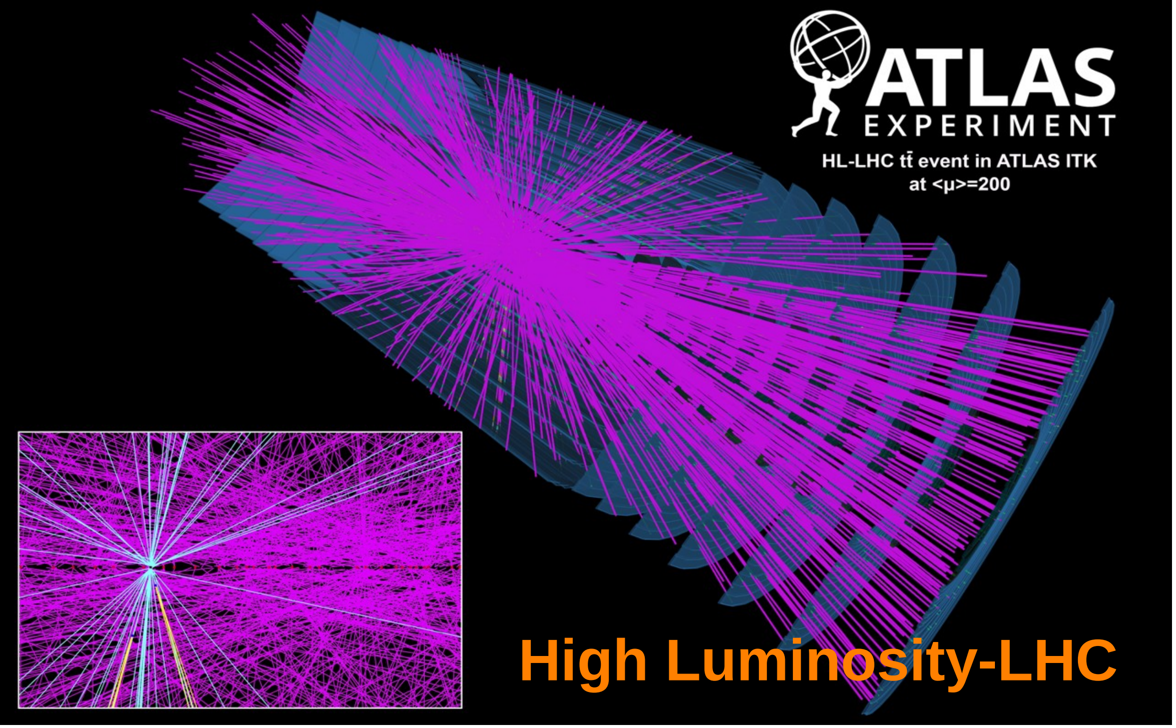
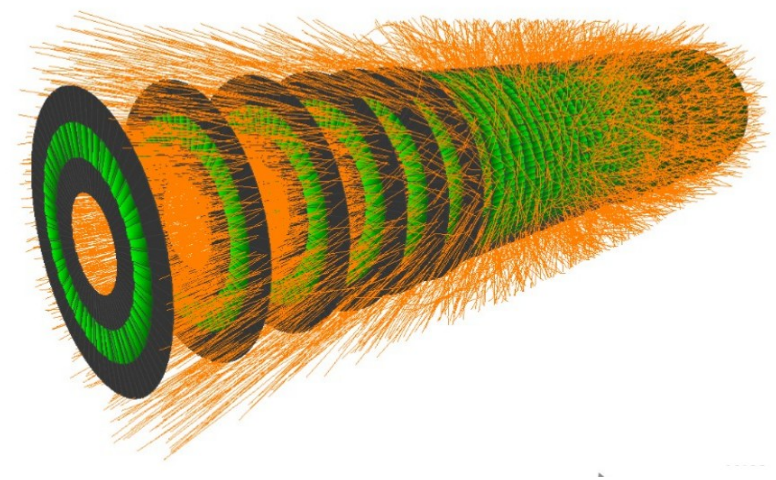


# Tracking with acts

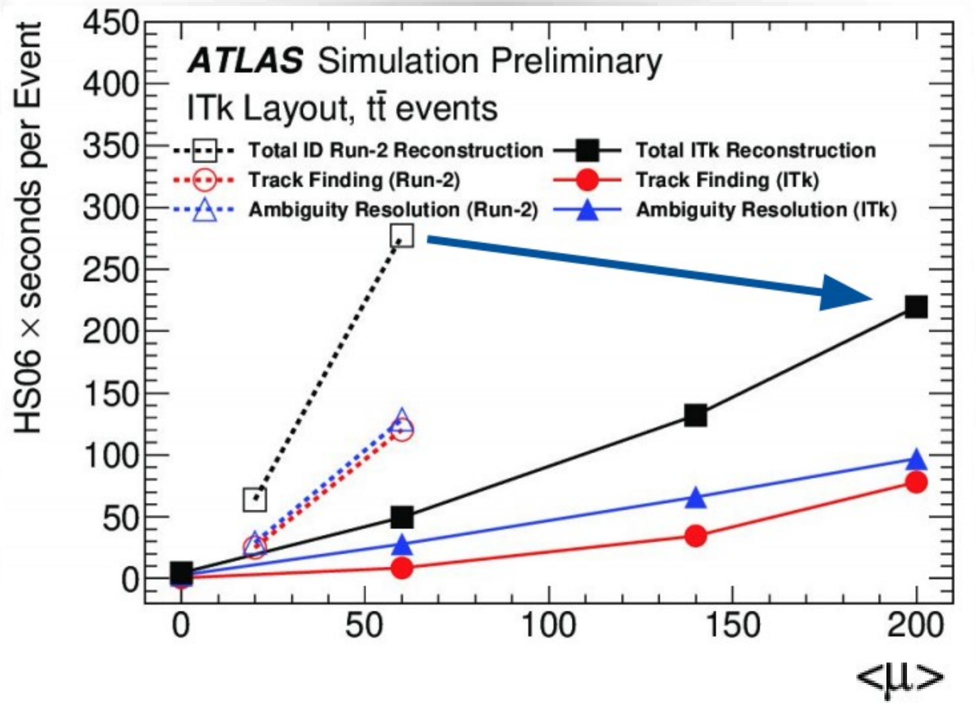
Based on my slides at ACTS tracking workshop 2020

Xiaocong Ai for the ACTS developers
Xiaocong.ai@berkeley.edu

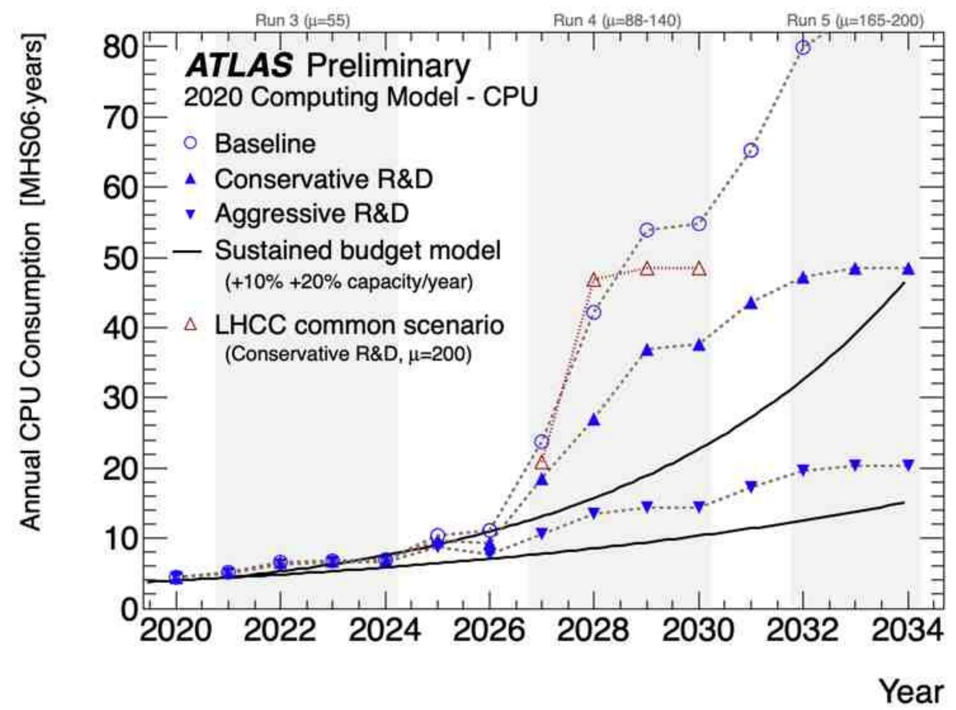
**EIC Software Seminar, Jun 17, 2020**

# The tracking challenge

- Much increased combinatorics with high pileup at future hadron colliders
  - e.g. ~6k particles/event with $\mu$ = 200 at HL-LHC




**Increased track reconstruction time**




**Increased CPU consumption**

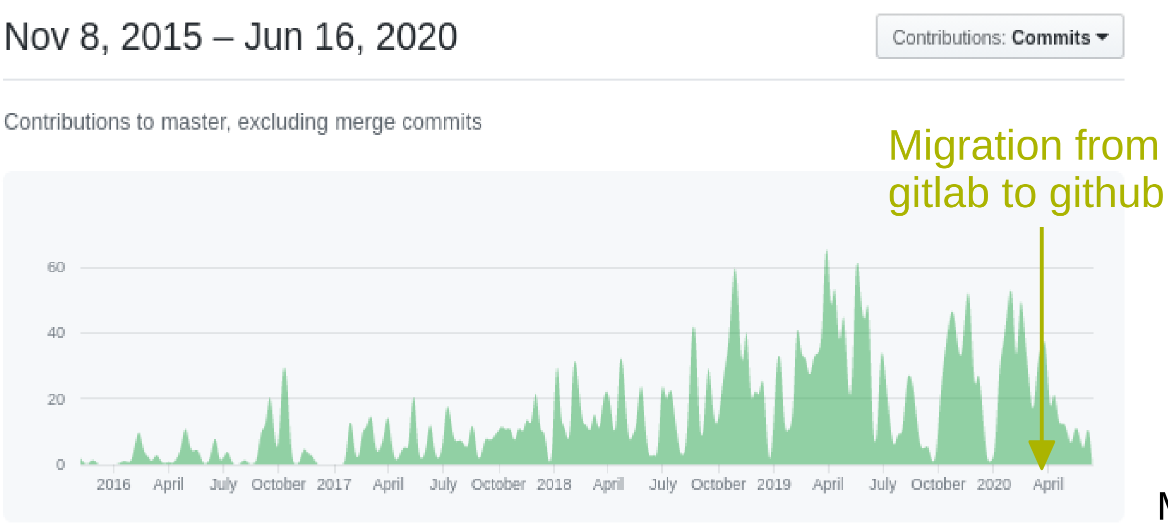
**Accurate, efficient and fast tracking software is needed to achieve physics goals**

- Could we benefit from fast tracking techniques, parallelism and acceleration?

# Goals, Design and Components

# ACTS goals

- Prepare an experiment-independent tracking toolkit for future detectors based on ATLAS tracking experience (well tested but thread-unsafe, difficult to maintain)
  - Targeting at ATLAS at HL-LHC, but also for other experiments, e.g. sPHENIX, Belle-II, CEPC etc.
- Provide an open-source R&D platform for new tracking techniques and hardware achitectures

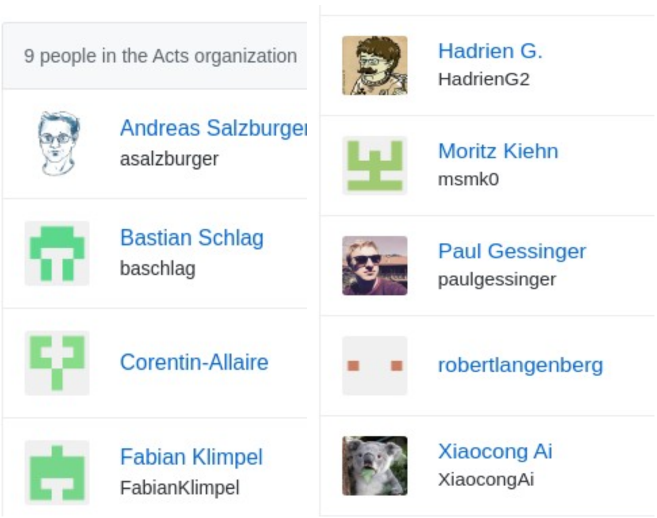



Jun 2016 - **v0.01.00**

Latest: **v0.26.00**

https://github.com/acts-project/acts

Current main contributors to the repository

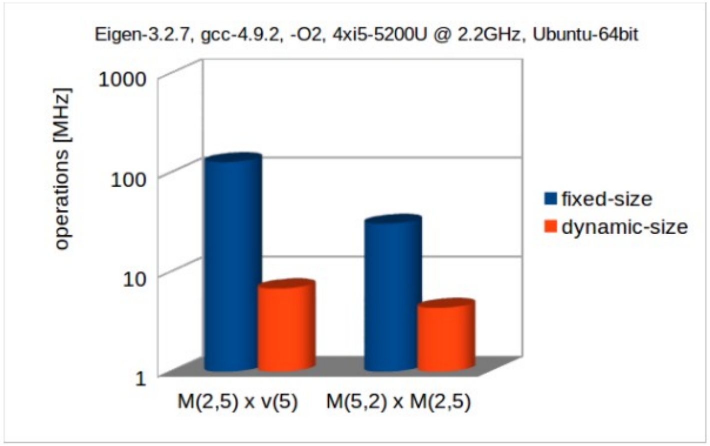



Members from US side:

- **UC Berkeley**: Xiaocong Ai, Heather Gray, Irina Ene
- **LBNL**: Charles Leggett
- **Stanford University**: Lauren Tompkins, Rocky Bala Garg
- **Oak Ridge National Laboratory**: Joe Osborn
- **Florida State University**: Tony Frawley

# ACTS concepts & design

- Modern C++ 17 concepts
- Highly-templated design to avoid virtual lookup
  - Detector and magnetic field agnostic
- Efficient memory allocation and access
  - Eigen-based Event Data Model (EDM)
  - Uses fixed-size EDM as much as possible
- Strict thread-safety to facilitate concurrency
  - Const-correctness, stateless tools
- Supports for contextual condition data
  - Allow event execution with different Geometry/Calibration/Magnetic field in flight




```
/// An Acts Tool that does some well defined job
class MyTool {
  public:
  /// Cache for Acts tool
  ///
  /// it is exposed to public for use of the expert-only
  /// propagate_with_cache method of the propagator
  struct Cache
  {
    variable_type var; ///< the local cache
  }

  /// Method that performs an operation and also relies on a cache
  ///
  /// @param cache object
  /// @param input variable that is needed by this method
  ///
  /// @return a direct output of this method
  return_type doSomethingWithCache(Cache& cachem, const input_type& input) const;

};
```

```
size_t algorithmNumber;             ///< Unique algorithm identifier
size_t eventNumber;                 ///< Unique event identifier
WhiteBoard& eventStore;             ///< Per-event data store
Acts::GeometryContext geoContext;   ///< Per-event geometry context
Acts::MagneticFieldContext
    magFieldContext;                     ///< Per-event magnetic Field context
Acts::CalibrationContext calibContext;  ///< Per-event calibration context
```

# ACTS concepts & design

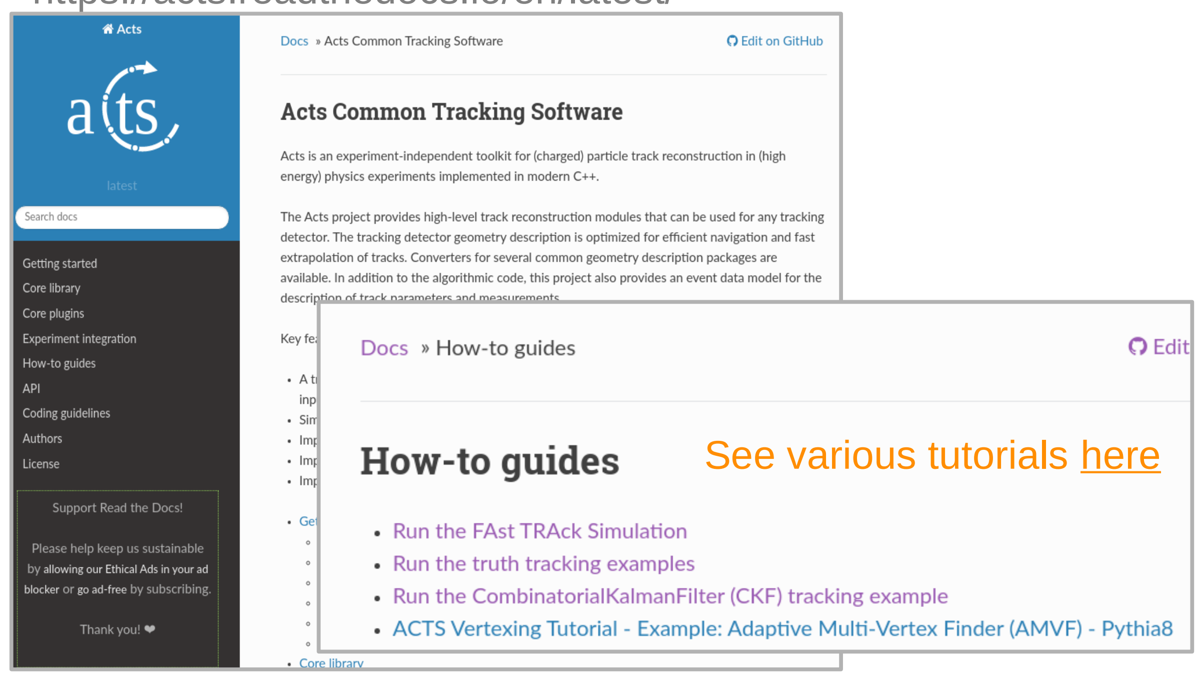
- Minimal dependencies
  - Eigen library
  - Boost (only for building unit test)
- Rigorous unit tests
- Highly configurable for usability
- Well-documented
  - On-going efforts for further improvement

```cpp
// Global definitions
// The path limit abort
using path_limit = PathLimitReached;

using BFieldType = ConstantBField;
using EigenStepperType = EigenStepper<BFieldType>;
using EigenPropagatorType = Propagator<EigenStepperType>;

const double Bz = 2_T;
BFieldType bField(0, 0, Bz);
EigenStepperType estepper(bField);
EigenPropagatorType epropagator(std::move(estepper));

auto mCylinder = std::make_shared<CylinderBounds>(10_mm, 1000_mm);
auto mSurface = Surface::makeShared<CylinderSurface>(nullptr, mCylinder);
auto cCylinder = std::make_shared<CylinderBounds>(150_mm, 1000_mm);
auto cSurface = Surface::makeShared<CylinderSurface>(nullptr, cCylinder);

const int ntests = 5;

// This tests the Options
BOOST_AUTO_TEST_CASE(PropagatorOptions_) {
  using null_optionsType = PropagatorOptions<>;
  null_optionsType null_options(tgContext, mfContext);
  // todo write null options test

  using ActionListType = ActionList<PerpendicularMeasure>;
  using AbortConditionsType = AbortList<>;

  using optionsType = PropagatorOptions<ActionListType, AbortConditionsType>;

  optionsType options(tgContext, mfContext);
}
```

```cpp
/// An Acts Tool that does some well defined job
class MyTool {
  public:
  /// Configuration struct
  struct Config {
    variable_type parameter;
  };

  /// Tool constructor
  ///
  /// @param cfg is the configuration struct
  MyTool(const Config&cfg) : m_cfg(cfg){}

  private:
    Config m_cfg; ///< the configuration object
};
```

```cpp
// Create a config object and set my configuration parameter
MyTool::Config mtConfig;
mtConig.parameter = 3.1415927;

// Now create a tool with this configuration
MyTool mt(mtConfig);
```

https://acts.readthedocs.io/en/latest/




See various tutorials here

# ACTS components and functionalities

Continuous tracking infrastructure consolidation and tools completation

acts-core

- CI
- Core
  - EventData
  - Fitter
  - Geometry
  - MagneticField
  - Material
  - Propagator
  - Seeding
  - Surfaces
  - TrackFinder
  - Utilities
  - Vertexing
  - Visualization
- Examples
- Fatras
- Plugins
- Tests

acts-fatras

- Fast simulation engine
- Integrated into acts-core

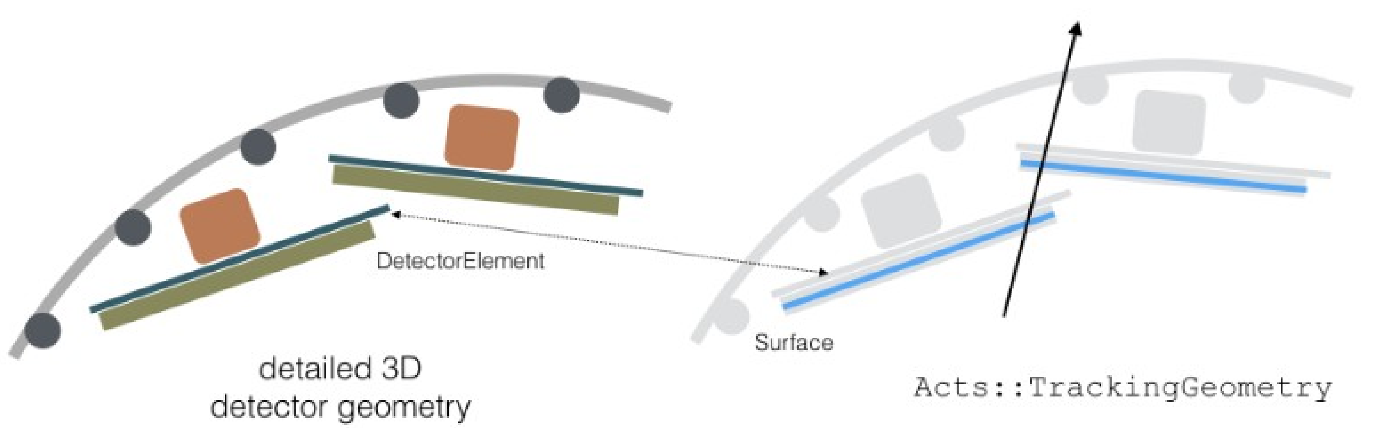
acts-framework

- A light-weight Gaudi-style test framework for event processing, integration and concurrency test
- Integration into acts-core as examples to test core implementation

# Geometry and Navigation

# Geometry building

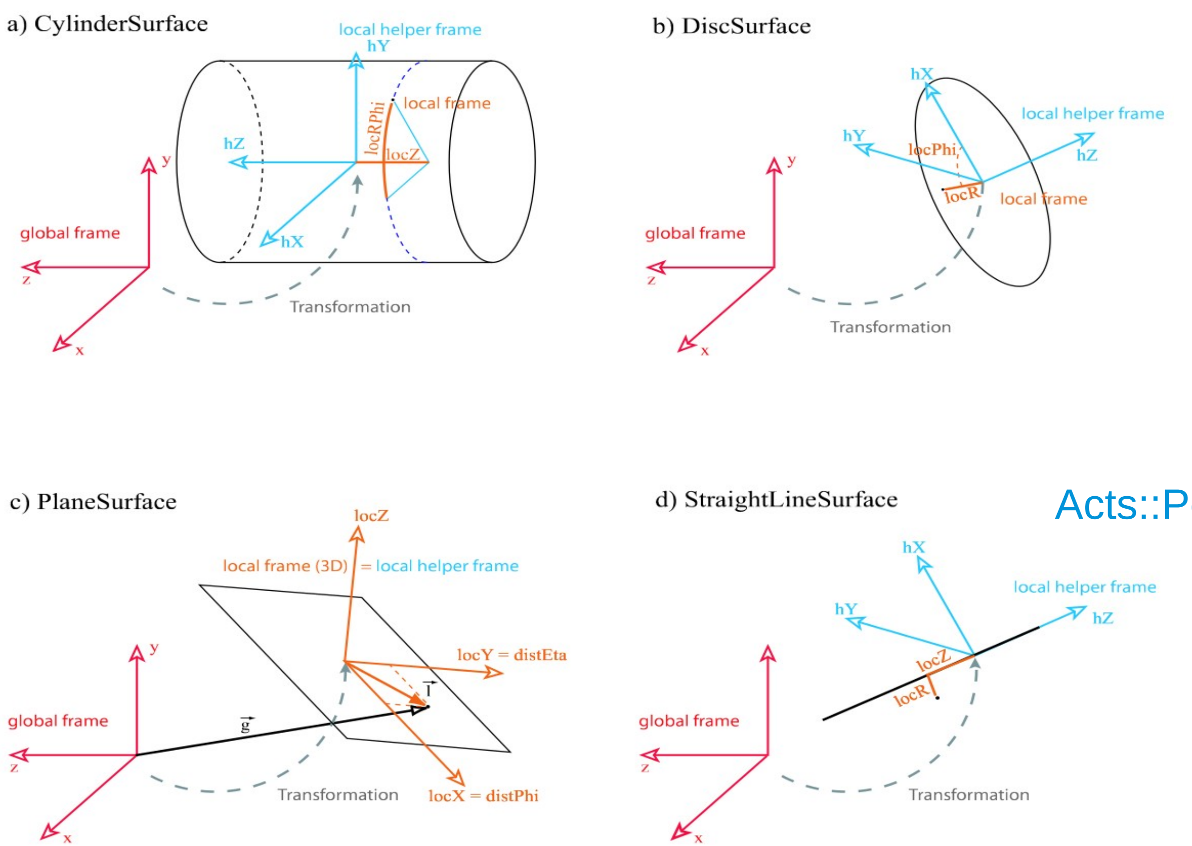
- To reduce CPU consumption and navigation speed-up, tracking geometry (i.e. geometry used for track reconstruction) is simplified from full simulation geometry
  - Binding via Acts::DetectorElementBase which can be converted from other detector element representation via geometry plugins:
    - TGeo (Acts::TGeoDetectorElement)
    - DD4hep (Acts::DD4HepDetectorElement)
- Implemented HEP detector geometry
  - Silicon, Calorimeter, MuonSpectrometer



# The Surface class

- Acts::Surface is the key component of tracking geometry
  - Surface concepts are largely transcribed from ATLAS SW
- Different concrete surfaces have different local coordinate definitions and shapes
  - Shape is described by Acts::SurfaceBounds

**Surface types in ATLAS SW**




Acts::LineSurface

Acts::PerigeeSurface    Acts::StrawSurface

Defined by beamline/wire's direction and vector from beamline/wire to closest approach of the track

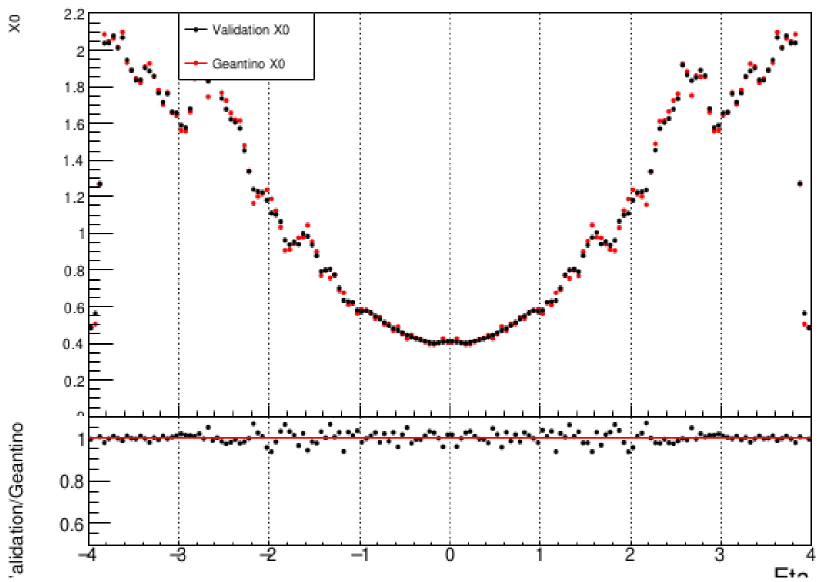
# Material description

see C. Allaire's slides

- Material effects need to be considered in tracking
- Material mapping tools allows to map (averaged) Gean4-based full detector material (recorded using Geantino scan) onto either surfaces or volumes

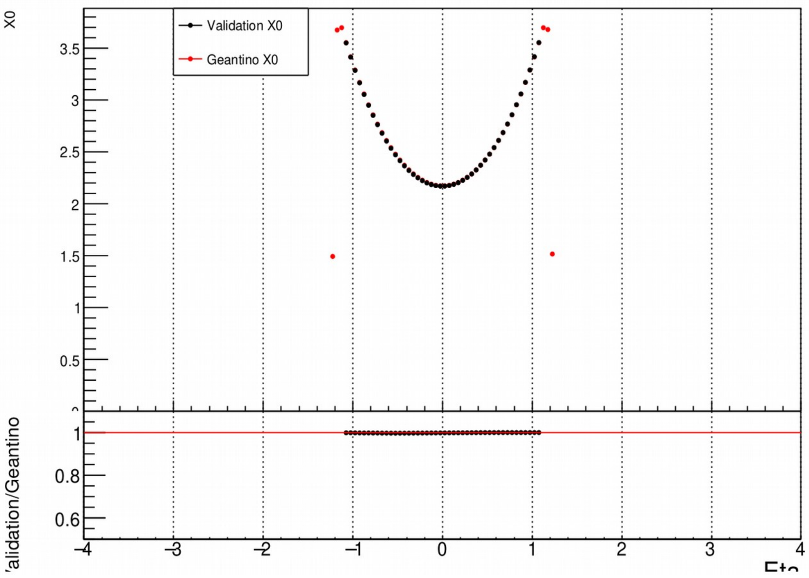
Surface mapping for e.g. Silicon:
- Mapping material to discrete binned surfaces
- Material is considered when surface is crossed

**X0 ratio Validation/Geantino vs η for ITk**



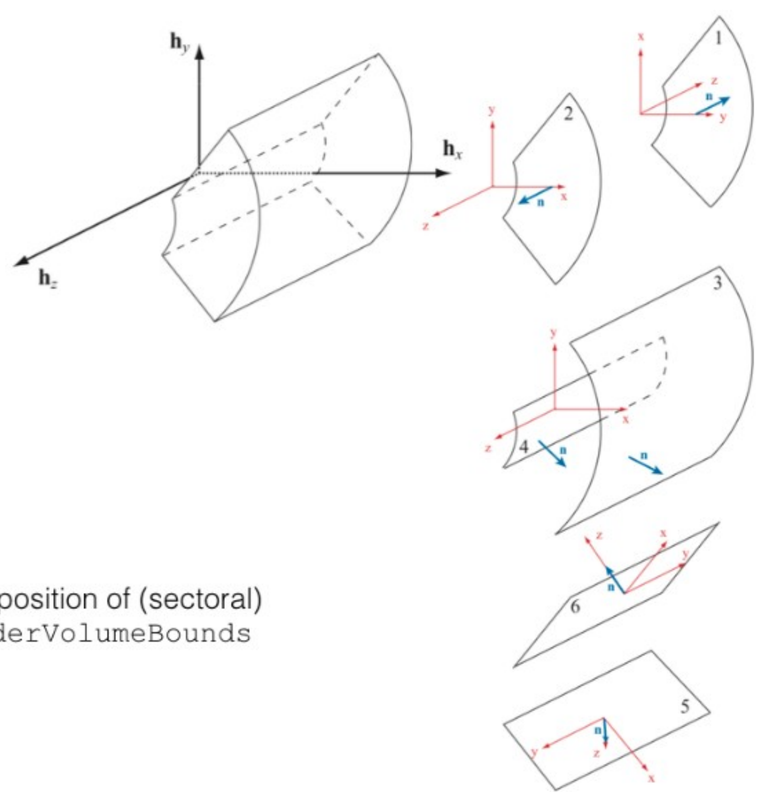

Volume mapping for e.g. Calorimeter:
- Mapping material to 3D volume grid points
- Material is considered at each propagation step

**X0 ratio Validation/Geantino vs η for a dummy Calorimeter**

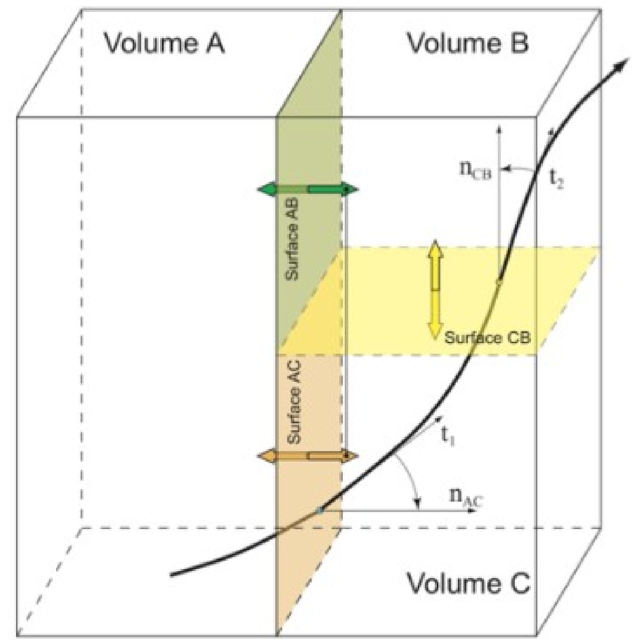


# Navigation

- Acts::TrackingGeometry is a collection of Acts::TrackingVolume fully connected via Acts::BoundarySurface
  - Acts::VolumeBounds classes defines the shape of volumes and create the corresponding boundary surfaces
- Boundary surfaces are the key component to navigate between volumes
  - The uniquely defined normal vector of the boundary surface helps define volumes on both sides

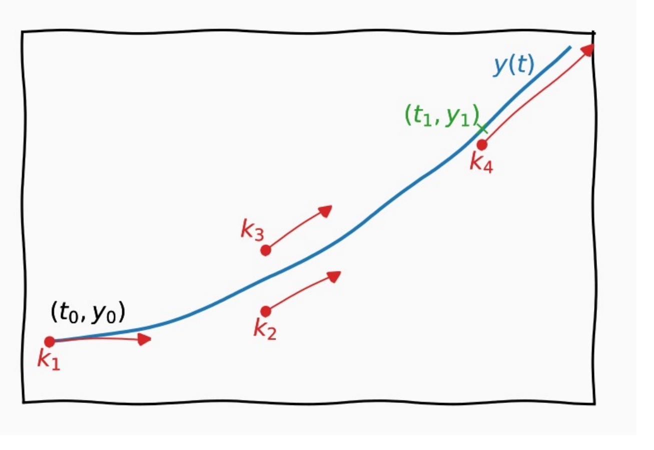



Decomposition of (sectoral) `CylinderVolumeBounds`




resulting navigation
through the boundary portals

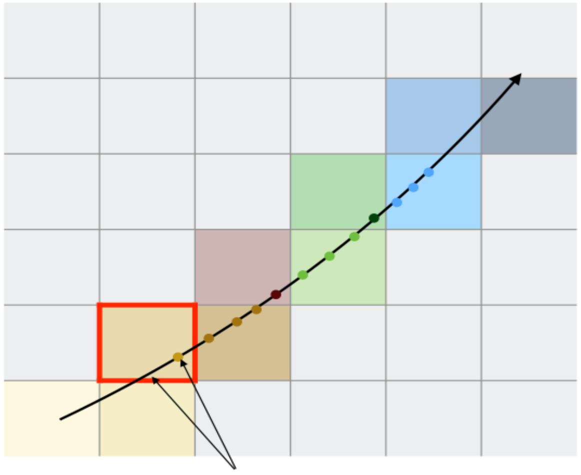
# Magnetic Field & Track Parameter Propagation



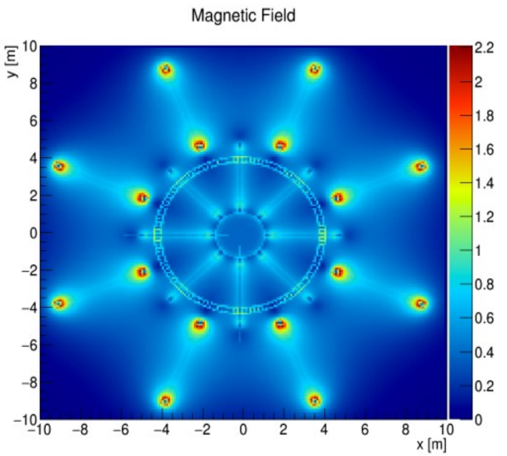
# Magnetic field

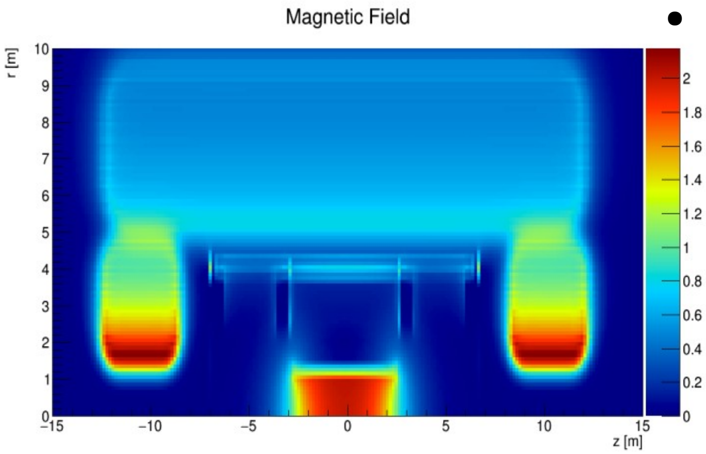
- Magnetic field interfaces:
  - Constant magnetic field
  - Interpolated magnetic field
    - Calculates an interpolated B field value from a grid of known field values
  - Analytical solenoid magnetic field
    - Calculates field vectors analytically for a solenoid field

**Interpolated magnetic field**




**ATLAS Magnetic field in ACTS**



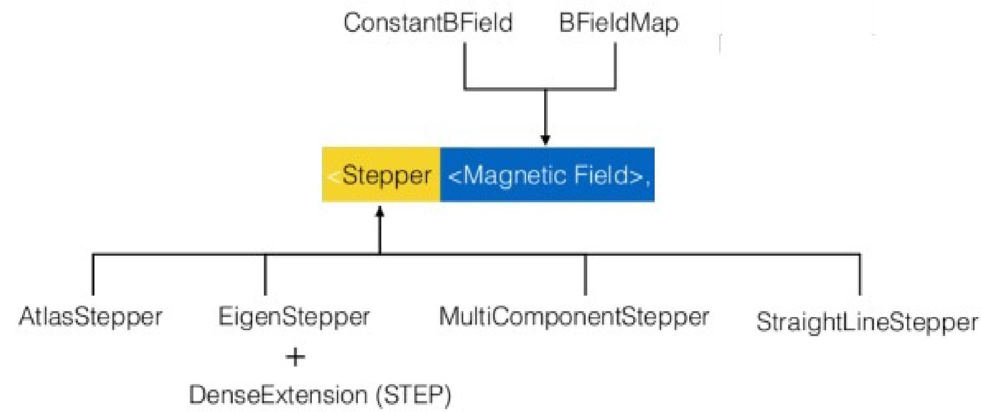




- Magnetic field access:
  - Cache of field value could make the access less expensive
  - To ensure thread-safety, the field cell is cached by client and passed between client and magnetic field service via client function argument

# Track parameter propagation

Integrating motion of particle transport in magnetic field

- Adaptive Runge-Kutta-Nyström method is implemented as the primary integration
  - ATLAS Stepper but rewritten using Eigen
- Dense Environment Extension for transport in dense volumes, e.g. calorimeter



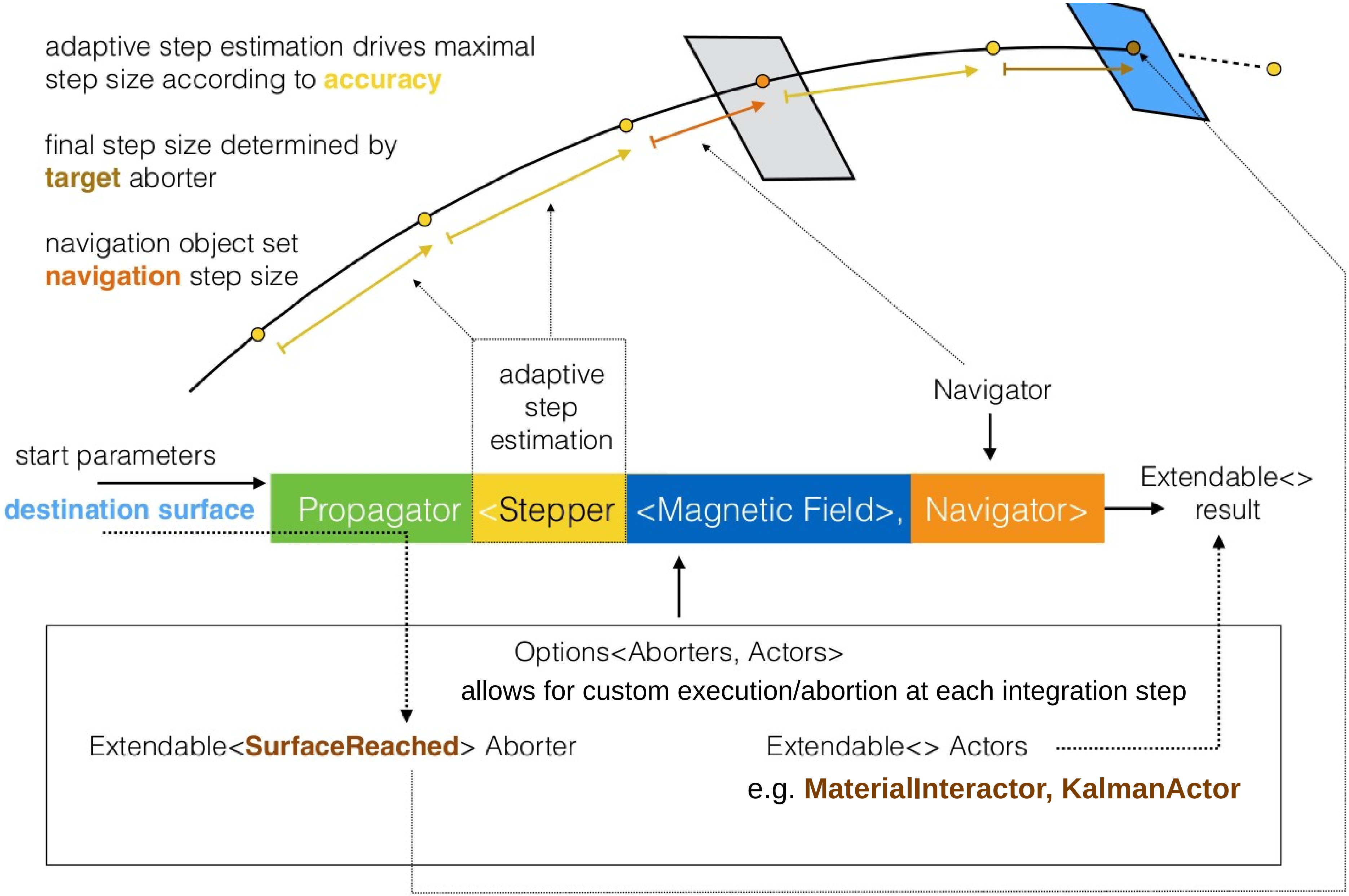

- Timing information is included in integration to allow for time measurement
  - No additional overhead
- WIP for full free parameters and covariance representation without binding to surface
  - Facilitates tracking for detector with many measurements, e.g. TPC, Drift Chamber

Concept of FreeMeasurementApproacher is tested

```
auto measurement = (*result.mt);

// Get the position & direction
auto position = stepper.position(state.stepping);
auto direction = stepper.direction(state.stepping);

// Let's create a hyperplane at the measurement point
using Plane = Eigen::Hyperplane<double, 3>;
using Line = Eigen::ParametrizedLine<double, 3>;

// Make the plane and intersect
auto plane = Plane(direction, measurement);
auto line = Line::Through(position, position + direction);
double path = line.intersection(plane);
if (std::abs(path) < s_onSurfaceTolerance) {
  result.approaches.push_back(position);
  ++result.mt;
  return;
}
```

# Propagator interface

Integrating particle transport & geometry navigation
Highly-templated design emphasizing on speed and customizability



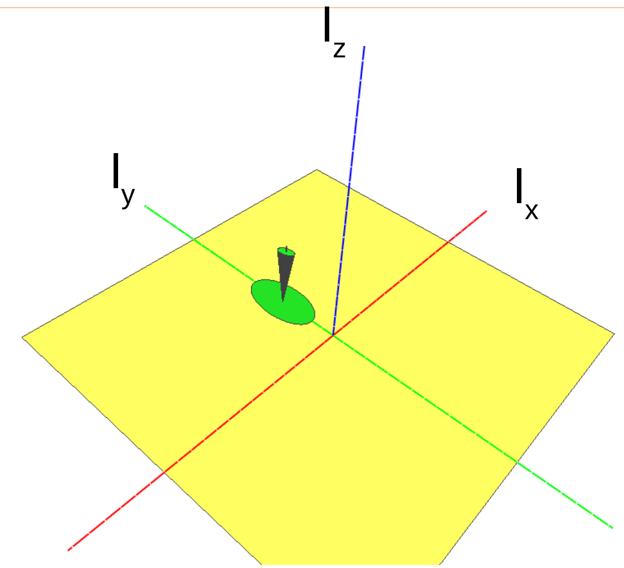
# Event Data Model

# Track Parameter EDM

Free (global) representation:

$G = (x, y, z, t, T_x, T_y, T_z, q/p)$

- $(x, y, z)$: global position
- $(T_x, T_y, T_z)$: momentum direction

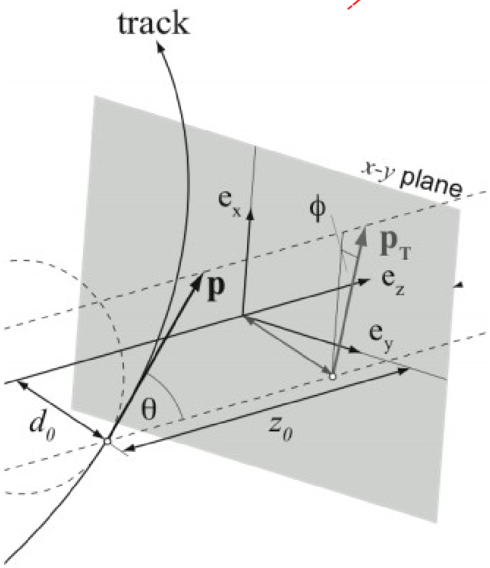



Transforms between them are handled by methods of surface and stepper engine

Local representation:

$L = (l_0, l_1, \phi, \theta, q/p, t)$

- $l_0$, $l_1$: Coordinates (could be non-Cartesian) in local frame:
  - Local surface frame: Acts::SingleBoundTrackParameters
  - Local frame moving along the track: Acts::SingleCurvilinearTrackParameters
- $p$, $\phi$, $\theta$: Momentum and direction
- $q$: Charge
- $t$: Per-track timing info




Meaning of $l_0$, $l_1$ for Bound parameters varies with surface type, e.g. for perigee track parameters at perigee surface $l_0 = d_0$, $l_1 = z_0$

# Measurement EDM

- Acts::Measurement is templated on source link, i.e. original detector measurement and sets of measured variables to support different detectors
  - Source link must satisfies defined source link concept
  - std::variant implemented Acts::FittableMeasurement is a wrapper of heterogeneous Acts::Measurement

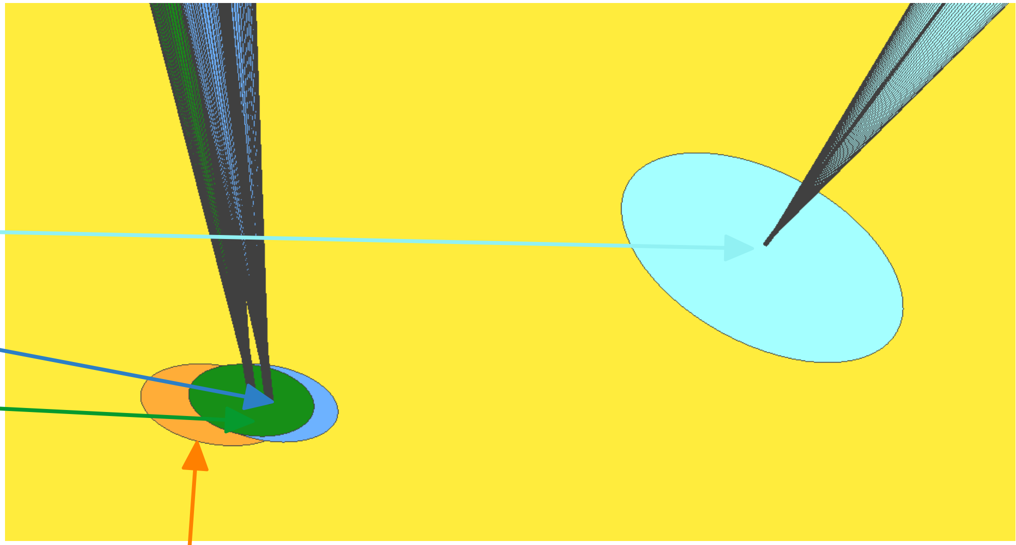
- Additional calibration of original detector measurement is allowed during fitting
  - Calibrator will turn the SourceLink into Acts::FittableMeasurement with the help of predicted track parameters

```
namespace concept {
  namespace detail_slc {
  template <typename T>
  using comparable_t = decltype(std::declval<T>() == std::declval<T>());

  template <typename T>
  using dereferenceable_t = decltype(*std::declval<T>());

  template <typename T>
  using surface_method_t = decltype(std::declval<T>().referenceSurface());

  template <typename T>
  struct SourceLinkConcept {
    constexpr static bool comparison_works =
        identical_to<bool, comparable_t, T>;
    static_assert(comparison_works,
                  "Source link does not implement equality operator");

    constexpr static bool surface_method_exists =
        converts_to<const Surface&, surface_method_t, T>;
    static_assert(
        surface_method_exists,
        "Source link does not have compliant referenceSurface method");

    constexpr static bool copyable = std::is_copy_constructible_v<T>;
    static_assert(copyable, "Source link must be copy constructible");

    constexpr static bool default_constructible =
        std::is_default_constructible_v<T>;
    static_assert(default_constructible,
                  "Source link must be default-constructible");

    constexpr static bool value =
        concept ::require<comparison_works, surface_method_exists, copyable,
                          default_constructible>;
  };
  }  // namespace detail_slc
}  // namespace concept
template <typename T>
constexpr bool SourceLinkConcept =
    concept ::detail_slc::SourceLinkConcept<T>::value;
```

# TrackState EDM

- Acts::TrackState is a wrapper of Track Parameters and Measurements
  - Based on concept of KalmanFilter

```
/// The parameter part
/// This is all the information that concerns the
/// the track parameterisation and the jacobian
/// It is enough to to run the track smoothing
struct {
  /// The predicted state
  std::optional<Parameters> predicted{std::nullopt};
  /// The filtered state
  std::optional<Parameters> filtered{std::nullopt};
  /// The smoothed state
  std::optional<Parameters> smoothed{std::nullopt};
  /// The transport jacobian matrix
  std::optional<Jacobian> jacobian{std::nullopt};
  /// The path length along the track - will help sorting
  double pathLength = 0.;
  /// chisquare
  double chi2 = 0;
} parameter;
```



```
/// @brief Nested measurement part
/// This is the uncalibrated and calibrated measurement
/// (in case the latter is different)
struct {
  /// The optional (uncalibrated) measurement
  std::optional<SourceLink> uncalibrated{std::nullopt};
  /// The optional calibrabed measurement
  std::optional<FittableMeasurement<SourceLink>> calibrated{std::nullopt};
} measurement;
```

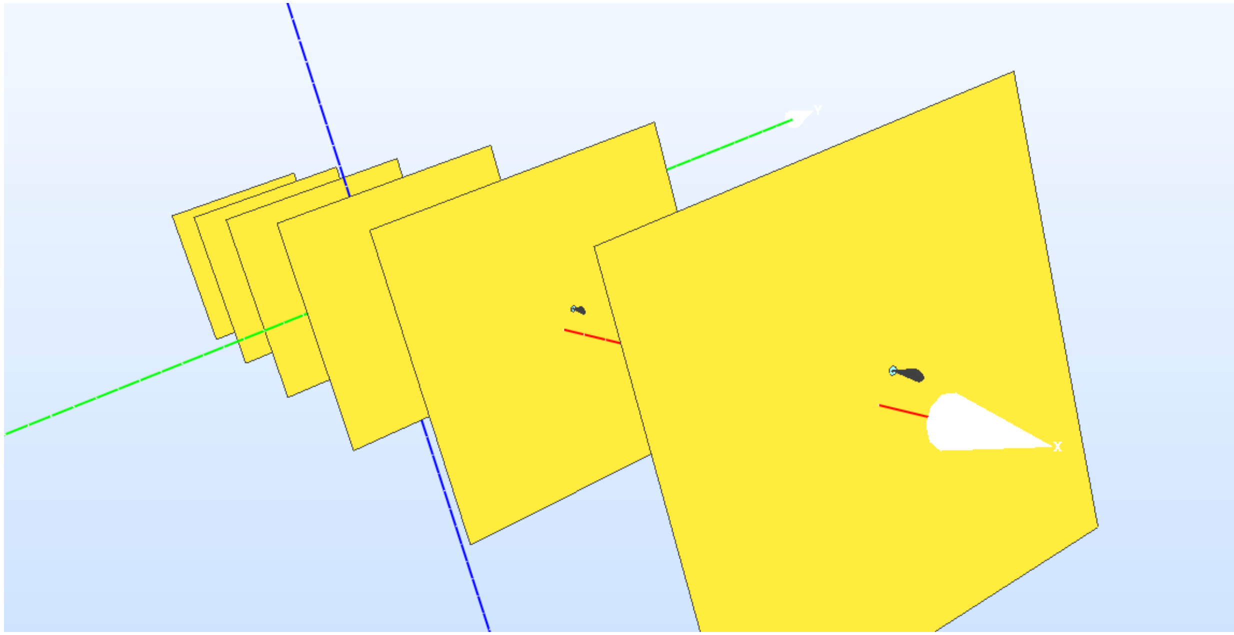
Via Calibrator during fitting

# Track EDM

- Eigen::Array based track EDM (Acts::MultiTrajectory), i.e. container of track states on trajectories
  - Provides read-write views into separate storage of parameter coefficients and covariance

```
using Coefficients = Eigen::Matrix<Scalar, Size, 1, Flags>;
using Covariance = Eigen::Matrix<Scalar, Size, Size, Flags>;
using CoefficientsMap = Eigen::Map<ConstIf<Coefficients, ReadOnlyMaps>>;
using CovarianceMap = Eigen::Map<ConstIf<Covariance, ReadOnlyMaps>>;
```

- Keeps track of storage index
  - Allows for branching of tracks (multi-trajectories case) via parent relationship
  - Avoids storage duplication for shared measurements and parameters



# Track/Vertex Fitting&Finding

# Track fitting with KalmanFilter

```
template <typename propagator_t, typename updater_t = VoidKalmanUpdater,
          typename smoother_t = VoidKalmanSmoother,
          typename outlier_finder_t = VoidOutlierFinder,
          typename calibrator_t = VoidMeasurementCalibrator,
          typename input_converter_t = VoidKalmanComponents,
          typename output_converter_t = VoidKalmanComponents>
class KalmanFitter {
```

- Acts::KalmanFitter (KF) is used as an Actor in Acts::Propagator
- Nested consideration of material effects at each filtering step
  - preUpdate → Kalman filtering → postUpdate
- Supports hole search and outlier rejection during the fitting
- Supports two different approaches for smoothing
  - Using 'smoothing-matrix' formalism based on Jacobians in forward filtering
  - Run an additional Kalman filtering in backward direction
- Extension for calculating global track parameters covariance
  - Fundamental ingredient for KF-based alignment approach
- Gaussian Sum Filter as non-gaussian extension is implemented ( yet to be finalized)

# KalmanFitter fit interface

- Fitting inputs:
  - Measurements (unsorted source links, e.g. clusters) on a known trajectory
  - Starting parameter (with large uncertainty)
  - User-defined KalmanFilter options

```
template <typename source_link_t, typename start_parameters_t,
          typename kalman_fitter_options_t,
          typename parameters_t = BoundParameters,
          typename result_t = Result<KalmanFitterResult<source_link_t>>>
auto fit(const std::vector<source_link_t>& sourcelinks,
         const start_parameters_t& sParameters,
         const kalman_fitter_options_t& kfOptions) const
    -> std::enable_if_t<!isDirectNavigator, result_t> {
```

- Fitting result:
  - A MultiTrajectory object with single track entry index
  - Fitted parameter at user-defined target surface

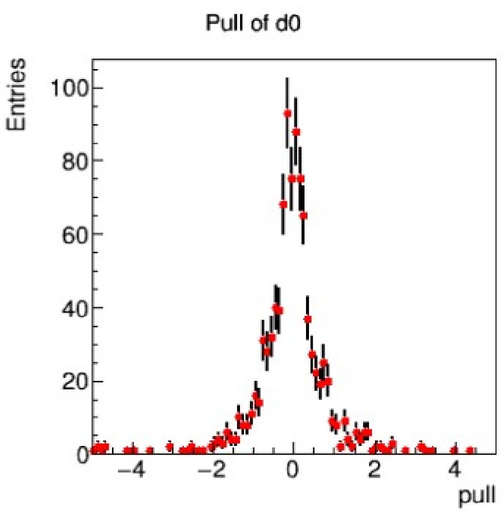
```
template <typename source_link_t>
struct KalmanFitterResult {
  // Fitted states that the actor has handled.
  MultiTrajectory<source_link_t> fittedStates;

  // This is the index of the 'tip' of the track stored in multitrajectory.
  // Since this KF only stores one trajectory, it is unambiguous.
  // SIZE_MAX is the start of a trajectory.
  size_t trackTip = SIZE_MAX;

  // The optional Parameters at the provided surface
  std::optional<BoundParameters> fittedParameters;

  // Counter for states with measurements
  size_t measurementStates = 0;

  // Counter for handled states
  size_t processedStates = 0;

  // Indicator if smoothing has been done.
  bool smoothed = false;
```

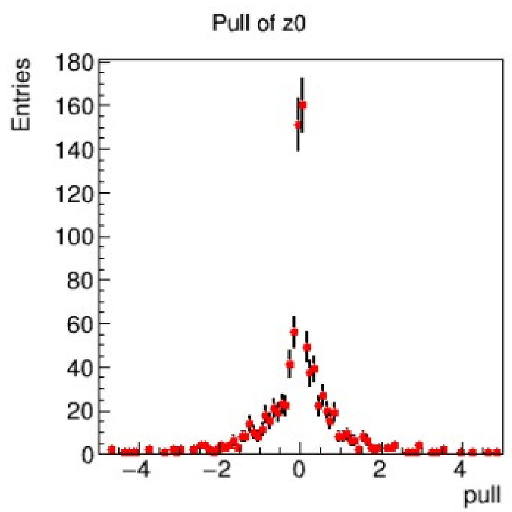
# KalmanFilter performance

TrackML detector, ATLAS B field

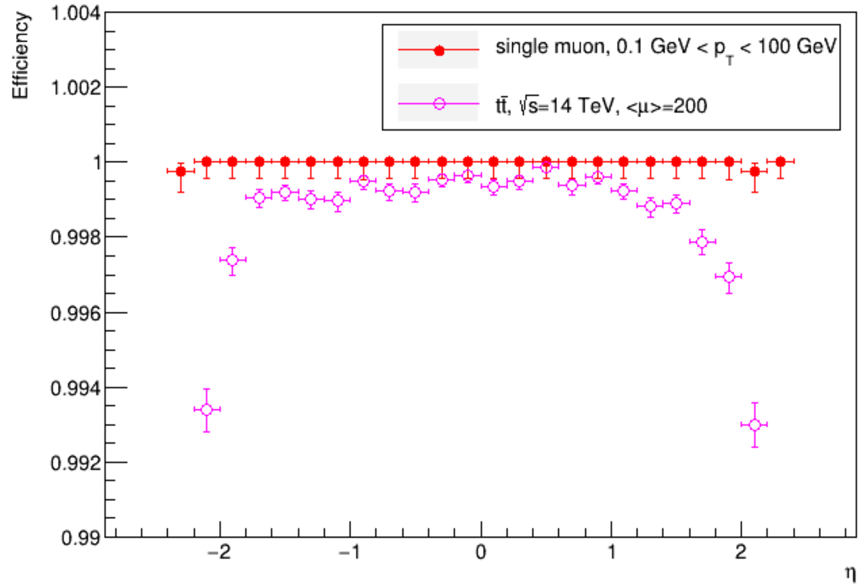
- Validated with $p_T$ down to 100 MeV
- 100% fitting efficiency
  - Defined as $\frac{N_{fit\ succeeds}}{N_{truth}}$

**Perigee track parameter resolution validation**

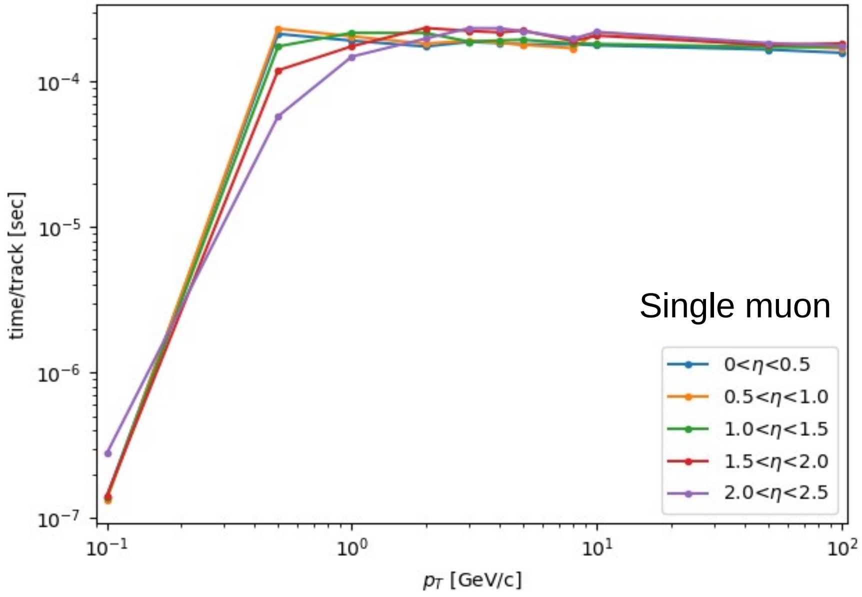



**Fitting efficiency vs. η**



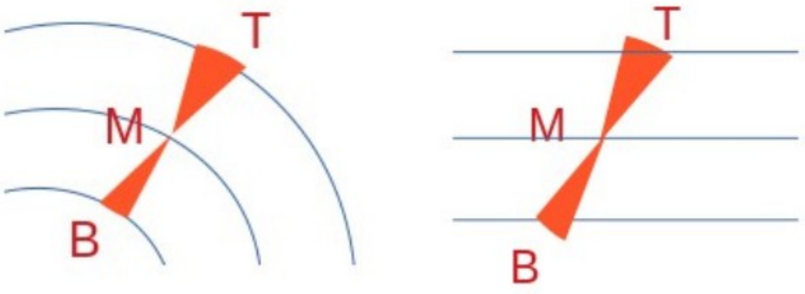

**Single track fitting time vs. pT**



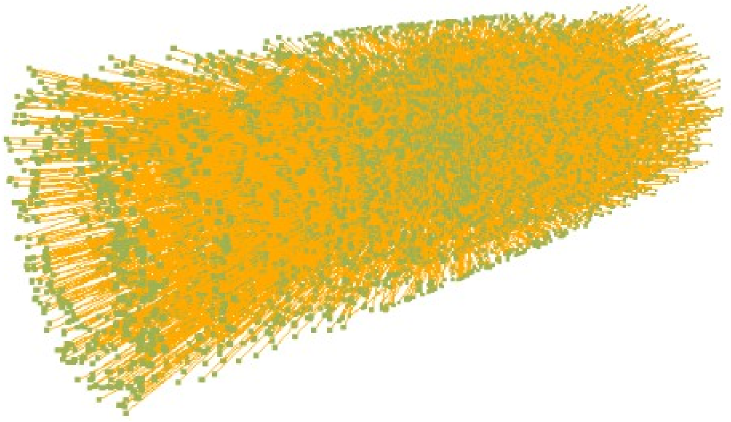
# Track finding

Local approach:  track seeding + track following

- A combinatorial seed finder for track seeding
  - Fine-grained parallelism (independent search of Top and Bottom SpacePoint for Middle SP)




- The Combinatorial Kalman Filter (CKF) for track following
  - Simultaneous tracking fitting and finding (no refitting is needed)
  - Allows track branching if more than one compatible measurement found on a surface
    - Supports user-defined measurement search and branching strategy
      - Default selection criteria is based on Kalman filtering $\chi^2$
  - Allows stopping of bad quality branch



CKF results for ttbar events with $\mu$ = 200 (~7k particles, ~80k hits)

# CKF track finding interface

- Track finding inputs:
  - All Measurements in one events
  - Starting parameter (from seeding)
  - User-defined CKF options

```
template <typename source_link_container_t, typename start_parameters_t,
          typename comb_kalman_filter_options_t,
          typename parameters_t = BoundParameters>
Result<CombinatorialKalmanFilterResult<
    typename source_link_container_t::value_type>>
findTracks(const source_link_container_t& sourcelinks,
           const start_parameters_t& sParameters,
           const comb_kalman_filter_options_t& tfOptions) const {
```

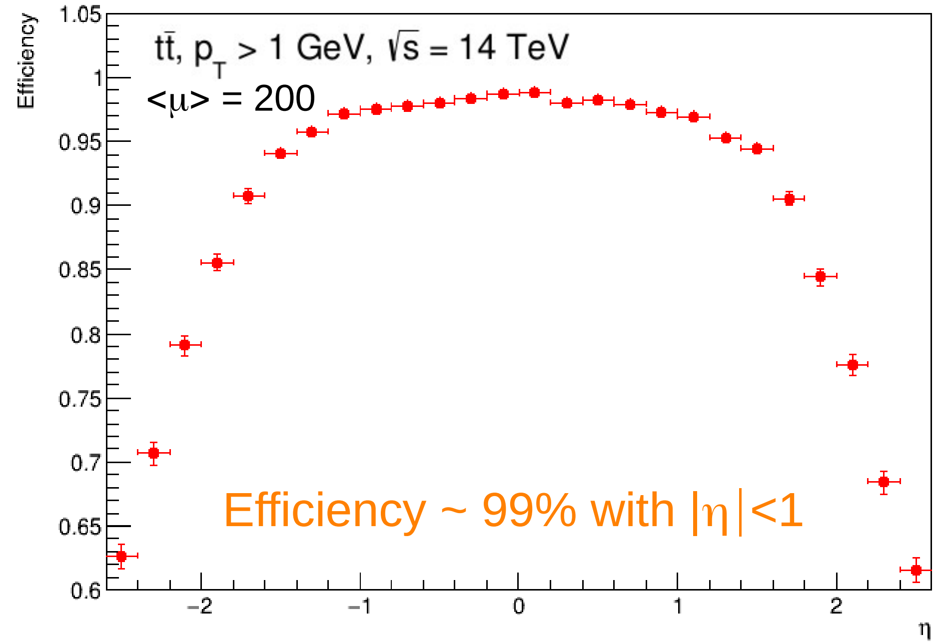
- Track finding result:
  - A MultiTrajectory object with multiple track entry indices
  - Fitted parameters at user-defined target surface

```
template <typename source_link_t>
struct CombinatorialKalmanFilterResult {
  // Fitted states that the actor has handled.
  MultiTrajectory<source_link_t> fittedStates;

  // The indices of the 'tip' of the tracks stored in multitrajectory.
  std::vector<size_t> trackTips;

  // The Parameters at the provided surface for separate tracks
  std::unordered_map<size_t, BoundParameters> fittedParameters;

  // The indices of the 'tip' of the unfinished tracks
  std::vector<std::pair<size_t, CombinatorialKalmanFilterTipState>> activeTips;
```

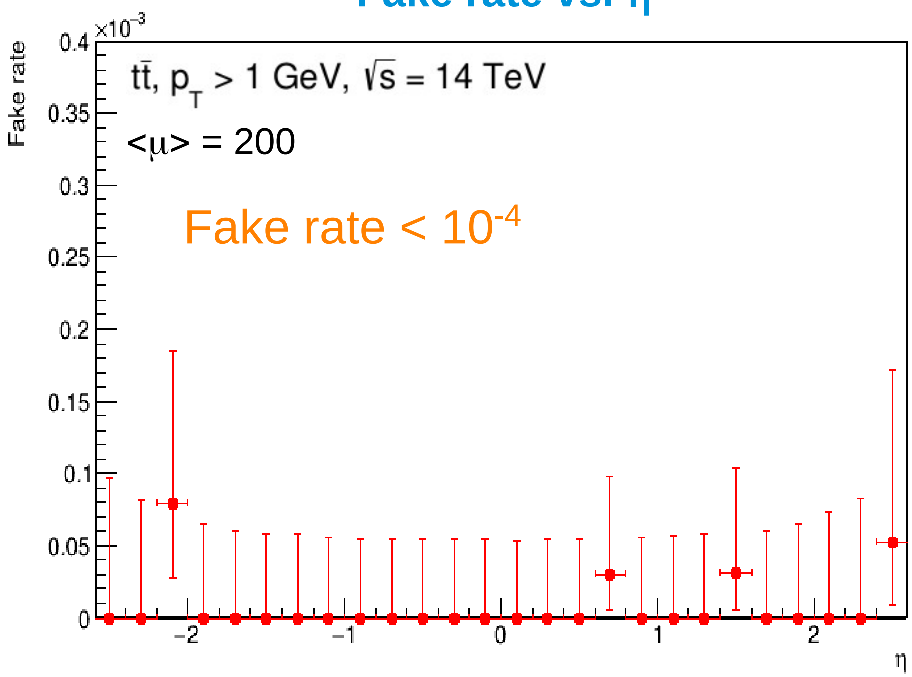
# CKF performance

- All hits from truth particles with $p_T$>100 MeV are considered
  - Track finding efficiency: $\frac{N_{reco}(selected, matched)}{N_{truth}(selected)}$
  - Fake rate: $\frac{N_{reco}(selected, unmatched)}{N_{reco}(selected)}$
  - Duplication rate: $\frac{N_{reco}(selected, matched, duplicated)}{N_{reco}(selected, matched)}$
    - ➔ Reco-truth matching: $\frac{N_{hits}(Majority)}{N_{hits}(Total)} > 0.5$
    - ➔ Simple track selection: $n_{Hits}$>=9

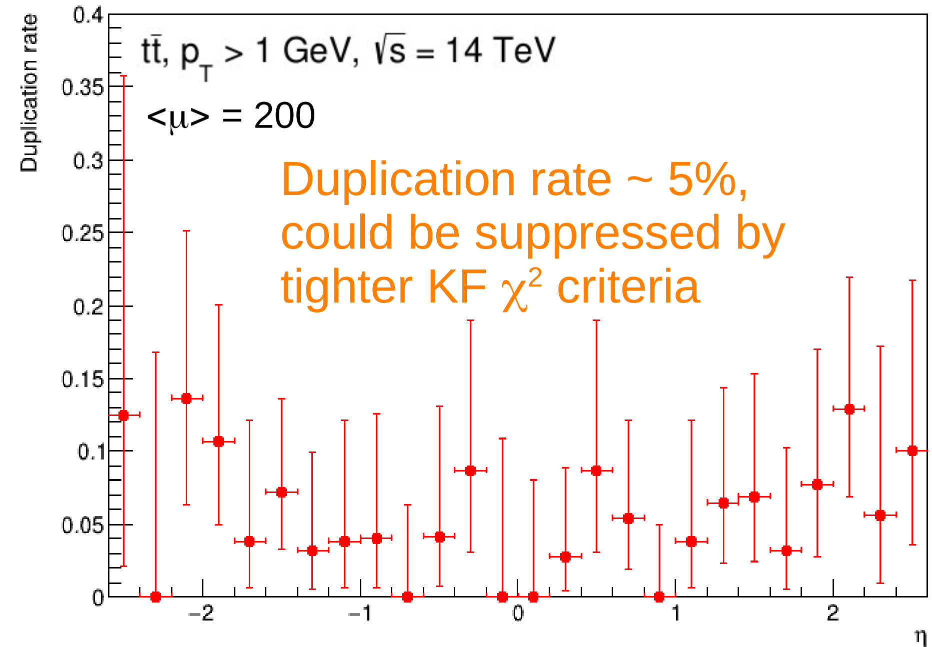
## Efficiency vs. η




## Fake rate vs. η




## Duplication rate vs. η



# Vertex finding/fitting

- Various vertexing tools have been transcribed from ATLAS vertexing algorithms with performance well validated against ATLAS SW
  - ML for Vertexing is being explored (see Bastian's CTD slides)
- Two approaches:
  - Iterative fitting-after-finding
    - **Iterative Vertex Finder (IVF)** (used at ATLAS Run-2)
  - Finding-through-fitting
    - **Adaptive Multi-Vertex Finder (AMVF)** (to be used at ATLAS Run-3)

**Portable tools used in IVF and AMVF**

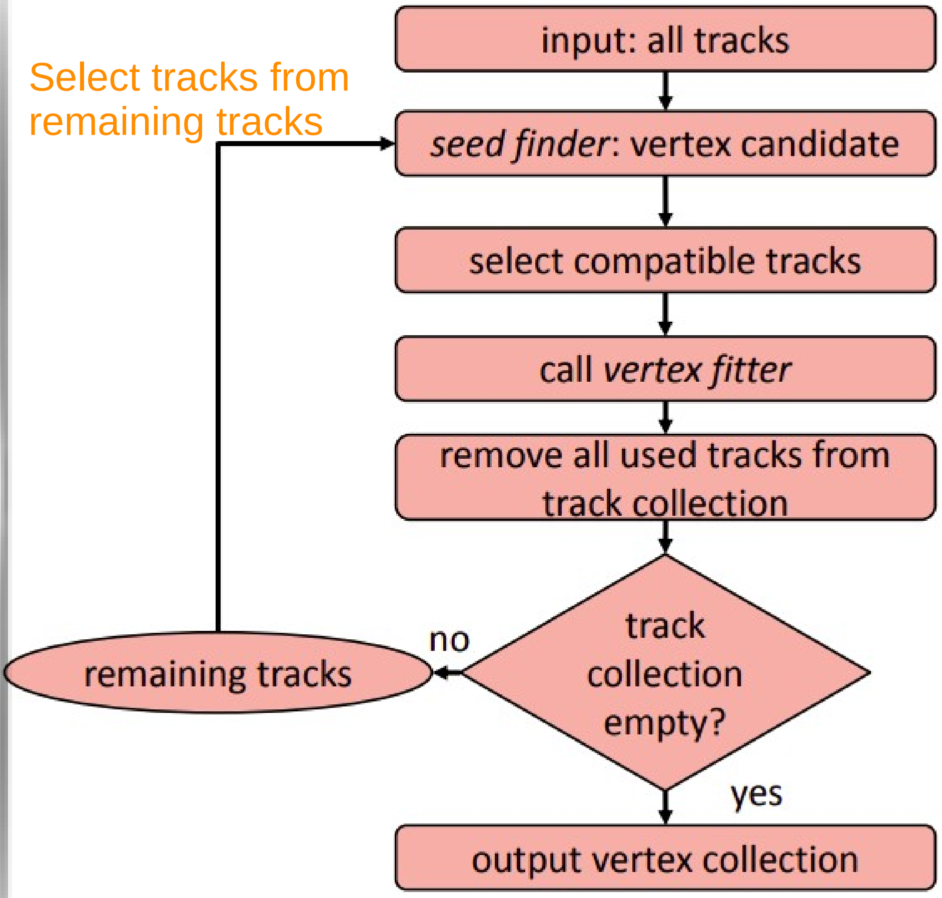
- Seed finder:
  - Z-Scan Seed Finder
  - Gaussian Track Density Vertex Finder
  - Gaussian Grid Track Density Vertex Finder

- Vertex fitter
  - Full-Billoir Vertex Fitter
  - Adaptive Multi-Vertex Fitter

- Utilities: track selection, track linearizer, impact point estimator, deterministic annealing tool etc.

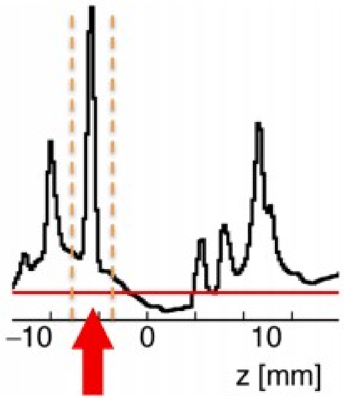
# Iterative Vertex Finder (IVF)

(see B.Schlag's slides)




**ZScanSeedFinder:**

- find mode value of all $z_0$ values
- vertex candidate at position ($z_0$, 0, 0)

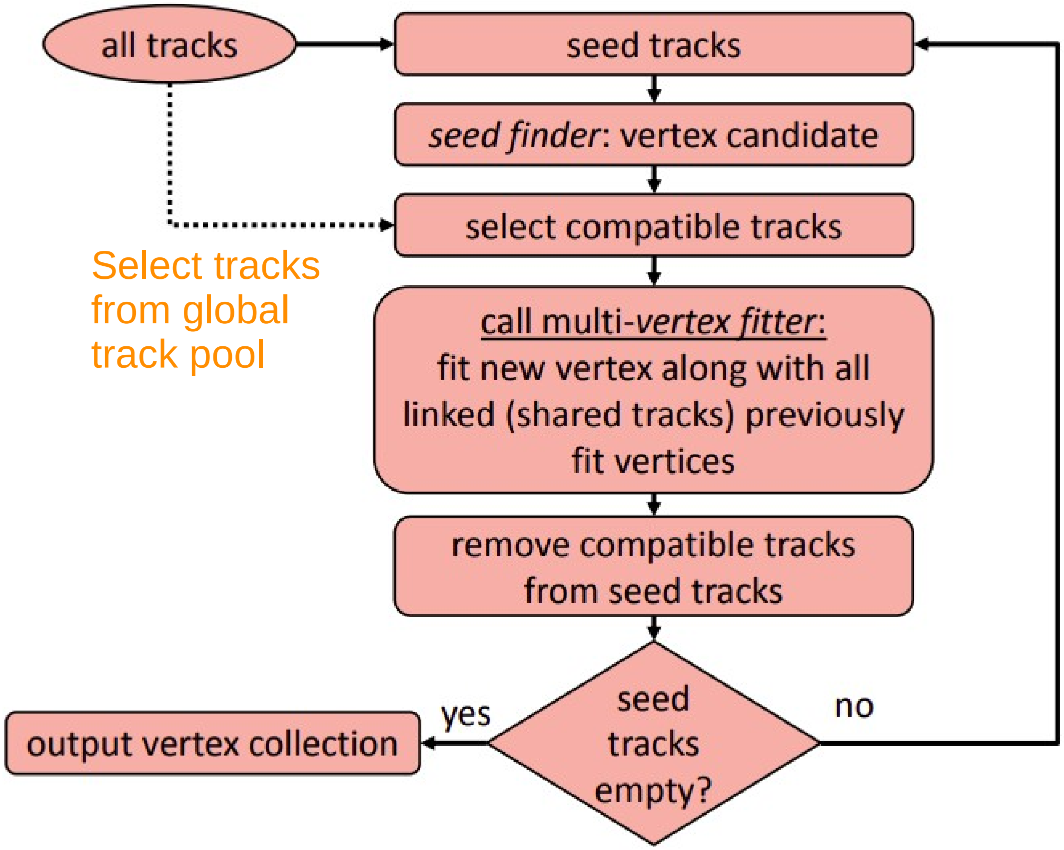



**Iterative fitting-after-finding approach:**

- iteratively find vertex and fit with compatible tracks
- single track always associated to at most one vertex
- tracks removed from pool after fitting

# Adaptive Multi-Vertex Finder (AMVF)

(see B.Schlag's slides)




**Gaussian Track Density Seed Finder:**

- model each track as 2-dim Gaussian distribution in $d_0$-$z_0$-plane around ($d_0$, $z_0$)
- find z value of highest track density along z-axis

**Adaptive Multi-Vertex Fit:** (strong binding with AMVF)

- weighted adaptive Kalman filter using deterministic annealing scheme
- subject to beamspot and seed constraint
- Simultaneous refit of all vertices connected through a chain of vertices and tracks, with weights:

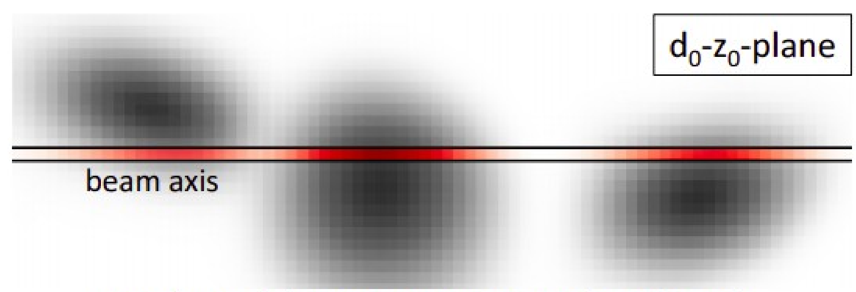
$$\omega_i(\chi_i^2, T) = \frac{e^{-\frac{1}{2}\chi_i^2/T}}{\sum_j e^{-\frac{1}{2}\chi_j^2/T} + e^{-\frac{1}{2}\chi_0^2/T}}$$

tracks can have weights to multiple vertices

→ *Finding-through-fitting approach*

**Gaussian Grid Track Density Vertex Seed Finder:**

- Model track as 2-dim Gaussian density grid in $d_0$-$z_0$-plane
- Interested only in density distribution along beam axis:
  → calculate only track contribution along beam axis (red)
- Superimpose all tracks and find maximum along beam axis




Example: Track density representations of 3 single tracks

# Vertex finding interface

- Highly configurable design of vertex finder
  - Vertex fitter
  - Seed finder
  - ImpactPoint estimator
  - Linearizer
  - ...
- Vertex finding inputs
  - A collection of tracks
  - Vertexing Options
- Vertex finding outputs
  - A collection of found vertices

```
/// @struct Config Configuration struct
struct Config {
  /// @brief Config constructor
  ///
  /// @param fitter The vertex fitter
  /// @param sfinder The seed finder
  /// @param ipEst ImpactPointEstimator
  /// @param lin Track linearizer
  Config(vfitter_t fitter, const sfinder_t& sfinder,
         const ImpactPointEstimator<InputTrack_t, Propagator_t>& ipEst,
         const Linearizer_t& lin)
      : vertexFitter(std::move(fitter)),
        seedFinder(sfinder),
        ipEstimator(ipEst),
        linearizer(lin) {}

  // Vertex fitter
  vfitter_t vertexFitter;

  // Vertex seed finder
  sfinder_t seedFinder;

  // ImpactPointEstimator
  ImpactPointEstimator<InputTrack_t, Propagator_t> ipEstimator;

  // Track linearizer
  Linearizer_t linearizer;

  // Use a beam spot constraint, vertexConstraint in VertexingOptions
  // has to be set in this case
  bool useBeamSpotConstraint = true;
```

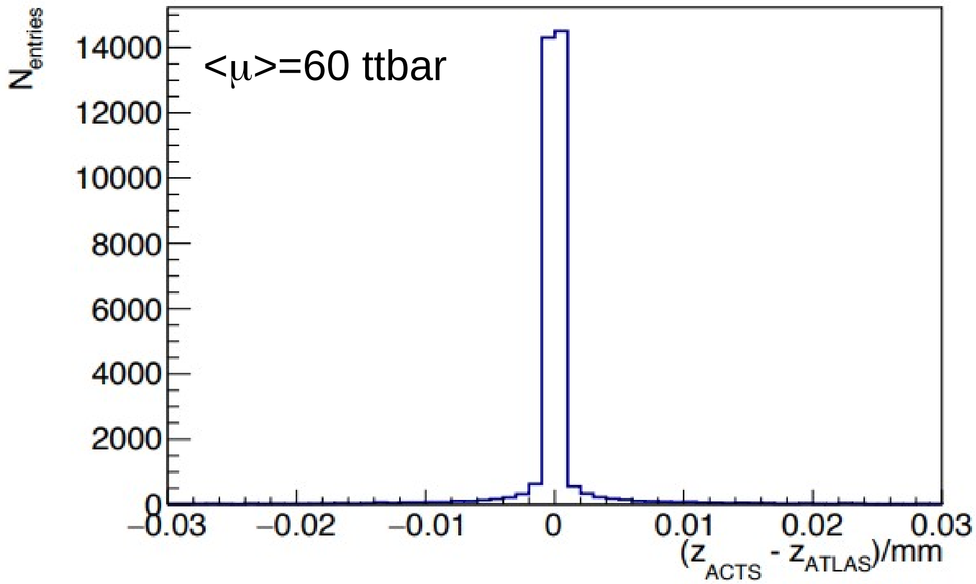
```
/// @brief Function that performs the adaptive
/// multi-vertex finding
///
/// @param allTracks Input track collection
/// @param vertexingOptions Vertexing options
/// @param state State for fulfilling interfaces
///
/// @return Vector of all reconstructed vertices
Result<std::vector<Vertex<InputTrack_t>>> find(
    const std::vector<const InputTrack_t*>& allTracks,
    const VertexingOptions<InputTrack_t>& vertexingOptions,
    State& state) const;
```

# Vertexing performance

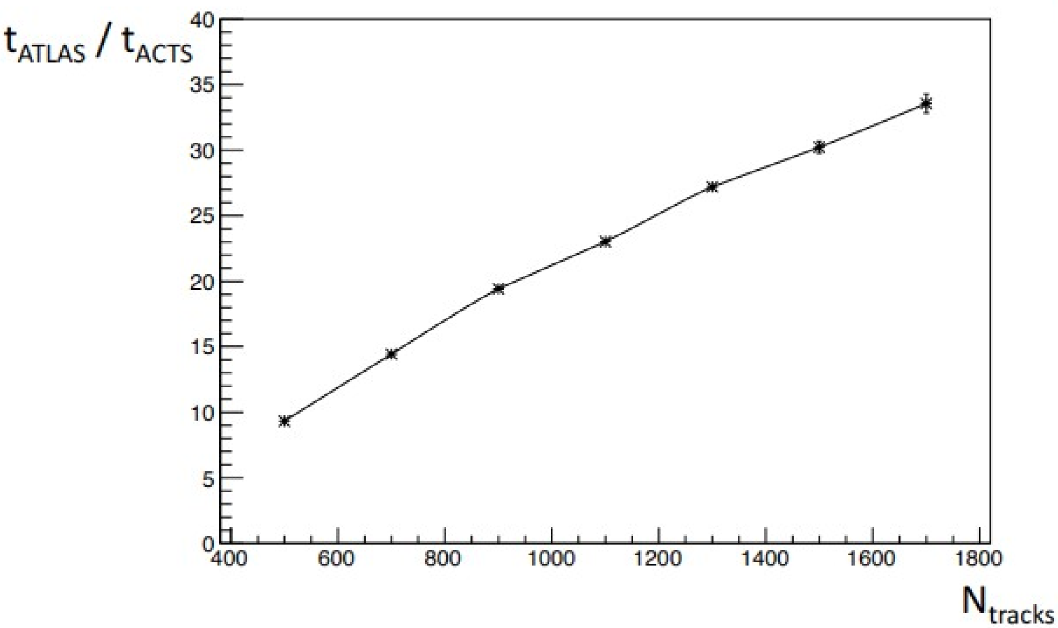
- Vertex position resolution agrees with ATLAS results on mircometer level
- Significant speed-up w.r.t. to ATLAS algorithm

(see B.Schlag's slides)

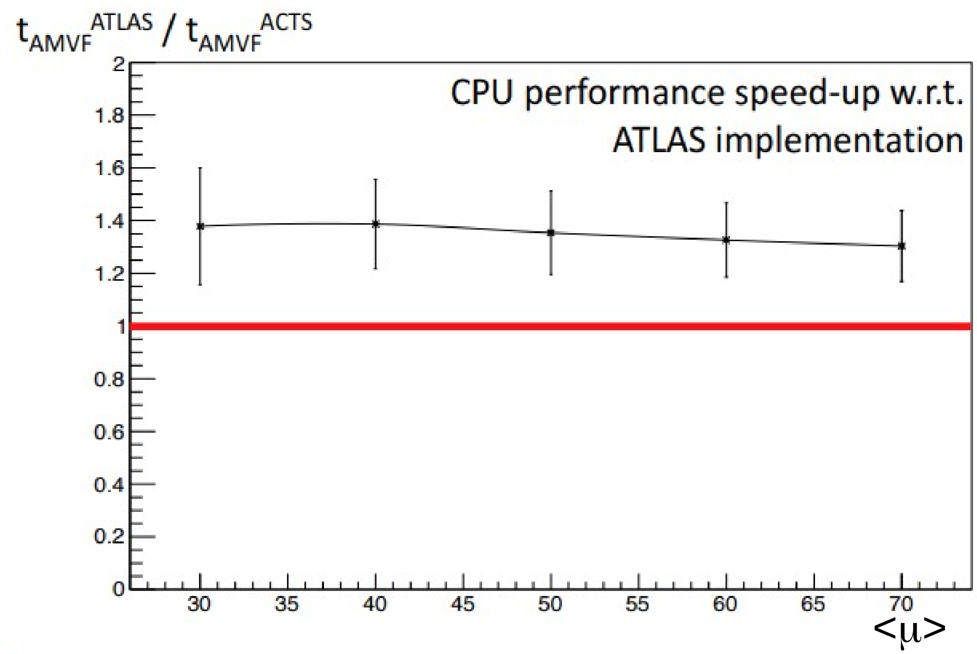
**AMVF Vertex z positon resolution**




**Gaussian Grid Track Density Vertex Finder timing performance**

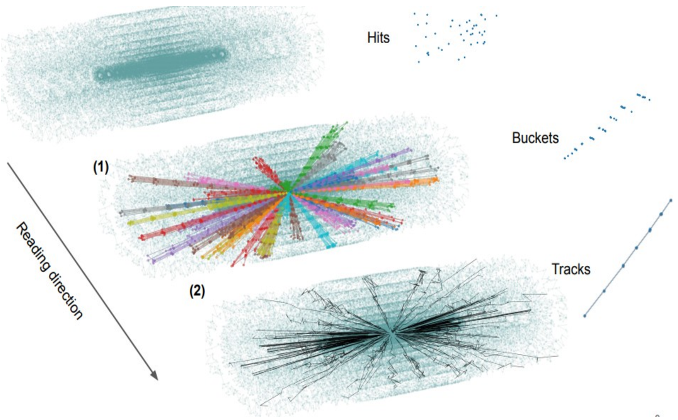



**AMVF timing performance**



# R&D and Applications

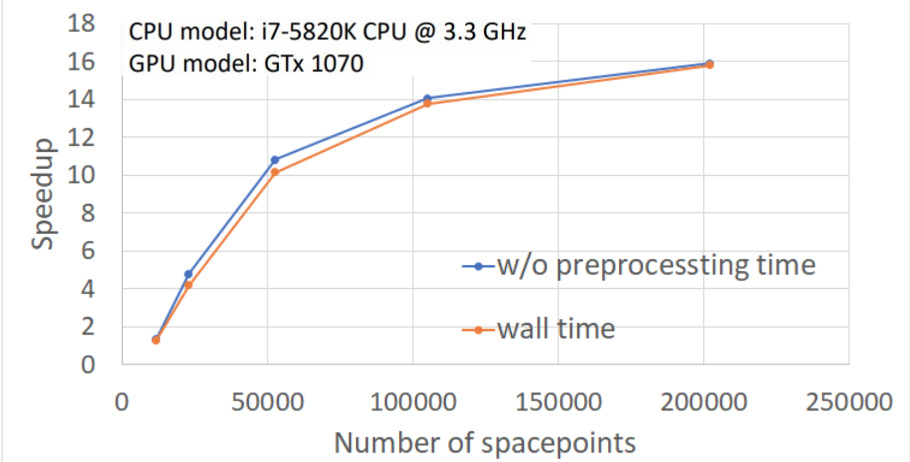
# R&D



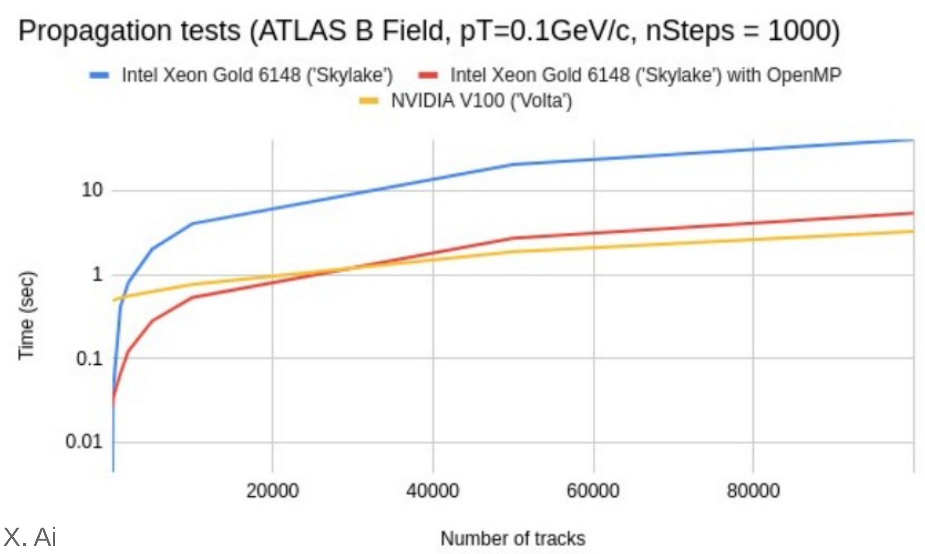

- Provides support for new tracking techniques R&D
  - Similarity Hashing and learning
  - Hep.TrkX & Exa.TrkX project
- Parallelism and acceleration facilitated by hardware architecture
  - Intra-event parallelism (track-level parallel fitting)
  - GPUs-accelerated tracking (e.g. seed finding, propagation, navigation)

see G. Mania's slides

ACTS seedfinder with CUDA (@NVIDIA GTX 1070)




B. Yeo, C. Legett et. al




X. Ai

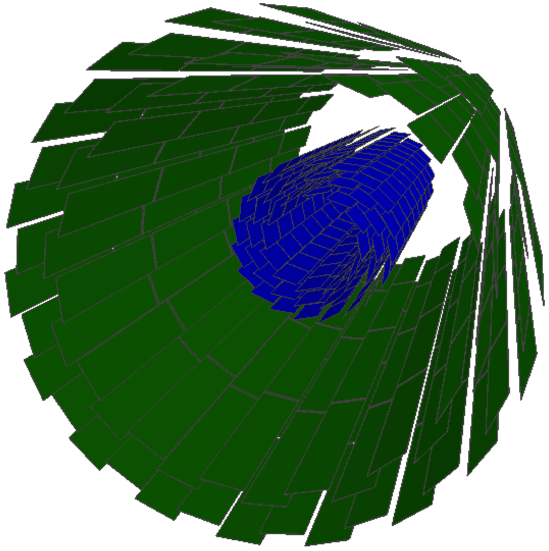
# Application to (experiment) detectors

- Detector geometry implemented:
  - **sPHENIX Silicon + TPC** (see Joe's talk later)
  - TrackML Detector
  - Open Data Detector
  - ATLAS ID+Calo, ATLAS ITK
  - FASER Silicon
  - CEPC Silicon+TPC
  - Belle-II Silicon
  - PANDA
  - FCC-hh

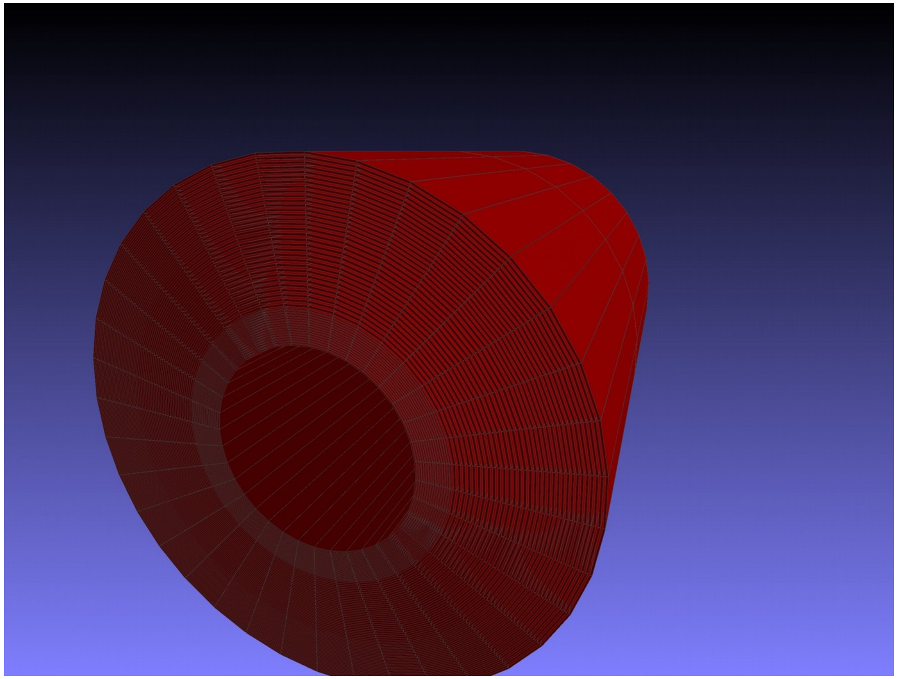
see slides here for more experiments experience

- On-going/planned implementation:
  - ATLAS Muon System
  - Belle-II Drift Chamber

sPHENIX Silicon



sPHENIX TPC

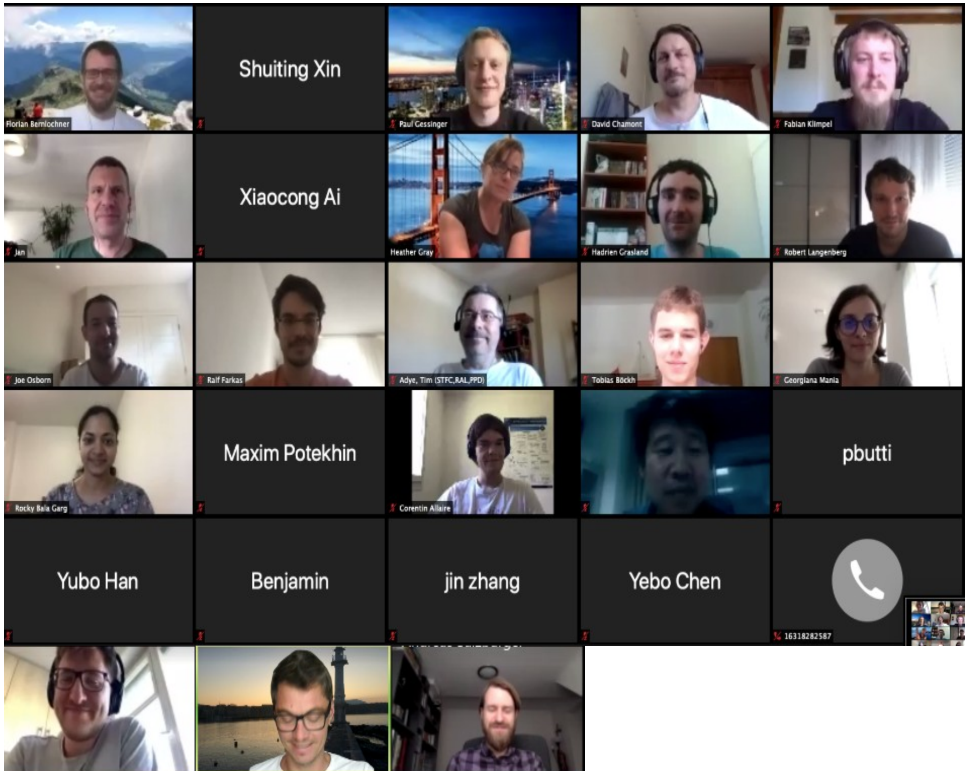


# Summary

- ACTS has matured a lot as a tracking toolkit over the past year
  - Consolidation of tracking infrastructure, e.g. geometry, propagator, EDM
  - Implementation of new tracking features, e.g. KalmanFilter, CKF, IVF, AMVF
- ACTS is an active R&D platform for new tracking techniques (ML) and hardware architectures
- Future focus will be facilitating application and optimization of the tracking toolkit
  - e.g. integration into ATLAS SW for ATLAS Run3

- Growing interest in experiment application& contribution
  - ATLAS ID+Calo, ATLAS ITK, FASER, CEPC, sPHENIX, BELLE-II, PANDA

**Up to 54 participants at the latest ACTS tracking workshop 2020**



# backup

# ACTS members/developers

Andreas Salzburger @asalzbur · acts
Given access 4 years ago

Noemi Calace @ncalace · acts
Given access 4 years ago

Valentin Volkl @vavolkl · acts
Given access 4 years ago

Benedikt Hegner @hegner
Given access 4 years ago

David Rousseau @droussea
Given access 4 years ago

Markus Elsing @elsing
Given access 4 years ago

Paolo Calafiura @calaf
Given access 4 years ago

ATS Jenkins @atsjenkins
Given access 4 years ago

Moritz Kiehn @msmk · acts
Given access 3 years ago

Hadrien Benjamin Grasland @hgraslar · acts
Given access 3 years ago

Stewart Martin-Haugh @smh · acts
Given access 3 years ago

Karolos Potamianos @karolos · acts
Given access 3 years ago

Edward Moyse @emoyse · acts
Given access 3 years ago

Nicholas Styles @nstyles · acts
Given access 3 years ago

Dmitry Emeliyanov @demelian · acts
Given access 3 years ago

Sarka Todorova @nova · acts
Given access 3 years ago

Wolfgang Liebig @liebig · acts
Given access 3 years ago

Shaun Roe @sroe · acts
Given access 3 years ago

Vincenzo Innocente @innocent · acts
Given access 3 years ago

Frank-Dieter Gaede @fgaede · acts
Given access 3 years ago

David Chamont @chamont · acts
Given access 2 years ago

Tobias Golling @golling · acts
Given access 2 years ago

Robert Johannes Langenberg @rlangenl
Given access 2 years ago

Felice Pantaleo @fpantale · acts
Given access 2 years ago

Marco Rovere @rovere · acts
Given access 2 years ago

Paul Gessinger-Befurt @pagessin
Given access 2 years ago

Julia Hrdinka @jhrdinka Blocked
Given access 2 years ago

Fabian Klimpel @fklimpel
Given access 2 years ago

John Smith @jrsmith Blocked · acts
Given access 1 year ago

Shih-Chieh Hsu @schsu · acts
Given access 1 year ago

Heather Gray @hgray · acts
Given access 1 year ago

Lauren Alexandra Tompkins @tompkins · acts
Given access 1 year ago

Jin Zhang @jinz · acts
Given access 1 year ago

Xiaocong Ai @xai 2FA · acts
Given access 1 year ago

Simone Pagan Griso @spagan · acts
Given access 1 year ago

Gang Zhang @gang · acts
Given access 1 year ago

Tim Adye @adye · acts
Given access 10 months ago

Ke Li @keli · acts
Given access 8 months ago

Jessica Leveque @leveque · acts
Given access 6 months ago

Corentin Allaire @corentin · acts
Given access 5 months ago

Georgiana Mania @gmania · acts
Given access 4 months ago

Laurent Roger Igor Basara @lbasara · acts
Given access 4 months ago

Stanislava Sevova @ssevova
Given access 4 months ago

Florian Urs Bernlochner @fbernloc · acts
Given access 4 months ago

Peter Alan Steinberg @steinber · acts
Given access 3 months ago

Vincent Pascuzzi @vpascuzz
Given access 2 months ago

Bastian Schlag @bschlag
Given access 2 months ago

Charles Leggett @leggett · acts
Given access 1 month ago

Tomohiro Yamazaki @toyamaza · acts
Given access 1 month ago

Ralf Farkas @rafarkas · acts
Given access 1 week ago

50 ACTS Members (~1 year ago: 27)

**Experiments:**

**ATLAS**

**LHCb**

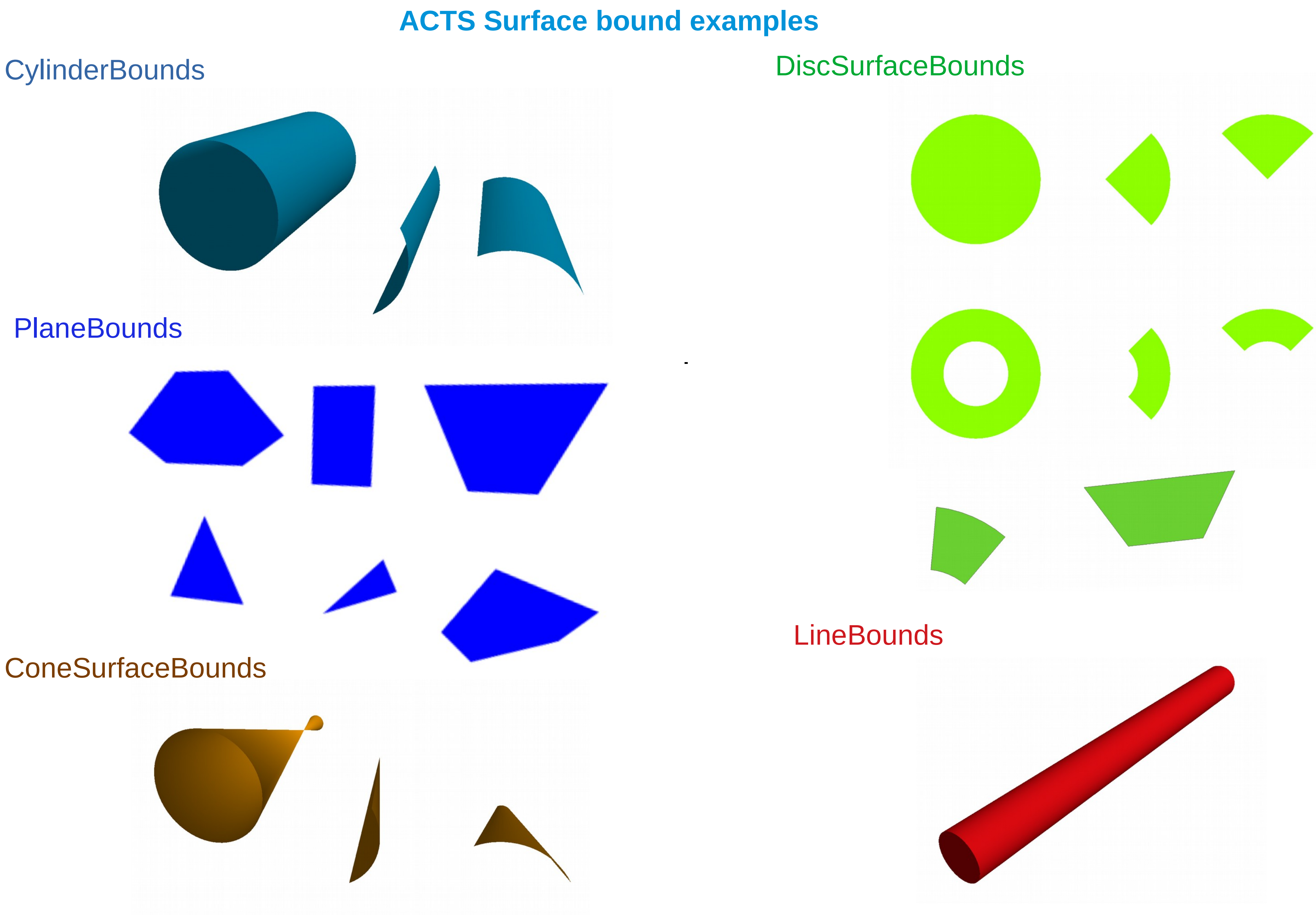
**CEPC**

**FASER**

**Belle-II**

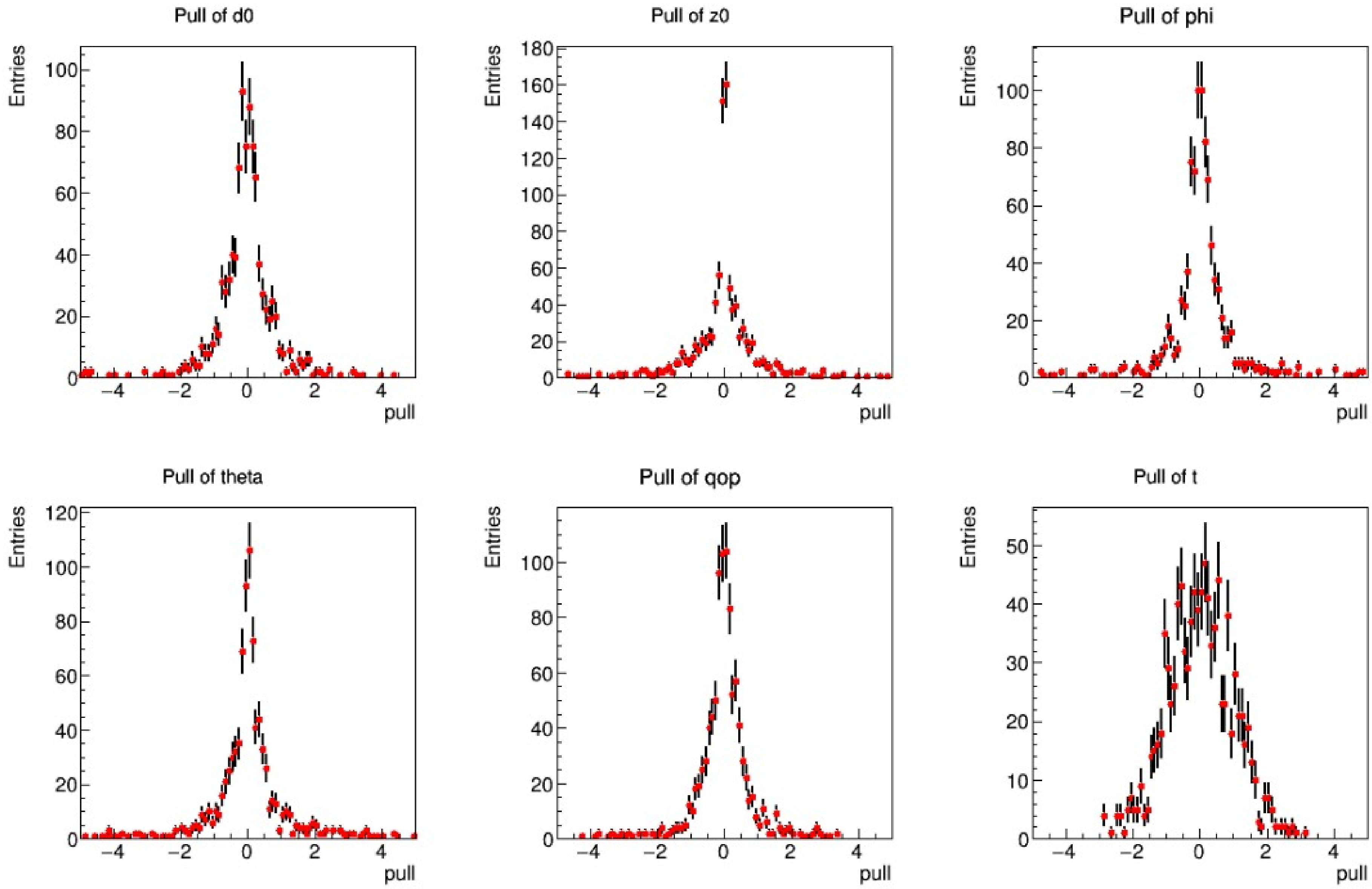
# The SurfaceBounds class

- The surface shape is described by Acts::SurfaceBounds



# KF fitting resolution

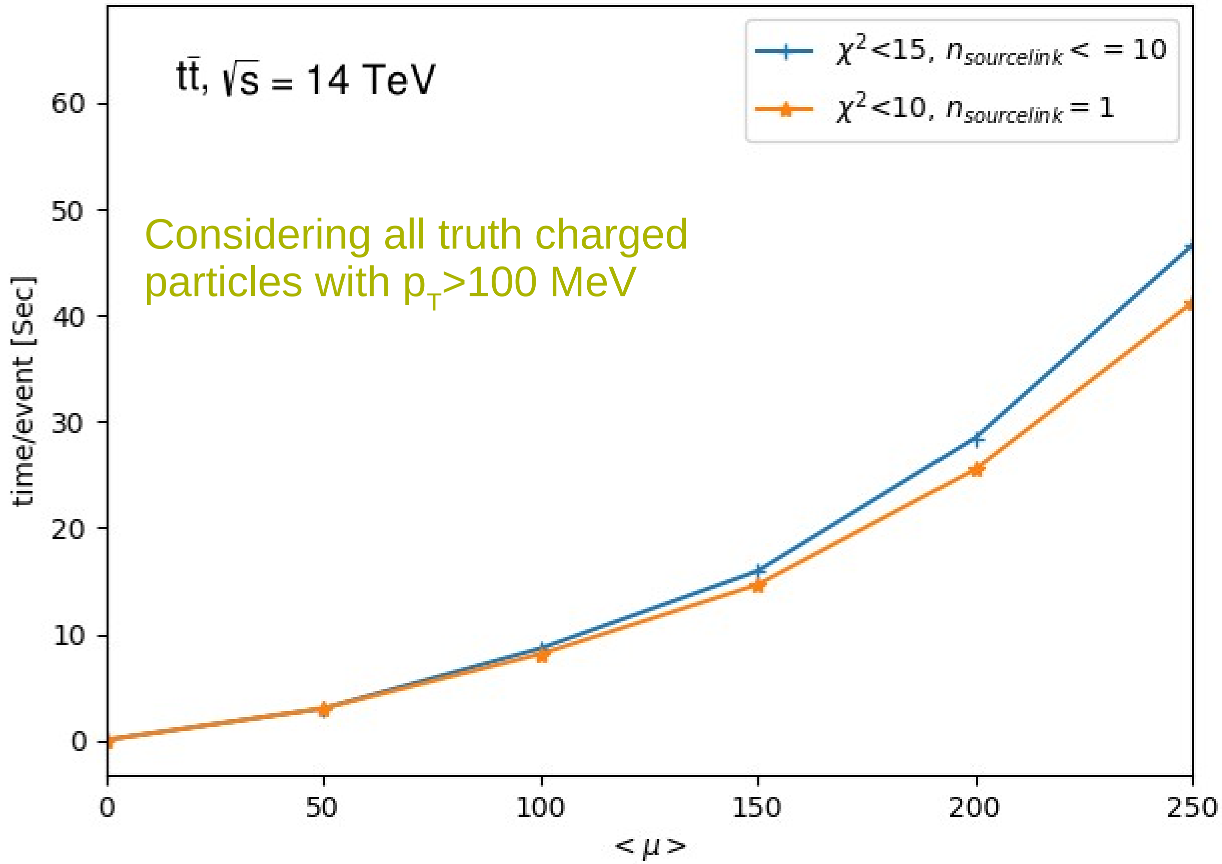
- Single muon, $0.1<p_T<100$ GeV, $|\eta|<2.4$
- TrackML detector, ATLAS B field



# CKF timing test

**CKF time/event vs. <μ>**

TrackML detector, ATLAS B fileld




Each filtering step needs to loop over all the source links on the surface for the source link selection, Could be speed-up by fast source link selection

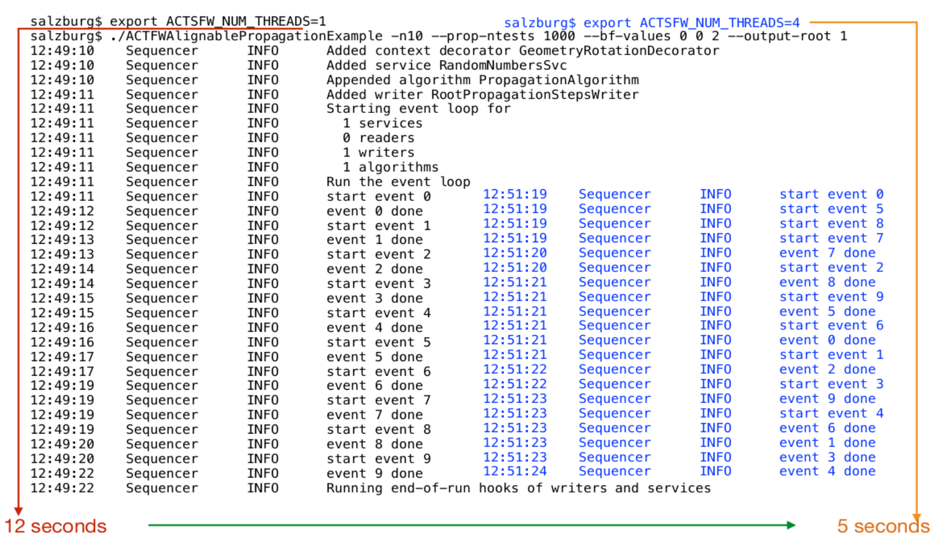
# Contextual alignment and calibration

- An AlgorithmContext object is used to support on-the-fly event-dependent changes of alignment/ calibration/magnetic field

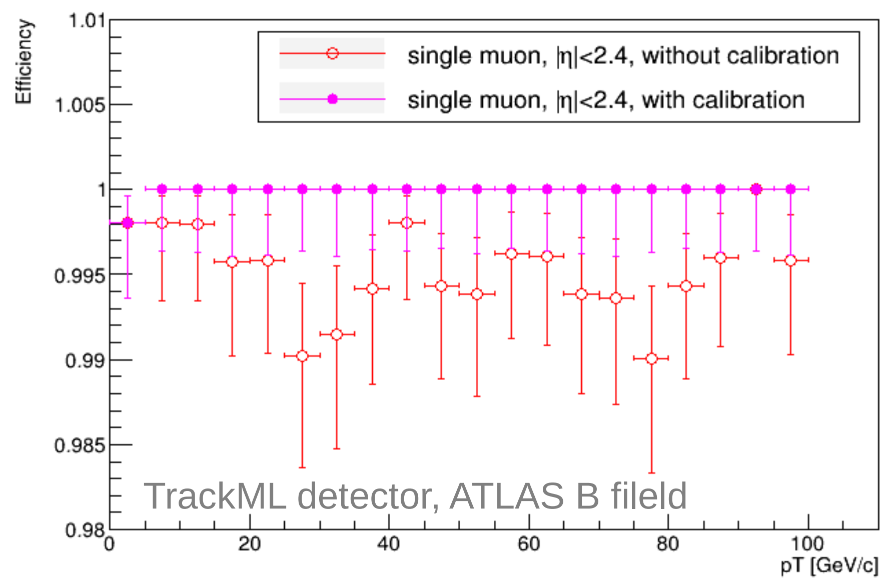
```
size_t                  algorithmNumber;  ///< Unique algorithm identifier
size_t                  eventNumber;      ///< Unique event identifier
WhiteBoard&             eventStore;       ///< Per-event data store
Acts::GeometryContext geoContext;         ///< Per-event geometry context
Acts::MagneticFieldContext
                           magFieldContext;  ///< Per-event magnetic Field context
Acts::CalibrationContext calibContext;    ///< Per-event calbriation context
```

- Concept of contextual alignment and calibration has been validated

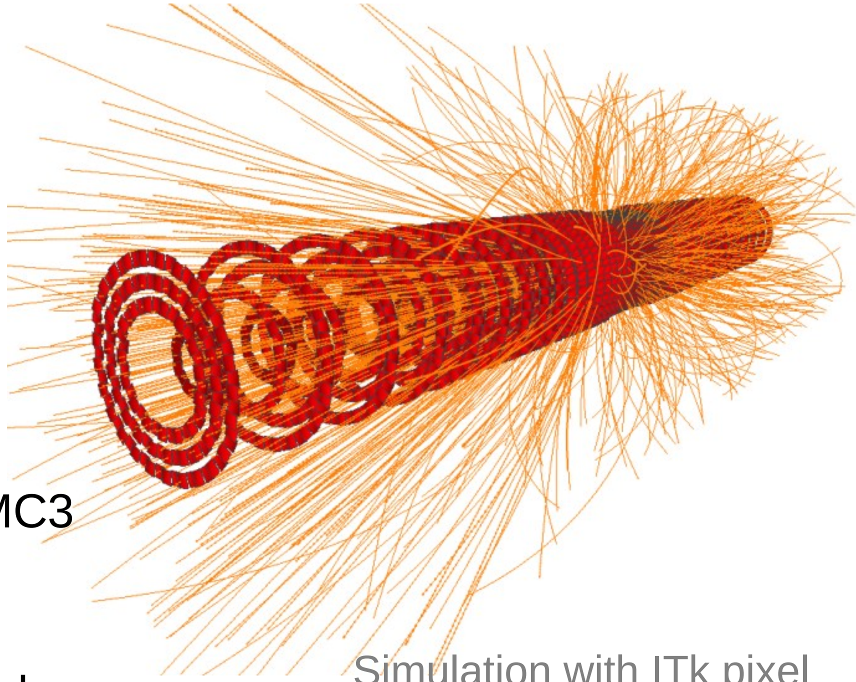
**Propagation tests with contextual alignment**
(Different alignment every single event, $n_{threads}$ = 4)




**Track fitting test with contextual calibration**
(Different calibration every 10 events, $n_{threads}$=8)



# Fast Simulation



Simulation with ITk pixel

- Event Data Model
  - Concrete particle and hit type
  - Flat, sorted data container for particle and hit
- Event generator
  - Particle Gun and interface to Pythia8 and HepMC3
- Detector material effects modeling
  - Energy loss and multiple scattering are validated
  - Hadronic interaction is currently reparameterised
  - Foreseen use of Geant4 for particle decay
  - Photon Conversion and positron annihilation are missing
- Detector response emulation (i.e. digitization)
  - Including pseudo-realistic clustering model (without clustering merging yet)
- Work-in-progress to use Json-based geometry/segmentation/material information at fast simulation chain

# The **detector**

Defined a Phase-2 like detector

- full silicon detector with realistic resolution, material budget, magnetic field
- composed as Pixel, short strip, long strip
- restricted to size of tracking volume to $|\eta| < 3$