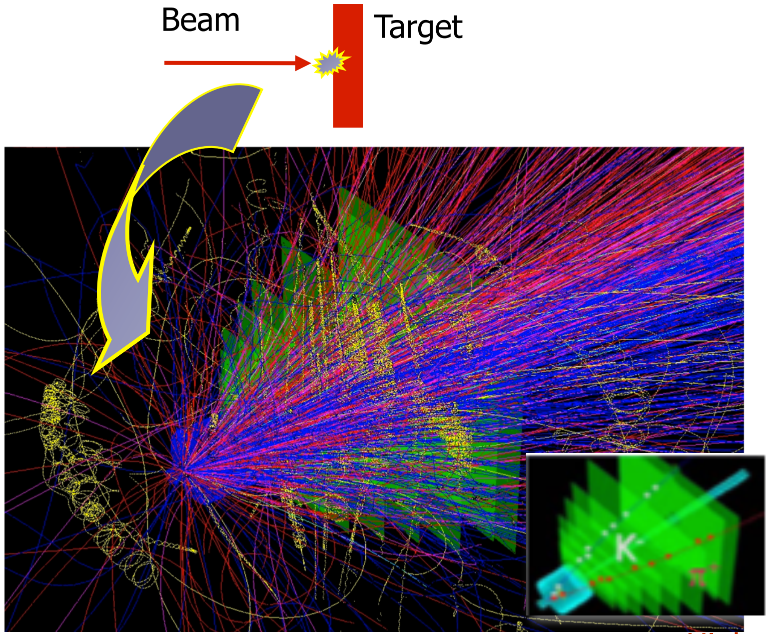
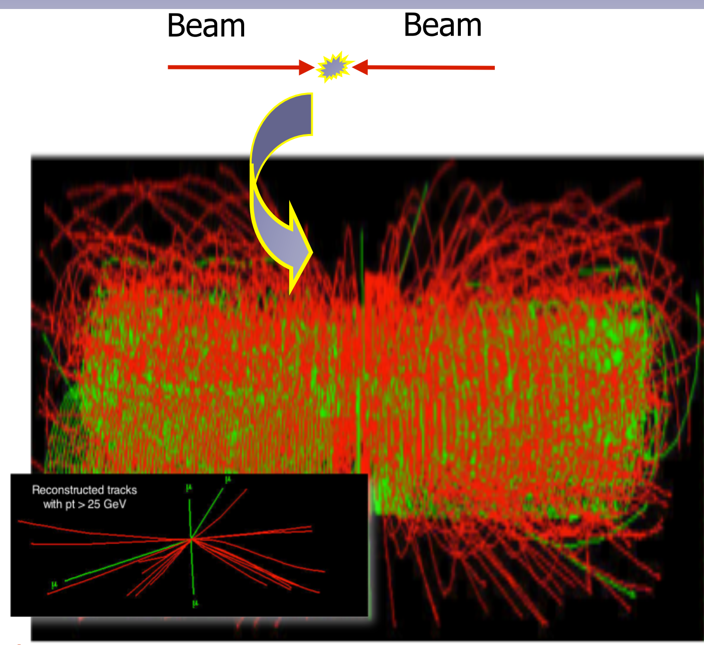
# HEP Experiments: Fixed-Target and Collider




Inelastic collisions
$10^7 - 10^9$

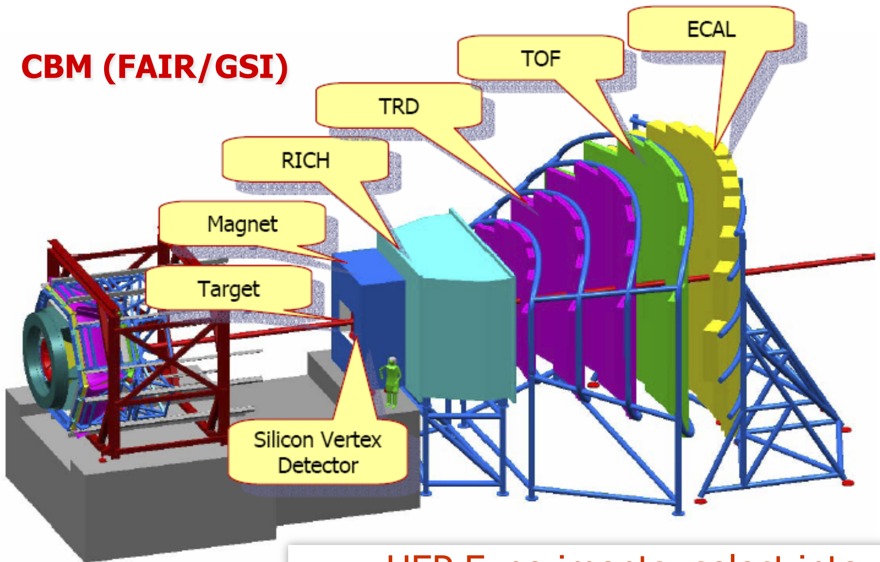
$10^{11}$

Signal events
$10^2 - 10^{-2}$

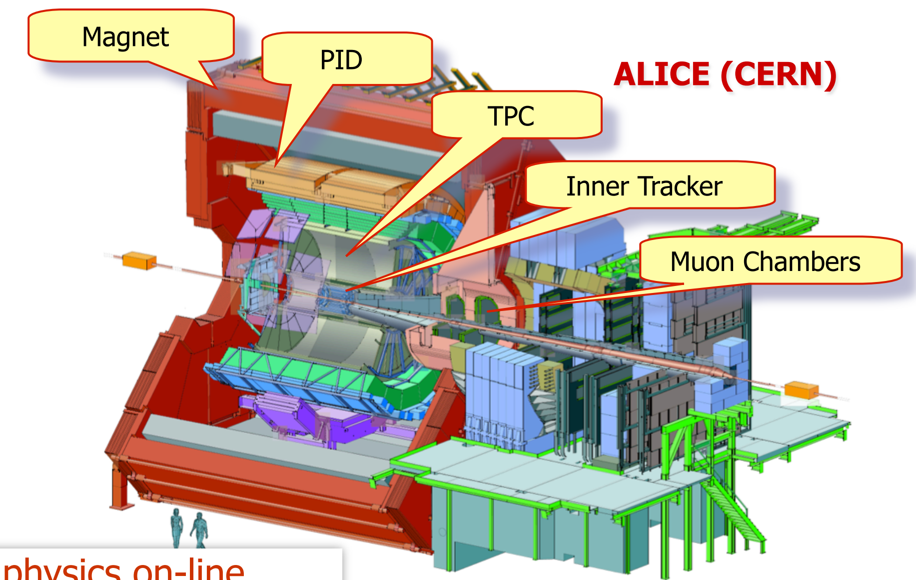



High energy = high density + high rate

**CBM (FAIR/GSI)**



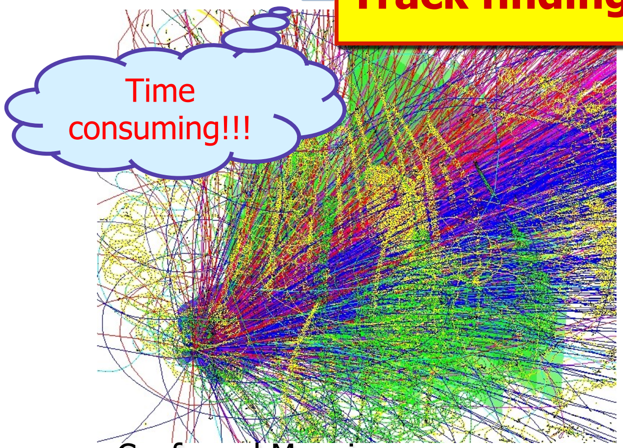

**ALICE (CERN)**




HEP Experiments: select interesting physics on-line

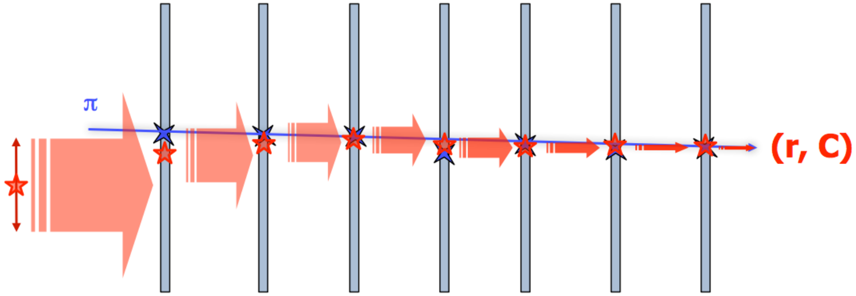
# Stages of Data Reconstruction

1 **Track finding**



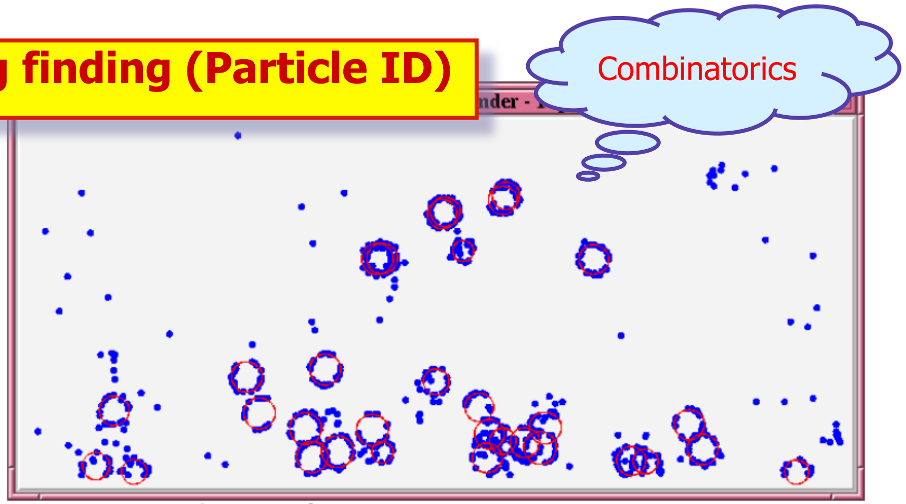

- Conformal Mapping
- Hough Transformation
- Track Following + Kalman Filter
- Cellular Automaton + Kalman Filter

2 **Track fitting**



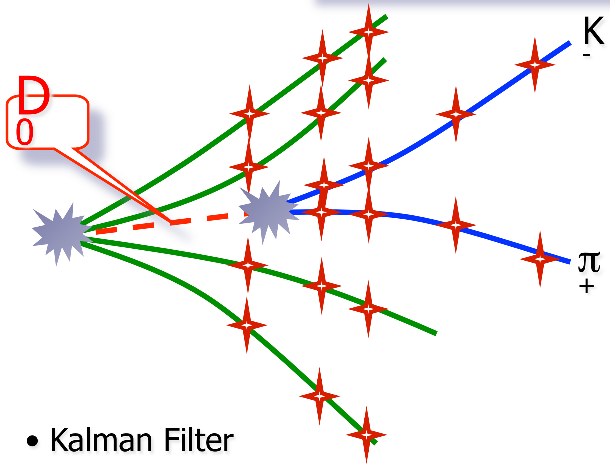

- Kalman Filter

3 **Ring finding (Particle ID)**




- Hough Transformation
- Elastic Neural Net

4 **Particles finding**




- Kalman Filter

# Global Methods: Conformal Mapping + Histogramming

Triggers

Global methods are especially suitable for fast tracking in projections

**Example:** Collider experiment with a solenoid, where tracks are circular trajectories

Simple

Conformal Mapping:

Transform circles into straight lines

$u = x/(x^2+y^2)$

$v = -y/(x^2+y^2)$

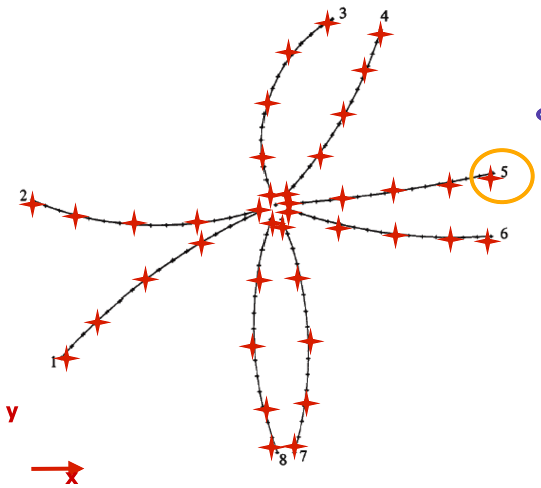
Histogram:

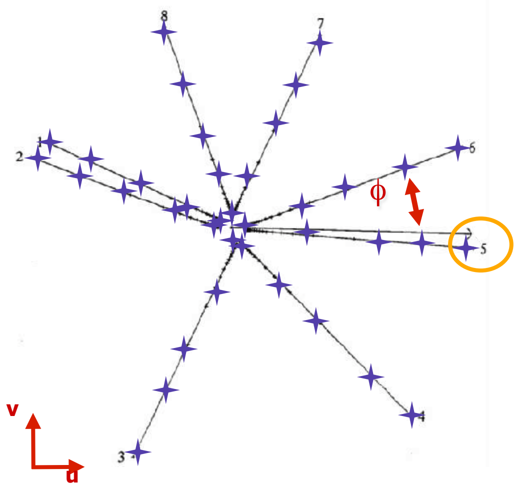
Collect a histogram of azimuth angles $\phi$

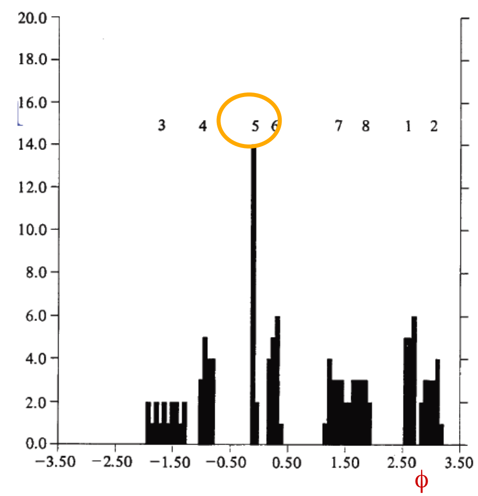
Find peaks in the histogram

Collect hits into tracks

Fast



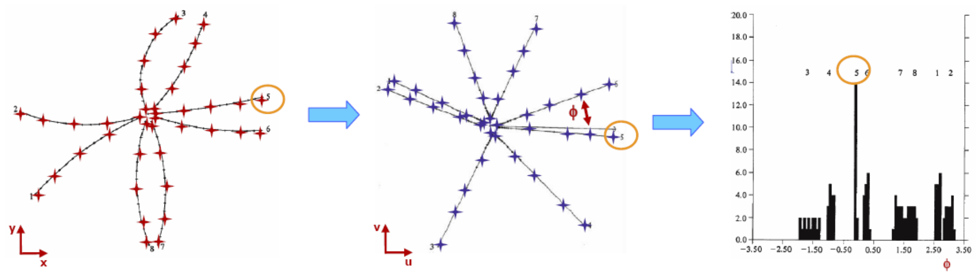



# Global Methods: Conformal Mapping + Histogramming

**Conformal Mapping:**
Transform circles into straight lines
$u = x/(x^2+y^2)$
$v = -y/(x^2+y^2)$

**Histogram:**
Collect a histogram of azimuth angles $\phi$
Find peaks in the histogram
Collect hits into tracks



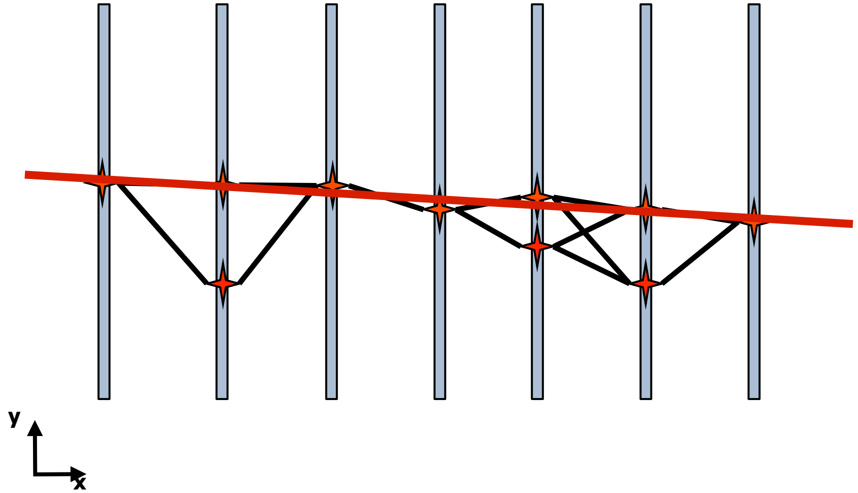
**Advantages:**
- Impressive visual simplification of the problem
- Each step is easy to implement in hardware
- This results in a fast algorithm

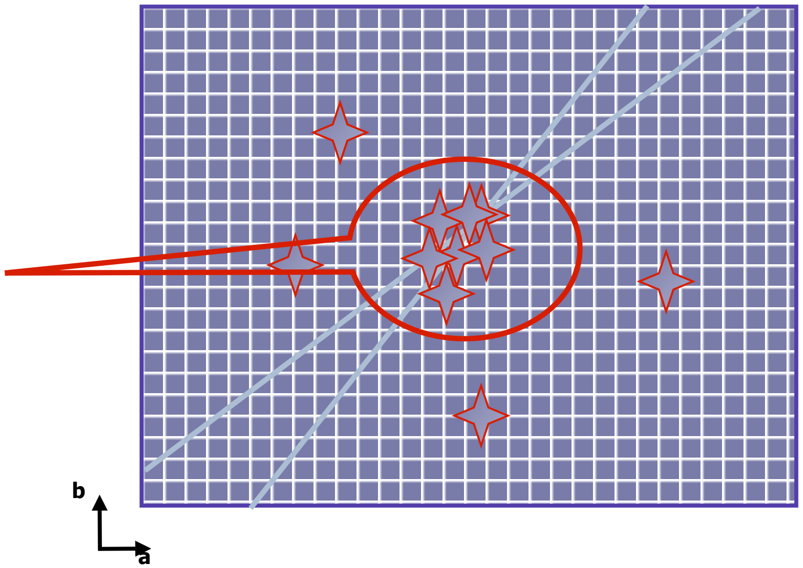
**Disadvantages:**
- Non-obvious complications of the problem
- Reverse order of the hits (last <-> first)
- Measurement errors are now no more uniform
- Geometry of detectors must be transformed
- Geometry of material walls must also be transformed
- What with the alignment constants ?
- A (non-uniform) magnetic field (map) must be transformed
- What with the Lorentz force: $\mathbf{F} = q(\mathbf{E}+\mathbf{v}\times\mathbf{B})$ ?
- Needs to know exact position of the interaction point
- Finds only primary tracks
- Does not find secondary tracks
- Is it possible to build a trigger on primary tracks only ?
- In fact, histogramming provides only track parameters
- No errors of track parameters estimates (covariance matrix)
- No hits grouping into track candidates
- Therefore, no possibility to refit tracks
- Histogramming needs access to main memory (slow)

**Conclusion:** Useful implemented in hardware and for very simple event topologies only

# Global Methods: Hough Transformation

| Measurement Space | Parameter Space |
| --- | --- |
| $y = a*x + b$ | $b = -x*a + y$ |



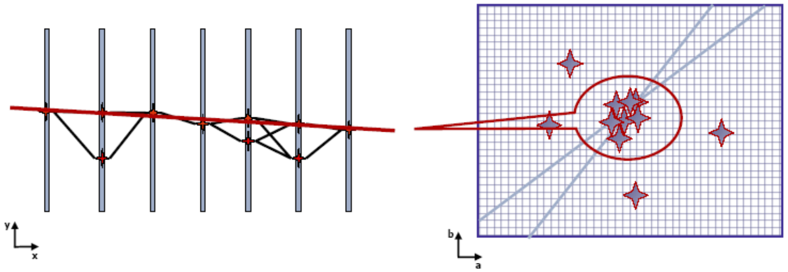
# Global Methods: Hough Transformation

Measurement Space

$y = a*x + b$

Parameter Space

$b = -x*a + y$



**Advantages:**

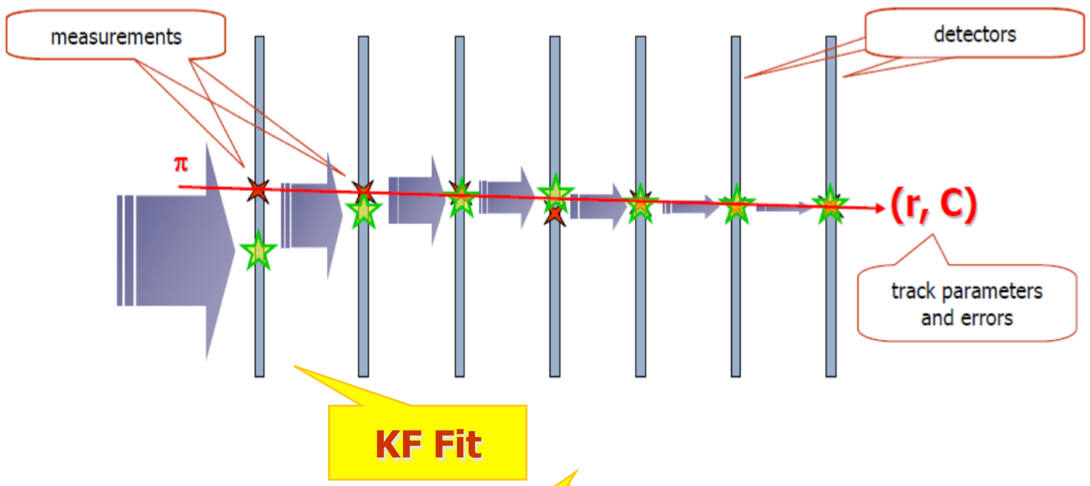
- Generalization of the histogramming method
- Easy to implement in hardware
- This results in a fast algorithm

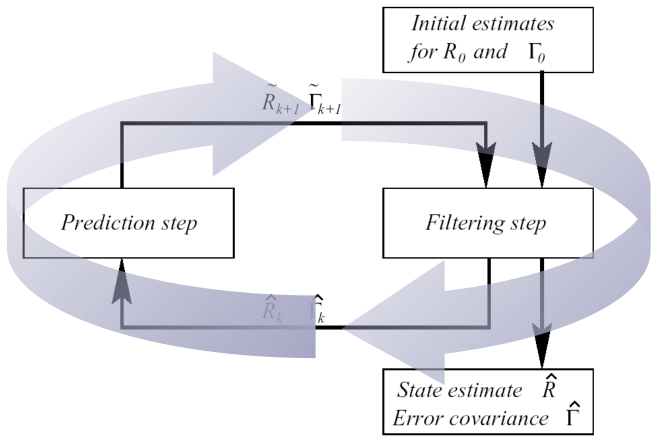
**Disadvantages:**

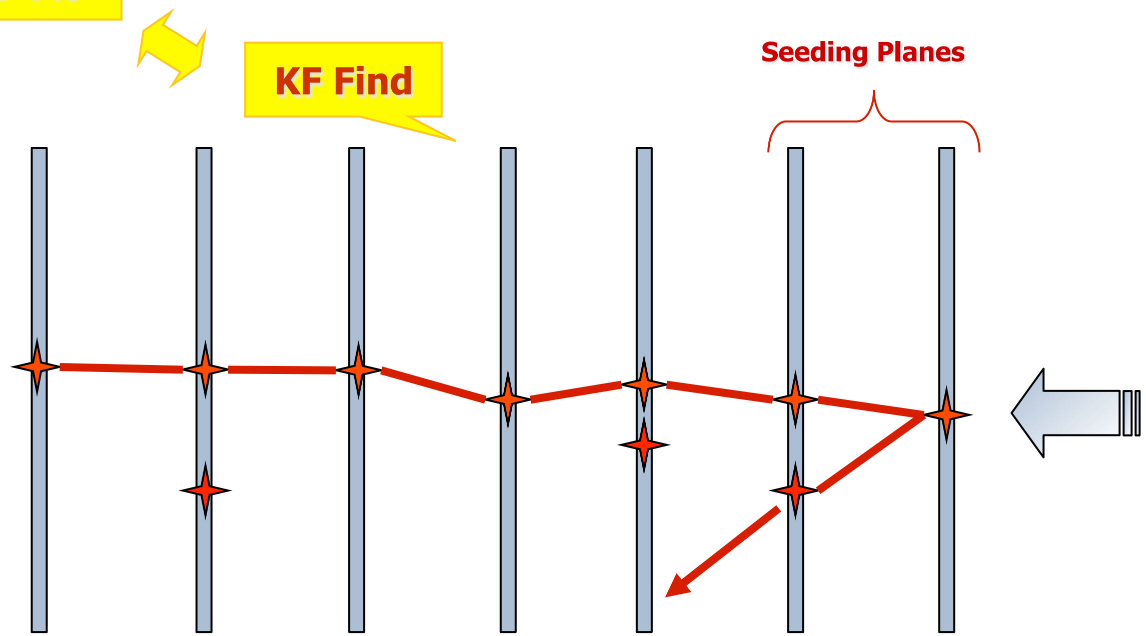
- Needs a global track model
- Therefore, appropriate for simple magnetic fields only
- Does not include multiple scattering
- Finds only fast tracks (not MF and MS dependent)
- Histogramming provides only track parameters
- No errors of track parameters estimates (covariance matrix)
- No hits grouping into track candidates
- Therefore, no possibility to refit tracks
- Not possible competition between track candidates
- Therefore, relatively large percentage of wrong tracks
- Histogramming needs access to main memory -> slow
- 5D histogramming (x, y, tx, ty, q/p) needs a lot of memory
- Precise tracking requires even more memory -> swapping ?
- Memory initialization for every event

**Conclusion:** Useful implemented in hardware and for simple event and trigger topologies

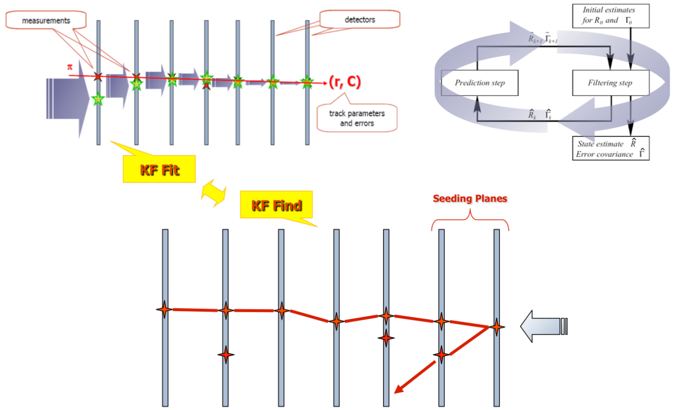
# Local Methods: Kalman Filter for Track Finding



# Local Methods: Kalman Filter for Track Finding




**Advantages:**

- Psychologically easy to accept hit by hit track finding
- Combined track finder and fitter based on KF
- Development of a new experiment starts with an ideal MC track finder and a realistic KF track fitter, therefore the next step to a realistic track finder is obvious – KF
- ...

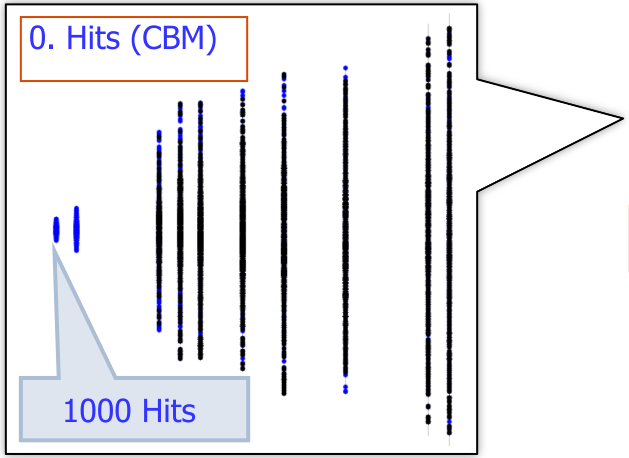
**Disadvantages:**

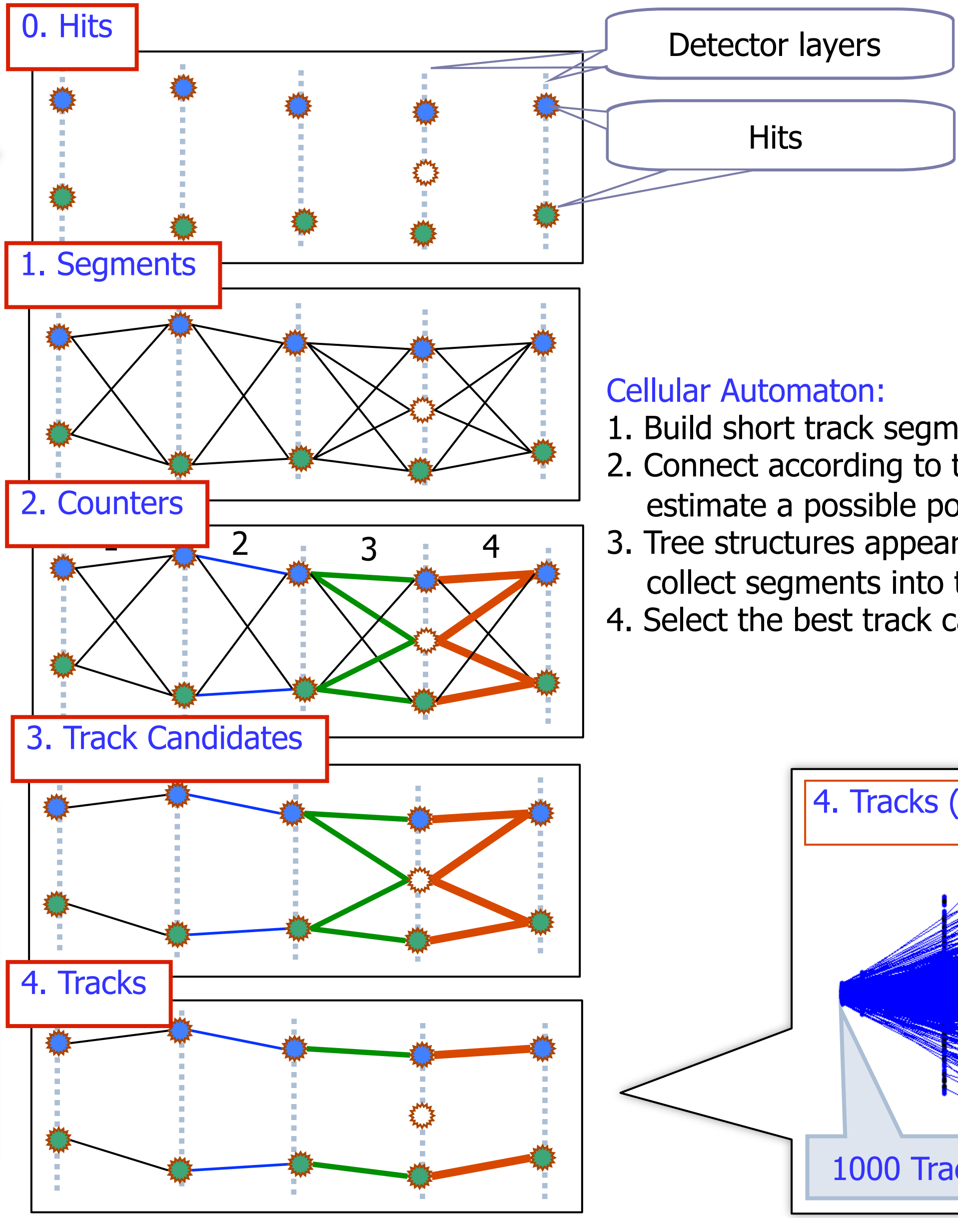
- Track finding – a combinatorial (NP) problem, can not be solved directly using methods suitable for single track
- Repeats the same calculations many times, when discarding track candidates
- Works at the hit level
- Needs seeding (starting short track segments)
- Final efficiency is always limited by seeding efficiency
- It is limited also by the efficiency of the seeding chambers
- Therefore needs a lot of seeds -> even larger combinatorics
- How many inefficient detectors can be tolerated in general ?
- How to include missing hits into the Kalman filter ?
- How to calculate chi^2 in this case ?
- Too early competition between track candidates
- ---

**Conclusion:** Useful for relatively simple event topologies and as initial (second after the ideal) track finder

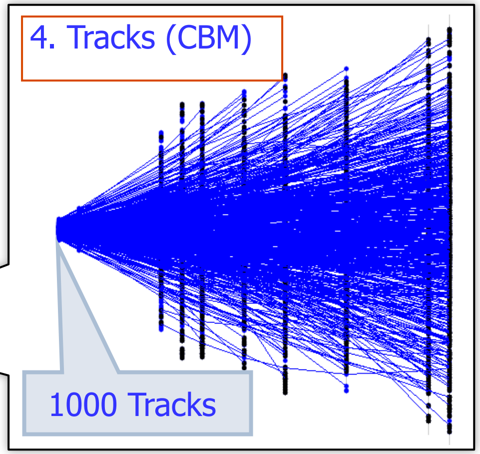
# Cellular Automaton (CA) as Track Finder

Track finding: Which hits in detector belong to the same track? – Cellular Automaton (CA)



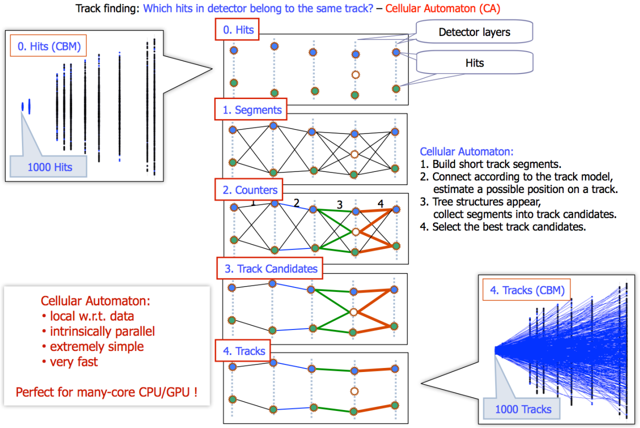

Cellular Automaton:
1. Build short track segments.
2. Connect according to the track model, estimate a possible position on a track.
3. Tree structures appear, collect segments into track candidates.
4. Select the best track candidates.




Cellular Automaton:
- local w.r.t. data
- intrinsically parallel
- extremely simple
- very fast

Perfect for many-core CPU/GPU !

# Cellular Automaton (CA) as Track Finder



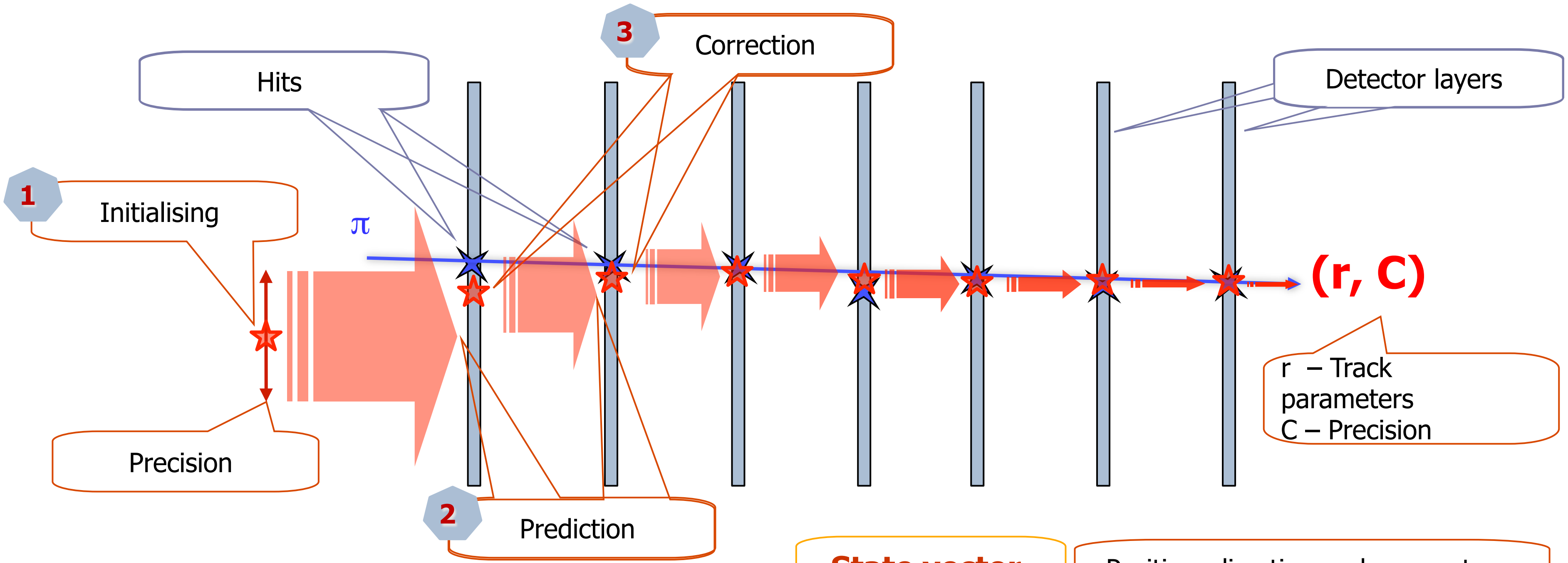

**Advantages:**
- Local relations -> simple calculations
- Local relations -> parallel algorithm
- Staged implementation: hits -> segments -> tracks
- Polynomial (2nd order?) combinatorics
- Track competition at the global level
- Includes the KF fitter, if necessary, for high track densities
- Detector inefficiency problem outside the combinatorics
- ...

**Disadvantages:**
- Not easy to understand a parallel algorithm (Game of Life)
- Currently implementations on sequential computers
- Parallel hardware is coming now
- ...

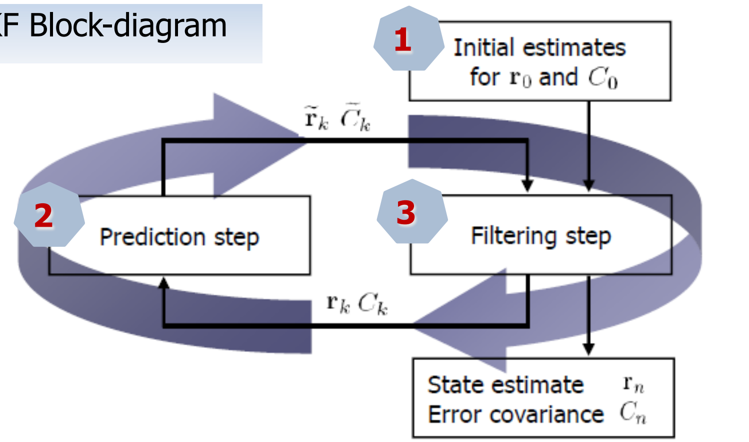
**Conclusion:** Useful for complicated event topologies with large combinatorics and for parallel hardware

# Kalman Filter (KF) based Track Fit

Track fit: Estimation of the track parameters at one or more hits along the track – Kalman Filter (KF)




KF Block-diagram




KF as a recursive least squares method

**State vector** — Position, direction and momentum

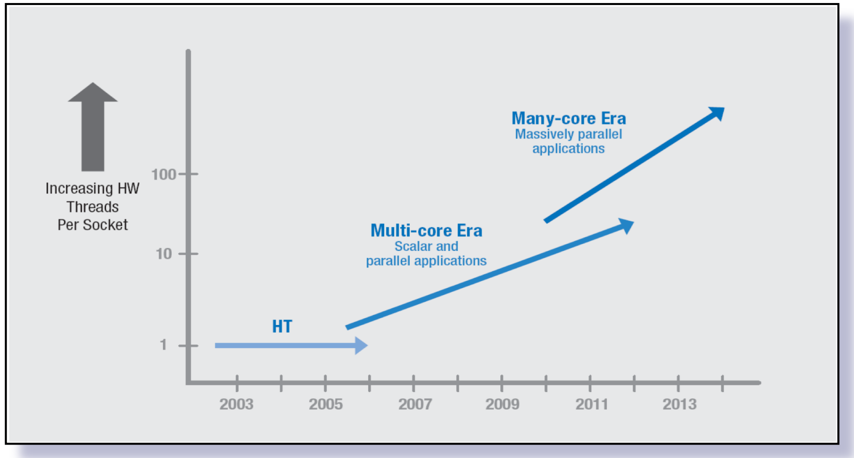
$$\mathbf{r} = \{ x, y, z, p_x, p_y, p_z \}$$

Kalman Filter:

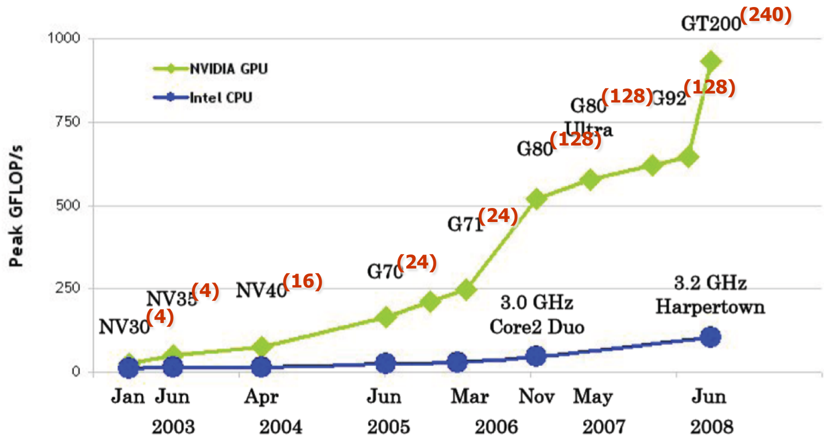
1. Start with an arbitrary initialization.
2. Add one hit after another.
3. Improve the state vector.
4. Get the optimal parameters after the last hit.

Nowadays the Kalman Filter is used in almost all HEP experiments

# Coming now: Many-core Era of HPC

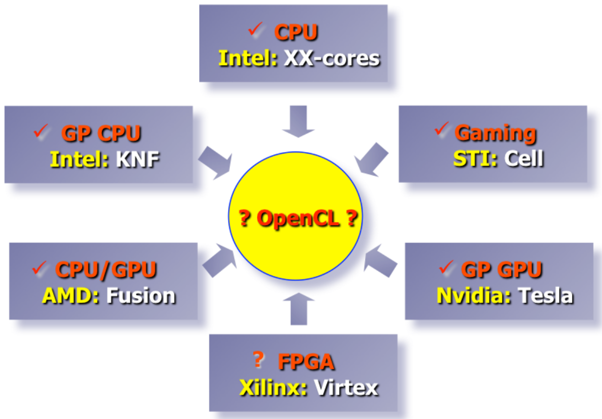


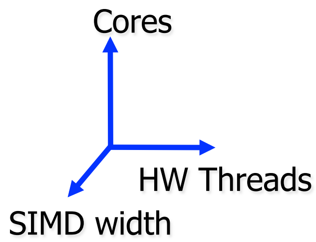

S. Borkar et al. (Intel), "Platform 2015: Intel Platform Evolution for the Next Decade", 2005.




| | | |
|---|---|---|
| GT200 = GeForce GTX 280 | G71 = GeForce 7900 GTX | NV35 = GeForce FX 5950 Ultra |
| G92 = GeForce 9800 GTX | G70 = GeForce 7800 GTX | NV30 = GeForce FX 5800 |
| G80 = GeForce 8800 GTX | NV40 = GeForce 6800 Ultra | |

Source: NVIDIA



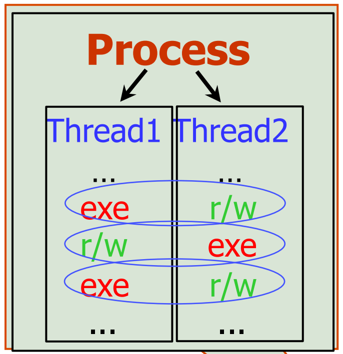

- On-line event selection
- Mathematical and computational optimization
- Optimization of the detector

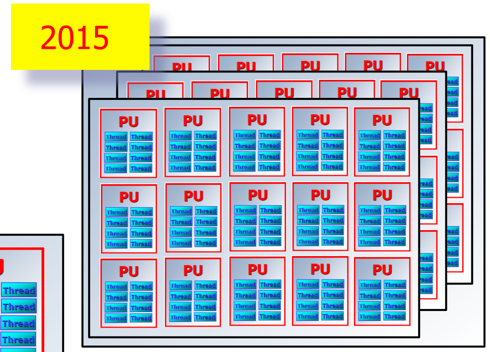
- Heterogeneous systems of many cores
- Uniform approach to all CPU/GPU families
- Similar programming languages (CUDA, ArBB, OpenCL)
- Parallelization of the algorithm (vectors, multi-threads, many-cores)

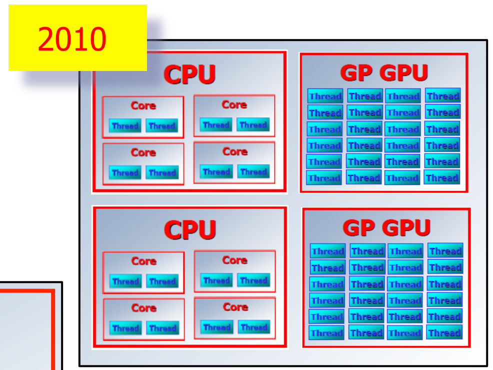
# Many-Core HPC: Cores, Threads and SIMD

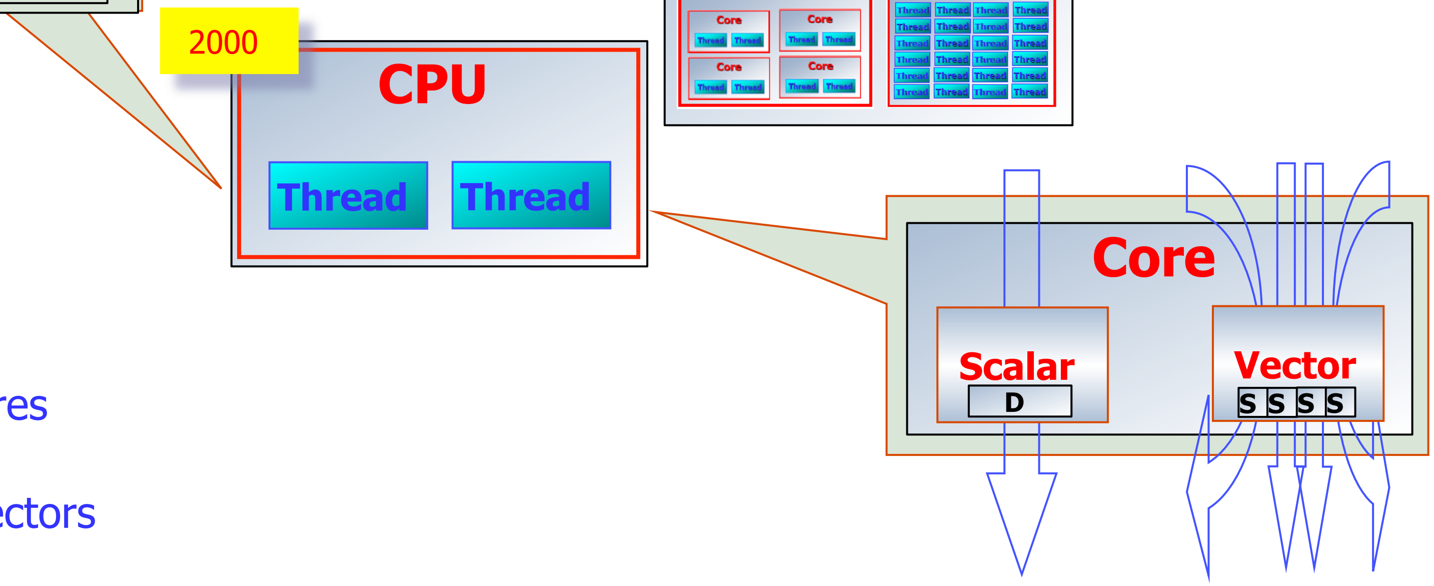
HEP experiments work with high data rates, therefore need High Performance Computing (HPC) !

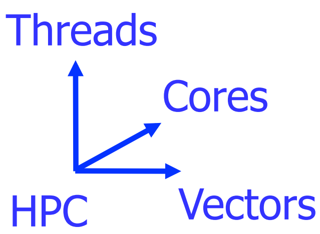
Cores and Threads realize the task level of parallelism




Vectors (SIMD) = data level of parallelism

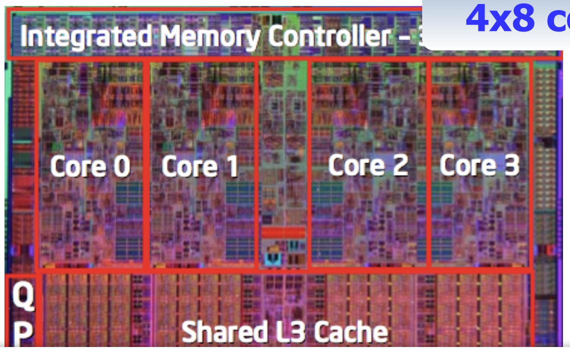
SIMD = Single Instruction, Multiple Data

Fundamental redesign of traditional approaches to data processing is necessary

# Many-Core CPU/GPU Architectures

## Intel/AMD CPU

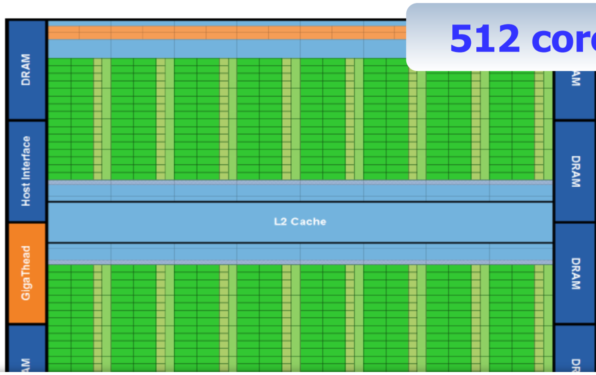
4x8 cores




- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution

## NVIDIA/AMD GPU

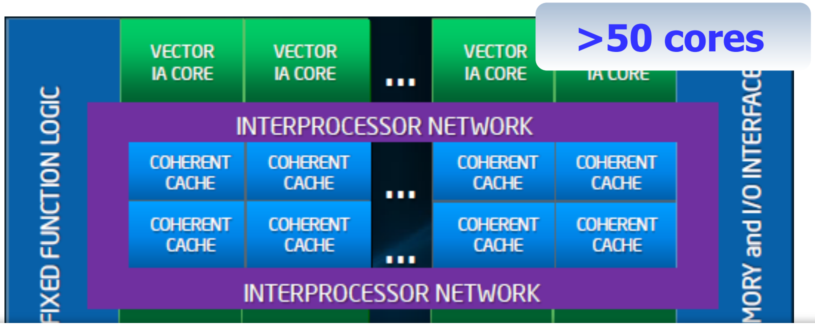
512 cores




- Optimized for data-parallel, throughput computation
- More transistors dedicated to computation

## Intel MIC

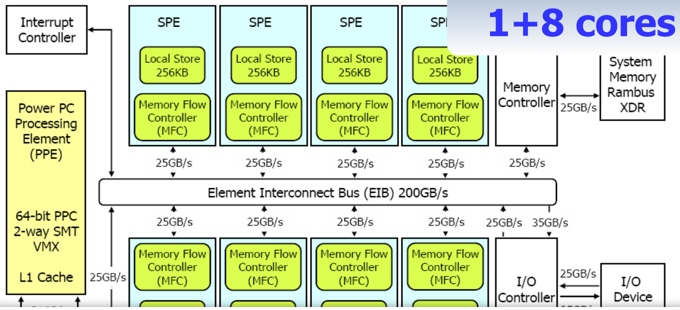
>50 cores




- Many Integrated Cores architecture announced at ISC10 (June 2010)
- Based on the x86 architecture
- Many-cores + 4-way multithreaded + 512-bit wide vector unit

## IBM Cell

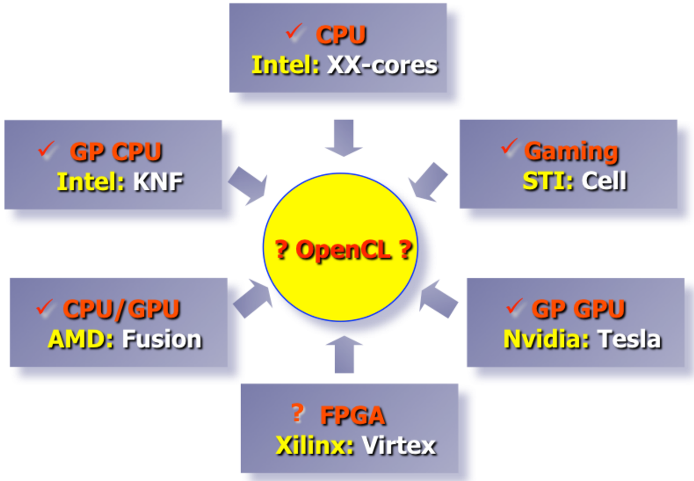
1+8 cores




- General purpose RISC processor (PowerPC)
- 8 co-processors (SPE, Synergistic Processor Elements)
- 128-bit wide SIMD units

Future systems are heterogeneous

# CPU/GPU Programming Frameworks




- Intel Ct (C for throughput), ArBB (Array Building Blocks)
- Extension to the C language
- Intel CPU/GPU specific
- SIMD exploitation for automatic parallelism

- NVIDIA CUDA (Compute Unified Device Architecture)
- Defines hardware platform
- Generic programming
- Extension to the C language
- Explicit memory management
- Programming on thread level

- OpenCL (Open Computing Language)
- Open standard for generic programming
- Extension to the C language
- Supposed to work on any hardware
- Usage of specific hardware capabilities by extensions

- Vector classes (Vc)
- Overload of C operators with SIMD/SIMT instructions
- Uniform approach to all CPU/GPU families
- Uni-Frankfurt/FIAS/GSI

ArBB, Vector classes: Cooperation with Intel

# Kalman Filter Track Fit on Cell

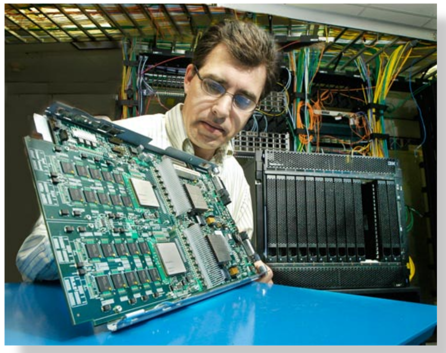
| Stage | Description | Time/track | Speedup |
|---|---|---|---|
| | Initial scalar version | 12 ms | – |
| 1 | Approximation of the magnetic field | 240 $\mu s$ | 50 |
| 2 | Optimization of the algorithm | 7.2 $\mu s$ | 35 |
| 3 | Vectorization | 1.6 $\mu s$ | 4.5 |
| 4 | Porting to SPE | 1.1 $\mu s$ | 1.5 |
| 5 | Parallelization on 16 SPEs | 0.1 $\mu s$ | 10 |
| | Final simdized version | 0.1 $\mu s$ | $\boxed{120000}$ |

Intel P4 (stages 1–3)

Cell (stages 4–5)

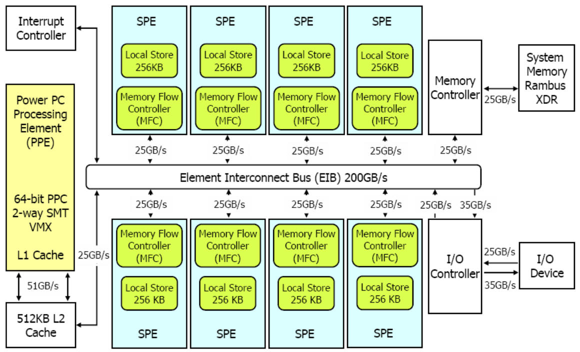
10000x faster on any PC

Comp. Phys. Comm. 178 (2008) 374-383



blade11bc4 @IBM, Böblingen:
2 Cell Broadband Engines, 256 kB LS, 2.4 GHz

The KF speed was increased by 5 orders of magnitude

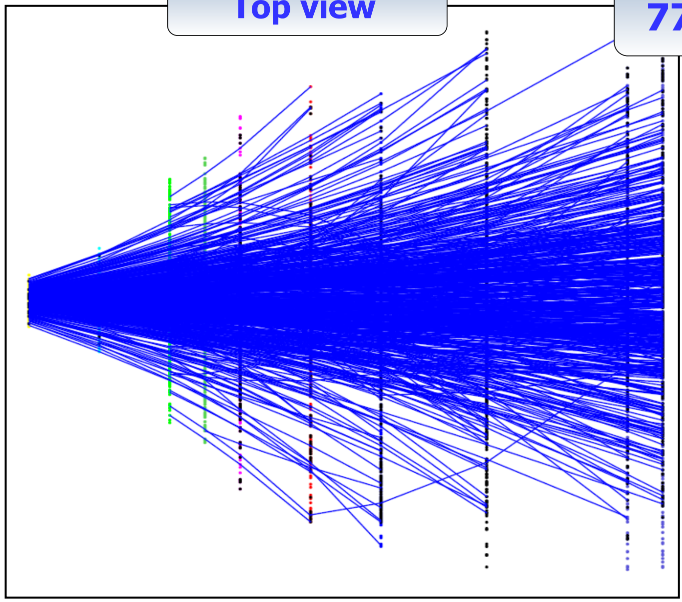



Motivated by, but not restricted to Cell !

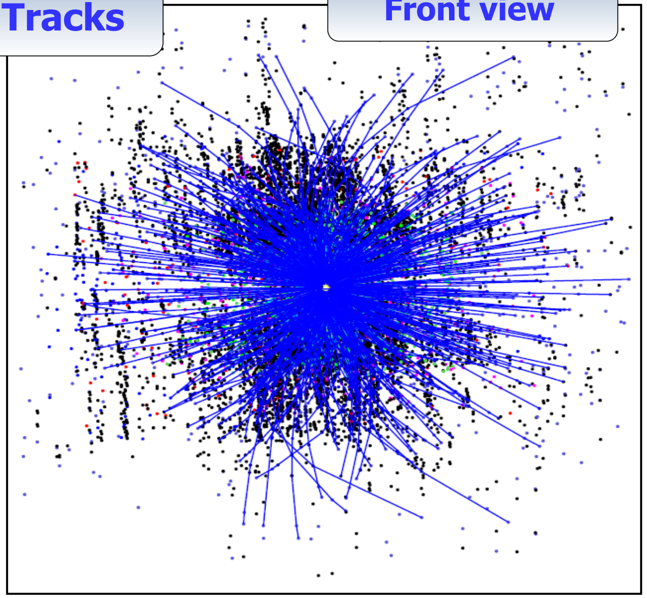
# CBM Cellular Automaton Track Finder

## 770 Tracks

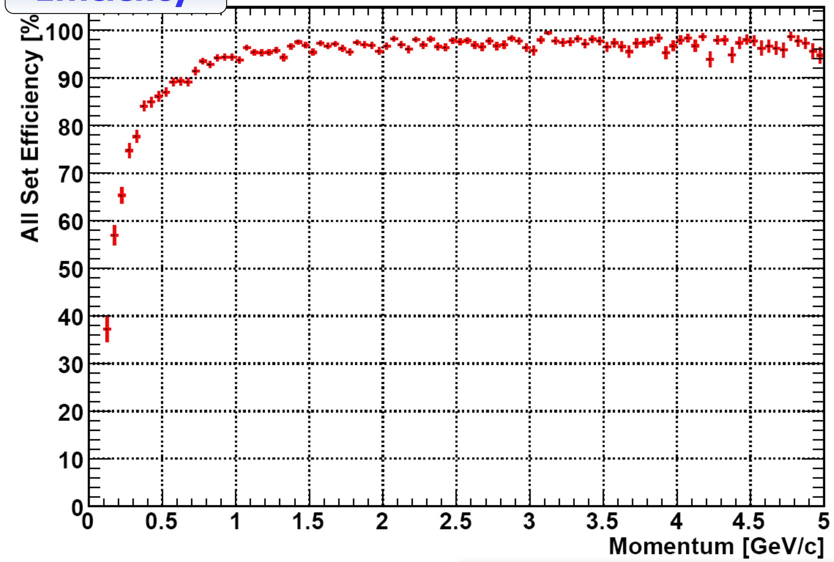
Top view



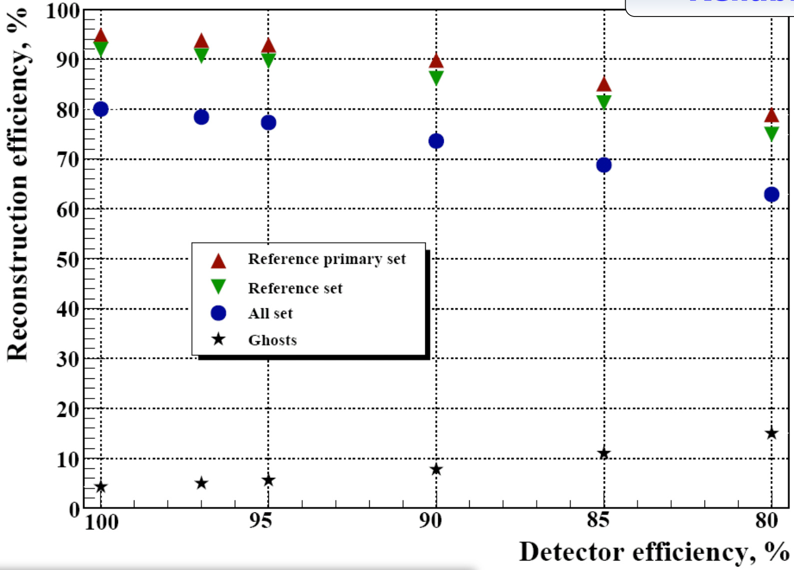
Front view



Efficiency



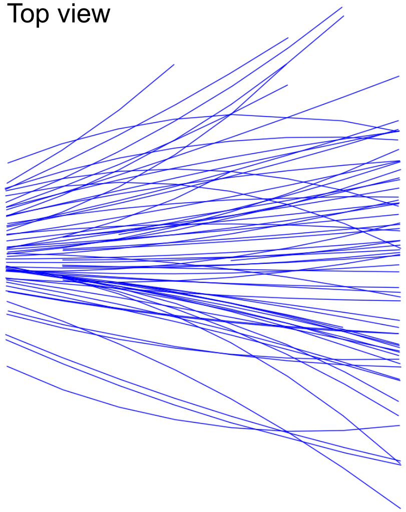

Reliability



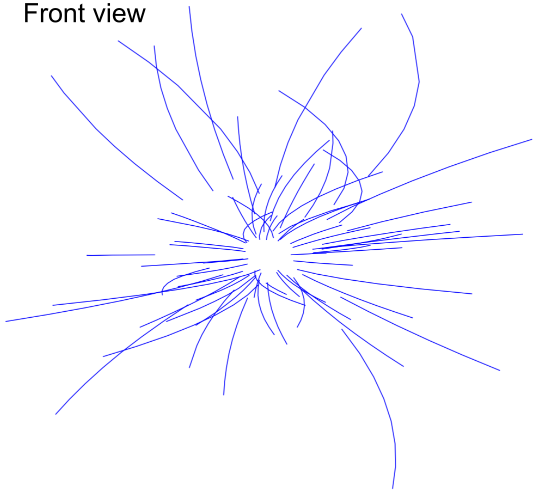

Highly efficient and reliable event reconstruction

# Track Finding at Low Track Multiplicity

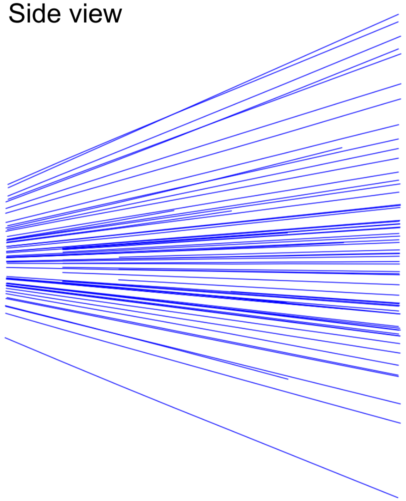
Top view



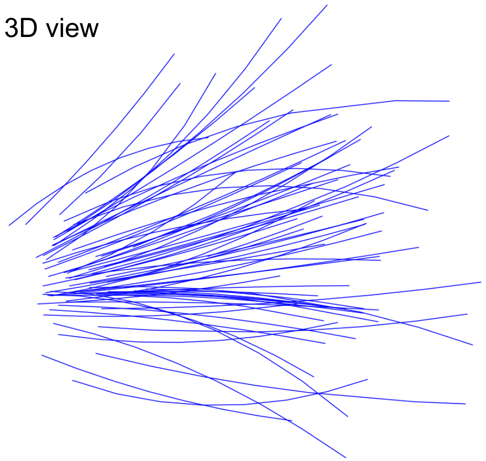
Front view



Side view



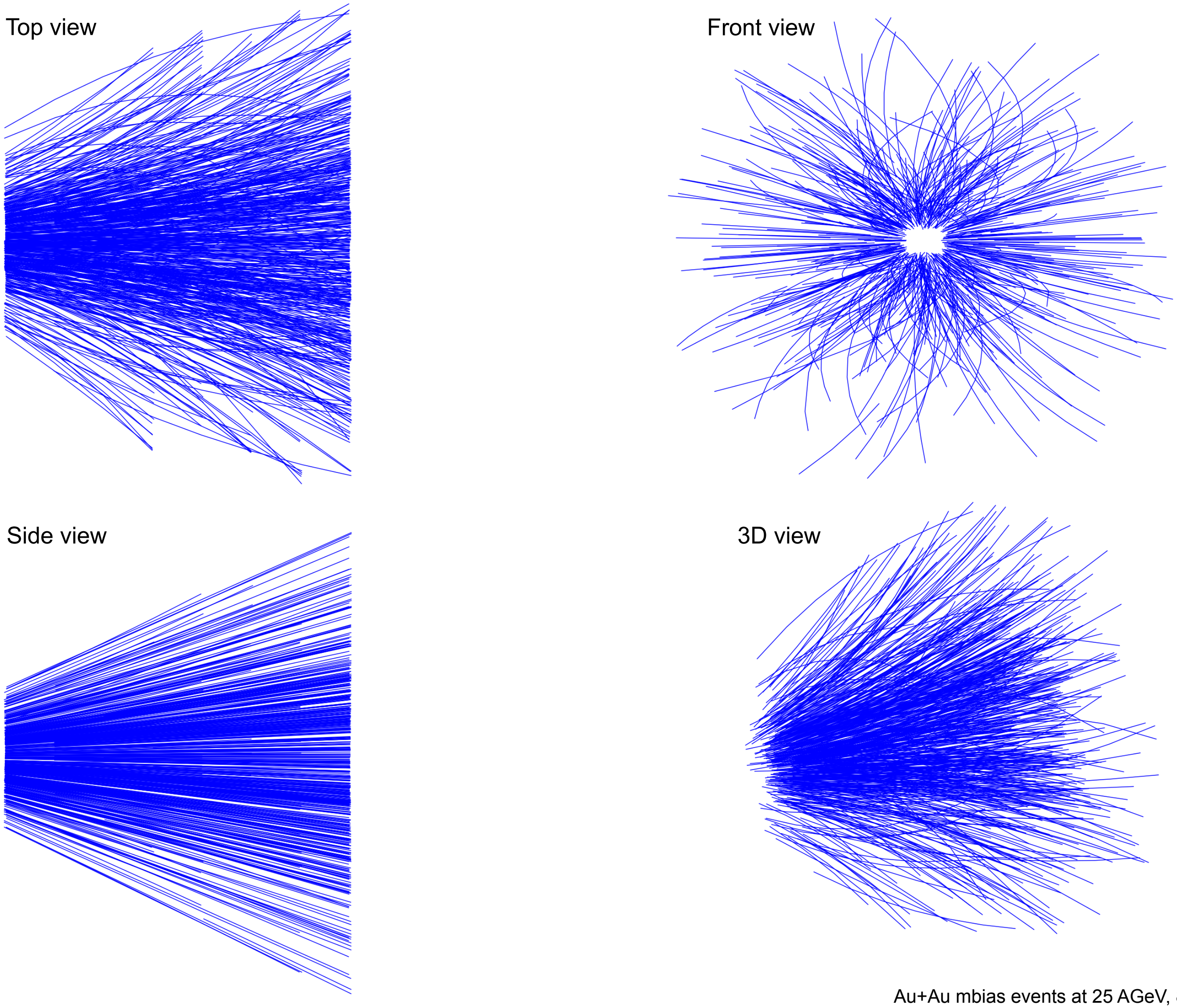
3D view



Au+Au mbias events at 25 AGeV, 8 STS, 0 x 7,5 strip angles

A minimum bias event: average reconstructed track multiplicity 109

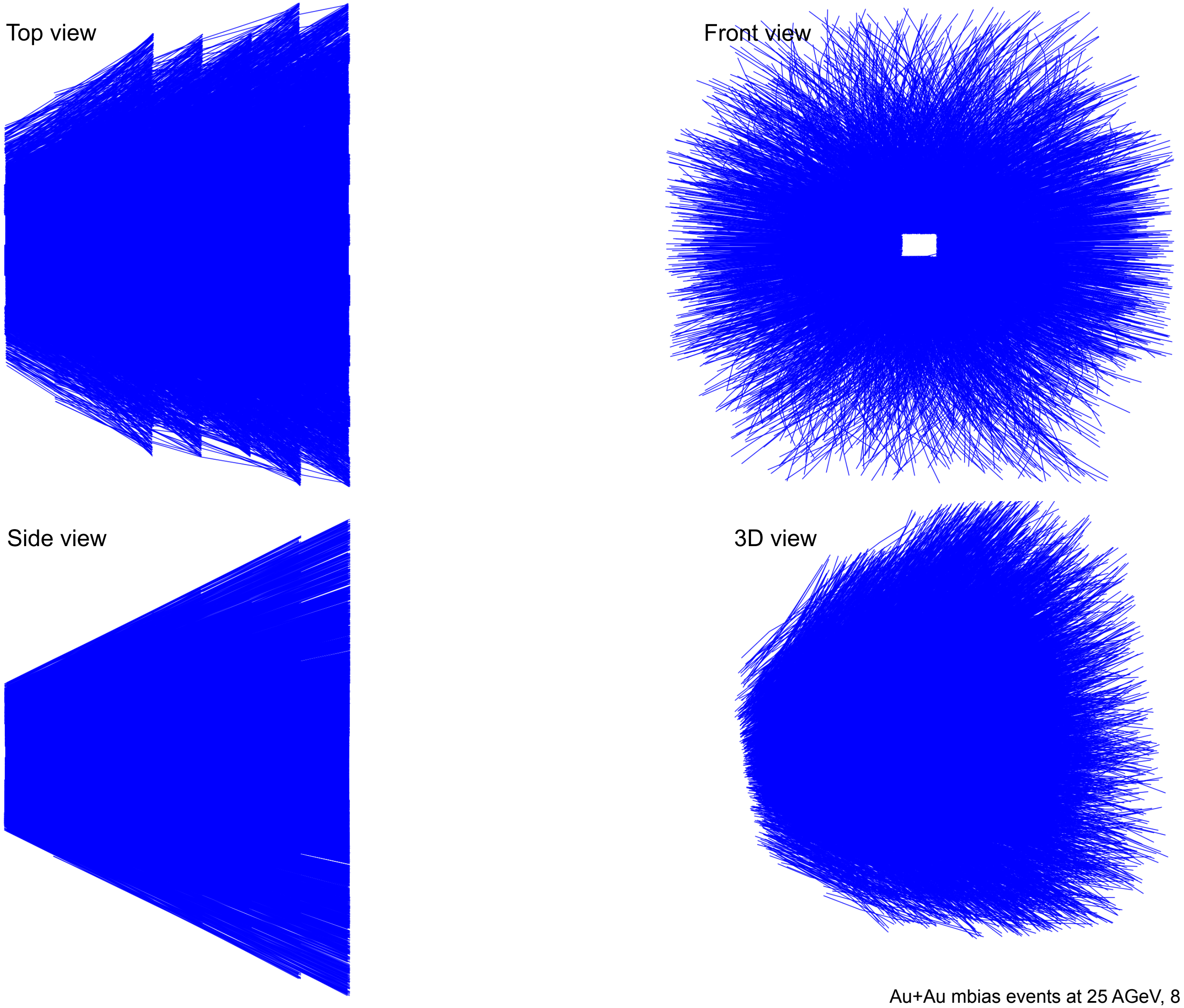
# Track Finding at Medium Track Multiplicity




Au+Au mbias events at 25 AGeV, 8 STS, 0 x 7,5 strip angles

A central event: average reconstructed track multiplicity 572

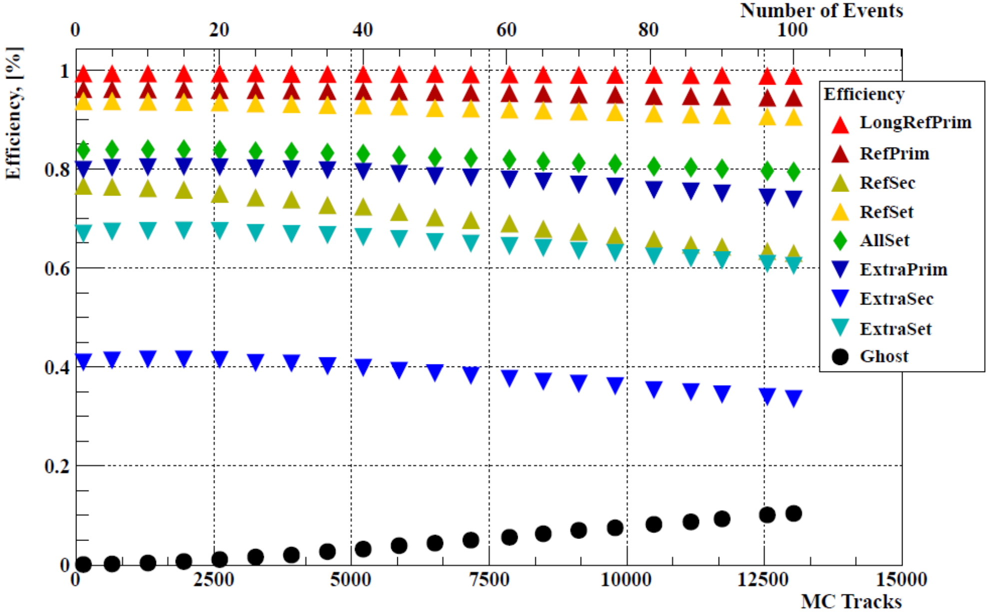
# Track Finding at High Track Multiplicity

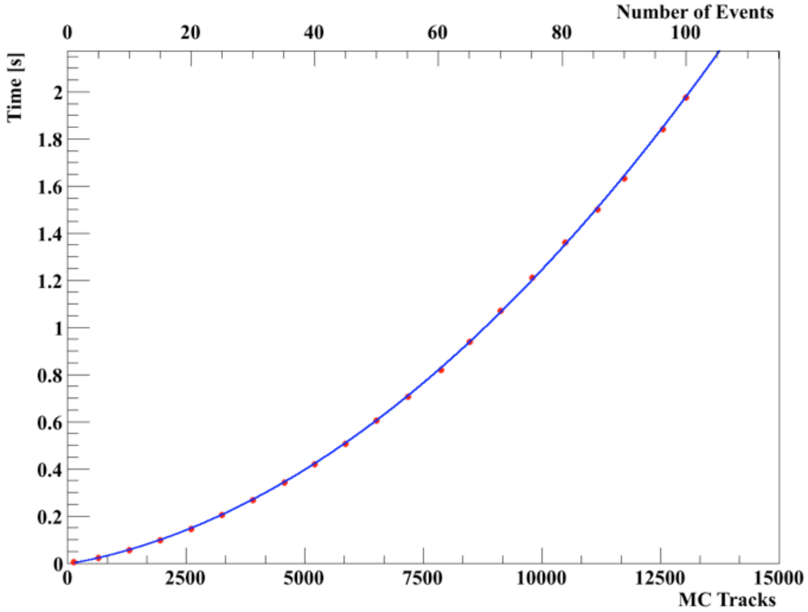



Au+Au mbias events at 25 AGeV, 8 STS, 0 x 7,5 strip angles

A group with 100 minimum bias events: average reconstructed track multiplicity 10340

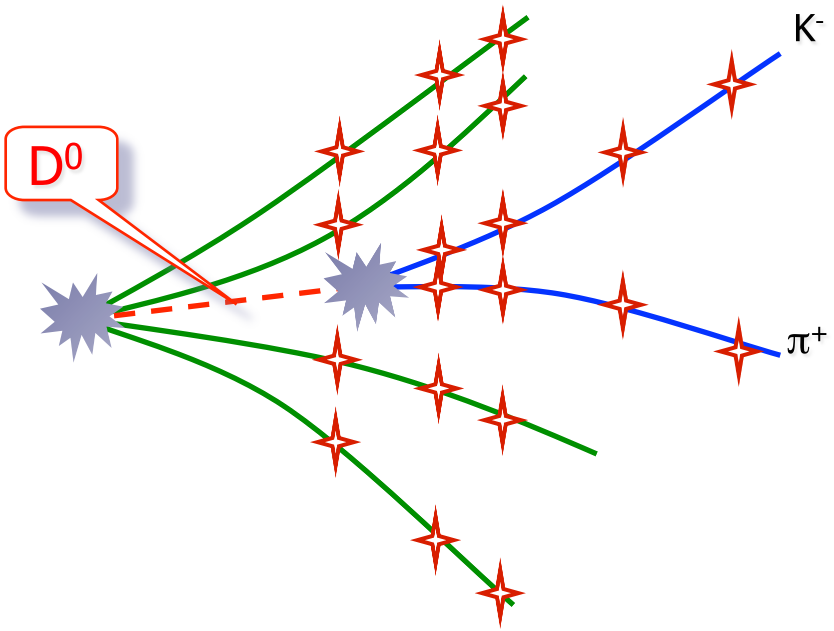
# CA Track Finder: Efficiency and Time vs. Track Multiplicity




Stable reconstruction efficiency and time as a second order polynomial up to 100 minimum bias events in a group

# KFParticle: Reconstruction of Vertices and Decayed Particles




**State vector** — Position, direction, momentum and energy

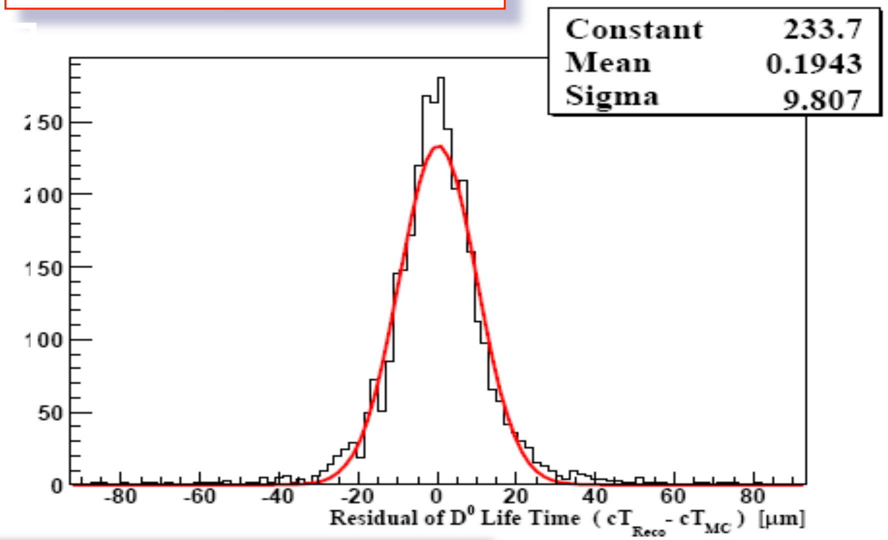
$$\mathbf{r} = \{\, x, y, z, p_x, p_y, p_z, E \,\}$$

$x, y, z, p_x, p_y, p_z, E, m, L, c\tau$

```
AliKFVertex PrimVtx( ESDPrimVtx );  // Set primary vertex
                                    // Set daughters
AliKFParticle K( ESDp1, -321 ), pi( ESDp2, 211 );

AliKFParticle  D0( K, pi );         // Construct mother
PrimVtx += D0;                      // Improve the primary vertex

D0.SetProductionVertex( PrimVtx );  // D0 is fully fitted
K.SetProductionVertex( D0 );        // K is fully fitted
pi.SetProductionVertex( D0 );       // pi is fully fitted
```




KFParticle provides uncomplicated approach to physics analysis

# KFParticle Finder for Physics Analysis and Selection

Tracks: $e^{\pm}$, $\mu^{\pm}$, $\pi^{\pm}$, $K^{\pm}$, $p^{\pm}$
secondary and primary

**Open-charm:**

$D^0 \to \pi^+ K^-$
$D^0 \to \pi^+ \pi^+ \pi^- K^-$
$\bar{D}^0 \to \pi^- K^+$
$\bar{D}^0 \to \pi^- \pi^- \pi^+ K^+$
$D^+ \to \pi^+ \pi^+ K^-$
$D^- \to \pi^- \pi^- K^+$
$D_s^+ \to \pi^+ K^+ K^-$
$D_s^- \to \pi^- K^+ K^-$
$\Lambda_c \to \pi^+ K^- p$

**Open-charm resonances:**

$D^{*0} \to D^+ \pi^-$
$\bar{D}^{*0} \to D^- \pi^+$
$D^{*+} \to D^0 \pi^+$
$D^{*-} \to \bar{D}^0 \pi^-$

**Strange particles:**

$K^0_s \to \pi^+ \pi^-$
$\Lambda \to p\, \pi^-$
$\bar{\Lambda} \to \pi^+ p^-$

**Multi-strange hyperons:**

$\Xi^- \to \Lambda\, \pi^-$
$\Xi^+ \to \bar{\Lambda}\, \pi^+$
$\Omega^- \to \Lambda\, K^-$
$\Omega^+ \to \bar{\Lambda}\, K^+$

**Multi-strange resonances:**

$\Xi^{*0} \to \Xi^- \pi^+$
$\bar{\Xi}^{*0} \to \Xi^+ \pi^-$
$\Omega^{*-} \to \Xi^- \pi^+ K^-$
$\Omega^{*+} \to \Xi^+ \pi^- K^+$

**Strange and multi-strange resonances:**

$\Sigma^{*+} \to \Lambda\, \pi^+$
$\bar{\Sigma}^{*+} \to \bar{\Lambda}\, \pi^-$
$\Sigma^{*-} \to \Lambda\, \pi^-$
$\bar{\Sigma}^{*-} \to \bar{\Lambda}\, \pi^+$
$K^{*-} \to K^0_s\, \pi^-$
$K^{*+} \to K^0_s\, \pi^+$
$\Xi^{*-} \to \Lambda\, K^-$
$\Xi^{*+} \to \bar{\Lambda}\, K^+$

**Gamma:**

$\gamma \to e^- e^+$

**Strange resonances:**

$\bar{K}^{*0} \to K^+ \pi^-$
$K^{*0} \to \pi^+ K^-$
$\bar{\Lambda}^* \to p\, K^-$
$\Lambda^* \to p^- K^+$

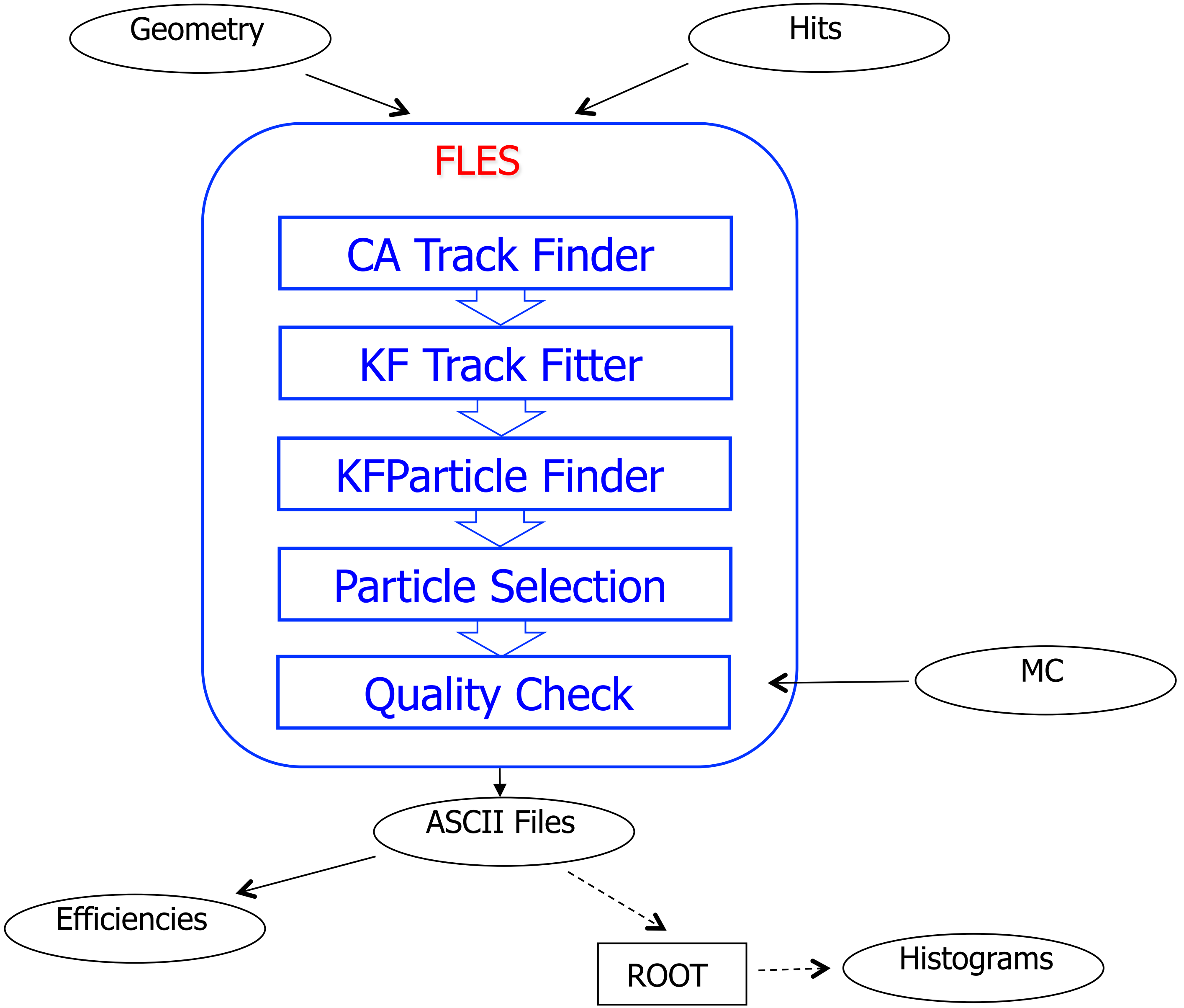
**Light vector mesons:**

$\rho \to e^- e^+$
$\rho \to \mu^- \mu^+$
$\omega \to e^- e^+$
$\omega \to \mu^- \mu^+$
$\varphi \to e^- e^+$
$\varphi \to \mu^- \mu^+$
$\varphi \to K^- K^+$

**Charmonium:**

$J/\Psi \to e^- e^+$
$J/\Psi \to \mu^- \mu^+$

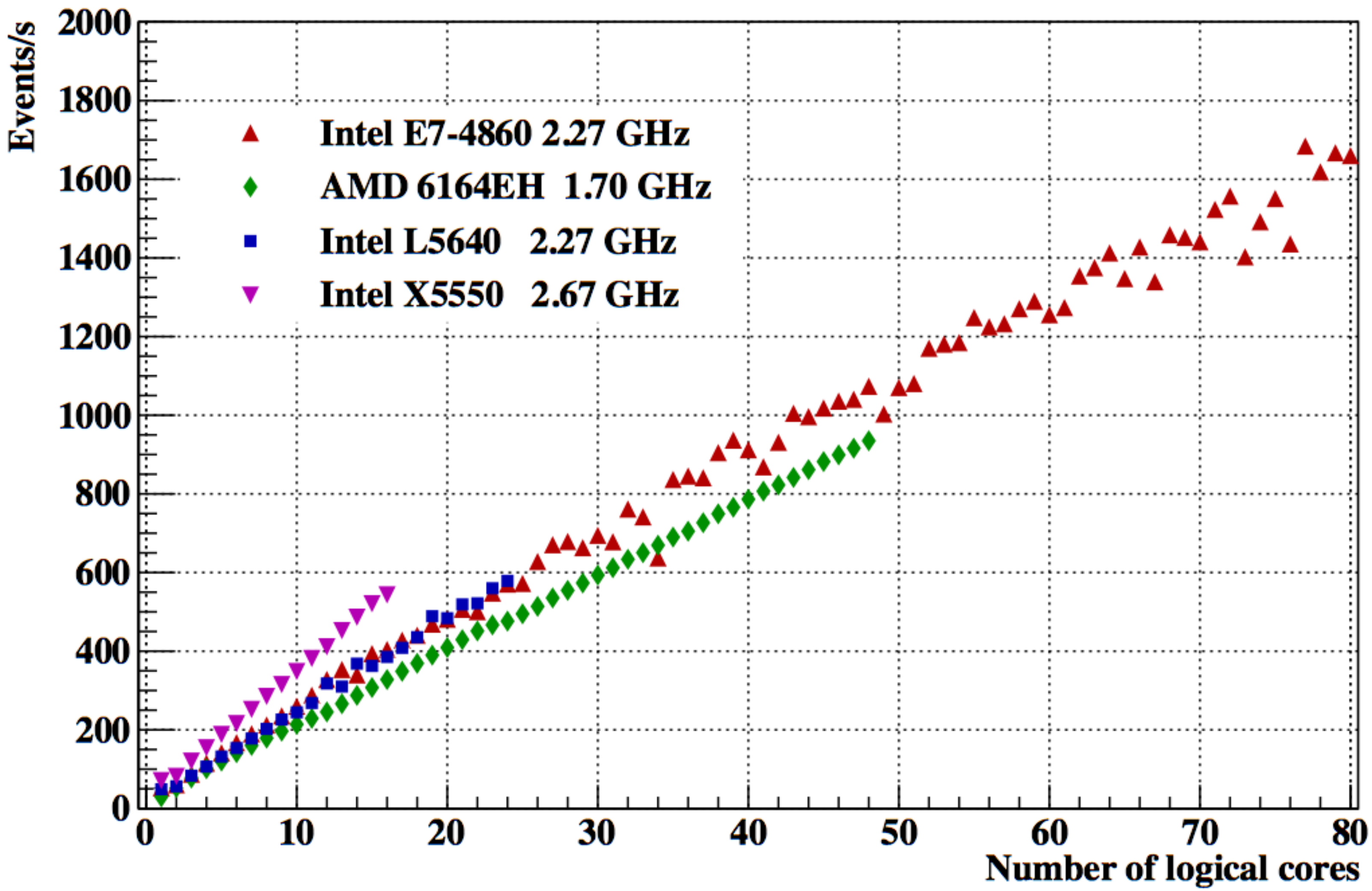
# Standalone First Level Event Selection (FLES) Package




The first version of the FLES package is portable, efficient, vectorized and parallelized

# FLES Package Scalability

Given n threads each filled with 1000 events, run them on specified n logical cores, 1 thread per 1 core.




The FLES package shows strong scalability on up to 80 cores

# Consolidate Efforts: Common Reconstruction Package

**Uni-Frankfurt/FIAS:**
Vector classes
CPU/GPU implementation

**GSI:**
Algorithms development
Many-core optimization

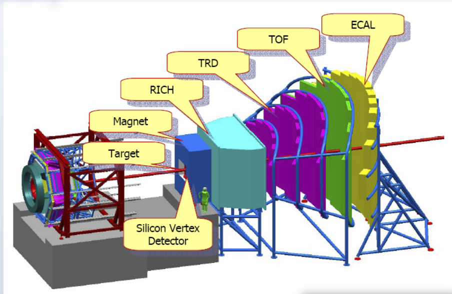
**OpenLab (CERN):**
Many-core optimization
Benchmarking

**HEPHY (Vienna)/Uni-Gjovik:**
Kalman Filter track fit
Kalman Filter vertex fit

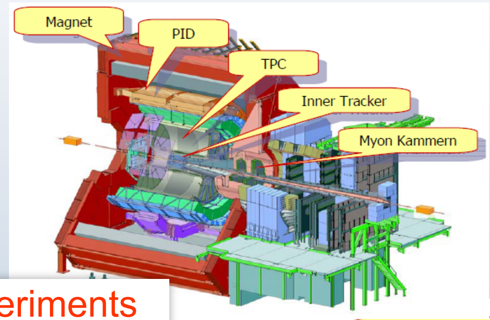
**Common Reconstruction Package**

**Intel:**
ArBB/OpenCL implementation
Many-core optimization
Benchmarking

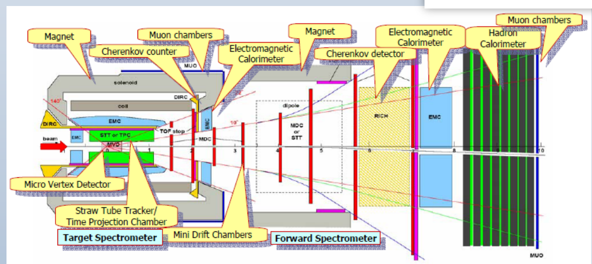
CBM (FAIR/GSI)



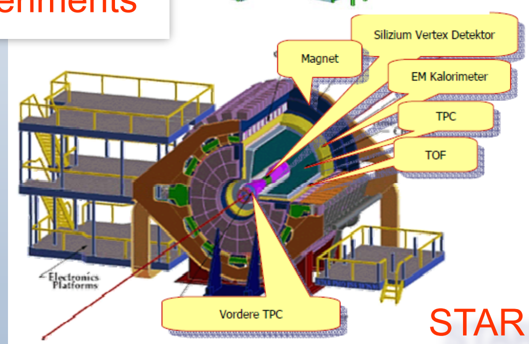

ALICE (CERN)




Host Experiments




PANDA (FAIR/GSI)




STAR (BNL)

# Consolidate Efforts: International Workshops

## International Workshop for Future Challenges in Tracking and Trigger Concepts

| | | |
|---|---|---|
| 1st | GSI, Darmstadt, Germany, | 07-11.06.2010; |
| 2nd | CERN, Geneva, Switzerland, | 07-08.07.2011; |
| 3rd | FIAS, Frankfurt, Germany, | 27-29.02.2012; |
| 4th | CERN, Geneva, Switzerland, | 28-30.11.2012. |