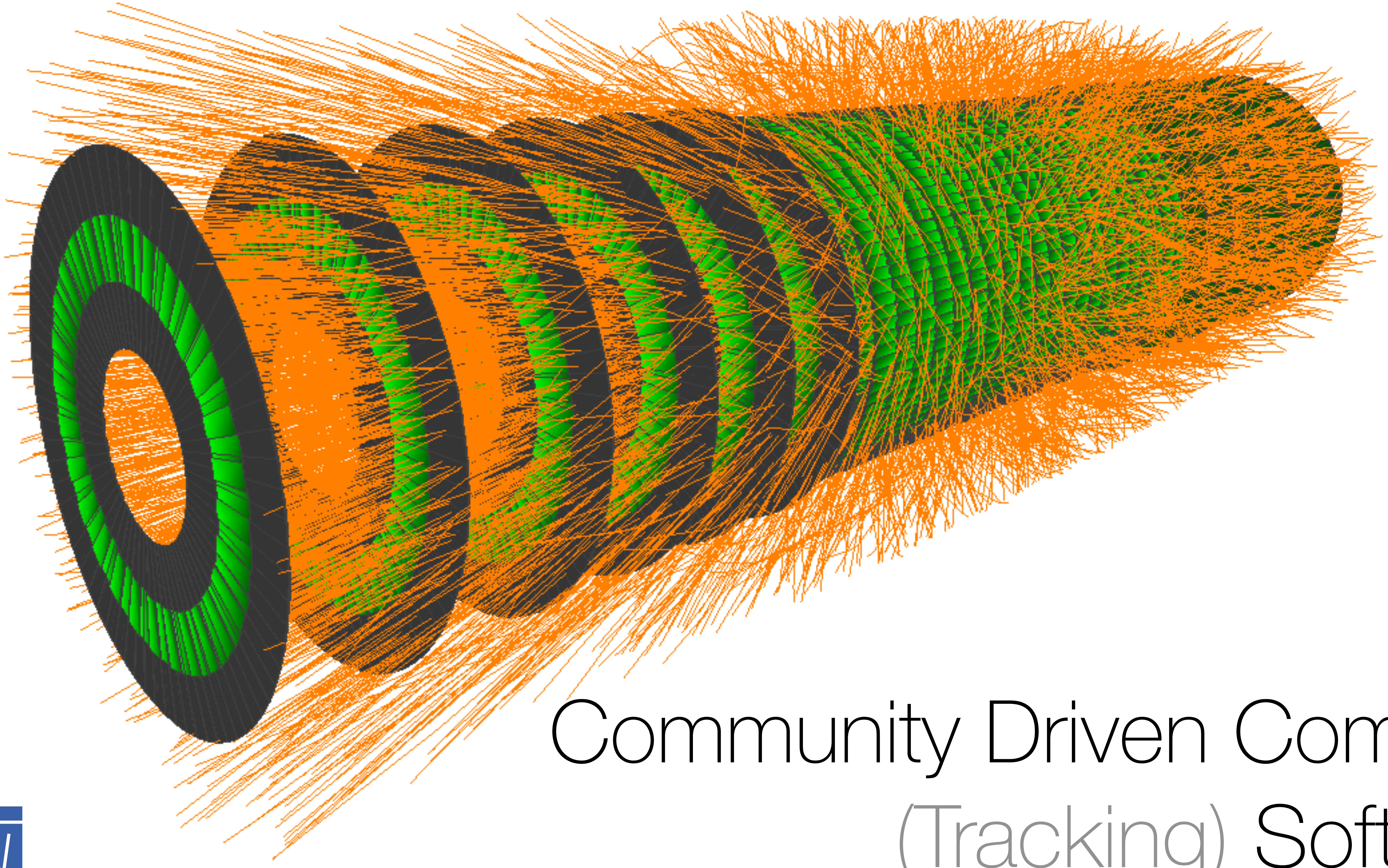
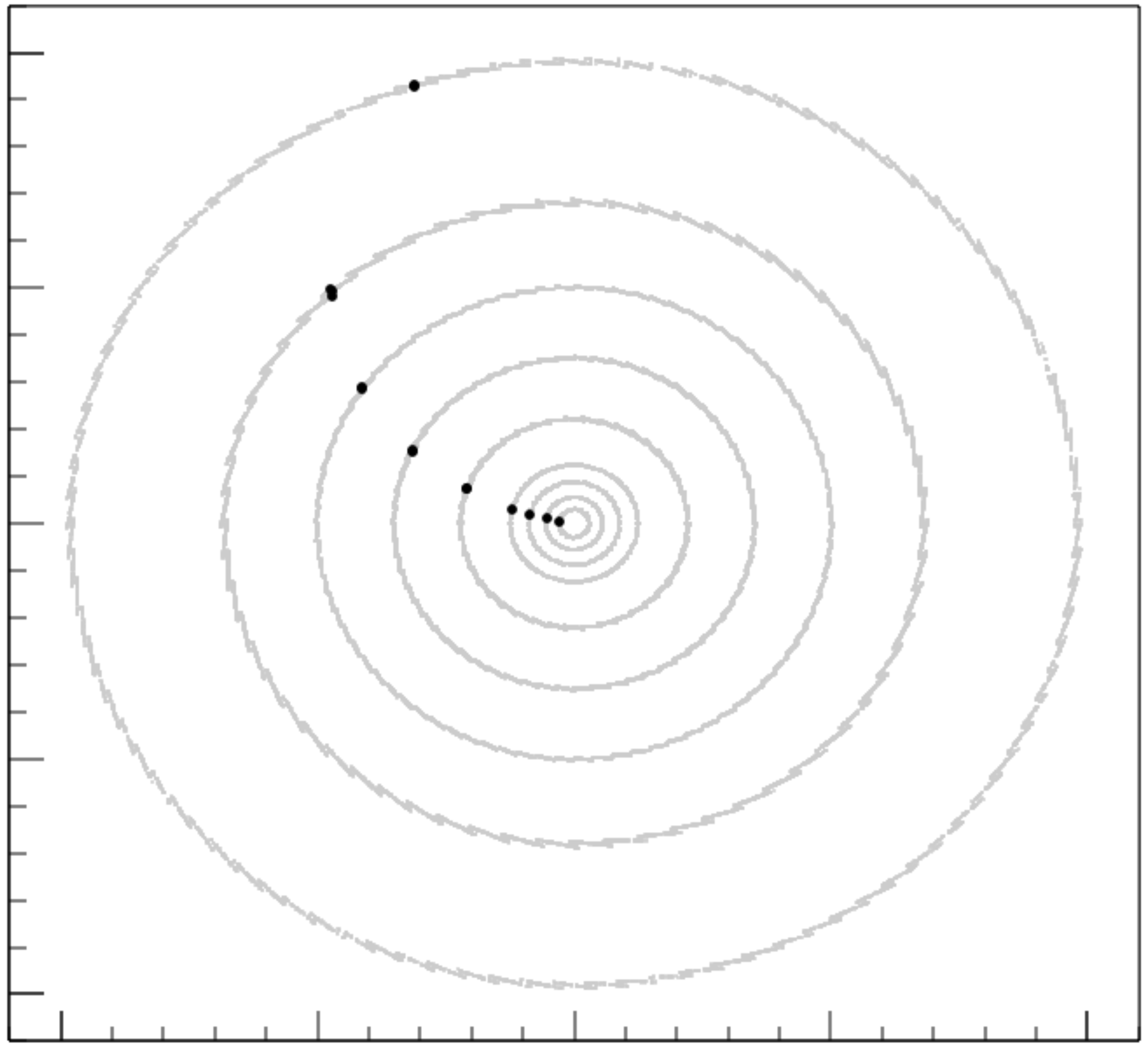


Community Driven Common (Tracking) Software

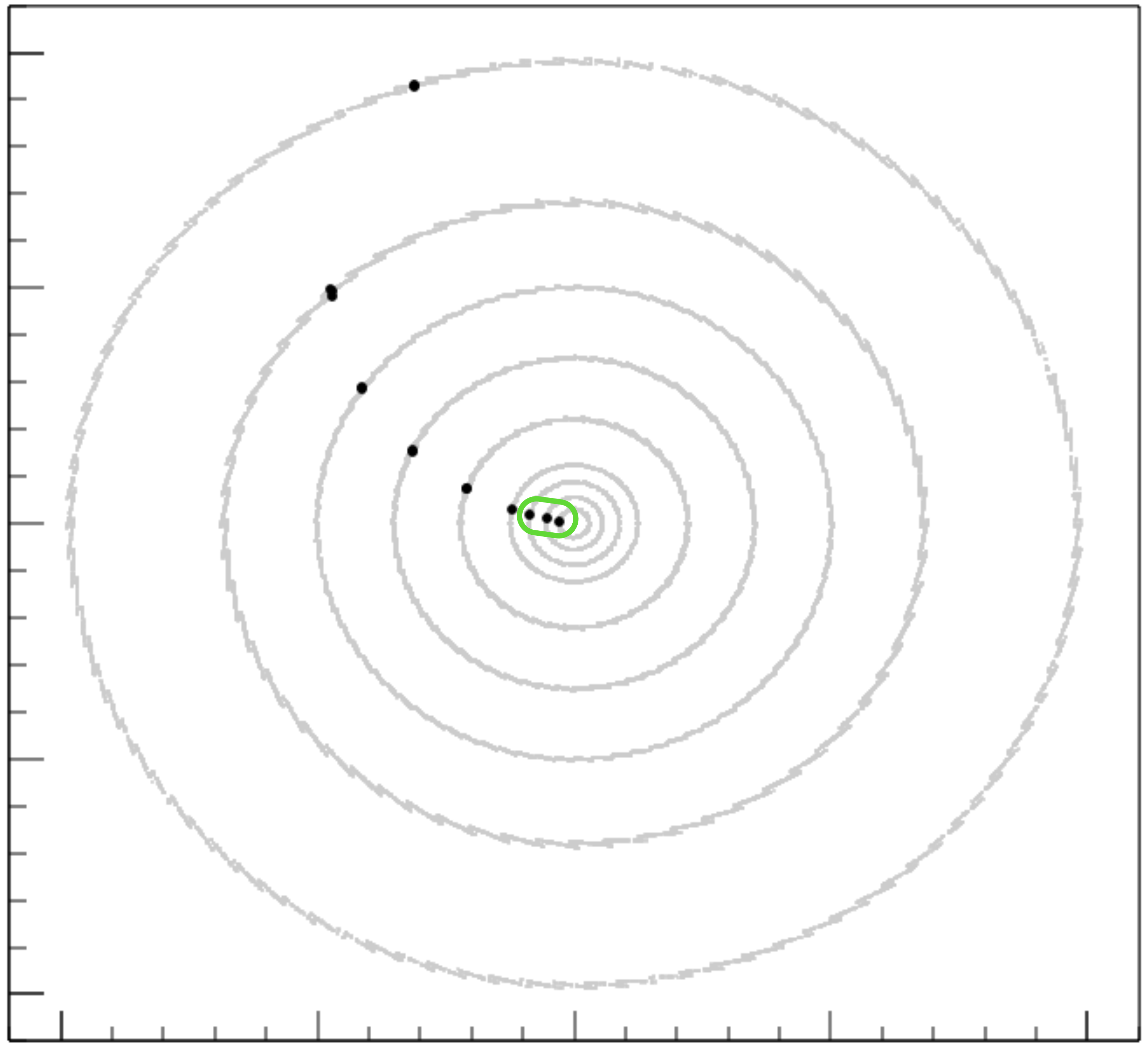
A. Salzburger (CERN)
for the ACTS team

Track Reconstruction

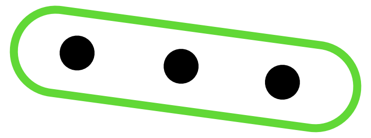


Hits from one particle

Track Reconstruction



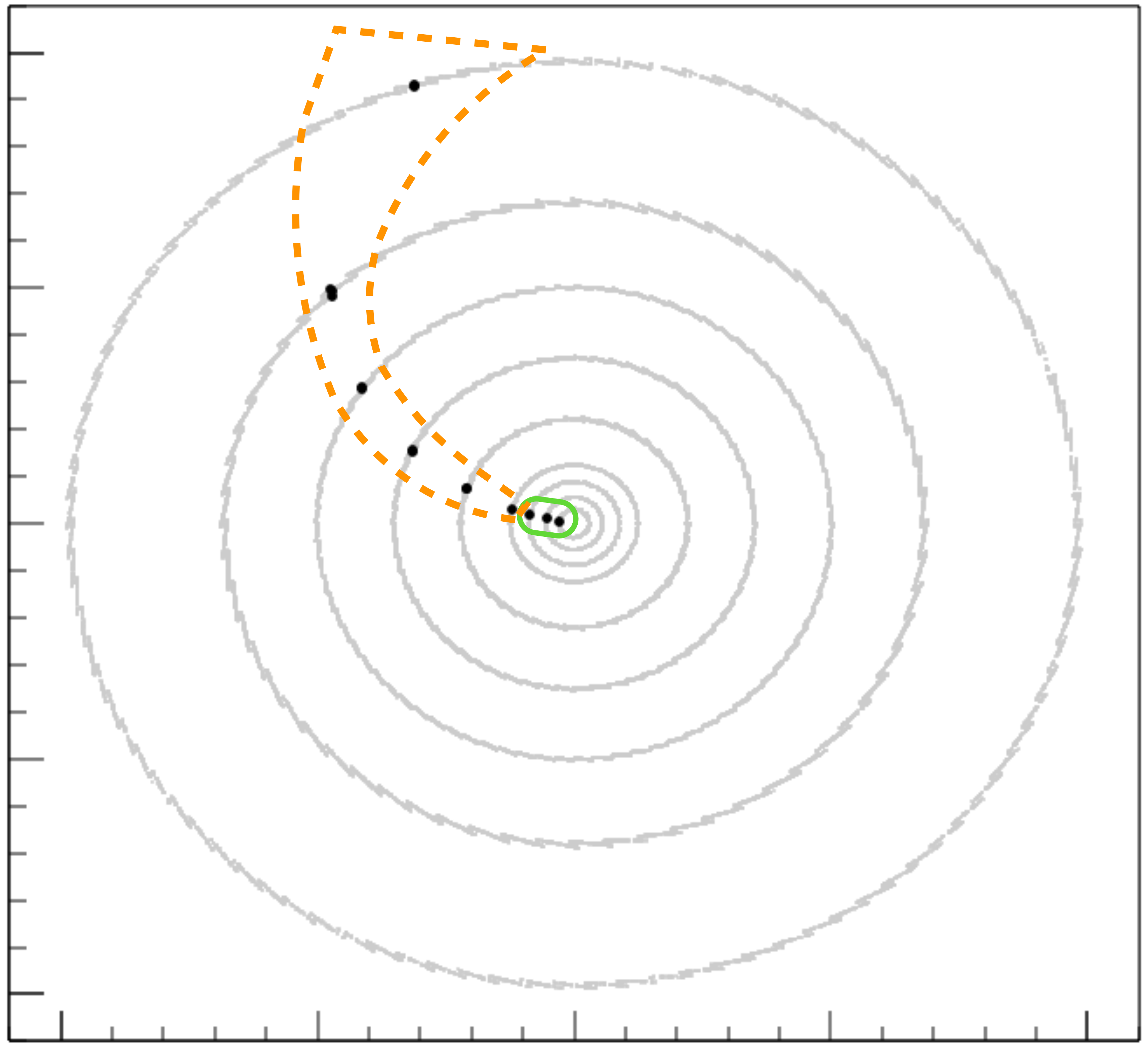
Hits from one particle



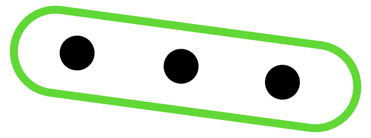
seeding

- triplet, quadruplet, n-plet finding

Track Reconstruction

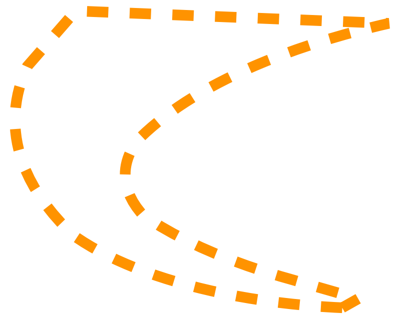


Hits from one particle



seeding

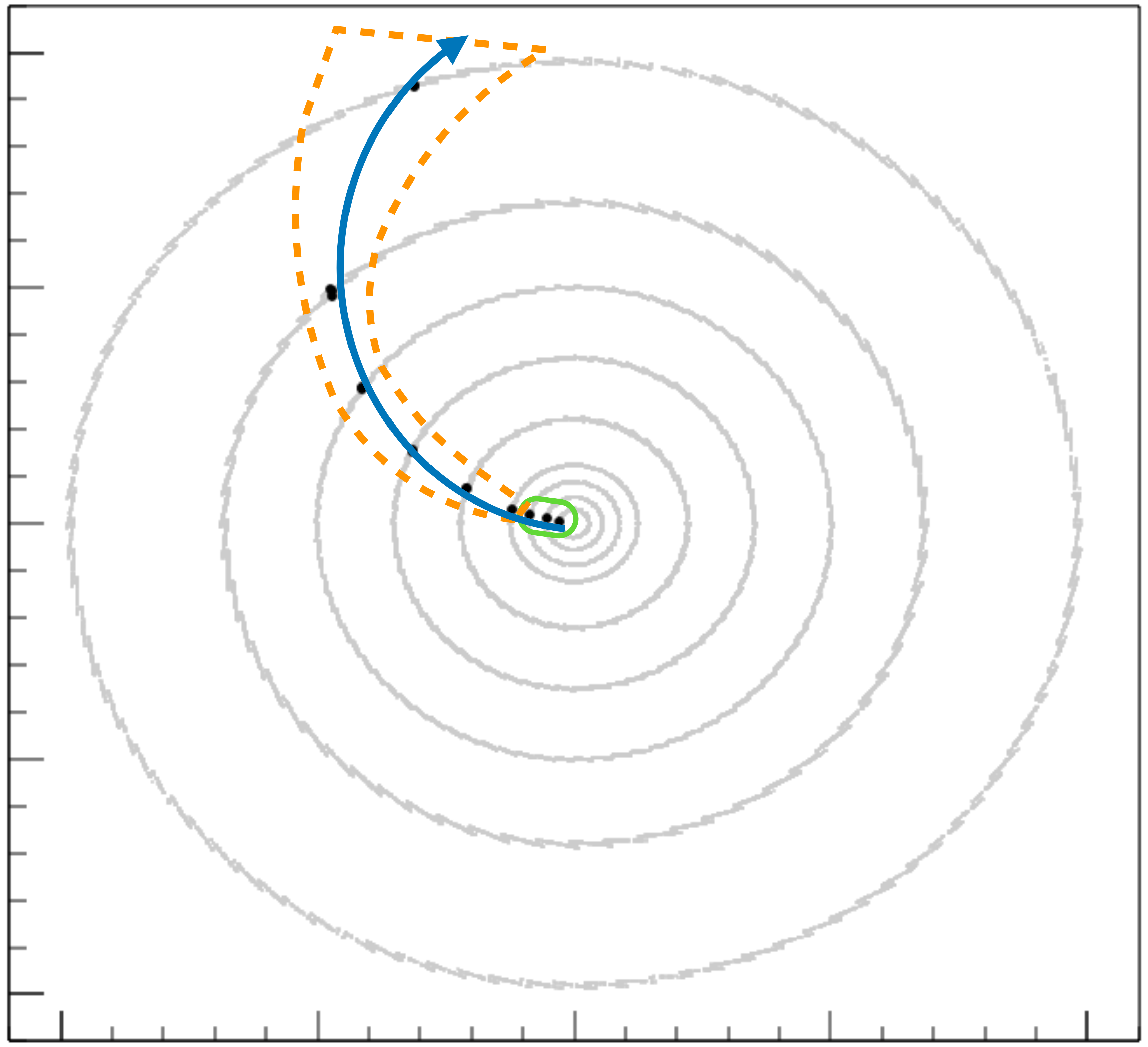
- triplet, quadruplet, n-plet finding



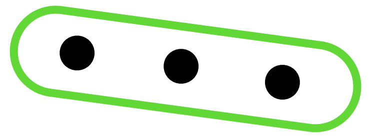
track finding

- combinatorial Kalman filtering

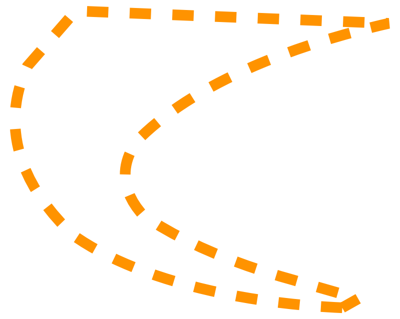
Track Reconstruction



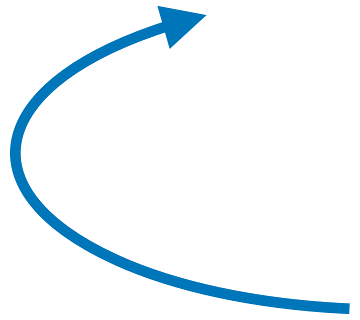
Hits from one particle



seeding
- triplet, quadruplet, n-plet finding

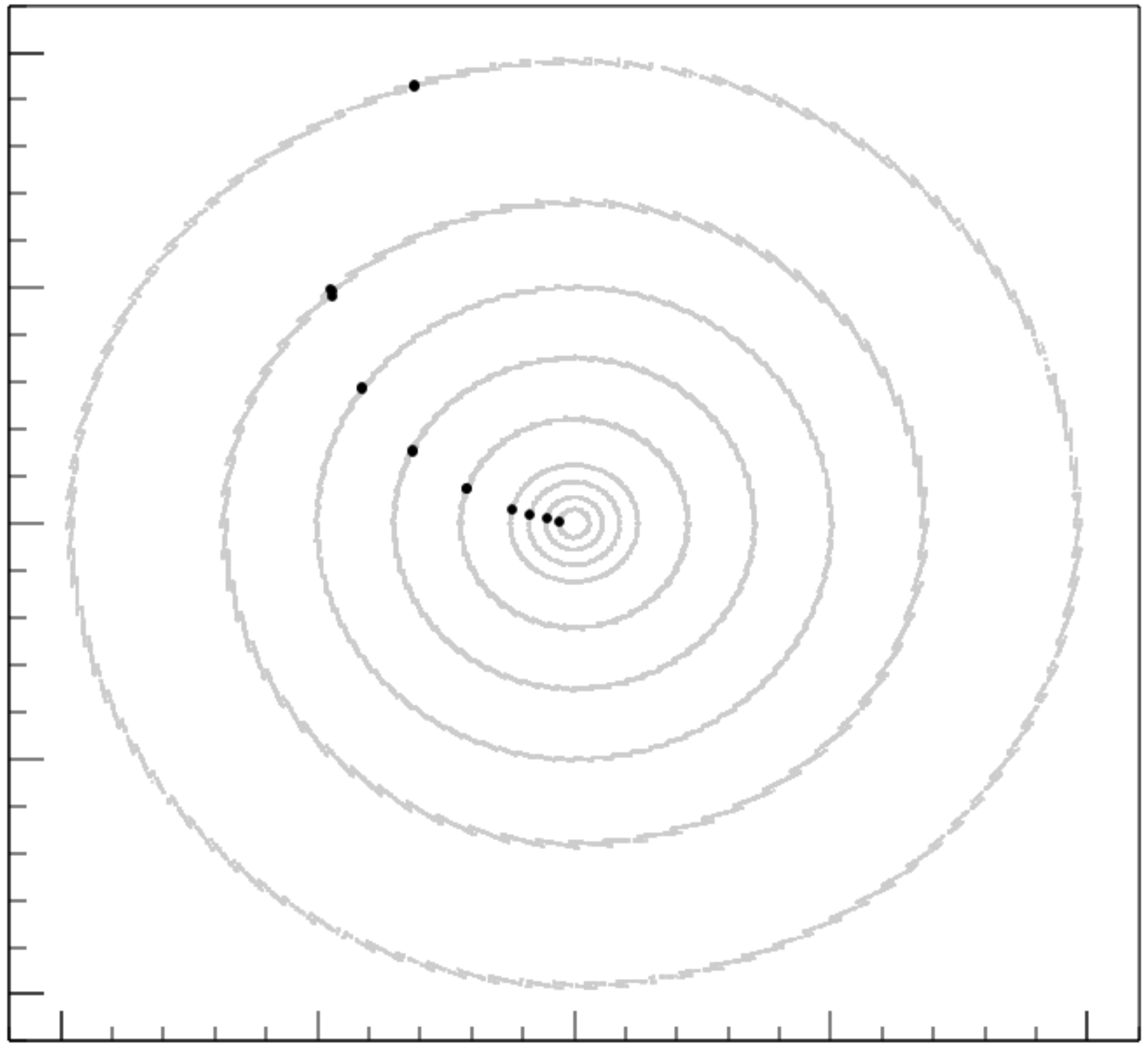


track finding
- combinatorial Kalman filtering

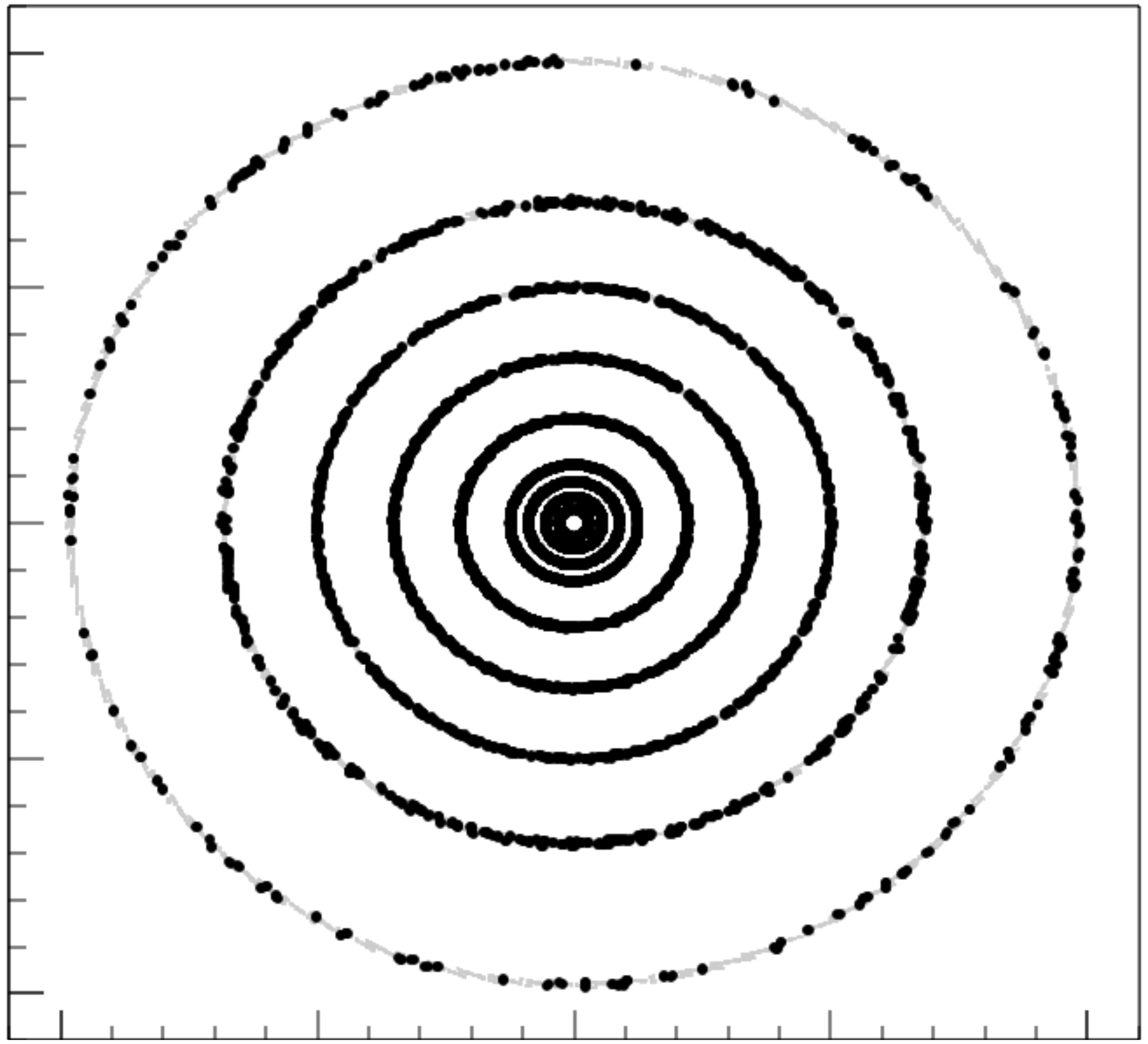


track fitting & classification
- Kalman filtering, smoothing
- χ^2 minimization
- track classification, etc.

Track Reconstruction



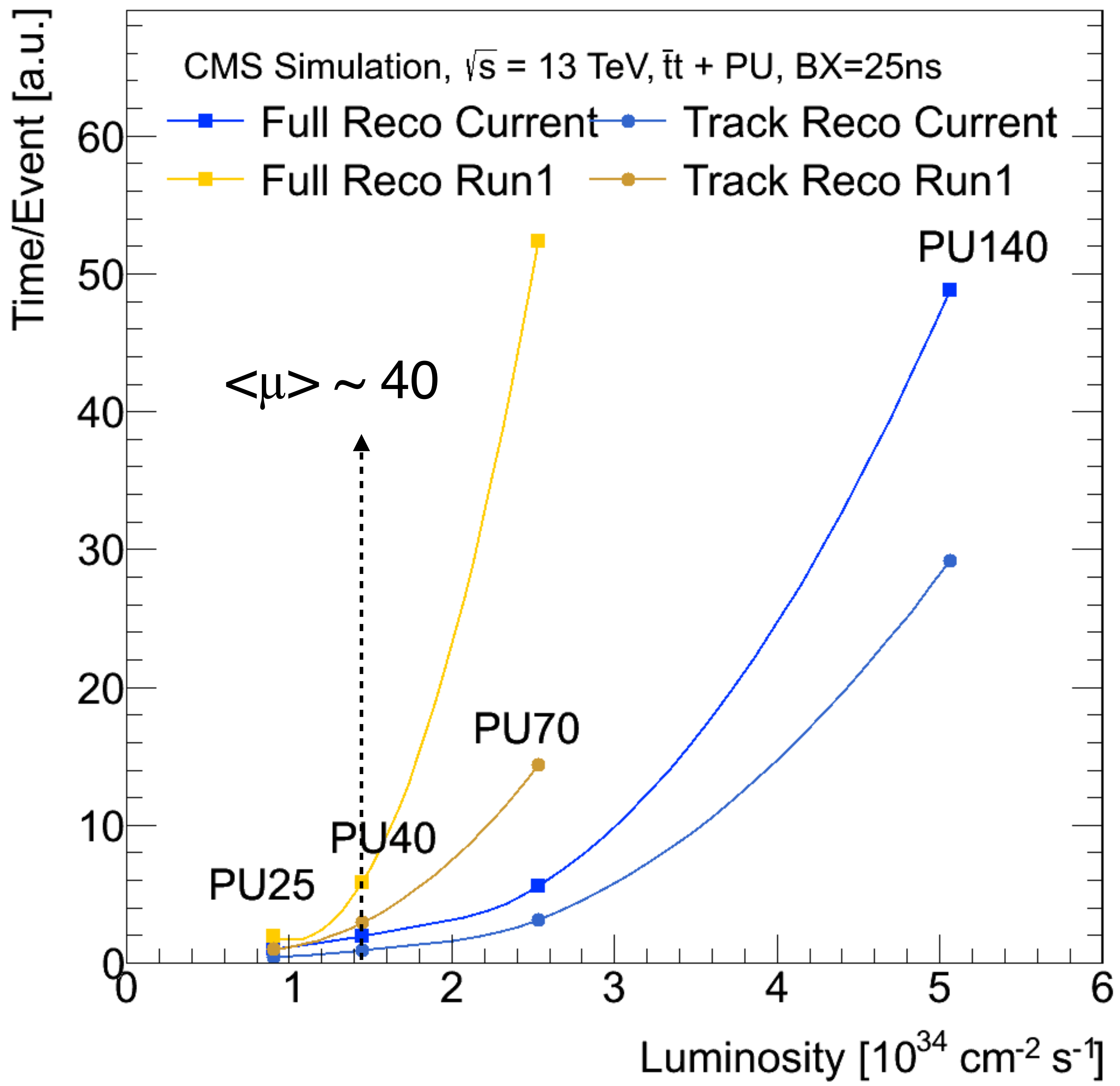
Hits from one particle



Hits from a fraction of particles
in HL-LHC environment

From CHEP2015 to CHEP2018

From CHEP2015 to CHEP2018



LHC Run-1 | LS 1 | LHC Run-2 | LS 2 | LHC Run-3

optimisation A

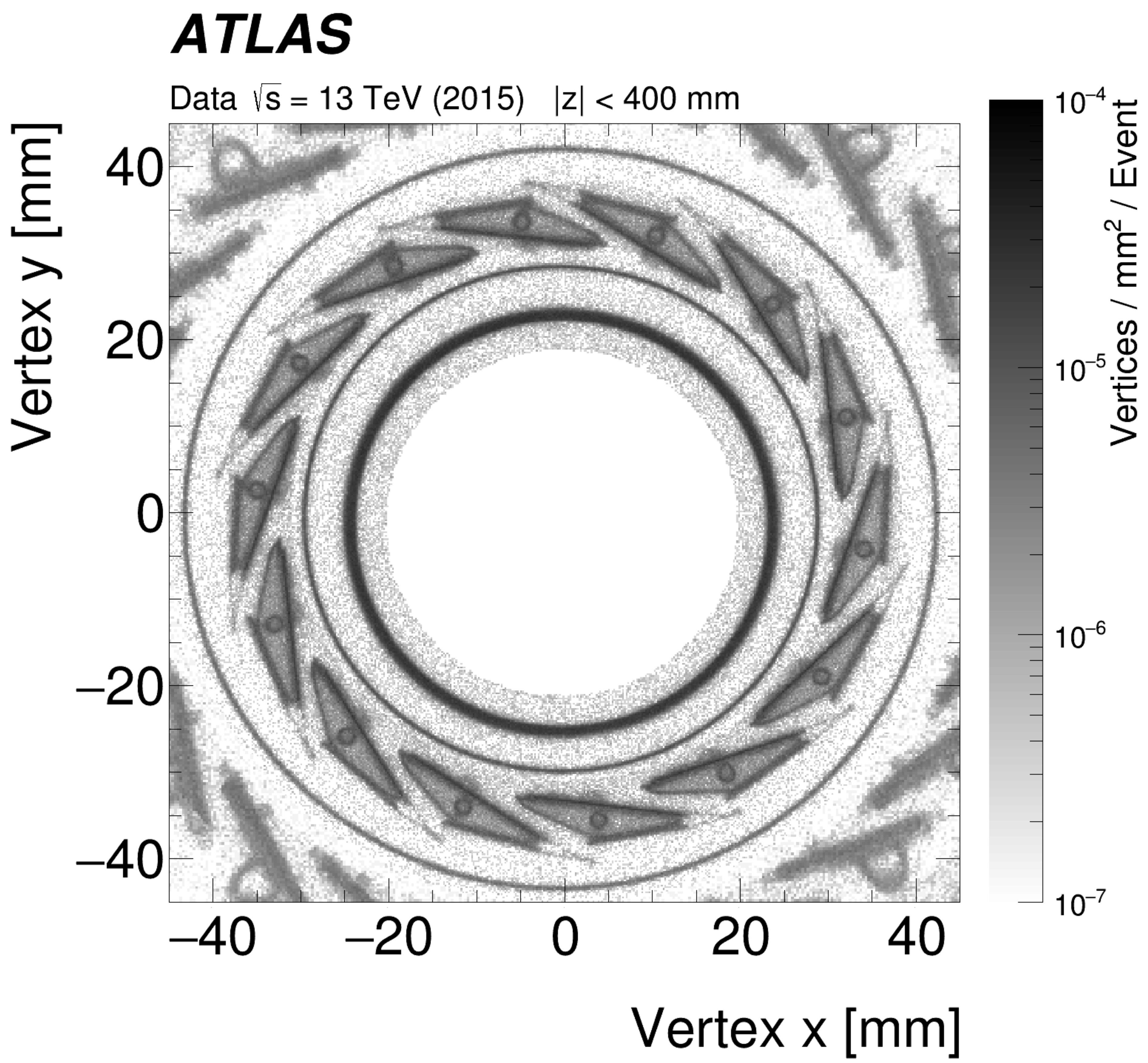
rewrite/redesign B

From CHEP2015 to CHEP2018

ATLAS Physics stream

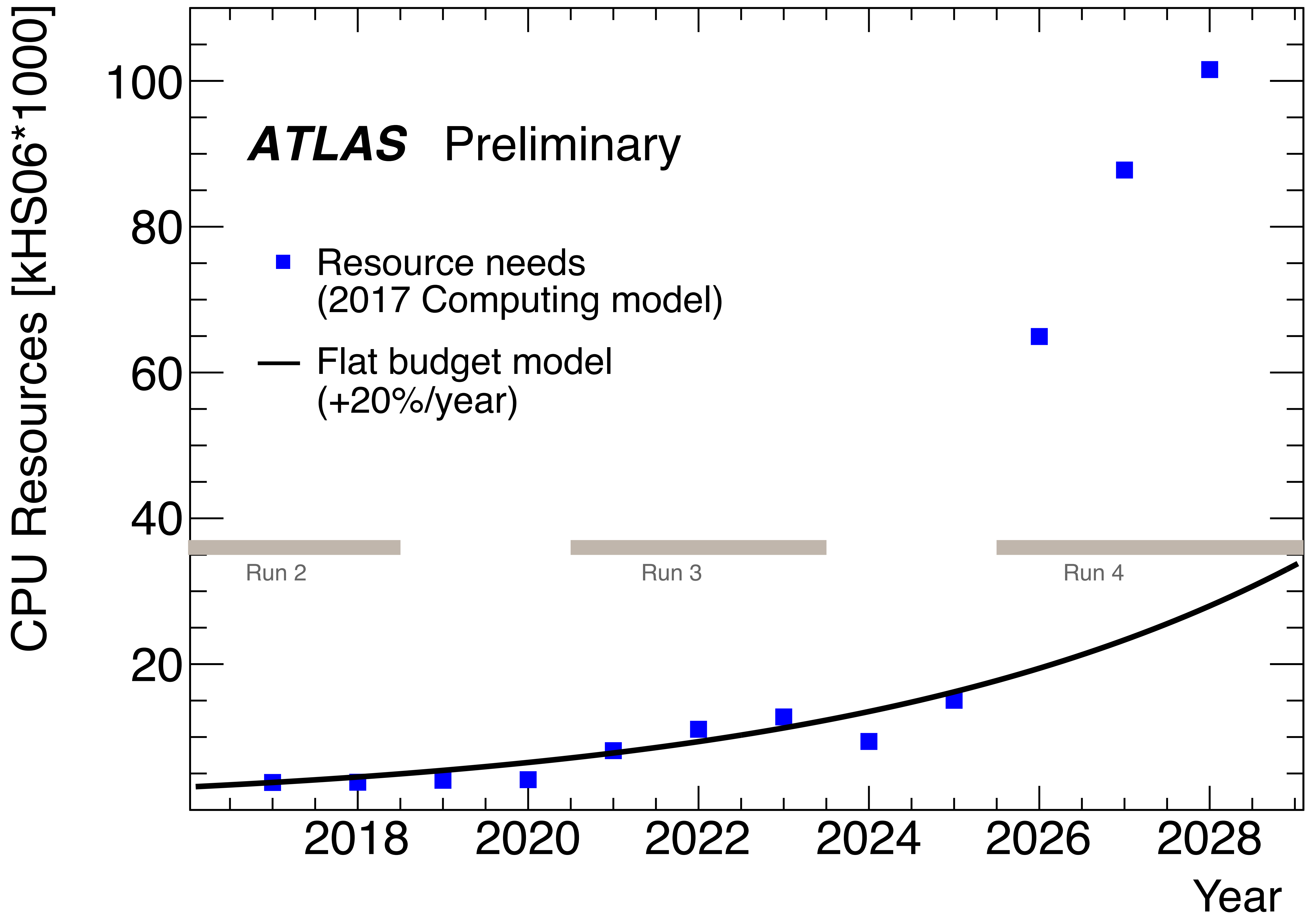
~ 3 * 10^{10} events

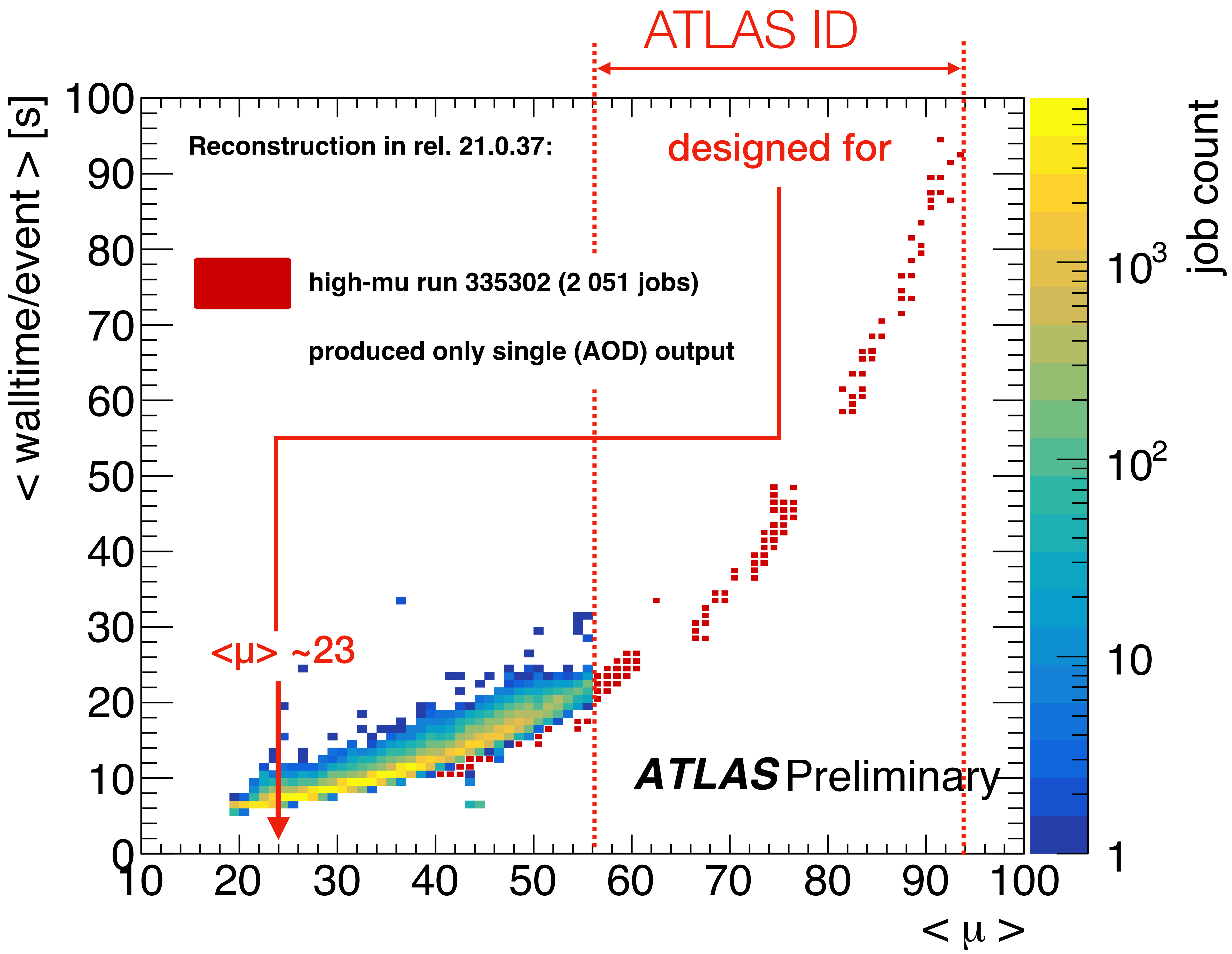
- ~ 10^{11} p-p collisions
- ~ 10^{13} tracks
- ~ 10^{14} track candidates
- ~ 10^{15} track seeds

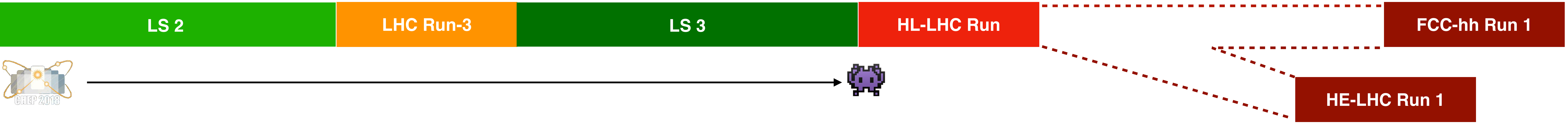


LHC Run-1	LS 1	LHC Run-2	LS 2	LHC Run-3

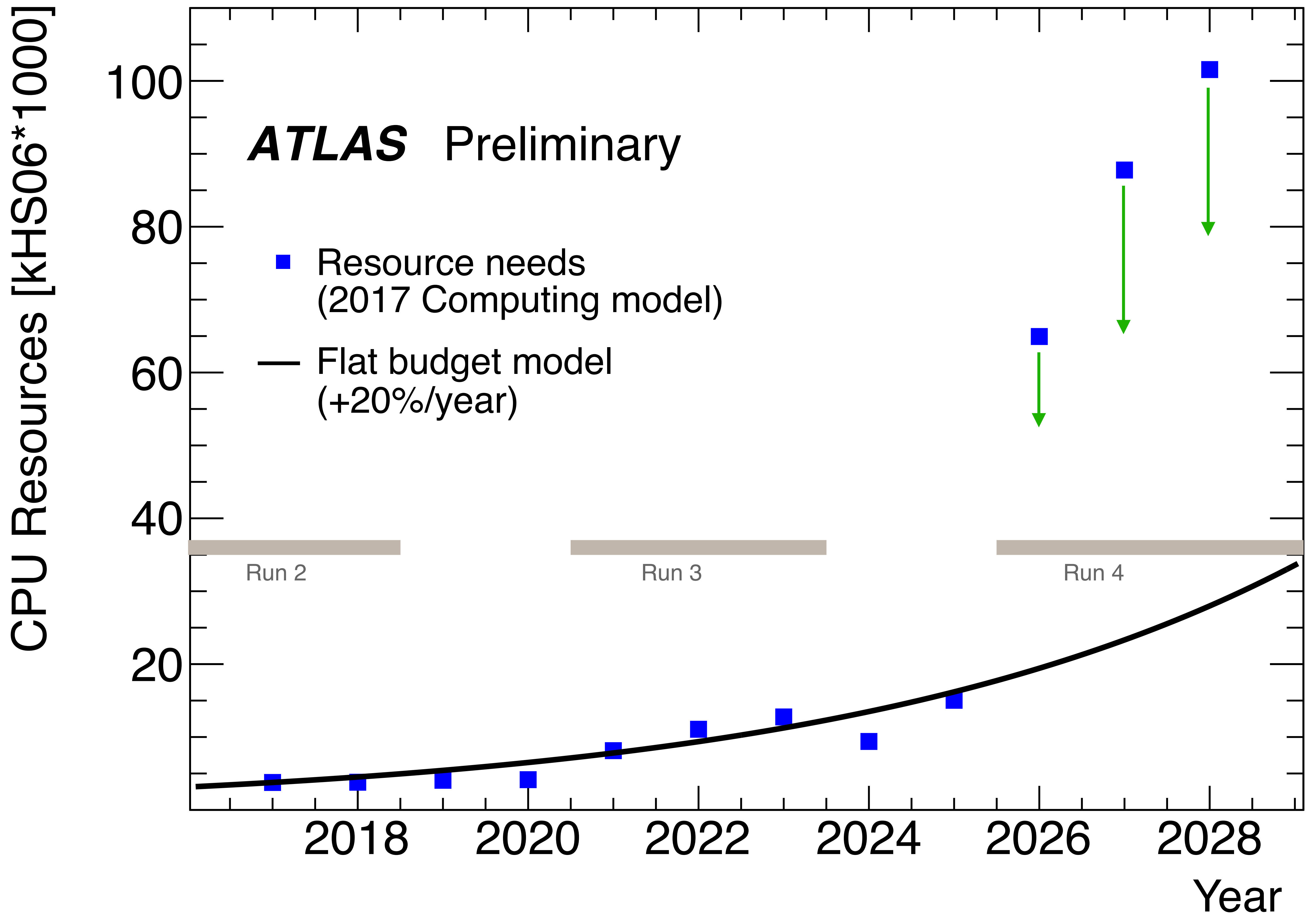
From CHEP2018 to CHEP2026

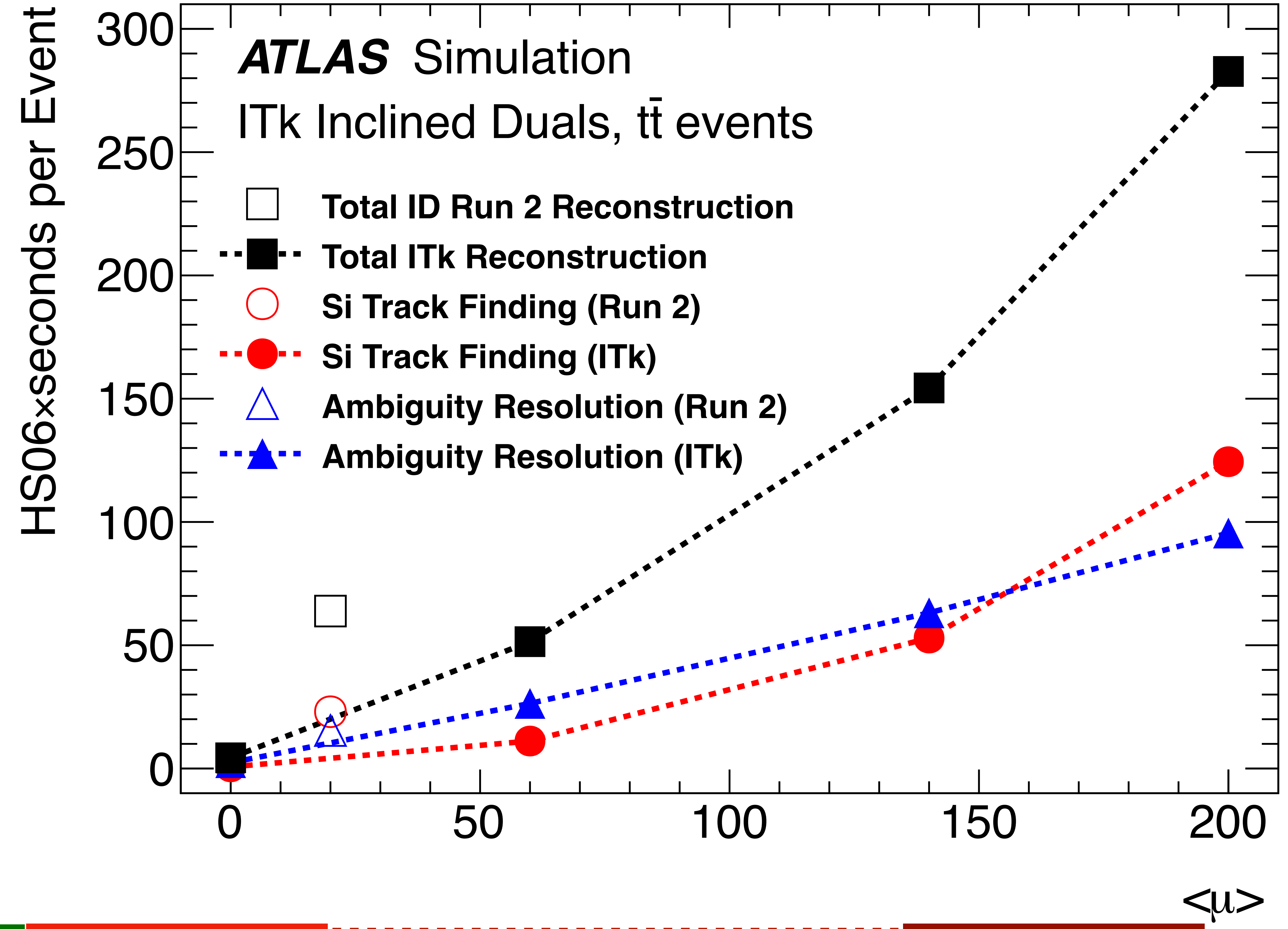


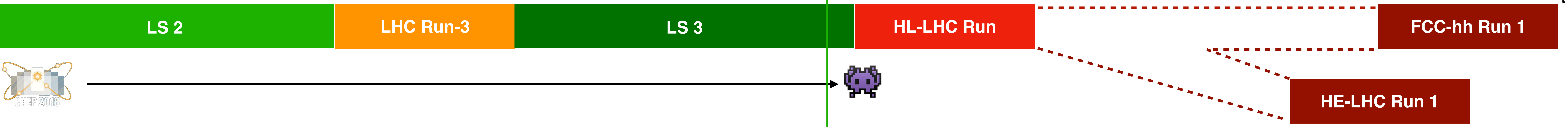




From CHEP2018 to CHEP2026

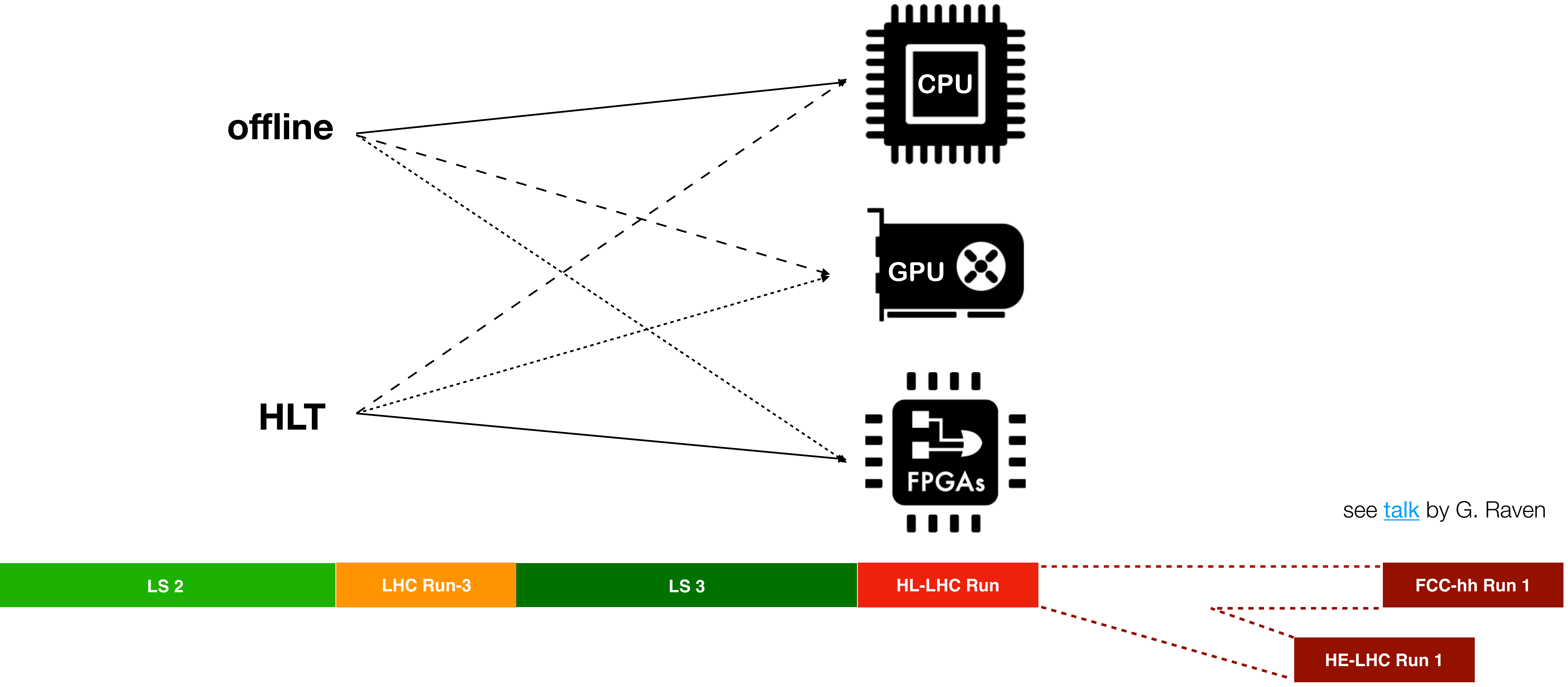






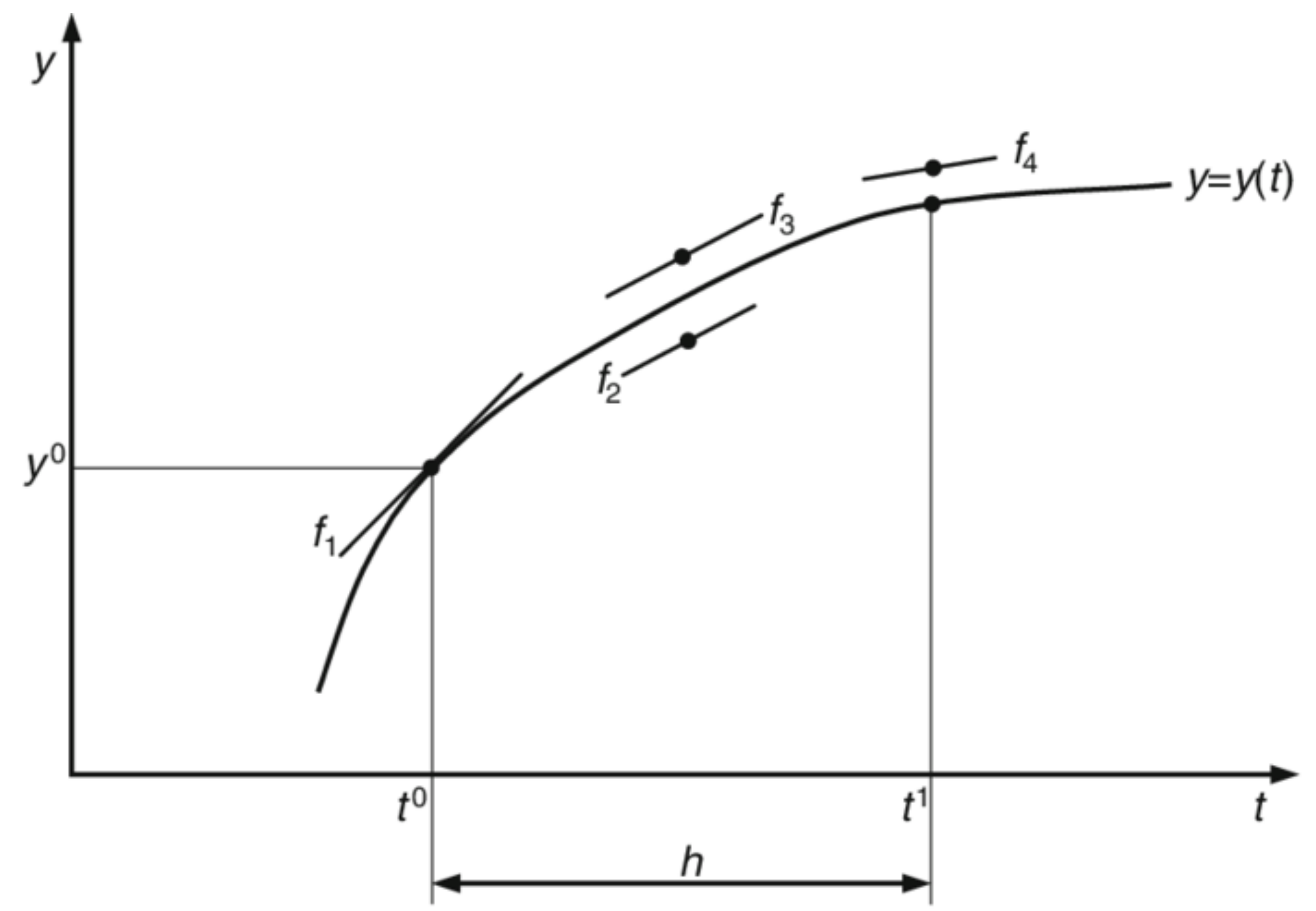
Upgraded detector & tighter tracking cuts

From CHEP2018 to CHEP2026

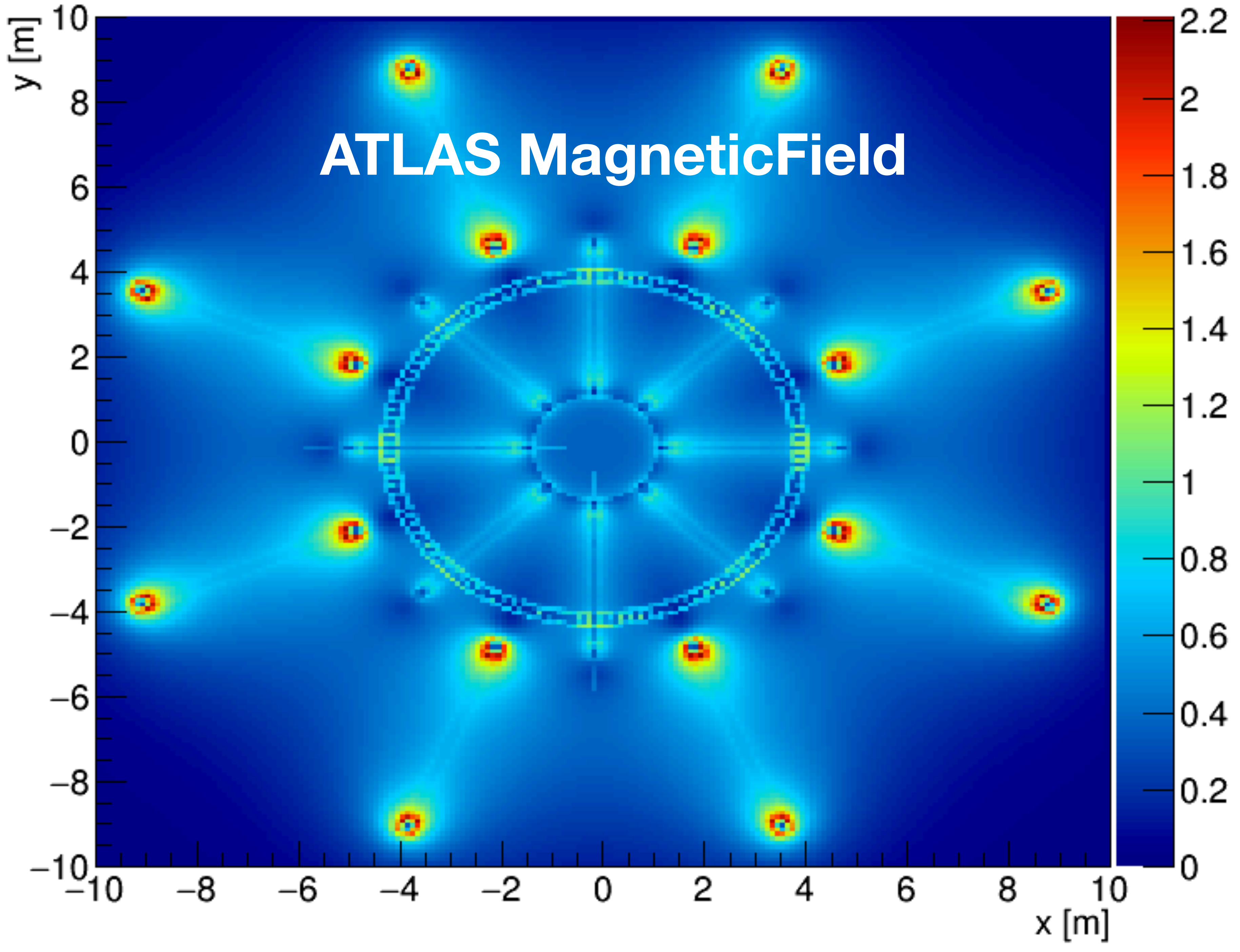


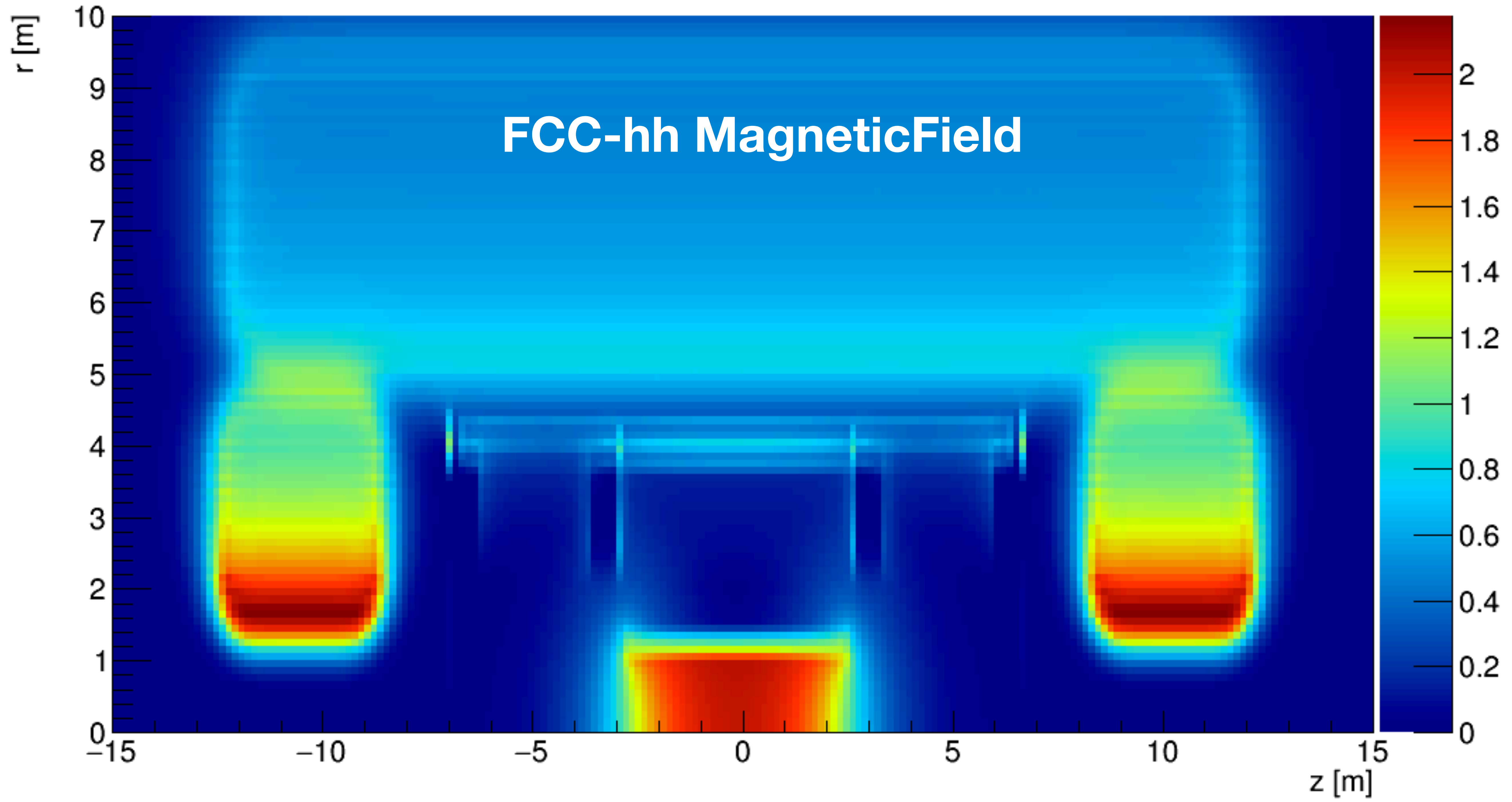
see talk by G. Raven

The story of a Runge-Kutta propagator

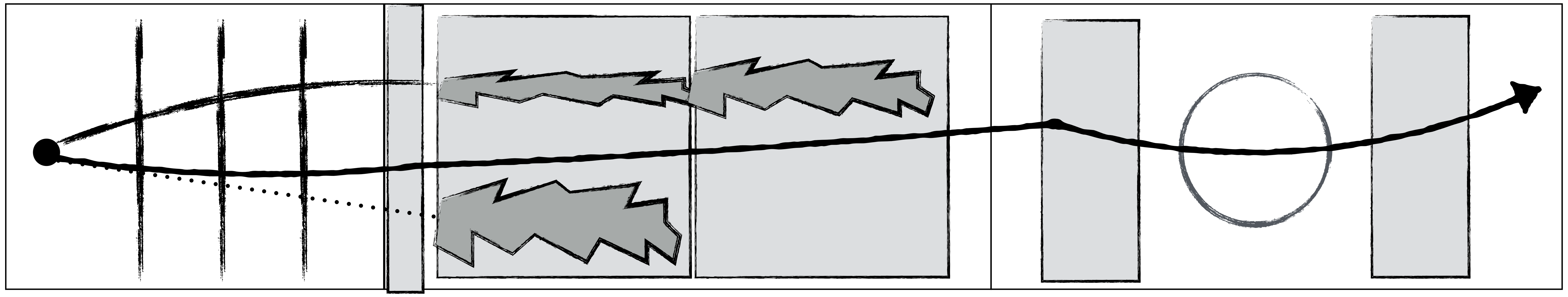


Numerical integration algorithm,
HEP usage for:
solving equation of motion through complex magnetic field
transporting the track covariance matrix





The story of a Runge-Kutta propagator




```
20      CALL THEFLSP ( IPR(3)   )   ! Track propogation through
                                    ! precision detectors
        CALL THEFILP (  1       )   ! Fill output track bank without TRT
        CALL THEFLST ( IPR(4)   )   ! Track propogation through TRT
        CALL THEFILT                ! Fill output track bank with TRT
        CALL THEBRE  ( IBREM    )   ! Brem. fit possibility investigation
        IF ( IBREM.NE.0  ) GO TO 20 ! Repeat fit with brem. conditions
                           GO TO 10 ! Go to next track candidate
30      CALL THERRO  (   IER    )   ! Test errors list
        CALL THELOOK                ! Tracks comparison
```

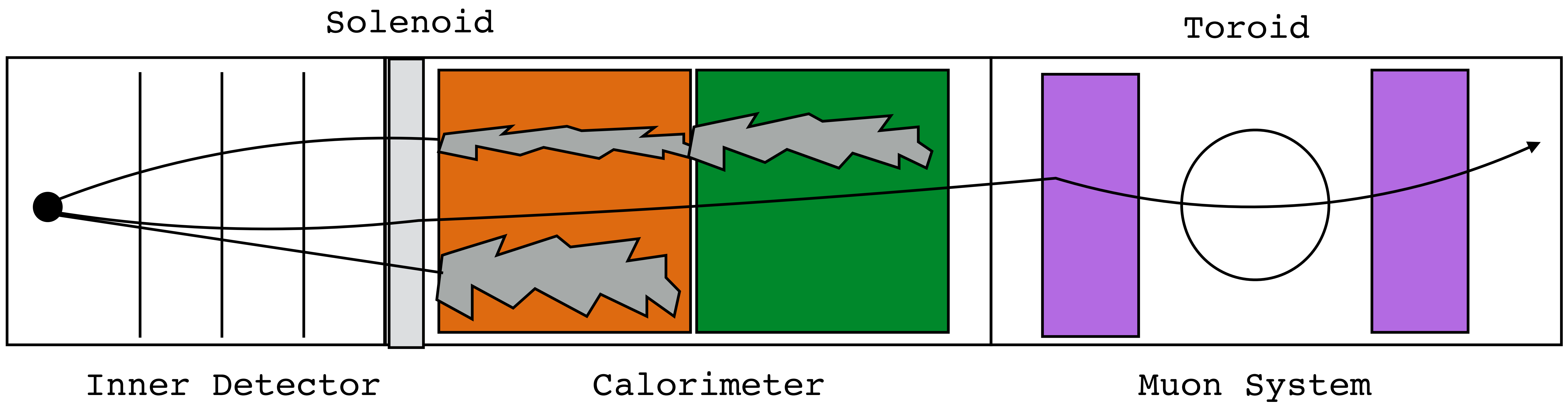
First implementations for ATLAS Track reconstruction

`atrecon` **(FORTRAN based)** based

propagation function

The **FORTRAN** times

The story of a Runge-Kutta propagator

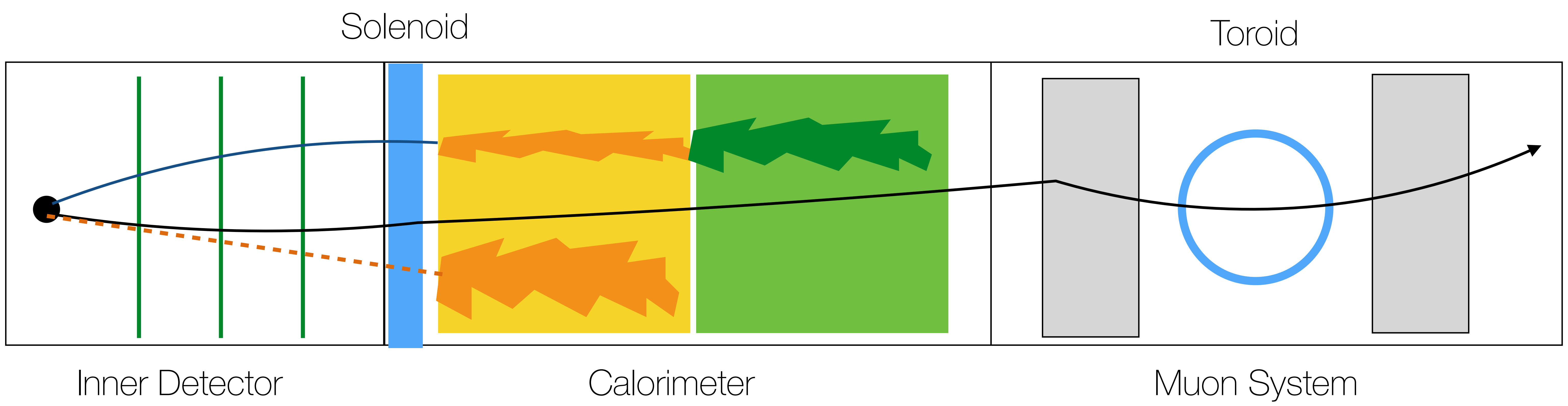


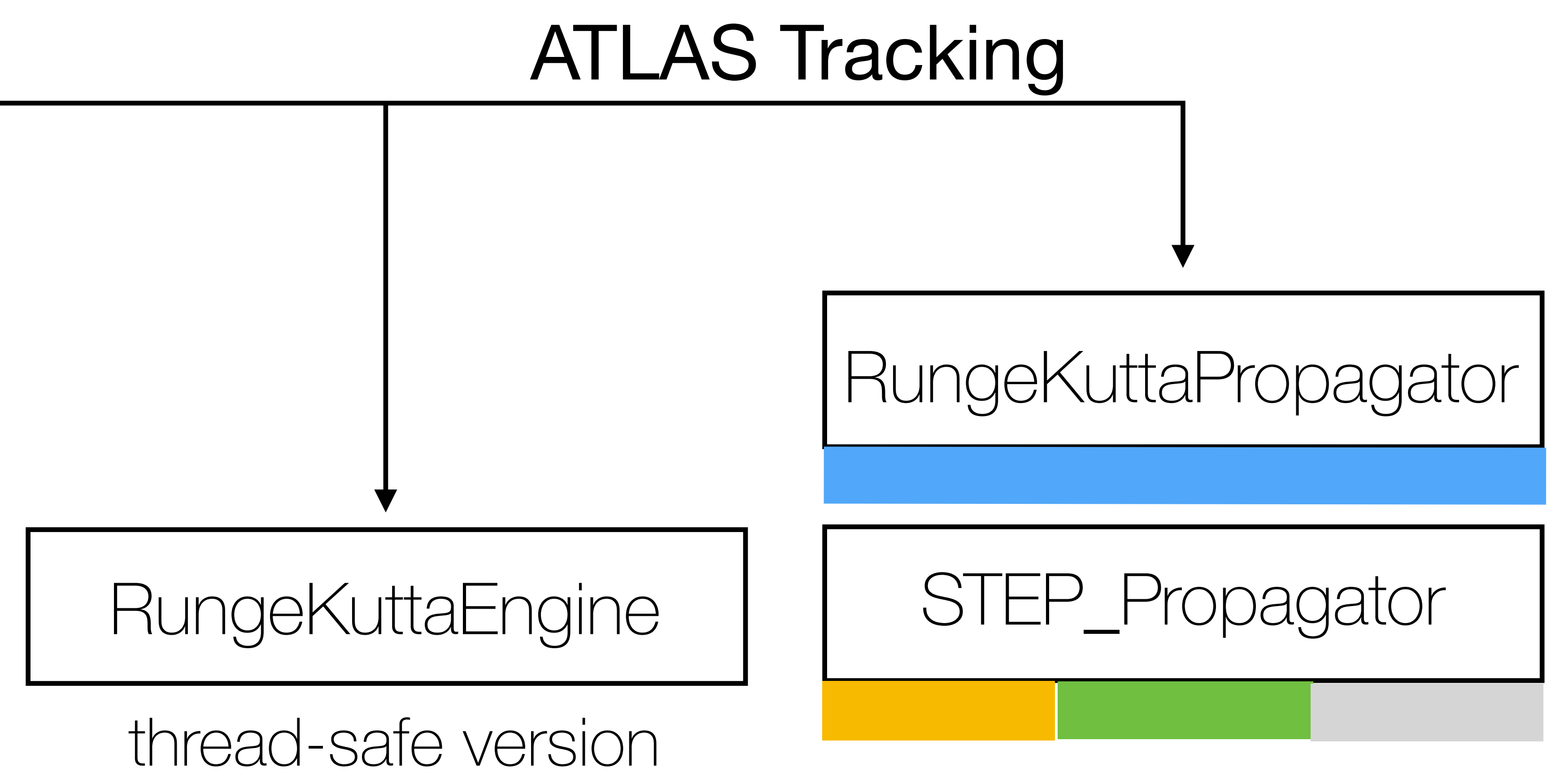

```
double s0 = P[ 7]*S[0]+P[ 8]*S[1]+P[ 9]*S[2];
double s1 = P[14]*S[0]+P[15]*S[1]+P[16]*S[2];
double s2 = P[21]*S[0]+P[22]*S[1]+P[23]*S[2];
double s3 = P[28]*S[0]+P[29]*S[1]+P[30]*S[2];
double s4 = P[35]*S[0]+P[36]*S[1]+P[37]*S[2];

P[ 7]-=(s0*P[ 3]); P[ 8]-=(s0*P[ 4]); P[ 9]-=(s0*P[ 5]);
P[10]-=(s0*P[42]); P[11]-=(s0*P[43]); P[12]-=(s0*P[44]);
P[14]-=(s1*P[ 3]); P[15]-=(s1*P[ 4]); P[16]-=(s1*P[ 5]);
P[17]-=(s1*P[42]); P[18]-=(s1*P[43]); P[19]-=(s1*P[44]);
P[21]-=(s2*P[ 3]); P[22]-=(s2*P[ 4]); P[23]-=(s2*P[ 5]);
P[24]-=(s2*P[42]); P[25]-=(s2*P[43]); P[26]-=(s2*P[44]);
P[28]-=(s3*P[ 3]); P[29]-=(s3*P[ 4]); P[30]-=(s3*P[ 5]);
P[31]-=(s3*P[42]); P[32]-=(s3*P[43]); P[33]-=(s3*P[44]);
P[35]-=(s4*P[ 3]); P[36]-=(s4*P[ 4]); P[37]-=(s4*P[ 5]);
P[38]-=(s4*P[42]); P[39]-=(s4*P[43]); P[40]-=(s4*P[44]);
```

Move to C(++)
- `Athena-Gaudi` framework
- xKalman monolithic C(++) program

The **C** times

The story of a Runge-Kutta propagator






```
double s0 = P[ 7]*S[0]+P[ 8]*S[1]+P[ 9]*S[2];
double s1 = P[14]*S[0]+P[15]*S[1]+P[16]*S[2];
double s2 = P[21]*S[0]+P[22]*S[1]+P[23]*S[2];
double s3 = P[28]*S[0]+P[29]*S[1]+P[30]*S[2];
double s4 = P[35]*S[0]+P[36]*S[1]+P[37]*S[2];

P[ 7]-=(s0*P[ 3]); P[ 8]-=(s0*P[ 4]); P[ 9]-=(s0*P[ 5]);
P[10]-=(s0*P[42]); P[11]-=(s0*P[43]); P[12]-=(s0*P[44]);
P[14]-=(s1*P[ 3]); P[15]-=(s1*P[ 4]); P[16]-=(s1*P[ 5]);
P[17]-=(s1*P[42]); P[18]-=(s1*P[43]); P[19]-=(s1*P[44]);
P[21]-=(s2*P[ 3]); P[22]-=(s2*P[ 4]); P[23]-=(s2*P[ 5]);
P[24]-=(s2*P[42]); P[25]-=(s2*P[43]); P[26]-=(s2*P[44]);
P[28]-=(s3*P[ 3]); P[29]-=(s3*P[ 4]); P[30]-=(s3*P[ 5]);
P[31]-=(s3*P[42]); P[32]-=(s3*P[43]); P[33]-=(s3*P[44]);
P[35]-=(s4*P[ 3]); P[36]-=(s4*P[ 4]); P[37]-=(s4*P[ 5]);
P[38]-=(s4*P[42]); P[39]-=(s4*P[43]); P[40]-=(s4*P[44]);
```

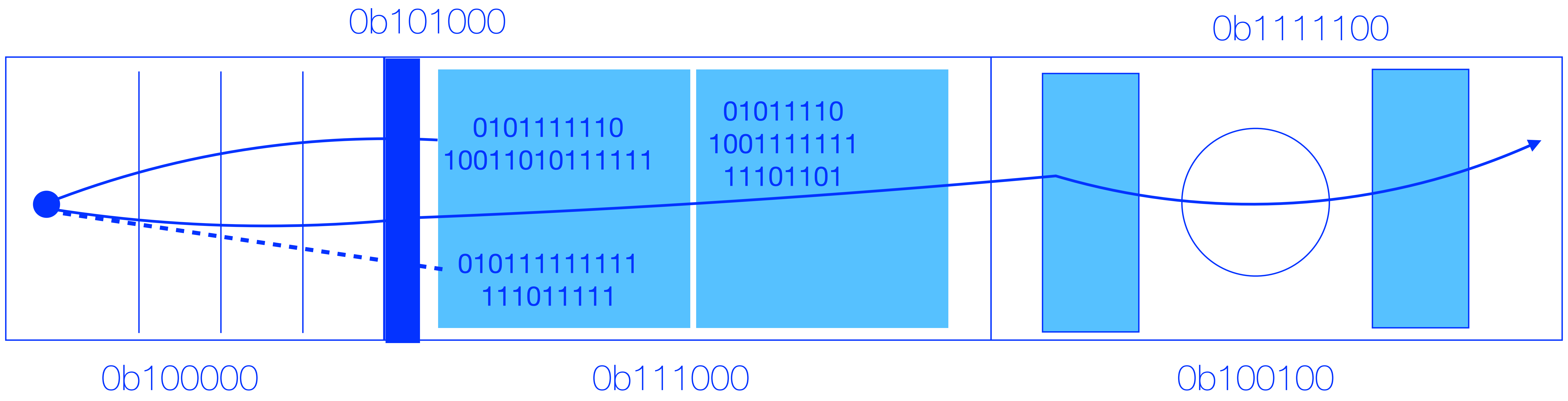
Geant4

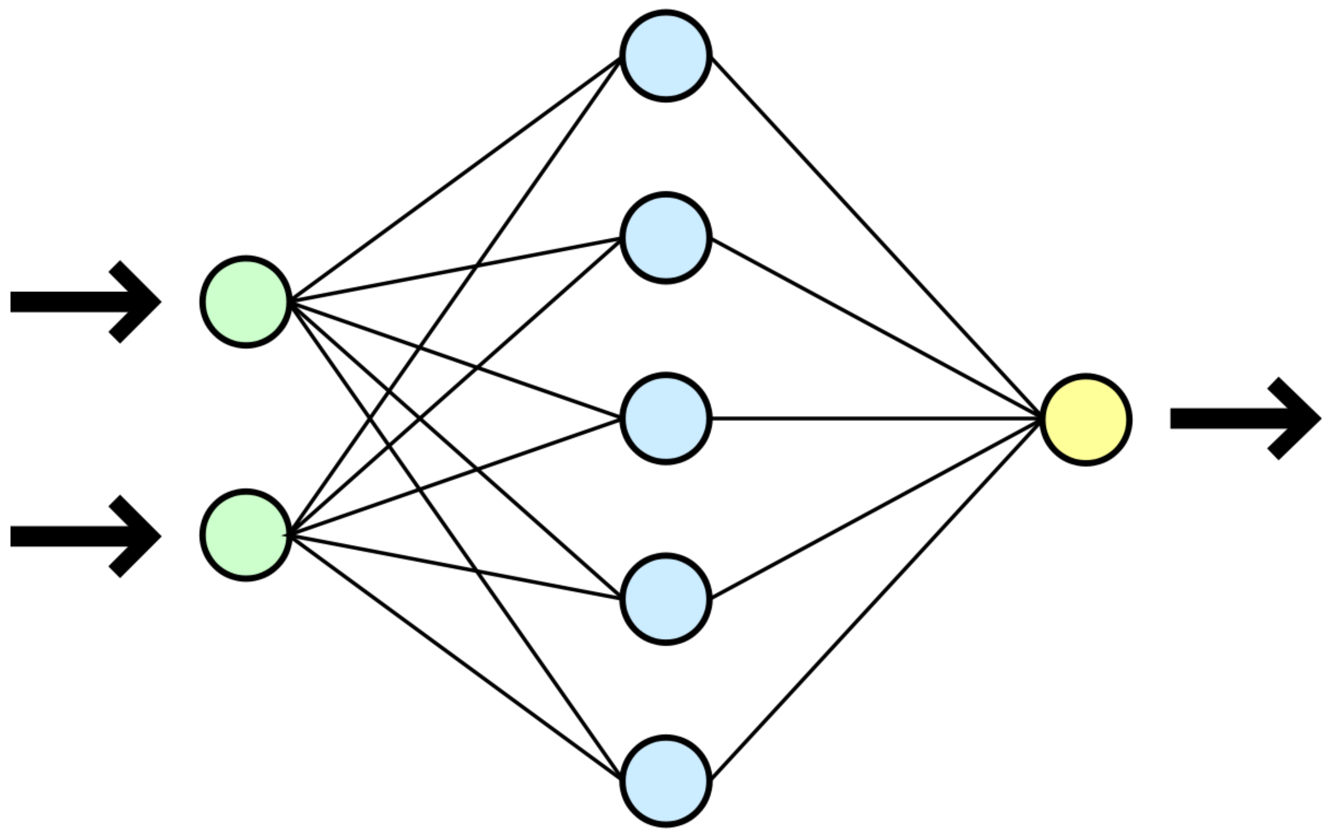
G4AtlasRk

```
// New direction
//
Po[3] = A[0]+S6*(K1[0]+K4[0]+2.*(K2[0]+K3[0]));
Po[4] = A[1]+S6*(K1[1]+K4[1]+2.*(K2[1]+K3[1]));
Po[5] = A[2]+S6*(K1[2]+K4[2]+2.*(K2[2]+K3[2]));

// Errors
//
Err[3] = S*fabs(K1[0]-K2[0]-K3[0]+K4[0]);
Err[4] = S*fabs(K1[1]-K2[1]-K3[1]+K4[1]);
Err[5] = S*fabs(K1[2]-K2[2]-K3[2]+K4[2]);
Err[0] = S*Err[3]                        ;
Err[1] = S*Err[4]                        ;
Err[2] = S*Err[5]                        ;
Err[3]*= m_mom                           ;
Err[4]*= m_mom                           ;
Err[5]*= m_mom                           ;
```

The **present** times

The story of a Runge-Kutta propagator





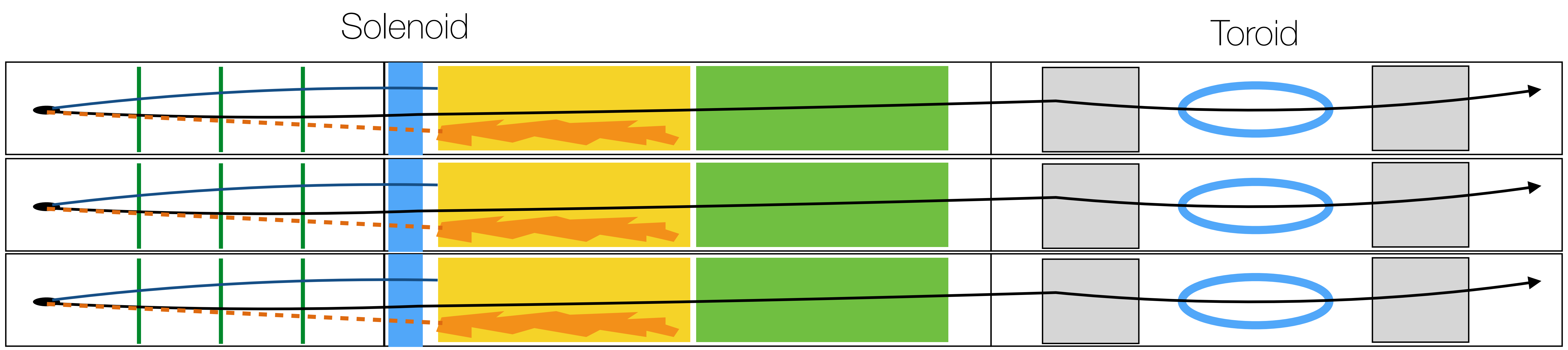
NN based Kalman filtering
AI pattern recognition techniques

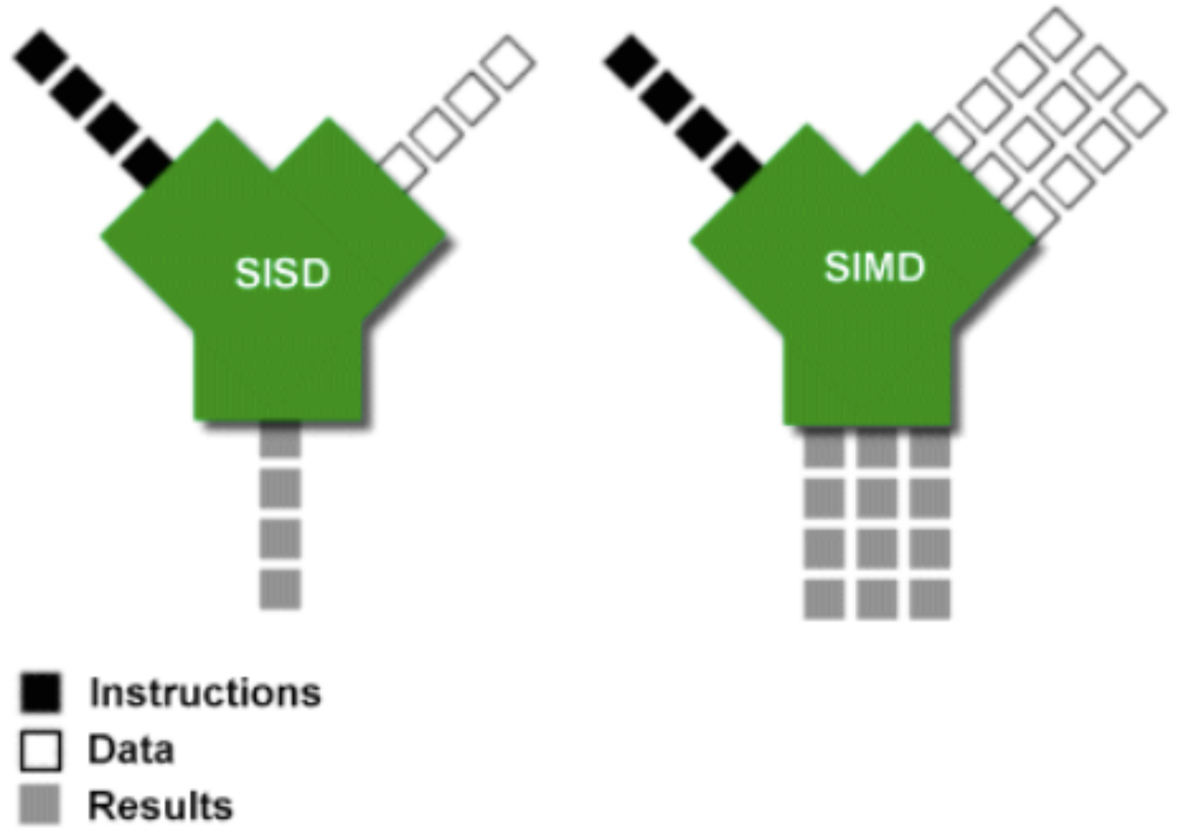
see TrackML [talk](#) by M. Kiehn

(and the entire Thursday session)

One possible future …

The story of a Runge-Kutta propagator



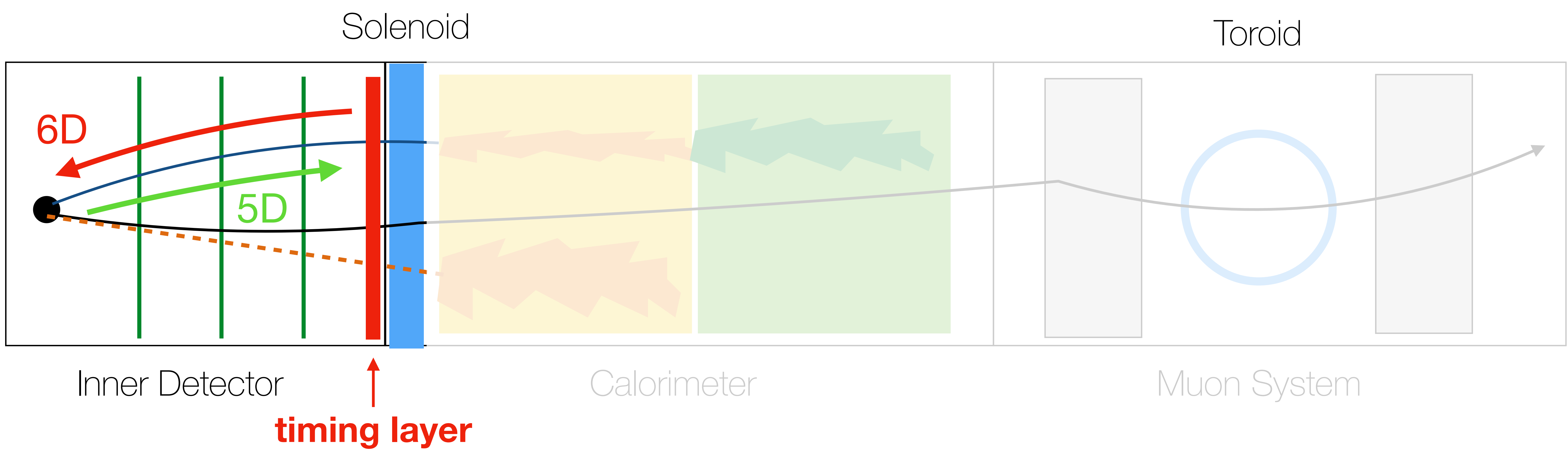


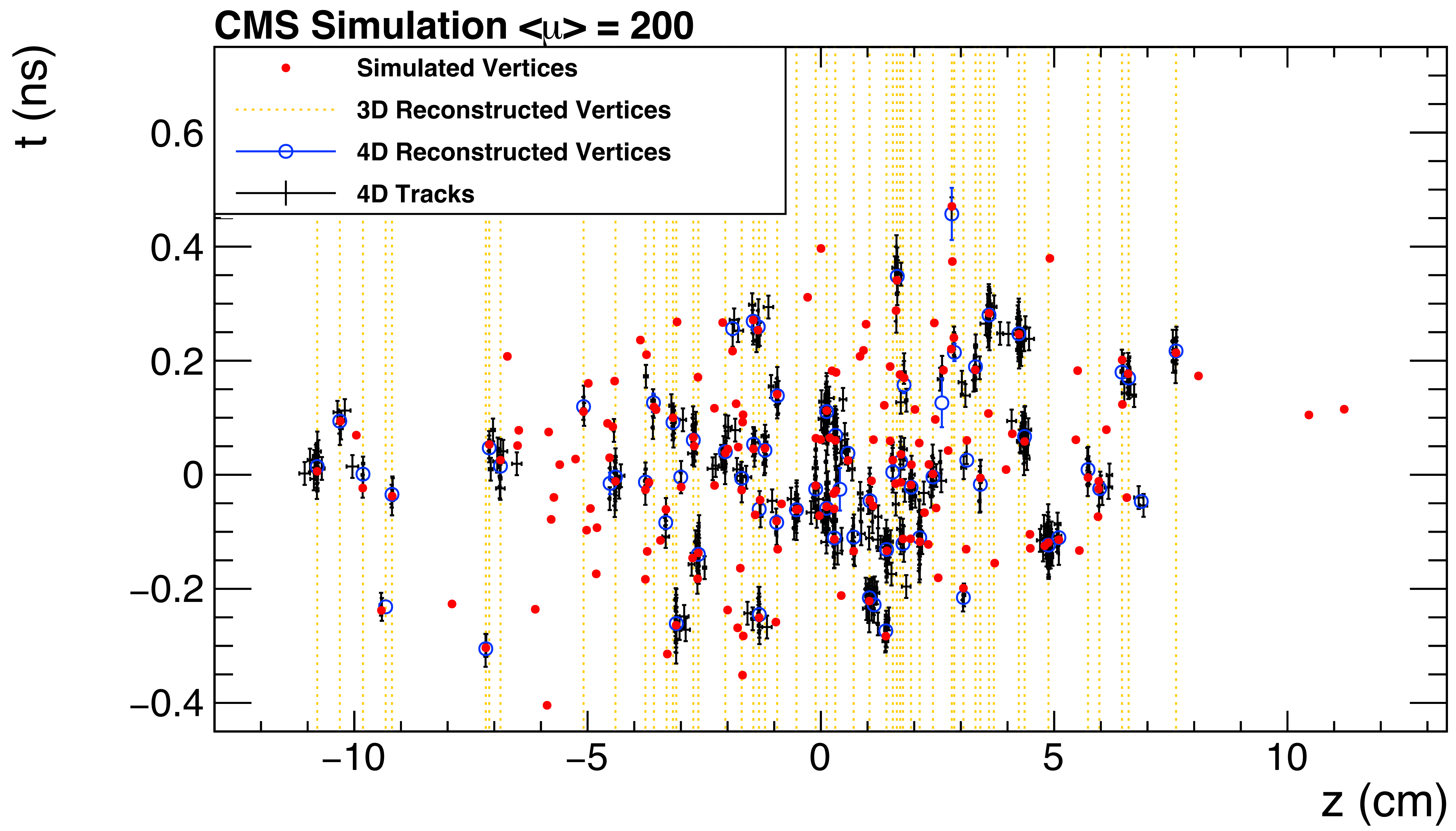
trend to concurrent algorithmic approaches
e.g. `AthenaMT` framework for ATLAS

10-15(+) years old HEP code in general not thread safe
- vectorization is only little used ins **offline** code
- caching is/was often used for performance

Another path … or better many other paths

Future detector technologies



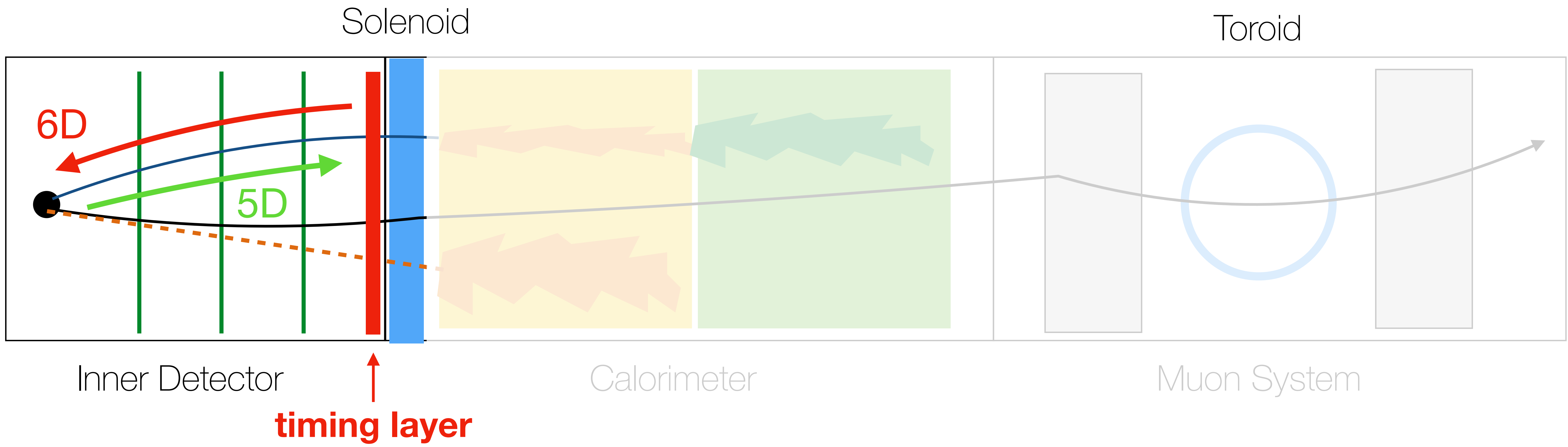


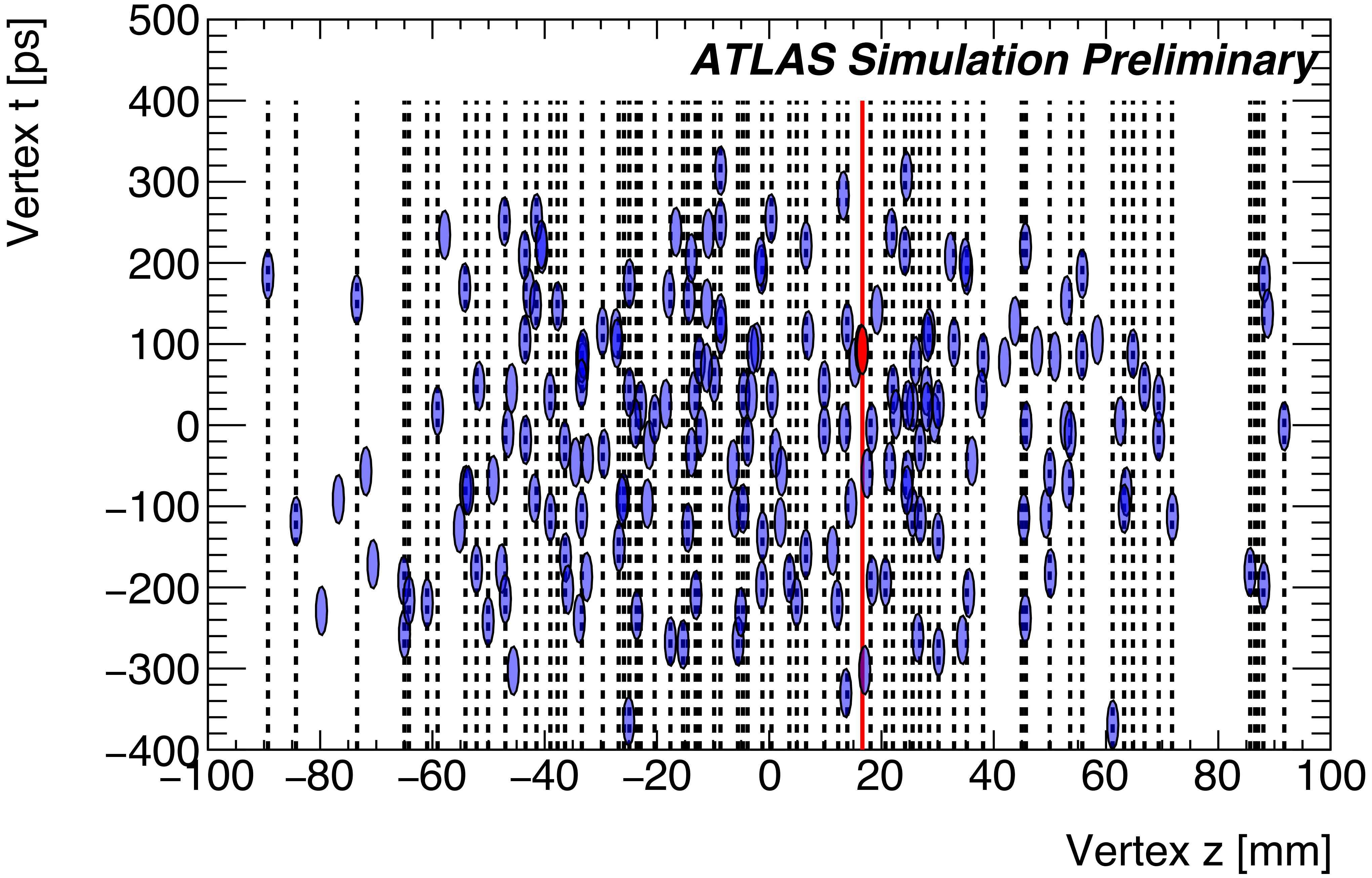
ATLAS and **CMS** will insert first timing detectors for HL-LH

- currently time tagging
- correct treatment: 5D tracking becomes 6D (math & code development)

see CTD2018 talk by L. Gray

Future detector technologies





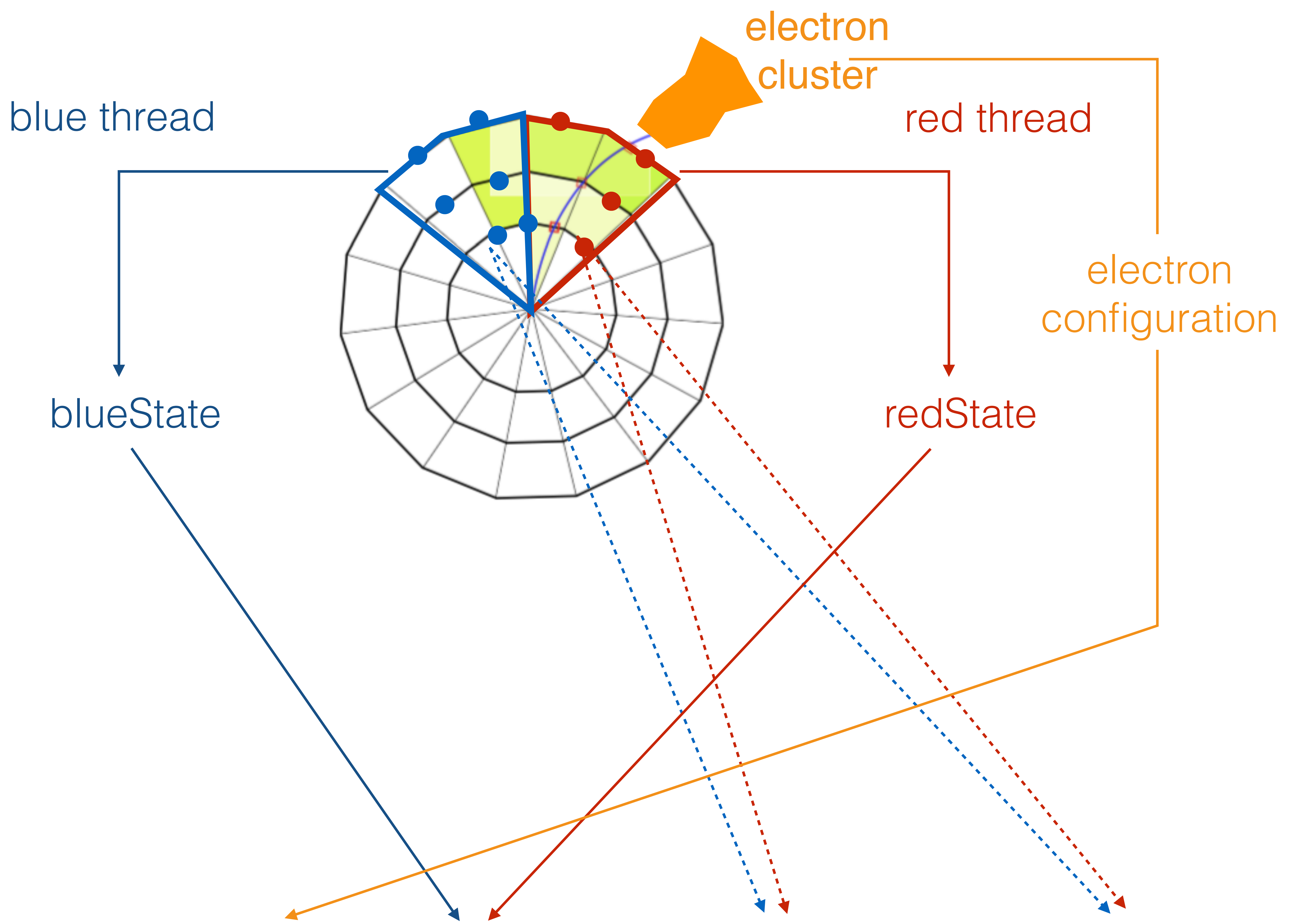
ATLAS and CMS will insert first timing detectors for HL-LH

- currently time tagging
- correct treatment: 5D tracking becomes 6D (math & code development)

see CTD2018 talk by L. Gray

R&D Research, Development

Turn-key software



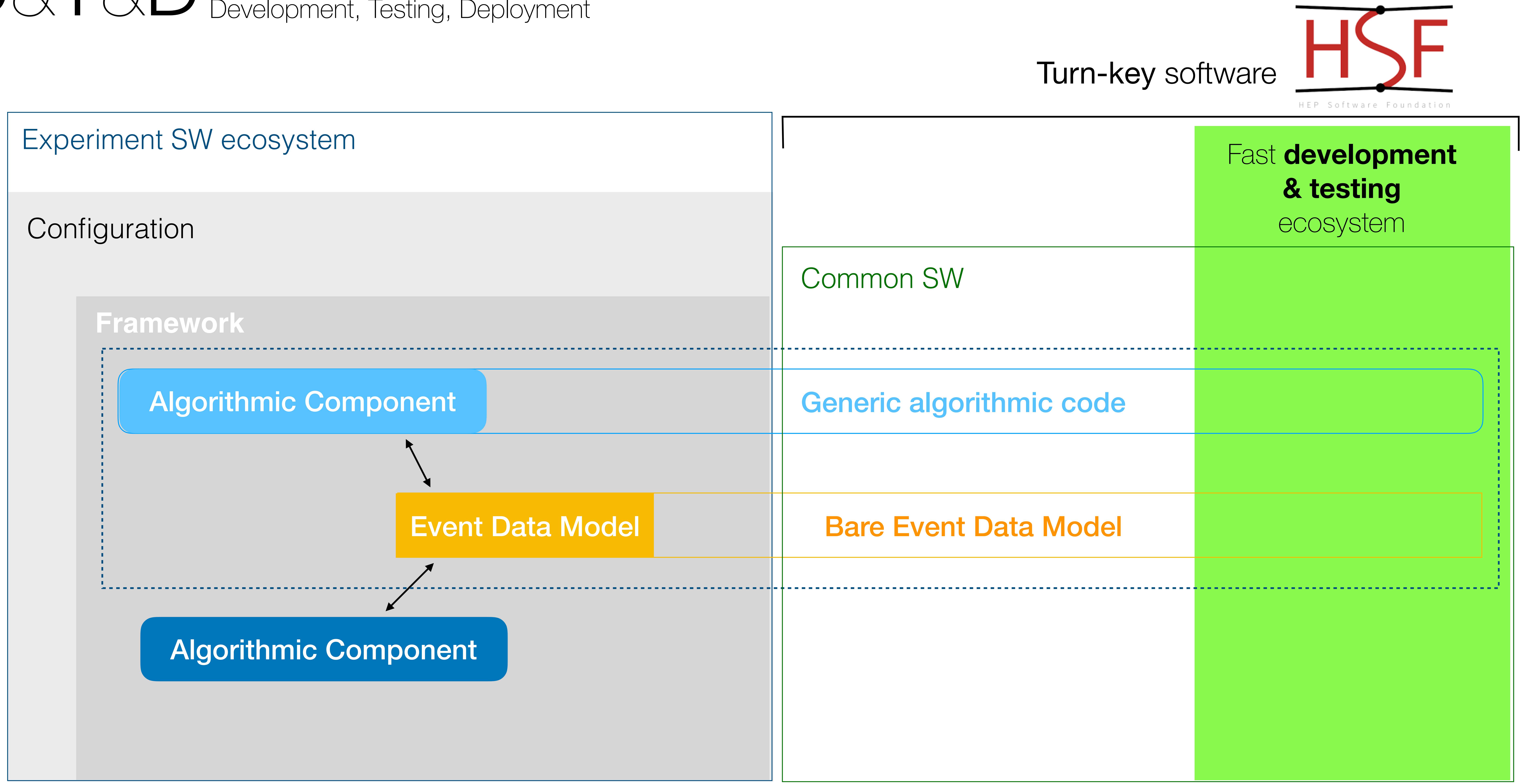

```
seedFinder(const Config& pConfig, State& pState, iterator_type start, iterator_type end)
```

```
SeedContainer
```

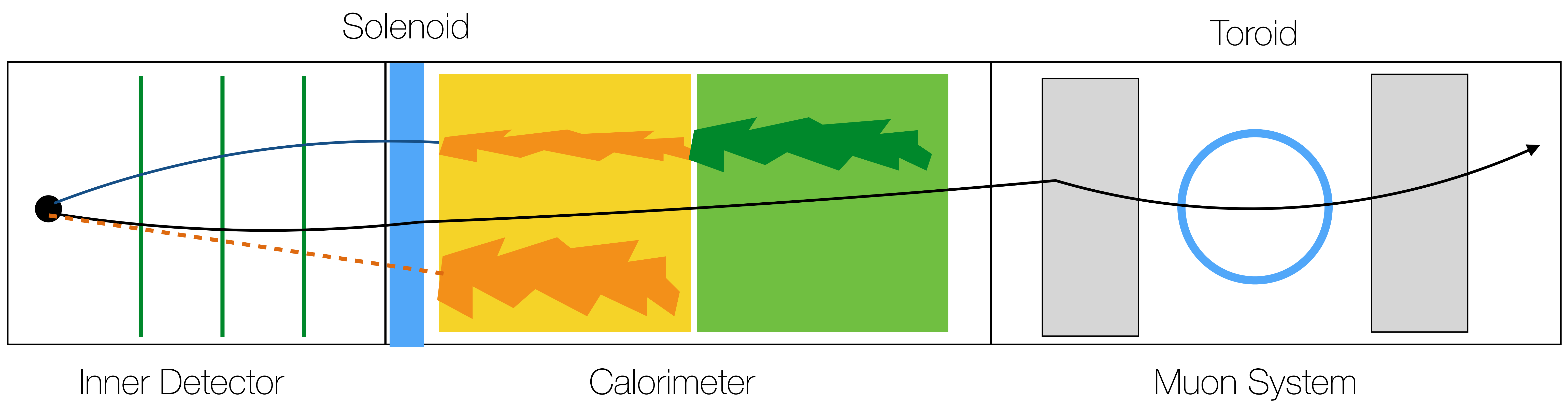
R&D testbed

- establish toolkit with thread safe tracking tools
- build up concurrent pattern recognition chains
- test physics driven recognition configuration

D&T&D Development, Testing, Deployment



ATLAS Detector & Tracking Software Structure



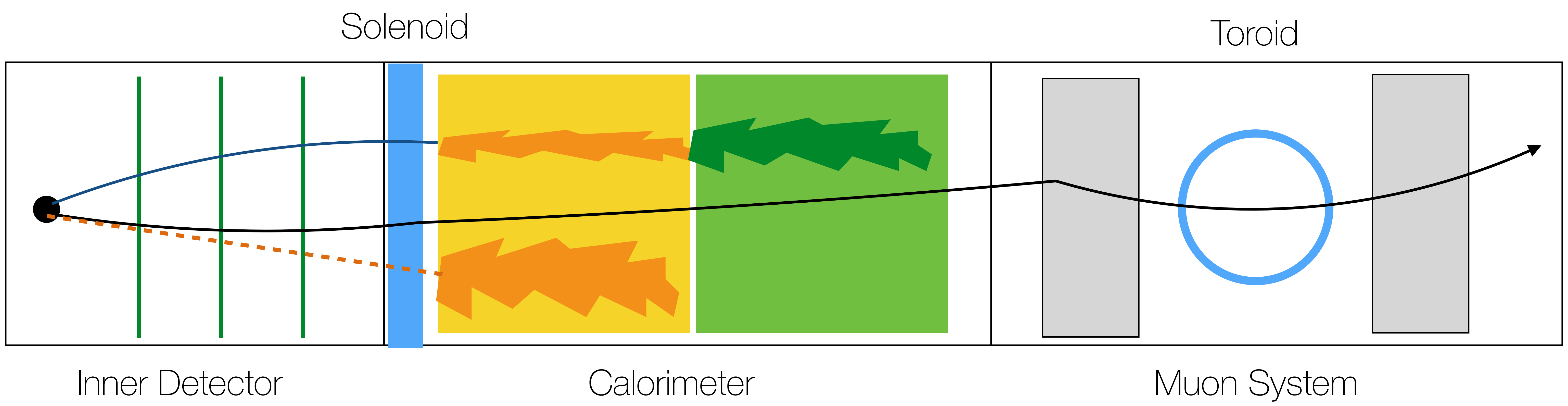
Tracking					
Geometry	Event Data Model	Extrapolation	Fitting	Calibration, general	Alignment

InnerDetector		
EDM	Geometry	Tools

Calorimeter	
EDM	Geometry

MuonSpectrometer		
EDM	Geometry	Tools

Common structure motivation



Generic solutions for general tasks

Tracking					
Geometry	Event Data Model	Extrapolation	Fitting	Calibration, general	Alignment

Keep detector expertise where it belongs

InnerDetector		
EDM	Geometry	Tools

Digitization, Calibration, Error Parameterization, …

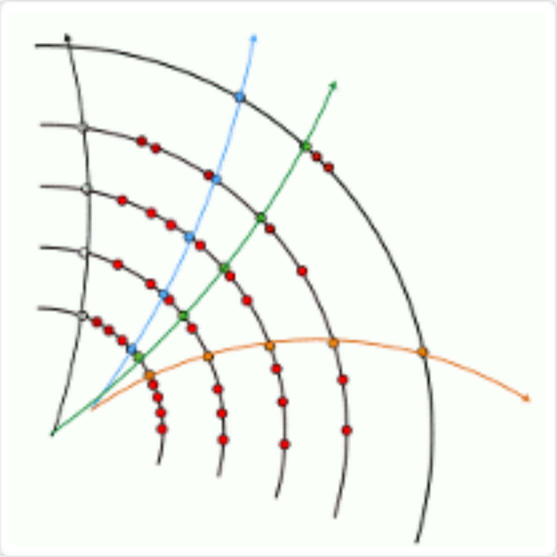
We

Turn-key software

ACTS | Home | About | Modules | Guides | Clients | Search | Gitlab

- A Common Tracking Software (ACTS)
- Latest release: v0.06.00
- Introduction
- Mailing list
- Repository structure
- acts-core
- acts-fatras
- acts-data
- acts-framework
- Releases
- History
- License and authors

A Common Tracking Software (ACTS)



1. Introduction
2. Repository structure
3. Releases
4. License and authors

https://acts.web.cern.ch

Prerequisites

Only few dependencies are required to build the Core library of Acts. A list of prerequisites required is given below with the version numbers indicating which versions were tested. Older versions may or may not work, feedback is very welcome.

The following dependencies are required:

- A C++14 compatible compiler, e.g. gcc (>= 6.2) or clang (>= 4.0)
- cmake (>= 3.7)
- boost (>= 1.62, with `program_options` and `unit_test_framework`)
- Eigen (>= 3.2.9)

Minimal requirements for **core** functionality
- first test deployment in ATLAS was extremely simple
- extended with plugins for ROOT, DD4Hep, Geant4, etc.

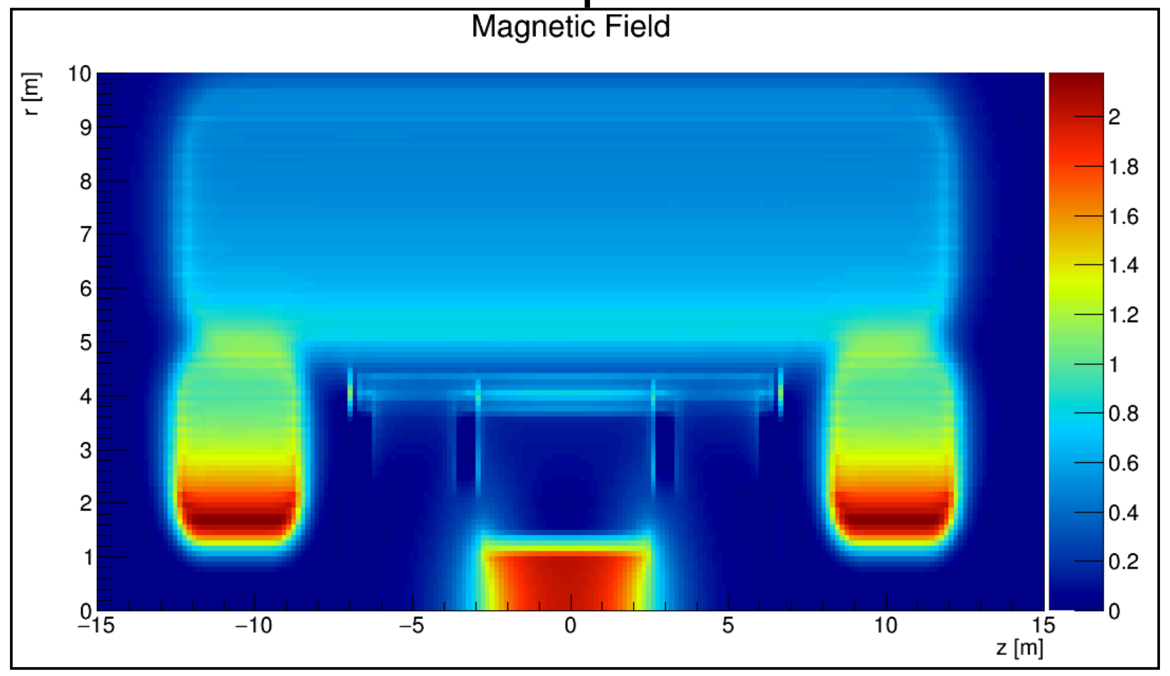
Repository&Testing

Turn-key software

Repository structure

https://gitlab.cern.ch/acts/

- **acts-core** ← Generic algorithmic code; Bare Event Data Model
- **acts-framework** ← Fast **development & testing** ecosystem
- **acts-data** ← Magnetic Field
- acts-fatras ← fast track simulation extension



Fast **development & testing** ecosystem

algorithms & tools are tested for thread safety and performance

Unit Test coverage

	Exec	Coverage	https://acts.web.cern.ch/ACTS/coverage/		
Date:	**2018-06-13**	**Lines:**	**2951**	**4708**	**62.7%**
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	**3091**	**22905**	**13.5%**

see also talk by H. Graslan

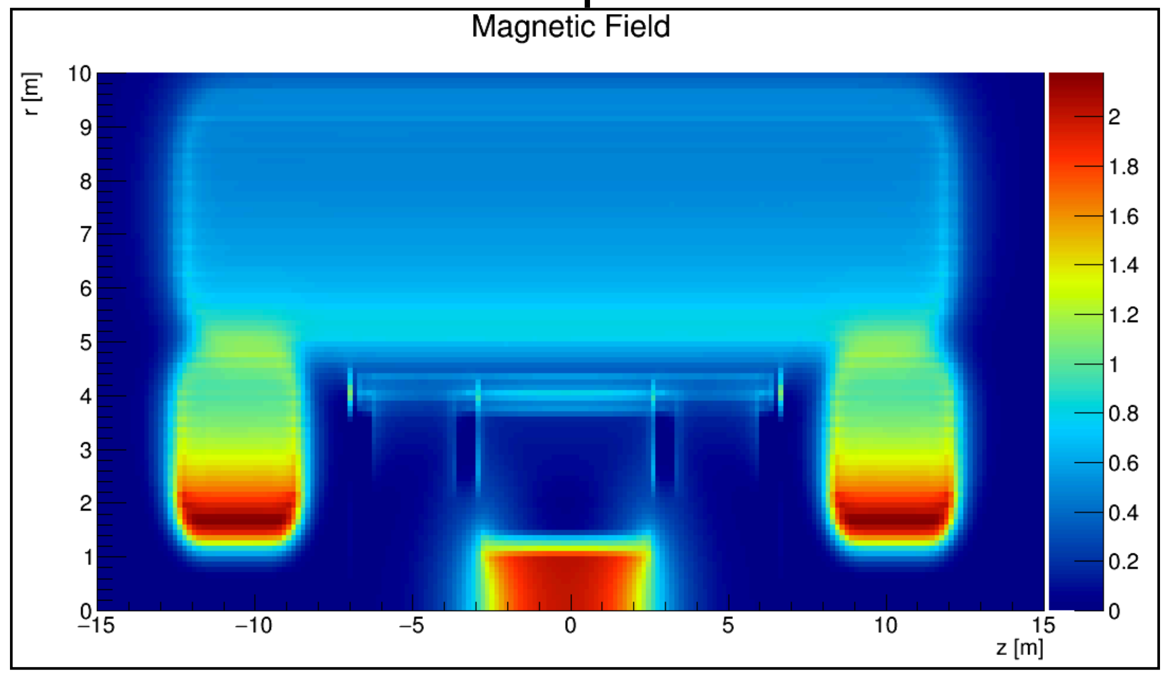
Repository&Testing

Turn-key software

Repository structure

https://gitlab.cern.ch/acts/

- **acts-core** ← Generic algorithmic code; Bare Event Data Model
- **GaudiHive** ← Fast **development & testing** ecosystem
- **acts-data** ← Magnetic Field
- acts-fatras ← fast track simulation extension



Fast **development & testing** ecosystem

algorithms & tools are tested for thread safety and performance

Unit Test coverage

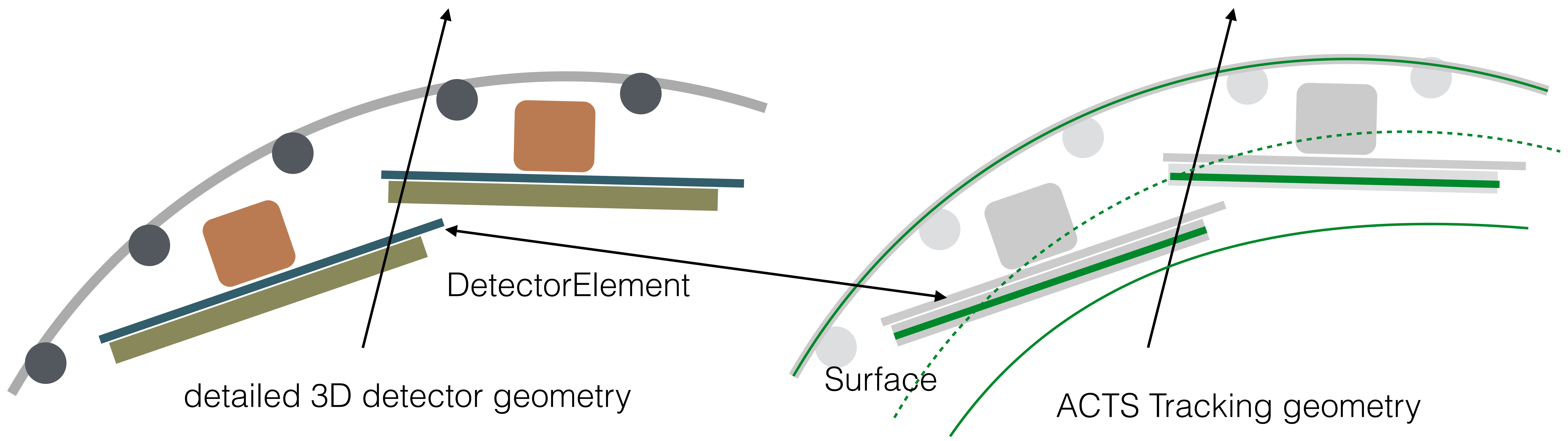
	Exec	Coverage	https://acts.web.cern.ch/ACTS/coverage/		
Date:	**2018-06-13**	**Lines:**	**2951**	**4708**	**62.7%**
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	**3091**	**22905**	**13.5%**

see also talk by H. Graslan

Geometry

```
namespace Acts {
  /// doxygen documentation
  class DetectorElementBase {
   /// the according represented surface
    virtual const Surface& associatedSurface() const = 0;
  };
}
```

```
class MyDetectorElement {
  /// @copydoc DetectorElementBase::asscociatedSurface
   const PlaneSurface& associatedSurface() const;
 };
```



Geometry binding with via a simple/single detector element class on detector software side

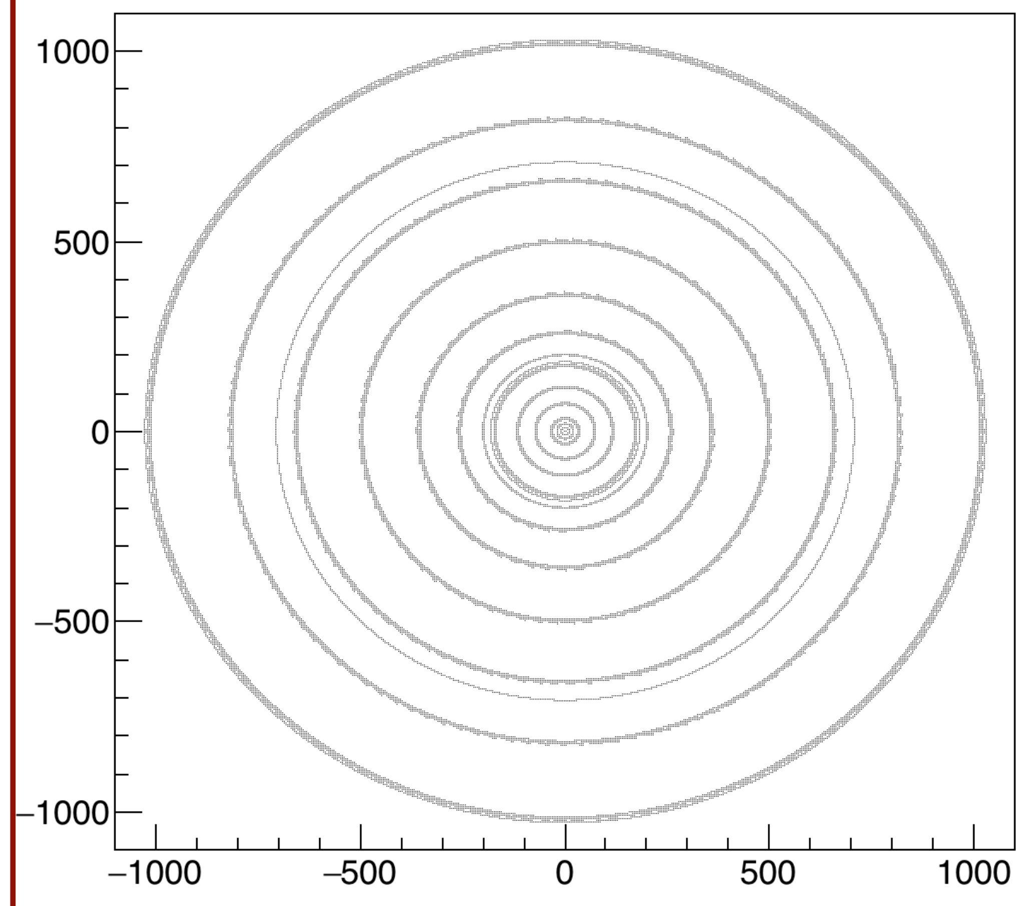
Problematic:

- units
- coordinate system conventions

Geometry

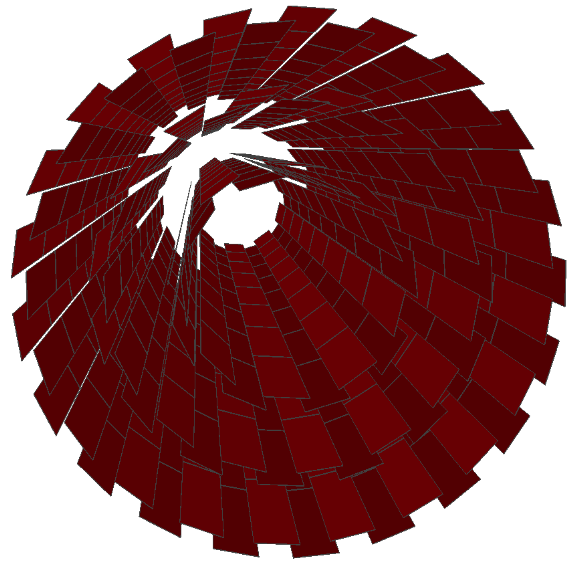
C++

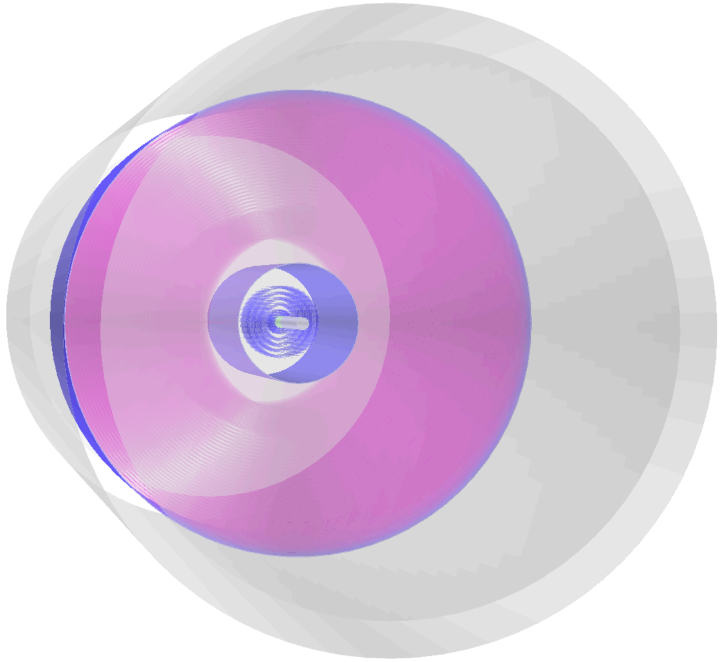
```
// BARREL :
// 4 pixel layers
// configure the central barrel
plbConfig.centralLayerBinMultipliers = {1, 1};
plbConfig.centralLayerRadii          = {32., 72., 116., 172.};
plbConfig.centralLayerEnvelopes
    = {pcEnvelope, pcEnvelope, pcEnvelope, pcEnvelope};
// material concentration alsways outside the modules
plbConfig.centralLayerMaterialConcentration = {1, 1, 1, 1};
plbConfig.centralLayerMaterialProperties
    = {pcmbProperties, pcmbProperties, pcmbProperties, pcmbProperties};
plbConfig.centralModuleBinningSchema = {{16, 14}, {32, 14}, {52, 14}, {78, 14}};
plbConfig.centralModuleTiltPhi       = {0.14, 0.14, 0.14, 0.14};
plbConfig.centralModuleHalfX         = {8.4, 8.4, 8.4, 8.4};
plbConfig.centralModuleHalfY         = {36., 36., 36., 36.};
plbConfig.centralModuleThickness     = {0.15, 0.15, 0.15, 0.15};
plbConfig.centralModuleMaterial
    = {pcMaterial, pcMaterial, pcMaterial, pcMaterial};
// pitch definitions
plbConfig.centralModuleReadoutBinsX = {336, 336, 336, 336};
plbConfig.centralModuleReadoutBinsY = {1280, 1280, 1280, 1280};
plbConfig.centralModuleReadoutSide  = {-1, -1, -1, -1};
plbConfig.centralModuleLorentzAngle = {0.12, 0.12, 0.12, 0.12};

// no frontside/backside
plbConfig.centralModuleFrontsideStereo = {};
plbConfig.centralModuleBacksideStereo  = {};
plbConfig.centralModuleBacksideGap     = {};
```

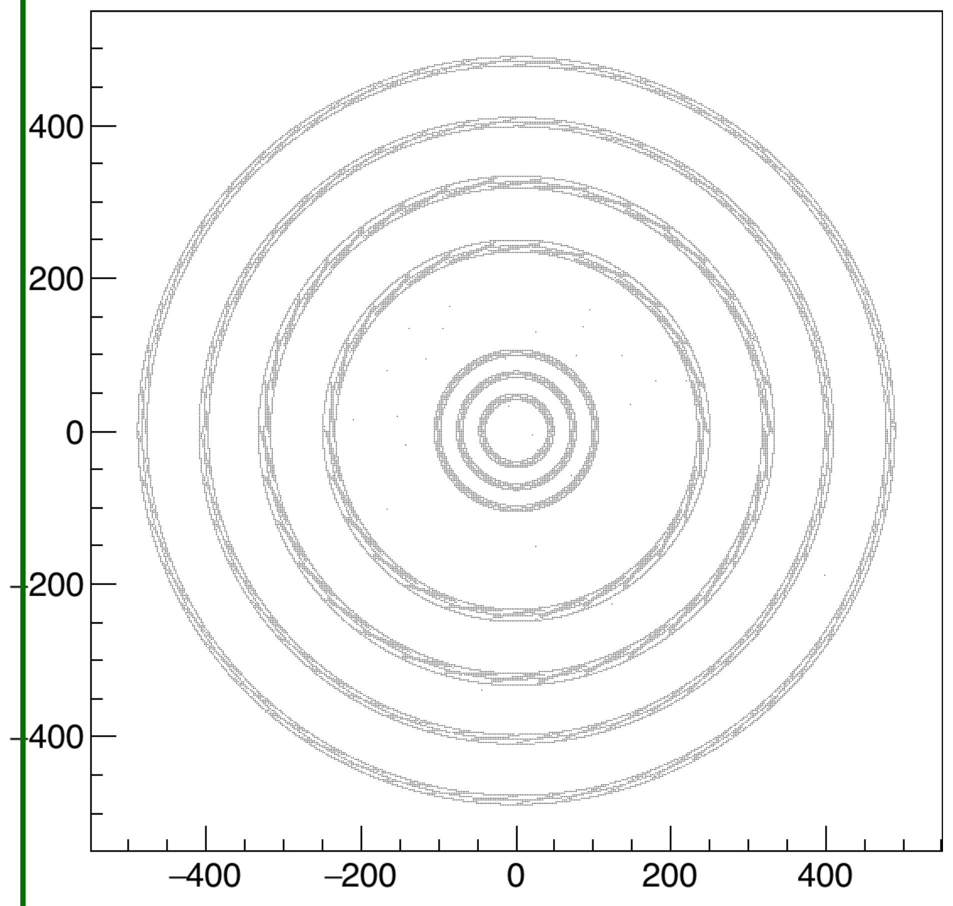


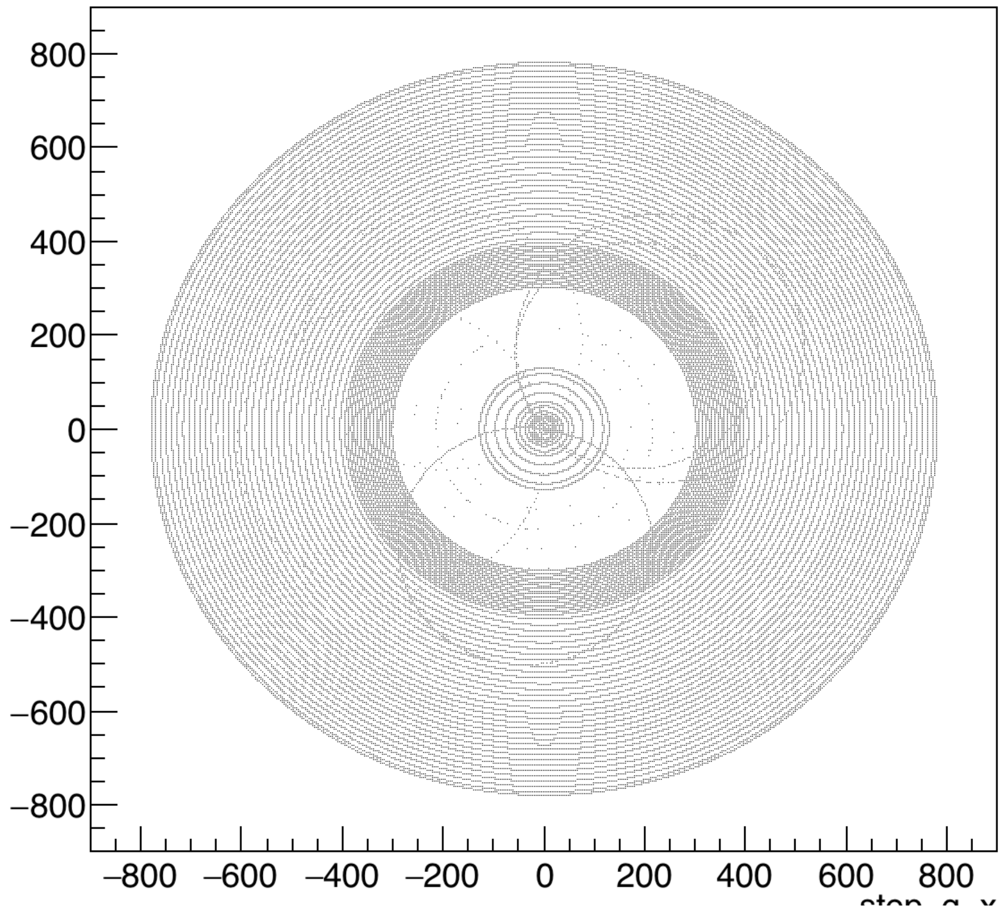
TrackML Detector

TGeo (Root)





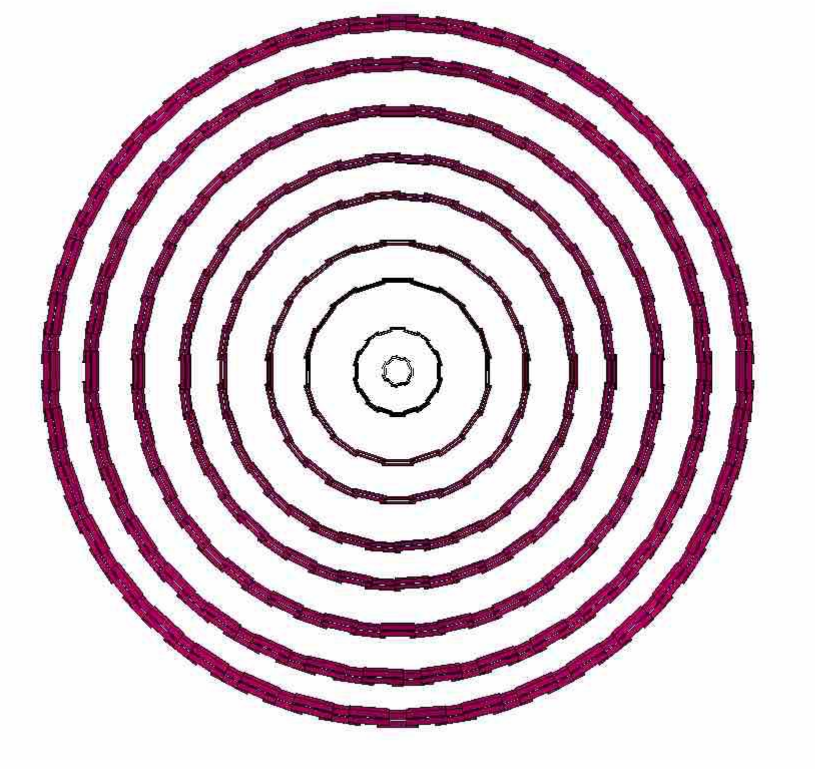


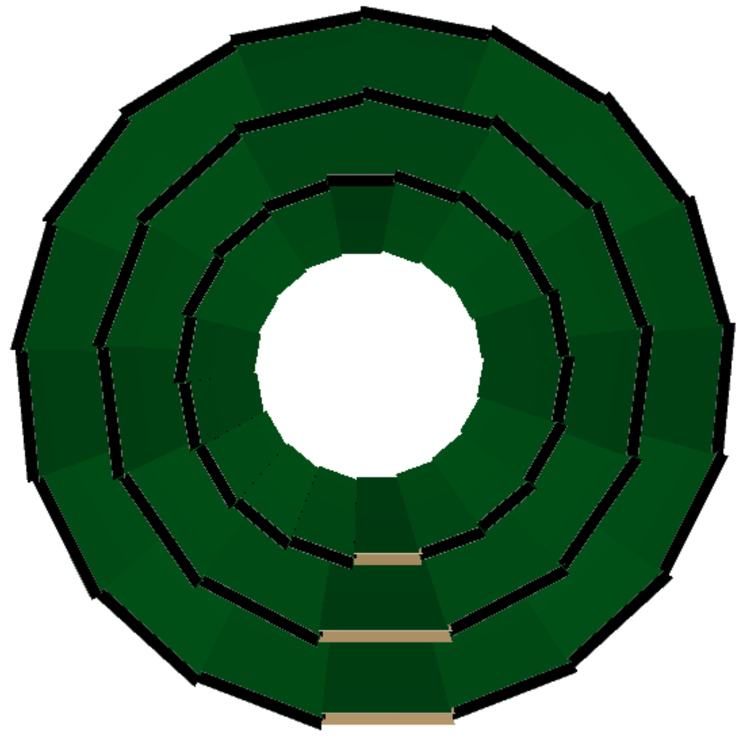


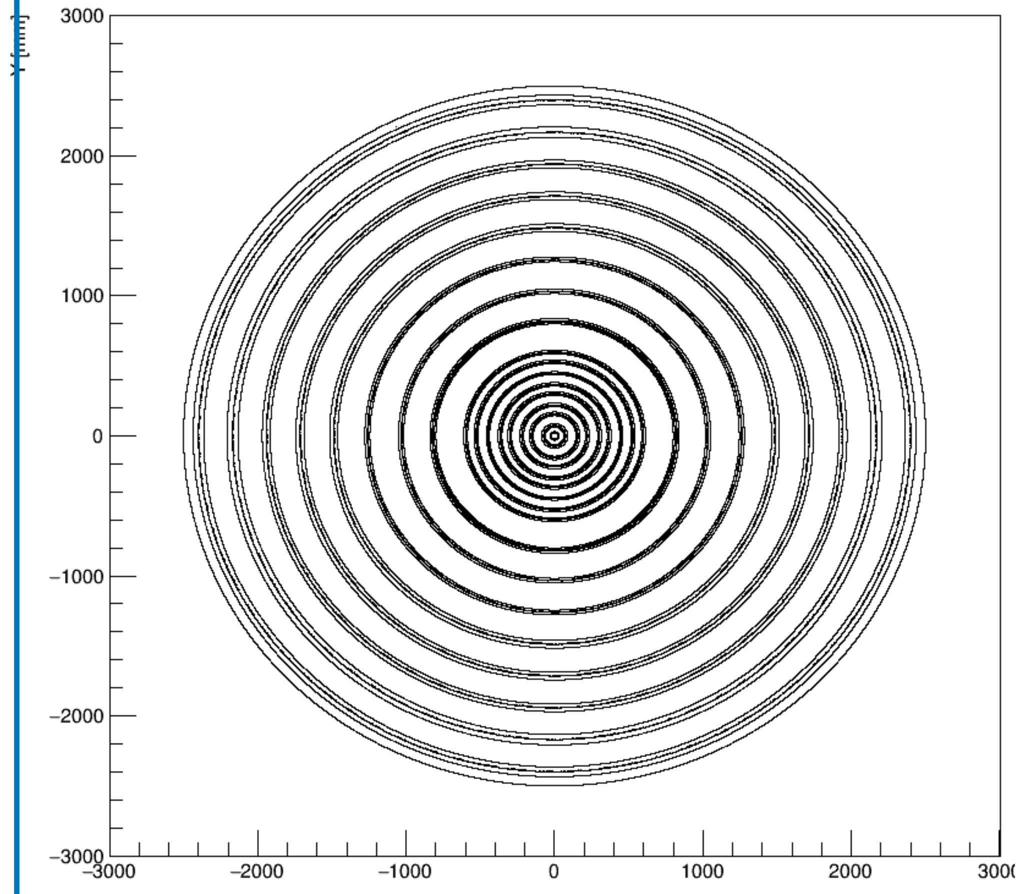
CMS Pixel & SVX

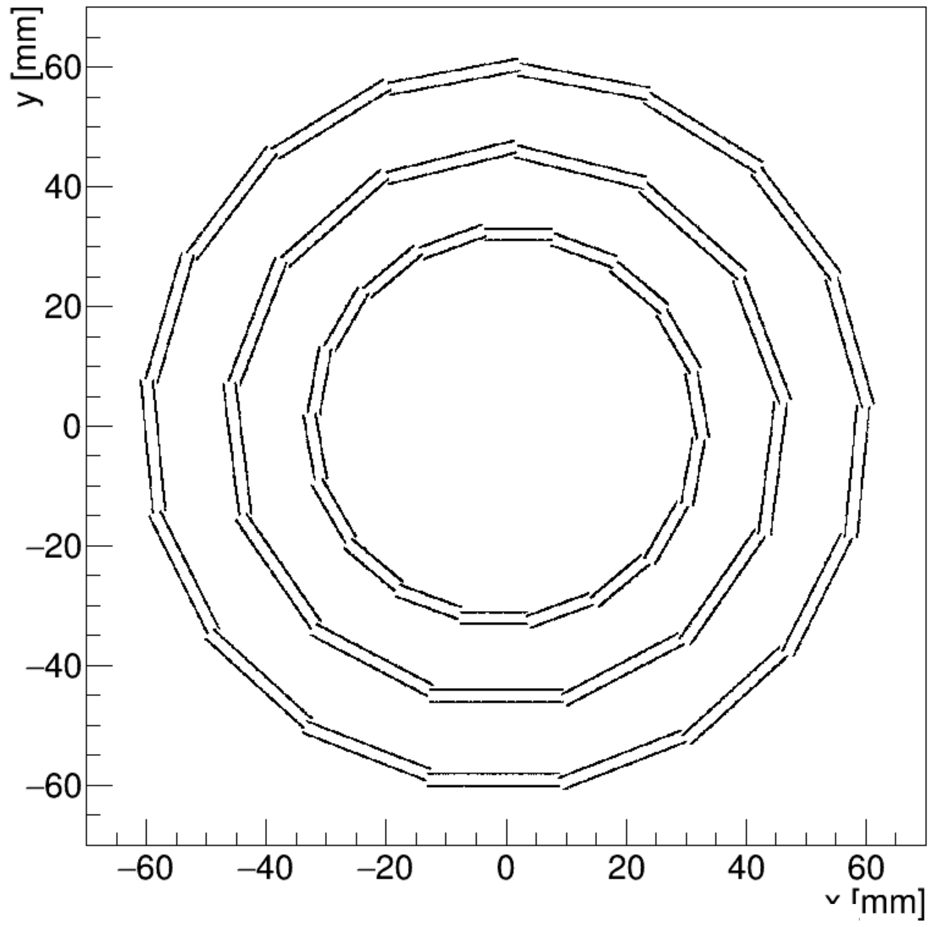
sPhenix

DD4Hep









FCC-hh

CLIC Vertex

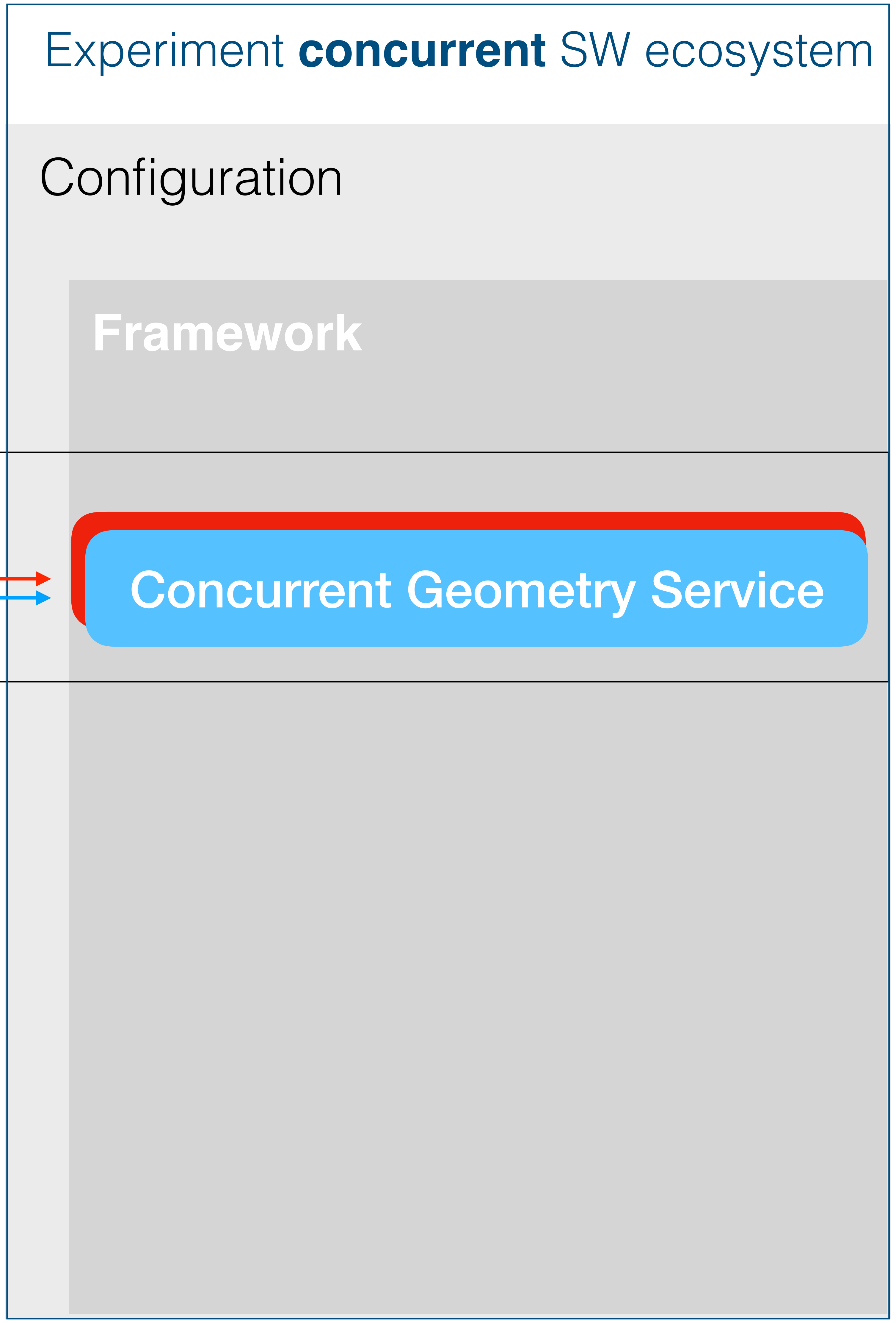
Geometry&Alignment

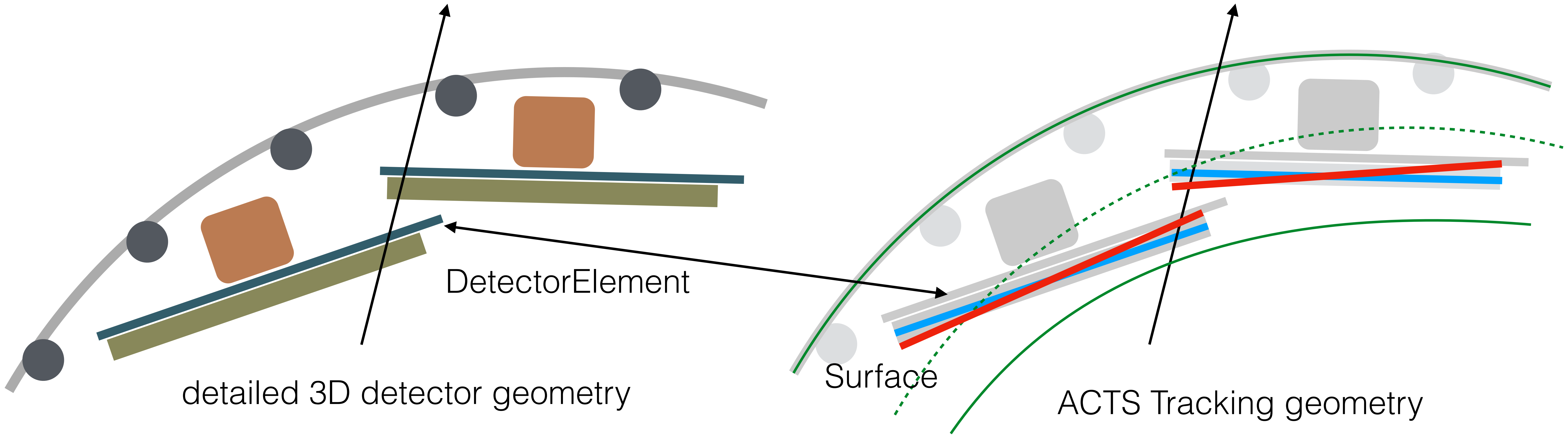
```
namespace Acts {
  /// doxygen documentation
  class DetectorElementBase {
   /// the according represented surface
    virtual const Surface& associatedSurface() const = 0;
  };
}
```

```
class MyDetectorElement {
  /// @copydoc DetectorElementBase::asscociatedSurface
   const PlaneSurface& associatedSurface() const;
 };
```

validity intervall 0

validity intervall 1





Event Data

Flat event data model design based on **Eigen**

- leave calibration/details with detector SW
- use fixed size data structures

Problematic:

- different local coordinate systems
- heterogenous containers for measurements

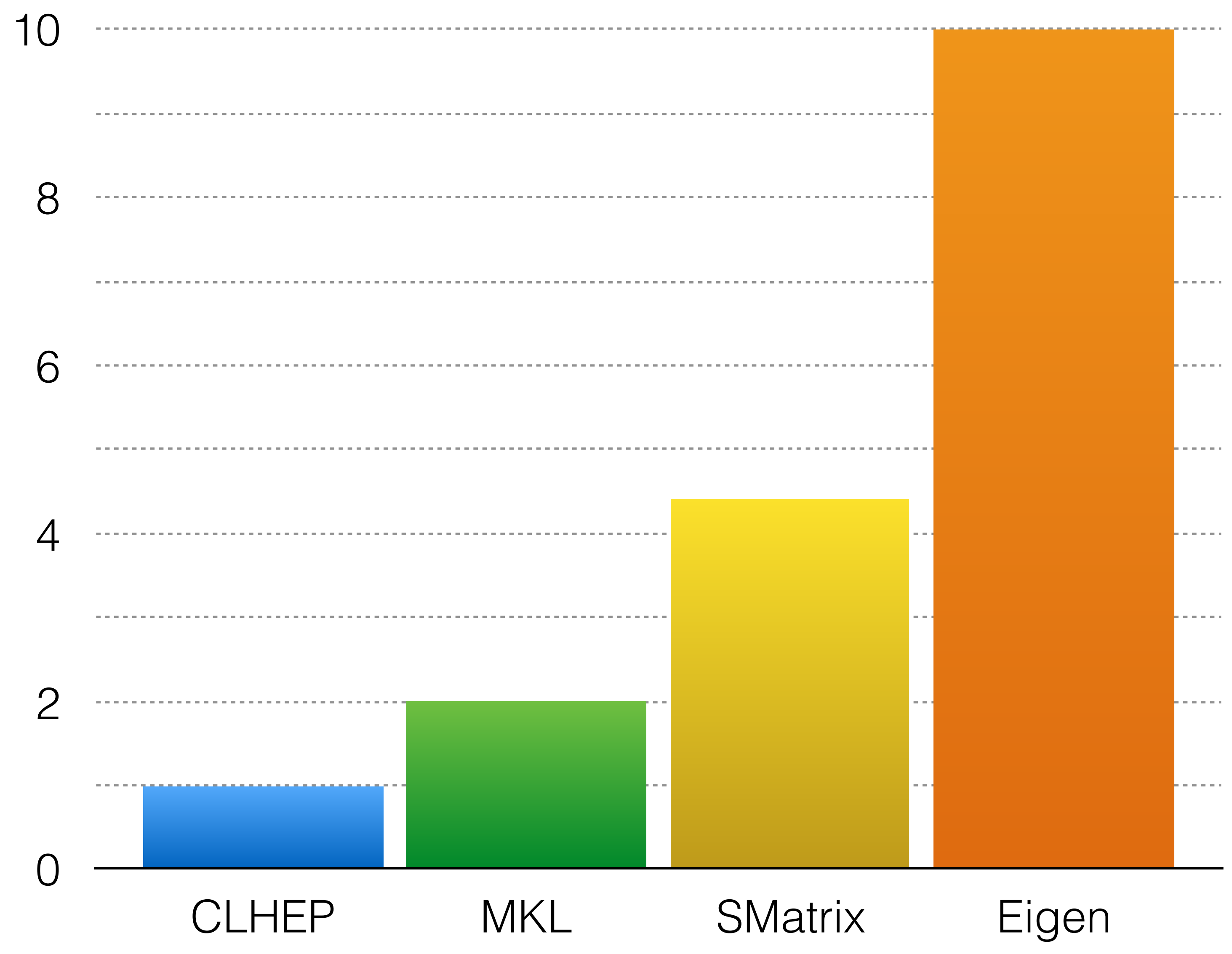
CDF $\mathbf{q}'' = (l_1, l_2, \phi, \cot(\theta), C)$

CMS $\mathbf{q}' = (l_1, l_2, \phi, \lambda, q/p)$

ATLAS $\mathbf{q} = (l_1, l_2, \phi, \theta, q/p)$

```
#pragma once
#ifdef ACTS_PARAMETER_DEFINITIONS_PLUGIN
#include ACTS_PARAMETER_DEFINITIONS_PLUGIN
#endif

/// testing requirements on parameter definitions
#include <type_traits>

/// typedefs for parameter identifier and parameter value must be present
static_assert(std::is_enum<Acts::ParID_t>::value,
              "'ParID_t' is not an enum type");
static_assert(std::is_floating_point<Acts::ParValue_t>::value,
              "'ParValue_t' is not floating point type");
```

#include PLUGIN in Eigen look and feel
(turns out to be rather difficult)



Achieved speed-up w.r.t. CLHEP in 5x5 matrix multiplication testbed

CHEP2015 talk by AS

http://eigen.tuxfamily.org/

Event Data

Flat event data model design based on **Eigen**
- leave calibration/details with detector SW
- use fixed size data structures

Problematic:
- different local coordinate systems
- heterogenous containers for measurements

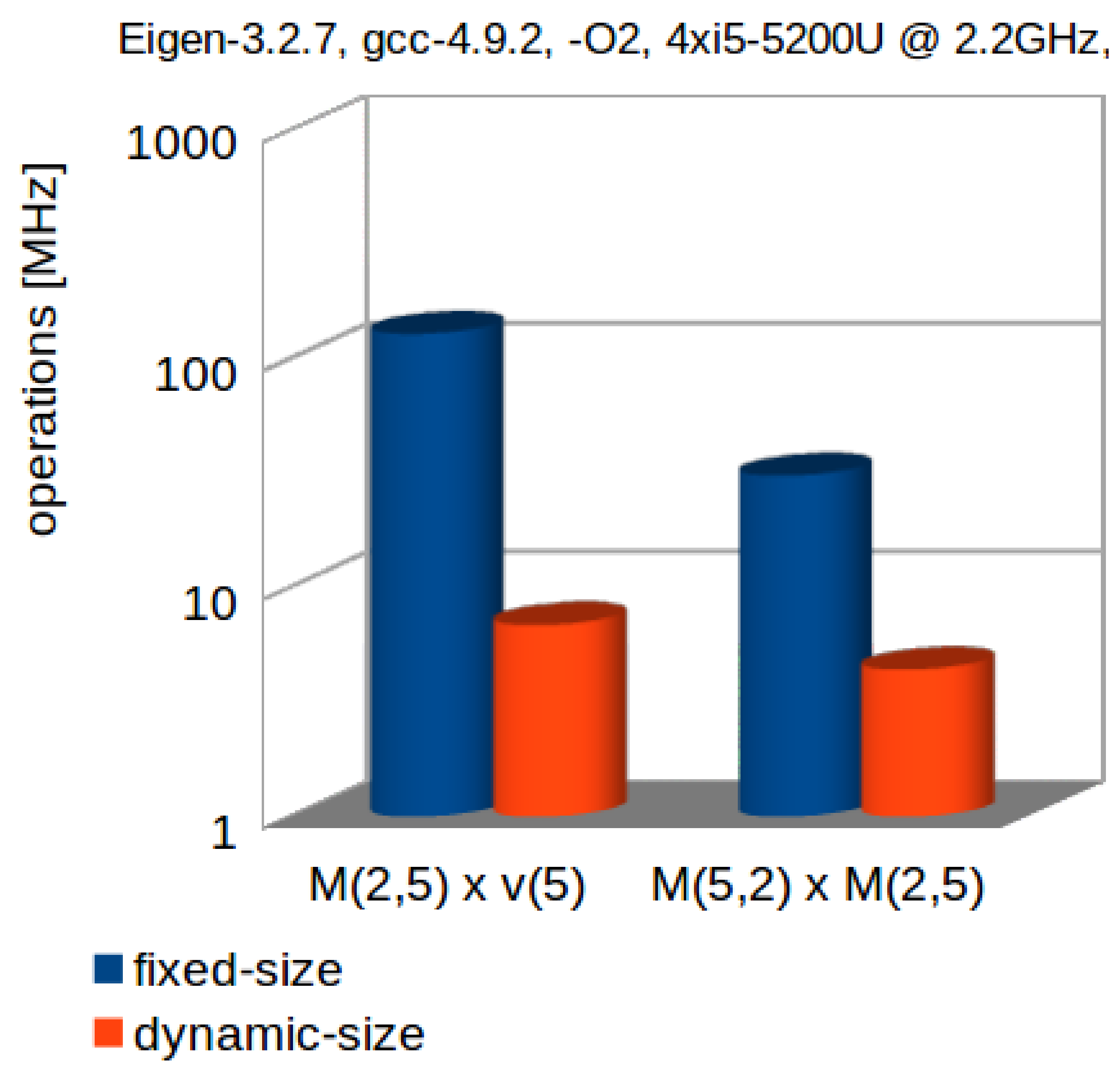
CDF $\mathbf{q}'' = (l_1, l_2, \phi, \cot(\theta), C)$

CMS $\mathbf{q}' = (l_1, l_2, \phi, \lambda, q/p)$

ATLAS $\mathbf{q} = (l_1, l_2, \phi, \theta, q/p)$

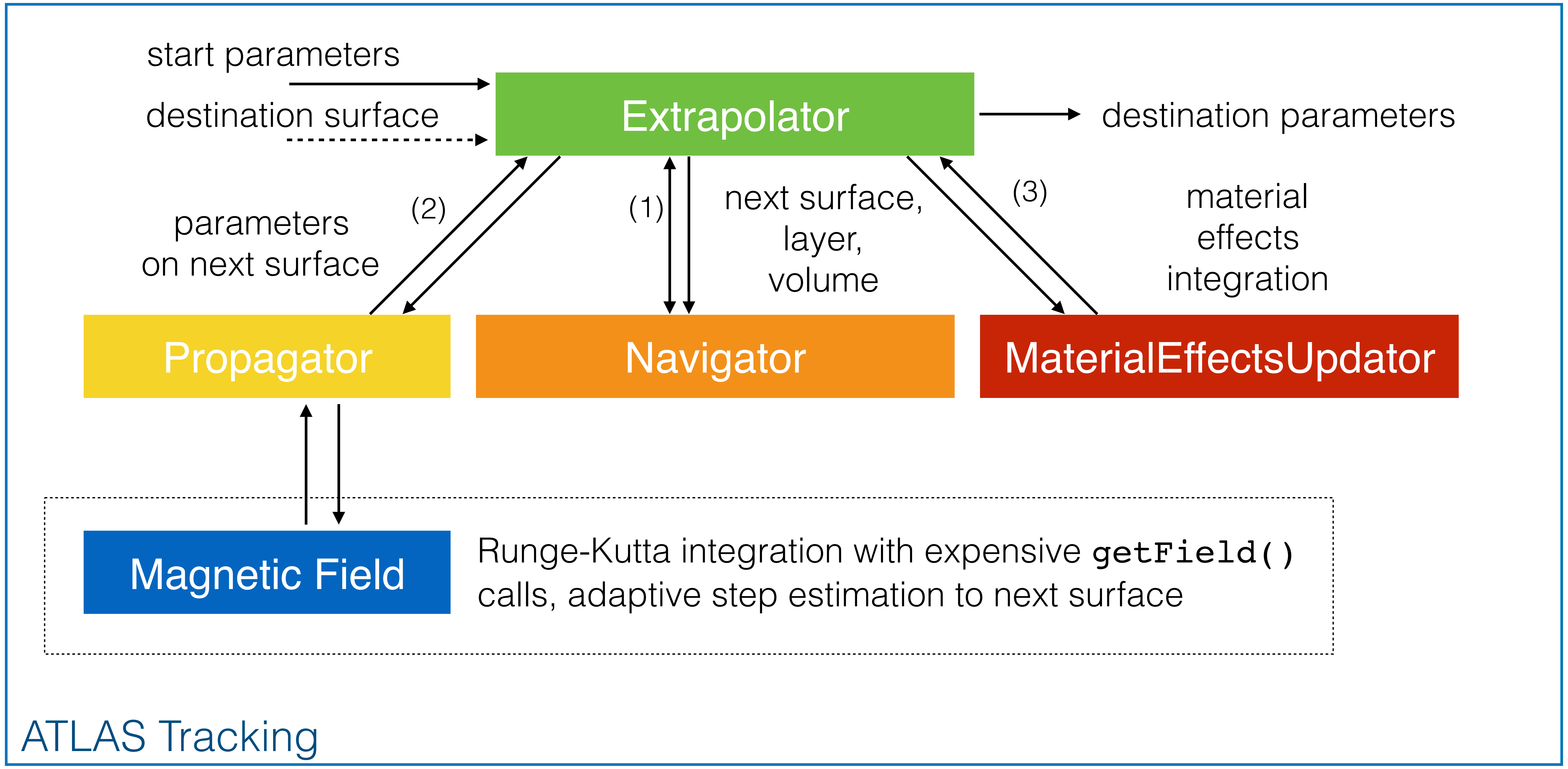
```
#pragma once
#ifdef ACTS_PARAMETER_DEFINITIONS_PLUGIN
#include ACTS_PARAMETER_DEFINITIONS_PLUGIN
#endif

/// testing requirements on parameter definitions
#include <type_traits>

/// typedefs for parameter identifier and parameter value must be present
static_assert(std::is_enum<Acts::ParID_t>::value,
              "'ParID_t' is not an enum type");
static_assert(std::is_floating_point<Acts::ParValue_t>::value,
              "'ParValue_t' is not floating point type");
```

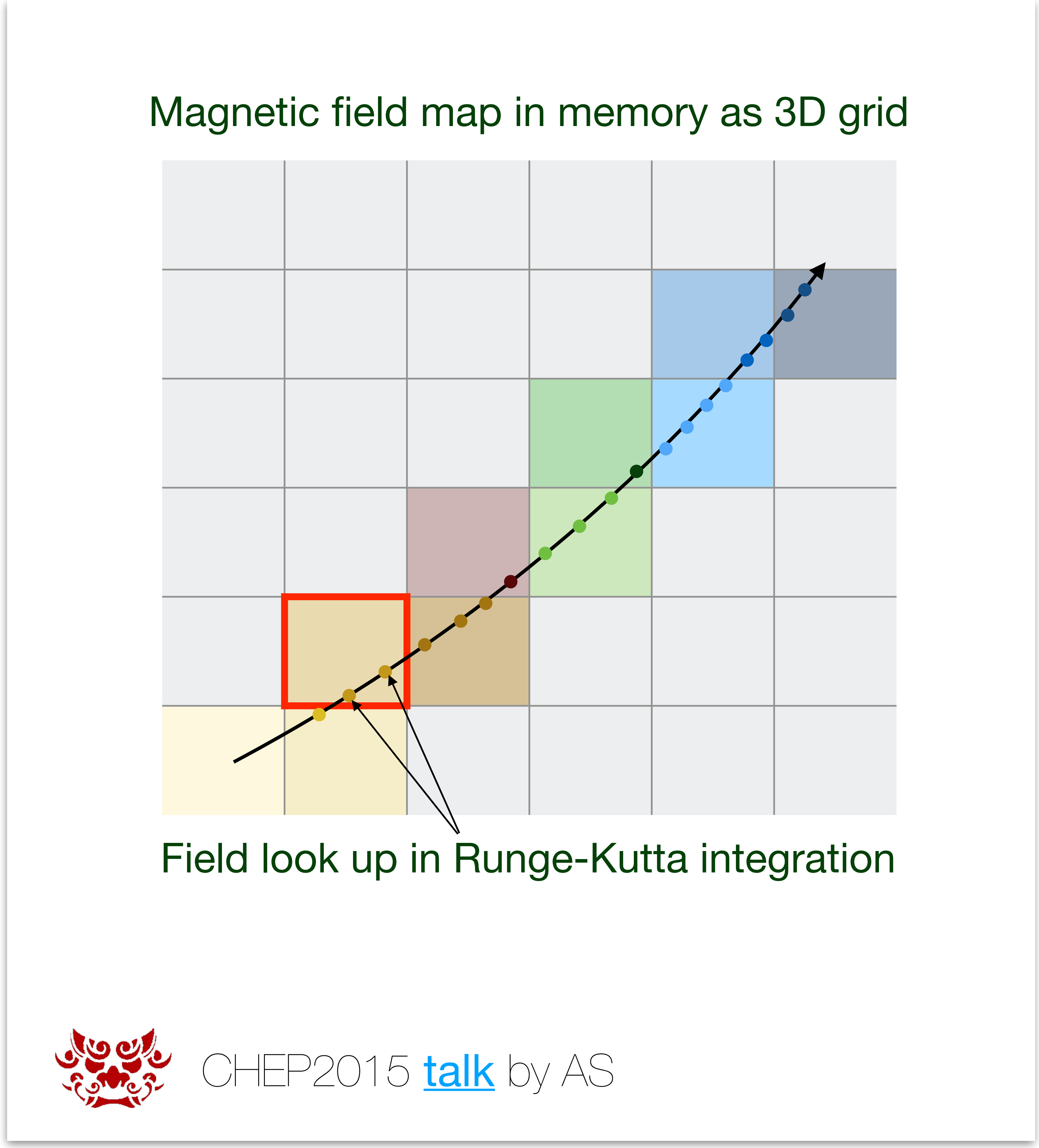
#include PLUGIN in Eigen look and feel
(turns out to be rather difficult)

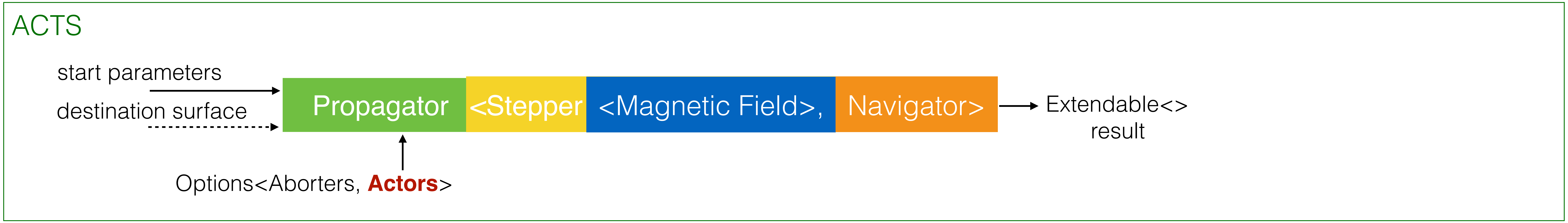


http://eigen.tuxfamily.org/

Propagation&MagneticField







Propagation&Extrapolation

start parameters → Propagator <Stepper <Magnetic Field>, Navigator> → Extendable<> result

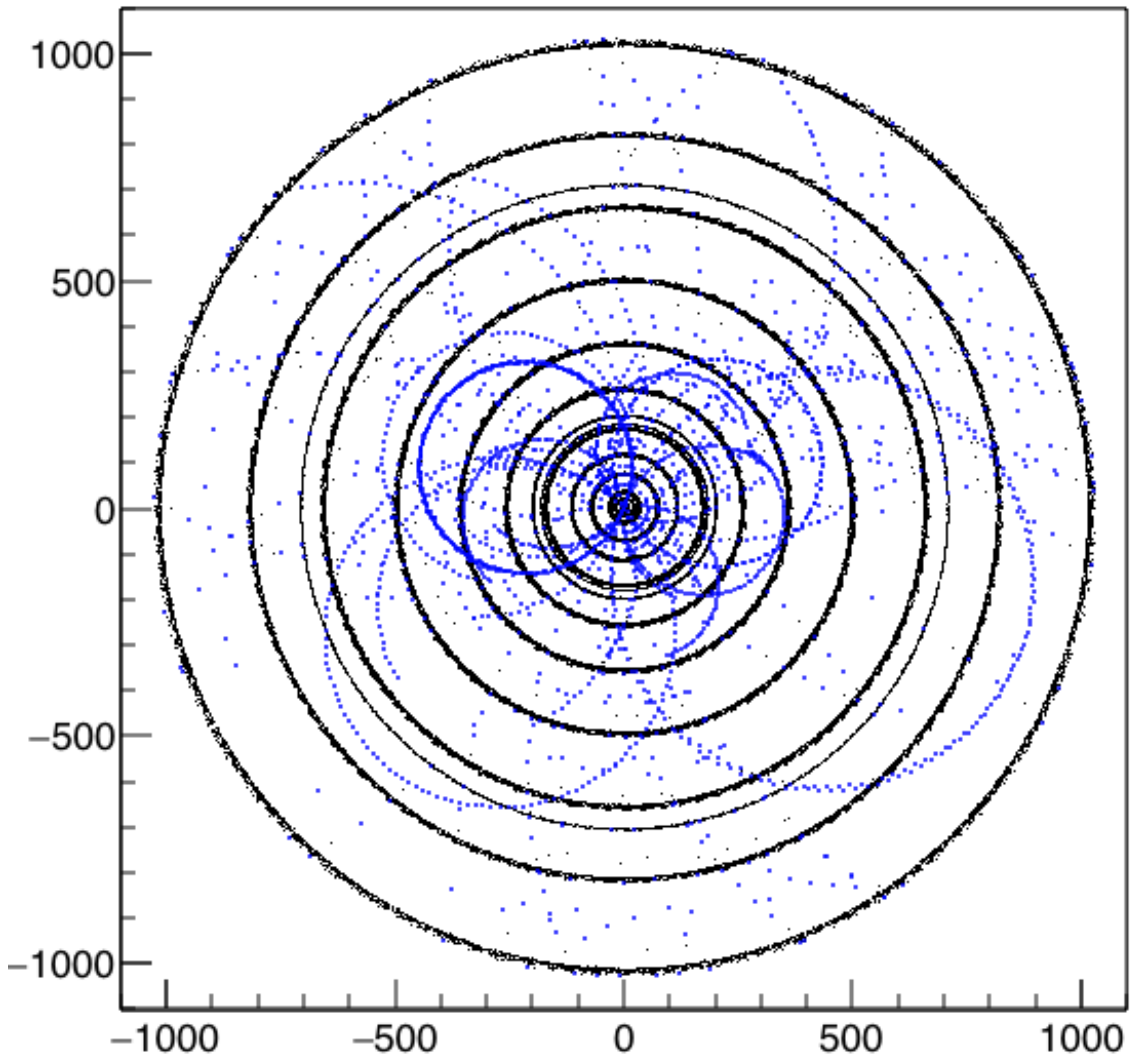
Options<Aborters, Actors>

AtlasStepper

```
if (Jac) {
  // Jacobian calculation
  //
  double* d2A  = &state.pVector[24];
  double* d3A  = &state.pVector[31];
  double* d4A  = &state.pVector[38];
  double  d2A0 = H0[2] * d2A[1] - H0[1] * d2A[2];
  double  d2B0 = H0[0] * d2A[2] - H0[2] * d2A[0];
  double  d2C0 = H0[1] * d2A[0] - H0[0] * d2A[1];
  double  d3A0 = H0[2] * d3A[1] - H0[1] * d3A[2];
  double  d3B0 = H0[0] * d3A[2] - H0[2] * d3A[0];
  double  d3C0 = H0[1] * d3A[0] - H0[0] * d3A[1];
  double  d4A0 = (A0 + H0[2] * d4A[1]) - H0[1] * d4A[2];
  double  d4B0 = (B0 + H0[0] * d4A[2]) - H0[2] * d4A[0];
  double  d4C0 = (C0 + H0[1] * d4A[0]) - H0[0] * d4A[1];
  double  d2A2 = d2A0 + d2A[0];
  double  d2B2 = d2B0 + d2A[1];
  double  d2C2 = d2C0 + d2A[2];
  double  d3A2 = d3A0 + d3A[0];
  double  d3B2 = d3B0 + d3A[1];
  double  d3C2 = d3C0 + d3A[2];
  double  d4A2 = d4A0 + d4A[0];
  double  d4B2 = d4B0 + d4A[1];
  double  d4C2 = d4C0 + d4A[2];
  double  d0   = d4A[0] - A00;
  double  d1   = d4A[1] - A11;
  double  d2   = d4A[2] - A22;
  double  d2A3 = (d2A[0] + d2B2 * H1[2]) - d2C2 * H1[1];
  double  d2B3 = (d2A[1] + d2C2 * H1[0]) - d2A2 * H1[2];
  double  d2C3 = (d2A[2] + d2A2 * H1[1]) - d2B2 * H1[0];
  double  d3A3 = (d3A[0] + d3B2 * H1[2]) - d3C2 * H1[1];
  double  d3B3 = (d3A[1] + d3C2 * H1[0]) - d3A2 * H1[2];
  double  d3C3 = (d3A[2] + d3A2 * H1[1]) - d3B2 * H1[0];
  double  d4A3 = ((A3 + d0) + d4B2 * H1[2]) - d4C2 * H1[1];
  double  d4B3 = ((B3 + d1) + d4C2 * H1[0]) - d4A2 * H1[2];
  double  d4C3 = ((C3 + d2) + d4A2 * H1[1]) - d4B2 * H1[0];
  double  d2A4 = (d2A[0] + d2B3 * H1[2]) - d2C3 * H1[1];
  double  d2B4 = (d2A[1] + d2C3 * H1[0]) - d2A3 * H1[2];
  double  d2C4 = (d2A[2] + d2A3 * H1[1]) - d2B3 * H1[0];
  double  d3A4 = (d3A[0] + d3B3 * H1[2]) - d3C3 * H1[1];
  double  d3B4 = (d3A[1] + d3C3 * H1[0]) - d3A3 * H1[2];
  double  d3C4 = (d3A[2] + d3A3 * H1[1]) - d3B3 * H1[0];
```

EigenStepper

```
/// Method for on-demand transport of the covariance
/// to a new curvilinear frame at current  position,
/// or direction of the state
///
/// @param reinitialize is a flag to steer whether the
///        state should be reinitialized at the new
///        position
///
/// @return the full transport jacobian
const ActsMatrixD<5, 5>
applyCovTransport(bool reinitialize = false)
{
  // Optimized trigonometry on the propagation direction
  const double x = dir(0);  // == cos(phi) * sin(theta)
  const double y = dir(1);  // == sin(phi) * sin(theta)
  const double z = dir(2);  // == cos(theta)
  // can be turned into cosine/sine
  const double cosTheta    = z;
  const double sinTheta    = sqrt(x * x + y * y);
  const double invSinTheta = 1. / sinTheta;
  const double cosPhi      = x * invSinTheta;
  const double sinPhi      = y * invSinTheta;
  // prepare the jacobian to curvilinear
  ActsMatrixD<5, 7> jacToCurv = ActsMatrixD<5, 7>::Zero();
  if (std::abs(cosTheta) < s_curvilinearProjTolerance) {
    // We normally operate in curvilinear coordinates defined as follows
    jacToCurv(0, 0) = -sinPhi;
    jacToCurv(0, 1) = cosPhi;
    jacToCurv(1, 0) = -cosPhi * cosTheta;
    jacToCurv(1, 1) = -sinPhi * cosTheta;
    jacToCurv(1, 2) = sinTheta;
  } else {
    // Under grazing incidence to z, the above coordinate system definition
    // becomes numerically unstable, and we need to switch to another one
    const double c    = sqrt(y * y + z * z);
    const double invC = 1. / c;
    jacToCurv(0, 1) = -z * invC;
    jacToCurv(0, 2) = y * invC;
    jacToCurv(1, 0) = c;
    jacToCurv(1, 1) = -x * y * invC;
    jacToCurv(1, 2) = -x * z * invC;
  }
```



Output of StepLengthLogger ("Actor")

Propagation&Extrapolation

start parameters → Propagator <Stepper <Magnetic Field>, Navigator> → Extendable<> result

Options<Aborters, Actors>

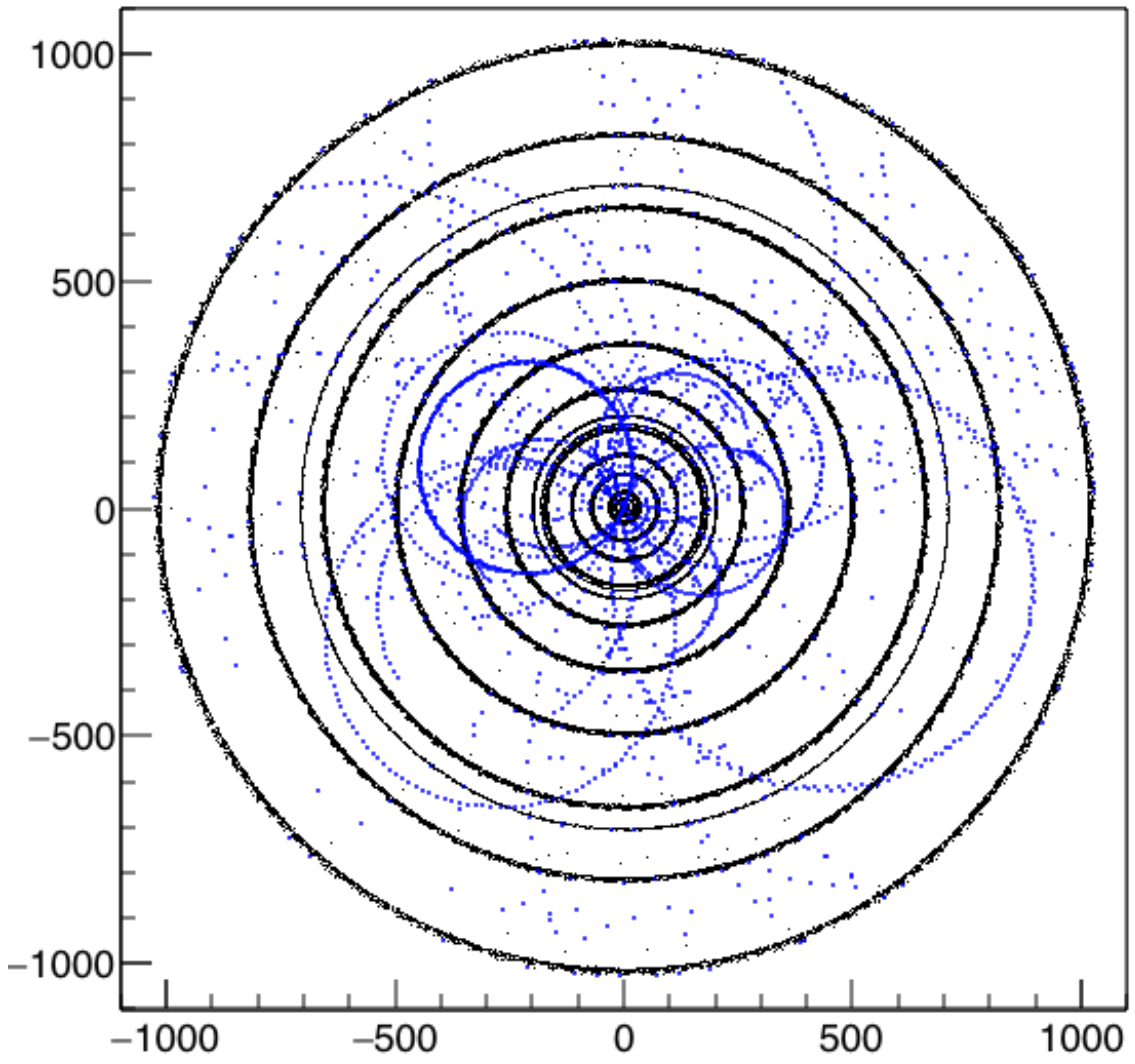
Vectorized versions

ROBERTO AGOSTINO VITILLO (2012)

STEP 3

PROFIT!

- Tested on a Sandy Bridge-EP CPU
- SSE version (assembly & intrinsics): 2.4x faster
- AVX version: 1.5x faster
  - slower than SSE because of costly cross lane permutations



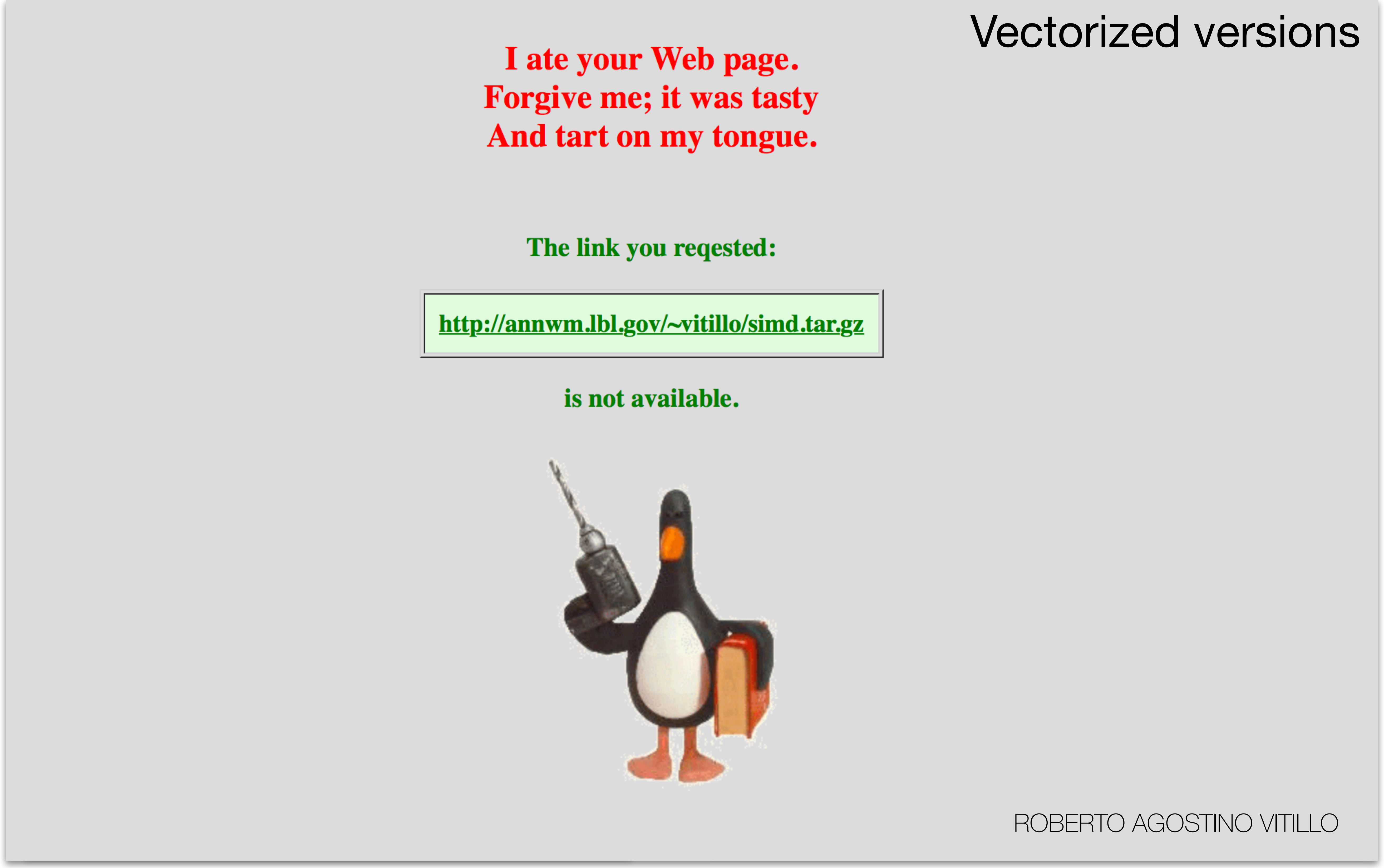
Output of StepLengthLogger ("Actor")

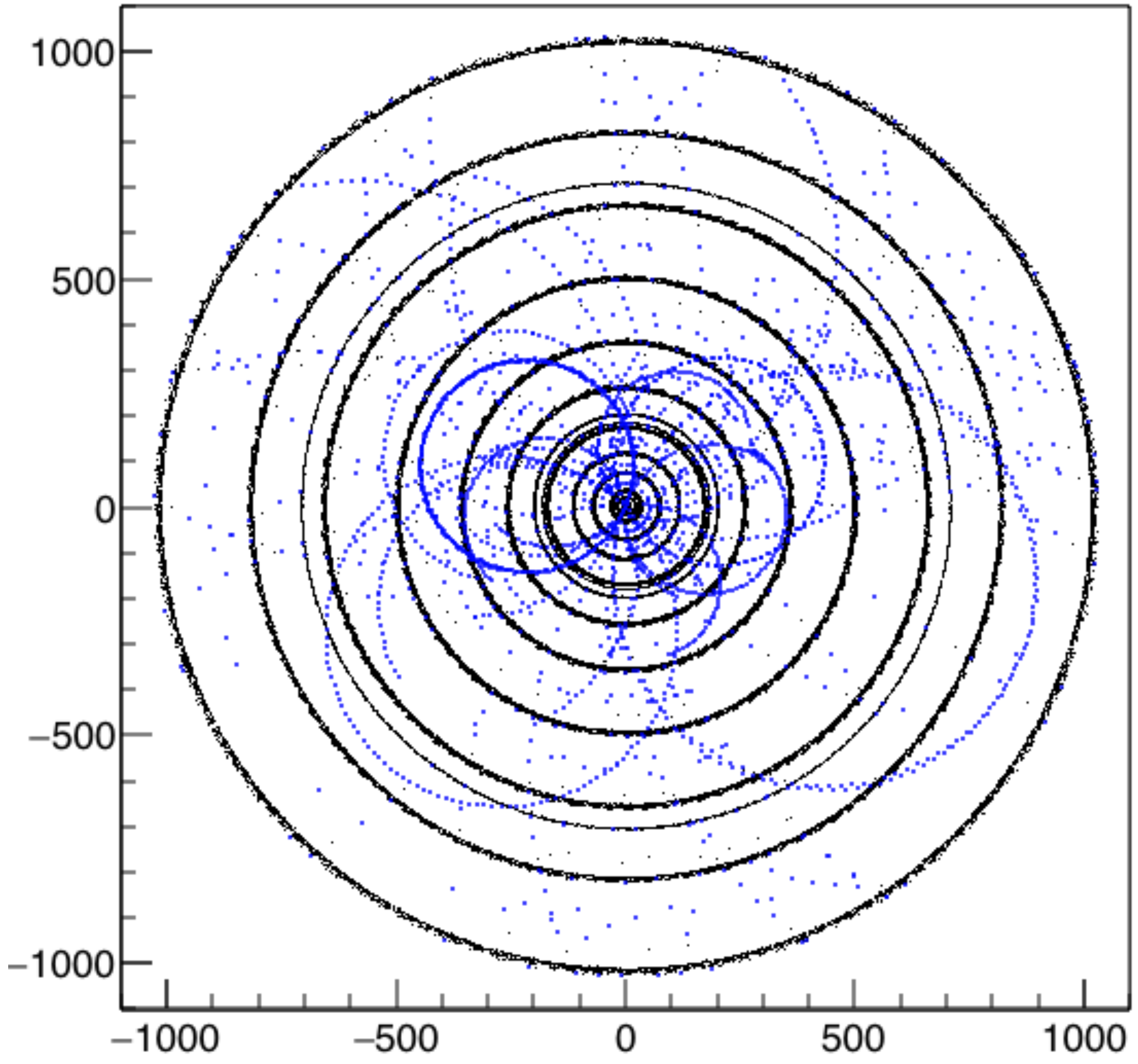
Propagation&Extrapolation

start parameters → Propagator <Stepper <Magnetic Field>, Navigator> → Extendable<> result

Options<Aborters, Actors>

Vectorized versions





Output of StepLengthLogger ("Actor")

Propagation&Extrapolation

start parameters → Propagator <Stepper <Magnetic Field>, Navigator> → Extendable<> result

Options<Aborters, Actors>

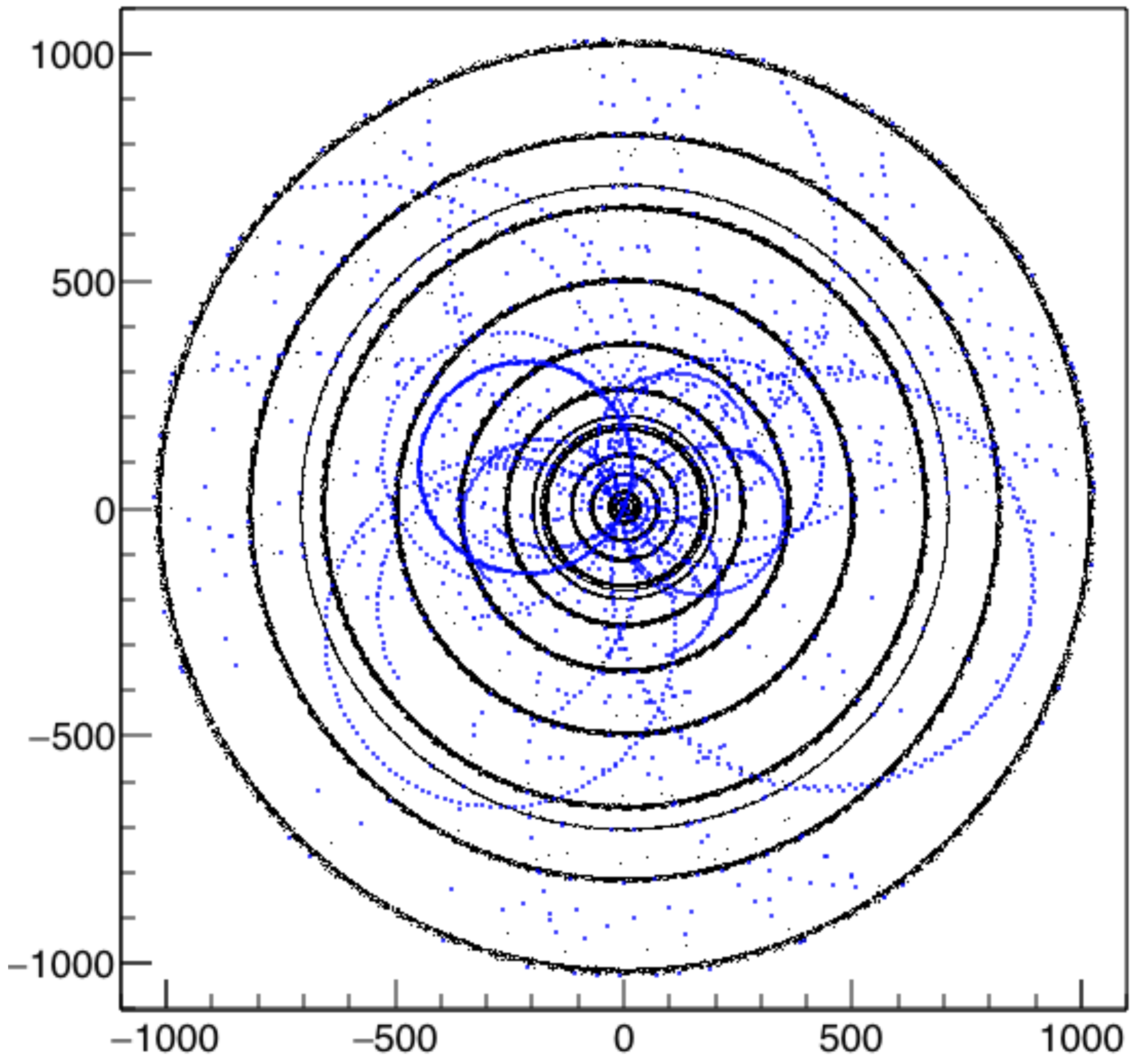
Vectorized versions

```
for(int j = 0; j < N; j++){
  for(int i = 0; i < 42; i+=7){
    __m256d dR = _mm256_loadu_pd(&P[i]);

    __m256d dA = _mm256_loadu_pd(&P[i + 3]);
    __m256d dA_201 = CROSS_SHUFFLE_201(dA);
    __m256d dA_120 = CROSS_SHUFFLE_120(dA);

    __m256d d0 = _mm256_sub_pd(_mm256_mul_pd(H0_201,

    if(i==35){
      d0 = _mm256_add_pd(d0, V0_012);
    }

    __m256d d2 = _mm256_add_pd(d0, dA);
    __m256d d2_201 = CROSS_SHUFFLE_201(d2);
    __m256d d2_120 = CROSS_SHUFFLE_120(d2);

    __m256d d3 = _mm256_sub_pd(_mm256_add_pd(dA, _mm2
    __m256d d3_201 = CROSS_SHUFFLE_201(d3);
    __m256d d3_120 = CROSS_SHUFFLE_120(d3);

    if(i==35){
      d3 = _mm256_add_pd(d3, _mm256_sub_pd(V3_012, A_
    }

    __m256d d4 = _mm256_sub_pd(_mm256_add_pd(dA, _mm2

    if(i==35){
      d4 = _mm256_add_pd(d4, _mm256_sub_pd(V4_012, A_
    }
```

SSE

```
for(int j = 0; j < N; j++){
  for(int i = 0; i < 42; i+=7){
    __m128 dR = _mm_loadu_ps(&P[i]);

    __m128 dA = _mm_loadu_ps(&P[i + 3]);
    __m128 dA_201 = _mm_shuffle_ps(dA, dA, 0xD2);
    __m128 dA_120 = _mm_shuffle_ps(dA, dA, 0xC9);

    __m128 d0 = _mm_sub_ps(_mm_mul_ps(H0_201, dA_12

    if(i==35){
      d0 = _mm_add_ps(d0, V0_012);
    }

    __m128 d2 = _mm_add_ps(d0, dA);
    __m128 d2_201 = _mm_shuffle_ps(d2, d2, 0xD2);
    __m128 d2_120 = _mm_shuffle_ps(d2, d2, 0xC9);

    __m128 d3 = _mm_sub_ps(_mm_add_ps(dA, _mm_mul_ps
    __m128 d3_201 = _mm_shuffle_ps(d3, d3, 0xD2);
    __m128 d3_120 = _mm_shuffle_ps(d3, d3, 0xC9);


    if(i==35){
      d3 = _mm_add_ps(d3, _mm_sub_ps(V3_012, A_012))
    }

    __m128 d4 = _mm_sub_ps(_mm_add_ps(dA, _mm_mul_ps

    if(i==35){
      d4 = _mm_add_ps(d4, _mm_sub_ps(V4_012, A_012))
    }
```

AVX



Output of StepLengthLogger (“Actor”)

Concurrency

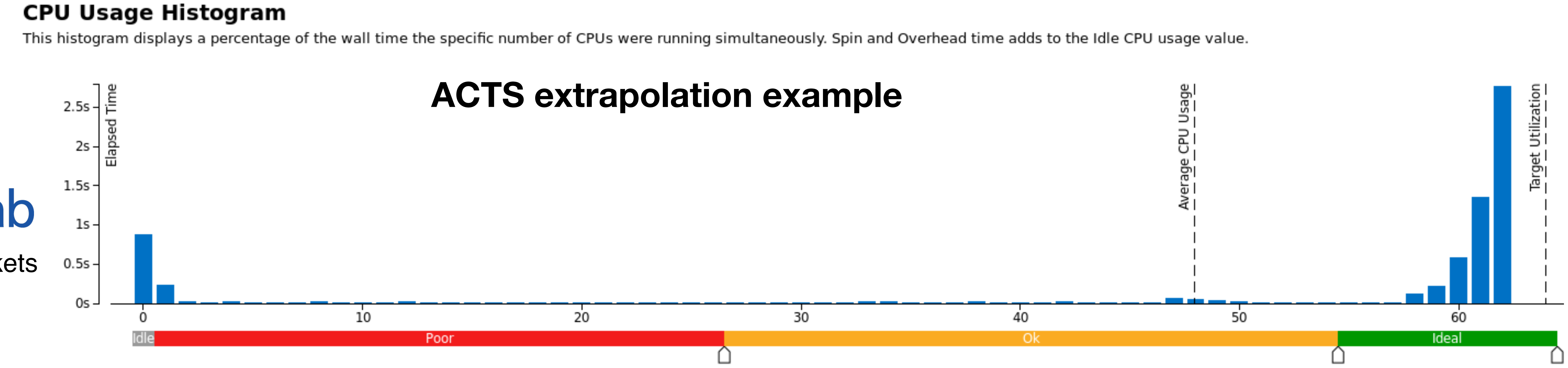
```
namespace Acts {
  /// doxygen documentation
  class WorkHorse {
   /// @struct Config for To
    struct State {
       float hoursWorked; ///< state parameter
    };

   /// @brief work method
   Result doWork(State& hState, const Input& workInput) const;


  };
} // end of namespace Acts
```

Caller must provide a by construction thread local state.

All modules are tested for consistency between single-threaded or multi-threaded mode.

CERN openlab

Intel Xeon e5-2698 v3, 2 sockets
32 Cores, 2 threads per core
64 Processors



Configuration

Configuration struct (nested)

```
namespace Acts {
  /// doxygen documentation
  class WorkHorse {
   /// @struct Config for To
    struct Config {
       float coatColor; ///< configure the coat color
       float maxHours;  ///< set the max hours this horse can work
    };
  };
} // end of namespace Acts
```

Binding to detector SW ecosystem

```
/// feed from Framework into ACTS configuration
declareProperty("CoatColor", m_cfg.coatColor);
declareProperty("MaxHours",  m_cfg.maxHours);
```

Problematic:

- chain of configured components is difficult to translate

Logging

Screen output logging is an important debugging tool

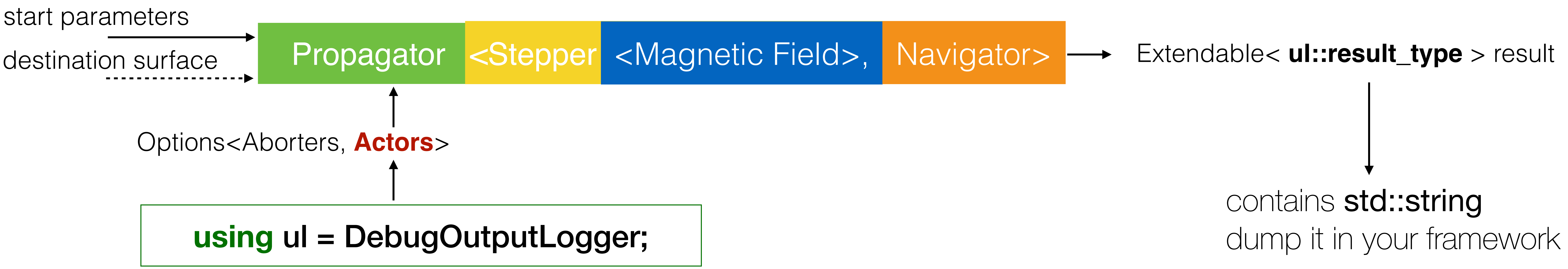
```
/// @brief macro for verbose debug output
/// @ingroup Logging
///
/// @param x debug message
///
/// @pre @c logger() must be a valid expression in the scope where this
///      macro is used and it must return a Acts::Logger object.
///
/// The debug message is printed if the current Acts::Logging::Level <=
/// Acts::Logging::VERBOSE.
#define ACTS_VERBOSE(x)
  if (logger().doPrint(Acts::Logging::VERBOSE))
    logger().log(Acts::Logging::VERBOSE) << x;
```

Problematic:

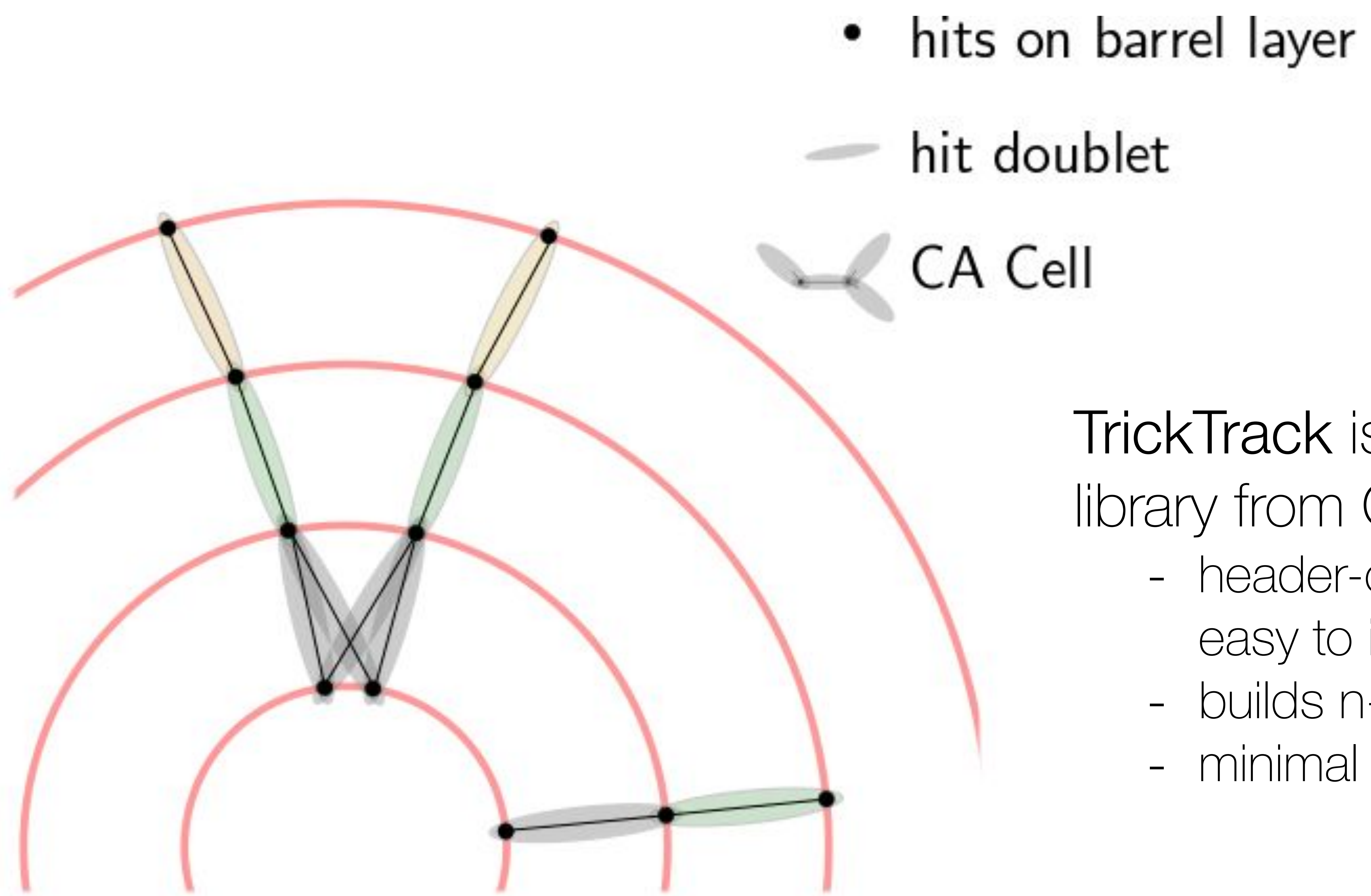
- binding different logging technologies is nasty
- currently SW needs to provide ACT_MSG(X) macros

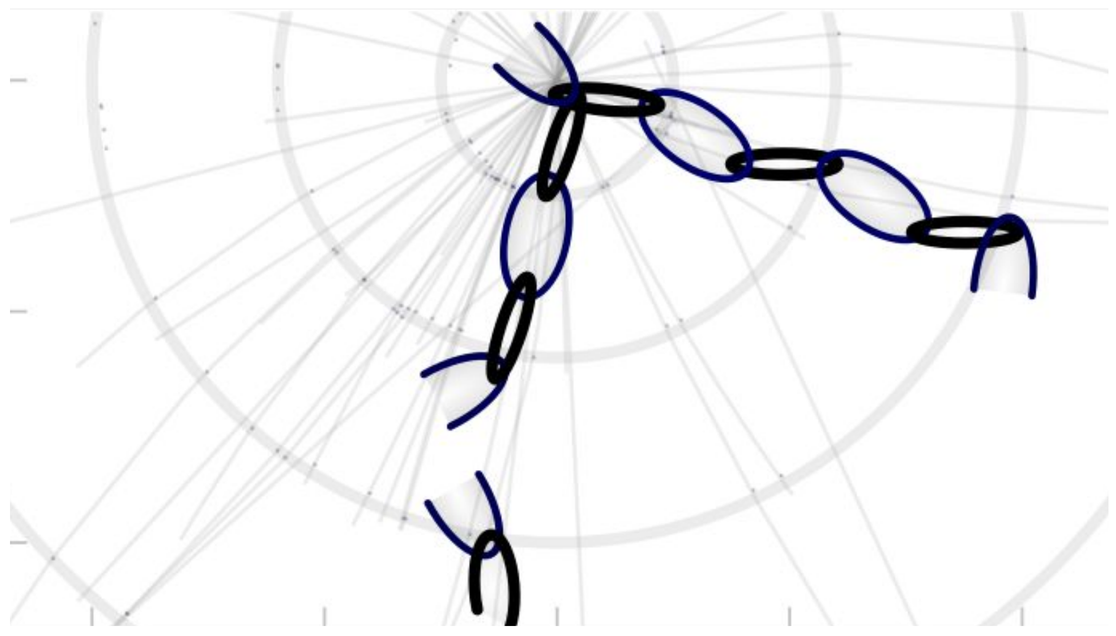
Tested successfully with Athena (ATLAS) & Gaudi (FCC)

Different approach prototyped:



Friends&Family: TrickTrack





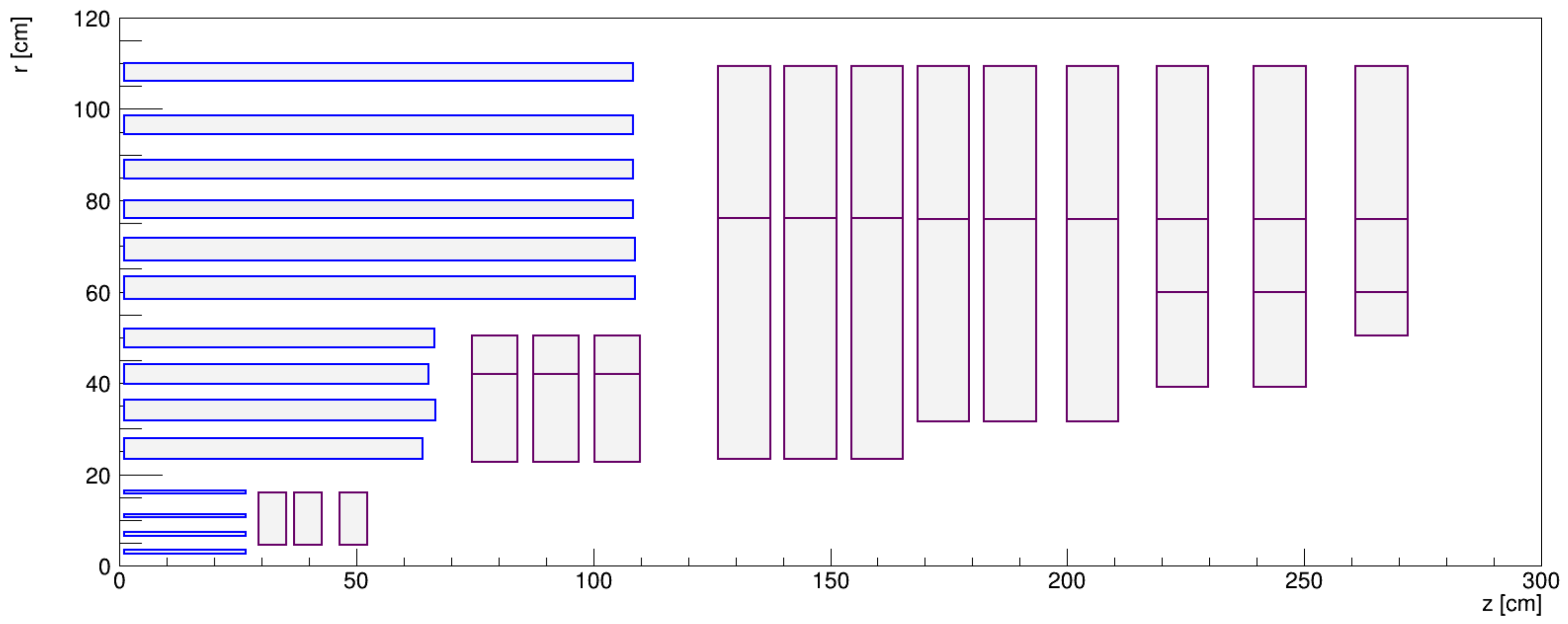
https://github.com/HSF/TrickTrack

TrickTrack is the encapsulated Cellular-Automaton library from CMSSW

- header-only library templated on input data type, easy to integrate into your SW framework
- builds n-dimensional track candidates
- minimal dependency: Eigen

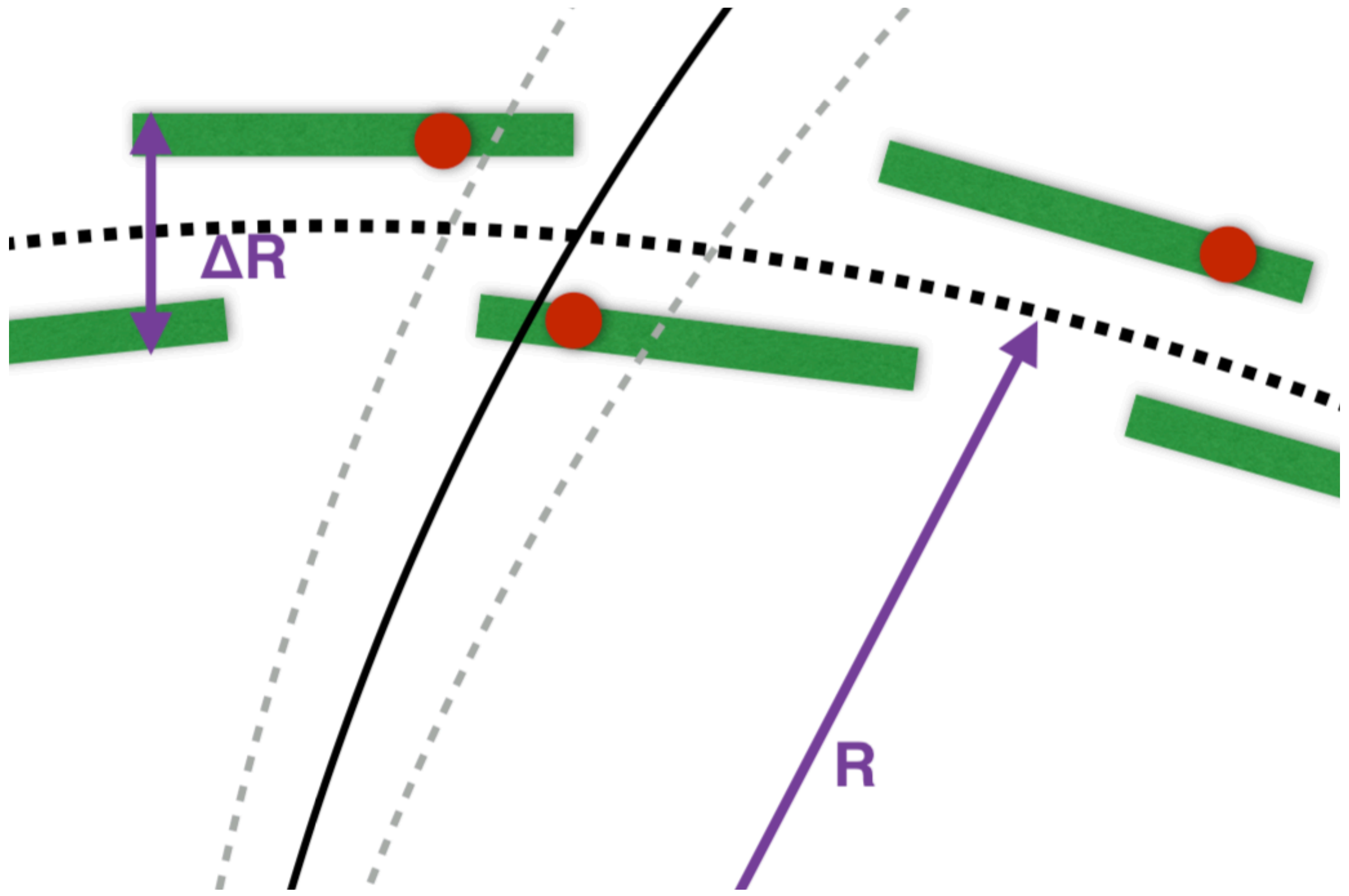
see CTD2018 talk by V. Voelkl

Friends&Family: mkFit



mkFit is a vectorized track fitting and track finding approach developed with CMSSW

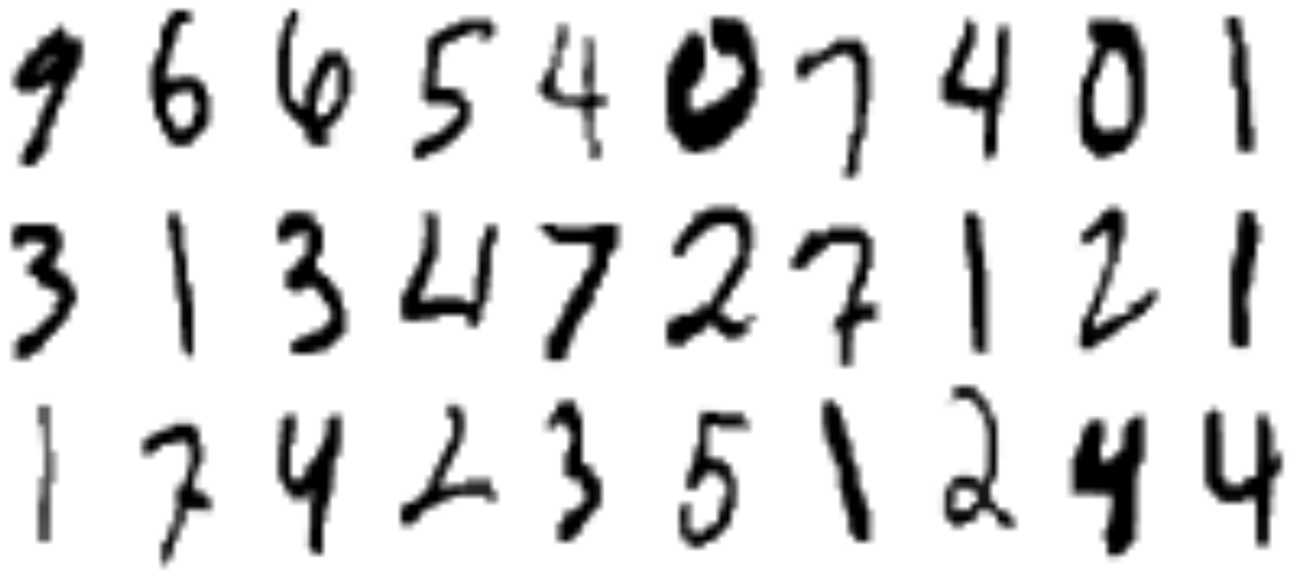
- embedded in CMSSW
- extended to realistic detector geometry
- core of vectorization is a code generator for matrix multiplications



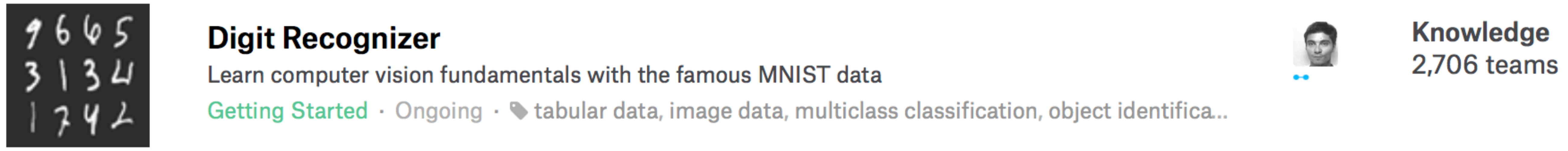
see CTD2018 talk by M. Tadel

Wednesday in T2

Reference dataset



Reference datasets used **very successfully** in many fields
- particularly machine learning



e.g. MNIST ("Modified National Institute of Standards and Technology")

Stating the obvious

- allows for traceability of algorithm development and progress
- allows apple-to-apple comparisons
- allows to perform mid-scale migrations (e.g. Eigen vs. SMatrix comparison)

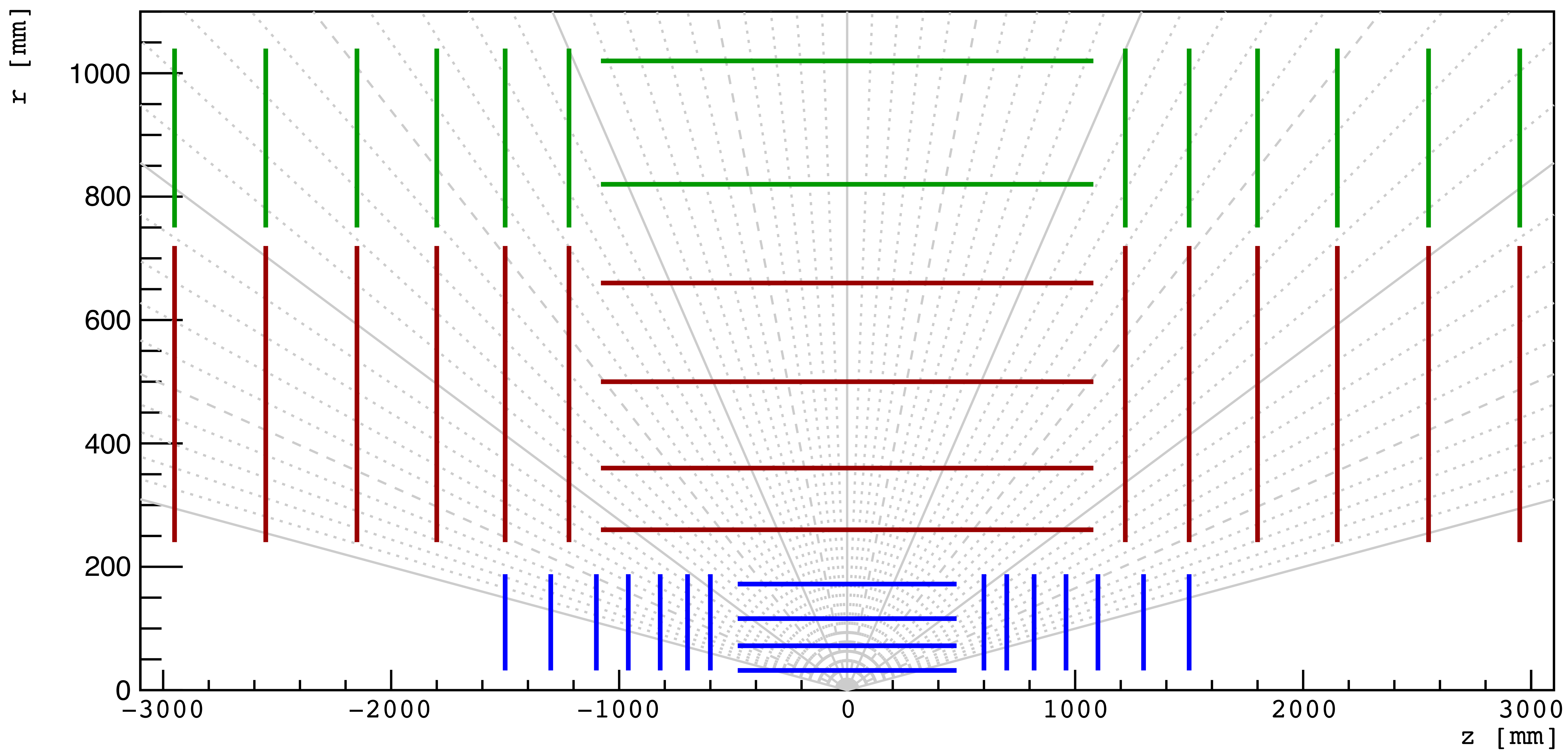
Stating the *less* obvious

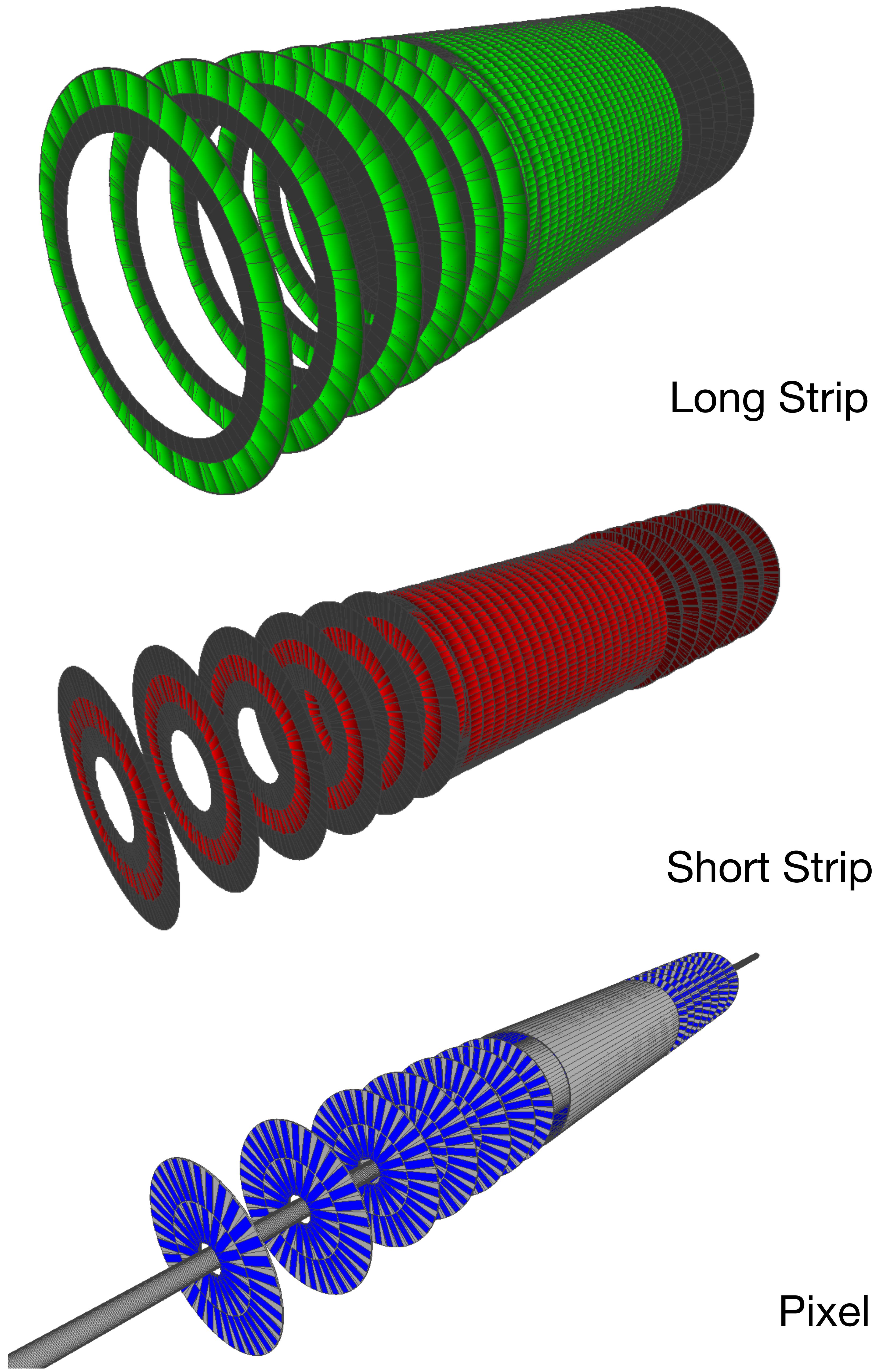
- eases publication of algorithmic research

Reference detector&dataset

Common dataset for development within the community

- detector used for TrackML
- dataset produced with **ACTS fast simulation**
- **proposal**: iron out the few features we discovered & dataset (LHC/HL-LHC), publish on opendata CERN

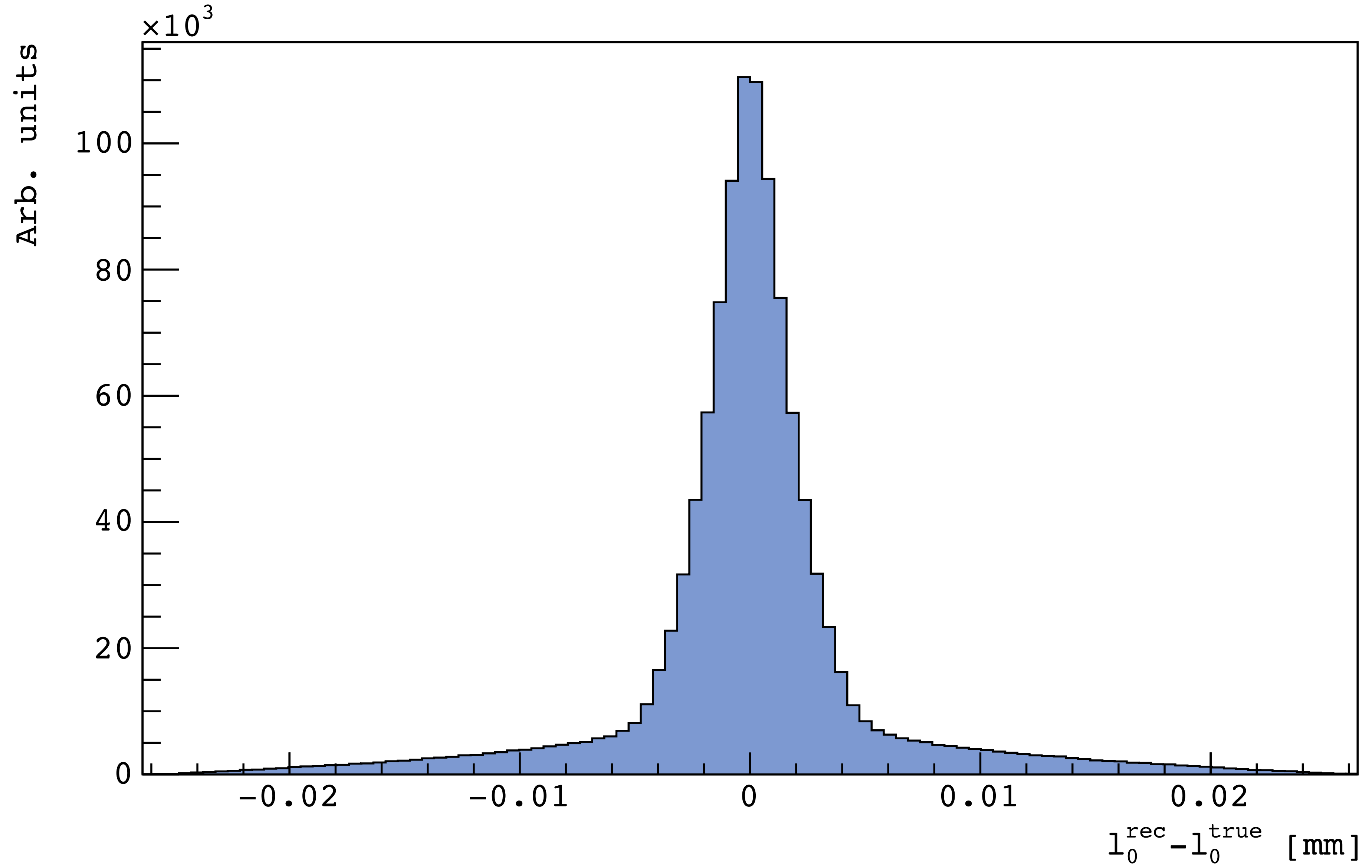




Reference detector&dataset

Quasi-realistic full silicon detector

- non-Gaussian measurements, with *realistic* cluster shapes
- *realistic* material budget
- main particle-material interactions



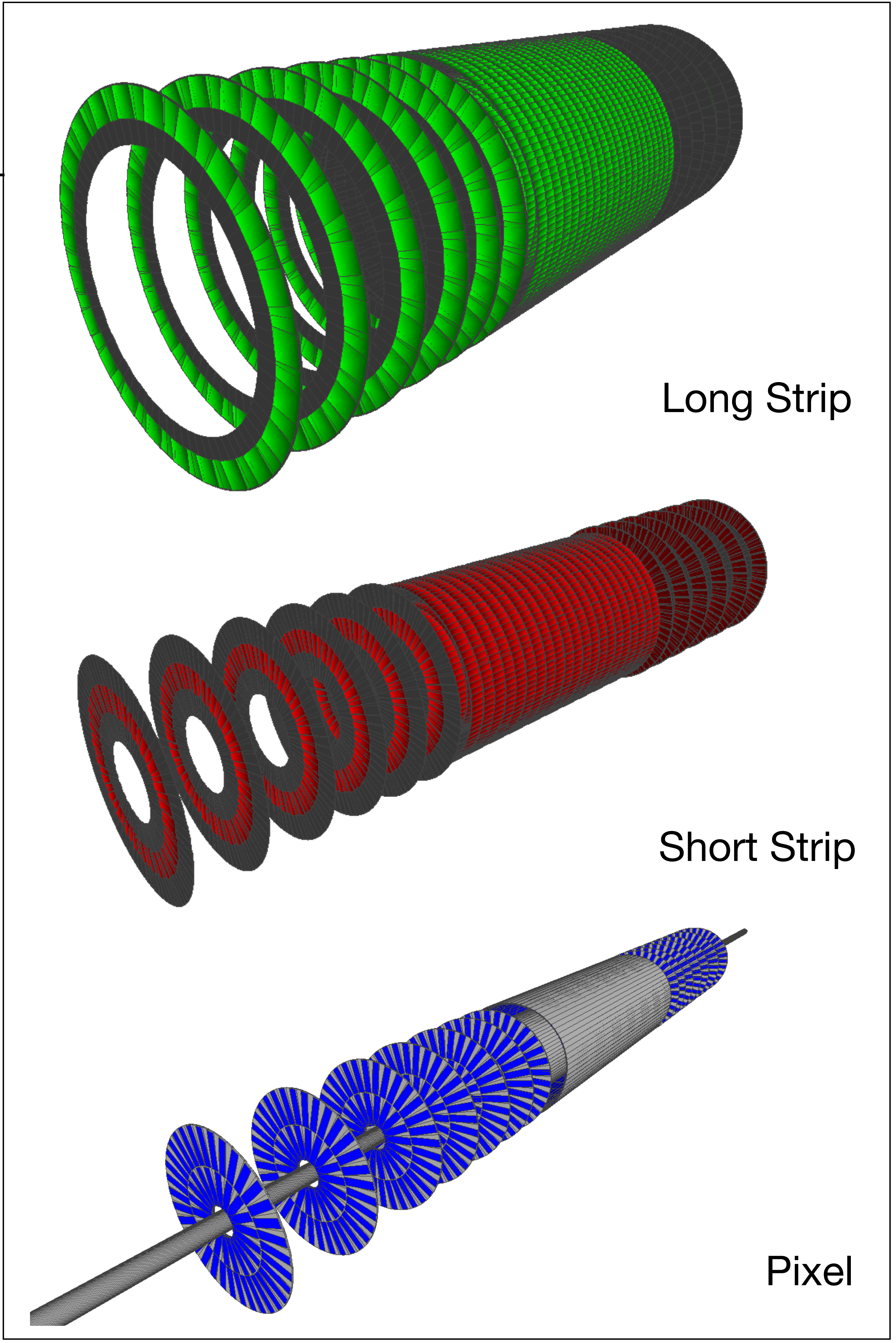
Pixel residuals for TrackML detector
(50 µm x 50 µm pixel size)

opendata CERN

development & testing ecosystem

Experiment SW ecosystem

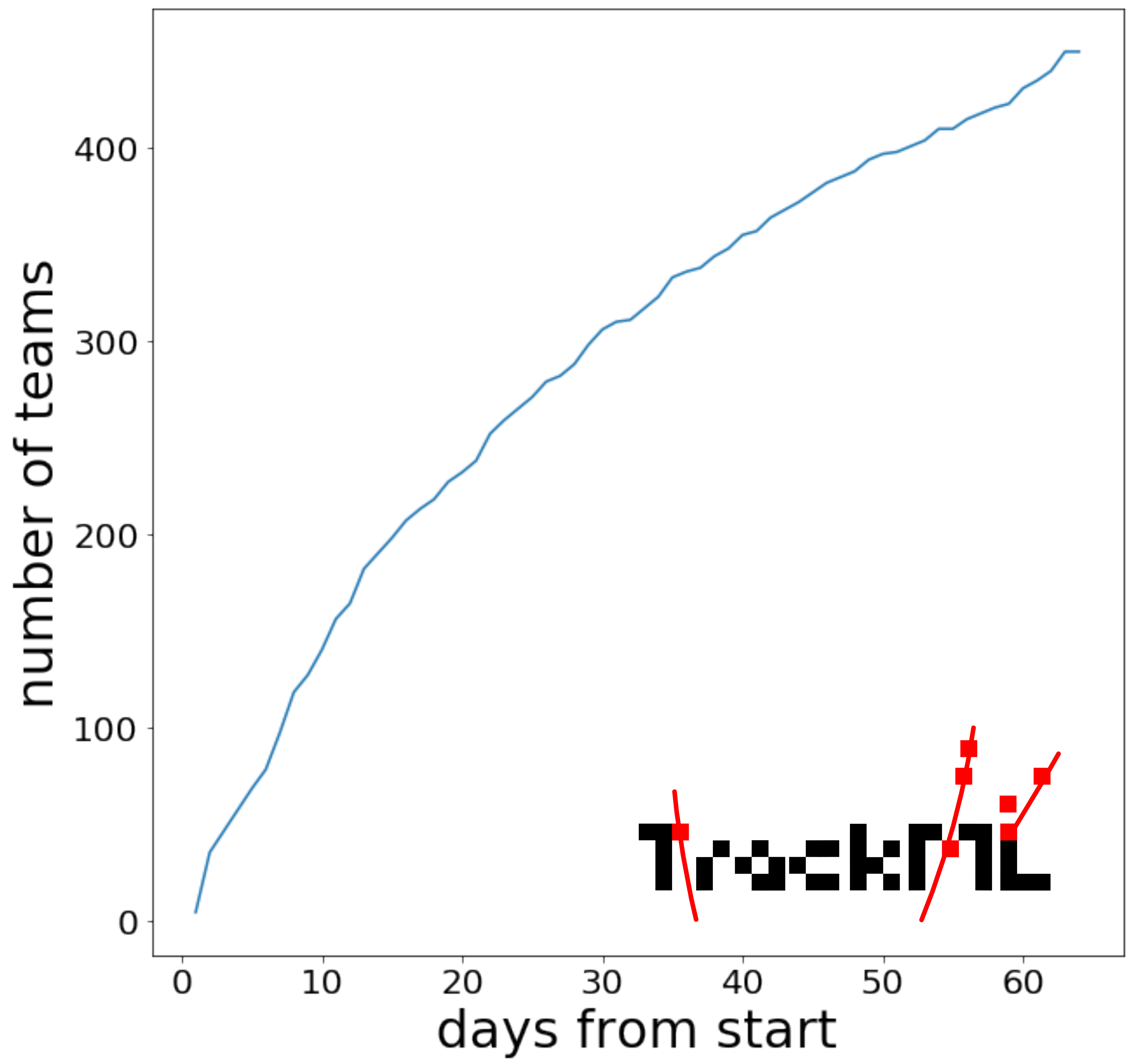
publish

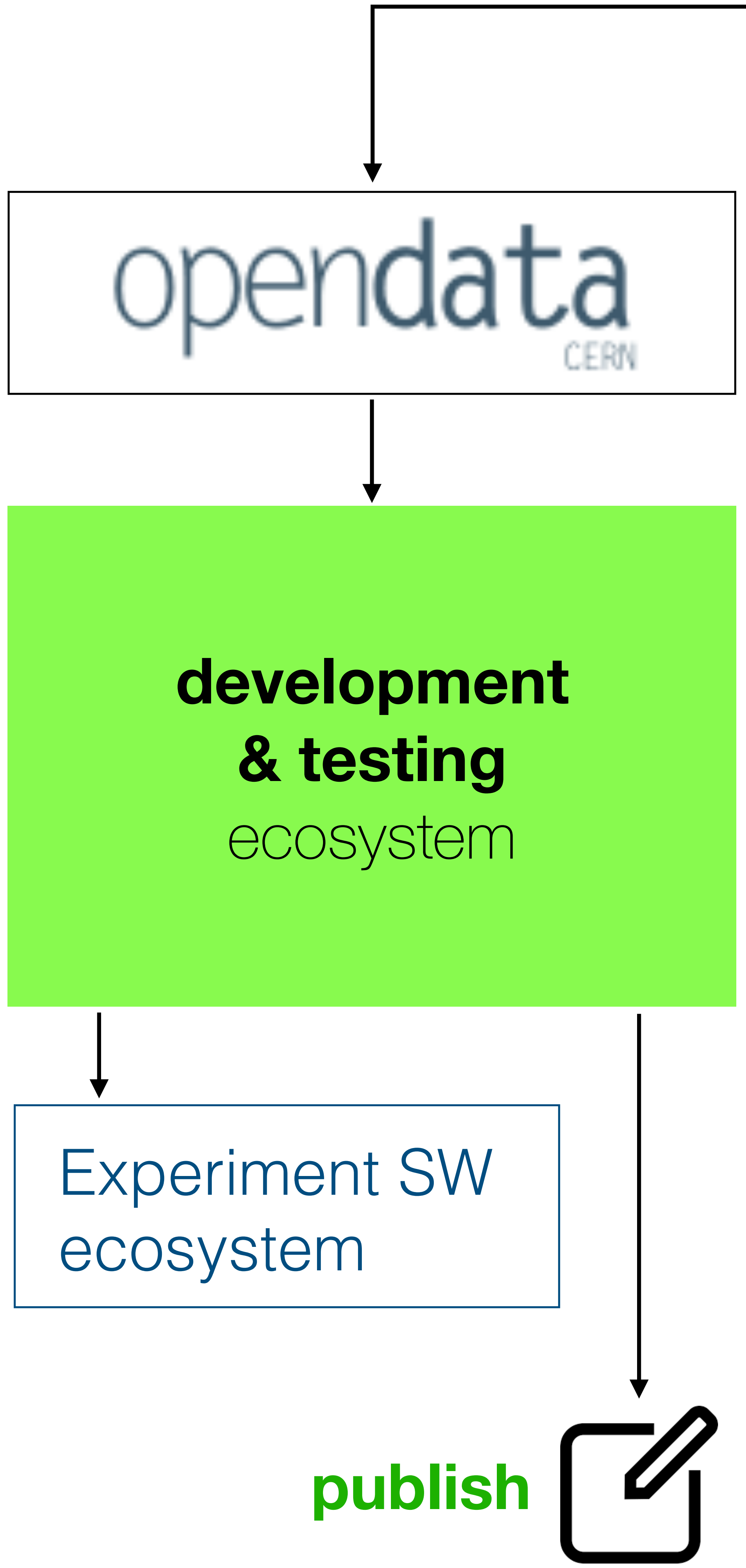


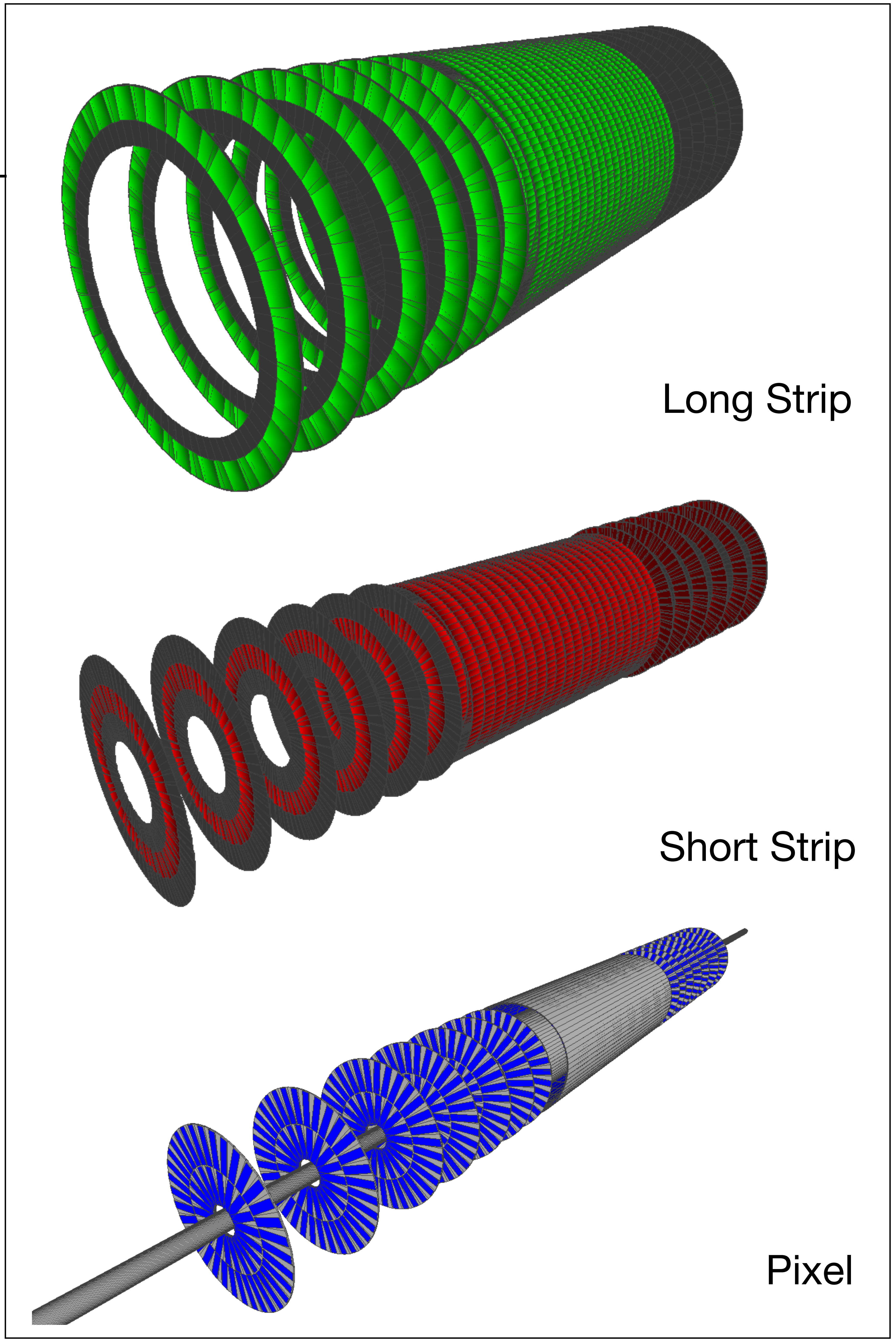
Reference detector&dataset

Enlarge community

- there are a lot of experts & enthusiasts







Summary

HL-LHC and FCC-hh environment will need re-thinking of current track reconstruction

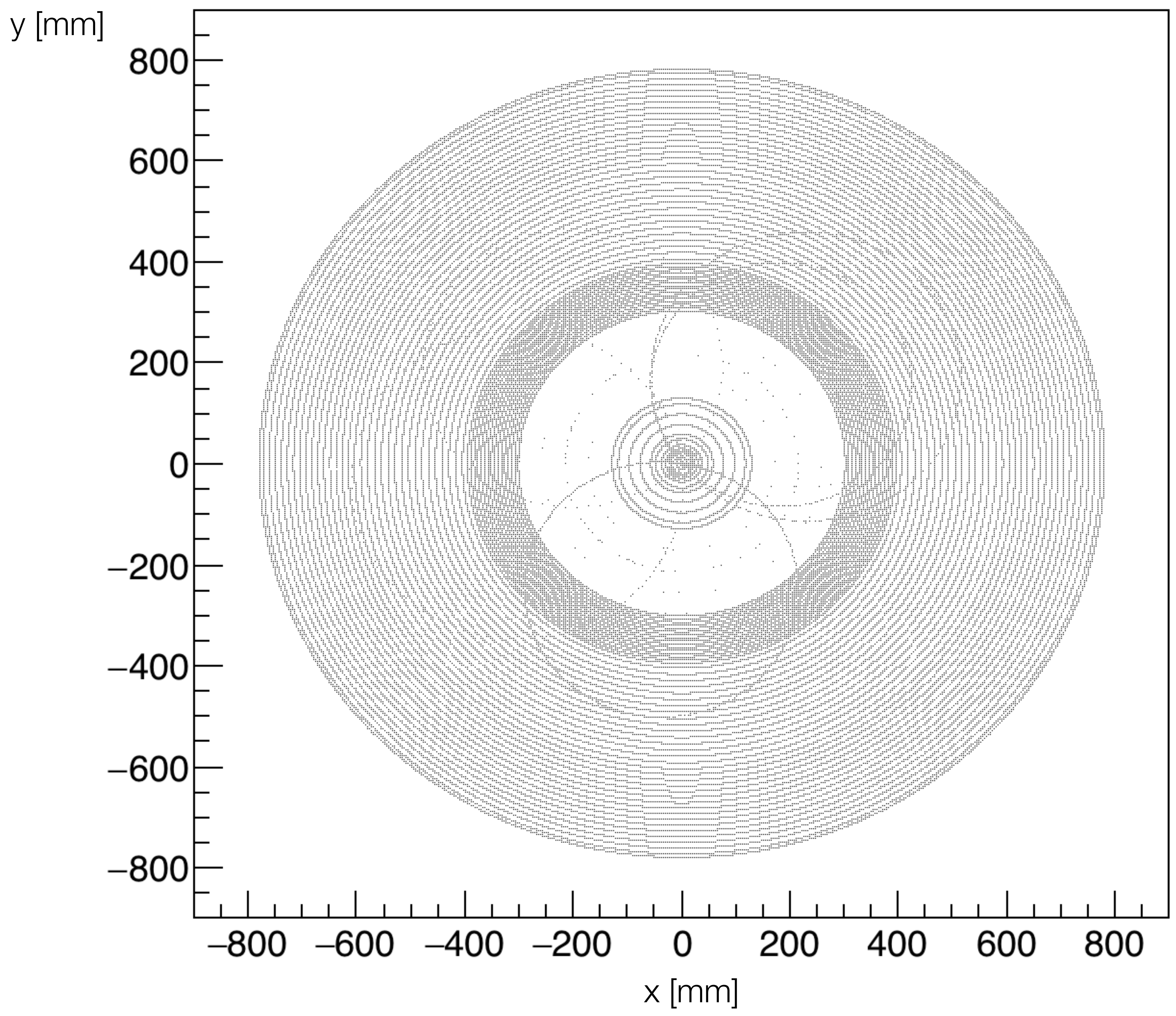
- time of low-hanging code optimization has passed
- need real R&D for data structures, algorithms, steering
  (and it is happening already)
- encourage to develop and share across experiment boarders
  (and it is happening already)
- open software and public license is key to this
- with ACTS we try to do achieve this regarding ATLAS tracking code
- we attempt to establish a turn-key software for development
  with a feedback path into the experiment code

Reference dataset

- a very useful concept that is common to several fields
- should help to share, interact & publish

Backup

Geometry



Problematic:

- pure parsing the geometry is always difficult
- not recommended, direct integration is certainly better

```
./ACTFWRootPropagationExample --bf-values 0 0 1.4 --prop-eta-range -0.5 0.5 --geo-root-filename=sPhenix.root --
geo-root-worldvolume="World"  --geo-subdetectors ITSU Silicon SVTX --geo-root-nlayers=0 0 0 --geo-root-clayers=3
4 48 --geo-root-players=0 0 0 --geo-root-claynames=ITSUHalfStave0 ITSUHalfStave1 ITSUHalfStave2
ladder_volume_3 ladder_volume_4 ladder_volume_5 ladder_volume_6 SVTX_7_ SVTX_8_ SVTX_9_ SVTX_10_ SVTX_11_
SVTX_12_ SVTX_13_ SVTX_14_ SVTX_15_ SVTX_16_ SVTX_17_ SVTX_18_ SVTX_19_ SVTX_20_ SVTX_21_ SVTX_22_ SVTX_23_
SVTX_24_ SVTX_25_ SVTX_26_ SVTX_27_ SVTX_28_ SVTX_29_ SVTX_30_ SVTX_31_ SVTX_32_ SVTX_33_ SVTX_34_ SVTX_35_
SVTX_36_ SVTX_37_ SVTX_38_ SVTX_39_ SVTX_40_ SVTX_41_ SVTX_42_ SVTX_43_ SVTX_44_ SVTX_45_ SVTX_46_ SVTX_47_
SVTX_48_ SVTX_49_ SVTX_50_ SVTX_51_ SVTX_52_ SVTX_53_ SVTX_54_ --geo-root-cmodulenames ITSUSensor0 ITSUSensor1
ITSUSensor2 siactive_volume_3 siactive_volume_4 siactive_volume_5 siactive_volume_6 SVTX_7_ SVTX_8_ SVTX_9_
SVTX_10_ SVTX_11_ SVTX_12_ SVTX_13_ SVTX_14_ SVTX_15_ SVTX_16_ SVTX_17_ SVTX_18_ SVTX_19_ SVTX_20_ SVTX_21_
SVTX_22_ SVTX_23_ SVTX_24_ SVTX_25_ SVTX_26_ SVTX_27_ SVTX_28_ SVTX_29_ SVTX_30_ SVTX_31_ SVTX_32_ SVTX_33_
SVTX_34_ SVTX_35_ SVTX_36_ SVTX_37_ SVTX_38_ SVTX_39_ SVTX_40_ SVTX_41_ SVTX_42_ SVTX_43_ SVTX_44_ SVTX_45_
SVTX_46_ SVTX_47_ SVTX_48_ SVTX_49_ SVTX_50_ SVTX_51_ SVTX_52_ SVTX_53_ SVTX_54_ --geo-root-cmoduleaxes=xzy xzy
xzy zyx zyx zyx zyx zyx xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz
xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz xyz
```

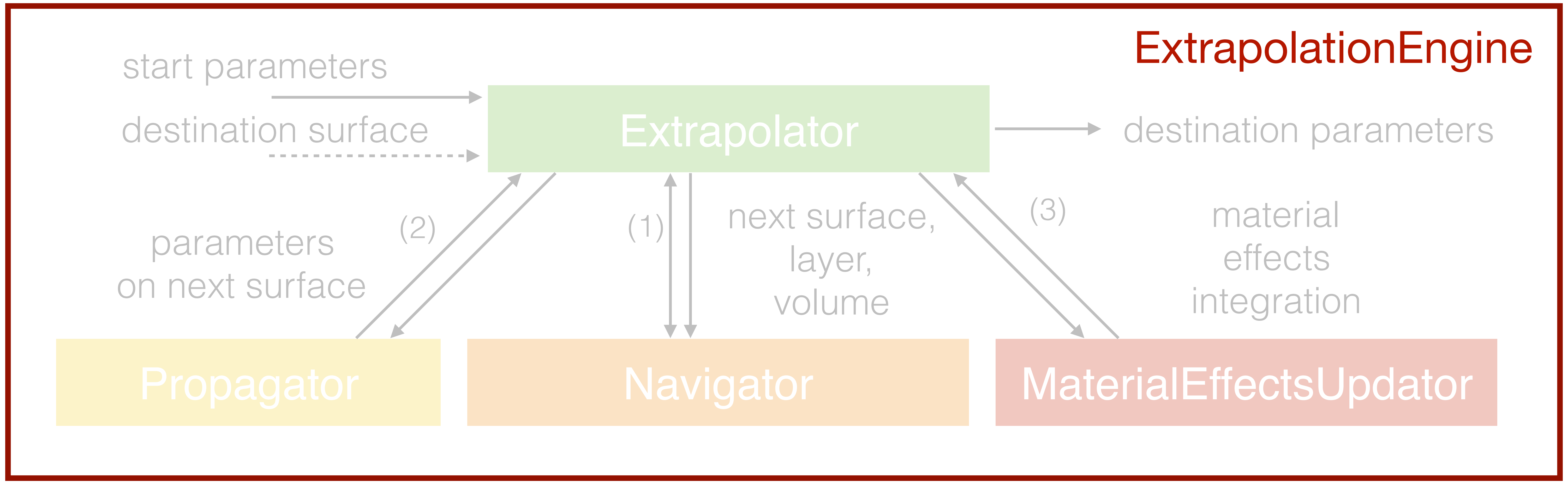
Units

Units are difficult to harmonize

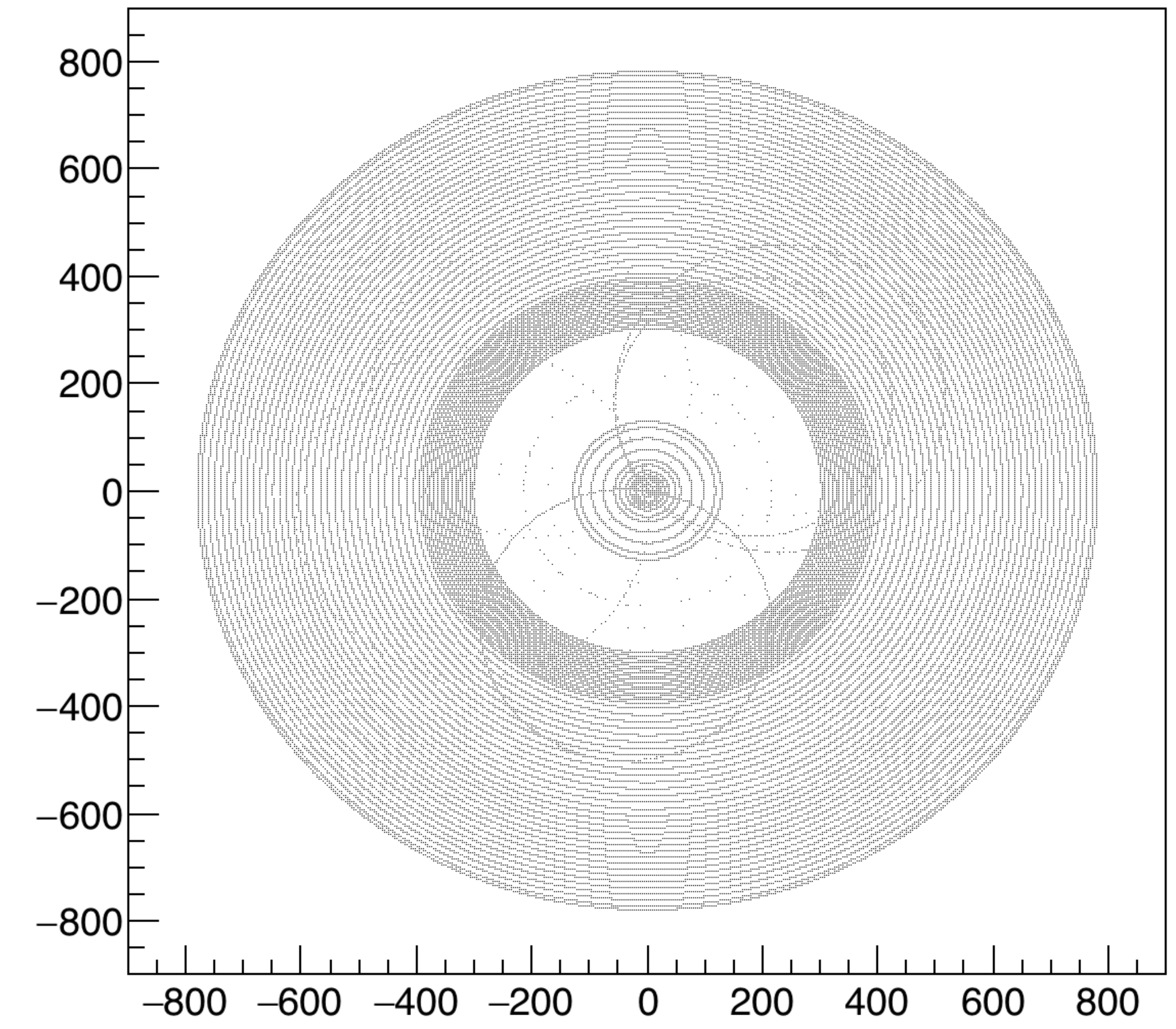
- try to use units consistently throughout the ACTS code (an we write a static code parser for this ?)
- can be interleaved with experiment SW with Units conversions constant

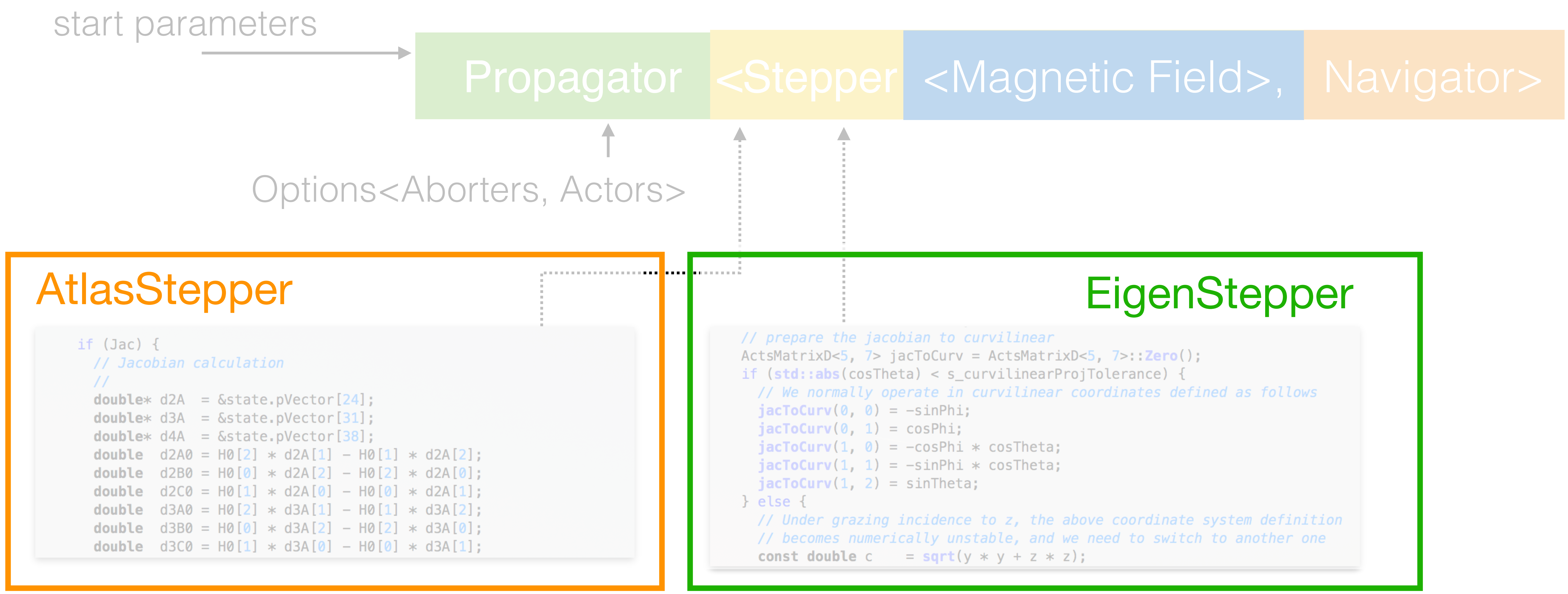
```
#pragma once
namespace Acts {

/// @brief Unit and conversion constants
///
/// In order to make sure that always consistent numbers and units are used in
/// calculations, one should make use of the constants defined in this
/// namespace to express the units of numerical values. The following
/// conventions are used:
/// - Input values to Acts must be given as `numerical_value * unit_constant`.
/// - Output values can be converted into the desired unit using
///   `numerical_value / unit_constant`.
/// - Converting between SI and natural units should be done using the template
///   functions `SI2Nat(const double)` and `Nat2SI(const double)`.
///
/// Examples:
/// @code
/// #include "Acts/include/Utilities/Units.hpp"
/// using namespace Acts::units;
/// // specify input variables
/// double momentum = 2.5 * _GeV;
/// double width    = 23 * _cm;
/// double velocity = 345 * _m/_s;
/// double density  = 1.2 * _kg/(_m*_m*_m);
/// double bfield   = 2 * _T;
///
/// // convert output values
/// double x_in_mm   = trackPars.position().x() / _mm;
/// double pt_in_TeV = trackPars.momentum().pT() / _TeV;
/// @endcode
namespace units {

/// @name length units
/// @{
#ifdef DOXYGEN
  const double _m = unspecified;
#else
  const double _m = 1e3;
#endif  // DOXYGEN
  const double _km = 1e3 * _m;
  const double _cm = 1e-2 * _m;
  const double _mm = 1e-3 * _m;
  const double _um = 1e-6 * _m;
  const double _nm = 1e-9 * _m;
  const double _pm = 1e-12 * _m;
  const double _fm = 1e-15 * _m;
/// @}
```

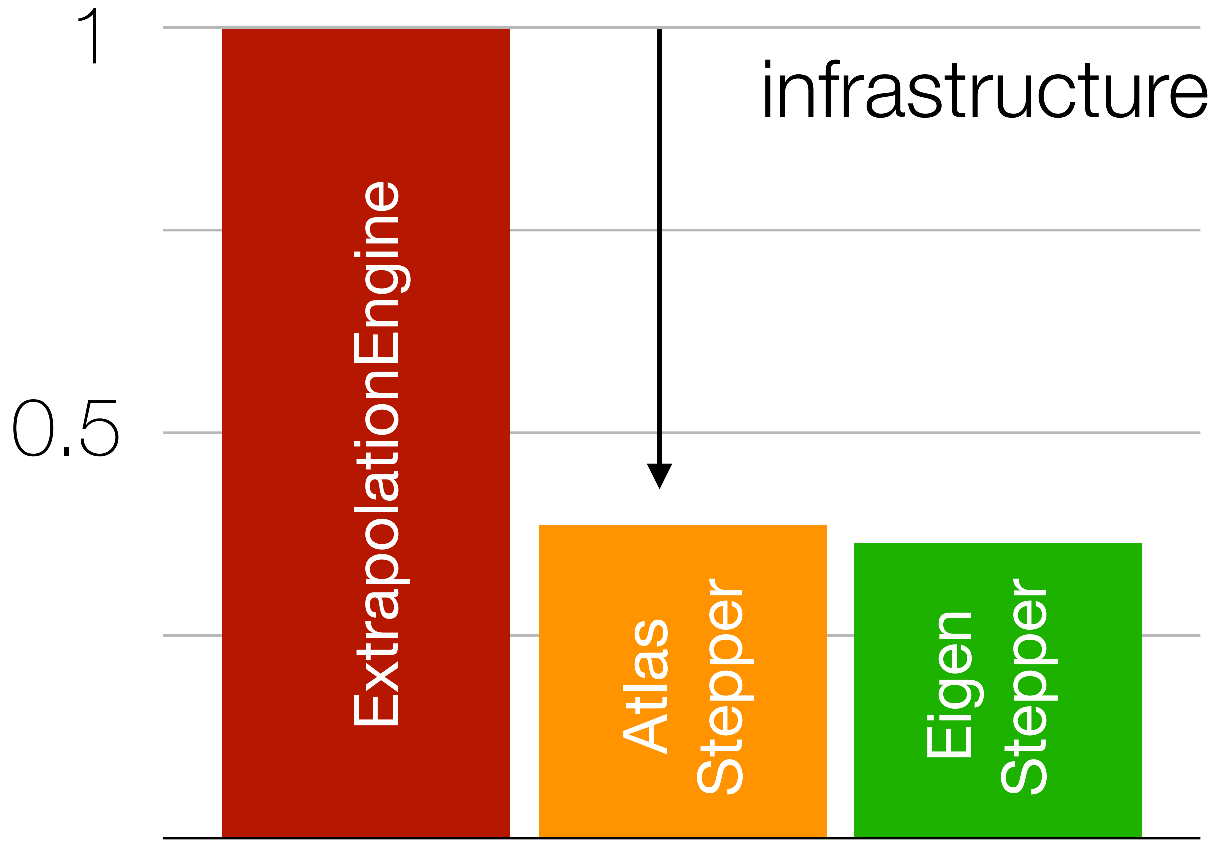
Timing



Constant field







Reference

Results shown are done with acts branches

acts-core: 1bdf3177..d30185b5 ACTS-431_Extrapolator
acts-fatras: 15ca56d..4d6e766 ACTSFW-112_
acts-framework: 6f0f1e2..344500b ACTSFW-112_Fatras_New_Propagator