



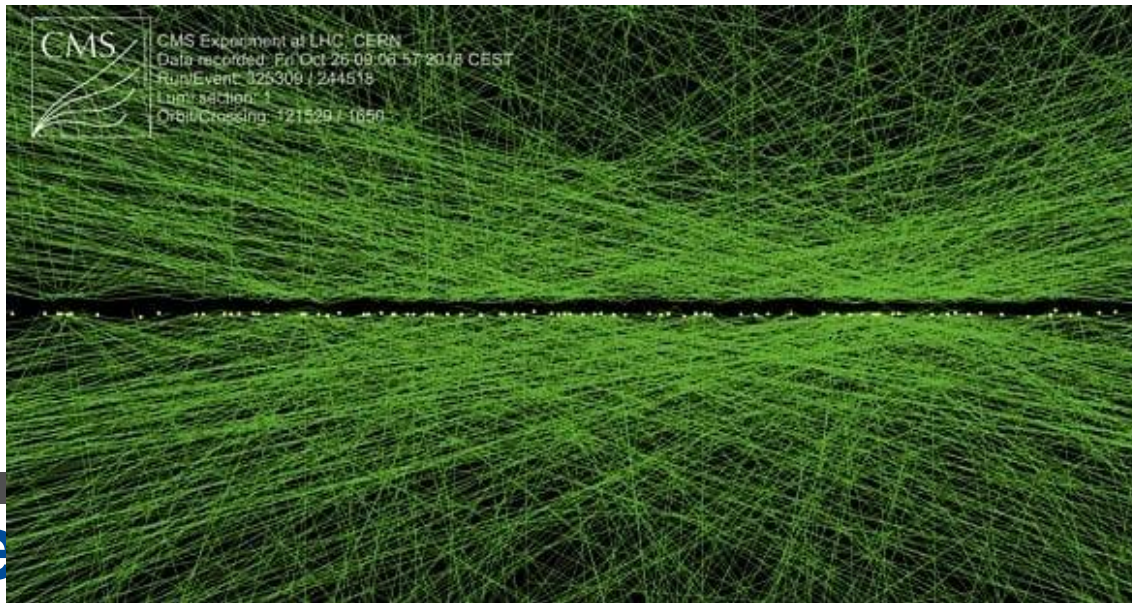
# mkFit Project: Speeding up particle track reconstruction using a vectorized and parallelized Kalman Filter algorithm

G. Cerati<sup>1</sup>, P. Elmer<sup>3</sup>, B. Gravelle<sup>5</sup>,  
M. Kortelainen<sup>1</sup>, S. Krutelyov<sup>4</sup>, S. Lantz<sup>2</sup>,  
M. Masciovecchio<sup>4</sup>, K. McDermott<sup>2</sup>, B. Norris<sup>5</sup>,  
**A. Reinsvold Hall**<sup>1</sup>, D. Riley<sup>2</sup>, M. Tadel<sup>4</sup>,  
P. Wittich<sup>2</sup>, F. Würthwein<sup>4</sup>, A. Yagil<sup>4</sup>

1. FNAL 2. Cornell 3. Princeton 4. UCSD 5. Oregon

# Setting the stage

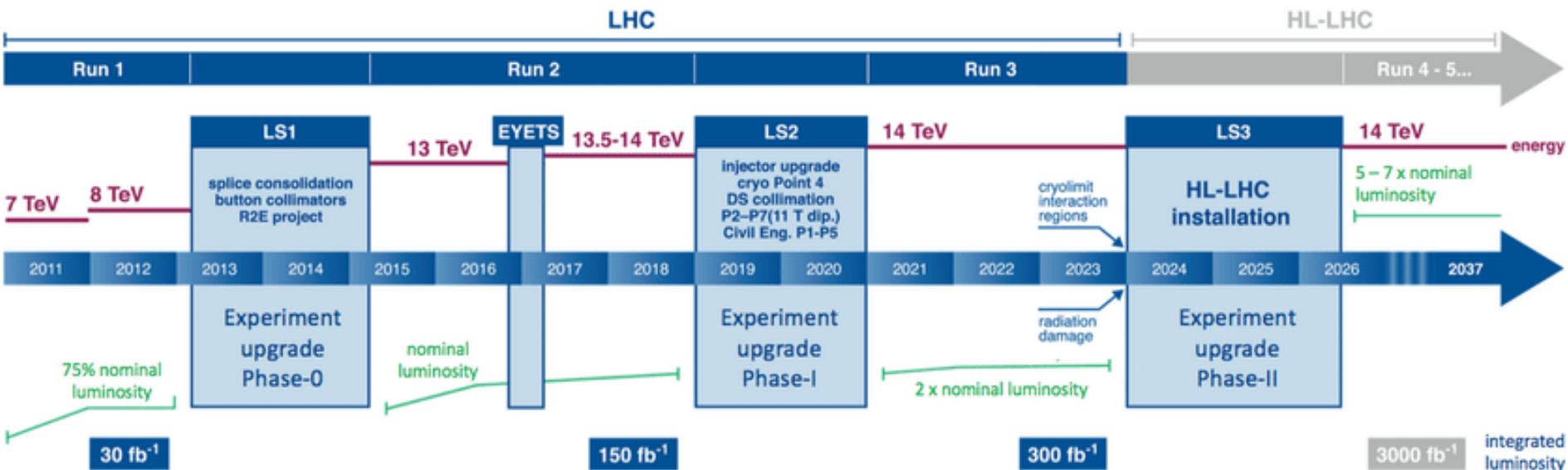
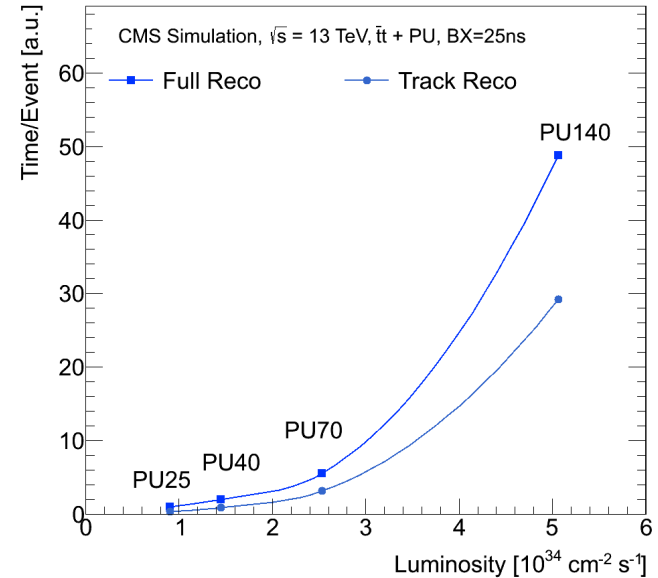
- Tracking is crucial for collider physics experiments
  - Charged particle momentum, particle flow, MET, b-tagging...
- Tracking is time-consuming
  - Uses approximately 50% of the time to reconstruct one event
- At the LHC, the problem is only getting worse as higher instantaneous luminosities are reached
  - Higher luminosity = more overlapping proton-proton collisions (pileup, PU) and increase in combinatorics to deal with



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

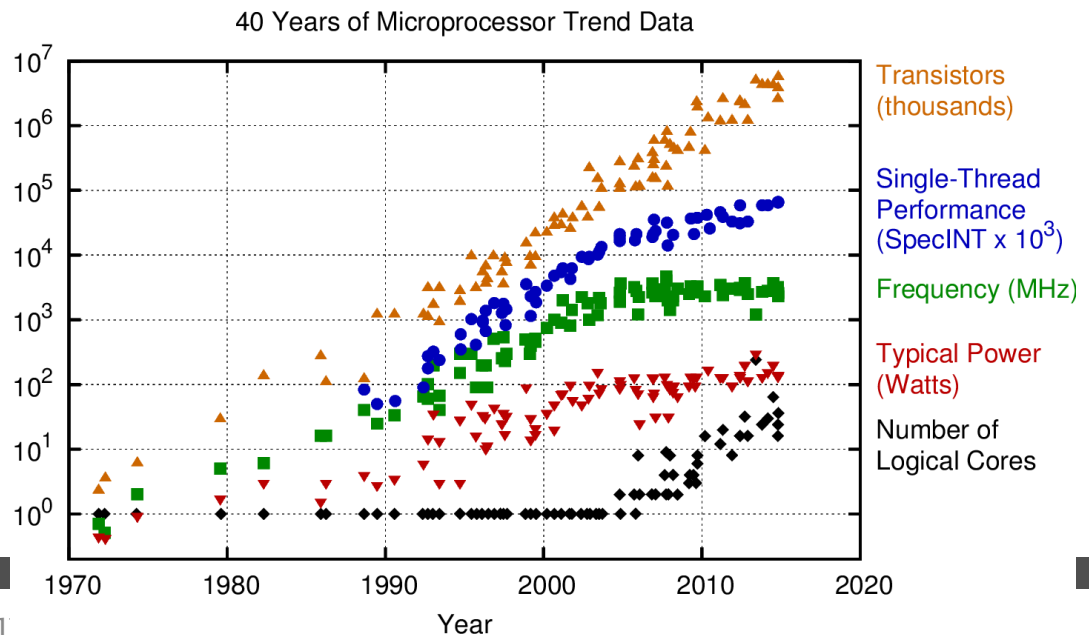
# LHC Schedule

- Time to reconstruct one event increases exponentially with pileup
- In the HL-LHC, the **average** PU will be 200 (instantaneous luminosity of  $5-7 \times 10^{34} \text{ cm}^{-2}\text{s}^{-1}$ )
- New solutions for tracking must be explored



# Hints at a solution

- Can no longer rely on **frequency** to keep growing exponentially
  - Nothing for free anymore
- Since 2005, most of the gains in **single-thread performance** come from SIMD or vector operations
  - Also starting to taper off
- Instead, **number of logical cores** continues to grow
- Need to rewrite our algorithms to take advantage of both parallelization and vectorization



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

# Parallelized Kalman Filter Project

Code name: mkFit

Matriplex Kalman Finder/Fitter

# Parallelized KF Tracking Project

- Ongoing project for 3+ years
- **Mission:** adapt traditional Kalman Filter (KF) tracking algorithms to maximize usage of **vector units** and **multicore** architectures
  - Testing on Intel Xeon and Intel Xeon Phi
  - Longer term: adapt algorithm for GPUs (not covered today)
- **Achievements shown today:**
  - Effective use of vectorization and multi-thread scaling
  - Physics performance comparable to offline CMS reconstruction
- **Aim:** Test algorithm online in Run 3 software-based High Level Trigger (HLT), extend to HL-LHC CMS geometry

See project website  
for details:

<http://trackreco.github.io/>

# SciDAC Project

- Approach can be applied outside of collider physics as well
- Fermilab and computer scientists at the University of Oregon are collaborating on a 3-year SciDAC project
- Two prongs:
  - Track reconstruction in collider experiments
  - Reconstruction for neutrino experiments using liquid argon time projection chamber detectors
- Goals for both cases:
  - Write faster code that takes advantage of parallel architectures
  - Eventually incorporate the algorithms back into the experiment's software frameworks

**See SciDAC project website for details:**  
<http://computing.fnal.gov/hepreco-scidac4/>

# Using the Kalman Filter

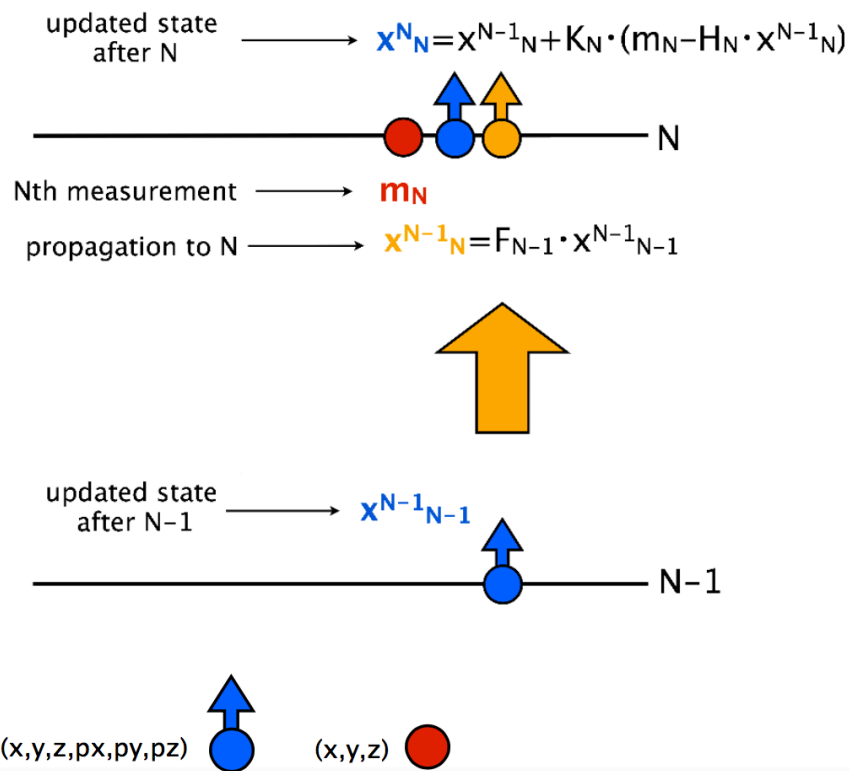
**Benefits** of the Kalman Filter for track finding/fitting:

- Robust handling of multiple scattering, energy loss, and other material effects
- Widely used in HEP
- Demonstrated physics performance

Two step process:

1. **Propagate** the track state from layer N-1 to layer N (prediction)
2. **Update** the state using the detector hit (measurement) on layer N

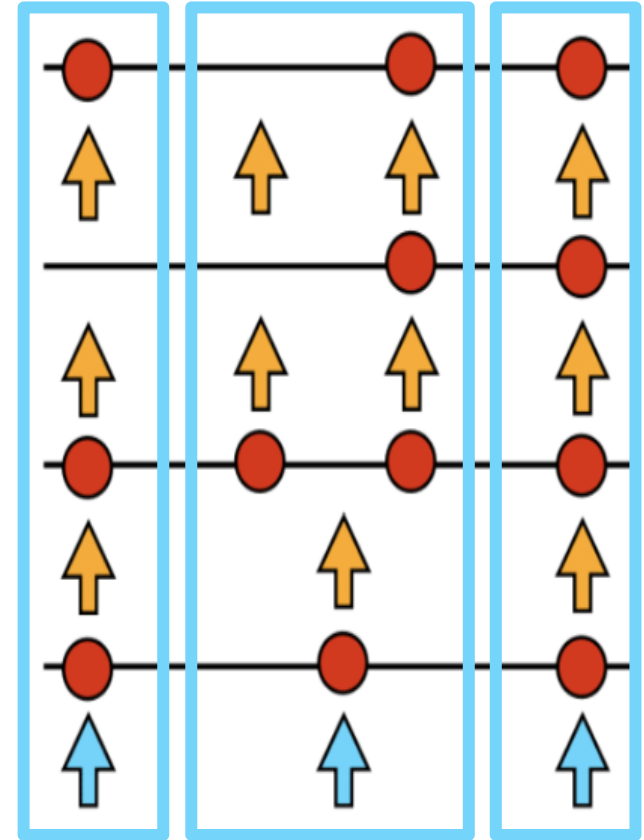
Predicted track state  
Detector measurement  
Updated track state





# Track building in a nutshell

- Start with a **seed track**
- **Propagate** track state to the next detector layer
- Find detector hits near projected intersection of track with layer
  - Problem of **combinatorics**: could find 0 hits, 1 hit, or several hits
- Select best fit track-hit combinations as track candidates
  - Update estimated state of all track candidates with new hit
  - At each layer, the number of possible track candidates per seed increases
- **Iterate**

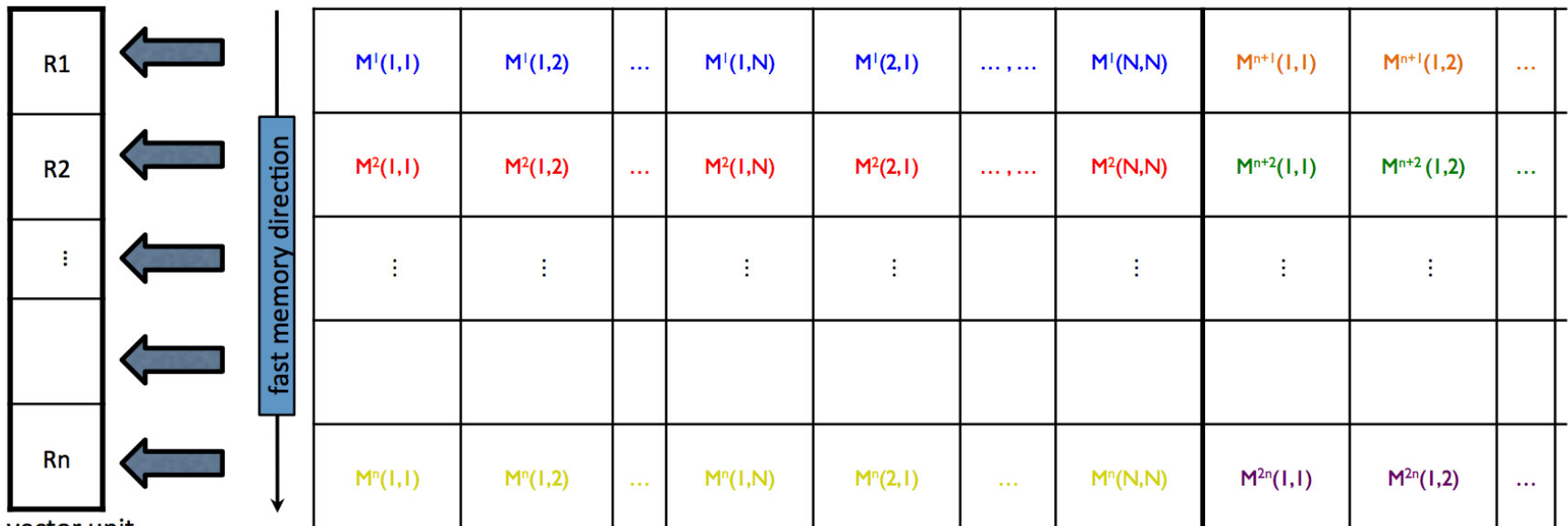


# Parallelization and Vectorization

- **Task** scheduling is handled via TBB library from Intel
- **Parallelization** at multiple levels
  - **parallel for:** N events in flight
    - **parallel for:** 5 regions in  $\eta$  in each event
      - **parallel for:** seed-driven batching, 16 or 32 seeds per batch
- **Vectorized** processing of individual track candidates where possible
  - Using both compiler vectorization and the Matriplex library

# Matrplex Library

- Custom library for vectorization of small matrix operations
- “Matrix-major” representation designed to fill a vector unit with  $n$  small matrices and operate on each matrix in sync
- Includes code generator to generate C++ code or intrinsics for matrix multiplication of a given dimension
  - Can be told about known 0 and 1 elements matrices to reduce number of operations by up to 40%
- Used for all Kalman filter related operations



Matrix size  $N \times N$ , vector unit size  $n$

# Integrating mkFit into CMS

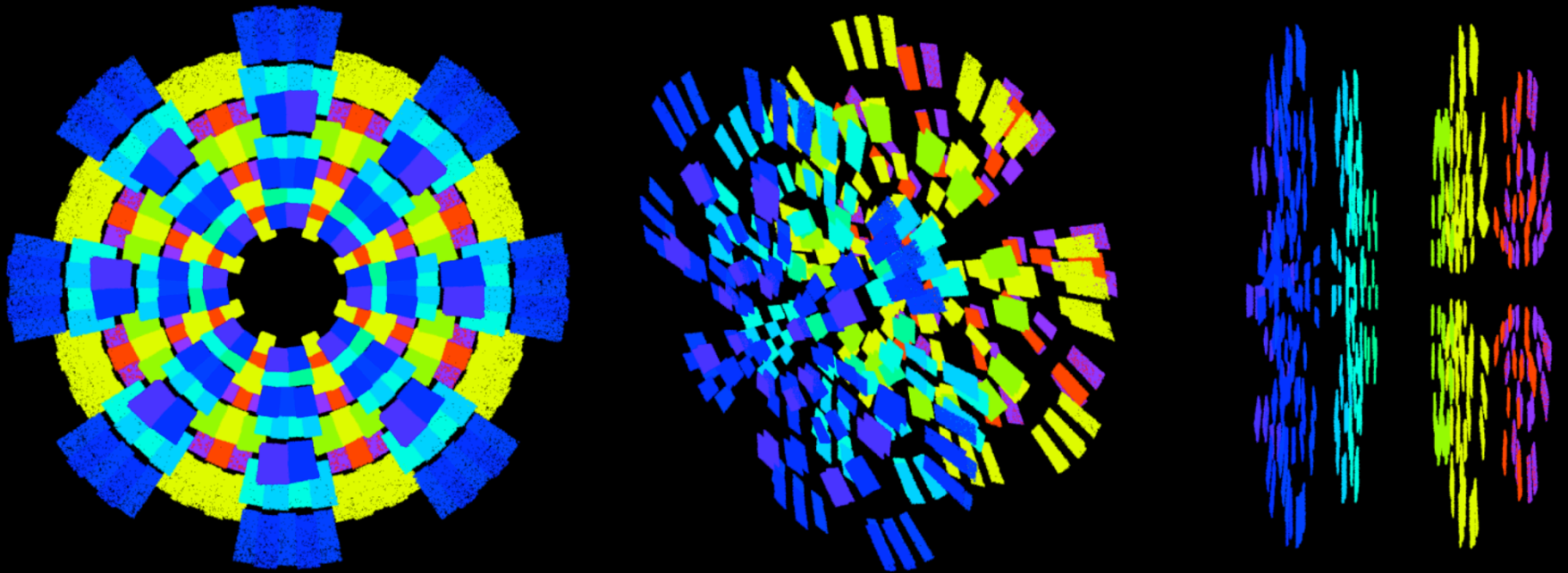
# Integration with CMSSW

- Matriplex library and overall strategy is not specific to a particular experiment
- Goal is to be useful in the CMS HLT
  - Test algorithm online in Run 3
  - Fully deploy algorithm in the HL-LHC
- Compare physics and compute performance against nominal CMS reconstruction software CMSSW
- Target architecture is the Intel Xeon Skylake (SKL)
- Focus is on **track building**
  - Use seeds from CMSSW or MC truth seeds
  - Continue to use CMSSW for the final fit because it has the most precise set of tools

# CMS Tracker

- CMS tracker uses silicon pixels and silicon strips
- Pixel region (closest to the beam pipe) used for seeding

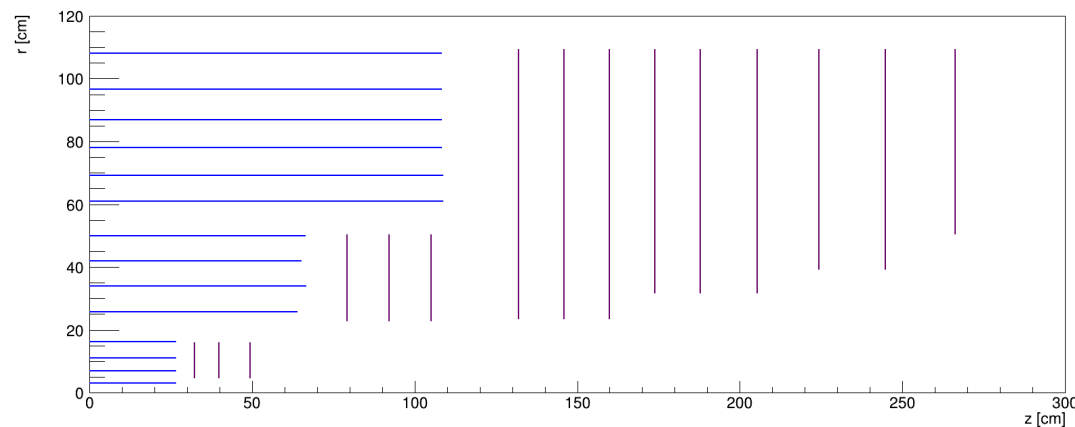
Example of a CMS endcap disk



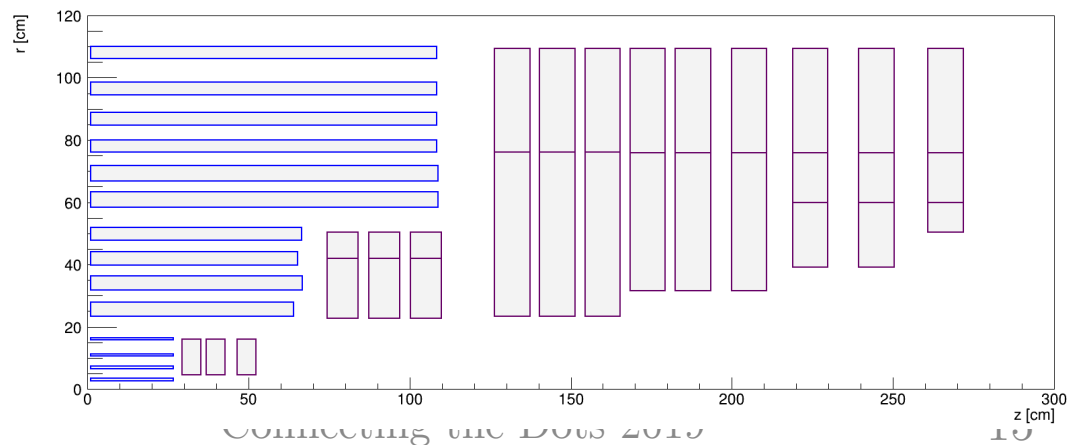
# CMS-2017 Geometry in mkFit

- Unlike CMSSW, **choose not** to deal with detector modules, only layers
  - Makes algorithm faster and more lightweight
- Geometry implemented as a plugin: core algorithm is **entirely separate** from detector geometry
- Track propagation to center of layer, then hit selection
- In overlap regions, only pick the single best hit

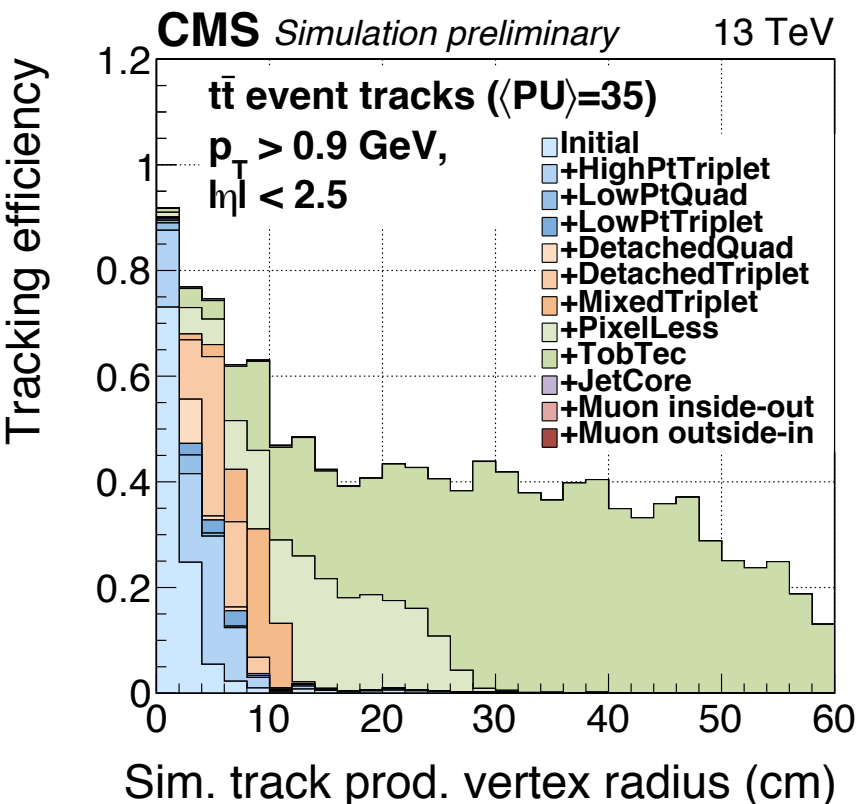
Actual geometry used by MkFit  
Layer centroids



Layer size



# CMS Iterative Tracking



- To reduce combinatorics, CMS performs track finding over several iterations
  - Start with tracks that are easiest to find, end with the most difficult tracks
  - Between each iteration remove hits that have been associated to a track
- mkFit focuses on initial iteration:
  - Seed tracks with 4 hits and no beam-spot constraint
  - Find most prompt tracks
- Could easily be extended to include other iterations



# Runtime Options

MkFit algorithm can be used in multiple setups:

## 1. Standalone code

- Input: simple data format created by doing a memory dump of data structures (hits, seeds, sim and reco tracks)
- Useful for development and validation of compute performance

## 2. Integrated into offline CMSSW

- Input: data are pulled from CMSSW data structures and formatted into mkFit data structures.
- After building, mkFit tracks are reformatted into CMSSW track candidates
- MkFit is compiled separately and used as an external package

## 3. Integrated into online (HLT) CMSSW

- Work in progress

# Runtime Options

MkFit algorithm can be used in multiple setups:

## 1. Standalone code

- Input: simple data format created by doing a memory dump of data structures (hits, seeds, sim and reco tracks)
- Useful for development and validation of compute performance

## 2. Integrated into offline CMSSW

- Input: data are pulled from CMSSW, converted to a format and formatted into a simple data format
- After building the track candidates, the track candidates are converted to a format CMSSW can use
- MkFit is compiled separately and used as an external package

**Data structure conversions are costly – need to find ways to mitigate this between algorithms**

## 3. Integrated into online (HLT) CMSSW

- Work in progress

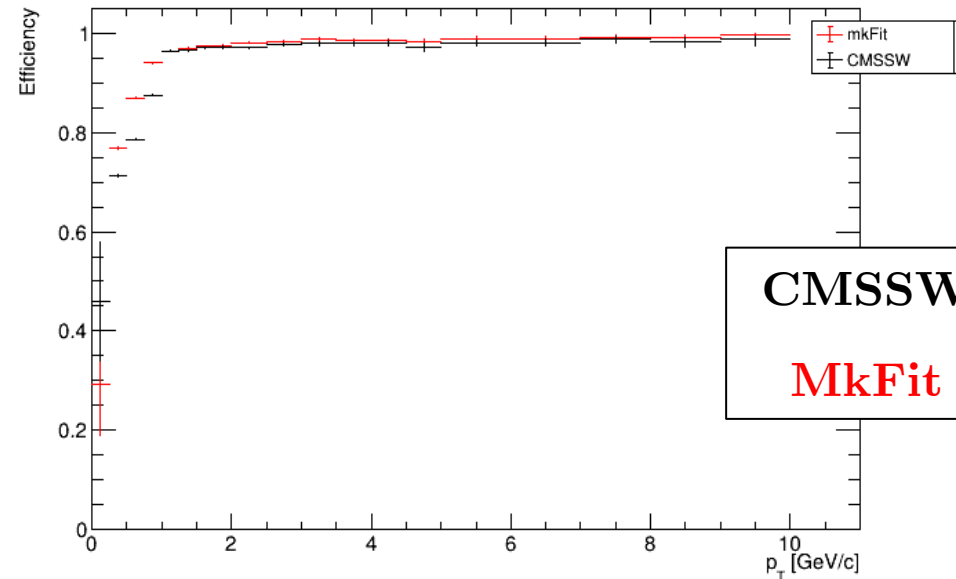


# Physics and Compute Performance

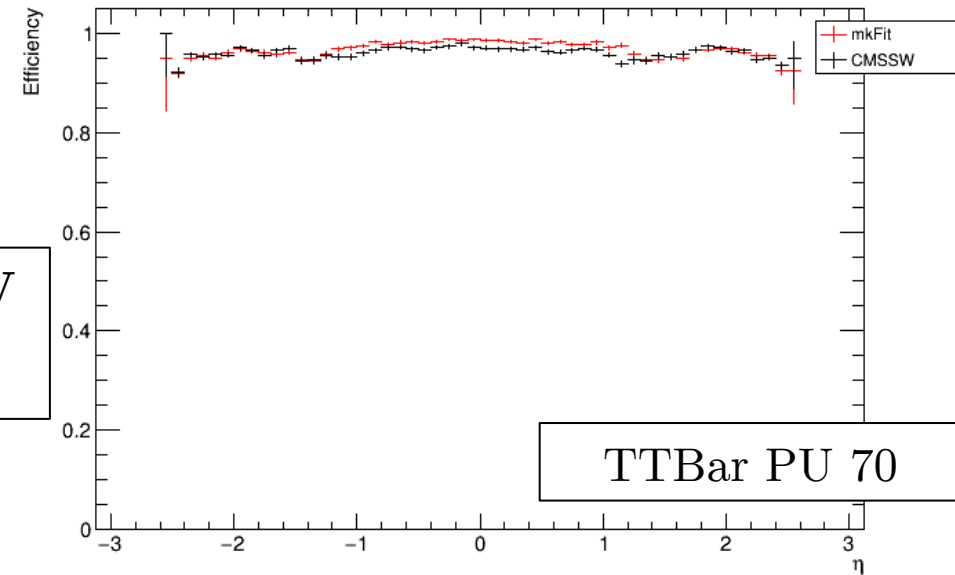
# Efficiency of mkFit

- Shown here: algorithm-level efficiency for long ( $\geq 12$  hit) tracks
- **mkFit** is at least as efficient as **CMSSW**, even for low  $p_T$  tracks
  - Crucial for accurate particle flow reconstruction
- Much of the effort in the last year has focused on achieving this important milestone
- Next steps: improve efficiency for short tracks. Development for this is already in progress

Build Track Efficiency vs Sim  $p_T$  ( $p_T > 0.0$  GeV/c)

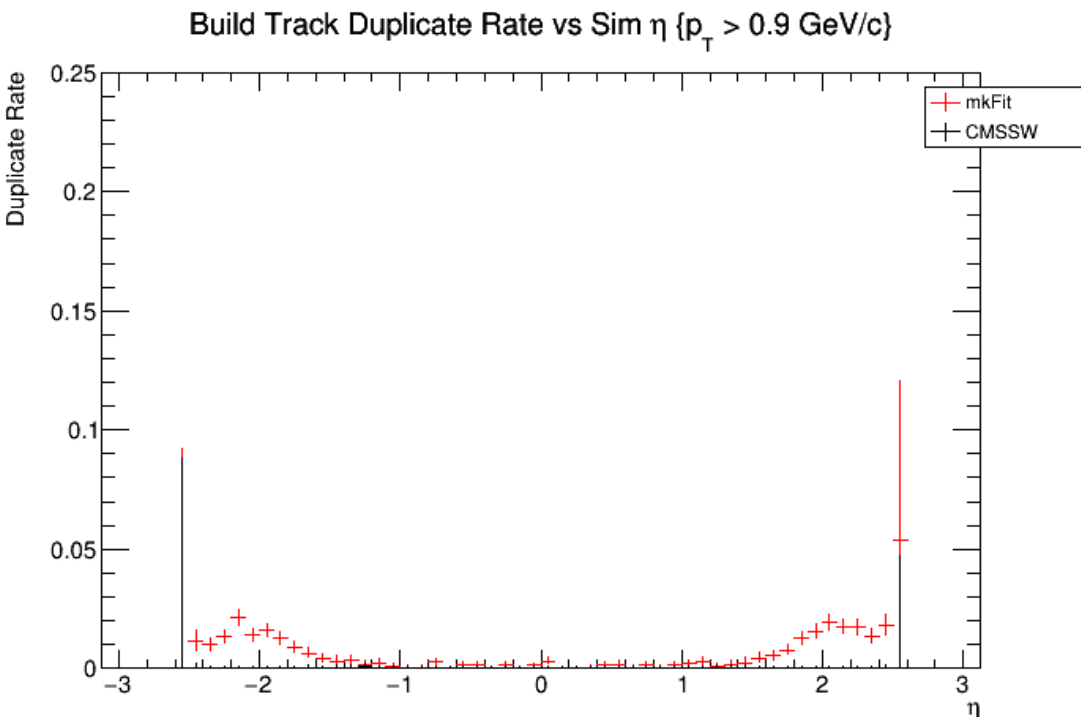


Build Track Efficiency vs Sim  $\eta$  ( $p_T > 0.9$  GeV/c)



# Duplicate Removal

- In CMSSW, tracks are built sequentially
  - Skip seeds that have already been included in a track candidate
- In mkFit, two step process to minimize duplicates:
  - Dedicated seed cleaning before track finding
  - Dedicated duplicate removal of track candidates after track finding

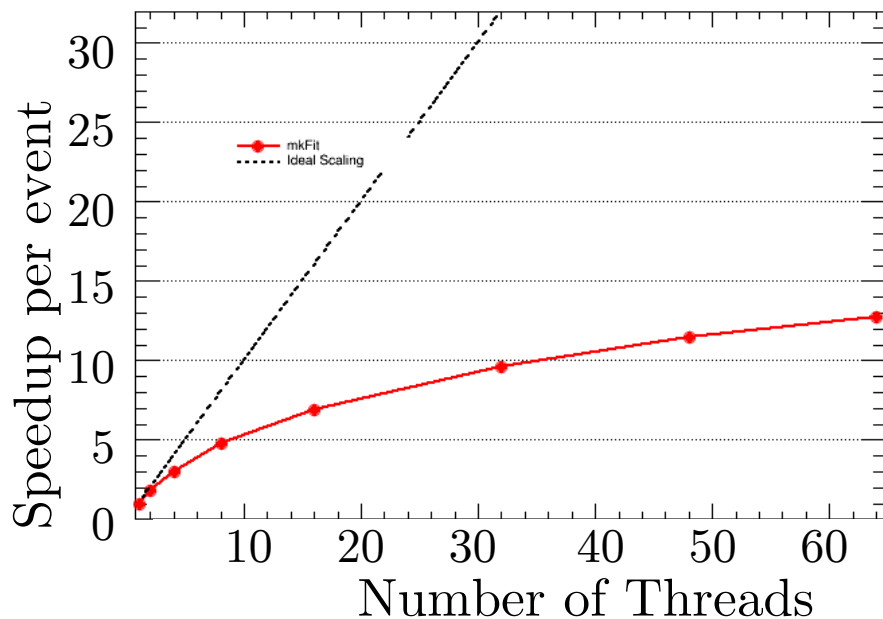


- Second step not optimized, still able to reduce rate to  $< 2\%$
- Similar dedicated cleaning will be used to lower fake rate; not implemented yet

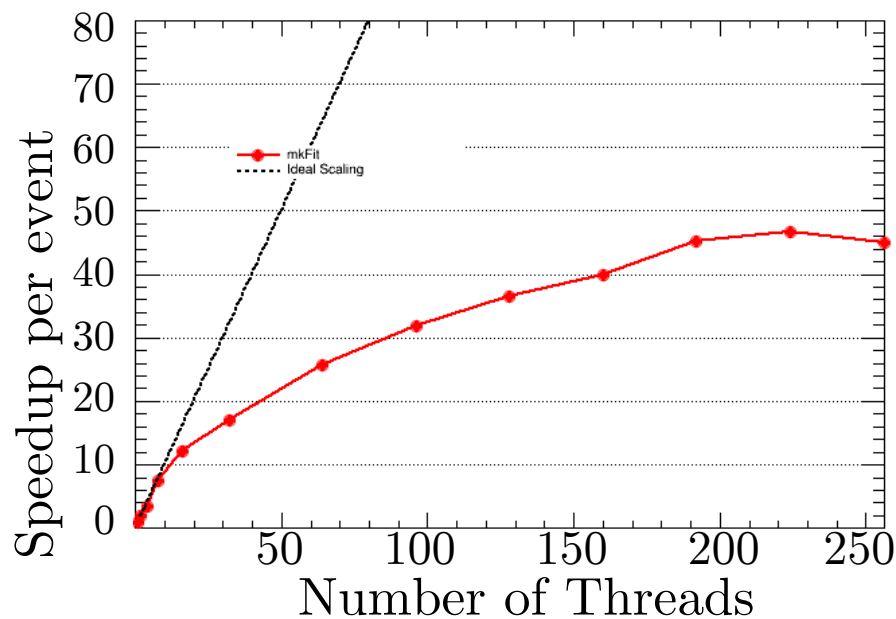
# Speedup vs # of Threads

Excellent scaling at low threads –  
**independent of exact architecture**

Intel Xeon  
Skylake SKL (Gold 6130)



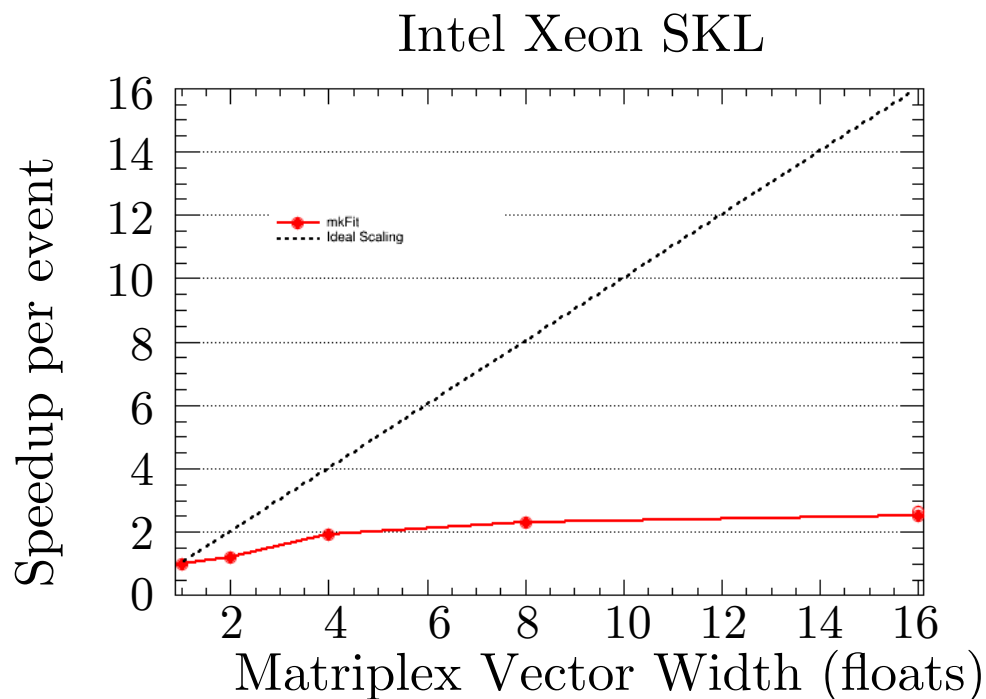
Intel Xeon Phi  
Knight's Landing KNL (7210)



- Results for track building only; does not include overhead
- Measured using standalone configuration, single event in flight
- Turbo boost disabled

# Speedup vs size of vector units

Algorithm uses vectorization successfully-  
**60 - 70% of code is vectorized!**

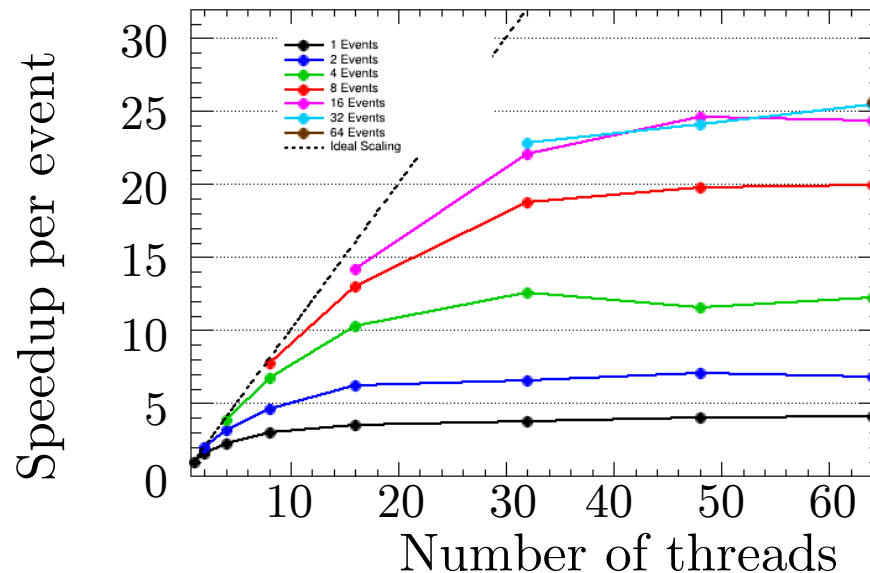


- Results for track building only; does not include overhead
- Measured using standalone configuration, single event in flight

# Speedup vs # of events in flight

Can get speedups up to x25 using multiple events in flight

Intel Xeon SKL



- Results include time for full loop, including I/O, handling the seeds, etc
- Measured using standalone configuration
- Previous plots used only a single event in flight



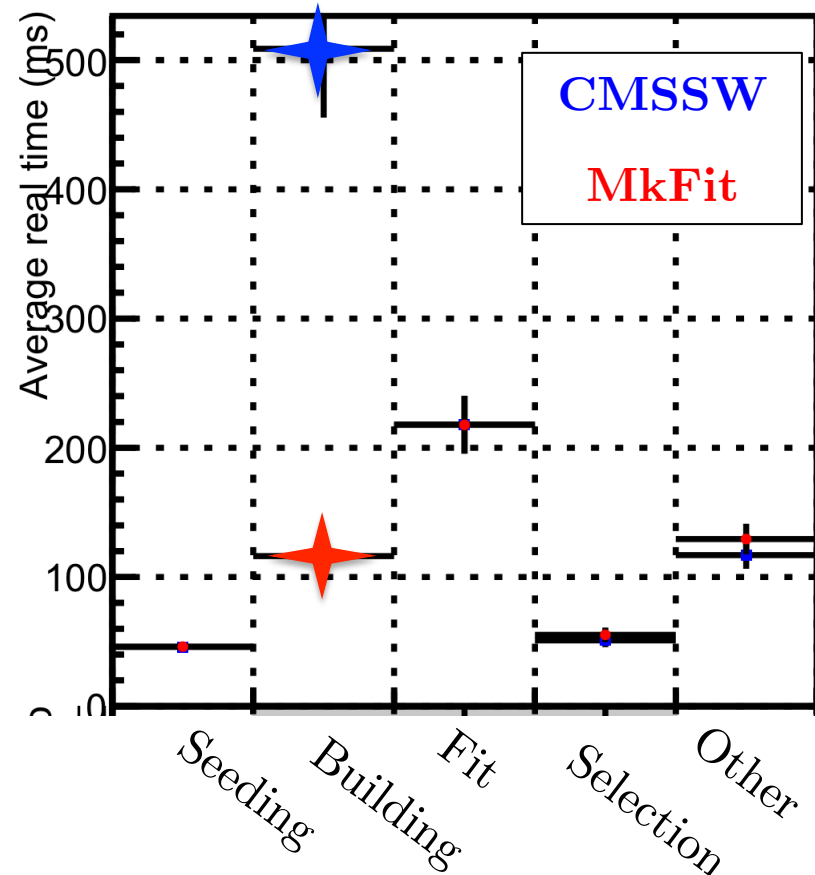
# Integrated Timing Performance

## Technical Details

- Run mkFit within CMSSW
- mkFit used for building only
- Single-thread test using TTBar PU 50

## Results

- Track building is **4.3x faster**
- 40% of time is spent in data format conversions – actual track finding is **7x faster**
- Track building now takes **less time than track fitting**
- Even larger speedup if multiple threads are used



\* Measured on SKL, mkFit compiled with AVX-512

# Conclusions

- Status of parallelized KF tracking (aka mkFit) is well advanced
- Physics performance **comparable to CMSSW**
- On a single thread, our core algorithm is **seven times faster** than offline CMSSW reconstruction (ignoring data conversions)
- Track building is now **faster** than track fitting
- Plan is to develop a GPU implementation as well
- Next steps: test algorithm in HLT environment



# Backup

# Track Finding Algorithms

- Standard algorithm
  - If a hit matches, then the candidate track is cloned and the hit is added to the track. After looping over all of the hits, the set of candidates is sorted and the best  $N$  candidates are kept.
- Best hit
  - No branching allowed. Select only the best hit from each layer.
- Clone engine
  - Same as standard, but the amount of copying is reduced by adding hits and track metrics to a bookkeeping list instead of cloning the candidate. After looping over all of the hits, the list is sorted and only the best  $N$  candidates are cloned and kept.
  - Expect to have the same output as the standard algorithm

# Key Differences wrt CMSSW

	CMSSW	MkFit
Seed Cleaning	Build tracks sequentially and reject seeds that have already been included in a track candidate	Everything is done in parallel. Apply seed cleaning before trying to build any tracks. Remove duplicates after track building
Hit Position	Reevaluate the hit position using the track direction	Hit position is taken from local reconstruction and not updated
Inactive modules	Able to access the detector status database to make sure modules were active	Cannot access DB so no knowledge of inactive modules
Geometry	Retains information about the detailed CMS geometry	Knows only about layers, not detector modules
Magnetic Field	Parameterized magnetic field	Currently using flat field. Will eventually use parameterized field
Module Overlaps	Can pick up multiple hits while track building	MkFit can only pick up one hit. We could pick up overlap hits during backward fit. Not implemented yet.

# Figures of Merit

- Different validation suites used for the two runtime options. Different choices and definitions made in order to achieve different goals (details on next slide)
- **mkFit Validation: algorithm-level** efficiency.
  - Used for standalone configuration
  - Goal is to make sure our algorithm is as efficient as CMSSW for long ( $\geq 10$  hits) tracks. Serves as a starting point to evaluate mkFit's performance.
- **Multi-track Validation (MTV): absolute** efficiency.
  - Used for mkFit integrated into CMSSW
  - Goal is to see the absolute performance of the tracking algorithm. Includes seed building efficiency.

**Efficiency:** fraction of reference tracks matched to a reco track

# Validation Definitions

	mkFit validation	MTV
Reference tracks	<ul style="list-style-type: none"><li>• SIM or CMSSW tracks with <math>\geq 12</math> layers (including 4 seed layers)</li><li>• SIM tracks must be matched to a seed</li></ul>	SIM tracks satisfying <ul style="list-style-type: none"><li>• <math>p_T &gt; 0.9</math> GeV</li><li>• <math> \eta  &lt; 2.5</math></li><li>• <math> dxy  &lt; 3.5</math> cm</li></ul> No seed matching requirement
<b>To-be-validated reconstructed tracks</b>	<ul style="list-style-type: none"><li>• Reco. tracks with <math>\geq 10</math> hits</li><li>• For mkFit tracks, 4 of the hits are required from the seed</li></ul>	No additional requirements
Matching criteria between ref. and reco. tracks	Considered matched if $\geq 50\%$ of the hits are shared, excluding the seed	Considered matched if $> 75\%$ of the clusters of the reco track contain charge induced by the reference track

# Architectures

- Intel Xeon Phi Knight's Landing KNL (7210)
  - 64 physical cores, 256 logical cores, 1.3 GHz, AVX512 support
- Intel Xeon Skylake SKL (Gold 6130)
  - 32 physical cores, 64 logical cores, 2.1 GHz, AVX512 support
- Intel Xeon Sandy Bridge SNB (E5-2620)
  - 12 physical cores, 24 logical cores, 2.0 GHz, AVX2 support