Workload Management System (WMS)

Valeria Ardizzone, Giuseppe La Rocca
INFN

EGEE Tutorial

Rome, 02-04.11.2005

www.eu-egee.org

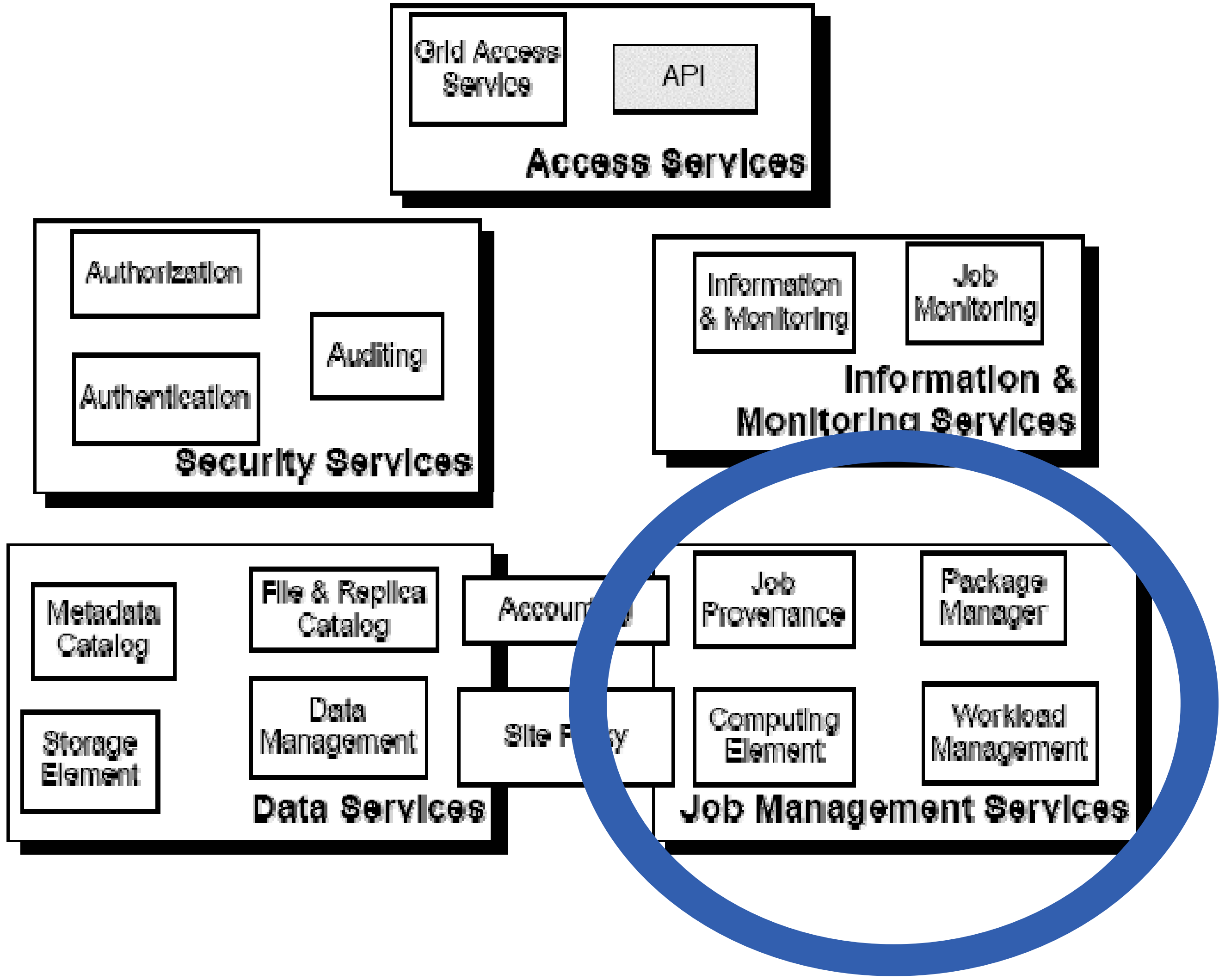
Outline

This part covers the following arguments:

- Overview of gLite middleware
- Overview of WMS Architecture
  - Task Queue, Information Supermarket, MatchMaker, Scheduling Policies, Job Submission Service, Job Logging & Bookkeeping.
- Job Description Language Overview
  - Principal Attributes

Enabling Grids for E-sciencE

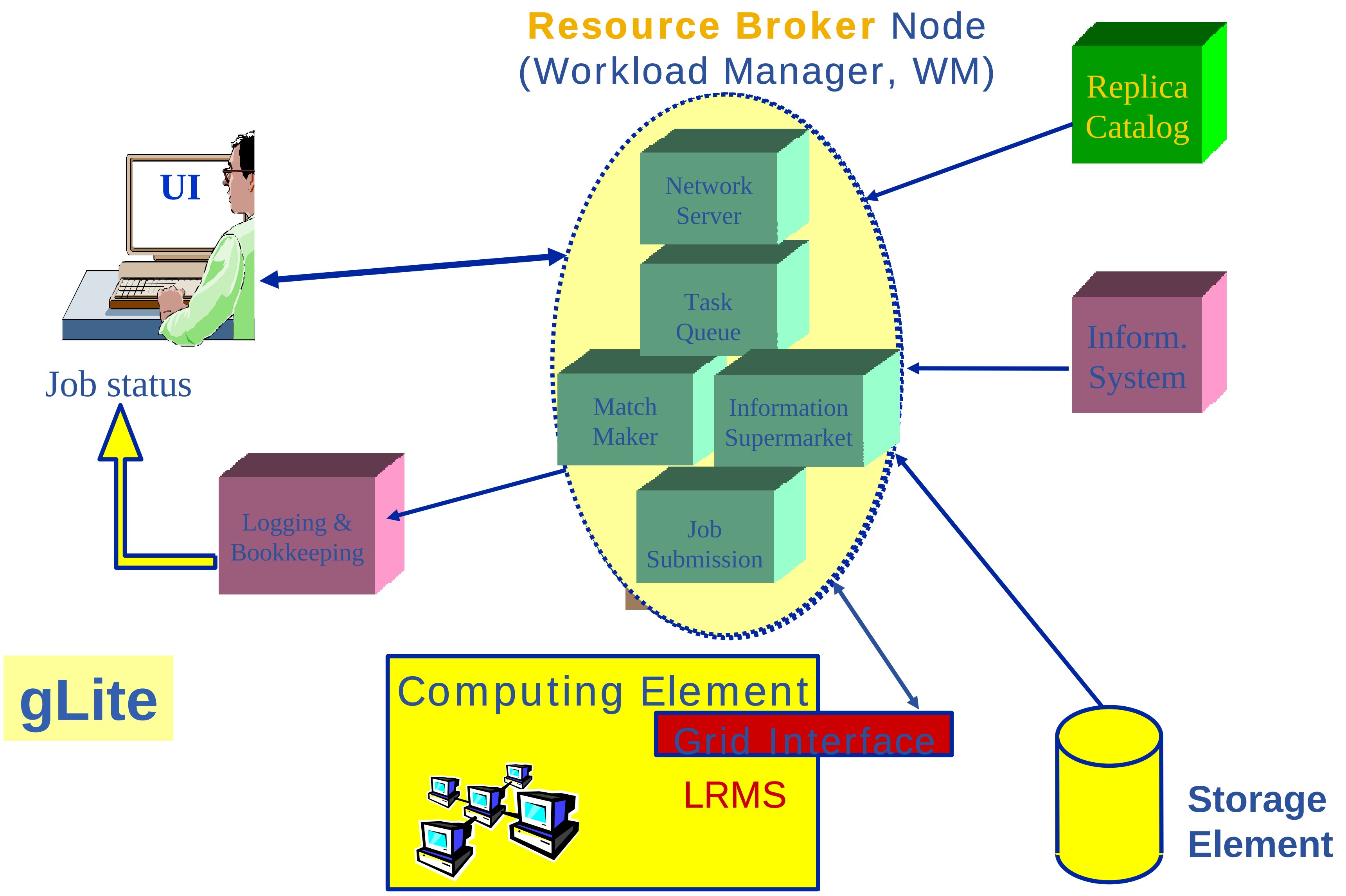
Overview of EGEE Middleware



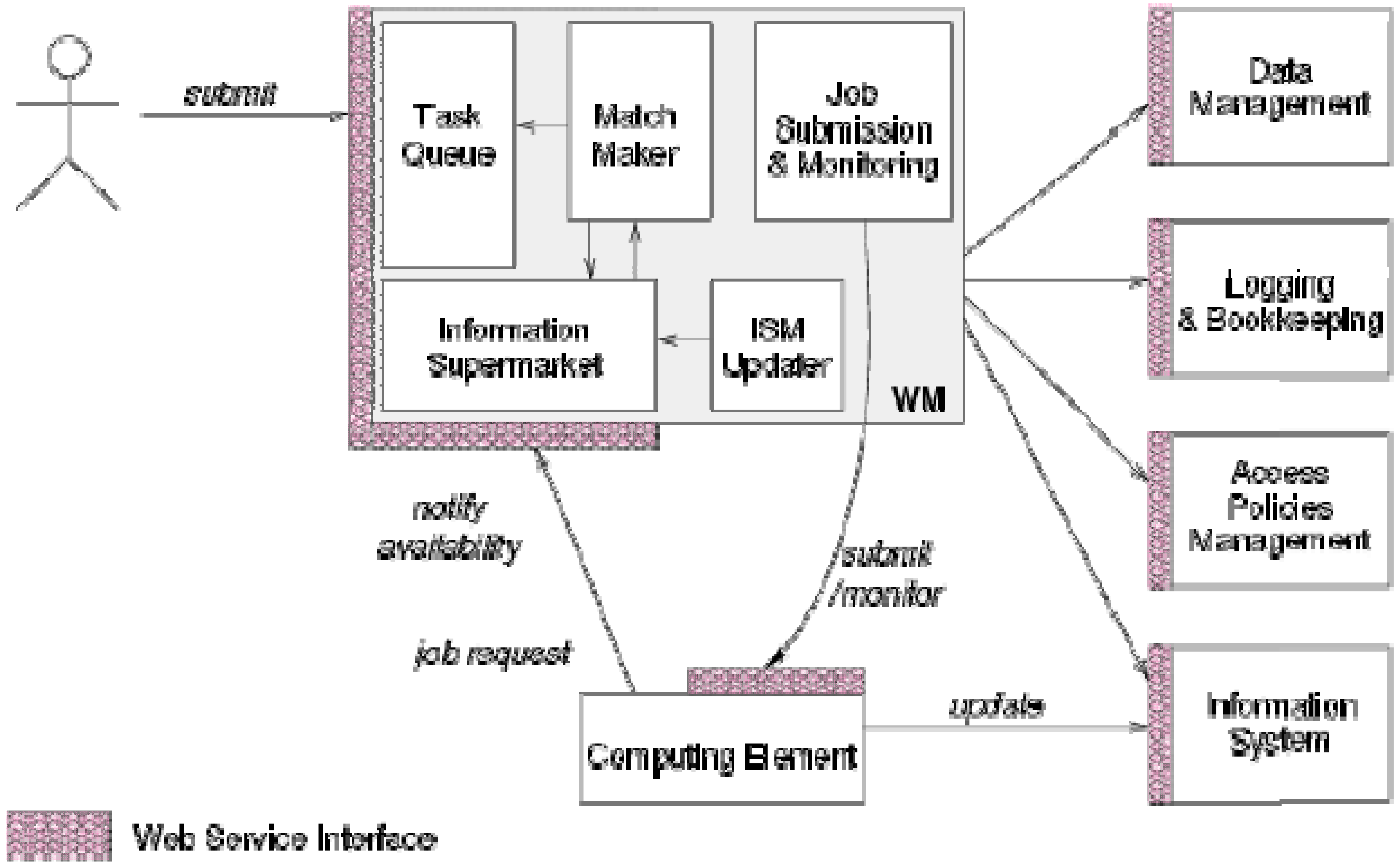
Overview of gLite Middleware

- **Job Management Services**
  - **main services related to job management/execution are**
    - **computing element**
      - ***job management (job submission, job control, etc.)***
      - ***provision of information about its characteristics and status***
    - **workload management**
      - ***core component discuss in details in the following***
    - **accounting**
      - ***special case as it will eventually take into account***
        - **computing, storage and network resources**
    - **job provenance (not yet implemented in gLite)**
      - ***keep track of the definition of submitted jobs, execution conditions and environment, and important points of the job life cycle for a long period***
        - **debugging, post-mortem analysis, comparison of job execution**
    - **package manager (not yet implemented in gLite)**
      - ***automates the process of installing, upgrading, configuring, and removing software packages from a shared area on a grid site.***
        - **extension of a traditional package management system to a Grid**

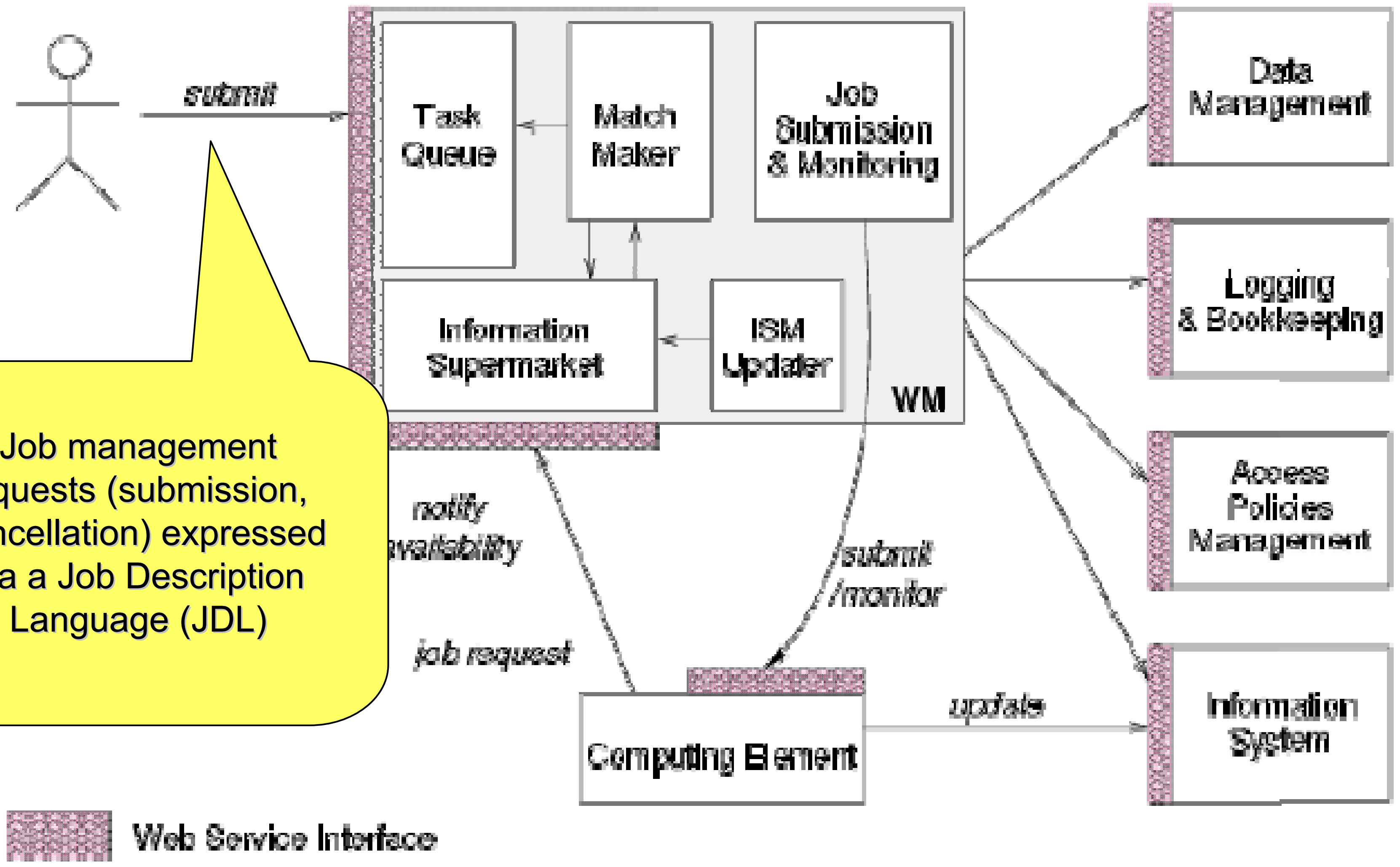
WMS Architecture Overview



WMS's Architecture

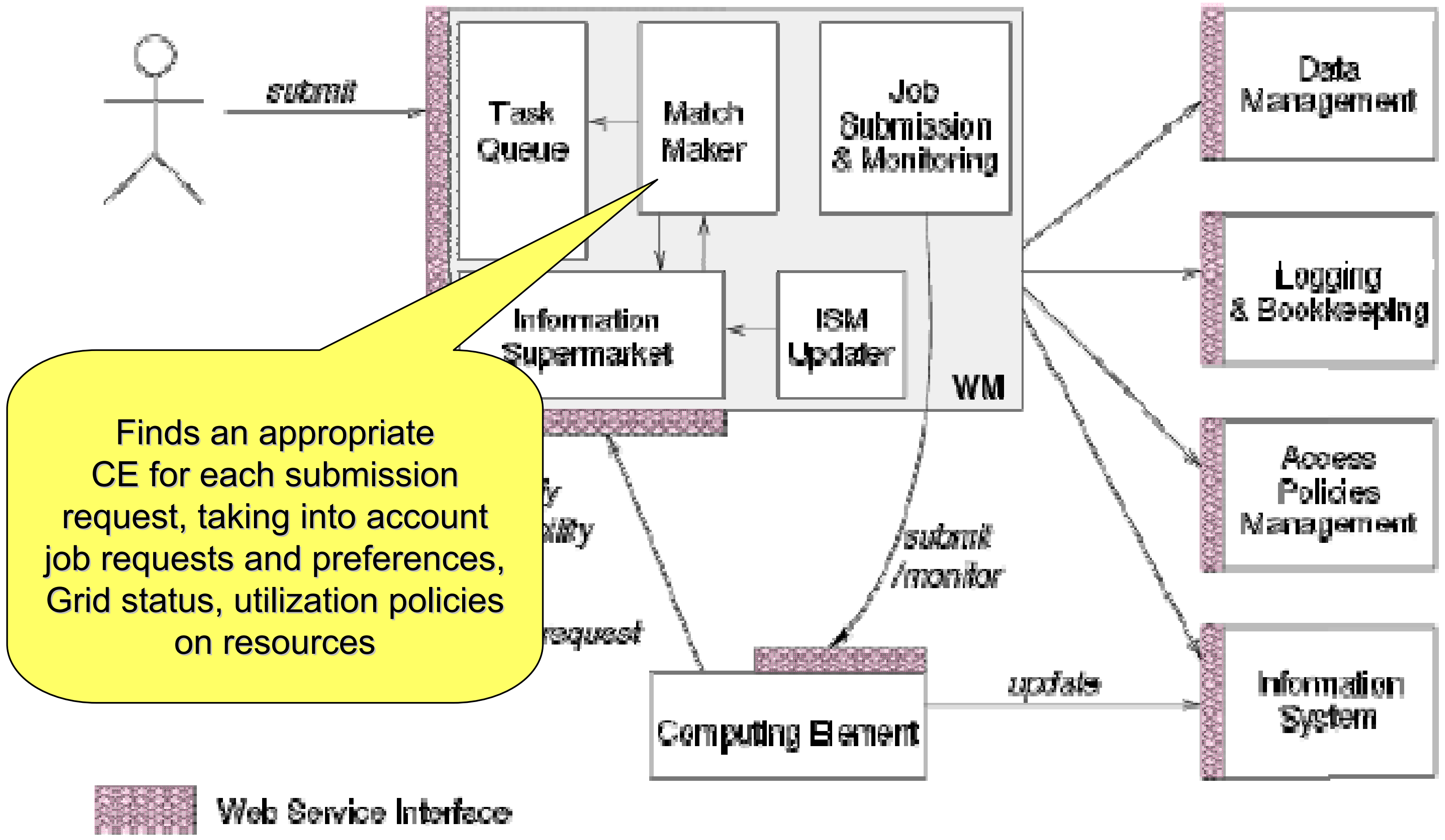


WMS's Architecture



Job management requests (submission, cancellation) expressed via a Job Description Language (JDL)

WMS's Architecture

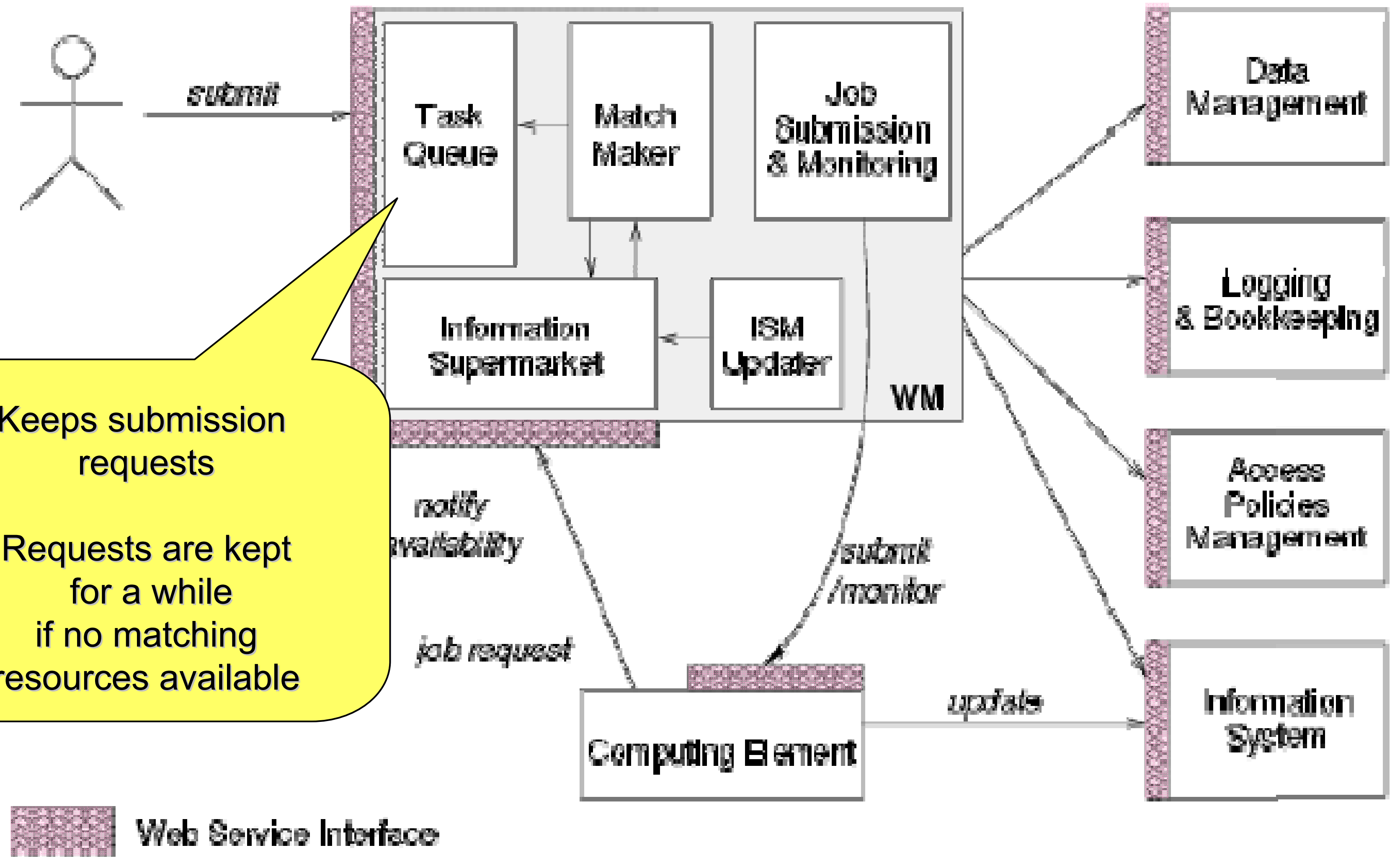


Finds an appropriate CE for each submission request, taking into account job requests and preferences, Grid status, utilization policies on resources

WMS's Scheduling Policies

- WM can adopt
  - **eager scheduling ("push" model)**
    - **a job is bound to a resource as soon as possible and, once the decision has been taken, the job is passed to the selected resource for execution**
  - **lazy scheduling ("pull" model)**
    - **foresees that the job is held by the WM until a resource becomes available, at which point that resource is matched against the submitted jobs**
      - ***the job that fits best is passed to the resource for immediate execution.***

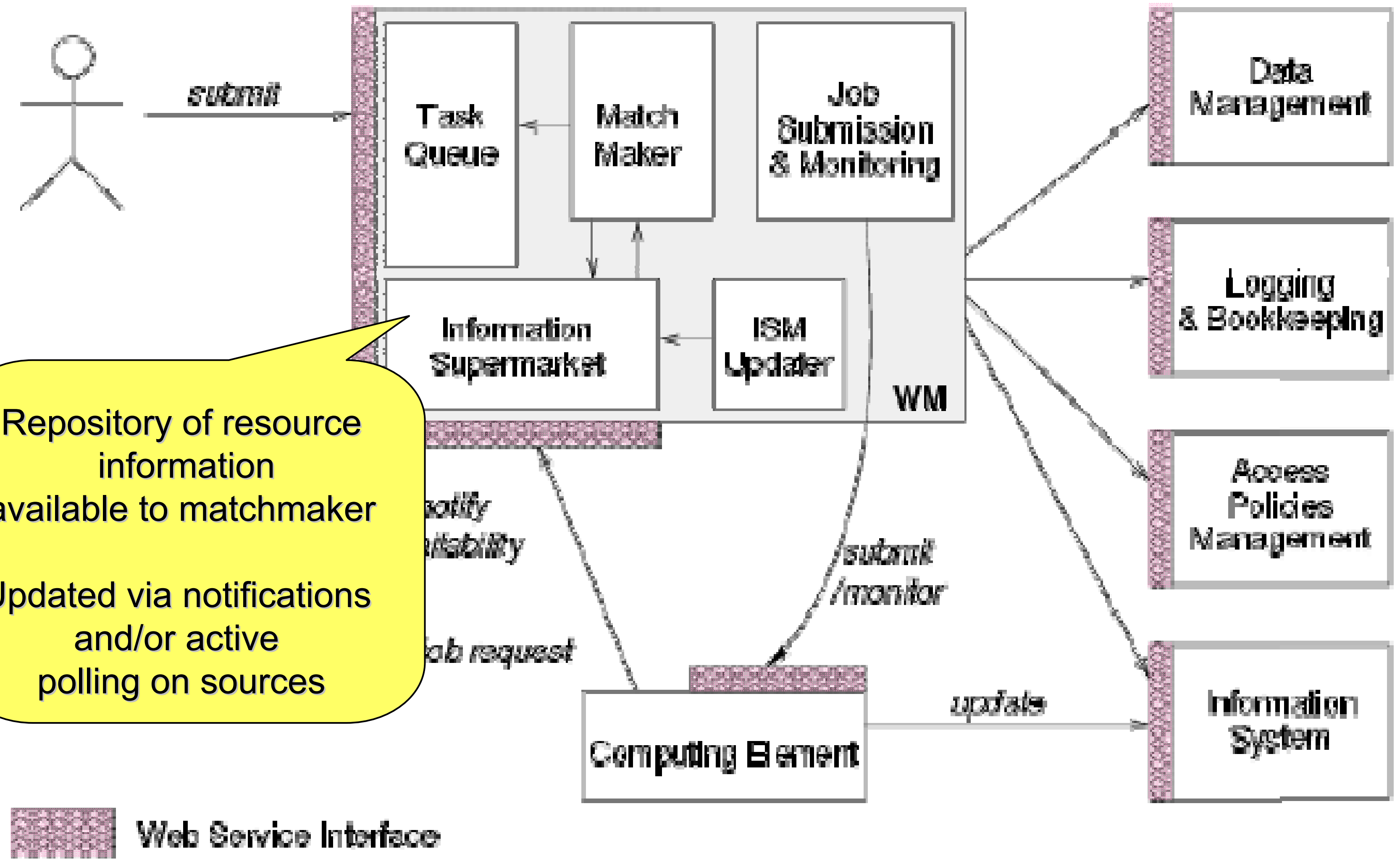
WMS's Architecture



Keeps submission requests

Requests are kept for a while if no matching resources available

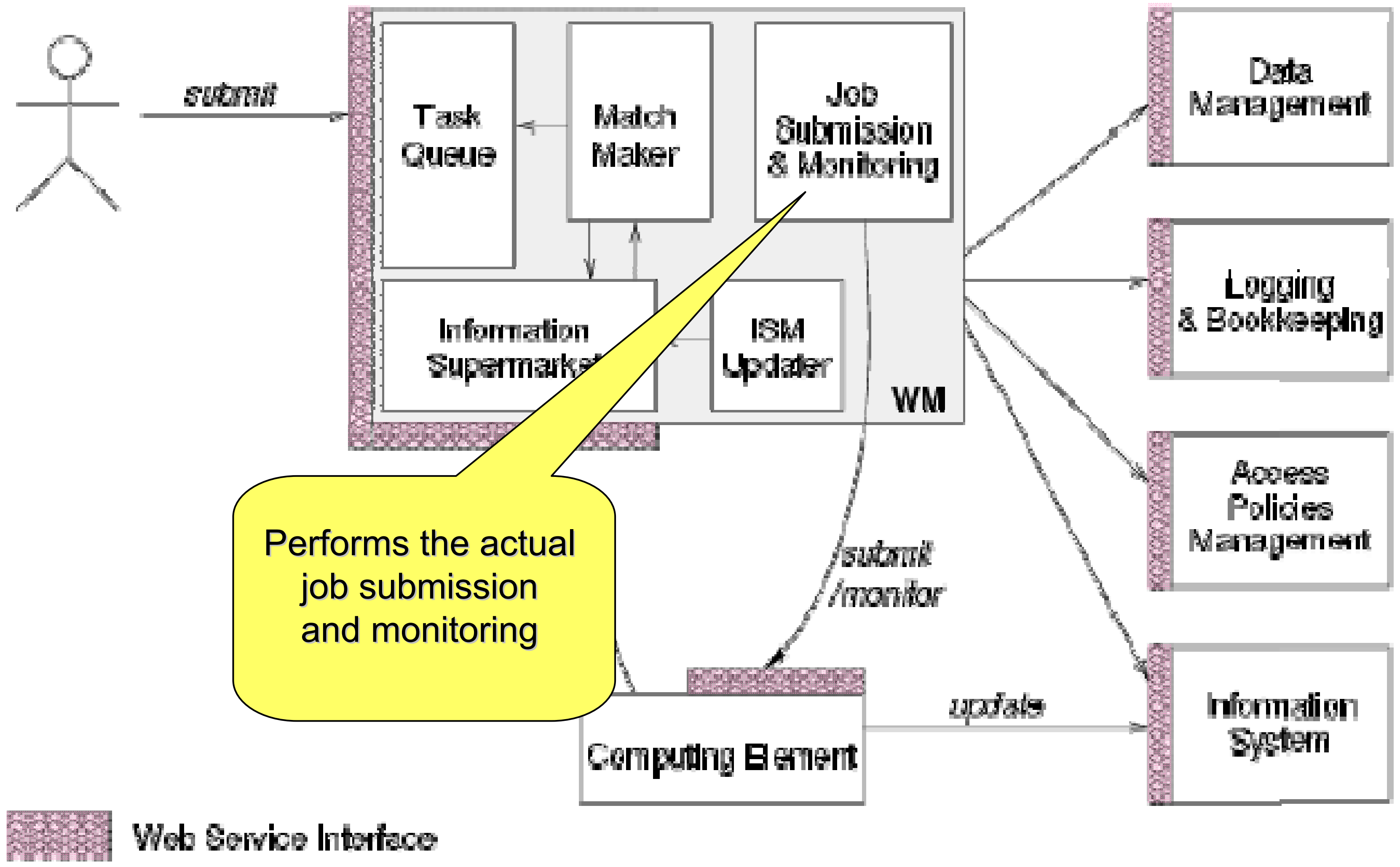
WMS's Architecture



Repository of resource information available to matchmaker

Updated via notifications and/or active polling on sources

WMS's Architecture



Performs the actual job submission and monitoring

The Information Supermarket

- ISM represents one of the most notable improvements in the WM as inherited from the EU DataGrid (EDG) project
  - **decoupling between the collection of information concerning resources and its use**
    - **allows flexible application of different policies**
- The ISM basically consists of a repository of resource information that is available in read only mode to the matchmaking engine
  - **the update is the result of**
    - **the arrival of notifications**
    - **active polling of resources**
    - **some arbitrary combination of both**
  - **can be configured so that certain notifications can trigger the matchmaking engine**
    - **improve the modularity of the software**
    - **support the implementation of lazy scheduling policies**

The Task Queue

- The Task Queue represents the second most notable improvement in the WM internal design
  - **possibility to keep a submission request for a while if no resources are immediately available that match the job requirements**
    - **technique used by the AliEn and Condor systems**

- Non-matching requests
  - **will be retried either periodically**
    - **eager scheduling approach**
  - **or as soon as notifications of available resources appear in the ISM**
    - **lazy scheduling approach**

Job Submission Services

WMS components handling the job during its lifetime and performing the submission

- Job Adapter
  - is responsible for
    - making the final touches to the JDL expression for a job, before it is passed to CondorC for the actual submission
    - creating the job wrapper script that creates the appropriate execution environment in the CE worker node
      - *transfer of the input and of the output sandboxes*
- CondorC
  - responsible for
    - performing the actual job management operations
      - *job submission, job removal*
- DAGMan
  - meta-scheduler
    - purpose is to navigate the graph
    - determine which nodes are free of dependencies
    - follow the execution of the corresponding jobs.
  - instance is spawned by CondorC for each handled DAG
- Log Monitor
  - is responsible for
    - watching the CondorC log file
    - intercepting interesting events concerning active jobs
      - *events affecting the job state machine*
    - triggering appropriate actions.

Job Logging & Bookkeeping

- L&B tracks jobs in terms of *events*
  - **important points of job life**
    - **submission, finding a matching CE, starting execution etc**
      - ***gathered from various WMS components***
- The events are passed to a physically close component of the L&B infrastructure
  - ***locallogger***
    - **avoid network problems**
      - ***stores them in a local disk file and takes over the responsibility to deliver them further***
- The destination of an event is one of *bookkeeping servers*
  - **assigned statically to a job upon its submission**
    - **processes the incoming events to give a higher level view on the job states**
      - **Submitted, Running, Done**
    - **various recorded attributes**
      - ***JDL, destination CE name, job exit code***
- Retrieval of both job states and raw events is available via legacy (EDG) and WS querying interfaces
  - **user may also register for receiving notifications on particular job state changes**

Job Description Language

Job Description Language

- In gLite **Job Description Language (JDL)** is used to describe jobs for execution on Grid.

- The JDL adopted within the gLite middleware is based upon Condor's **CLASSified Advertisement language (ClassAd)**.
  - A ClassAd is a record-like structure composed of a finite number of attribute separated by semi-colon (;)
  - A ClassAd is highly flexible and can be used to represent arbitrary services

*The JDL is used in gLite to specify the job's characteristics and constrains, which are used during the **match-making process** to select the best resources that satisfy job's requirements.*

Job Description Language (cont.)

- The JDL syntax consists on statements like:

  Attribute = value;

- Comments must be preceded by a sharp character (**#**) or have to follow the C++ syntax

WARNING: The JDL is sensitive to blank characters and tabs. No blank characters or tabs should follow the semicolon at the end of a line.

Job Description Language (cont.)

- In a JDL, some attributes are mandatory while others are optional.
- An “essential” JDL is the following:

```
Executable = “test.sh”;
StdOutput = “std.out”;
StdError = “std.err”;
InputSandbox = {“test.sh”};
OutputSandbox = {“std.out”,”std.err”};
```

- If needed, arguments to the executable can be passed:

```
Arguments = “Hello World!”;
```

Job Description Language (cont.)

- If the arguments contains quoted strings, the quotes must be escaped with a backslash

  e.g. Arguments = “\”Hello World!\“ 10”;

- Special characters such as &, |, >, < are only allowed if specified inside a quoted string or preceded by triple \

  (e.g. Arguments = "-f file1\\\&file2";)

Job Description Language (cont.)

- The supported attributes are grouped in two categories:
  - **Job Attributes**
    - Define the job itself
  - **Resources**
    - Taken into account by the RB for carrying out the matchmaking algorithm (to choose the “best” resource where to submit the job)
    - *Computing Resource*
      - *Used to build expressions of Requirements and/or Rank attributes by the user*

```
Requirements=other.GlueCEUniqueID ==
   “adc006.cern.ch:2119/jobmanager-pbs-infinite”


Requirements=Member(“ALICE-3.07.01”,
   other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Job Description Language (cont.)

- *Data and Storage resources*
  - *Input data to process, SE where to store output data, protocols spoken by application when accessing Ses*

```
InputData = {"lfn:cmstestfile",
        "guid:135b7b23-4a6a-11d7-87e7-
9d101f8c8b70"};
```

JDL: Relevant Attributes

- **JobType** (optional)
  - ***Normal*** (simple, sequential job), ***Interactive***, ***MPICH***, ***Checkpointable***, ***Partitionable***
  - **Or combination of them**
    - Checkpointable, Interactive
    - Checkpointable, MPI

E.g. JobType = “Interactive”;

JobType = {“Interactive”,”Checkpointable”};

“Interactive” + “MPI” not yet permitted

JDL: Relevant Attributes (cont.)

- **Type** (mandatory, default “Job”)
  - This is a representing the type of the request described by the JDL.
  - Possible values are:
    - Job
    - DAG
    - Reservation
    - Co-allocation

E.g.: **Type = “Job”;**

JDL: Relevant Attributes (cont.)

- **Executable** (mandatory)
  - This is a string representing the executable/command name.
  - The user can specify an executable which is already on the remote CE
  - **Executable = {“/opt/EGEODE/GCT/egeode.sh”};**
  - The user can provide a local executable name, which will be staged from the UI to the WN.

Executable = {“egeode.sh”};

InputSandbox = {“/home/larocca/egeode/ egeode.sh”};

JDL: Relevant Attributes (cont.)

- **Arguments** (optional)

  This is a string containing all the job command line arguments.

  E.g.: If your executable sum has to be started as:

  ```
  $ sum N1 N2 –out result.out
  ```

  ```
  Executable = "sum";
  Arguments = "N1 N2 –out result.out";
  ```

JDL: Relevant Attributes (cont.)

- **Environment** (optional)
  - List of environment settings needed by the job to run properly

    E.g. **Environment = {“JAVABIN=/usr/local/java”};**

- **InputSandbox** (optional)
  - List of files on the UI local disk needed by the job for running
  - The listed files will automatically staged to the remote resource

  E.g. **InputSandbox ={“myscript.sh”,”/tmp/cc,sh”};**

JDL: Relevant Attributes (cont.)

- **OutputSandbox** (optional)
  - List of files, generated by the job, which have to be retrieved

  E.g. **OutputSandbox ={ “std.out”,”std.err”, “image.png”};**

- **VirtualOrganisation** (optional)
  - This is a string representing the name of the VO the submitting user is currently working for.

  E.g. **VirtualOrganisation =“gilda”;**

JDL: Relevant Attributes (cont.)

- **Requirements** (optional)
  - Job requirements on computing resources
  - Specified using attributes of resources published in the Information Service
  - If not specified, default value defined in UI configuration file is considered

    Default. **Requirements =**
    other.GlueCEStateStatus == "Production“;

Requirements=other.GlueCEUniqueID ==
“adc006.cern.ch:2119/jobmanager-pbs-infinite”

Requirements=Member(“ALICE-3.07.01”,
other.GlueHostApplicationSoftwareRunTimeEnvironment);

JDL: Relevant Attributes (cont.)

- **Rank** (optional)
  - Floating-point expression used to ranks CEs that have already met the *Requirements* expression.
  - The Rank expression can contain attributes that describe the CE in the Information System (IS).
  - The evaluation of the rank expression is performed by the Resource Broker (RB) during the match-making phase.
  - A higher numeric value equals a better rank.
  - If not specified, default value defined in the UI configuration file is considered

  Default: **Rank = - *other.GlueCEStateFreeCPUs;***

JDL: Relevant Attributes (cont.)

- **InputData** (optional)
  - This is a string or a list of strings representing the *Logical File Name (LFN)* or *Grid Unique Identifier (GUID)* needed by the job as input.
  - The list is used by the RB to find the CE from which the specified files can be better accessed and schedules the job to run there.

InputData = {“lfn:cmstestfile”,

“guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70”};

JDL: Relevant Attributes (cont.)

- DataAccessProtocol (mandatory if InputData has been specified)
  - The protocol or the list of protocols which the application is able to “speak” with for accessing files listed in *InputData* on a given SE.

- Supported protocols in gLite are currently gsiftp, and file.

```
DataAccessProtocol = {“file”,“gsiftp”};
```

JDL: Relevant Attributes (cont.)

- **StorageIndex** (mandatory if InputData or OutputData have been specified)
  - Representing the URL of the StorageIndex Service to contact for resolving the file names specified in the InputData attributes.

```
StorageIndex = “https://glite.org:9443/StorageIndex”;
```

JDL: Relevant Attributes (cont.)

- **OutputSE** (optional)
  - This string representing the URI of the **Storage Element (SE)** where the user wants to store the output data.
  - This attribute is used by the Resource Broker to find the bestCE "*close*" to this SE and schedule the job there.

OutputSE = "grid009.ct.infn.it";

JDL: Relevant Attributes (cont.)

- **OutputData** (optional)
  - **This attribute allows the user to ask for the automatic upload and registration of datasets produced by the job on the Worker Node (WN).**
  - **This attribute contains the following three attributes:**
    - ***OutputFile***
    - ***StorageElement***
    - ***LogicalFileName***

JDL: Relevant Attributes (cont.)

- **OutputFile** (mandatory if OutputData has been specified)
  - This is a string attribute representing the name of the output file, generated by the job on the WN, which has to be automatically uploaded and registered by the WMS.
- **StorageElement** (optional)
  - This is a string representing the URI of the Storage Element where the output file specified in the OutputFile has to be uploaded by the WMS.
- **LogicalFileName** (optional)
  - This is a string representing the LFN user wants to associate to the output file when registering it to the Catalogue.

JDL: Relevant Attributes (cont.)

- **NodeNumber** (mandatory if JobType=MPICH)
  - NodeNumber attribute is an integer specifying the number of nodes needed for a MPI job.
  - The RB uses this attribute during the matchmaking for selecting those CE having a number of CPUs equals or greater the one specified in NodeNumber.

```
NodeNumber = 5;
```

JDL: Relevant Attributes (cont.)

- **JobSteps** (mandatory for checkpointable or partitionable jobs)
  - JobSteps attribute can be either an integer representing the number of steps for a checkpointable or partitionable job e.g.:

```
JobSteps = 100000;
```

  - or a list of strings representing labels associated to the steps of a checkpointable or partitionable job e.g.:

```
JobSteps =  {“d0”, “d1”, ”gmos”};
```

JDL: Relevant Attributes (cont.)

- **CurrentStep** (mandatory for checkpointable or partitionable jobs)
  - CurrentStep attribute indicating the initial step when submitting a checkpointable or partitionable job.

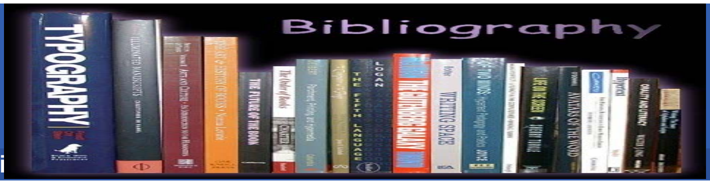
```
CurrentStep = 2;
```

JDL: Relevant Attributes (cont.)

- **max_nodes_running**
  - Representing the maximun number of nodes of a DAG that can be submitted by DGAMan

```
max_nodes_running = 25;
```

- **dependencies**
  - A list of dependencies between the nodes of the DAG.



References

- **JDL Attributes**

  http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf

- **LCG-2 User Guide Manual Series**

  https://edms.cern.ch/file/454439/LCG-2-UserGuide.html

DGAS
(Data Grid Accounting System)

Grid Accounting - Overview

A generic Grid accounting process involves many phases that can be divided in:

- **Metering:** collection of usage metrics on computational resources.
- **Accounting:** storage of such metrics for further analysis.
- **Usage Analysis:** Production of reports from the available records.
- **Pricing:** Assign and manage prices for computational resources.
- **Billing:** Assign a cost to each user for his operations on the Grid .

In this presentation we briefly describe these steps and give a quick overview of DGAS, the accounting system included in the gLite middleware.

Metering

- The metering phase in Grid accounting is probably the most important of the whole process.
- During this phase the user payload on a resource needs to be correctly measured, and assigned to the Grid User.
- This requires the system collects information from the operating system (or the LRMS for batch jobs) and from the grid middleware. This information forms the **Usage Record** for the user process.
- This usage record must include at least the *Grid Unique Identifier* for the Grid User, the CEid as well as the JobID.

Metering (cont.)

- *Usage Metering* on Computing Elements is usually done by lightweight sensors installed on them.
- These sensors parse the LRMS event logs to build *Usage Records* that can be passed to the accounting layer.

Accounting

- Once collected, usage records need to be properly archived in databases for further analysis.

- These information should be available to the Users responsible for the payload, to the Site Managers of the Grid Resources and to the VO administrator of the user, but not to other people. In other words, information must be confidential.

- Usage records must be sent encrypted and signed by the Accounting Services.

Usage Analysis

- Information stored in the Accounting databases are rather complex. Not all the ‘users’ are interested in all of them. So there is the necessity for a system to analyses them and produce ***reports***.
- Different *types of users* are interested in different views of the usage records, for example:

- A user will simply want to know how (s)he used the grid resources.
- A site manager needs to know who used his resources.
- A VO manager needs to trace what the VO users are doing on the Grid.

Pricing & Billing

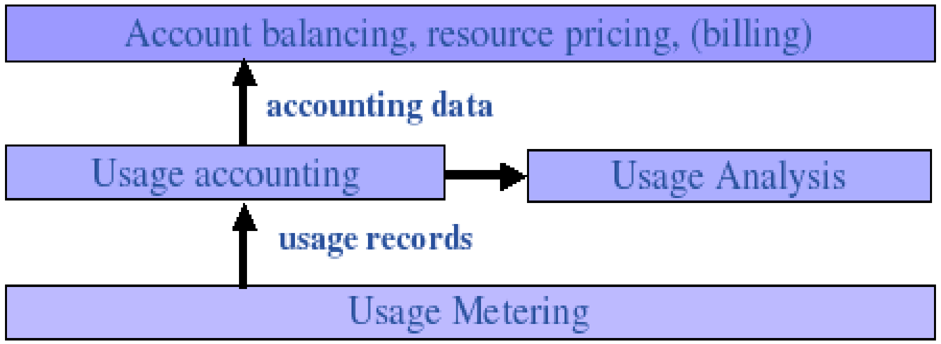
Resources' owners may want to charge users for their use, so it is necessary to establish a cost for the service provided to the user.

"A cost is usually computed according to a price assigned to the unit of usage of a computing resource and to the usage measured for the same resource."

Thus a service responsible for managing the resource prices and communicating them to all the partners is needed.

DGAS

- The *Data Grid Accounting System* was originally developed within the EU Datagrid Project and is now being maintained and re-engineered within the EU EGEE Project.
- The Purpose of *DGAS* is to implement *Resource Usage Metering*, *Accounting* and *Account Balancing* (through *resource pricing*) in a fully distributed Grid environment. It is conceived to be distributed, secure and extensible.
- DGAS system can be described using a three-layer model as shown in figure



DGAS Server Side

- The Server Side contains :
  - Price Authority (PA)
    - Provides the features necessary for Economic Accounting.
  - Home Location Register (HLR)
    - Responsible for keeping the accounting information.
  - High Availability Daemon (HAD)
    - Responsible for monitoring the status of the service.
    - In case of failure it restarts the daemon avoiding long down periods due to service failures.

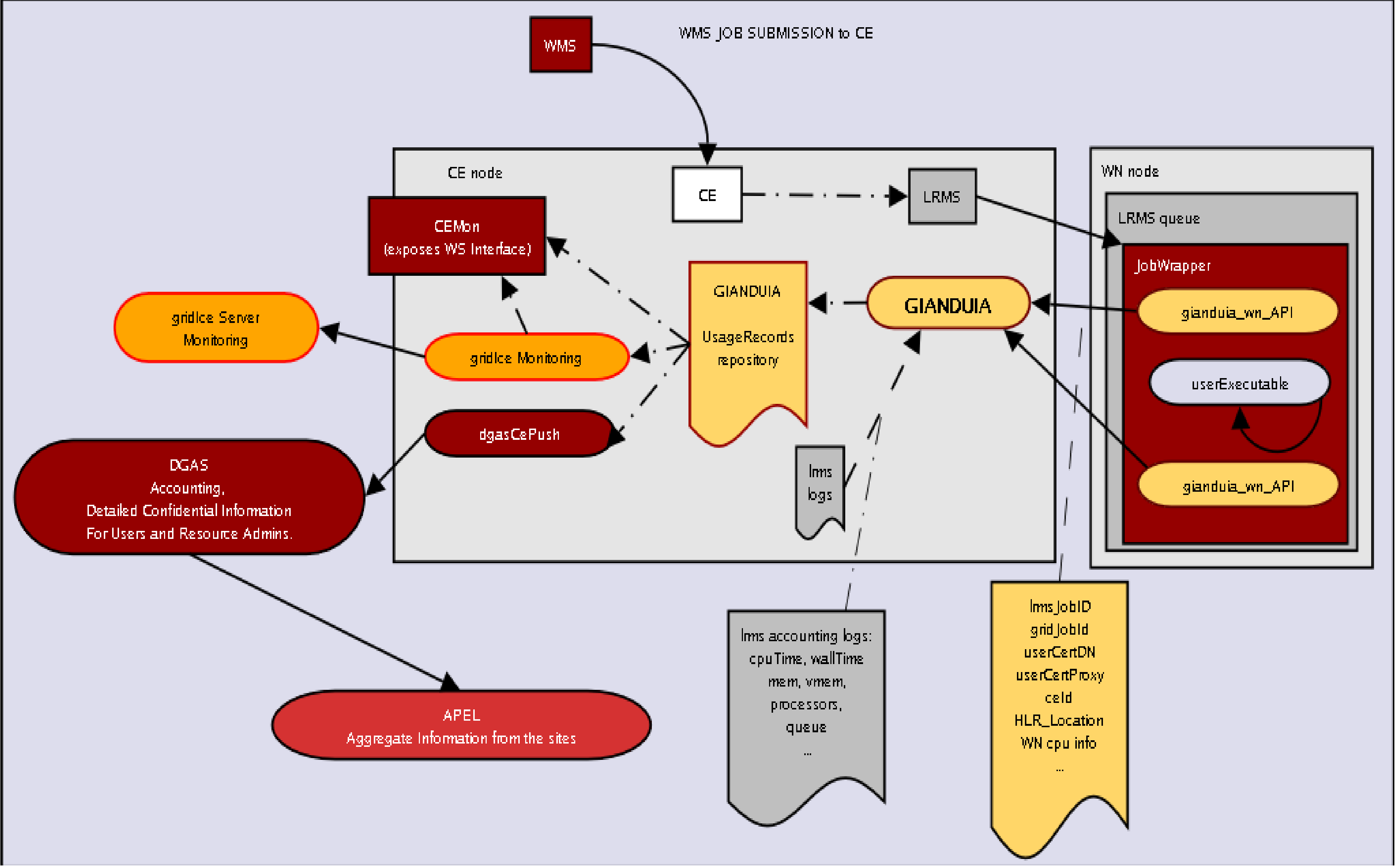
DGAS Client Side

- The Client Side contains :
  - Gianduia
    - It is installed on a Computing Element.
    - Collect the usage records of the job.
    - The files created by Gianduia are treated in a queue.
    - When job's usage record is correctly sent to the HLR, the corresponding file is removed from the queue.
    - If job's usage record can't be transmitted, the process will be retrieved for a tunable amount of times.

DGAS Client Side (cont.)

- **CEPushD**
  - Send the files created by Gianduia to the HLR.
- **ceServed**
  - Collect the information transmitteed from the WN.
- **HAD**
  - Monitor the status of the service and restart it in case it dies.

DGAS Metering: Gianduia



DGAS Accounting

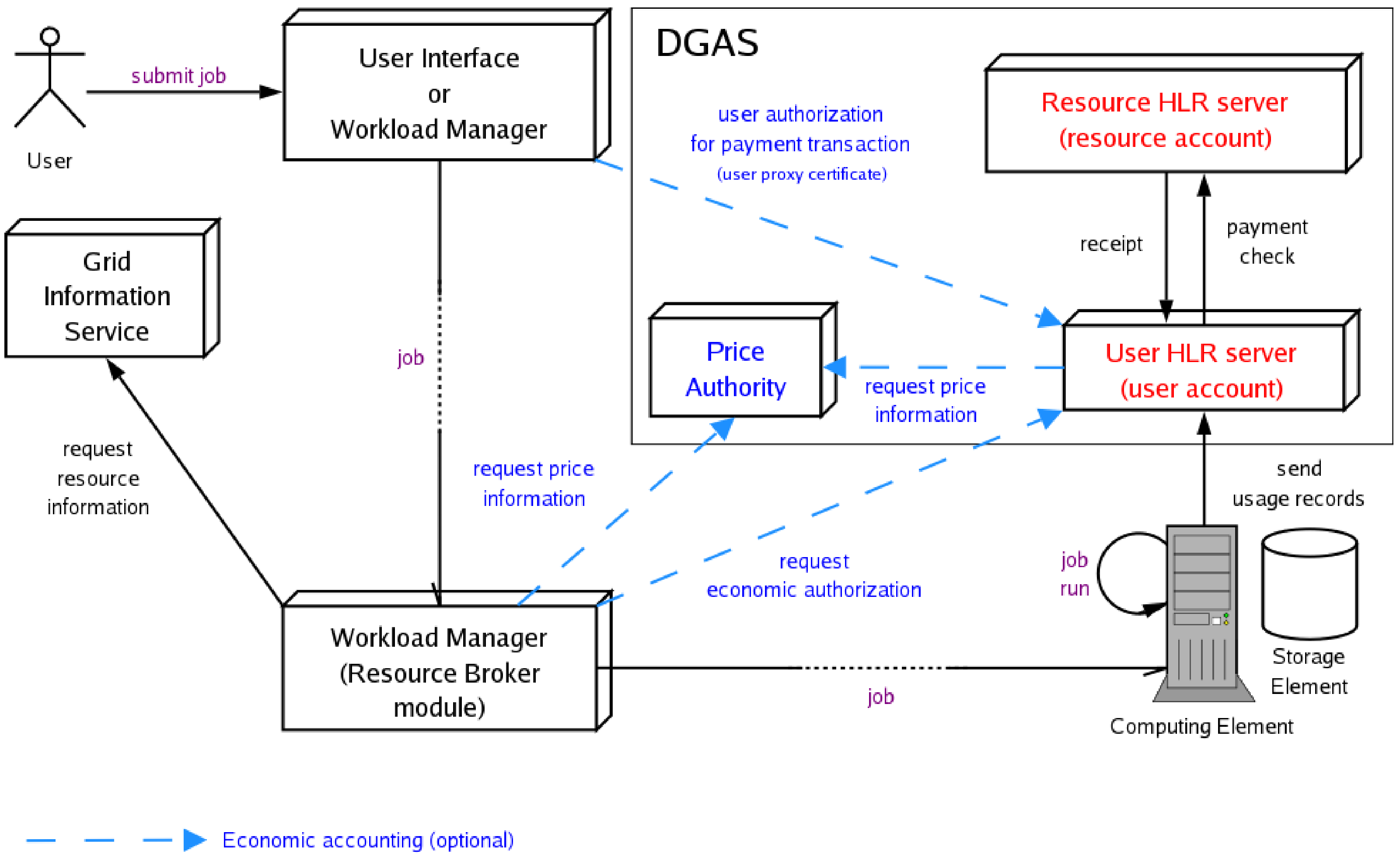
- The Home Location Register (HLR) service is the part of DGAS that is responsible for keeping the accounting information (usage records) for both grid resources and users.

- The usage records are used to define a job's cost which can be debited to the user.

- In order to achieve scalability, accounting records can be stored on an arbitrary number of independent HLRs. At least one HLR per VO is foreseen, although a finer granularity is possible.

DGAS accounting architecture

A simplified view of DGAS within the WMS context.



DGAS: Pricing & Billing

- **Price Authority (PA)** is a key component of the DGAS toolkit because it provides the features necessary for the Economic Accounting.
- The PA Server is an entity that assigns the prices to the resources.
- The prices, that are kept in a historic price database, can be assigned manually or using different dynamic pricing algorithms.

References

- ***Further information and documentation about DGAS can be found at:***

  http://www.to.infn.it/grid/accounting/main.html