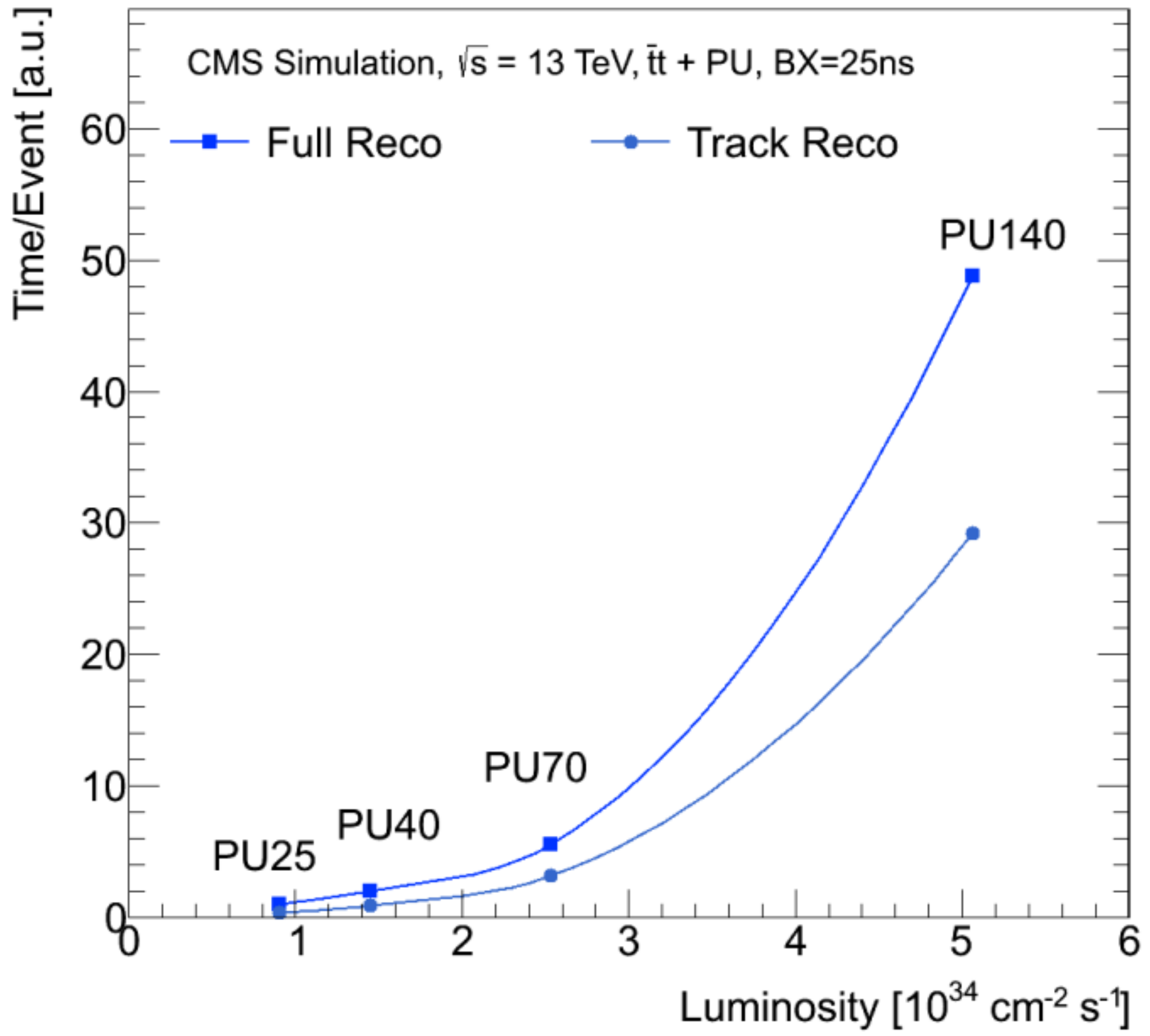
Parallelized Kalman-Filter-Based Reconstruction of Particle Tracks on Many-Core Architectures

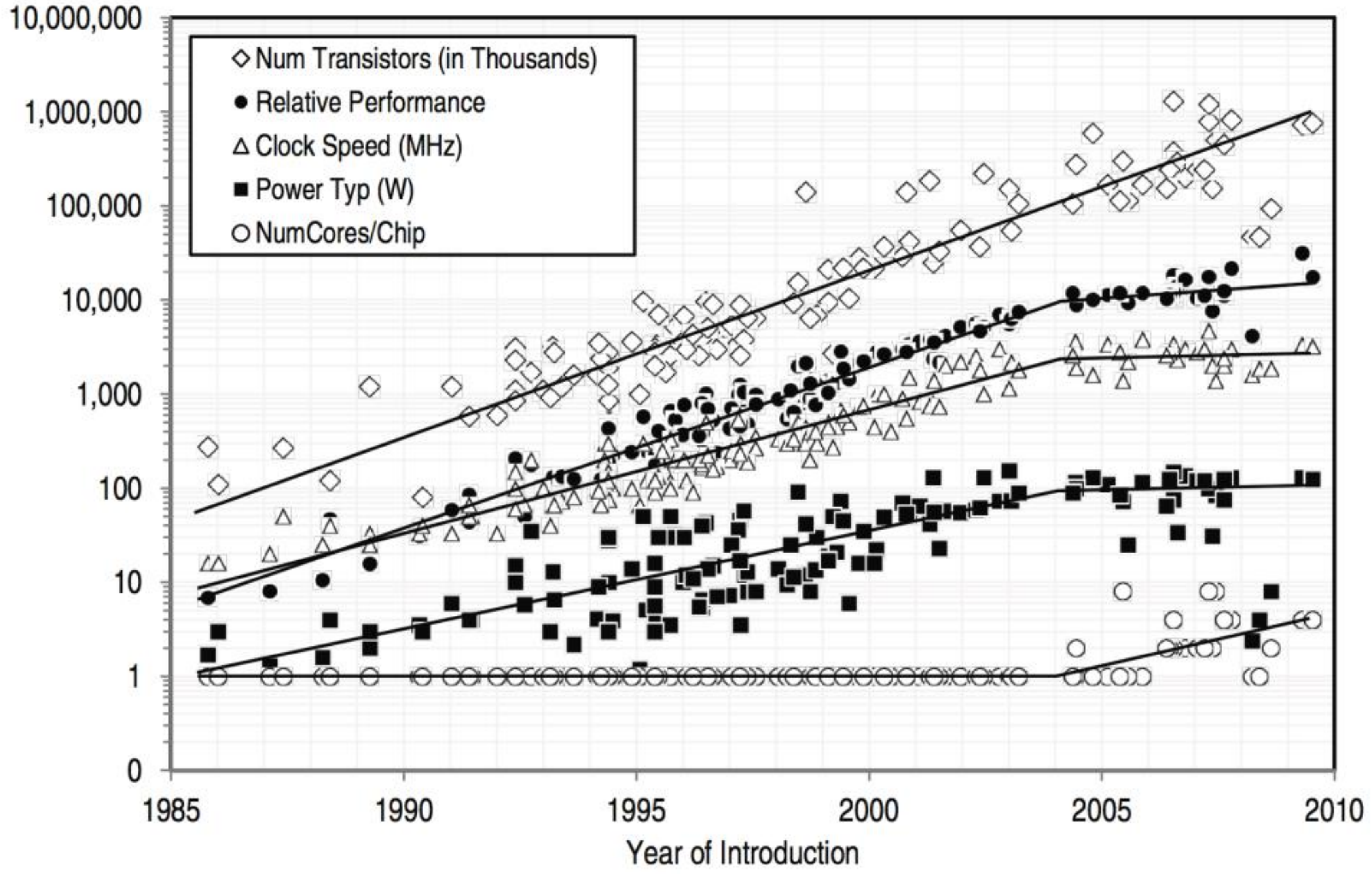
ACAT2017

G. Cerati[4], P. Elmer[3], S. Krutelyov[1], S. Lantz[2], M. Lefebvre[3], M. Masciovecchio[1], K. McDermott[2], D. Riley[2], M. Tadel[1], P. Wittich[2], F. Würthwein[1], A. Yagil[1]

1. University of California – San Diego
2. Cornell University
3. Princeton University
4. Fermi National Accelerator Laboratory

Why Many-Core?





- Instantaneous luminosity of the LHC is expected to continue increasing the High Luminosity era
- Higher detector occupancy means more time spent in event reconstruction

- Clock speed has stopped scaling (power consumption, heat dissipation, etc.)
- Number of transistors is still increasing
- More cores/chip, more SIMD

Kalman Filter

Kalman Filter two-step:

- Produce an estimate of the current state (prediction)
- Update the state with the next measurement

Why use it for tracking:

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

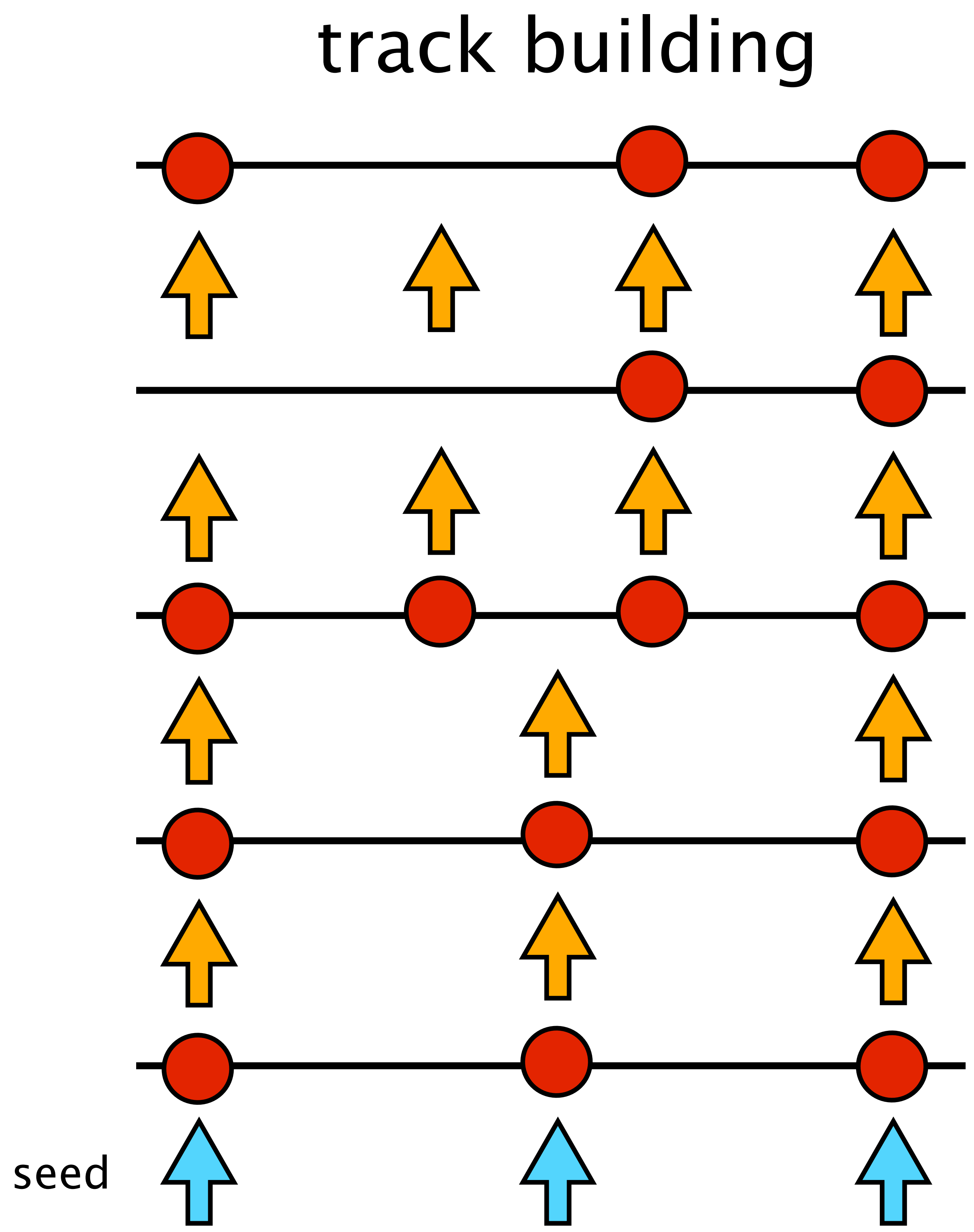
Our goals for Kalman Filter (KF) track building on many-core architectures

- Make effective use of parallel and vector architectures
- Maintain physics performance
- Preserve consistent systematics across platforms

Track Building Basics

Algorithm (for a single seed):

- Start with a seed track from 3 or more measurements
  - Seed finding is currently out of scope for us
- Estimate the track state from the seed track
- Propagate the track state to the next detector layer
- Find candidate detector response "hits" near the projected intersection point(s) of the track with the detector
- Evaluate the goodness of fit of each hit wrt the track
- Select the best fit track/hit combinations as track candidates
- Update the estimated state of all track candidates with the new hit
- Propagate all track candidates to the next layer and iterate



Track Building Challenges

Good efficiency requires considering multiple hypotheses

- In a dense detector, many tracks will find hit candidates that are the best local fit but lead to a globally poor fit
- Acceptable efficiency typically requires considering ~6 or more track hypotheses for every seed depending on detector occupancy

Track building involves multiple branch points

- Selecting candidate hits at each layer
- Evaluating a variable number of track candidate/hit candidate combinations
- Selecting the best combinations for propagation to the next layer
- Many seeds turn out to be false leads, dying out after a few layers

Branch points lead to irregular work loads and memory access patterns

Our Approach

Start simple:

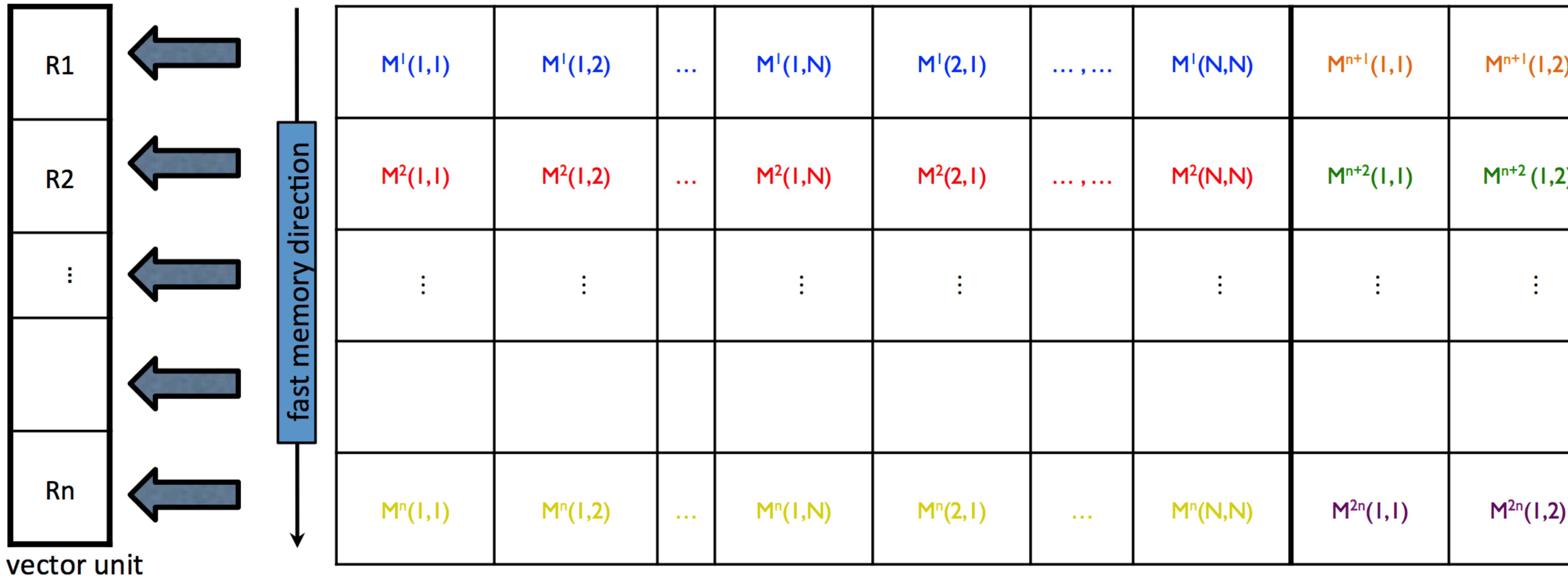
- Knights Corner (KNC) Xeon PHI and Sandy Bridge (SNB) Xeon
- Regular cylindrical geometry
- Lots of tracks per event, uniformly distributed in η, simplifying work distribution
- Tracks seeds from Monte Carlo “truth”
- Track fitting (all hits known) as a warm up exercise before track building
- Develop measurement and validation tools, techniques and intuition

Then add complications—this is where we are now:

- Realistic geometry with endcaps and transition regions
- Realistic events from CMS simulation
- Seeds from CMS track finding
- Additional platforms: Knights Landing, GPGPU

Data structure: Matriplex

"Matrix-major" matrix representation designed to fill a vector unit with **n** small matrices operated on in synch



Use vector-unit width on Xeons

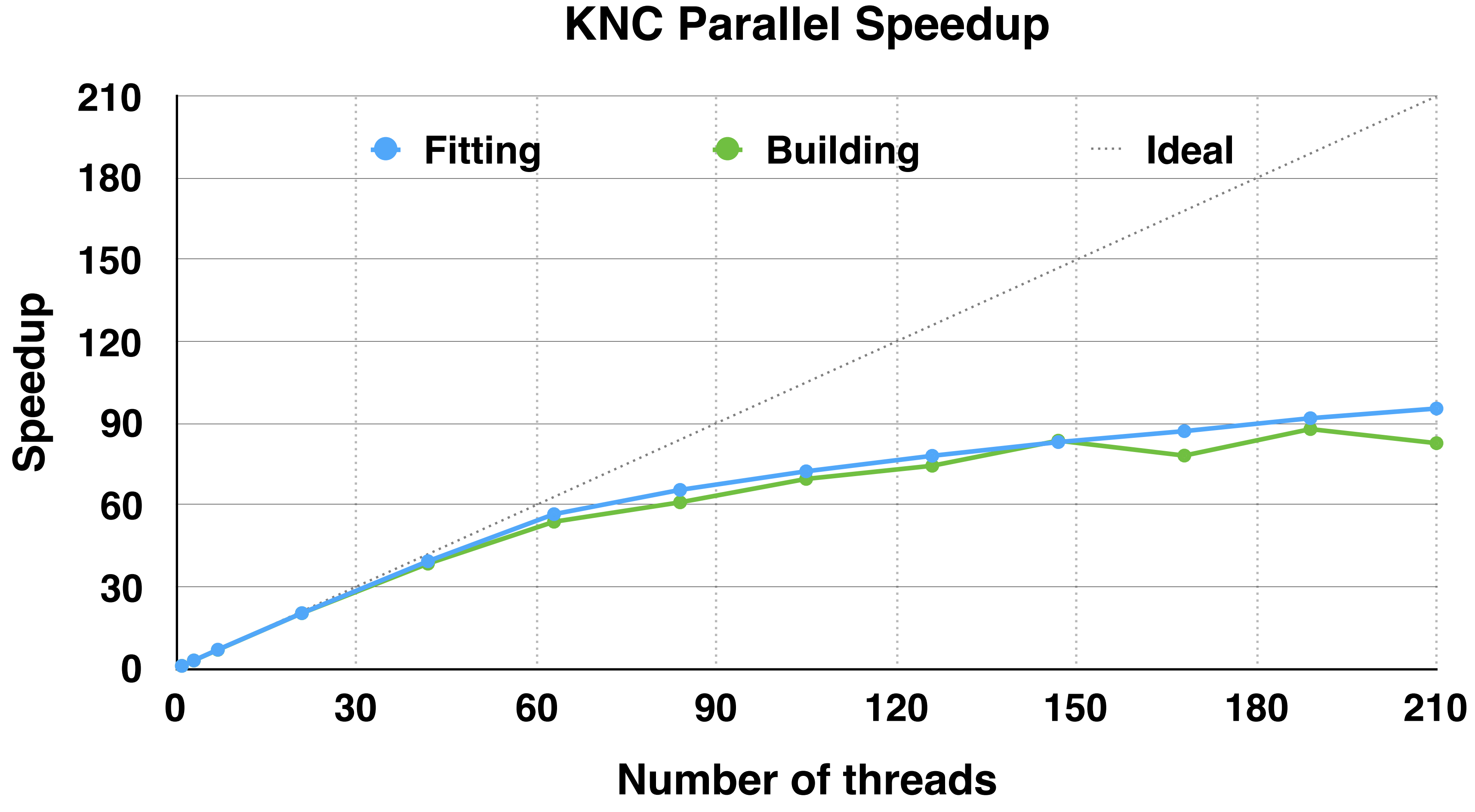
- With or without intrinsics
- Shorter vector sizes w/o intrinsics
- For GPUs, use the same layout with very large vector width

Interface template common to Xeon and GPU versions

Results from Starting Simple

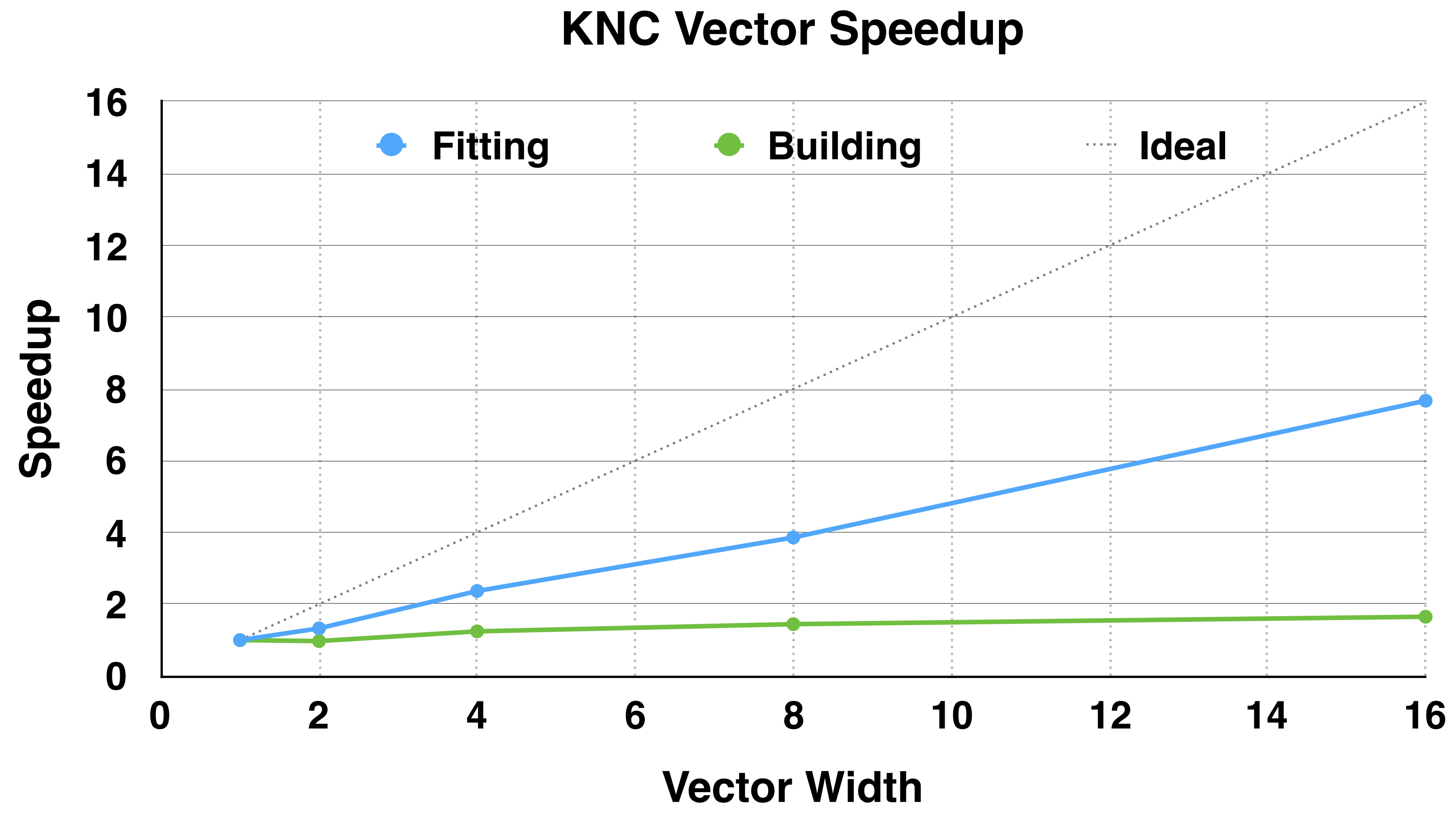
Results are from a KNC Xeon Phi 7120P

- 61 cores, but needs 122 threads to utilize all clock cycles
- AVX-512 vector width gives 16 single-precision floats
- SNB Xeon results generally better
  - But not as interesting



Parallelization:

- Matriplexes are assigned to threads via Threading Building Blocks (TBB) tasks
- Near ideal up to the number of physical cores, some resource contention past that



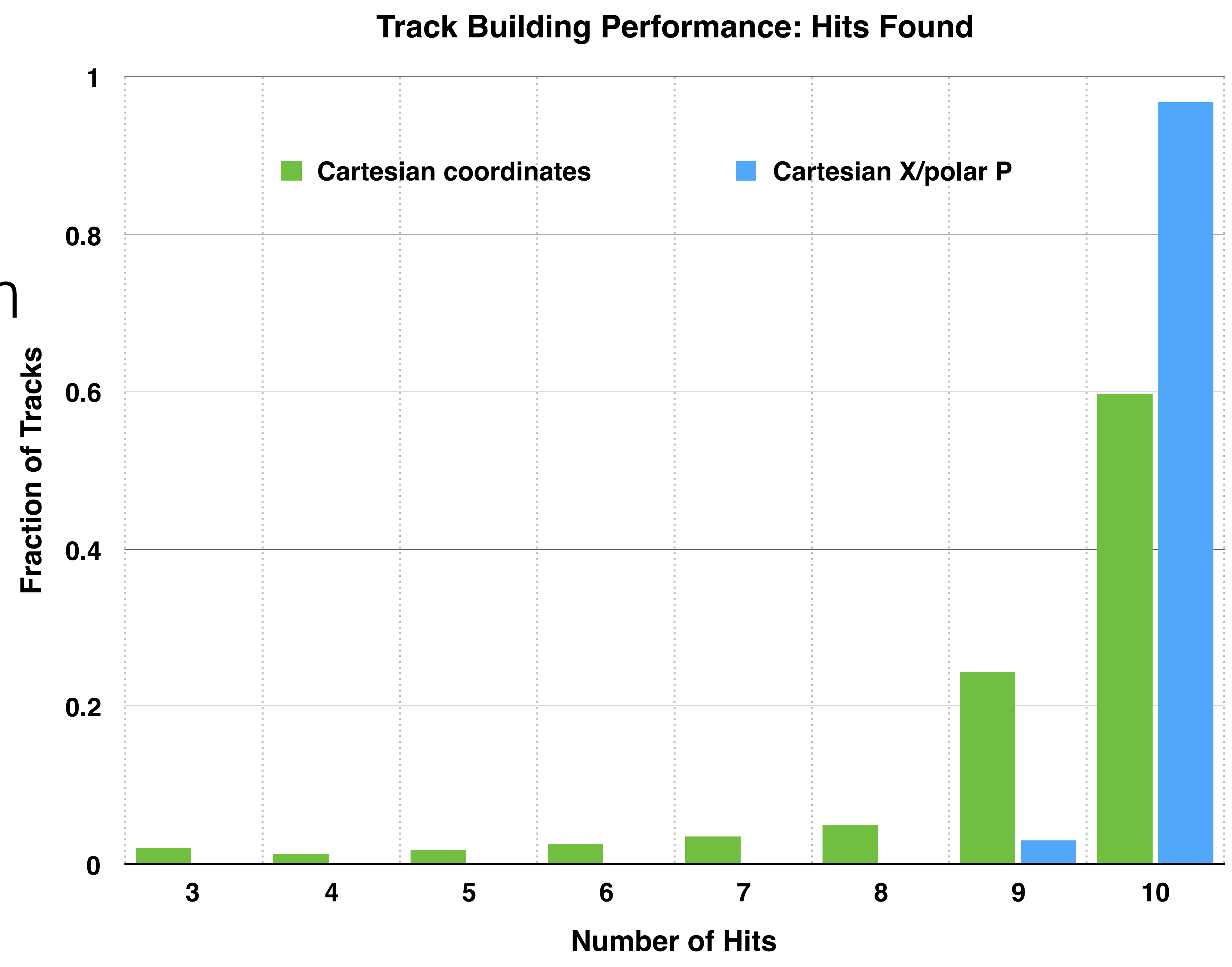
Vectorization:

- Track fitting achieves about half the ideal vector speedup
- Track building vectorization still needs work

Lessons from the Simple Version

Physics performance lessons:

- Coordinate system choice matters
  - We eventually adopted spatial Cartesian, polar momentum
  - Error matrix is more complex
  - Better prediction performance speeds up track finding



Computing performance:

- Keep data structures and memory allocations minimal
- Data locality is critical
- Reduce tail effects in the work distribution via TBB work stealing
- Pay attention to vectorization reports
  - Avoid unaligned accesses and type conversions
  - Use prefetching, scatter/gather
  - Use 'const' and minimize the scope of variables

Adding Complications

Realistic detector geometry

- Endcaps and transition region present new challenges
- Real detectors can have very complex geometries
  - Contributes to memory pressure, takes time to navigate

Realistic events

- Real events may have lower occupancy and less uniform distribution than our simplified events
  - New issues with even distribution of work

New platforms

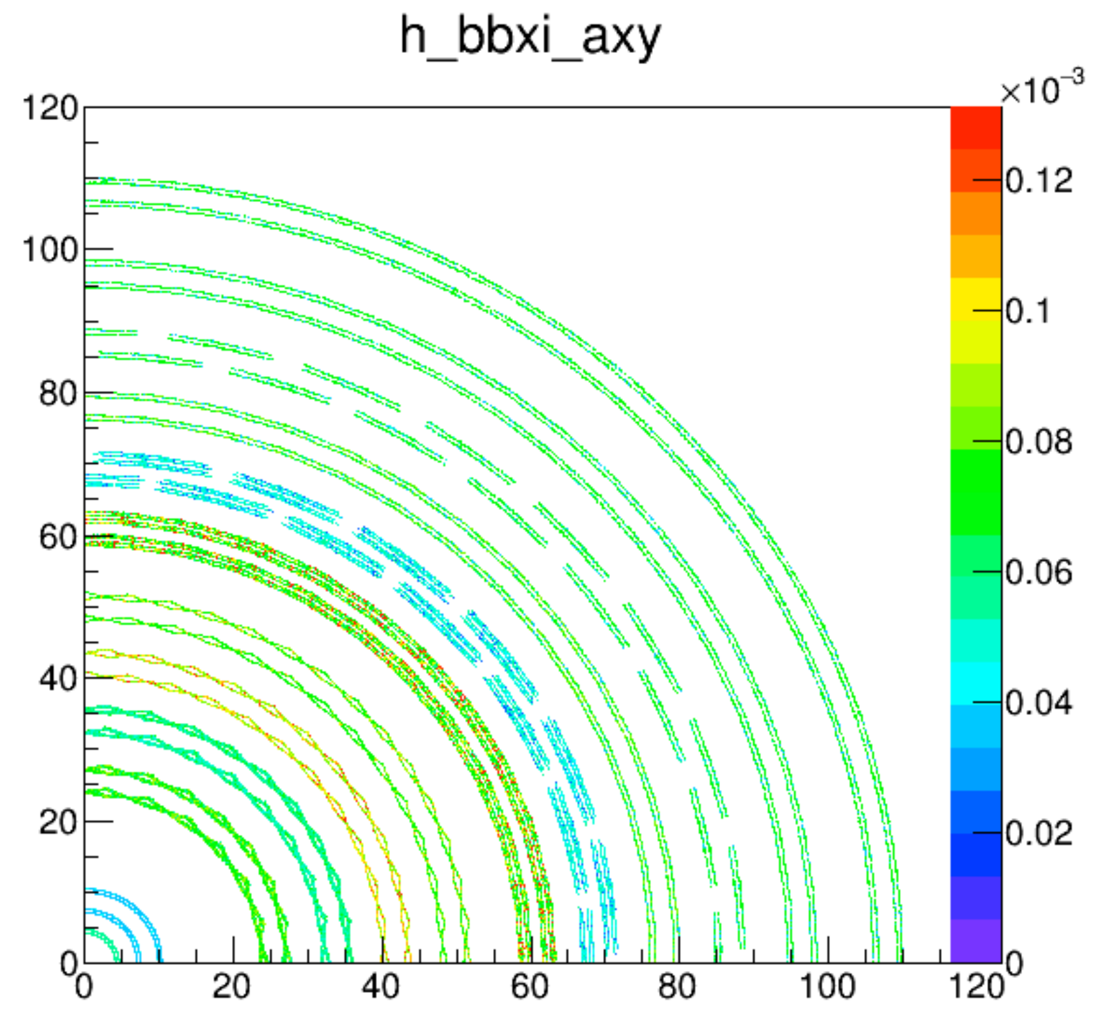
- KNL: similar to KNC, but new memory organization & CPU micro-architecture
- GPU: different programming model, how well can our code adapt?

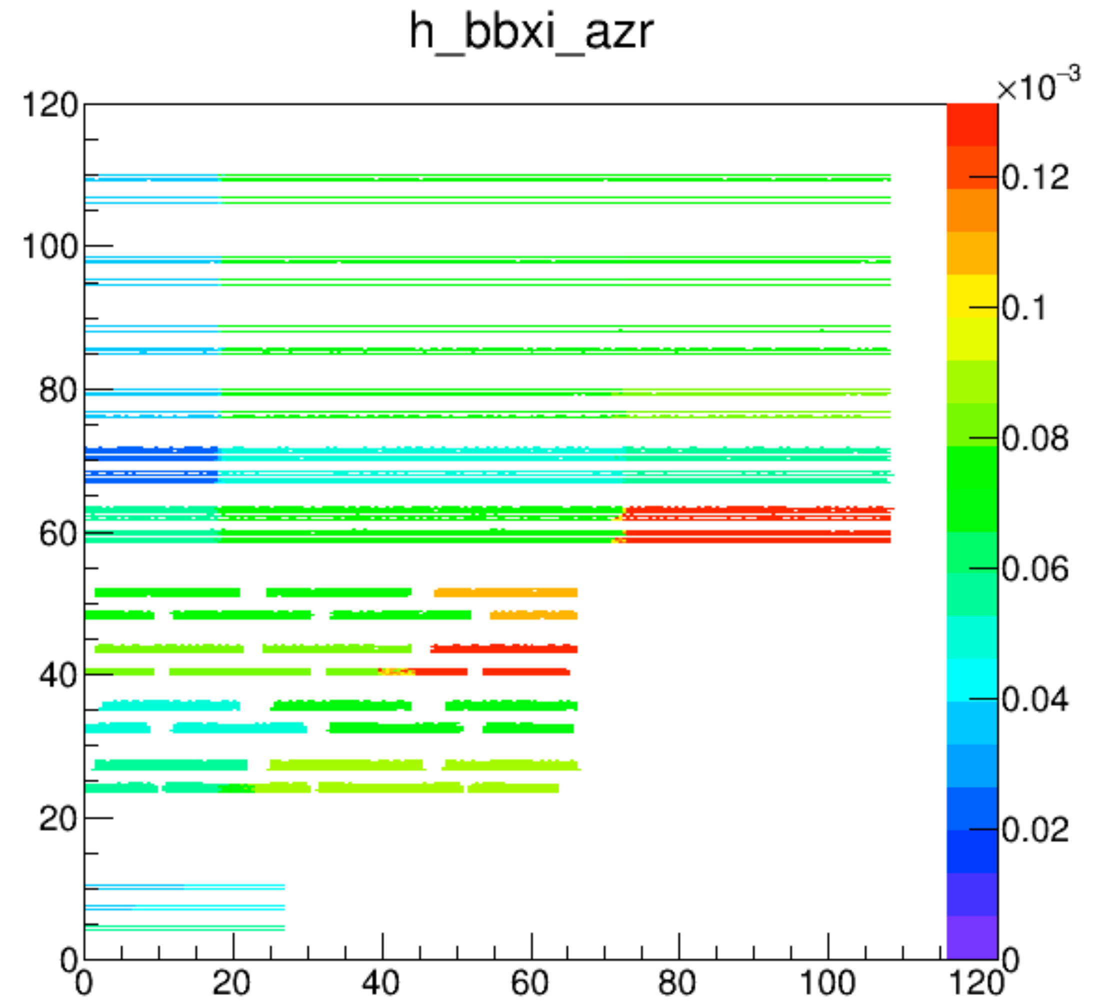
This is work in progress, so the rest of the talk will be more anecdotal

Realistic Geometry

Adding two new geometries:

- "Cylinder with lids" adds endcaps to our idealized geometry
    - Use for algorithm development for endcaps and transition region
- CMS geometry using CMS data
    - Propagate tracks to average radius of the layer
    - Find hits in the compatibility window
    - Propagate to each hit location and compute the χ^2
    - Advantage: work with a simplified geometry
    - Disadvantage: have to inflate the search window





Status:

- Barrel and endcaps implemented, still working on transition region
- Performance has not been tuned or tested
    - Doing physics validation first

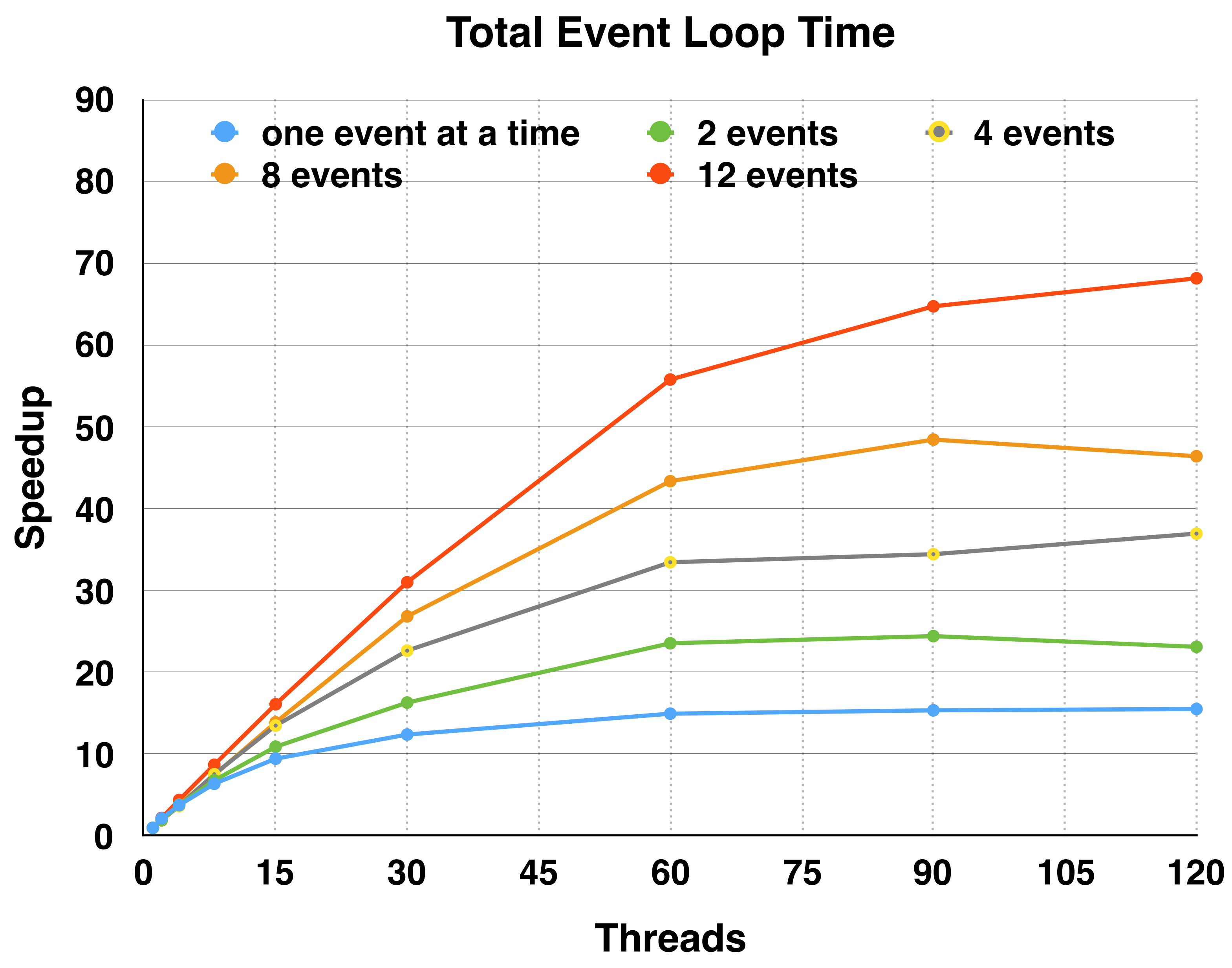
Compensating for Variable Occupancy

CMS events often have fewer good tracks than the simulated events in our simple setup

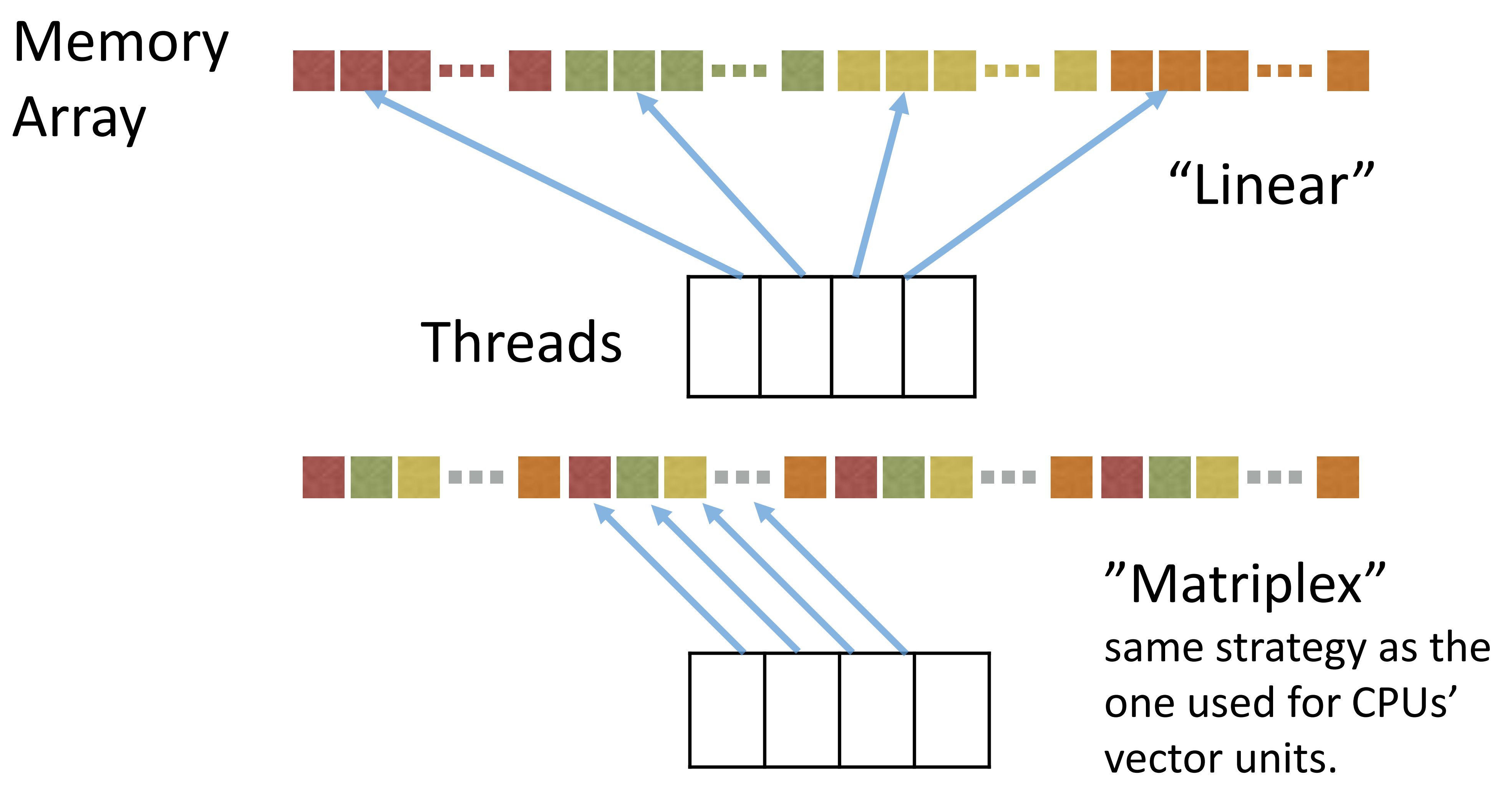
- Lower occupancy causes difficulties keeping the processors busy and vector units full

Process multiple events at the same time

- Multiple events can fill in gaps in parallelism due to varying levels of parallelism within an event
- Still scaling limits due to per event data structures
  - Tradeoffs due to granularity vs. memory usage of the binning structure used for finding candidate hits
  - At very low occupancy *k*-d trees can be effective



GPU: Choice of Memory Layout

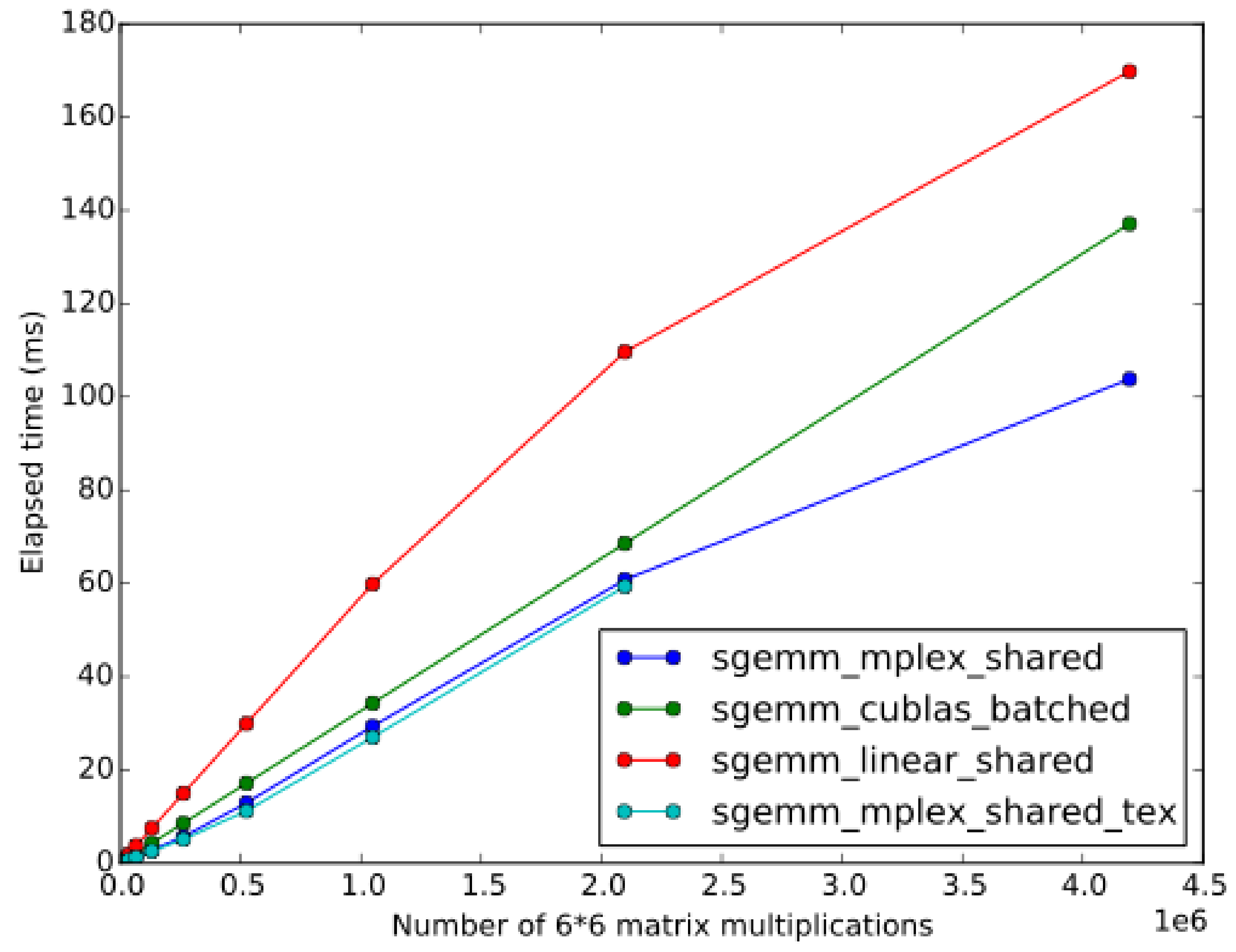


Linear vs. Matriplex

- For multiplying lots of 6x6 matrices, the Matriplex layout gave better performance than the obvious alternatives

Use a very large Matriplex-style structure

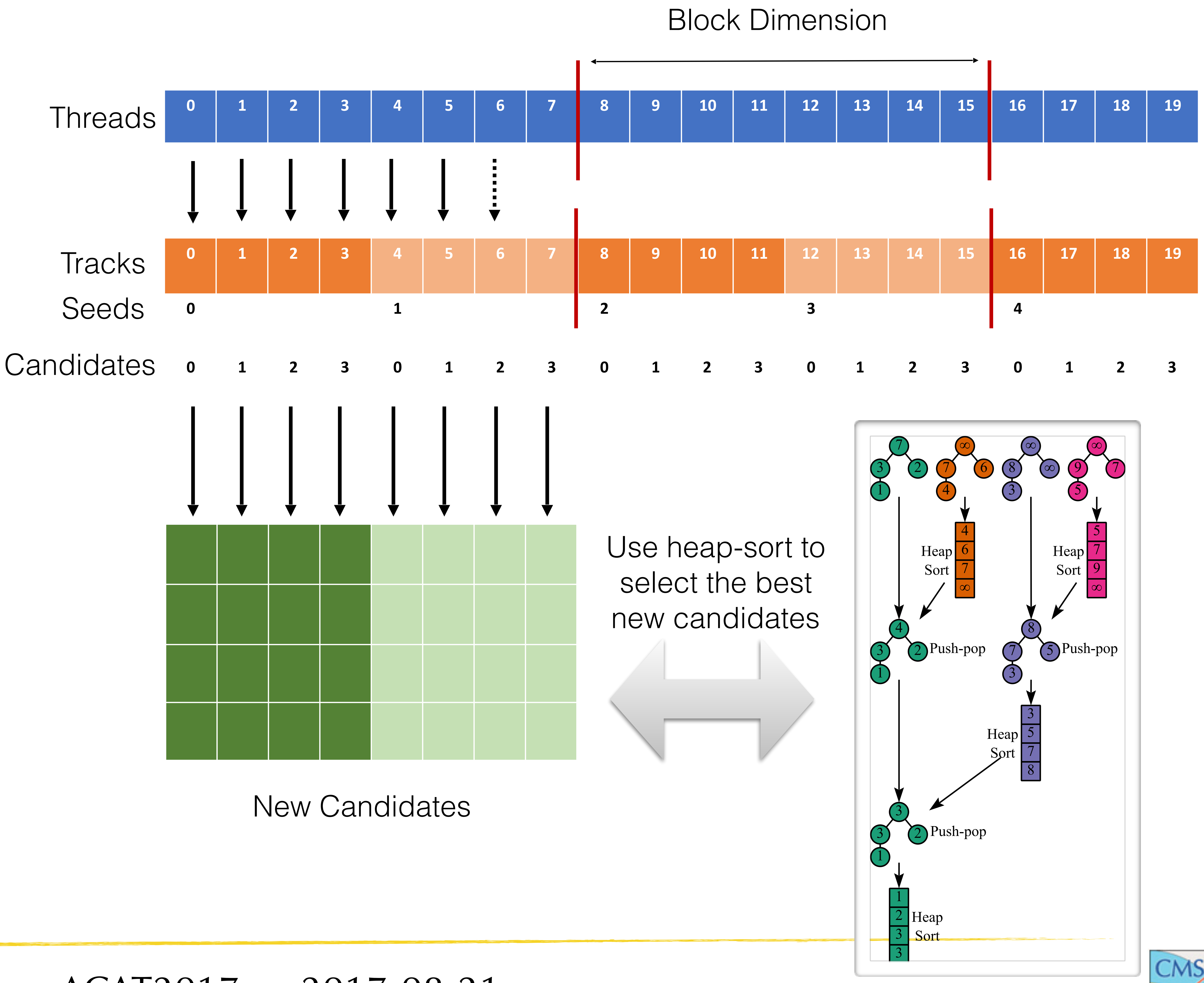
- GPlex: same interface as Matriplex, but customized for GPU/CUDA
- Opens the possibility of templating many of the core Kalman routines to accept either



GPU: Handling Branching

Moving tracks in global memory is prohibitively expensive

- For parallelization, one GPU thread per candidate
- Heap-sort the new candidate list to select the best new candidates



CPU: Track Building Vectorization

How can we improve track building vectorization on CPUs?

- Most of the non-vector sections are moving track candidates around in memory
  - Considering copying the GPU approach of fixed assignments of vector units to seed candidates
- Finding candidate hits to add to the track, naively implemented, vectorizes poorly
  - Search window varies, number of hits found per track candidate varies
  - Split into three loops, two out of three can vectorize
  - Smarter data structure choices?

```
for track : tracks
  calculate z, φ windows
  find bins in z, φ windows
  for zBin : zBins
    for phiBin : phiBins
      for hit : hits[zBin][phiBin]
        calculate hit-track dphi, dz
        if ok(dz) && ok(dphi) && track.candidates < candMax
          add hit.hitid to track.candidates
```

Could Vectorize

Vector Problem

Perspective

The “start simple” plan has worked well for us

- Achieved good parallelization and (mostly) good vectorization on KNC
- Having a baseline for comparison has been a great help as we tackle the complications

Complications are on track

- Realistic geometry and events are nearing completion
- Lessons learned seem to be carrying over well to new architectures
- Progress is being made on the GPU front

Lessons learned on architecture can be valuable on others

- CPU choices of data structures influenced the GPU version
- Some sharing of low level code (but steering logic differs)
- Lessons learned from GPU are starting to be applied back to the CPU version

Backup Slides

Splitting vector vs. non-vector loops

Overview:

- First loop calculates the search windows; this trivially vectorizes
- Second loop make a list of hits within the search windows
- Third loop is reorganized to check every hit against every track
  - the loop over tracks vectorizes

Problems:

- There's only a benefit if the track candidates have many candidate hits in common
  - This should be true if the candidate tracks are mostly from the same seed

```
for track : tracks
  calculate vector of z, φ windows
  find vector of bins in z, φ windows

for track : tracks
  for zBin : zBins
    for phiBin : phiBins
      add z/phi bin to bins

for bin : bins
  for hit : bin.hits
    clear hitmask
    for track : tracks
      calculate hit-track dphi, dz
      hitmask[track] = ok(dz) && ok(dphi)
    for track : tracks
      if hitmask[track] && track.candidates < candMax
        add hit.hitid to track.candidates
```