

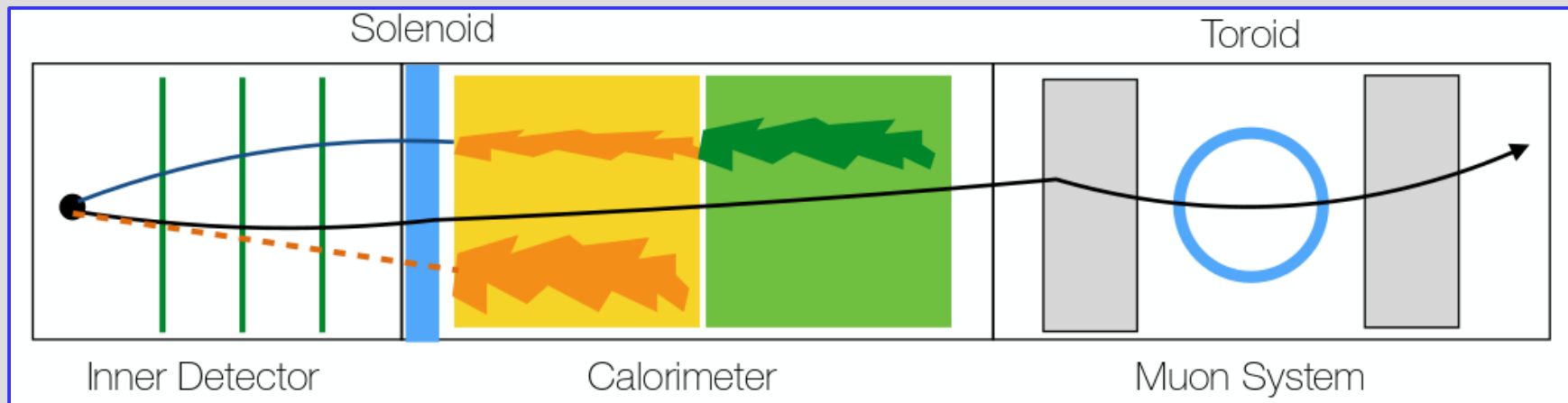
# Updates on the ATS project

Christian Gumpert on behalf of  
the development team









# Recap


- **First Common Track Reconstruction Software Forum**
- Andi gave a nice **talk** summarizing why ATLAS tracking software is a good starting point



- two precision tracking detectors with:
  - very different magnetic field setups
  - very different detector technologies
  - very different dimensions

# Current ATLAS Tracking SW ...

- solves the common tracking problem ... 
- ... in a real-world environment 
- is heavily field-tested 
- delivers excellent performance 
- relies on well-established concepts 
- is not thread-safe 

- 
- *decouple ATLAS tracking code from ATLAS specifics*
  - *review code: keep good things, improve bad things*
  - *make it thread-safe*

# Goals of “A Tracking Software”

- **suite of tools for track reconstruction as external library**
- double built strategy:
  - versus Gaudi(Hive) using cmake
  - versus Gaudi-Athena using cmt
- minimal external dependencies:
  - Eigen as matrix library
  - boost library

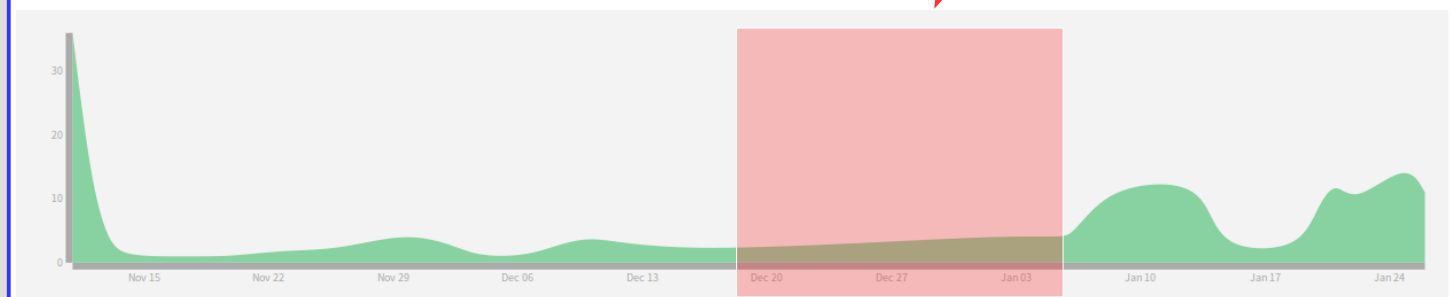
Commit statistics for **master** Nov 11 - Jan 26

- **177** commits during **76** days
- Average **2.3** commits per day
- Contributed by **5** authors

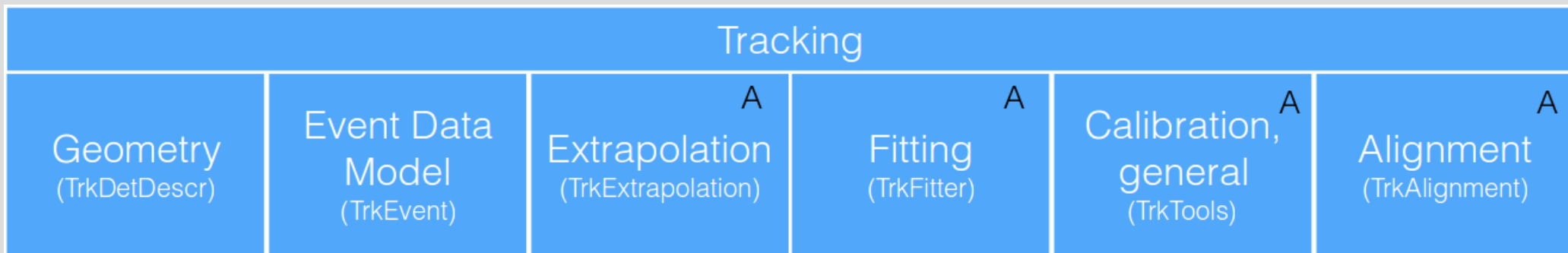
Christmas break

November 11 2015 - January 26 2016

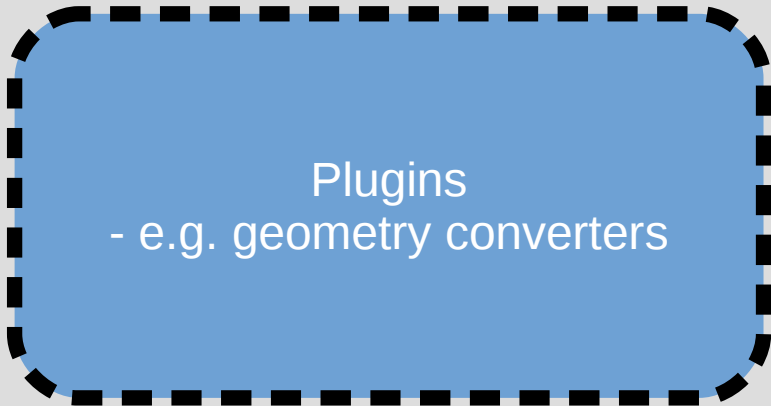
Commits to master, excluding merge commits. Limited by 6,000 commits



# Package structure



under discussion



Name	Last Update	Last Commit
..		7db7ee1e - Merge branch 'extrapolation' into 'master'
TrkDetDescr	about 24 hours ago	Merge remote branch 'origin/master' into extrapolation
TrkEvent	a day ago	be consistent in returning internals by value
TrkExtrapolation	about 24 hours ago	merge extrapolation/master
TrkFitter/TrkKalmanFitter	14 days ago	committing extrapolation changes
TrkMagneticField	2 days ago	add TrkMagneticField package

# DD4HEP-to-TrackingGeometry

- Julia has a written a converter from a DD4HEP detector description into a `ATS::TrackingGeometry` (in the context of the FCC software)
- plan is to also provide converters for other common geometry models (TGeo, GDML, ...)

- tracking geometry in memory, just didn't get a nice visualisation in time



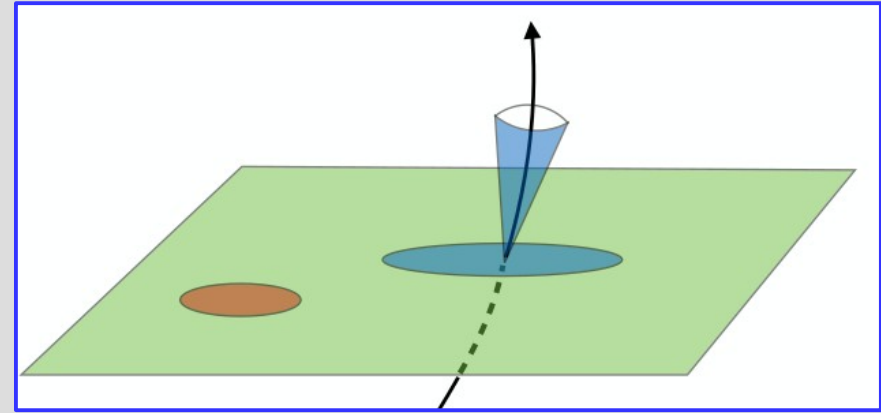
# Extrapolation

- extrapolation of track parameters through tracking geometry is implemented and fully functional
- using Runge-Kutta method for magnetic field integration
- thread-safe by making services stateless (apart from global configuration)
- caching important for performance  
=> pass cache objects as function arguments

```
182     /** Templated RungeKutta propagation method - charged/neutral */  
183     template <class T> bool propagateRungeKuttaT(ExtrapolationCell<T>& eCell,  
184                                                PropagationCache& pCache,  
185                                                const T& tParameters,  
186                                                const Surface& sf) const;
```

# Track parametrization

- How can a track be parametrized?
  - ATS uses surface centric concept
    - =>  $(l_0, l_1, \varphi, \theta, \frac{q}{p})$
  - how flexible should the code be?
    - e.g.  $(l_0, l_1, \varphi, \eta, \frac{q}{p_T})$  or even  $(l_0, l_1, \varphi, \eta, \frac{q}{p_T}, m)$



- algorithmic code (e.g. of Kalman Filter) is independent of parametrization, but of course the form of your Jacobians changes
- technically, track parameters is just a vector of numbers
  - => meaning of individual numbers is usually a convention
- value range for some parameters is limited (e.g. angles)
  - => hard to enforce on a generic level



# ParameterSet class

```
ParameterSet<loc1,phi> p1(0.5, 1.5*M_PI);  
cout << p1->get<loc1>() <<endl; // prints 0.5  
cout << p1->get<loc2>() <<endl; // gives compiler error  
cout << p1->get<phi>() <<endl; // gives -0.5*M_PI  
p1->getParameters(); // returns fixed size vector  
p1->projector(); // returns fixed size projection matrix
```

- perform consistency checks at compile time whenever possible  
=> improved performance and code stability
- provide a mechanism for checking/transforming parameter values  
caveat: at the moment only for non-local parameters
- make use of fixed size matrices/vectors whenever possible
- **fixed size** projection matrix from full parameter space created at compile time

$$\begin{pmatrix} l_1 \\ \varphi \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}}_{\text{projector}} \begin{pmatrix} l_0 \\ l_1 \\ \varphi \\ \eta \\ q \\ p \end{pmatrix}$$

# Measurement class

**dynamic sized vector**

```
class MeasurementBase
{
    virtual VectorX parameters() const = 0;
};
```

```
class Measurement1D: public MeasurementBase
{
    VectorX parameters() const override
    {
        return getParameters();
    }

    Vector1 getParameters() const;
};
```

```
class Measurement2D: public MeasurementBase
{
    VectorX parameters() const override
    {
        return getParameters();
    }

    Vector2 getParameters() const;
};
```

**fixed size vectors**

- concrete measurements are of different (C++) types  
=> need common base class to store concrete measurements in a single vector, right?
- measurement base class must rely on dynamically sized vectors/matrices

# Dynamic vs. fixed size matrices

- ATS is using Eigen as algebra library
- What is the performance penalty when using dynamic instead of fixed size matrices?
  - 1) Eigen-3.2.7, g++ 4.9.2

	Dynamic sized	Fixed size	converted
M(2,5) x v(5)	1.7277s	0.797514s	1.45323s
M(5,2) x M(2,5)	3.53559s	2.67478s	3.48556s

	Dynamic sized	Fixed size	converted
M(2,5) x v(5)	0.154546s	0.00747539s	0.0116945s
M(5,2) x M(2,5)	0.217145s	0.031977s	0.0326164s

=> optimization compiler flags give huge speedup, fixed size operations are a factor 8-20 faster

# ATS approach: boost::variant

- keep performance benefit from using fixed size matrix operations while allowing to treat different concrete measurements uniformly

uniformly

```
enum class ParDefs {loc1, loc2, eta, phi, qop};
```

```
template<ParDefs... pars>  
class Measurement
```

```
{  
    typedef Vector<sizeof...(pars)> ParVector;  
    ParVector parameters() const;  
};
```

```
typedef boost::variant<...> FittableMeasurement;
```

```
Measurement<loc1,phi> m1;  
Measurement<loc0,loc1> m2;
```

```
std::vector<FittableMeasurement> vMeasurements = {m1,m2};
```

```
struct print_visitor: public boost::static_visitor<>  
{  
    template<typename T>  
    void operator()(const T& m) const  
    {  
        std::cout << "Generic measurement with " << m.parameters() << std::endl;  
    }  
  
    void operator()(const Measurement<phi>& m) const  
    {  
        std::cout << "Phi measurement: " << m.parameters() << std::endl;  
    }  
};
```

```
print_visitor v;  
for(const auto& m: vMeasurements)  
    boost::apply_visitor(v,m);
```

generic measurement definition

all possible measurement types

type which can hold any possible concrete measurement

# Summary and outlook

- active development ongoing with significant progress over the last weeks
- design is driven by flexibility and performance aspects
- applicability in real-world experiments is highest priority (e.g. allow for measurement recalibration during fitting)
- major work is still needed on documentation and tests
- aim for first prototype release in April

# Detailed performance comparison

Execution time per operation in  $\mu\text{s}$  (measured from 10M operations in 20 runs)

Eigen	Flags	code	Dynamic size		Fixed size		Converted	
			M(2,5) x v(5)	M(5,2) x M(2,5)	M(2,5) x v(5)	M(5,2) x M(2,5)	M(2,5) x v(5)	M(5,2) x M(2,5)
3.1.3			1.7932 +/- 0.0673	3.4949 +/- 0.1281	0.8227 +/- 0.0525	2.7014 +/- 0.1134	1.5273 +/- 0.0466	3.6980 +/- 0.2273
	<b>-O2</b>		<b>0.1514 +/- 0.0085</b>	<b>0.2260 +/- 0.0088</b>	<b>0.0076 +/- 0.0002</b>	<b>0.0336 +/- 0.0016</b>	<b>0.0317 +/- 0.0010</b>	<b>0.0581 +/- 0.0024</b>
	-O2	.noalias	0.1136 +/- 0.0031	0.1778 +/- 0.0094	0.0026 +/- 0.0002	0.0294 +/- 0.0033	0.0305 +/- 0.0017	0.05878 +/- 0.0016
	-O3		0.1481 +/- 0.0041	0.2207 +/- 0.0058	0.0076 +/- 0.0003	0.0305 +/- 0.0028	0.0270 +/- 0.0005	0.0588 +/- 0.0018
3.2.7	-O2		0.1450 +/- 0.0020	0.2268 +/- 0.0097	0.0077 +/- 0.0003	0.0319 +/- 0.0023	0.0119 +/- 0.0003	0.0405 +/- 0.0010
	-O2	.noalias	0.1156 +/- 0.0099	0.1735 +/- 0.0087	0.0027 +/- 0.0002	0.0258 +/- 0.0008	0.0070 +/- 0.0002	0.0358 +/- 0.0024
	-O3		0.1489 +/- 0.0060	0.2222 +/- 0.0057	0.0076 +/- 0.0001	0.0298 +/- 0.0007	0.118 +/- 0.0003	0.0333 +/- 0.0006
3.3-beta1	-O2		0.0594 +/- 0.0006	0.1576 +/- 0.0032	0.0097 +/- 0.0001	0.0399 +/- 0.0020	0.0158 +/- 0.0002	0.0452 +/- 0.0010
	<b>-O2</b>	<b>.noalias</b>	<b>0.0214 +/- 0.0002</b>	<b>0.0975 +/- 0.0011</b>	<b>0.0053 +/- 0.0010</b>	<b>0.0305 +/- 0.0012</b>	<b>0.105 +/- 0.0001</b>	<b>0.0356 +/- 0.0004</b>
	-O3		0.0625 +/- 0.0097	0.1596 +/- 0.0145	0.0057 +/- 0.0010	0.0399 +/- 0.0040	0.0108 +/- 0.0003	0.0448 +/- 0.0035
	-O3	.noalias	0.0203 +/- 0.0003	0.1002 +/- 0.0006	0.0053 +/- 0.0001	0.0283 +/- 0.0002	0.0105 +/- 0.0001	0.0347 +/- 0.0002