



BioinfoGRID

Bioinformatics Grid Application for life science



Architecture of the gLite Workload Management System

Giuseppe LA ROCCA

INFN Catania

giuseppe.larocca@ct.infn.it



dkfz.





This presentation will cover the following arguments:

➤ **Overview of WMS Architecture**

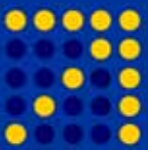
- Task Queue, Information Supermarket, MatchMaker, Scheduling Policies, Job Submission Service, Job Logging & Bookkeeping.

➤ **Job Description Language Overview**

- Principal Attributes

➤ **Overview of WMPProxy**

- New features
- New request types

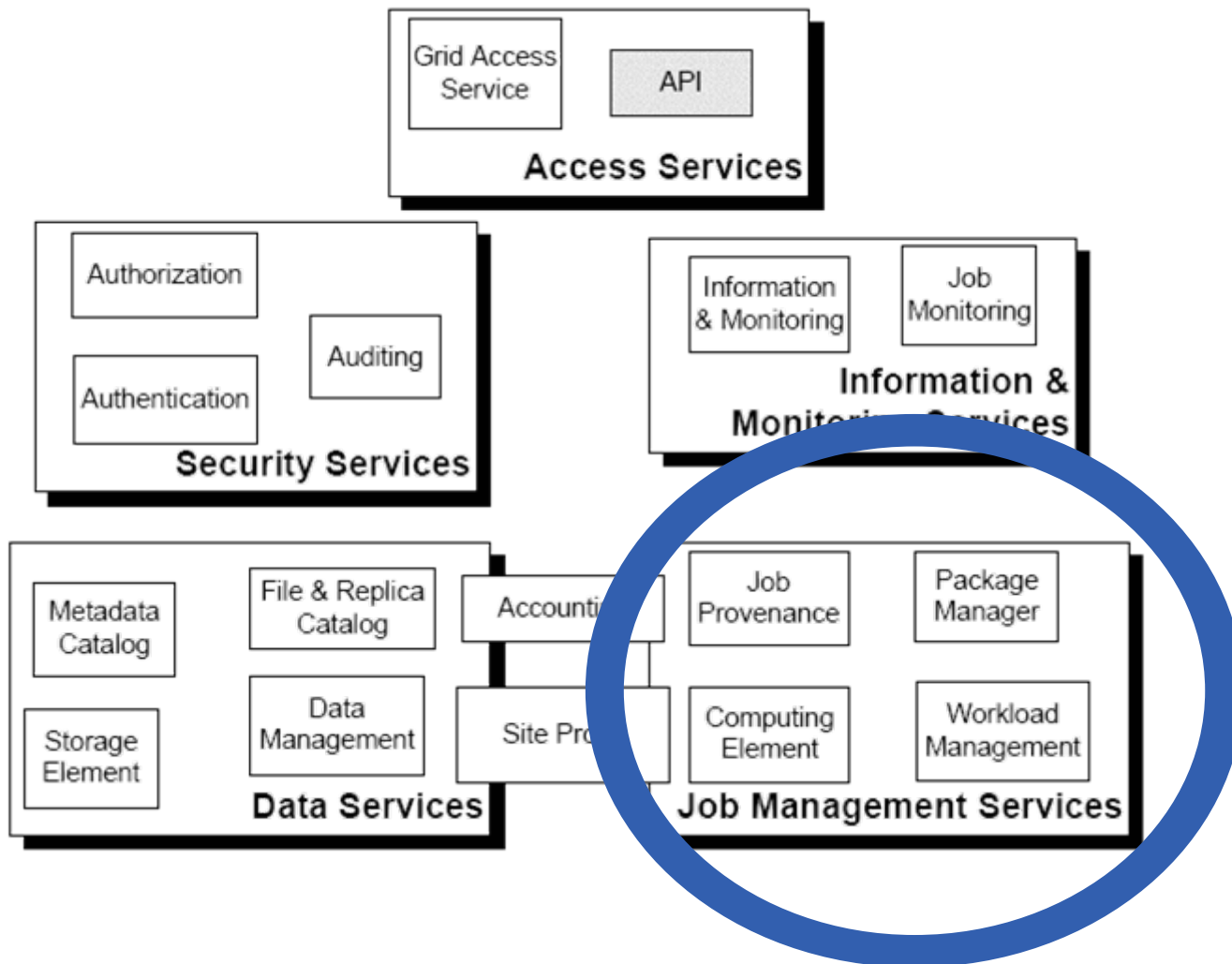


First Part

Overview of the gLite WMS



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.

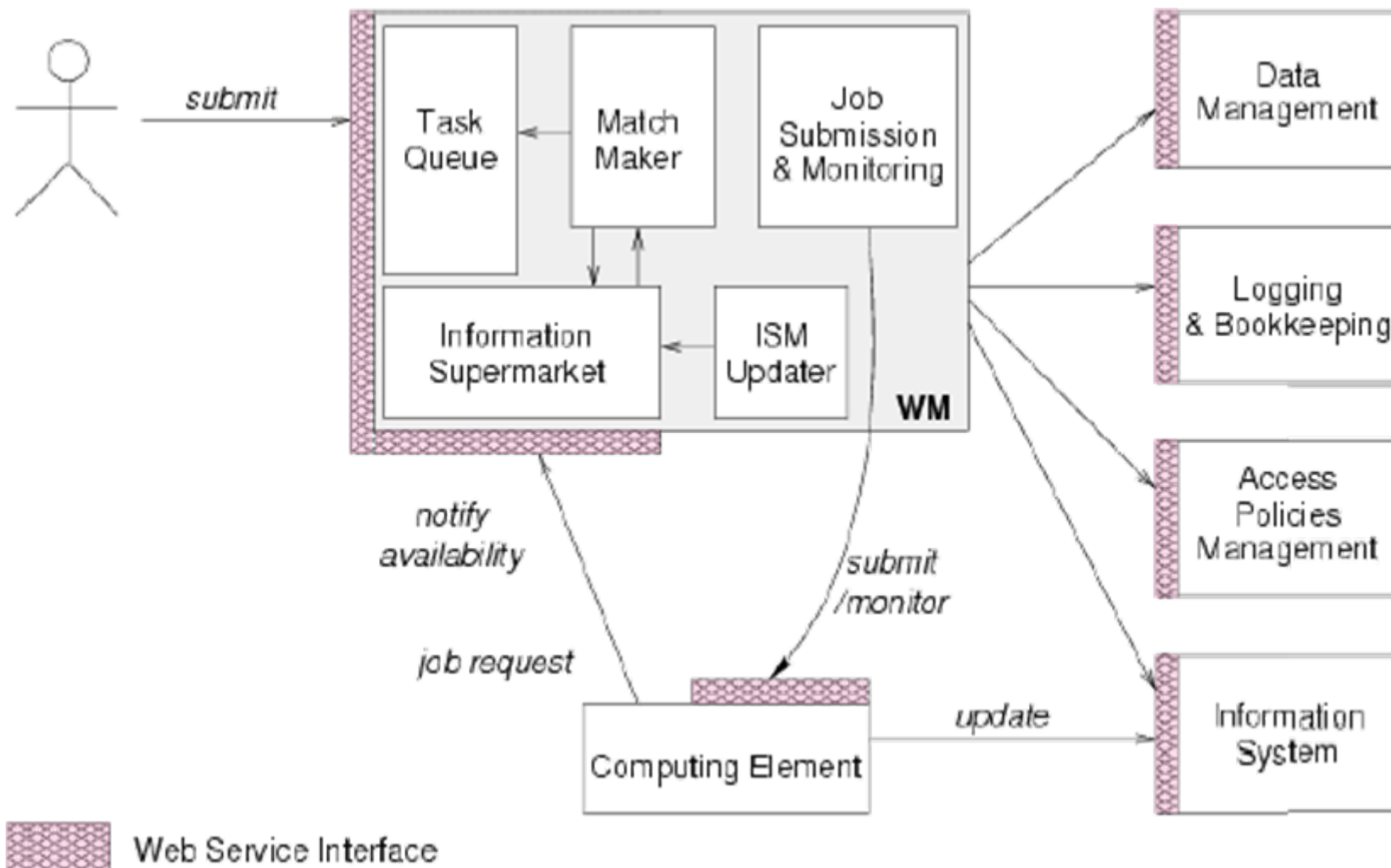
Purpose of **Workload Manager (WM)** is accept and satisfy requests for job management coming from its clients meaning of the submission request is to pass the responsibility of the job to the WM.

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.

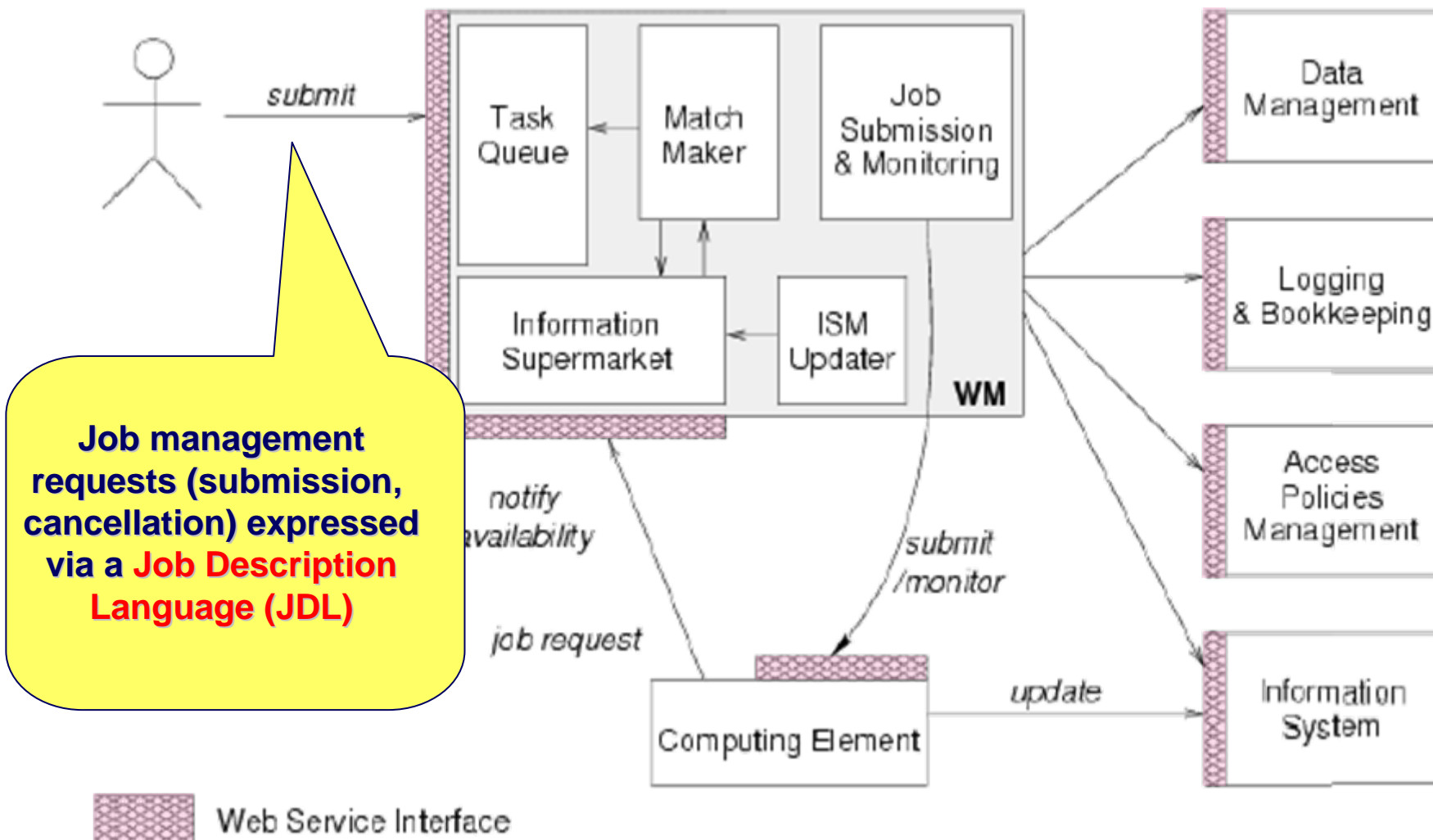


WMS Architecture



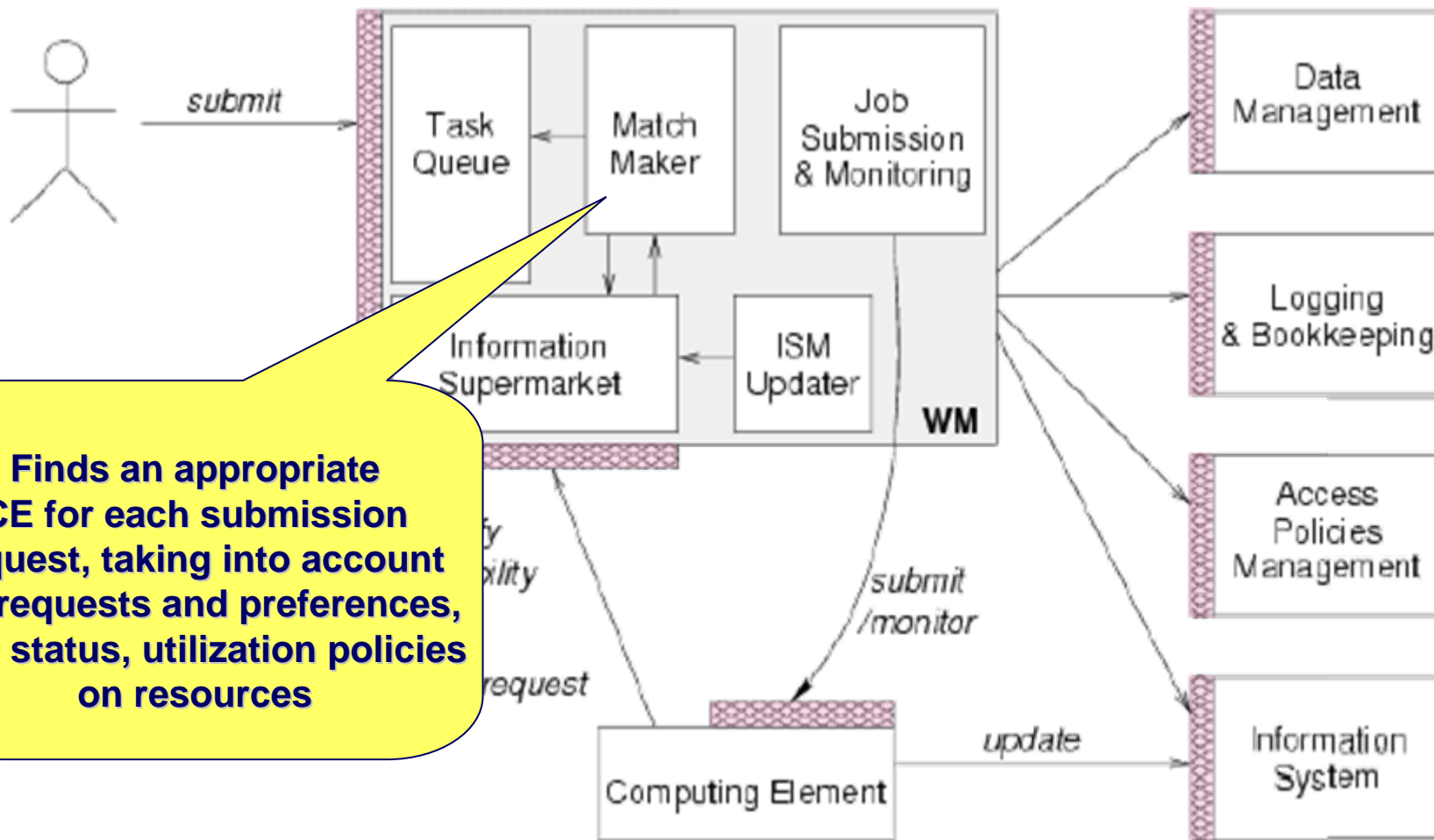


WMS Architecture





WMS Architecture

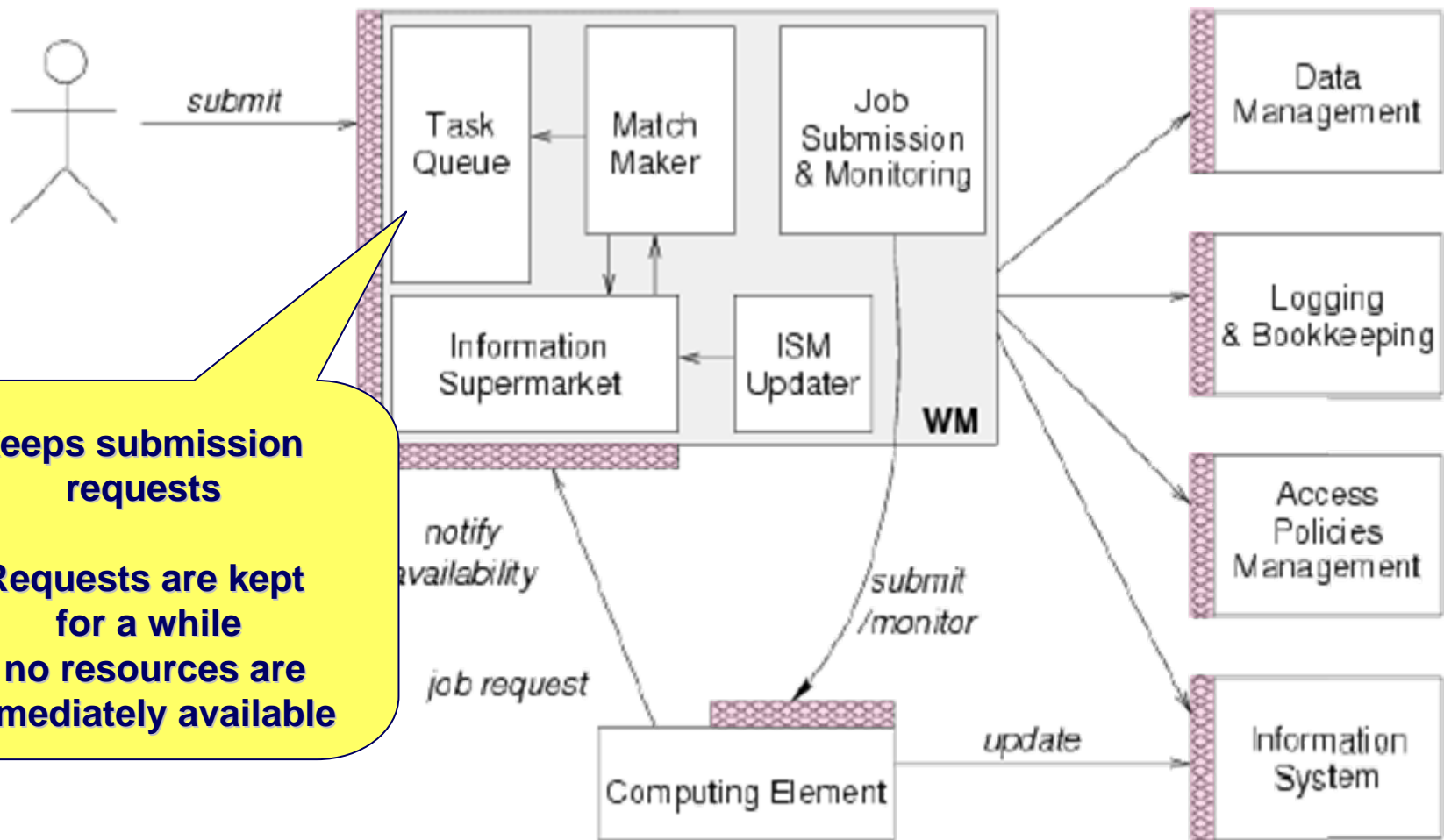


Finds an appropriate CE for each submission request, taking into account job requests and preferences, Grid status, utilization policies on resources

 Web Service Interface




WMS Architecture



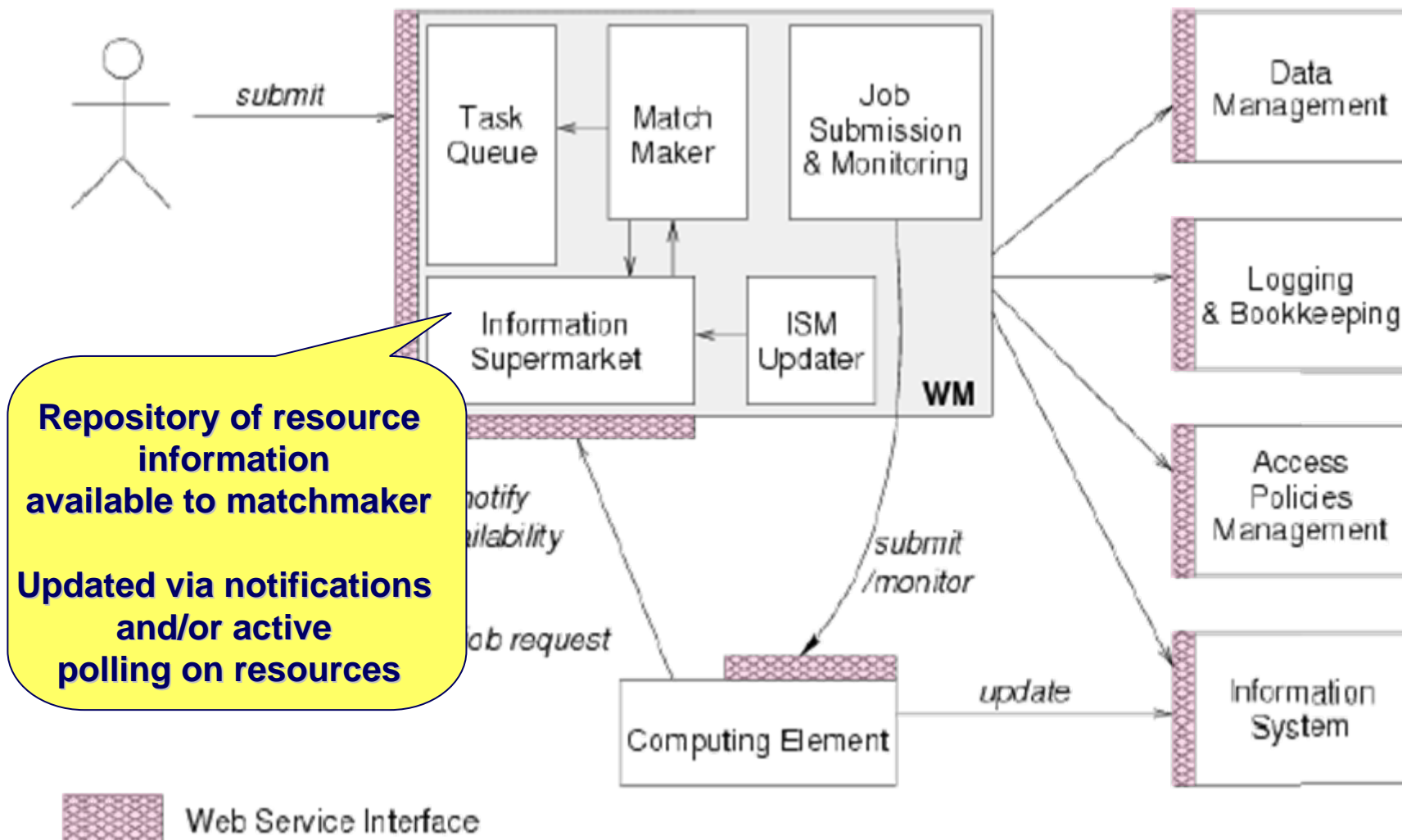
Keeps submission requests

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

 Web Service Interface

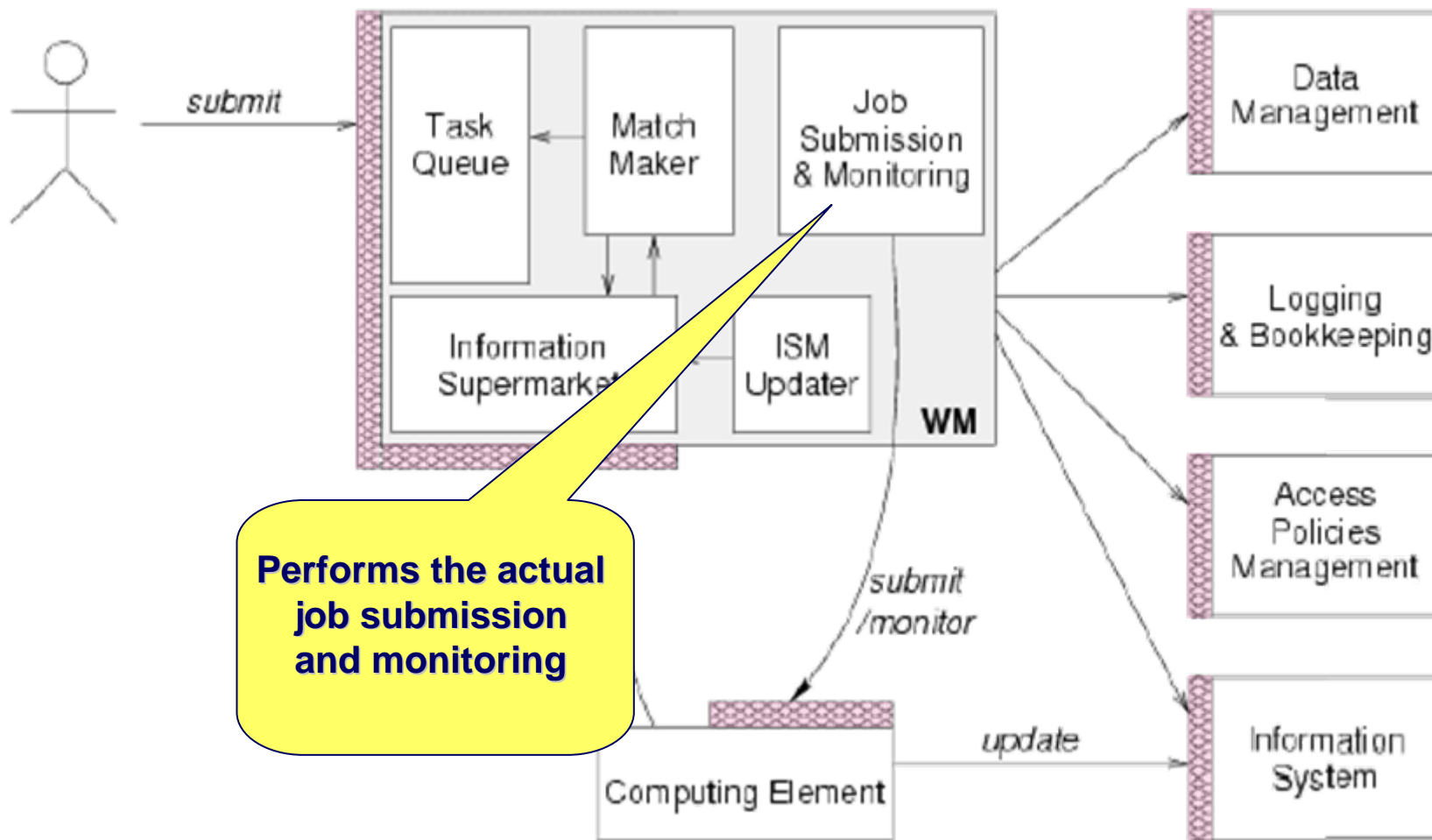


WMS Architecture





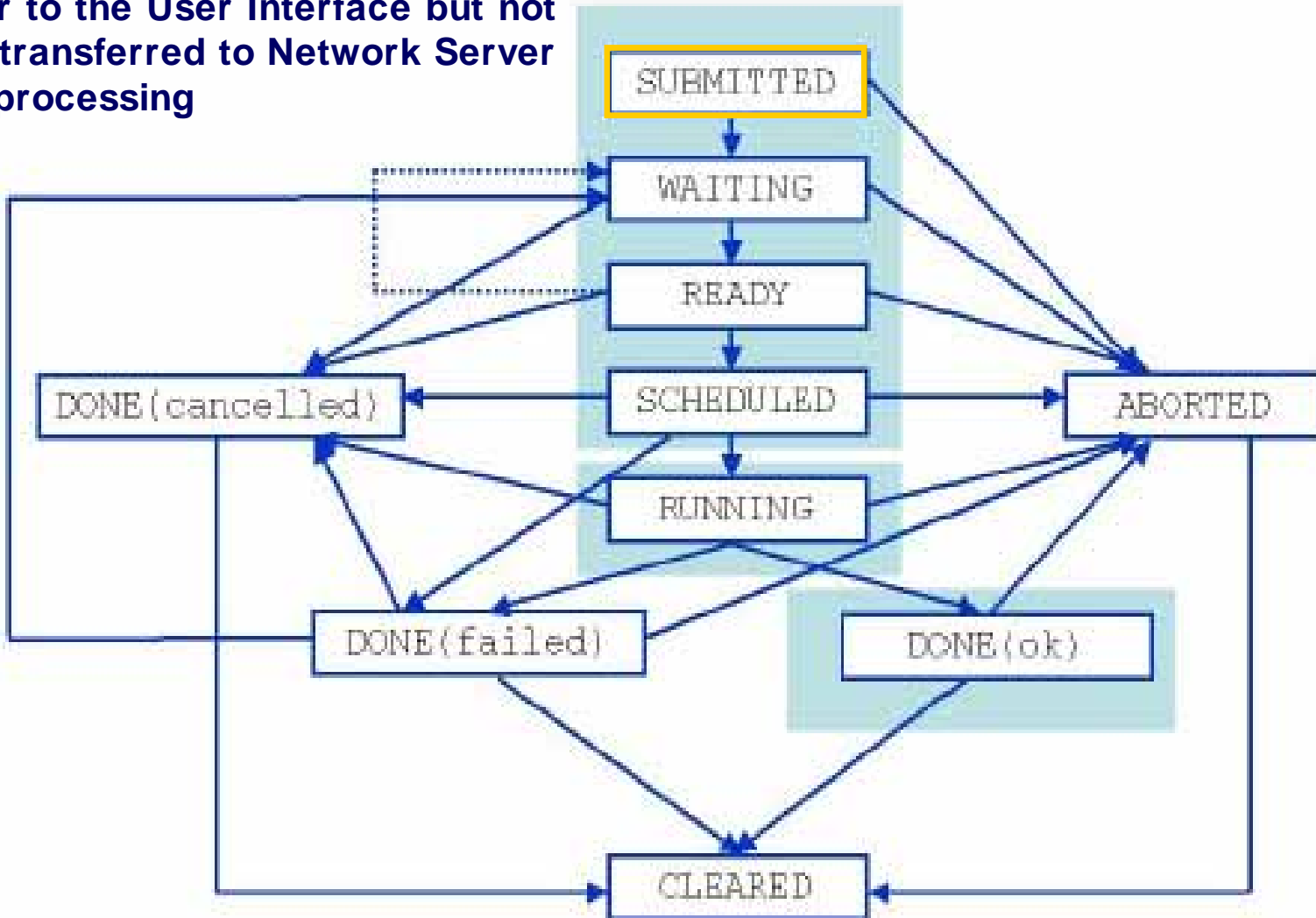
WMS Architecture

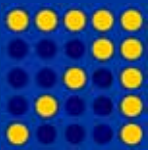


Web Service Interface



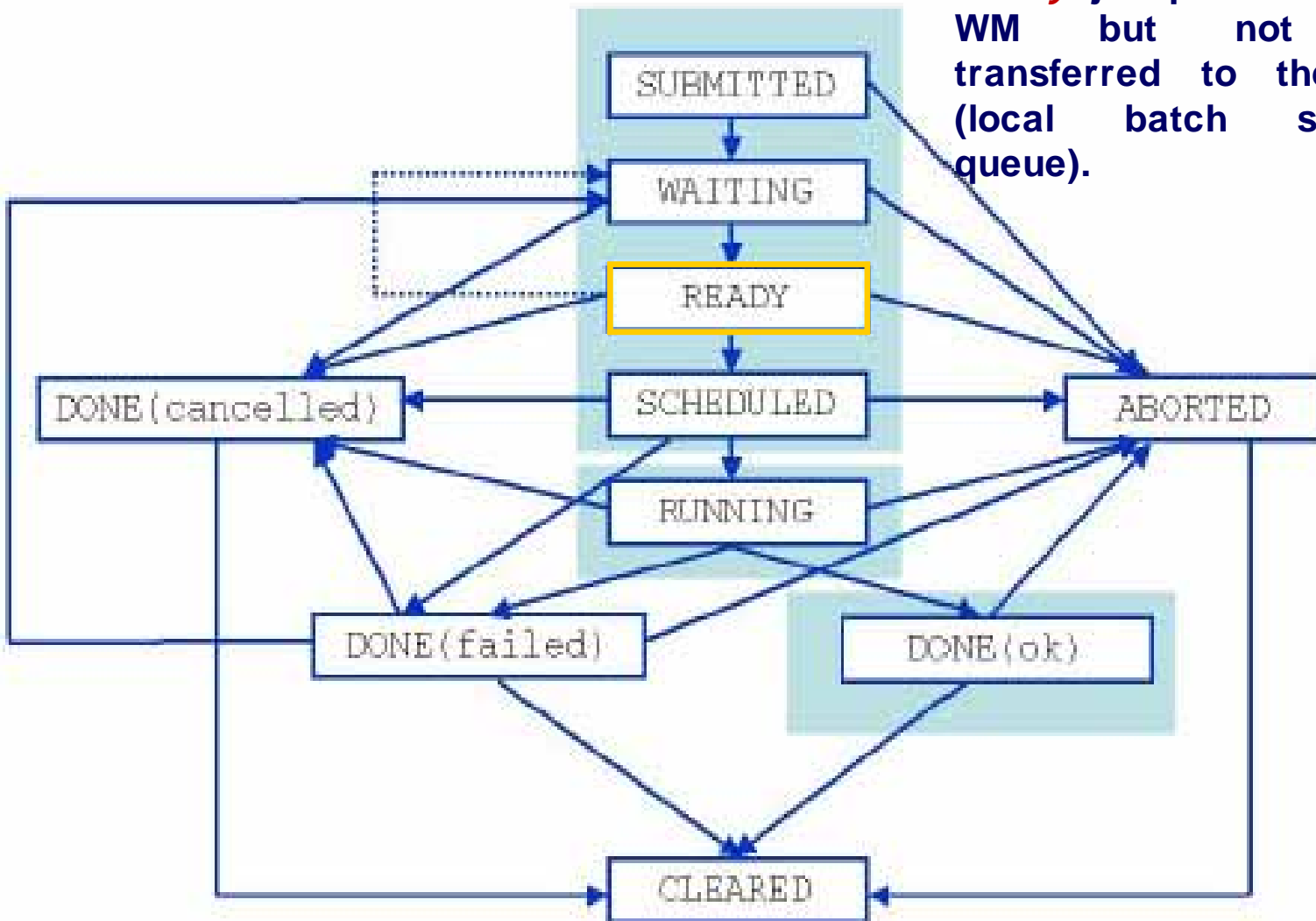
Submitted job is entered by the user to the User Interface but not yet transferred to Network Server for processing

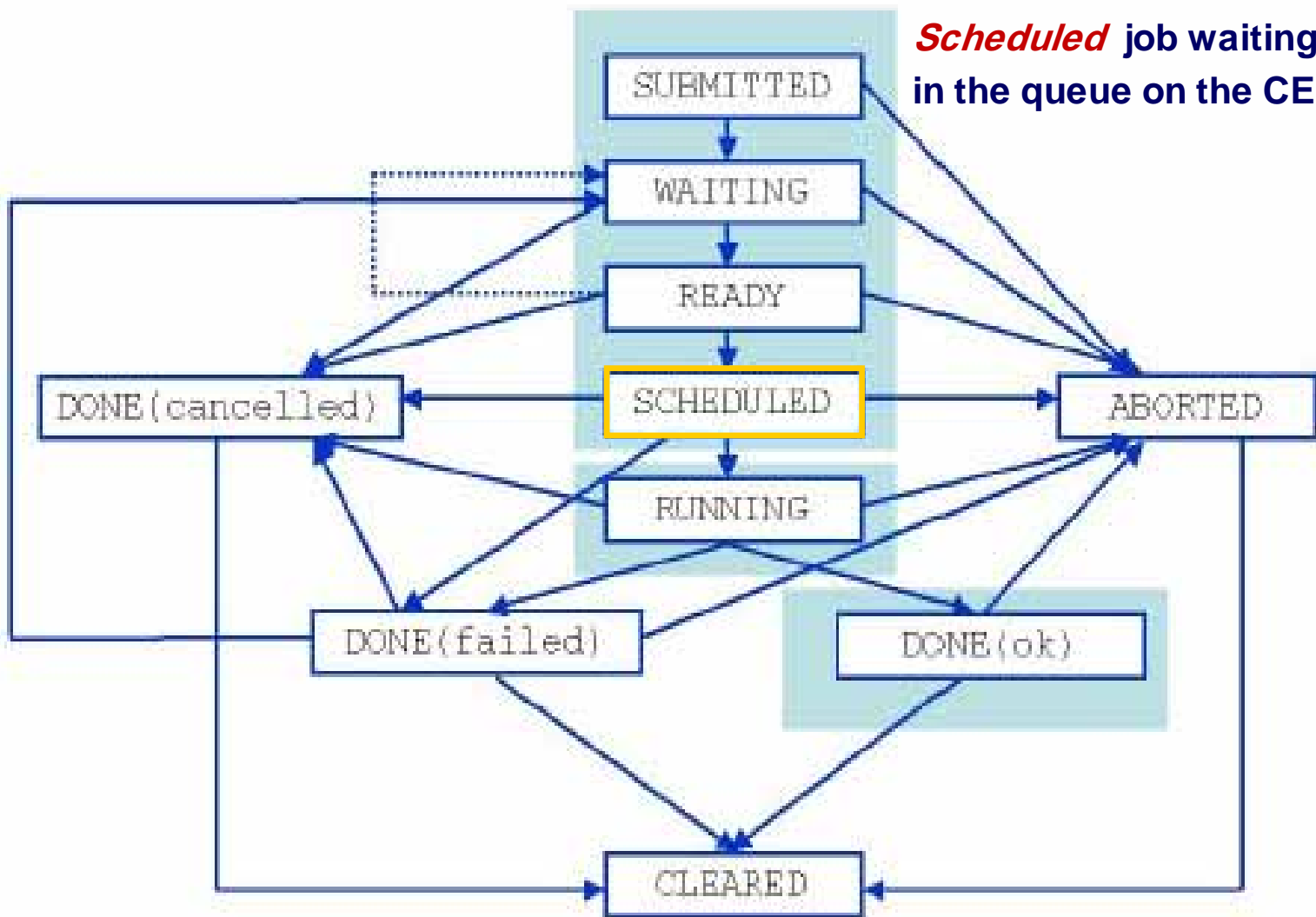
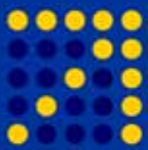


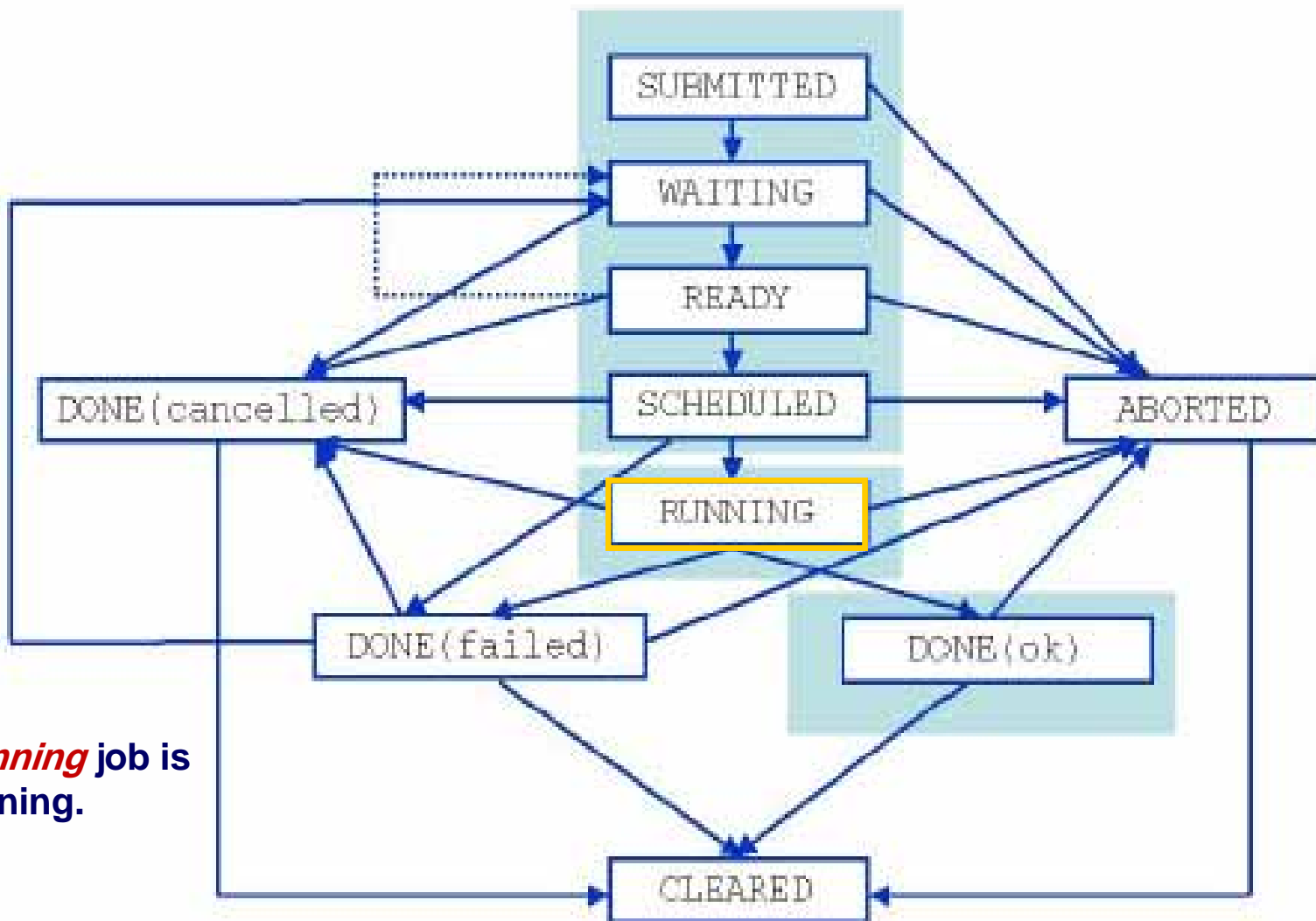


Jobs State Machine (3/9)

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



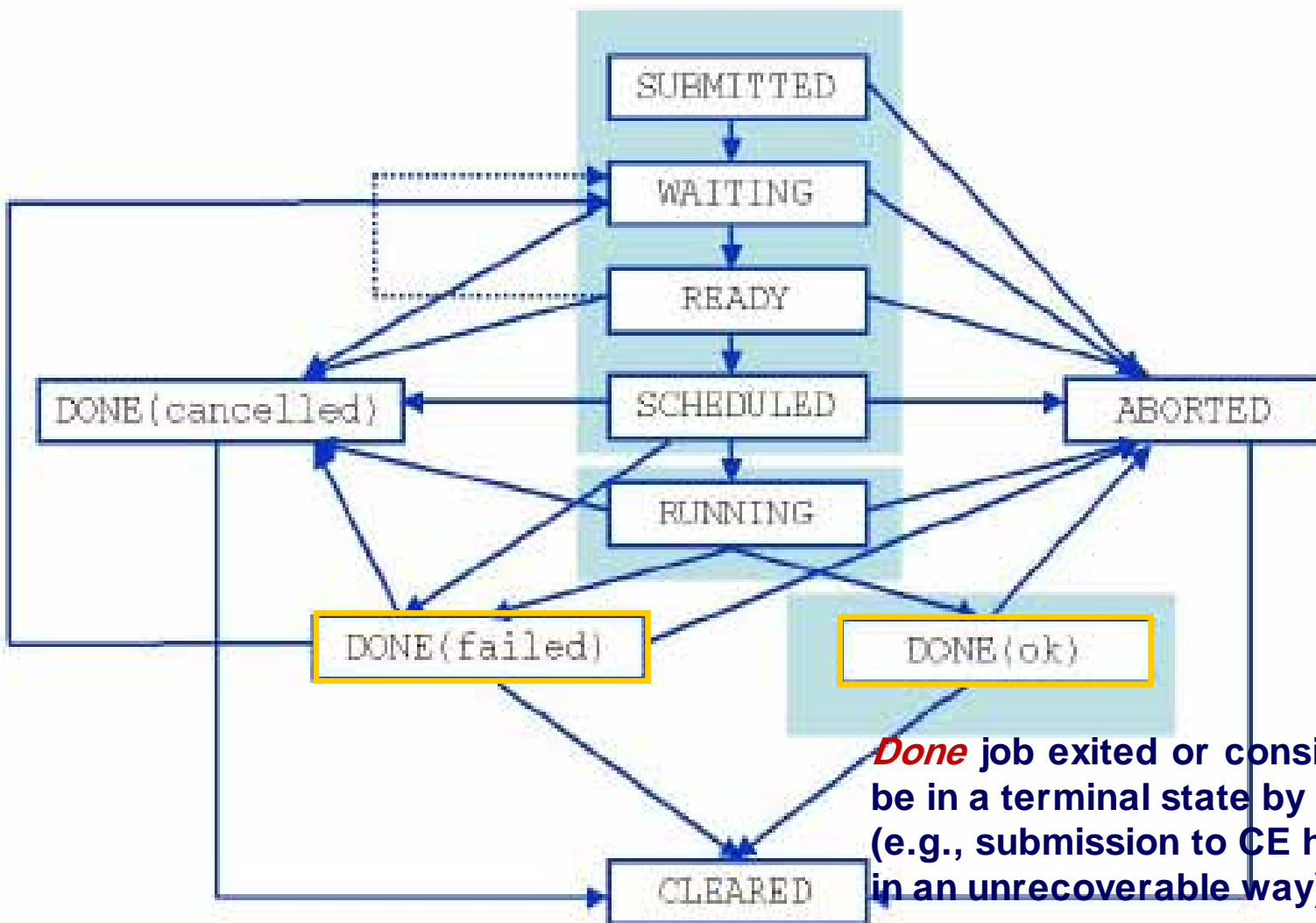




Running job is running.



Jobs State Machine (6/9)

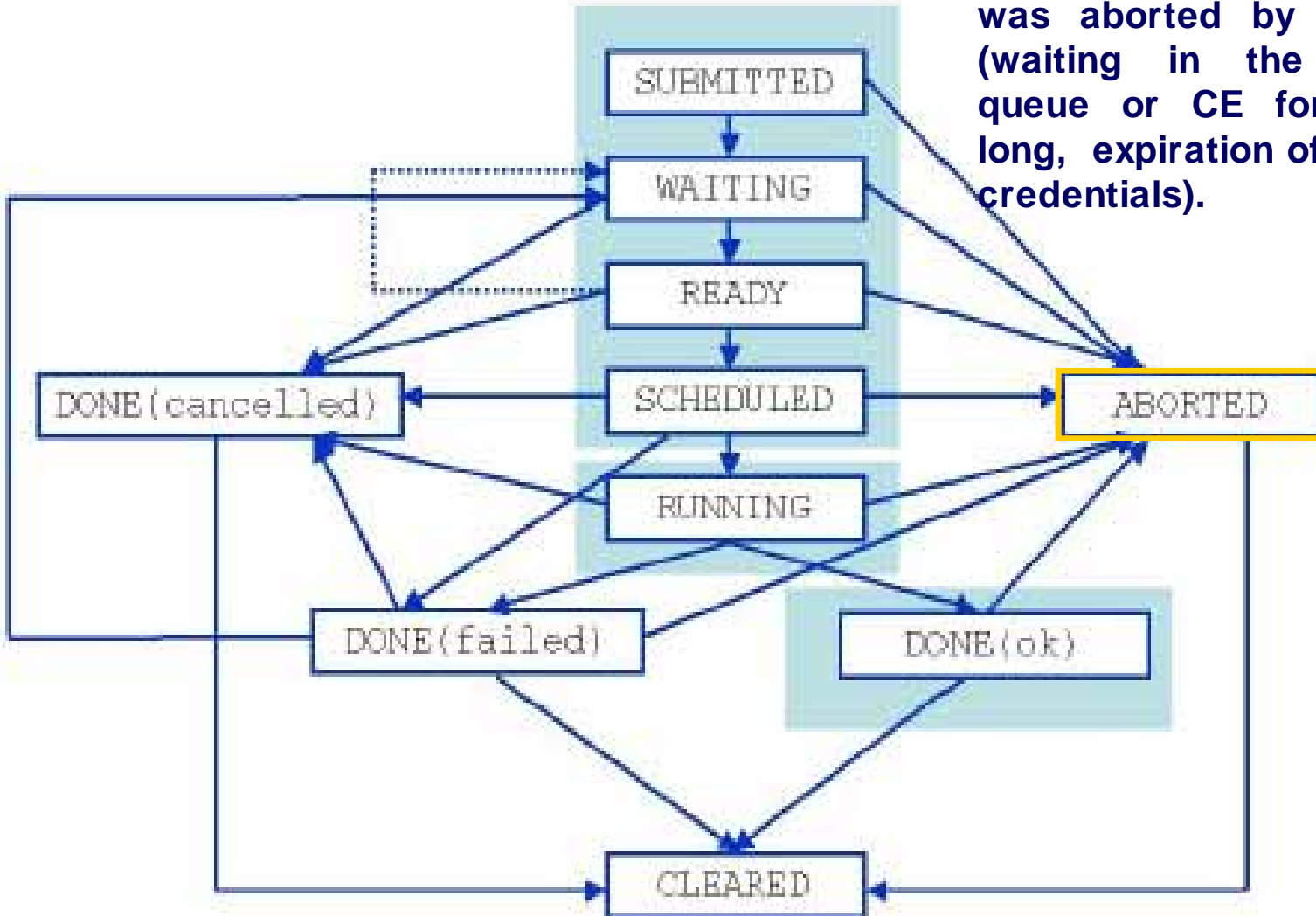


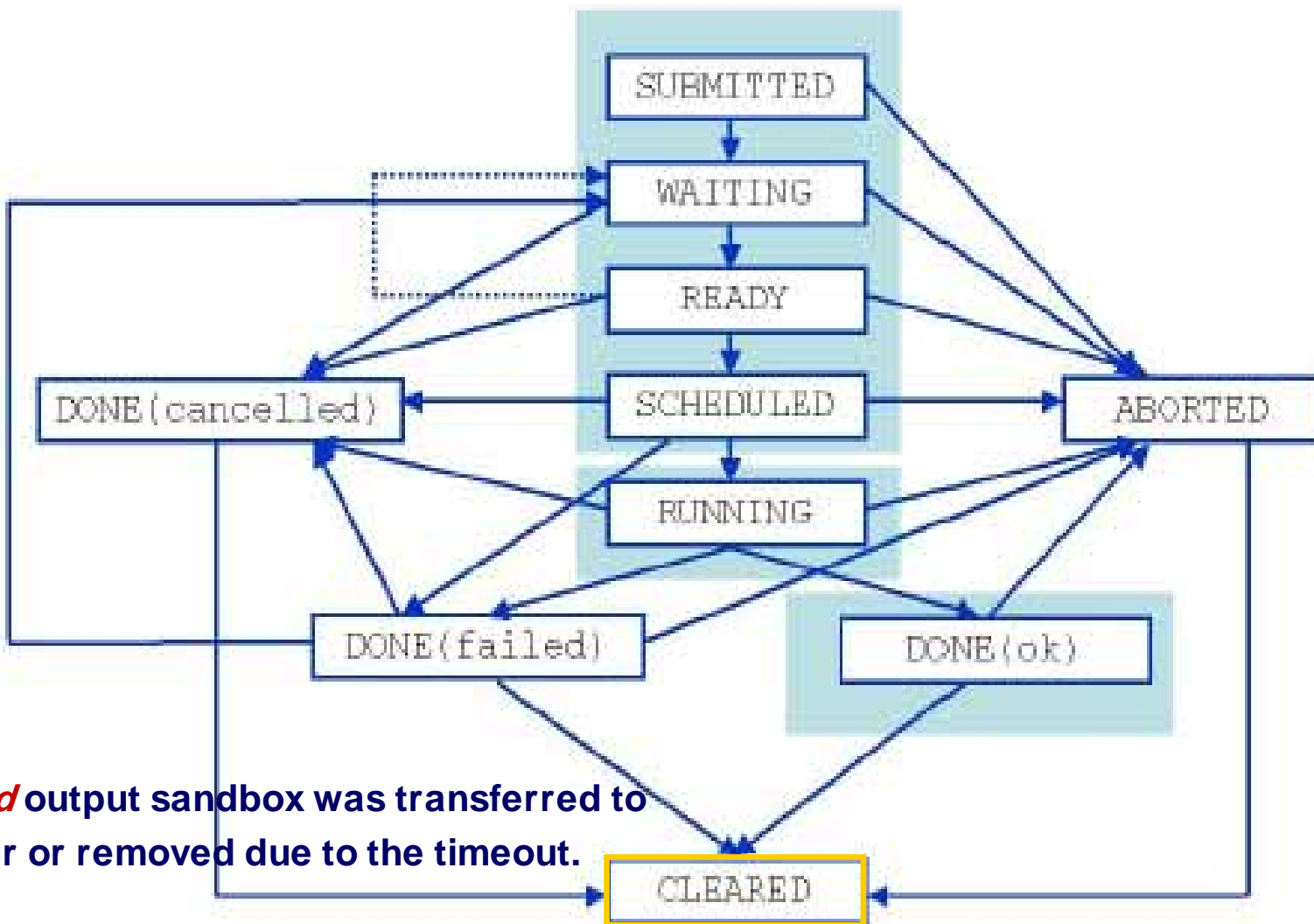
Done job exited or considered to be in a terminal state by CondorC (e.g., submission to CE has failed in an unrecoverable way).



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).





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



- **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 jobld 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 typical 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 from zero will be returned.



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

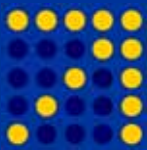
COMPUTING ELEMENT IDs LIST

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: Job successfully submitted to Globus

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

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



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.**



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>

=====



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.

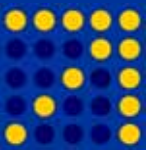


- ✚ In a JDL, some attributes are mandatory while others are optional.
- ✚ An “essential” JDL is the following:

```
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.



JobType	Specify the type of jobs (Normal, MPICH, Parametric, Interactive, Checkpointable, Partitionable)
Executable	Shell script/command to execute on the WN
Arguments	List of arguments to the Executable attribute
Environment	List of environment settings needed by the job
InputSandbox	List of file(s) needed by the job for running
OutputSandbox	List of file(s) to retrieve at the end of the computation
Requirements	Job Requirements on computation
InputData	List of LFN or GUID needed by the job as input
NodeNumber	Number of CPU needed for a MPI job
OutputData	File to be automatic uploaded by the job



Third Part

WMPProxy Overview



- **The Workload Manager Proxy (WMPProxy) is the service responsible to provide access to the WMS functionalities through a Web Service Interface**
 - It has been designed to handle a large number of requests for job submission.
 - 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*.



- **The client must be properly authorized when interacts with the WMPProxy 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 WMPProxy machine.**

```
[root@glite-rb2 etc]# cat glite_wms_wmproxy.gacl
<gacl version='0.0.1'>
  <entry>
    <voms><fqan>bio/Role=NULL</fqan></voms>
    <allow><exec/></allow>
  </entry>
</gacl>
```



- **Each job submitted to a WMPProxy Service must be associated with user's credentials previously delegated.**
- **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 mechanisms to ask for a delegation of the user credentials:**
 - **asking the “automatic” delegation of the credentials during the submission operation**
 - **asking for an “explicit” delegation**



WMPProxy 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
 - Further information about how to interact with the services exposed by the gLite WMPProxy through the provided API Java can be found here:

<https://grid.ct.infn.it/view/twiki/bin/view/GILDA/ApiJavaWMPProxy>



WMPProxy C++ client commands

The commands to interact with WMPProxy 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>



New Features



- **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 transferred only once by the new WMS client commands but can be accessed by all sub-jobs of the compound job**
 - **Each sub-jobs can refer to a single file 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" };
```

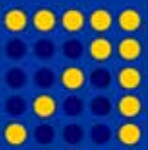
- **A base URI to be applied to all sandbox files can also be specified**

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



'Compressed' Sandboxes

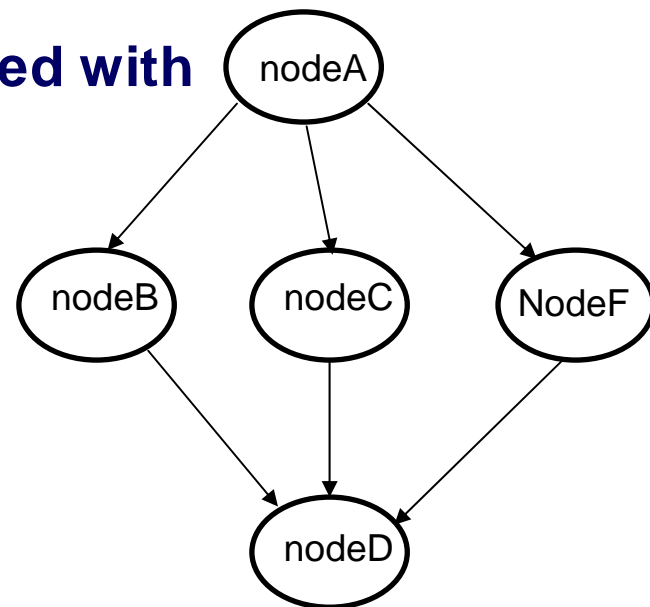
- A compressed archive is created with the input sandboxes files using **libtar** and **zlib** libraries
 - This is done automatically by WMPProxy client commands
 - This mechanism can be enabled/disabled by the user through the JDL (*AllowZippedISB* attribute)
- The archive is transferred (instead of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



New type of request



- **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'





```
[
  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 }
    }
  ];
];
```



WMPProxy : submission & monitoring

- In order to submit job with WMPProxy, it's mandatory 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 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

```
[  
  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);
```

```
  ]};
```

```
]
```



- 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



- The **Parameters** attribute is 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 used to identify 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:
$$N = (\text{Parameters} - \text{ParameterStart}) / \text{ParameterStep}$$



```
[
  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
```



- **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>

