Line Segment Tracking in the HL-LHC

Tres Reid, Peter Wittich, Gavin Niendorf (Cornell), Peter Elmer, Bei Wang (Princeton), Philip Chang, Yanxi Gu, Vyacheslav Krutelyov, **Balaji Venkat Sathia Narayanan**, Matevž Tadel, Emmanouil Vourliotis, Avi Yagil (UCSD)

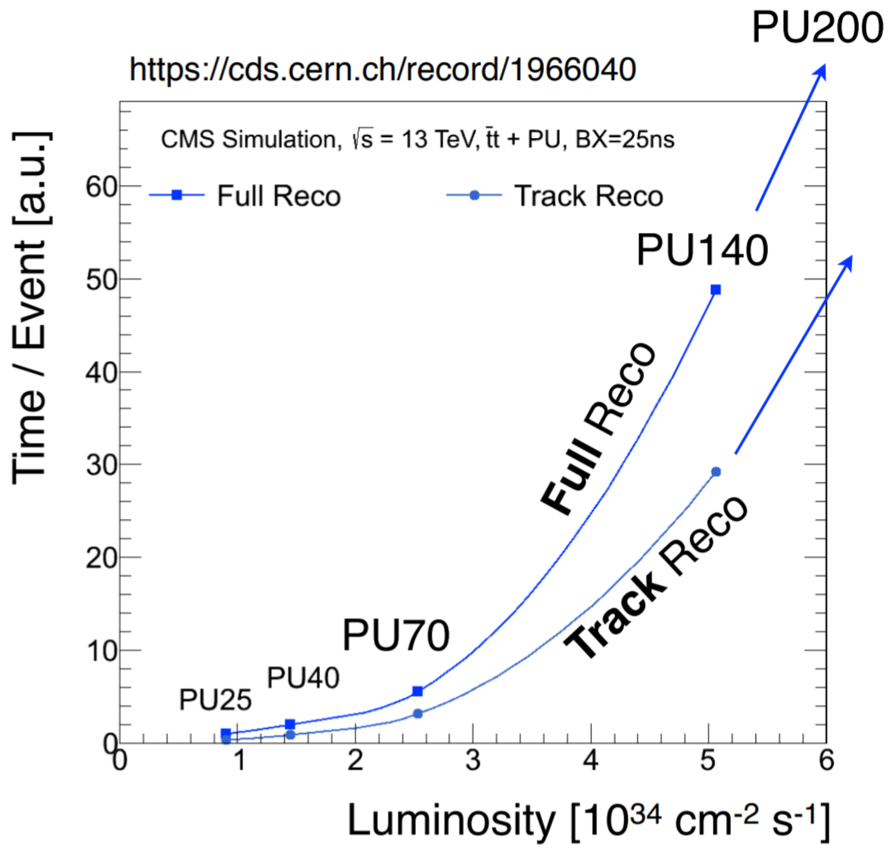
31 May 2022

Outline

- Tracking Challenges at the HL-LHC
- Line Segment Tracking (LST)
- Physics performance
- LST on the GPU
- What the future holds
- Summary

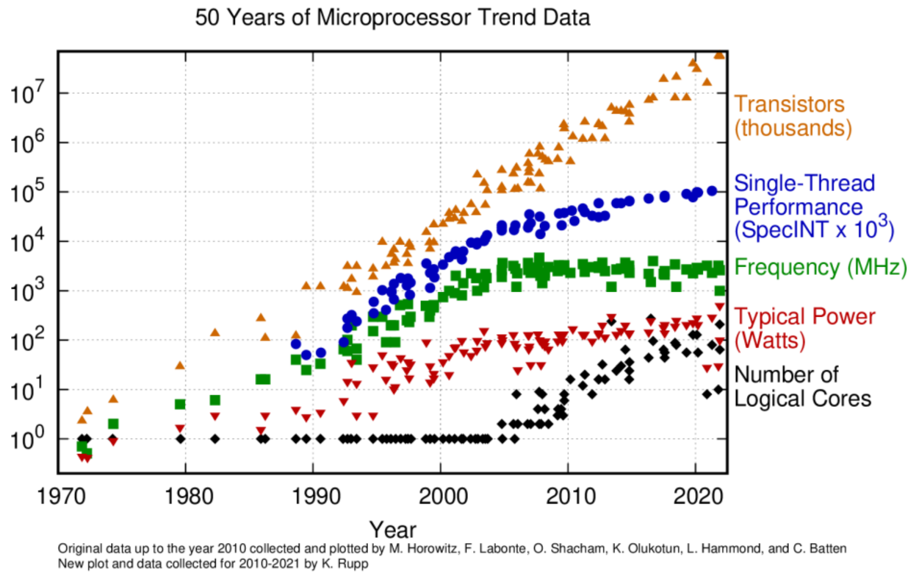
Tracking at HL-LHC

- Track finding is a combinatorics problem
- More collisions ⇒ more hits ⇒ more ways to connect hits ⇒ time and computational expense grows exponentially
- Pile-up 200 at HL-LHC increases number of tracks to be reconstructed
- Need approx 100x more time to reconstruct using current methods and Run-2 detector hardware



New frontiers require new computing paradigms

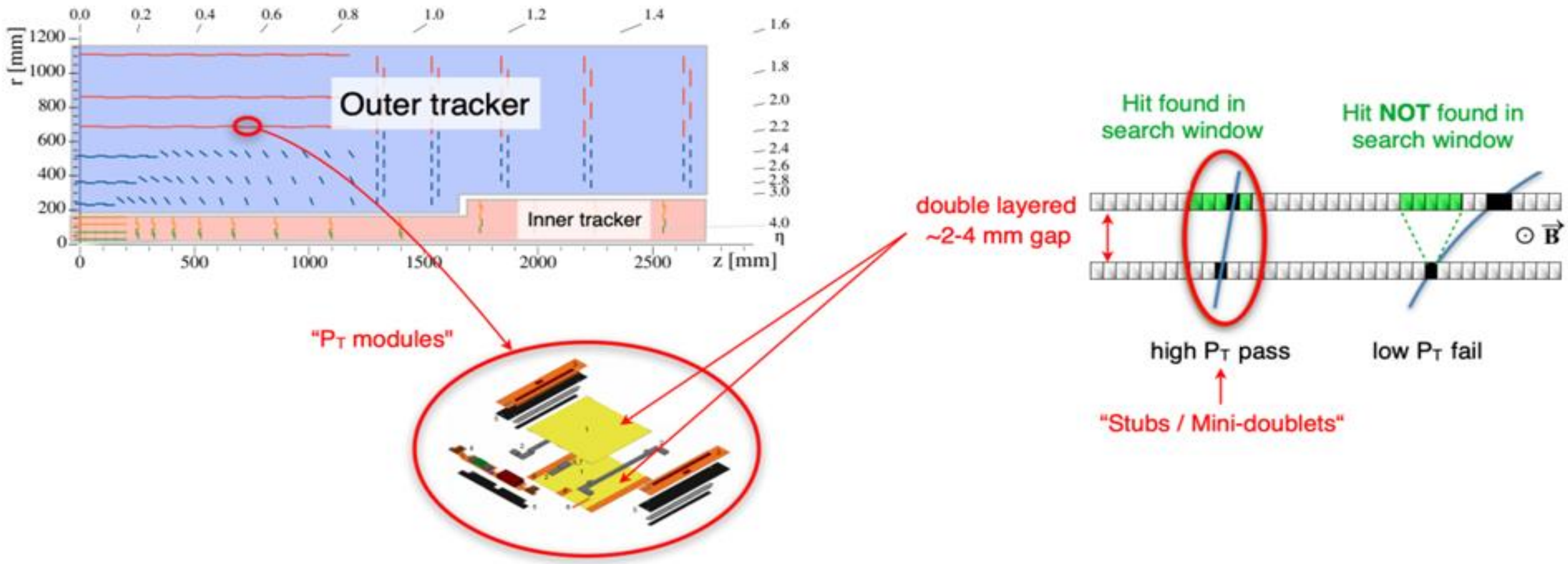
- Moore's Law coming to an end – Single thread CPU performance plateaued
  - Computational demands increasing everyday!
- Future HPC workflows expected to be dominated by GPUs
- Redesign track reconstruction algorithms to be bottom up and take advantage of parallelism provided by GPUs



Source : https://github.com/karlrupp/microprocessor-trend-data

CMS Tracker Geometry in HL-LHC

- Silicon tracker has an inner tracker (silicon pixels) and an outer tracker (silicon strips)
- The outer tracker is made of of two closely sandwiched bi-layer " P_T modules"
  - Allow building of small track stubs based on typical P_T values of tracks



Line Segment Tracking

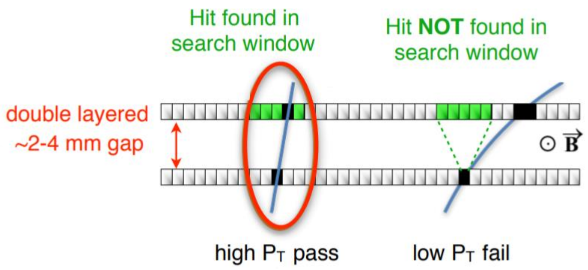
A novel approach to parallelizable tracking

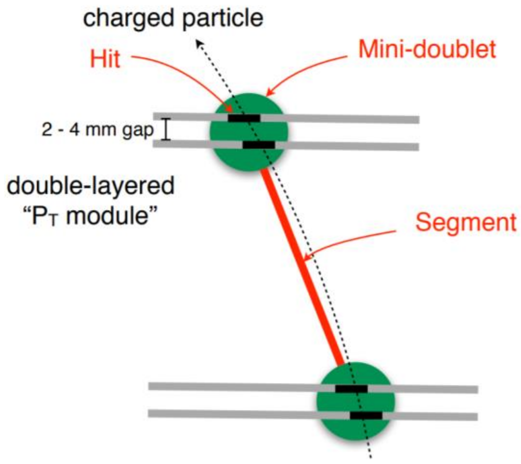
- Bottom-up localized approach to track reconstruction in the outer tracker of the CMS detector at HL-LHC
  - First introduced in CTD2020 https://indico.cern.ch/event/831165/contributions/3717125/
- Charged particle hits "clustered" to reconstruct entire tracks
- Two hits correlated to form a small track, two small tracks join to form a longer track, and so on till tracks are reconstructed
- Can be readily parallelized, since only local information required to reconstruct objects
- Algorithm inspired by the one used in the CDF Detector at the Tevatron (eXtremely Fast Tracker)

The bottom-up construction approach

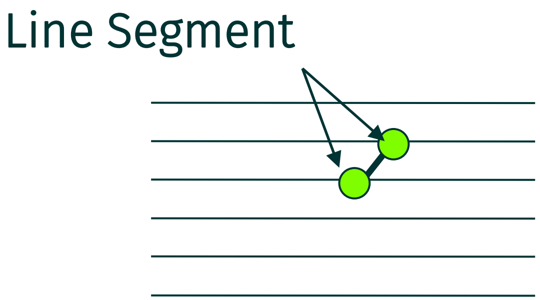
Mini-doublets and segments

- Mini-doublet created from two hits in a bi-layer P_T module (2 hits)
  - Only those consistent with track $P_T > 0.8$ GeV reconstructed
- Two mini-doublets link up to form a line segment (4 hits)
  - Map of valid "module connections" derived from simulations to avoid iterating over the full detector



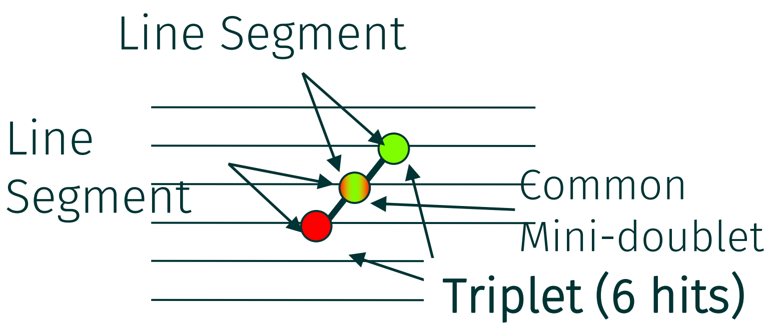


The bottom-up construction approach
Higher Order Objects



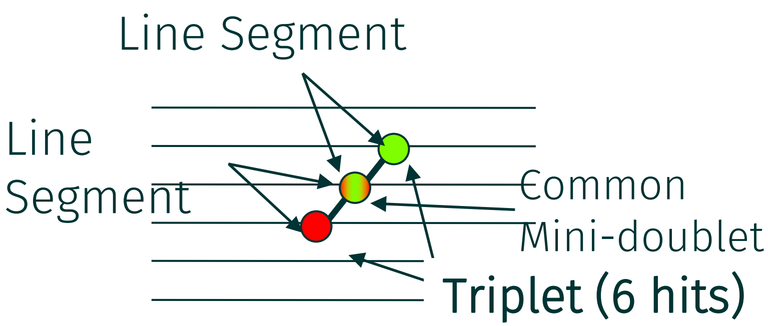
The bottom-up construction approach

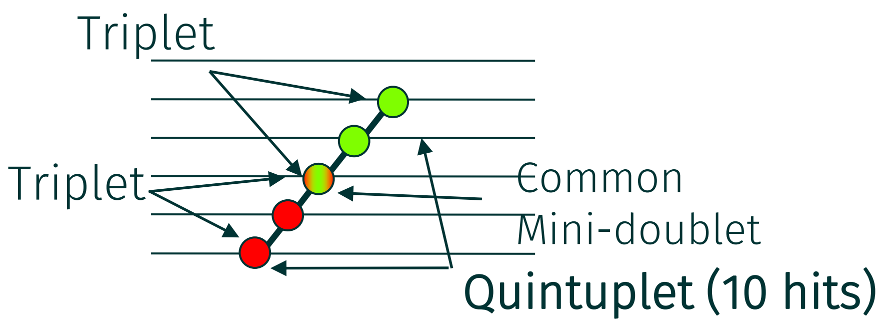
Higher Order Objects



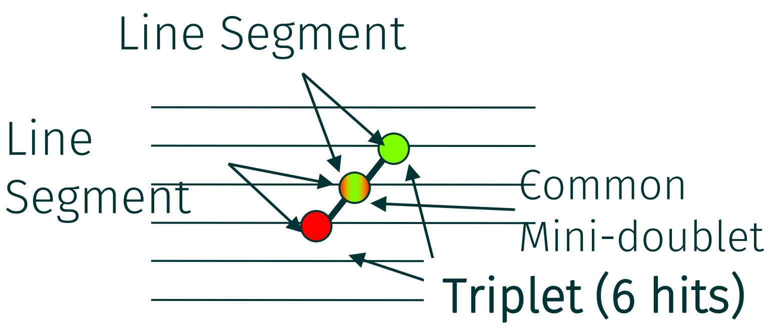
The bottom-up construction approach

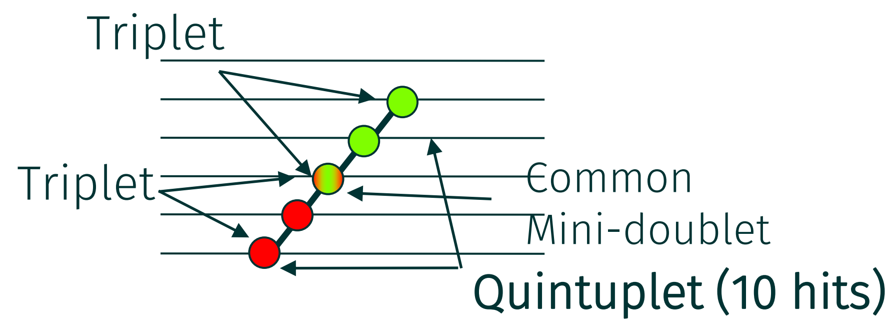
Higher Order Objects

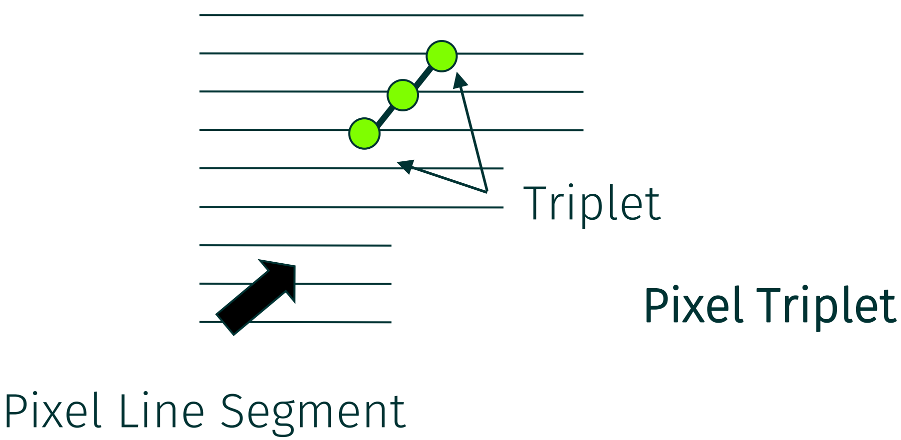




The bottom-up construction approach
Higher Order Objects

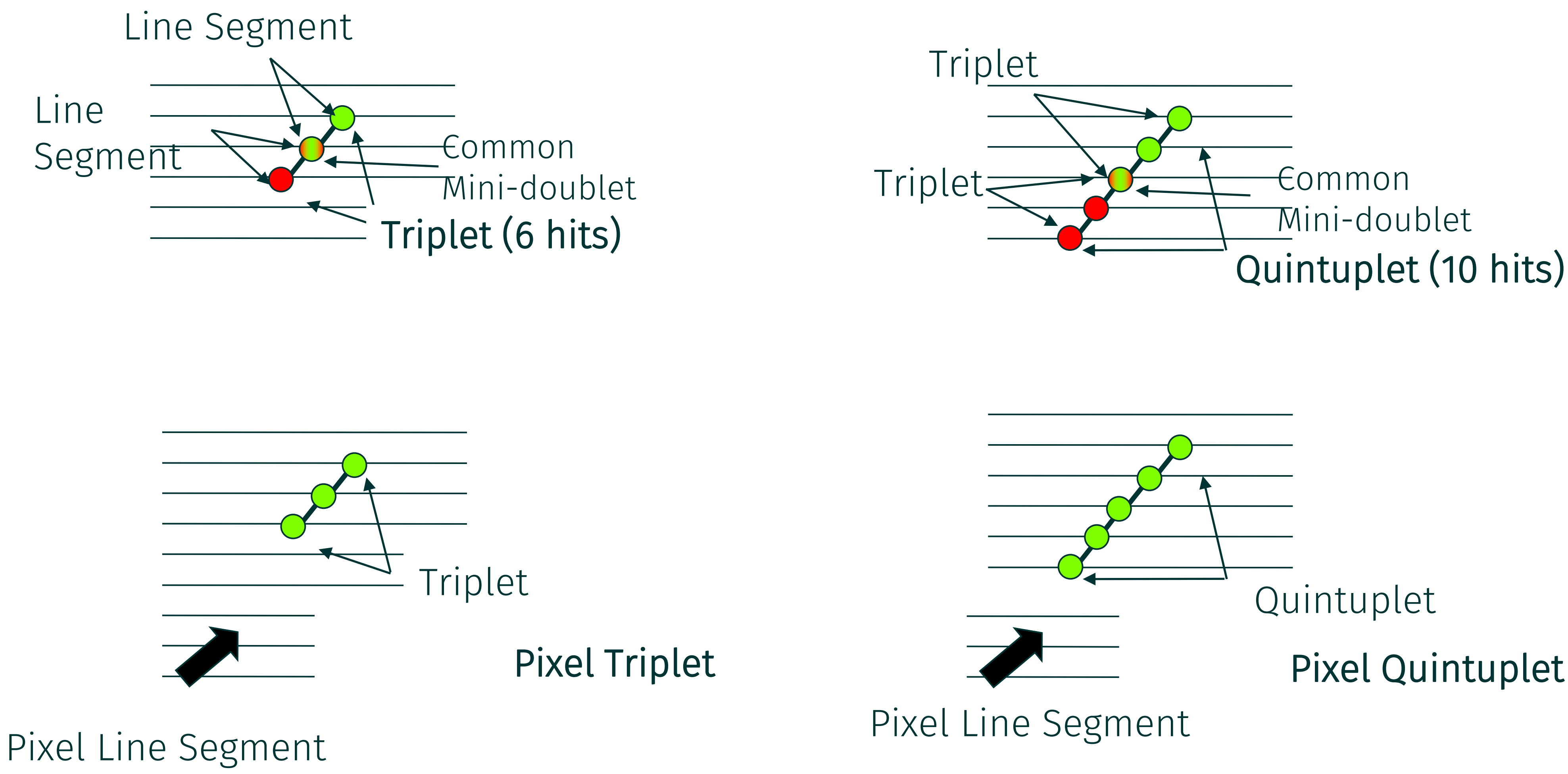






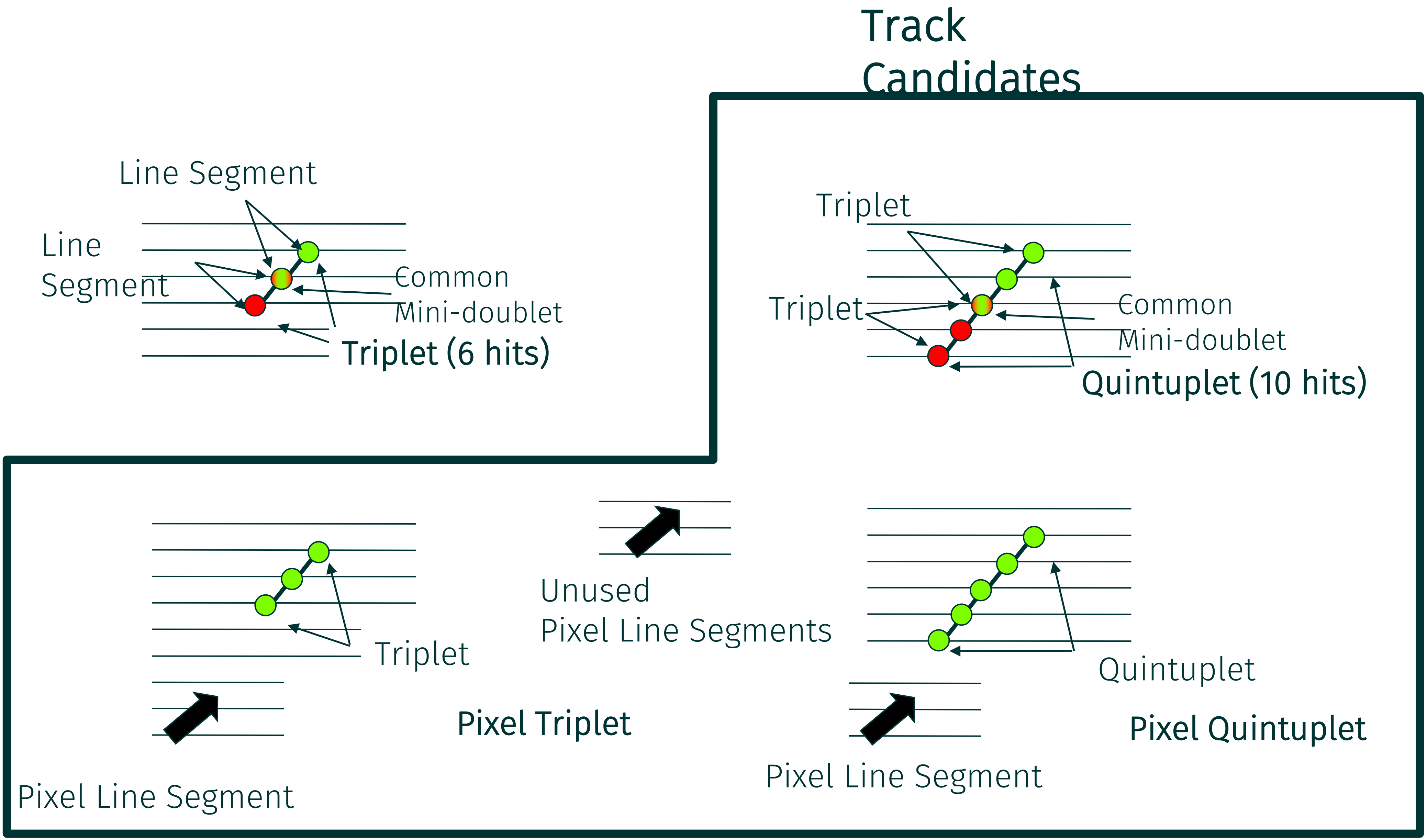
The bottom-up construction approach

Higher Order Objects

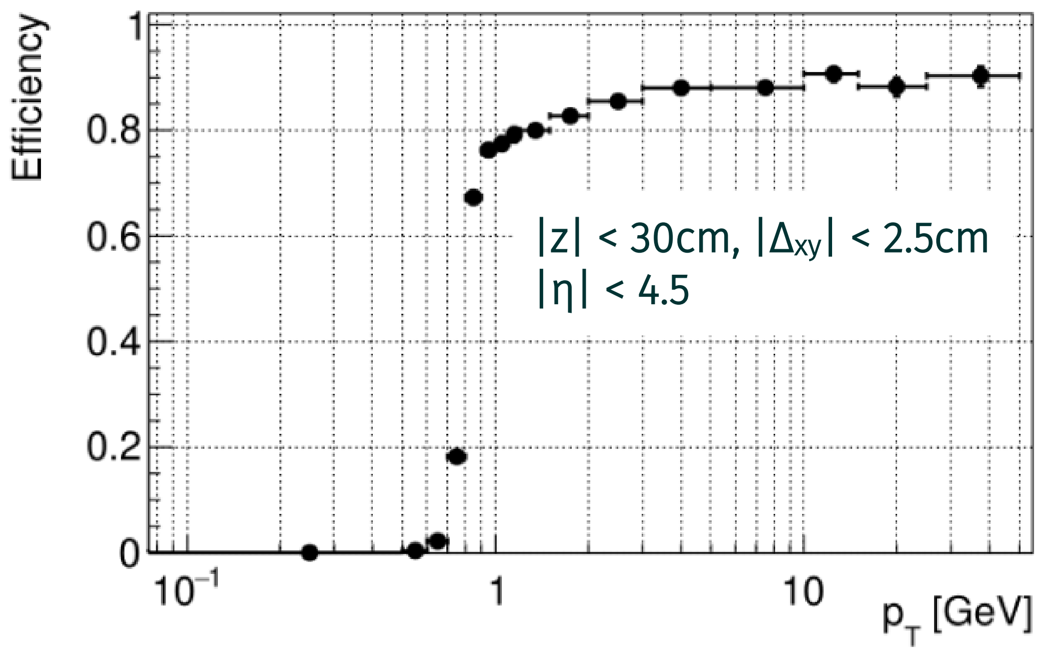


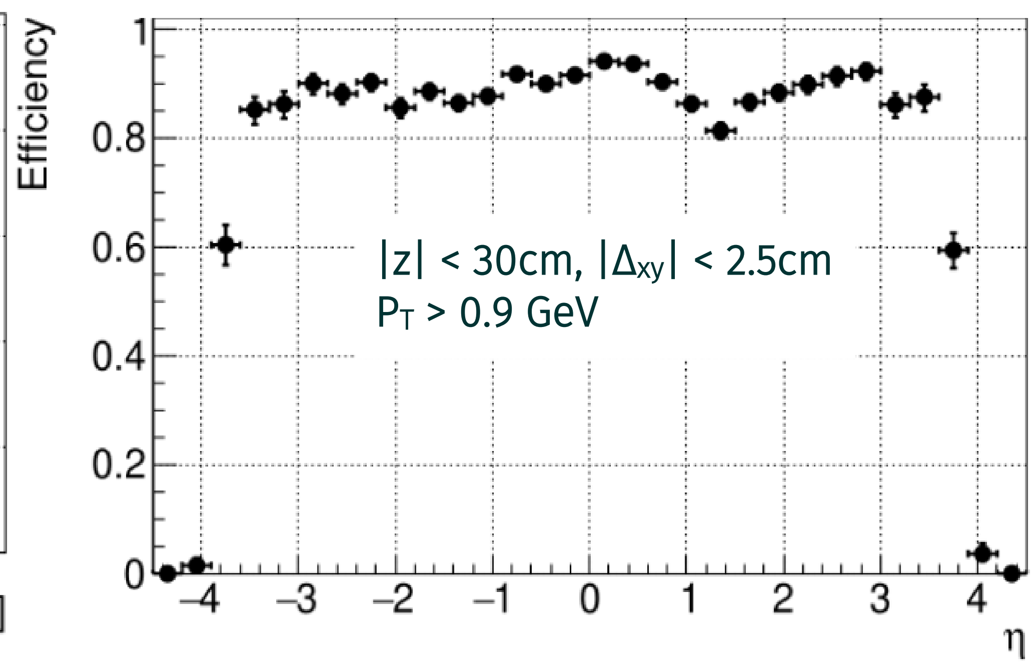
The bottom-up construction approach

Track Candidates



- Measured using 200 events from $t\bar{t}$ + PU200 sample
- Track matching : 75% hits match to a simulated track
- Efficiency saturates at 90%, algorithm competitive with existing CMS Tracking algorithms (TDR : https://cds.cern.ch/record/2759072)
- Turn on at 0.8 GeV since only P_T > 0.8 GeV tracks reconstructed

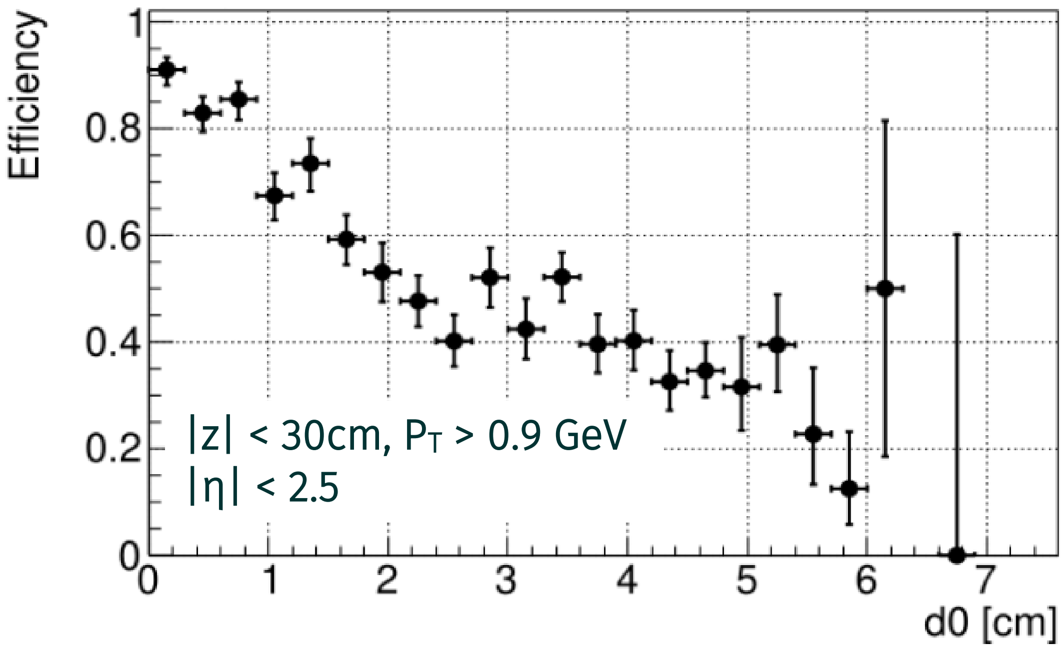




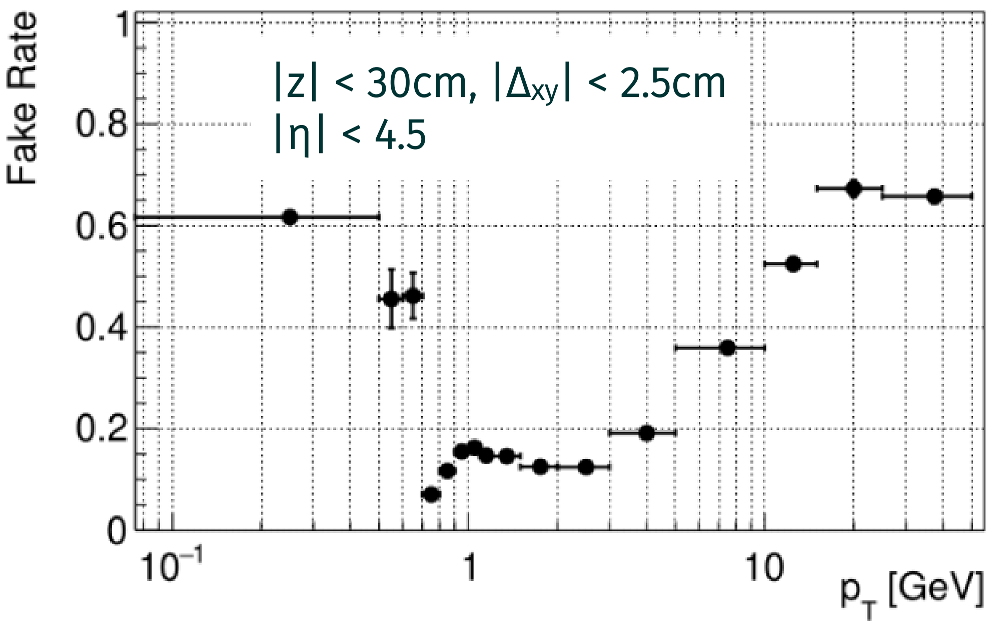
Physics Performance

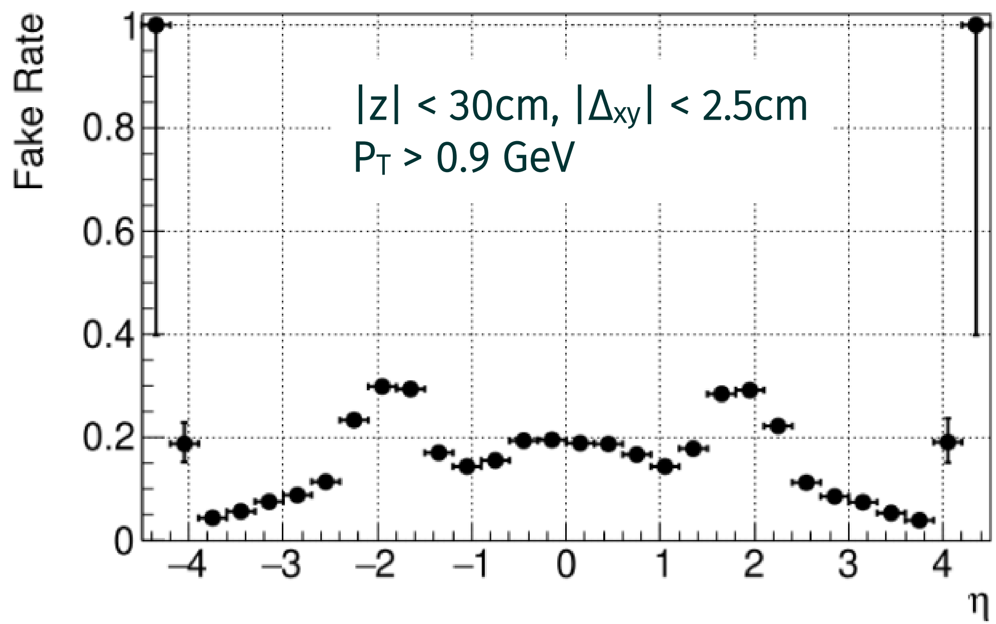
Efficiency distributions – displaced tracks

- Efficiency Measured in a sample of displaced muon tracks in a 5cm cube around the interaction point
- Good reconstruction efficiency achieved, can be improved further



- Fake rates in $t\bar{t}$ + PU200 comparable with the Kalman Filter based CMS Tracking algorithm
  - Can be reduced further with a full fit of hit patterns
- Highest contribution to fakes comes from the Pixel Line Segments, especially in the forward region

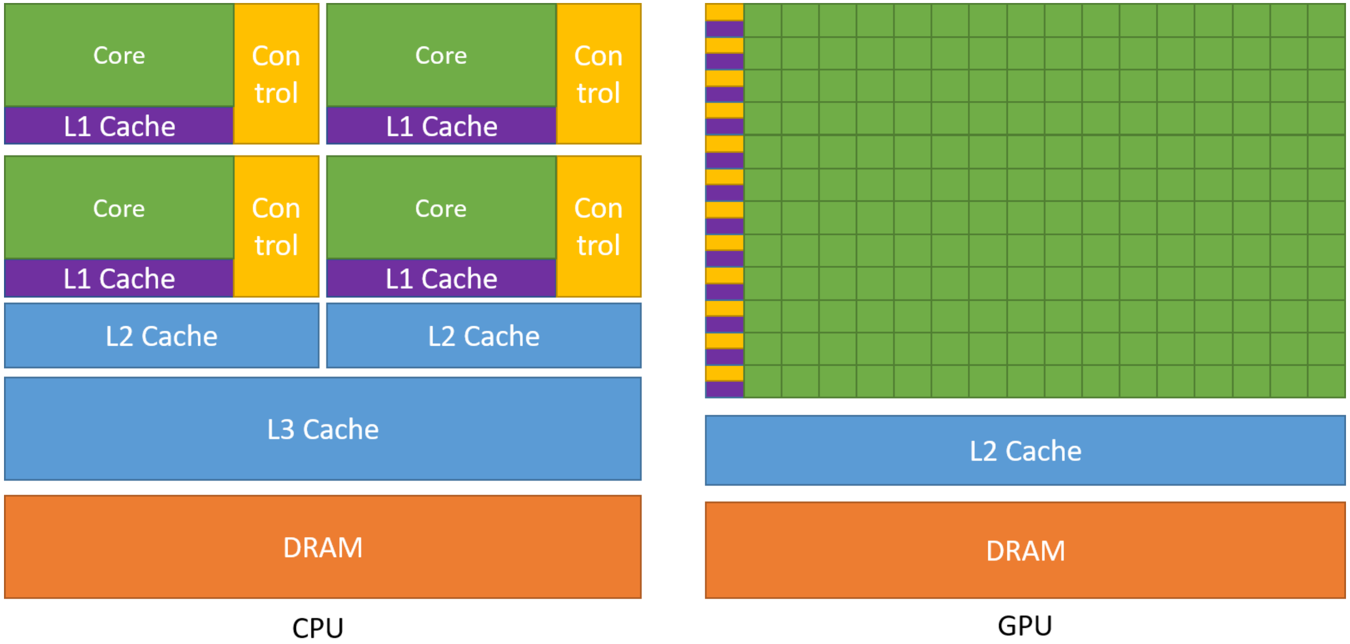




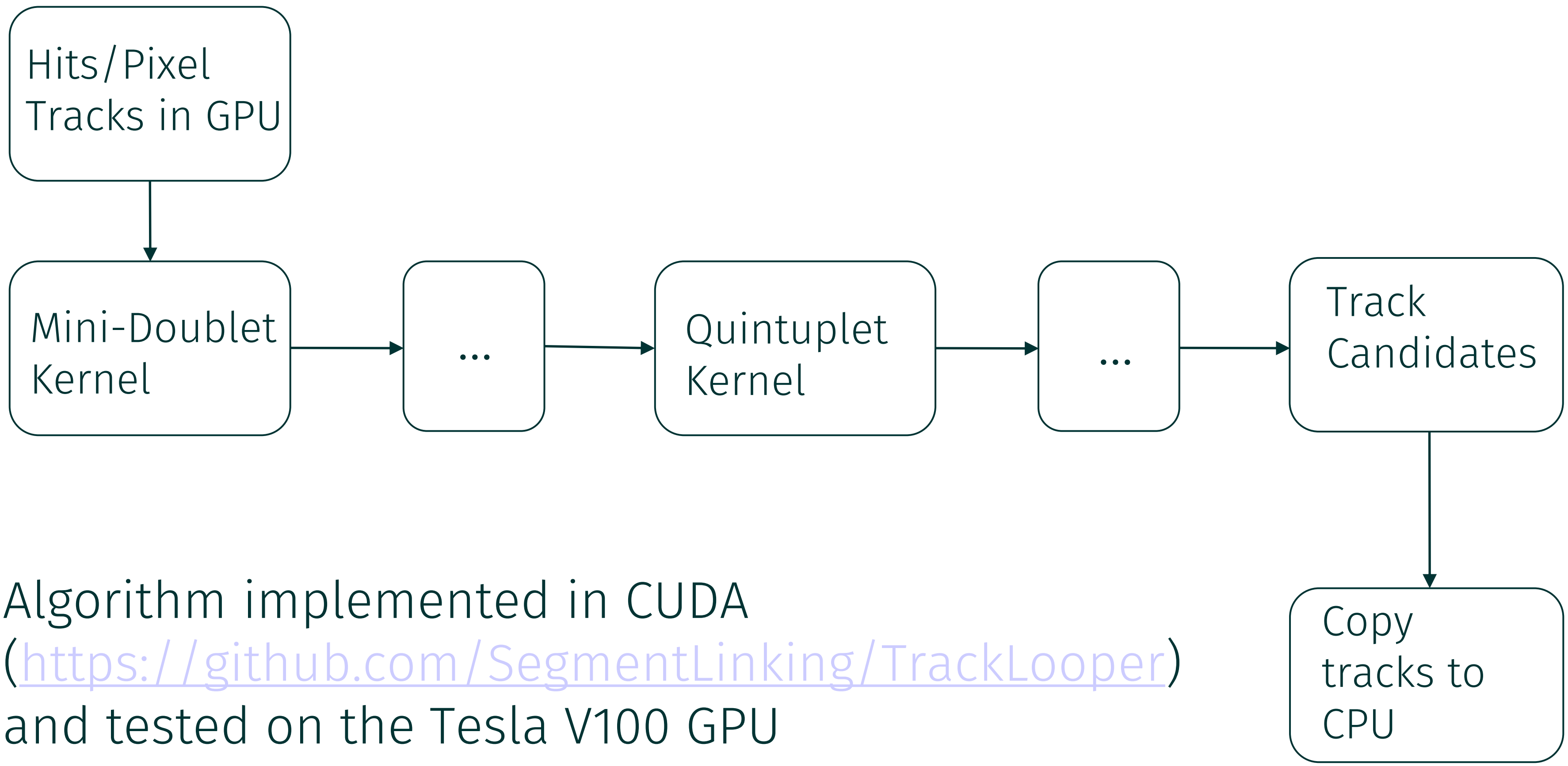
Line Segment Tracking on the GPU

GPU Architecture 101

- GPUs have lots of compute cores (green) compared to CPUs, but compromise on caches and data transport
- Compute cores work on existing data while waiting for new data ⇒ significant speed-ups

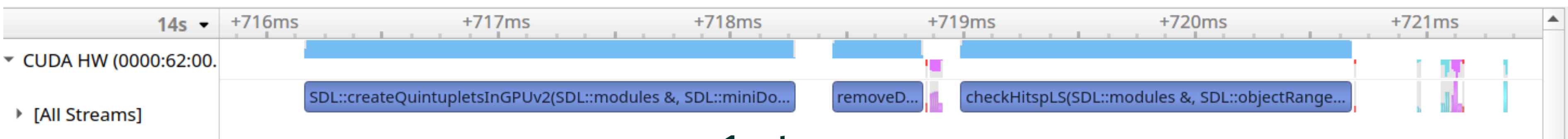


- Data stored in structure of Arrays (SoA) for efficient access
  - Custom caching allocators for fast and efficient memory access in GPU

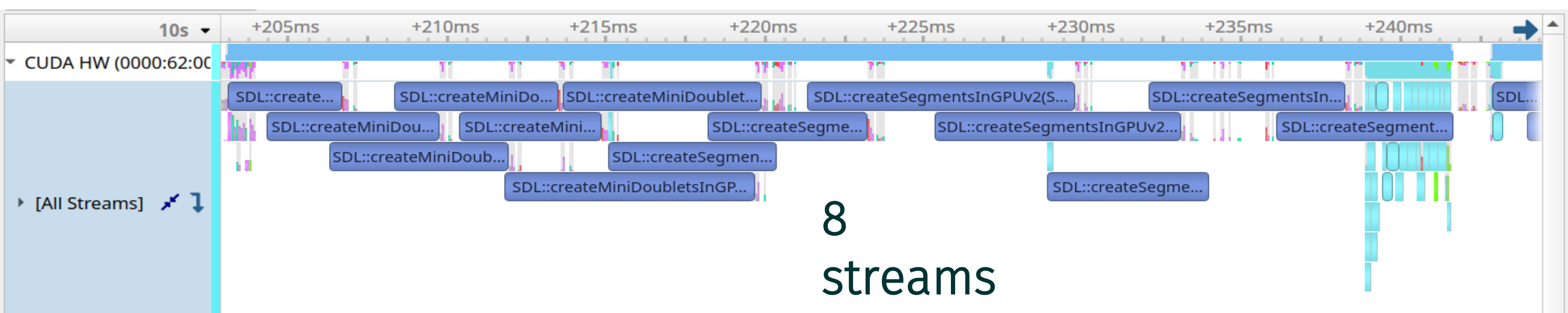


Algorithm implemented in CUDA (https://github.com/SegmentLinking/TrackLooper) and tested on the Tesla V100 GPU

- Multi-streaming : One stream per event
- Kernels too large - Entire kernels cannot run in parallel!
- Kernel pipelining - free cores can run parts of kernels from different streams
- Individual kernels take longer but overall throughput improves by 25%



1 stream



8 streams

Line Segment Tracking on the GPU

Lessons learned – More details on Implementation

- Memory allocation has overheads. Custom caching allocator exponentially allocates memory and avoids reallocation costs. Improves timing by 25%
- Thread divergence issues minimized in new GPUs. Threads that fail any step of physics selections exit immediately. Improves timing by 33%
- Register overhead reduced by 50% when kernel and called functions in same source file
- Objects passing all selections saved first come first serve. Location of new objects computed using atomics to prevent race conditions
- Kernel launches : Block scheduling now smarter. The higher the number of blocks the better the scheduling
  - Thumb rule : 128 or 256 threads per block
- Memory pre-allocation using multiplicity distributions. Reduces memory footprint to 1GB per event, enables multi-streaming
- L1 and L2 caches better and smarter : Shared memory not that important anymore

Line Segment Tracking on the GPU

Timing performance

- Average time per $t\bar{t}$ + PU200 event on a single stream : 34 ms/event
  - Note : Timing measured without final transfer of outputs to host
- Our best average time : 26ms/event running on 8 concurrent streams
  - Takes advantage of using empty cores; 25% faster than single-stream
- Line Segment Tracking on par price wise with the CMS Track pattern Recognition algorithm on 64 CPU cores
  - CMS Track pattern Recognition takes around 30ms (50% of all tracking, scaled to 64 cores)

  (DP Note : https://cds.cern.ch/record/2792313/files/DP2021_013.pdf)
  - Two socket 64 cores Skylake Gold Xeon processor has a similar price to a V100 GPU

What the future holds

- Physics algorithm optimizations
  - Full fit of tracks to further reduce fake rates
  - Better reconstruction of displaced tracks
- Code optimizations
  - Mathematical optimizations :efficient computation of physics parameters
  - Data type optimizations : half precision
  - Timing optimizations : Reducing register usage, improving memory coalescion
- Final target : deploy in the CMS software backend for HLT and offline reconstruction in time for HL-LHC

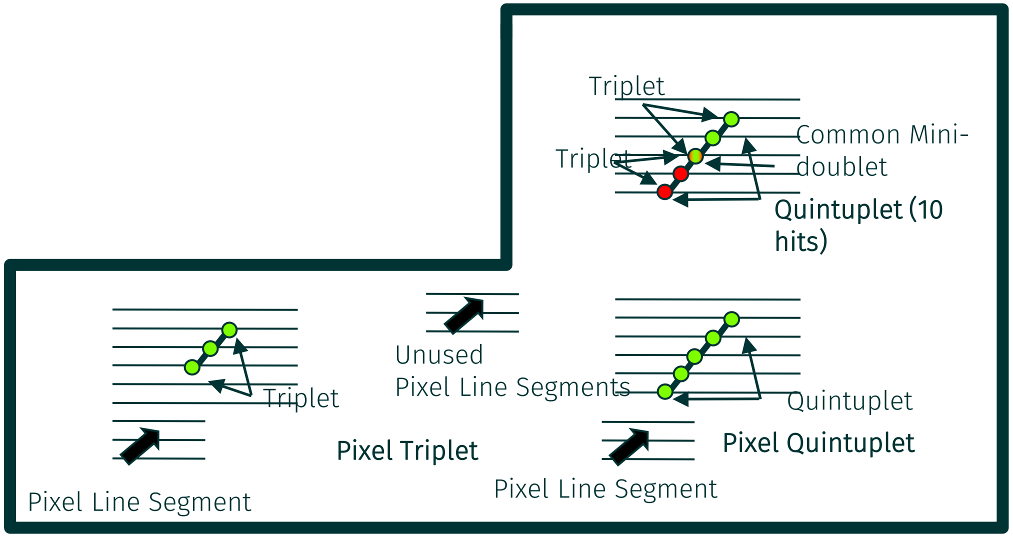
Summary

- Track reconstruction challenges are getting more computationally expensive in the HL-LHC
- Need to look into parallelizable algorithms to take advantage of new computing architectures like GPUs
- Line Segment Tracking : A bottom-up localized algorithm that can reconstruct tracks in parallel
- GPU implementation produces good efficiency with low fake rates, and is competitive with target CPU reconstruction times (best time : 26ms/event)

Backup

Track Candidates and Track Candidate Extensions

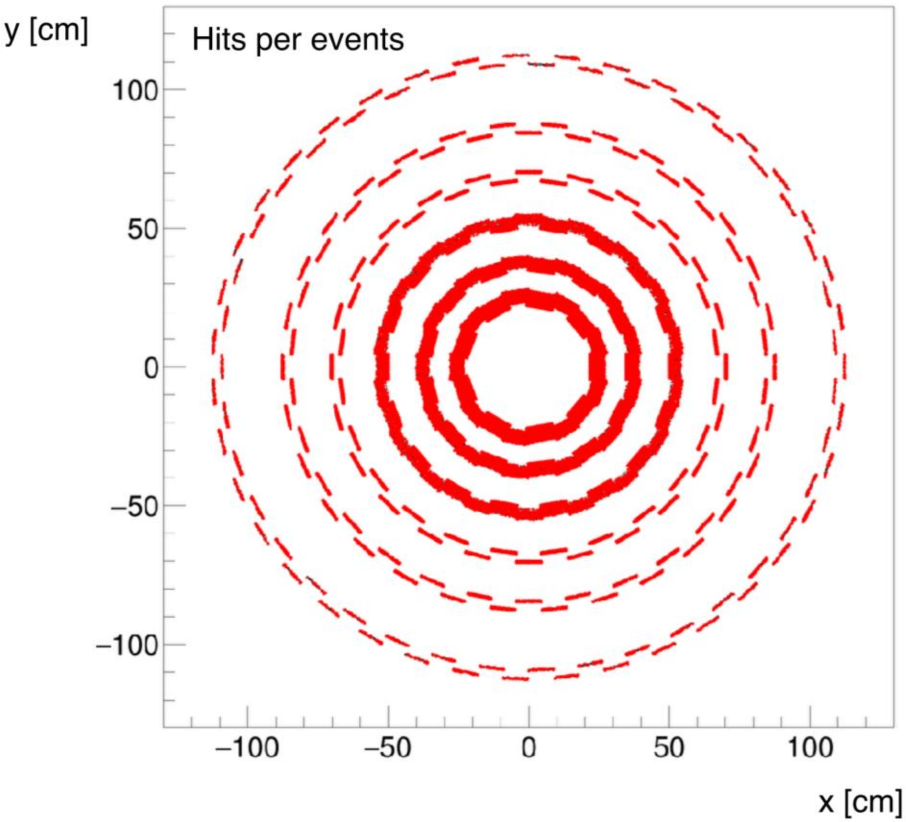
- Pixel Quintuplets and Pixel Triplets cover tracks from interaction point
- Quintuplets reconstruct displaced tracks
- Pixel Line Segments cover tracks in the forward (|η| > 2) region
- "Cross cleaning" in η-ϕ plane reduces duplicates



Physics and Geometrical Considerations

Dealing with Combinatorics

- A typical Pile-up 200 event has approx 100K hits. Naive linking will lead to explosion in tracks
- Physics selections
  - For lower order objects (Mini-doublets, Line Segments), limited information about slope consistent with P_T thresholds
  - For higher order objects (Quintuplets, Pixel Quintuplets, Pixel Triplets) , linear fits in r-z and circle fits in r-ϕ dimensions, and track quality χ^2 cuts



Circle fit

- The circle equation

$$x^2 + y^2 - 2gx - 2fy + c = 0$$

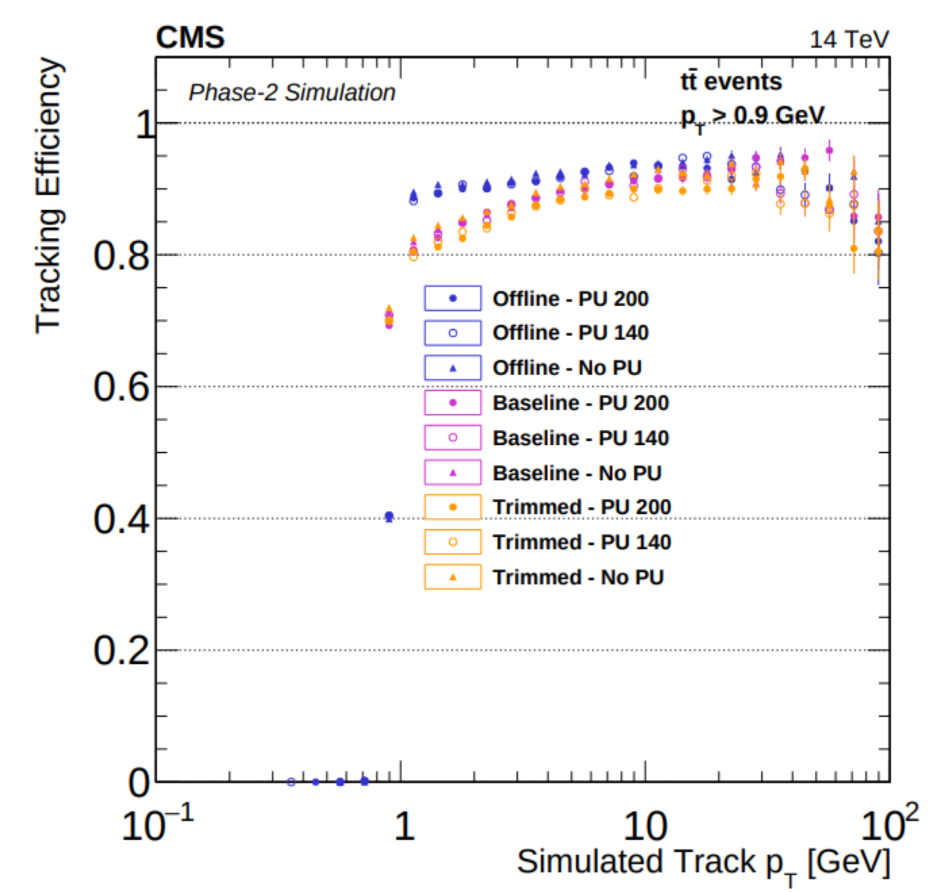
is linear in the parameters (g,f,c)

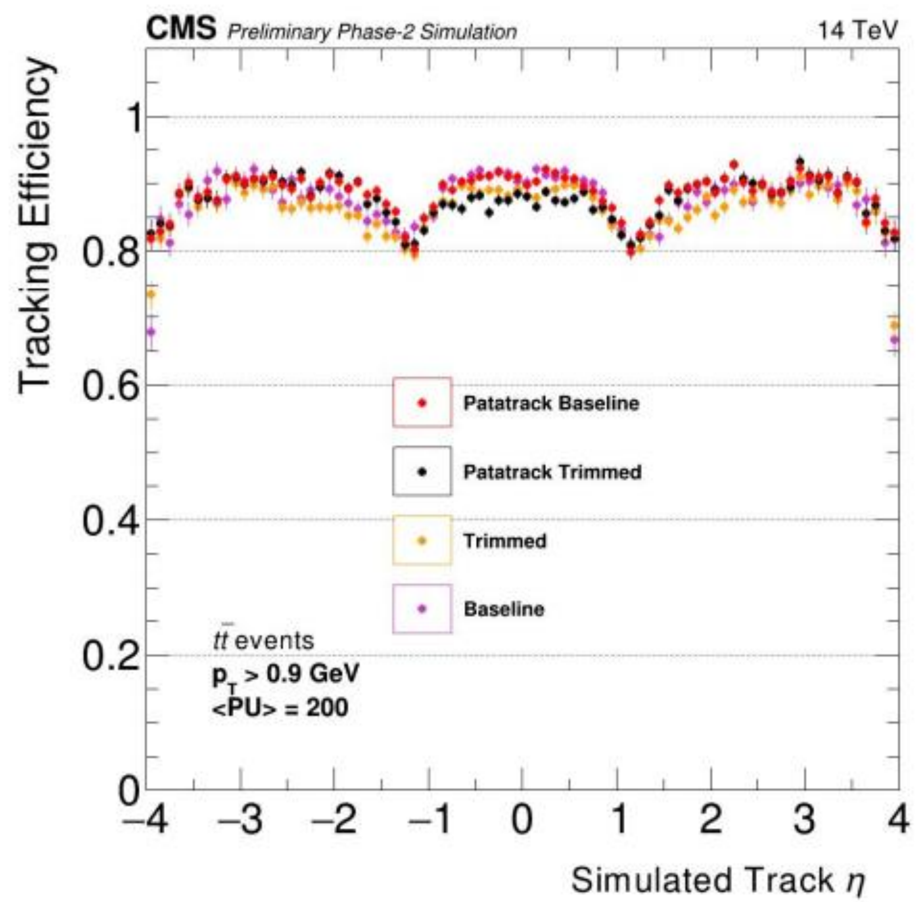
- This equation can be rewritten as

$$(2g)x + (2f)y - c = (x^2 + y^2)$$

- "Target variable" : $x^2 + y^2$, "Feature variables" : (x,y), linear parameters = (2g, 2f, -c)
- Linear fit to these parameters if we have more than three points (akin to fitting a plane in 3D space)
- The number of parameters and the nature of the equation (linear) is known – hardcode least fit solution

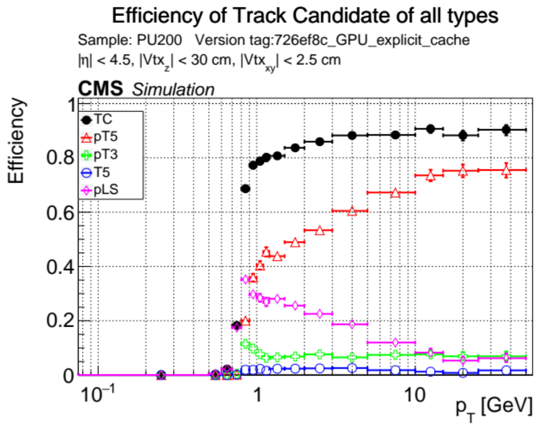
CMS Baseline Efficiency Plots

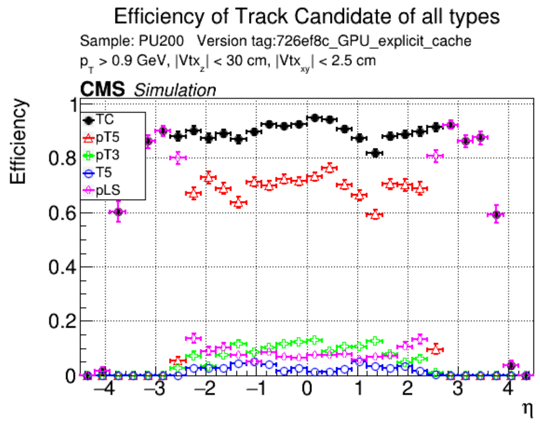


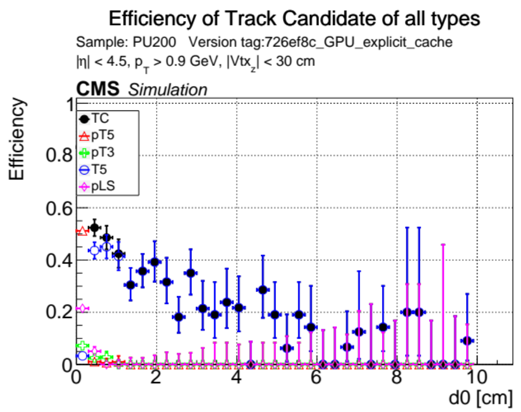


Physics performance

Split by components







Line Segment Tracking on the GPU

Implementation

- Algorithm implemented in CUDA (code) and tested on the Tesla V100 GPU
- Larger objects created from smaller objects – Every step is parallelizable
  - Each object creation step is a separate kernel; relies on results from previous kernel(s)
- Inputs for the algorithm (pixel track stubs from inner tracker, outer tracker hits) already expected to be on the GPU
- Relevant data stored in structure of Arrays (SoA) for efficient SIMT access
  - Custom caching allocators for fast and efficient memory access in GPU
  - 1.2 - 1.5 GB per event pre-allocation
- Transfer to CPU happens at end of the event