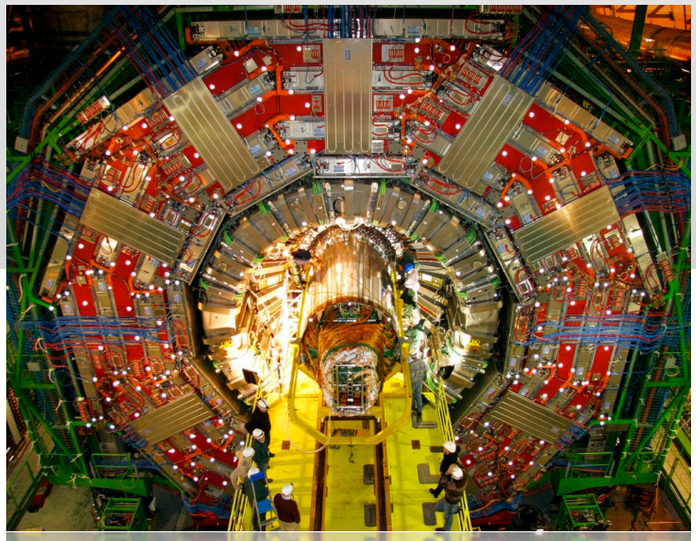
CMS reconstruction overview and plans



28.11.2012

Fourth international workshop for Future Challenges in Tracking and Trigger concepts

GSguazzoni

Outline

= A glimpse on the CMS tracking implementation

= The tracking evolution from 2011 to 2012

= The challenge of 2015 data taking

= Multi threading, vectorization, parallelization

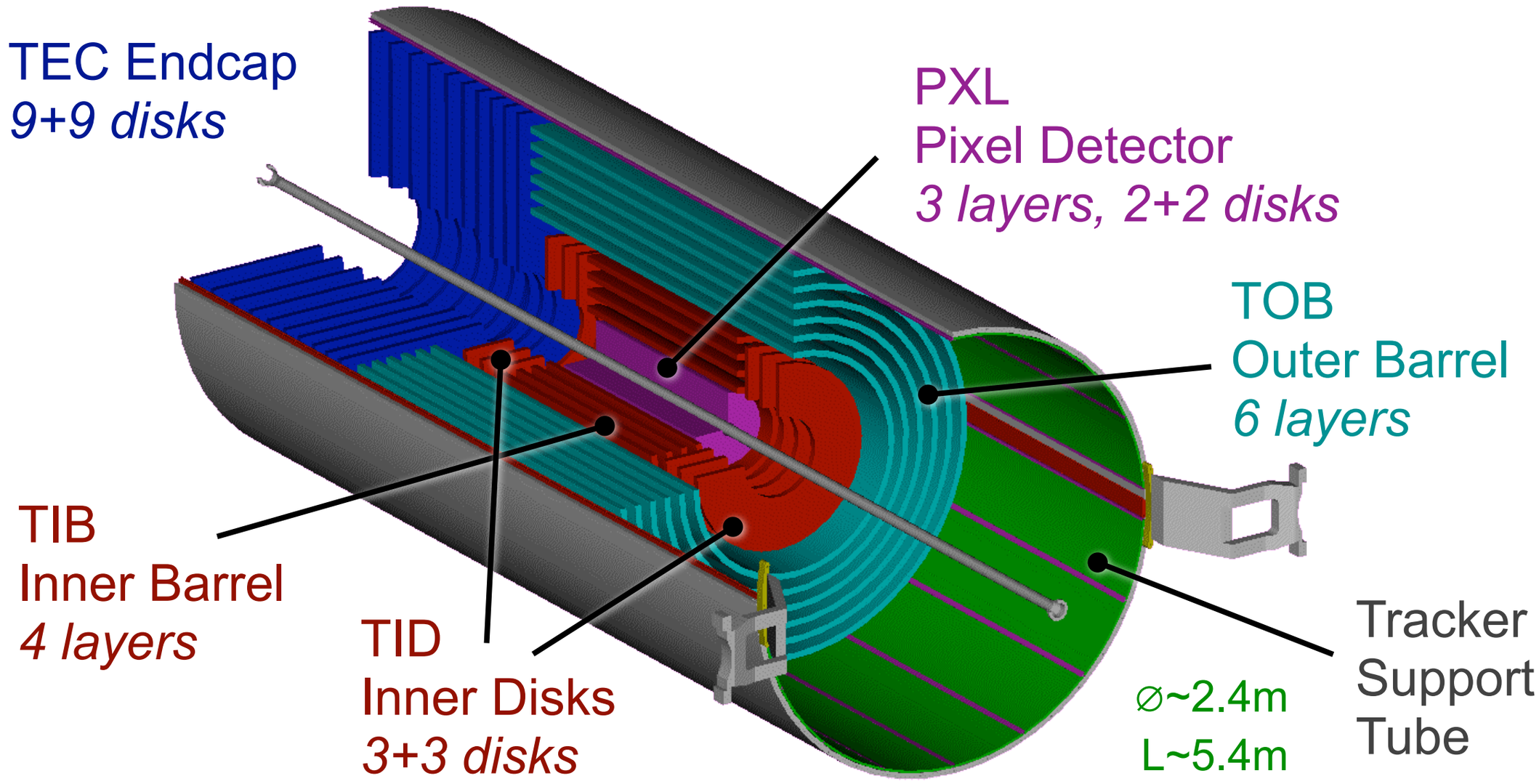
= Raw ideas for new tracking algorithms

For help and material, many thanks to: KStenson, GCerati, THauth, CJones, GEulisse, ...

The world largest Silicon Tracker

Pixel Detector
66M channels
100x150 μm² pixel
LHC radiation resistant

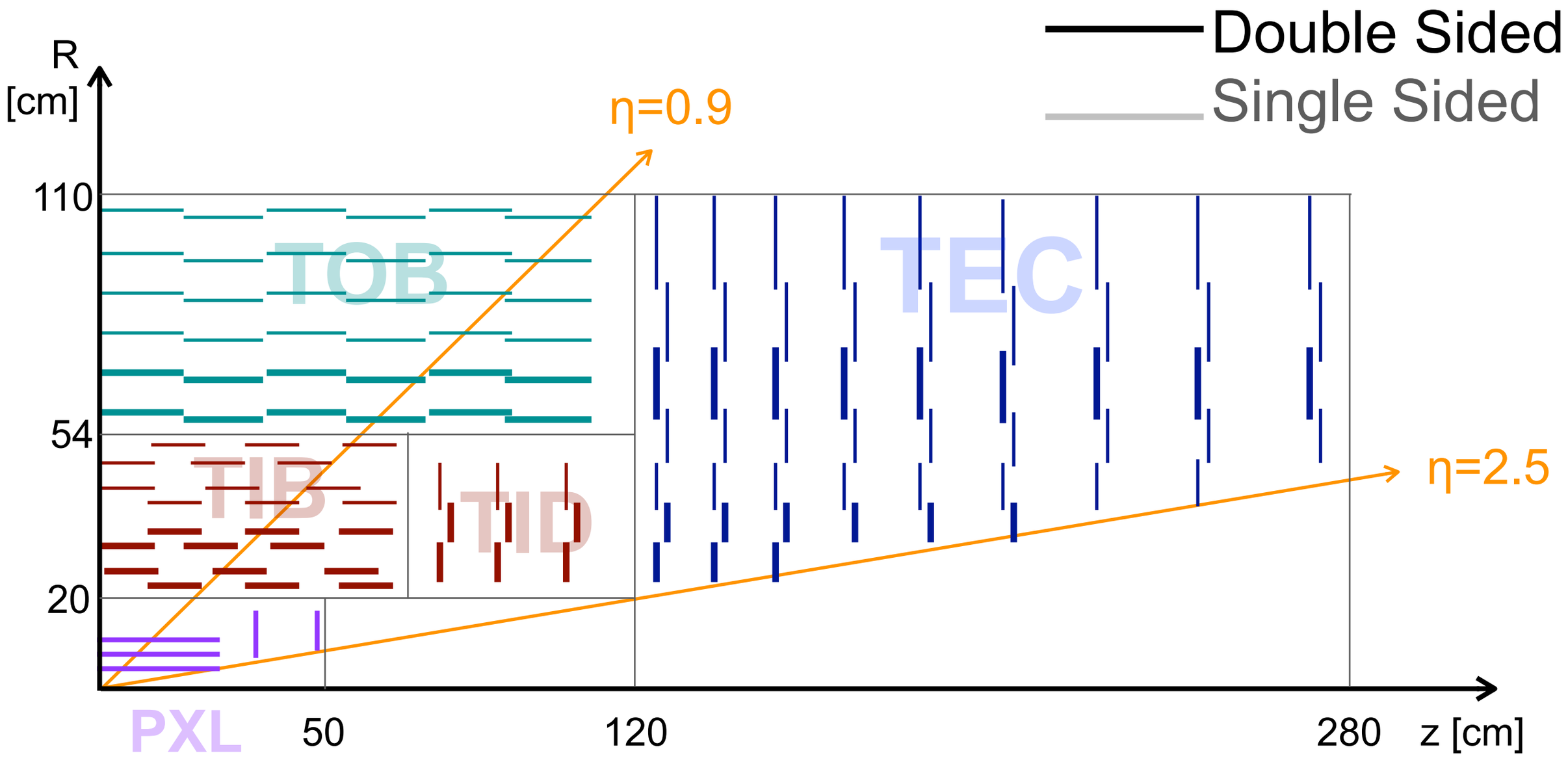
Si-Strip detector
~23m³; ~200m² of Si area;
~9x10⁶ channels;
LHC radiation resistant



The CMS Silicon Tracker Layout

Basic Performances

$\sigma(P_T)/P_T$ ~1-2% (P_T~100 GeV/c)
IP resolution ~10-20μm (P_T=100-10 GeV/c)



Tracker Material

The CMS Tracker is a ~4T object made up of assorted and diverse materials!

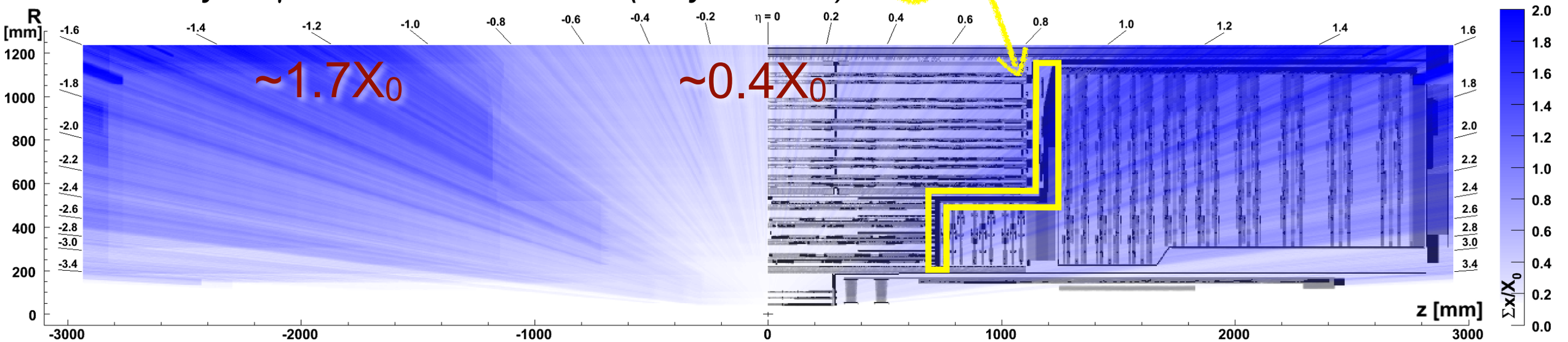
Accurate simulation is crucial.

Large contribution comes from *service volumes* for 1.0<η<1.6; some tracks intercept these volumes three times.

	Name	Mass [kg]	Fraction
1	CarbonFiber	1144.503	27.631%
2	Copper	595.674	14.381%
3	Aluminium	594.960	14.364%
4	PE	354.633	8.562%
5	C6F14	258.890	6.250%
6	Silicon	225.847	5.453%
7	Nomex	123.331	2.978%
8	G10	110.180	2.660%
9	FR4	103.238	2.492%
10	Ceramic	91.062	2.198%

$\sum x^* X_0^{-1}[cm^{-1}]$ ~density of γ conversion tracks

X_0^{-1} ~density of γ conversion vertices (only z+ side)

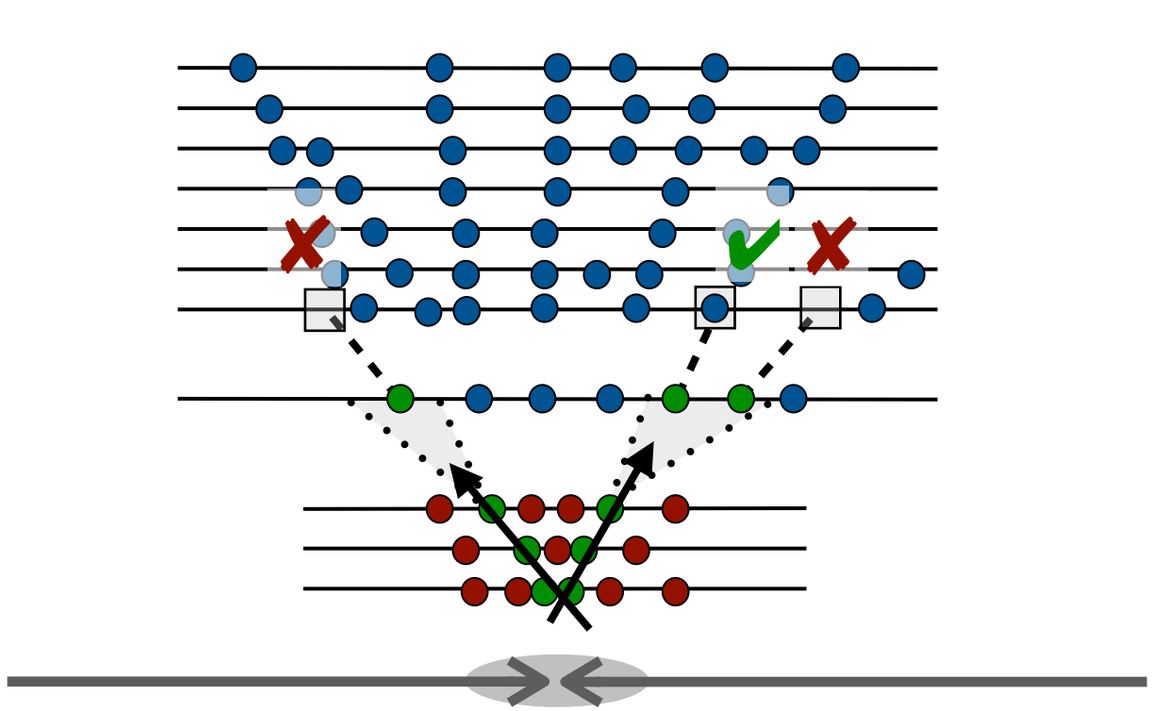


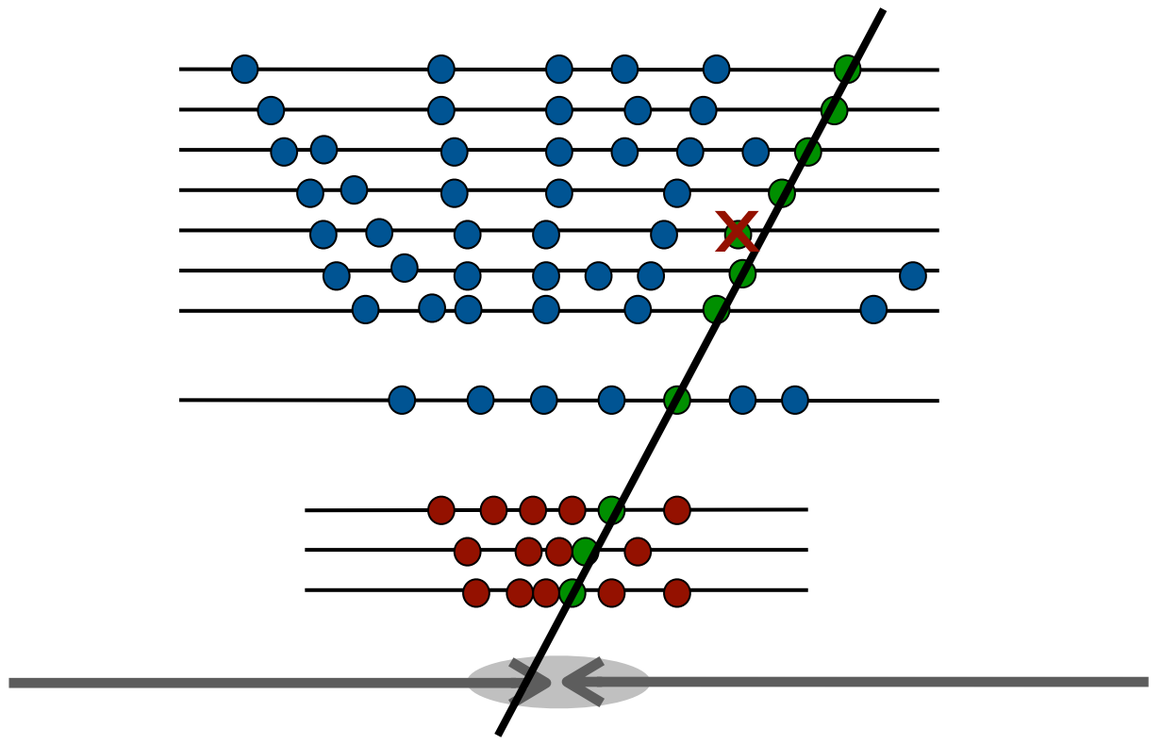
Overview of track reconstruction in CMS

CMS tracking in a nutshell

Seeding starts from innermost pixel layers (pairs + PV, triplets).
Inside-out trajectory building through pattern recognition (based on Kalman Filter).

Rejection of outlier hits and final fit, also based on Kalman Filter.
Final quality selection of tracks.
Primary Vertex used in tracking derived from pixel-based algorithm.

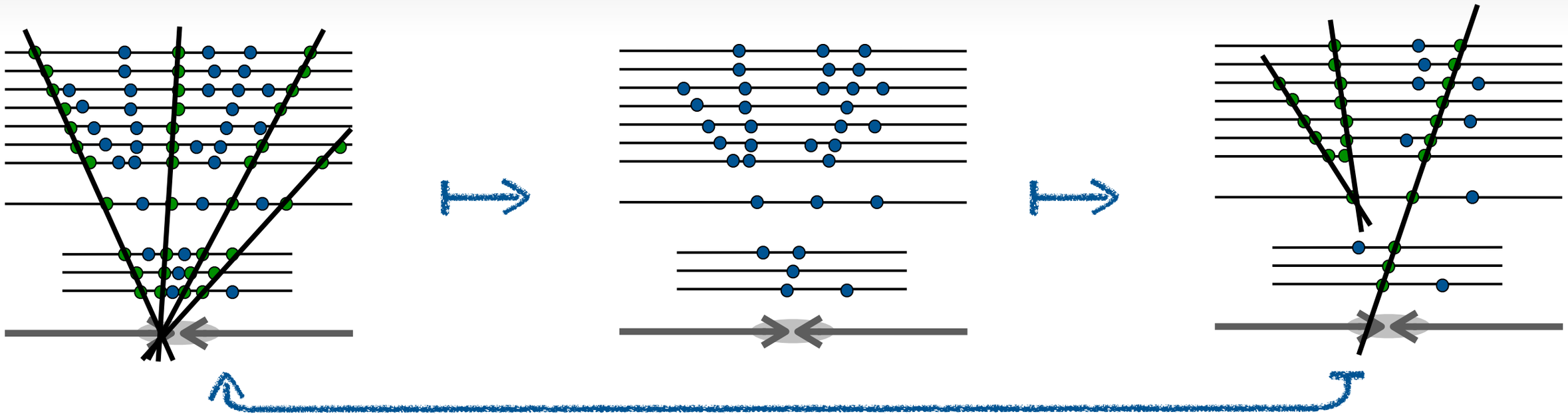




Track Parameters: q/p, η, φ, d_Z, d_{XY}

Parameters propagated through magnetic field inhomogeneities using **Runge-Kutta** propagator

Iterative tracking



The CMS tracking relies on iterations (*steps*) of the tracking procedure; each step works on the remaining not-yet-associated hits and is optimized with respect to the seeding topology and to the final quality cuts.

#step	seed type	seed subdetectors	$P_T^{\min}$ [GeV/c]	d_0 cut	z_0 cut
0	triplet	pixel	0.8	0.2 cm	3.0σ
1	pair	pixel/TEC	0.6	0.05 cm	0.6 cm
2	triplet	pixel	0.075	0.2 cm	3.3σ
3	triplet	pixel/TIB/TID/TEC	0.25-0.35	2.0 cm	10.0 cm
4	pair	TIB/TID/TEC	0.5	2.0 cm	12.0 cm
5	pair	TOB/TEC	0.6	6.0 cm	30.0 cm

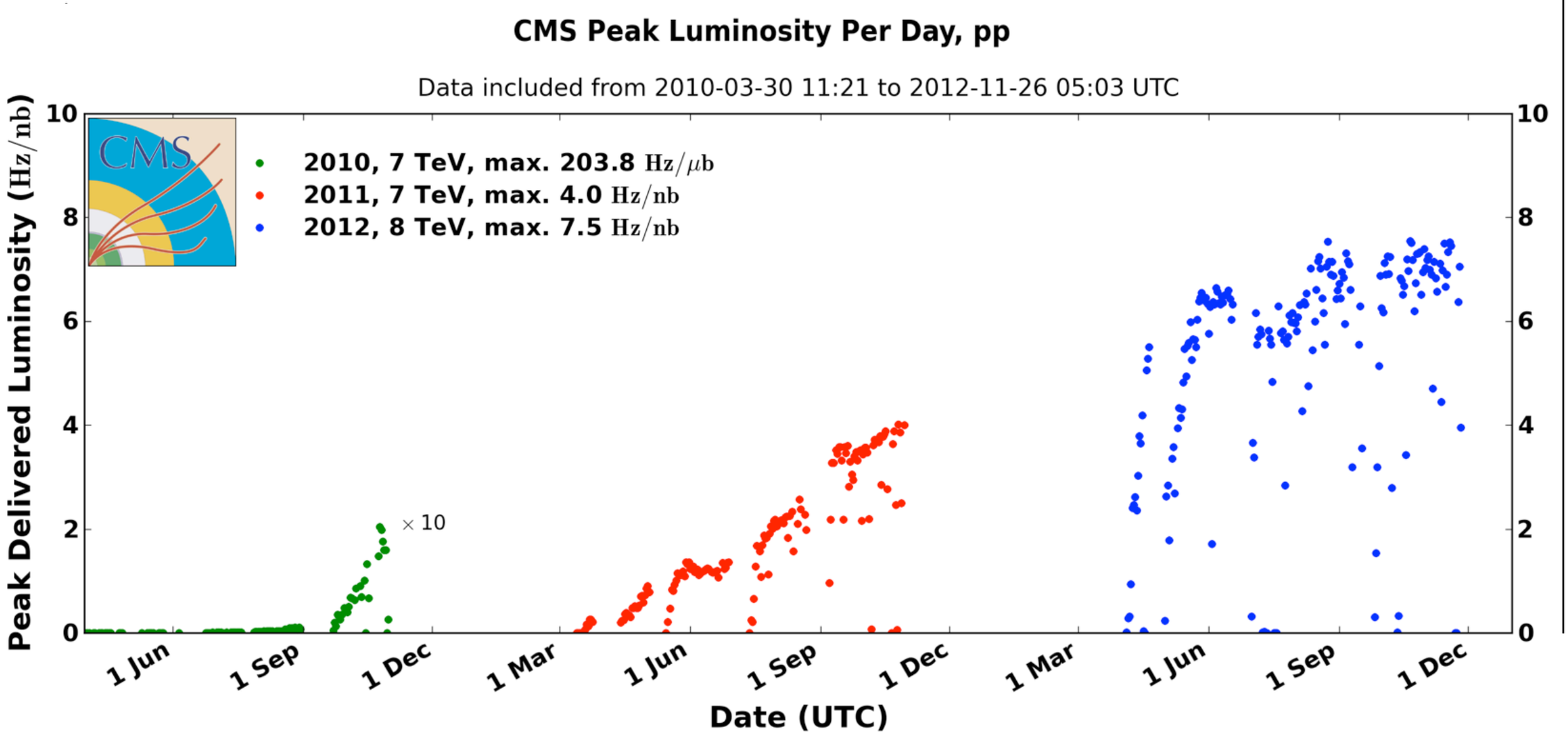
Iterative tracking in 2011 (CMSSW 42x)

Tracking evolution from from $10^{32}/cm^2/s$ (2011) to $8\times10^{33}/cm^2/s$ (2012)

aka the standard way to improve...

The constraint of prompt reconstruction

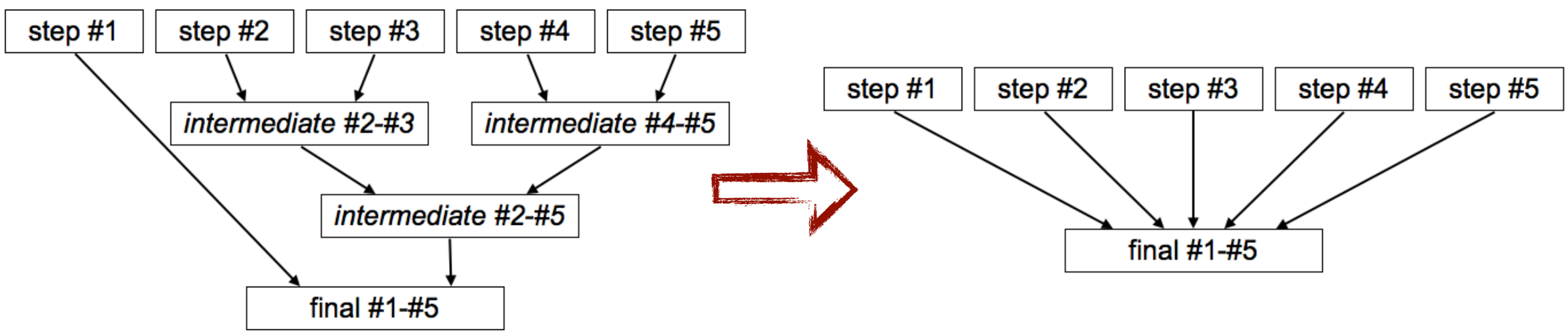
Prompt reconstruction is crucial for a discovery experiment: quasi real-time physics results, fast deep feedback on detector conditions. It requires data to be processed at the same pace as they are produced. Resources and algorithm speed must adapt to the instantaneous luminosity. The tracking reconstruction software was too heavy (CPU time and memory) for prompt reconstruction and it was improved in two phases: fall 2011,



Fall 2011 campaign: from CMSSW42x to 44x (1)

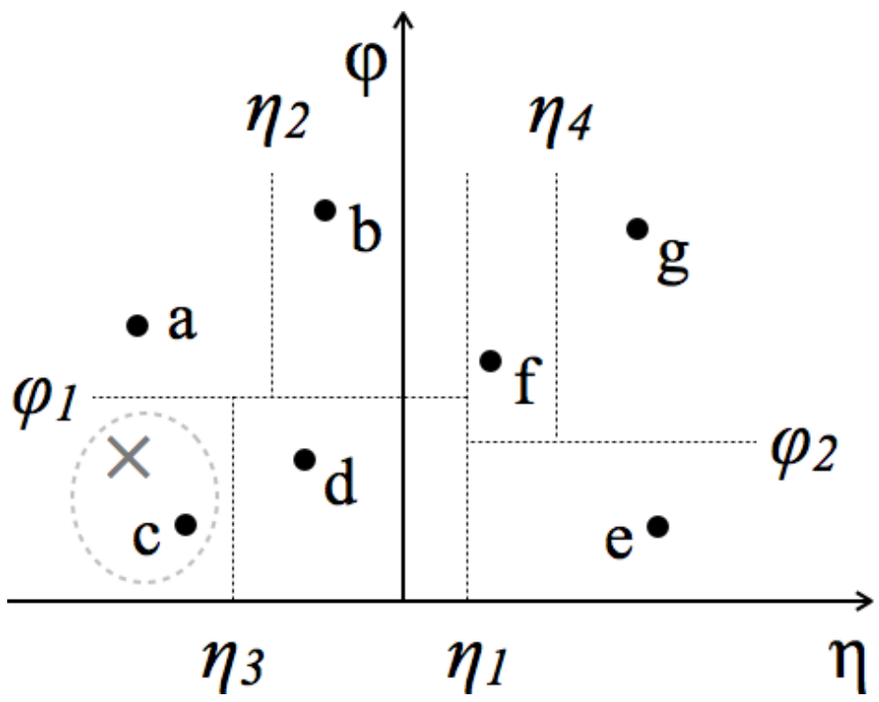
Copy-less hit masking Each step of the iterative tracking works on the hits not yet associated to any track. This was done by creating a new collection of surviving hits at each step. Implemented a data member to store masking bits
Batch cleaning of seeds successfully propagated (*track candidates*) The track candidates are filtered in 1k batches to avoid storing too many of them
Efficient quality assignment Each iterative step assigns tracks to a quality tier. Old implementation just created a track copy per each tier; implemented a data member to store the quality tier bits

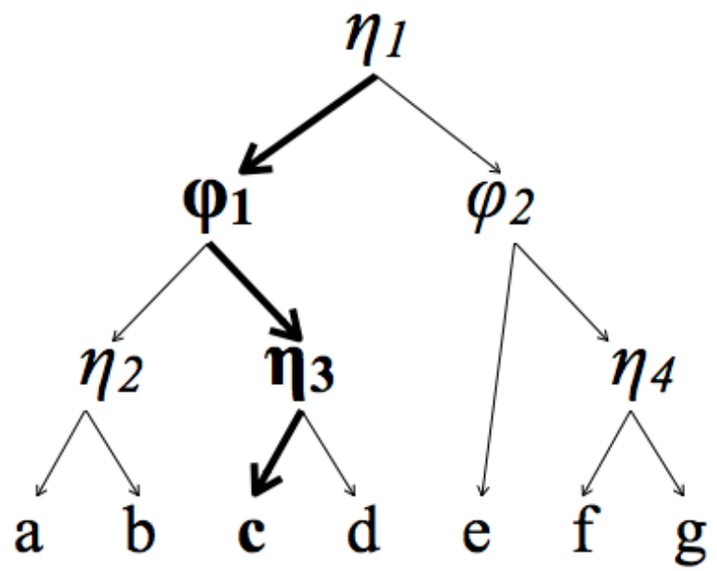
Efficient track merging The track collections resulting from steps have to be merged and cleaned. The merging algorithm has been improved by getting rid of intermediate collections.



Fall 2011 campaign: from CMSSW42x to 44x (2)

Particle flow links The PF algorithm links tracks to calorimetric clusters in the (η,φ) space. Done in 42x by CPU intensive nested loops, with a complexity that scales quadratically with the multiplicity N. In 44x implemented a ***kd-tree*** based algorithm: the (η,φ) space is split into appropriate domains, each containing one single object, organized in a tree. The cluster closest to a given track is found with a very fast binary search that ends up in the closest neighbor domain. The complexity that scales as N·logN. Already extended to other CMSSW modules by the implementation of a generic kd-tree class.





Fall 2011 campaign: from CMSSW42x to 44x (3)

Several optimization in object reconstruction like photon conversion, vertices, nuclear interactions with significant CPU time gain
Iterative tracking A factor 2.5 of improvement in the CPU time has been obtained by optimizing the iterative tracking. The net effect is an increase of the effective P_T threshold for track reconstruction together with tighter constraints on impact parameter. This configuration results in a reduced efficiency for P_T <300MeV/c but an efficiency for P_T>0.9 GeV/c larger by ~1% with a ~8% reduction of the fake rate.

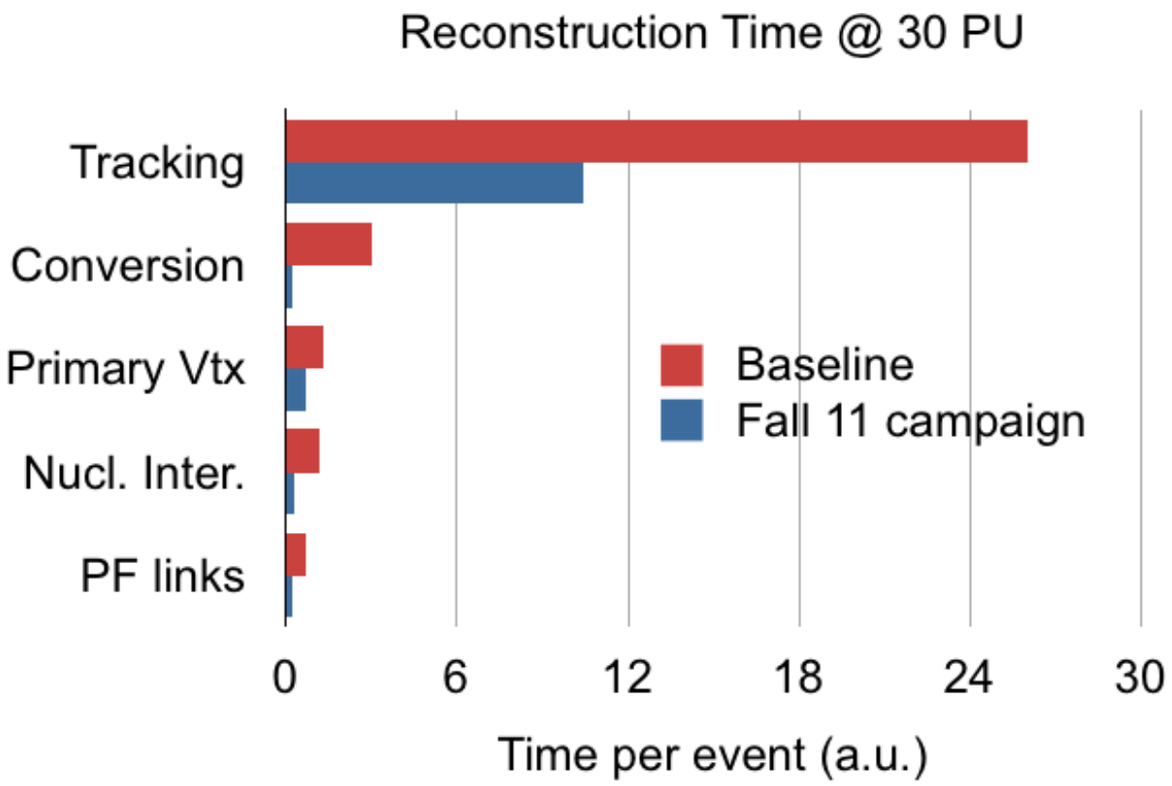
#step	seed type	seed subdetectors	$P_T^{\min}$ [GeV/c]	d_0 cut	z_0 cut
0	triplet	pixel	**0.6**	**0.03** cm	**4.0**σ
1	**triplet**	**pixel**	**0.2**	**0.03** cm	**4.0**σ
2	pair	**pixel**	0.6	**0.01** cm	**0.09** cm
3	triplet	pixel	**0.2**	**1.0** cm	**4.0**σ
4	triplet	pixel/TIB/TID/TEC	**0.35-0.5**	2.0 cm	10.0 cm
5	pair	TIB/TID/TEC	**0.6**	2.0 cm	**10.0** cm
6	pair	TOB/TEC	0.6	2.0 cm	30.0 cm

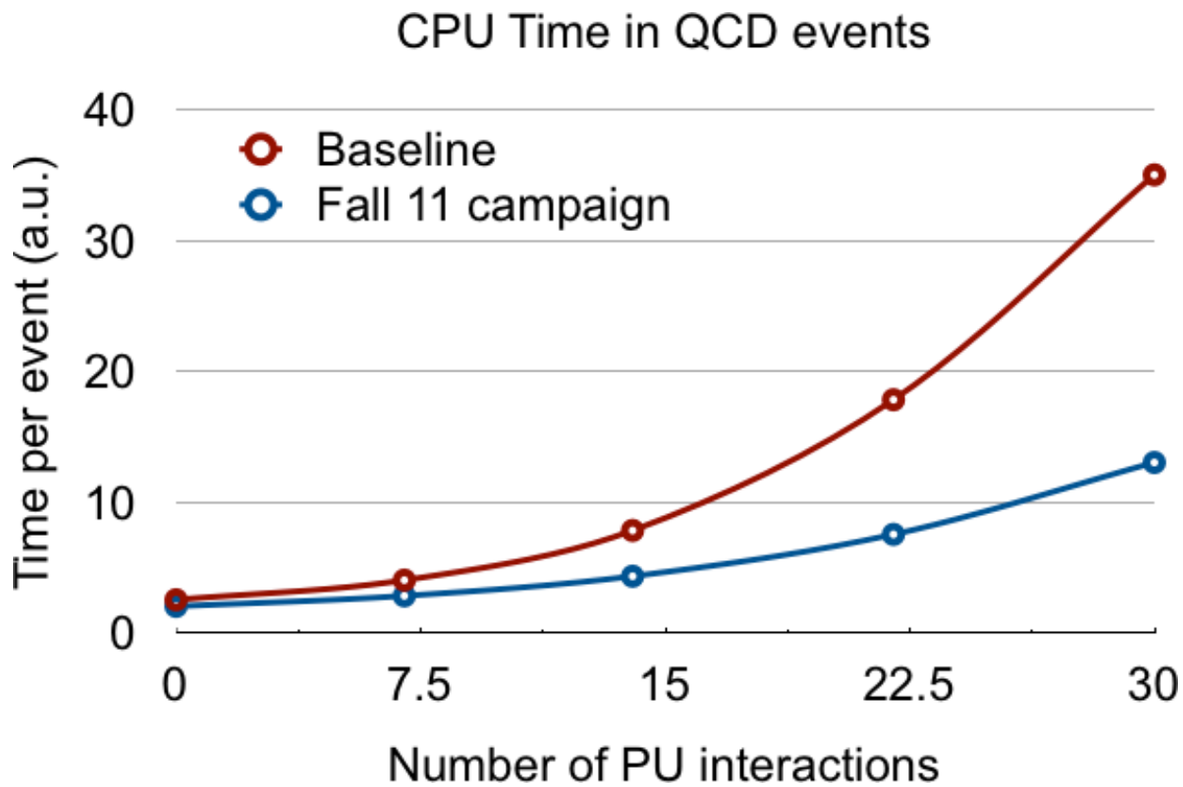
Iterative tracking in late 2011 (CMSSW 44x) / In **bold** the changes with respect to 42x

Results of fall 2011 campaign

reconstruction CPU time @30PU | reconstruction CPU time vs. PU

Simulated QCD events



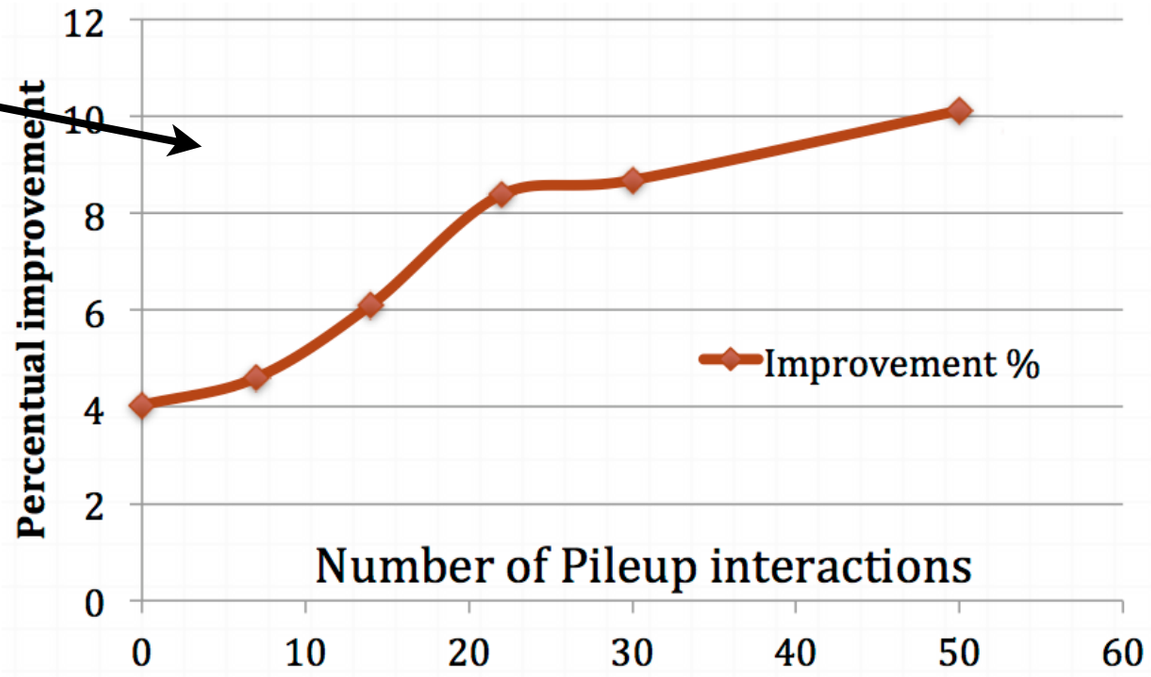


Spring 2012 campaign: from CMSSW44x to 52x (1)

Change of compiler switch from gcc 4.3.4 to gcc 4.6.2: faster code generated (compiler specific optimizations), C++11 support and autovectorization

JEMalloc standard malloc replaced by JEMalloc, highly performant and able to better redeem memory

Relative change of CPU reconstruction time vs. PU Simulated QCD events



Improved ROOT version from 5.27 to 5.32 that features several improvements, especially in I/O with less memory required.

Several design modifications to improve speed and memory consumption; for example, 10% gain in speed and in some 100MB of resident set size (RSS) saved per event from the devirtualization of the BasicTrajectoryState class (an ancillary class for track reconstruction); stereo hit class reduced a factor three in size with RSS memory down to 50MB from 150MB

Spring 2012 campaign: from CMSSW44x to 52x (2)

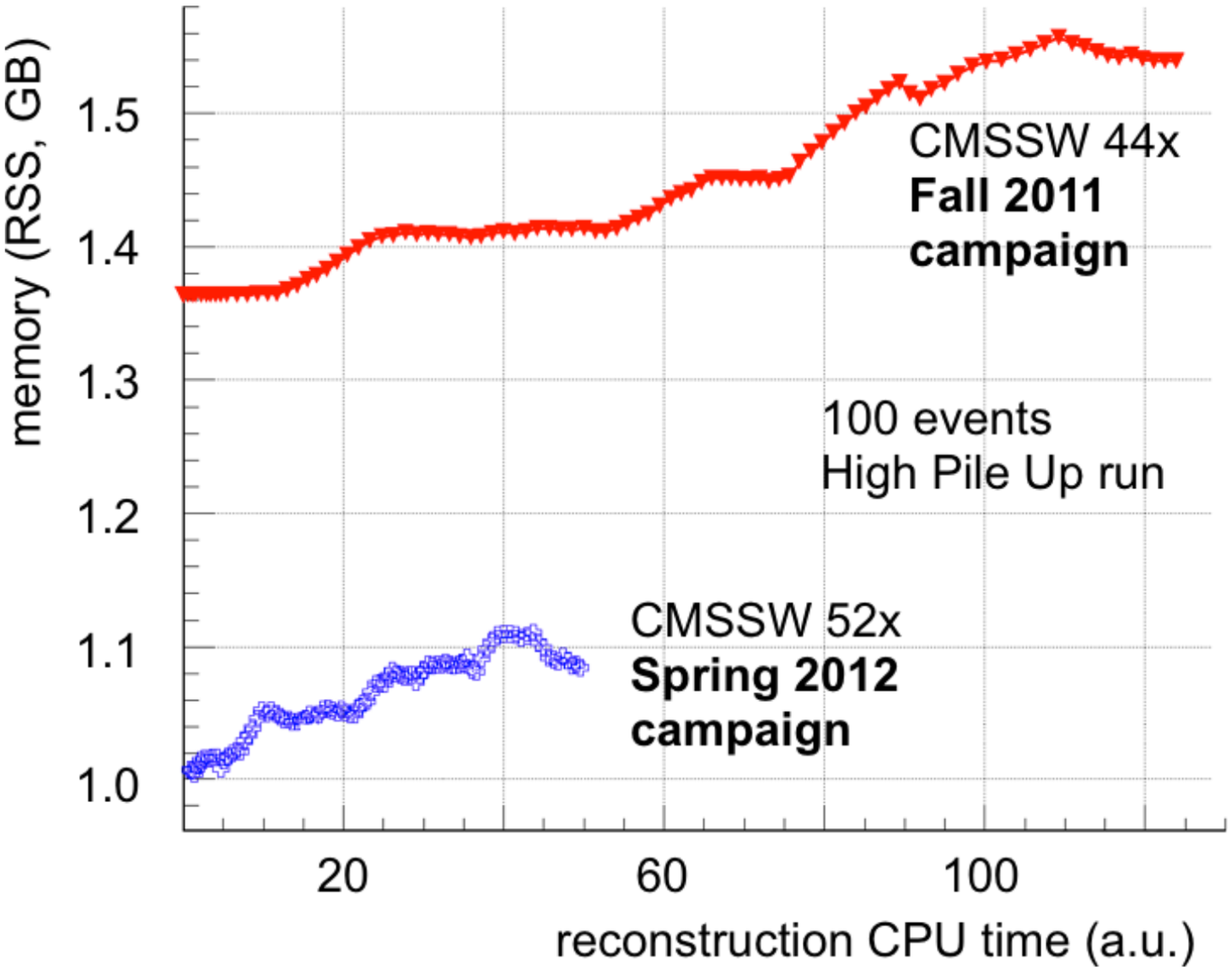
Offline vertexing based on a deterministic annealing algorithm improved: loops autovectorized (new compiler), exponential functions replaced with fast autovectorizable inlined double precision versions; some configuration parameters optimized. 3x gain in CPU time with no change in performances
Cluster-shape based seed filtering extended to almost all seeding step. 1.5x improvement in CPU time. Fake rate is reduced by ~ 20%.
Iterative tracking Tiny optimization plus upgrade of the final track cleaning and selection criteria. No efficiency change for prompts tracks with P_T>0.9 GeV/c, but fake rate ~35% down.

#step	seed type	seed subdetectors	$P_T^{\min}$ [GeV/c]	d_0 cut	z_0 cut
0	triplet	pixel	0.6	**0.02** cm	4.0σ
1	triplet	pixel	0.2	**0.02** cm	4.0σ
2	pair	pixel	0.6	**0.015** cm	0.09 cm
3	triplet	pixel	**0.3**	**1.5** cm	**2.5**σ
4	triplet	pixel/TIB/TID/TEC	**0.5-0.6**	**1.5** cm	10.0 cm
5	pair	TIB/TID/TEC	0.6	2.0 cm	10.0 cm
6	pair	TOB/TEC	0.6	2.0 cm	30.0 cm

Iterative tracking in 2012 (CMSSW 52x) / In **bold** the changes with respect to 44x

Results of spring 2012 campaign

RSS memory as a function of running time
in CMSSW 44x and CMSSW 52x for a reconstruction job of 100 real data events from the 2011 special run with high PU.



The challenge of 2015

2015: Scary PU scenarios...

Potential Performance after LS1

- Determined by the performance of the injector chain
- Different collimator scenarios, not detailed here
- LHC Injector Upgrade (LIU) fruits after LS2

J. Uythoven
CMS week Lisbon

	Number of bunches	β* [m]	Half X-angle [μrad]	Ib SPS	Emit SPS [um]	Peak Lumi [cm^{-2}s^{-1}]	~Pile-up	Int. Lumi [fb^{-1}]
25 ns	2800	0.50	190	1.2e11	2.8	1.1e34	23	~30
50 ns	1380	0.40	140	1.7e11	2.1	1.8e34 β* level	81 β* level	?
25 ns low emit	2600	0.40	150	1.15e11	1.4	2.0e34	48	52
50 ns low emit	1200	0.40	120	1.71e11	1.5	2.2e34	113	?

Presently at 4 TeV, β = 0.6 m, half X-angle 145 μrad*

Strategies for 2015

= **Tracking code reengineering**; major redesign of the tracking code to implement parallelization and vectorization / joint effort between offline people for the framework (modifications almost transparent for the user) and tracking developers (for modifications to be implemented straight into the tracking code)
= **Generic improvements as in 2011/2012** (smarter coding, compilers, seed cleaning) and iterative tracking tuning | tracking developers
= **New tracking algorithms** (Hough transform) | tracking developers

Several invited talks in the past months on this subject, with contributions from present and past experiments, also heavy ion.

Developments to be done during LS1 but mainly in 2013, with 2014 devoted to full validation and MC productions

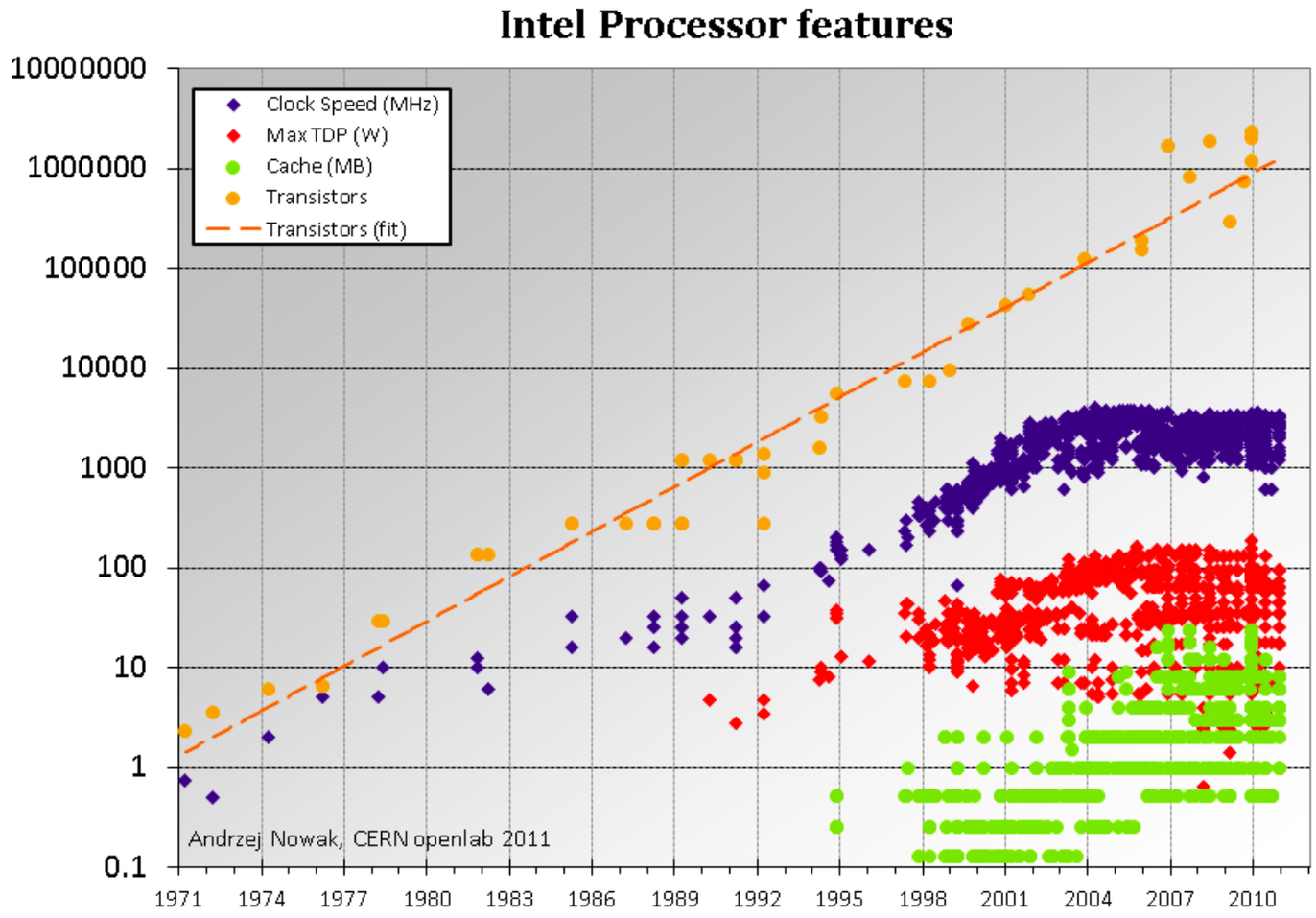
Fishing for new ideas also in this workshop ;-)

The Performance Challenge

- Since circa 6 years the single CPU clock frequency has not increased anymore:

  "The free lunch is over"

- The additional transistors are mainly used to implement:
  - More CPU Cores
  - Larger Caches
  - Larger Vector Units



Source: Andrzej Nowak – CERN OpenLab

To be able to take advantage of the available hardware, software needs to:

- Use Multi-Process / Multi-Core techniques to fully load the machine's cores
- Access the vector units provided by the machine for calculations

THauth

Parallelization in CMS...

CMSSW is presently designed to analyze/process events serially and is organized in modules; CMSSW can be made parallel:
i) **at the module level**: several tasks are performed in parallel (example: several track building in parallel within the same iterative step)
ii) **at the event level**: several modules running in parallel
iii) **at the framework level**: several events running in parallel

Go parallel for CPU time but for memory also. In this respect running jobs in parallel is not equivalent to ***iii)***.

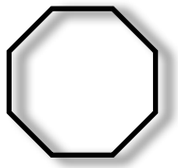
INTEL TBB chosen as parallelization libraries for the maximum flexibility.

Parallelization at the event level

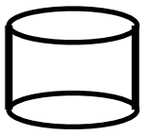
Simple event analysis flow...

An Event is filtered by Paths

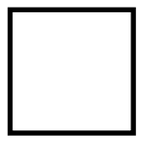
Paths run in parallel

Paths hold a list of Filters 

Filter runs only if previous Filter passes

EndPaths hold OutputModules 

EndPaths run in parallel after Paths finish

Producers make data 

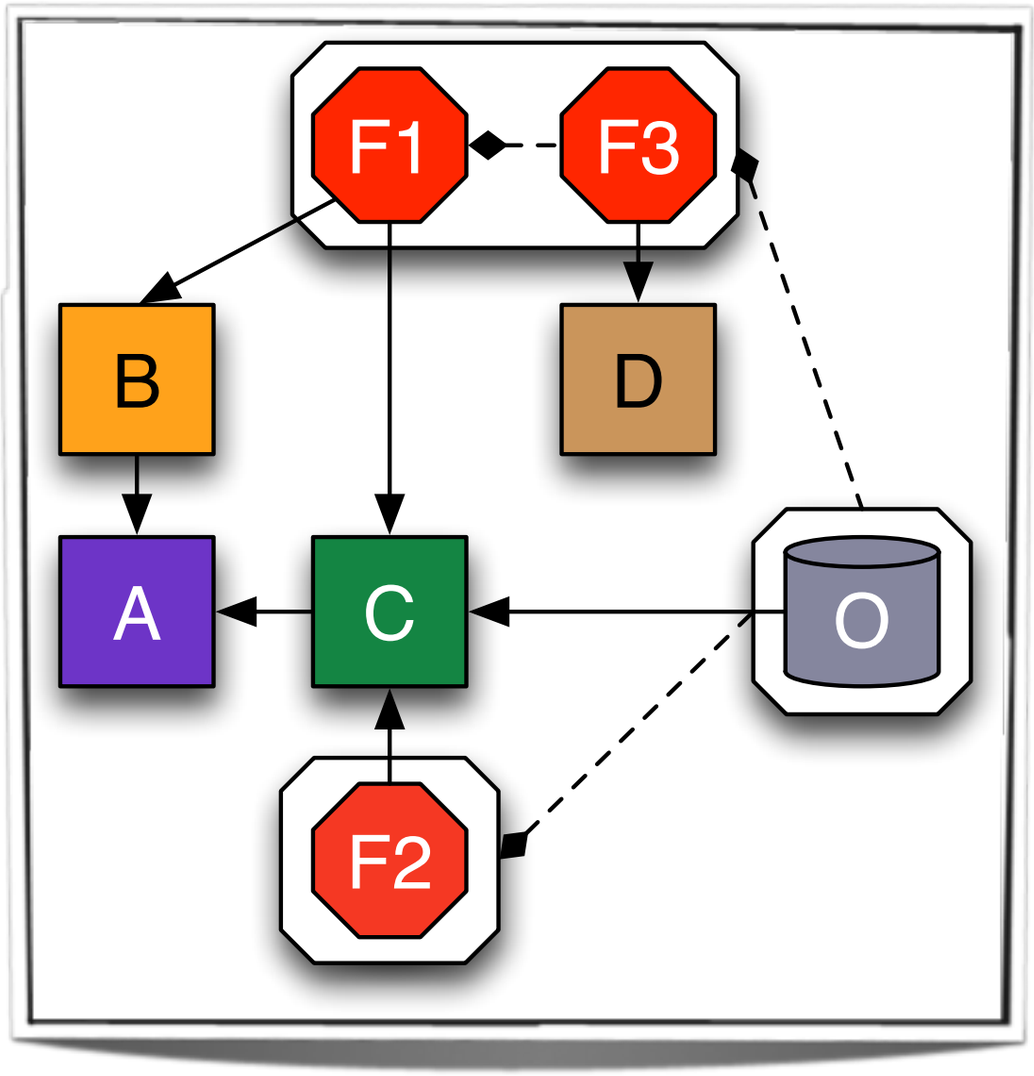
Run first time their data is requested

Producers run in parallel

Filters, Producers & OutputModules

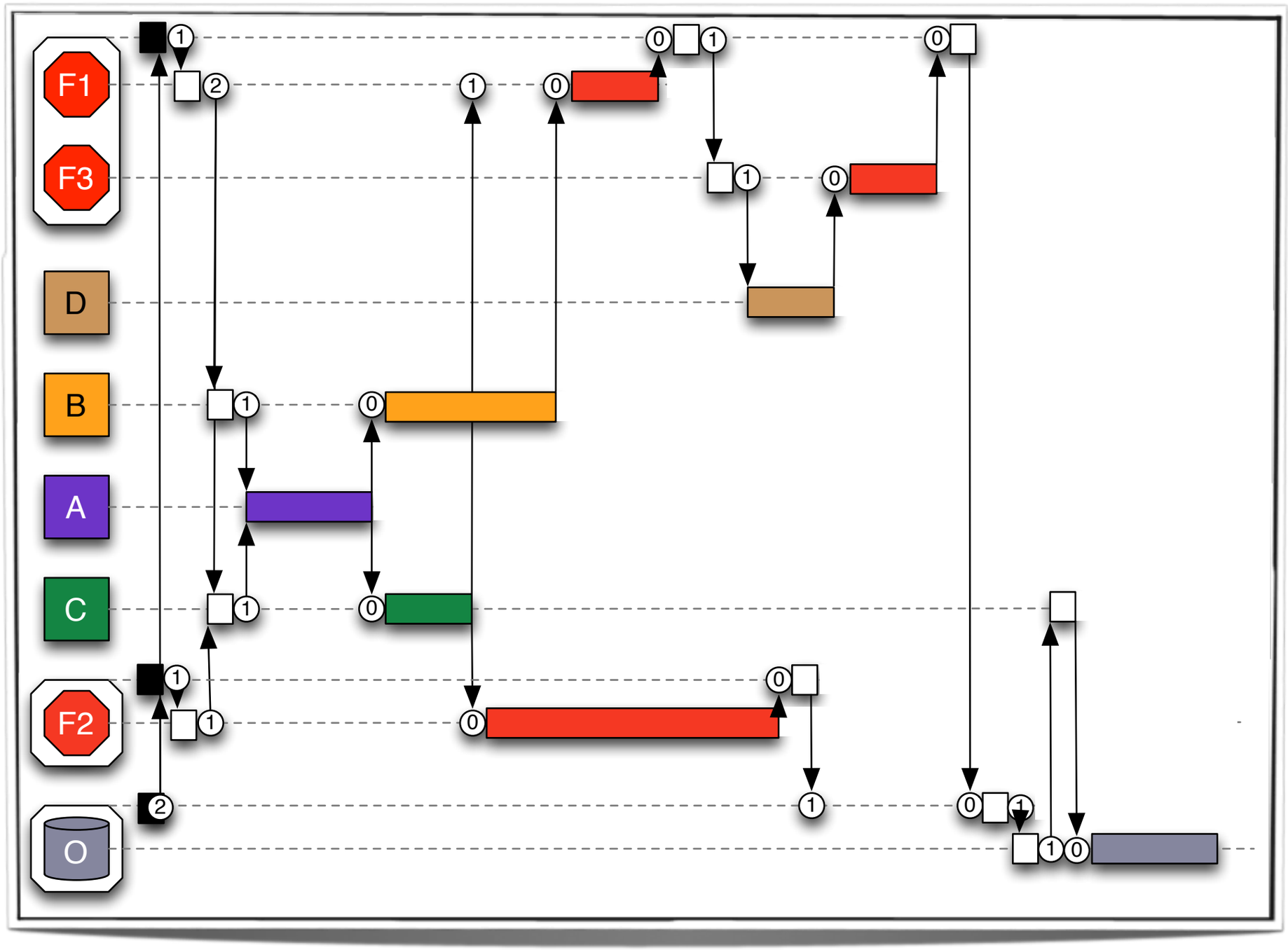
All are referred to as Modules

Run only after their input data is available



CJones

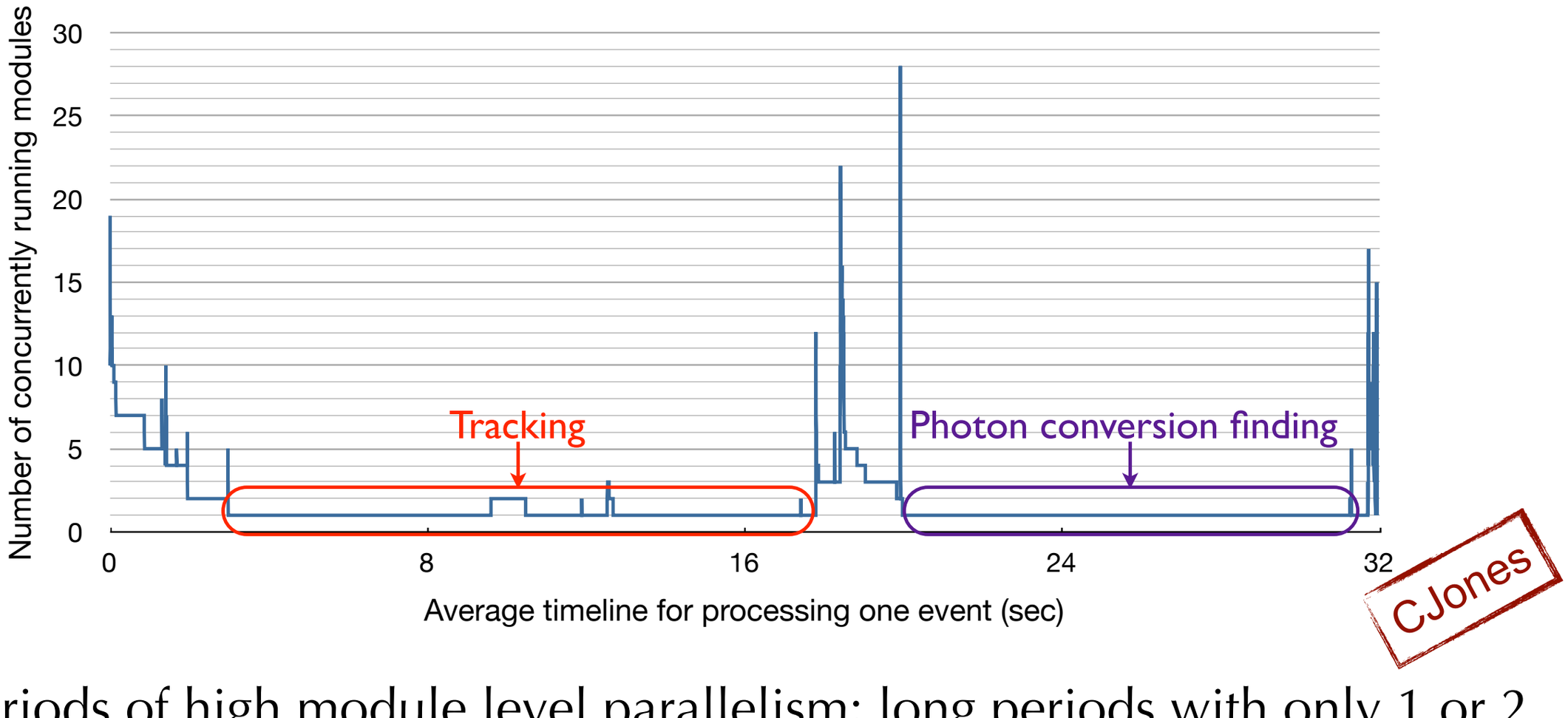
...that can be made parallel



First establish dependencies; then run in parallel as many modules is possible...

Simulation

Done by implementing module dependencies and running time for a typical full reconstruction path (this in an old CMSSW!)

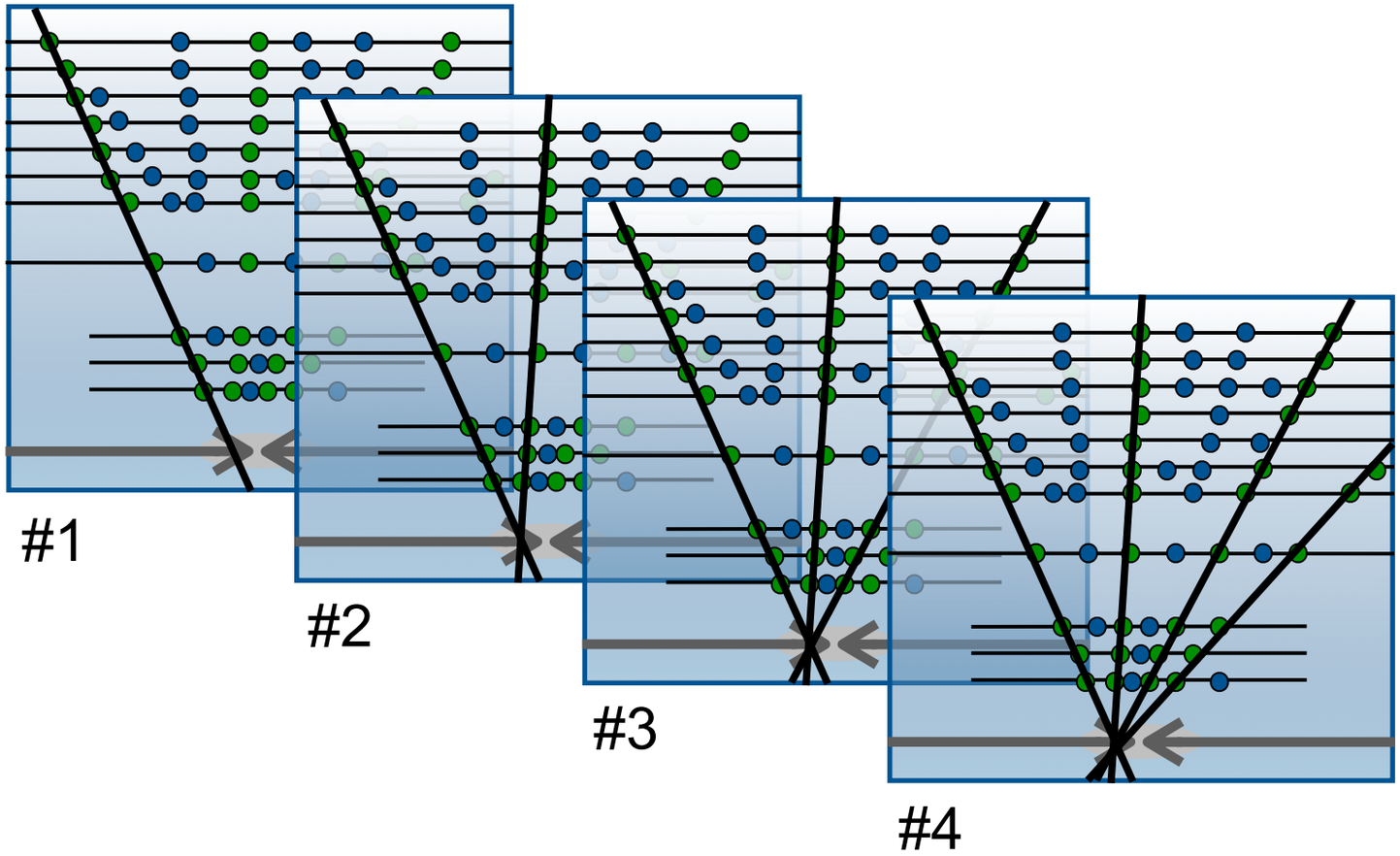


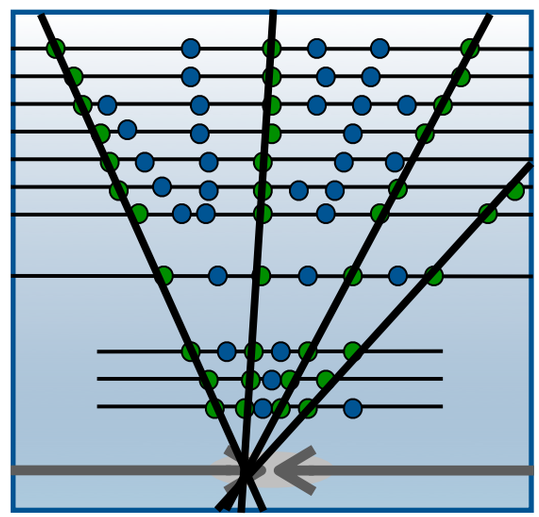
Short periods of high module level parallelism; long periods with only 1 or 2 modules; first period is tracking; second period is photon conversion finding ***Parallelizing within those module*** or ***running events in parallell*** would be beneficial

Parallelization at the module level

Parallelization within the modules

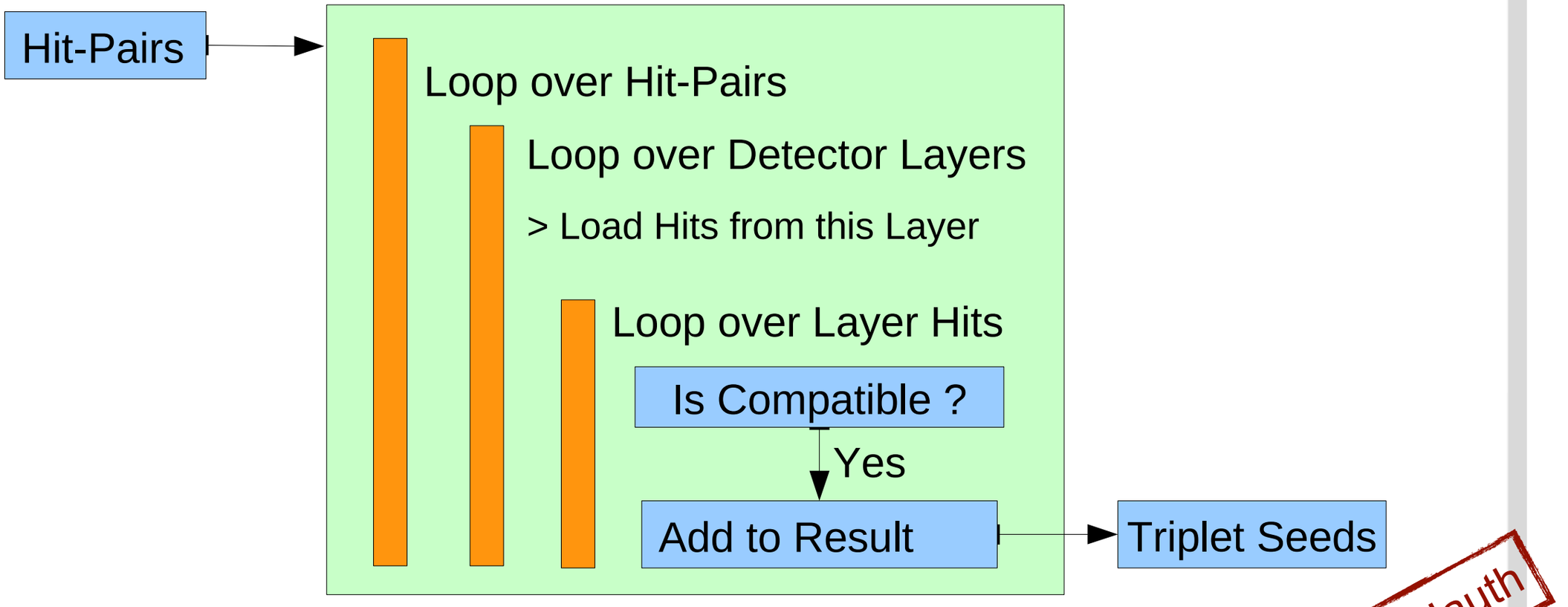
Identify modules that perform tasks where parallelization can be easily implemented; tracking is a clear candidate, i.e. track building after seeding (pattern recognition) within an iterative step
Made the algorithms thread safe (could not be trivial in case of tracking)
CAVEAT: going thread safe could result in significant memory overhead...





Triplet Seeding in CMS

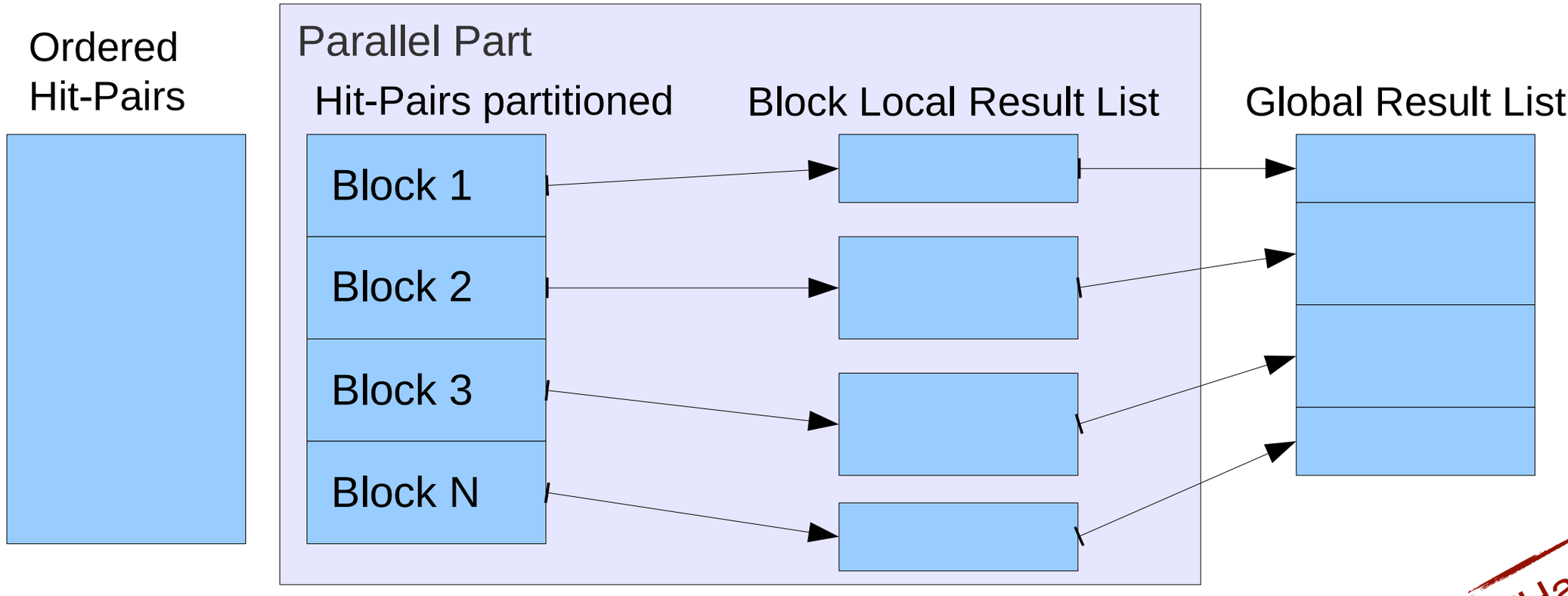
- Energy deposits of charged particles in the CMS tracker are reconstructed as hits
- Before starting the track reconstruction, seeds from three topologically compatible hits in the tracker are searched: hit-triplets
- Starting with two hits which have been already found to be compatible (hit-pair) possible hits of subsequent tracker layers are evaluated
- This seeding procedure amounts to about 14% of the overall runtime of the CMS Reconstruction



THauth

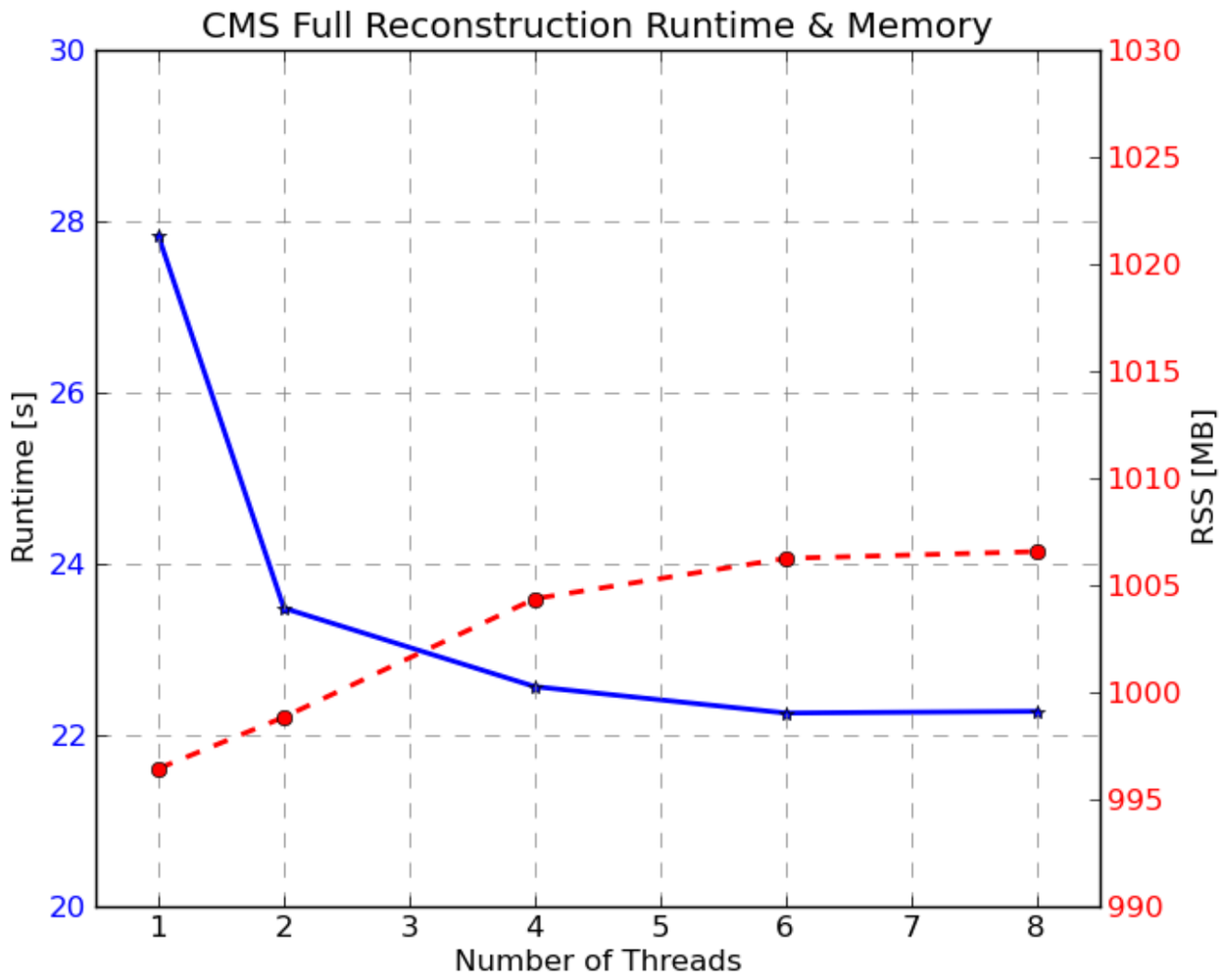
Triplet Seeding in Parallel

- Preserving the ordering of the output collection is essential for subsequent algorithms and validation purposes
- Filling an unsorted output collection with multiple threads at the same time can result in non-reproducible results
- We used a scheme to partition the input collection of hit-pairs in equally sized blocks
- A private result list is associated with every block and is merged in the correct order into the global result list at the end of the algorithm execution. No explicit sorting needed.
- The distribution of the blocks to the available threads is handled by TBB



THauth

CMS Reconstruction Runtime and Memory



- Each thread adds about 1 MB to the overall memory consumption. Negligible compared to the memory footprint of the application (~ 1 GB) > lightweight scaling
- Higher-than-expected scaling from 1 to 2 cores, probably due to the positive effects of using the L1/L2 caches of two cores simultaneously

THauth

Parallelization at the event level

Event parallelization: parallel transition management

In a serial event processing let's call a **transition** the change of boundary conditions; change of conditions implies loading into memory of data (i.e. detector conditions) needed by the event analysis, to be kept up to next transition or last event.

Transitions in CMS: Begin of Run / End of Run; Begin Lumisection / End of Lumisection

A lumisection is the smallest time entity with respect to conditions; the shortest possible IOV (Interval-of-Validity) of a condition record in the DB is the lumisection; a lumisection corresponds to ~23s

New Concepts

Global

Sees transitions on a 'global' scale
- see begin of Run and begin of Lumi when source first reads them
- sees end of Run and end of Lumi once all processing has finished for them

Multiple transitions can be running concurrently
- two or more begin or end Runs (for different runs)
- two or more begin or end Lumis
- and end can be occurring while another begin is running

Events are not seen 'globally'

Stream

Processes transitions serially
- begin run, begin lumi, events, end lumi, end run

Multiple streams can be running concurrently each with own events
- One stream only sees a subset of the events in a job

Present cmsRun is equivalent to running with only one stream

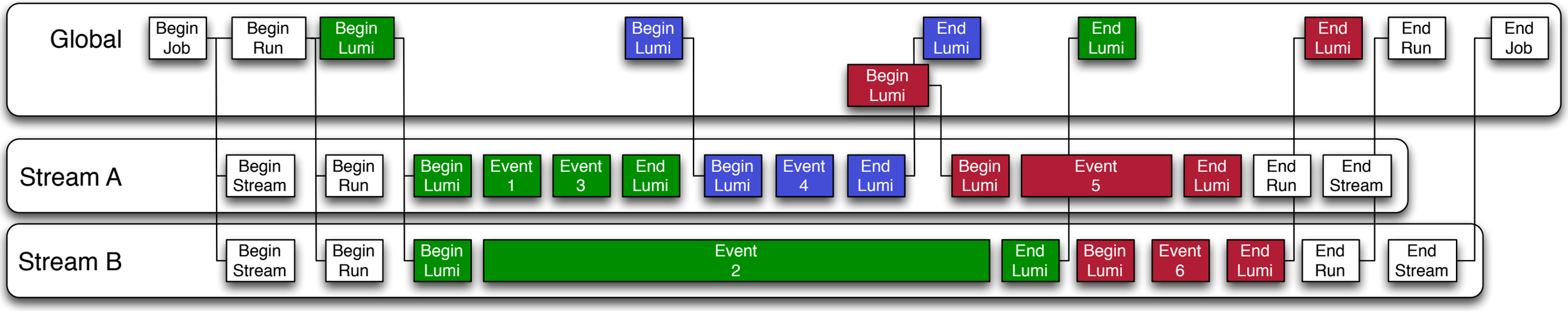
Paths and EndPaths are a per Stream construct
- The same module can be shared across Streams
- The Stream knows if a module was run for a particular event

CJones

Transitions

Minimal dependencies between transitions seen by a module

BeginStreams can't run until BeginJob finishes
Stream's BeginRun can't run until Global's BeginRun finishes
Global BeginLumi can't run until Global's BeginRun finishes
Stream's BeginLumi can't run until Global's BeginLumi finishes
Global EndLumi can't run until all Streams' EndLumi's finish
Global EndRun can't run until all Streams' EndRun finish
EndJob can't run until all EndStreams have finished



NOTE

Global BeginLumi can happen before previous Global EndLumi is called
Same for BeginRun and EndRun
Stream transitions always occur in the 'expected' order

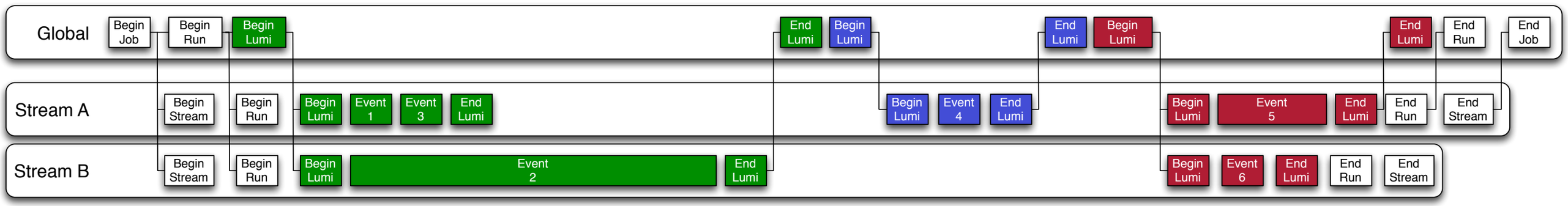
CJones

Serial Transitions

Case where Global transitions must be serial

Bad when

Large variation in the time it takes to process an event
Very few events in Lumis



Present DQM requires this serialization

DQMStore works as a giant globally shared memory
Same histograms are reused for different Runs and Lumis
Would require framework to enforce a global serialization instead of just a per module serialization

CJones

Ideas for new tracking algorithms

Bringing new ideas in CMS

Manpower is the main limitation in developing “new” ideas in CMS.
Past experience teaches us that it is difficult to bring into production and maintain several tracking algorithms that need to coexist:

In the past developments for:
= road search
= deterministic annealing filter
= elastic arm

Currently used:
= Combinatorial track finder (kalman filter)
= GSF

Our first thoughts about new algorithms to be imported in CMS is limited to Hough transform.

Hough transform in CMS?

Hough transform techniques are widely used in high energy physics experiments for pattern recognition / track reconstruction

Typical use cases are fast reconstruction in high multiplicity environments

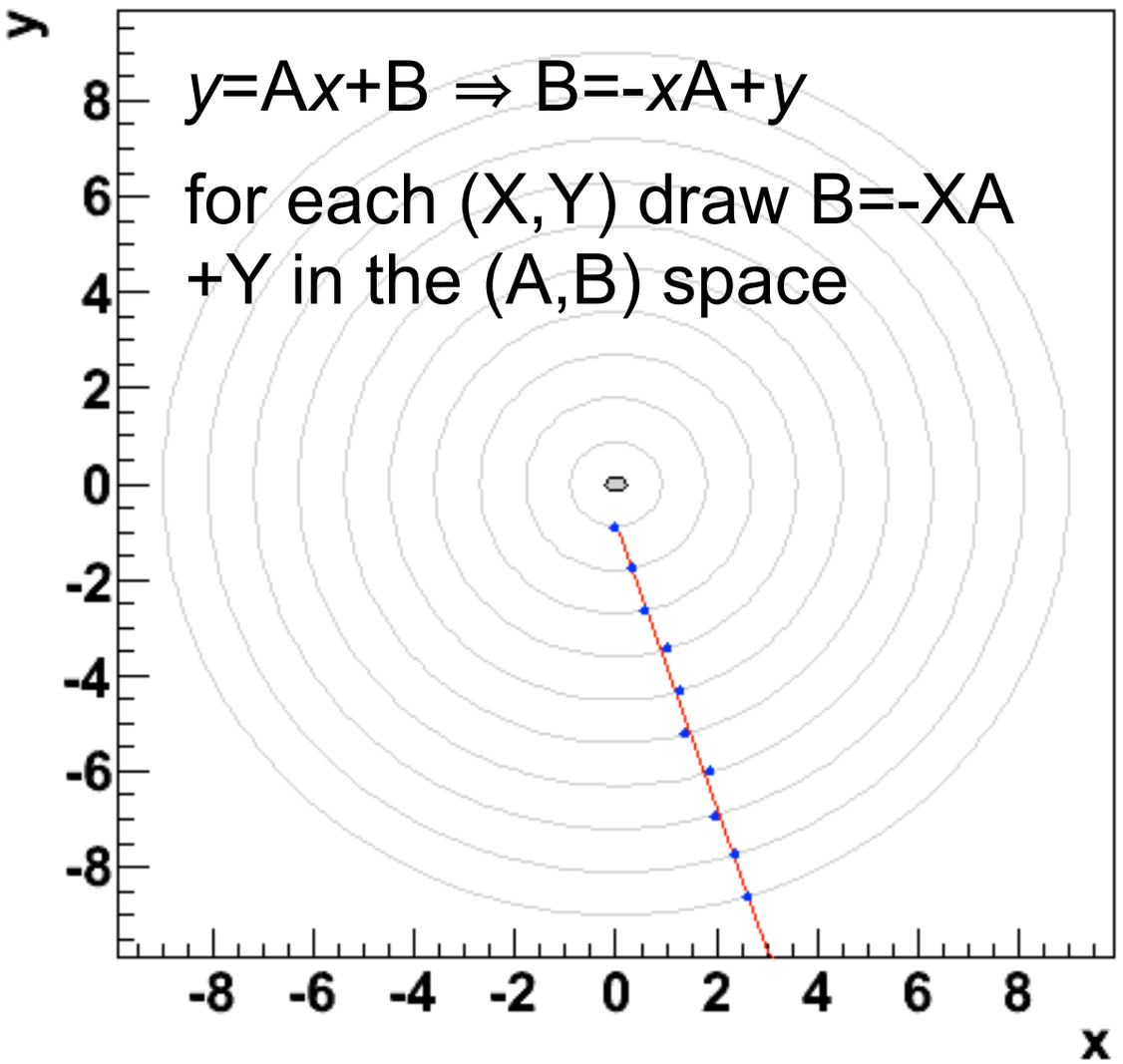
Used in trigger algorithms (Hera-B) and in heavy ion experiments (Phenix, Alice)

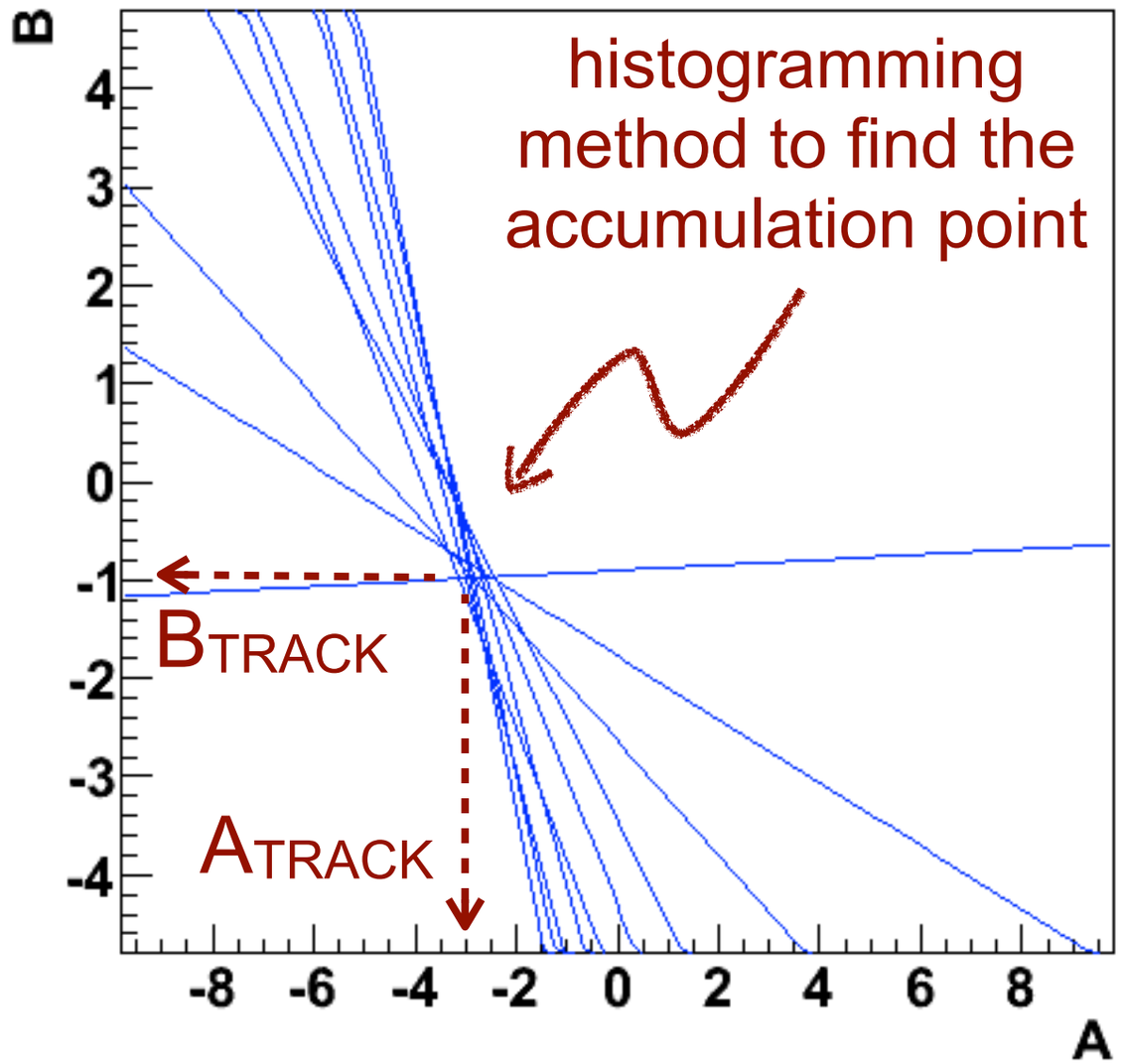
The technique is completely new in CMS

Need to understand if it is worthwhile to endeavour in the project

Hough transform basics

Each hit is compatible to many trajectory hypotheses that can be represented by curves in an appropriate trajectory parameter space (typically straight lines); the intersection between many of these curves is a reconstructed track. So hits are transformed into lines (or curves, more in general) in the track parameter space by an appropriate conformal transformation, and accumulation points are identified.





Current tracking limitations

Iterative tracking is performing well in early steps; early steps are able to provide fast the bulk of tracking efficiency and, timing wise, they are pretty stable with respect to PU.

Problems are in later steps designed (loosen cuts) to recover efficiency for more difficult (e.g. displaced) tracks.

Late steps result in too many fake seeds with respect to early steps; each seed needs to be propagated in the attempt of building a track. This is time consuming and time per final good track is not favorable for these later steps.

The long story short: combinatorics...

Track building (e.g. full track reconstruction) is eventually able to get rid of these fake proto-tracks; similarly, more information should be used to clean up not useful seeds in advance. But this requires time.

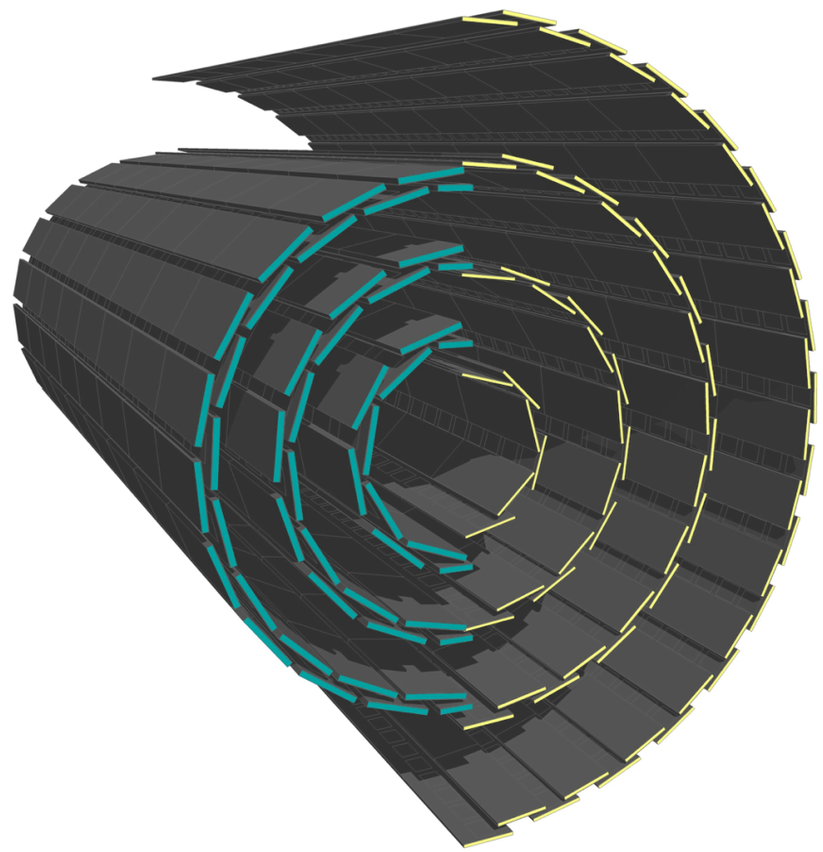
New developments should address this paradox.

Proposal for Hough transform applications in CMS

Hough transform methods cannot handle energy loss and multiple scattering; they are probably not suitable for full track reconstruction in CMS (where material effects are substantial).

Nevertheless, Hough transform could represent a natural way to combine more information than just two/three hits at the seeding level in a fast way and without entering in the time consuming propagation. Given the reduced lever arm and the reduced resolution needed, material effects can be probably neglected at the seeding level.

Proposal for Hough transform method implementation:
= seeding in the outer tracker layers combining information from more than three layers;
= 4-layer seeding for the phase-I upgraded pixel detector.



Conclusions

CMS tracking performances are excellent

CMS tracking has been updated to match the steady increase amount of luminosity from 2010 to 2012; important lessons learned for the big step facing us in 2015... the performance gains are the result of several different improvements deployed at any level: hardware architecture, framework, analysis code...

A major tracking reengineering effort is being organized; the main target is bring parallelization at several level of tracking (and reconstruction) software. Net year will be crucial in this respect.

Meanwhile, innovative (CMS-wise) tracking algorithms should be developed and evaluated; plan to test Hough transforms for specific applications.

Any other idea is very welcome; everybody is encouraged to join CMS!