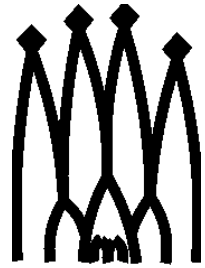


Gaudi Framework Overview and The Daya Bay Experience

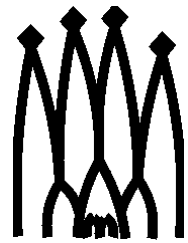
Brett Viren

Physics Department

BROOKHAVEN NATIONAL LABORATORY

Future Neutrino Software, FNAL 2009/3/13

Outline



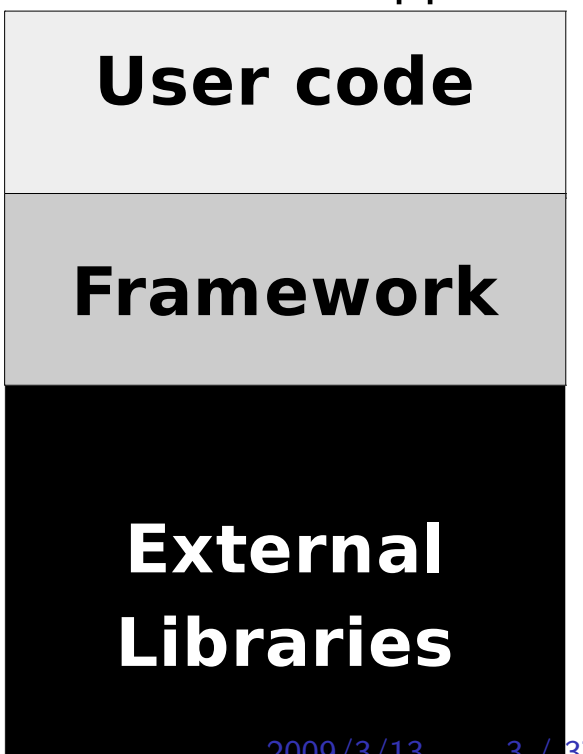
Framework Context

Analogous to a building's framework, a software framework:

- supports the intended use of its context.
  - Defines the playground and helps the user play in it.
- provides well defined entries for user code to exploit.
- reduces the amount of details users must understand.
- supports collaborative development through standardization.

A framework insinuates itself between user code and an ocean of support libraries.

- User code: reconstruction, simulation, analysis
- Provides self-consistent set of features required to support user code.
- Low level external packages (eg. ROOT, Python, Geant4, CLHEP, BOOST) that the framework uses to deliver the features.



Gaudi's Provenance

Select moments in Gaudi history[1]:

- Development started by/for LHCb starting Sept 1998.
- First adoption by other experiments '99-'00
  - ATLAS: Gaudi + ATLAS-specific code = "Athena".
  - FGST (neé GLAST), HARP
- Recent re-convergence of divergence between LHCb and ATLAS
- Newbies: BES III, MINERνA, Daya Bay

Community centers:

- Savannah project: `https://savannah.cern.ch/projects/gaudi/` bugs, patches, repository, developer mailing lists
- General discussion mailing list: `gaudi-talk@lists.bnl.gov`. `https://lists.bnl.gov/mailman/listinfo/gaudi-talk` open to the public, please join if interested.

[1] Thanks go to Pere Mato, any omission is my fault.

Gaudi Components

One way to describe Gaudi's organization is as a set of cooperative software **components**. There are a number of component categories defining general behavior. A sample:

Service	A shared computational or data resource.
Algorithm	A modular unit of processing.
AlgTool	A sub-algorithm that can be shared.
Converter	Converts information between different representations.
Property	Key-value pairs for user configuration of other components.

Concrete examples to follow....

Component Interfaces

- Specific behavior (class methods) are defined through special, pure-virtual abstract Interface classes.
- Concrete components inherit from one or more Interface classes (directly or through a base providing partial implementation).
- Most interaction with a component is done through its Interface(s) and not through the concrete class.

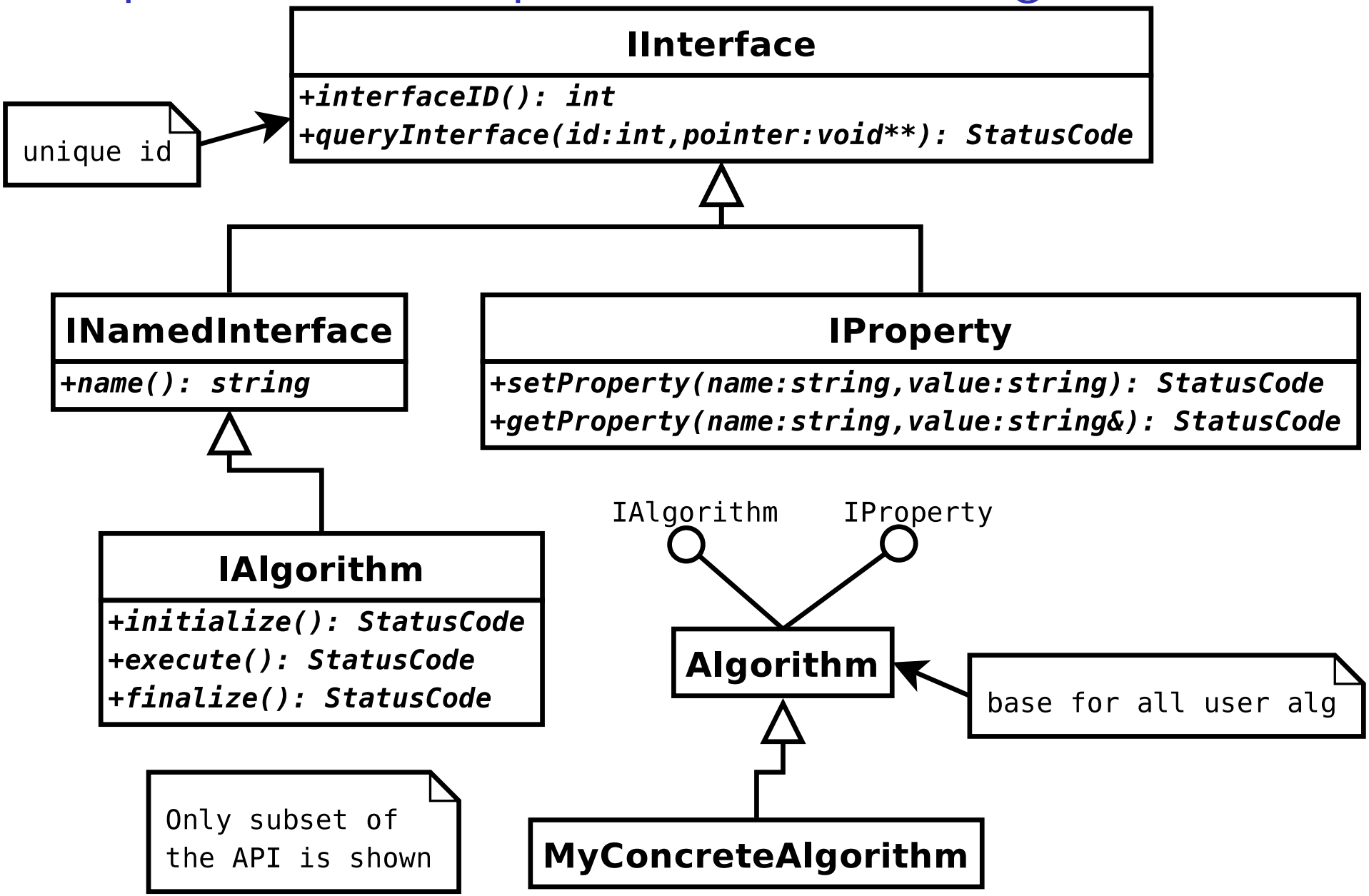
Using Interfaces provide these benefits:

- Greatly reduces compile time and dependencies.
- Supports configuration time plugin mechanisms.
- Clarifies concrete class's responsibilities and behavior.
- Simplifies access of components from Python.
- Provides a cross-casting mechanism (`queryInterface()`) faster than `dynamic_cast`.

But, they do require a little extra work:

- Some boilerplate coding required.

Example of some Component Interfaces - Algorithm



Algorithms

The main thing users care about.

- User code's primary entrance into the framework.
- User implements (at least) `Algorithm::execute()`, which is entered once per "event".
- Additional methods, eg: `beginRun()`, `beginRun()`
- Base Algorithm class provides many helper methods to access common services - simplify user code.

Types of algorithms:

- Analysis algorithms producing histograms or ntuples.
- Event reconstruction, selection or simulation
- Driving framework services, such as file I/O.
- Sequence algorithm to let sub-chain of algorithms run or implement processing branches.

Examples of Services

Some Services:

Event Data Service Manages the Transient Event Store (TES), an in-memory map of a unix-like path string to an object.

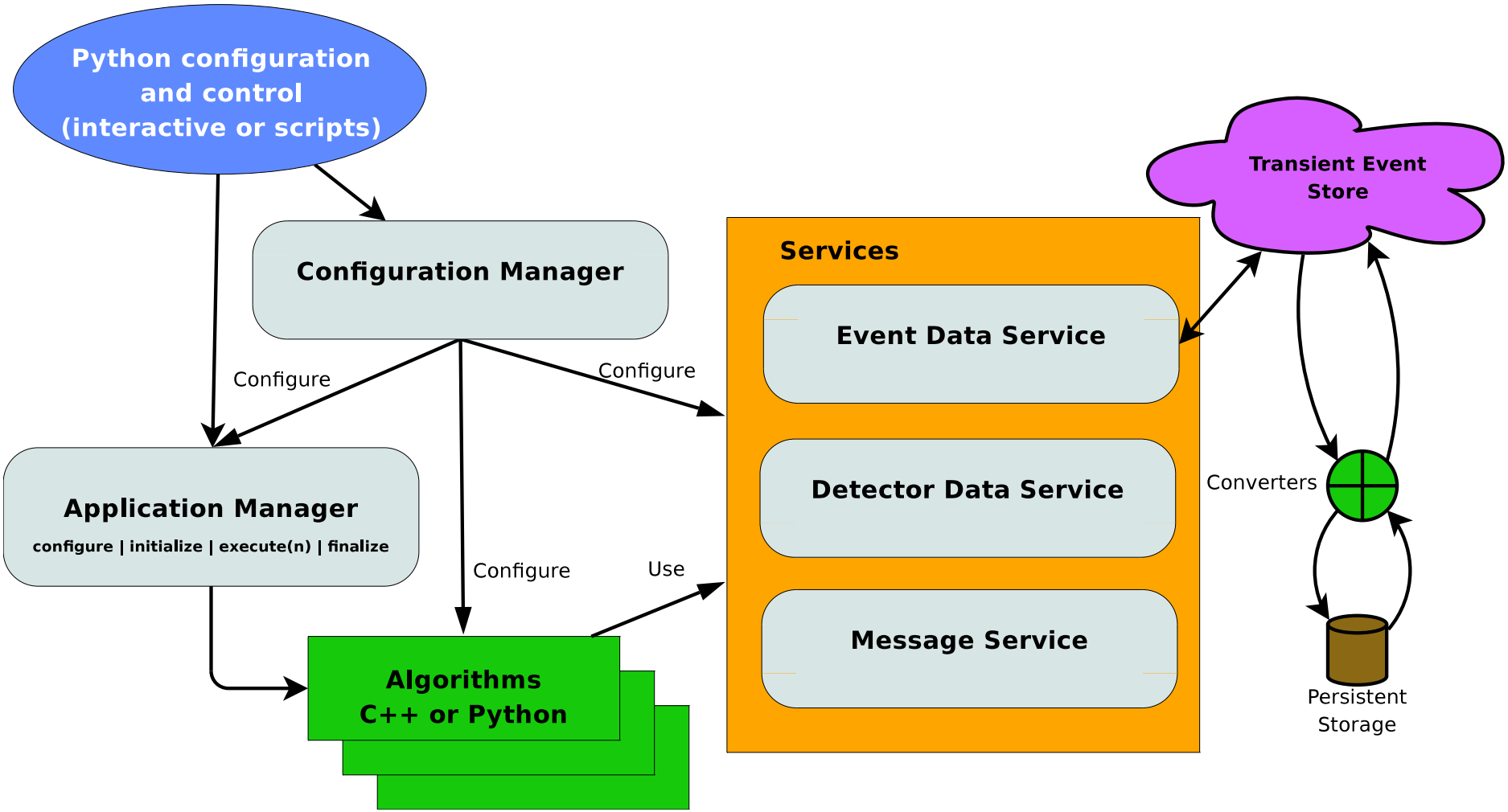
Message Service Logging messages at different importance levels.

Random Numbers Unified access streams of pseudo random numbers.

Detector Data Service Unified (MC and analysis) access to detector geometry and material properties. (more later)

Detector Simulation Manager Geant4 application accessed through Gaudi AlgTool components. (more later)

Example Job Makeup



Simplified picture, typically more components in a real job.

Job Configuration with Python

Jobs are configured using Python[2]

- Every component has a Python shadow class ("`Configurable`") to hold configuration information.
- `Configurables` self-register with a `ConfigurationManager`.
- The ConfigurationManager finally applies the configurations to the real C++ components.

`Configurables` are Python objects and let you fully program the configuration of a job. You are not limited to some little, custom language, CINT's marginal C++ or hand-parsing string values.

[2] An older C++-like text format is also available.

Example Configuration of an Algorithm

```
# These are CLHEP units, same as on C++ side
import GaudiKernel.SystemOfUnits as units

# Get configurable object for MyAlg algorithm
from PackageName.PackageNameConfig import MyAlg
ma = MyAlg()
ma.EnergyCut = 6*units.MeV

# Get AppMgr and add the alg instance to the TopAlg list
from Gaudi.Configuration import ApplicationMgr
appMgr = ApplicationMgr()
appMgr.TopAlg.append(ma)
```

This is low level. Per-package Python modules can provide sane defaults that can be easily customized by the user.

Gaudi Build and Runtime

Gaudi build and runtime environment is managed by CMT[3].

- CMT is a meta-make build system with the concept of "projects" holding "packages" and is becoming a de-facto HEP standard.
- Gaudi is a CMT project containing its various packages.
- Experiments using Gaudi likely want to use CMT for their own code.
- Relies on a second project, LCGCMT, which primarily supplies CMT "glue" packages to integrate external packages.
- There are many external packages required depending on how much of Gaudi one adopts.
  - Minimal (no POOL for I/O): AIDA, Boost, GSL, HepMC, HepPDT, Python, XercesC, CLHEP, Geant4, ROOT.
  - POOL I/O brings in others, including products from CERN.
  - Can get binary externals from CERN for some platforms or can build from source.

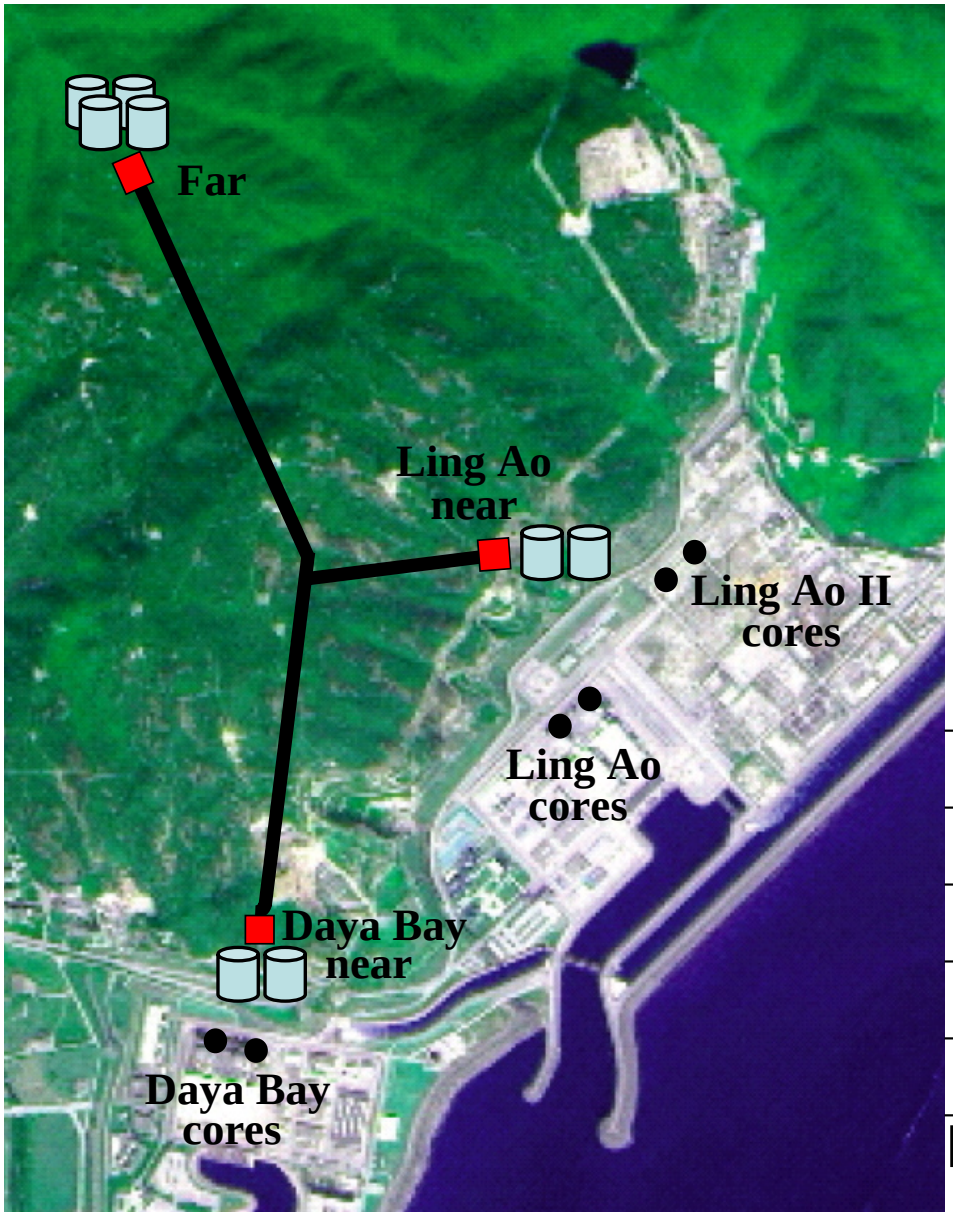
[3] `http://www.cmtsite.org/`

Gaudi Software Repositories

Gaudi's software repository recently transitioned:

- Previously in CVS:
  `http://isscvs.cern.ch/cgi-bin/cvsweb.cgi/?cvsroot=Gaudi`
  `CVSROOT=:pserver:anonymous@isscvs.cern.ch:/local/reps/Gaudi`
- Now in SVN:
  Anonymous read-only: `https://svnweb.cern.ch/guest/gaudi`
  Authenticated read-write: `svn+ssh://svn.cern.ch/reps/gaudi`
  See: `https://twiki.cern.ch/twiki/bin/view/Gaudi/GaudiSVNRepository`

Daya Bay Experiment - Disappearance Search for $\theta_{13} \neq 0$



- ~200 Collaborators from Asia, USA & Europe
- Located in ShenZhen, China.
- Detect $\bar{\nu}_e$ from nearby Daya Bay nuclear reactor plant.
- 4 cores @ 11.6 GW (+2 in 2011)
- 3 detector sites:

	2 Near	1 Far
$\bar{\nu}_e$ targets	2×20 ton	4×20 ton
BL (m)	360/500	1600/1900
$\bar{\nu}_e$/day	~1200	~350
B/S ratios	~0.4% @	~0.2%

Raw data rate: 80 TB/year.

Daya Bay Computing Model

- Computing centers: IHEP (China), BNL (RACF + local cluster), LBL (PDSF) + couple major universities.
- Mix of 32 & 64 bit Linux (various) + Mac OS X (no Windows!)
- Data tape archiving at IHEP & LBL,
  complete, reduced sample on disk @ BNL
- Geographically diverse developers (timezones are a killer!)
- Supporting infrastructure:
  - SVN and Trac[4] hosted by IHEP
  - Mailing lists and MySQL database hosted by LBL
  - MediWiki and Doxygen hosted by BNL

[4] A truly useful bug tracker!

Considerations in Choosing a Framework

- Daya Bay is a relatively small/simple experiment
- Short on time and developers
- Initial simulation & analysis that was centered on a monolithic, organic Geant4-based application.
  - Well suited for initial results, fast turn-around
  - Brought us to the TDR stage
  - Obvious we could not continue with that "design"
- Considered:
  - IceCube's IceTray framework,
  - MINOS's framework,
  - Starting fresh and
  - Gaudi (obviously)

Gaudi provided the most general framework, already proven to be useable by "outside" experiments $\Rightarrow$ best chance for success.

Serious Concerns

Many collaborators expressed **Serious Concerns**™®

- Gaudi is too complicated!
  - Spoke to **users** of Gaudi in BNL's ATLAS group: "easy for users, needs experts for initial integration and any core development".
- Gaudi does more than we need!
  - Don't use the parts that are not needed
- Gaudi is a black box!
  - Well, for a user, that is sort of the point.
  - And, for a "core" software developer, it isn't **that** hard to understand the internals!
- Gaudi and its dependencies are hard to compile!
  - Damn right they are! This was probably the largest technical hurdle.
  - You'll get over it.

Adopting Gaudi and Friends

Adoption strategy:

- Want to be able to build everything from source
  - We have platforms that were/are not supported by CERN
- Want to leverage the good work of others as much as possible (steal before write).

Adoption overview:

- Core Gaudi but no POOL related packages (I failed miserably to build POOL in its pre-CMT days, probably better now).
- Application-agnostic packages from LHCb:
  - GaudiObjDesc - define data model classes
  - DetDesc - XML based detector geometry and materials description
  - GiGa - interface Geant4 to Gaudi
  - Panoramix - geometry visualization, maybe one day event display
  - Some interest to move some of this into Gaudi proper.
- Only the externals needed for the above.

Compilation of Externals, Gaudi and Other Projects

Much more difficult than expected, most problematic were the externals. Learn the "gotchyas" and it's smooth. Ended up with install script that:

- Pulls down CMT, builds and installs
- Downloads and installs all external packages
  - Each package handled by a dedicated sub-script
  - Driven by LCGCMT requirements files, adapts to new versions.
  - Allows patching or platform-specific builds where needed
- Checks out and builds CERN's CMT based projects: LCGCMT, Gaudi, LHCb and Relax
- Checks out and builds Daya Bay's CMT project.

Notes:

- All CMT projects are maintained in Daya Bay's SVN repository, sync'ed to upstream via GIT, merging any minor local mods.
- From empty directory to full install is $\sim$3 hours, depending on CPU, disk and network speeds.

Trac, Nose and Bitten, Oh My!

- Bug/feature tracking
- Milestones, timeline, wiki
- Repository browsing
- Commit-triggered compilation and unit testing

12:07 Build of detdesc [5670] on i686-slc46-gcc346 completed

11:13 Changeset [5670] by bv

dybgaudi/trunk/Detector/DetHelpers/python/DetHelpers/TestCoordSysSvc.py

Remove stupid last minute bug addition.

05:51 Build of dybinst [5669] on i686-slc46-gcc346 failed

Step *test-rootio* failed

Ticket	Summary	Component	Version	Milestone	Type	Priority	Created
#10	XmlDetDesc should provide for differing near detectors	dybgaudi//Detector	trunk		enhancement	major	09/12/2008
#101	Provide for split up output files	dybgaudi//RootIO	trunk	NuWa 2.0.0 - Major release for AD Dry Run	enhancement	minor	11/13/2008
#102	Add IProperty interface to converter base class.	dybgaudi//RootIO	trunk		enhancement	minor	11/14/2008
#103	Inter-HeaderObject references can not be persisted when they span RegistrationSequences	dybgaudi//Simulation /Fifteen	trunk		defect	major	11/14/2008
#117	RegistrationSequence converter should ignore select streams	dybgaudi//RootIO	trunk		enhancement	minor	12/10/2008
#119	RunInfo objects, where/how to store.	dybgaudi//	trunk	NuWa 1.5.0 - Release for Online/Offline	defect	major	12/10/2008

Main Software Elements

Currently we have built up:

- Data Model Classes (LHCb's `GaudiObjDesc`)
- Detector Geometry and Materials Description (LHCb's `DetDesc`)
- Simulation:
  - Kinematic Generators (homegrown)
  - Detector Simulation (LHCb's `GiGa`)
  - Electronics Simulation (homegrown)
  - Trigger Simulation (homegrown)
  - DAQ Readout Simulation (homegrown)
- Initial calibration (homegrown)
- Initial reconstruction (homegrown)
- File I/O (homegrown)
- Database Interface (stolen from MINOS, thanks!)

Data Model Definition with GauidObjDesc

GaudiObjDesc:

- XML based class schema definition
- Python parser producing:
  - Header file
  - Reflex ROOT/Python dictionaries
- Hand coded implementation files allowed or can inline code in the XML.

Why use?

- Produces consistent class definitions, conventions
- Automates some aspects
- Hoped to also generate persistent classes, but not yet

Detector Geometry and Materials Description - DetDesc

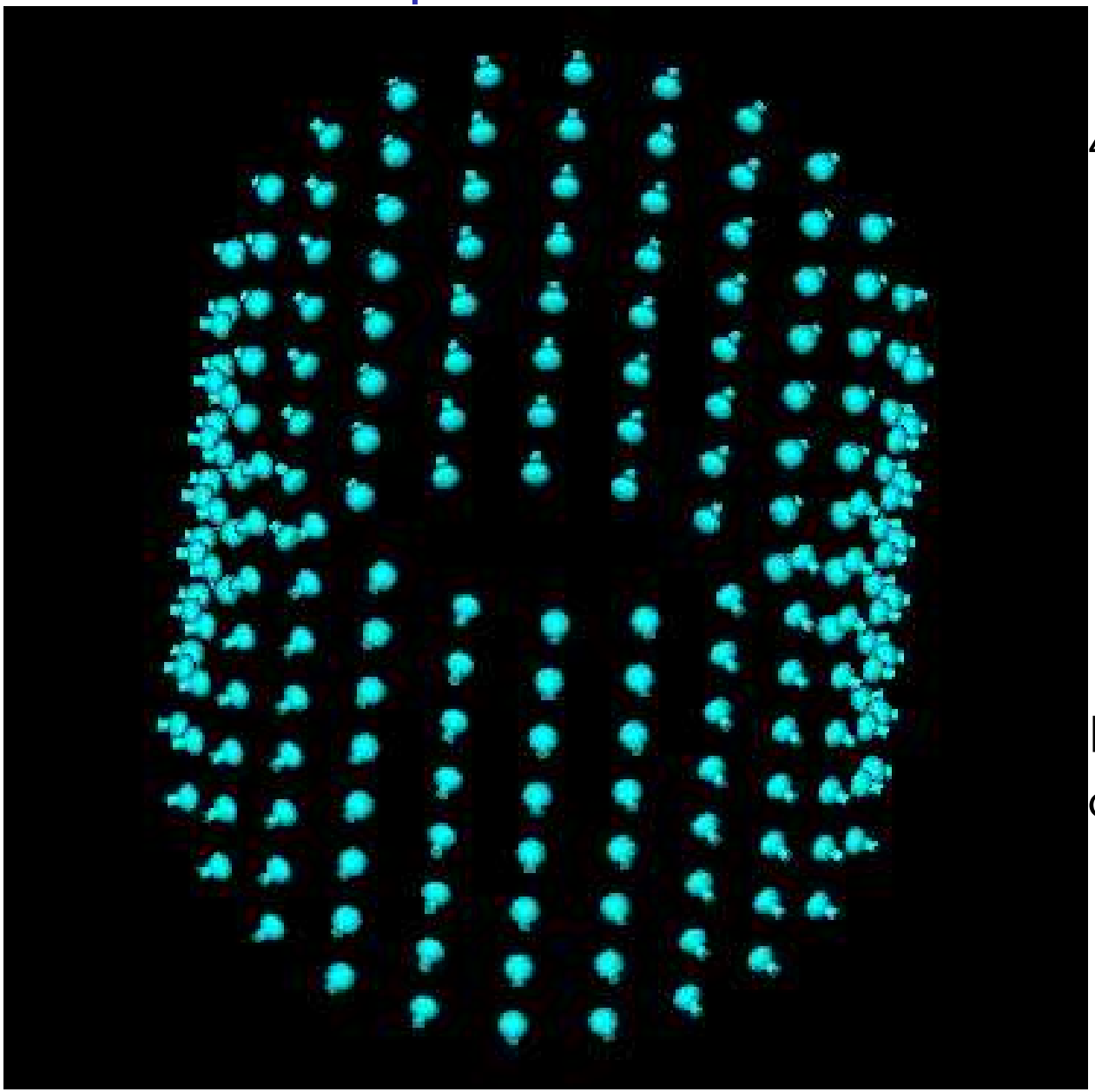
DetDesc:

- Expressive and concise XML description
- Includes multi-placements, parameters and user-extensibility.
- Supports offsets from ideal geometry.
- Three categories of description:

  Geometry Logical/physical volumes
  Structure Touchable volumes
  Materials Materials and their optical properties
- XML parsed into DetDesc objects in the Transient Detector Store
- User code has full access to all info
- Get DetDesc $\rightarrow$ Geant4 converters "for free".

Such easy flexibility is crucial for detector R&D!

DetDesc Example - $\bar{\nu}$-detector PMT Array



4 steps:

1. Position one PMT at bottom
2. Copy to make ring of 24 PMTs
3. Copy ring 8 times
4. Rotate everything 1/2 angular period.

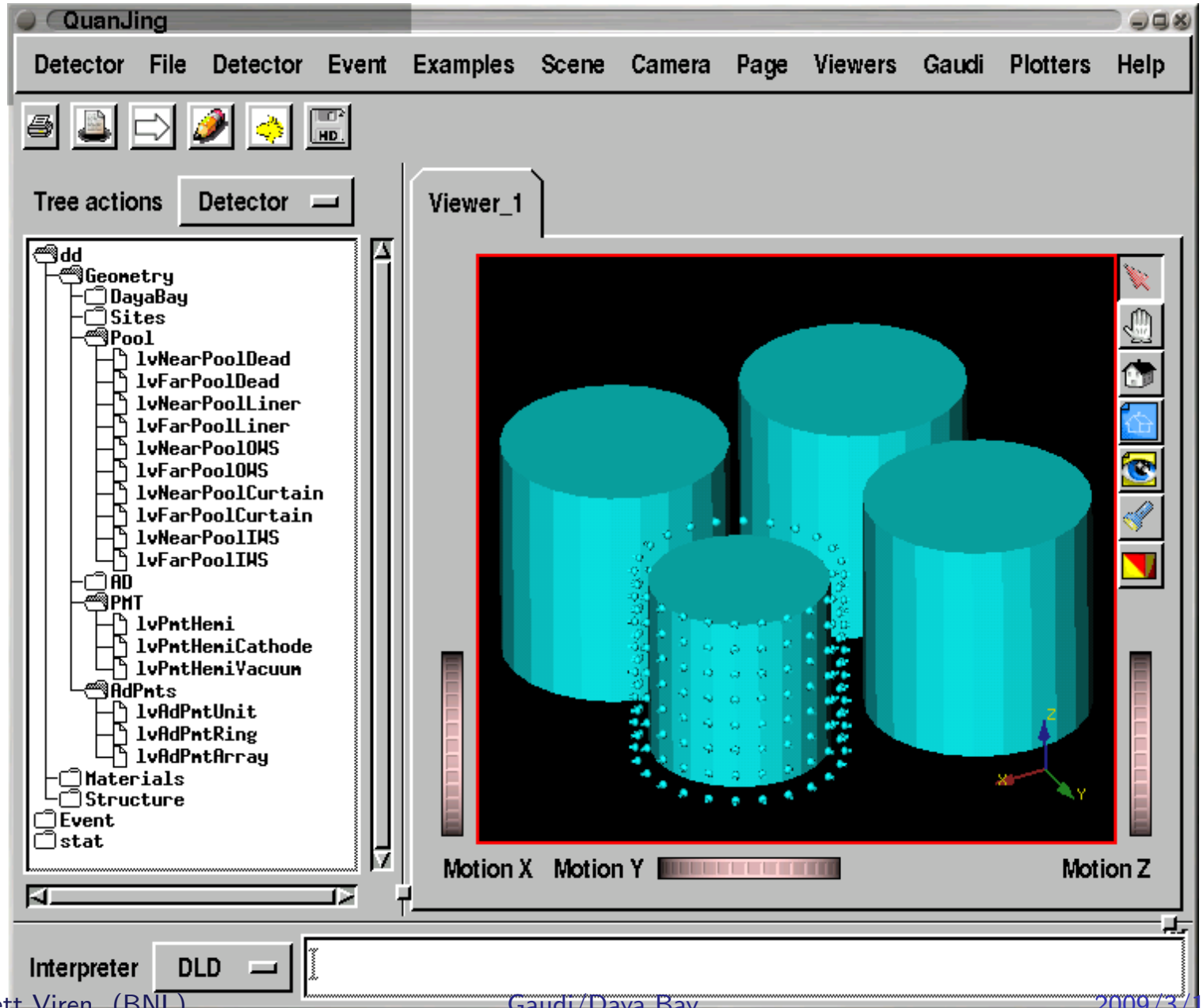
Result can be placed as one volume.

Building the AD PMT array with parameterized placement

```
<logvol name="lvAdPmtUnit">              <!-- step 1 -->
  <physvol name="pvAdPmtUnit" logvol="/dd/Geometry/PMT/lvPmtHemi">
    <posXYZ x="AdPmtRadialPos" z="-0.5*(AdPmtNrings-1)*AdPmtZsep"/>
    <rotXYZ rotY="-90*degree" />
  </physvol>
</logvol>

<logvol name="lvAdPmtRing">              <!-- step 2 -->
  <paramphysvol number="AdPmtNperRing">
    <physvol name="pvAdPmtInRing:1" logvol="/dd/Geometry/AdPmts/lvAdPmtUnit" />
    <posXYZ/>
    <rotXYZ rotZ="AdPmtAngularSep" />
  </paramphysvol>
</logvol>

<logvol name="lvAdPmtArrayZero">         <!-- step 3 -->
  <paramphysvol number="AdPmtNrings">
    <physvol name="pvAdPmtRingInCyl:1" logvol="/dd/Geometry/AdPmts/lvAdPmtRing"/>
    <posXYZ z="AdPmtZsep"/>
  </paramphysvol>
</logvol>

<logvol name="lvAdPmtArray">             <!-- step 4 -->
  <physvol name="pvAdPmtArray" logvol="/dd/Geometry/AdPmts/lvAdPmtArrayZero">
    <posXYZ/>
    <rotXYZ rotZ="0.5*AdPmtAngularSep"/>
  </physvol>
</logvol>
```

Fully parameter driven! Simulate Super-Kamiokande's or DUSEL 100 kTon detector's wall PMT arrays by changing just a few parameters.

The QuanJing (neé Panoramix) Display



Kinematics Generator - GenTools

GenTools:

- In house package.
- Generator built from one or more AlgTools.
- Each "GenTool" generates some part of kinematics:

Positioner	various ways to select vertex
Timerator	various ways to set vertex time
Gun	usual particle gun
Transformer	transform from volume-local to global coords
Specialty	particle beams, LED diffuser balls, etc.
External	HepEVT format from external program or file.

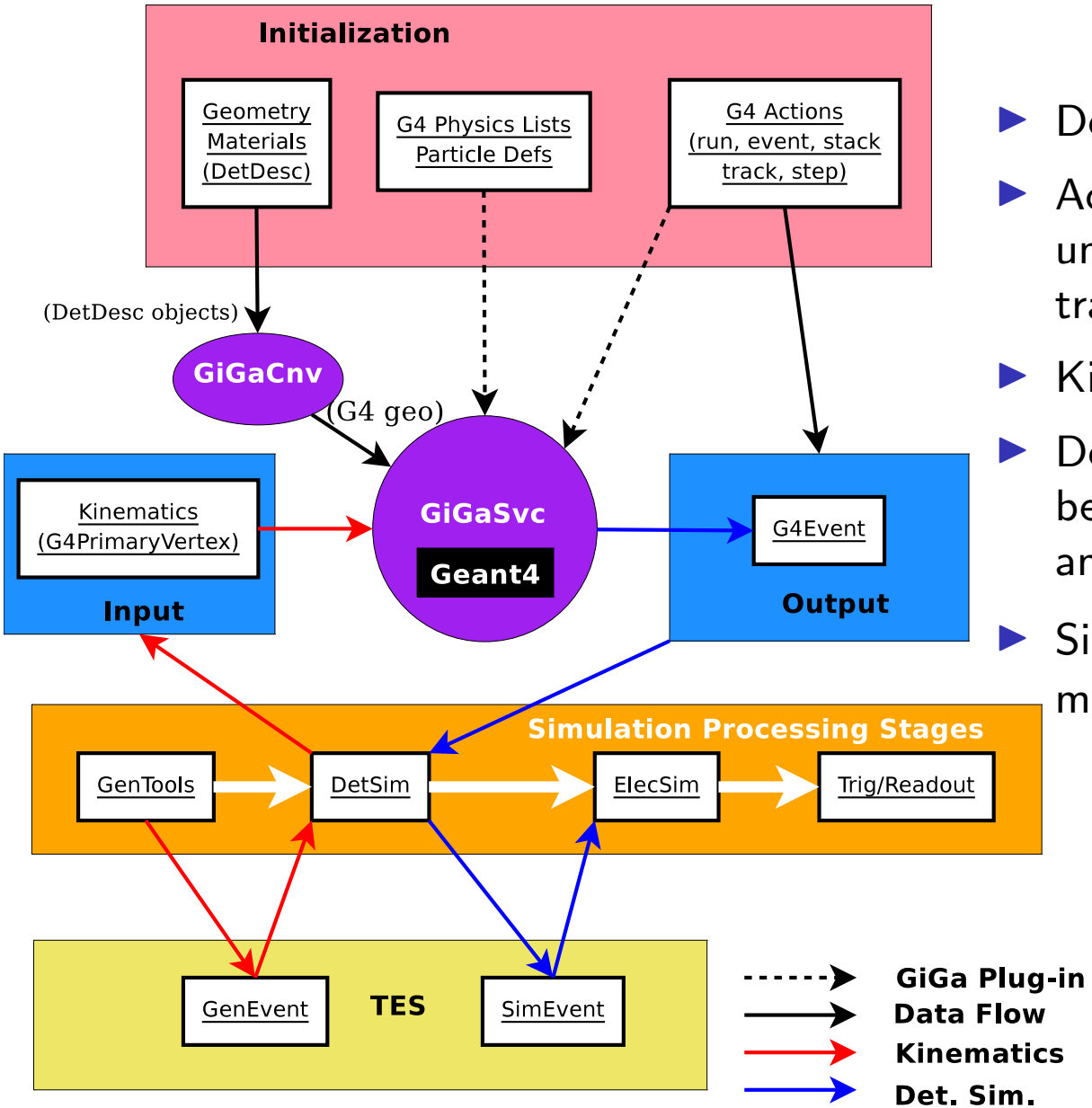
Detector Simulation - DetSim

DetSim:

- Application of GiGa = Geant4 Interface to Gaudi.
- Connection to DetDesc:
  - GiGa converts DetDesc to Geant4 geometry and material objects
  - Optical properties tunable in XML w/out touching C++
  - Sensitive detectors named in DetDesc matched to C++ class, $\rightarrow$ swap different PMT geometry and behavior in XML.
- Customize Geant4 via pluggable AlgTools:
  - Physics lists & actions (stack, step, track, etc)
  - Multiple actions of a given type possible.
  - Easier configuration than G4's "Messenger" mechanism.

$\rightarrow$ user-modification w/out user-code interference.

Interface to Geant4



- DetDesc → G4 geometry
- Action classes for unobservable statistics & trajectory recording
- Kine in, G4 data out
- DetSim algs interface between Kine & GiGa/G4 and Transient Event Store
- Simple linear processing model shown as example.
  - Alternative processor algorithm for "just in time" style model.

Daya Bay Specific Requirements

Daya Bay is not a collider experiment.

- Need to form delayed time coincidences over various scales.
  - $\rightarrow$Framework needs to support a look-back in time
- Have multiple, asynchronous and overlapping sources of events.
  - Radioactive decays, cosmogenic products, inverse-β decay signal events.
  - Overlaps due to detector extent, electronic delays/resolutions, or "lucky" vertex times.
  - $\rightarrow$Simulation must properly mix and overlap events from different sources of kinematic.

DUSEL / Long Baseline Neutrino Experiment shares these requirements.

File I/O - RootIO

- Rejected POOL due to large code base and compilation problems[5].
- In-house package: `RootIO` based on ideas from GLAST's `RootCnvSvc`

Features:

- 1-to-1 mapping from TES location to an abstract "stream" of data to a sink or from a source.
- Streams manage a TTree located in a TFile using the TES path.
- Streams can be spread among multiple parallel files and files can be split up sequentially.
- Default event selector: "next event" is next entry of each stream.
- Daya Bay event selector is more complex to handle case of zero or more objects placed in a TES location per one execution cycle.
- Simple usage from bare-ROOT session possible.

Goal: exactly reproduce state of TES on input that it was on output.

[5] These may be non-issues now.

Summary

- Gaudi provides a general framework that provides a useful basis for Daya Bay's offline software.
- Gaudi is not just for the big boys and can be adopted with modest effort.
- LHCb provides some additional, very useful and general packages.
- The combination of Gaudi+DetDesc+GiGa provides an extremely flexible code base particularly suited for detector R&D.
- Daya Bay has developed an automatic Gaudi + externals installation method and integrated with a Trac-based auto-build and test system.
- Daya Bay has successfully extended Gaudi to satisfy unique requirements.

Thanks go to Gaudi, LHCb and ATLAS developers and the rest of the community for much useful software and help.

Backup Slides

Archive Event Store (AES)

- When data added to TES also added to an ordered collection in AES.
- 1-to-1 mapping of TES and AES locations.
- Each AES location has an associated time window.
- As data falls out of window it is removed from AES (and offered for output to file).
- Algorithms can iterate over AES collection when searching for delayed time coincidences.

Mixing simulated events

Normal simulation processing model:

- Data is pushed through simulation stages: kine→detsim→elecsim→trigsim→readout
- Doesn't allow proper event mixing and overlapping

"Pull" or "just in time" processing model:

- Break up simulation chain into stages (kine, detsim, elecsim, etc)
- Stage manages one or more algorithms providing stage data (eg, one kinematics generator for each event type)
- Stage maintains a current "now" time
- Each request for data triggers processor to supply data until passing "now".
- Stage orders and buffers results, returns most current one.
- →Requesting top level (eg DAQ readout) data, pulls results from all stages.