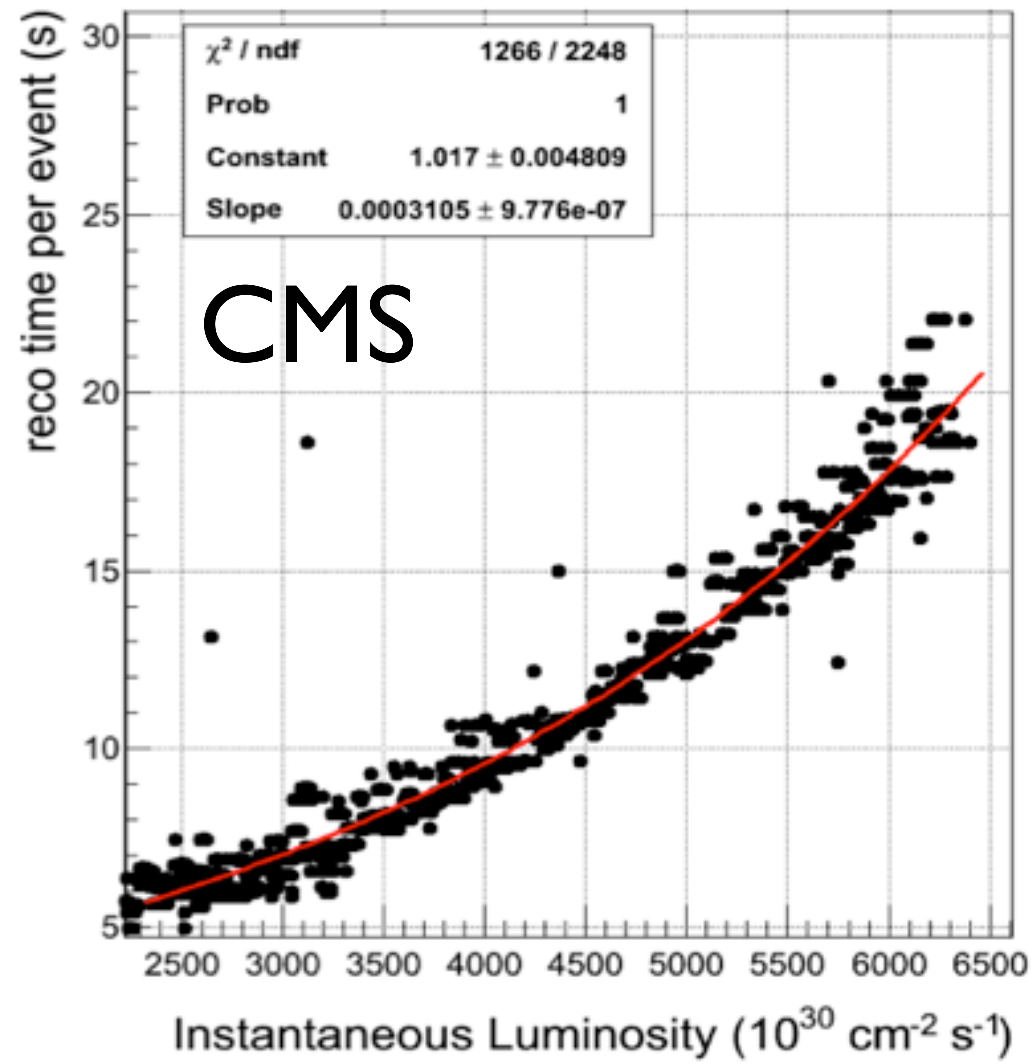
Status of Pile-Up tracking

G. Cappello, C. Civinini, L. Silvestris, A.Tricomi

AIDA 3rd Annual Meeting - Mar. 27th 2014

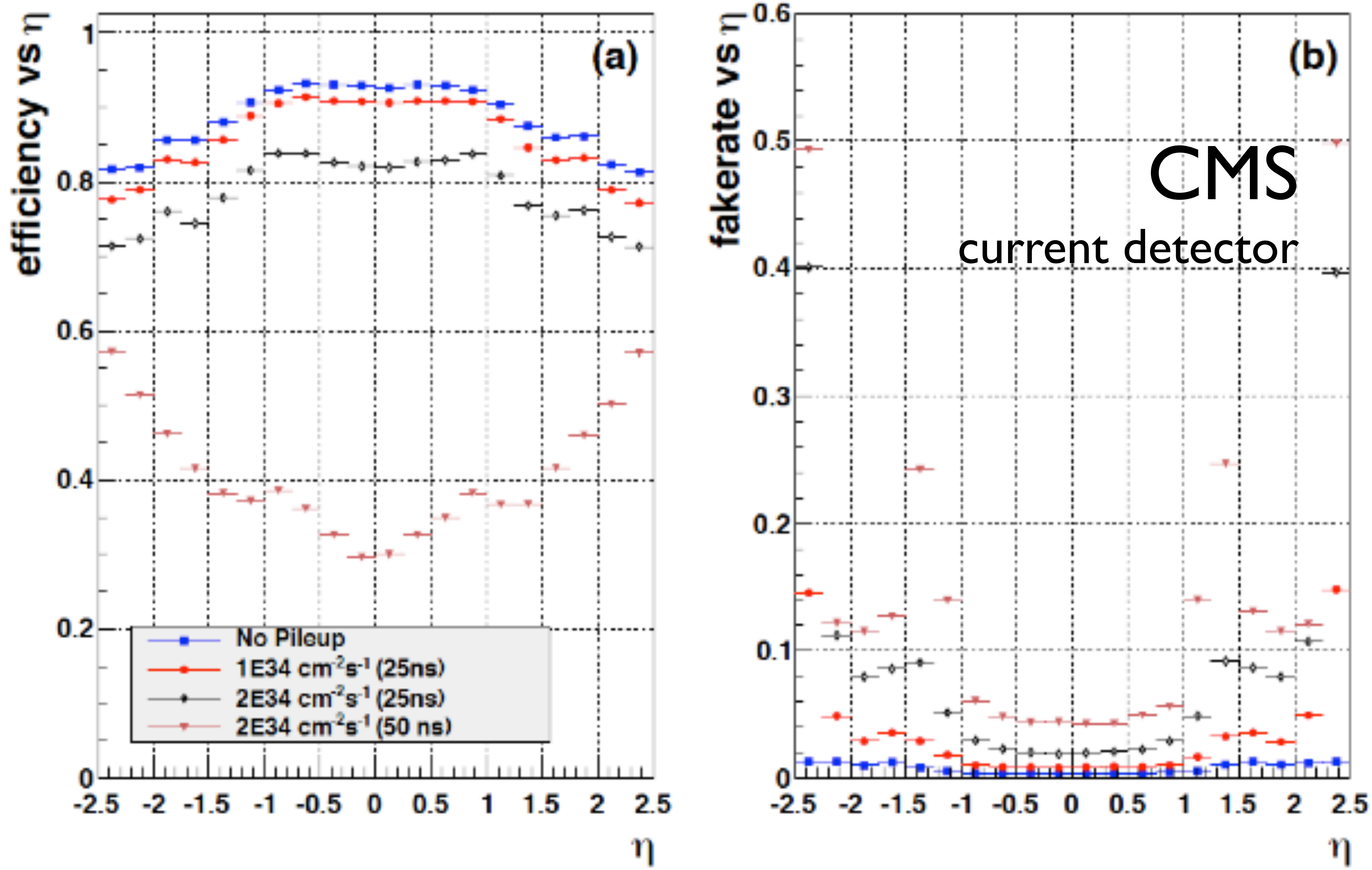
Handling the Pile-Up

Bunch spacing	Peak luminosity	Av. number of Pile-Up
25ns	1.0e34	25
25ns (low emit.)	2.0e34	40
50ns	2.0e34	100
50ns (low emit.)	2.5e34	140



- Rising of the number of hits
- Increasing of fake rate, lowering of efficiency in track reconstruction
- Tracking algorithms become the most CPU consuming task in HTL and offline reconstruction

Handling the Pile-Up



Tracking in CMSSW

Local Hit reconstruction

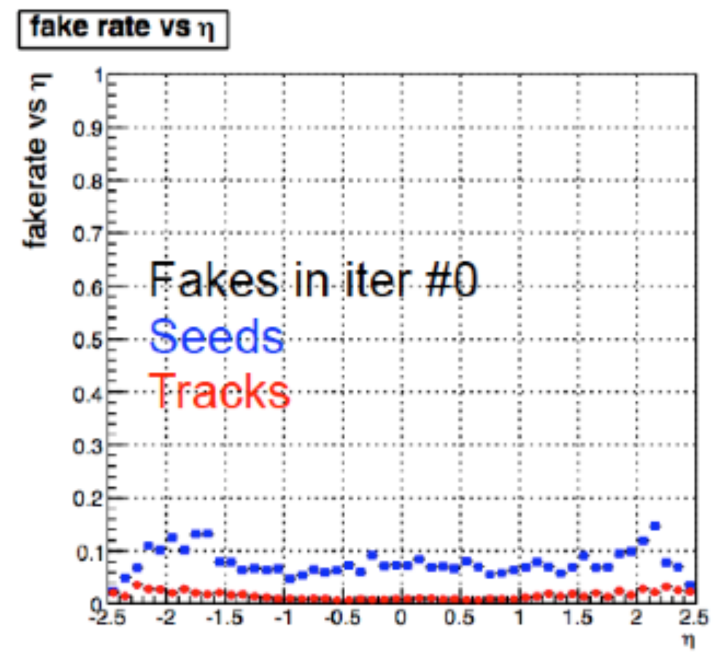
Iterative Tracking (IT)

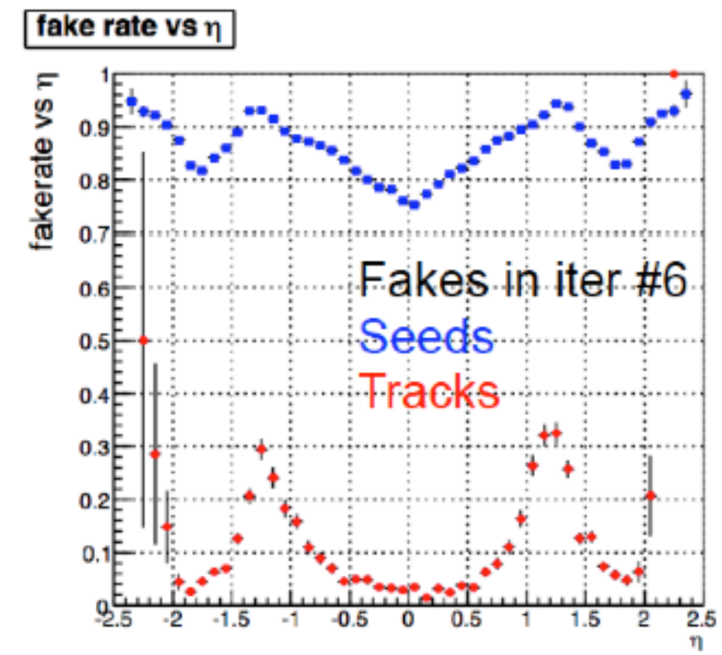
Iteration #0
Iteration #1
...
Iteration #n

(change seeding and track selection cuts, remove used hits at every iteration)

Combinatorial track finding (CTF)

Seeding
Track finding
Track fitting
Final selection





CMS
current detector

Requirement of tracking at High Pile-Up

- Reduce the number of seeds to be propagated to the outer layers
  - Seeds with more hits:
    - with high purity (smart association algorithms)
    - reduction of building time without affecting the efficiency

philosophy: spend a little more time in the seeding step (w.r.t. triplet or doublet seeds generation), but gain very much in the track building

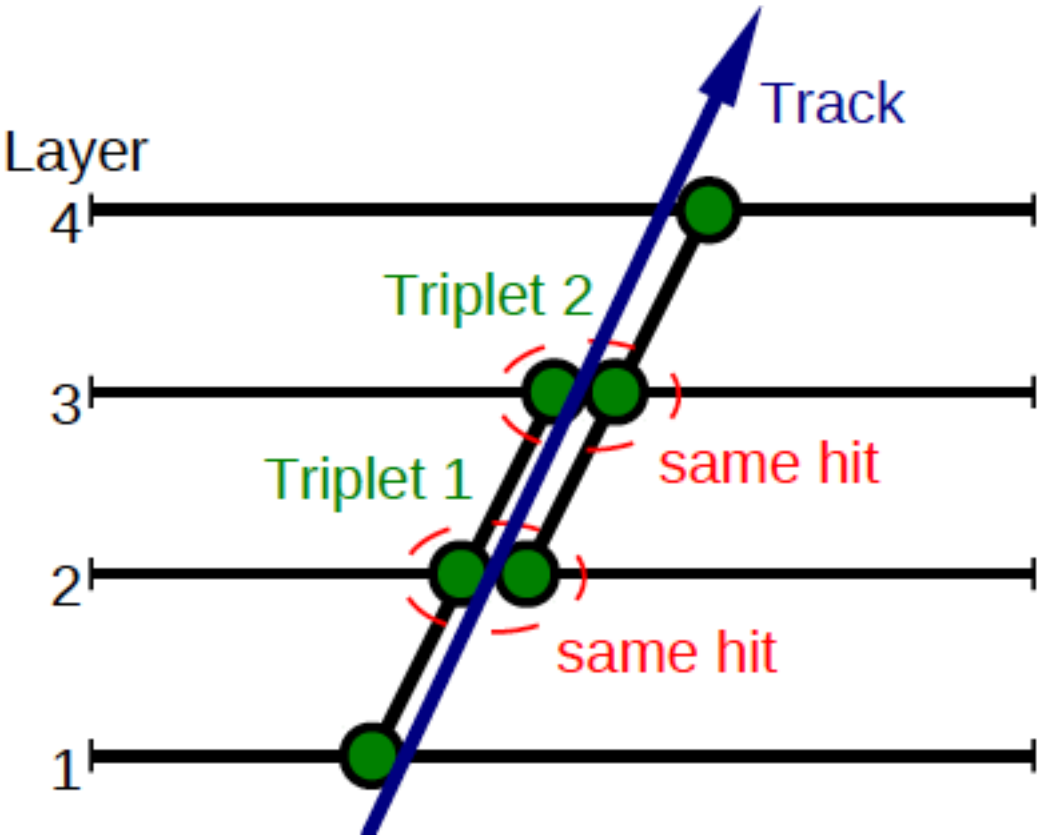
- Re-engineering of the seeding code
  - Tuning of the seed building criteria
  - Development of new seeding algorithms (e.g. Callular Automata based)
  - Code re-engineering (e.g. GPU usage...)

Cellular Automata approach to seeding

In Cellular Automata (CA), a grid of cells evolves in time from an initial state according to some rules and depending only from the values of the cell neighbors.

CA has been applied to the tracking problem in the past.
What's different:

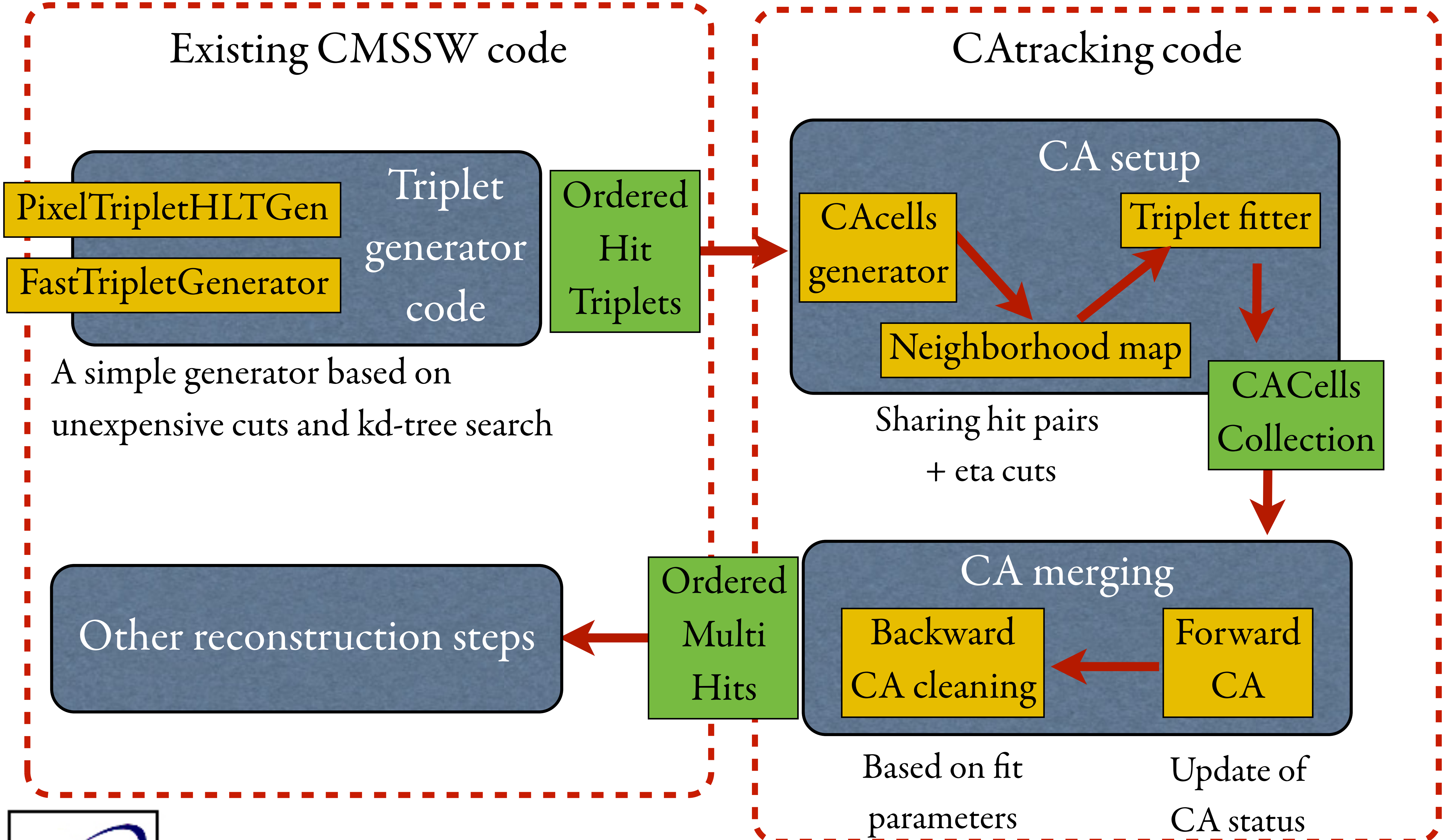
- Cell = A triplet instead of segments (reduction of the cells)
- Neighborhood conditions = pair of hits in common and close track parameters
- CA applied to the Seeding step only (less detector dependent)



Combine triplets from multiple layers using the cellular automata technique:

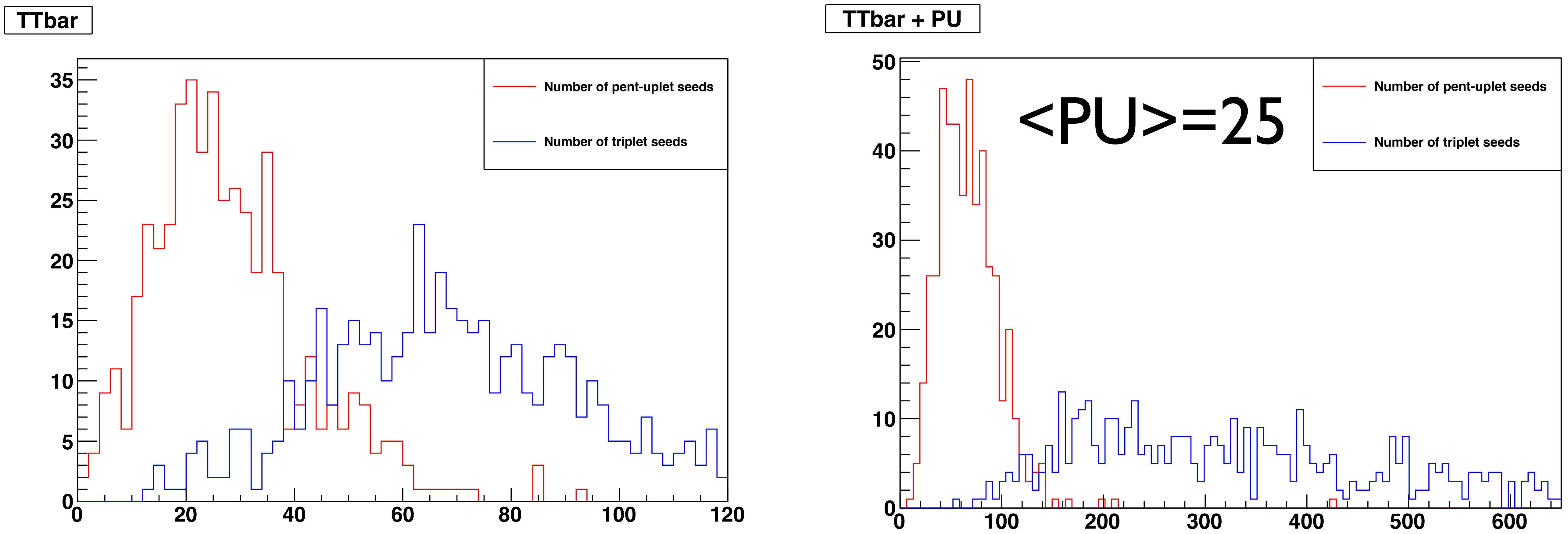
- Produce triplets
- Define a neighborhood map
- Fit triplets (only those with neighbors)
- join triplets into longer seeds (e.g. 5 hits or pent-uplets)

CASeedGenerator (CMSSW version)



CASeedGenerator (CMSSW version)

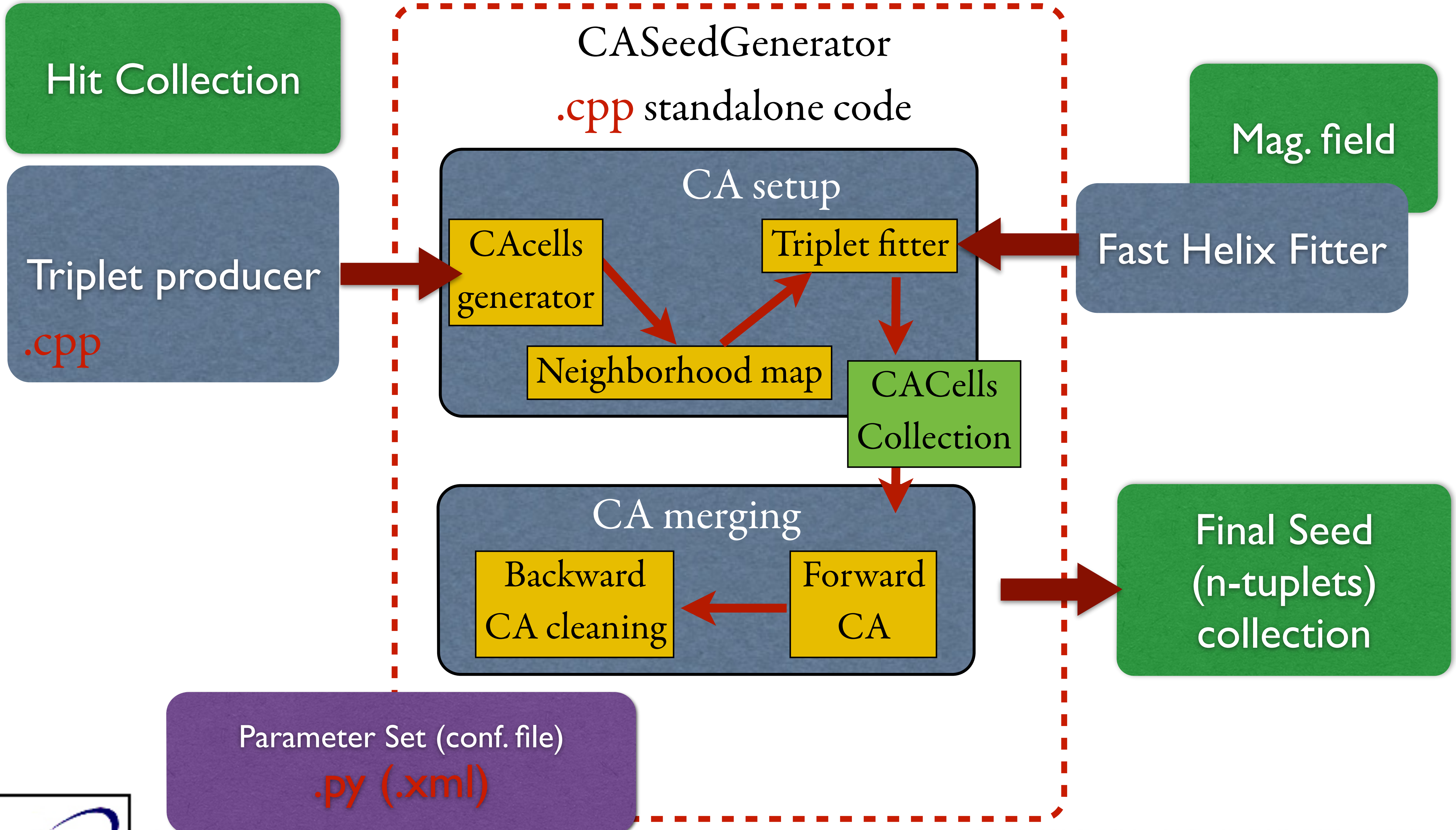
The output is a collection of 'pent-uplets' (multiplets with 5 hits), to be used for seeding



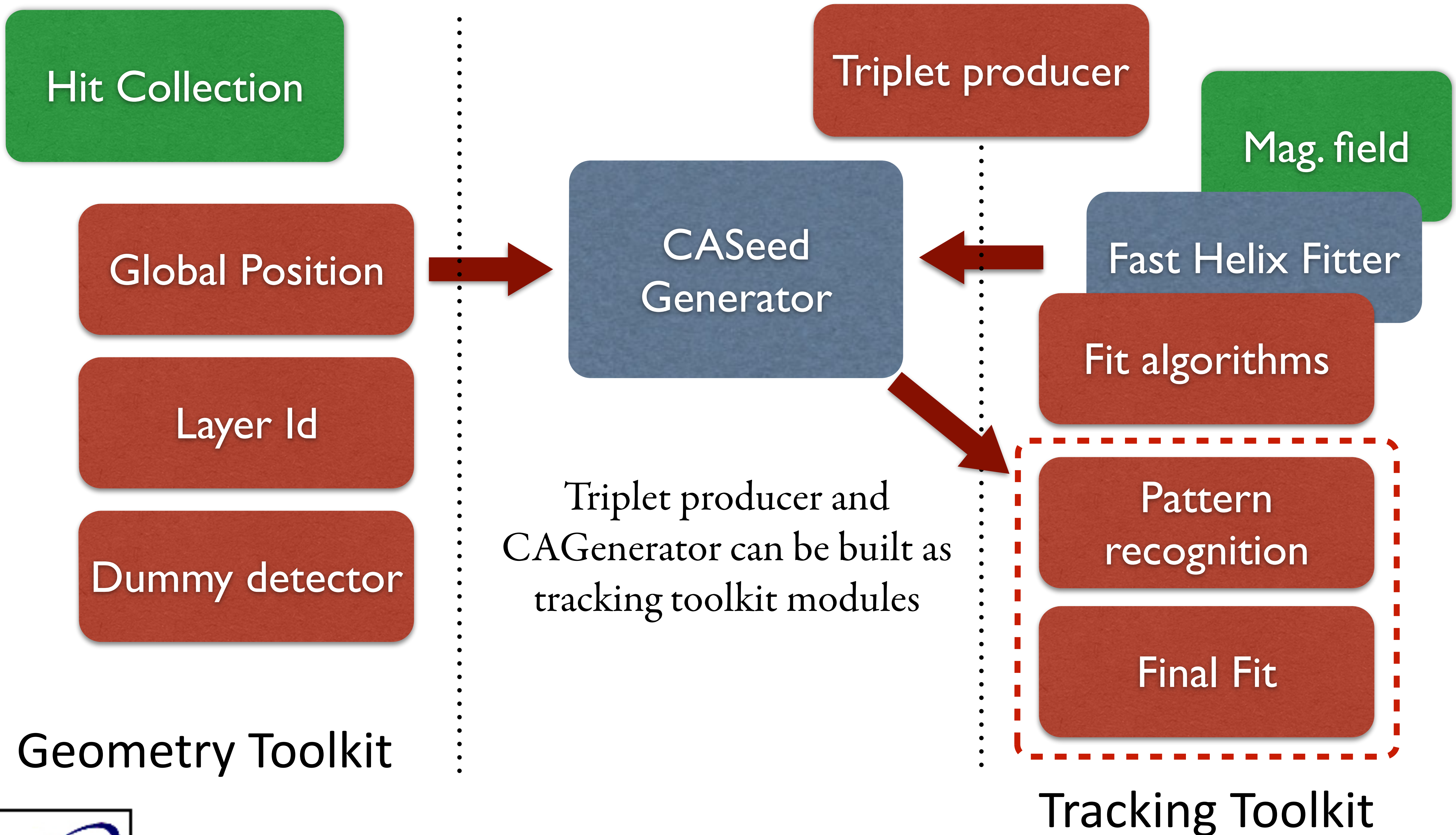
Reduction of the number of seeds

We expect 'cleaner seeds', hence a time reduction in track building (work in progress)

How to package the code



Interplay with other tools from WP2



Documentation

- Two different tags of the code can be viewed in the following repositories:

  Version 1.0 (working as a standard CMSSW EDProducer, more portable)

  Triplet producer:

  https://github.com/gigicap/cmssw/tree/from-CMSSW_6_2_0_pre8/Triplets/TripProducer

  CASeedGenerator:

  https://github.com/gigicap/cmssw/tree/from-CMSSW_6_2_0_pre8/CAtracker/CAtracker

  Version 2.0 (rebuilt as a CMSSW standard SeedGenerator, more CMSSW frindly)

  CASeedGenerator:

  https://github.com/gigicap/cmssw/tree/from-CMSSW_7_0_0_pre3/CAtracker/CAtracker

- As soon as the standalone .cpp + .py will be ready and integrated with other WP2 tool, it will be moved into the AIDA code repository.
- Deliverable report

  http://cds.cern.ch/record/1610541

- Running prototype report

  http://cds.cern.ch/record/1664548

Timeline

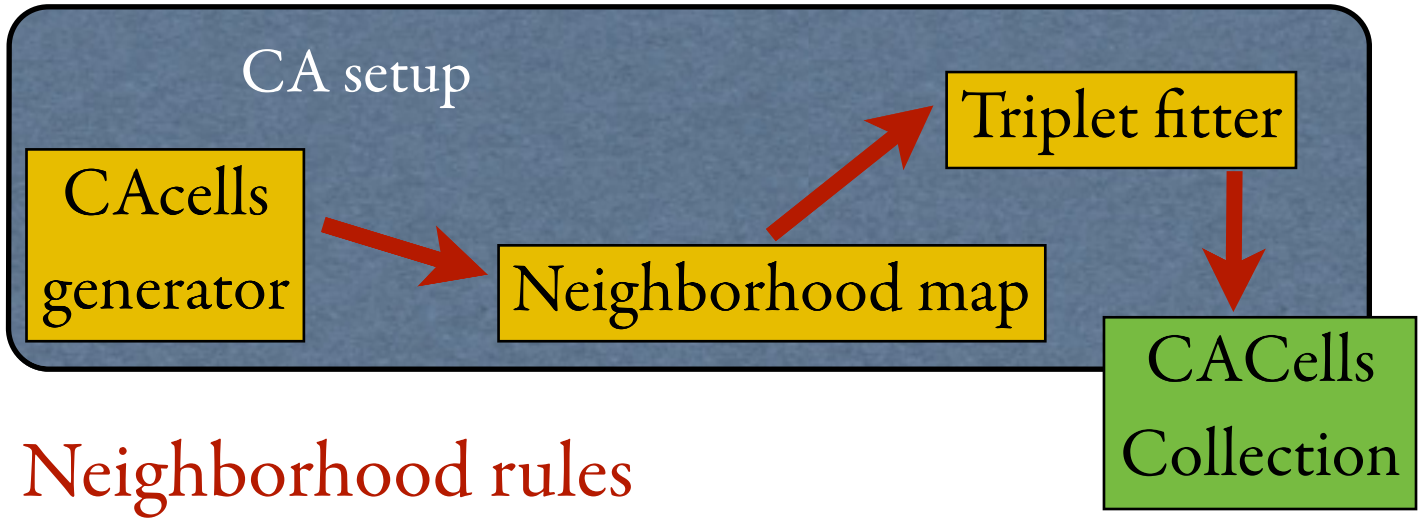
- The algorithm is completely developed and works within the CMSSW software
- It has been built in such a way to be easily exported as a standalone code
- Given a 'fake geometry' to run with and the support of geometry and tracking tools, we plan to deliver a working code inside the AIDA framework within May 2014
- Next step: optimize the code inside the framework (2nd half of 2014)

Open issues

- The algorithm is completely developed and works within the CMSSW software
- It is still under optimization
  - Neighborhood selection, backward selection cuts, seed length…
- A development of the algorithm in OpenCL (for GPU applications) is now on going
- Other algorithms (e.g. Hough transform) can be investigated
  - Pile-Up toolkit?

Backup

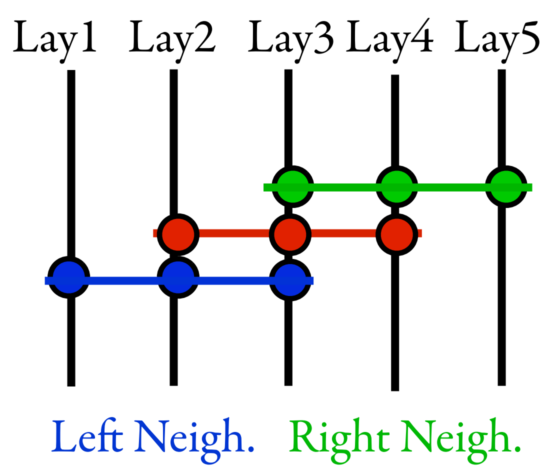
CAsetup



Neighborhood rules

Two cells (triplets) are considered neighbors if:

- They share two hits:



The neighborhood map structure is built in such a way that it can be filled in a fast way (only one loop on the triplet collection is needed). At the end of this step, each CAcell will contain a list of pointers to left and right neighbors and a neighborly parameter.

- They have similar Eta

The CA tracking can be thought as a maximum problem within the set U of connected triplets of the function:

$$F(\mathrm{U}) = \mathrm{N} - \sum_{i=1}^{p} \alpha_i \sum_{j=1}^{N-1} \phi(t_{j+1}, t_j)$$

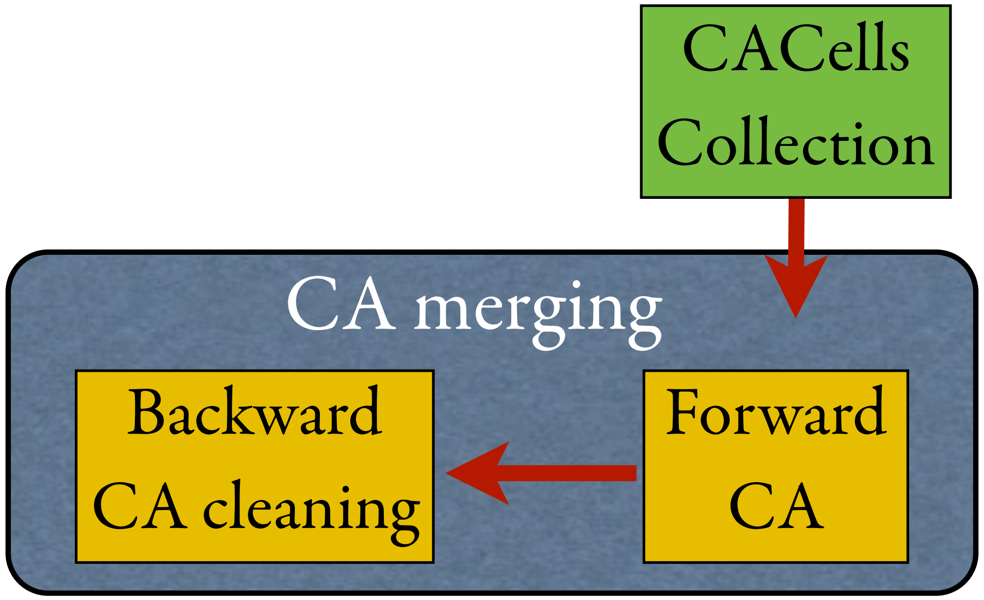
Minimize some 'continuity variables'

e.g: $\min |p_T(t_{j+1} - t_j)|$

Backward CA cleaning

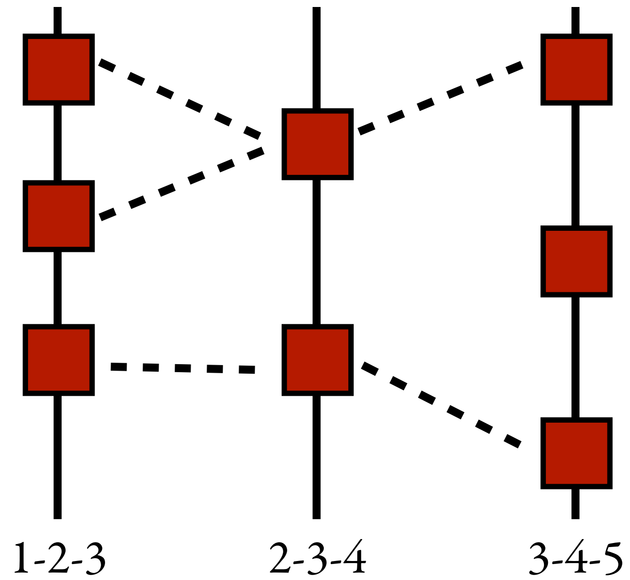
Maximize the length of the 'multiplet'

Forward CA



Forward CA

Step 0: the CAstatus of the cells is set to 1

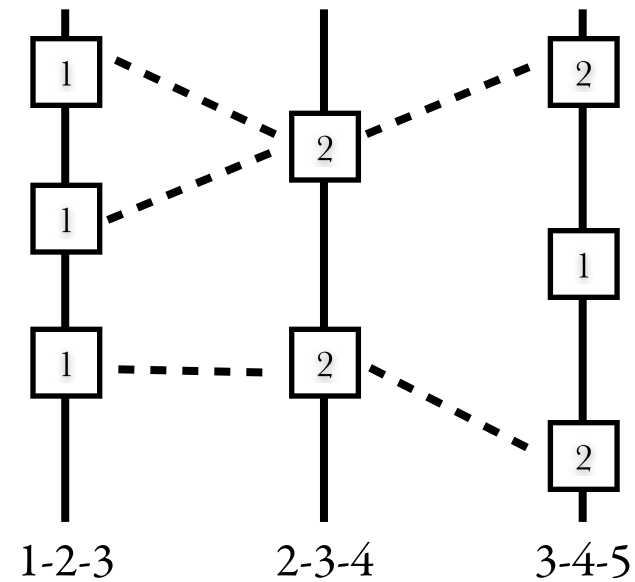


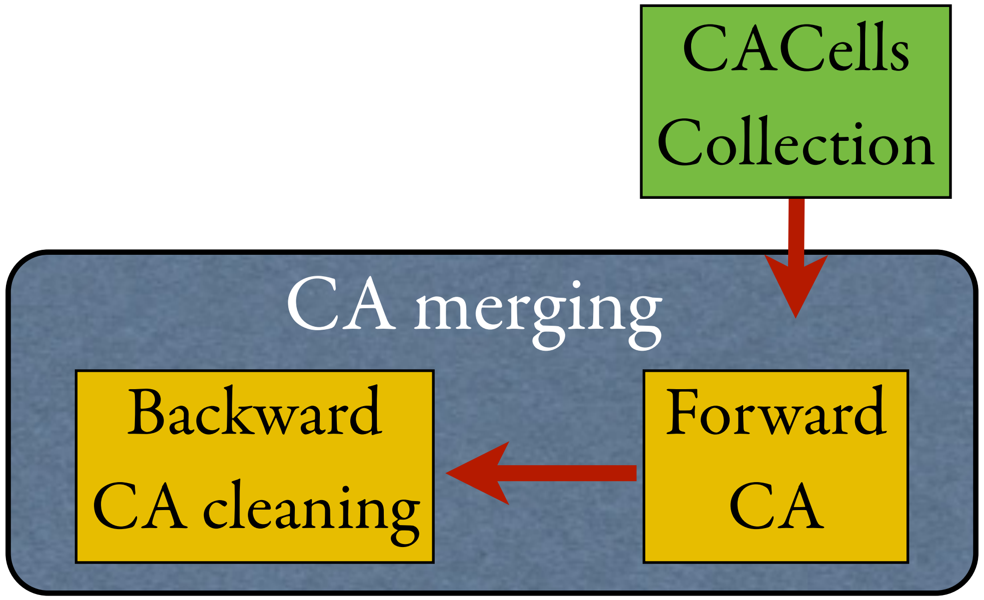
■ = Cell (triplet)

— = Layer configuration

- - - = Neighborhood map

Step >0: increase (at the same time) the CAstatus of the cells by one if a cell has at least a left neighbor with the same status





Backward CA

Starting from the candidates with maximum ending status, move backwards, choosing the left neighbors that minimize the pT difference.

NB: CA is a local algorithm (no global conditions on track candidates are considered)