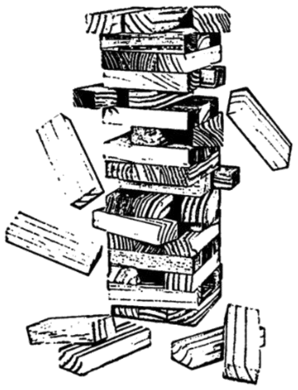
PATATRACK

Accelerated Pixel Tracks at the HLT starting from Run 3

D. Bacciu[1], A. Bocci[2], E. Brondolin[8], C. Calabria[3], A. Carta[1], S. Roy Chowdhury[4], E. Corni[5], A. Di Florio[3], S. Di Guida[6], S. Dubey[7], S. Dugad[7], S. Dutta[4], R. Fruhwirth[8], V. Innocente[2], V. Khristenko[2], M. Kortelainen[2], P. Mal[4], D. Menasce[6], **F. Pantaleo**[2], M. Pierini[2], M. Rovere[2], S. Sarkar[4], V. Volkl[2]

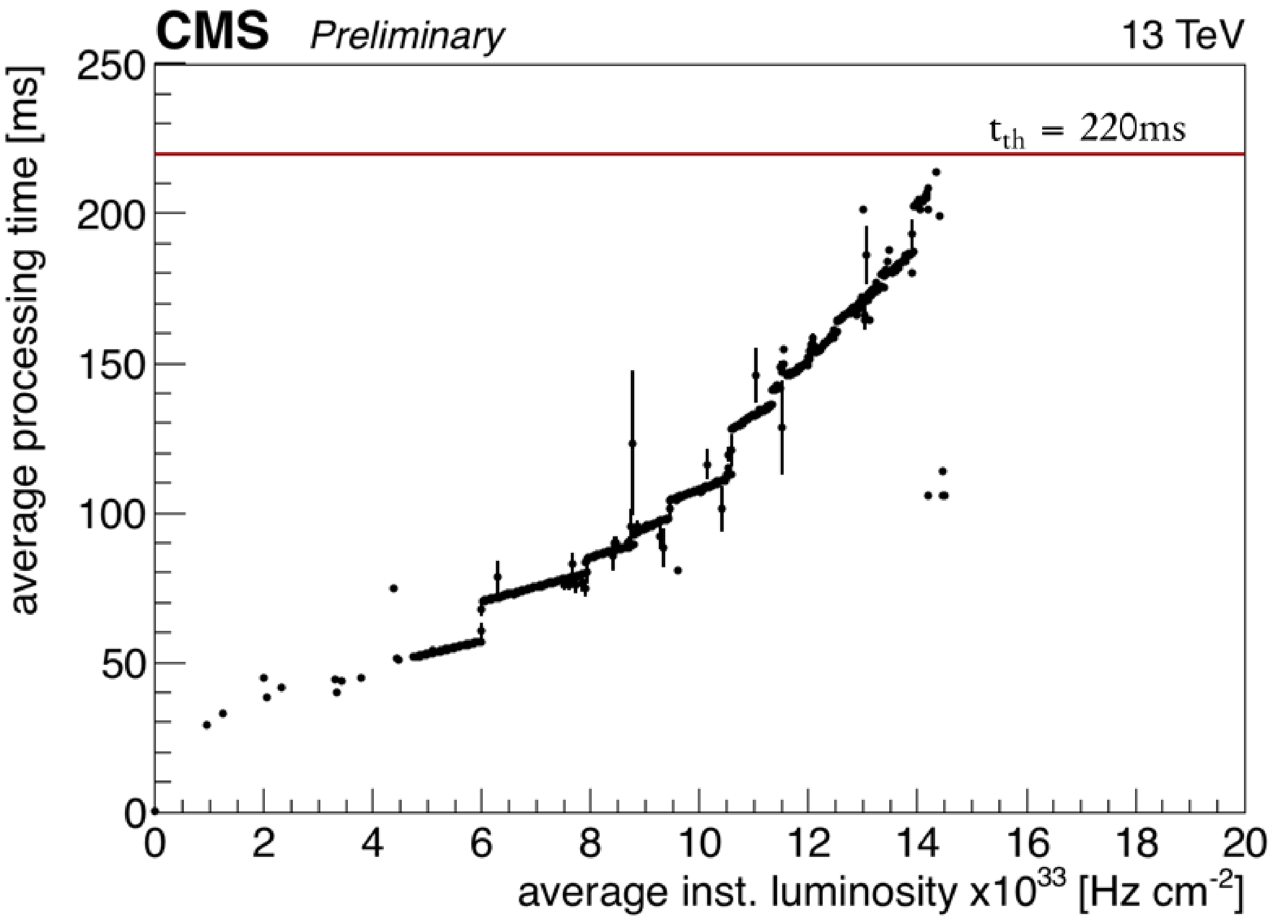
University of Pisa[1], *CERN*[2], *INFN Bari*[3], *SINP*[4], *INFN CNAF*[5], *INFN Milano Bicocca*[6], *TIFR*[7] , Austrian Academy of Sciences[8]

felice@cern.ch



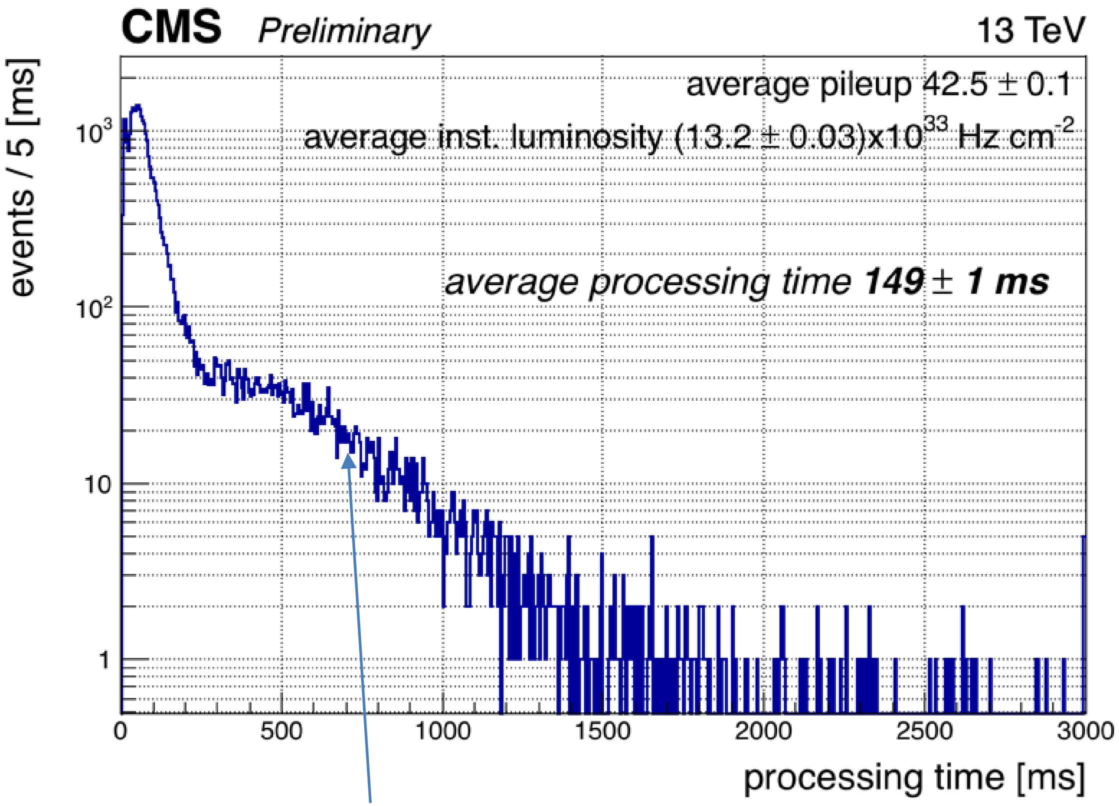
CMS High-Level Trigger in Run 2 (1/2)

- Today the CMS online farm consists of ~26k Intel Xeon cores
  - The current approach: one event per logical core
- Pixel Tracks are not reconstructed for all the events at the HLT
- This will be even more difficult at higher pile-up
  - More memory/event



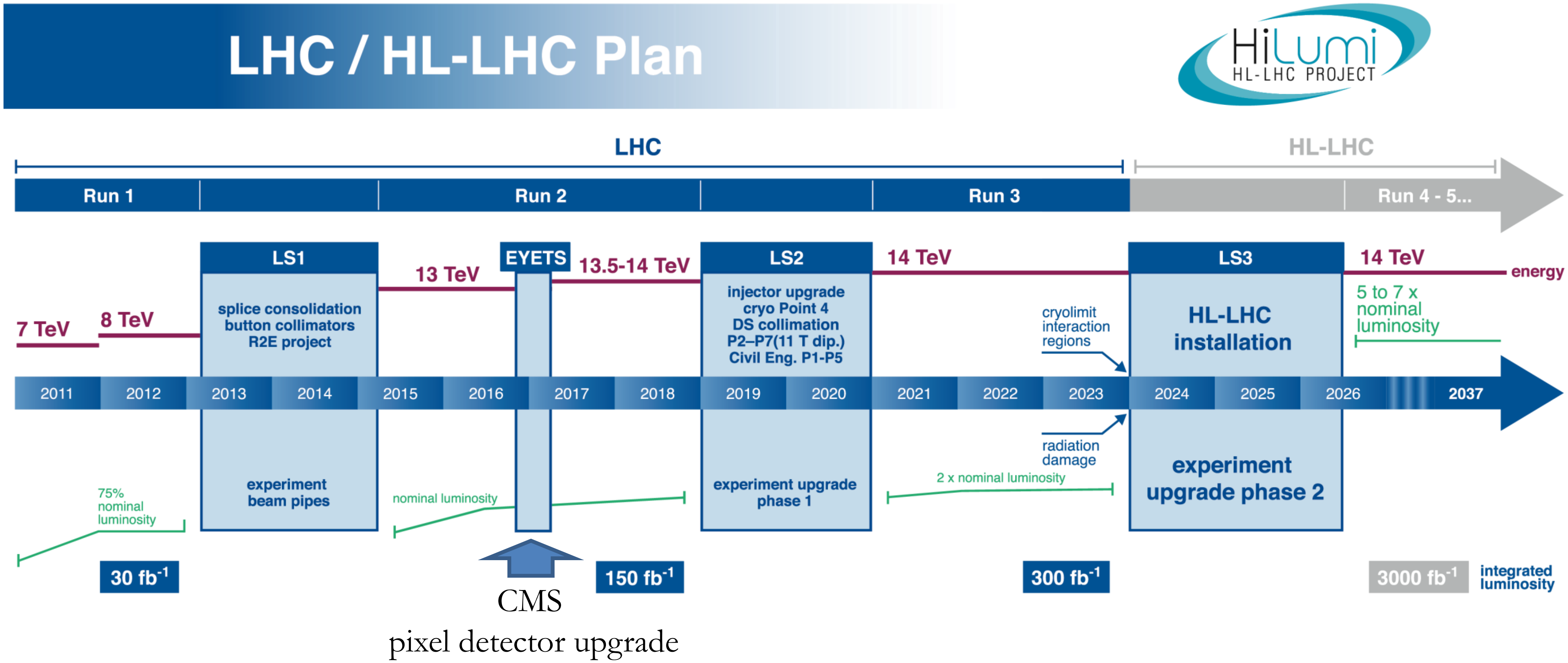
CMS High-Level Trigger in Run 2 (2/2)

- Today the CMS online farm consists of ~22k Intel Xeon cores
  - The current approach: one event per logical core
- Pixel Tracks are not reconstructed for all the events at the HLT
- This will be even more difficult at higher pile-up
  - More memory/event



full track reconstruction and particle flow e.g. jets, tau

CMS and LHC Upgrade Schedule



Pixel Tracks

- Evaluation of Pixel Tracks combinatorial complexity could easily be dominated by track density and become one of the bottlenecks of the High-Level Trigger and offline reconstruction execution times.
- The CMS HLT farm and its offline computing infrastructure cannot rely on an exponential growth of frequency guaranteed by the manufacturers.
- Hardware and algorithmic solutions have been studied

Pixel Tracks on GPUs during Run-3

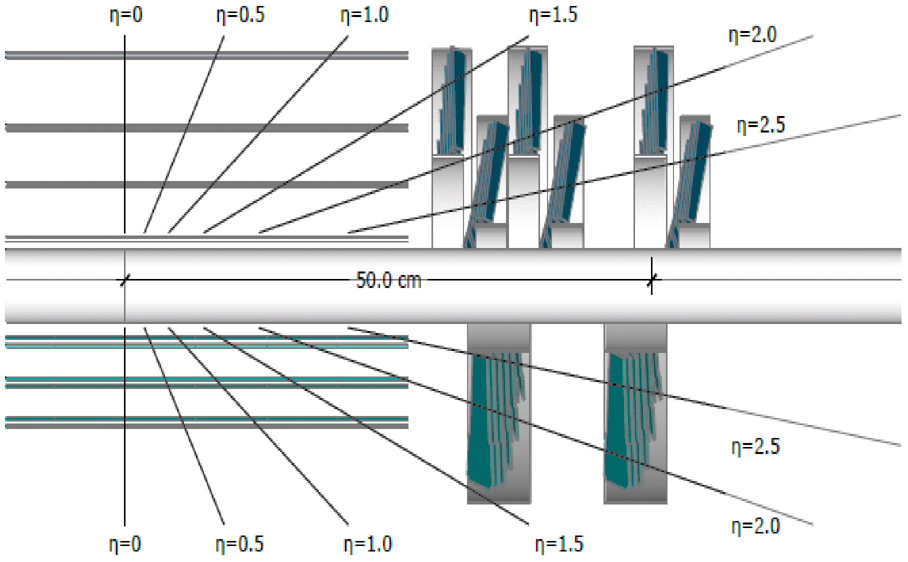
PATATRACK

- Curiosity-driven project started in 2016 by a very small group of passionate people, right after I gave a GPU programming course…
- CMS-specific project
- Started with no funding, no EPRs: people joined because it's fun and interesting to work at the forefront of technology
- Soon grown:
  - CERN: F. Pantaleo, V. Innocente, M. Rovere, A. Bocci, M. Kortelainen, M. Pierini, V. Volkl (SFT), V. Khristenko (IT, openlab)
  - Austrian Academy of Sciences: E. Brondolin, R. Fruhwirth
  - INFN Bari: A. Di Florio, C. Calabria
  - INFN MiB: D. Menasce, S. Di Guida
  - INFN CNAF: E. Corni
  - SAHA: S. Sarkar, S. Dutta, S. Roy Chowdhury, P. Mal
  - TIFR: S. Dugad, S. Dubey
  - University of Pisa (Computer Science dep.): D. Bacciu, A. Carta
  - Thanks also to the contributions of many short term students (Bachelor, Master, GSoC): Alessandro, Ann-Christine, Antonio, Dominik, Jean-Loup, Konstantinos, Kunal, Luca, Panos, Roberto, Romina, Simone, Somesh

- Interests: algorithms, HPC, heterogeneous computing, machine learning, software eng., FPGAs…
- Lay the foundations of the online/offline reconstruction starting from 2020s (tracking, HGCal)
- Website under construction: PATATRACK , contact: patatrack-rd@cern.ch
- Meetings: https://indico.cern.ch/category/7804/

From RAW to Tracks during run 3

- Profit from the end-of-year upgrade of the Pixel to redesign the seeding code from scratch
  - Exploiting the information coming from the 4th layer would improve efficiency, b-tag, IP resolution
- Trigger avg latency should stay within max average time
- Reproducibility of the results (equivalence CPU-GPU)
- Integration in the CMS software framework
- Targeting a complete demonstrator by 2018 H2
- E-group: gpu-cms-pixel-phase1



- Ingredients:
  - Massive parallelism within the event
  - Independence from thread ordering in algorithms
  - Avoid useless data transfers and transformations
  - Simple data formats optimized for parallel memory access
- Result:
  - A GPU based application that takes RAW data and gives Tracks as result

Tracking at HLT

- Pixel hits are used for pixel tracks, vertices, seeding
- HLT Iterative tracking:

Iteration name	Phase0 Seeds	Phase1 Seeds	Target Tracks
Pixel Tracks	triplets	quadruplets	
Iter0	Pixel Tracks	Pixel Tracks	Prompt, high p_T
Iter1	triplets	quadruplets	Prompt, low p_T
Iter2	doublets	triplets	High p_T, recovery

Algorithm Stack

Input, size linear with PU

Raw to Digi

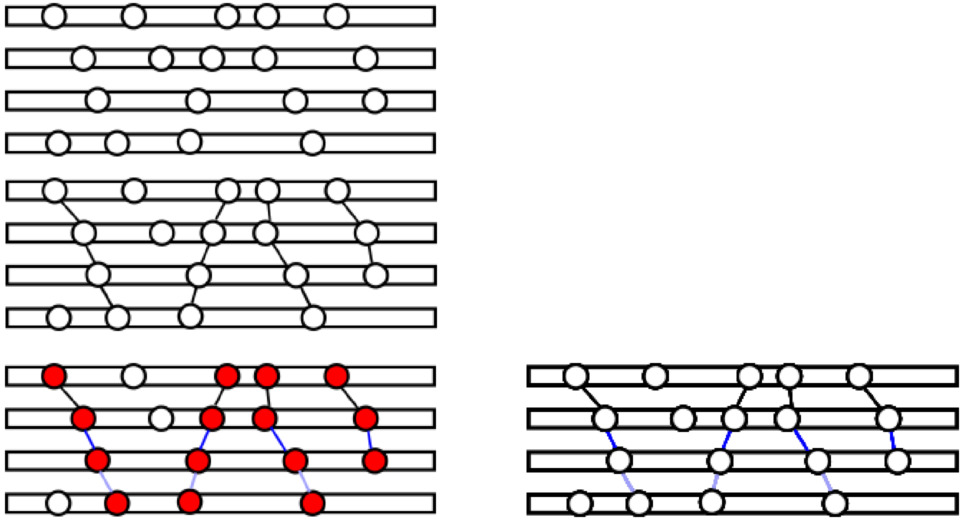
Hits - Pixel Clusterizer

Hit Pairs + DNN Filter

CA-based Hit Chain Maker

Riemann Fit

Output, size ~linear with PU + dependence on fake rate



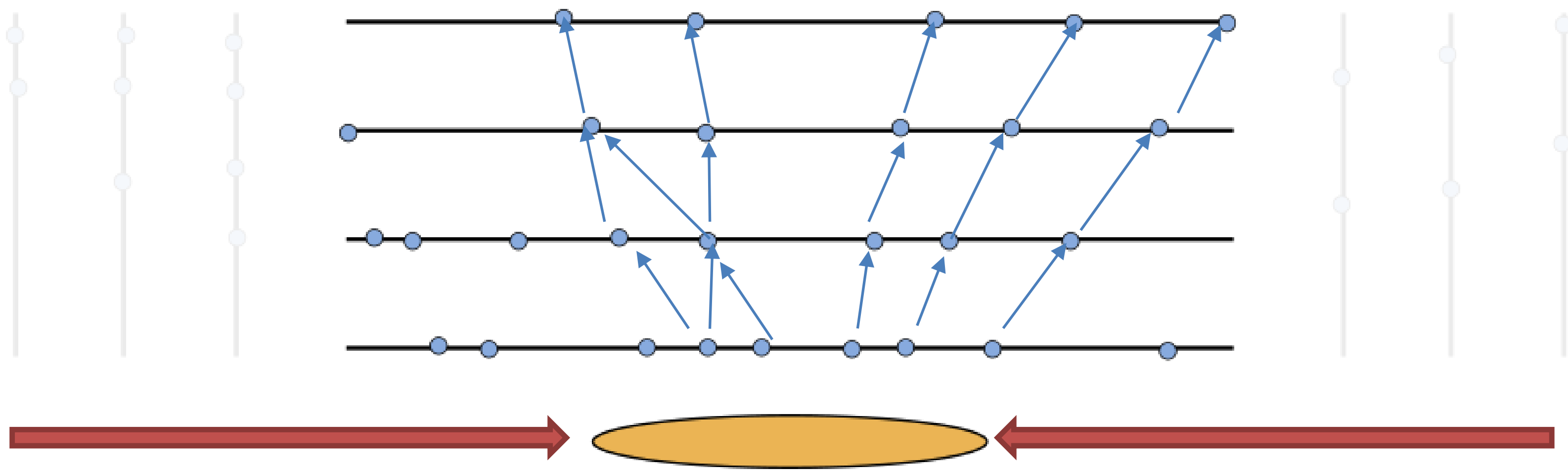
Overall status

- RAW to DIGI
  - Complete
- Clustering
  - Complete
- CPE
  - Almost complete
- Doublet generation
  - Ongoing
- Cellular Automaton
  - Complete, aligned to CMSSW
- Riemann Fit
  - CPU version implemented using Eigen (see talk by Roberto ~1month ago), GPU version missing
- Overall integration in CMSSW
  - In preparation

Massive parallelization?

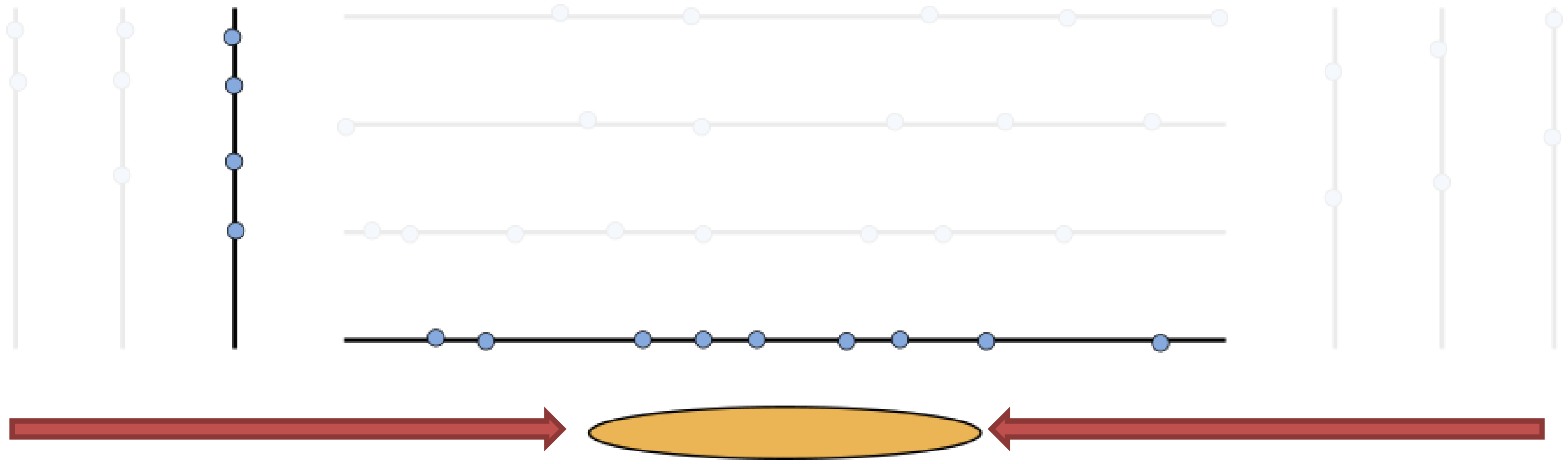
Our typical algorithms

- First create doublets from hits of pairs
- Take a third layer and propagate only the generated doublets
- Consider a fourth layer and propagate triplets
- Store found quadruplets and start from another pair of layers



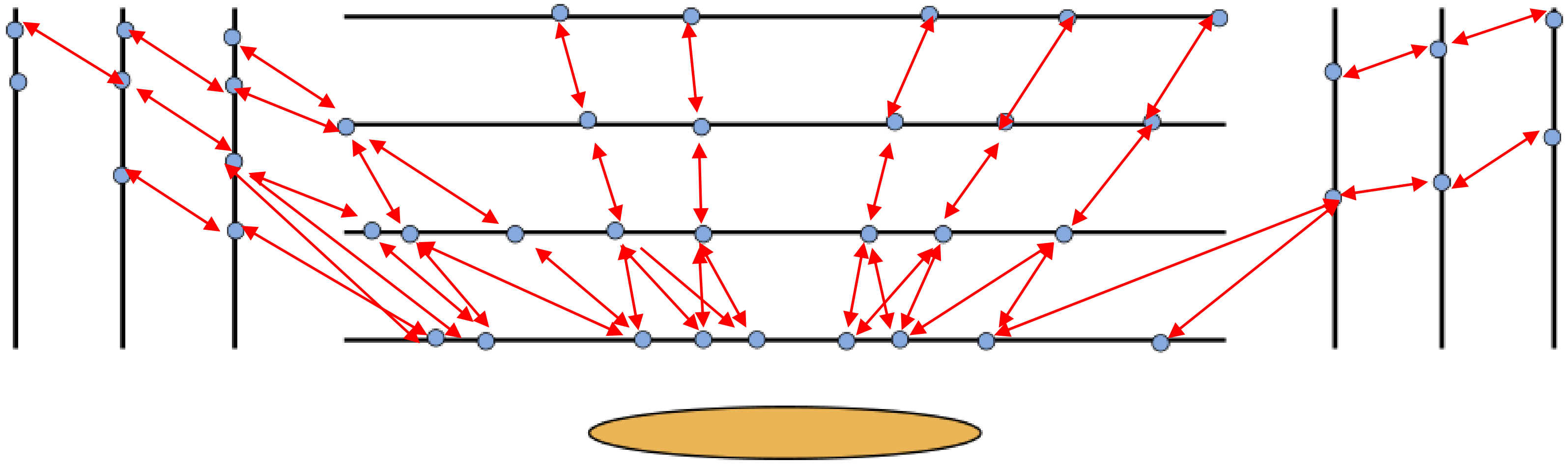
Our typical algorithms

- First create doublets from hits of pairs
- Take a third layer and propagate only the generated doublets
- Consider a fourth layer and propagate triplets
- Store found quadruplets and start from another pair of layers
- Repeat until happy…
- Does this fit the idea of massively parallel computation? I don't really think so…



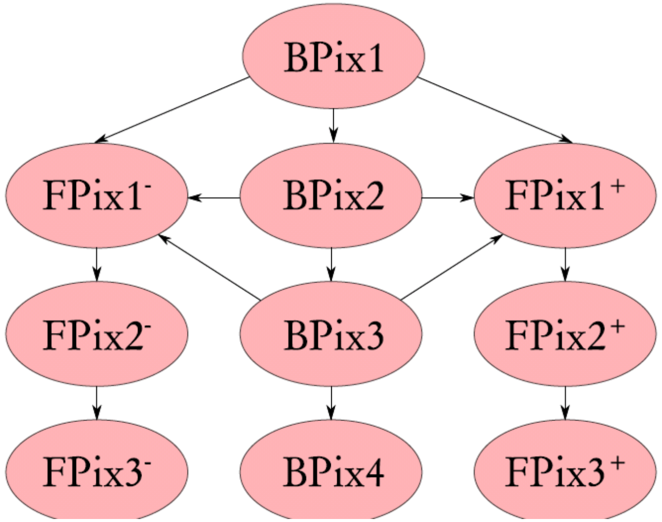
Cellular Automaton (CA)

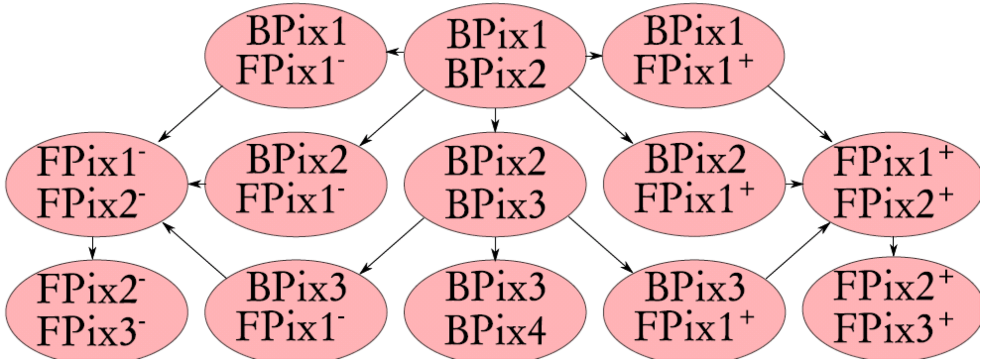
- The CA is a track seeding algorithm designed for parallel architectures
- It requires a list of layers and their pairings
  - A graph of all the possible connections between layers is created
  - Doublets aka Cells are created for each pair of layers (compatible with a region hypothesis)
  - Fast computation of the compatibility between two connected cells
  - No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize
- However this is not a static problem, not at all…



CAGraph of seeding layers

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

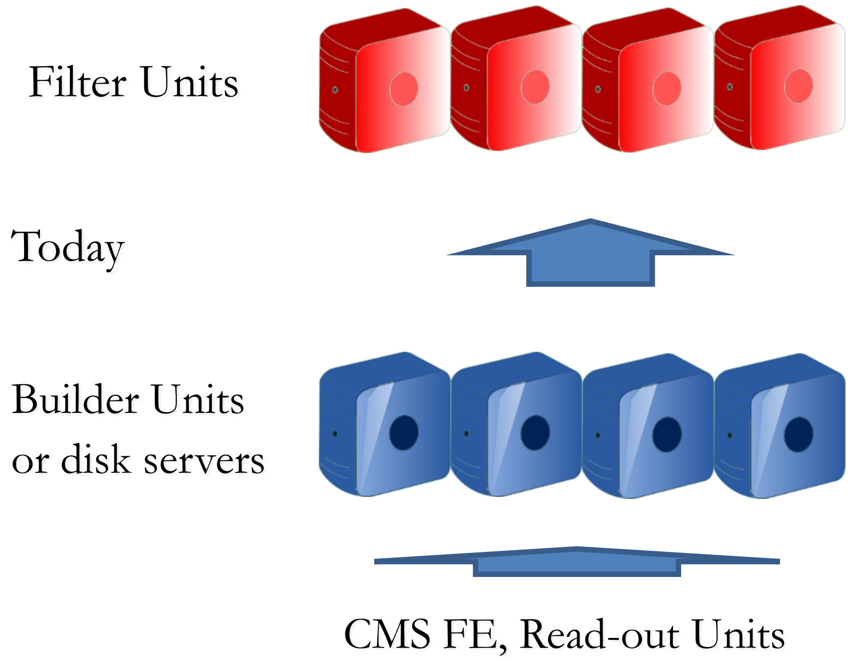




Integration studies

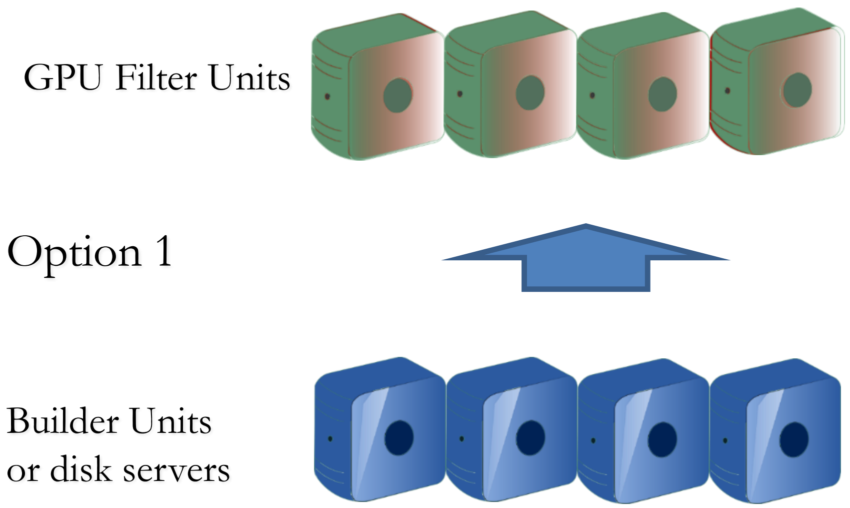
Integration in the Cloud and/or HLT Farm

- Different possible ideas depending on :
  - the fraction of the events running tracking
  - other parts of the reconstruction requiring a GPU



Integration in the Cloud/Farm

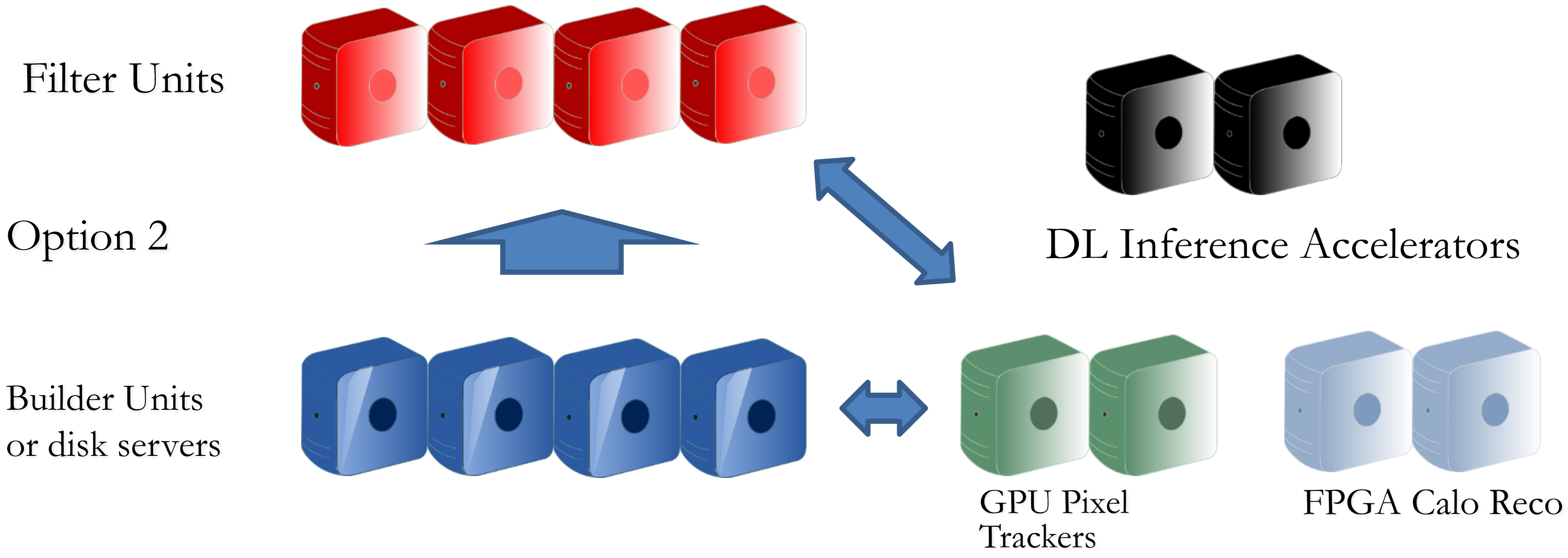
- Every FU is equipped with GPUs
  - tracking for every event



- Rigid design
  + easy to implement
  - Requires common acquisition, dimensioning etc

Integration in the Cloud/Farm

- A part of the farm is dedicated to a high density GPU cluster
- Tracks (or other physics objects like jets) are reconstructed on demand



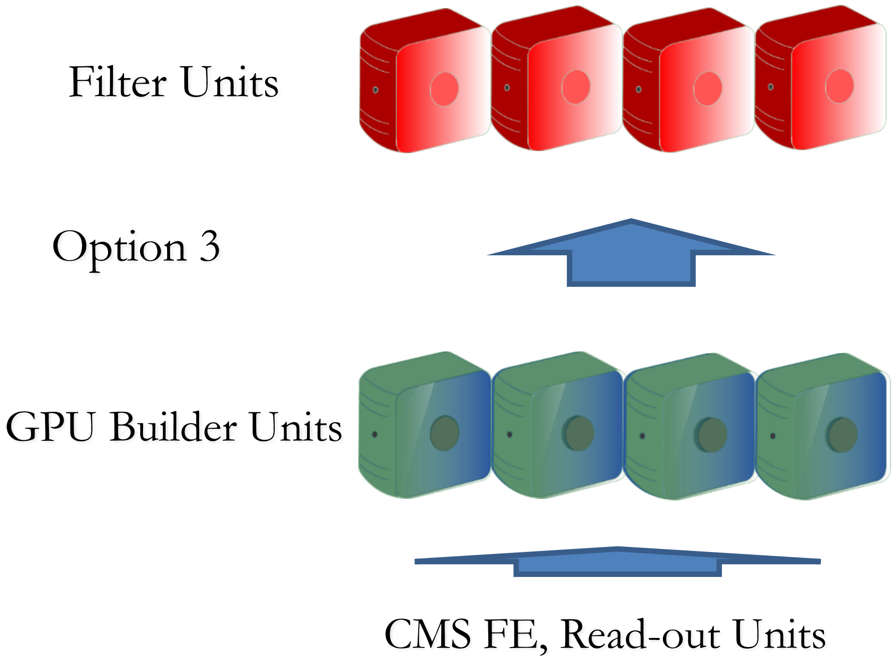
- Flexible design

  \+ Expandible, easier to balance

  \- Requires more communication and software development (e.g. HPX, MPI)

Integration in the HLT Farm

- Builder units are equipped with GPUs:
  - events with already reconstructed tracks are fed to FUs with GPUDirect
  - Use the GPU DRAM in place of ramdisks for building events.



- Very specific design

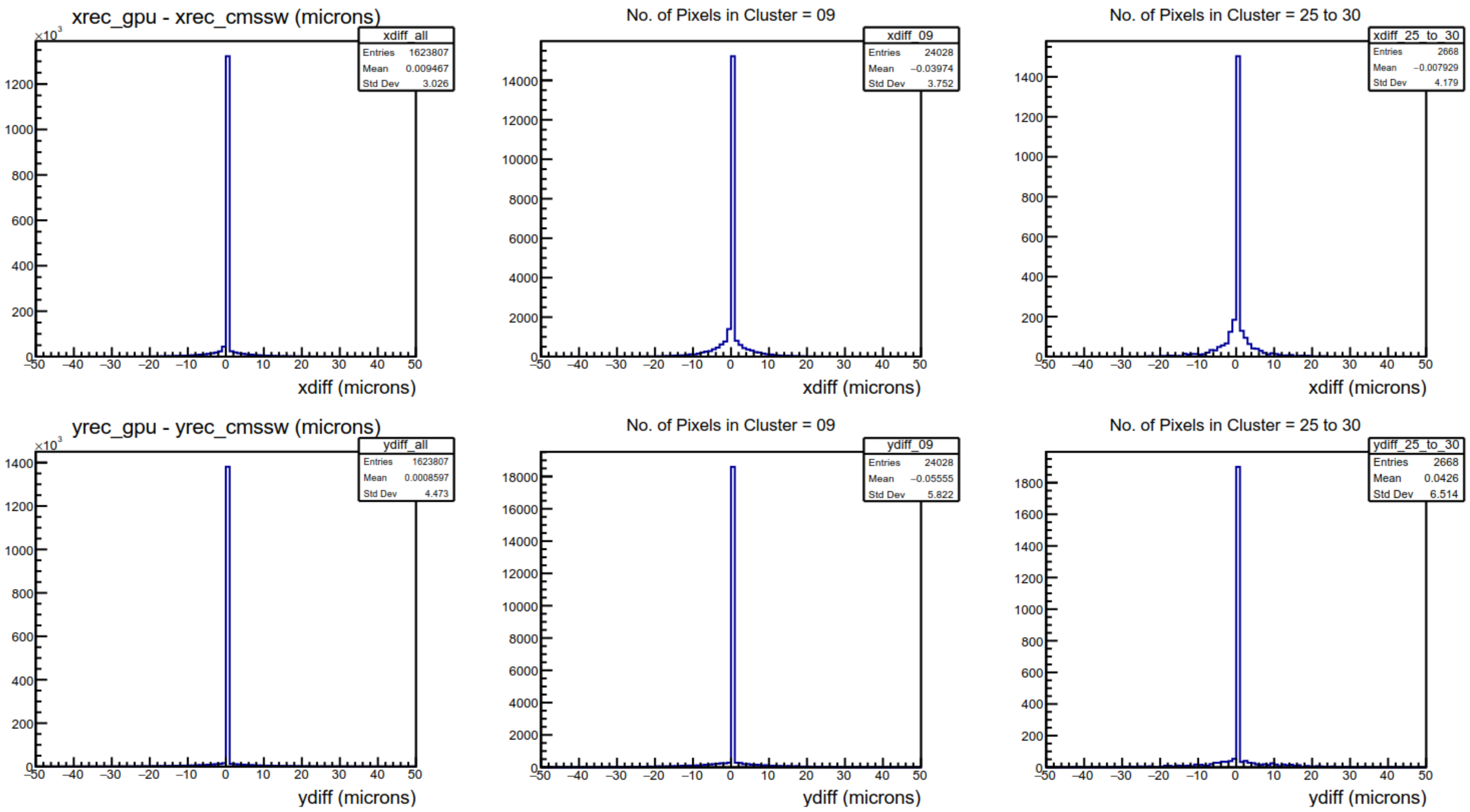
  + fast, independent of FU developments, integrated in readout

  - Requires specific DAQ software development: GPU “seen” as a detector element

Tests

GPU Pixel Clusterizer

- New Clusterizer algorithm
- Excellent agreement with CMSSW



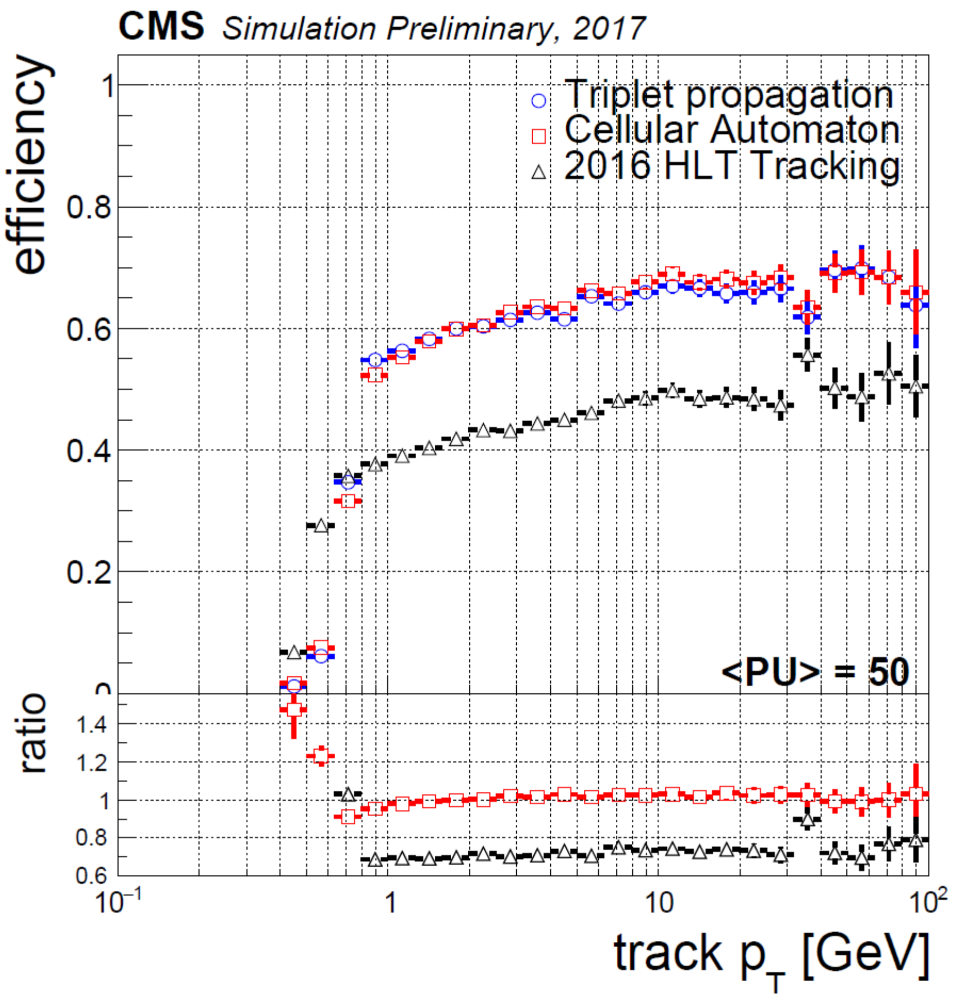
Raw To CPE

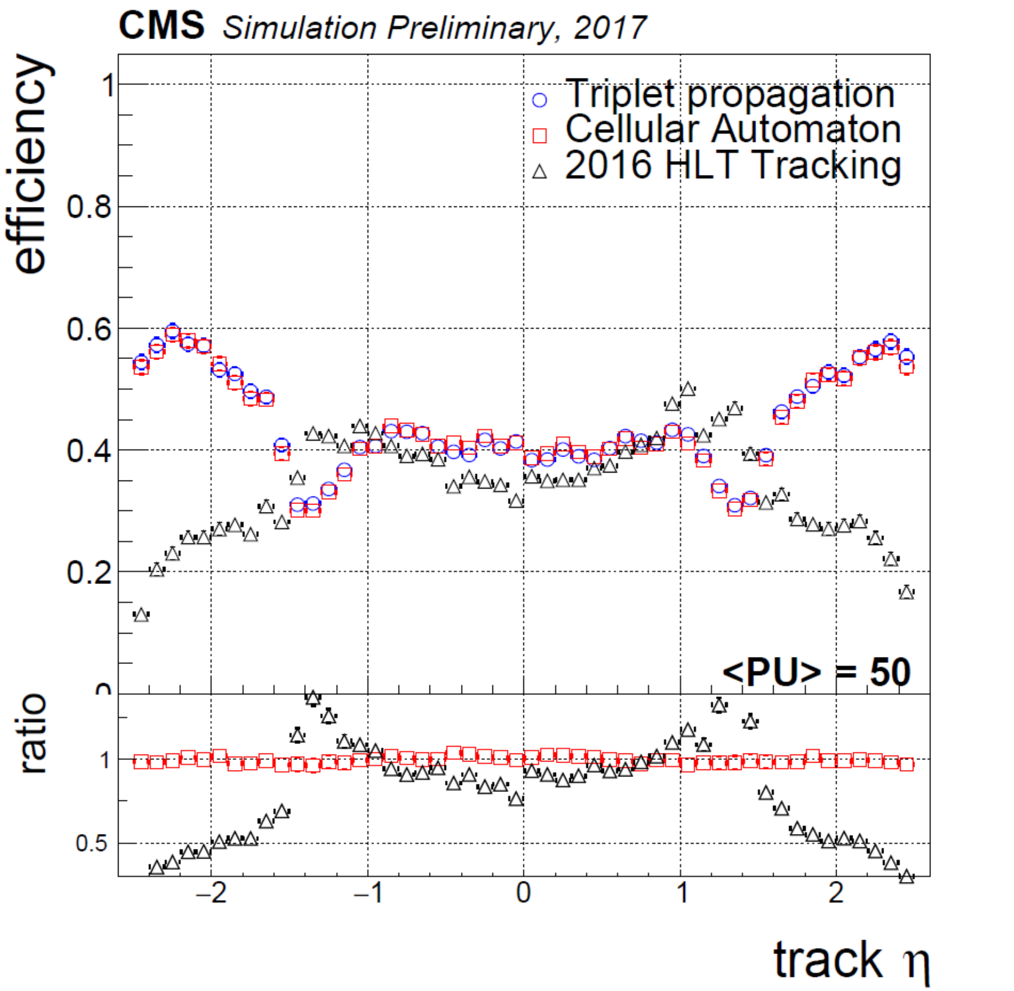
- GPU and CPU performance of RawToCPE on felk40 machine

GPU Time (ms)						
Event Size	Total pixel Hits	RawToDigi	Cluster	CPE	Total time	Time/Event
1	81001	0.19756	1.836	0.406	2.43956	2.43956
4	254282	1.215	6.76	1.086	9.061	2.2652
8	471554	1.5263	9.47658	1.59	12.59	1.5738
16	972836	5.7726	13.9083	2.804	22.4849	1.4053
32	1860016	7.285	20.6788	4.9358	32.8996	1.0281
64	3516714	7.645	34.55	8.97	51.165	0.7995
128	7002424	10.2067	67.8441	17.5768	**92.62**	0.7236

CPU Time (ms)*						
128	7002424	457.068	2696.32	282.984	3436.372	26.8466
Gain (CPU Time /GPU Time)		44.78	39.74	16.099	**37.101**	

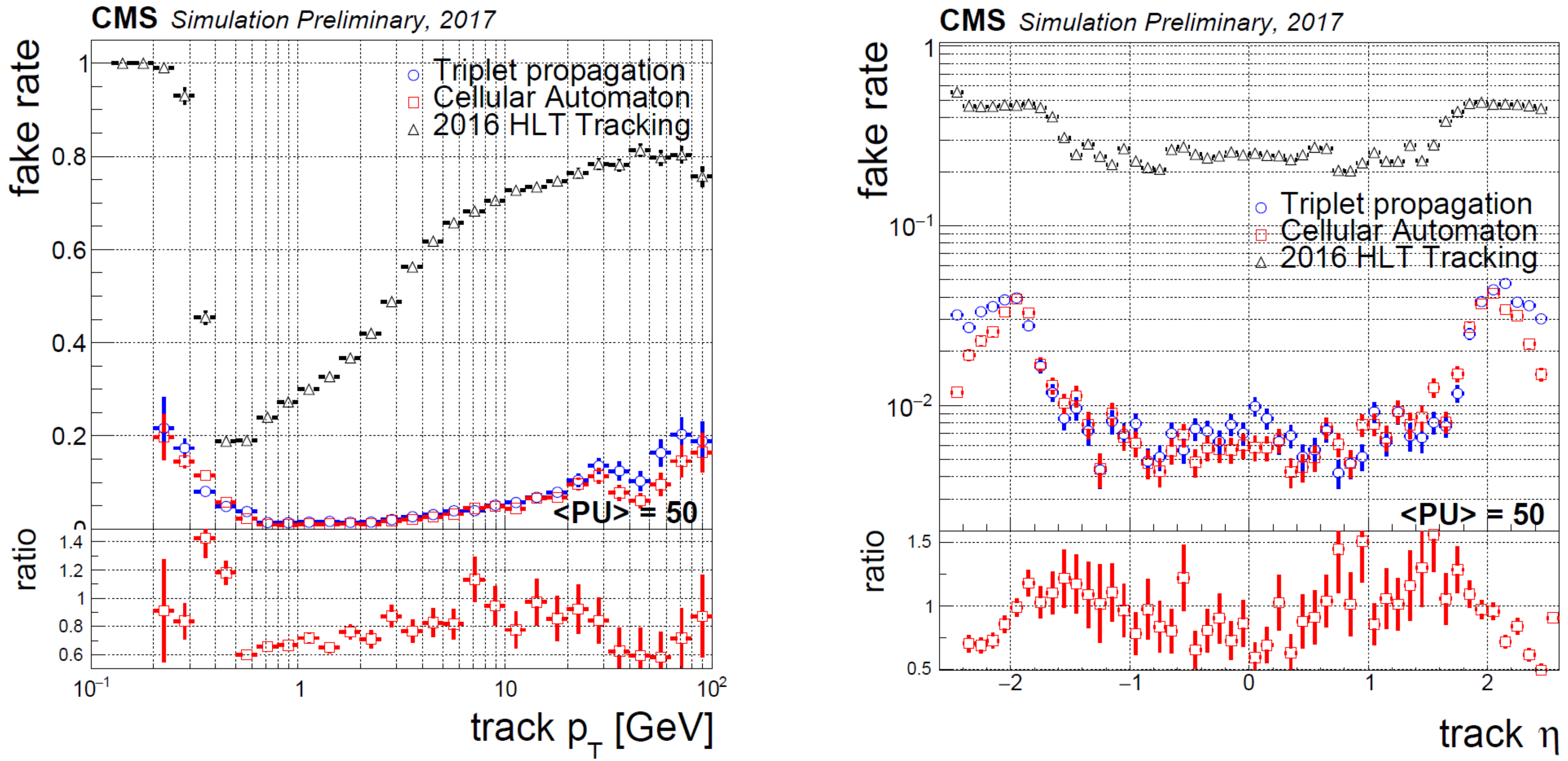
CA - Simulated Physics Performance PixelTracks





- CA tuned to have same efficiency as Triplet Propagation
- Efficiency significantly larger than 2016, especially in the forward region ($|\eta|>1.5$).

CA - Simulated Physics Performance PixelTracks



- Fake rate up to 40% lower than Triplet Propagation
- Two orders of magnitudes lower than 2016 tracking thanks to higher purity of quadruplets wrt to triplets

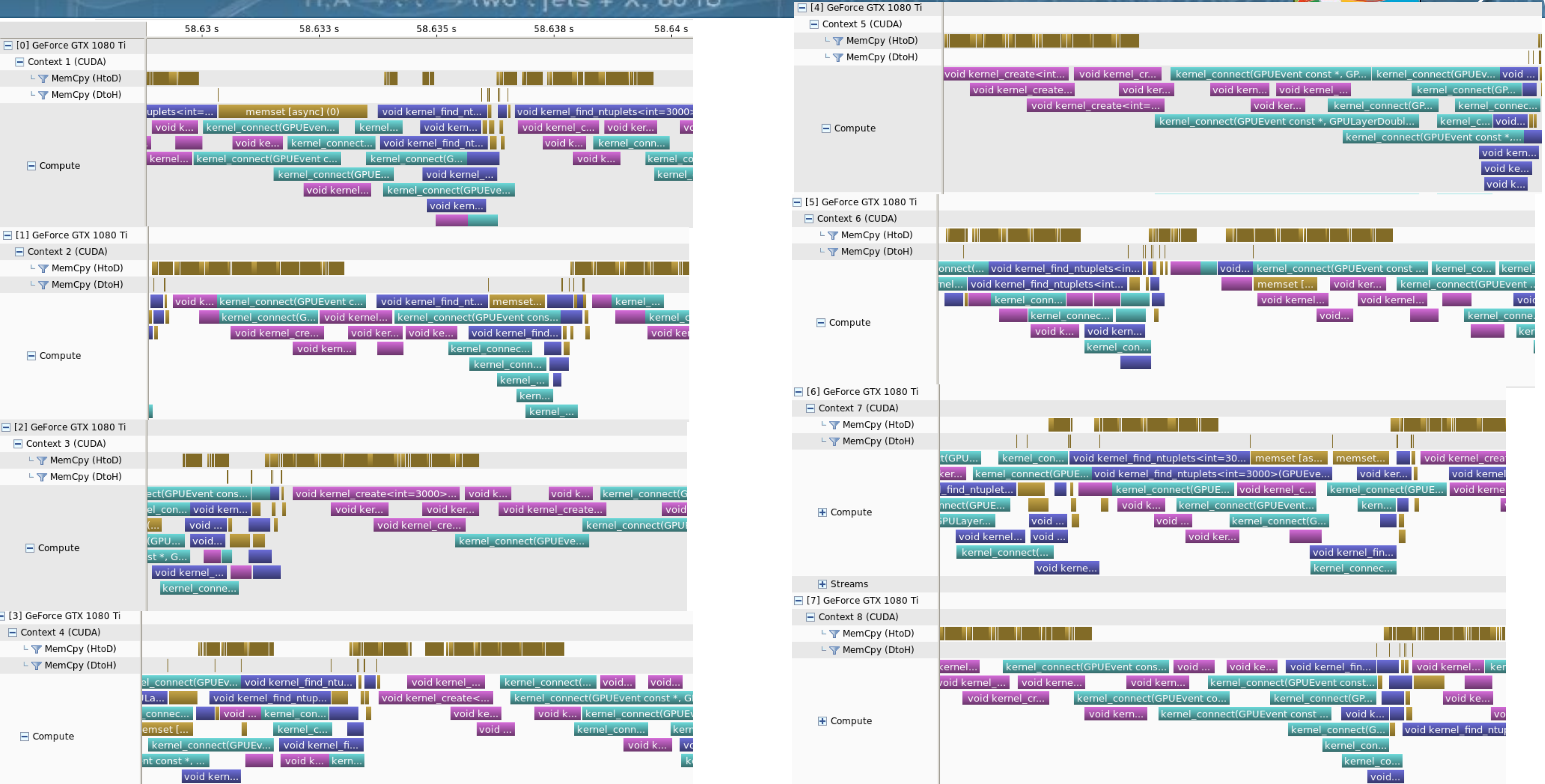
Hardware on the bench

- We acquired a small machine for development and testing:
  - 2 sockets x Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz (12 physical cores)
  - 256GB system memory
  - 8x GPUs NVIDIA GTX 1080Ti
  - Total cost: 5x 🍌

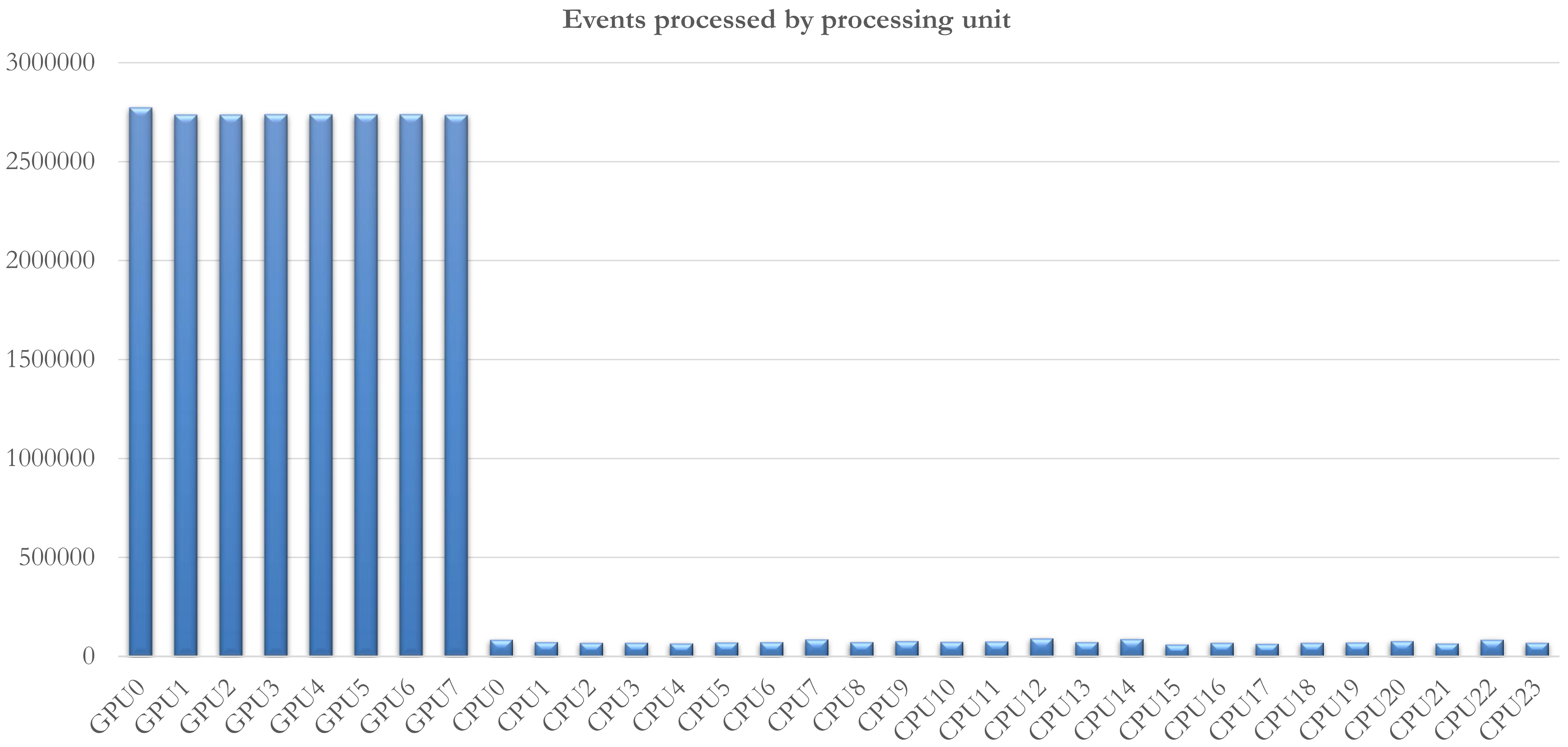
Rate test

- The rate test consists in:
  - preloading in host memory few hundreds events
  - Assigning a host thread to a host core
  - Assigning a host thread to a GPU
  - Preallocating memory for each GPU for each of 8 cuda streams
  - Filling a concurrent queue with event indices
  - During the test, when a thread is idle it tries to pop from the queue a new event index:
    - Data for that event are copied to the GPU (if the thread is associated to a GPU)
    - processes the event (exactly same code executing on GPUs and CPUs)
    - Copy back the result
  - The test ran for approximately one hour
  - At the end of the test the number of processed events per thread is measured, and the total rate can be estimated

What happens in 10ms



Rate test

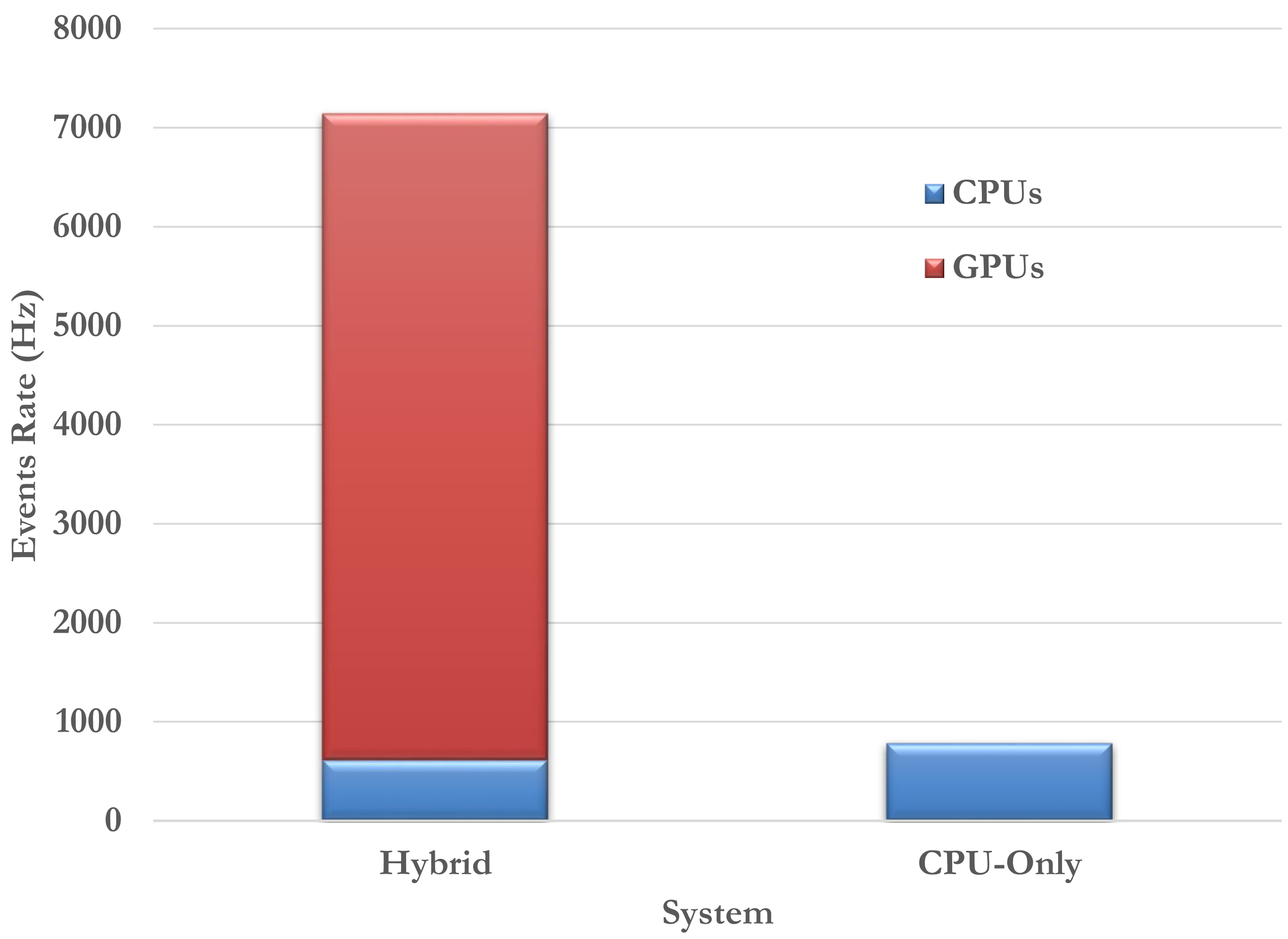


Events processed by processing unit
3000000
2500000
2000000
1500000
1000000
500000
0
GPU0
GPU1
GPU2
GPU3
GPU4
GPU5
GPU6
GPU7
CPU0
CPU1
CPU2
CPU3
CPU4
CPU5
CPU6
CPU7
CPU8
CPU9
CPU10
CPU11
CPU12
CPU13
CPU14
CPU15
CPU16
CPU17
CPU18
CPU19
CPU20
CPU21
CPU22
CPU23

Rate test

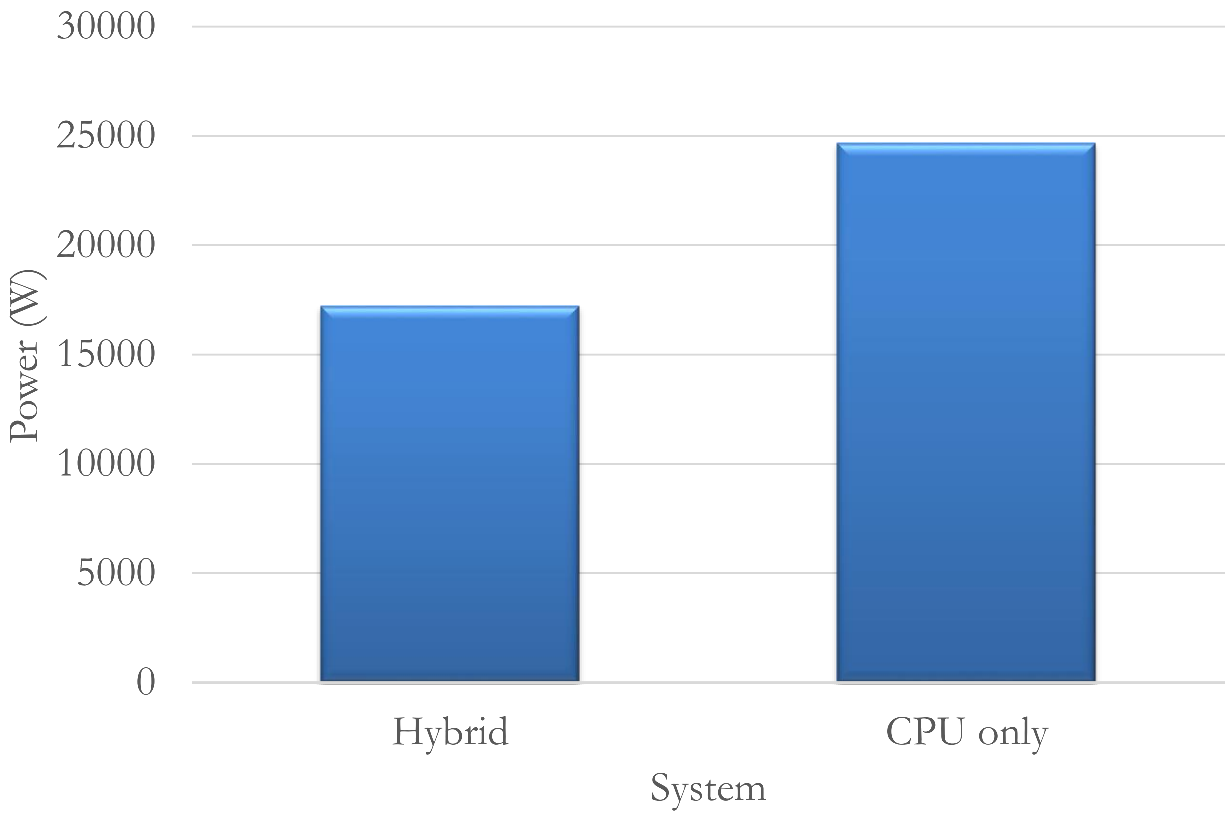
- Total rate measured:
  - 8xGPU: 6527 Hz
  - 24xCPUs: 613 Hz
- Number of nodes to reach 100kHz: ~14
- Total Price: 70x

- When running with only 24xCPUs
  - Rate with 24xCPUs: 777 Hz
- Number of nodes to reach 100kHz: ~128
- Total Price: 320x
  - Assuming an initial cost of 2.5 per node



Energy efficiency

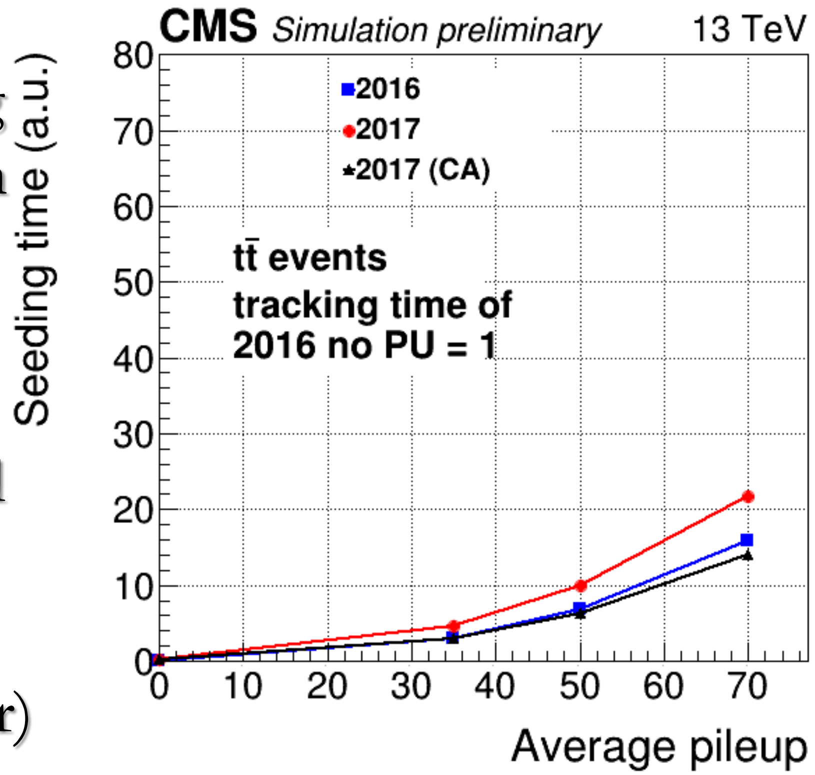
- During the rate test power dissipated by CPUs and GPUs was measured every second
  - Nvidia-smi for GPUs
  - Turbostat for CPUs
- 8 GPUs: 1037W
  - 6.29 Events per Joule
  - 0.78 Events per Joule per GPU
- 24 CPUs in hybrid mode: 191W
  - 3.2 Events per Joule
  - 0.13 Events per Joule per core
- 24 CPUs in CPU-only test: 191W
  - 4.05 Events per Joule
  - 0.17 Events per Joule per core



- That is 1/3 more 🍌s in the energy bill when processing 100kHz input

Algorithmic Innovation benefits offline reco

- CA track seeding at same level of the 2016 seeding
- More robust, smaller complexity vs PU than 2016 track seeding despite the increased number of layer combinations involved in the seeding phase with respect to the 2016 seeding
- ~25% faster track reconstruction wrt to 2016 tracking at avg PU70
- Replacing the CMS Phase2 offline track seeding with sequential CA
  - Overall tracking 2x faster at PU200
  - T(PU=200 – Phase2 detector) = 4xT(PU50 – 2017 detector)
    - Detector and algorithms defeated combinatorial complexity
- Innovation at algorithmic level often underestimated
  - We believe algorithmic modernization should be more encouraged and promoted by CMS



Conclusion

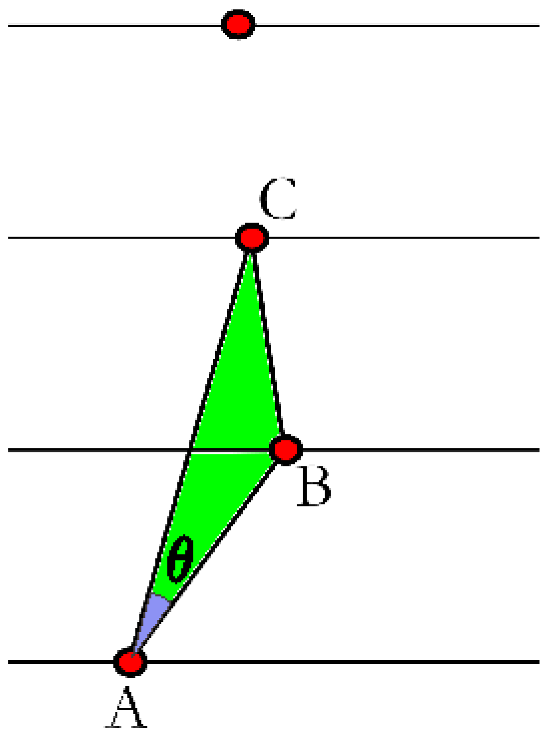
- Pixel Track seeding algorithms have been redesigned with high-throughput parallel architectures in mind
- Improvements in performance may come even when running sequentially
  - Factors at the HLT, tens of % in the offline, depending on the fraction of the code that use new algos
- Graph-based algorithm are very powerful
  - By adding more Graph Theory sugar, steal some work from the track building and become more flexible
- The GPU and CPU algorithms run in CMSSW and produce the same bit-by-bit result
  - Transition to GPUs@HLT during Run3 smoother
- Running Pixel Tracking at the CMS HLT for every event would become cheap @PU ~ 50 – 70
  - Integration in the CMS High-Level Trigger farm under study
- DNNs under development for early-rejection of doublets based on their cluster shape and track classification

Questions?

Back up

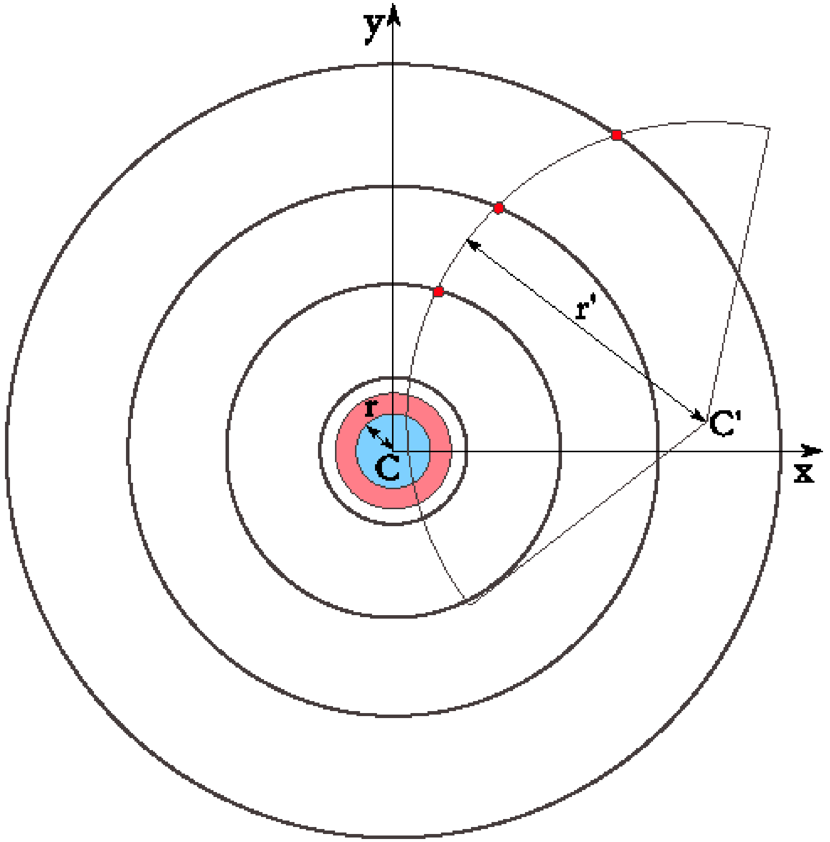
CA: R-z plane compatibility

- 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



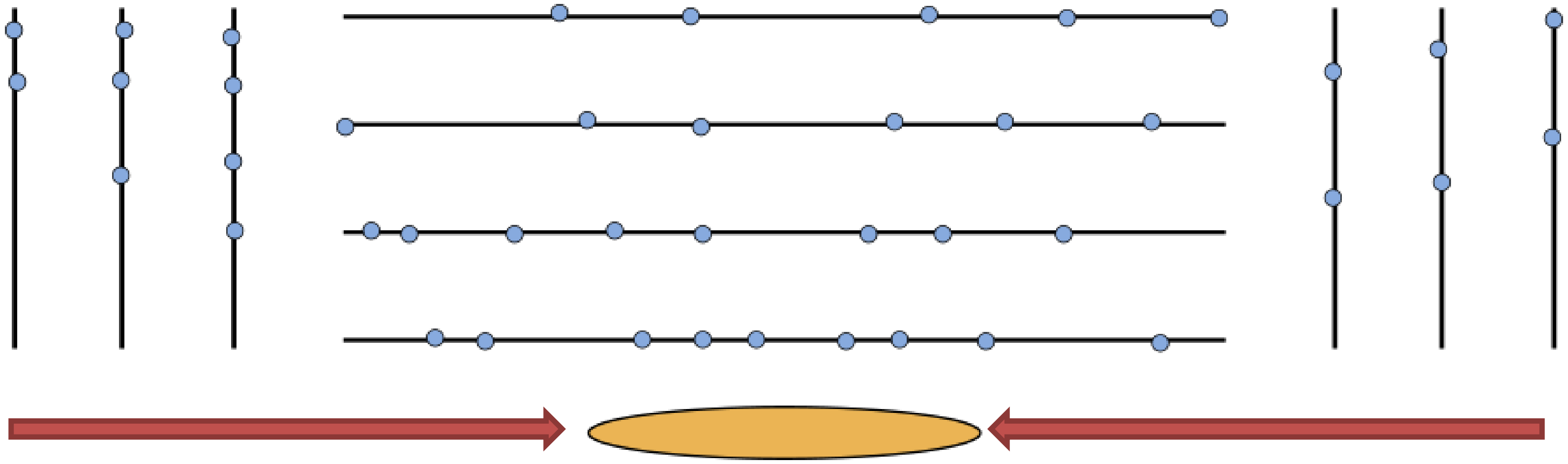
CA: x-y plane compatibility

- 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'



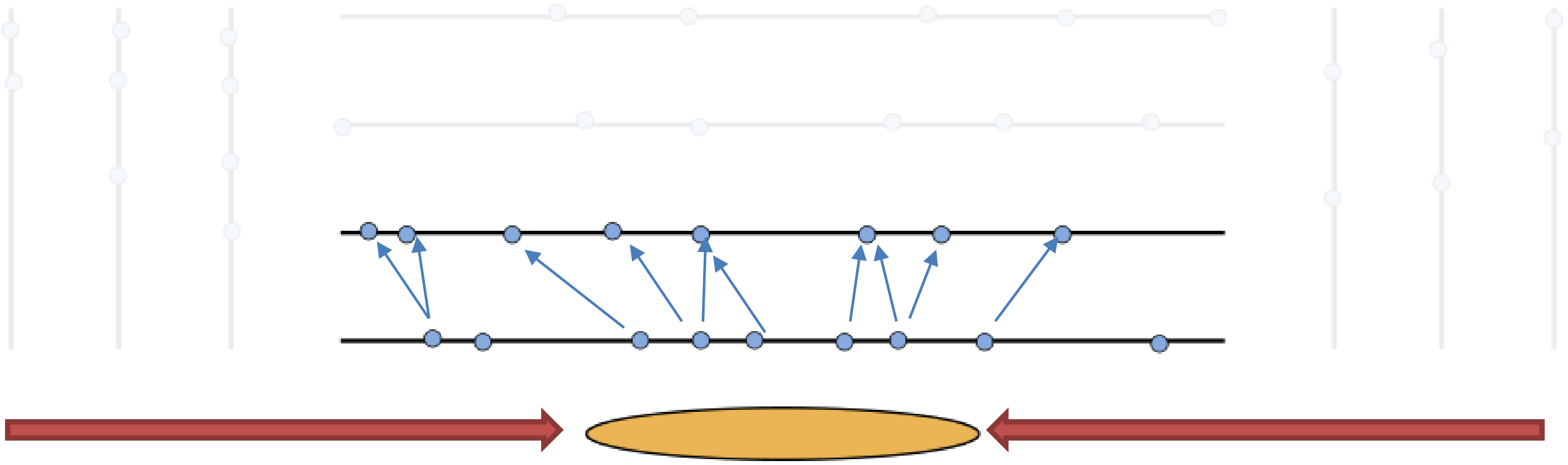
RMS HEP Algorithm

- Hits on different layers
- Need to match them and create quadruplets
- Create a modular pattern and reapply it iteratively



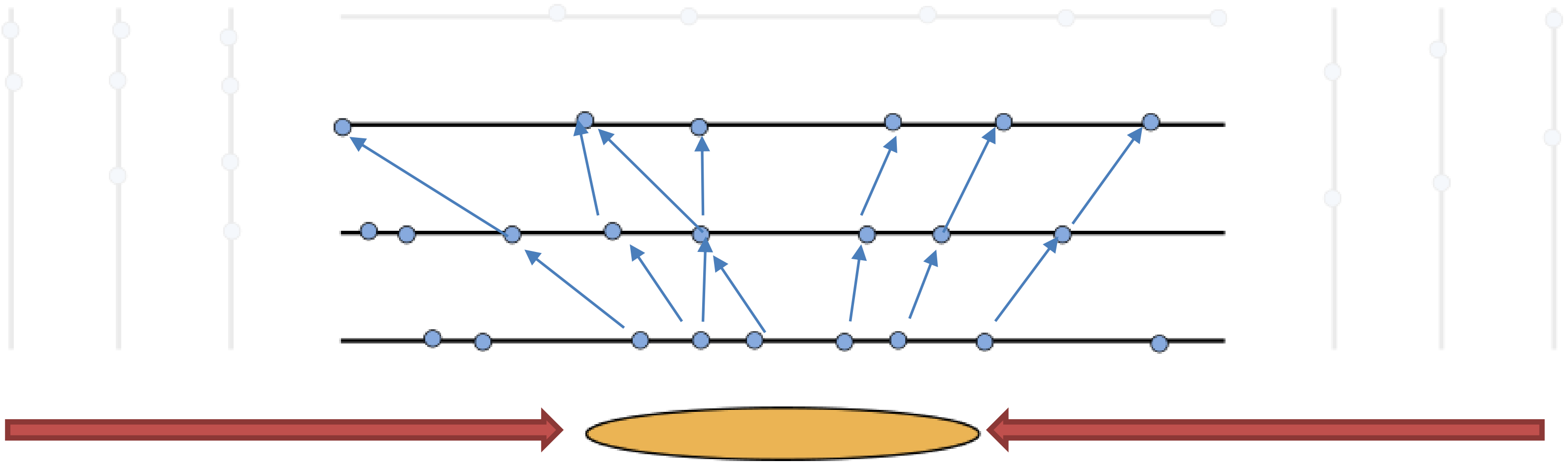
RMS HEP Algorithm

- First create doublets from hits of pairs



RMS HEP Algorithm

- First create doublets from hits of pairs
- Take a third layer and propagate only the generated doublets



RMS HEP Algorithm

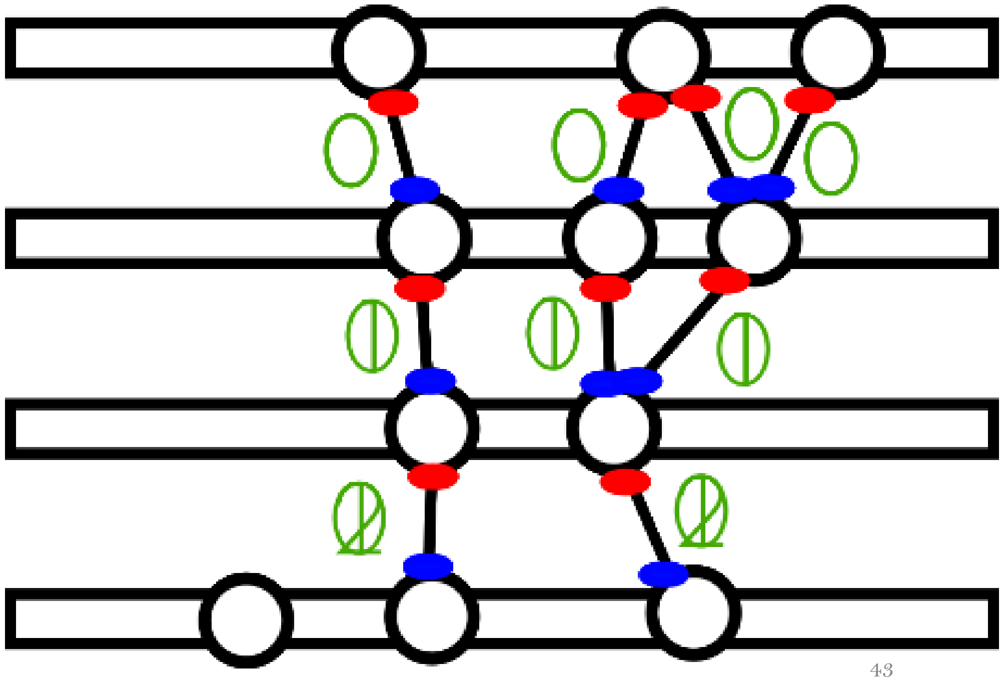
This kind of algorithm is not very suitable for GPUs:

- Absence of massive parallelism
- Poor data locality
- Synchronizations due to iterative process
- Very Sparse and dynamic problem (that's the hardest part, still unsolved)
- Parallelization does not mean making a sequential algorithm run in parallel
  - It requires a deep understanding of the problem, renovation at algorithmic level, understanding of the computation and dependencies

The algorithm was redesigned from scratch getting inspiration from Conway's Game of Life

- Traditional Cellular Automata excluded because 2x slower
  - quadruplets by triplets sharing a doublet

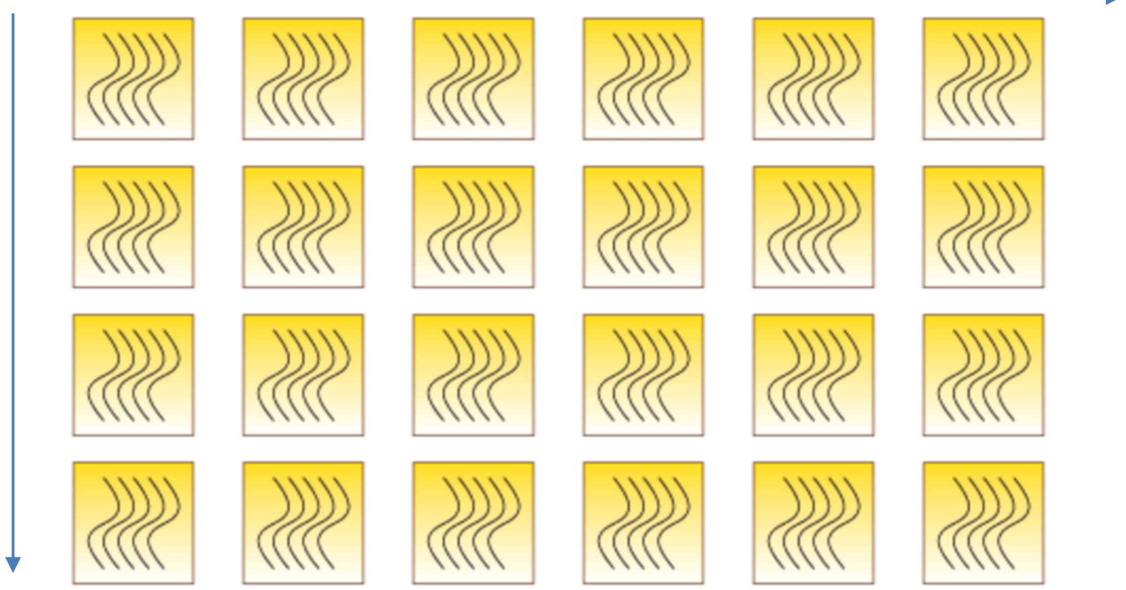
T=0



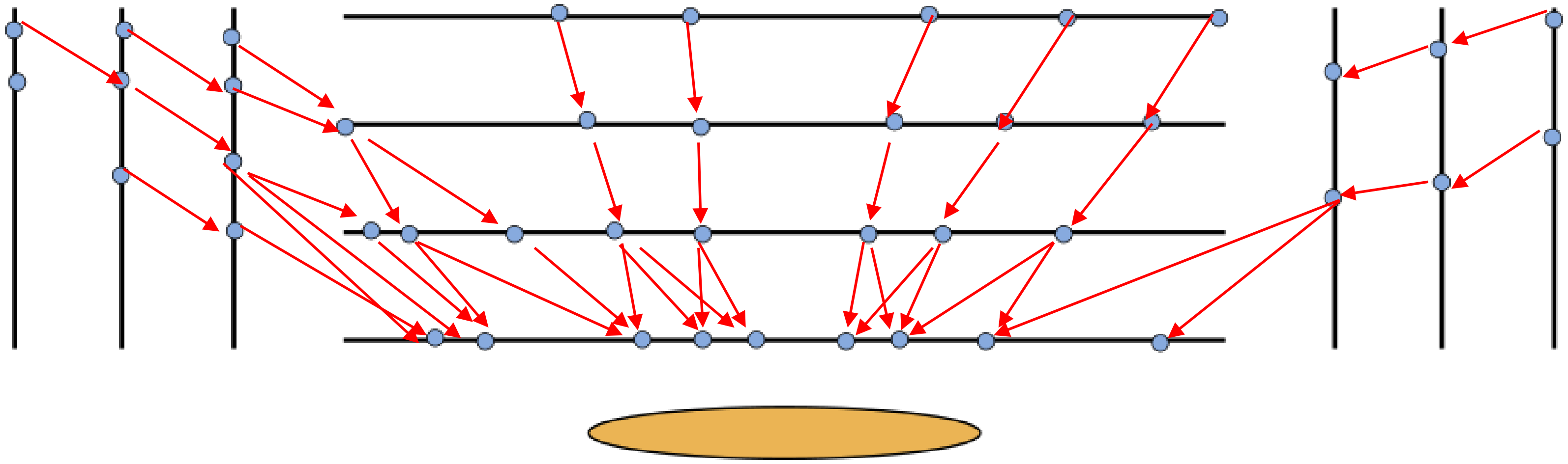
Cells Connection

blockIdx.x and threadIdx.x = Cell id in a LayerPair

blockIdx.y = LayerPairIndex [0,13)



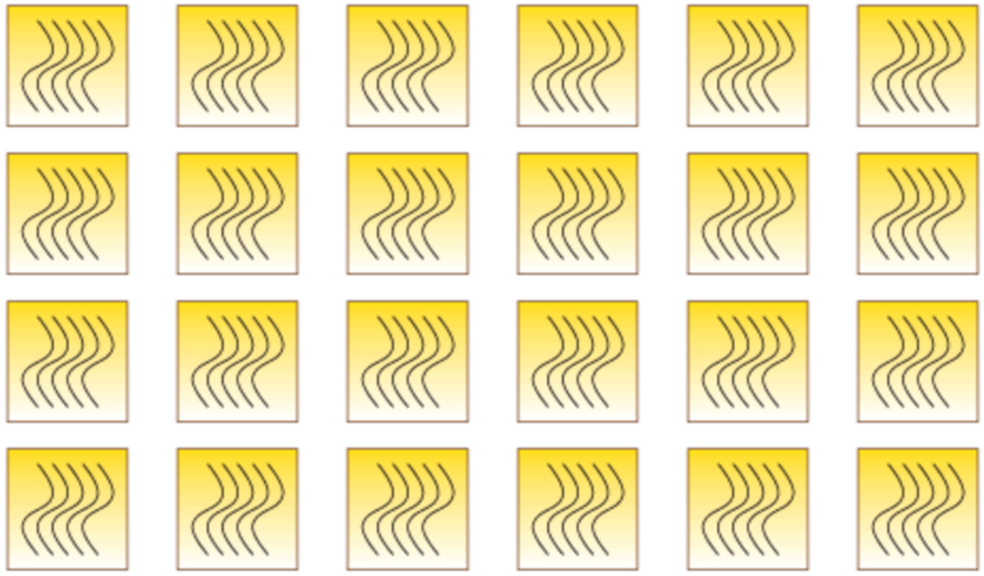
Each cell asks its innermost hits for cells to check compatibility with.



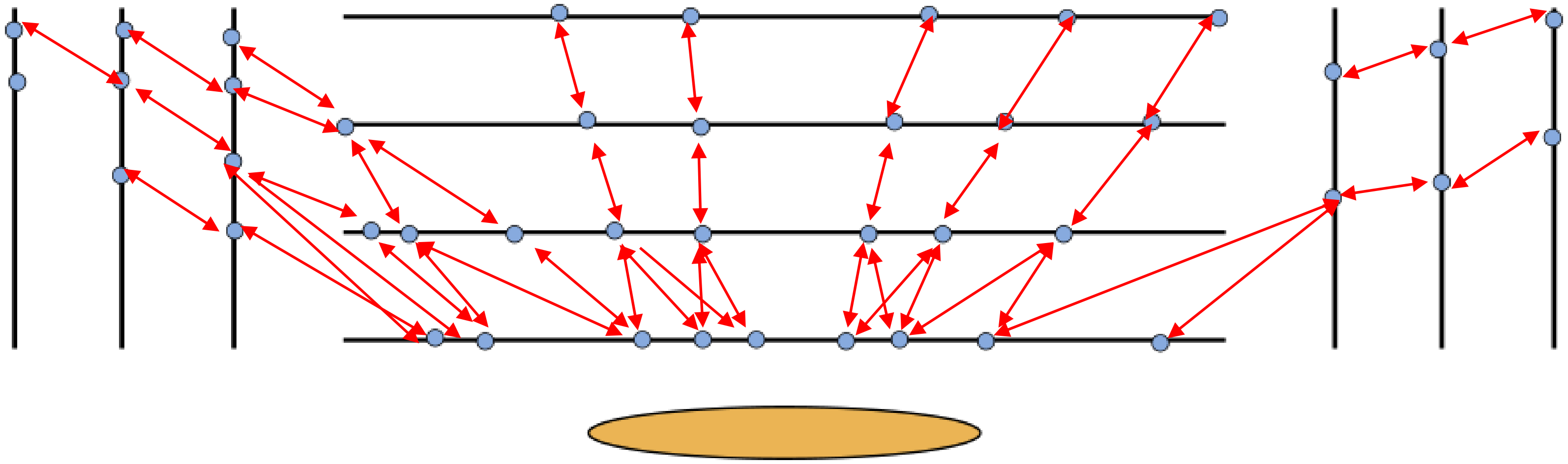
Quadruplets finding

blockIdx.x and threadIdx.x = Cell id in a Root LayerPair

blockIdx.y = LayerPairIndex in RootLayerPairs



Each cell on a root layer pair will perform a parallel DFS of depth = 4 following outer neighbors.



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.