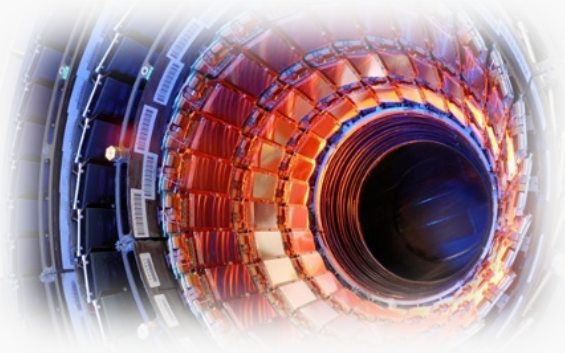
Tracking and Vertexing Short Exercise Introduction



CMS Data Analysis School at the LPC 2023

Brunella D'Anzi, Caleb Smith, Nicola De Filippis, Susan Dittmer

January 9, 2023

Introduction

- Welcome to the Tracking and Vertexing Short exercise (**Twiki** , **Mattermost Channel**)

Motivation

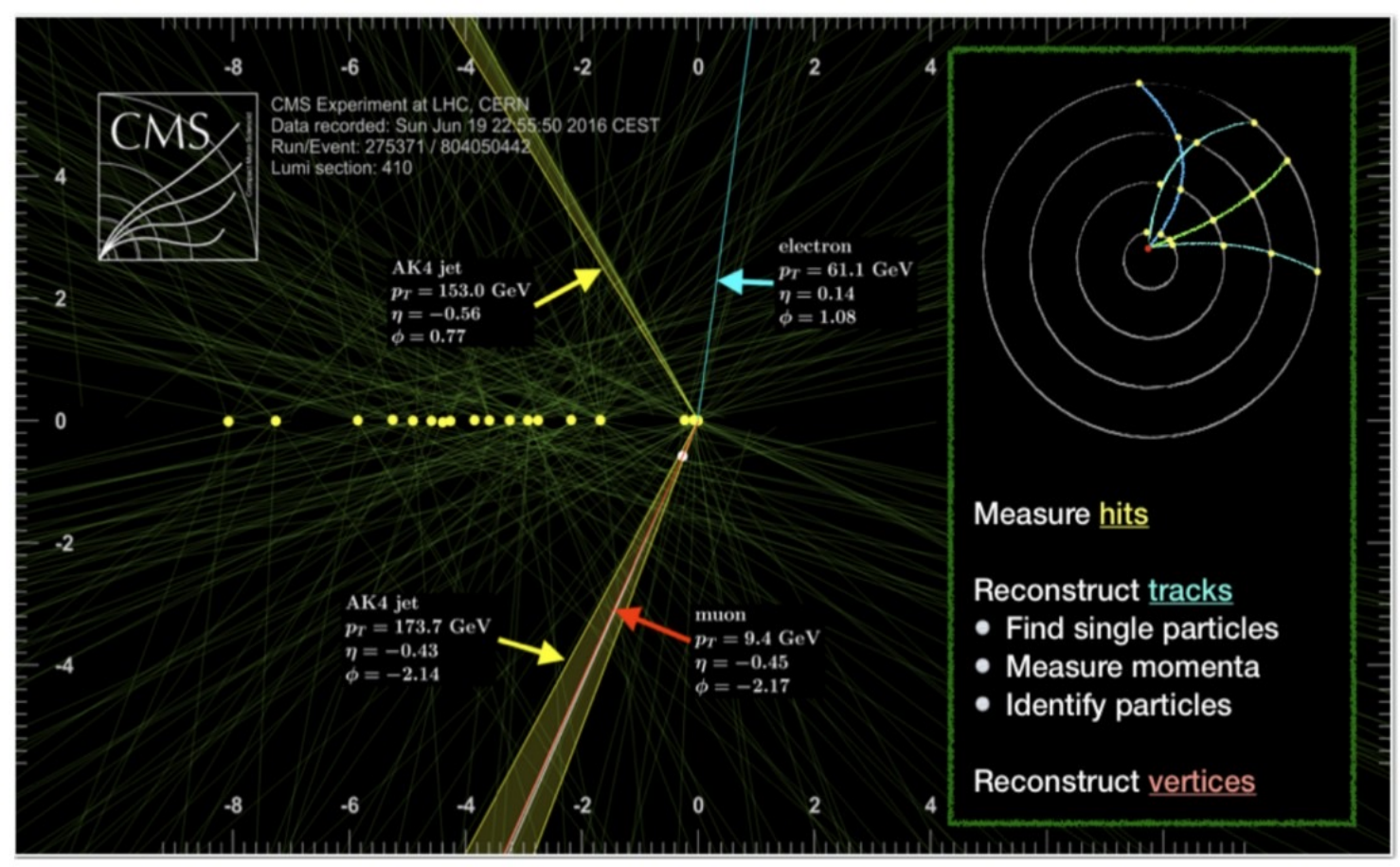
The **reconstruction of charged particle tracks** and **vertices** is fundamental to the reconstruction of every type of physics event in CMS:

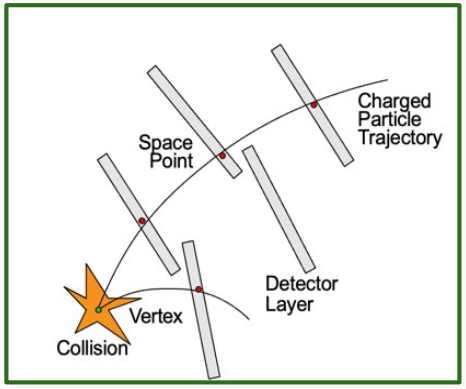
- Directly used in the **reconstruction** of **charged hadrons**, **electrons**, and **muons**
- Needed to distinguish **charged** and **neutral hadrons** and telling **electrons** from **photons**
- Crucial ingredient for **higher level objects** like (b-tagged) jets or taus
- Association of tracks to vertices needed to **distinguish particles from the hard interaction** from **pileup vertices**
- Secondary vertices crucial to **track the decay chains of particles**

Short Exercise Goals

- Get a basic idea of the **CMS silicon tracker** and how tracks and vertices are reconstructed
- Learn how to **access track information** for analysis purposes in different data formats
- Use track information to **understand the particle interactions** in an event
- Use track information to retrieve the **tracking efficiency**

Tracking in CMS

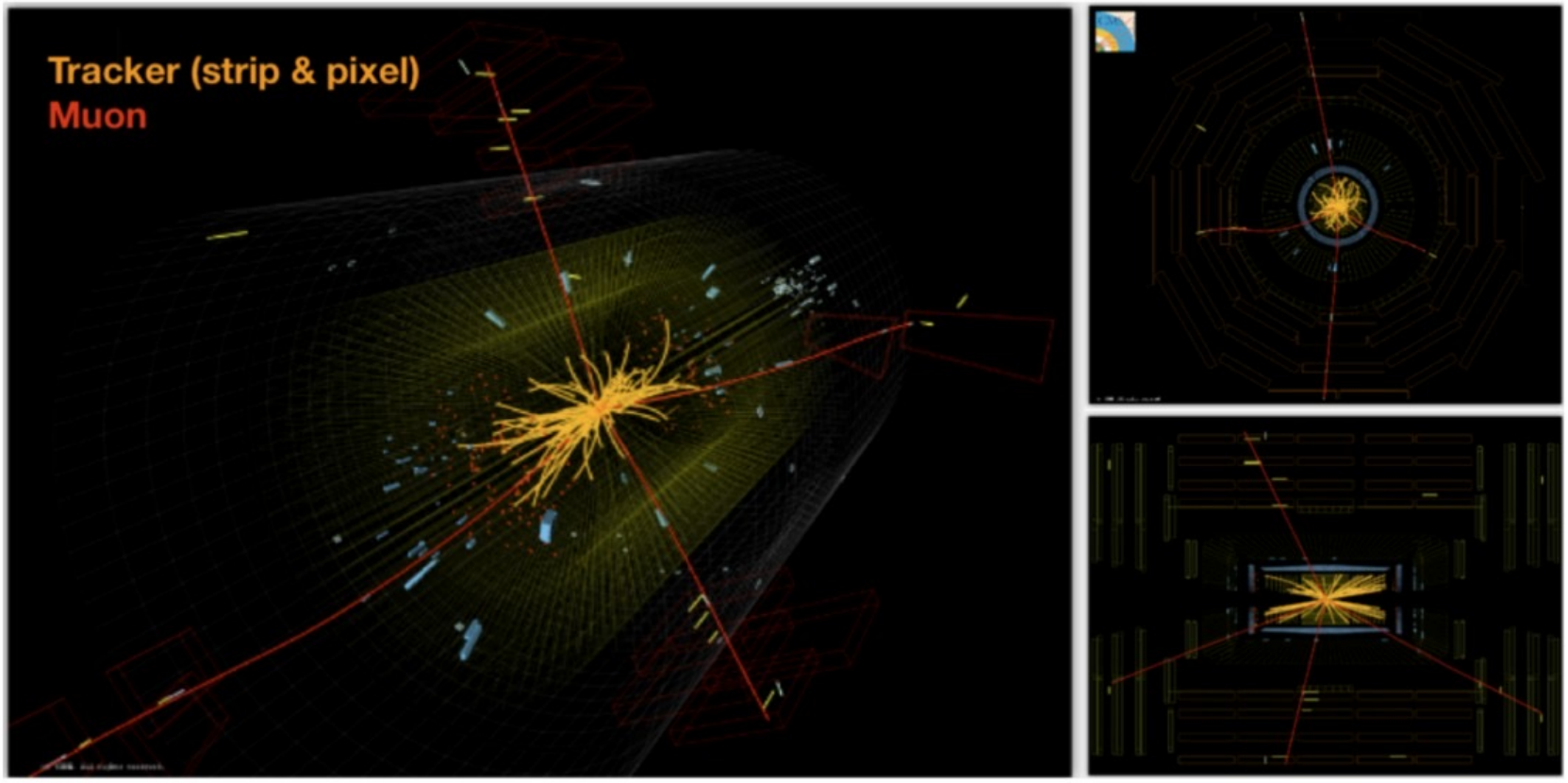




The tracking challenge at the LHC during Run 2

Typically **30 charged particles** within the tracker acceptance per proton-proton collision and **~ 25-40 collisions per event**: **~O(1000) charged particles** per event need to be reconstructed.

Tracking Ingredients



CMS Tracker

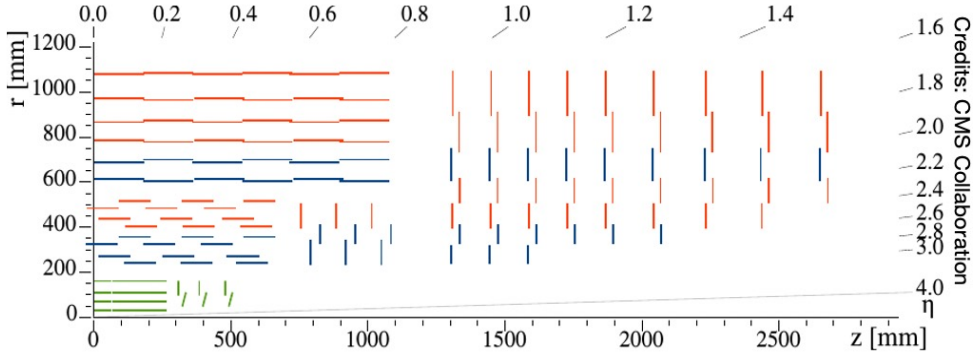
Position information from finely segmented silicon sensors:

- Record the path of charged particles
- Measure momentum from bending radius in the magnetic field
- Reconstruct primary and secondary vertices

performance:
[typically ~15 hits per track]
$\sigma(p_T)/p_T$ ~ 1-2% @ 100 GeV/c
σ(IP) ~ 10-20 µm @ 10-100 GeV/c

Requirements:

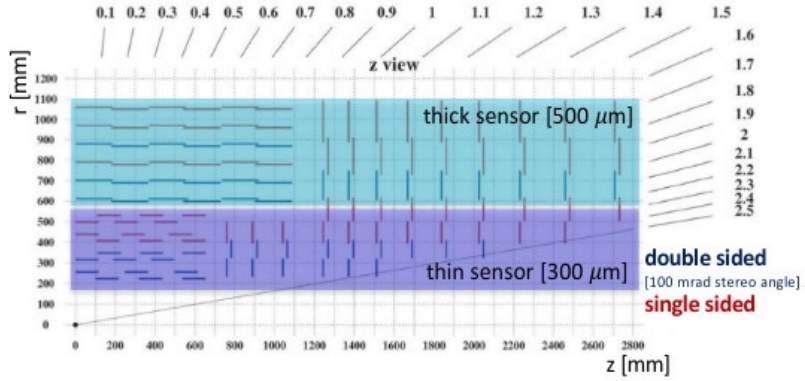
- **High resolution** & **low occupancy**:
  resolve and isolate individual tracks, reconstruct vertices
  Finer granularity is needed closer to the IP [High particle density, small tracking volume]
- **High rate capability**:
  fast charge collection time and read-out electronics to keep up with the expected event rates
- **Low material budget**:
  minimize multiple scattering
- **Radiation hardness**:
  innermost subdetectors
  ⇒ receive highest particle fluence

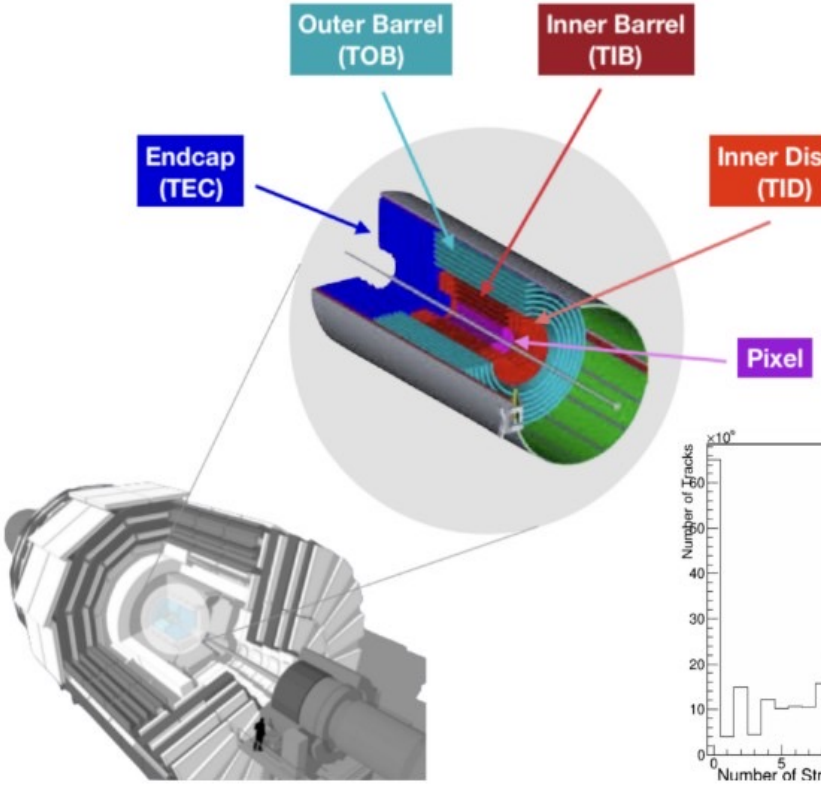


immersed in a 3.8 T magnetic field

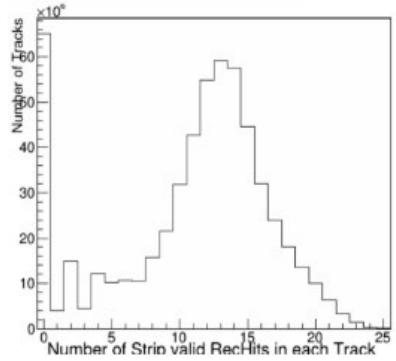
CMS Tracker: Silicon Strips

- O(10) million strips
- O(200) m^2 of sensors
- hit resolution: (10,40)x(230,530) μm
- occupancy: 1-3%
- coverage up to $|\eta| < 2.5$





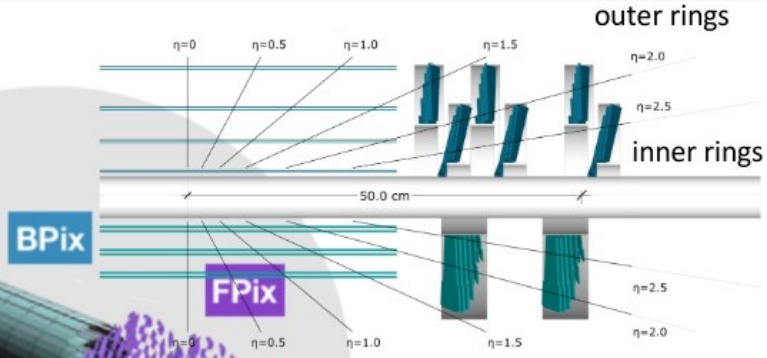
Sub-detectors
- Inner Barrel (**TIB**): 4
- Inner Disks (**TID**): 3 (x 2)
- Outer Barrel (**TOB**): 6
- Endcap (**TEC**): 9 (x 2)

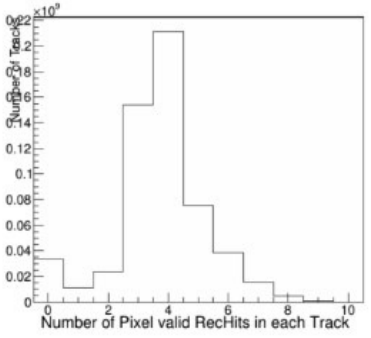


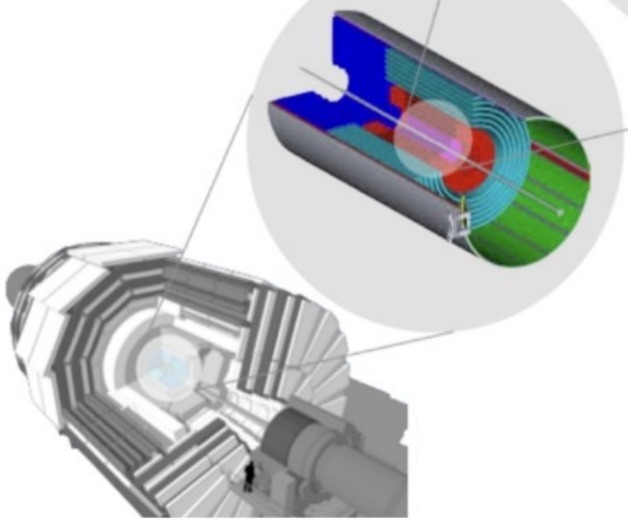
Many layers:
redundancy
12 hits per track on average

CMS Tracker: Silicon Pixels

- 127 million pixels,
- 100x150 μm^2 in size
- hit resolution: 10x(20,40) μm
- occupancy: 10^{-3}
- coverage up to $|\eta| < 2.5$ (even 3.0)
- layer position [cm]:
  - BPix: 2.9, 6.8, 10.9, 16.0
  - FPix: 29.1, 39.6, 51.6
- high segmentation:
  - high quality seeds for offline tracking







Since 2017,
one additional tracking point,
in both barrel and forward regions:
- 4-hit seeds
- lower fake rate!

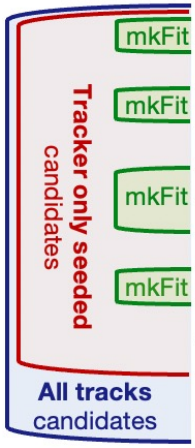
smaller radius of the innermost pixel layer
- closer to the interaction region
- improve tracking and vertexing performance

reduced material budget
- reduce multiple scattering

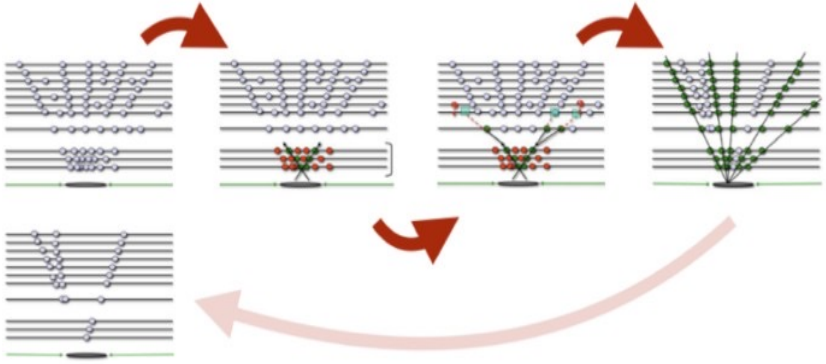
Track reconstruction

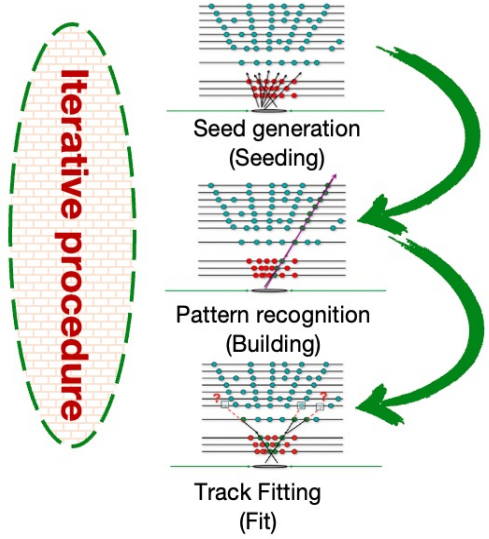
in each iteration, tracks are reconstructed in four steps:

1. **seeding**:
   provides track candidates,
   with an initial estimate of the trajectory parameters and their uncertainties
2. **pattern recognition**:
   track candidates are propagated to find new compatible hits
   track parameters are updated
3. **final fitting**:
   provides the best estimate of the parameters of each smooth trajectory
   after combining all associated hits
4. **selection**:
   sets quality flags based on the fit χ^2 and the track compatibility w/ interaction region
   aims to reject fake tracks



Iteration	Seeding	Target track
Initial	pixel quadruplets	prompt, high p_T
LowPtQuad	pixel quadruplets	prompt, low p_T
HighPtTriplet	pixel triplets	prompt, high p_T recovery
LowPtTriplet	pixel triplets	prompt, low p_T recovery
DetachedQuad	pixel quadruplets	displaced--
DetachedTriplet	pixel triplets	displaced-- recovery
MixedTriplet	pixel+strip triplets	displaced-
PixelLess	inner strip triplets	displaced+
TobTec	outer strip triplets	displaced++
JetCore	pixel pairs in jets	high-p_T jets
Muon inside-out	muon-tagged tracks	muon
Muon outside-in	standalone muon	muon





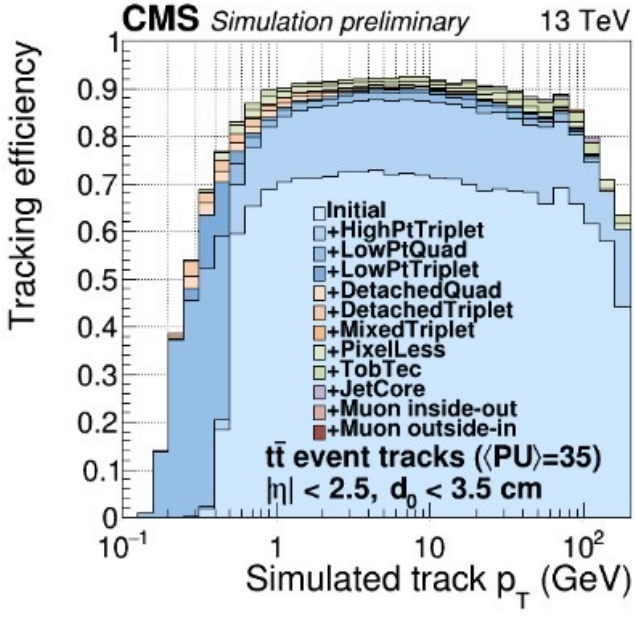
[1] CMS Collaboration, CMS-TRK-11-001, Published in: JINST 9 (2014) 10, P10009.
[2] CMS Collaboration, CMS-DP-2022-018.

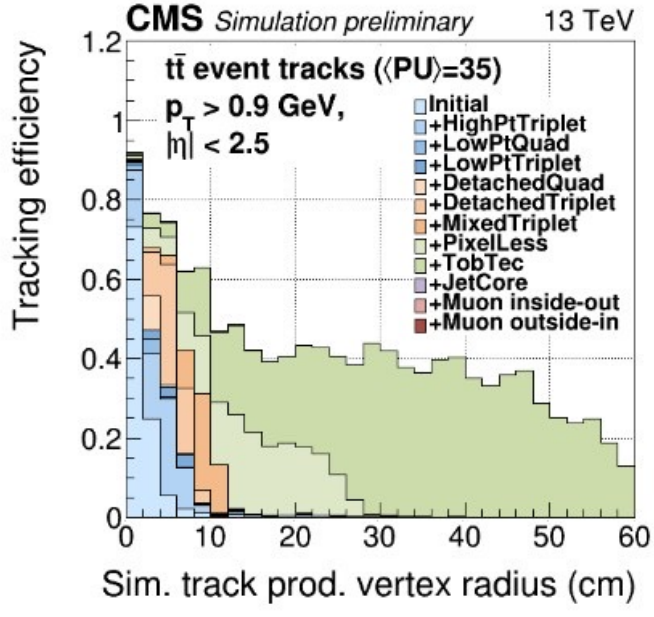
Iterative Tracking

In CMS, tracks reconstruction is an iterative procedure:

high-quality tracks are reconstructed first,
their hits are removed,
and other tracks are reconstructed from the remaining hits

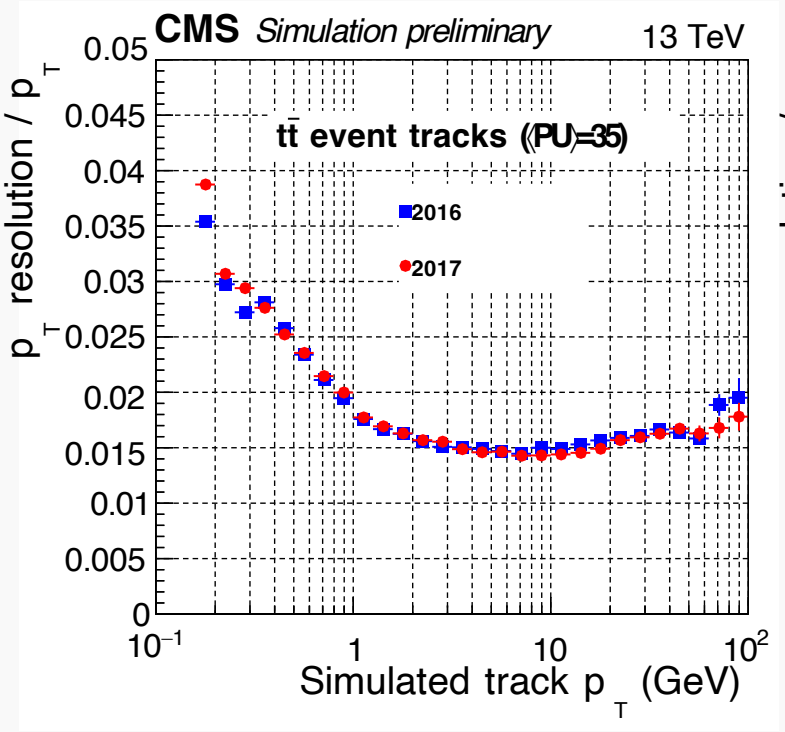
- in the InitialStep, high-p_T quadruplets coming from the beam spot region are used
- subsequent steps use triplets, or improve the acceptance either in p_T or in displacement
- the later steps use seeds with hits from the strip detector to find detached tracks,
- final steps are dedicated to special phase-space
  - highly dense enviroment (i.e. w/in jets)
  - clean enviroment (i.e. muons)

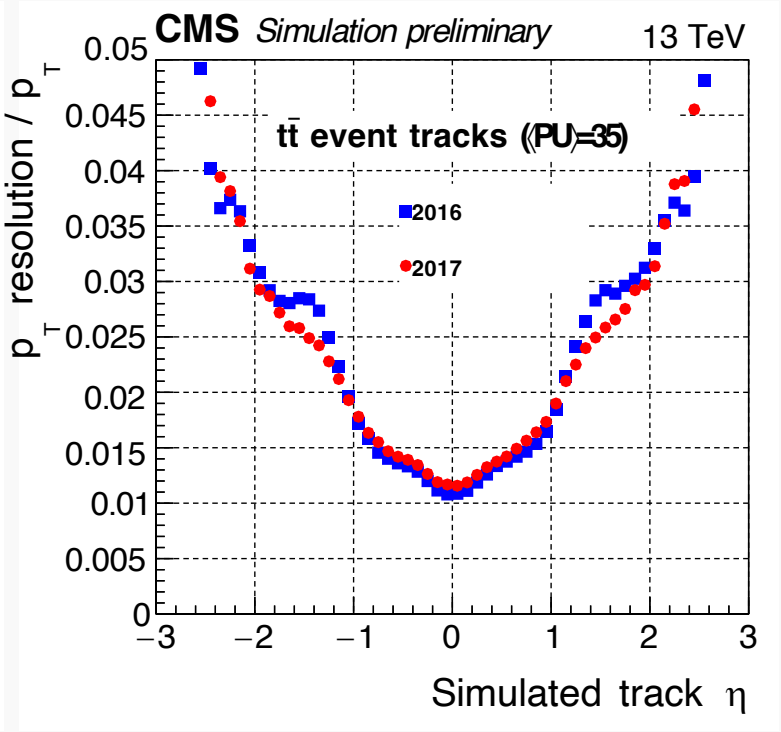




Tracking performance: momentum resolution

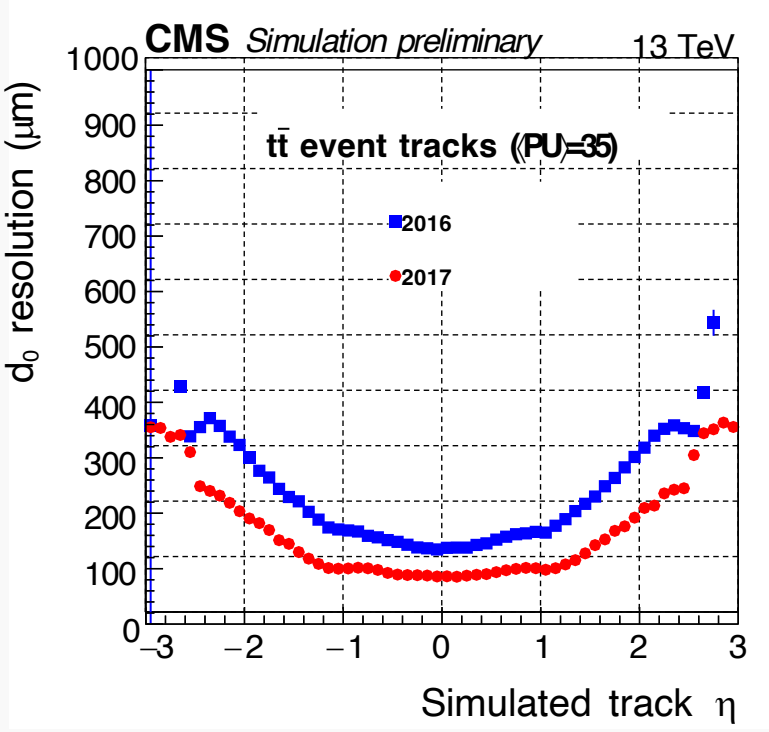
- Relative p_T resolution is 3-4% for very low p_T tracks because of multiple scattering
- Best performance of ≈1.5% for tracks of $O(10)$ GeV
- Resolution degrades at high p_T because tracks are less bend in the magnetic field
- Resolution is best for central tracks

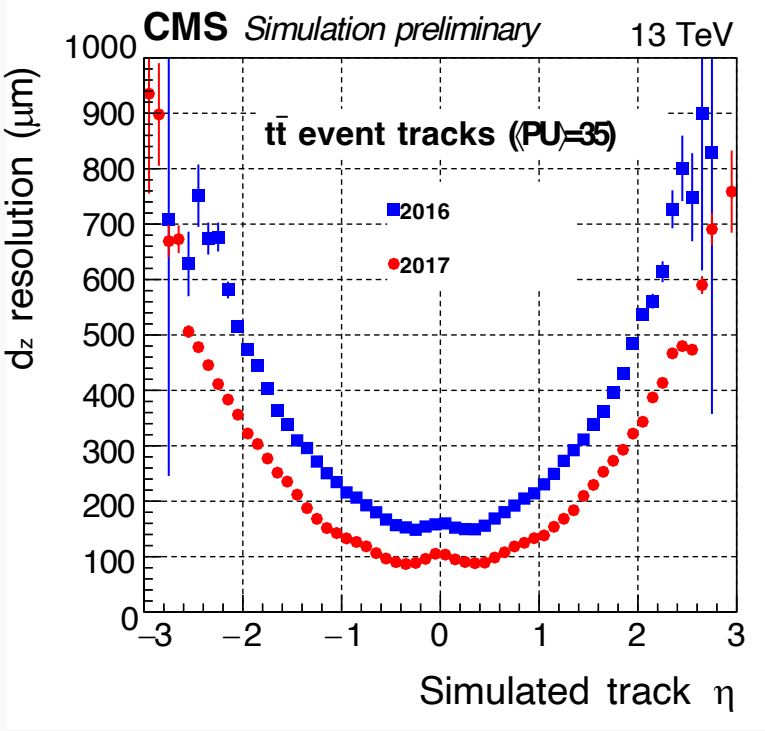




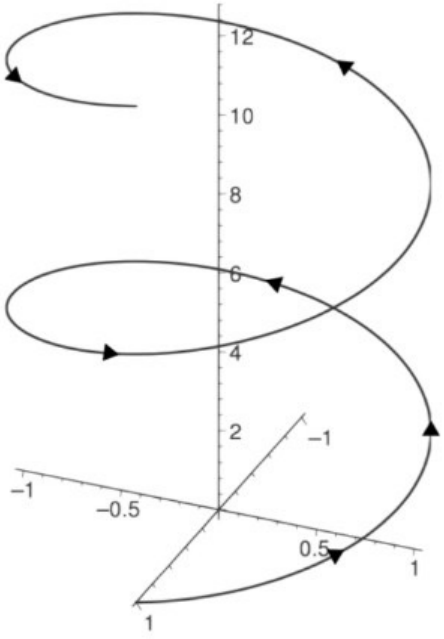
Tracking performance: impact parameter resolution

- With the current detector, the impact parameter resolution is about ≈100 μm for central tracks
- Degrades for forward tracks to up to ≈350 μm for the transverse and >500 μm for forward tracks





Trajetory Parametrization



A helical trajectory can be expressed by 5 parameters,
but the parameterization is not unique.

Given one parameterization, we can always re-express the same trajectory in another parameterization.

In general terms, the five parameters are:

- **signed radius of curvature** (units of cm),
  which is proportional to particle charge
  divided by the **transverse momentum**, $\mathbf{p_T}$, (units of GeV);
- **angle of the trajectory** at a given point on the helix,
  in the plane **transverse to the beamline** (usually called **φ**);
- **angle of the trajectory** at a given point on the helix
  with respect to the beamline (θ, or equivalently $\lambda = \pi/2 - \theta$),
  which is usually expressed
  in terms of pseudorapidity $\boldsymbol{\eta} = -\ln(\tan(\theta/2))$;
- offset or "**impact parameter**" relative to some reference point
  (usually the beamspot or a selected primary vertex),
  in the plane **transverse to the beamline** (usually called $\mathbf{d_{xy}}$);
- **impact parameter** relative to a reference point
  (beamspot or a selected primary vertex),
  along the beamline (usually called $\mathbf{d_z}$)

Vertexing

- The reconstruction and correct identification of the vertex of the hard interaction in an event is of critical importance to correctly selected the final state objects
- We also need to reconstruct as many of the PU vertices as possible to allow for efficient PU suppression with CHS or Puppi

- The vertexing algorithm selects good tracks originating from the interaction region around the beam spot and clusters them according to the z coordinate of their point of closest approach to the center of the beam spot
- When we cluster tracks into vertices, there are two counteracting goals
    - We want to resolve nearby vertices to separate the primary interaction from PU vertices and prevent vertex merging
    - But we also don't want to split genuine vertices in two

Gap clustering algorithms

- A large variety of algorithms are available to cluster tracks into vertices
- Let us first consider a simple gap clustering algorithm used at HLT as *DivisiveVertexFinder'*

- The clustering starts with all tracks sorted according to their z position
- In the beginning, all tracks are considered to be part of the same vertex
- When any two neighboring tracks are have a gap exceeding a given threshold (currently 5 mm), the vertex is split

- Fast, easy algorithm, but not optimal at high PU

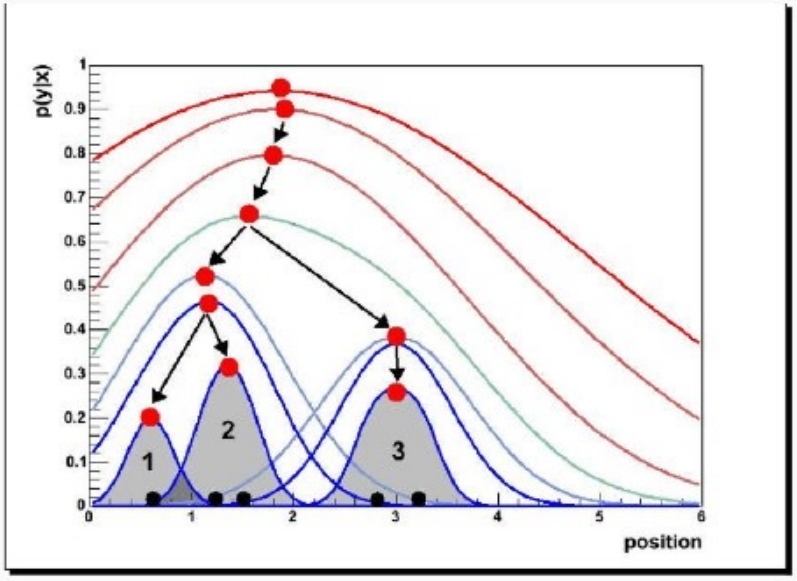
Deterministic Annealing

- Offline vertex clustering uses an algorithm inspired by an analogy to a thermodynamic system:
  - We are trying to find the global minimum for a problem with many degrees of freedom, analogous to a physical system approaching a state of minimal energy through a series of gradual temperature reductions
- We define a free energy of the system:

$$F = -T \sum_{i}^{\text{Ntracks}} p_i \log \sum_{k}^{\text{Nvertices}} \rho_k \exp\left(-\frac{1}{T}\frac{\left(z_i^T - z_k^V\right)^2}{\sigma_i^{z2}}\right)$$

- At high temperature, all tracks are compatible with originating from the same vertex

- As the temperature is lowered, the vertices are split if it falls below a critical temperature on the (temperature dependent) probability of the tracks to be associated to this vertices
- This procedure is repeated until a minimum temperature is reached that balances vertex merging and splitting



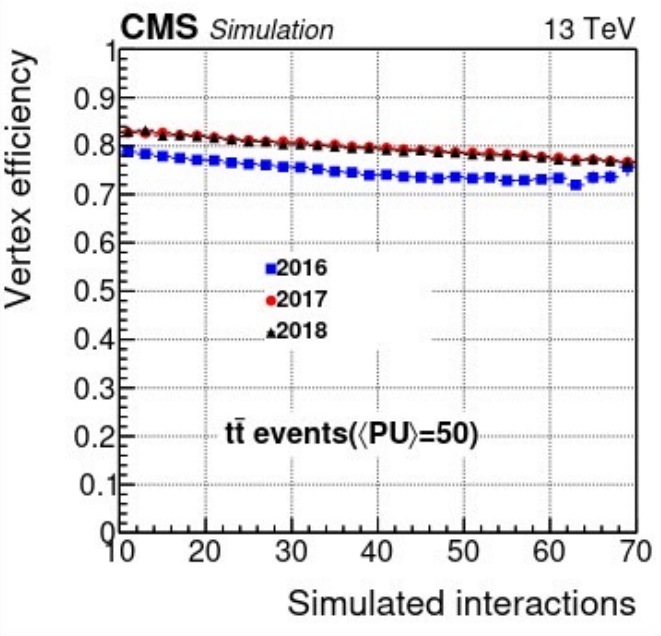
Vertex Fitting

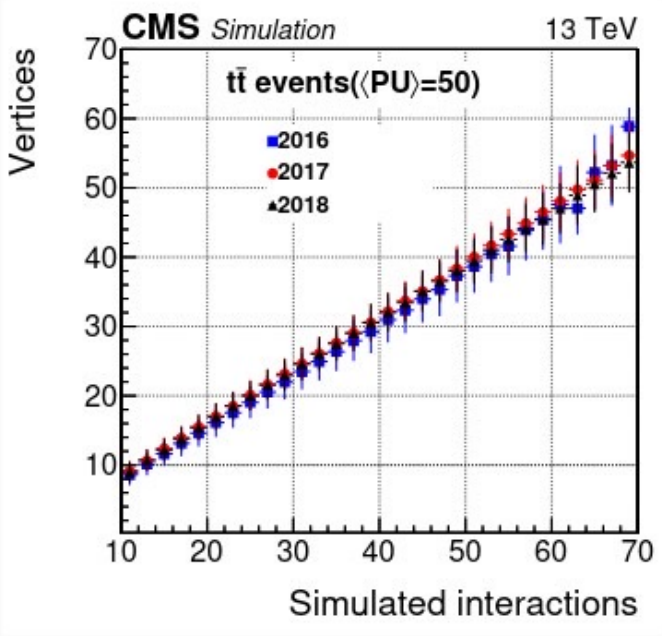
- With the tracks clustered into vertices, the 3D position of the vertex can be fitted
- A good candidate for a vertex fitter is a KalmanFilter which is a least-squares estimator which minimizes the sum of the squared standardized distances of all tracks from the vertex position
- Can be used iteratively, refitting the tracks taking into account the vertex position
- Does not properly handle outlier tracks, leading to bad fits if tracks are included in the vertex that do not belong

- This problem is solved by the *AdapativeVertexFitter* used in CMS offline reconstruction
- Each track is assigned a weight representing the probability that it belongs to the vertex
- This allows to down weight outlier tracks, making the vertex fit much more robust

Vertexing Performance

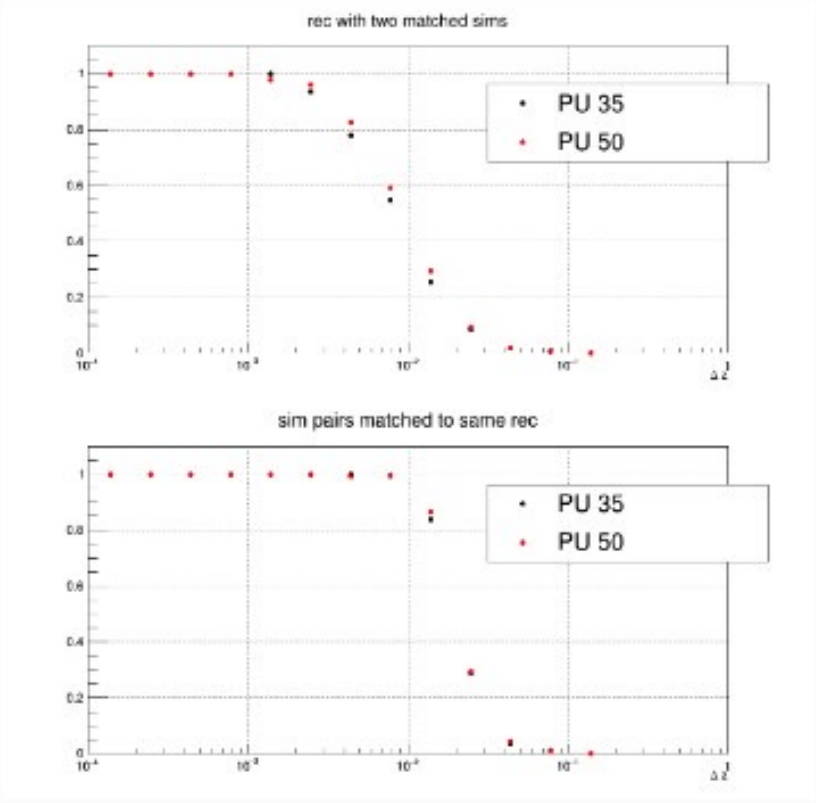
- Vertexing efficiency increased significantly with the Phase-I pixel detector upgrade
- $\approx$ 85% at low PU, slowly decreasing with the number of interactions
- This is the efficiency including PU efficiency. Efficiency for the signal vertex will be much higher
- Still, $\approx$ linear relation between the number of reconstructed vertices and the number of PU interactions





Vertexing Performance

- Loss of efficiency at high PU attributable to vertex merging
- Merging starts for distances closer than 0.3 mm
- Vertices closer than 100 μm can't be separated



Tracks in CMS data formats

- CMS provides several different data formats for analysis
- In general, we want to reduce as much information as possible to save storage space and speed up event processing by the analysts
- However, this means significant compromises in the accessibility of “low-level” information, such as individual tracks
- Currently the analysis data formats are:

  - **AOD**: Save all reconstructed objects and drops only low-level detector information. All tracks passing loose selection requirements are saved in the **generalTracks** collection. Note that the use of this format is to be avoided if at all possible and it can be tricky to even access it as few copies are stored to save space
  - **MiniAOD**: Stripped down version of AOD, keeping only high level objects necessary for most analyses. No single collection of all tracks is kept. Tracks associated with PF candidates are available through the **packedPFCandidate** object. Tracks not associated with PF candidates are stored in the **lostTracks** collection if they have $p_T > 0.95\,\text{GeV}$
  - **NanoAOD**: Store only the most relevant information for analysis in flat tuples. Track information ist stored only for some isolated tracks

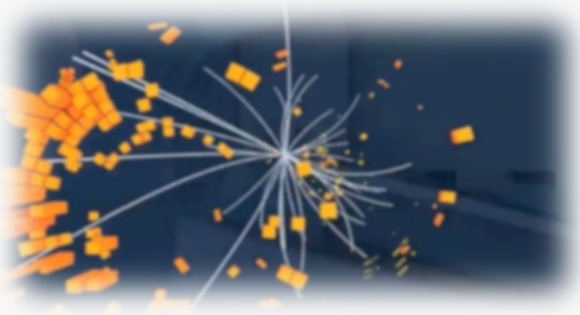
Vertices in CMS data formats

A similarly for vertices:

- **AOD**: The **offlinePrimaryVertices** collection contains all reconstructed vertices
- **MiniAOD**: To save space, the **offlineSlimmedPrimaryVertices** drop the references from the vertices to the associated tracks and the numerical precision of vertex parameters is reduced
- **NanoAOD**: basic information about the main primary vertex is available, as well as number and z position of additional PVs. Some information about secondary vertices is also available

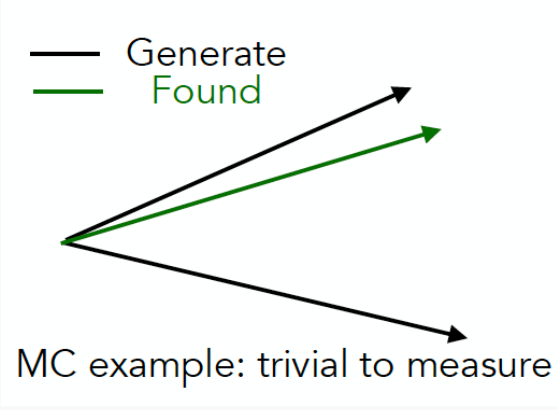
Physics object efficiency

- One of the **most crucial measurements** in every analysis
- **Incorrect measurements** of physics object efficiency may lead:
  - Incorrect prediction of how many events we expect
  - Incorrect SF/ uncertainties and finally limits
  - Biased results in case of high precision measurement
- **Reconstruction software** feedback



What is Efficiency?

- How probable is to reconstruct something
- Defined as the ratio: $\frac{N(reco/ID/trigger/...)}{N(produced)}$
- In **Monte-Carlo (MC)**: N(produced) ≡ N(Generated)
- For many aspects on the analysis we rely on MC simulation but **simulations are very often "optimistic"** and one need to validate the extracted quantity with data.
  - Usage of the data-driven **Tag and Probe (T&P) technique**



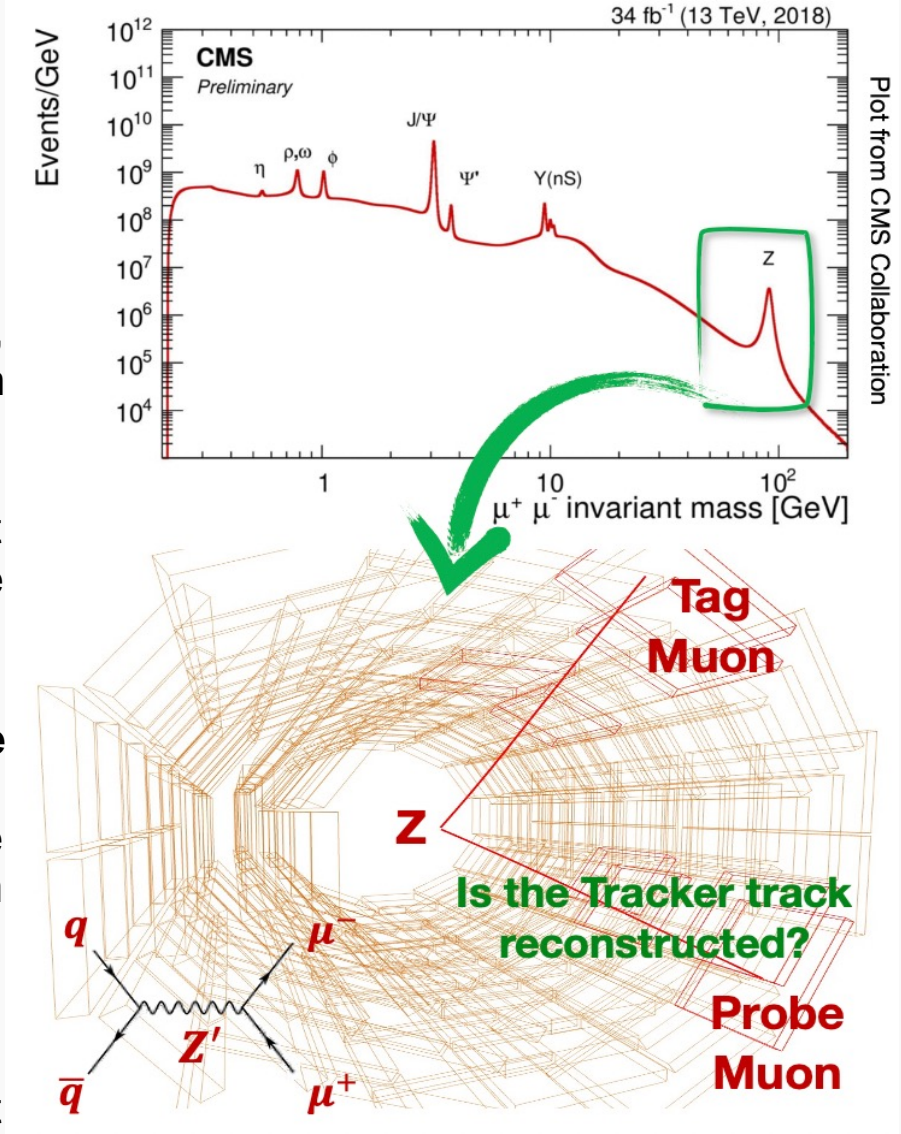
The Tag and Probe technique

The objective

- To identify similar signal in data and in MC and compare the results to **validate MC**.

Strategy for Tracking Reconstruction efficiency

- Use **known resonances** decaying to two muons, produced copiously, attributes measured with high precision.
- Require a muon identified by stringent requirements and reconstructed using both the **muon chambers** and the **tracker** (**Tag**).
- Take all muons reconstructed using **only signals from the muon system** and identified by looser requirements that, combined with Tag, defines a good candidate passing cuts that clean the sample from random combinations (like invariant mass, dz etc). Successful muons are known as **Probes.**
- Check if probe muons can be **matched** to at least **one Tracker track** in a cone around the probe muon (**Passing Probes**).



Background rejection in T&P method

Efficiency computation

As the **selected events** may not come from the resonance, there could be a **bias** in the efficiency measurement.

- To avoid this, a **simultaneous fit** to the signal **(Z peak)** and the background is performed (for both **passing** and **failing** probes, **data** and **MC**).

- Then the efficiency is computed as:

$$\epsilon = \frac{passing\ probes}{passing\ probes + failing\ probes}$$

- This process will have to be repeated for each variable bin if the measurement efficiency as function of one kinematic variable.

efficiency = (99.32 ± 0.06) %

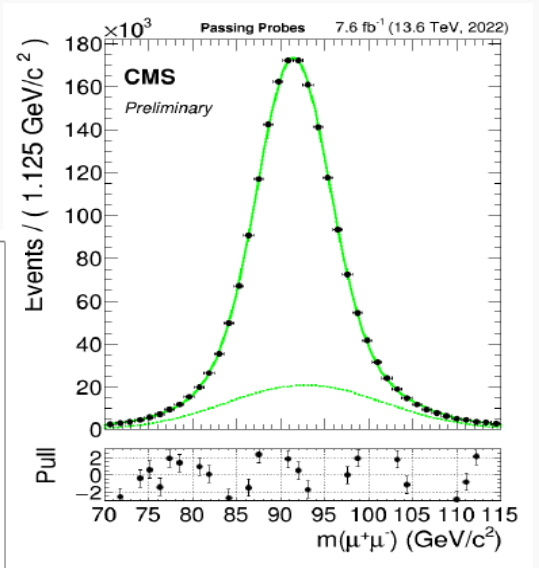
parameters

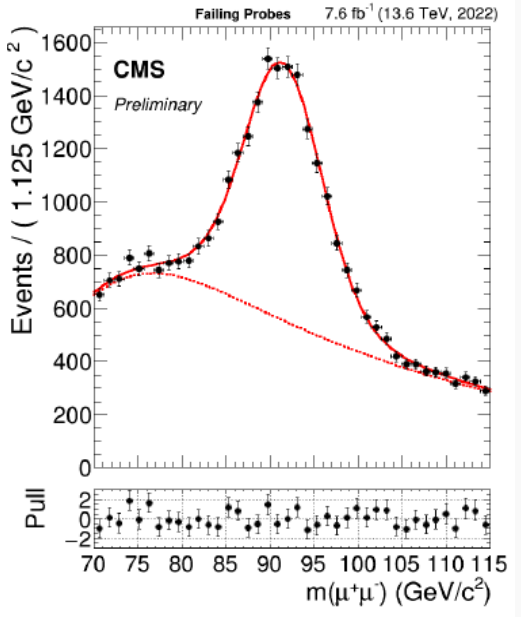
Passing Probes

- αP = (13945.05 ± 7.12) × 10^{-2}
- βP = (8.00 ± 0.00) × 10^{-2}
- γP = (61.74 ± 0.16) × 10^{-2}
- μP = (9138.22 ± 0.62) × 10^{-2}
- nBkgP = (412.36 ± 11.32) × 10^{3}
- nSigP = (1533.45 ± 9.82) × 10^{3}
- σP = (3704.67 ± 7.08) × 10^{-3}
- ΓP = (228.02 ± 3.70) × 10^{-2}

Failing Probes

- αF = (6688.51 ± 126.60) × 10^{-2}
- βF = (8.00 ± 0.81) × 10^{-2}
- γF = (2.83 ± 0.15) × 10^{-2}
- μF = (9139.77 ± 10.86) × 10^{-2}
- nBkgF = (21.36 ± 0.98) × 10^{3}
- nSigF = (10.49 ± 0.98) × 10^{3}
- σF = (3991.92 ± 489.43) × 10^{-3}
- ΓF = (253.40 ± 186.70) × 10^{-2}





Further Information

Technical information

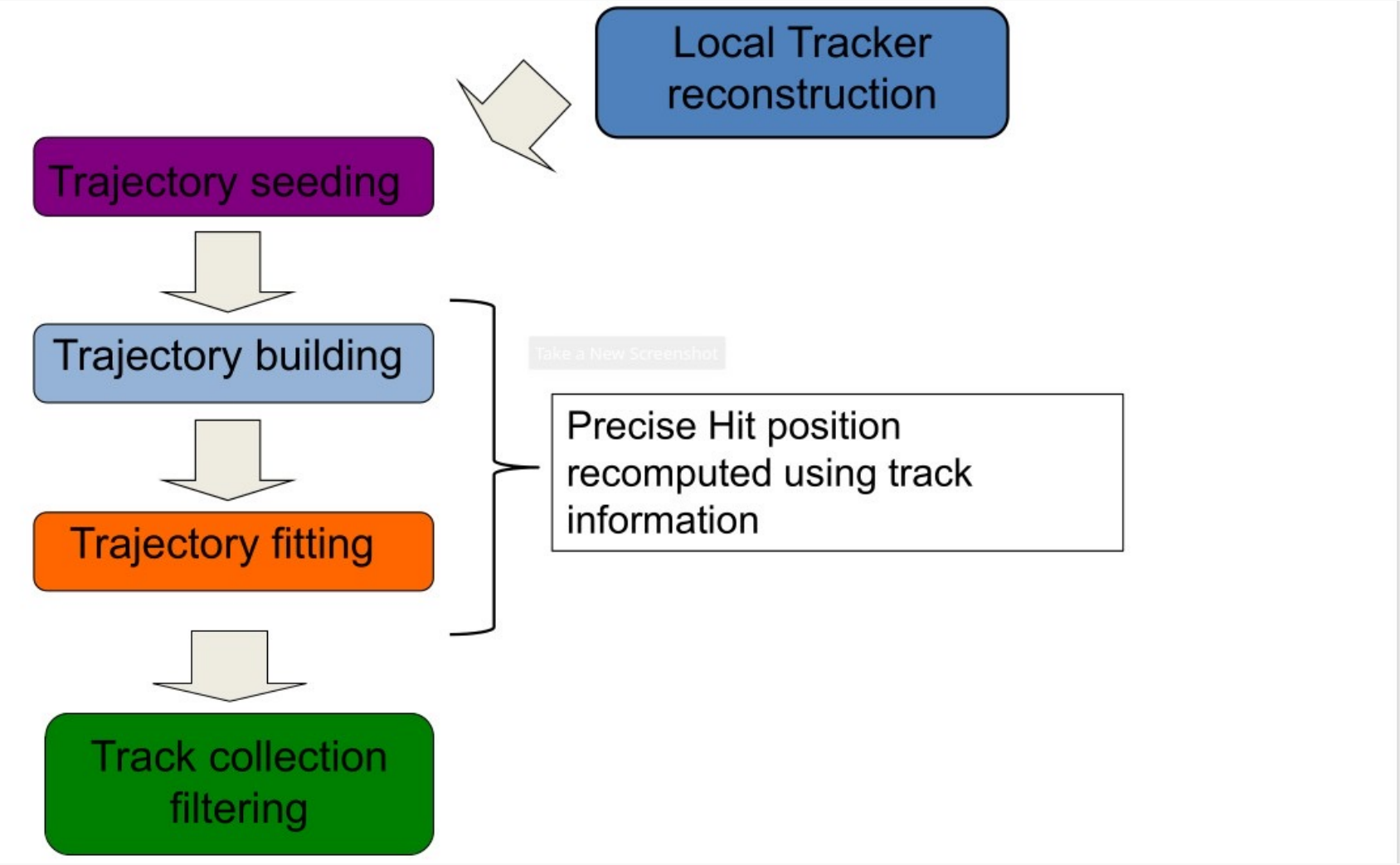
- Run 1 tracking performance paper: arxiv link
- TrackReco Twiki: Not always up to date, but provides technical information about most CMSSW modules used in the reconstruction
- VertexReco Twiki: Same for vertexing
- Tracking Trainig Day in 2019: Agenda

Contact information

- Tracking POG Twiki
- Hypernews: hn-cms-tracking@cern.ch

Backup

Modules for track reconstruction

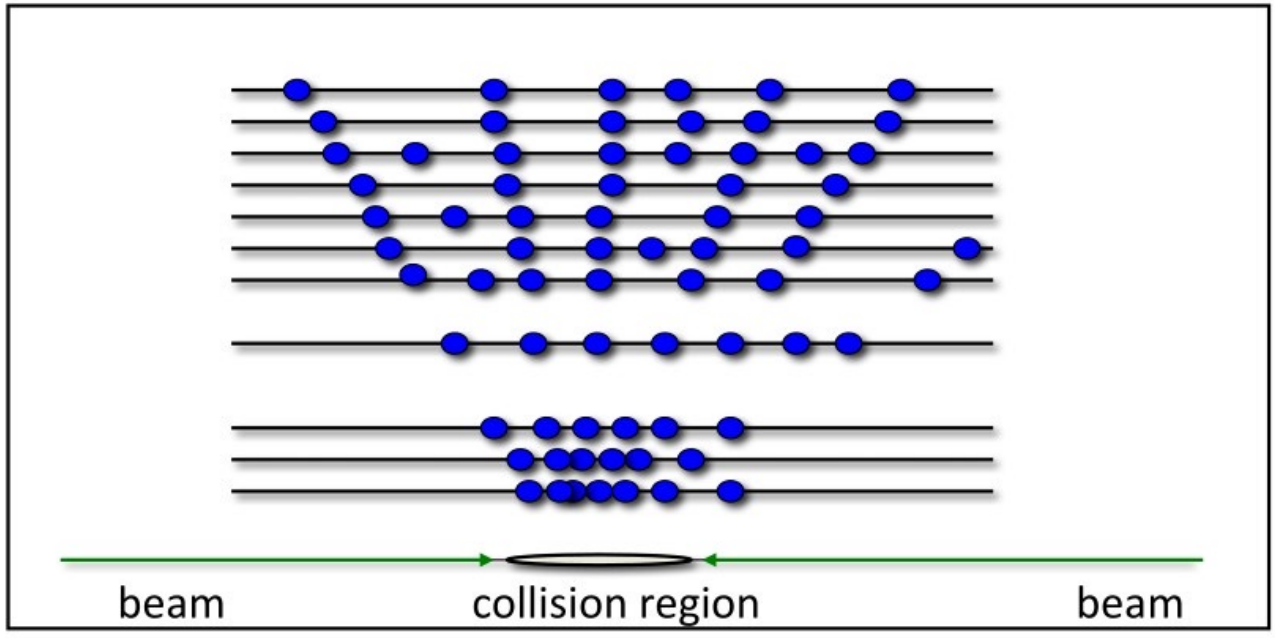


Modules for track reconstruction (II)

Local Tracker reconstruction

- Pixel and silicon-strip signals are clustered into "hits".
- "Coarse" position and corresponding error matrix of each hit are evaluated.

Output are collections of pixel and strip `TrackingRecHit` (three different C++ types)



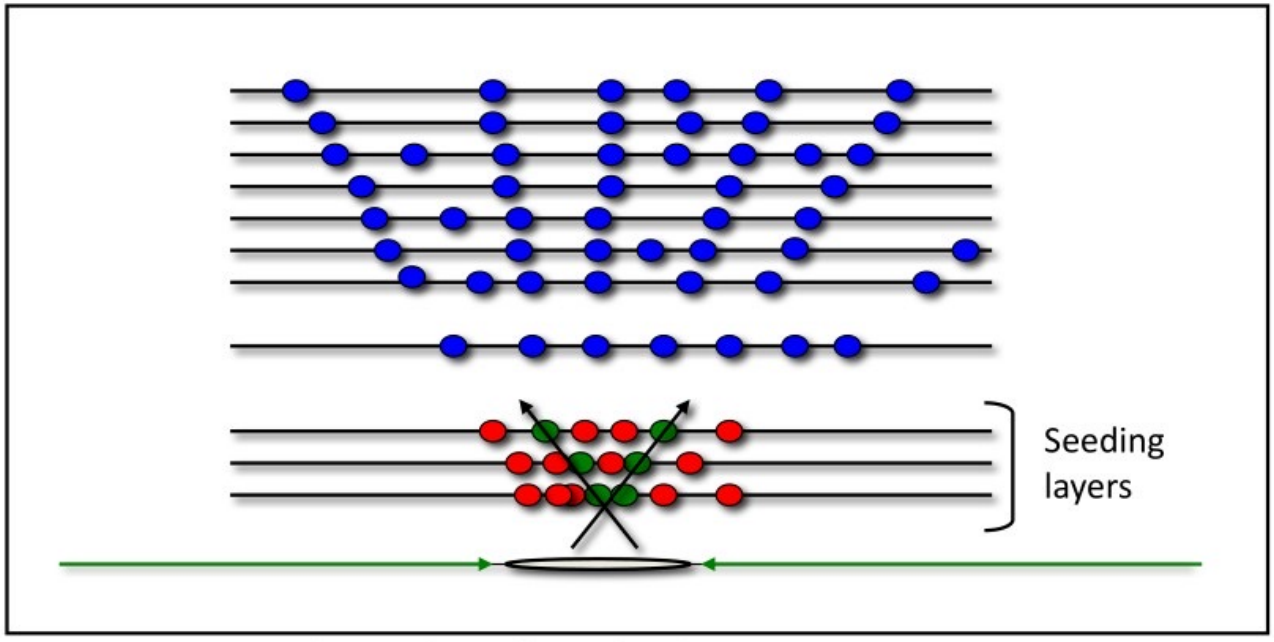
Modules for track reconstruction (III)

Trajectory seeding

Initial estimate of trajectory parameters from a small subset of measurements, i.e. the hits on the ***seeding*** layers of the detector.

Output is collection of `TrajectorySeed`

Later iterations build seeds in External layers as well

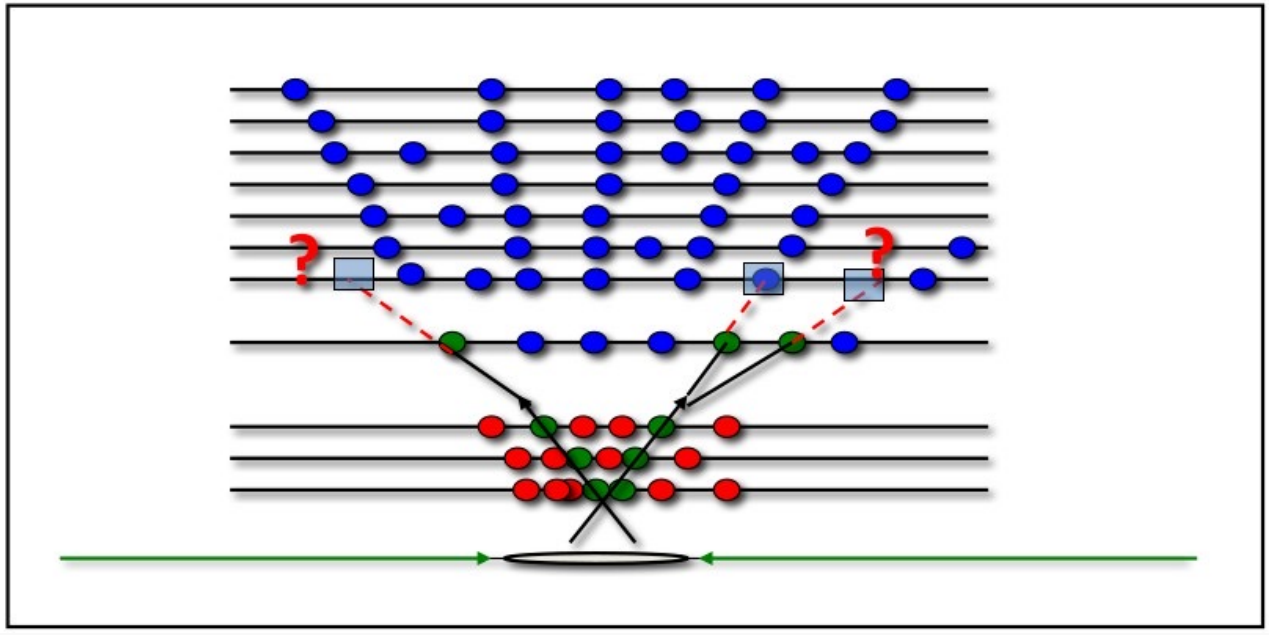


Modules for track reconstruction (IV)

Trajectory building

Iterative process that aims to collect all hits originating from the same charged particle.

Output is collection of `TrackCandidate`

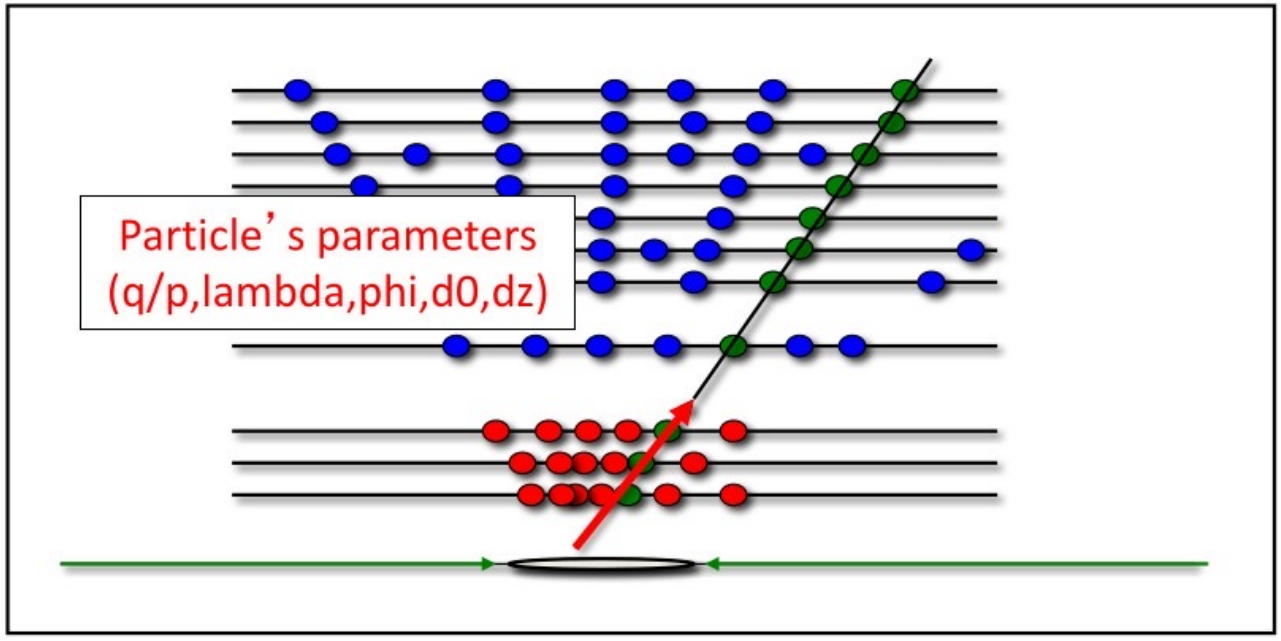


Modules for track reconstruction (V)

Trajectory fitting

Estimation of final track parameters from the Kalman Filter+Smoother fit of the full set of hits associated to the same charged particle.

Output is collection of `reco::Track`

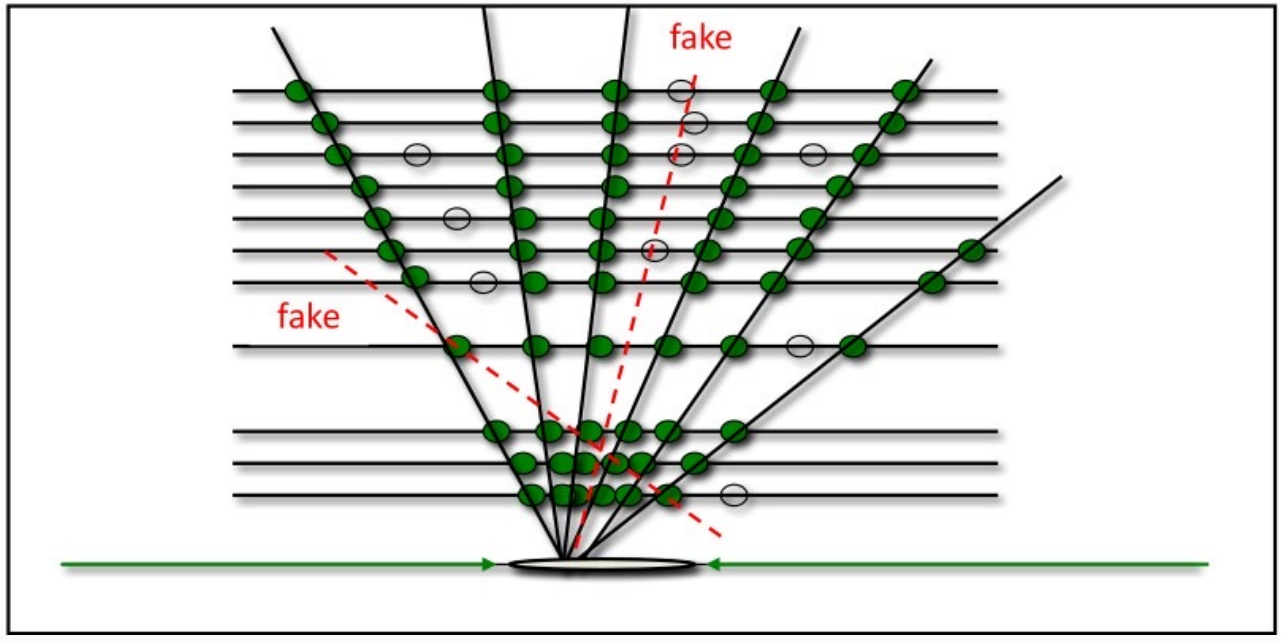


Modules for track reconstruction (VI)

Track collection filtering

Removal of ***fake or badly reconstructed*** tracks (make use of BDTs trained for each iteration).

Output is a smaller collection of `reco::Track`



Seeding

Seeding

- The pattern recognition needs an initial estimate of the track parameters as a starting point
- Can be derived from a small number of hits in a subset of detectors
- This choice of layers to look for seeds is the main difference between tracking iterations

- The best initial seeds can be derived from pixel hits due to their very good spatial resolution
- Hits from the outer tracker or the muon system can also be used

- Discussion today will focus on how we find seeds in the pixel detector using the Cellular Automaton algorithm

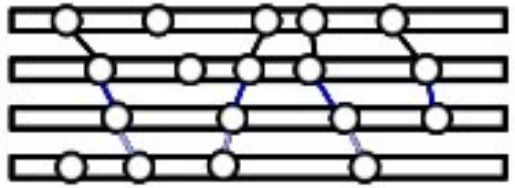
Input, size linear with PU

Raw to Digi

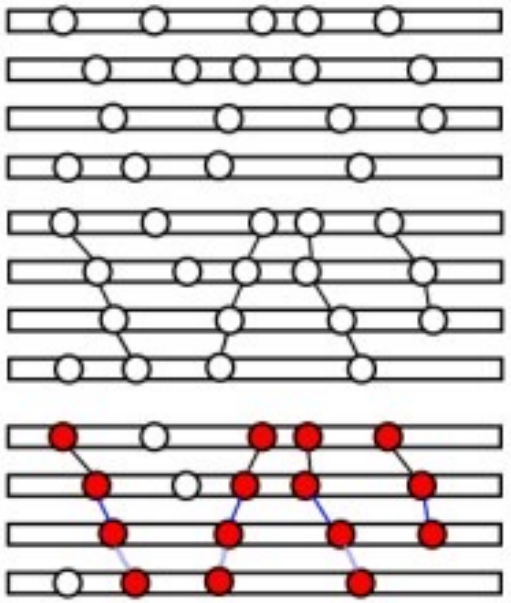
Hits - Pixel Clusterizer

Hit Pairs

Ntuplets - Cellular Automaton

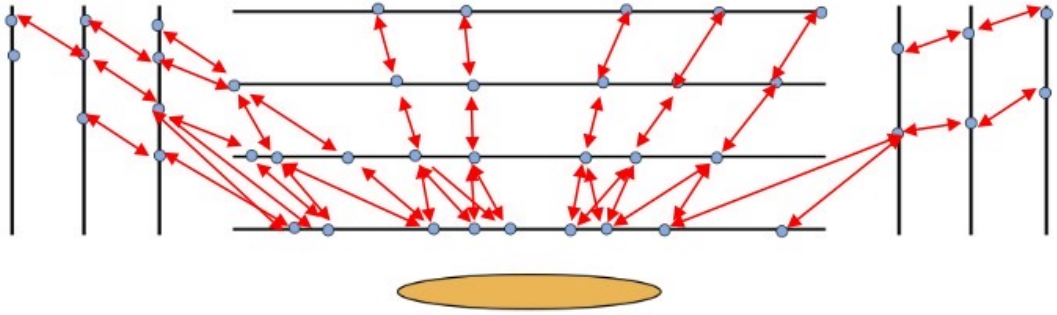


Output, size ~linear with PU



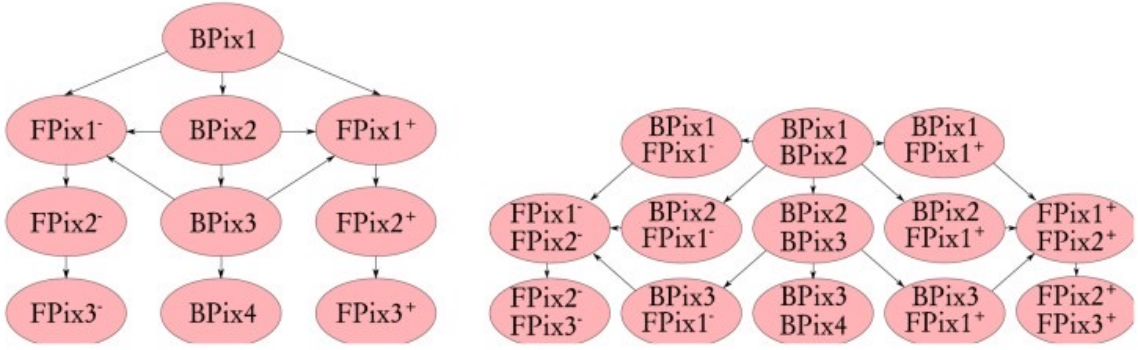
Cellular Automaton (CA)

- The CA 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



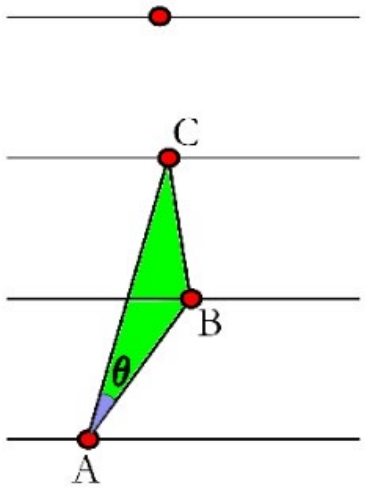
CAGraph of seeding layers

- Seeding layers interconnections
- Hit doublets for each layer pair can be computed independently by sets of threads



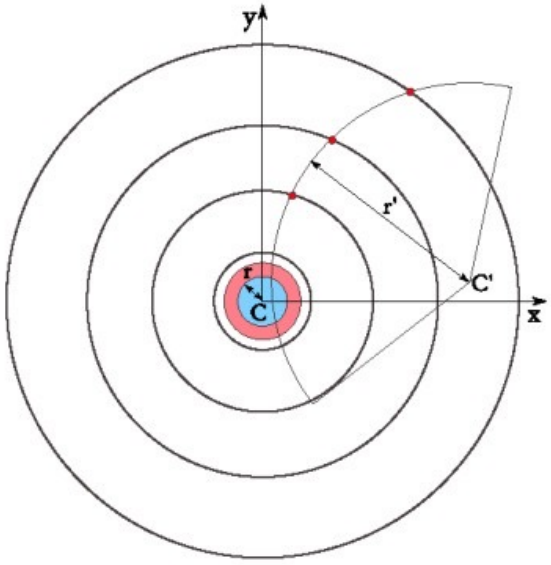
CA: R-z plane compatibility

- The compatibility between two cells is checked only if they share one hit
  - AB and BC share hit B
- In the R-z plane a requirement is alignment of the two cells:
  - There is a maximum value of ϑ that depends on the minimum value of the momentum range that we would like to explore

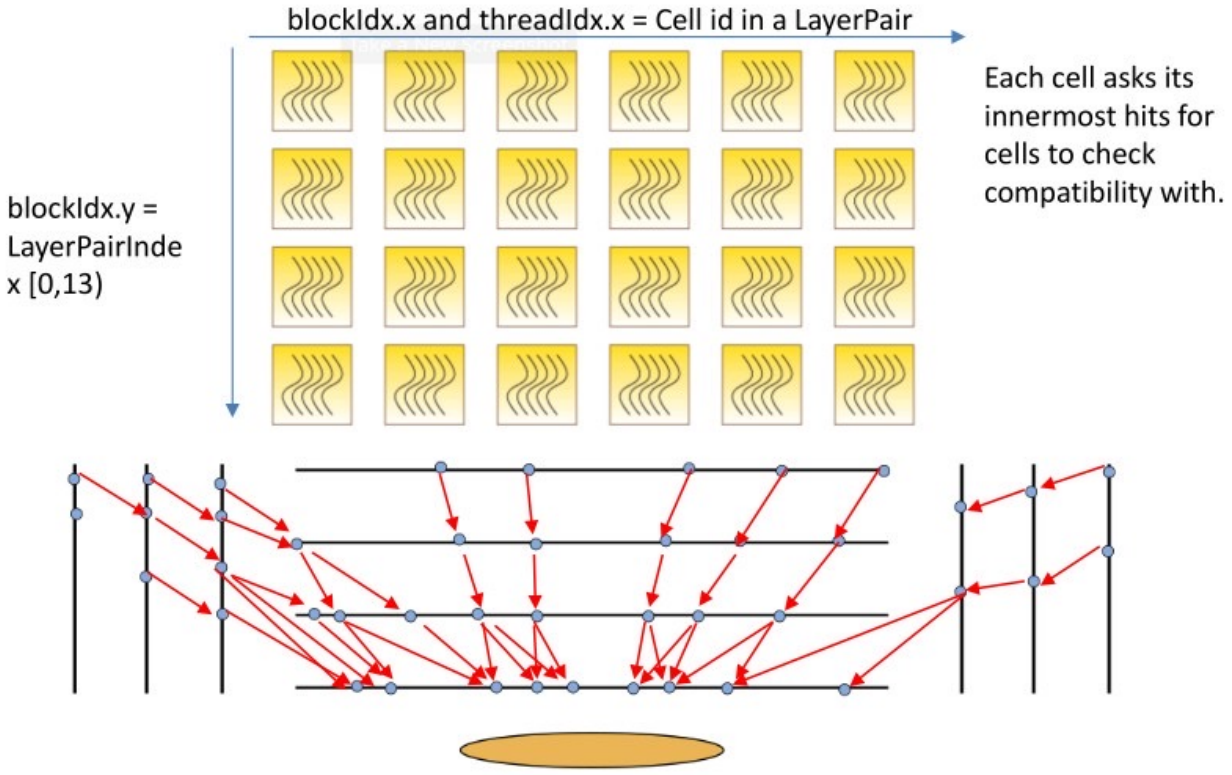


CA: x-y plane compatibility

- In the transverse plane, the intersection between the circle passing through the hits forming the two cells and the beamspot is checked:
  - They intersect if the distance between the centers d(C,C') satisfies:
    $r'-r < d(C,C') < r'+r$
  - Since it is a Out – In propagation, a tolerance is added to the beamspot radius (in red)
- One could also ask for a minimum value of transverse momentum and reject low values of r'

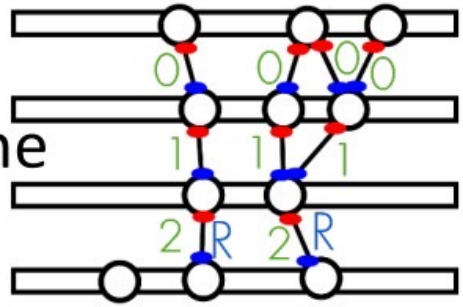


Cells Connection



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.



- Seeds are then built using a track fit similar to the the Kalman Filter used for the full tracks using a *SeedFromConsecutiveHitsCreator*

Trajectory Building and Fitting

CkfTrajectoryBuilder

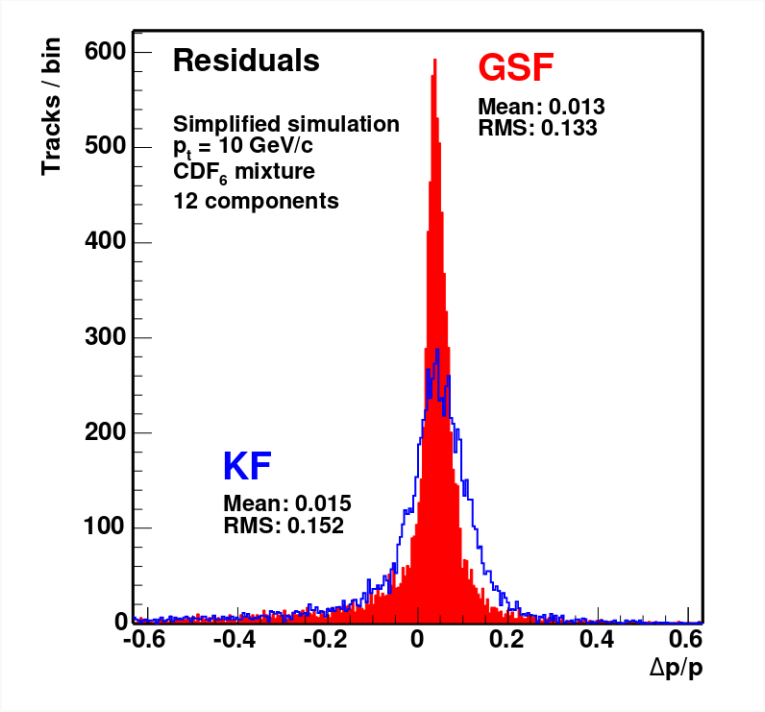
- Performs the *pattern recognition* (trajectory building) using a a Kalman Filter approach
  - Starts from the seed layer and uses the seed parameters as the initial state
  - Uses the **navigation** to find the compatible detector layers the trajectory is expected to intersect next
  - Uses the **propagator** to go to the surface of those layers
  - Searches for **compatible measurements** (hits) on these detectors, taking into account the angle of the trajectory wrt the detector surface
  - Updates the trajectory state taking into account the added hit
- This is repeated until some stopping criterion is reached (no more detector layers, too many layers without compatible measurements, etc.)
- If multiple compatible hits are found on one layer, the trajectory building branches
- Have to make sure to limit this branches, only retain a certain number of trajectories at each step

Smoothing/Fitting

- The TrackCandidates produced in the trajectory building do not necessarily have the optimal track parameters yet (they can for example be biased by constraints on the seeds from the seed building step)
- To find the best fit parameters, a final fit using a Kalman filter and smoother
  - Starts from the seed layer and uses the seed parameters as the initial state
  - A new initial track state is obtained from the innermost hits of the track
  - It is then propagated outwards, updating the state with each hit sequentially
  - The final track state at the outermost hit is then used as the initial state for another round of Kalman filtering going from the outside in
  - The track state at each surface can then be obtained as the weighted average of the two trajectories, making maximal use of the available information
- During this procedure, the hit position uncertainty (and in the pixel detector the hit position itself) are updated using the track parameters
- An outlier hit rejection is performed based on χ^2 criteria, triggering a new filtering and smoothing if a hit is removed

GSF Tracking

- Kalman Filters for track reconstruction have their limitations:
  - It is a linear least-squares estimator and is optimal if all PDFs involved are Gaussian
  - Therefore, uses single Gaussians to model the probability for energy losses in the detector material when propagating from layer to layer
  - Not a good choice for electrons, which have a high probability for large non-Gaussian energy losses due to bremsstrahlung
- For electrons, we a non-linear extension of the KF, the Gaussian Sum Filter (GSF)
  - Approximates the energy loss distribution with a mixture of several Gaussians, based on the Bethe-Heitler model
  - This improved modeling of radiative energy losses leads to an improved resolution for track parameters



Track selection

Track selection

- Tracks are selected based on their hit pattern (number of hits in the pixels, strip, number of lost hits, number of 3D hits...), impact parameters, fit χ^2
- Optimal cuts obtained for each track iteration using at BDT
- Selection is then applied cut-based using the MultiTrackSelector

- 3 working points: Loose, Tight, HighPurity
- Only high purity tracks are actually used in almost all cases

The exercise

- This short exercise will provide an introduction to using tracks for analysis in both AOD and MINIAOD data formats
  - Extracting basic track parameters
  - selecting tracks for analysis using track quality cuts
  - reconstructing invariant masses from tracks
  - extracting basic parameters for primary vertices
  - Reconstructing secondary vertices and composite particles
  - Compute tracking efficiency via Tag and Probe

- Useful links:
  - Exercise Twiki: https://twiki.cern.ch/twiki/bin/viewauth/CMS/SWGuideCMSDataAnalysisSchoolPisa2019TrackingAndVertexingShortExercise
  - CMSSW DXR https://cmssdt.cern.ch/dxr/CMSSW/source/
  - CMSSW LXR: https://cmssdt.cern.ch/lxr/ or http://cmslxr.fnal.gov/lxr/
  - CMSSW doxygen: http://cmsdoxygen.web.cern.ch/cmsdoxygen/
  - Software Guide Track Reconstruction: https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideTrackReco
  - Tracking Long Exercise at CMSPOS 2019: https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideCMSPhysicsObjectSchoolAACHEN2019Tracking

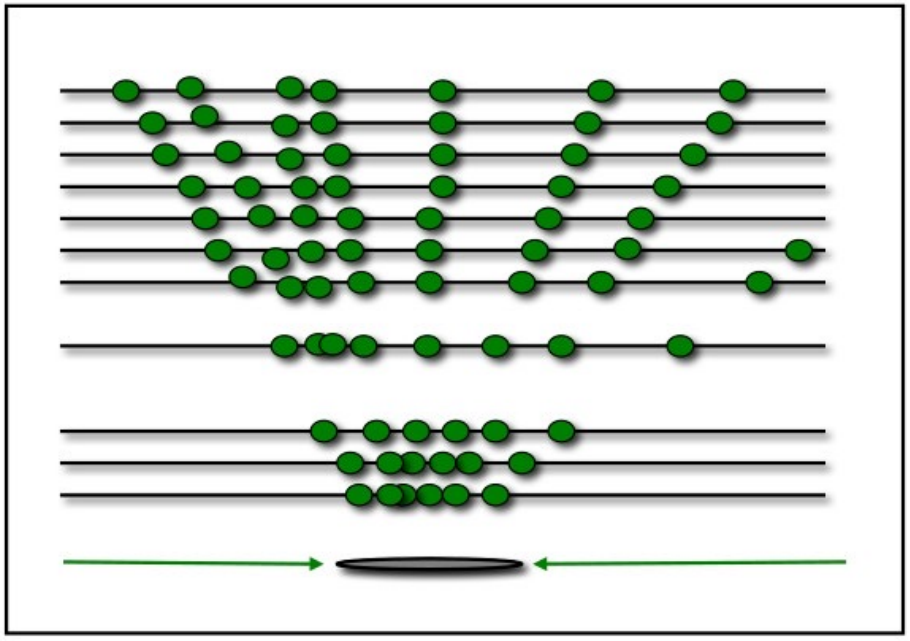
Iterative Tracking

Iterative Tracking

- In general, high momentum primary tracks are easy to reconstruct and do not take too much time
  - primary vertex and beamspot constraint
  - high precision pixel hits give high quality seeds
  - less multiple scattering, smaller search window
- Relaxing these requirements increases the combinatorics, i.e. the following kind of tracks are more difficult: low p_T, primary tracks missing a pixel hit, displaced tracks
  - conversions, nuclear interactions, heavy flavor decays
  - tracks crossing inefficient parts of the pixel detector
- Iterative tracking aims at reducing the combinatorial problem so that problematic tracks can also be reconstructed with the CPU time budget
- The idea is to run track reconstruction several times, and each time hits used by tracks from previous iterations are masked

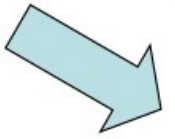
Iterative Tracking

Initial collection of hits

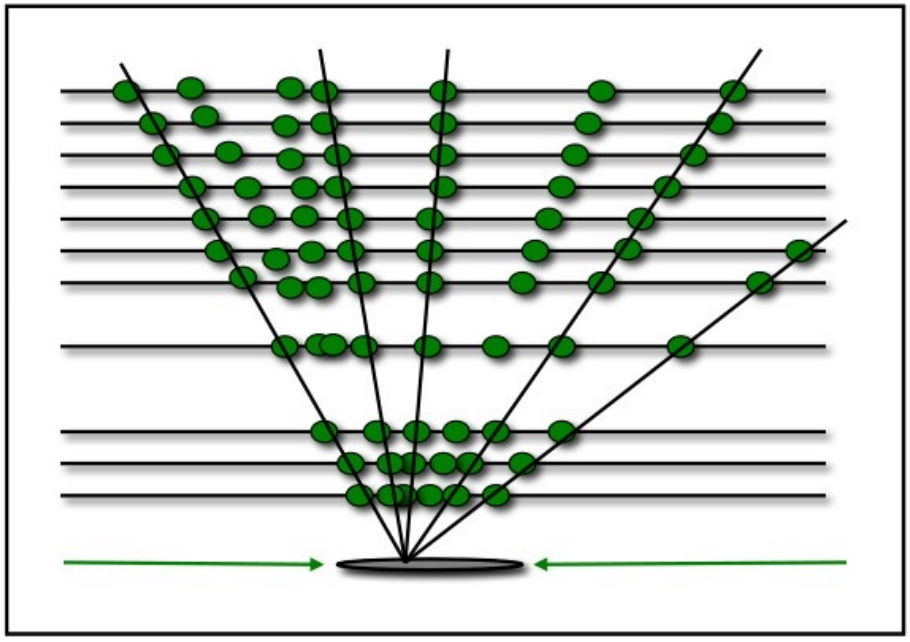


Iterative Tracking

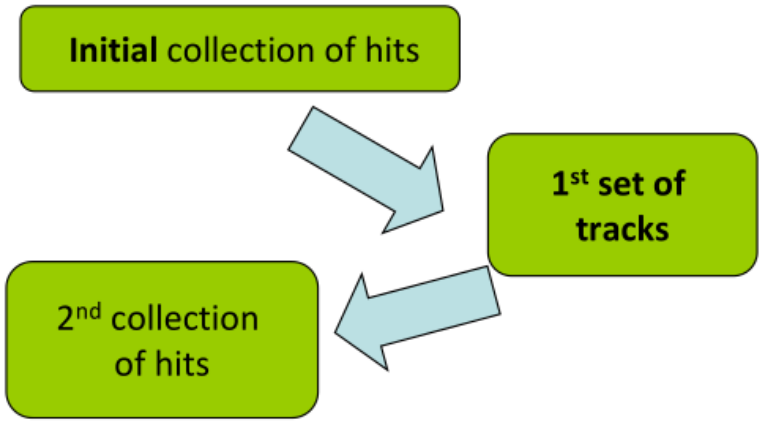
Initial collection of hits

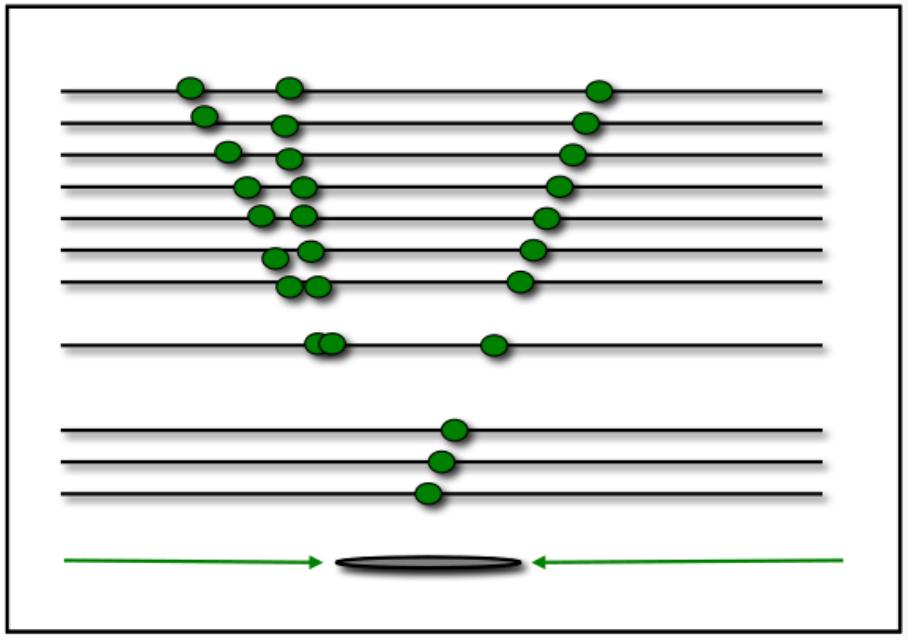


1st set of tracks

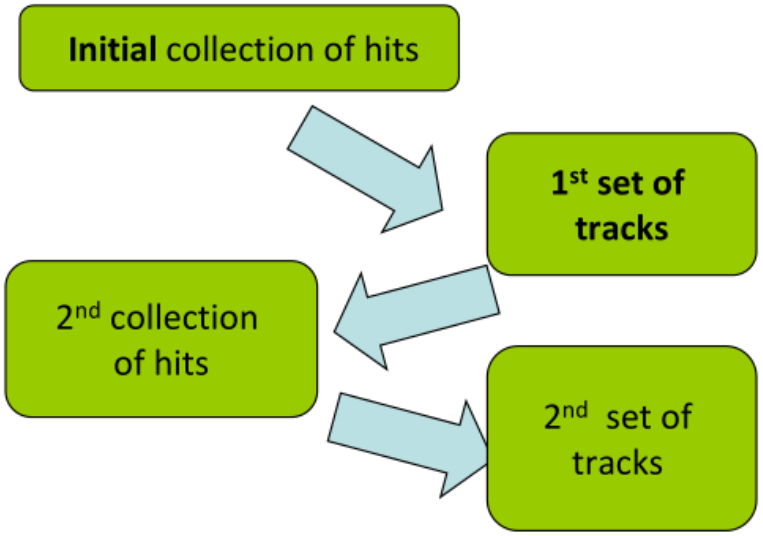


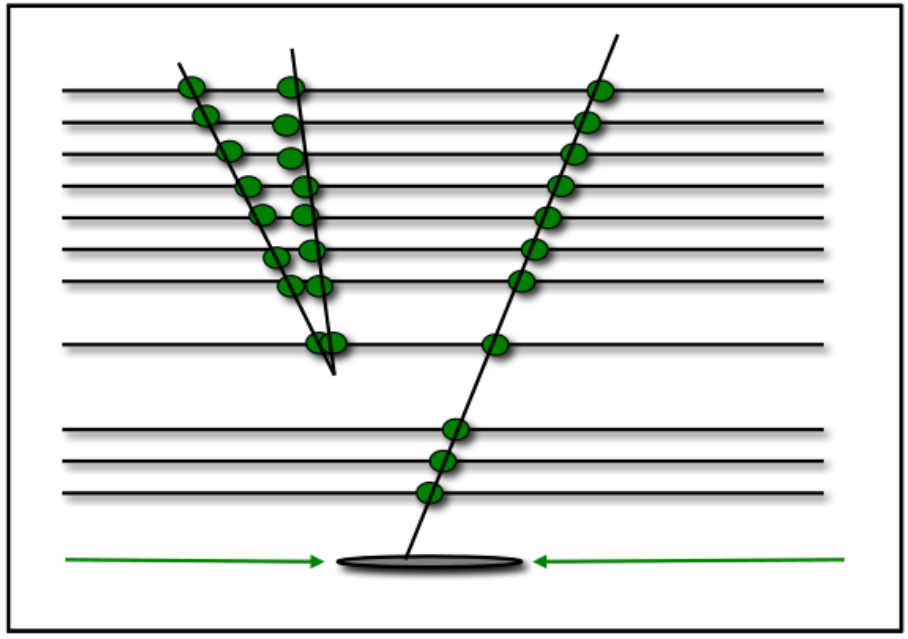
Iterative Tracking



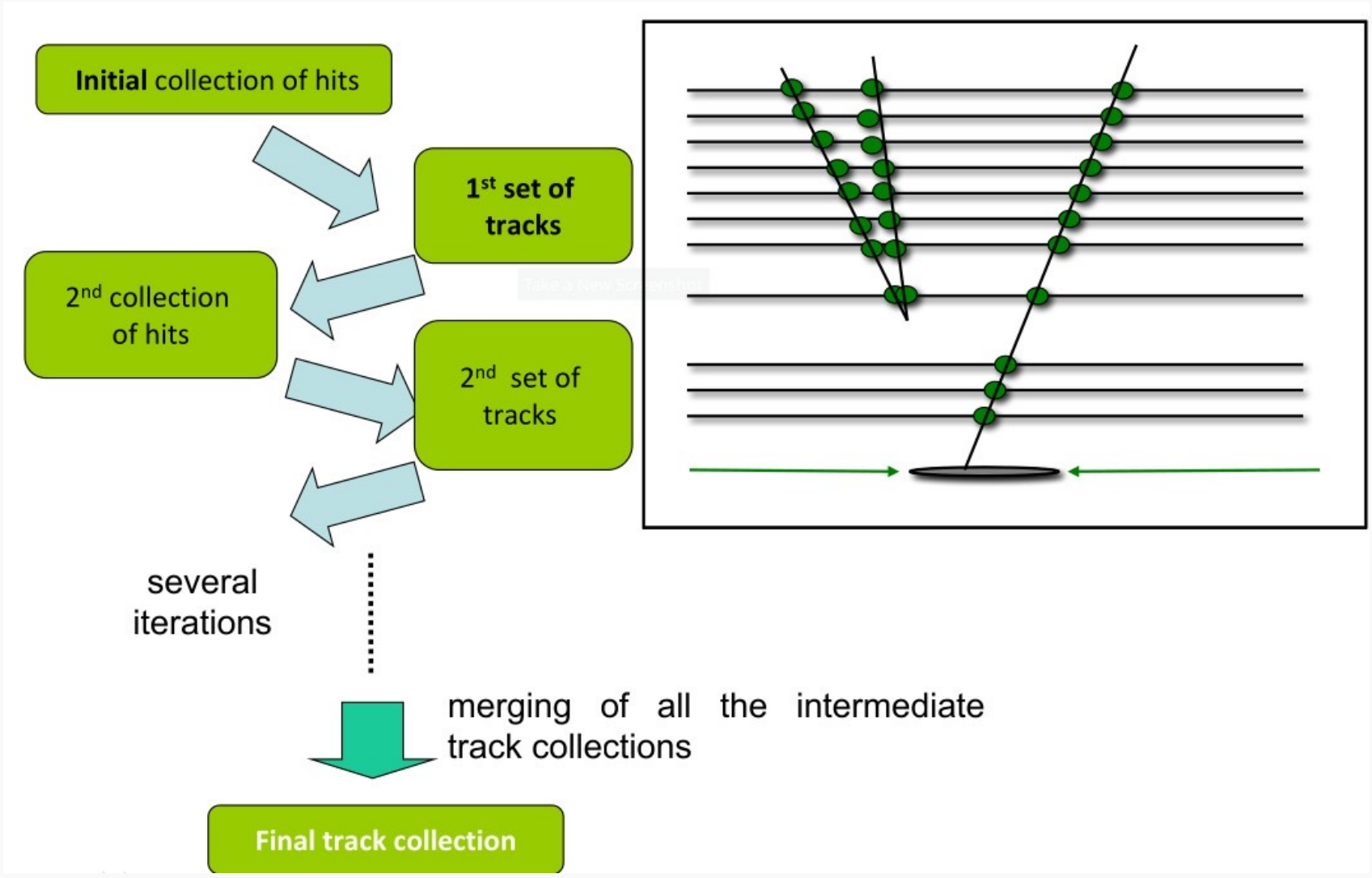


Iterative Tracking

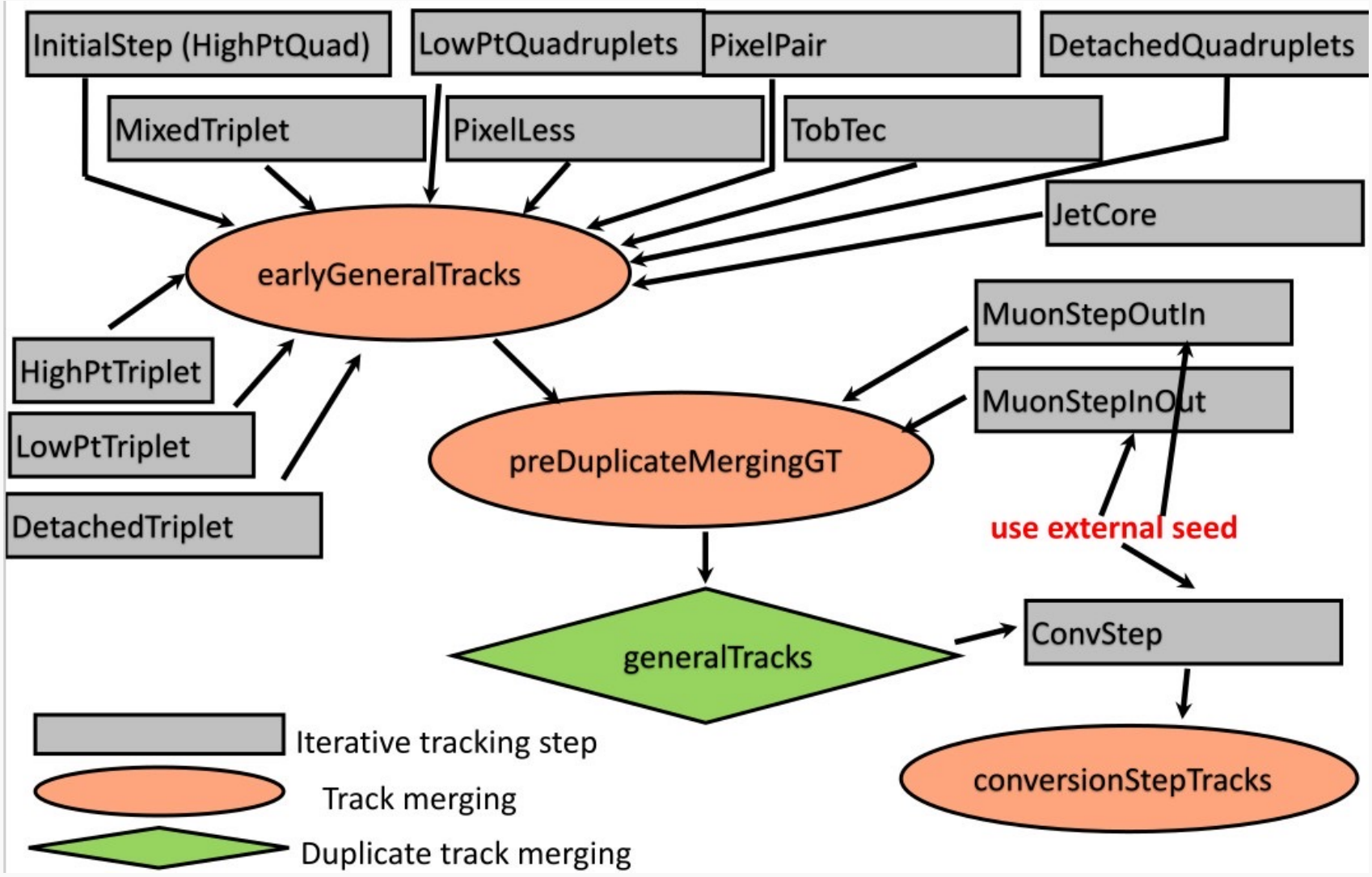




Iterative Tracking



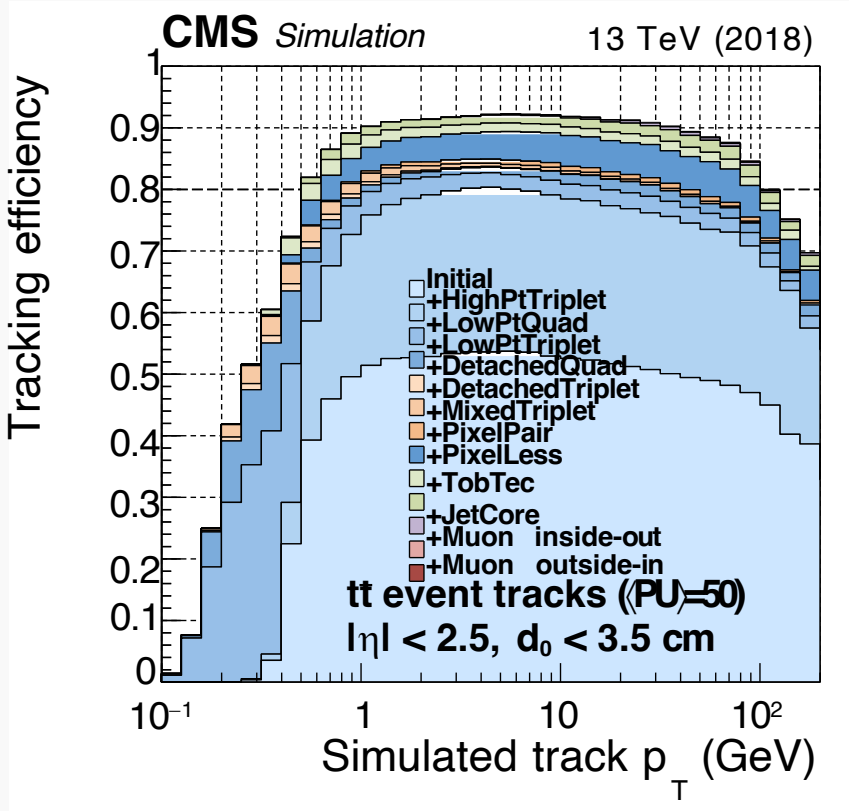
Iterative Tracking Sequence



Performance of the iterative steps

- Thanks to iterative tracking CMS can reconstruct with good efficiency low p_T and displaced tracks

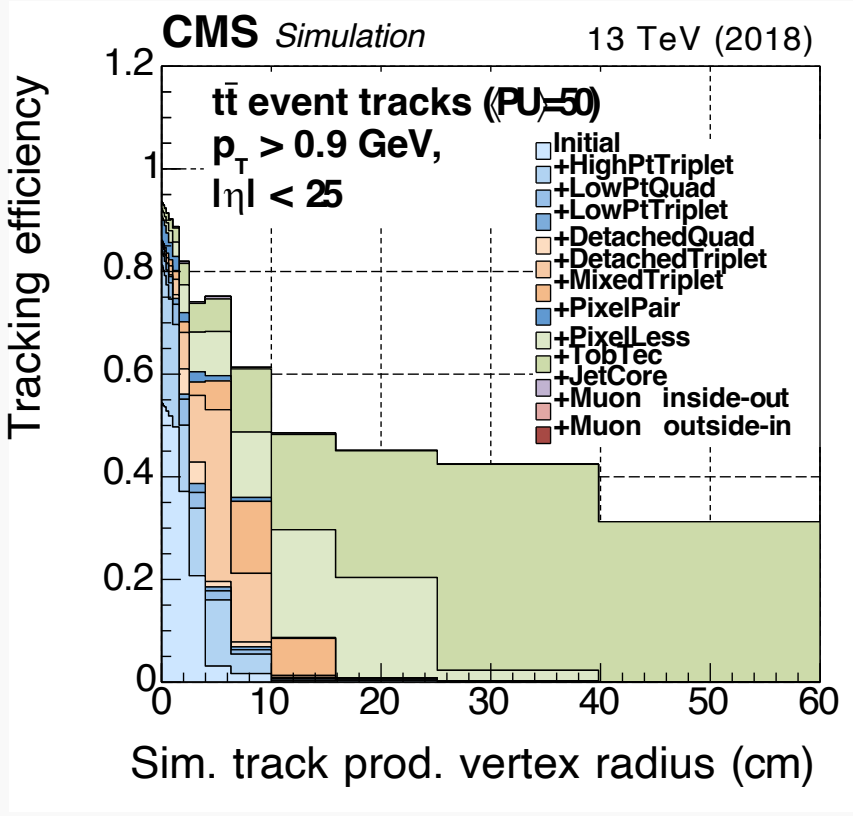
Iteration	Seeding	Target track
Initial	pixel quadruplets	prompt, high p_T
LowPtQuad	pixel quadruplets	prompt, low p_T
HighPtTriplet	pixel triplets	prompt, high p_T recovery
LowPtTriplet	pixel triplets	prompt, low p_T recovery
DetachedQuad	pixel quadruplets	displaced--
DetachedTriplet	pixel triplets	displaced-- recovery
MixedTriplet	pixel+strip triplets	displaced-
PixelLess	inner strip triplets	displaced+
TobTec	outer strip triplets	displaced++
JetCore	pixel pairs in jets	high-p_T jets
Muon inside-out	muon-tagged tracks	muon
Muon outside-in	standalone muon	muon



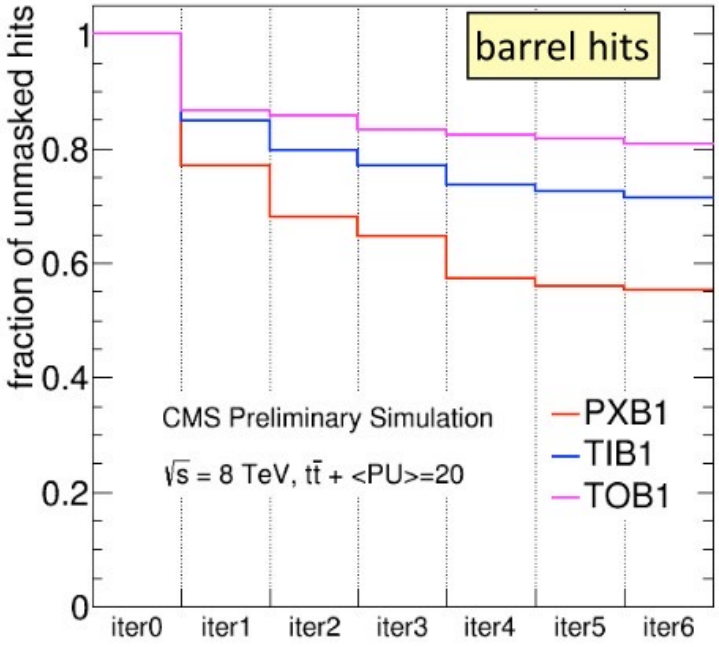
Performance of the iterative steps

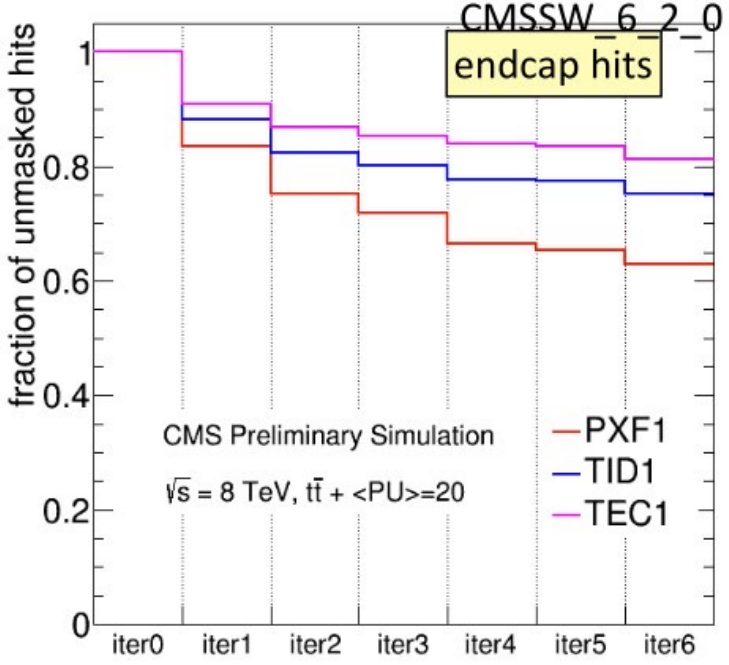
- Thanks to iterative tracking CMS can reconstruct with good efficiency low p_T and displaced tracks

Iteration	Seeding	Target track
Initial	pixel quadruplets	prompt, high p_T
LowPtQuad	pixel quadruplets	prompt, low p_T
HighPtTriplet	pixel triplets	prompt, high p_T recovery
LowPtTriplet	pixel triplets	prompt, low p_T recovery
DetachedQuad	pixel quadruplets	displaced--
DetachedTriplet	pixel triplets	displaced-- recovery
MixedTriplet	pixel+strip triplets	displaced-
PixelLess	inner strip triplets	displaced+
TobTec	outer strip triplets	displaced++
JetCore	pixel pairs in jets	high-p_T jets
Muon inside-out	muon-tagged tracks	muon
Muon outside-in	standalone muon	muon



So, is tracking easy in the iterative world?

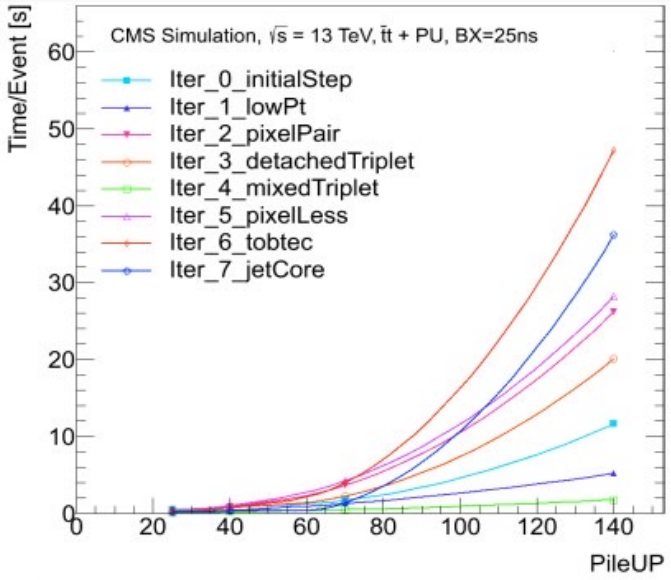


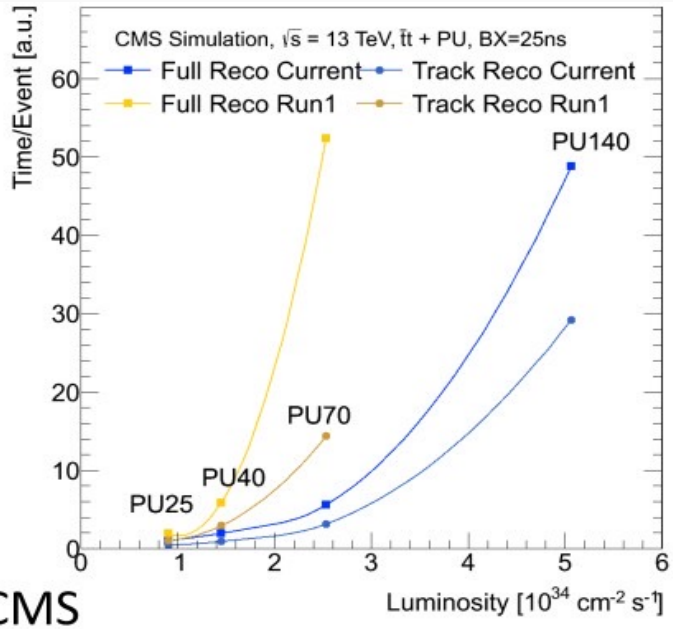


First iteration is most effective in reducing the number of hits. After all iterations, more than half of the hits are still not associated to tracks.

The iterative tracking approach reduces the combinatorics but tracking is still a big challenge.

Timing

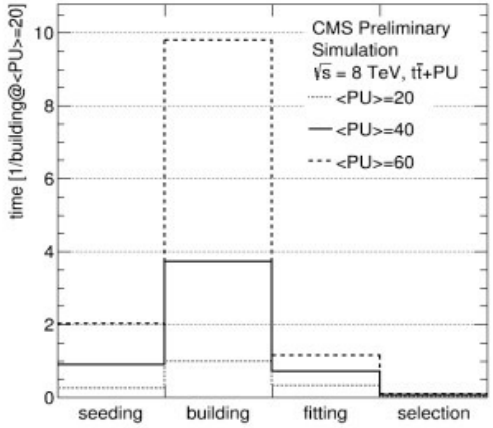




Tracking takes most of the computing time of CMS event reconstruction. Timing grows with PU, especially for PU>70.

Most of the time is spent during pattern recognition (building).

Steps not using pixel quad&triplets are those taking most of the time (tobTec, pixelLess, pixelPair).



Geometry

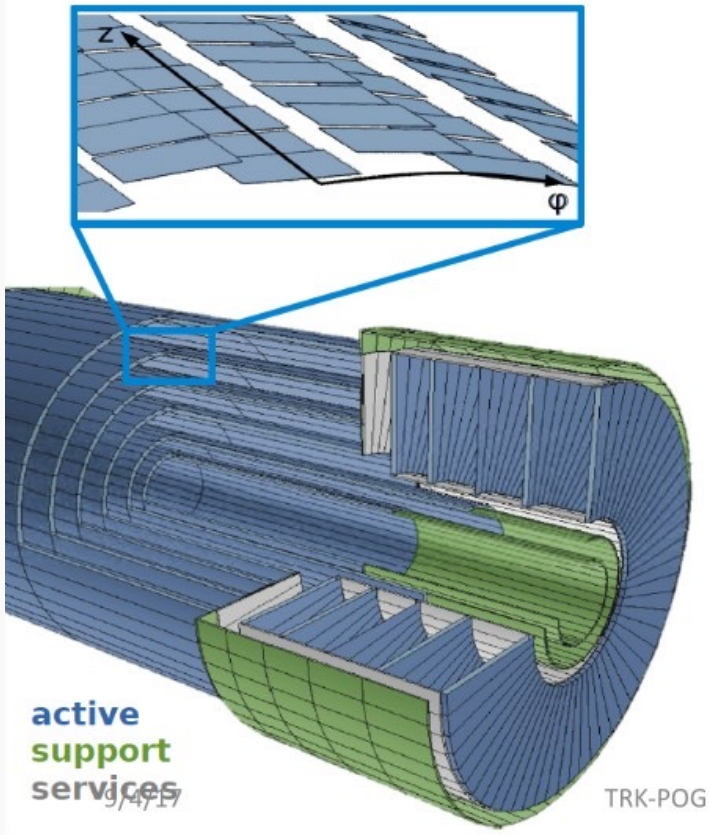
Layout Modeling

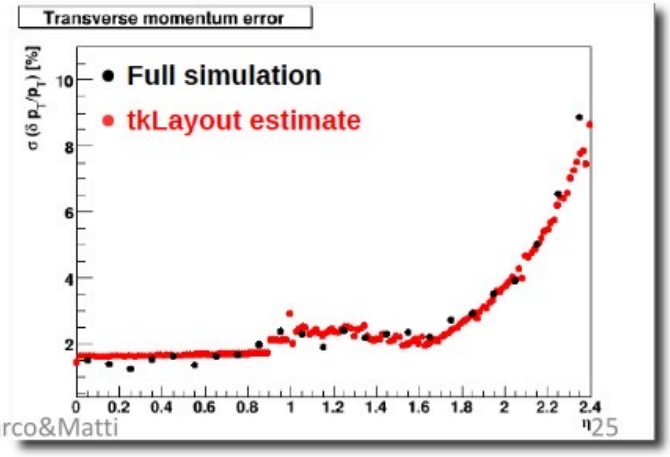
layout modelling

Design of detector: a many-parameters problem

Innovative solution: **tkLayout**

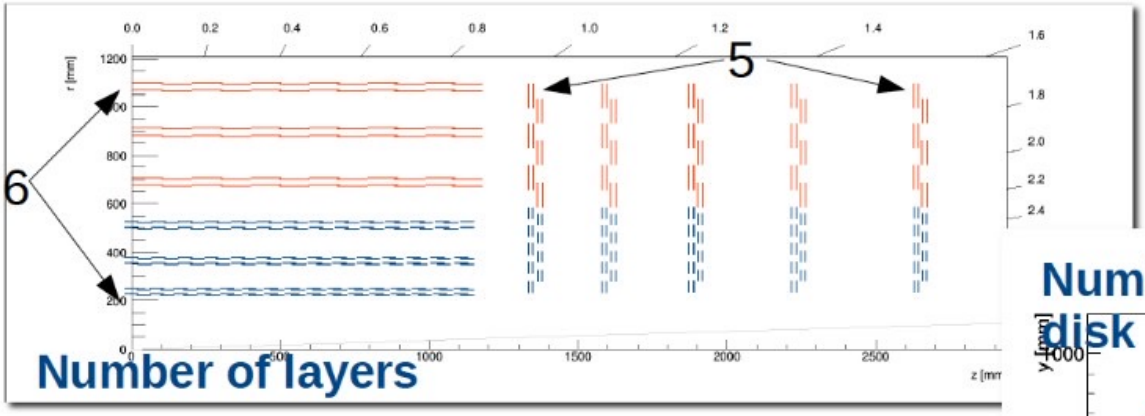
- detector layout generator software using a **simple** description of design parameters
- creates a 3D model of the layout with a description of the materials
- makes an *a priori* estimate of **tracking resolution** (much faster than a traditional simulation)
- **output was validated** by comparing with full simulation of present detector



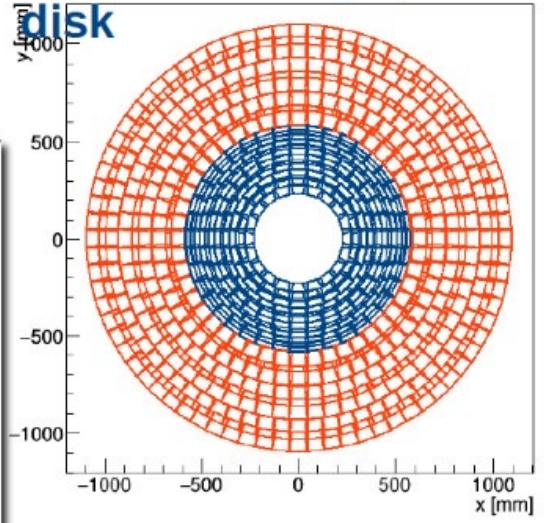


Detector geometry

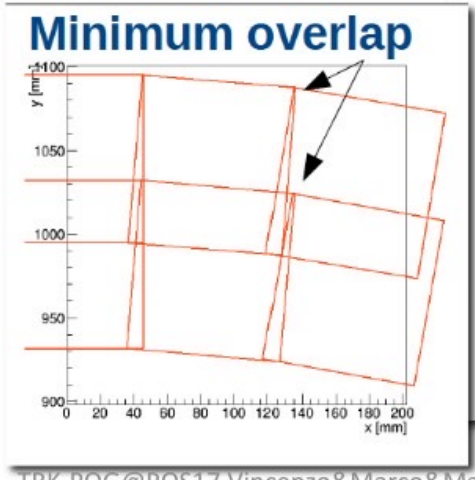
Simple parameters define the entire geometry, like



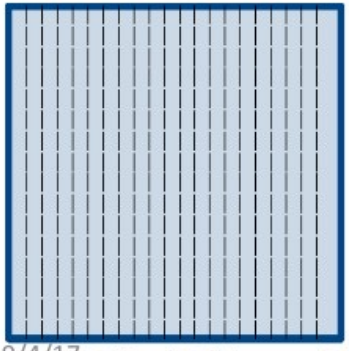
Number of rings per disk



Minimum overlap



Sensor geometry



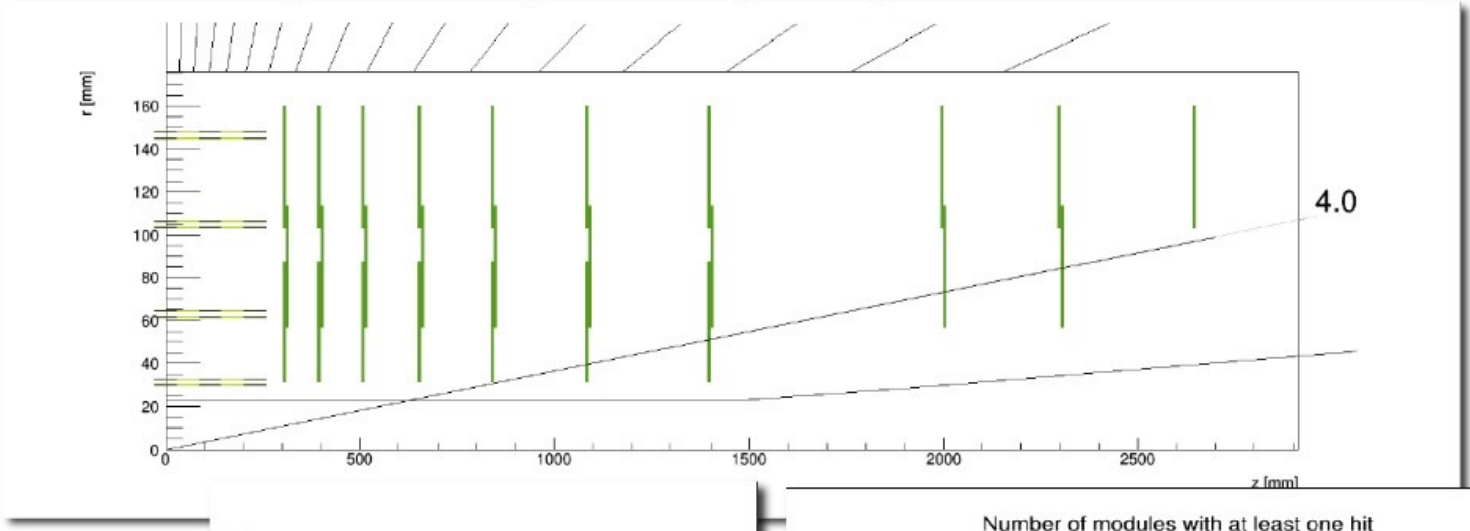
(ex: 1024×16 MacroPixel)

2

Coverage Check

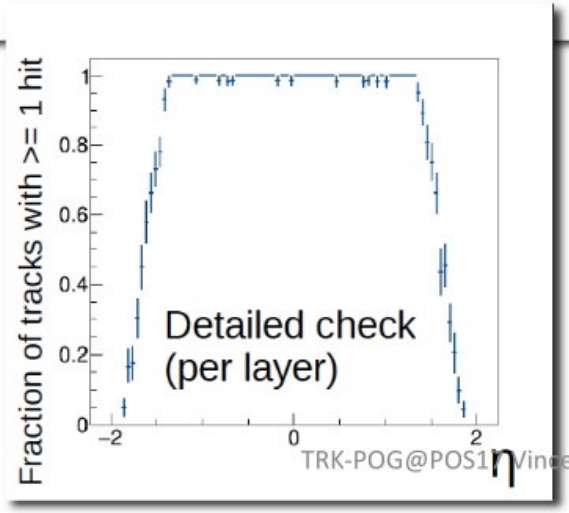
Coverage check

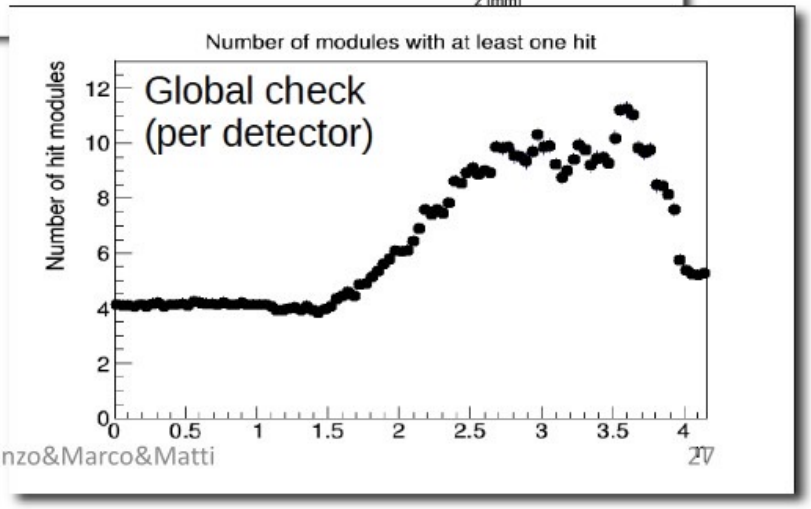
To validate geometry and qualify hermetic coverage



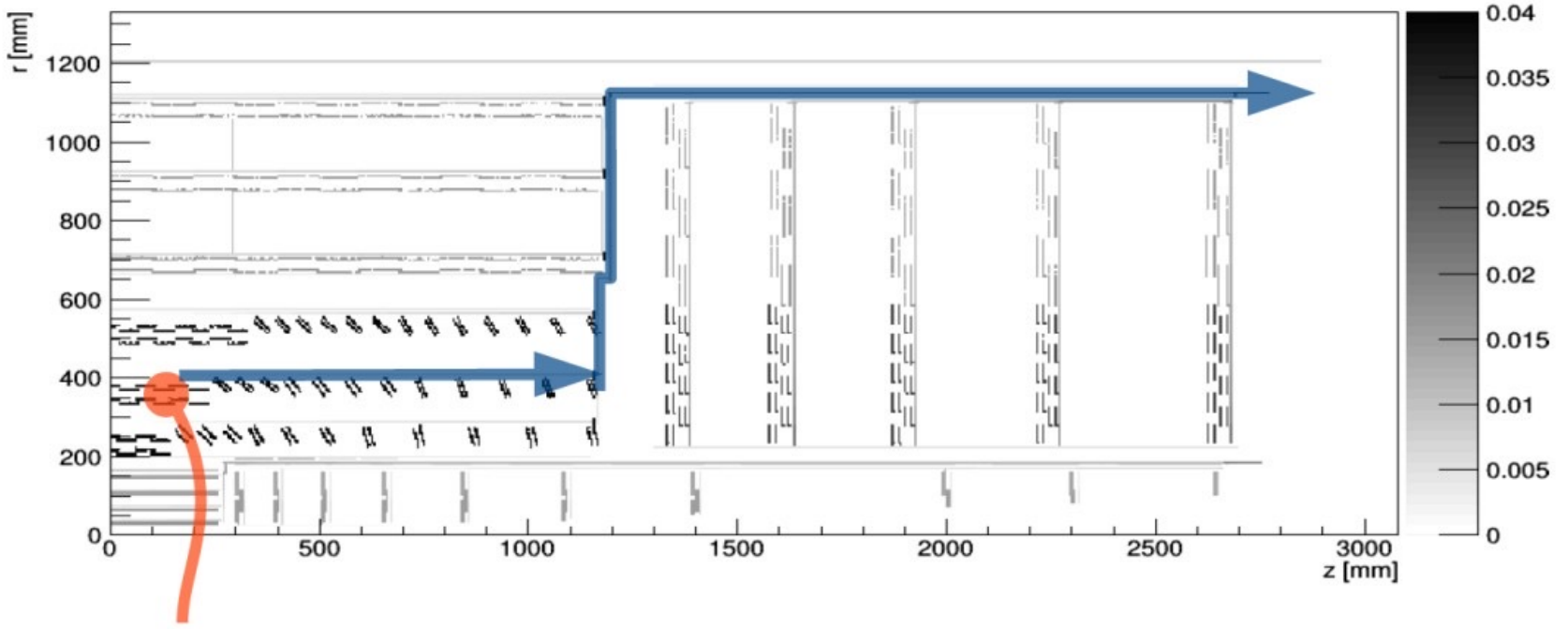
Beam spot shape and size are taken into account

"Tracks" all have $p_T=\infty$





Material



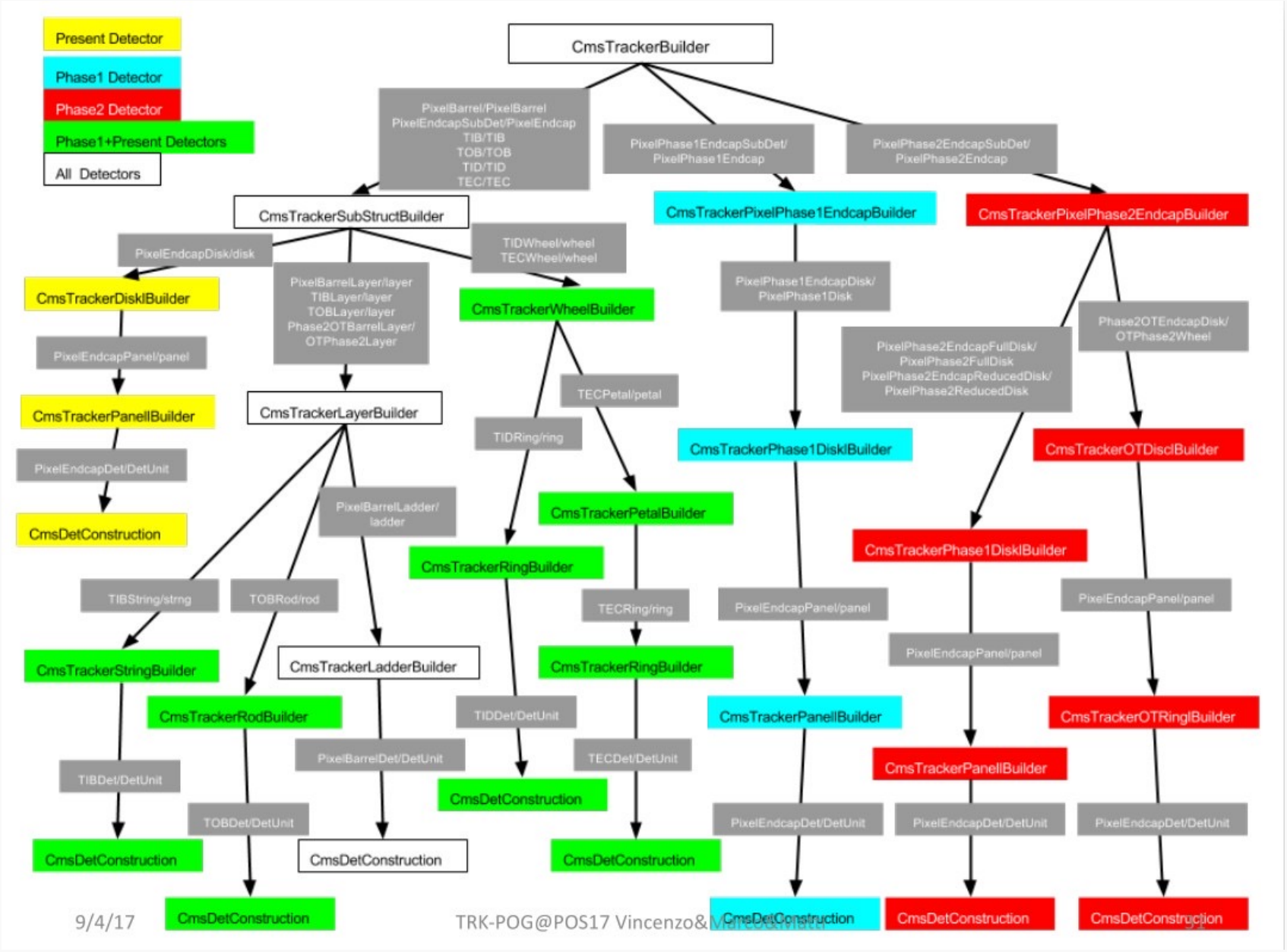
Material on active elements + Material for services automatically routed

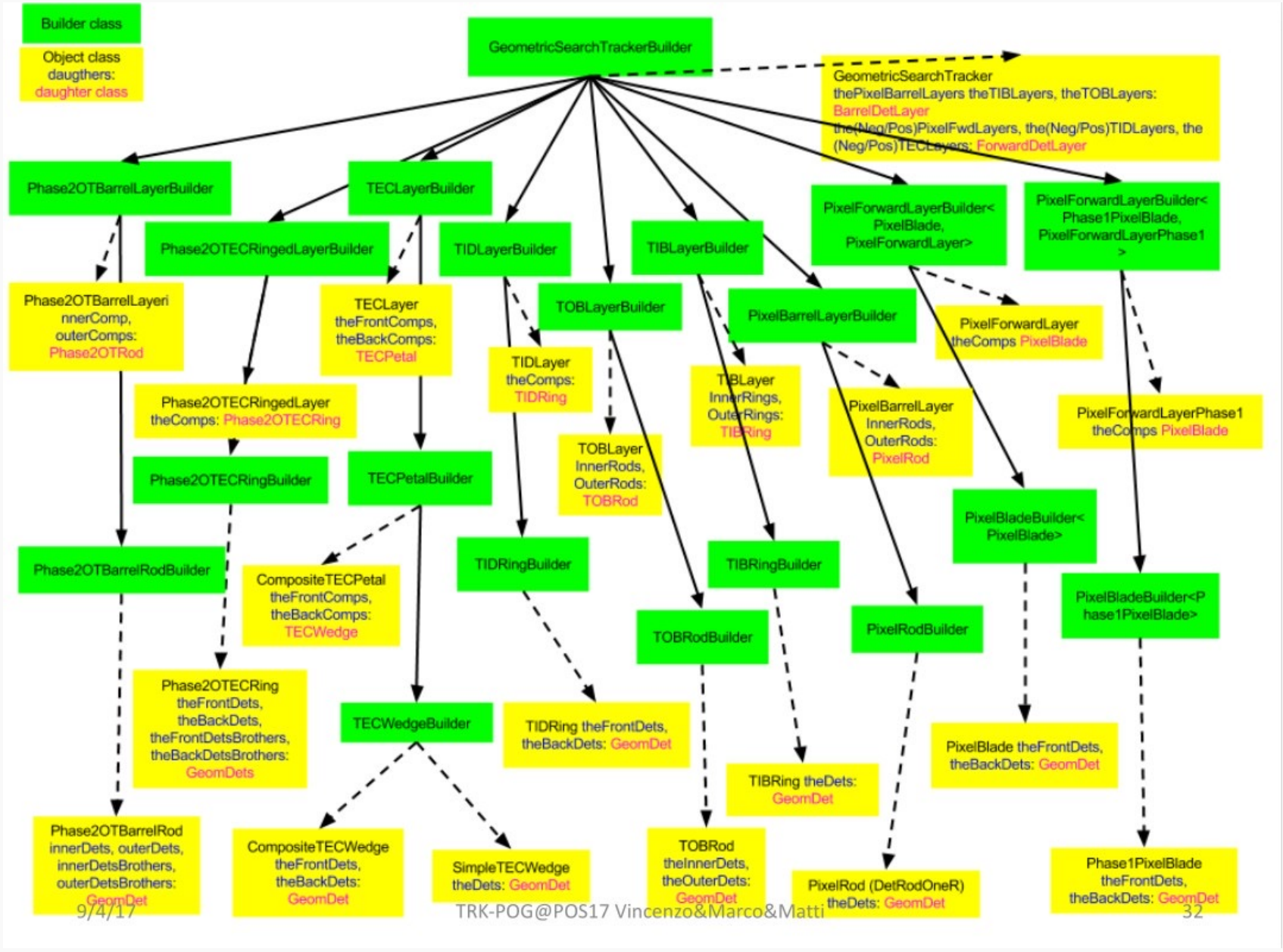
From TkLayout to C++

- TkLaylout: tool used by detector group to design and characterize the detector
  - Proven to reproduce detailed simulation “well”
- TkLaylout produces a hierarchical geometry description in XML
  - Input to build Detailed Simulation Geometry
  - And the reconstruction geometry
    - hierarchical (layers, “sub-layers” …., sensitive modules)
      - Sort in one dimention at each level of sub-component
    - Only sensitive volumes
      - Material estimated used G4 tracking and “lamped” on closest sensor

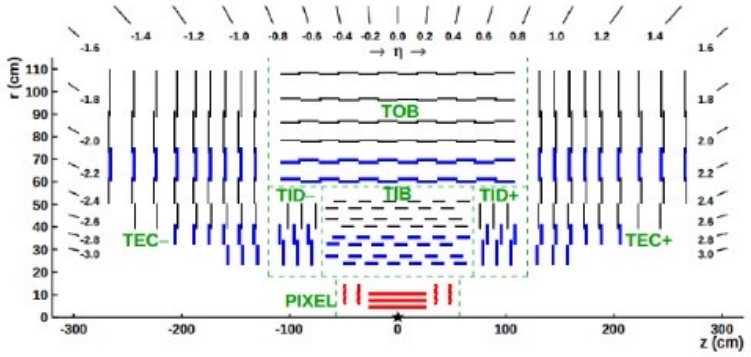
“Geometry” Implementation

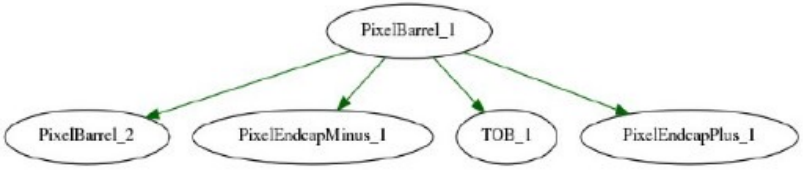
- XML description (and its in memory representation)
  - Pretty generic and “verbose”
- DB and in memory representations (different!)
  - Optimized for disk and memory space
  - Basic component is a “DetUnit”: thick bounded plane with all sorts of attached properties
    - Cached derived quantities added for use in time-critical algorithms
  - Basic hand-coded “DetUnits” containers (sorted vectors, and “maps”)
  - OO Naïve (but all const!)
- SearchGeometry Hierarchy: *defines layer’s content*
  - Heavily hand coded, specialized and optimized
  - “hard coded” containment hierarchy. Dimensions and composition determined at run time
  - Still OO Naïve (Objects owning “sorted” vectors of objects, etc.)
- Navigation Geometry: *links layers*
  - Built automatically at runtime using hand coded generic algorithms
- Measurement tracker: *interface to run-time data*
  - Heavily hand coded, specialized and optimized

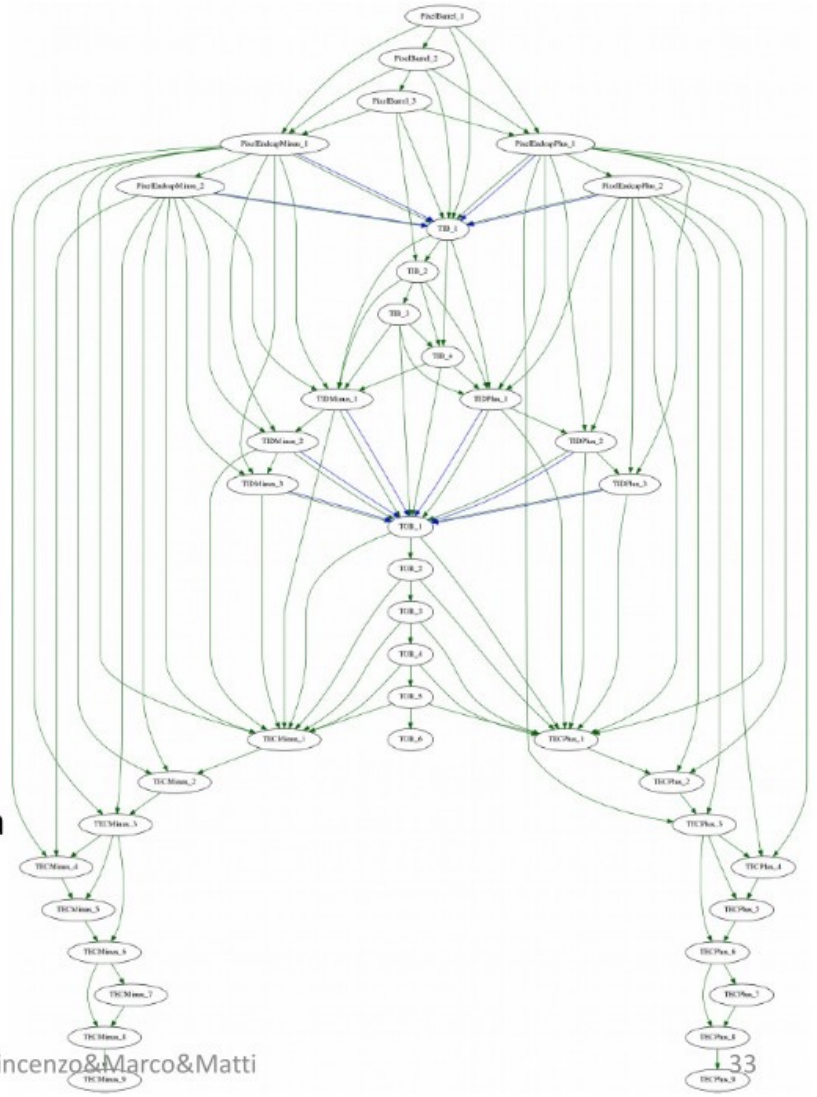

Present Detector
Phase1 Detector
Phase2 Detector
Phase1+Present Detectors
All Detectors
CmsTrackerBuilder
PixelBarrel/PixelBarrel
PixelEndcapSubDet/PixelEndcap
TIB/TIB
TOB/TOB
TID/TID
TEC/TEC
PixelPhase1EndcapSubDet/
PixelPhase1Endcap
PixelPhase2EndcapSubDet/
PixelPhase2Endcap
CmsTrackerSubStructBuilder
CmsTrackerPixelPhase1EndcapBuilder
CmsTrackerPixelPhase2EndcapBuilder
PixelEndcapDisk/disk
TIDWheel/wheel
TECWheel/wheel
CmsTrackerDiskBuilder
PixelBarrelLayer/layer
TIBLayer/layer
TOBLayer/layer
Phase2OTBarrelLayer/
OTPhase2Layer
CmsTrackerWheelBuilder
PixelPhase1EndcapDisk/
PixelPhase1Disk
PixelPhase2EndcapFullDisk/
PixelPhase2FullDisk
PixelPhase2EndcapReducedDisk/
PixelPhase2ReducedDisk
Phase2OTEndcapDisk/
OTPhase2Wheel
PixelEndcapPanel/panel
CmsTrackerPanelBuilder
CmsTrackerLayerBuilder
TECPetal/petal
TIDRing/ring
CmsTrackerPhase1DiskBuilder
CmsTrackerOTDiscBuilder
PixelEndcapDet/DetUnit
CmsDetConstruction
PixelBarrelLadder/
ladder
CmsTrackerPetalBuilder
CmsTrackerRingBuilder
CmsTrackerPhase1DiskBuilder
TIBString/strng
TOBRod/rod
TECRing/ring
PixelEndcapPanel/panel
PixelEndcapPanel/panel
PixelEndcapPanel/panel
CmsTrackerStringBuilder
CmsTrackerLadderBuilder
CmsTrackerRingBuilder
CmsTrackerRodBuilder
TIDDet/DetUnit
CmsTrackerPanelBuilder
CmsTrackerOTRingBuilder
PixelBarrelDet/DetUnit
TECDet/DetUnit
CmsTrackerPanelBuilder
TIBDet/DetUnit
CmsDetConstruction
TOBDet/DetUnit
PixelEndcapDet/DetUnit
PixelEndcapDet/DetUnit
PixelEndcapDet/DetUnit
CmsDetConstruction
CmsDetConstruction
CmsDetConstruction
CmsDetConstruction
CmsDetConstruction
CmsDetConstruction
CmsDetConstruction


Builder class
Object class daugthers: daughter class
GeometricSearchTrackerBuilder
GeometricSearchTracker thePixelBarrelLayers theTIBLayers, theTOBLayers: BarrelDetLayer the(Neg/Pos)PixelFwdLayers, the(Neg/Pos)TIDLayers, the (Neg/Pos)TECLayers: ForwardDetLayer
Phase2OTBarrelLayerBuilder
TECLayerBuilder
Phase2OTECRingedLayerBuilder
TIDLayerBuilder
TIBLayerBuilder
PixelForwardLayerBuilder< PixelBlade, PixelForwardLayer>
PixelForwardLayerBuilder< Phase1PixelBlade, PixelForwardLayerPhase1 >
Phase2OTBarrelLayerInnerComp, outerComps: Phase2OTRod
TECLayer theFrontComps, theBackComps: TECPetal
TOBLayerBuilder
PixelBarrelLayerBuilder
PixelForwardLayer theComps PixelBlade
TIDLayer theComps: TIDRing
TIBLayer InnerRings, OuterRings: TIBRing
PixelBarrelLayer InnerRods, OuterRods: PixelRod
PixelForwardLayerPhase1 theComps PixelBlade
Phase2OTECRingedLayer theComps: Phase2OTECRing
TOBLayer InnerRods, OuterRods: TOBRod
Phase2OTECRingBuilder
TECPetalBuilder
PixelBladeBuilder< PixelBlade>
Phase2OTBarrelRodBuilder
CompositeTECPetal theFrontComps, theBackComps: TECWedge
TIDRingBuilder
TIBRingBuilder
PixelBladeBuilder<Phase1PixelBlade>
TOBRodBuilder
PixelRodBuilder
Phase2OTECRing theFrontDets, theBackDets, theFrontDetsBrothers, theBackDetsBrothers: GeomDets
TECWedgeBuilder
TIDRing theFrontDets, theBackDets: GeomDet
PixelBlade theFrontDets, theBackDets: GeomDet
TIBRing theDets: GeomDet
Phase2OTBarrelRod innerDets, outerDets, innerDetsBrothers, outerDetsBrothers: GeomDet
CompositeTECWedge theFrontDets, theBackDets: GeomDet
SimpleTECWedge theDets: GeomDet
TOBRod theInnerDets, theOuterDets: GeomDet
PixelRod (DetRodOneR) theDets: GeomDet
Phase1PixelBlade theFrontDets, theBackDets: GeomDet

Navigation School



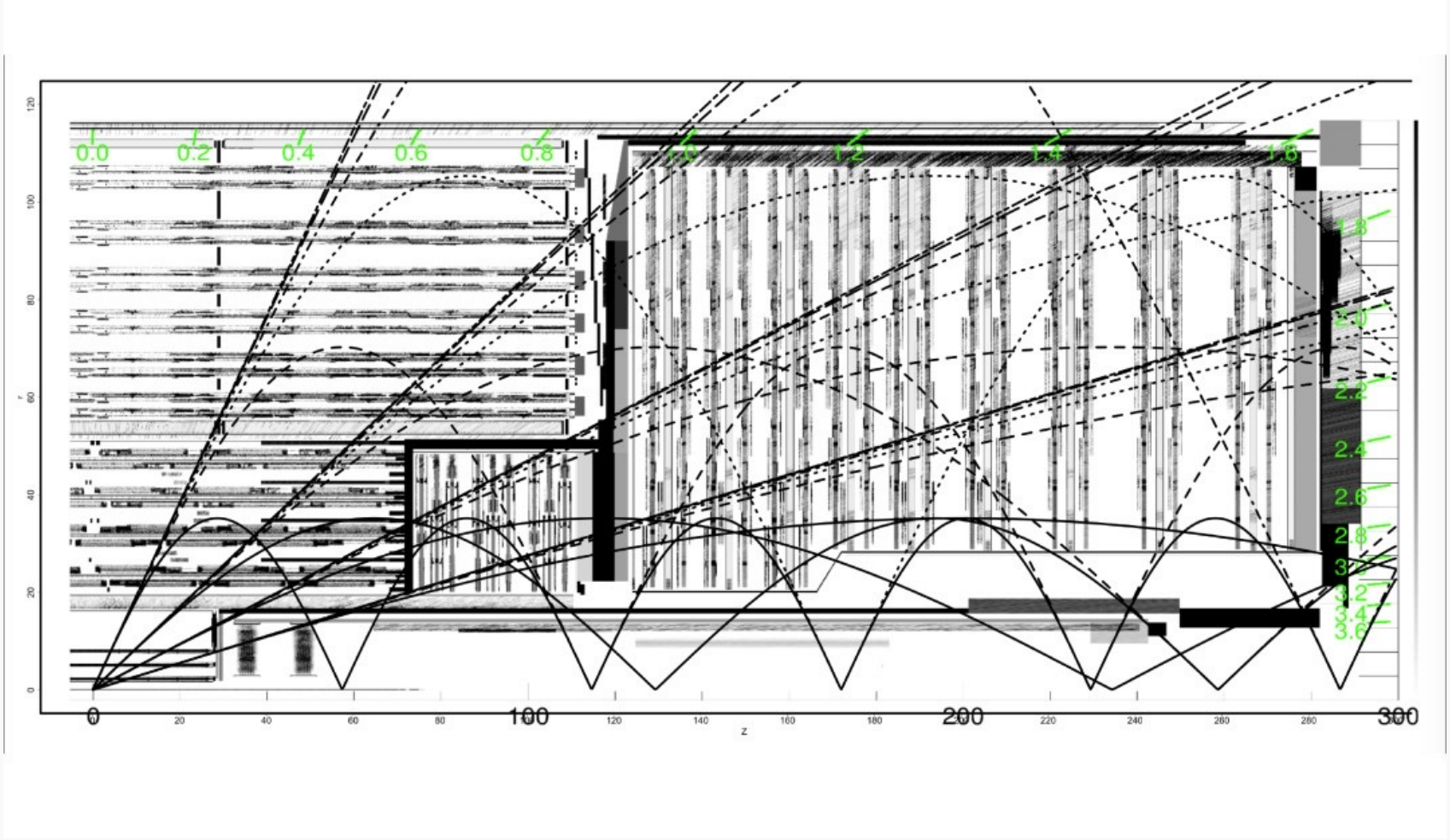




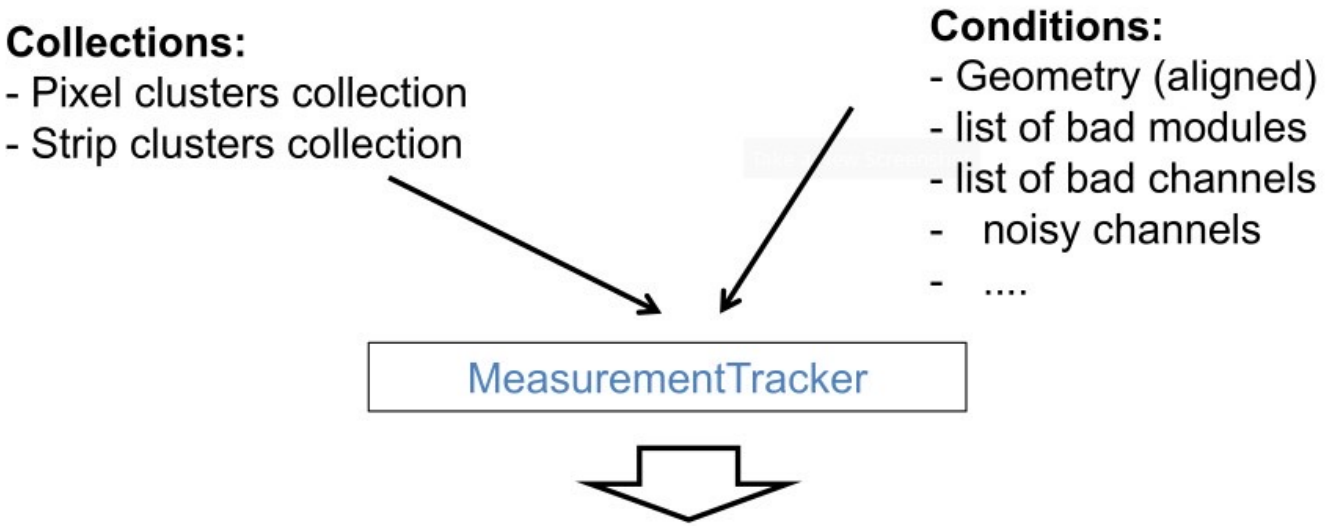
Connect each layer to any “next layer” with a possible line-of-sight

At run rime search stops as soon a trajectory is “well inside” one of the next-layers

“Real” trajectories in the real detector



MeasurementTracker



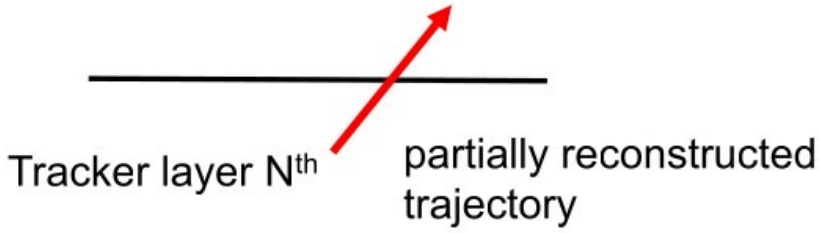
Everything cached, at a per-module level, in MeasurementDet instances.

Goal is to provide fast access to per-module information to the rest of the tracking code

Navigation

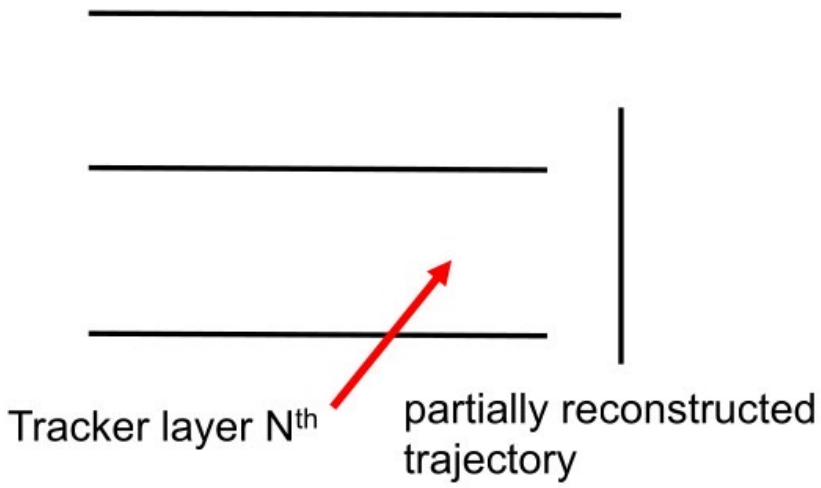
Trajectory builder: smallest logic unit

input: trajectory state on Layer N^{th},
i.e. `TrajectoryStateOnSurface`



Trajectory builder: smallest logic unit

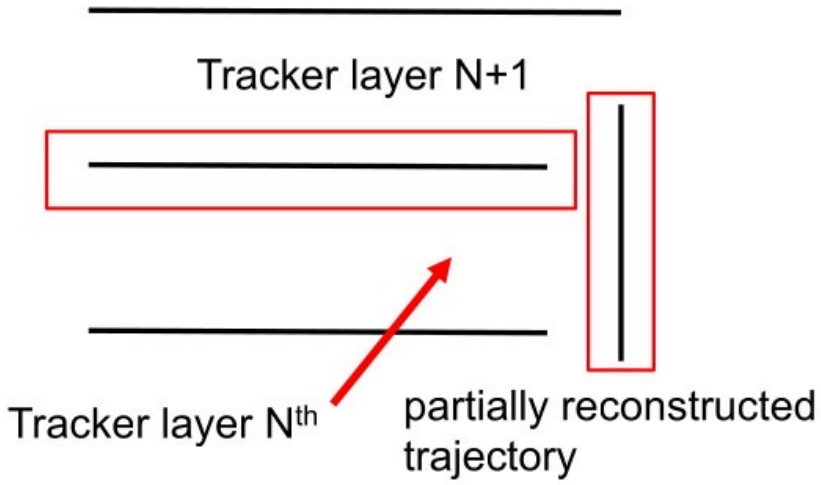
input: trajectory state on Layer N^{th}



1) find **next** compatible layer (or layers)

Trajectory builder: smallest logic unit

input: trajectory state on Layer Nth

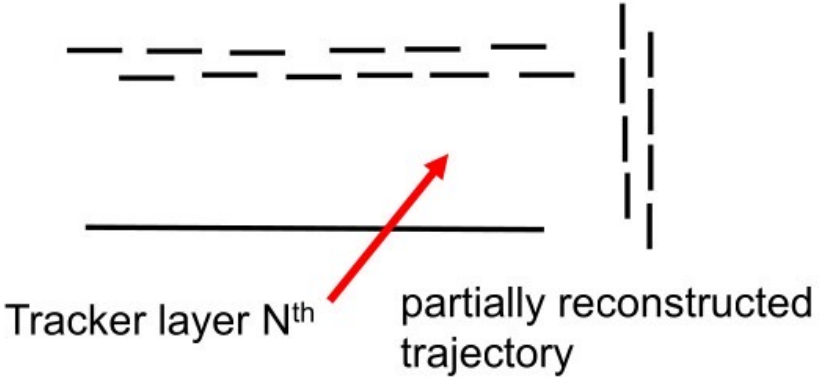


1) find **next** compatible layer (or layers)

Role of the "NavigationSchool"

Trajectory builder: smallest logic unit

input: trajectory state on Layer Nth

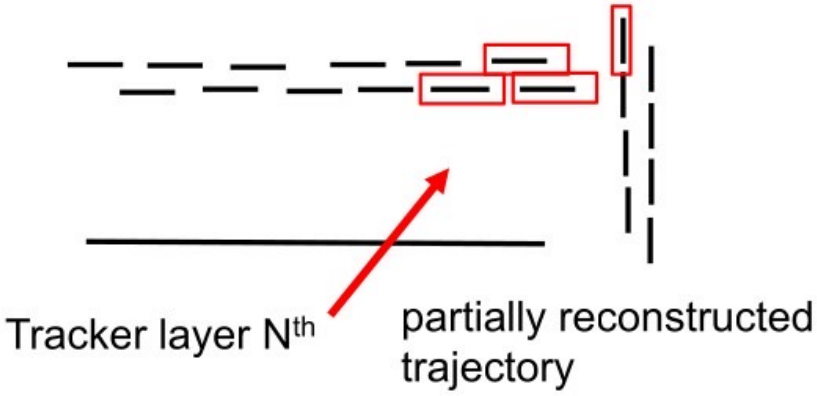


1) find **next** compatible layer (or layers)

1) find list of compatible detectors on layers N+1

Trajectory builder: smallest logic unit

input: trajectory state on Layer Nth



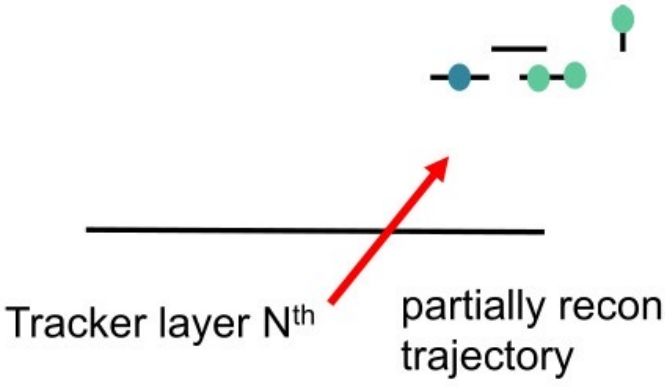
1) find **next** compatible layer (or layers)

1) find list of compatible detectors on layers N+1

Role of the “SearchDetGeometry”

Trajectory builder: smallest logic unit

input: trajectory state on Layer Nth



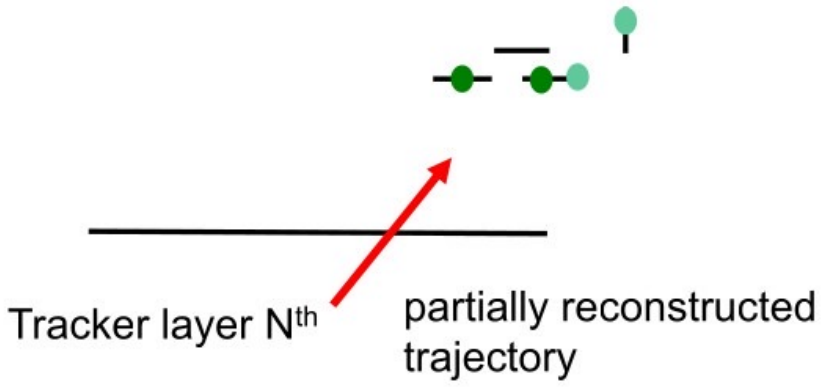
1) find **next** compatible layer (or layers)

1) find list of compatible detectors on layers N+1

1) find list of compatible hits on compatible detectors

Trajectory builder: smallest logic unit

input: trajectory state on Layer Nth



1) find **next** compatible layer (or layers)

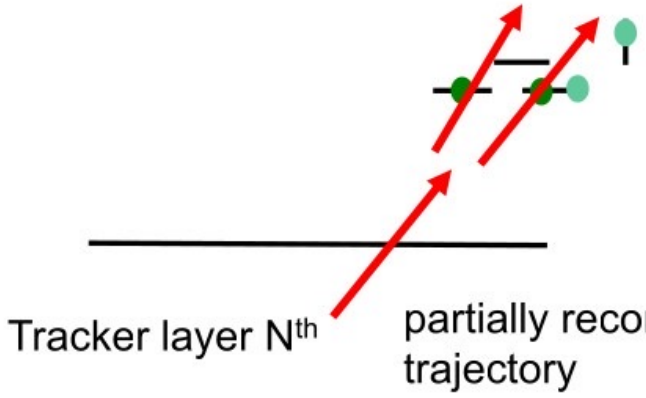
1) find list of compatible detectors on layers N+1

1) find list of compatible hits on compatible detectors

Role of the "MeasurementDet"

Trajectory builder: smallest logic unit

output: Two new states: they will be projected onto the next Tracker layer



input: trajectory state on Layer Nth

1) find **next** compatible layer (or layers)

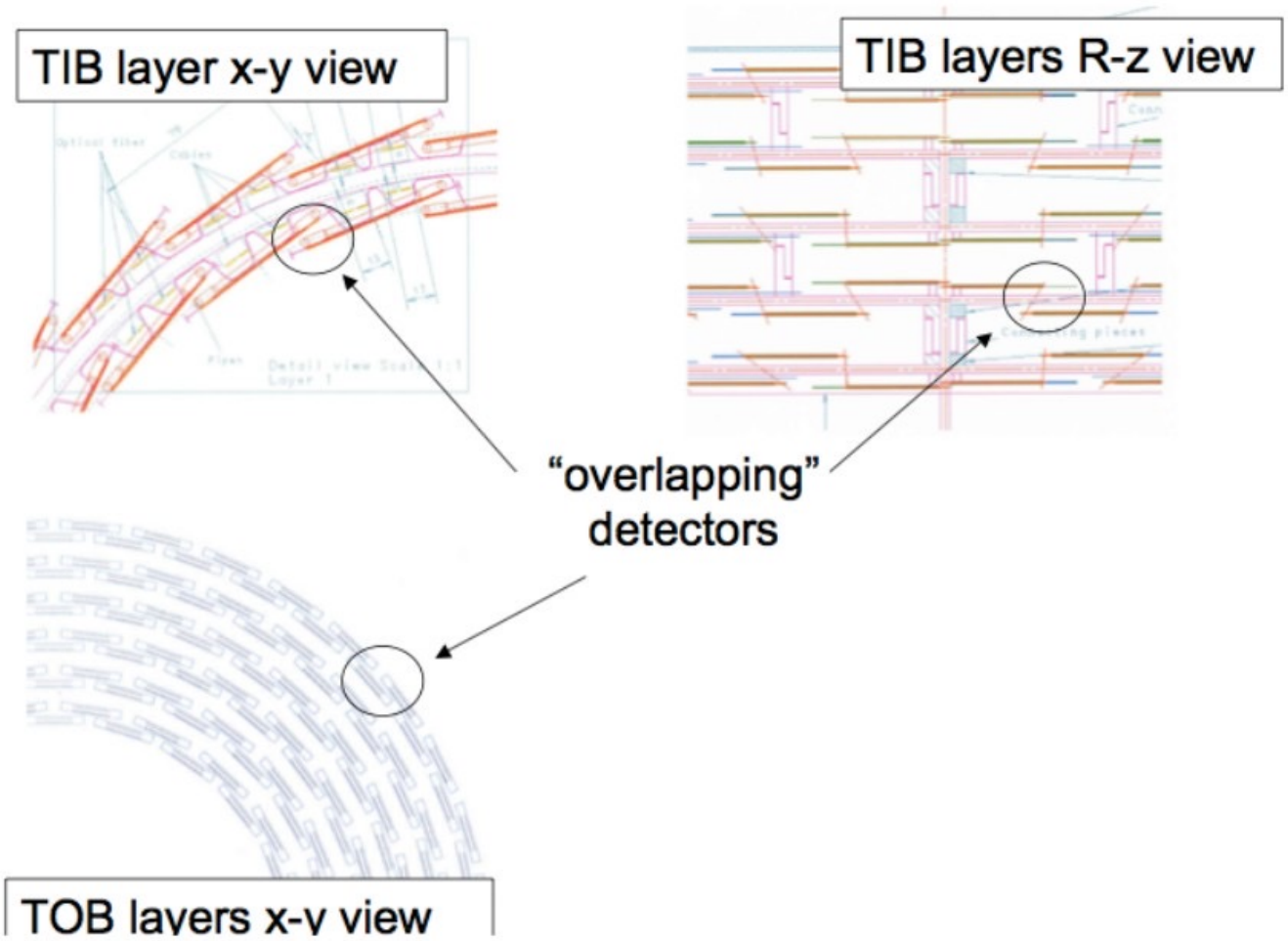
1) find list of compatible detectors on layers N+1

1) find list of compatible hits on compatible detectors

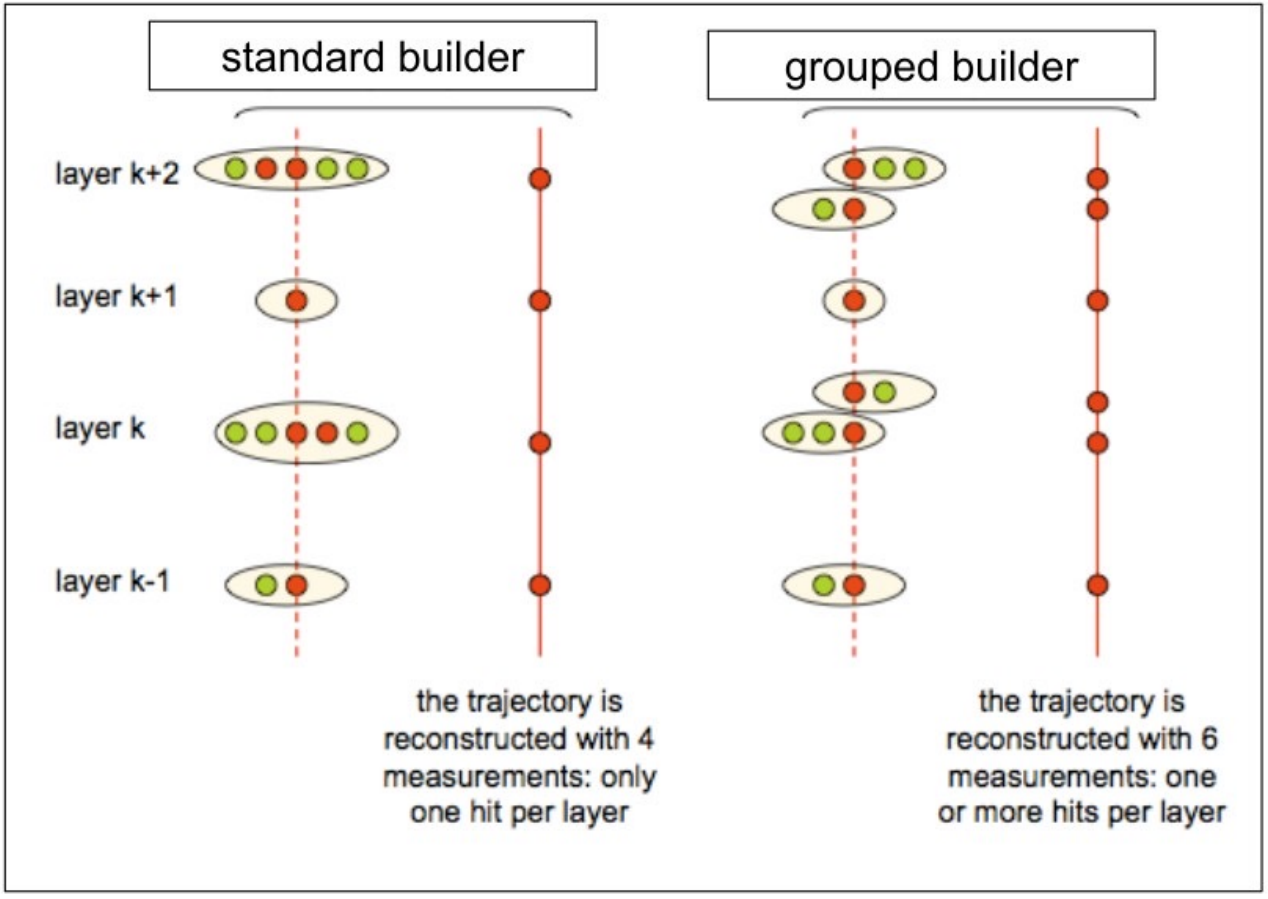
1) Create new trajectory with updated state (pt,lambda,phi,..) for each compatible hit

Role of the "KalmanUpdator(s)"

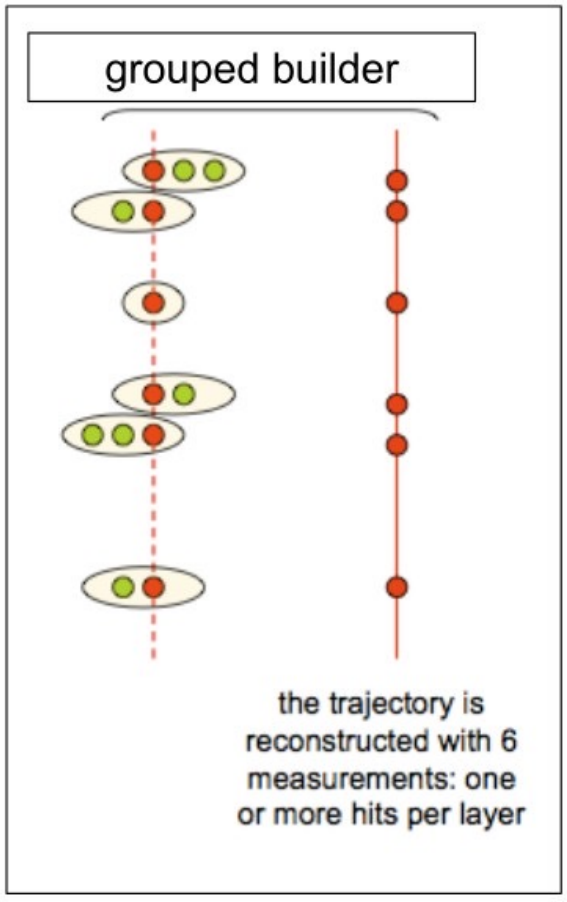
standard vs “grouped” builder



standard vs “grouped” builder

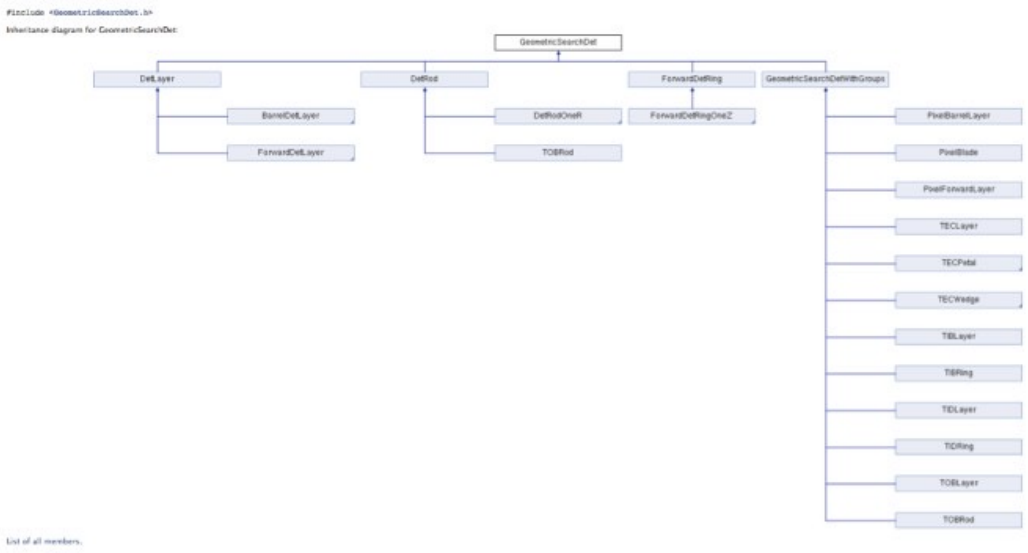


standard vs “grouped” builder



A fast sorting of measurements in groups of mutually exclusive hits is far from trivial.

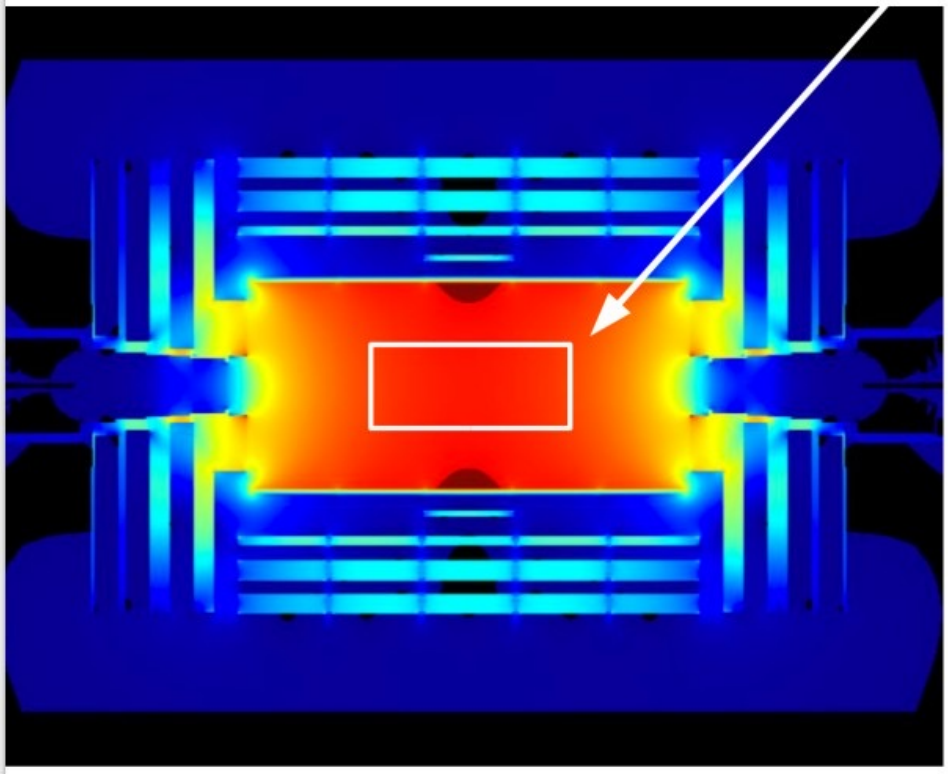
It is implemented through the hierarchy of classes of the `GeometricSearchDet tree`



Propagation

Magnetic field

Magnetic Field parametrization



$$
\begin{aligned}
B_z(0,z) &= \frac{1}{2} B_0 \sqrt{1+\bar{a}^2}\,[f(u)+f(v)] \\
u &= (1-\bar{z})/\bar{a} \\
v &= (1+\bar{z})/\bar{a} \\
f(x) &= x/\sqrt{1+x^2}
\end{aligned}
$$

a = radius, L = length.

$\bar{z} = 2z/L$, $\bar{a} = 2a/L$,

$$
\begin{aligned}
B_z(r,z) &= \sum_{\nu=0}^{\infty} \frac{(-1)^\nu}{(\nu!)^2} \frac{\partial^{2\nu}}{\partial z^{2\nu}} B_z(0,z) \left(\frac{r}{2}\right)^{2\nu} \\
B_r(r,z) &= \sum_{\nu=1}^{\infty} \frac{(-1)^\nu}{\nu!(\nu-1)!} \frac{\partial^{2\nu-1}}{\partial z^{2\nu-1}} B_z(0,z) \left(\frac{r}{2}\right)^{2\nu-1}
\end{aligned}
$$

Also a “parabolic” approximation used in HLT and early stages of tracking

Propagators

- Propagators are needed to extrapolated a trajectory from one surface (or space point) to another, taking into account
  - The magnetic field
  - Material effects including energy losses
- Tracking generally makes use of three propagators
  - AnalyticalPropgator
  - PropagatorWithMaterial
  - RungeKuttaPropagator

Propagators

AnalyticalPropagator

- Simply calculates analytically the intersection between a helix and with a detector plane or cylinder

PropagatorWithMaterial

- Builds on the AnalyticalPropagator but adds energy losses in material (Bethe-Bloch/Bethe-Heitler) and accounts for multiple scattering
- Used for most steps of the track reconstruction, except the final fit

RungeKutta

- Uses Runge-Kutta methods for iteratively solving first order differential equations (such as a particle in a B-filed)
- Can take into account non-uniformities in the magnetic field
- Used in the final track fit

- More precise but also slower going from top to bottom

Further propagators in CMS

- There are a few additional propagators used for special purposes in CMS

SteppingHelix

- Designed for volumes containing lots of material
- Used in the muon reconstruction to propagate through the calorimeters and the magnet yoke

GEANE propagator

- Propgator and fitter developed for GEANT3 in the early 90s
- Used in several previous experiments
- Very precise, but CPU intensive