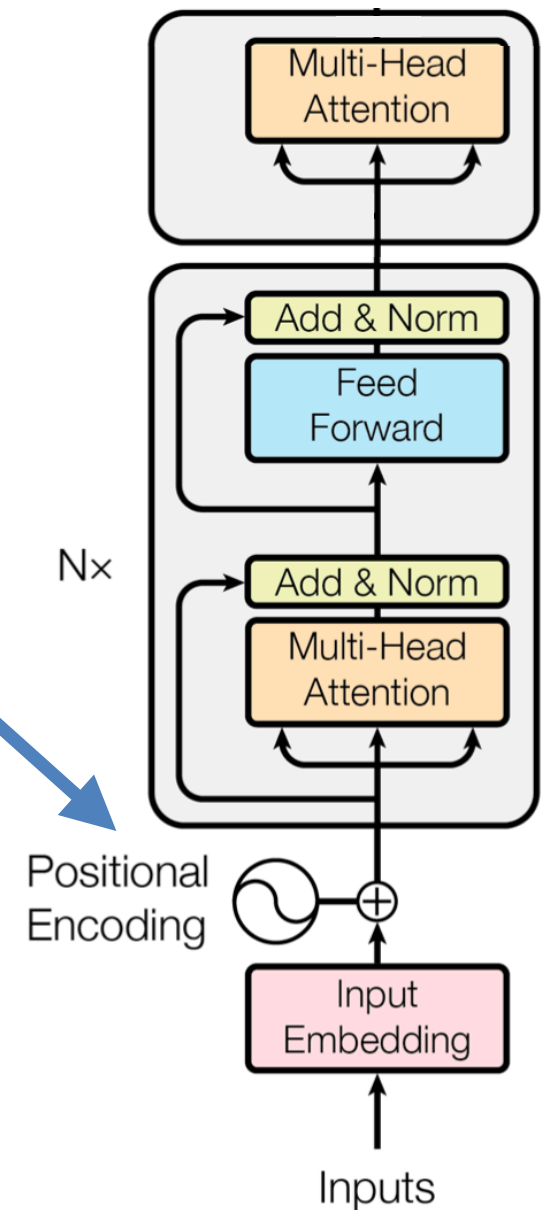
Positional encoding POV

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

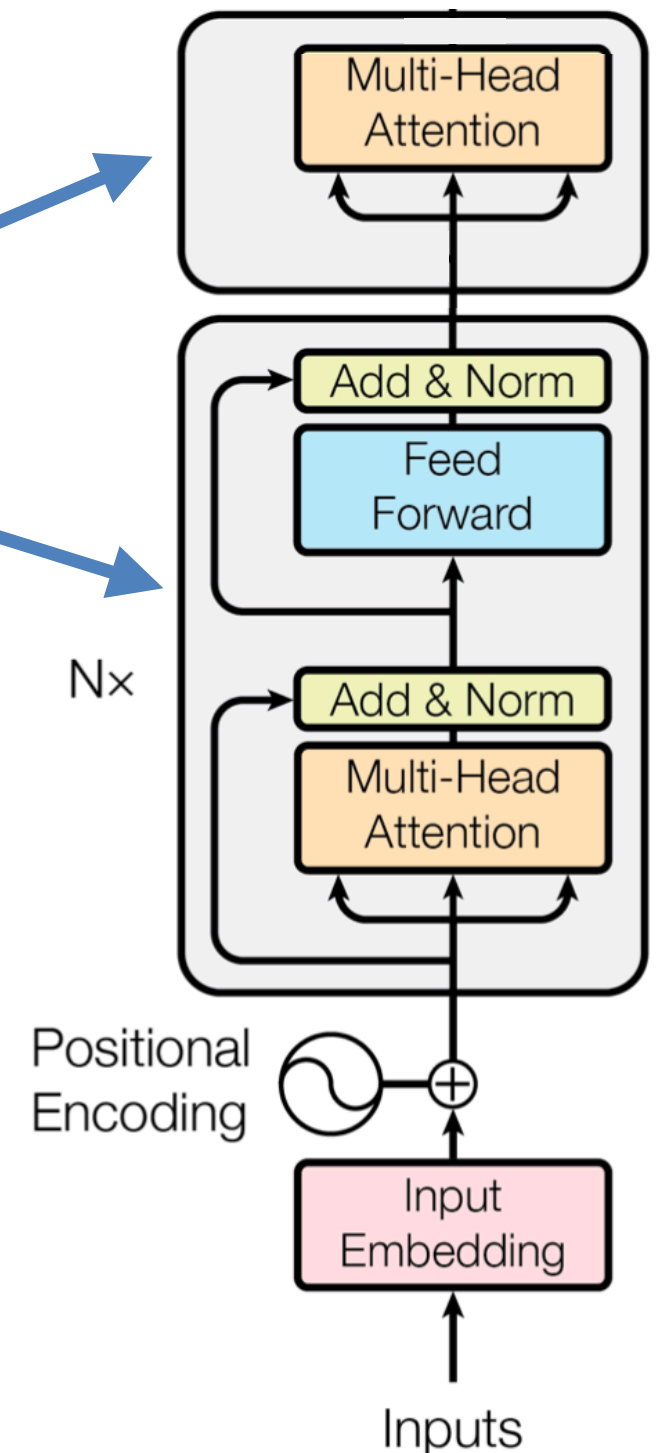
Attention Is All You Need

- Used in language transformers to **encode positions** in the input sentence
- Encode **1D position** to a vector of **cos and sin**
- Vector then added to the Input Embedding
- **Fourier Embedding** ~ Same embedding for all hits with **3D positional encoding**



Seeding Transformer's architecture

- Hits binning
- **Fourier Embedding**
- 4+1 transformer layers:
 - 4 with **FlashAttention** and 4 heads
 - 1 without and with only 1 head; Used in **hit pair** reconstruction
- ~5M parameters
- Training on A100 GPU
- **One week** of training:
 - Short compare to LLM
 - Long compared to usual physics networks



Reconstruction algorithm

Reconstruction modes using the **last attention matrix**:

- « **Connecting the dots** » :
 - For each hit, try to predict the **next** one from the same particle
 - Use the **attention score**
 - **Cross-entropy** on the matrix line
➡ target: next **same particle hit**

	1	2	3	4	5	6	7	8	9	10
1	-inf	0	0	0.9	0.05	0				
2	0	-inf	0.7	0	0					
3			-inf		0	0.85	0	0		
4				-inf	0	0	0	0	0.8	0
5					-inf			0.7		
6					0	-inf	0.98	0		
7					0	0.98	-inf	0	0	
8				0				-inf		0.87
9			0	0.7	0	0		0	-inf	0
10							0	0.8		-inf

Seeds: **[1]**

Reconstruction algorithm

Reconstruction modes using the **last attention matrix**:

- « **Connecting the dots** » :
 - For each hit, try to predict the **next** one from the same particle
 - Use the **attention score**
 - **Cross-entropy** on the matrix line
➡ target: next **same particle hit**

	1	2	3	4	5	6	7	8	9	10
1	-inf	0	0	0.9	0.05	0				
2	0	-inf	0.7	0	0					
3			-inf		0	0.85	0	0		
4				-inf	0	0	0	0	0.8	0
5					-inf			0.7		
6					0	-inf	0.98	0		
7					0	0.98	-inf	0	0	
8				0				-inf		0.87
9			0	0.7	0	0		0	-inf	0
10							0	0.8		-inf

Seeds: [1,4]

Reconstruction algorithm

Reconstruction modes using the **last attention matrix**:

- « **Connecting the dots** » :
 - For each hit, try to predict the **next** one from the same particle
 - Use the **attention score**
 - **Cross-entropy** on the matrix line
➡ target: next **same particle hit**
 - Stop after **5 hits** or if there is no future hit above a **threshold**
 - For the last hit: use the previous one

	1	2	3	4	5	6	7	8	9	10
1	-inf	0	0	0.9	0.05	0				
2	0	-inf	0.7	0	0					
3			-inf		0	0.85	0	0		
4				-inf	0	0	0	0	0.8	0
5					-inf			0.7		
6					0	-inf	0.98	0		
7					0	0.98	-inf	0	0	
8				0				-inf		0.87
9			0	0.7	0	0		0	-inf	0
10							0	0.8		-inf

Seeds: **[1,4,9]**

Reconstruction algorithm

Reconstruction modes using the **last attention matrix**:

- « **Connecting the dots** » :
 - For each hit, try to predict the **next** one from the same particle
 - Use the **attention score**
 - **Cross-entropy** on the matrix line
➡ target: next **same particle hit**
 - Stop after **5 hits** or if there is no future hit above a **threshold**
 - For the last hit: use the previous one

	1	2	3	4	5	6	7	8	9	10
1	-inf	0	0	0.9	0.05	0				
2	0	-inf	0.7	0	0					
3			-inf		0	0.85	0	0		
4				-inf	0	0	0	0	0.8	0
5					-inf			0.7		
6					0	-inf	0.98	0		
7					0	0.98	-inf	0	0	
8				0				-inf		0.87
9			0	0.7	0	0		0	-inf	0
10							0	0.8		-inf

Seeds: [1,4,9] ; [2,3,6,7]

Reconstruction algorithm

Reconstruction modes using the **last attention matrix**:

- « **Connecting the dots** » :
 - For each hit, try to predict the **next** one from the same particle
 - Use the **attention score**
 - **Cross-entropy** on the matrix line
➡ target: next **same particle hit**
 - Stop after **5 hits** or if there is no future hit above a **threshold**
 - For the last hit: use the previous one
- In reality: **beam search**

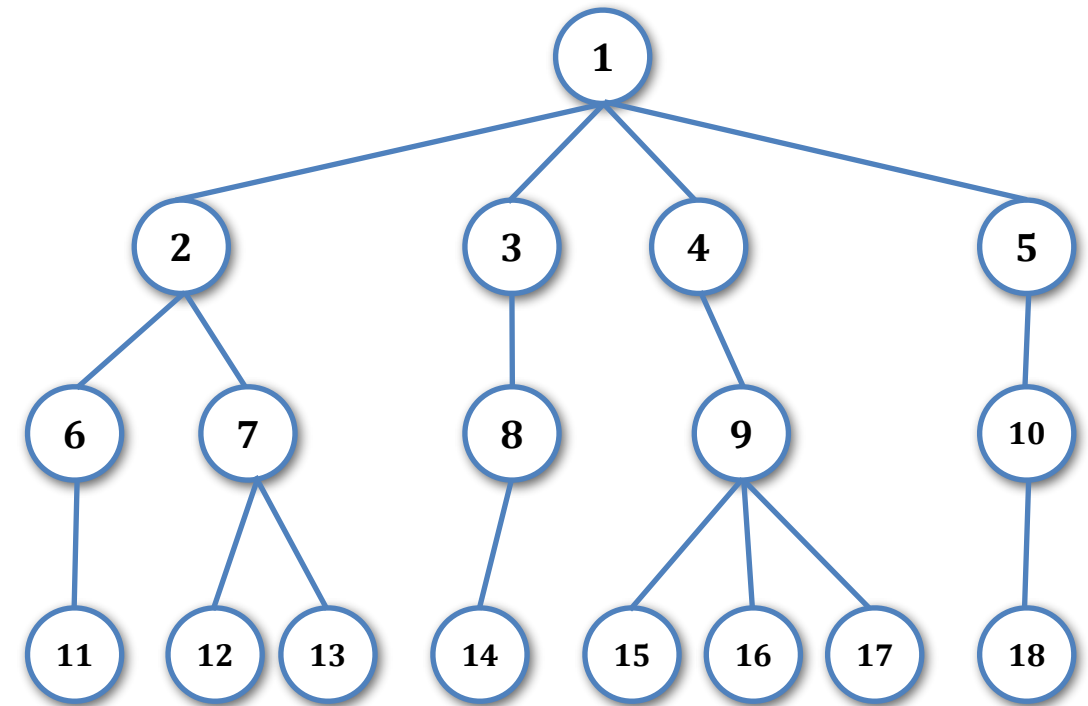
	1	2	3	4	5	6	7	8	9	10
1	-inf	0	0	0.9	0.05	0				
2	0	-inf	0.7	0	0					
3			-inf		0	0.85	0	0		
4				-inf	0	0	0	0	0.8	0
5					-inf			0.7		
6					0	-inf	0.98	0		
7					0	0.98	-inf	0	0	
8				0				-inf		0.87
9			0	0.7	0	0		0	-inf	0
10							0	0.8		-inf

Seeds: [1,4,9] ; [2,3,6,7] ; [5,8,10]

Beam Search

Tree-based explorations

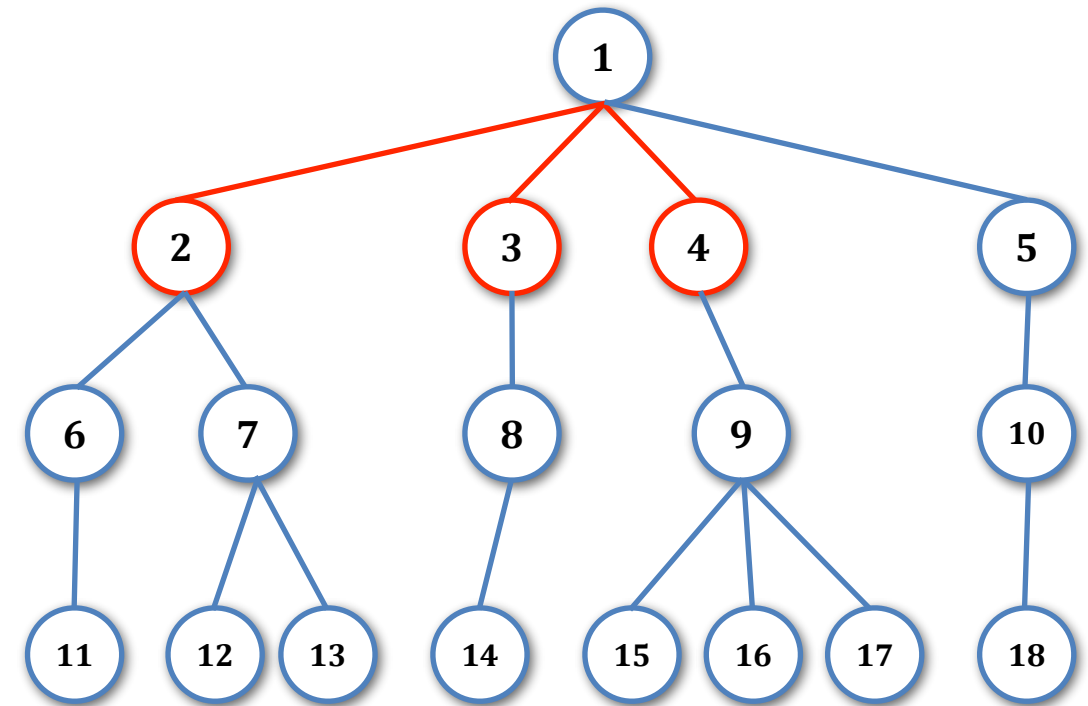
- Find the **k best nodes** based on a score



Beam Search

Tree-based explorations

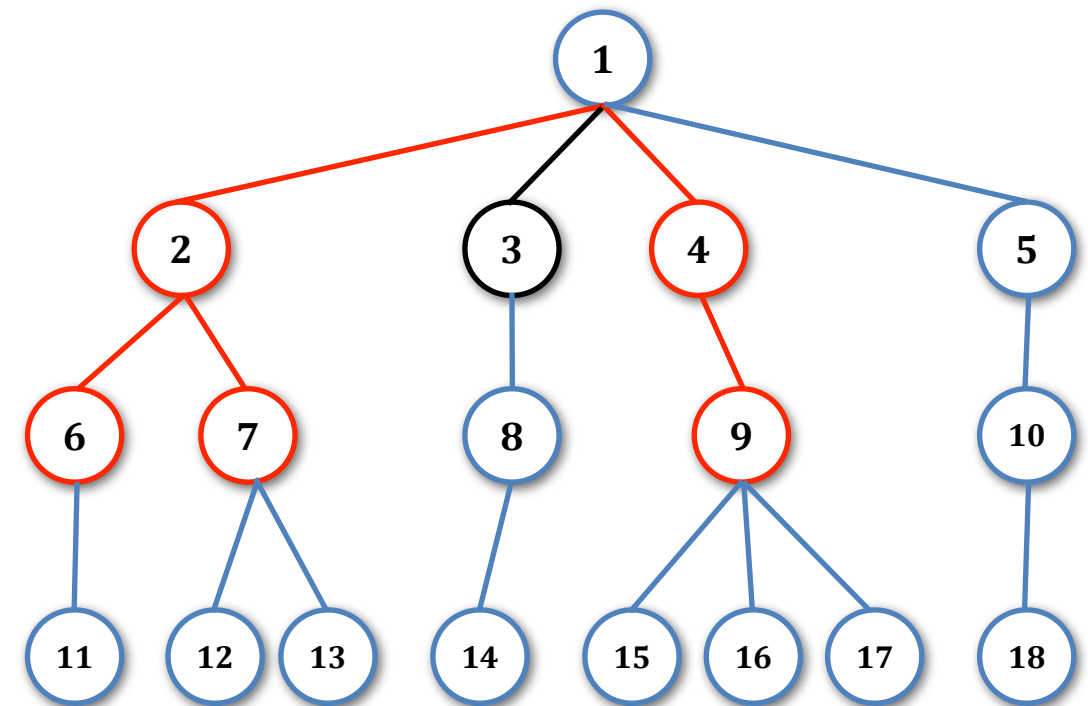
- Find the **k best nodes** based on a score
- Repeat for each layer



Beam Search

Tree-based explorations

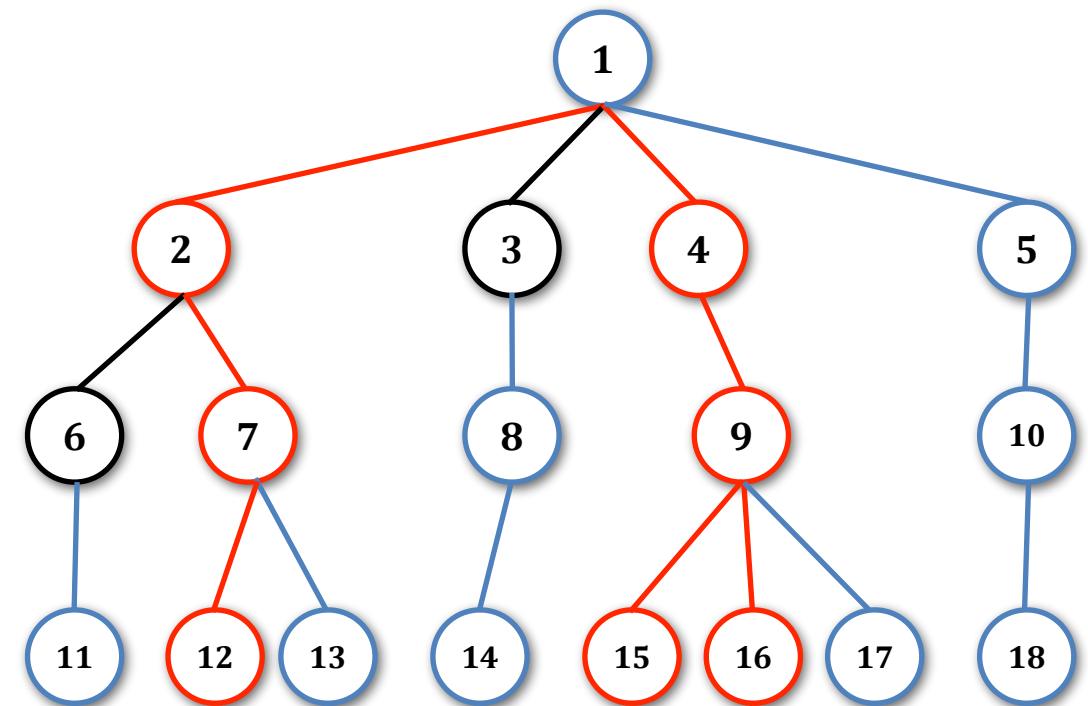
- Find the **k best nodes** based on a score
- Repeat for each layer
- Continue until the **end** of the tree or a **max size**



Beam Search

Tree-based explorations

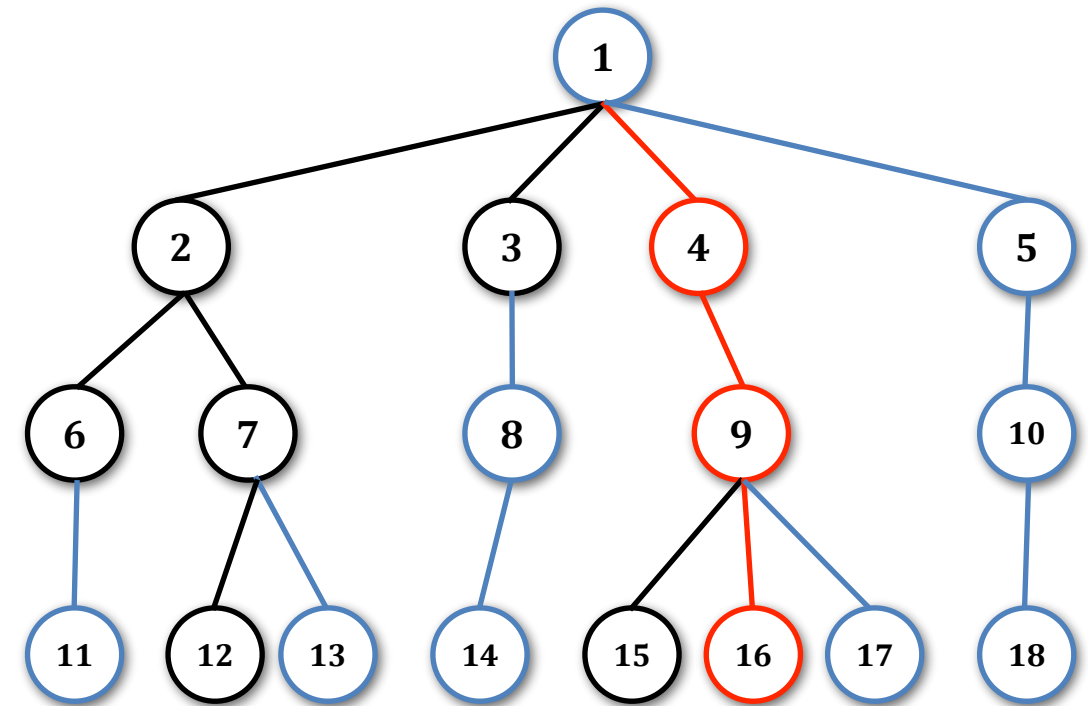
- Find the **k best nodes** based on a score
- Repeat for each layer
- Continue until the **end** of the tree or a **max size**
- Finally, select the **best path**



Beam Search

Tree-based explorations

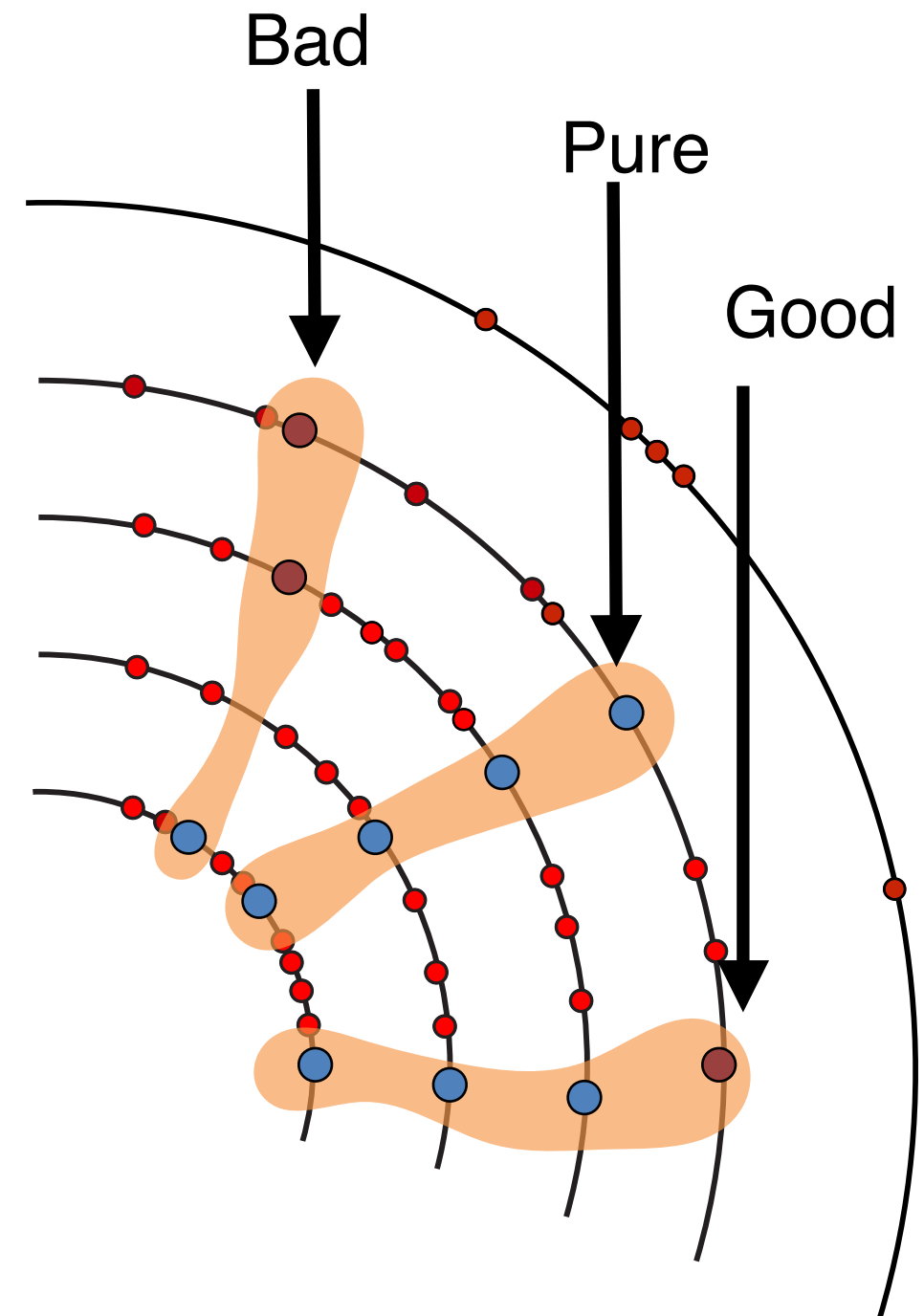
- Find the **k best nodes** based on a score
- Repeat for each layer
- Continue until the **end** of the tree or a **max size**
- Finally, select the **best path**
- Here: 1,4,9,16
- Use PyTorch **topk** to perform the search on the **GPU with the tensors**



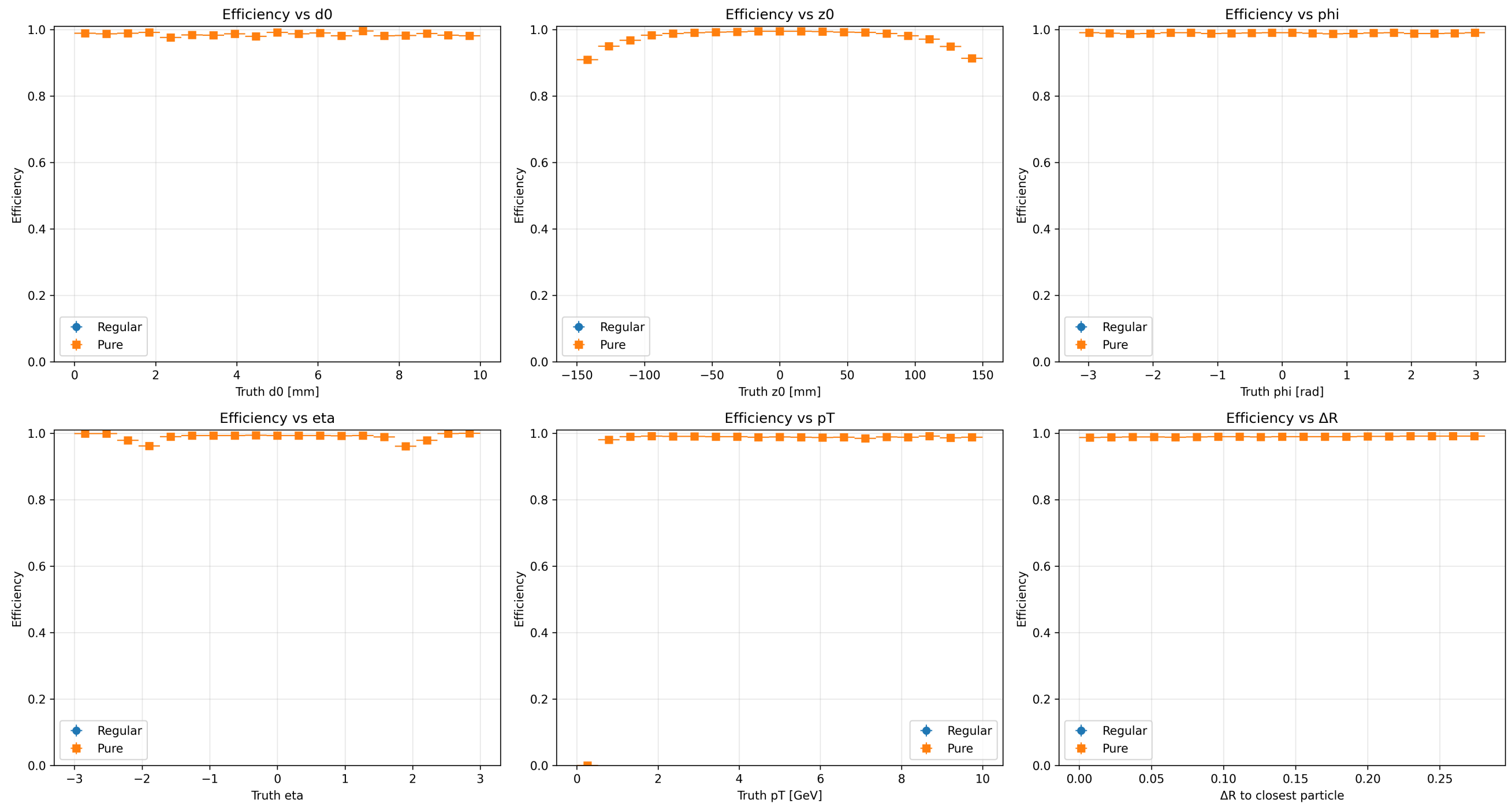
Performance Metrics

How do we evaluate the performance of the seeding?

- Now: 3-5 hits seeds, **associated particle** for each hit
- **Seed-particle match**: ≥ 3 hits from the particle
- **Pure match**: ≥ 3 hits all from the same particle
- Currently only looking at Seed with **exactly 3 hits**



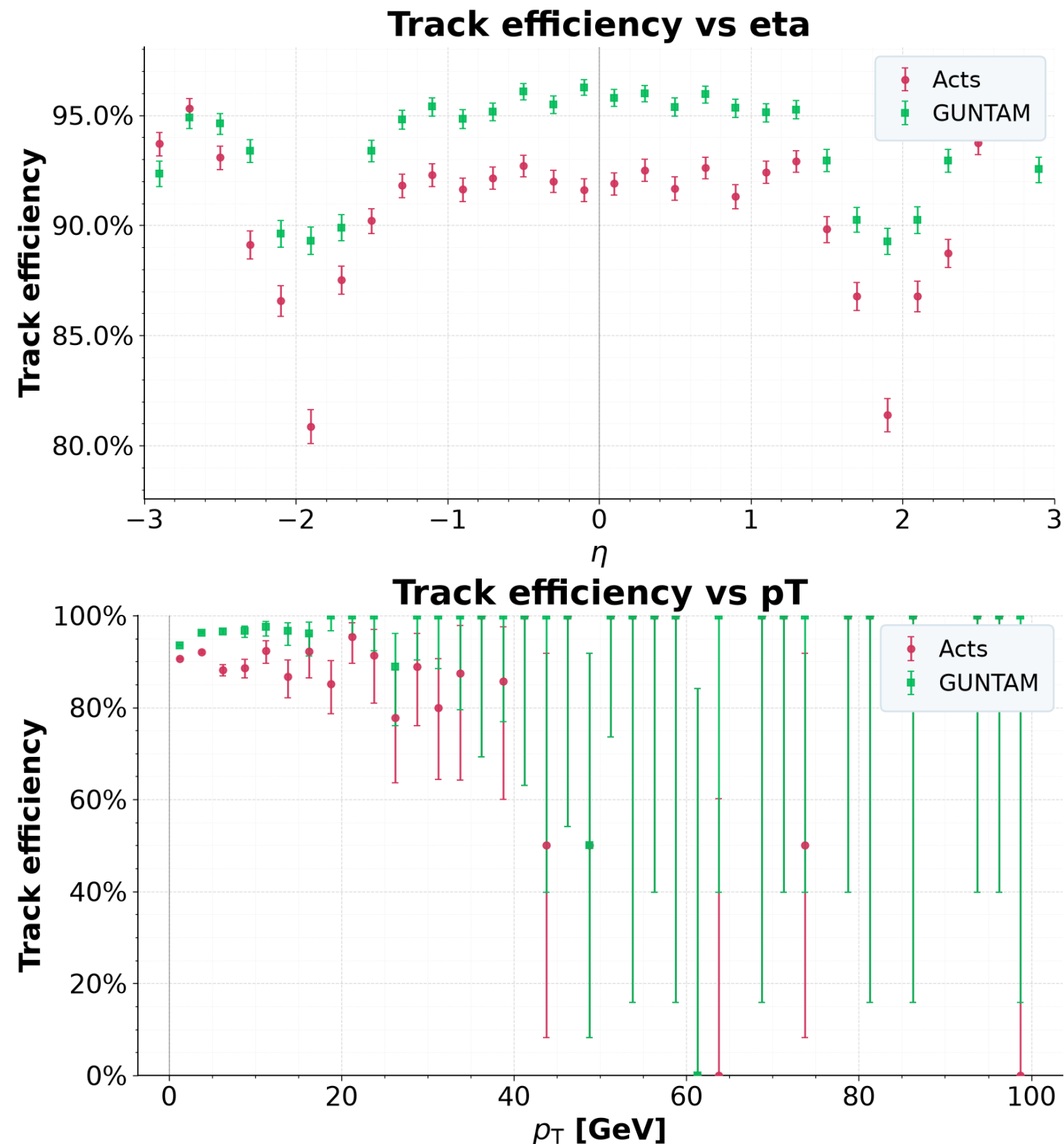
Results: Seed efficiency



- Efficiency: **99%** (3 hits from the same particles)

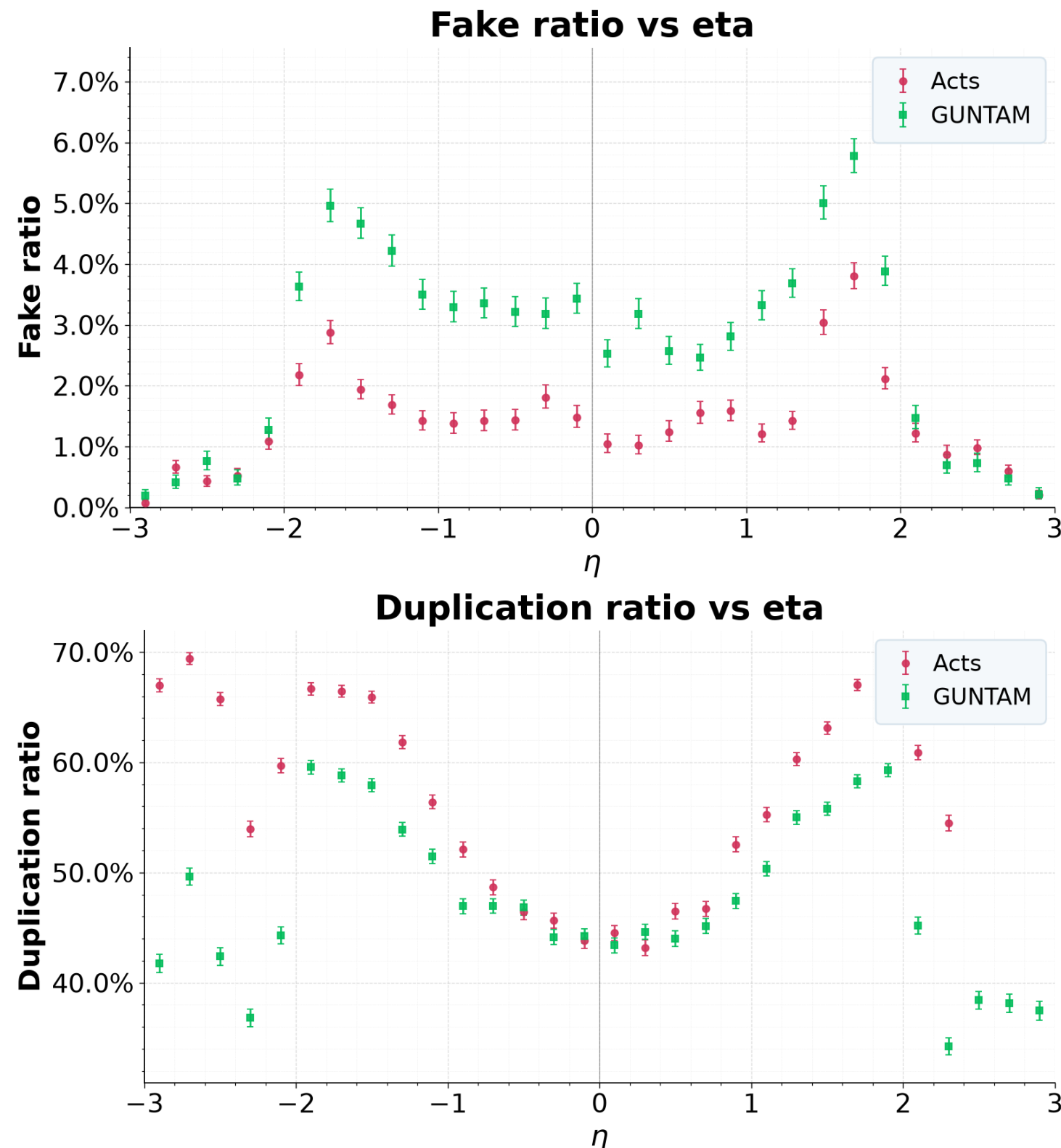
Results: Comparison with ACTS

- Comparison with ACTS (Open-sourced tracking library)
- Performed on the Open Data Detector
- Improve tracking efficiency in the eta gap
- Performance improvement over the entire pT range



Results: Comparison with ACTS

- Increase in the fake rate, too many fake seed
- Plan to add a Classifier after the Transformer to remove bad seeds
- Less duplicate tracks (track reconstructed multiple times)



Results

- **~3x speed up** can be achieved compared to classical seeding on CPU
- Using **float32** right now an extra **2x** can be achieved with float16 (requires retraining, out of the box: 45% efficiency)
- Investigating the use of PyTorch Autocast to improve memory and speed

	CPU (Macbook M4)	GPU (A3000)	GPU (A100)
Transformer Inference [ms/event]	2264	555	116
Beam search Reconstruction [ms/event]	108	64	16
Total [ms/event]	2372	619	132

Conclusion

- Transformers perform well in trajectory reconstruction
- Fourier-based position embedding allows the use of continuous positional input
- The attention matrix of the transformer can be turned into a graph/adjacency prediction
- **GUNTAM Seed**: open source library available online
<https://github.com/Corentin-Allaire/GUNTAM>

BACKUP

Generic
Unilateral
Neural
Track
Approximator
from **M**easurement

Track reconstruction: Kalman Filter

System equation:

$$\mathbf{x}_{k,t} = \mathbf{F}_{k-1} \mathbf{x}_{k-1,t} + \mathbf{w}_{k-1}. \quad (5)$$

Measurement equation:

$$\mathbf{m}_k = \mathbf{H}_k \mathbf{x}_{k,t} + \epsilon_k, \quad (6)$$

$$E\{\mathbf{w}_k\} = 0, \text{cov}\{\mathbf{w}_k\} = \mathbf{Q}_k,$$

$$E\{\epsilon_k\} = 0, \text{cov}\{\epsilon_k\} = \mathbf{V}_k = \mathbf{G}_k^{-1},$$

with

$\mathbf{x}_{k,t}$ = true value of the state vector at time k ;

$\mathbf{x}_k^i$ = estimate of $\mathbf{x}_{k,t}$, using measurements up to time i
($i < k$: prediction, $i = k$: filtered estimate, $i > k$:
smoothed estimate), $\mathbf{x}_k^k$ is simply written as $\mathbf{x}_k$;

$$\mathbf{C}_k^i = \text{cov}\{\mathbf{x}_k^i - \mathbf{x}_{k,t}\};$$

$$\mathbf{r}_k^i = \text{residual } \mathbf{m}_k - \mathbf{H}_k \mathbf{x}_k^i;$$

$$\mathbf{R}_k^i = \text{cov}\{\mathbf{r}_k^i\}.$$

Prediction:

Extrapolation of the state vector:

$$\mathbf{x}_k^{k-1} = \mathbf{F}_{k-1} \mathbf{x}_{k-1}.$$

Extrapolation of the covariance matrix:

$$\mathbf{C}_k^{k-1} = \mathbf{F}_{k-1} \mathbf{C}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}.$$

Residuals of predictions:

$$\mathbf{r}_k^{k-1} = \mathbf{m}_k - \mathbf{H}_k \mathbf{x}_k^{k-1}. \quad (7)$$

Covariance matrix of predicted residuals:

$$\mathbf{R}_k^{k-1} = \mathbf{V}_k + \mathbf{H}_k \mathbf{C}_k^{k-1} \mathbf{H}_k^T.$$

Filtering (gain matrix formalism):

Update of the state vector:

$$\mathbf{x}_k = \mathbf{x}_k^{k-1} + \mathbf{K}_k (\mathbf{m}_k - \mathbf{H}_k \mathbf{x}_k^{k-1}).$$

Kalman gain matrix:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{C}_k^{k-1} \mathbf{H}_k^T (\mathbf{V}_k + \mathbf{H}_k \mathbf{C}_k^{k-1} \mathbf{H}_k^T)^{-1} \\ &= \mathbf{C}_k \mathbf{H}_k^T \mathbf{G}_k. \end{aligned}$$

Update of the covariance matrix:

$$\mathbf{C}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{C}_k^{k-1}. \quad (8a)$$

Filtered residuals:

$$\mathbf{r}_k = \mathbf{m}_k - \mathbf{H}_k \mathbf{x}_k = (\mathbf{I} - \mathbf{H}_k \mathbf{K}_k) \mathbf{r}_k^{k-1}.$$

Covariance matrix of filtered residuals:

$$\mathbf{R}_k = (\mathbf{I} - \mathbf{H}_k \mathbf{K}_k) \mathbf{V}_k = \mathbf{V}_k - \mathbf{H}_k \mathbf{C}_k \mathbf{H}_k^T.$$

χ^2 increment:

$$\chi_+^2 = \mathbf{r}_k^T \mathbf{R}_k^{-1} \mathbf{r}_k.$$

χ^2 update:

$$\chi_k^2 = \chi_{k-1}^2 + \chi_+^2.$$

Filtering (weighted means formalism):

Update of the state vector:

$$\mathbf{x}_k = \mathbf{C}_k \left[(\mathbf{C}_k^{k-1})^{-1} \mathbf{x}_k^{k-1} + \mathbf{H}_k^T \mathbf{G}_k \mathbf{m}_k \right].$$

Update of the covariance matrix:

$$\mathbf{C}_k = \left[(\mathbf{C}_k^{k-1})^{-1} + \mathbf{H}_k^T \mathbf{G}_k \mathbf{H}_k \right]^{-1}. \quad (8b)$$

χ^2 increment:

$$\chi_+^2 = \mathbf{r}_k^T \mathbf{G}_k \mathbf{r}_k + (\mathbf{x}_k - \mathbf{x}_k^{k-1})^T (\mathbf{C}_k^{k-1})^{-1} (\mathbf{x}_k - \mathbf{x}_k^{k-1}).$$

χ^2 update:

$$\chi_k^2 = \chi_{k-1}^2 + \chi_+^2.$$

Smoothing:

Smoothed state vector:

$$\mathbf{x}_k^n = \mathbf{x}_k + \mathbf{A}_k (\mathbf{x}_{k+1}^n - \mathbf{x}_{k+1}^k).$$

Smoother gain matrix:

$$\mathbf{A}_k = \mathbf{C}_k \mathbf{F}_k^T (\mathbf{C}_{k+1}^k)^{-1}.$$

Covariance matrix of the smoothed state vector:

$$\mathbf{C}_k^n = \mathbf{C}_k + \mathbf{A}_k (\mathbf{C}_{k+1}^n - \mathbf{C}_{k+1}^k) \mathbf{A}_k^T. \quad (9)$$

Smoothed residuals:

$$\mathbf{r}_k^n = \mathbf{r}_k - \mathbf{H}_k (\mathbf{x}_k^n - \mathbf{x}_k) = \mathbf{m}_k - \mathbf{H}_k \mathbf{x}_k^n.$$

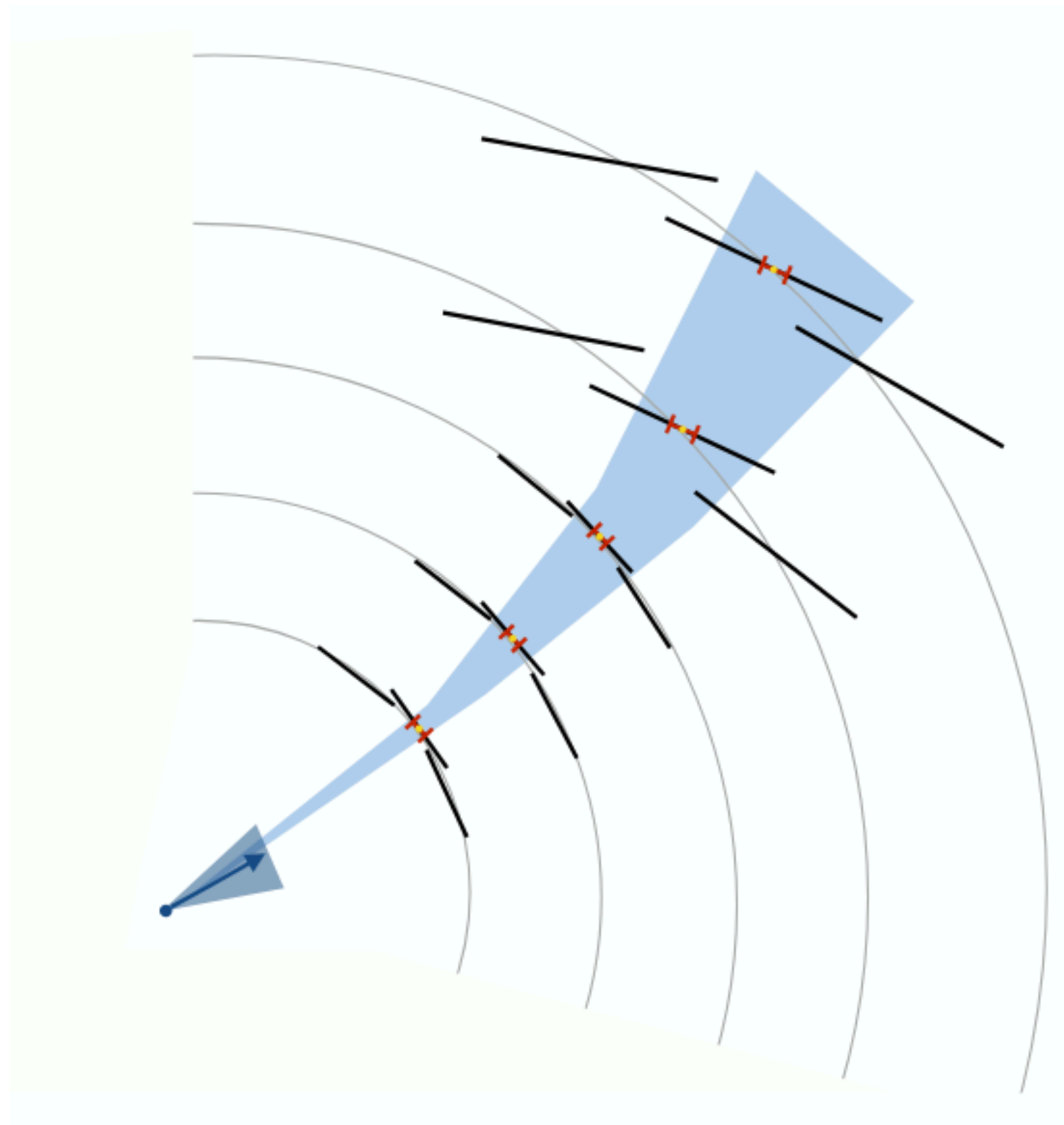
Covariance matrix of smoothed residuals:

$$\mathbf{R}_k^n = \mathbf{R}_k - \mathbf{H}_k \mathbf{A}_k (\mathbf{C}_{k+1}^n - \mathbf{C}_{k+1}^k) \mathbf{A}_k^T \mathbf{H}_k^T = \mathbf{V}_k - \mathbf{H}_k \mathbf{C}_k^n \mathbf{H}_k^T.$$

Frühwirth

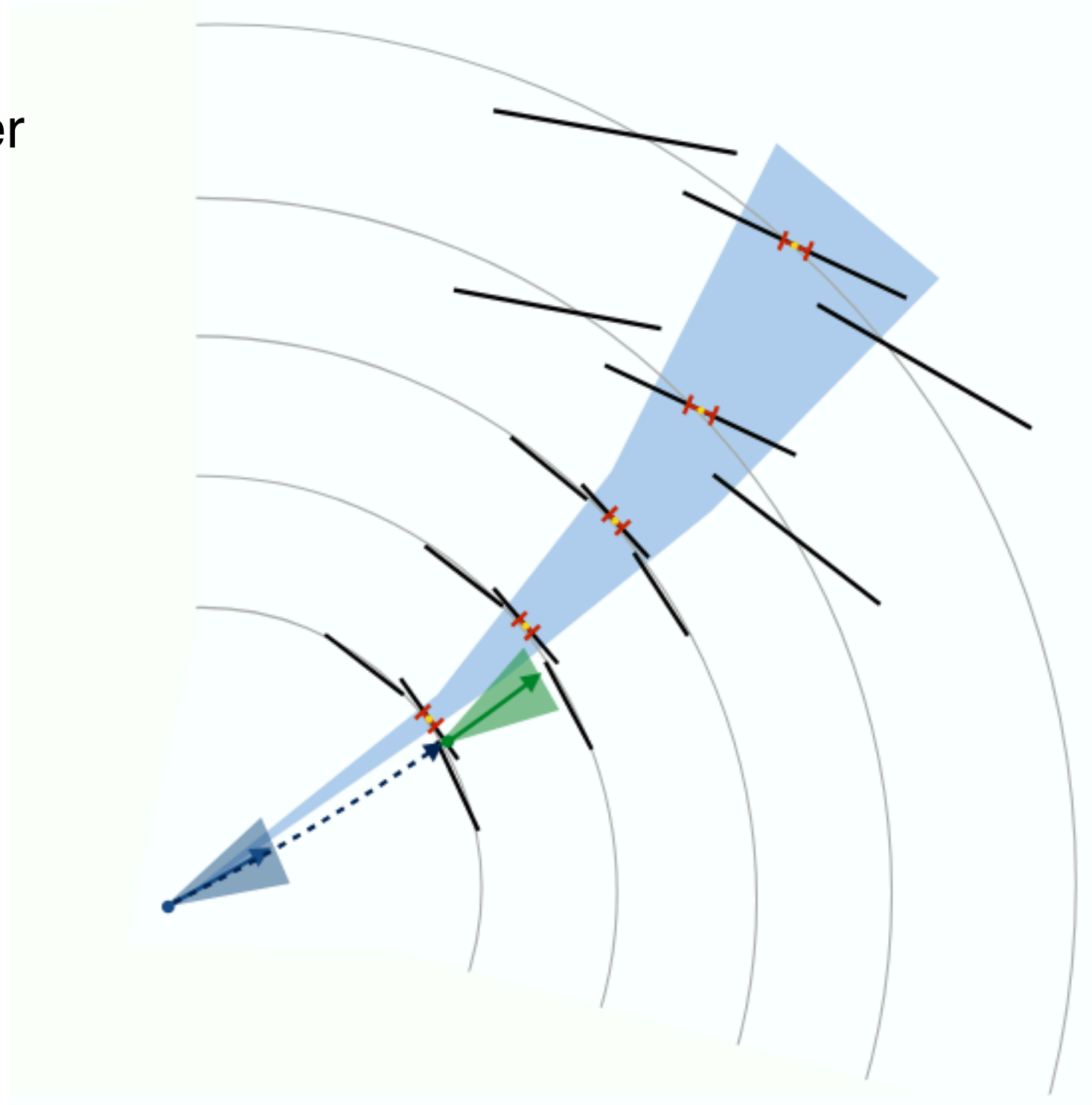
Track reconstruction: Kalman Filter

- Start from the seed direction



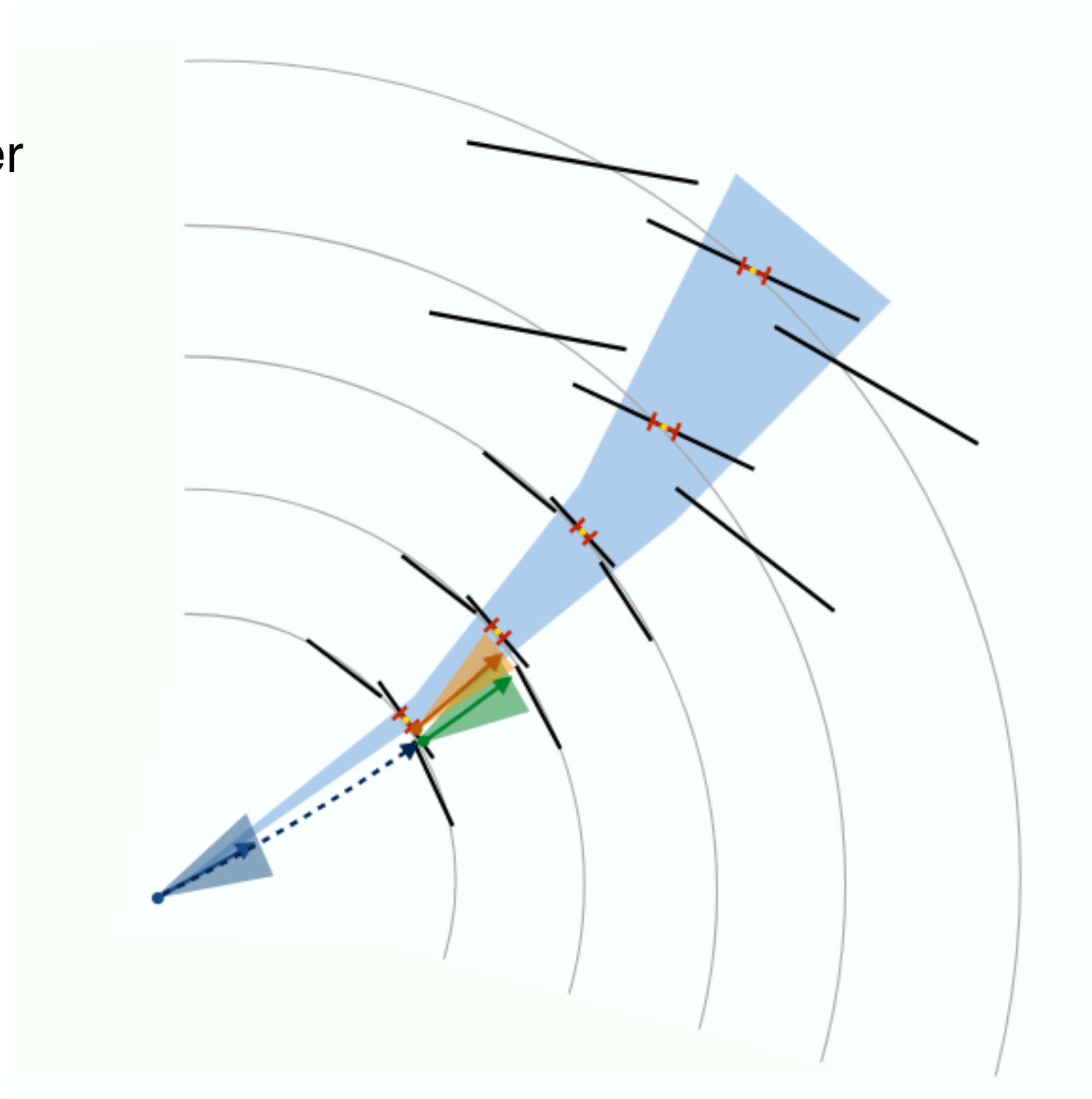
Track reconstruction: Kalman Filter

- Start from the seed direction
- **Extrapolation** to the next layer



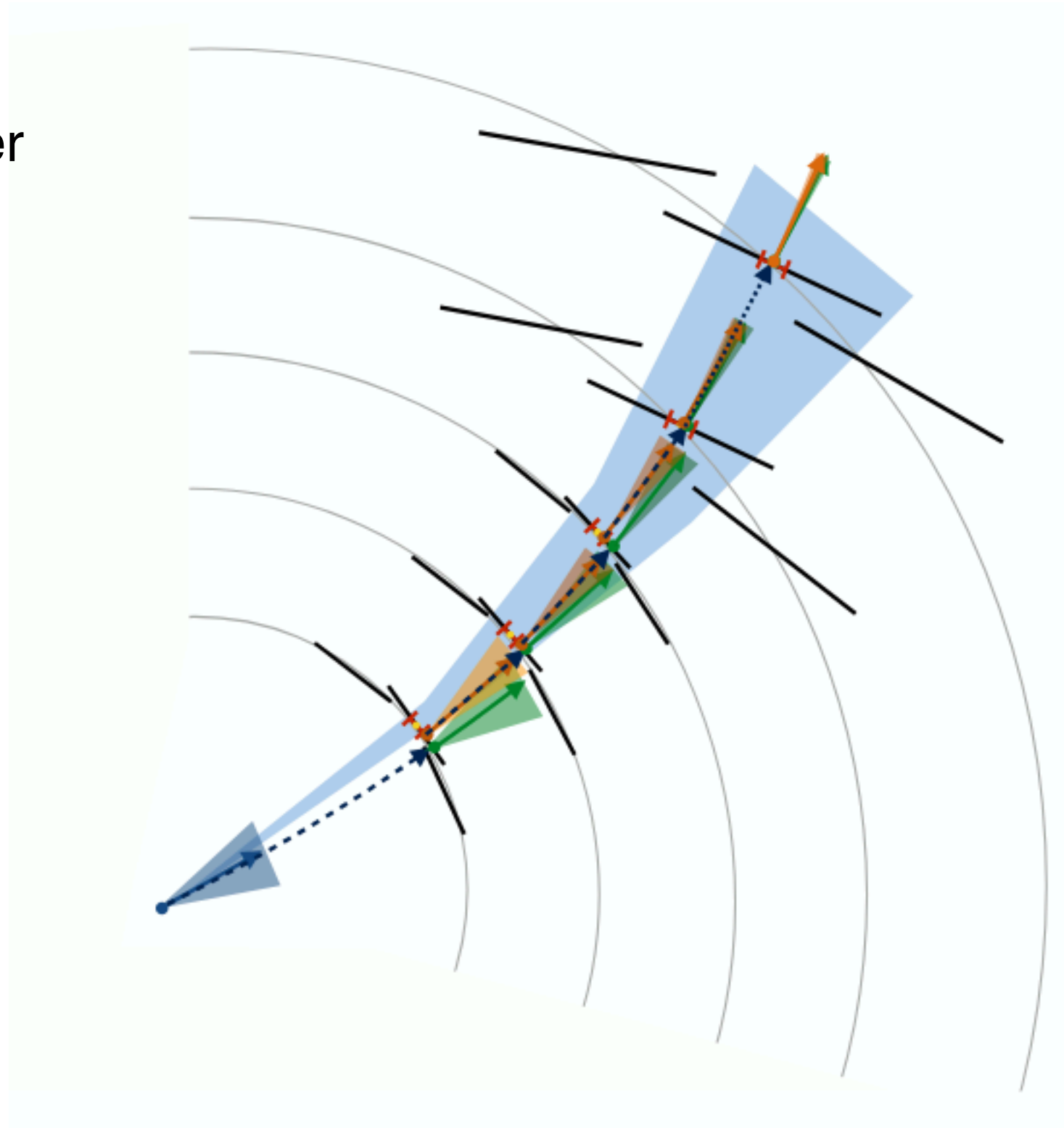
Track reconstruction: Kalman Filter

- Start from the seed direction
- **Extrapolation** to the next layer
- **Correction** based on the layer hit



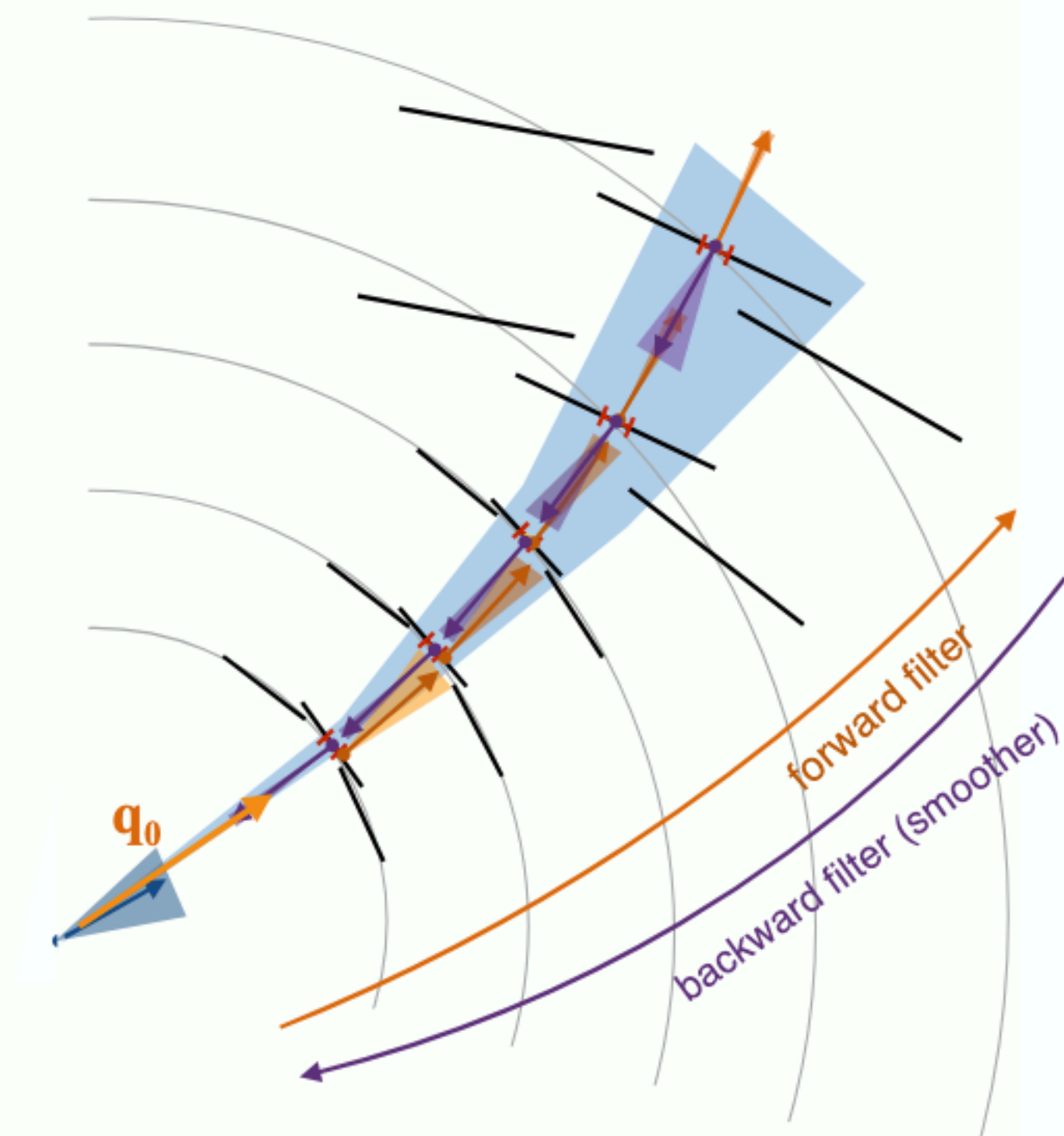
Track reconstruction: Kalman Filter

- Start from the seed direction
- **Extrapolation** to the next layer
- **Correction** based on the layer hit
- Iterate through the detector



Track reconstruction: Kalman Filter

- Start from the seed direction
- **Extrapolation** to the next layer
- **Correction** based on the layer hit
- Iterate through the detector
- **Fit** of the trajectory



Attention Map for a bin

