Hall-D Software

David Lawrence

June 7, 2012

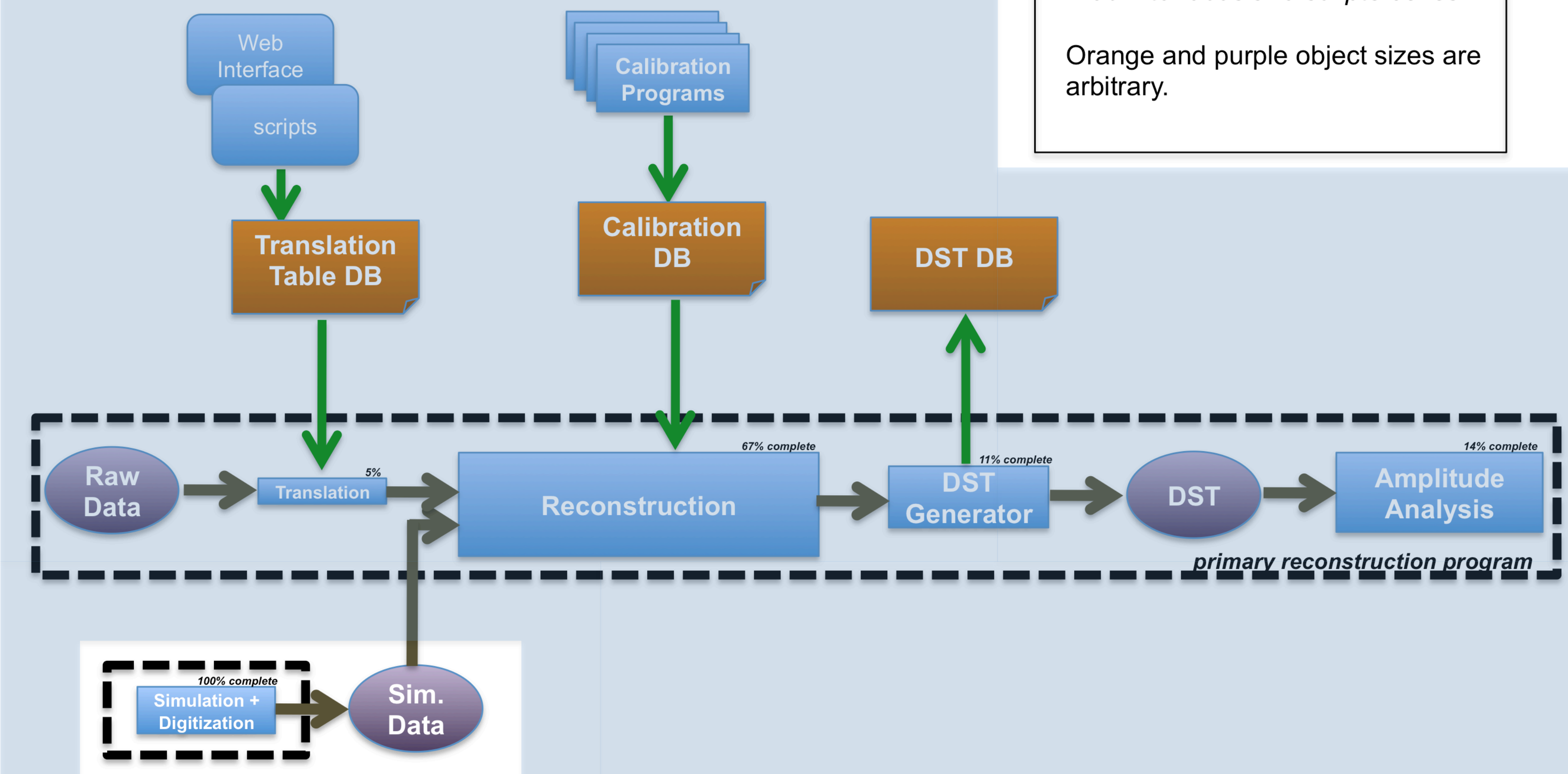
Outline

- Simulation
- Reconstruction
- Calibration Database
- Analysis (DST, Amplitude Analysis)
- Documentation
- Areas of greatest concern

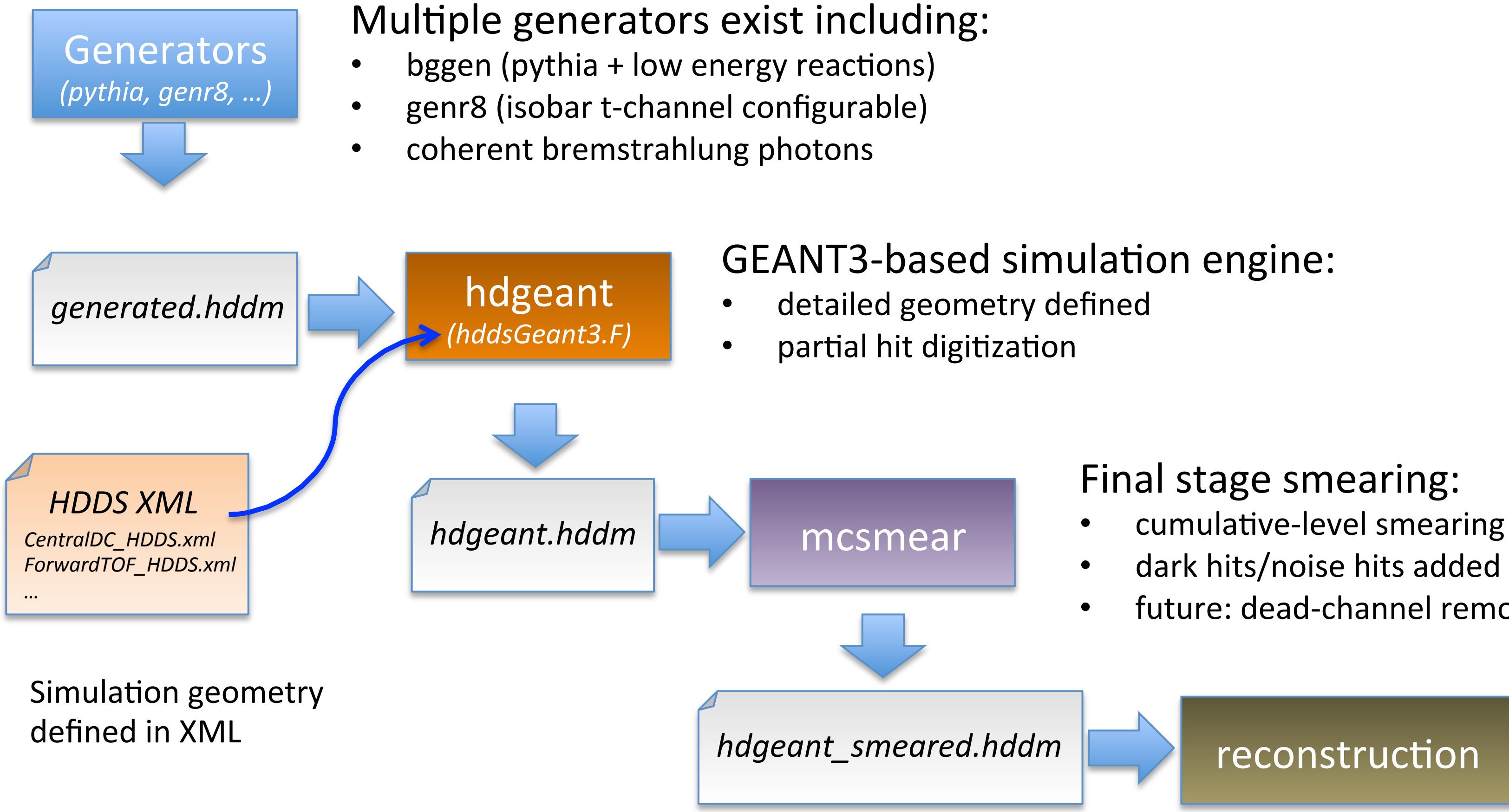
Rough Diagram of GlueX Software

Blue boxes are sized proportional to total estimated manpower with the exception of the *Web Interfaces* and *scripts* boxes.

Orange and purple object sizes are arbitrary.



Simulation



Multiple generators exist including:

- bggen (pythia + low energy reactions)
- genr8 (isobar t-channel configurable)
- coherent bremstrahlung photons

GEANT3-based simulation engine:

- detailed geometry defined
- partial hit digitization

Final stage smearing:

- cumulative-level smearing done
- dark hits/noise hits added
- future: dead-channel removal

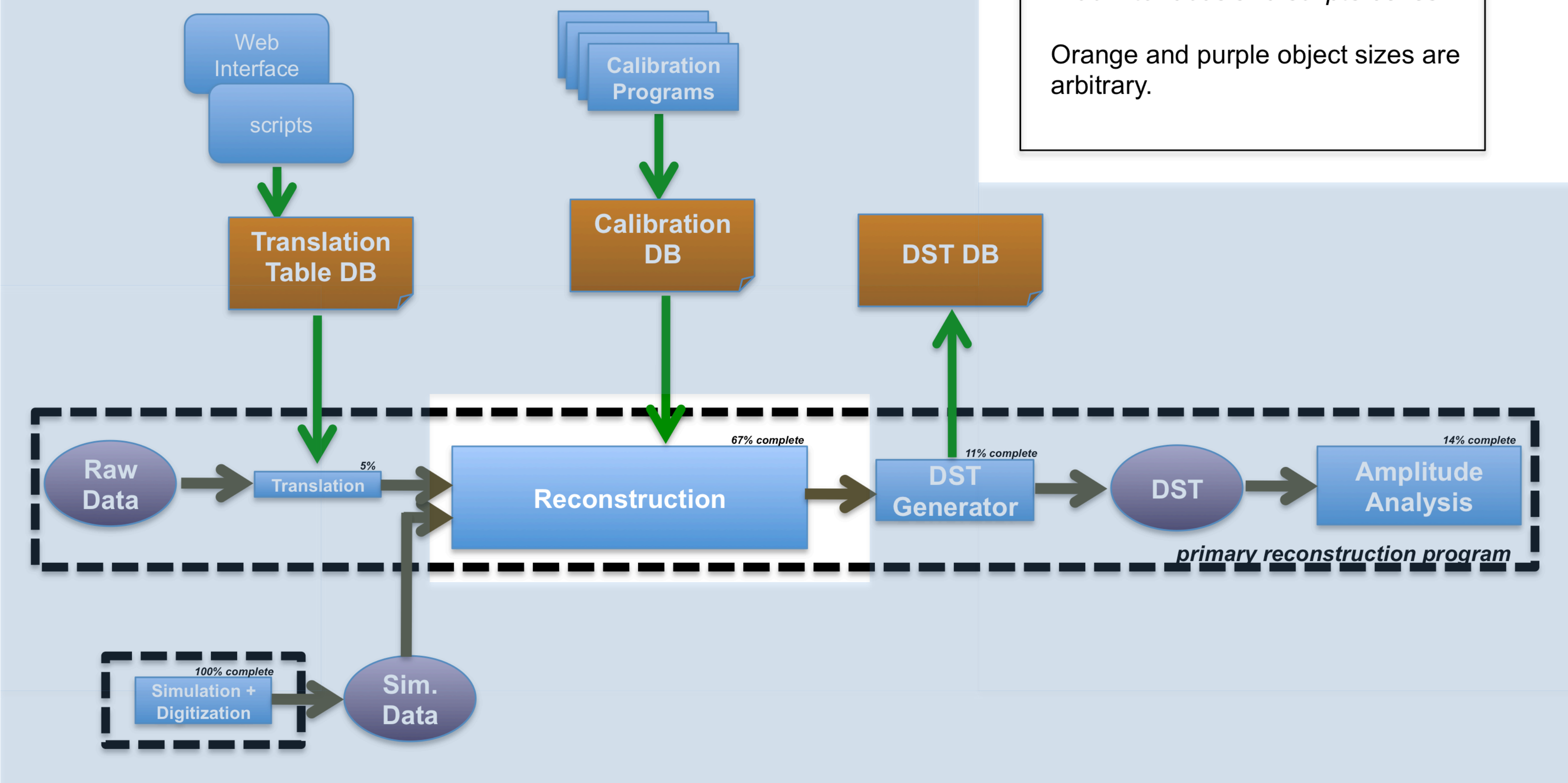
Simulation geometry defined in XML

(see backup slides 21, 22)

Rough Diagram of GlueX Software

Blue boxes are sized proportional to total estimated manpower with the exception of the *Web Interfaces* and *scripts* boxes.

Orange and purple object sizes are arbitrary.

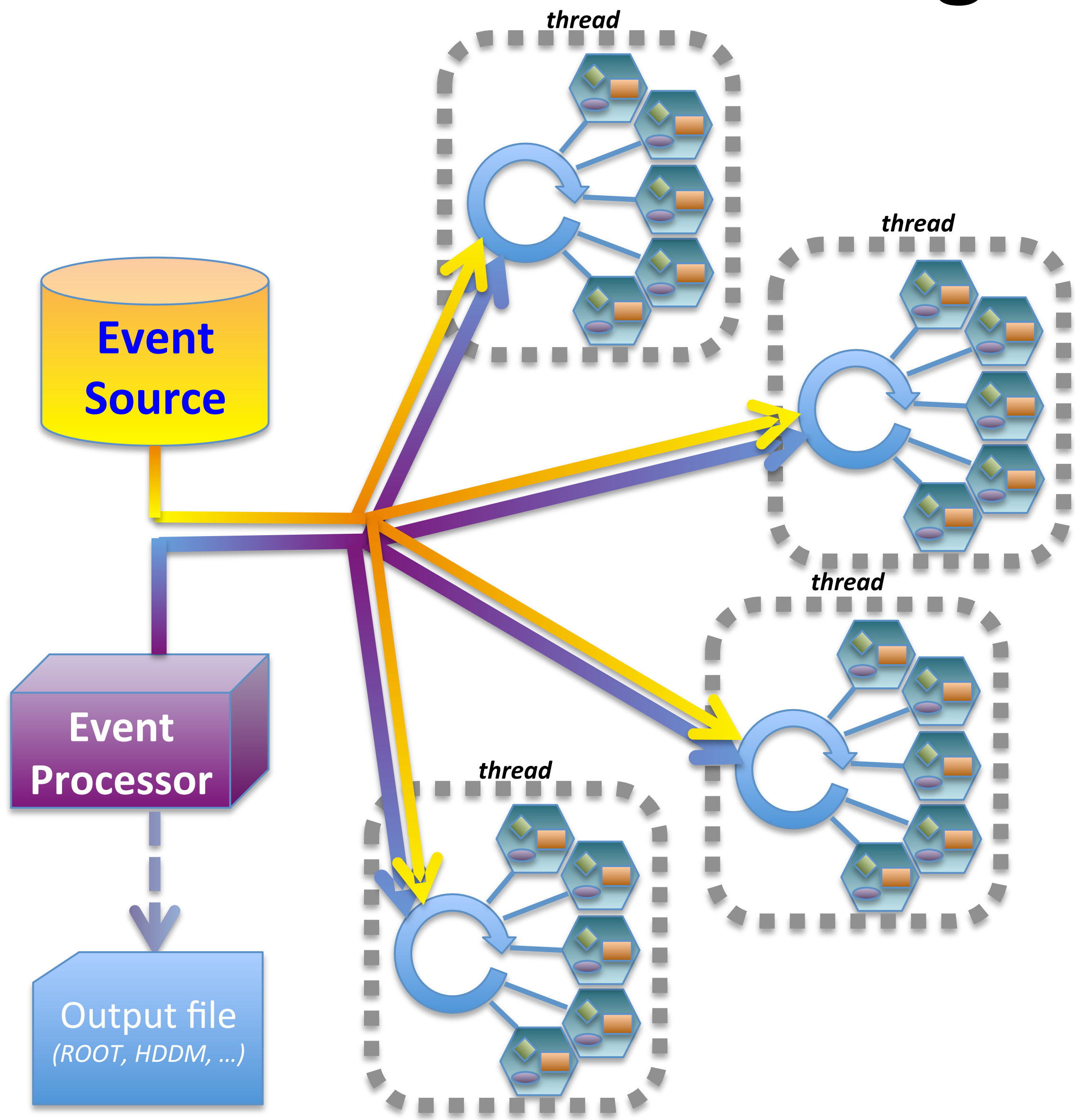


JANA Framework: Multi-threading

○ *Each thread has a complete set of factories making it capable of completely reconstructing a single event*

○ *Factories only work with other factories in the same thread eliminating the need for expensive mutex locking within the factories*

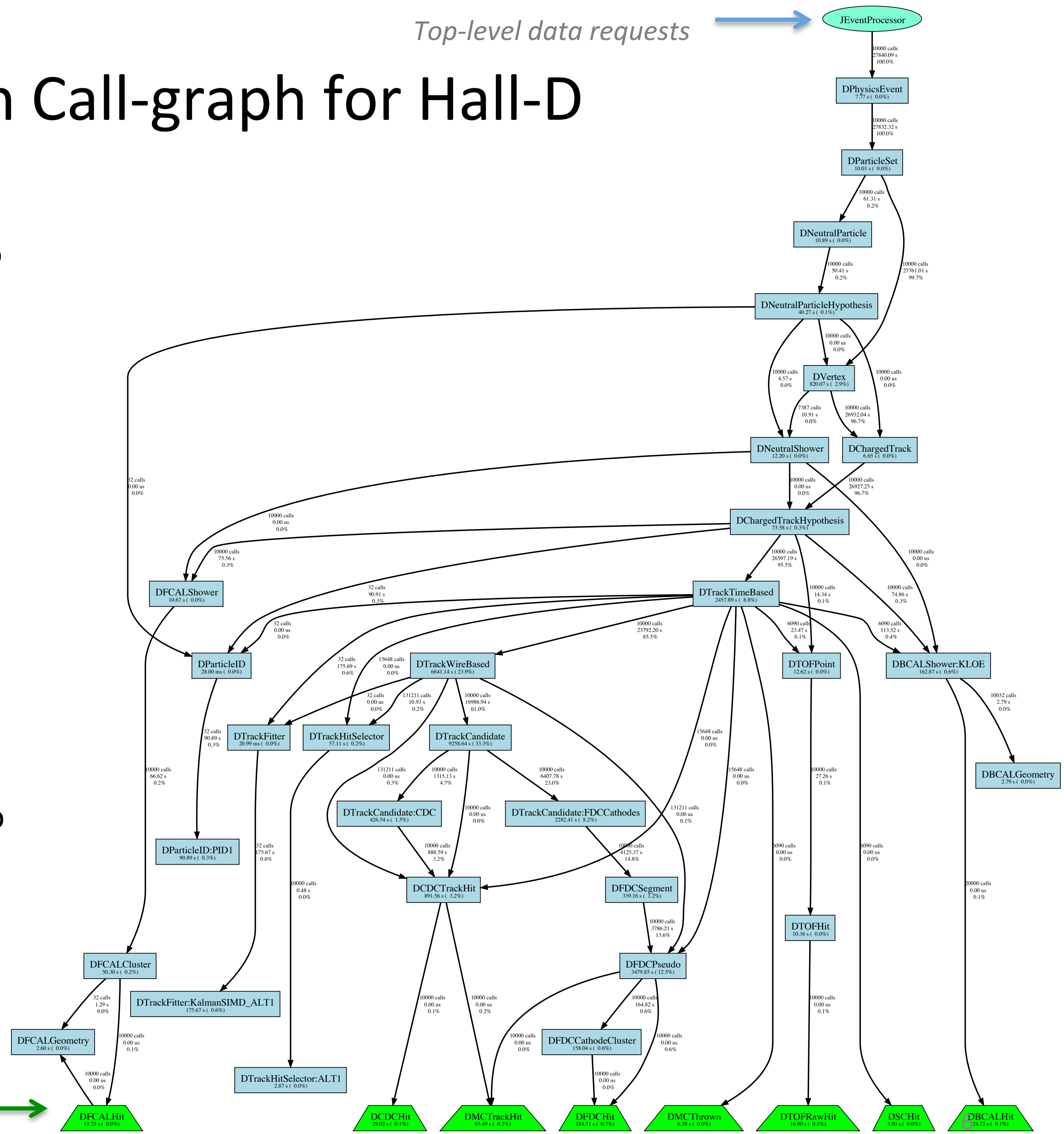
○ *All events are seen by all Event Processors (multiple processors can exist in a program)*



Reconstruction Call-graph for Hall-D

- Plugin (*janadot*) activates profiling features built into JANA framework
- Results written to output file in format compatible with graphviz's *dot* program
- Factory dependencies are recorded as well as time spent in each factory and time spent fulfilling each request
- Clickable HTML image map can be autogenerated from this with links pointing back to Doxygen documentation

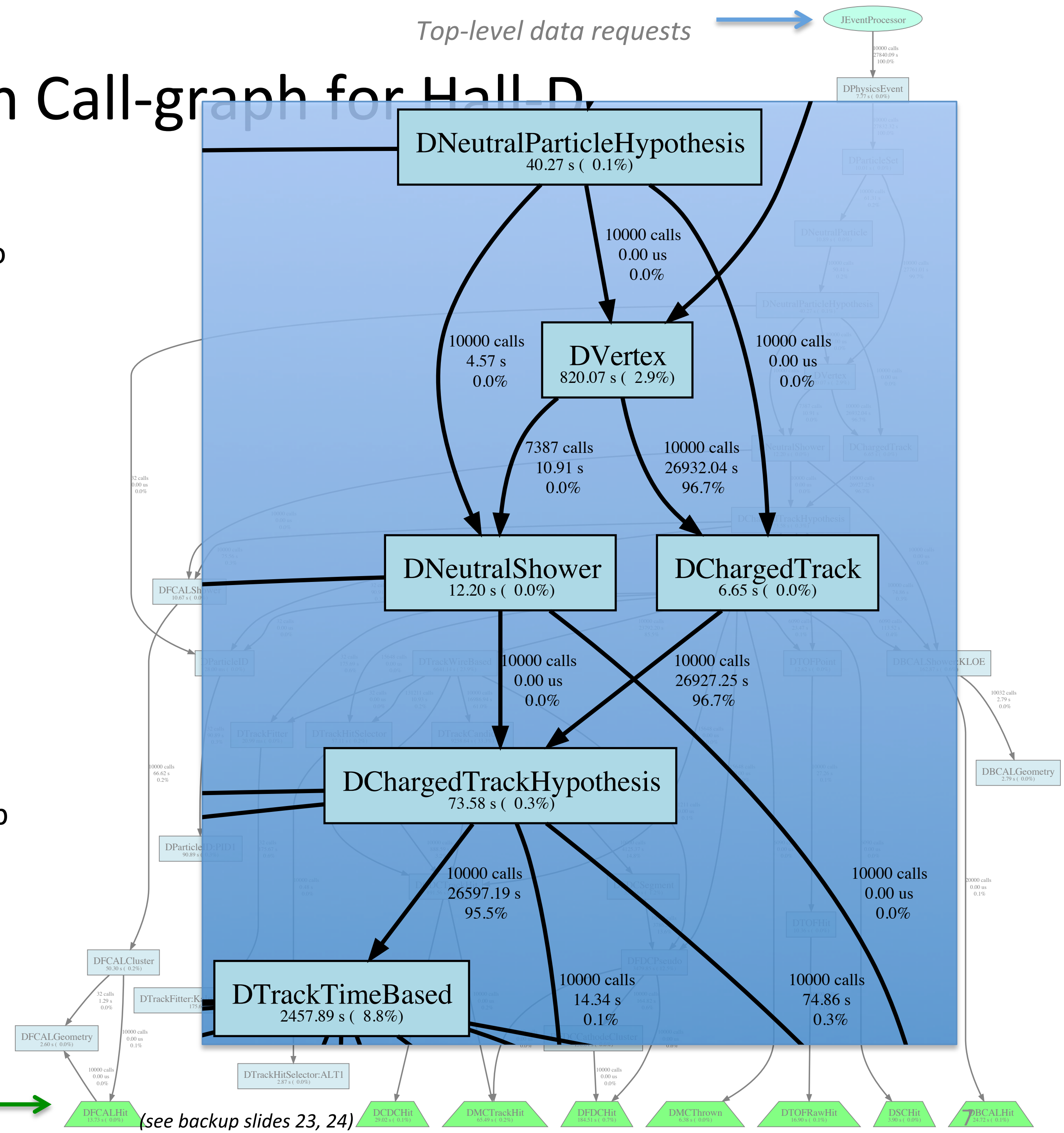
Top-level data requests



Objects read from event source (e.g. file)

Reconstruction Call-graph for Hall-D

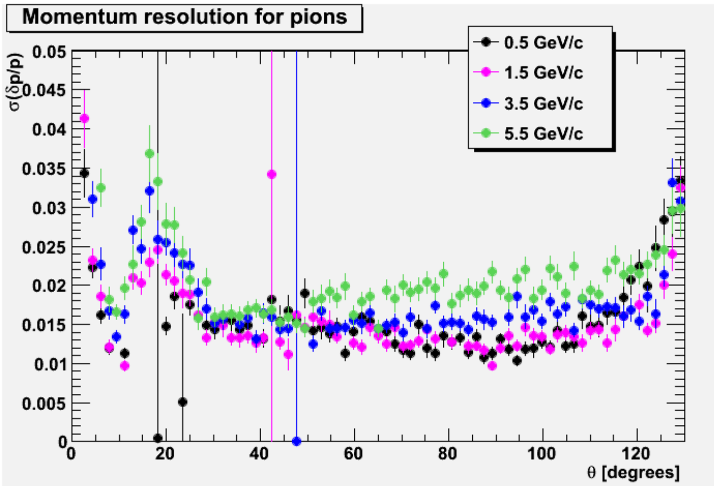
- Plugin (*janadot*) activates profiling features bulit into JANA framework
- Results written to output file in format compatible with graphviz's *dot* program
- Factory dependencies are recorded as well as time spent in each factory and time spent fulfilling each request
- Clickable HTML image map can be autogenerated from this with links pointing back to Doxygen documentation

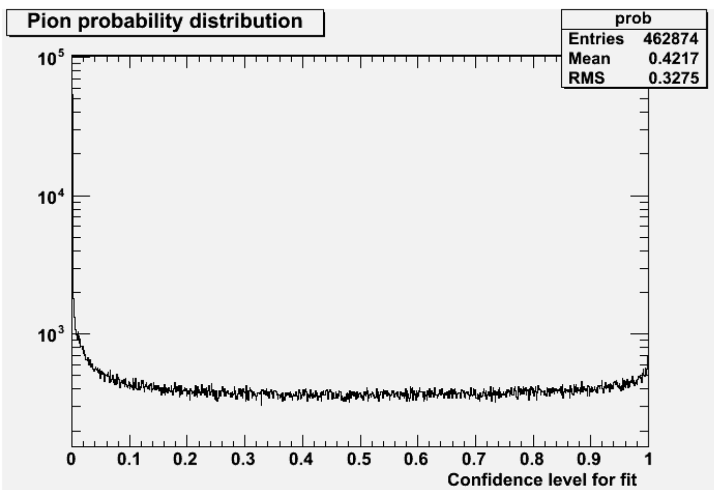


Objects read from event source (e.g. file)

(see backup slides 23, 24)

Charged Particle Tracking





Plots from 5/4/12 semi-weekly single track tests

- Tracking code development began in 2004

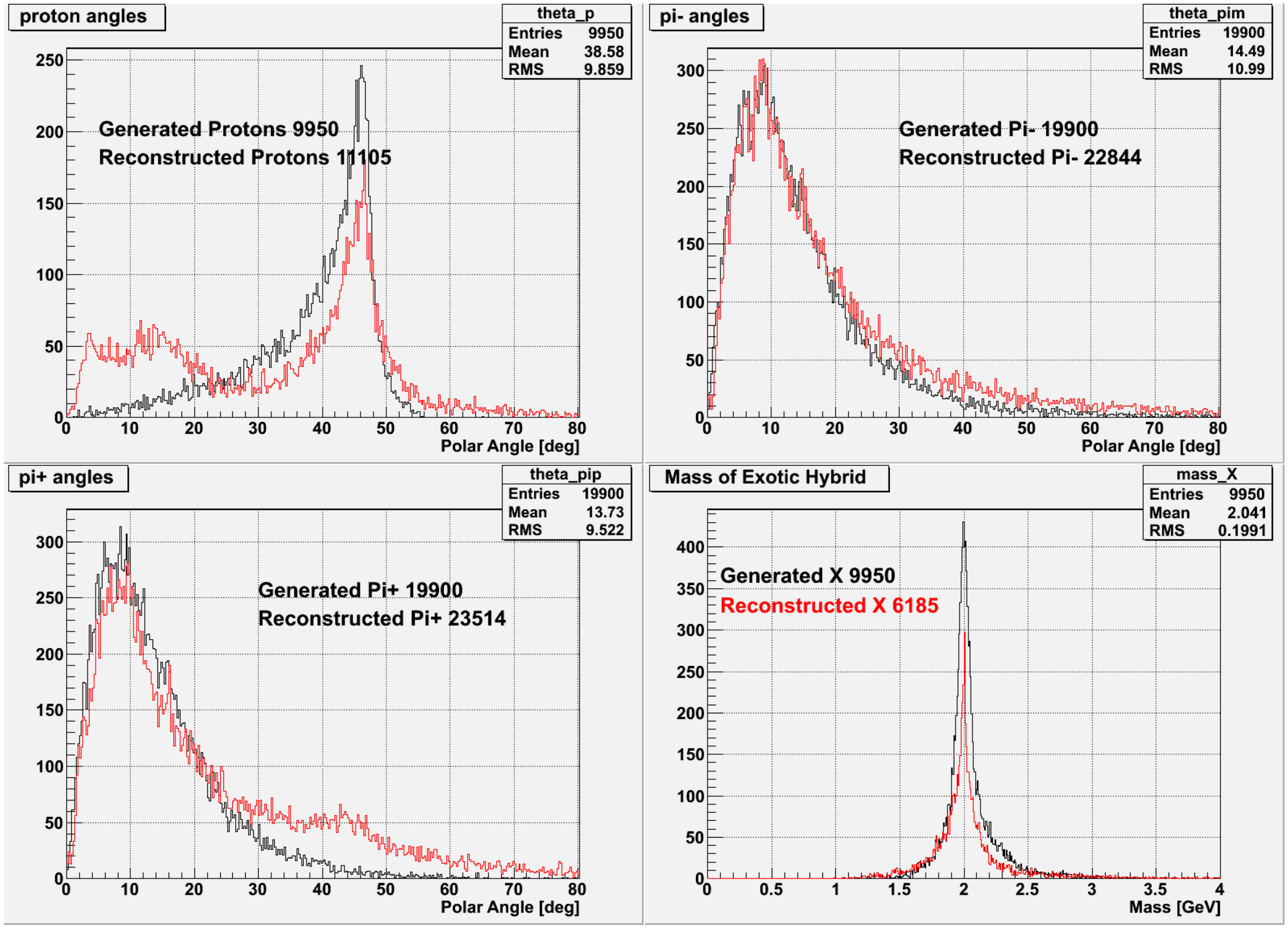
- Tracking is done in multiple stages:
  - Track Finding
  - Wire-based fitting *(wire positions only)*
  - Time-based fitting *(drift times used)*

- Fitting done using a Kalman Filter
  (replaced original least-squares fitter)

Source code is checked out and built via nightly cron job on 3 platforms

Twice a week cron jobs automatically simulate and reconstruct single track events and multi-track $b_1\pi$ events

Charged Particle Tracking



Plots from 5/7/12 semi-weekly $b_1\pi$ tests

Results of recent semi-weekly test doing full reconstruction of $b_1\pi$ events

Final state: p $\pi^+ \pi^+ \pi^- \pi^- \gamma\gamma$ (5 charged tracks)

Some mis-identification of π^+ and proton exists

>60% reconstruction efficiency of “X” meson

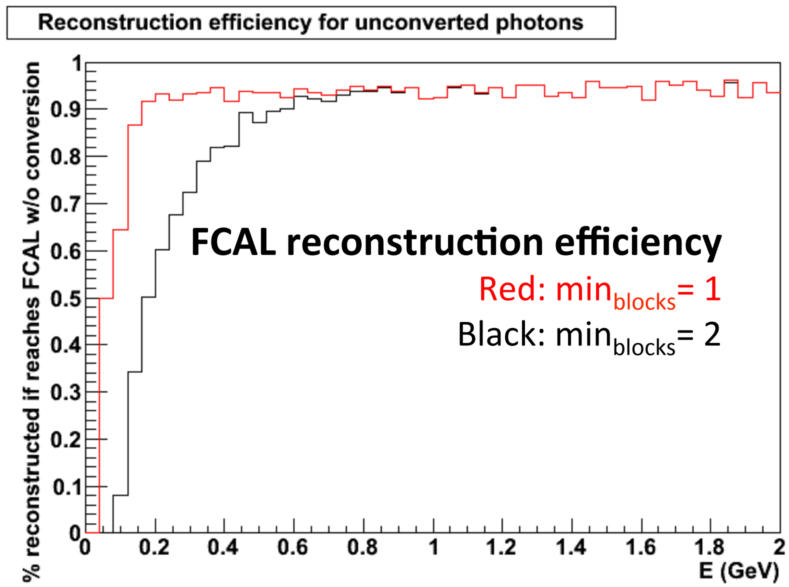
(see backup slide 25)

Calorimetry

<u>FCAL</u>

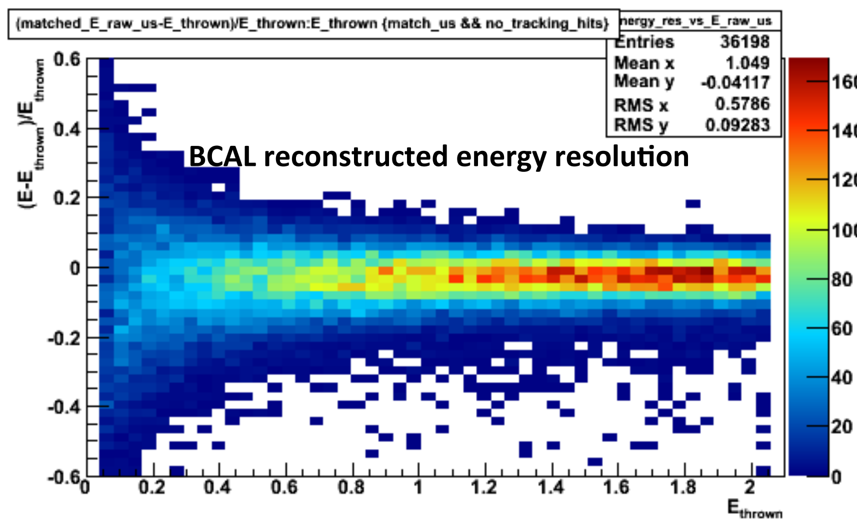
Code developed based on experience with Rad-ϕ experiment in Hall-B

- Full reconstruction (GlueX-doc-823)
- Depth corrections (GlueX-doc-1093)
- Calibration procedure established

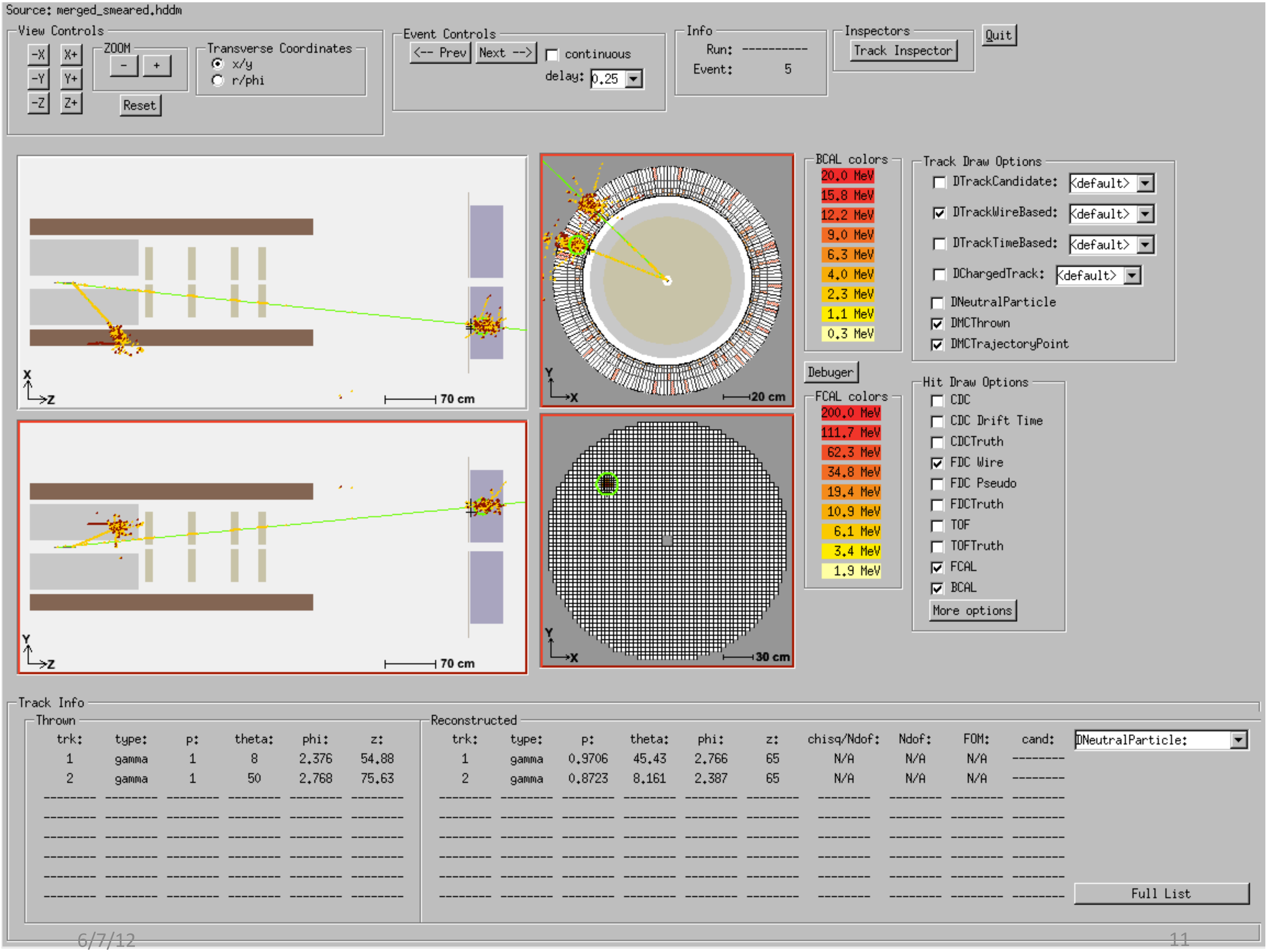


<u>BCAL</u>

- 1st generation Code developed copied from KLOE and adapted to GlueX
- 2nd generation currently under development by GlueX
  - Improved angular resolution
  - Better error estimation
  - Increased π^o reconstruction efficiency
    (61%->73% 11/28/11 report)



(see backup slide 26)



trk:	type:	p:	theta:	phi:	z:
1	gamma	1	8	2.376	54.88
2	gamma	1	50	2.768	75.63

trk:	type:	p:	theta:	phi:	z:	chisq/Ndof:	Ndof:	FOM:	cand:
1	gamma	0.9706	45.43	2.766	65	N/A	N/A	N/A	--------
2	gamma	0.8723	8.161	2.387	65	N/A	N/A	N/A	--------

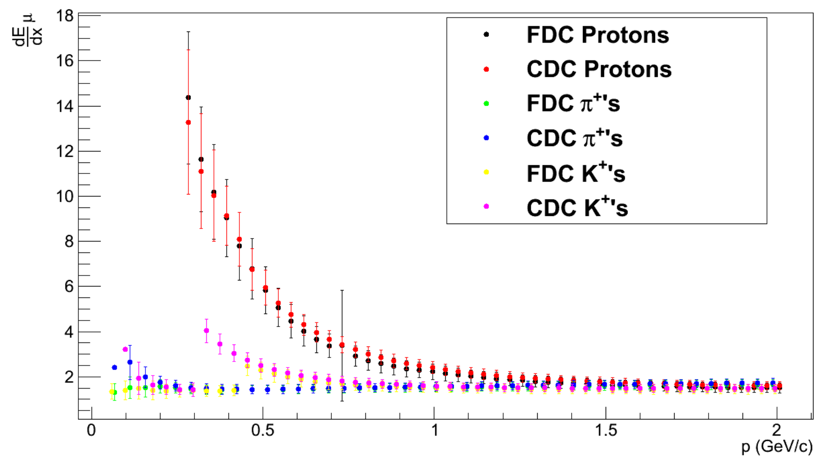
Particle Identification

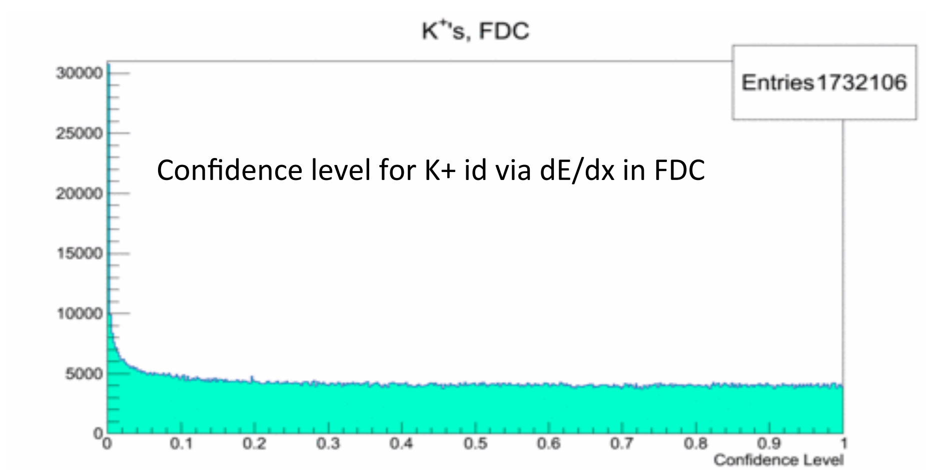
3 main handles on PID with baseline GlueX

1. dE/dx in drift chambers
2. Tracking χ^2
3. Time of flight (β)

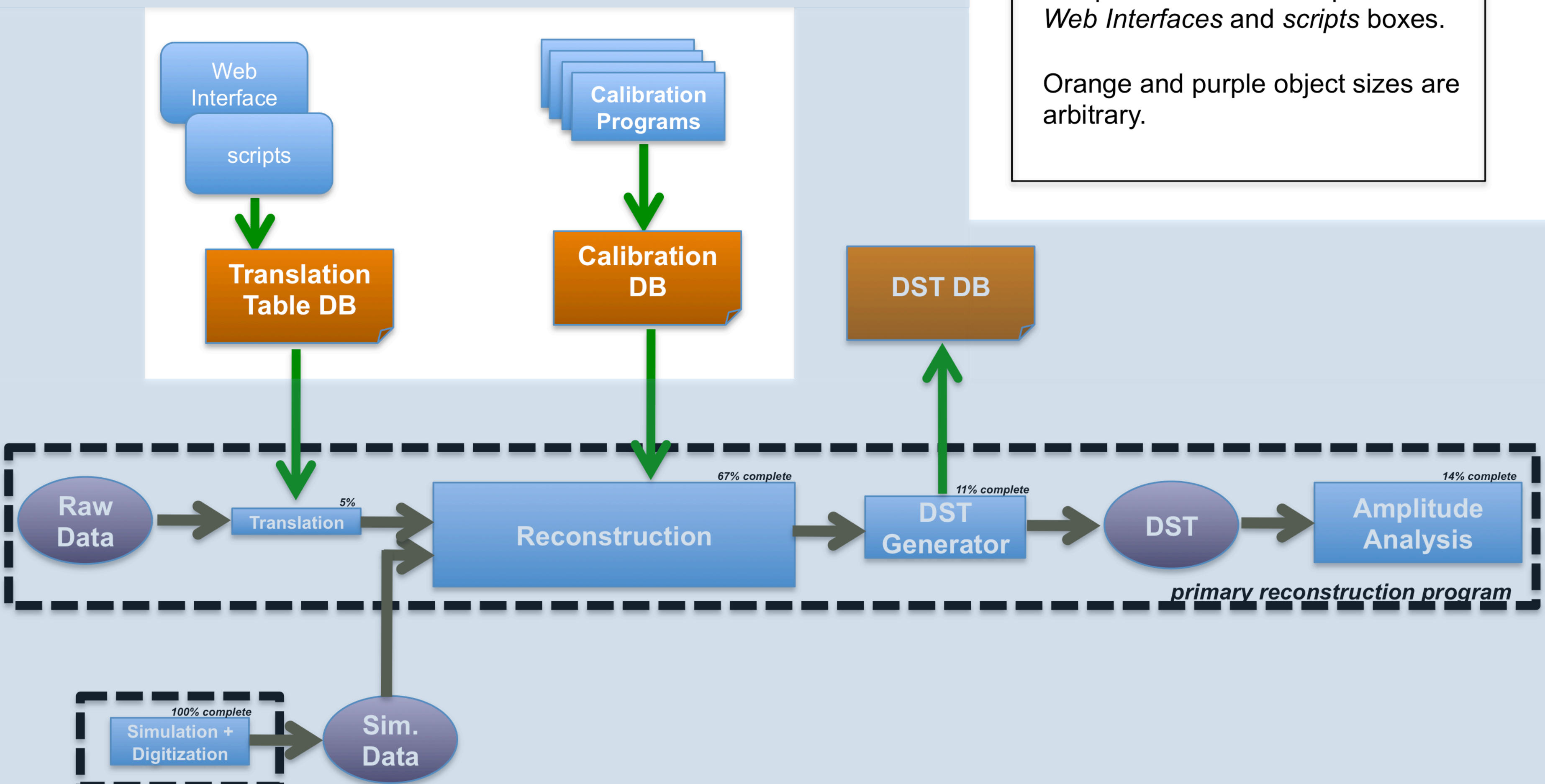
Three elements are combined into single confidence level for the probability of a given track being a specific particle type

Multiple mass hypotheses are fit for each charged track so end analysis may use all appropriate info





Rough Diagram of GlueX Software

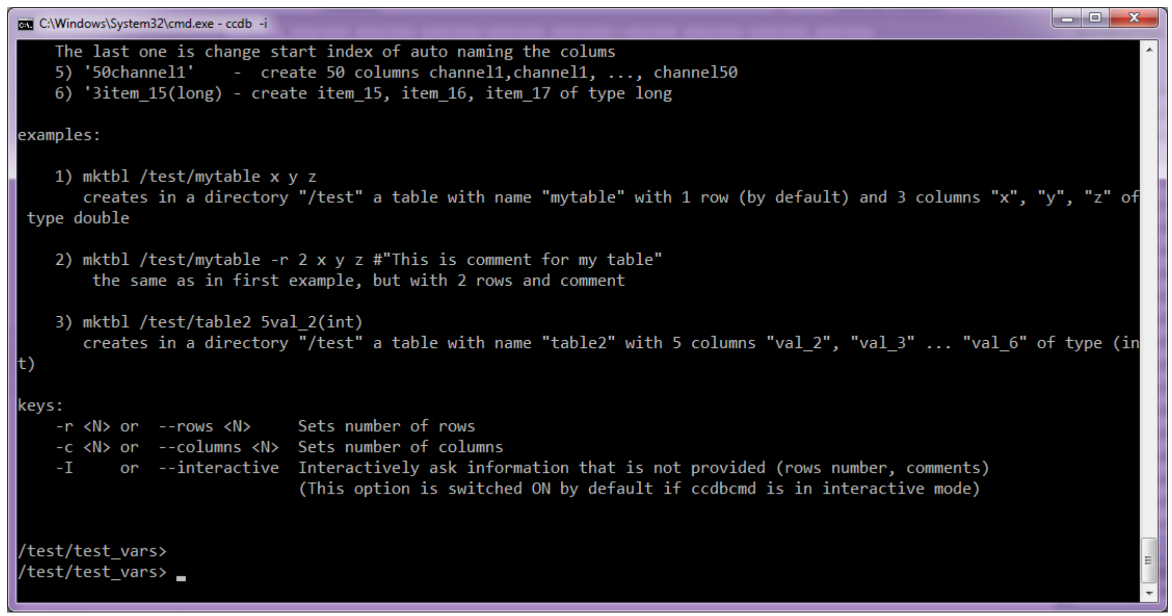


Blue boxes are sized proportional to total estimated manpower with the exception of the *Web Interfaces* and *scripts* boxes.

Orange and purple object sizes are arbitrary.

Calibration DB: CCDB

- MySQL backed *(file option exists)*
- C++, Python, and PHP interfaces *(additional interfaces available via SWIG)*
- Full history mechanism *(allows one to obtain constants should the query been performed at a specific time in the past)*
- Variation mechanism to allow shallow copies of complete sets to be made and modified
- Calibrations indexed by run number
- 1-D arrays or 2-D tables indexed via position or name
- Benchmark: >125k constants read in 0.7 s *(includes B-field and material maps)*
- Constants named via hierarchical /name/paths
  (e.g. Magnets/Solenoid/solenoid_1500)
- Shell interface tool

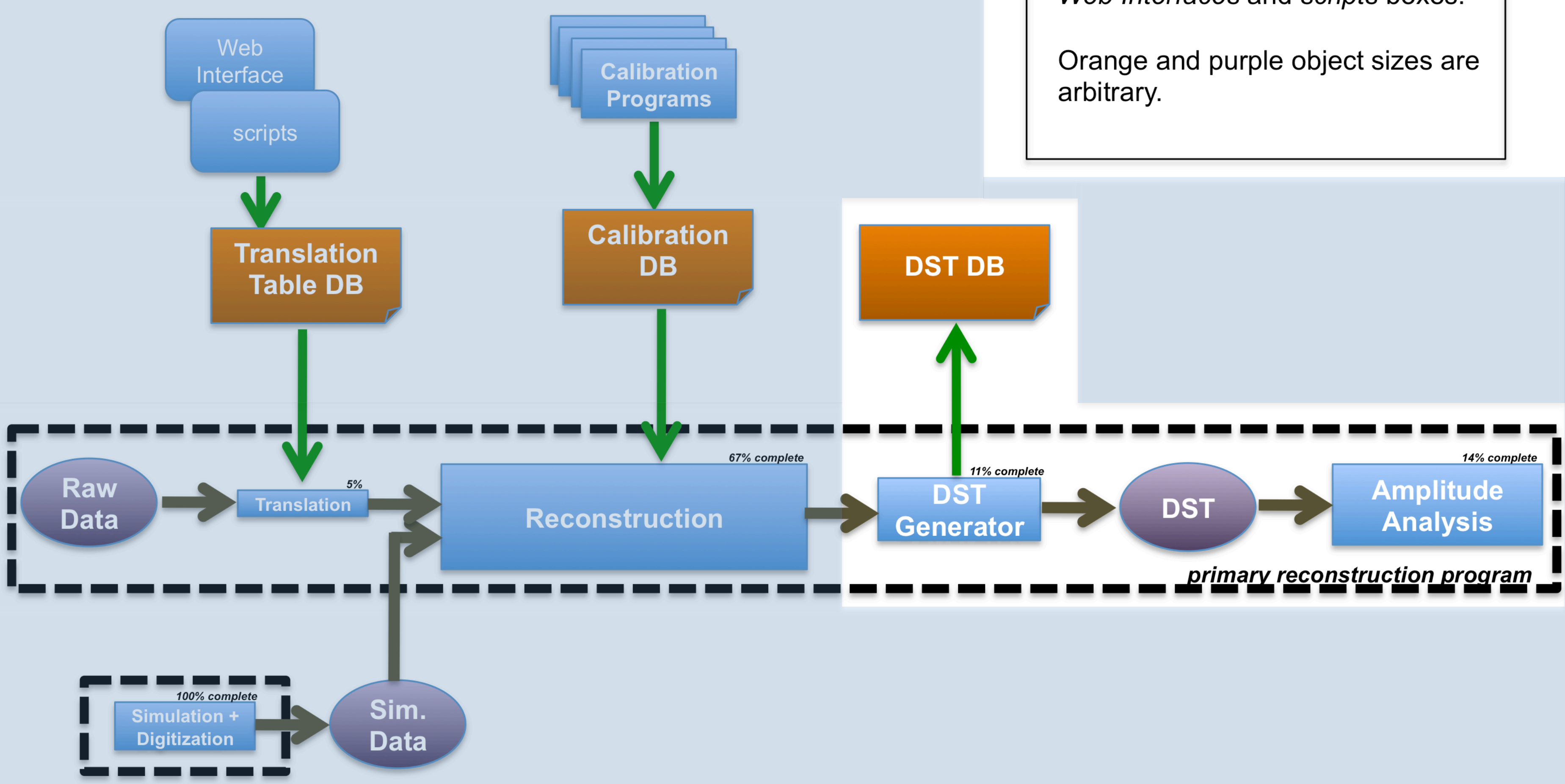


(see backup slide 27)

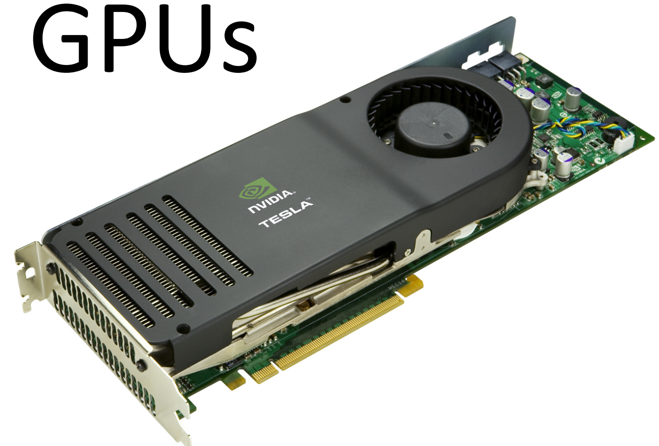
Rough Diagram of GlueX Software

Blue boxes are sized proportional to total estimated manpower with the exception of the *Web Interfaces* and *scripts* boxes.

Orange and purple object sizes are arbitrary.



Amplitude Analysis and GPUs



Graphics Processing Units (GPUs) developed for the gaming industry are finding many applications in the scientific community.

Amplitude Analysis (a.k.a Partial Wave Analysis) tools have been developed that take advantage of GPU parallelism. Documentation and examples exist.

Fit Configuration	Time to Converge (seconds)
Single CPU	150.7
Single CPU + 1 GPU	23.6
CPU Master + 4 (CPU + GPU)	6.3
CPU Master + 11 CPU Workers	17.8

(All fits converge to the same minimum with variations in iterations of ±1-2%)

Time for 10^6 Amplitude Computations (ms)

Amplitude	CPU	GPU*
Breit-Wigner	800	8
Ang. Dist. (D-functions)	15,000	87

* includes time to copy result from GPU memory

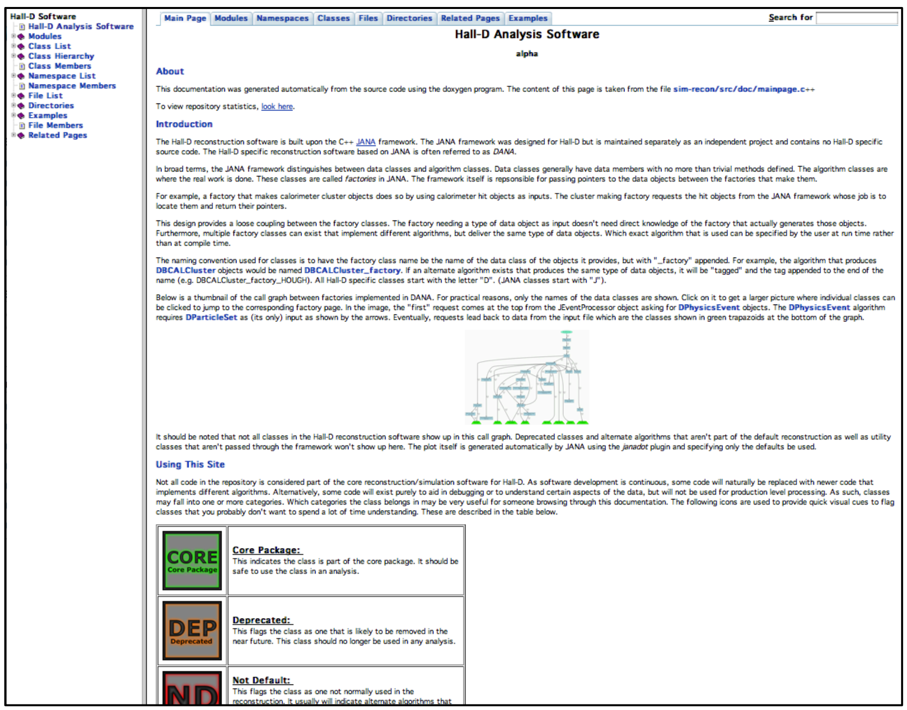
Tables from Jan. 6, 2011 talk by Matt Shepherd at "Parallelism in Nuclear Physics Workshop" at JLab

(see backup slide 28)

Documentation

- Multiple layers of documentation exist
- Effort is being made to maintain
- Major updates occasionally *(~1-2 years)*
- Multiple software workshops
  - ➢ 2005, 2006, 2010 (CLAS12), 2011
  - ➢ 2011 workshop included full set of working examples on VirtualBox machine

JANA Manual



Hall-D Software
Hall-D Analysis Software
Modules
Class List
Class Hierarchy
Class Members
Namespace List
Namespace Members
File List
Directories
Examples
File Members
Related Pages

Main Page | Modules | Namespaces | Classes | Files | Directories | Related Pages | Examples — Search for

Hall-D Analysis Software

alpha

About

This documentation was generated automatically from the source code using the doxygen program. The content of this page is taken from the file **sim-recon/src/doc/mainpage.c++**

To view repository statistics, look here.

Introduction

The Hall-D reconstruction software is built upon the C++ JANA framework. The JANA framework was designed for Hall-D but is maintained separately as an independent project and contains no Hall-D specific source code. The Hall-D specific reconstruction software based on JANA is often referred to as *DANA*.

In broad terms, the JANA framework distinguishes between data classes and algorithm classes. Data classes generally have data members with no more than trivial methods defined. The algorithm classes are where the real work is done. These classes are called *factories* in JANA. The framework itself is repsonsible for passing pointers to the data objects between the factories that make them.

For example, a factory that makes calorimeter cluster objects does so by using calorimeter hit objects as inputs. The cluster making factory requests the hit objects from the JANA framework whose job is to locate them and return their pointers.

This design provides a loose coupling between the factory classes. The factory needing a type of data object as input doesn't need direct knowledge of the factory that actually generates those objects. Furthermore, multiple factory classes can exist that implement different algorithms, but deliver the same type of data objects. Which exact algorithm that is used can be specified by the user at run time rather than at compile time.

The naming convention used for classes is to have the factory class name be the name of the data class of the objects it provides, but with "_factory" appended. For example, the algorithm that produces **DBCALCluster** objects would be named **DBCALCluster_factory**. If an alternate algorithm exists that produces the same type of data objects, it will be "tagged" and the tag appended to the end of the name (e.g. DBCALCluster_factory_HOUGH). All Hall-D specific classes start with the letter "D". (JANA classes start with "J").

Below is a thumbnail of the call graph between factories implemented in DANA. For practical reasons, only the names of the data classes are shown. Click on it to get a larger picture where individual classes can be clicked to jump to the corresponding factory page. In the image, the "first" request comes at the top from the JEventProcessor object asking for **DPhysicsEvent** objects. The **DPhysicsEvent** algorithm requires **DParticleSet** as (its only) input as shown by the arrows. Eventually, requests lead back to data from the input file which are the classes shown in green trapazoids at the bottom of the graph.

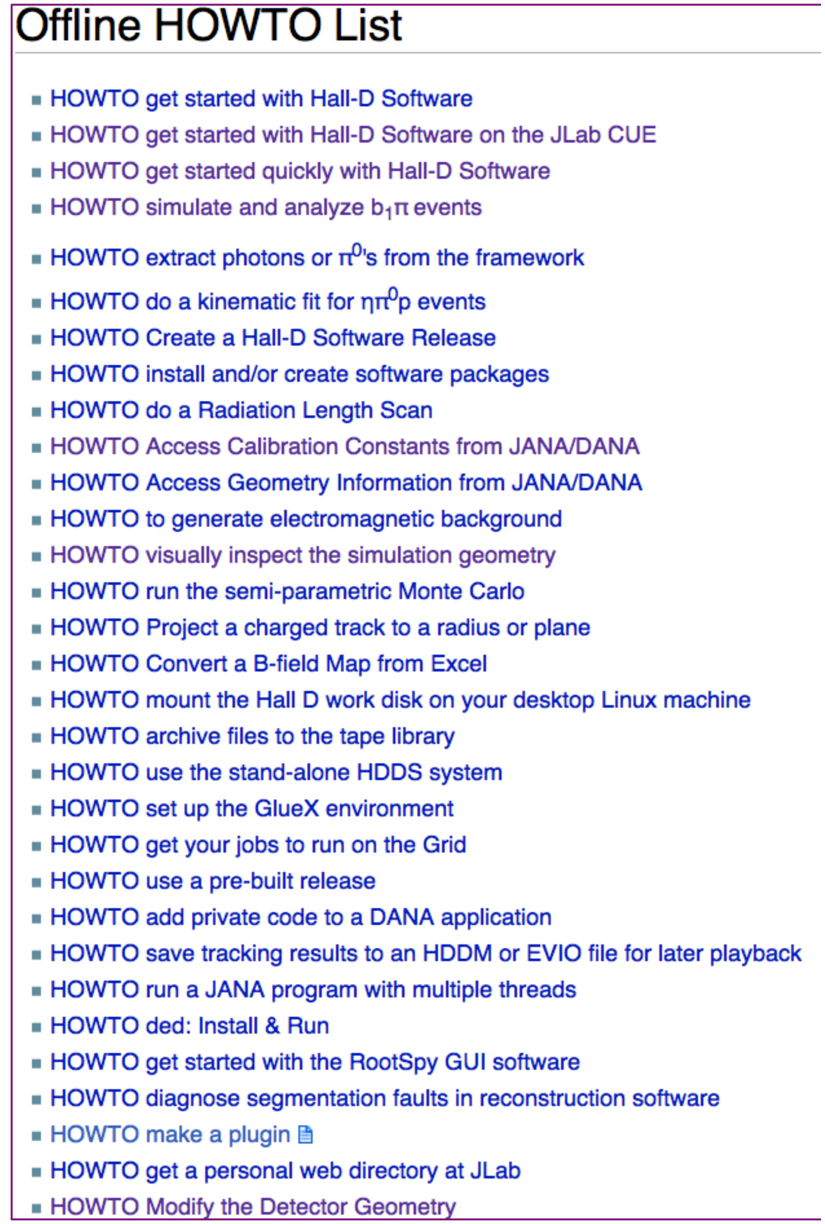
It should be noted that not all classes in the Hall-D reconstruction software show up in this call graph. Deprecated classes and alternate algorithms that aren't part of the default reconstruction as well as utility classes that aren't passed through the framework won't show up here. The plot itself is generated automatically by JANA using the *janadot* plugin and specifying only the defaults be used.

Using This Site

Not all code in the repository is considered part of the core reconstruction/simulation software for Hall-D. As software development is continuous, some code will naturally be replaced with newer code that implements different algorithms. Alternatively, some code will exist purely to aid in debugging or to understand certain aspects of the data, but will not be used for production level processing. As such, classes may fall into one or more categories. Which categories the class belongs in may be very useful for someone browsing through this documentation. The following icons are used to provide quick visual cues to flag classes that you probably don't want to spend a lot of time understanding. These are described in the table below.

CORE Core Package	**Core Package:** This indicates the class is part of the core package. It should be safe to use the class in an analysis.
DEP Deprecated	**Deprecated:** This flags the class as one that is likely to be removed in the near future. This class should no longer be used in any analysis.
ND	**Not Default:** This flags the class as one not normally used in the reconstruction. It usually will indicate alternate algorithms that

Doxygen-generated HTML documentation



Offline HOWTO List

- HOWTO get started with Hall-D Software
- HOWTO get started with Hall-D Software on the JLab CUE
- HOWTO get started quickly with Hall-D Software
- HOWTO simulate and analyze $b_1\pi$ events
- HOWTO extract photons or π^0's from the framework
- HOWTO do a kinematic fit for $\eta\pi^0 p$ events
- HOWTO Create a Hall-D Software Release
- HOWTO install and/or create software packages
- HOWTO do a Radiation Length Scan
- HOWTO Access Calibration Constants from JANA/DANA
- HOWTO Access Geometry Information from JANA/DANA
- HOWTO to generate electromagnetic background
- HOWTO visually inspect the simulation geometry
- HOWTO run the semi-parametric Monte Carlo
- HOWTO Project a charged track to a radius or plane
- HOWTO Convert a B-field Map from Excel
- HOWTO mount the Hall D work disk on your desktop Linux machine
- HOWTO archive files to the tape library
- HOWTO use the stand-alone HDDS system
- HOWTO set up the GlueX environment
- HOWTO get your jobs to run on the Grid
- HOWTO use a pre-built release
- HOWTO add private code to a DANA application
- HOWTO save tracking results to an HDDM or EVIO file for later playback
- HOWTO run a JANA program with multiple threads
- HOWTO ded: Install & Run
- HOWTO get started with the RootSpy GUI software
- HOWTO diagnose segmentation faults in reconstruction software
- HOWTO make a plugin
- HOWTO get a personal web directory at JLab
- HOWTO Modify the Detector Geometry

(see backup slides 29,30,31,32)

Hall-D Software: Areas of greatest concern

must haves

1. *Calibration
   - Code to generate constants
   - Code to implement constants

2. *Details of mechanism by which mini-DSTs of simulated data and real data are brought to the same site for Physics Analysis

3. Manpower
   - appears to be sufficient, but could be absorbed by other projects if a need arises

should haves

4. Tracking enhancements
   - efficiency optimization
   - error estimation

5. Calorimetry reconstruction enhancements
   - Shallow angle photons in BCAL
   - Dark hits in SiPMs

6. No Legacy 3rd party software dependencies that lack long-term support
   - GEANT3/CERNLIB
   - XERCES 2.8

7. Documentation
   - Maintaining organization is challenging

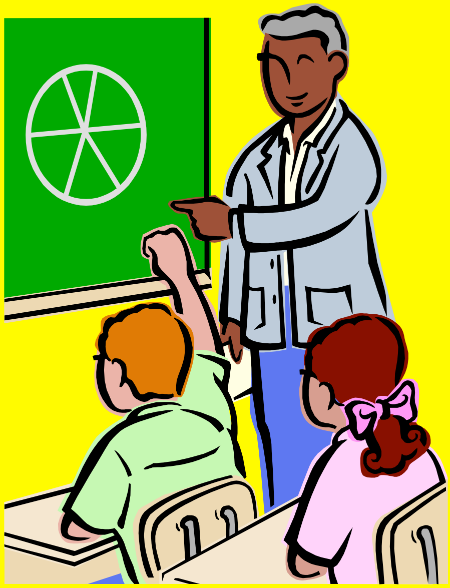
In addition, adherence to future cyber-security policies are always a concern

** Items 1 and 2 are in relatively early stages of development. They are considered higher risk at the moment due to lack of completed work so far.*

Summary

- Simulation
  - Mature GEANT3-based simulation package
  - XML Geometry definition *(HDDS)*
  - Secondary smearing + efficiency program *(mcsmear)*
- Reconstruction
  - Framework *(JANA)* has been in production use for years
    (studies involving simulated data)
  - Kalman filter implemented for charged particle tracking
  - Calorimetry reconstruction exists and is in use *(some parts being rewritten)*
  - Full event reconstruction exists including particle ID and structure to support detached vertexes
- Analysis
  - Amplitude Analysis fitting code has been written and tested on GPUs
  - Complete end-to-end test done using 3π channel
- Documentation
  - Web-based documentation exists in the form of HOWTO wiki pages and Doxygen generated HTML

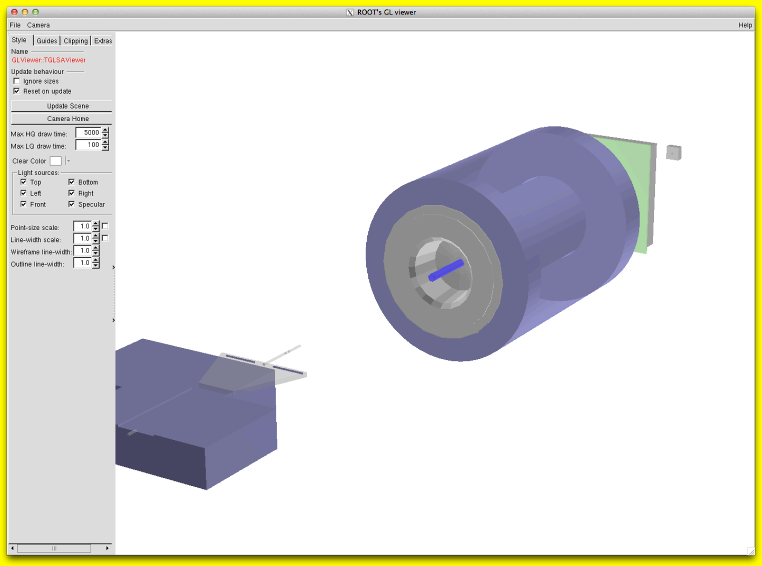
Backup Slides



HDDS

Hall-D Detector Specification

- Based on ATLAS AGDD
- Multiple XML files (*17*)
- Code generators used to convert into:
    - GEANT3-compatible FORTRAN
    - ROOT TGeo
- Designed for GEANT4 compatibility
- Accessible dynamically from JANA using *Xpath* syntax



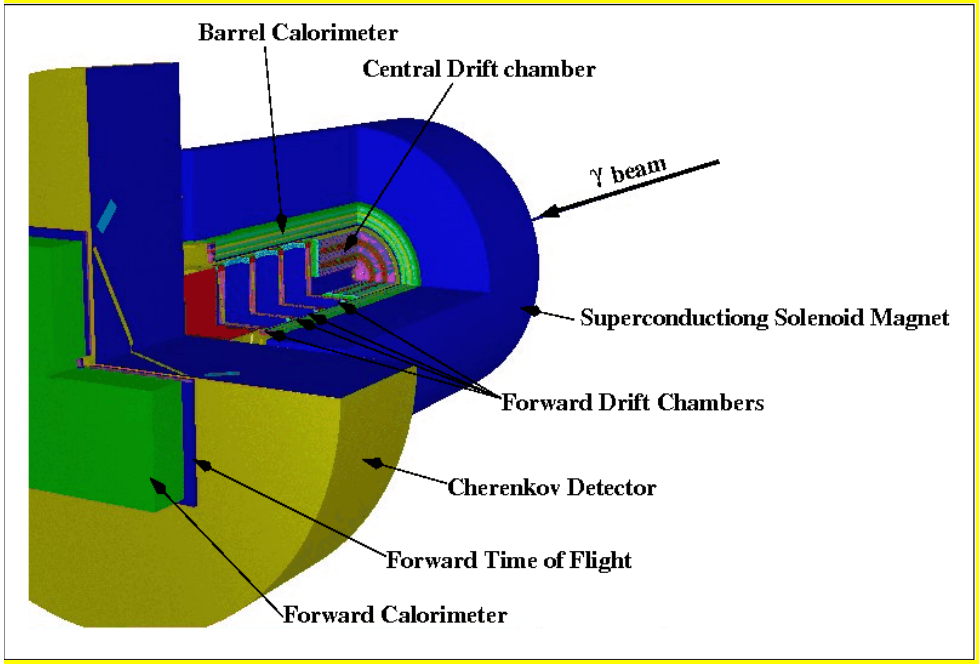
ROOT OpenGL rendering of GlueX geometry derived from HDDS. Beam comes from lower left towards upper right.

Example XML file defining Forward TOF geometry

ForwardTOF_HDDS.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE HDDS>

  Hall D Geometry Data Base: Forward TOF counter
  **********************************************

     version 1.0: Initial version   -rtj

<HDDS specification="v1.0" xmlns="http://www.gluex.org/hdds">
-->

<section name        = "ForwardTOF"
         version     = "2.0"
         date        = "2006-11-22"
         author      = "M. Shepherd, R.T. Jones"
         top_volume  = "FTOF"
         specification = "v1.0">

<!-- Origin of ForwardTOF is center of entrance plane of detector -->

<!-- upstream plane has vertical oriented bars -->
<!-- downstream plane has horizontal oriented bars -->

<!-- Fri Jan 16 08:14:30 EST 2009                                        -->
<!-- changed order of paddle counting: 1-40 bottom to top and south to north      -->
<!-- the half paddles are counted separately 41,43 north halfs, 42,44 south halfs -->

  <composition name="ForwardTOF">
    <posXYZ volume="forwardTOF" X_Y_Z="0.0  0.0 1.27" rot="0 0 -90">
      <plane value="0" />
    </posXYZ>
    <posXYZ volume="forwardTOF" X_Y_Z="0.0  0.0 3.85" rot="0 0 0">
      <plane value="1" />
    </posXYZ>
  </composition>

  <composition name="forwardTOF" envelope="FTOF">
    <posXYZ volume="forwardTOF_bottom" X_Y_Z="0.0 -66.0 0.0" />
    <posXYZ volume="forwardTOF_north" X_Y_Z="+66.0 0.0 0.0" />
    <posXYZ volume="forwardTOF_south" X_Y_Z="-66.0 0.0 0.0" />
    <posXYZ volume="forwardTOF_top" X_Y_Z="0.0 +66.0 0.0" />
  </composition>

<!-- the attribute 'row' is synonymous with 'bar' -->
  <composition name="forwardTOF_top" envelope="FTOT">
    <mposY volume="FTOC" ncopy="20" Z_X="0.0 0.0" Y0="-57.0" dY="6.00">
      <row value="21" step="1" />
      <column value="0" />
    </mposY>
  </composition>
  <composition name="forwardTOF_bottom" envelope="FTOB">
    <mposY volume="FTOC" ncopy="20" Z_X="0.0 0.0" Y0="-57.00" dY="6.00">
      <row value="1" step="1" />
      <column value="0" />
    </mposY>
  </composition>
  <composition name="forwardTOF_north" envelope="FTON">
    <mposY volume="FTOH" ncopy="2" Z_X = "0.0 0.0" Y0="-3.0" dY="6.0">
      <row value="42" step="2" />
      <column value="1" />
    </mposY>
  </composition>
  <composition name="forwardTOF_south" envelope="FTOS">
    <mposY volume="FTOH" ncopy="2" Z_X="0.0 0.0" Y0="-3.0" dY="6.0">
      <row value="41" step="2" />
      <column value="2" />
    </mposY>
  </composition>

  <box name="FTOF" X_Y_Z="252.0 252.0 2.55" material="Air" />
  <box name="FTOB" X_Y_Z="252.0 120.0 2.55" material="Air" />
  <box name="FTOT" X_Y_Z="252.0 120.0 2.55" material="Air" />
  <box name="FTON" X_Y_Z="120.0  12.0 2.55" material="Air" />
  <box name="FTOS" X_Y_Z="120.0  12.0 2.55" material="Air" />
  <box name="FTOC" X_Y_Z="252.0   6.0 2.54" material="Scintillator"
                        sensitive="true" />
```

hdgeant



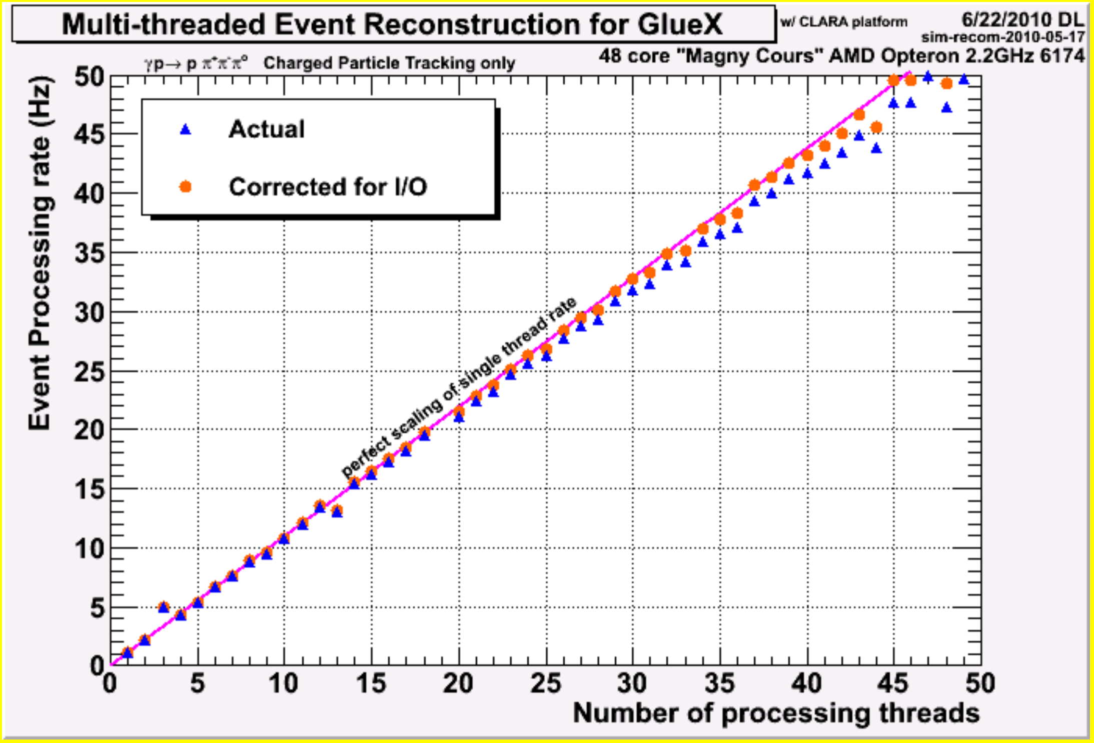
GEANT3 rendering of GlueX detector with cut-away to show inner detail. View is from the back of the detector.

- GEANT3 simulation
- Built in particle gun generator
- Can read events from external generator
- Optionally overlay EM background using built in coherent bremstrahlung generator

➢ Some detector elements are fully described. E.g. CDC straws and FDC lead glass blocks

➢ Some are modeled with a homogenous material made to match the correct number of radiation lengths. E.g. FDC cathode strips are a solid plane of copper and BCAL fiber/lead layers are solid homogenous material

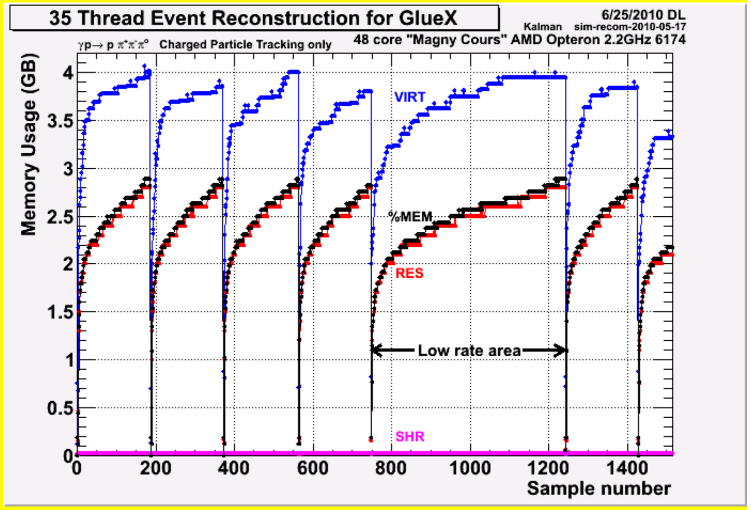
JANA Framework: Testing on a 48-core "Magny Cours"



Event reconstruction using 48 processing threads on a CPU with 48 cores generally scales quite well.

Eventually, an I/O limit will be encountered

Memory Usage vs. time while repeatedly running the 35 thread test. The marked area indicates one test where the program ran slower.

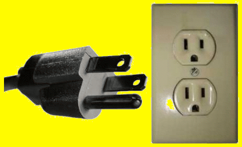


- Occasionally some problems with inexplicably lower rates.
- Program appears to simply run slower while not operating any differently.
- Unclear if this is due to hardware or Linux kernel

JANA Framework: Additional Features

JANA has numerous features:

- Event-level multi-threading (*pthreads*)
- Data-on-demand



- Plugins

- Configuration Parameters
- APIs
  - Calibration Database
  - XML Geometry (Xpath)

- Automated ROOT-tree generation (*janaroot)*

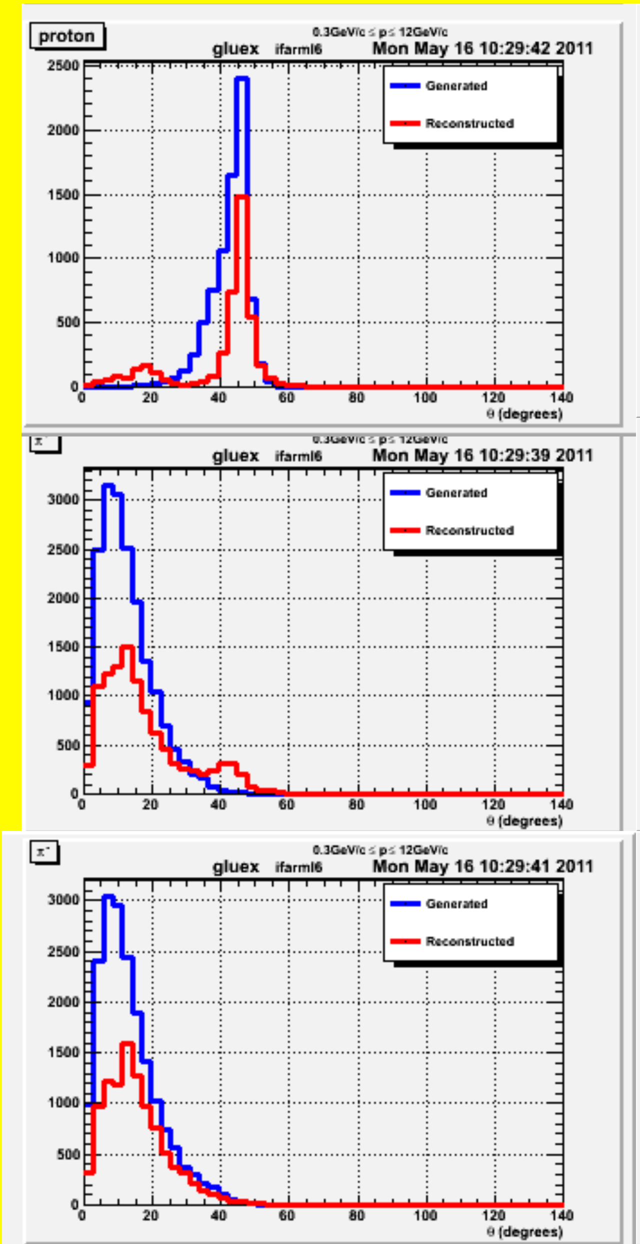
- Built-in profiling features (call-graphs and hi-res timers)

Charged Particle Tracking Progress

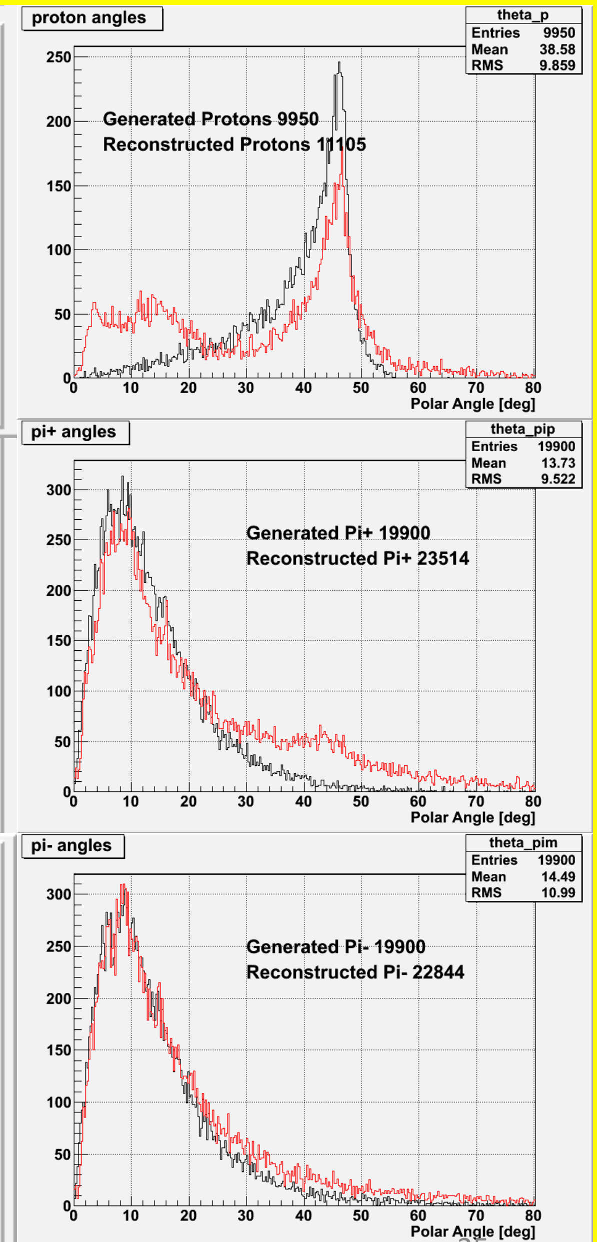
These plots are from archives of semi-weekly code builds and tests using the $b_1\pi$ channel

They indicate the progress that has been made over the past year.

May 2011

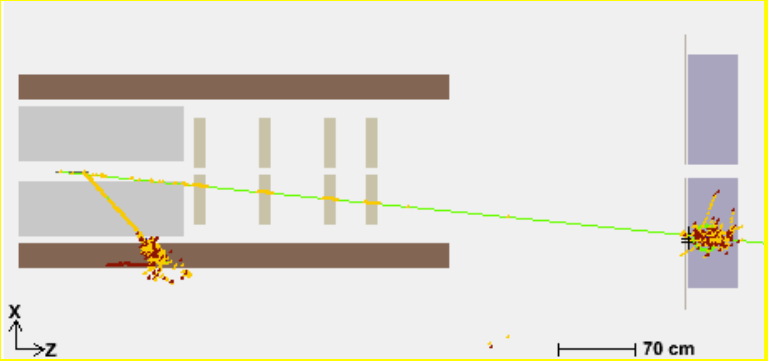


May 2012



Calorimetry Simulation

- FCAL
  - Readout threshold applied
  - Time smearing
- BCAL
  - SiPM dark hits
  - Sampling fluctuations
  - Poisson Statistics
  - Time jitter
  - Pulse shape + threshold for timing *(includes timewalk)*



Event with 2 photons from the particle gun. This is one of the views presented by the hdview2 ROOT-based event viewer.

Conditions DB Translation DB

- Most conditions information will come from EPICS
- EPICS values will be stored in an EPICS DB
- Program will be run periodically to copy relevant values into calibration DB
- Some values, such as DAQ configuration, may include only a reference value that can be used to look up details in the configuration DB maintained as part of the online systems
  (Not all configuration details need to be accessible to offline reconstruction)
- Translation DB will be used to translate DAQ id *(crate, slot, channel)* to detector id *(TOF paddle 4, top PMT)*
  - *Translation DB has yet to be developed*

DST, mini-DST

- Some work has been done on this for simulated data.
- Expect to build on this for raw data
- Multiple options for file format
  - EVIO (used for raw data)
  - HDDM (used for simulated data)
  - ROOT (easy access)
  - Other …

Simulated data resource usage rates

Processing 10k events

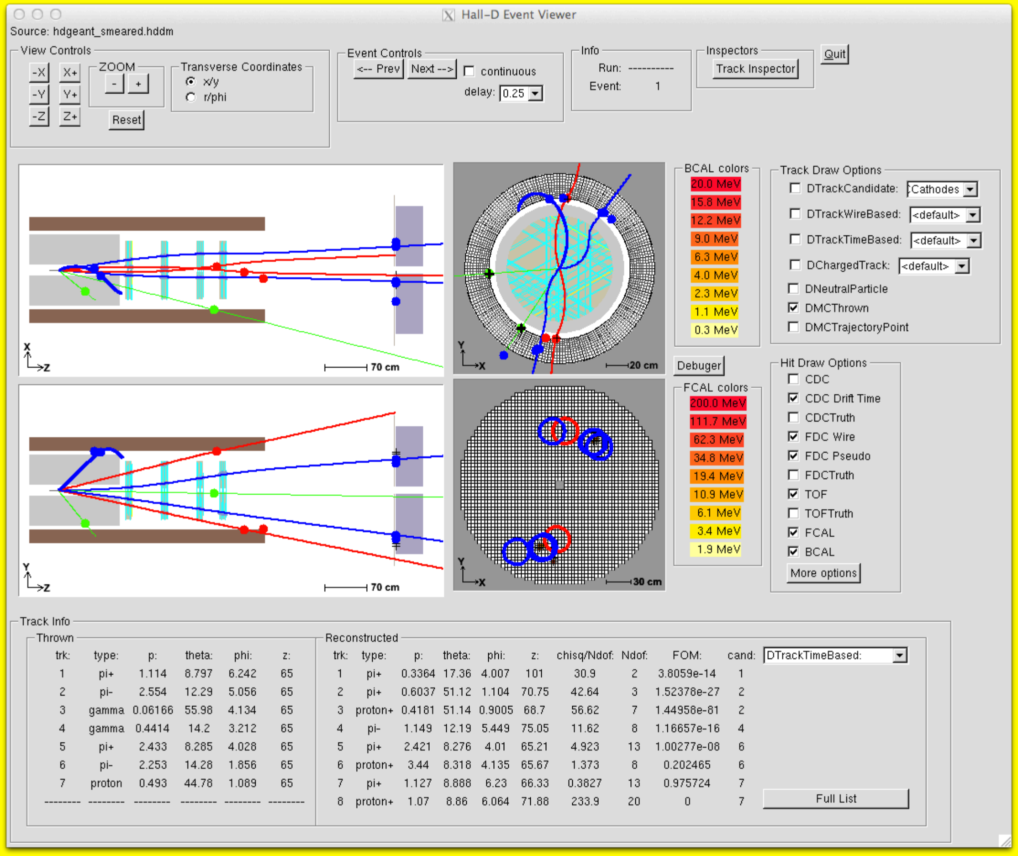
bggen	< 1 min
hdgeant	62 min
mcsmear	3 min
dana_hddm	40 min
dana_b1pi	7 min
total	112 min

Storage for 10k events

bggen	8 MB
hdgeant	670 MB
mcsmear	980 MB
dana_hddm	1.0 GB
dana_b1pi	6 MB
Compact DST	15 MB

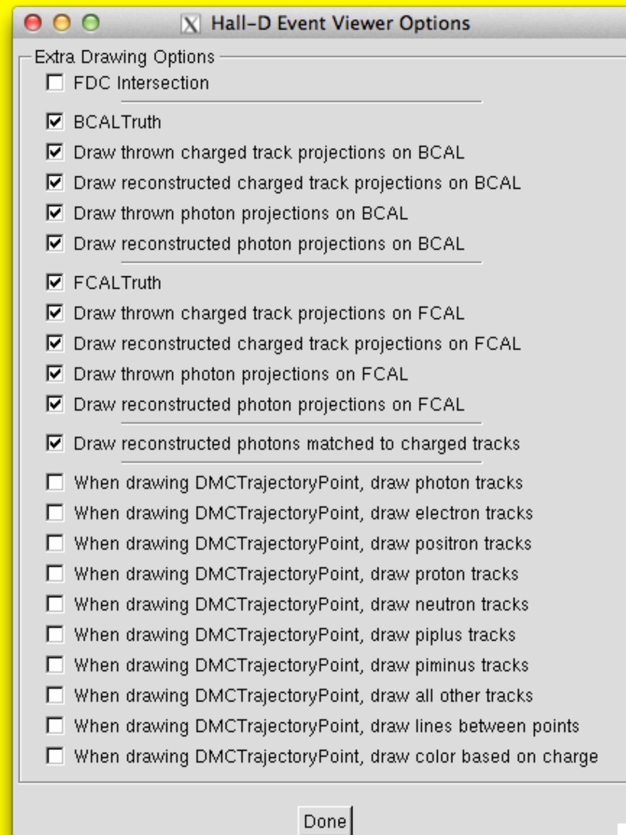
From study in fall 2011 (GlueX-doc-1849)

Event Viewer: hdview2



A b1p event displayed on the hdview2 ROOT-based event viewer. The generated trajectories are drawn. Check-buttons control which types of trajectories are drawn and can be changed dynamically.

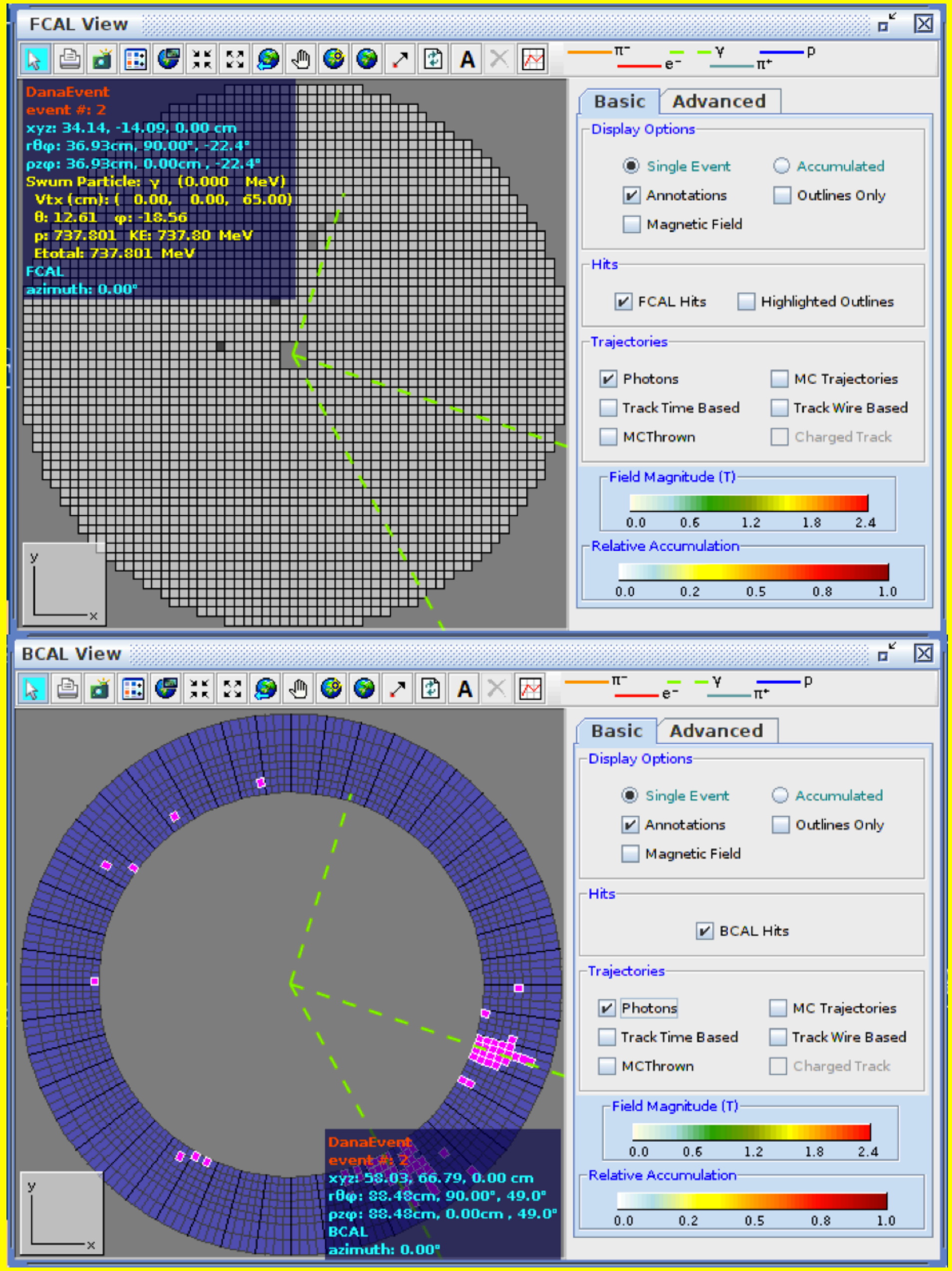
- ROOT-based
- Multiple 2-D views (non-clickable)
- Full reconstruction included
- Numerous drawing options



Event Viewer: ded

<u>Hall-**D** Event **D**isplay</u>

- Developed using bCNU framework developed in Hall-B for CLAS12
  - Basic geometry defined
  - Hits displayed for most detectors
  - Reconstructed tracks
- Summer student will continue development this summer *(Andrew Garmon)*

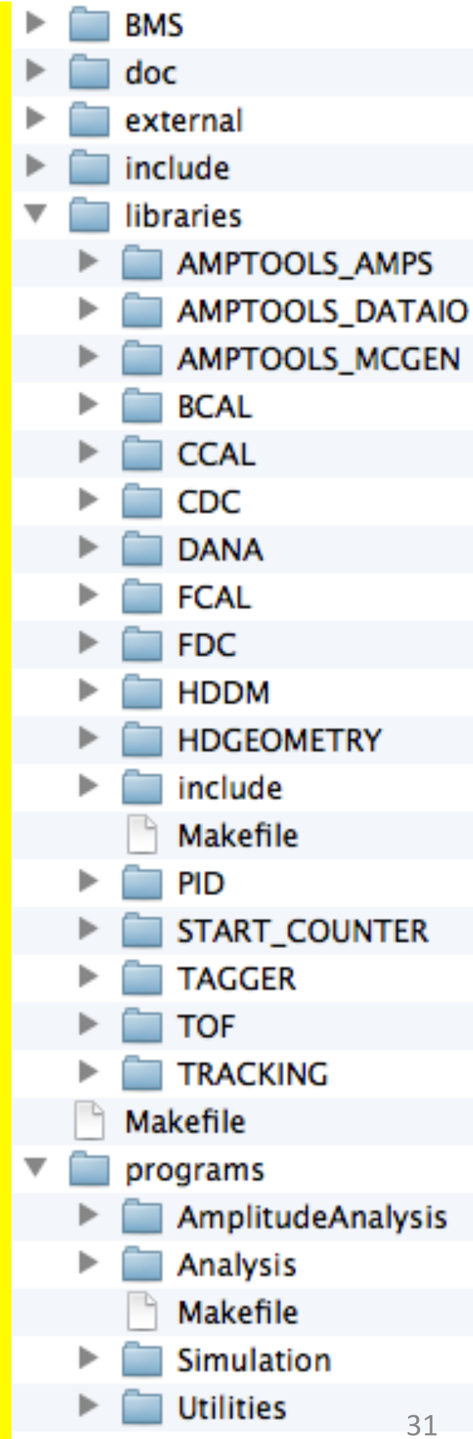


Primary difficulty is that the JAVA-based program does not couple easily to C++ reconstruction code.

Repository

<u>Source Code Repository: Subversion</u>

- Available publically via SSL
  (Anonymous checkouts from anywhere)
- Check-ins require:
  - CUE authentication
  - halld unix group membership
- Regularly tag releases
  - As needed, but that is about once every 1.5 months
- Code used in multiple executables kept in libraries



BMS (Build Management System)

- BMS is a set of Makefiles used by the GNU “make” system to build Hall-D core software
  - Generic makefiles compile all files with .c, .cc, .cpp, .cxx, or .F suffix in the current directory
  - Directories can contain code for one of:
    - static library
    - shared library (includes plugins)
    - Executables
  - Libraries have names based on directory name (e.g. *libTOF.a*)
  - Files defining a “main(…)” routine are linked as executables
    - Files without main() in the same directory are compiled and linked into all executables in that directory
    - Files with main() are linked as executables with a name that is the base name of the source file containing main() (e.g. hd_ana.cc -> *hd_ana*)

Example Makefiles using BMS

```
PACKAGES = ROOT:DANA

include $(HALLD_HOME)/src/BMS/Makefile.bin
```

Used to build one or more executables from source in the current directory. This will include compile and link flags for both ROOT and DANA.

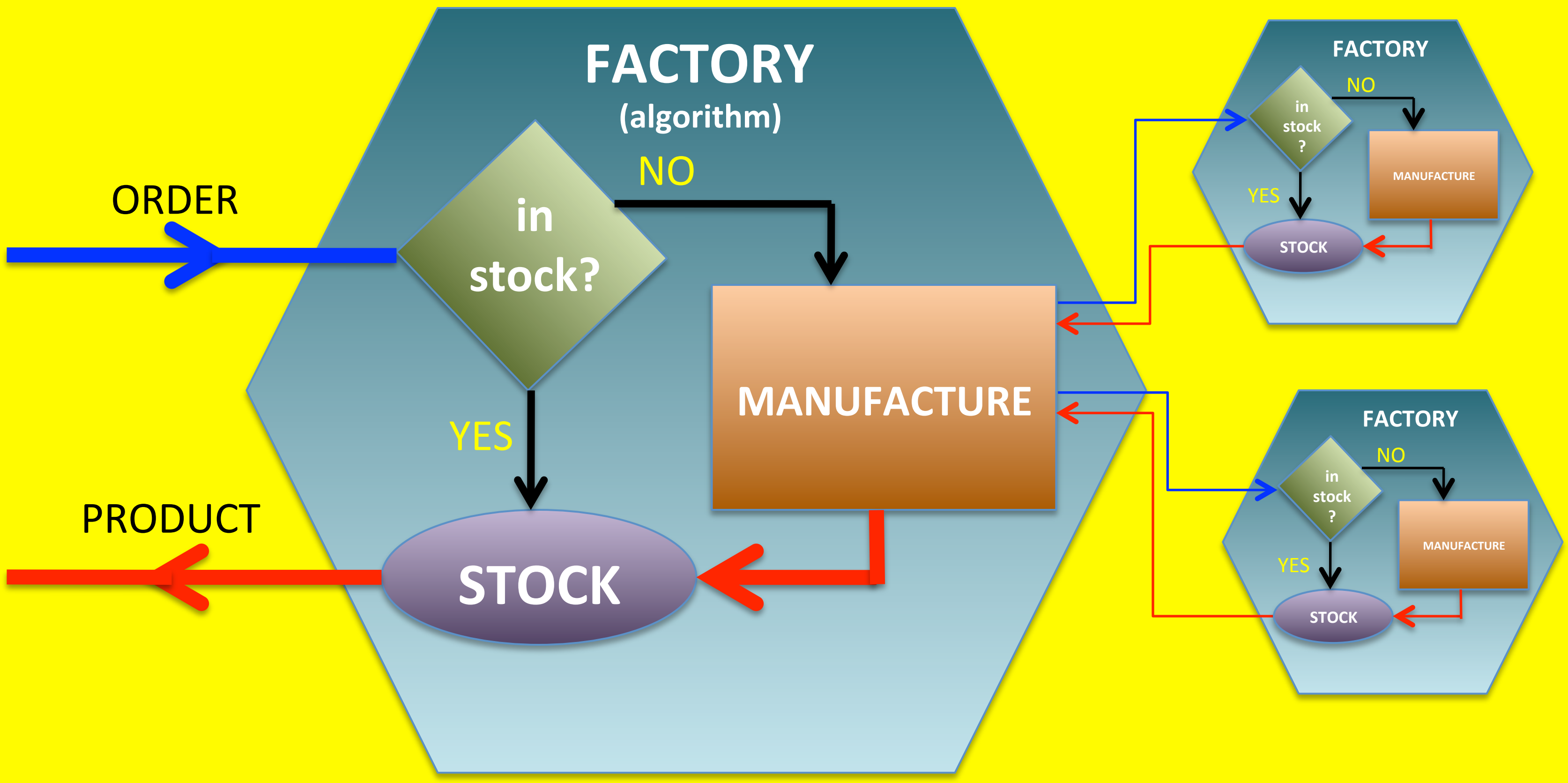
```
PACKAGES = ROOT:DANA

include $(HALLD_HOME)/src/BMS/Makefile.lib
```

Used to a static library from source in the current directory. This will include compile and link flags for both ROOT and DANA.

```
PACKAGES = ROOT:JANA

include $(HALLD_HOME)/src/BMS/Makefile.shlib
```

plugins don't typically link in the DANA reconstruction libraries so that they will be supplied by the executable. Hence, using JANA instead of DANA.

Factory Model

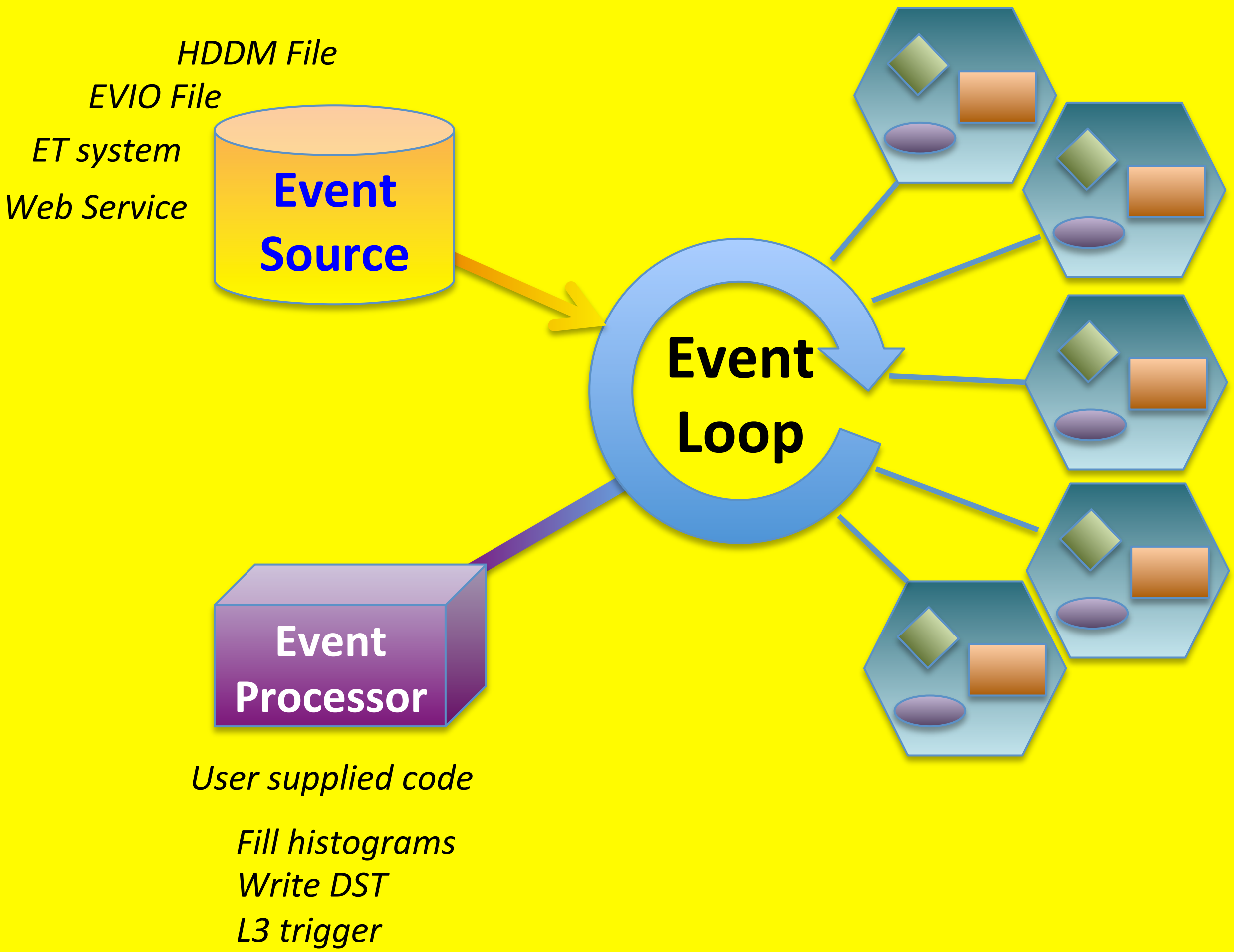


Data on demand = Don't do it unless you need it

Stock = Don't do it twice

Conservation of CPU cycles!

Complete Event Reconstruction

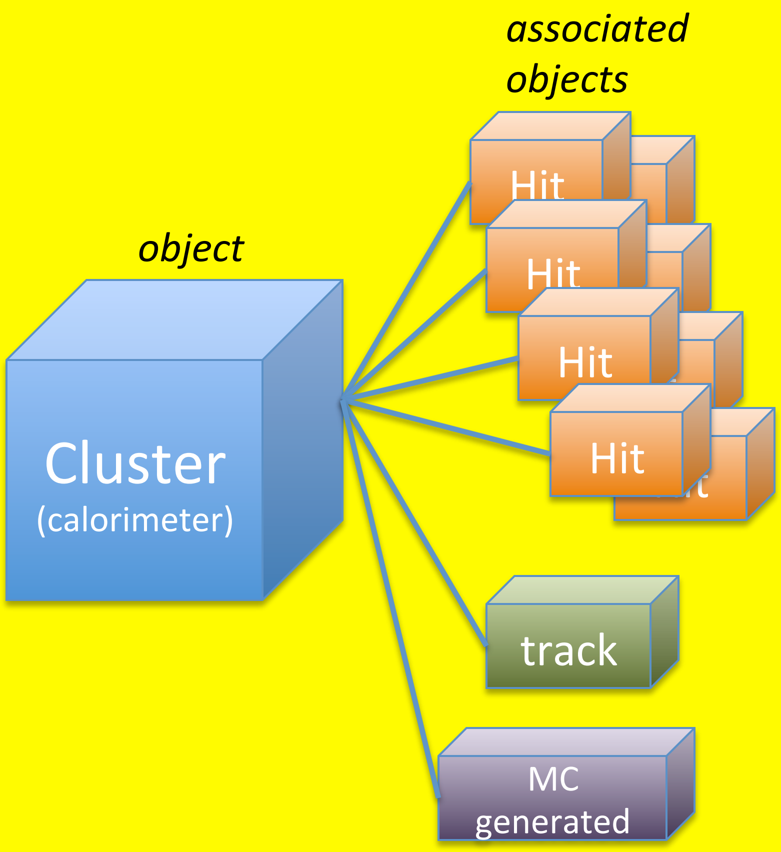


Framework has a layer that directs object requests to the factory that completes it

Multiple algorithms (factories) may exist in the same program that produce the same type of data objects

This allows the framework to easily redirect requests to alternate algorithms specified by the user at run time

Associated Objects



○ A data object may be associated with any number of other data objects having a mixture of types

- ○ Each data object has a list of "associated objects" that can be probed using a similar access mechanism as for event-level object requests

```
vector<const DCluster*> clusters;
loop->Get(clusters);
for(uint i=0; i<clusters.size(); i++)
{
      vector<const DHit*> hits;
      clusters[i]->Get(hits);
      // Do something with hits …
}
```

Configuration Parameters

in a factory's init method one might write …

Variables are data members of factory class

```
MIN_SEED_HITS = 4;
MAX_STEP_SIZE = 3.0; // cm
```

Value <u>may</u> be overwritten if user specifies a value at run time

```
gPARMS->SetDefaultParameter("TRKFIND:MIN_SEED_HITS",MIN_SEED_HITS);
gPARMS->SetDefaultParameter("TRK:MAX_STEP_SIZE" , MAX_STEP_SIZE
    , "Maximum step size in cm to take when swimming a track with adaptive step sizes");
```

***NEW:** Optional 3rd argument allows description to be stored with parameter*

- Parameters can be set via command line or configuration file
  - Complete list of parameters can be dumped using option `--dumpconfig`
    - Parameters can read in using option `--config=`*`filename`*

Multiple cores + memory

Multi-core processors are already here and commonly used. Industry has signaled that this will be the trend for the next several years. Consequence: Parallelism is ***required***

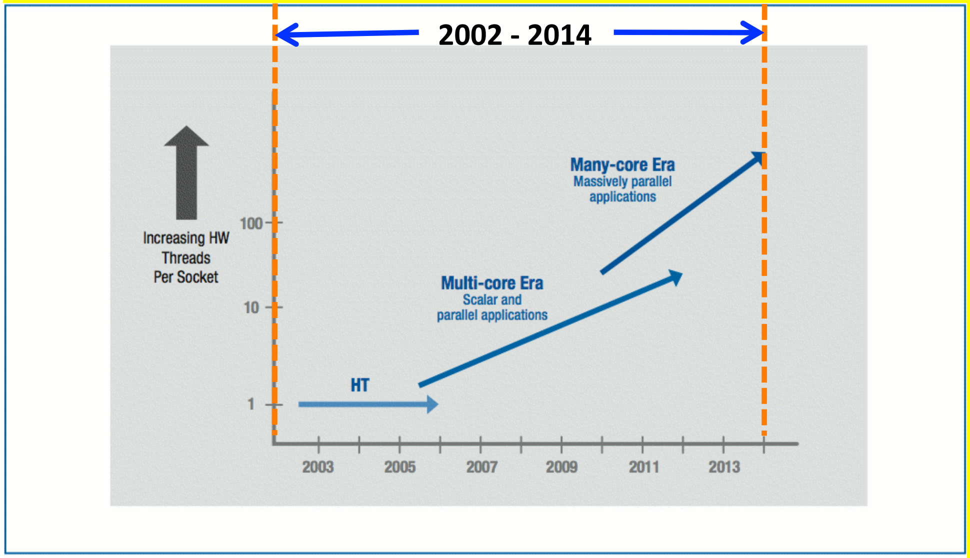
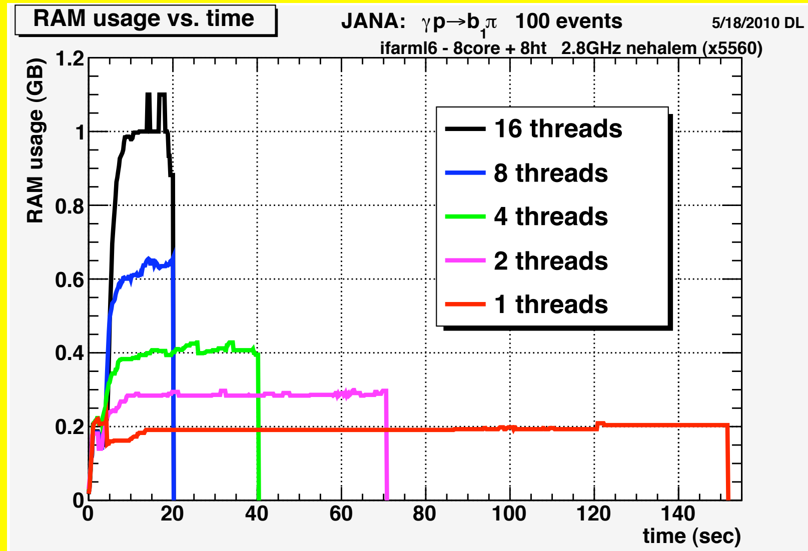
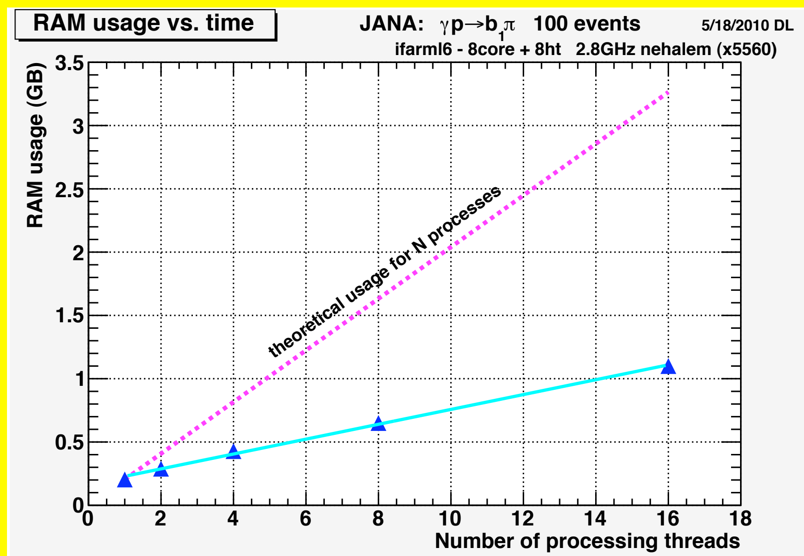


Figure 1: Current and expected eras of Intel® processor architectures





Maintaining a fixed memory capacity per core will become increasingly expensive due to limitations on the number of controllers that can be placed on a single die (#pins).

Prediction is that number of cores in the "Many-core Era" will increase faster than Moore's law adding to the difficulty in maintaining a fixed memory capacity per core.