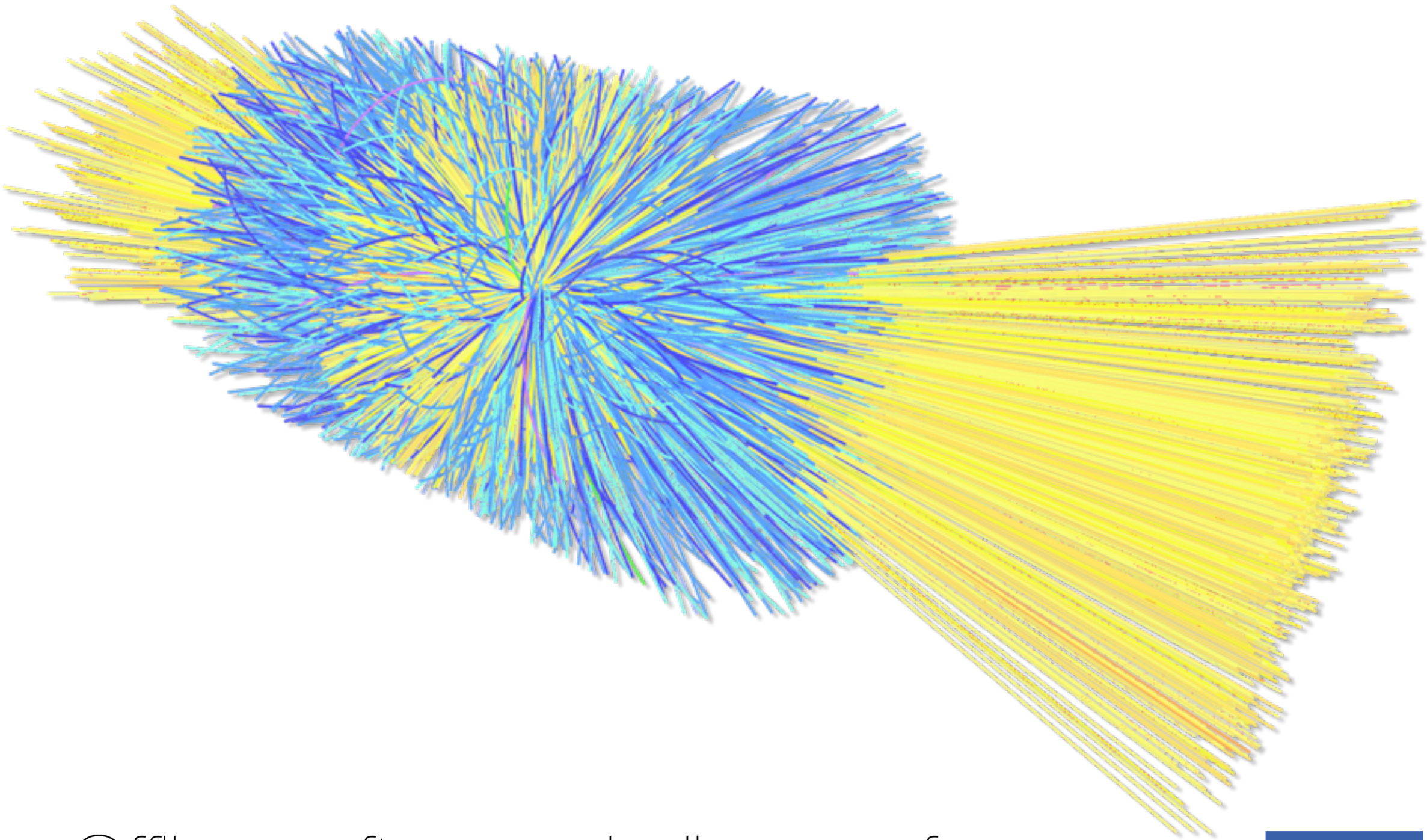


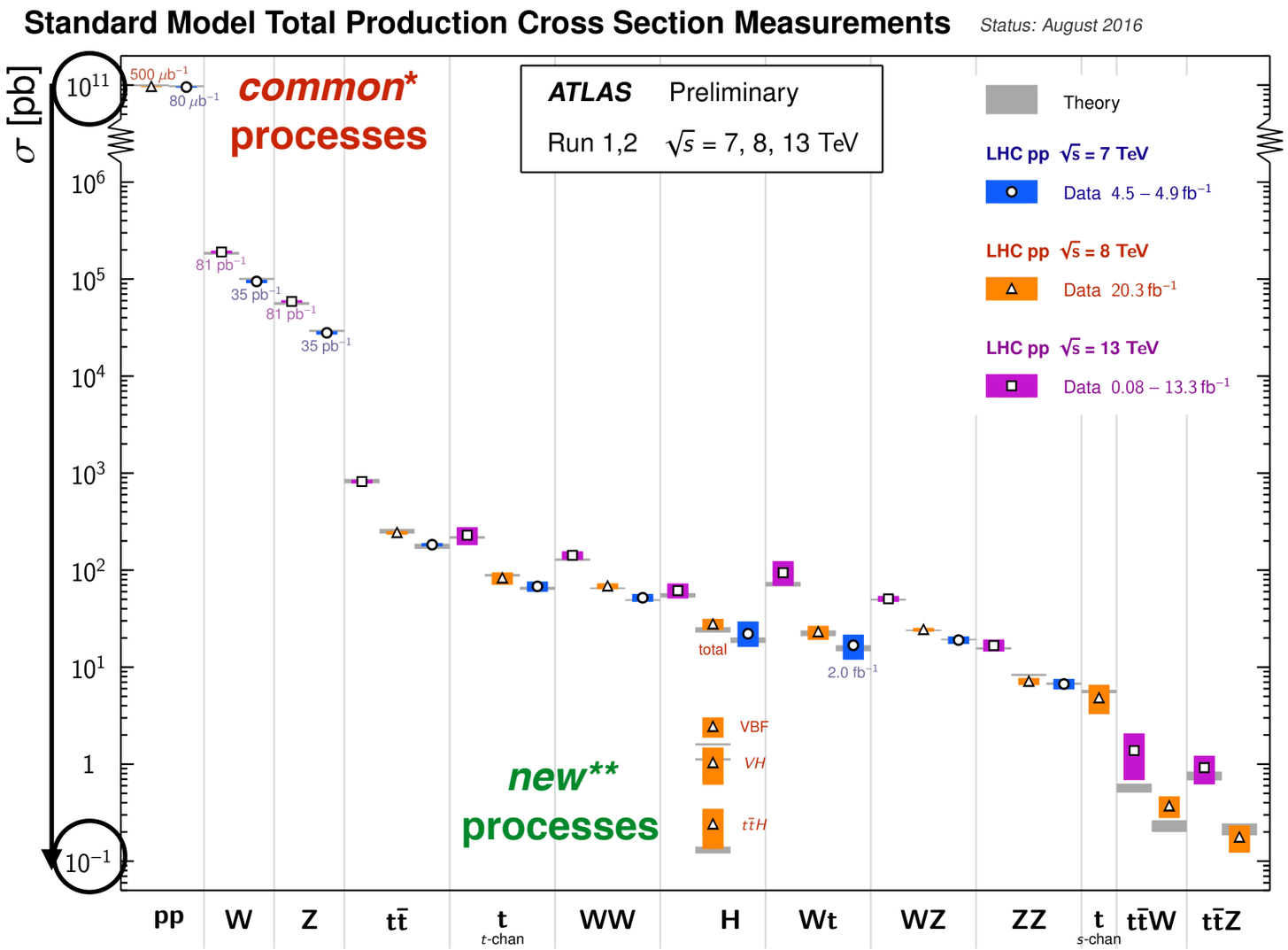
Offline software challenges for HL-LHC and FCC

A. Salzburger (CERN)

Environment

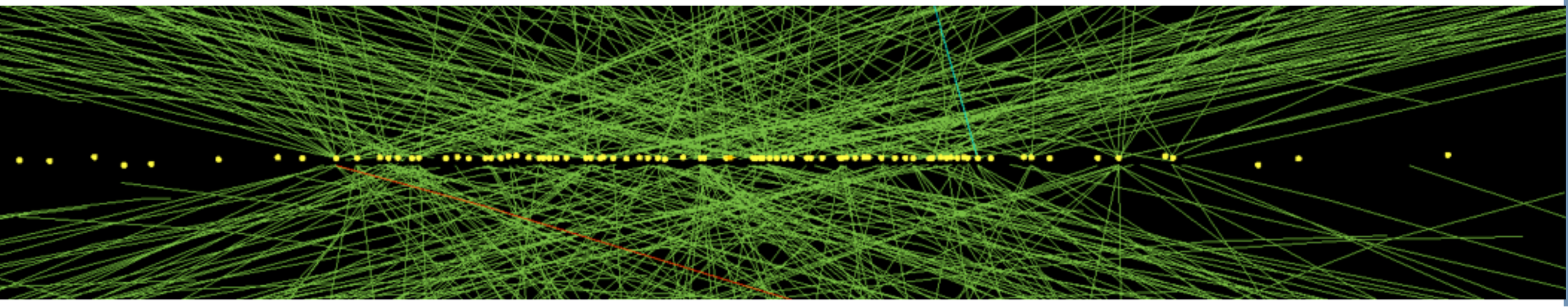
- HL-LHC beam parameters
  - expected pile-up (μ): 140-200
    (FCC: μ ~ 1000 in consideration)
  - bunch spacing of 25 ns

- Objective
  - do precision physics
    in this environment



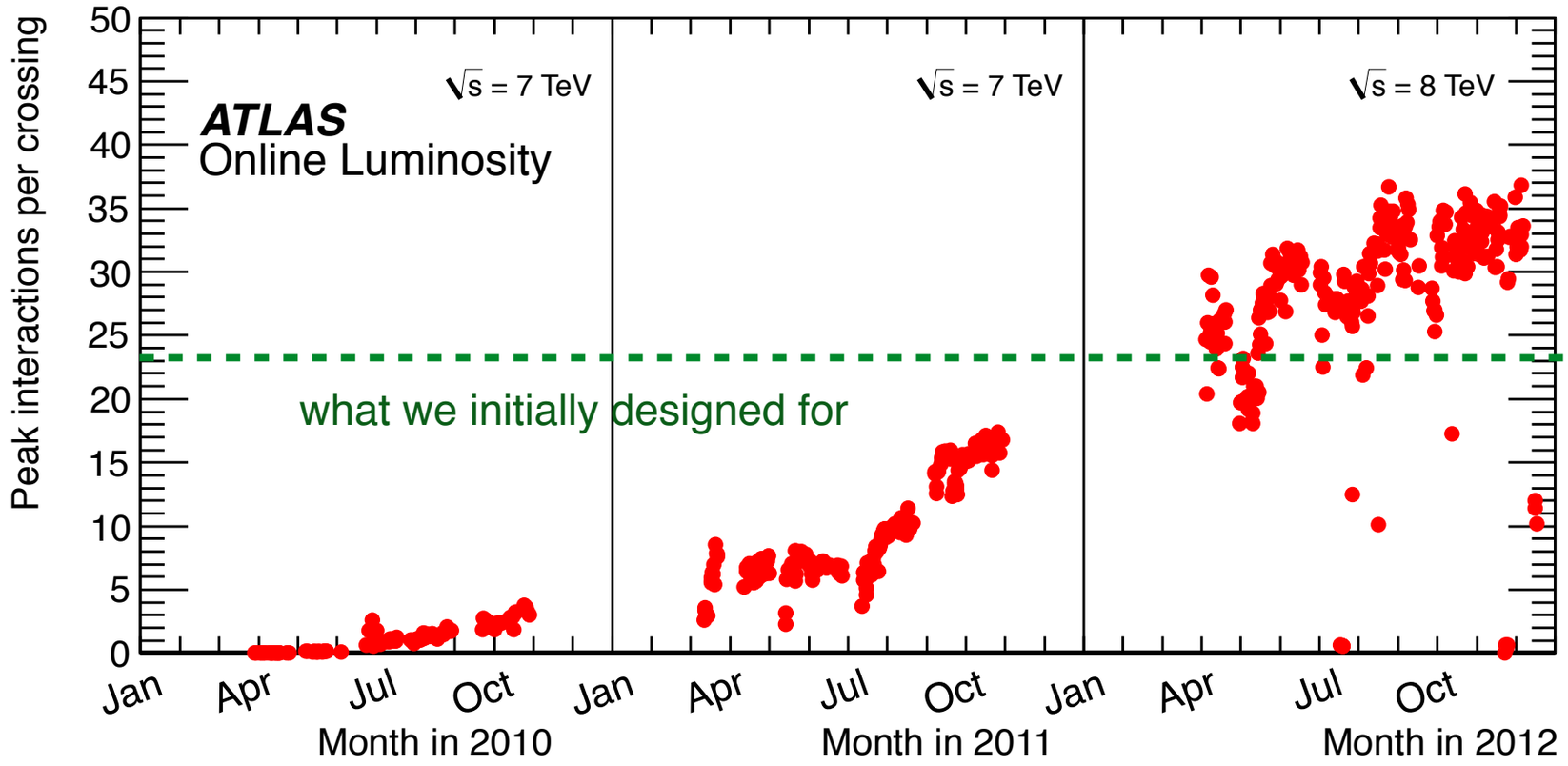
pile-up increases probability of new** processes

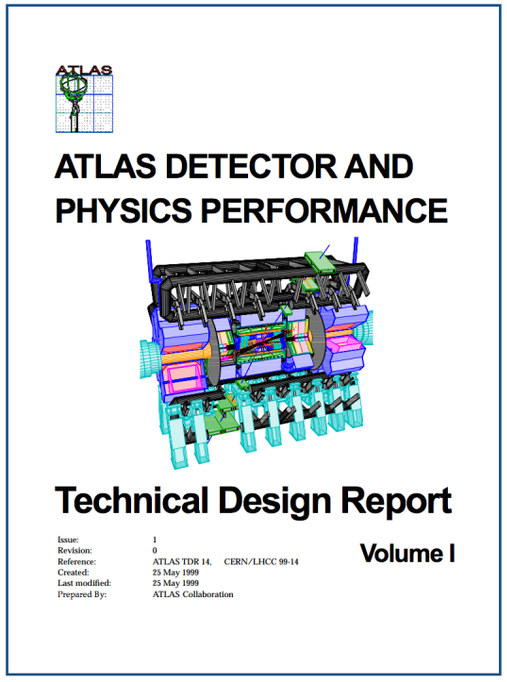
*)**) some people would say: common ~ boring, new ~ exciting



Pileup - the current environment

- Run-1/Run-2 already exceeded design-specifications for pile-up

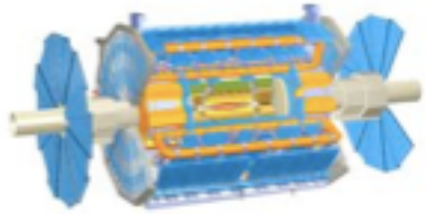
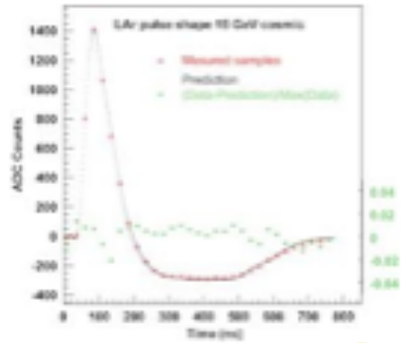
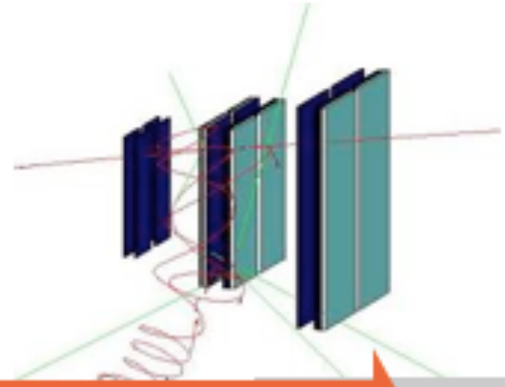




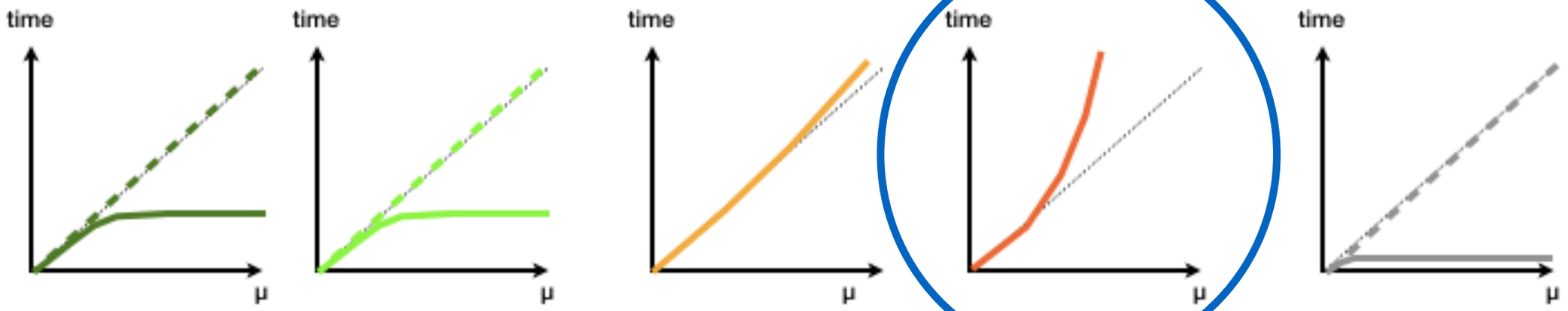
- HL-LHC is aiming for pile-up of 140 - 200

- Huge challenge for offline software
  - data volume / what to write on disk
  - event processing / what to process

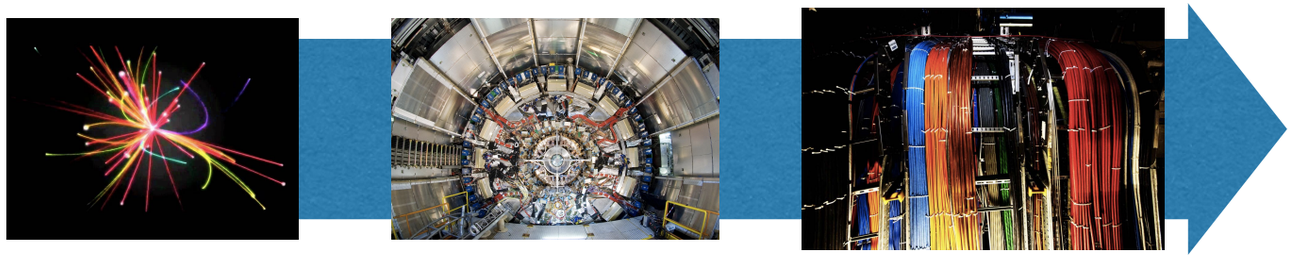
Pileup - CPU scaling


MC

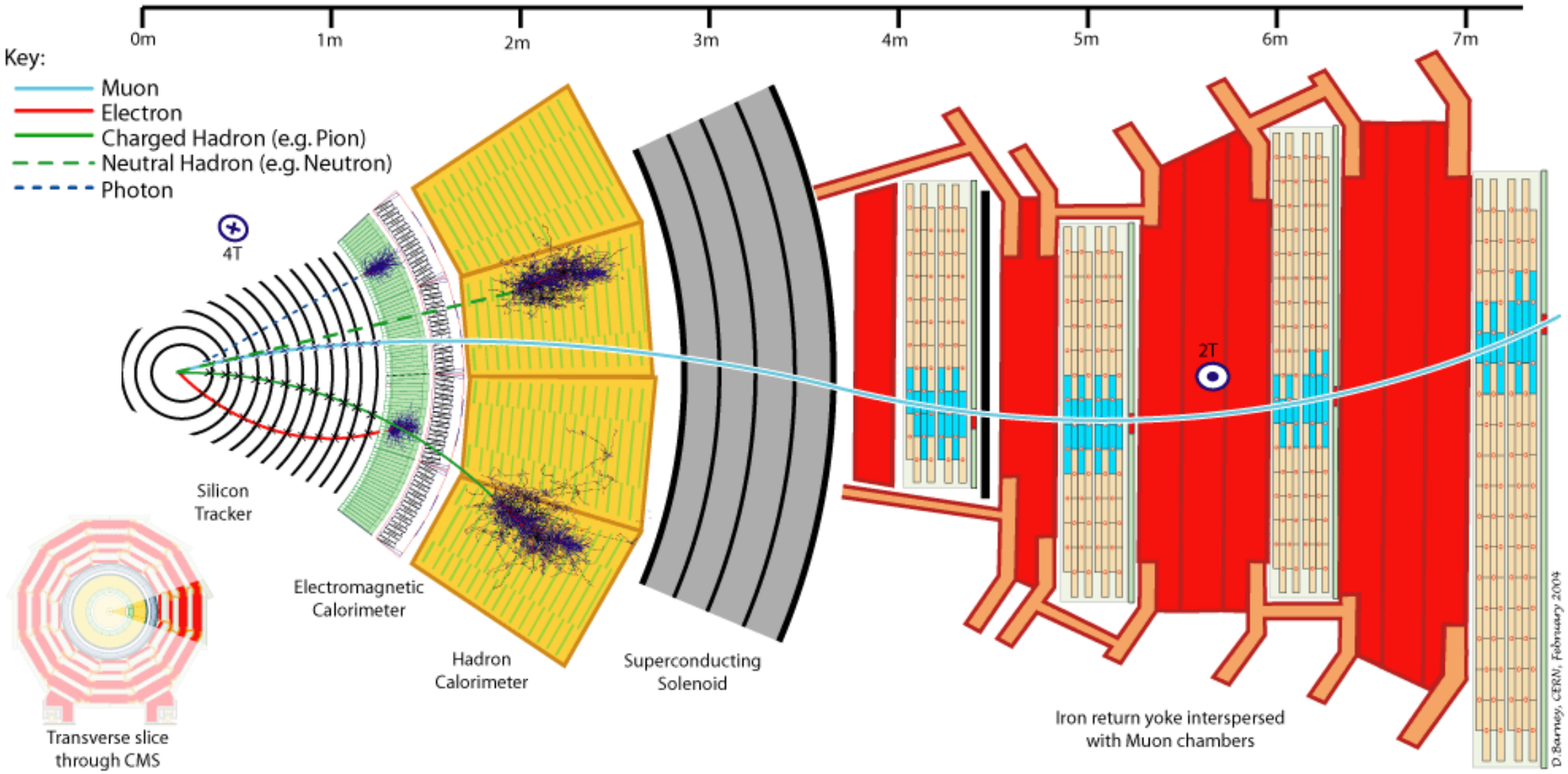


Data



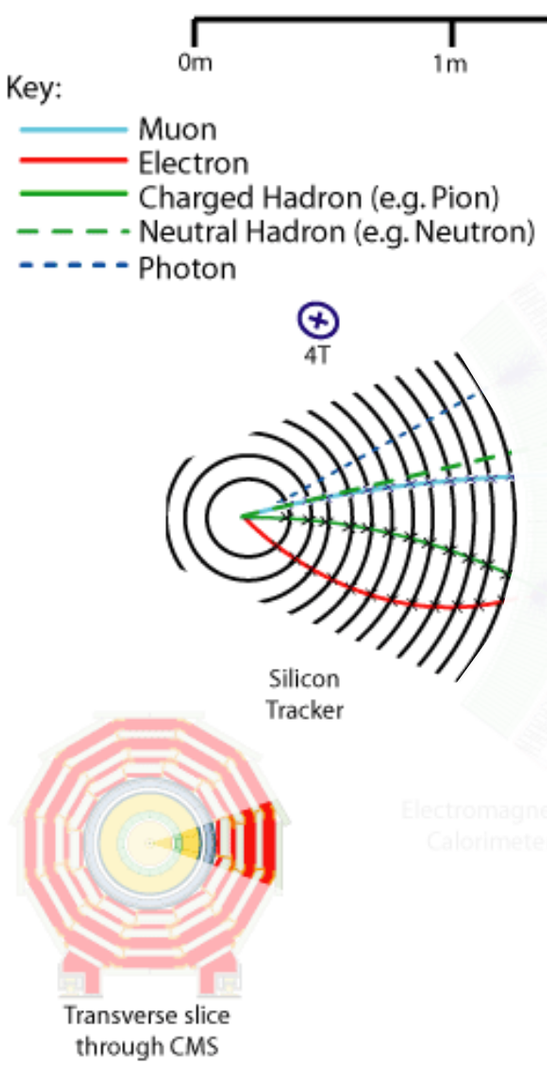
reconstruction time dominates at high pile-up

Event reconstruction



- process of analysing detector/simulation data into physics objects
  - different detectors need different algorithmic approaches
  - multi stage program with complex data flow

Event reconstruction - Inner Tracker

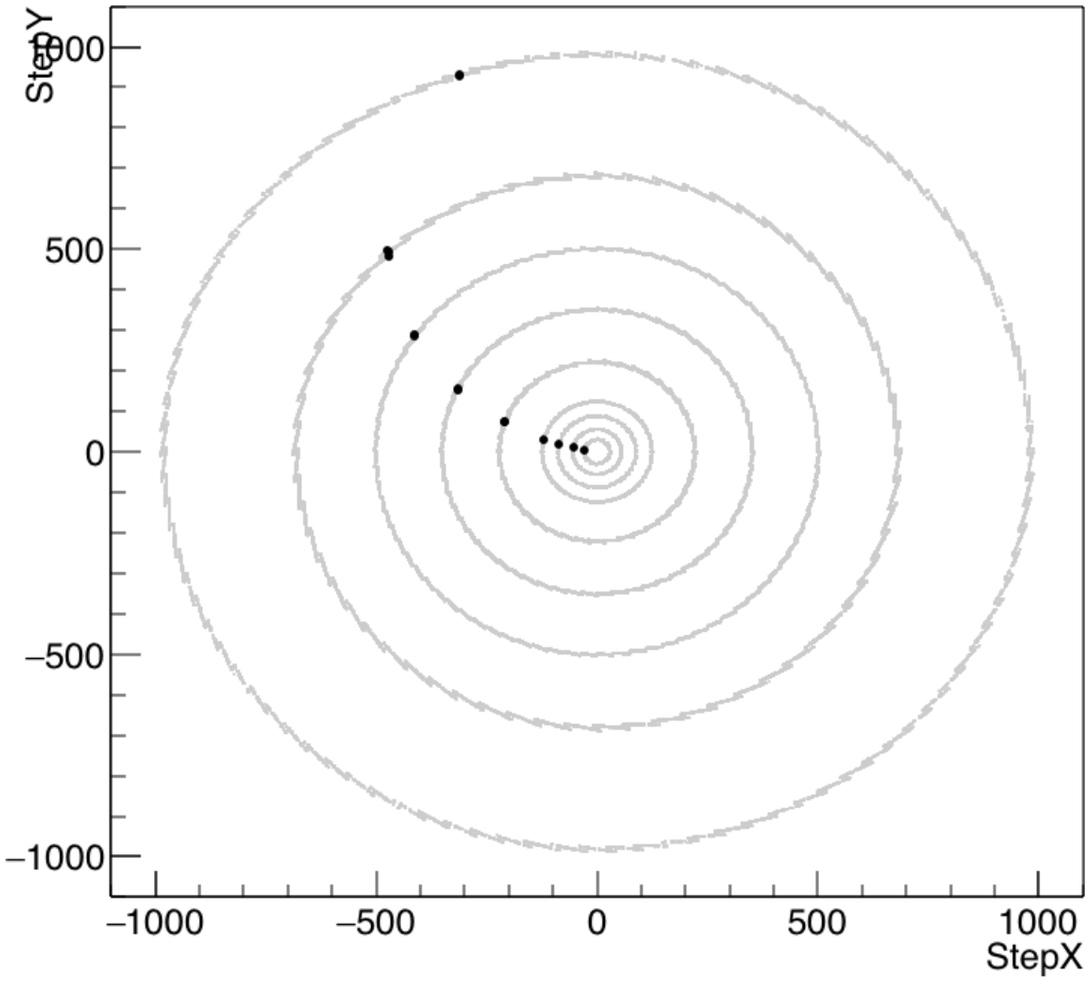


"Track reconstruction"

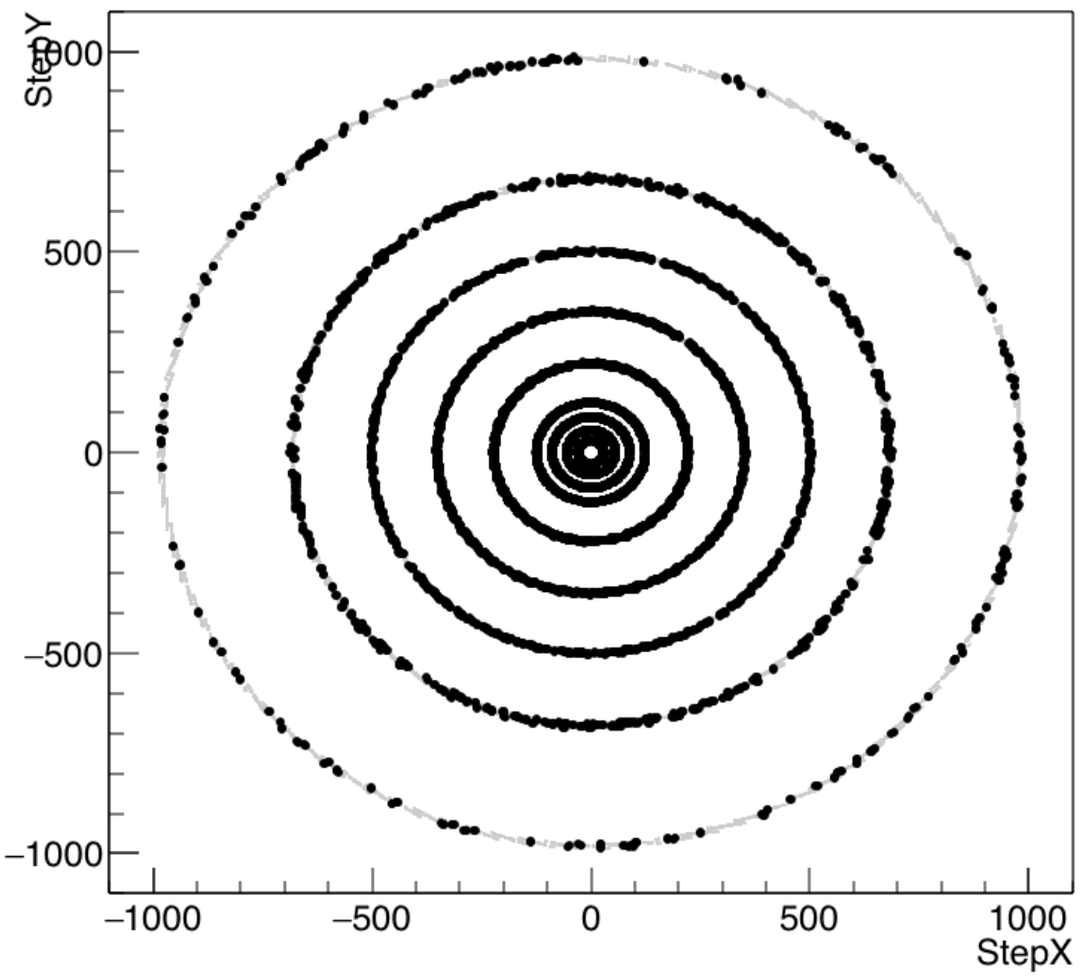
- finding ensembles of hits created by one particle
  - highly combinatorial problem
  - global and local pattern recognition methods
- fitting of the trajectories using a track model
  - least-square estimator based fitting
- track classification
  - quality requirements on track candidates

→ **dominates CPU** time of full event reconstruction

Combinatorics (1)



hits from 1 particle

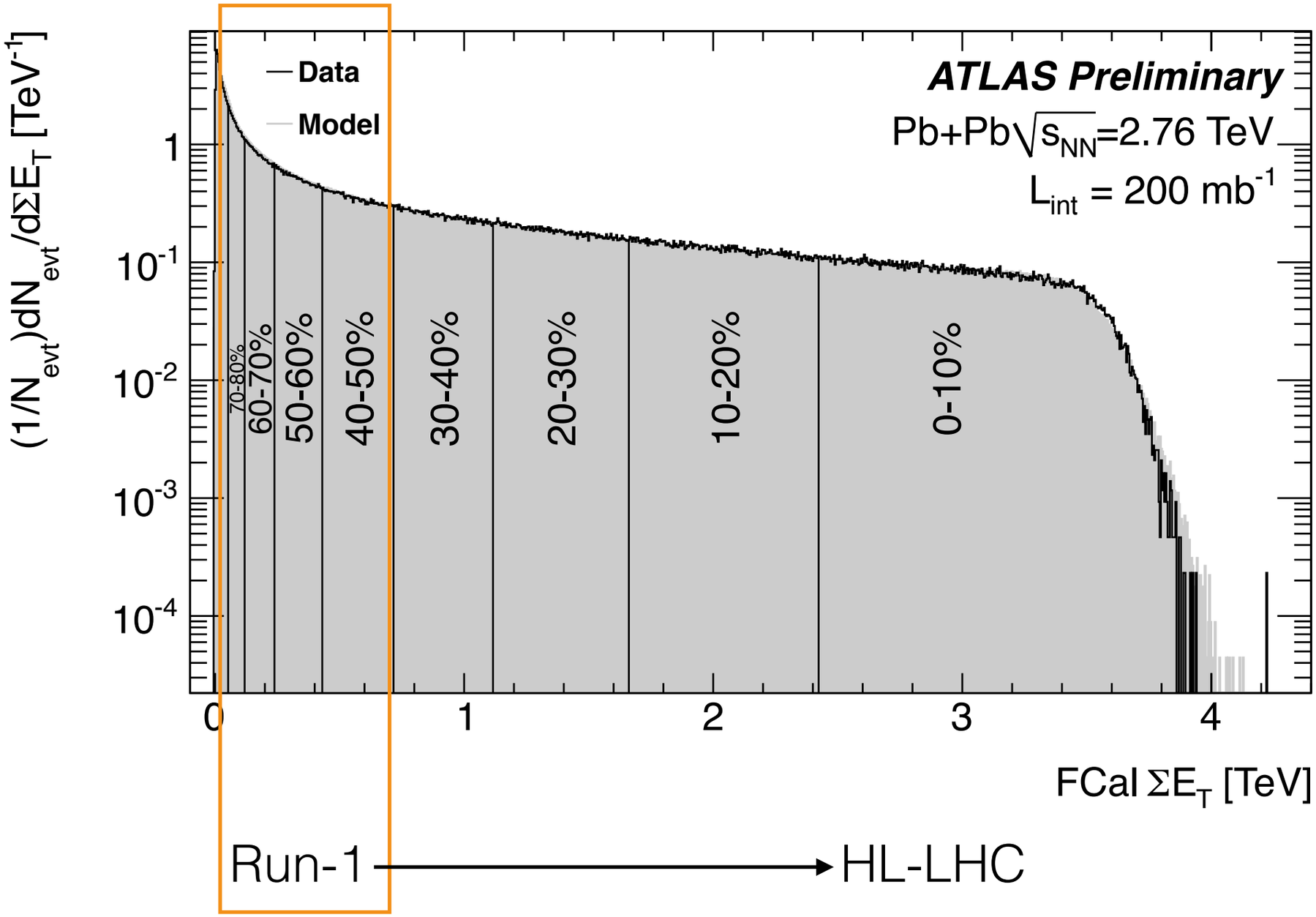


fraction of hits
from particles
in 200 pile-up events

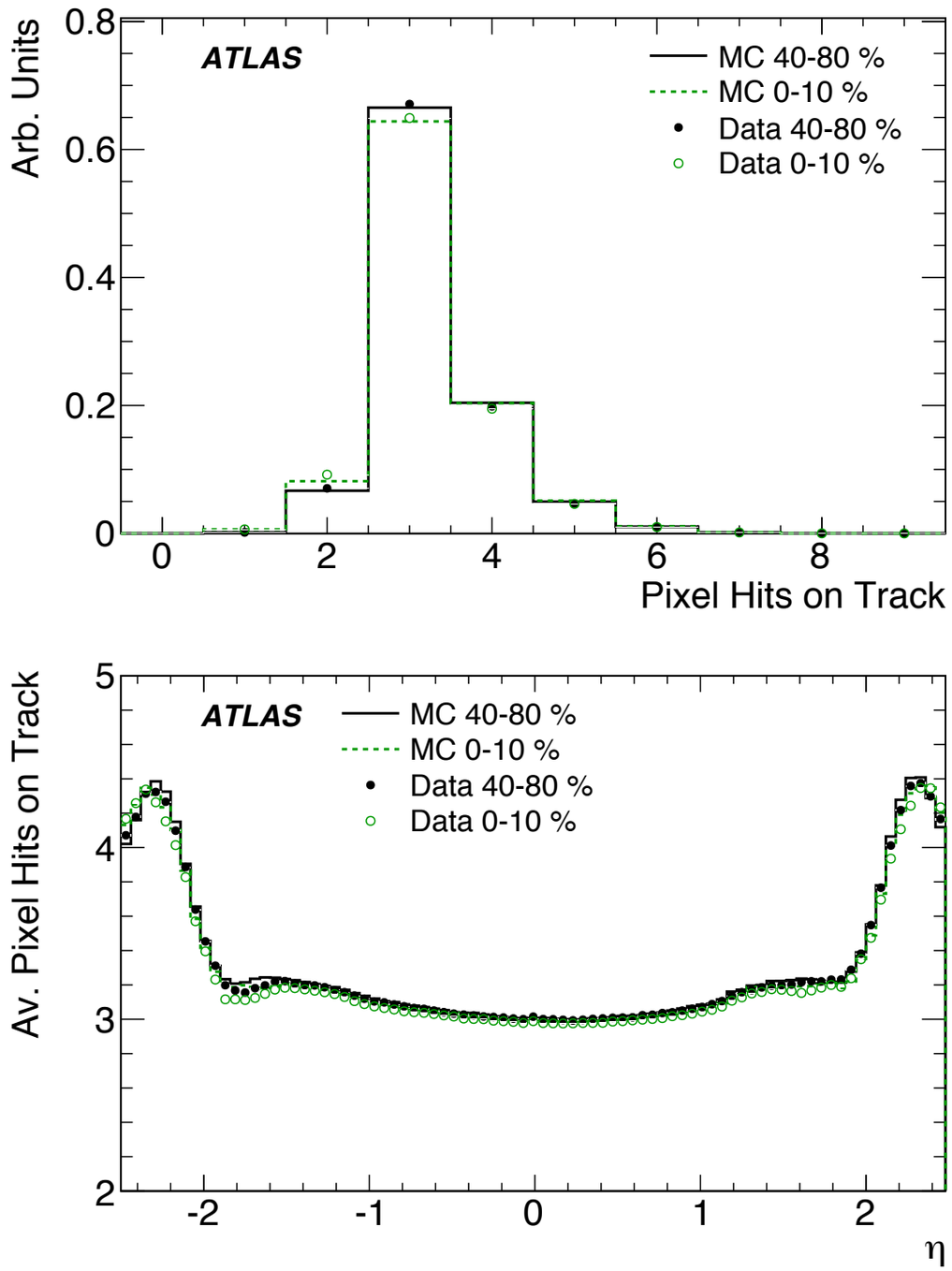
(total number of hits O(10k))

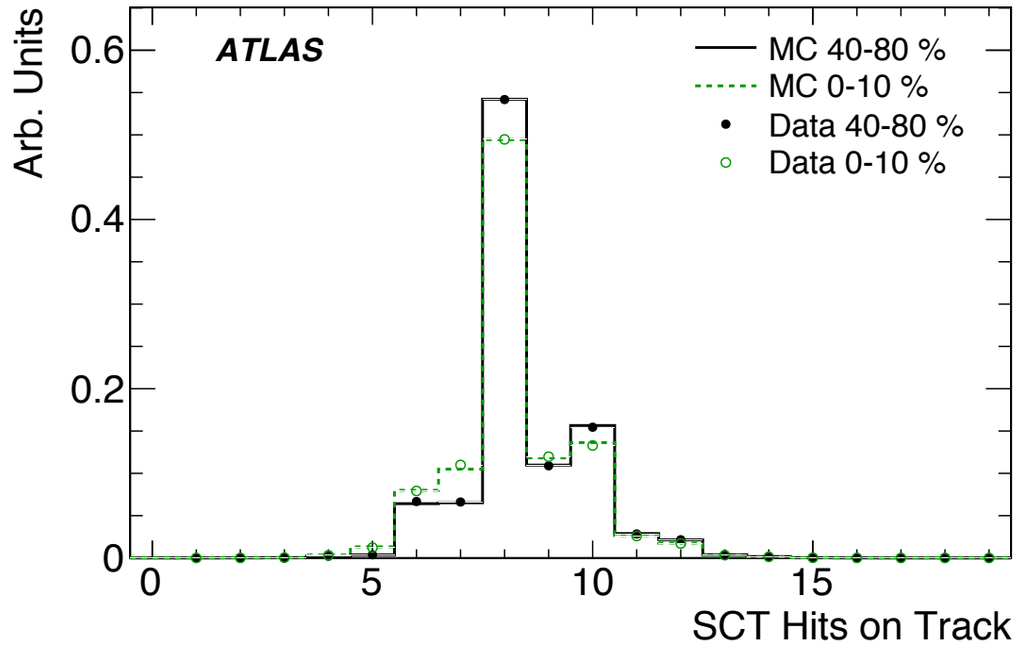
Combinatorics (2)

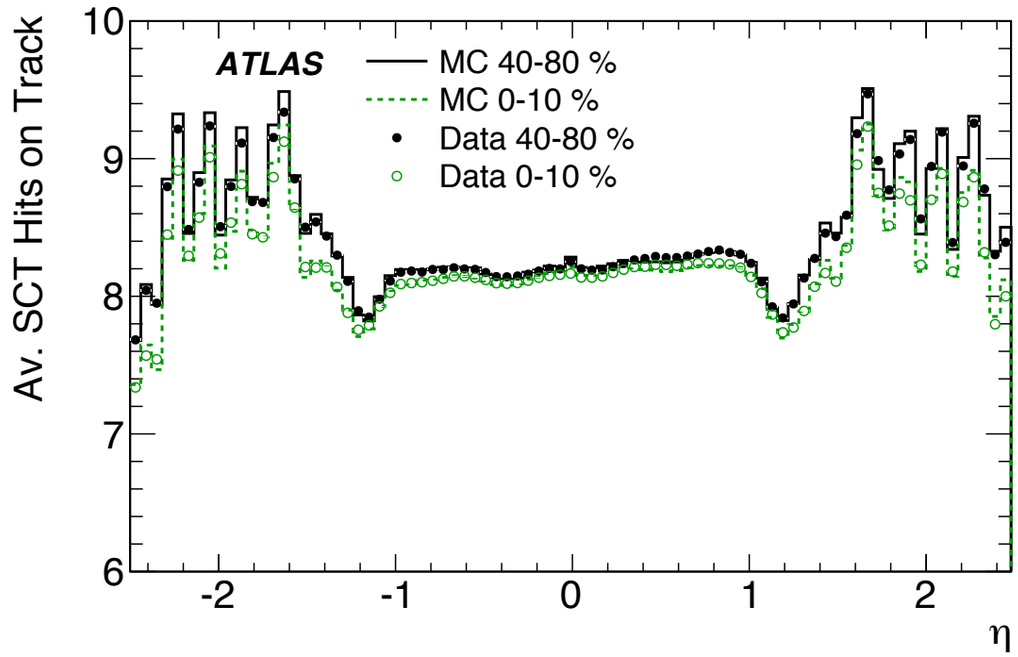
- Is the track finding our problem ?
  - **NO,** we have proven to be able to find tracks with high accuracy in events with even more activity than 200 pile-up: **heavy-ion collisions**



- Showed very stable performance of the track reconstruction
  - however, without single-vertex constraint, processing time was very long
  - finding: **HL-LHC environment not a physics performance but CPU challenge**

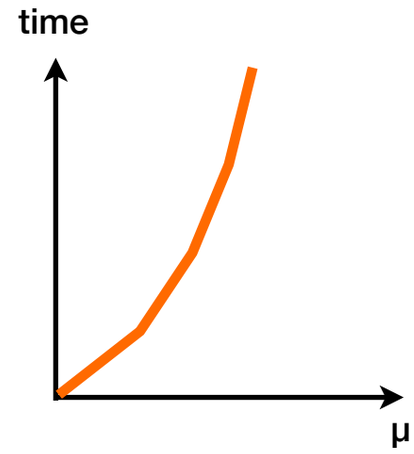






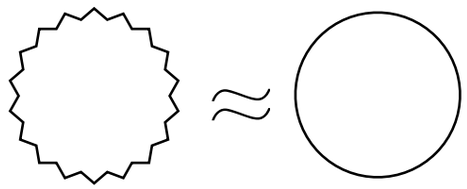
Challenges

- **CPU consumption** is our real challenge
  - (and data volume, but this will not be covered here)
  - particularly the scaling with pile-up



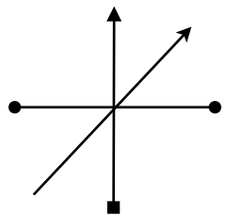
- Some of **algorithm concepts** and part of **code base** is 20 years old
  - we need both R&D in algorithms and solutions
    - is track following (backup) still the right approach for HL-LHC environment ?
    - what are alternatives ? Revival of the global pattern recognition ?
  - adapt our code to modern code patterns and architectures
    - make code fit for the future (maintenance, scaling, hardware)
    - how to adapt our code to be flexible towards changes in the market ?

Software/Algorithms - ways to speed up



approximate reality,
simplify your models

$\pi \approx 3$



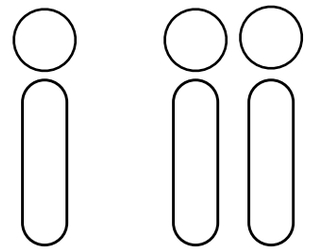
optimise & update your code

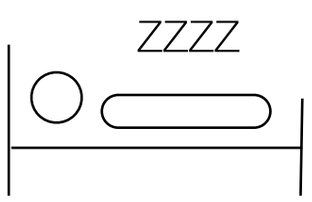
std::move(result)

1 €	2 DM
2 €	4 DM

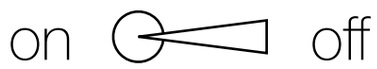
prepare your work,
use look-up tables

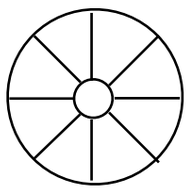
take shortcuts,
or simply cheat



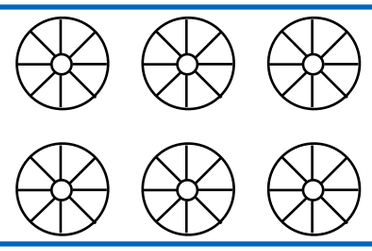


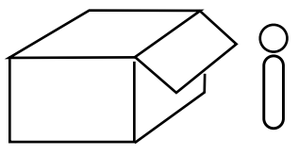
don't do anything, do less,
work on demand



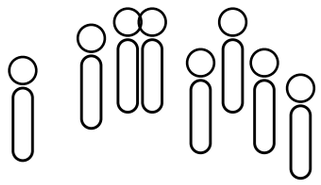


use new technologies,
increase your work force





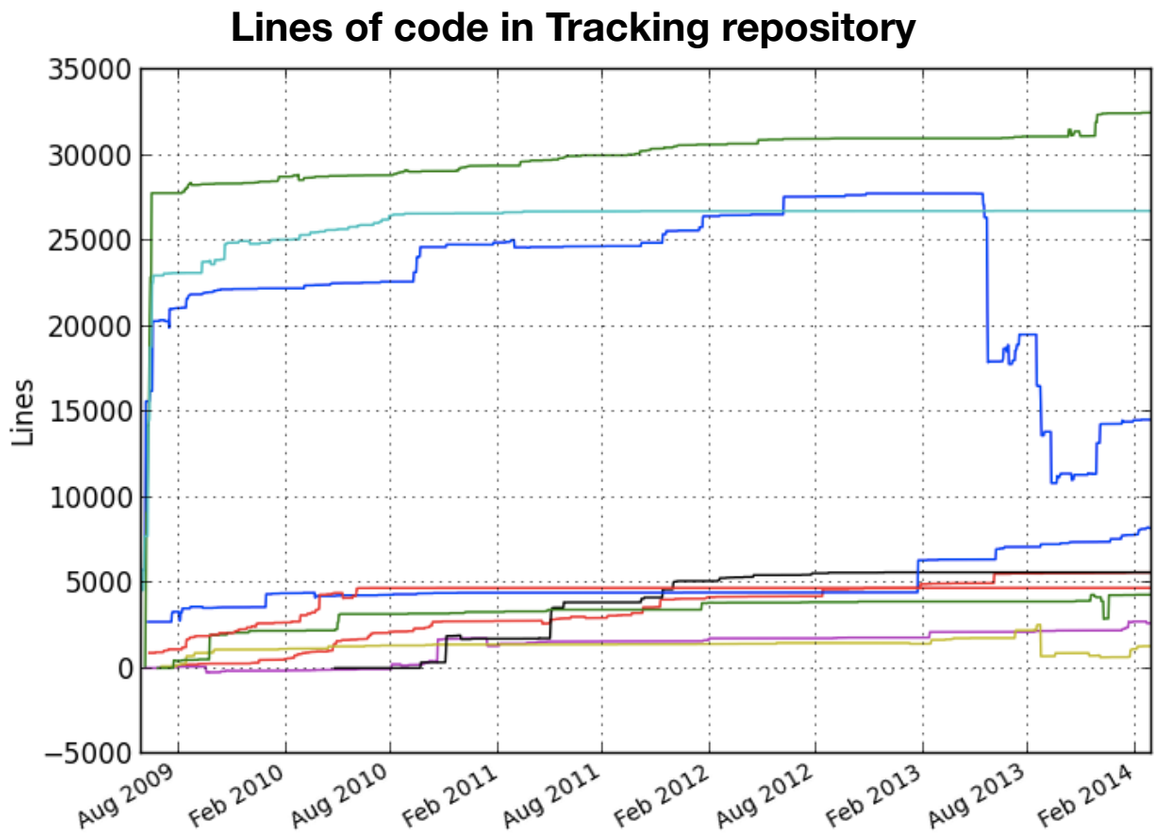
think outside the box,
learn from others



Review, clean up (1)

std::move(result)

- LHC Run-1 software was in some parts over-designed
  - developed before Run-1
  - left with a high level of flexibility, as written before commissioning
  - as a consequence, not always the optimal implementation used (particularly in terms of performance)

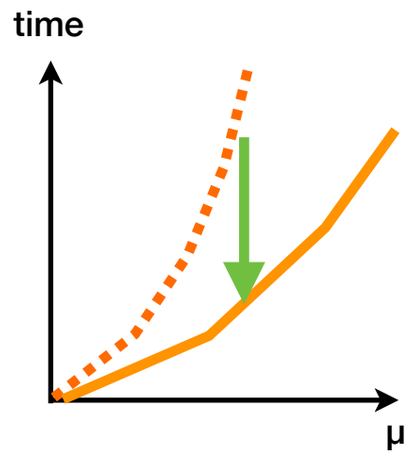


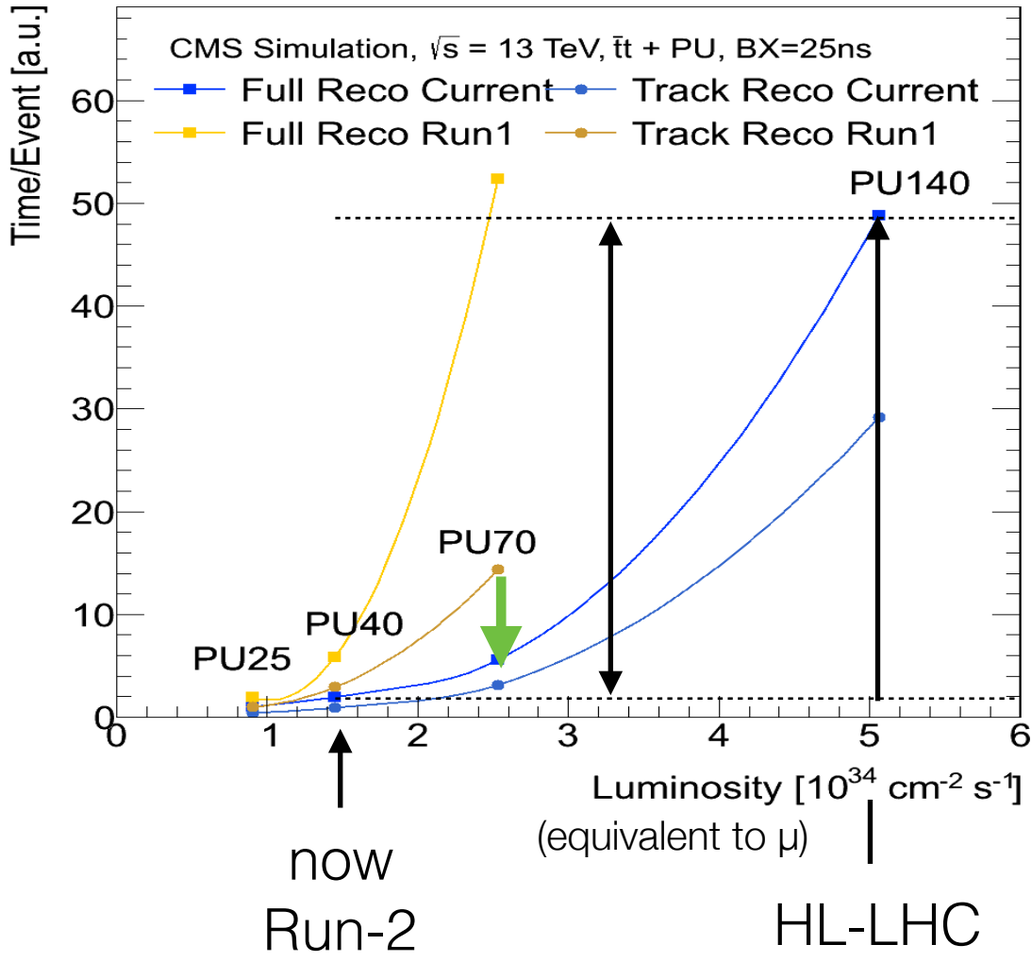
- Both, ATLAS & CMS started a big software cleanup campaign in LS-1
  - for ATLAS: simplified data structures look a lot like they looked in FORTRAN times :-P

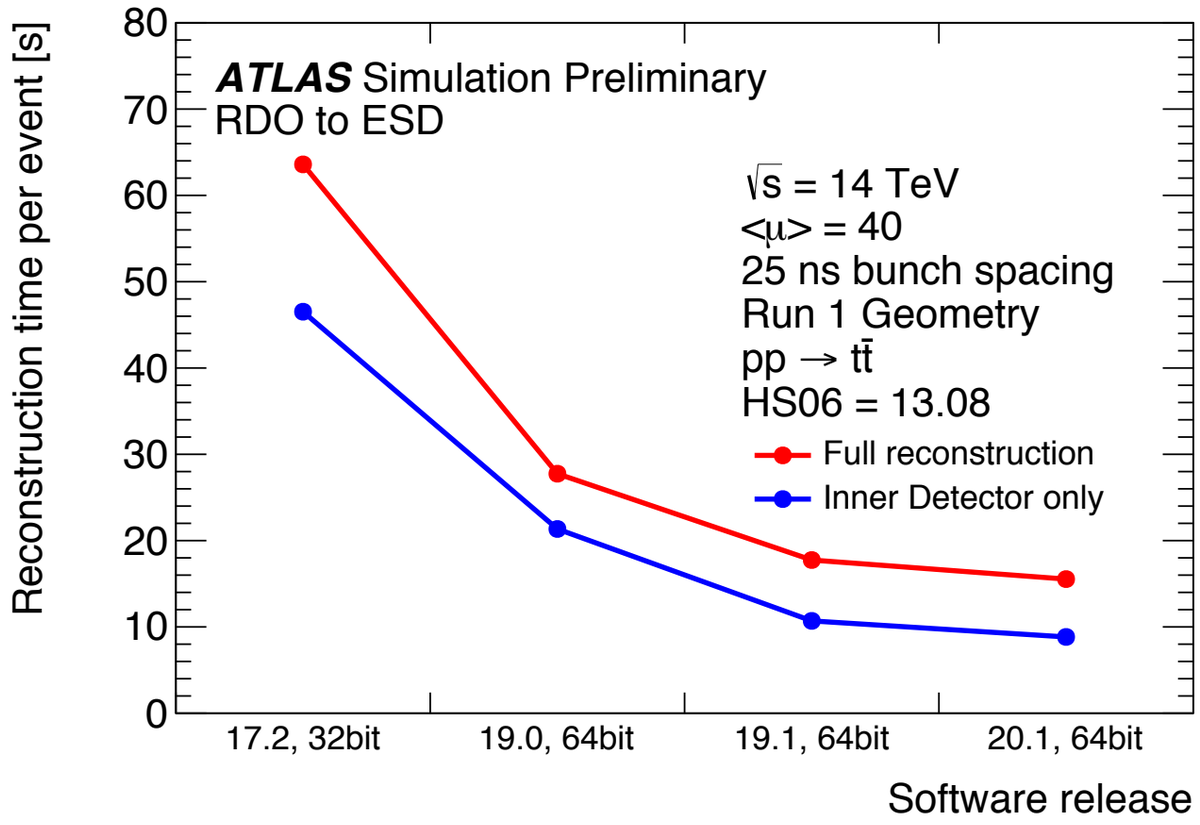
Review, clean up (2)

std::move(result)

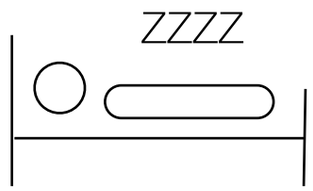
- Software cleanup gained factor ~4 for ATLAS/CMS
  - **drastic change of scaling** with pile-up achieved
  - no major hotspots left, similar gain
    can not be done by *"just code cleanup"*
  - highly non-linear scaling with pile-up stays







Software/Algorithms - do less/nothing



- HL-LHC (and further projects) raise the question what is worth to reconstruct
  - the challenge of HL-LHC will not be to reconstruct 90 % of the tracks, it's to reconstruct the right tracks
  - the physics challenge will be boosted environments (jets, heavy particles), b-tagging (even in the forward region)

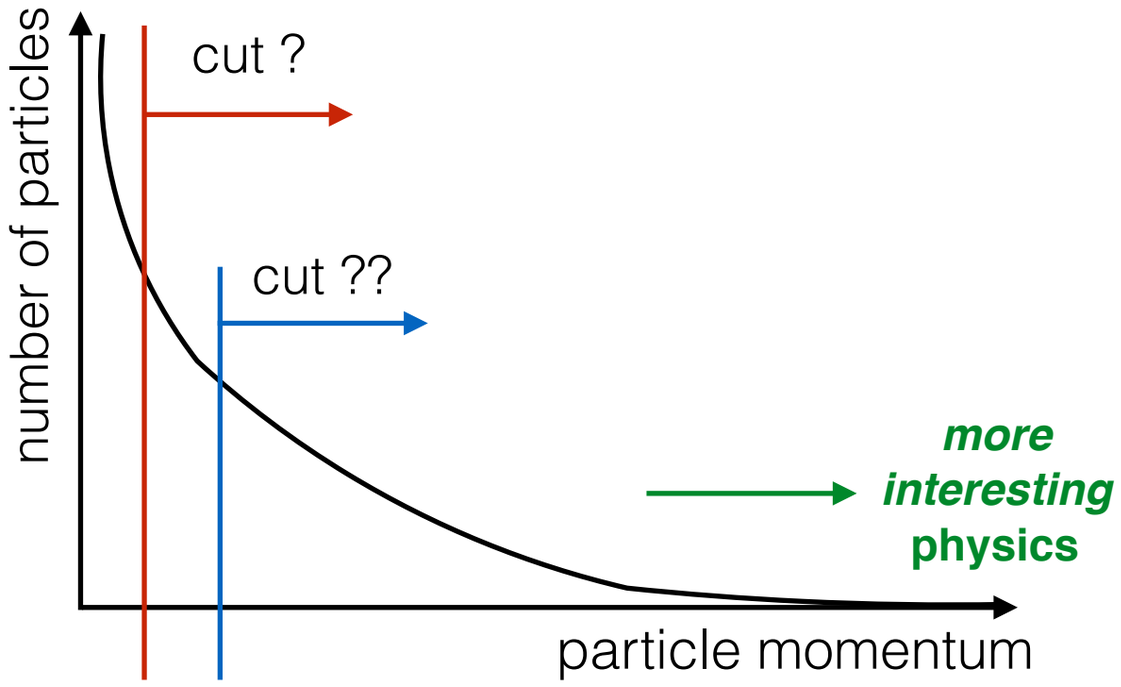
- Cutting on the momentum is cutting into physics - **ultima ratio**
  - Tracking still serves many combined reconstruction applications

    b-tagging

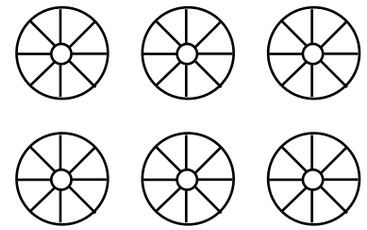
    (pile-up) vertex & jet tagging

    pile-up evaluation

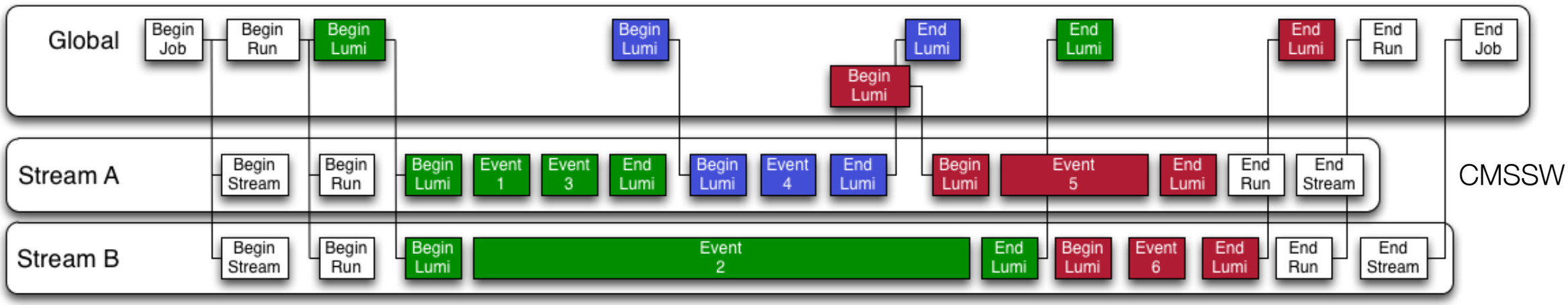
    particle flow/MET



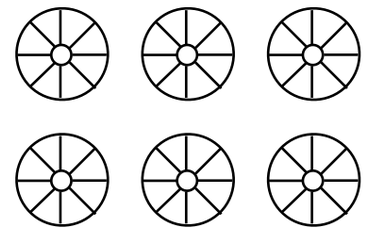
Software/Algorithms - go parallel (1)



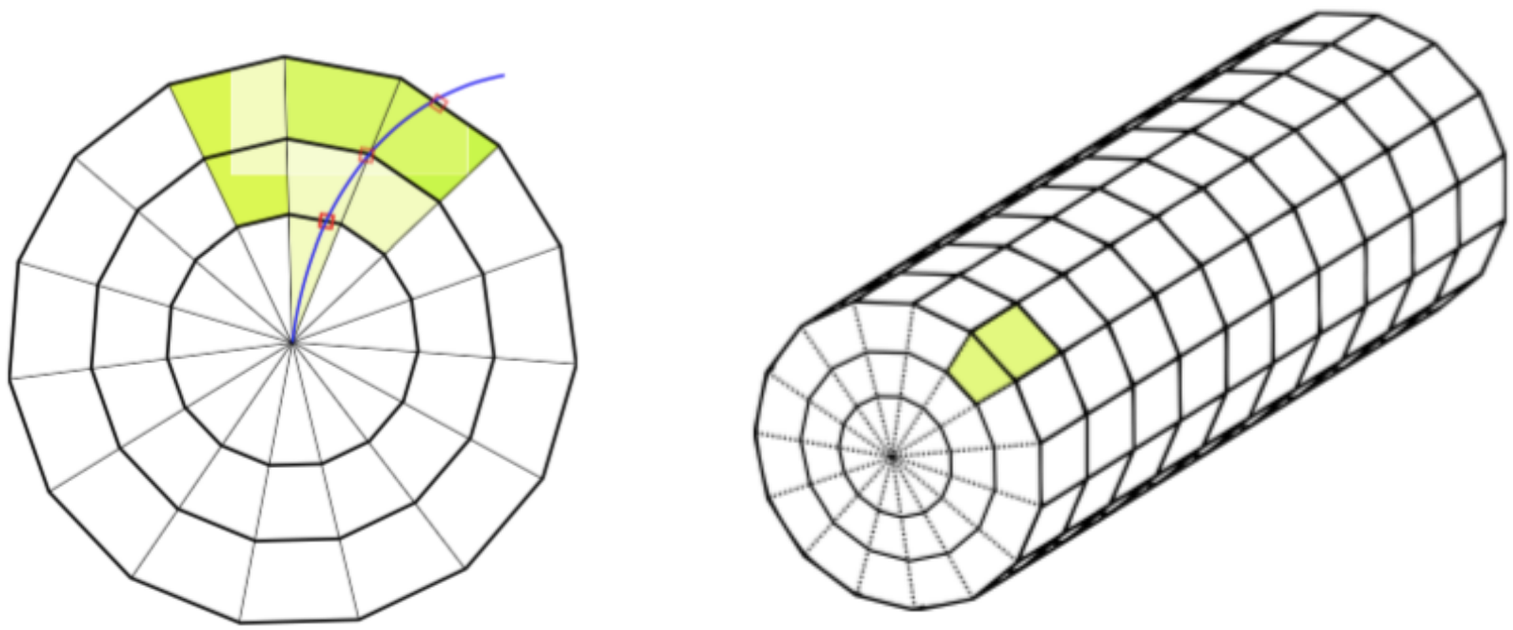
- Run-1/2 software was written (long) before 2008
  - **event processing frameworks** were not/little able to work with parallel loads
    - CMSSW started to move to a threaded framework during Run-1
      - supports single-threaded and multi-threaded components
    - ATLAS (GaudiHive) is in preparation for Run-3: event and within-event parallelism
      - currently ATLAS is running parallelism on event level as AthenaMP
  - **algorithmic code is/was largely not thread-safe:**
    - needs more than just *"const correctness"*
    - some algorithms may need rethinking
  - **internal vectorisation is very little used in reconstruction code**



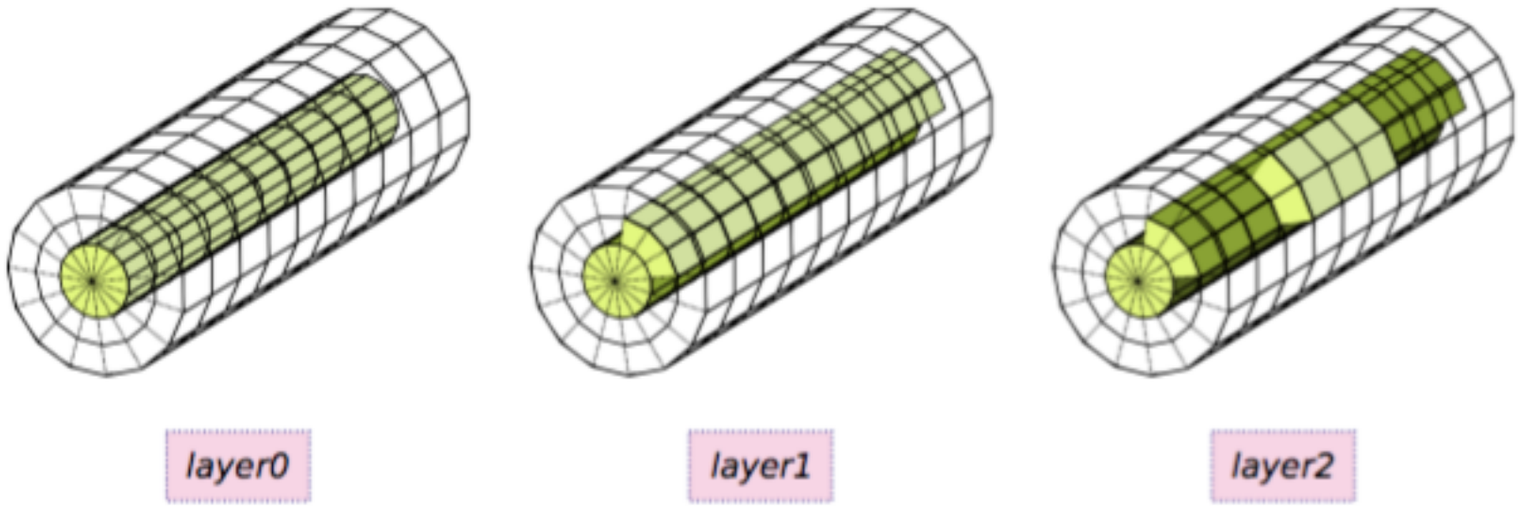
Software/Algorithms - go parallel (2)



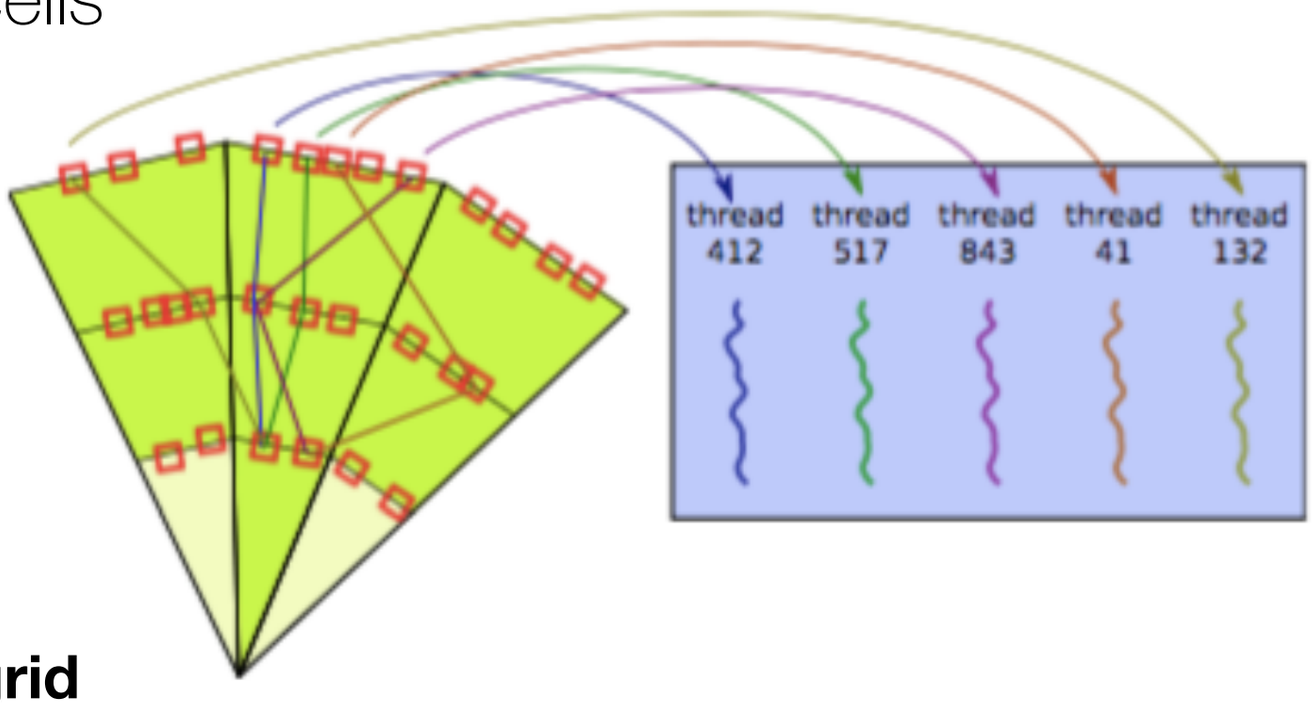
- Example: region-based reconstruction

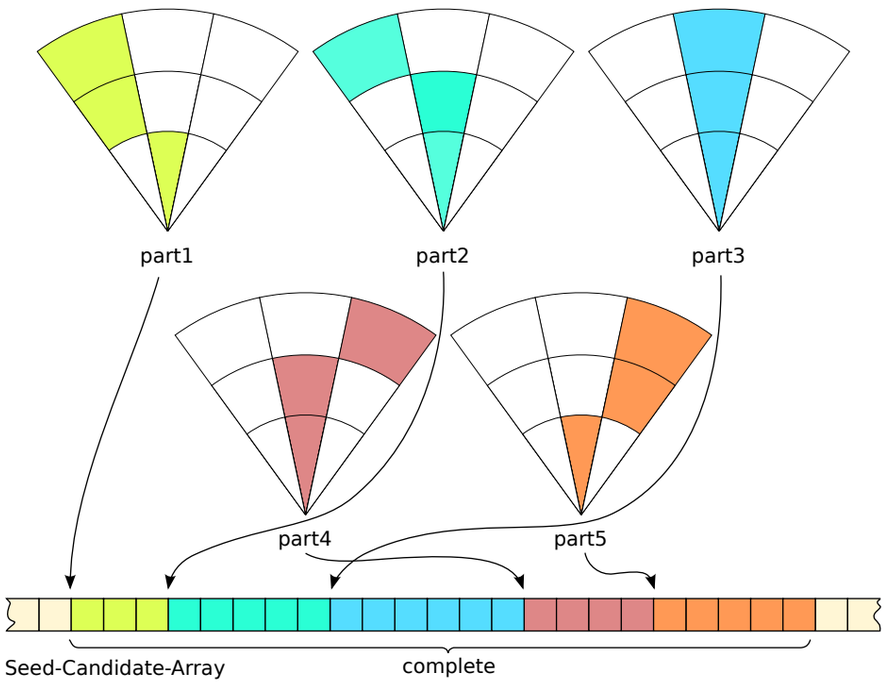


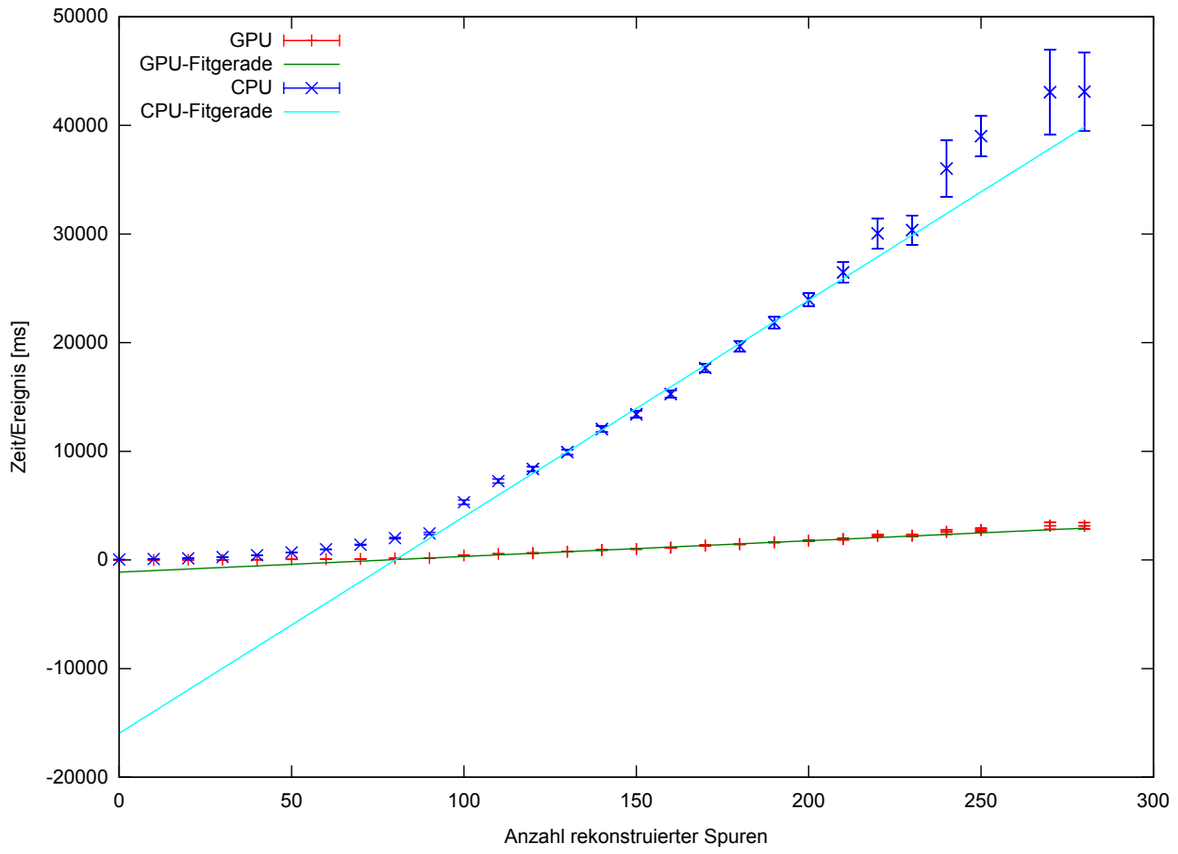
- Seed finding is done in these regions, moving in layers outwards
  - control the search regions with particle bending



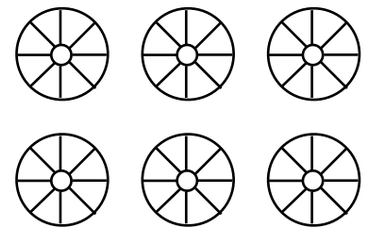
- Seed finding in geometrical cells
  - expands to neighbouring cell respecting the bending
  - cell size is related to minimal momentum cut
  - fork seeds into different threads, e.g. on GPU cores
  - we do not have **full control of grid hardware** (GPU/no GPU)
  - needs overlap resolving **- gain ?**







Software/Algorithms - go parallel (4)



- Parallel operations exploiting **vector registers**
  - price to pay: code get's *"less readable"* *(way out: VecGeom/VecCore ?)*

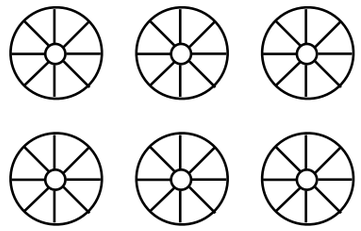
```
for(int i=0; i<42; i+=7) {
    double* dR   = &P[i];
    double* dA   = &P[i+3];

    double dA0   = H0[ 2]*dA[1]-H0[ 1]*dA[2];
    double dB0   = H0[ 0]*dA[2]-H0[ 2]*dA[0];
    double dC0   = H0[ 1]*dA[0]-H0[ 0]*dA[1];

    if(i==35) {dA0+=A0; dB0+=B0; dC0+=C0;}

    double dA2   = dA0+dA[0];
    double dB2   = dB0+dA[1];
    double dC2   = dC0+dA[2];

    double dA3   = dA[0]+dB2*H1[2]-dC2*H1[1];
    double dB3   = dA[1]+dC2*H1[0]-dA2*H1[2];
    double dC3   = dA[2]+dA2*H1[1]-dB2*H1[0];

    if(i==35) {dA3+=A3-A00; dB3+=B3-A11; dC3+=C3-A22;}

    double dA4   = dA[0]+dB3*H1[2]-dC3*H1[1];
    double dB4   = dA[1]+dC3*H1[0]-dA3*H1[2];
    double dC4   = dA[2]+dA3*H1[1]-dB3*H1[0];

    if(i==35) {dA4+=A4-A00; dB4+=B4-A11; dC4+=C4-A22;}

    double dA5   = dA4+dA4-dA[0];
    double dB5   = dB4+dB4-dA[1];
    double dC5   = dC4+dC4-dA[2];

    double dA6   = dB5*H2[2]-dC5*H2[1];
    double dB6   = dC5*H2[0]-dA5*H2[2];
    double dC6   = dA5*H2[1]-dB5*H2[0];

    if(i==35) {dA6+=A6; dB6+=B6; dC6+=C6;}

    dR[0]+=(dA2+dA3+dA4)*S3; dA[0]=(dA0+dA3+dA3+dA5+dA6)*.33333333;
    dR[1]+=(dB2+dB3+dB4)*S3; dA[1]=(dB0+dB3+dB3+dB5+dB6)*.33333333;
    dR[2]+=(dC2+dC3+dC4)*S3; dA[2]=(dC0+dC3+dC3+dC5+dC6)*.33333333;
}
```

```
for(int i = 0; i < 42; i+=7){
    __m256d dR = _mm256_loadu_pd(&P[i]);

    __m256d dA = _mm256_loadu_pd(&P[i + 3]);
    __m256d dA_201 = CROSS_SHUFFLE_201(dA);
    __m256d dA_120 = CROSS_SHUFFLE_120(dA);

    __m256d d0 = _mm256_sub_pd(_mm256_mul_pd(H0_201, dA_120), _mm256_mul_pd(H0_120, dA_201));

    if(i==35){
        d0 = _mm256_add_pd(d0, V0_012);
    }

    __m256d d2 = _mm256_add_pd(d0, dA);
    __m256d d2_201 = CROSS_SHUFFLE_201(d2);
    __m256d d2_120 = CROSS_SHUFFLE_120(d2);

    __m256d d3 = _mm256_sub_pd(_mm256_add_pd(dA, _mm256_mul_pd(d2_120, H1_201)), _mm256_mul_pd(d2_2
    __m256d d3_201 = CROSS_SHUFFLE_201(d3);
    __m256d d3_120 = CROSS_SHUFFLE_120(d3);

    if(i==35){
        d3 = _mm256_add_pd(d3, _mm256_sub_pd(V3_012, A_012));
    }

    __m256d d4 = _mm256_sub_pd(_mm256_add_pd(dA, _mm256_mul_pd(d3_120, H1_201)), _mm256_mul_pd(d3_2

    if(i==35){
        d4 = _mm256_add_pd(d4, _mm256_sub_pd(V4_012, A_012));
    }

    __m256d d5 = _mm256_sub_pd(_mm256_add_pd(d4, d4), dA);
    __m256d d5_201 = CROSS_SHUFFLE_201(d5);
    __m256d d5_120 = CROSS_SHUFFLE_120(d5);

    __m256d d6 = _mm256_sub_pd(_mm256_mul_pd(d5_120, H2_201), _mm256_mul_pd(d5_201, H2_120));

    if(i==35){
        d6 = _mm256_add_pd(d6, V6_012);
    }

    _mm256_storeu_pd(&P[i], _mm256_add_pd(dR, _mm256_mul_pd(_mm256_add_pd(d2, _mm256_add_pd(d3, d4)
    _mm256_storeu_pd(&P[i + 3], _mm256_mul_pd(C_012, _mm256_add_pd(d0, _mm256_add_pd(d3, _mm256_add
}
```

- Runge-Kutta integration but up to 2.4x faster using SIMD instructions
- Future CPUs are expected to further extend SIMD capacities

Roberto Agostino Vitillo

Software/Algorithms - go parallel (5)



- Prepare software toolkit for threat-safe processing
  - for example ACTS Tracking code project: http://acts.web.cern.ch/ACTS/
  - trying to collect the track reconstruction code from LHC experiments:

    move towards C++14 standard, simplify, document

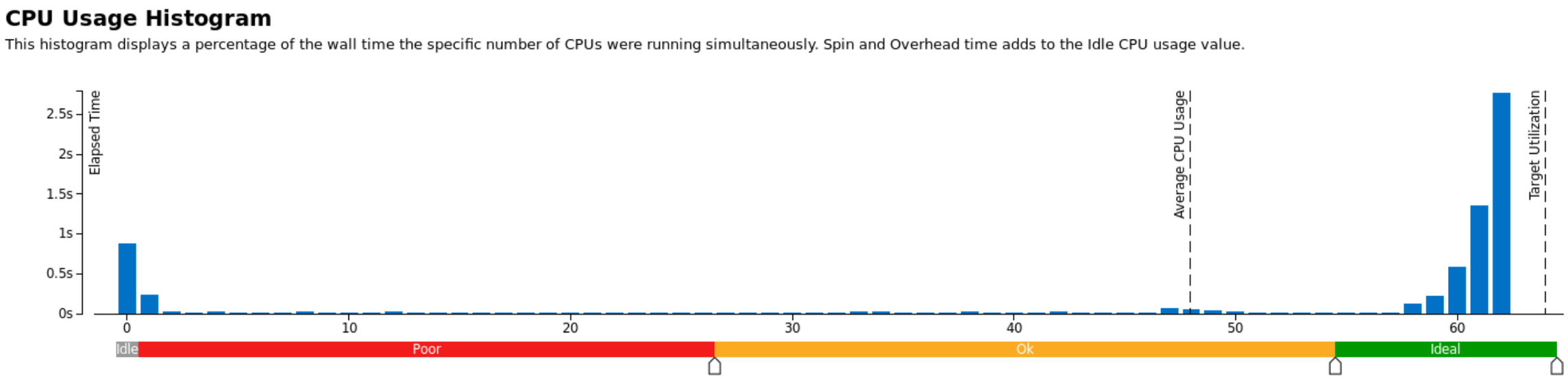
    test modules/components for thread-safety, CI test

    create a testbed for track reconstruction R&D (see ML slide)

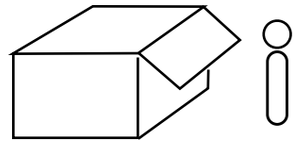
Example using intel VTune analysis for ACTS track extrapolation test

2nd CERN/Intel openlab workshop Nov 24-25, 2016

Running on Intel Xeon e5-2698 v3



Outside the box - Machine learning ?



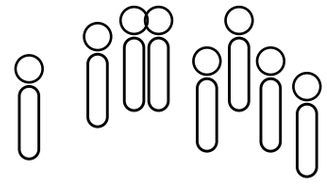
- Feature/pattern finding is not a unique problem to HEP
  - machine learning (ML) techniques are on the rise for e.g. image (pattern) recognition
  - big players are on the market pushing these

- HEP has already experience with ML techniques
  - usage of ML was mostly focussed on analysis techniques so far
  - used extensively in classification problems (also in track reconstruction)

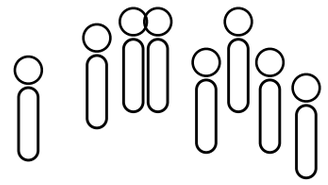
- HEP also has experience with ML challenges
  - see, e.g. very successful Higgs ML challenge: https://www.kaggle.com/c/higgs-boson
  - new Tracking ML in preparation, trying to learn from the community

    can we teach the computer to find us the tracks ?

    is that any faster ?



Tracking ML challenge



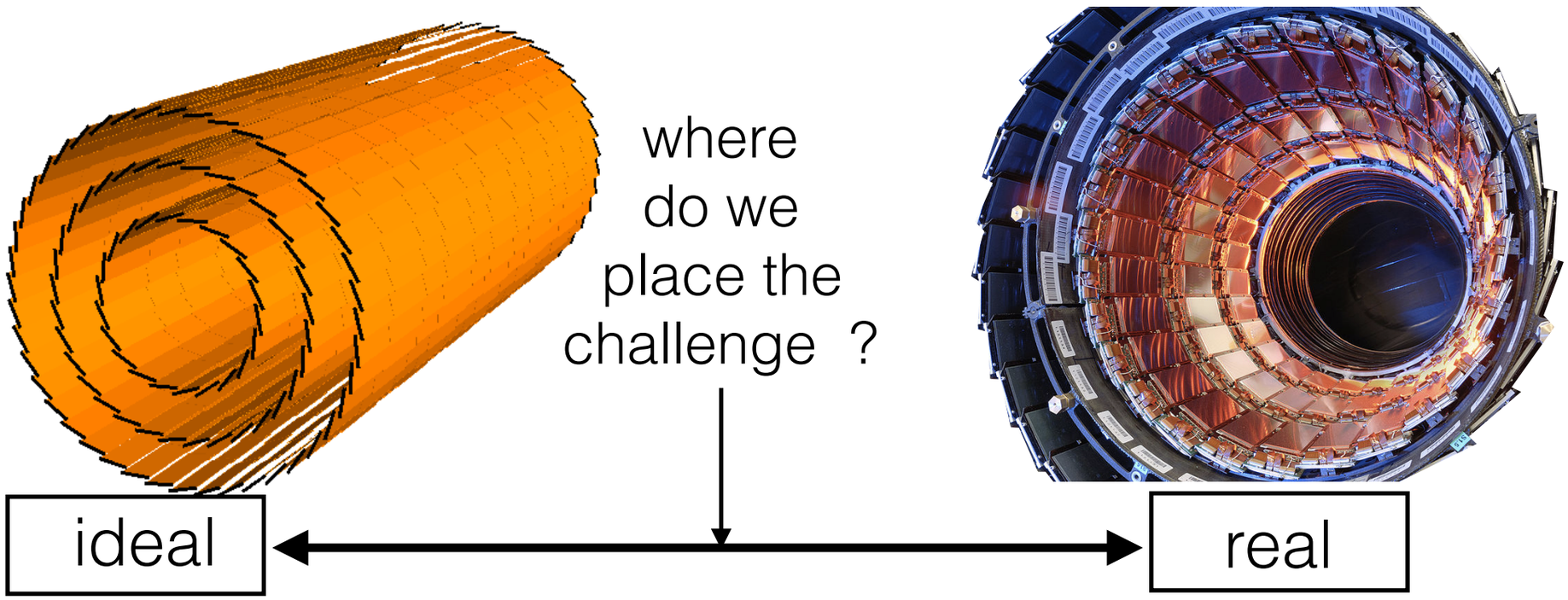
- Currently in preparation phase
  - **abstraction of the problem** to open up for outside experts

    keeping it simple enough while making it realistic enough for us to learn something
  - **formulation of the problem**

    what do we want to be solved, what is a good solution to the problem ?
  - **preparation of the problem**

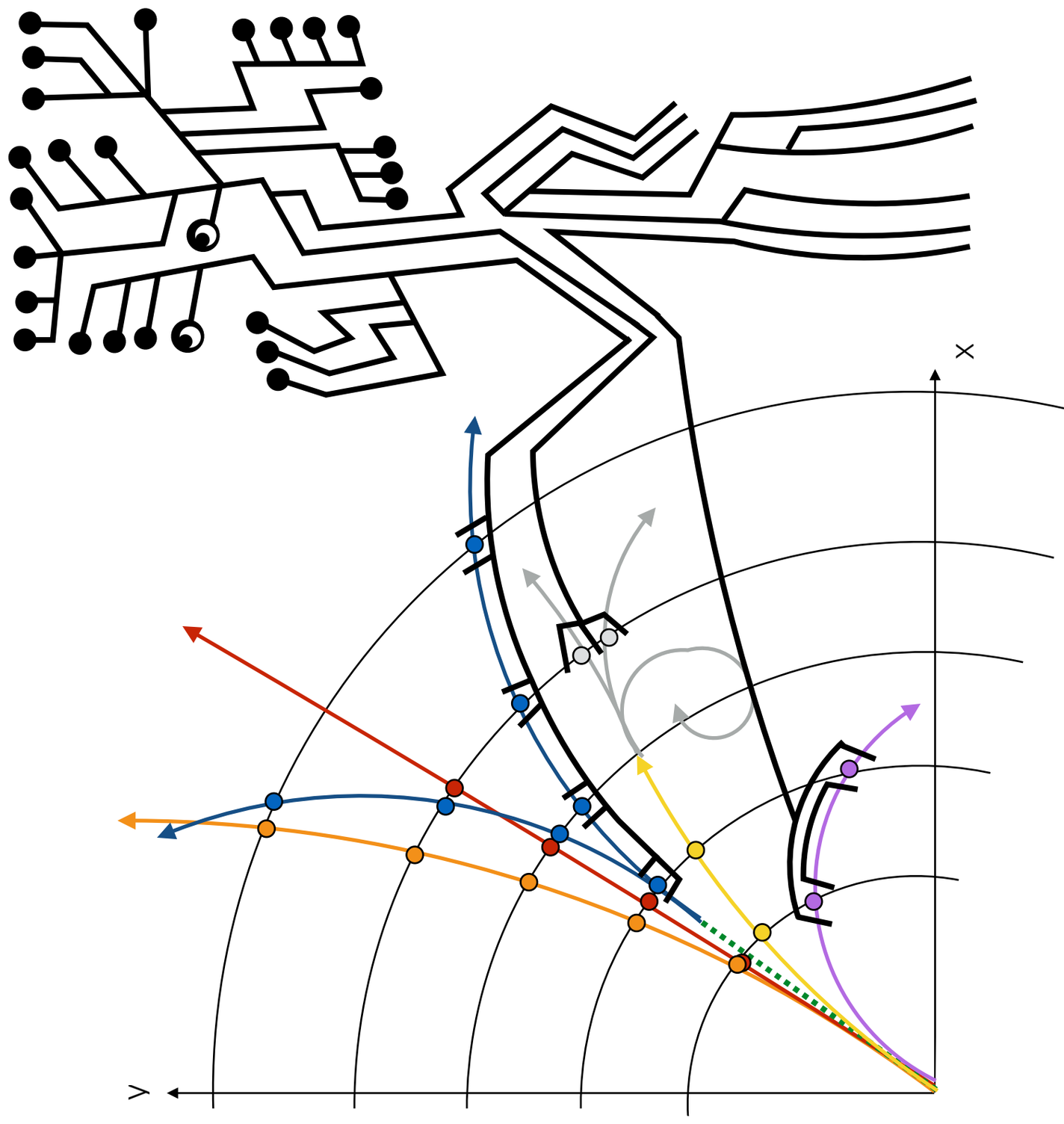
    training dataset, test dataset, starter kit

    platform, measure (target function & CPU measure)



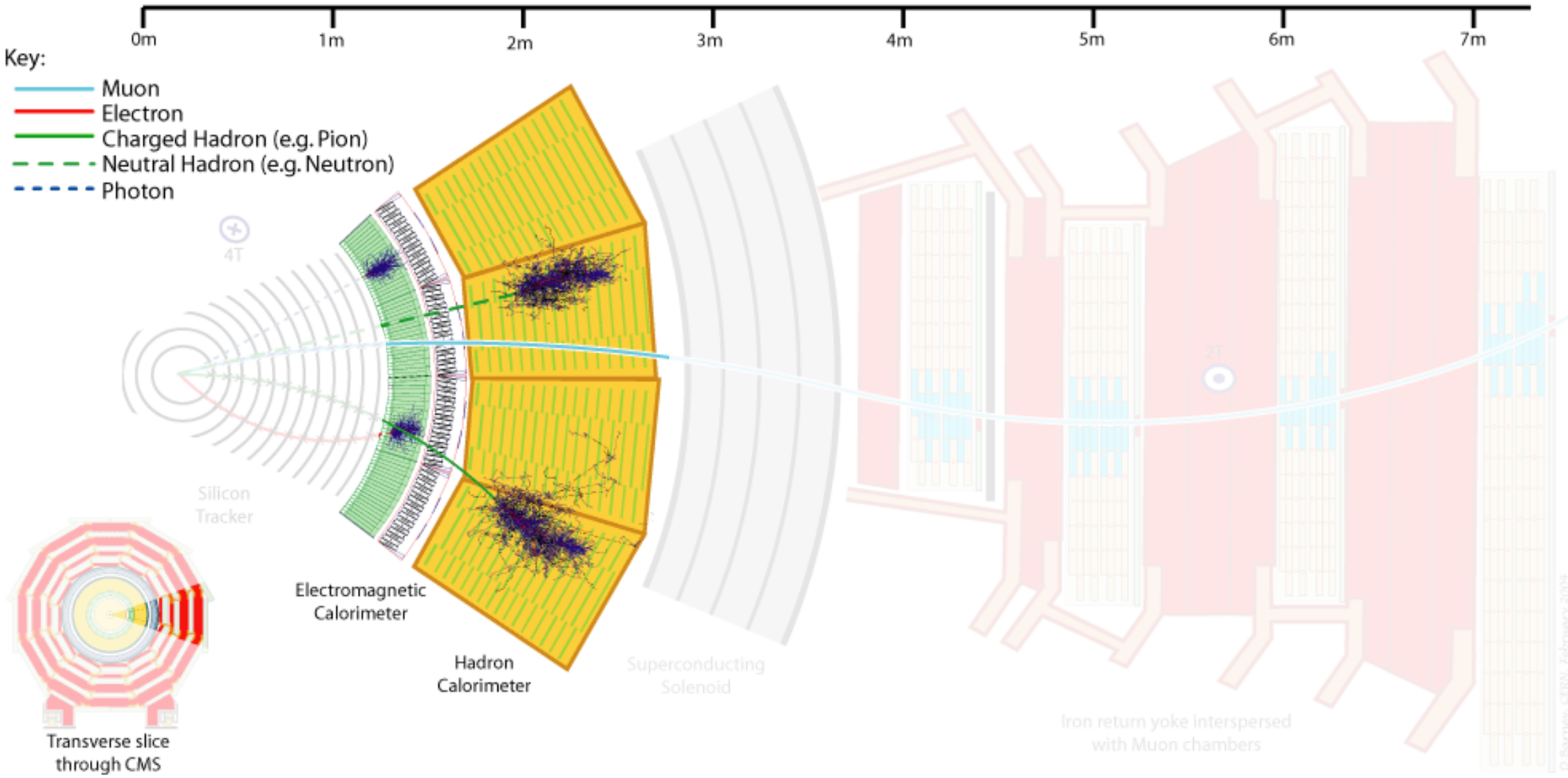
Summary

- Inner tracker reconstruction is the CPU driver for HL-LHC scenario
  - due to combinatorics from many O(10k) hits at input
- LHC experiments invested a lot of effort in improving the software
  - no hotspots left, no low hanging fruits to grab
  - some current limitations due to Run-1 software design left
- New concepts/technologies need to be followed
  - event processing frameworks update for parallel processing on the way/done
  - code update to be thread safe ongoing
  - R&D for pattern recognition at very high pile-up scenarios
- Summary of the summary
  - a lot to do on all ends


x
y

Some backup slides to follow

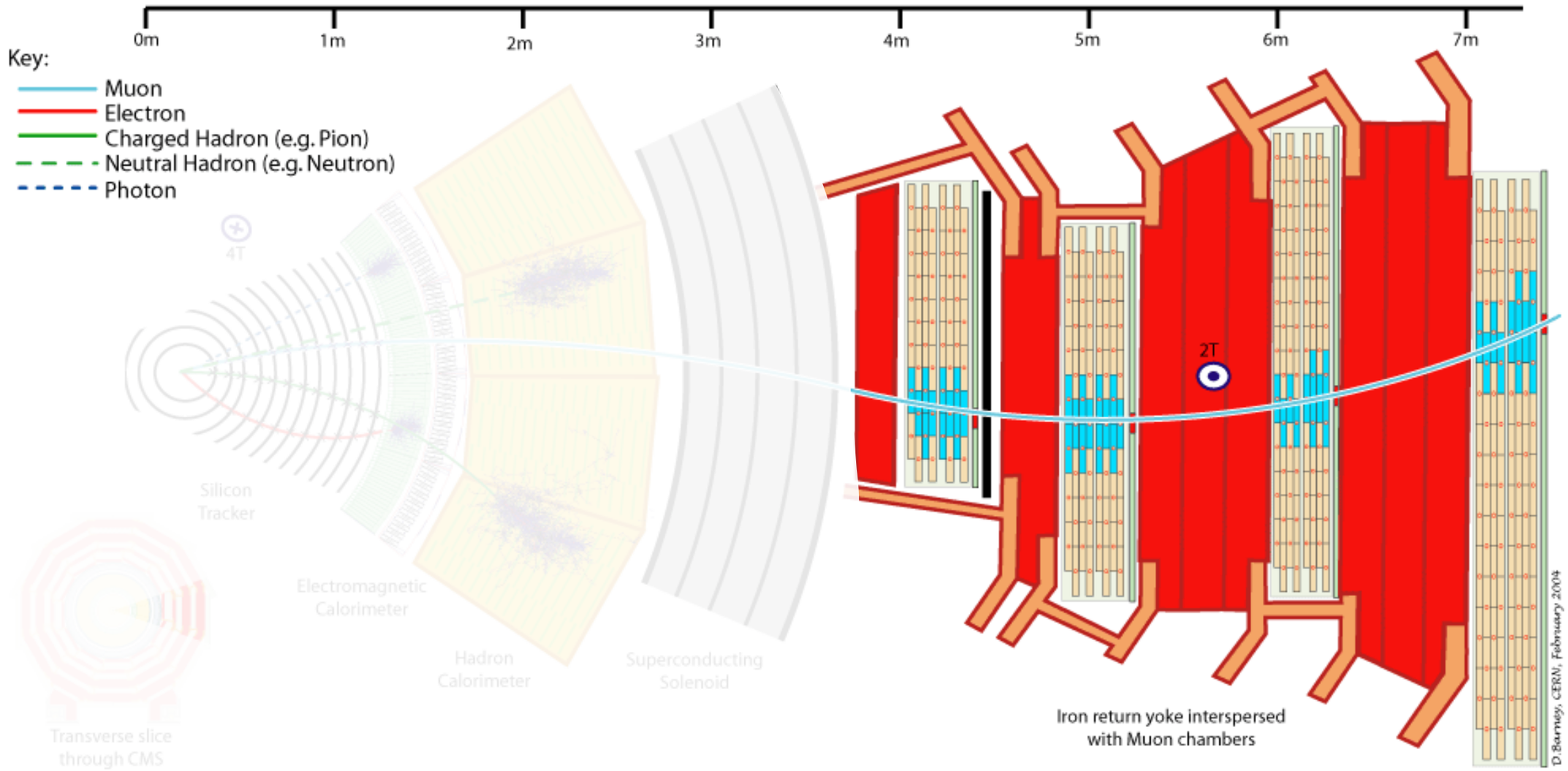
Event reconstruction - Calorimeter



- clustering algorithms to find energy clusters from particle showers
  - mainly driven by calorimeter granularity (not on pile-up)

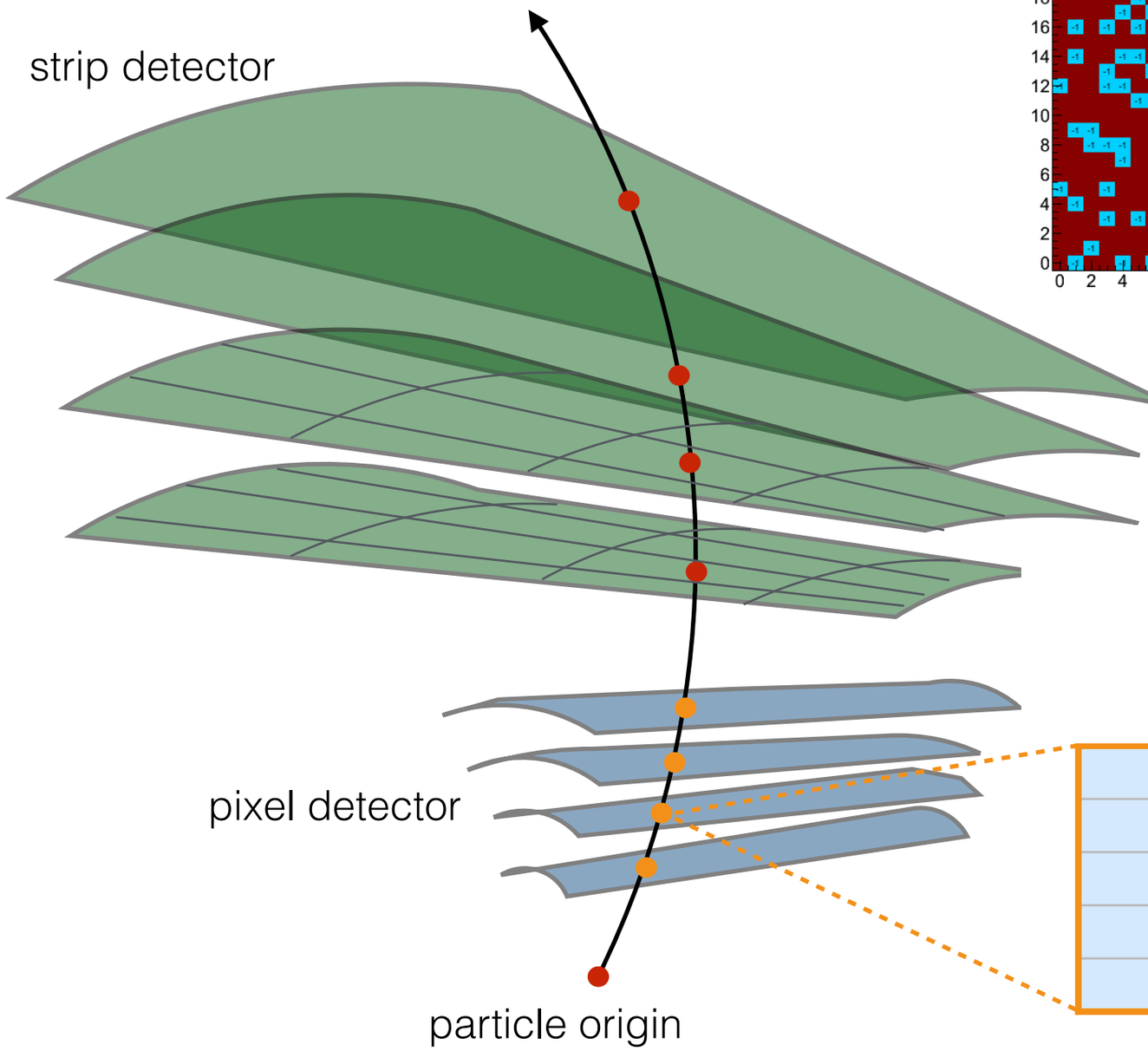
- subsequent jet reconstruction is pile-up dependent (~linear)

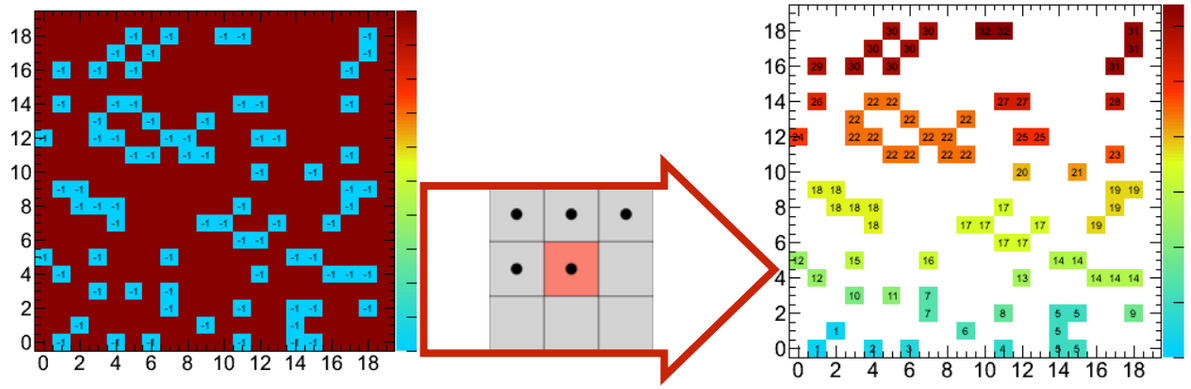
Event reconstruction - Muon System



- trajectory finding (as in Inner Tracker) problem
  - number of muons is orders of magnitude lower than total number of charged particles
  - usually no issue for CPU time

The nutshell view - data preparation in silicon (1)



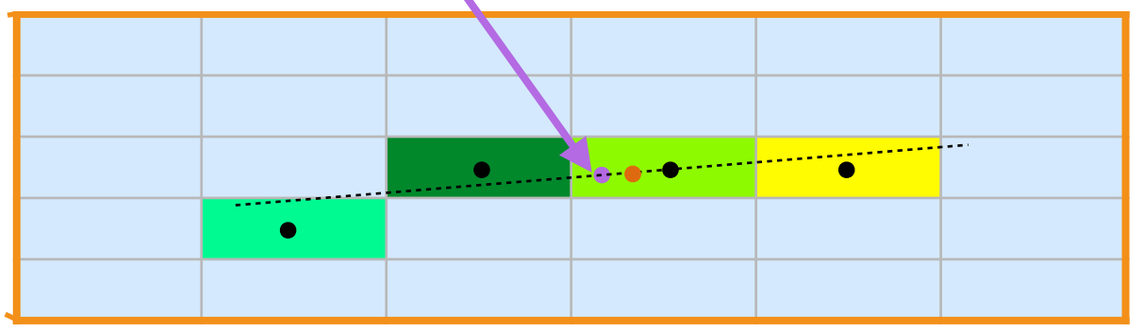


followed by estimation of cluster postion

e.g. the charge-weighted approach :

$$\mathbf{m} = \frac{1}{\sum_{i=1,N} q_i} \sum_{i=1,N} q_i l_i$$

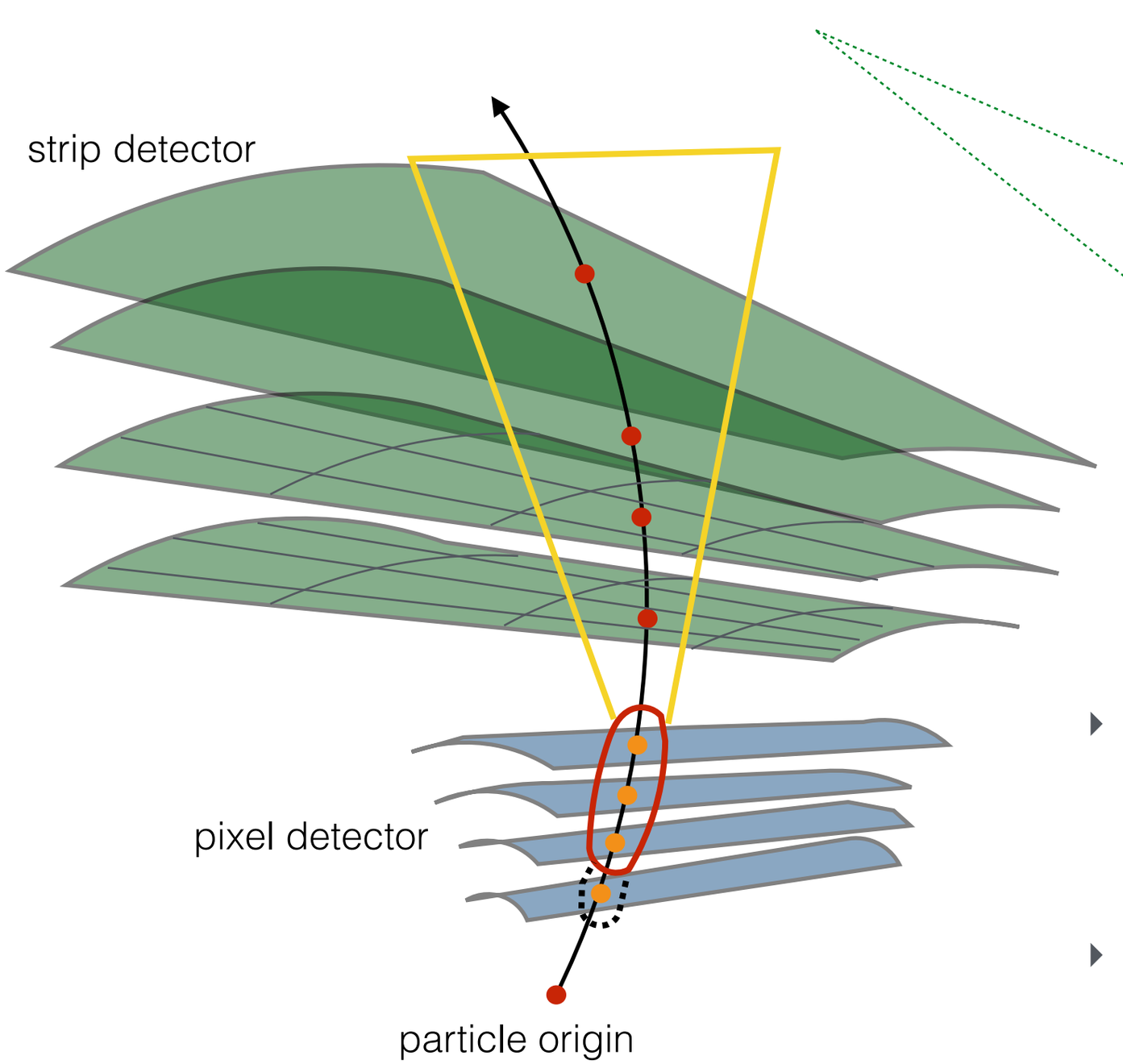
charge collected in cell i

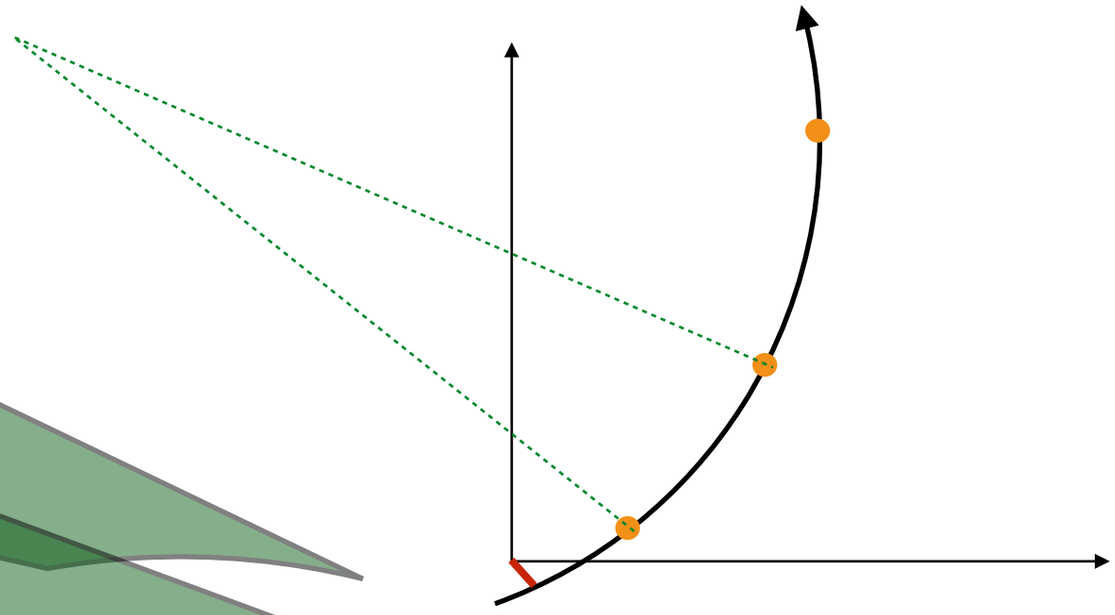


Tracking nutshell view - track finding (1)

Space Point seeding

- building triplets of space points





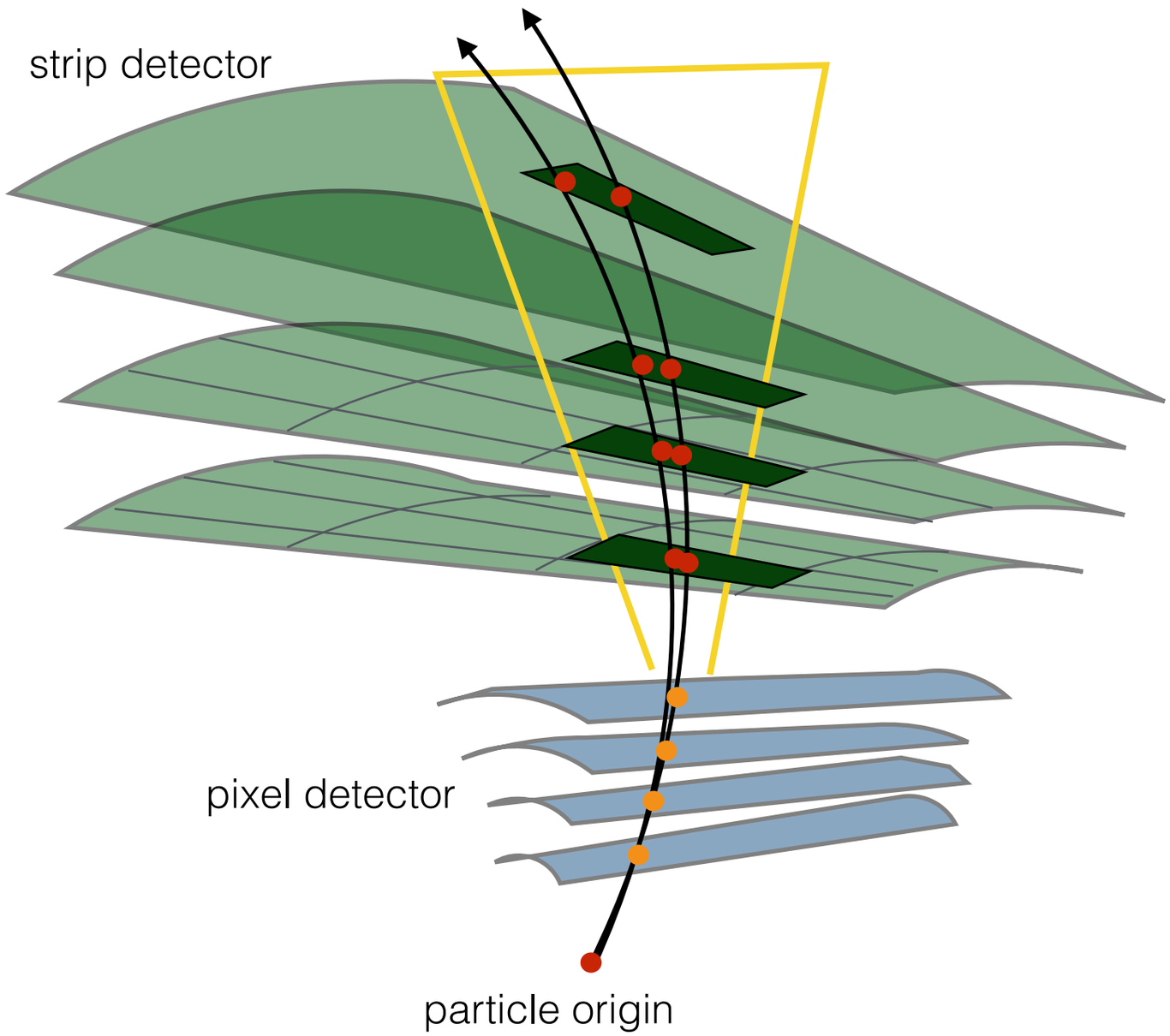
- initial cuts on d_0, p_T

- confirmation with an additional space point

- road building for combinatorial Kalman filter

Tracking nutshell view - track finding (2)

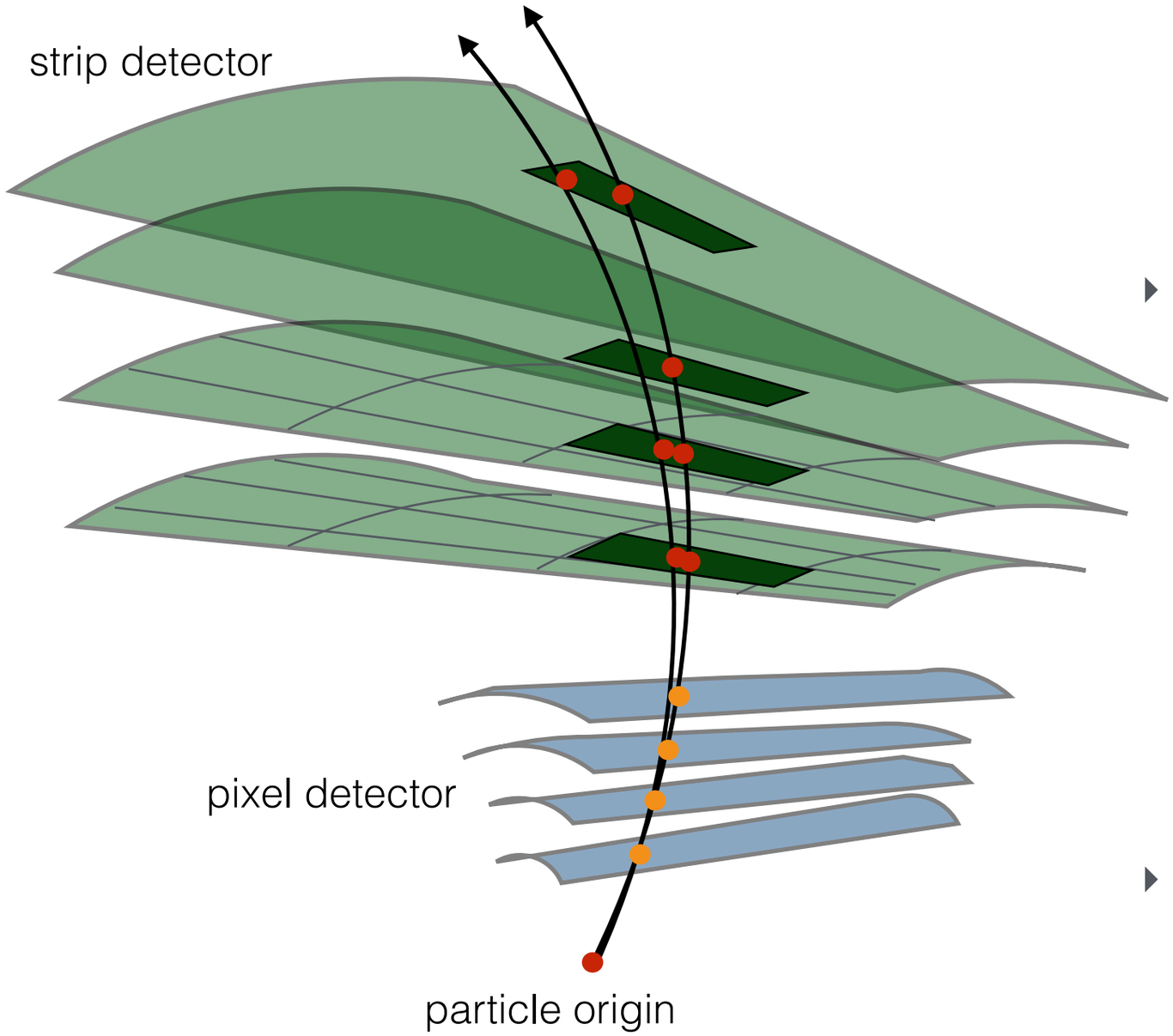
Seeded track finding



- resolve detector elements in a given road and start track candidate search
  - there can be more than one candidate per seed
  - there can be shared hit
  - there can be disregarded seeds
- dedicated road search for electrons allowing kinks due to energy loss
  - only allowed in seeded regions from the calorimeter

Tracking nutshell view - track classification

Evaluate the track candidates

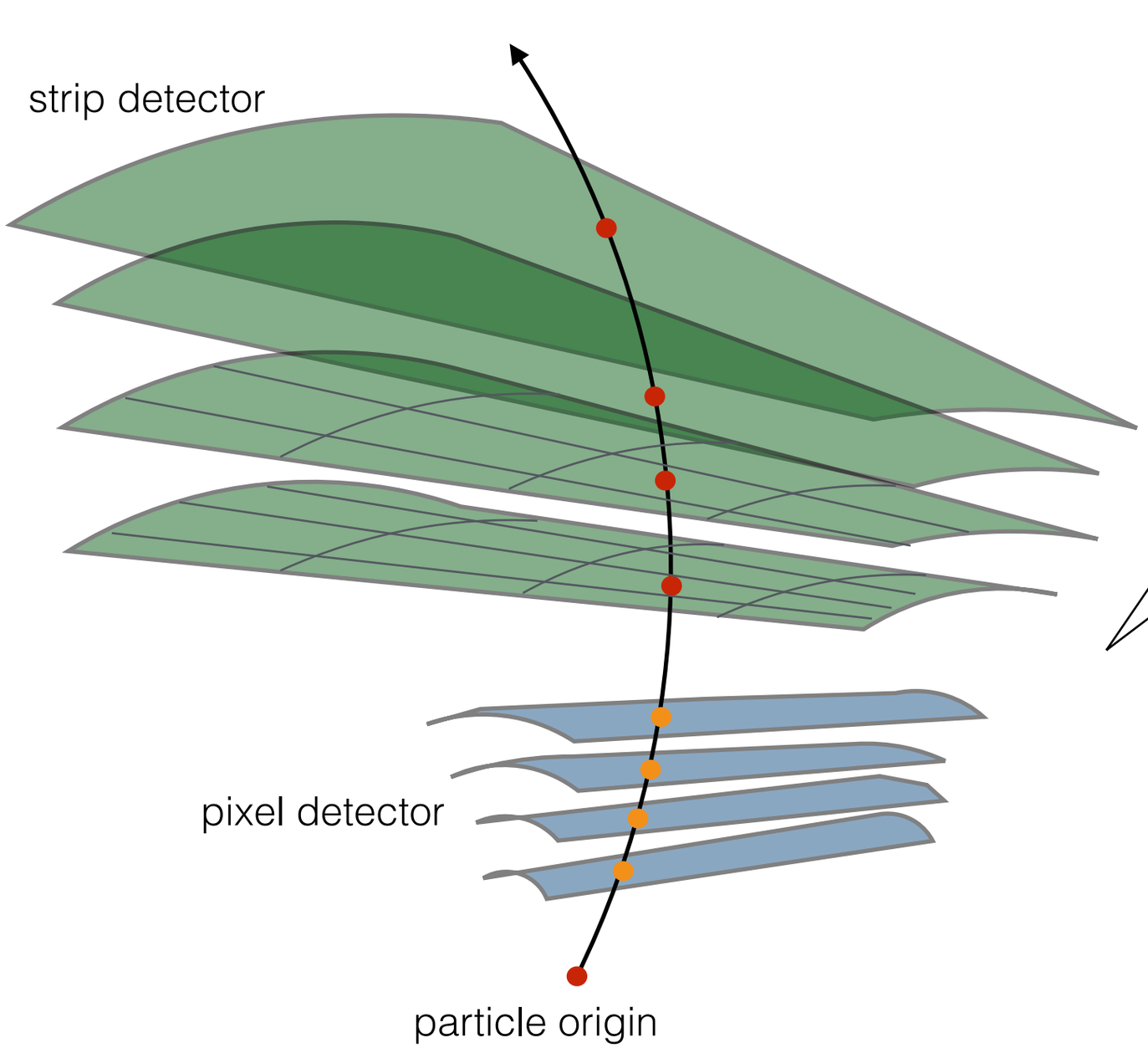


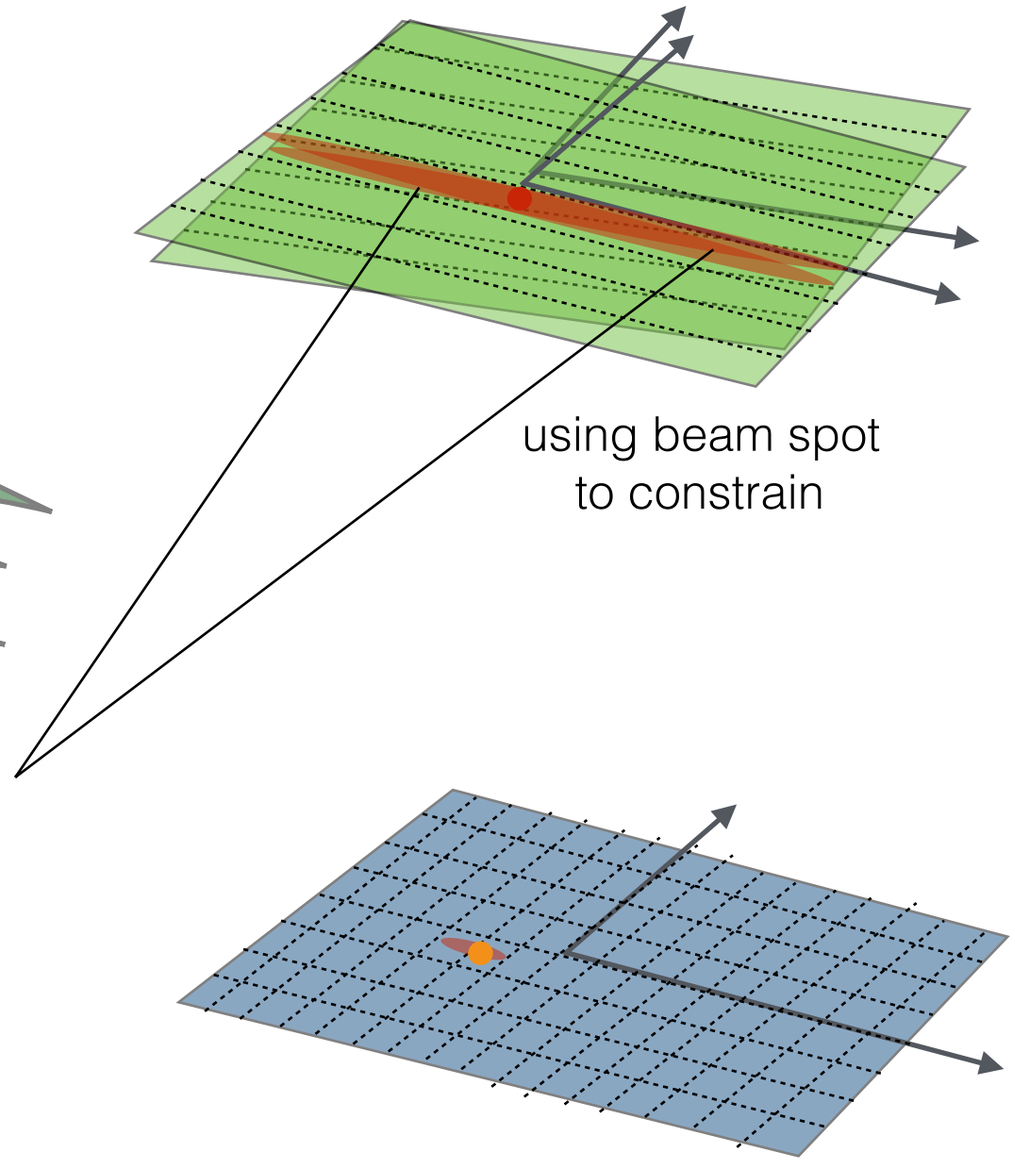
- hit associations
  - using associative maps
  - finding shared hits
  - evaluated shared/split probability
- tracks are ranked (scored)
  - using a scoring tool:
    penalties for holes/shared hits
    bonus points for hits/good chi2
  - classification problem
    (NN based version exists)
- let tracks with highest score survive

The nutshell view - data preparation in silicon (2)

Space Point Formation

using the local cluster positions & sensor surface to form 3D space points

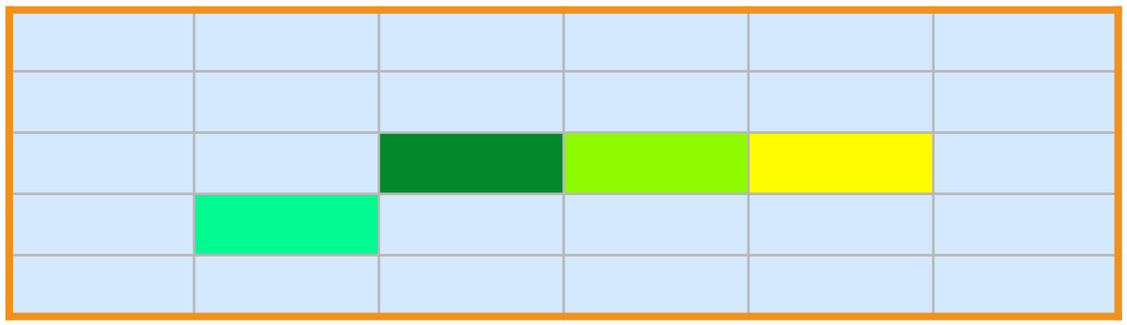




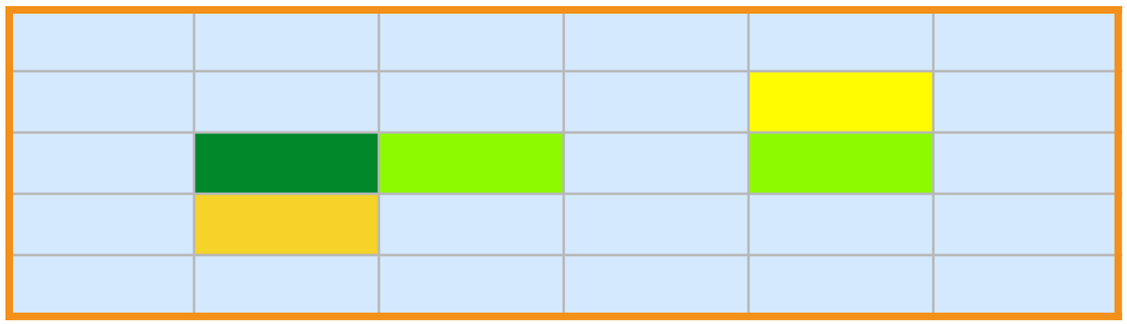
Concurrency usage - Cluster creation

- Cluster creation is the first stage of local pattern recognition

module i



module i+1

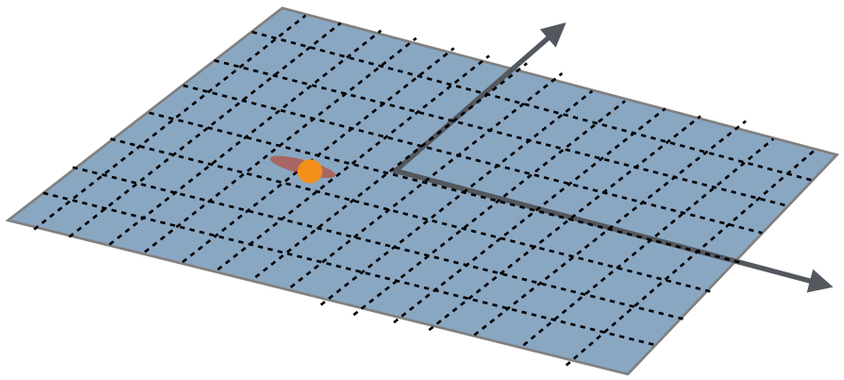


- Connected component analysis (CCA) runs per module
- Could easily be parallelised
  - a perfect use-case for a HiveAlgorithm that opens up many (100s) threads
- No bookkeeping needed, since every module represents one collection in our Identifiable container module
- It will not gain us all too much CPU gain, but it is a very good testbed

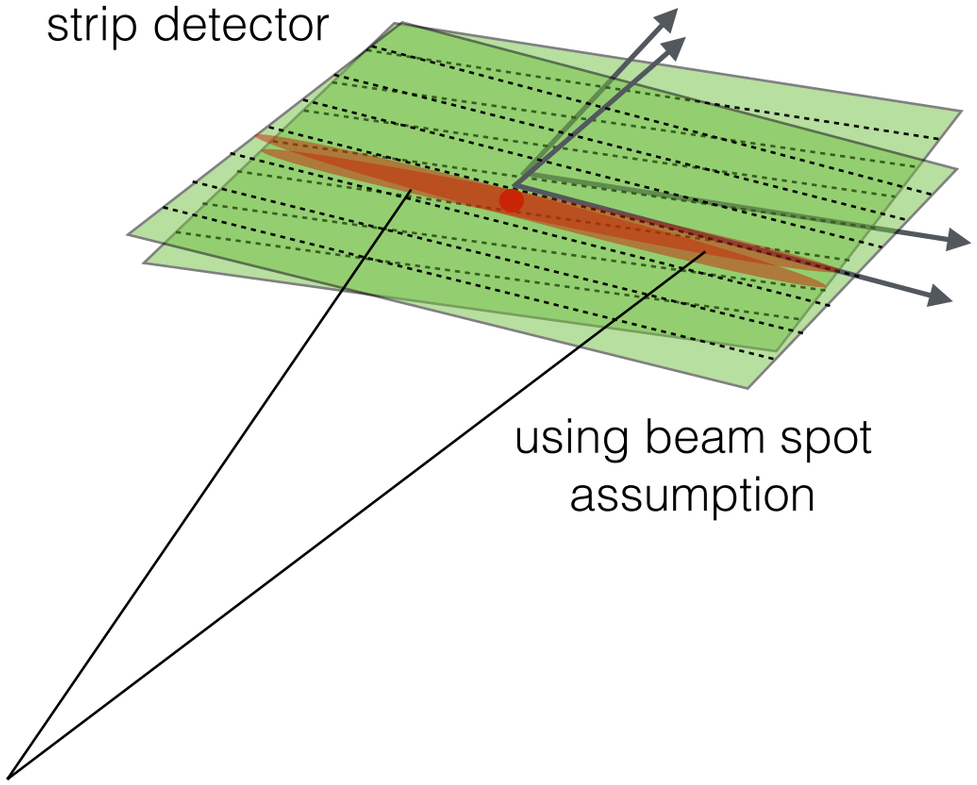
Concurrency usage - Space point creation

- Space point creation is a very simple activity

pixel detector



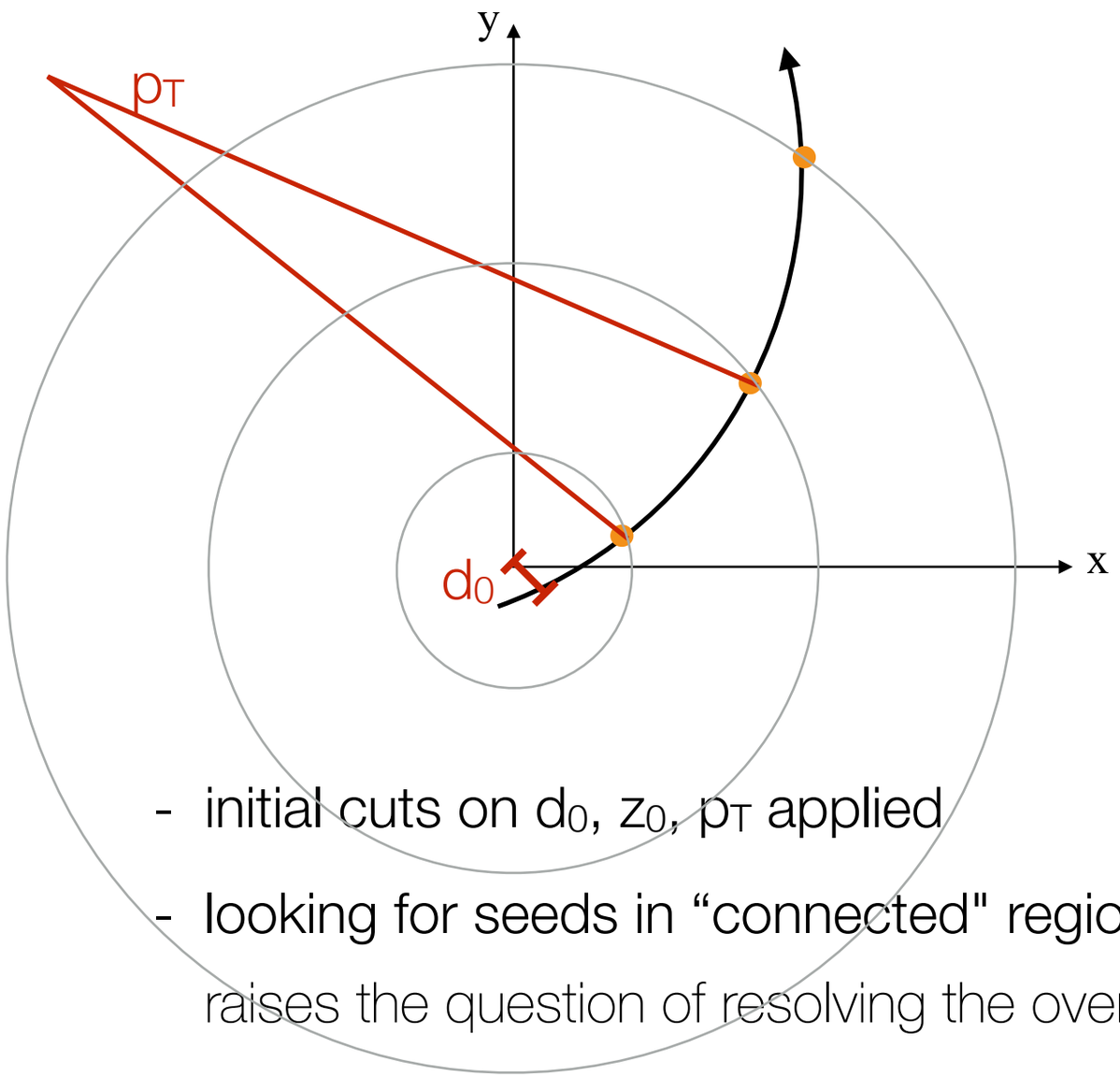
strip detector

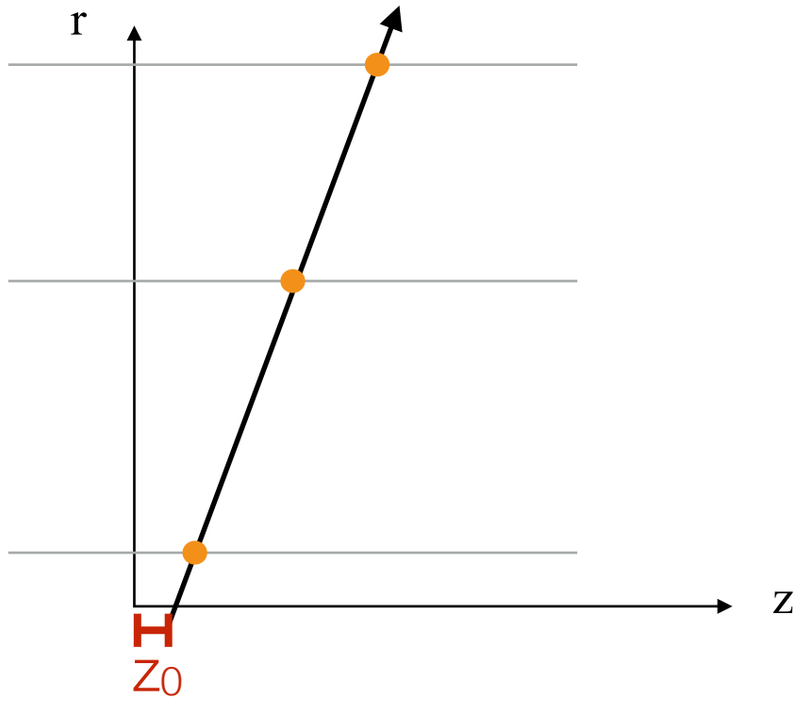


- Simple local to global transformations
  - in principle a vec(3) x mat(3,3) + vec(3) operation, can be done as vec(4) X mat(4,4)
  - not much relative CPU time to gain, however a perfect testbed for e.g. aggressive compiler settings & Eigen
  - number of cells is huge (e.t. 10k space points) and memory requirement is minimal

Concurrency usage - seed creation

- Step 1: finding triplets of space points and evaluate their compatibility

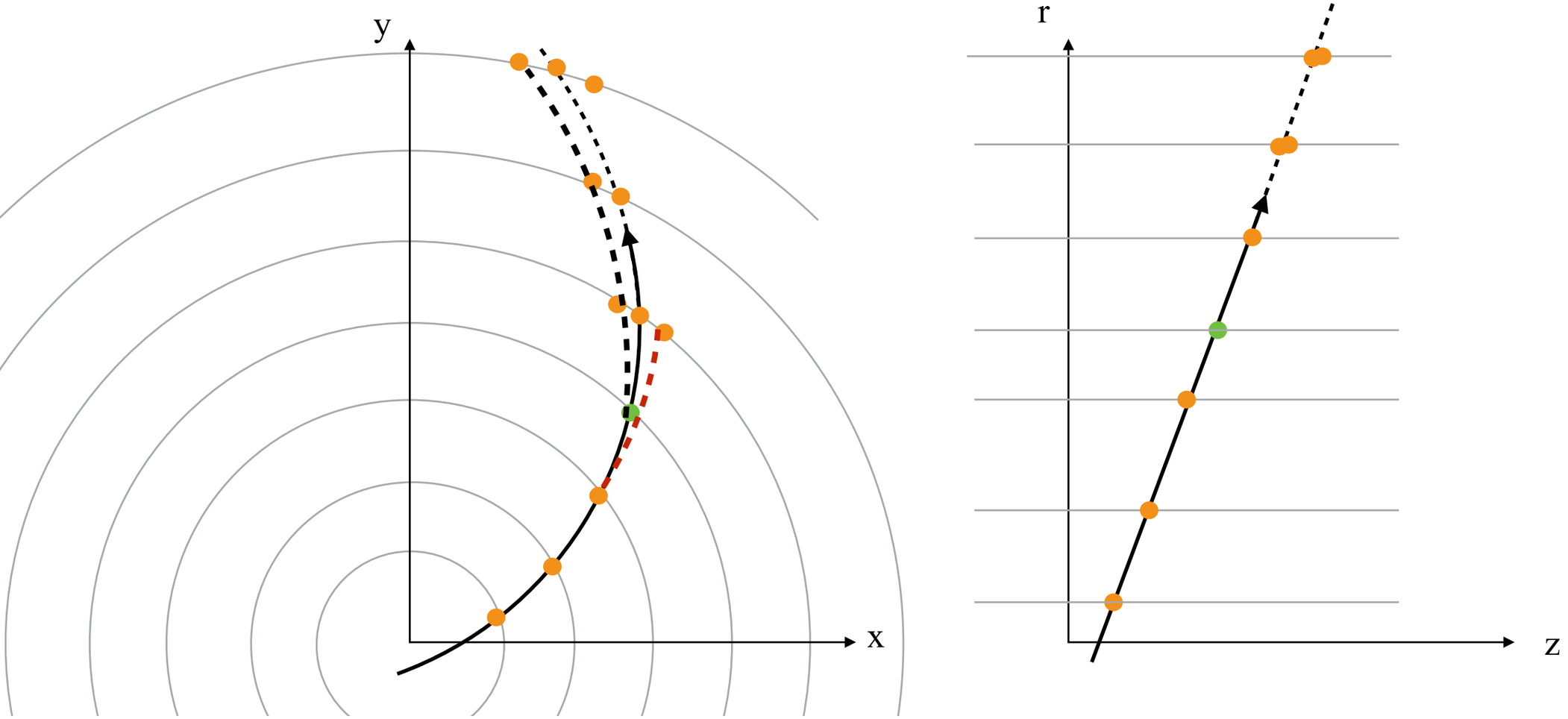




- initial cuts on d_0, z_0, p_T applied
- looking for seeds in “connected" regions, not just random space point combinations
  raises the question of resolving the overlap regions
- currently some book-keeping is done whether space points are already used
  makes **concurrency** a bit tricky
  having space points multiply used is not a problem per se (-> can be resolved later)

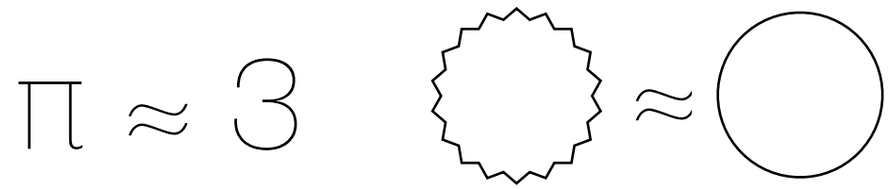
Concurrency usage - track finding (1)

- Step 3: seed defines a road, track finding within road



- Seeds can lead to several track candidates
  - parallel following of track candidates possible
  - open and close threads

Simplification



- Common pattern in track reconstruction code
  - simplification of geometry
  - simplification of magnetic field

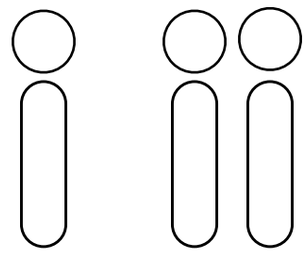
Lookup patterns

1 €	2 DM
2 €	4 DM

- precomputed results

Cheating

- truth reconstruction for MC samples

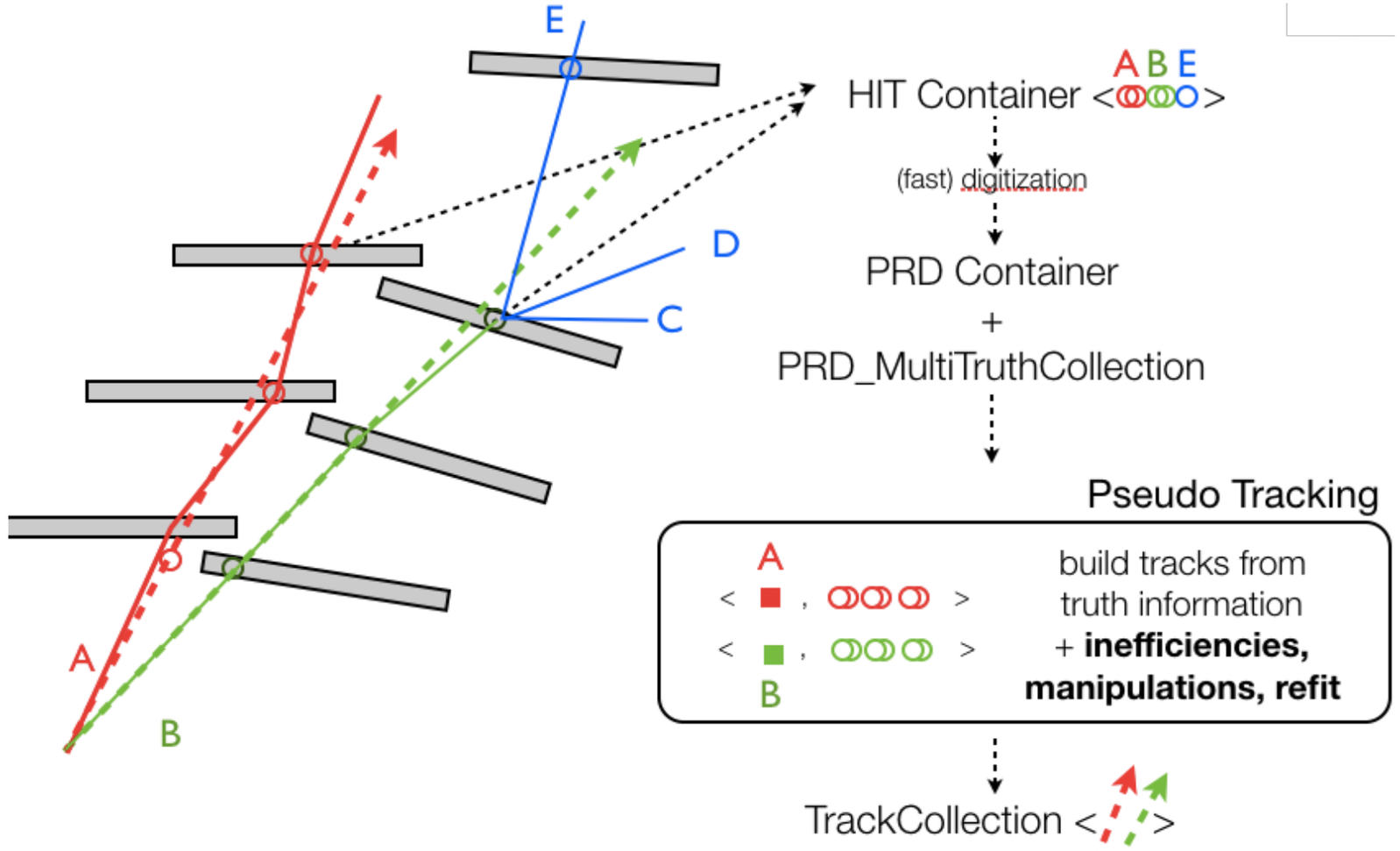


Software/Algorithms - cheat

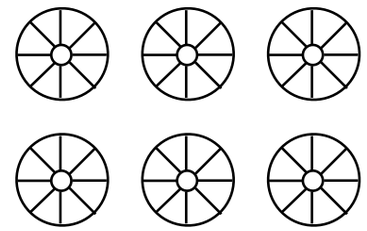
- track reconstruction not only needs to support data taking
  - typical MC campaign of ATLAS/CMS is of O(10^{9}-10^{10}) events
  - unless we change quite substantially how we do high energy physics we should always keep in mind how to simulate (and that we reconstruct simulated) events

- truth (aided) tracking exploited by several experiments
  - works, because the pattern recognition is so efficient
  - does not produce fake tracks at all

    event more stress on a low fake detector



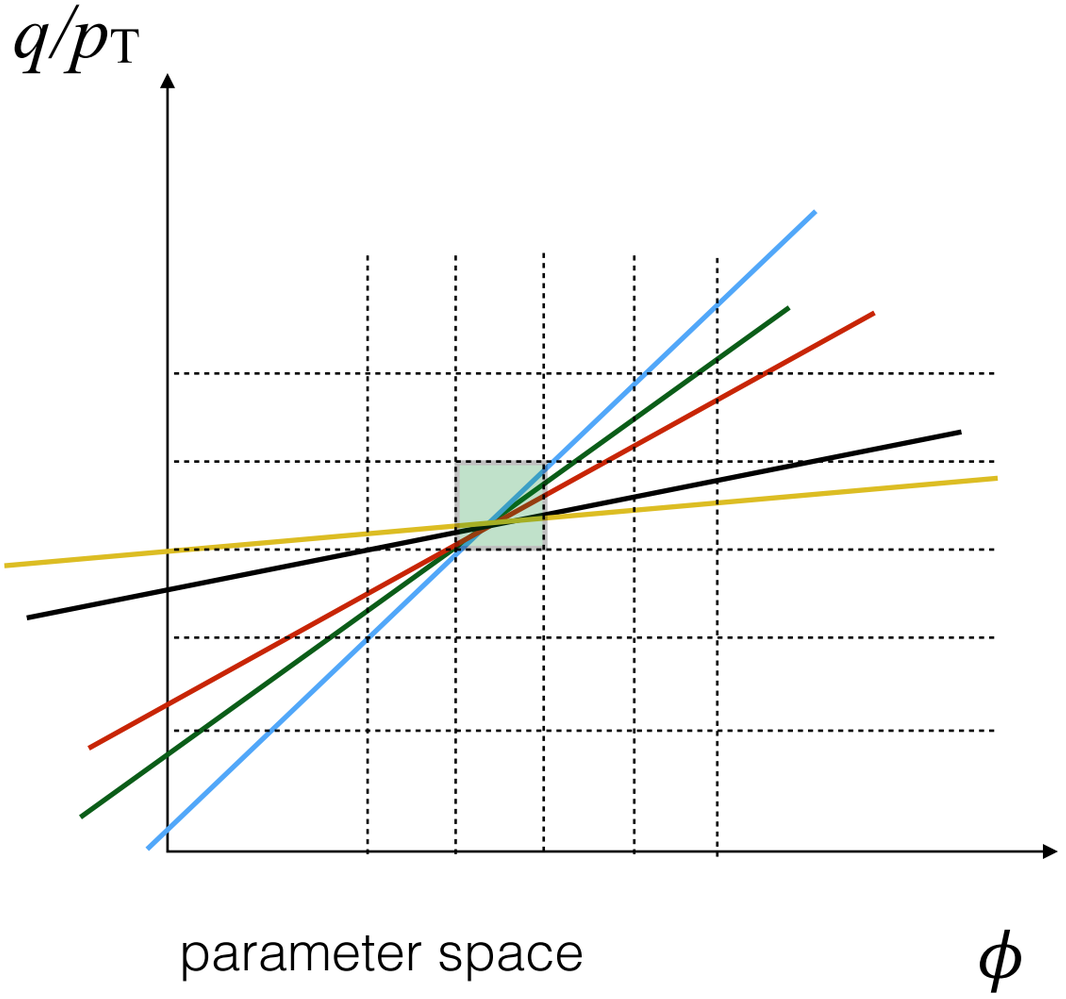
Hough transform

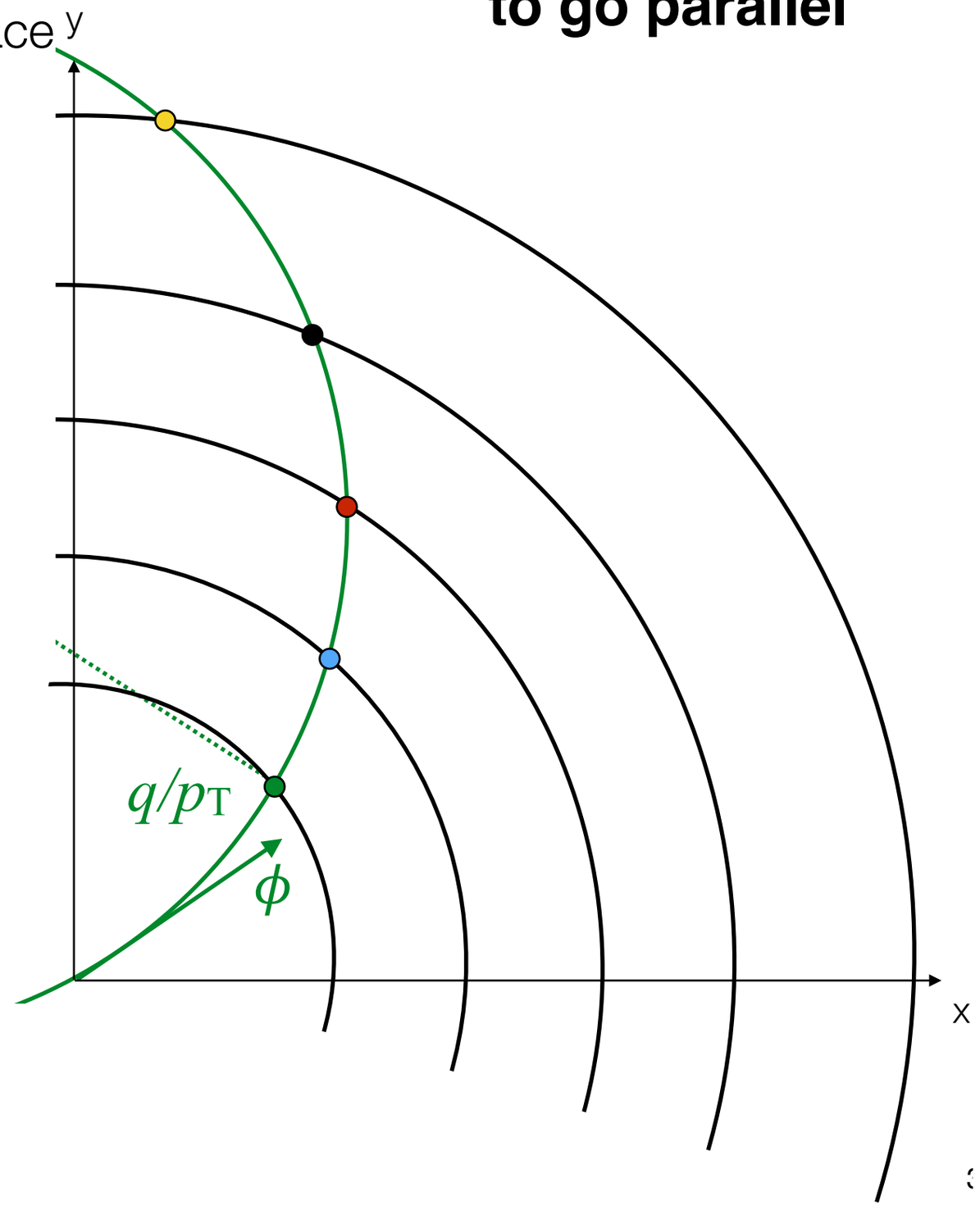


quite a potential to go parallel

- Conformal mapping/Hough transform
  - transform your track hits from the x, y space

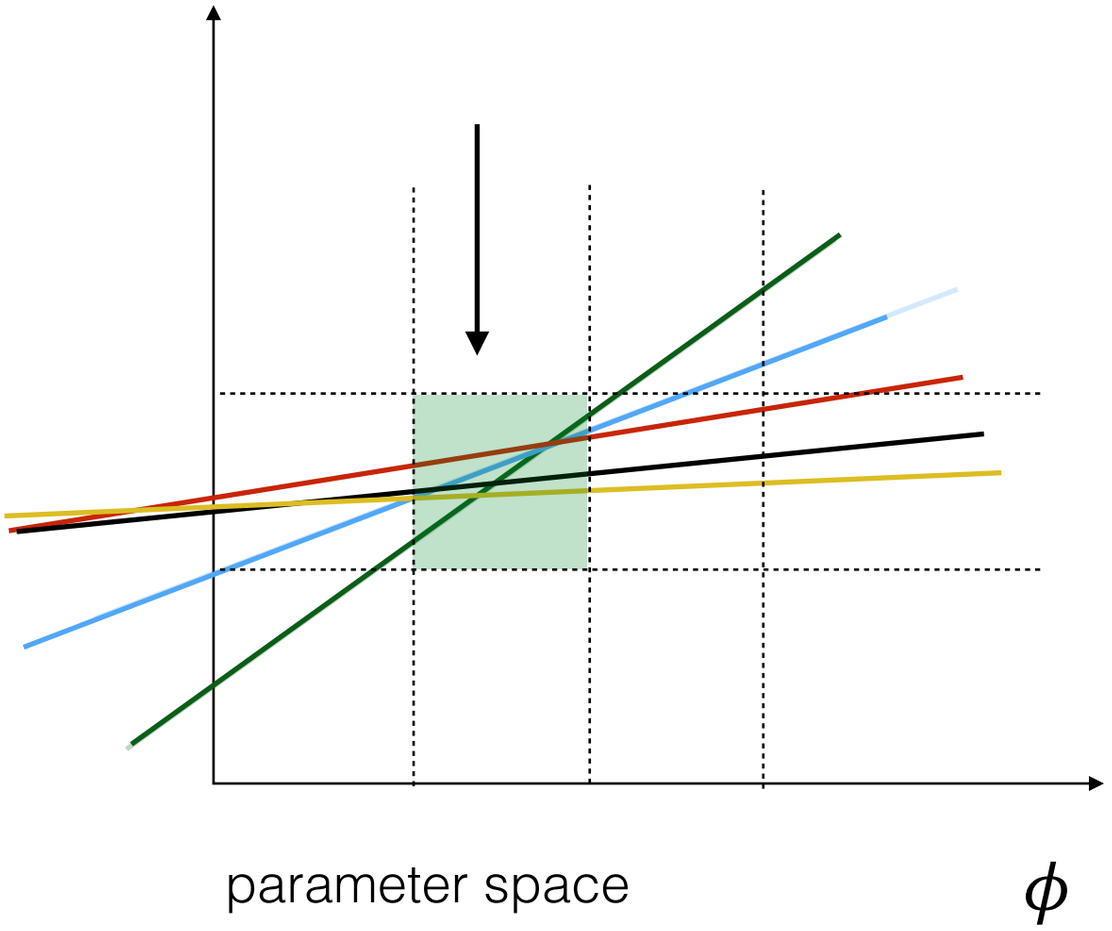
$$\mathbf{q} = (\cancel{d_0}, \cancel{z_0}, \phi, \cancel{\theta}, q/p_T)$$

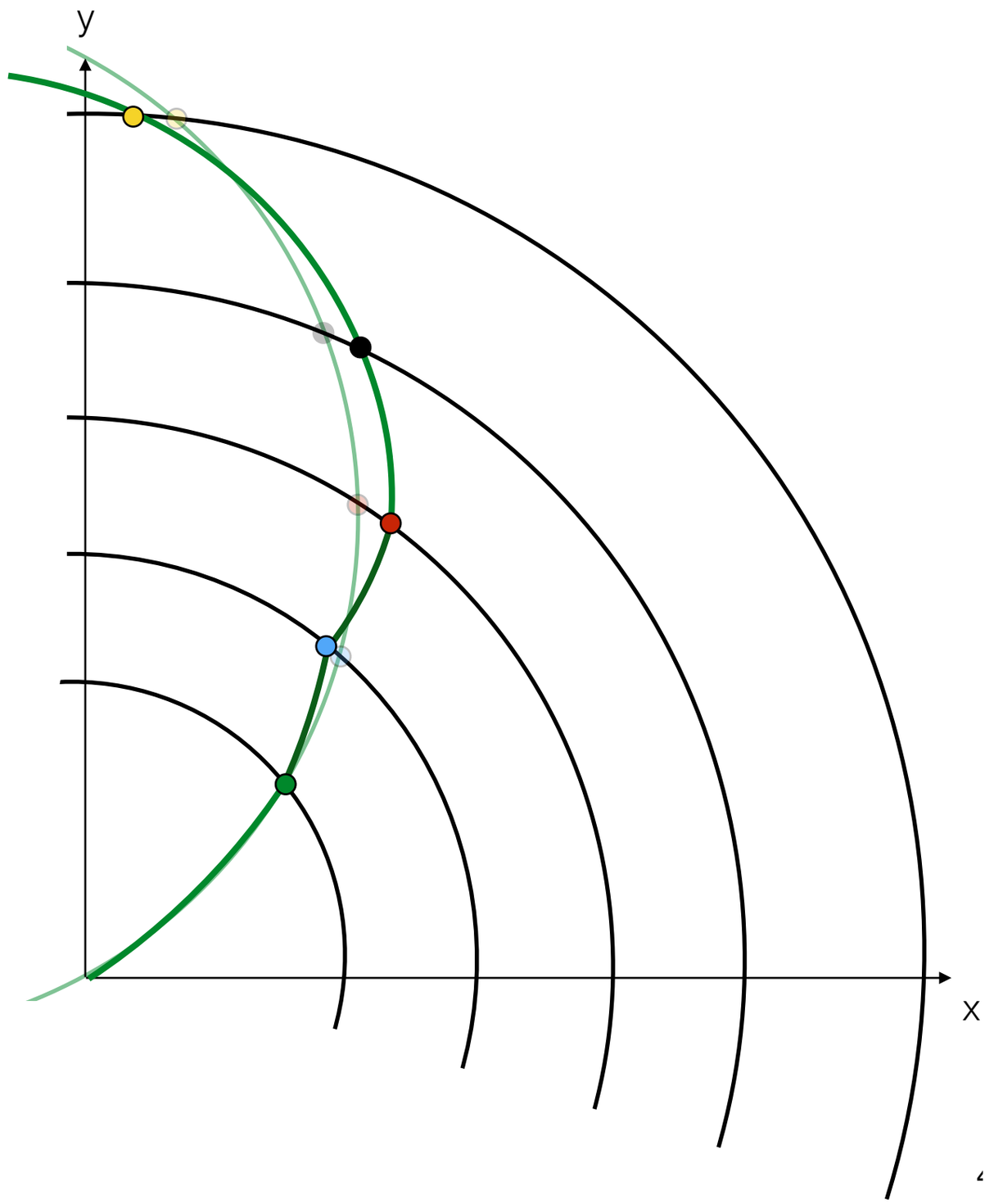




Conformal mapping - the problems

- Conformal mapping works nicely in an ideal world
  - reality is a bitch though:
    material interaction, inhomogenous
    magnetic field





Pattern banks

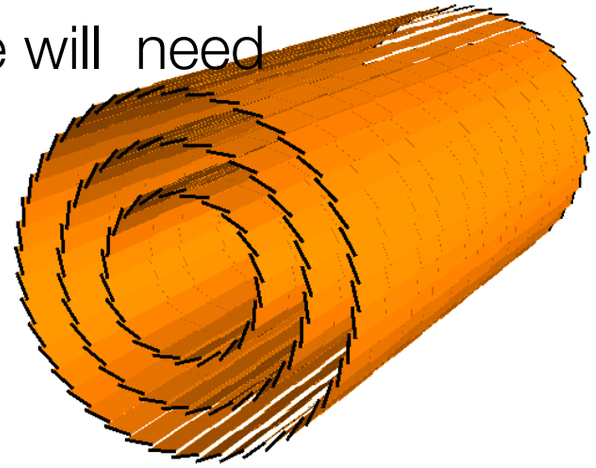
1 €	2 DM
2 €	4 DM

- Pattern banks are interesting for Track triggering
  - ATLAS L1Track/FTK++
  - CMS L1 Track trigger
- Pattern banks & look-up tables have limitations for offline reconstruction
  - usually come with a granularity (
  - work well for certain features, don't work well for others (potential bias)
  - work well in reality, but can be pretty hard to simulate
- L1 track triggers are certainly very useful for triggering, event classification
  - also may be used as seeds for a dedicated reconstruction in special regions
  - I personally do not see pattern banks as a possible replacement for offline reconstruction

A HEP Tracking Machine Learning Challenge (3)

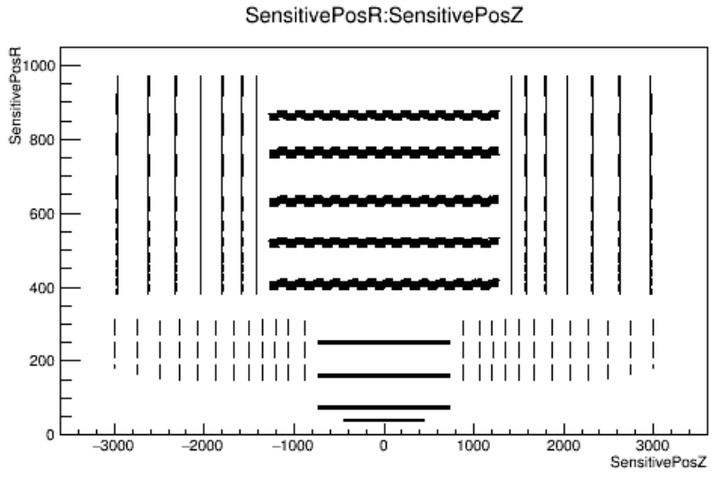
- Challenge will have to take several stages

- we will need



detector geometry

planar barrel/EC type detector
pixel/strip system



simulation

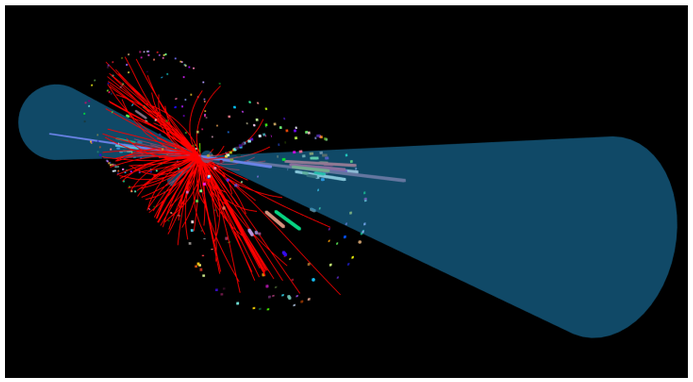
with the possibility to
simplify where possible

```
{
    "hits": [
        23.04,
        -123.2,
        83.22
    ]
}
```

Valid JSON

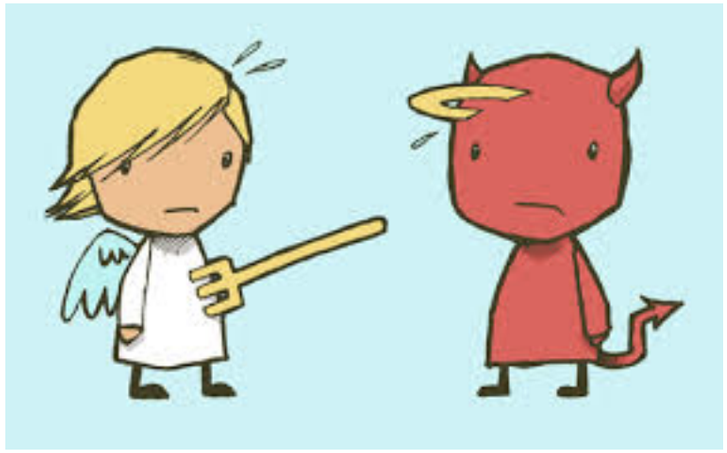
event data

easily readable,
platform independent



visualisation

of geometry,
hits & found tracks



well defined goal

what is success
and how we measure it

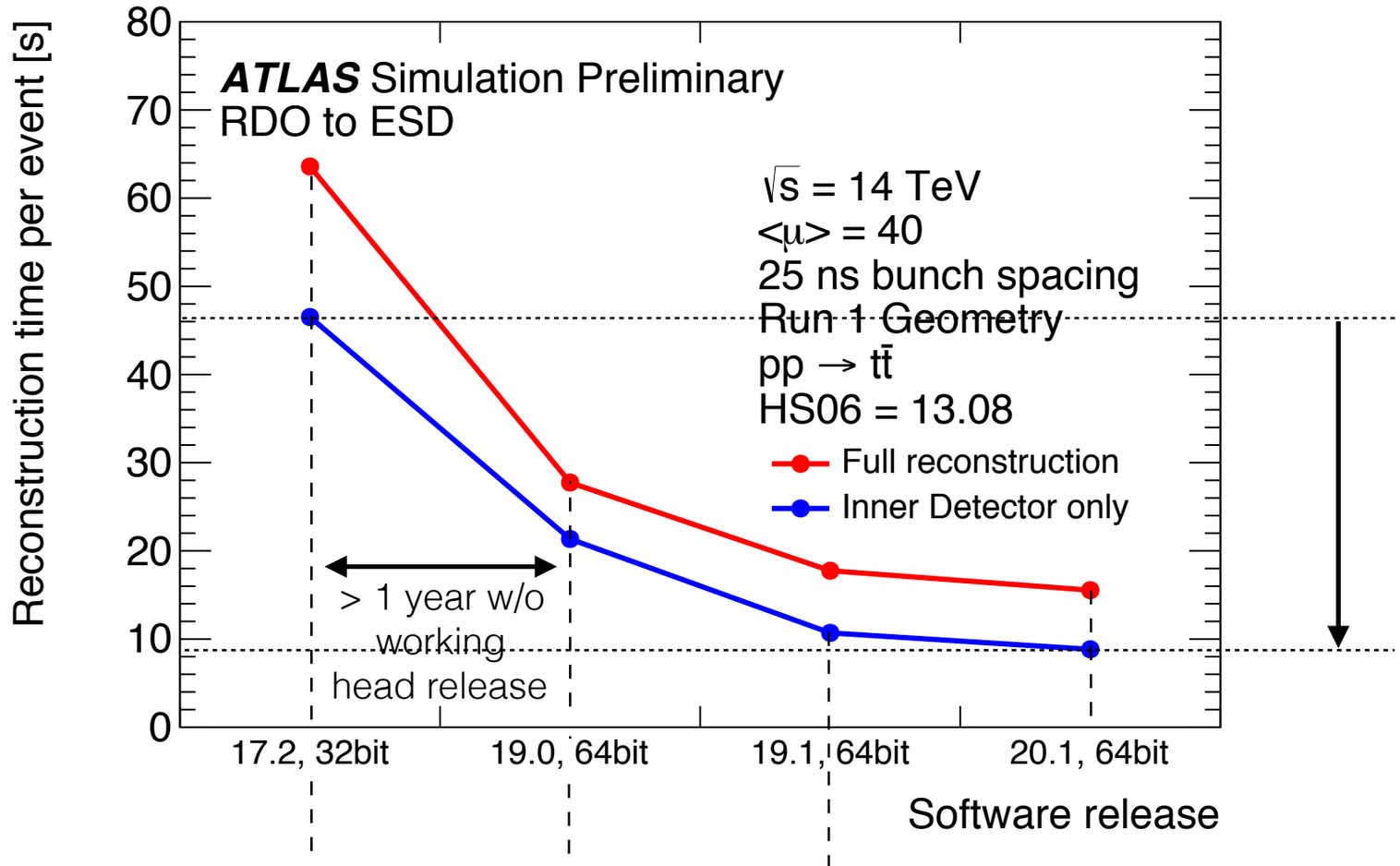
different categories

for different
solutions

Review, clean up (3)

std::move(result)

- SW cleanup campaign was a huge manpower effort



> 5 CPU gain,
no loss of physics performance

A. Salzburger, CHEP2015