Cross-architecture Kalman Filer Investigation

Prof. Dr. Ivan Kisel

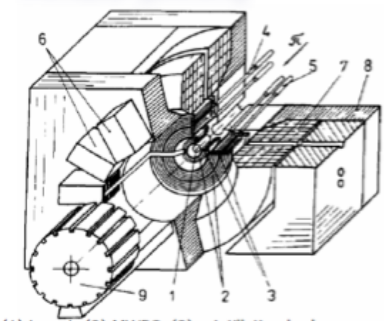
Goethe University Frankfurt am Main, Germany

FIAS Frankfurt Institute for Advanced Studies, Germany

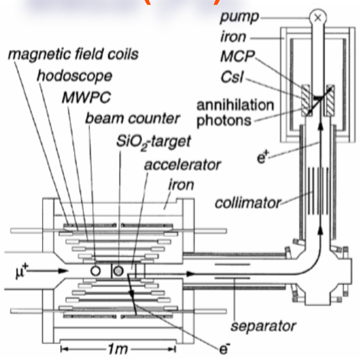
GSI Helmholtz Center for Heavy-Ion Research, Darmstadt, Germany

My research experience

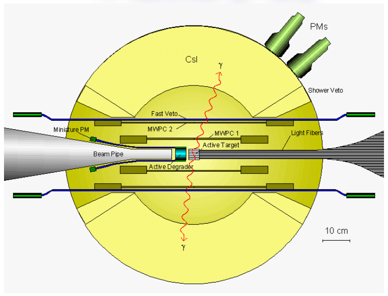
ARES (JINR)



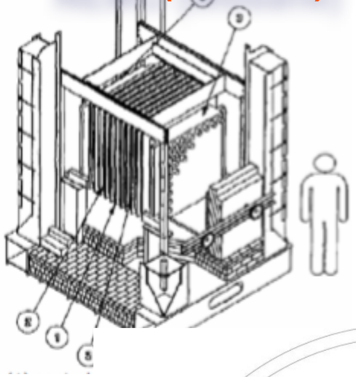
MMbar (PSI)



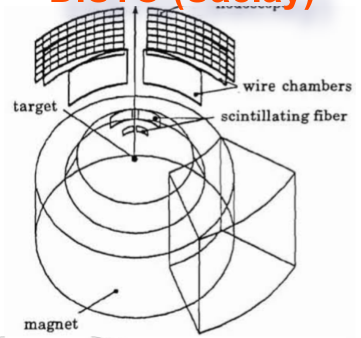
PiBeta (PSI)



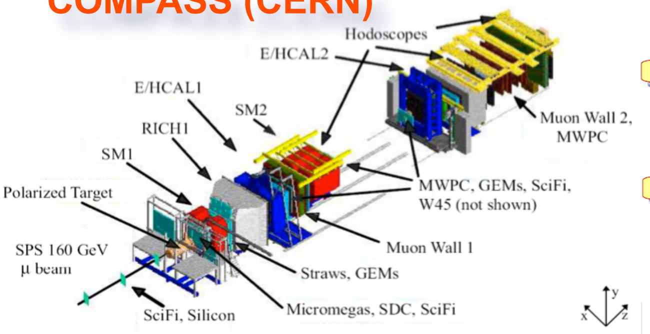
NEMO (Modane)



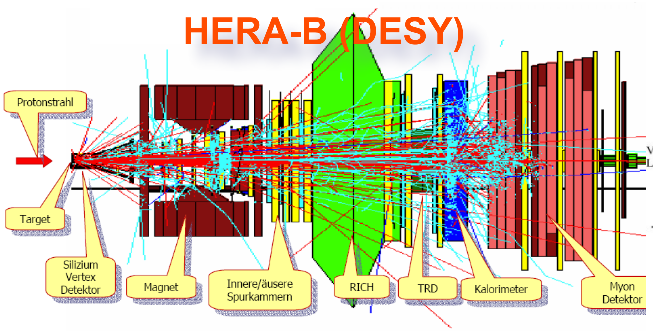
DISTO (Saclay)



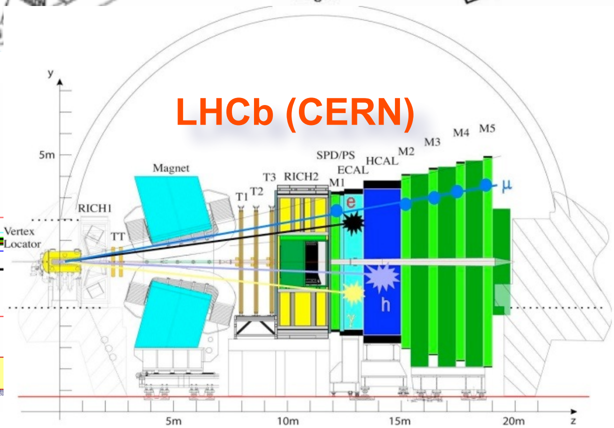
COMPASS (CERN)



HERA-B (DESY)

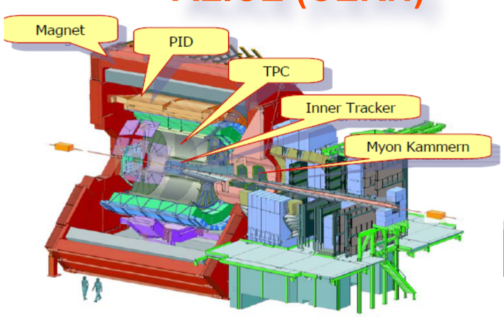


LHCb (CERN)

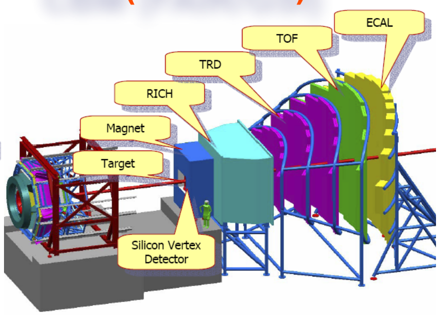


STAR (BNL)

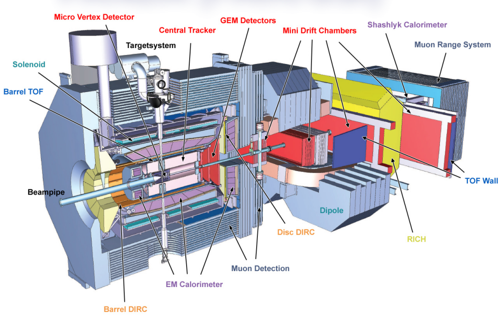
ALICE (CERN)



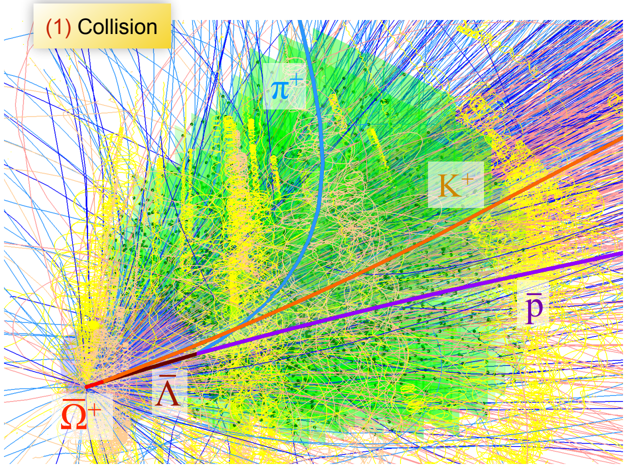
CBM (FAIR/GSI)

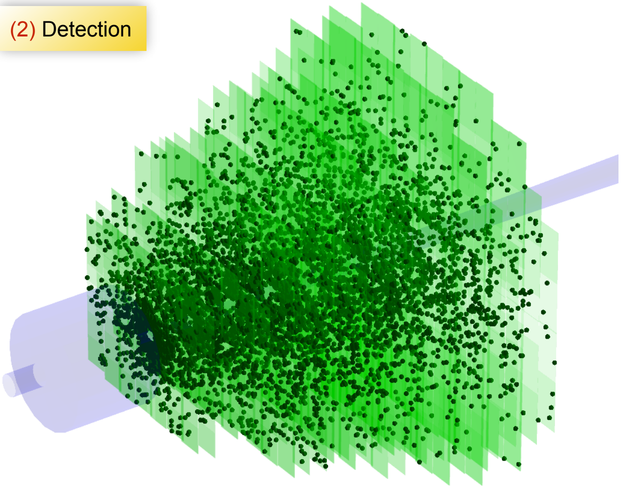


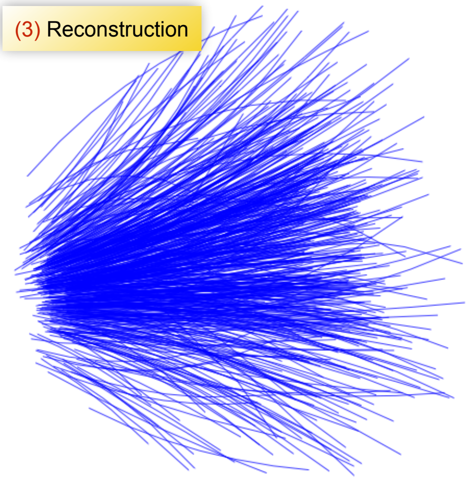
PANDA (FAIR/GSI)



Reconstruction challenge in CBM at FAIR/GSI





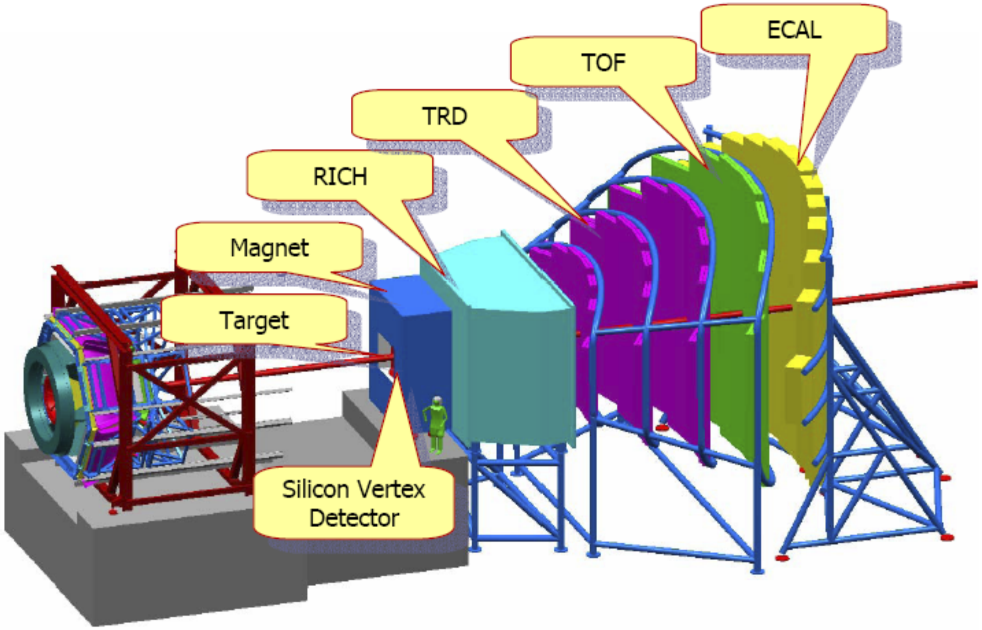


- Future fixed-target heavy-ion experiment
- 10^7 Au+Au collisions/sec
- ~ 1000 charged particles/collision
- Non-homogeneous magnetic field
- Double-sided strip detectors (85% fake space-points)

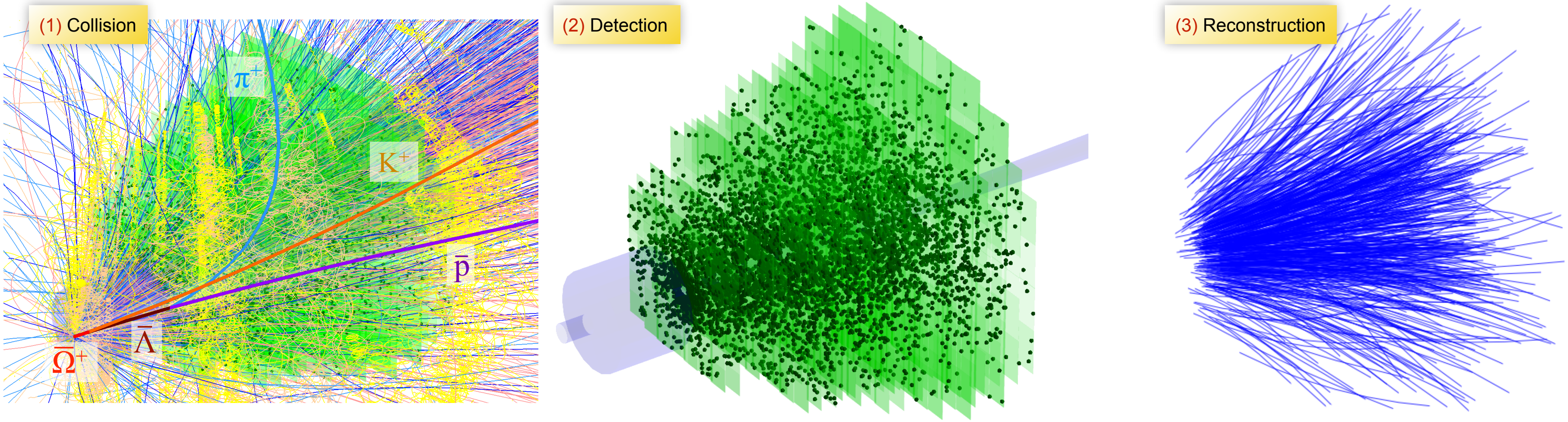
Full event reconstruction will be done
on-line at the First-Level Event Selection (FLES) and
off-line using the same FLES reconstruction package.

Cellular Automaton (CA) Track Finder
Kalman Filter (KF) Track Fitter
KF short-lived Particle Finder

All reconstruction algorithms are vectorized and parallelized.



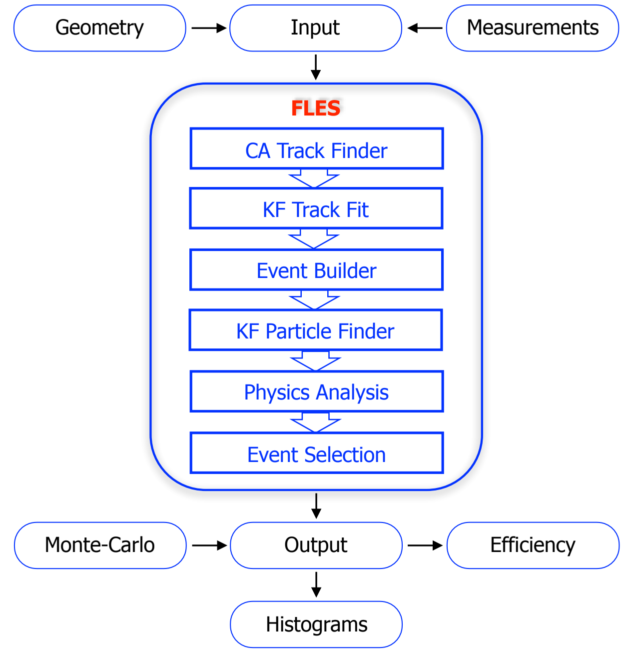
Reconstruction challenge in CBM at FAIR/GSI



The full event reconstruction will be done on-line at the First-Level Event Selection (FLES) and off-line using the same FLES reconstruction package.

- Cellular Automaton (CA) Track Finder
- Kalman Filter (KF) Track Fitter
- KF short-lived Particle Finder

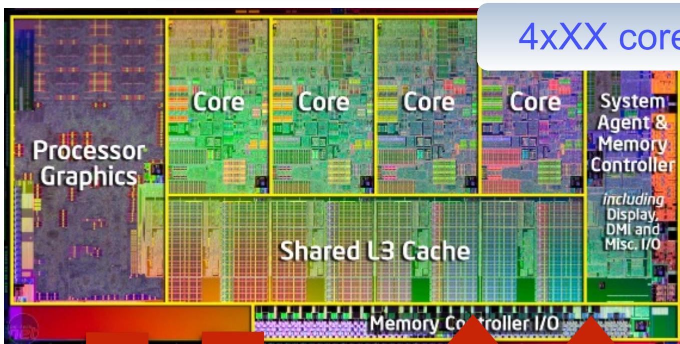
All reconstruction algorithms are vectorized and parallelized.



Many-Core CPU/GPU Architectures

Intel/AMD CPU

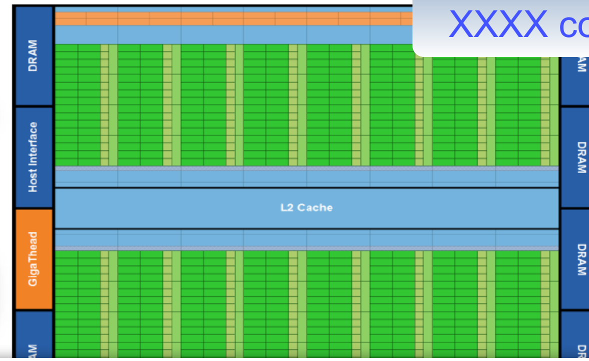
4xXX cores



- Optimiz or lo tency access ache ata sets
- Control for -of-order and cula execution

Nvidia/ATI GPU

XXXX cores



- ptimized for data-parallel, throughput computation
- lore transistors dedicated to computation

Math →

← Memory

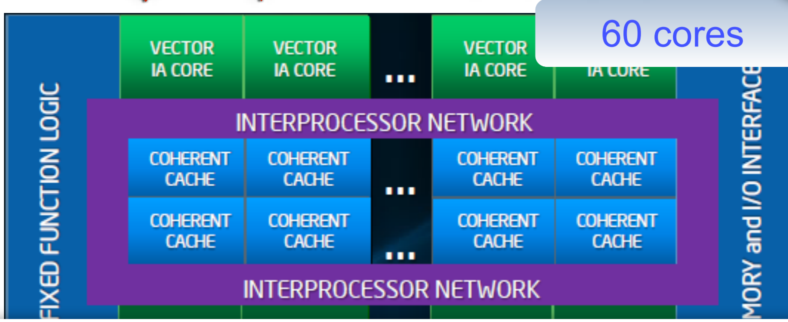
Parallelism ↓ Math ↓ Memory ↑ #Cores ↑

Stability

Memory

Intel Phi

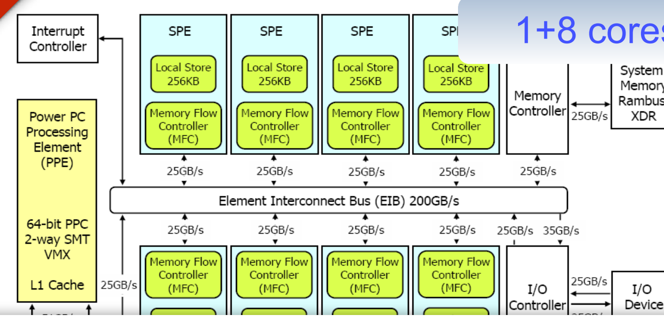
60 cores



- 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. Fundamental redesign of traditional approaches to data processing is necessary

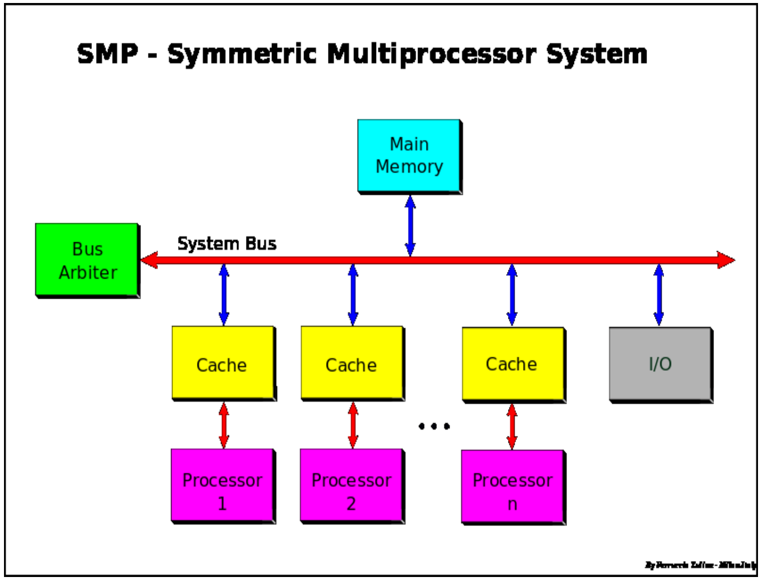
Multi-/Many-Core CPU systems

SMP (symmetric **multi**-processor) systems:

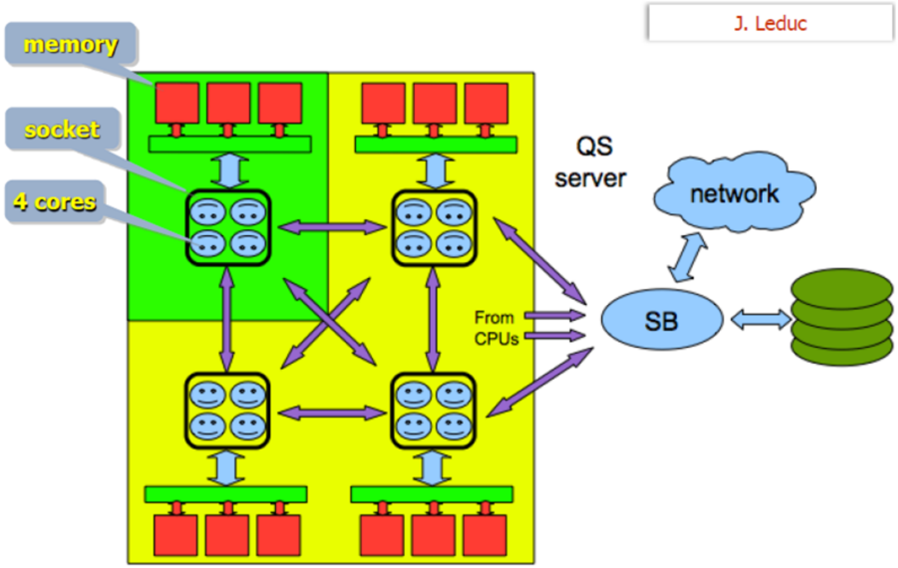
- Homogeneous
- "Equal-time" access for each processor to any part of the memory

NUMA (non-uniform memory access) systems:

- Heterogeneous
- Non uniform access to different parts of the main memory – different speed, data should be close to the CPU



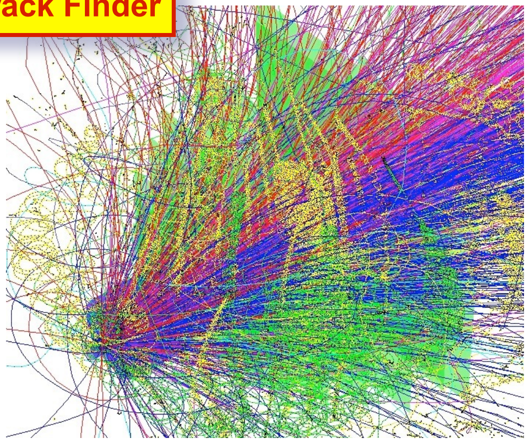
Multi-core system with ≤ **8** cores



Many-core system with > **8** cores

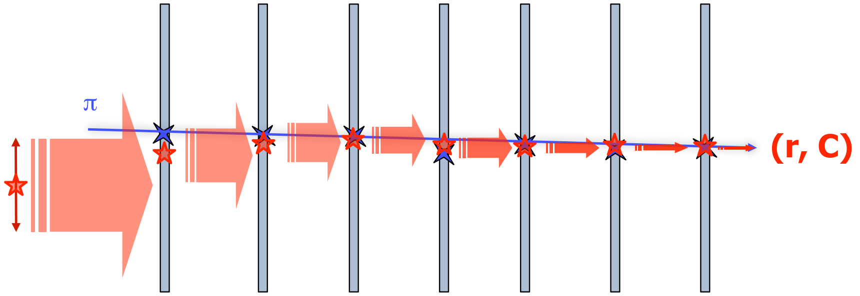
Stages of Event Reconstruction

1 // Track Finder



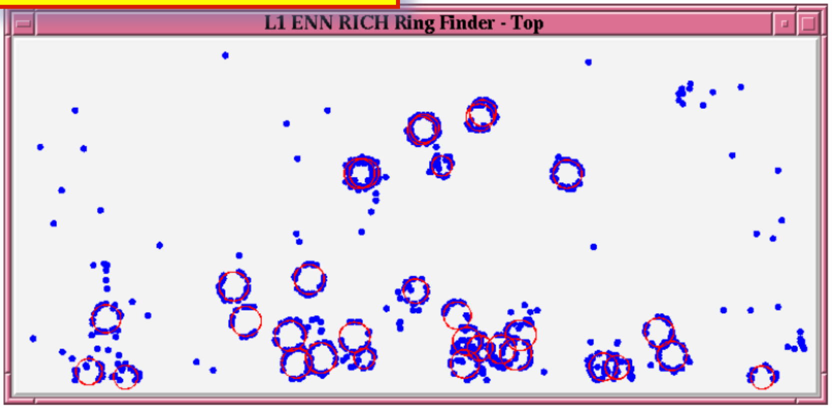
- Conformal Mapping
- Hough Transformation
- Track Following
- **Cellular Automaton**

2 // Track Fitter



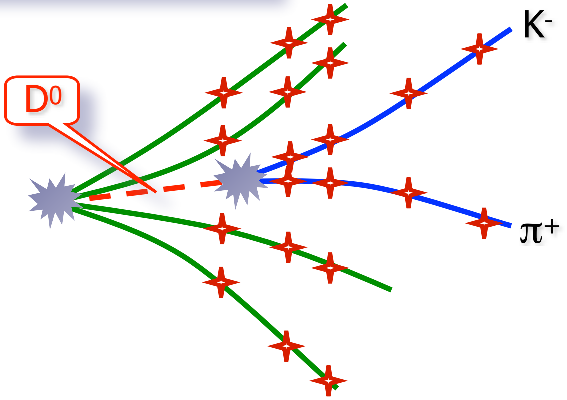
- **Kalman Filter**

3 // Ring Finder (Particle ID)



- Hough Transformation
- Elastic Neural Net

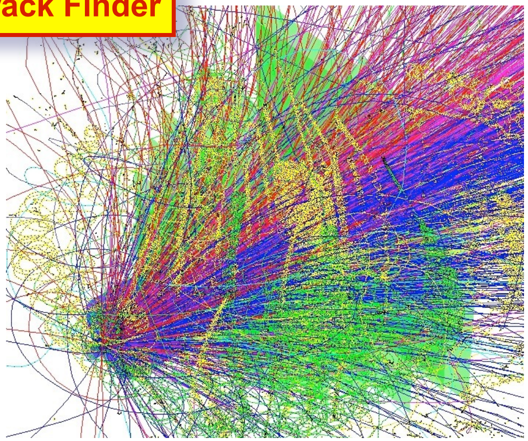
4 // Short-Lived Particles Finder



- **Kalman Filter**

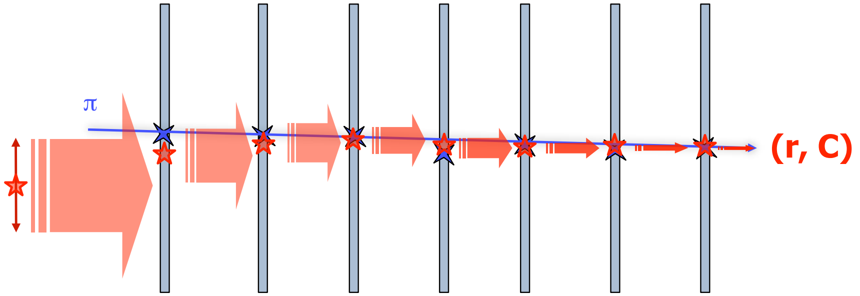
Stages of Event Reconstruction

1 // Track Finder



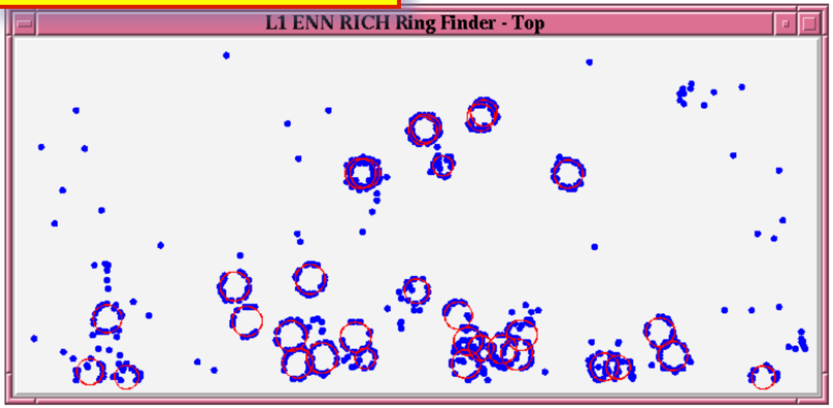
- Conformal Mapping
- Hough Transformation
- Track Following
- **Cellular Automaton**

2 // Track Fitter



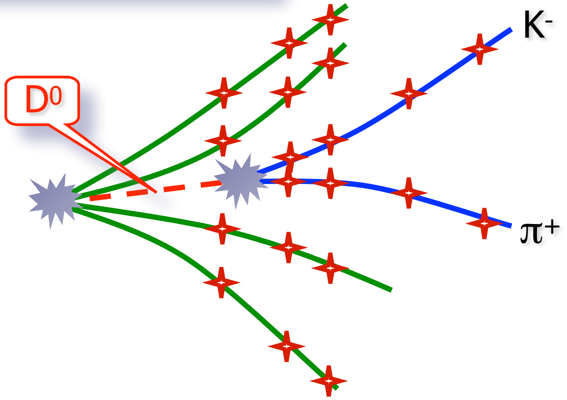
- **Kalman Filter**

3 // Ring Finder (Particle ID)



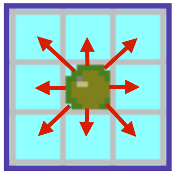
- Hough Transformation
- Elastic Neural Net

4 // Short-Lived Particles Finder

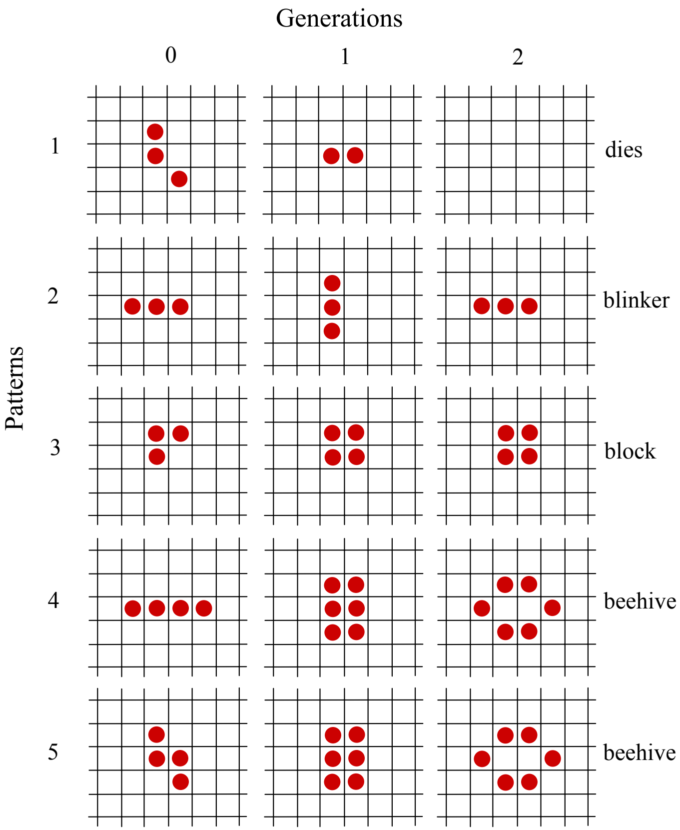


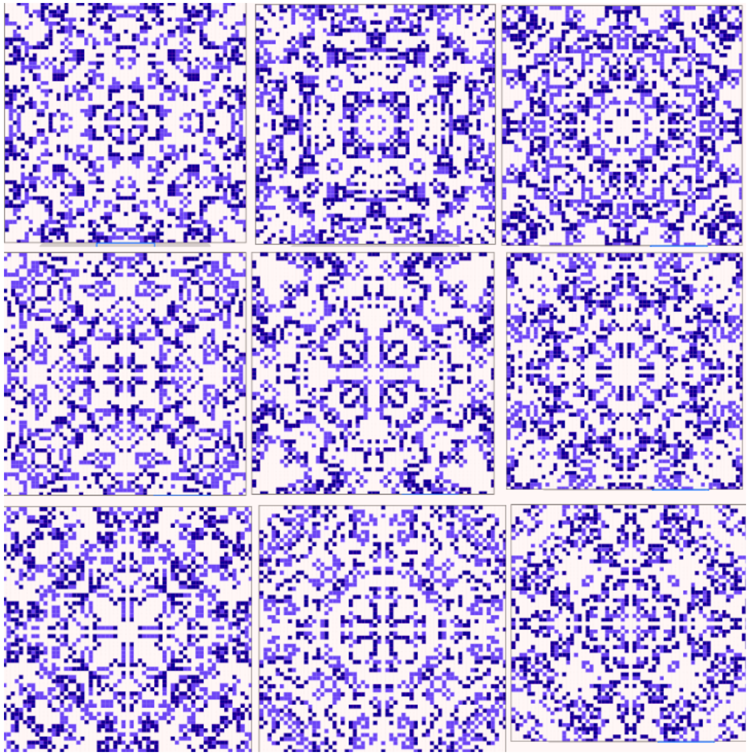
- **Kalman Filter**

Cellular Automaton - Game “Life”



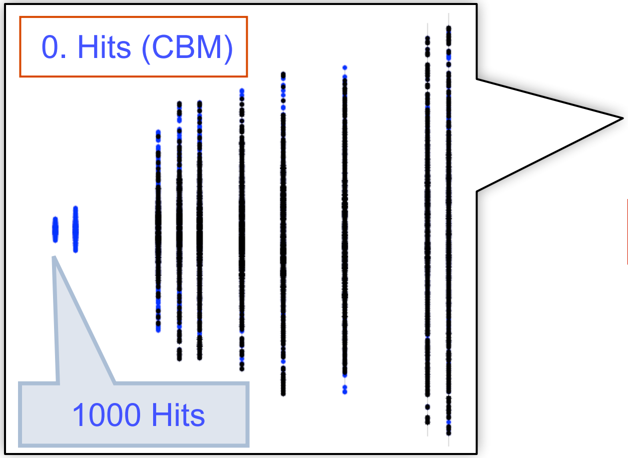
Each **cell** has 8 neighboring cells: 4 adjacent orthogonally, 4 adjacent diagonally. The **rules** are:
Survival: Each living cell with **2** or **3** adjacent living cells survives for the next generation.
Death: Each living cell with **4** or **more** living neighbors dies of overpopulation, with **1** or **none** neighbor dies of isolation.
Birth: Each empty cell adjacent to exactly **3** living neighbors is a birth cell.
All births and deaths (the transition to a new *generation*) occur for all cells *simultaneously*, i.e. in *discrete steps*.

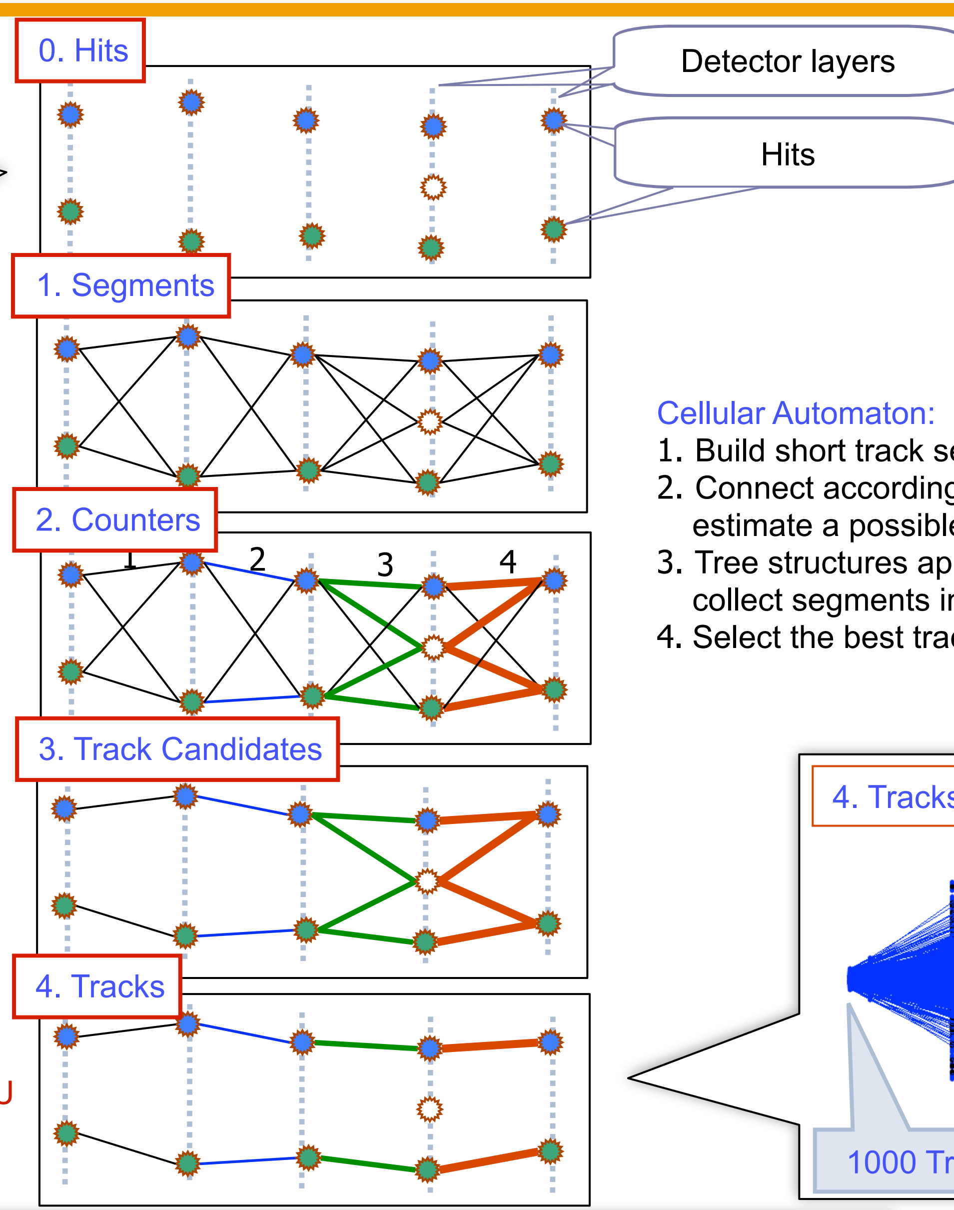


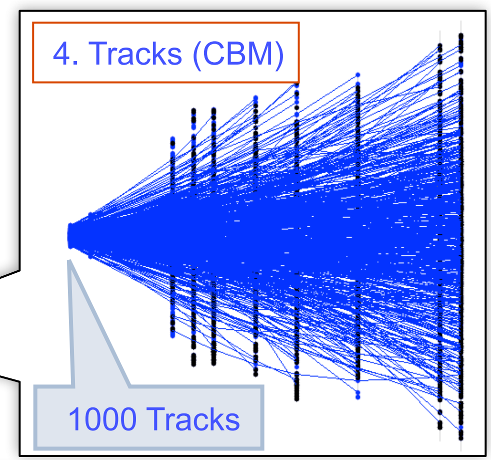


Martin Gardner, Mathematical games: The fantastic combinations of John Conway‘s new solitaire game „Life“, Sci. Amer., 223 (1970) 120-123

Cellular Automaton (CA) Track Finder







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.

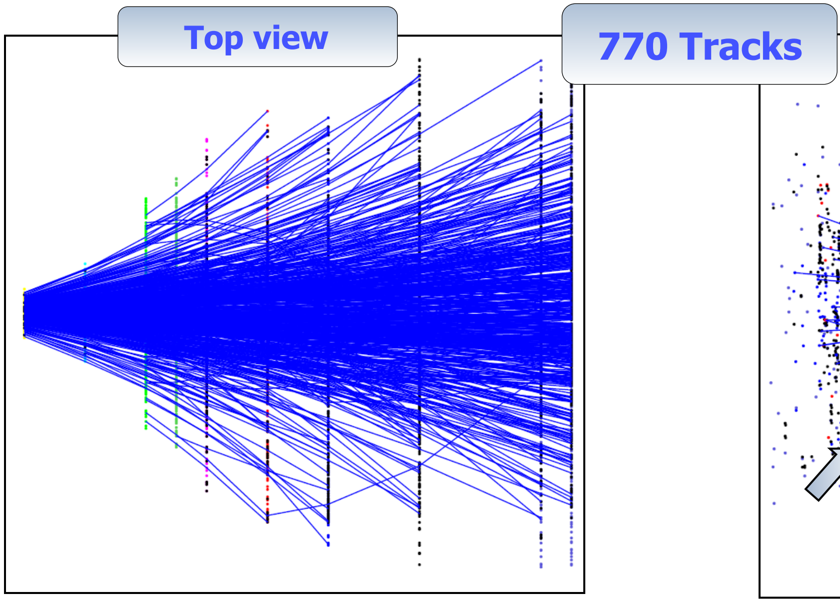
Cellular Automaton:

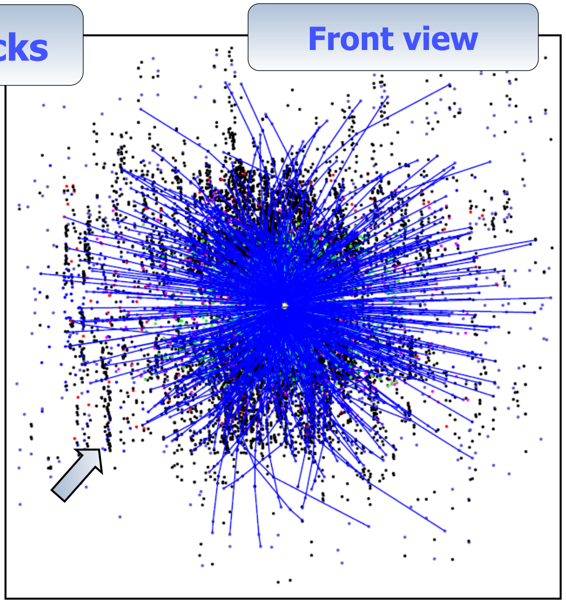
- local w.r.t. data
- intrinsically parallel
- extremely simple
- very fast

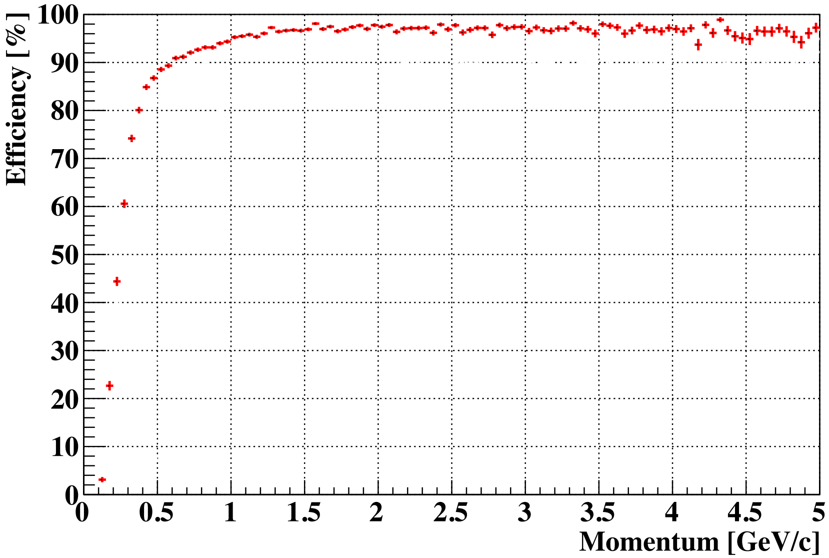
Deeply appropriate for many-core CPU/GPU

Useful for complicated event topologies with heavy combinatorics

Cellular Automaton (CA) Track Finder





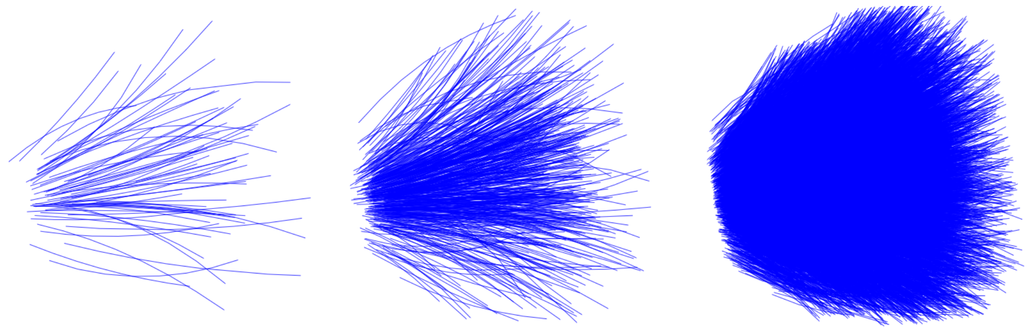


Track category	Eff, %
All tracks	90.9
Primary high-p	97.5
Primary low-p	92.6
Secondary high-p	91.1
Secondary low-p	63.8
Clone level	0.4
Ghost level	5.9
MC tracks found	134
Time, ms/ev	10

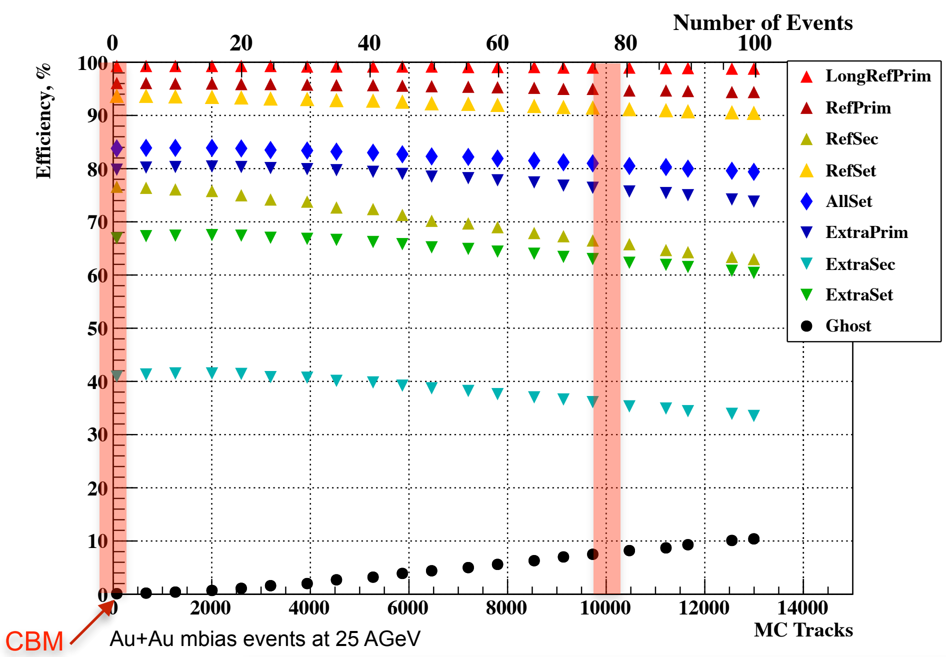
Fast and efficient track finder

CA Track Finder at High Track Multiplicity

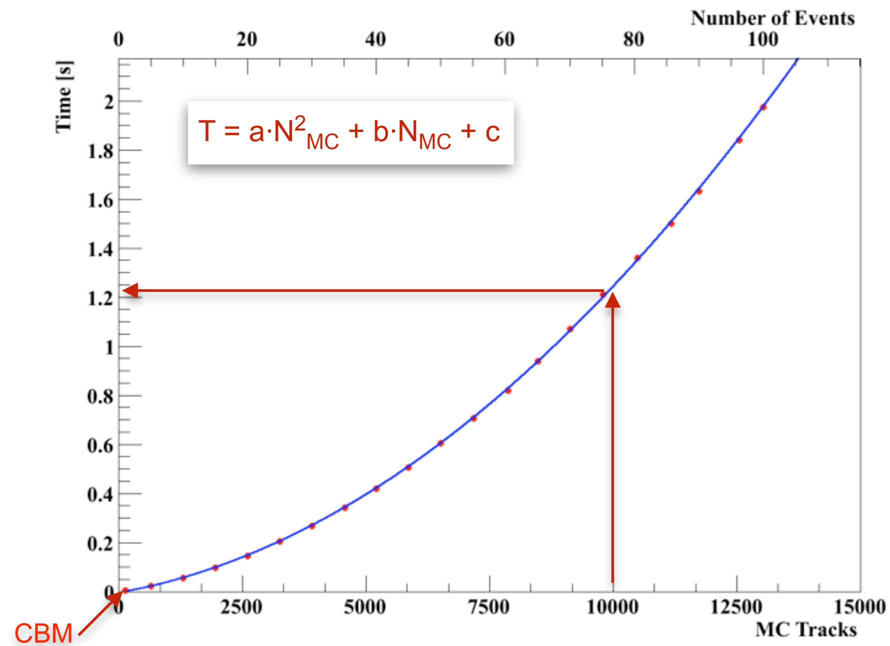
A number of minimum bias events is gathered into a group (super-event), which is then treated by the CA track finder as a single event.



1 mbias event, <N_{reco}> = 109   5 mbias events, <N_{reco}> = 572   100 mbias events, <N_{reco}> = 10340

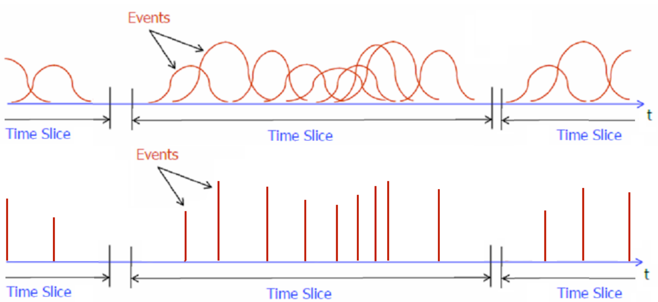


Au+Au mbias events at 25 AGeV



Reliable reconstruction efficiency and time as a second order polynomial w.r.t. to the track multiplicity

Time-based (4D) Track Reconstruction

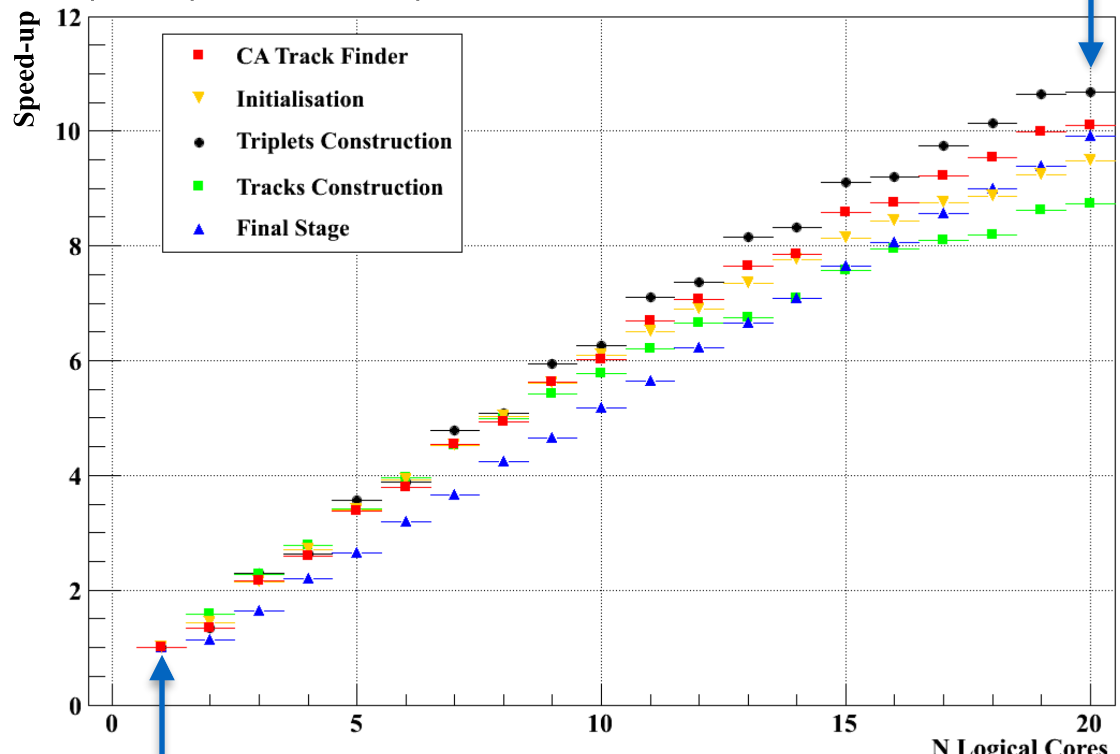


- The beam in the CBM will have no bunch structure, but continuous.
- Measurements in this case will be 4D (x, y, z, t).
- Significant overlapping of events in the detector system.
- Reconstruction of time slices rather than events is needed.

Efficiency, %	3D	4D
All tracks	83.8	83.0
Primary high-p	96.1	92.8
Primary low-p	79.8	83.1
Secondary high-p	76.6	73.2
Secondary low-p	40.9	36.8
Clone level	0.4	1.7
Ghost level	0.1	0.3
Time/event/core, ms	8.2	8.5

Total CA time = 84 ms

Speed-up factor due to parallelization within the time-slice



Total CA time = 849 ms

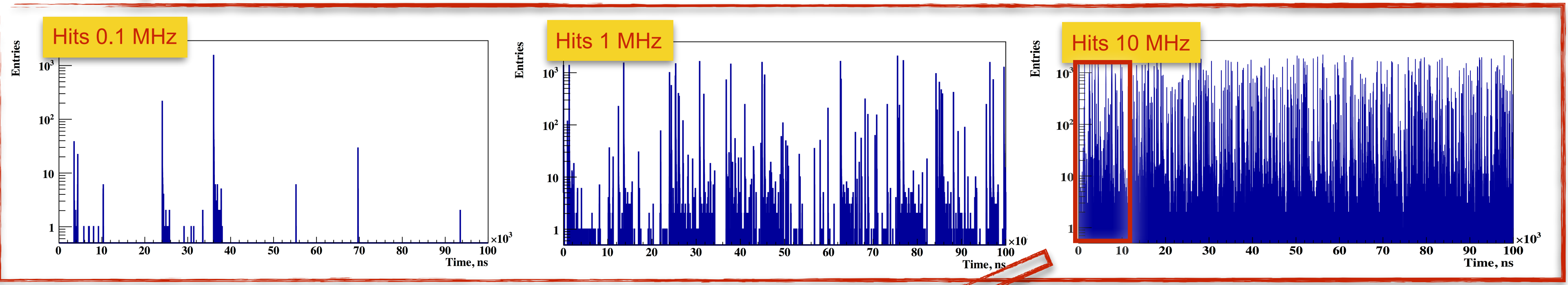
100 mbias events in a time-slice

The reconstruction time 8.2 ms/event in 3D is recovered in 4D case as well

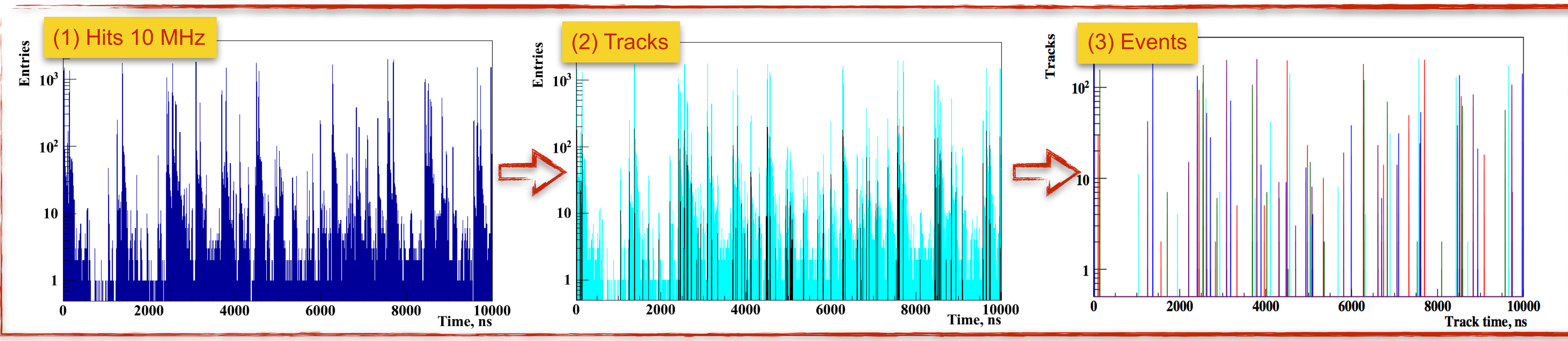
1

4D Event Building at 10 MHz

Hits at high input rates



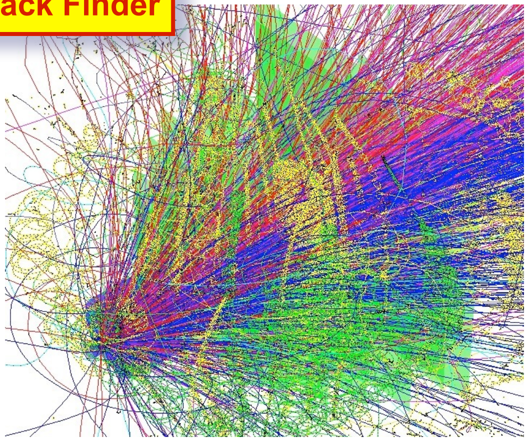
From hits to tracks to events



Reconstructed tracks clearly represent groups, which correspond to the original events

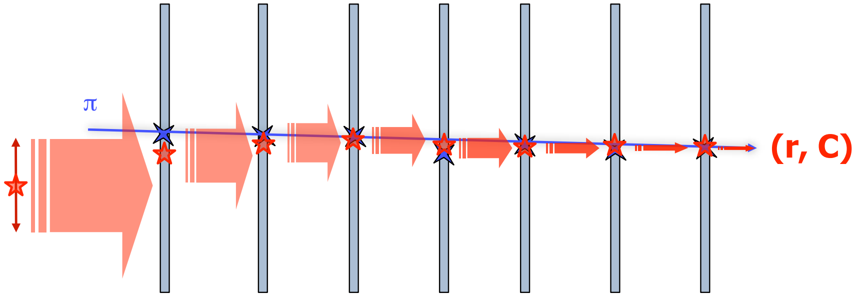
Stages of Event Reconstruction

1 **// Track Finder**



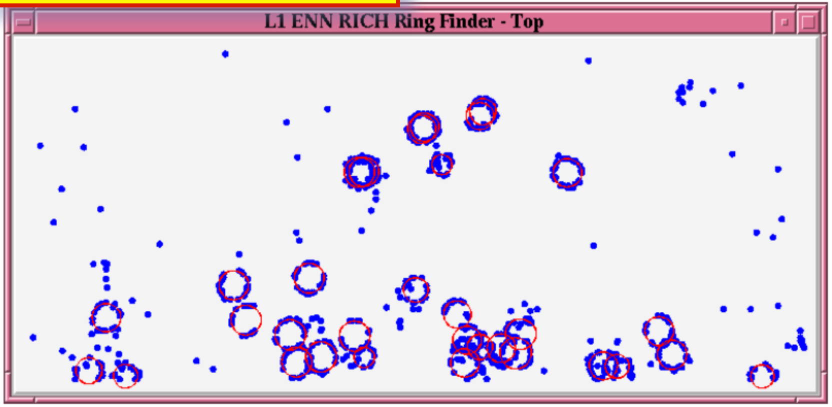
- Conformal Mapping
- Hough Transformation
- Track Following
- **Cellular Automaton**

2 **// Track Fitter**



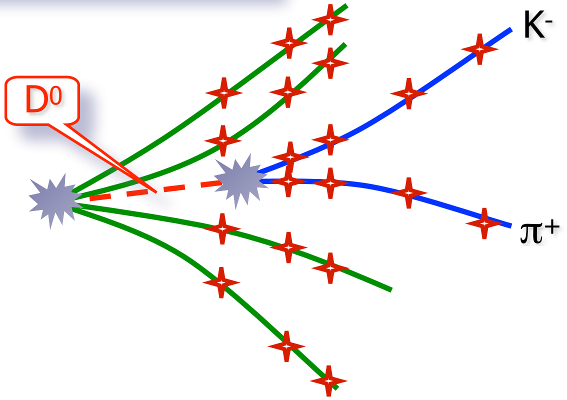
- **Kalman Filter**

3 **// Ring Finder (Particle ID)**



- Hough Transformation
- Elastic Neural Net

4 **// Short-Lived Particles Finder**

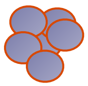


- **Kalman Filter**

Kalman Filter Algorithm

The Kalman filter is a recursive estimator – only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state.

mean value over n measurements

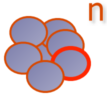


n

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i$$

previous estimation

new measurement



n+1

$$\mu_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{n}{n+1}\left(\frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}x_{n+1}\right) = \frac{n}{n+1}\mu_n + \frac{1}{n+1}x_{n+1} = \mu_n + \frac{1}{n+1}(x_{n+1} - \mu_n)$$

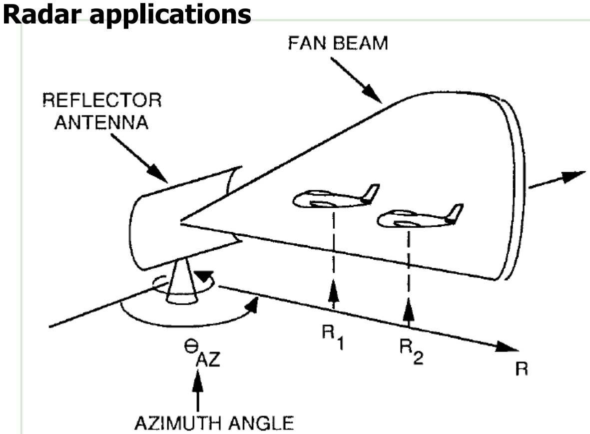
mean value over n+1 measurements

weight

correction

For this work, U.S. President Barack Obama rewarded Rudolf Kálmán with the National Medal of Science on October 7, 2009.

Radar applications



state vector:

$$\mathbf{r} = \{ x, y, z, v_x, v_y, v_z \}$$

covariance matrix:

$$C = \begin{Bmatrix} \sigma^2_x & & & & & \\ & \sigma^2_y & & \dots & & \\ & & \sigma^2_z & & & \\ & & & \sigma^2_{vx} & & \\ & \dots & & & \sigma^2_{vy} & \\ & & & & & \sigma^2_{vz} \end{Bmatrix}$$

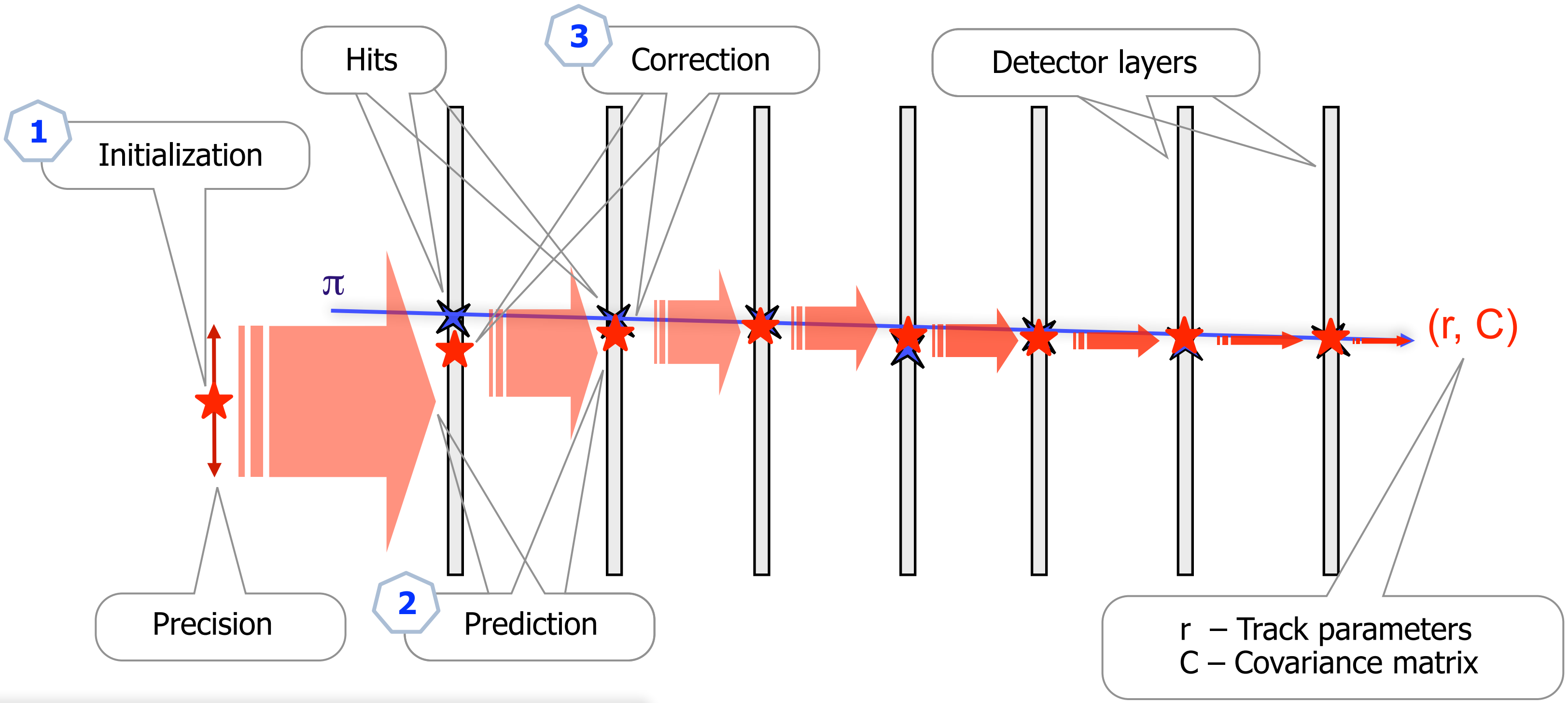
Apollo Flight Journal

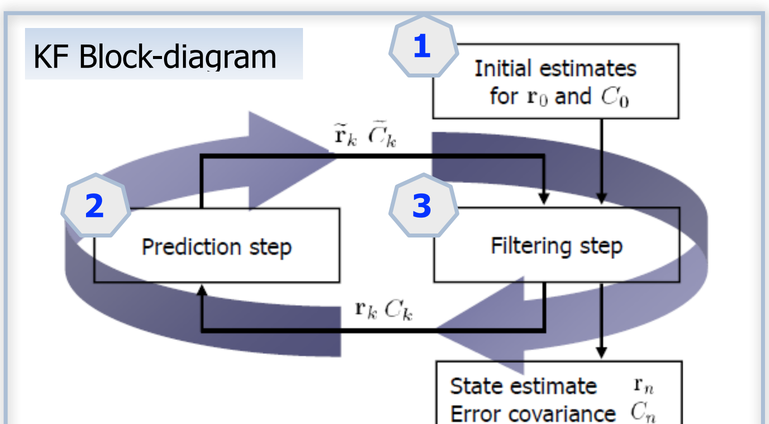
December 21, 1968. The Apollo 8 spacecraft has just been sent on its way to the Moon.

003:46:31 Collins: Roger. At your convenience, would you please go P00 and Accept? We're going to update to your W-matrix.

Kalman Filter (KF) based Track Fit

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





KF as a recursive least squares method

State vector

Position, direction and momentum

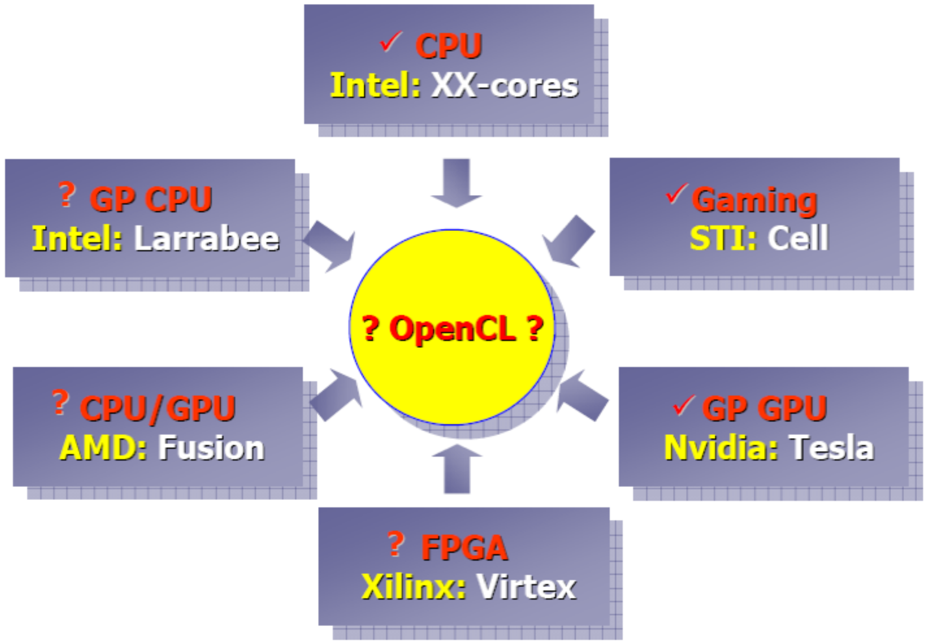
$r = \{ x, y, z, p_x, p_y, p_z \}$

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

CPU/GPU Programming Frameworks



- Cg, OpenGL Shading Language, Direct X
  - Designed to write shaders
  - Require problem to be expressed graphically
- AMD Brook
  - Pure stream computing
  - No hardware specific
- AMD CAL (Compute Abstraction Layer)
  - Generic usage of hardware on assembler level
- NVIDIA CUDA (Compute Unified Device Architecture)
  - Defines hardware platform
  - Generic programming
  - Extension to the C language
  - Explicit memory management
  - Programming on thread level
- Headers and Vector classes (Vc)
  - Overload of C operators with SIMD/SIMT instructions
  - Uniform approach to all CPU/GPU families
  - Uni-Hedeilberg/Uni-Frankfurt/FIAS/GSI
- Intel Ct (C for throughput)
  - Extension to the C language
  - Intel CPU/GPU specific
  - SIMD exploitation for automatic parallelism
- 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

2

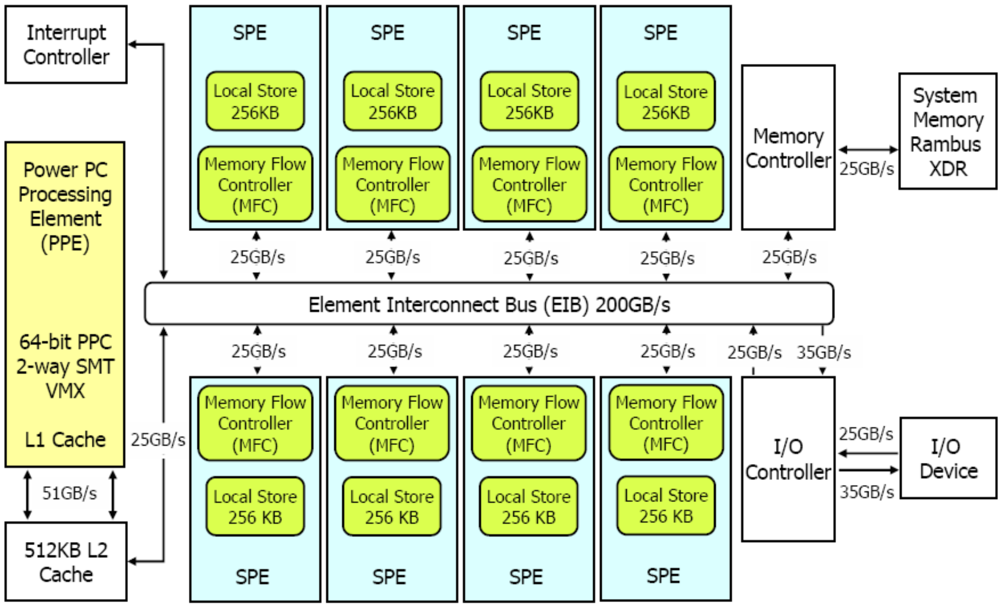
Cell Processor: Supercomputer on a Chip

Since 2005 ...

Power Processor Element (PPE):
- General Purpose, 64-bit RISC Processor (PowerPC AS 2.0)
- 2-Way Hardware Multithreaded
- L1 : 32KB I ; 32KB D
- L2 : 512KB
- Coherent load/store
- VMX
- 3.2 GHz

Synergistic Processor Elements (SPE):
- 8 per chip
- 128-bit wide SIMD Units
- Integer *and* Floating Point capable
- 256KB Local Store
- Up to 25.6 GF/s per SPE --- 200GF/s total *

** At clock speed of 3.2GHz*



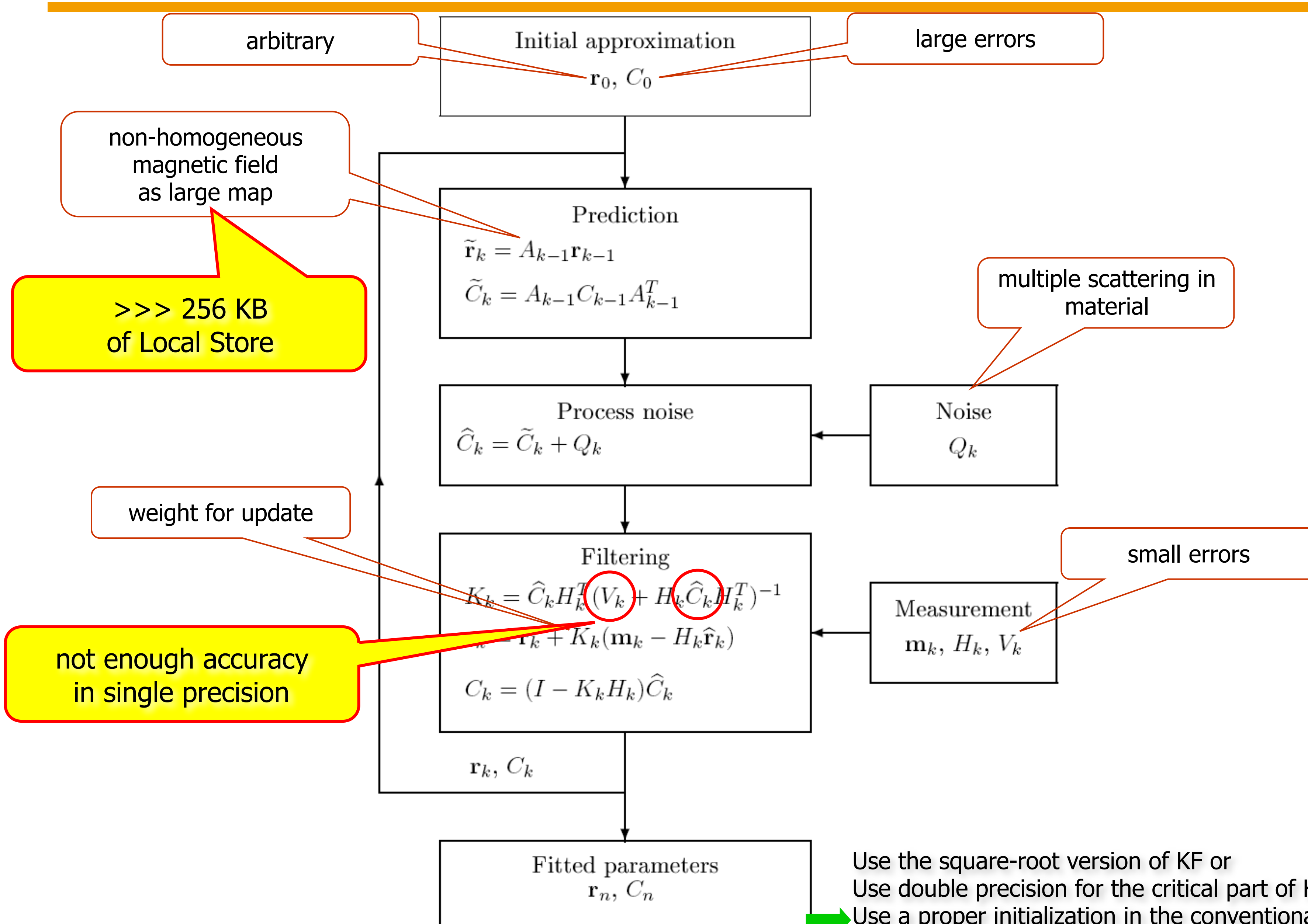
Internal Interconnect:
- Coherent ring structure
- 300+ GB/s total internal interconnect bandwidth
- DMA control to/from SPEs supports >100 outstanding memory requests

External Interconnects:
- 25.6 GB/sec BW memory interface
- 2 Configurable I/O Interfaces
  - Coherent interface (SMP)
  - Normal I/O interface (I/O & Graphics)
  - Total BW configurable between interfaces
  - Up to 35 GB/s out
  - Up to 25 GB/s in

- Sony PlayStation-3 -> cheap
- 32 (8x4) times faster !

Memory Management & Mapping
- SPE Local Store aliased into PPE system memory
- MFC/MMU controls SPE DMA accesses
  - Compatible with PowerPC Virtual Memory architecture
  - S/W controllable from PPE MMIO
- Hardware or Software TLB management
- SPE DMA access protected by MFC/MMU

Kalman Filter for Track Fit



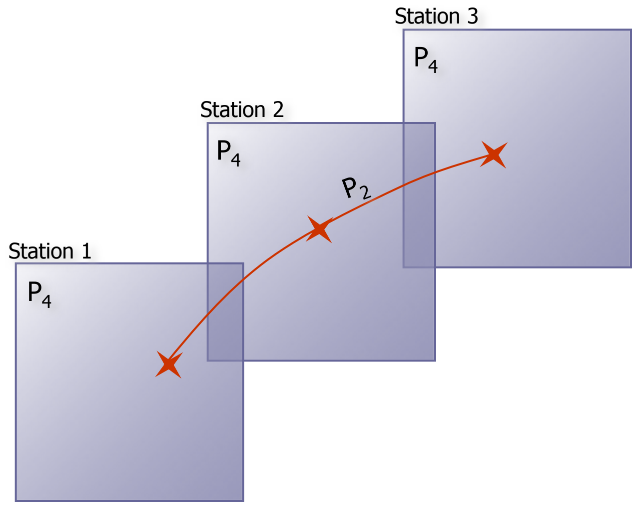
Use the square-root version of KF or
Use double precision for the critical part of KF or
Use a proper initialization in the conventional KF

„Local“ Approximation of the Magnetic Field

2

Problem:

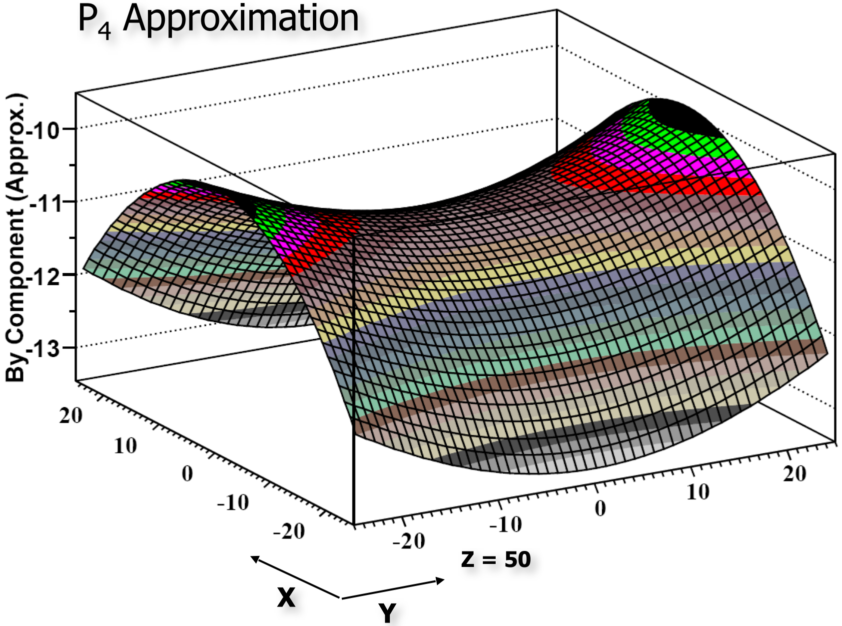
- Full reconstruction must work within **256 kB** of the Local Store.
- The magnetic field map is too large for that (**70 MB**).
- A position (x,y), to which the track is propagated, is unknown in advance.
- Therefore, access to the magnetic field map is a blocking process.



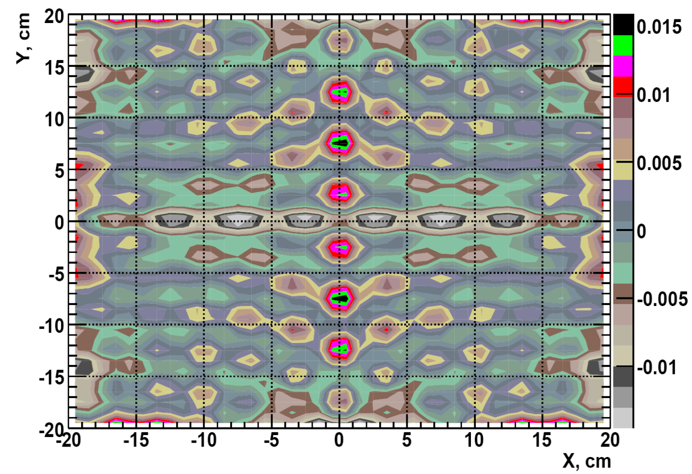
Solution:

1. Use a polynomial approximation (4-th order) of the field in **XY** planes of the stations.
2. Assuming a parabolic behavior of the field between stations calculate the magnetic field **along the track** based on 3 consecutive measurements.

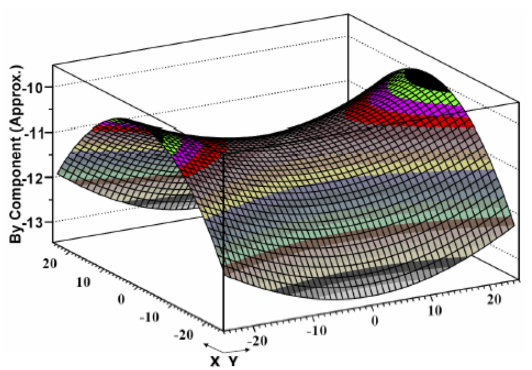
P_4 Approximation



Difference



Kalman Filter Track Fit on Cell



Stage	Description	Time/track	Speedup
	Initial scalar version	12 ms	–
1	Approximation of the magnetic field	240 μs	50
2	Optimization of the algorithm	7.2 μs	35

- The initial track parameters are directly estimated from the input data.
- The propagation step is performed directly from measurement to measurement without intermediate steps.
- Matrix multiplications have been replaced by direct operations on only non-trivial matrix elements.
- Most loops have been unrolled in order to provide additional instructions for interleaving.
- All branches have been eliminated from the algorithm to avoid branch misprediction penalty.
- Calculations have been reordered for better use of the processors pipeline.

Kalman Filter Track Fit on Cell

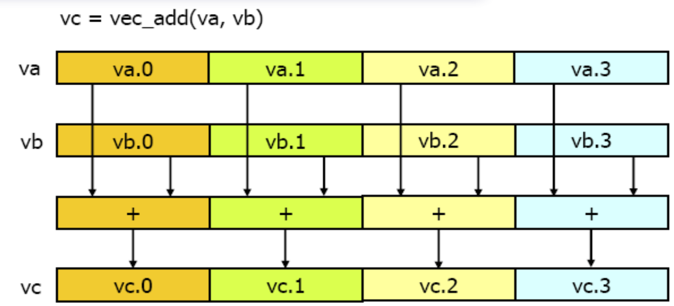
Approach:
1. Universality (any multi-core architecture)
2. Vectorization (SIMDization)
3. Run SPEs independently (one collision per SPE)

Use headers to overload +, -, *, / operators --> the source code is unchanged !

Data Types:

1. Scalar double
2. Scalar float
3. Pseudo-vector (array)
4. Vector (4 float)

c = a + b



Platform:

1. Linux (**SSE2**)
2. Virtual machine: (**SSE2**)
   - ✓ Red Hat (Fedora Core 4)
   - ✓ Cell Simulator:
     - ❖ PPE (**AltiVec**)
     - ❖ SPE (**Specialized SIMD**)
3. Cell Blade

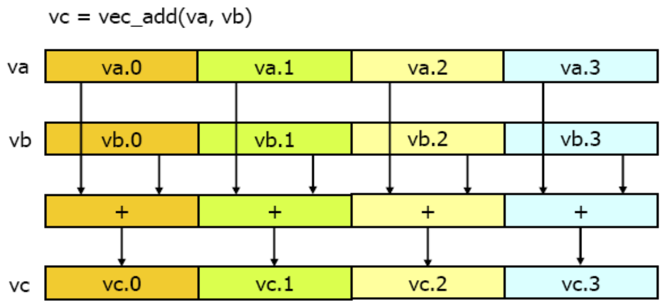
Header (Intel SSE), later Vector Classes (Vc)

SIMD instructions

```
typedef F32vec4 Fvec_t;
 /* Arithmetic Operators */
 friend F32vec4 operator +(const F32vec4 &a, const F32vec4 &b) { return _mm_add_ps(a,b); }
 friend F32vec4 operator –(const F32vec4 &a, const F32vec4 &b) { return _mm_sub_ps(a,b); }
 friend F32vec4 operator *(const F32vec4 &a, const F32vec4 &b) { return _mm_mul_ps(a,b); }
 friend F32vec4 operator /(const F32vec4 &a, const F32vec4 &b) { return _mm_div_ps(a,b); }
 /* Functions */
 friend F32vec4 min( const F32vec4 &a, const F32vec4 &b ){ return _mm_min_ps(a, b); }
 friend F32vec4 max( const F32vec4 &a, const F32vec4 &b ){ return _mm_max_ps(a, b); }
 /* Square Root */
 friend F32vec4 sqrt ( const F32vec4 &a ){ return _mm_sqrt_ps (a); }
 /* Absolute value */
 friend F32vec4 fabs( const F32vec4 &a){ return _mm_and_ps(a, _f32vec4_abs_mask); }
 /* Logical */
 friend F32vec4 operator&( const F32vec4 &a, const F32vec4 &b ){ // mask returned
   return _mm_and_ps(a, b);
 }
 friend F32vec4 operator|( const F32vec4 &a, const F32vec4 &b ){ // mask returned
   return _mm_or_ps(a, b);
 }
 friend F32vec4 operator^( const F32vec4 &a, const F32vec4 &b ){ // mask returned
   return _mm_xor_ps(a, b);
 }
 friend F32vec4 operator!( const F32vec4 &a ){ // mask returned
   return _mm_xor_ps(a, _f32vec4_true);
 }
 friend F32vec4 operator||( const F32vec4 &a, const F32vec4 &b ){ // mask returned
   return _mm_or_ps(a, b);
 }
 /* Comparison */
 friend F32vec4 operator<( const F32vec4 &a, const F32vec4 &b ){ // mask returned
   return _mm_cmplt_ps(a, b);
 }
```

Source Code (Part of the Kalman Filter Track Fit)

```
inline void AddMaterial( TrackV &track, Station &st, Fvec_t &qp0 )
{
  cnst mass2 = 0.1396*0.1396;

  Fvec_t tx = track.T[2];
  Fvec_t ty = track.T[3];
  Fvec_t txtx = tx*tx;
  Fvec_t tyty = ty*ty;
  Fvec_t txtx1 = txtx + ONE;
  Fvec_t h = txtx + tyty;
  Fvec_t t = sqrt(txtx1 + tyty);
  Fvec_t h2 = h*h;
  Fvec_t qp0t = qp0*t;

  cnst c1=0.0136, c2=c1*0.038, c3=c2*0.5, c4=-c3/2.0, c5=c3/3.0, c6=-c3/4.0;

  Fvec_t s0 = (c1+c2*st.logRadThick + c3*h + h2*(c4 + c5*h +c6*h2) )*qp0t;

  Fvec_t a = (ONE+mass2*qp0*qp0t)*st.RadThick*s0*s0;

  CovV &C = track.C;

  C.C22 += txtx1*a;
  C.C32 += tx*ty*a; C.C33 += (ONE+tyty)*a;
}
```



Use headers to overload +, -, *, / operators --> the source code is unchanged !

Kalman Filter Track Fit on Cell

```
mysim/SPE4: Statistics

SPU DD3.0
***
Total Cycle count                335660
Total Instruction count          643
Total CPI                        522.02
***
Performance Cycle count          7076
Performance Instruction count    6890 (6638)
Performance CPI                  1.03 (1.07)

Branch instructions              26
Branch taken                     16
Branch not taken                 10

Hint instructions                7
Hint hit                         10

Contention at LS between Load/Store and Prefetch 405

Single cycle                                              4440 ( 62.7%)
Dual cycle                                                1099 ( 15.5%)
Nop cycle                                                   16 (  0.2%)
Stall due to branch miss                                   137 (  1.9%)
Stall due to prefetch miss                                   0 (  0.0%)
Stall due to dependency                                   1365 ( 19.3%)
Stall due to fp resource conflict                            1 (  0.0%)
Stall due to waiting for hint target                        18 (  0.3%)
Stall due to dp pipeline                                     0 (  0.0%)
Channel stall cycle                                          0 (  0.0%)
SPU Initialization cycle                                     0 (  0.0%)
-----------------------------------------------------------------------
Total cycle                                               7076 (100.0%)

Stall cycles due to dependency on each pipelines
 FX2         36 (  2.6% of all dependency stalls)
 SHUF        92 (  6.7% of all dependency stalls)
 FX3         0 (  0.0% of all dependency stalls)
 LS          285 ( 20.9% of all dependency stalls)
 BR          0 (  0.0% of all dependency stalls)
 SPR         0 (  0.0% of all dependency stalls)
 LNOP        0 (  0.0% of all dependency stalls)
 NOP         0 (  0.0% of all dependency stalls)
 FXB         0 (  0.0% of all dependency stalls)
 FP6         873 ( 64.0% of all dependency stalls)
 FP7         79 (  5.8% of all dependency stalls)
 FPD         0 (  0.0% of all dependency stalls)

The number of used registers are 128, the used ratio is 100.00
dumped pipeline stats
```

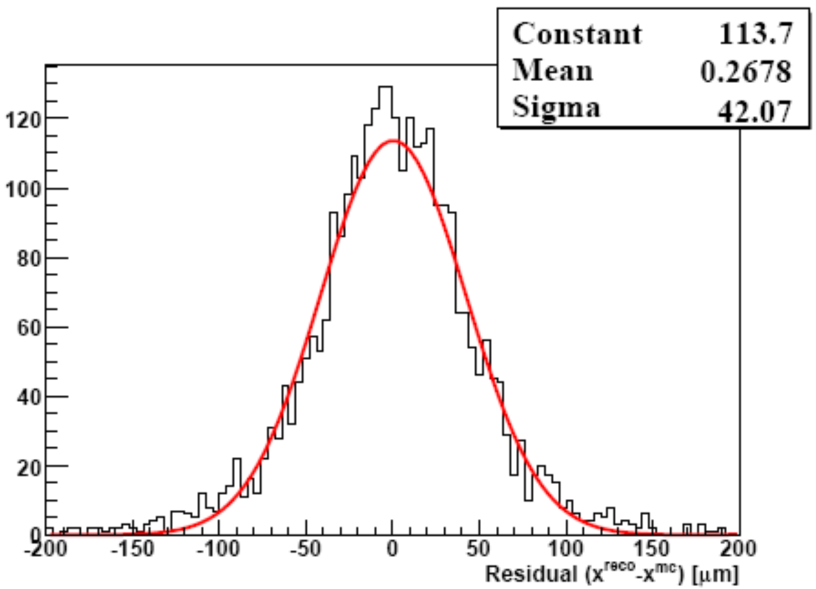
72.2%

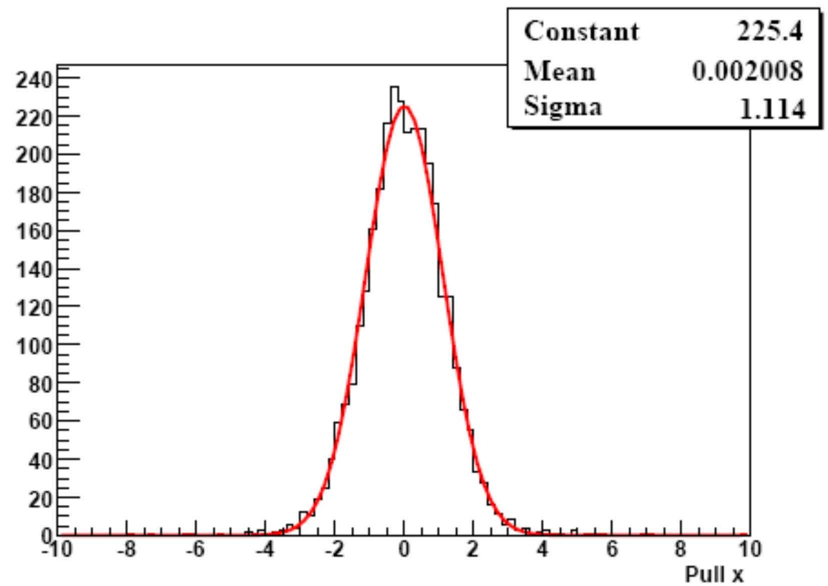
Timing profile !

No need to check
the assembler code !

Kalman Filter Track Fit on Cell

	Stage	Description	Time/track	Speedup
Intel P4		Initial scalar version	12 ms	–
Intel P4	1	Approximation of the magnetic field	240 μs	50
Intel P4	2	Optimization of the algorithm	7.2 μs	35
Intel P4	3	Vectorization	1.6 μs	4.5
Cell	4	Porting to SPE	1.1 μs	1.5
Cell	5	Parallelization on 16 SPEs	0.1 μs	10
		Final simdized version	0.1 μs	120000



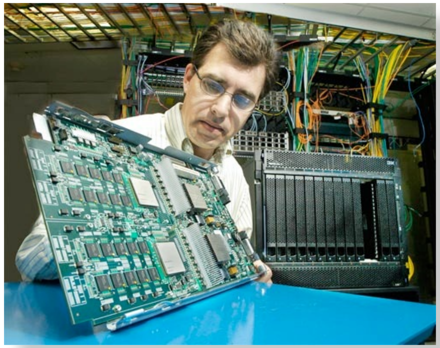


Kalman Filter Track Fit on Cell

	Stage	Description	Time/track	Speedup
		Initial scalar version	12 ms	–
Intel	1	Approximation of the magnetic field	240 μs	50
Intel	2	Optimization of the algorithm	7.2 μs	35
Intel	3	Vectorization	1.6 μs	4.5
Cell	4	Porting to SPE	1.1 μs	1.5
Cell	5	Parallelization on 16 SPEs	0.1 μs	10
		Final simdized version	0.1 μs	120000

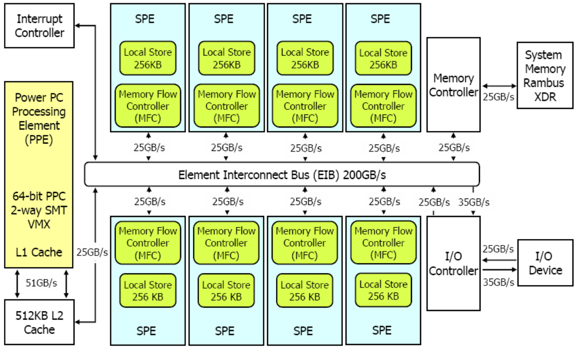
10000x faster on any PC

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 !

Kalman Filter Track Fit Library

Kalman Filter Methods

Kalman Filter Tools:
- KF Track Fitter
- KF Track Smoother
- Deterministic Annealing Filter

Kalman Filter Approaches:
- Conventional DP KF
- Conventional SP KF
- Square-Root SP KF
- UD-Filter SP
- Gaussian Sum Filter
- 3D (*x*,*y*,*z*) and 4D (*x*,*y*,*z*,*t*) KF

Track Propagation:
- Runge-Kutta
- Analytic Formula

Detector Types:
- Pixel
- Strip
- Tube
- TPC

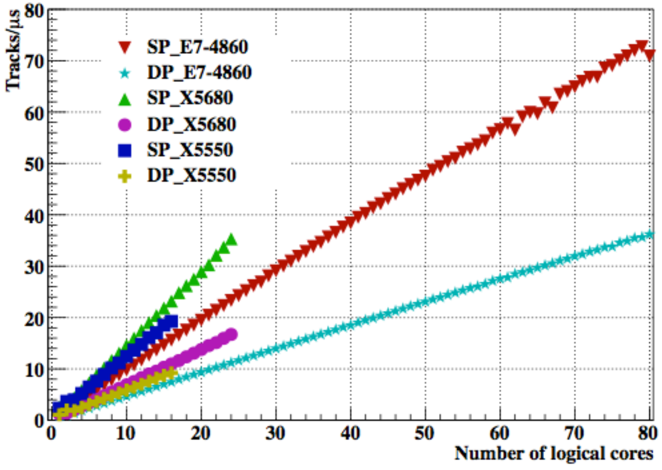
Implementations

Vectorization (SIMD):
- Header Files
- Vc Vector Classes
- ArBB Array Building Blocks
- OpenCL

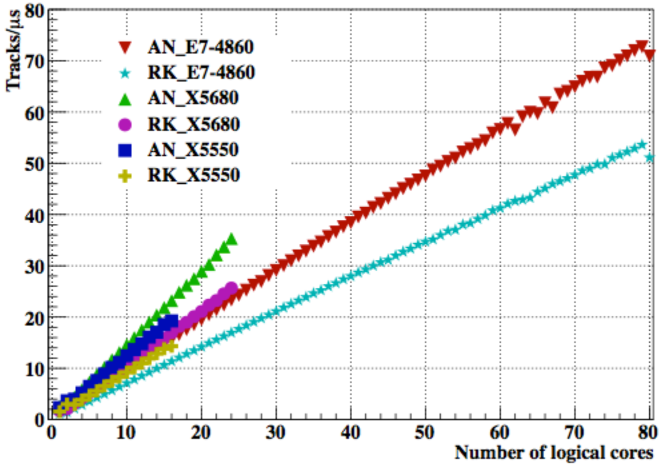
Parallelization (many-cores):
- Open MP
- ITBB
- ArBB
- OpenCL

Precision:
- single precision SP
- double precision DP

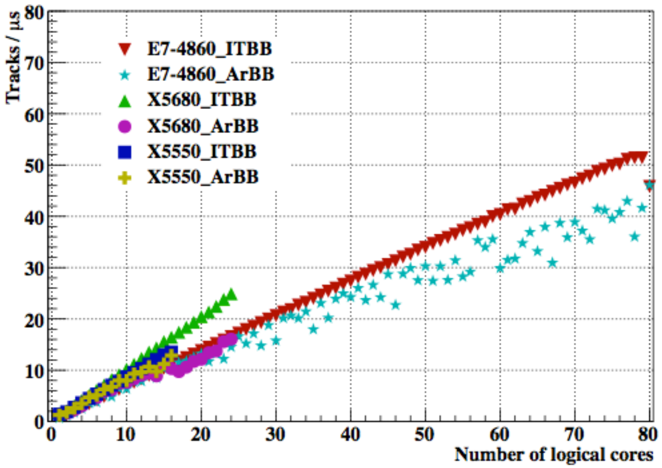
Conventional KF DP vs. SP



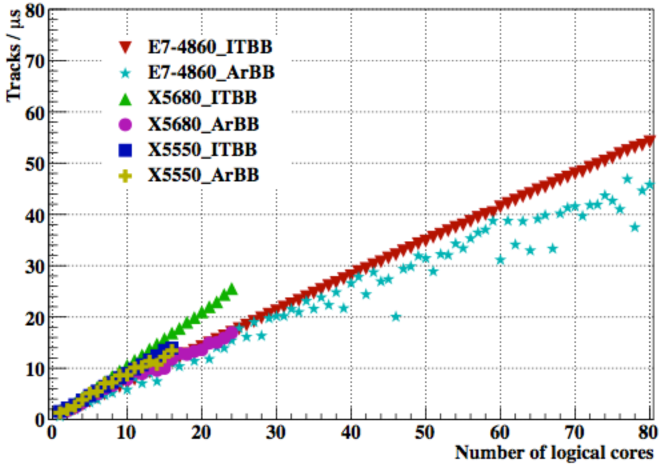
Conventional KF RK4 vs. Analytical



Square-Root KF



UD KF

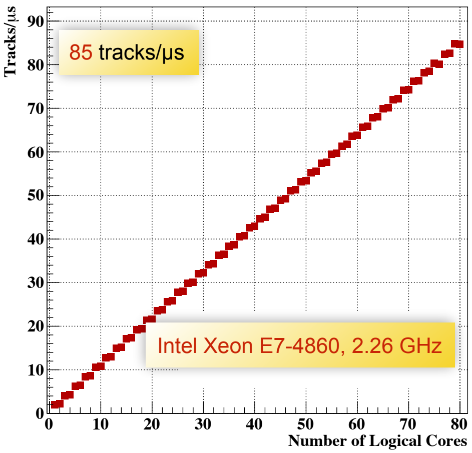


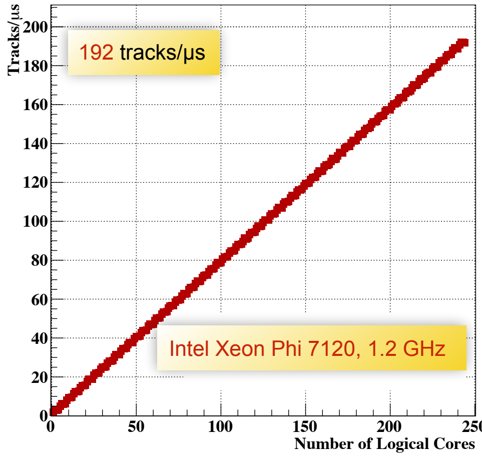
Strong many-core scalability of the Kalman filter library

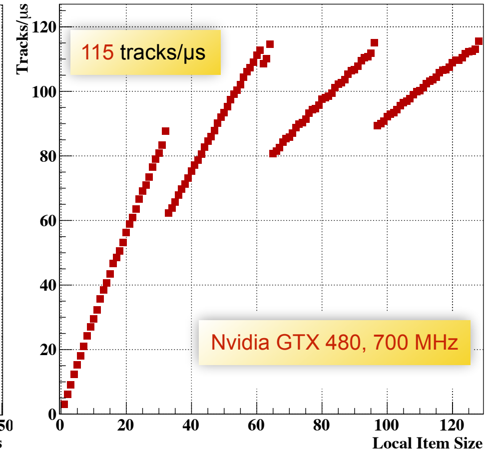
with I. Kulakov, H. Pabst* and M. Zyzak (*Intel)

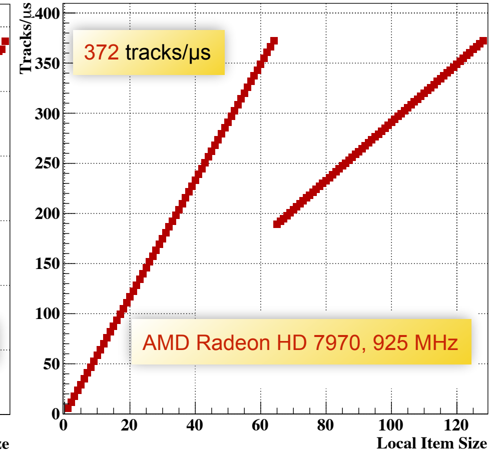
2

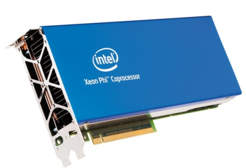
Kalman Filter (KF) Track Fit











- Precise estimation of the parameters of particle trajectories is the core of the reconstruction procedure.
- Scalability with respect to the number of logical cores in a CPU is one of the most important parameters of the algorithm.
- The scalability on the Intel Xeon Phi coprocessor is similar to the CPU, but running four threads per core instead of two.
- In case of the graphics cards the set of tasks is divided into working groups of size *local item size* and distributed among compute units (or streaming multiprocessors) and the load of each compute unit is of the particular importance.
- The track fit performance on a single node: **2*CPU+2*GPU = 10^9 tracks/s** = (100 tracks/event)* 10^7 events/s = 10^7 events/s.
- A single compute node is enough to estimate parameters of all particles produced at the maximum 10^7 interaction rate!

The fastest implementation of the Kalman filter in the world

Kalman Filter Track Fit on Cell

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Computer Physics Communications 178 (2008) 374–383

Computer Physics Communications

www.elsevier.com/locate/cpc

Fast SIMDized Kalman filter based track fit

S. Gorbunov [a,b], U. Kebschull [b], I. Kisel [b,c,*], V. Lindenstruth [b], W.F.J. Müller [a]

[a] *Gesellschaft für Schwerionenforschung mbH, 64291 Darmstadt, Germany*
[b] *Kirchhoff Institute for Physics, University of Heidelberg, 69120 Heidelberg, Germany*
[c] *Laboratory of Information Technologies, Joint Institute for Nuclear Research, 141980 Dubna, Russia*

Received 17 February 2007; received in revised form 29 August 2007; accepted 2 October 2007

Available online 7 October 2007

Abstract

Modern high energy physics experiments have to process terabytes of input data produced in particle collisions. The core of many data reconstruction algorithms in high energy physics is the Kalman filter. Therefore, the speed of Kalman filter based algorithms is of crucial importance in on-line data processing. This is especially true for the combinatorial track finding stage where the Kalman filter based track fit is used very intensively. Therefore, developing fast reconstruction algorithms, which use maximum available power of processors, is important, in particular for the initial selection of events which carry signals of interesting physics.

One of such powerful feature supported by almost all up-to-date PC processors is a SIMD instruction set, which allows packing several data items in one register and to operate on all of them, thus achieving more operations per clock cycle. The novel Cell processor extends the parallelization further by combining a general-purpose PowerPC processor core with eight streamlined coprocessing elements which greatly accelerate vector processing applications.

In the investigation described here, after a significant memory optimization and a comprehensive numerical analysis, the Kalman filter based track fitting algorithm of the CBM experiment has been vectorized using inline operator overloading. Thus the algorithm continues to be flexible with respect to any CPU family used for data reconstruction.

Because of all these changes the SIMDized Kalman filter based track fitting algorithm takes 1 μs per track that is 10000 times faster than the initial version. Porting the algorithm to a Cell Blade computer gives another factor of 10 of the speedup.

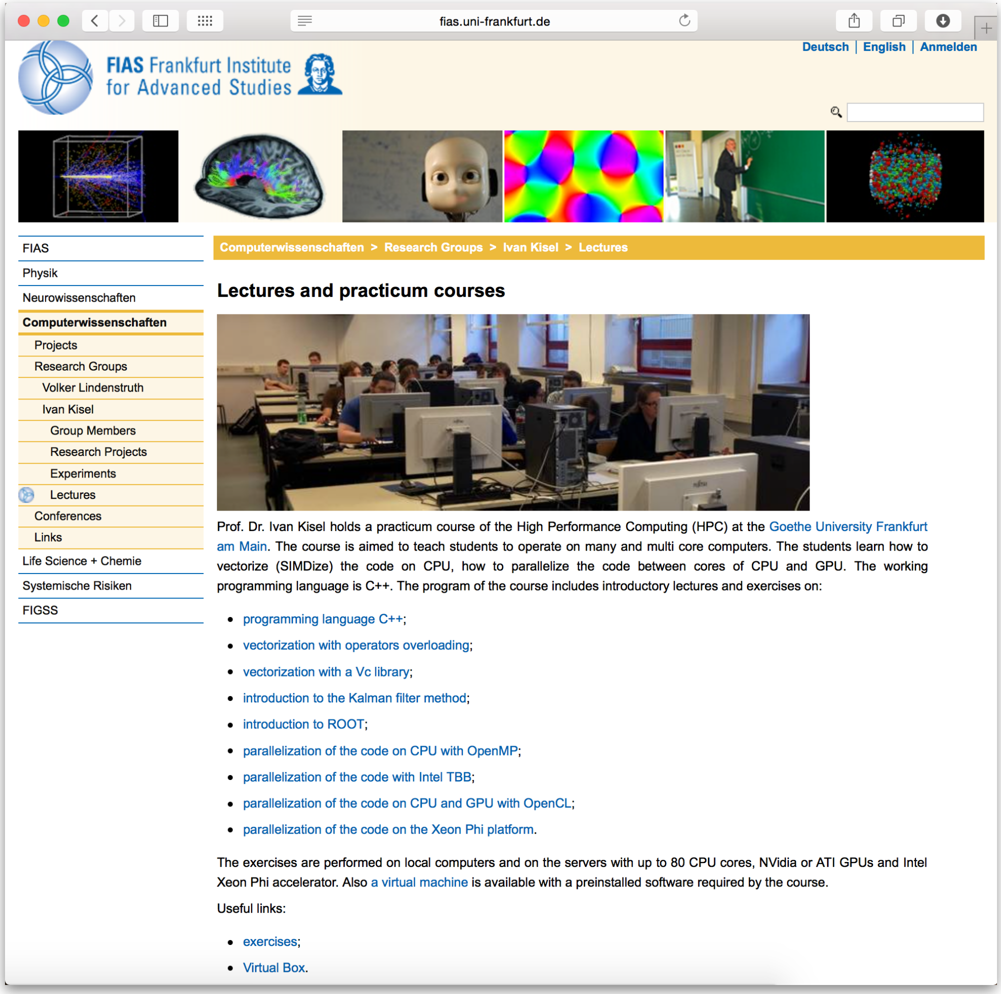
Finally, we compare performance of the tracking algorithm running on three different CPU architectures: Intel Xeon, AMD Opteron and Cell Broadband Engine.

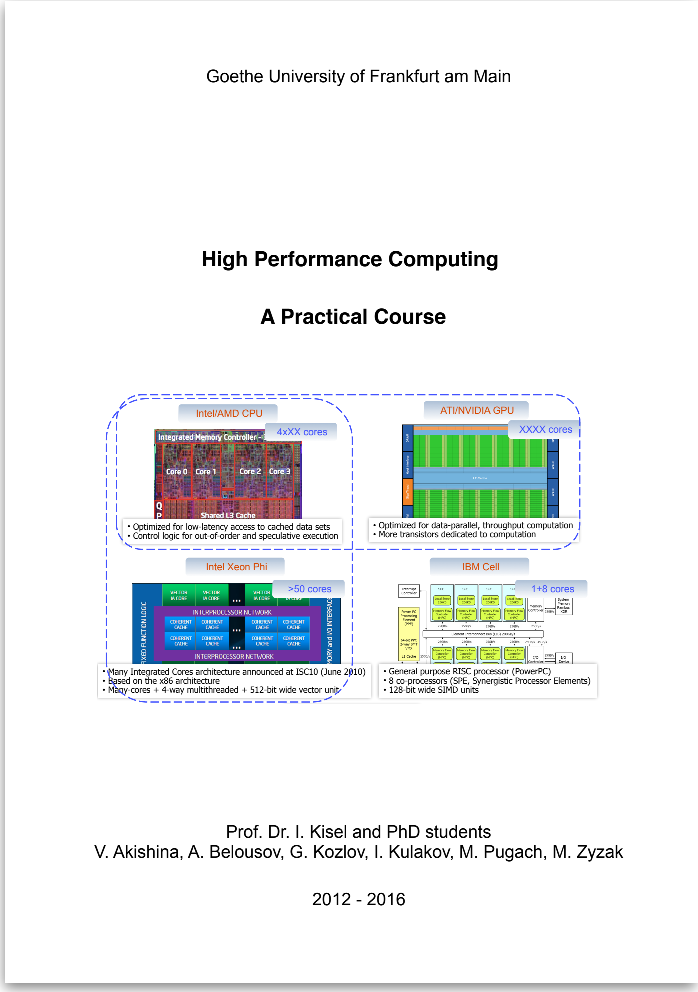
PACS: 02.60.Pn; 02.70.-c; 07.05.-t; 07.05.Bx; 07.05.Kf

Keywords: High energy physics; CBM experiment; Data reconstruction; Track fit; Kalman filter; SIMD instruction set; Cell Broadband Engine

HPC Practical Course at the Goethe University Frankfurt

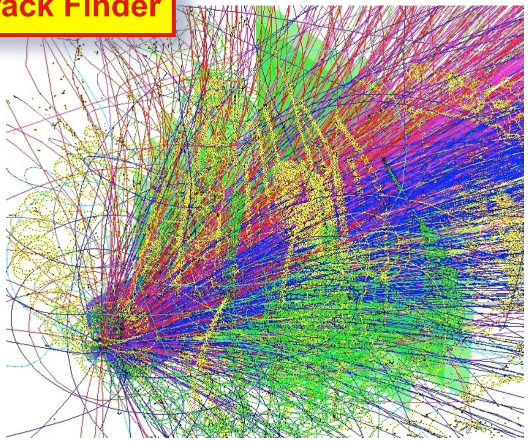
http://fias.uni-frankfurt.de/de/cs/kisel/lectures/





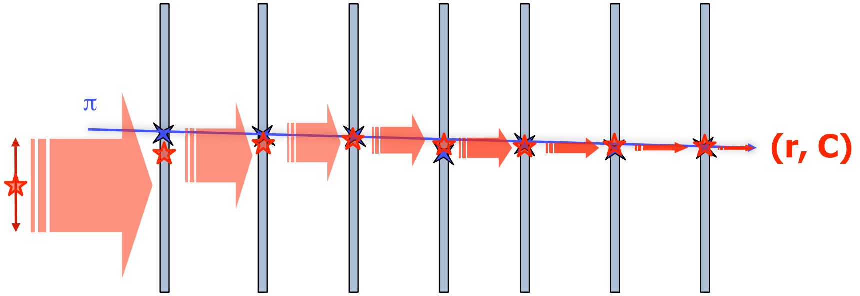
Stages of Event Reconstruction

1 // Track Finder



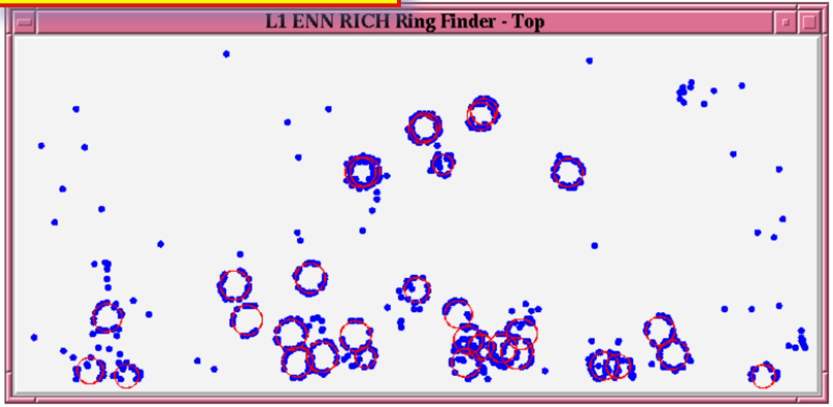
- Conformal Mapping
- Hough Transformation
- Track Following
- **Cellular Automaton**

2 // Track Fitter



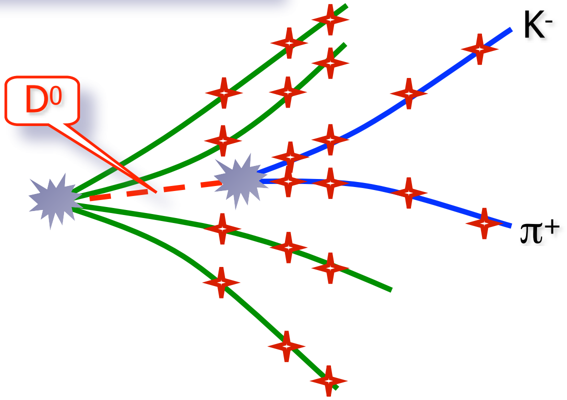
- **Kalman Filter**

3 // Ring Finder (Particle ID)



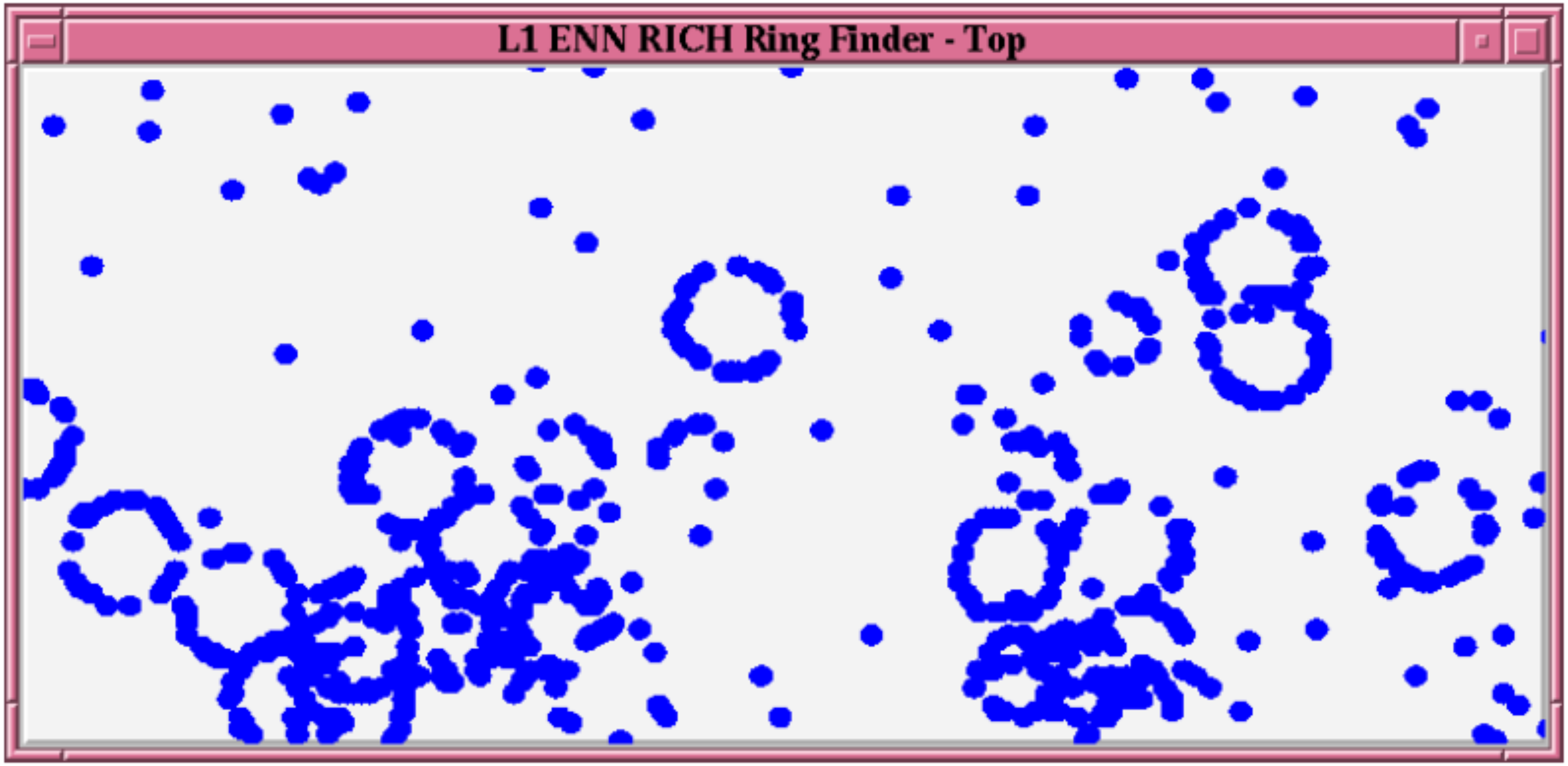
- Hough Transformation
- Elastic Neural Net

4 // Short-Lived Particles Finder



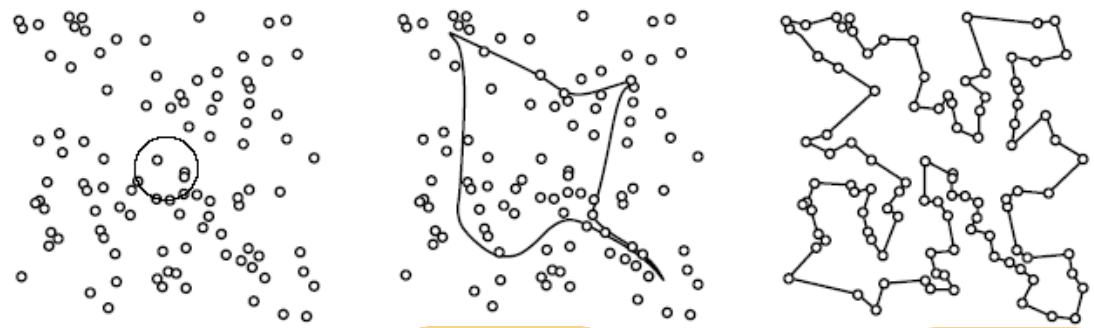
- **Kalman Filter**

Ring-Imaging Cherenkov Detector (RICH)

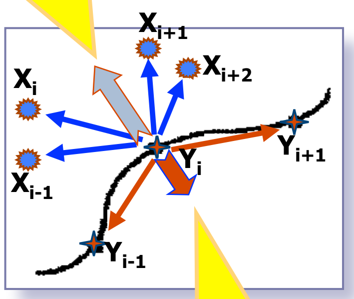


Elastic Neural Net (EN) for TSP

R. Durbin and D. Willshaw, An analogue approach to the travelling salesman problem, Nature, 326 (1987) 689



Attraction



Elasticity

Attraction

Elasticity

Simple

$$E(s_{ia},\vec{y}_a) = \sum_{ia} s_{ia}\cdot|\vec{x}_i-\vec{y}_a|^2 + \gamma\cdot\sum_a |\vec{y}_a-\vec{y}_{a+1}|^2$$

$$\Delta\vec{y}_a = \eta\left[2\sum_i v_{ia}\cdot(\vec{x}_i-\vec{y}_a)+\gamma\cdot(\vec{y}_{a+1}-2\vec{y}_a+\vec{y}_{a-1})\right]$$

Discrete EN

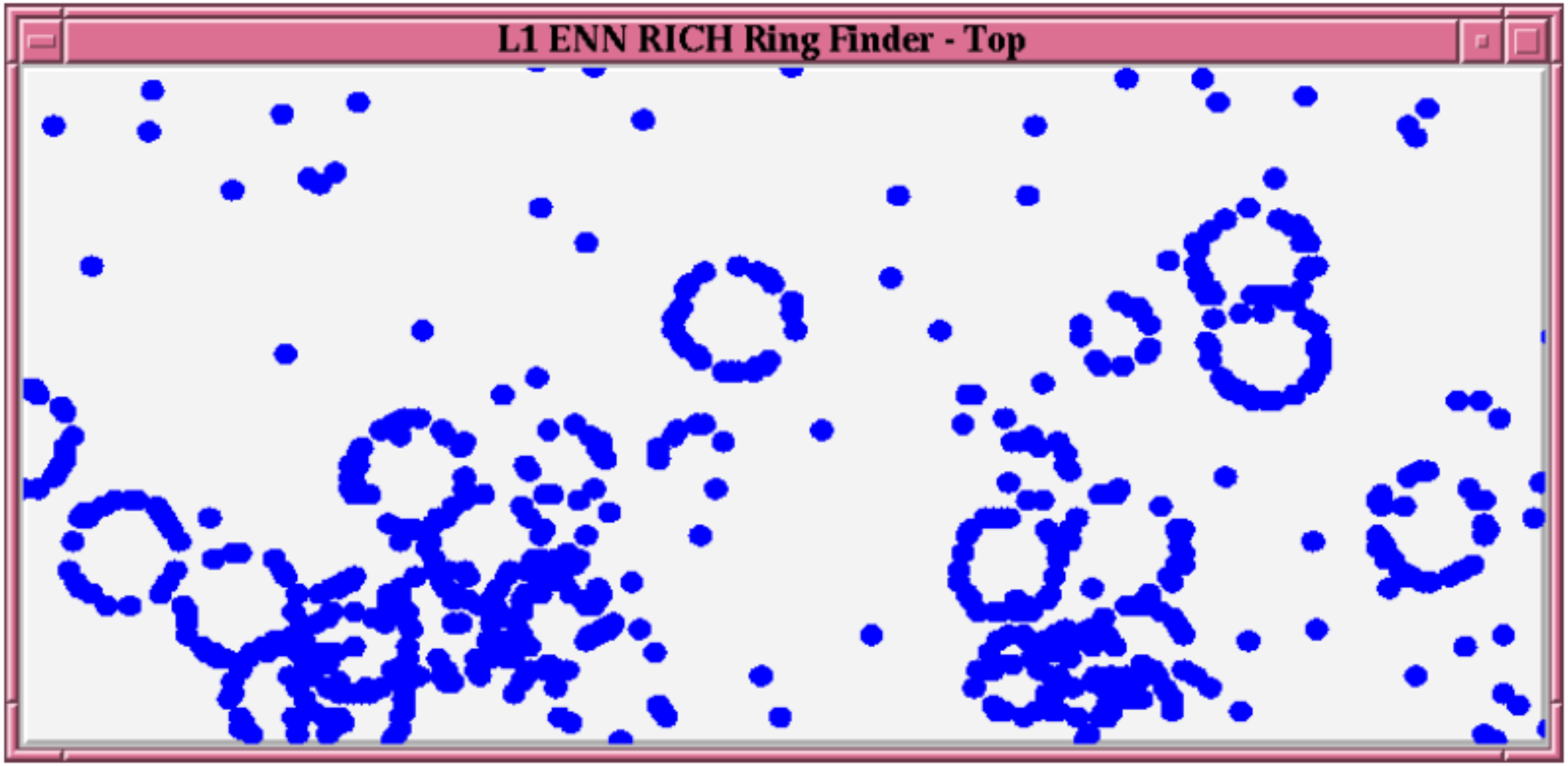
File name	Number of cities	Extra path (%)	Time, ms	Time per city, μs
berlin52	52	0.00	0.98	19
st70	70	4.27	1.27	18
kroA100	100	3.03	1.46	15
lin105	105	0.78	1.84	18
ch130	130	5.59	2.56	20
tsp225	225	5.34	4.36	19
pcb442	442	8.37	12.35	28
pr1002	1002	6.12	24.94	25
pr2392	2392	8.42	58.53	24

(*) Pentium IV/2.4 GHz

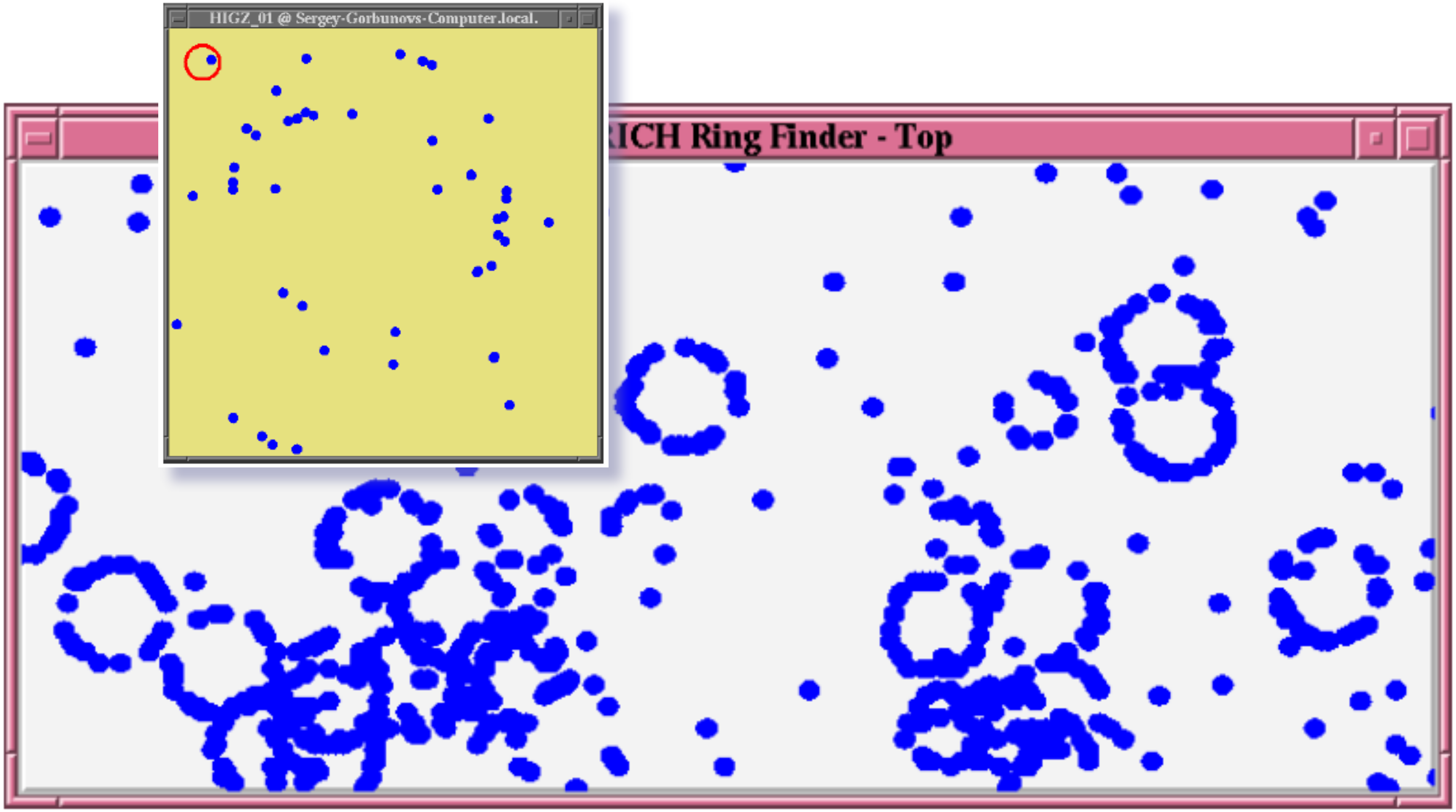
Fast

S. Gorbunov and I. Kisel, Elastic net for standalone RICH ring finding, CBM-SOFT-note-2005-002

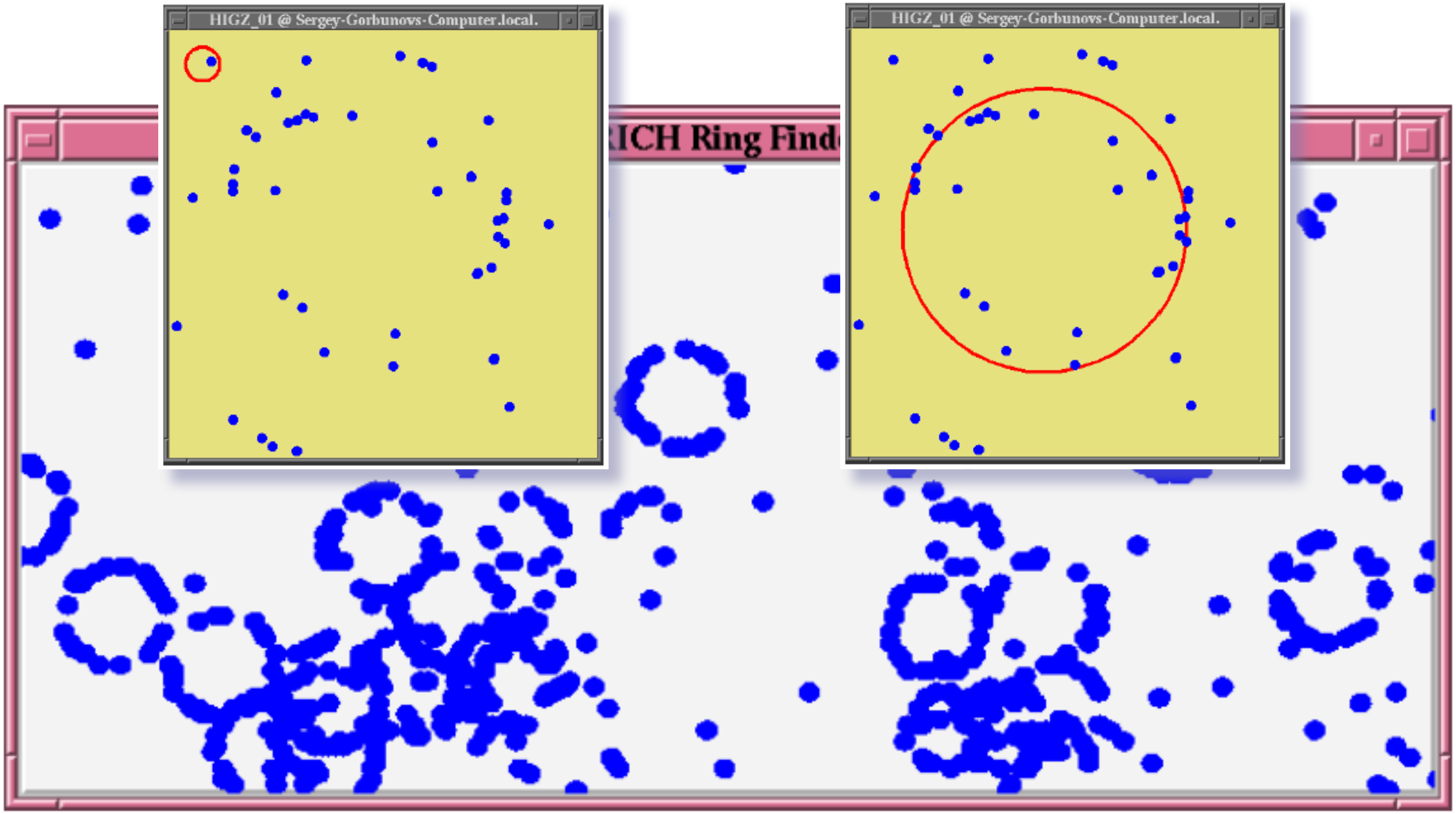
Elastic Neural Net (EN) for RICH



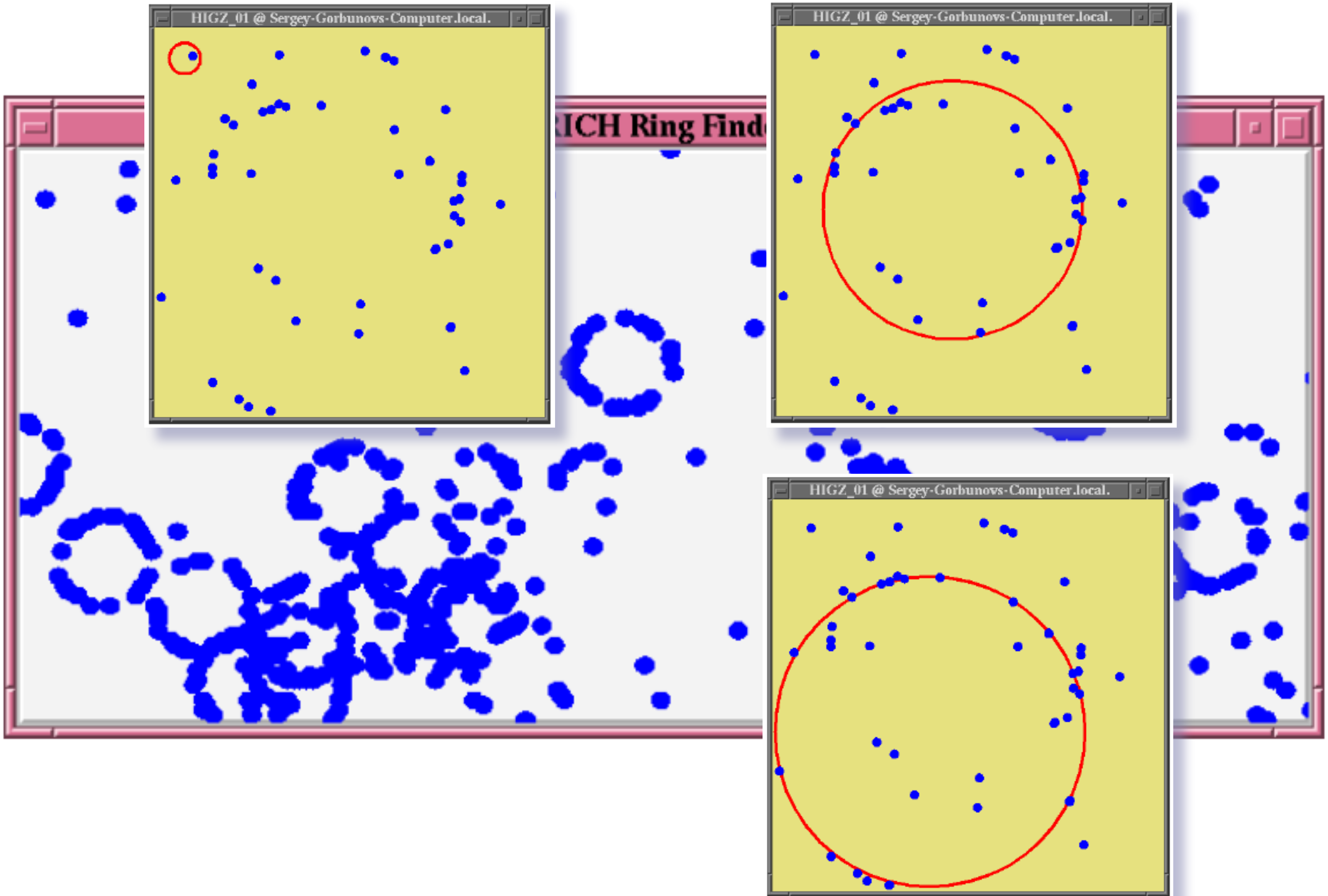
Elastic Neural Net (EN) for RICH



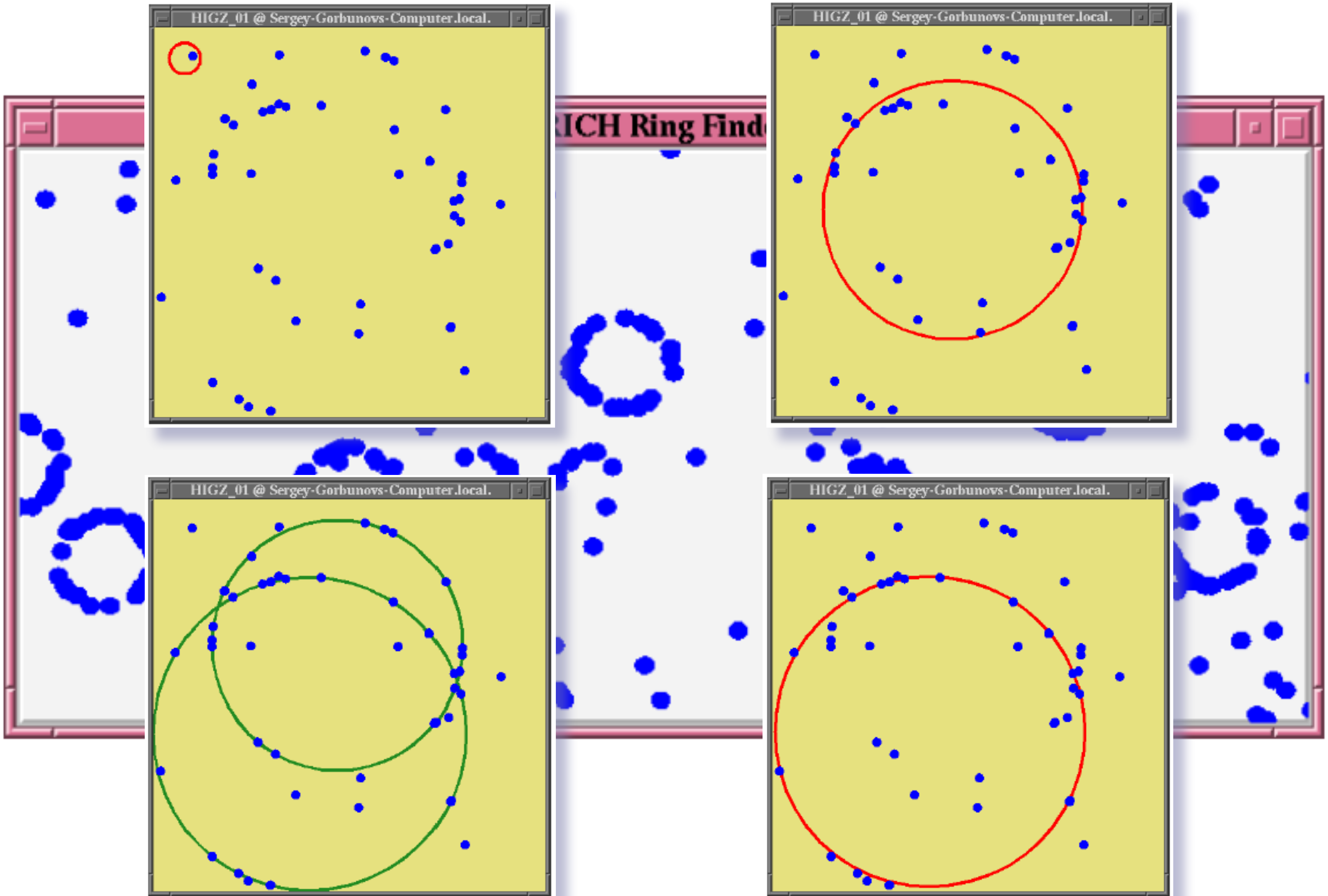
Elastic Neural Net (EN) for RICH



Elastic Neural Net (EN) for RICH

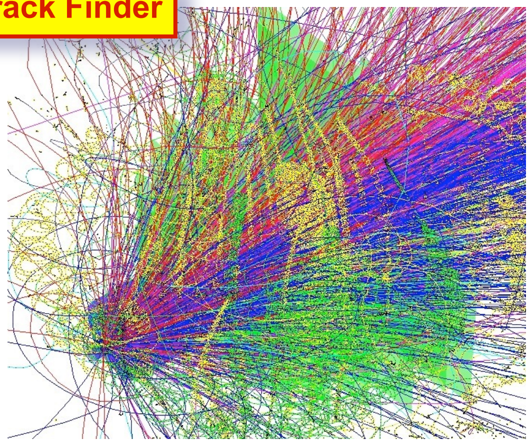


Elastic Neural Net (EN) for RICH



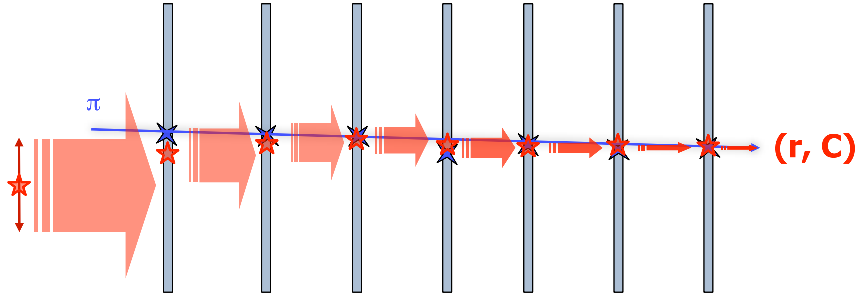
Stages of Event Reconstruction

1 **// Track Finder**



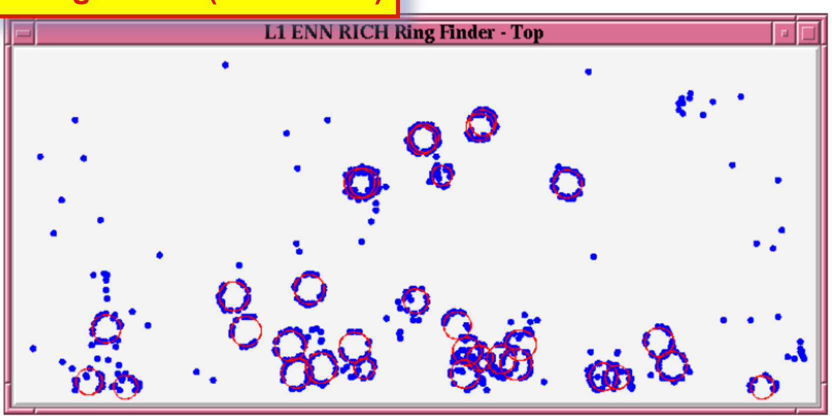
- Conformal Mapping
- Hough Transformation
- Track Following
- **Cellular Automaton**

2 **// Track Fitter**



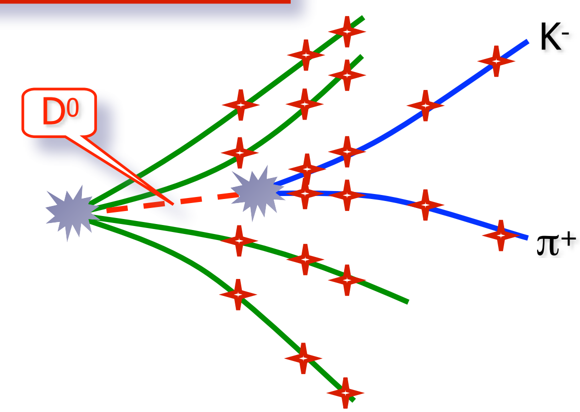
- **Kalman Filter**

3 **// Ring Finder (Particle ID)**



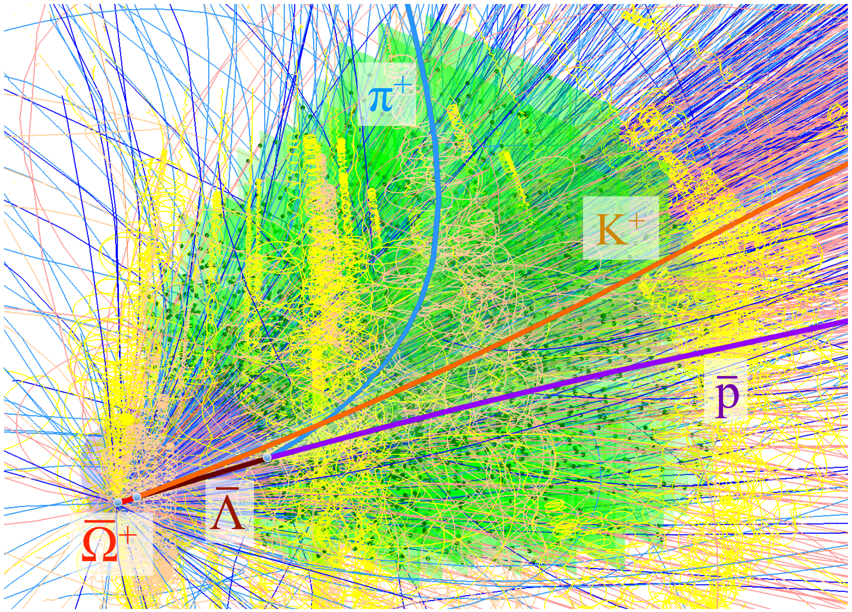
- Hough Transformation
- Elastic Neural Net

4 **// Short-Lived Particles Finder**



- **Kalman Filter**

KF Particle: Reconstruction of short-lived Particles



Simulated AuAu collision at 25 AGeV

$$\bar{\Omega}^{+} \leftarrow \bar{\Lambda}\, K^{+}$$
$$\bar{\Lambda} \rightarrow \bar{p}\, \pi^{+}$$

```
KFParticle Lambda(P, Pi);                      // construct anti Lambda
Lambda.SetMassConstraint(1.1157);              // improve momentum and mass
KFParticle Omega(K, Lambda);                   // construct anti Omega
PV -= (P; Pi; K);                              // clean the primary vertex
PV += Omega;                                   // add Omega to the primary vertex
Omega.SetProductionVertex(PV);                 // Omega is fully fitted
(K; Lambda).SetProductionVertex(Omega);        // K, Lambda are fully fitted
(P; Pi).SetProductionVertex(Lambda);           // p, pi are fully fitted
```

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

State vector

$$\mathbf{C} = \langle \mathbf{r}\mathbf{r}^T \rangle = \begin{bmatrix} \sigma_x^2 & C_{xy} & C_{xz} & C_{xp_x} & C_{xp_y} & C_{xp_z} & C_{xE} \\ C_{xy} & \sigma_y^2 & C_{yz} & C_{yp_x} & C_{yp_y} & C_{yp_z} & C_{yE} \\ C_{xz} & C_{yz} & \sigma_z^2 & C_{zp_x} & C_{zp_y} & C_{zp_z} & C_{zE} \\ C_{xp_x} & C_{yp_x} & C_{zp_x} & \sigma_{p_x}^2 & C_{p_xp_y} & C_{p_xp_z} & C_{p_xE} \\ C_{xp_y} & C_{yp_y} & C_{zp_y} & C_{p_xp_y} & \sigma_{p_y}^2 & C_{p_yp_z} & C_{p_yE} \\ C_{xp_z} & C_{yp_z} & C_{zp_z} & C_{p_xp_z} & C_{p_yp_z} & \sigma_{p_z}^2 & C_{p_zE} \\ C_{xE} & C_{yE} & C_{zE} & C_{p_xE} & C_{p_yE} & C_{p_zE} & \sigma_E^2 \end{bmatrix}$$

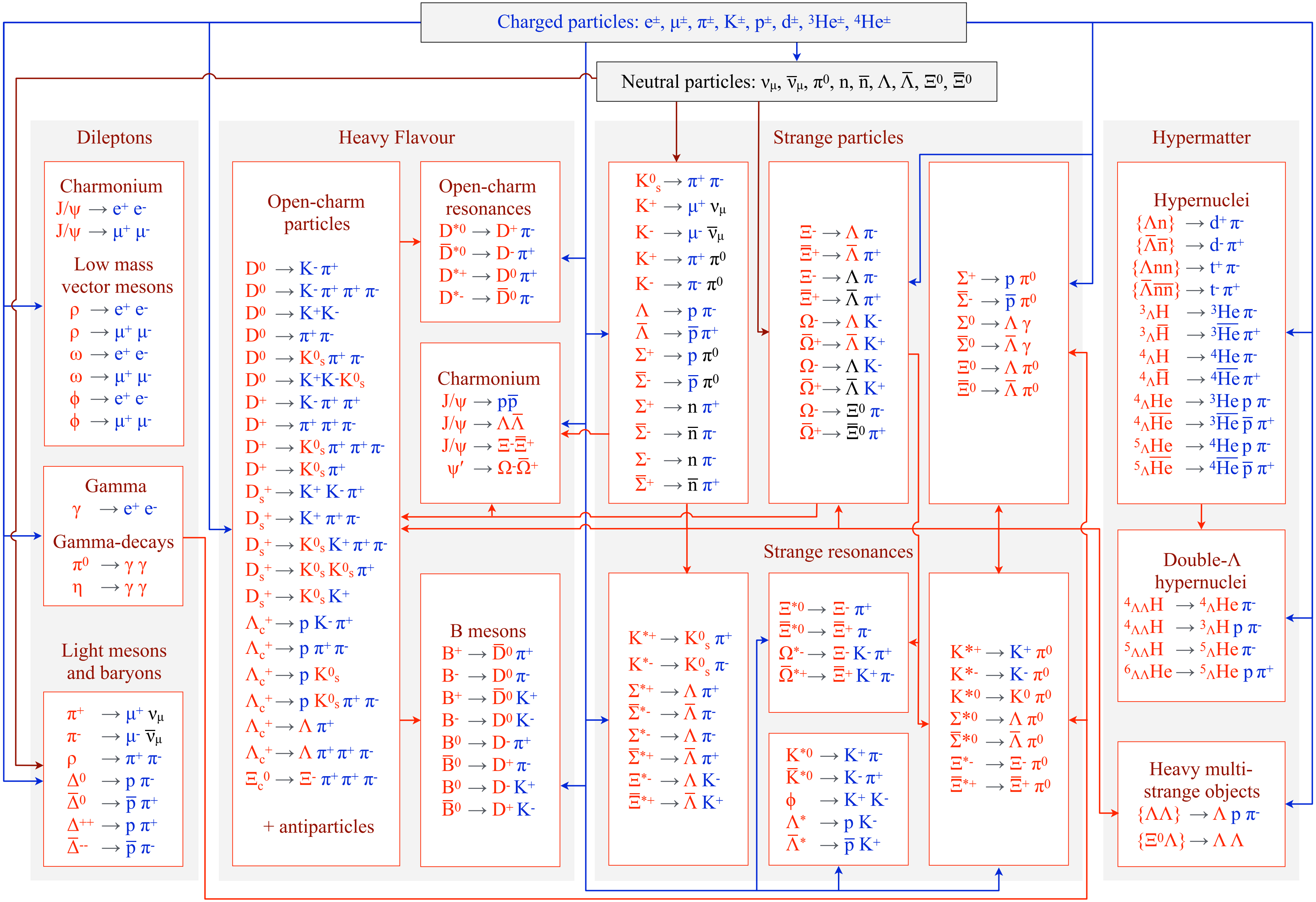
Covariance matrix

Features:

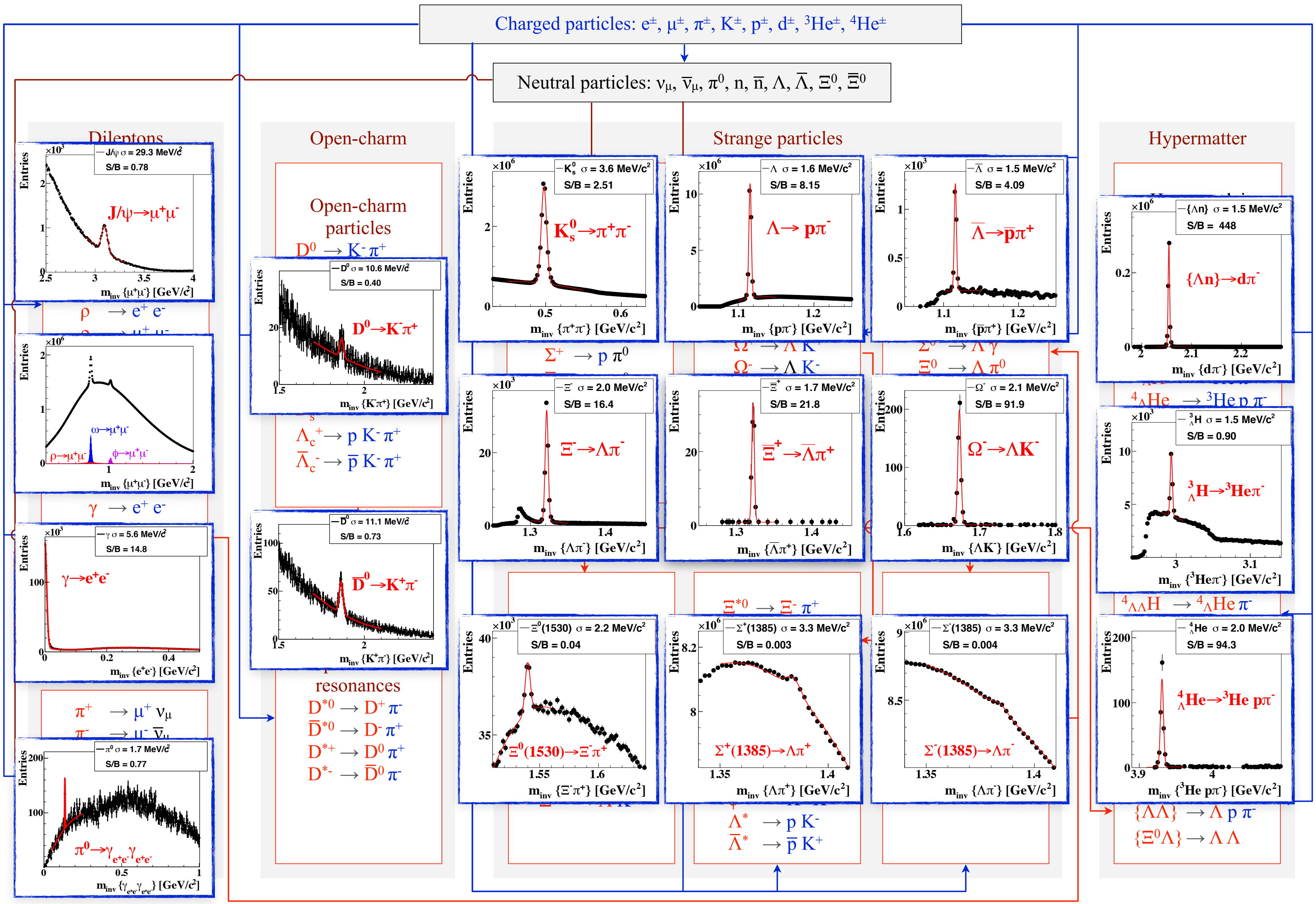
- KF Particle class describes particles by the state vector and the covariance matrix.
- Covariance matrix contains essential information about tracking and detector performance.
- The method for mathematically correct usage of covariance matrices is provided by the KF Particle package based on the Kalman filter (KF).
- Heavy mathematics of KF requires fast and vectorised algorithms.
- Mother and daughter particles are treated in the same way.
- The natural and simple interface allows two reconstruct easily complicated decay chains.
- The package is geometrically independent and can be adapted to different experiments (CBM, ALICE, STAR).

KF Particle provides a simple and very efficient approach to physics analysis

KF Particle Finder for short-lived particles

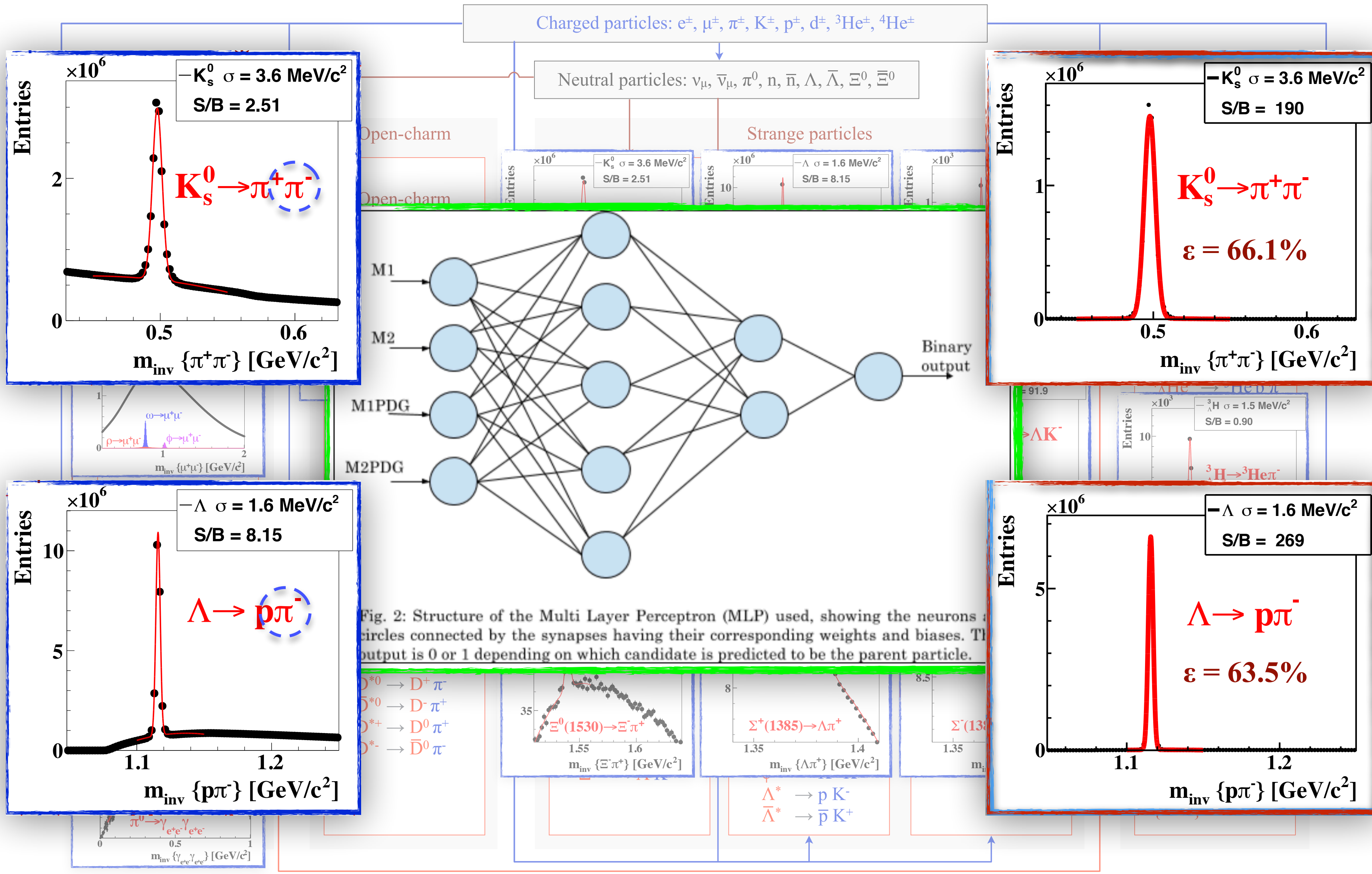


KF Particle Finder for short-lived particles

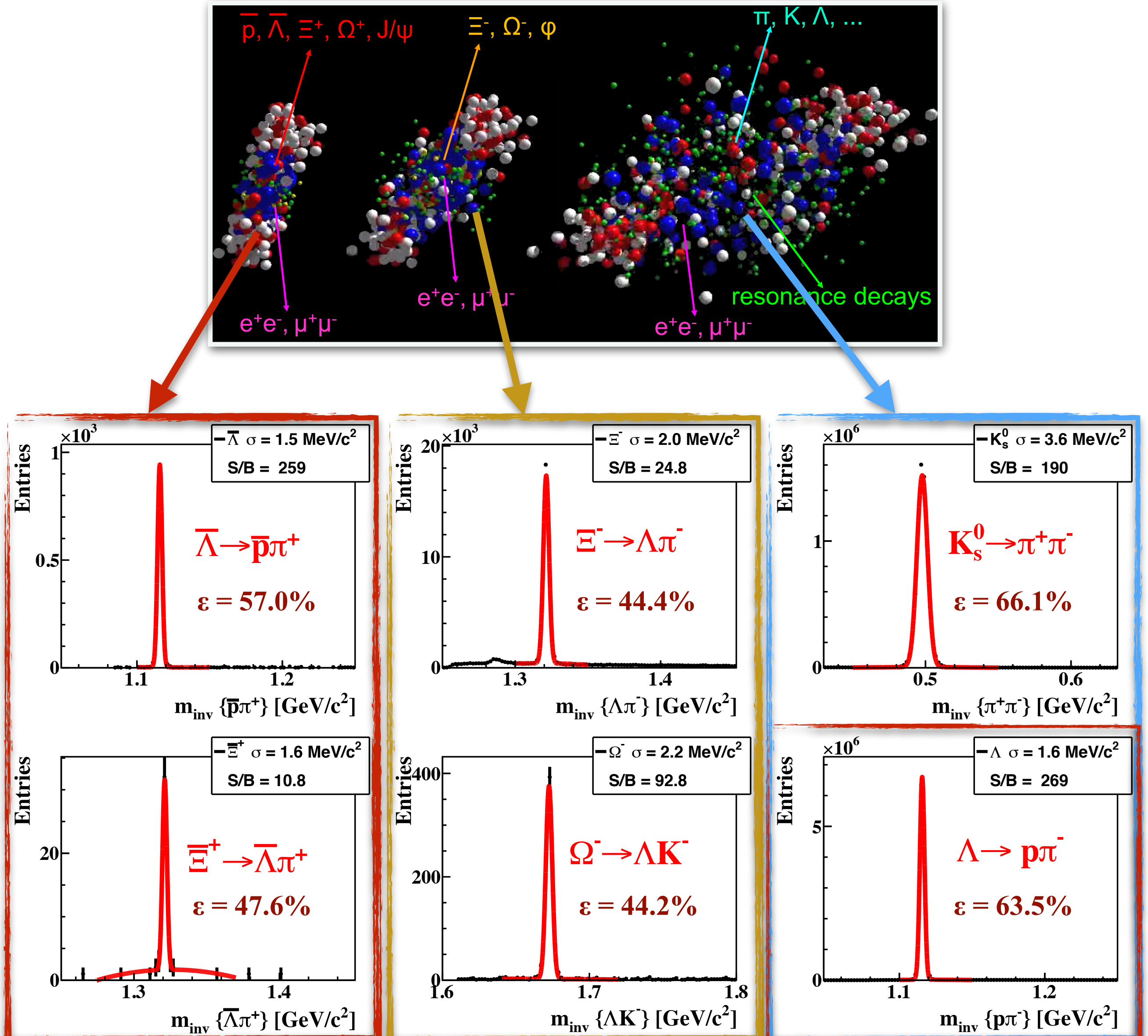


4

Artificial Neural Network

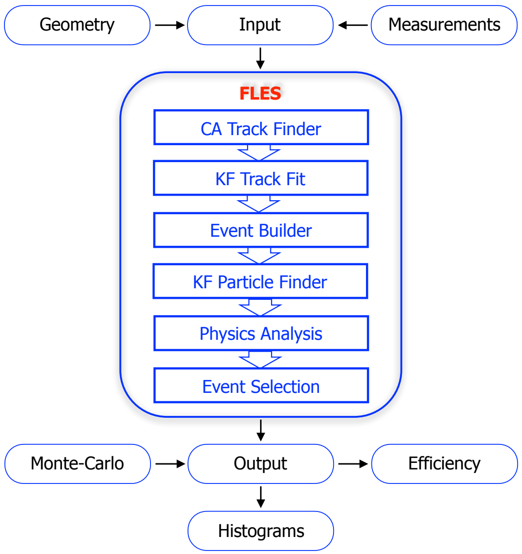


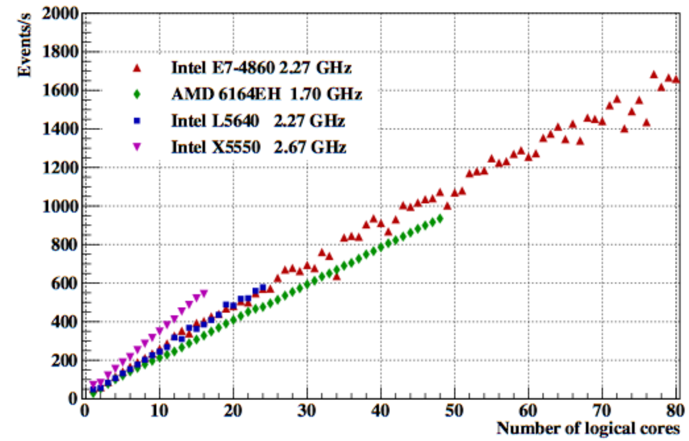
Very Clean Probes of Collision Stages

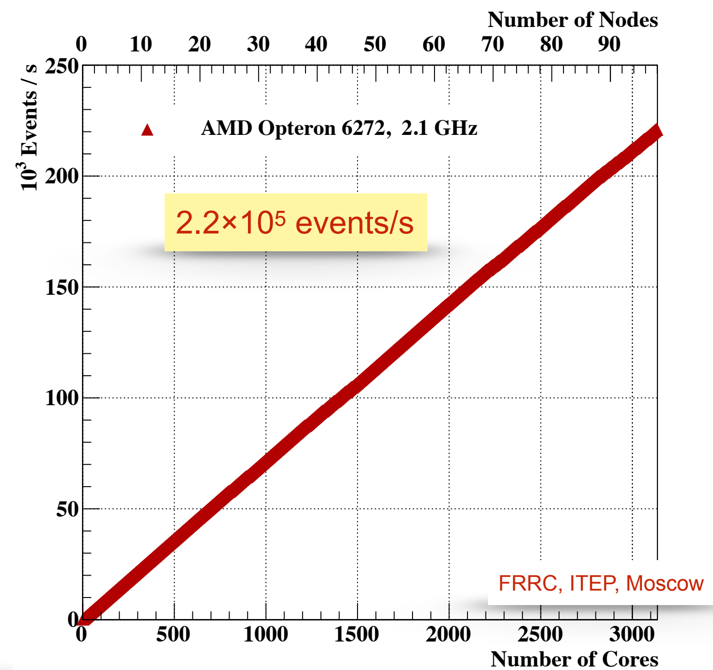


AuAu, 10 AGeV, 3.5M central UrQMD events, **MC PID**

Running FLES on HPC Node/Farm







The FLES package is vectorized, parallelized, portable and scalable up to 3 200 CPU cores

Consolidate Efforts: A Common Reconstruction Package