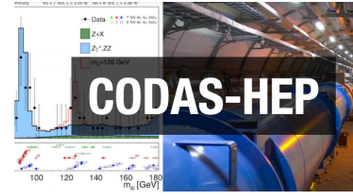


Charged Particle Tracking Reconstruction

Slava Krutelyov

CODAS-HEP 2018

July 25, 2018

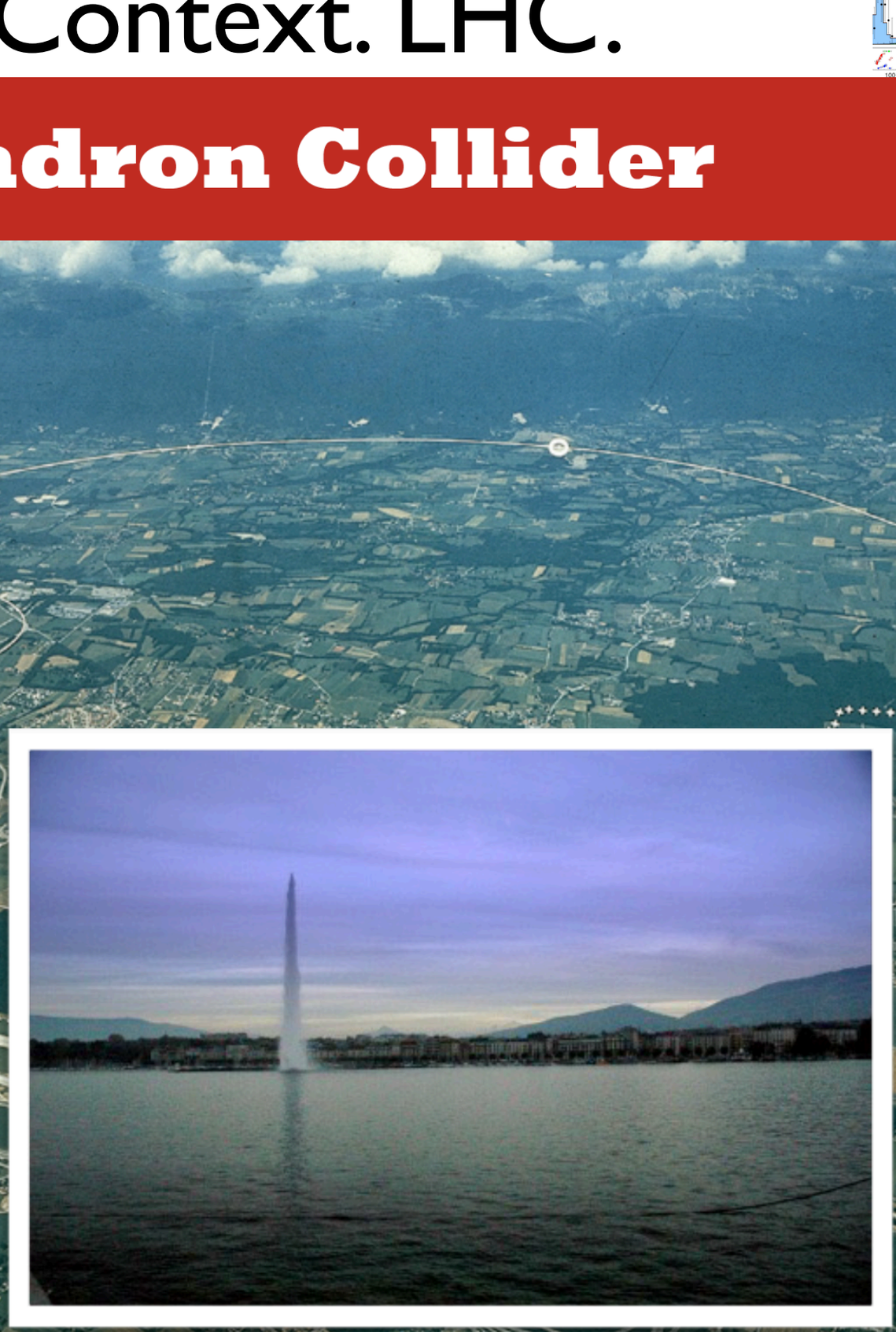
— Tracking
  — Context/detectors
  — Concept and Formalism
    — Several exercises here
  — CMS tracking example
— Pileup: computational challenge
— Parallelization

Tracking: Context. LHC.

Large Hadron Collider

Tracking: Context. LHC.

Large Hadron Collider



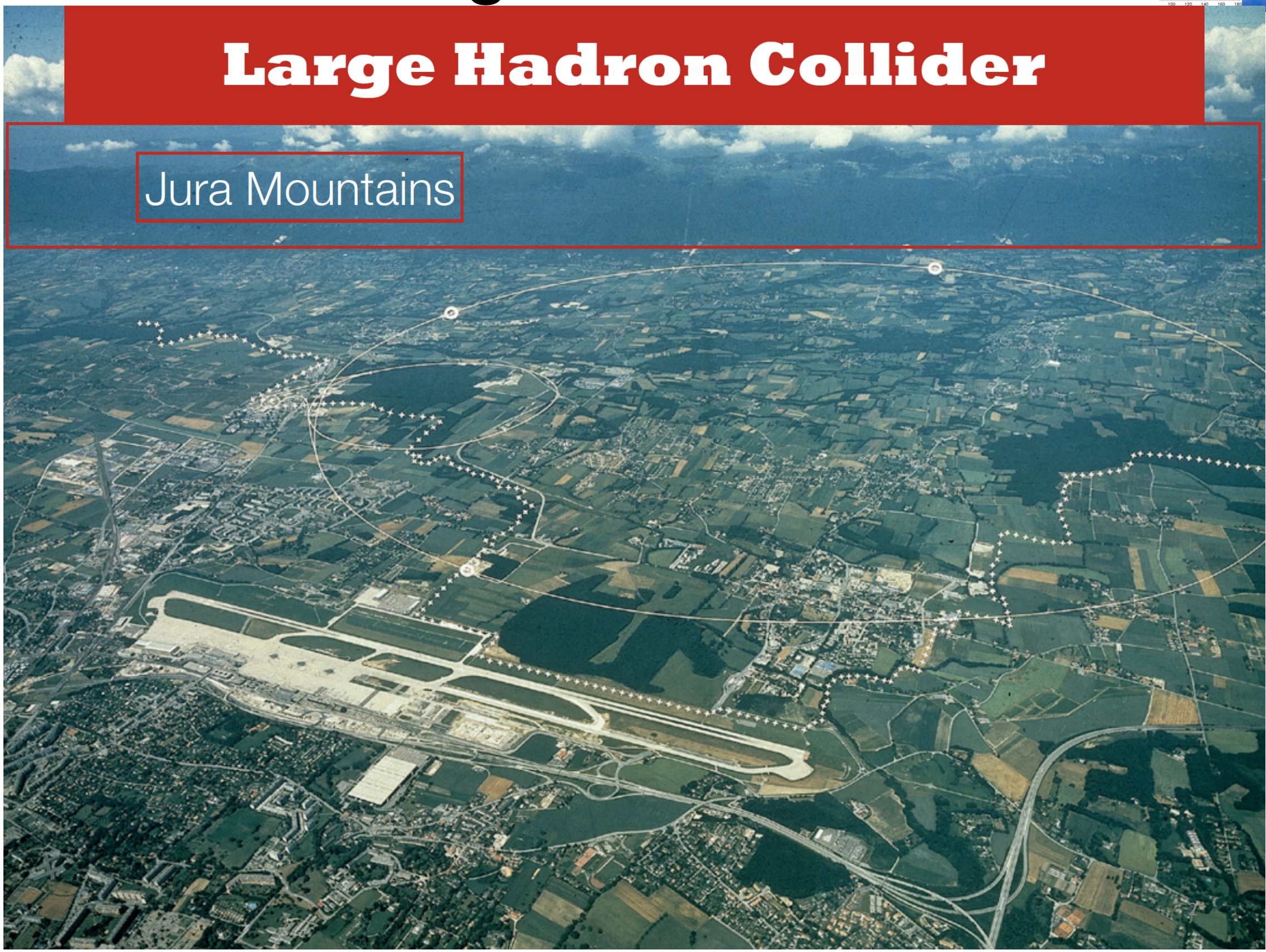
downtown GVA, Alps

Tracking: Context. LHC.

Large Hadron Collider

GVA airport

Tracking: Context. LHC.

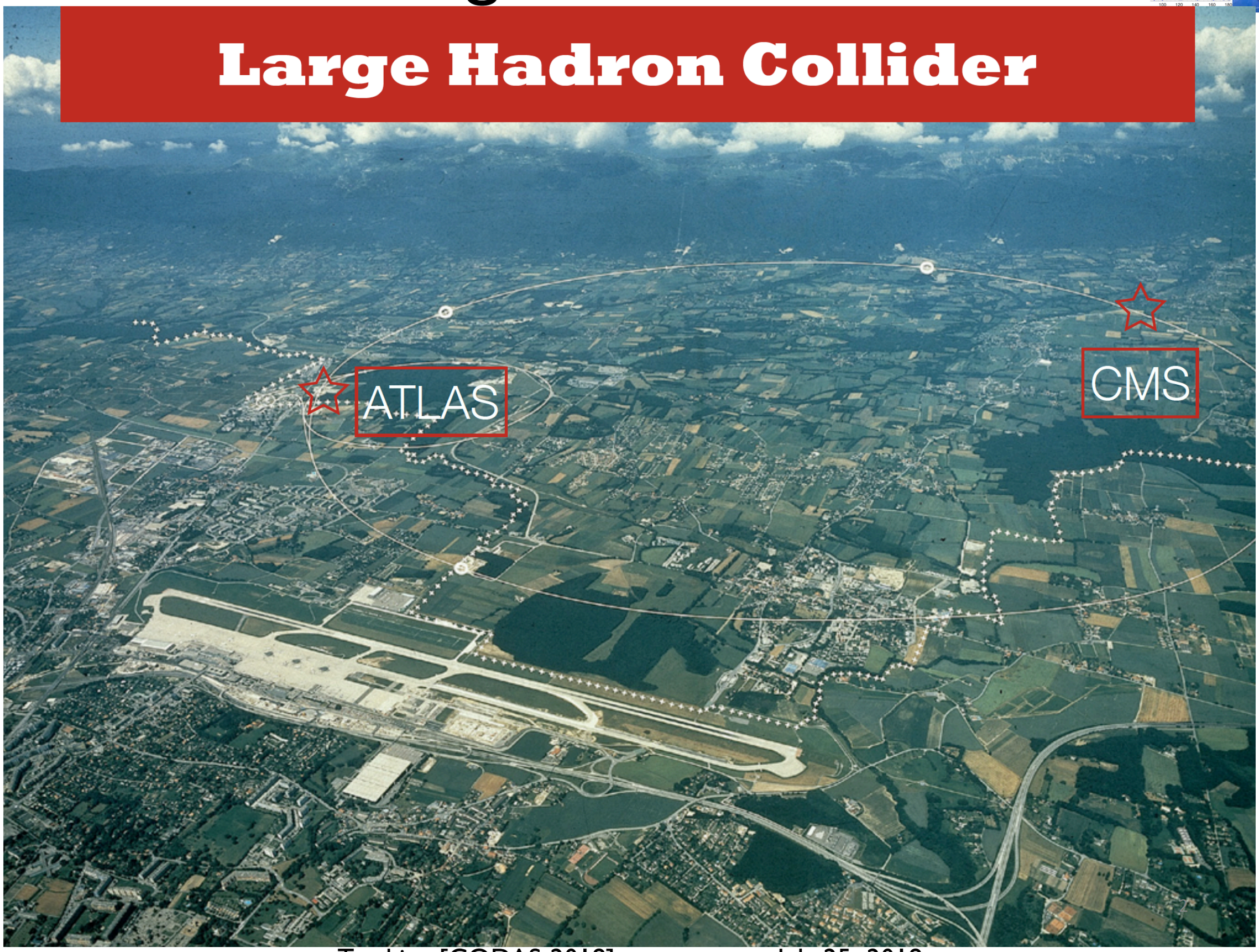


Tracking: Context. LHC.

Tracking: Context. LHC.

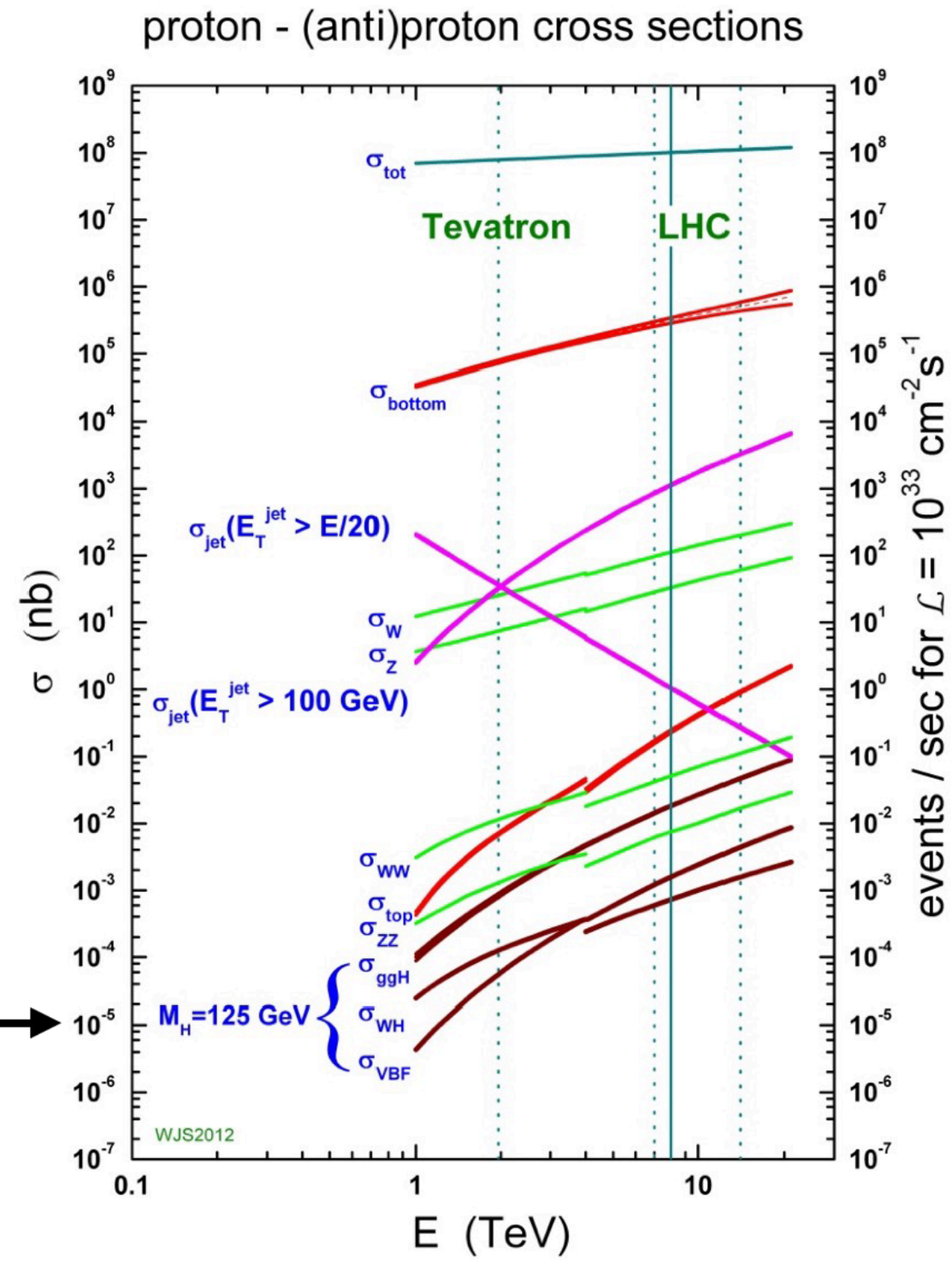
Tracking: Context. LHC.

Tracking: Context. LHC.



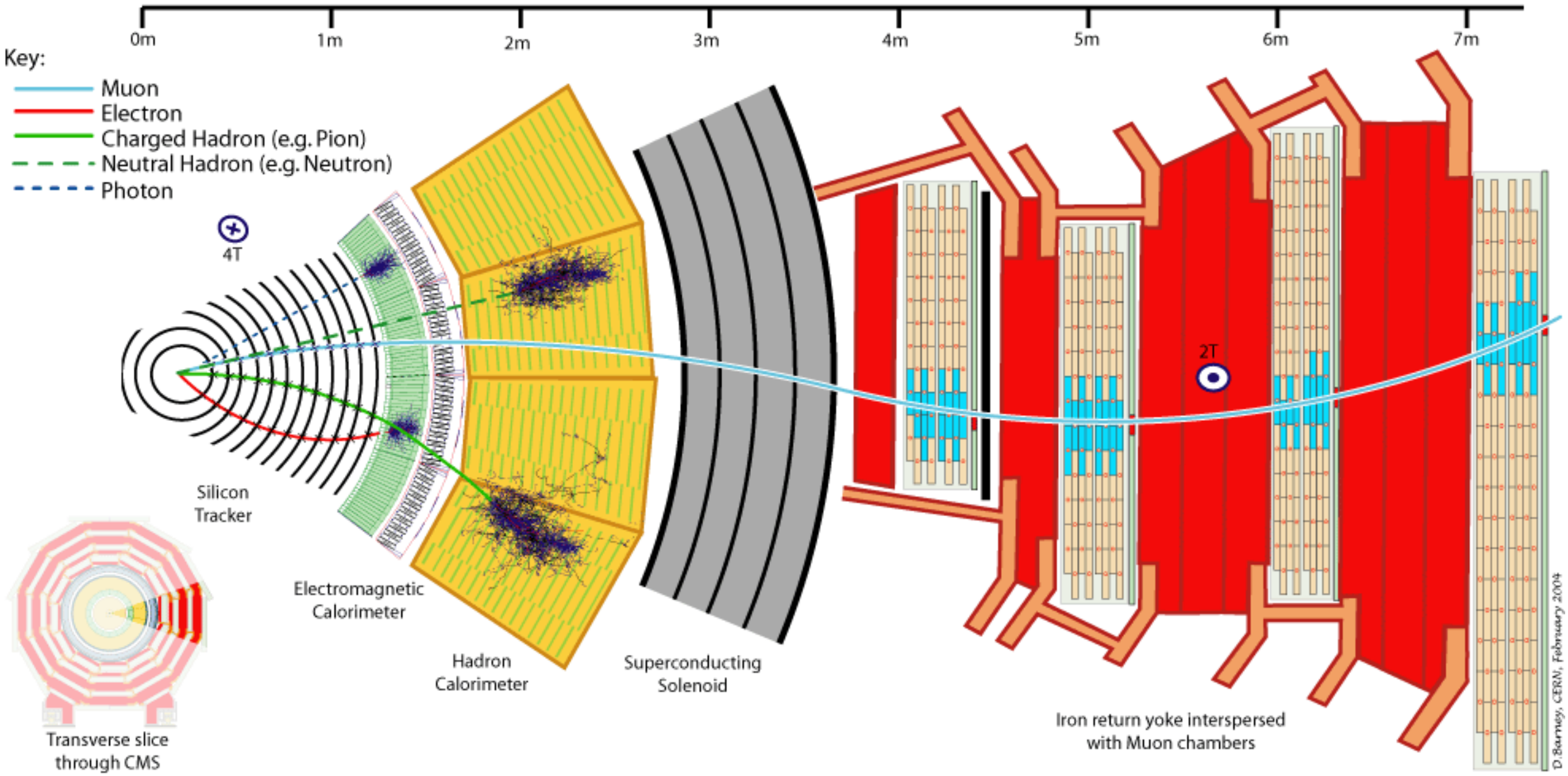
Big Data Challenge

- 40 million collisions a second
- Most are boring
  - ✓ **Dropped within 3 μs**
- 0.1% are interesting
  - ✓ **Worthy of reconstruction...**
- Higgs events: super rare
  - ✓ **10^{16} collisions → 10^{6} Higgs**
  - ✓ **Maybe 1% of these are found**
- Ultimate "needle in a haystack"
- First "Big Data" problem



http://www.hep.ph.ic.ac.uk/~wstirlin/plots/plots.html

Tracking: Context. Detector.



- Particles interact differently, so CMS is a detector with different layers to identify the decay remnants of Higgs bosons and other unstable particles

Tracking: Concept

Track reconstruction =
pattern recognition + tracking fitting

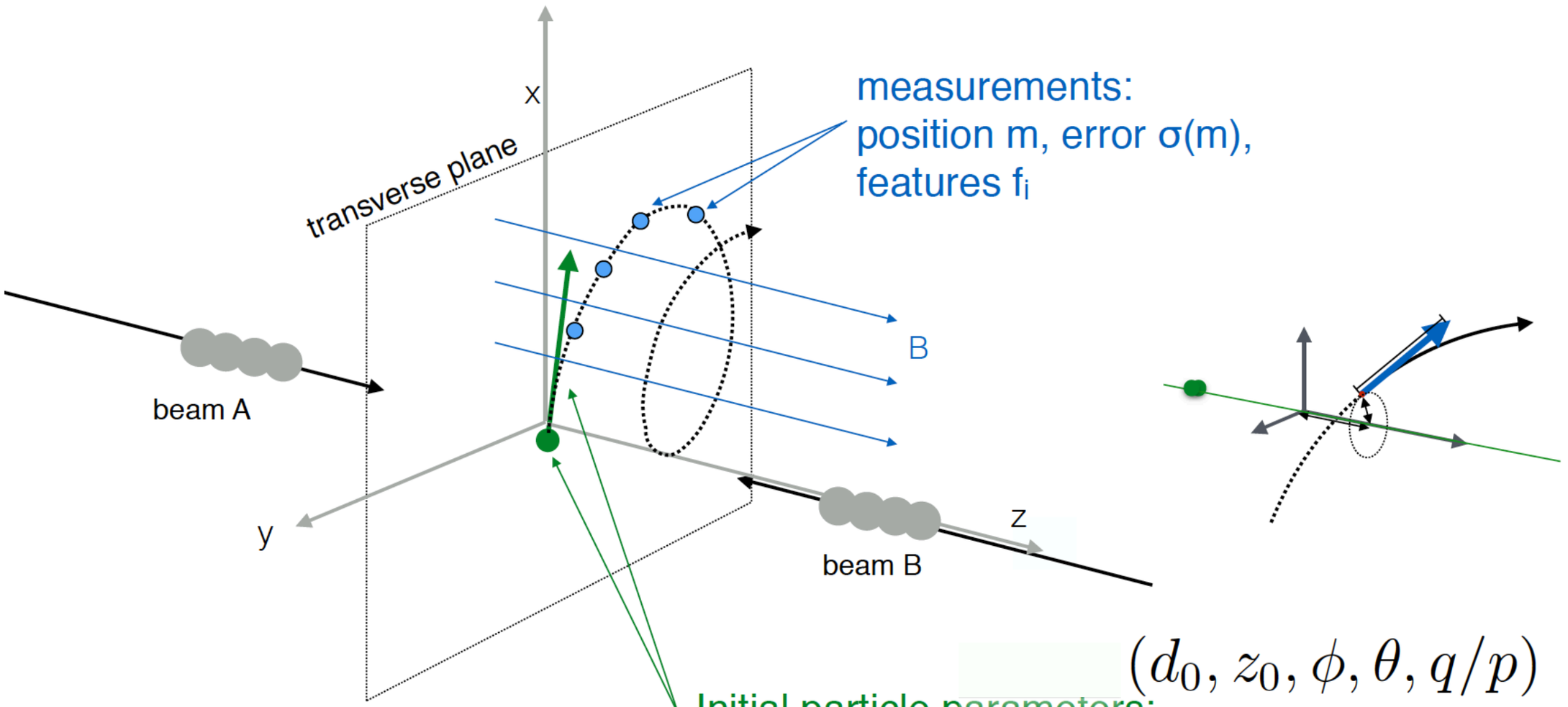
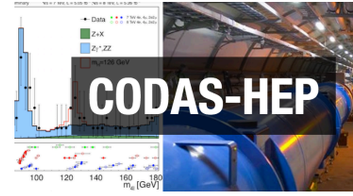


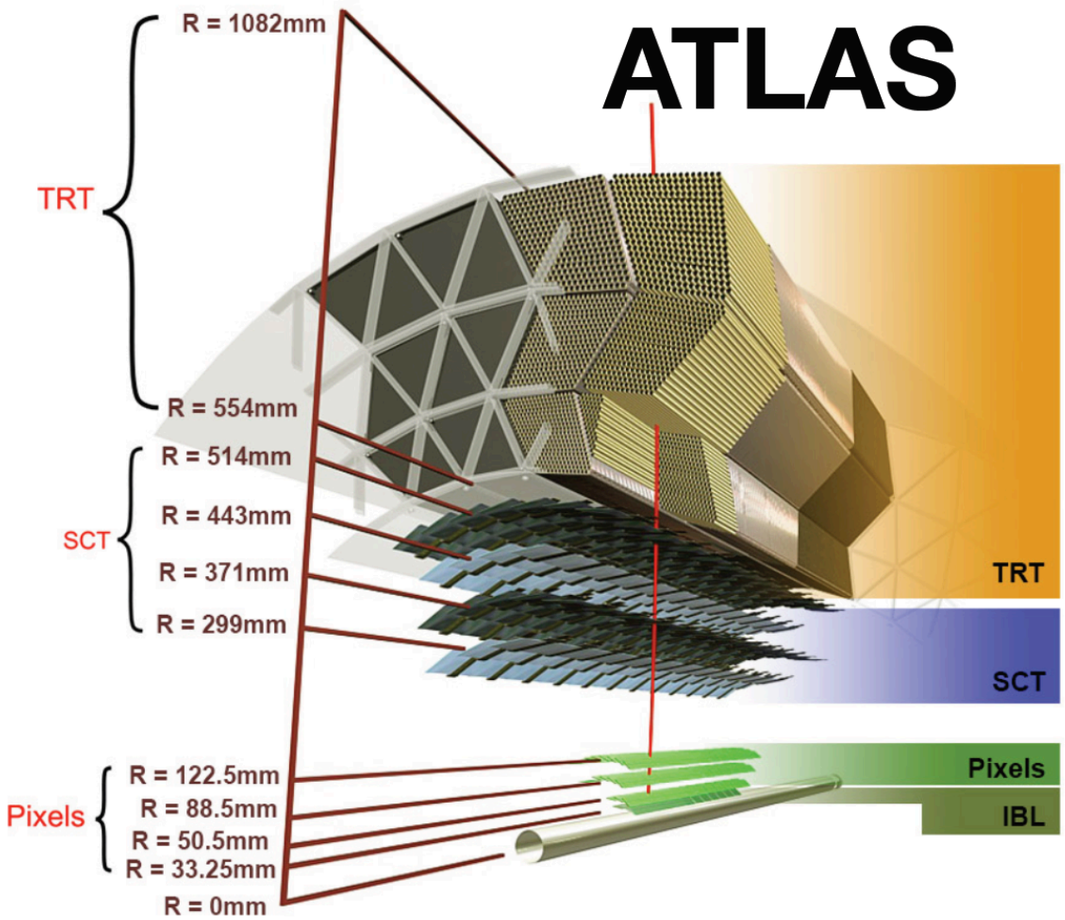
Illustration:
A schematic view of a particle in a magnetic field.

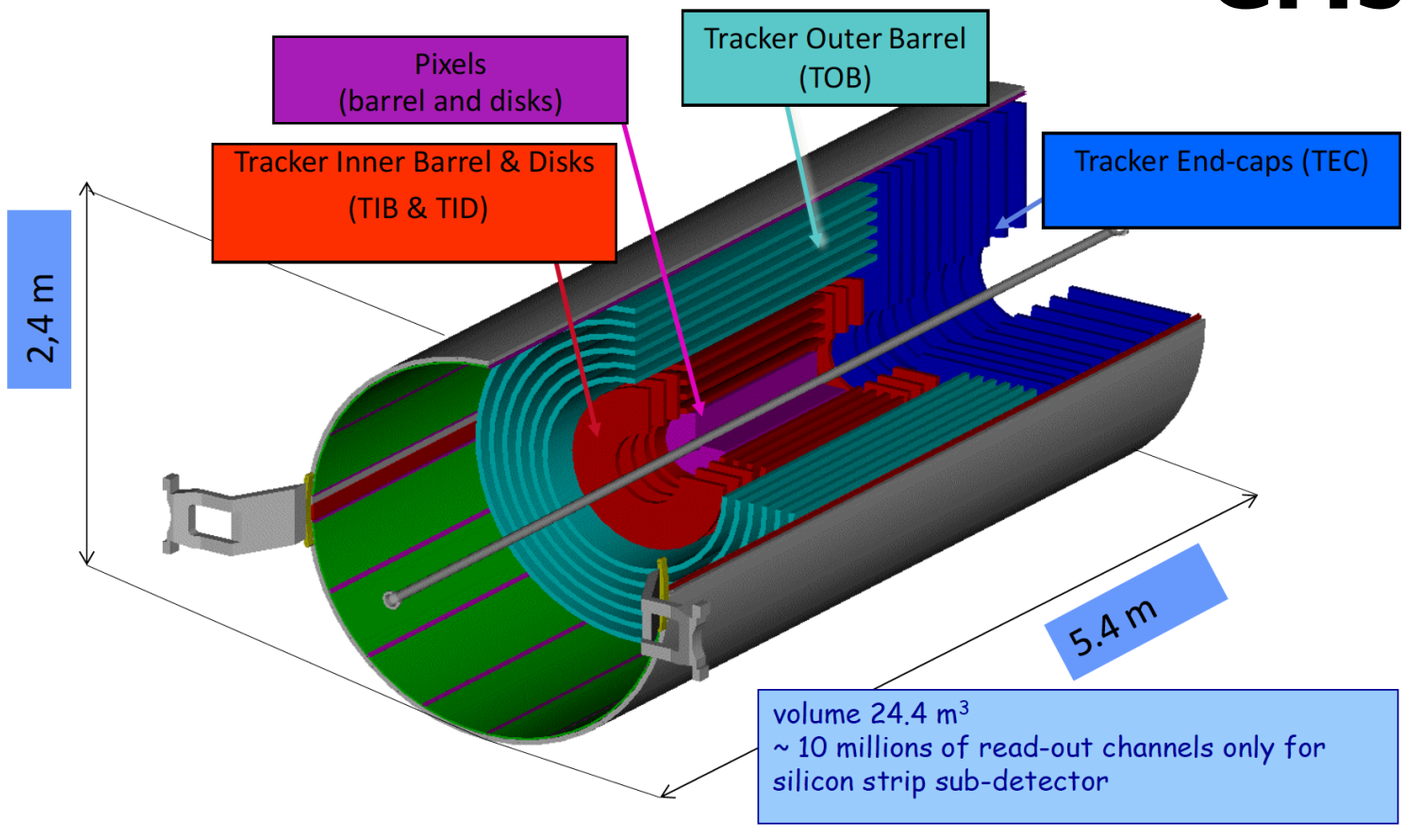
H.Gray ACAT2017

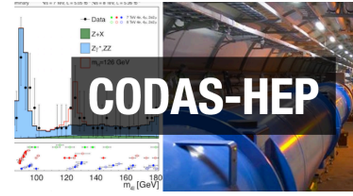
Tracking: finding trajectories



- Variety of experimental solutions
- Solenoidal magnetic field is more common in colliders

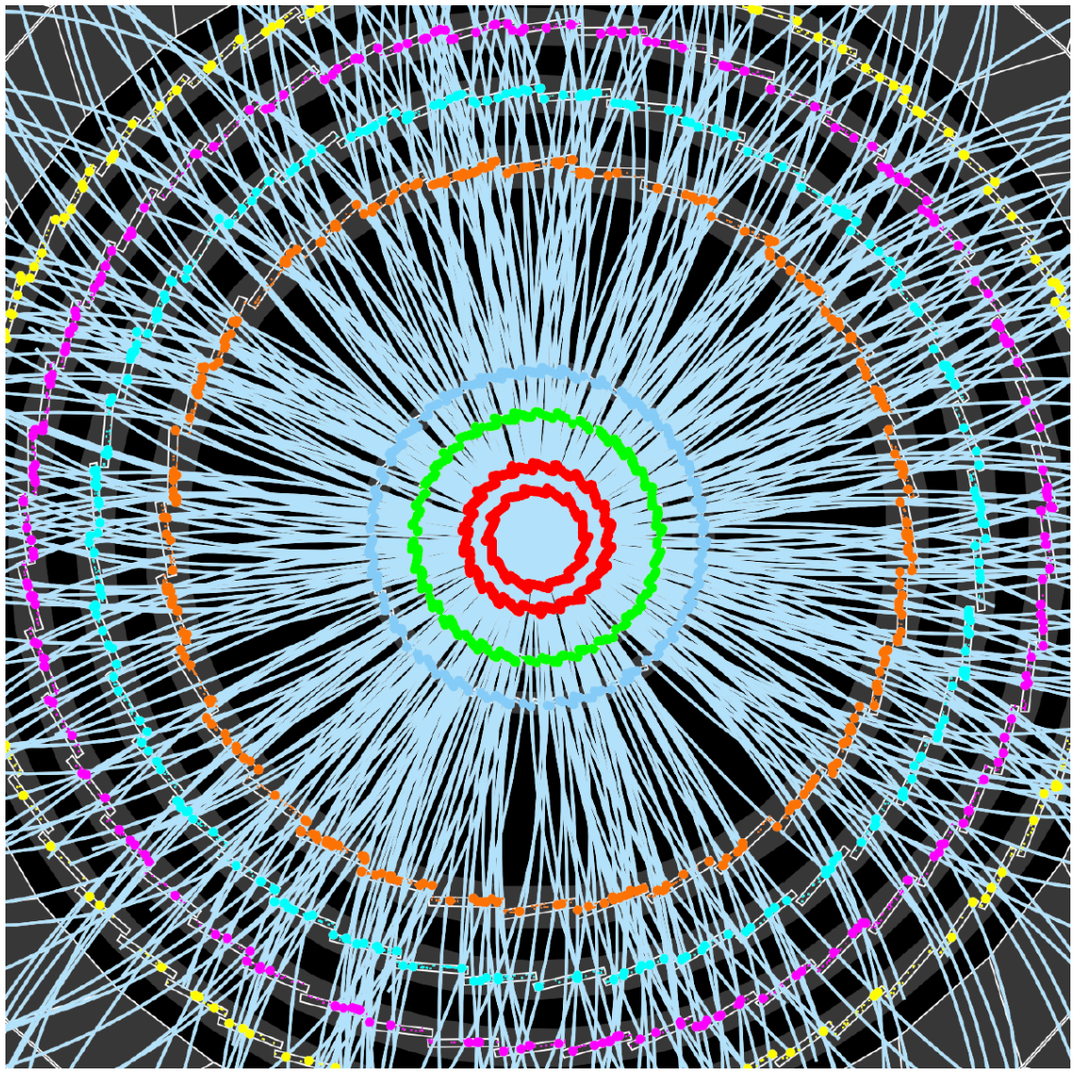


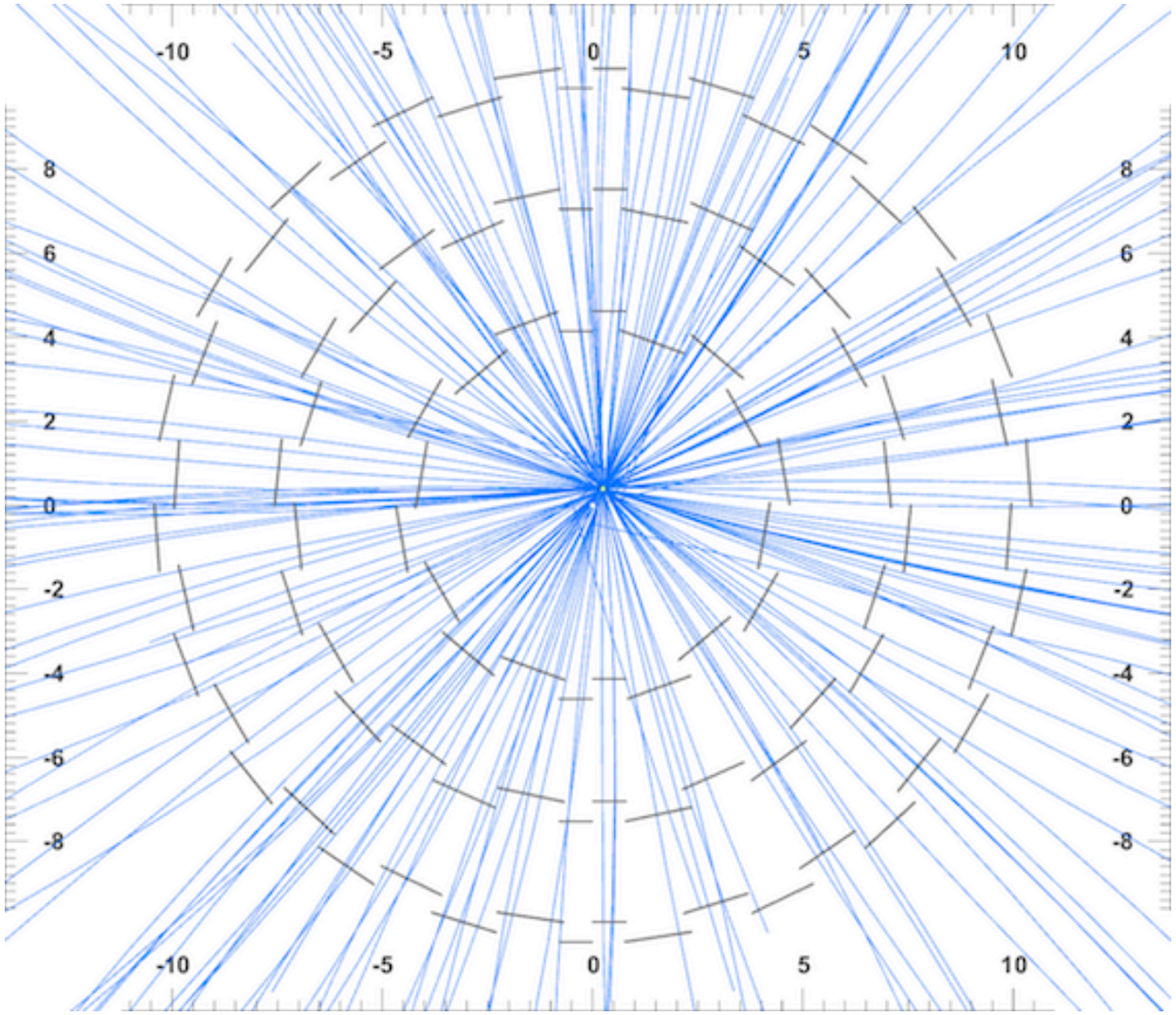




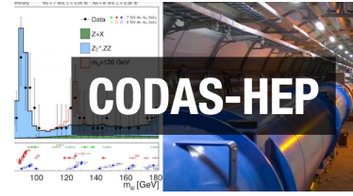
Tracking: finding trajectories

- Variety of experimental solutions
- Solenoidal magnetic field is more common in colliders

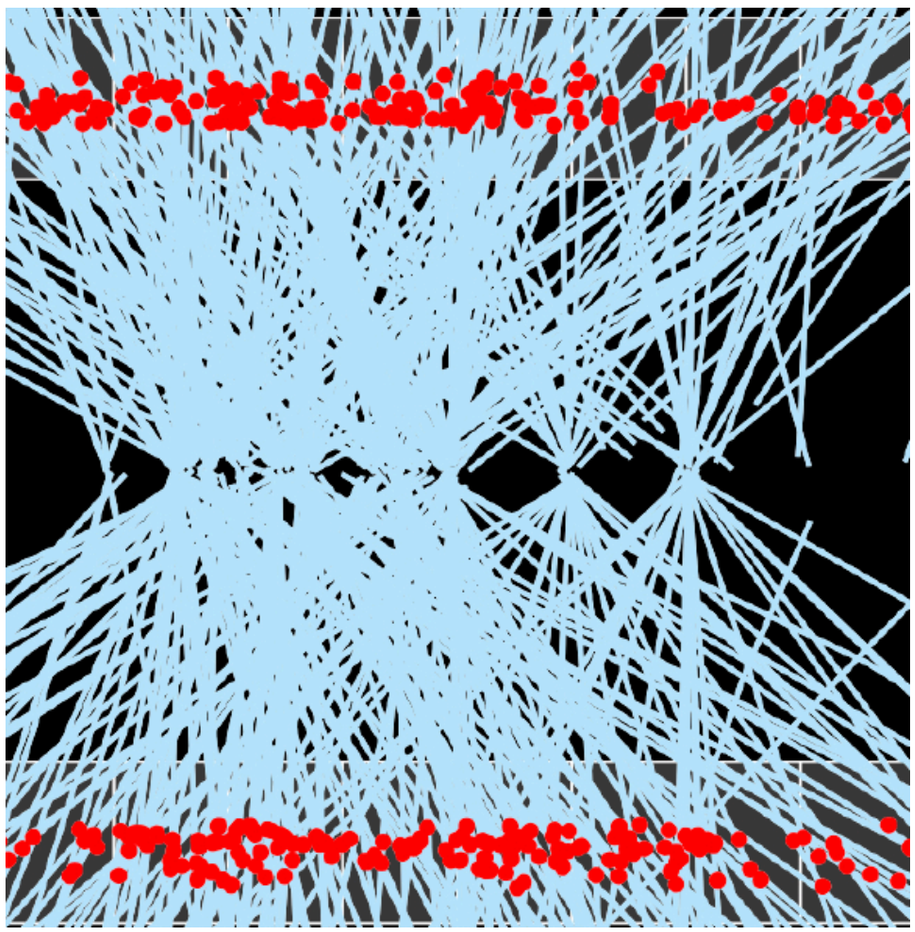




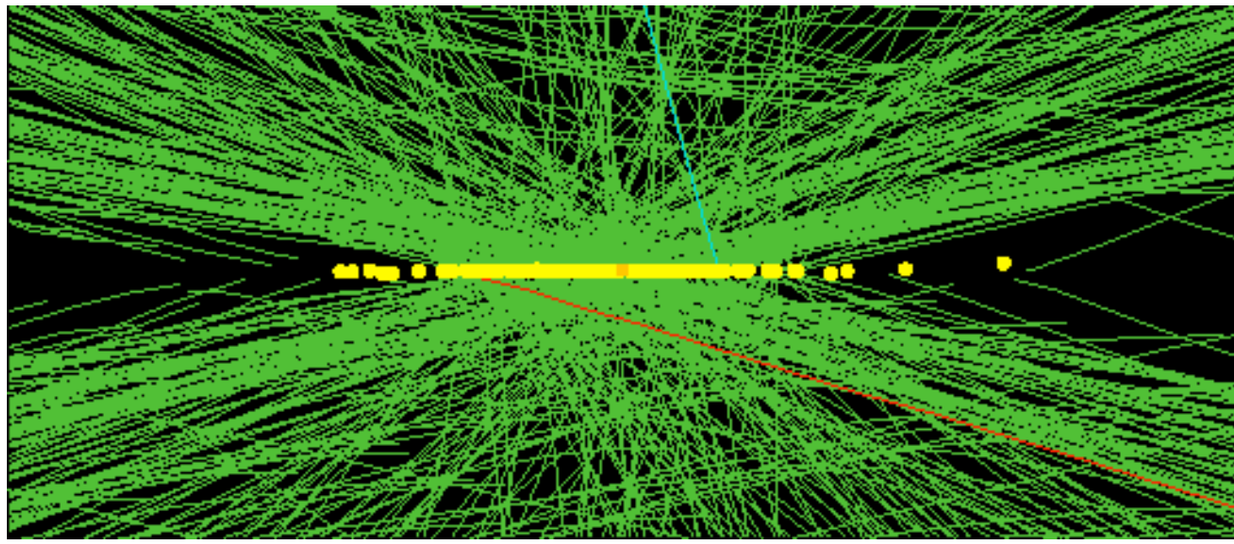
Tracking: finding trajectories



- Variety of experimental solutions
- Solenoidal magnetic field is more common in colliders

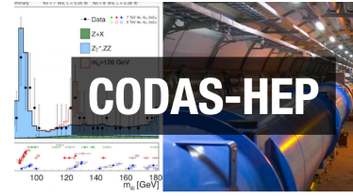


PU~100 actual event with 78 vertices
CMS-PHO-EVENTS-2012-006



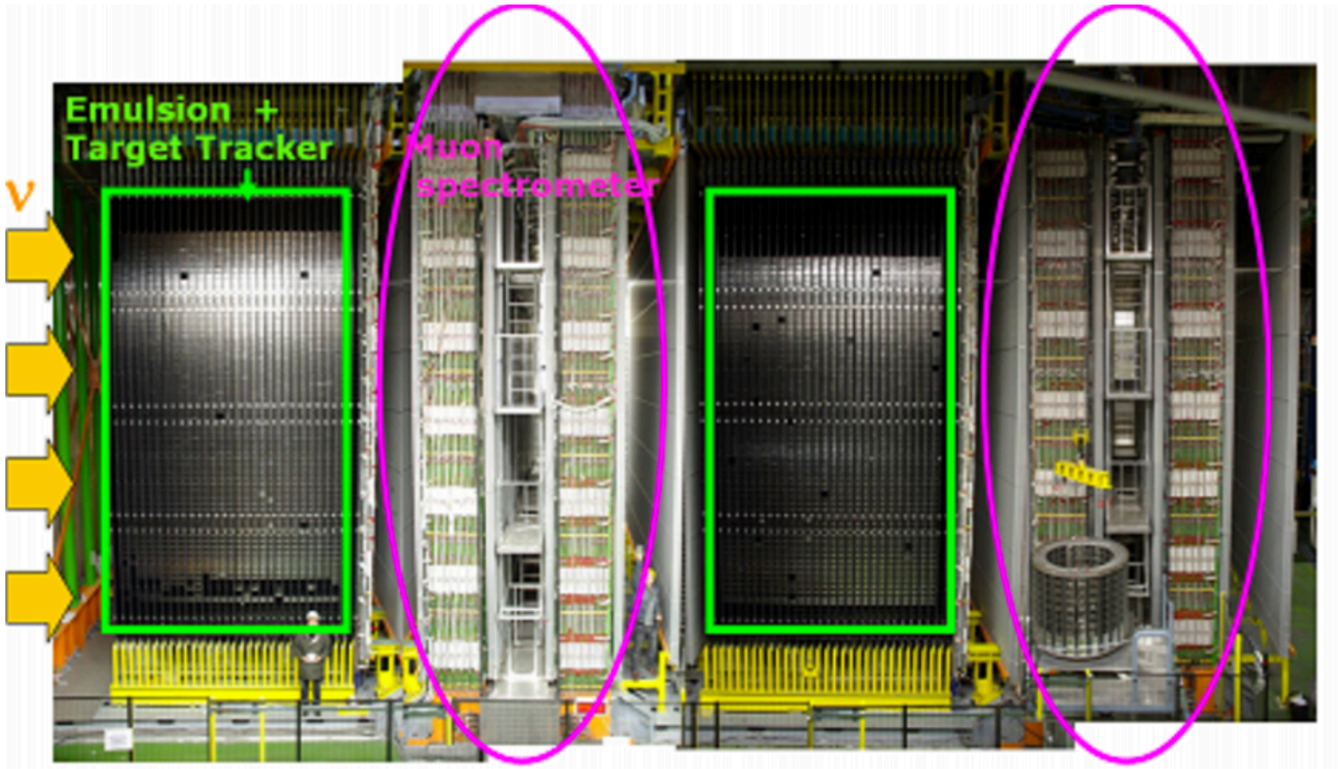
- Multiple proton-proton collisions ==> pileup is a common problem

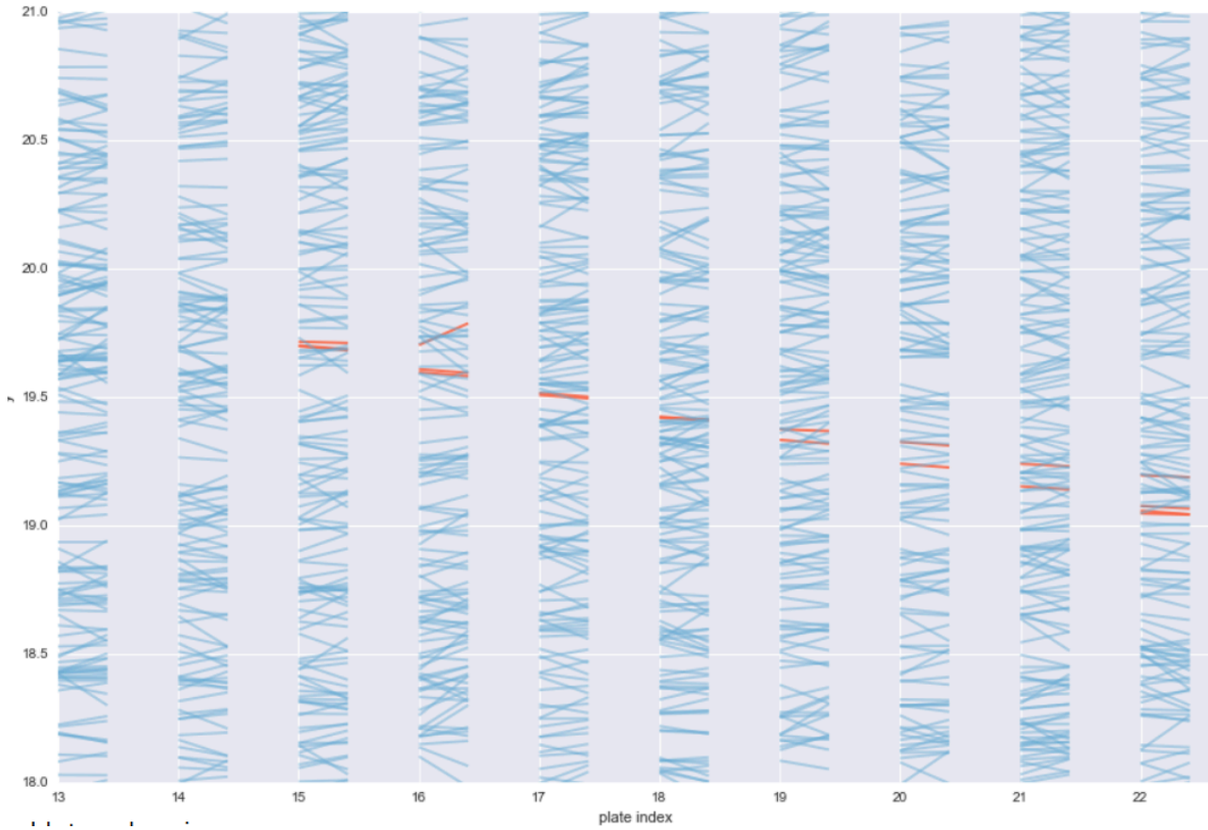
Tracking: finding trajectories



- Variety of experimental solutions
- Some trackers are quite different

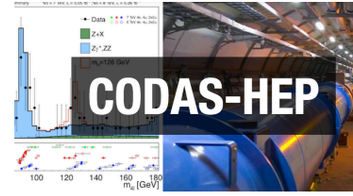
OPERA detector



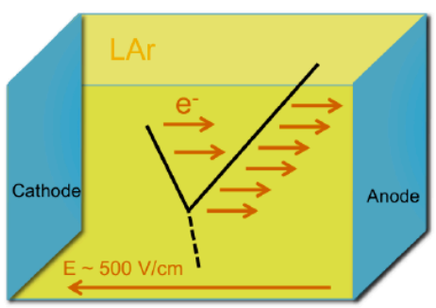


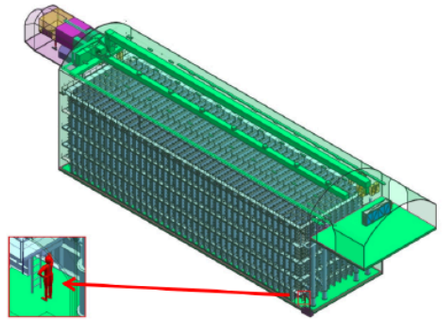
- Pileup is also a problem, but it has a bit different meaning

Tracking: finding trajectories

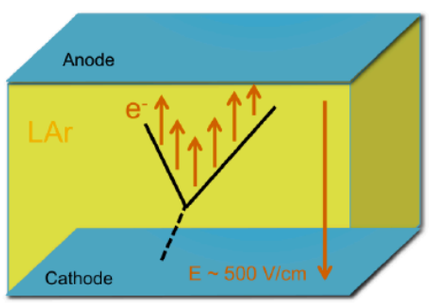


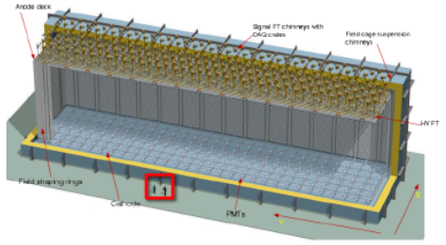
- Variety of experimental solutions
- Some trackers are quite different





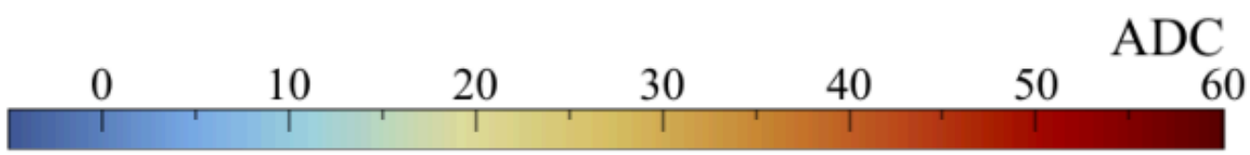
- 150 Anode Plane Assemblies (**APAs**) [2.3 × 6 m]
- 384,000 readout wires

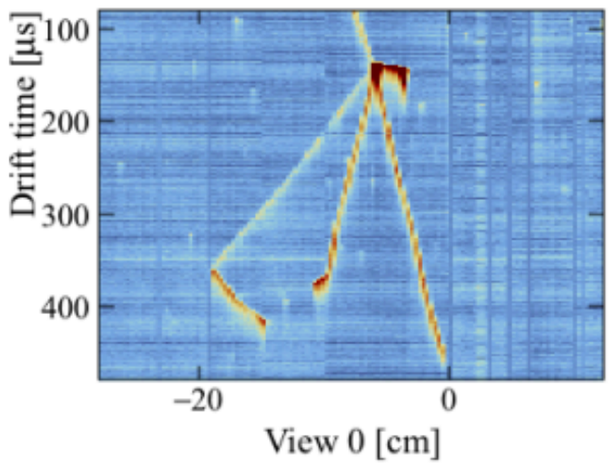


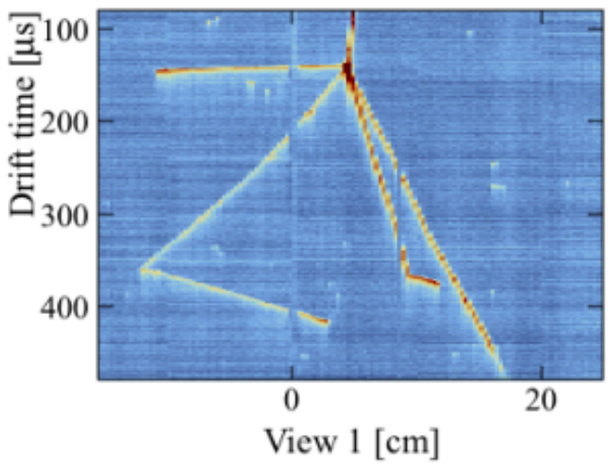


- signal amplification in gas phase
- 80 Charge Readout Planes (**CRPs**) [3 × 3 m^2]

DUNE detector

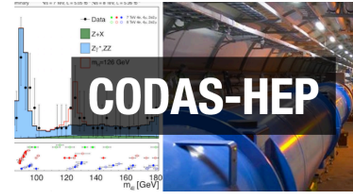




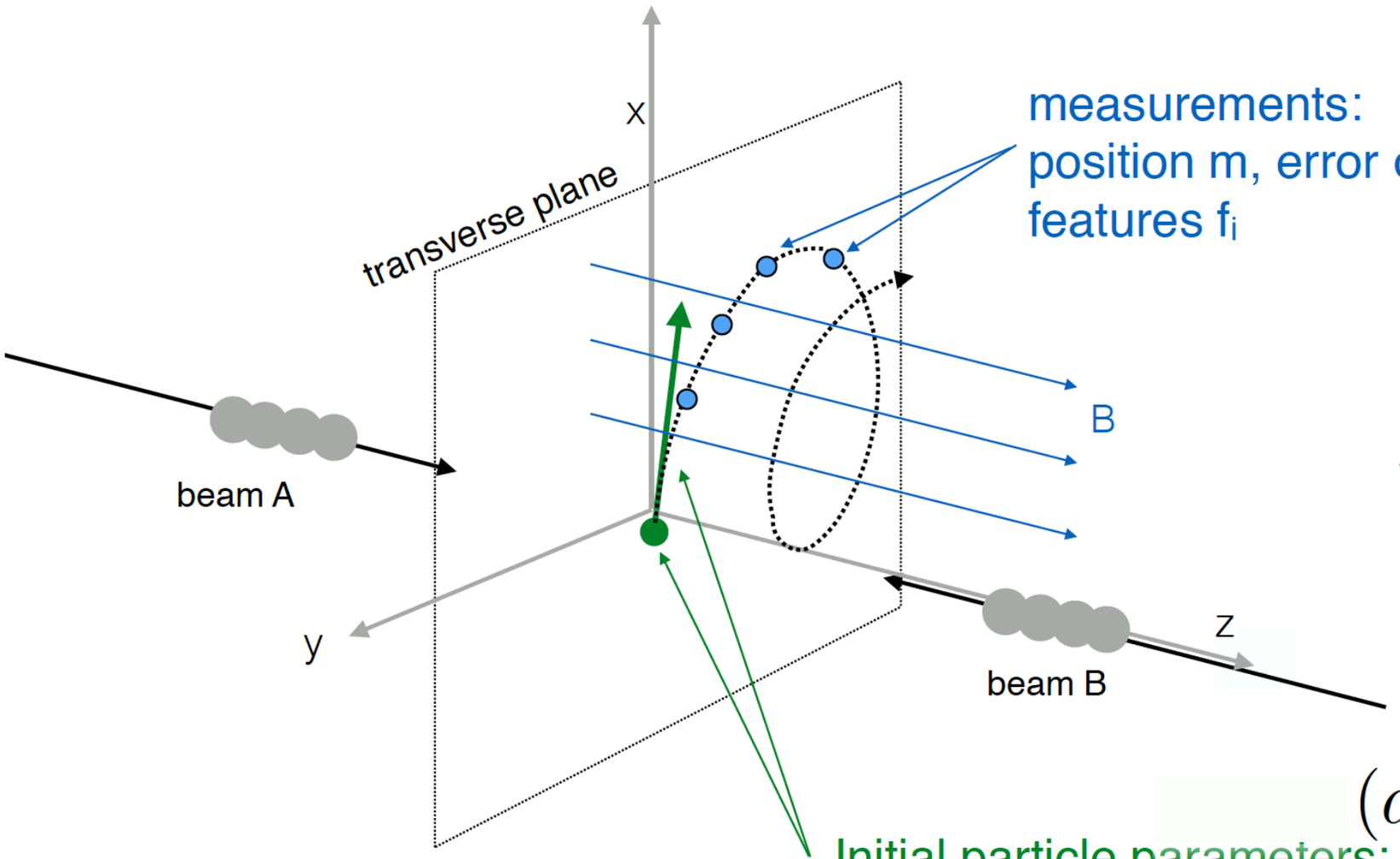


Cosmic ray (raw) events recorded in the 3×1×1m^3

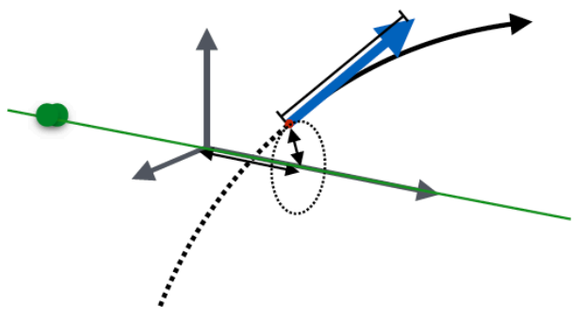
Tracking: basics



- Let's look get back to the "simple" case of tracking in solenoidal field



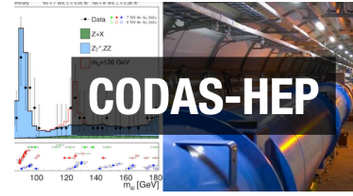
measurements:
position m, error σ(m),
features f_i



$(d_0, z_0, \phi, \theta, q/p)$

Initial particle parameters:
position x, momentum p, charge q

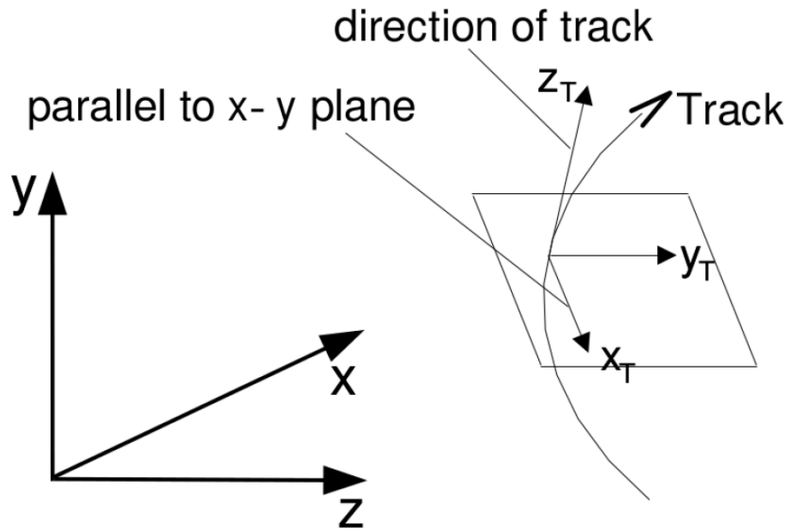
Tracking: formalism



- It turns out there are many "convenient" ways to define a track or its state relative to some reference

$$\text{Cartesian} \quad (x, y, z, p_x, p_y, p_z)$$

$$\text{Curvilinear} \quad (q/p, \lambda, \phi, x_T, y_T)$$

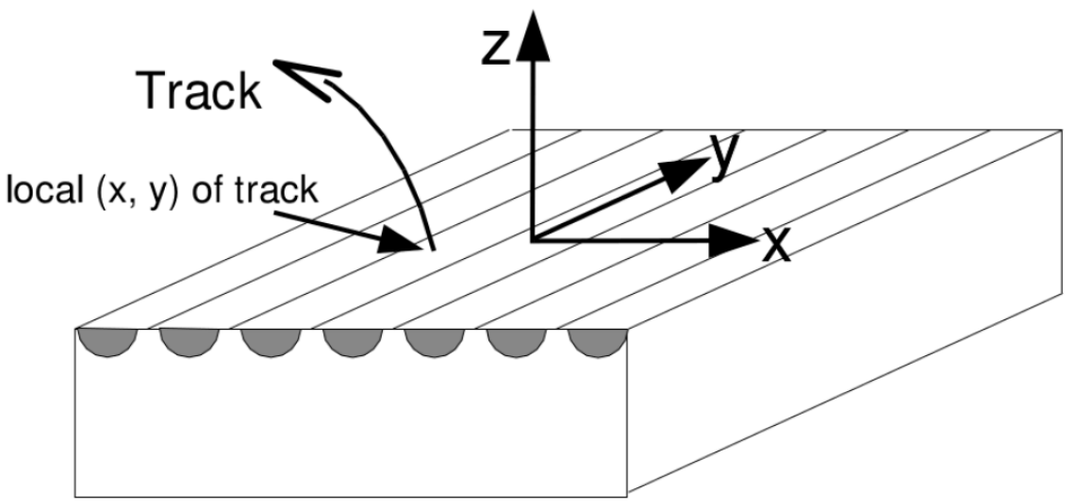


- Curvilinear actually used in CMS:
  - $q/|p|$, signed inverse momentum measured in GeV $^{-1}$
  - $\lambda = \pi/2 - \theta$, where θ is the polar angle
  - ϕ, the azimuthal angle
  - $d_{xy} = -v_x \sin\phi + v_y \cos\phi$, measured in cm
  - $d_{sz} = v_z \cos\lambda - (v_x \cos\phi + v_y \sin\phi)\sin\lambda$, measured in cm

✳ **(vx,vy,vz) is the point of closest approach to (0,0,0)**

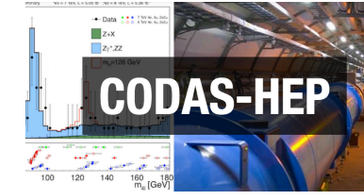
$$\text{Local} \quad (q/p, \frac{dx}{dz}, \frac{dy}{dz}, x, y)$$

- Local is used in CMS tracking



-

Tracking: the circle



- Circle: most relevant part of tracking in xy plane
  - **common shape for track trajectories**
- Radius of the trajectory

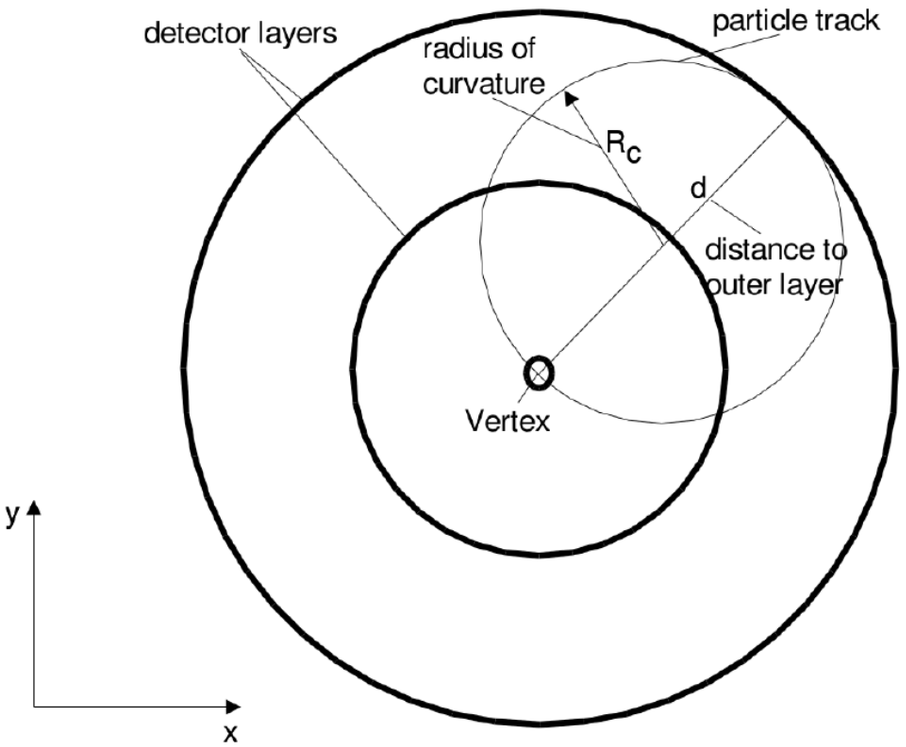
$$R_c \approx \frac{p_T}{0.3B_z}$$

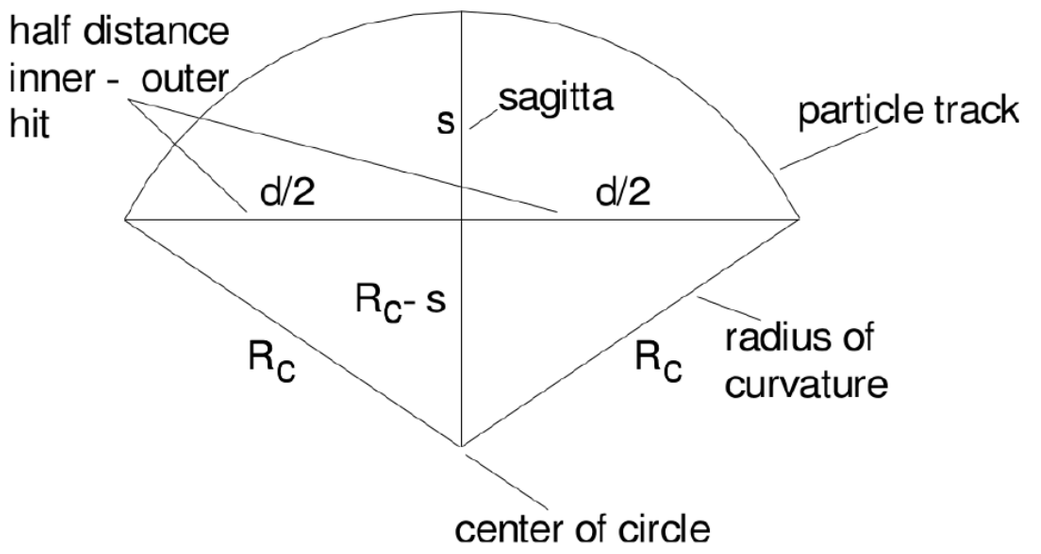
- max p_T for trajectory that loops inside tracker
  - **For CMS d ~ 1 m**

$$p_{T,d} \approx 0.3\frac{d}{2}B_z[CMS :\approx 0.6\text{GeV}]$$

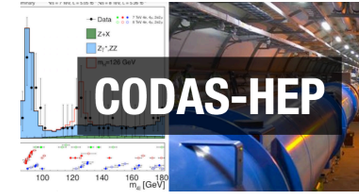
- Sagitta, s, relates to track p_T in a simple way

$$p_T \approx p_{T,d}\frac{d}{4s}$$

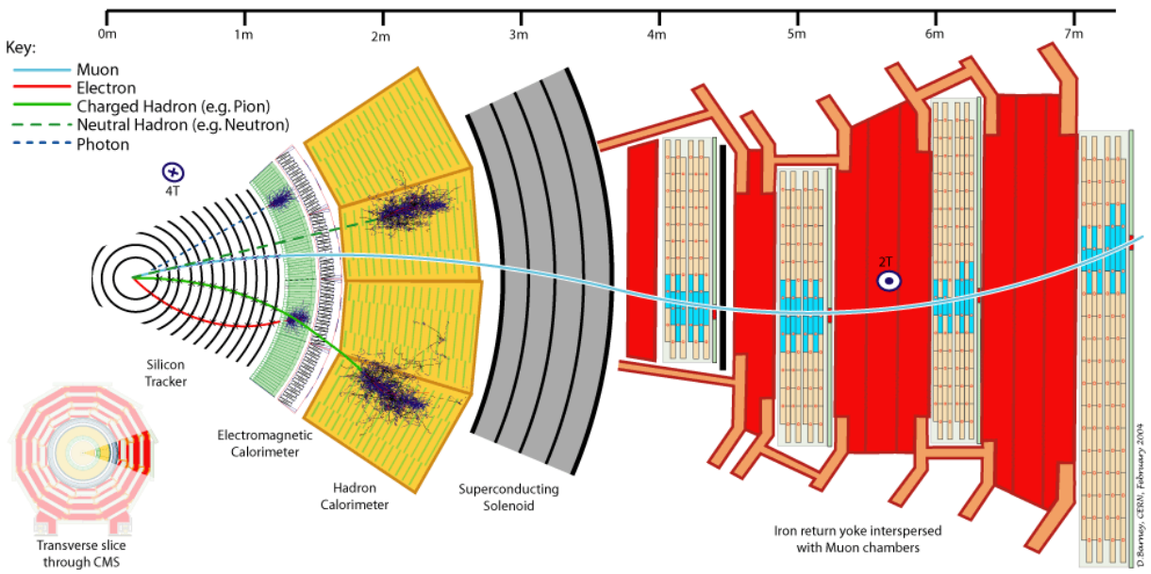


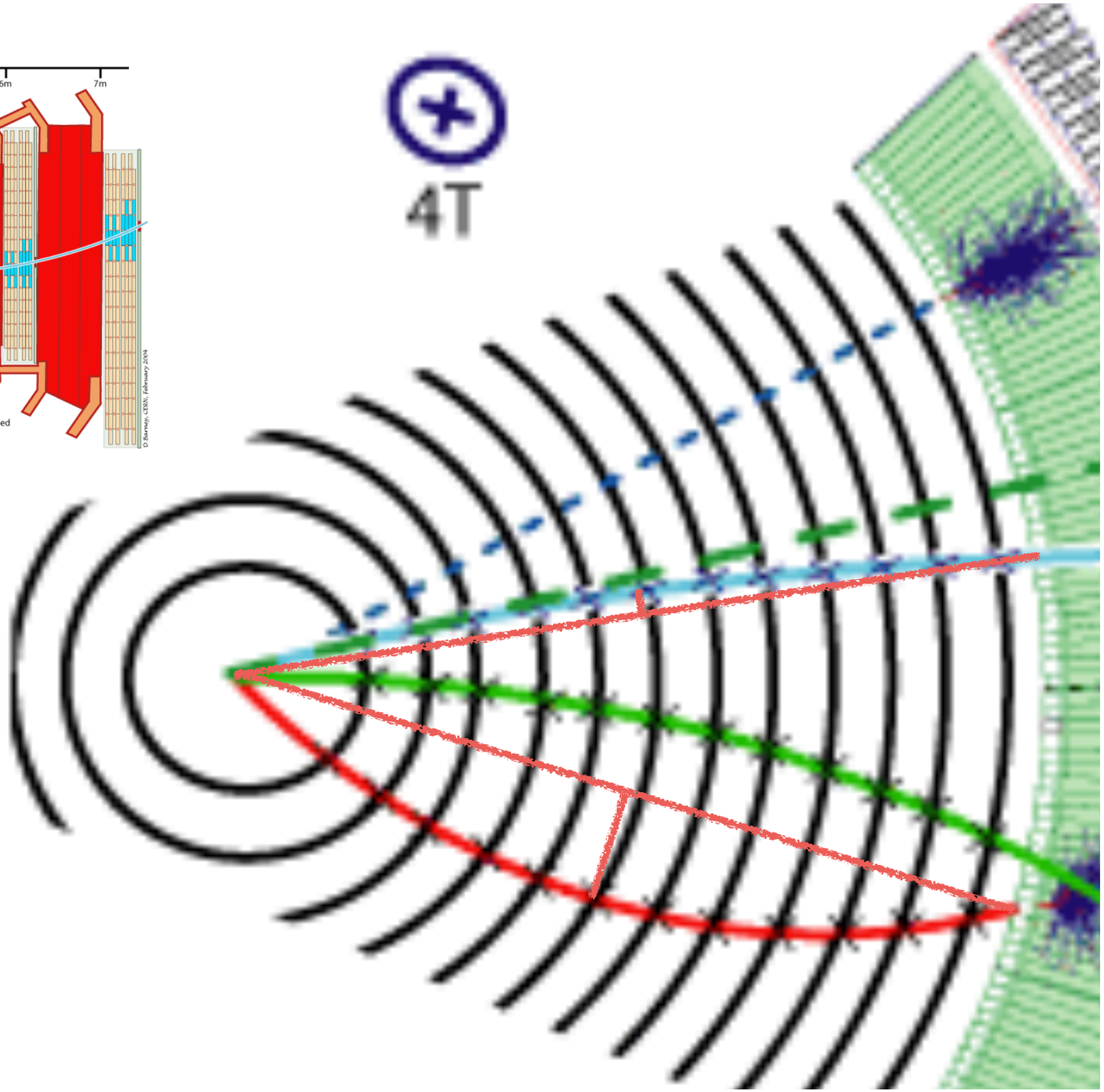


Exercise: find pT of e and μ



- Recall the wedge chart



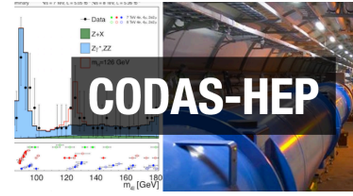


- Simple relationship to pT

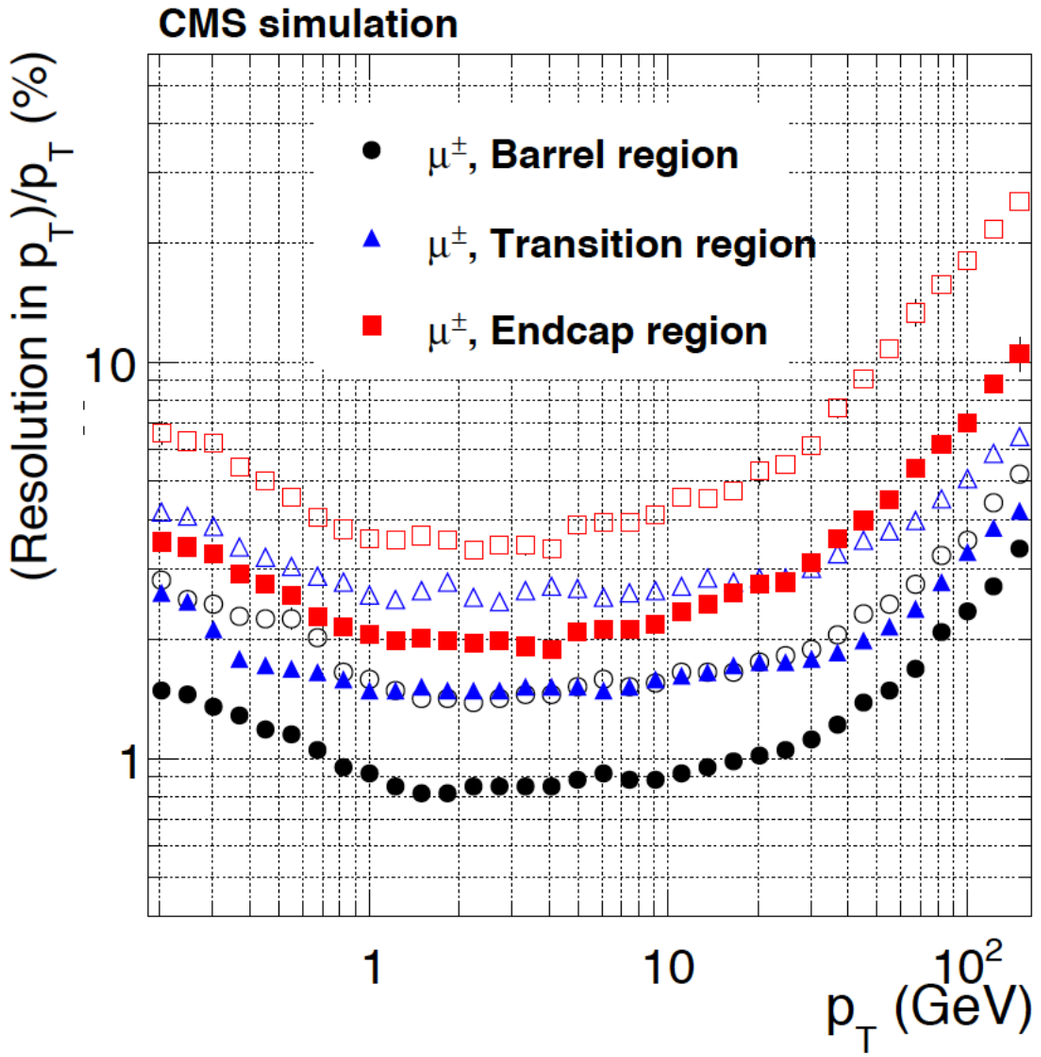
$$p_T \approx p_{T,d}\frac{d}{4s}$$

- muon: d/s ~ 30
- electron: d/s ~ 7.3

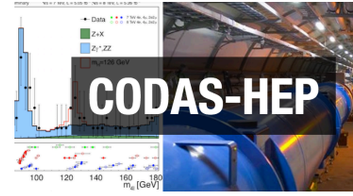
Tracking: performance



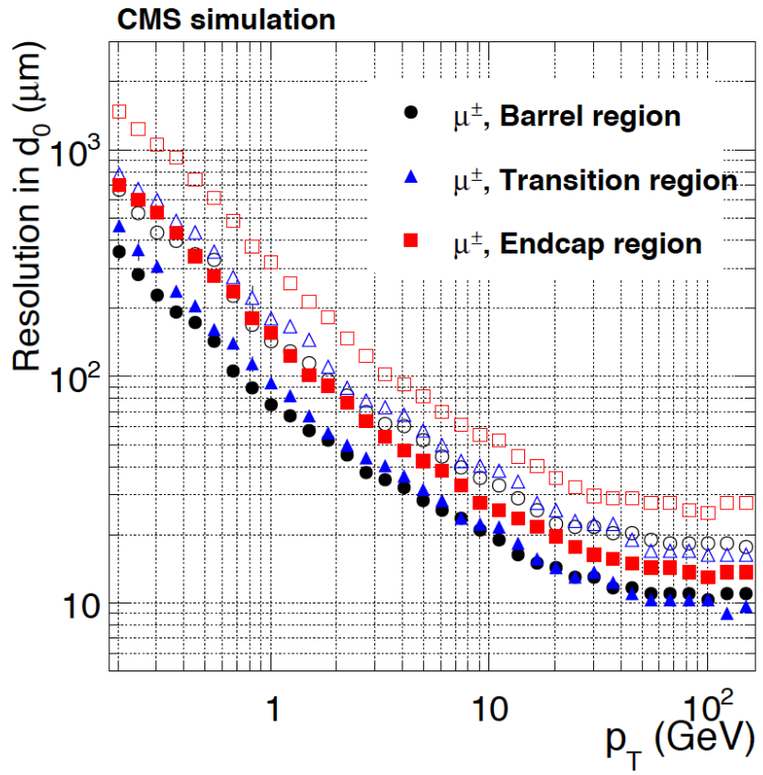
- Momentum resolution degrades with p_T
- Question: at which p_T does it go to ~100%?
  - **Hint: use point position resolution of ~100 μm**



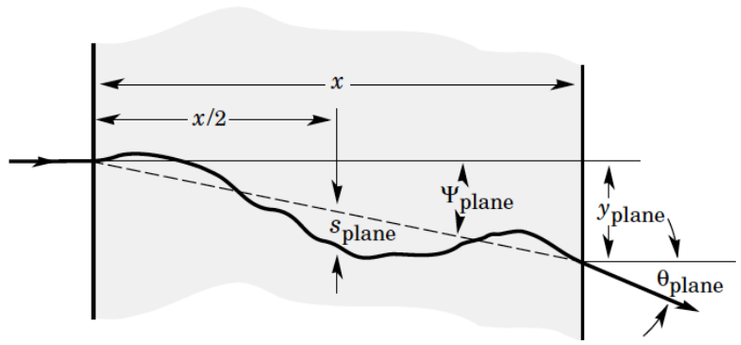
Tracking: performance



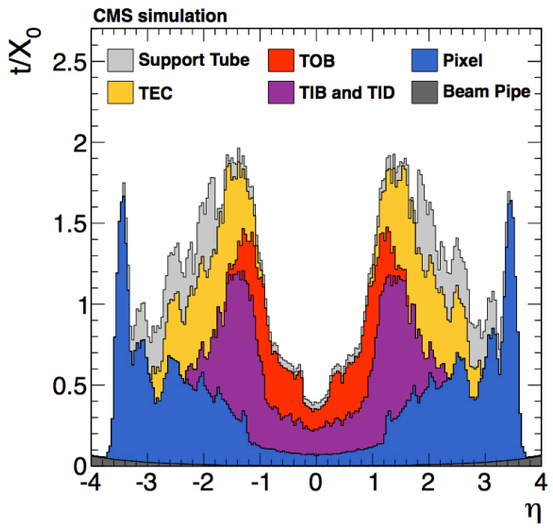
- Impact parameter resolution increases with decreasing p_T
  - ✓ **Limited by hit resolution and alignment at high end**
  - ✓ **Limited by multiple scattering at low end**

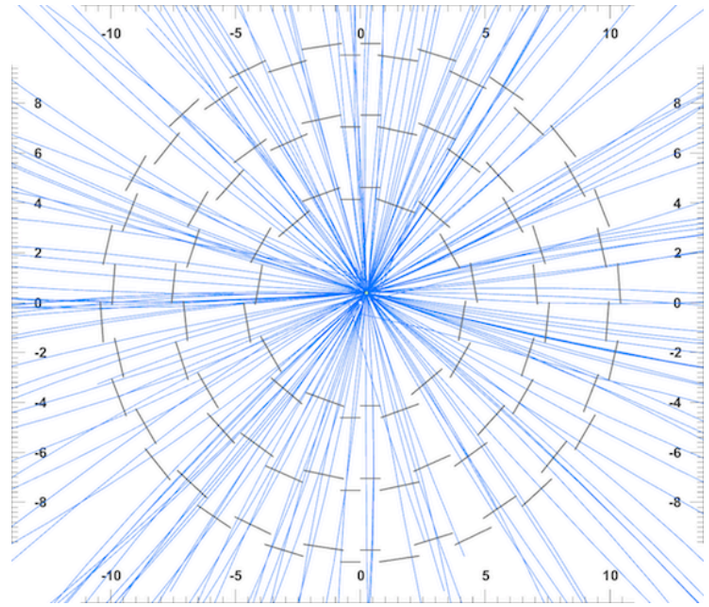


- Recall: multiple scattering



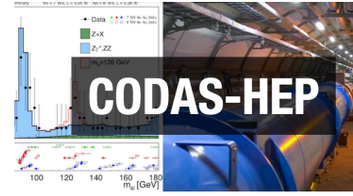
$$\theta_{\text{plane}} \approx \frac{14\text{MeV}}{p}\sqrt{x/X_0}$$





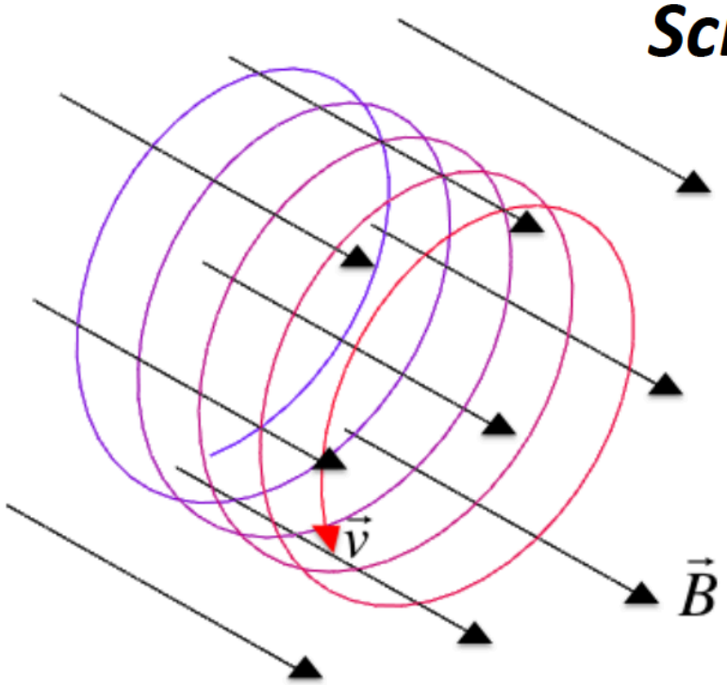
- Homework: find the fraction of the d_0 resolution due to scattering on the innermost Si sensor (0.3 mm) for muon with η~0 and p_T of 1 GeV

Kalman Filter



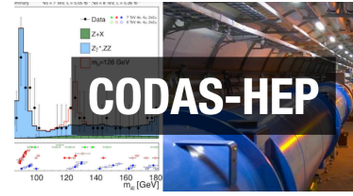
Science Fiction

- We use Kalman-Filter based tracking. Here is why.
- Naively, the particle's trajectory is described by a single helix



- …

Kalman Filter

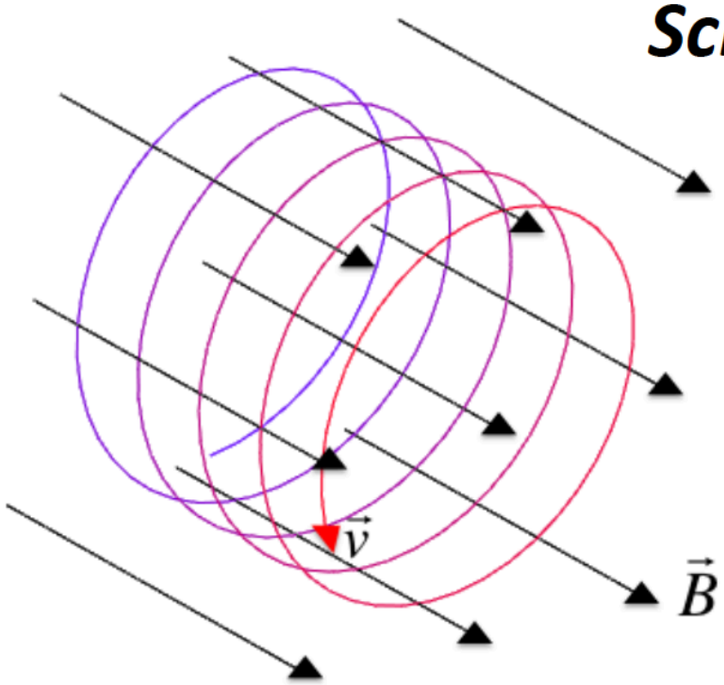


- We use Kalman-Filter based tracking. Here is why.
- Naively, the particle's trajectory is described by a single helix
- Forget it: the B-field is not really uniform, but there are also random processes changing the trajectory
  - **scattering**
  - **energy loss**

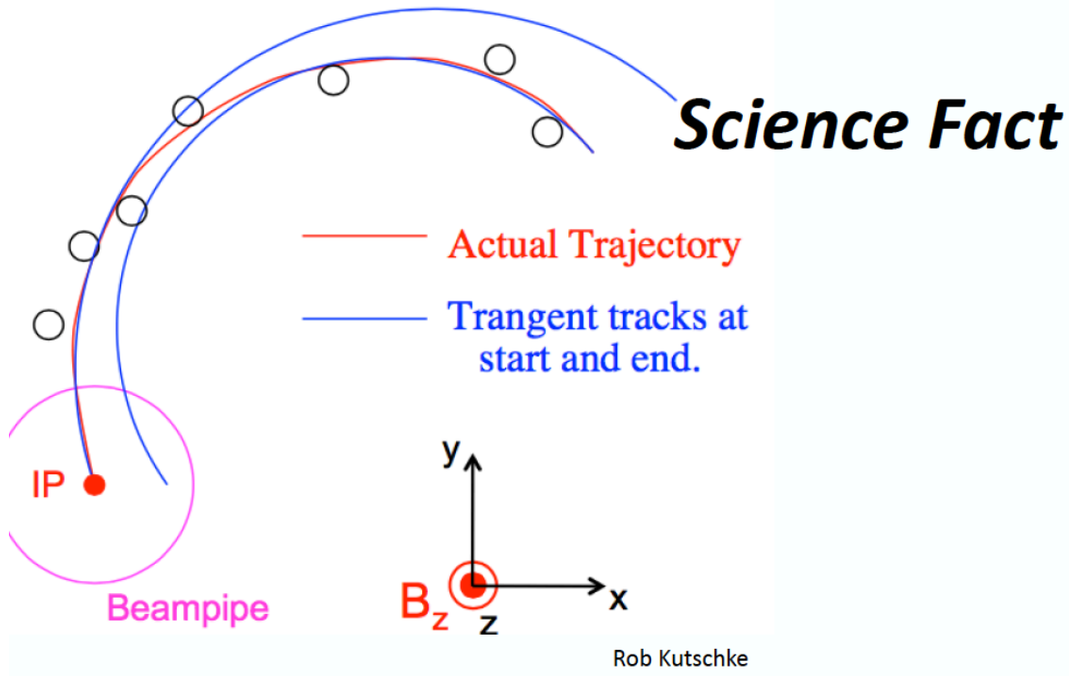
➡ **trajectory is only locally helical**

- Kalman Filter allows us to take these effects into account, while preserving a locally smooth trajectory

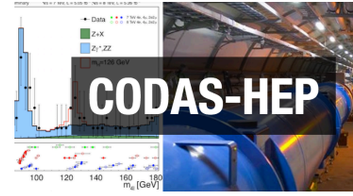
Science Fiction



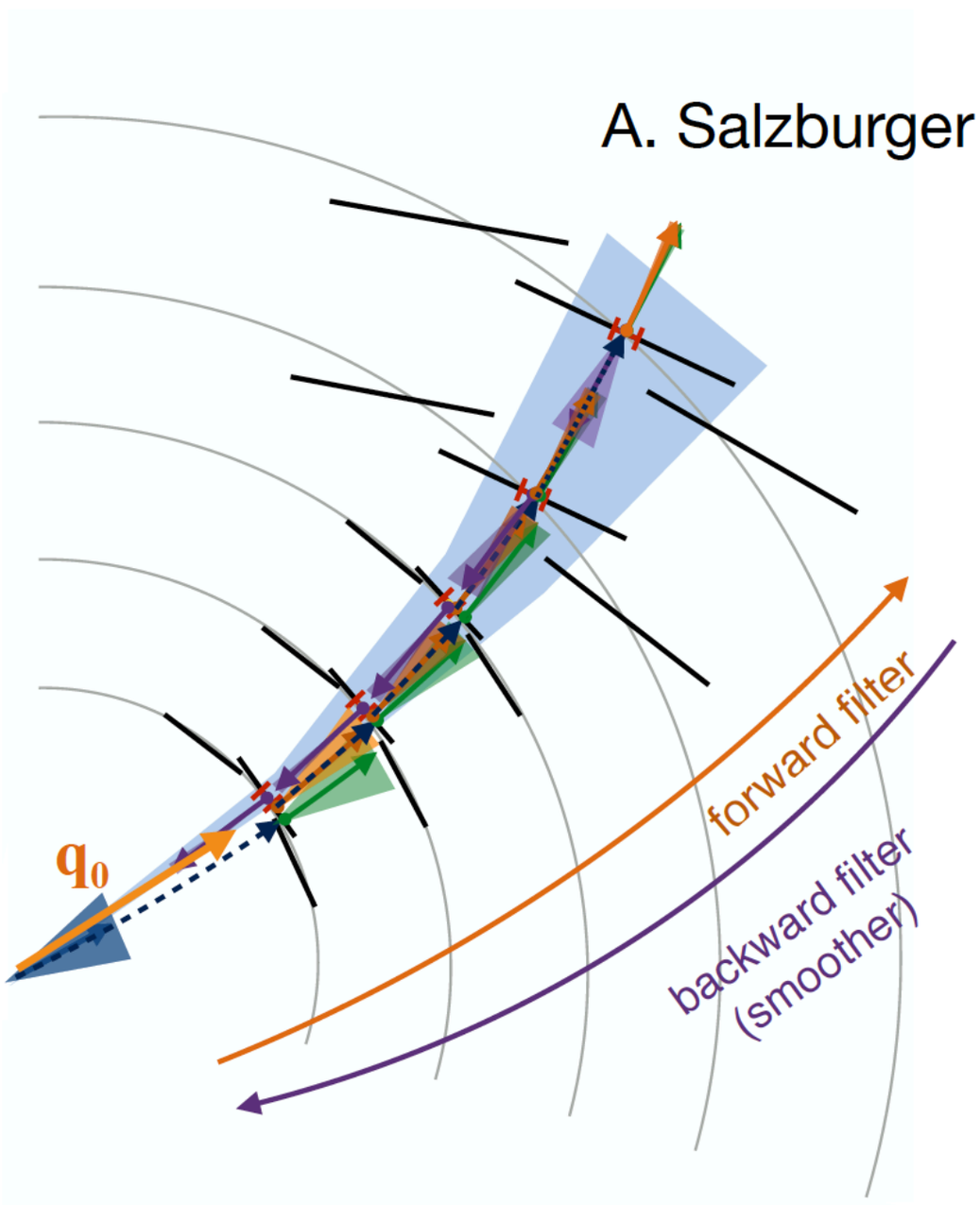
Science Fact



Kalman Filter



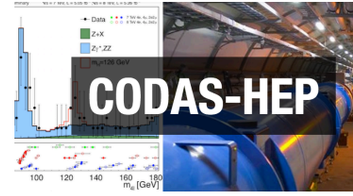
- Initially developed by R. Kalman to track missiles
- Pioneered by Billoir and R. Fruehwirth for HEP
- Progressive least square estimation
  - ✓ **equivalent to a χ2 fit (if run with a smoother)**



- Start with transport of track parameters (and covariances) to measurement surface, create predicted parameters ("predicted state")
- Combine/update predicted parameters with measurement to updated parameters ("filtered state")

R. Frühwirth, Nucl. Instr. Meth. A 262, 444 (1987), DOI:10.1016/0168-9002(87)90887-4

Kalman Filter: formalism



- https://en.wikipedia.org/wiki/Kalman_filter (it's that easy to find)

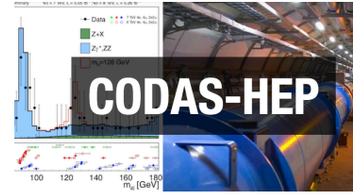
Predict [edit]

Predicted (*a priori*) state estimate	$\hat{\mathbf{x}}_{k\mid k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1\mid k-1} + \mathbf{B}_k\mathbf{u}_k$
Predicted (*a priori*) error covariance	$\mathbf{P}_{k\mid k-1} = \mathbf{F}_k\mathbf{P}_{k-1\mid k-1}\mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k$

Update [edit]

Innovation or measurement pre-fit residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\mid k-1}$
Innovation (or pre-fit residual) covariance	$\mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k\mathbf{P}_{k\mid k-1}\mathbf{H}_k^{\mathrm{T}}$
Optimal Kalman gain	$\mathbf{K}_k = \mathbf{P}_{k\mid k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1}$
Updated (*a posteriori*) state estimate	$\hat{\mathbf{x}}_{k\mid k} = \hat{\mathbf{x}}_{k\mid k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
Updated (*a posteriori*) estimate covariance	$\mathbf{P}_{k\mid k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k\mid k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T$
Measurement post-fit residual	$\tilde{\mathbf{y}}_{k\mid k} = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\mid k}$

The formula for the updated (*a posteriori*) estimate covariance above is valid for any gain $\mathbf{K}_k$ and is sometimes called the **Joseph form**. For the optimal Kalman gain the formula further simplifies to $\mathbf{P}_{k\mid k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k\mid k-1}$, in which form it is most widely used in applications. However, one must keep in mind, that it is valid only for the optimal gain that minimizes the residual error. Proof of the formulae is found in the



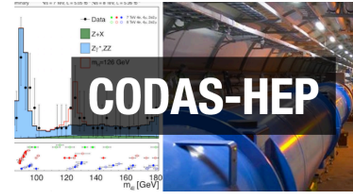
KF tracking overview

- Seeding.
  - ✓ **Select hits to get an initial track candidate.**
  - ✓ **Sort candidates by some criterion. Criteria that lead to higher quality tracks are tried first, then progressively less stringent criteria**
  - ✓ **Each hit can only be used once**

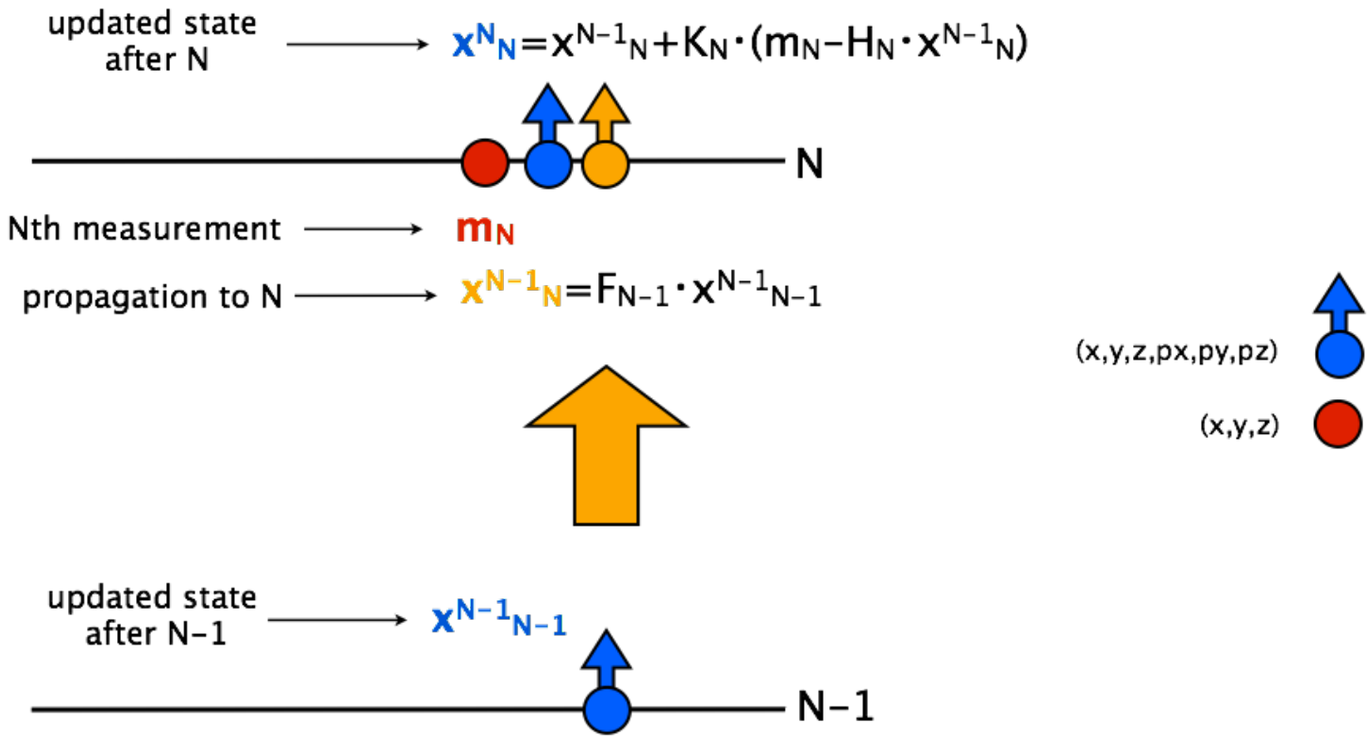
- Main tracking loop over track candidates
  - ✓ **Propagate each helix to next detector layer, taking into account the uncertainty of the current estimate and the amount of material in the way**
  - ✓ **Look for hits in the next layer consistent with the current candidate track**
    - ◉ **update the track parameters to include new hit (Kalman)**
  - ✓ **Remove hits from list of available hits**
  - ✓ **Repeat Step 2 until iterated over all layers, removing used hits and updating track parameters as we go along**
  - ✓ **Once all hits are attached re-fit final track parameters to get best estimate**

- Return to Seeding step and generate new seeds on the remaining hits

KF tracking overview

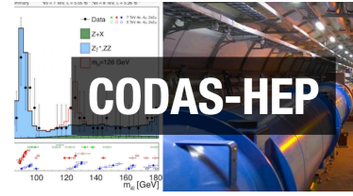


- KF track reconstruction can be divided into 2 main steps: building, and fitting.
- Both track building and track fitting are based on Kalman Filter.

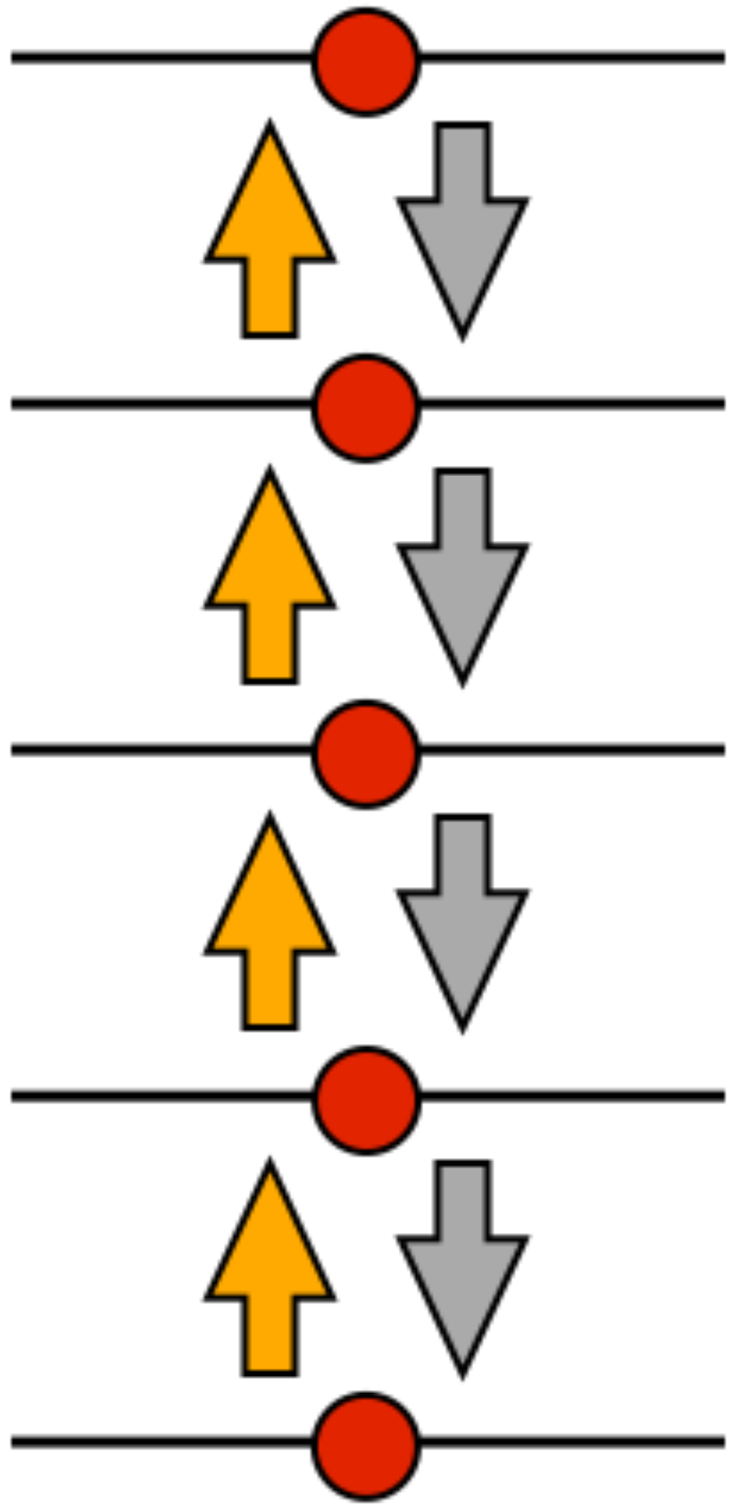


- The Kalman Filter is an iterative procedure of a basic logic unit consisting of the propagation of parameters and uncertainties (track state) from a layer to the next one, where the track state is updated (filtered) with the hit measurement information.

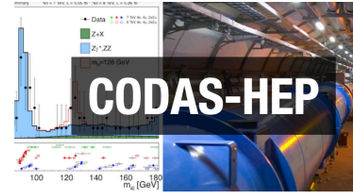
Kalman Track Fitting



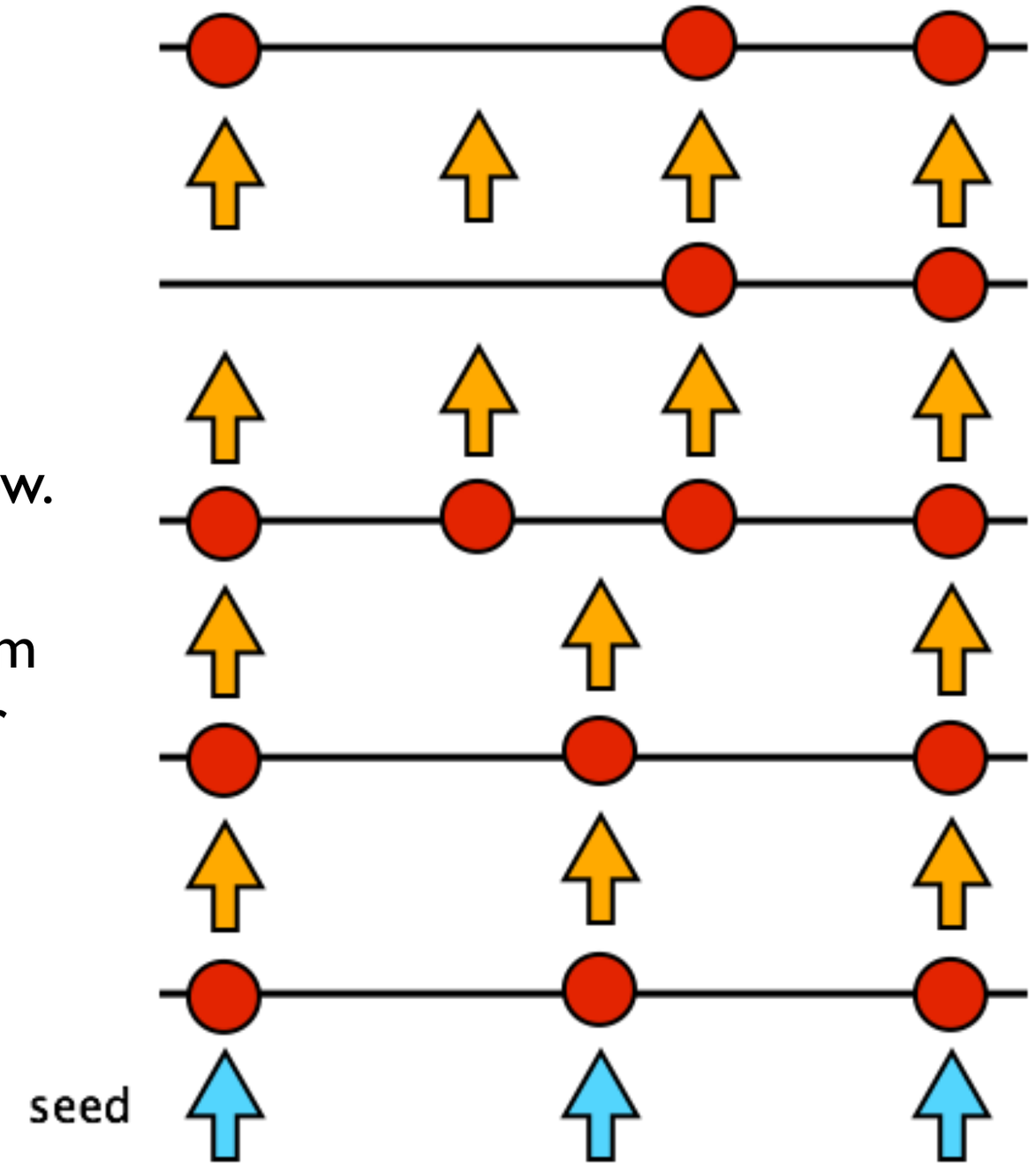
- The track fit consists of the simple repetition of the basic logic unit for all the pre-determined track hits
- It is divided in two steps
  - ✓ **a forward fit**
  - ✓ **a backward smoothing stage**
- Forward fit: best estimate at interaction point
- Smoothing stage: best estimate at face of calorimeter
- Computationally, the Kalman filter is a set of matrix operations with small matrices (dimension 6 or less)



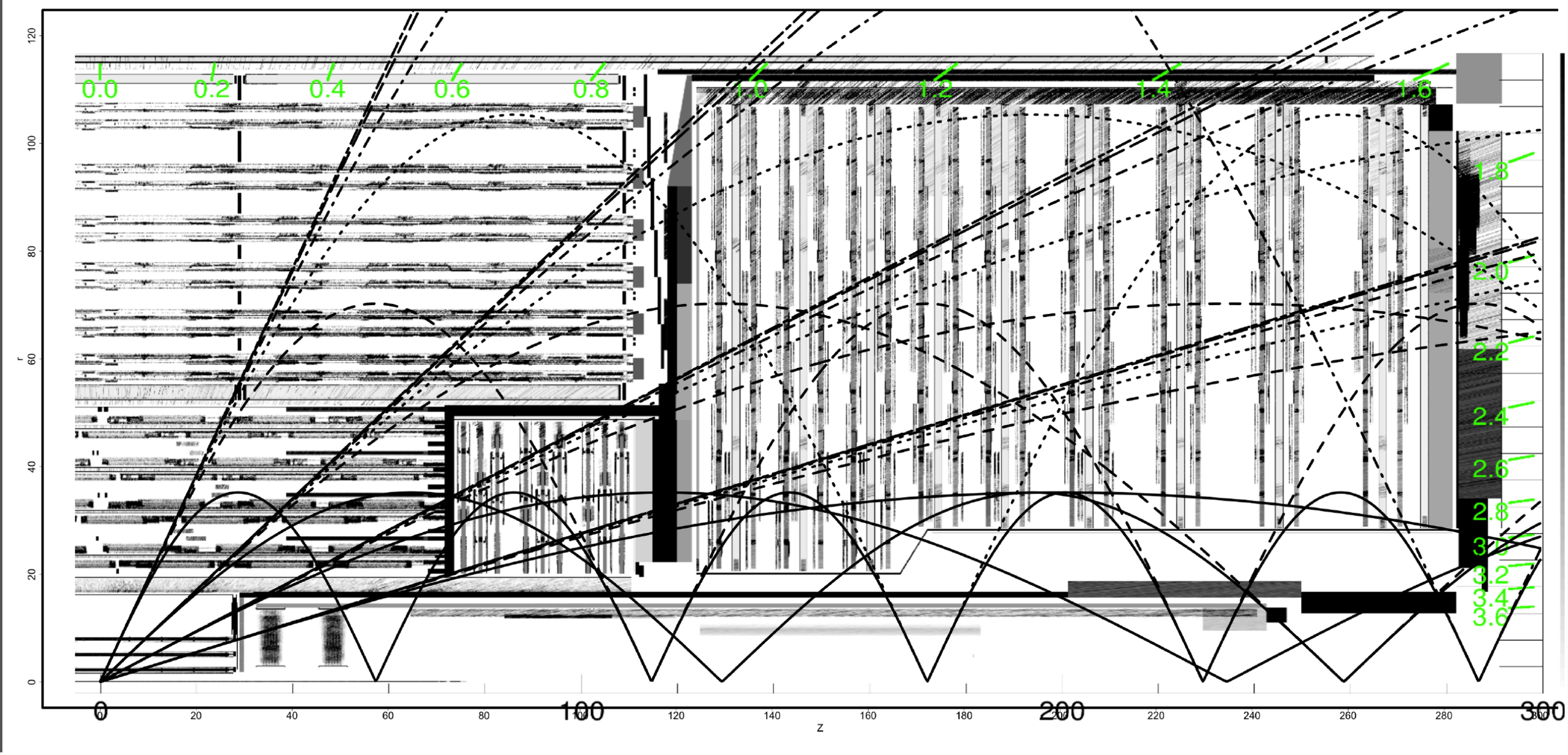
Track Building



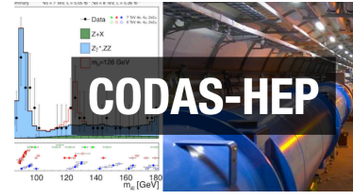
- Track building adds complexity to the problem.
- When moving to the next layer, hits are searched for within a compatibility window.
- The track candidate needs to branch in case of multiple matches and the algorithm needs to be robust against missing/outlier hits.
- Track Building is by far the most time consuming step in the whole event reconstruction



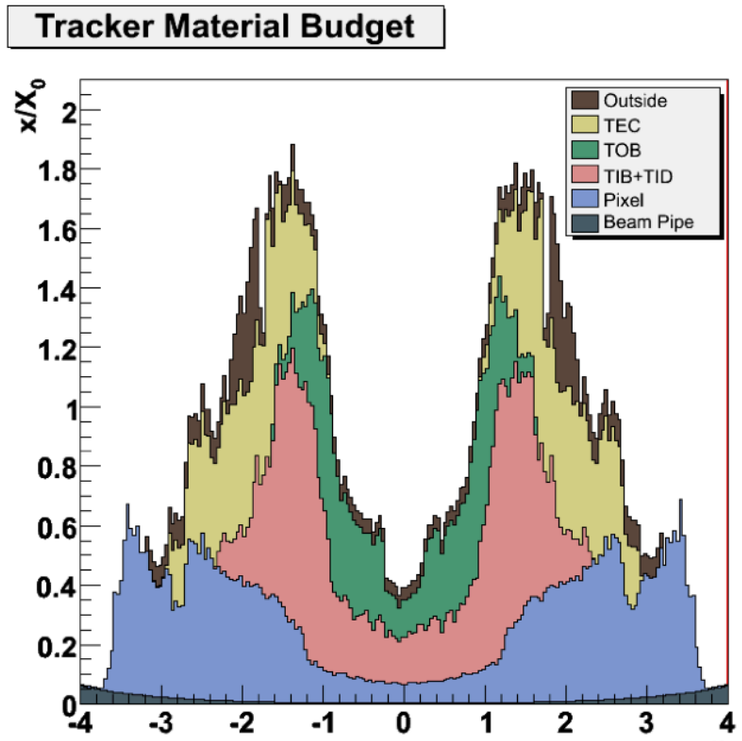
Tracks in real detector

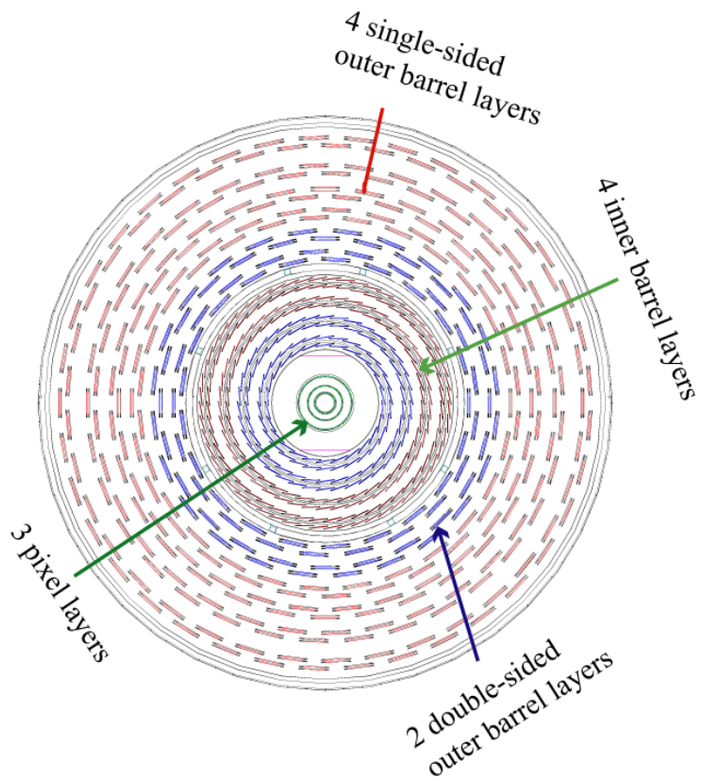


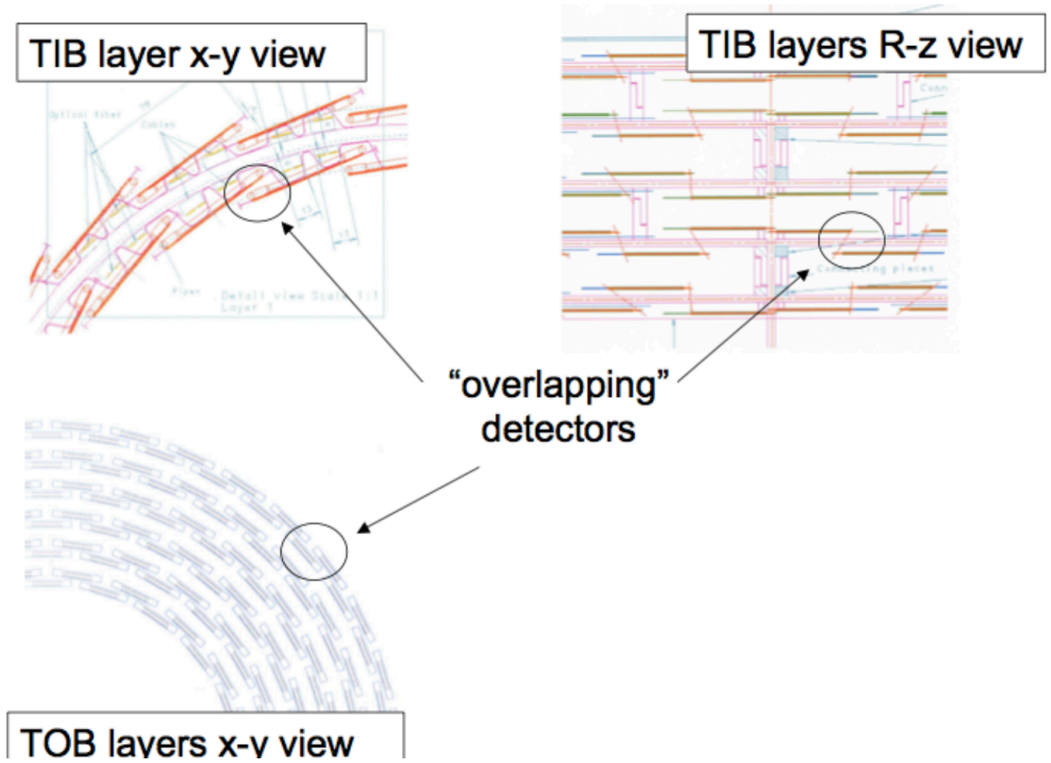
Things to keep in mind

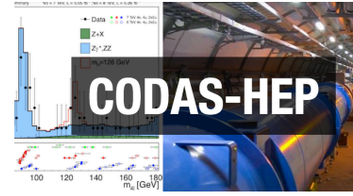


- Geometry is complicated and has no symmetries, even before accounting for alignment (diff btw ideal and real geometry)
  - ✓ **no "circular cows"**
- Material maps are complicated, big, and not negligible
  - ✓ **Algorithm uses full information about material map to estimate multiple scattering, radiation, especially for electrons**



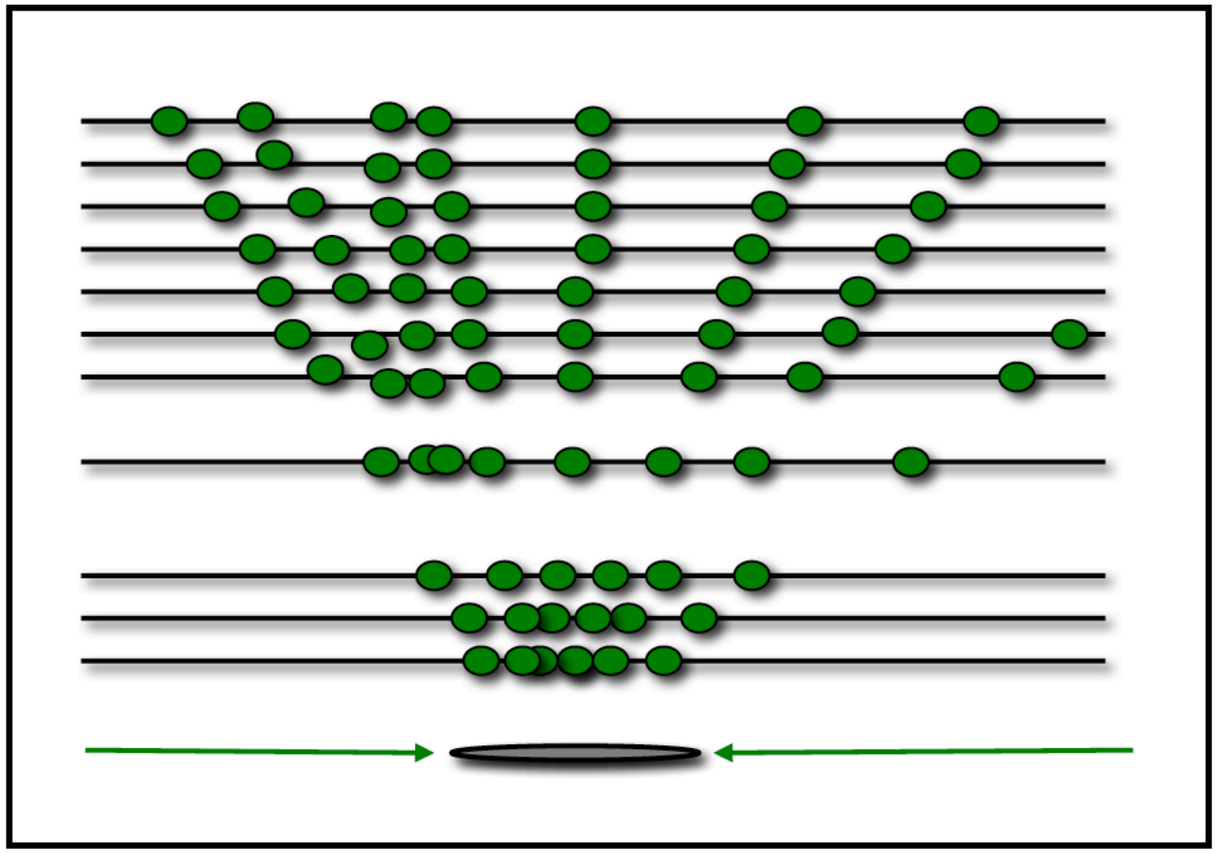




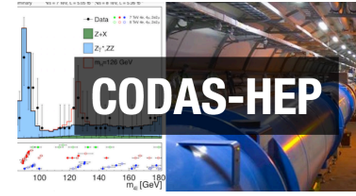


Iterative tracking

Initial collection of hits

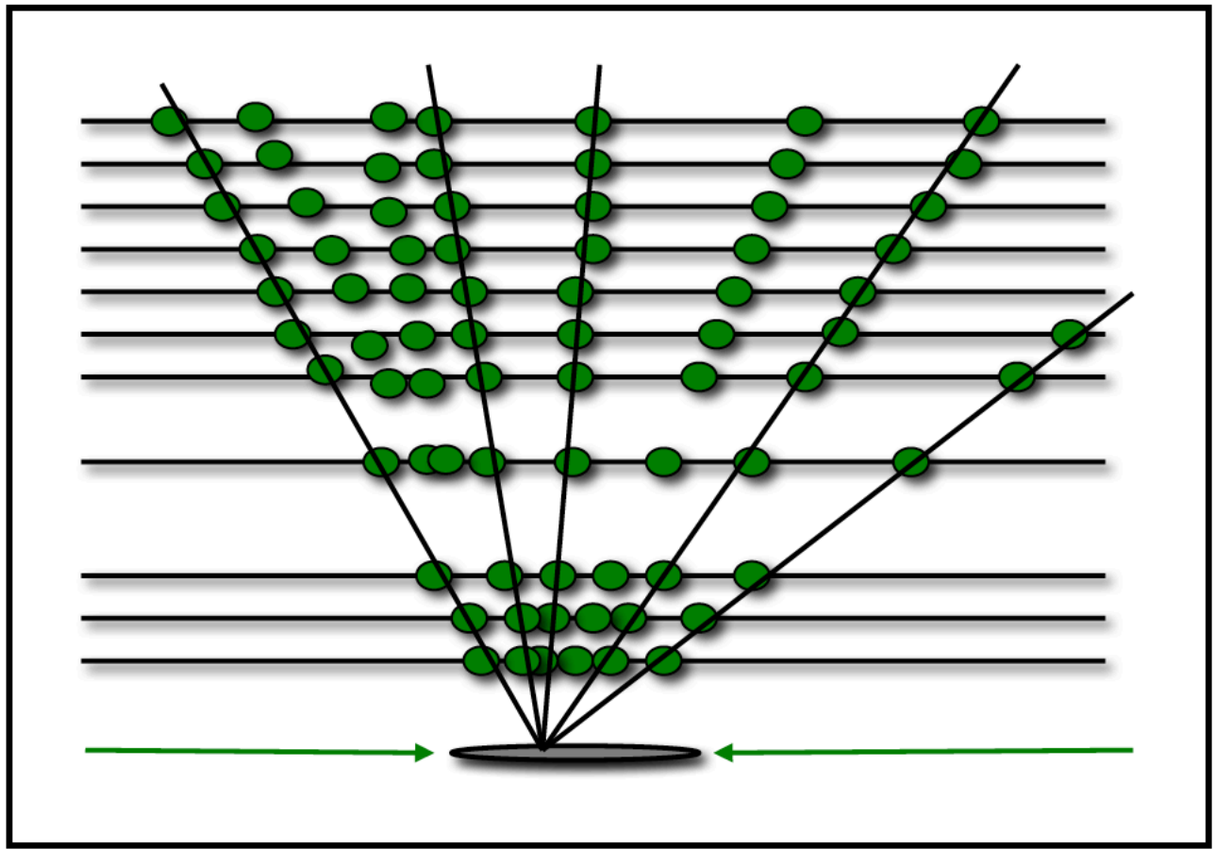


Iterative tracking

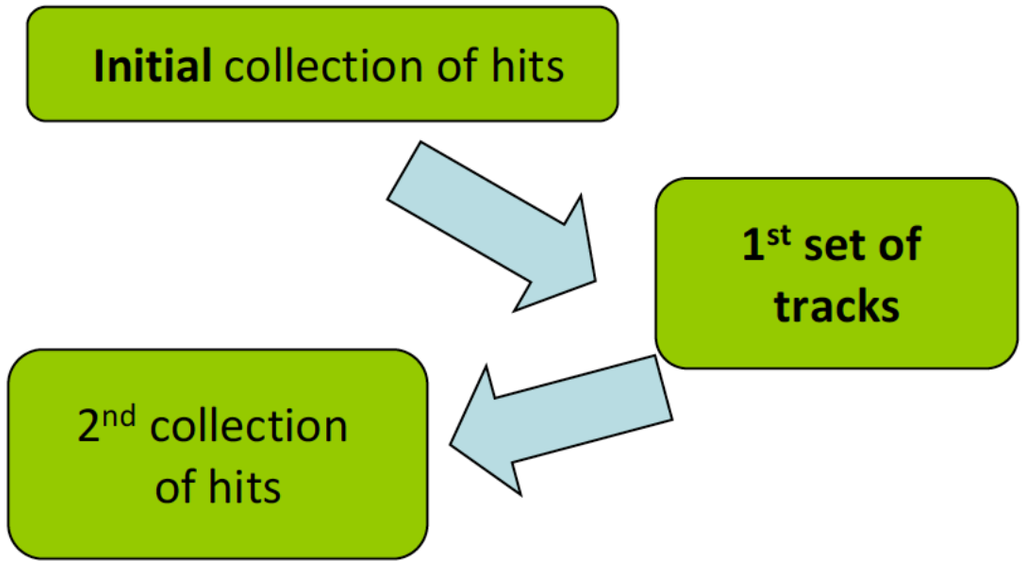


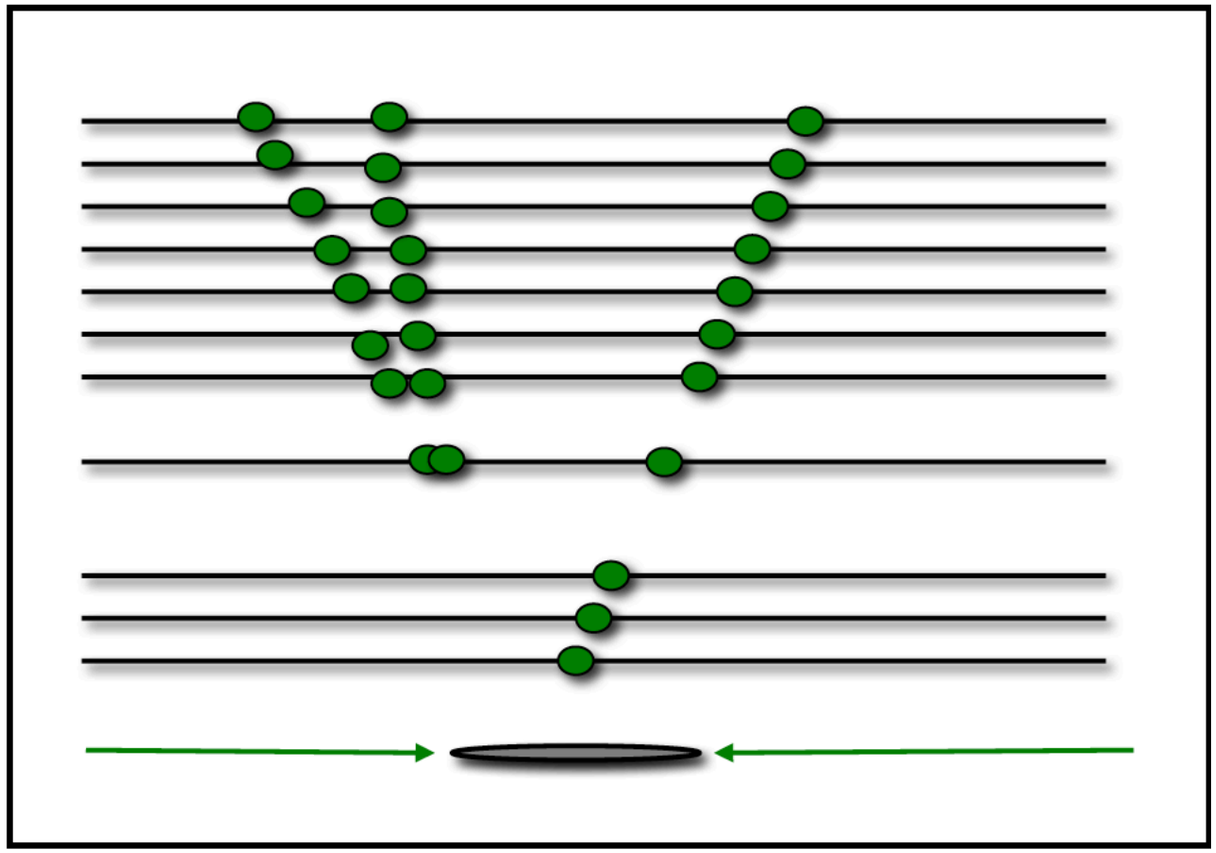
Initial collection of hits

1st set of tracks

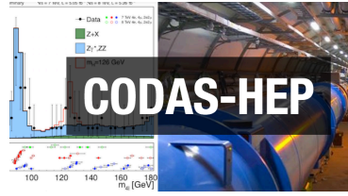


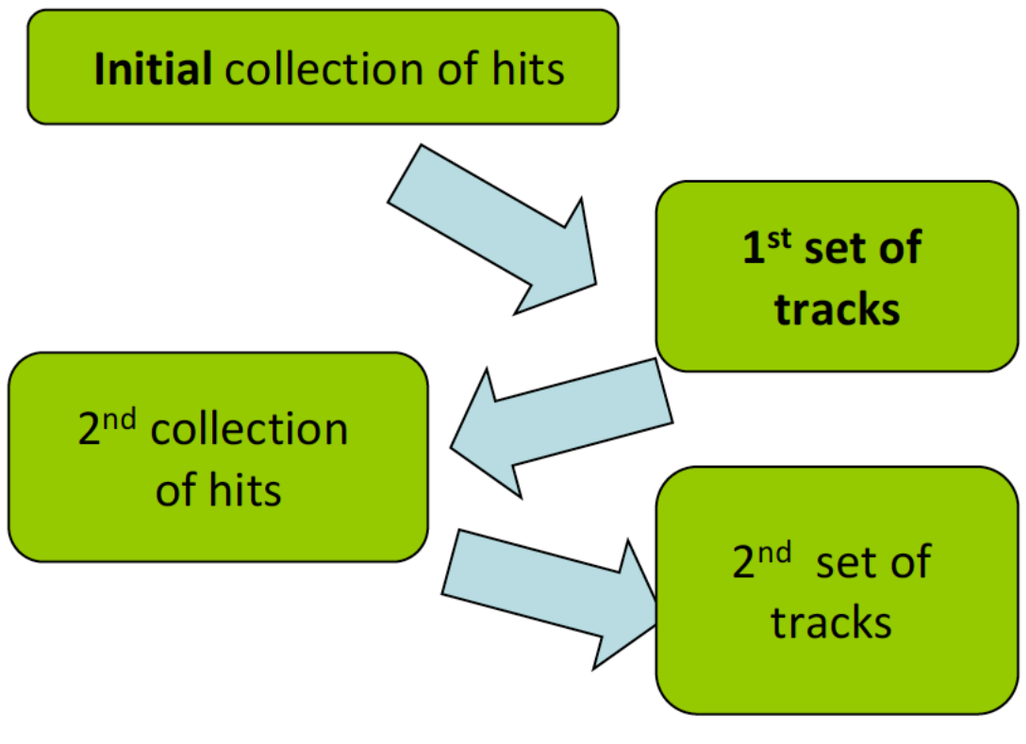
Iterative tracking

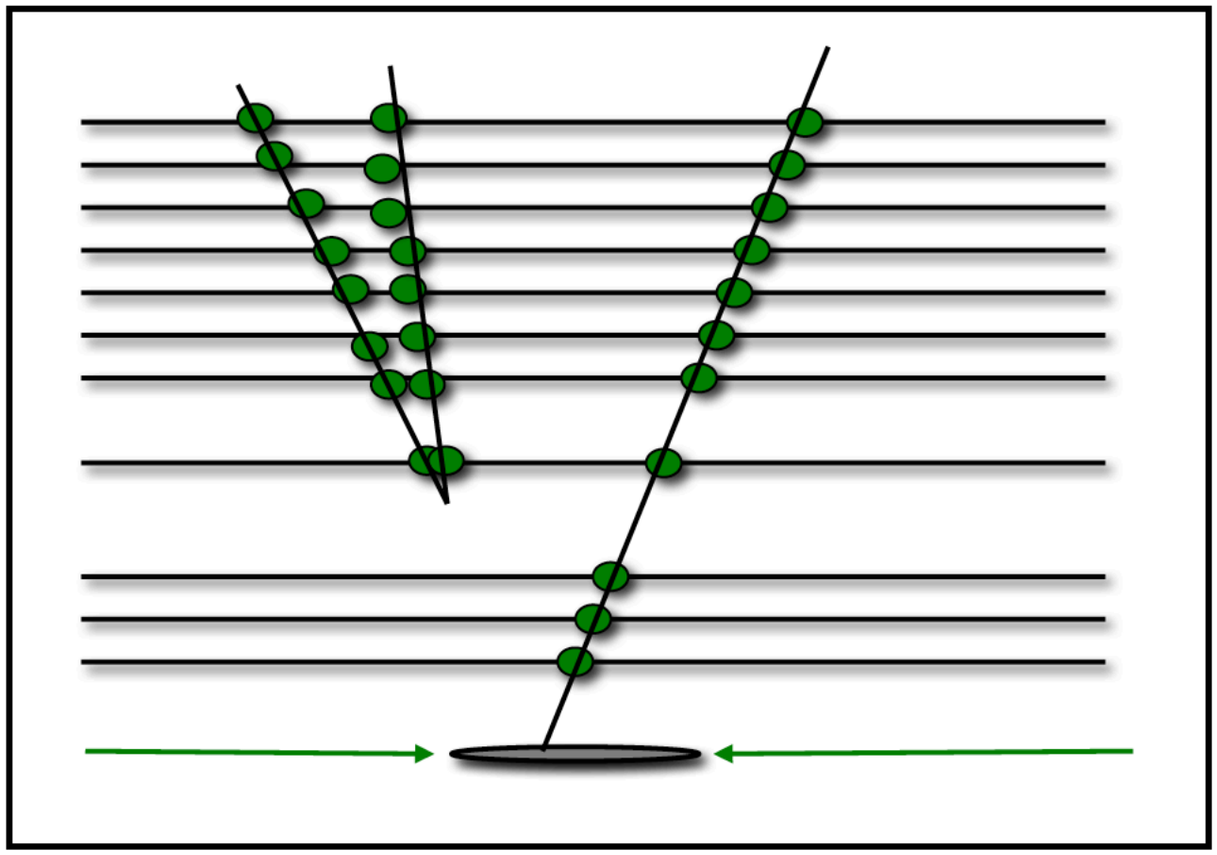




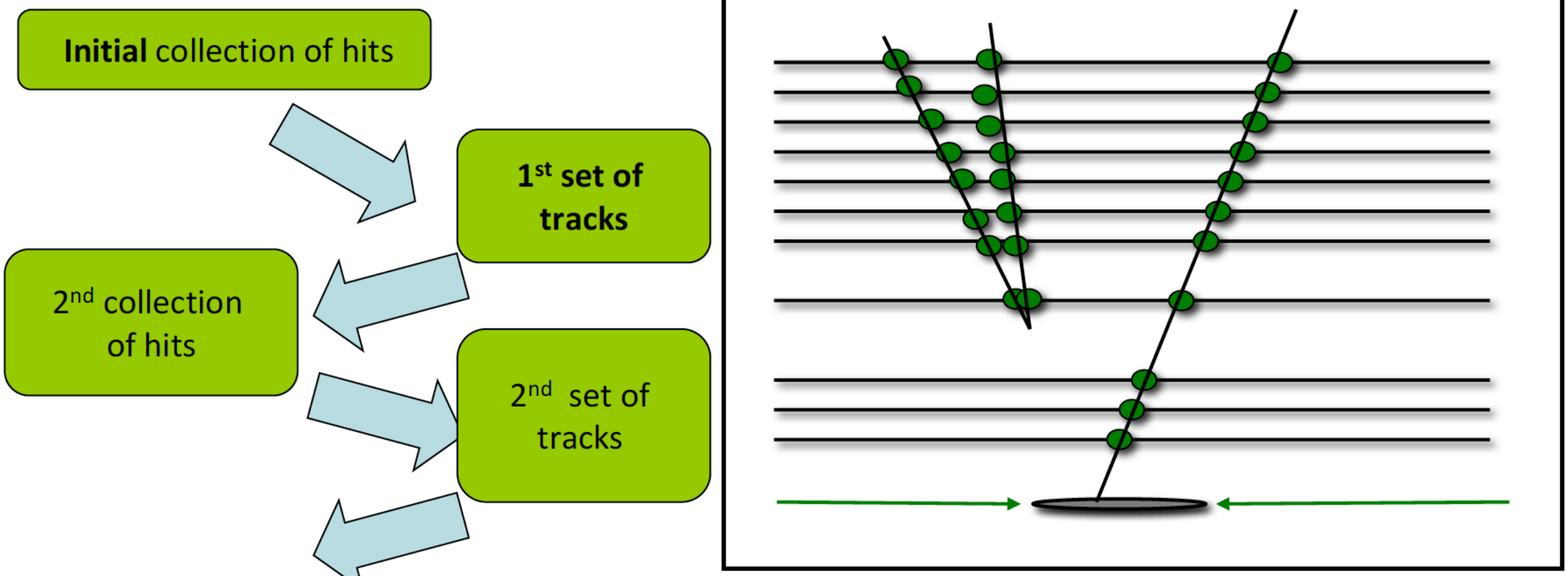
Iterative tracking



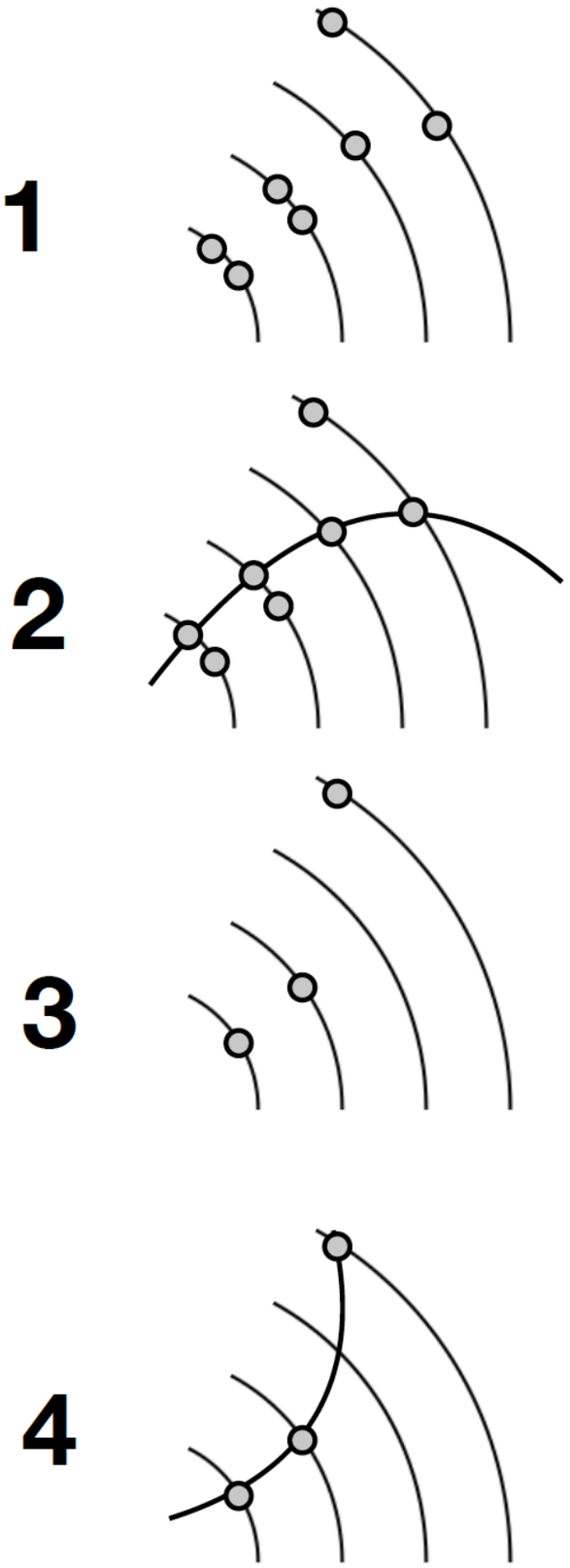


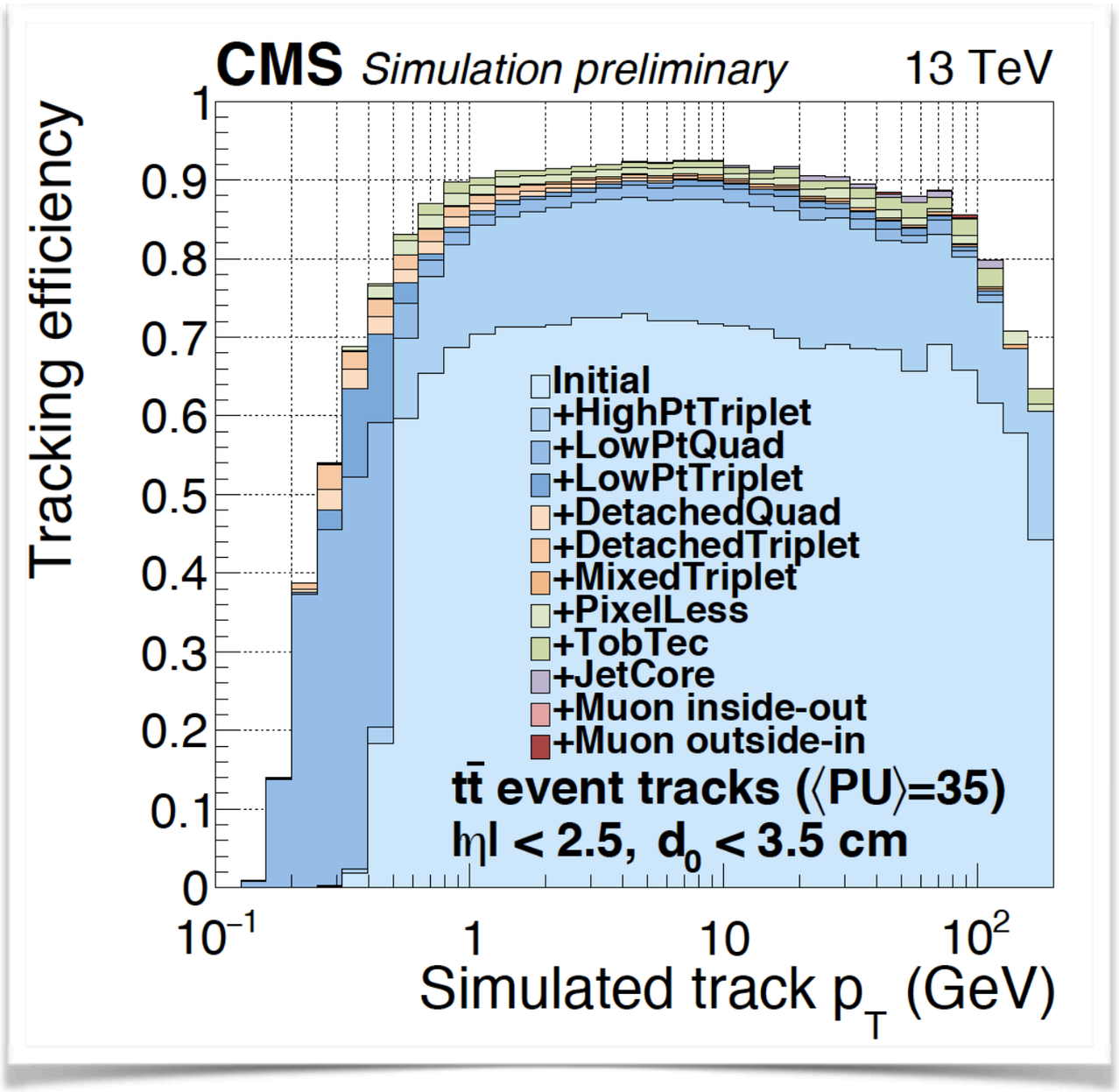


Iterative tracking

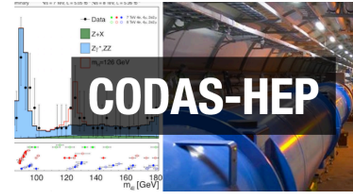


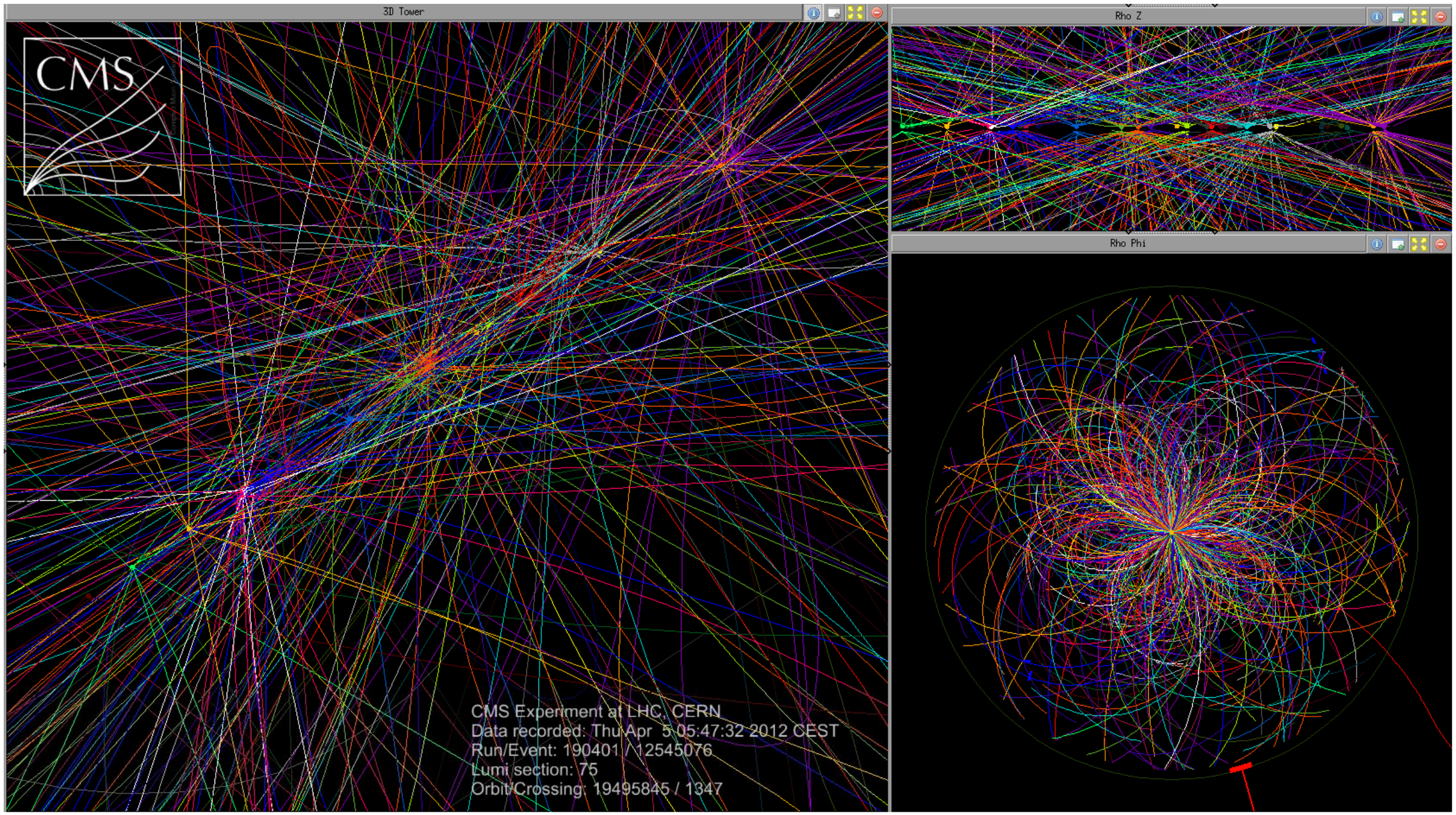
Iterative tracking: CMS





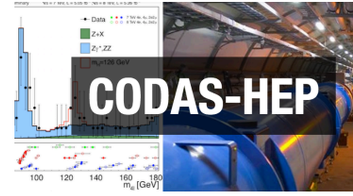
Tracking: computational challenge

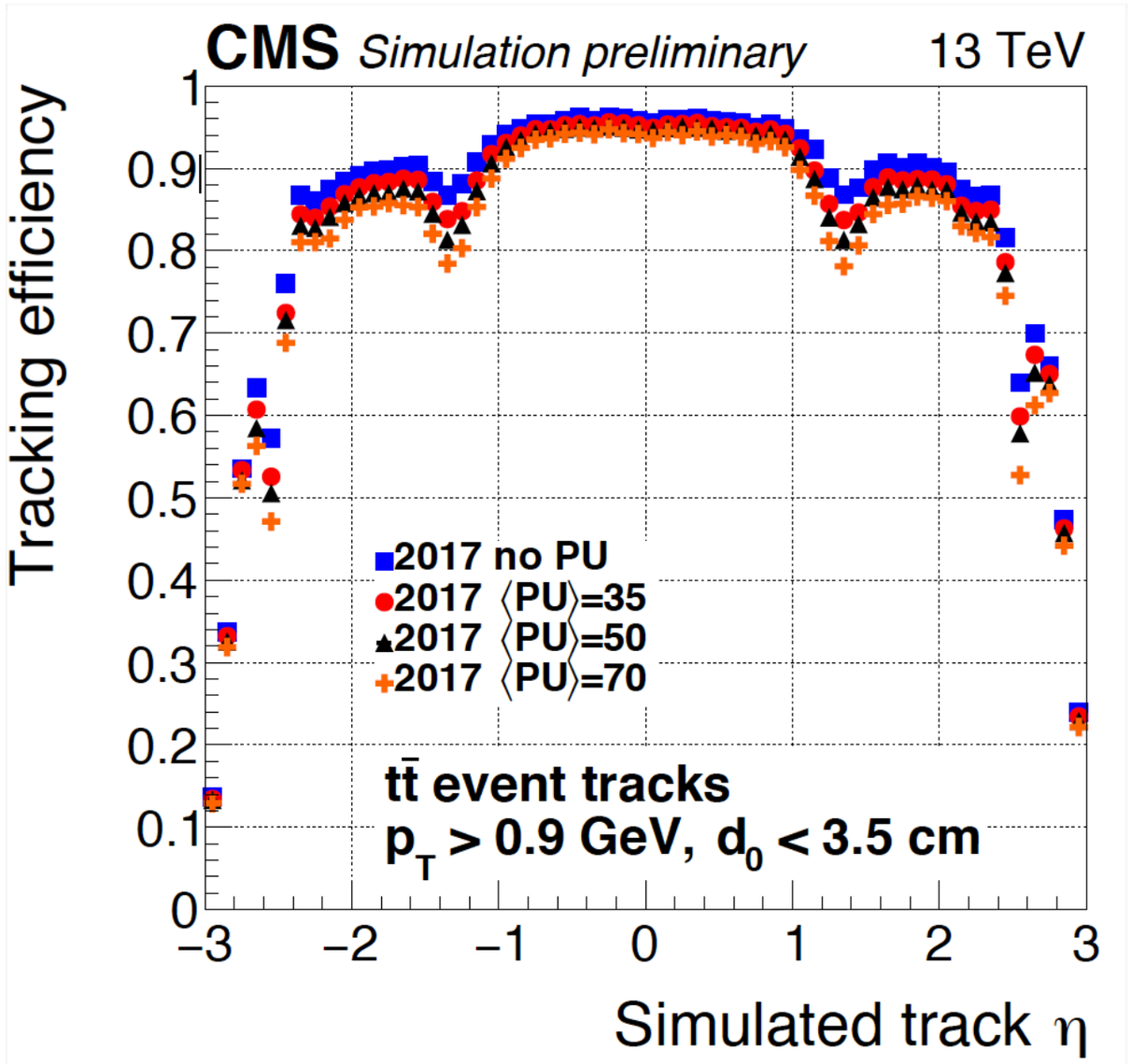


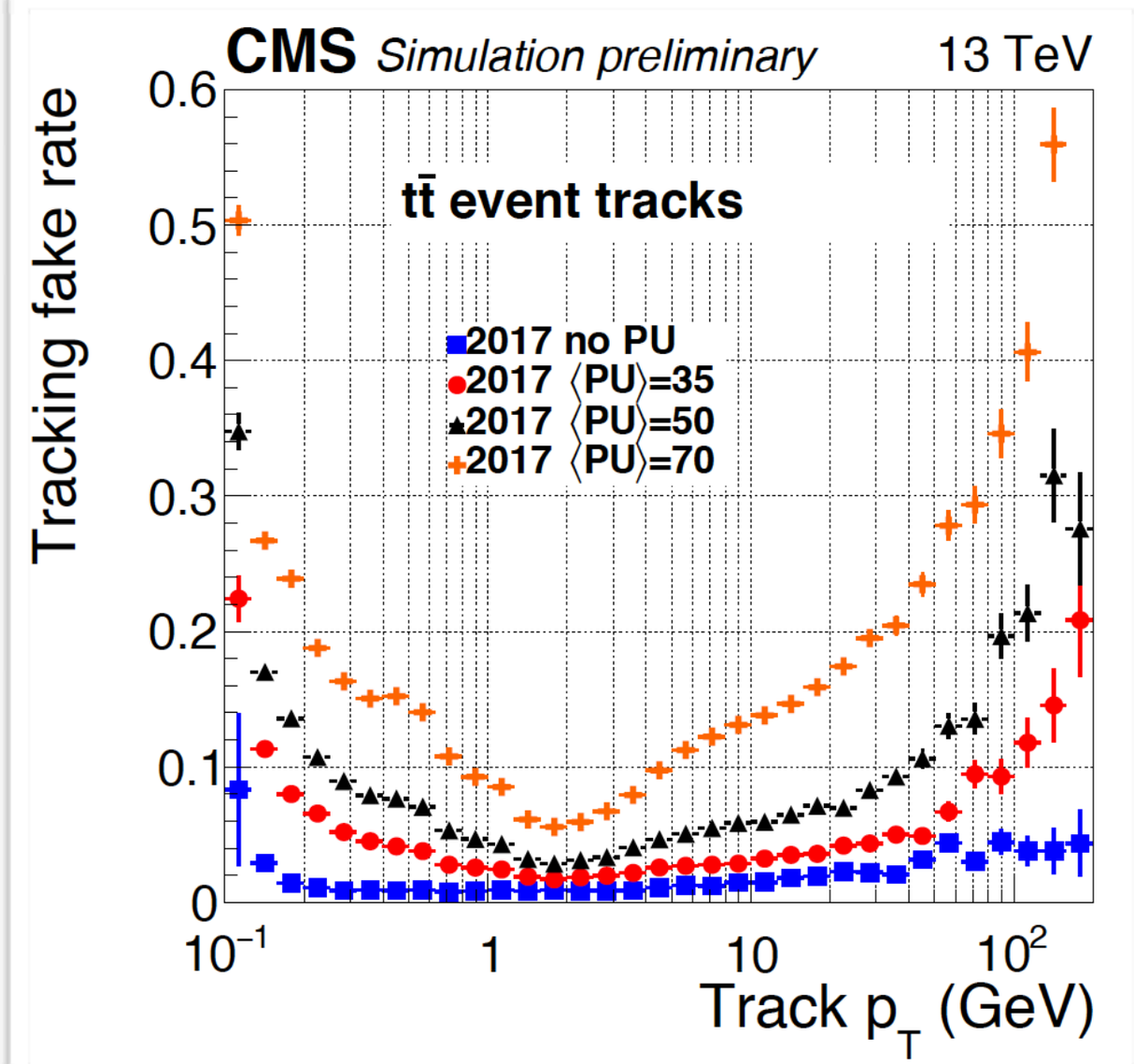


- By 2025, the instantaneous luminosity of the LHC will increase to $7.5 \times 10^{34} cm^{-2} s^{-1}$ — High Luminosity LHC (HL-LHC)
- Significant increase in number of interactions per bunch crossing ("pile-up"), on the order of 140–200 per event

Tracking with pileup

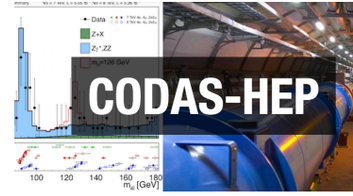






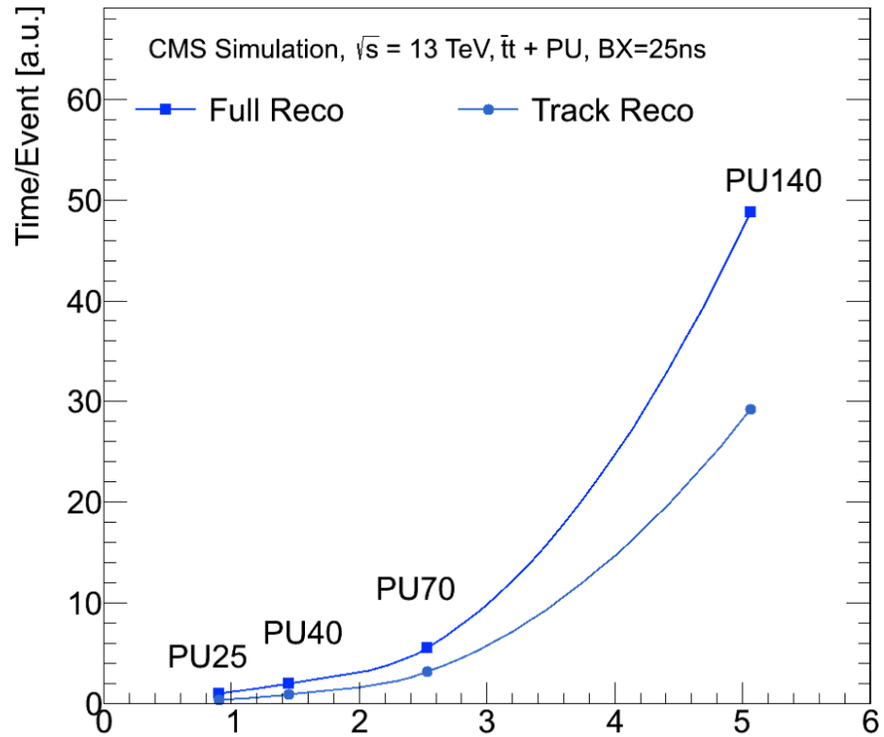
- Efficiency falls slightly and fake rates increase

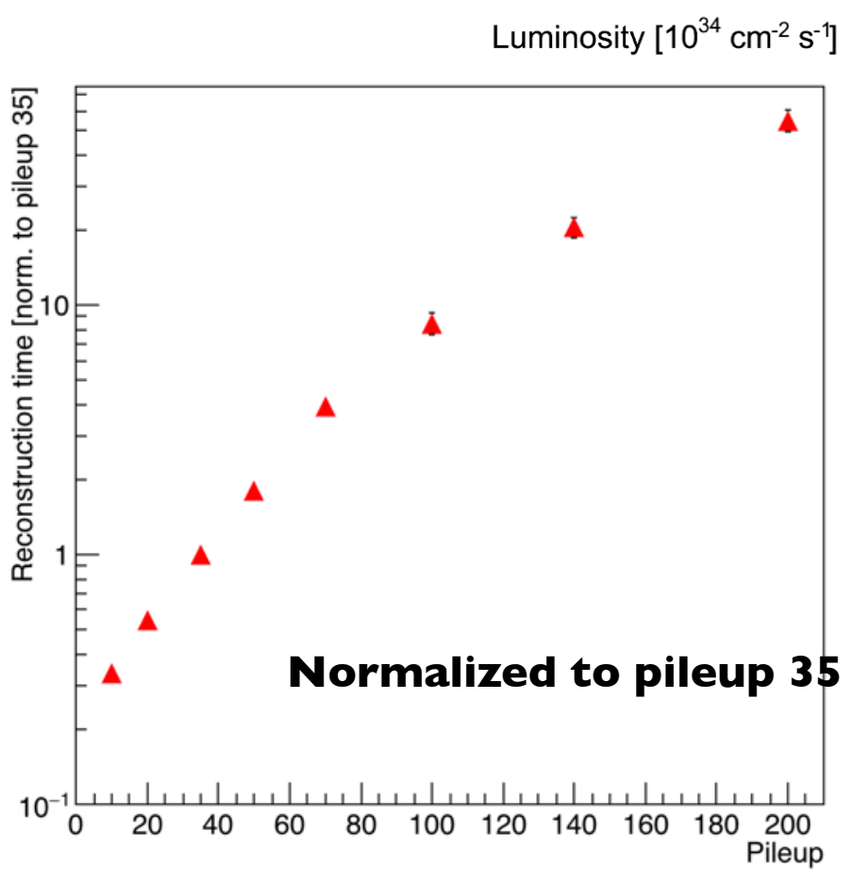
Tracking challenge with pileup



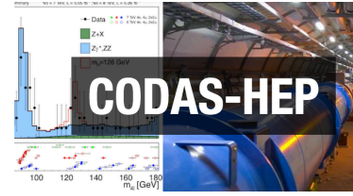
- Going from detector primitive (energy deposits in various elements) to particles: **“reconstruction”**
- Tracking is the most time-intensive part of reconstruction — combining the hits in the tracker to form the trajectories of the charged particles
- O(10^6) measurement stations per event, across many layers

- **Can we make the tracking algorithm concurrent and speed up the reconstruction?**





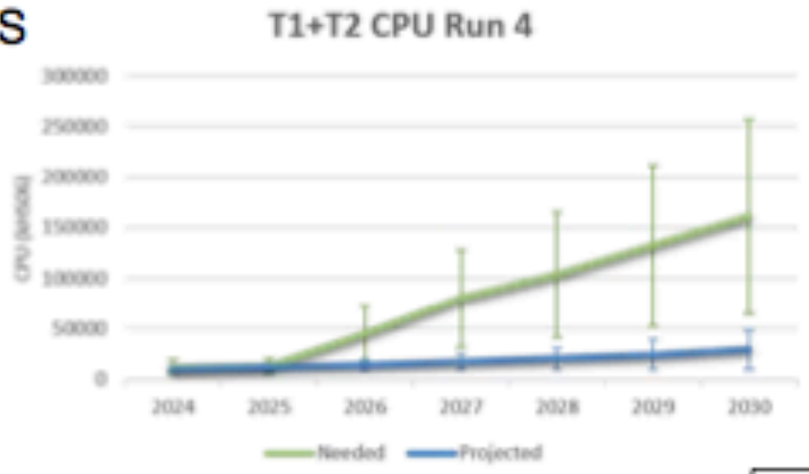
Why explore new solutions?



- Projected shortfalls in HEP computing resources
- Expectation of ~×10 shortfalls in compute by 2025

Naive extrapolation from Run 2: CPU

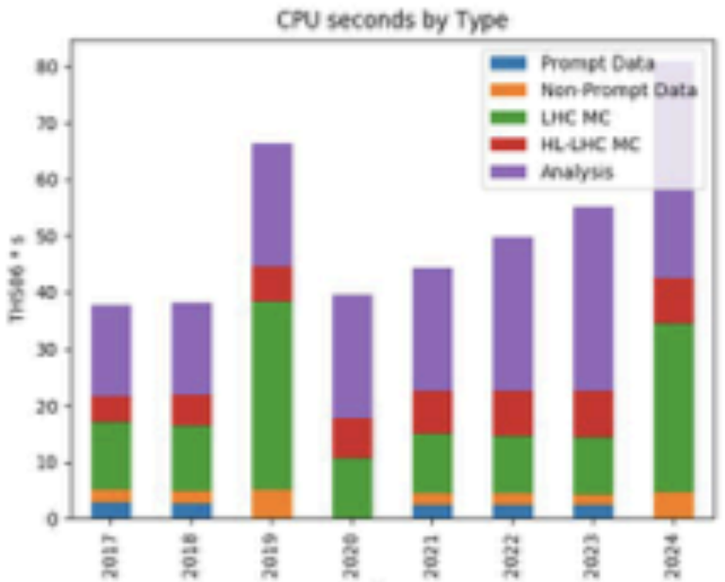
ATLAS

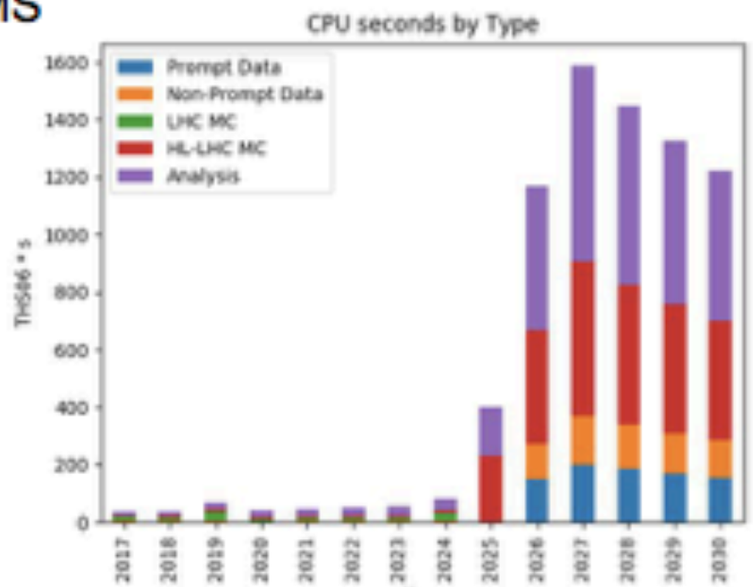


ATLAS Assumptions:

- Flat budgets
- WLCG cost model + US ATLAS procurement history
- Resource needs estimated from upgrade studies, (e.g. ATLAS ITK) and linear extrapolations

CMS



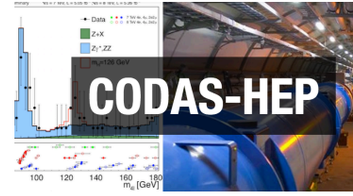


- Upper plots in HS06: constant capacity for 1 year
- Lower plots in HS06*s: CPUhours needed over 1 year
- 100 kHS06 for 1 year: 3.1 THS06*s

CMS Assumptions:

- Resource needs estimated from upgrade studies for TDRs and software optimization efforts
- Naive extrapolation of Run 2 computing model

7



Why explore new solutions?

- US [and the world] will soon be entering the Exascale Era
- National Strategic Computing Initiative ((NSCI)

➡ **White House sponsored Initiative**

- Goal is to have one machine ”on the floor” in 2021 and two additional machines by 2023

➡ **These will not look like our common few-core x86 CPUs**

- US conducted an Exascale requirements review in 2016 for each of the major sciences to gather requirements for this machine.
- HEP Report Available at https://arxiv.org/abs/1603.09303

- An Exascale computer provides a lot of compute. 1% of such a machine is more than is available to HEP today worldwide

Parallelizing seeding: CA

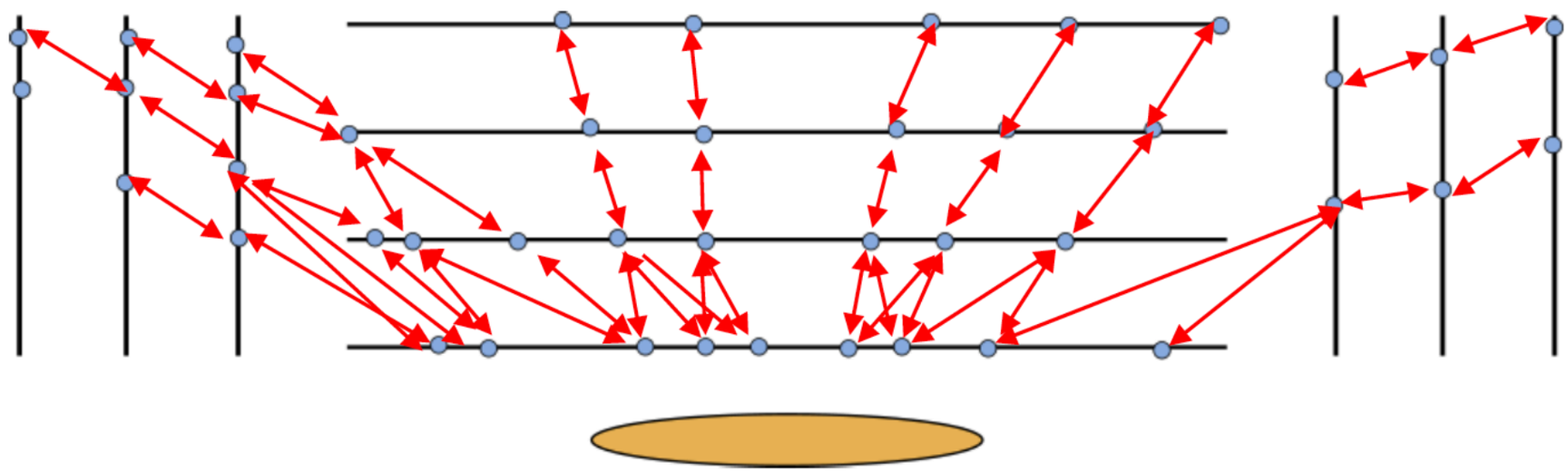
- The Cellular Automata (CA) seeding is a track seeding algorithm designed for parallel architectures
- It requires a list of layers and their pairings

➡ **A graph of all the possible connections between layers is created**

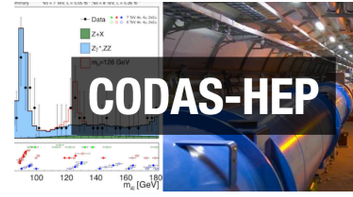
➡ **Doublets aka Cells are created for each pair of layers (compatible with a region hypothesis)**

➡ **Fast computation of the compatibility between two connected cells**

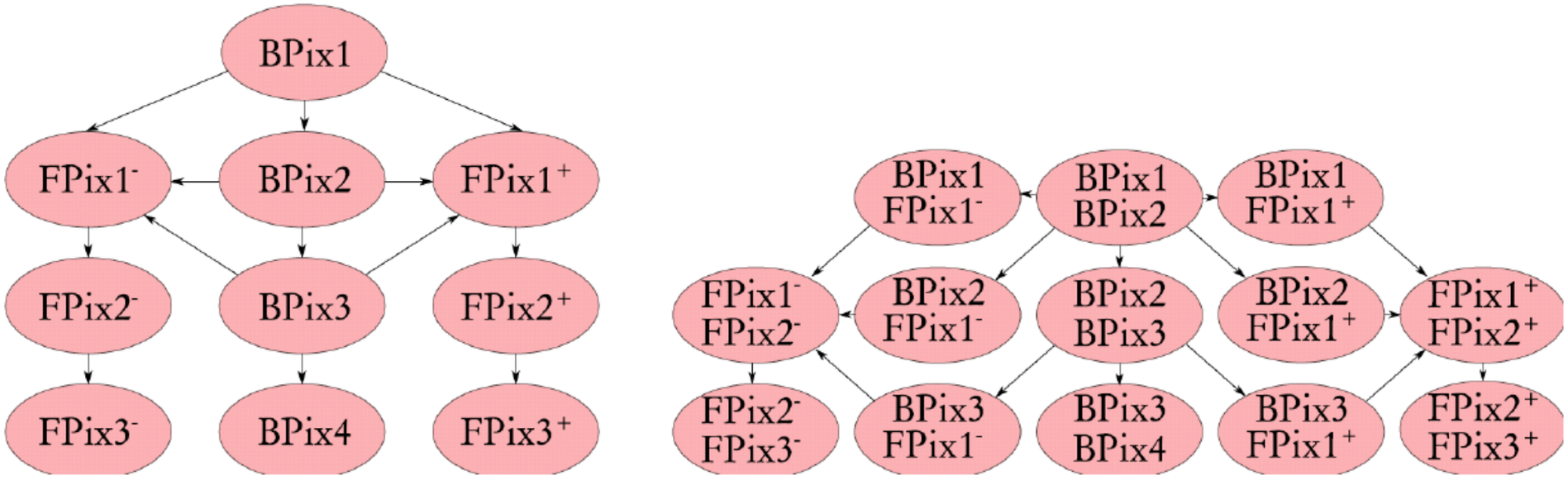
➡ **No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize**



CA seeding layers



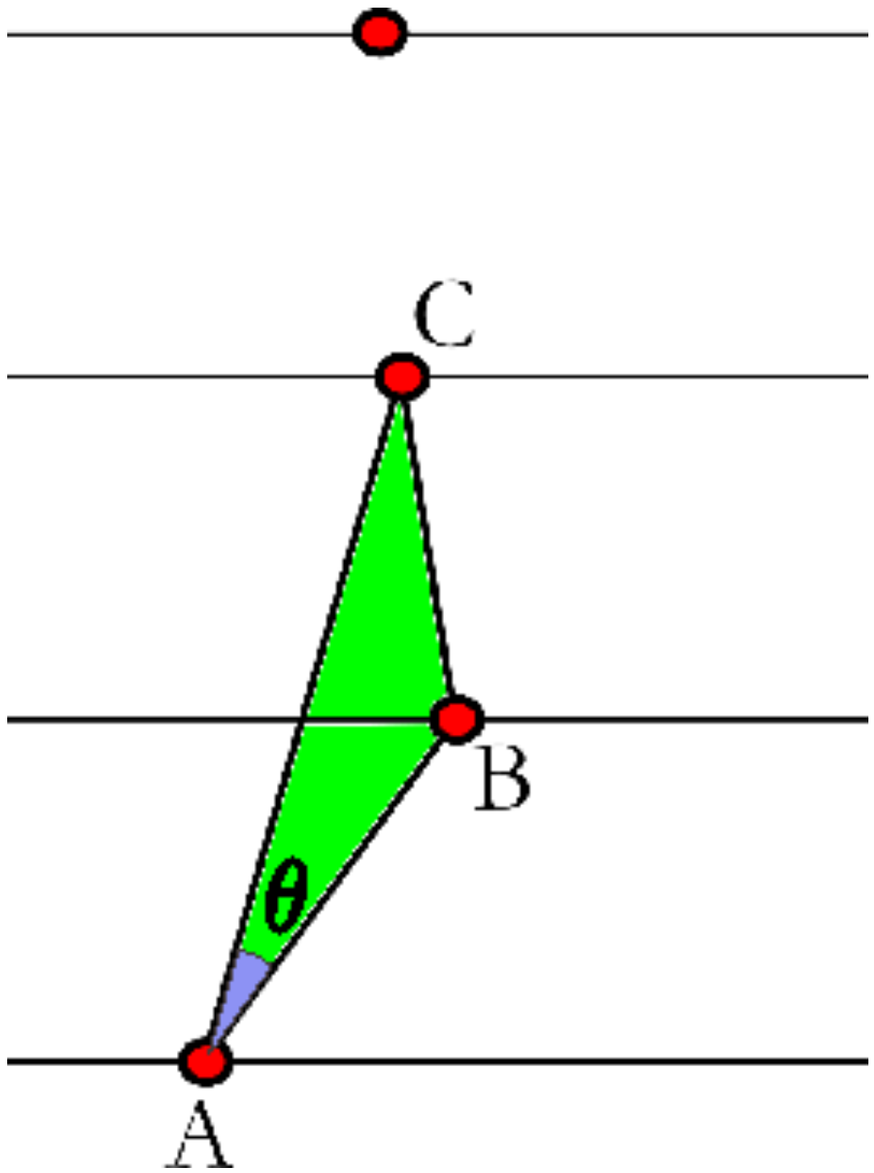
- Seeding layers interconnections are not random. Patterns/pairings of layers follow normally produced tracks.
- Hit doublets for each layer pair can be computed independently by sets of threads

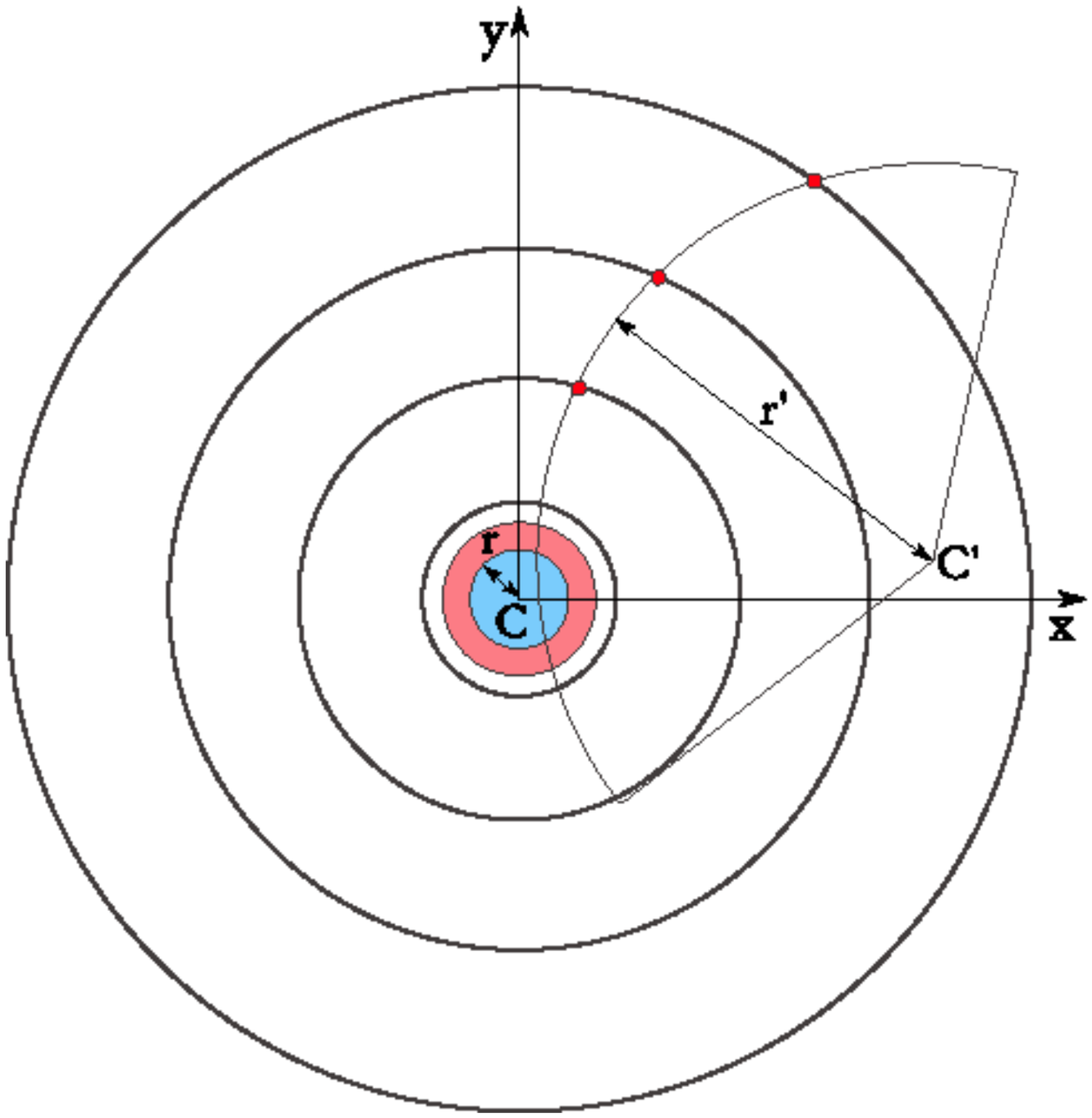


- This graph also identifies which pair combinations are worth checking when building triplets

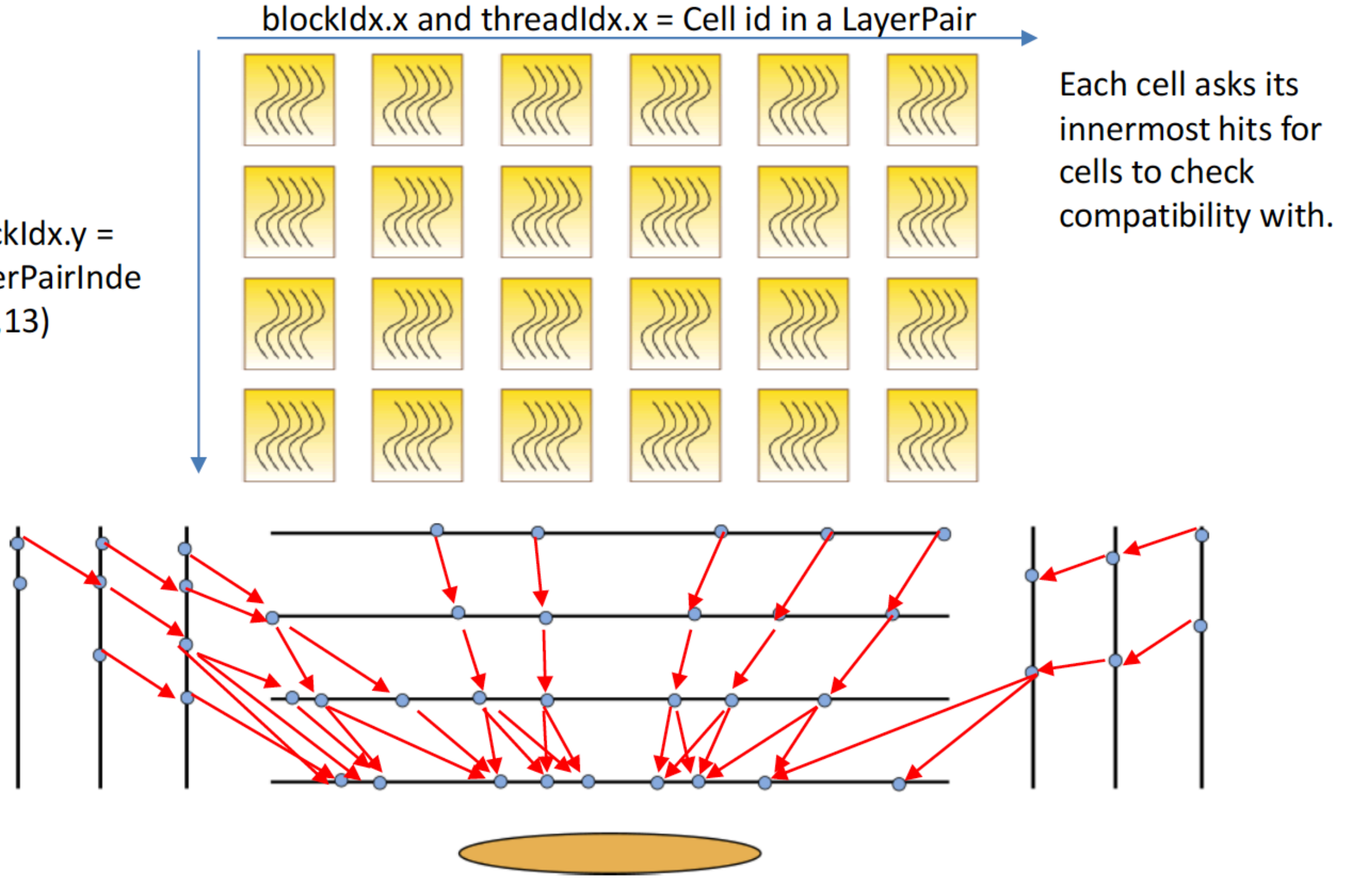
CA cell combination

- Compatibility is checked in rz and in xy



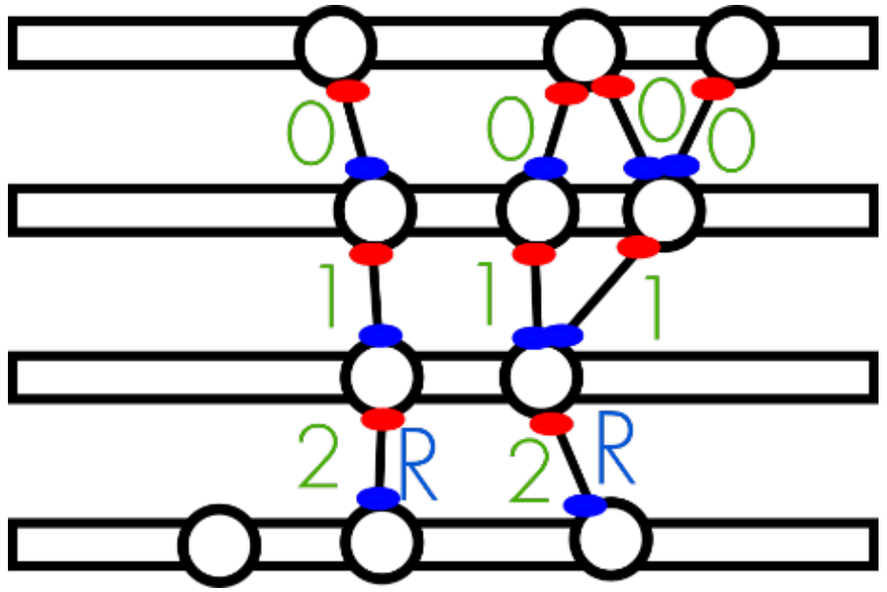


CA cell combination

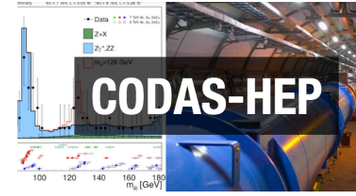


CA: evolution

- If two cells satisfy all the compatibility requirements they are said to be neighbors and their state is set to 0
- In the evolution stage, their state increases in discrete generations if there is an outer neighbor with the same state



- At the end of the evolution stage the state of the cells will contain the information about the length
- If one is interested in quadruplets, there will be surely one starting from a state 2 cell, pentuplets state 3, etc.

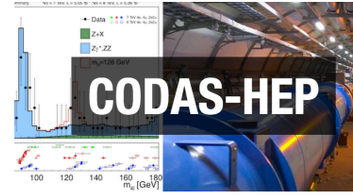


Parallelizing Kalman Filter Tracking

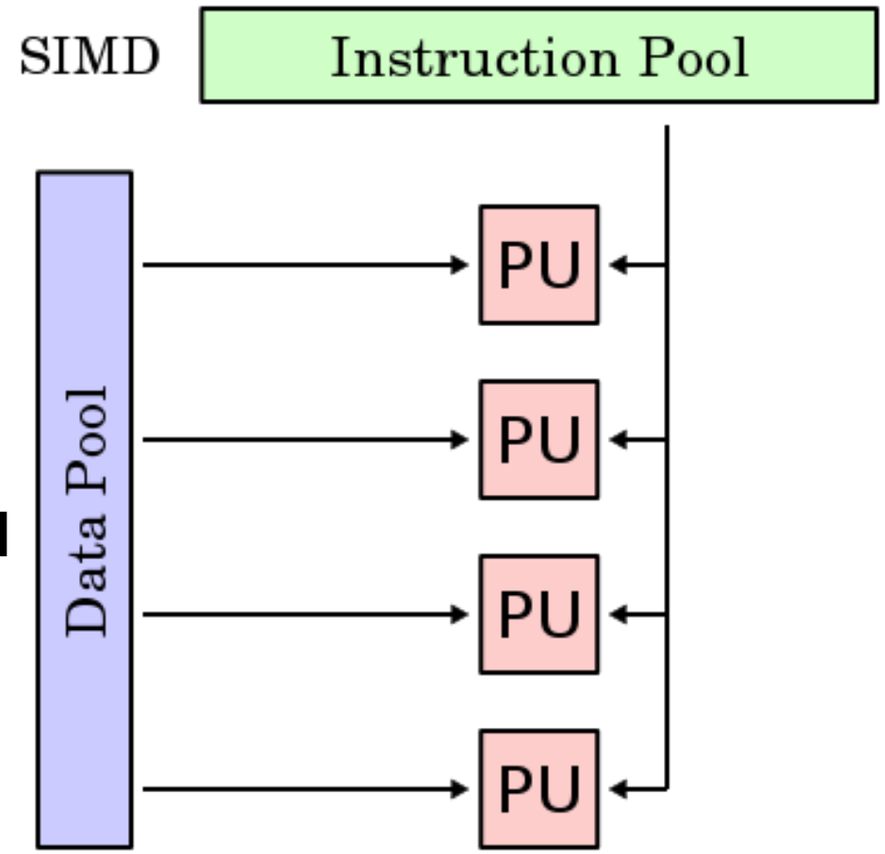
G. Cerati[4], P. Elmer[3], M. Kortelainen[4], S. Krutelyov[1], S. Lantz[2], M. Lefebvre[3], M. Masciovecchio[1], K. McDermott[2], B. Norris[5], D. Riley[2], M. Tadel[1], P. Wittich[2], F. Würthwein[1], A. Yagil[1]

1. UCSD 2. Cornell
3. Princeton 4. FNAL 5. U of Oregon

Challenges to Parallel KF Tracking



- KF tracking cannot be ported in straightforward way to run in parallel
- Need to exploit two types of parallelism with parallel architectures

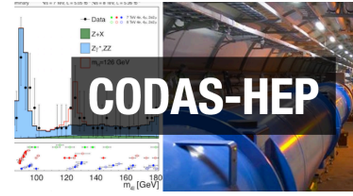


- Vectorization
  - **Perform the same operation at the same time in lock-step across different data**

  ➡ **Challenge: branching in track building – exploration of multiple track candidates per seed**
- Parallelization
  - **Perform different tasks at the same time on different pieces of data**

  ➡ **Challenge: thread balancing – splitting the workload evenly is difficult as track occupancy in the detector not uniform on a per event basis**

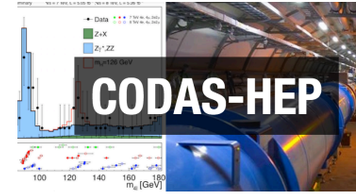
Selected parallel architectures



	Xeon E5-2620 Sandy Bridge (SNB)	Xeon Phi 7120P Knights Corner (KNC)	Xeon Phi 7230 Knights Landing (KNL)	Tesla K40
Logical Cores	6x2x2	61x4	64x4	2880 CUDA cores
Clock rate	2.5 GHz	1.24 GHz	1.3 GHz	875 MHz
GFLOPS	120	1208	2660	1430
SIMD width	256 bits	512 bits	2x512 bits	32 thread warp
Memory	~64-384 GB	16 GB	16 & 384 GB	12 GB
Bandwidth	42.6 GB/s	352 GB/s	475 & 90 GB/s	288 GB/s

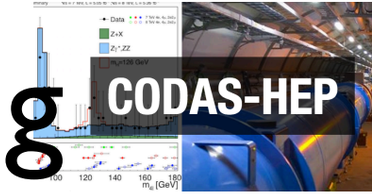
- We recently added Skylake Gold [6130]:
  - **Logical cores: 16x2x2**
  - **Clock rate*: 2.1 GHz**
  - **SIMD width: 512**

Parallelization and Vectorization



- Threading (task parallelism)
  - ✓ **Choices: OpenMP, Cilk Plus, TBB, Pthreads, CUDA kernels, etc.**
  - ✓ **It's all about sharing work and scheduling**
- Vectorization (data parallelism)
  - ✓ **"Lock step" Instruction Level Parallelization (SIMD)**
  - ✓ **Requires management of synchronized instruction execution**
  - ✓ **It's all about finding simultaneous operations**
- To utilize advanced architectures fully, both types of parallelism need to be identified and exploited
  - ✓ **Need 2–4+ threads to keep a core busy (in-order execution stalls)**
  - ✓ **Vectorized loops gain 8x or 16x performance on AVX or AVX512**

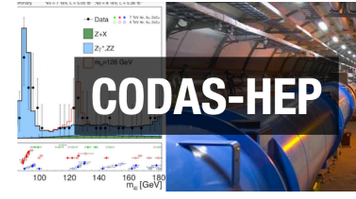
Strategy for track building and fitting



- Vectorization via Matriplex library
  - ✓ **all Kalman operations (matrix operations) involve this library to use vector registers**

- Parallelization using TBB
  - ✓ **different threads handle groups of seeds (building) or groups of tracks (fitting)**

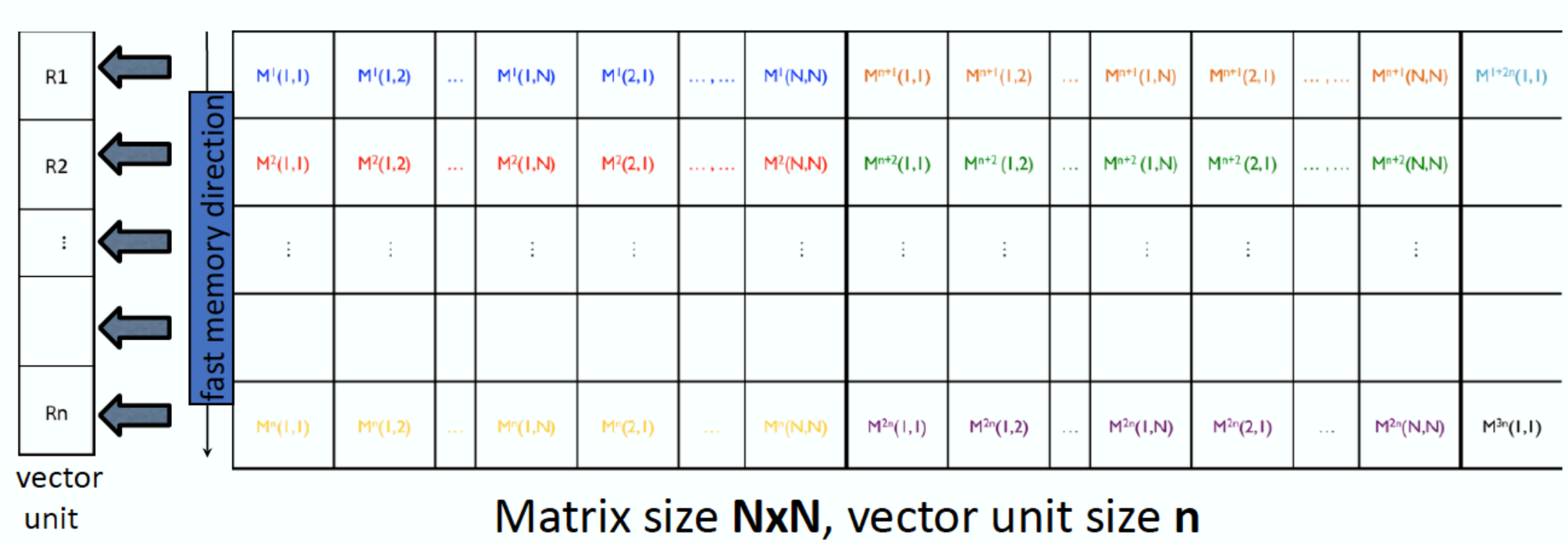
Custom tool: Matriplex



- Matrix operations of KF ideal for vectorized processing: however, requires synchronization of operations
- Most matrix libraries are for large matrices (ours are small)

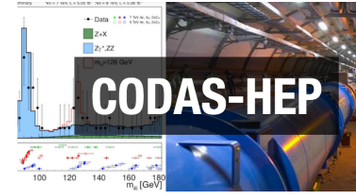
- Arrange data in such a way that it can loaded into the vector units of Xeon and Xeon Phi with Matriplex

  ✓ **Fill vector units with the same matrix element from different matrices: n matrices working in sync on same operation**

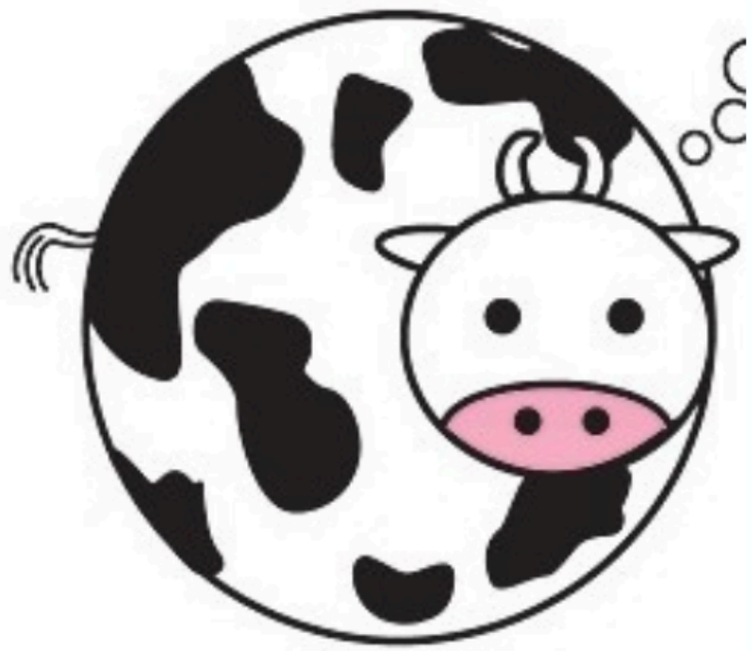


- See talk tomorrow by Steve Lantz

Initial Experimental Setup

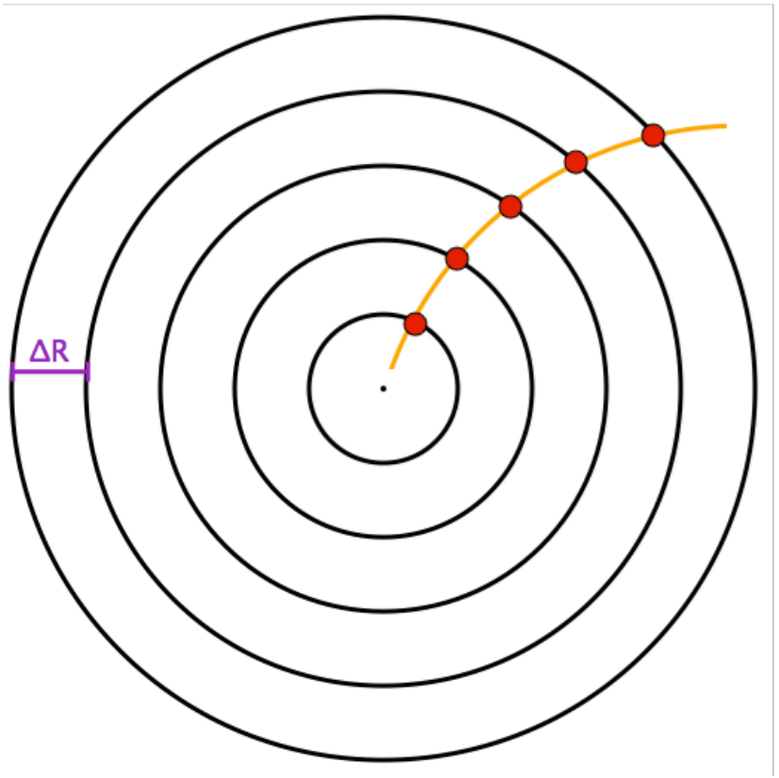


- Simple starting point:
  - **“Cylindrical Cow”:10 barrel layers, △R = 4cm, |η|<1, 3.8T magnetic field**
  - **Beam spot 1mm in xy, 1cm in z**
  - **Hit resolution 100μm in r-phi, 1mm in z**
  - **Uncorrelated tracks, no scattering**

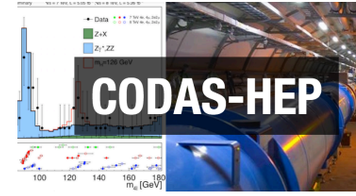


https://sites.google.com/site/lauranstoner/

- Simplest case — we’d better understand this
- Expect performance under these circumstances to be upper limit on how well you can do
- Move to realistic detector after this has been understood



How well does it perform?

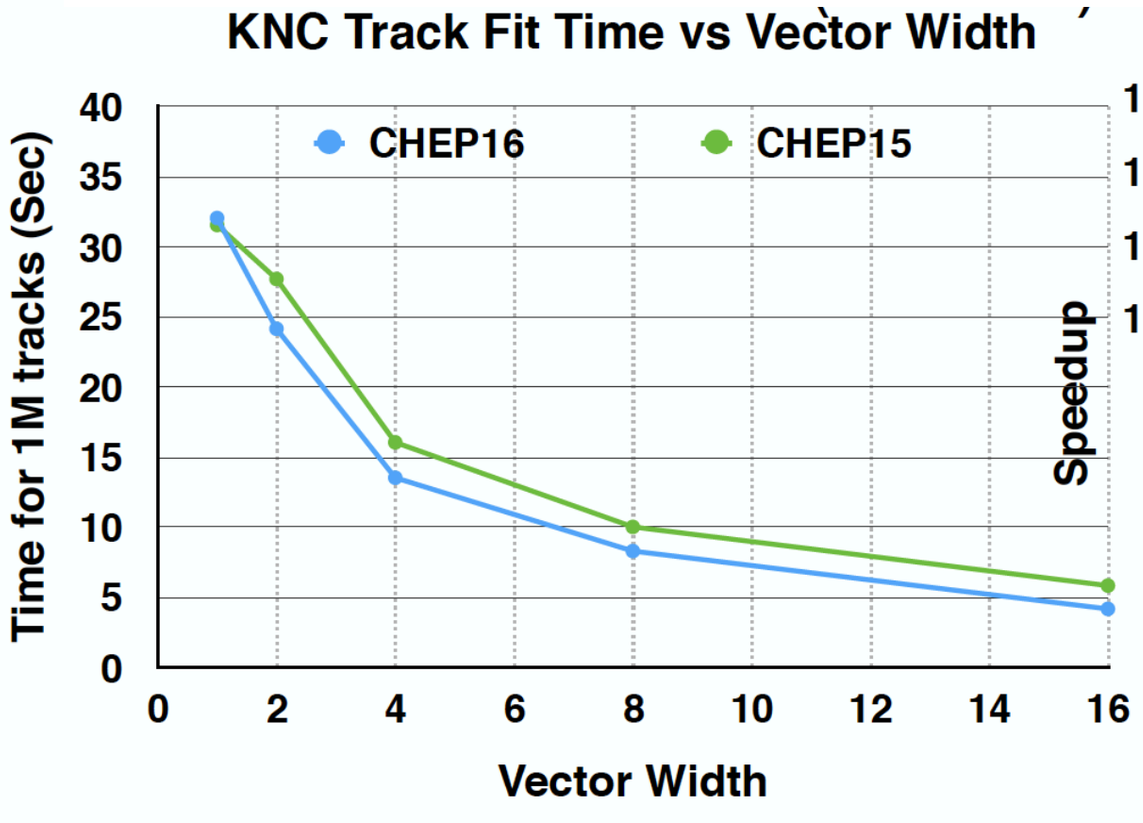


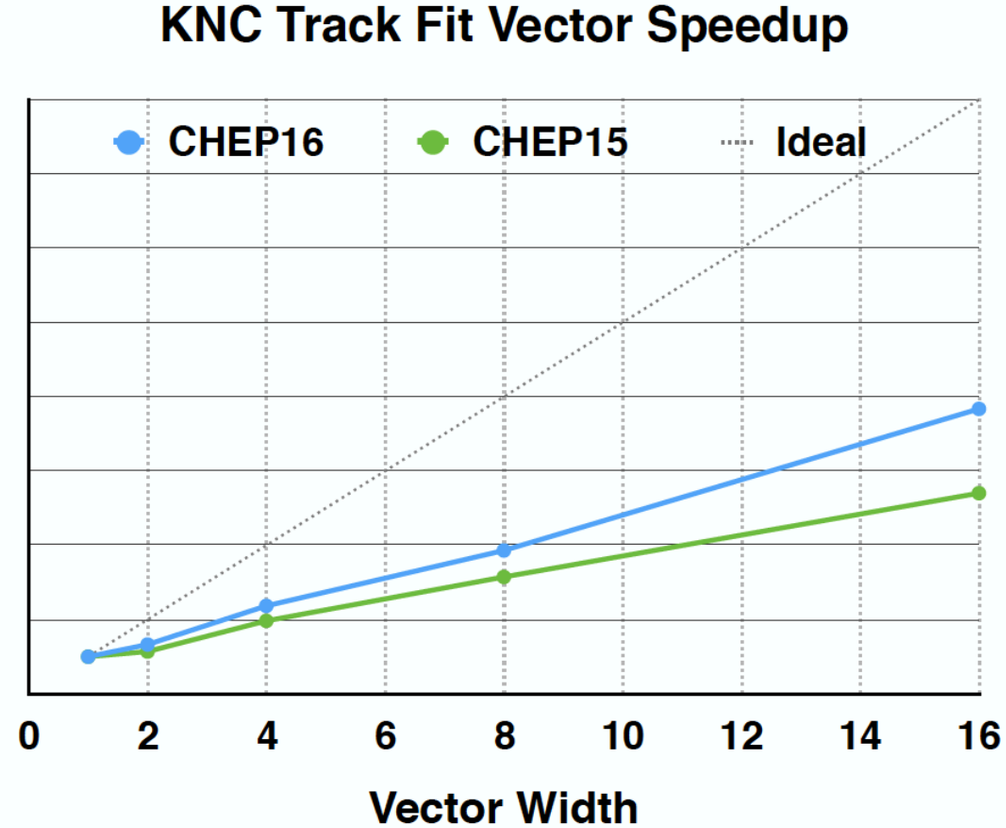
- Naively might expect speed-ups of 200+ on Xeon Phi (cf scalar single threaded code). What actually happens?
  - **(remember — toy detector)**
- Test Track Building. Simplest case:

➡ **KF calculation is just a repetition of propagate & update steps**

➡ **No branching, all tracks do the same thing, only 1 path to follow**

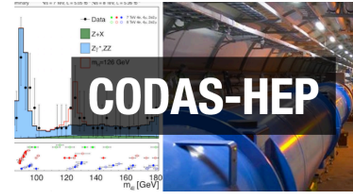
➡ **Vectorization results (16 max):**



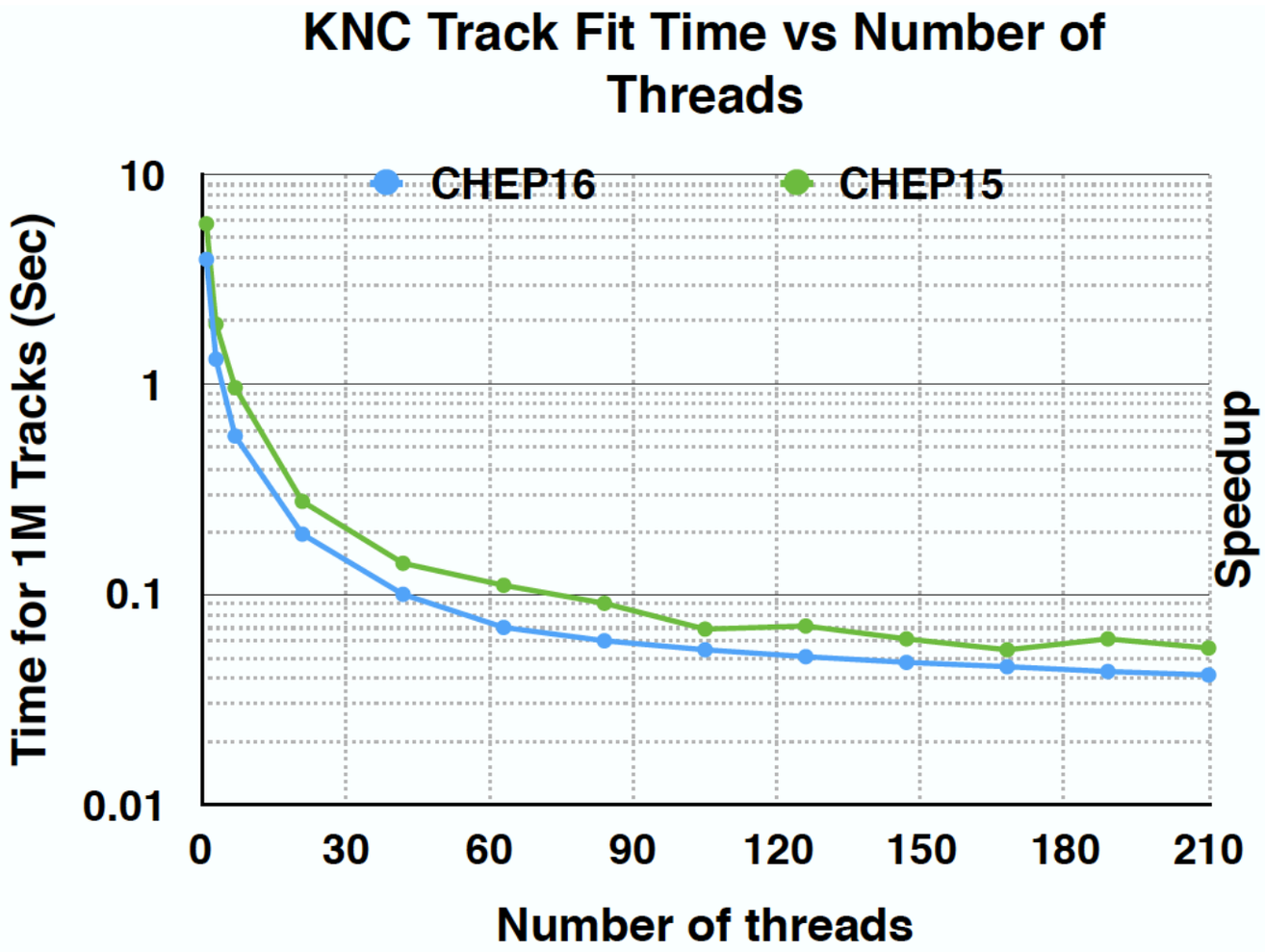


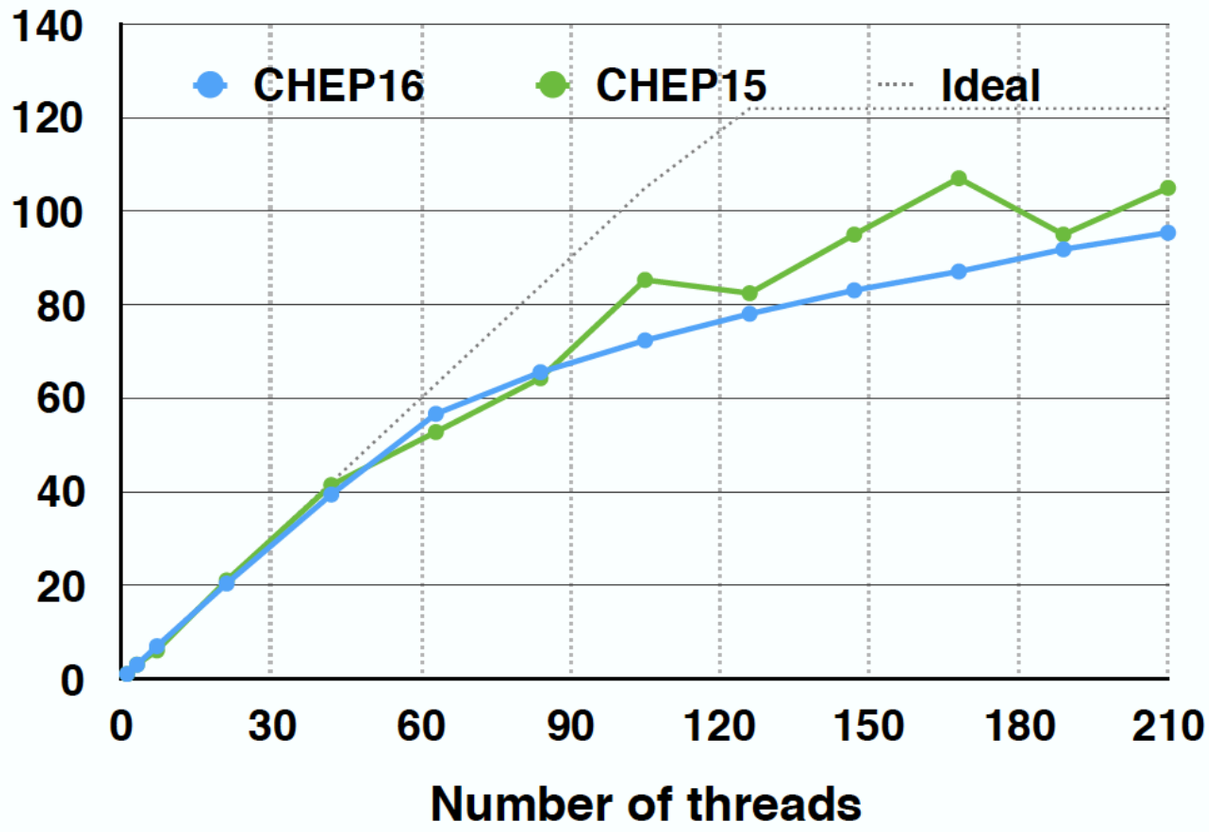
- Speed-up by factor ≲ 8 — impressive, but only 1/2 of theoretical maximum

How well does it perform?



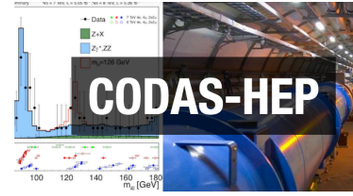
- What about parallelization?



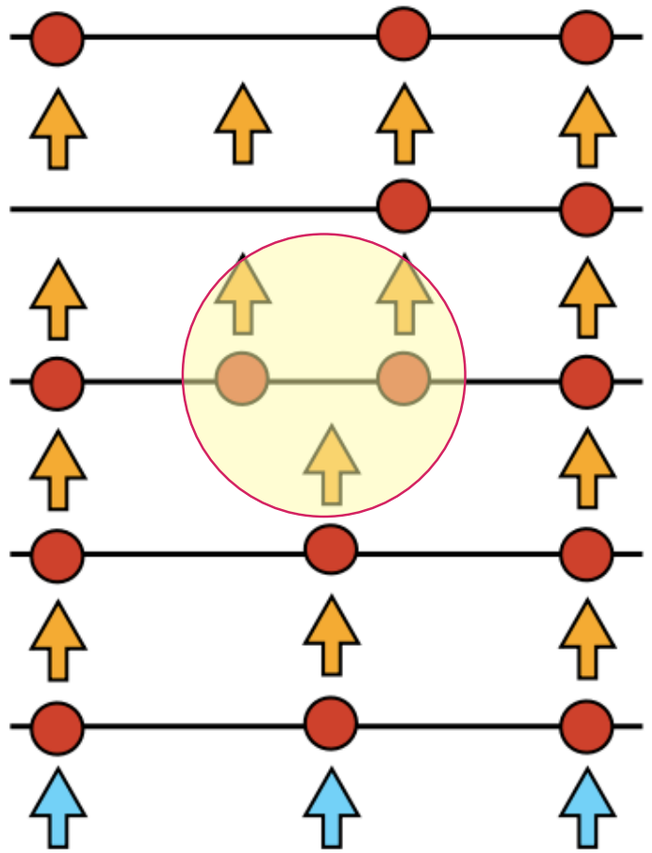


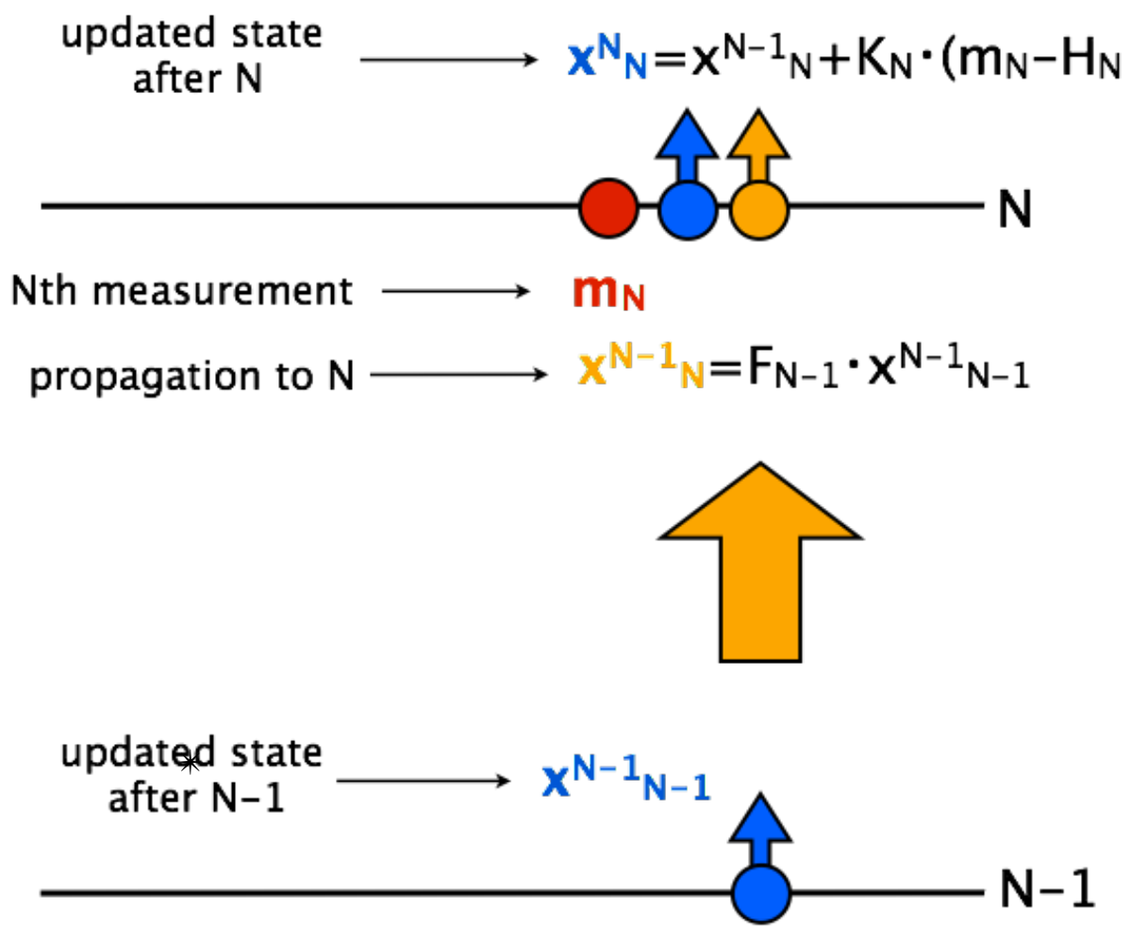
- Parallelization near ideal up to 61 threads (number of physical cores)
- Reach ~100x speedup at ~200 threads
- Ideally ≥122x to occupy available instruction slots
- CHEP2016 faster due to better vectorization

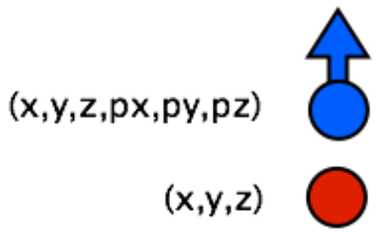
Track building



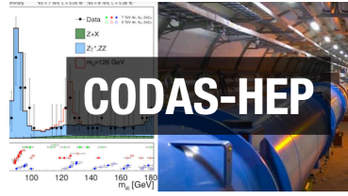
- Remember this is harder
- branches, variable execution, etc etc etc …
- expect performance to degrade

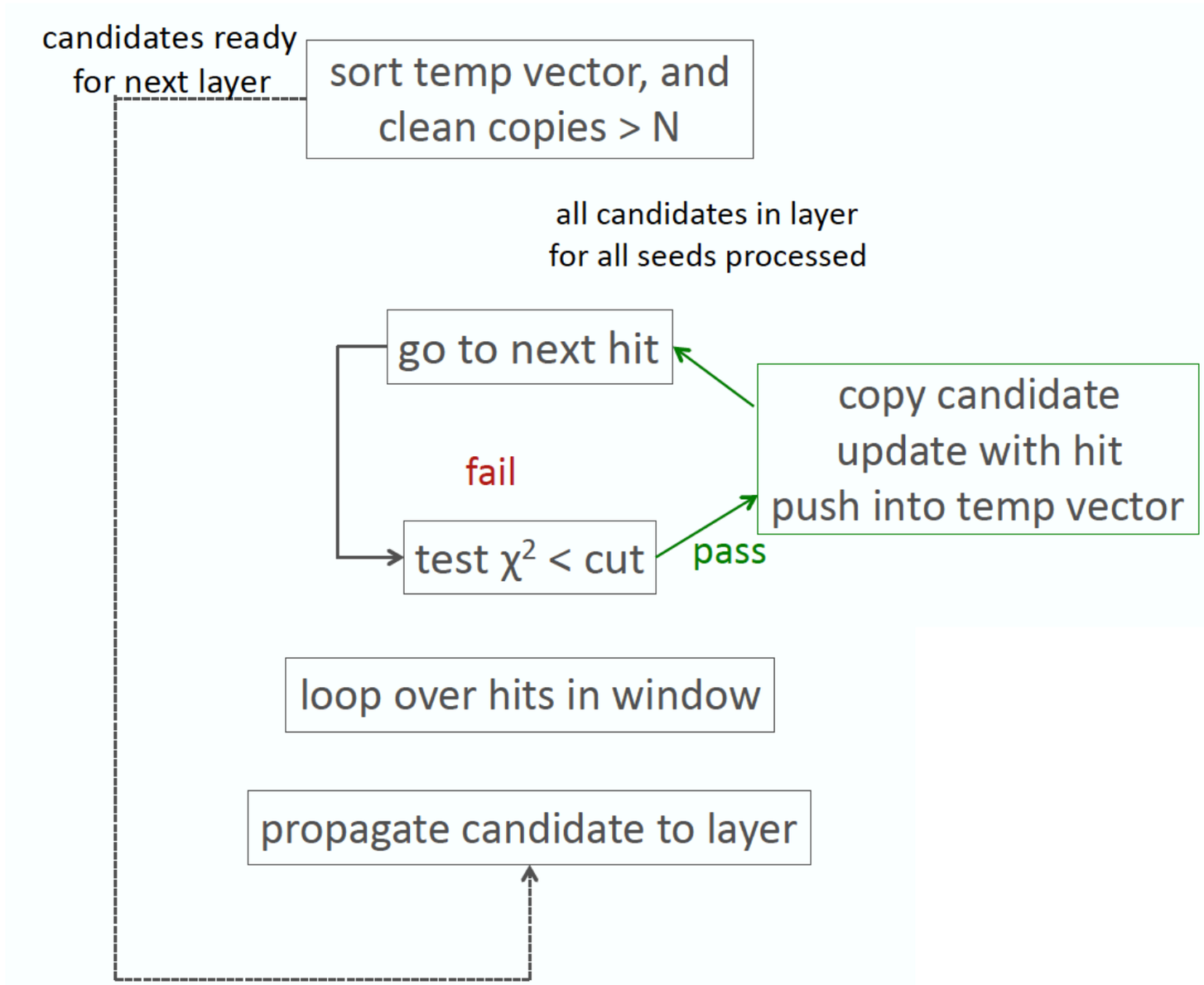




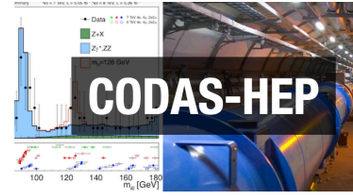


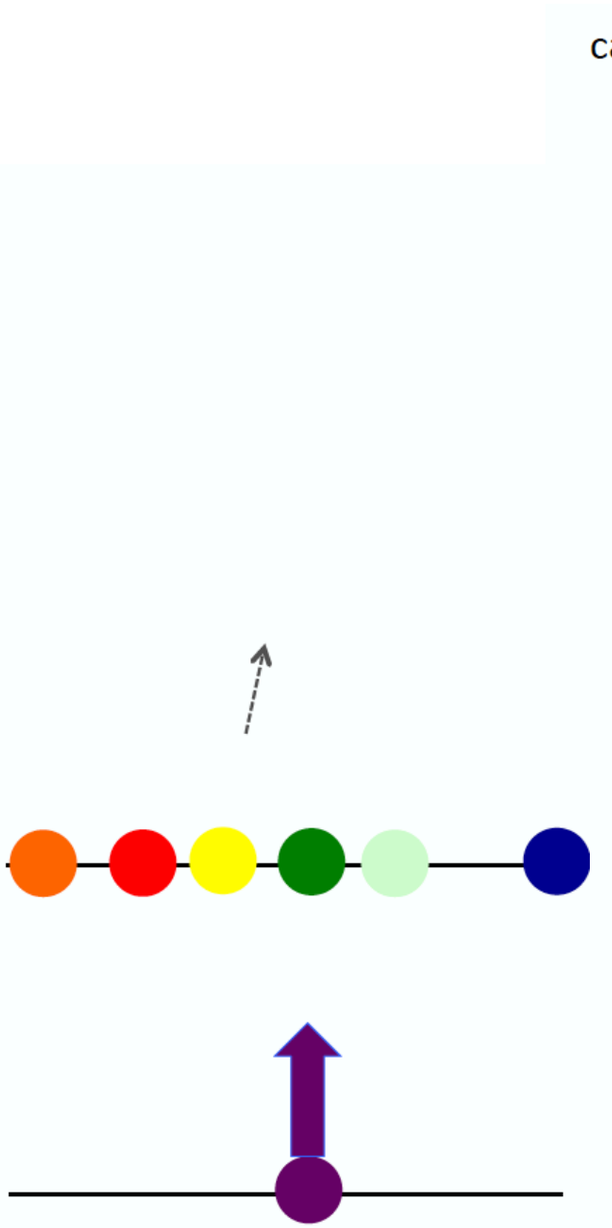
Handling Multiple Candidates (1)

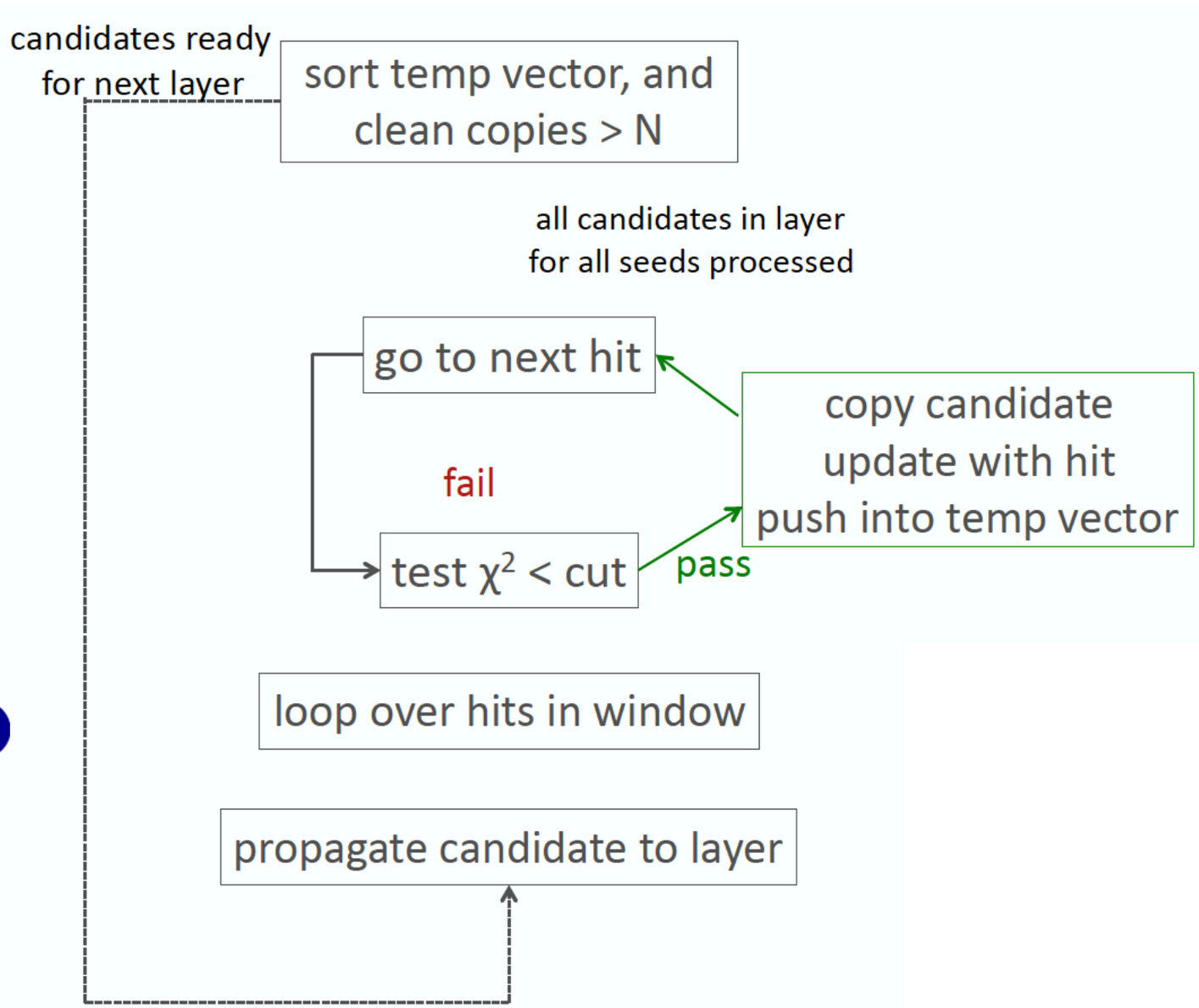




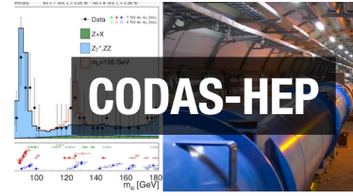
Handling Multiple Candidates (1)

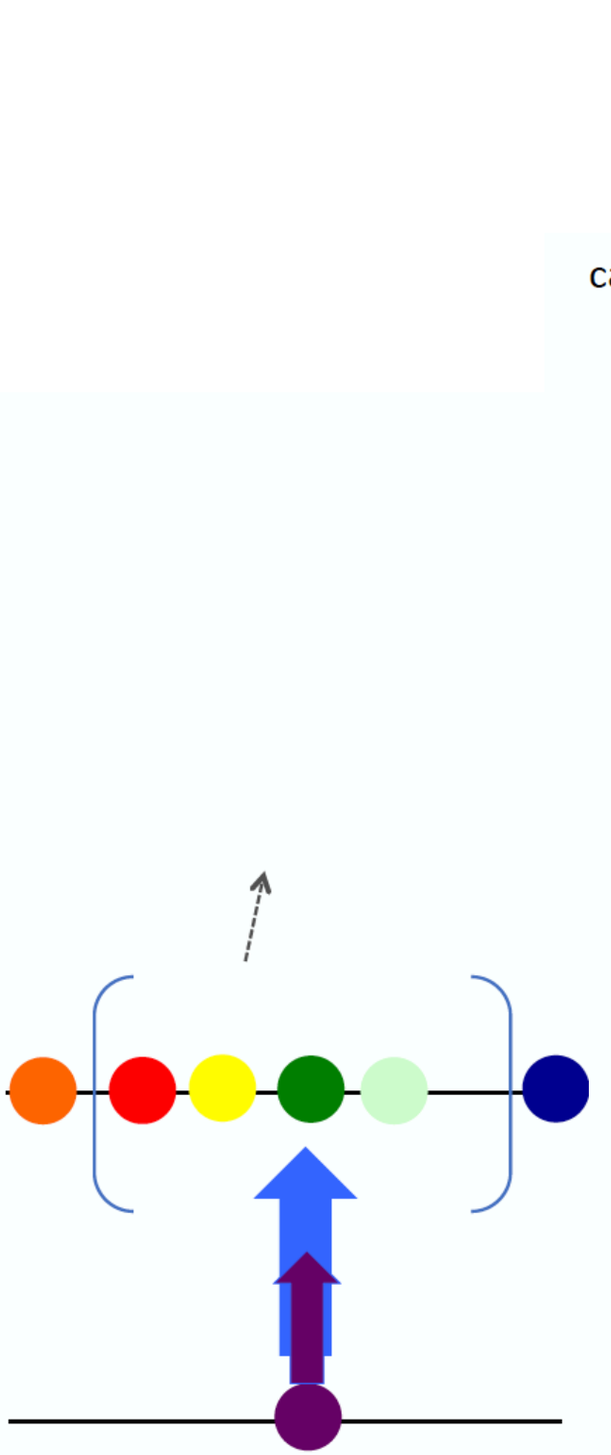


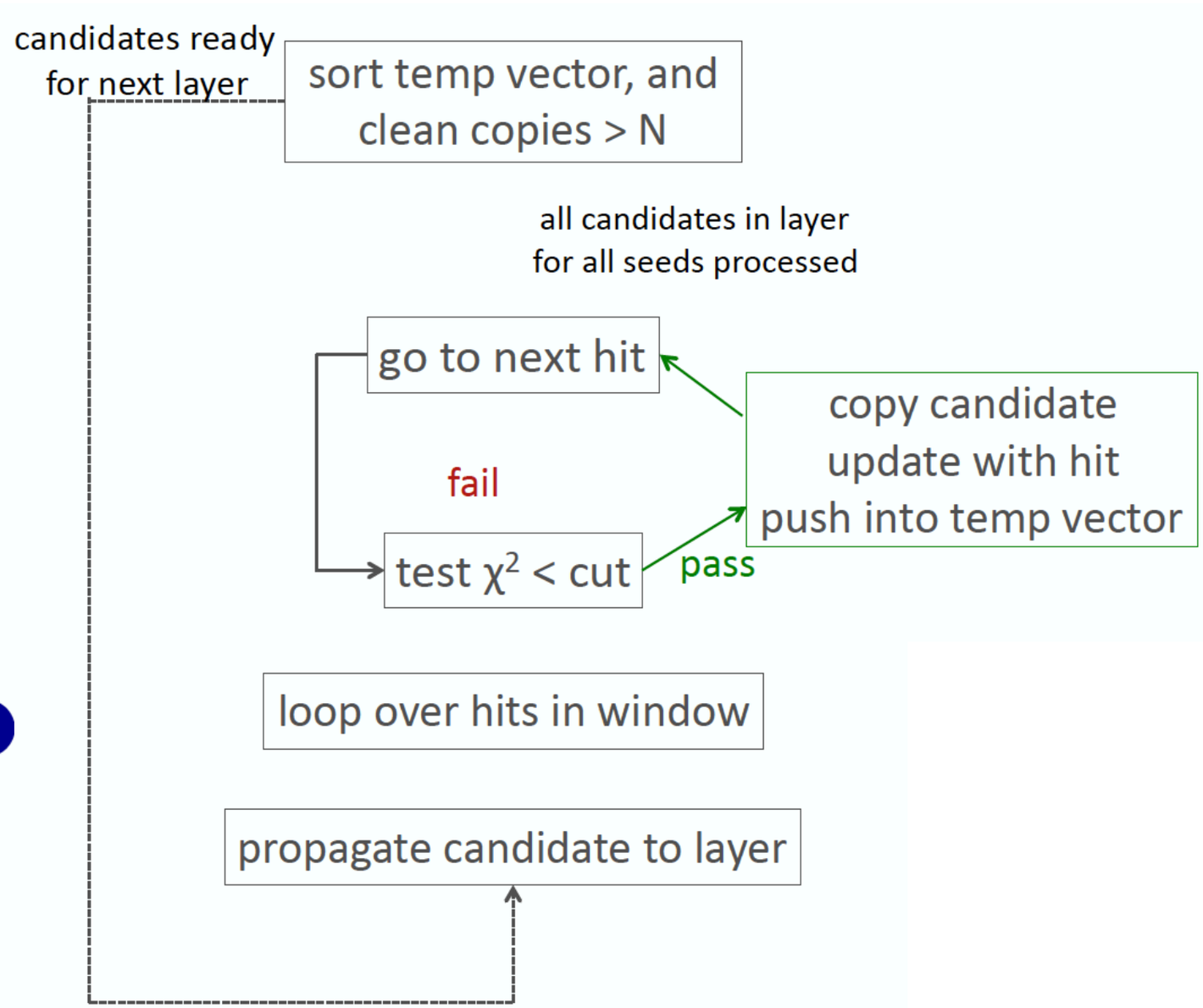




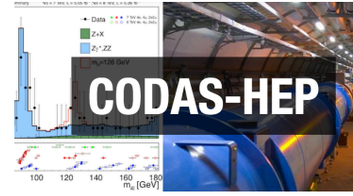
Handling Multiple Candidates (1)

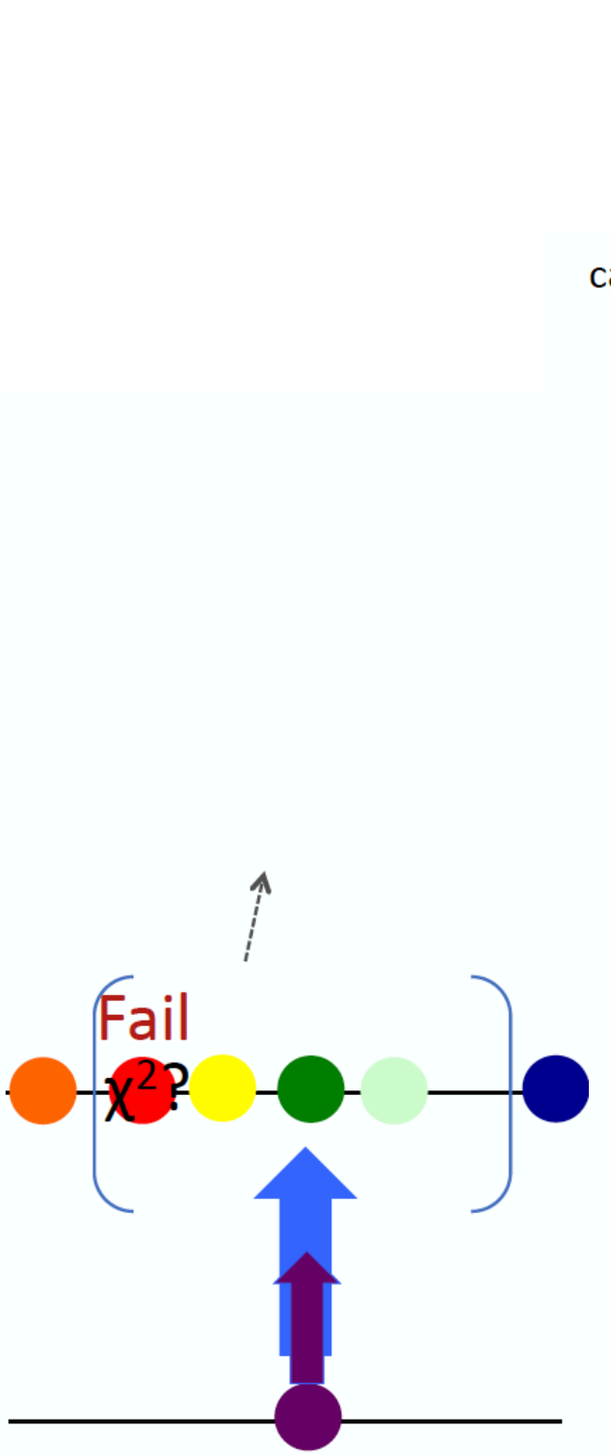


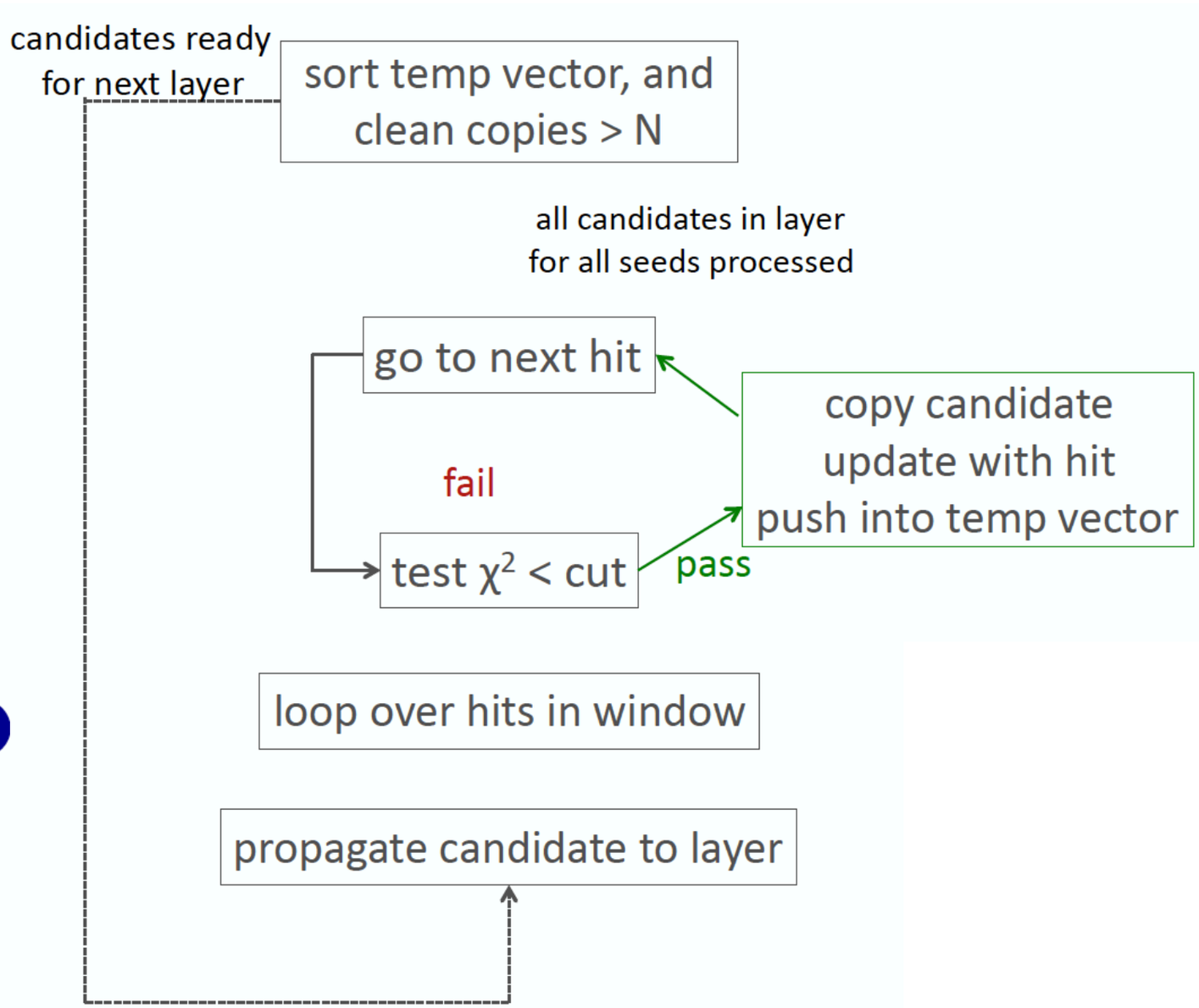




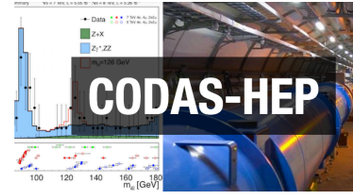
Handling Multiple Candidates (1)

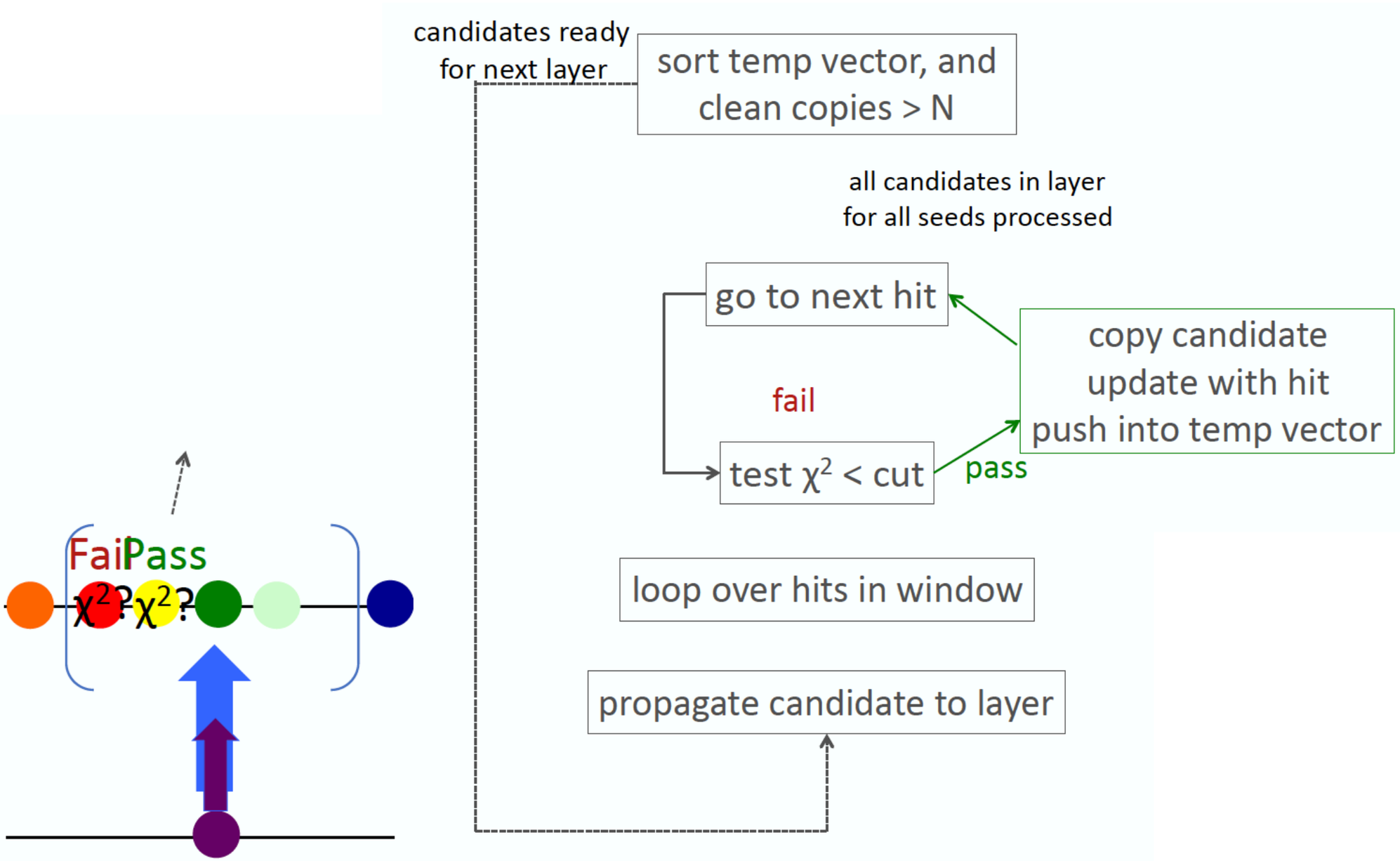




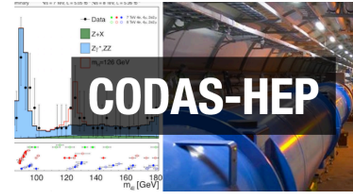


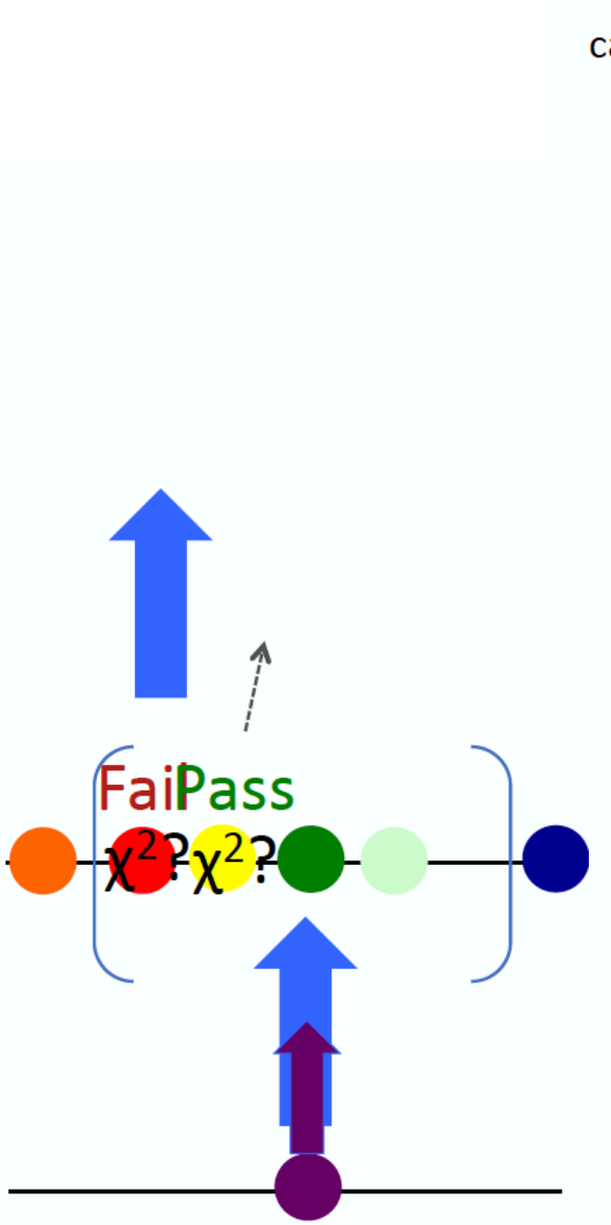
Handling Multiple Candidates (1)

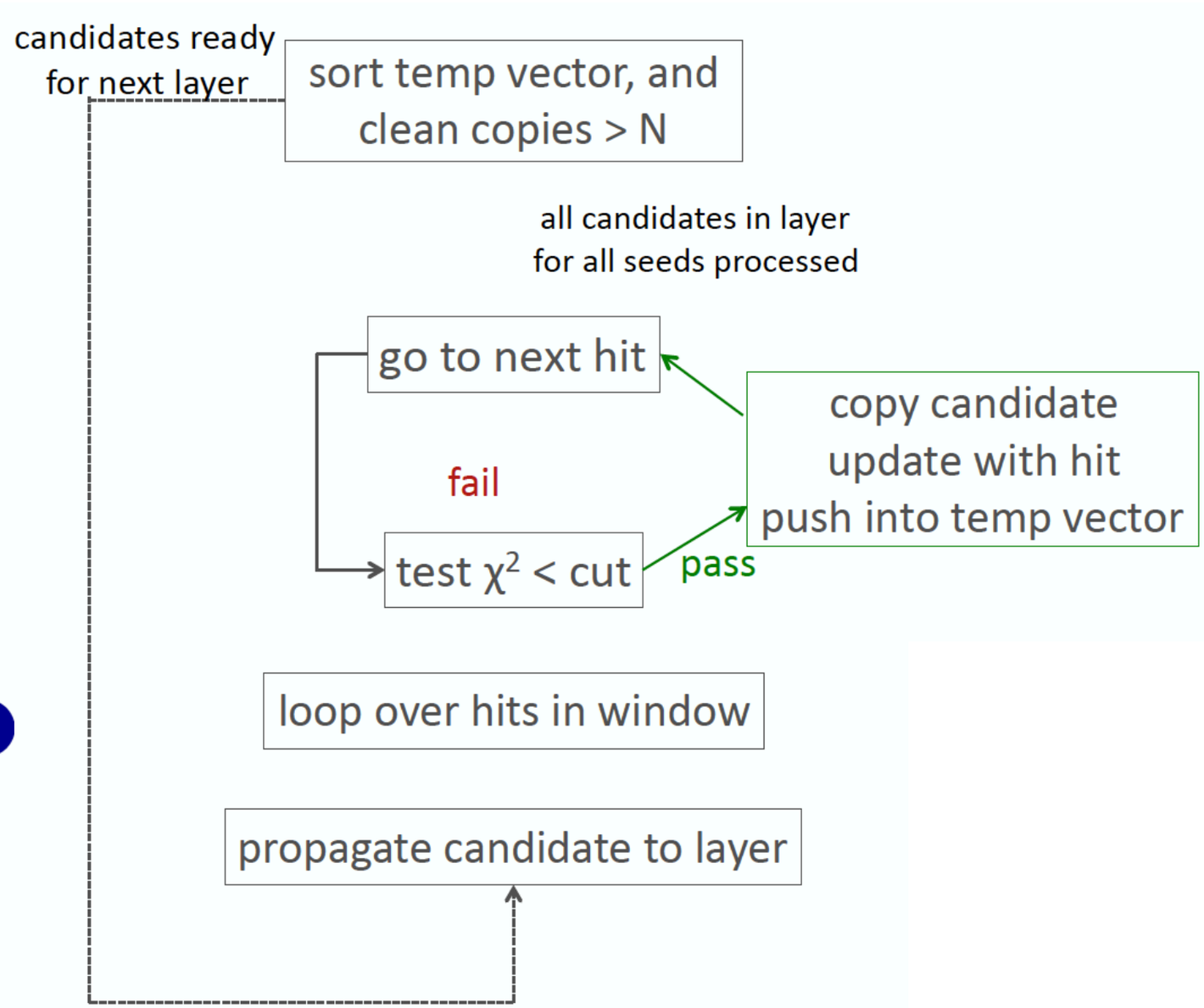




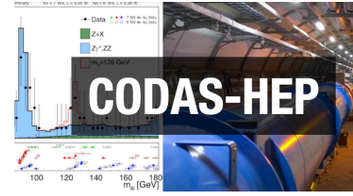
Handling Multiple Candidates (1)

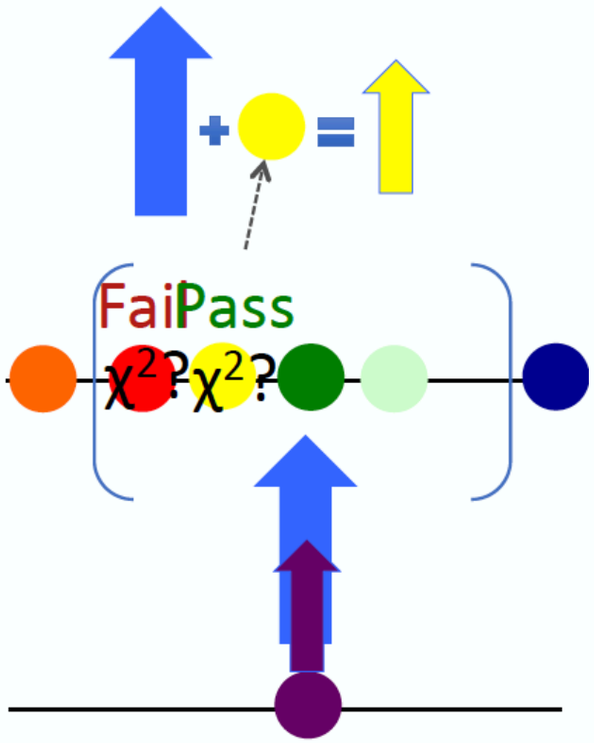


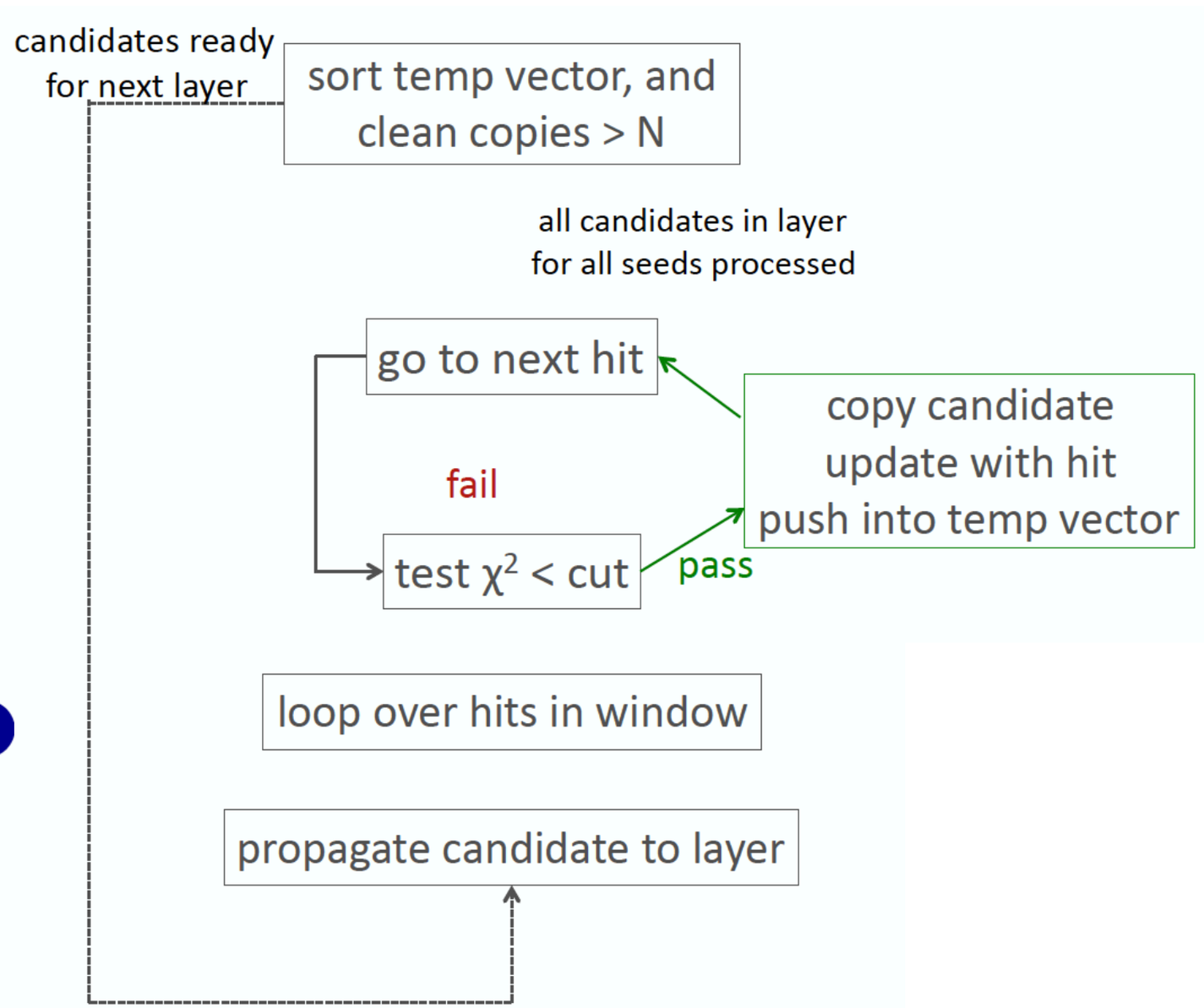




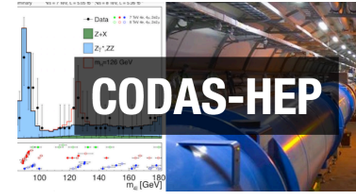
Handling Multiple Candidates (1)

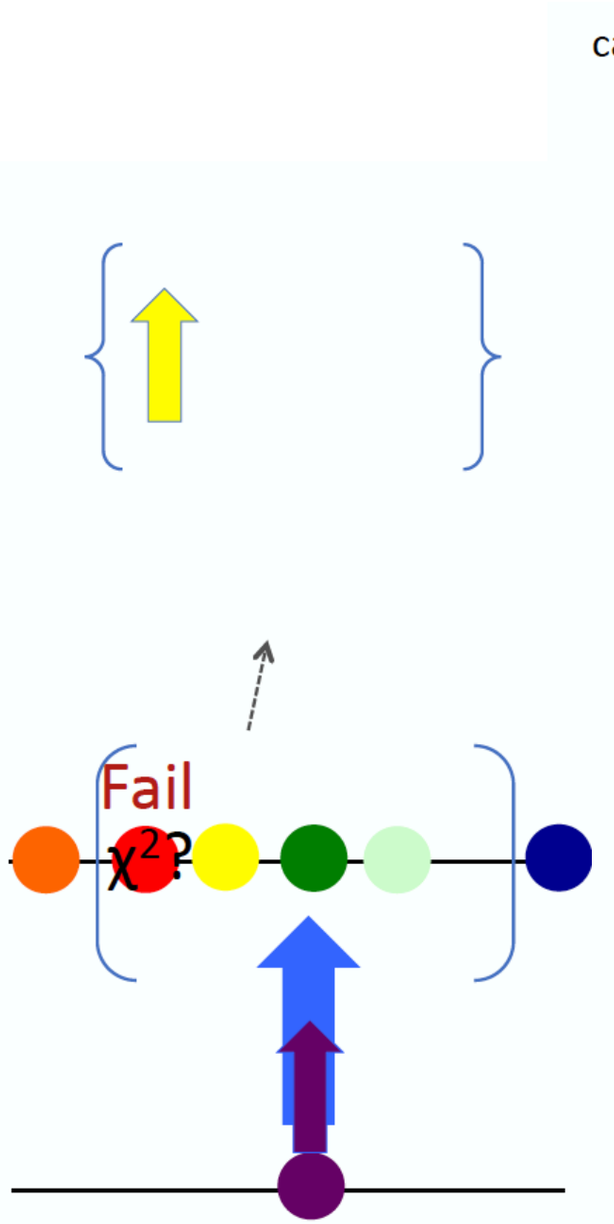


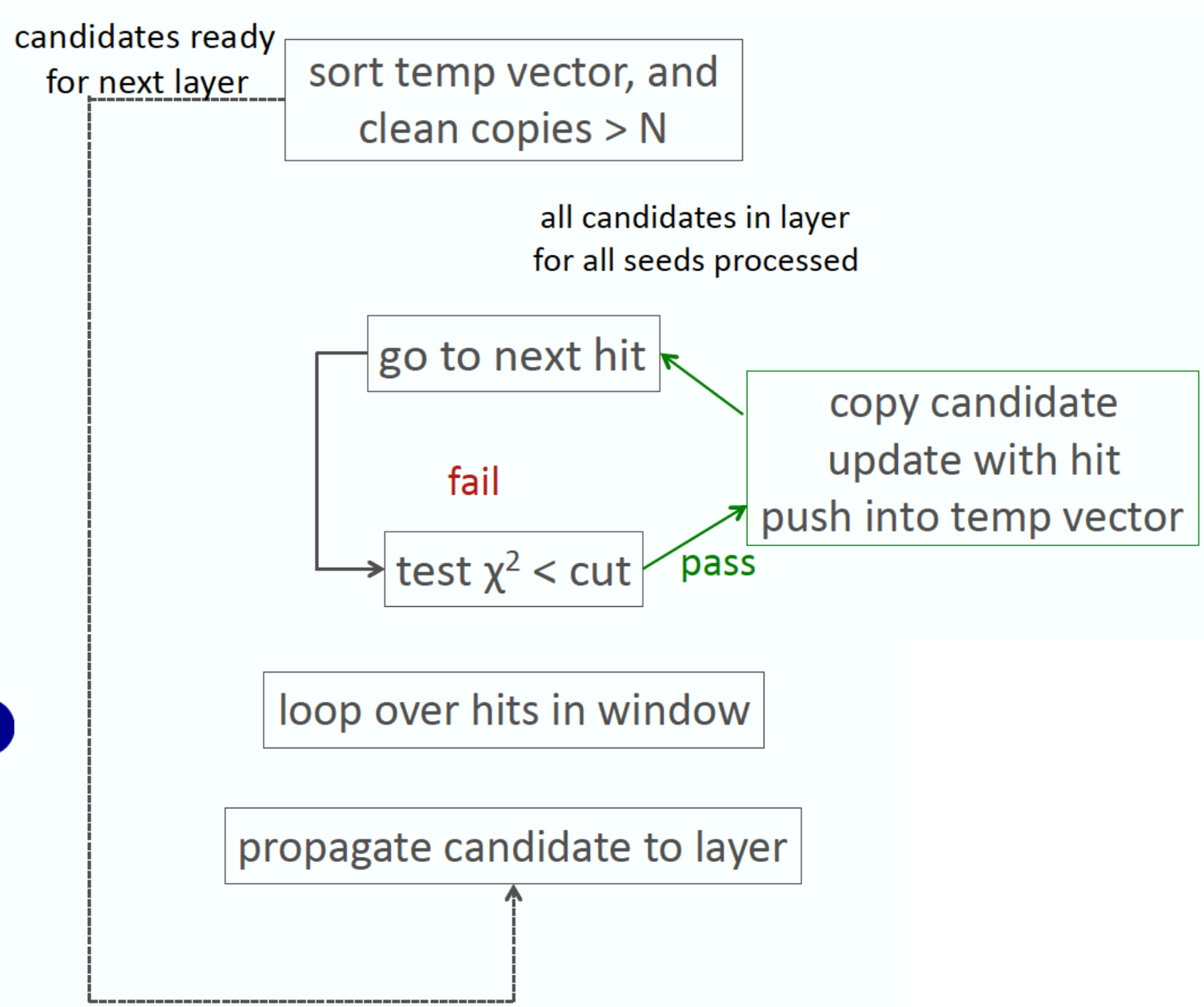




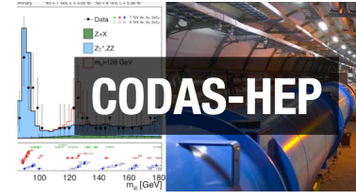
Handling Multiple Candidates (1)

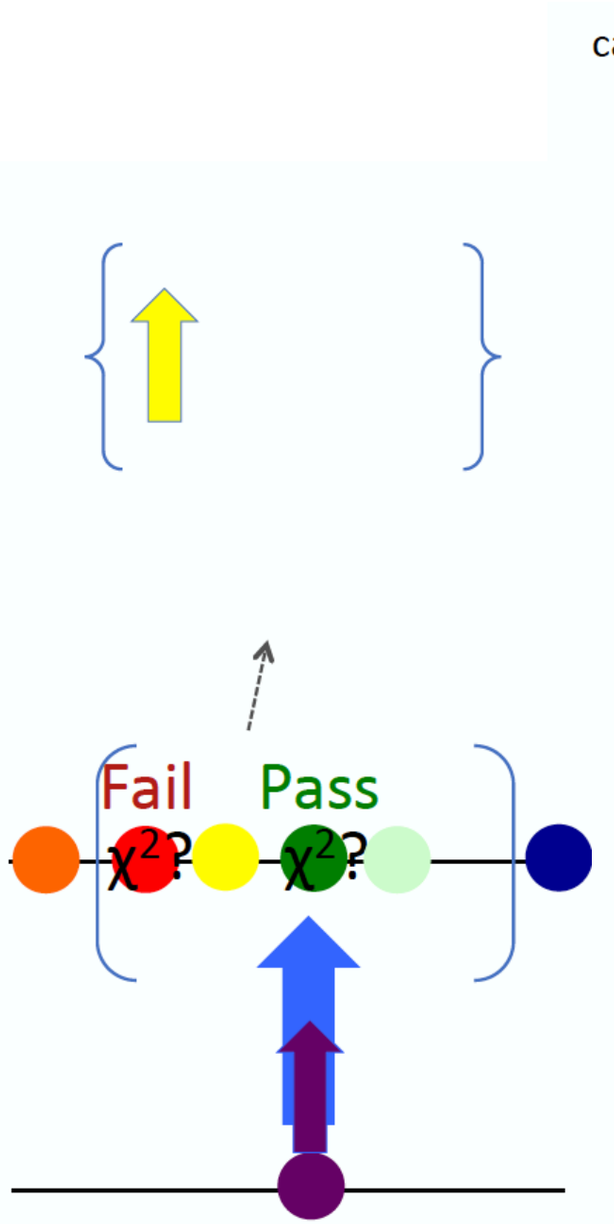


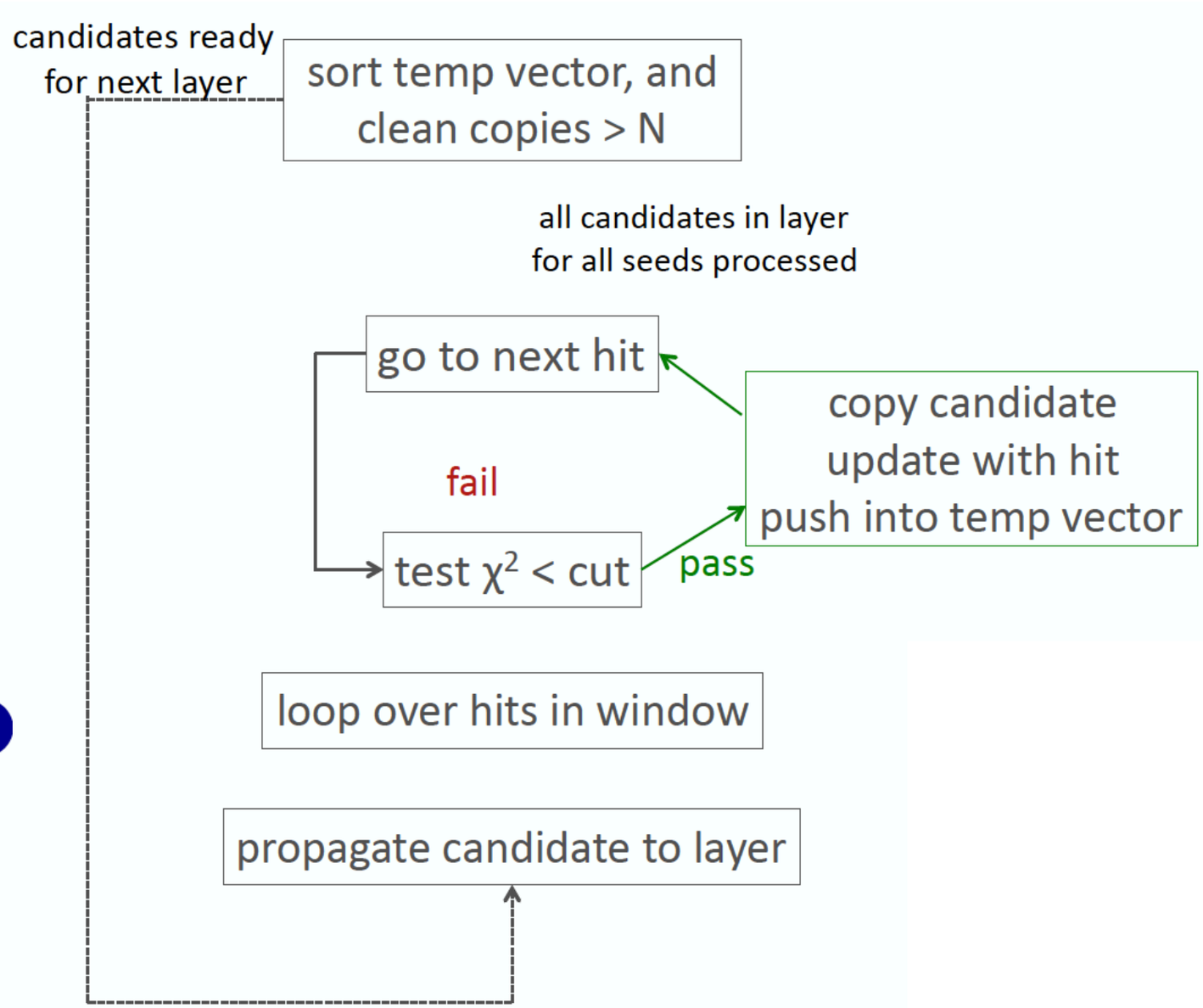




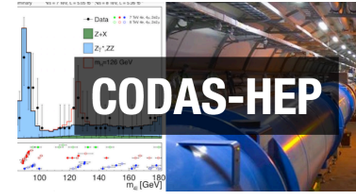
Handling Multiple Candidates (1)

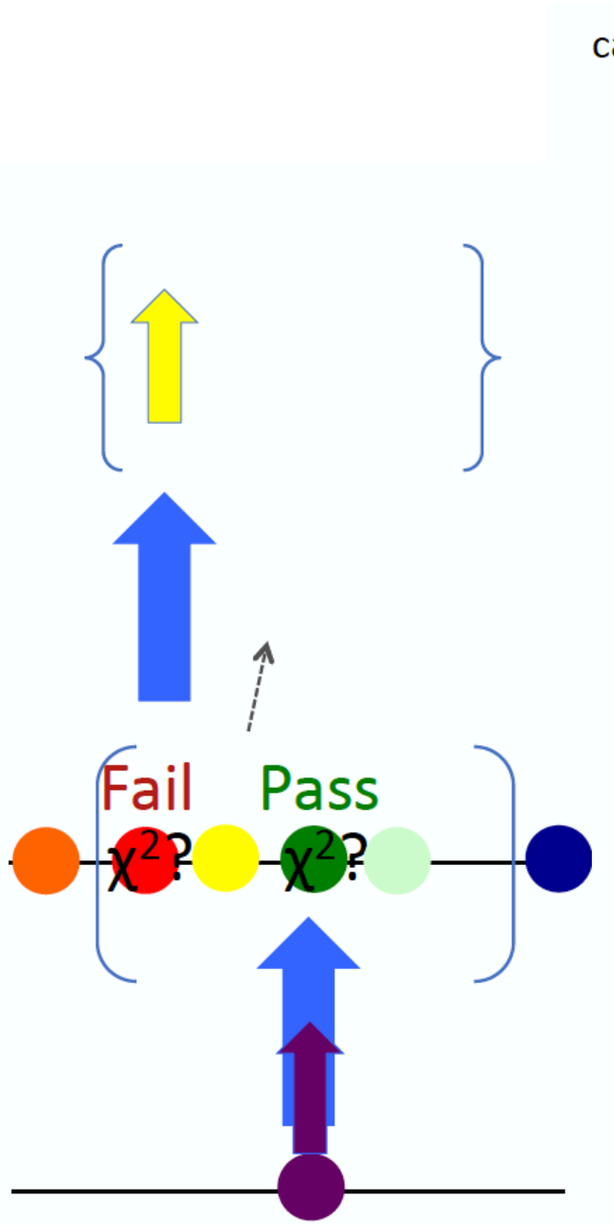


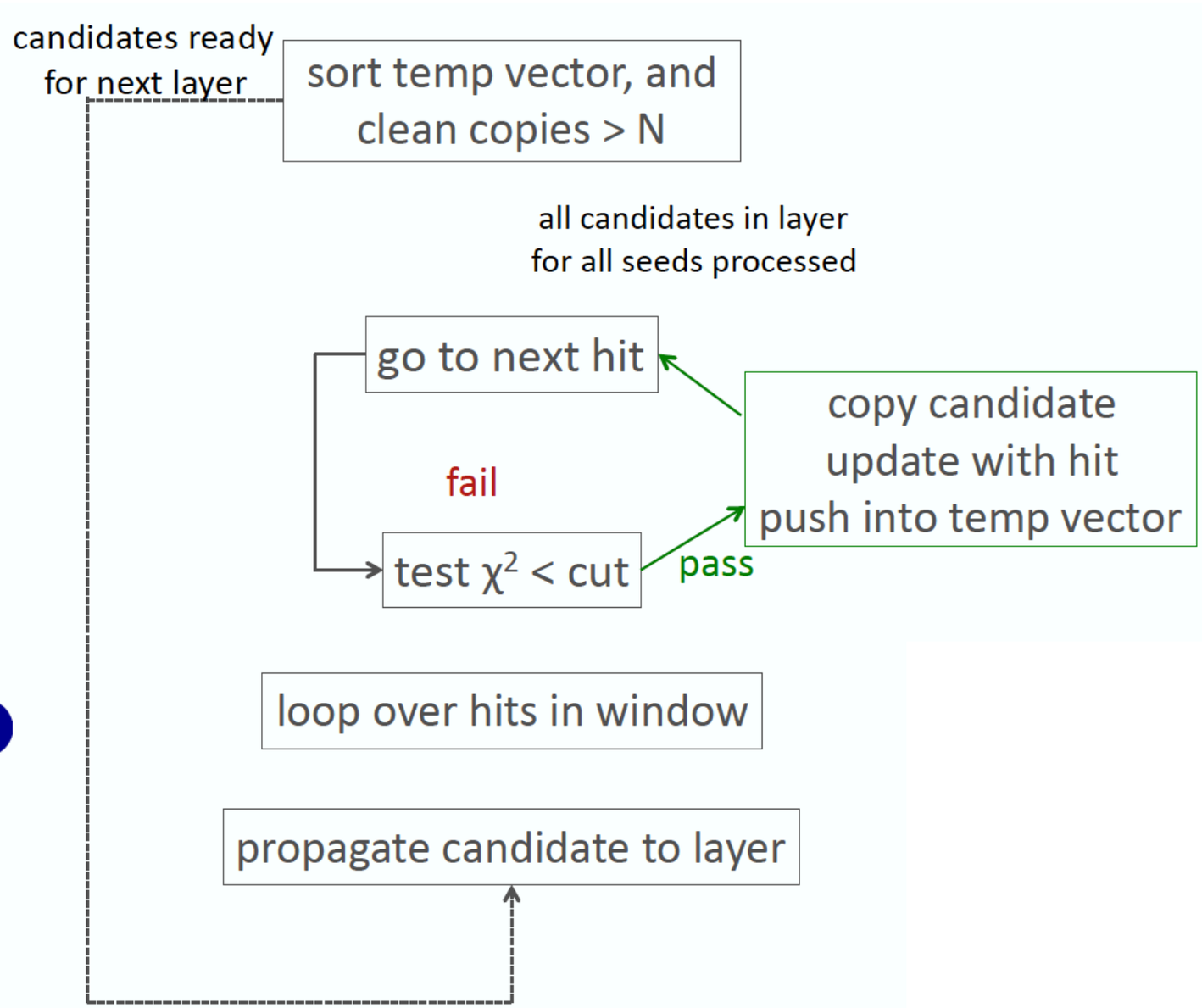




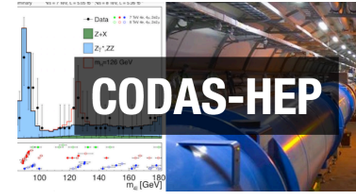
Handling Multiple Candidates (1)

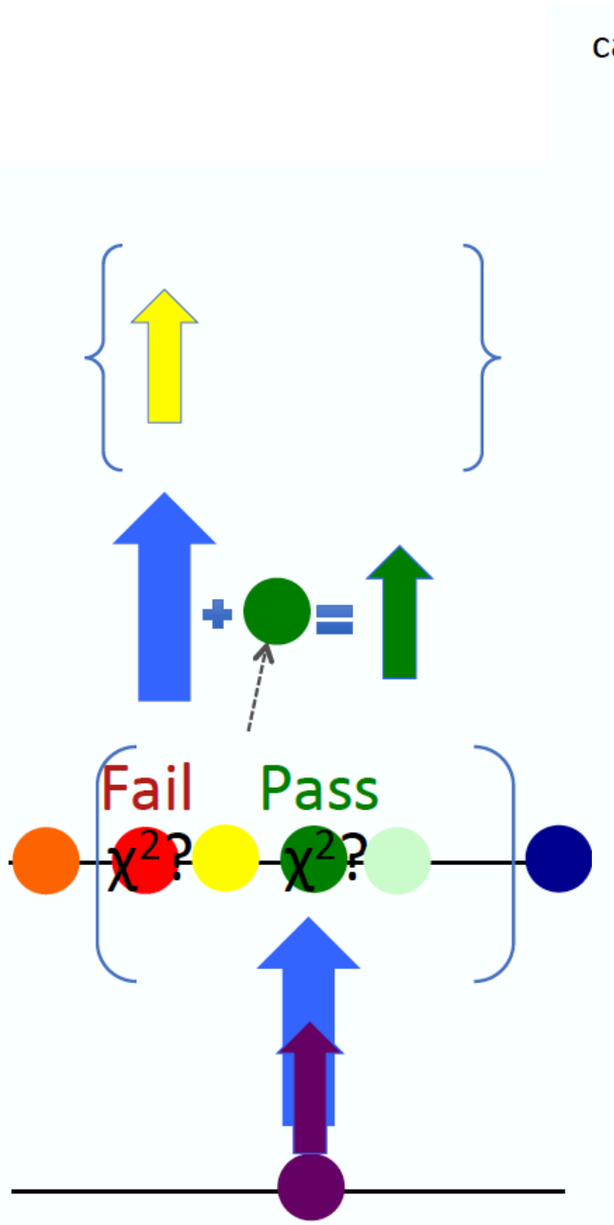


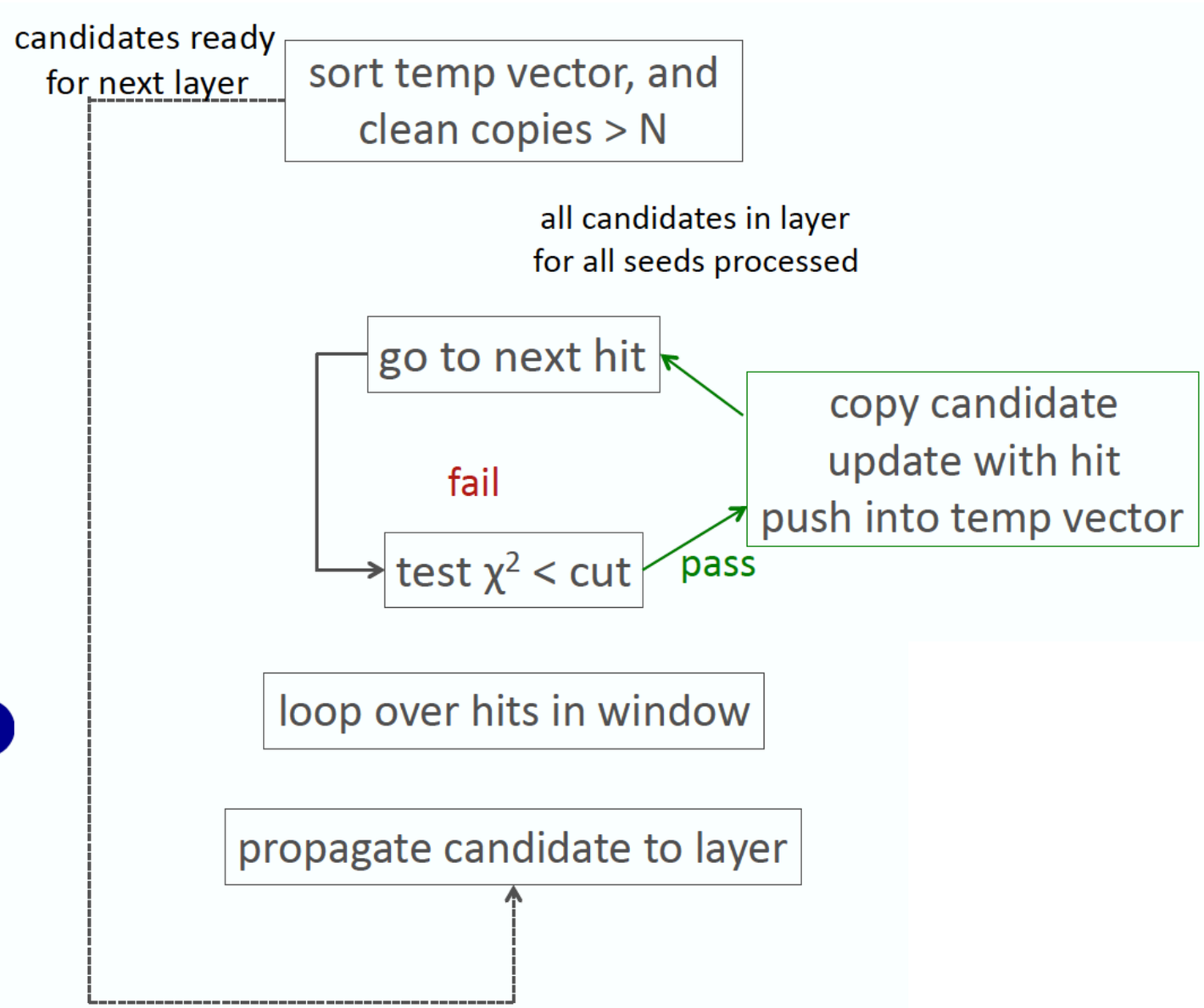




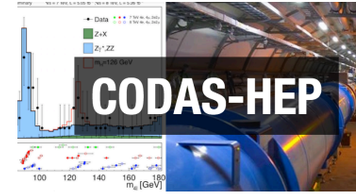
Handling Multiple Candidates (1)

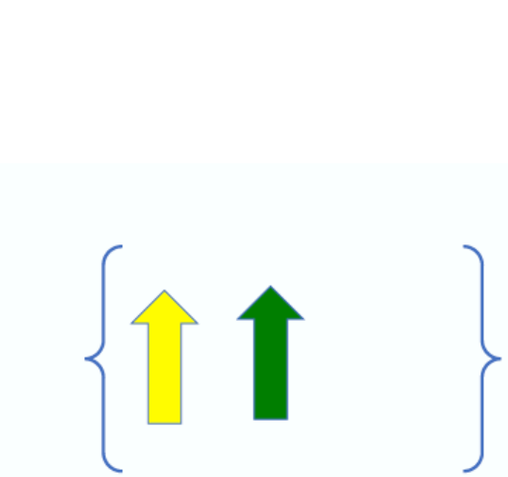


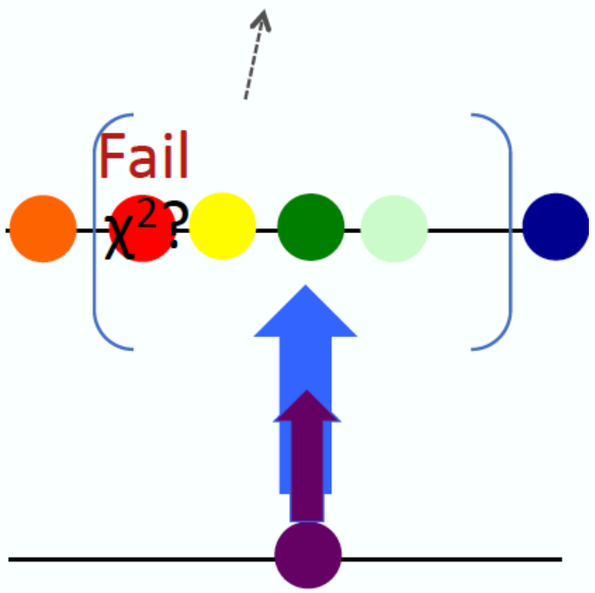


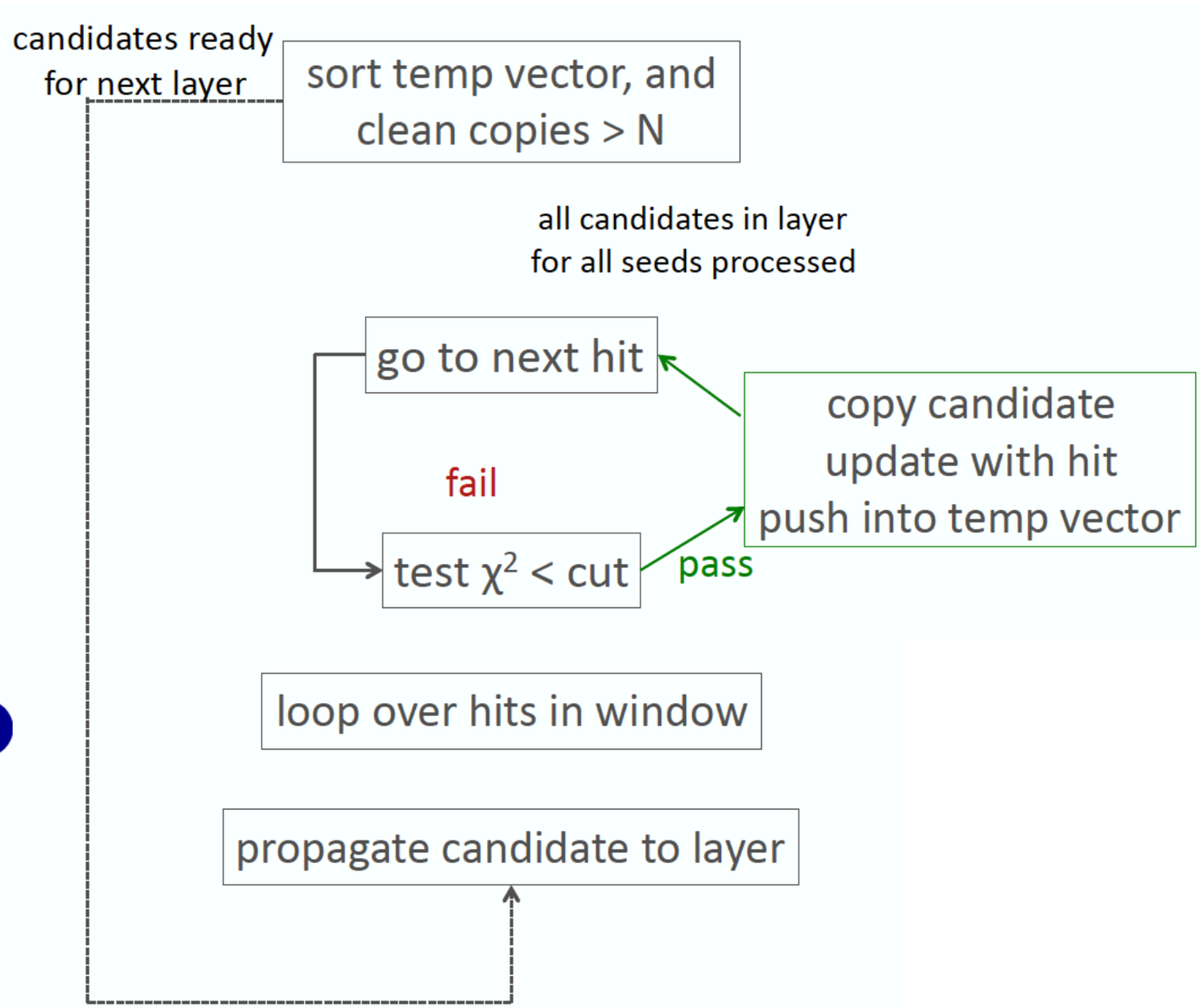


Handling Multiple Candidates (1)

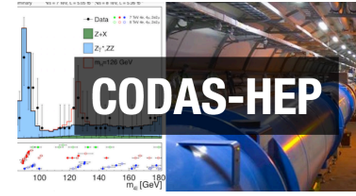


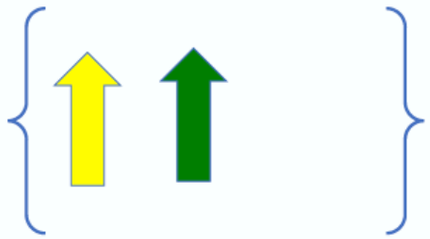


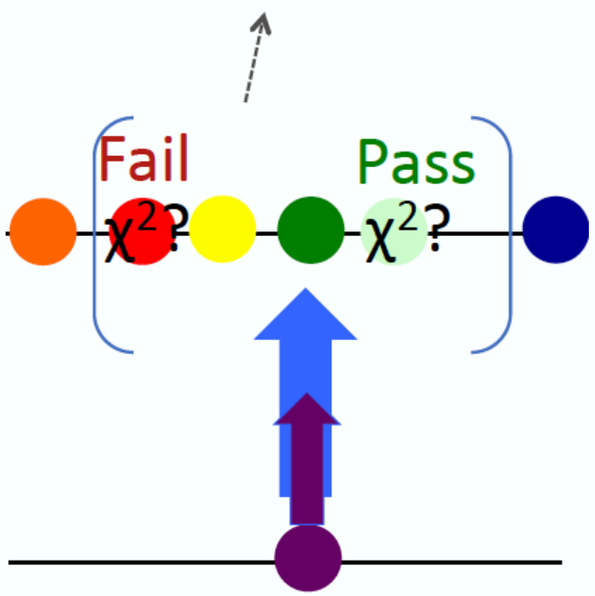


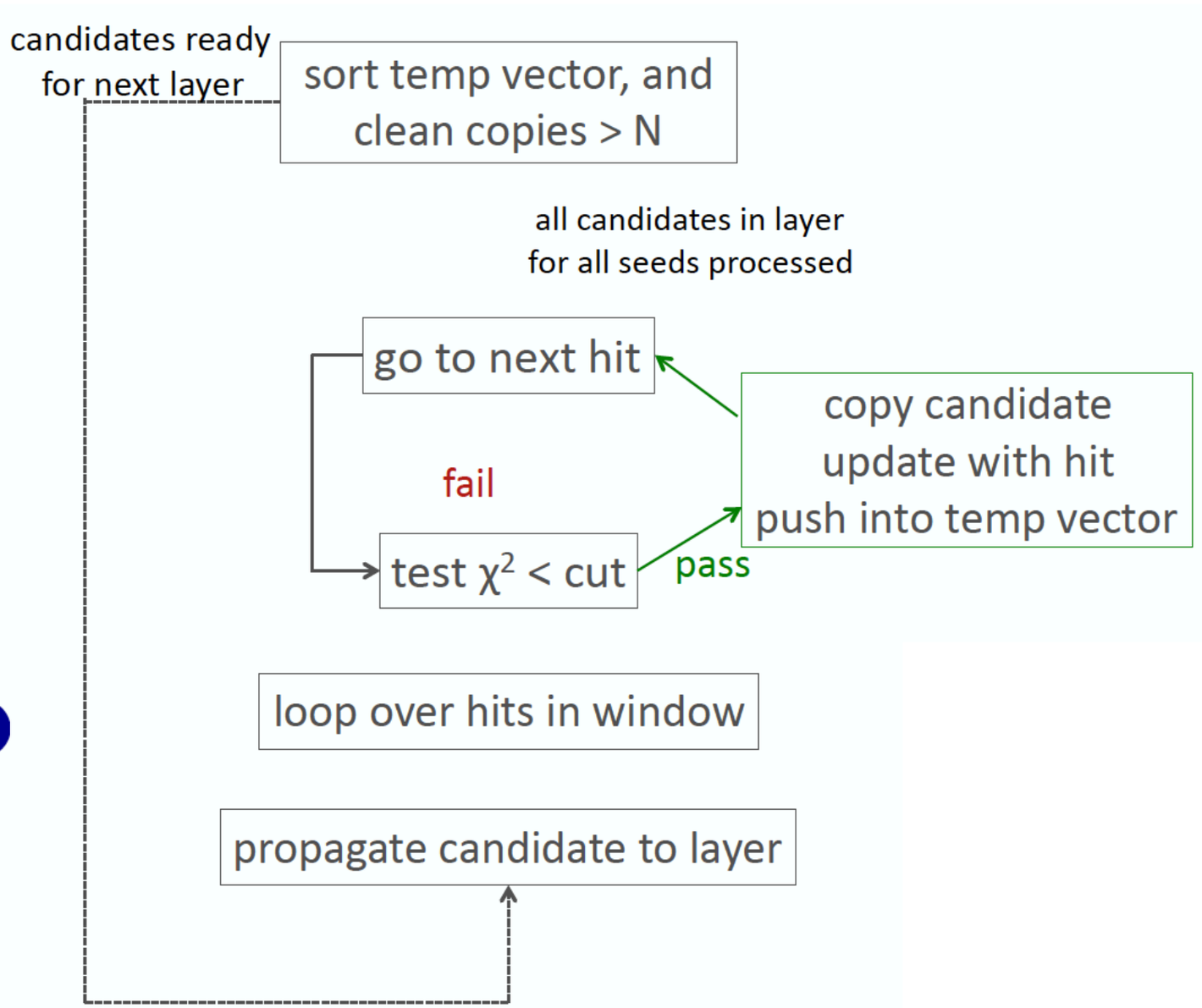


Handling Multiple Candidates (1)

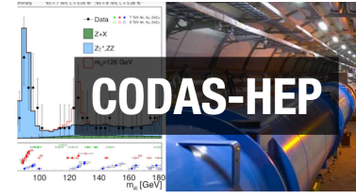


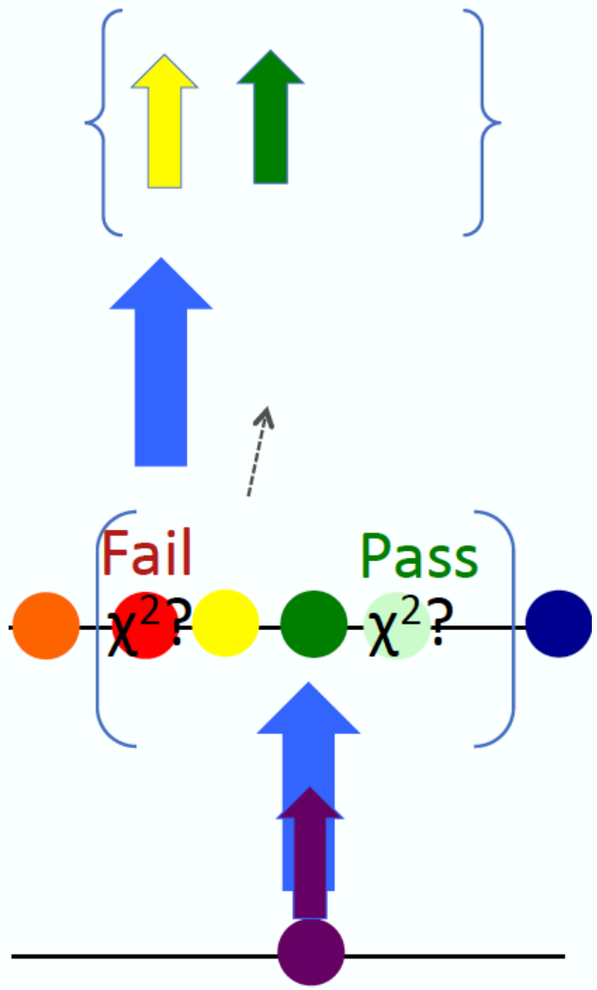


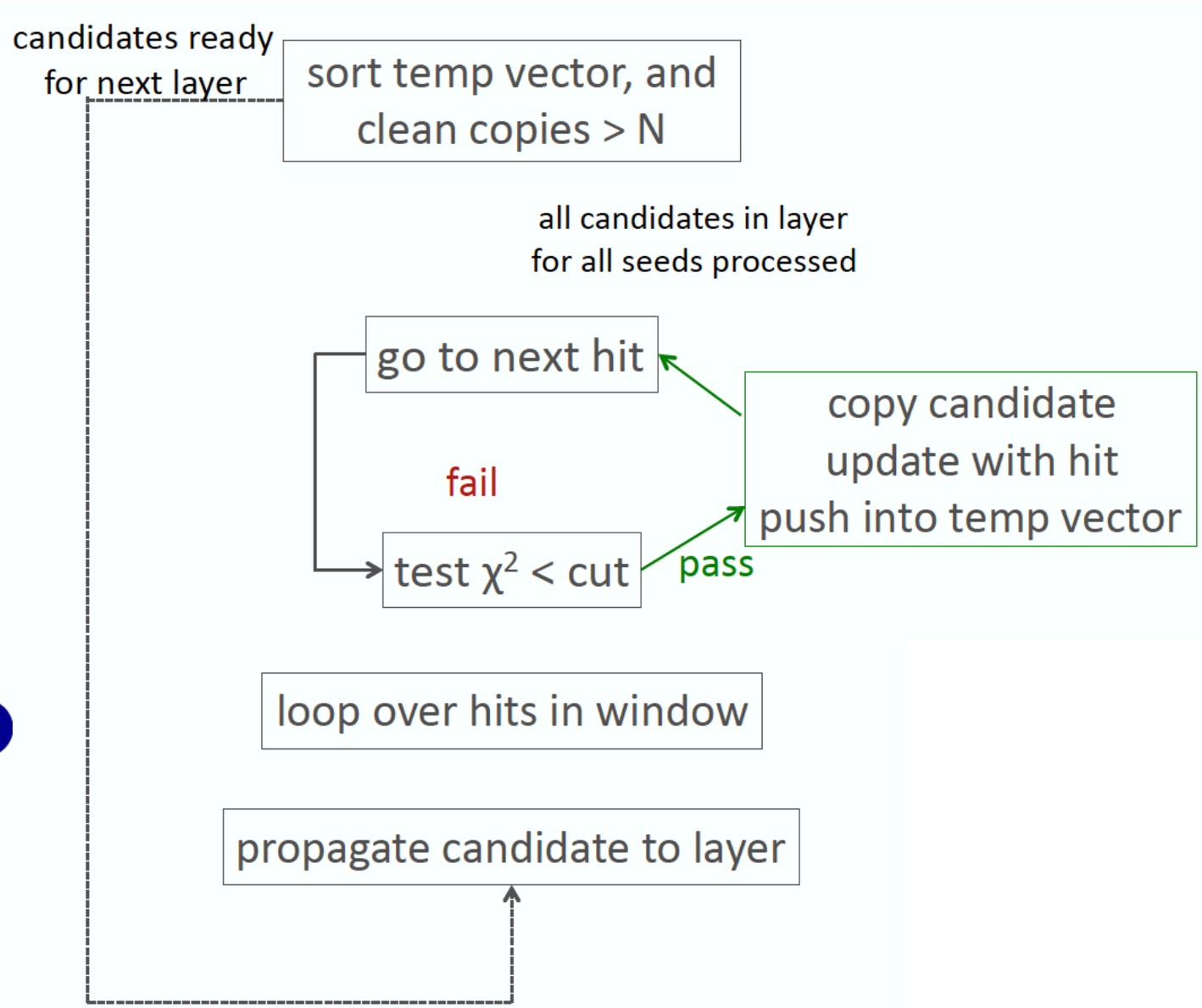




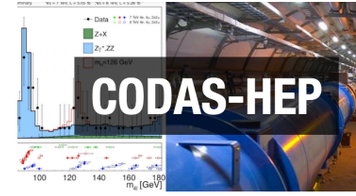
Handling Multiple Candidates (1)

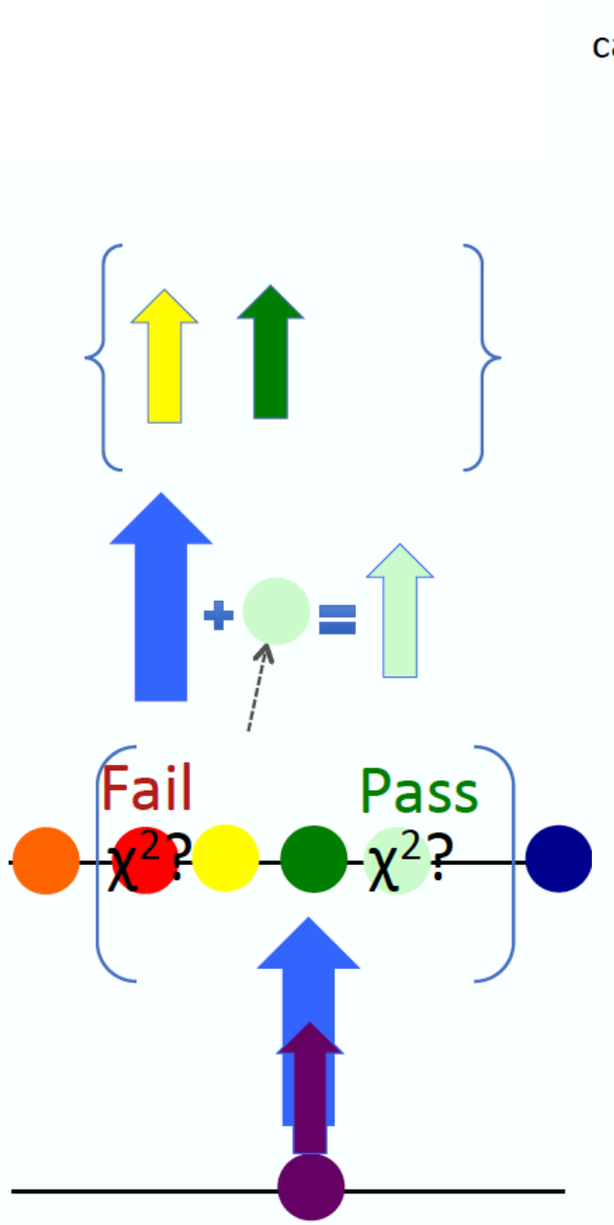


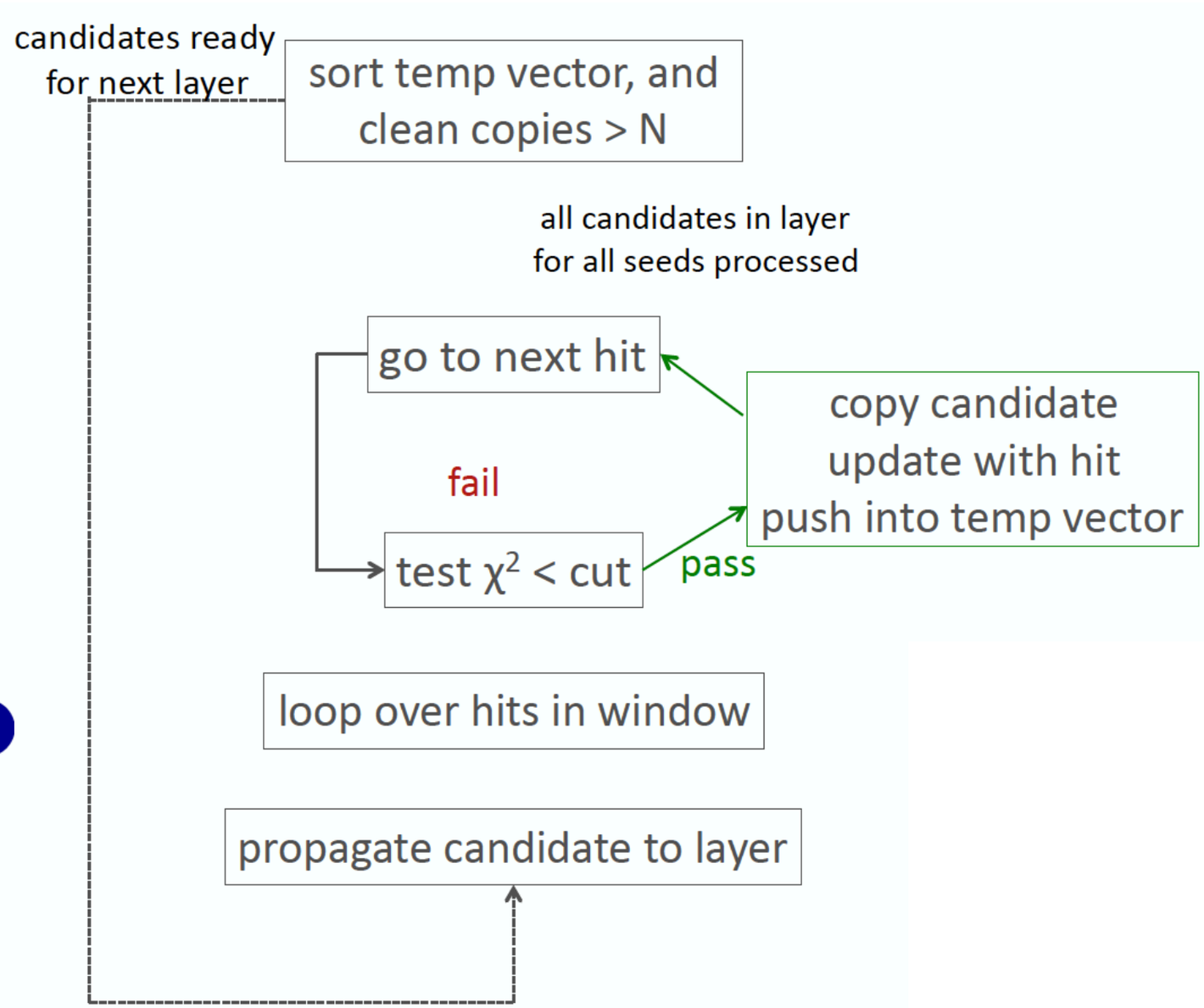




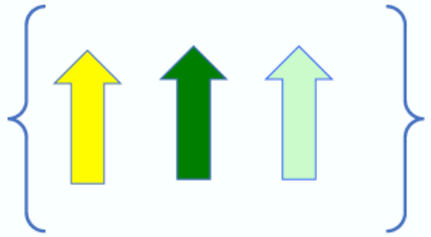
Handling Multiple Candidates (1)

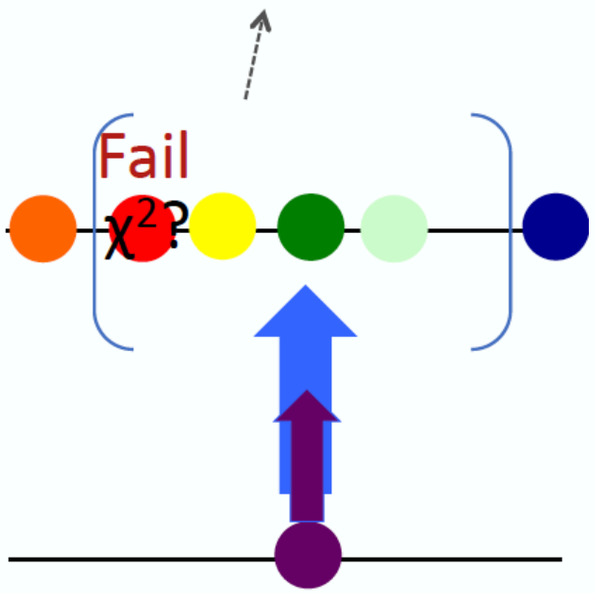


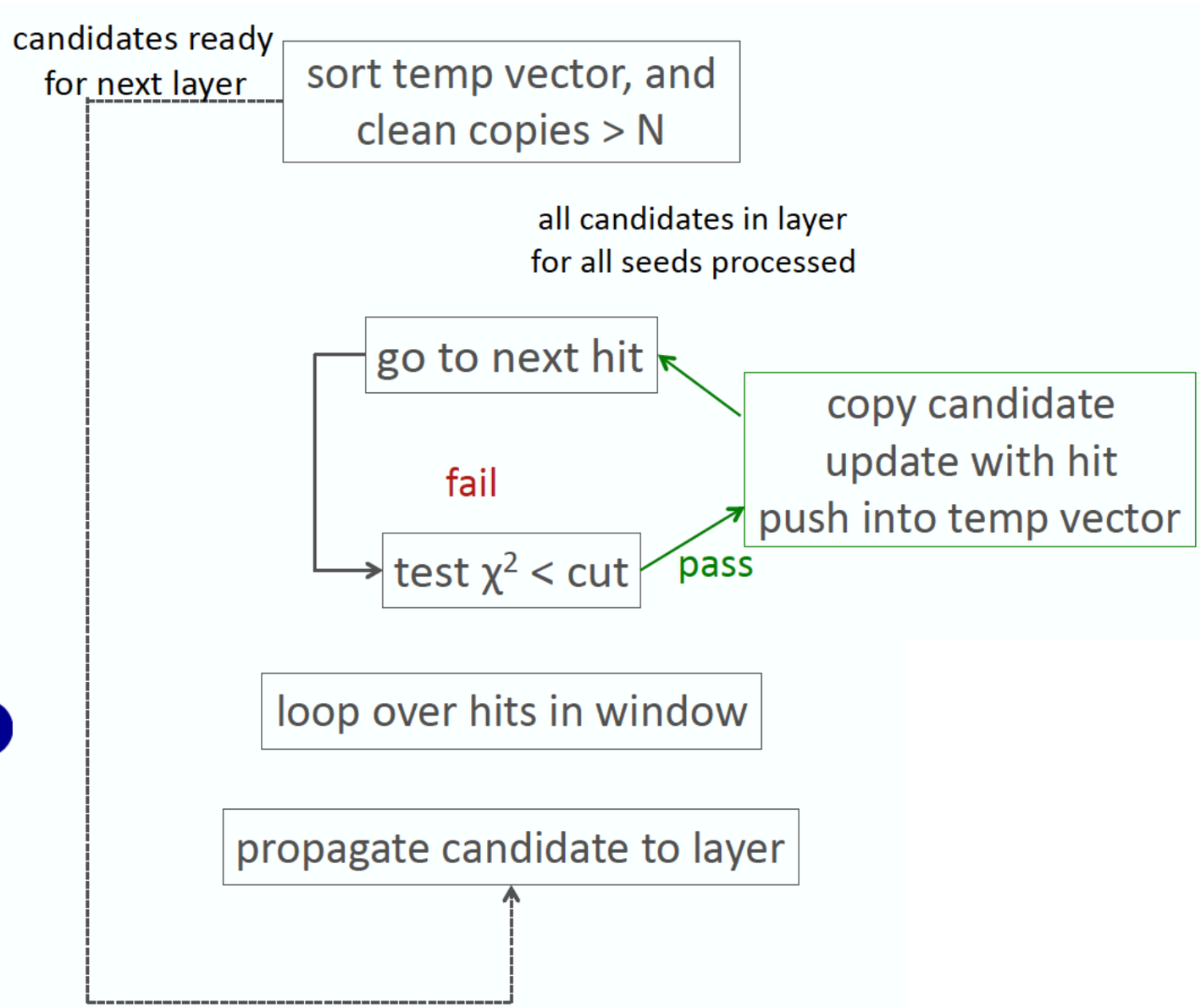




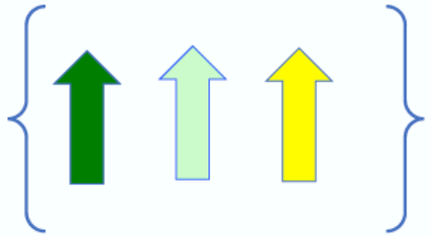
Handling Multiple Candidates (1)

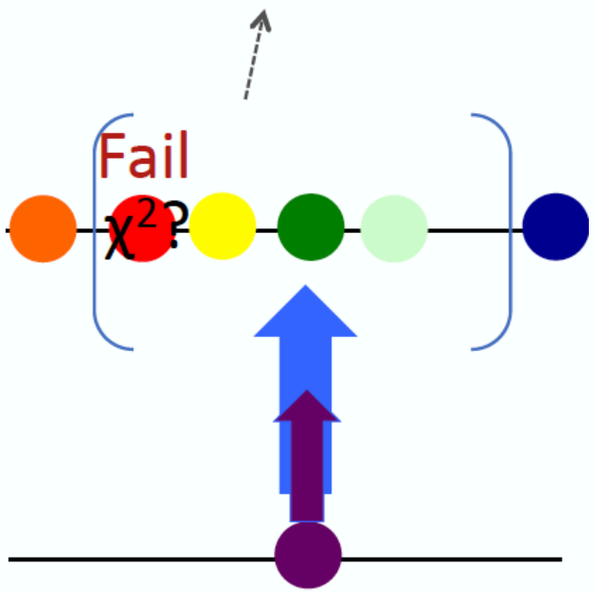


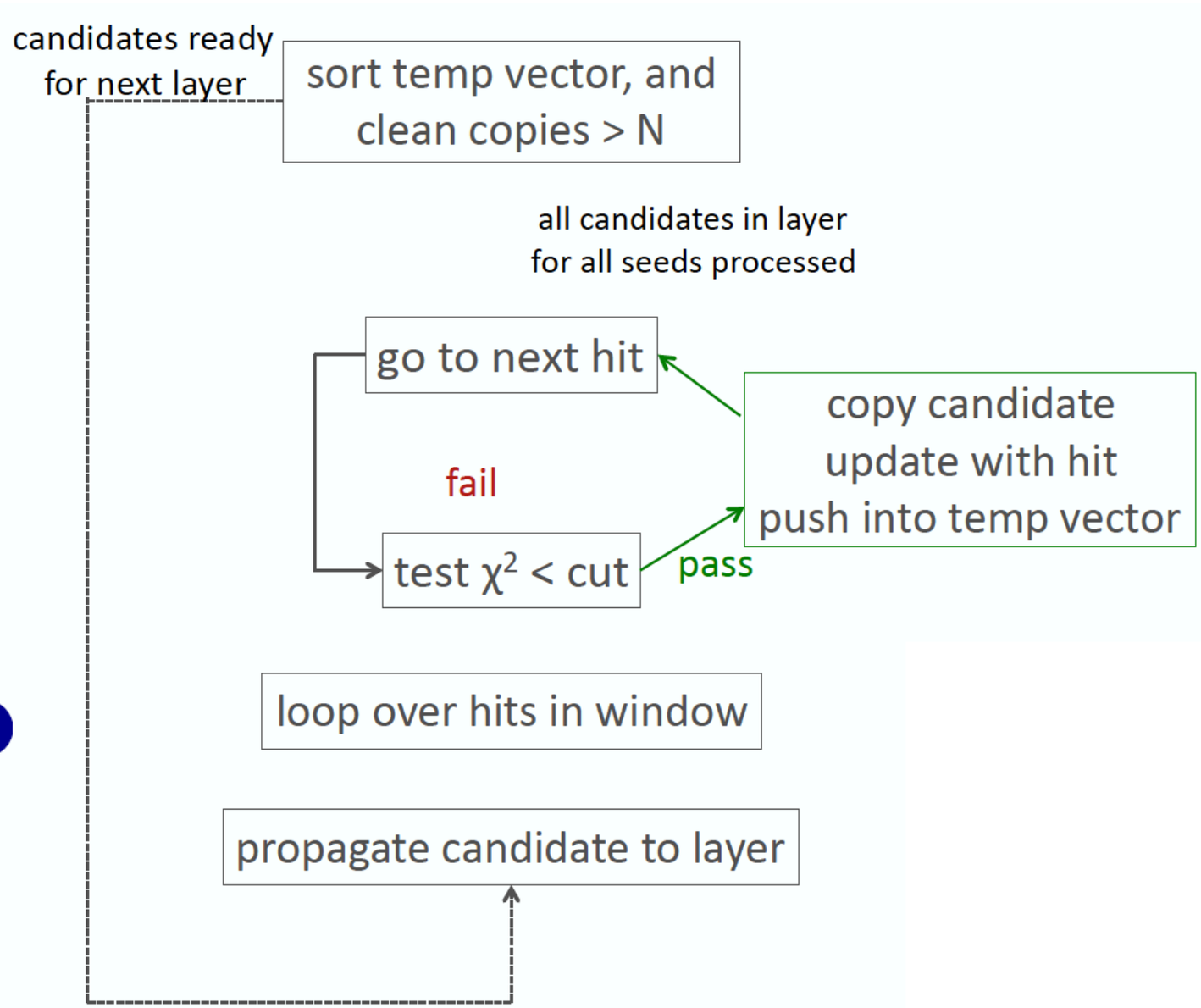




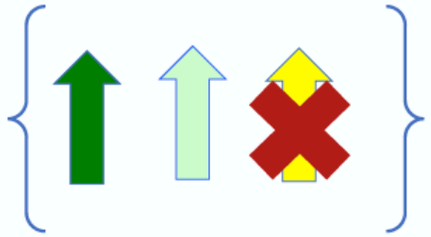
Handling Multiple Candidates (1)

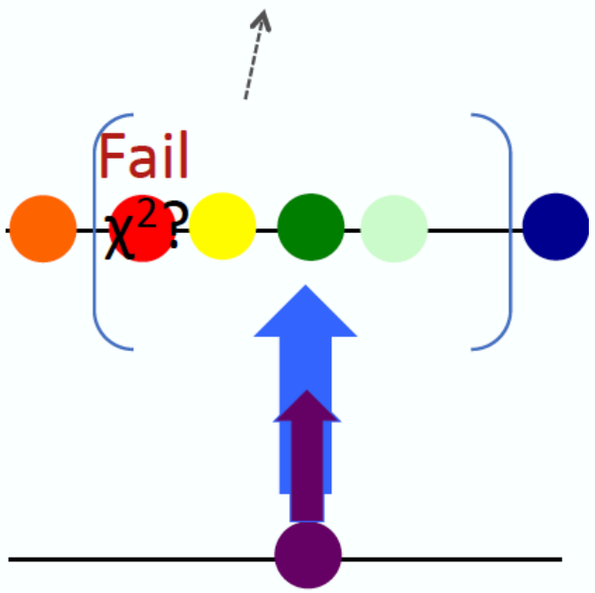


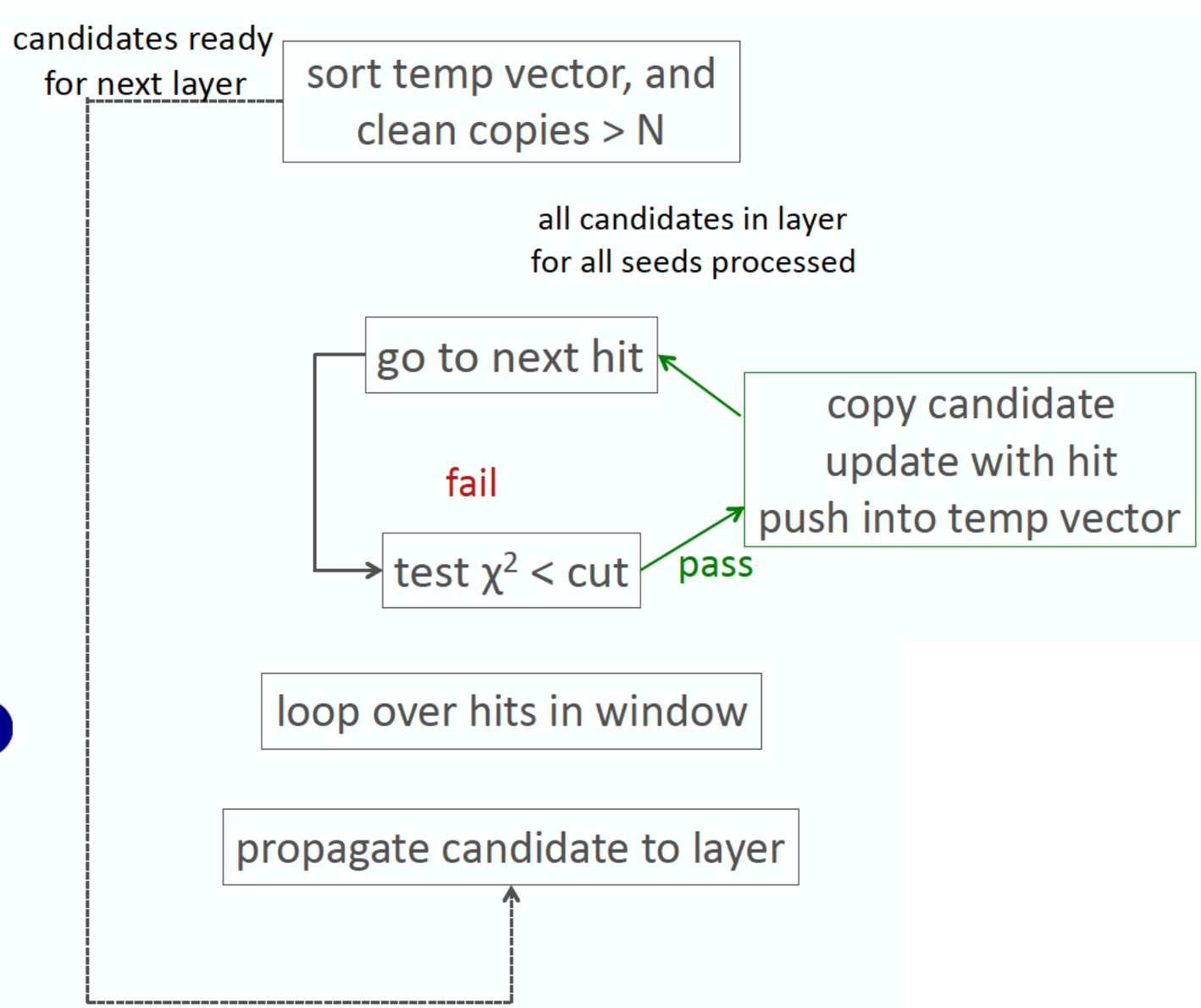




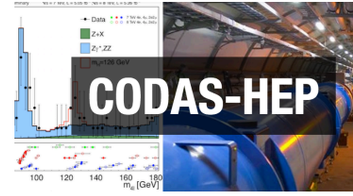
Handling Multiple Candidates (1)



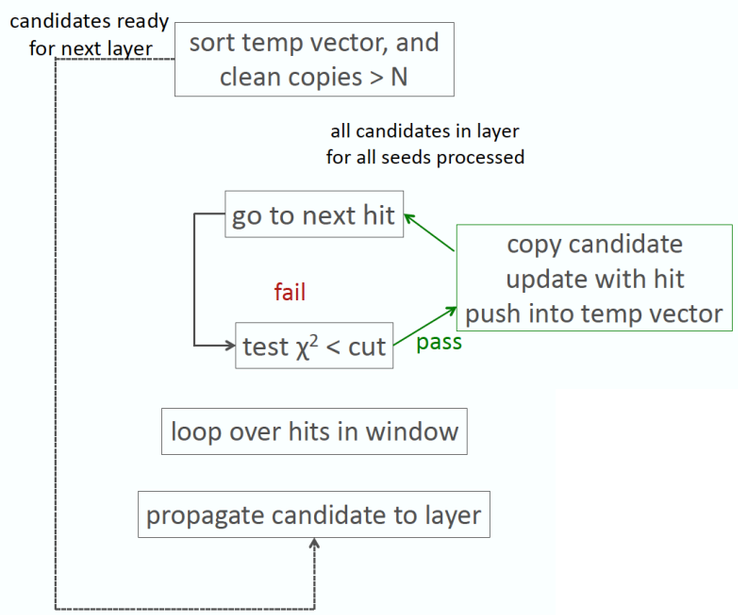




Handling Multiple Candidates (1)

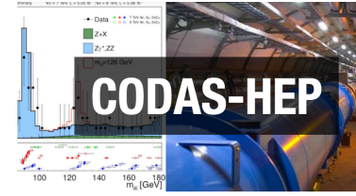


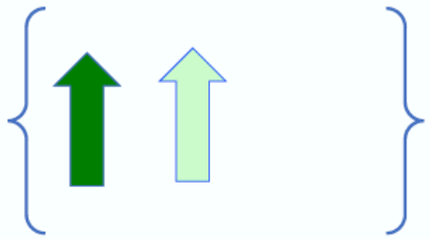
- Parallelize and vectorize this:

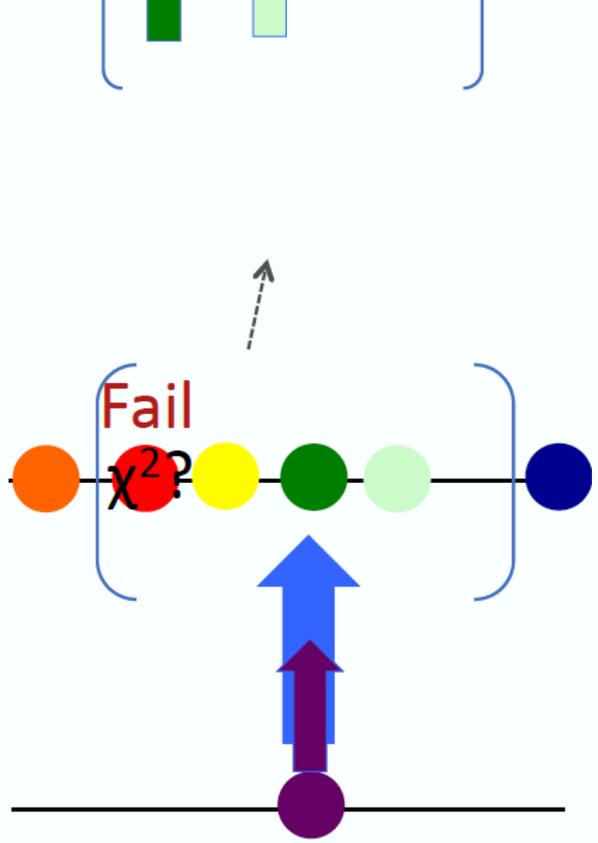


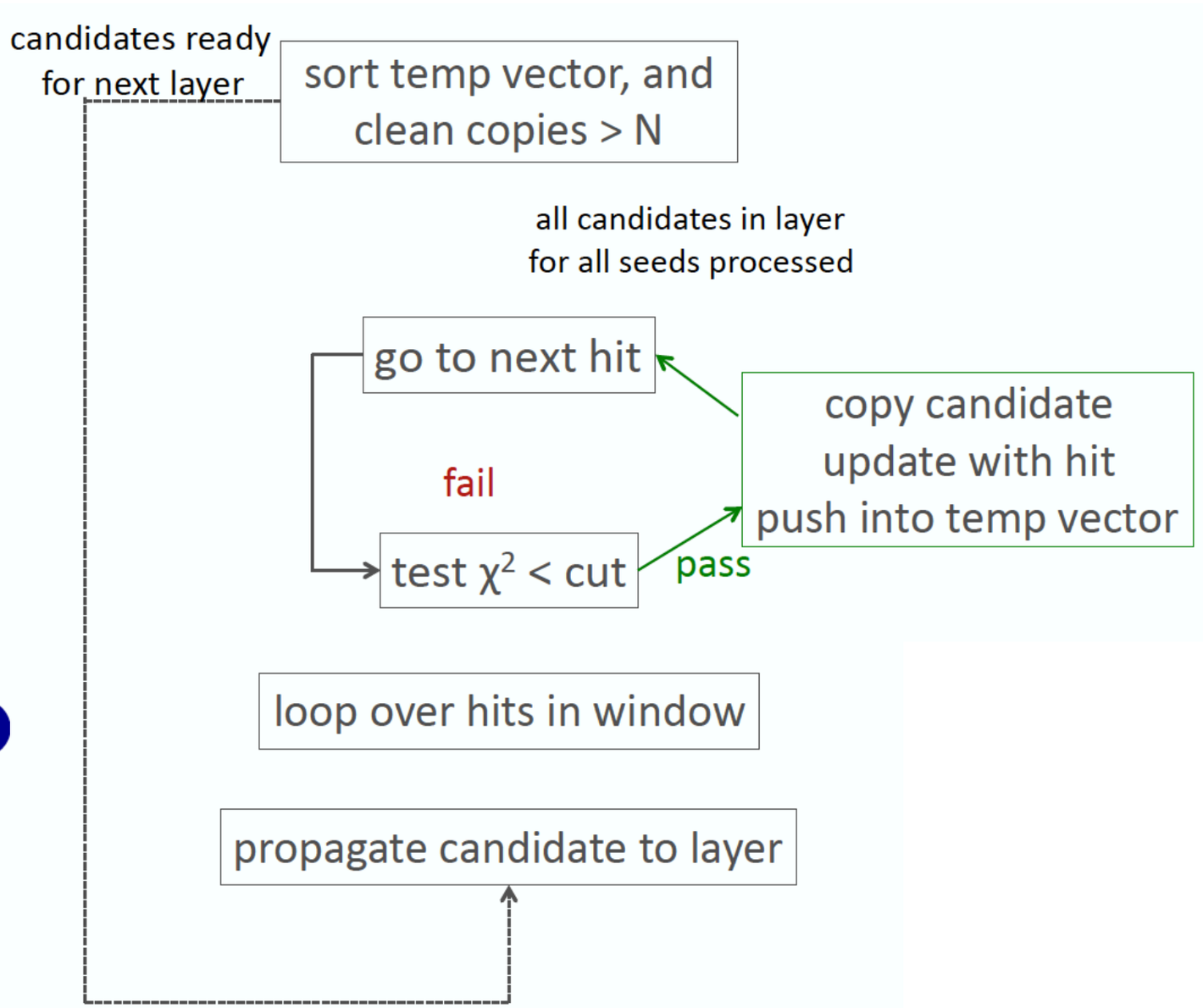
- Split work into independent task elements. This is enough for parallelization. Pretty much required for vectorization.
  - ✳ **Depth 1: Different events are independent**
    - ✳ **Depth 2: Regions in |η|**
      - ✳ **Depth 3: Candidates starting from different seeds**
        - ✳ **What's left here has an opportunity for vectorization.**
- For vectorization group independent execution elements which can be done in exactly the same sequence of operations
  - ✓ **We group what goes into a single task at "Depth 3" in chunks that can fill a SIMD register width.**
    - ◉ **AVX512 with single precision can fit computation of 16 track candidates simultaneously**

Handling Multiple Candidates (1)





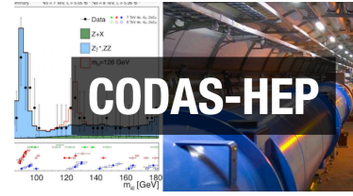




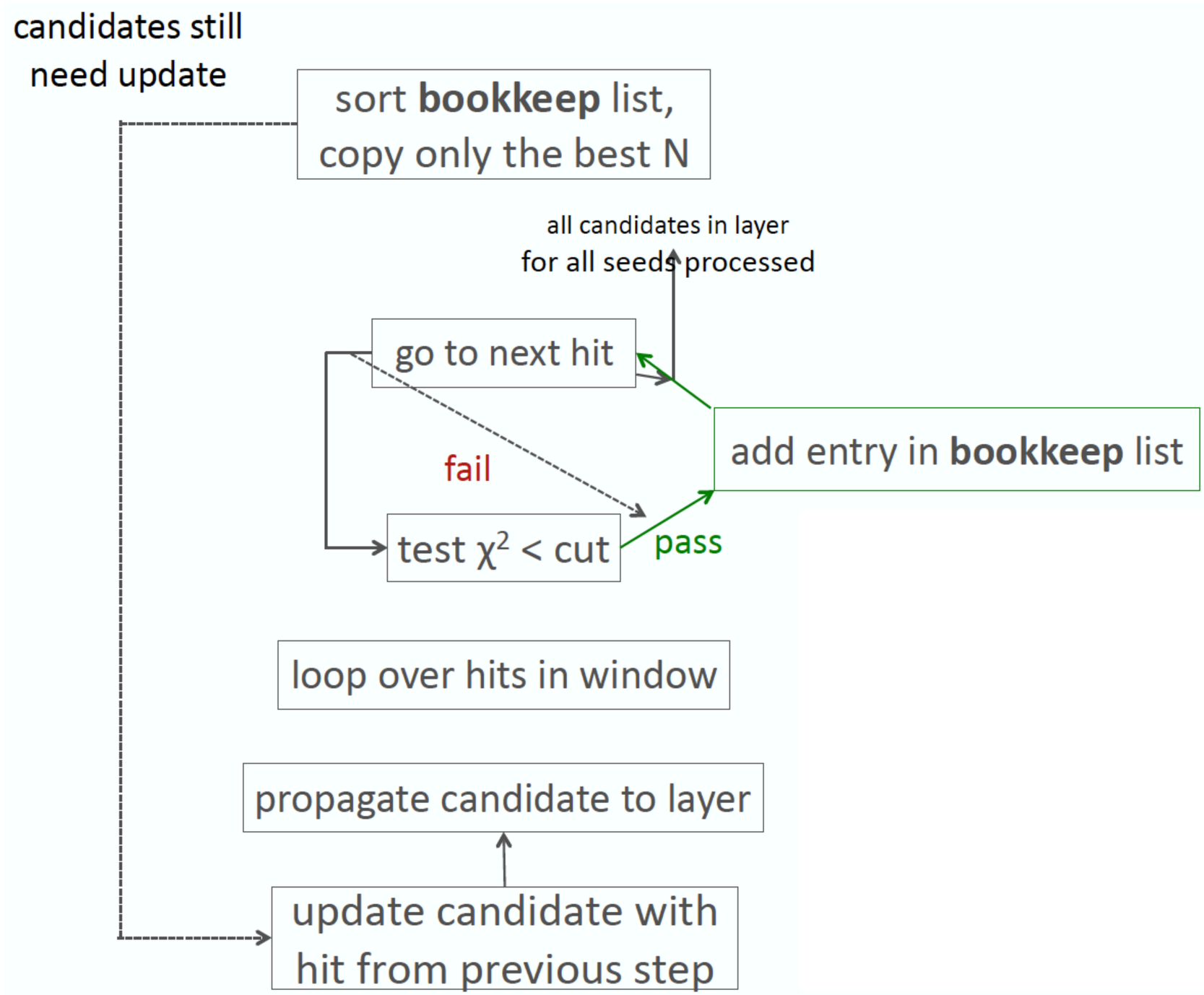
- Processing tracks in parallel, copy + update forces other processes to wait!

➡ **We need another approach**

“Clone engine”

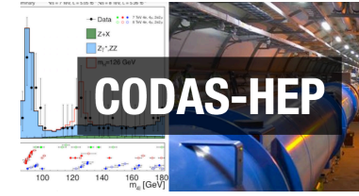


- “Clone Engine”: reorganize the operations of the simple approach to enable more complete vectorization

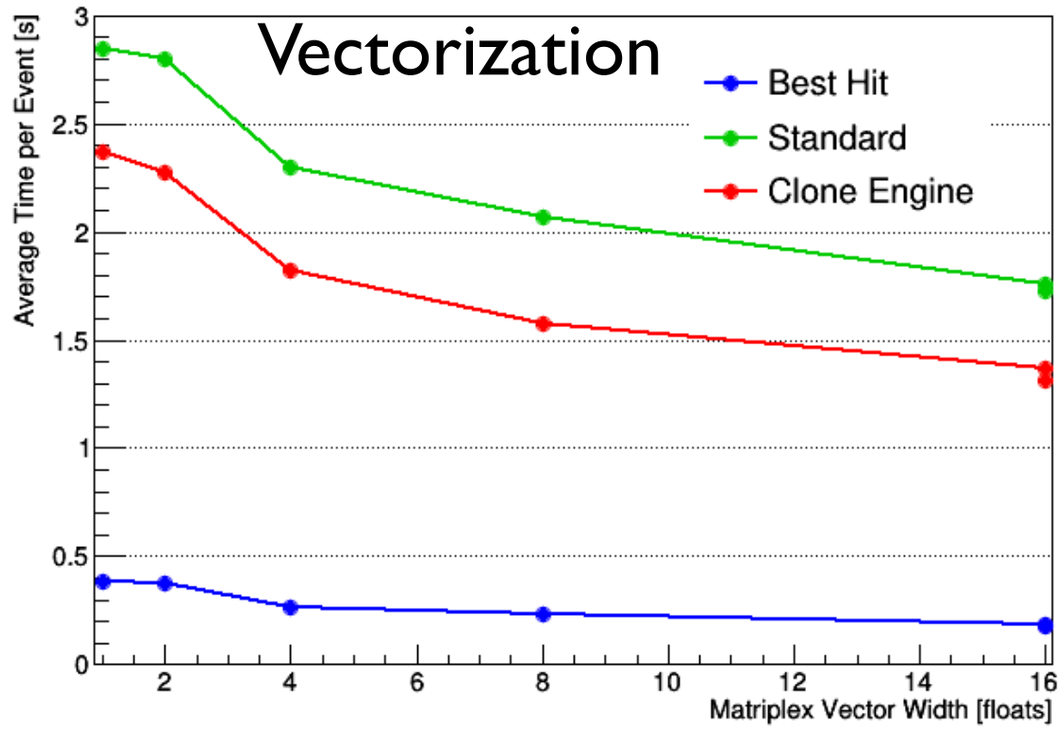


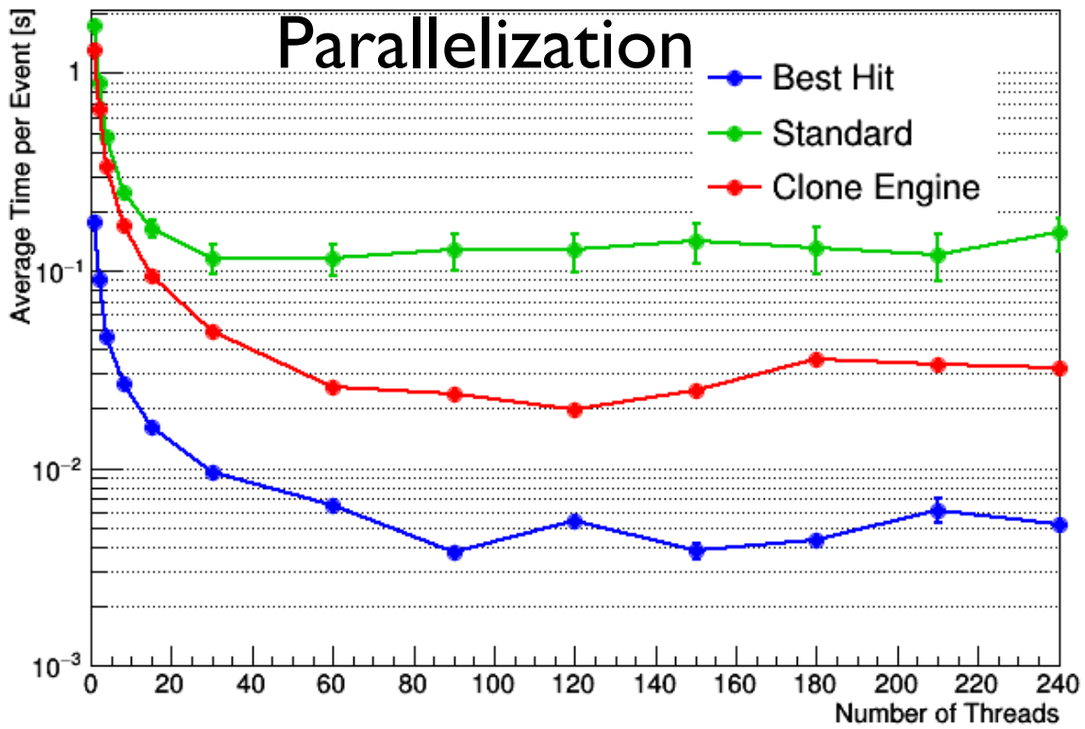
- Clone Engine approach matches physics performance of previous approach!

Compare performance

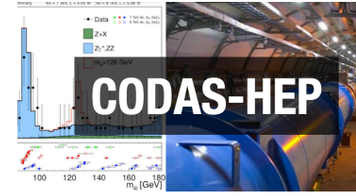


- "Clone engine" is evidently faster
  - ✓ **Scaling vs vector width, however, is similar**
- Smarter arrangement of data structures and operations made on them allows for more effective use of memory I/O



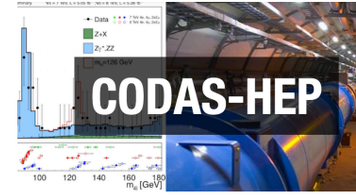


Track building: lessons

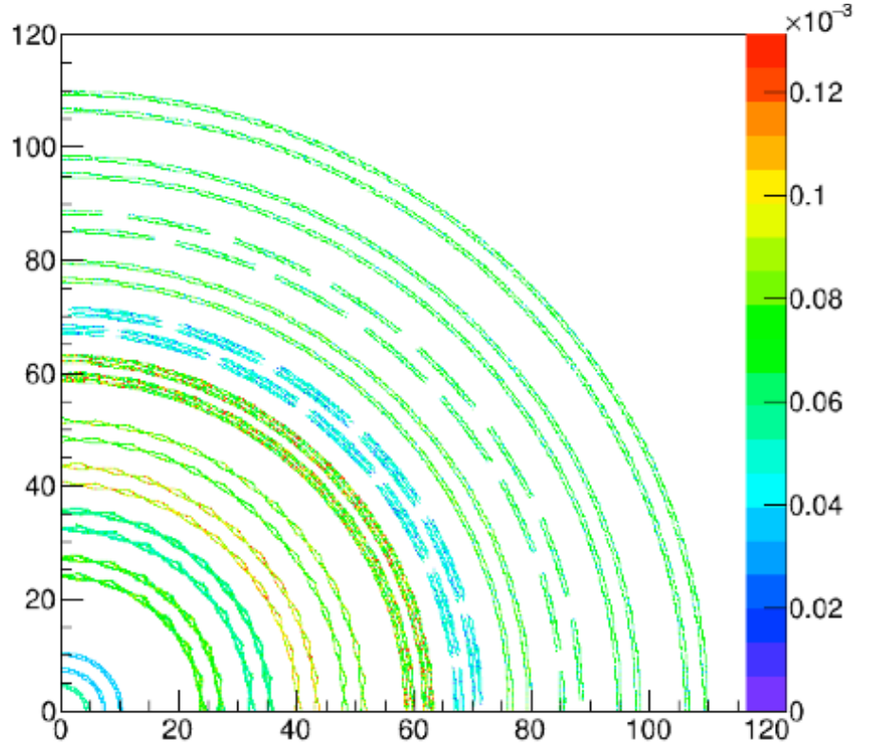


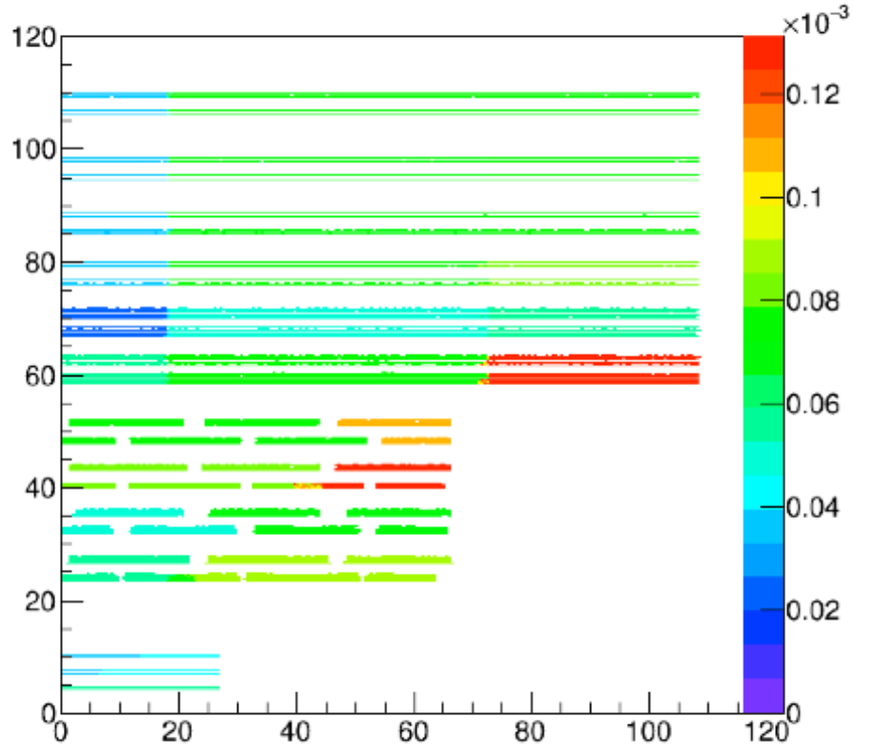
- Data locality is critical (w/speedups, compared to earlier version):
  - ✓ **Optimize/vectorize copying of tracks into Matriplex (+20%)**
  - ✓ **Minimize dynamic memory allocations (+45%)**
  - ✓ **Avoid unnecessary object instantiations, copies (+25%)**
  - ✓ **Minimize size of data structures, smarter low-level algorithms (+30%)**
- Parallelization — different toolsets (OpenMP vs TBB)
  - ✓ **Static binning with OpenMP led to "tail effects" due to variable distribution of work**
  - ✓ **TBB work-stealing is an easy way to even out load variability**
  - ✓ **Optimizing work partition size still critical—too large doesn't allow enough balancing, too small has high over head costs**

Parallel KF: adding realism

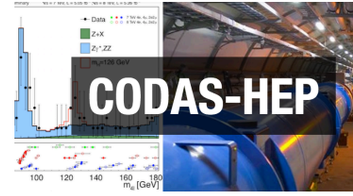


- Move beyond our circular cow
  - ✓ **Ultimately need to include realistic geometry, material effects, inefficiencies, overlaps, etc.**
  - ✓ **Use CMS simulation, add complexity in incremental steps**
- Two step propagation to avoid using the full geometry
  - ✓ **Simple parameterization of CMS geometry and material**
  - ✓ **Step 1: propagate to the average radius of the layer**
  - ✓ **Step 2: propagate to the exact hit radius**
- Endcap/Disks
  - ✓ **Propagate to z, similar handling of material and propagation**
- Use seeds from CMS (CA algorithm)
  - ✓ **Focus on the seeds made for the first iteration of CMS tracking**

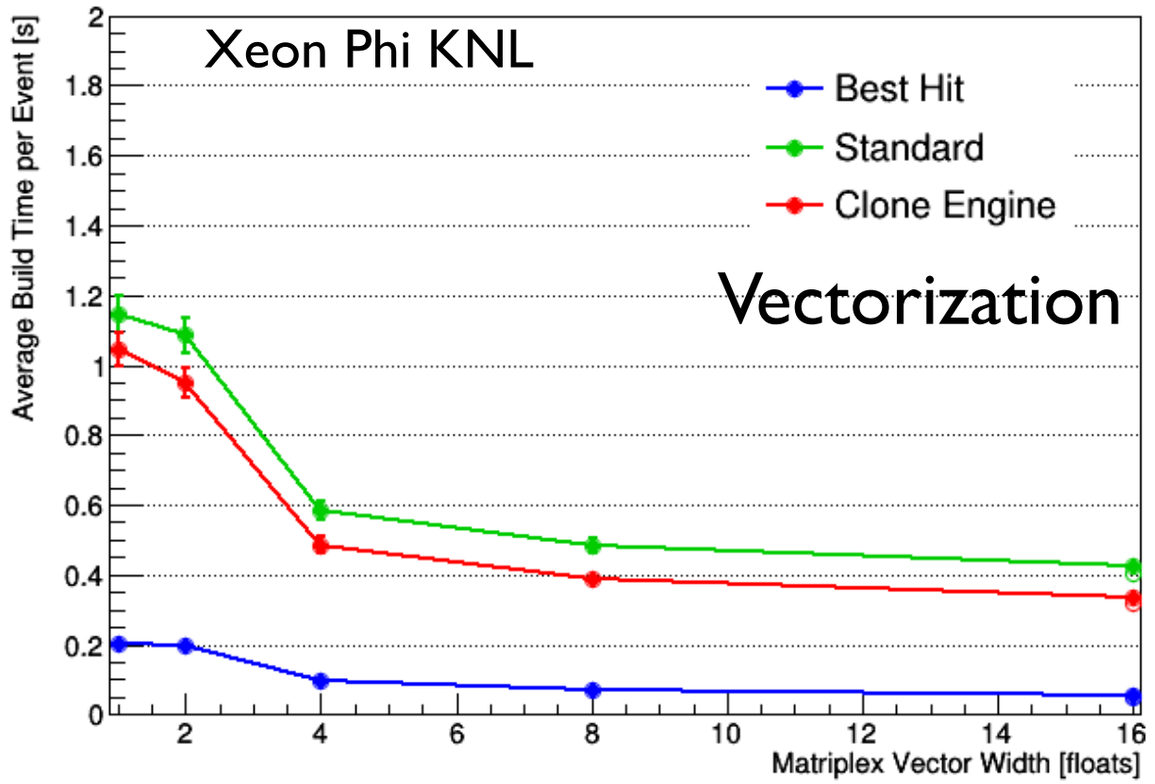




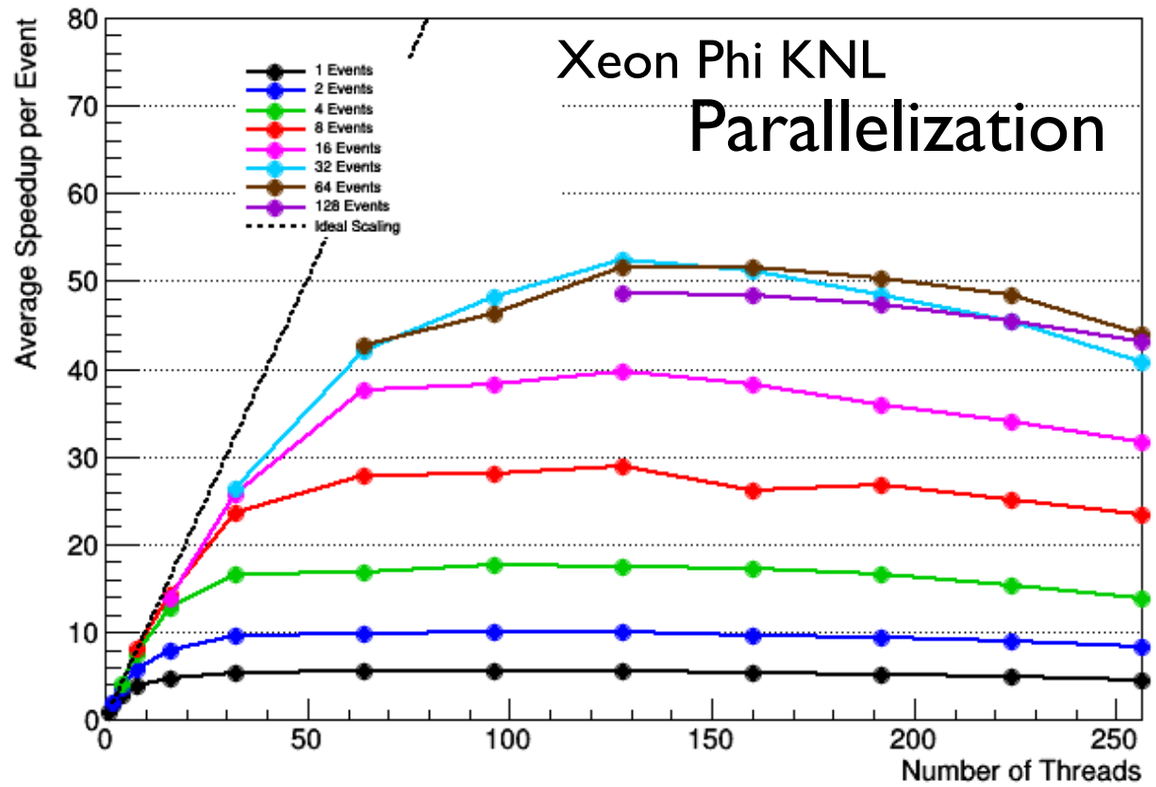
Performance on events with PU70



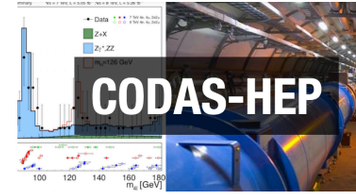
- Significant gains from vectorization and parallelization
- NB: the CMS simulated PU70 events actually have fewer tracks per event than the toy detector tests
- For effective parallelization it is important to be able to run multiple events in parallel to evenly distribute work among threads



- Compared to CMSSW, mkFit is about 10x faster (both single-thread).
  - **Intentionally vague [work in progress]**
- Event throughput under full node load
  - **KNL: ~120 Hz**
  - **Skylake: 250 Hz**

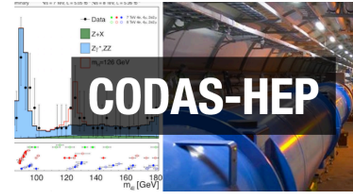


Wrapping up



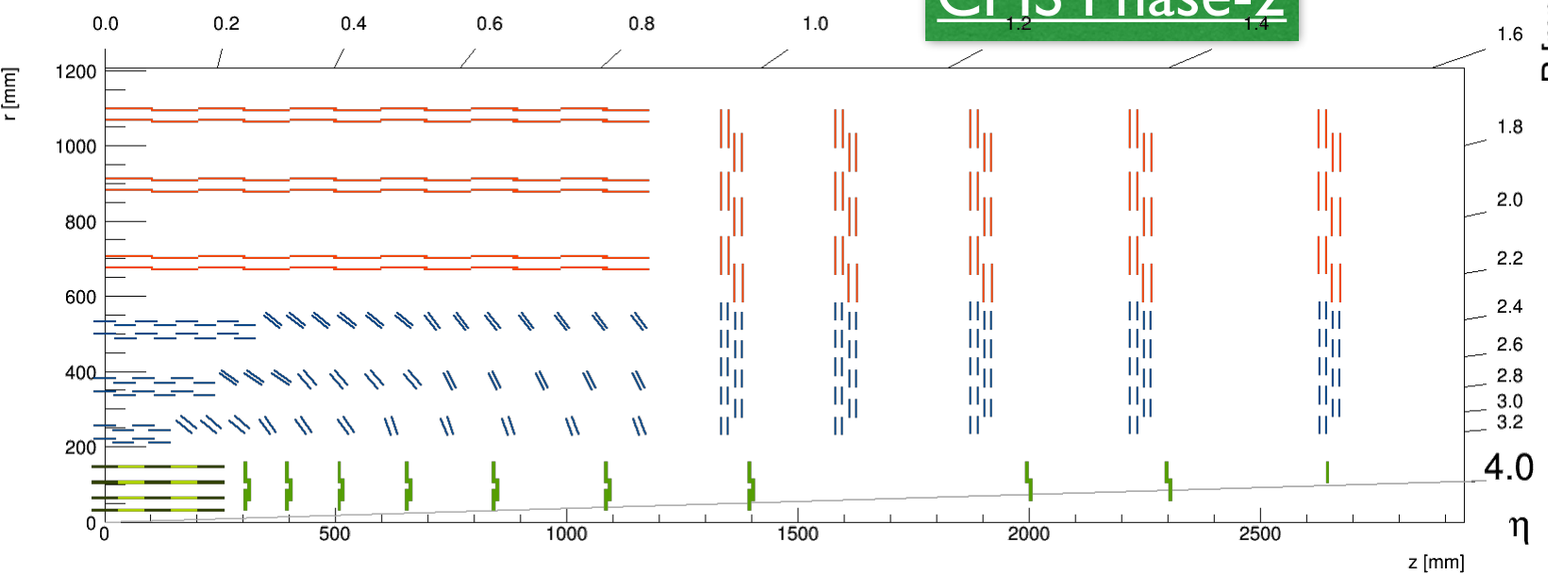
- Gave you a flavor or track reconstruction in collider experiments
  - ✓ **Starting from basics**
- Covered specific parallel tracking solutions, including parallel KF project
  - ✓ **Not enough time to really talk about the full project - a lot of work done on GPGPU on the same project too …**
  - ✓ **Attempt to take a real problem and use some of the tools you've heard about this week (parallel programming)**
- Ties into HPC computing are essential
  - ◉ **tracking is the biggest part of event reconstruction timing**
- Showed you some of the challenges you run into, and a scale of the improvements we are able to get at this point in time
- These problems are hard!!!!
  - ✓ **Requires rethinking of algorithms, reorganization of data structures, keeping the resources busy**
  - ✓ **careful measurement and tuning to get at theoretically available performance**

Random backup follows

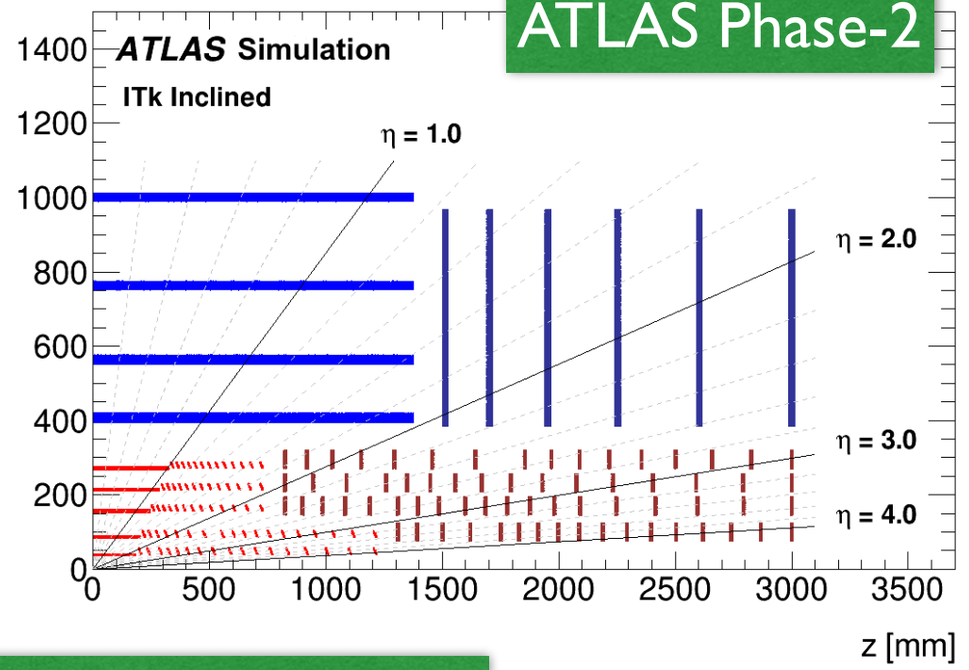


Future trackers

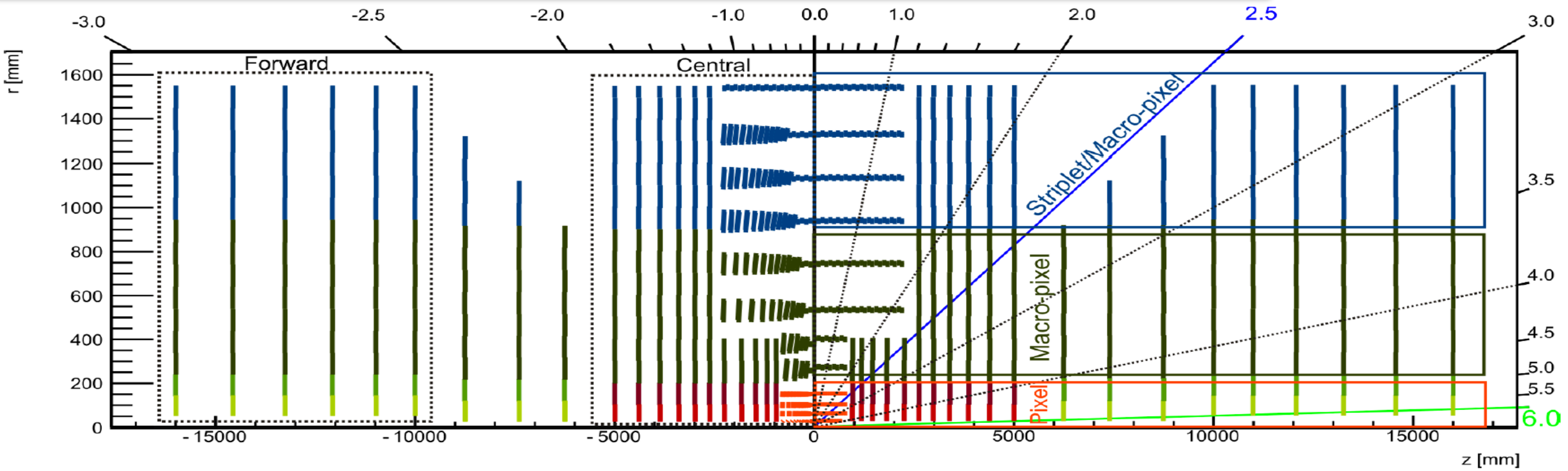
CMS Phase-2



ATLAS Phase-2

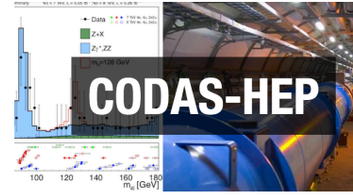


FCC-hh CDR: tilted (left) and flat (right) options https://indico.cern.ch/event/723382/



- Outer tracker design has a common ~equidistant layout theme

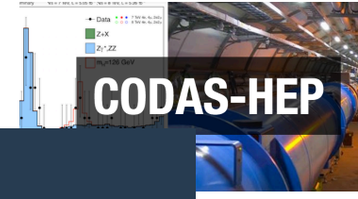
[ATLAS] Hardware track finding



From

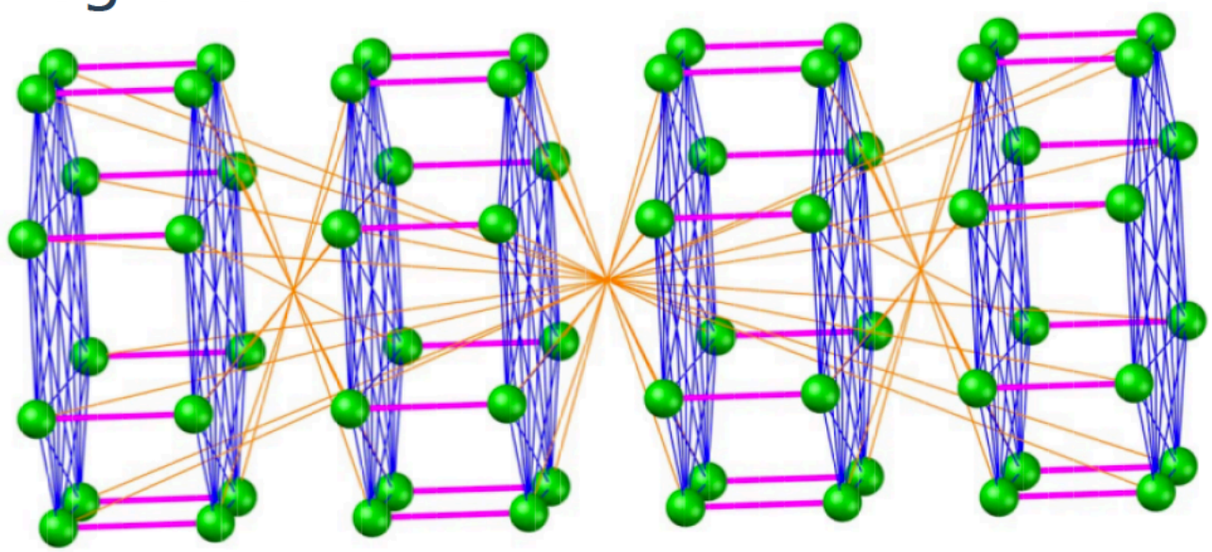
Todd Seiss

On behalf of the ATLAS Collaboration for Computing in High Energy Physics 2018



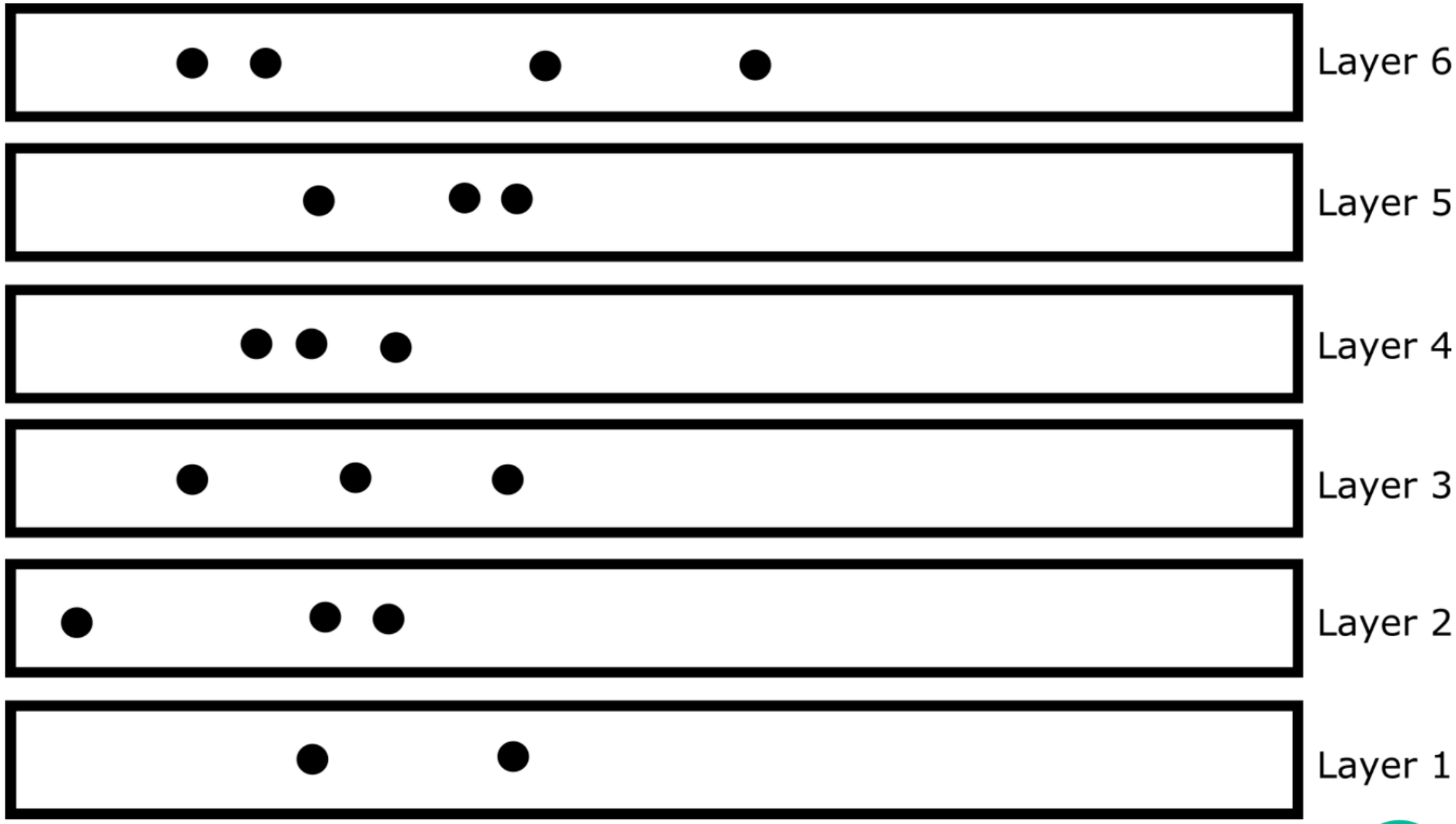
Hardware Tracking: Preprocessing

- **Cluster raw hits**
  - Sliding window clustering
- **Route Clusters into tracking regions**
  - Inner detector is divided into independent parallel tracking regions

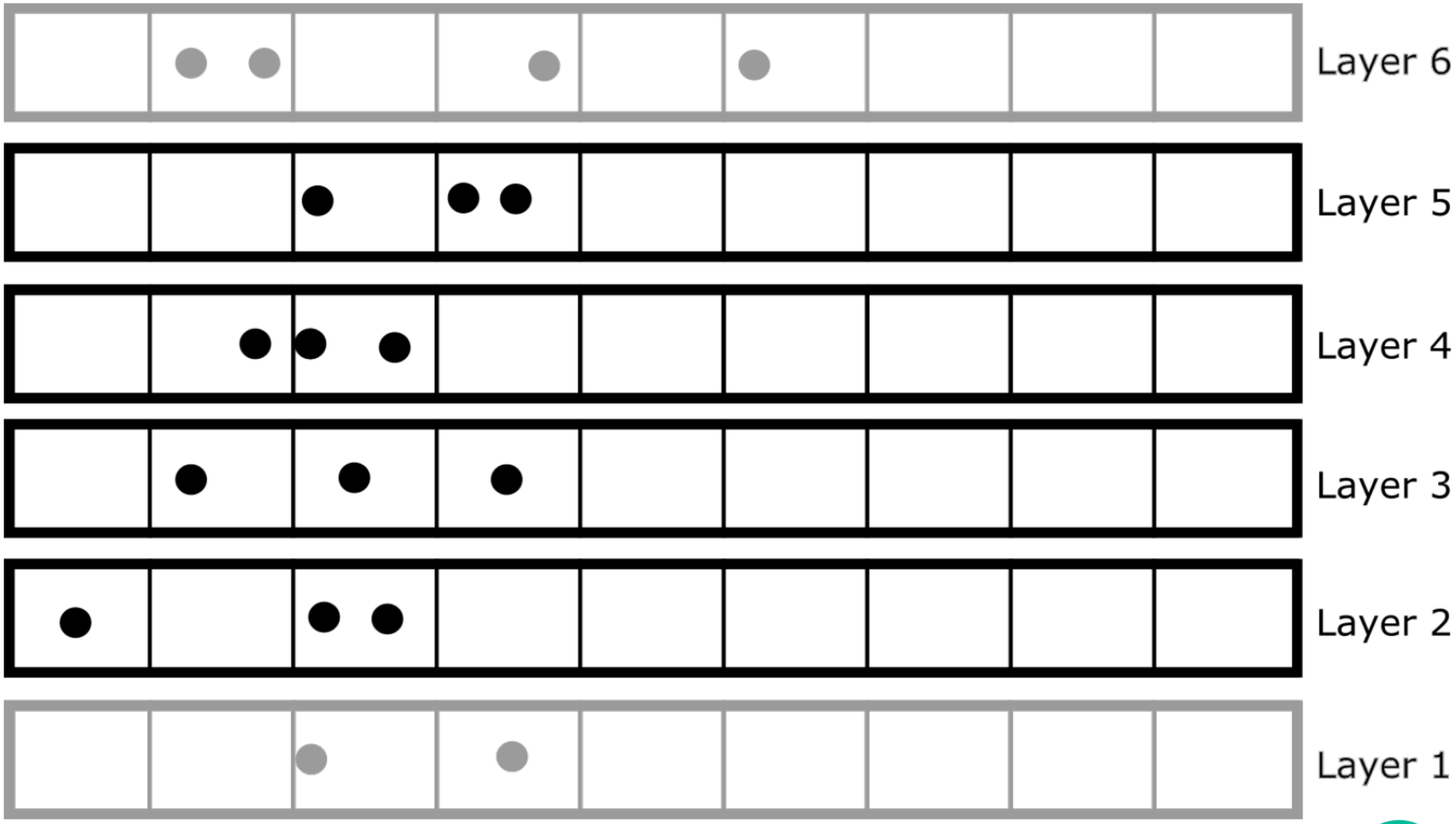


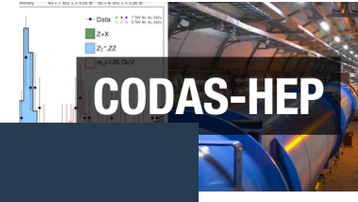
FTK routing network

Input Clusters

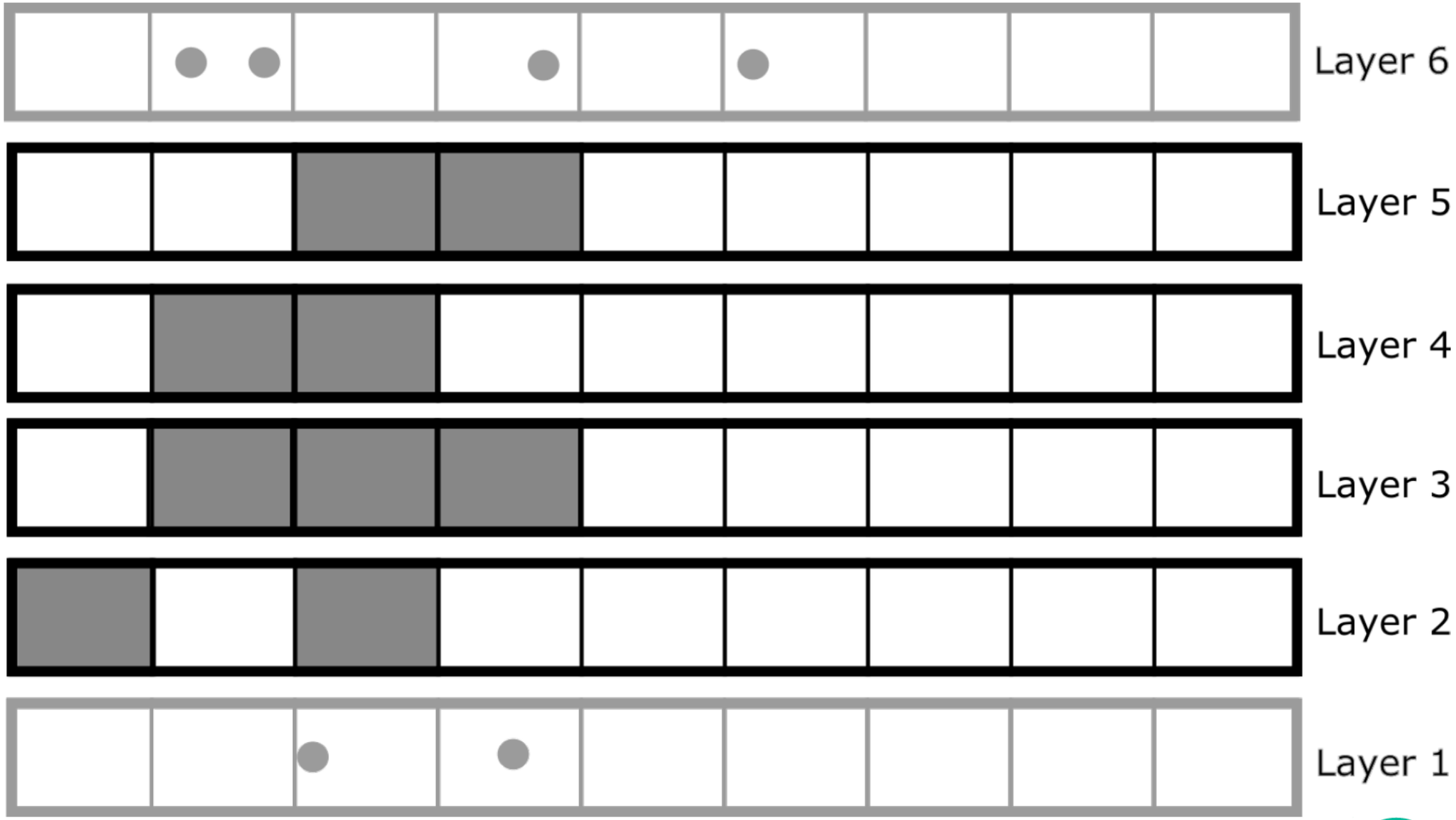


Define superstrips (coarser resolution)

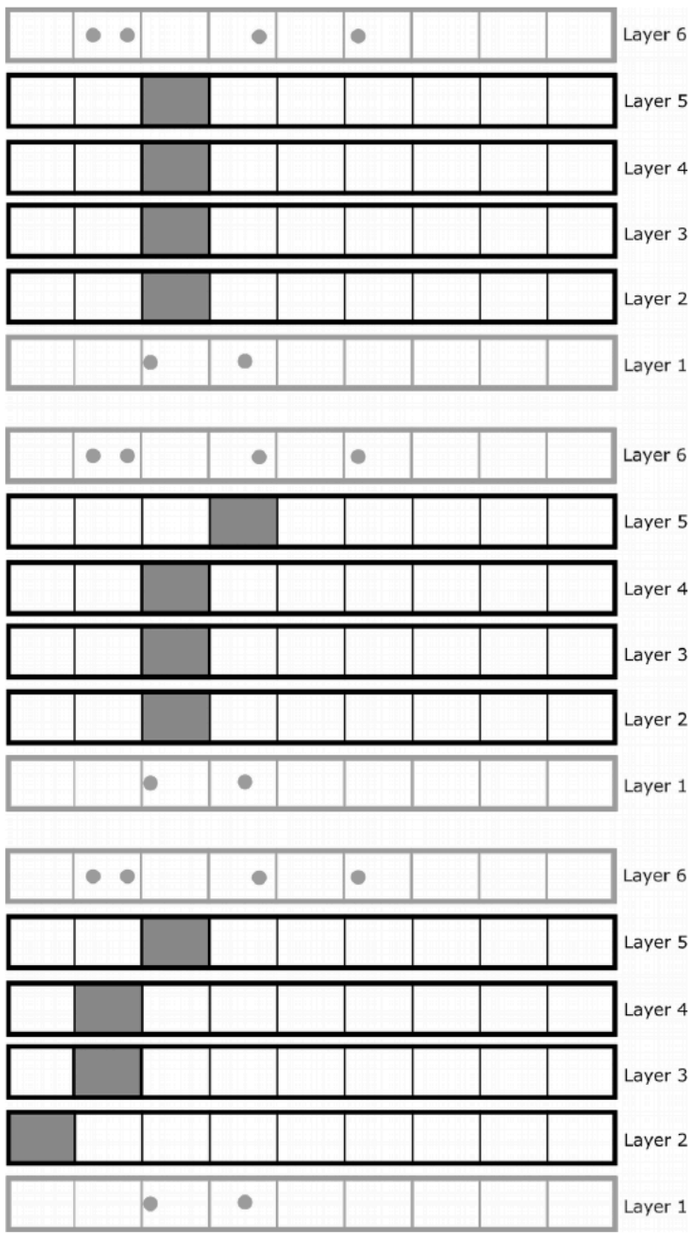




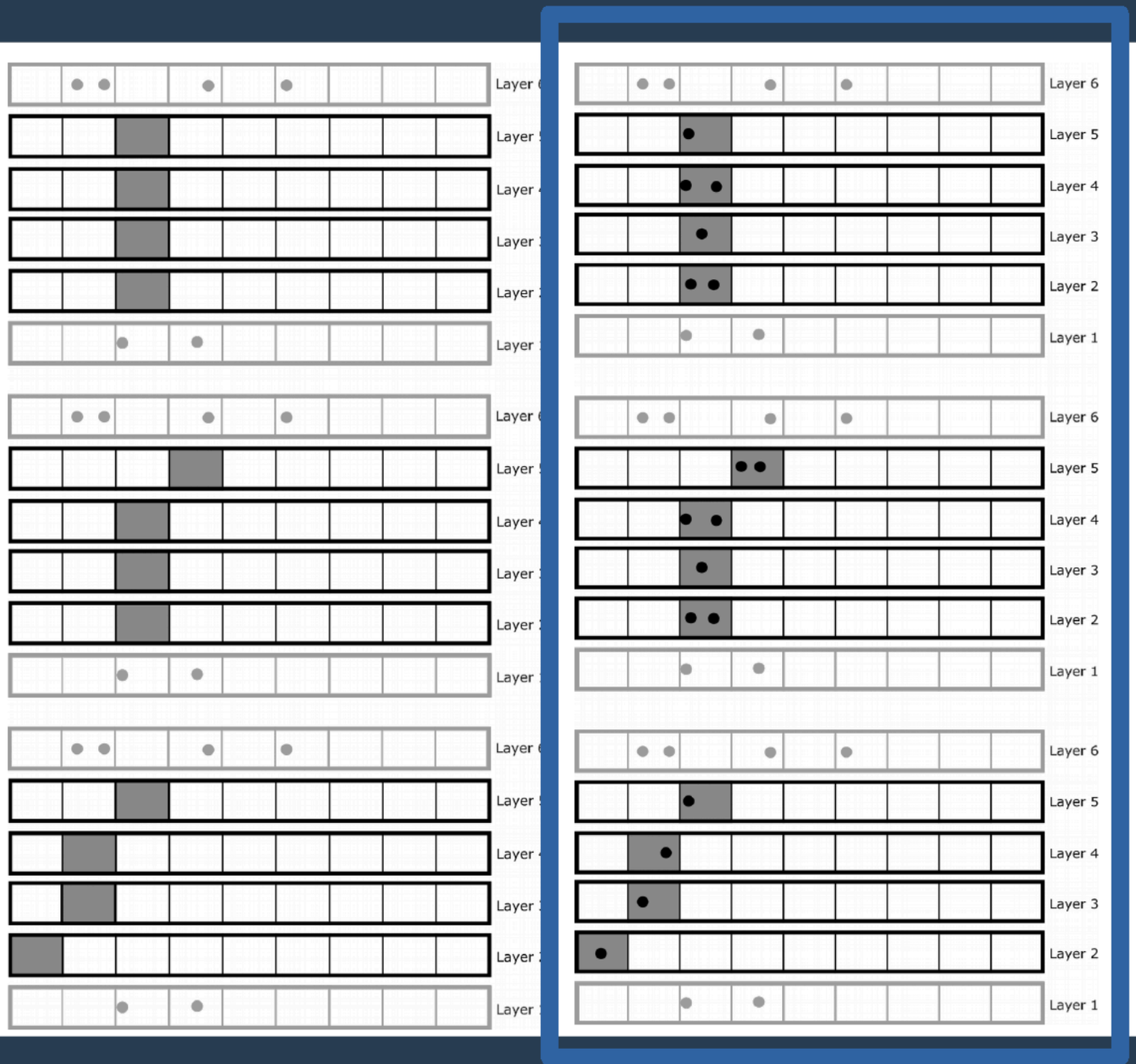
Look up superstrips for each cluster in first-stage layers



Pattern matching for superstrips

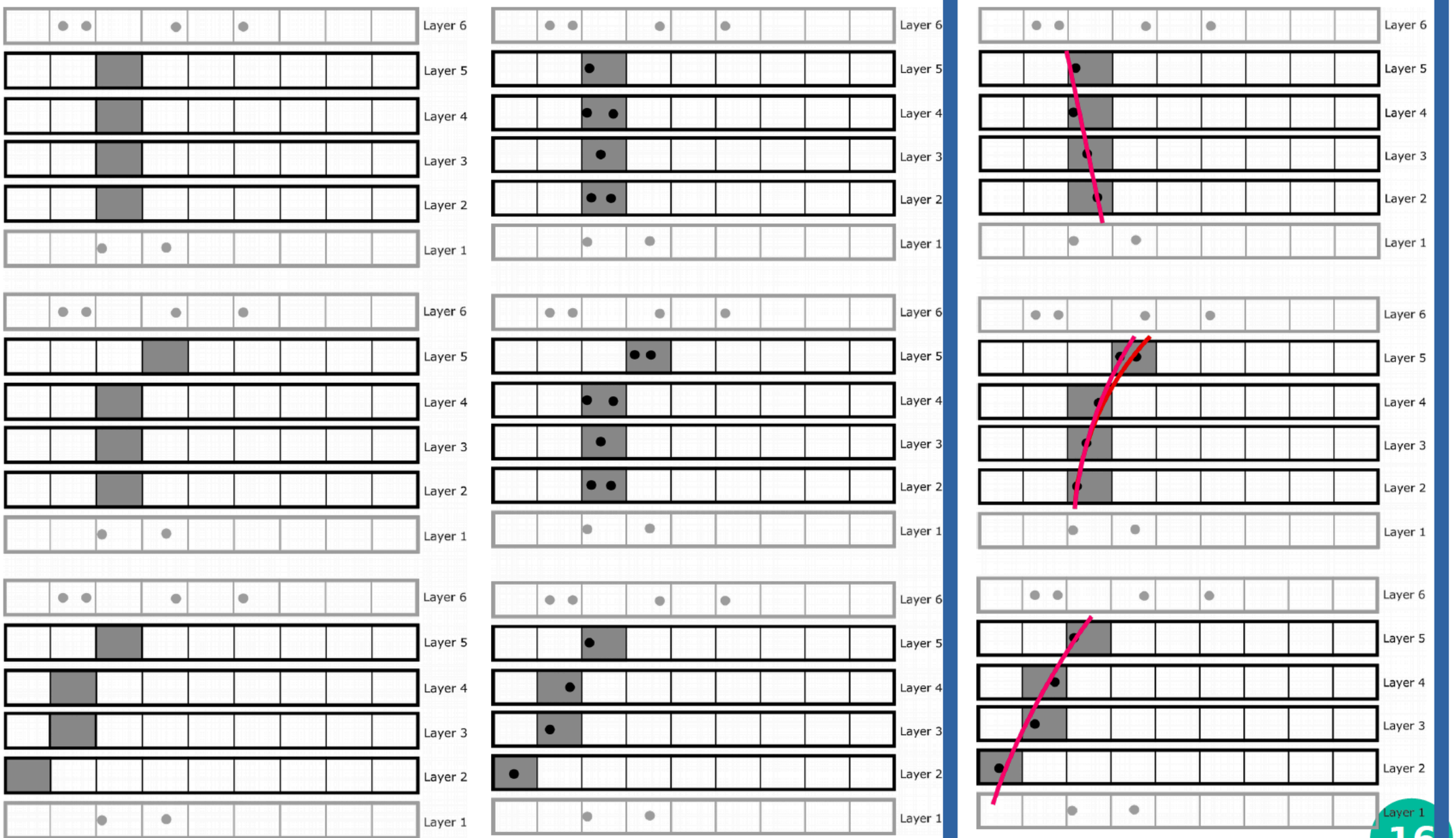


Get full resolution clusters in superstrips

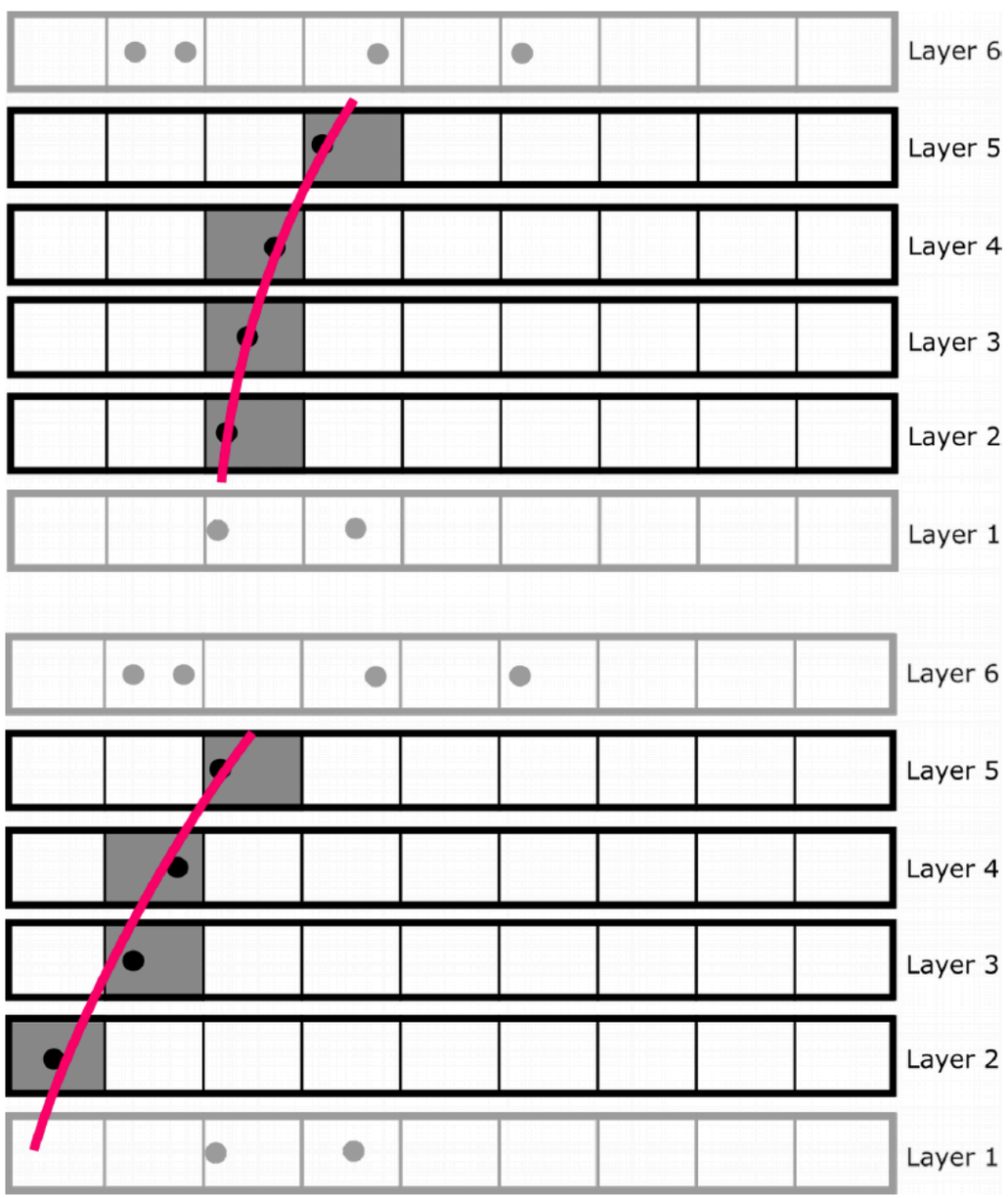


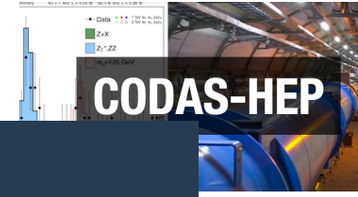
15

Fit all possible combinations of clusters (i.e., compute χ^2)

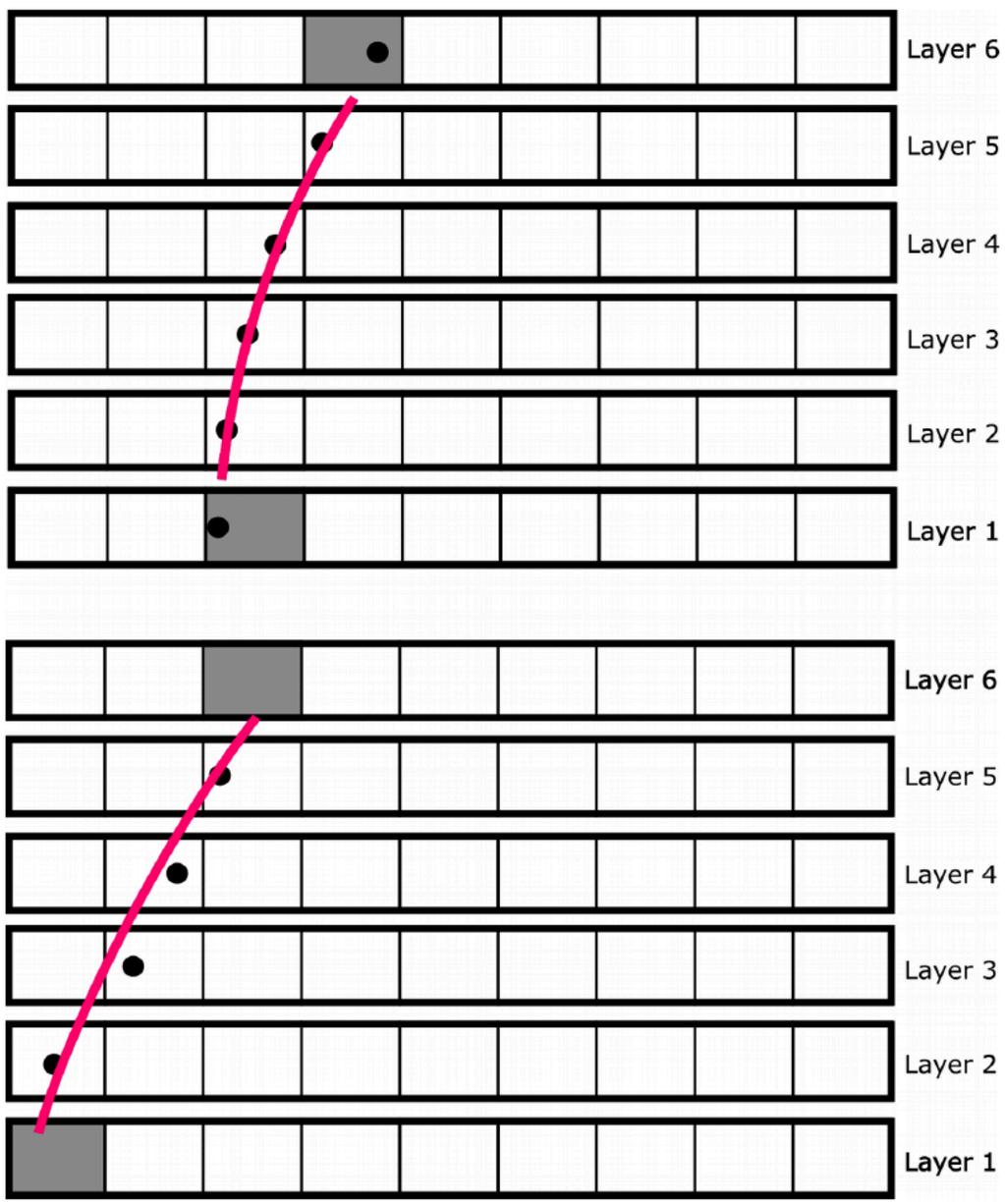


Cut on chi-squared and remove duplicates

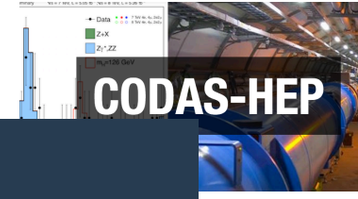




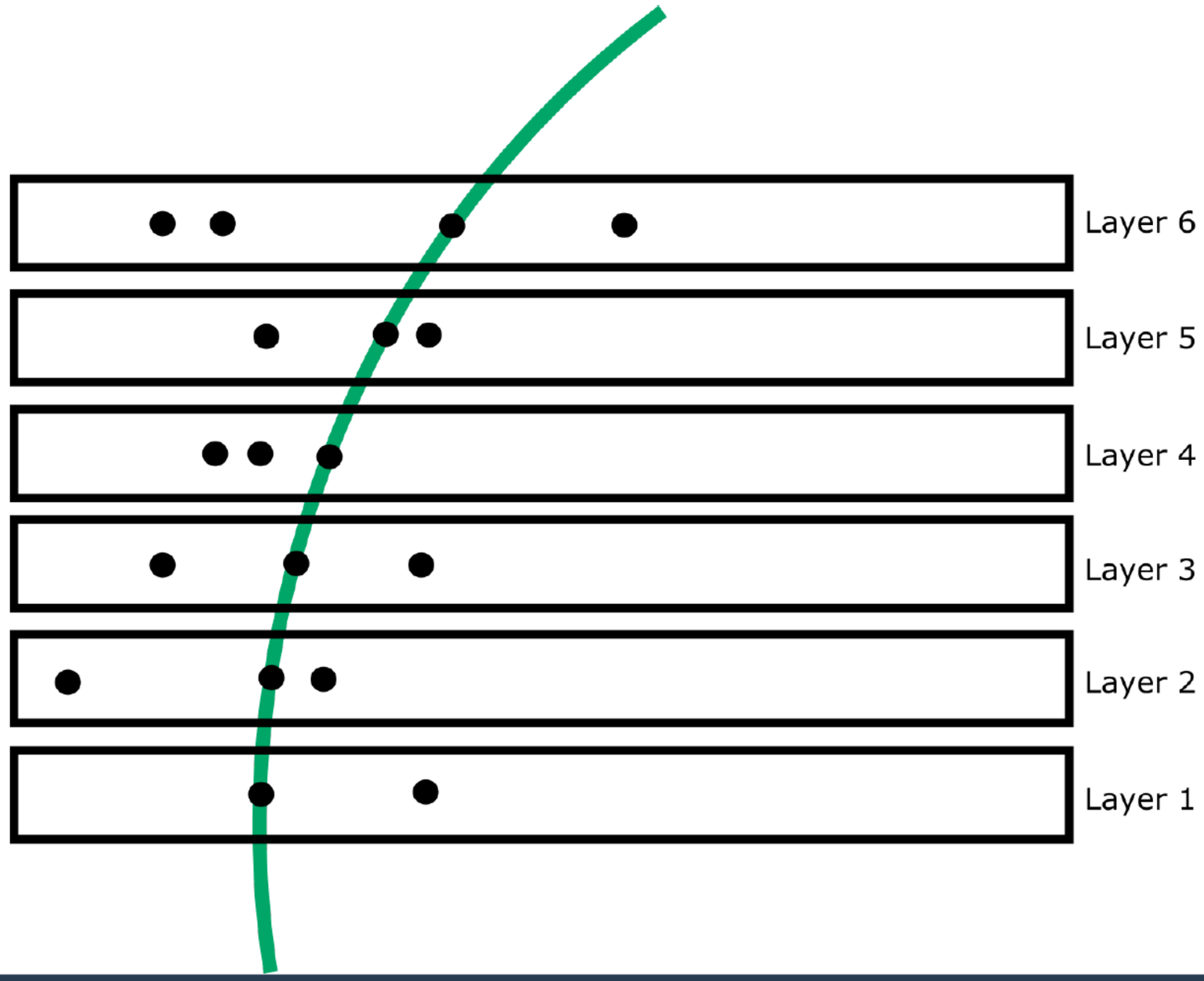
Extrapolate to superstrips in remaining layers

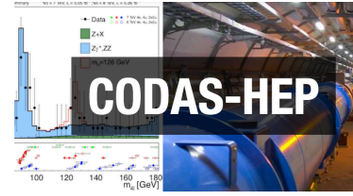


Compute full χ^2, make cuts



Compute track parameters





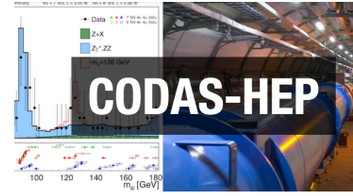
Fast Kalman Filtering: new approaches for the LHCb upgrade

CHEP 2018, Sofia, Bulgaria

Plácido Fernández Declara on behalf of the LHCb collaboration

July 10, 2018

CERN



Vectorized implementation

- Using SIMD, various filter steps are calculated for N tracks, in parallel
- Maximize Vector units usage. (Tracks have different number of hits)

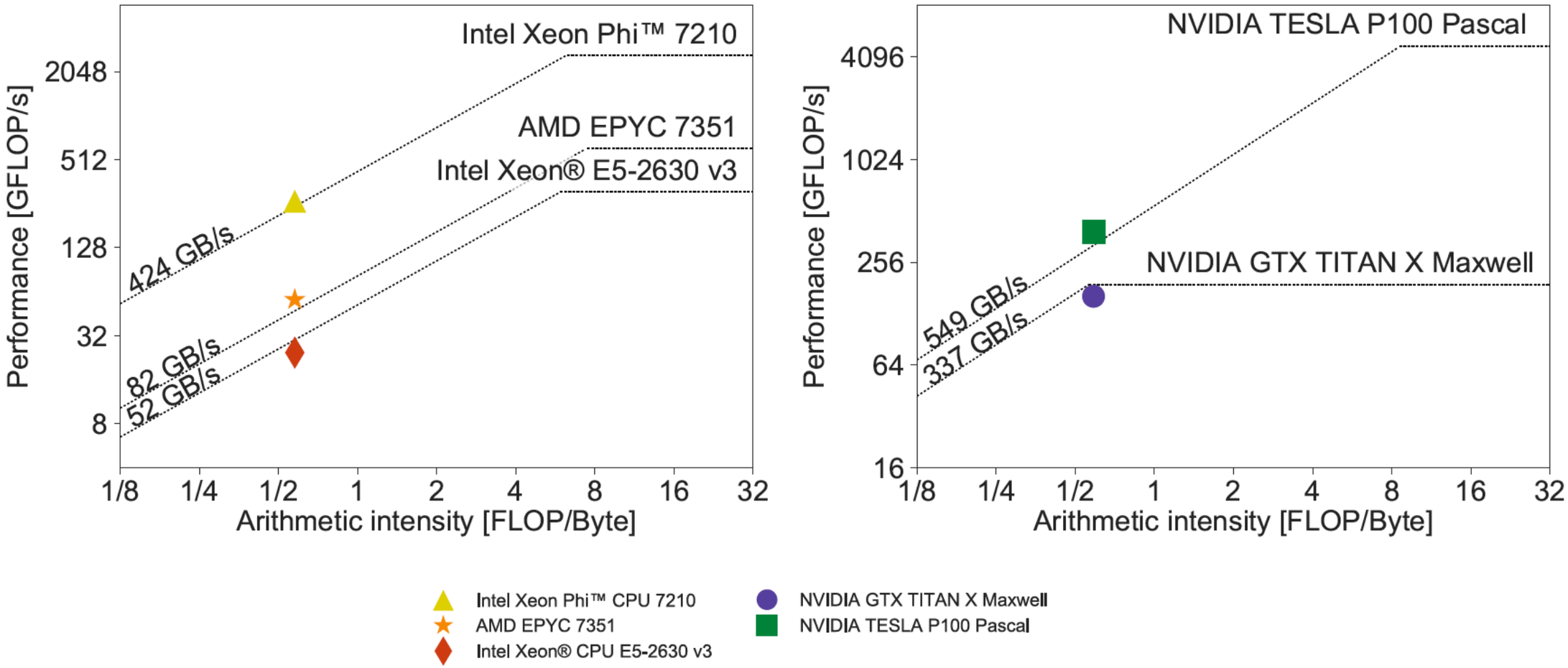
- Scheduler
  - Use of static scheduler for available cores and vector processing units
  - The scheduling applies to all steps (forward, backward and smoother)

- Data layout
  - AOSOA: Array Of Structure Of Array
  - Benefit from both SIMD and cache
  - Adapt to vector width in compile time (cross-architecture)

- Precision can be changed between single and double to test stability of the calculations, and exploit different hardware.

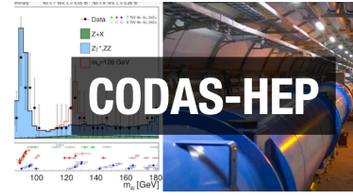
$$
\begin{vmatrix}
x_0 & x_1 & x_2 & x_3 \\
y_0 & y_1 & y_2 & y_3 \\
tx_0 & tx_1 & tx_2 & tx_3 \\
ty_0 & ty_1 & ty_2 & ty_3 \\
\underline{q} & \underline{q} & \underline{q} & \underline{q} \\
p_0 & p_1 & p_2 & p_3 \\
\sigma_{0,0} & \sigma_{1,0} & \sigma_{2,0} & \sigma_{3,0} \\
\vdots & \vdots & \vdots & \vdots \\
\sigma_{0,14} & \sigma_{1,14} & \sigma_{2,14} & \sigma_{3,14} \\
\chi^2{}_0 & \chi^2{}_1 & \chi^2{}_2 & \chi^2{}_3
\end{vmatrix}
$$

Cross-architecture Kalman fit - Roofline



[Cámpora Pérez e.4483]

8



Parametrized Kalman filter

- The slow parts of the Kalman filter are:
  - The extrapolation through the magnetic field
  - The magnetic field and the material look up
- We replace this parts with parametrizations for the extrapolations between layers in the detector.
  - We apply "simple" functions outside the magnet region, and more complex functions inside it.
- Extrapolation from one detector layer to the next is done with functions that map the state at position z to a state at position z'
- The magnetic look up is not necessary since each detector layer has its own tuned parametrized extrapolation.
- Material effects are modelled for every extrapolation function with a noise matrix added to the state covariance matrix. Energy-loss is not directly modelled.