Event Reconstruction in High Energy Physics Experiments

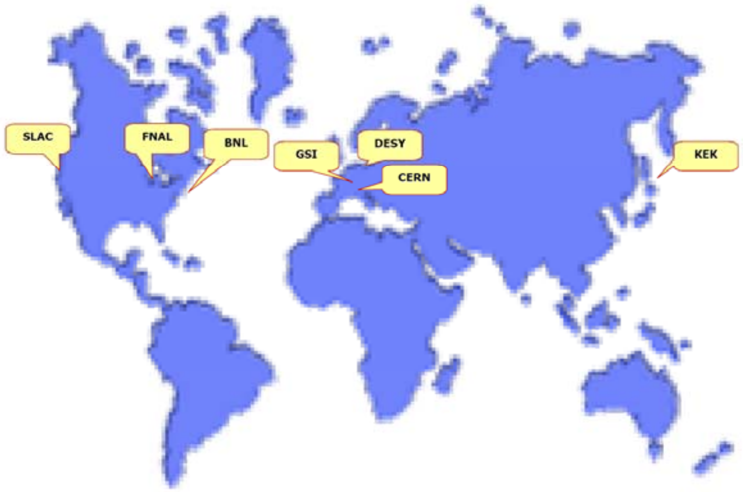
I. Kisel

GSI/Uni-Heidelberg, Germany

CERN, February 07, 2008

HEP Research Centers

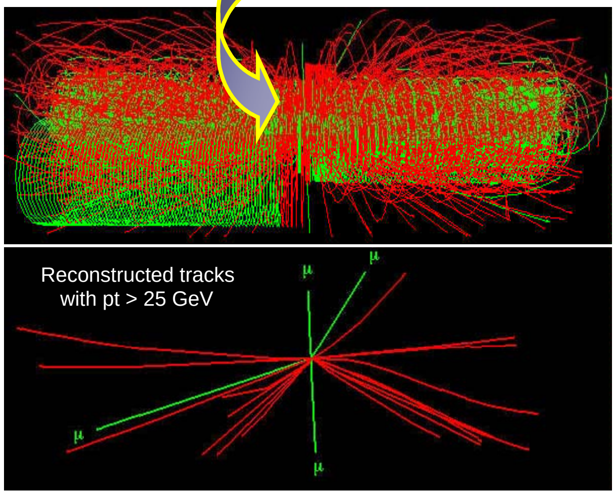
Research Center	Accelerator (GeV)	Experiment	Physics
SLAC, USA	PEP-II, e^- x e^+ (9 x 3.1)	BaBar	B-Physics
Fermilab, USA	Tevatron, p x p (1000 x 1000)	D0	Universal
		CDF	Universal
BNL, USA	RHIC, Heavy Ions	PHENIX	Quark-Gluon-Plasma
		STAR	Quark-Gluon-Plasma
KEK, Japan	KEK-B, e^- x e^+ (8 x 3.5)	BELLE	B-Physics
CERN, Switzerland	LHC, p x p (7000 x 7000)	ATLAS	Universal
		CMS	Universal
		ALICE	Quark-Gluon-Plasma
		LHCb	B-Physics
DESY, Germany	HERA, $e^{+/-}$ x p (27.5 x 920)	ZEUS	Proton-Physics
		H1	Proton-Physics
		HERMES	Spin-Physics
		HERA-B	B-Physics
GSI, Germany	SIS 100/300, p, Heavy Ions	PANDA	Quark-Physics
		CBM	Quark-Gluon-Plasma



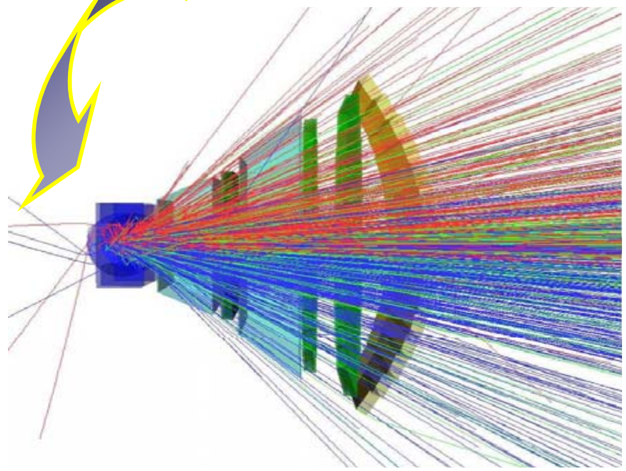
Different experiments for different physics, but with common tracking problems

HEP Experiments: Collider and Fixed-Target

Beam Beam



Beam Target

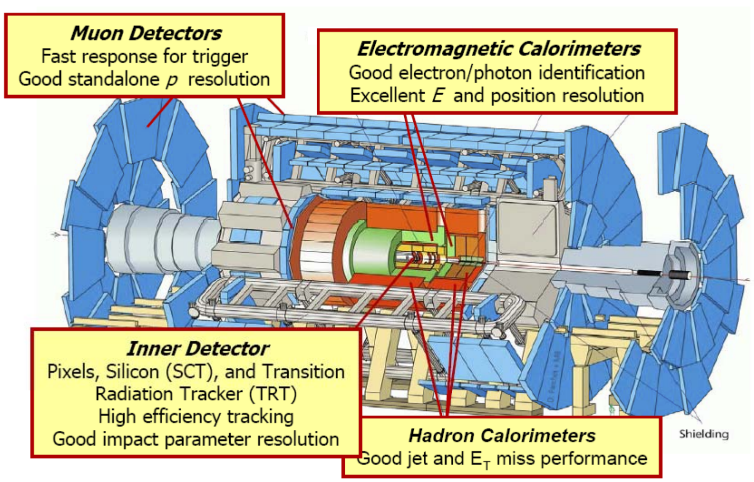


Inelastic collisions
$10^7 - 10^9$

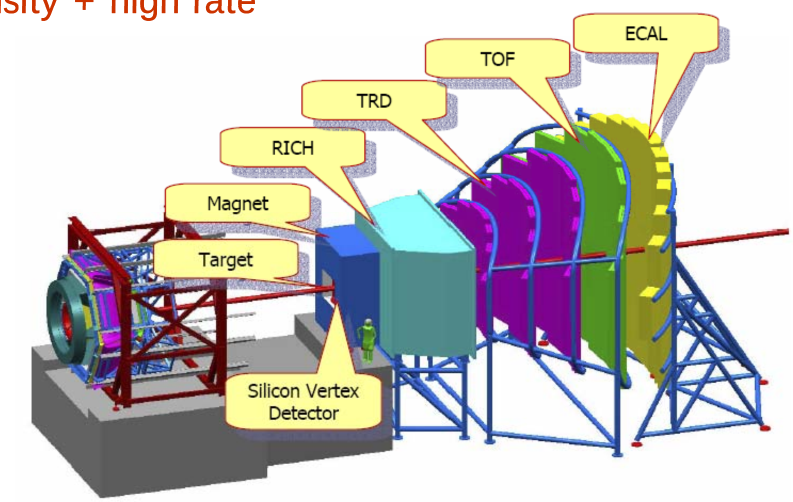
10^{11}

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

High energy = high density + high rate



ATLAS (CERN)

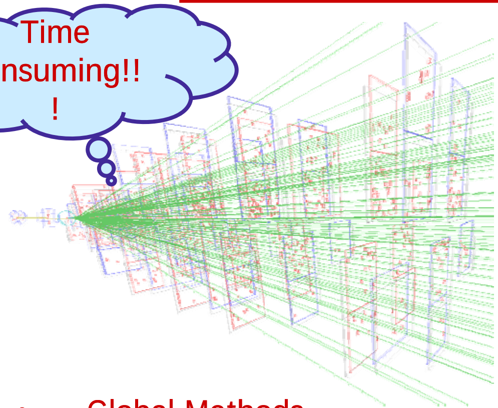


CBM (FAIR/GSI)

Methods for Event Reconstruction

1 Track finding

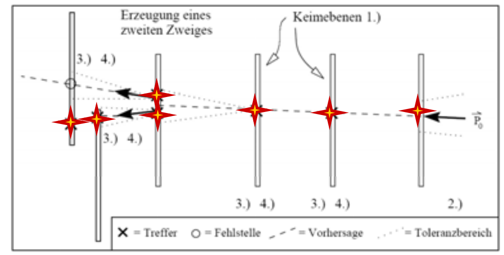
Time consuming!!!



Combinatorics
+ Precision
= Speed ?

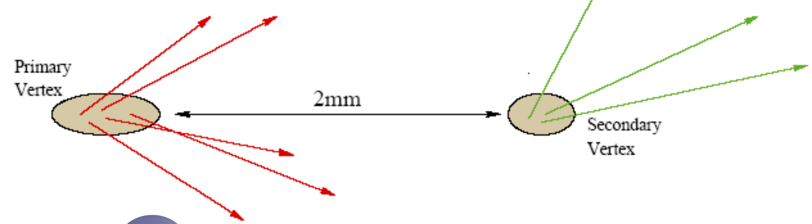
2 Track fitting

Kalman Filter



3 Vertex finding/fitting

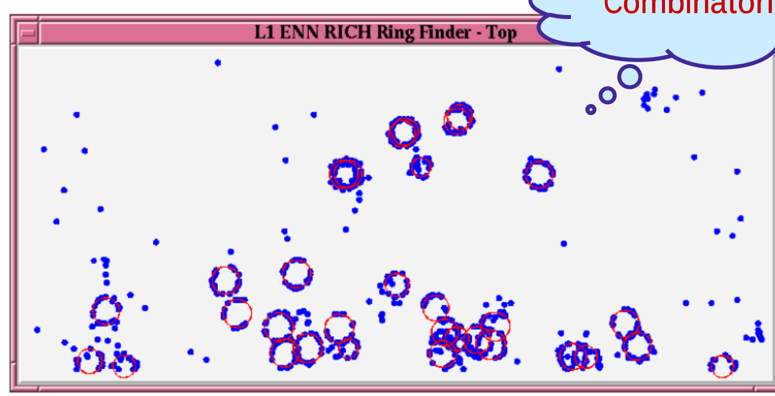
Kalman Filter



- Global Methods
  - all hits are treated equivalently
  - typical methods:
    - Conformal Mapping_
    - Histogramming
    - Hough Transformation
- Local Methods
  - sequential selection of candidates
  - typical methods:
    - Track following
    - Kalman Filter
- Neural Networks
  - combine local and global relations
  - typical methods:
    - Perceptron
    - Hopfield network
    - Cellular Automaton
    - Elastic Net

4 PID: Ring finding

Combinatorics



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

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

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

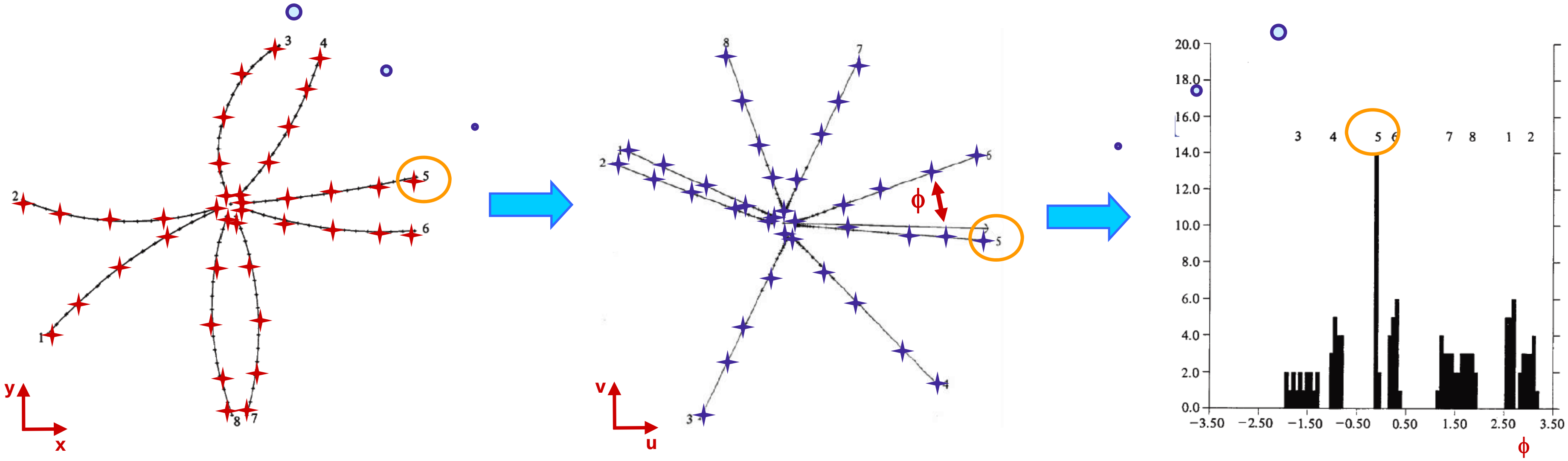
Histogram:

Collect a histogram of azimuth angles ϕ

Find peaks in the histogram

Collect hits into tracks

Fast

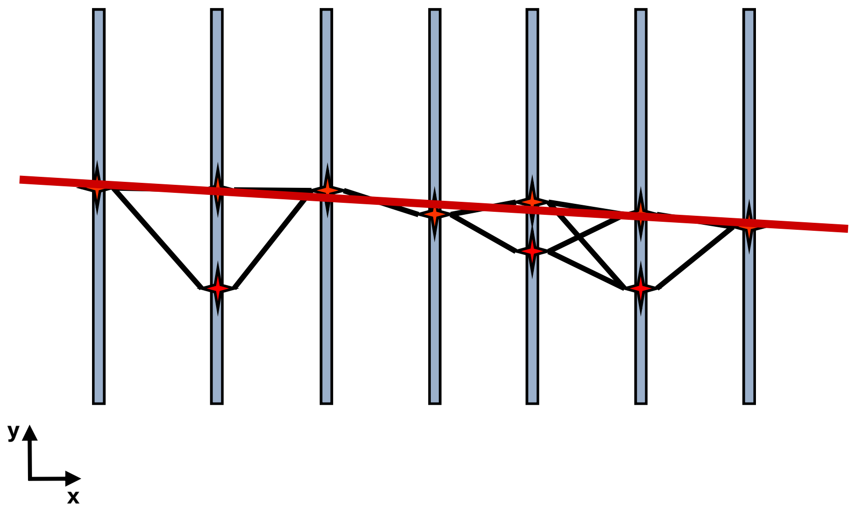


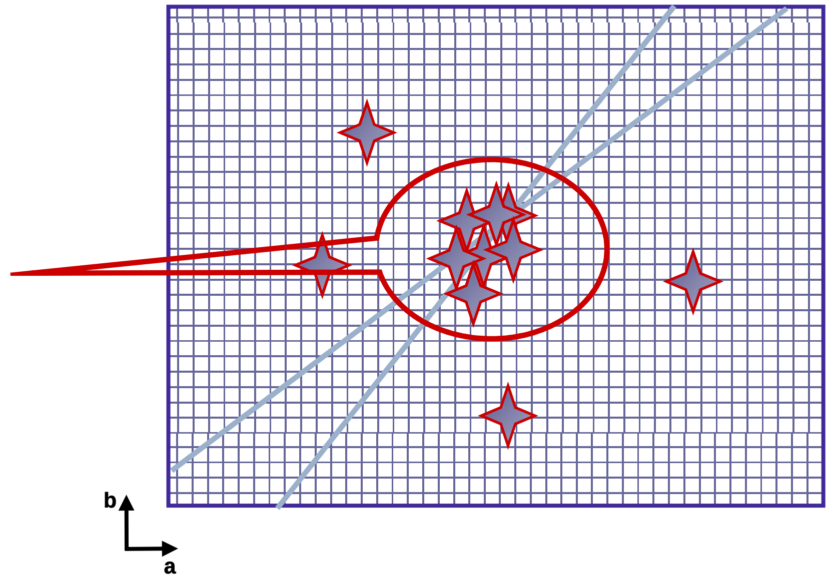
Useful implemented in hardware and for very simple event topologies

Global Methods: Hough Transformation

Measurement Space → **Parameter Space**

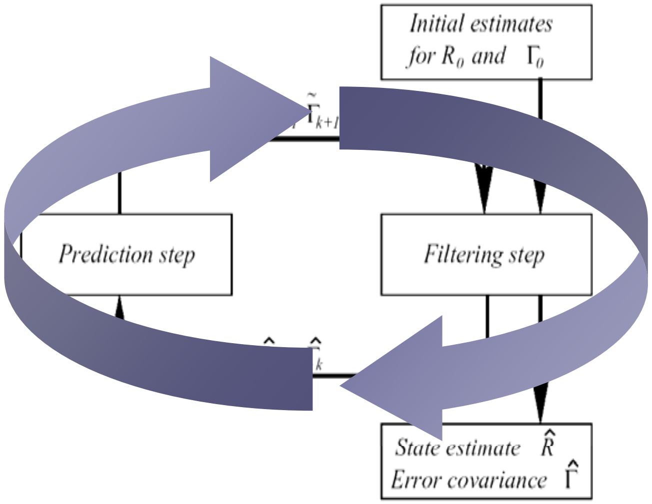
$$y = a*x + b \quad \Rightarrow \quad b = -x*a + y$$

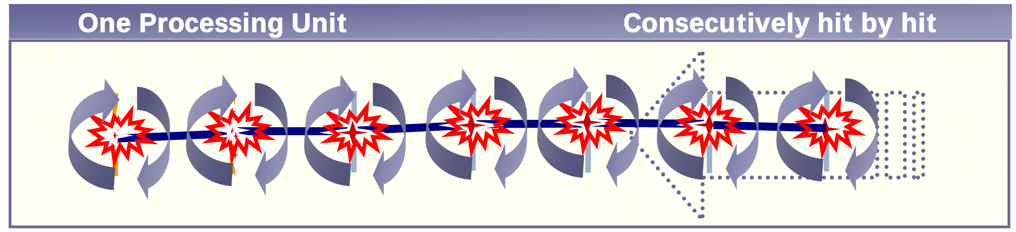


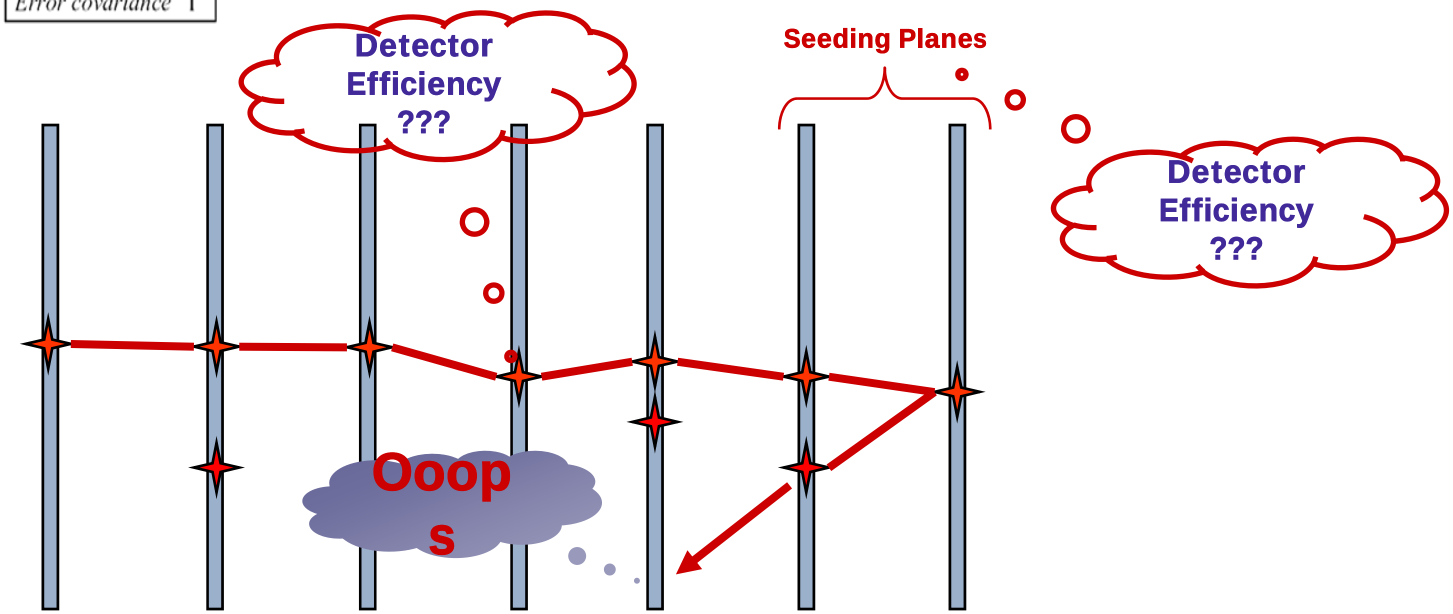


Useful implemented in hardware and for simple event and trigger topologies

Local Methods: Kalman Filter for Track Finding





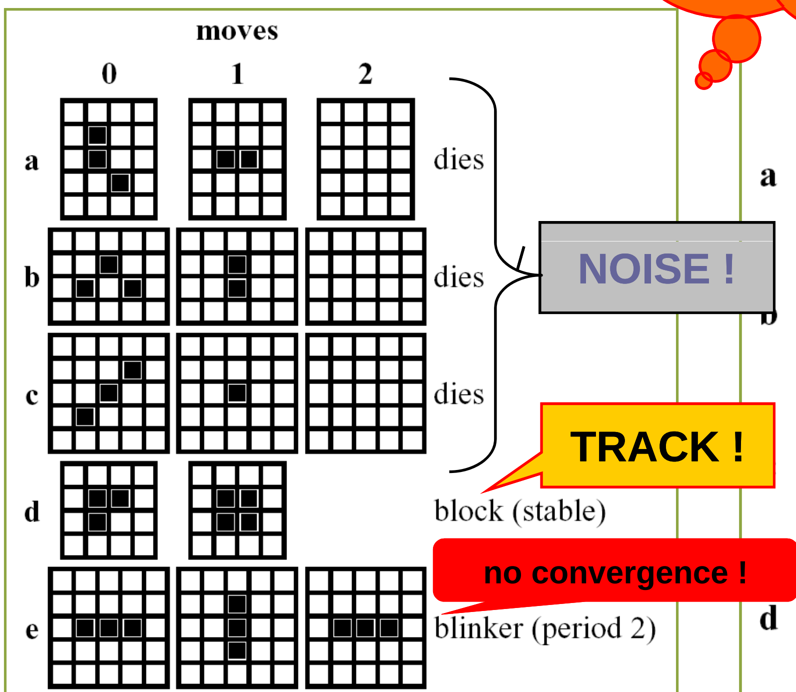


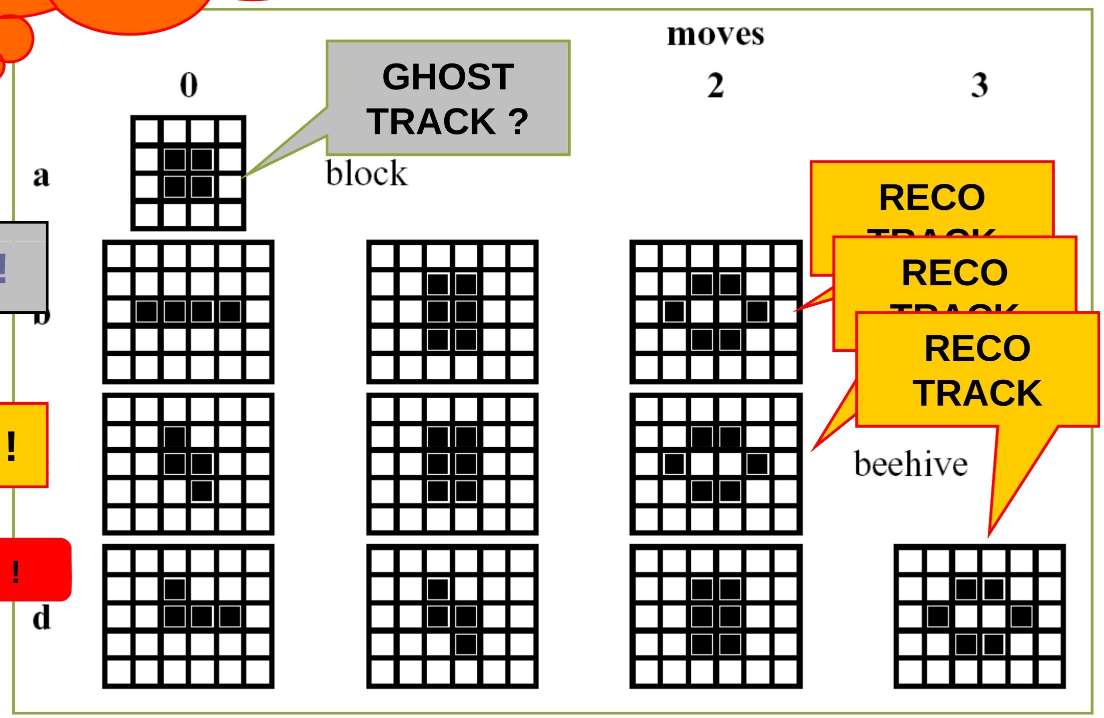
Useful for final track fitting and for Monte Carlo analysis of an experiment

M. Gardner, **Scientific American, 223** (October 1970), 120-123

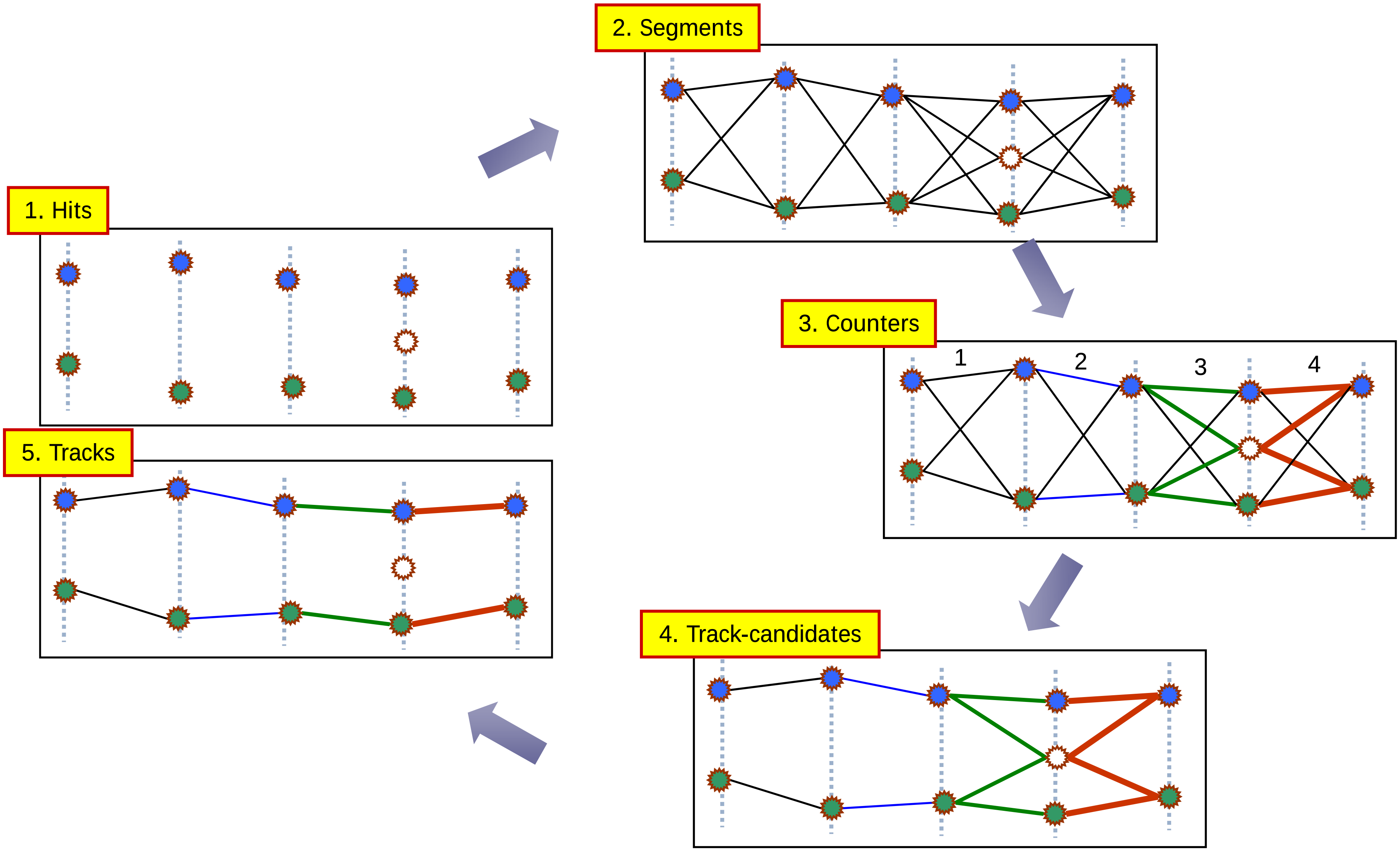
Each **cell** has 8 neighboring cells, 4 adjacent orthogonally, 4 adjacent diagonally. The **rules** are:
Survivals. Every counter with 2 or 3 neighboring counters survives for the next generation.
Deaths. Each counter with 4 or more neighbor… …pulation. Every counter with 1 neighbor or none dies from isolation.
Births. Each empty cell adjacent to …
It is important to understand that all … *…eously*.

TRACKING !





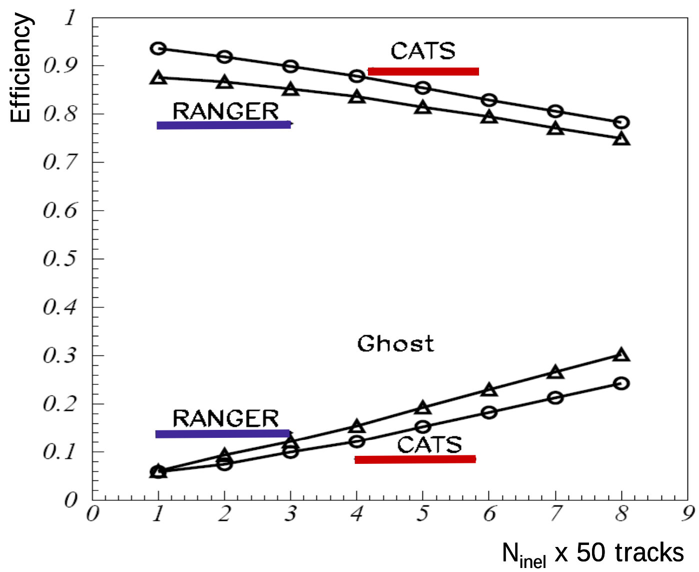
Neural Networks: Cellular Automaton – Animation



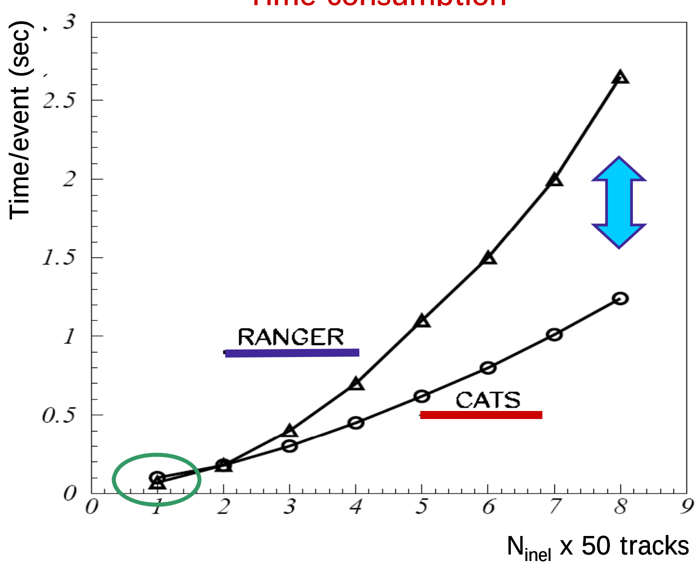
Useful for analysis of experiments with real data

Competition CATS(CA) / RANGER(KF) / TEMA(HT) (HERA-B, DESY)

Tracking efficiency



Time consumption



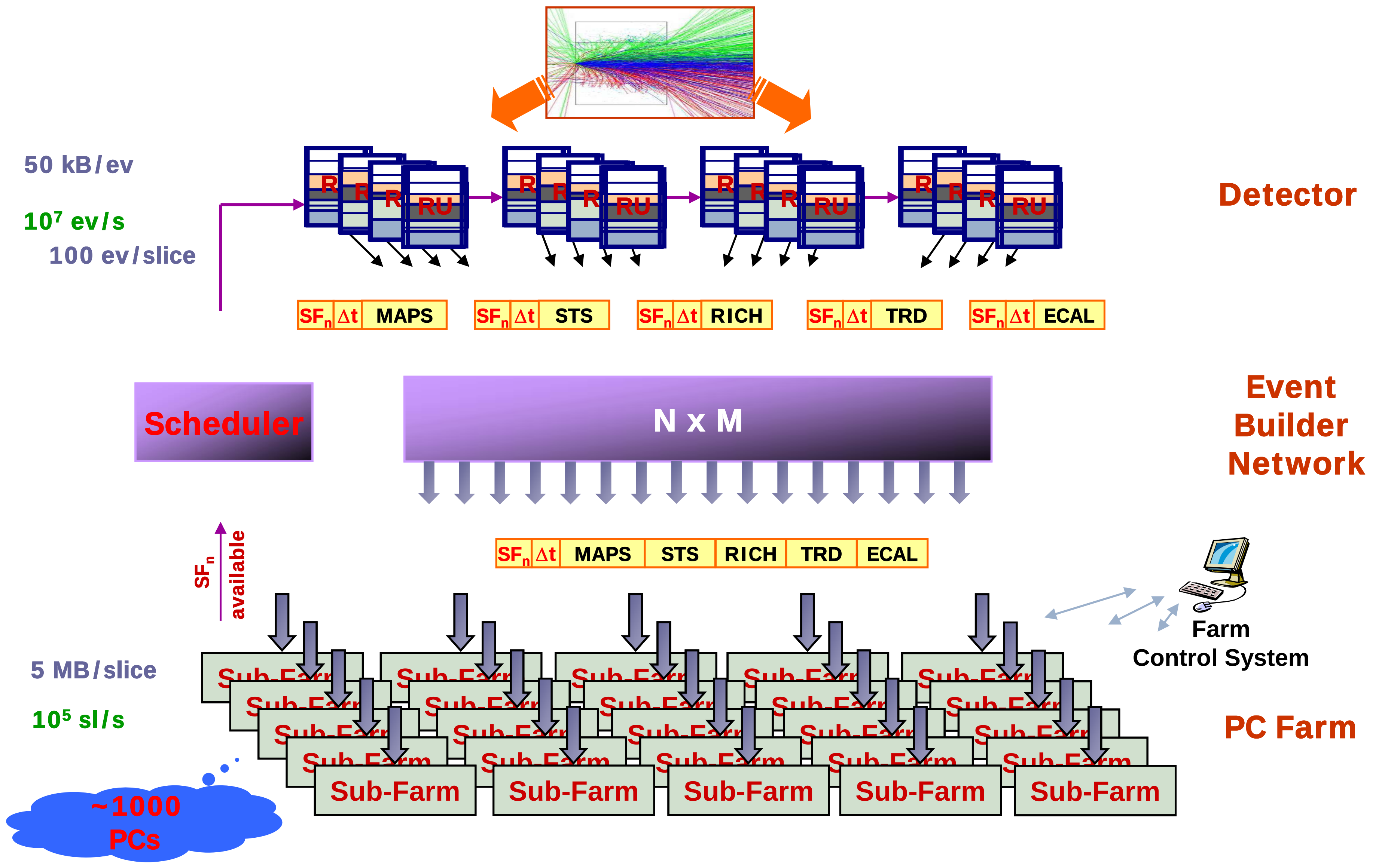
Tracking quality

Resolutions, pulls P and mean length of reconstructed primary tracks.

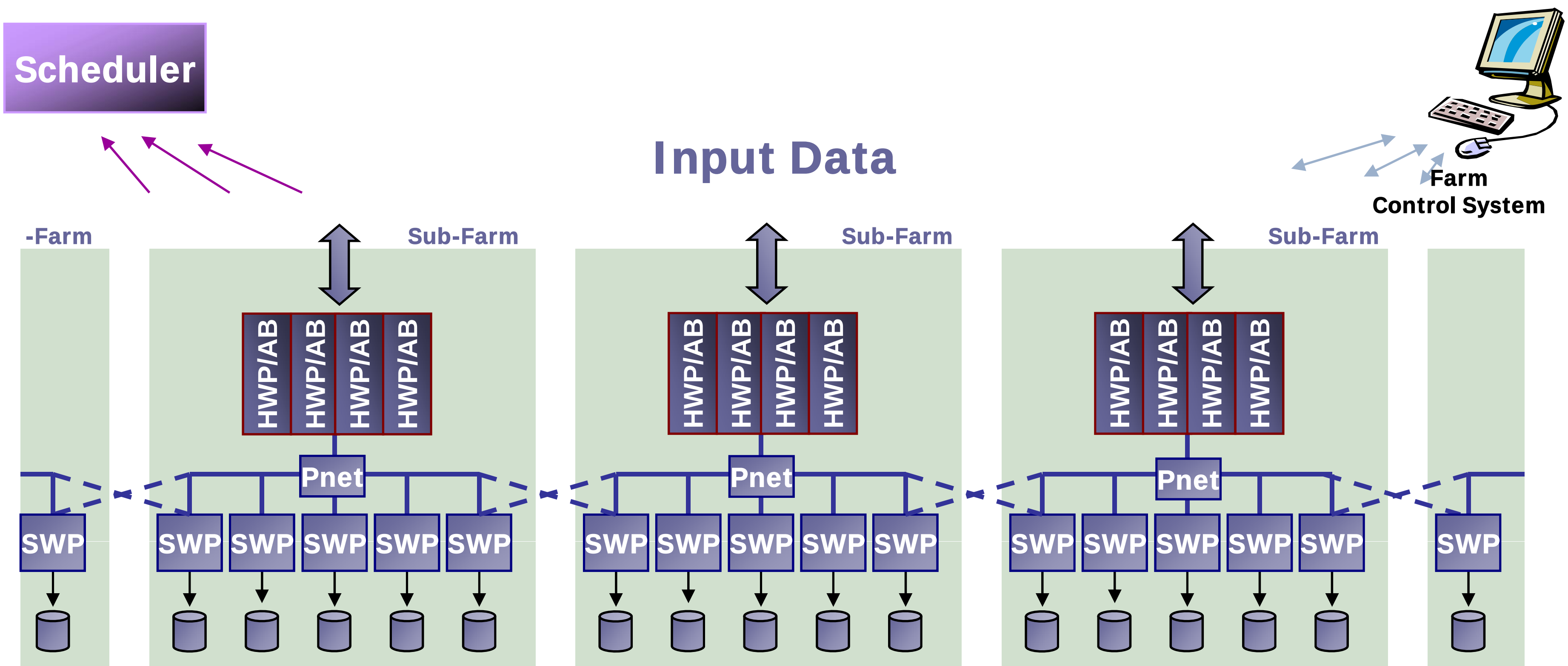
	CATS		RANGER		TEMA	
Resolutions	OTR	ITR	OTR	ITR	OTR	ITR
x, μm	246	93	322	91	291	98
y, mm	3.7	1.4	5.0	1.4	4.1	1.4
t_x, mrad	0.62	0.24	0.71	0.24	0.76	0.26
t_y, mrad	4.73	1.79	6.96	1.79	5.39	1.87
Pulls						
$P(x)$	1.59	1.11	1.37	1.10	1.45	1.06
$P(y)$	1.52	0.98	1.25	1.11	1.81	1.16
$P(t_x)$	1.16	0.93	1.25	0.89	1.18	1.15
$P(t_y)$	1.53	0.99	1.39	1.15	1.92	1.23
Hits/track	31	23	26	21	31	21

The reconstruction package **CATS** based on the Cellular Automaton for track finding and the Kalman Filter for track fitting outperforms alternative packages (SUSi, HOLMES, L2Sili, OSCAR, RANGER, TEMA) based on traditional methods in efficiency, accuracy and speed

Data Acquisition System



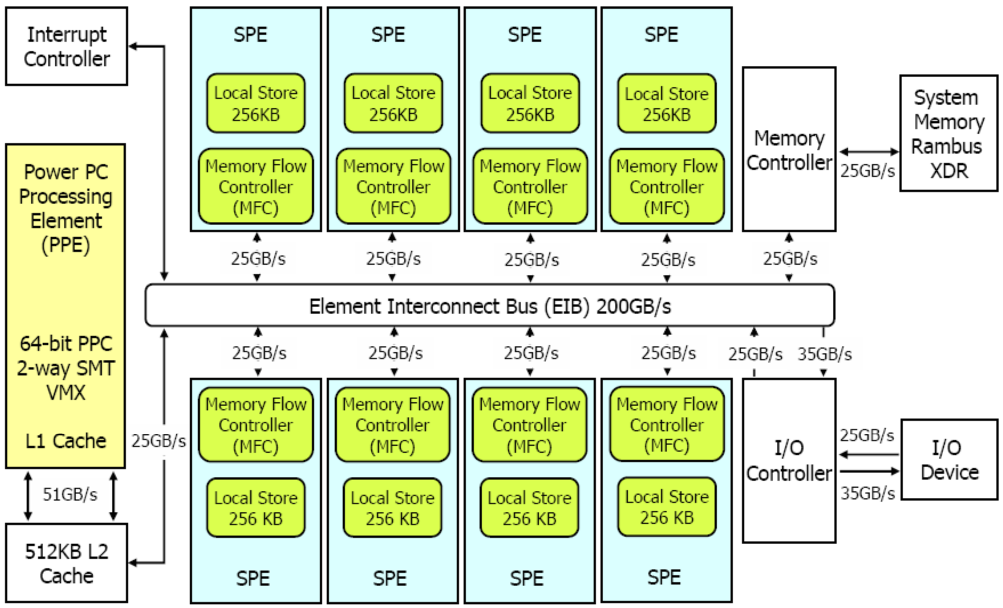
CBM: PC Sub-Farm



ready ⇒	HLT	C++, Framework, GEANT
started ⇒	L1 CPU	C++, Framework, GEANT
future ⇒	L1 FPGA	C++, SystemC, SystemCrafter, VHDL

Cell Processor: Supercomputer-on-a-Chip

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 *

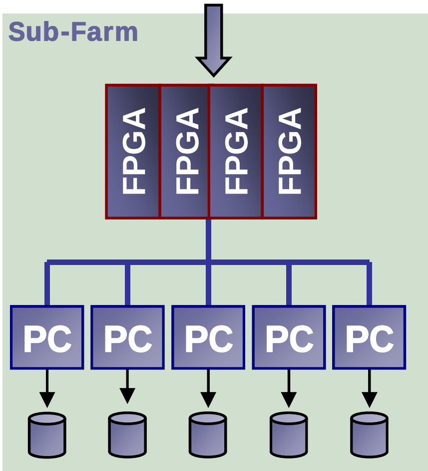
** At clock speed of 3.2GHz*

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

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

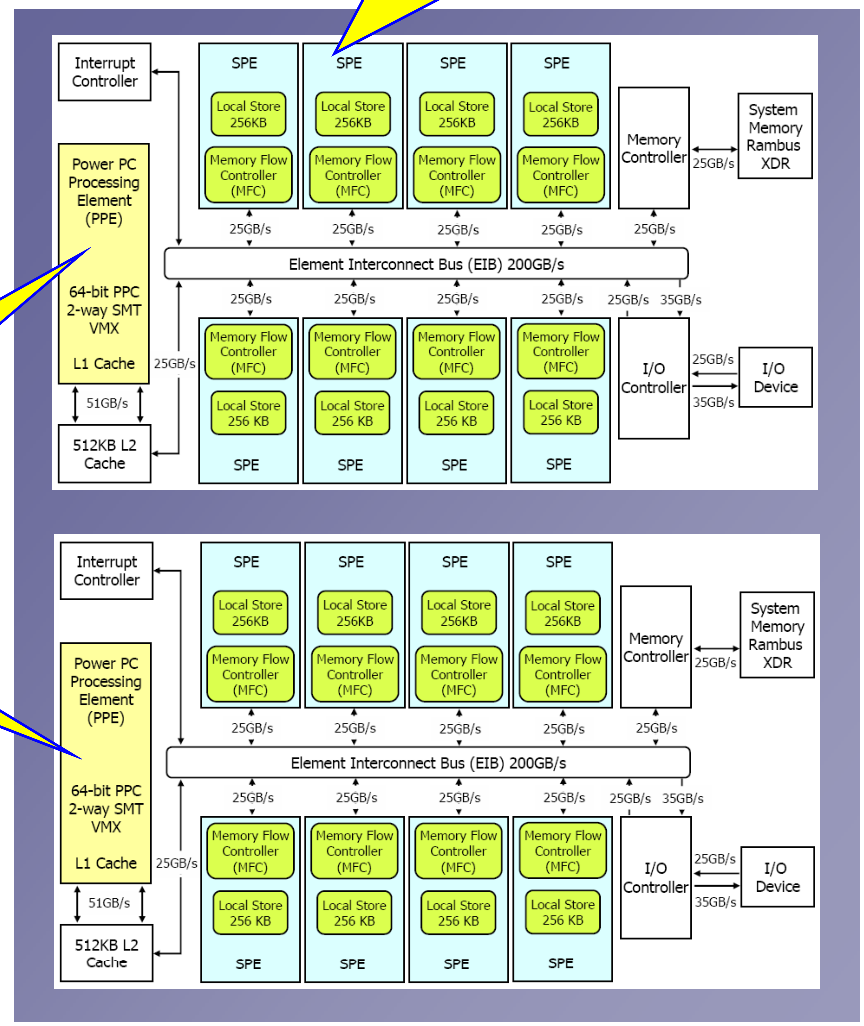
Cell Blade as Sub-Farm

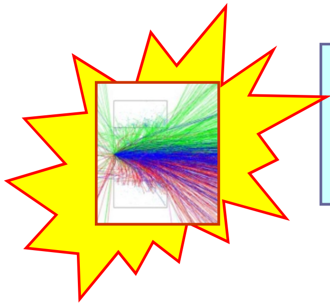


Tracking and Vertexing Units

Sub-Farm
Management Unit

Sub-Farm
Decision/Selection Unit

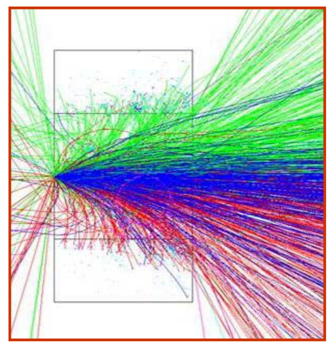


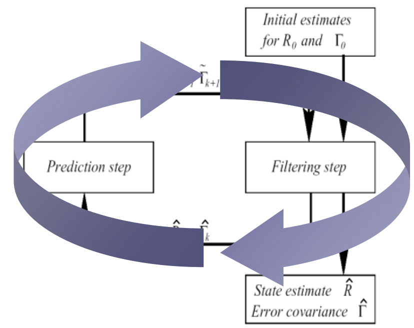


On-line Selection (Trigger)

Sub-Farm

... Cell, Cell, Cell, Cell ...
... Cell, Cell, Cell, Cell ...
... Cell, Cell, Cell, Cell ...





1. Distribution of Data

Sub-Farm Demonstrator

2. Track Finding

Cellular Automaton

3. Track Fit

Kalman Filter
Intel, AMD and Cell

Development

SIMDized Kalman Filter Track Fit

ARTICLE IN PRESS

COMPHY:3445

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Computer Physics Communications ••• (••••) •••–•••

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

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.

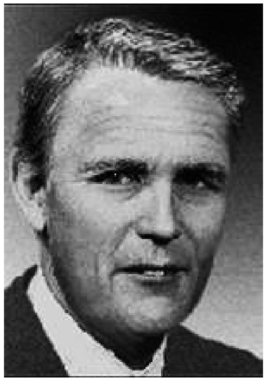
PACS: 02.60.Pn; 02.70.-c; 07.05.-t; 07.05.Bx; 07.05.Kf

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

The Kalman Filter

The Kalman filter is a recursive algorithm which estimates the state of a dynamic system from a series of incomplete and noisy measurements.

The filter was developed in papers by Swerling (1958), Kalman (1960), and Kalman and Bucy (1961).

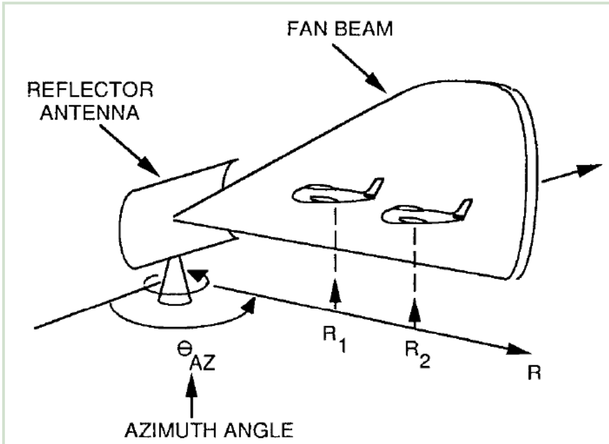


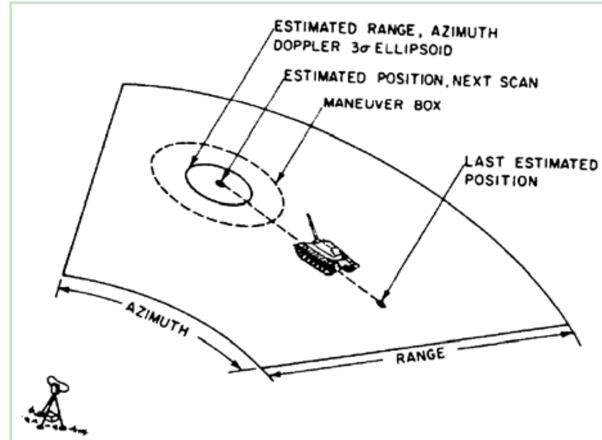
The filter is named after Rudolf E. Kalman.

An example of an application would be to provide accurate continuously-updated information about the position and velocity of an object given only a sequence of observations about its position, each of which includes some error.

It is used in a wide range of engineering applications from radar to computer vision.

A wide variety of Kalman filters have now been developed, from Kalman's original formulation, now called the *simple* Kalman filter, to *extended* filter, the *information* filter and a variety of *square-root* filters.





In a radar application, where one is interested in following a target, information about the location, speed, and acceleration of the target is measured at different moments in time with corruption by noise.

State vector

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

position (x, y, z) velocity (v_x, v_y, v_z)

Covariance matrix

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

error of x

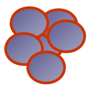
Apollo Flight Journal

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.

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.

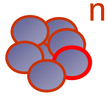
mean value over n measurements



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

previous estimation

new measurement

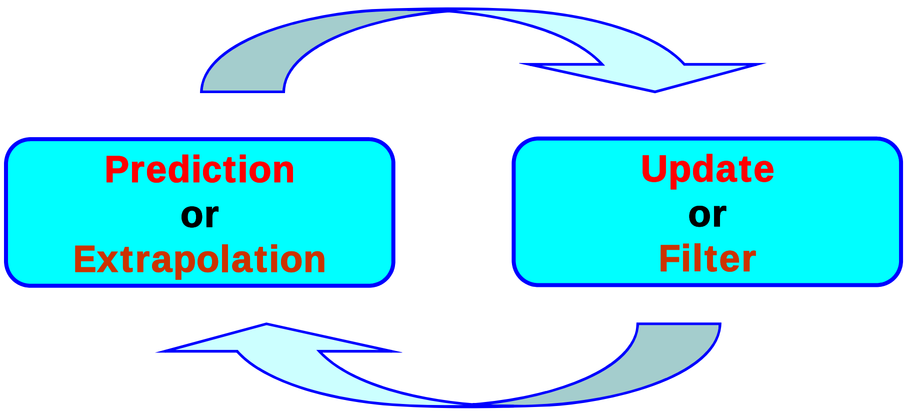


$$\mu_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{n}{n+1}\left(\frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}x_{n+1}\right) = \frac{n}{n+1}\mu_n + \frac{1}{n+1}x_{n+1} = \mu_n + \frac{1}{n+1}(x_{n+1} - \mu_n)$$

mean value over n+1 measurements

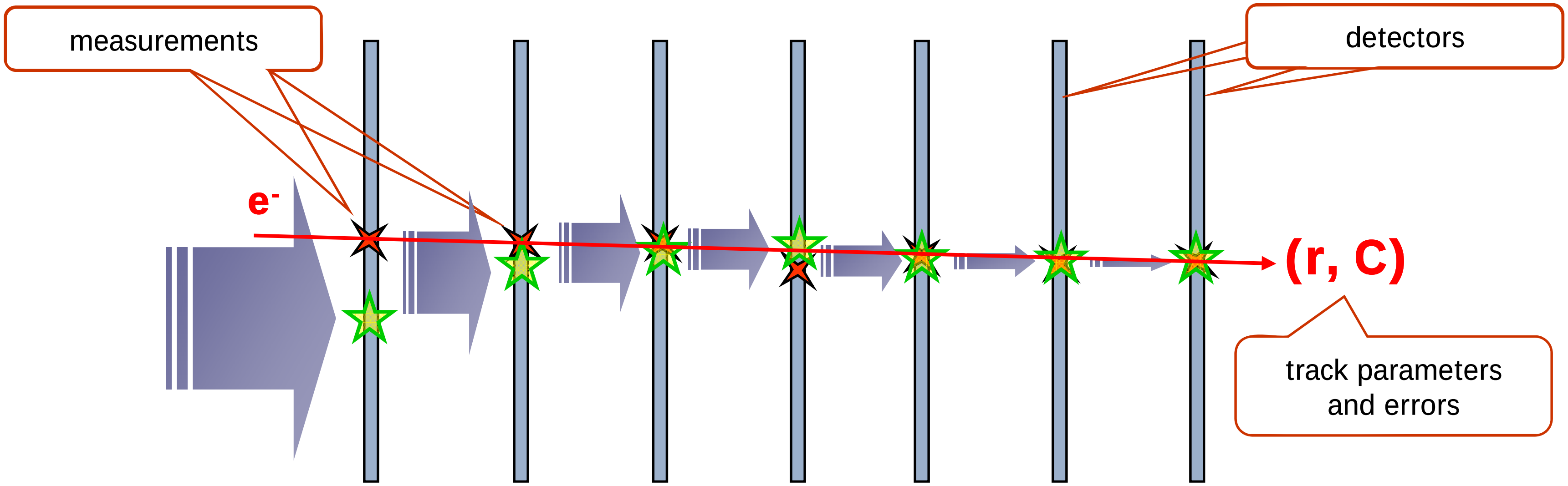
weight

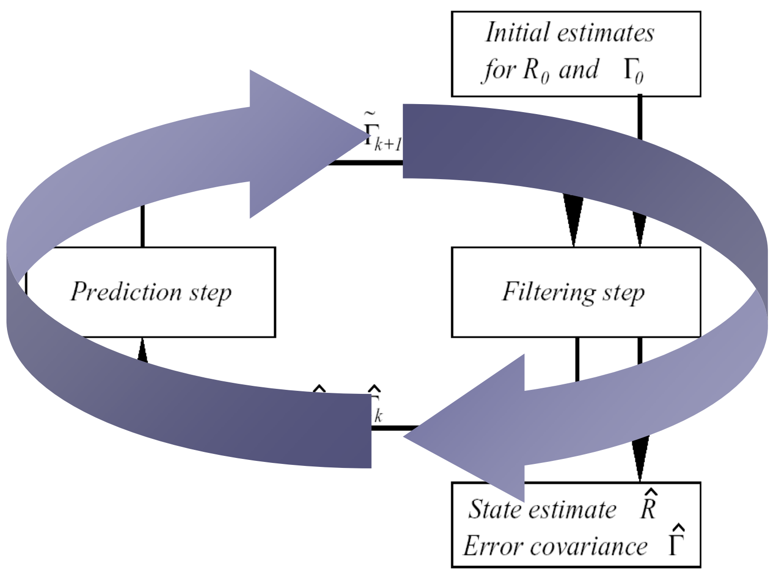
correction



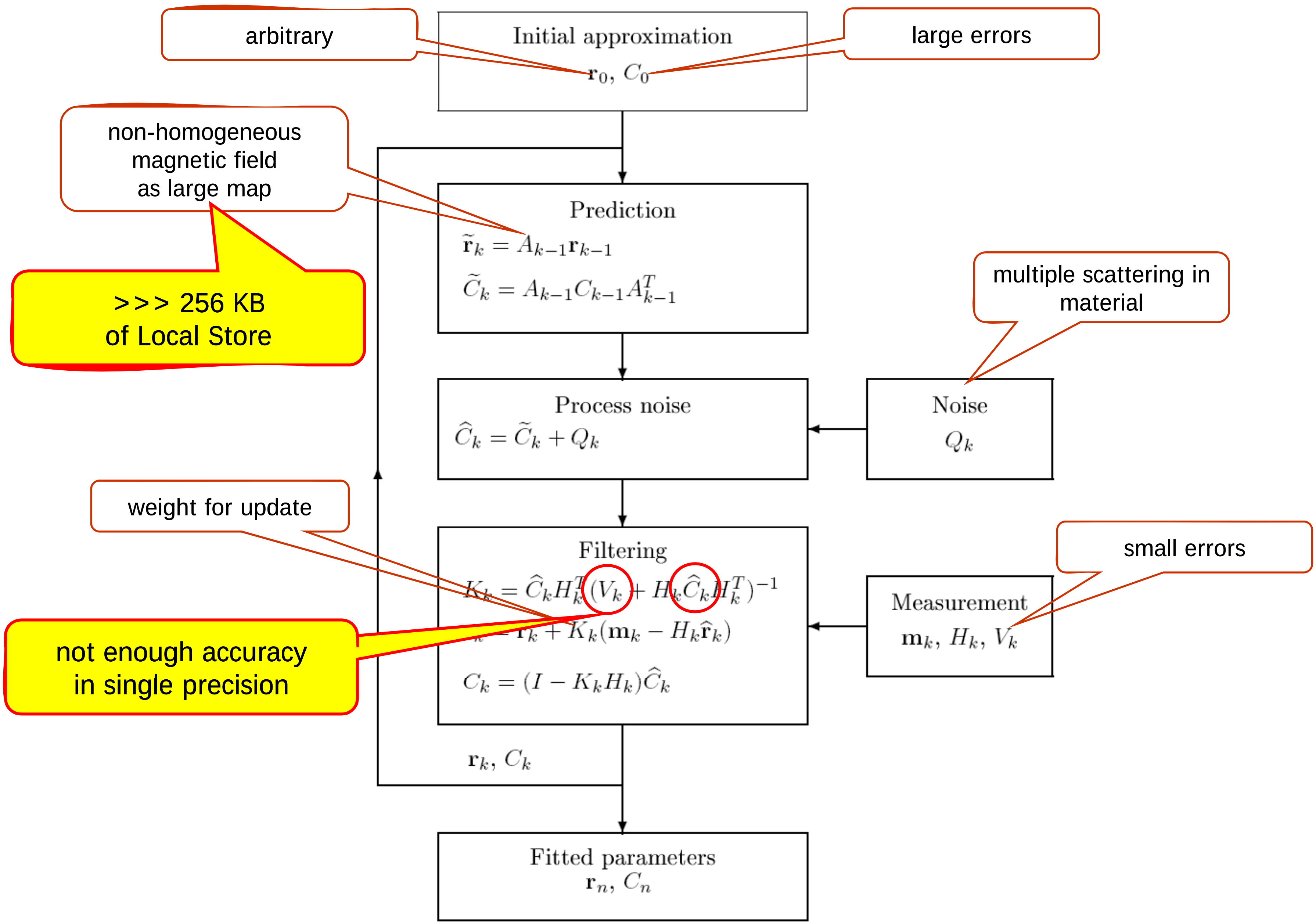
The Kalman filter exploits the dynamics of the target, which govern its time evolution, to remove the effects of the noise and get a good estimate of the location of the target

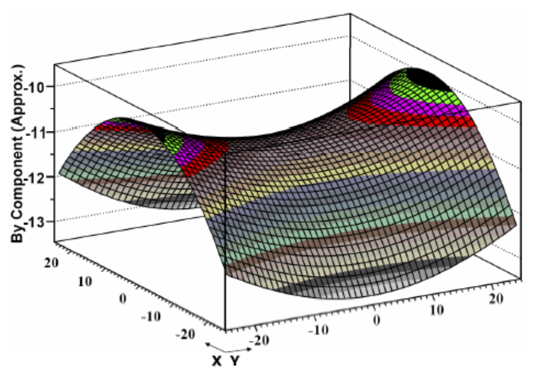
- at the present time (filtering),
- at a future time (prediction), or
- at a time in the past (interpolation or smoothing).


measurements
detectors
e-
(r, C)
track parameters
and errors


Initial estimates
for R0 and Γ0
Prediction step
Filtering step
State estimate
Error covariance

The Kalman Filter for Track Fit





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

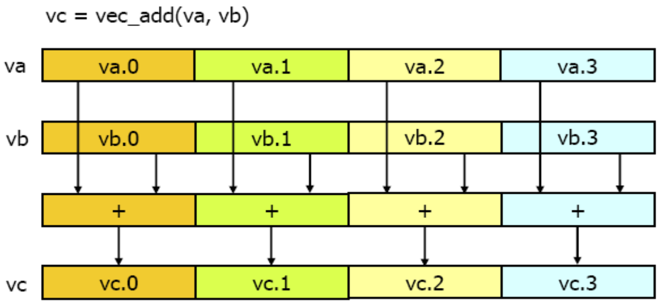
- The initial track parameters are directly estimated from the input data.
- The propagation step is performed directly from measurement to measurement without intermediate steps.
- Matrix multiplications have been replaced by direct operations on only non-trivial matrix elements.
- Most loops have been 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.

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

Data Types:

- Scalar double
- Scalar float
- Pseudo-vector (array)
- Vector (4 float)

c = a + b



Platform:

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

SPE Statistics

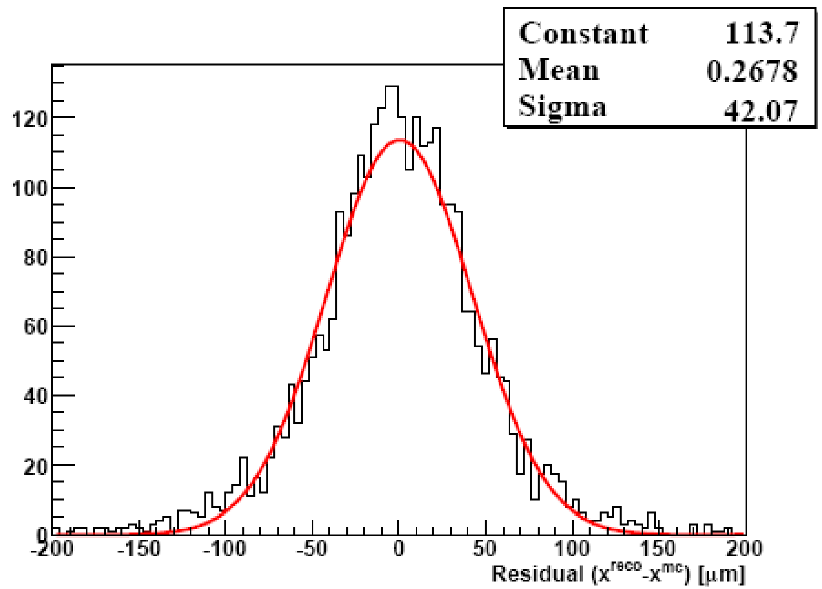
mysim/SPE4: Statistics

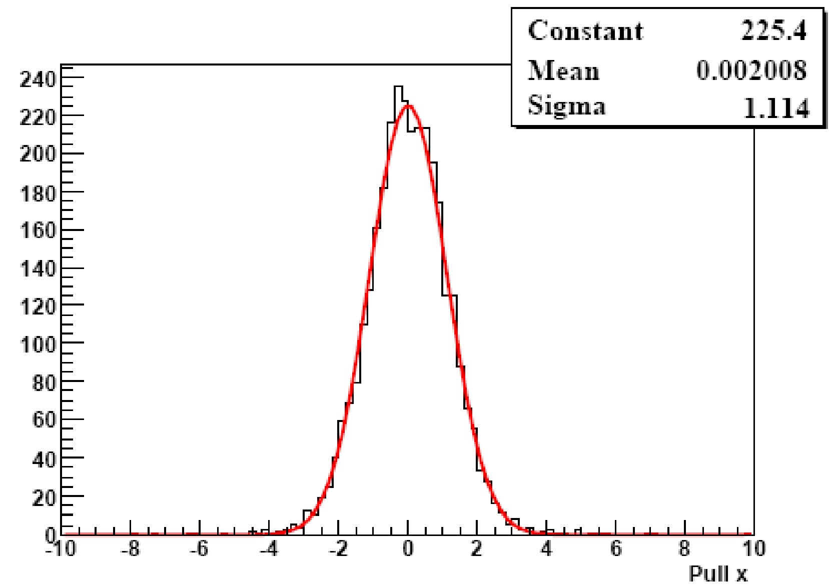
```
SPU DD3.0
***
Total Cycle count               335660
Total Instruction count         643
Total CPI                       522.02
***
Performance Cycle count         7076
Performance Instruction count   6838 (6638)
Performance CPI                 1.03 (1.07)

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
```

Timing profile !

No need to check
the assembler code !

	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





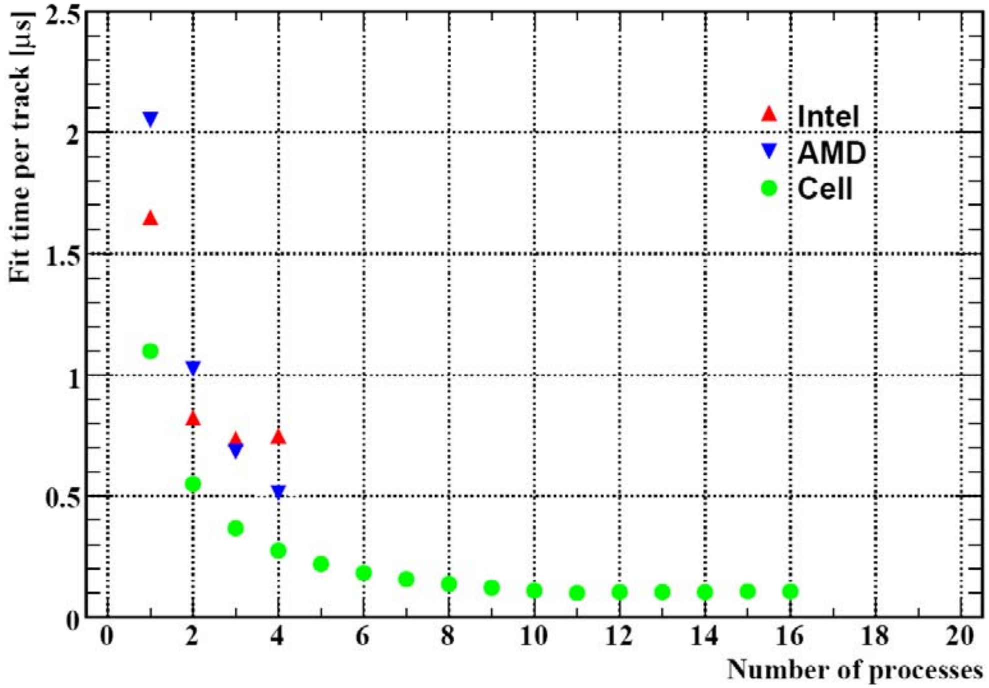
Kalman Filter on Intel Xeon, AMD Opteron and Cell

Fit of a single track:

	Processing Units	Cache/LS, kB	Clock, GHz	Time, μs	kCycle/Track
lxg1411	2 Intel Xeon with HT	512	2.66	1.47	3.91
eh102	2 Dual Core AMD Opteron	1024	1.8	1.86	3.35
blade11bc4	2 Cell Broadband Engine	256	2.4	0.87	2.09

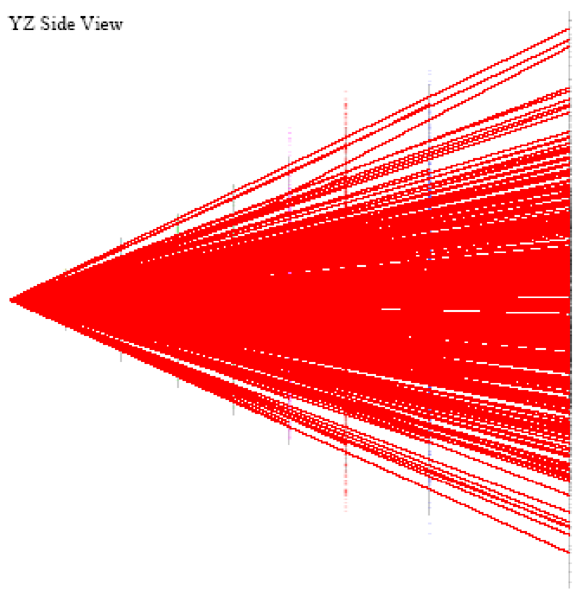
Time ratios: 1.7 (Intel Xeon to Cell), 2.1 (AMD Opteron to Cell); kCycle/Track ratios: 1.9 (Intel Xeon to Cell), 1.6 (AMD Opteron to Cell)

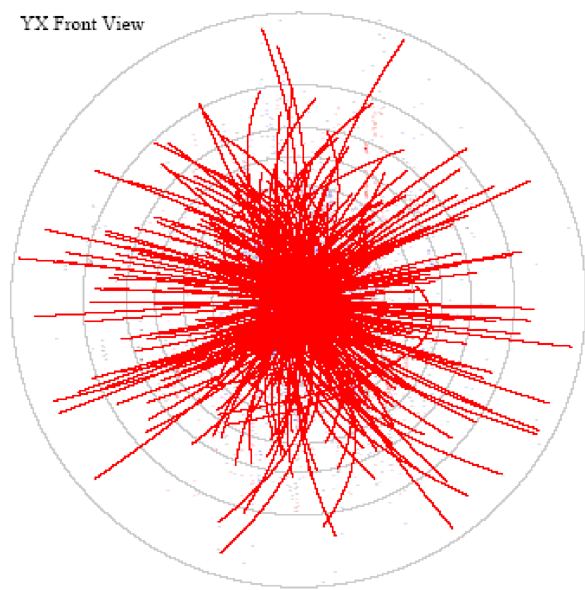
Fit of thousands of tracks:

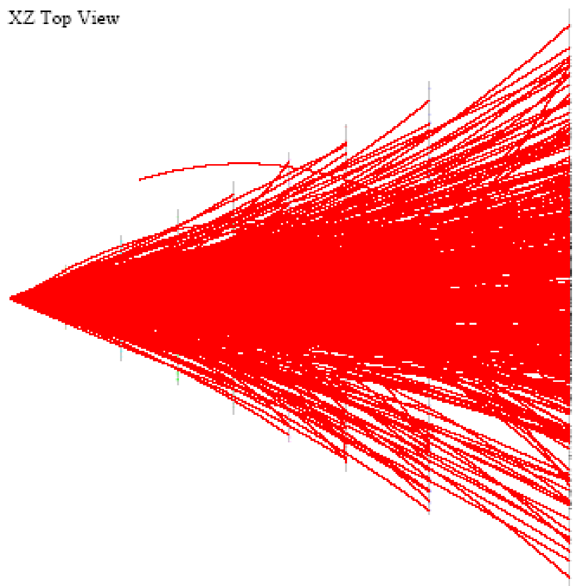


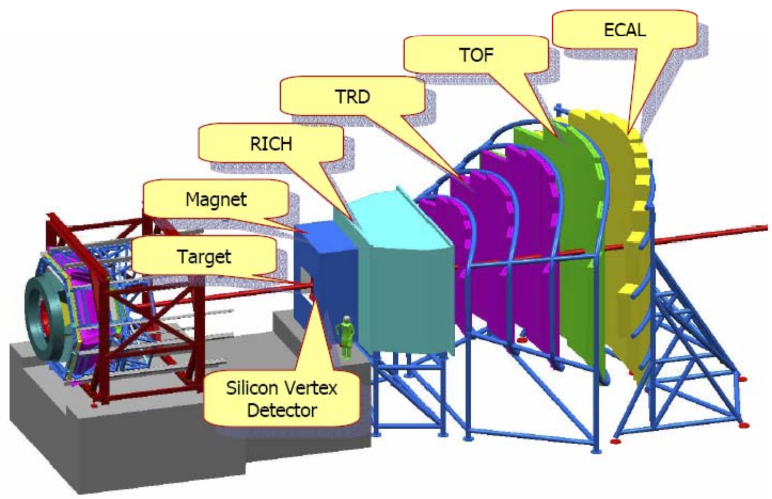
Cell SPE is: 1.5 times faster than Intel Xeon
and 2 times faster than AMD Opteron

CBM: Track Finding Challenge



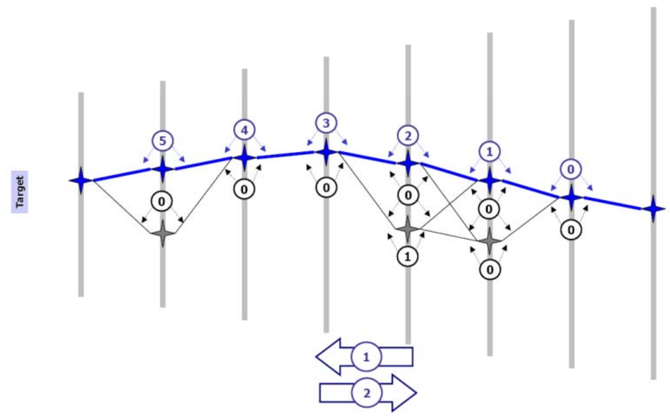






CBM (FAIR/GSI)

Cellular Automaton Track Finder: Pseudocode



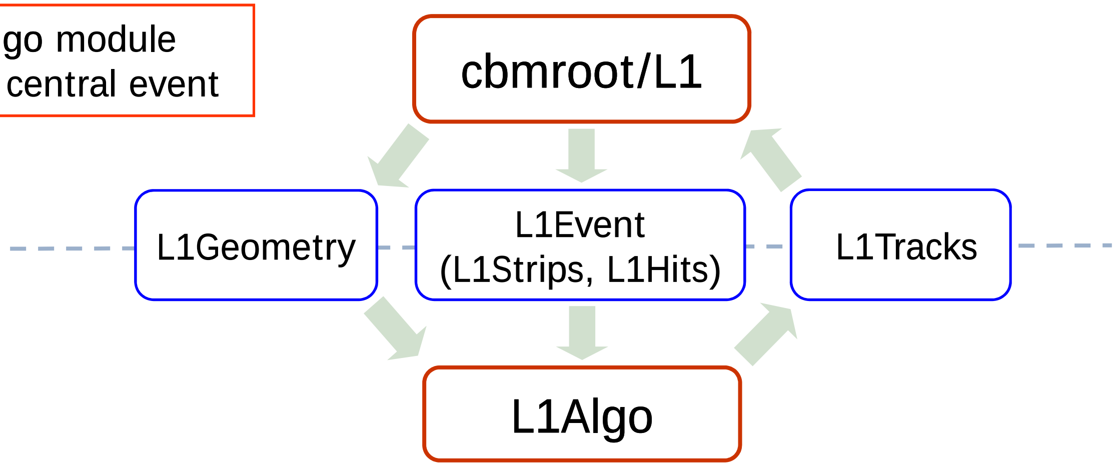
1 Create tracklets

2 Collect tracks

```
void CATrackFinder(){
  for (step = 0; step <= 2; step++){
    if (step == 0) MinMom = 1.0; else MinMom = 0.2;
    if (step <= 1) TargetConstrHit(ht); else NoTargetConstrHit(ht);
    for (sta = NStations-3; sta >= 0; sta--){
      for (hl = FirstHit[sta]; hl <= LastHit[sta]; hl++){
        if (hl.used) continue;
        Triplet trl = Triplet(ht, hl, MinMom);
        trl.Propagate(sta+1);
        for (hm = FirstHit[sta+1]; hm <= LastHit[sta+1]; hm++){
          if (hm.used) continue;
          Triplet trm = Triplet(trl);
          trm.AddHit(hm);
          if (trm.chi2 > MaxChi2) continue;
          trm.Propagate(sta+2);
          for (hr = FirstHit[sta+2]; hr <= LastHit[sta+2]; hr++){
            if (hr.used) continue;
            Triplet trr = Triplet(trm);
            trr.AddHit(hr);
            if (trr.chi2 > MaxChi2) continue;
            for (tr = FirstTriplet[hm]; tr <= LastTriplet[hm]; tr++){
              if (trr.hr != tr.hm) continue;
              if (fabs(trr.p-tr.p)/trr.errp > MaxDistP) continue;
              if (trr.level <= tr.level) trr.level = tr.level+1;
            }
            trr.Store();
            hl.StorePointer(trr);
          }
        }
      }
    }
    for (level = NStations-3; level >= 0; level--){
      for (tri = FirtsTriplet; tri <= LastTriplet; tri++){
        if (tri.level != level) continue;
        if ((tri.hl.used)||(tri.hm.used)||(tri.hr.used)) continue;
        Track tra = FindBestBranchRecursive(tri);
        if ((tra.chi2 > MaxChi2)||(GhostTrack(tra))) continue;
        tra.StoreAsCandidate();
      }
      SortTrackCandidates();
      for (tra = FirstTrackCand; tra <= LastTrackCand; tra++){
        if (tra.NUsedHits() != 0) continue;
        tra.MarkHitsAsUsed();
        tra.Store();
      }
    }
    MergeClones();
    GatherHits();
  }
}
```

Structure and Data

- A standalone L1Algo module
- About 300 kB per central event



Input:

Strips:

```
float          vStripValues[NStrips];  // strip coordinates (32b)
unsigned char  vStripFlags [NStrips];  // strip iStation (6b) + used (1b) + used_by_dublets (1b)
```

Hits:

```
struct L1StsHit {
  unsigned short int f, b;             // front (16b) and back (16b) strip indices
};
L1StsHit vHits[NHits];
```

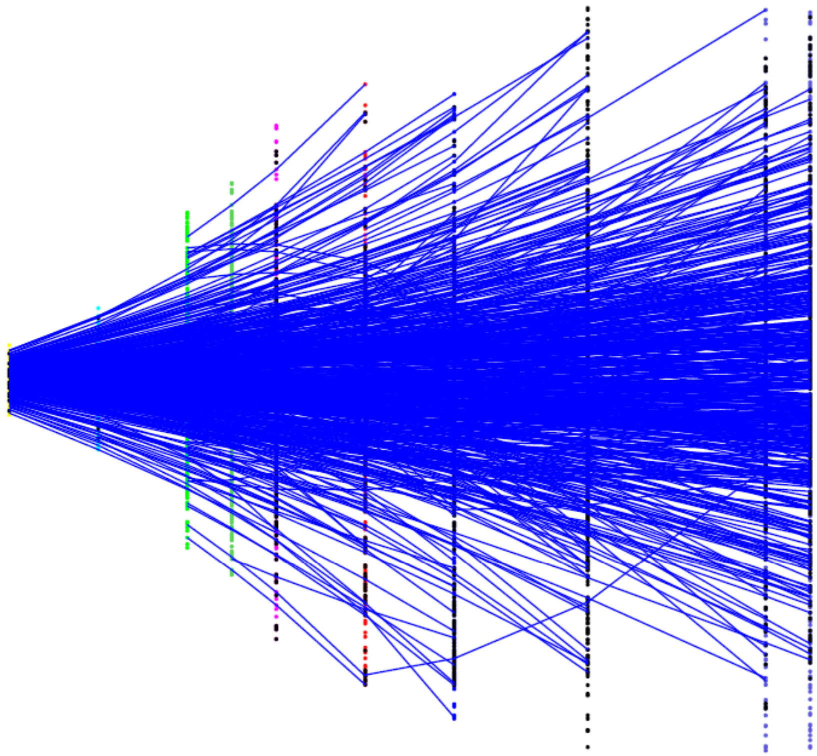
Output:

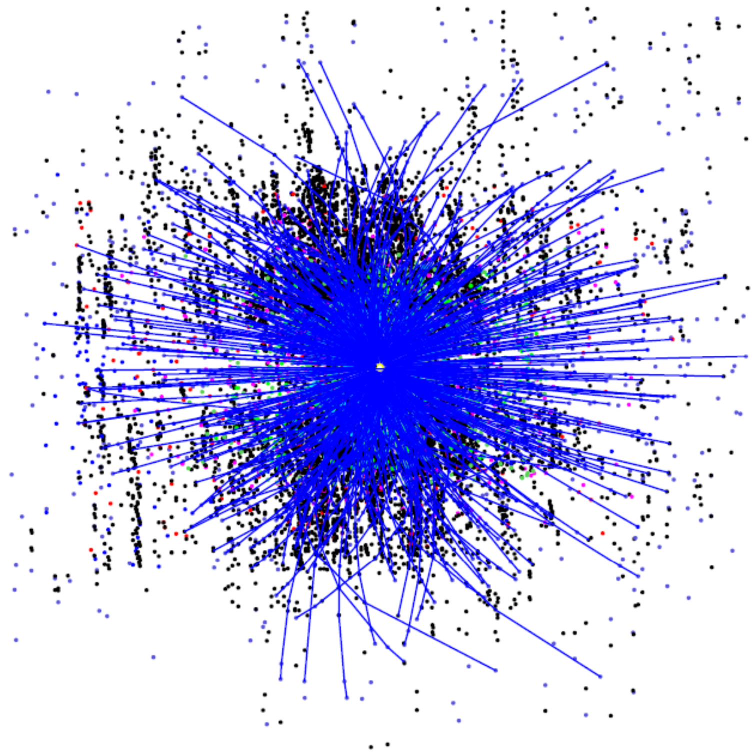
```
unsigned short int vRecoHits   [NRecoHits];    // hit index (16b)
unsigned char      vRecoTracks [NRecoTracks];  // N hits on track (8b)
```

Internal:

```
class L1Triplet{
  unsigned short int w0;   // left hit (16b)
  unsigned short int w1;   // first neighbour (16b) or middle hit (16b)
  unsigned short int w2;   // N neighbours (16b) or right hit (16b)
  unsigned char      b0;   // chi2 (5b) + level (3b)
  unsigned char      b1;   // qp (8b)
  unsigned char      b2;   // qp error (8b)
}
```

Reconstructed Event



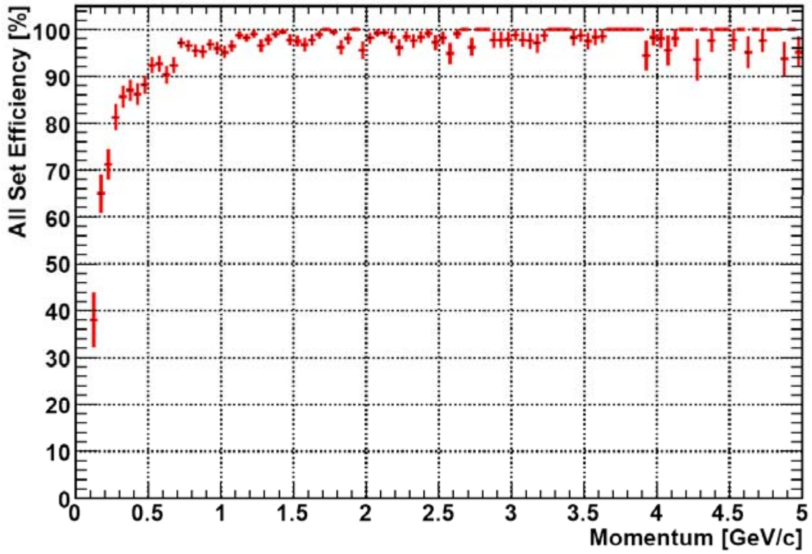


Chamber		1	2	3	4	5	6	7	8
	true	126	133	169	179	175	185	165	153
Min. bias	fake	4	4	0	0	650	606	395	257
	total	130	137	169	179	825	791	560	410
	true	588	615	775	821	803	805	760	696
Central	fake	18	18	2	2	5095	4158	3014	1953
	total	606	636	777	823	5898	4963	3774	2649

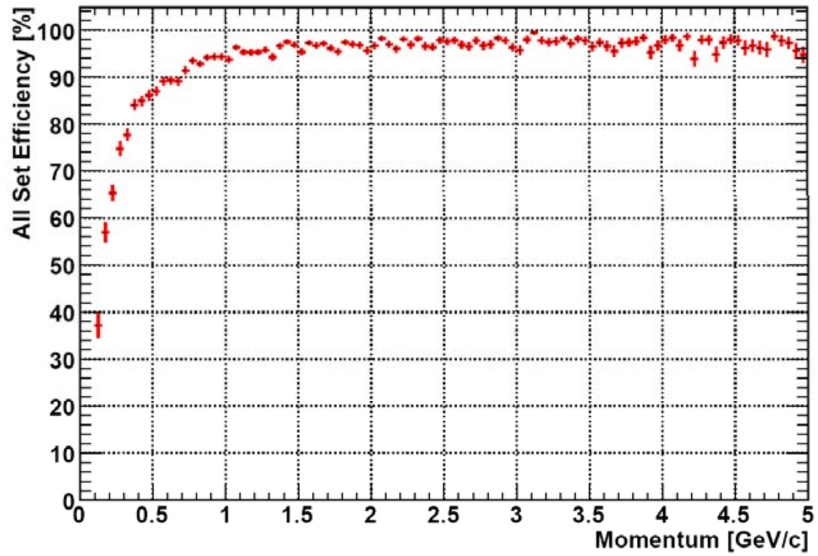
CA Track Finder Efficiency

Standard geometry: 2M2P4S

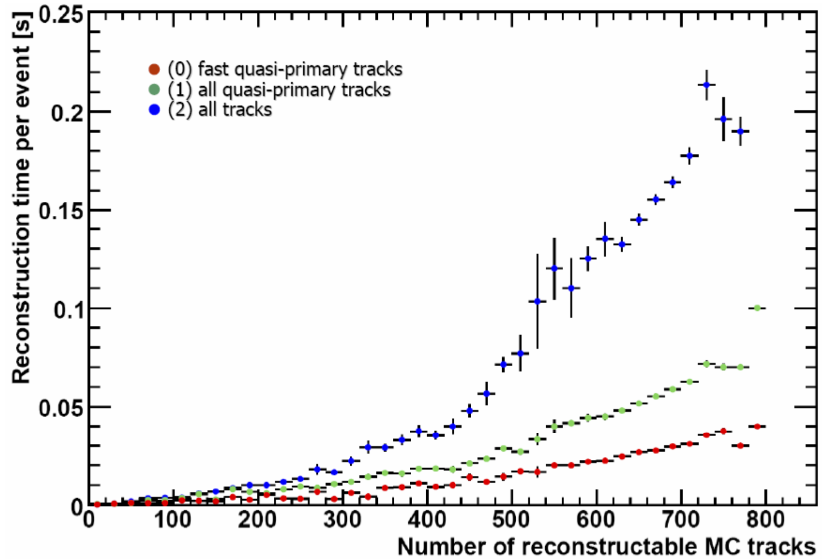
MBias events



Central events



Efficiency, %	Track category	Efficiency, %
98.0	Reference set (>1 GeV/c)	96.6
95.4	All set (>=4 hits, >100 MeV/c)	93.5
89.1	Extra set (<1 GeV/c)	85.9
0.4	Clone	0.4
1.6	Ghost	4.7
140	MC tracks/event found	633



Summary and Conclusion

- Precise fit using the Kalman filter
- Track finding algorithms that can be parallelized (CA)
- Use of the SIMD architecture (4x)
- Single-precision floating point (speed and size)
- Limited data manipulation
- Multi-core CPUs
- Other hardware for large combinatorics (GPU, FPGA, ?)
- Tools for debugging (timing profile, ...)
- Portable code (Intel, AMD, Cell, ...)

- Efficient event reconstruction is very expensive – thousands of CPUs !
- Inefficient event reconstruction is even more expensive $\varepsilon_{tot} = (\varepsilon_{phys} * \varepsilon_{det} * \varepsilon_{elctr}) * \varepsilon_{reco}$!!!
- Reconstruction = Physics + Mathematics + Computers + Detectors + Electronics