# Patatrack: Accelerated Pixel Track reconstruction in CMS

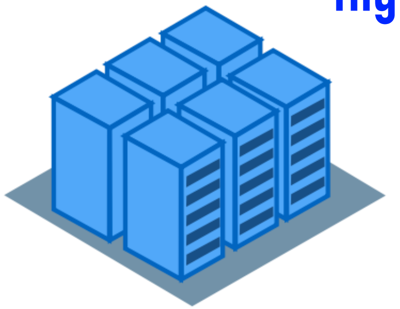
Adriano Di Florio on behalf of the Patatrack Team

Felice Pantaleo, Maurizio Pierini, Vincenzo Innocente, Marco Rovere, Andrea Bocci, Matti Kortelainen

# Two Level Event Selection @ CMS

## L1 TRIGGER

- 40 MHz input / 100 KHz output
- Processing time: ~10 µs
- Coarse local reconstruction
- FPGAs / Hardware implemented

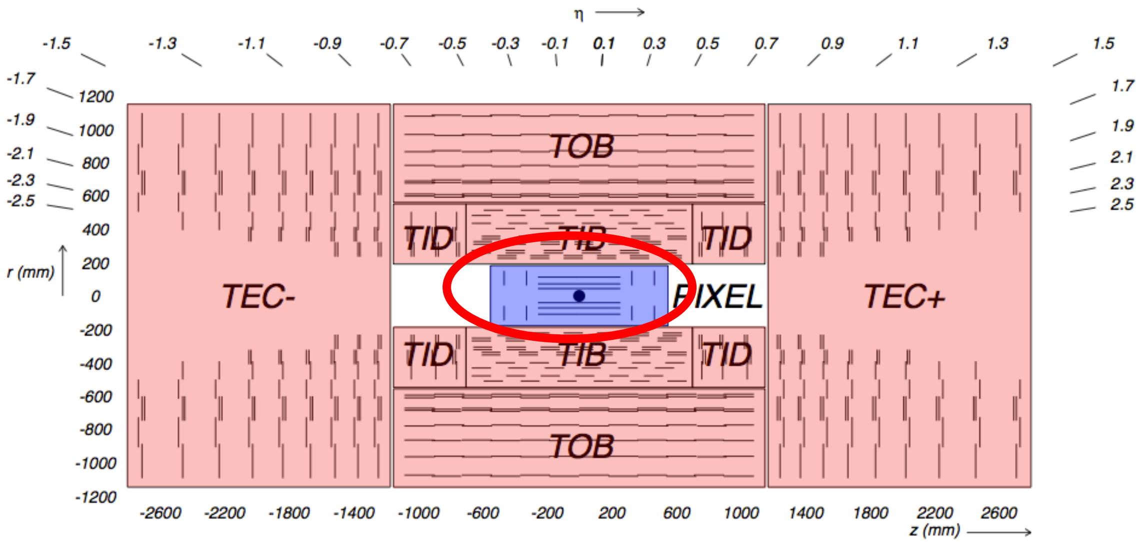


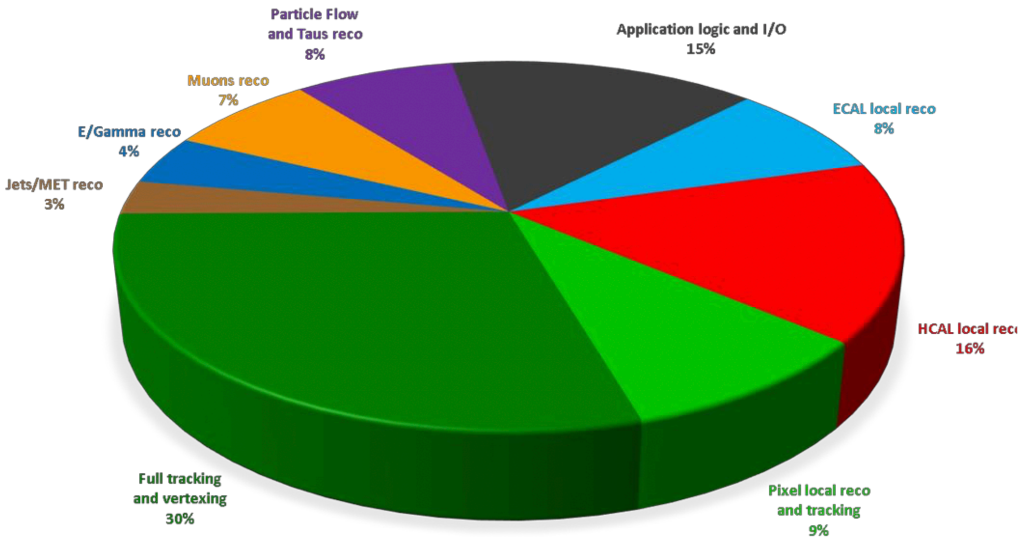
## High Level Trigger (HLT)

- 100 KHz in / 1 KHz out
- 500 KB / event
- Processing time: ~30 ms
- Simplified global reconstruction
- Software implemented on CPUs

TO STORAGE

In CMS, the tracking algorithm consists of an iterative procedure, in which tracks are reconstructed according to progressively looser quality criteria starting from hits on the silicon tracker detector.



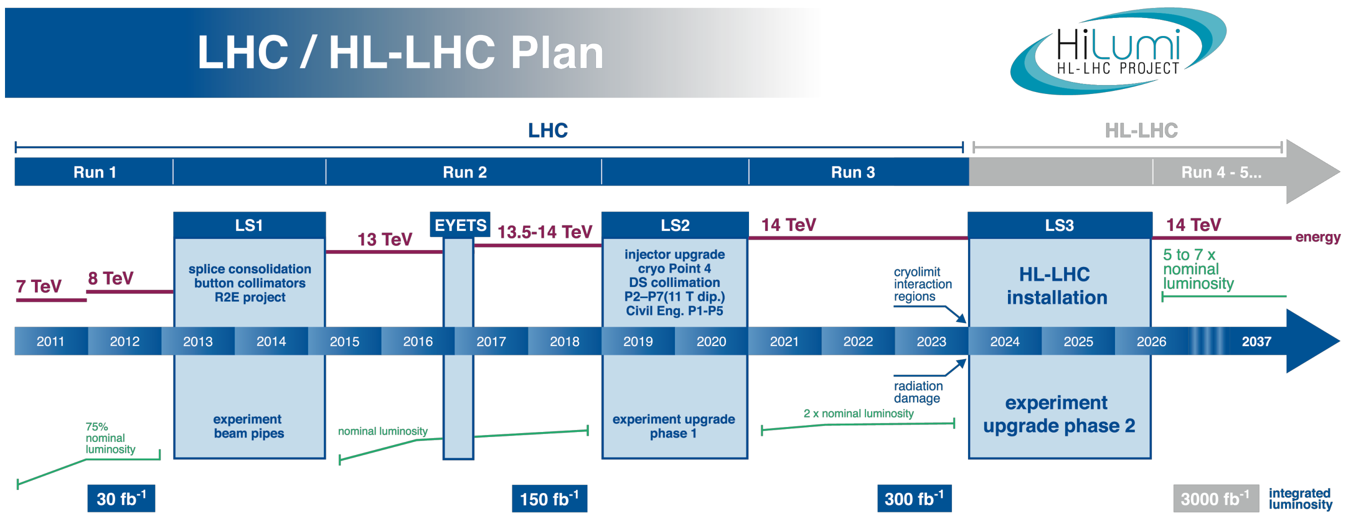




## ONLINE RECONSTRUCTION (HLT)

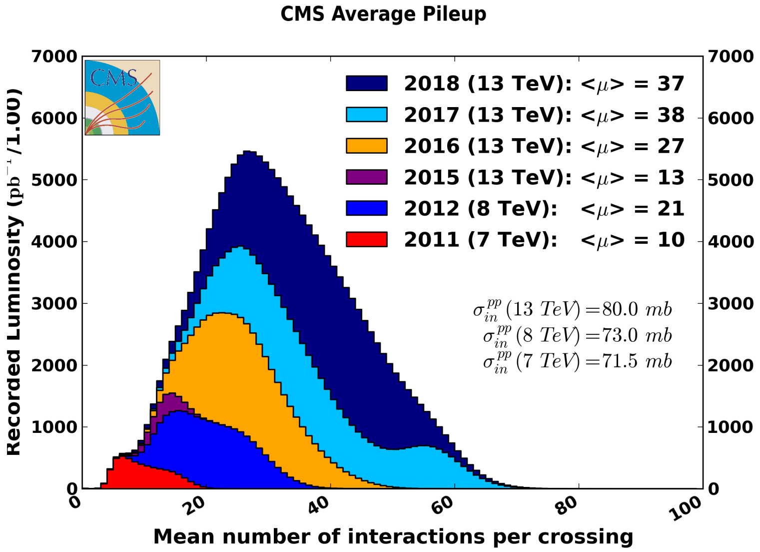
Practically the same reconstruction procedure as the one run offline. It has to undergo stringent time limits : O(100) ms. Some iterations are on based pixel-only reconstruction and the very first step (seeding) consists in building compatible hit pairs.

# Four Horsemen

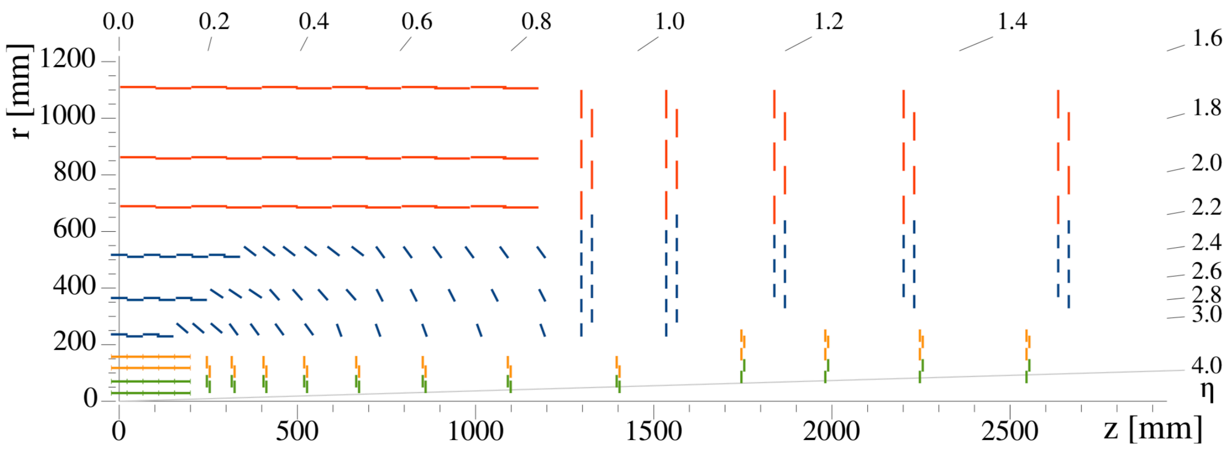
## Istantaneous luminosity: $\mathcal{L} = 5 \cdot 10^{34} cm^2 s^{-1}$




## Pile Up: $<PU>\sim 200$



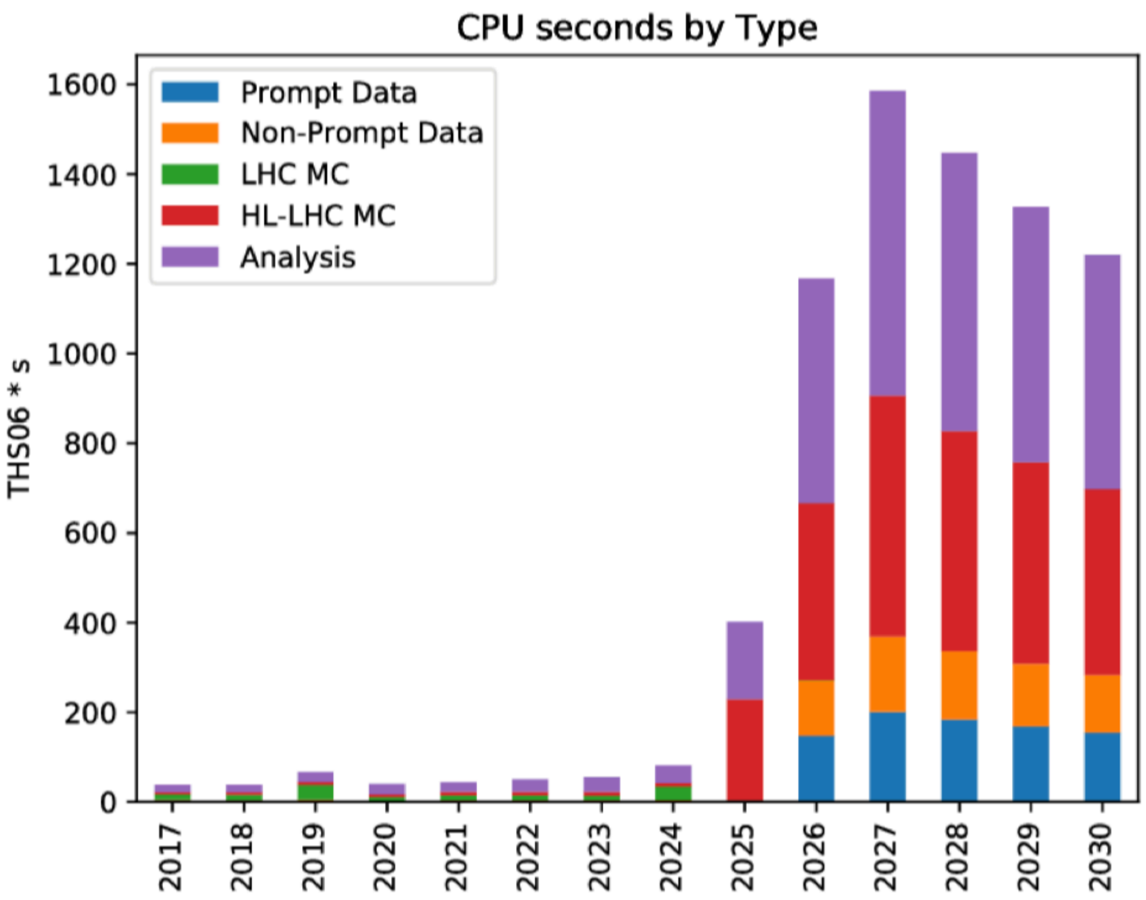

## Pixel Detector: 24 Layers $(4 + 10 + 10)$

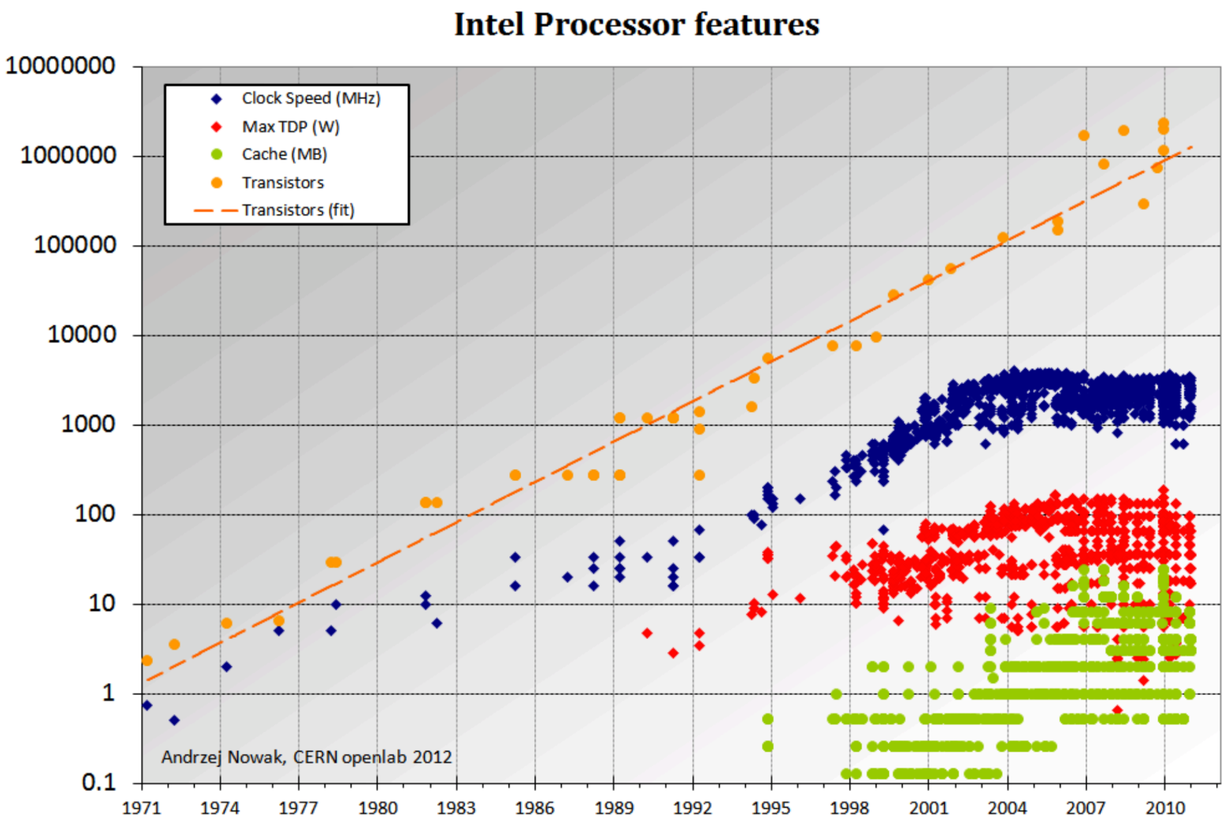



## L1 Trigger Rate: 750kHz

# Legacy Computing Model

Continuing to pursue the computing model used since now is unfeasible.

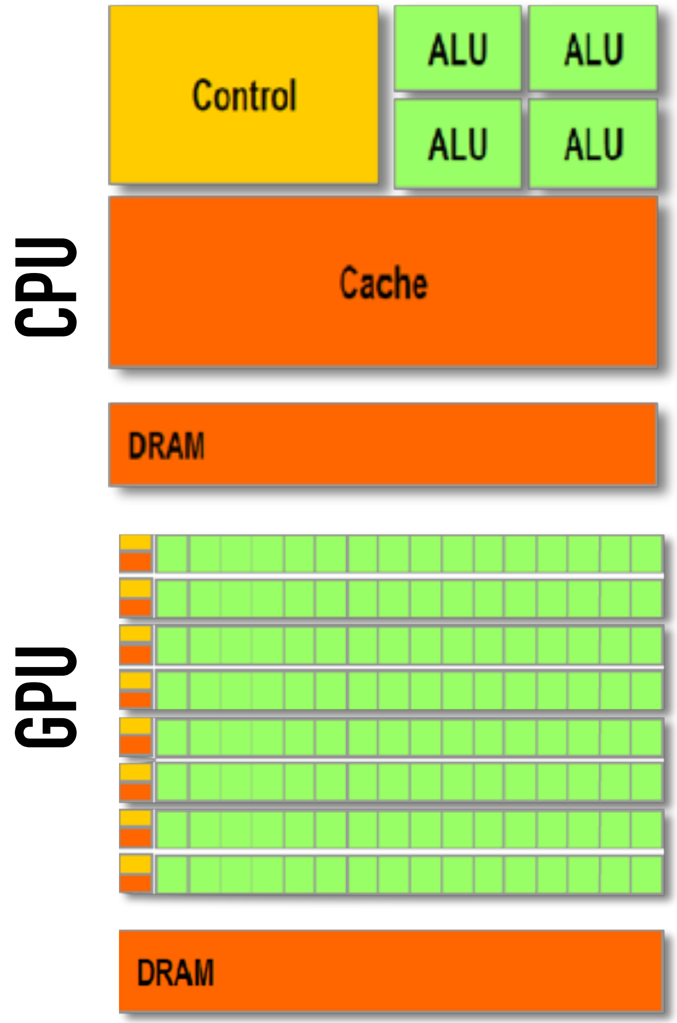



A new approach is neeed . . .

# Patatrack – Heterogenous Computing

Patatrack is a software R&D incubator born in 2016 by a very small group of passionate people. Interests: algorithms, HPC, heterogeneous computing, machine learning, software engineering.




Main goal: to lay the foundations of the online/offline heterogeneous reconstruction starting from 2020s

What's a GPU?

- 1970s: first graphical user interface produced requiring dedicated microchips
- Video games and 3D graphics: strong economic stimulus for GPU development

Consequences on GPU architecture:

Many [O(1000)] Streaming Multiprocessors execute parallel functions using thousands of threads concurrently

Handle big loads of data

Low frequency clock (~1GHz)

Arithmetical operations in a single clock cycle (*sin,cos,sqrt,1/x, ...*)

Higher bandwidth to memory (up to 1TB/s)

# Patatrack Pixel Reconstruction Workflow

The idea, for CMS, would be to unroll and offload each event's combinatorics to many threads in parallel.

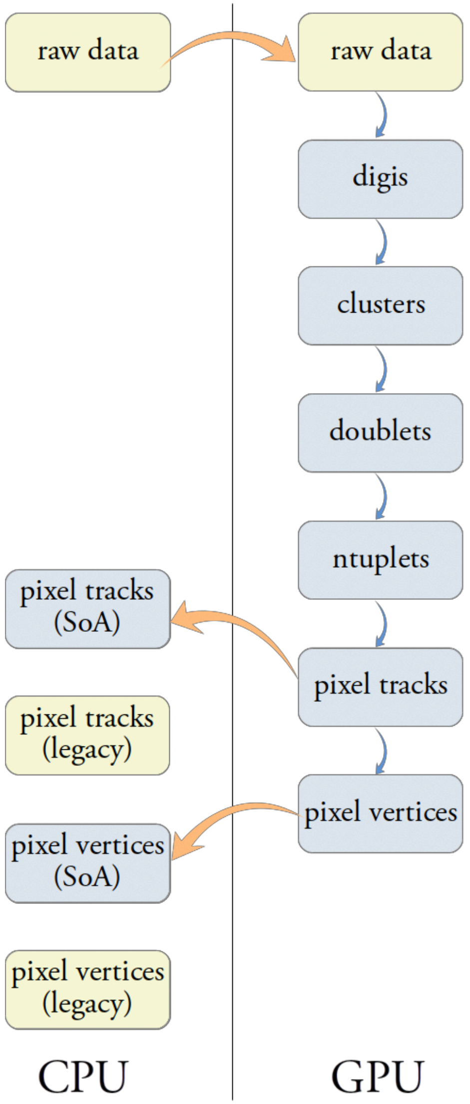

So, within the Patatrack group, the Full Pixel Track reconstruction has been reinplemented and integrated in CMSSW from Raw data decoding to Primary Vertices determination

Raw data for each event is transferred to the GPU initially (~250 kB/event)

At each step data can be transferred to CPU and used to populate "legacy" event data

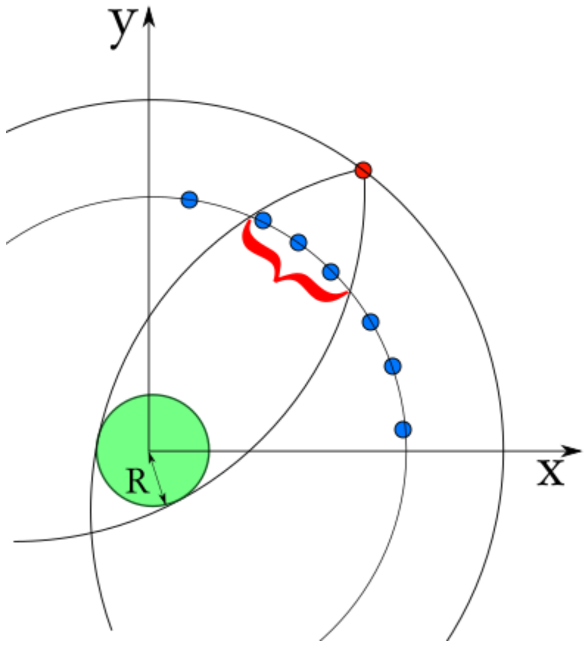
The standard validation is fully supported

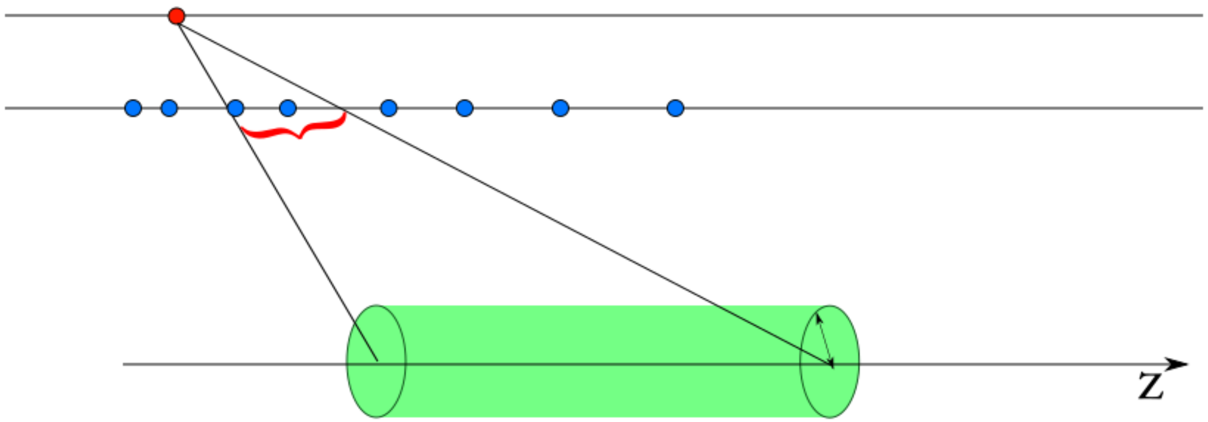
Integer results are identical and small differences in the results of floating point can be explained by differences in re-association



# Doublets

Once the local reconstruction produces hits; doublets are created opening a window depending on the tracking region/beamspot and layer-pair. The cluster size along the beamline can be required to exceed a minimum value for barrel hits connecting to an endcap layer




Hits within the bins are connected to form doublets if they pass further "alignment cuts" based on their actual position

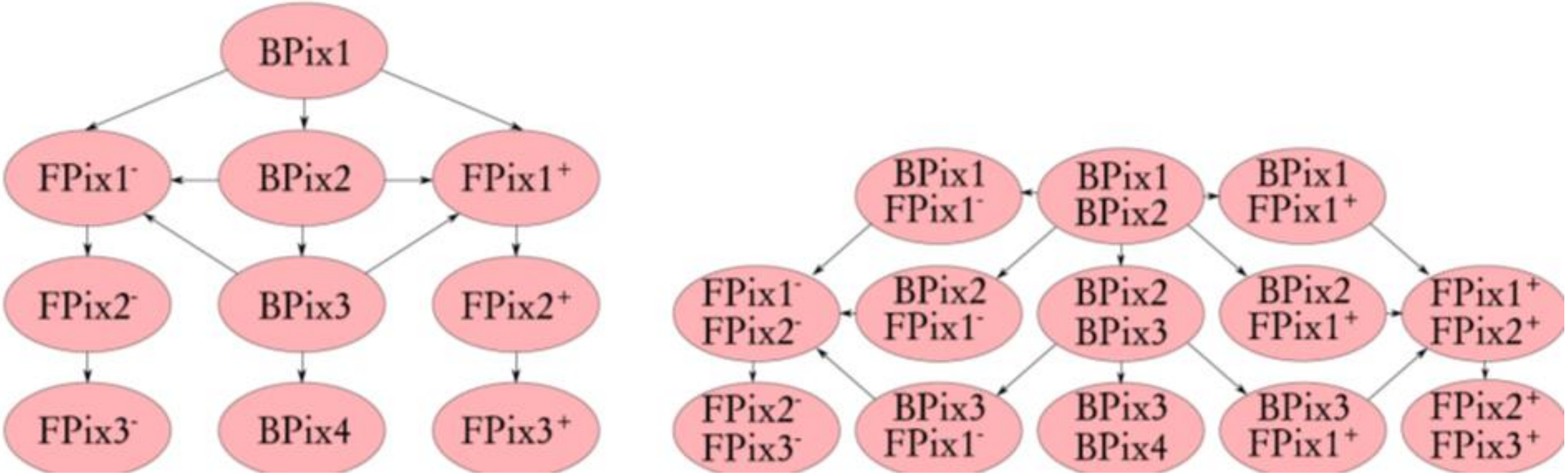
- In the barrel the compatibility of the cluster size along the beamline between the two hits can be required

These geometrical cuts reduce the number of doublets by an order of magnitude and the combinatorics by a factor 50

# CA Hit Chain Maker

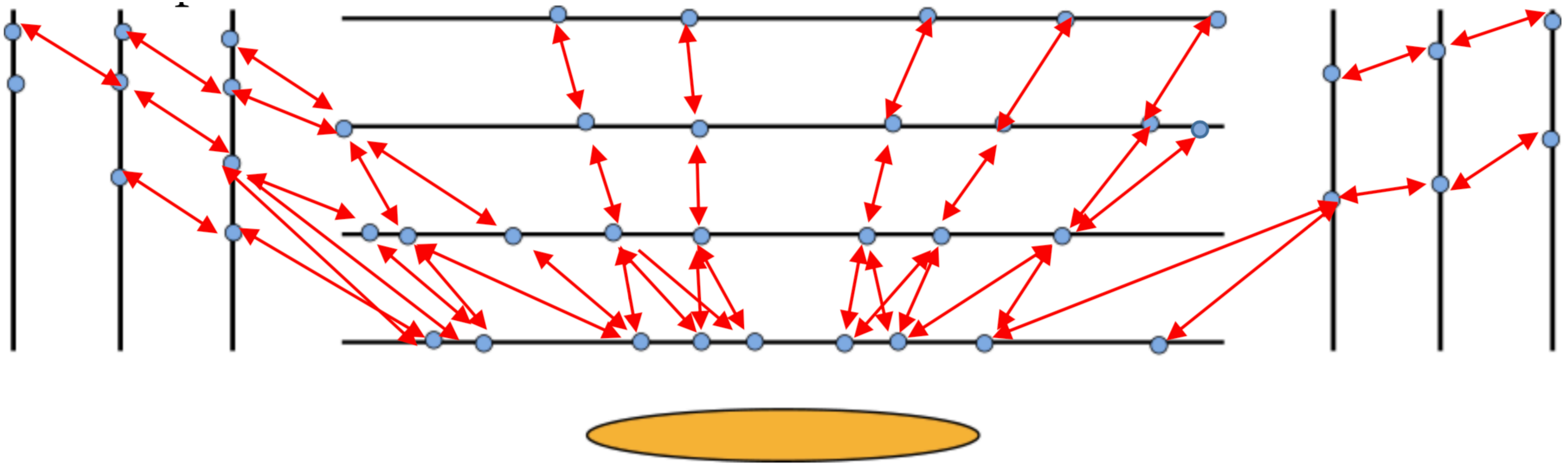
The CA is a track seeding algorithm designed for parallel architectures and it requires a list of (all) layers and their pairings.

1. As first step a graph of all the possible connections between layers is created
2. Doublets aka Cells are created for each pair of layers, in parallel at the same time. Hit doublets for each layer pair can be computed independently by sets of threads



The CA is a track seeding algorithm designed for parallel architectures and it requires a list of (all) layers and their pairings.

3. Fast computation of the compatibility between two connected cells, in parallel.
4. No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize.

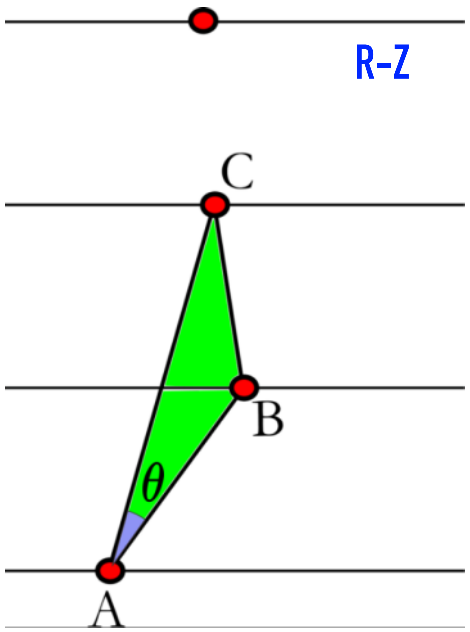


Better efficiency and fake rejection wrt previous algorithm. Since 2017 data-taking has become the default track seeding algorithm for all the pixel-seeded online and offline iterations. In the following, at least four hits are required, but triplets can be kept to recover efficiency where geometric acceptance lacks one hit .

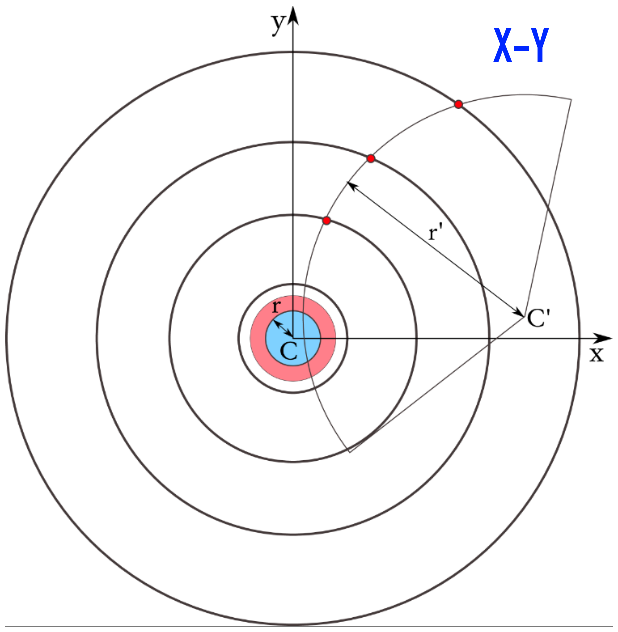
# CA Compatibility Cuts & Fishbone

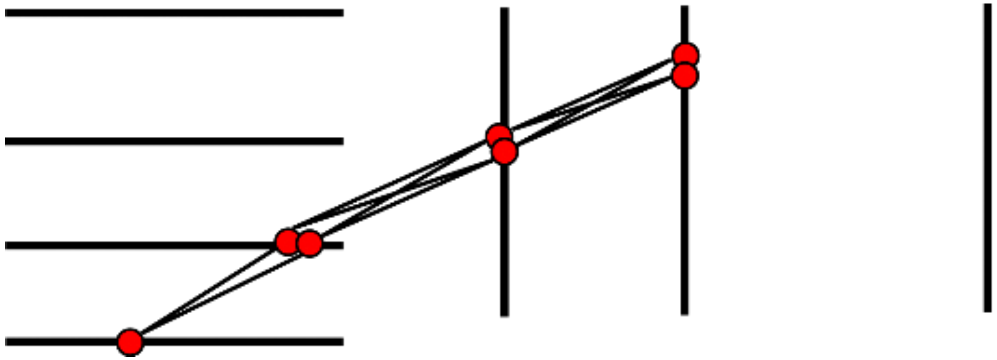
The compatibility between two cells is checked only if they share one hit

- in the R-Z plane a requirement is the alignment of the two cells
- in the cross plane the compatibility is checked with the beamspot region




(AB and BC share hit B)



After using the CA for producing N-tuplets, *fishbone* seeds can be produced to account for module/layer overlaps. To filter out duplicates doublets only the highest grade n-tuplet is fitted.

# Track Fitting

Pixel track "fit" at the HLT is still using 3 points for quadruplets and errors on parameters are loaded from a look-up table in [eta][pT]

The Patatrack Pixel reconstruction includes two Multiple Scattering- aware fits:

**Riemann Fit**

MS included in the covariance matrix

**Broken Line Fit**

Fit of the broken line includes MS kinks in the design

Both the Riemann and the Broken Line fits have been implemented using Eigen that is a C++ template library for linear algebra, matrix and vector operations. This allows perfect code portability between CPU and GPU implementation and bitwise-matching of the results

## Fitting Procedure and Results:

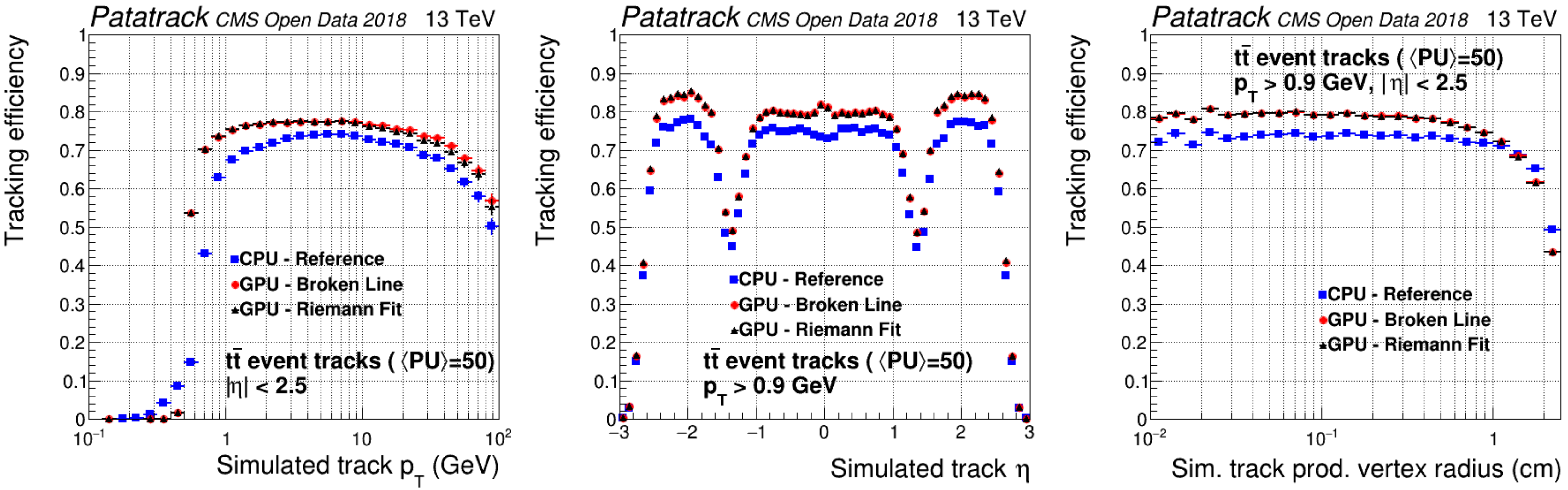
- Fast circle fit: estimate of p for MS, estimate of the radius/center
- Circle fit: d0, pT, phi
- Line fit: $d_z$, cot(theta)

## Tracks Cleaning

- Among Tracks with at least one shared doublet only one with best chi2 retained
- Tracks "rejected" if fails chi2, TIP, ZIP or pt cut
- $p_T > 0.3$ GeV, $|d_0| < 0.5$ cm, $|z_0| < 12$ cm

# Physics Performace – Efficiency

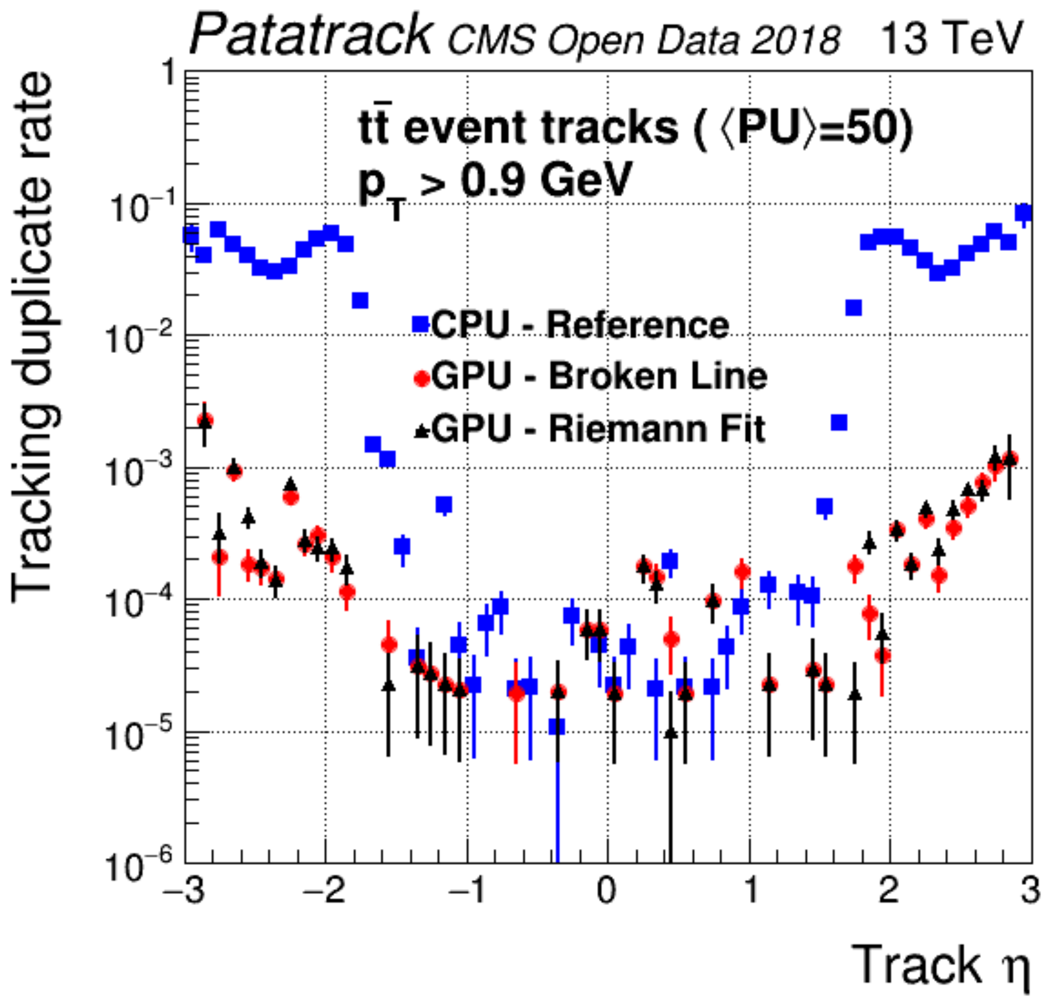
Tracking Efficiency: number of matched reconstructed tracks divided by number of simulated tracks. Computed only with respect to the hard scatter and with $|d_0| < 3.5\ \mathrm{cm}$ additionally to the cuts quoted in the plots. Function of simulated track $\eta$, $p_T$ , and production vertex radius

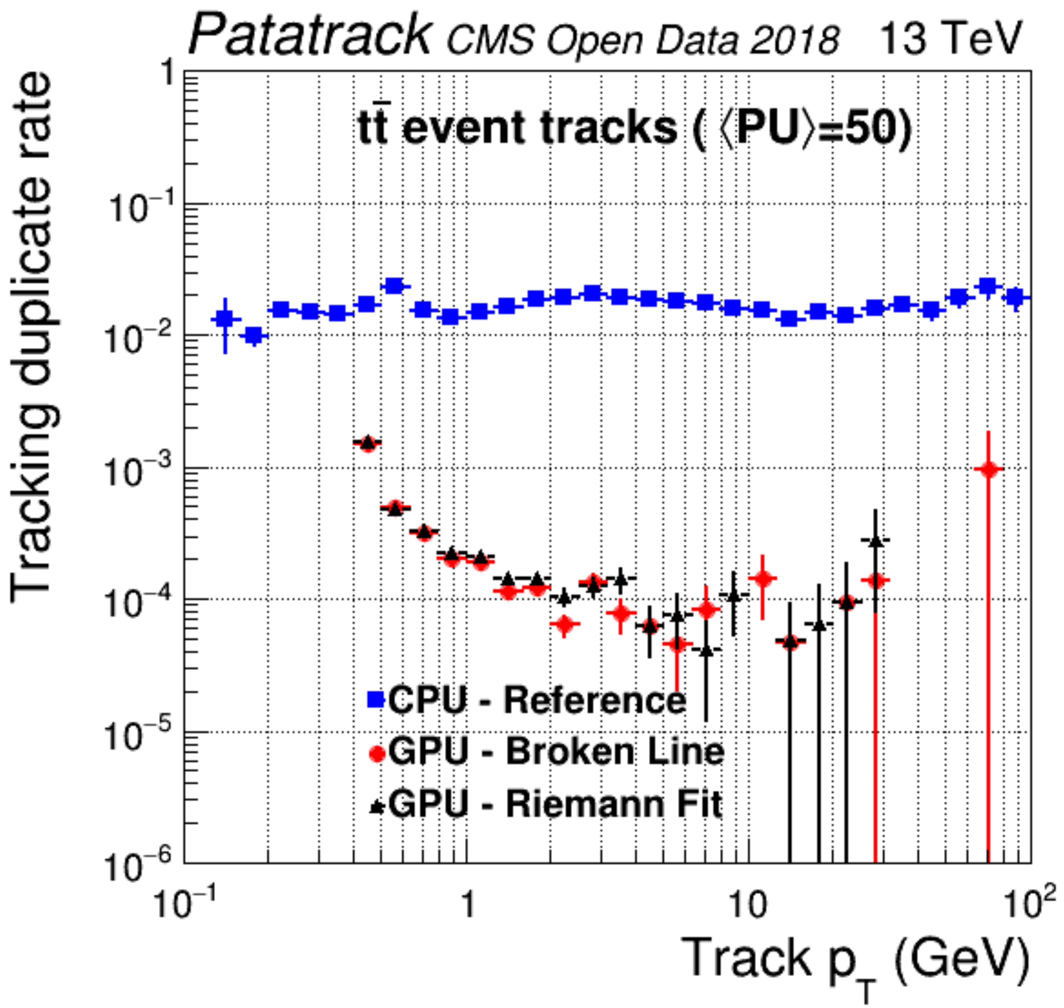



20000 simulated $t\bar{t}$ events at $\sqrt{s} = 13$TeV with $< PU > = 50$ , 25ns, simulated events. Matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track.

# Physics Performance – Duplicates

Duplicate: a reconstructed track matching to a simulated track that itself is matched to >= 2 tracks. Function of reconstructed tracks η, $p_T$

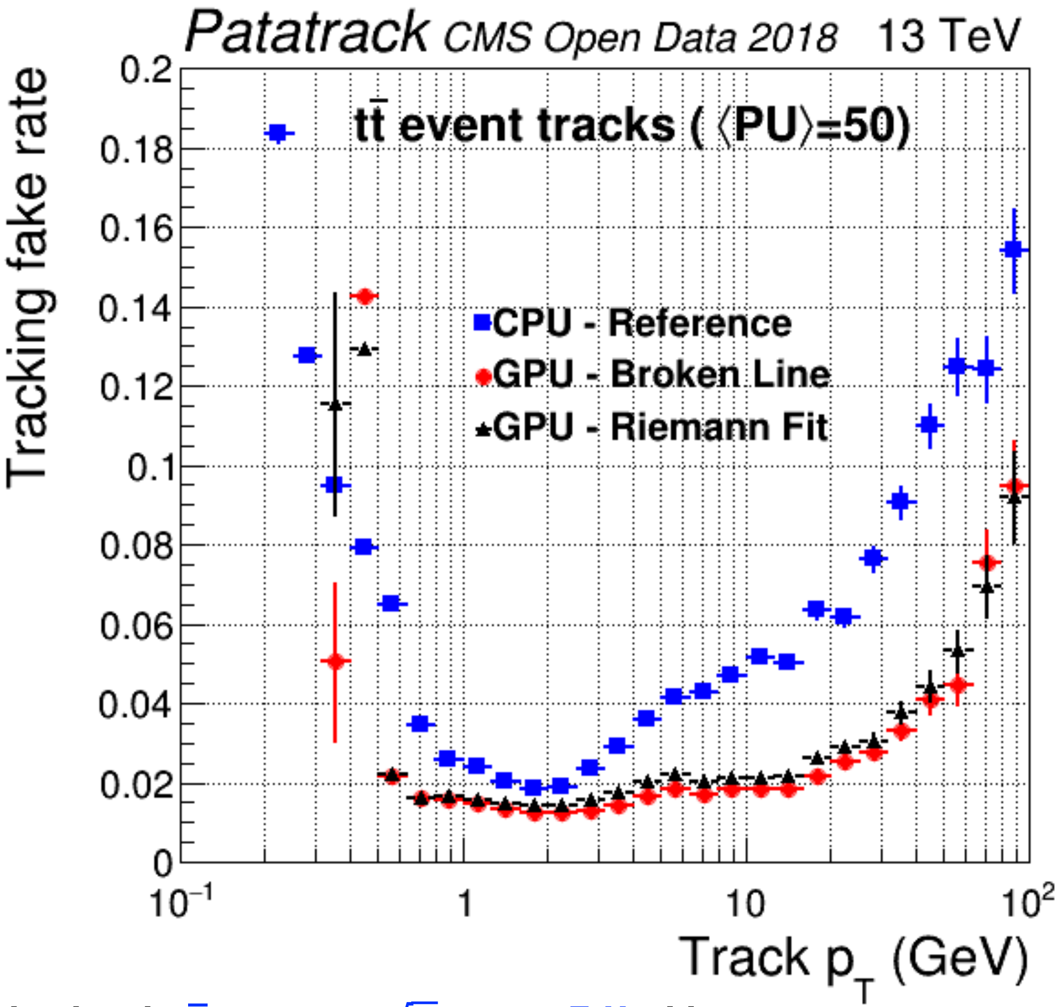


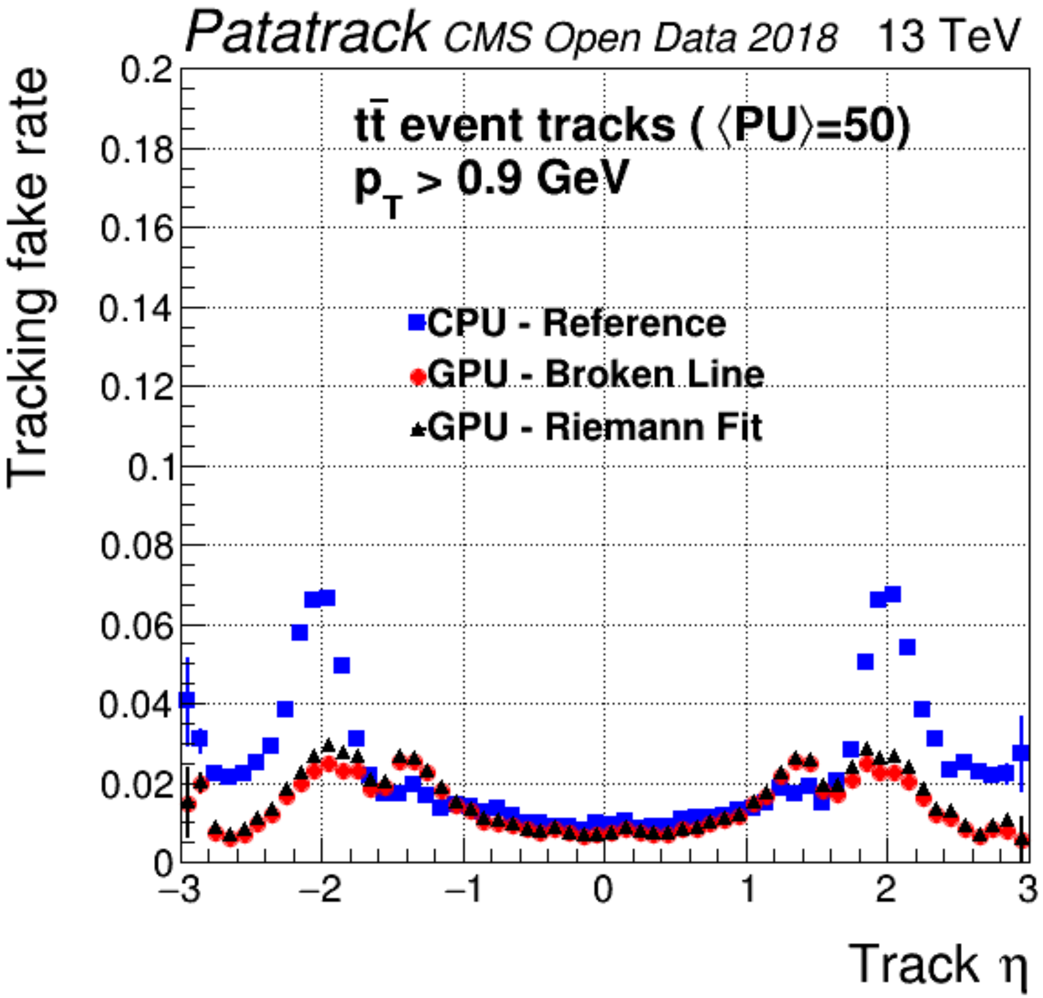




20000 simulated $t\bar{t}$ events at $\sqrt{s} = 13$TeV with $< PU > = 50$ , 25ns, simulated events. Matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track.

# Physics Performance – Fake Rate

**Fake Rate:** number of non-matched reconstructed tracks divided by number of reconstructed tracks. Function of reconstructed tracks $\eta$, $p_T$

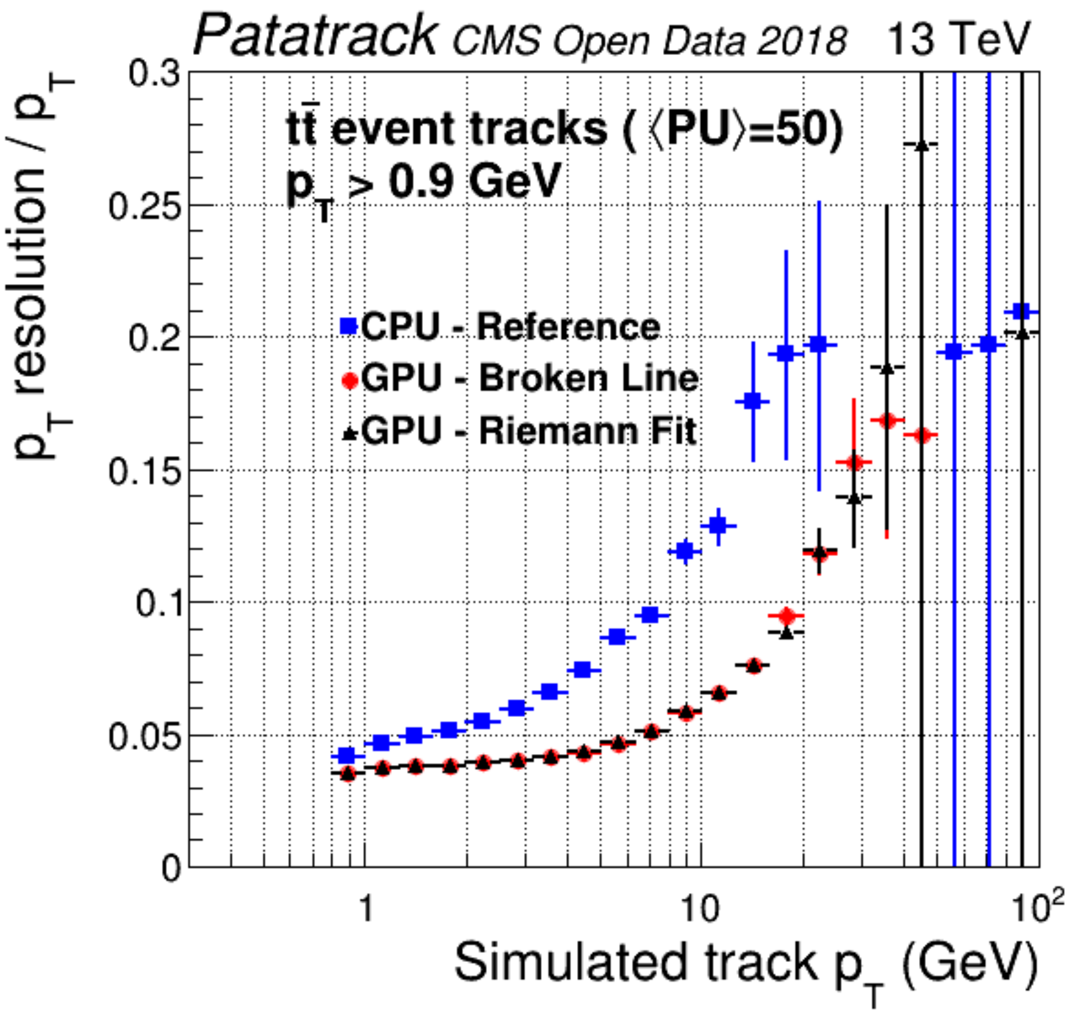


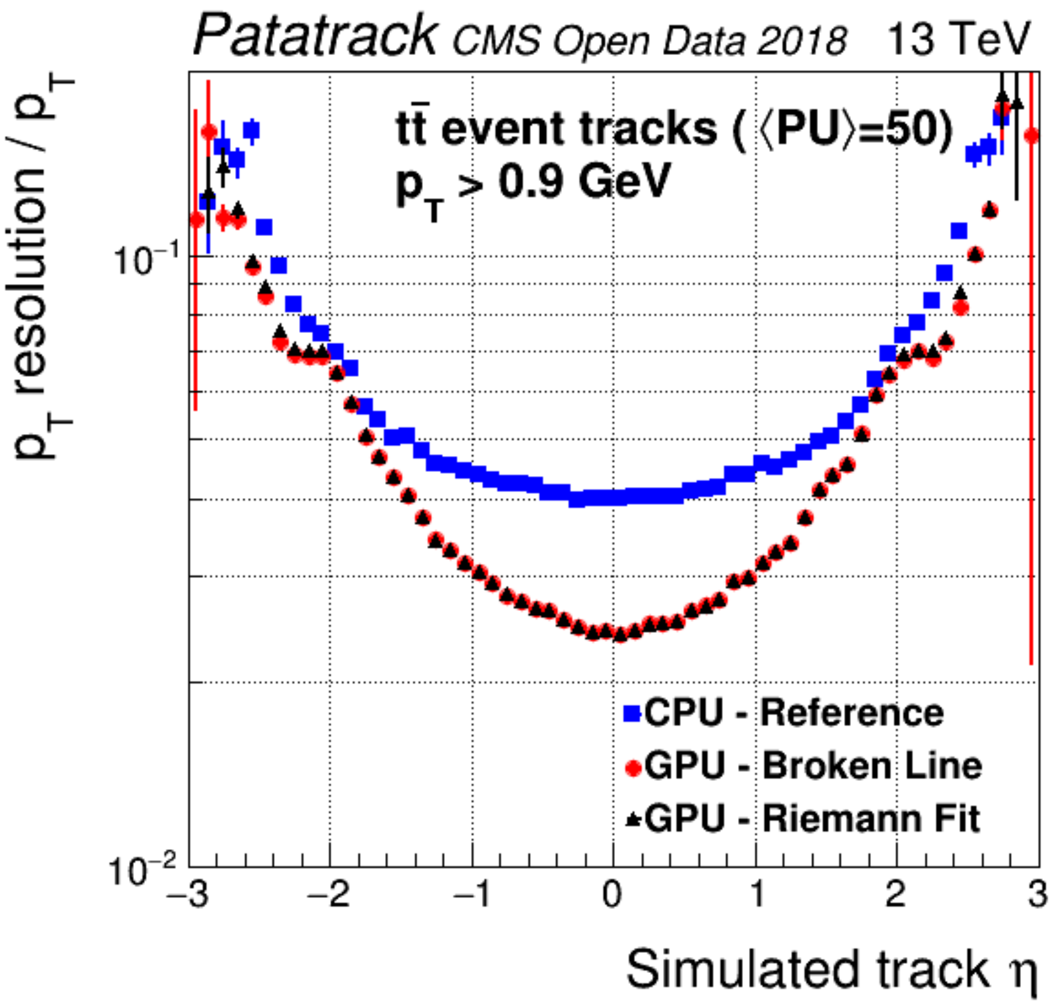




20000 simulated $t\bar{t}$ events at $\sqrt{s} = 13$TeV with $< PU > = 50$ , 25ns, simulated events. Matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track.

# Physics Performance – Resolutions

Track resolution of the $p_T$ . Function of simulated tracks $\eta$, $p_T$

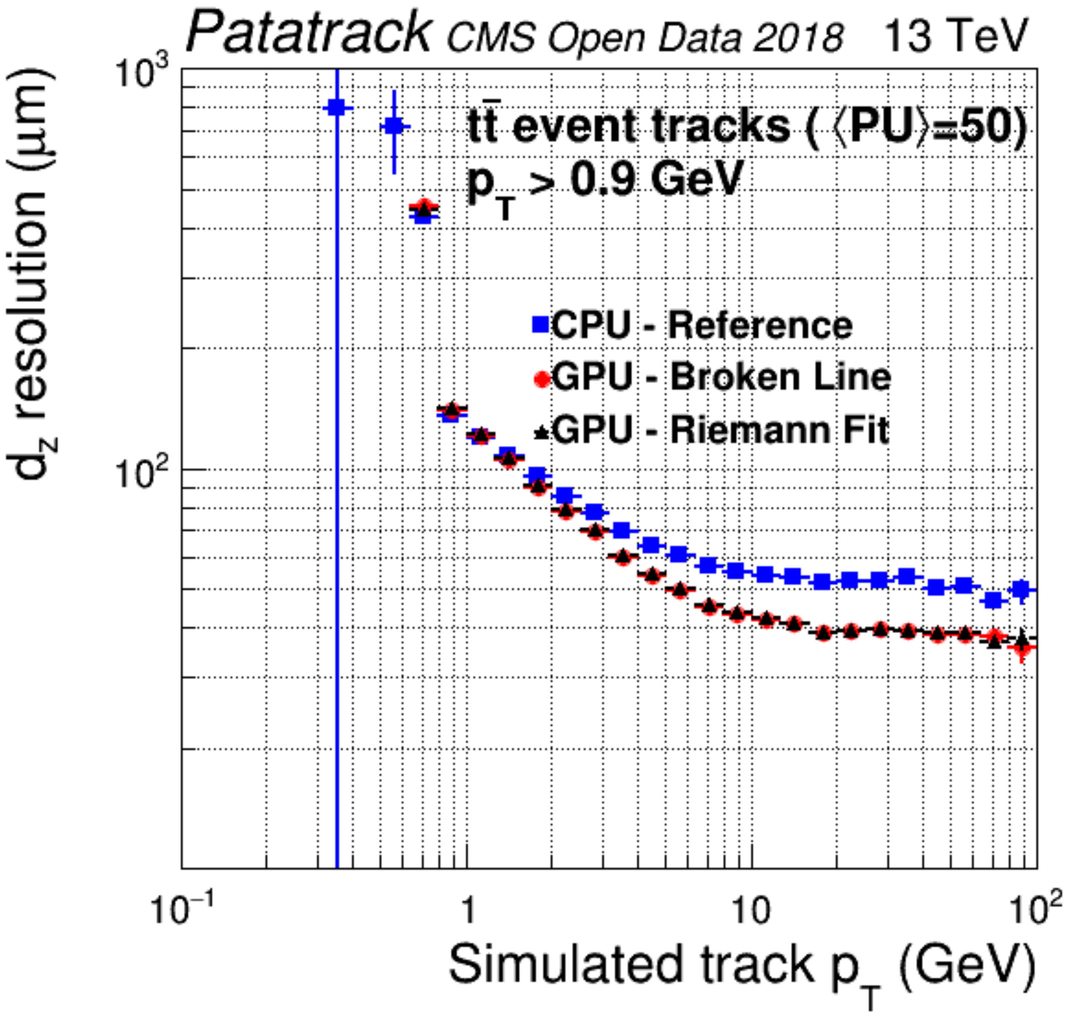


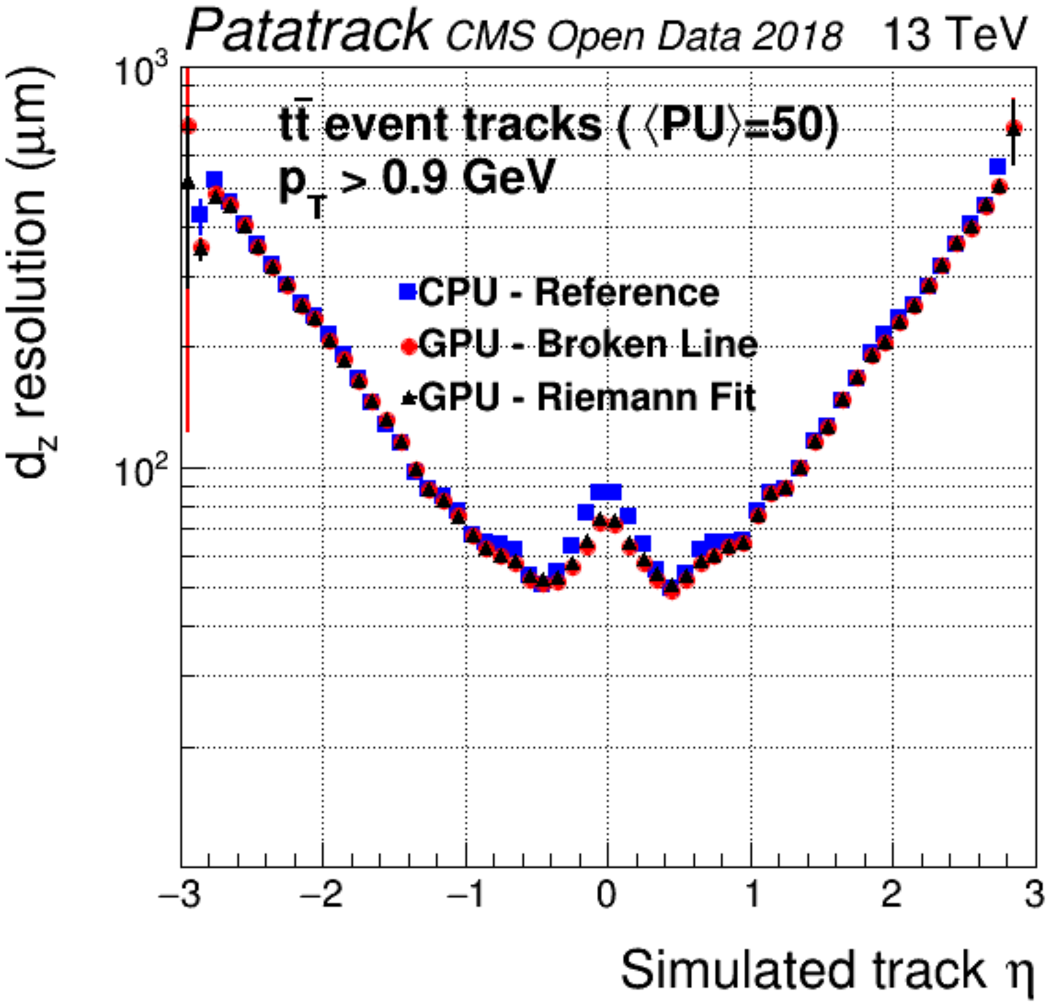




20000 simulated $t\bar{t}$ events at $\sqrt{s} = 13$TeV with $< PU > = 50$ , 25ns, simulated events. Matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track.

# Physics Performance – Resolutions

Track resolution of the longitudinal impact parameter . Function of simulated tracks η, $p_T$

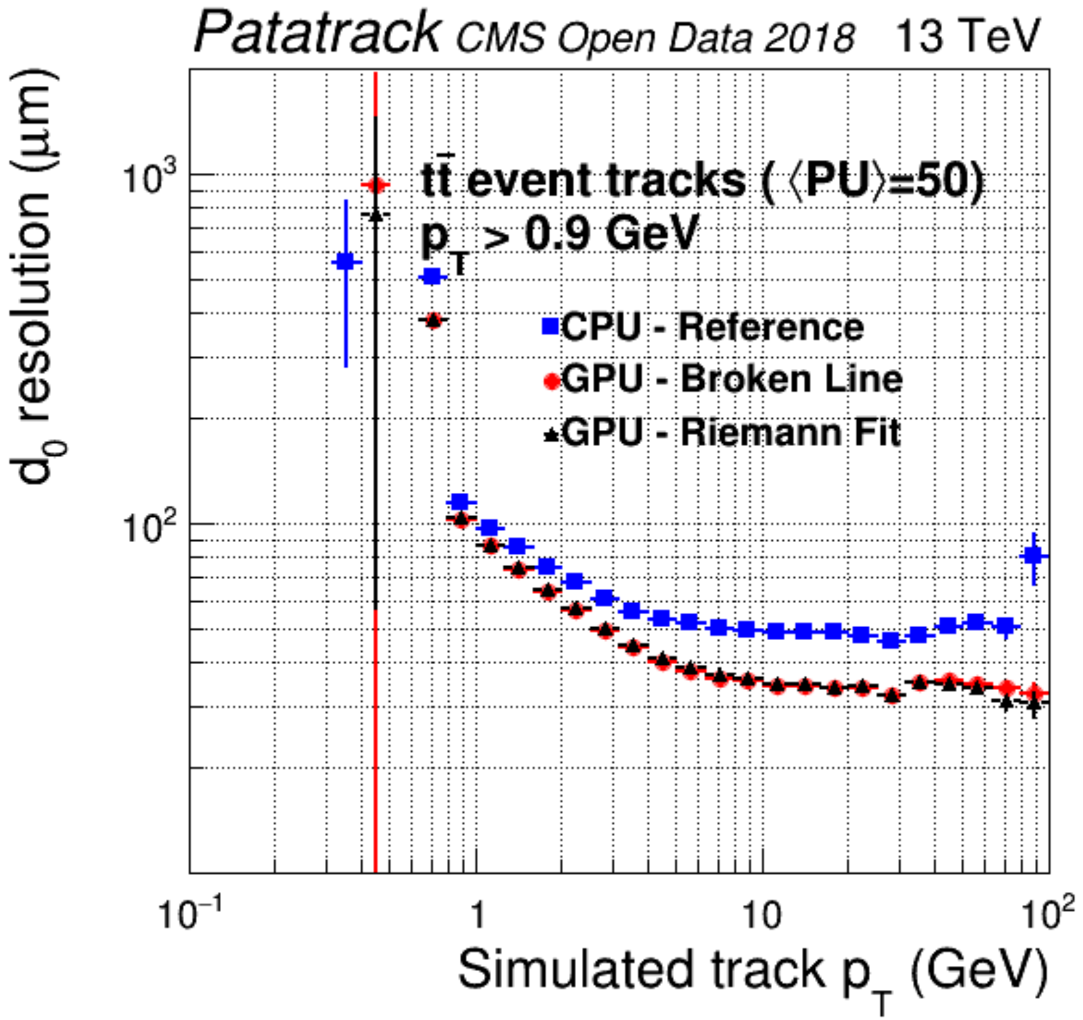


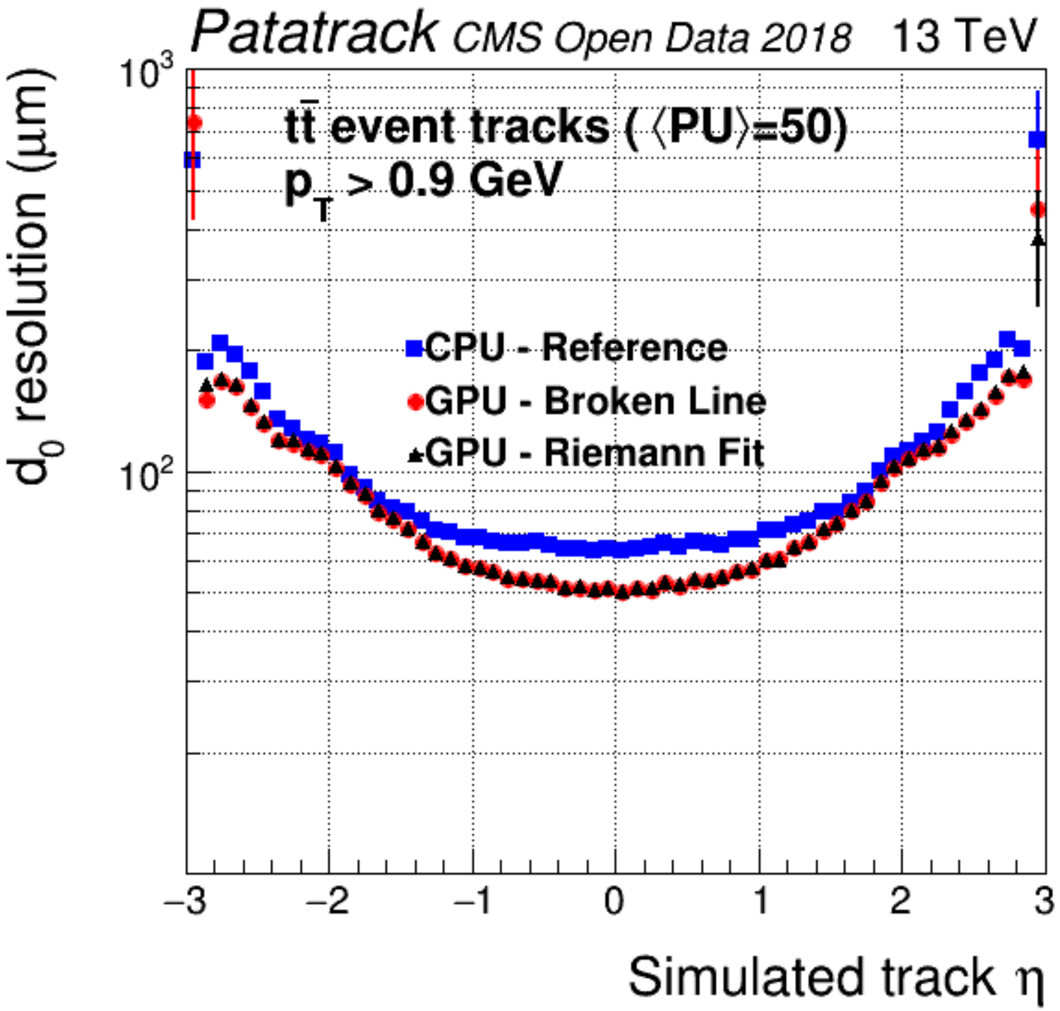




20000 simulated $t\bar{t}$ events at $\sqrt{s} = 13$TeV with $< PU > = 50$ , 25ns, simulated events. Matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track.

# Physics Performance – Resolutions

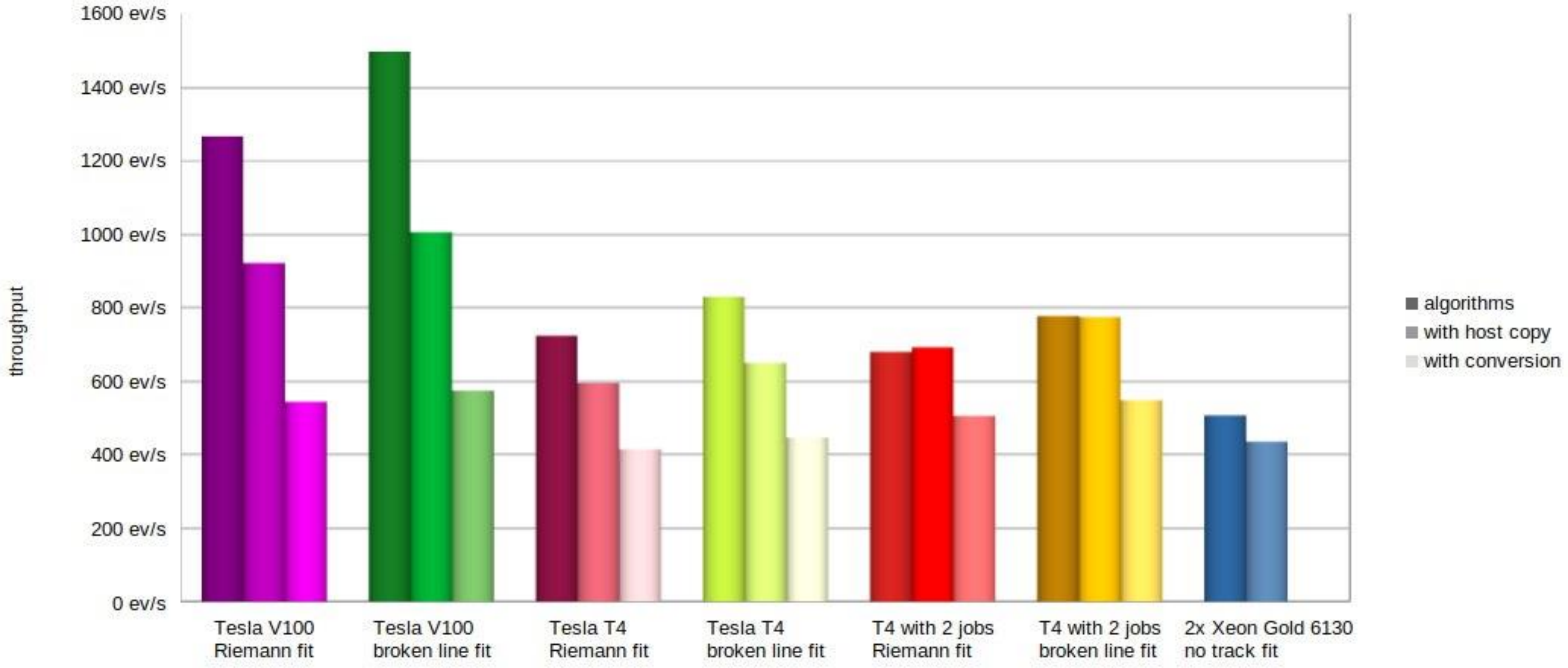
Track resolution of the transverse impact parameter . Function of simulated tracks η, $p_T$




20000 simulated $t\bar{t}$ events at $\sqrt{s} = 13$TeV with $< PU > = 50$ , 25ns, simulated events. Matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track.

# Computational Performance




Pixel reconstruction consumers can either work directly on the GPU or ask for a copy of the tracks and vertices on the host

# Code portability tests (Alpaka + Cupla)

Alpaka is a library to write accelerator-agnostic code

- framework to write “kernels” and run them with different backends
  - CUDA, HIP (in progress)
  - TBB
  - OpenMP 2, OpenMP 4, std::threads, boost::fibers serial execution
- header only
- syntax is similar to CUDA
- based on CMake

| Accelerator Back-end | gcc 4.9.4 (Linux) | gcc 5.5 (Linux) | gcc 6.4/7.3 (Linux) | gcc 8.1 (Linux) | clang 4 (Linux) | clang 5 (Linux) | clang 6 (Linux) | clang 7 (Linux) | clang 8 (Linux) |
|---|---|---|---|---|---|---|---|---|---|
| Serial | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| OpenMP 2.0+ blocks | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| OpenMP 2.0+ threads | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| OpenMP 4.0+ (CPU) | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| std::thread | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| Boost.Fiber | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| TBB | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| CUDA (nvcc) | ✅ (CUDA 8.0-10.1) | ✅ (CUDA 9.0-10.1) | ✅ (CUDA 9.2-10.1) | ❌ | ✅ (CUDA 9.1-10.1) | ✅ (CUDA 10.1) | ✅ (CUDA 10.1) | ✅ (CUDA 10.1) | ✅ (CUDA 10.1) |
| CUDA (clang) | - | - | - | - | ✅ (CUDA 8.0) | ✅ (CUDA 8.0) | ✅ (CUDA 8.0-9.0) | ✅ (CUDA 8.0-9.2) | ✅ (CUDA 8.0-10.0) |
| HIP (nvcc) | ✅ (nvcc 8.0) | ❌ | ❌ | ❌ | ❌ | ❌ | ❌ | ❌ | ❌ |

Cupla is a thin wrapper around Alpaka to make porting from CUDA easy, can be used as a header-only library, or prebuilt .

# Code portability tests (Alpaka + Cupla) – CUDA Code

```
#include <cuda_runtime.h>


__device__
void positive(float x, int &counter) {
  if (x > 0.) {
    atomicAdd(&counter, 1);
  }
}




__global__
void kernel(float *data, int *counter, unsigned int size) {
  __shared__ int shared;
  if (threadIdx.x == 0) {
    shared = 0;
  }

  __syncthreads();

  auto start  = threadIdx.x + blockIdx.x * blockDim.x;
  auto stride = gridDim.x * blockDim.x;
  for (auto i = start; i < size; i += stride) {
    positive(data[i], shared);
  }




  __syncthreads();

  if (threadIdx.x == 0) {
    atomicAdd(counter, shared);
  }
}
```

```
#include <iostream>
#include <random>

int main() {
  unsigned int size = 1024;

  int counter = 0;
  std::vector<float> data(size, 0.);
  std::default_random_engine generator;
  std::uniform_real_distribution<double> distribution(-1.0, 1.0);
  for (auto &x : data)
    x = distribution(generator);

  float *data_d;
  cudaMalloc(&data_d, size * sizeof(float));
  cudaMemcpy(data_d, data.data(), size * sizeof(float), cudaMemcpyHostToDevice);

  int *counter_d;
  cudaMalloc(&counter_d, sizeof(int));
  cudaMemset(counter_d, 0, sizeof(int));

  kernel<<<32, 32>>>(data_d, counter_d, size);

  cudaMemcpy(&counter, counter_d, sizeof(int), cudaMemcpyDeviceToHost);

  std::cout << counter << std::endl;
}
```

# Code portability tests (Alpaka + Cupla) – Cupla Port

```
#include <cuda_to_cupla.hpp>

template <typename T_Acc>
ALPAKA_FN_ACC
void positive(T_Acc const& acc, float x, int &counter) {
  if (x > 0.) {
    atomicAdd(&counter, 1);
  }
}

struct kernel {

template <typename T_Acc>
ALPAKA_FN_ACC
void operator()(T_Acc const& acc, float *data, int *counter, unsigned int size) const
  sharedMem(shared, int);
  if (threadIdx.x == 0) {
    shared = 0;
  }

  __syncthreads();

  auto start  = threadIdx.x + blockIdx.x * blockDim.x;
  auto stride = gridDim.x * blockDim.x;
  for (auto i = start; i < size; i += stride) {
    positive(acc, data[i], shared);
  }



  __syncthreads();

  if (threadIdx.x == 0) {
    atomicAdd(counter, shared);
  }
}

};
```

```
#include <iostream>
#include <random>

int main() {
  unsigned int size = 1024;

  int counter = 0;
  std::vector<float> data(size, 0.);
  std::default_random_engine generator;
  std::uniform_real_distribution<double> distribution(-1.0, 1.0);
  for (auto &x : data)
    x = distribution(generator);

  float *data_d;
  cuplaMalloc((void **) &data_d, size * sizeof(float));
  cuplaMemcpy(data_d, data.data(), size * sizeof(float), cuplaMemcpyHostToDevice);

  int *counter_d;
  cuplaMalloc((void **) &counter_d, sizeof(int));
  cuplaMemset(counter_d, 0, sizeof(int));

  CUPLA_KERNEL_OPTI(kernel)(32, 32)(data_d, counter_d, size);

  cuplaMemcpy(&counter, counter_d, sizeof(int), cuplaMemcpyDeviceToHost);

  std::cout << counter << std::endl;
}
```

# Where are we now . . .

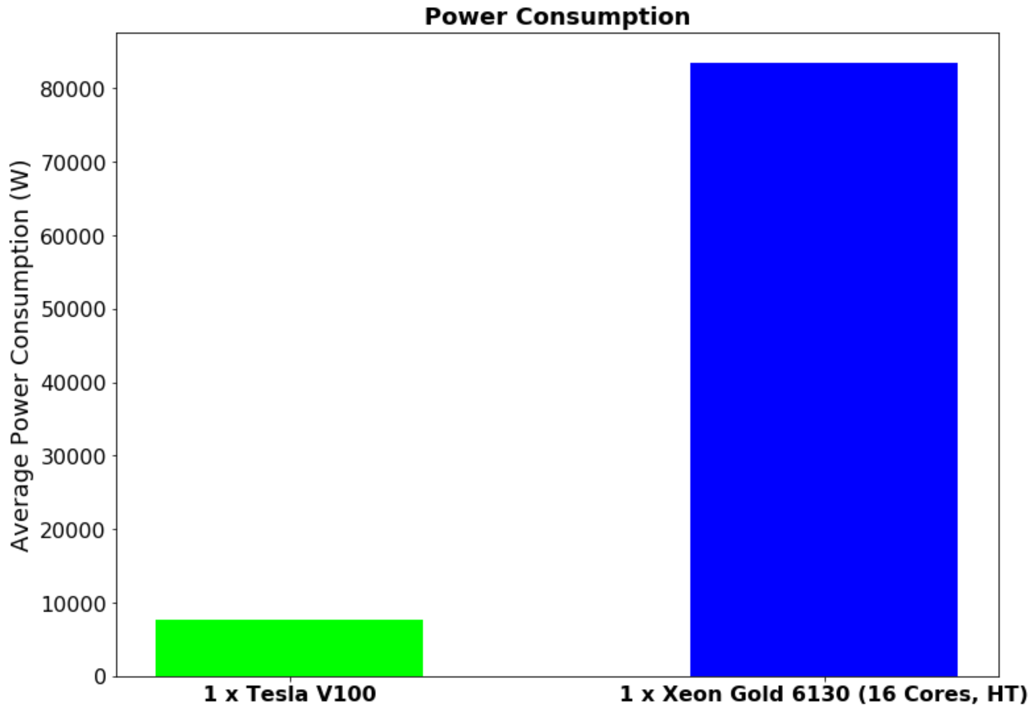
To make a comparison let us consider, for example,2 Xeon Gold, corresponding to 32 cores with HT, and 4 Tesla V100. A hybrid approach would be

1. Faster: approximately 6000 events per second are processed with the hybrid setup against a CPU-only event rate of ∼400 Hz.

2. Cheaper: assuming the current target input rate from L1 of 100 kHz, ∼16 hybrid nodes would be needed compared ∼256 CPU-only nodes. This would mean approximately a factor eight for the hardware costs between the two set-ups.

3. With lower consumption: almost an order of magnitude of power would be saved, still assuming the 100 kHz goal, with the GPU+CPU solution.

4. Better performing : in terms of tracking efficiency, fakes and duplicates rejection and resolutions.



# …and where are we going.

The Patatrack Team has implemented a GPU-based full reconstruction of the Pixel detector from RAW data decoding to Pixel Tracks and Vertices determination. This reconstruction is fully integrated in the CMS Software.

Can achieve better physics performance, faster computational performance at a lower cost with respect to the baseline solution.

Still there's work to do (and only LS2 to do it):

Better understanding portability issues with the same workflow running on GPUs and CPUs on demand

Architecture and hardware studies

Final and transparent integration within the CMS HLT framework

# *Thank You*

*"I am putting myself to the fullest possible use, which is all I think that any conscious entity can ever hope to do"*