Status of the Cellular Automaton track finder

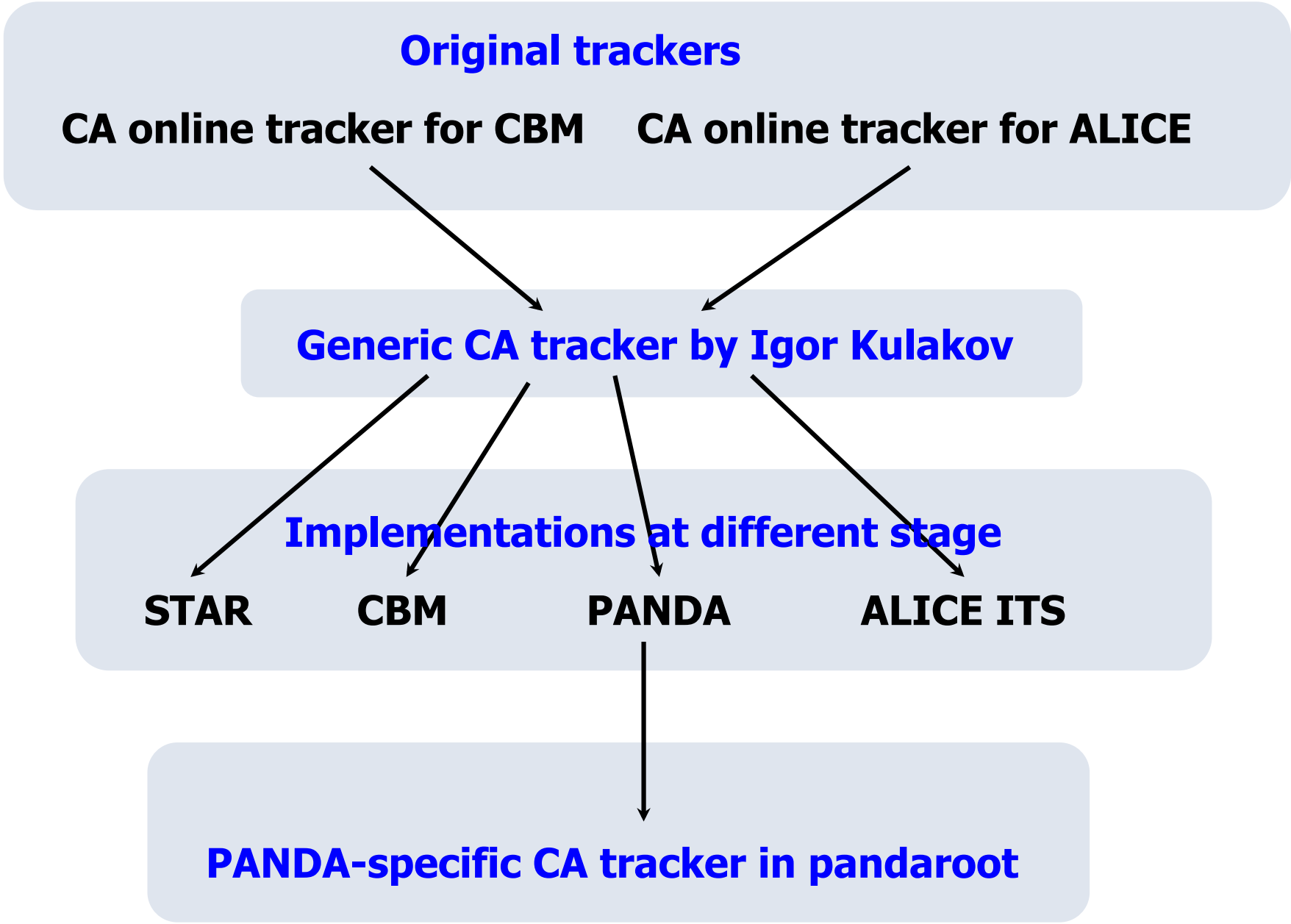
S. Gorbunov, I. Kisel, I. Kulakov, M. Zyzak

June 8, 2015

FIAS Frankfurt Institute for Advanced Studies

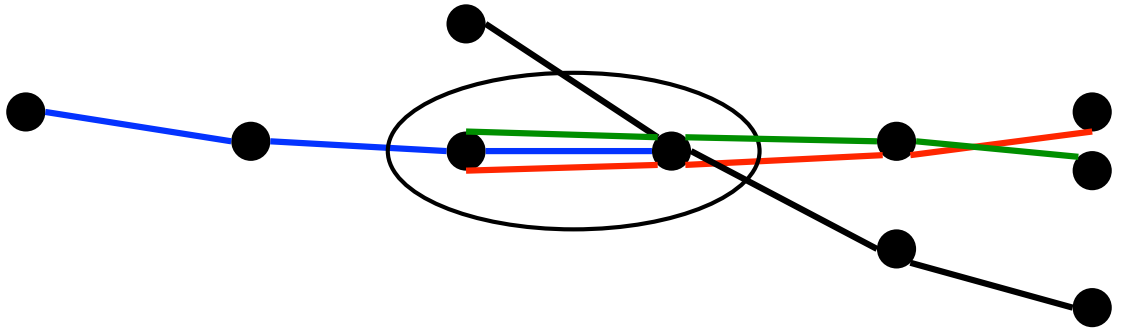
JOHANN WOLFGANG GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

CA tracker development history



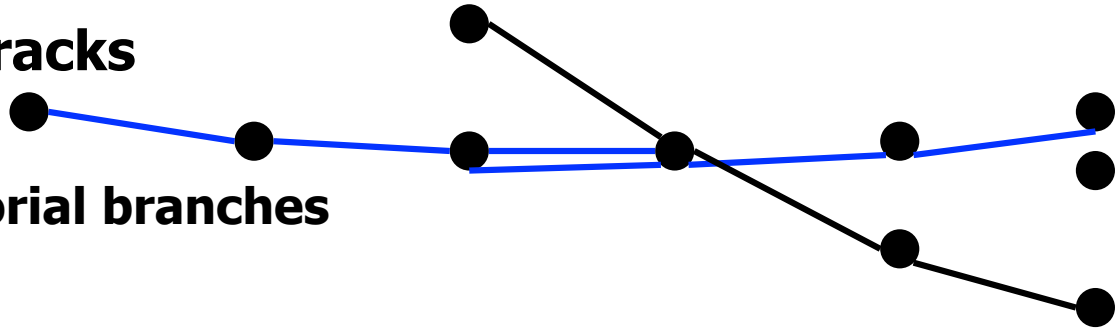
The Cellular Automaton method

1. **Create tracklets**
   - everywhere
   - simple math



2. **Find neighbours**
   - look for common hits
   - look for same geo params

3. **Merge tracklets + fit tracks**
   - Kalman filter
   - select best combinatorial branches
   - suppress fakes



4. **Final tuning - pick up missed hits, merge gaps, etc**

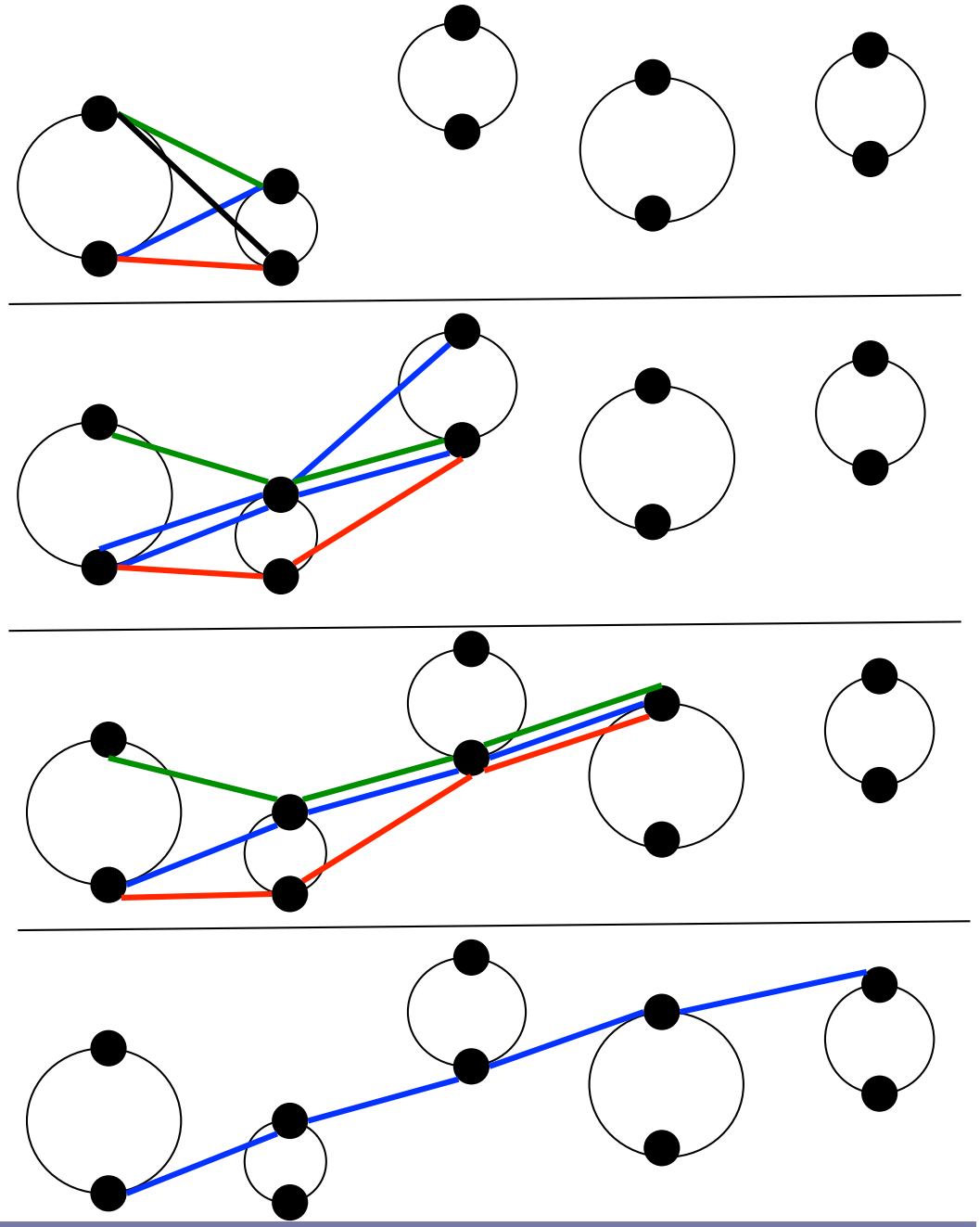
CA tracking in PANDA

Tracking in PANDA:

- **tracks are well separated**
- **tens of detector layers**
- **constant field**
- **~no scattering**

CA features (generic algorithm):

- **5-station-tracklets**
- **separate left & right hits**
- **full combinatorics: all hit combinations -> tracklets**
- **Kalman filter at early stage, already for tracklet creation**



Optimization – Kalman filter

Use of "light" Kalman filter for track finding

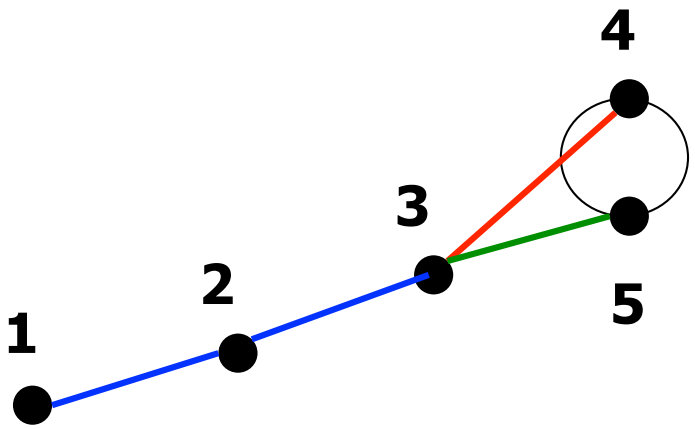
Example: linearisation at q/p=0 :

```
const float_v ex = t0.CosPhi();
const float_v ey = t0.SinPhi();
const float_v k  = t0.QPt() * Bz;
const float_v dx = x - X();

 float_v ey1 = k * dx + ey;
 float_v ex1 = CAMath::Sqrt( 1.f - ey1 * ey1 );
 ex1( ex < Vc::Zero ) = -ex1;
 const float_v dx2 = dx * dx;
 const float_v ss = ey + ey1;
 const float_v cc = ex + ex1;
 const float_v cci = 1.f / cc;
 const float_v exi = 1.f / ex;
 const float_v ex1i = 1.f / ex1;
 float_m mask = _mask && CAMath::Abs( ey1 ) <= maxSinPhi && CAMath::Abs( cc ) >= 1.e-4f && CAMath::Abs( ex ) >= 1.e-4f && CAMath::Abs( ex1 ) >= 1.e-4f;
 const float_v tg = ss * cci; // tan((phi1+phi)/2)
 const float_v dy = dx * tg;
 float_v dl = dx * CAMath::Sqrt( 1.f + tg * tg );
 dl( cc < Vc::Zero ) = -dl;
 const float_v dSin = CAMath::Max( float_v( -1.f ), CAMath::Min( float_v( Vc::One ), dl * k * 0.5f ) );
 const float_v dS = ( CAMath::Abs( k ) > 1.e-4f )  ? ( 2 * CAMath::ASin( dSin ) / k ) : dl;
 const float_v dz = dS * t0.DzDs();
```

Optimization of data structures

Example: storing hits, associated to track during branching



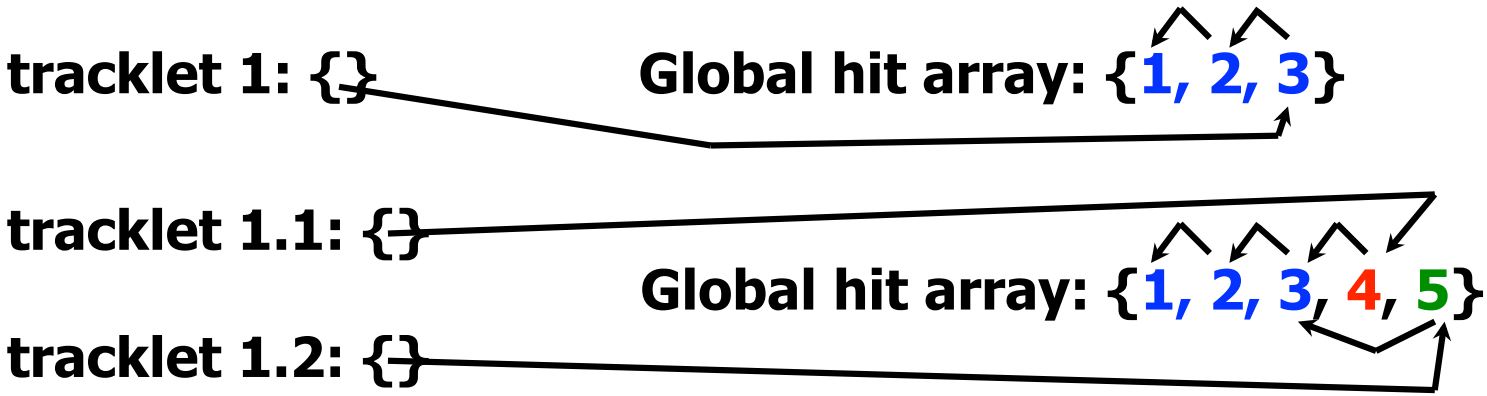
tracklet 1.1: {1,2,3,4}

tracklet 1: {1,2,3}

Array copy!

tracklet 1.2: {1,2,3,5}

New structure: 1 link to the last hit per track. No copy, no arrays.



Other optimizations

- **Using of SIMD vectors for calculations (VC package)**
  (SIMD was already implemented, but with almost no gain)
- **Optimizing calculations**

Current status

Efficiency: ~100.0 % (of MC tracks are reconstructed)
Fake rate: 1.5 % (of reco tracks are fake)
Clone rate: 4.8 % (of reco tracks are duplicated)
Speed : 1.0 ms (/event on single core AMD 2.9GHz)
+Final Fit: +0.7 ms (/event " - ")

- the code is in PandaRoot/catracking/*
- compiles (also on Mac) and runs
- sim/digi/reco macros are provided: macro/catracking/*
- tested (thanks to Stefano!) and ready to use

To do:

- further clean up the code
- tune the algorithm for the standard QA