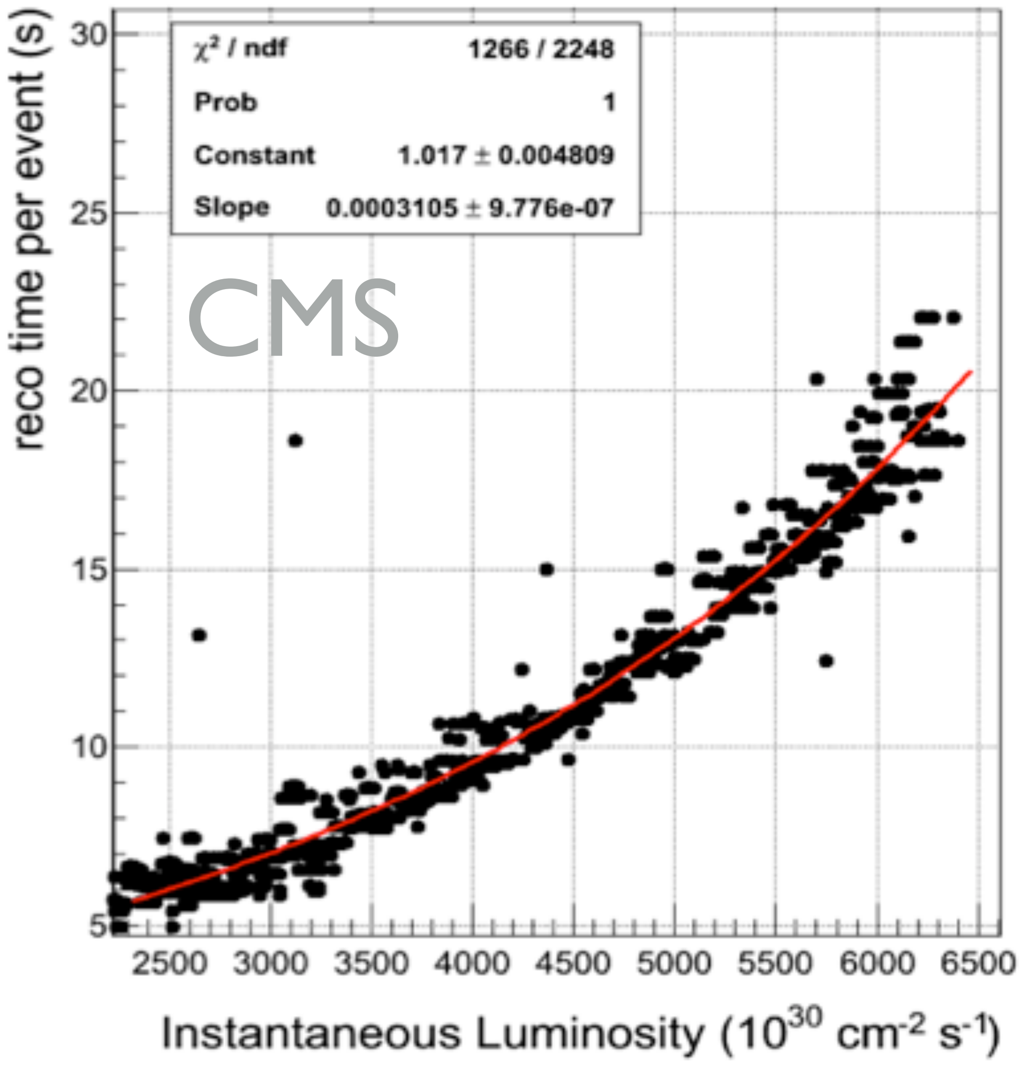
Tracking tool for Pile-Up

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

AIDA final Meeting - Dec. 10th 2014

Handling the Pile-Up

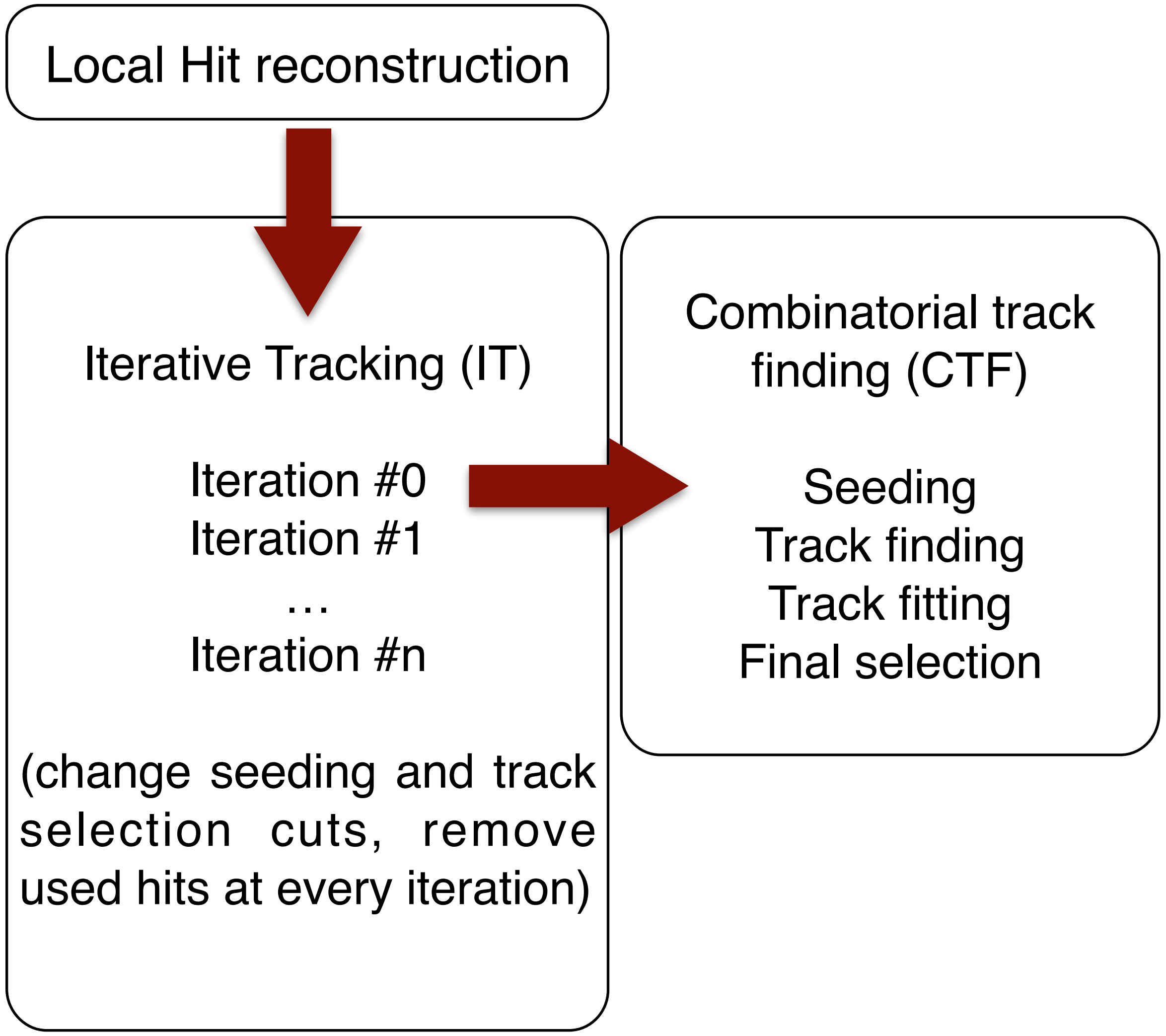
Bunch spacing	Peak luminosity	Av. Pile-Up
25ns	1.0e34	25
25ns (low emit.)	2.0e34	40
50ns	2.0e34	100
50ns (low emit.)	2.5e34	140



- 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

Tracking in CMSSW at High Pile-Up

The CMS tracking philosophy is the same as most of the tracking detectors



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

- Reduction of the number of seeds to be propagated to the outer layers

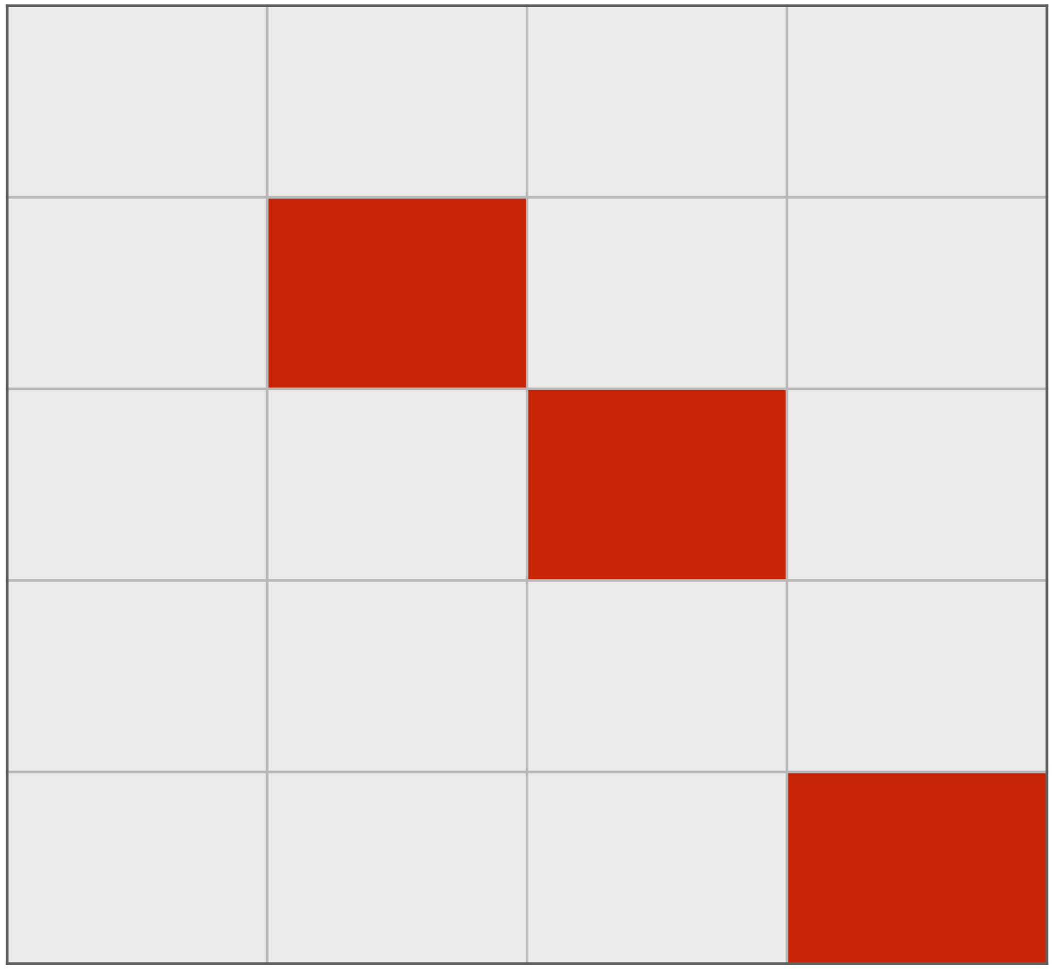
philosophy: spend a little more time in the seeding step (w.r.t. triplet or doublet seeds generation), but gain in the track building .

- Seeds with more hits:
  - with high purity (smart association algorithms);
  - reduction of building time.

Development of a new seeding algorithm based on Cellular Automaton

Cellular Automaton tracking

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 neighbours.



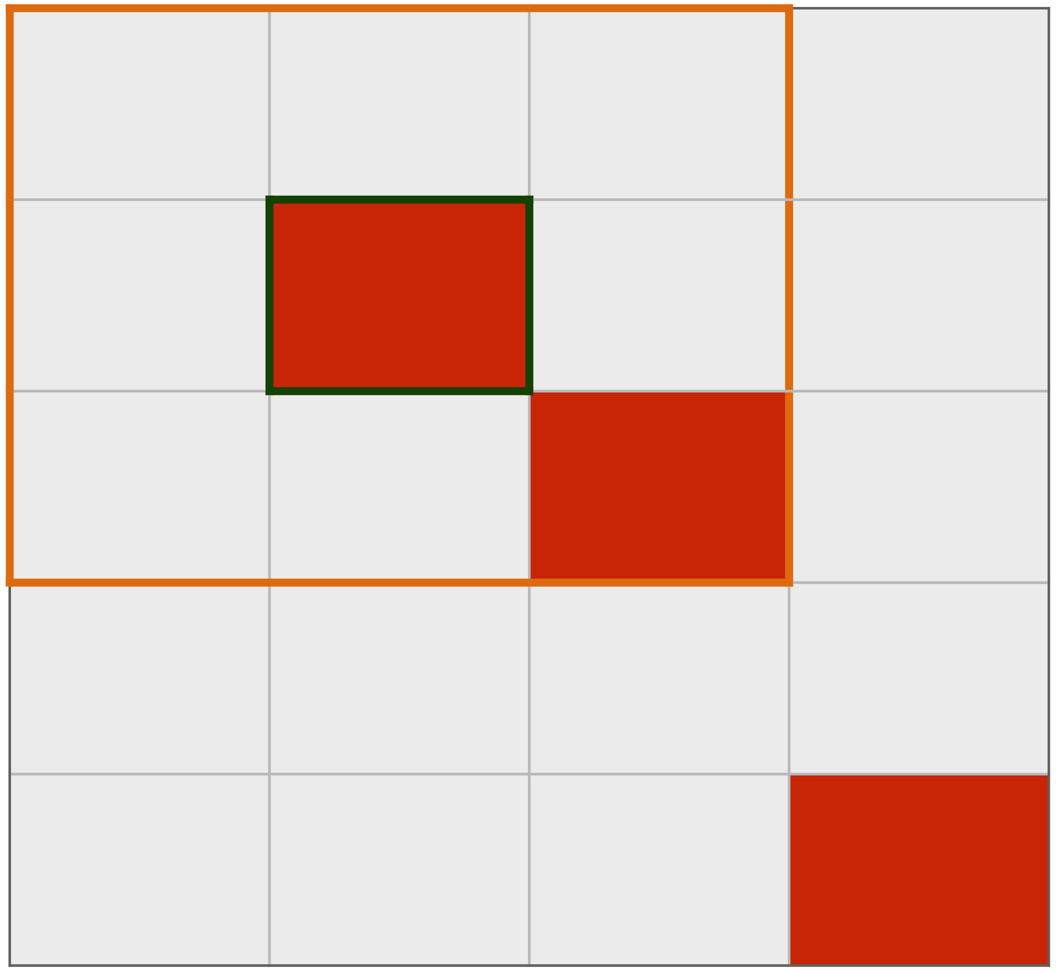
Example

Rules: if cell is **on** and has at least one neighbour on then it turns off. If is off and has at least one neighbour on then it turns **on.**

step 0: CA setup
set the initial status of the cells

Cellular Automaton tracking

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 neighbours.



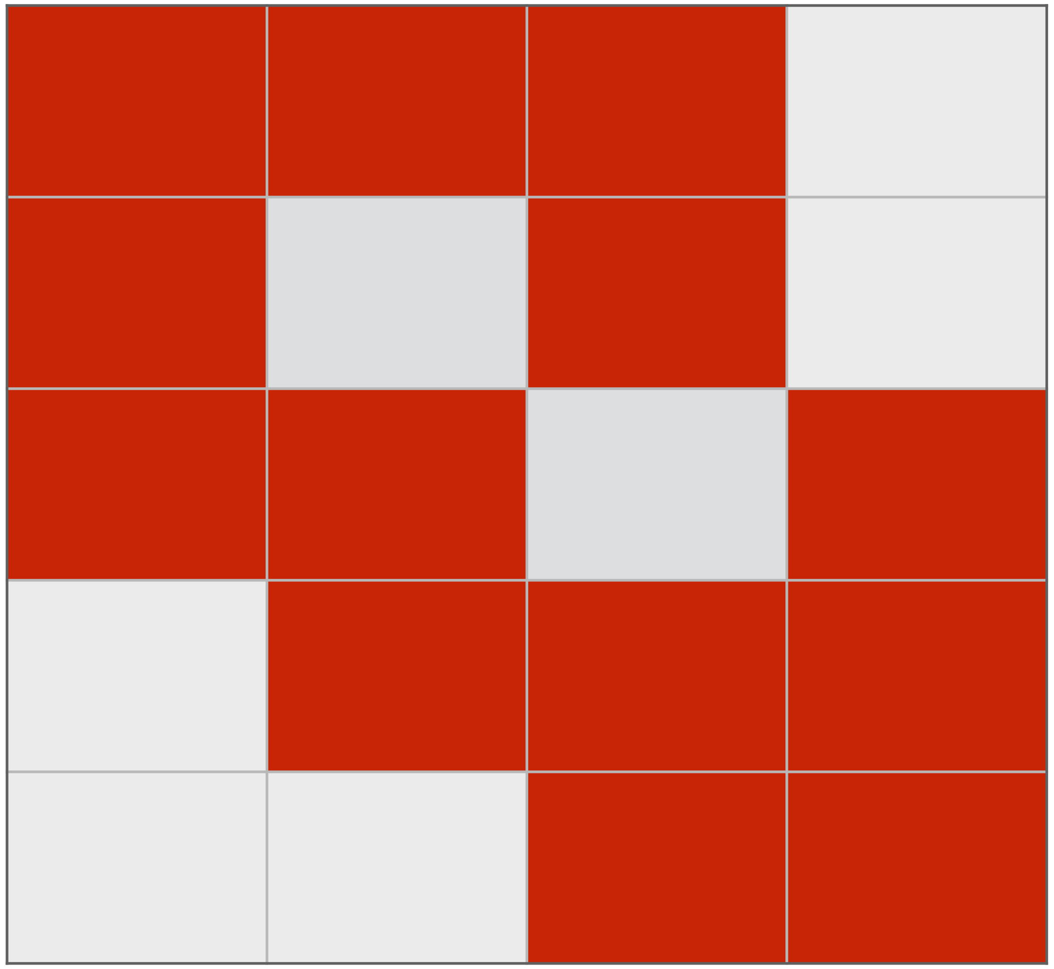
Example

Rules: if cell is **on** and has at least one neighbour on then it turns off. If is off and has at least one neighbour on then it turns **on**.

step 0: CA setup
set the neighbourhood rules

Cellular Automaton tracking

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 neighbours.



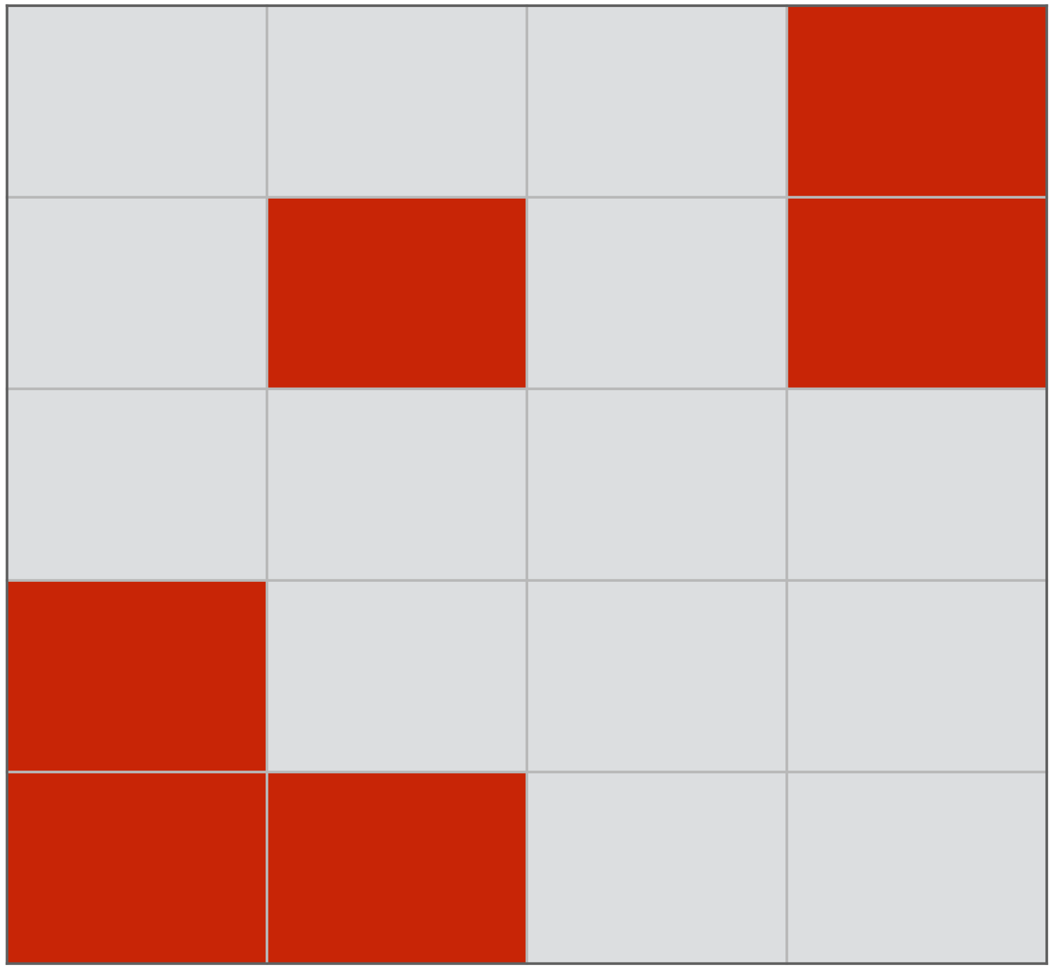
Example

Rules: if cell is **on** and has at least one neighbour on then it turns off. If is off and has at least one neighbour on then it turns **on.**

step 1

Cellular Automaton tracking

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 neighbours.



Example

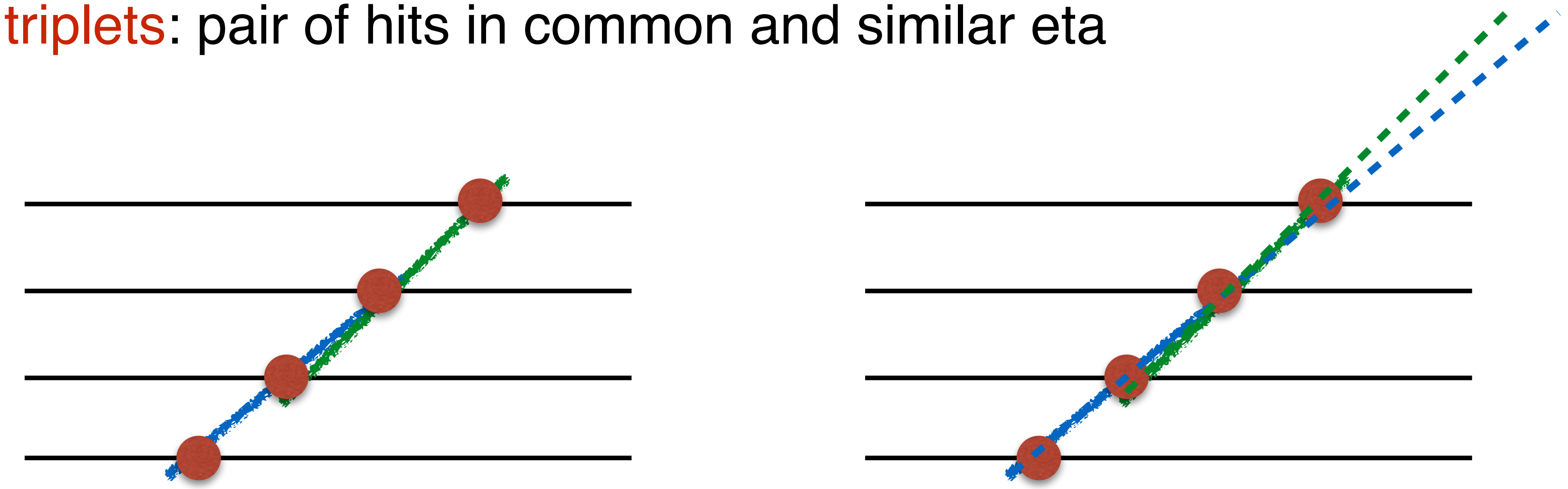
Rules: if cell is **on** and has at least one neighbour on then it turns off. If is off and has at least one neighbour on then it turns **on.**

step 2

Cellular Automata approach to seeding

CA can applied to the tracking problem and in particular to seeding.

- Cell: A segment defined by two hits or a triplet (more suitable for our purpose)
- Neighbourhood rules:
  - If cells are pairs: sharing one hit and similar eta
  - If cells are triplets: pair of hits in common and similar eta

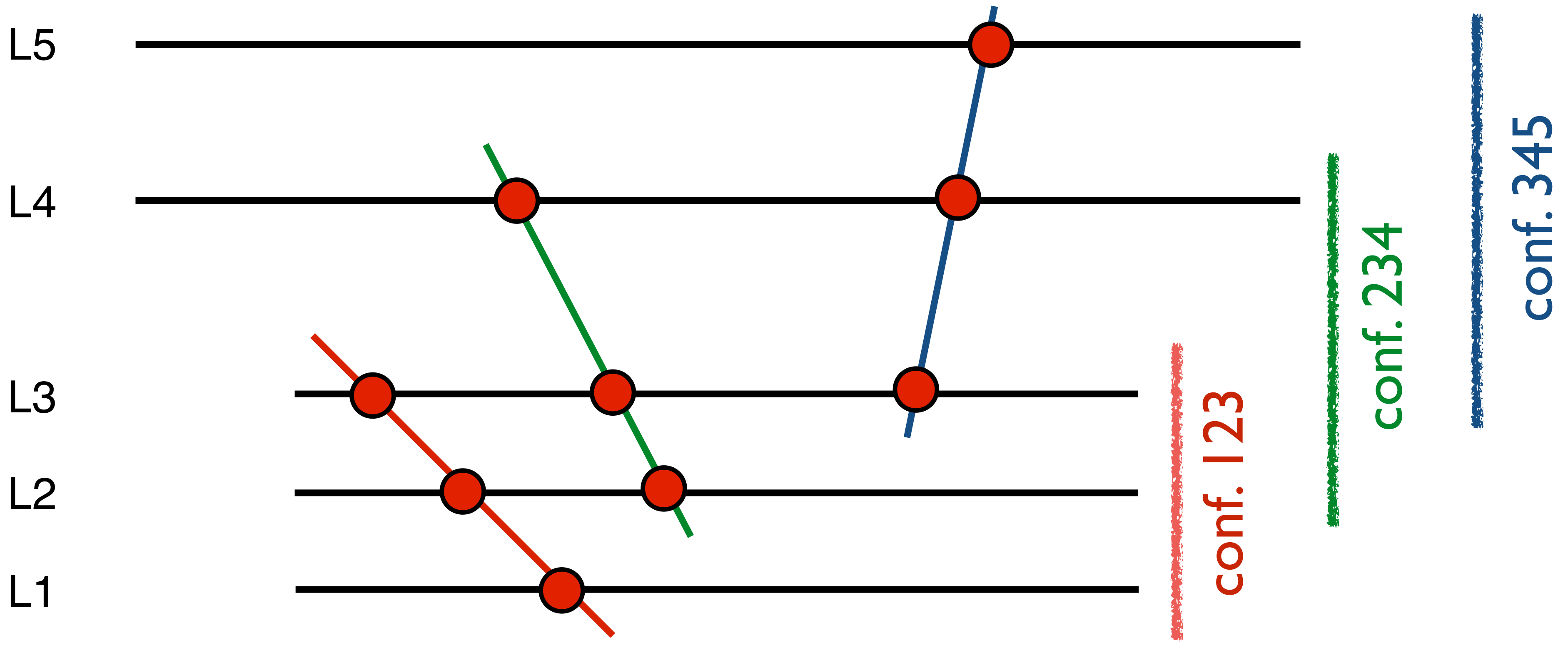


Combine triplets from multiple layers using the cellular automata technique:

- Produce triplets
- Define a neighbourhood map
- Fit triplets (only those with neighbours)
- join triplets into longer seeds (e.g. 4 or 5 hits)

Cellular Automata approach to seeding

Cells (triplets or doublets) are built in different layer configurations
Neighbourhood map is built taking the layer configs into account



The possible layer configurations depend on the particular detector

Cellular Automata approach to seeding

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

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

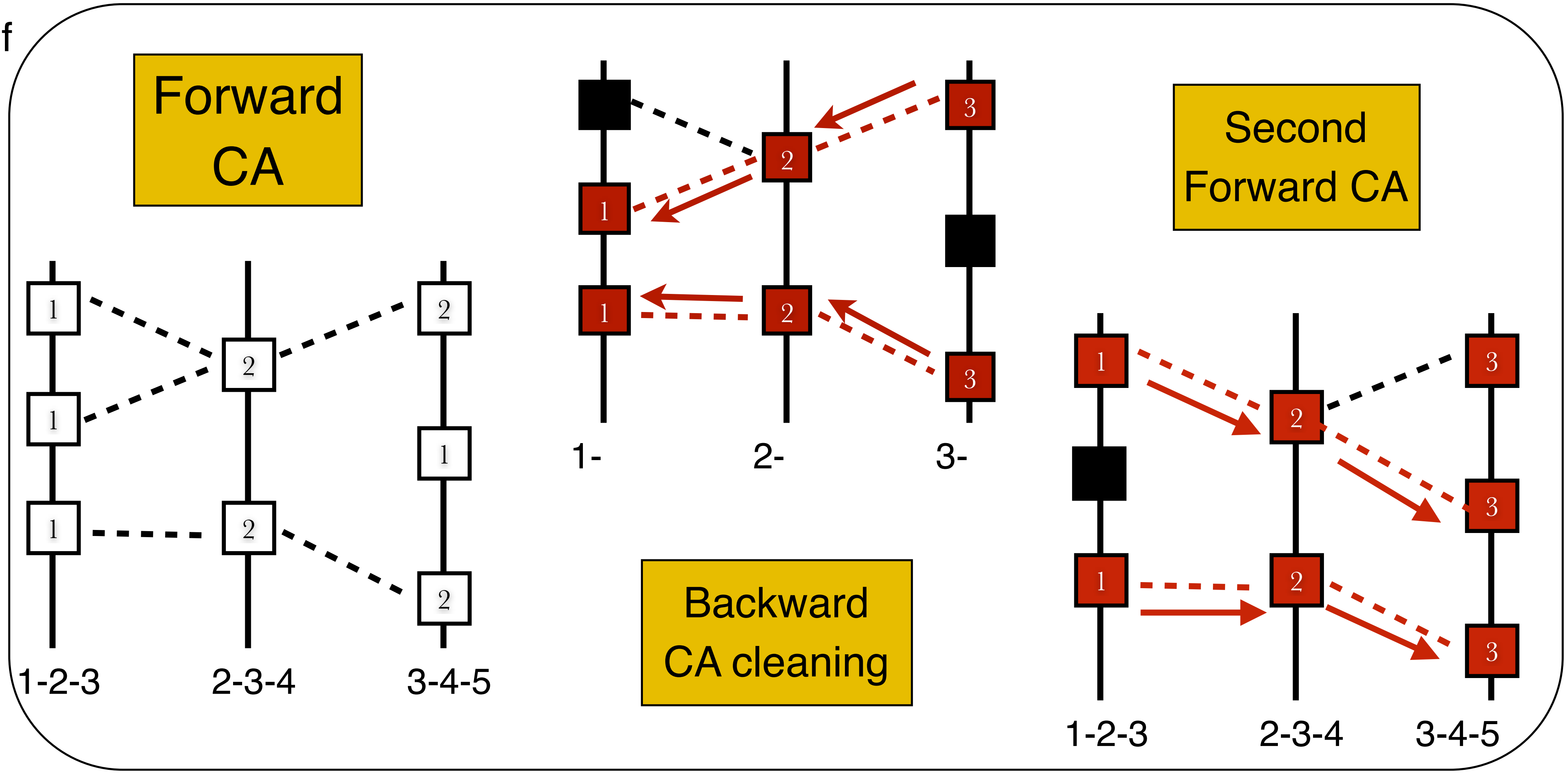
‘quality’ parameters (e.g. estimated tp)

Backward CA cleaning

Second Forward CA

Maximize the length of the ‘multiplet’

Forward CA

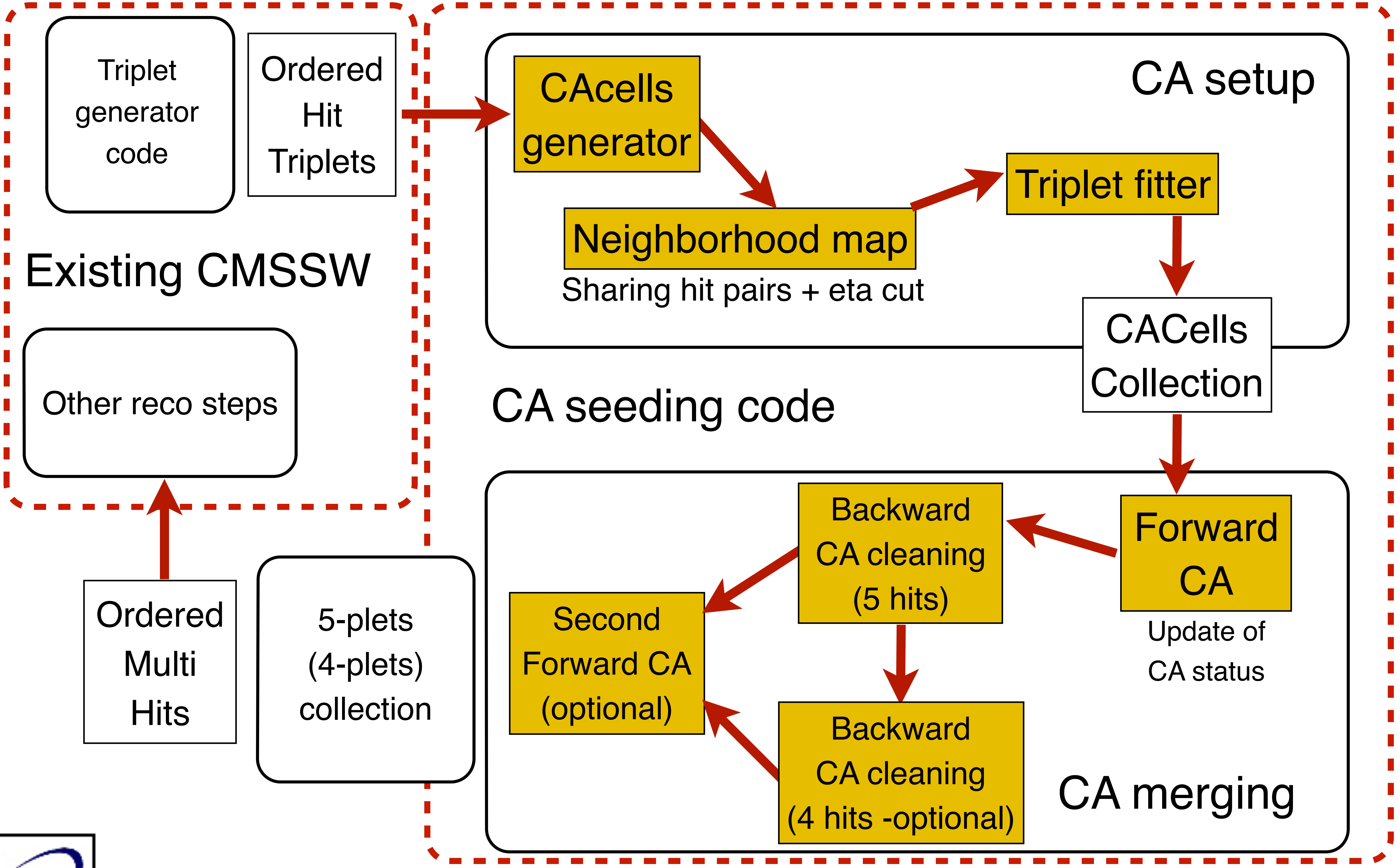


Forward CA

Backward CA cleaning

Second Forward CA

CASeedGenerator (CMSSW version)



Existing CMSSW

Triplet generator code

Ordered Hit Triplets

Other reco steps

Ordered Multi Hits

5-plets (4-plets) collection

CA seeding code

CA setup

CAcells generator

Neighborhood map

Sharing hit pairs + eta cut

Triplet fitter

CACells Collection

CA merging

Forward CA

Update of CA status

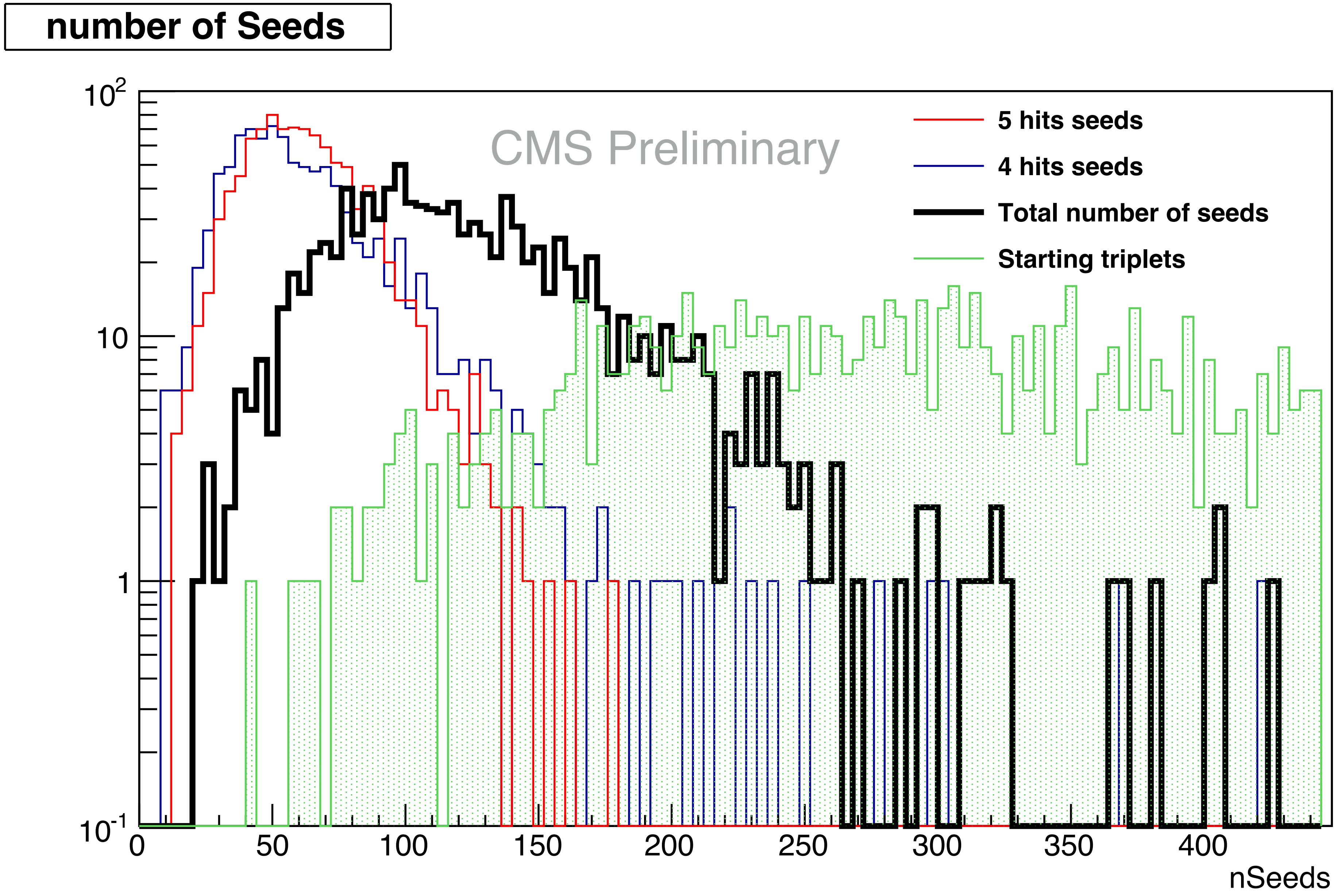
Backward CA cleaning (5 hits)

Backward CA cleaning (4 hits -optional)

Second Forward CA (optional)

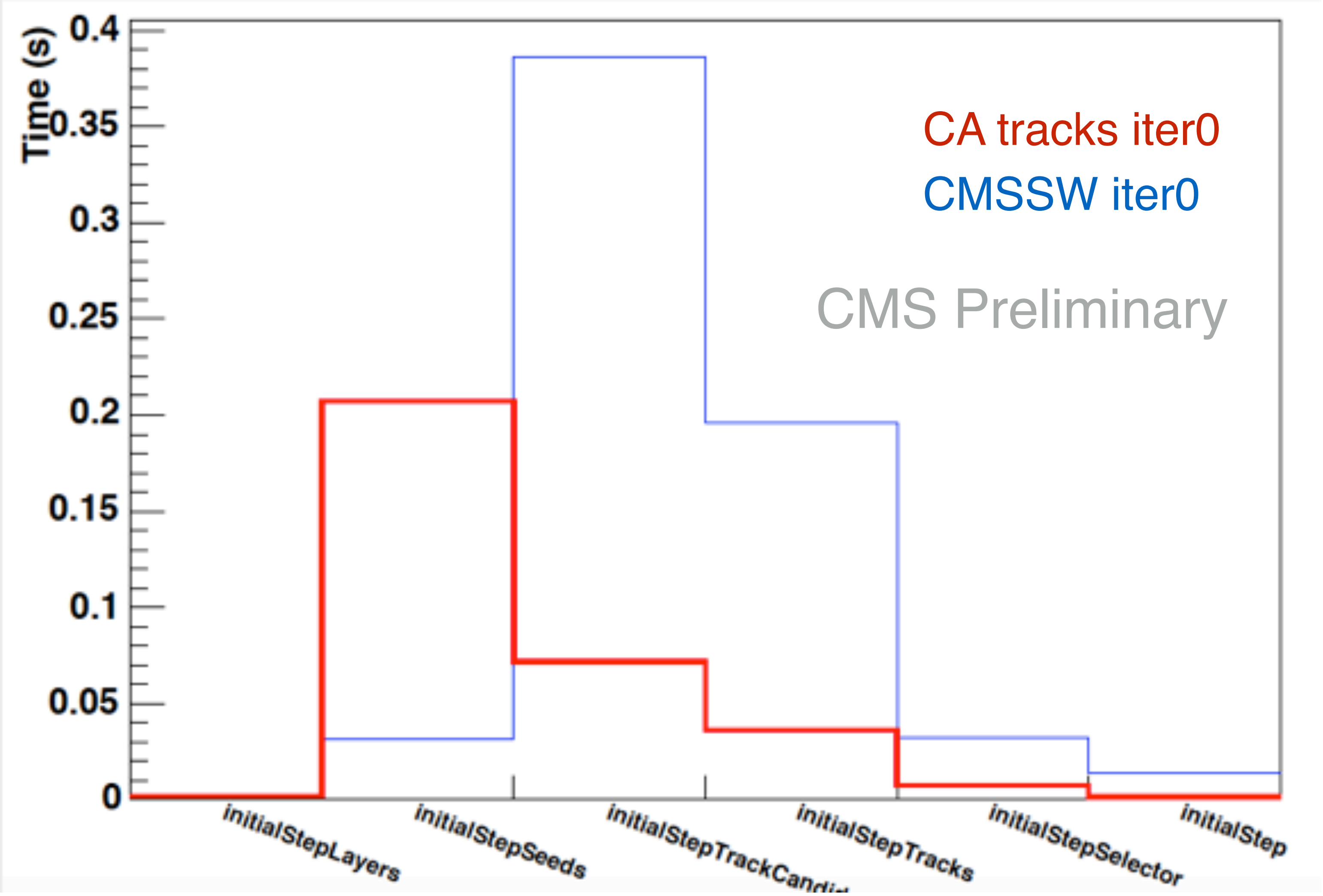
CASeedGenerator (CMSSW version)

Seed production (TTbar + PU)



CASeedGenerator (CMSSW version)

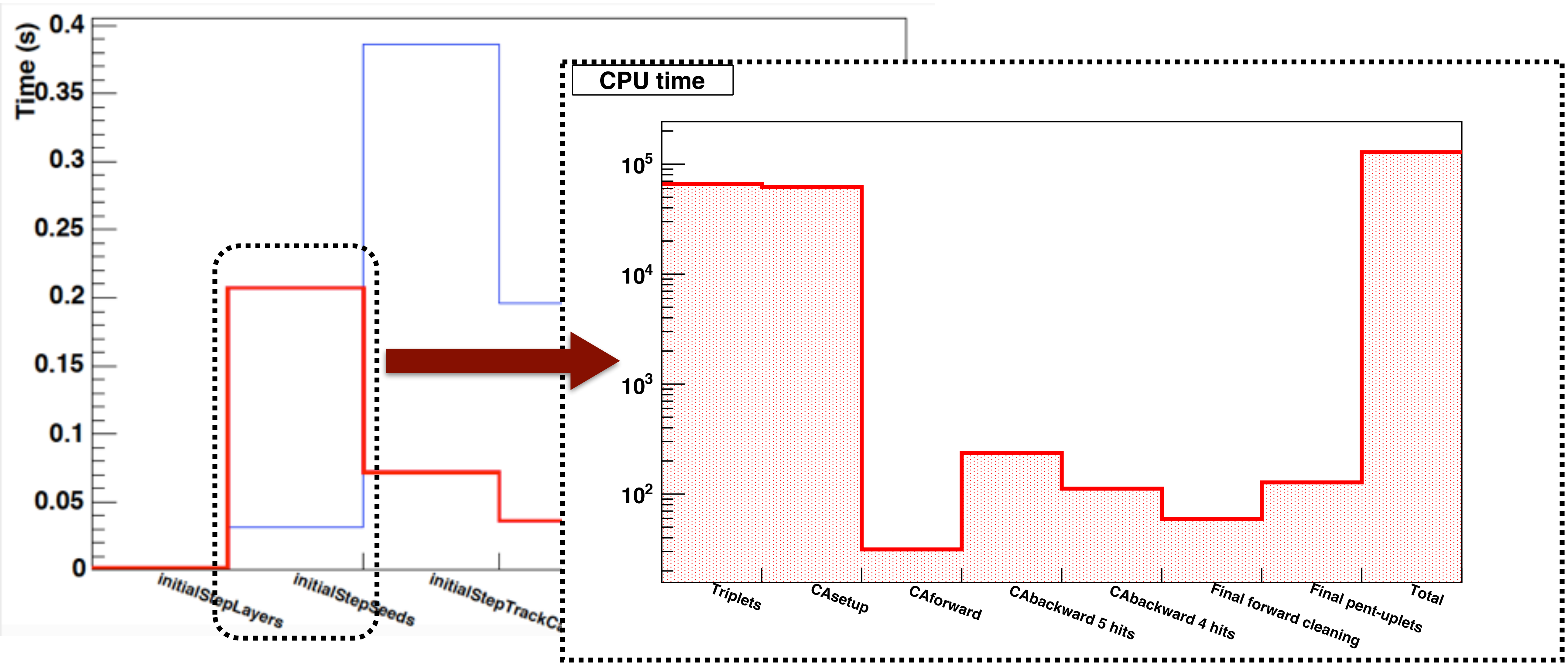
Time performances (TTbar + PU)



Initial Step tracks (iter0)

CASeedGenerator (CMSSW version)

Time performances (TTbar + PU)

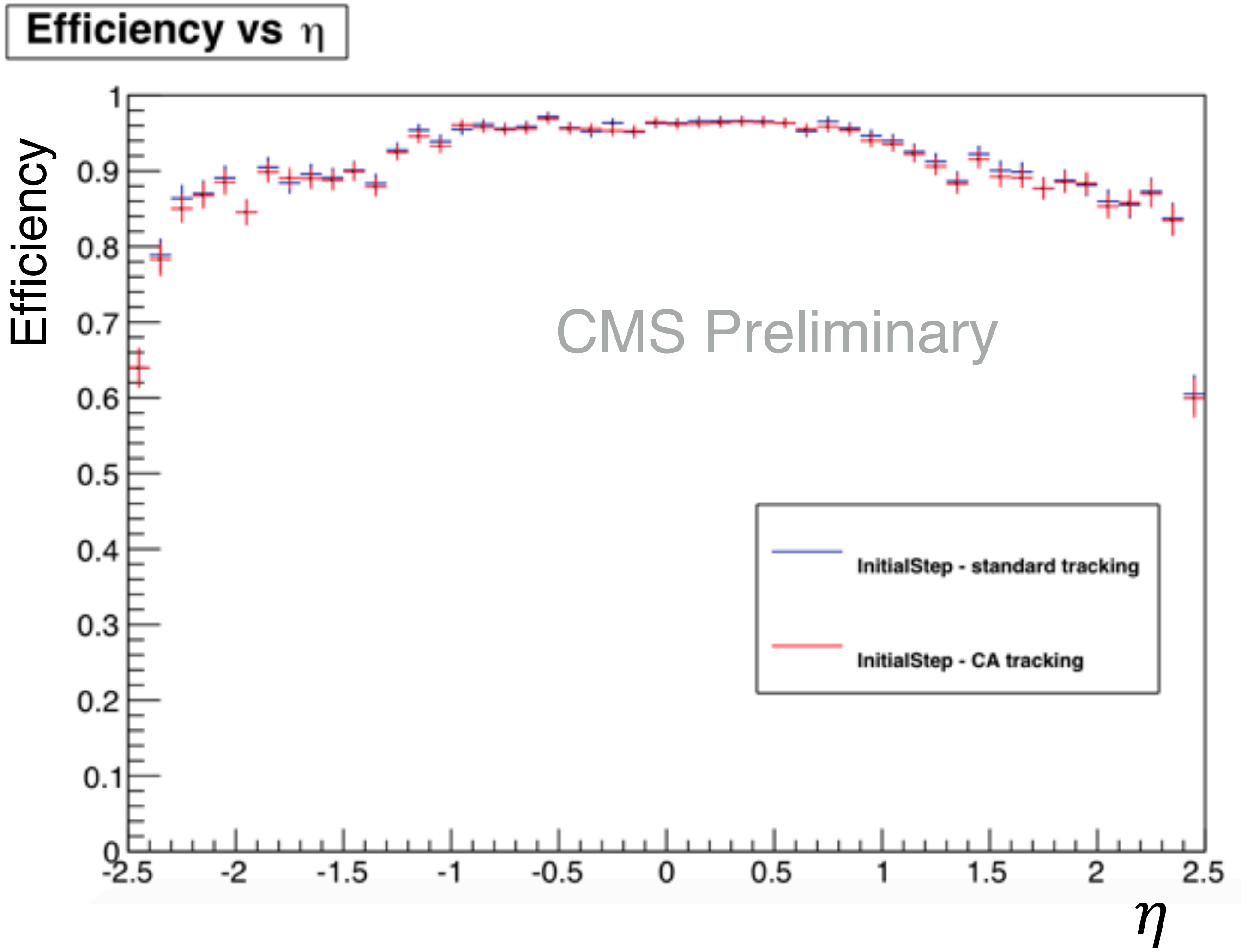


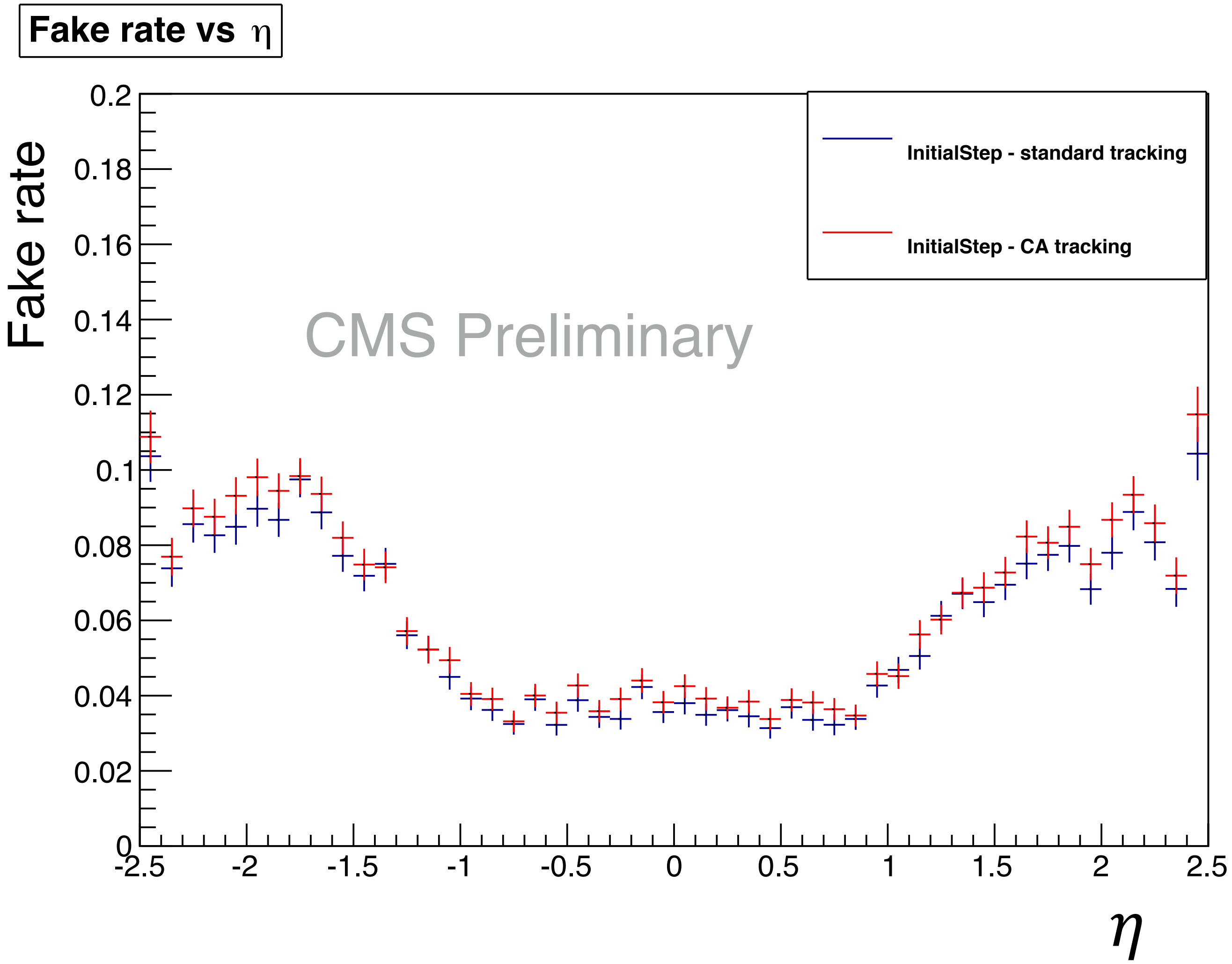
Initial Step tracks (iter0)

Code blocks of the CA algorithm

CASeedGenerator (CMSSW version)

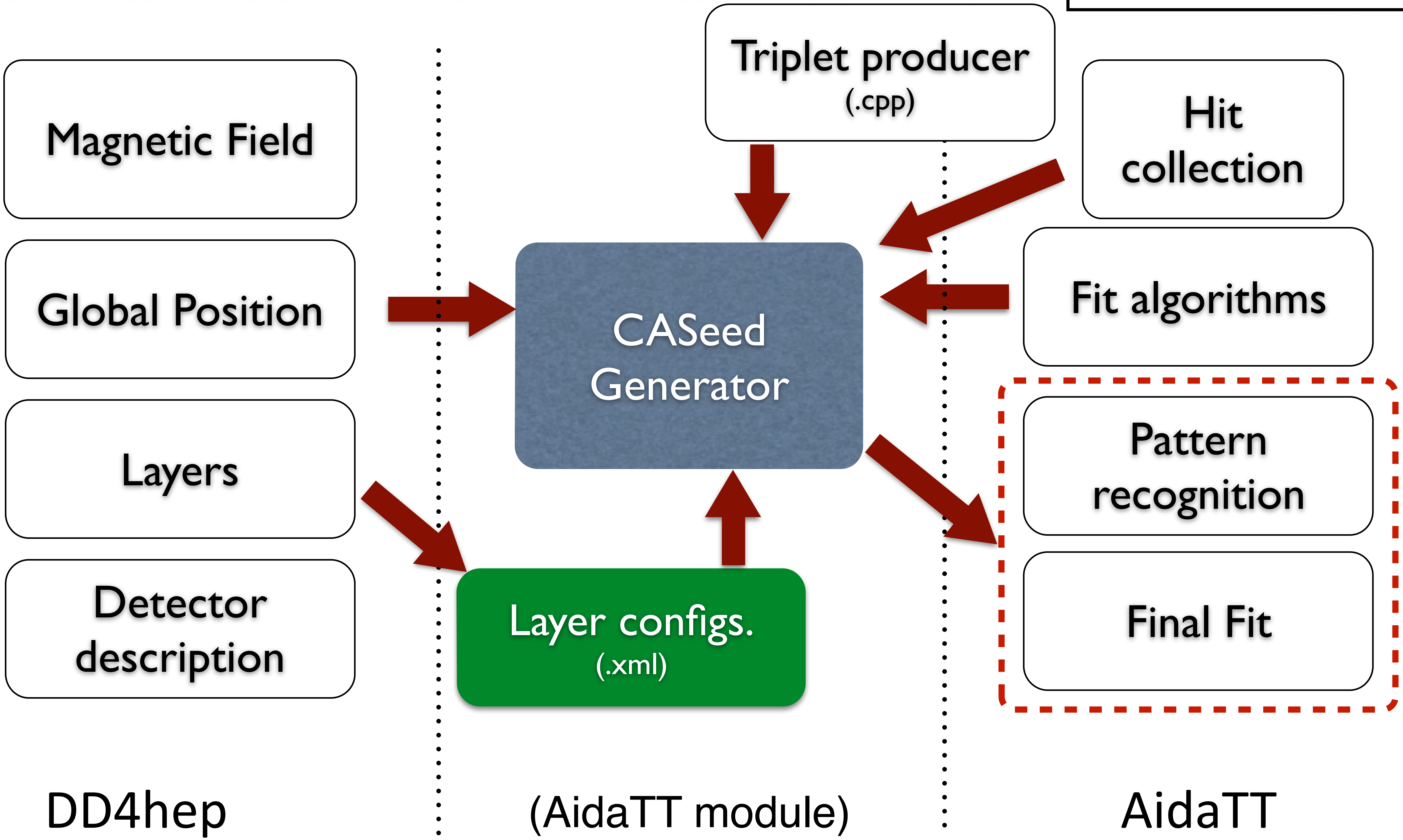
Efficiency - Fakerate (TTbar + PU)





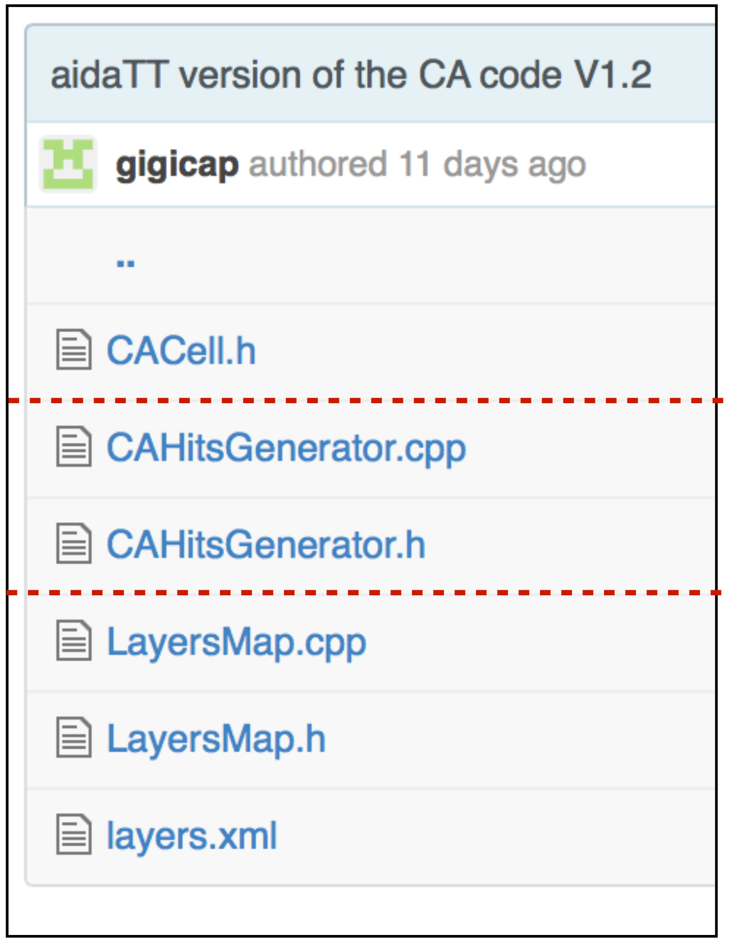
Full tracking

CASeedGenerator (AidaTT module)



CASeedGenerator (AidaTT tool)

The Aida CA code can be found in: https://github.com/gigicap/AidaCA



aidaTT version of the CA code V1.2

gigicap authored 11 days ago

..

CACell.h

CAHitsGenerator.cpp

CAHitsGenerator.h

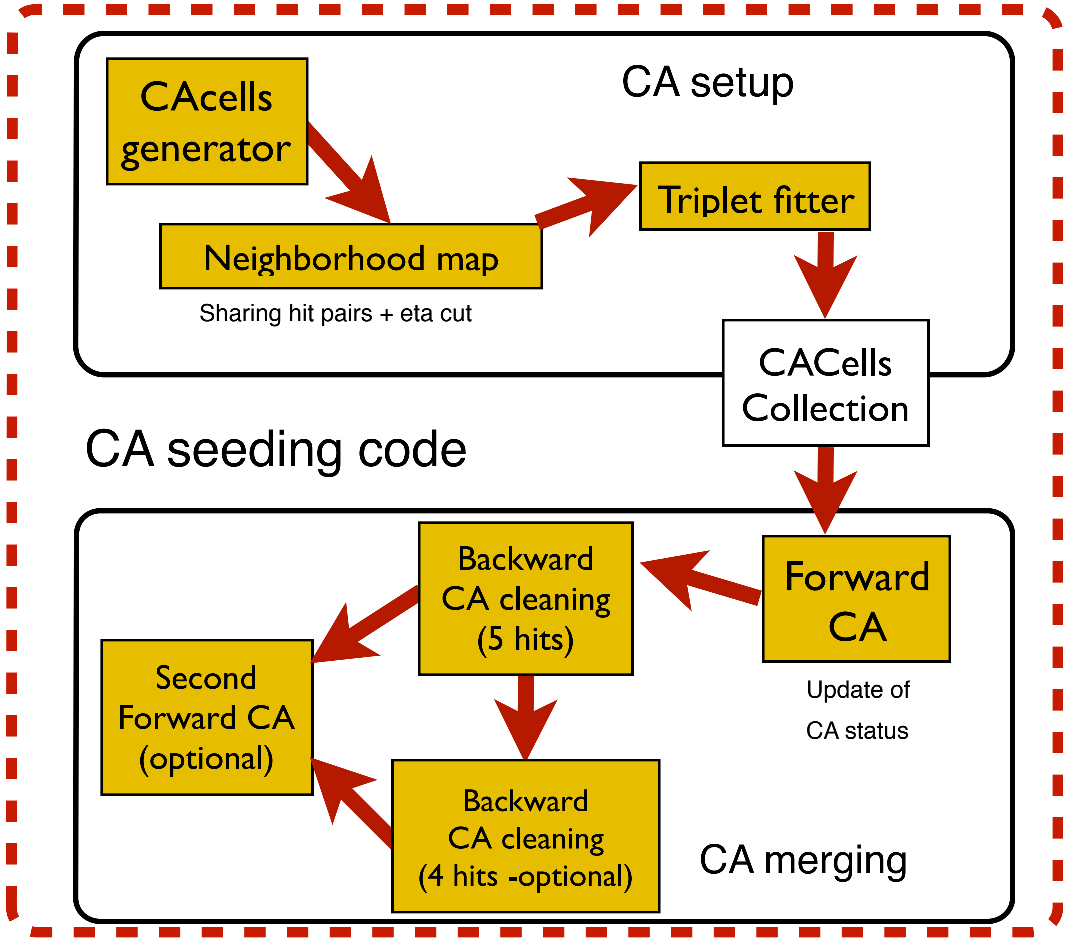
LayersMap.cpp

LayersMap.h

layers.xml

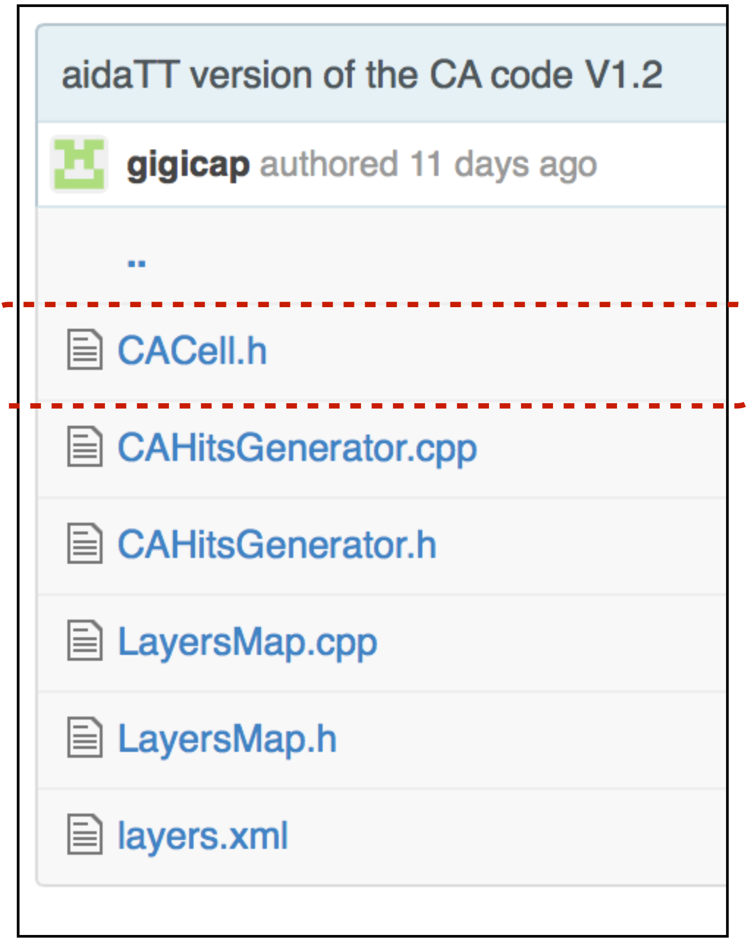
Main part of the code
CA algorithm resides here

+ the other AidaTT xml
(materials, elements…)



CASeedGenerator (AidaTT tool)

The Aida CA code can be found in: https://github.com/gigicap/AidaCA



aidaTT version of the CA code V1.2

gigicap authored 11 days ago

..

CACell.h

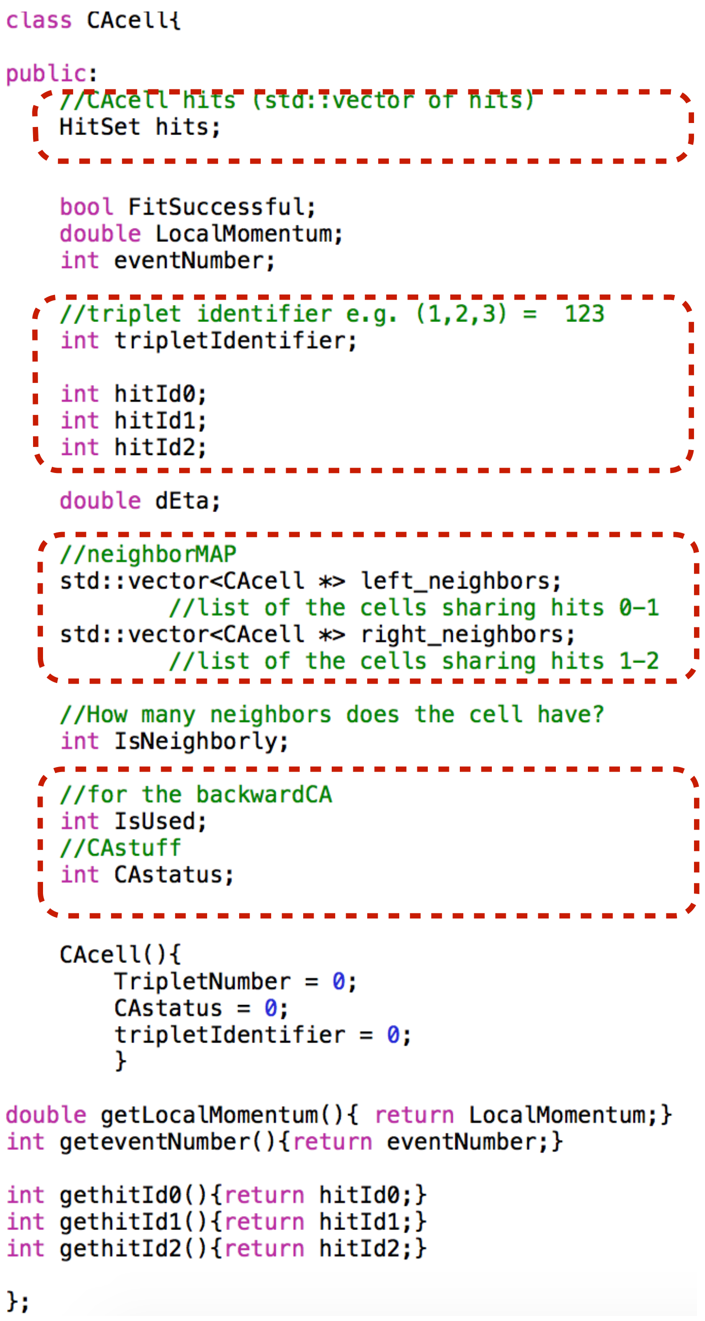
CAHitsGenerator.cpp

CAHitsGenerator.h

LayersMap.cpp

LayersMap.h

layers.xml



```
class CAcell{

public:
   //CAcell hits (std::vector of hits)
   HitSet hits;

   bool FitSuccessful;
   double LocalMomentum;
   int eventNumber;

   //triplet identifier e.g. (1,2,3) =  123
   int tripletIdentifier;

   int hitId0;
   int hitId1;
   int hitId2;

   double dEta;

   //neighborMAP
   std::vector<CAcell *> left_neighbors;
           //list of the cells sharing hits 0-1
   std::vector<CAcell *> right_neighbors;
           //list of the cells sharing hits 1-2

   //How many neighbors does the cell have?
   int IsNeighborly;

   //for the backwardCA
   int IsUsed;
   //CAstuff
   int CAstatus;

   CAcell(){
       TripletNumber = 0;
       CAstatus = 0;
       tripletIdentifier = 0;
       }

double getLocalMomentum(){ return LocalMomentum;}
int geteventNumber(){return eventNumber;}

int gethitId0(){return hitId0;}
int gethitId1(){return hitId1;}
int gethitId2(){return hitId2;}

};
```

The three hits of the cell

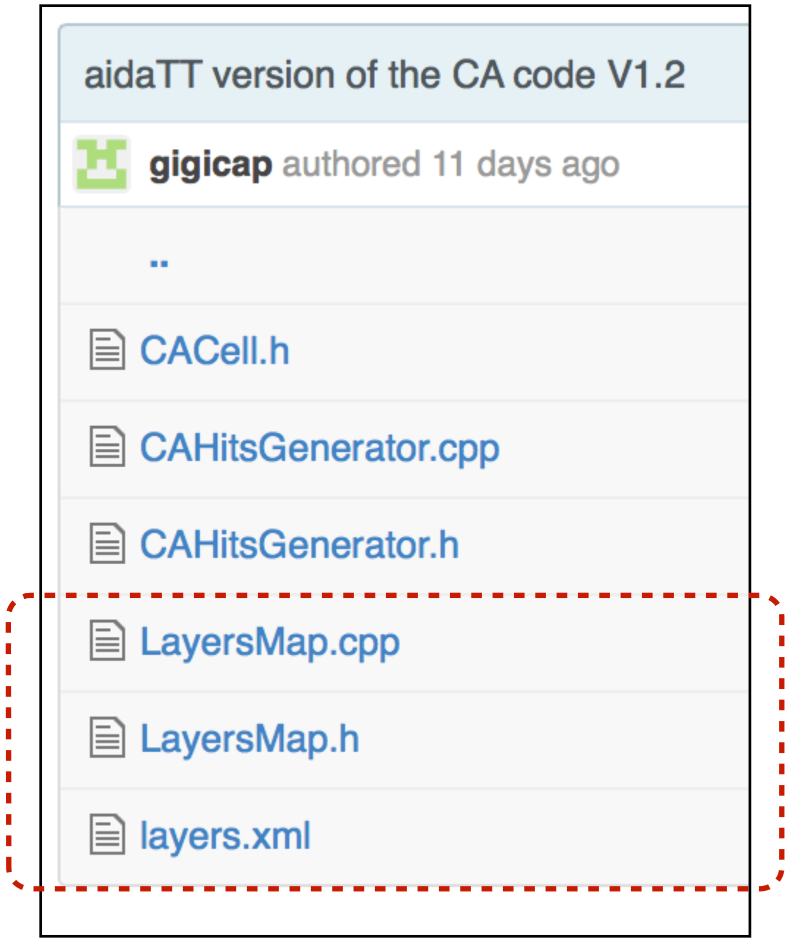
Information about layers configurations

Lists of the neighbour cells

Evolution variables of the CA

CASeedGenerator (AidaTT tool)

The Aida CA code can be found in: https://github.com/gigicap/AidaCA



aidaTT version of the CA code V1.2

gigicap authored 11 days ago

..

CACell.h

CAHitsGenerator.cpp

CAHitsGenerator.h

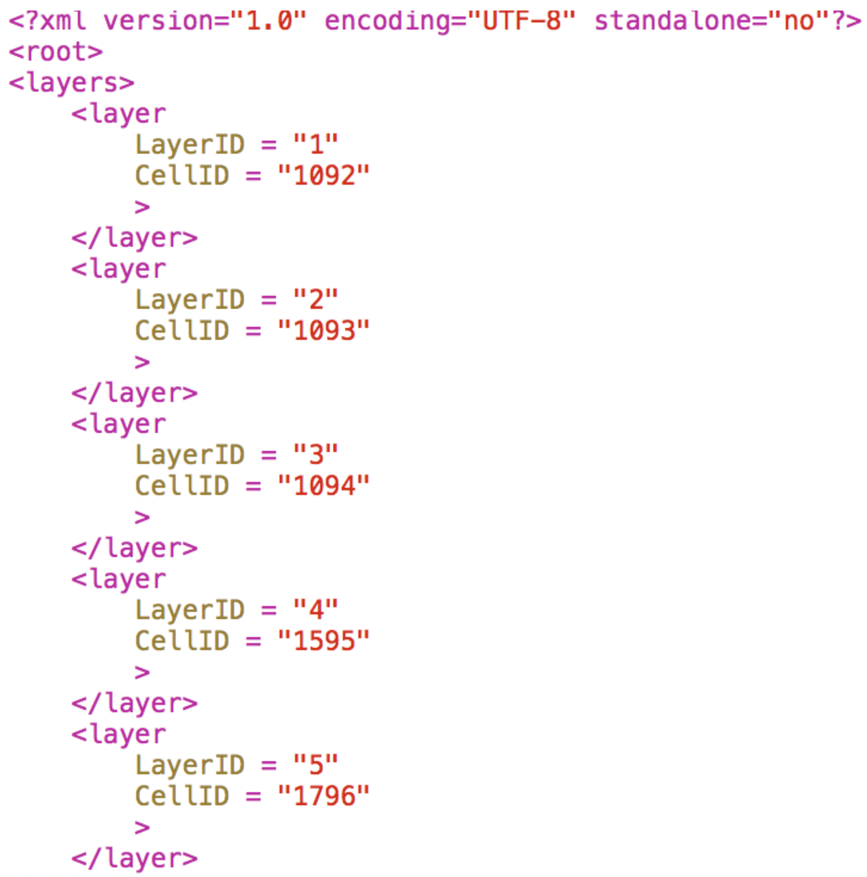
LayersMap.cpp

LayersMap.h

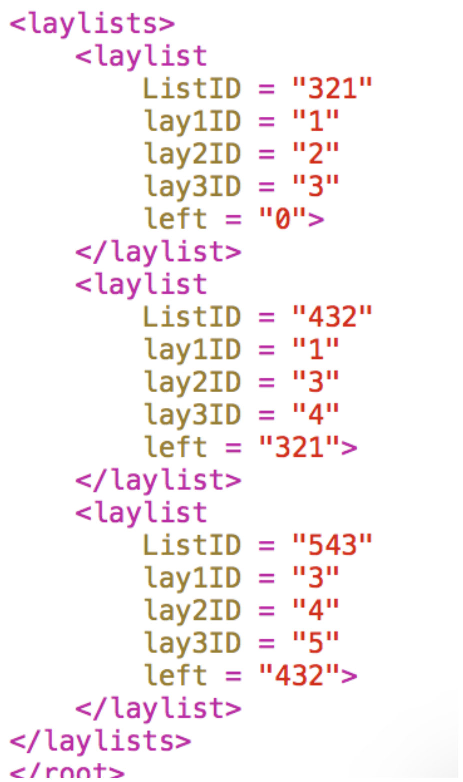
layers.xml

Objects for Layers configurations handling and xml parser (xercesC) in LayersMap.cpp and LayersMap.h

The editable layers.xml contains dictionaries to ‘translate’ the detector layers into CA layers configurations



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
<layers>
    <layer
        LayerID = "1"
        CellID = "1092"
        >
    </layer>
    <layer
        LayerID = "2"
        CellID = "1093"
        >
    </layer>
    <layer
        LayerID = "3"
        CellID = "1094"
        >
    </layer>
    <layer
        LayerID = "4"
        CellID = "1595"
        >
    </layer>
    <layer
        LayerID = "5"
        CellID = "1796"
        >
    </layer>
```



```
<laylists>
    <laylist
        ListID = "321"
        lay1ID = "1"
        lay2ID = "2"
        lay3ID = "3"
        left = "0">
    </laylist>
    <laylist
        ListID = "432"
        lay1ID = "1"
        lay2ID = "3"
        lay3ID = "4"
        left = "321">
    </laylist>
    <laylist
        ListID = "543"
        lay1ID = "3"
        lay2ID = "4"
        lay3ID = "5"
        left = "432">
    </laylist>
</laylists>
</root>
```

Summary

- The task of handling the pile-up requires the definition of new tracking algorithm:
  - Computationally smart, simple and fast (in order to be less time-greed as possible)
  - Parallelizable (in order to be easily exported also to GPU architectures)
  - Less detector-dependent as possible (Aida WP2 philosophy requirements)
- A tracking based on long seeds built with Cellular Automaton is complete:
  - It has been developed for CMS fully works in CMSSW (put in official release in these days)
    - Fully tested and validated with the current geometry (most challenging in terms of inhomogeneities)
    - Built also for the upgraded geometry (although not fully tested yet)
  - An AidaTT module version has also been written
    - All the detector infos are read from an xml configuration file

Backup

Cellular Automata approach to seeding

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

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

‘quality’ parameters (e.g. estimated tp)

Backward CA cleaning

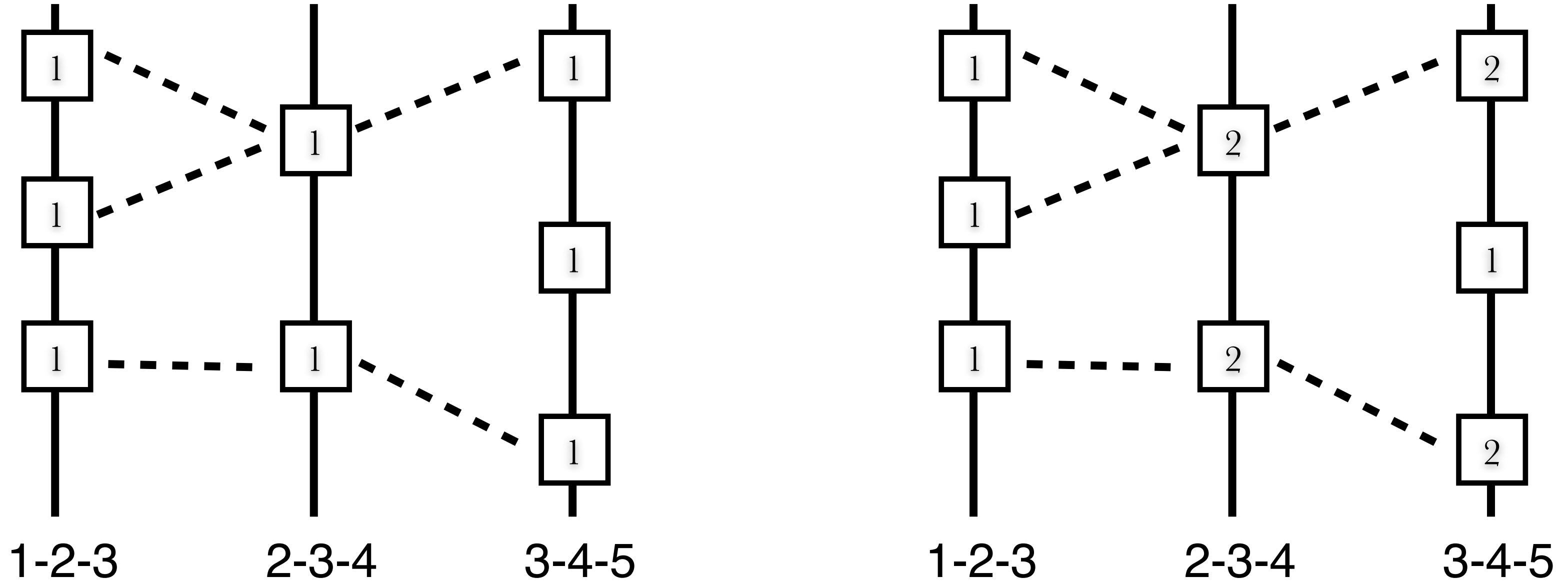
Second Forward CA

Maximize the length of the ‘multiplet’

Forward CA

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

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



Continue until no more cells will increase their status.

Cellular Automata approach to seeding

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

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

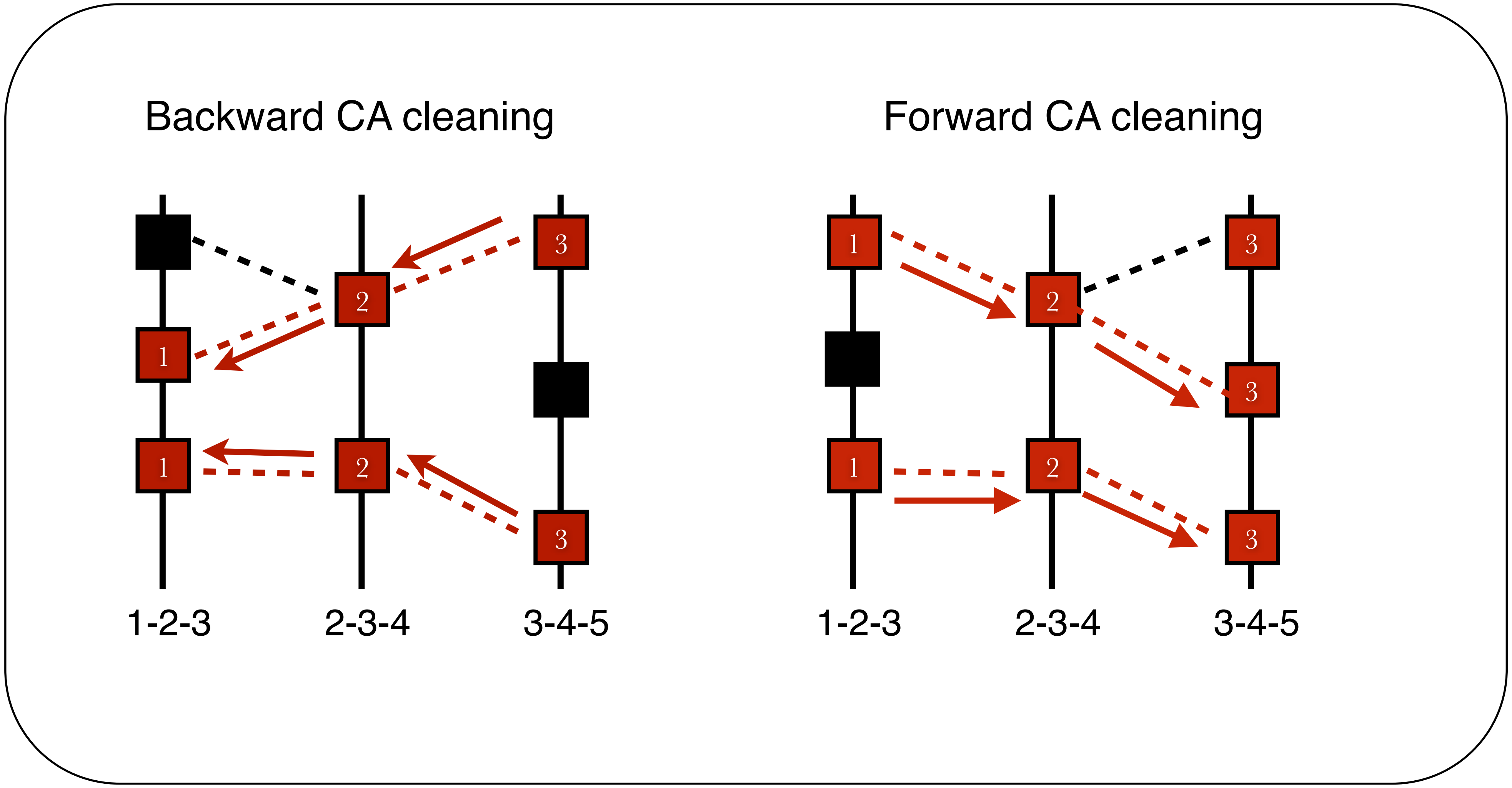
'quality' parameters (e.g. estimated tp)

Backward CA cleaning

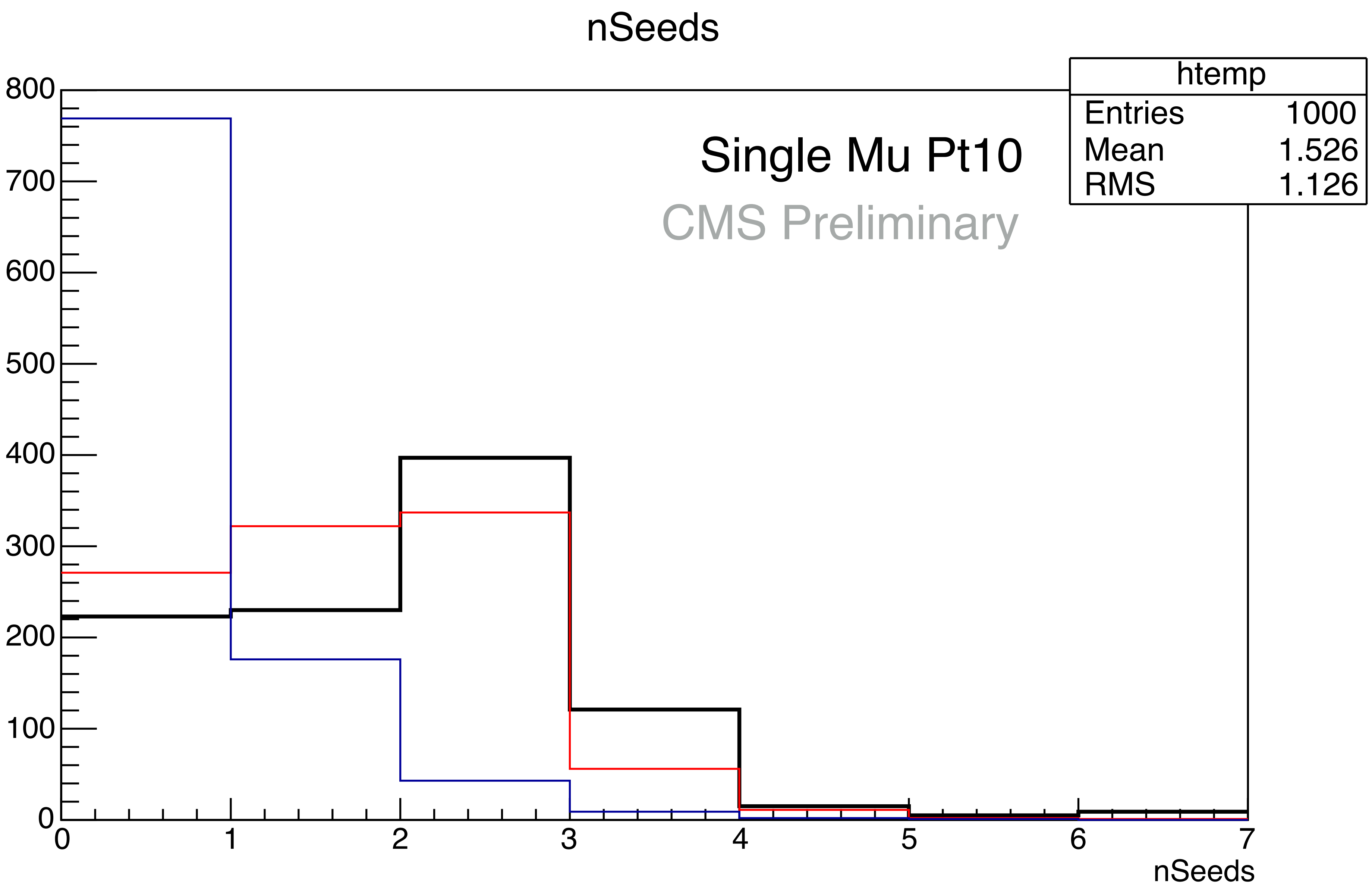
Second Forward CA

Maximize the length of the 'multiplet'

Forward CA

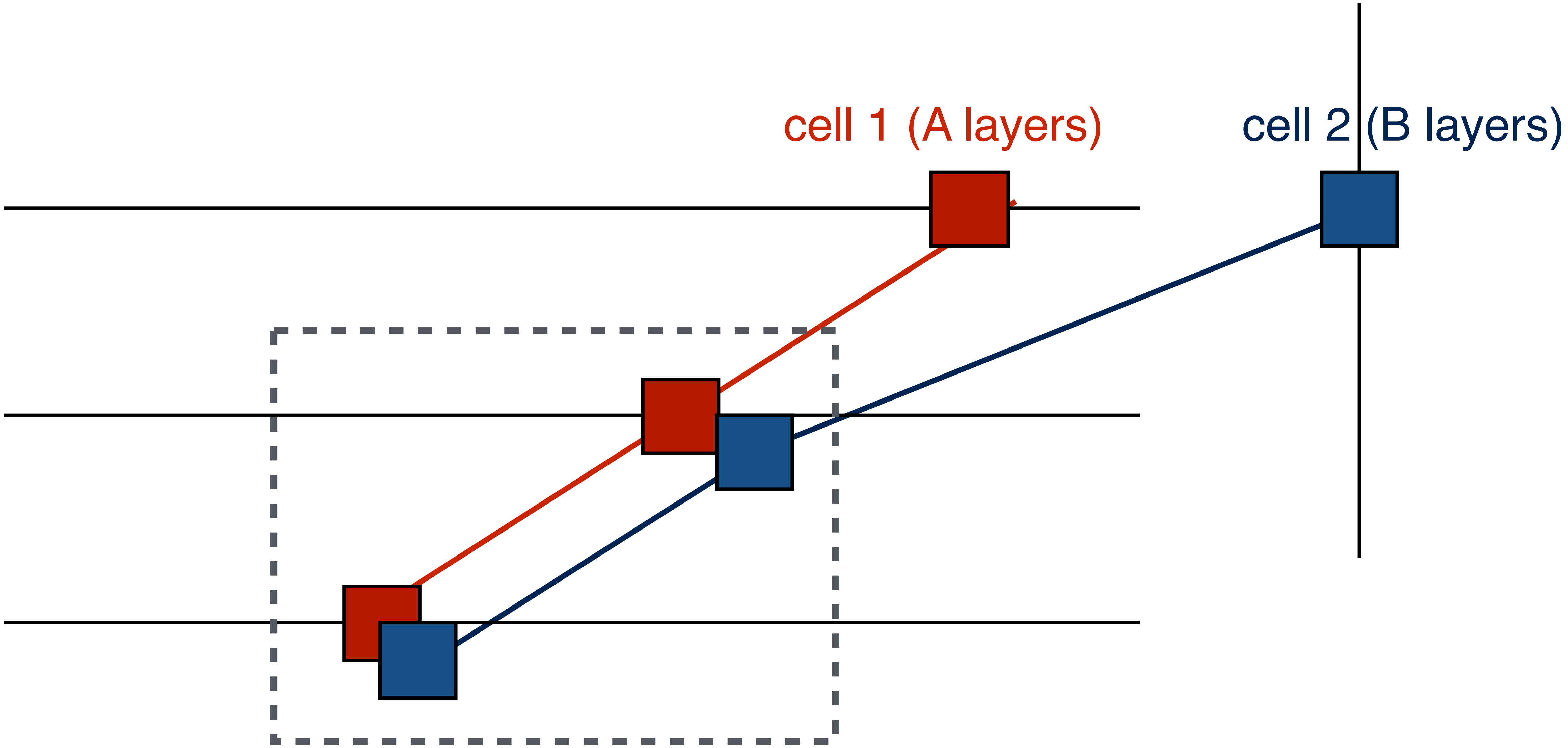


Backup


nSeeds
htemp
Entries 1000
Mean 1.526
RMS 1.126
Single Mu Pt10
CMS Preliminary
800
700
600
500
400
300
200
100
0
0
1
2
3
4
5
6
7
nSeeds

Backup

Why using the layer configurations for neighbourhood?


cell 1 (A layers)
cell 2 (B layers)