Imperial College
London

Ganga and distributed analysis in LHC*b*

Ulrik Egede
On behalf of the LHC*b* collaboration and the Ganga Core development team

Sinaia, Romania, 13-18 Oct 2006

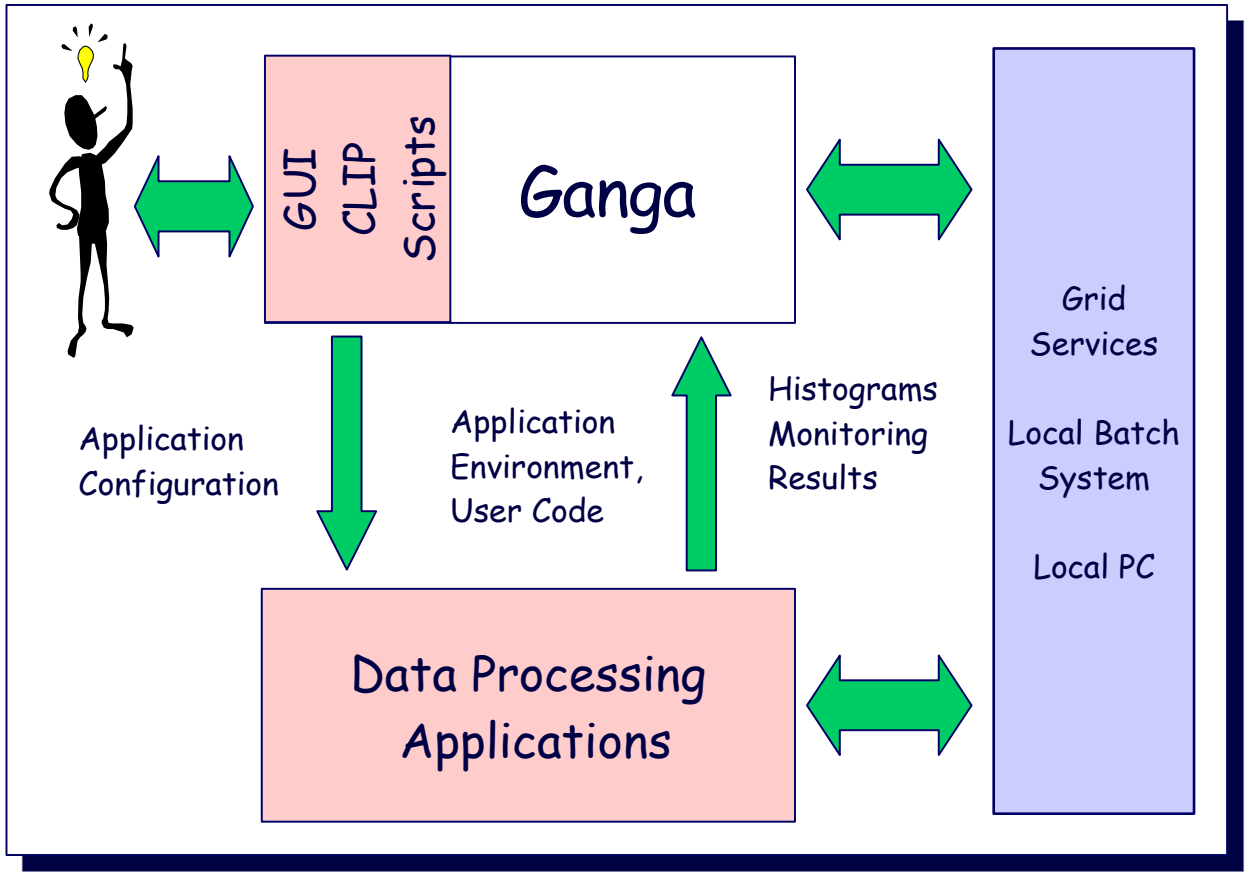
Overview

- The user analysis framework Ganga
  - As a general framework
  - As used within LHCb
  - From a developers point of view
- Where Ganga is currently used
- A demonstration of using Ganga

The Ganga framework

General Overview

- Ganga is a Grid user interface
  - It is a key piece of the distributed-analysis systems for ATLAS and LHCb
- Ganga takes care of
  - Configuring applications
  - Switching between local testing and large-scale processing on the Grid
  - Keeping track of results
  - Discover datasets by direct interfacing to file catalogues



- All written in portable Python code
  - ~20k lines of code
- A joint UK/CERN project

Command Line Interface

Ganga provides interactive access to objects either via the an enhanced Python shell (IPython)

Fully object oriented approach

```
HiggsFit(const char* SUSYModel, double HiggsMass) {  ...
TFile f("output.root","RECREATE");
 ...
}
```

An example of a ROOT script “HiggsFit.C”

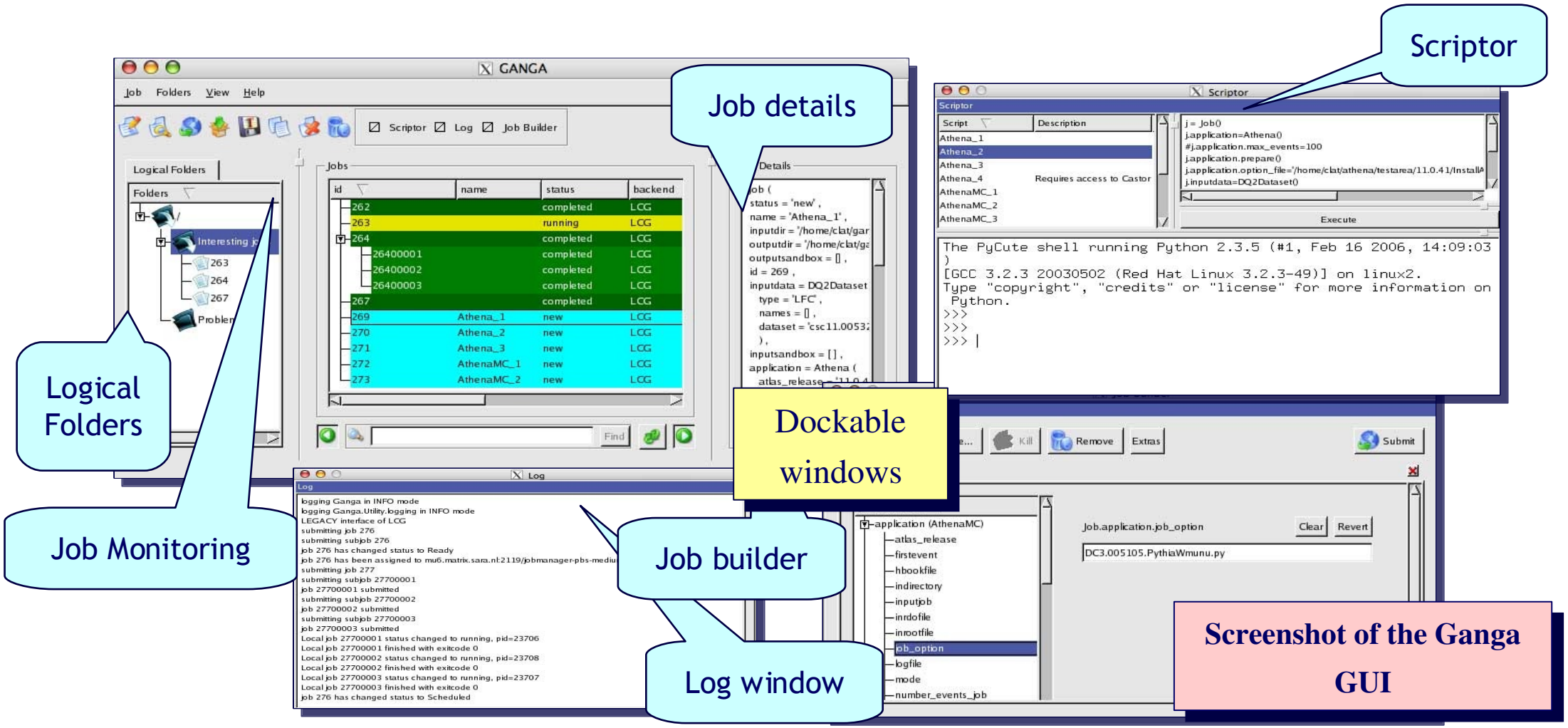
```
j = Job(application = Root(), backend = LCG(), splitter = ArgSplitter())
j.application.script = 'HiggsFit.C'
j.splitter = [['A‘, 110],['A‘, 130],['B‘, 270]]
j.outputsandbox = 'output.root'
j.submit()
```

An example of commands for job submission

```
help> index
Ganga Public Interface (GPI) Index
Classes:
   File        Represent the files, both local and remote and provide an interface to transparently get  access
to them.
   ..........
```

Interactive help system is available from the command line

Ganga GUI



The GUI interface is built on the top of the GPI using the PyQt graphics libraries
Most of the panels are build dynamically using the widget description from the class schema

Deployment

Tutorials for ATLAS and LHCb in various locations => Ganga tried out by close to a 100 people

CERN, September 2005

Cambridge, January 2006

Bologna, June 2006

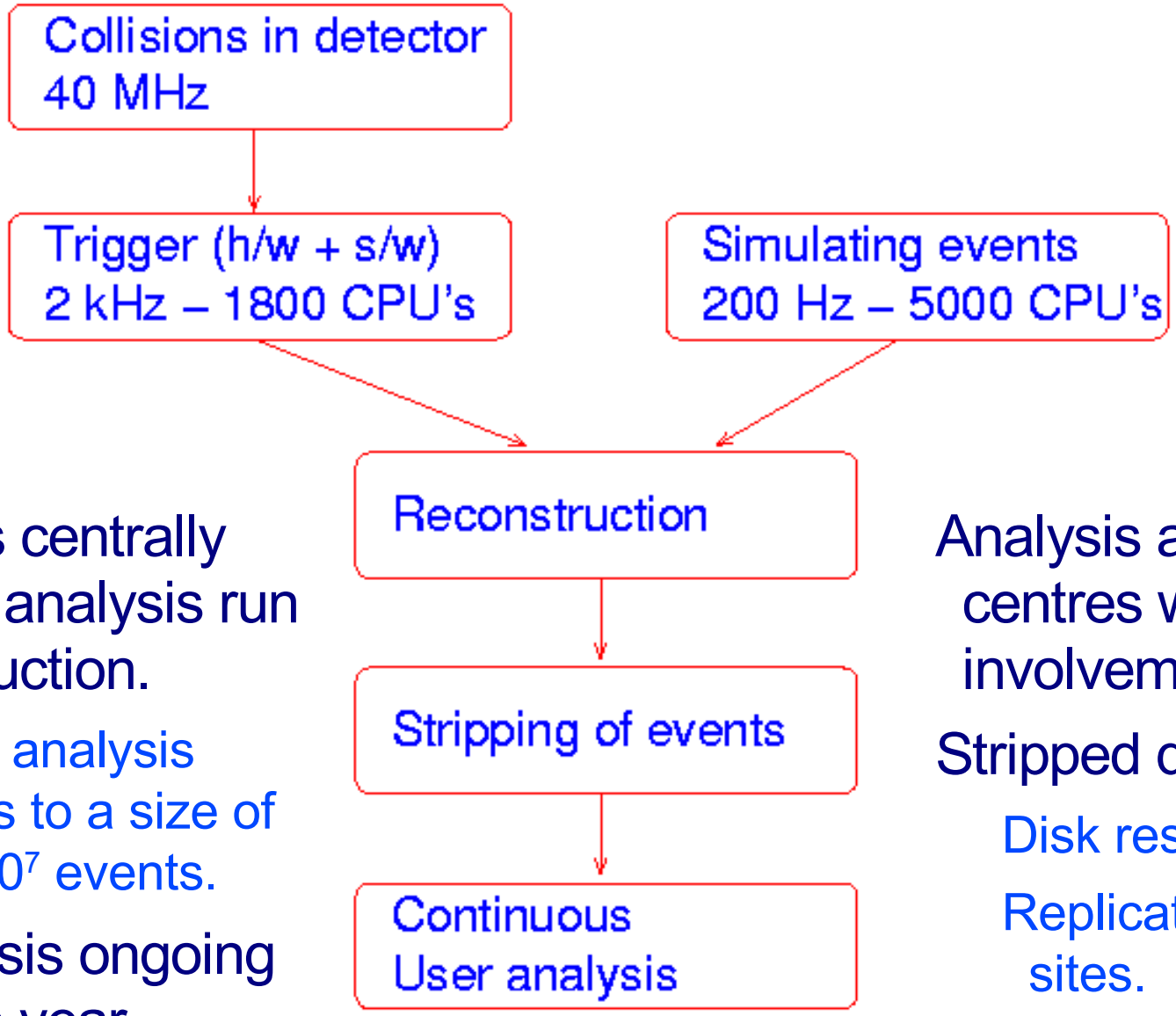
Distributed analysis recently integrated into the overall training for analysis in LHC*b*.

Regular set of tutorials within ATLAS

Support provide by developers through the CERN Savannah system.

Ganga as used in LHCb

LHC*b* analysis model (1/2)



Stripping is centrally managed analysis run as a production.

- Reduces analysis datasets to a size of 10^6 to 10^7 events.

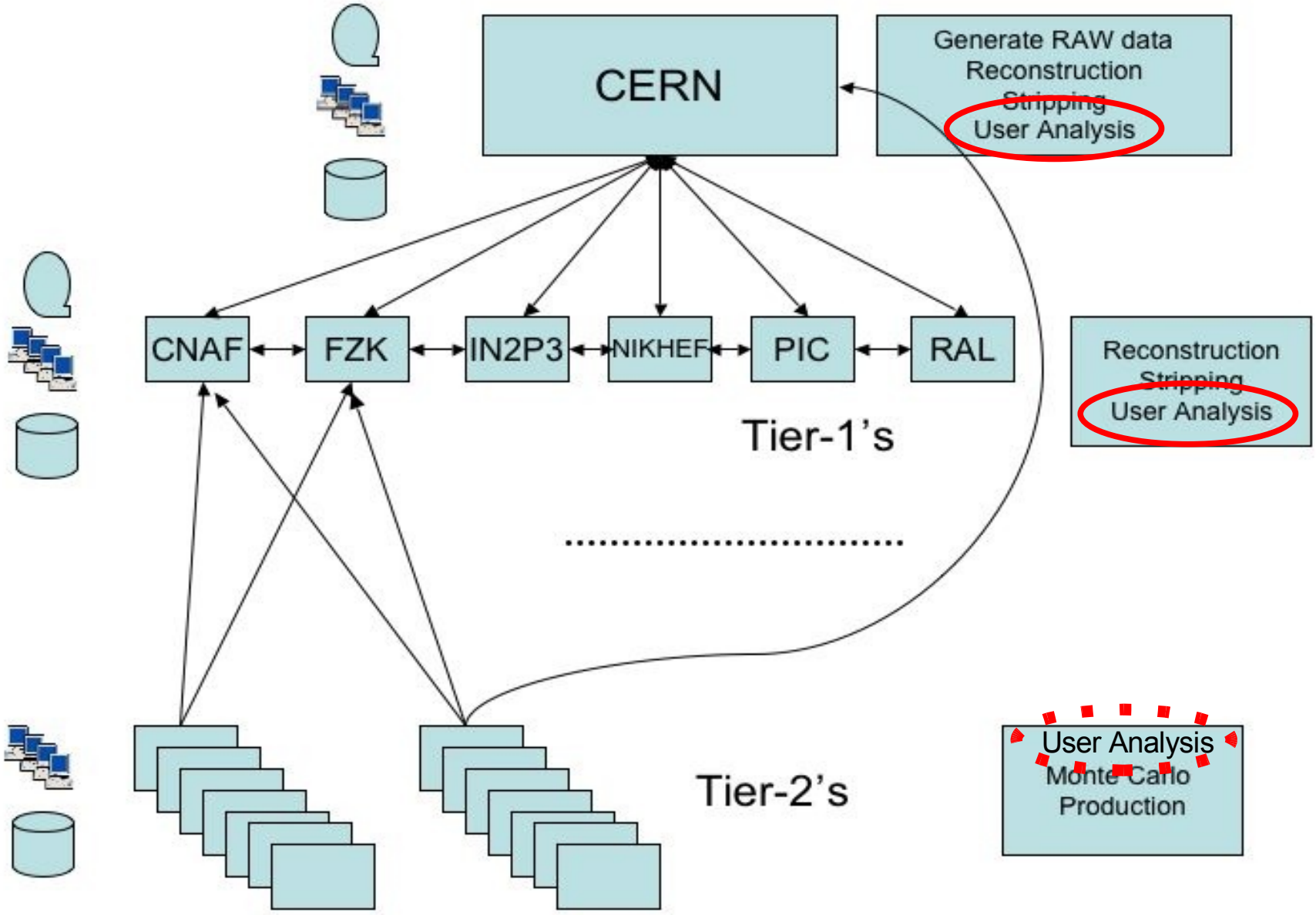
User analysis ongoing during the year.

Analysis at all Tier 1 centres with LHCb involvement.

Stripped data will be:

- Disk resident
- Replicated to all sites.

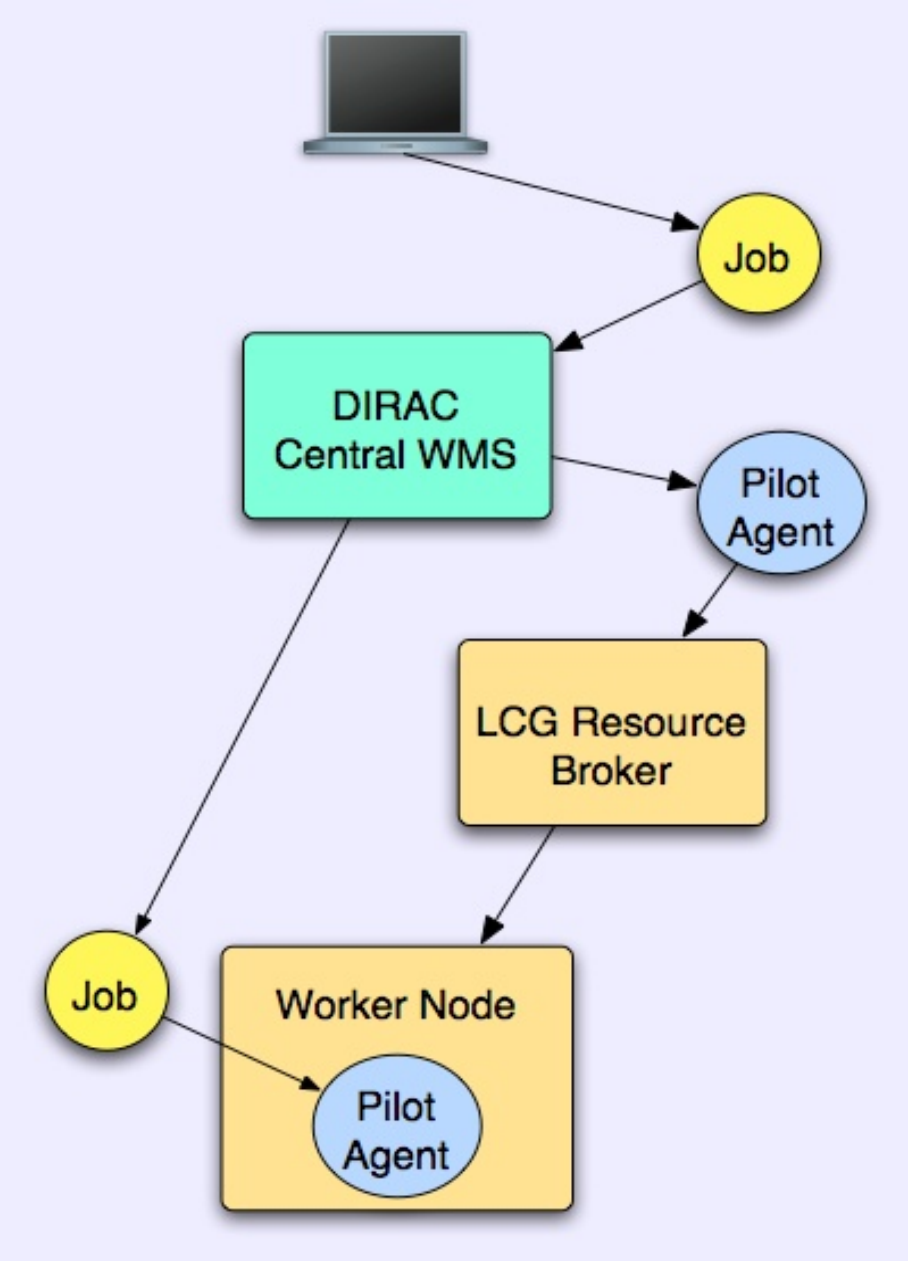
LHC*b* analysis model (2/2)



Analysis access to the Grid (1/2)

- No direct submission of jobs to LCG for LHC*b*
  - Analysis jobs are submitted to the Dirac workload management system (WMS) originally developed for LHC*b* Monte Carlo production.
- This gives us the advantage to:
  - Reduce the knowledge required of users.
  - Provides transparent access for reading and writing data on SE's
  - Allows LHC*b* to set priorities and/or restrictions for analysis jobs.
- See presentation by Andrei Tsaregorodtsev for many more details on the DIRAC system.

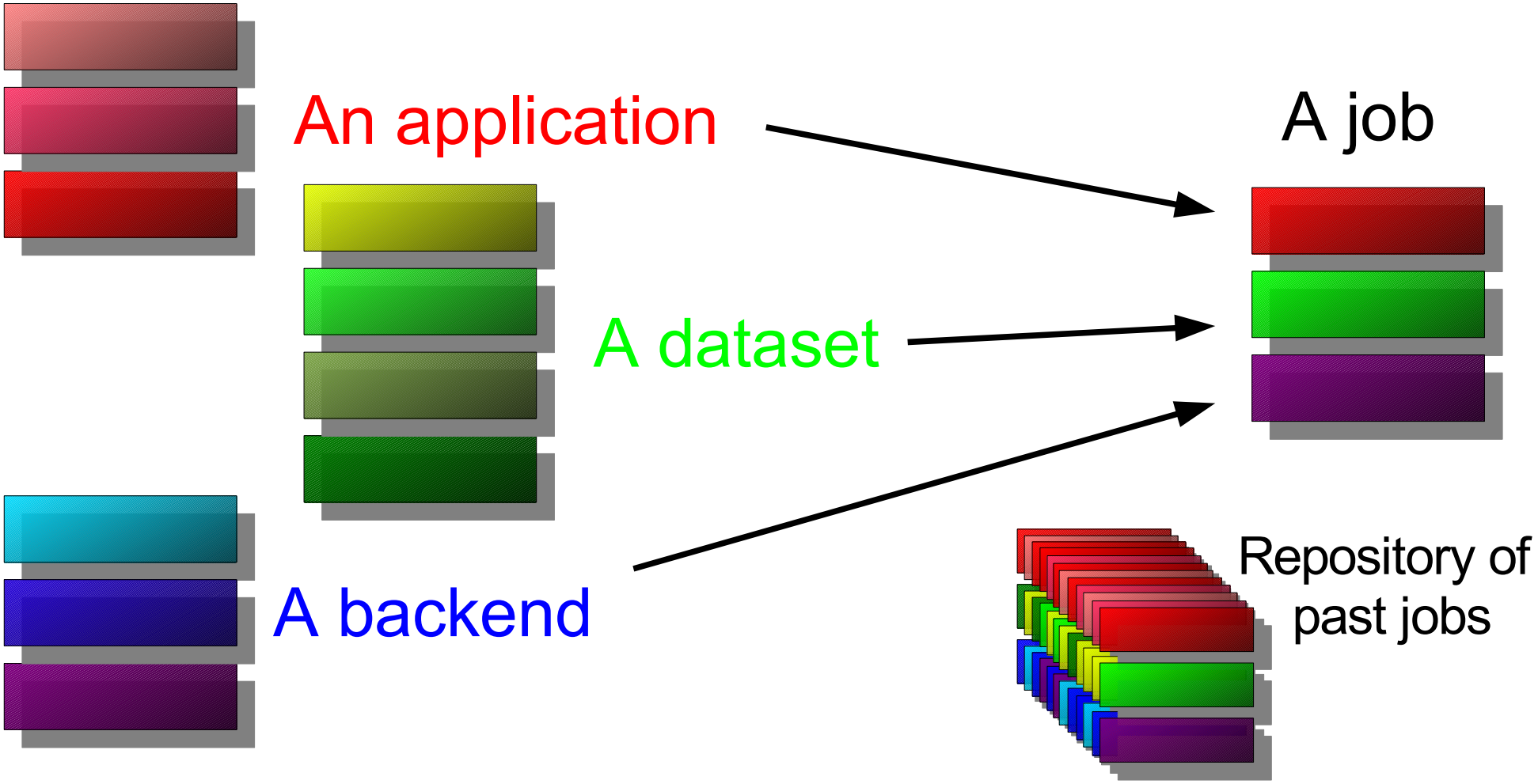
Analysis access to the Grid (2/2)



- User sends job to the DIRAC WMS
- WMS sends a pilot agent as an LCG job
- When pilot agent runs safely on a worker node it fetches job from WMS
  - Small data files returned to WMS
  - Large files registered in LFC file catalogue
- User query WMS for status and finally retrieve output from there.

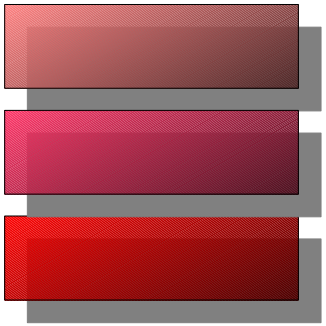
User view of analysis using Ganga

To define a job we combine different parts to create a job



Creating an analysis (1/3)

Predefined Python classes with specific knowledge about LHCb applications:



Gauss – for simulating events

DaVinci – for physics analysis

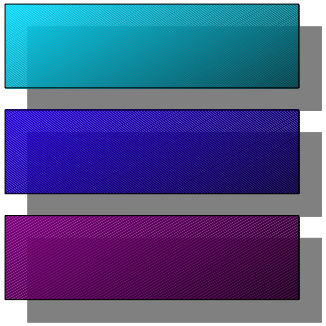
5 other LHCb applications

Objects know how to compile code, extract configuration, place user DLLs in input sandbox, specify files for output sandbox etc.

```
# Define an application object
app = DaVinci(version = 'v12r15',
        cmt_user_path = '~/public/cmt',
        optsfile = 'myopts.opts',
        extraopts = 'ApplicationMgr.EvtMax = 10;')
```

Creating an analysis (2/3)

A backend describes how the job will be executed



Local – run in the background on the client

LSF/PBS/SGE – submit to the batch system

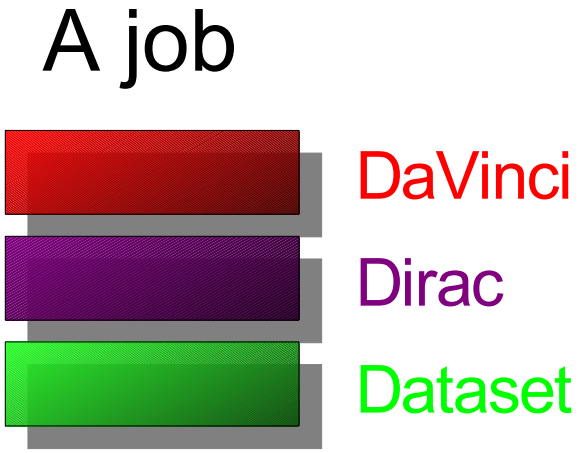
Dirac – submit to the Grid via Dirac

```
# Define a Dirac backend object
d = Dirac(CPUtime=3600)
print d
Out[34]: Dirac (
 status = None ,
 CPUtime = 3600 ,
 id = None
 )
```

Creating an analysis (3/3)

To put together and submit a job is simply by combining the different parts:

```
# Create an LHCb job and submit
j = Job(name='MyJob',
        application=app,
        backend=d,
        inputdata = ...)
print j
Out[38]: Job(
 status = 'new' ,
 name = 'MyJob' ,
 application = DaVinci (...)
 backend = Dirac (...)
 dataset = LHCbDataset(...) )
j.submit()
```



From a developers view

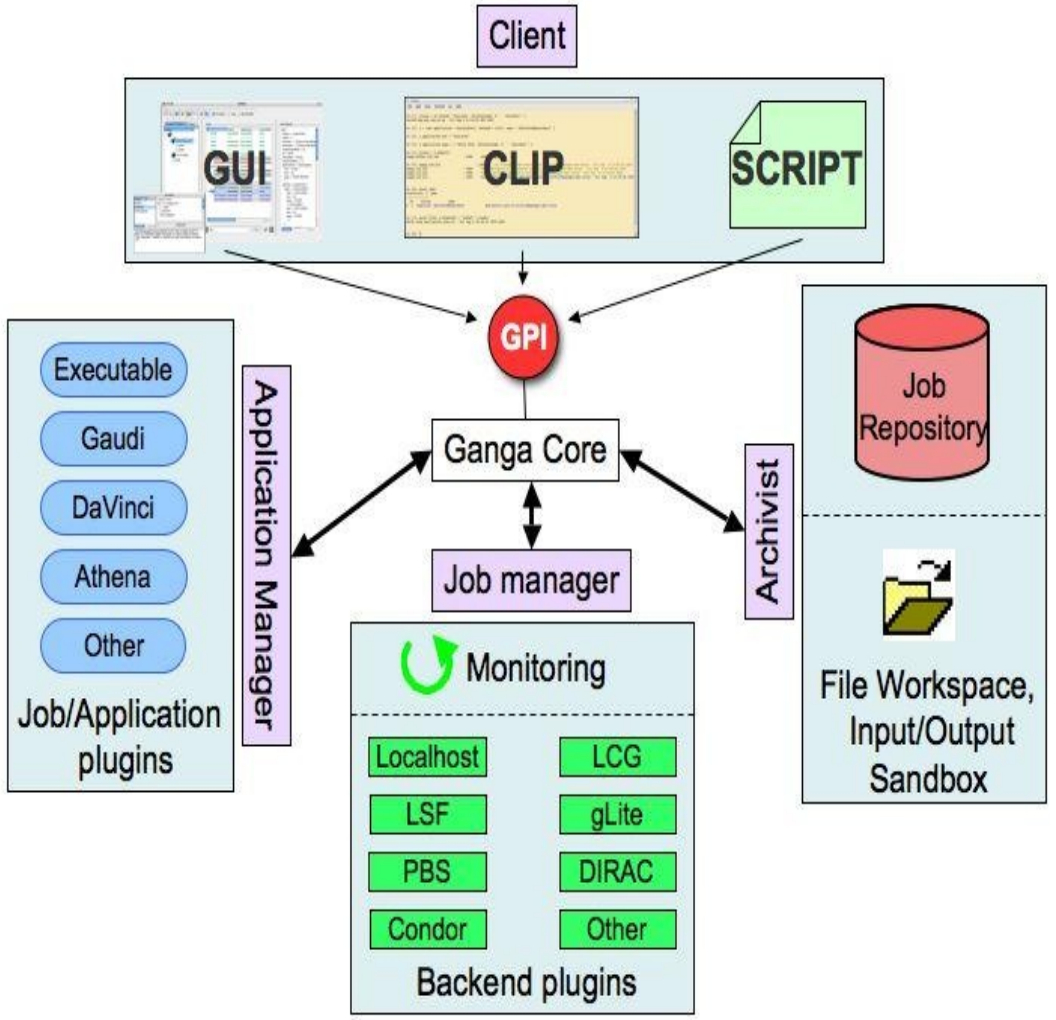
Architecture

Ganga Core performs most common tasks. It is represented by 4 main components:

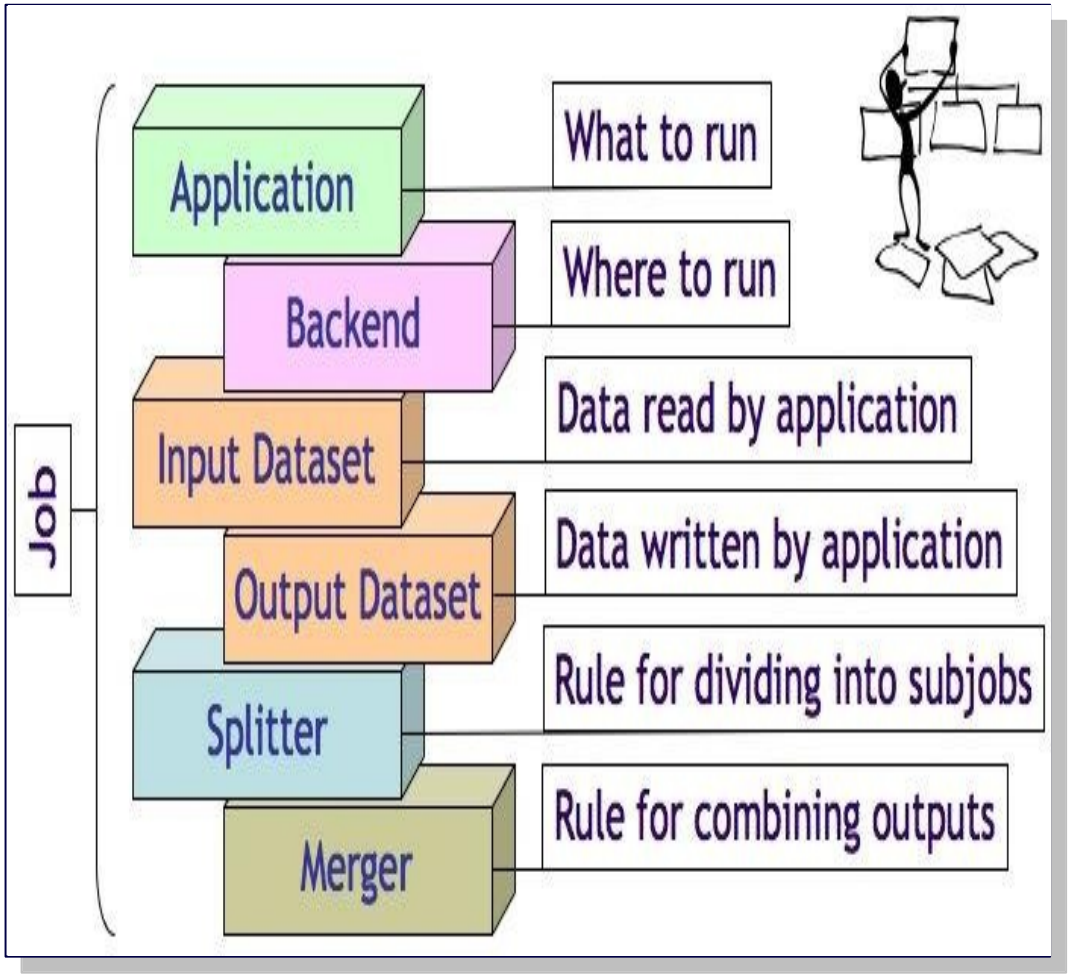
- Client
- Application Manager
- Job Manager
- Job Repository and File Workspace

Plugin components provide specific functionality

All components are linked together and communicate via the Ganga Public Interface (GPI)



Job Representation



The building blocks are implemented as plugin classes

Applications:

- Generic Executable application
- ROOT application
- ATLAS ATHENA application
- GAUDI-based applications of LHCb

Backend Plugins:

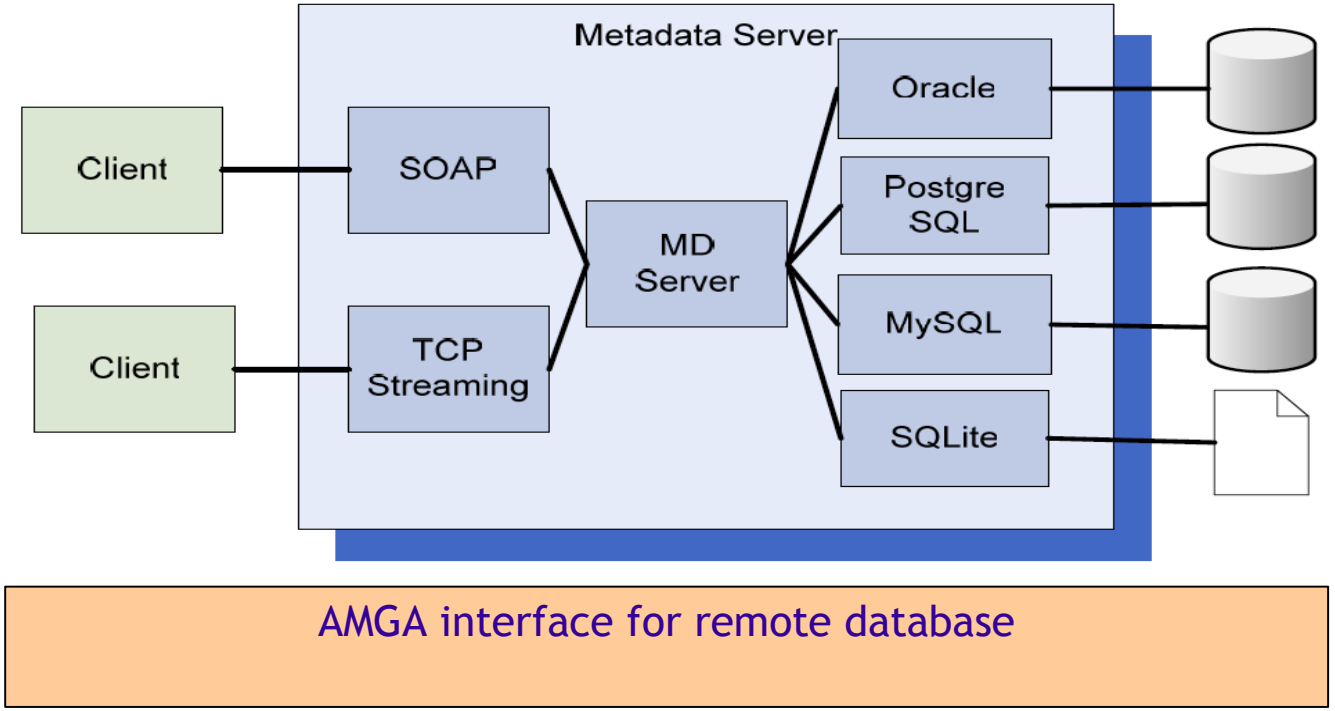
- LCG, gLITE
- DIRAC
- LSF, PBS, Condor, SGE
- Local PC

Each plugin has its own schema, which describes the configurable properties

Persistency

Job repository provides for storage and retrieval of job objects

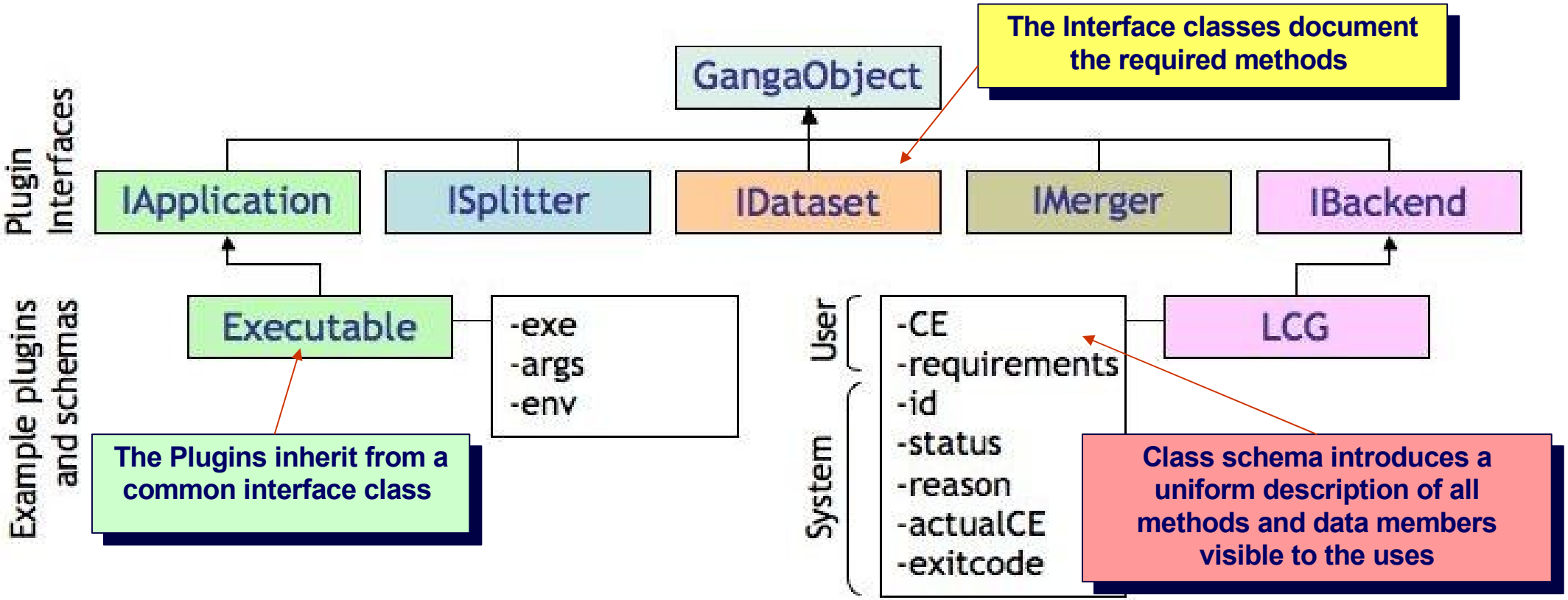
Either on local file system, or with repository on remote server using certificate-based authentication.



API for local and remote repositories are identical

- Support for selections, bulk operations, and fast retrieval of summary data
- Good scalability (has tested up to10 thousand jobs per user)
- Average time of job creation being 0.2 and 0.4 second for the local and remote repository respectively

Plugins



The plugin modules represent different types of applications and backends.

Such a modular design allows the functionality to be extended in an easy way to suit particular needs of the experiments

Extensions

```
class Executable(IApplication):
        """ Executable application -- running arbitrary programs. """
```
1

```
_schema = Schema(Version(2,0), {
      'exe' : SimpleItem(defvalue= '',
                         doc='File object with executable.'),
      'args' : SimpleItem(defvalue=[],sequence=1,
                          doc="List of arguments." )} )
```
2

```
# preferences for the GUI...
_GUIPrefs = [{ 'attribute' : 'exe', 'widget' : 'File' },
             { 'attribute' : 'args', 'widget' : 'String_List' }]
```
3

```
def configure(self, masterappconfig):
      # do the validation of input attributes
```
4

```
[Executable_Properties]
exe = echo
args = ["Hello World"]
```
5

```
[Configuration]
RUNTIME_PATH = /my/SpecialExtensions:GangaAtlas
```
6

GUI and CLIP representations of the plugin classes can be built automatically

Can add a new plugin without special knowledge of the GUI and CLIP frameworks

To build an extension:

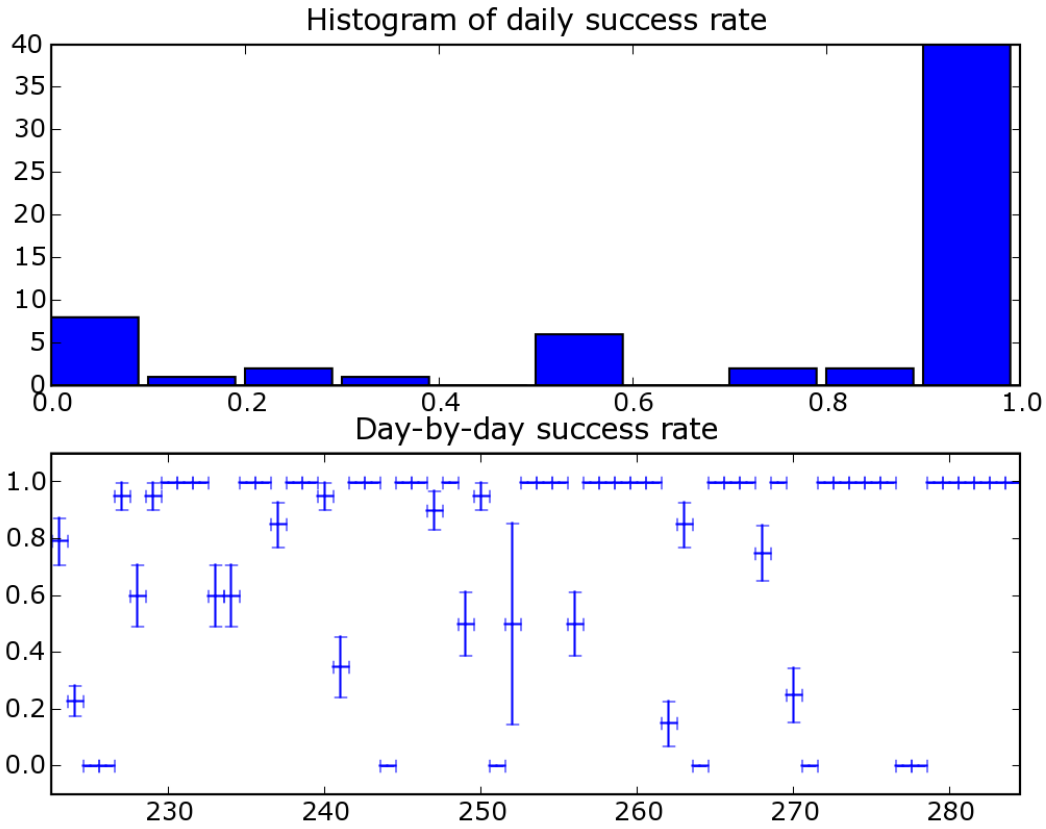
1. Derive a class from corresponding plugin interface
2. Describe the class schema
3. Specify GUI appearance (if any)
4. Implement plugin methods
5. Create a configuration unit to set up default values (if required)
6. Specify path to the extension module in the configuration file

No need to change anything in the released code!

Current uses of Ganga

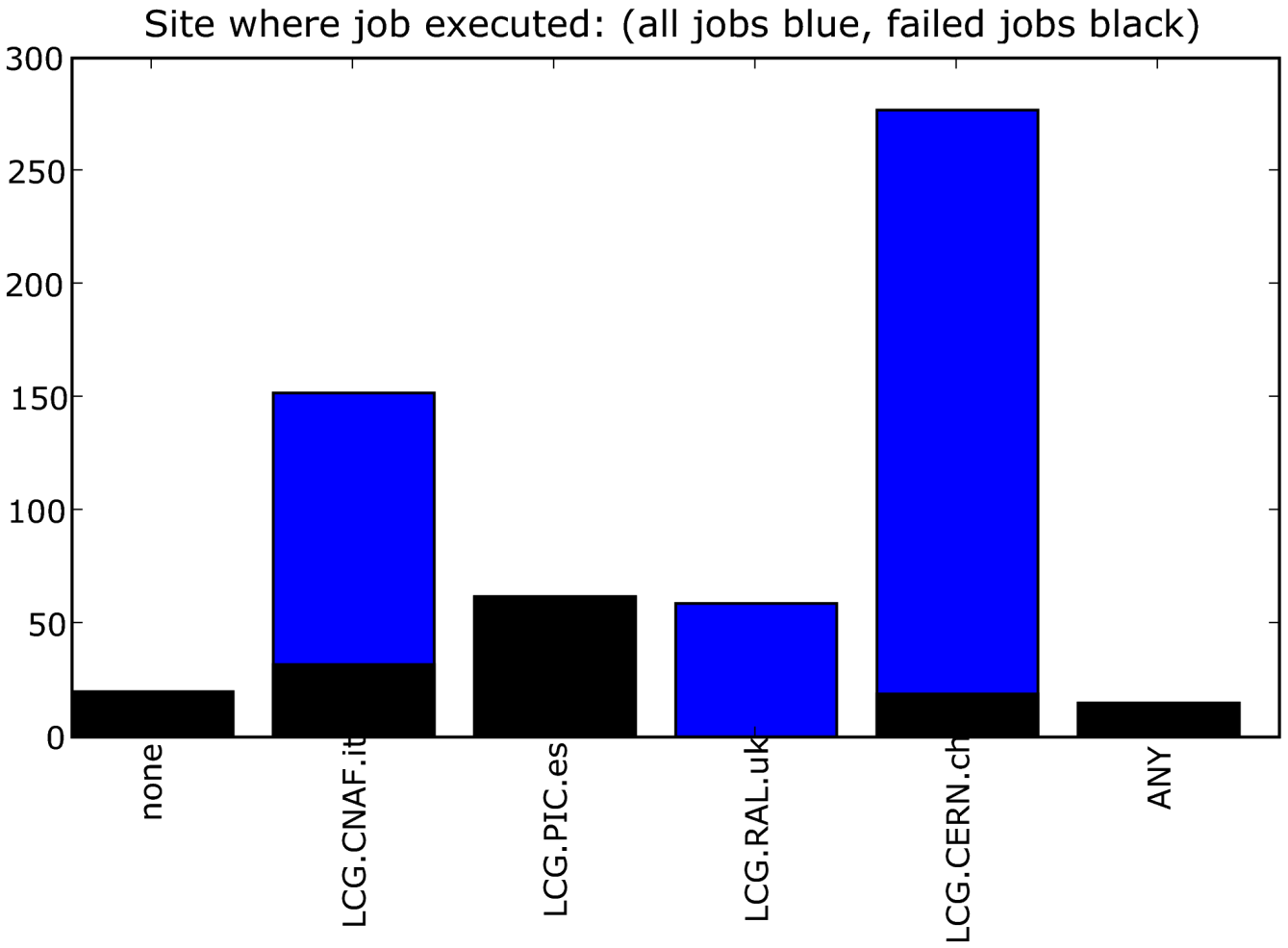
LHCb distributed analysis performance

- Monitor daily performance of LHCb analysis jobs sent to the DIRAC WMS.
- Evaluates performance of ful analysis chain right to the correct output.
- Automatic notifications.
  - Will get triggered when performance goes below a certain level.



LHCb distributed analysis performance

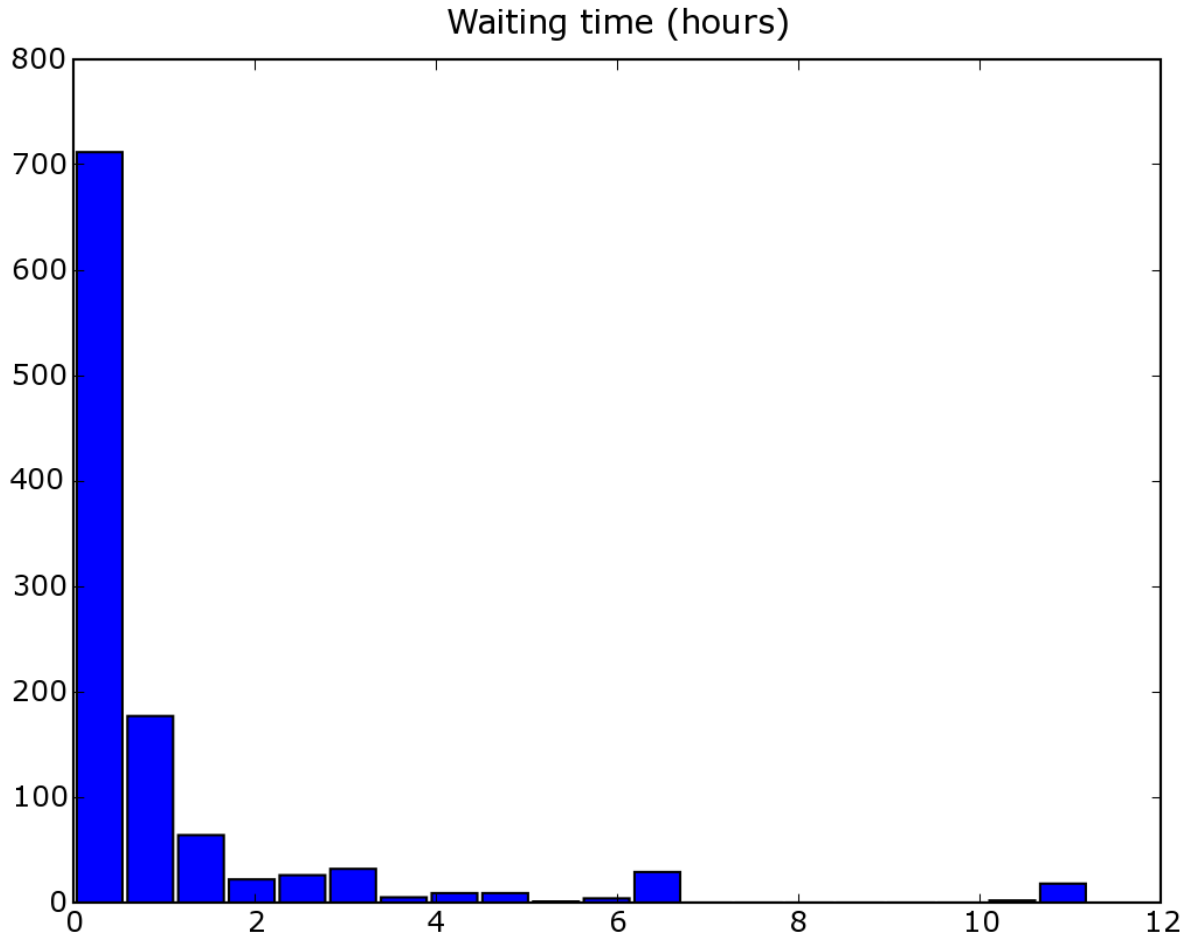
Also monitor where the analysis jobs run and their relative success



LHCb distributed analysis performance

Look at waiting time from job submission until it starts consuming CPU

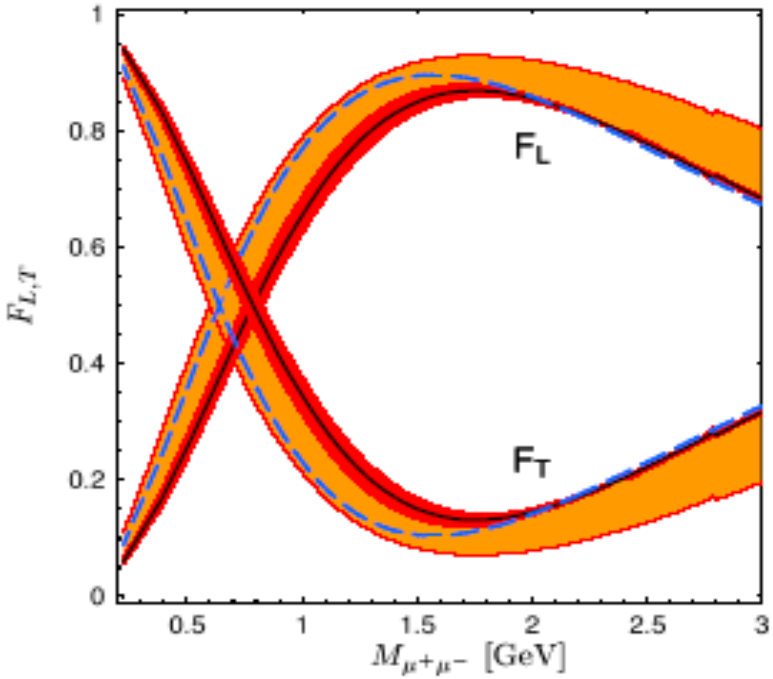
All jobs runs in competition with very high rate of production jobs

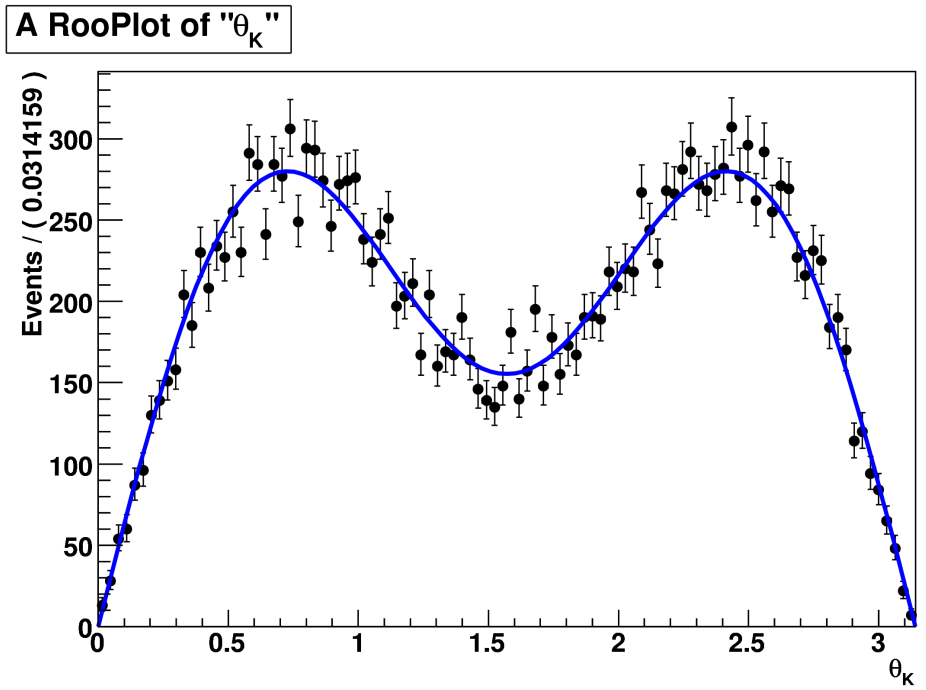


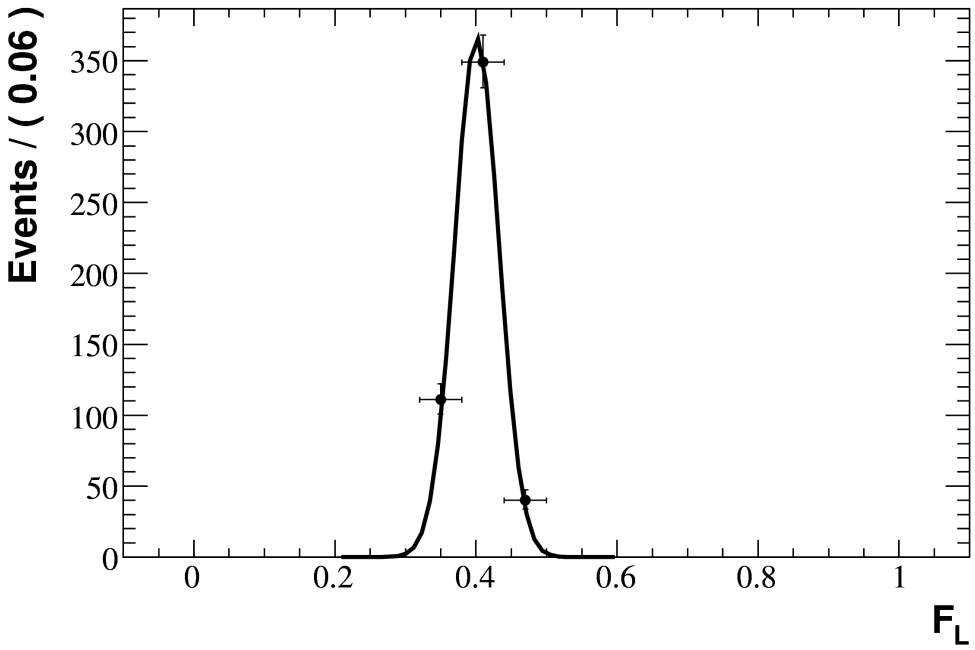
Toy Monte Carlo studies

Run large number of toy MC studies for resolution and correlation studies in $B_d \rightarrow K^{*0}\mu^+\mu^-$ decay.

- Using RooFit package
- Ganga jobs submitted to LCG







Use within EGEE

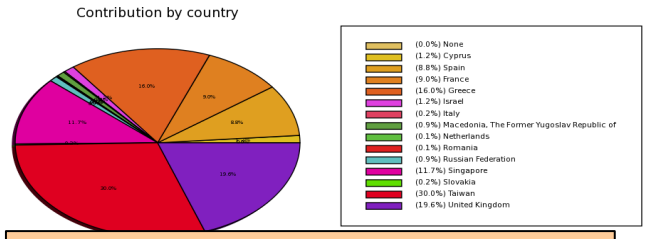
In EGEE, Ganga is used as submission engine and monitoring system for the DIANE job-distribution framework

BBC NEWS

Grid searches for avian flu cure

A cure for bird flu is being sought by computers that usually search for the fundamental elements of matter.

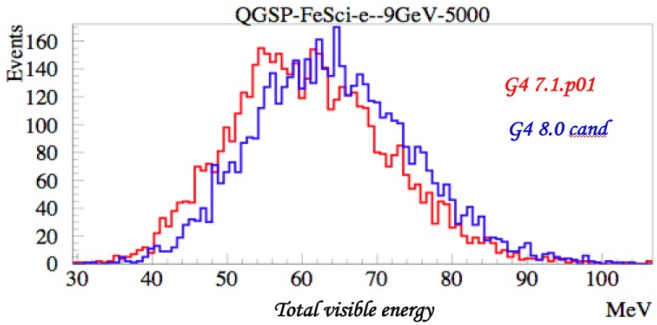
- Use of Grid in search for drugs against avian flu widely reported
- About one eighth of jobs submitted using Ganga/DIANE



Job statistics from Ganga

Geant 4

- Geant 4 regression tests performed for major releases (twice per year)
  ⇒ Search for differences in simulation results
- Ganga/DIANE adopted for running these tests on the Grid
  ⇒ First use December 2005



International Telecommunication Union

RRC 2006

- ITU Regional Radio Conference held in Geneva, May-June 2006
- Required real-time optimisation of evolving plan for sharing frequencies between 120 countries
  ⇒ Maximise number of satisfied requests
  ⇒ Minimise interference
- Ganga/DIANE used to run optimisation jobs on the Grid

Demonstration

Conclusion

The Ganga framework allows to submit jobs to the Grid in an easy and transparent way.

The Ganga framework makes it trivial to perform testing on local system and then transfer to the Grid for full scale analysis.

Steady build-up of user base within both LHCb and ATLAS for distributed analysis.

General implementation of Ganga allows the use in may other areas without modifications to the Core.

Documentation of Ganga system is available at http://cern.ch/ganga.