Architecture of the gLite Workload Management System

LA ROCCA, Giuseppe
INFN Catania
giuseppe.larocca@ct.infn.it

Outline

This presentation will cover the following arguments:

- Overview of WMS Architecture
  - Task Queue, Information Supermarket, MatchMaker, Scheduling Policies, Job Submission Service, Job Logging & Bookkeeping.
- Job Description Language Overview
  - Principal Attributes
- Overview of WMProxy
  - New features
  - New request types

First Part

Glossary

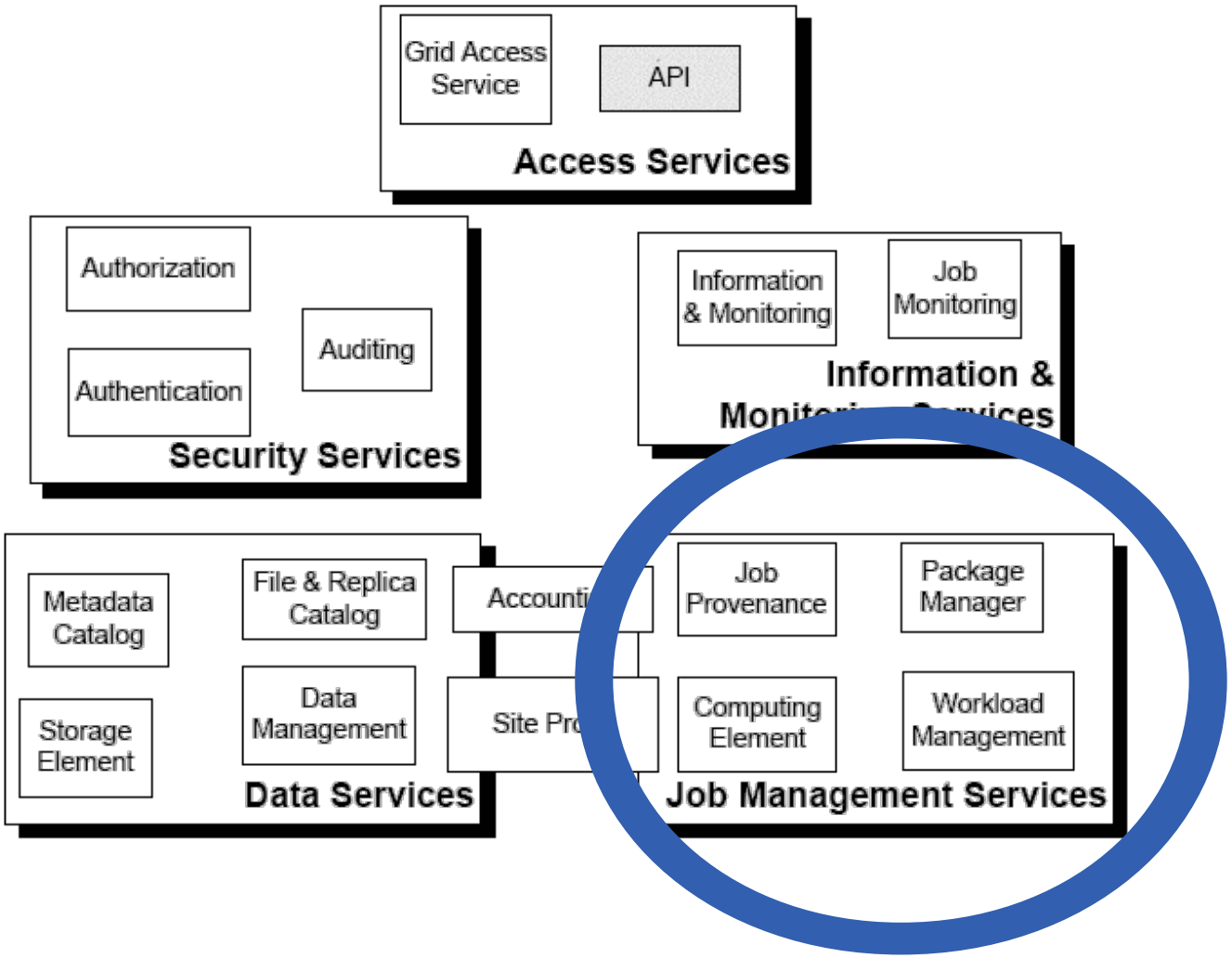
CE	Computing Element
FTA	File Transfer Agents
FTS	File Transfer Service
LB	Logging & Bookkeeping
R-GMA	Relational Grid Monitoring Architecture
SC	Single Catalog
SD	Service Discovery
UI	User Interface
VOMS	Virtual Organization Membership Service
WMS	Workload Management Service
WN	Worker Node

Services and Components

The following high-level services are part of this release of the gLite middleware (in alphabetical order):

- **Authorization, Authentication and Delegation Services**
- **Computing Element (CE)**
- **DGAS Server and Client**
- **File Transfer Service (FTS)**
- **File Transfer Agents (FTA)**
- **Logging and Bookkeeping Server (LB)**
- **R-GMA Servers, Client, Site Publisher, Service Tools**
- **Service Discovery (SD)**
- **Standard Worker Node**
- **User Interface (UI)**
- **VOMS and VOMS administration tools**
- **Workload Manager System (WMS)**
- **LCG File Catalog (LFC)**

Overview of gLite Middleware



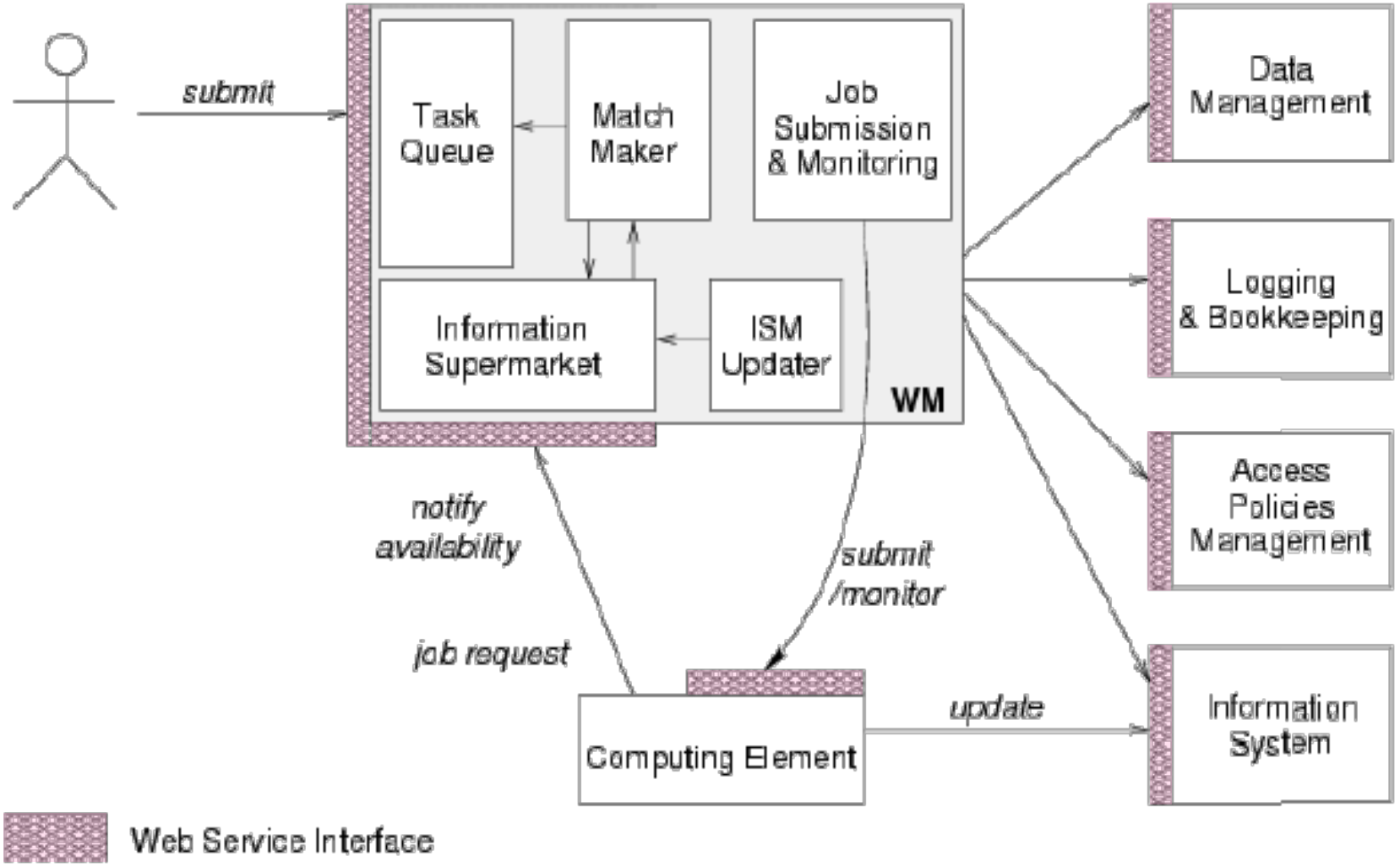
Workload Management System

- **Workload Management System (WMS) comprises a set of Grid middleware components responsible for distribution and management of tasks across Grid resources.**
- **Purpose of Workload Manager (WM) is accept and satisfy requests for job management coming from its clients**
  - meaning of the submission request is to pass the responsibility of the job to the WM.
    - WM will pass the job to an appropriate CE for execution
      - taking into account requirements and the preferences expressed in the job description
- **The decision of which resource should be used is the outcome of a matchmaking process.**

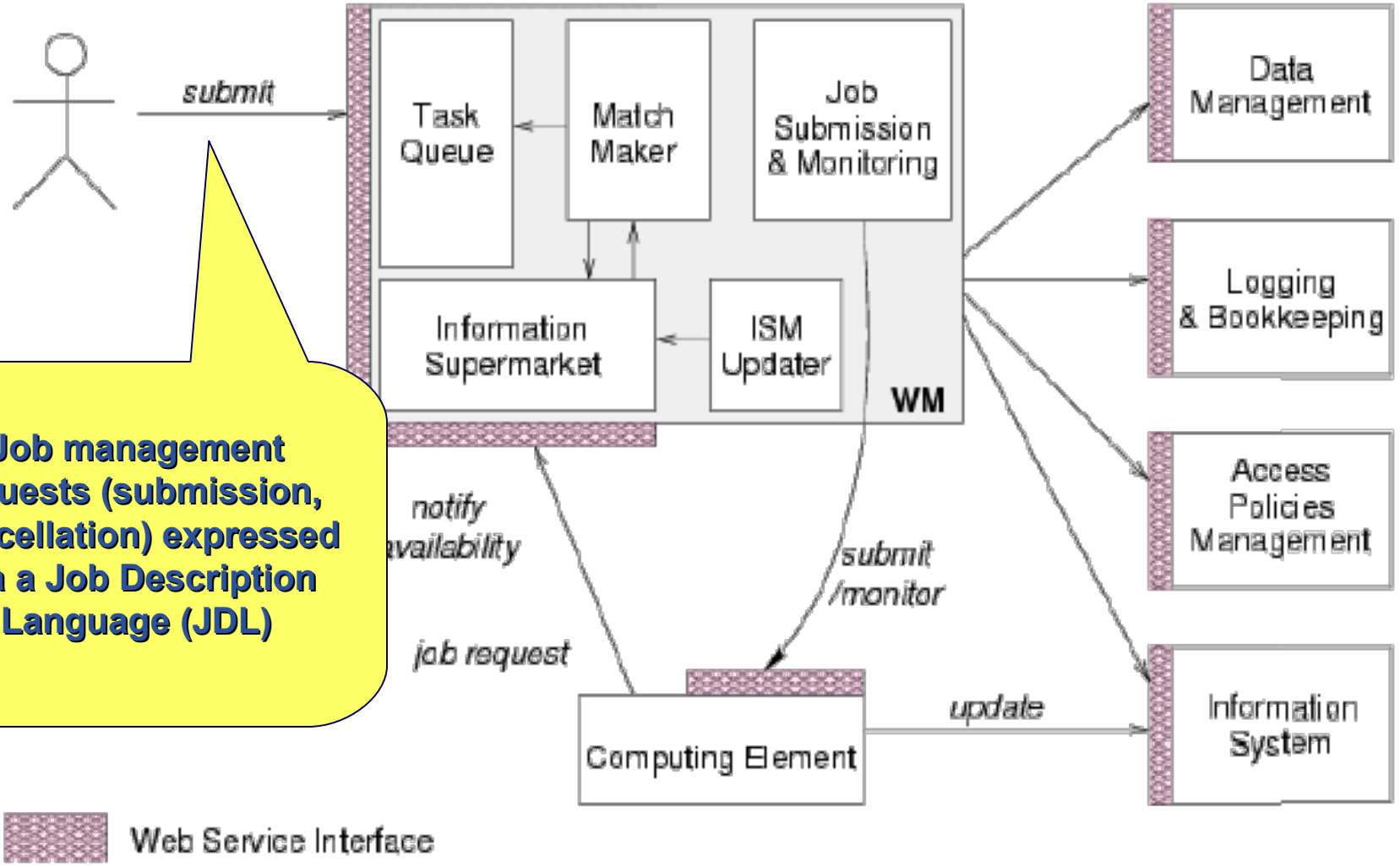
WMS's Scheduling Policies

- WMS can adopt:
  - **eager scheduling ("push" model)**
    - a job is bound to a resource as soon as possible. Once the decision has been taken, the job is passed to the selected resource for execution.
  - **lazy scheduling ("pull" model)**
    - the job is held by the WM until a resource becomes available. When this happens the resource is matched against the submitted job.

WMS's Architecture

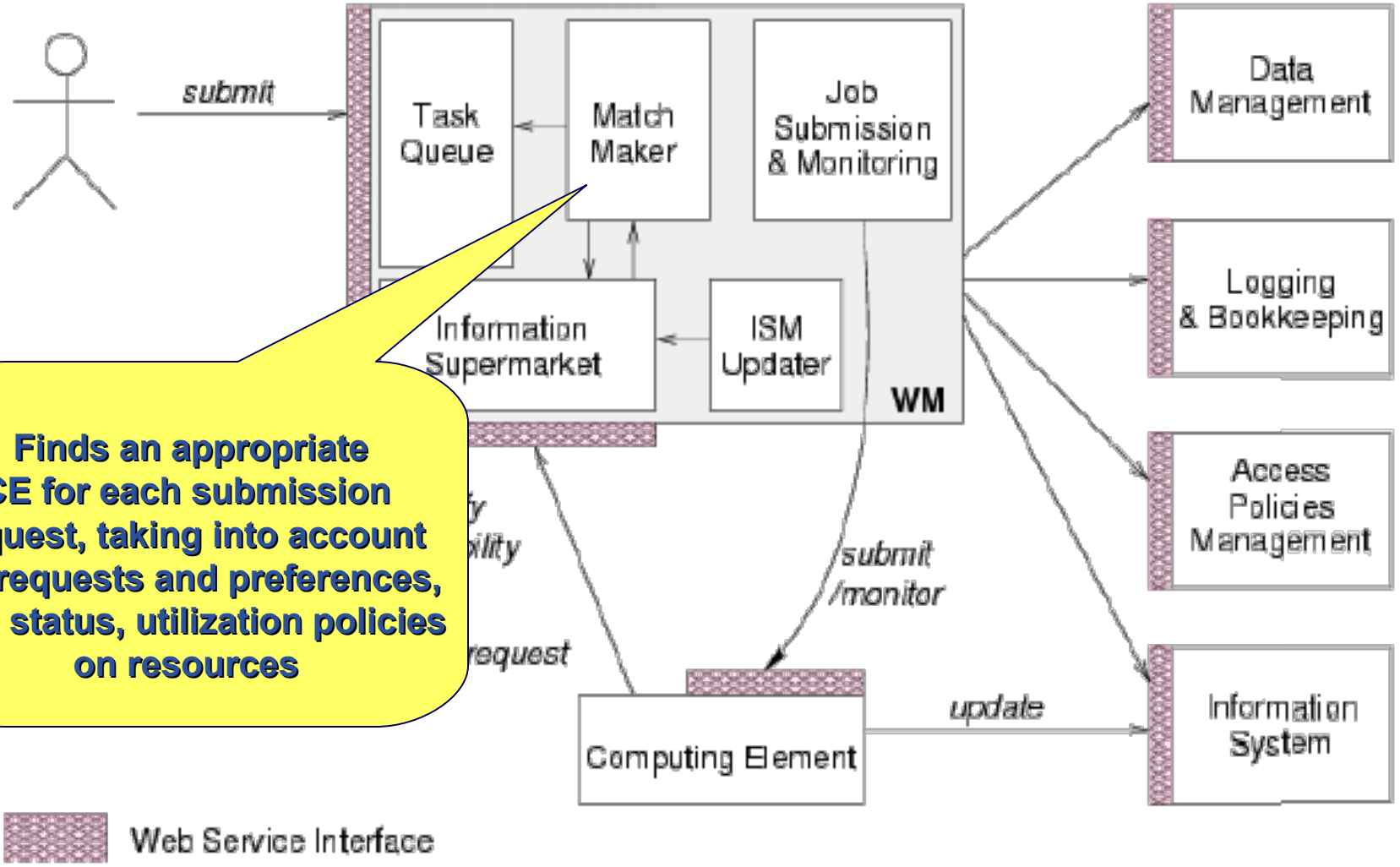


WMS's Architecture



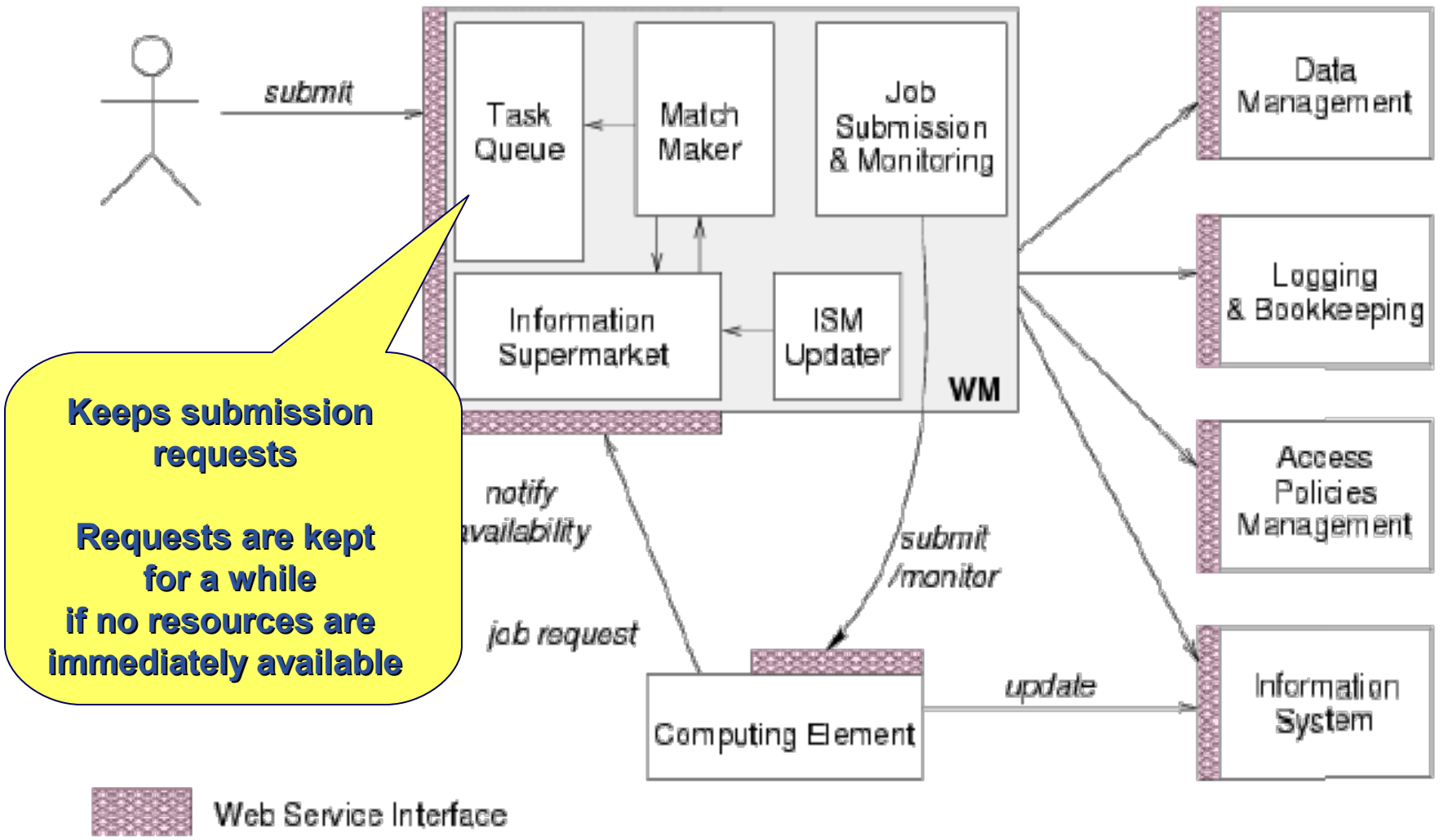
Job management requests (submission, cancellation) expressed via a Job Description Language (JDL)

WMS's Architecture



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

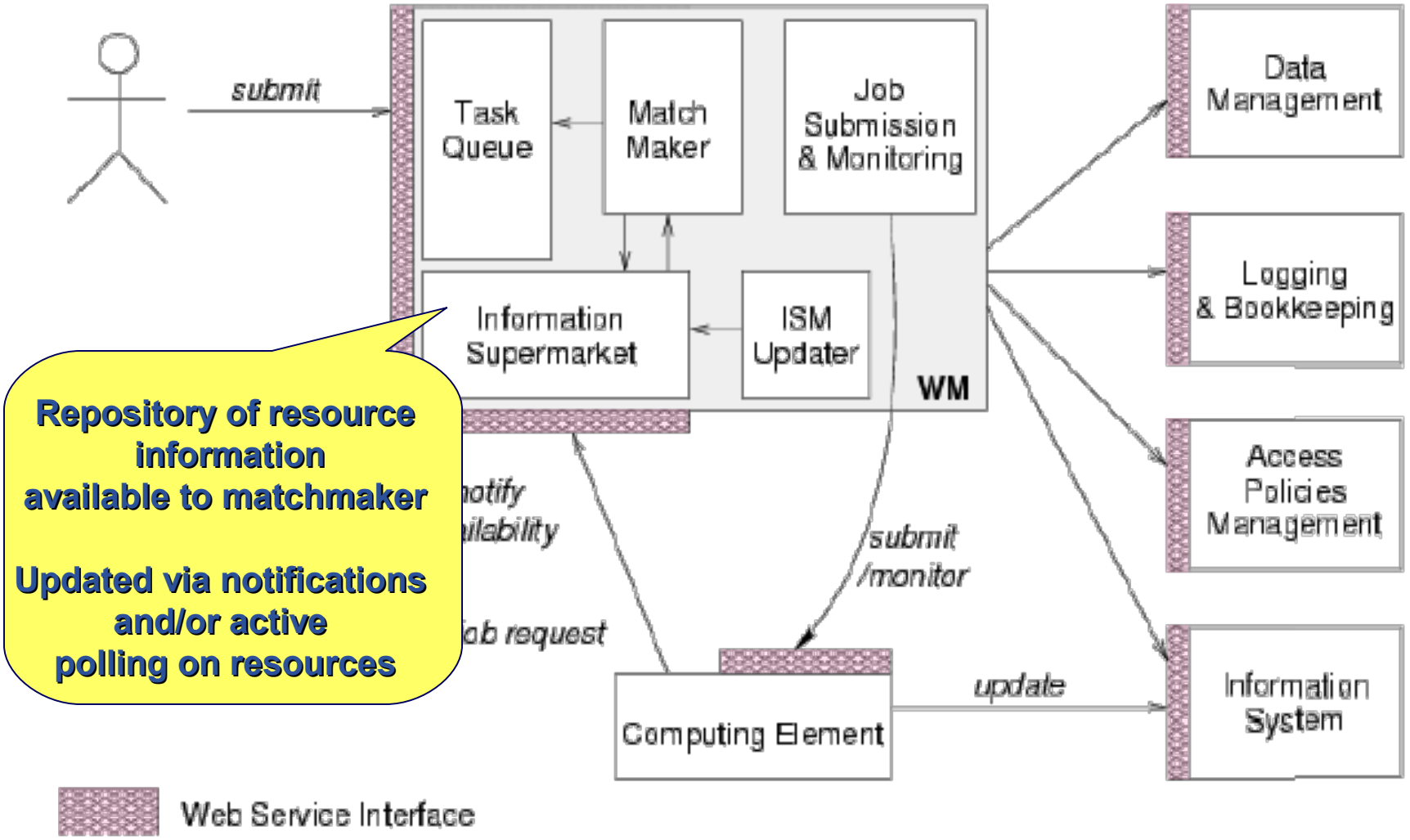
WMS's Architecture



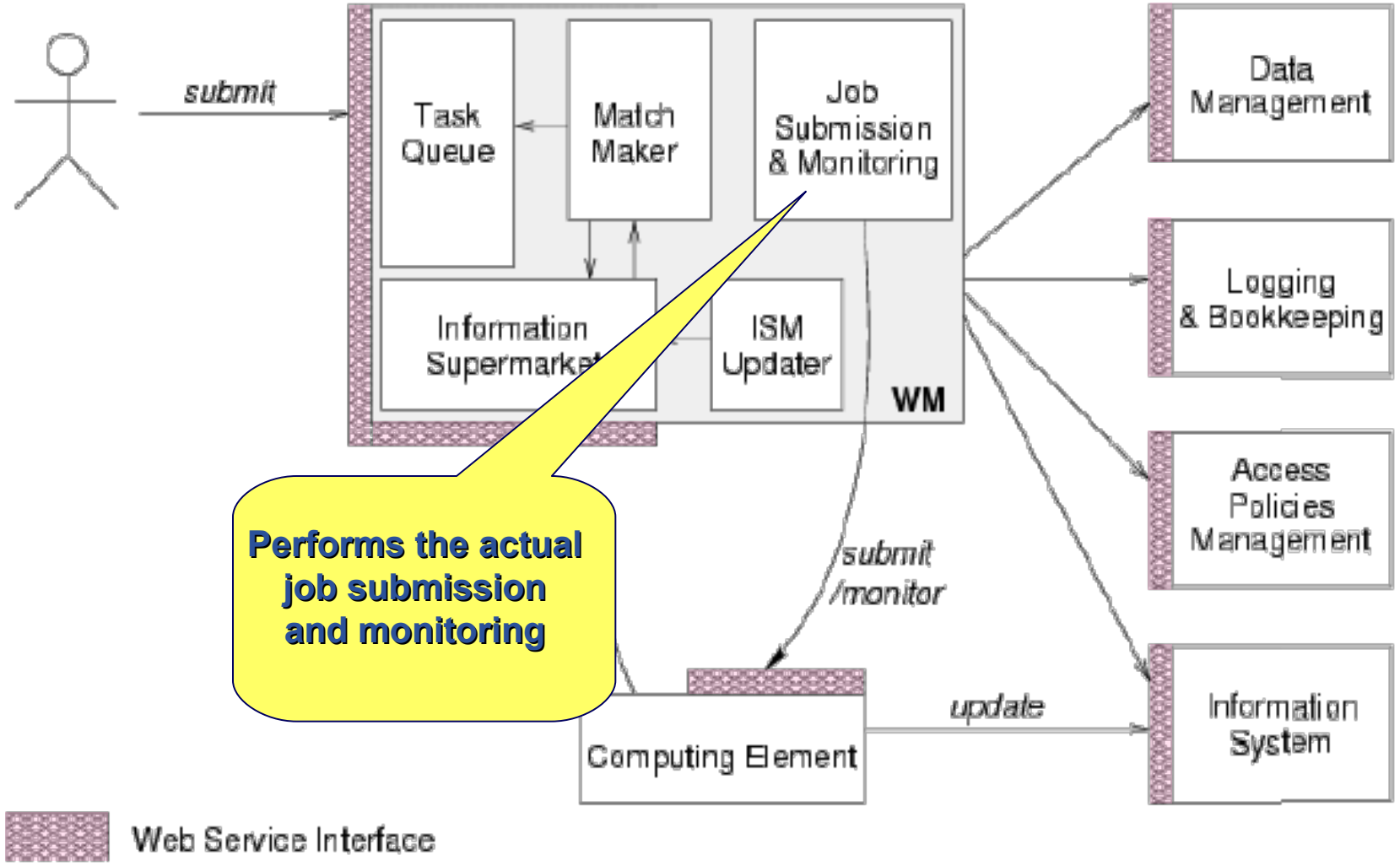
Keeps submission requests

Requests are kept for a while if no resources are immediately available

WMS's Architecture



WMS's Architecture



The Information Supermarket

- ISM represents one of the most notable improvements in the WM
- 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

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 performs the submission

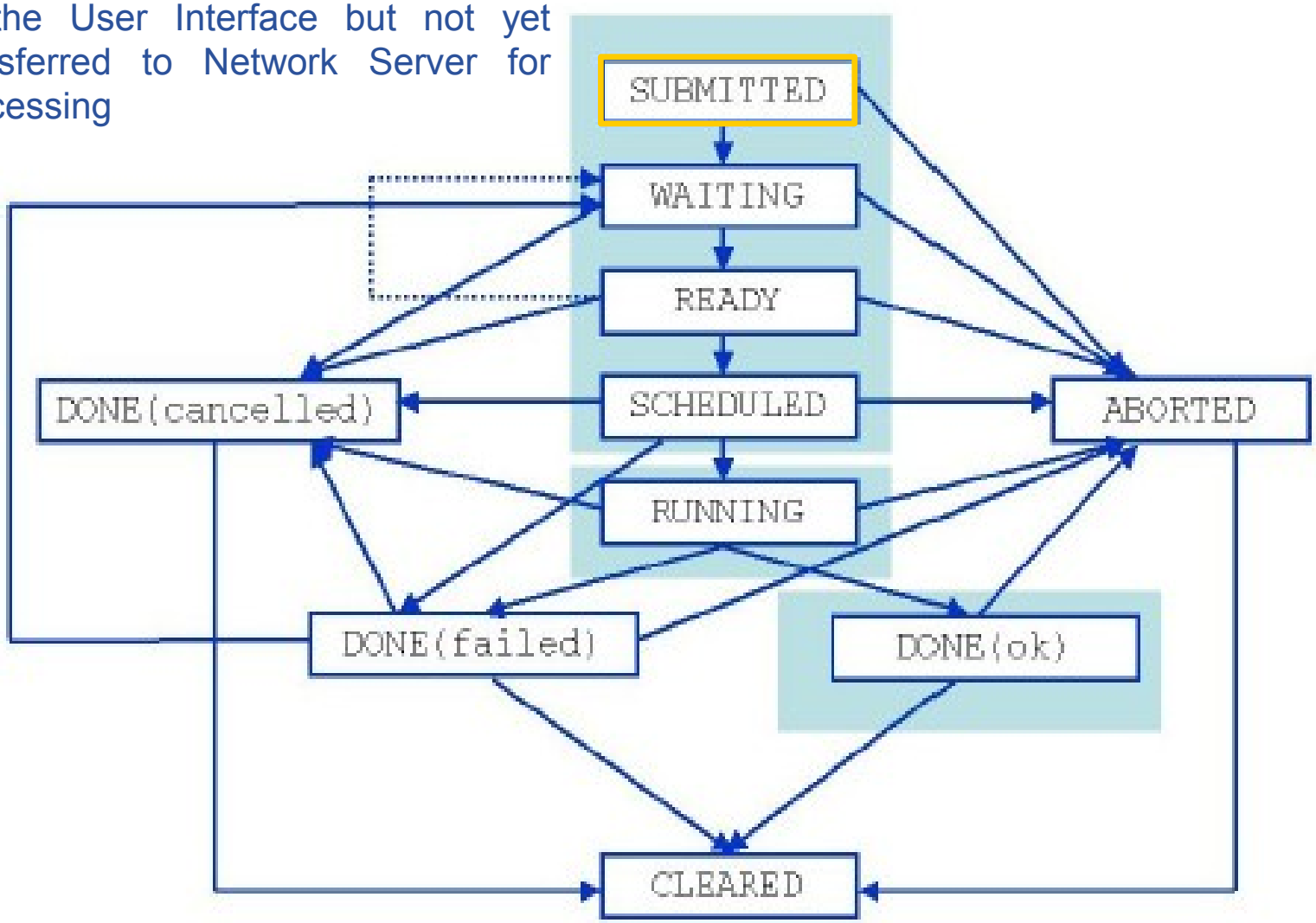
- Job Adapter (JA)
  - **is responsible for**
    - **making the final touches to the JDL expression for a job, before it is passed to CondorC for the actual submission**
    - **creating the job wrapper script that creates the appropriate execution environment in the CE worker node**
      - **transfer 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**

Job Submission Services

- Log Monitor (LM)
  - **is responsible for**
    - **watching the CondorC log file**
    - **intercepting interesting events concerning active jobs**
- Proxy Renewal Service
  - **is responsible to assure that,**
    - **for all the lifetime of a job, a valid user proxy exists within the WMS**
    - **MyProxy Server is contacted in order to renew the user's credential**
- Logging & Bookkeeping (LB)
  - **is responsible to**
    - **Stores events generated by the variuos components of the WMS**
    - **Querying the LB user can retrieve information about the job‘ status**

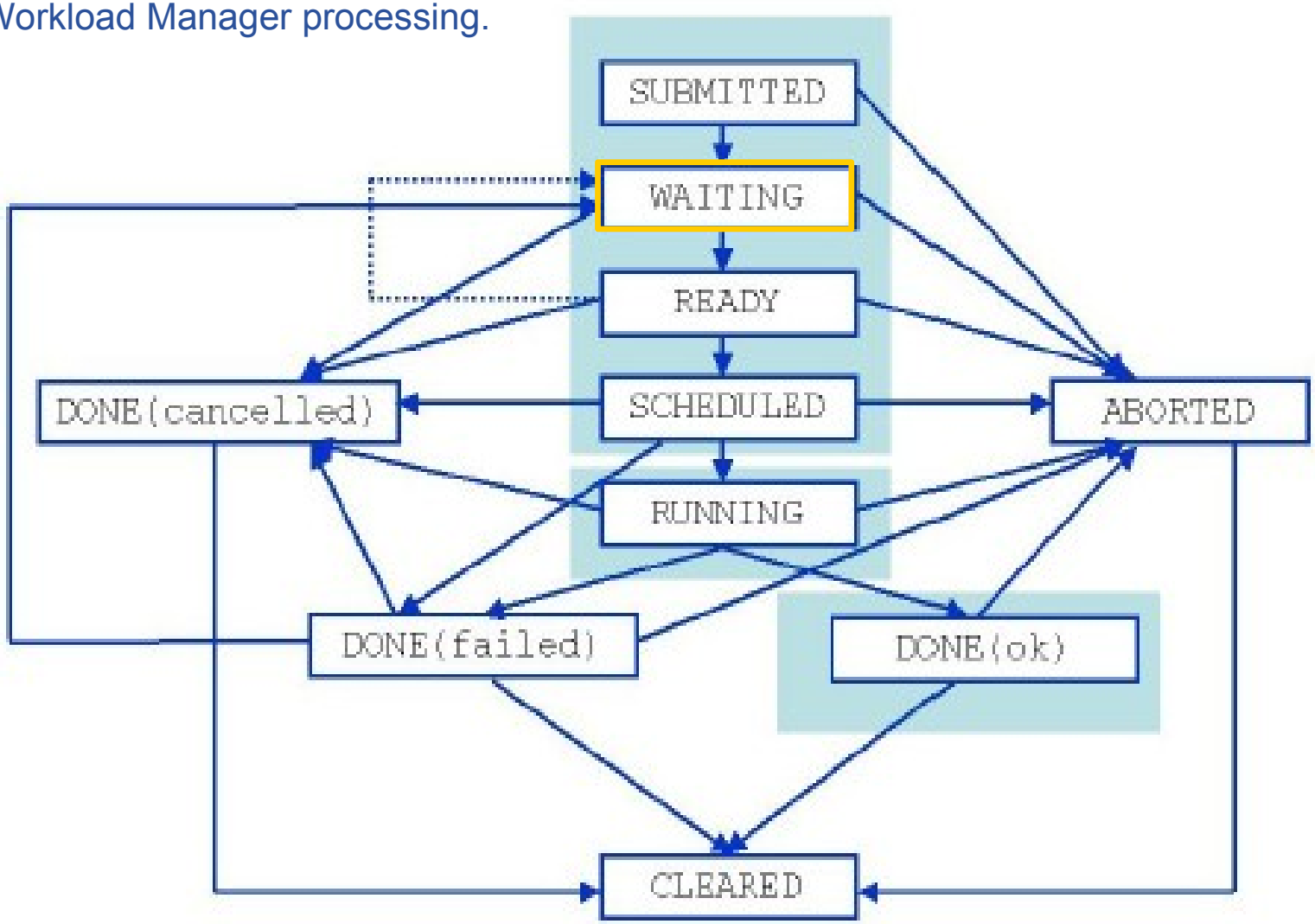
Jobs State Machine (1/9)

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



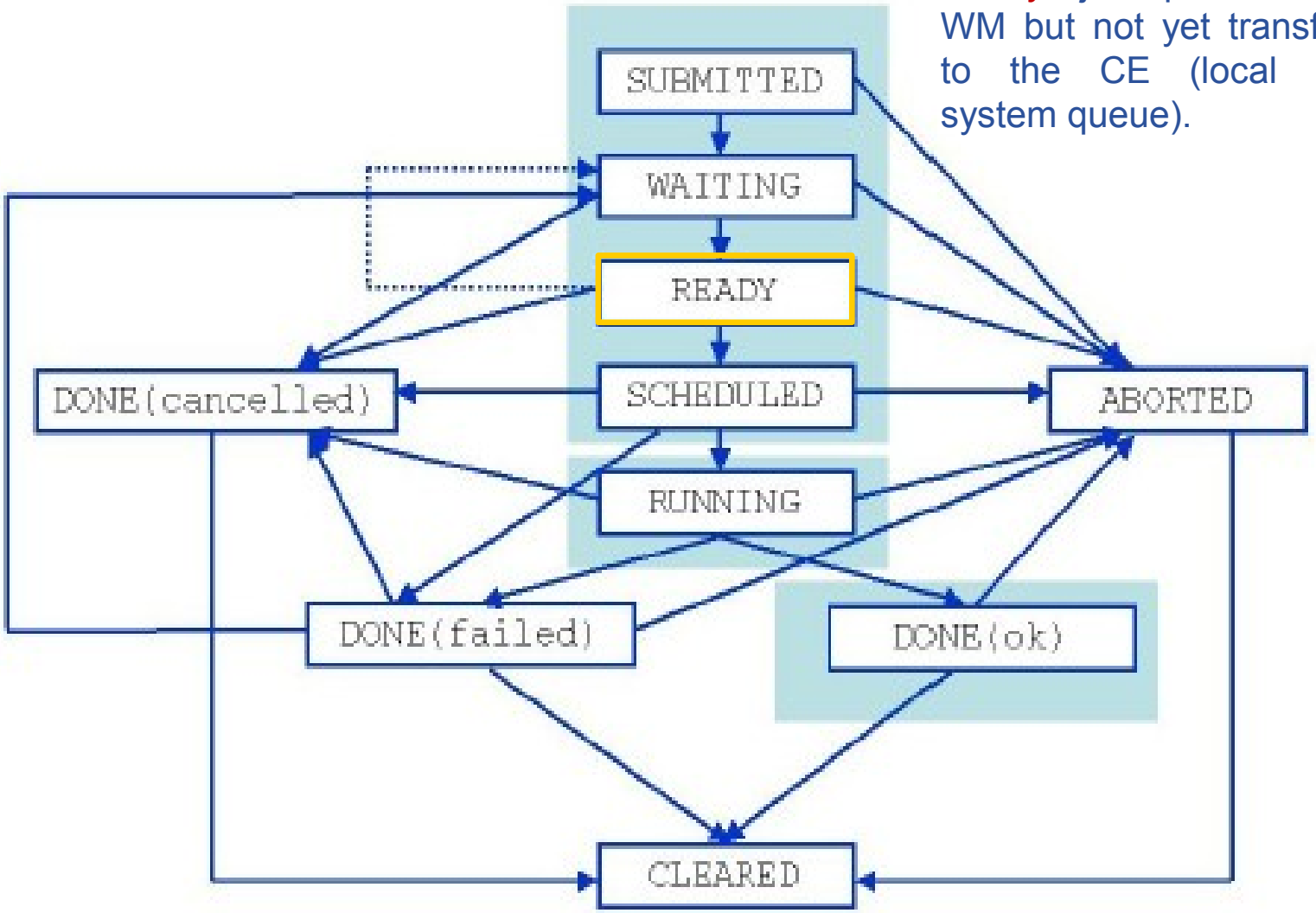
Jobs State Machine (2/9)

Waiting job accepted by NS and waiting for Workload Manager processing.

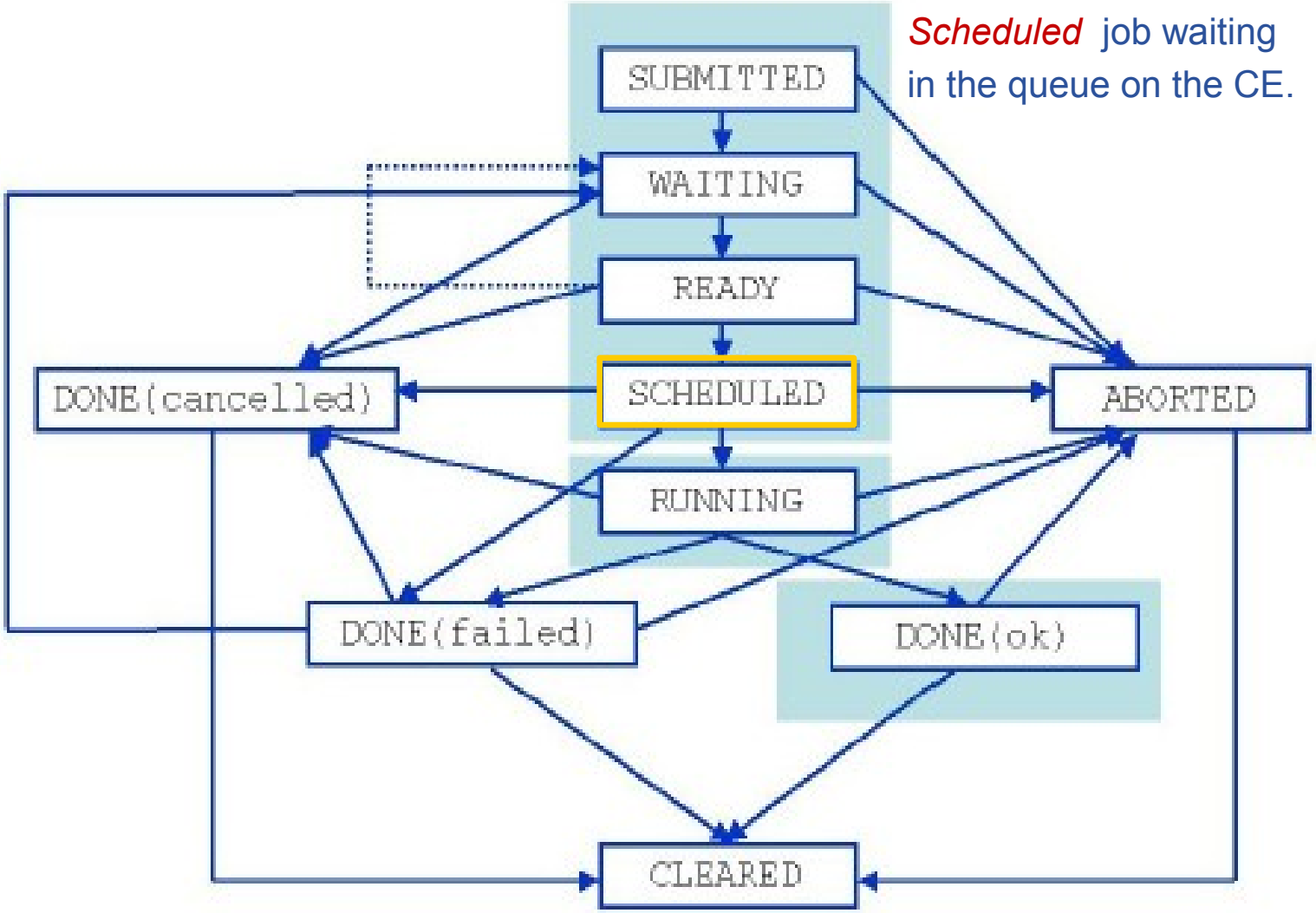


Jobs State Machine (3/9)

Ready job processed by WM but not yet transferred to the CE (local batch system queue).

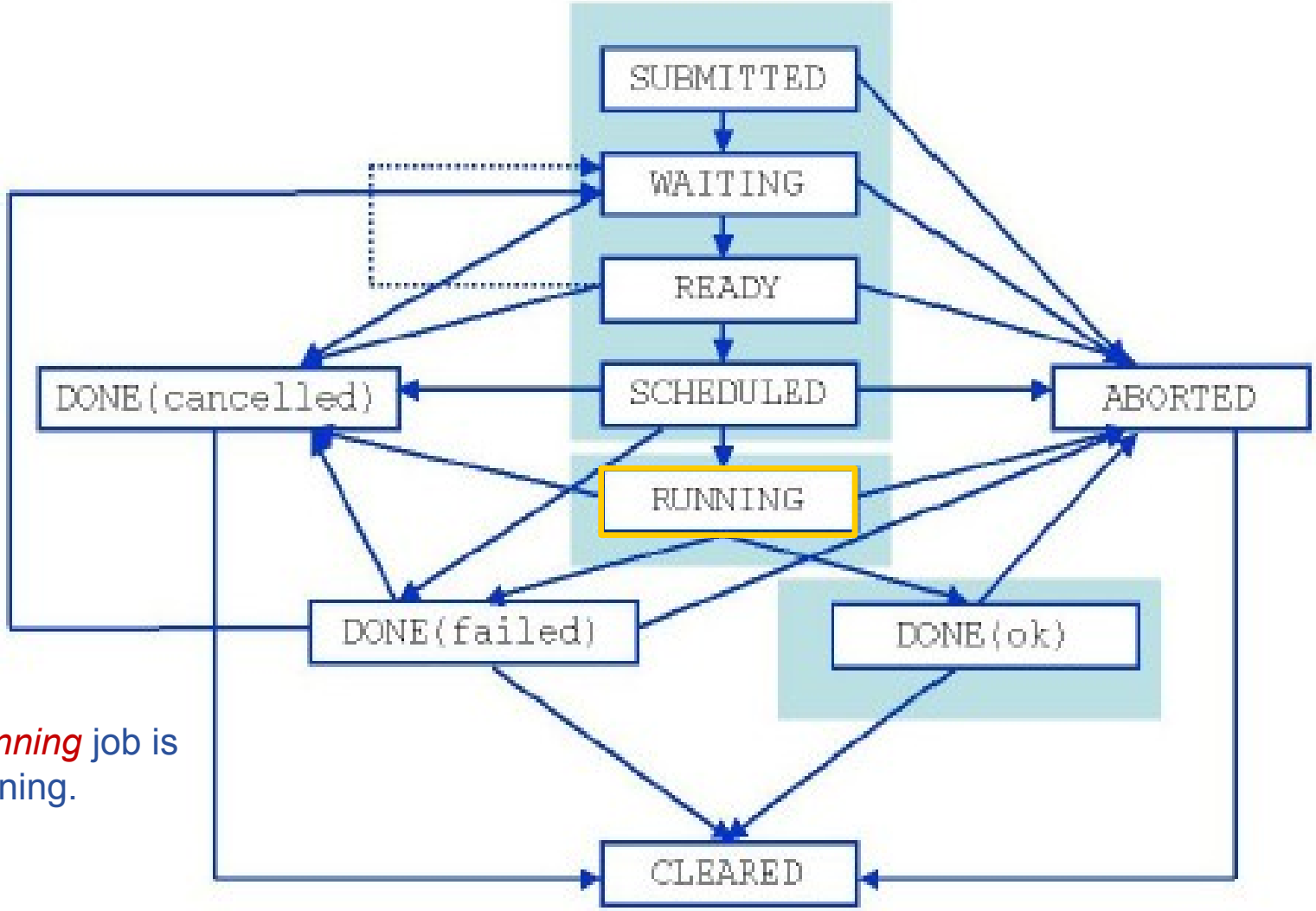


Jobs State Machine (4/9)



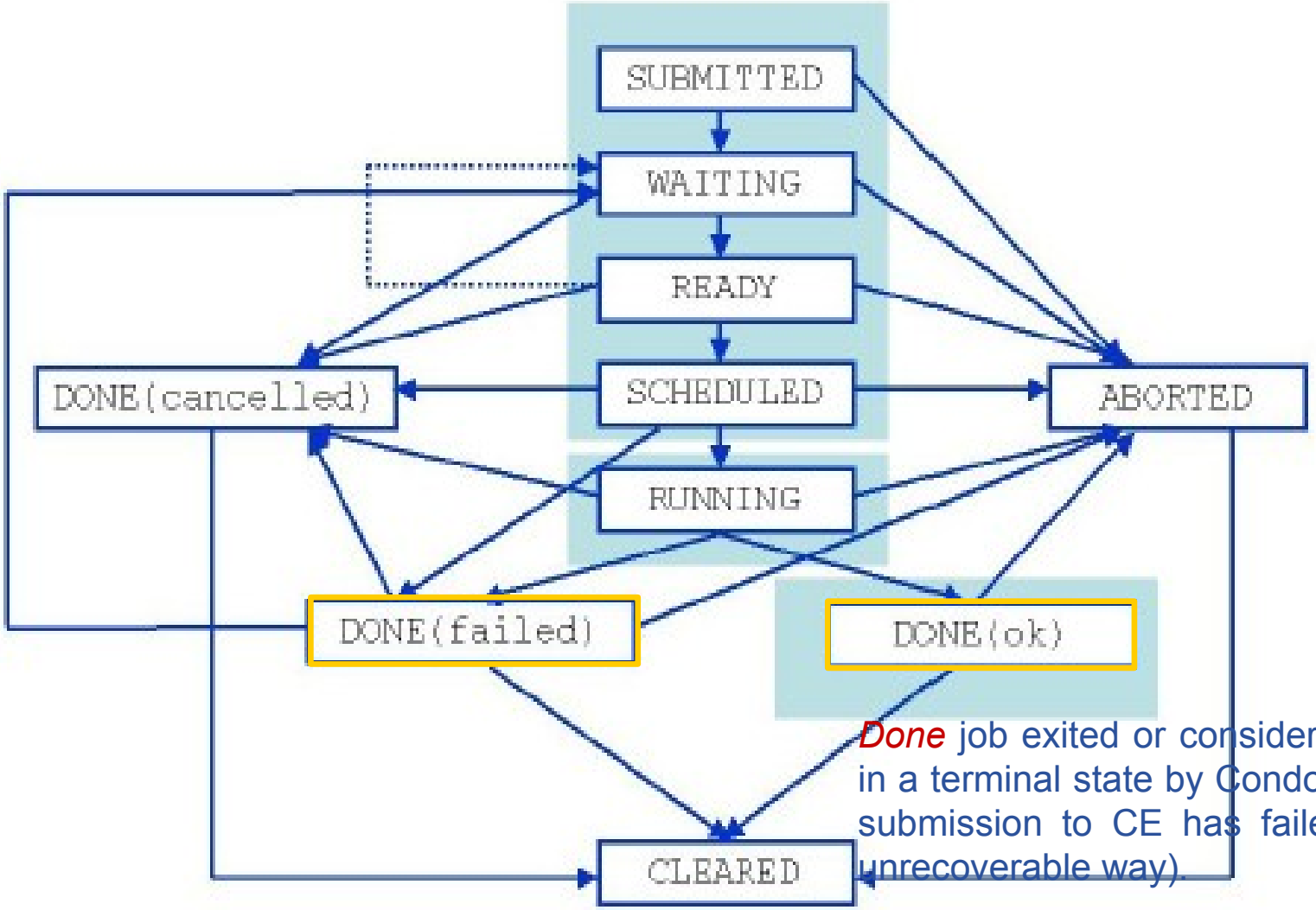
Scheduled job waiting in the queue on the CE.

Jobs State Machine (5/9)



Running job is running.

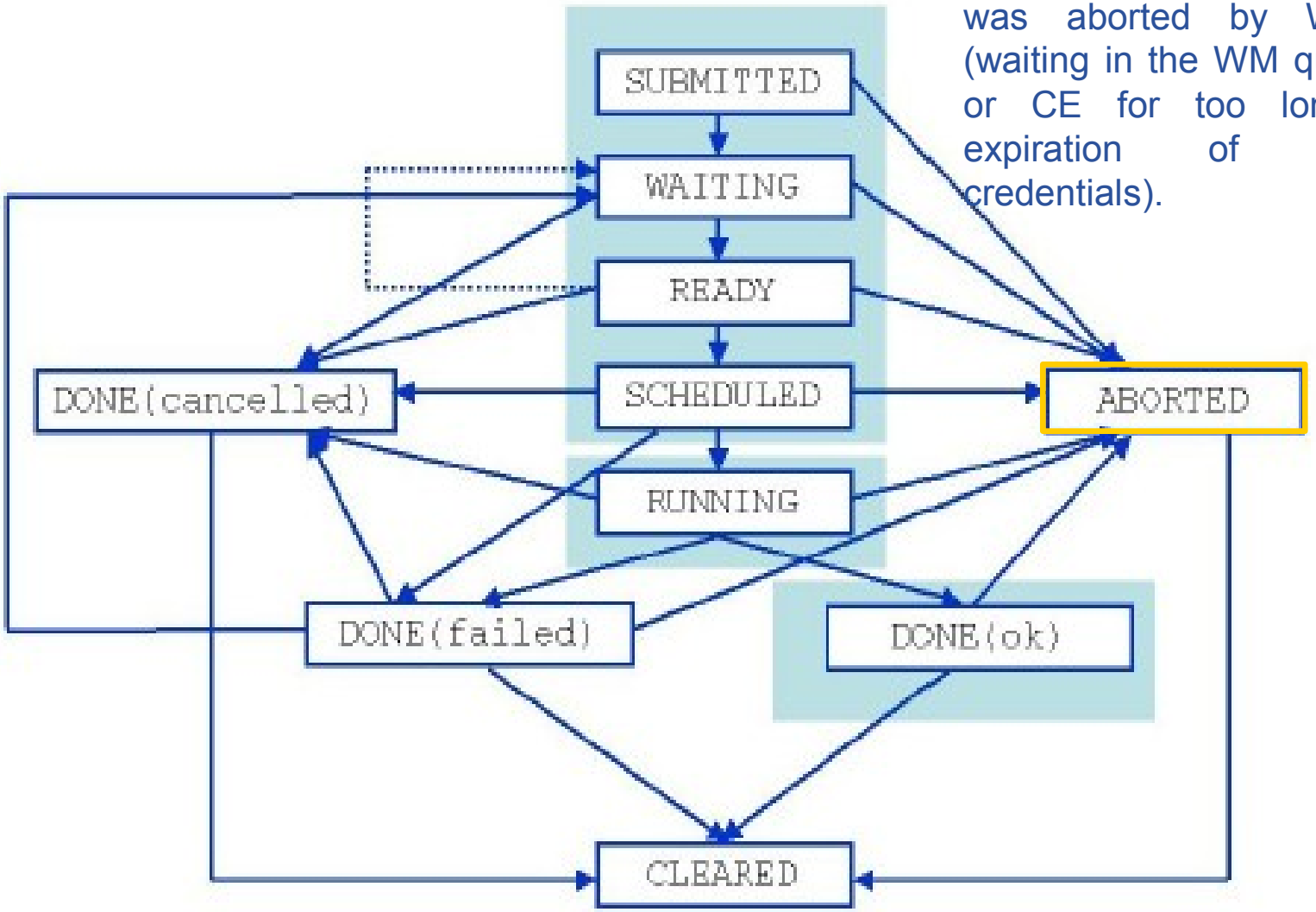
Jobs State Machine (6/9)



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

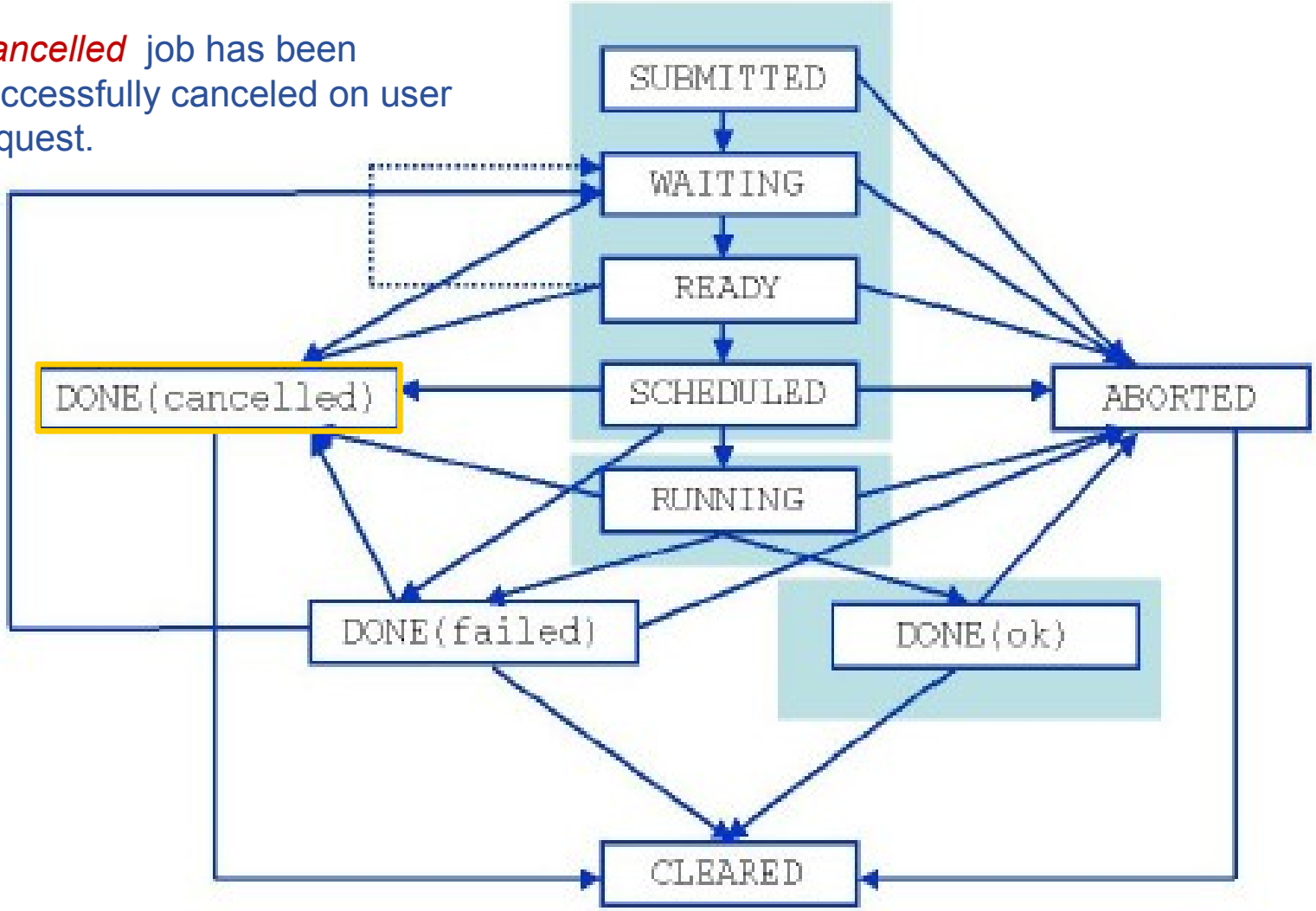
Jobs State Machine (7/9)

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

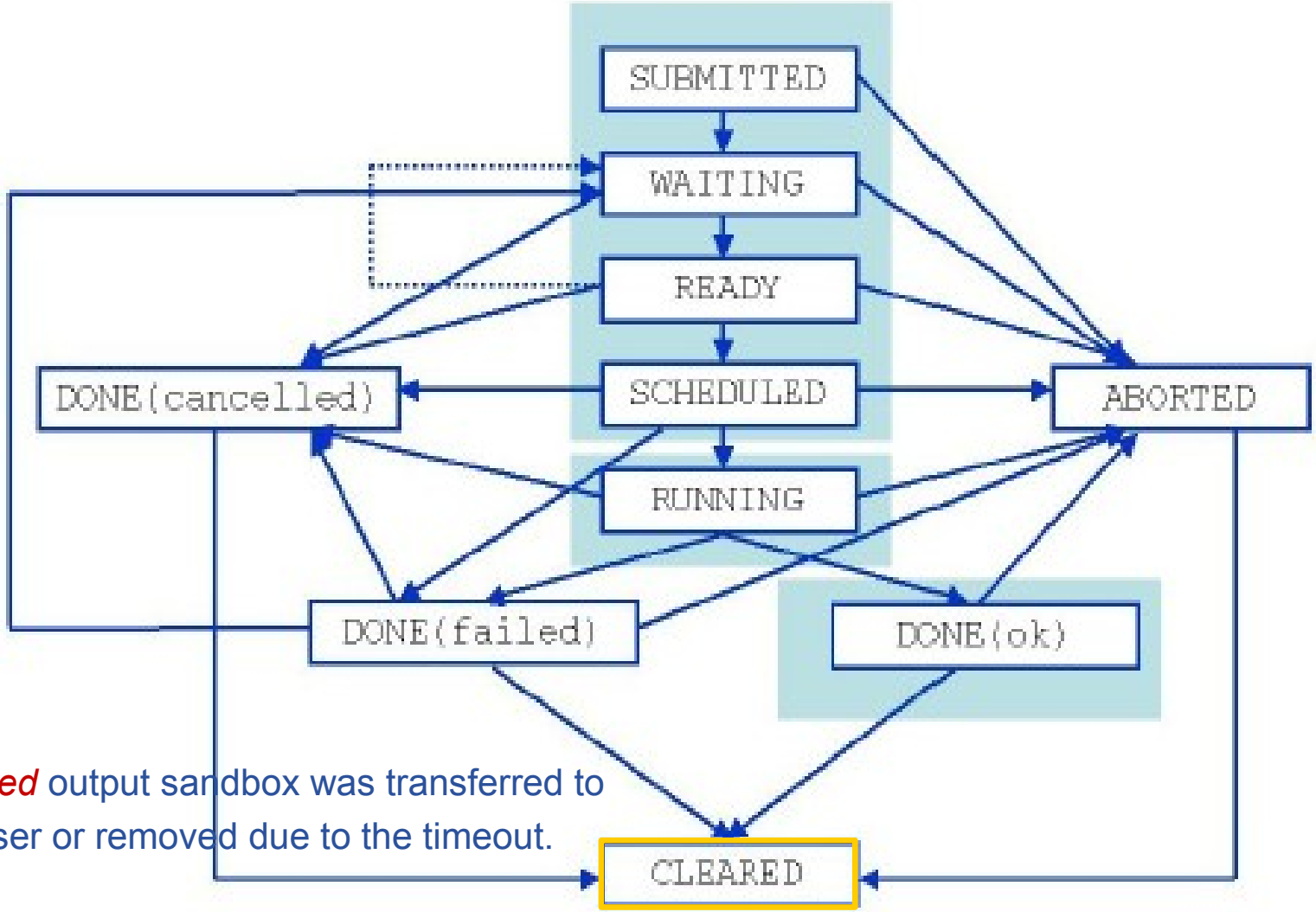


Jobs State Machine (8/9)

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.

Command Line Interface

- Job Submission
  - Perform the job submission to the Grid.

```
$ glite-job-submit  [options] <jdl_file>
```

  - where <jdl file> is a file containing the job description, usually with extension .jdl.

--vo **<vo name> : perform submission with a different VO than the UI default one.**

--output, -o **<output file> save jobId on a file.**

--resource, -r **<resource value> specify the resource for execution.**

--nomsgi **neither message nor errors on the stdout will be displayed.**

If the request has been correctly submitted this is the tipical output that you can get:

glite-job-submit test.jdl

==========glite-job-submit Success ==========

The job has been successfully submitted to the Network Server.

Use glite-job-status command to check job current status.

Your job identifier (edg_jobId) is:

- https://lxshare0234.cern.ch:9000/rIBubkFFKhnSQ6CjiLUY8Q

==========

In case of failure, an error message will be displayed instead, and an exit status different form zero will be retured.

If the command returns the following error message:

****** Error: API_NATIVE_ERROR ******
Error while calling the "NSClient::multi" native api
AuthenticationException: Failed to establish security context...
****** Error: UI_NO_NS_CONTACT ******
Unable to contact any Network Server

it means that there are authentication problems between the UI and the *Network Server* (check your proxy or contact the site administrator).

It is possible to see which CEs are eligible to run a job specified by a given JDL file using the command

glite-job-list-match test.jdl

Connecting to host lxshare0380.cern.ch, port 7772

Selected Virtual Organisation name (from UI conf file): dteam

****

COMPUTING ELEMENT IDs LIST

The following CE(s) matching your job requirements have been found:

adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite

adc0015.cern.ch:2119/jobmanager-lcgpbs-long

adc0015.cern.ch:2119/jobmanager-lcgpbs-short

****

After a job is submitted, it is possible to see its status using the glite-job-status command.

glite-job-status https://lxshare0234.cern.ch:9000/X-ehTxfdlXxSoIdVLS0L0w

****

BOOKKEEPING INFORMATION:

Printing status info for the Job:

https://lxshare0234.cern.ch:9000/X-ehTxfdlXxSoIdVLS0L0w

Current Status: Scheduled

Status Reason: unavailable

Destination: lxshare0277.cern.ch:2119/jobmanager-pbs-infinite

reached on: Fri Aug 1 12:21:35 2003

****

The option -i <file path> can be used to specify a file with a list of job identifiers (saved previously with the -o option of glite-job-submit).

```
glite-job-status -i jobs.list
------------------------------------------------------------------------------------------------
1 : https://lxshare0234.cern.ch:9000/UPBqN2s2ycxt1TnuU3kzEw
2 : https://lxshare0234.cern.ch:9000/8S6IwPW33AhyxhkSv8Nt9A
3 : https://lxshare0234.cern.ch:9000/E9R0Yl4J7qgsq7FYTnhmsA
a : all
q : quit
------------------------------------------------------------------------------------------------
Choose one or more edg_jobId(s) in the list - [1-3]all:
```

If the - -all option is used instead, the status of all the jobs owned by the user submitting the command is retrieved.

The --status <state> (-s) option makes the command retrieve only the jobs that are in the specified state, and the --exclude

<state> (-e) option makes it retrieve jobs that are not in the specified state.

This two lasts options are mutually exclusive, although they can be used with --from and --to.

Example: All jobs of the user that are in the state DONE or RUNNING are retrieved.

```
glite-job-status --all -s Done -s Running
```

A job can be canceled before it ends using the command glite-job-cancel.

glite-job-cancel https://lxshare0234.cern.ch:9000/dAE162is6EStca0VqhVkog

Are you sure you want to remove specified job(s)? [y/n]n :y

========== glite-job-cancel Success==========

The cancellation request has been successfully submitted for the following job(s)

- https://lxshare0234.cern.ch:9000/dAE162is6EStca0VqhVkog

==========

After the job has finished (it reaches the DONE status), its output can be copied to the UI

```
glite-job-output https://lxshare0234.cern.ch:9000/snPegp1YMJcnS22yF5pFlg

Retrieving files from host lxshare0234.cern.ch
************************************************************
JOB GET OUTPUT OUTCOME
Output sandbox files for the job:
- https://lxshare0234.cern.ch:9000/snPegp1YMJcnS22yF5pFlg
have been successfully retrieved and stored in the directory:
/tmp/glite/glite-ui/snPegp1YMJcnS22yF5pFlg
************************************************************
```

By default, the output is stored under /tmp, but it is possible to specify in which directory to save the output using the - -dir <path name> option.

Second Part

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.*

- 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.

- 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!”;
```

- If the argument 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"**;)

- The *backtick* character ` cannot be specified in the JDL.

- 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);

- *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, Parametric*
  - Or combination of them
    - Checkpointable, Interactive
    - Checkpointable, MPI

E.g. **JobType = "Interactive";**

JobType = {"Interactive","Checkpointable"};

"Interactive" + "MPI" not yet permitted

- **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"};

- **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”;

- **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"};**

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

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

- **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);

Rank (optional)

- Floating-point expression used to rank 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.

E.g.: **Rank = *other.GlueCEStateFreeCPUs;***

- **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”};

- **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”};

- **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 = "aliserv6.ct.infn.it";

- **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***

- **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.

Automatic uploading mechanism NOT yet supported in gLite

- **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;
```

- **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”};
```

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

```
CurrentStep = 2;
```

Exercise 1

Run an *ls* command on a resource

Create or modify ls.jdl and ls.sh as follow:

```
[
  Executable = "ls.sh";
  Arguments = "-al";
  StdError = "stderr.log";
  StdOutput = "stdout.log";
  InputSandbox = "ls.sh";
  OutputSandbox = {"stderr.log", "stdout.log"};
]
```

Create and make executable with chmod +x ls.sh script

```
#!/bin/sh
/bin/ls $1
```

Exercise 2

Create a C program as follow:

```
#include <stdio.h>
int main(int argc, char **argv) {
   FILE *fileinput;
   int numero,somma=0;

   if(( fileinput = fopen( "input.dat" , "r" )) == NULL ){
       printf( "Errore apertura file input.dat \n" ) ;
       return 1;
     }
   while (fscanf(fileinput, "%d",&numero) != EOF) {
    somma=somma+numero;
     }
   fclose(fileinput);
  printf("\n");
  printf("La somma dei numeri = %d",somma);
  return 0;
}
```

Esercizio 3

Compile the C program using the gcc compiler: **gcc –o esempio2.x esempio2.c**

Create “input.dat” as follow:

11

223

45

…

Create start_esempio2.sh in order to run *esempio2.x*

```
#!/bin/sh
chmod 777 esempio2.x
./esempio2.x
```

Exercise 3

Job + Data Management

- Upload your file to the catalog.

  E.g.:

```
lcg-cr –vo gilda
  -l lfn:/grid/gilda/<YOUR_LFN>.dat
  -d aliserv6.ct.infn.it
  file:${HOME}/your_file
```

- Create a bash script as follow (scriptInput.sh):

```
#!/bin/sh
lcg-cp --vo gilda lfn:/grid/gilda/<YOUR_LFN>.dat  file:
  `pwd`/file.dat
echo "Hello $1 and Welcome to S.Margherita di
  Pula!!">>`pwd`/file.dat
cat `pwd`/file.dat
```

Job + Data Management

- Create a JDL file as follow (JobWithInput.jdl):

```
[
   VirtualOrganisation = "gilda";
   Executable = "scriptInput.sh"
   Argument = "your_NAME";
   StdOutput = "std.out";
   StdError = "std.err";
   InputSandbox = "scriptInput.sh";
   OutputSandbox = {"std.out","std.err"};
   DataCatalog = "http://lfc-gilda.ct.infn.it:8085";
   InputData = "lfn:/grid/gilda/<YOUR_LFN>.dat";
   DataAccessProtocol = {"gridftp","rfio","gsiftp"};
   RetryCount = 0;
]
```

Third Part

WMProxy

- WMProxy (Workload Manager Proxy)
  - is a new service providing access to the gLite Workload Management System (WMS) functionality through a simple Web Services based interface.
    - has been designed to handle a large number of requests for job submission
      - gLite 1.5 => ~180 secs for 500 jobs
      - Goal is to get in the short term to ~60 secs for 1000 jobs
    - it provides additional features such as *bulk submission* and the support for *shared and compressed* sandboxes for *compound jobs*.
    - It's the natural replacement of the NS in the passage to the *SOA approach*.

Authorization

- **The client must be properly authorized when interacts with the WMProxy service.**
  - **This means that either the FQAN or the DN (in case of globus-style proxies) of the client must be properly listed and authorized in the *glite_wms_wmproxy.gacl* file on the WMProxy machine.**

- Each job submitted to a WMProxy Service is given the *delegated of the user 's credentials* who submitted it.
- These credentials can then be used to perform operation that require interaction with other services
  - (e.g. submission to the CE, a GridFTP file transfer etc.)
  - There are two possible mechanism to ask for a delegation of the user credentails:
    - asking the "automatic" delegation of the credentials during the submission operation
    - asking for an "explicit" delegation

WMProxy client tools

WMProxy can be accessed through:

- published WSDL
  - Developers can generate themselves client stubs in their favourite language from the published WSDL
- C++/Java/Python API
  - Light client libraries generated using respectively gSoap, Axis and SOAPpy
  - Hides WSDL/SOAP tooling dirty details
  - Python API available starting from gLite 1.5

WMProxy C++ client commands

The commands to interact with WMProxy Service are:

glite-wms-delegate-proxy <delegation_Id>

glite-wms-job-submit <jdl_file>

glite-wms-job-list-match <jdl_file>

glite-wms-job-cancel <job_Ids>

glite-wms-job-output <job_Ids>

WMProxy client commands (cont.)

- glite-wms-delegate-proxy **allows the user to delegate his proxy credential to the WMProxy service.This delegated credential can then be used for job submissions.**
- glite-wms-job-submit **submits a job to a WMProxy Service. It requires a JDL file as input and returns a WMS job identifier.**
- glite-wms-job-list-match **lists the available resources where the job can be submitted.**
- glite-wms-job-cancel **cancels one or more jobs previously submitted to WMProxy Service.**
- glite-wms-job-output **Retrieve output files of a job, when finished.**

glite-wms-job-delegate-proxy [options]

options:

```
--version
--help
--config, -c <configfile>
--vo <voname>
--debug
--logfile <filepath>
--noint
--delegationid, -d <idstring>
--autm-delegation, -a
--endpoint, -e <serviceURL>
--output, -o <filepath>
```

if specified, the operations on the WMProxy will be associated to the credential previously delegated with the *idstring* delegation string.

New Features

Shared Sandboxes

for sub-jobs of compound jobs

- JDL has been extended to allow specification of the input sandbox at the level of the compound request (i.e. DAGs, Collections and Parametric jobs)
  - This Input sandbox is trasferred only once by the new WMS client commands but can be accessed by all sub-jobs of the compound job
  - Each sub-jobs sandboxes can refers to a single files of the “shared sandbox”, e.g.

```
InputSandbox = root.InputSandbox[0];
```

  - or to sandboxes of other sub-jobs, e.g.,

```
InputSandbox = root.nodes.nodeA.description.OutputSandbox[2];
```

‘Scattered’ Input Sandboxes

- Input Sandbox can contain
  - file paths on the UI machine (i.e. the usual way)
  - URI pointing to files on a remote gridFTP/HTTPS server

```
InputSandbox = {
  "gsiftp://neo.datamat.it:2811/var/prg/sim.exe",
  "https://ghemon.cnaf.infn.it:8443/data/idat_1",
  "file:///home/pacio/myconf"};
```

- A base URI to be applied to all sandbox files can also be specified

```
InputSandboxBaseURI =
  “gsiftp://matrix.datamat.it:2811/var";
```

‘Scattered’ Output Sandboxes

- JDL has been enriched with new attributes for specifying the destinations for the files listed in the OutputSandbox attribute list

```
OutputSandbox = {       "jobOutput",
                        "run1/event1",
                        "jobError"
                };

OutputSandboxDestURI = {
  "gsiftp://matrix.datamat.it/var/jobOutput",
  "https://grid003.ct.infn.it:8443/home/cms/event1",
  "gsiftp://matrix.datamat.it/var/jobError"};
```

- A base URI to be applied to all sandbox files can also be specified

```
OutputSandboxBaseDestURI =
  "gsiftp://neo.datamat.it/home/run1/";
```

‘Compressed’ Sandboxes

(gLite version >= 1.5)

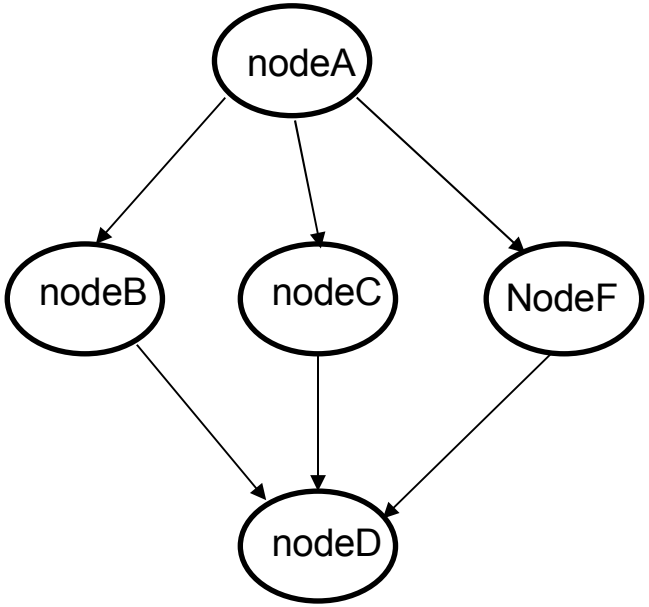
- A compressed archive is created with the input sandboxes files using libtar and zlib libraries
  - This is done automatically by WMProxy client commands
  - This mechanism can be enabled/disabled by the user through the JDL (*AllowZippedISB* attribute)
- The archive is transferred (instead of single files) to the WMS
- WMProxy service *untars* the files in the jobs directories when the job is ‘started’ and removes the archive

New type of request

DAG job

- DAG is a set of jobs where the input, output, or execution of one or more jobs depends on one or more other ones
  - The jobs are nodes (vertices) in the graph
  - the edges (arcs) identify the dependencies

  - Their management has been improved with
    - Shared sandboxes
    - Attributes Inheritance
    - Attribute references between nodes and with the ‘parent’



DAG jdl

```
[
  type = "dag";
  max_nodes_running = 4;
  nodes = [
    nodeA = [
      file ="nodes/nodeA.jdl" ;
    ];
    nodeB = [
      file ="nodes/nodeB.jdl" ;
    ];
    nodeC = [
      file ="nodes/nodeC.jdl" ;
    ];
    nodeF = [
      file ="nodes/nodeF.jdl";
    ];
   dependencies = {
      {nodeA, nodeB},
      {nodeA, nodeC},{nodeA, nodeF},
      { {nodeB,nodeC,nodeF}, nodeD }
    }
  ];
;
]
```

WMProxy : submission & monitoring

- In order to submit job with WMProxy, it's mandatatory to use credentials delegation

```
glite-wms-job-delegate-proxy -d del_ID_01
```

- The submission/monitoring commands are slightly different, but most of "old" options are supported

```
glite-wms-job-submit -d del_ID_01 myjob.jdl

glite-wms-job-status \
https://glite-rb.ct.infn.it:9000/LHIIGaCVdl7Olm
sz0jpI_g

glite-wms-job-output \
https://glite-rb.ct.infn.it:9000/LHIIGaCVdl7Olm
sz0jpI_g
```

Job Collection

- Job collection is a set of independent jobs that user can submit and monitor as it was a single job
- Jobs of a collection are submitted as DAG nodes, without dependencies
- The JDL is a list of ClassAds which describe the subjobs

```
[
        Type = "collection“;
        nodes = {
                [  <job descr 1 >],
                [  <job descr 2 >],
                 …
               };
        ...
 ]
```

Job collection examples

BioinfoGRID

```
[
    Type = “Collection";
    RetryCount = 0;

    nodes={ [
                    Executable = "/bin/hostname";
                    Arguments = “-f";
                    StdOutput = "hostname.out";
                    StdError = "hostname.err";
                    OutputSandbox = {"hostname.err","hostname.out"};
   ] , [
                    Executable = "/bin/sh";
                    Arguments = "start_povray_valve.sh";
                    StdOutput = “povray.out";
                    StdError = “povray.err";
                    InputSandbox = {“start_povray_valve.sh"};
                    OutputSandbox = {“povray.err",“povray.out"};

                    Requirements = Member (“POVRAY-3,5”,
          other.GlueHostApplicationSoftwareRunTimeEnvironment);
   ] };
]
```

Submitting the collection..

- glite-wms-delegate-proxy **allows the user to delegate her proxy credential to the WMProxy service.**
  - `$ glite-wms-job-delegate-proxy -d myWMProxy`
- glite-wms-job-submit **submits a job to a WMProxy Service. It requires a JDL file as input and returns a WMS job identifier.**
  - `$ glite-wms-job-submit -d myWMProxy collection.jdl`

```
$ glite-wms-job-status https://glite-rb.ct.infn.it:9000/XRse9WKW0c9ltFTD6-OSOw

*************************************************************
BOOKKEEPING INFORMATION:
Status info for the Job : https://glite-rb.ct.infn.it:9000/XRse9WKW0c9ltFTD6-OSOw
Current Status:     Done (Success)
Exit code:          0
Status Reason:      Job terminated successfully
Destination:        dagman
Submitted:          Mon Jan 30 12:43:18 2006 CET
*************************************************************

- Nodes information for:
    Status info for the Job : https://glite-rb.ct.infn.it:9000/rgzQISTy6VLev3G8Enjefw
    Current Status:     Done (Success)
    Exit code:          0
    Status Reason:      Job terminated successfully
    Destination:        dgt01.ui.savba.sk:2119/jobmanager-lcgpbs-infinite
    Submitted:          Mon Jan 30 12:43:18 2006 CET
    Parent Job:         https://glite-rb.ct.infn.it:9000/XRse9WKW0c9ltFTD6-OSOw
*************************************************************

    Status info for the Job : https://glite-rb.ct.infn.it:9000/p9qxZlS5yDewQnd7kyN0NA
    Current Status:     Done (Success)
    Exit code:          0
    Status Reason:      Job terminated successfully
    Destination:        dgt01.ui.savba.sk:2119/jobmanager-lcgpbs-infinite
    Submitted:          Mon Jan 30 12:43:18 2006 CET
    Parent Job:         https://glite-rb.ct.infn.it:9000/XRse9WKW0c9ltFTD6-OSOw
*************************************************************
```

- glite-wms-job-output **Retrieve output files of a job, when finished.**
  - **$ glite-wms-job-output --dir ./collection-output https://glite-rb.ct.infn.it:9000/XRse9WKW0c9ltFTD6-OSOw**

```
ll collection-output/

drwxr-xr-x 2 larocca  users  larocca_p9qxZlS5yDewQnd7kyN0NA
drwxr-xr-x 2 larocca  users  larocca_rgzQISTy6VLev3G8Enjefw
```

Parametric Jobs (1/3)

- A parametric job is a job where one or more of its attributes are parametric
- Value of attributes varies according to parameter

```
[
  JobType = "Parametric";
  Executable = "/bin/echo";
  Arguments = "_PARAM_";
  StdOutput = "myoutput_PARAM_.txt";
  StdError = "myerror_PARAM_.txt";
  Parameters = 3;
  ParameterStep = 1;
  ParameterStart = 1;
  OutputSandbox  = {"myoutput_PARAM_.txt"};

]
```

- Job monitoring / managing is always done through an unique jobID, as if the job was single

Parametric Jobs (2/3)

- The Parameters attribute is either an integer representing the upper bound (not included) or the lower bound, in case it is negative, for the values that can be assumed by "_PARAM_").
- The ParameterStart attribute is an integer indicating the initial value to take into account during the creation of a Parametric job.
- The ParameterStep attribute is an integer representing the size of each variation of the parameter.

- The N jobs generated is given by:

  N = (Parameters – ParameterStart)/ParameterStep

Parametric jobs (2/2)

- Parameter can be either integer and string.

```
[
 JobType = "Parametric";
 Executable = "/bin/cat";
 Arguments = "input_PARAM_.txt";
 InputSandbox = "input_PARAM_.txt";
 StdOutput = "myoutput_PARAM_.txt";
 StdError = "myerror_PARAM_.txt";
 Parameters = {EARTH,MOON,MARS};
 OutputSandbox  = {"myoutput_PARAM_.txt"};
]

ls
inputEARTH.txt  inputMARS.txt  inputMOON.txt
```

References

- WSDL documentation
  - http://lxmi.mi.infn.it/egee-jra1-wm/wmproxy
  - http://jra1mw.cvs.cern.ch:8180/cgi-bin/jra1mw.cgi/org.glite.wms.wmproxy-interface/interface/WMProxy.wsdl
- WMProxy User's Guide
  - https://edms.cern.ch/document/674643/1
- JDL Attributes Specification
  - https://edms.cern.ch/document/590869/1
  - http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/api_doc/wms_jdl/index.html
- API documentation
  - http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/glite-wmproxy-api-index.shtml

Questions…

Exercise 1

Create and submit to WMProxy the following parametric's job

```
[
  JobType = "Paramatric";
  Executable = "/bin/echo";
  Arguments = "_PARAM_";
  StdError = "stderr_PARAM_.dat";
  StdOutput = "stdout_PARAM_.dat";
  Parameters = 3;
  ParameterStep = 1;
  ParameterStart = 1;
  OutputSandbox = {"stderr_PARAM_.dat",
                   "stdout_PARAM_.dat"};
]
```

Exercise 2

```
[
  JobType = "Paramatric";
  Executable = "/bin/cat";
  Arguments = "input_PARAM_.txt";
  StdError = "stderr_PARAM_.dat";
  StdOutput = "stdout_PARAM_.dat";
  Parameters = {EARTH,MOON,MARS};
  InputSandbox = {"input_PARAM_.txt"};
  OutputSandbox = {"stderr_PARAM_.dat",
                   "stdout_PARAM_.dat"};
]
```

Create the files inputEARTH.txt, inputMOON.txt and inputMARS.txt as follow:

echo "Welcome to XXX" > inputXXX.txt where XXX=EARTH, MOON, MARS

Exercise 3

Submit the following collection's job

```
[
  Type = "collection";
  InputSandbox = {
    "input_common.txt",
    "inputEARTH.txt"
  };
  Requirements = !
   RegExp("gilda01.ihep.ac.cn",other.GlueCEUniqueId);
  nodes = {
   [
     JobType = "Normal";
     NodeName = "nodo1";
     Executable = "/bin/sh";
     Arguments = "script_node1.sh";
     InputSandbox = {"script_node1.sh",
                       root.InputSandbox
                    };
     StdOutput = "myoutput1";
     StdError  = "myerror1";
     OutputSandbox = {"myoutput1","myerror1"};
   ],
```

```
[
      JobType = "Normal";
      NodeName = "nodo2";
      Executable = "/bin/sh";
      InputSandbox = {"script_node2.sh",
                      root.InputSandbox
                     };
      Arguments = "script_node2.sh";
      StdOutput = "myoutput2";
      StdError  = "myerror2";
      OutputSandbox = {"myoutput2","myerror2"};
    ]
 };
]
```

Create script_node1.sh and script_node2.sh as follow:

```
#!/bin/sh
hostname -f
```

Exercise 4

Submit DAG.jdl from gLite/DAG/identity_ex/ folder

```
[
 type = "dag";
 max_nodes_running = 4;
 nodes = [
      nodeD = [
          file = "nodes/nodeD.jdl";
      ];
      nodeC = [
         file = "nodes/nodeC.jdl";
      ];
      nodeB = [
          file = "nodes/nodeB.jdl";
      ];
      nodeA = [
         file = "nodes/nodeA.jdl";
      ];
      dependencies = {{nodeB, nodeC},{nodeA, nodeB}}
 ];
```