Parallel Charged Particle Tracking Reconstruction

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. Fermilab

Large Hadron Collider

Large Hadron Collider

downtown GVA, Alps

Large Hadron Collider

Large Hadron Collider

Large Hadron Collider

main lab

Large Hadron Collider

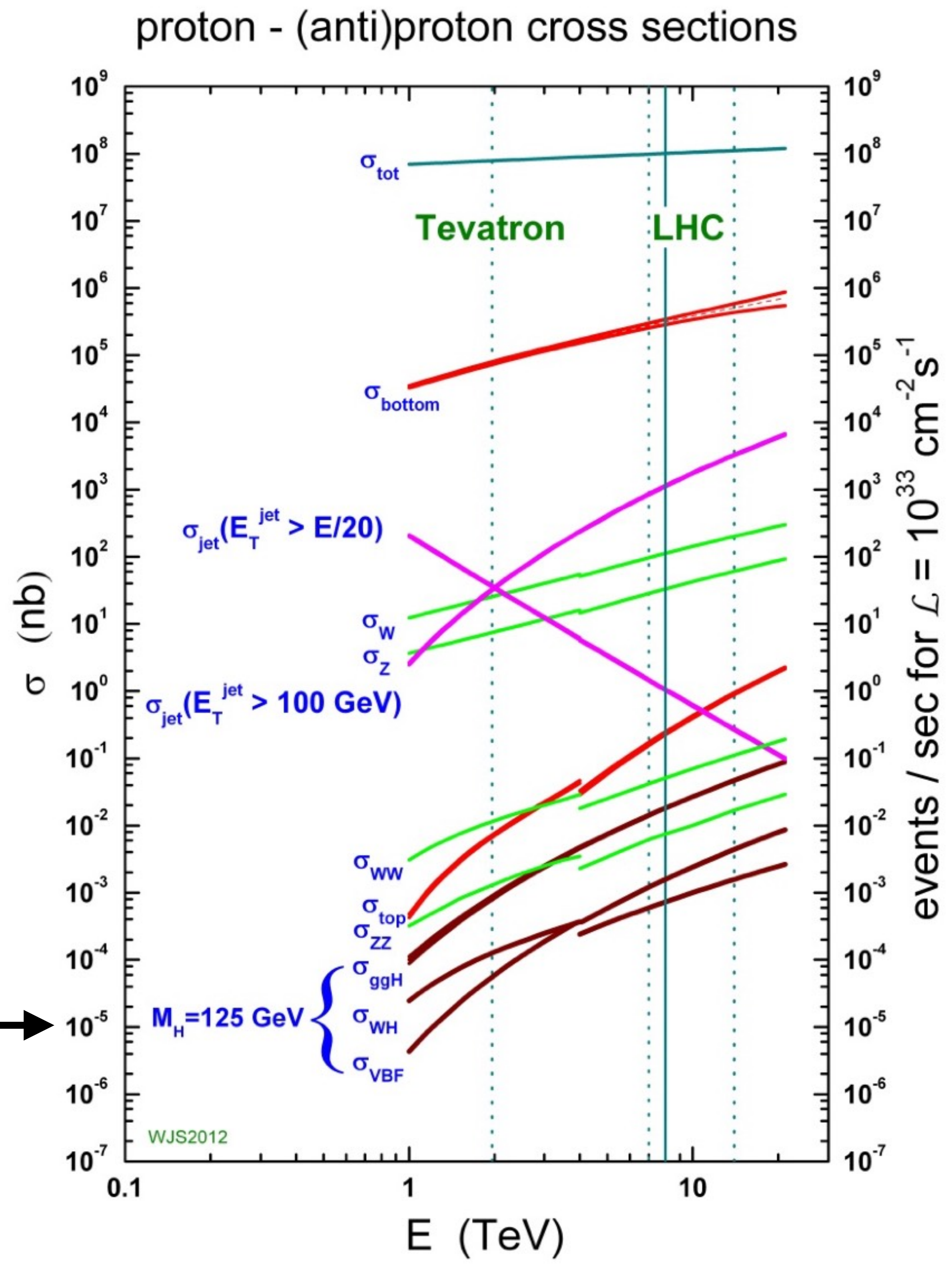
Large Hadron Collider

Jura Mountains

Big Data Challenge

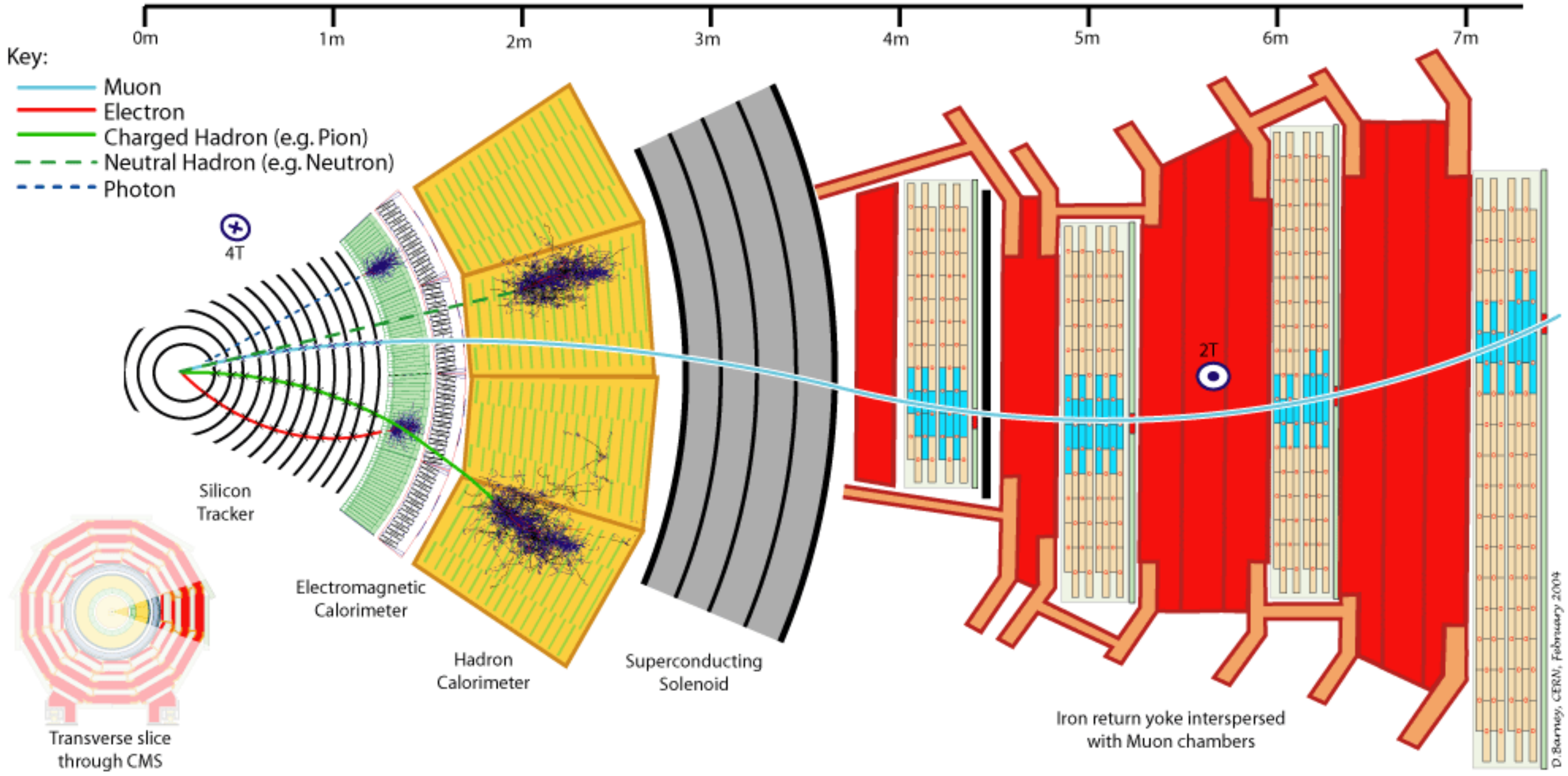
- 40 million collisions a second
- Most are boring
  - Dropped within 3 μs
- 0.5% are interesting
  - Worthy of reconstruction...
- Higgs events: super rare
  - 10^{16} collisions → 10^{6} Higgs
  - Maybe 1% of these are found

- Ultimate “needle in a haystack”
- First “Big Data” problem



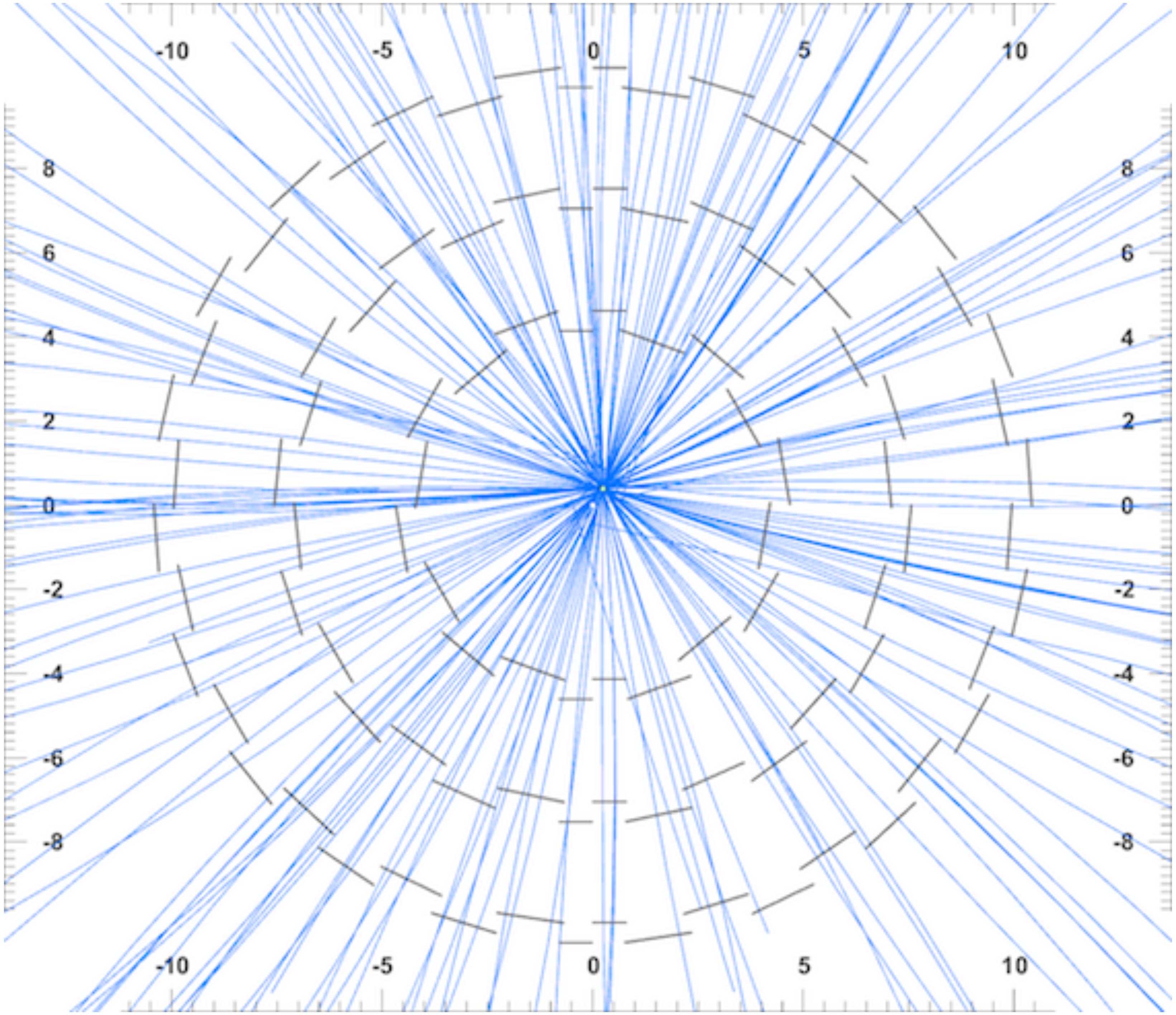
http://www.hep.ph.ic.ac.uk/~wstirlin/plots/plots.html

Collider detector



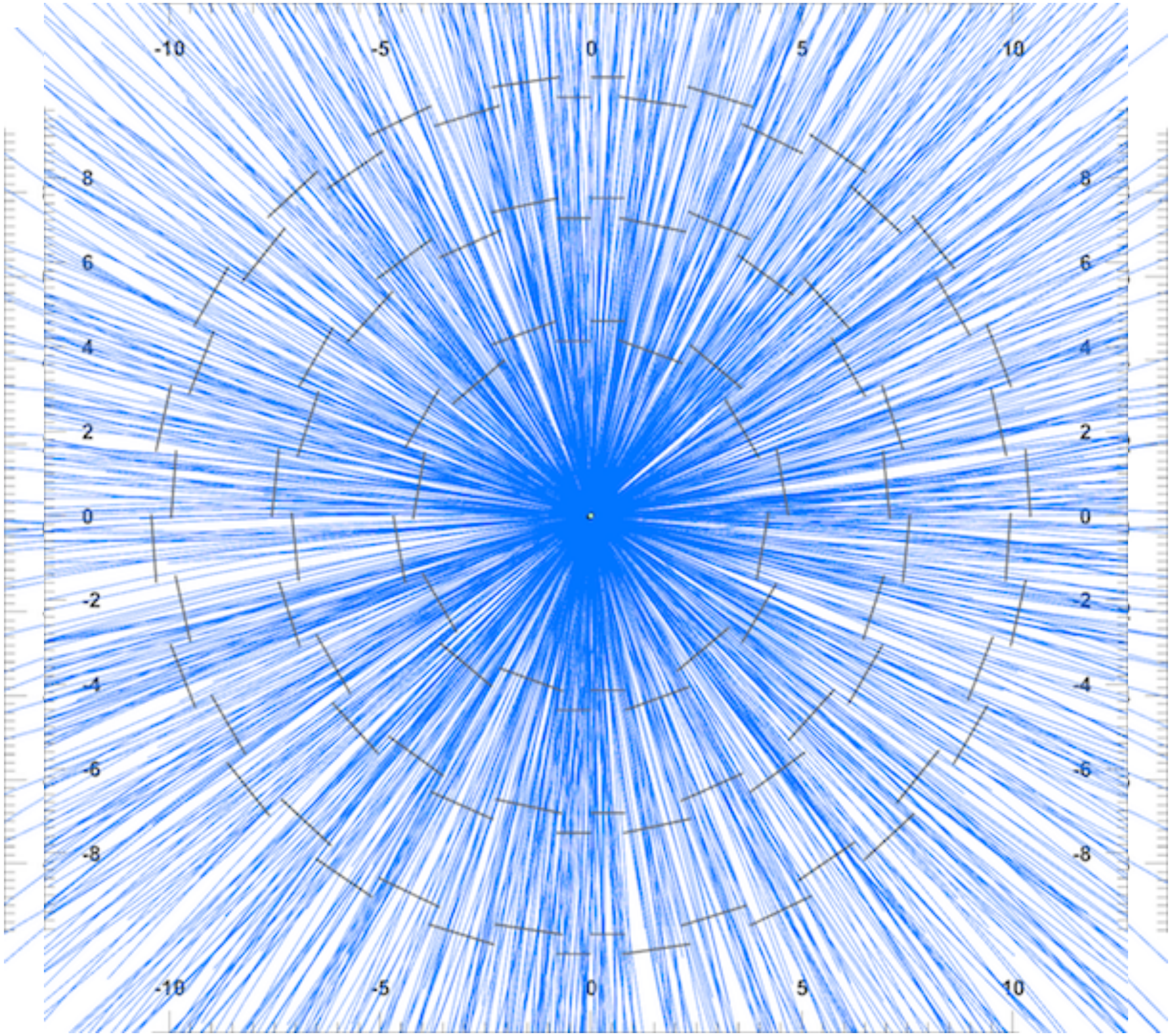
Particles interact differently, so CMS is a detector with different layers to identify the decay remnants of Higgs bosons and other unstable particles

Pile-up



Future holds big increases in pile-up: from 20 to 200

Pile-up

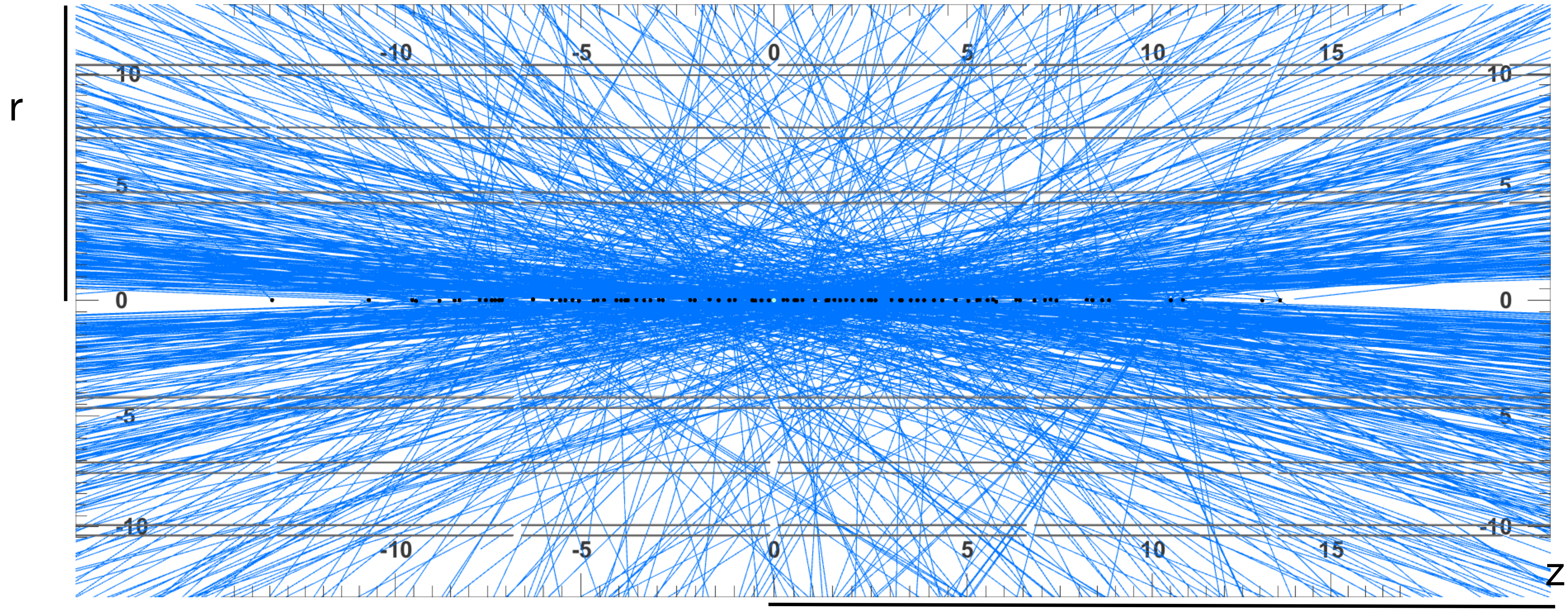


High Luminosity LHC: increased beam intensity

Future holds big increases in pile-up: from 20 to 200

CMS Is About to Get Busier

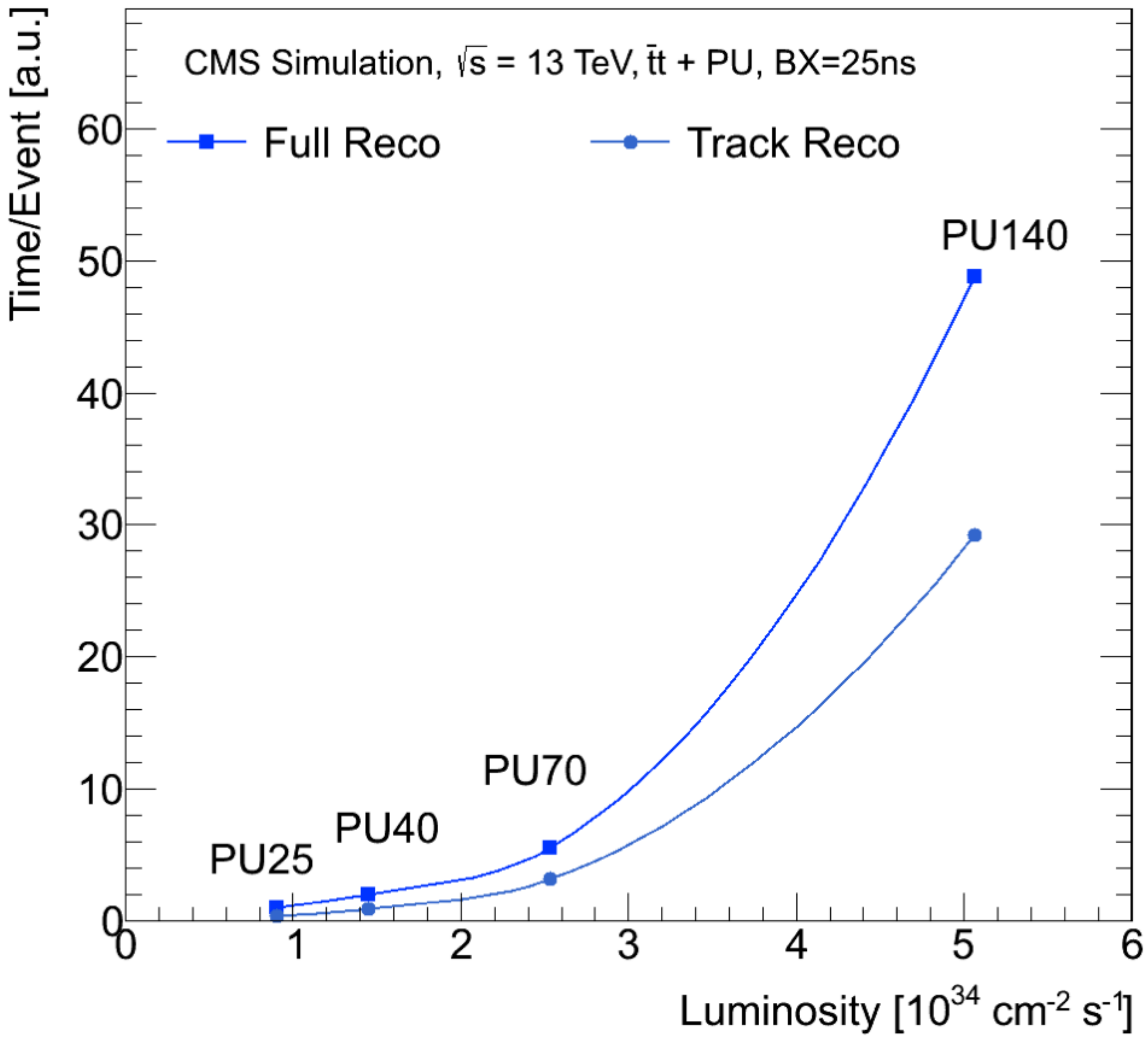
Simulation of pile-up = 140
at CMS in r-z plane



- By 2025, the instantaneous luminosity of the LHC will increase to 7.5e34/cm^2/s — High Luminosity LHC (HL-LHC)
- Significant increase in number of interactions per bunch crossing, i.e., “pile-up”, on the order of 140–200 per event

Reconstruction

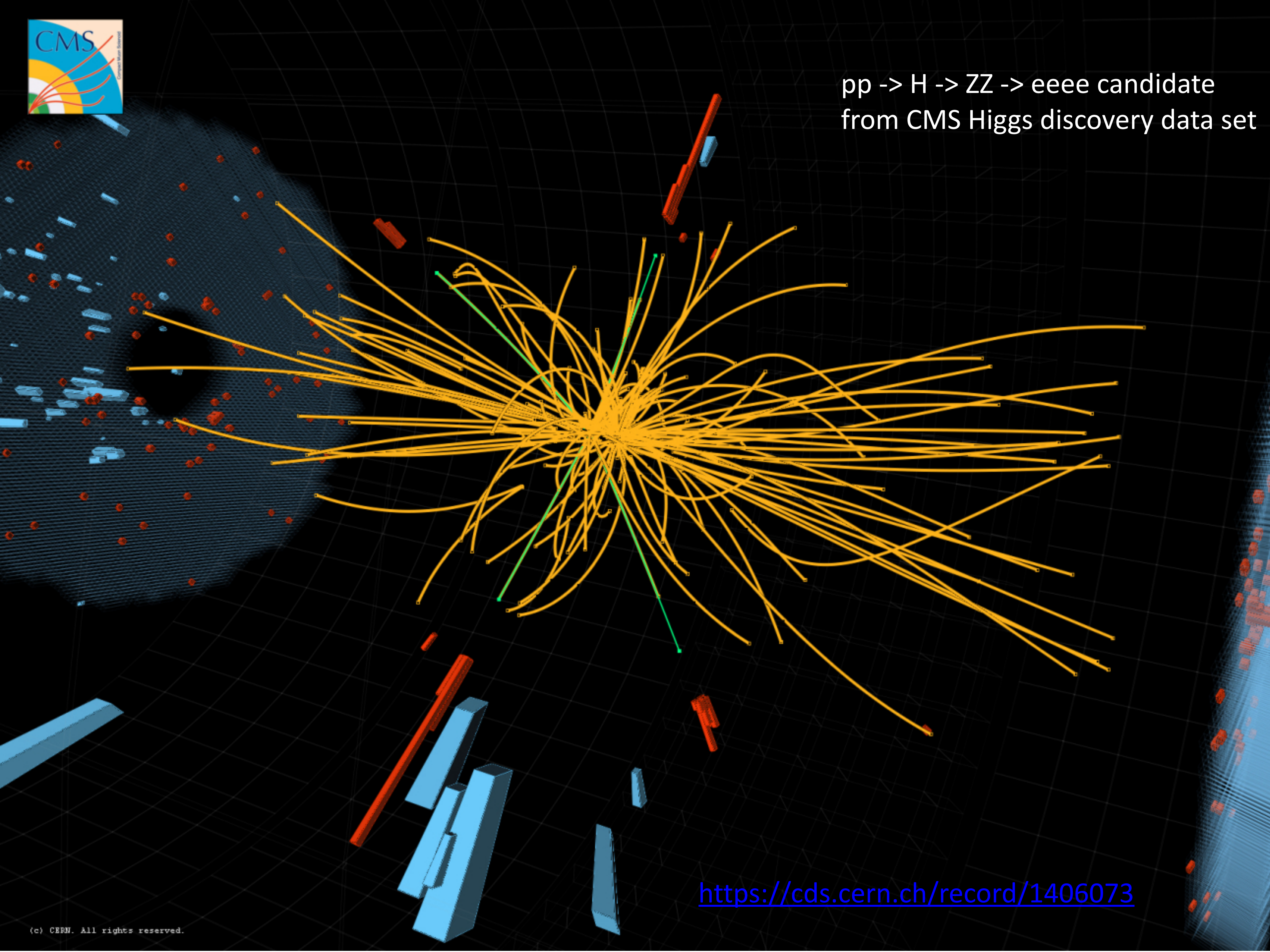
- Going from detector primitives (energy deposits in various elements) to particles: “reconstruction”
- Tracking is the most time-intensive part of reconstruction — combining the hits in the ***tracker*** to form the trajectories of the charged particles
- O(1e6) measurement stations per event, across many layers
- Can we make the tracking algorithm concurrent and speed up the reconstruction?



Modern collider tracker

- solid-state detector
- similar tech as your cell phone camera
- exquisite position measurement ability
  - tens of μm spatial resolution in rϕ plane
- but ***massive*** detectors
  - 200 m^2 of silicon area, 10^7 channels
  - particle trajectories are affected by the device measuring it (scattering, nuclear interactions)

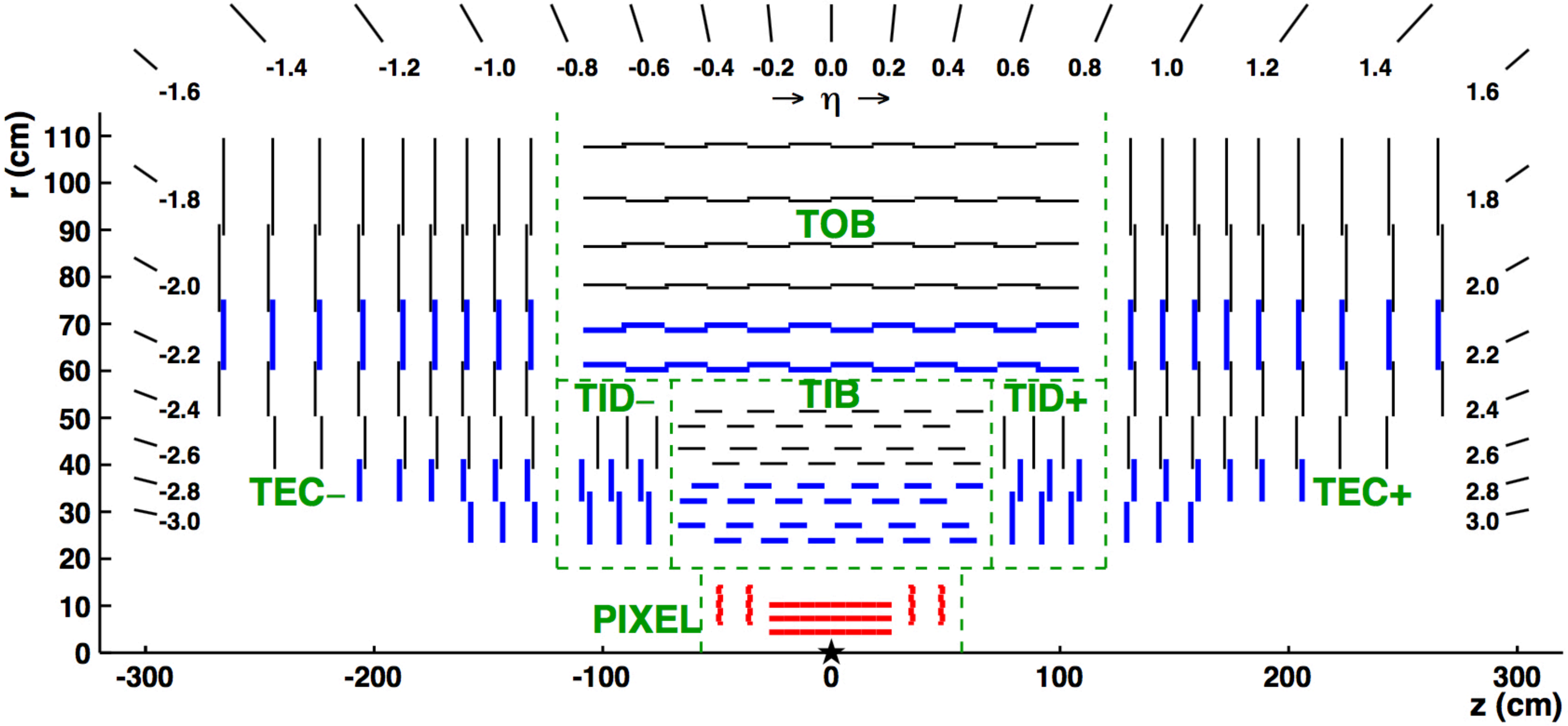
- CMS: doi:10.1088/1748-0221/9/10/P10009

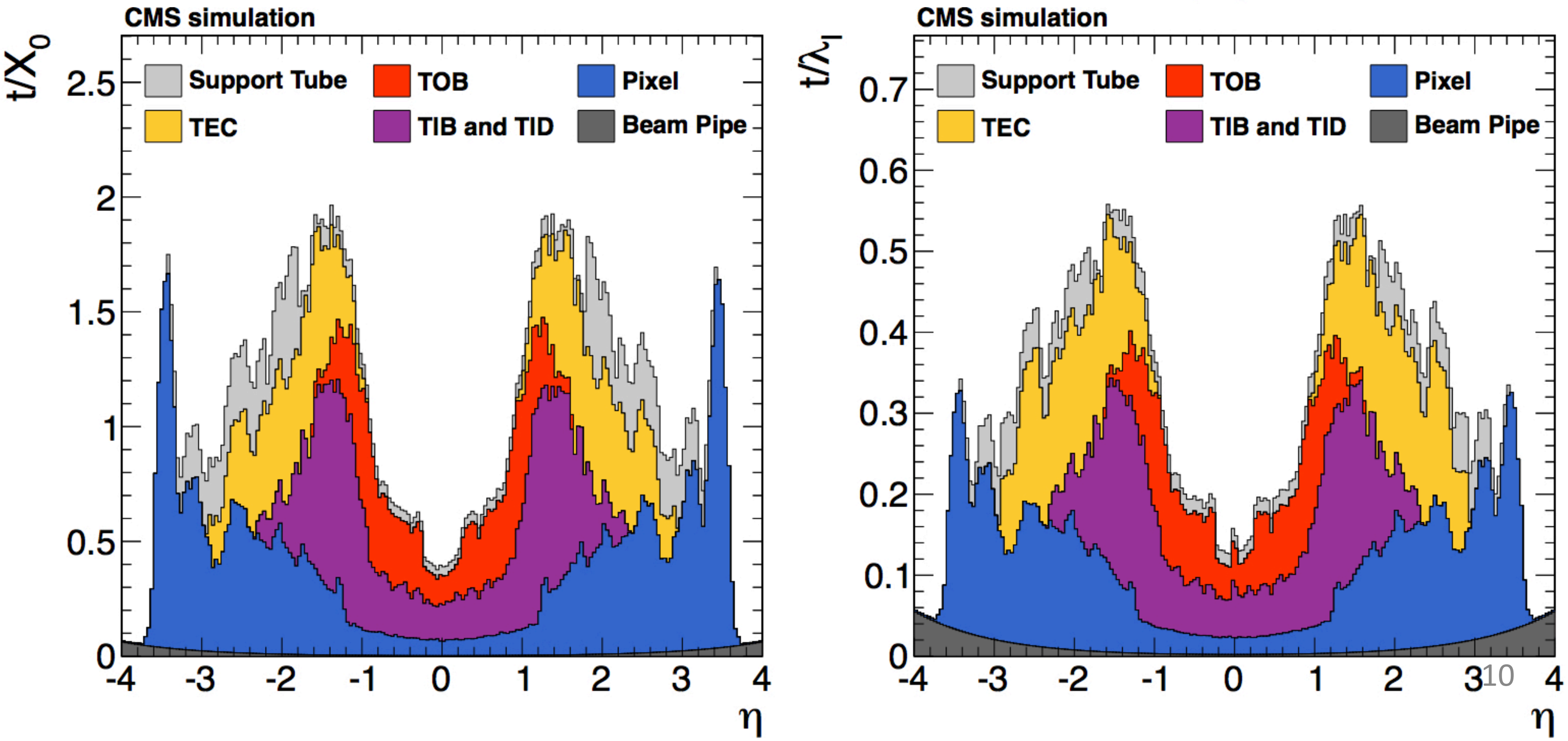


pp -> H -> ZZ -> eeee candidate
from CMS Higgs discovery data set

https://cds.cern.ch/record/1406073

CMS tracker detector



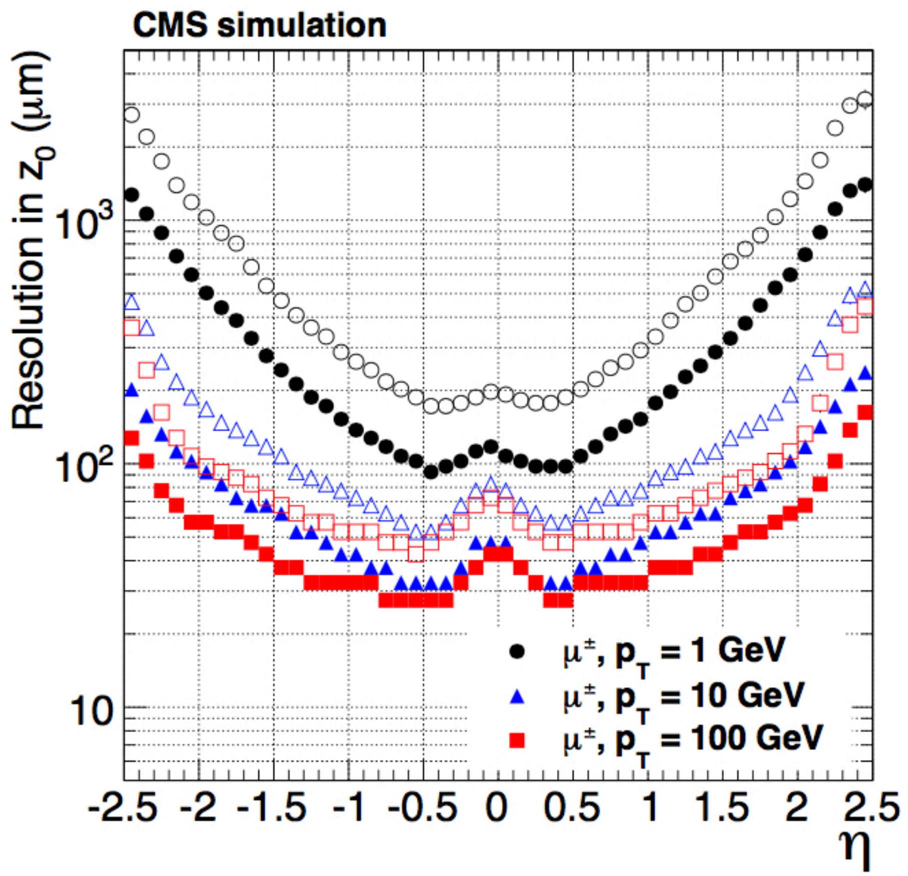


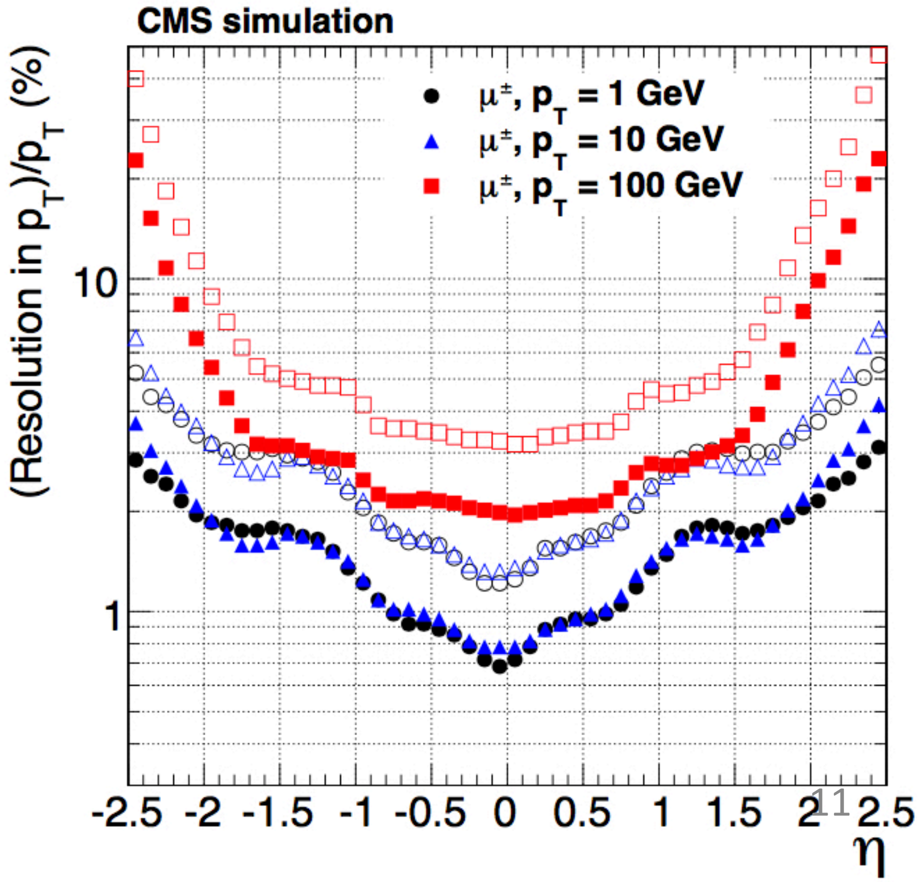
x_0: radiation lengths

λ_I: nuclear interaction lengths

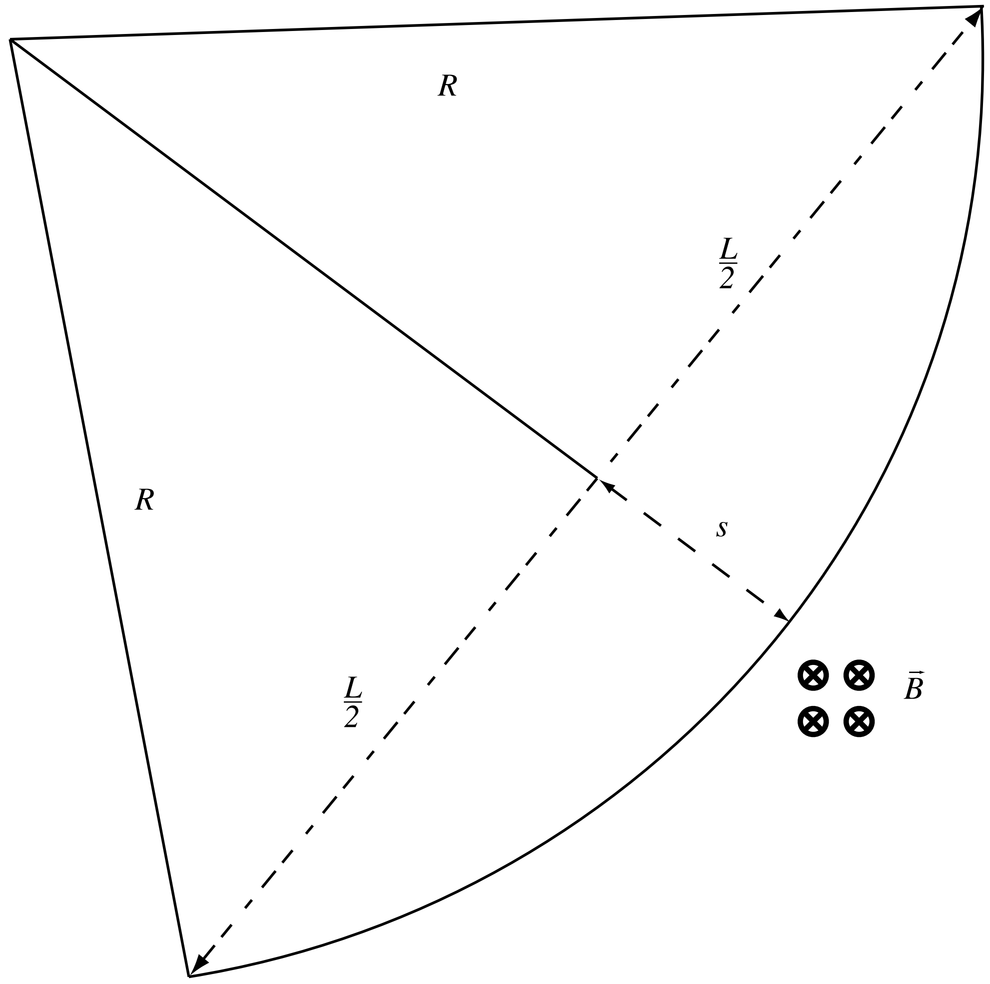
Job of tracker detector

- Measure passage of charged particles
- Find helical trajectory (p_T, η, φ, z_0, d_0)
    - Solenoidal B field - bending in one plane (“transverse”)
- find position of track in 2 places
    - close to beamline — to learn about hard scatter
    - at exit of tracker — to extrapolate to other detectors
- Measure charge of charged particle +/-
- Distinguish primary interaction from secondary interaction
    - decays of heavy particles
- Distinguish hard scatter from secondary interactions





Measuring momentum

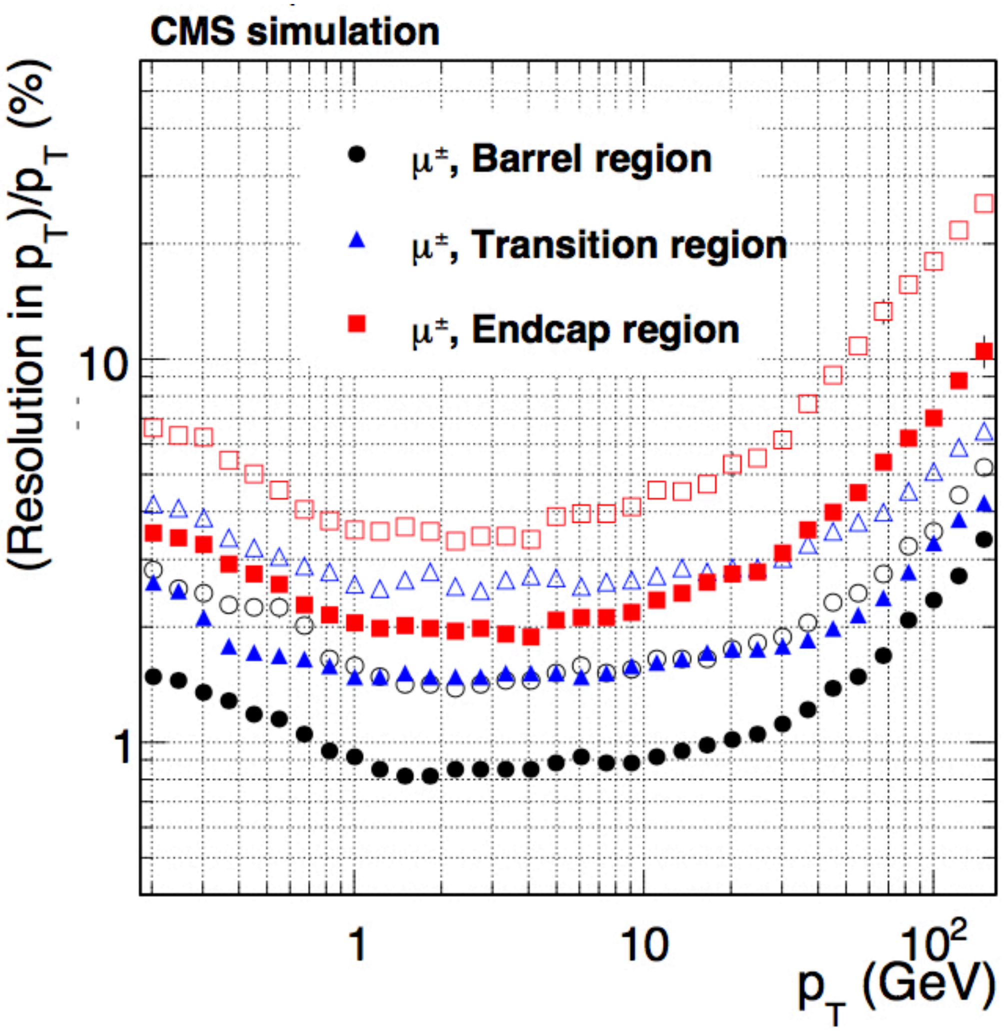


$$s \simeq \frac{L^2}{8R} \propto \frac{BL^2}{p_T}$$

$$\delta p_T / p_T \propto c \times p_T$$

$$p_T[\mathrm{GeV}] = 0.3\, B[T] \times R[m]$$

- s: sagitta
- R: radius of curvature
- sagitta — **bigger B, L** is better
- ***<u>—> large detector with strong B field</u>***
- pt resolution peters out at higher momentum
- CMS: 2-3% resolution at pT ~ 100 GeV



Tracking algorithm

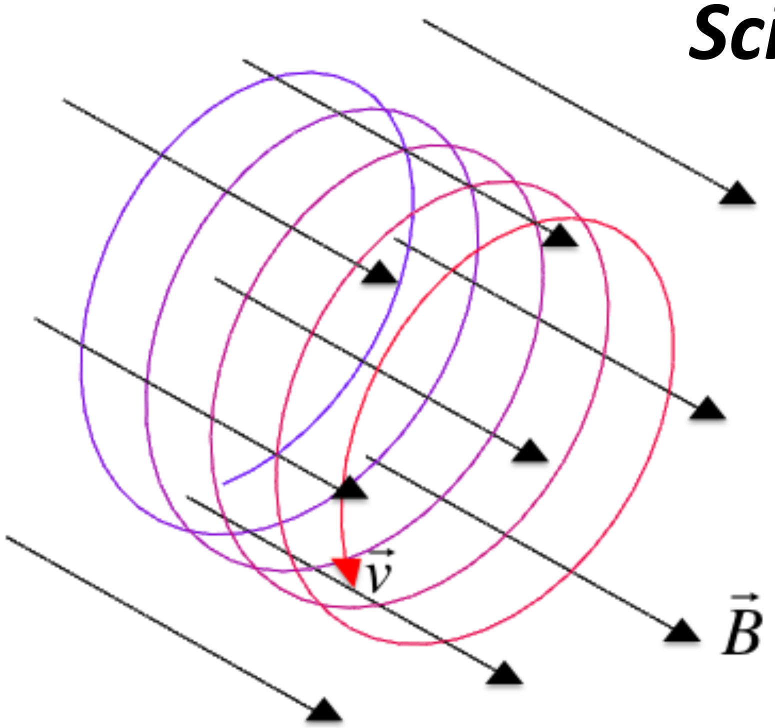
- Job: reconstruct the trajectory of a charged particle
- Get estimate of the track parameters, and related uncertainties
  - p_T, η, φ, z_0, d_0
- **Two parts**
  - ***<u>PATTERN RECOGNITION</u>*** (what hits come from one particle)
  - ***<u>FITTING</u>*** (best estimate of track parameters)
- more interested in high-momentum (high-pt) tracks than in low-pt tracks
- Account for trajectory changes due to interaction in material
- Focus on the ***initial track parameters*** (for physics!) and the ***exit position to the calorimeters*** (for reconstruction)

- ***We use a Kalman Filter-based technique***

Why Kalman Filter for particle tracking?

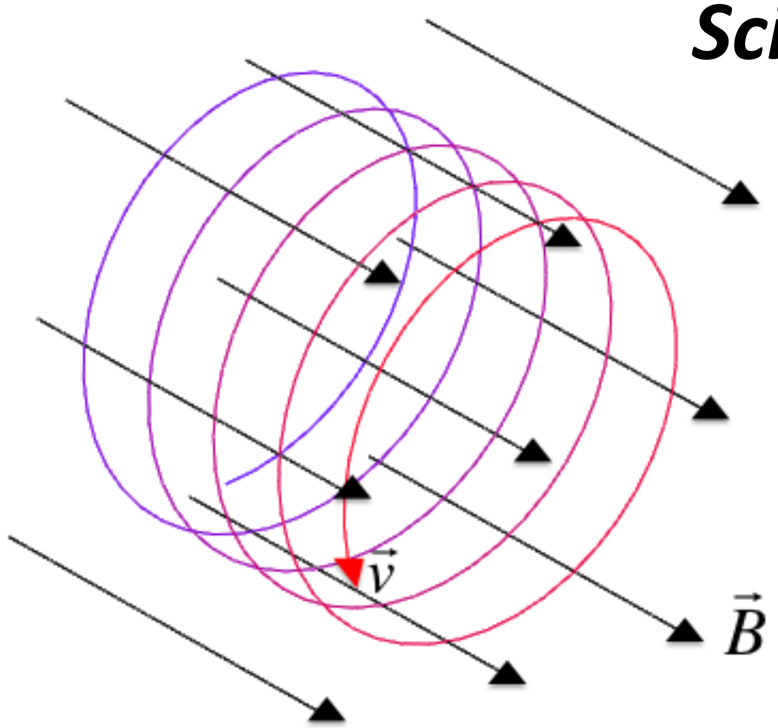
- Naively, the particle's trajectory is described by a single helix
- Forget it
  - Non-uniform B field
  - scattering
  - energy loss
  - …
- trajectory is only *locally helical*
- Kalman Filter allows us to take these effects into account, while preserving a locally smooth trajectory

Science Fiction

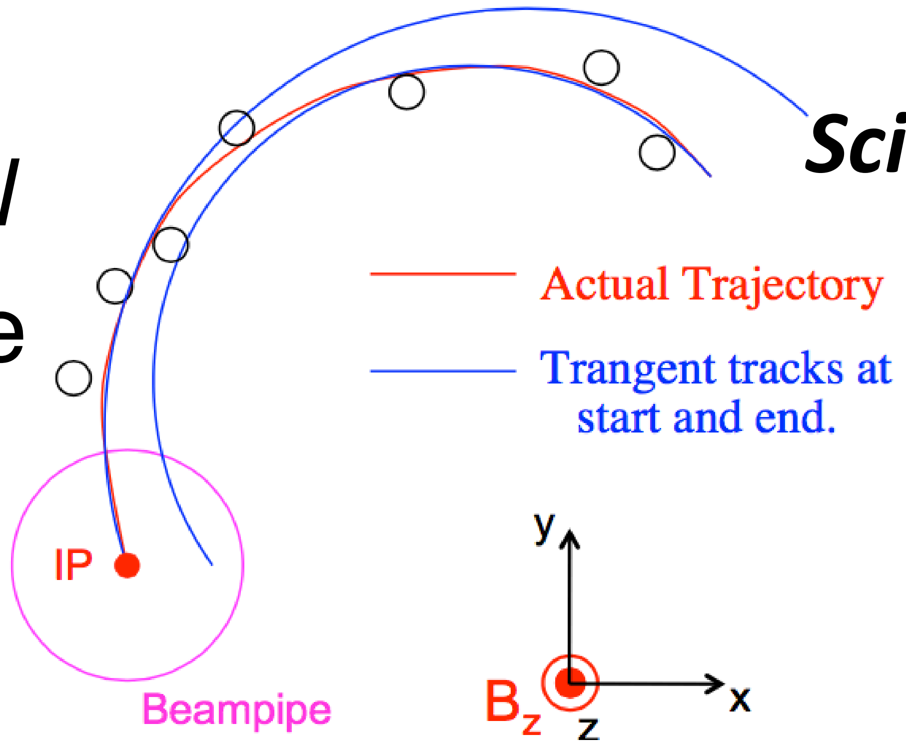


Why Kalman Filter for particle tracking?

- Naively, the particle's trajectory is described by a single helix
- Forget it
  - Non-uniform B field
  - scattering
  - energy loss
  - …
- trajectory is only *locally helical*
- Kalman Filter allows us to take these effects into account, while preserving a locally smooth trajectory



Science Fiction

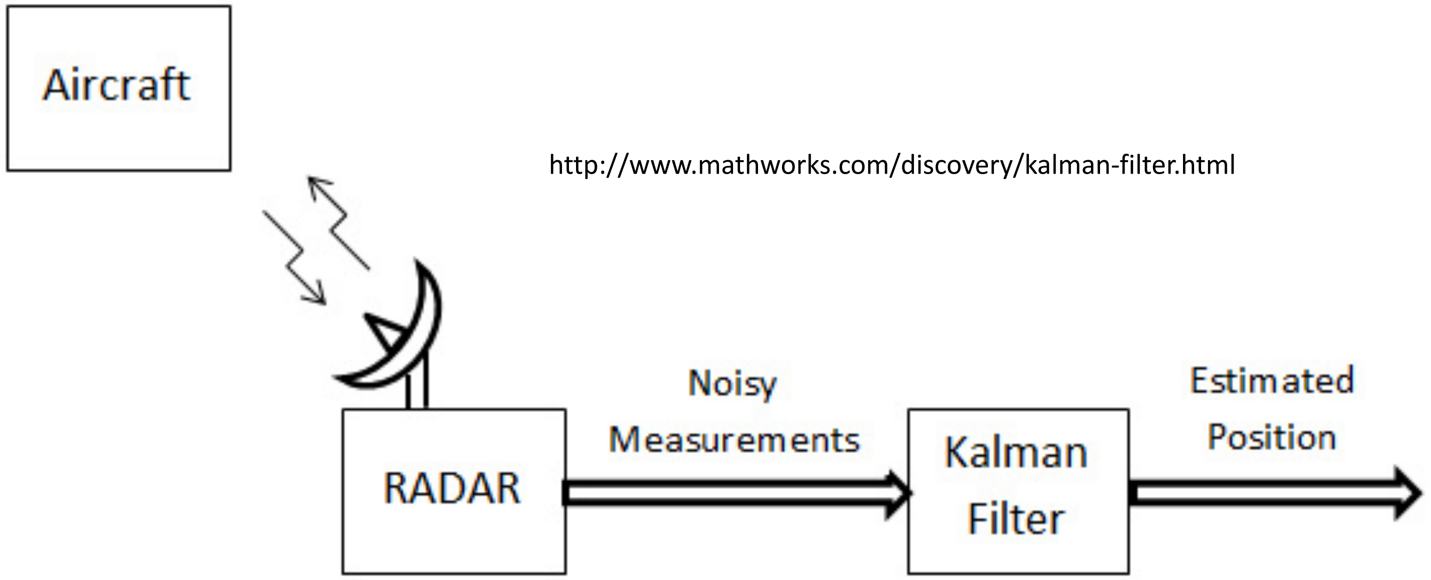


Science Fact

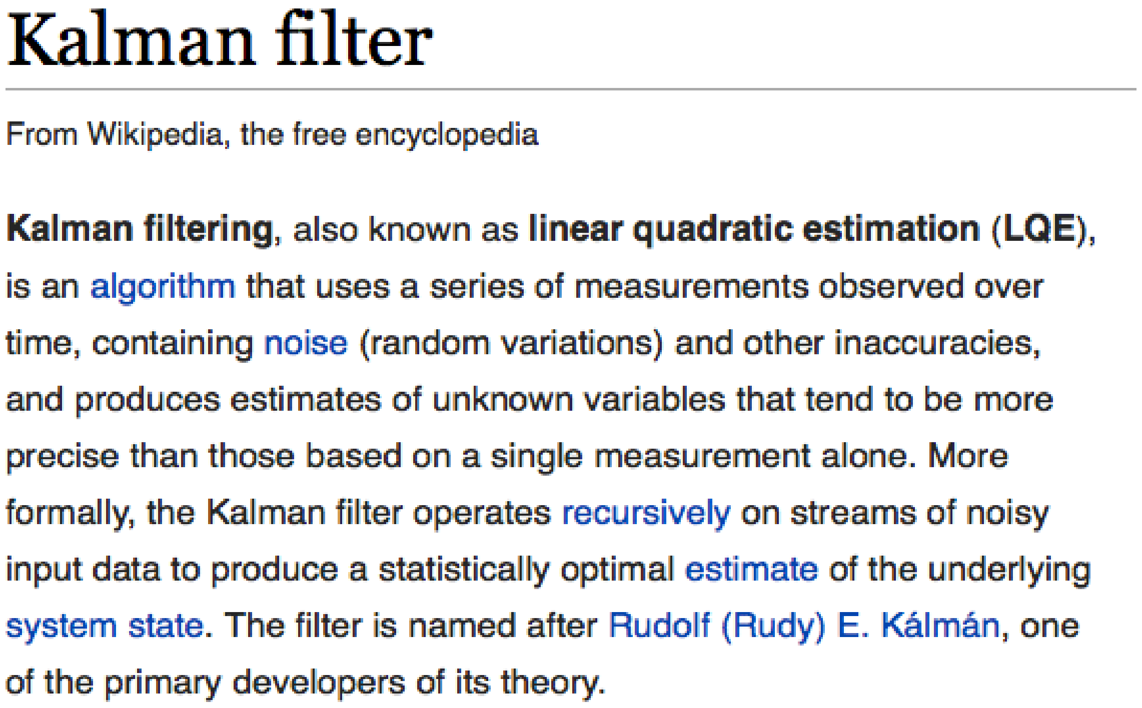
Rob Kutschke

Kalman Filter

- Method for obtaining best estimate of the five track parameters
- Natural way of including interactions in the material (<u>process noise</u>) and hit position uncertainty (<u>measurement error</u>)
- Used both in ***pattern recognition*** (i.e., determining which hits to group together as coming from one particle) and in ***fitting*** (i.e., to determine the ultimate track parameters)



http://www.mathworks.com/discovery/kalman-filter.html



Kalman filter

From Wikipedia, the free encyclopedia

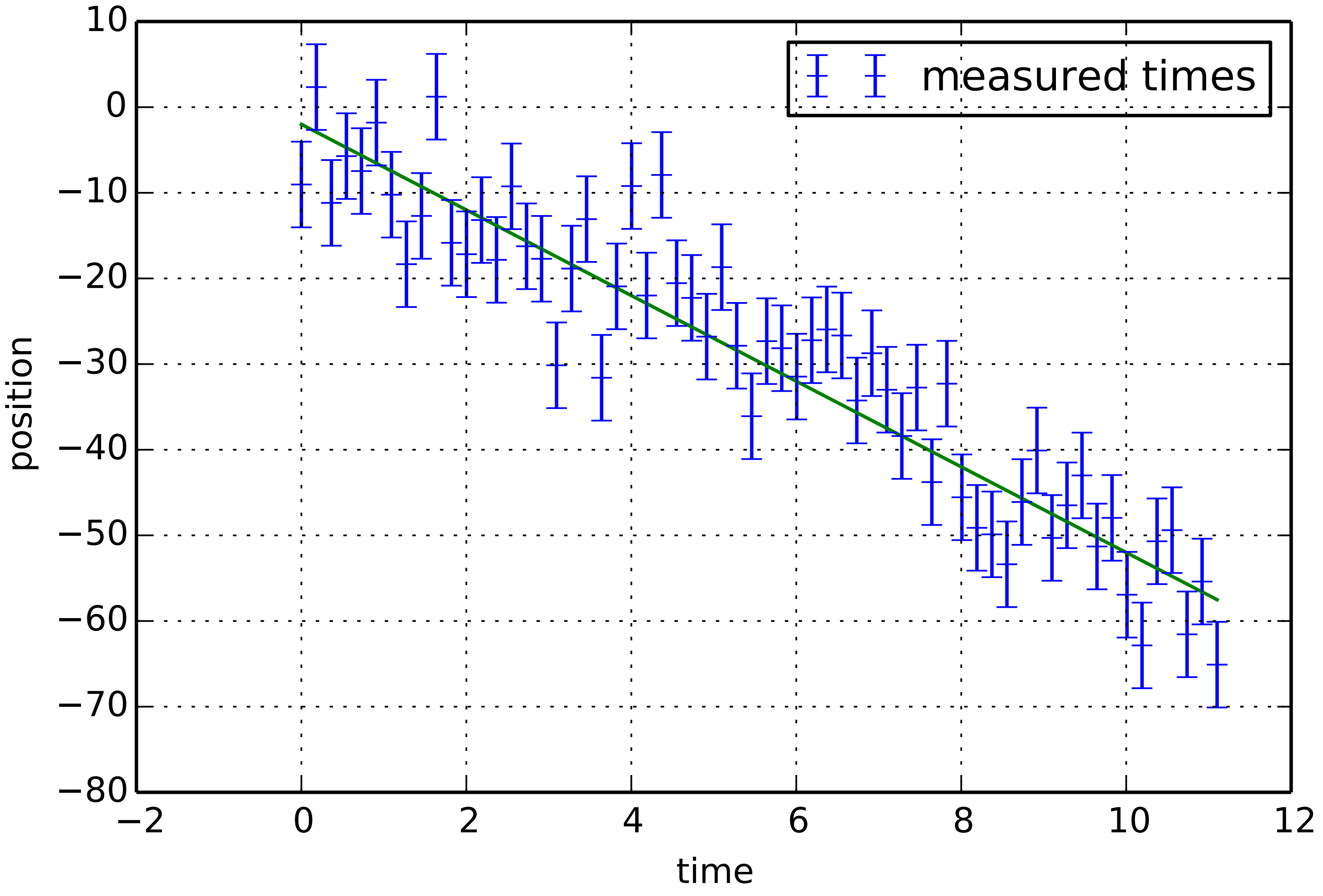
Kalman filtering, also known as **linear quadratic estimation** (**LQE**), is an algorithm that uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. More formally, the Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state. The filter is named after Rudolf (Rudy) E. Kálmán, one of the primary developers of its theory.

http://en.wikipedia.org/wiki/Kalman_filter

R. Frühwirth, *Nucl. Instr. Meth. A* **262**, 444 (1987), DOI:10.1016/0168-9002(87)90887-4

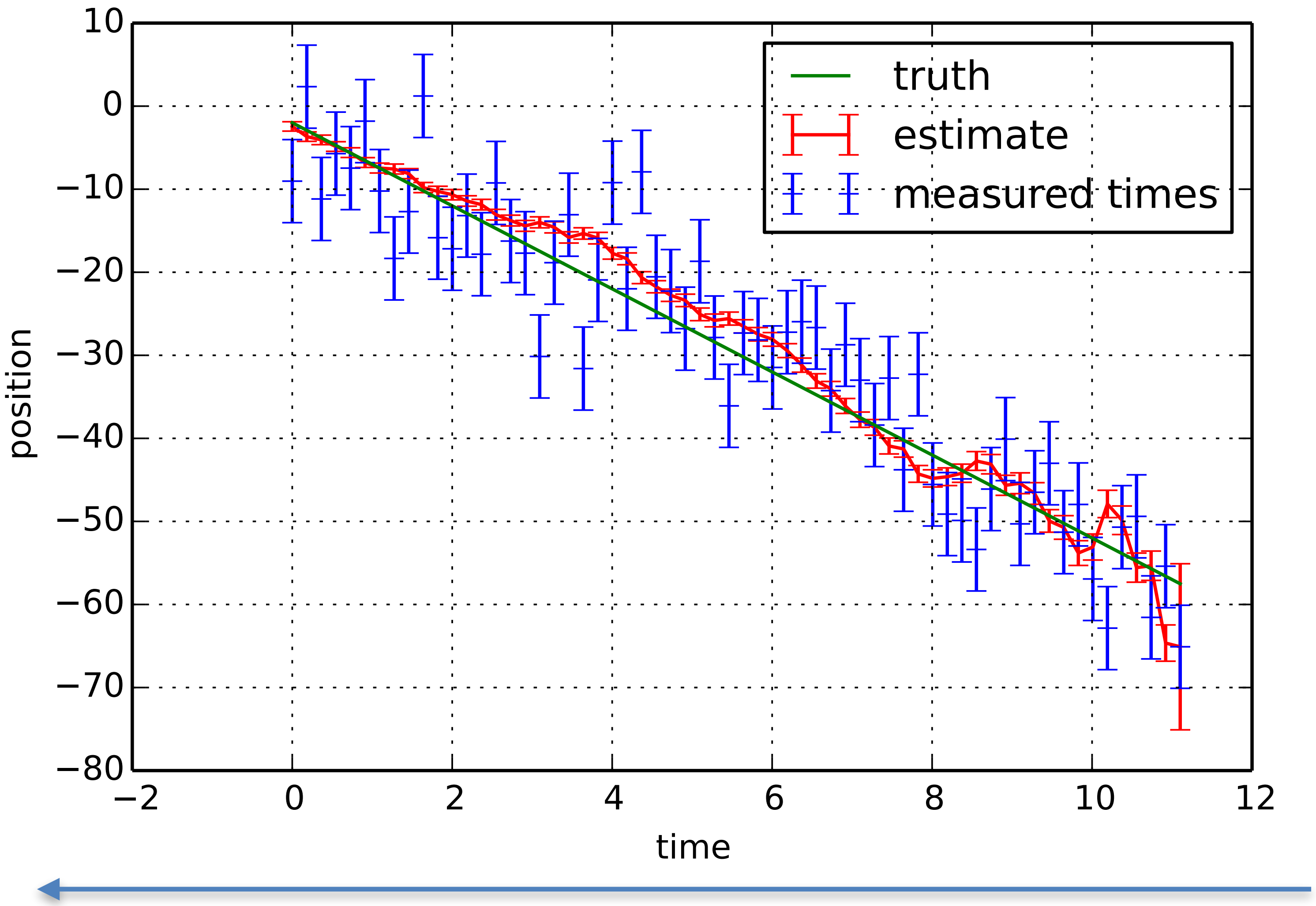
Kalman Example

Estimate for intercept and slope for noisy data



Kalman Example

Estimate for intercept and slope for noisy data

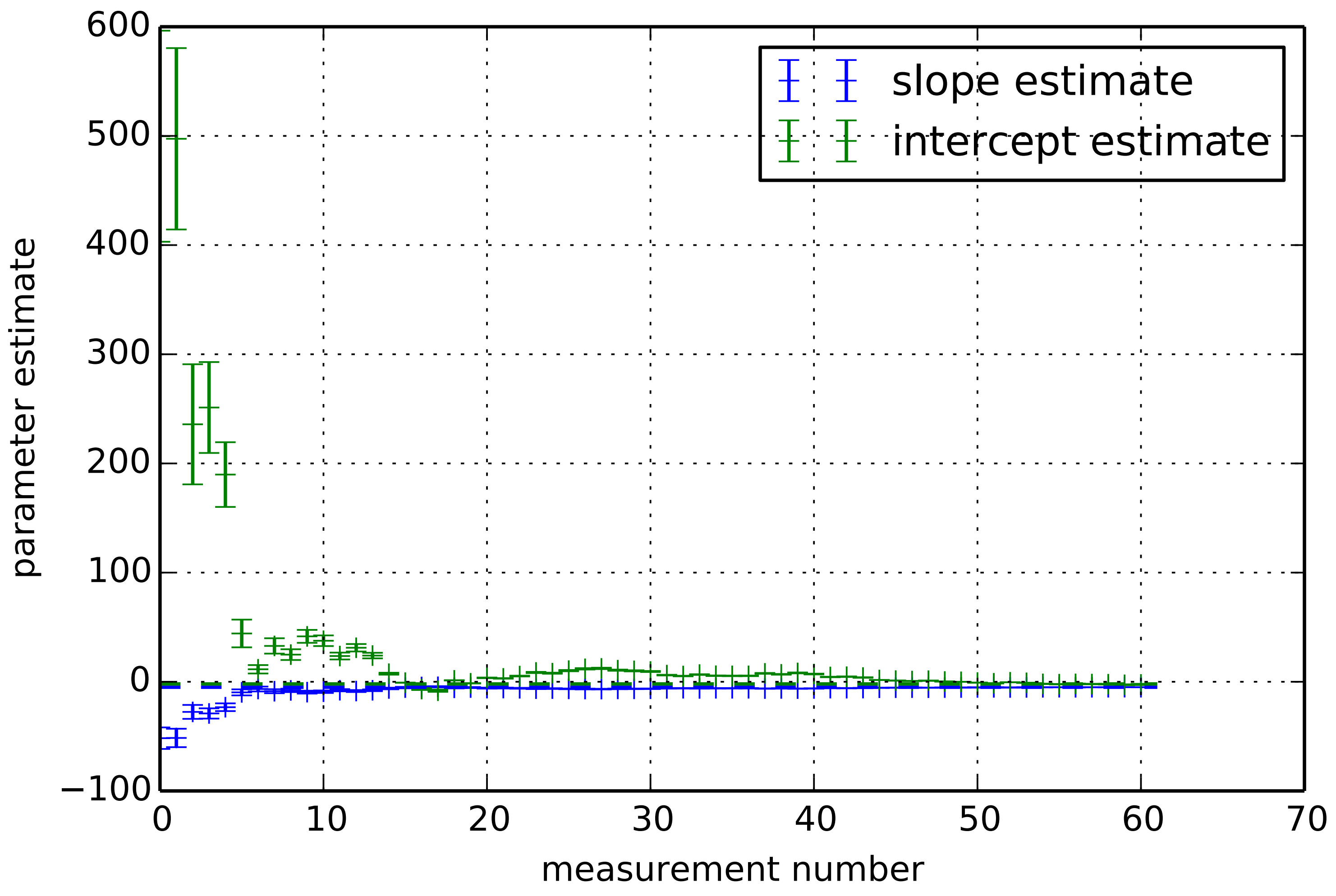


evaluation runs this way

Kalman Example

Estimate for intercept and slope for noisy data

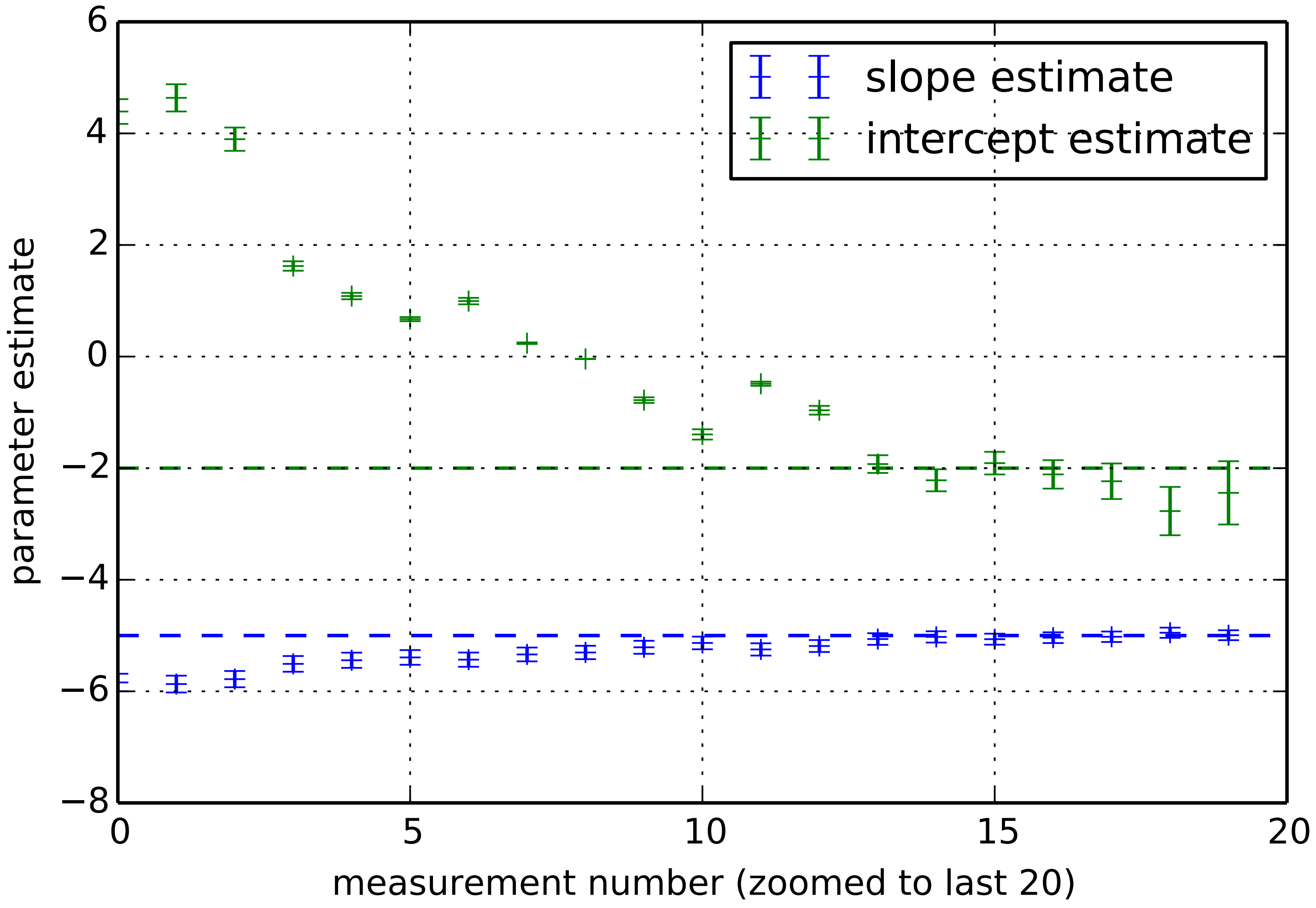
data series run the other way



Kalman Example

Estimate for intercept and slope for noisy data

data series run the other way



Algorithm overview

- ***Seeding***.
  - Select hits to get an *initial track candidate*.
  - Sort candidates by some criterion. Criteria that lead to higher quality tracks are tried first, then progressively less stringent criteria
  - Each hit can only be used once
- ***Main tracking loop*** over track candidates
  - Propagate each helix to next detector layer, taking into account the uncertainty of the current estimate and the amount of material in the way
  - Look for hits in the next layer consistent with the current candidate track
    - update the track parameters to include new hit (Kalman)
  - Remove hits from list of available hits
  - Repeat Step 2 until iterated over all layers, removing used hits and updating track parameters as we go along
  - Once all hits are attached re-fit final track parameters to get best estimate
- Return to ***Seeding*** step and generate new seeds on the remaining hits

Algorithm overview

- ***Seeding***.
  - Select hits to get an *initial track candidate*.
  - Sort candidates by some criterion. Criteria that lead to higher quality tracks are tried first, then progressively less stringent criteria
  - Each hit can only be used once
- ***Main tracking loop*** over track candidates
  - Propagate each helix to next detector layer, taking into account the uncertainty of the current estimate and the amount of material in the way
  - Look for hits in the next layer consistent with the current candidate track
    - update the track parameters to include new hit (Kalman)
  - Remove hits from list of available hits
  - Repeat Step 2 until iterated over all layers, removing used hits and updating track parameters as we go along
  - Once all hits are attached re-fit final track parameters to get best estimate
- Return to ***Seeding*** step and generate new seeds on the remaining hits

Algorithm overview

- ***Seeding***.
  - Select hits to get an *initial track candidate*.
  - Sort candidates by some criterion. Criteria that lead to higher quality tracks are tried first, then progressively less stringent criteria
  - Each hit can only be used once
- ***Main tracking loop*** over track candidates
  - Propagate each helix to next detector layer, taking into account the uncertainty of the current estimate and the amount of material in the way
  - Look for hits in the next layer consistent with the current candidate track
    - update the track parameters to include new hit (Kalman)
  - Remove hits from list of available hits
  - Repeat Step 2 until iterated over all layers, removing used hits and updating track parameters as we go along
  - Once all hits are attached re-fit final track parameters to get best estimate
- Return to ***Seeding*** step and generate new seeds on the remaining hits

Algorithmic overview

```
for all input hits do
    create seed tracks
    calculate initial track parameters for seed track
end for
for all seed tracks do
    create track candidate, track state from seed
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state for layer $i$
        look for hits to add to track candidate on this layer
        for all candidate hits to add  do
            create a new track candidate
            add hit to this track candidate
            remove hit from list of available hits
            update Kalman state with new hit for this track candidate
        end for
        Prune track candidate list
    end for
end for
for all Track candidates do
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
    for all $i = n, 0$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
end for
```

Algorithmic overview

```
for all input hits do
    create seed tracks
    calculate initial track parameters for seed track
end for
for all seed tracks do
    create track candidate, track state from seed
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state for layer $i$
        look for hits to add to track candidate on this layer
        for all candidate hits to add  do
            create a new track candidate
            add hit to this track candidate
            remove hit from list of available hits
            update Kalman state with new hit for this track candidate
        end for
        Prune track candidate list
    end for
end for
for all Track candidates do
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
    for all $i = n, 0$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
end for
```

Seeding

Algorithmic overview

Seeding

```
for all input hits do
    create seed tracks
    calculate initial track parameters for seed track
end for
```

Building

```
for all seed tracks do
    create track candidate, track state from seed
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state for layer $i$
        look for hits to add to track candidate on this layer
        for all candidate hits to add do
            create a new track candidate
            add hit to this track candidate
            remove hit from list of available hits
            update Kalman state with new hit for this track candidate
        end for
        Prune track candidate list
    end for
end for
```

```
for all Track candidates do
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
    for all $i = n, 0$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
end for
```

Algorithmic overview

Seeding

```
for all input hits do
  create seed tracks
  calculate initial track parameters for seed track
end for
```

Building

```
for all seed tracks do
  create track candidate, track state from seed
  for all $i = 0, n$ in detector layers do
    propagate track to layer $i$
    update uncertainty and Kalman state for layer $i$
    look for hits to add to track candidate on this layer
    for all candidate hits to add  do
      create a new track candidate
      add hit to this track candidate
      remove hit from list of available hits
      update Kalman state with new hit for this track candidate
    end for
    Prune track candidate list
  end for
end for
```

Fitting (and smoothing)

```
for all Track candidates do
  for all $i = 0, n$ in detector layers do
    propagate track to layer $i$
    update uncertainty and Kalman state
  end for
  for all $i = n, 0$ in detector layers do
    propagate track to layer $i$
    update uncertainty and Kalman state
  end for
end for
```

Algorithmic overview

```
for all input hits do
    create seed tracks
    calculate initial track parameters for seed track
end for
```

Seeding

```
for all seed tracks do
    create track candidate, track state from seed
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state for layer $i$
        look for hits to add to track candidate on this layer
        for all candidate hits to add do
            create a new track candidate
            add hit to this track candidate
            remove hit from list of available hits
            update Kalman state with new hit for this track candidate
        end for
        Prune track candidate list
    end for
end for
```

Building

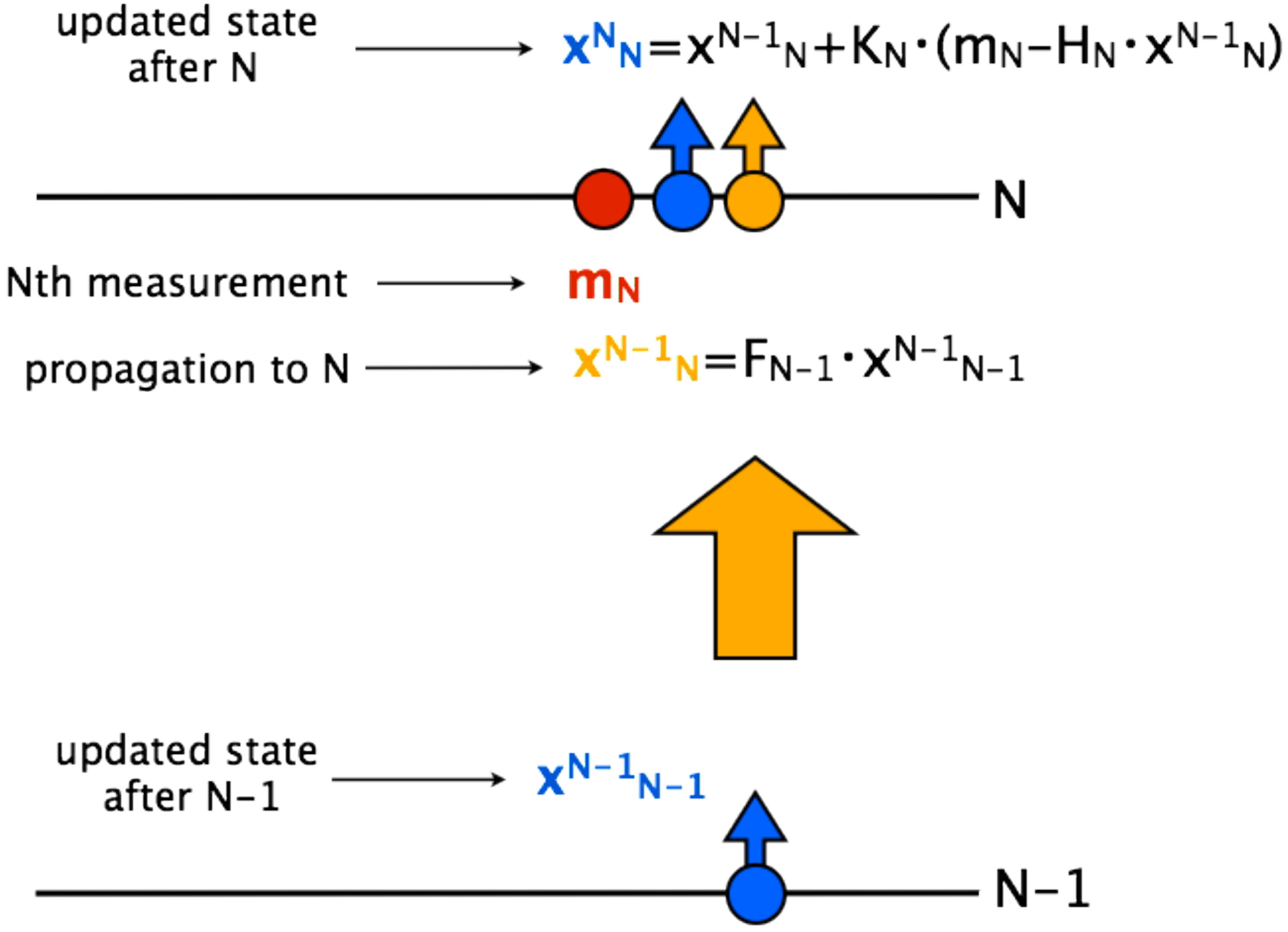
could be 0, 1, many hits

```
for all Track candidates do
    for all $i = 0, n$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
    for all $i = n, 0$ in detector layers do
        propagate track to layer $i$
        update uncertainty and Kalman state
    end for
end for
```

Fitting (and smoothing)

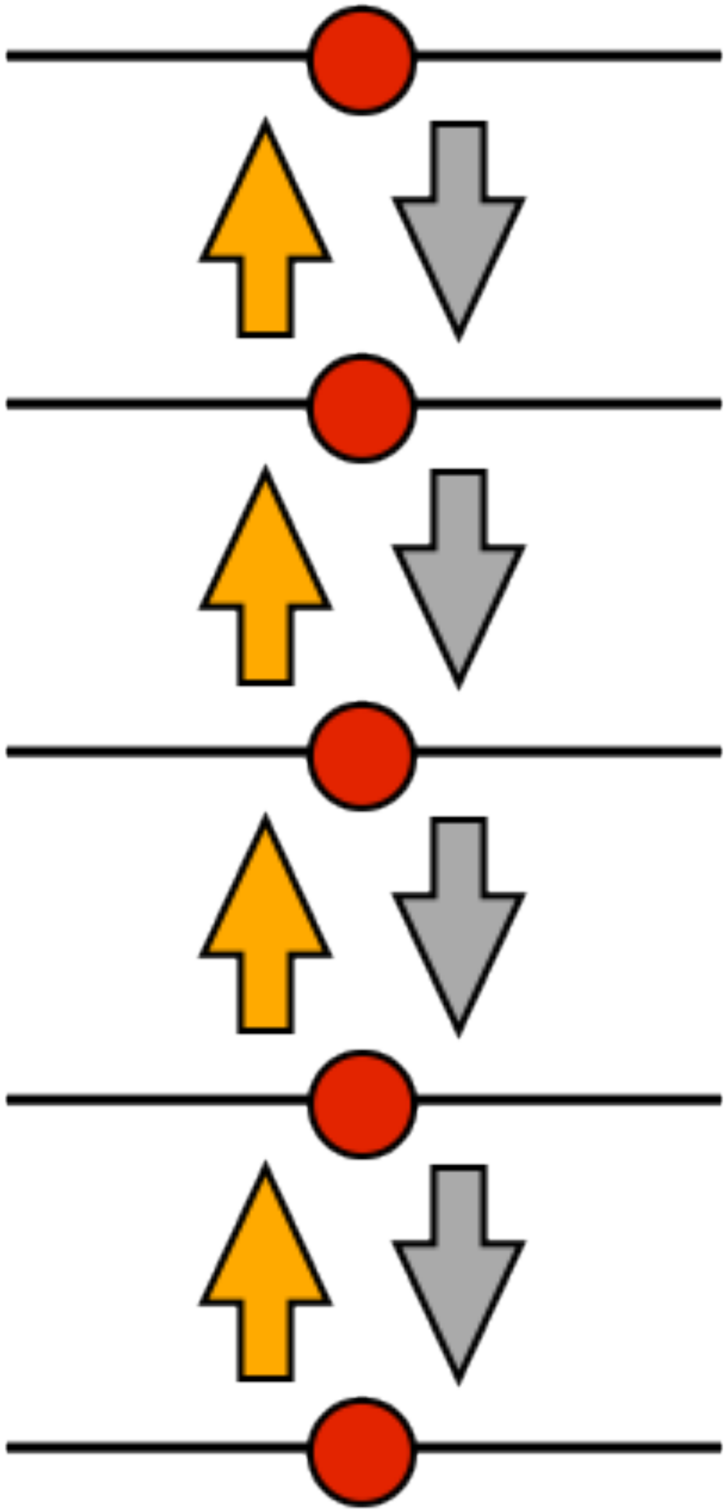
Tracking as Kalman Filter

- KF track reconstruction can be divided into 2 main steps: ***building, and fitting***.
- Both track building and track fitting are based on Kalman Filter.



The Kalman Filter is an ***iterative procedure*** of a basic logic unit consisting of the ***propagation*** of parameters and uncertainties (track state) from a layer to the next one, where the track state is ***updated (filtered)*** with the hit measurement information.

Track Fitting as Kalman Filter

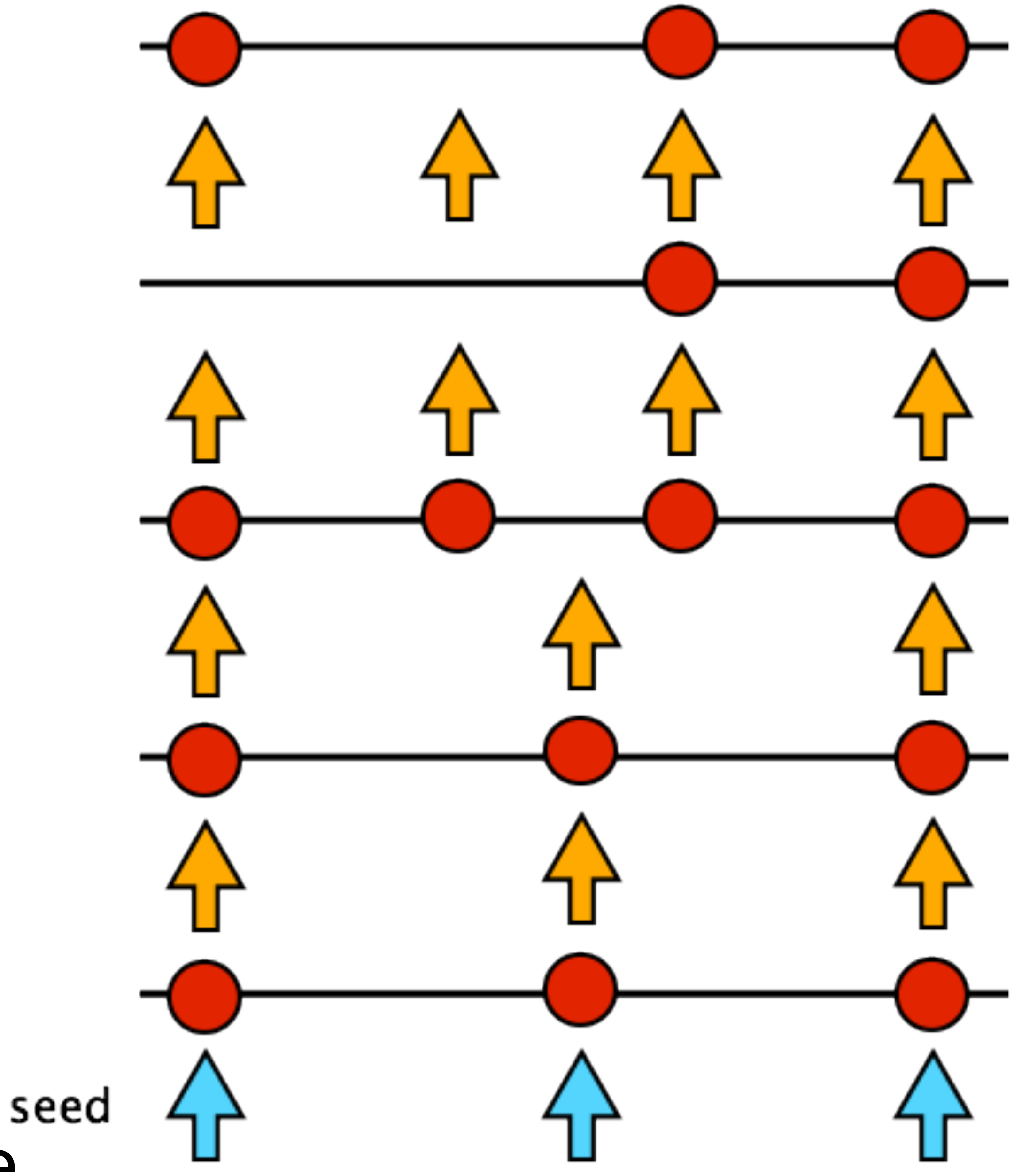


- The track fit consists of the ***simple repetition of the basic logic unit*** for all the pre-determined track hits
- It is divided in two steps
  - a forward fit
  - a backward smoothing stage
- Forward fit: best estimate at interaction point
- Smoothing stage: best estimate at face of calorimeter

- Computationally, the Kalman filter is a set of ***matrix operations*** with small matrices (dimension 6 or less)

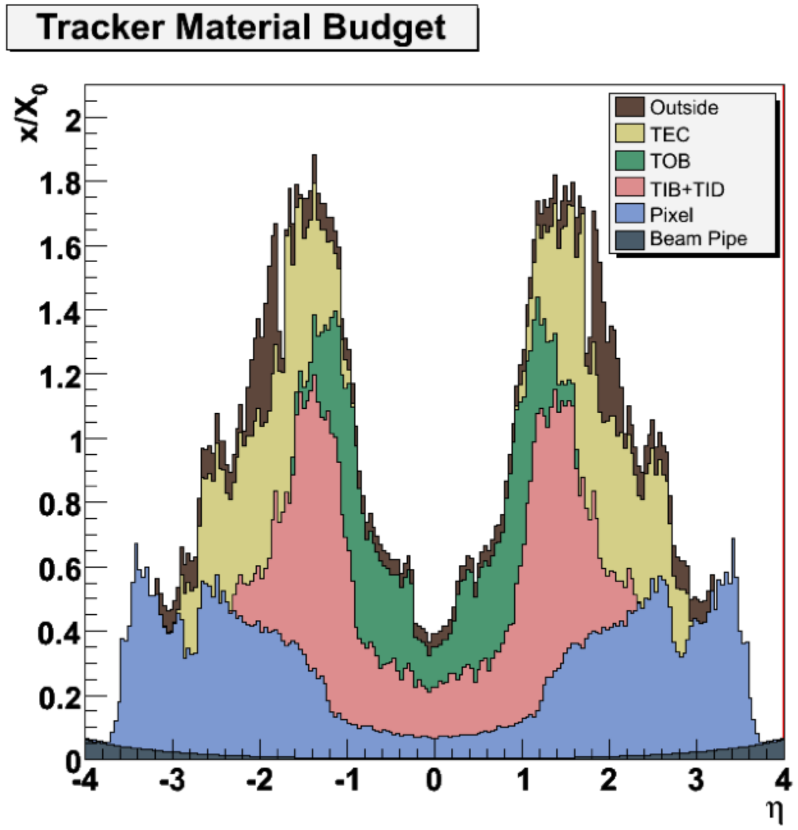
Track Building

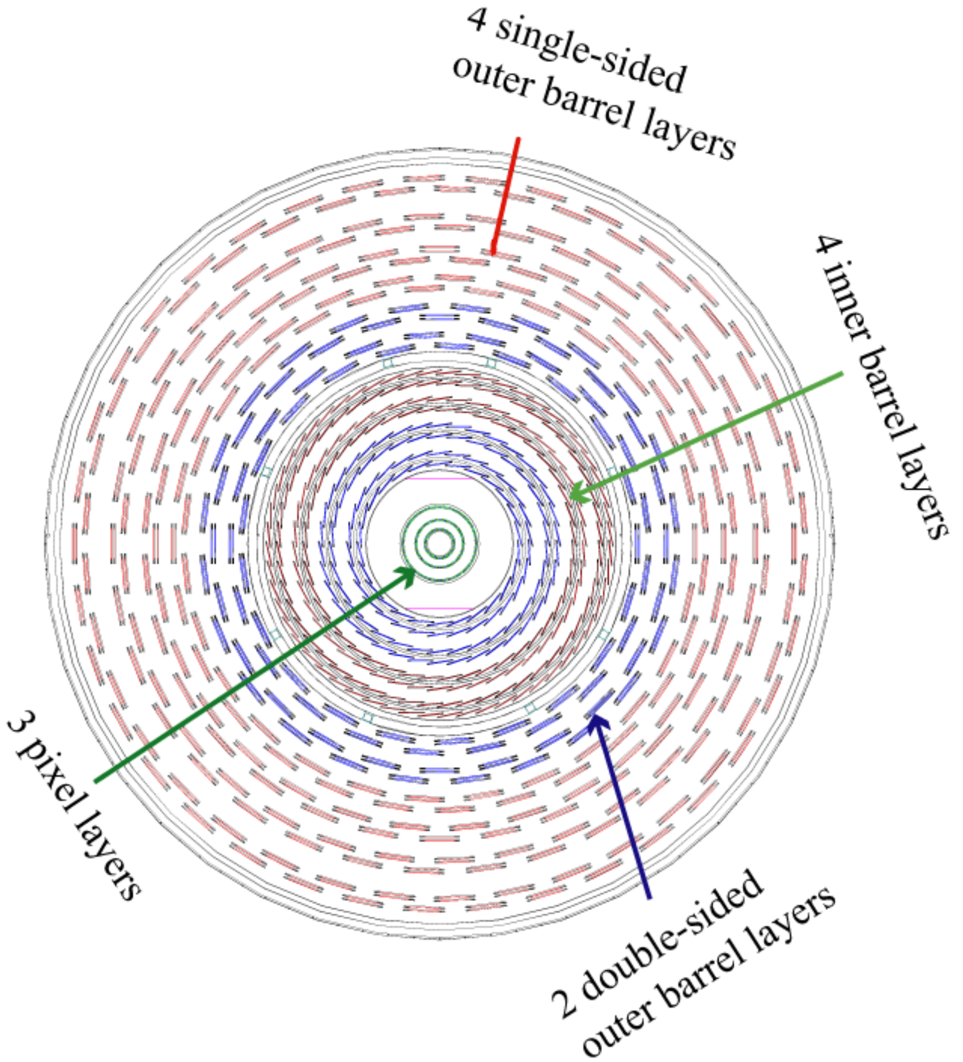
- Track building adds complexity to the problem.
- When moving to the next layer, hits are searched for within a compatibility window.
- The track candidate needs to ***branch*** in case of multiple matches and the algorithm needs to be robust against missing/outlier hits.
- Track Building is by far the ***most time consuming step*** in the whole event reconstruction
  - specific design choices have to be made to boost its performance on the coprocessor.
- Preliminary tests with 500 tracks/event show hit finding efficiency close to 100%



Full algorithm complications

- Geometry is complicated and has no symmetries, even before accounting for alignment (diff btw ideal and real geometry)
  - no “circular cows”
- Material maps are complicated, big, and not negligible
  - Algorithm uses full information about material map to estimate multiple scattering, radiation, esp for electrons
- Hits can only be used once
  - synchronization

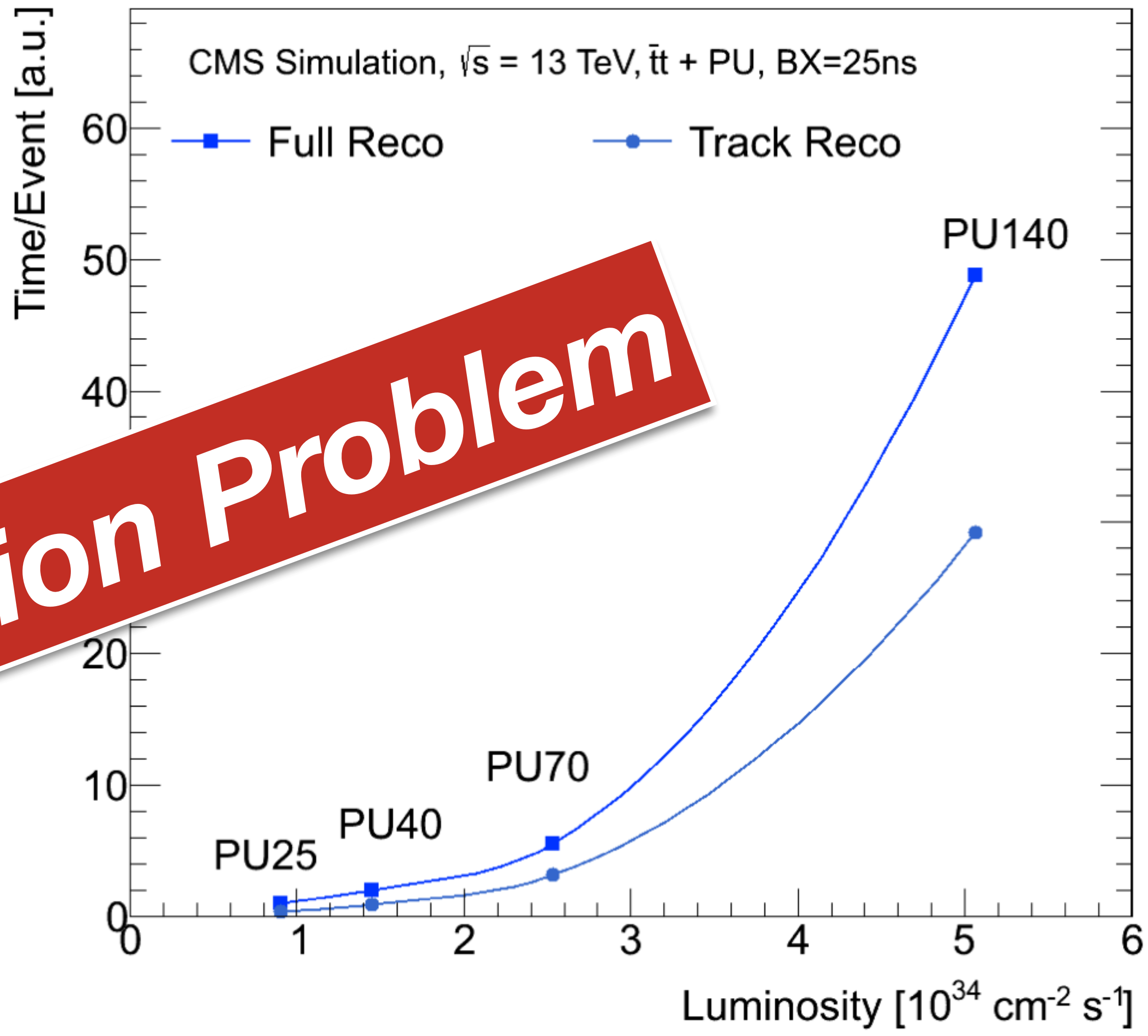




Reconstruction Redux

- Going from detector primitives (energy deposits in various elements) to particles: “reconstruction”
- Tracking is the most time-intensive part of reconstru combining the hits i to form the t r the char
- O easurement stations per event, across many layers
- Can we make the tracking algorithm concurrent and speed up the reconstruction?

Back to Reconstruction Problem

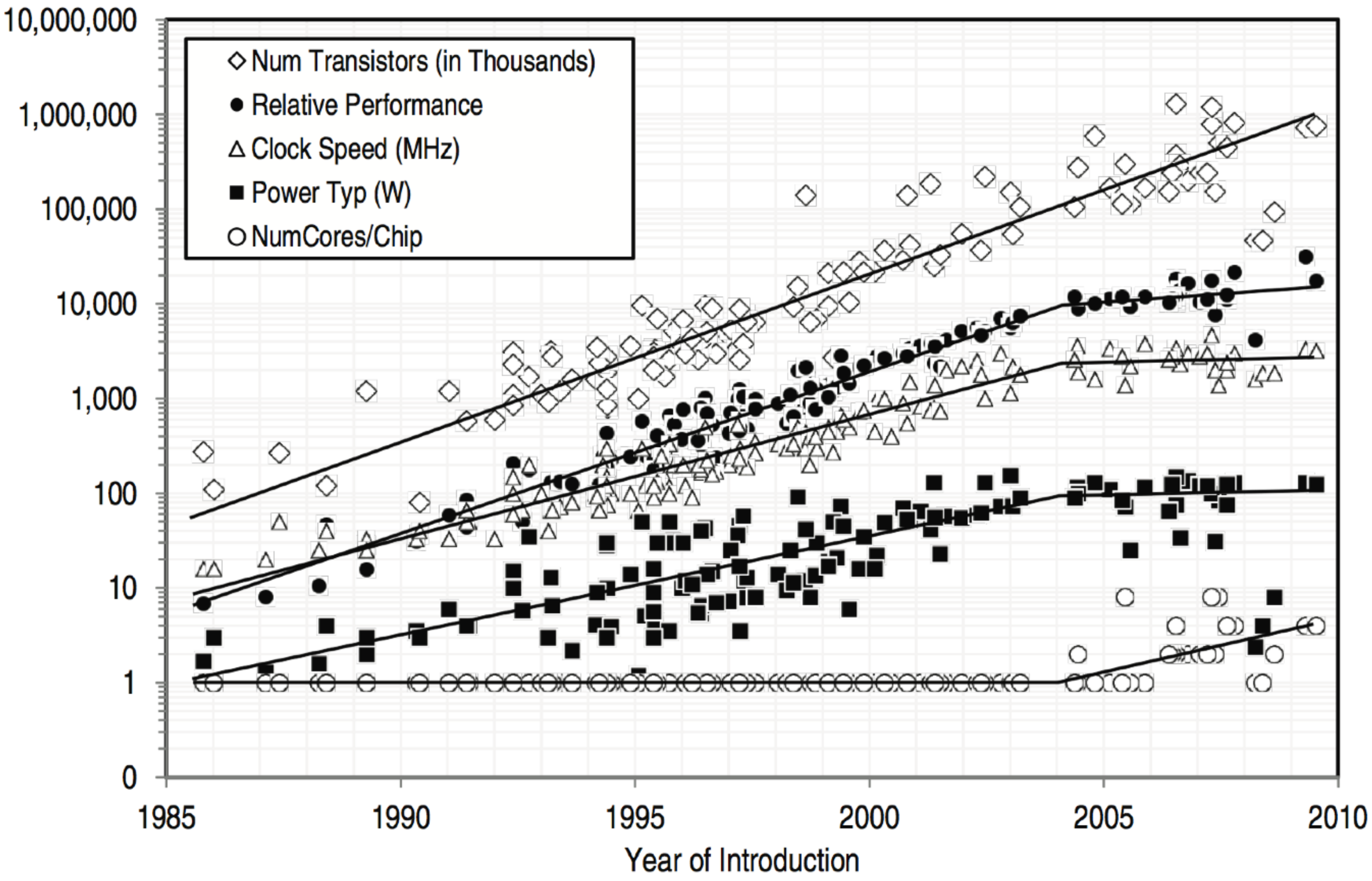


Is Moore's law dead?

- Moore's law: ***transistor count doubles every 18 months***
- For a long time implied increases in clock frequency
  - not actually what Moore said!
  - but it was really nice …
- Moore's law continues but clock frequencies stalled at <4 GHz
  - heat dissipation and power requirements now drive CPU industry
    - smart phones
  - A consequence: can no longer just wait for our code to run faster
  - Laziness is no longer an option

- How does Moore's law look today?
  - New transistors go into multi-core, many-core, SOC and other devices
  - Need to learn how to use these devices to attack the physics problems we are interested in
- Parallelization and vectorization is the key

Different views on Moore's Law

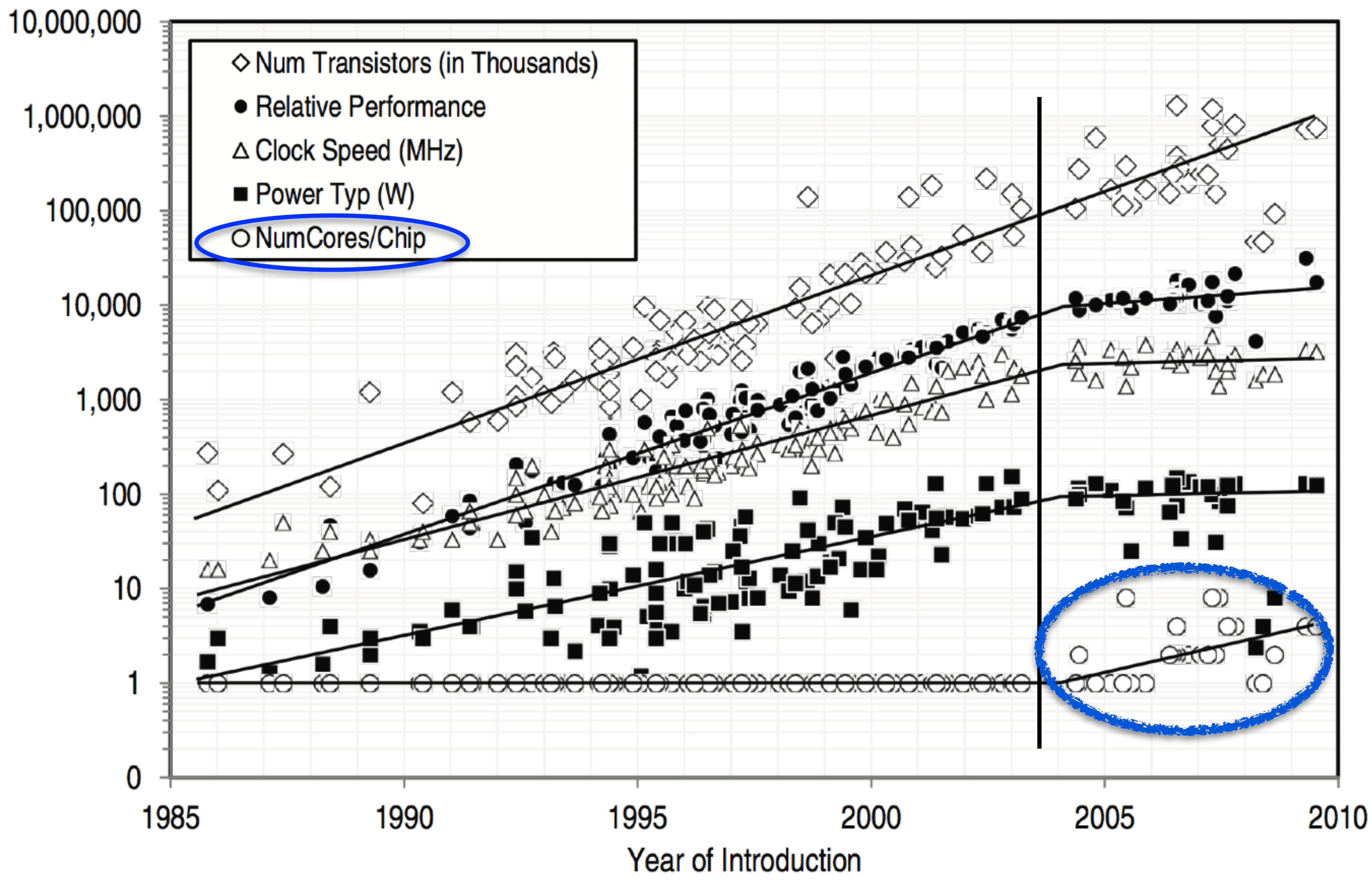


Committee on Sustaining Growth in Computing Performance, National Research Council.
"What Is Computer Performance?"
In The Future of Computing Performance: Game Over or Next Level?
Washington, DC: The National Academies Press, 2011.

doi:10.17226/12980

Different views on Moore’s Law

discontinuity in ~2004

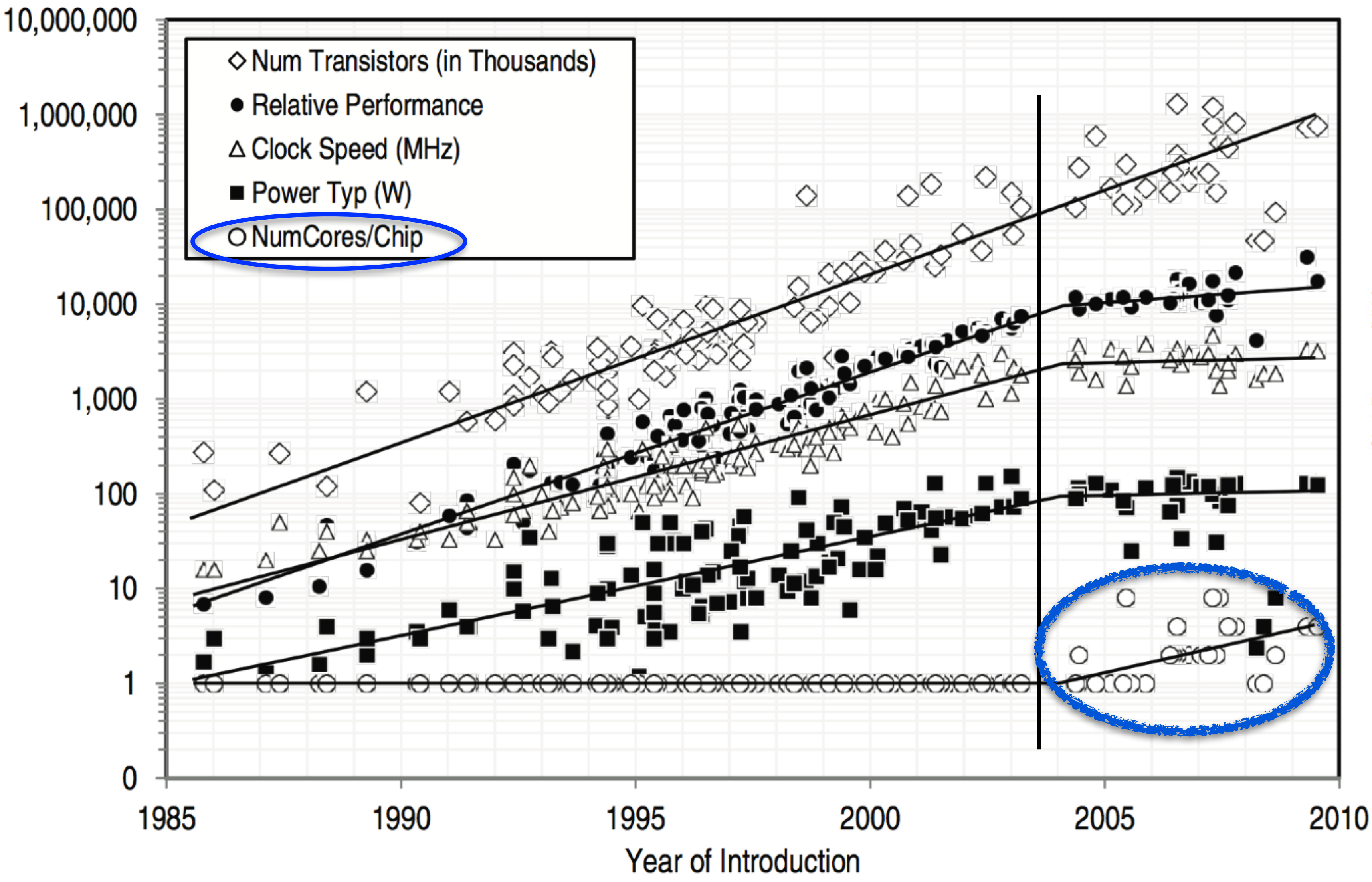


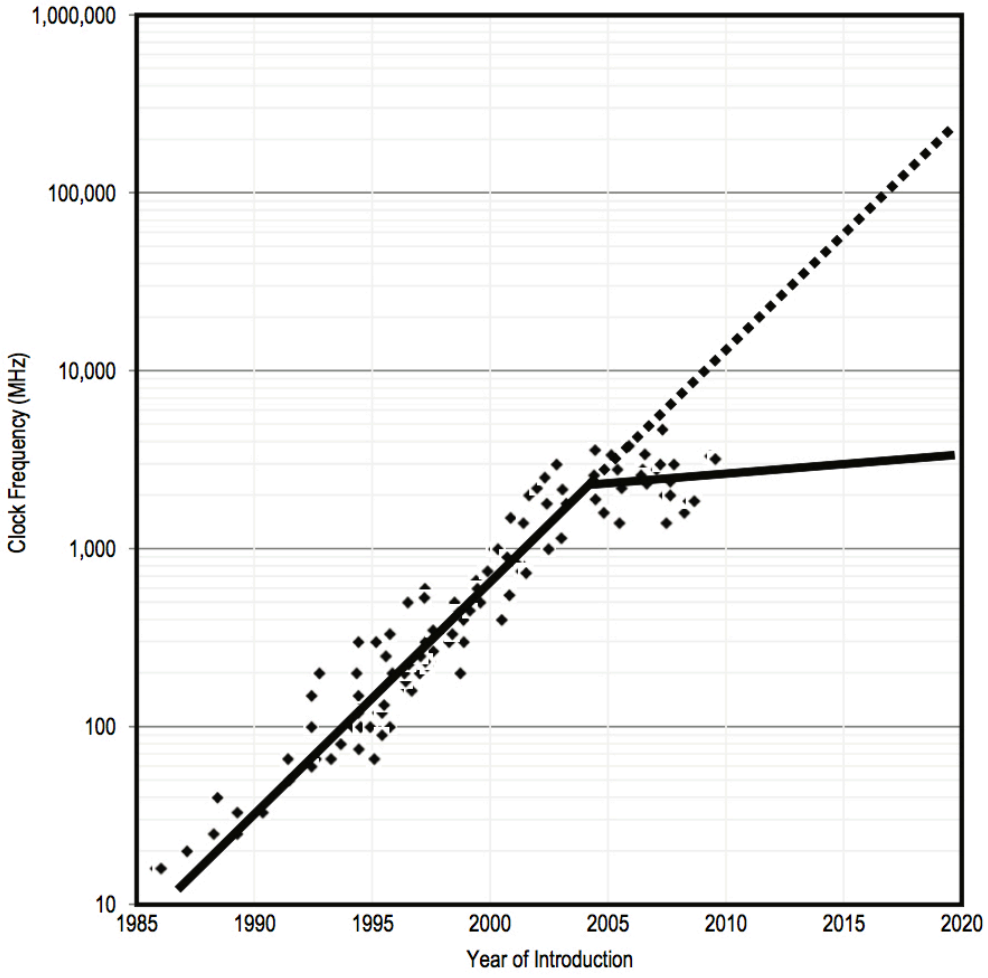
Committee on Sustaining Growth in Computing Performance, National Research Council.
"What Is Computer Performance?"
In The Future of Computing Performance: Game Over or Next Level?
Washington, DC: The National Academies Press, 2011.

doi:10.17226/12980

Different views on Moore’s Law

discontinuity in ~2004





Committee on Sustaining Growth in Computing Performance, National Research Council.
"What Is Computer Performance?"
In The Future of Computing Performance: Game Over or Next Level?
Washington, DC: The National Academies Press, 2011.

doi:10.17226/12980

Selected Parallel Architectures

	Xeon E5-2620	Xeon Phi 7120P	Tesla K20m	Tesla K40
Cores	6 x 2	61	13	12
Logical Cores	12 x 2	244	2496 CUDA cores	2880
Max clock rate	2.5 GHz	1.333 GHz	706 MHz	745 MHz
GFLOPS (double)	120	1208	1170	1430
SIMD width	64 bytes	128 bytes	Warp of 32	Warp of 32
Memory	~64-384 GB	16 GB	5 GB	12 GB
Memory B/W	42.6 GB/s	352 GB/s	208 GB/s	288 GB/s

Xeon — CPU
Xeon Phi — Many integrated cores
Tesla — GPU

Selected Parallel Architectures

	Xeon E5-2620	Xeon Phi 7120P	Tesla K20m	Tesla K40
Cores	6 x 2	61	13	12
Logical Cores	12 x 2	244	2496 CUDA cores	2880
Max clock rate	2.5 GHz	1.333 GHz	706 MHz	745 MHz
GFLOPS (double)	120	1208	1170	1430
SIMD width	64 bytes	128 bytes	Warp of 32	Warp of 32
Memory	~64-384 GB	16 GB	5 GB	12 GB
Memory B/W	42.6 GB/s	352 GB/s	208 GB/s	288 GB/s

Xeon — CPU
Xeon Phi — Many integrated cores
Tesla — GPU

Selected Parallel Architectures

	Xeon E5-2620	Xeon Phi 7120P	Tesla K20m	Tesla K40
Cores	6 x 2	61	13	12
Logical Cores	12 x 2	244	2496 CUDA cores	2880
Max clock rate	2.5 GHz	1.333 GHz	706 MHz	745 MHz
GFLOPS (double)	120	1208	1170	1430
SIMD width	64 bytes	128 bytes	Warp of 32	Warp of 32
Memory	~64-384 GB	16 GB	5 GB	12 GB
Memory B/W	42.6 GB/s	352 GB/s	208 GB/s	288 GB/s

Xeon — CPU
Xeon Phi — Many integrated cores
Tesla — GPU

Selected Parallel Architectures

	Xeon E5-2620	Xeon Phi 7120P	Tesla K20m	Tesla K40
Cores	6 x 2	61	13	12
Logical Cores	12 x 2	244	2496 CUDA cores	2880
Max clock rate	2.5 GHz	1.333 GHz	706 MHz	745 MHz
GFLOPS (double)	120	1208	1170	1430
SIMD width	64 bytes	128 bytes	Warp of 32	Warp of 32
Memory	~64-384 GB	16 GB	5 GB	12 GB
Memory B/W	42.6 GB/s	352 GB/s	208 GB/s	288 GB/s

Xeon — CPU
Xeon Phi — Many integrated cores
Tesla — GPU

Sample Supercomputer:

TACC Stampede~10 Petaflop/s

- 2+ petaflop/s of Intel Xeon E5
- 7+ additional petaflop/s of Intel Xeon Phi™ SE10P coprocessors
- Follows the hardware trend of the last 10 years: processors gain cores (execution engines) rather than clock speed
- So is Moore's Law dead? No!
  - Transistor densities are still doubling every 2 years
  - Clock rates have stalled at < 4 GHz due to power consumption
  - Only way to increase flop/s/watt is through greater on-die parallelism
- Architectures are therefore moving from multi-core to many-core

Photo by TACC, June 2012

Many-Core Elements in Petaflop/s Machines

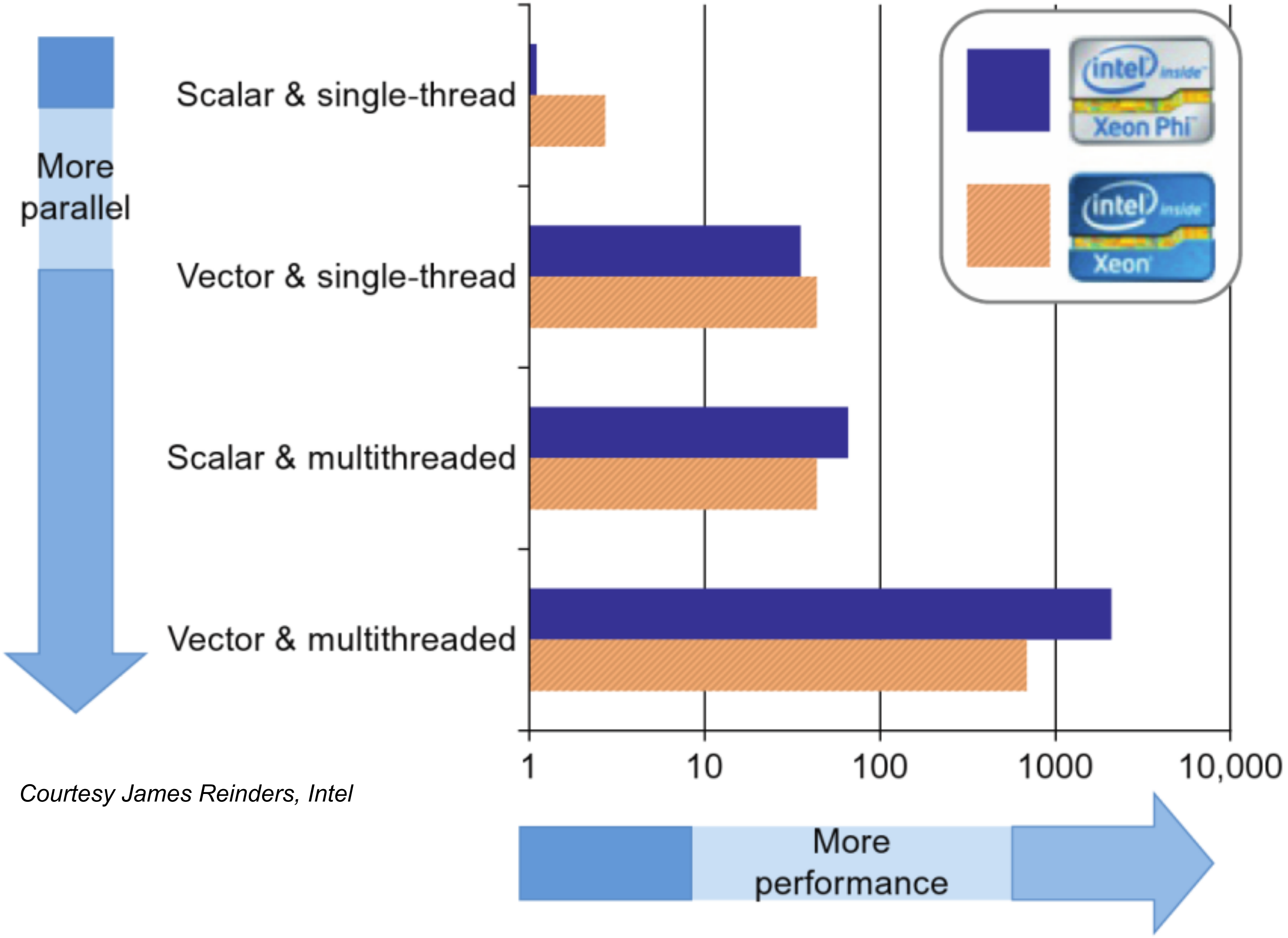
- **CPUs**: Wider vector units, more cores
  - AVX instructions crunch 8 or 16 floats at a time
  - Single thread runs well; dozens are needed
  - Stampede example: peak DP, dual Xeon E5-2680 - 0.34 Tflop/s, 260W
- **MICs**: 60+ CPU cores, floating-point efficiency
  - Slow clock, yet high flop/s from more/wider vectors, more cores
  - Intel compiler handles vectorization and multithreading code
  - Stampede example: peak DP, Xeon Phi SE10P - 1.06 Tflops/s, 300W
  - Next generation “Knight’s Landing” (KNL): ~3 Tflop/s, ~300W
- **GPUs**: 1000s of simple stream processors
  - Single Instruction, Multiple Thread (SIMT): think vector units, not cores
  - Special APIs are required: CUDA, OpenCL, OpenACC
  - Stampede example: peak DP, NVIDIA Tesla K20 - 1.17 Tflop/s, 225W

Xeon Phi vs. Xeon

	SE10P	Xeon E5	Xeon Phi is…
Number of cores	61	8	much higher
Clock speed (GHz)	1.01	2.7	lower
SIMD width (bits)	512	256	higher
DP Gflop/s/core	16+	21+	lower
HW threads/core	4	1*	higher

- Xeon designed for all workloads, high single-thread performance
- Xeon Phi also general purpose, but optimized for number crunching
  - High aggregate throughput via lots of weaker threads, more SIMD
  - Possible to achieve >2x performance compared to dual E5 CPUs

Parallelism and Performance on Xeon Phi vs. Xeon



Courtesy James Reinders, Intel

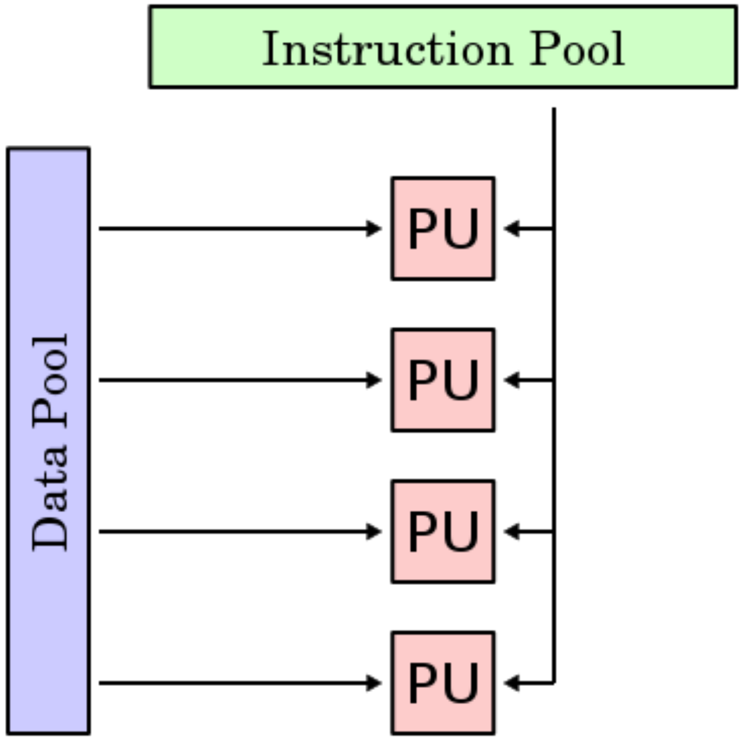
Only upon using many parallel resources does a Phi-like platform start being more performant than a traditional CPU

Challenges to Parallel Processing in KF tracking

- KF tracking cannot be ported in straightforward way to run in parallel
- Need to exploit two types of parallelism with parallel architectures

- **Vectorization**
  - Perform the same operation at the same time in lock-step across different data
  - **Challenge: branching** in track building - exploration of multiple track candidates per seed

- **Parallelization**
  - Perform different tasks at the same time on different pieces of data
  - **Challenge: thread balancing** – splitting the workload evenly is difficult as track occupancy in the detector not uniform on a per event basis



Vectorization

Another take

- Threading (task parallelism)
  - OpenMP, Cilk Plus, TBB, Pthreads, CUDA kernels, etc.
  - It’s all about sharing work and scheduling
- Vectorization (data parallelism)
  - “Lock step” Instruction Level Parallelization (SIMD)
  - Requires management of synchronized instruction execution
  - It’s all about finding simultaneous operations
- To utilize advanced architectures fully, both types of parallelism need to be identified and exploited
  - Need 2–4+ threads to keep a core busy (in-order execution stalls)
  - Vectorized loops gain 8x or 16x performance on MIC!
  - Important for CPUs as well: gain of 4x or 8x on Sandy Bridge

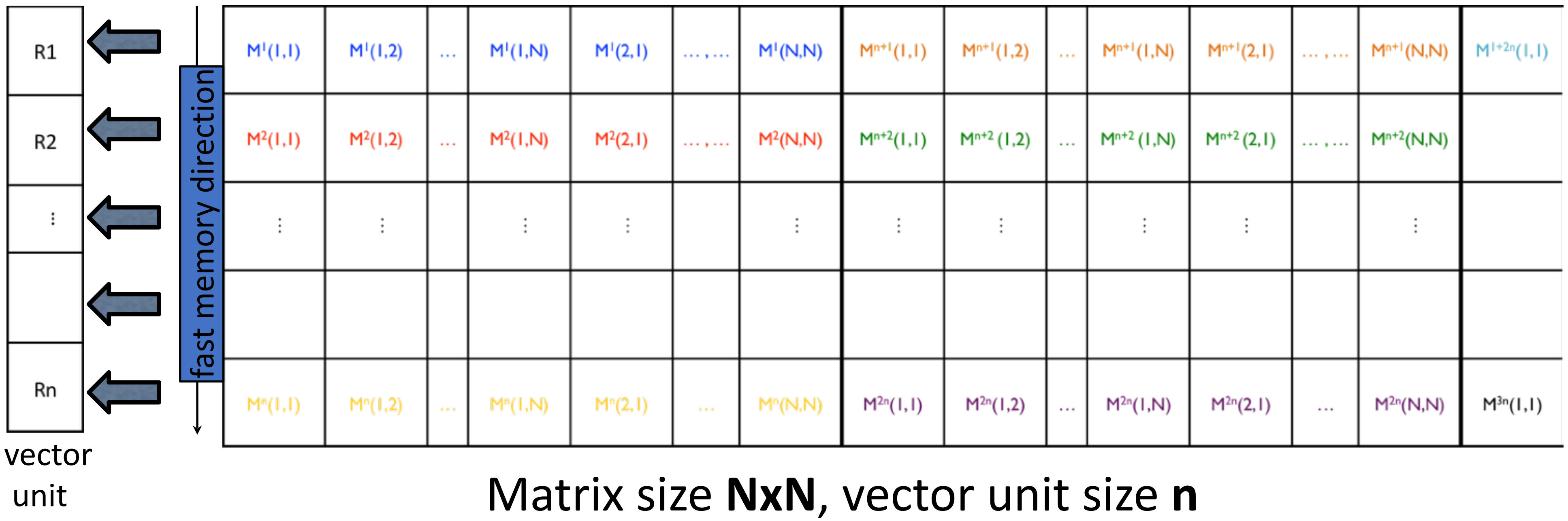
Strategy for track building & fitting

- **Vectorization** via Matriplex library
  - all Kalman operations (matrix operations) involve this library to use vector registers
- **Parallelization** using TBB
  - different threads handle groups of seeds (building) or groups of tracks (fitting)

Custom tool: Matriplex

- Matrix operations of KF **ideal for vectorized processing:** however, requires **synchronization** of operations
- Most matrix libraries are for large matrices (ours are small)

- Arrange data in such a way that it can loaded into the vector units of Xeon and Xeon Phi with ***Matriplex***
  - Fill vector units with the same matrix element from different matrices: ***n* matrices working in sync on same operation**



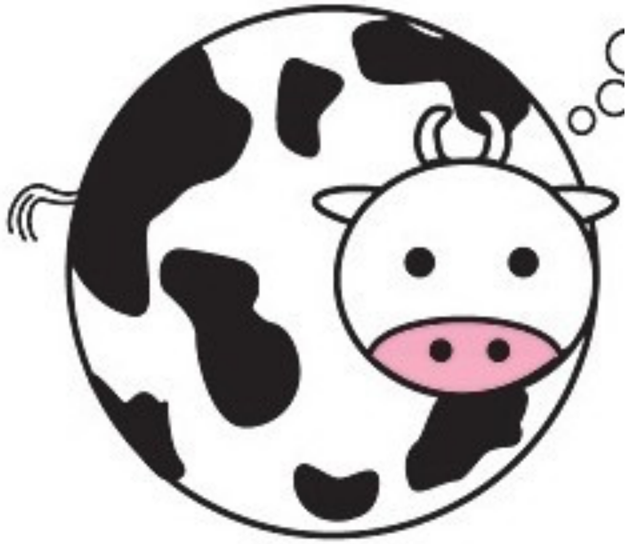
R1	$M^1(1,1)$	$M^1(1,2)$	…	$M^1(1,N)$	$M^1(2,1)$	…, …	$M^1(N,N)$	$M^{n+1}(1,1)$	$M^{n+1}(1,2)$	…	$M^{n+1}(1,N)$	$M^{n+1}(2,1)$	…, …	$M^{n+1}(N,N)$	$M^{1+2n}(1,1)$
R2	$M^2(1,1)$	$M^2(1,2)$	…	$M^2(1,N)$	$M^2(2,1)$	…, …	$M^2(N,N)$	$M^{n+2}(1,1)$	$M^{n+2}(1,2)$	…	$M^{n+2}(1,N)$	$M^{n+2}(2,1)$	…, …	$M^{n+2}(N,N)$	
⋮	⋮	⋮		⋮	⋮		⋮	⋮	⋮		⋮	⋮		⋮	
Rn	$M^n(1,1)$	$M^n(1,2)$	…	$M^n(1,N)$	$M^n(2,1)$	…	$M^n(N,N)$	$M^{2n}(1,1)$	$M^{2n}(1,2)$	…	$M^{2n}(1,N)$	$M^{2n}(2,1)$	…	$M^{2n}(N,N)$	$M^{3n}(1,1)$

Matrix size **NxN**, vector unit size **n**

See talk tomorrow by Matevz Tadel

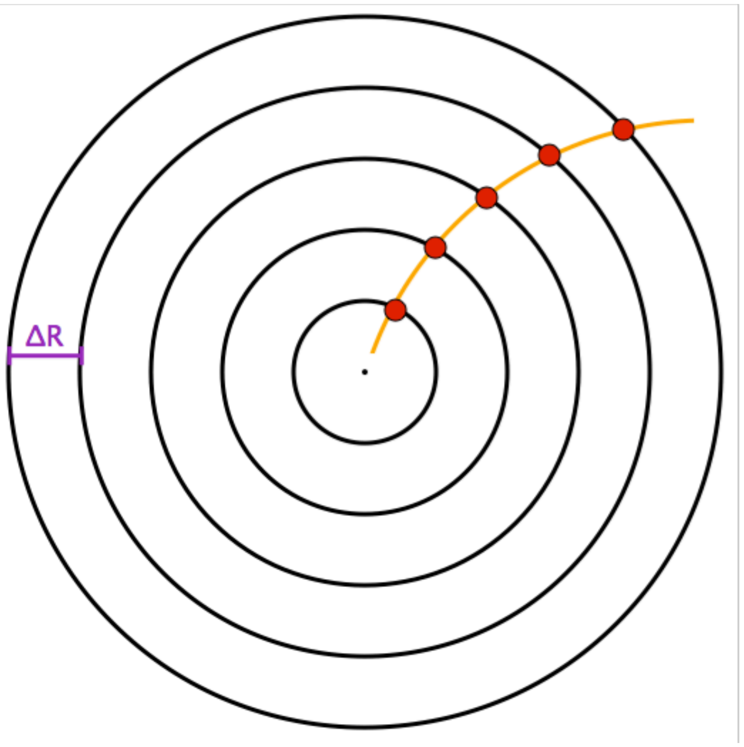
Initial Experimental Setup

- Simple starting point:
  - "Cylindrical Cow":10 barrel layers, △R = 4cm, |η|<1, 3.8T magnetic field
  - Beam spot 1mm in xy, 1cm in z
  - Hit resolution 100μm in r-phi, 1mm in z
  - Uncorrelated tracks, no scattering



https://sites.google.com/site/lauranstoner/

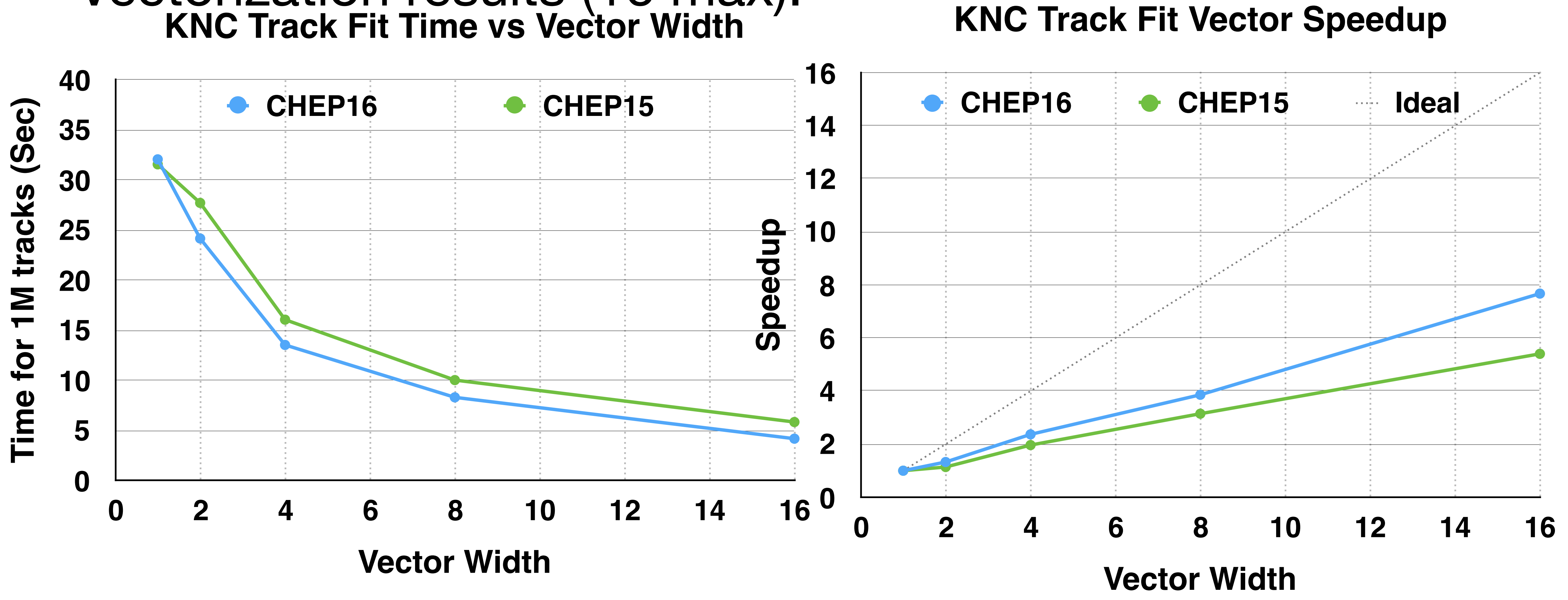
- Simplest case — we'd better understand this
- Expect performance under these circumstances to be upper limit on how well you can do



- move to realistic detector after this has been understood

How well does it perform?

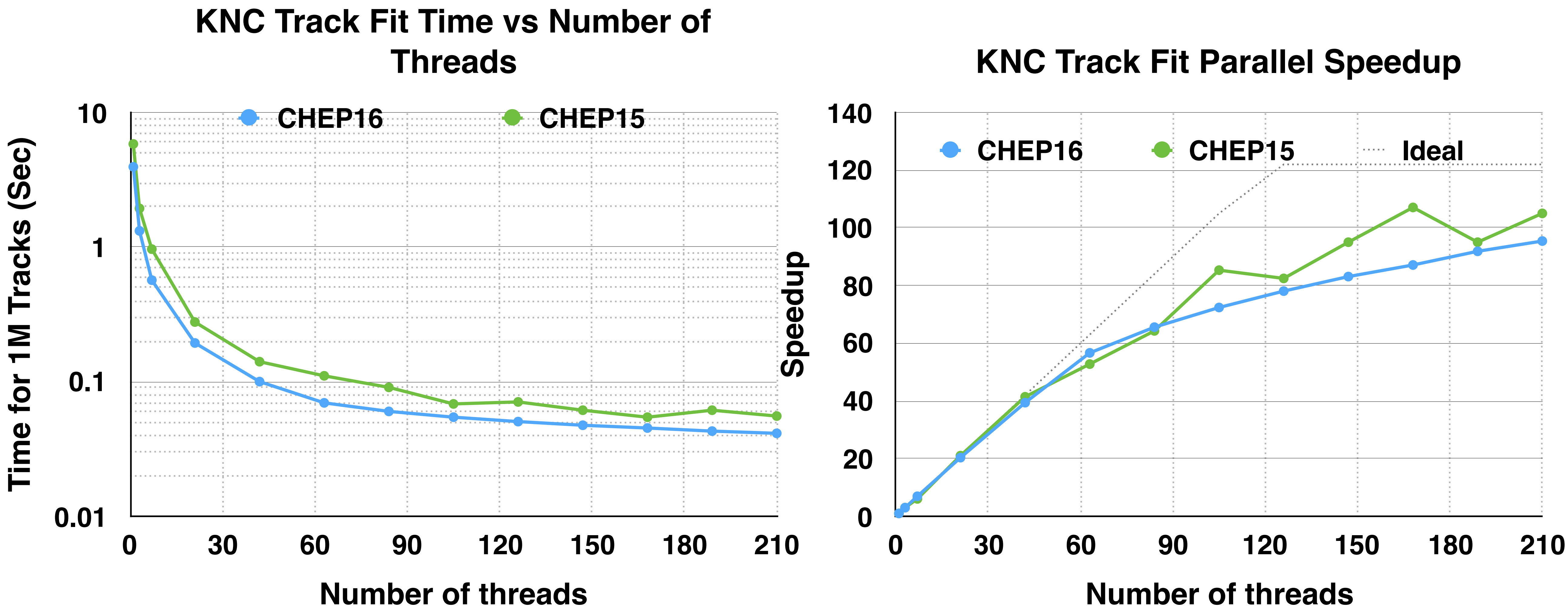
- Naively might expect speed-ups of 200+ on Xeon Phi (cf scalar single threaded code). What actually happens?
  - (remember — toy detector)
- Test ***Track Building***. Simplest case:
  - KF calculation is just a repetition of propagate & update steps
  - No branching, all tracks do the same thing, only 1 path to follow
  - Vectorization results (16 max):



Speed-up by factor of up to 8 — impressive, but only ½ of theoretical maximum

How well does it perform?

- What about parallelization?



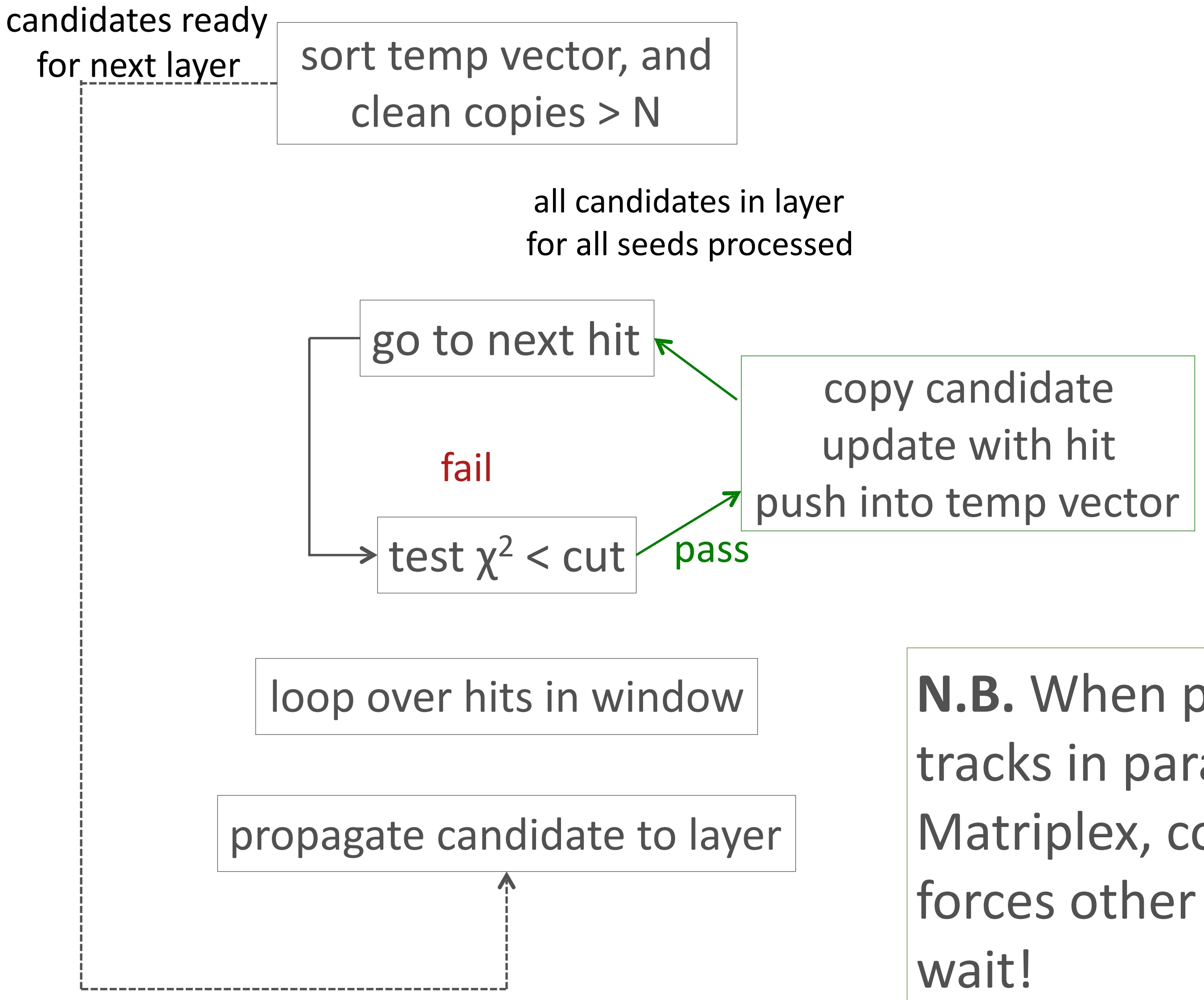
- Parallelization near ideal up to 61 threads
- Reach ~100x speedup at ~200 threads
- Ideally ≥122x to occupy available instruction slots
- CHEP2016 faster due to better vectorization

Gains deviate from ideal at around 60 threads

Track building?

- Remember this is harder
- branches, variable execution, etc etc etc …
- expect performance to degrade

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test $\chi^2 < \mathrm{cut}$

pass

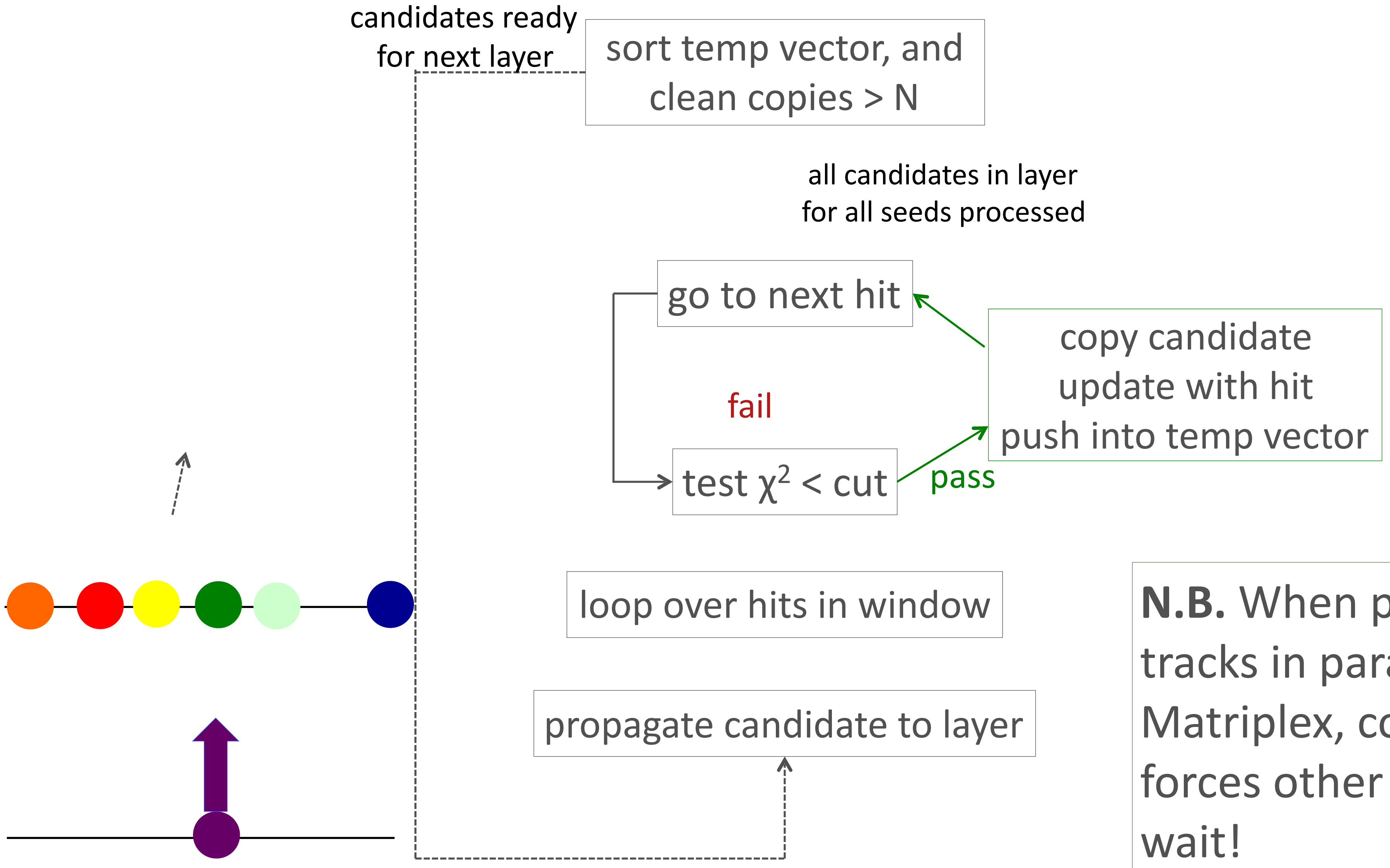
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

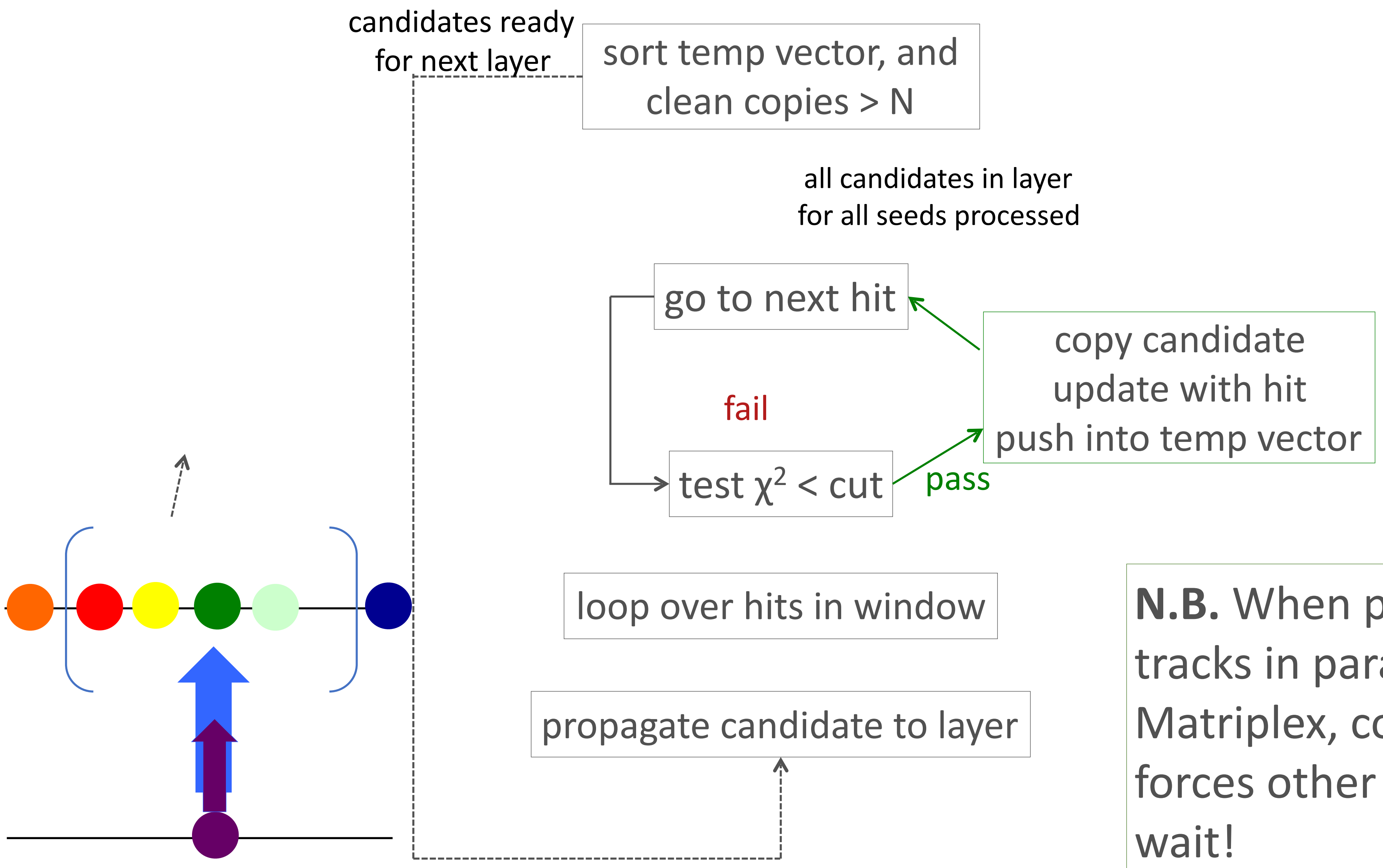
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready
for next layer

sort temp vector, and
clean copies > N

all candidates in layer
for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

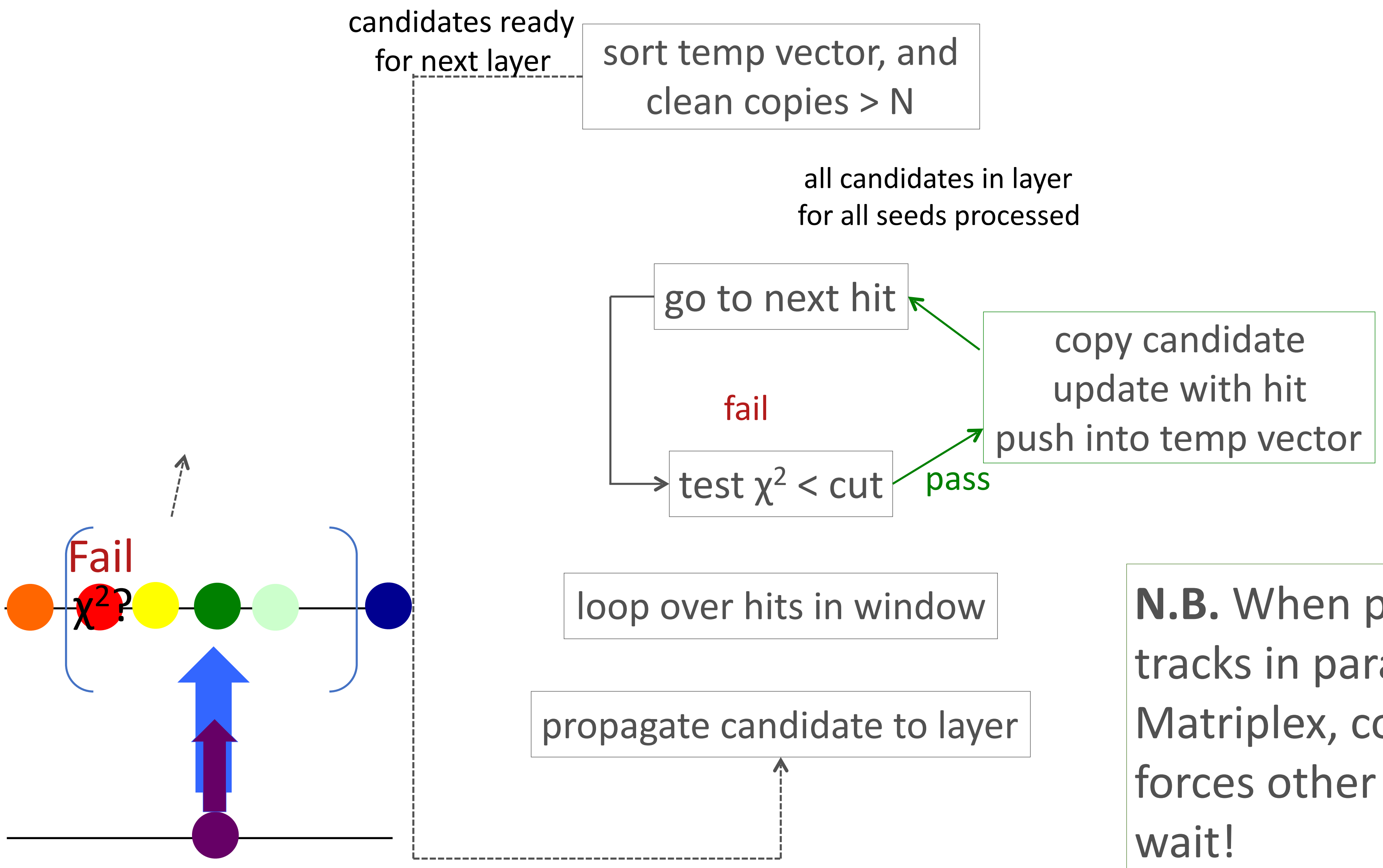
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

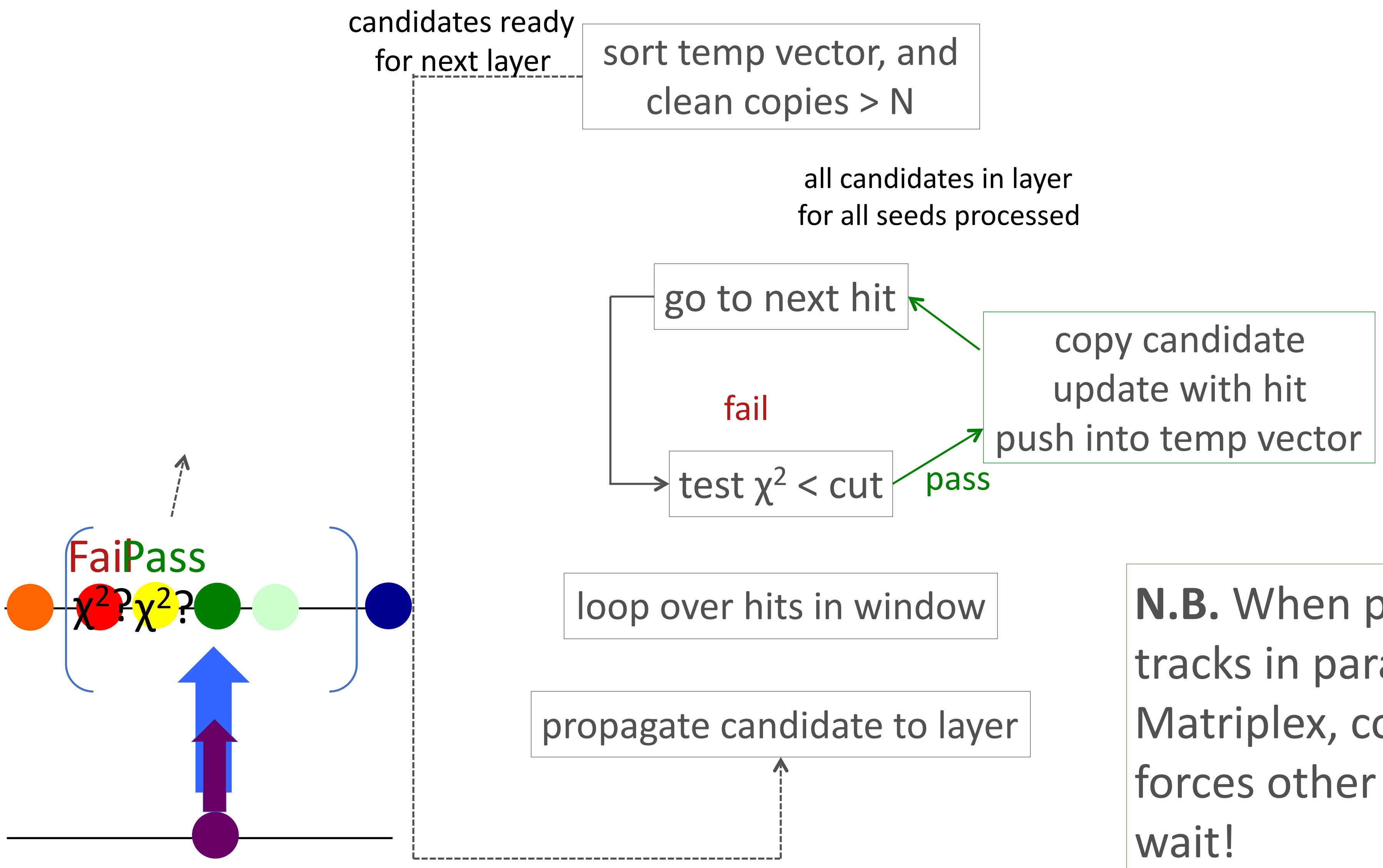
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

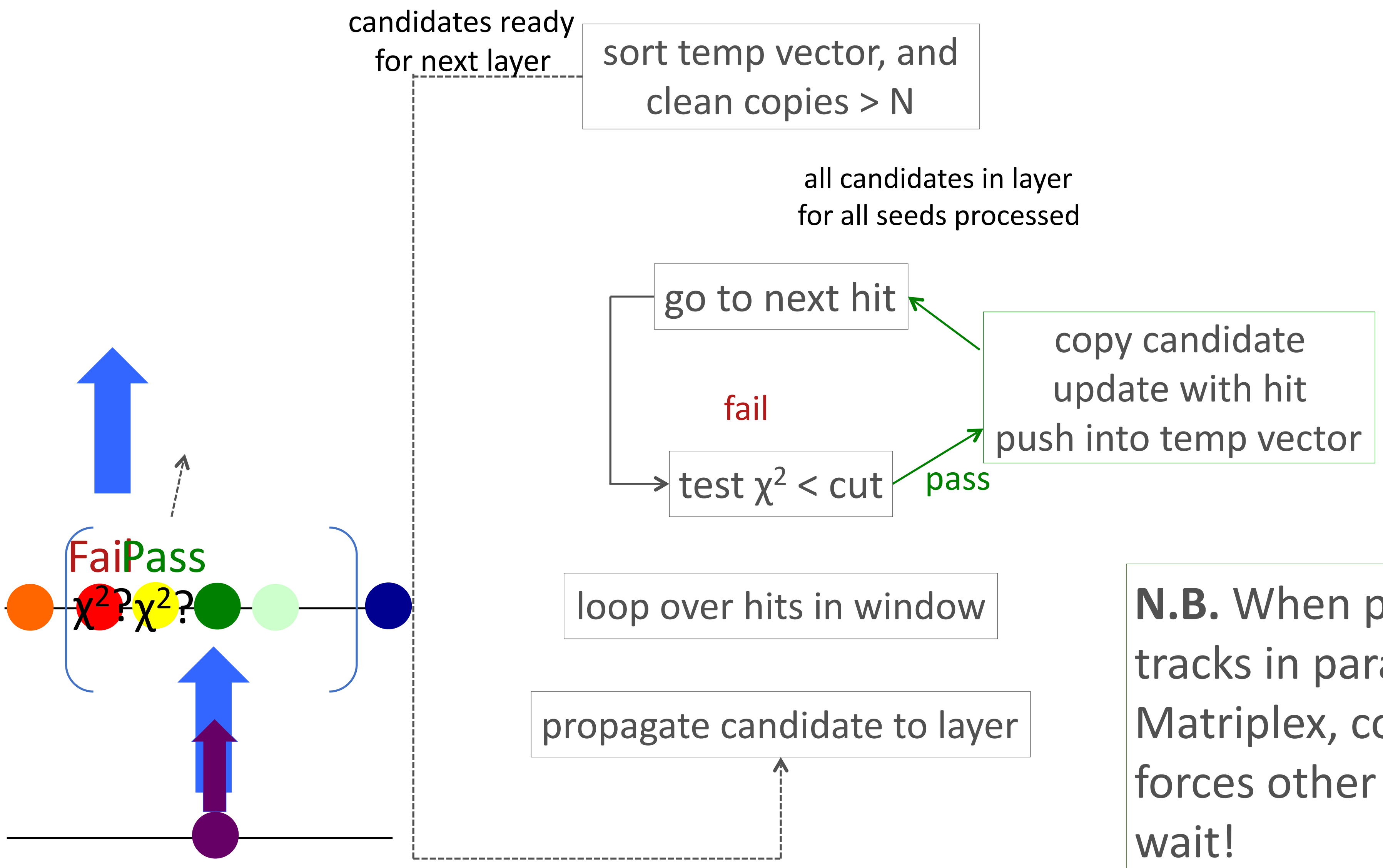
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

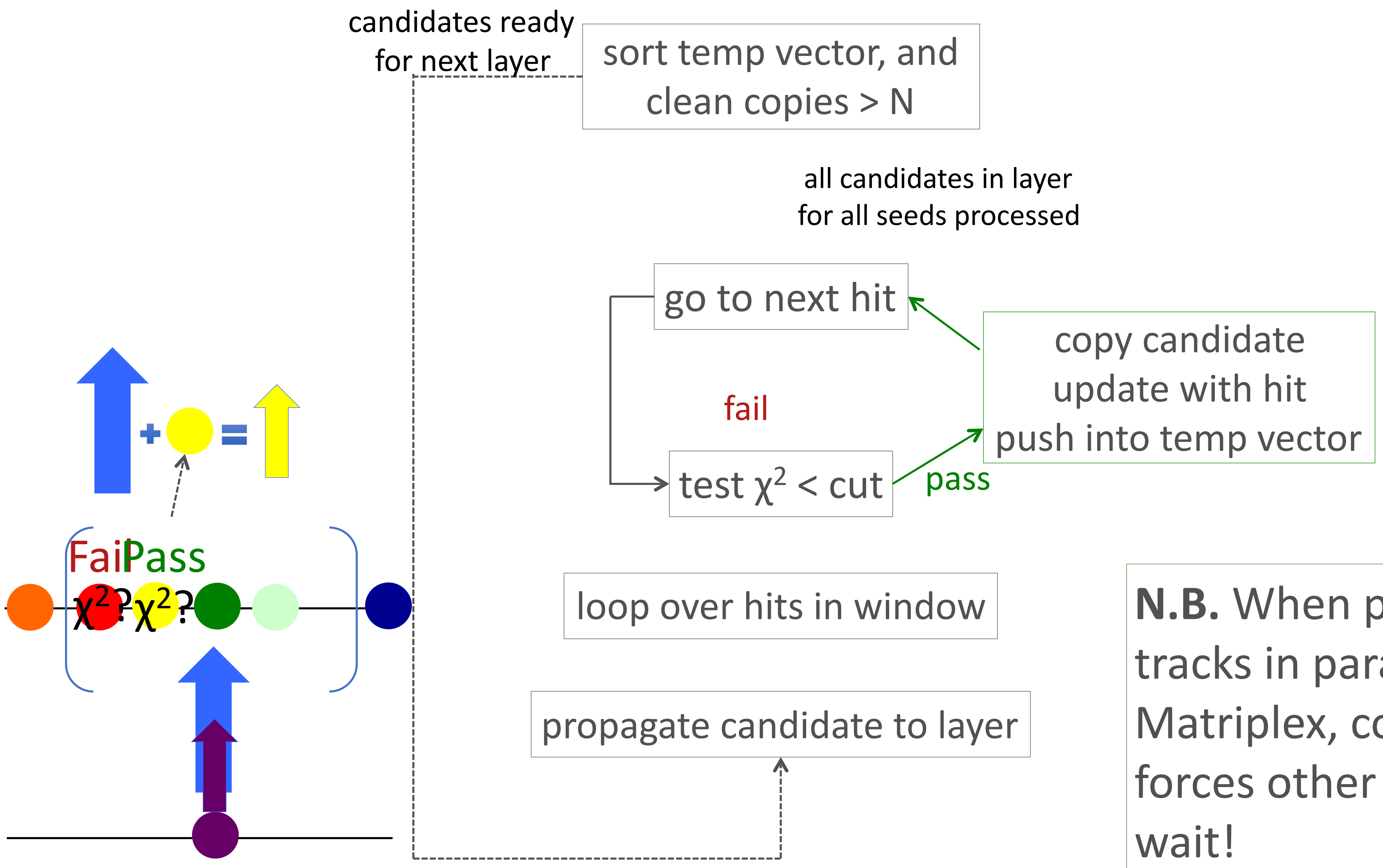
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test $\chi^2 <$ cut

pass

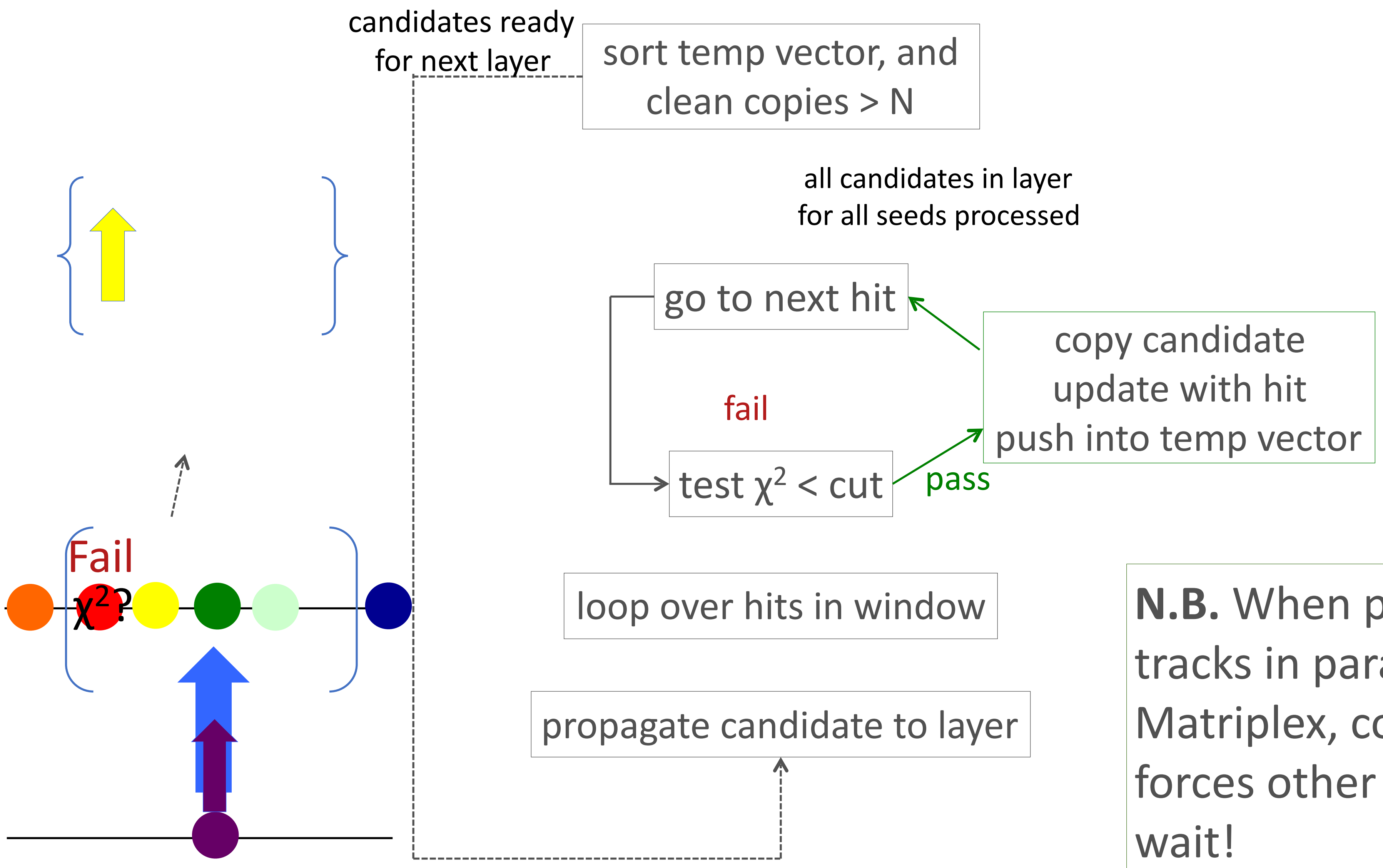
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

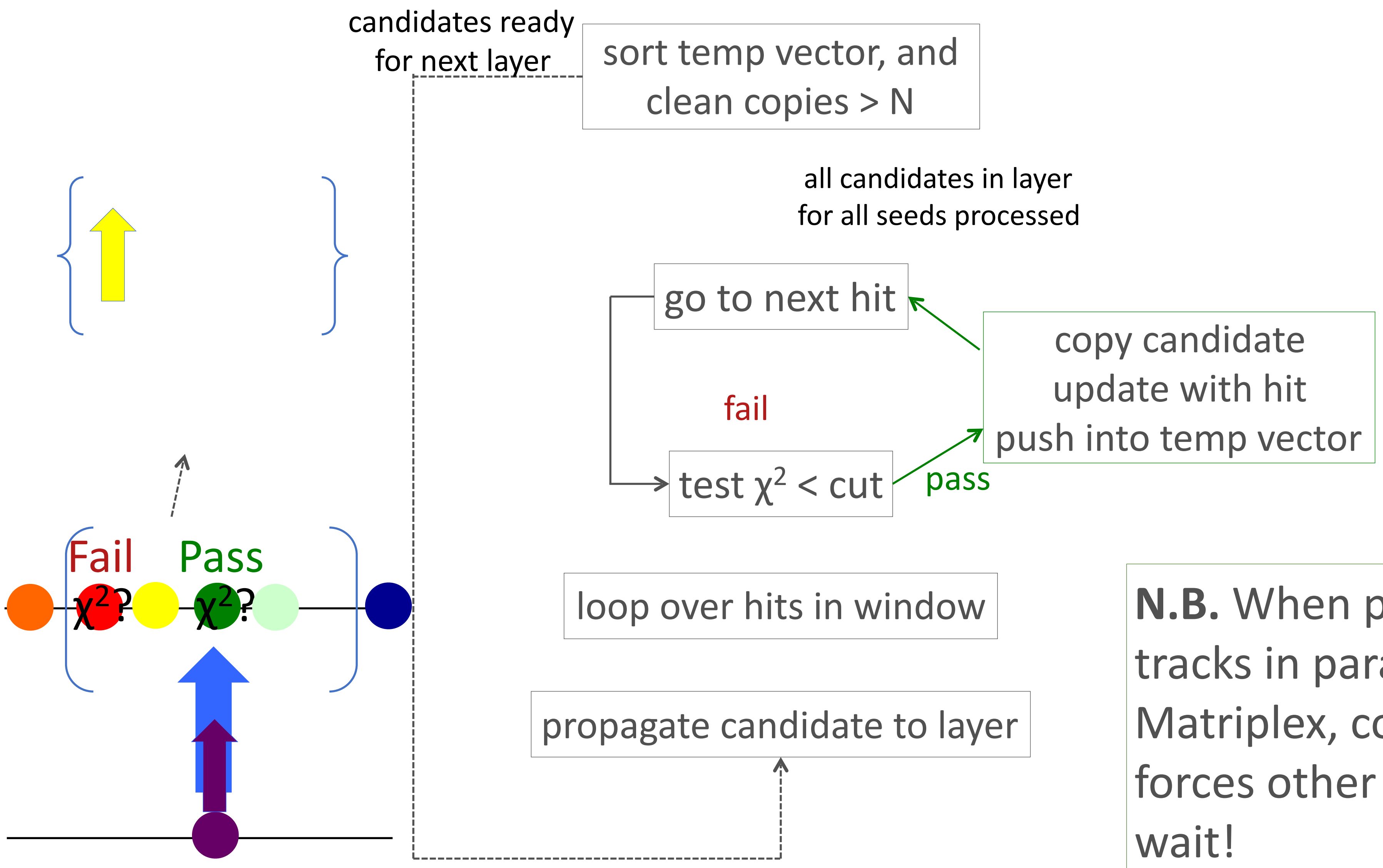
Handling Multiple Track Candidates: First Approach



N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

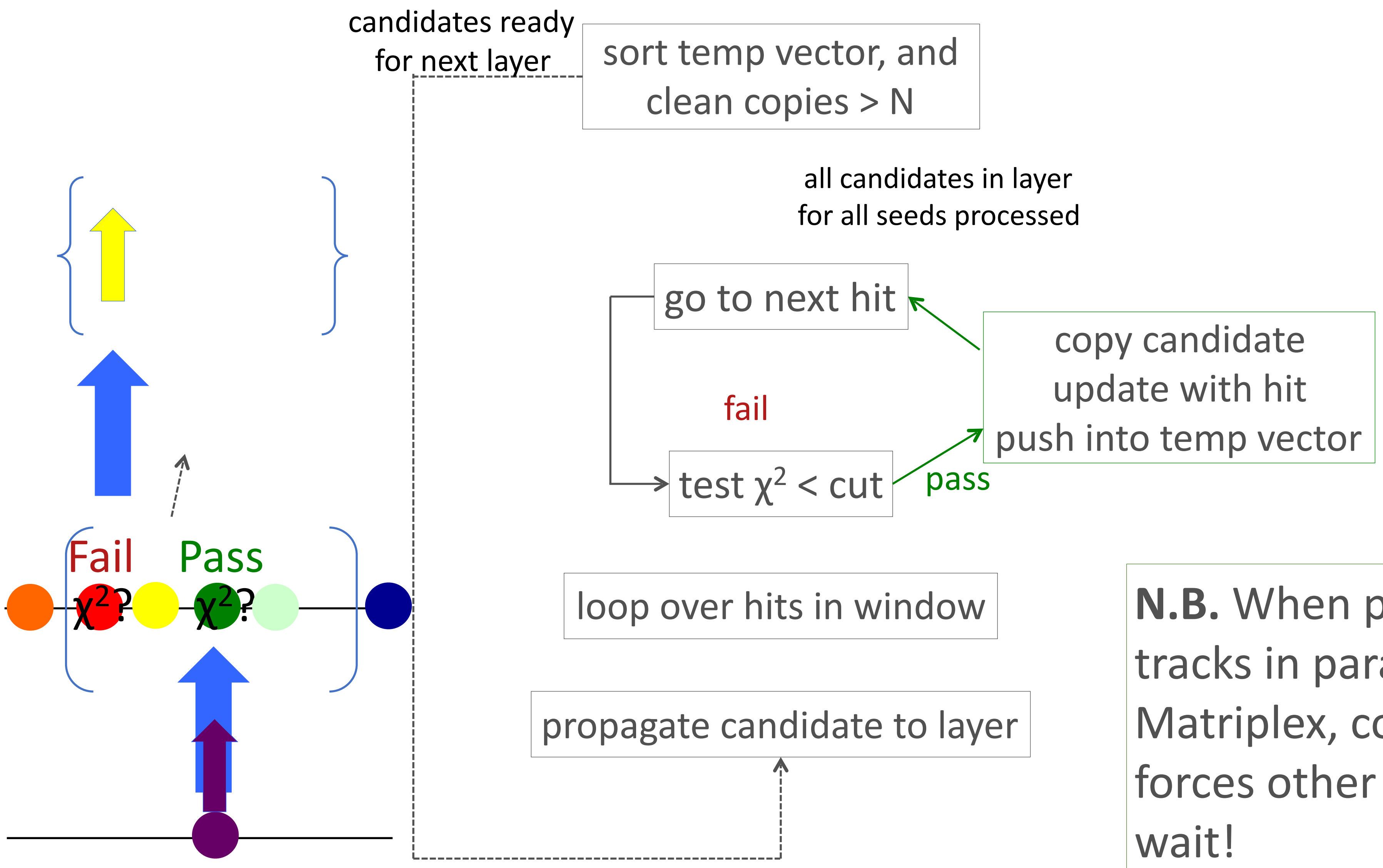
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

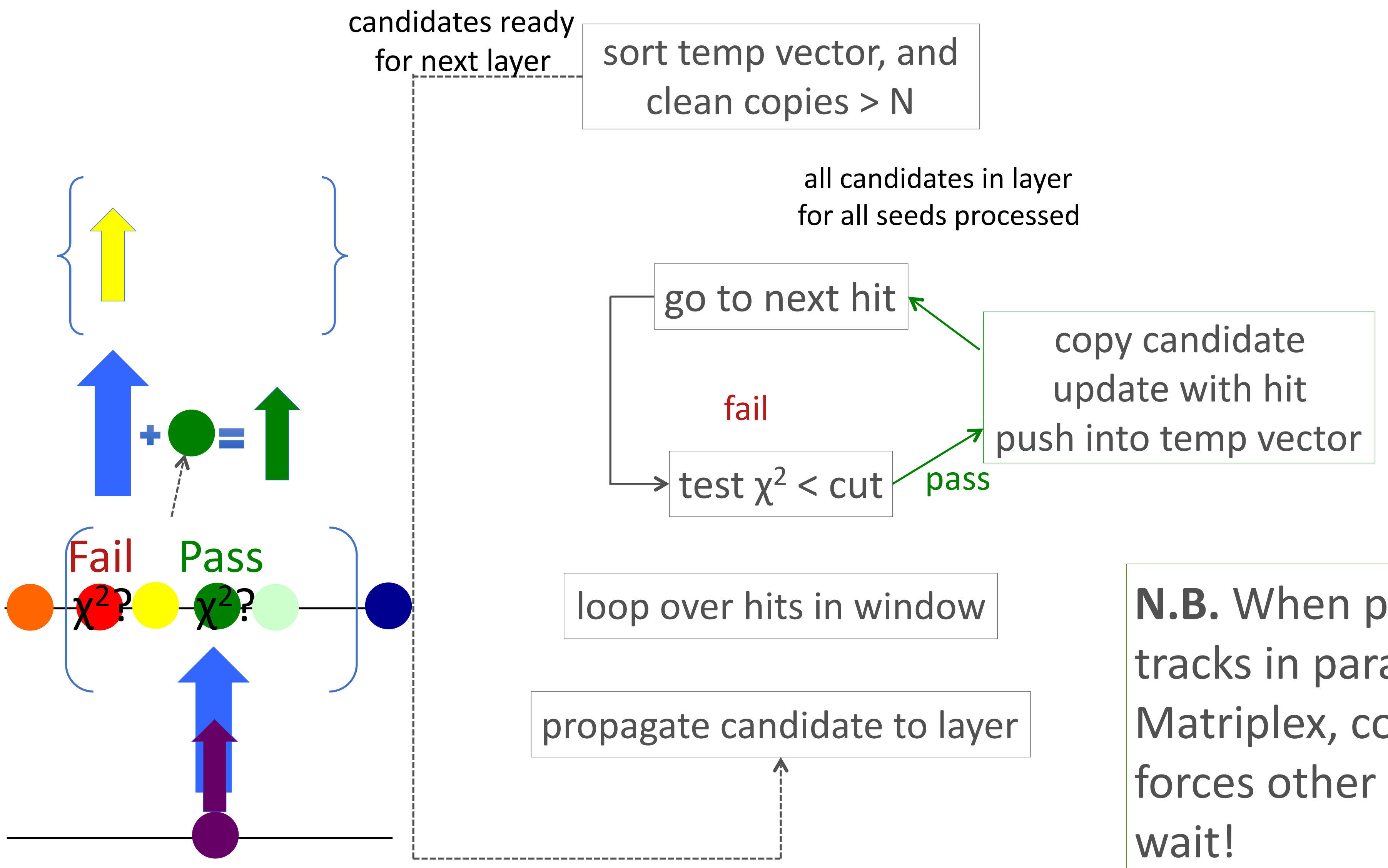
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

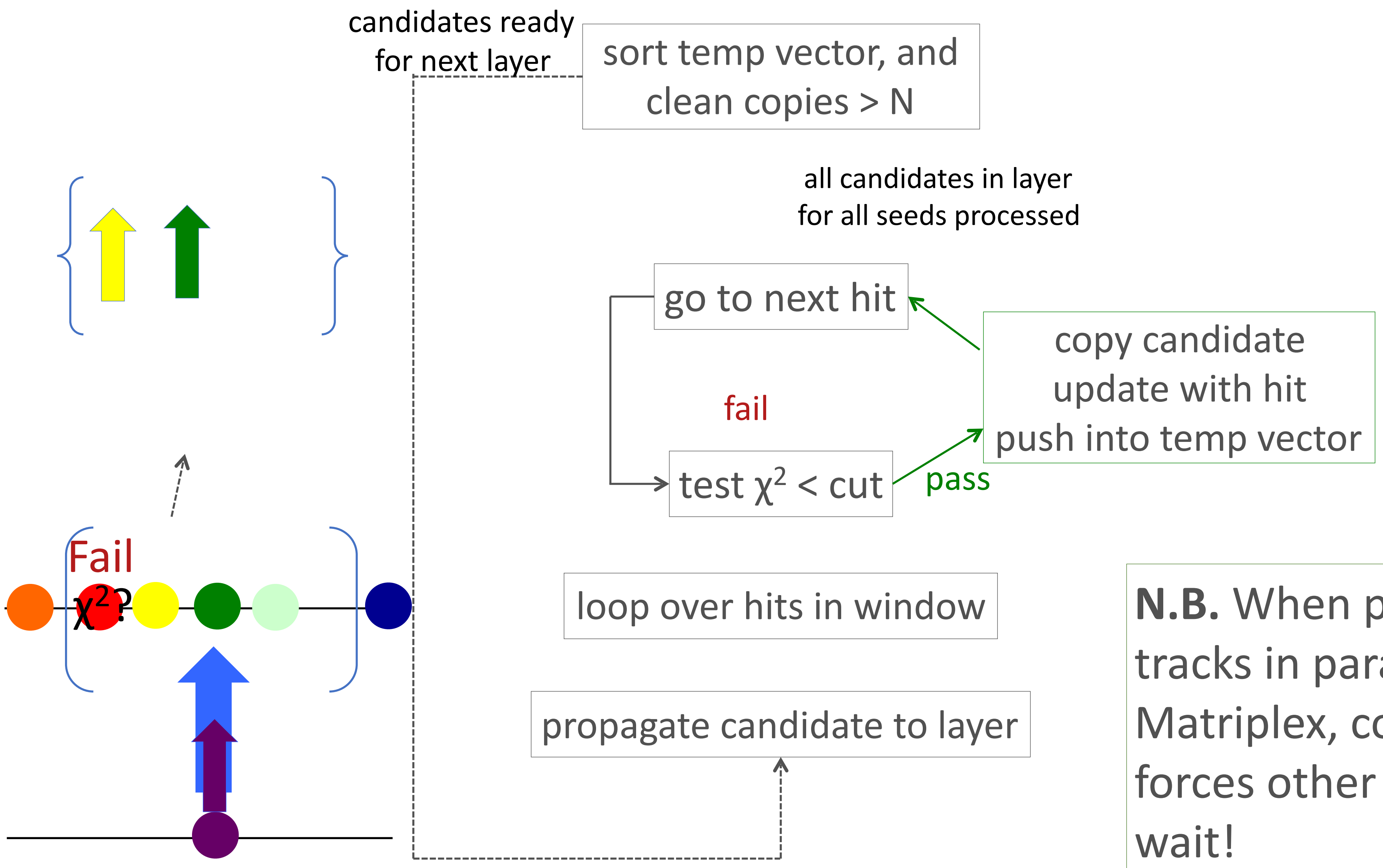
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test $\chi^2 <$ cut

pass

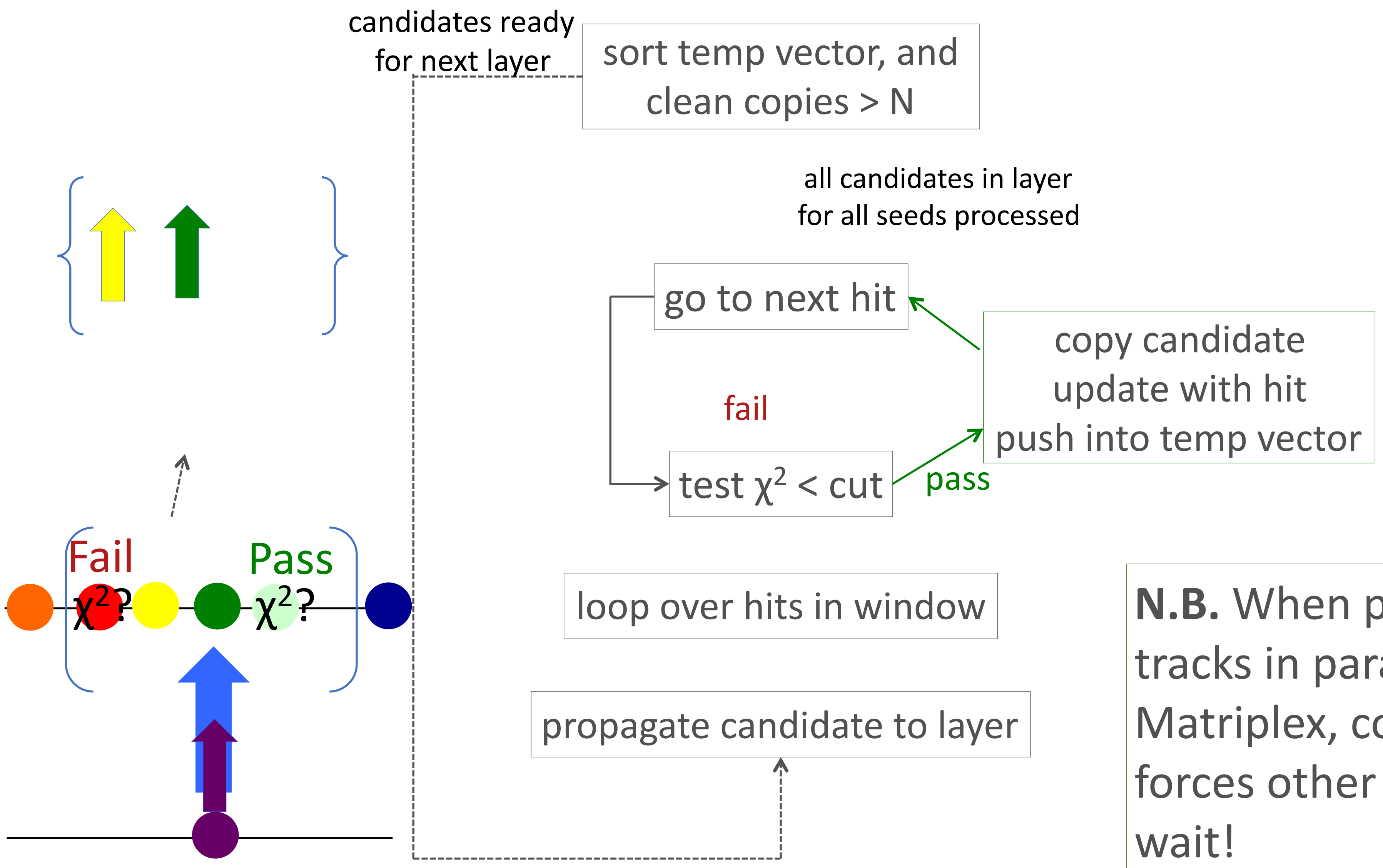
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

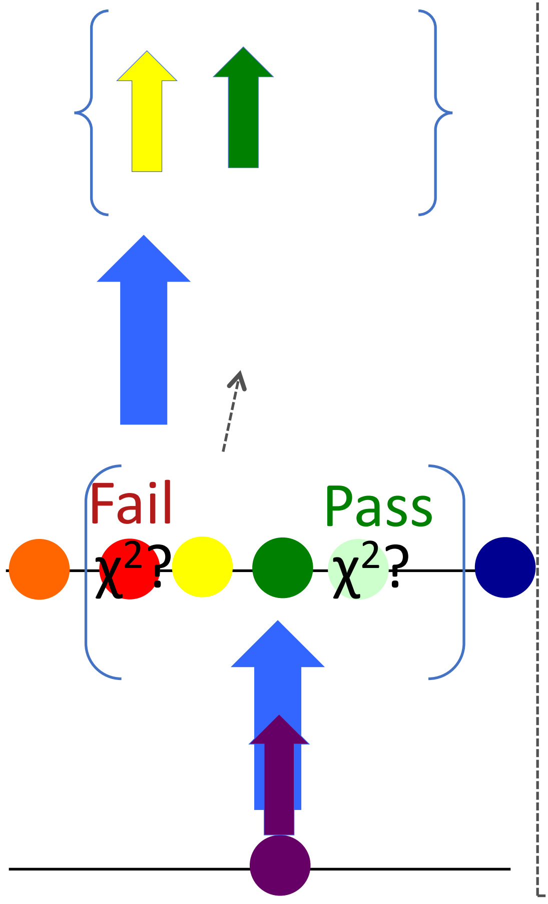
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test χ^2 < cut

pass

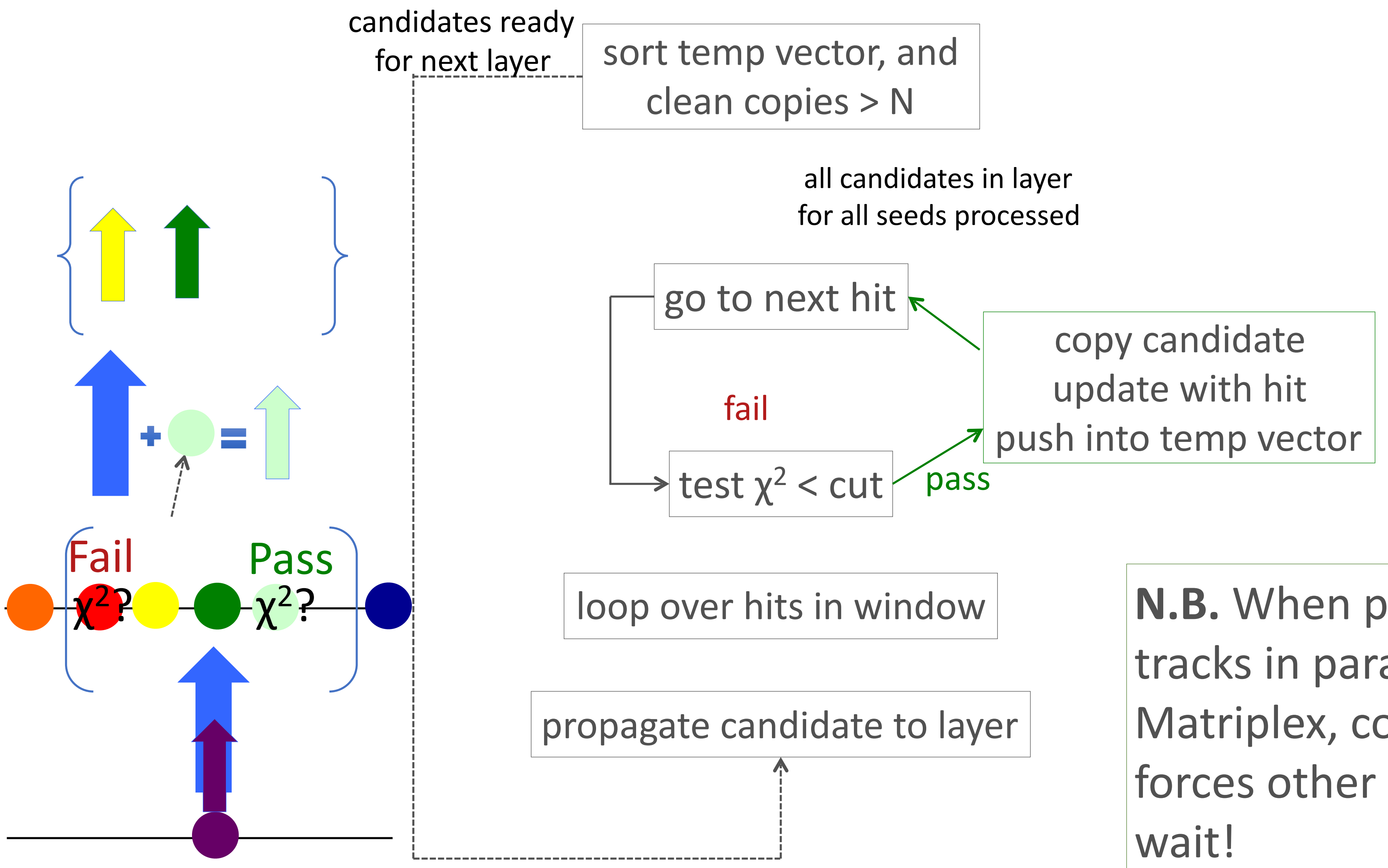
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test $\chi^2 <$ cut

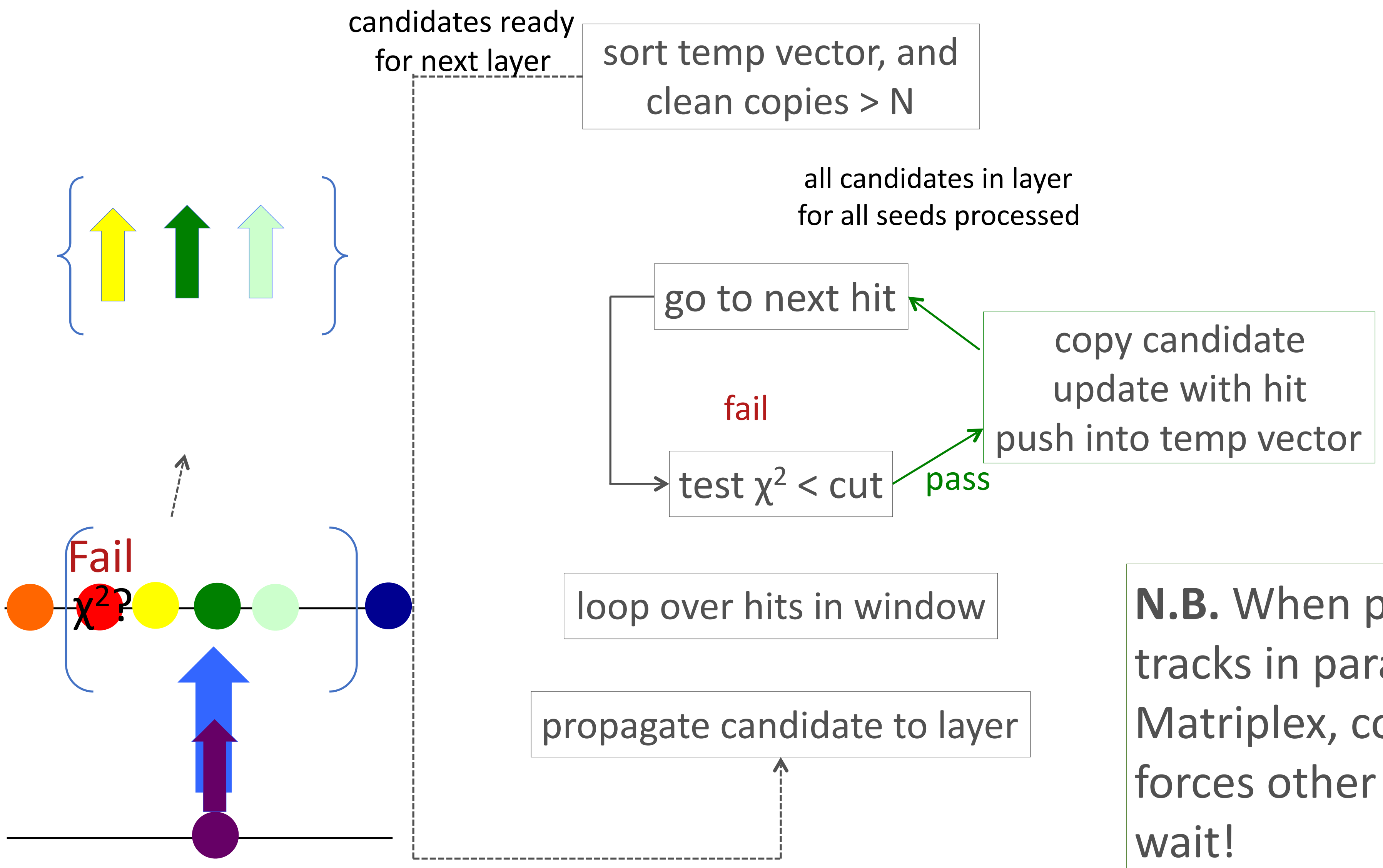
pass

loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!
➔ We need an other approach

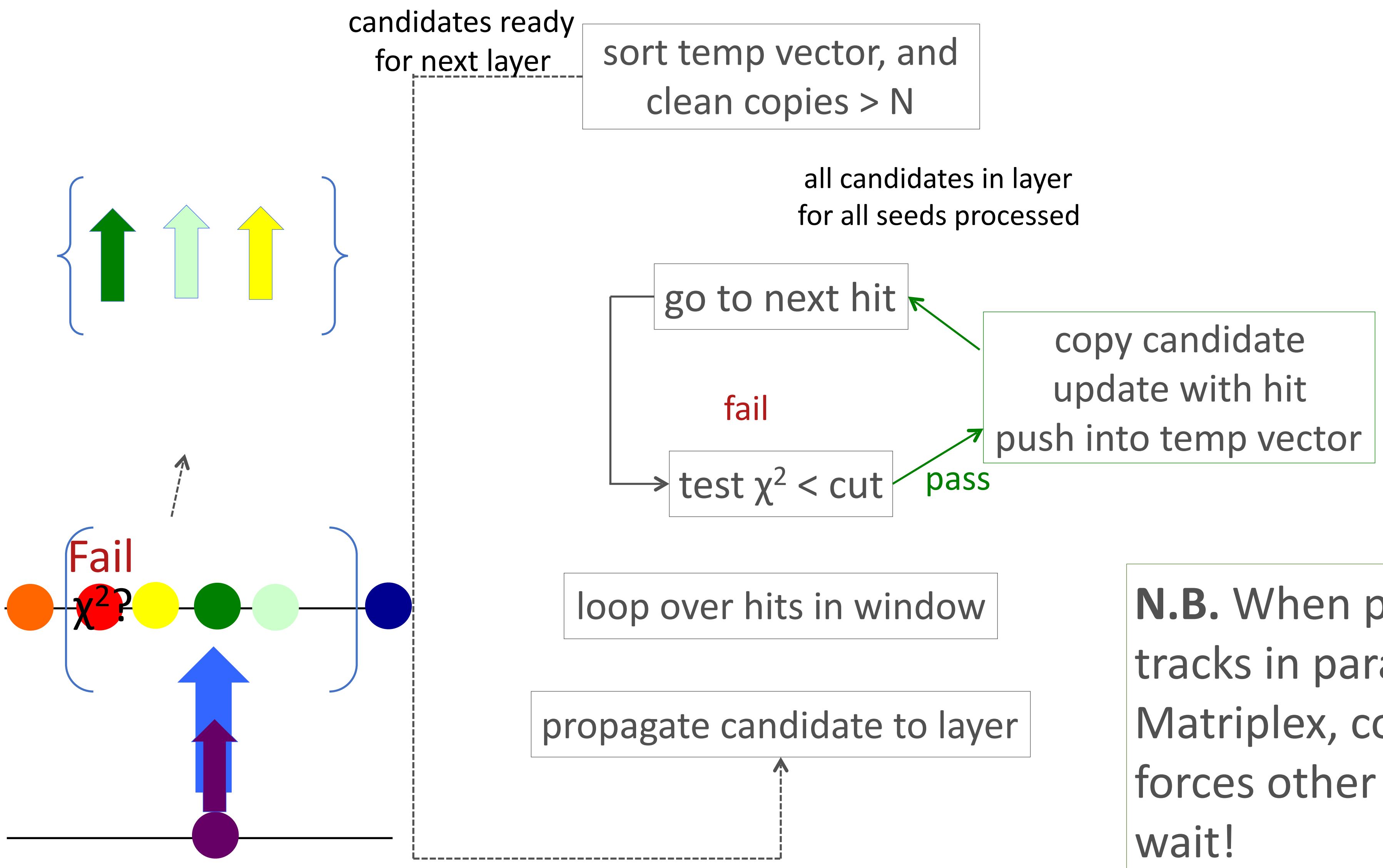
Handling Multiple Track Candidates: First Approach



N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

fail

copy candidate
update with hit
push into temp vector

test χ^2 < cut

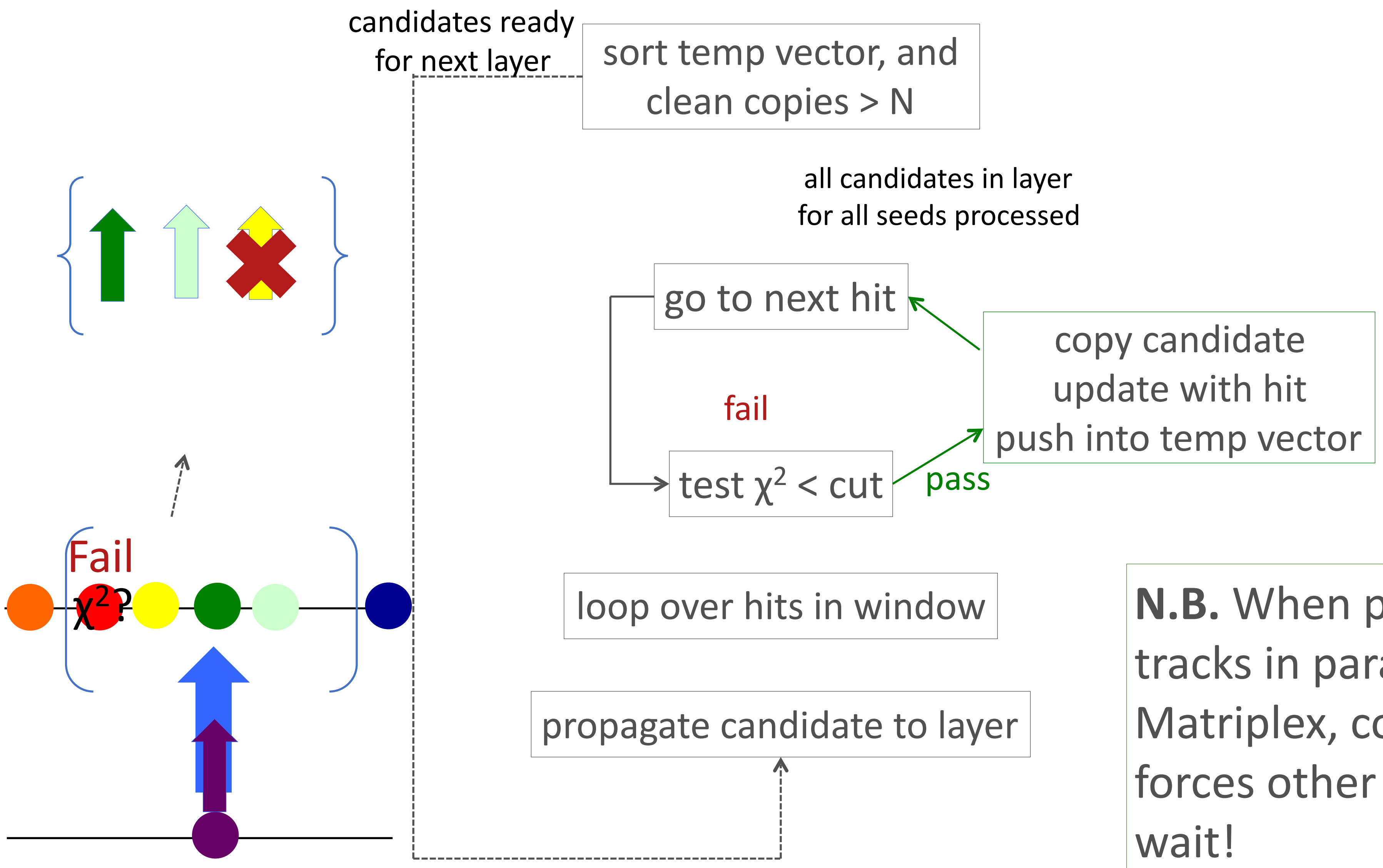
pass

loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!
➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready
for next layer

sort temp vector, and
clean copies > N

all candidates in layer
for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test $\chi^2 < \text{cut}$

pass

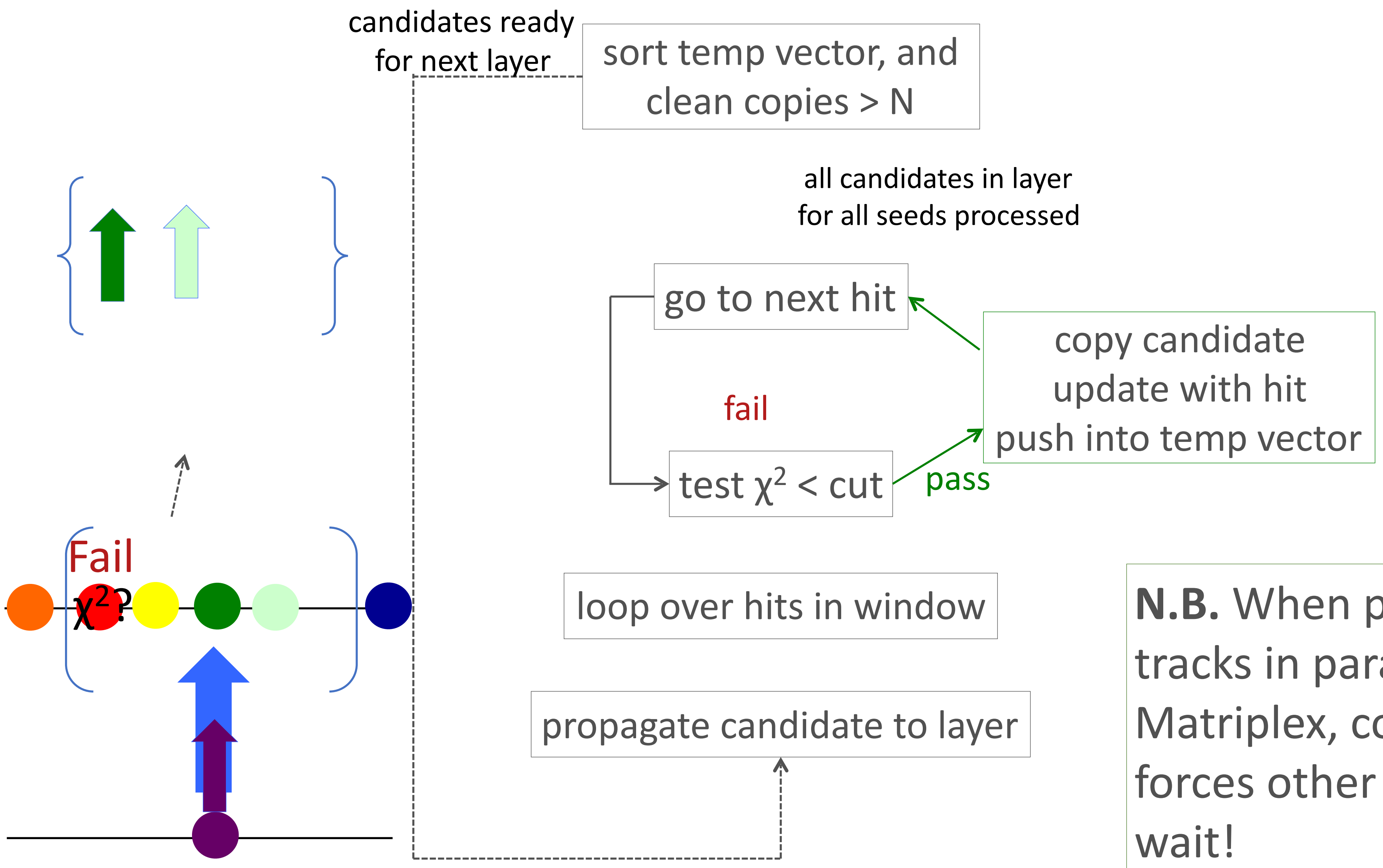
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Handling Multiple Track Candidates: First Approach



candidates ready for next layer

sort temp vector, and clean copies > N

all candidates in layer for all seeds processed

go to next hit

copy candidate
update with hit
push into temp vector

fail

test $\chi^2 <$ cut

pass

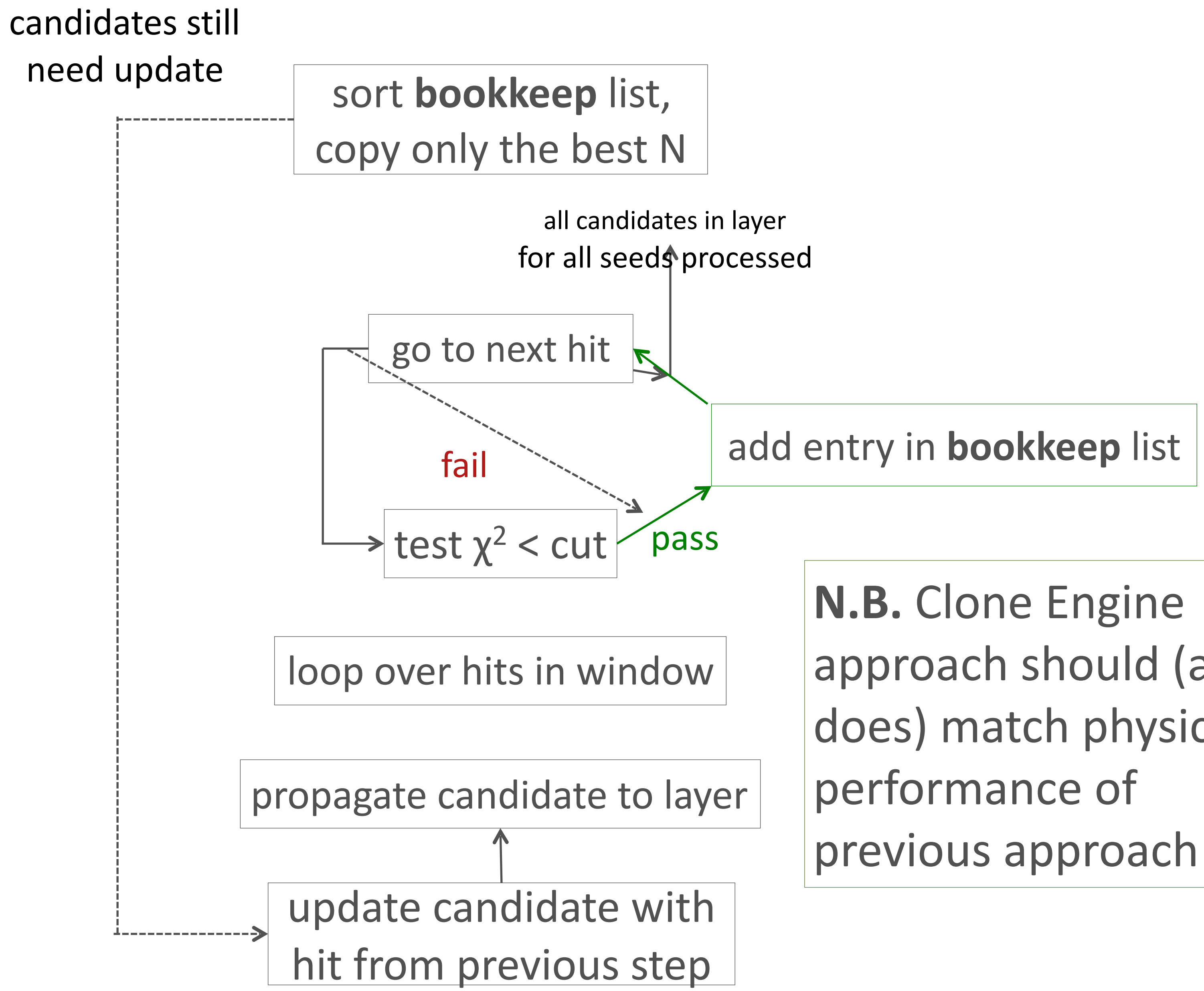
loop over hits in window

propagate candidate to layer

N.B. When processing tracks in parallel with Matriplex, copy + update forces other processes to wait!

➔ We need an other approach

Optimized handling of multiple candidates: “Clone Engine”



candidates still need update

sort **bookkeep** list, copy only the best N

all candidates in layer for all seeds processed

go to next hit

add entry in **bookkeep** list

fail

test χ^2 < cut

pass

loop over hits in window

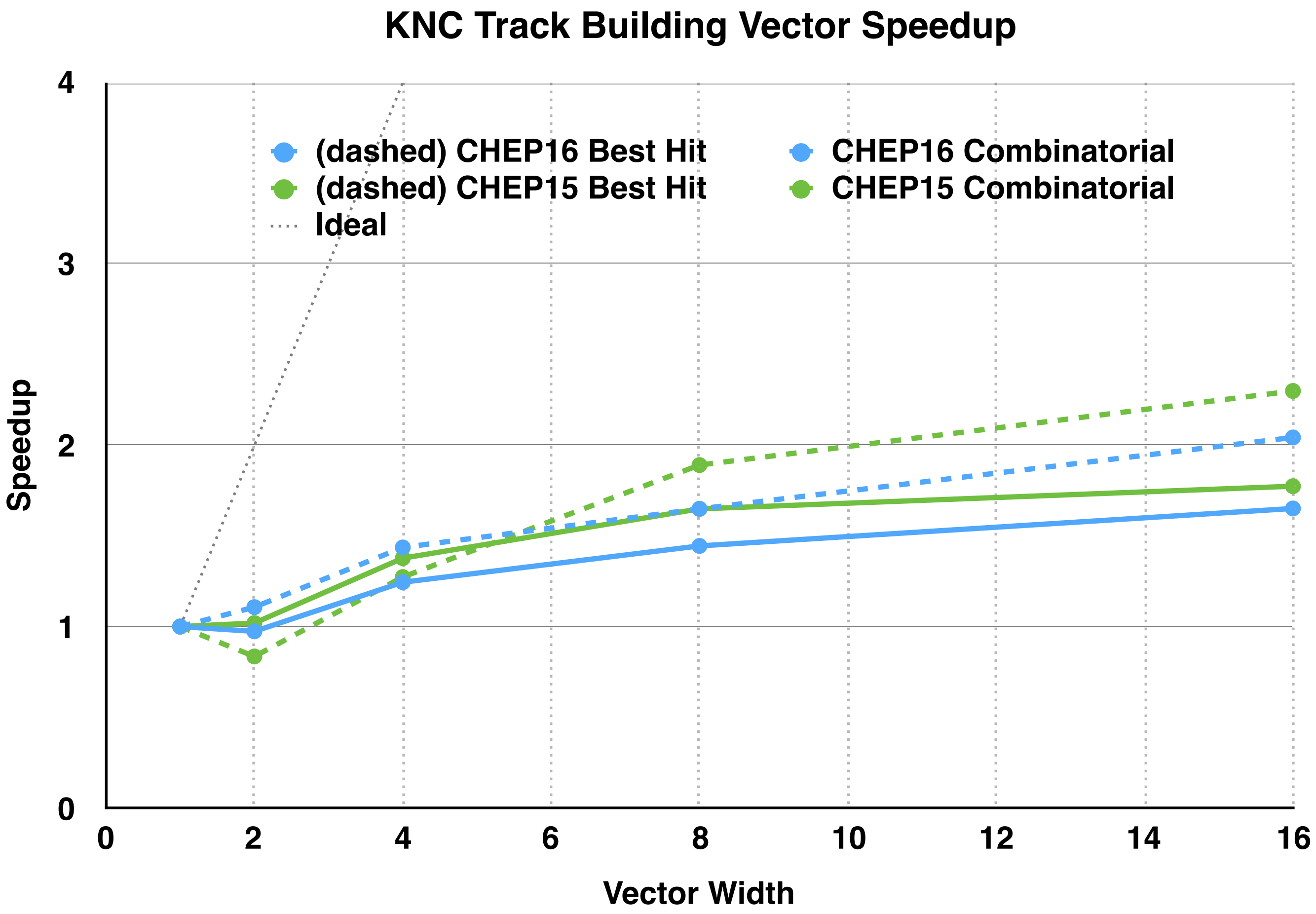
propagate candidate to layer

update candidate with hit from previous step

N.B. Clone Engine approach should (and does) match physics performance of previous approach!

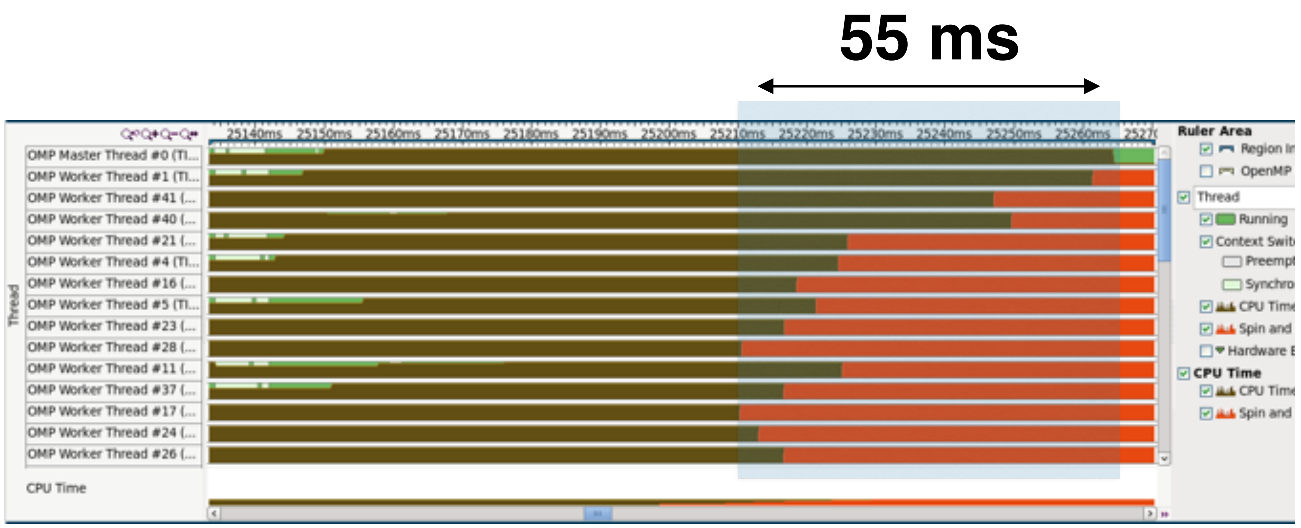
Vectorization: Track Building

- Much more challenging:
  - Branches to select candidates impairs vectorization
  - Adding multiple candidates at each layer leads to frequent data repacking
  - More complicated data structures and poorer data locality stress **cache size** and **memory bandwidth**
- Lots of work to understand results
  - ~2x speedup (SNB: also ~2x speedup)
  - Improving this becomes more critical as number of vector registers increases

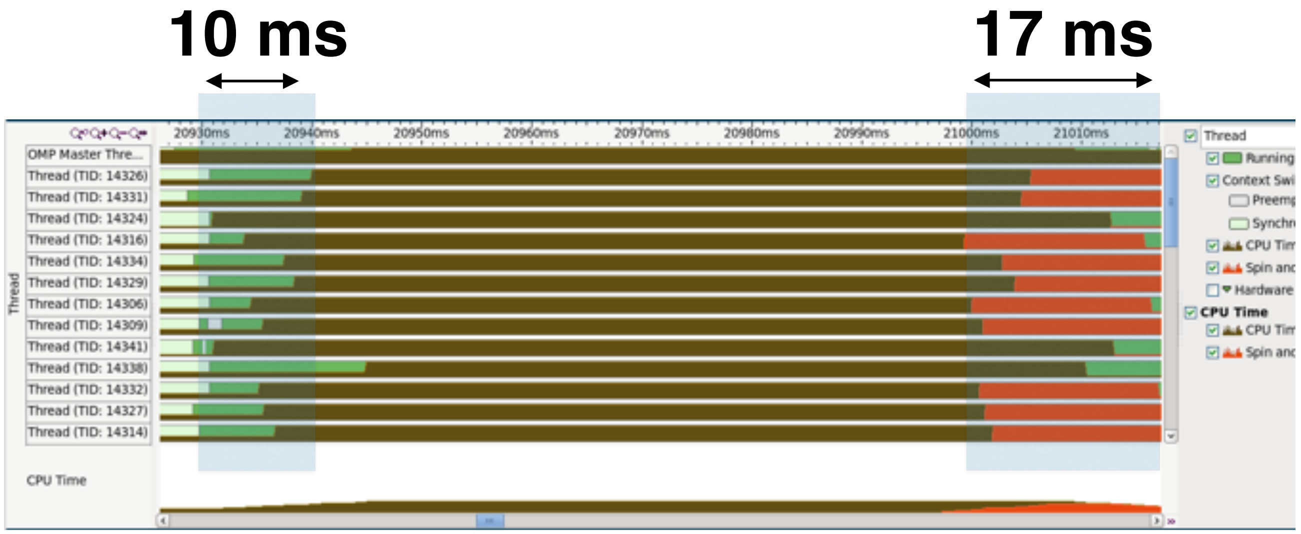


different parallelization schemes

- OpenMP shows large tail effects due to uneven distribution of work from **static partitioning**



- TBB work stealing, wth smaller units of work and **dynamic partitioning**, reduces tail effects



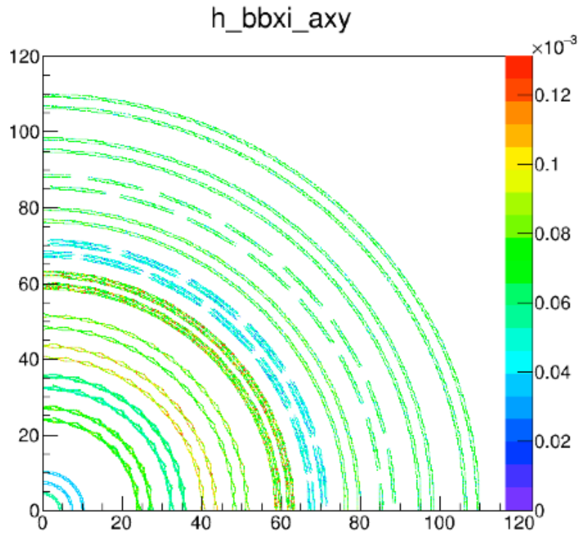
Challenge: keep all the resources busy

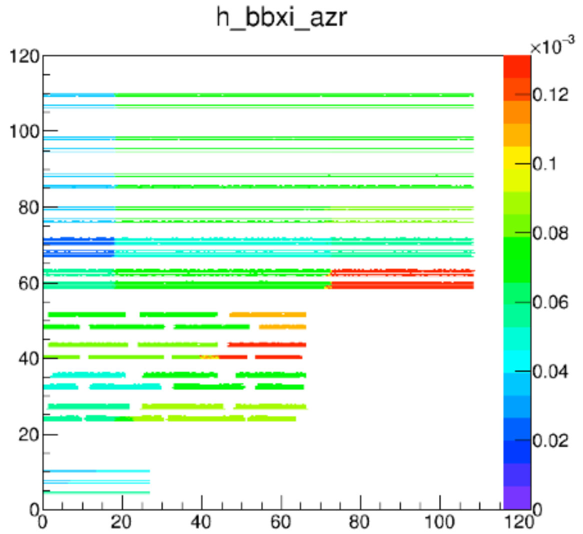
Track Building Lessons

- Data locality is critical (w/speedups, compared to earlier version):
  - Optimize/vectorize copying of tracks into Matriplex (+20%)
  - Minimize dynamic memory allocations (+45%)
  - Avoid unnecessary object instantiations, copies (+25%)
  - Minimize size of data structures, smarter low-level algorithms (+30%)
- Parallelization — different toolsets (OpenMP vs TBB)
  - Static binning with OpenMP led to “tail effects” due to variable distribution of work
  - TBB work-stealing is an easy way to even out load variability
  - Optimizing work partition size still critical—too large doesn’t allow enough balancing, too small has high over head costs

Adding Realism

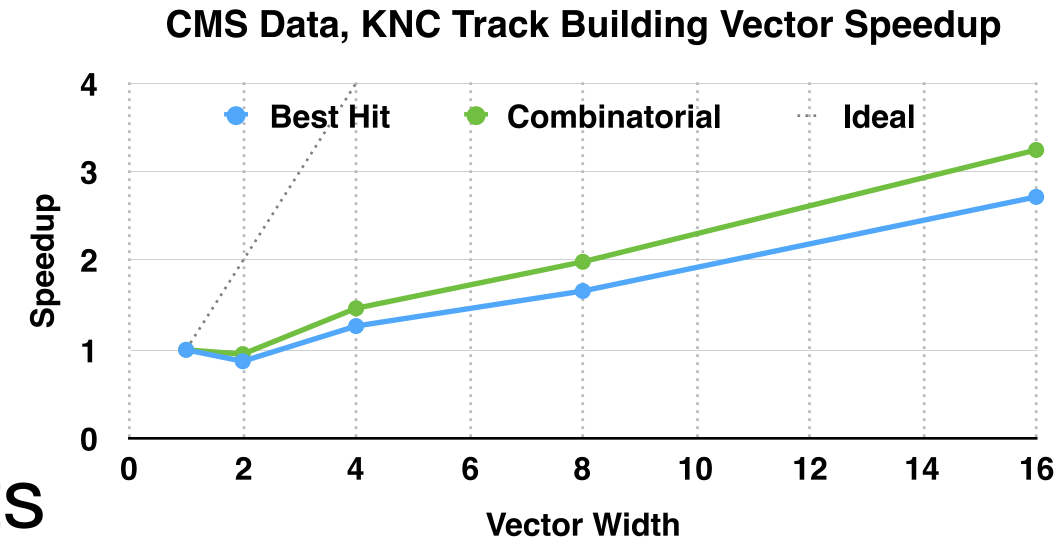
- Move beyond our circular cow
  - Ultimately need to include realistic geometry, material effects, inefficiencies, overlaps, etc.
  - Use CMS simulation, add complexity in incremental steps
- Two step propagation to avoid using the full geometry
  - Simple parameterization of CMS geometry and material
  - Step 1: propagate to the average radius of the layer
  - Step 2: propagate to the exact hit radius
- Endcap/Disks
  - Propagate to z, similar handling of material and propagation

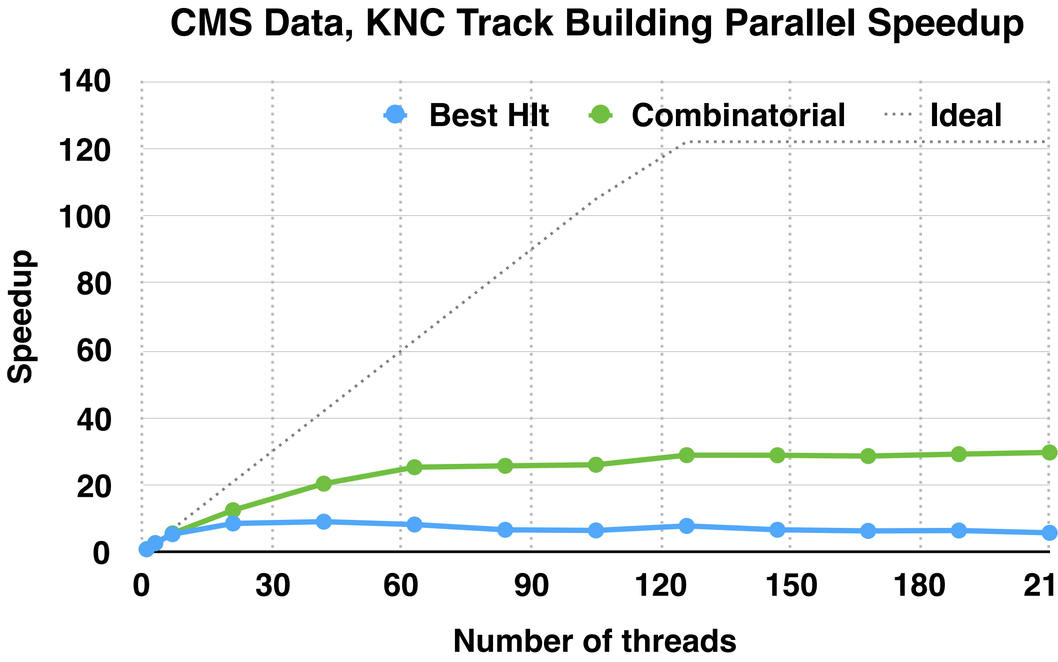




Early realistic simulation results

- Early tests with CMS simulation data
  - Hits from full CMS simulation
  - Parameterized geometry & material effects
- Vectorization is better
  - more complicated propagation results in more time spent in well-vectorized routines
- Parallelization speedup is worse than toy setup
  - Events are smaller than toy events, increasing parallelization overhead
    - Multiple events in flight
  - Possibly other effects from more complex geometry



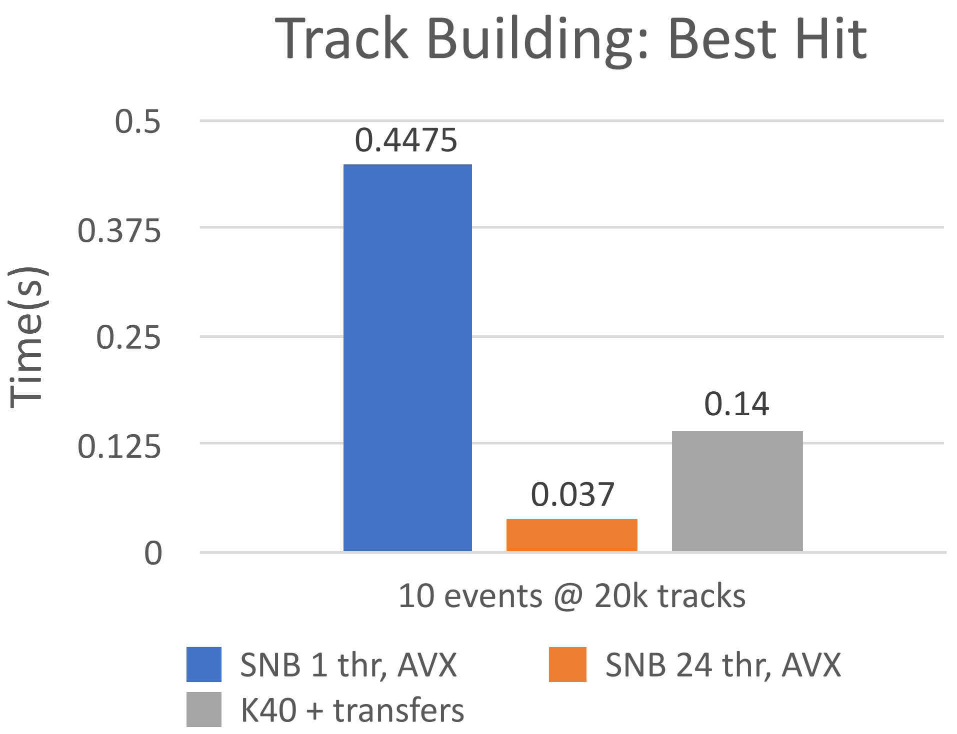


Conclusions

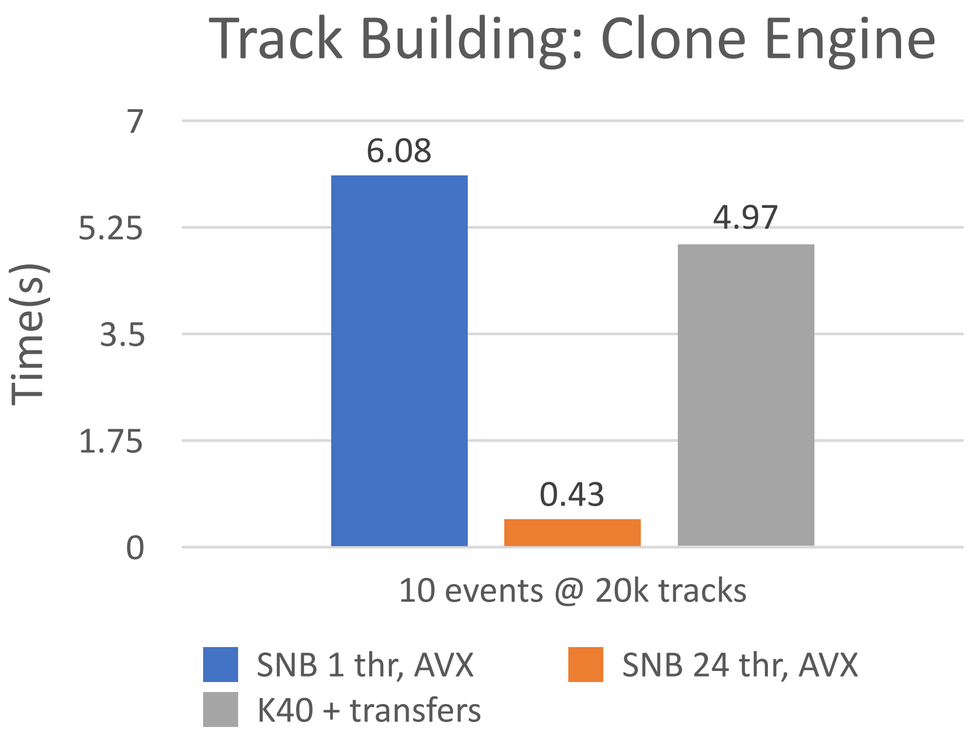
- Gave you a flavor or track reconstruction in collider experiments
  - Not enough time to really talk about the full project - a lot of work done on GPGPU on the same project too …
- Ties into HPC computing — biggest part of event reconstruction timing
- Attempt to take a real problem and use some of the tools you've heard about this week (parallel programming)
- Showed you some of the challenges you run into, and a scale of the improvements we are able to get at this point in time
- These problems are hard!!!!
  - Requires rethinking of algorithms, reorganization of data structures, keeping the resources busy
  - careful measurement and tuning to get at theoretically available performance

Backup

GPU Track Building: Initial Performance



- 20K tracks per event is not enough to give good performance
- Need to increase the number of events concurrently fed to the GPU by using different streams



- Too many synchronizations
- Sorting’s branch predictions
- Idling threads when number of candidates per seed is not maximum
- Transfer account for 46% of the time

GPGPU coprocessor: interaction between CPU and GPU brings new complications that need to be managed, such as data transfers and optimal reuse