

**Imperial College
London**

Ganga
GridPP13, Durham

Ulrik Egede
On behalf of the Ganga team

If you are bored (and connected) try:

```
/afs/cern.ch/sw/ganga/install/slc3_gcc323/4.0.0-alpha7/  
    prototype/Ganga4/python/Ganga/bin/ganga
```

Then just follow examples in presentation

Demonstration can be given afterwards!

Who is involved in Ganga

GridPP

Alexander Soroko, Karl Harrison, Chun Lik (Alvin) Tan

ARDA

Andrew Maier, Kuba Moscicki

UK academic

Ulrik Egede

Also involved

Glenn Patrick, Roger Jones, Stuart Paterson, Dietrick Liko, Birger Koblitiz,
Massimo Lamanna, Philippe Charpentier

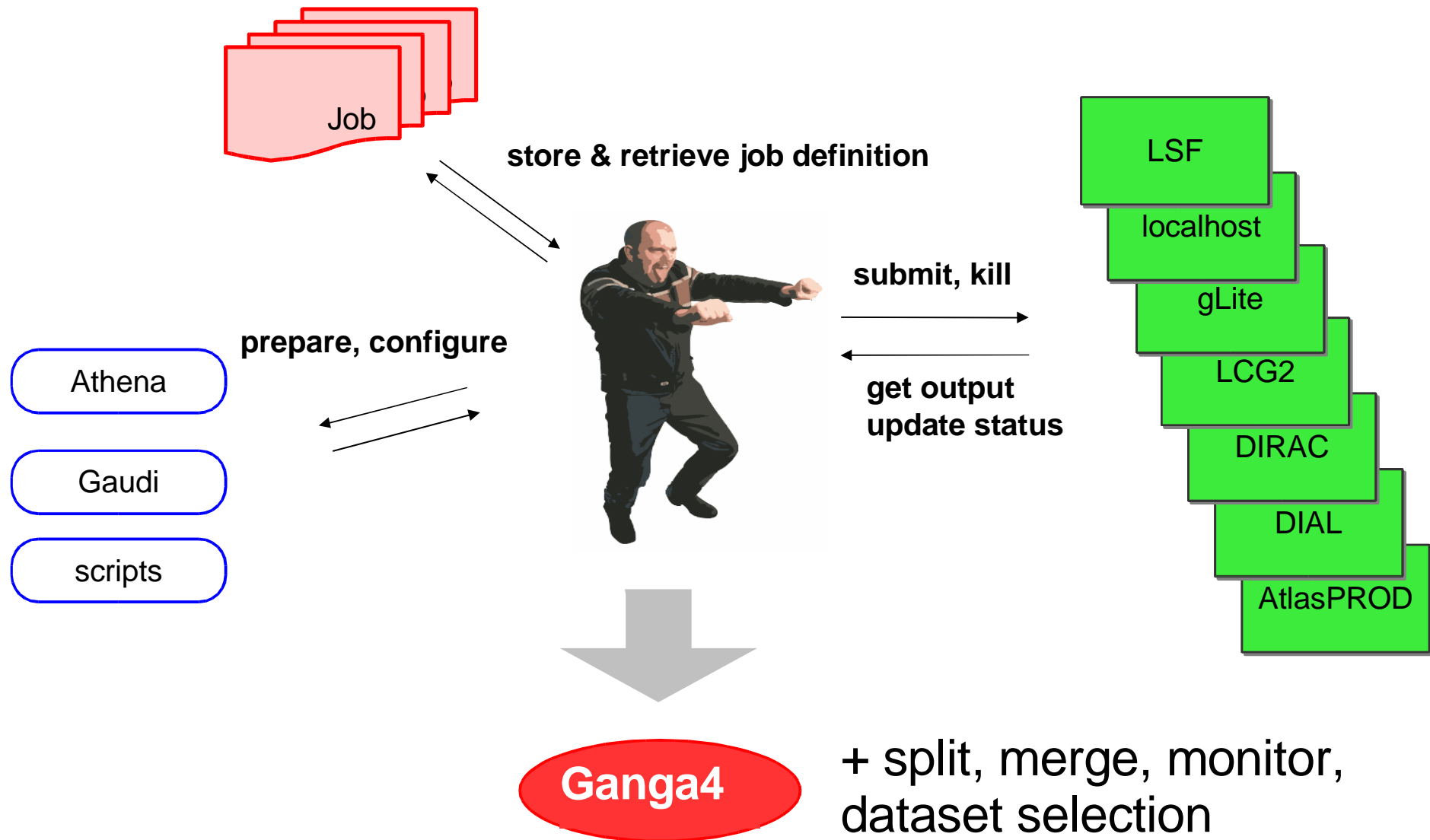
If you are bored (and connected) try:

```
/afs/cern.ch/sw/ganga/install/slc3_gcc323/4.0.0-alpha7/  
prototype/Ganga4/python/Ganga/bin/ganga
```

Then just follow examples in presentation

Demonstration can be given afterwards!

What is Ganga

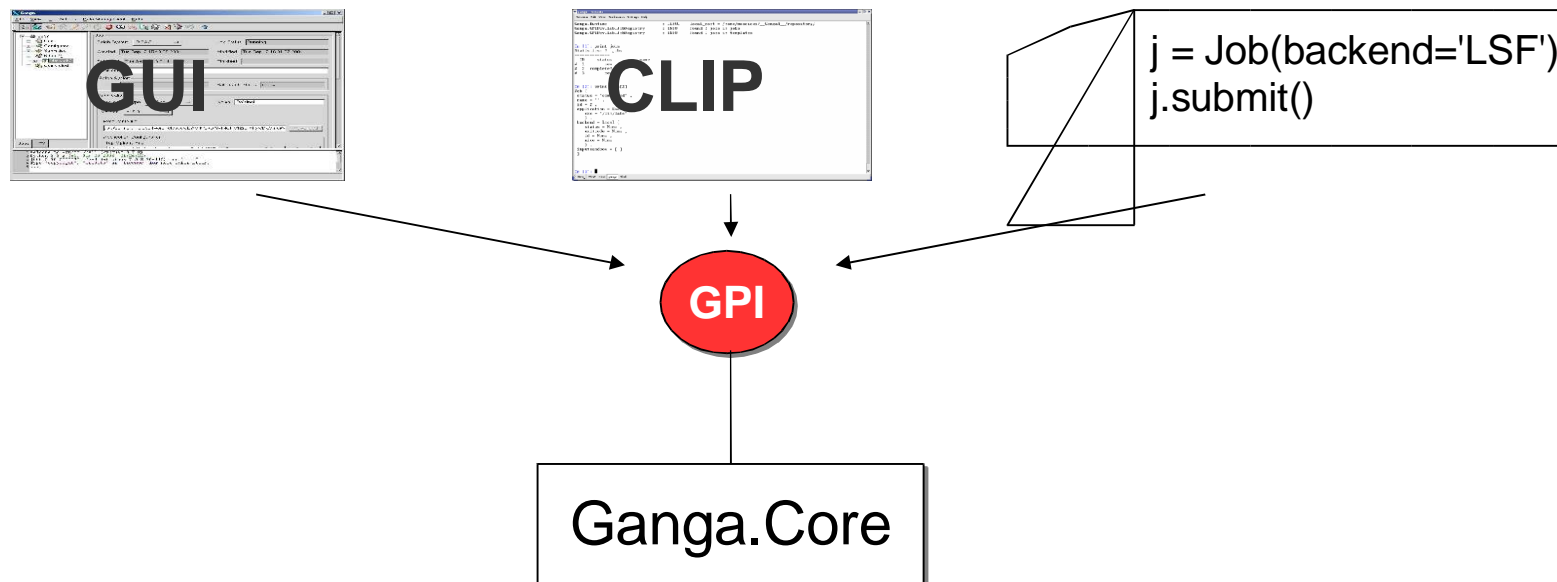


Ganga Public Interface

Ganga Public Interface: GPI

High-level, user-friendly Python API for job manipulation.

Combines consistency and flexibility of programming language with clarity and ease of use.



Ganga Public Interface

Hello World

```
In [1]: job = Job(name='Simply job')
In [2]: job.application.exe='/bin/hostname'
In [3]: job.submit()
```

```
Ganga.GPIDev.Lib.Job : INFO      submitting job 5
Ganga.Lib.Localhost  : INFO      job 5 submitted, local pid = 6871
```

```
In [5]: print file(job.outputdir+'/stdout').read()
```

```
lxplus010.cern.ch
```

```
In [10]: job2 = job.copy()
In [11]: job2.backend='LSF'
In [12]: job2.submit()
```

```
Ganga.GPIDev.Lib.Job : INFO      submitting job 6
Ganga.Lib.LSF        : INFO      LSF: submitting job 6
Ganga.Lib.LSF        : INFO      using default queue "8nm"
Ganga.Lib.LSF        : INFO      job 6 submission OK
Ganga.Lib.LSF        : INFO      6: LSF job status changed to RUN
Ganga.Lib.LSF        : INFO      6: LSF job status changed to DONE
```

Ganga Public Interface

Get an overview

```
In [38]: print job.id
```

```
5
```

```
In [39]: print jobs
```

```
Statistics: 6 jobs
```

```
-----  
   ID      status      name  
#   5  completed  Simply job  
#   6  completed  Simply job  
#   7           new  Another copy  
#   9           new    DaVinci  
#  10           new  Yet Another copy  
#  11           new  DaVinci v12r7
```

```
In [40]: newjobs = [i for i in jobs if jobs[i].status=='new']
```

```
In [41]: for j in newjobs:
```

```
.....:     print jobs[j].id
```

```
7
```

```
9
```

```
10
```

```
11
```

```
In [42]:
```

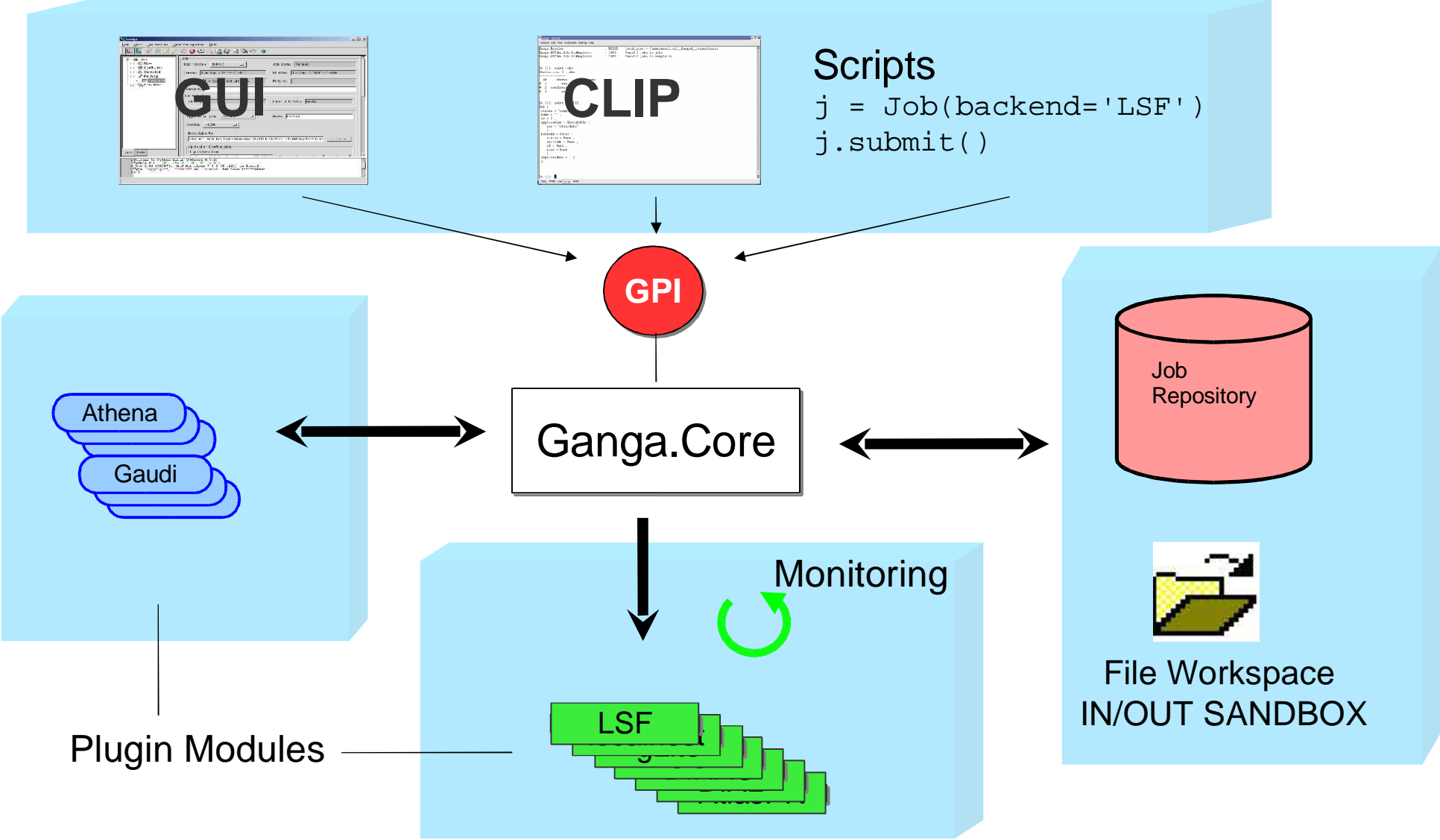
Ganga Public Interface

Complex scenarios

```
>>> j = Job()
>>> j.application = DaVinci()
>>> j.application.cmt_user_path = '/afs/cern.ch/user/u/uegede/cmttest'
>>> j.backend = Glite()
>>> j.backend.requirements = 'other.GlueCEUniqueID == "grid-
                             ce.desy.de:2119/jobmanager-lcgpbs-short"'

>>> for i in range(100):
    j = Job()
```

Ganga Architecture



Ganga Tool vs Framework

Ganga is a lightweight user tool

- Easy to install (pure Python, around 500 kB)

- Focus on use cases when developing syntax

But Ganga is also a developer framework

- Plugin model

 - Independent and rapid development of handlers (backends, applications)

- Promote but not force common GPI abstractions

 - We do not require nor invent abstract base classes which are least common denominators between systems, example:

 - You may implement very complex application (e.g. ADA) and enable submission to DIAL only if that's your main case.

 - The design of the framework does not attempt to match all possible applications with all possible backends.

This provides means to build common tools on top of GPI: GUI, scripts,...

Extending Ganga

Creating a Ganga handler is easy

Use the plug-in mechanism

Examples:

Application handler for Gaudi framework jobs

Backend handler for DIAL

Create code:

Create a class derived from GangaObject

Define a schema of data the handler requires

Implement the interface for the handler

Update the config file ~/.ganga4

```
[Configuration]
RUNTIME_PATH = ~/MyExtensions
```

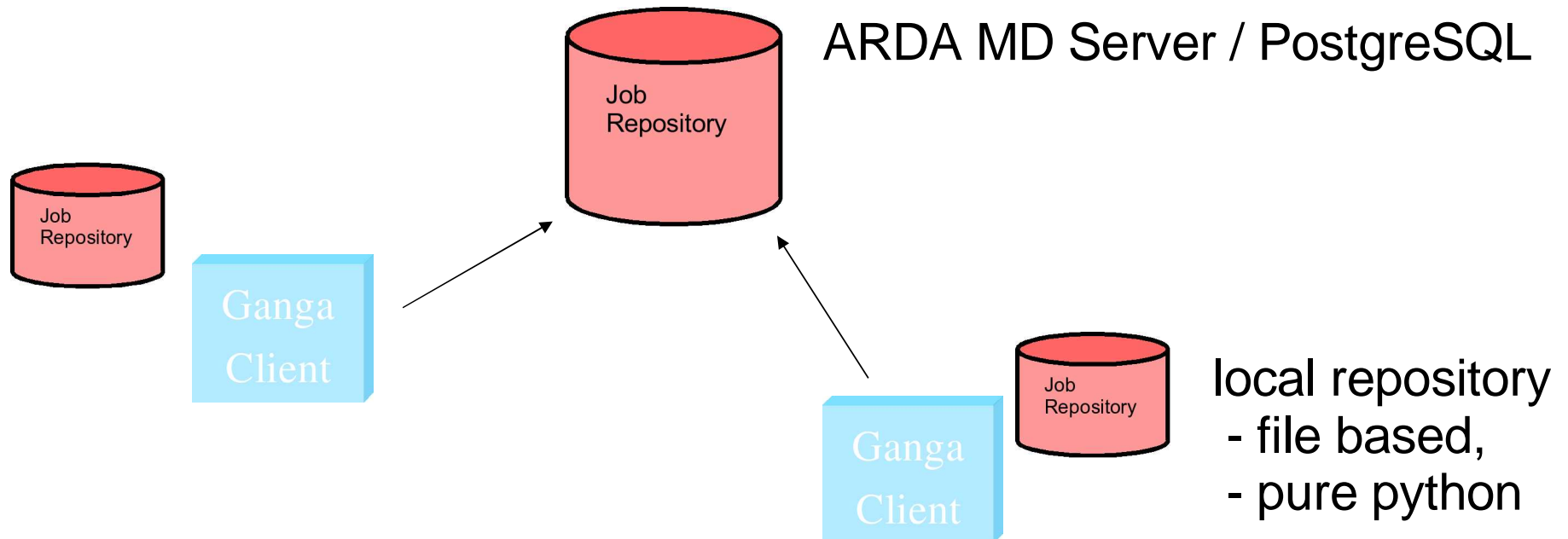
No changes required to centrally released code

Remote Job Repository

Remote job repository

Access to the your jobs from multiple locations

In the future possibility to share jobs between users



Ganga 3

Ganga 3.0 released March 2005

User guide available on Ganga web site:

<http://ganga.web.cern.ch/ganga/>

A lot of useful functionality included

Job configuration and management through Python CLI or through GUI

Support for Gaudi and Athena applications

Job-Options Editor (JOE)

Access to AMI

Submission to backends including PBS, LSF, LCG, gLite, Dirac

Issues:

Plug-in mechanism hard to use.

Performance problems with job repository.

Problems addressed in redesign for Ganga 4

Features of 4-0-0-alpha7

Existing functionality:

- Basic job manipulation via GPI

- Easy configuration / extension

- Local and remote registry, local workspace

- Local host, LSF, DIAL backends

- Gaudi (DaVinci, Gauss,...) , Athena, ADA applications

In testing:

- LCG, DIRAC, GLite, ATLAS ProdSys backends

Future functionality:

- Splitting/merging

- Remote application and job manager

- Monitoring component

Experiment integration: LHCb

Full support of LHCb analysis jobs in the Gaudi framework:

Automatically guess properties for Gauss (simulation), Boole (digitisation), Brunel (reconstruction), DaVinci (analysis)

Applications are configured via CMT to:

- Flatten the options file

- Extract shared libraries modified with respect to release

Integration of capabilities with Dirac production system

- Use of Python API for job submission

- Consider using Dirac Python wrappers to achieve backend independence of jobs.

- Will use Dirac interface for contact to Storage Elements

Ongoing tests in Bologna of full functionality

Data for analysis is becoming available now at Tier-1's so LHCb needs Ganga now.

Experiment integration: ATLAS

DIAL submission from Ganga

- Create instances of DIAL catalogues in Ganga

 - Plug-in that interacts with DIAL backend

 - Plug-in that uses ADA/DIAL job description

 - use of object to XML converters

- From Ganga can do following:

 - Query the DIAL catalogues

 - Submit jobs to (remote) DIAL services

 - Retrieve job output (also partial results)

- Via pyROOT, display output histograms

Direct submission of Athena jobs to ATLAS Prodsys

Future integration will depend on outcome of ATLAS distributed analysis review

Conclusion

Ganga is a framework that allows individual physicists to perform specific tasks in an easy way on the Grid.

Provides an easy transition from current batch based analysis to Grid based analysis.

Users can write scripts or work interactively with Ganga using concepts they are already familiar with.

Developers can easily extend Ganga to provide new functionality via plugin modules.

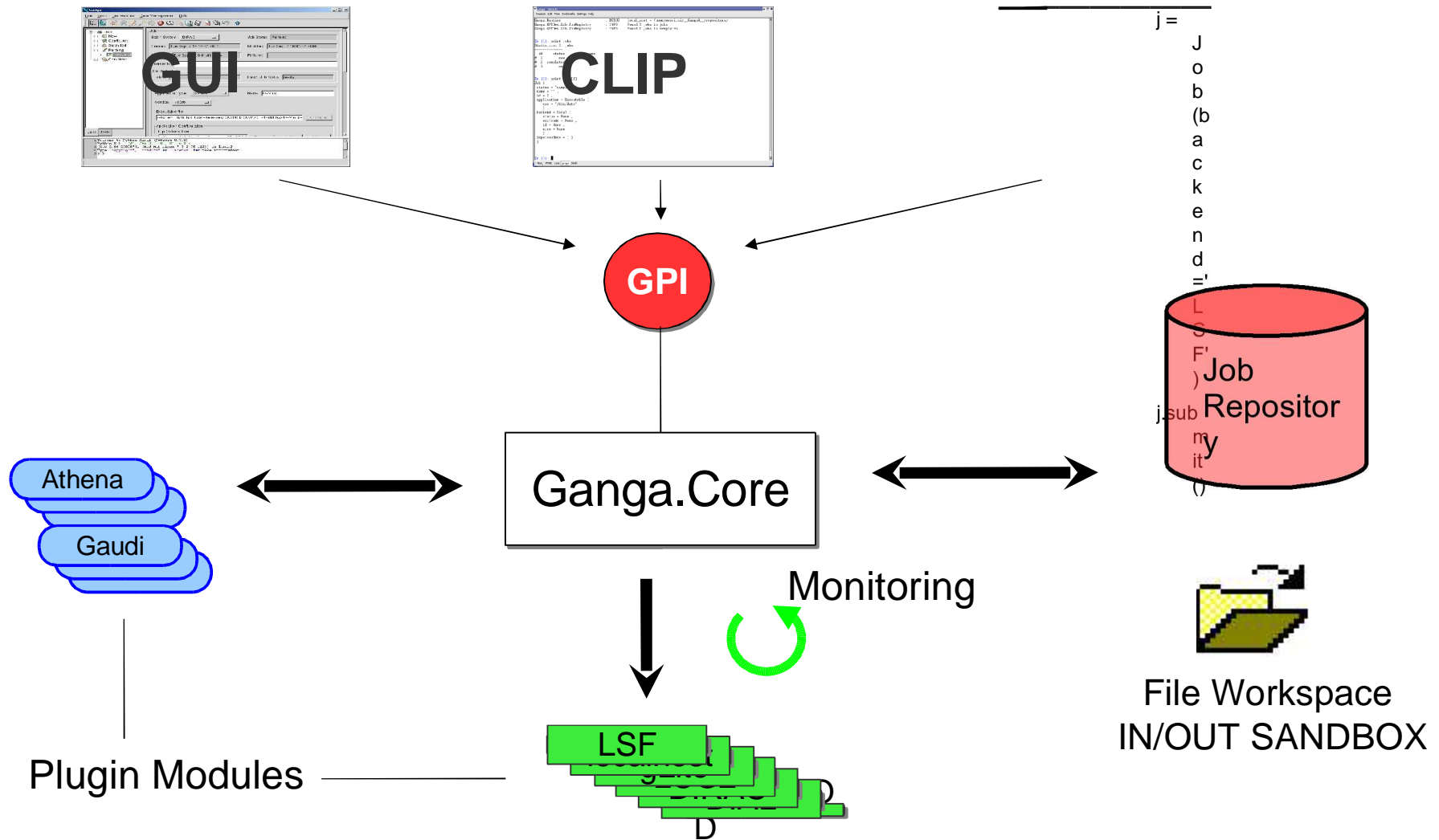
Beta release will take place this week.

For more information look at <http://cern.ch/ganga>.

I can provide a demonstration later today.

Backup Slides

Ganga Architecture

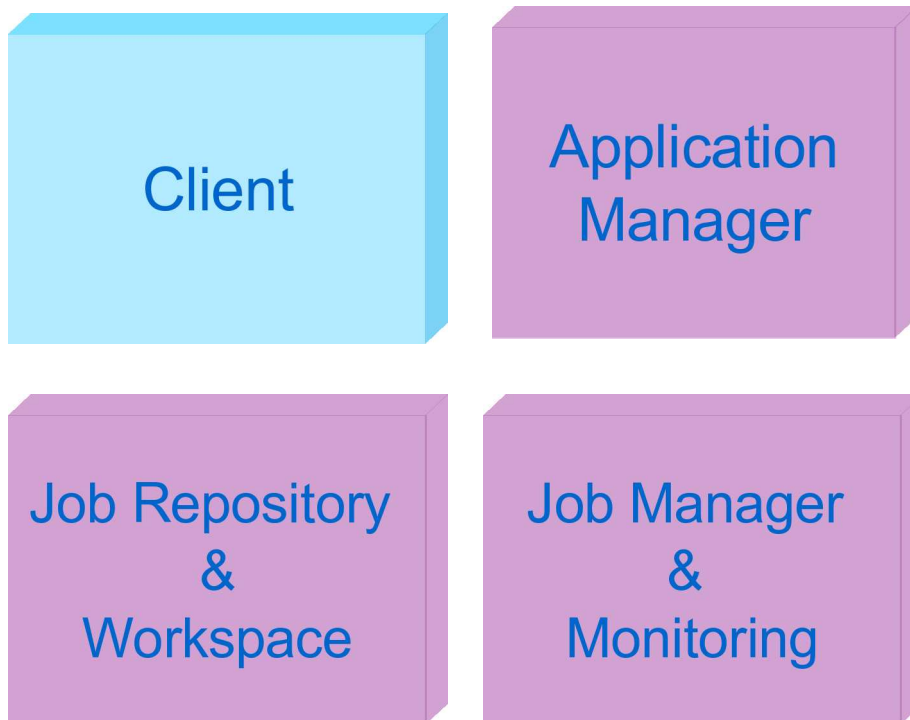


Internal architecture



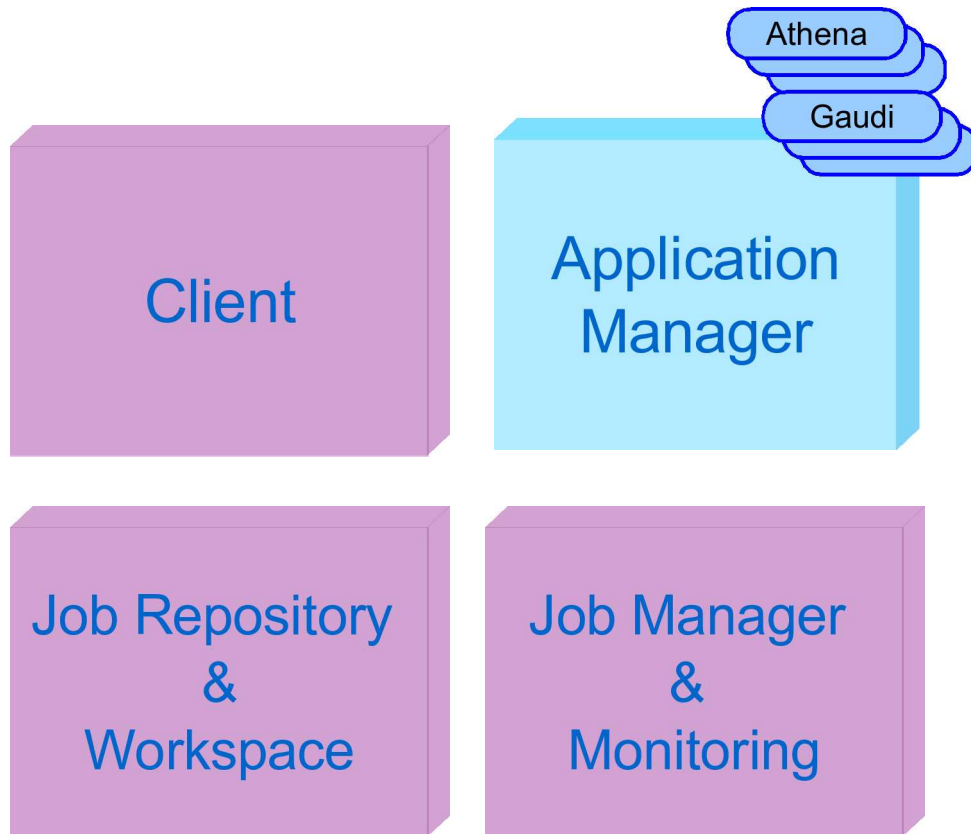
- Ganga 4 is decomposed into **4 functional components**
- These components also describe the components in a **distributed model**.
- Strategy: Design each **component** so that it could be a **separate service**.
- But **allow to combine** two or more **components** into a single service

Client



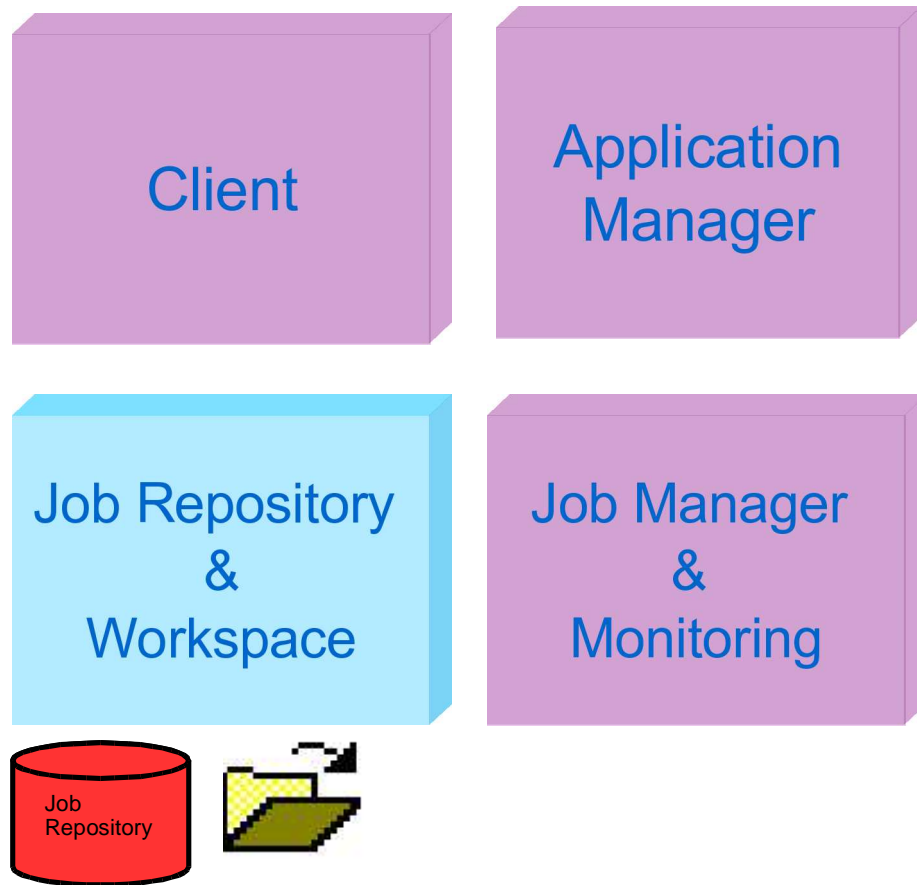
- Client runs CLIP, GUI and GPI scripts.
- **User interacts exclusively via the Client.**
- Implementation:
 - pure python
 - small size (~500 kB)

Application Manager



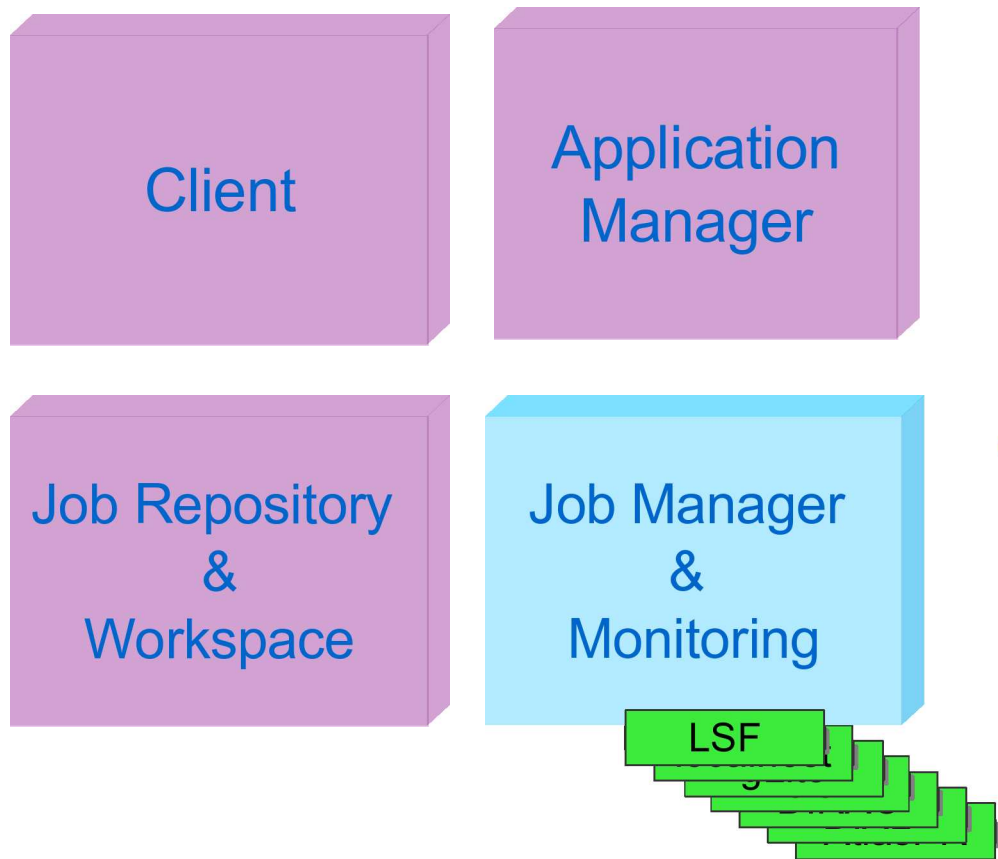
- Prepares and configures the application, e.g.:
 - preprocess options
 - compile user code
 - setup environment
- The work is done by **Application Handlers**

Job Repository and Workspace



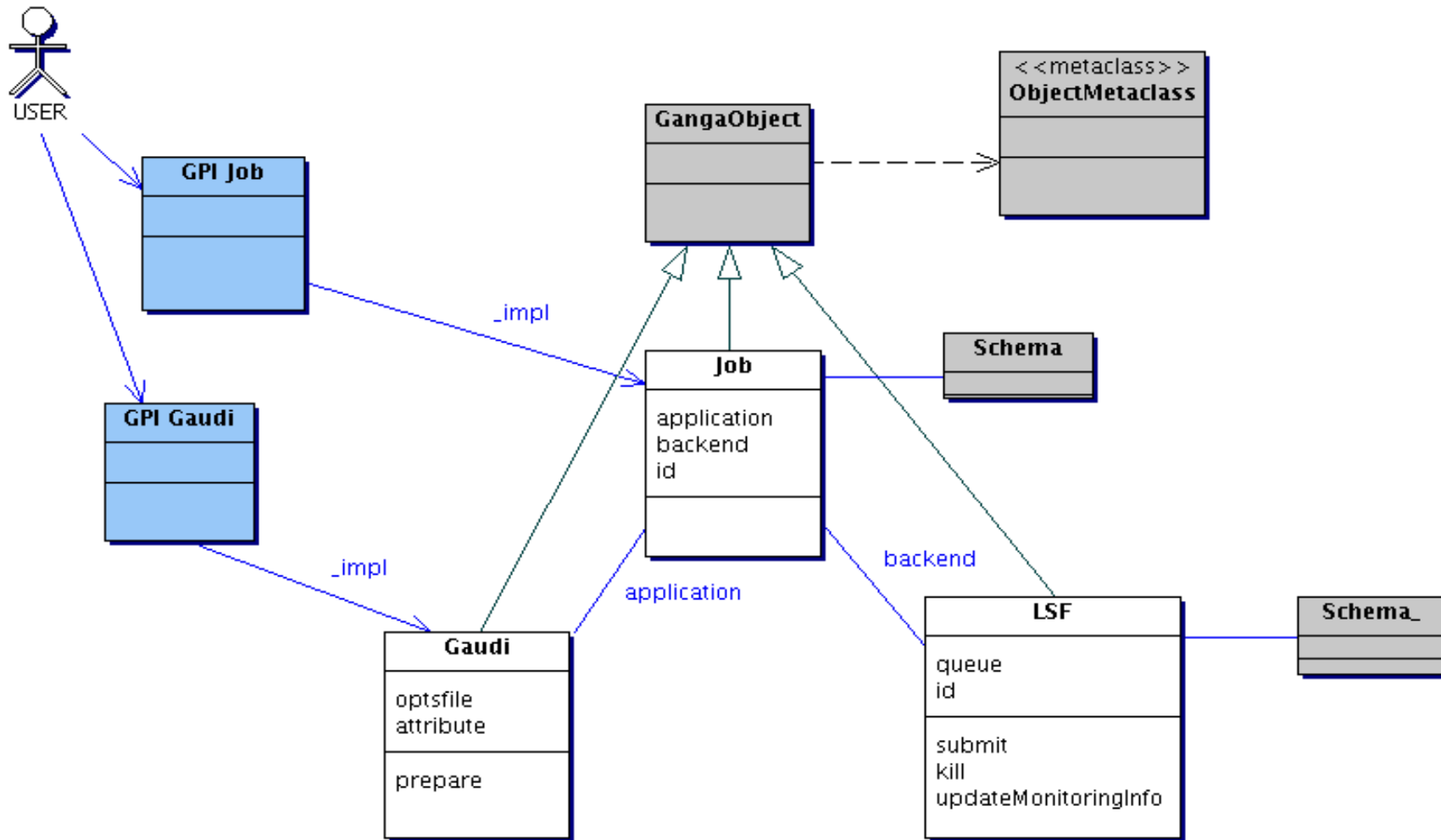
- Job Repository:
 - keeps track of jobs
 - stores job metadata “roaming-profile”
 - may be Local or Remote
- File Workspace
 - caches the job output
 - allows to share files within Ganga

Job Manager



- Submits configured jobs to the submission backends using **BackendHandlers**, e.g.:
 - creates wrappers scripts
 - created JDL files
 - etc.
- Monitors the status of jobs
 - by polling the backends
 - by monitoring notificationsNotifications allow to see changes in jobs submitted outside of Ganga

Ganga Object Model



Handler Plugin Example

```
## Import the GPI
from Ganga.GPIDEv.Base import GangaObject
from Ganga.GPIDEv.Schema import *
[...]
```

```
class Gaudi(GangaObject):
```

```
    _schema = Schema(Version(1,0), {'optsfile':FileItem(),
                                   'version':SimpleItem(None),
                                   'platform':SimpleItem(None),
                                   'package': SimpleItem(None),
                                   'appname':SimpleItem(None)})
```

```
    _category='applications'
```

- Create a class based on GangaObject.
- Provide a Schema for you data
- Fulfil the interface for the type of handler

Describe the data needed to
configure and run an application

Set the type of your plugin

Handler Plugin Example

```
def configure(self):  
    [...]  
    extra=GaudiExtras()  
    [...]  
    extra.flatopts=FileParser.writeString(gaudiopts, "expand")  
    return (None,extra)  
  
def list_choices(self,property):  
  
class GaudiExtras(GangaObject):  
    _schema = Schema(Version(1,0),{"flatopts": SimpleItem(None)})  
    _name="GaudiExtras"  
    _category="extras"  
    _name='Gaudi'
```

Use a GaudiExtras object to return information added by configure

Create a Schema with the data of the configured application
Set the category of this class to "extras"

Gaudi Application Object

```
class Gaudi(GangaObject):
    _schema = Schema(Version(1,0),{
        'optsfile': FileItem(),
        'version': SimpleItem(None),
        'platform': SimpleItem(None),
        'package': SimpleItem(None),
        'appname': SimpleItem(None),
        'cmt_release_area': SimpleItem(None),
        'cmt_user_path': SimpleItem(None),
        'masterpackage': SimpleItem(None),
        'extraopts': SimpleItem(None)})

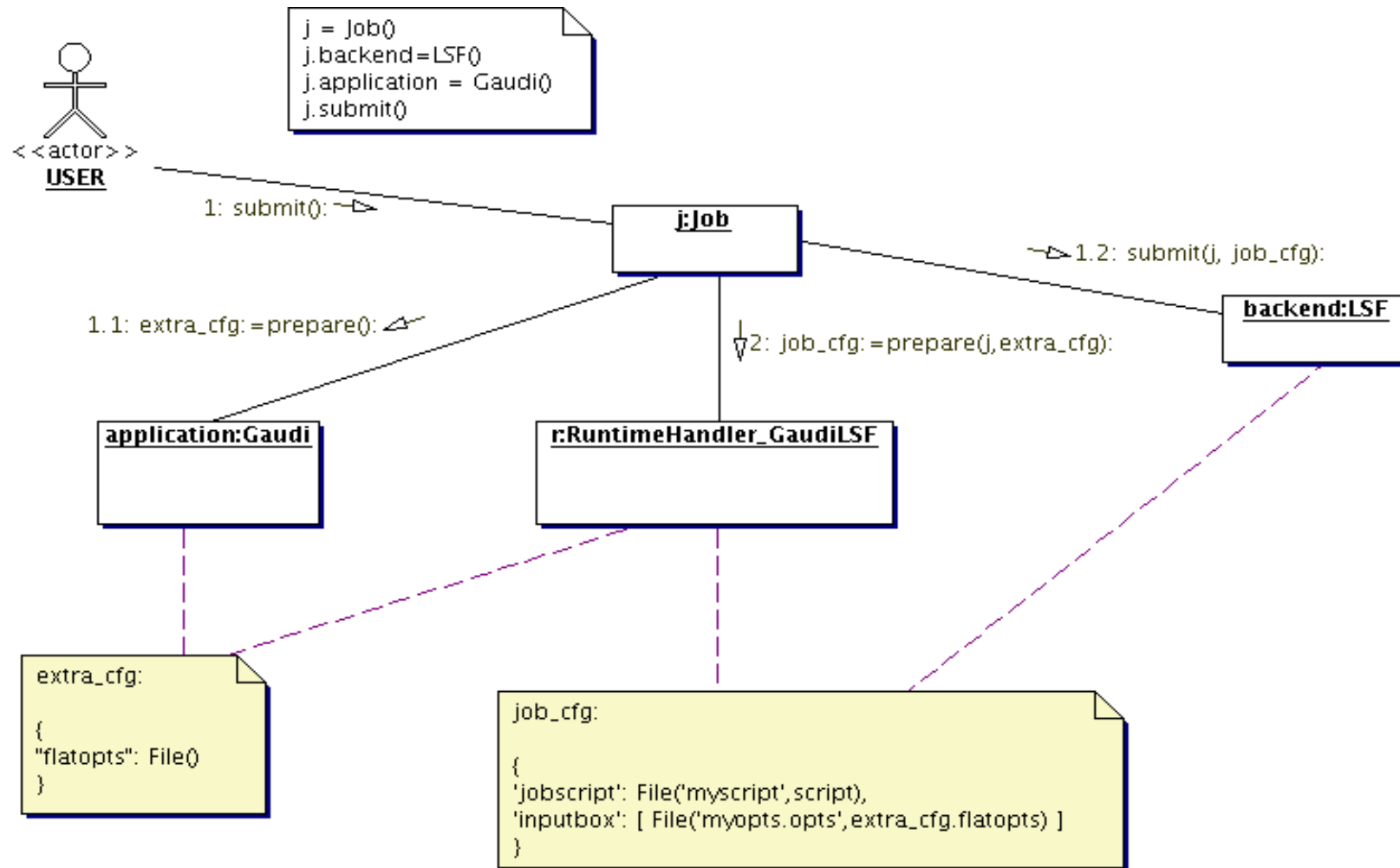
    _category='applications'
    _name='Gaudi'

    def __auto__init__(self):
        ...

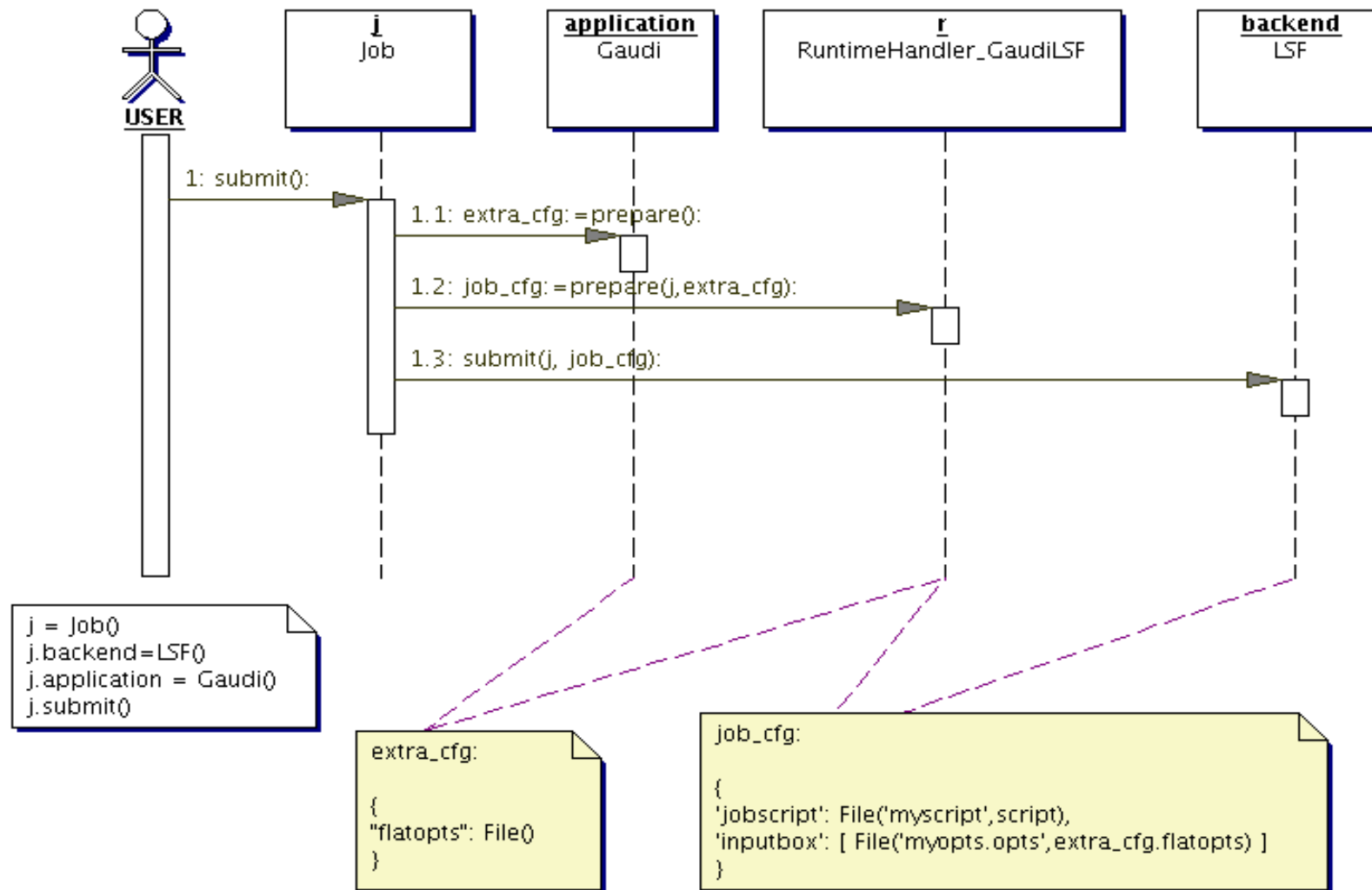
    def configure(self):
        ...
        extra_cfg=GaudiExtras()
        extra_cfg.flatopts=FileParser.writeString(gaudiopts,"expand")
        return (modified, extra_cfg)

    def list_choices(self,property):
        ...
```

Job Submit



Job Submit Sequence



Files/Job Repository

- File Workspace
 - ~/__Ganga4__/workspace/input/*
 - ~/__Ganga4__/workspace/output/*
- Job Repository
 - ~/__Ganga4__/repository/ganga_user

Design Principles

CLI Design Principles

Be predictable and follow python way of thinking

Increase complexity of interface with complexity of task:

Simple tasks – simple!

Complicated tasks – also simple ;) !

Try to prevent users from silent mistakes:

`job.id = 5` # *FAILS: id is a read-only property*

`finished_job.name = 'newname'` # *FAILS: job is finished so can't modify*

Hide implementation:

`job._impl.attrs['id'] = 5`

Be convenient and guide users

`j.application.exe` \Leftrightarrow `j.exe` # *ALIASES of properties*

TAB completion shows properties and hides internals

Be flexible: good for writing complex macros/scripts...