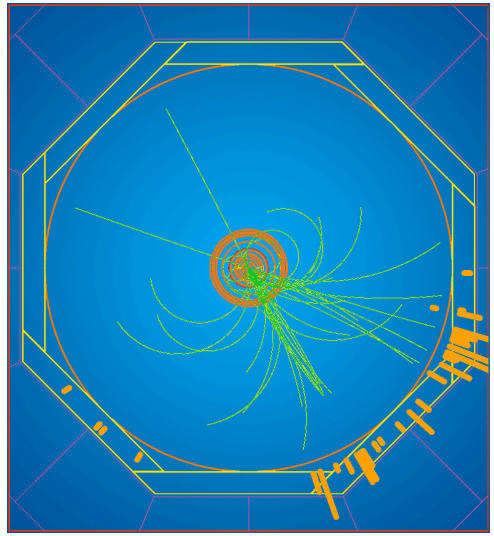
ILD Tracking – Framework Status

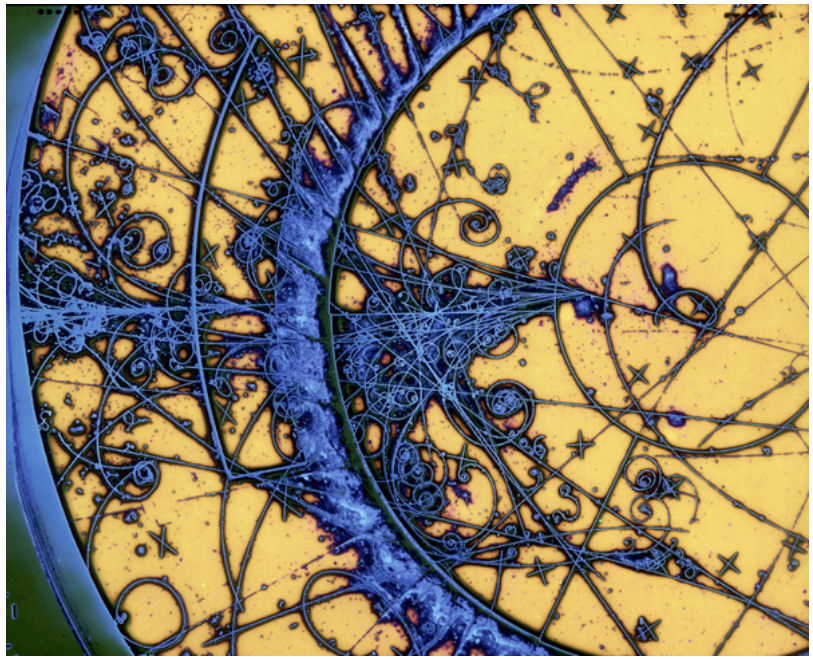
Steve Aplin
DESY

ILD Software Meeting
24th August 2011



iLCSOft

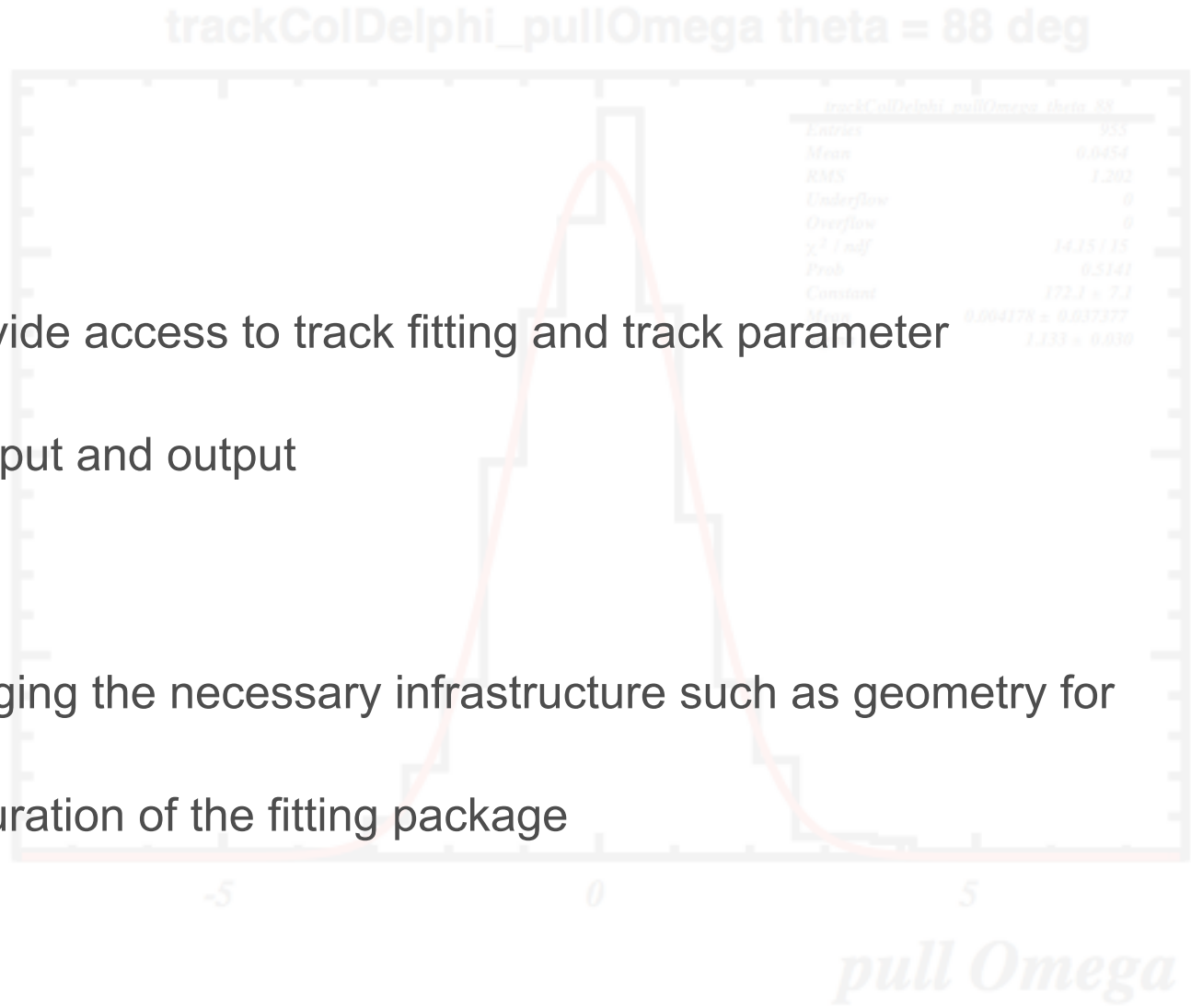
- Current Status
- Plans



IMarlinTrack and IMarlinTrkSystem

- **IMarlinTrack**
  - interface class to provide access to track fitting and track parameter propagation in **Marlin**
  - uses **LCIO** for both input and output

- **IMarlinTrkSystem**
  - responsible for managing the necessary infrastructure such as geometry for the track fitting
  - controlling the configuration of the fitting package



IMarlinTrack and IMarlinTrkSystem

- **IMarlinTrack** interface extended to be more convenient when using an iterative fitter.
- Additional Methods provided:

```
//** initialise the fit using the supplied hits only, using the given order to determine the direction of the track
virtual int initialise( bool direction ) = 0 ;

//** initialise the fit with a track state
virtual int initialise( const IMPL::TrackStateImpl& ts) = 0 ;

//** update the current fit using the supplied hit, return code via int. Provides the Chi2 increment to the fit from adding the hit via reference.
virtual int addAndFit( EVENT::TrackerHit* hit, double& chi2increment, double maxChi2Increment=DBL_MAX ) = 0 ;

//** get track state, return code via int
virtual int getTrackState( IMPL::TrackStateImpl& ts ) = 0 ;

//** get track state at measurement associated with the given hit, return code via int
virtual int getTrackState( EVENT::TrackerHit* hit, IMPL::TrackStateImpl& ts ) = 0 ;
```

continued …

IMarlinTrack and IMarlinTrkSystem

```
//** propagate track state at measurement associated with the given hit, the fit to the point of closest approach to the given point.
virtual int propagate( const gear::Vector3D& point, EVENT::TrackerHit* hit, IMPL::TrackStateImpl& ts) = 0 ;

//** propagate track state at measurement associated with the given hit, to numbered sensitive layer, returning TrackState via provided reference
virtual int propagateToLayer( bool direction, int layerNumber, EVENT::TrackerHit* hit, IMPL::TrackStateImpl& ts) = 0 ;

//** extrapolate track state at measurement associated with the given hit, to the point of closest approach to the given point.
virtual int extrapolate( const gear::Vector3D& point, EVENT::TrackerHit* hit, IMPL::TrackStateImpl& ts) = 0 ;

//** extrapolate track state at measurement associated with the given hit, to numbered sensitive layer, returning TrackState via provided reference
virtual int extrapolateToLayer( bool direction, int layerNumber, EVENT::TrackerHit* hit, IMPL::TrackStateImpl& ts) = 0 ;

//** extrapolate track state at measurement associated with the given hit, to numbered sensitive layer, returning intersection point in global coordinates
virtual int intersectionWithLayer( bool direction, int layerNumber, EVENT::TrackerHit* hit, gear::Vector3D& point) = 0 ;
```

Marlin and KalTest

- MarlinTrk and MarlinTrkProcessors packages provided in the **MarlinReco** svn repository:
  - **MarlinTrk** – this contains the interface classes as well as the implementation of the interfaces, presently only for **KalTest**
  - **MarlinTrkProcessors** – Contains example **Processors** which use the functionality provided in **MarlinTrk.** Presently an example Refitter processor is provided as well as simple planar digitiser, demonstrating how to use the new **TrackerHitPlane** class, as well as the use of **CellID0** for the track reconstruction.

Marlin and KalTest

- Currently implementations of the **VXD**, **TPC** and **FTD** sub-detectors are provided in **KalDet**.
- The description of the **FTD** in **GEAR** (FTDParametersImpl) has been provided by Jordi Duarte, which he has implemented in the Mokka driver SFtd06.
- For the **SIT**, **SET** and **ETD** we are currently working with Aurore et al. to get the description in **GEAR** and write this out in **Mokka** during the construction of the detectors.
- For the SIT and SET a first shot at this has been to rename the **VXParamters** and **VXDLayerLayout** to **ZPlanarParameters** and **ZPlanarLayerLayout** in **GEAR**.

Marlin and KalTest

- By last months meeting mainly only the fitting methods had been implemented using Marlin and KalTest

- This has now been extended to provide almost the complete interface, e.g. **extrapolateToLayer** which is very useful during PatRec, as well as the recently added methods mentioned above.

- Currently the only methods missing are those which involve navigating to “xxxNextLayer”, e.g.

```
virtual int intersectionWithNextLayer( bool direction, EVENT::TrackerHit* hit, int& layerNumber,
gear::Vector3D& point) = 0 ;
```

- This is due to the fact that it is not straight forward in many areas of the detector, and requires dedicated navigation for effective implementation.

Cell ID Numbering

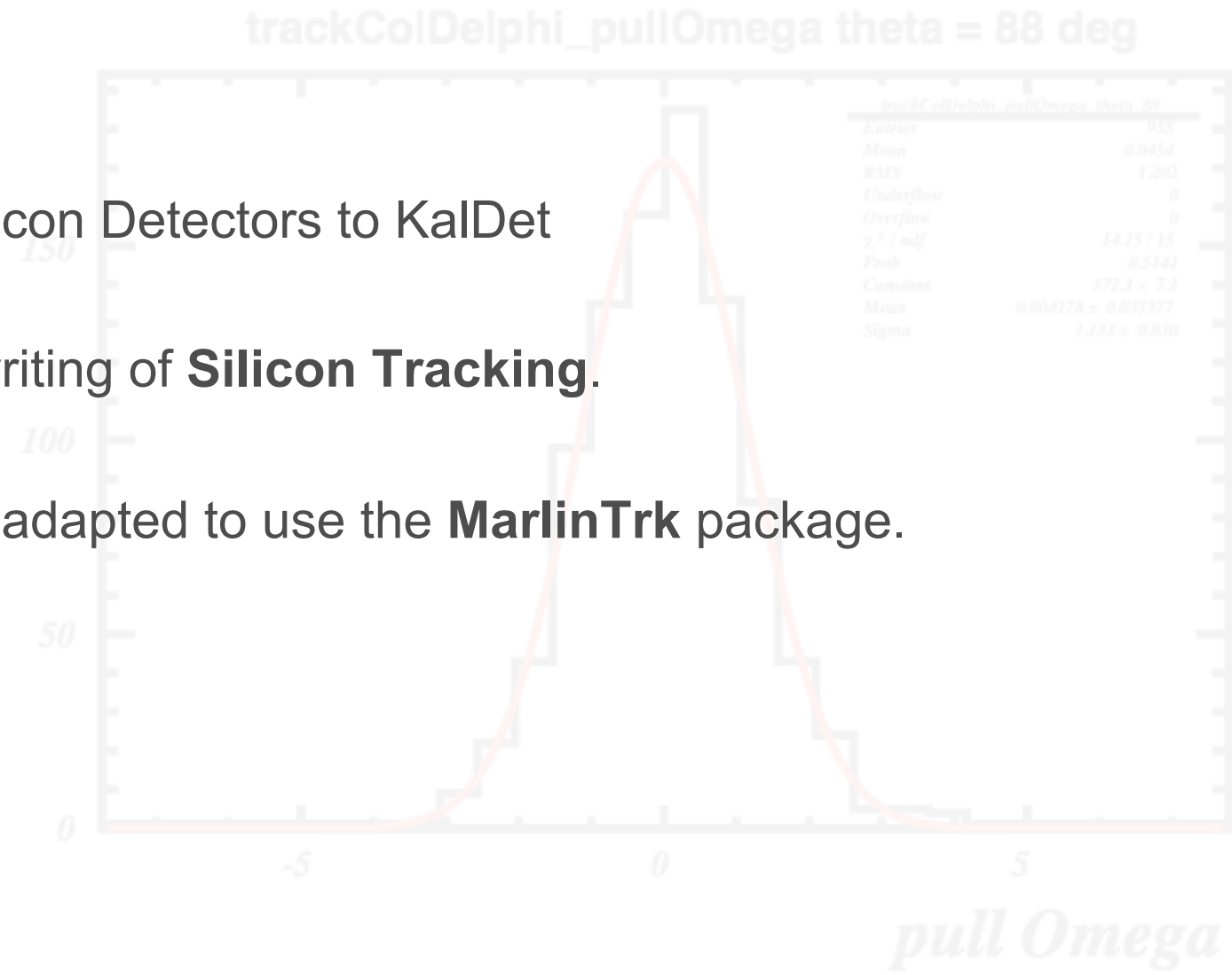
- MarlinKalTest now uses the CellID Numbering scheme as shown by Frank in the previous talk.

  Sub-detectors: VXD, SIT, TPC, SET, FTD, ETD

nbits	key	use
5	subdet	these ID's will be assigned centrally
2	side	signed to allow us to store +1 and -1 for forward detectors and 0 for barrel
9	layer	provides a maximum of 512 layers easily sufficient for the TPC, indexed in increasing r for barrel, increasing |z| for forward
8	module	refers to the assembly holding the sensors, e.g. ladder in the case of VXD and SIT and SET, and Petal in FTD indexed in increasing phi
8	sensor	refers to the element containing a group of channels with a common local coordinate system e.g. a wafer

Plans

- Add remaining Silicon Detectors to KalDet
- Start with the re-writing of **Silicon Tracking**.
- **Clupatra** is being adapted to use the **MarlinTrk** package.



Summary

- Updated implementations of **MarlinTrk** and **MarlinTrkProcessors** provided in svn:
  - https://svnsrv.desy.de/public/marlinreco/MarlinTrk/trunk
  - https://svnsrv.desy.de/public/marlinreco/MarlinTrkProcessors/trunk
  - These now provide, both fitting code and the necessary intersection, extrapolation and propagation methods needed for pattern recognition
- Have now agreed on the use of **CellID0** and **CellID1**.
- Move forward with the Pattern Recognition: this requires digitisers