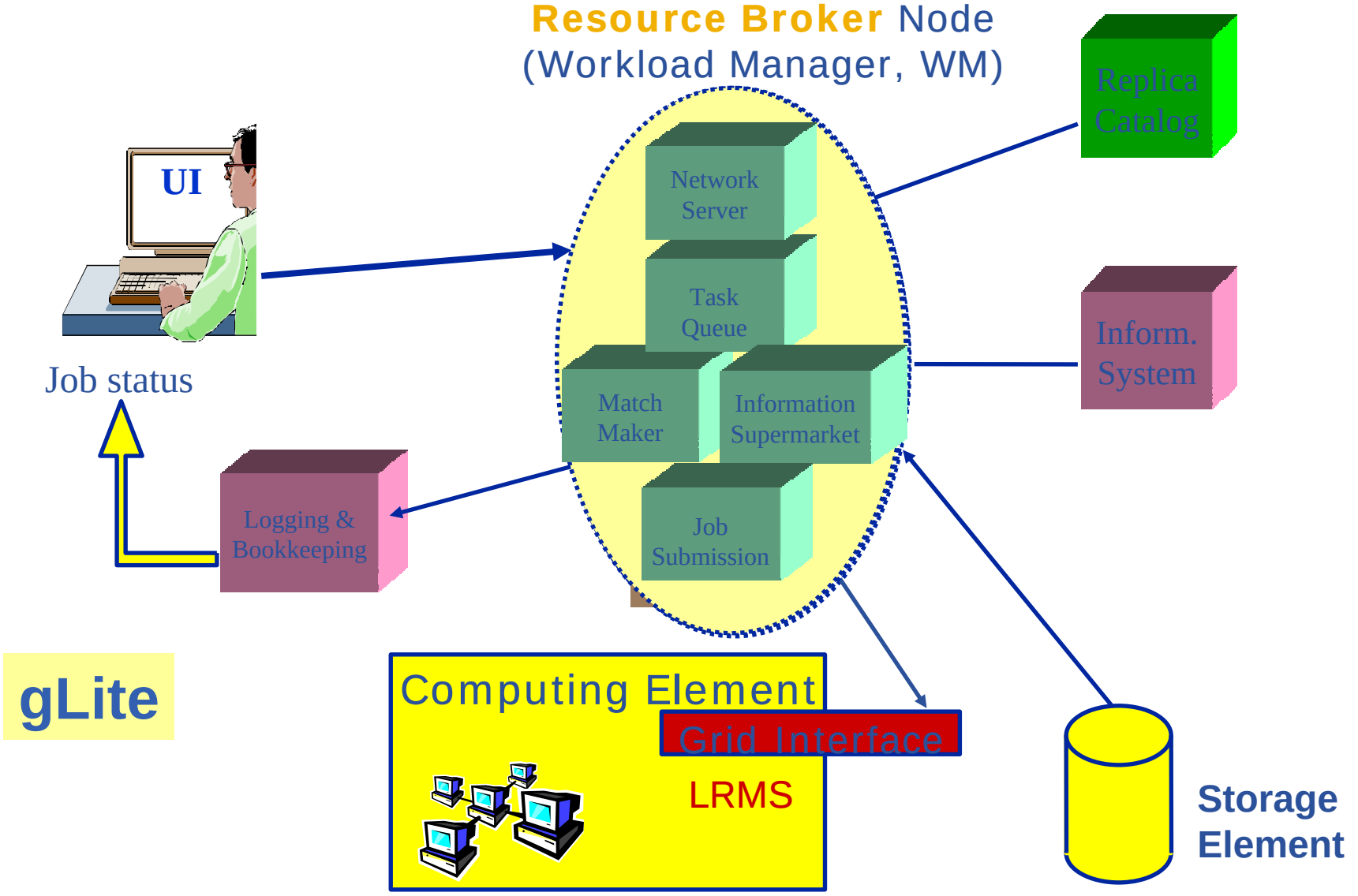
Architecture of the gLite Workload Management System

Giuseppe Andronico
INFN
EGEE Tutorial
Taipei, 22-23.08.2005

Overview of gLite WMS

- **Job Management Services**
  - **main services related to job management/execution are**
    - **computing element**
      - ***job management (job submission, job control, etc.), but it must also provide***
      - ***provision of information about its characteristics and status***
    - **workload management**
      - ***core component discussed in details***
    - **accounting**
      - ***special case as it will eventually take into account***
        - **computing, storage and network resources**
    - **job provenance**
      - ***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**
      - ***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**

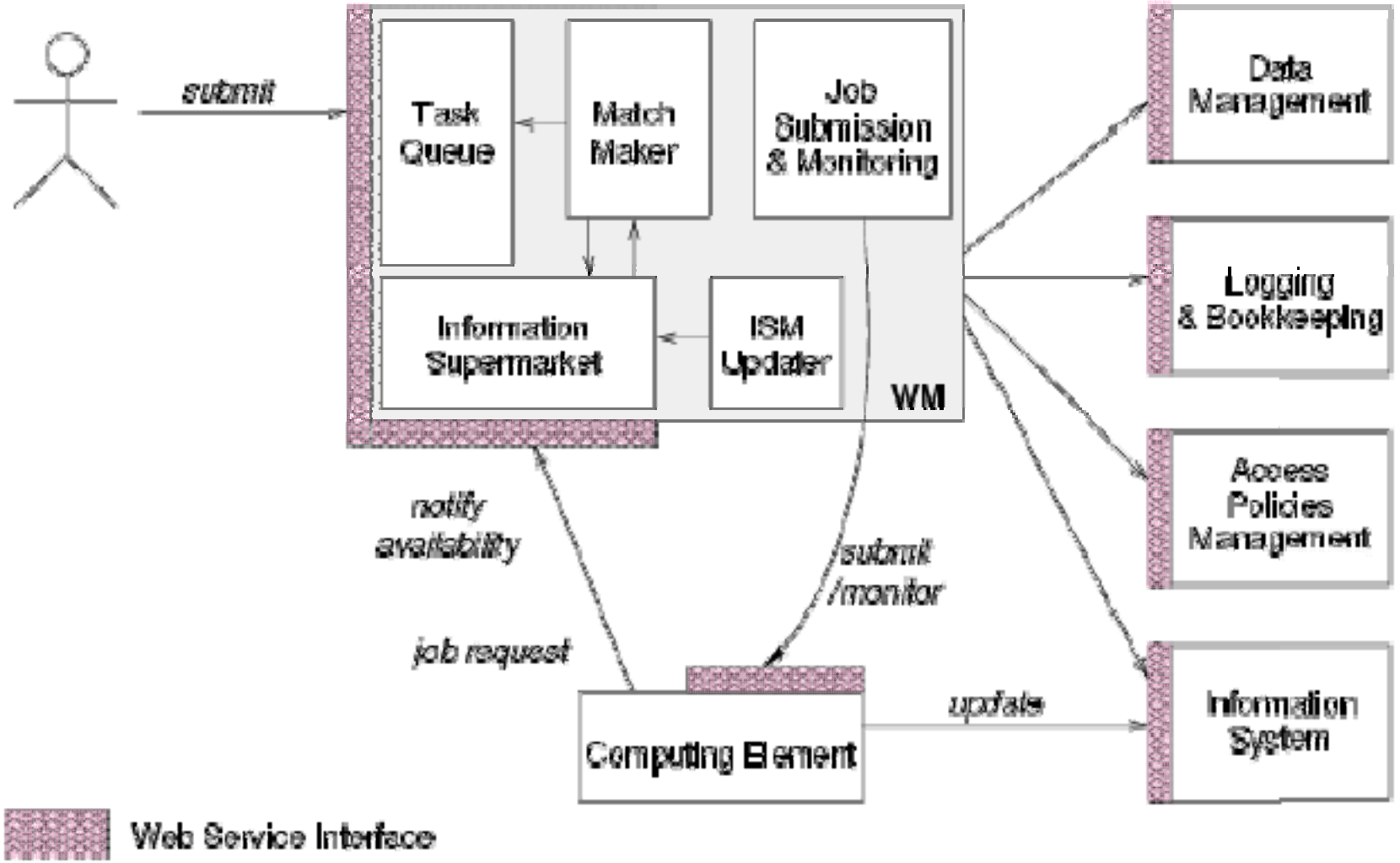
Architecture Overview



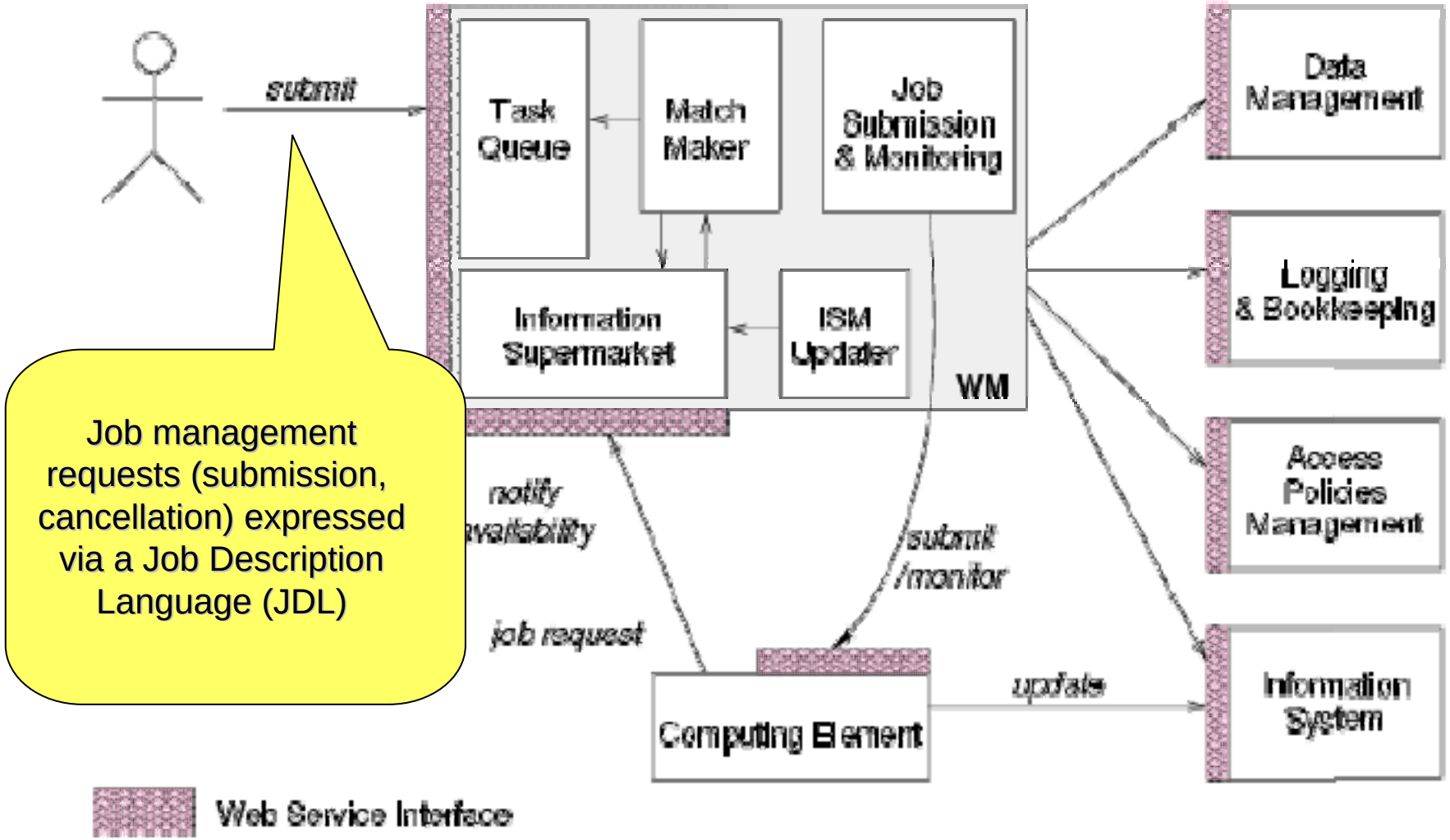
WMS's Scheduling Policies

- 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.***
- Varying degrees of eagerness (or laziness) are applicable
  - **match-making level**
    - **eager scheduling**
      - ***implies matching a job against multiple resources***
    - **lazy scheduling**
      - ***implies matching a resource against multiple jobs***

WMS's Architecture

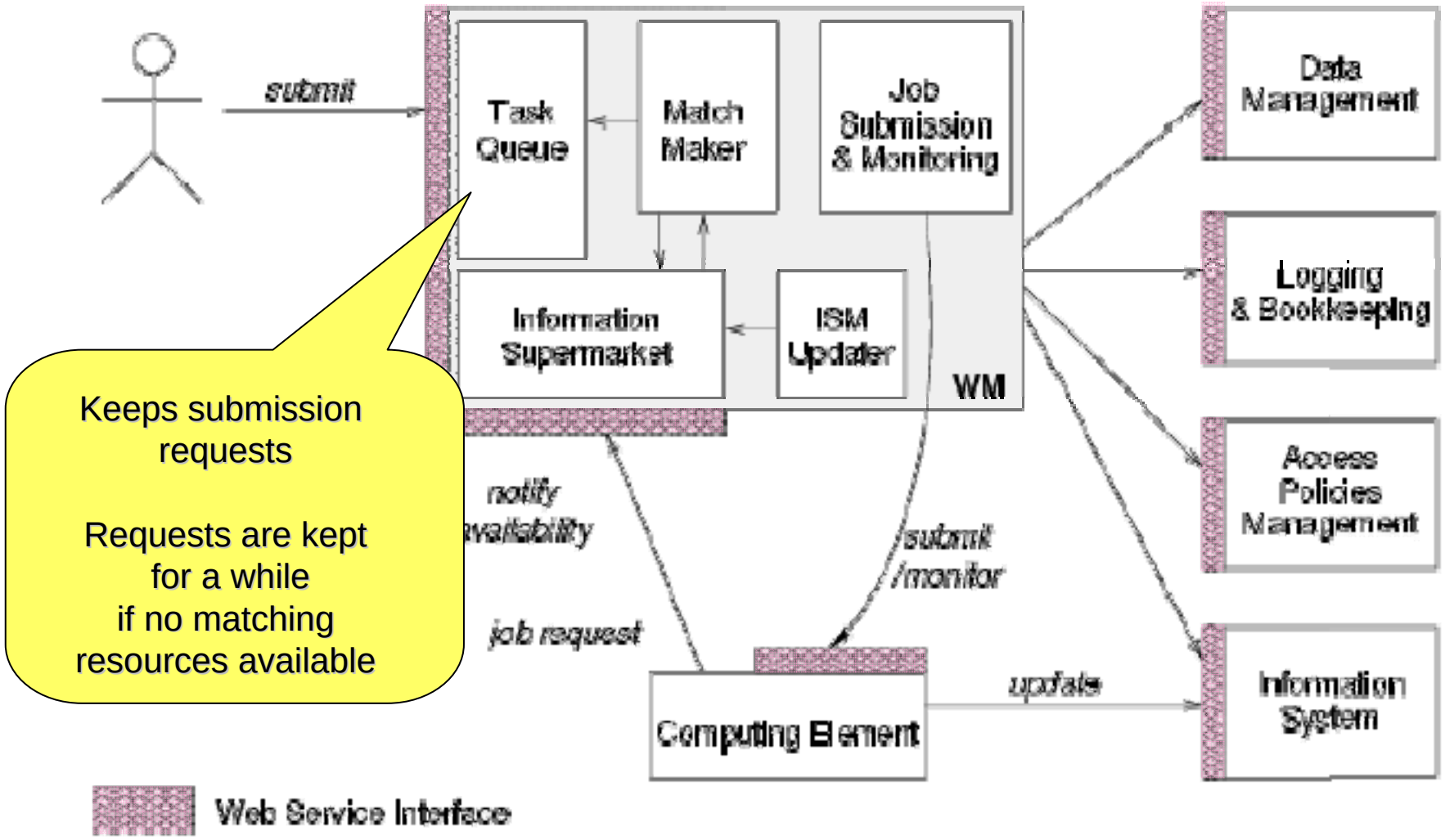


WMS's Architecture



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

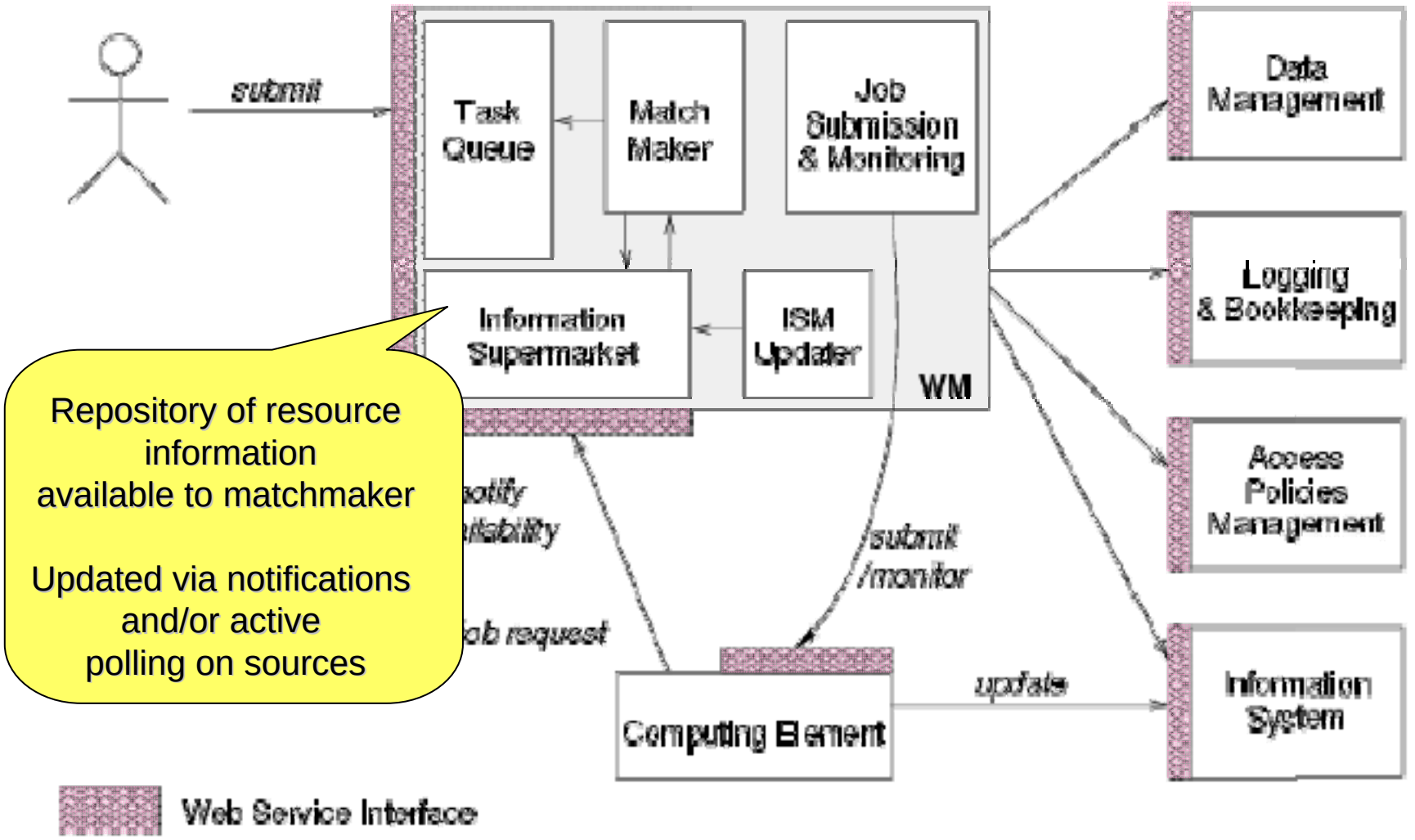
WMS's Architecture



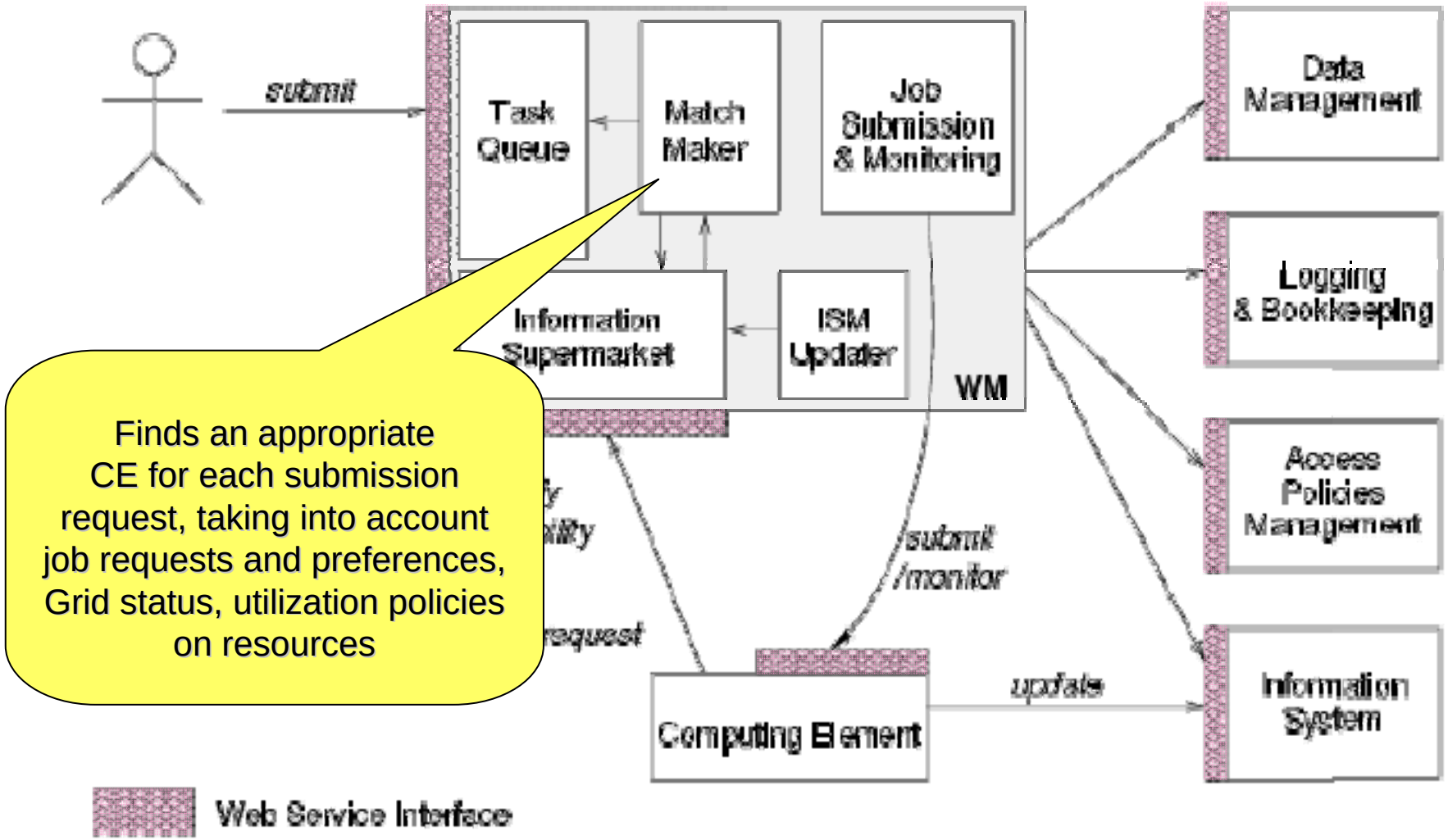
Keeps submission requests

Requests are kept for a while if no matching resources available

WMS's Architecture

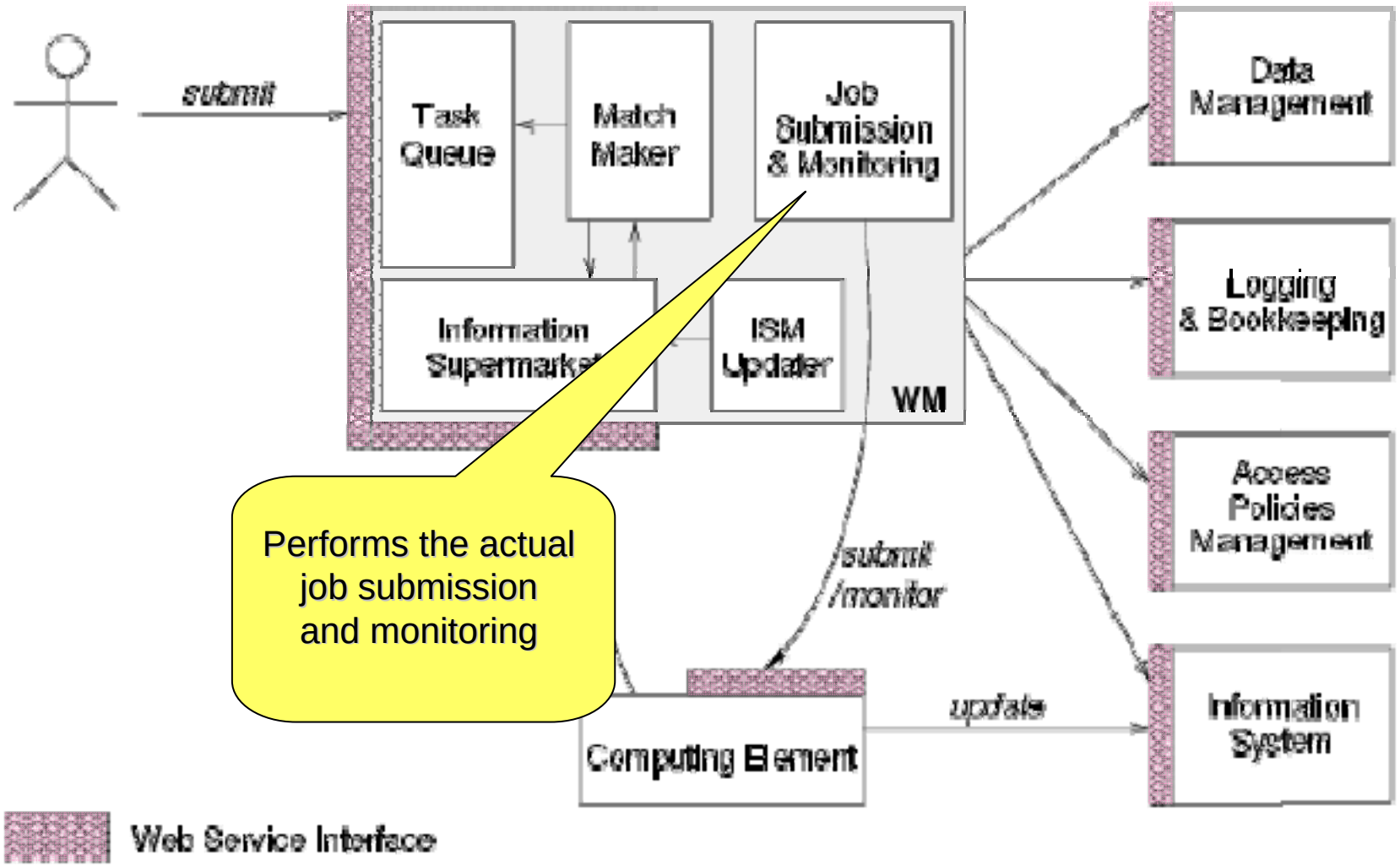


WMS's Architecture



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

WMS's Architecture



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 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 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 Preparation

- **Information to be specified when a job has to be submitted:**
  - Job characteristics
  - Job requirements and preferences on the computing resources
    - Also including software dependencies
  - Job data requirements
- **Information specified using a Job Description Language (JDL)**
  - Based upon **Condor's *CLASSified ADvertisement*** *language (ClassAd)*
    - Fully extensible language
    - A ClassAd
      - *Constructed with the classad construction operator []*
      - *It is a sequence of attributes separated by semi-colons.*
      - *An attribute is a pair (key, value), where value can be a Boolean, an Integer, a list of strings, …*
        - <attribute> = <value>;

Job Description Language (JDL)

- **The supported attributes are grouped into two categories:**
  - Job Attributes
    - Define the job itself
  - Resources
    - Taken into account by the Workload Manager 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*
      - *Have to be prefixed with “other.”*
    - ***Data and Storage resources***
      - *Input data to process, Storage Element (SE) where to store output data, protocols spoken by application when accessing SEs*

JDL: Relevant Attributes (1)

- JobType
  - *Normal* (simple, sequential job), *DAG*, *Interactive*, *MPICH*, *Checkpointable*
- Executable **(mandatory)**
  - The command name
- Arguments **(optional)**
  - Job command line arguments
- StdInput**,** StdOutput**,** StdError **(optional)**
  - Standard input/output/error of the job
- Environment
  - List of environment settings
- InputSandbox **(optional)**
  - List of files on the UI's local disk needed by the job for running
  - The listed files will be staged automatically to the remote resource
- OutputSandbox **(optional)**
  - List of files, generated by the job, which have to be retrieved

JDL: Relevant Attributes (2)

- Requirements
  - 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: *other.**GlueCEStateStatus*** == *"Production"* (the resource has to be able to accept jobs and dispatch them on WNs)
- Rank
  - **Expresses preference** (how to rank resources that have already met the Requirements expression)
  - Specified using attributes of resources published in the Information Service
  - If not specified, default value defined in the UI configuration file is considered
    - Default: - *other.**GlueCEStateEstimatedResponseTime*** (the lowest estimated traversal time)
    - Default: *other.**GlueCEStateFreeCPUs*** (the highest number of free CPUs) for parallel jobs (see later)

JDL: Relevant Attributes (3)

- **InputData**
  - Refers to data used as input by the job: these data are published in the Replica Catalog and stored in the Storage Elements
  - LFNs and/or GUIDs
- **InputSandbox**
  - Executable, files etc. that are sent to the job
- **DataAccessProtocol (mandatory if InputData has been specified)**
  - The protocol or the list of protocols that the application is able to speak with for accessing *InputData* on a given Storage Element
- **OutputSE**
  - The Uniform Resource Identifier of the output Storage Element
  - RB uses it to choose a Computing Element that is compatible with the job and is close to Storage Element

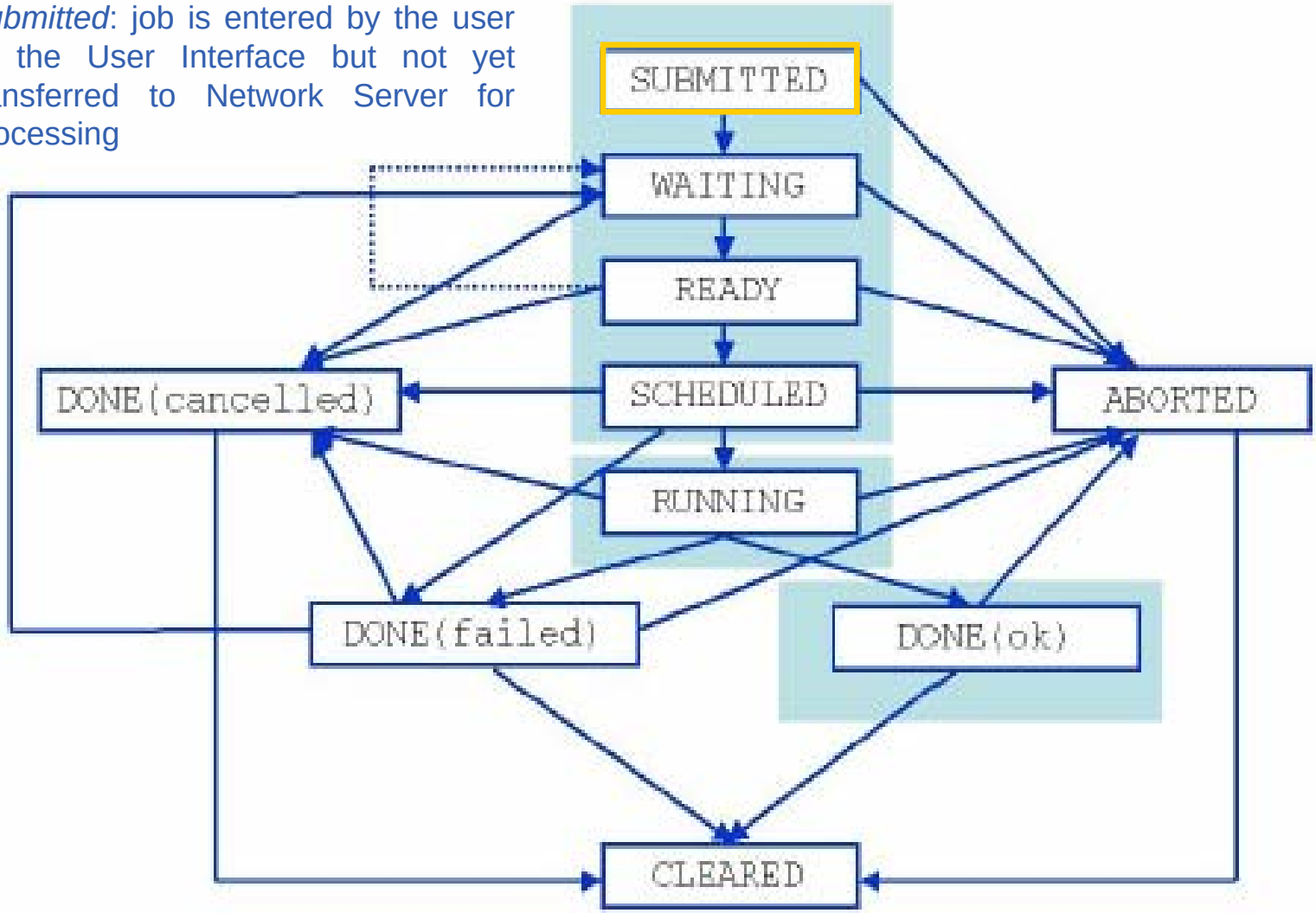
Details in Data Management lecture

Example of JDL File

```
[
JobType="Normal";
Executable = "gridTest";
StdError = "stderr.log";
StdOutput = "stdout.log";
InputSandbox = {"/home/mydir/test/gridTest"};
OutputSandbox = {"stderr.log", "stdout.log"};
InputData = {"lfn:/glite/myvo/mylfn" };
DataAccessProtocol = "gridftp";
Requirements = other.GlueHostOperatingSystemNameOpSys
  == "LINUX"
                && other.GlueCEStateFreeCPUs>=4;
Rank = other.GlueCEPolicyMaxCPUTime;
]
```

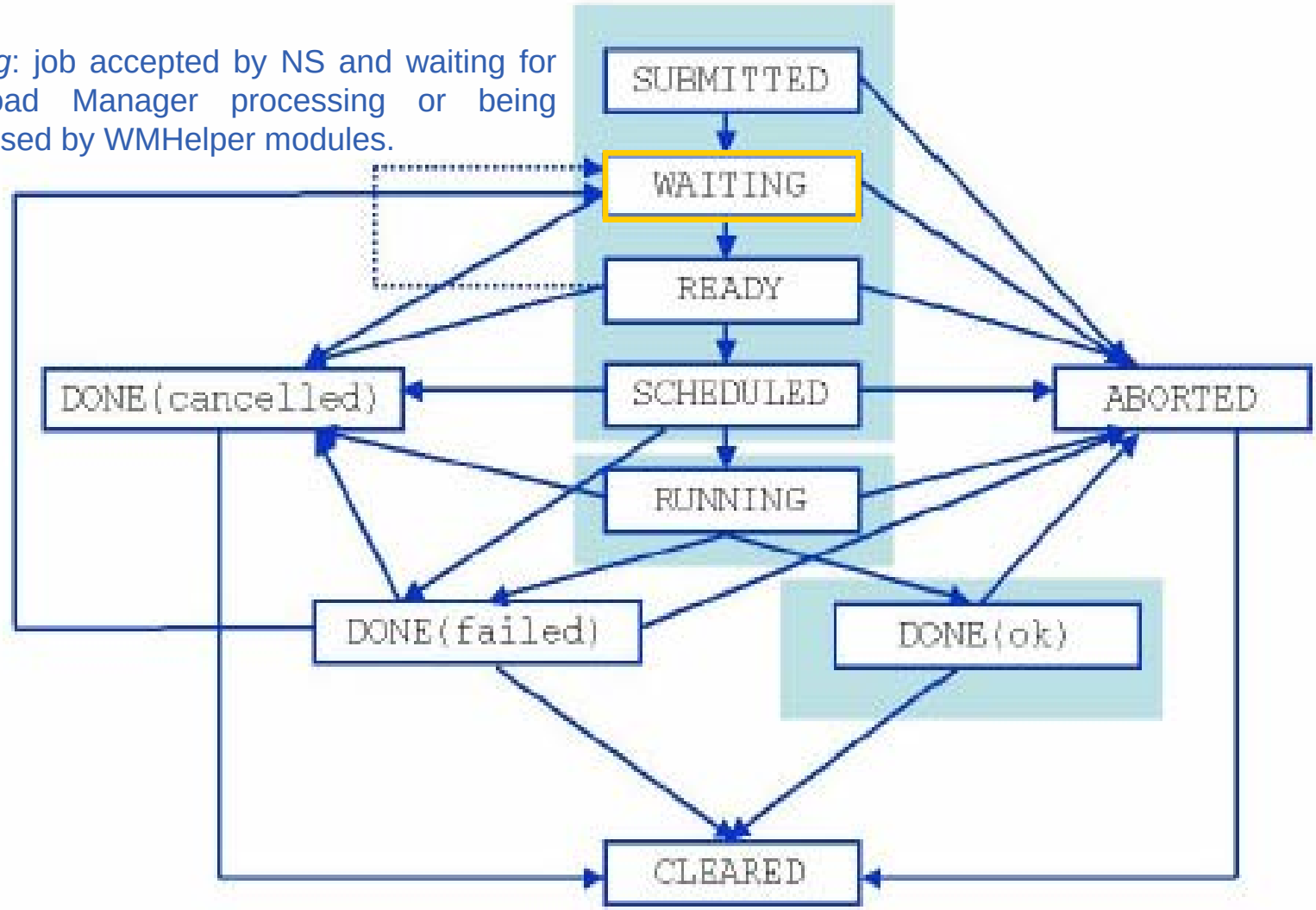
Jobs State Machine (1/9)

Submitted: job is entered by the user to the User Interface but not yet transferred to Network Server for processing

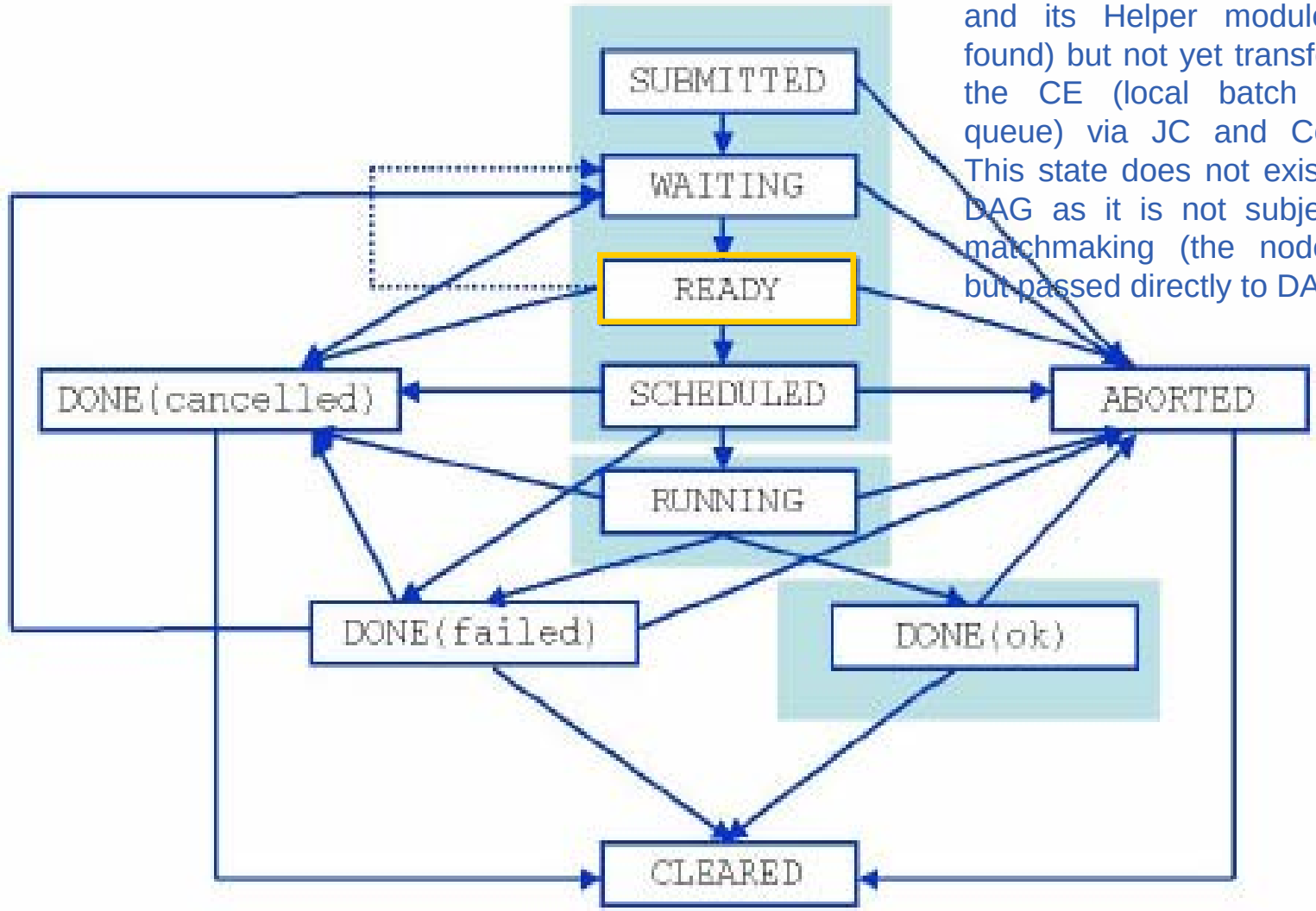


Jobs State Machine (2/9)

Waiting: job accepted by NS and waiting for Workload Manager processing or being processed by WMHelper modules.

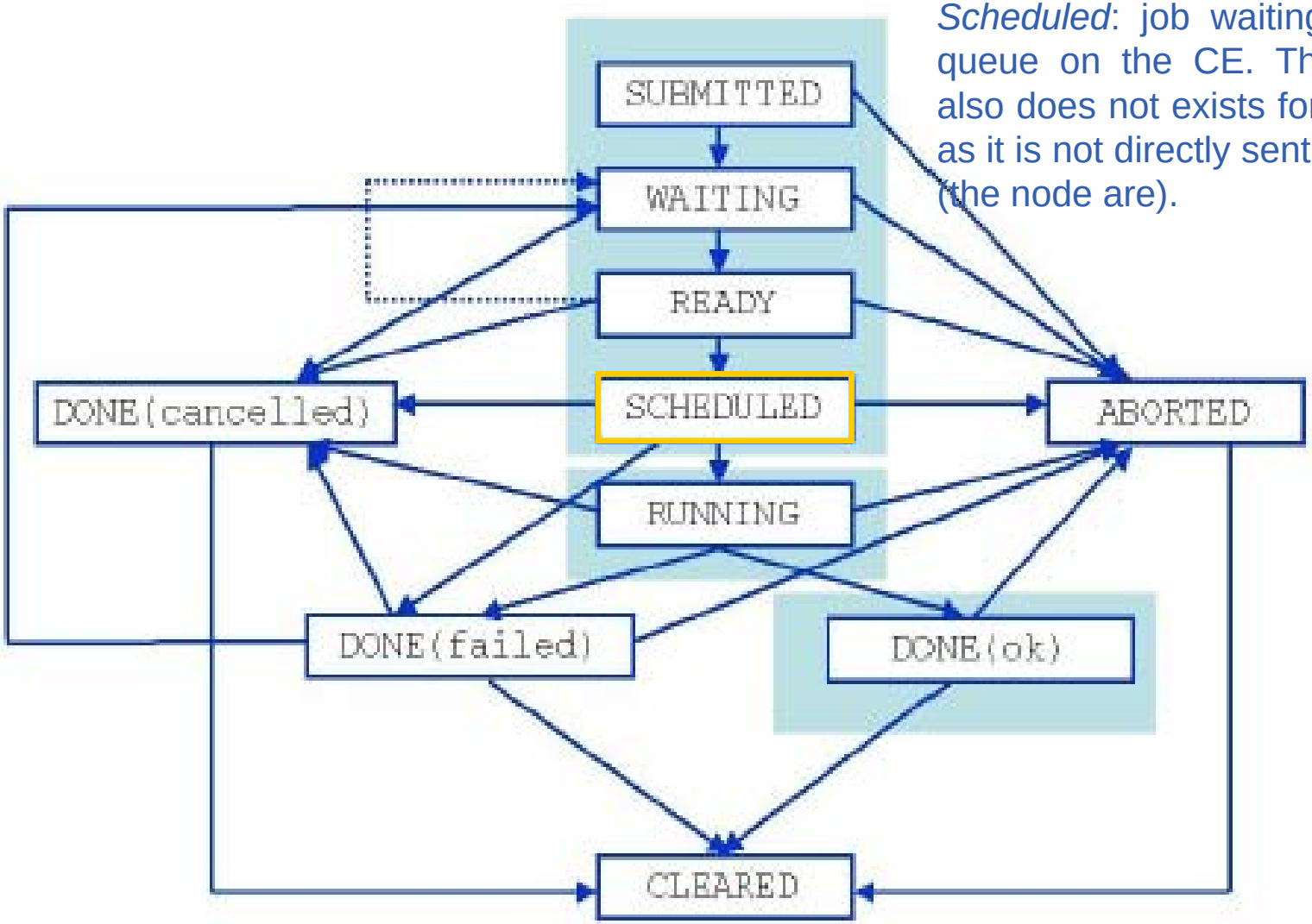


Jobs State Machine (3/9)



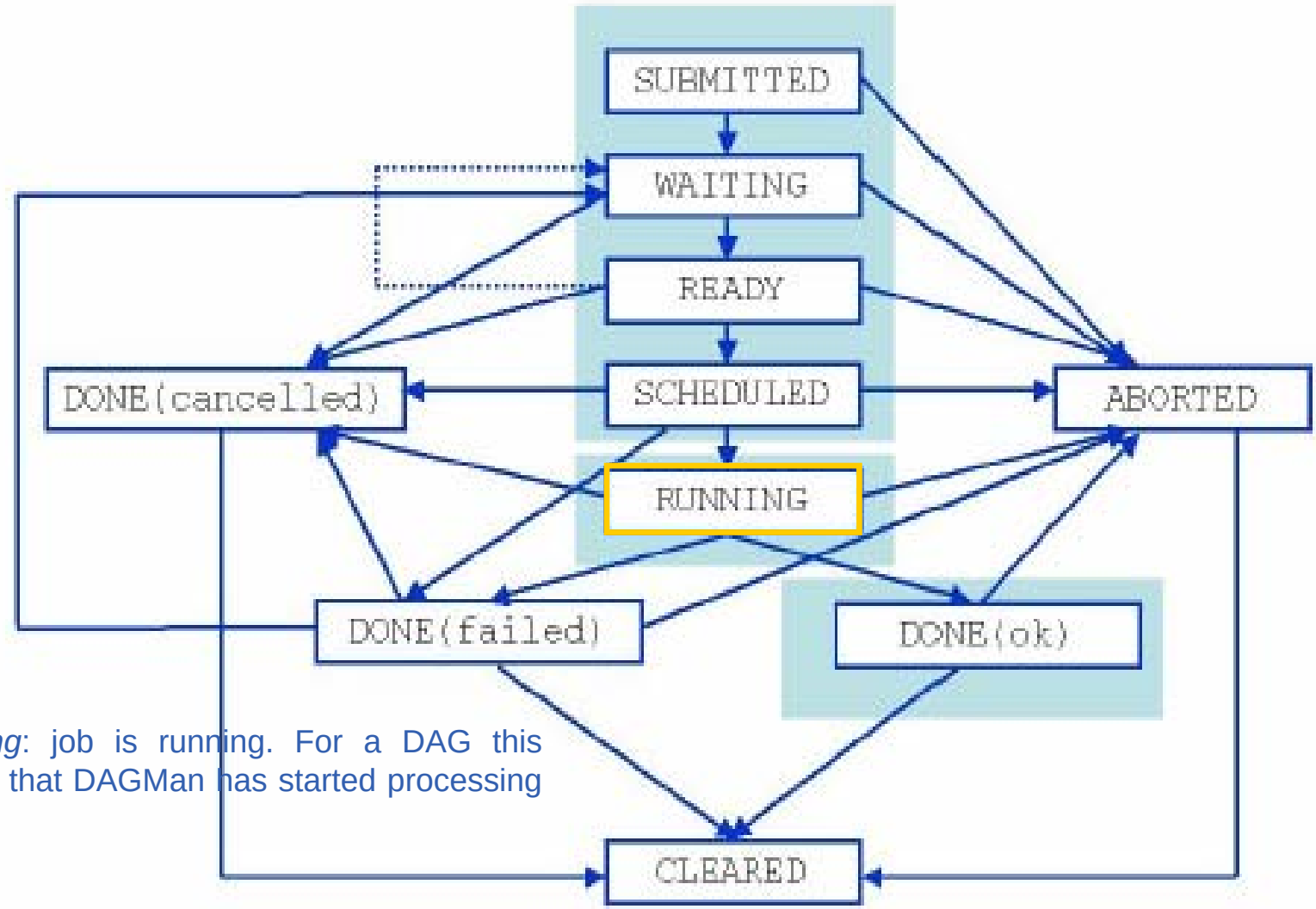
Ready: job processed by WM and its Helper modules (CE found) but not yet transferred to the CE (local batch system queue) via JC and CondorC. This state does not exists for a DAG as it is not subjected to matchmaking (the nodes are) but passed directly to DAGMan.

Jobs State Machine (4/9)



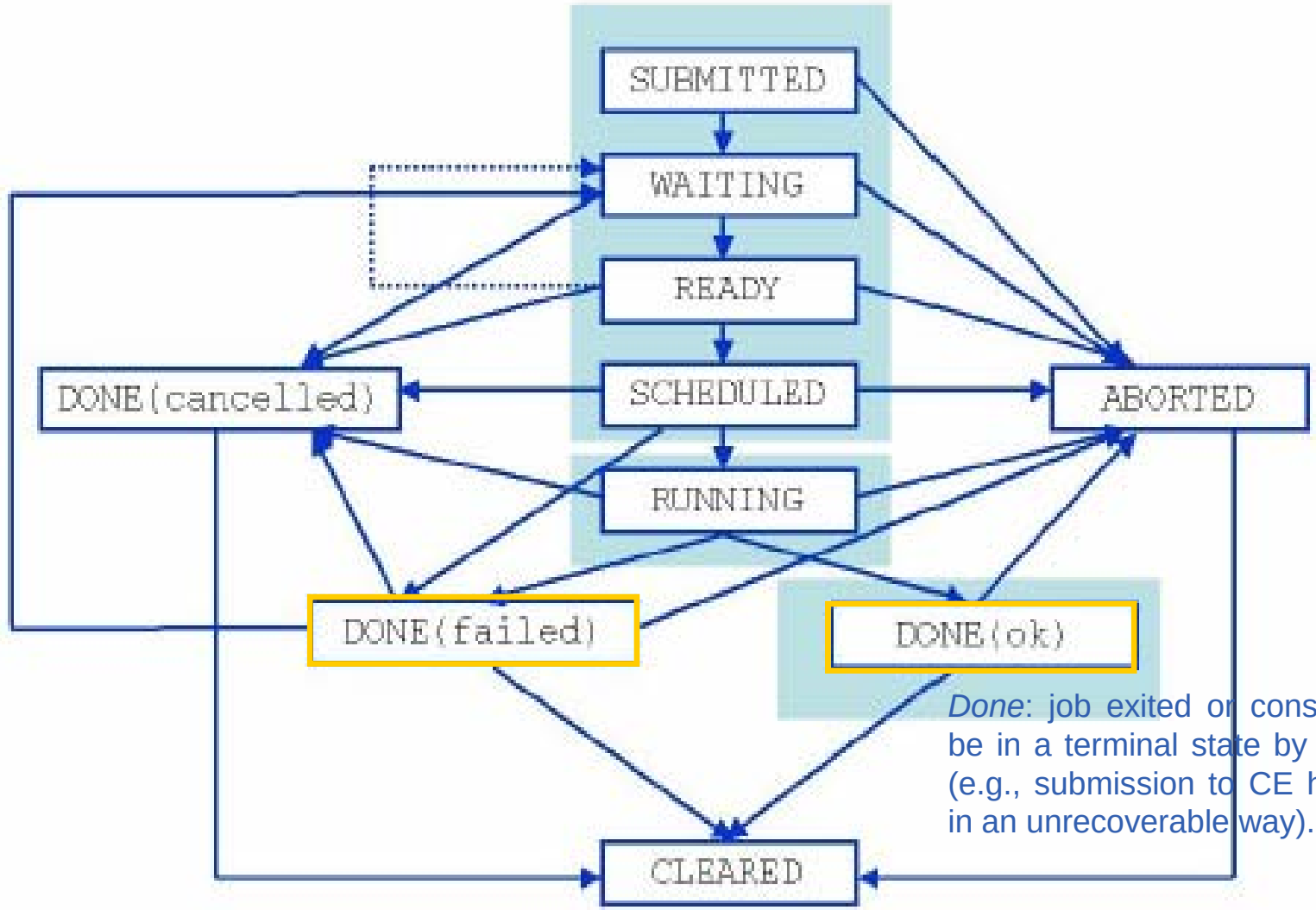
Scheduled: job waiting in the queue on the CE. This state also does not exists for a DAG as it is not directly sent to a CE (the node are).

Jobs State Machine (5/9)



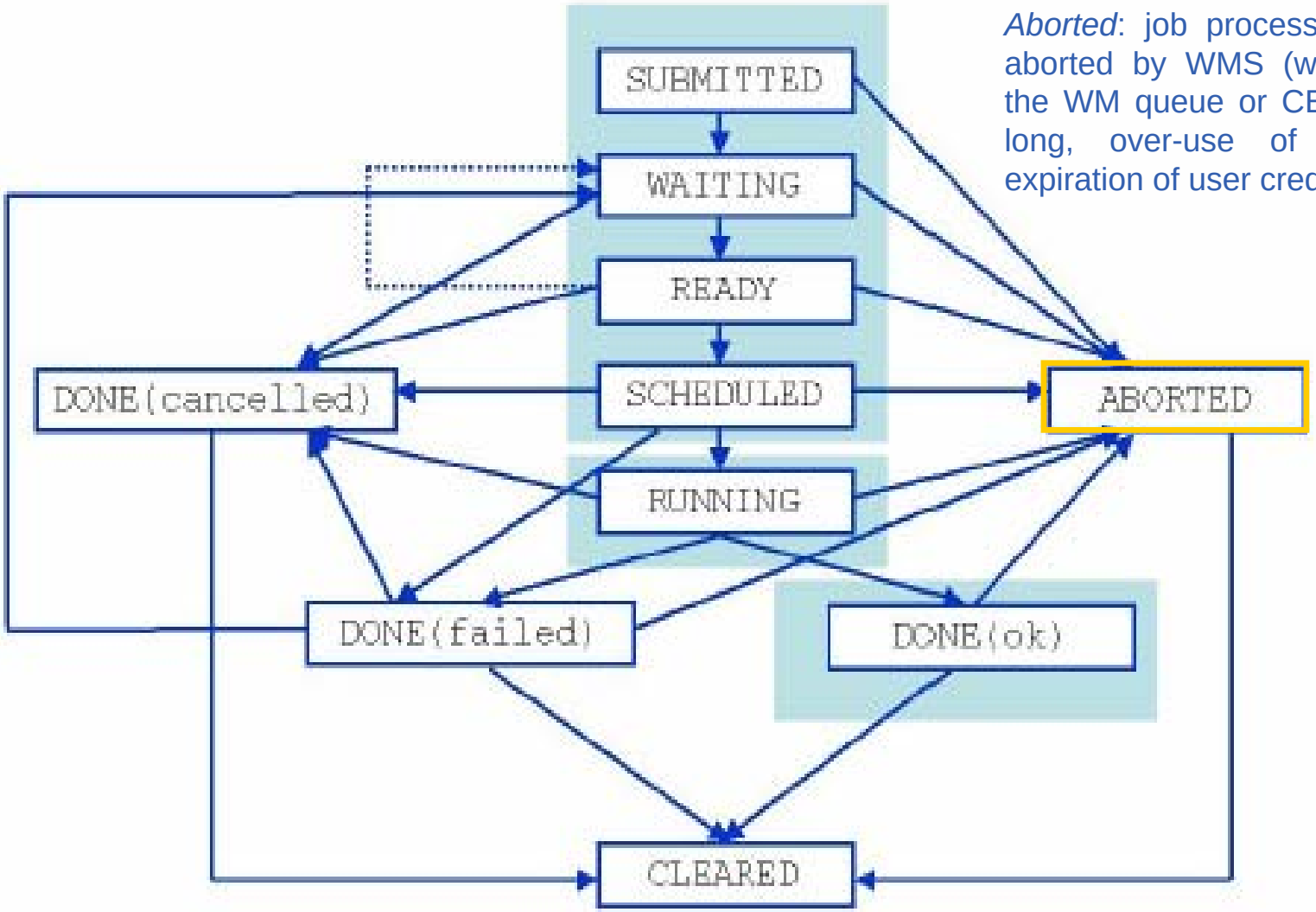
Running: job is running. For a DAG this means that DAGMan has started processing it.

Jobs State Machine (6/9)



Done: job exited or considered to be in a terminal state by CondorC (e.g., submission to CE has failed in an unrecoverable way).

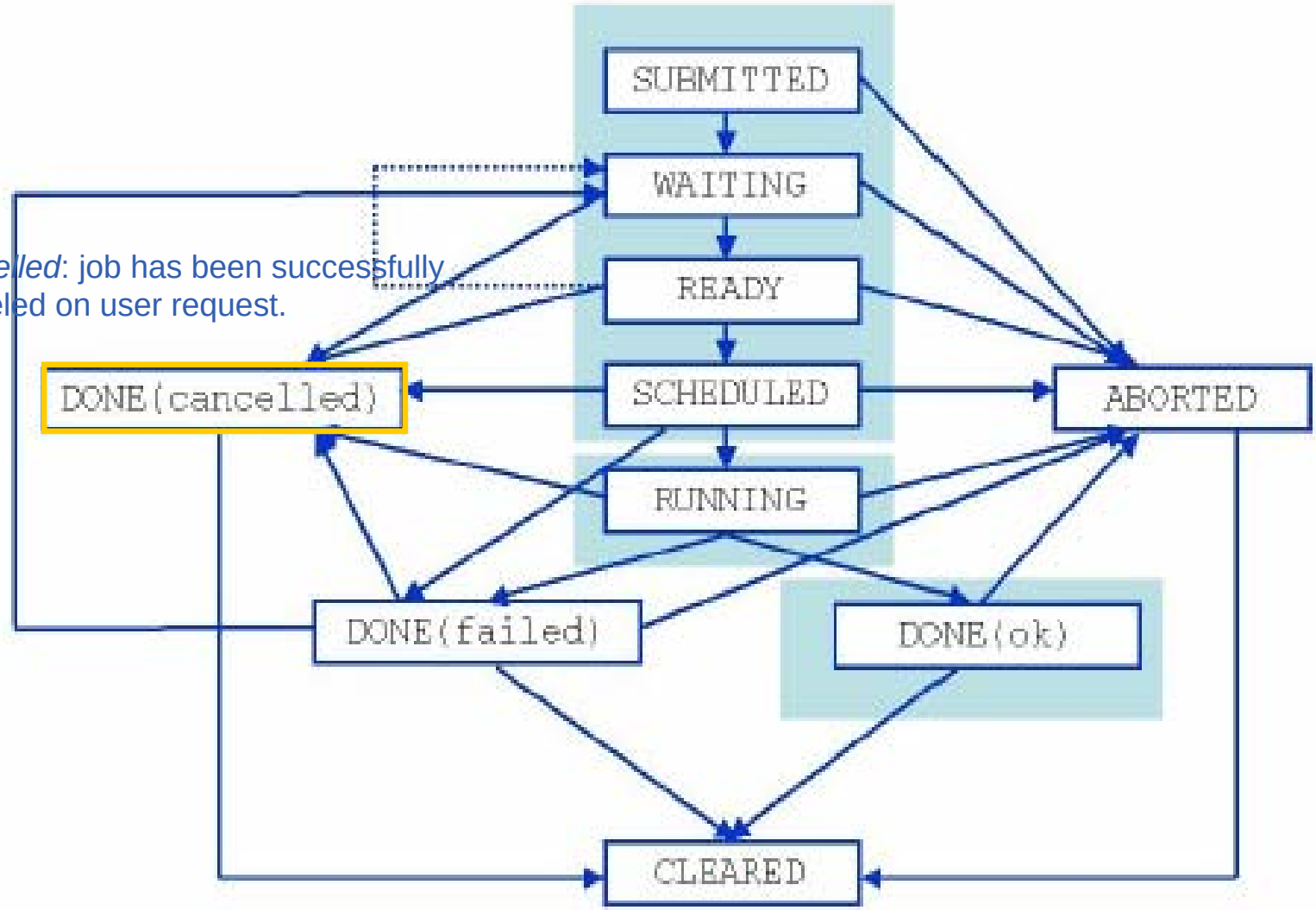
Jobs State Machine (7/9)



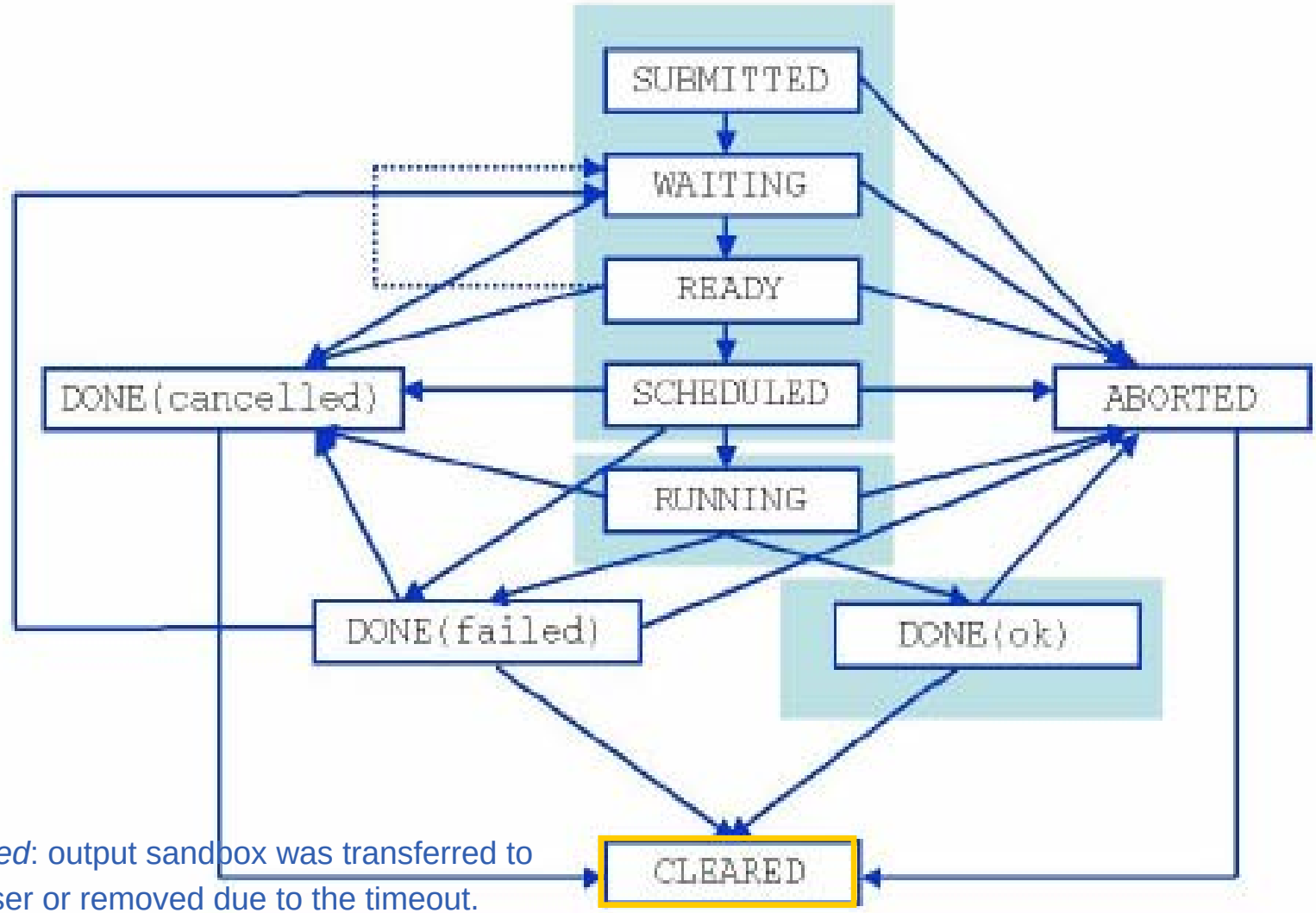
Aborted: job processing was aborted by WMS (waiting in the WM queue or CE for too long, over-use of quotas, expiration of user credentials).

Jobs State Machine (8/9)

Cancelled: job has been successfully canceled on user request.



Jobs State Machine (9/9)



Cleared: output sandbox was transferred to the user or removed due to the timeout.

Job Submission Command Line Interface

```
glite-job-submit [-r <res_id>] [-c <config file>]
 [--vo <VO>] [-o <output file>]  <job.jdl>
```

-r the job is submitted directly to the computing element identified by <*res_id*>

-c the configuration file <*config file*> is pointed by the UI instead of the standard configuration file

--vo the Virtual Organisation (if user is not happy with the one specified in the UI configuration file)

-o the generated edg_jobId is written in the <*output file*>

Useful for other commands, e.g.:

glite-job-status -i *<input file>* (or jobId)

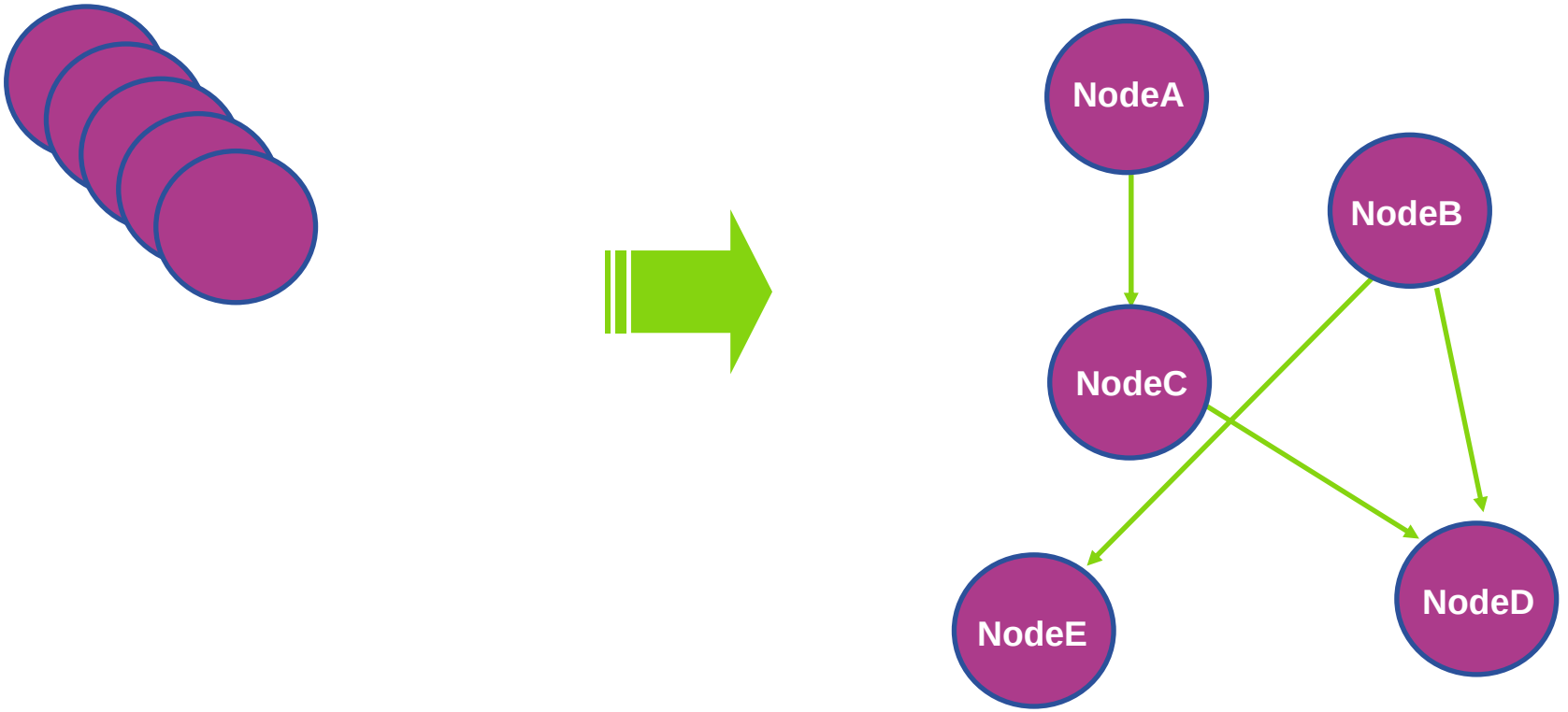
Job Resubmission

- **If something goes wrong, the WMS tries to reschedule and resubmit the job (possibly on a different resource satisfying all the requirements)**
- **Maximum number of resubmissions:**
  min(RetryCount, MaxRetryCount)
  - RetryCount: JDL attribute
  - MaxRetryCount: attribute in the “RB” configuration file
- **One can disable job resubmission for a particular job: *RetryCount=0;* in the JDL file**

Directed Acyclic Graphs (DAGs)

- **A DAG represents a set of jobs:**

Nodes* = *Jobs ***Edges* = *Dependencies***



DAG: JDL Structure

- Type = “DAG” → *Mandatory*
- VirtualOrganisation = “yourVO” → *Mandatory*
- Max_Nodes_Running = int >0 → *Optional*
- MyProxyServer = “…” → *Optional*
- Requirements = “…” → *Optional*
- Rank = “…” → *Optional*
- InputSandbox =  more later! → *Optional*
- ~~OutSandbox = “…”~~
- Nodes = nodeX  more later! → *Mandatory*

  Dependencies  more later! → *Mandatory*

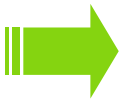
Attribute: Nodes

The *Nodes* attribute is the core of the DAG description;

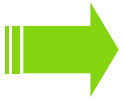
```
....
Nodes = [  nodefilename1 = [...]
           nodefilename2 = [...]
           .......

           dependencies = ...

]
```



```
Nodefilename1 = [ file = "foo.jdl"; ]
Nodefilename2 =
  [   file = "/home/vardizzo/test.jdl";
      retry = 2;          ]
```



```
Nodefilename1 = [
    description = [ JobType = "Normal";
                    Executable = "abc.exe";
                    Arguments = "1 2 3";
                    OutputSandbox = [...];
                    InputSandbox = [...];
                    .....  ]
    retry = 2;
                ]
```

Attribute: Dependencies

- **It is a list of lists representing the dependencies between the nodes of the DAG.**

```
....
Nodes = [  nodefilename1 = [...]
           nodefilename2 = [...]
           .......

           dependencies = ...

]
```

dependencies =
{nodefilename1, nodefilename2}

MANDATORY : YES!

dependencies = {};

{ nodefilename1, nodefilename2 }

{ { nodefilename1, nodefilename2 }, nodefilename3 }

{ { { nodefilename1, nodefilename2}, nodefilename3}, nodefilename4 }

InputSandbox & Inheritance

- **All nodes inherit the value of the attributes from the one specified for the DAG.**

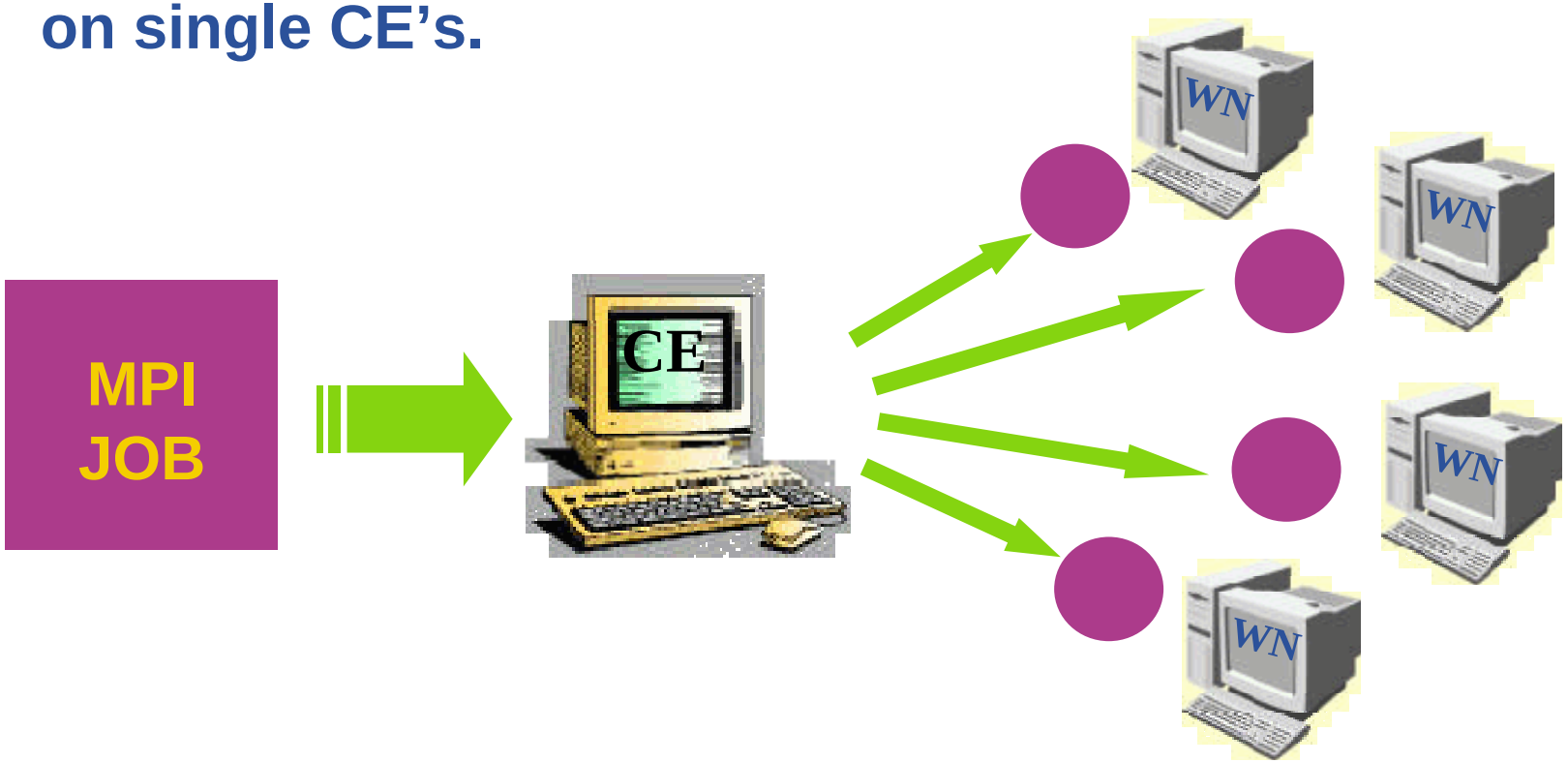
- **Nodes without any InputSandbox values, have to contain in their description an empty list:**

InputSandbox = *{ };*

```
NodeA= [
description = [
    JobType = "Normal";
    Executable = "abc.exe";
    OutputSandbox = {"myout.txt"};
    InputSandbox = {
        "/home/vardizzo/myfile.txt",
        root.InputSandbox;  };
    ]
]
```

Message Passing Interface (MPI)

- The MPI job is run in parallel on several processors.
- Libraries supported for parallel jobs: MPICH.
- Currently, execution of parallel jobs is supported only on single CE's.



MPI: JDL Structure

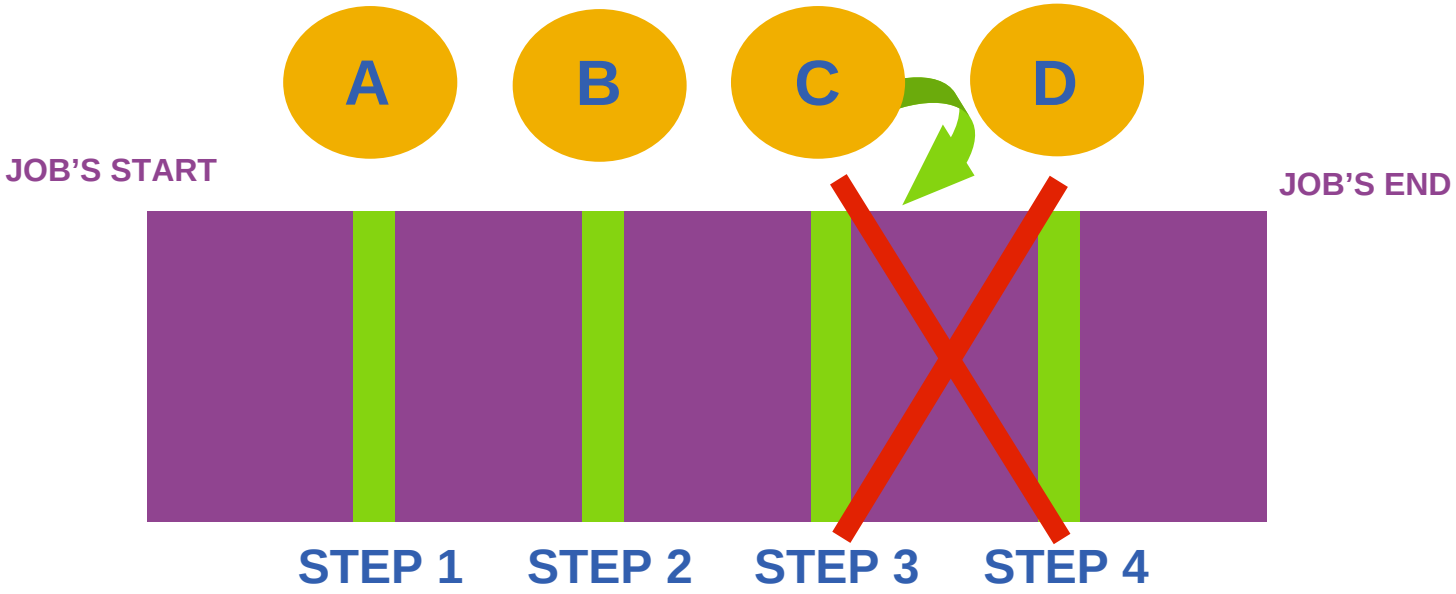
- Type = “job”; → *Mandatory*
- JobType = “MPICH”; → *Mandatory*
- Executable = “…”; → *Mandatory*
- NodeNumber = “int > 1”; → *Mandatory*
- Argument = “…”; → *Optional*
- Requirements = → *Mandatory*

  Member(“MpiCH”, other.GlueHostApplicationSoftwareRunTimeEnvironment)
  && other.GlueCEInfoTotalCPUs >= NodeNumber ;

- Rank = *other.GlueCEStateFreeCPUs;* → *Mandatory*

Logical Checkpointable Jobs

- It is a job that can be decomposed in several steps;
- In every step the job state can be saved in the LB and retrieved later in case of failures;
- The job can start running from a previously saved state instead from the beginning again.

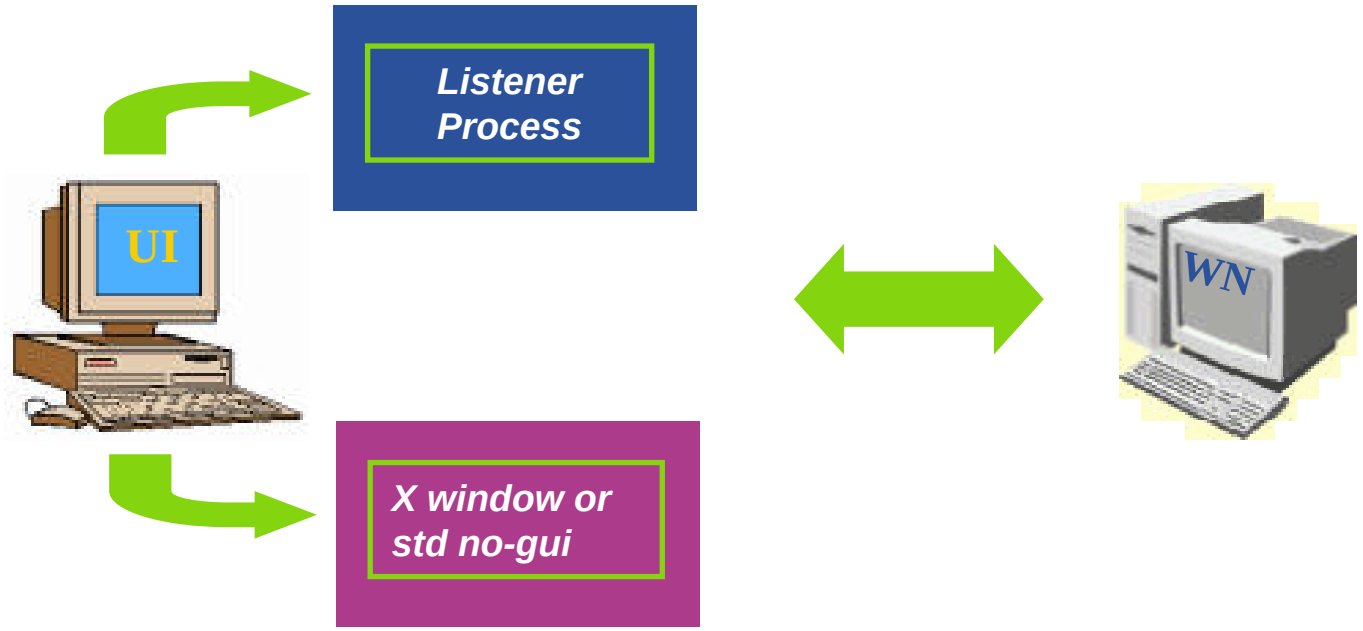


Checkpointable Jobs: JDL Structure

- Type = “job”; → *Mandatory*
- JobType = “checkpointable”; → *Mandatory*
- Executable = “…”; → *Mandatory*
- JobSteps = “list int | list string”; → *Mandatory*
- CurrentStep = “int > = 0”; → *Mandatory*
- Argument = “…”; → *Optional*
- Requirements = “…”; → *Optional*
- Rank = “”; → *Optional*

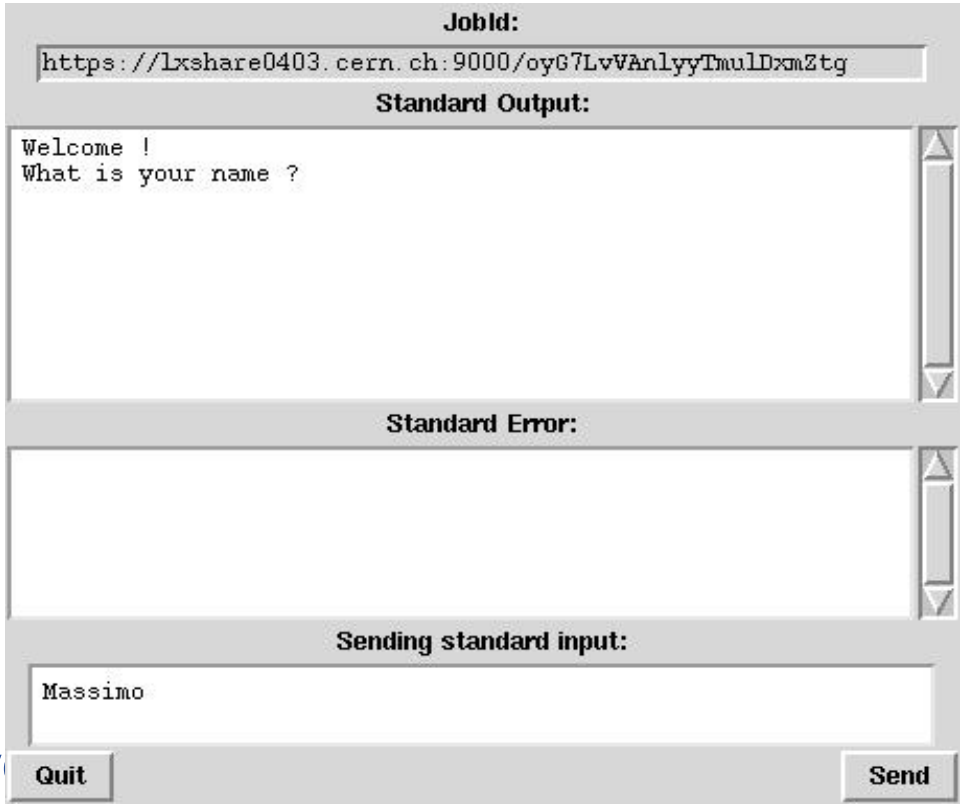
Interactive Jobs

- It is a job whose standard streams are forwarded to the submitting client.
- The DISPLAY environment variable has to be set correctly, because an X window is open.



Interactive Jobs

- **Specified setting JobType = “Interactive” in JDL**
- **When an interactive job is executed, a window for the stdin, stdout, stderr streams is opened**
  - Possibility to send the stdin to
  - the job
  - Possibility the have the stderr
  - and stdout of the job when it
  - is running
- **Possibility to start a window for**
- **the standard streams for a**
- **previously submitted interactiv**
- **job with command glite-job-attach**



Interactive Jobs: JDL Structure

- Type = “job”; → *Mandatory*
- JobType = “interactive”; → *Mandatory*
- Executable = “…”; → *Mandatory*
- Argument = “…”; → *Optional*
- ListenerPort = “int > 0”; → *Optional*
- OutputSandbox = “”; → *Optional*
- Requirements = “…”; → *Mandatory*
- Rank = “”; → *Mandatory*

gLite Commands:

glite-job-attach [options] <jobID>

gLite Commands

- **JDL Submission:**

  glite-job-submit –o guidfile jobCheck.jdl

- **JDL Status:**

  glite-job-status –i guidfile

- **JDL Output:**

  glite-job-output –i guidfile

- **Get Latest Job State:**

  glite-job-get-chkpt –o statefile –i guidfile

- **Submit a JDL from a state:**

  glite-job-submit -chkpt statefile –o guidfile jobCheck.jdl

- **See also [options] typing –help after the commands.**

Grid Accounting

A generic Grid accounting process accumulates info on Grid Usage by users/groups (VOs) and involves many subsequent phases as:

- **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 user operations and charge them.
- **To be used:** To track resource usage | To discover abuses (and help avoiding them).

- **Allows implementation of submission policies based on resource usage**
  - Exchange market among Grid users and Grid resource owners, which should result in market equilibrium → Load balancing on the Grid

During the metering phase the user payload on a resource needs to be correctly measured, and unambiguously assigned to the Grid User that directly or indirectly requested it to the Grid → Load Dedicated Sensors for Grid Resources

These pieces of information, when organized, form the Usage Record **for the user process → *Grid Unique Identifier* (for User, Resource, Job) plus the metrics of the resource consumption.**

A distributed architecture **is essential, as well as reliable and** fault tolerant communication **mechanisms.**

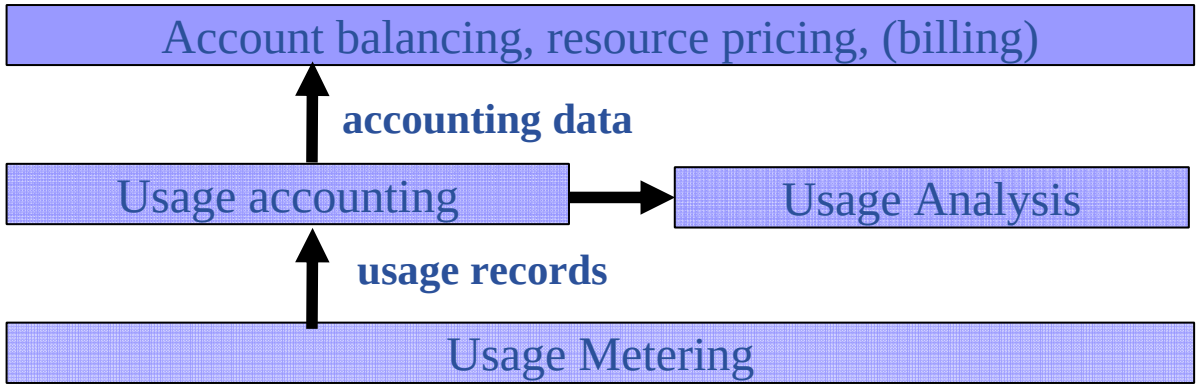
Different types of users are interested in different views of the usage records.

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.

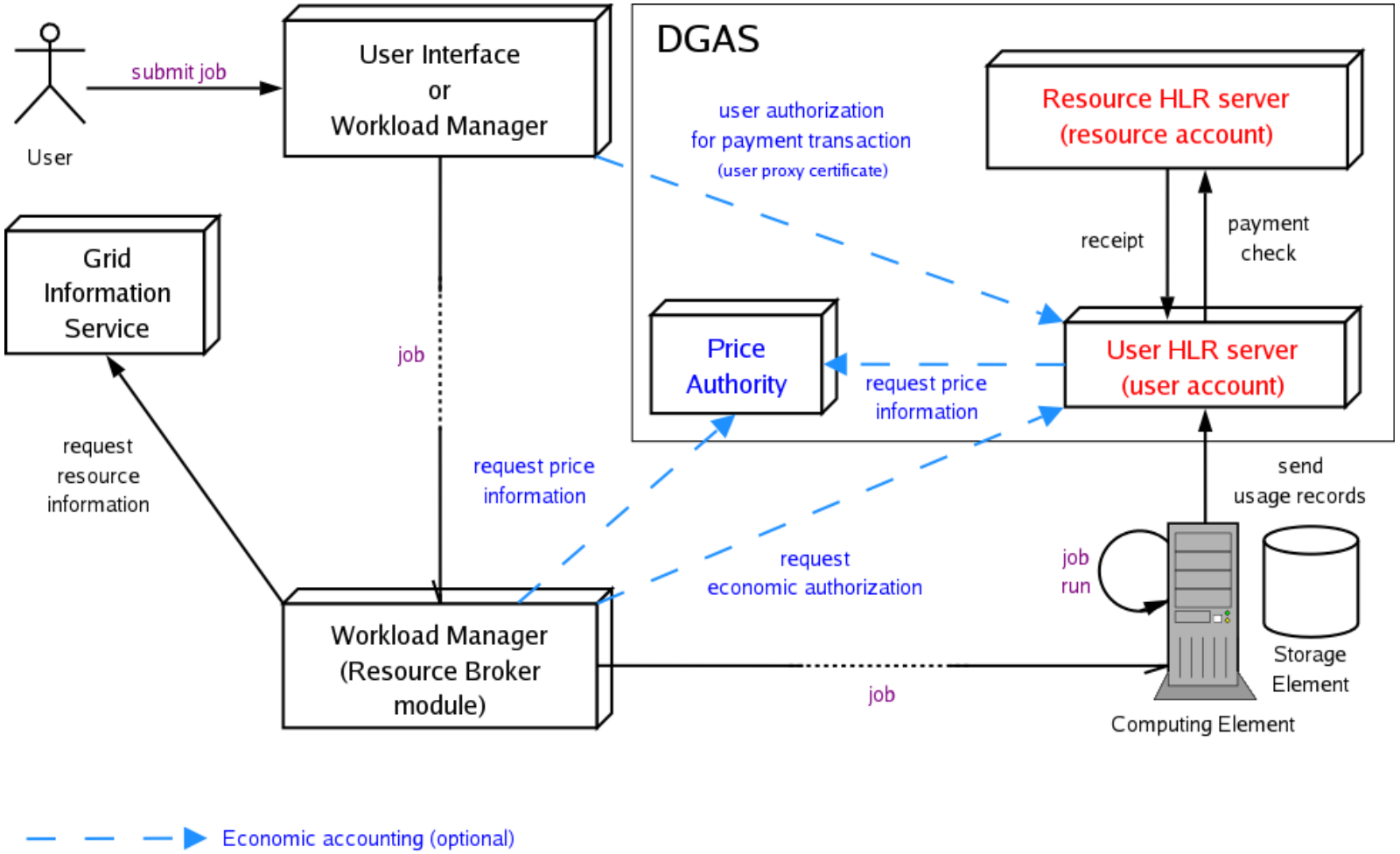
The Purpose of *DGAS* is to implement *Resource Usage Metering*, *Accounting* and *Account Balancing* (through *resource pricing*) in a fully distributed Grid environment. It is conceived to be distributed, secure and extensible**.**

The system is designed in order for Usage Metering, Accounting and Account Balancing (through resource pricing) to be indipendent layers.



DGAS Accounting Architecture

A simplified view of DGAS within the WMS context.



Further information

- **Workload Management**
  - `http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/`
  - **In particular WMS User & Admin Guide and JDL docs**
- **Condor ClassAd**
  - `http://www.cs.wisc.edu/condor/classad`
- **Condor DAGman**
  - `http://www.cs.wisc.edu/condor/dagman/`

References

- **gLite WMS's User Guide**
  - https://edms.cern.ch/document/572489/1
- **EGEE Middleware Architecture DJRA1.1**
  - https://edms.cern.ch/document/476451/
- **Practical approaches to Grid workload management in the EGEE project – CHEP 2004**
  - https://edms.cern.ch/document/503558
- **Grid accounting in EGEE, current practices – Terena Network Conference 2005**
  - http://www.terena.nl/conferences/tnc2005/programme/presentations/show.php?pres_id=107