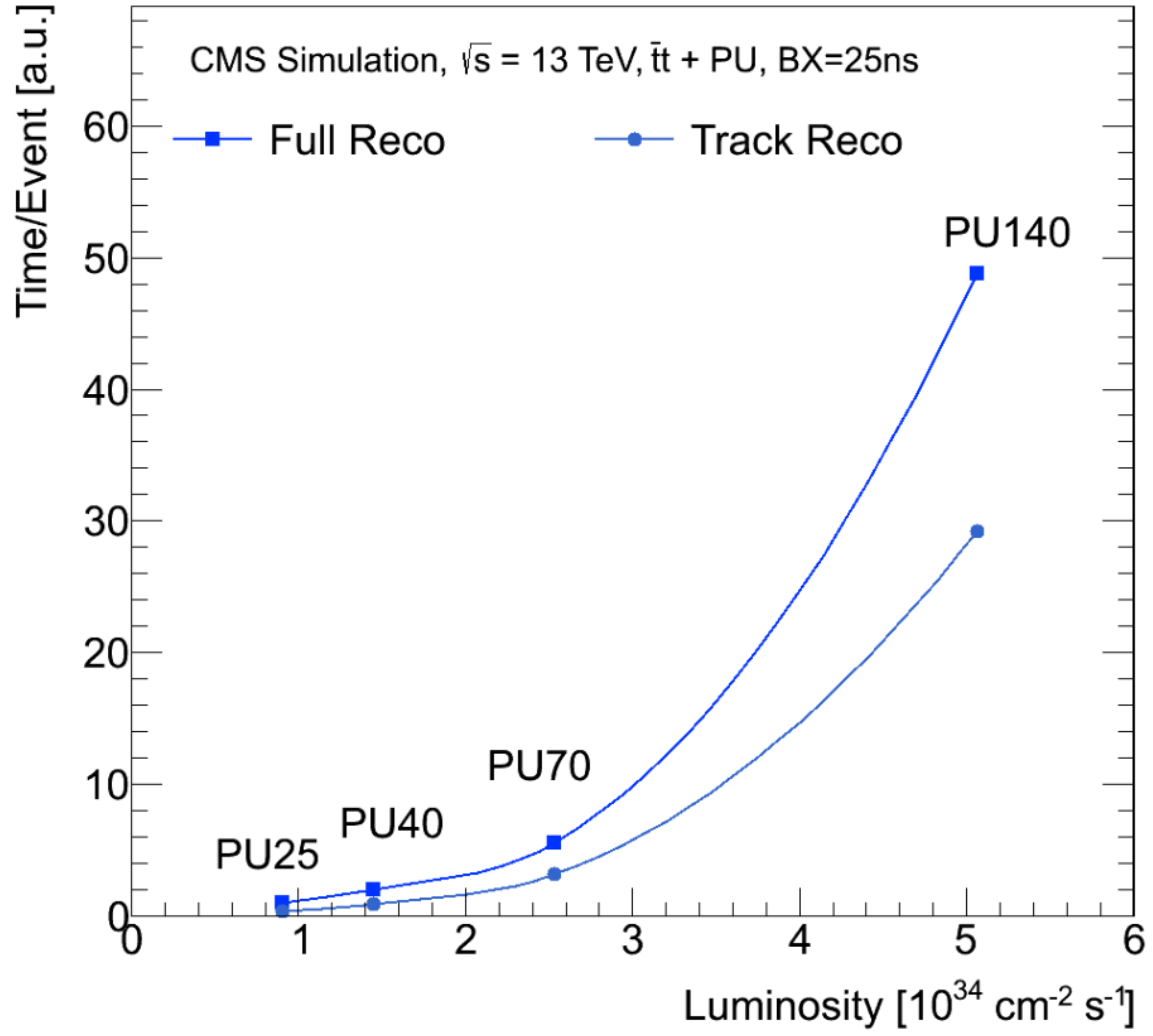
Reconstruction of Charged Particle Tracks in Realistic Detector Geometry Using a Vectorized and Parallelized Kalman Filter Algorithm

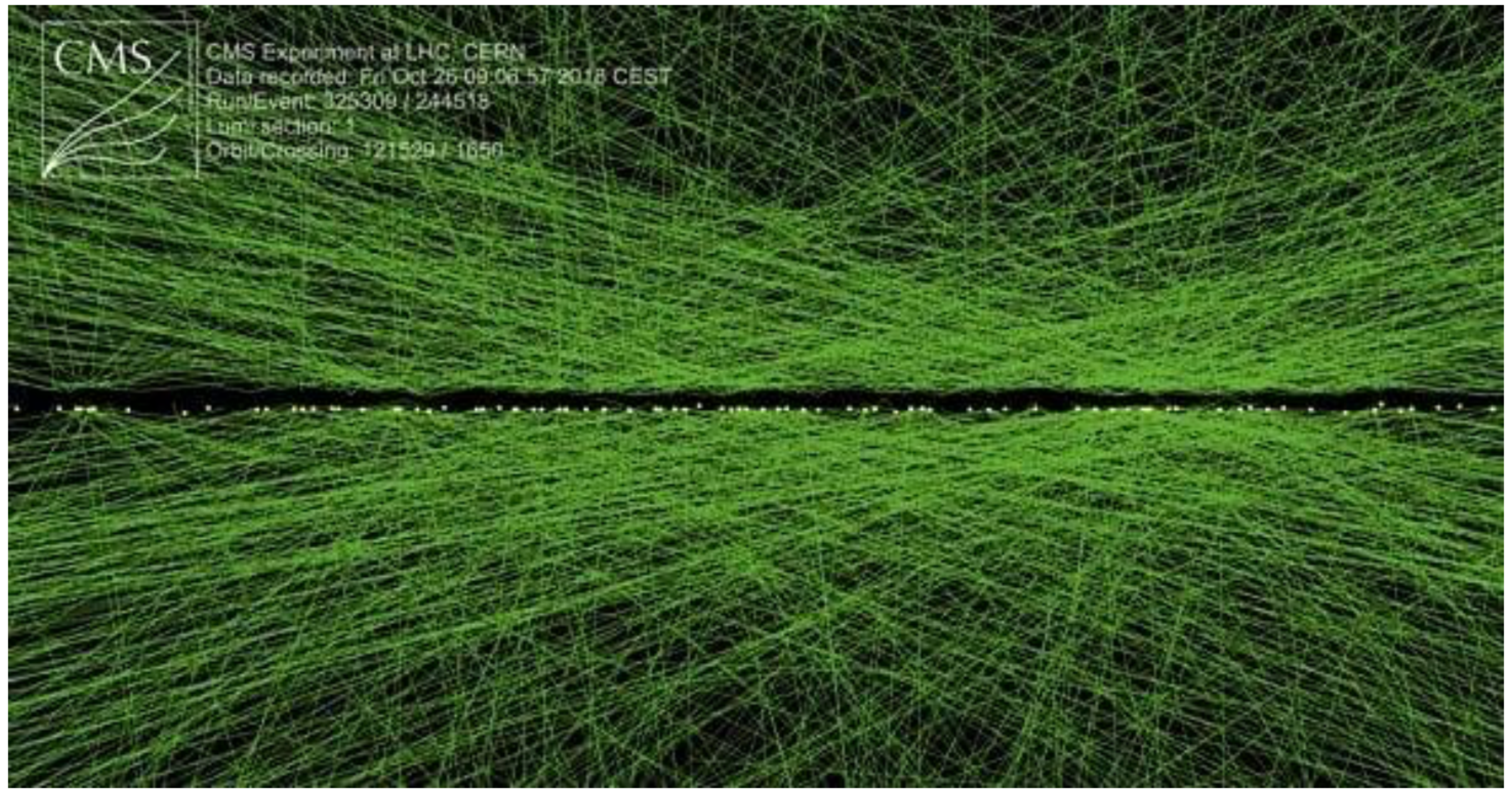
S.Lantz, K.McDermott, M.Reid, D.Riley, P.Wittich (Cornell); G.Cerati, A.Reinsvold Hall, M.Kortelainen (Fermilab); P.Elmer, B.Wang (Princeton); S.Krutelyov, M.Masciovecchio, M.Tadel, F.Würthwein, A.Yagil (UCSD); B.Gravelle, B.Norris (UOregon)

CHEP2019 - Nov. 07, 2019

Tracking "Problem"

- Tracking is **crucial for the physics** goals of the LHC experiments
  - charged particle momenta, particle ID, jet tagging, jet&MET resolution
- It is the **most time consuming** reco task
  - and scales poorly with pile-up, problem for HL-LHC
  - challenge especially for High Level Trigger (HLT)
- Two options in front of us:
  - save time by reducing the tracking phase space
    - with consequences on the experiments' physics reach
  - save time by making tracking faster! Requires **R&D**…



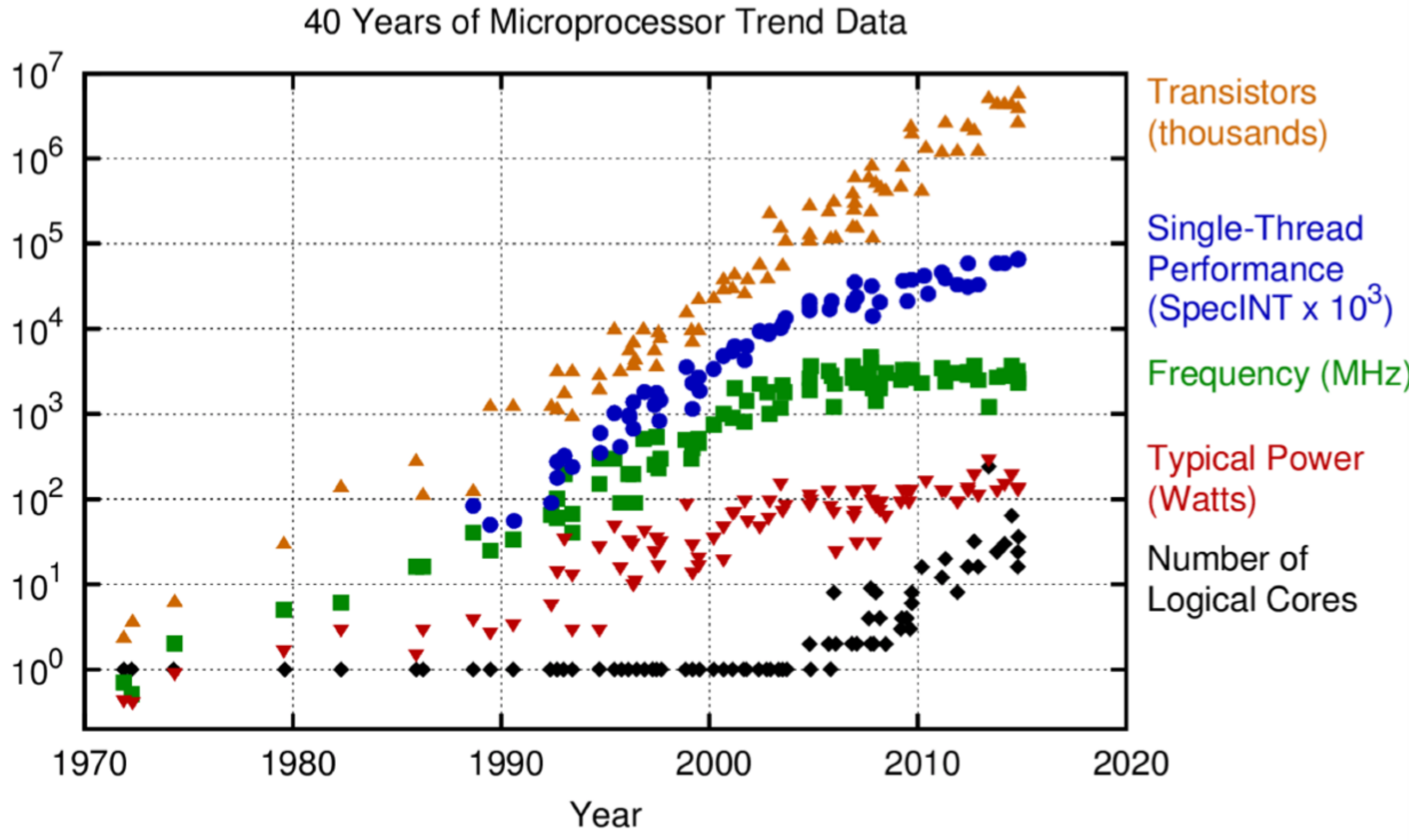


Event display, CMS 2018 high PU run (PU 136)

Moore’s Law

- CPU frequency stopped growing exponentially:
  - nothing for free anymore
- Since 2005, most of the gains in single-thread performance come from **vector operations**
- But number of logical cores is rapidly growing too: **multi-threading**

- Must exploit both levels of parallelization to avoid sacrificing on physics performance!

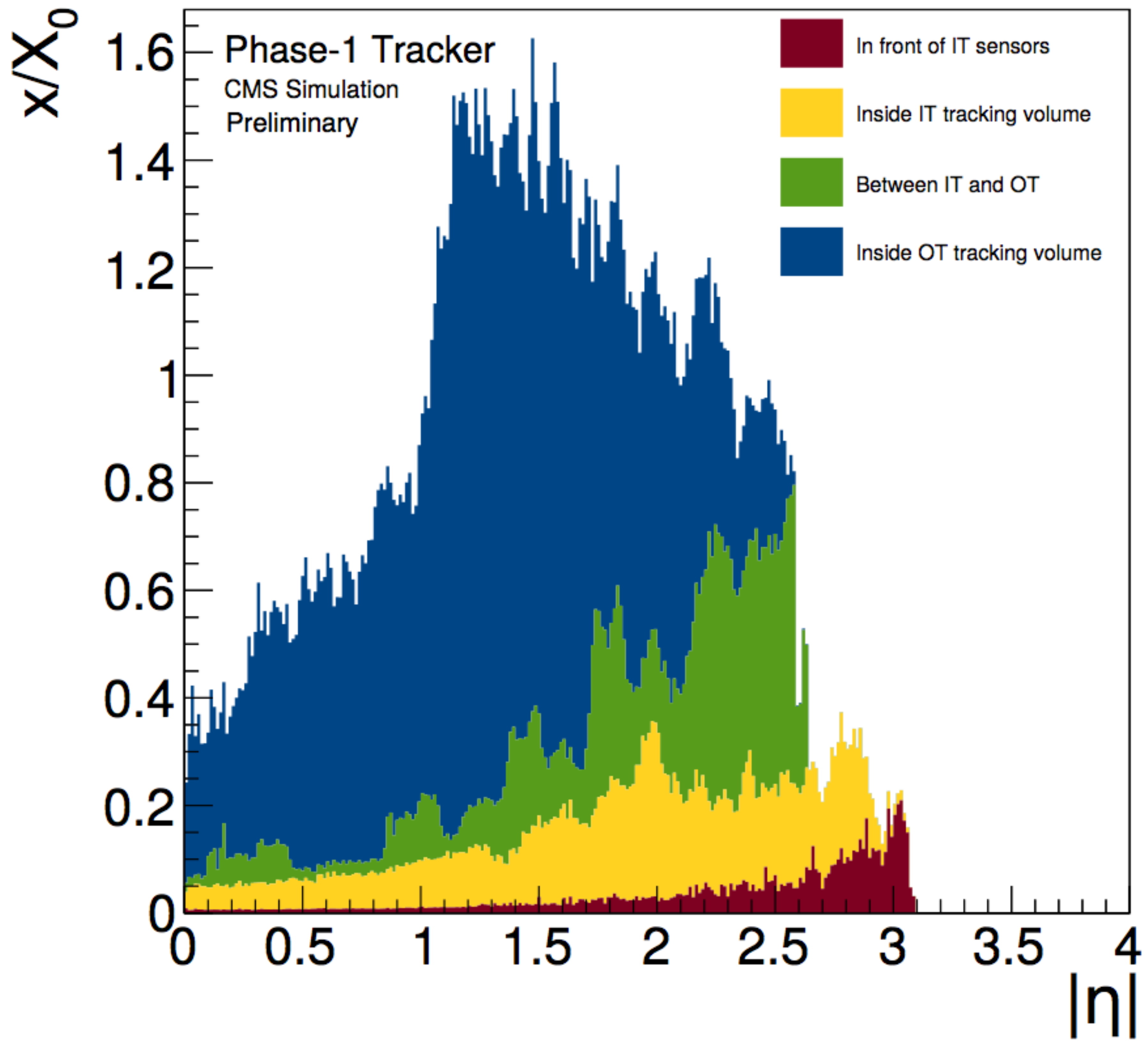


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

mkFit Project

- Ongoing for ~5 years, well advanced
- Collaboration between physicists and computer scientists from Cornell, Fermilab, Princeton, UCSD, UOregon
  - funding from NSF IRIS-HEP, DOE SciDAC, USCMS
  - http://trackreco.github.io/

- Mission: **speedup Kalman filter (KF) tracking** algorithms using highly parallel architectures
- Why sticking to KF?
  - Widely used in HEP in general, and CMS in particular
  - Demonstrated **high efficiency** physics performance
  - Robust handling of **material effects**

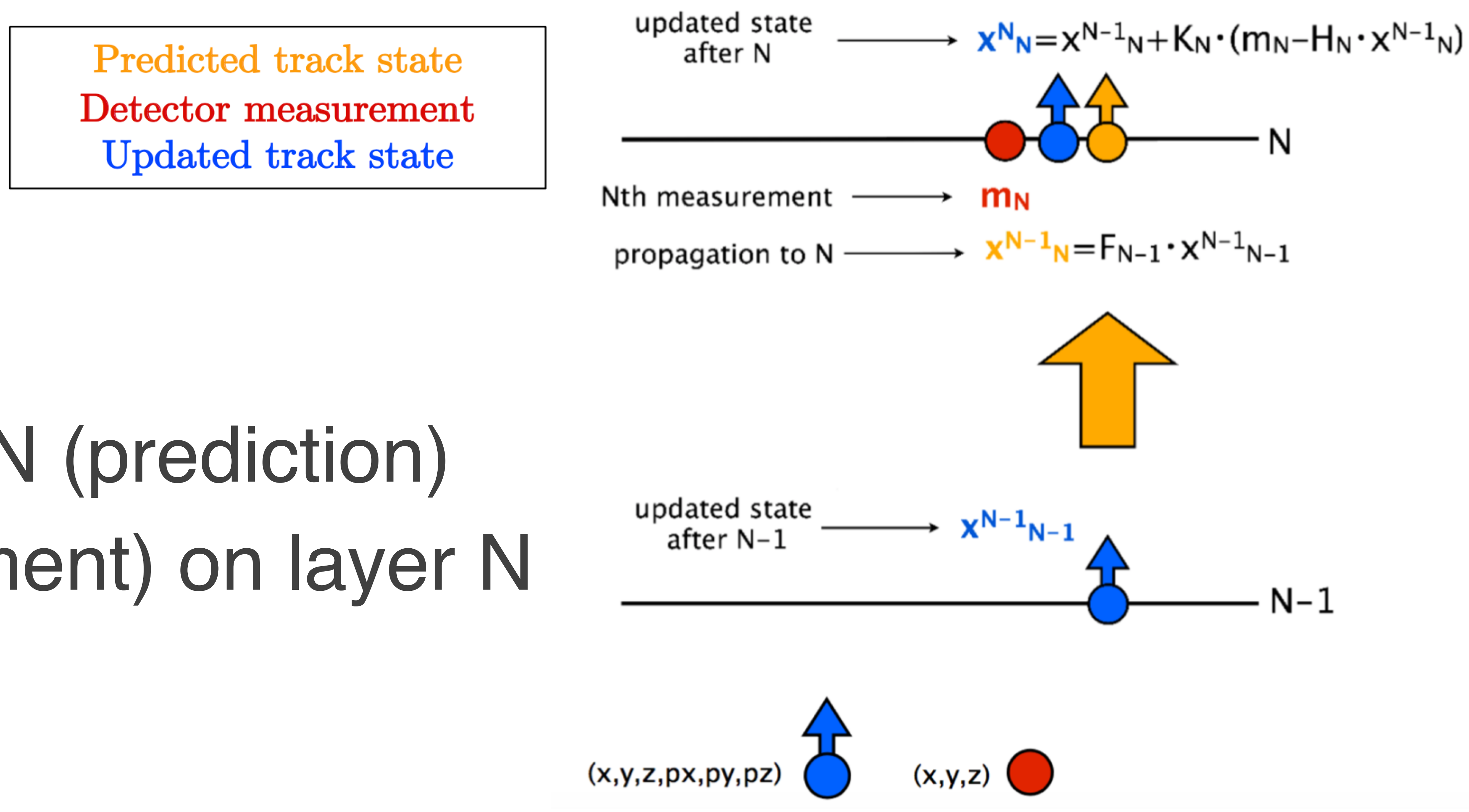


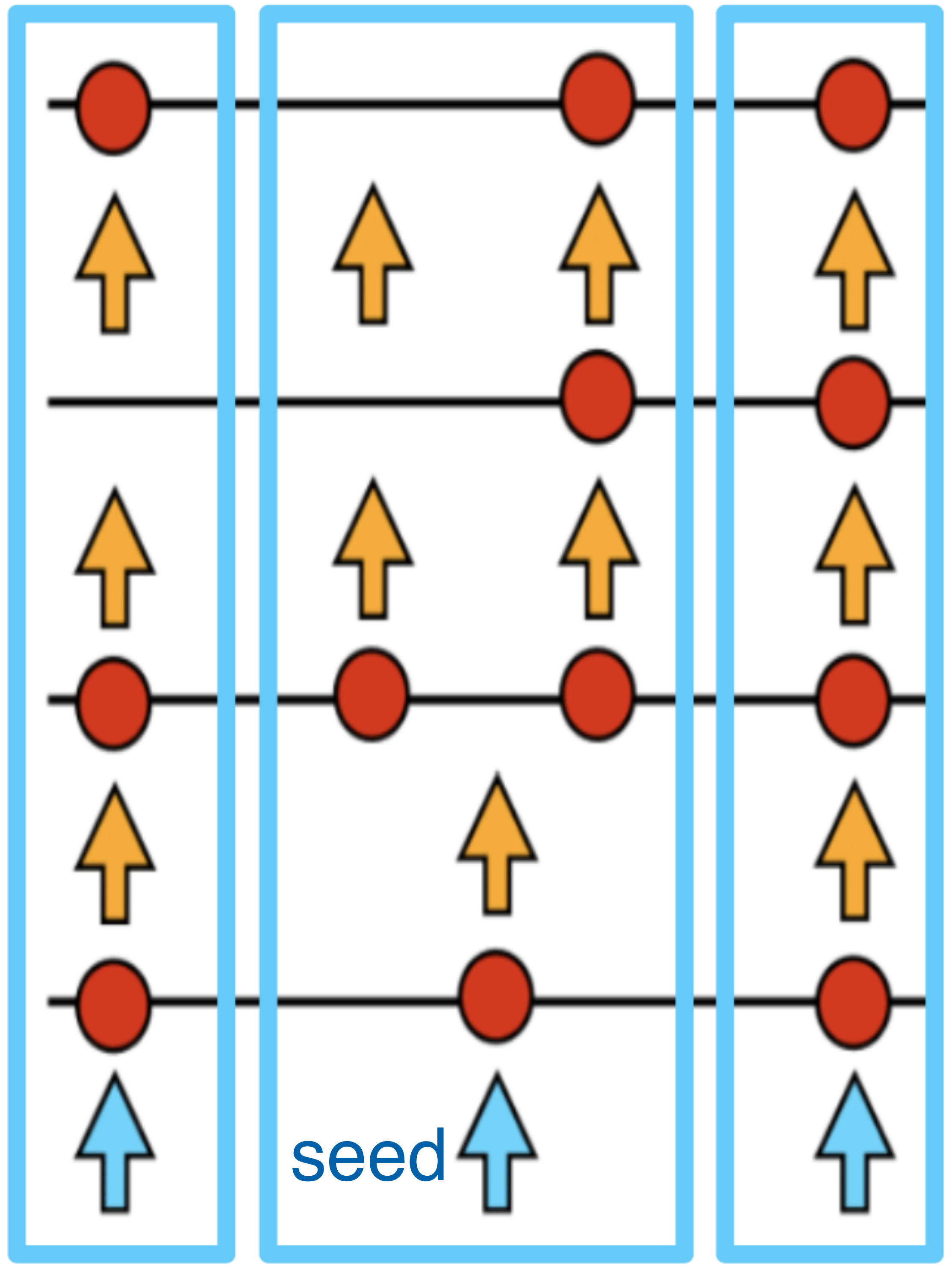
Kalman Filter

- Two step iterative process:
  - **Propagate** the track state from layer N-1 to layer N (prediction)
  - **Update** the state using the detector hit (measurement) on layer N

- Computing **challenges**:
  - Many operations with small matrices, low arithmetic intensity
  - O(2k) seeds and O(100k) hits/event @PU=70

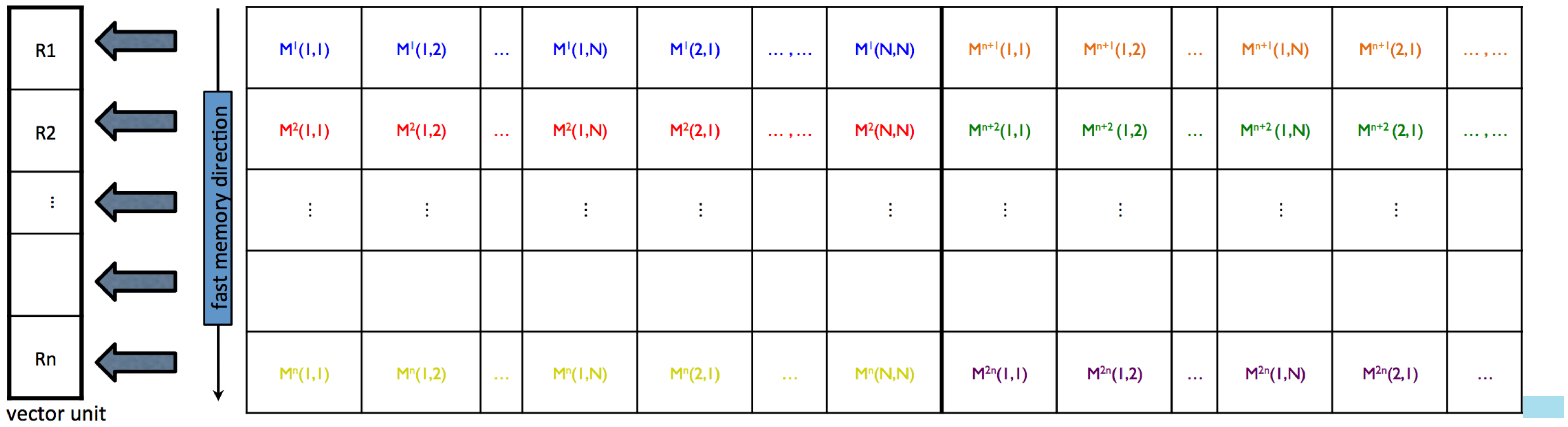
- KF track finding is not straightforward to parallelize
  - Combinatorial algorithm: **branching** to explore many candidates
  - **Heterogeneous** environment:
    different number of hits per track and tracks per event

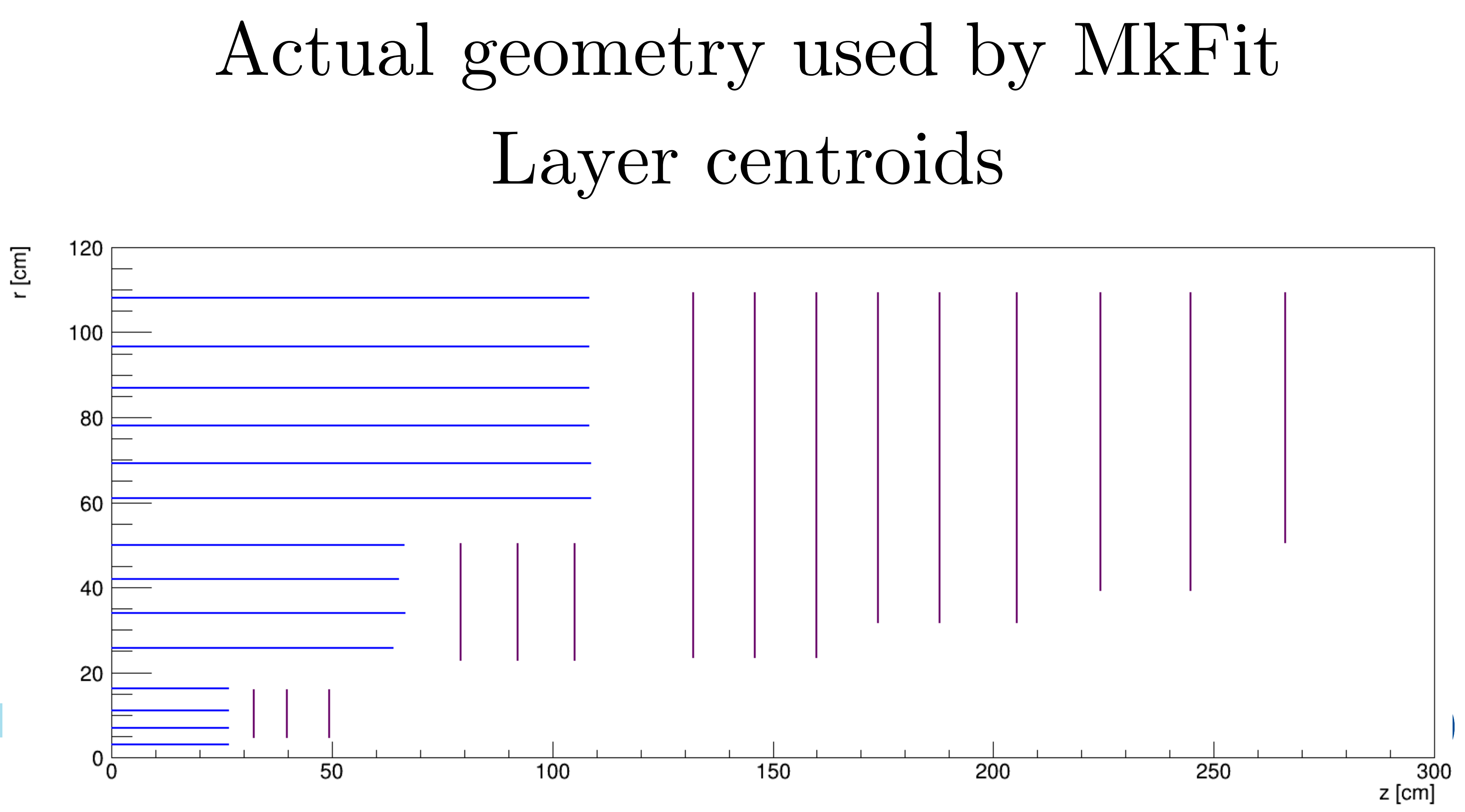




Key Features of the Algorithm

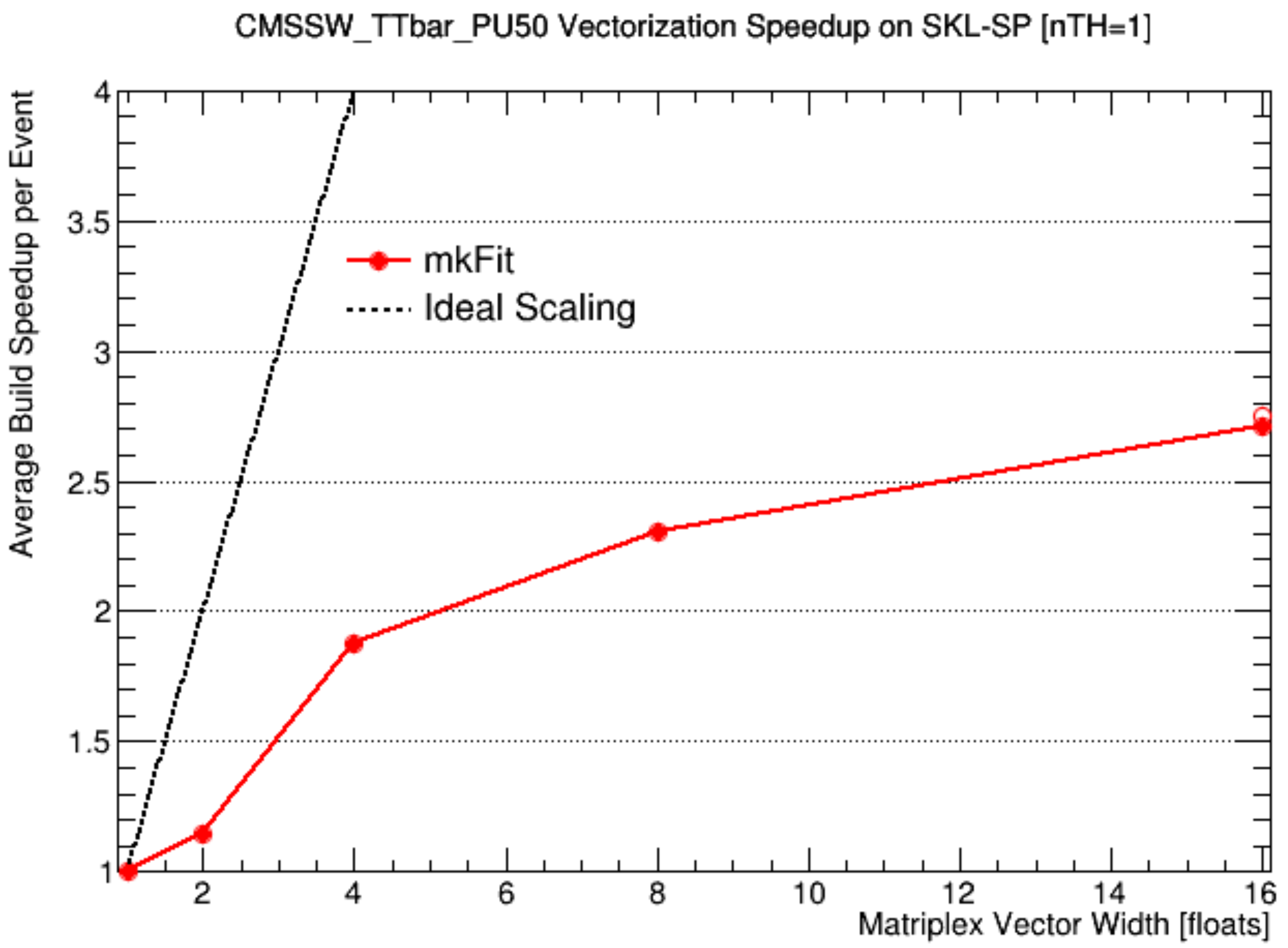
- Kalman filter operations use **Matriplex library**: SIMD processing of track candidates
  - auto-generated vectorized code is aware of matrix sparsity
- Algorithm multithreaded at multiple levels with **TBB tasks**
  - events, detector regions, bunches of seeds
- **Lightweight description** of detector in terms of geometry, material, magnetic field
  - collapse barrel (endcap) layers at average r (z), use 3D position of hits
- **Minimize memory operations** (number and size) within combinatorial branching
  - bookkeeping of explored candidates, clone only best ranking ones at each layer (with per seed cap)

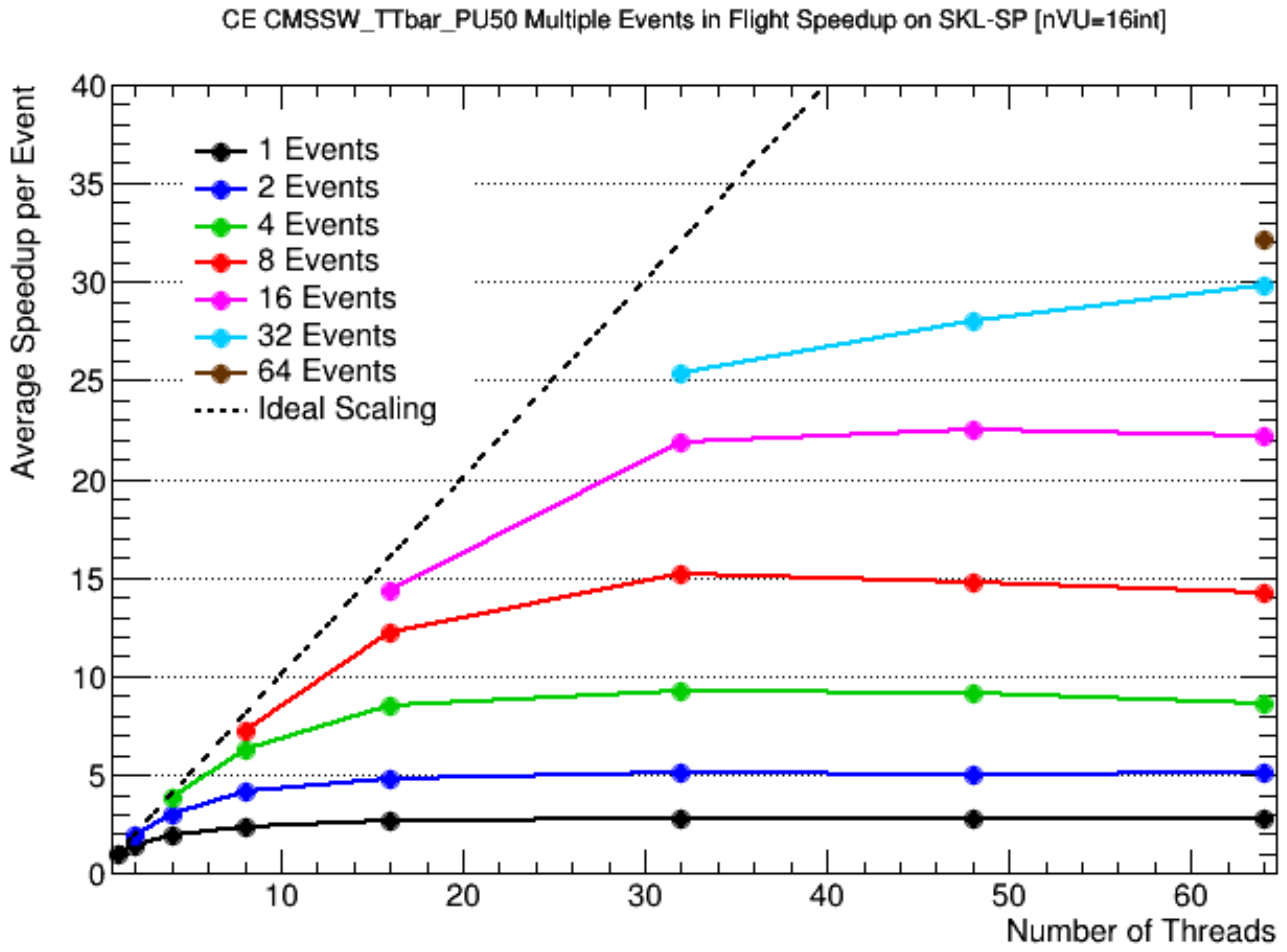




Timing Results for Standalone Application

- Showing results on Intel Skylake Gold processor (SKL)
- Core of algorithm achieves nearly **3x** speedup from **vectorization**
  - Ahmdal’s law: 60-70% of core algorithm code is effectively vectorized
- Full application achieves **30x** speedup with **multi-threading**
  - close to ideal scaling when all threads dedicated to different events

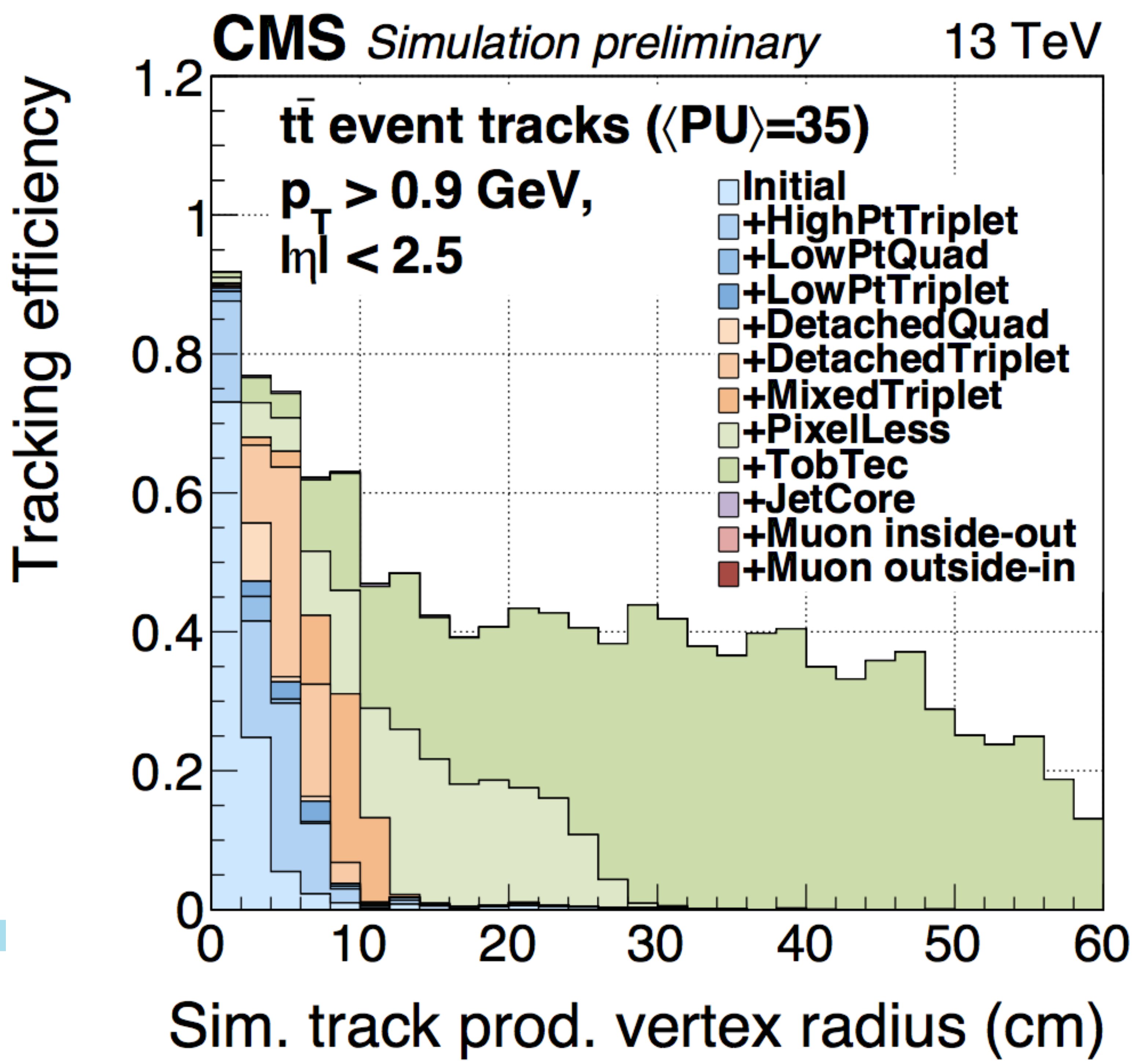




Deployment in CMS: CMSSW integration

- **Integration** in CMSSW has recently been the **main focus** of the group
  - Github repository made public, mkFit is now integrated into CMSSW as an external
- Two aspects are not ideal in the first integration:
  - When distributed in central CMSSW release, mkFit is compiled with gcc/core2
  - Dedicated steps are used to convert CMSSW data formats to/from mkFit

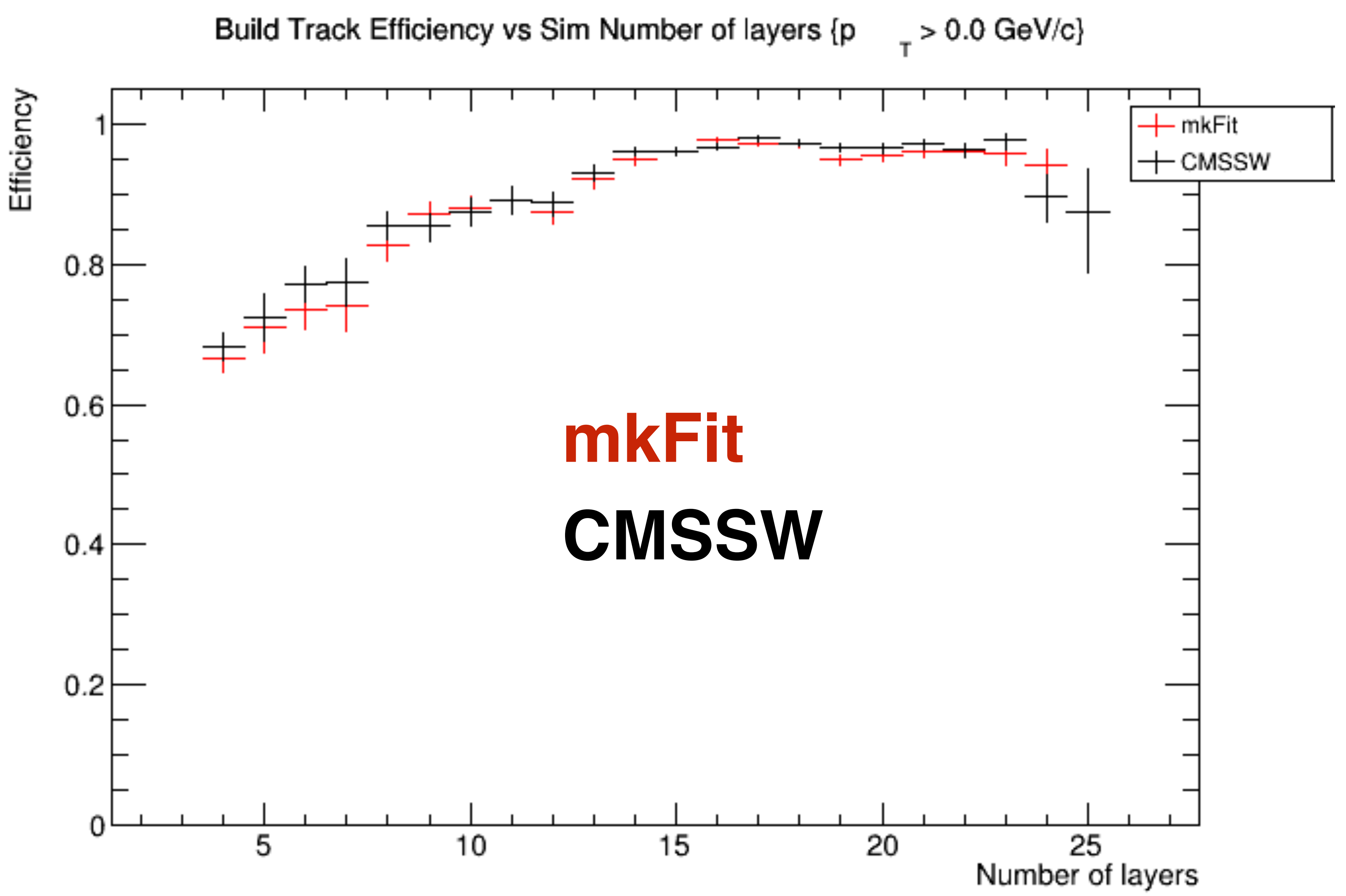
- CMS tracking structured in 10+ iterations
  - Seeding+building = combinatorial algorithms
  - Fitting+selection+masking = linear algorithms
- First milestone: track building for **initial iteration**
  - Seeds made of 4 hits, finds most prompt tracks
  - Could easily be extended to include other iterations

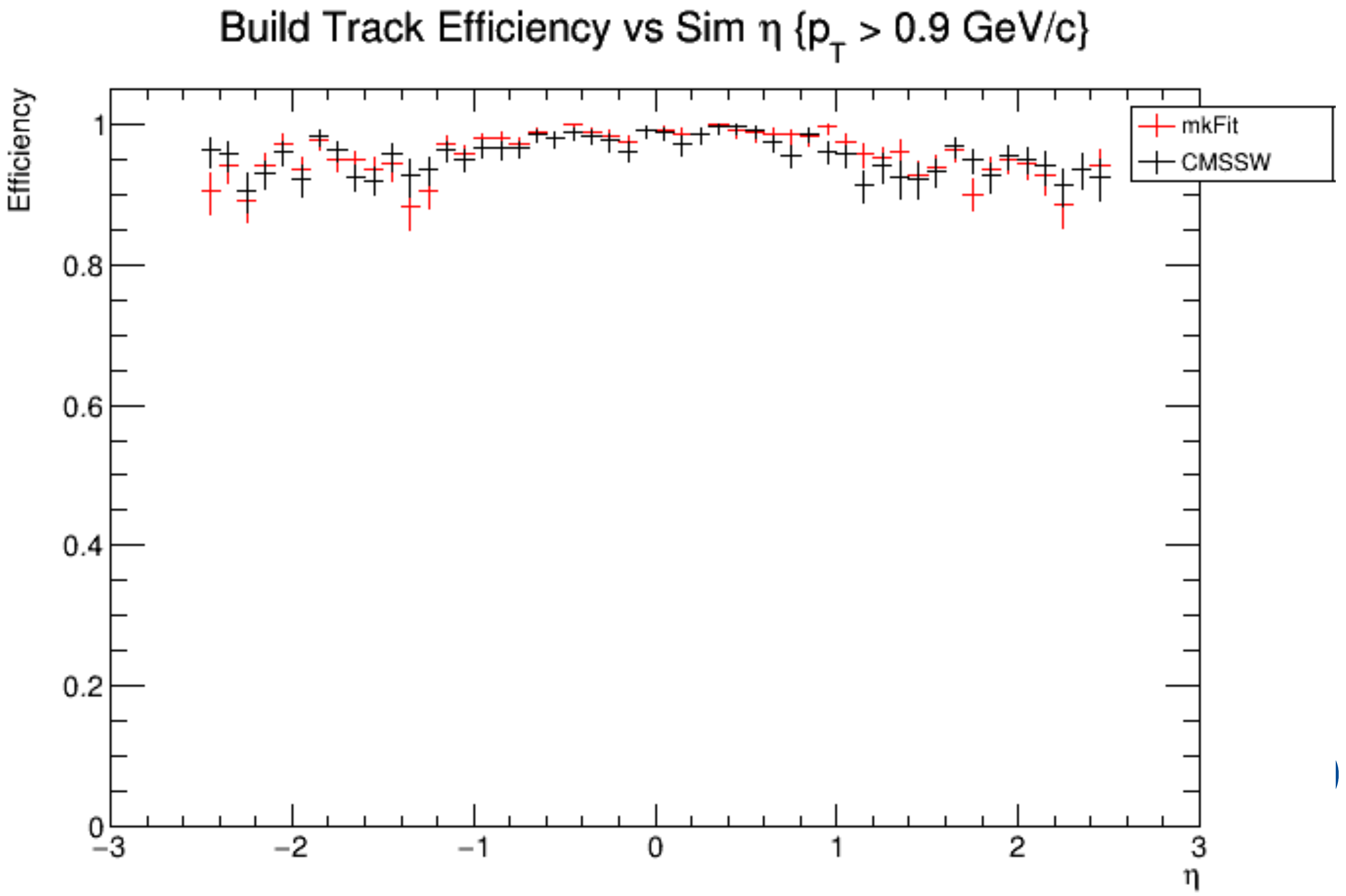


Physics Performance Improvements

- Integration in CMSSW gave access to central validation tools, which revealed a phase space where mkFit physics performance was suffering: **short tracks**
- Dedicated effort to **recover efficiency** at low number of crossed layers
  - Updated logic to count the number of missing hits in a track in a consistent way
  - Updated candidate score used to decide which is the best track candidate
- Efficiency now on par with CMSSW across the board
  - some more work needed to reduce fakes and duplicates, also need to recover overlap hits

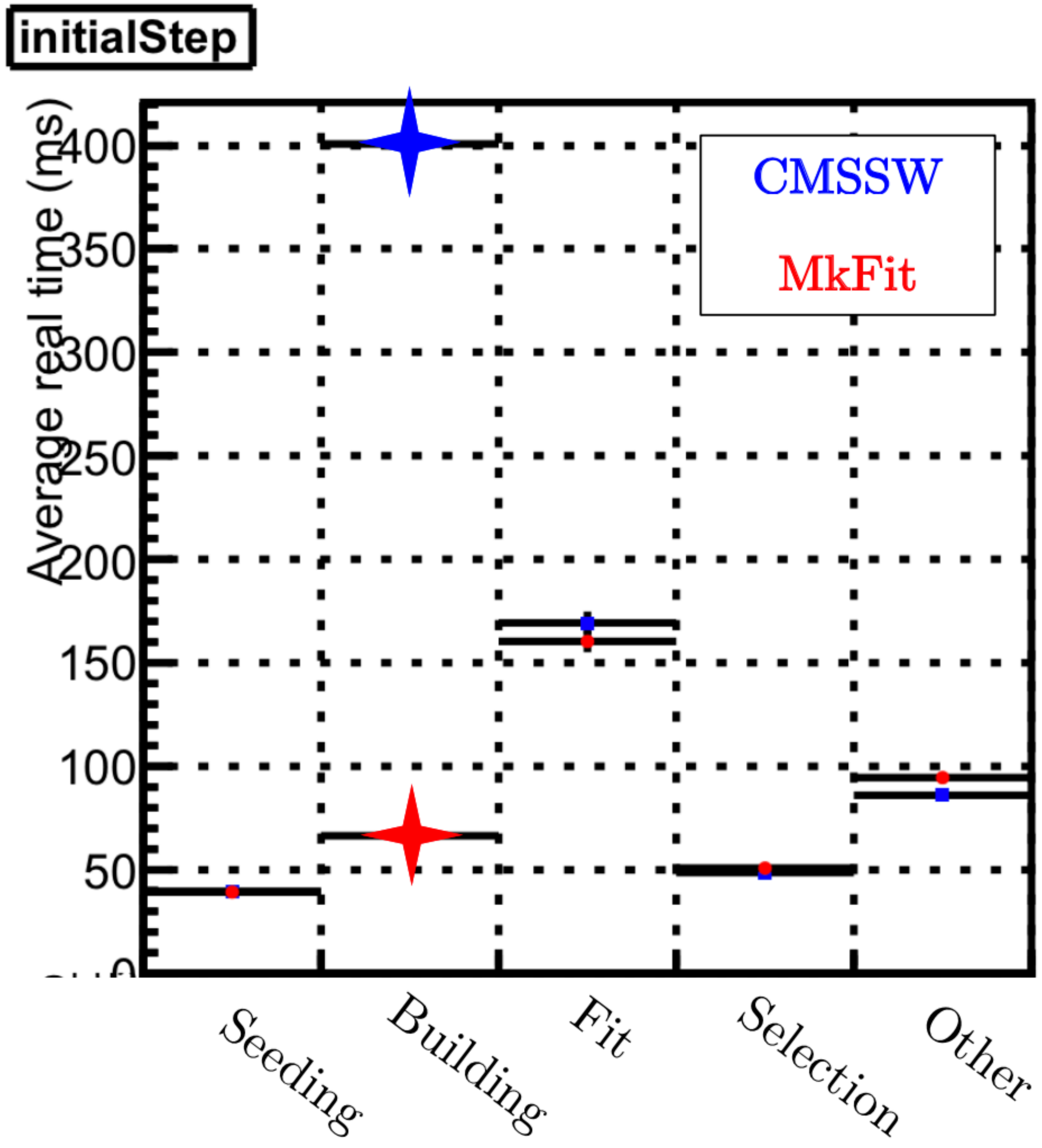
ttbar events, <PU>=50
Algorithmic efficiency: require Initial Iteration seed in denominator





Timing Performance of Initial Iteration

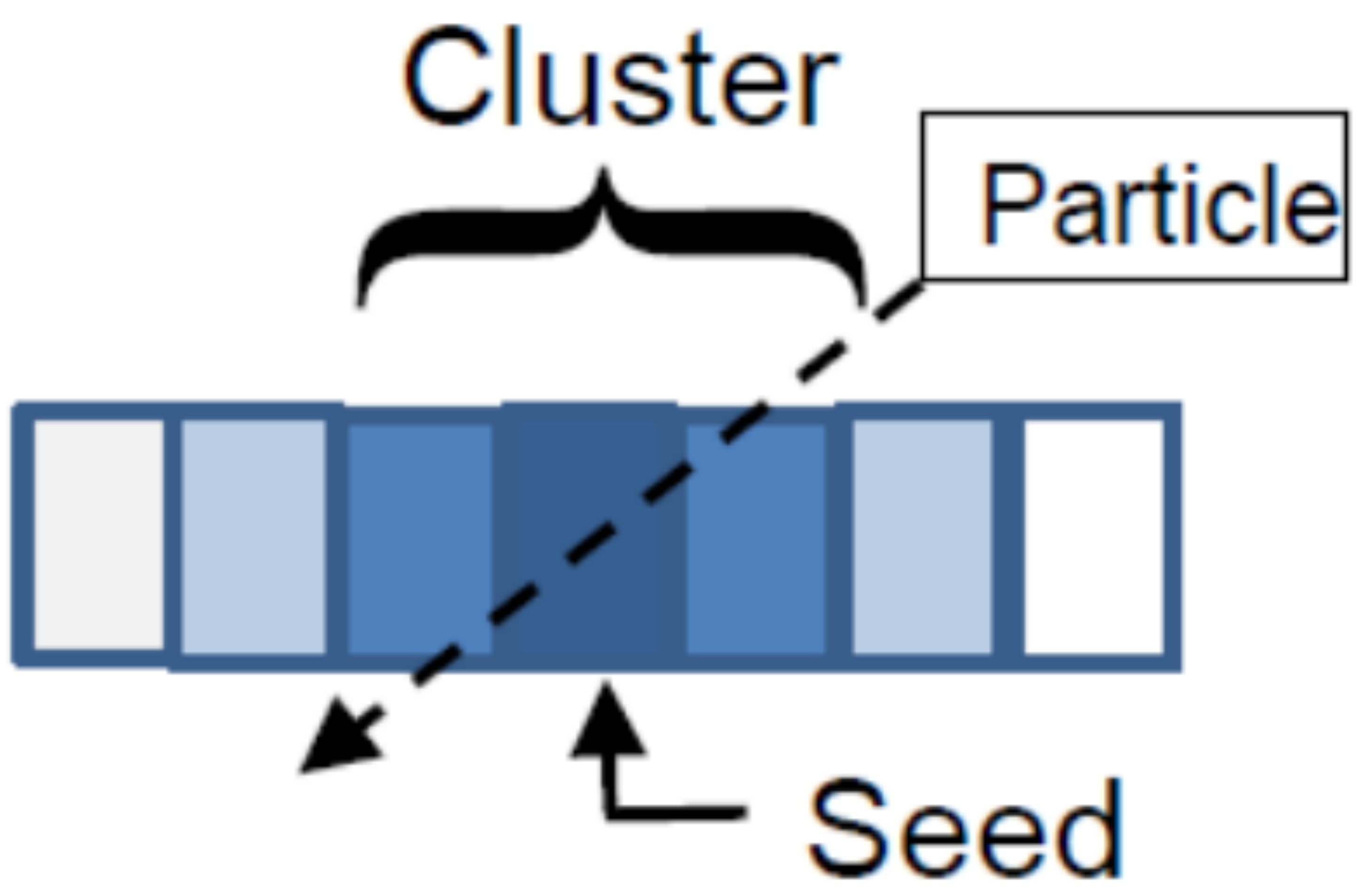
- **Single-thread performance** on Intel SKL
  - use ttbar events with <PU>=50
- Speedup of **6.2x** compared to CMSSW
  - track building is not the slowest component anymore!
- Data format **conversions** between CMSSW and mkFit account for **~25%** of mkFit time
  - larger speedup possible if data formats are harmonized
- Here mkFit is compiled with icc and AVX-512
  - with gcc speedup reduces to ~2.5x



Towards HLT Integration

- Work so far mostly focused on offline configuration
- However, HLT is the natural application environment for mkFit
- **HLT** configuration has different **challenges** with respect to offline
  - for many HLT paths, tracking is done in regions of interest
  - silicon strip local reconstruction is **on-demand** within the track pattern recognition
- mkFit aims at performing global tracking at HLT: read all hits as an input
- Global **strip reco** is currently costly, investigate **faster implementation**:
  - ideally start from raw and produce hits in the mkFit data format; compatible with GPU
- Current status:
  - raw data unpacking and remapping to DetIds: implementation in progress
  - strip data calibration: implementation in progress
  - strip data clustering: initial implementation made and begin tested

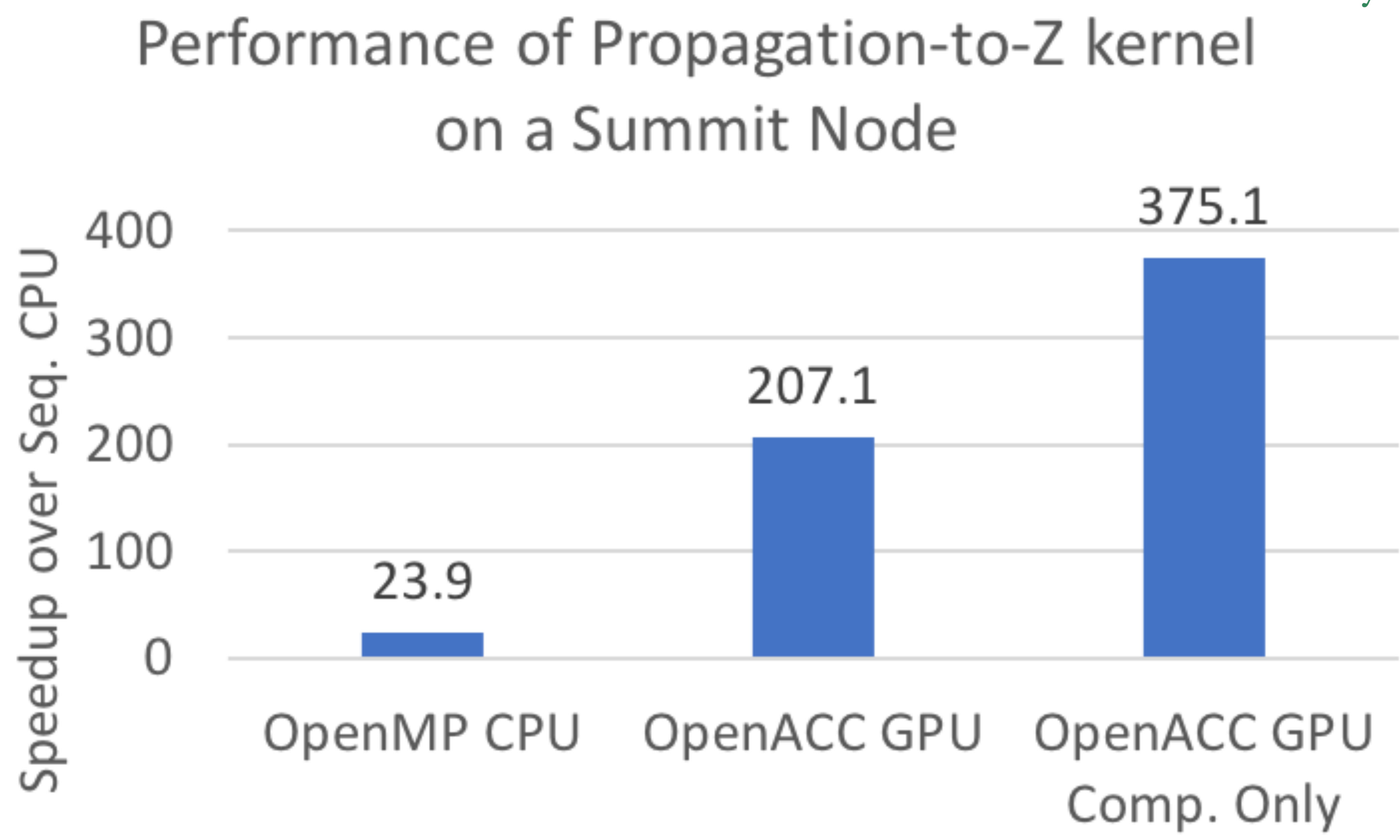
Strip Clustering Results on CPU and GPU



- Clustering Algorithm (current implementation):
  - Identify seeds: ≥1 strip must have ADC > 3x noise
  - Seek L/R boundaries:
    - (1) included strips must have ADCs > 2x noise
    - (2) Strips must be consecutive or have gap <= N strips (N depends on good/bad strips)
  - Final checks: quadrature sum of ADCs >= 5x quadrature sum of noise; total charge > min
- Standalone implementation on **CPU** (OpenMP for now) and **GPU** (CUDA)
  - initial version processes a single event
  - GPU version (P100) is ~3x faster than CPU (14-core Broadwell), including overhead
  - overheads currently include data transfer and memory allocation; actual kernel time 7% only
- Working on improved version that will **reduce overheads**
  - processing multiple events concurrently: asynchronous memory transfer
  - using memory pool to pay the allocation overhead only at begin and end of job

Exploration of Portable Implementations

- Exploration of GPU-compatible, **portable implementations** of track building
  - pros: maintainable, minimal diffs between CPU and GPU code
  - cons: may require trade-offs in terms of performance
- Started collaboration with **RAPIDS@ORNL** to explore usage of portable compiler directives
  - version of full application with **OpenMP** (CPU for now)
  - **OpenACC** in PropagationToZ function (out of ~100) from full code, get large speedups on GPU
  - challenges ahead: data transfer, CMSSW interface
- Other tests towards GPU-compatible code:
  - **array programming**: xtensor/numpy/cupy
  - plan to try portable **libraries** and revisit **CUDA** implementation



Conclusions

- mkFit code integrated in CMSSW as external library
- Physics performance (efficiency) on par with current tracking
- Speedup of >6x when compiled with icc and AVX-512
- Exploring utilization of GPUs at different stages
  - strip local reconstruction
  - portable implementation of algorithm
- Plans to publish a paper with detailed results soon - stay tuned!

Backup

mkFit: early GPU results

- Explore GPU-friendly data structures
- Matrix layout: Linear vs. Matriplex
  - For 6x6 matrix multiplications, the Matriplex layout (with large size) gives better performance than alternatives
  - Share same templated interface as CPU version, but implementation customized for GPU/CUDA
- Candidate cloning: avoid moving tracks in global memory
  - Parallelization implemented as one GPU thread per candidate
  - Select the best new candidates for each seed in shared memory
  - Process the list of new candidates with a heap-sort algorithm
- These developments were successful for track fitting while track building on K40 showed no significant speedups with respect to the CPU version
  - Including data transfers (taking about half of build time)
  - Building code was still in embryonal stage, missing important features like multiple events in flight (event: detector readout at beam crossing)