# Extending ALICE's GPU tracking capabilities

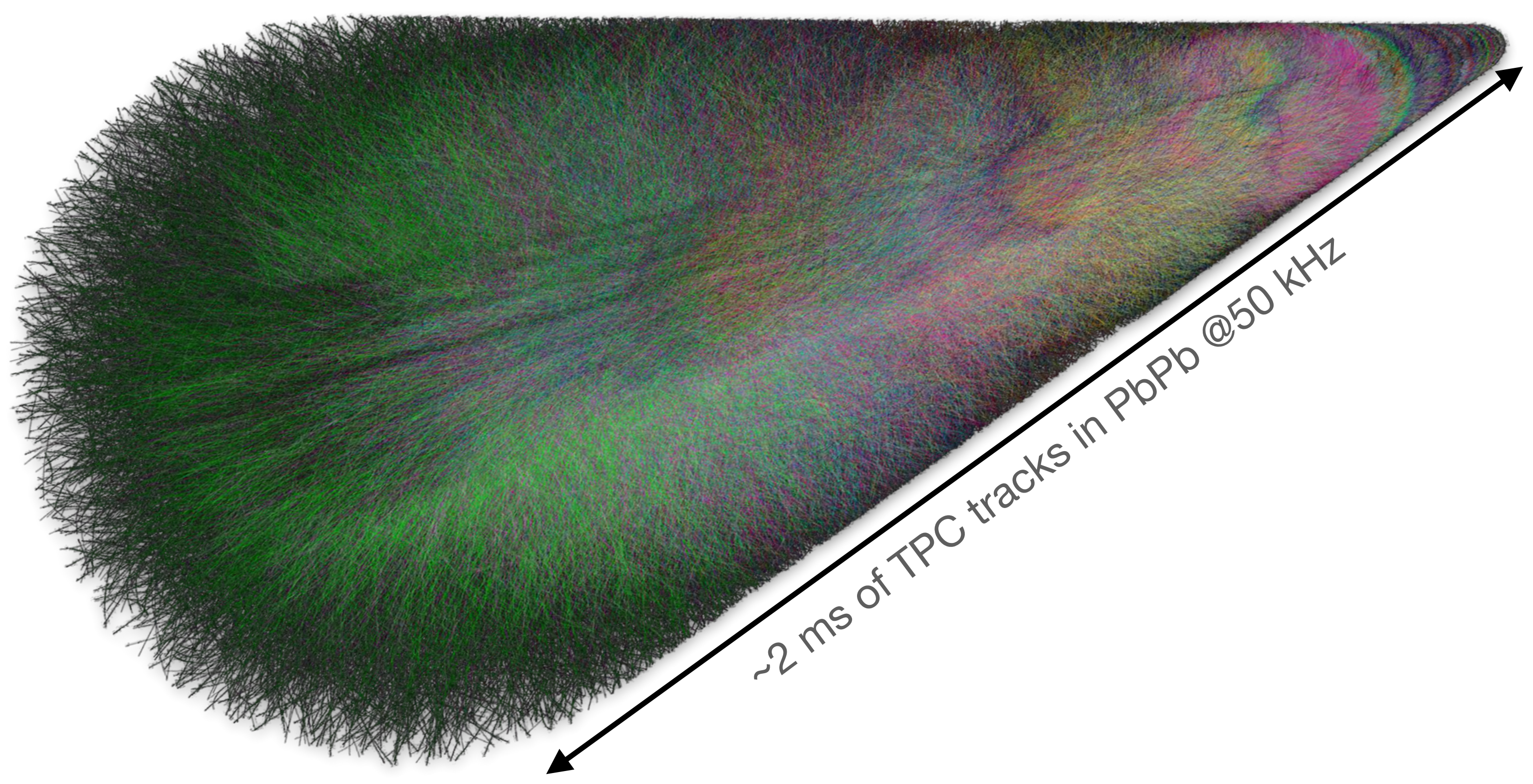
## Towards a comprehensive accelerated barrel reconstruction

Matteo Concas, for the ALICE collaboration

CHEP 2024, October 19th-25th

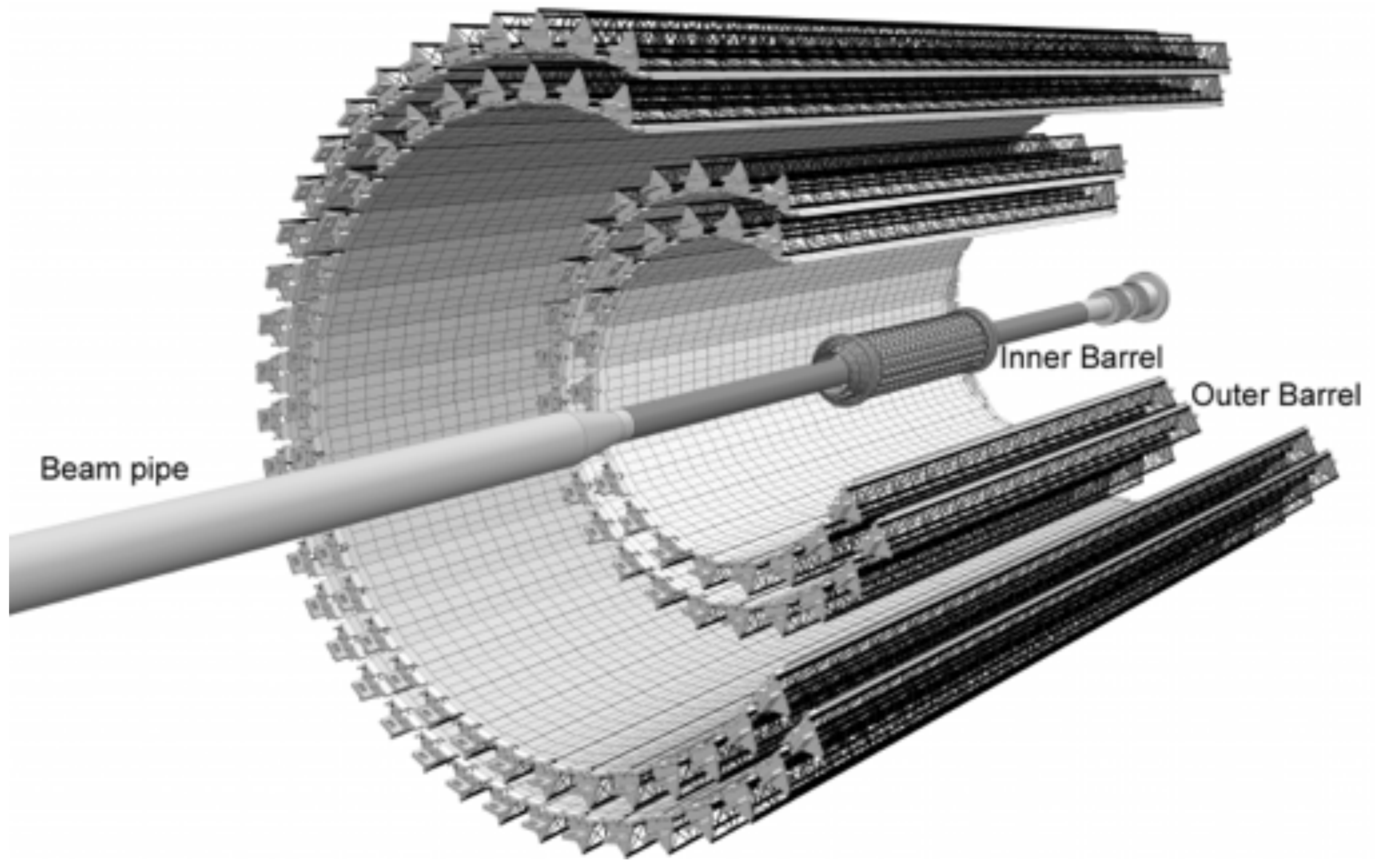
# ALICE reconstruction in Run 3

- Trigger-less acquisition: continuous readout
  - The stream of data is split into O(ms) timeframes.
  - $L_{int}$ >10 $nb^{-1}$ of Pb-Pb data at 50kHz: 50x more than Run 2.
- Reconstruction is two-stepped
  - Synchronous phase (beam circulating): for calibration and data compression.
  - Asynchronous phase (no beam): full processing and production of the AODs.



# ITS reconstruction in Run 3

- A new upgraded Inner Tracking System
  - Provides spatial information in the form of clusters of fired pixels.
- Continuous readout: continuous track reconstruction
  - The atomic time unit is the ITS Readout Frame (ROF): ~4μs.
- Standalone vertex seeding and tracking algorithm
  - During the asynchronous phase is sensitive to secondaries and tracks lower $p_T$.
  - Extensions and adjustments still happen to address, e.g. resource footprint.




ITS tracking

| Timeframe | | | |
|---|---|---|---|
| **ROF 0** | **ROF 1** | **ROF ...** | **ROF N** |
| - clusters<br>- vertices<br>- tracklets<br>- cells<br>- roads<br>**- tracks** | - clusters<br>- vertices<br>- tracklets<br>- cells<br>- roads<br>**- tracks** | ... | - clusters<br>- vertices<br>- tracklets<br>- cells<br>- roads<br>**- tracks** |

**Various intermediate data formats of the ITS tracking.**
**Finally we would like to load clusters on the GPU ad download only the tracks**

# The optimistic scenario

ALICE

- ALICE uses GPUs in production to accelerate the processing
  - During the synchronous TPC processing, the GPU occupancy goes beyond 99%.
- In the asynchronous phase, the fraction of available GPU increases
  - Running additional reconstruction steps on GPU would optimise the resource usage.
- ALICE is working towards having full-barrel tracking on GPU[1]

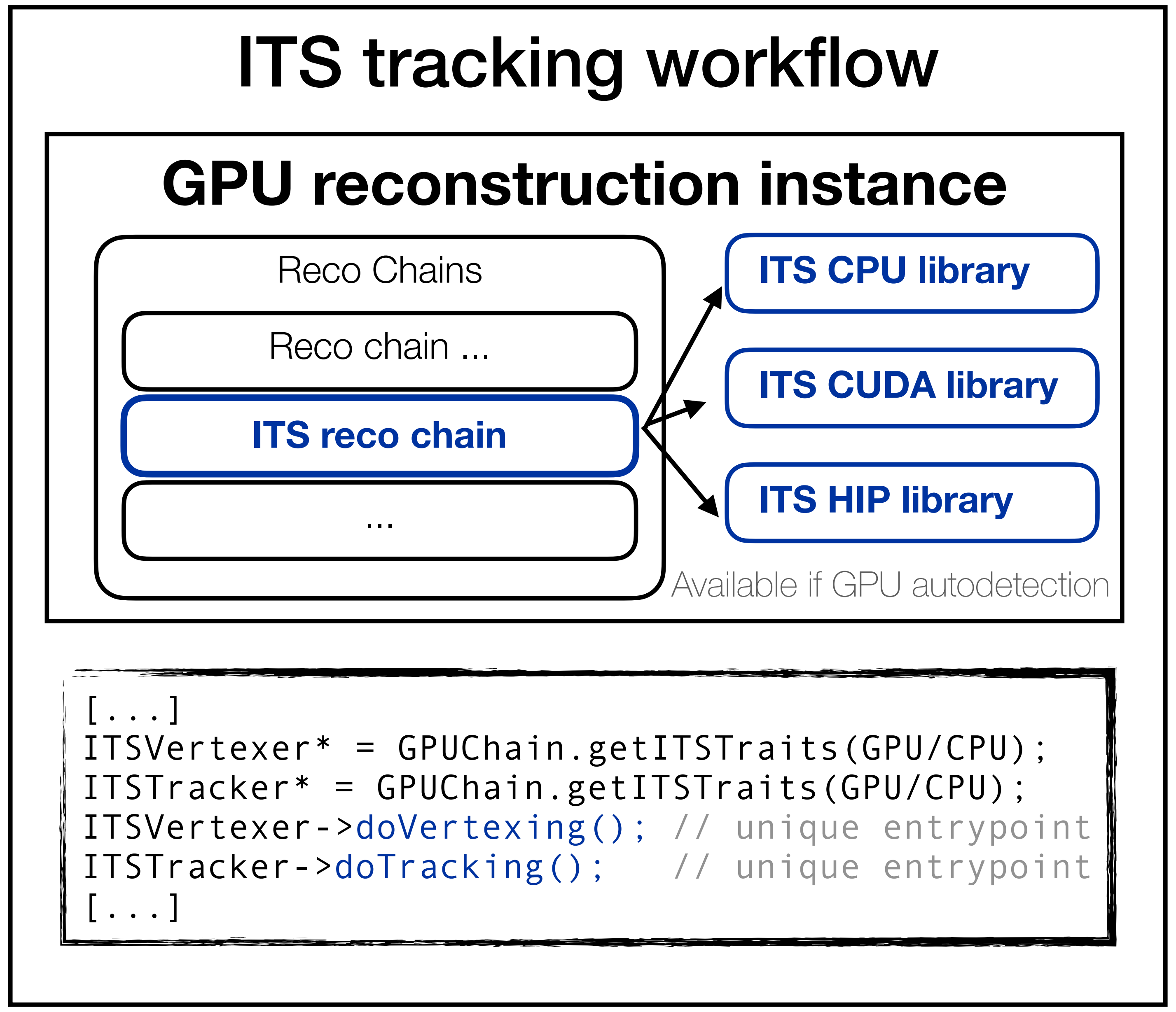
| Processing step | % of time |
|---|---|
| TPC Processing | 52.39 % |
| **ITS Tracking** | 12.65 % |
| **Secondary Vertexing** | 8.97 % |
| MCH | 5.28 % |
| TRD Tracking | 4.39 % |
| TOF Matching | 2.85 % |
| ITS TPC Matching | 2.64 % |
| Entropy Decoding | 2.63 % |
| AOD Production | 1.72 % |
| Quality Control | 1.64 % |
| Rest | 4.84 % |

**Asynchronous processing of Pb-Pb @ 47kHz:**
**Relative percentages change with different interaction rate**

[1] Improvements of the GPU Processing Framework for ALICE (D. Rohr)

# Integrate GPU usage for the ITS in O²

- **GPU reconstruction workflow** steers any GPU-related task.
- Framework[1] for a centralised management of the GPUs
  - Dynamically load the required libraries as additional plug-in components.
  - It abstracts access to the GPU resources and singletons.
- There is flexibility in designing the porting of more components
  - ITS GPU tracking is a standalone library pluggable into the primary GPU framework!




**Sketch of the integration of the ITS GPU libraries as plugins for the framework**

# ITS tracking on GPU

ALICE

- Hybrid implementation
  - Choice of which step to run on CPU or GPU to facilitate the debugging.
- Currently migrating from a standalone implementation 🚧
  - Previously: manual memory allocation and independent access to GPU.
  - Now: integrating steps within the GPU main framework
- Track fitting is now ported and fully operational
  - Propagation utility, the critical component, is provided by the central framework.
- Support for AMD and Nvidia
  - Plain CUDA codebase, automatically translated to HIP at compile time.

| Vertexer | | |
|---|---|---|
| Tracklet Finder | ✅ | standalone |
| Tracklet Selection | ✅ | standalone |
| Vertex Fitter | ✅ | standalone |

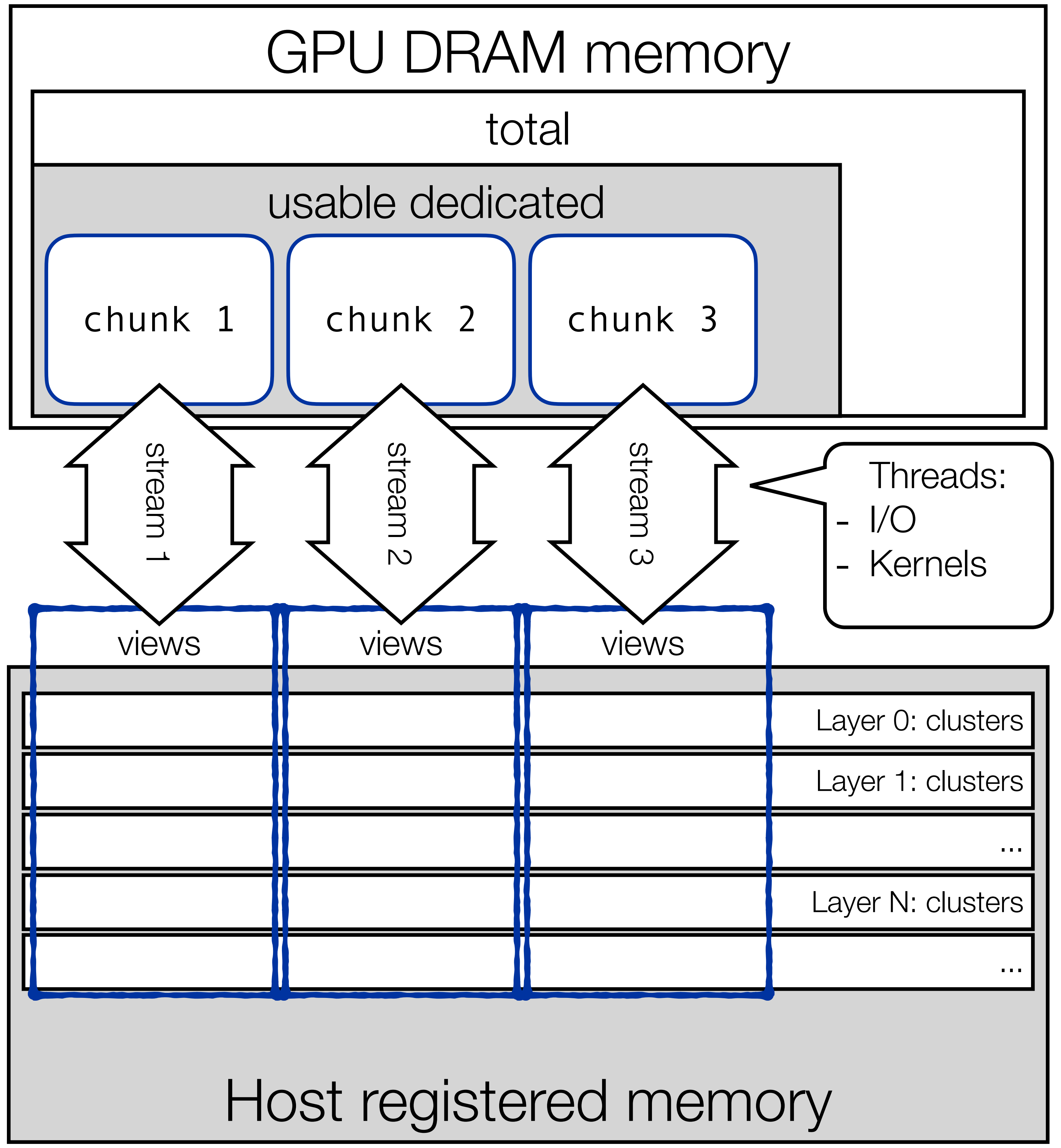
| Tracker | | |
|---|---|---|
| Tracklet Finder | ✅ | *standalone* |
| Tracklet duplicate finder | ✅ | *standalone* |
| Cell finder | ✅ | *standalone* |
| Cell neighbour finder | ✅(*) | *standalone* |
| **Track fitting** | ✅(*) | **integrated** |

**Teardown of the ITS tracking reconstruction steps.**
**In light blue are the standalone routines.**
**In yellow are the Framework-compatible ones.**

(*) Recent improvements and refactoring of the CPU algorithm footprint broke the hybrid compatibility.
GPU code is being updated accordingly.

# Cornerstones of the GPU pattern recognition

- **Cellular Automaton**: provides track candidates to the fitting
  - Highest memory usage: due to the combinatorial nature of the algorithm.
- **Total** available memory is **partitioned into chunks**
  - Timeframes are fractioned and processed in chunks.
- Multi-stream processing of **bunches of ROFs**
  - Each tracking instance is almost independent of the others (shared borders).
  - I/O operations on one stream are hidden behind kernel executions.
- Finalised already[2], it is being integrated with the framework

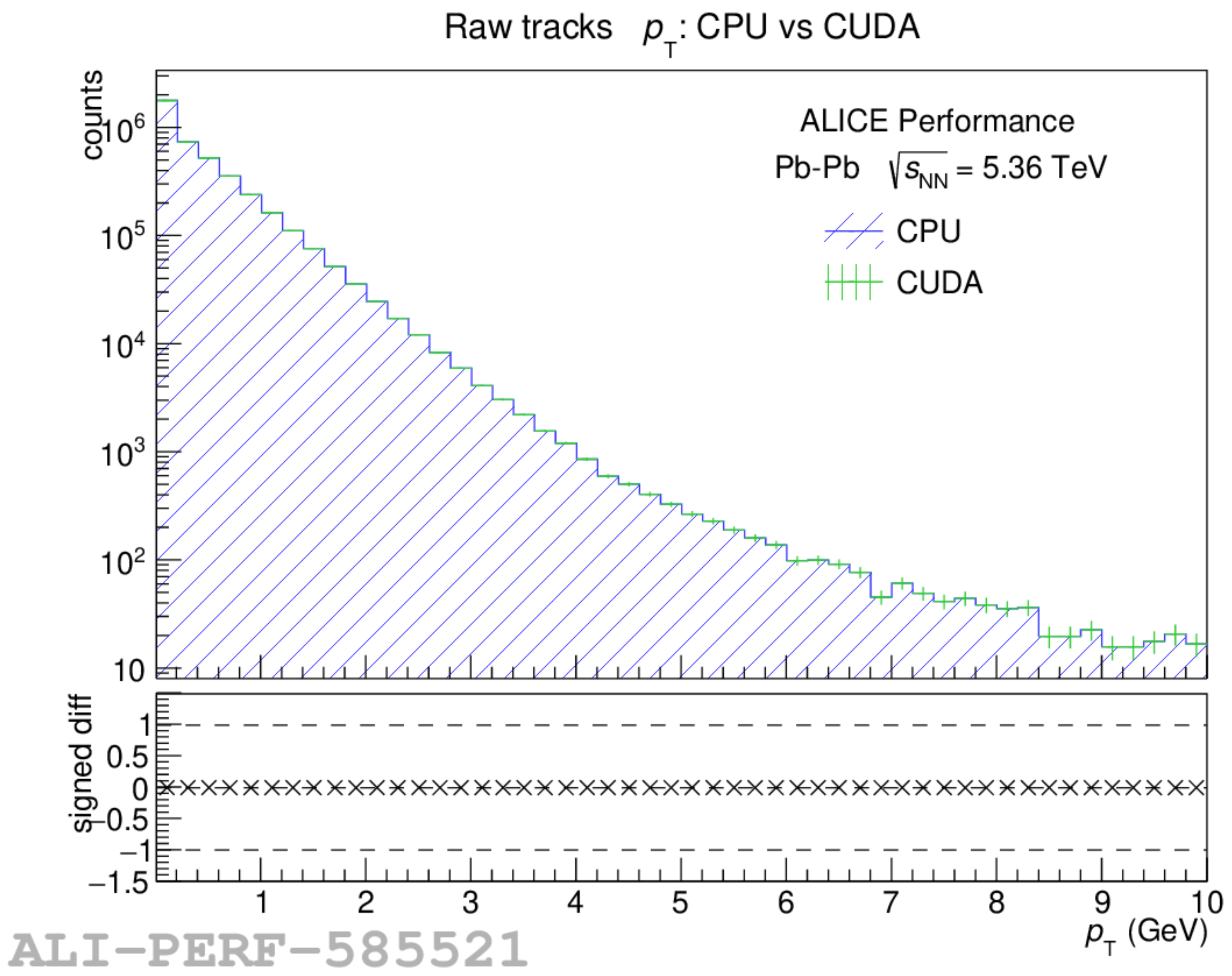


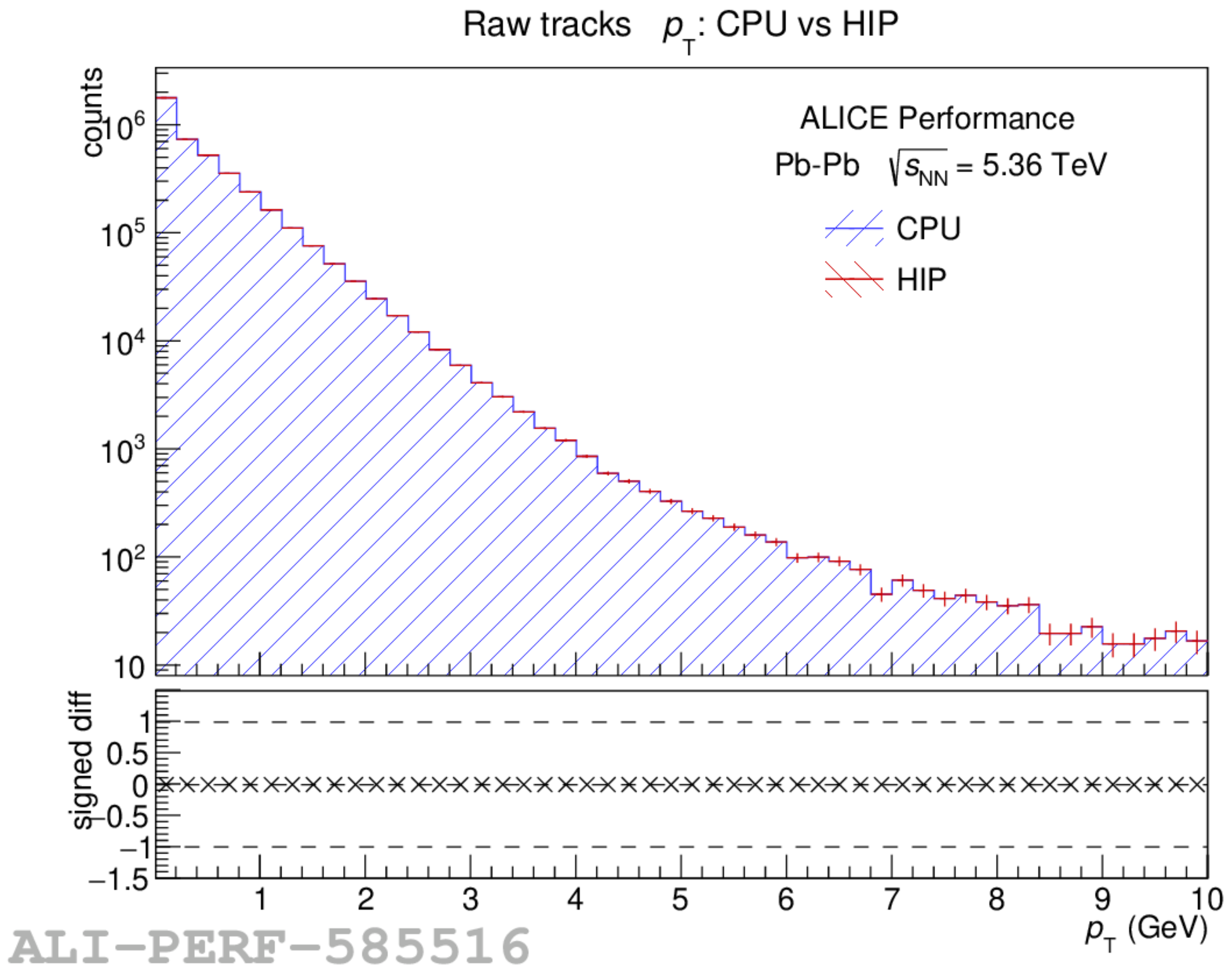

**Memory partition in the multi-streamed ITS pattern recognition part**

[2] CHEP 2023 reminder

# Comparing results with deterministic mode

- Results discrepancy in CPU vs GPU typically expected
  - Due to inherently different computing architectures.
  - Usually accepted and added to the systematics.
- A *deterministic* mode is available for $O^2$
  - It just requires a re-compilation.
- Ensures perfect consistency of the output
  - It kills the performance, and it is to be used for checks.
  - A potent debugging tool: spotted several bugs and hiccups.

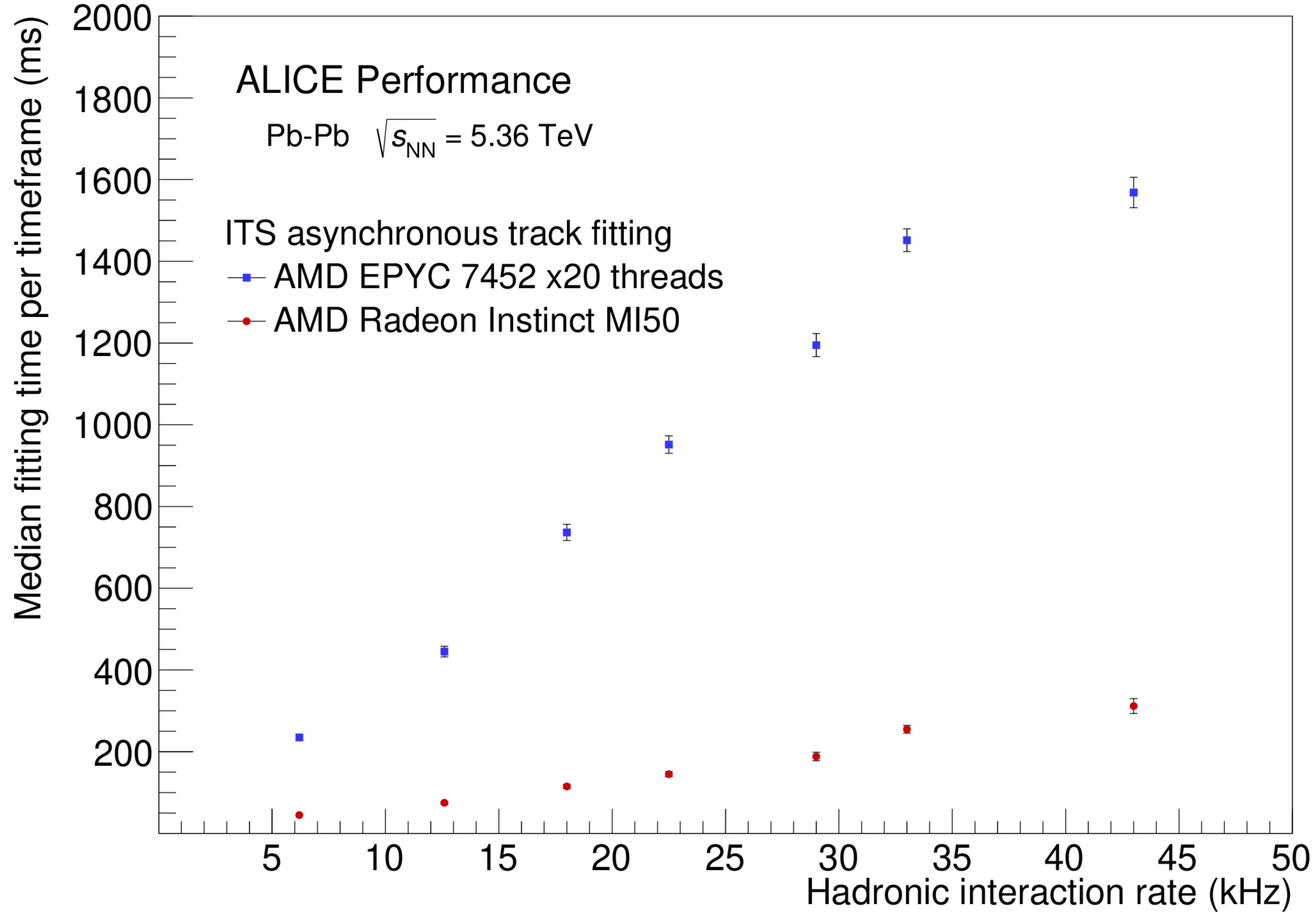



**Comparison of $p_T$ distribution of raw reconstructed tracks using ITS CPU and GPU with CUDA and HIP**

# ITS track fitting on GPU

- A timeframe of data is processed at once
  - In Pb-Pb, the number of fits is up to ~300K/TF.
  - At the highest Pb-Pb rate, memory is up to 500 MB.
- ITS tracking runs with up to 20 threads
  - GPU has a broader computing scaling for the ITS fitting.
- Is this useful already for Run 3?
  - Having just the ITS fitting on GPU would help.

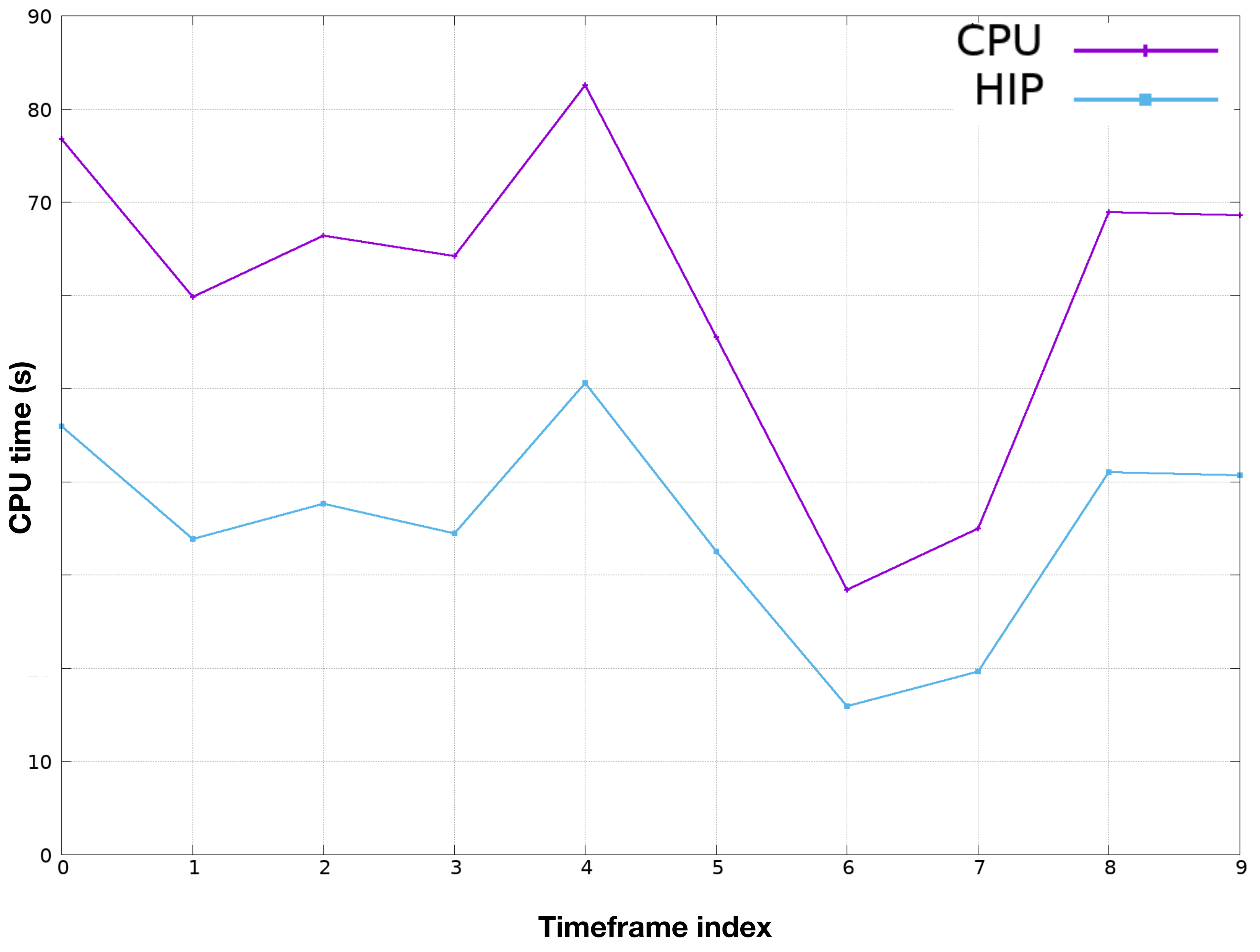



ALI−PERF−585476

**Time comparison for ITS track fitting per timeframe on CPU using 20 threads and GPU as a function of the average hadronic interaction rate.**

# In the optimistic scenario

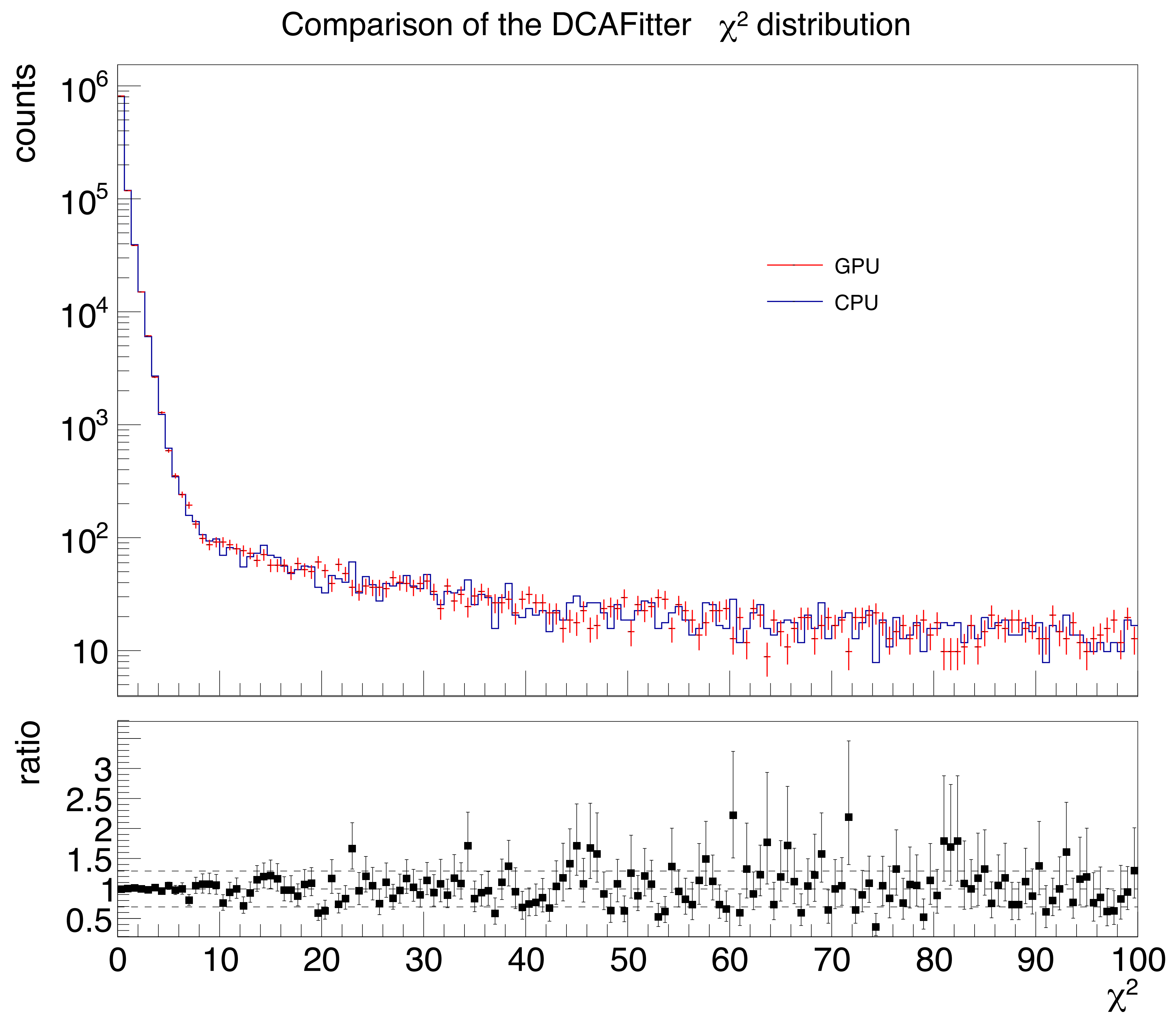
- Track fitting was the most impactful step to the CPU time
- Refactoring increased the relevance of pre-selection part
  - To reduce the memory footprint and cope with Grid job constraints.
  - Pre-selections are inherently parallel and use fits!
- Porting the pre-selections on GPU: ~50% of the total time
  - Moving it to GPU would improve our resource efficiency.




Showcase of the elapsed wall time for one thread CPU (purple) vs GPU(light blue).

# Secondary vertexing on GPU

- **DCAfitter**: a well-established tool used across $O^2$ code
  - Associate tracks using relative DCAs with different minimisation options.
- C++ class successfully ported and usable on GPU
  - Dependency from ROOT SMatrix: A minimal copy of it ported to $O^2$.
  - It is not yet possible to use deterministic mode for the validation.
- Currently a proof of concept, but promising results already
  - Speedup will be measured on actual use cases.
  - A first toy demonstrator has been used in a physics analysis as a p.o.c.




**Comparison of the $\chi^2$ distribution on a synthetic test of 1M fits. Results are promising but need to be better understood.**

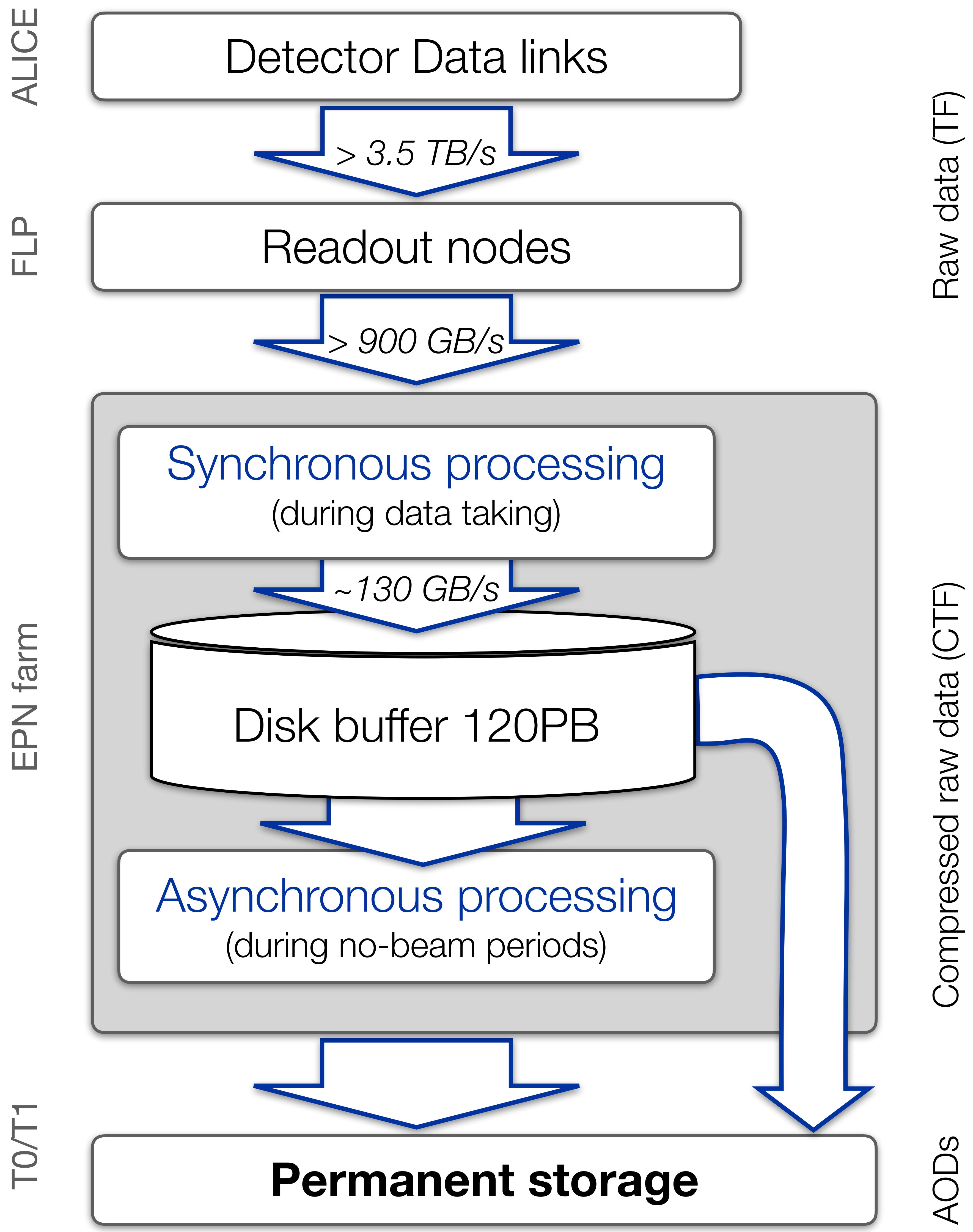
# Conclusions and outlook

- ALICE is pursuing the optimistic scenario for GPU processing
  - The target is to have the full barrel tracking running on GPUs.
- ITS has a GPU implementation for all of the components of the tracker
  - ITS Track fitting is the most promising and already integrated: we aim to move it to the GPU.
  - A good check is to target the asynchronous reconstruction of PbPb 2024 with GPU track fitting.
- DCA fitter has been successfully ported on the GPU
  - It is spread across many $O^2$ use cases, including the secondary vertex reconstruction.
  - Its adoption in some combinatorics-dominated physics analyses would be a nice by-product.

# Backup

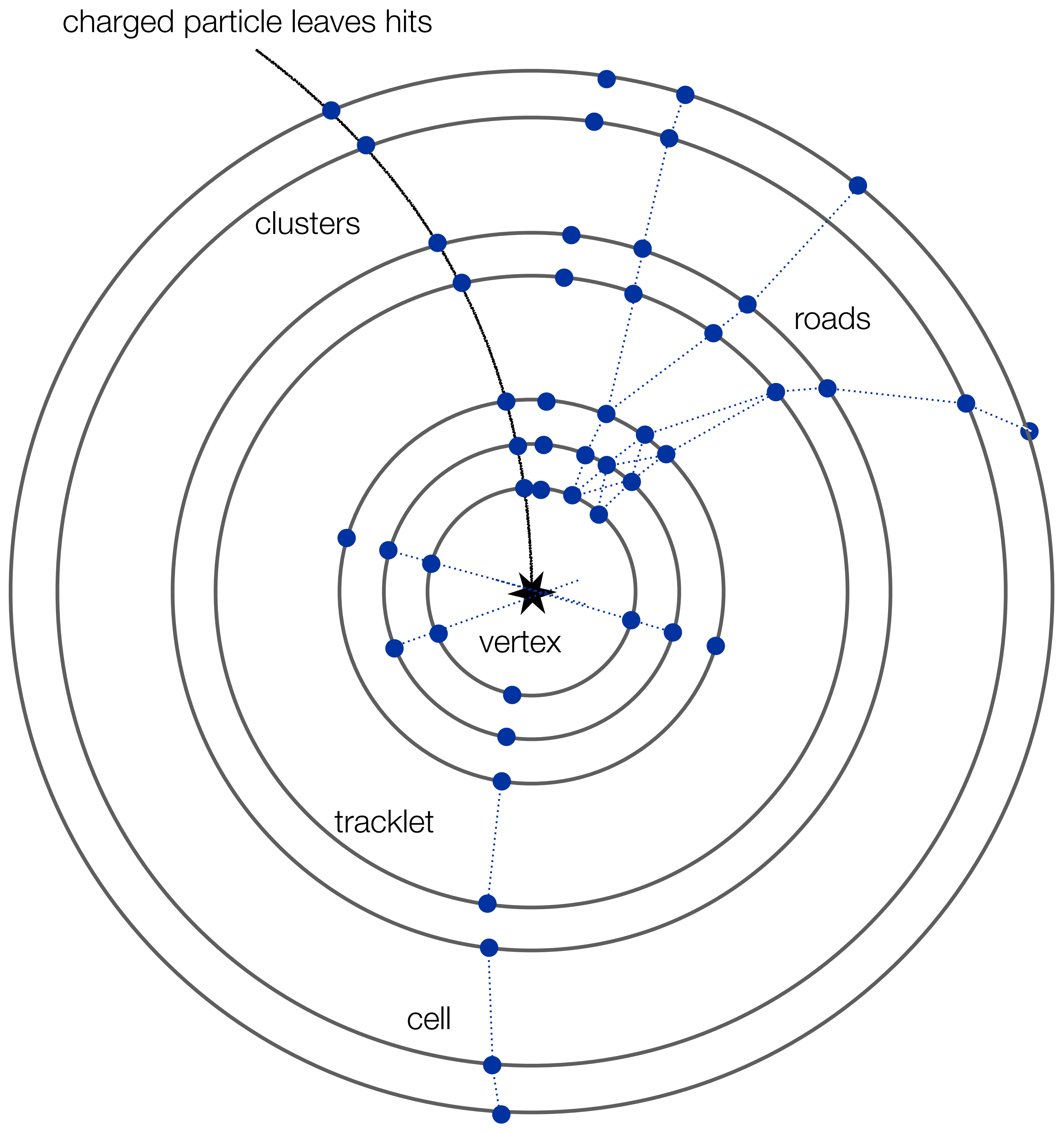
# ALICE data processing for Run 3

- Online reconstruction and calibration for data compression
  - Synchronous: TPC full reconstruction and calibration.
  - Asynchronous: all compressed data are reconstructed.
  - Single computing framework for online-offline computing: $O^2$.
- Operate part of the reconstruction on GPUs is *mandatory*
  - Minimise the cost/performance ratio for online farm
  - 250x Event Processing Nodes (EPNs), 8x AMD MI50 GPUs
- Efficient utilisation of available computing resources is desired
  - A larger fraction of GPUs available during the asynchronous phase



# ITS vertexing and tracking

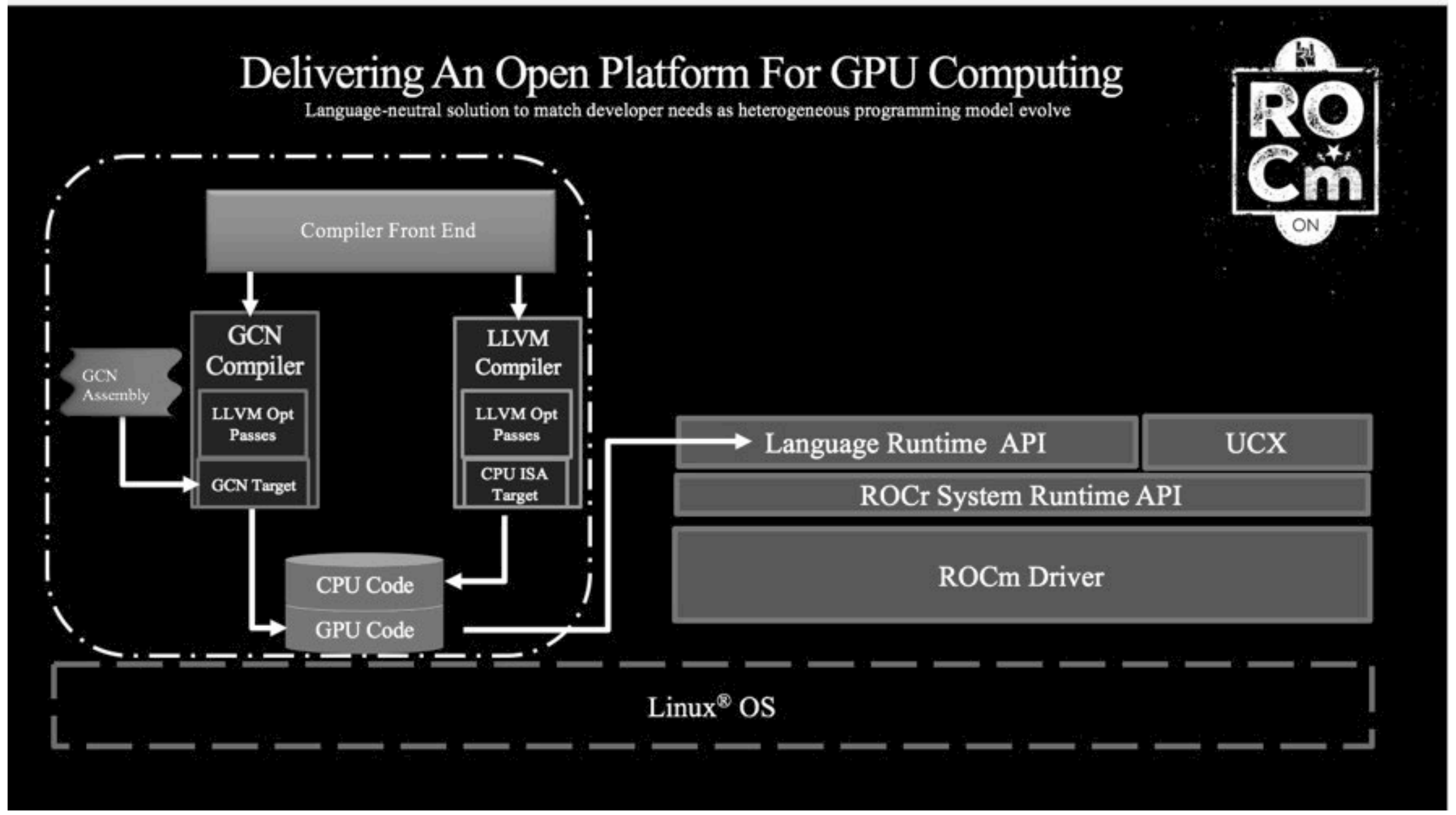
- Primary vertex seeding
  - Combinatorial matching followed by linear extrapolations of *tracklets.*
  - Unsupervised clustering to find the collision point(s).
- Track finding and track fitting
  - It uses vertex position to reduce the combinatorics in matching the hits.
  - Connect segments of tracks, the *cells*, into a tree of candidates: *roads.*
  - Kalman filter to fit tracks from candidates.
- The algorithm is decomposable into multiple parallelisable steps
  - Each ROF can be processed independently(*).
  - In-frame combinatorics can be processed simultaneously.




(*) Information from adjacent ROFs can be used to recover from information splitting

# Heterogeneous-Compute Interface for Portability

- Support GPUs from two main vendors:
  - CUDA language and runtime for Nvidia
  - HIP language and ROCm runtime for AMD
- HIP: a C++ Runtime API and Kernel language
  - Portable AMD and NVIDIA applications from single source code
  - It is shaped around CUDA APIs to ease translation
  - CUDA libraries, like Thrust and CUB, have their HIP versions using ROCm
- ROCm has tools to translate CUDA to HIP automatically
  - `hipify-clang`: based on Clang, actual code translation
  - `hipify-perl`: script for line-by-line code conversion
- Strategy: maintain only the CUDA code and generate HIP



# Cross-platform on-the-fly code generation

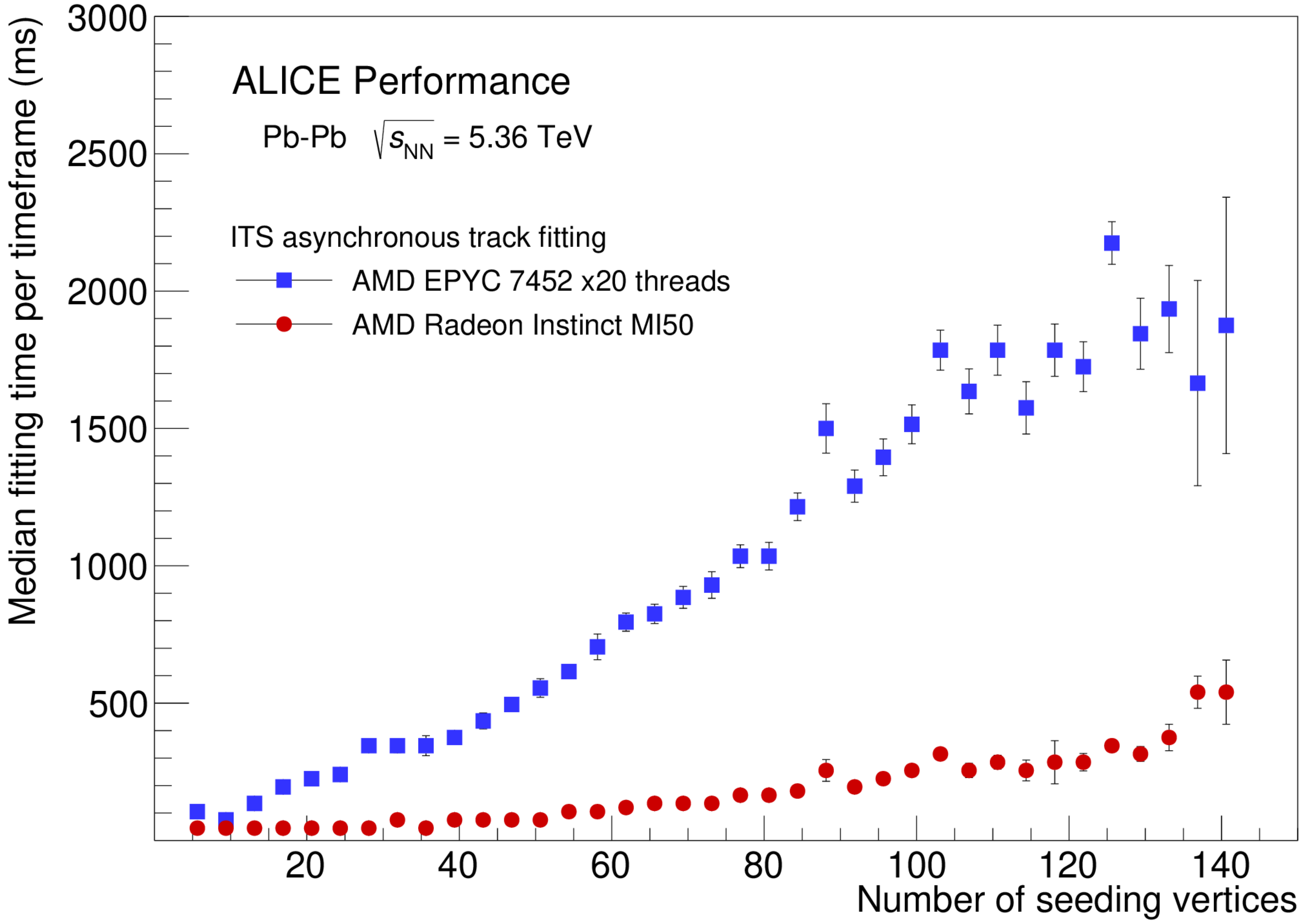
ALICE

- The O2 compilation via CMake, provides
  - Platform autodetection and production of corresponding target libraries
  - Custom commands setting dependencies between targets
- HIP code is generated in place from CUDA sources
  - Build source of targets parsing CUDA files and generating HIP versions
  - Currently based on `hipify-perl`: is run on all `.cu` files to produce HIP
- Headers files are shared across both the compilations
  - Negligible boilerplate (<0.1% LoCs) to cope with some architectural differences

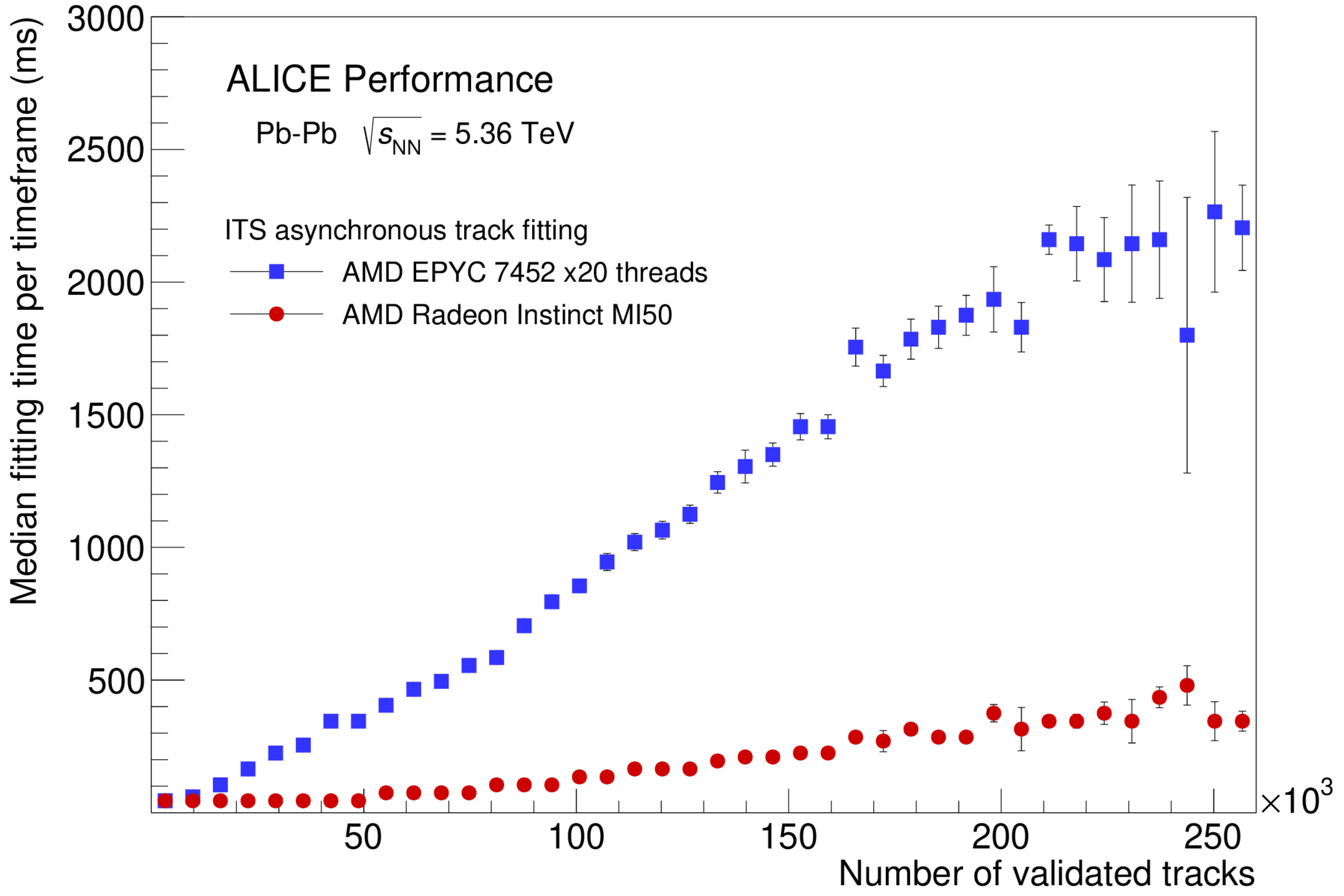
```
// CUDA code
cudaMalloc(&A_d, Nbytes);
cudaMalloc(&C_d, Nbytes);
cudaMemcpy(A_d, A_h, Nbytes, cudaMemcpyHostToDevice);

vector_square <<<512, 256>>> (C_d, A_d, N);
cudaMemcpy(C_h, C_d, Nbytes, cudaMemcpyDeviceToHost);

// HIP code, translated
hipMalloc(&A_d, Nbytes);
hipMalloc(&C_d, Nbytes);
hipMemcpy(A_d, A_h, Nbytes, hipMemcpyHostToDevice);

hipLaunchKernelGGL(vector_square, 512, 256, 0, 0, C_d, A_d, N);
hipMemcpy(C_h, C_d, Nbytes, hipMemcpyDeviceToHost);
```

# Scaling of the ITS fitting

- Showcase of the scaling of the computing time for the track fitting




ALI-PERF-585491




ALI-PERF-585486

Time comparison for ITS track fitting per timeframe on CPU using 20 threads and GPU as a function of the number of seeding vertices (left) and validated track multiplicity (right).