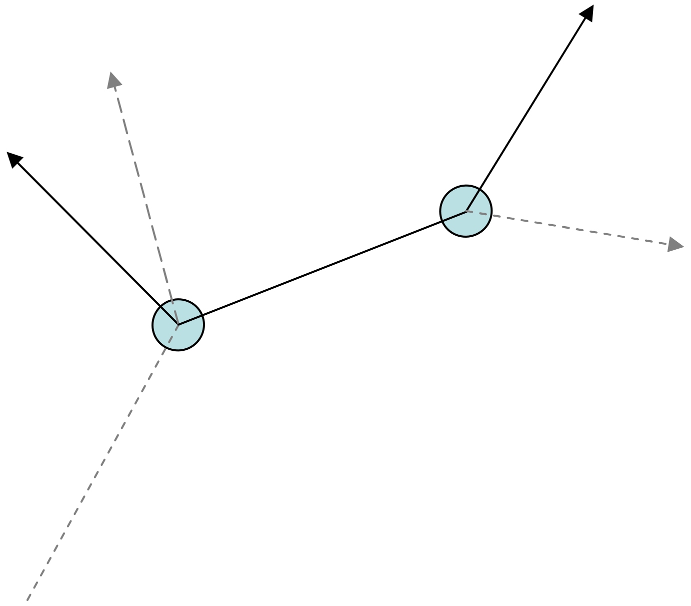
# The LCFI Vertex Package Details and Testing

Ben Jeffery
Oxford University
on behalf of the LCFI collaboration



## *Summary*

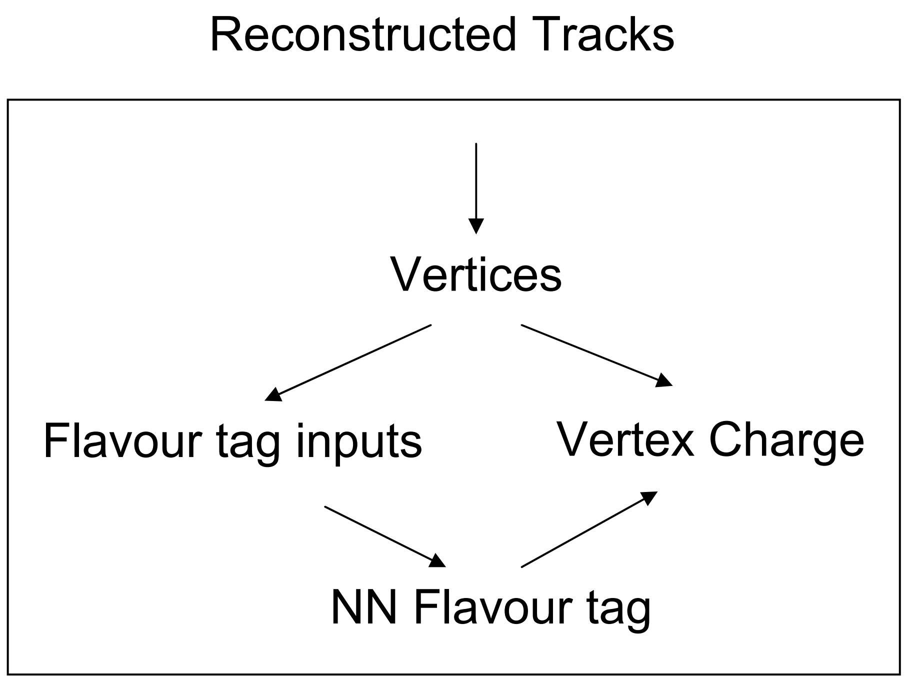
Vertex package consists of ~20k lines of new *documented, structured* C++ code replacing relatively undocumented FORTRAN used at SLD, OPAL and for TESLA TDR studies.

*Performance should be maintained* (at least!) while providing increased flexibility, maintainability and LCIO interface.

Where possible practical development led by high level concepts, *NOT* copying FORTRAN nested loops and gotos

*Comparison* with previously used code needed at every level of the package.

Test each layer of reconstruction after confirming performance of the layer below it.



## *Flexibility of the package*

•Vertexing, calculation of neural net inputs and neural net *decoupled* into separate Marlin processors while retaining option to run together outside Marlin framework.

•Each can be run *independently* – eg when only vertexing is needed or to save vertex result to disk for later tagging.

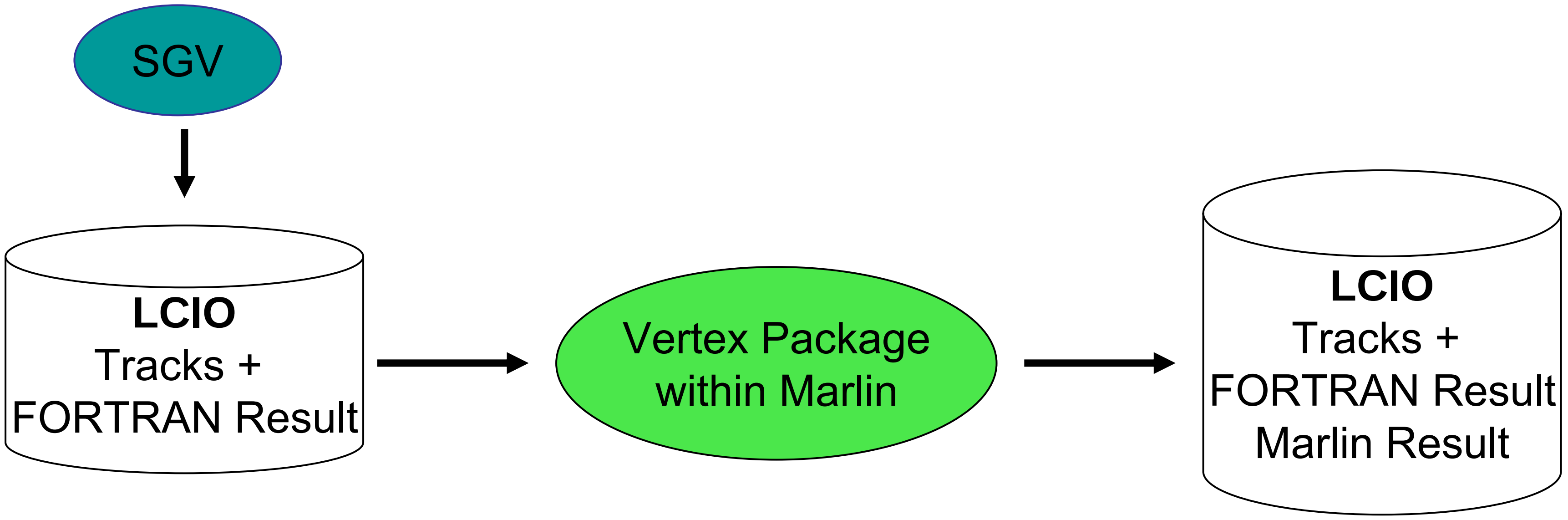
•Configuration of neural network inputs simple – can even use variables from code outside this package (as long as its in LCIO)

•Networks described by XML files read at run time.

•A repository for sharing network descriptions has been agreed.

•Class level and high level documentation will be provided on release.

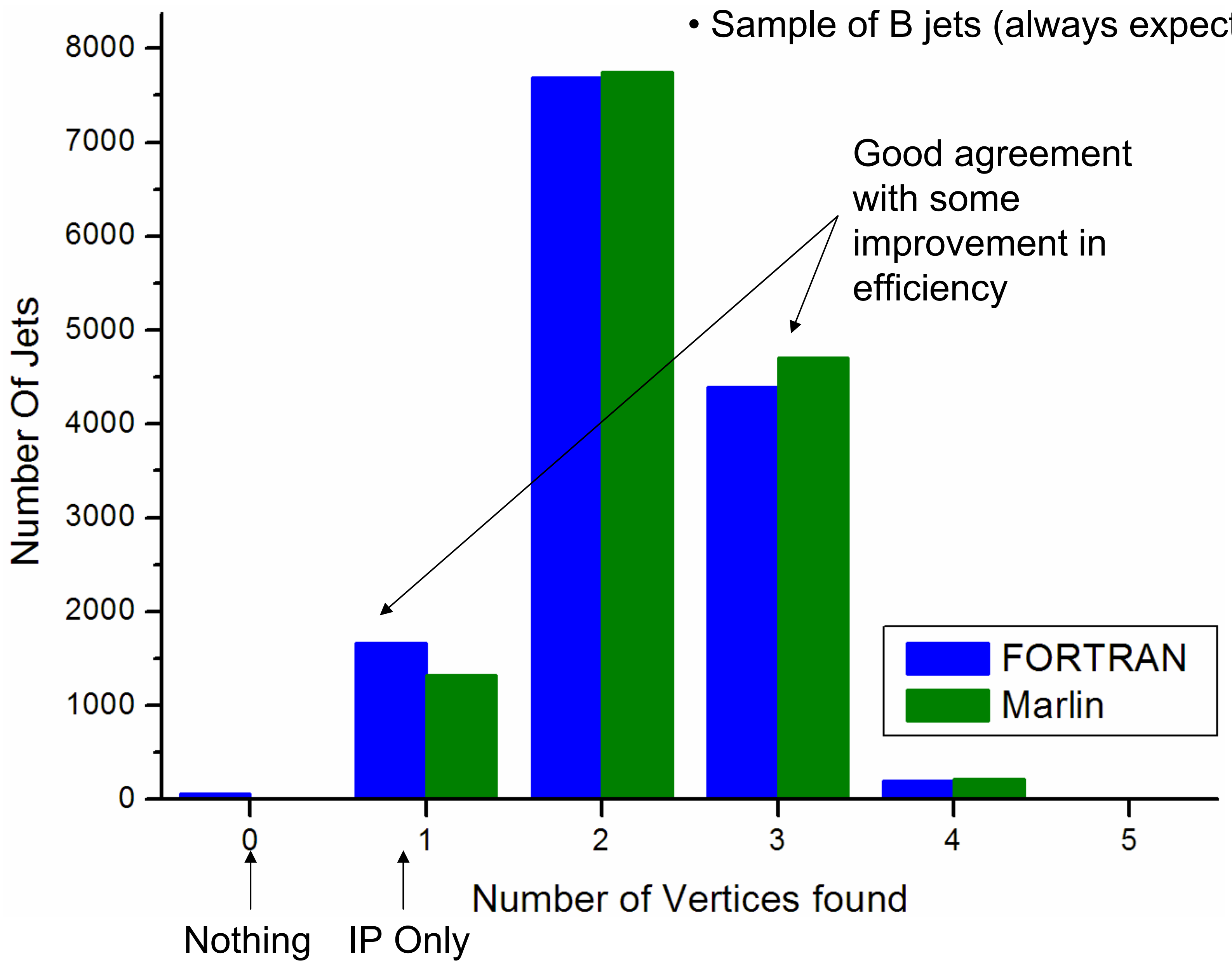
## *Current testing setup*



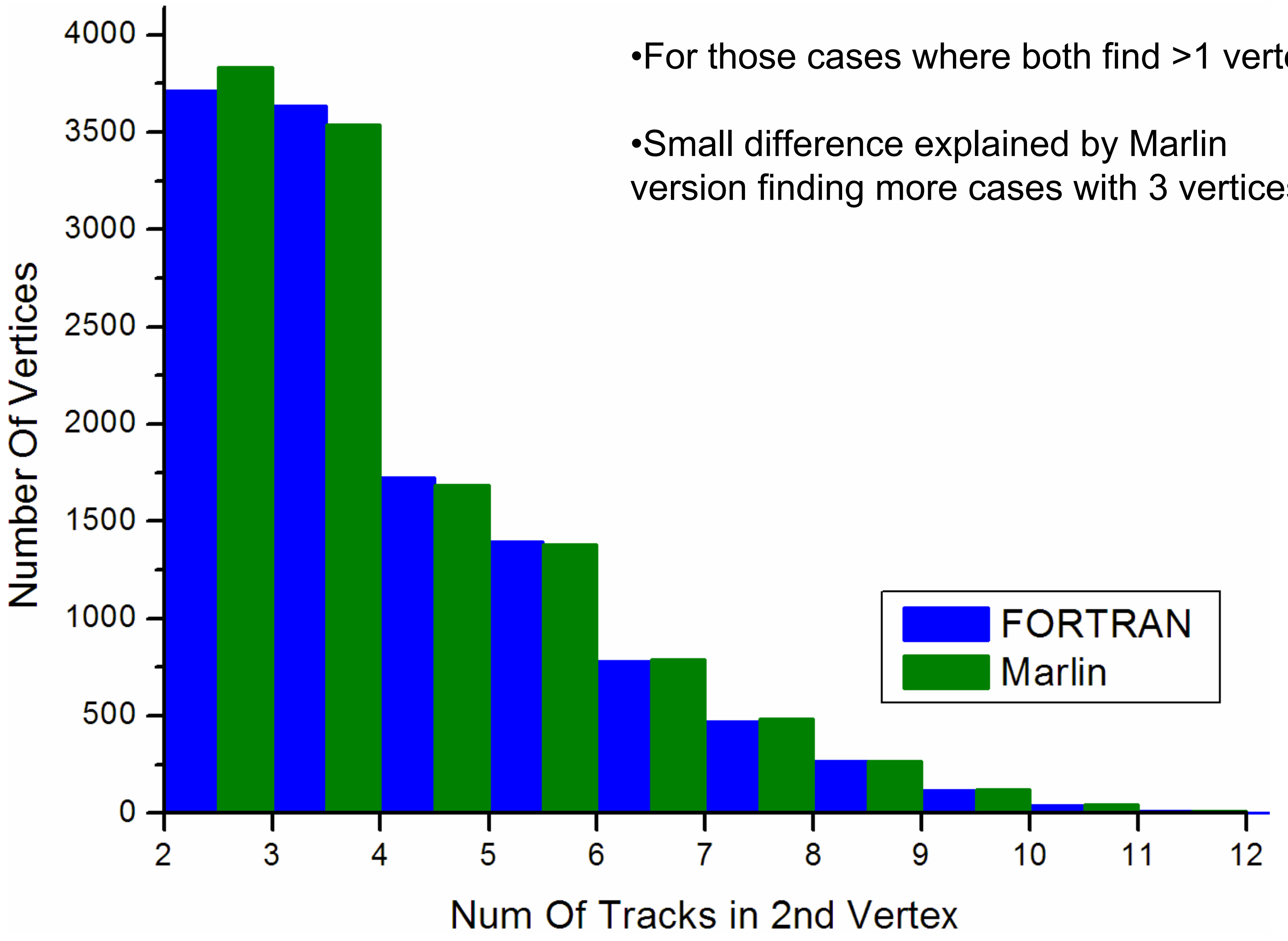
- FORTAN runs within (and is very highly coupled to) SGV fast Monte Carlo
- New rudimentary LCIO output code in SGV produces reconstructed tracks and results of FORTRAN ZVRES, flavour tag and vertex charge
- Marlin/LCIO framework runs new C++ ZVTOP, flavour tag and vertex charge
- Allows direct comparison running on identical reconstructed tracks
- No comparison available for ZVKIN – not in SGV version of ZVTOP
- Run on jets from 200GeV centre of mass as default

## *ZVRES – Comparison with FORTRAN*

- Sample of B jets (always expect secondary)



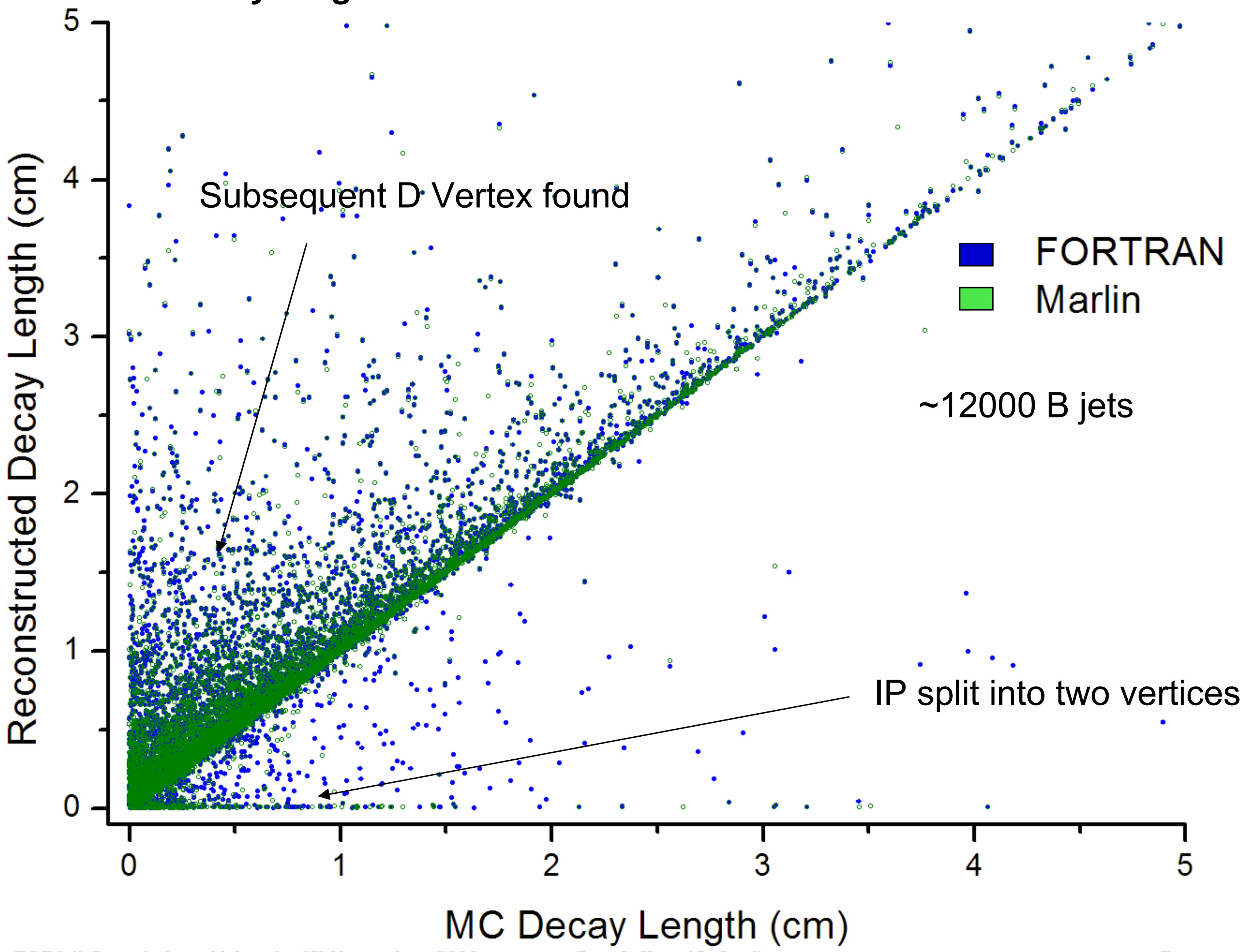
## *ZVRES – Comparison with FORTRAN*



•For those cases where both find >1 vertex

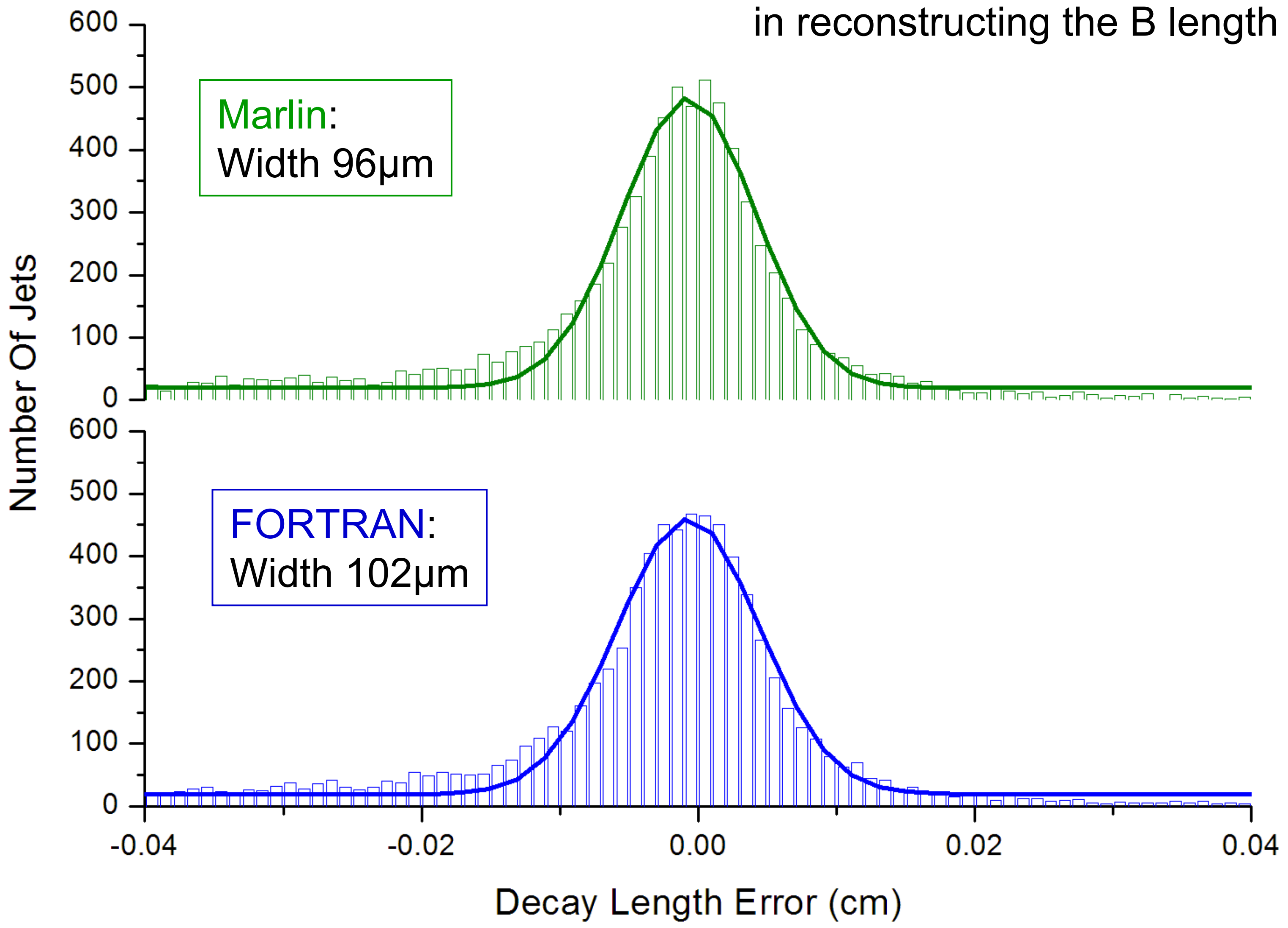
•Small difference explained by Marlin version finding more cases with 3 vertices

## *ZVRES – B Decay length reconstruction*

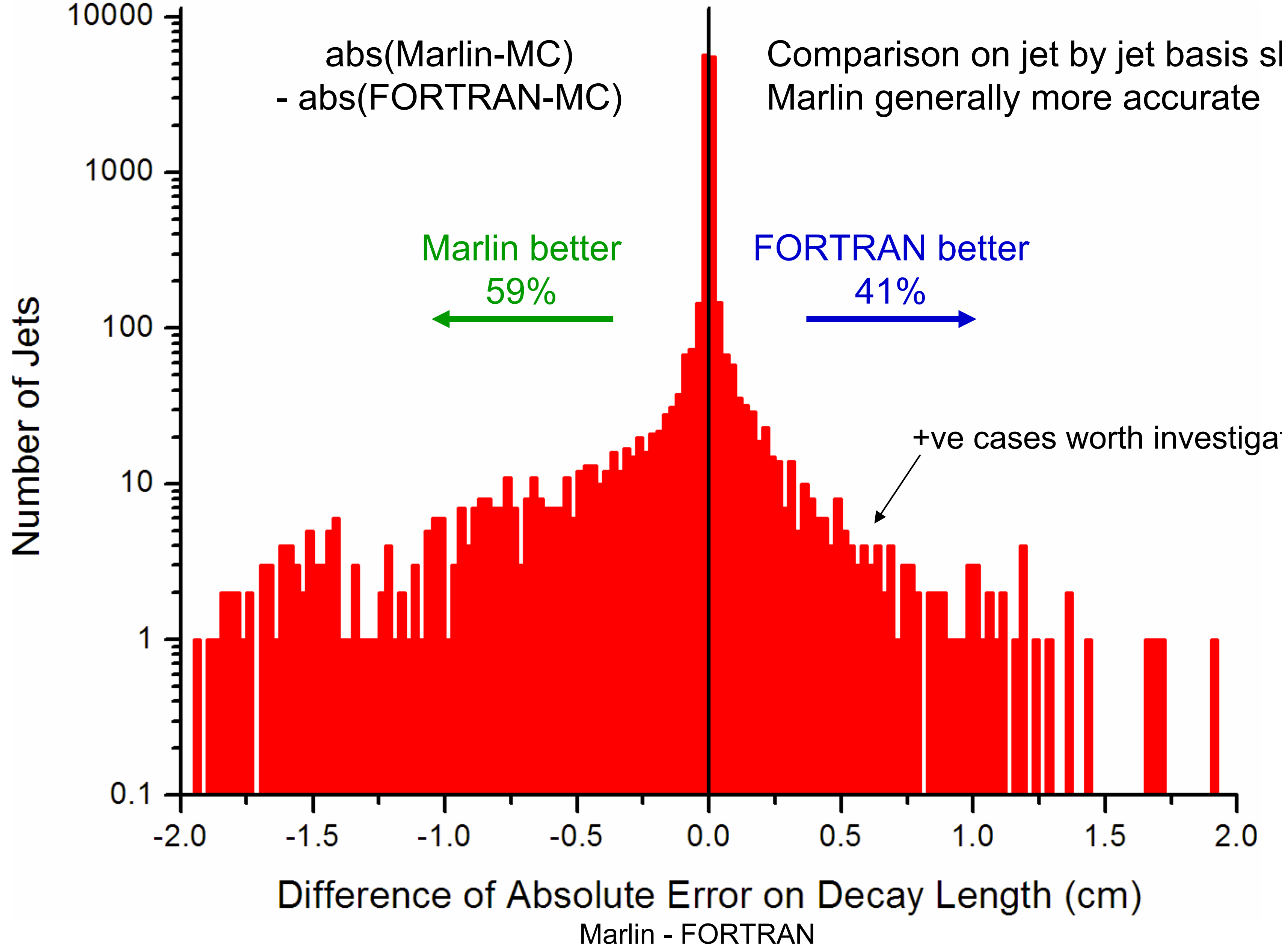


## *ZVRES – Decay length reconstruction*

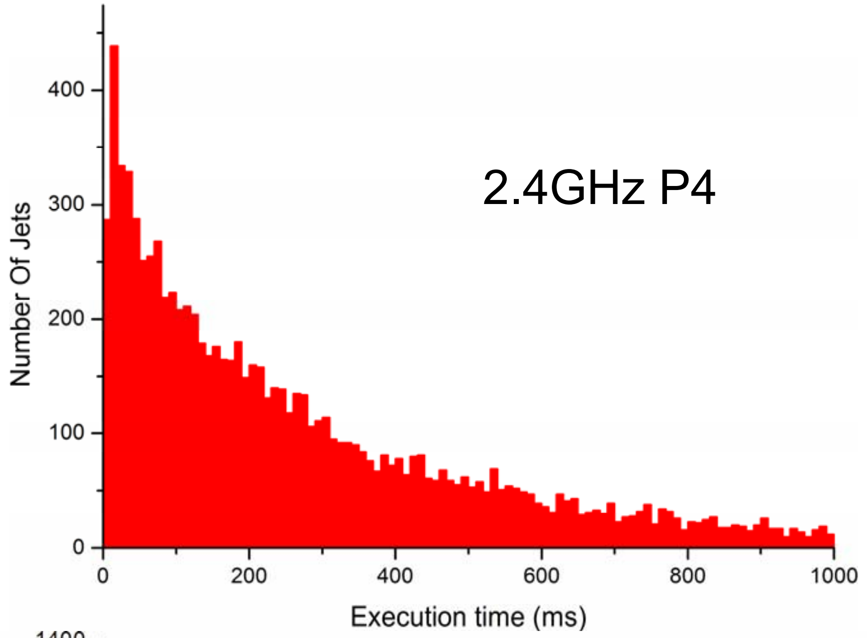
•Central peak shows similar performance in reconstructing the B length

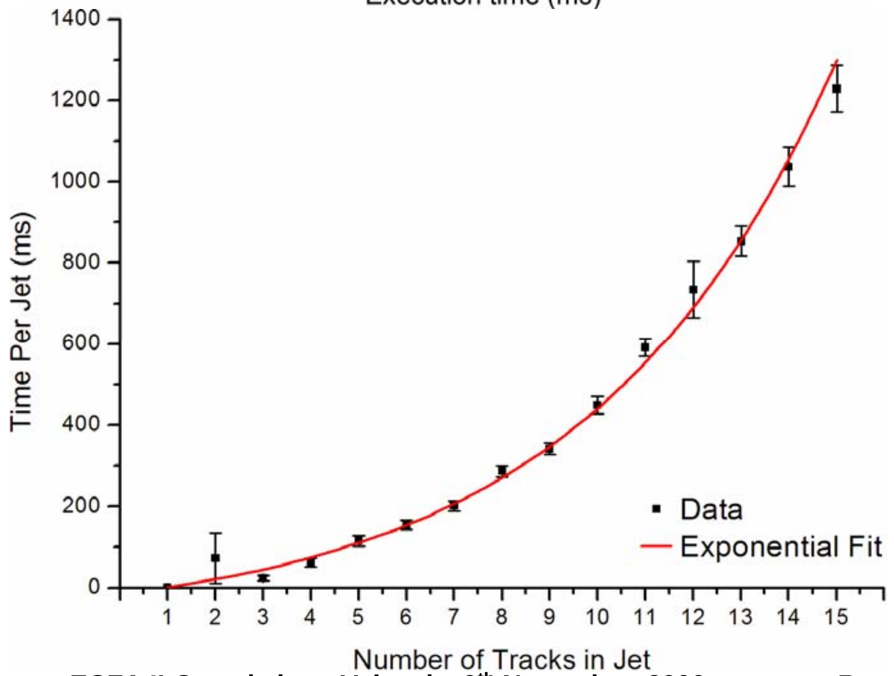


## *ZVRES – Decay Lengh Reconstruction*



## *ZVRES – Performance*





•Reasonable runtime performance achieved after profiler led optimisation.
•Factor ~20 slower than FORTRAN

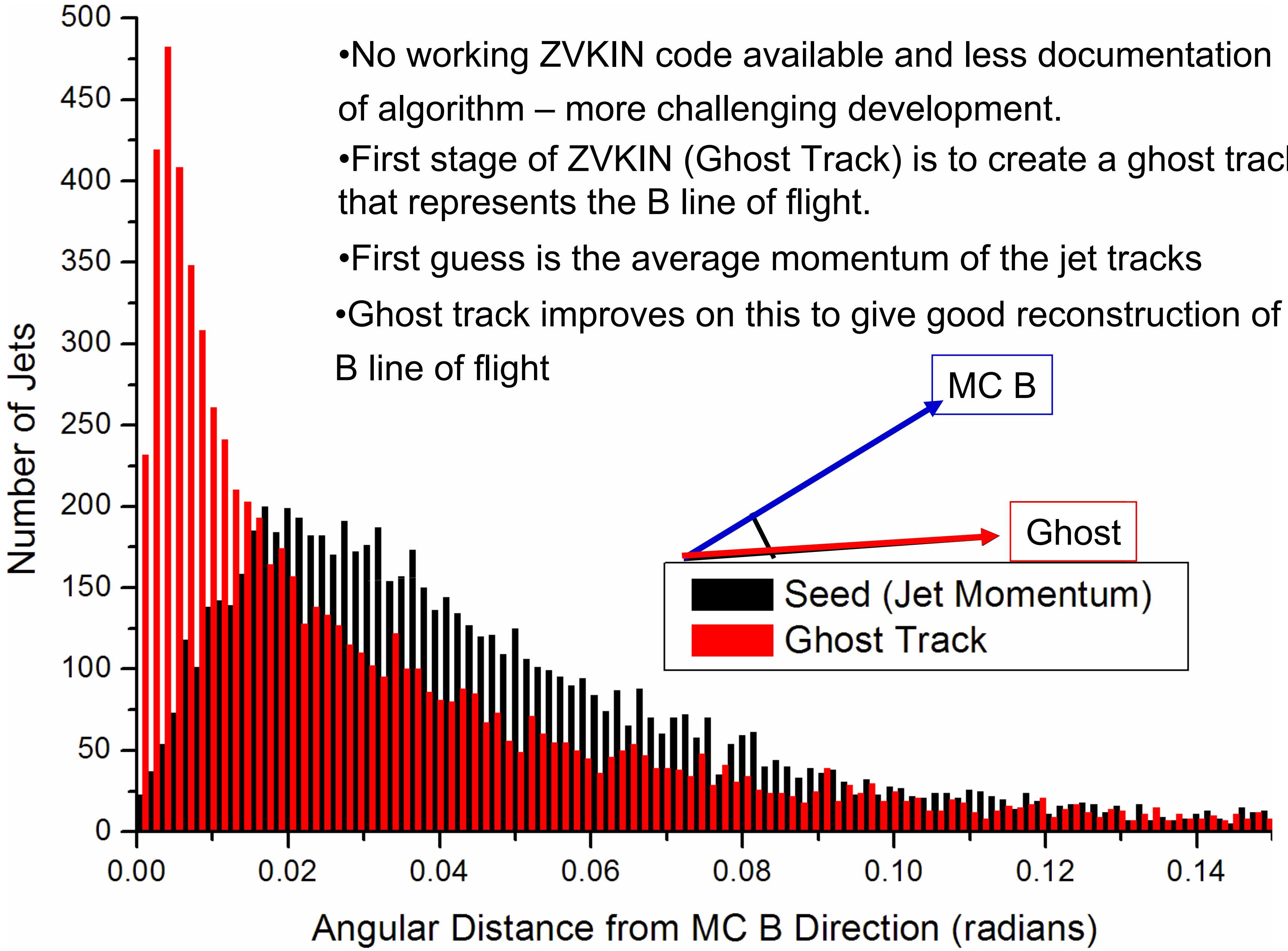
•Vertex fitter limited (fitter upgrade possible)

•Exponential in number of tracks

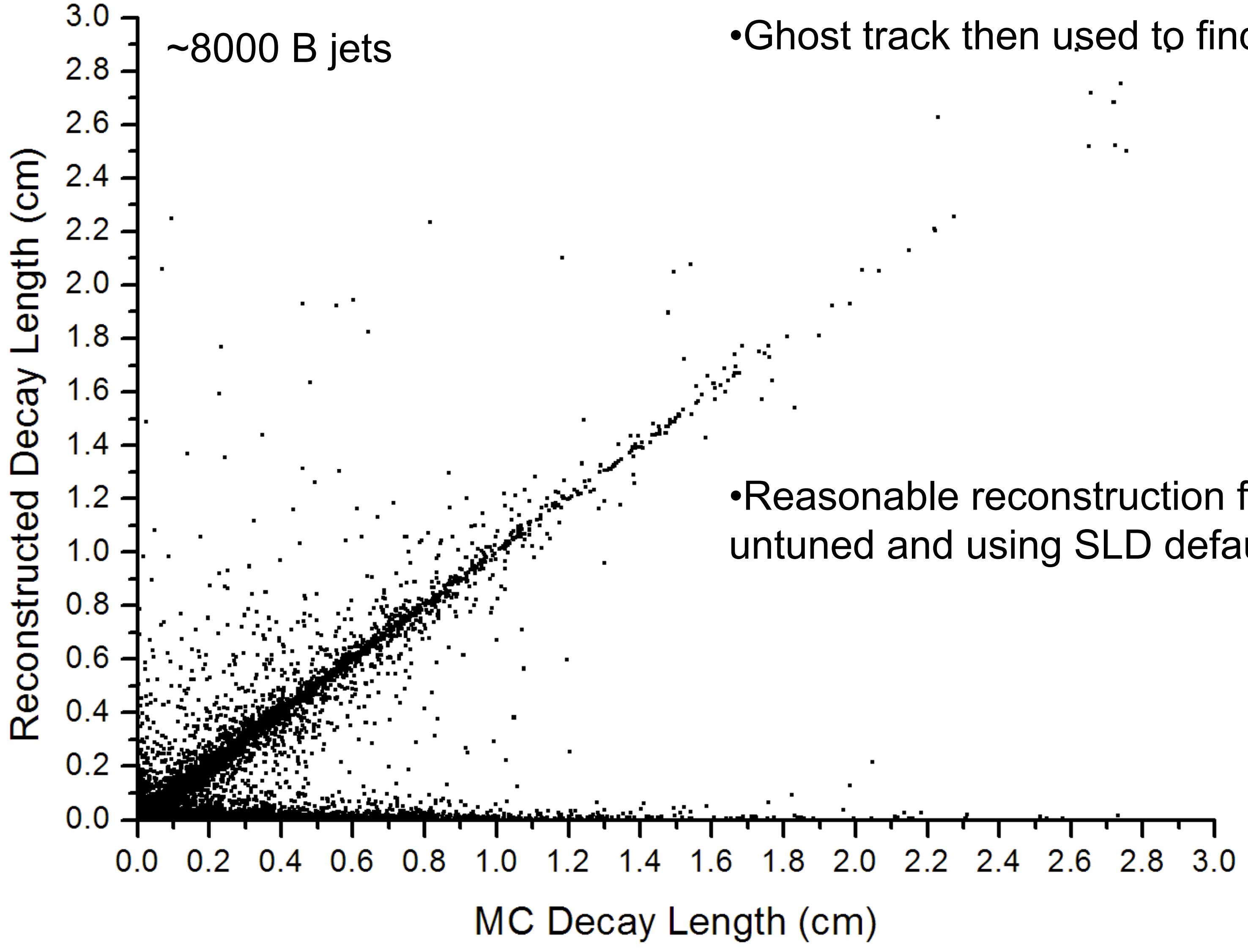
•Tests run in a virtual machine framework (Valgrind) confirm no memory leaks, double frees etc.

## *ZVKIN – Ghost track finding*

- No working ZVKIN code available and less documentation of algorithm – more challenging development.
- First stage of ZVKIN (Ghost Track) is to create a ghost track that represents the B line of flight.
- First guess is the average momentum of the jet tracks
- Ghost track improves on this to give good reconstruction of B line of flight

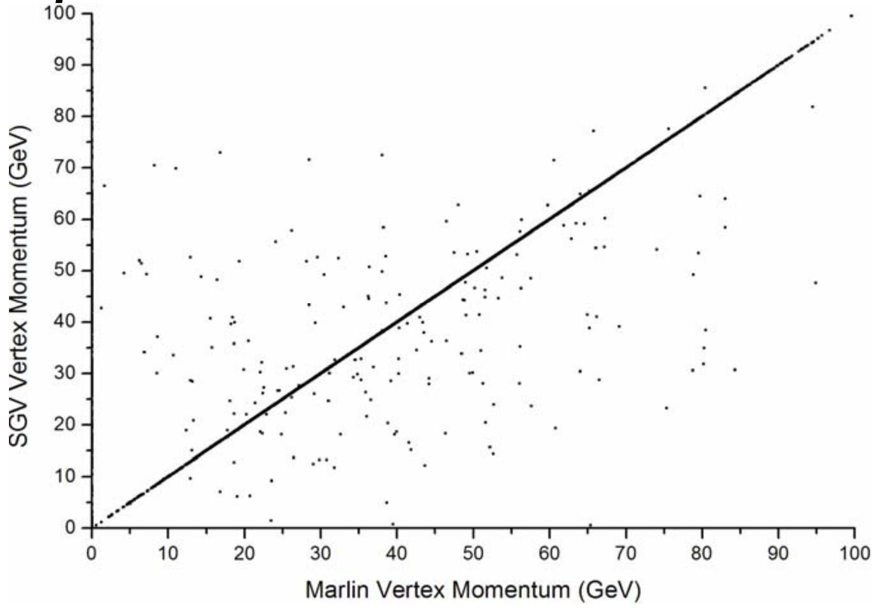


## *ZVKIN – Preliminary Performance*



- Ghost track then used to find vertices
- Reasonable reconstruction for first go – untuned and using SLD defaults

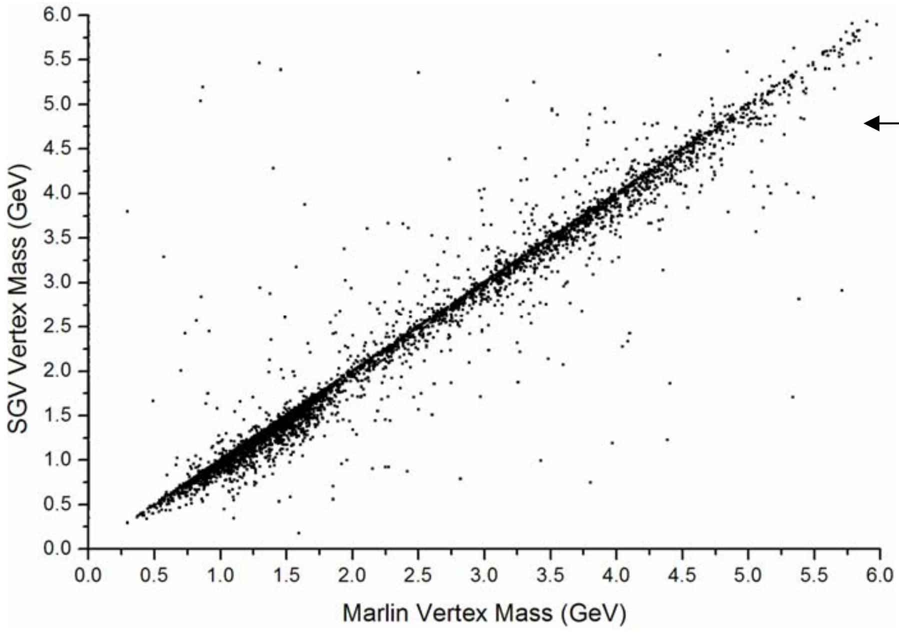
## *Inputs to Neural Network*



2 of the 9 different inputs:
20000 u,d,s,c & b jets

**Secondary vertex momentum**

Scattered points due to different vertexing results

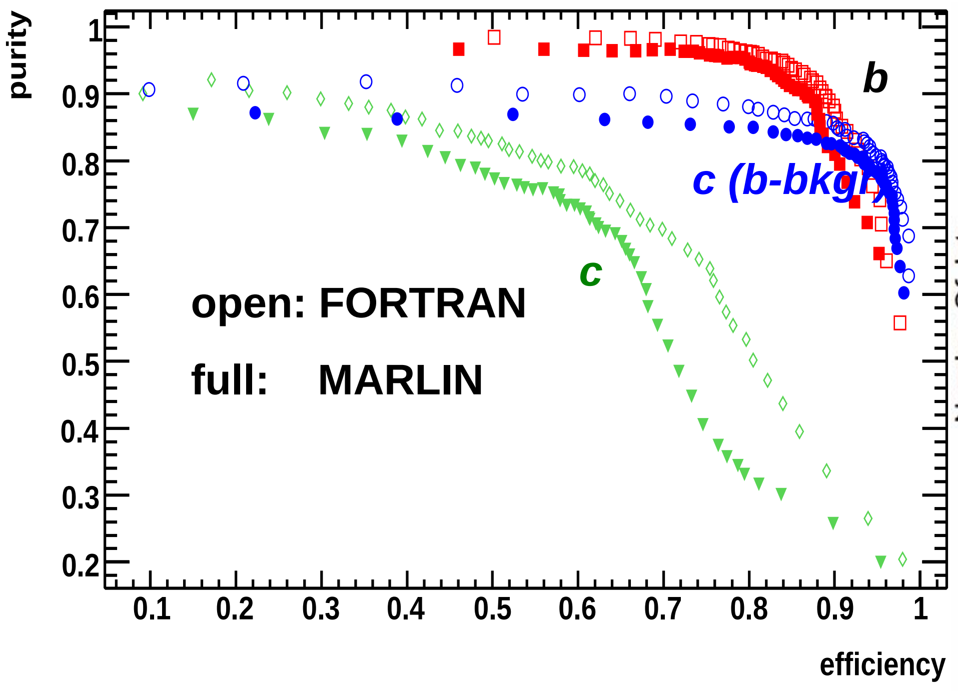


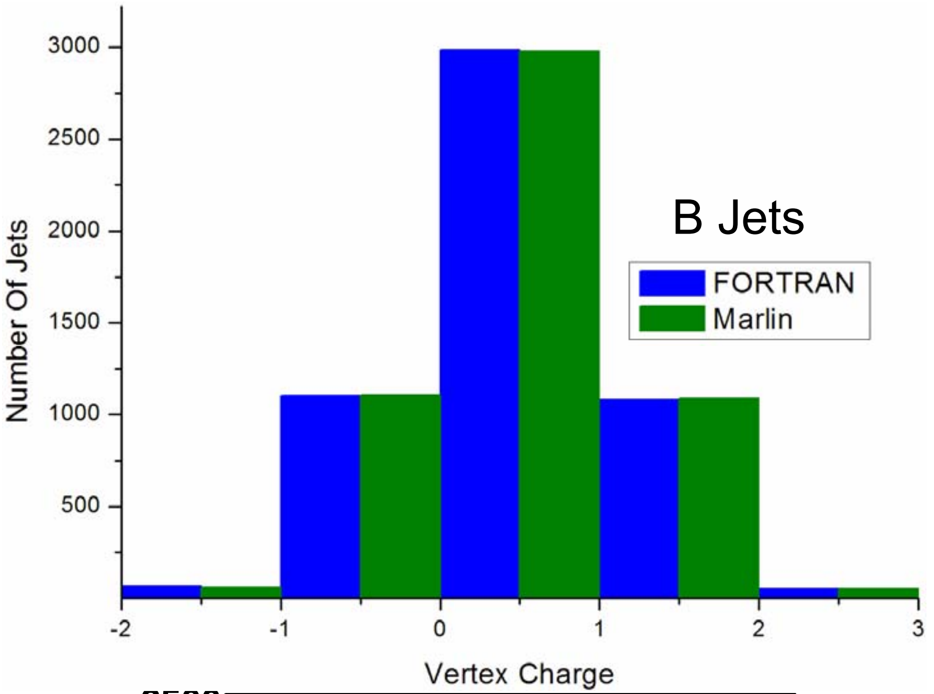
**Secondary vertex pt corrected mass**

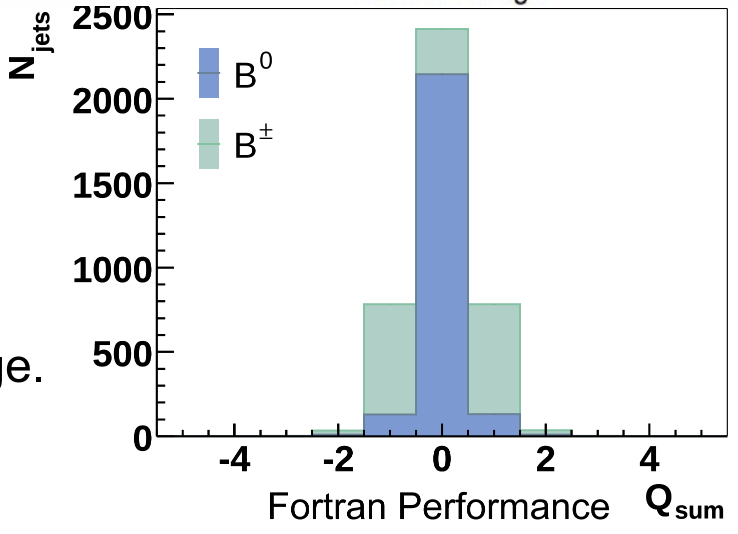
Extra differences here under investigation
- differing error propagation
- possible problem with FORTRAN?

Other inputs show good agreement
± undocumented features of FORTRAN being analyzed, documented and added.

## *Flavour tag and vertex charge*







Performance of the whole package working together approaching that of FORTRAN.

Good agreement for vertex charge.

## *Summary of Testing*

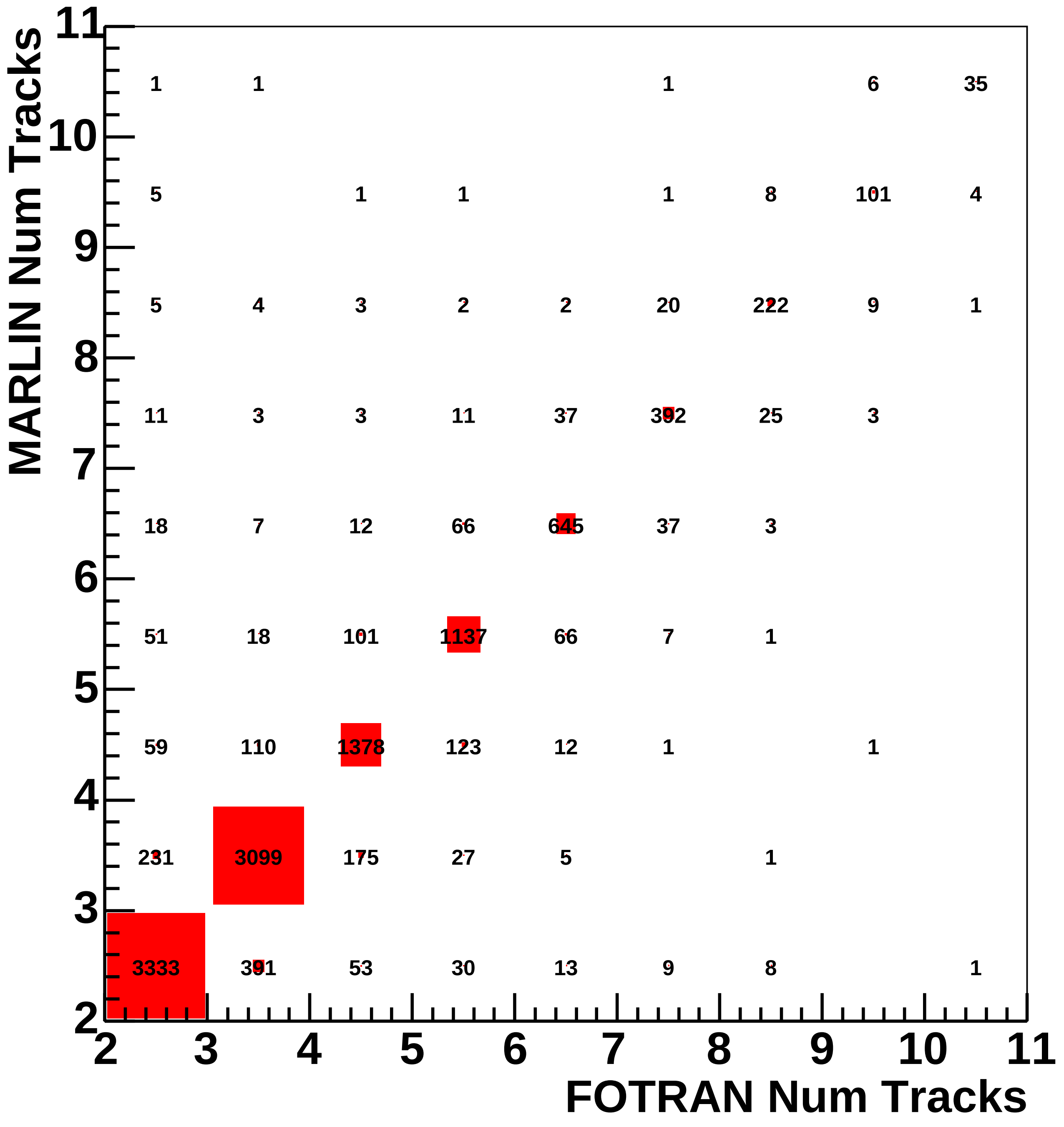
•All parts of package performing well in comparison with previous tools
± small details being fixed now.

•Next steps to replace SGV input with MarlinReco and MOKKA
$\longrightarrow$ (still LCIO but technical differences)
should enable easier analysis of performance vs MC
for comprehensive study and tuning after release.

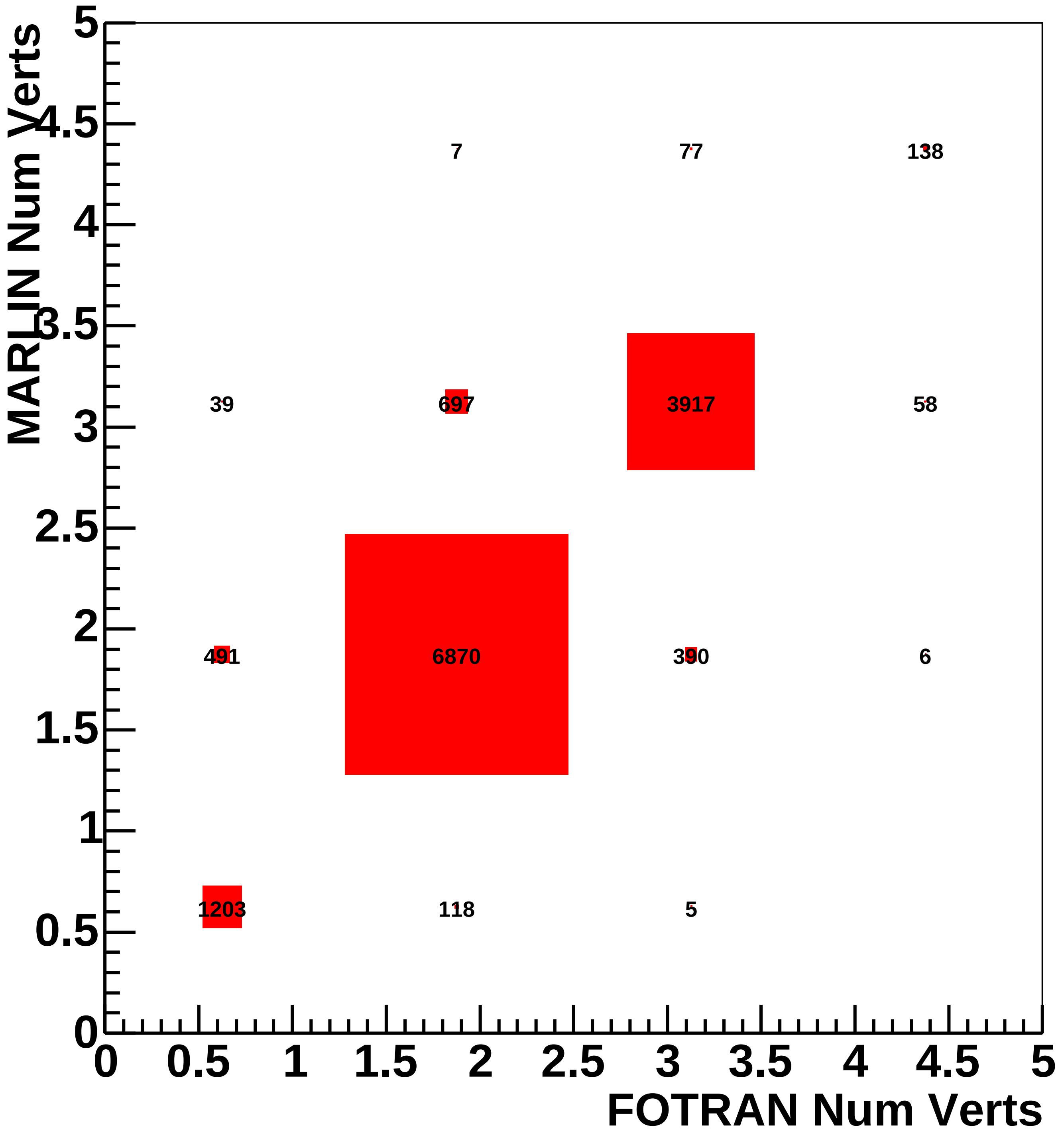
•Test plot codes will be released with future upgrade to enable end user checks.

$\longrightarrow$ **Package close to reaching convergence**

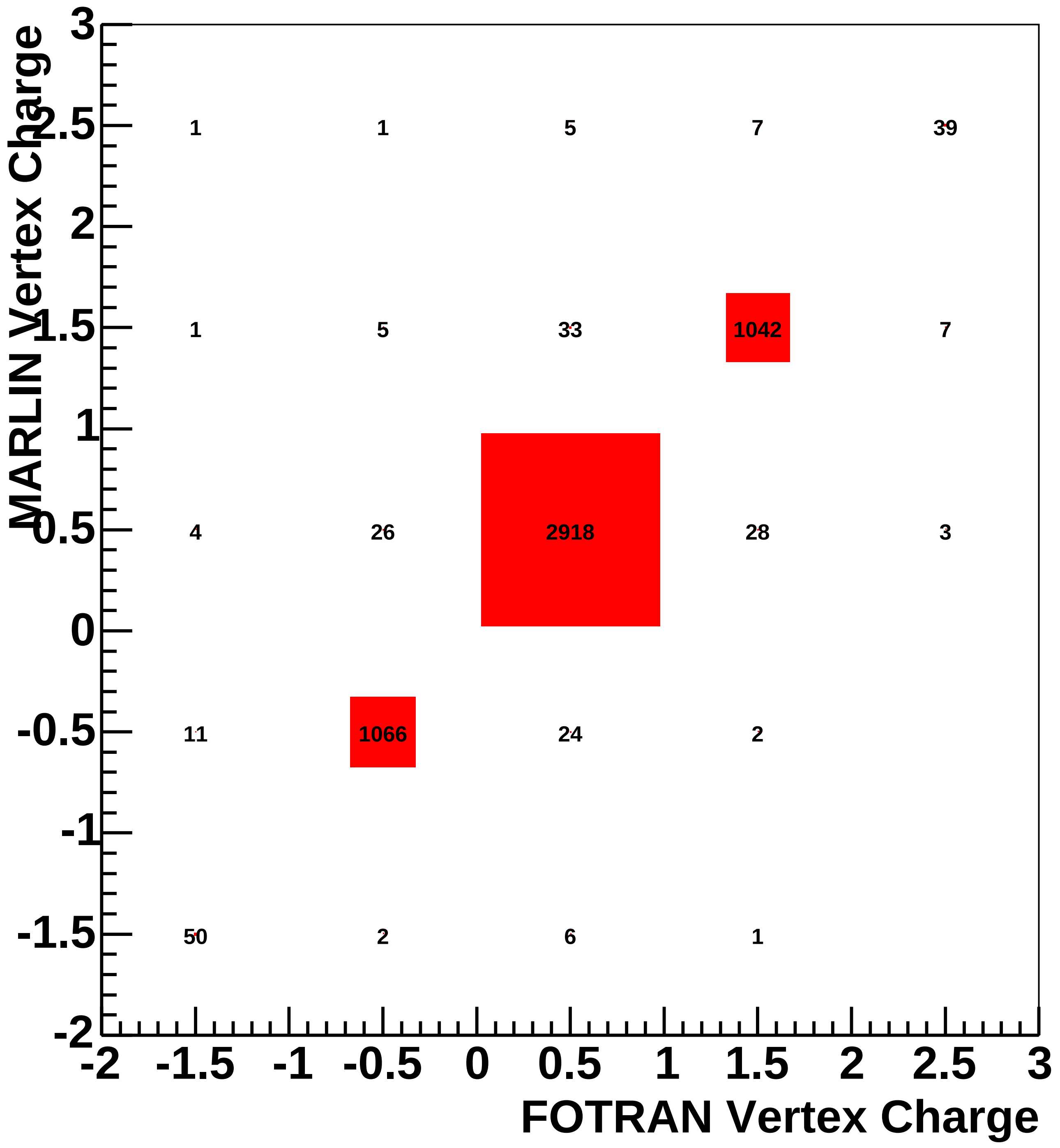
## Extra Slides



MARLIN Num Tracks
FOTRAN Num Tracks
2
3
4
5
6
7
8
9
10
11
1 1 1 6 35
5 1 1 1 8 101 4
5 4 3 2 2 20 222 9 1
11 3 3 11 37 392 25 3
18 7 12 66 645 37 3
51 18 101 1137 66 7 1
59 110 1378 123 12 1 1
231 3099 175 27 5 1
3333 391 53 30 13 9 8 1


MARLIN Num Verts
FOTRAN Num Verts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
7
77
138
39
697
3917
58
491
6870
390
6
1203
118
5

ECFA

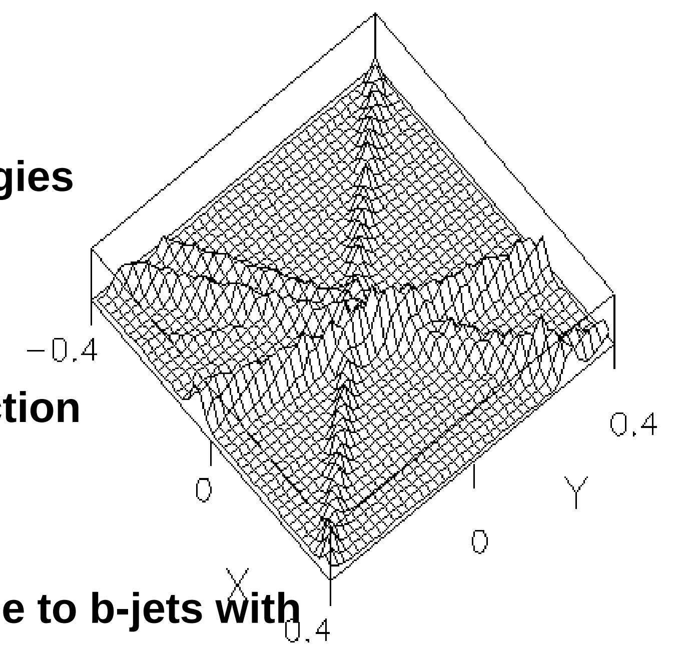

MARLIN Vertex Charge
FOTRAN Vertex Charge
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
1
1
5
7
39
1
5
33
1042
7
4
26
2918
28
3
11
1066
24
2
50
2
6
1

## The ZVTOP vertex finder

*D. Jackson,*

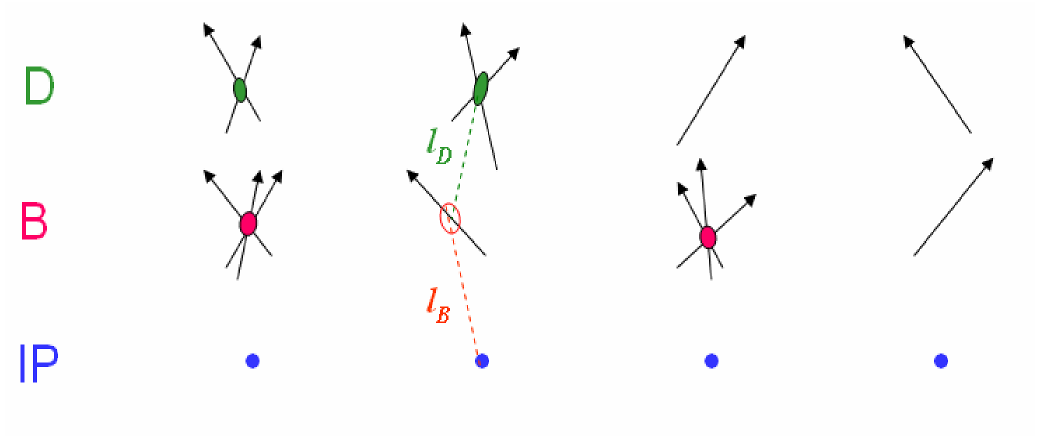
*NIM A 388 (1997) 247*

- two branches: ZVRES and ZVKIN (also known as ghost track algorithm)



- **The ZVRES algorithm:** very general algorithm that can cope with arbitrary multi-prong decay topologies
  - 'vertex function' calculated from Gaussian ´probability tubes´ representing tracks
  - iteratively search 3D-space for maxima of this function and minimise $\chi^2$ of vertex fit

- **ZVKIN:** more specialised algorithm to extend coverage to b-jets with 1-pronged vertices and / or a short-lived B-hadron not resolved from the IP



- additional kinematic information (IP-, B-, D-decay vertex approximately lie on a straight line) used to find vertices
- should improve flavour tag efficiency and determination of vertex charge

# The ZVTOP vertex finder

*D. Jackson,*

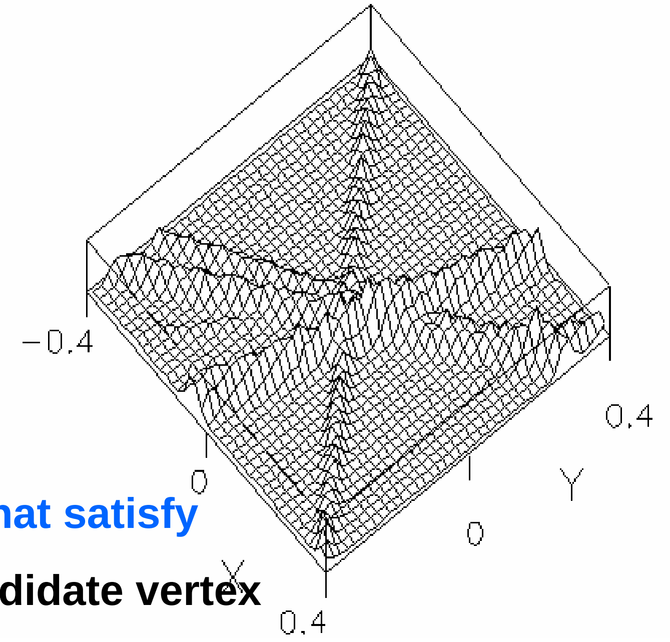
*NIM A 388 (1997) 247*

two branches: ZVRES and ZVKIN (also known as ghost track algorithm)

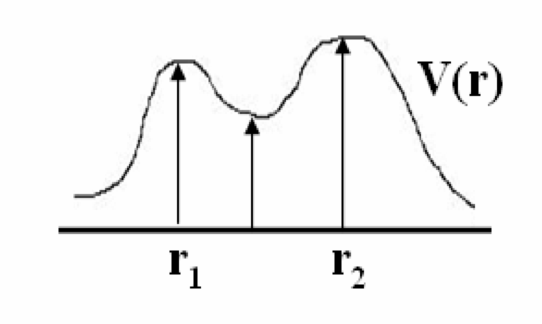
## The ZVRES algorithm:

- tracks approximated as Gaussian ´probability tubes´
- from these, a ´vertex function´ is obtained:

$$V(\mathbf{r}) = \sum_{i=0}^{N} f_i(\mathbf{r}) - \frac{\sum_{i=0}^{N} f_i^2(\mathbf{r})}{\sum_{i=0}^{N} f_i(\mathbf{r})}$$



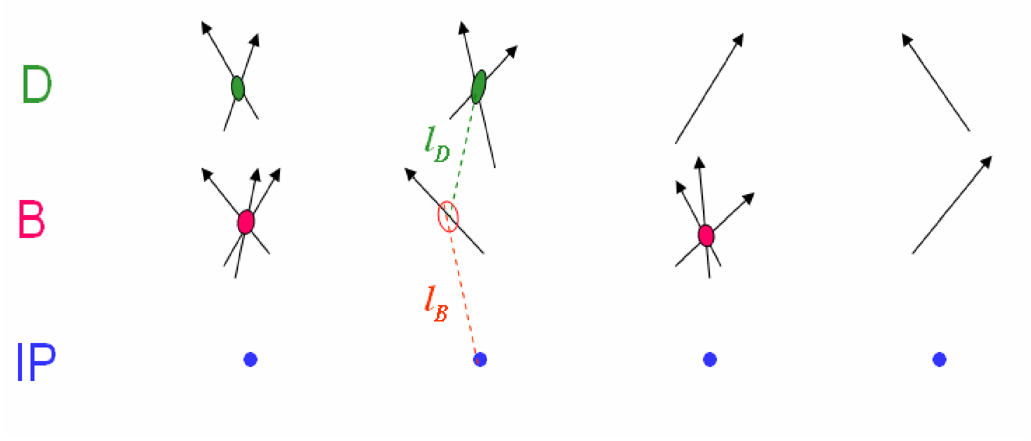
- 3D-space searched for maxima in the vertex function that satisfy resolubility criterion; track can be contained in > 1 candidate vertex
- iterative cuts on $\chi^2$ of vertex fit and maximisation of vertex function results in unambiguous assignment of tracks to vert

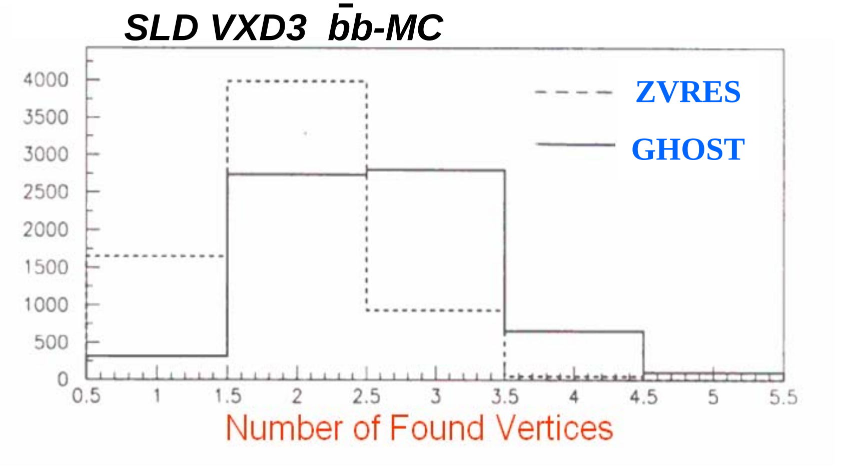


- has been shown to work in various environments differing in energy range, detectors used and physics extracted
- very general algorithm that can cope with arbitrary multi-prong decay topologie

# The ZVKIN (ghost track) algorithm

- more specialised algorithm to extend coverage to b-jets in which one or both secondary and tertiary vertex are 1-pronged and / or in which the B is very short-lived;
- algorithm relies on the fact that IP, B- and D-decay vertex lie on an approximately straight line due to the boost of the B hadron



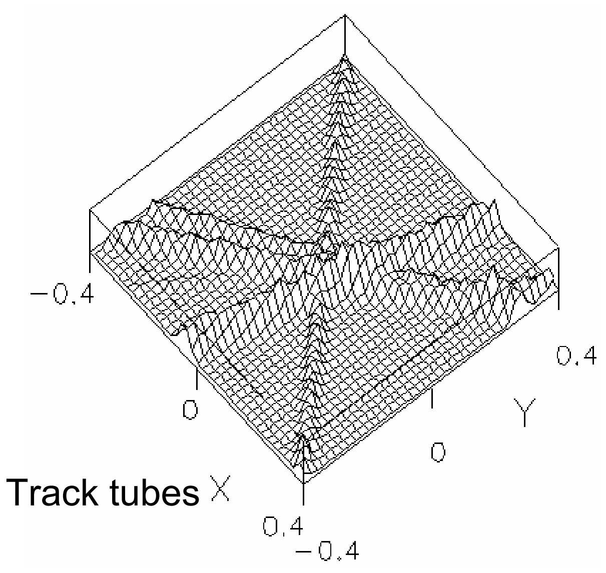


- should improve flavour tagging capabilities

## ZVRES - Introduction

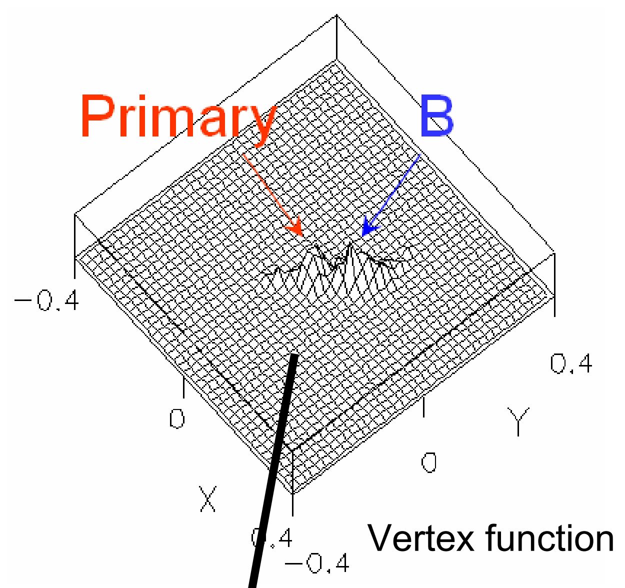
Basic Idea: Tracks represented by Gaussian error 'tubes'

Tubes combined to give vertex function:

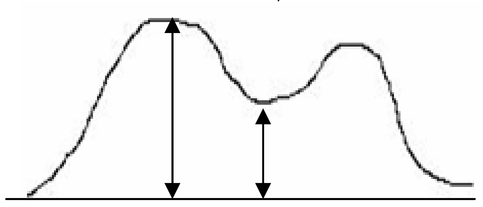


Track tubes

$$\sum_{i=1}^{n} Tube_i - \frac{\sum_{i=1}^{n} Tube_i^2}{\sum_{i=1}^{n} Tube_i}$$



Vertex function



Vertex function peaks resolved into distinct vertices by cut on peak-valley ratio.

Remaining ambiguities in track assignment resolved by magnitude of vertex function.

## ZVTOP - Motivation

Verified reference implementation needed in the ILC software framework

$\longrightarrow$ Object oriented C++

Existing FORTRAN code used at SLD exists, and has been used, in fast simulation SGV. (Having come with some modifications via OPAL)

Was a possibility just to wrap this with C++, BUT:

- •ZVKIN not included - needed for vertex charge tagging.
- •Minimal documentation.
- •Difficulty of additions or changes:
  - •ILC boundary conditions - all scale dependant parts needed updating.

## Approach

Design and code from the original ZVTOP paper.

- •Complete understanding and documentation.
- •Direct rewrite would not be object oriented.
- •Identifies undocumented parts of the FORTRAN by comparison.

## ZVTOP - Changes

Some approximations in FORTRAN removed for C++:

Tubes: FORTRAN has parabolic approximation with only diagonal error matrix terms.

$$\text{Tube} = f_i(\mathbf{r}) = \exp\left\{-\frac{1}{2}\left[\left(\frac{x' - (x'_0 + \kappa y'^2)}{\sigma_T}\right)^2 + \left(\frac{z - (z_0 + \tan(\lambda) y')}{\sigma_L}\right)^2\right]\right\}$$

C++ uses helix and full error:

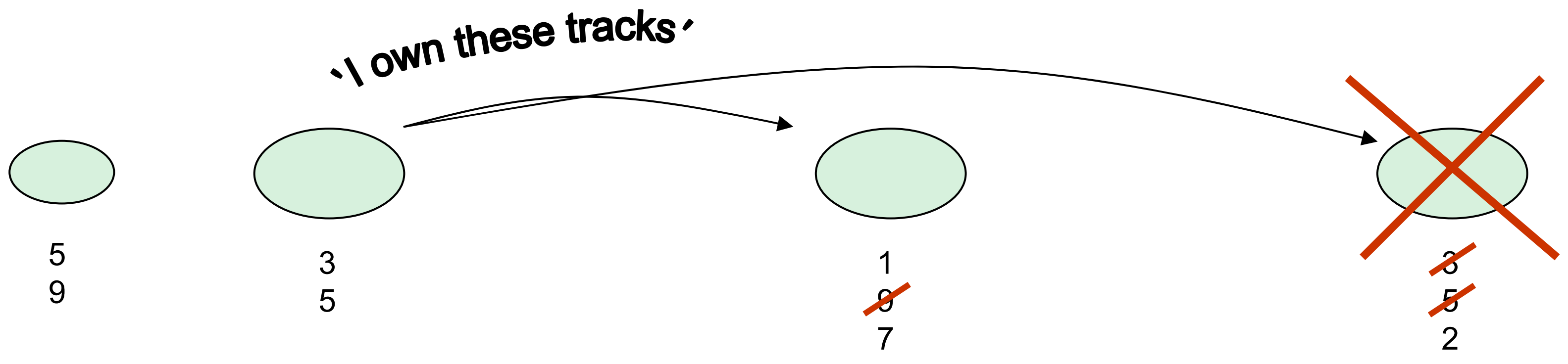
$$\text{Tube} = f_i(\underline{r}) = e^{(-\frac{1}{2}\underline{p}\,\underline{\underline{V}}^{-1}\,\underline{p}^{T})}$$

p = Residual to track
V = Covariance Matrix

Track-Interaction Point and two track fitting changed from analytic approximation to full fit.

Algorithm structure changed for object orientation:

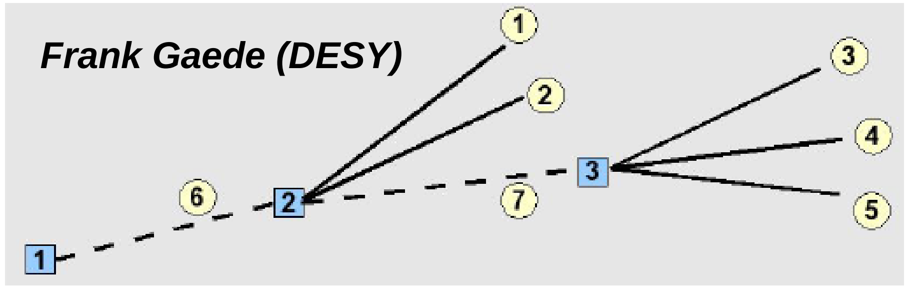
Based around idea of candidate vertices – Merging, track removal etc.
Gives flexibility and can be reused for ZVKIN.

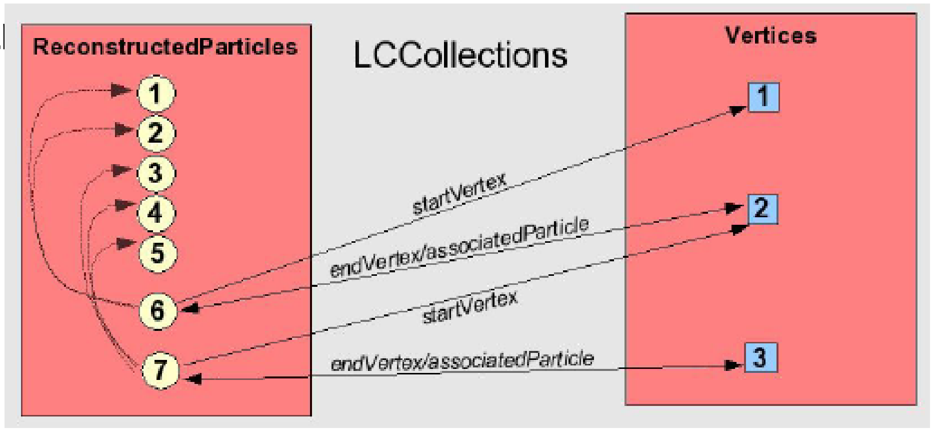


Modular – should allow for change of vertex fitters etc
Current fitter thanks to Mark Grimes at Bristol

# Interfacing the Vertex Package

➢ LCIO persistency framework has been extended by dedicated vertex class to accommodate the output of our softwa





- each ReconstructedParticle points to one vertex from which it originated & to decay ve

➢ will provide MARLIN processors (modules) giving example code for

- running ZVTOP (one processor for each of the two branches ZVRES, ZVKIN)
- calculating neural net input variables from input to package & ZVTOP output
- training neural nets for flavour tag, obtaining NN outputs, determine purity vs efficienc
- vertex charge calculation
- combined processor: ZVRES + Hawkings flavour tag + vertex charge calculation

## Current status

- performance of ZVRES branch has been shown to be at least as good as FORTRAN in detailed tests of increasing complexity (Ben Jeffery, Mark Grimes)

- ZVKIN branch implemented, first tests successful (Ben Jeffery)

- calculation of flavour tag inputs coded (C++) and tested within SGV (Erik Devetak)

- designed & implemented a set of internal 'working classes' linking ZVTOP with the other parts of the package (Ben Jeffery)

- code ported into MARLIN framework;
  MARLIN processors providing examples how to use our code implemented,
  'full chain test' (ZVRES, tag, vertex charge) with SGV event reconstruction beginning, initial results promising (BJ, MG, ED, SH)

- work on LCIO interface ongoing;
  storage of output in LCIO implemented using the new Vertex class (Ben Jeffery)

## Strategy for validating the code

➢ **Tests using SGV event reconstruction**

**permits direct comparisons with results from FORTRAN version using identical input ever**

- **standalone test of ZVRES, input / output directly from / to SGV common blocks** ✓

- **separate tests of Marlin processors for ZVRES, ZVKIN, flavour tag input calculation** ✓

  **FORTRAN-LCIO interface used to write out lcio file from SGV, read in by Marlin proces**

  **and used to feed values into internal working classes of our package**

  **results from those tests: Ben Jeffery's talk in this session**

- **full-chain test of ZVRES + flavour tag + vertex charge using same setup** (✓)

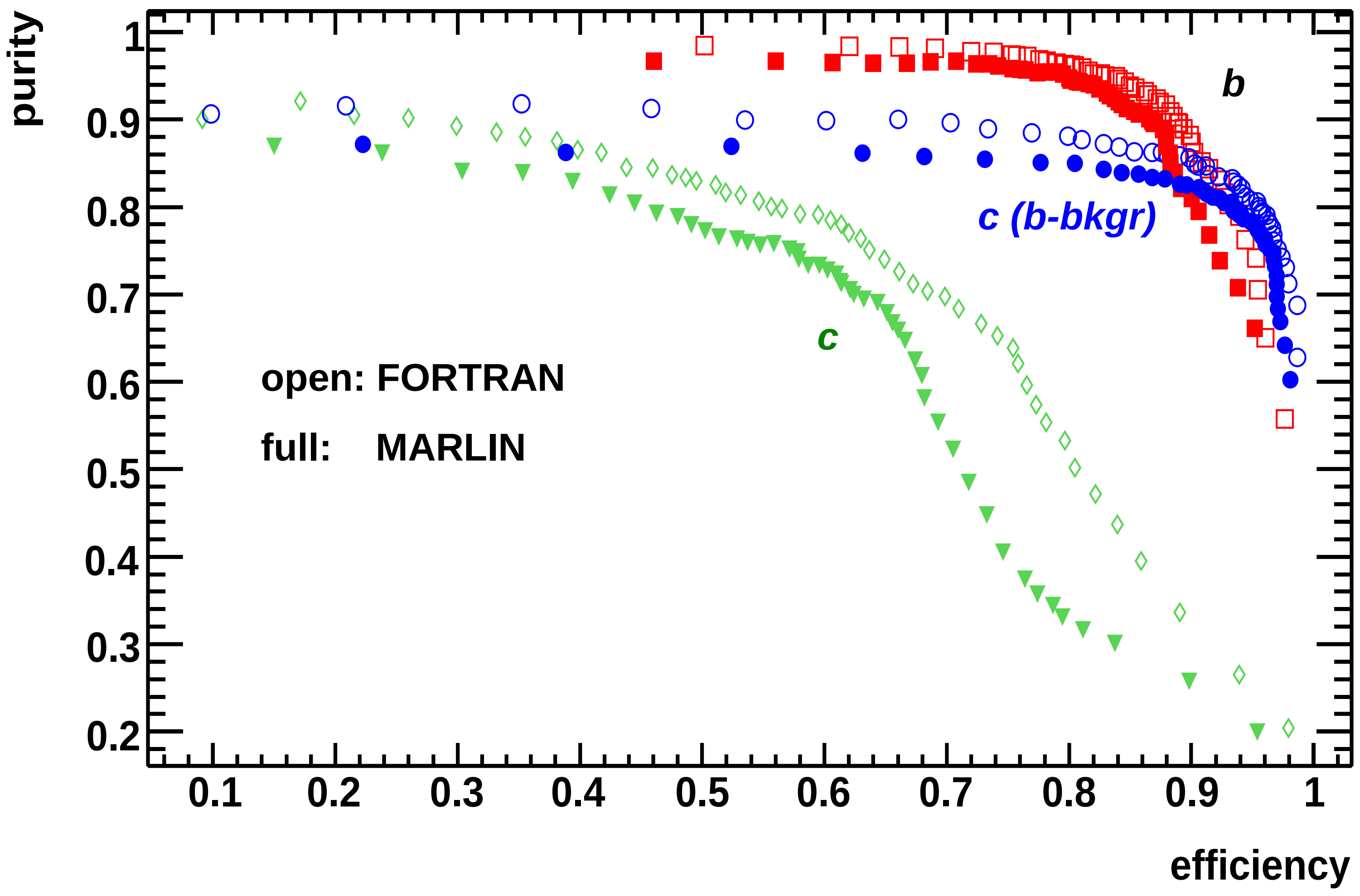
  **convert Marlin output to root & use analysis software previously developed for FORTRAN setup**

➢ **Tests using MarlinReco event reconstruction**

- **once interface from MarlinReco to our working classes is in place, will repeat full chain**

# Test of Marlin-ZVRES + Marlin flavour tag

➢ Comparison of MARLIN and FORTRAN at the Z-peak, identical input events



➢ good result for a first attempt, differences to be looked into in more detail

## Areas needing further work

- ➢ interfacing to event-input from MarlinReco-based event reconstruction

  (for initial tests will only use track cheaters)

- ➢ make code more robust by including handling of bad user input and other erro

- ➢ system test of full chain (ZVRES + flavour tag + vertex charge)
  - run using SGV input needs to be understood
  - repeat tests using input from MarlinReco-based event reconstruction

- ➢ general usage documentation (independent class documentation mainly complete)

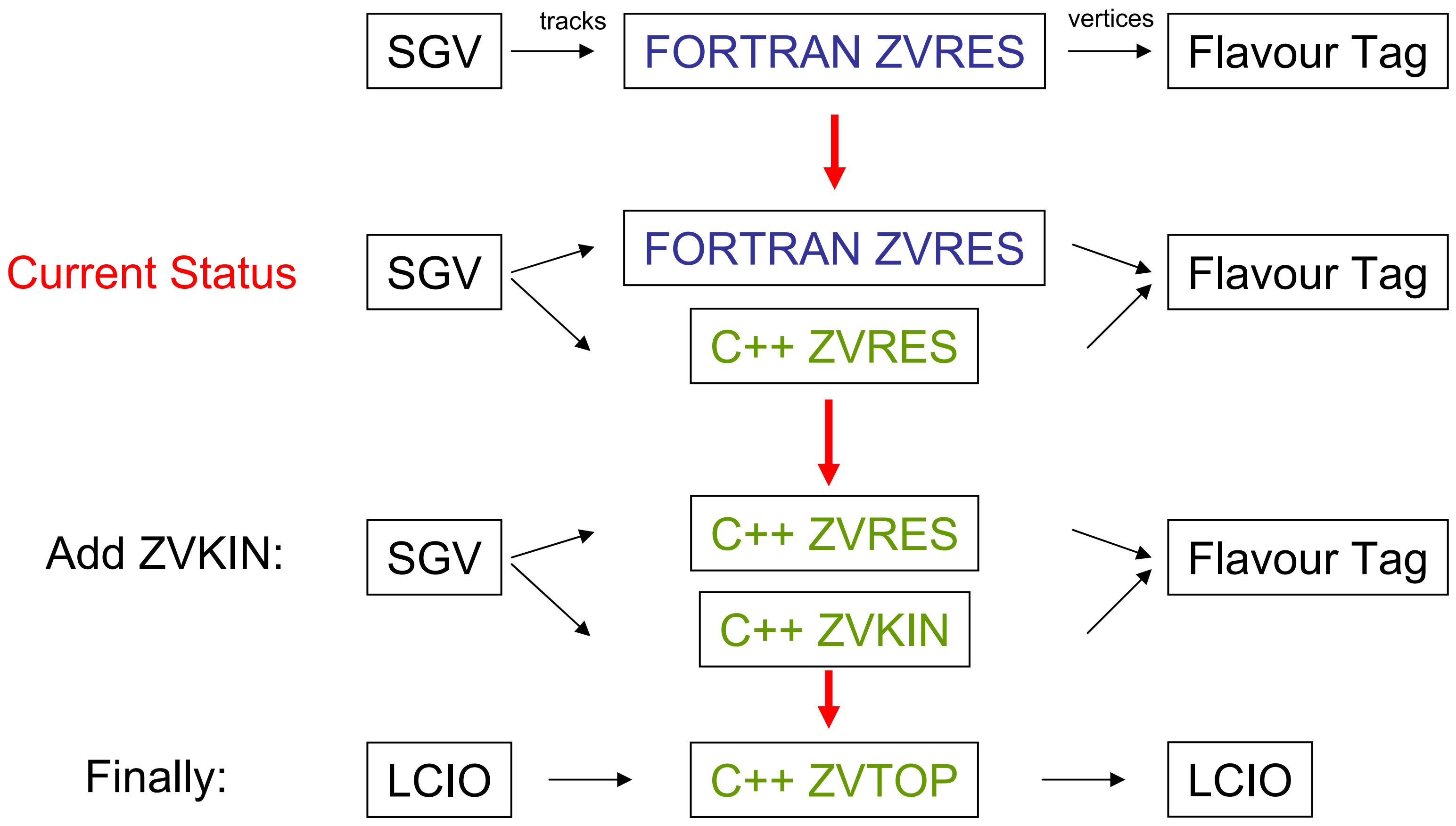
## Summary and outlook

- Development and validation of the LCFI Vertex Package are far advanced.
- A new Vertex class has been introduced into LCIO. Integration of our package in MarlinReco is in progress. Running code from JAS environment to be investigat to ensure interoperability of the reconstruction frameworks in this area (N Graf).
- Interfacing to event-input from MarlinReco event reconstruction needs further w
- First results from a full-chain run with SGV input are promising, but need to be understood further. A full-chain test with MarlinReco reconstruction will follow.
- The first release of the code is planned in a few weeks.
- Detailed comparisons with MarlinReco input and quantitative exploration of improvements from the ghost track algorithm will be the next steps after the rele

## ZVTOP - Progress

Initial aim: replace FORTRAN ZVRES in SGV for testing

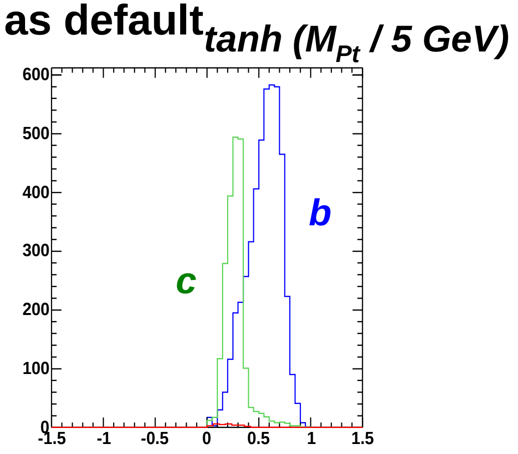
- allows comparison of intermediate algorithm states when working on identical tracks
- new version can be verified to be at least as good as FORTRAN

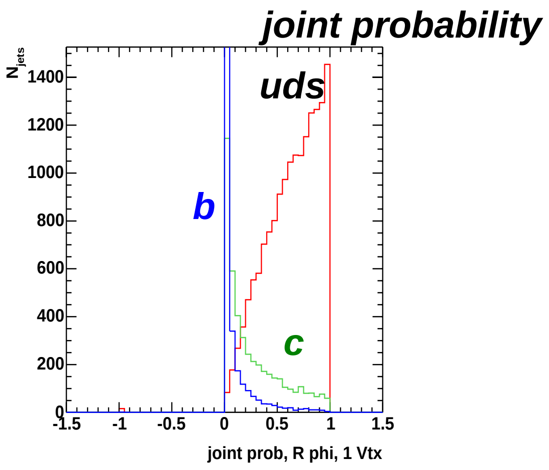


# Flavour tag

- **Vertex package will provide flavour tag procedure developed by R. Hawkings et**
  **(LC-PHSM-2000-021) and recently used by K. Desch / Th. Kuhl as default**

- **NN-input variables used:**
  - **if secondary vertex found: $M_{Pt}$ , momentum of secondary vertex, and its decay length and decay length significance**
  - **if only primary vertex found: momentum and impact parameter significance in R-$\phi$ and z for the two most-significant tracks in the jet**
  - **in both cases: joint probability in R-$\phi$ and z (estimator of probability for all tracks to originate from primary vertex)**



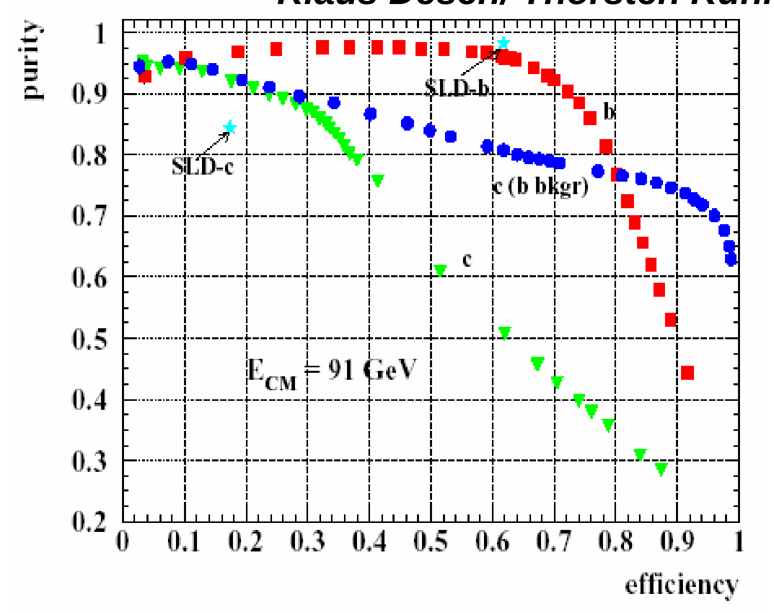


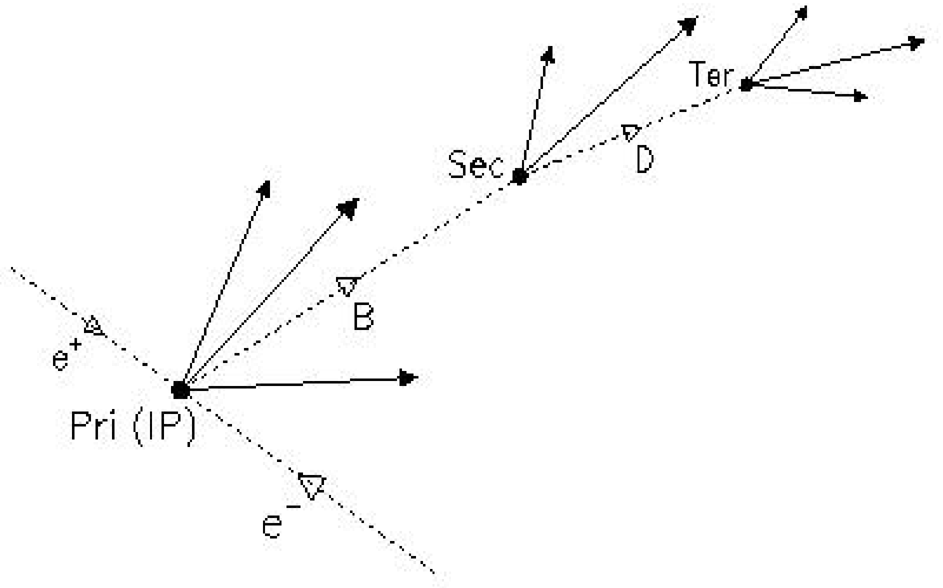
- **will be flexible enough to permit user further tuning of the input variables for the neural n**
  **and of the NN-architecture (number and type of nodes) and training algorithm**

# Flavour tag and quark charge sign selection

- aim of flavour tag: distinguish between b-jets, c- jets and light-quark / gluon jets
- heavy flavour jets contain secondary decays, generally observed as secondary vertices
- NN-approach to combine inputs; most sensitive: secondary vtx Pt-corrected mass & mom

*Klaus Desch/ Thorsten Kuhl*



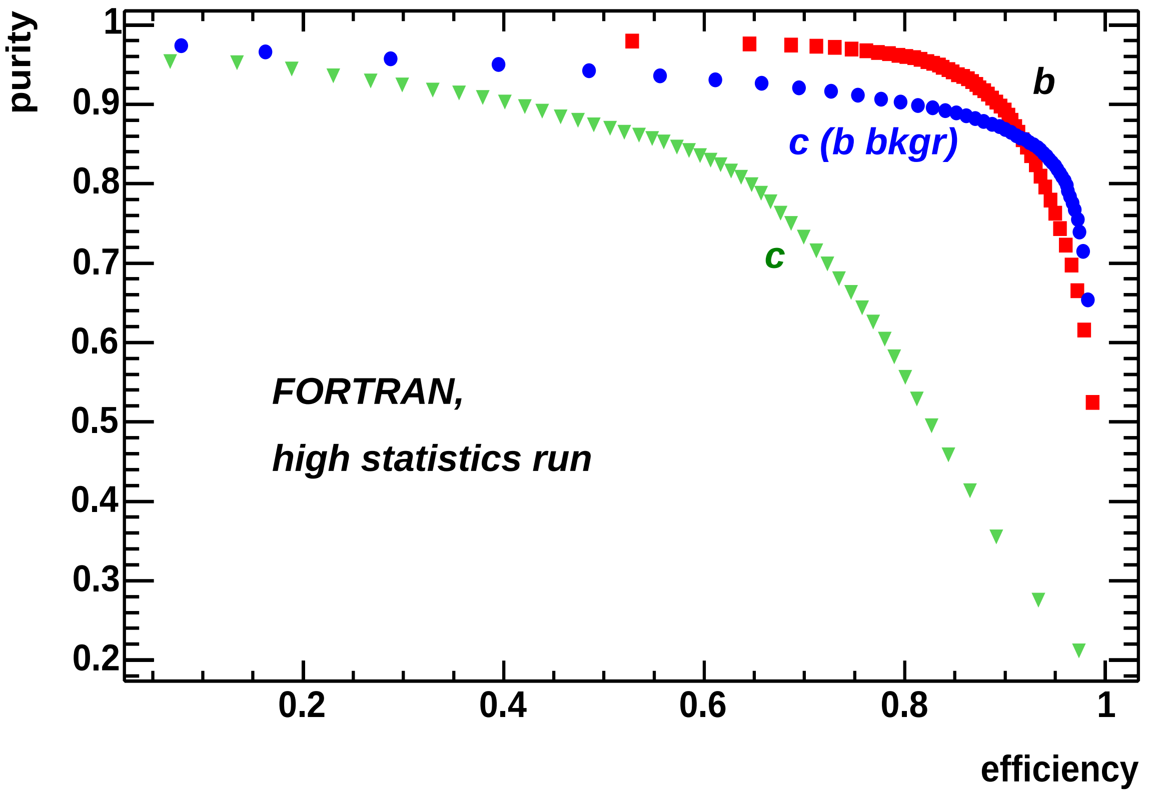


- for charged B-hadrons (40% of b-jets): quark sign can be determined from vertex charge

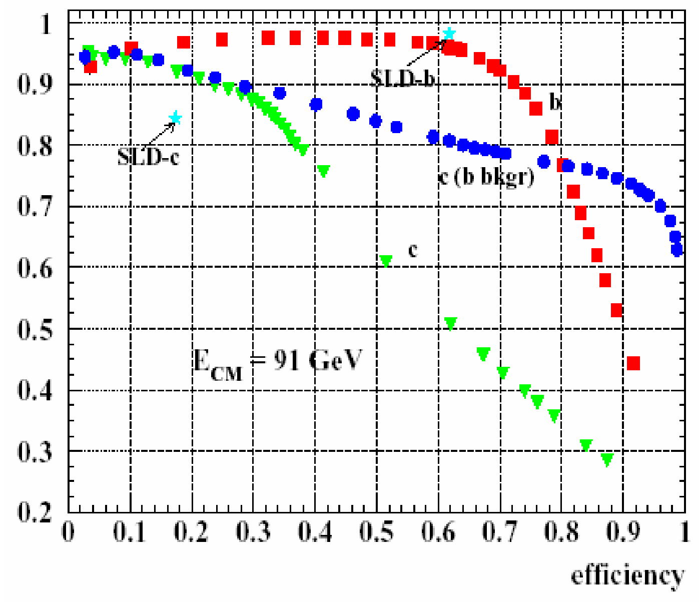
  need to find all stable tracks from B-decay chain

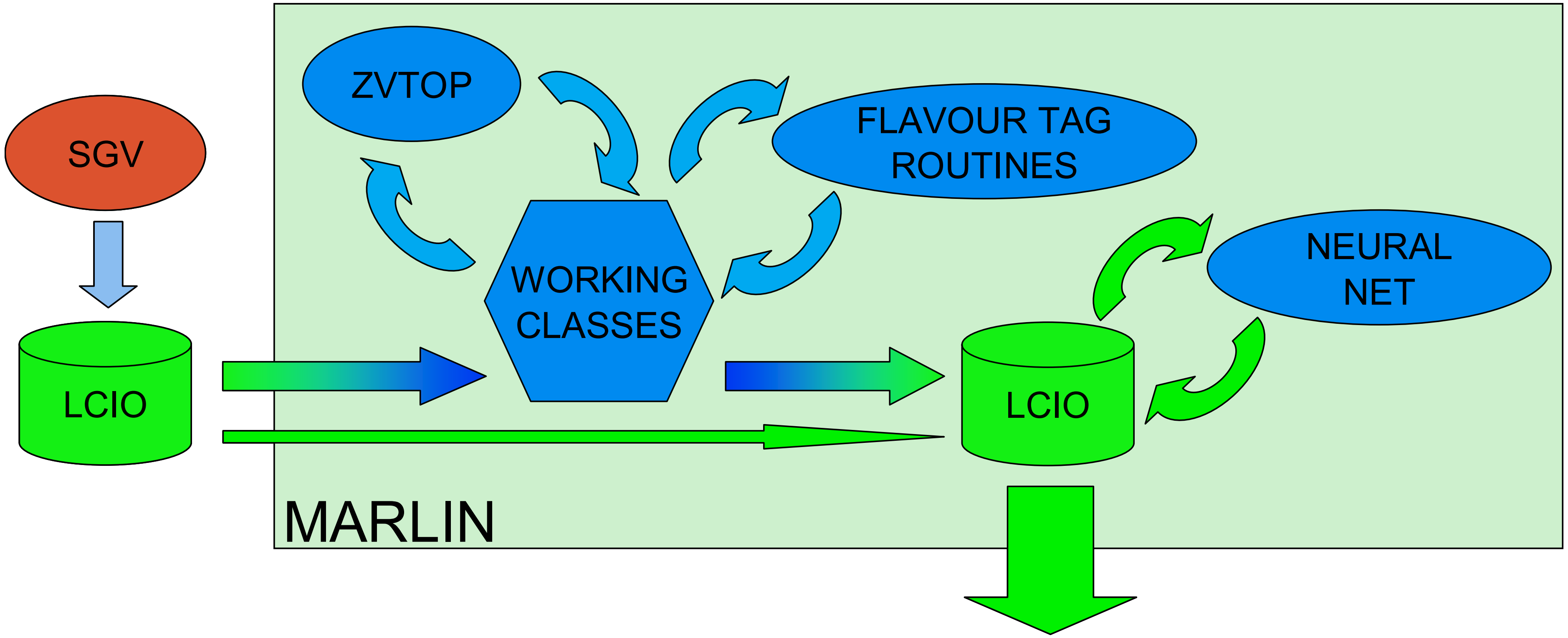
- probability of mis-reconstructing vertex charge small for both charged and neutral case
- neutral B-hadrons require 'charge dipole' procedure from SLD still to be developed for I

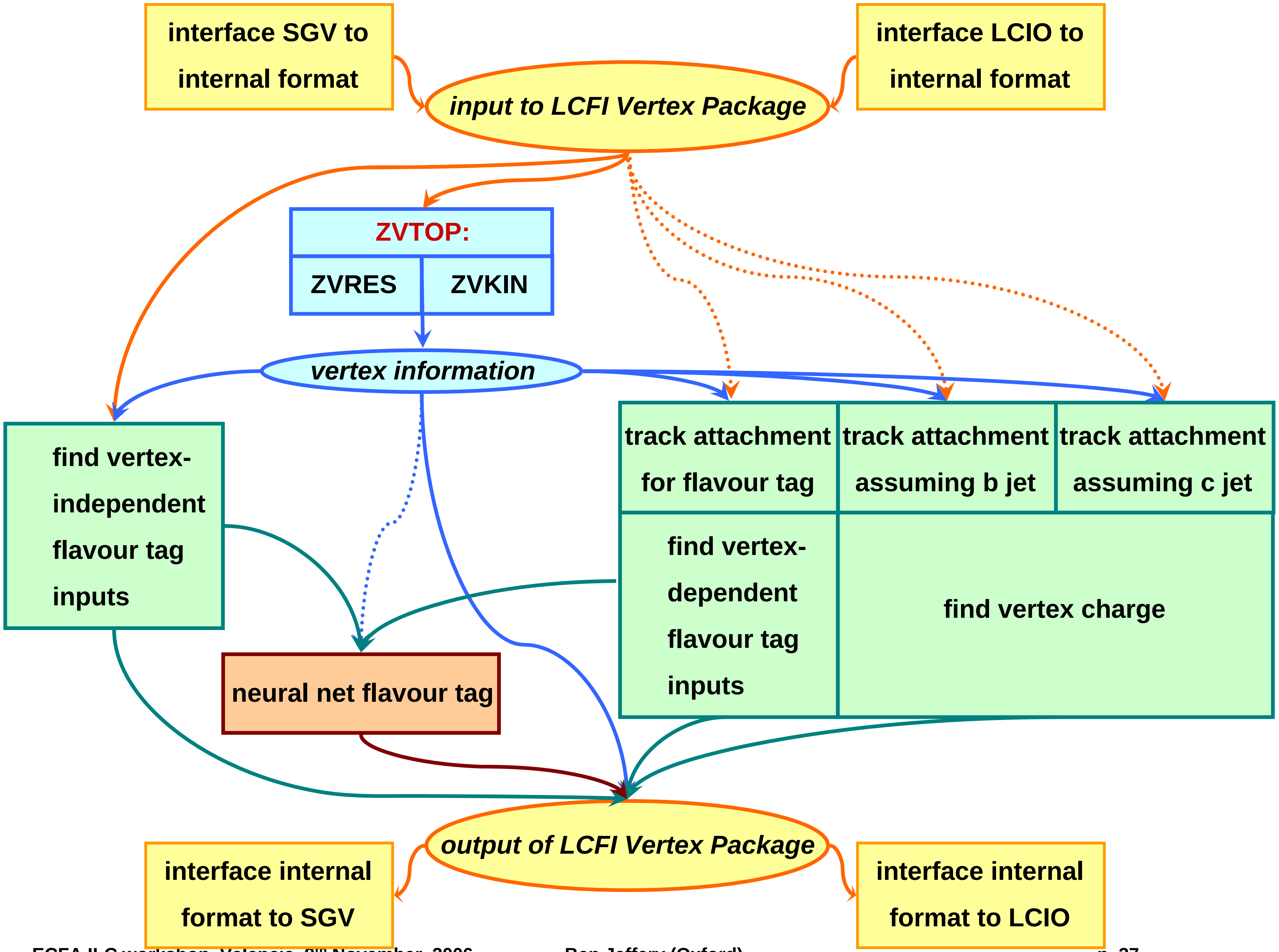
# Flavour tag purity vs efficiency at the Z-peak



K Desch/ Th Kuhl




ZVTOP
FLAVOUR TAG ROUTINES
SGV
NEURAL NET
WORKING CLASSES
LCIO
LCIO
MARLIN


interface SGV to internal format
interface LCIO to internal format
input to LCFI Vertex Package
ZVTOP:
ZVRES
ZVKIN
vertex information
find vertex-independent flavour tag inputs
track attachment for flavour tag
track attachment assuming b jet
track attachment assuming c jet
find vertex-dependent flavour tag inputs
find vertex charge
neural net flavour tag
output of LCFI Vertex Package
interface internal format to SGV
interface internal format to LCIO