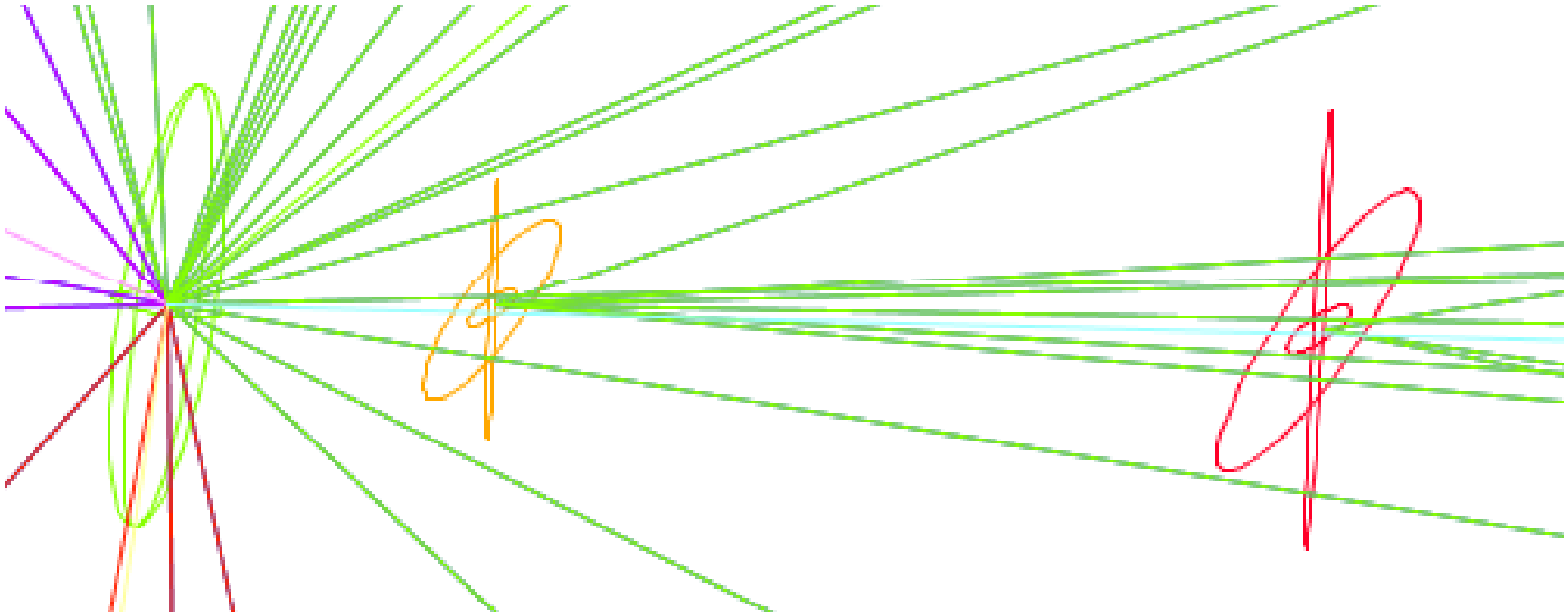


Vertexing How To

(and flavour tagging/vertex charge!)

Ben Jeffery
Oxford University
(on behalf of the LCFI collaboration)

Outline

- Two vertexing algorithms, flavour tagging and vertex charge measurement are available from the LCFI Vertex Package
- Current release runs in the Marlin framework, but can be driven with input from MarlinReco and org.lcsim as all interfaces are LCIO
- Use of package directly from java in development *(proof of principle demonstrated)*

- The questions I hope to answer:
  - Where's all this documented?
  - How do I make the input to vertexing?
  - How do I make vertices and tags?
  - How do I access the vertices and tags?
  - How do I look at the vertices?
  - Where are the known potential slip-ups?

What's in the package?

Ten modular parts (Processors in Marlin, drivers in org.lcsim):

In run order:

- **TrueJetFlavour** - Determine MC flavour and charge of reconstructed jets
- **RPCut** - Select ReconstructedParticles based on Track parameters, number of hits etc.
- **PerEventIPFitter** - Find the event IP
- **ZVTOPZVRES** - Find secondary vertices topologically
- **ZVTOPZVKIN** - Find secondary vertices kinematically
- **FlavourTagInputs** - From vertices and tracks calculate discriminating variables for the neural net
- **FlavourTag** - Calculate network output (tag value)
- **VertexCharge** - Calculate charge of decaying hadron *(next release)*
- **LCFIAIDAPlot** - Create AIDA file with many diagnostic plots *(next release)*
- **NeuralNetTrainer** - Train networks

The example steering files combine these to make the Hawking's bc tagging procedure *(LC-PHSM-2000-021)* parameters for these from previous Brahms study.

Documentation is in "LCFIVertex/doc" in the form of doxygen generated web pages, the Processor class pages are a good starting point. There are also several extra pages of explanatory prose under "related pages"
Code and docs at: http://ilcsoft.desy.de/portal/software_packages/lcfivertex/index_eng.html

Combining the processors

Flow diagram for example steering
In “LCFIVertex/steering”

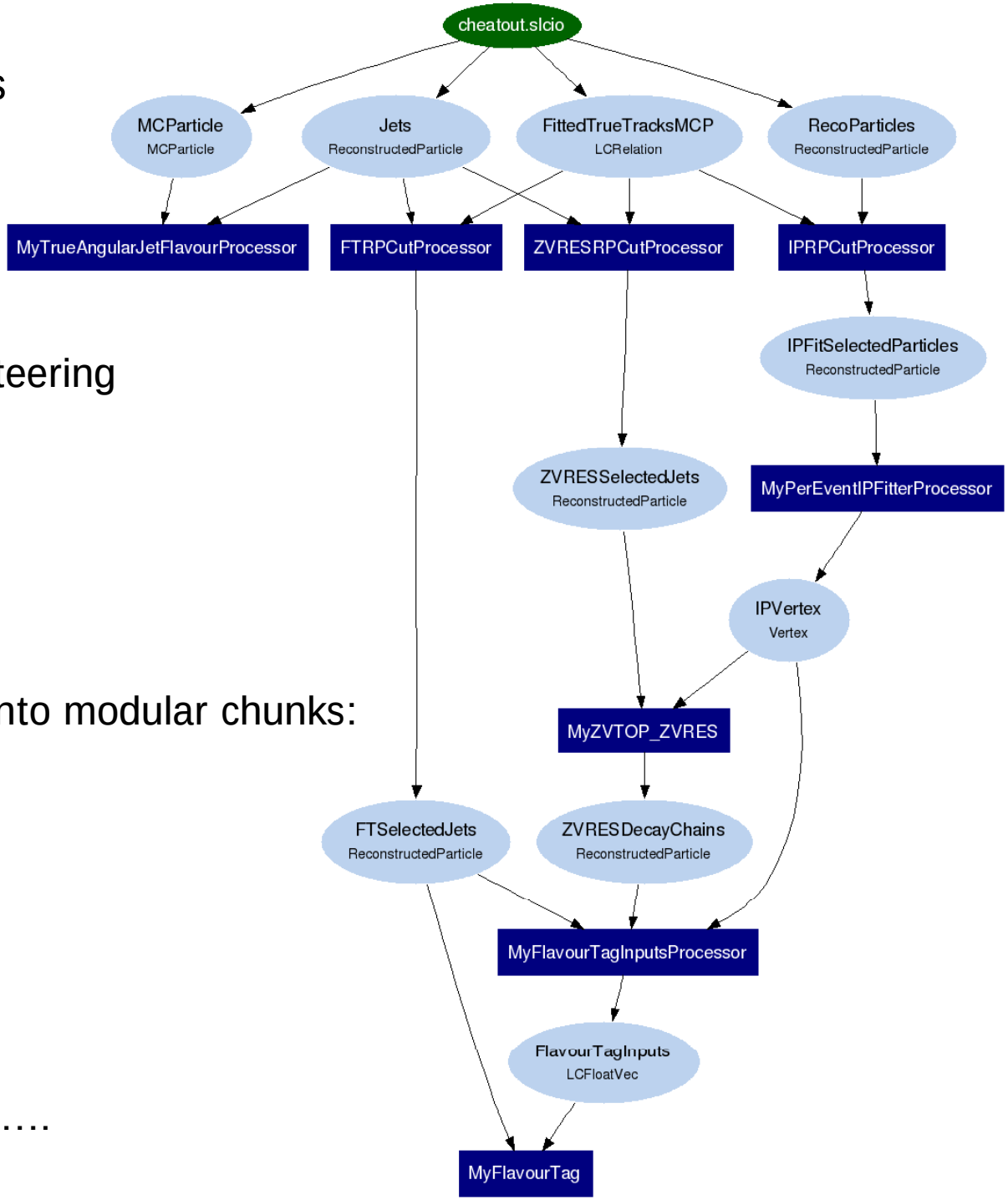
= Processors

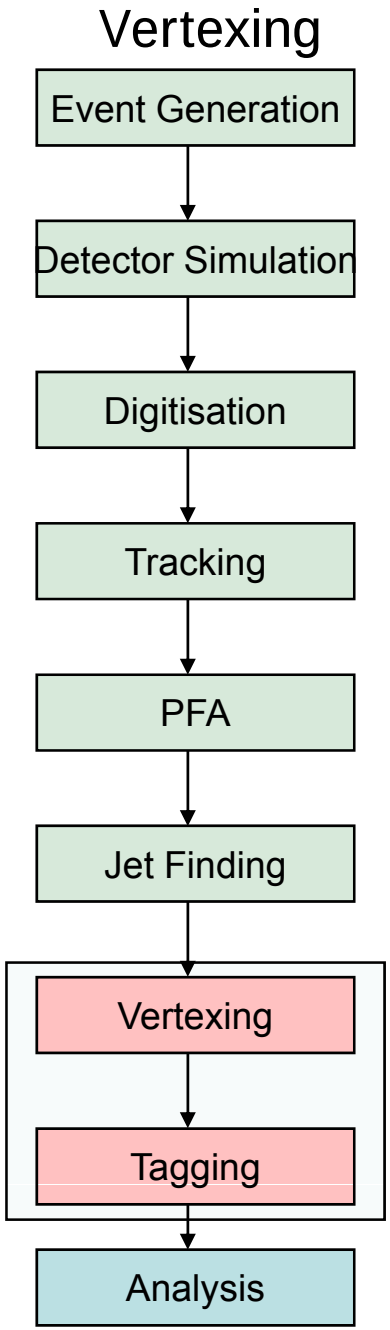
= LCIO Collections

Example steering files split into modular chunks:

1. truejetflavour.xml
2. ipfit.xml
3. zvres.xml
4. fti.xml
5. ft.xml

But first we need an input….

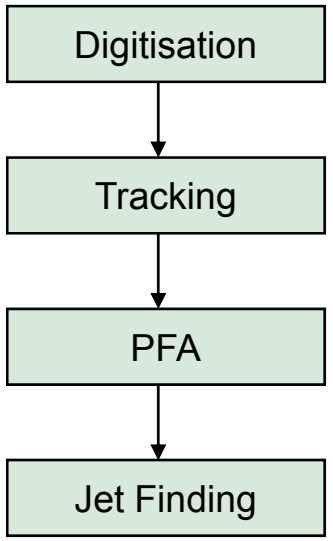




- Vertexing at the bottom of the reconstruction food chain!
  - *Very* sensitive to quality of input tracks
  - Correct track covariance matrices particularly important
  - Need to understand tracking and digitisation codes well
- Many thanks to the responsive MarlinReco and org.lcsim developers (esp Alexei Raspereza and Norman Graf)
- Vertexing and tagging performed on jet basis so good jet finding also needed
- Now possible using input from both MarlinReco and org.lcim through the magic of LCIO
- Will detail process with MarlinReco input, then mention details of using org.lcsim input

Making Tracks *(Mokka/MarlinReco)*

- The output from Mokka needs to go through several stages in MarlinReco to produce input suitable for the vertexing.

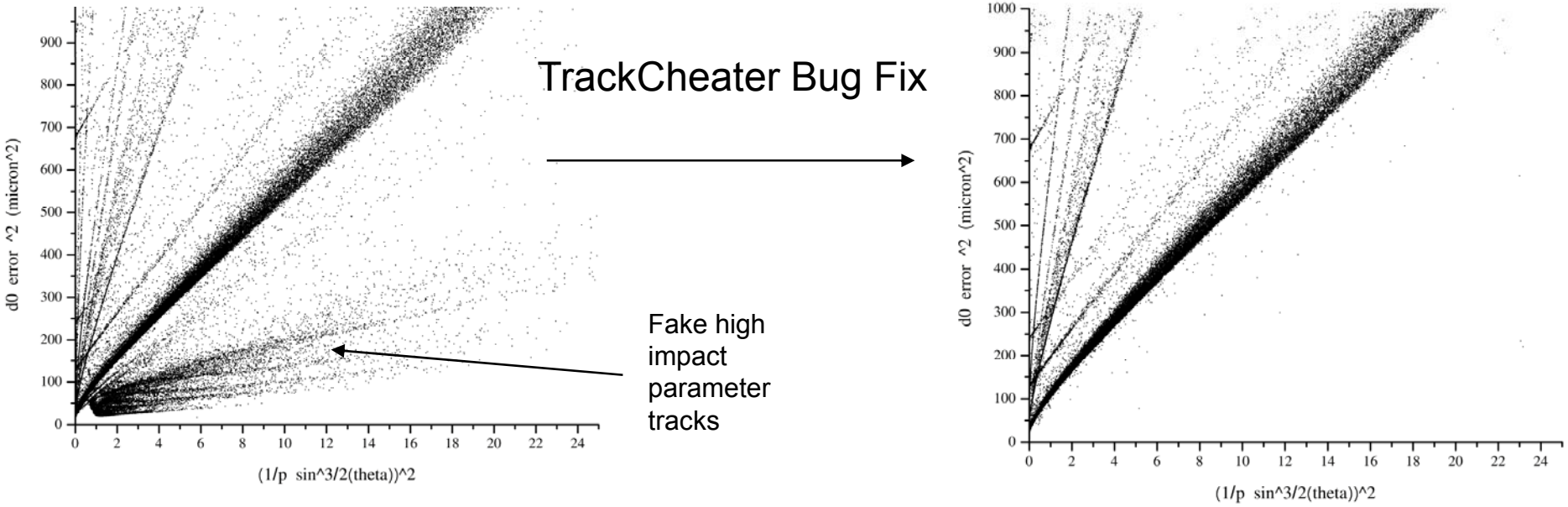


- An example steering file to perform the simplest reconstruction with cheated tracks and clusters is included in the vertex package: "LCFIVertex/steering/cheattracks+jetfind.xml"
  - http://www-zeuthen.desy.de/lc-cgi-bin/cvsweb.cgi/~checkout~/LCFIVertex/steering/cheattracks%2bjetfind.xml?rev=1.1;content-type=text%2Fplain;cvsroot=marlinreco

- Full tracking is available with *FullLDCTracking* see "MarlinReco/doc/FullLDCTracking_Manual.pdf"
  - http://www-zeuthen.desy.de/lc-cgi-bin/cvsweb.cgi/~checkout~/MarlinReco/doc/FullLDCTracking_Manual.pdf?rev=1.1;content-type=text%2Fplain;cvsroot=marlinreco
- Both work well with some caveats:

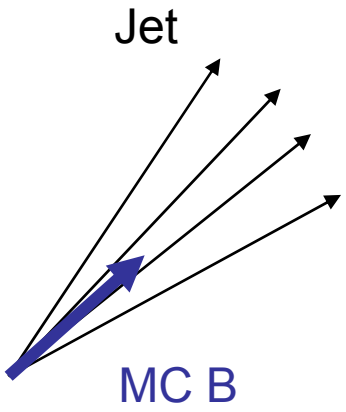
Making Tracks *(Mokka/MarlinReco)*

- Only the **cvs head** of MarlinReco gives quality tracks for vertexing from both TrackCheater and LDCFullTracking.
- Only LDC01Sc currently fully supported
  - Issues with TPC digitisation and VXD ladders
- SatoruJetFinderProcessor has limit of 300 particles
  - Low momentum cut used as un-ideal work around
- Geometry information from GEAR xml file must be correct
  - MokkaGEAR improvements in cvs head to automate this
- *Very* active development, tests of vertexing software lead to several bug fixes – mostly to do with track covariance matrices



Finding the truth and the IP

Now we have jets we start using the vertex package:



Jets are reconstructed objects so determination of MC flavour and charge needs a little work

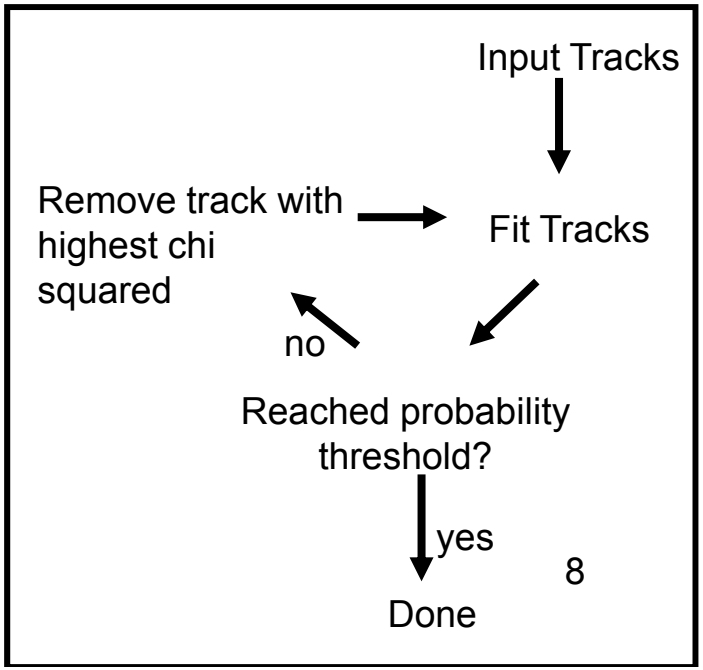
- *TrueAngularJetFlavourProcessor*
- Matches direction of MC heavy flavour and jets
- Output to *LCIntVec* collection of PDG codes of initial parton and its charge

Some versions of Mokka have broken MC tree information: 6-04-03 onwards known to work

Need to find IP before finding secondaries

- *PerEventIPFitter*
- Tear down fitter of event tracks
- Output to *Vertex* collection

Long run time *(fixed in imminent release)*



Find secondary vertices

- ZVRES is the standard for vertexing in the package. Many previous talks explaining principles
- Input is collection of *ReconstructedParticle* **jets** and an IP *Vertex* (default is a manually set IP)
- Output is three collections:
  - *Vertex* **Found vertices**
  - *ReconstructedParticle* **Tracks in vertices**
  - *ReconstructedParticle* **Decay chain**
  - For details of how these work see my Orsay workshop talk: http://ilcagenda.linearcollider.org/getFile.py/access?contribId=53&sessionId=20&resId=1&materialId=slides&confId=1446

Badly formed tracks can cause *very* long execution times *(fix planned)*

- Primary vertex in ZVTOP result is a copy of the input IP, not the one produced by the algorithm

Calculating discriminating variables for flavour tag

- The vertices (if any) and tracks are used to calculate a set of discriminating variables – see Orsay talks for details
- *FlavourTagInputsProcessor*
- Output is a collection of *LCFloatVec*, named in the run header.

Running the neural net

- The *LCFloatVec* is read in and fed to a neural net
- *FlavourTagProcessor*
- Output is collection of LCFloatVec, named in the run header
- Trained nets can be downloaded from special cvs at

http://www-zeuthen.desy.de/lc-cgi-bin/cvsweb.cgi/?cvsroot=tagnet


None yet trained on ILC full simulation


Hawkings method calls for 9 different nets – explained in *LC-PHSM-2000-021*

General Comments

So if you have a recent Mokka file all this should be as simple as:

```
Marlin cheattracks+jetfind.xml
Marlin truejetflavour.xml
Marlin ipfit.xml
Marlin zvres.xml
Marlin fti.xml
Marlin ft.xml
```

Of course these could be combined in one steering file.

- Currently LDC VXD00 geometry hard coded into fiducial cut *(fix using GEAR in development)*
- V0's are cheated, and there is no treatment of photon conversions yet.

As the processors use LCIO run header information "SkipNEvents" in Marlin causes problems as the header can be skipped.

How do I get at the vertex and tagging information?

- With standard LCIO calls
- This is explained in the documentation at:http://www-pnp.physics.ox.ac.uk/~hillert/VP/LCFIVertex-v00-01/LCIO.html

- It's as easy as this in your analysis processor's *processEvent():*
  - plus a little in the *processHeader()*

```
lcio::LCCollection* pInputs=pEvent->getCollection( _FlavourTagInputsCollectionName );
LCFloatVec* FTInputs = dynamic_cast<lcio::LCFloatVec*>( pInputs->getElementAt(0) );

double SecProb = (*FTInputs)[IndexOf["SecondaryVertexProbability"]];
double VertexMass = (*FTInputs) [IndexOf["PTCorrectedMass"]];
```

Getting at tracks in vertices is a tiny bit more work – a code example is at the link above.

Bridging the Atlantic

Using the package from within Java is being developed. (JNI)

Currently can be used in Marlin with org.lcsim generated LCIO file as input!

Only need to install Marlin + LCFIVertex *(MarlinReco not needed)*

- Currently only tested with org.lcsim.mc.fast.tracking.MCFastTracking
- See example in JAS (Help -> Examples -> org.lcsim)

Use head version of org.lcsim - generates LCRelations needed to use MCParticle information

Minimal changes to example steering needed

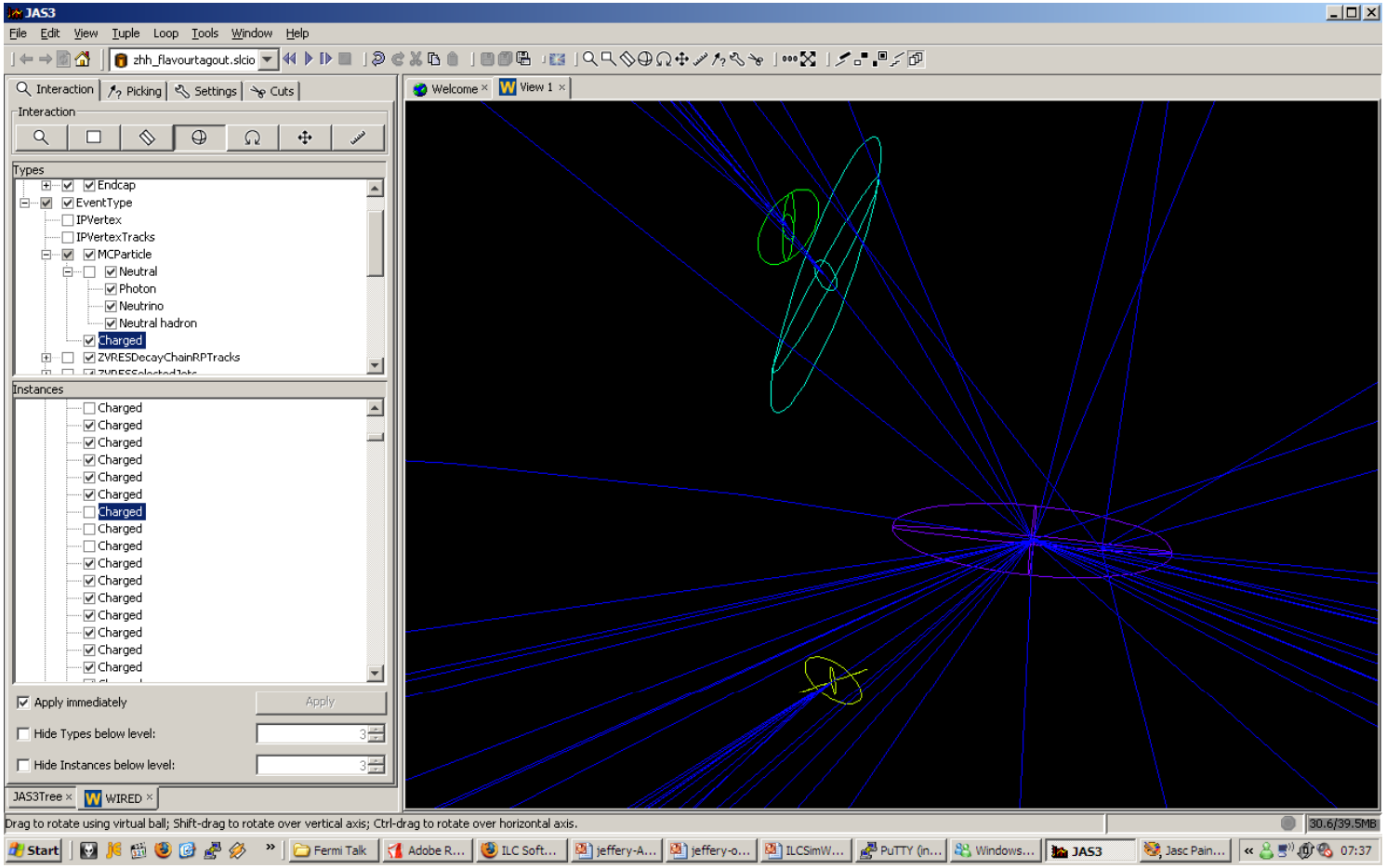
- Rename input collections to match those generated by org.lcsim

Disable cut based on sub-detector hit numbers array as it is not filled – hence out of array bounds crash *(fix planned)*

- In future will need GEAR file of vertex detector geometry for fiducial cut – should be simple conversion from compact.xml

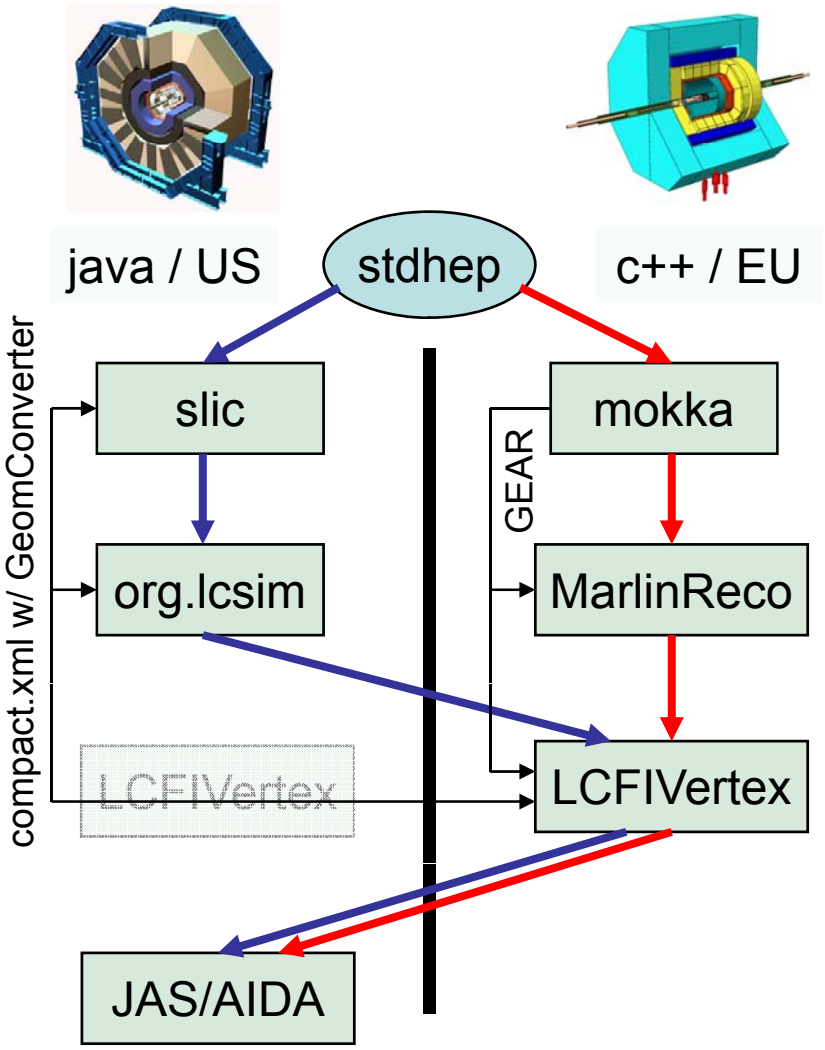
Bridging the Atlantic - display

- Whether the reconstruction was performed in org.lcsim, Marlin or a mix the physics analysis can be performed in JAS and visualised using WIRED4:



org.lcsim tracking/Marlin vertexing vertices in the most recent release of WIRED4

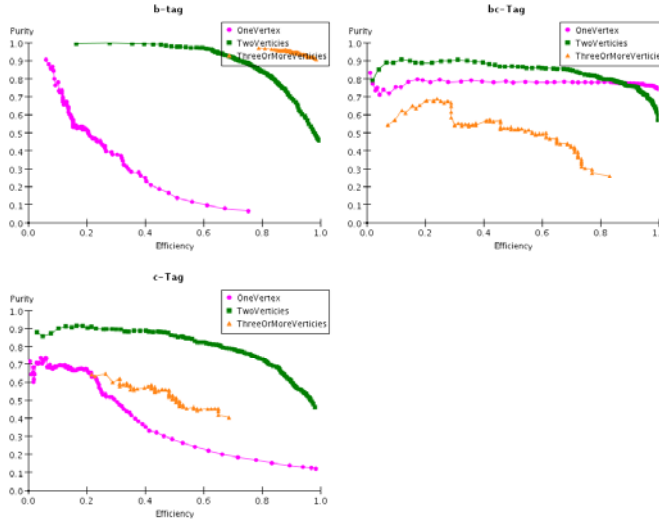
Bridging the Atlantic - Comparison Studies

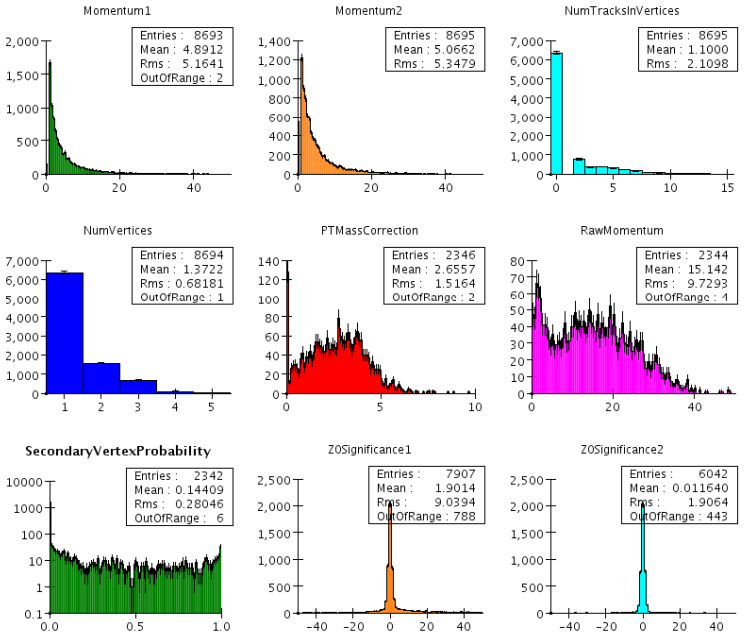


- The possibilities afforded by LCIO enable running the same events in both EU and US frameworks with the same vertexing code.
- Identical analysis using the *ReconstructedParticles* from both frameworks
- Can choose c++ or java for analysis

Release Schedule and future plans

- Soon after this workshop
  - Kalman fitter from CBM experiment to replace slow iterative fitter
  - Bug fixes
  - Documentation update
  - Separate VertexCharge processor
  - New comprehensive diagnostic plot processor
- Beyond
  - GEAR for fiducial cut
  - V0, photon conversion treatment
  - Charge dipole technique

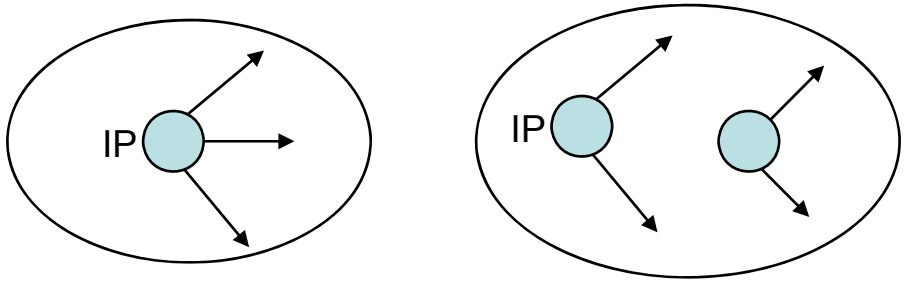




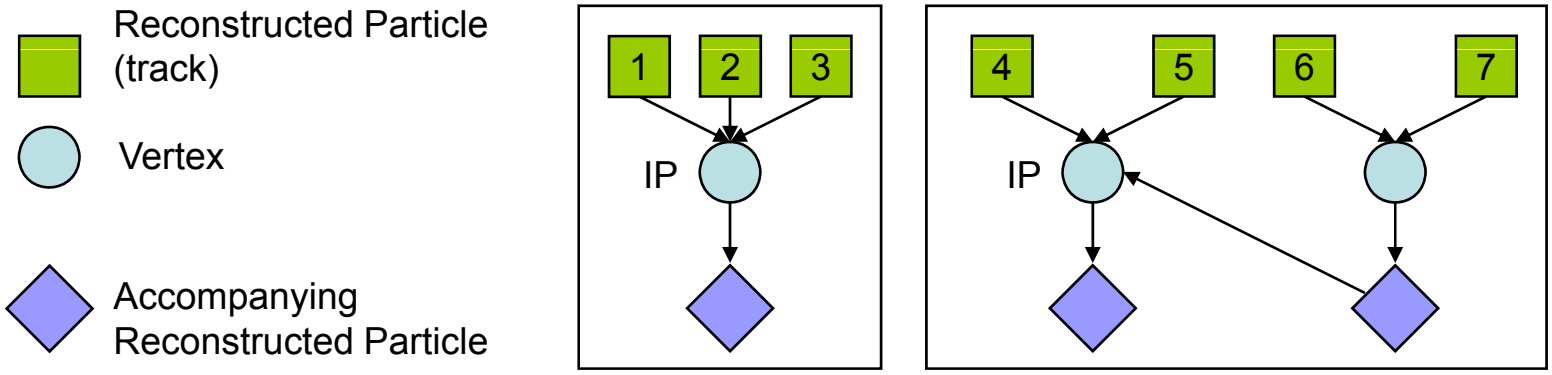
Backup Slides

Storage Of Vertexing Result

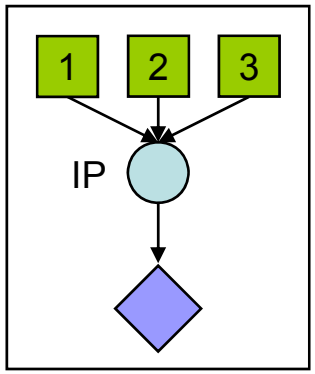
- ZVTOP Produces “Decay Chains” – sets of one or more vertices eg:

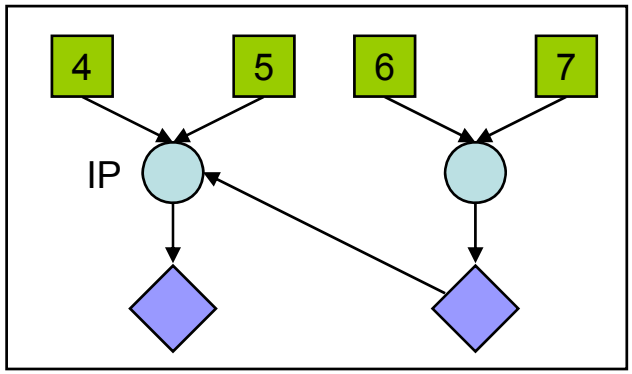


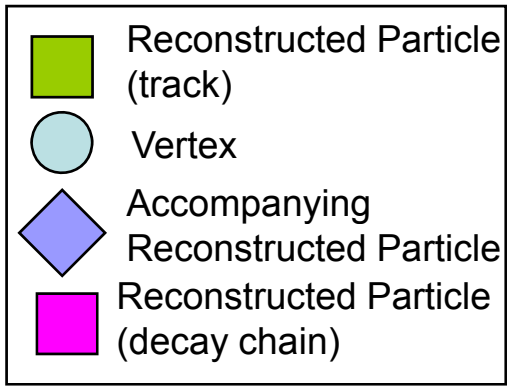
In LCIO each Vertex has an accompanying ReconstructedParticle which represents the decaying particle. This holds kinematic information. Each ReconstructedParticle points to its “startVertex”.



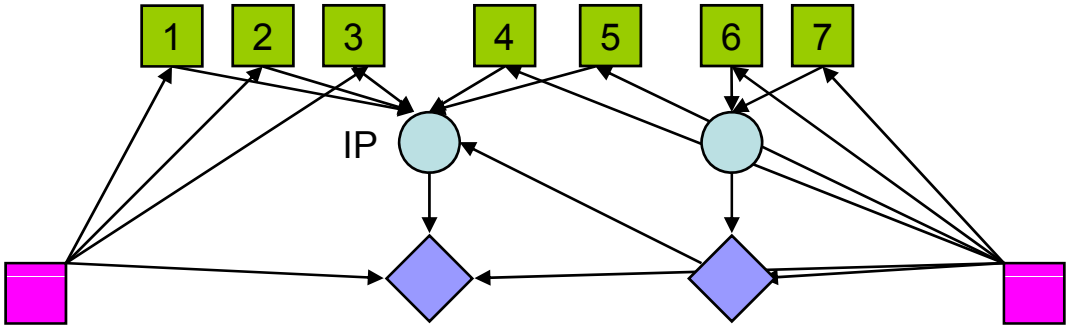
Storage Of Vertexing Result





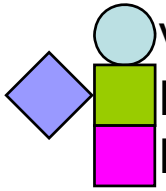


If these are in the same event they share the IP Vertex:



ReconstructedParticle representing decay chain points to all tracks in that chain (one for each jet)

Result is three collections:

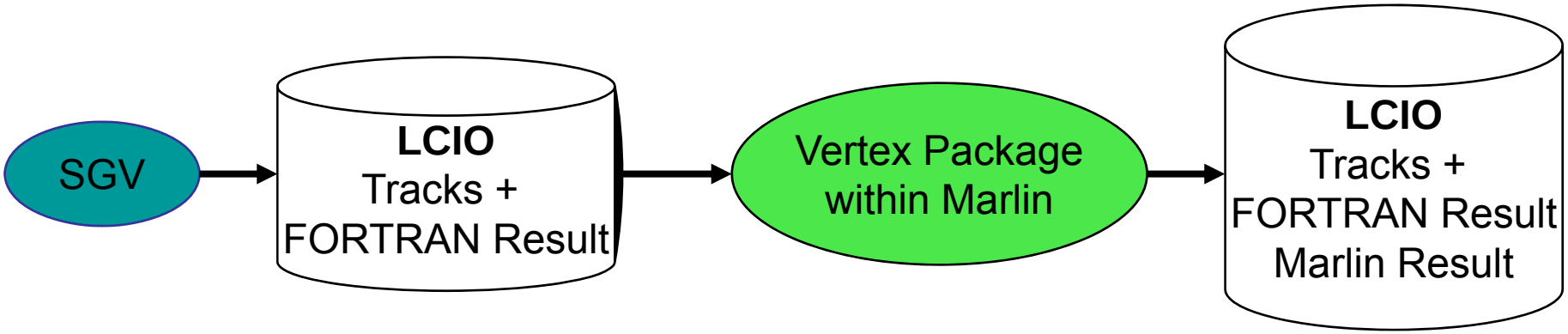


Vertices
DecayChainTracks
DecayChains

Code examples of accessing this information are in the documentation

Fast MC Testing

- Initially needed to run identical results through legacy FORTRAN code (embedded in fast MC SGV) and Marlin vertex package.
- LCIO's FORTRAN interface made this very easy:



- Just add a couple of header files to the SGV code – enabled detailed debugging of ZVTOP.
- Interface code given SGV author (Mikael Berggren) and has since been used for other studies.