# IceCube software Overview and IceTray Framework

Erik Blaufuss - University of Maryland

MANTS 2009

# Software in IceCube

- A tour of IceCube software
- Software challenges
- IceTray development status

# System overview - Pole

I3Live
Python

DAQ
Java, C, Perl, Python, VHDL

Calibration
Java, C

Database
MySQL, C++

SuperNova DAQ
C

Online Filter

Monitoring/ Verification

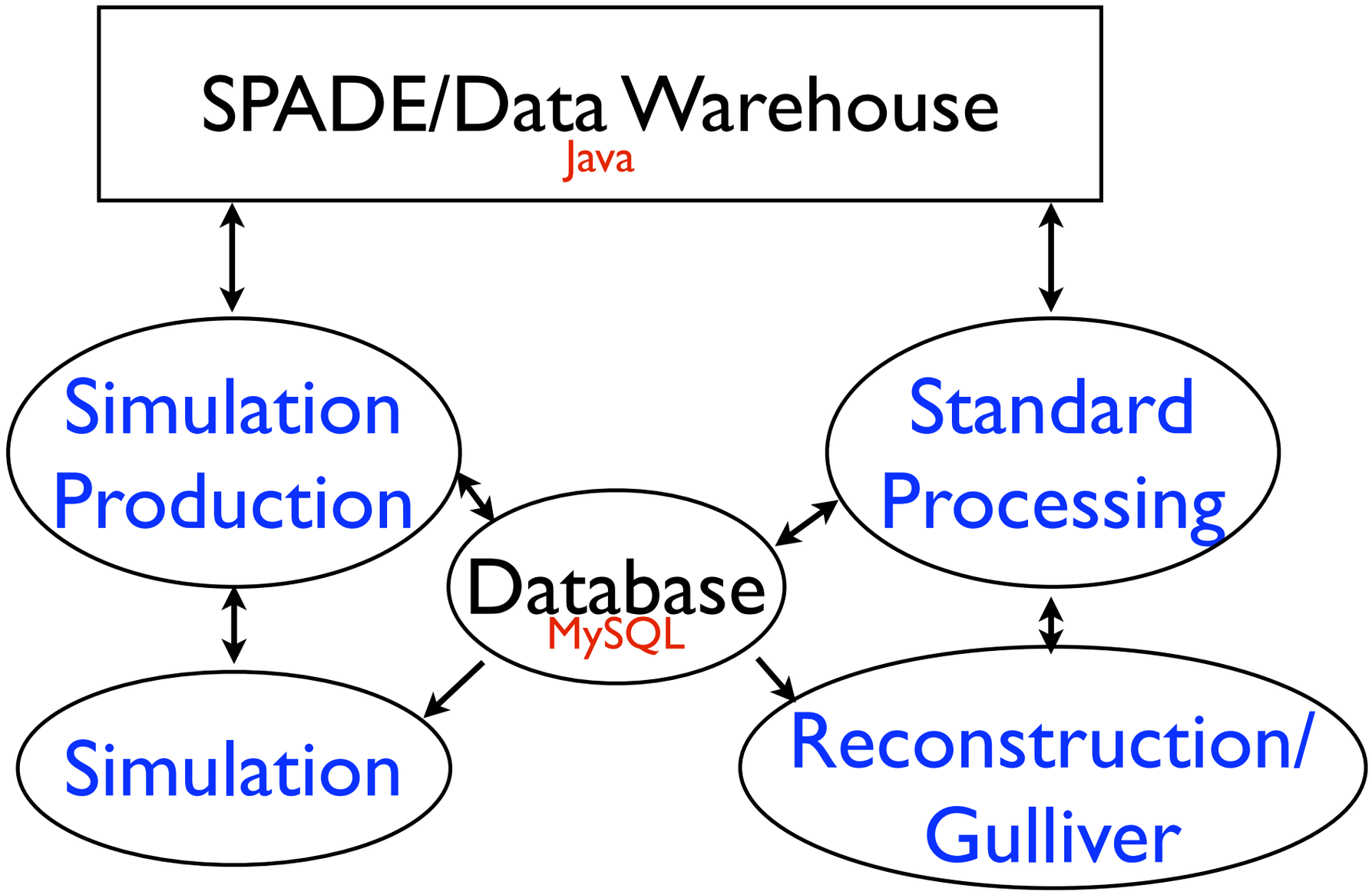
SPADE
Java

North/Data Warehouse

C++, Python

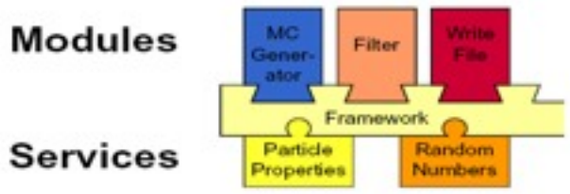
IceTray based systems drawing pieces from Reconstruction and Gulliver

# System overview - North



# Icetray framework

- Large collection of IceTray modules available in IceCube now.
  - Users free to select from existing tools and modules, chain together in a python script
    - Several reconstructions
    - Several event format decoders (I3DAQ, Amanda TWR and F2k)
    - Several useful services (DB, random, Ice, etc)
  - Very flexible, easy to put together analysis from existing tools
  - Modules extremely flexible
  - Same module: desktop user analysis, mass processing, simulation and online filtering.



# The Data

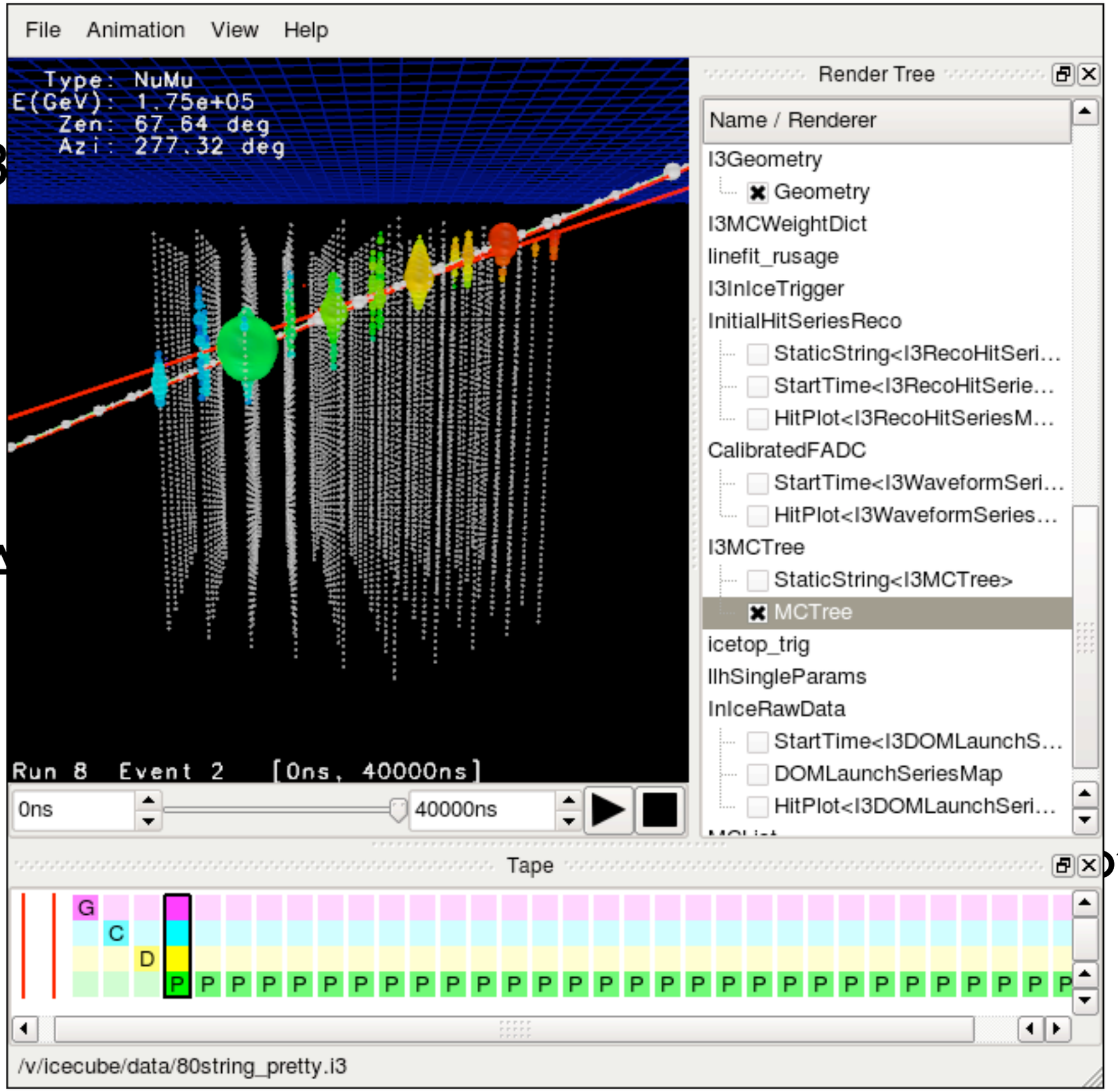
- Files
  - Data files are stored in “i3” file format
    - Boost serialized versions of classes (standard set of classes AND user defined)
    - Binary blocks of raw DAQ data.
    - Generally contain triggered events.
      - Often store “GCD” information for convenience and DB server performance
- Database
  - Store non-event data:  Geometry, Calibrations, DetectorStatus, other “constants”

# Other tools

- 3d event display
  - GLShovel can show hit selections, track reconstructions
  - Python version also being developed
- Analysis-level tools
  - Set of modules to generate ROOT files for analysis users (analysis-tree, flat-ntuple)
  - Python binding allow connection to many other tools (hdf5 tables, HippoDraw, PyROOT)

# Other tools

- 3
- 
- A
- 
- 
- other



# Other tools

- 3d event display
  - GLShovel can show hit selections, track reconstructions
  - Python version also being developed
- Analysis-level tools
  - Set of modules to generate ROOT files for analysis users (analysis-tree, flat-ntuple)
  - Python binding allow connection to many other tools (hdf5 tables, HippoDraw, PyROOT)

# Release processes

- Several "meta-projects" available for task-specific use (icerec, simulation, std-processing, jeb, etc)
  - Each is a collection of released projects based on a fixed release of IceTray ("offline-software")
    - Use svn:externals to organize release contents
- Requirements to get into a release
  - Code review of project: code standards, tests, documentation, example scripts
  - Testing by release manager ahead of release.

# Software Challenges

- Interfaces between all software systems in IceCube
  - DAQ/DOMCal/DB/IceTraySoftware needs constant attention for changes.
  - High-level overall coordination needed given heterogenous nature of systems
- Steep learning curve for new students/postdocs
  - Lots to learn coming in: Icetray/Modules/ Python/Dataclasses/Tools/DataWarehouse
  - Bootcamps have been successful in helping get started

# Software Challenges

- Avoid drawing a line between "Icetray experts" and ROOT file users.
  - Support several python interfaces and have custom modules to generate ROOT files
  - Better .i3 file to analysis level file tools needed
  - Need to encourage everyone to dig into data at lowest levels to track down bugs
- Code reviews and testing mechanisms not perfect
  - Several bugs still found, some with large impacts
  - Often hard to find experts to do this well, pressure to get into release
  - Several "older" modules predate code reviews

# Software Challenges

- Better, more uniform scripting needed for standard processing.
  - Done in several places (online filter, offline processing, simulation filtering/processing)
  - Often independently written, need more uniform
- Lots of code locked up in personal workspaces.
  - Ideally, all code and scripts needed for each analysis would be available to all, especially for published analysis.
  - Users encouraged to use SVN sandbox and include pointers to scripts in unblinding....poor response so far

# IceTray Status

- "V3" version of IceTray now available
  - Major addition of a rich python interface to modules, data files and classes.
    - Python-like module interface to Tray.
    - Several bug fixes and added new features.
    - Greatly improved documentation set
- These are additions to the current interfaces.
  - With a few minor exceptions, old scripts, modules will work without changes

# An example Python Module

- I3Modules can be written in python as well.

```
from icecube import icetray, dataclasses

class MyModule(icetray.I3Module):

    def __init__(self, context):                          ## Constructor
        icetray.I3Module.__init__(self, context)
        self.AddParameter('OutputName',                   # name
                          'Where to get input',           # doc
                          'FastReco')                     # default

    def Configure(self):
        self.outputname = self.GetParameter('OutputName')

    def Physics(self, frame):
       geo = frame.Get("I3Geometry")
       hits = frame.Get("FEHits")
       particle = dataclasses.I3Particle()
       ## Do something with the geo/hits -> fill particle attr.
        frame[self.outputname] = particle      # put it in the frame
        self.PushFrame(frame)                  # push the frame
```

# Additional python friendliness...

- Simple python functions can be easily used as Modules, and in this case, an event filter:

```
def reco_cut(frame,  particle,  threshold):
    frameval = frame[key].GetZenith()/I3Units.degree
    return frameval > threshold

tray.AddModule(reco_cut,
               key = 'MyReco',
               threshold = 80)
```

- Modules can execute conditionally based on a python function:

```
tray.AddModule('ComplexReco', 'MultipleIteraions',
               If    = lambda frame: 'ComplexReco_flag' in frame)
```

# Even more python friendliness...

```
In [1]: from icecube import icetray, dataclasses, dataio

In [2]: file = dataio.I3File("selectedEvents.i3")

In [3]: frame = file.pop_physics()

In [4]: print frame
[ I3Frame  (Physics):
  'DrivingTime' [Physics] ==> I3Time (38)
  'I3EventHeader' [Physics] ==> I3EventHeader (83)
  'I3PfFilterMask' [Physics] ==> I3PfFilterMask (38)
  'I3SkipNEventFilter' [Physics] ==> I3Bool (27)
  'I3TriggerHierarchy' [Physics] ==> I3Tree<I3Trigger> (126)
  'IceTopRawData' [Physics] ==> I3Map<OMKey, std::vector<I3DOMLaunch, std::allocator<I3DOMLaunch> > > (46)
  'InIceRawData' [Physics] ==> I3Map<OMKey, std::vector<I3DOMLaunch, std::allocator<I3DOMLaunch> > > (11000)
]

In [5]: hits = frame.Get("InIceRawData")

In [6]: print hits
<icecube.dataclasses.I3DOMLaunchSeriesMap object at 0x5d5df0>

In [7]: nchannel = len(hits)

In [8]: nchannel
Out[8]: 10
```

- I3 files can be opened and worked with interactively from a python prompt:

PyROOT, Hippodraw, matplotlib, etc. all immediately usable via python interface

# IceTray development status

- No major features known to be needed at this time.
  - Support bugfix and general maintenance releases as needed
  - Email developers, submit a ticket for requests/bugs
- Help available
  - Docs  (Sphinx/doxygen/icetray-inspect buildable)
    - http://software.icecube.wisc.edu/offline-software.trunk/
  - Bug tracking, separate SVN repository
    - http://code.icecube.wisc.edu/projects/icetray
  - Email developers
    - icetray-dev@icecube.umd.edu