



# Harnessing Grid Resources with Ganga and Dirac



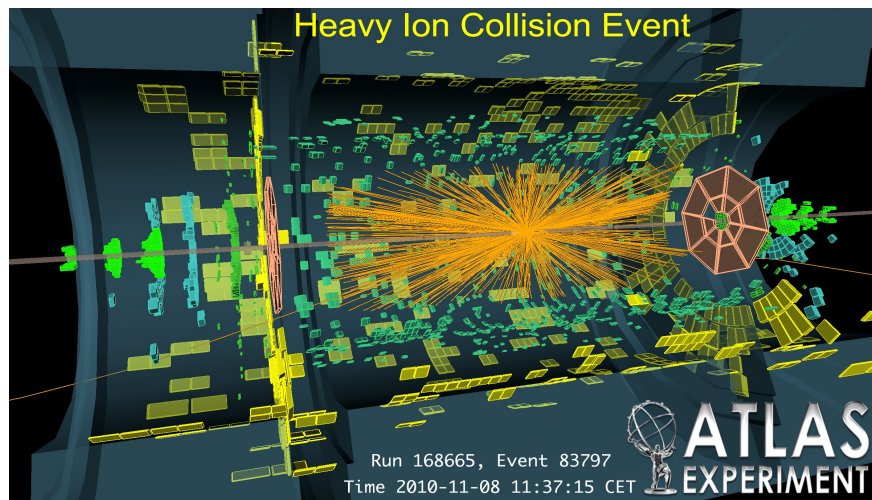
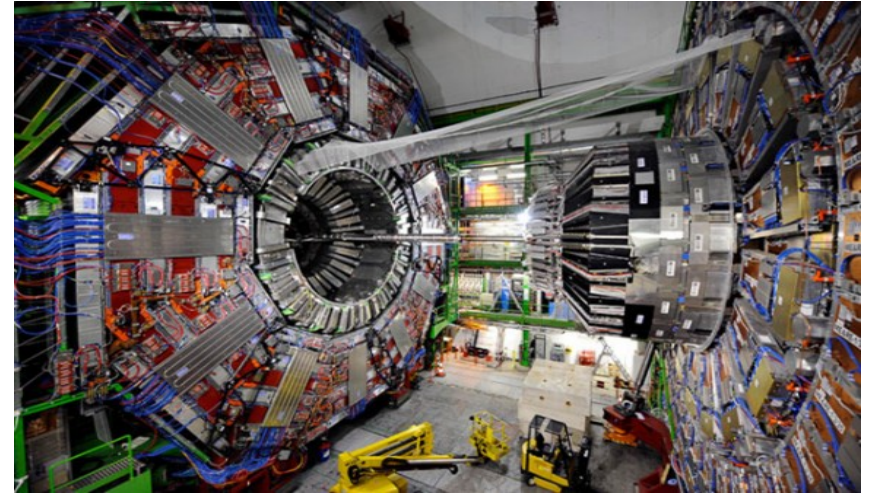
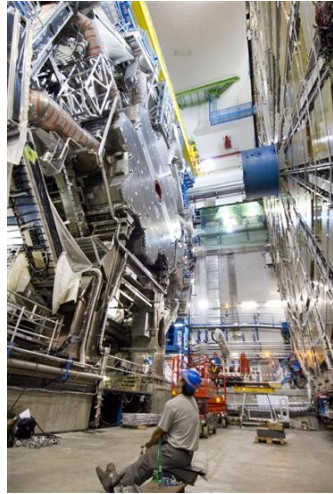
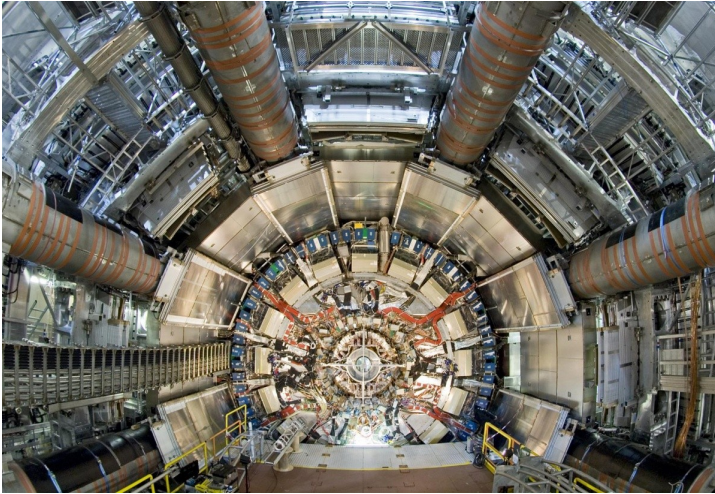
***Durham Seminar, 15th October, 2014***  
*Mark Slater, Birmingham University*

# An Introduction to the Grid



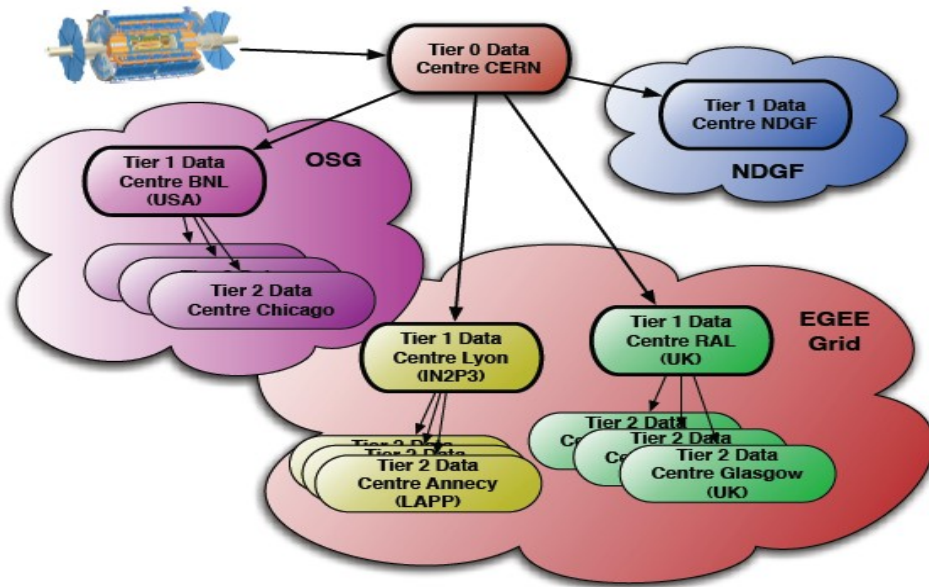
# Why we need the Grid

As you all probably know, the LHC detectors are very large and complex:



The many sub-detectors and systems not only produce vast amounts of data (**30 Petabytes annually!**) but also require a lot of computational analysis to measure the small signals we're looking for

To store, process and provide access to this amount of data was not practical (or desirable) from one location, so the Worldwide LCG Computing Grid was developed consisting of ~170 computing centres around the world



- ~250000 CPU cores are available ●
- ~350PB of disk storage ●
- ~130PB of Tape storage ●

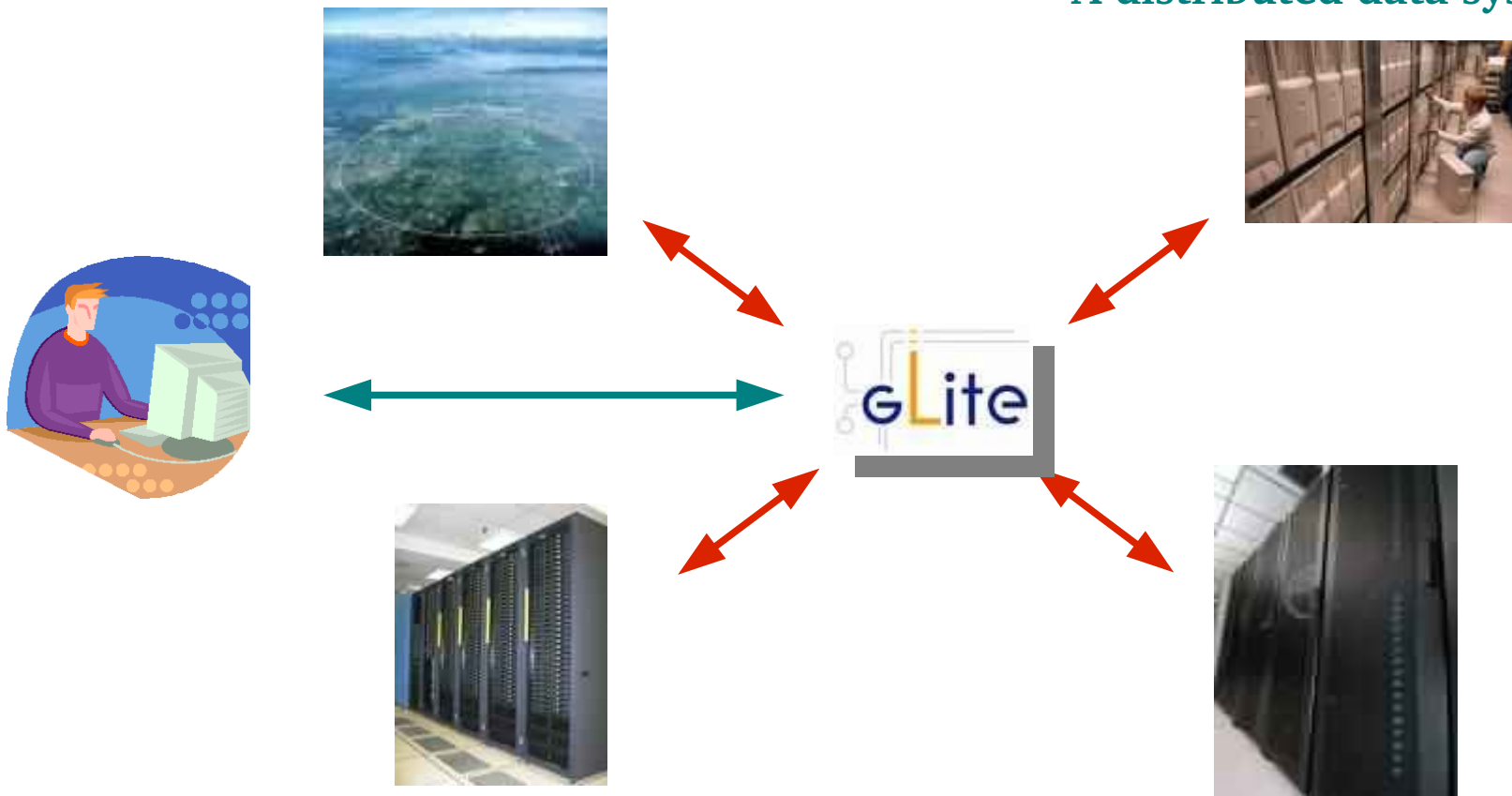
All this is arranged in a 'Tier' system so the data flows from the detectors at CERN, through 'countrywide' Tier 1's and finally out to smaller Tier 2's

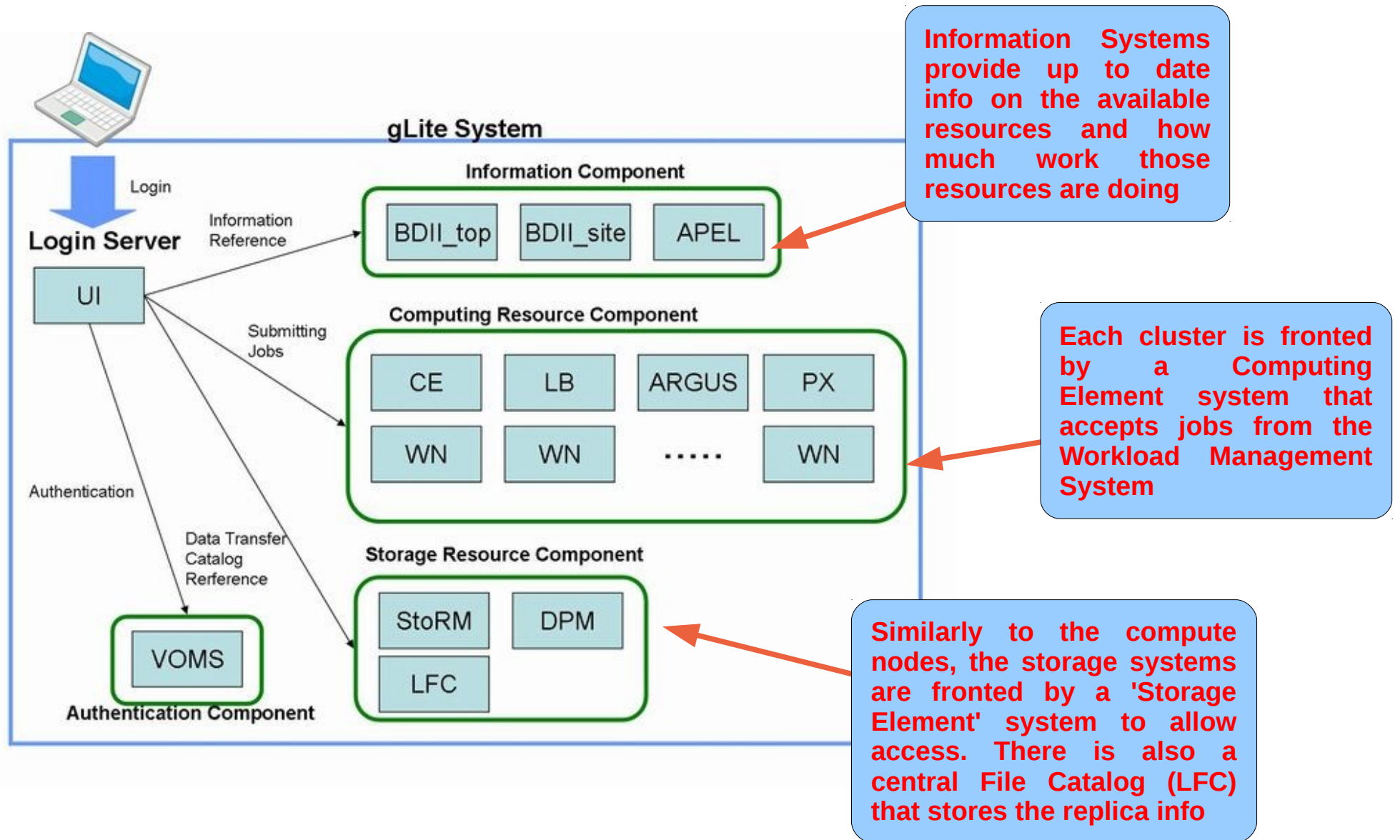
Here is a 'live' snapshot from a few weeks ago of grid activity. In a 10min window, there were ~200000 running jobs and a transfer rate of 7 GiB/s. In 24 hours, there are ~1 million jobs run.



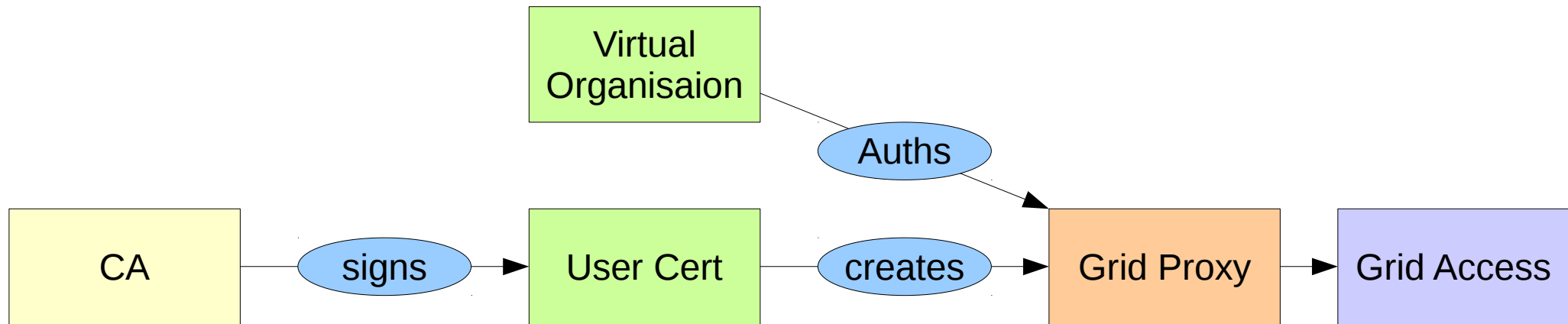
The 'glue' that sticks all of these disparate computing clusters and resources together is the Grid Middleware. This allows:

- Jobs to be submitted from anywhere ●
- An up-to-date reflection of a site's status ●
- A Workload Management system to ease job submission ●
- A distributed data system/catalog ●





The security of the grid was a top priority. We could not let just anyone have access to such a massive resource, not only because of the chances of interfering with other peoples work but also due to intentionally (or not!) causing DDOS attacks or other problems



The security chain starts with a user requesting a Grid Certificate from a Certifying Authority. They then have to request membership of at least one 'Virtual Organisation' that will be somewhat 'responsible' for their actions on the grid

The user is then able to create a Grid Proxy file (which will have a limited lifetime) that is actually used to access resources

If any problems are found with a particular user, their proxy and certificate can be blocked within a few hours

As time went on, the LHC experiments developed their own solutions to go 'on top' of the existing infrastructure to improve efficiency. In particular, Atlas and LHCb developed the following:

## Pilot Systems

To avoid the high failure rate of jobs, small 'Pilot' jobs were sent instead that would request work from a central queue only after they were established on a machine.

## Experimentally specific File Catalogs and Bookkeeping

Organising data is very specific to an experiment's need and so different implementations of a File Catalog are required along with particular ways of accessing the information



The difficulty was making these improvements available to the wider community of grid users – **GridPP** are now able to offer a DIRAC service with support through Ganga

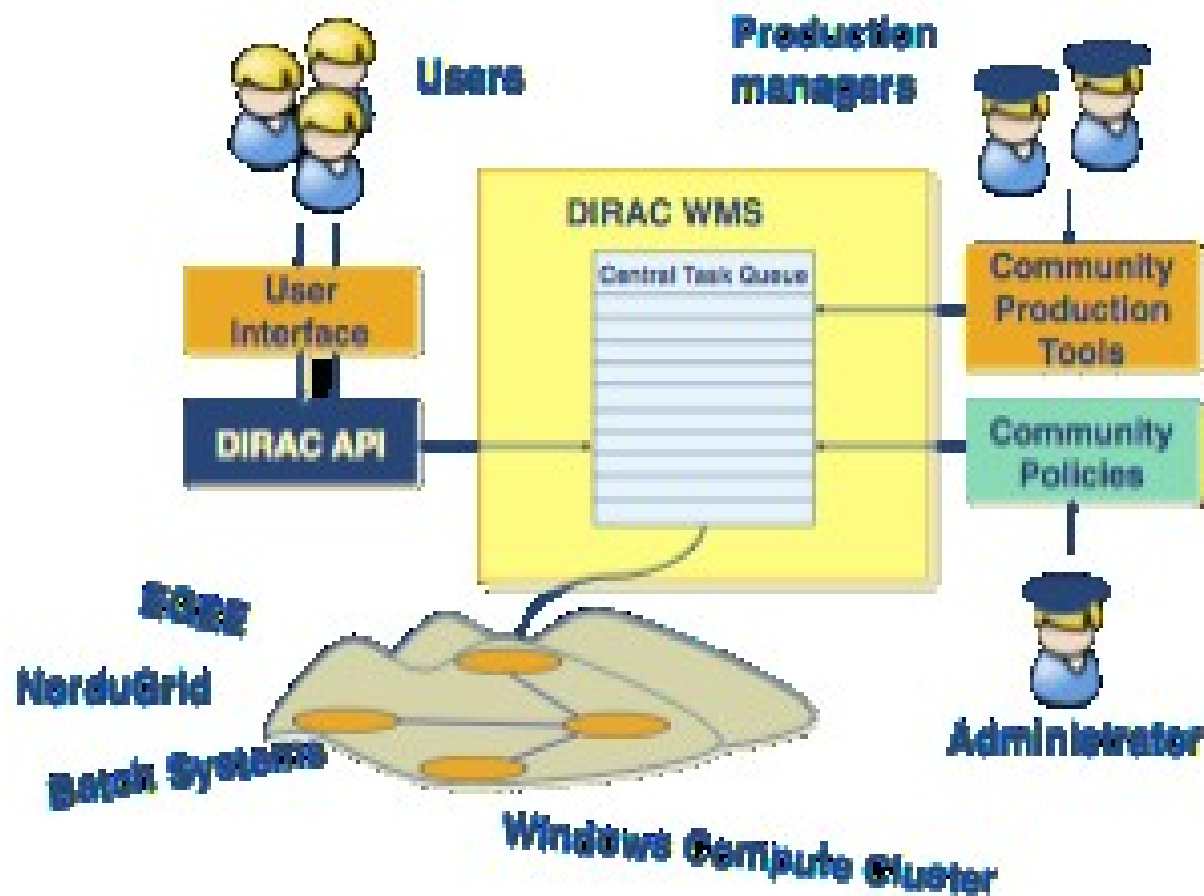




# An Introduction to DIRAC



DIRAC is Pilot based production management system that is designed to abstract the user from the various implementations of different sets of computing resources and other non-homogenous parts of the Grid



The DIRAC (Distributed Infrastructure with Remote Agent Control) system is build on the idea of interconnecting:

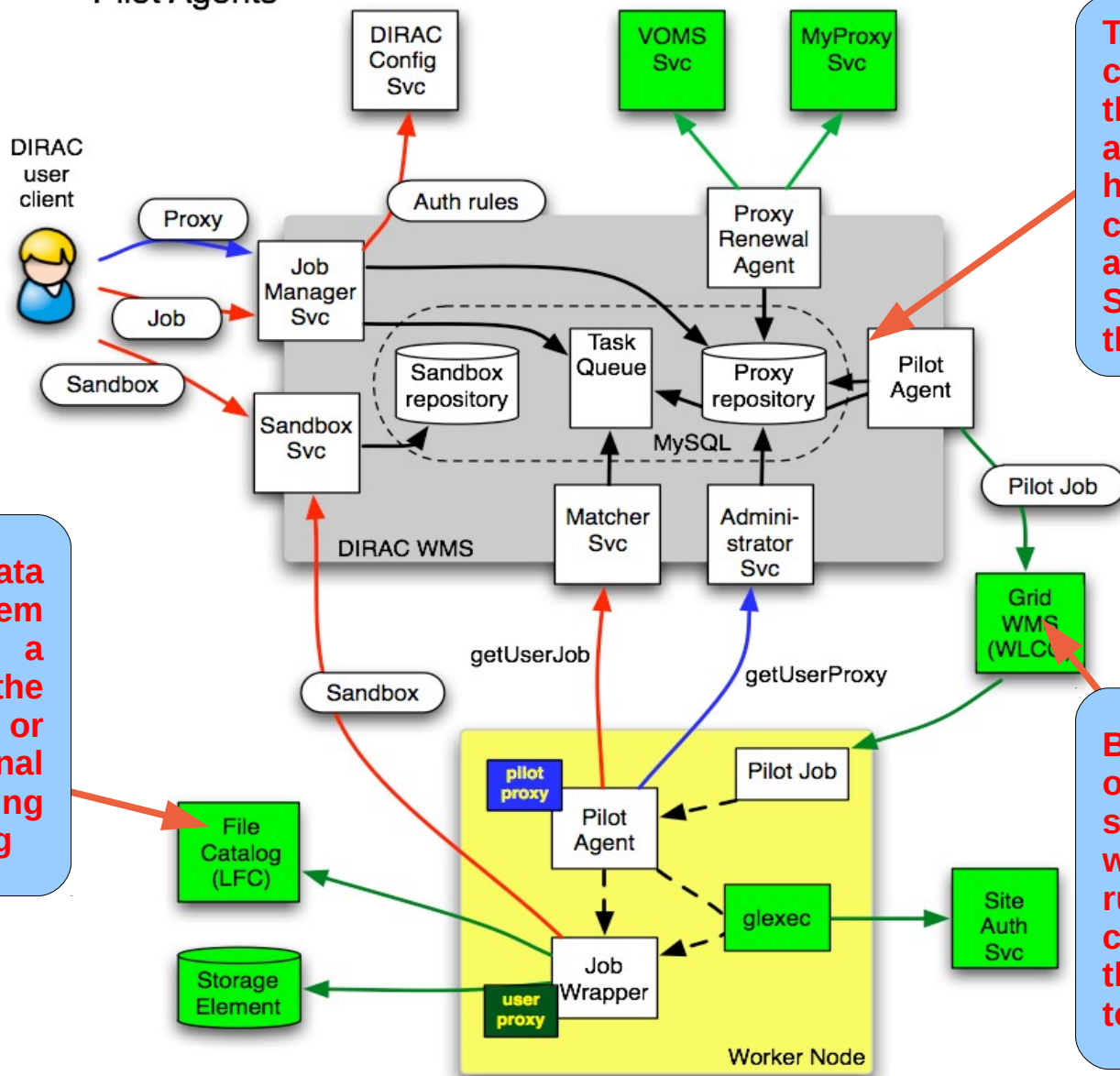
- Databases:** Used to persist the state of a system ●
- Services:** Passive, listen for reqs and serve/update the database ●
- Agents:** Active, Periodically performing tasks on the databases ●

These then combine to form *Systems*.

*For example: DIRAC's central Task/Job Queue is the Database, a service shows the status of the task queue and updates it when requested and the Pilot agent queries what tasks are required and sends pilots to perform them at sites*

*It was designed to replace and improve on the Glite WMS and LFC Data Management systems and offers many advantages to both*

DIRAC WMS with generic Pilot Agents



The DIRAC WMS contains the 'meat' of the job management and production handling and comprises Request and Transform Systems as well as the main Task Queue

The DIRAC Data Management system can be used as a frontend for the existing LFC or provide additional functionality using it's own File Catalog

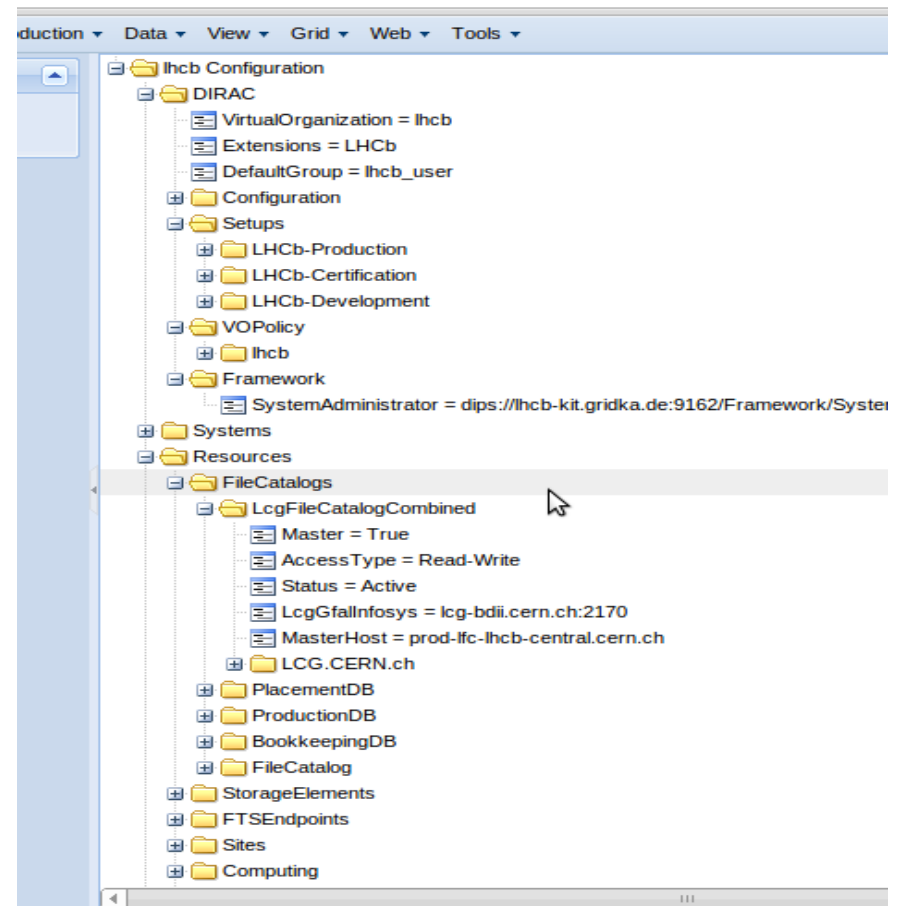
Because DIRAC pilots operate as a 'pull' system - asking for work when they start running - DIRAC jobs can run on anything that the pilots can get to, e.g. Clouds

The configuration system is a (nominally!) static description of resources, variables, etc. that is used by other systems

It is used as a global 'configuration file' with a service controlling the access to the data

```

/Resources/Sites
    /CERN.ch
    ...
    /IN2P3.fr
        /Domains = EGI, LCG
        /ContactEmail =
someone@somewhere
        /MoreDetails =
blah, blah, blah
        /Computing
            /...
        /Storage
            /...
    /PIC.es
    ...
    
```



The Resource Status System (RSS) is the way DIRAC keeps track of the status of sites, storage elements, Catalogs, etc. by working in conjunction with the Configuration Service that holds the definitions of the 'Grid Elements').

In order to have a more fine grained status than ON/OFF, 4 major statuses are possible for each Grid Element: Active, Degraded, Probing, Banned and even have multiple statuses

Monitoring Agents perform regular checks on each type of Grid Element and makes changes to the RSS in response to the policy settings stored in the CS

This allows a VO to set the policy on what constitutes a 'working site/service' (based on some standard tests) and tailor the job and data submissions appropriately

The Data Management System (DMS) together with the Storage Management System (SMS) provides the following:

## Upload and manage files on Storage Elements ●

*DIRAC provides an abstraction of the normal SE interfaces to provide access through a single interface, the configuration of each is handled by the Configuration Service*

## Manage FTS requests ●

*The DMS agents interface with the FTS servers assigned to your VO to manage any FTS requests. After setting the channels up in the CS, retrievals, transfers and removals can all be added to the Task Queue and thus automating the bulk transfer of data*

## Interface to the LFC ●

*DIRAC can provide it's own File Catalog but it also has an interface to the current LFC (or other FC as required). Through this both users and agents have access to Registrations, replica management, etc.*

*If you want to interact with DIRAC directly, there are two ways – Command line or Python interface*

```
> dirac-wms-job-submit job.jdl
JobID = 11758

> dirac-wms-job-status 11758
JobID=11758 Status=Waiting; MinorStatus=Pilot
Agent Submission; Site=CREAM.CNAF.it;

> dirac-wms-job-get-output 11702
Job output sandbox retrieved in 11702/
```

The command line tools allow you to submit, (using a JDL config file), get the status and retrieve the output of jobs

There is also a number of useful commands to interact with the Data Management Service

Similar results can be achieved using the DIRAC Python interface

From this, you can setup and submit jobs, manage data, etc. just like the cmd line

```
from DIRAC.Interfaces.API.Job import Job
from DIRAC.Interfaces.API.Dirac import Dirac

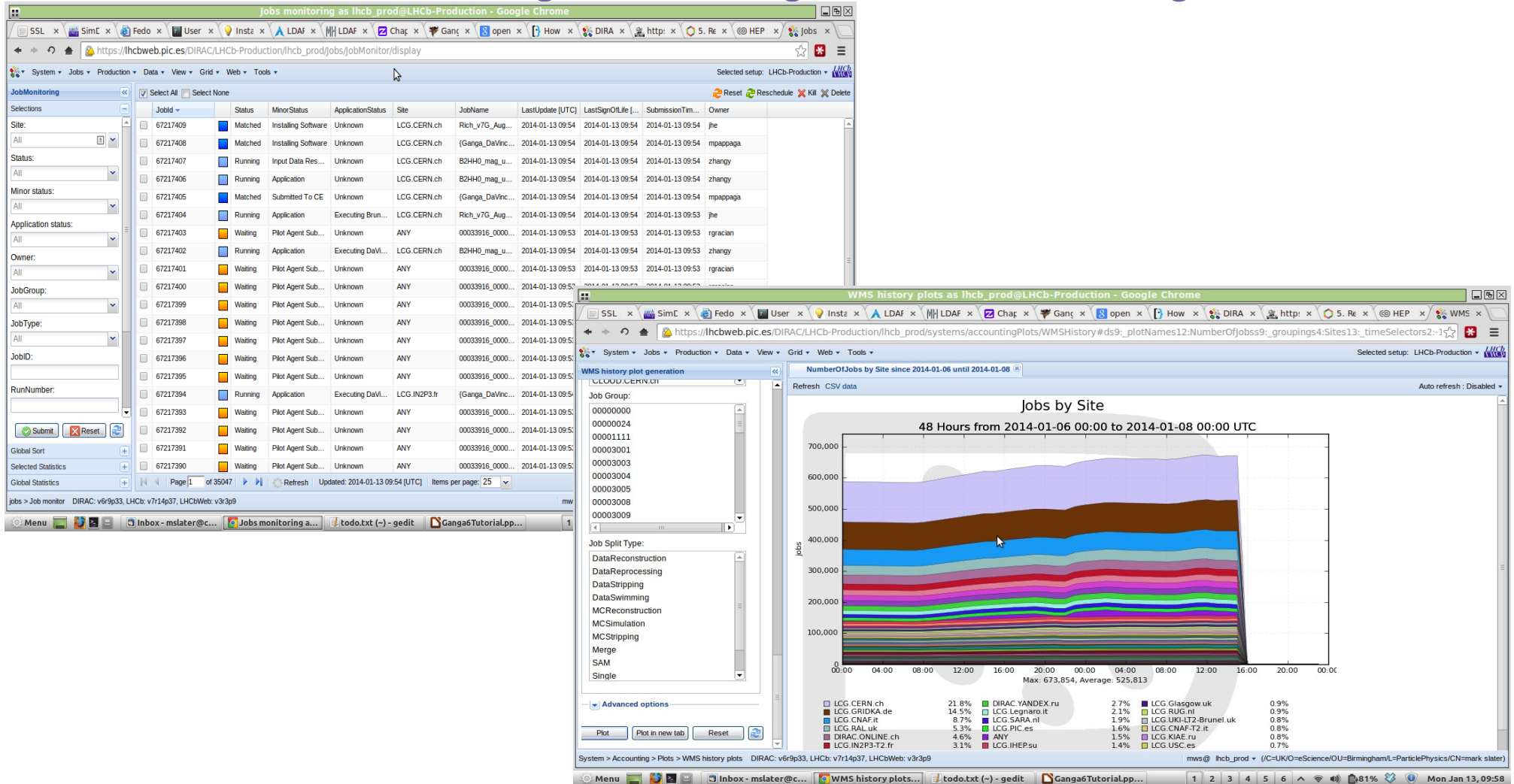
dirac = Dirac()
j = Job()

j.setCPUTime(500)
j.setExecutable('/bin/echo hello')
j.setExecutable('/bin/hostname')
j.setExecutable('/bin/echo hello again')
j.setName('API')

jobID = dirac.submit(j)
print 'Submission Result: ',jobID
```



DIRAC provides a fully-featured web portal to handle all aspects of the DIRAC instance from Job Monitoring, Data Management to Accounting



The image displays two screenshots of the DIRAC web portal. The top screenshot shows the 'Job Monitoring' interface, which includes a table of job details and various filtering options.

JobID	Status	MinorStatus	ApplicationStatus	Site	JobName	LastUpdate [UTC]	LastSignOffLife [...]	SubmissionTim...	Owner
67217409	Matched	Installing Software	Unknown	LCG.CERN.ch	Rich_v7G_Aug...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:54	je
67217408	Matched	Installing Software	Unknown	LCG.CERN.ch	(Ganga_DaVinc...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:54	mpappaga
67217407	Running	Input Data Res...	Unknown	LCG.CERN.ch	B2HH0_mag_u...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:54	zhangy
67217406	Running	Application	Unknown	LCG.CERN.ch	B2HH0_mag_u...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:54	zhangy
67217405	Matched	Submitted To CE	Unknown	LCG.CERN.ch	(Ganga_DaVinc...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:54	mpappaga
67217404	Running	Application	Executing Brun...	LCG.CERN.ch	Rich_v7G_Aug...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:53	je
67217403	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217402	Running	Application	Executing DaVi...	LCG.CERN.ch	B2HH0_mag_u...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:53	zhangy
67217401	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217400	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217399	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217398	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217397	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217396	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217395	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217394	Running	Application	Executing DaVi...	LCG.IN2P3.fr	(Ganga_DaVinc...	2014-01-13 09:54	2014-01-13 09:54	2014-01-13 09:54	rgracian
67217393	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217392	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217391	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian
67217390	Waiting	Pilot Agent Sub...	Unknown	ANY	00033916_0000...	2014-01-13 09:53	2014-01-13 09:53	2014-01-13 09:53	rgracian

The bottom screenshot shows the 'WMS history plots' interface, featuring a 'Jobs by Site' area chart. The chart displays the number of jobs over a 48-hour period from 2014-01-06 00:00 to 2014-01-08 00:00 UTC. The Y-axis represents the number of jobs, ranging from 0 to 700,000. The X-axis represents time in UTC. The chart shows a significant increase in job volume starting around 2014-01-07 00:00 UTC, peaking at approximately 673,854 jobs. A legend below the chart lists various sites and their corresponding job counts, such as LCG.CERN.ch (21.8%), DIRAC.YANDEX.ru (2.7%), and LCG.Glasgow.uk (0.9%).



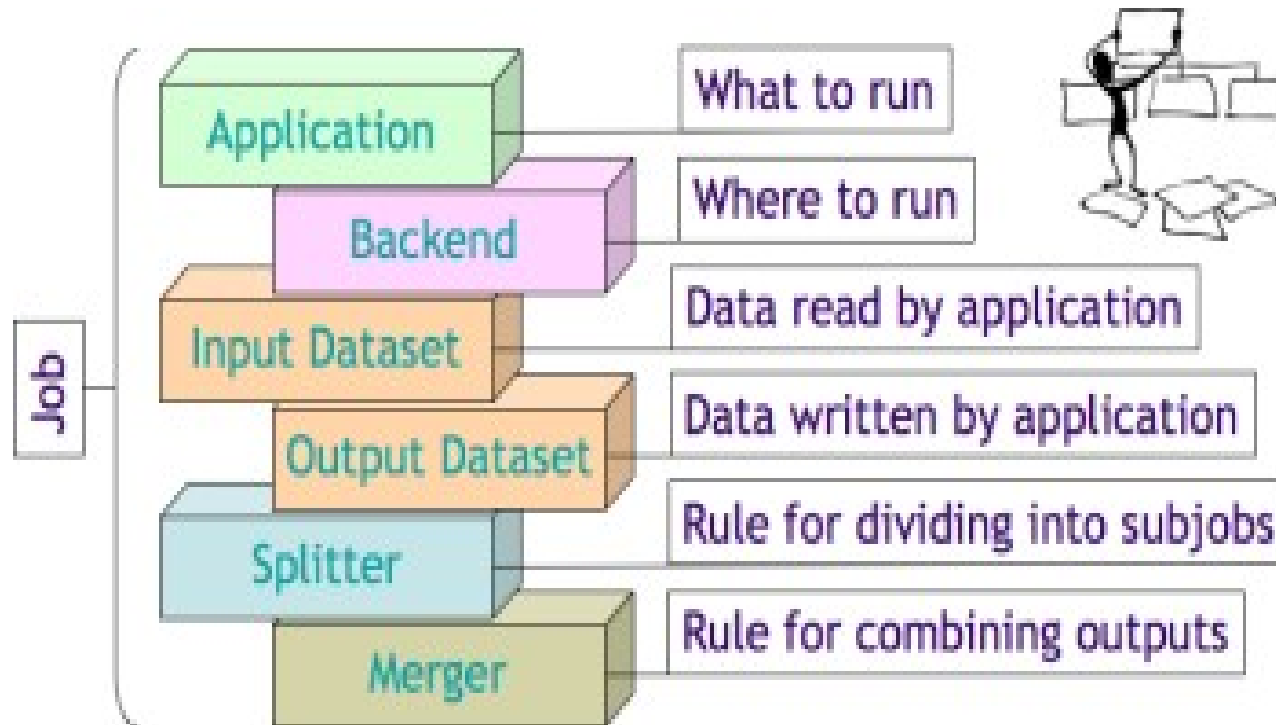


# An Introduction to Ganga



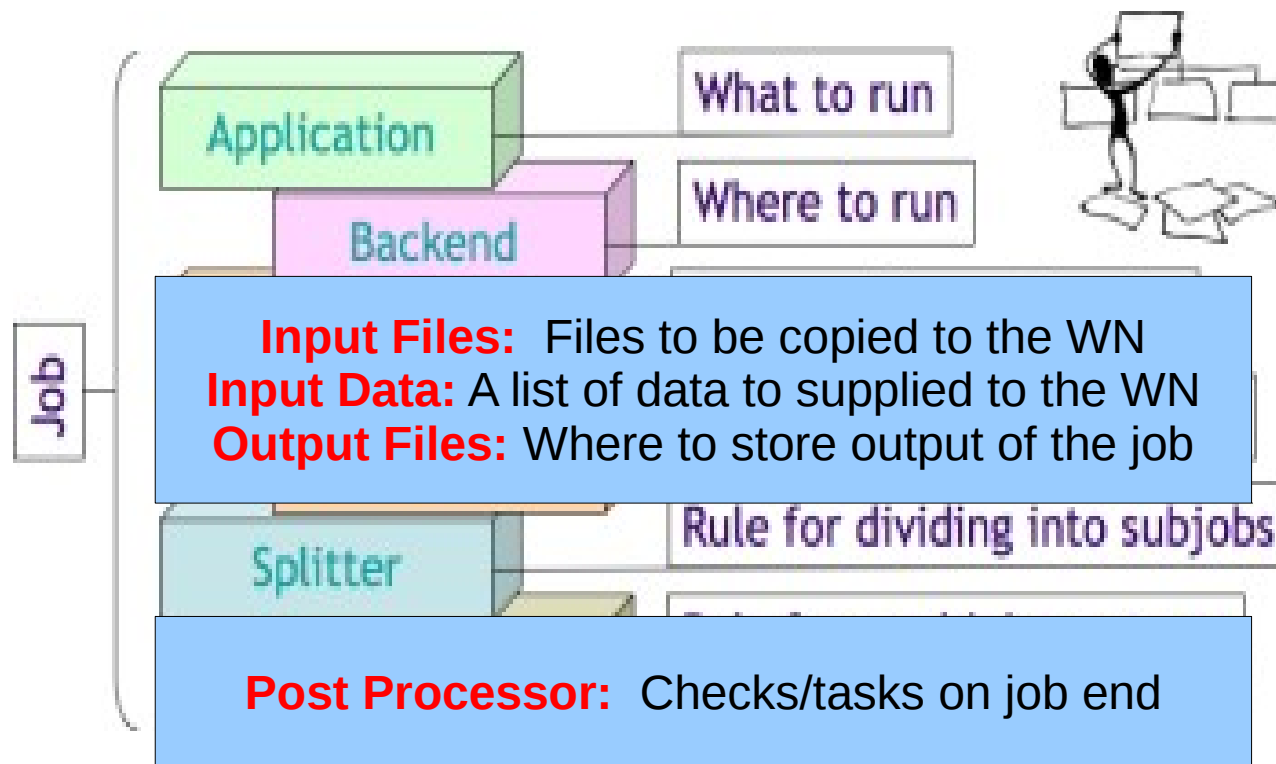
Ganga is a general job management tool used by many HEP experiments (and beyond) to simplify the submission and monitoring of both local and grid based tasks

It is built on the idea of *independent modules* that perform the various functions required by a typical job



Ganga is a general job management tool used by many HEP experiments (and beyond) to simplify the submission and monitoring of both local and grid based tasks

It is built on the idea of *independent modules* that perform the various functions required by a typical job



Most users deal with multiple computing backends – one of the principles of Ganga is to simplify changing from e.g. LSF → Dirac



There are several reasons why Ganga can help with job submission and management:

- It provides a common 'API' for many different backends
- Multiple ways of submission (command line, IPython, Service, etc)
- Hides much of the complexity for monitoring, etc.
- Easily customisable/expandable to suit the need and situation
- Written in plain python so will work with almost everything
- Active developers on hand to help
- Significantly lowers the barrier to entry for the grid
- Has many advanced features for performing complex tasks

# Typical Job Submission

*Ganga provides you with interfaces to the application your running (if available) and the backend you're running on. This allow you to completely configure the job and how it's run*

Start by creating a basic Job object

```
In [2]:j = Job()
```

```
In [3]:j.application = Executable()
```

```
In [4]:j.application.exe = File("test.sh")
```

```
In [5]:j.outputfiles = LocalFile("out.txt")
```

```
In [6]:j.backend = LCG()
```

```
In [7]:j.backend.requirements.cputime = 3600
```

```
In [8]:j.submit()
```

Set what you want the job to do (run the test.sh script)

What output is this script going to produce?

Where do you want it to run (the LCG grid in this case) and configure this as you want

And finally, submit the job!

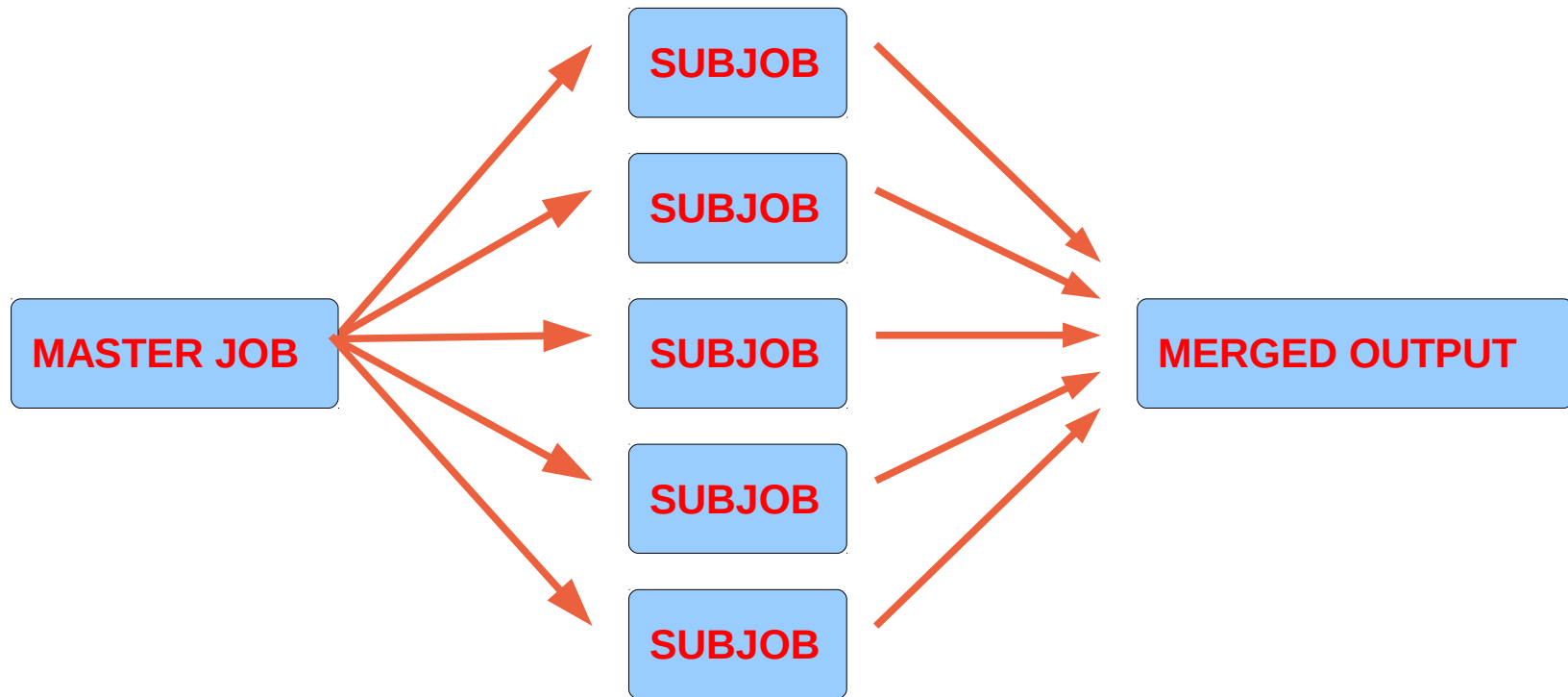


# Splitting and Merging

Where Ganga starts to show some of its power is through the splitting of jobs – i.e. taking a single job definition and creating a number of subjobs from this 'master' job config

For example, running the same program over a number of input files or running many MC generation jobs with different initial states

In addition to the splitting, Ganga can also merge the output using the 'PostProcessors' field



There are a number of plugins supplied with Ganga Core outside of the experimentally specific ones for e.g. LHCb and Atlas:

## *Backends*

*Local – The computer you're on*  
*PBS – Torque/Slurm style batch system*  
*SGE – Sun Grid Engine*  
*Condor*  
*LCG – Glite WMS Grid*  
*CREAM – Direct CREAM CE*  
*Dirac – The Dirac WMS*

## *Input/OutputFiles*

*LocalFile – A Local (direct access) File*  
*DiracFile – File handled by DIRAC DMS*  
*MassStorageFile – e.g. CASTOR*  
*GoogleFile – Google Drive*  
*CERN Box File – The new Cloud service*

## *Applications and Splitters* ●

*As each individual/VOs needs are going to be very dependent on the software they use and how it can be split, there are only a few generic ones available to run Executable or Root jobs and split any particular parameter of the application*

Ganga is incredibly configurable – the behaviour of all Ganga objects, including default values, can be set in three places:

- `.gangarc` file which lists all available variables and their description
- From the command line by specifying the options when running Ganga
- At runtime with Ganga using the 'config' dictionary style variable

In addition to this, you can also get help on all the Ganga Objects using the help system and (type 'help(<object>)')

A typical IPython tab-complete service is also available for specifying arguments as well as viewing methods/variables of a class

*As Ganga is written in Python and runs through the IPython interface, you have complete flexibility about how you submit and manipulate your jobs*

```
# only tar up once - use this for all jobs
a = Athena()
a.prepare()

for dsln in open("ds_input.txt").readlines():

    # assume you have <dsname> <jobname>
    toks = dsln.strip().split()

    # submit the job over this dataset
    j = Job()
    j.name = toks[1]
    j.application = a
    j.inputdata = DQ2Dataset()
    j.inputdata.dataset = toks[0]
    j.splitter = DQ2JobSplitter()
    j.splitter.numsubjobs = 20
    j.backend = Jedi()
    j.submit()
```

The script to the right loops over any completed jobs and retrieves the datasets produced by each

*To run these, do either:*

*ganga <scriptname> OR execfile('<scriptname>') at the python prompt*

The script to the left loops over an input file containing a dataset names and job names and submits a job for each. Note that the preparation of the Athena area only happens once!

*Also, note that Ganga can store 'job templates' as well so reducing the code to 2-3 lines!*

```
# loop over the set of completed jobs and
# grab the datasets
# (could also use jobs.select(status='completed'))

for j in jobs:

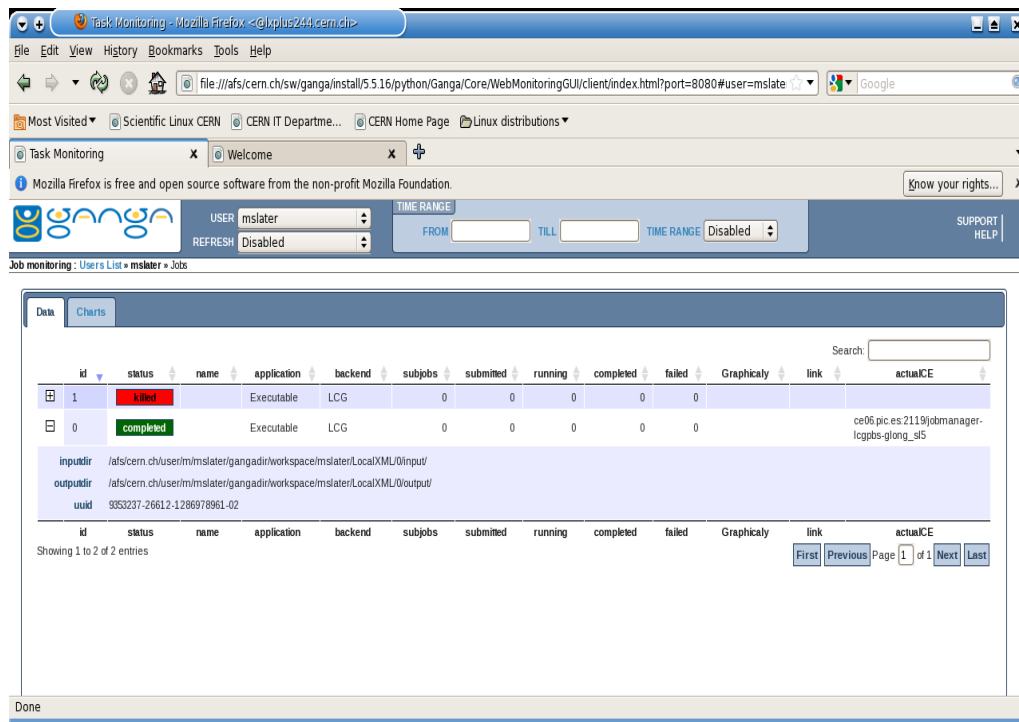
    if j.status == 'completed':
        for sj in j.subjobs():
            sj.outputdata.retrieve()

    # could also do:
    # os.system('dq2-get %s' % \
        j.outputdata.datasetname)
```

After submission, you can use Ganga to monitor and manage your jobs using Ganga's IPython interface by just typing 'ganga'

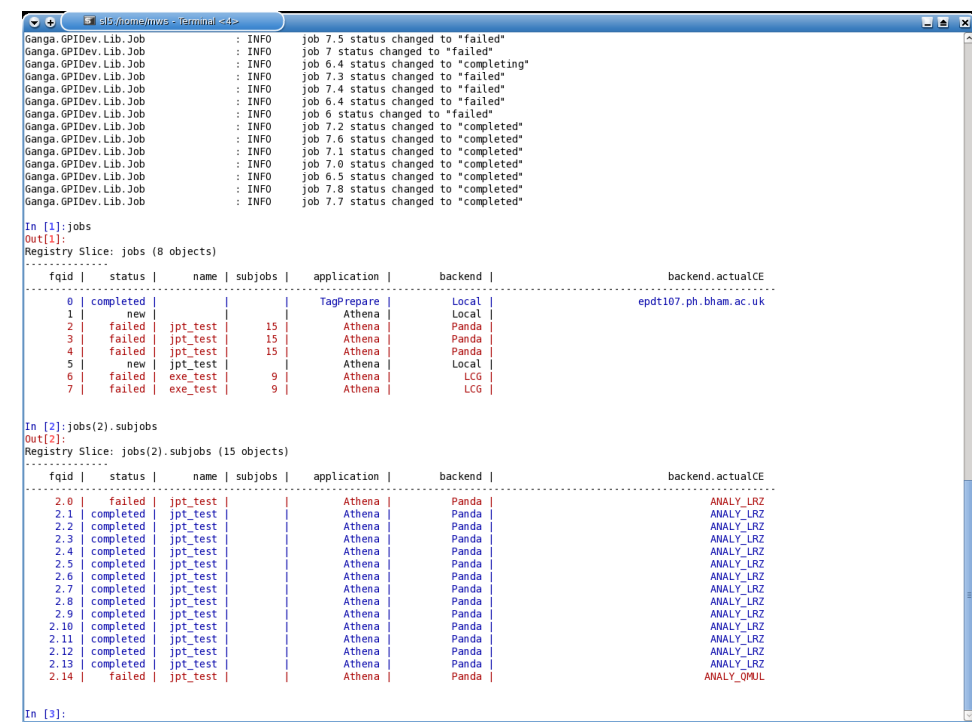
Ganga keeps track of all your jobs over all backends and gives you access to all the information using a local job repository

There is also a web gui available by starting Ganga with the --webgui option



The screenshot shows a web browser window displaying the Ganga web GUI. The main content is a table with columns for job details. The table shows two jobs: one with status 'killed' and another with status 'completed'. Below the table, there are fields for 'inputdir', 'outputdir', and 'uuid'.

id	status	name	application	backend	subjobs	submitted	running	completed	failed	Graphically	link	actualCE
1	killed		Executable	LCG	0	0	0	0	0			
0	completed		Executable	LCG	0	0	0	0	0			ce06.pic.es:2119/jobmanager- lcpbbs-glong_s45



The screenshot shows a terminal window with Ganga IPython output. It displays a list of jobs with their status and a table of job details. The table has columns for fqid, status, name, subjobs, application, backend, and backend.actualCE.

```
In [1]: jobs
Out[1]:
Registry Slice: jobs (8 objects)
.....
fqid | status | name | subjobs | application | backend | backend.actualCE
-----|-----|-----|-----|-----|-----|-----
0 | completed | | | TagPrepare | Local | epdt107.ph.bham.ac.uk
1 | new | | | Athena | Local |
2 | failed | jpt_test | 15 | Athena | Panda |
3 | failed | jpt_test | 15 | Athena | Panda |
4 | failed | jpt_test | 15 | Athena | Panda |
5 | new | jpt_test | 9 | Athena | Local |
6 | failed | exe_test | 9 | Athena | LCG |
7 | failed | exe_test | 9 | Athena | LCG |

In [2]: jobs(2).subjobs
Out[2]:
Registry Slice: jobs(2).subjobs (15 objects)
.....
fqid | status | name | subjobs | application | backend | backend.actualCE
-----|-----|-----|-----|-----|-----|-----
2.0 | failed | jpt_test | | Athena | Panda | ANALY_LRZ
2.1 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.2 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.3 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.4 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.5 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.6 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.7 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.8 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.9 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.10 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.11 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.12 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.13 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.14 | failed | jpt_test | | Athena | Panda | ANALY_QMUL
```



# Advanced Ganga Topics



***Durham Seminar, 15th October, 2014***  
*Mark Slater, Birmingham University*

As analyses progress, you will probably encounter scaling problems:

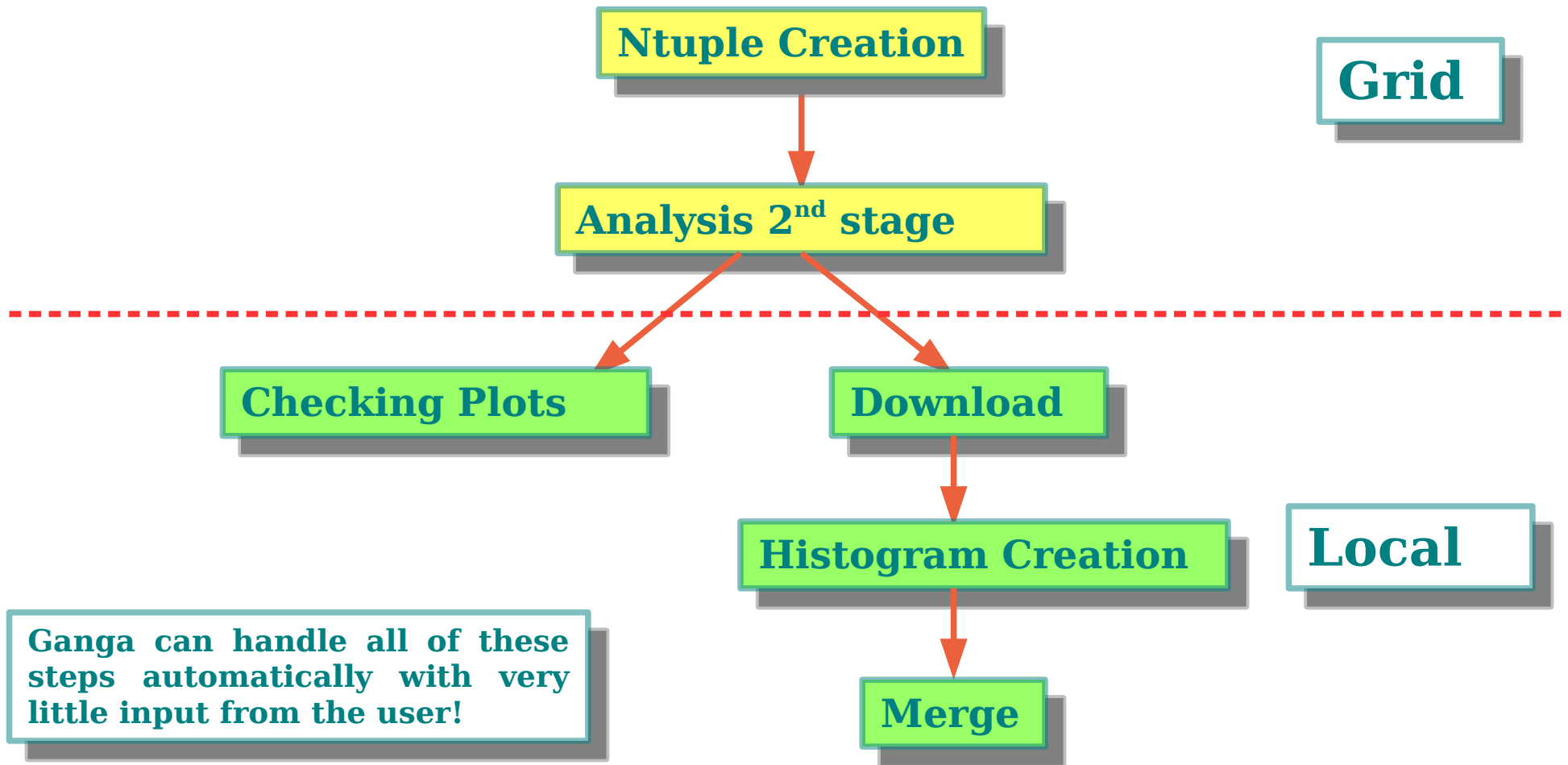
- More failures on both grid and locally ●
- Keeping track of 100s-1000s of jobs both locally and on the grid ●
- Having multi-stage analyses across multiple systems ●

Originally developed for Atlas, Ganga provides a framework to help you handle these issues – Tasks – which will (while Ganga is running):

- Resubmit failed jobs if it seems sensible to do so ●
- Submit more jobs when others complete ●
- Monitor and track running jobs, storing output data in appropriate containers ●
- Automatically retrieve and merge data ●
- Chain types of jobs together across any backend ●



# Typical Analysis Example



More info on this will soon be up on the Ganga and GridPP Twiki!

A number of jobs in Ganga can take some time, e.g. job submission, output download, etc. and quite often, these happen synchronously and hold up the user.

A similar problem existed in monitoring in that, even though jobs could in theory be monitored in parallel, this wasn't possible given the previous implementation

To get around this, a thread pool was created to handle system and user generated 'Ganga-aware' threads that:

- Execute commands asynchronously ●
- Allow tasks that can be carried out independently to be done so ●
- Balance performance issues by limiting the no. of concurrent threads ●

## Viewing the status of the queues:

```
In [4]: queues
Out[4]:
```

Ganga user threads:			Ganga monitoring threads:		
Name	Command	Timeout	Name	Command	Timeout
Worker_0	idle	N/A	Worker_0	idle	N/A
Worker_1	idle	N/A	Worker_1	idle	N/A
Worker_2	idle	N/A	Worker_2	idle	N/A
Worker_3	idle	N/A	Worker_3	idle	N/A
Worker_4	idle	N/A	Worker_4	idle	N/A

```
Ganga user queue:
-----
[]
Ganga monitoring queue:
-----
[]
```

## Adding function calls to the queue:

```
In [3]: queues.add(j.remove)
```

```
In [4]: Ganga.GPIDev.Lib.Job : INFO removing job 8
```

```
In [5]: def f(x):
...:     print x
...:
```

```
In [6]: queues.add(f, args=(123,))
```

```
In [7]: 123
```

Ganga already provides a Template system for storing often used job descriptions. As an aid to the user, we have extended this to allow these templates to be named and included in a Ganga release

This allows documentation to simply say: Use template 'Basic Analysis' and change X and Y values

This also now includes everything that can be put in the Ganga Box (the storage area for Ganga Objects), e.g. “Use Backend template 'LCG setup 1' and it should work.”

These will start to be included in GangaCore to go along with the GridPP User Guides

The Input/Output File system is very powerful and can be used to include and send files to and from all sorts of systems:

```
In [1]:j=Job()
In [2]:
In [2]:j.outputfiles = [ SandboxFile('BdKsMuMu.root'), DiracFile('Stripped.Dimuon.dst') ]
In [3]:j.outputfiles
Out[3]: [SandboxFile (
  namePattern = 'BdKsMuMu.root' ,
  compressed = False ,
  localDir = ''
), DiracFile (
  namePattern = 'Stripped.Dimuon.dst' ,
  guid = '' ,
  compressed = False ,
  localDir = None ,
  lfn = '' ,
  failureReason = '' ,
  locations = []
)]
In [4]:j.submit()
```

A LocalFile (NOT SandboxFile!) has been declared that will be copied back to the local machine

A DiracFile will copy the output to to Dirac storage and can be downloaded using the 'get' method on job completion

Previously, it was only possible for a user to specify the type of Merger they wanted to be performed on a job after completion

We have now extended this to include 3 types of post processor (but with the option to extend to arbitrary objects):

- Mergers ●
- Checkers ●
- Notifiers ●

Several types of these objects are already available, e.g. LHCbFileMerger, FileChecker, EmailNotifier

These are added to the new postprocessor job attribute that replaces the merger attribute. This is a list and so you can have as many as you want operate on one job

A Custom Merger is also provided so you can execute any arbitrary code at job completion

Simply provide a python file with a check function that returns a bool:

```
import ROOT
import os

def check(job):
    outputfile = os.path.join( job.outputdir, 'BdKsMuMu.root' )
    if not os.path.exists( outputfile ): return False
    if os.path.getsize( outputfile ) <= 100: return False

    tfile = ROOT.TFile( outputfile )
    tree = tfile.Get( 'Jpsi_Tuple/DecayTree' )
    return tree.GetEntries() >= int( job.events['input'] )
```

```
In [5]: cc = CustomChecker()
In [6]: cc.module = './checker.py'
In [7]: cc.checkSubjobs = True
In [8]: j.postprocessors.append( cc )
```

It is also relatively simple to create your own from scratch!

The GangaService was developed to fill more production roles with Ganga or when long term submissions, etc. need to be run

It behaves as you would expect a service to behave. It can:

Run Ganga as a daemon ●

Access a running Ganga instance using a simple API ●

A typical use of the GangaService is shown:

```
from GangaService.Lib.ServiceAPI.ServiceAPI import
GangaService

gs = GangaService()
gs.gangadir = "/home/mws/TaskTest/gangadir-server"
gs.prerun = "export GANGA_CONFIG_PATH='GangaAtlas/Atlas.ini'"
gs.gangacmd = "/home/mws/Ganga/install/6.0.21-
hotfix2/bin/ganga"

print gs.sendCmd("""
j = Job()
j.submit()
""")

gs.killServer()
```

Import the API

Setup how you want  
the Ganga service to  
run

Send the commands as  
if you were in IPython.  
The returned string is  
stoud/err



The current credential system is having a complete overhaul in the near future to provide:

- Multiple local and backend credentials handled (not just grid and AFS) ●
- Easy runtime management of required credentials ●
- Credential dependent submission/monitoring ●

This is in beta and should be available by the end of the year

```
# examine startup creds
for c in CredentialStore.getCredentials():
    print c.identity()
    print c.timeleft()

# create a new credential
c = VomsProxy( role = "..", proxy_path = ".." )
c.renew()

# submit job with a particular cred
j = Job()
j.credentials = [ AFSToken(), c ]
j.submit()

# Define default credentials in .gangarc:
DefaultCredentials = { {'AFSToken' : {'Backend':'LCG', 'Application':'Athena'}},
                       {'UID/Type' : {'Backend':'westgrid', 'Application':'Athena'} } }
```

One of the main advantages with Ganga is that you can relatively easily develop plugins to handle the specifics of certain elements of your use case

Any plugin type can be added, however the most usual are:

## Applications

*These usually provide a set of parameters that map to the specific program to be run and perform some checks to make sure they are sensible*

## Splitters

*These can be dependant on how you want to split up a master job into smaller subjobs, e.g. by input files or parameters and can therefore provide a simpler interface than the generic splitter provided*

If these plugins are designed correctly, they will fit seamlessly into the Ganga framework and will be able to take advantage of all the other functionality

*Note that though it is relatively easy to develop plugins for Ganga, it's recommended to get in touch with the developers to get some guidance!*



## 'Quickstart' Guide



To get going on the grid, you need a Grid Certificate and be a member of a VO:

## Getting a Grid Certificate:

<http://ngs.ac.uk/ukca/certificates>

Follow the link to 'Apply for a new certificate' and go through the instructions.

## Becoming a member of a VO:

[https://www.gridpp.ac.uk/wiki/GridPP\\_approved\\_VOs#Approved\\_EGI\\_VO](https://www.gridpp.ac.uk/wiki/GridPP_approved_VOs#Approved_EGI_VO)s

<http://www.phenogrid.dur.ac.uk/>

This depends very much on the VO but for IPPP, the Pheno VO is what you want!

*You should now have a valid grid certificate in your browser that is registered with a VO!*

Now you'll want to try to submit jobs. It's recommended to move straight to DIRAC so please use these links to get you started

*Sign up to the DIRAC instance by emailing and giving your VO:*  
*janusz.martyniak <AT> imperial.ac.uk*

*Export your Cert to your login machine:*

```
< Export your certificate from your browser as *.p12 >  
> mkdir ~/.globus  
> cd ~/.globus  
> mv ~/grid.p12 .  
> openssl pkcs12 -in grid.p12 -clcerts -nokeys -out usercert.pem  
> openssl pkcs12 -in grid.p12 -nocerts -out userkey.pem  
> chmod 400 userkey.pem  
> chmod 600 usercert.pem
```

*Install and test the DIRAC Client:*

*[https://www.gridpp.ac.uk/wiki/Quick\\_Guide\\_to\\_Dirac](https://www.gridpp.ac.uk/wiki/Quick_Guide_to_Dirac)*

Install Ganga using these instructions (if there is no 'global' version):

<http://ganga.web.cern.ch/ganga/user/installation/other.php>

When running for the first time:

Before running Ganga, setup DIRAC and run '`env > /my/dirac/envfile`'

Run ganga with the '-g' flag to create a .gangrc file

Edit this file and set '`[Configuration]RUNTIME_PATH=GangaDirac`'

In addition, set '`[Dirac]DiracEnvFile=/my/dirac/envfile`'

When running Ganga after that:

*Always have the DIRAC environment setup!*

There will be an in-depth Twiki appearing at:

[https://www.gridpp.ac.uk/wiki/Guide\\_to\\_Ganga](https://www.gridpp.ac.uk/wiki/Guide_to_Ganga)

You can always get help at:

[project-ganga<AT>cern.ch](mailto:project-ganga@cern.ch)

[project-ganga-developers<AT>cern.ch](mailto:project-ganga-developers@cern.ch)

Or emailing me: [mws@hep.ph.bham.ac.uk](mailto:mws@hep.ph.bham.ac.uk)