gLite Workload Management System

Emidio Giorgio
INFN Catania

www.eu-egee.org

Information Society

Outline

Workload Management System

WMS Architecture - Job state machine

Job Description Language Overview

- compound jobs

The Computing Element

WMS Objectives

The **Workload Management System** (WMS) comprises a set of Grid middleware components responsible for distribution and management of tasks across Grid resources.

The purpose of the Workload Manager (WM) is to accept and satisfy requests for job management coming from its clients

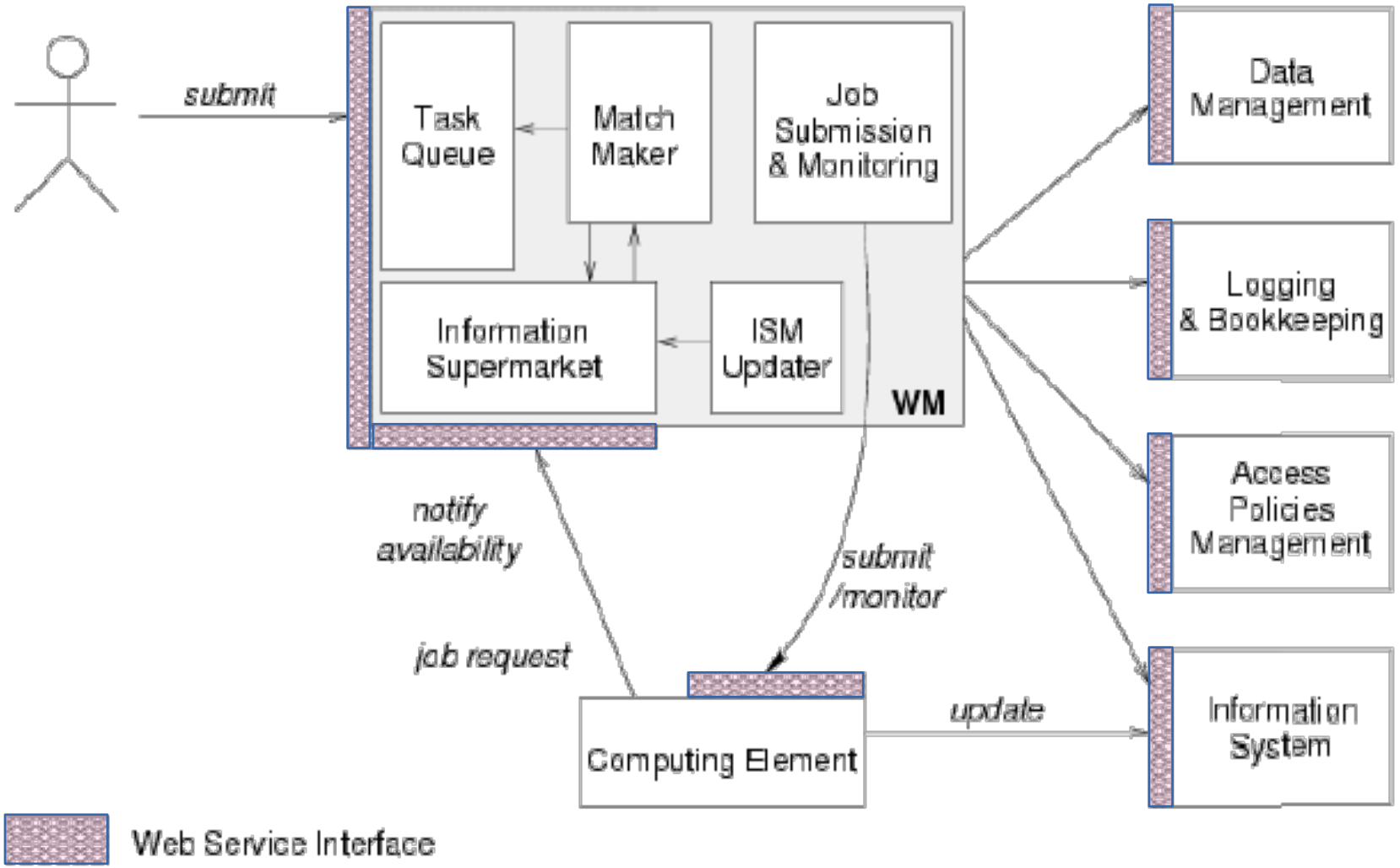
meaning of the submission request is to pass the responsibility of the job to the WM.

WM will pass the job to an appropriate CE for execution

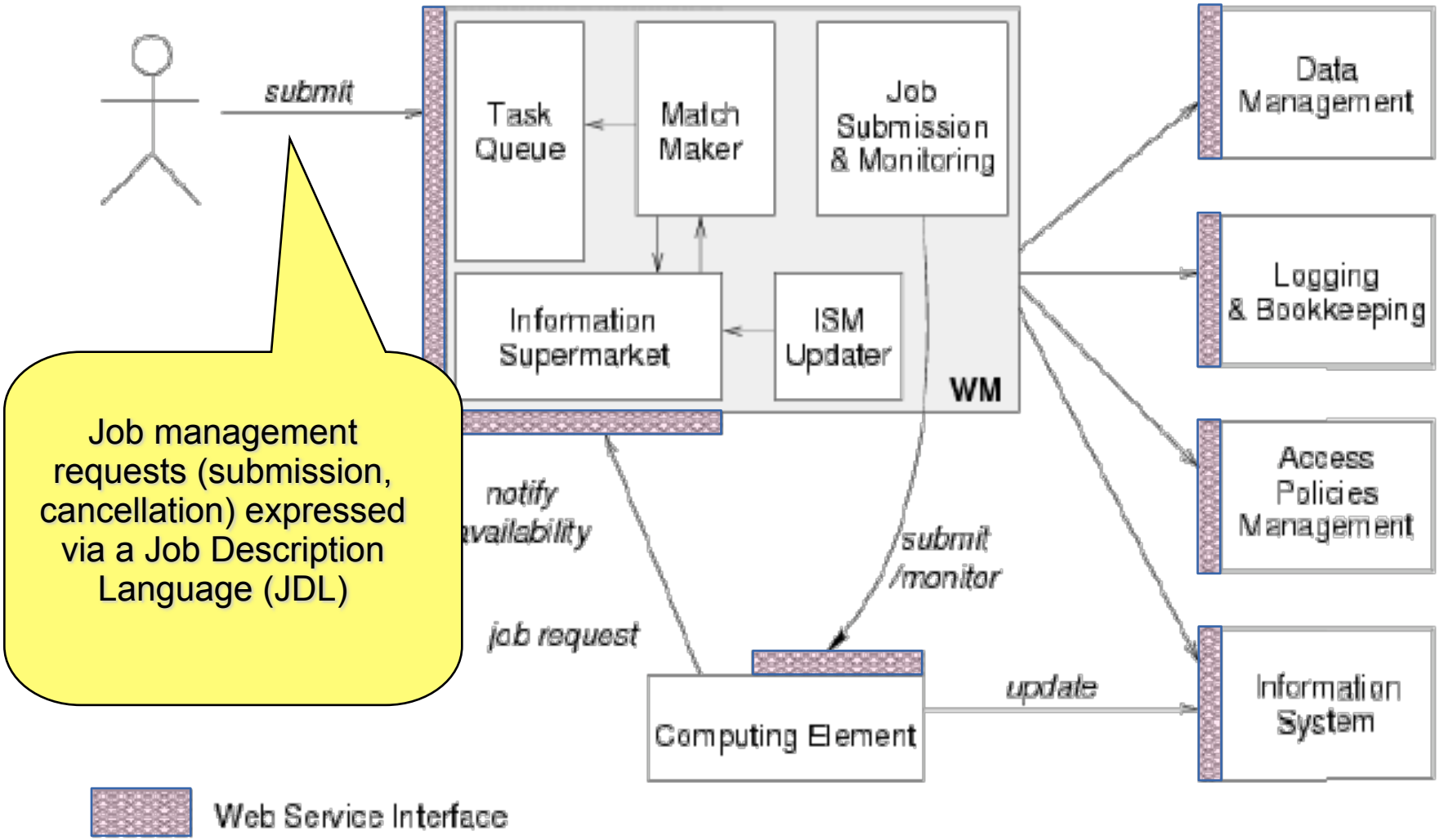
taking into account requirements and the preferences expressed in the job description file

The decision of which resource should be used is the outcome of a **matchmaking** process.

WMS Architecture

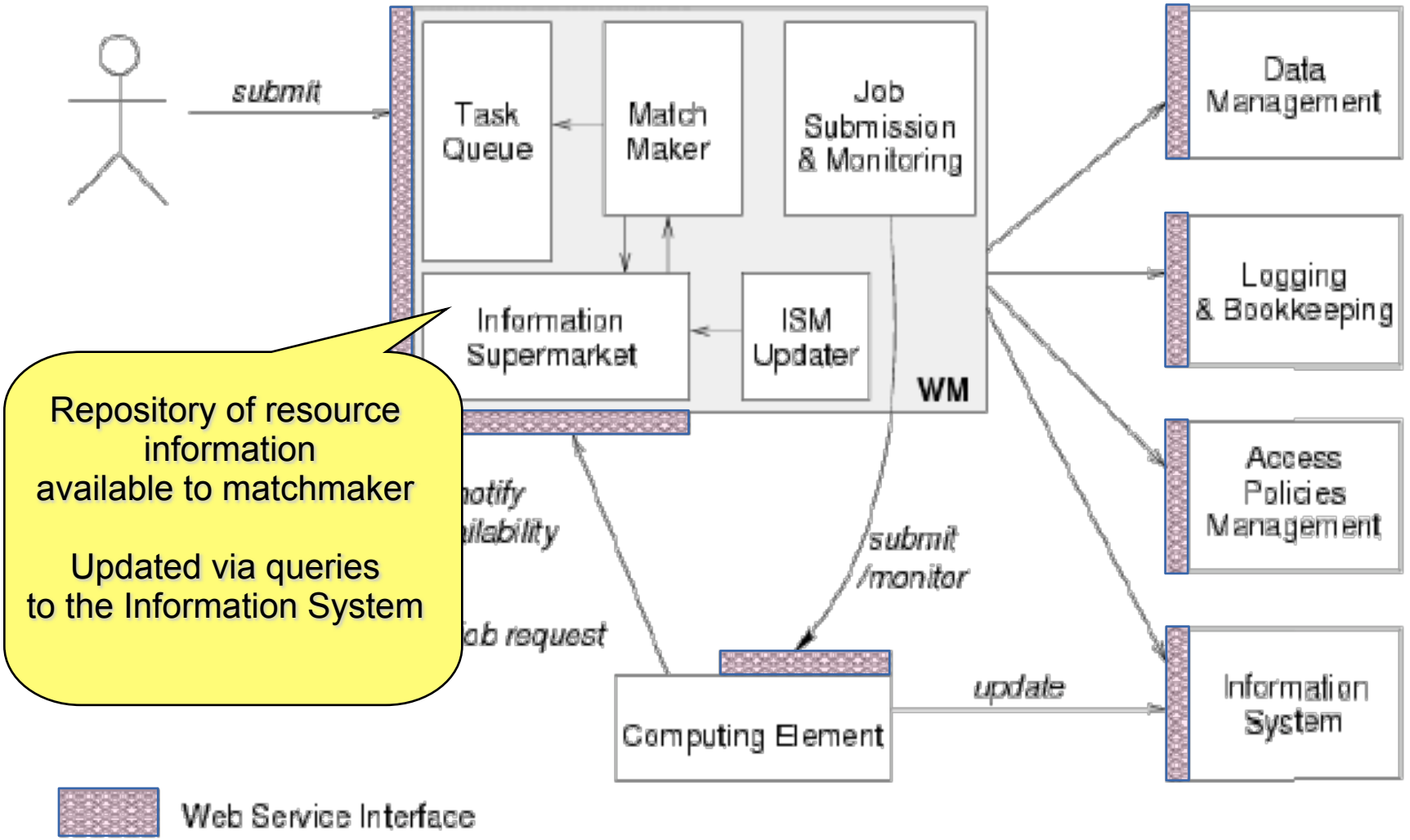


WMS Architecture



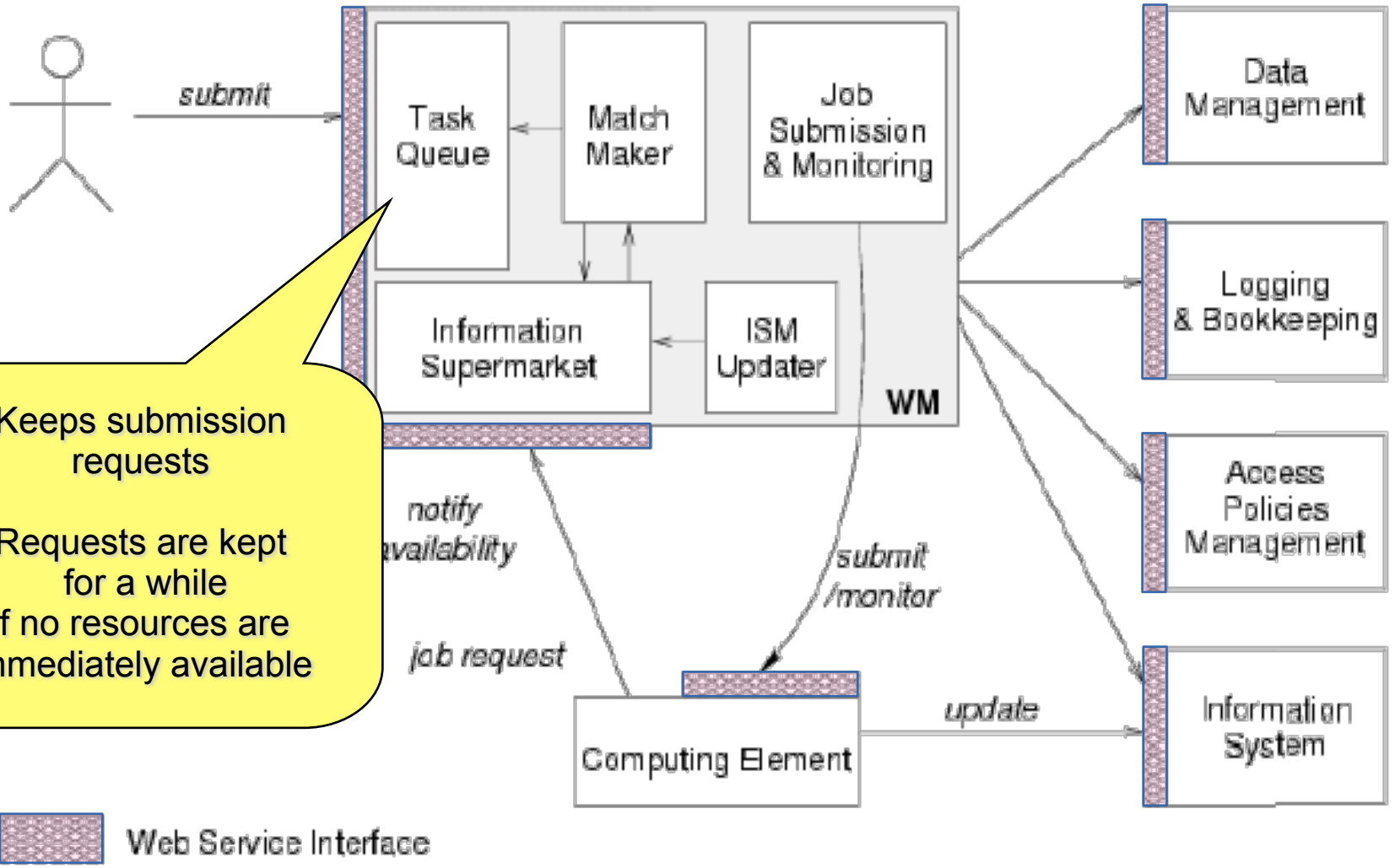
Job management requests (submission, cancellation) expressed via a Job Description Language (JDL)

WMS Architecture



WMS Architecture

State = **Submitted**

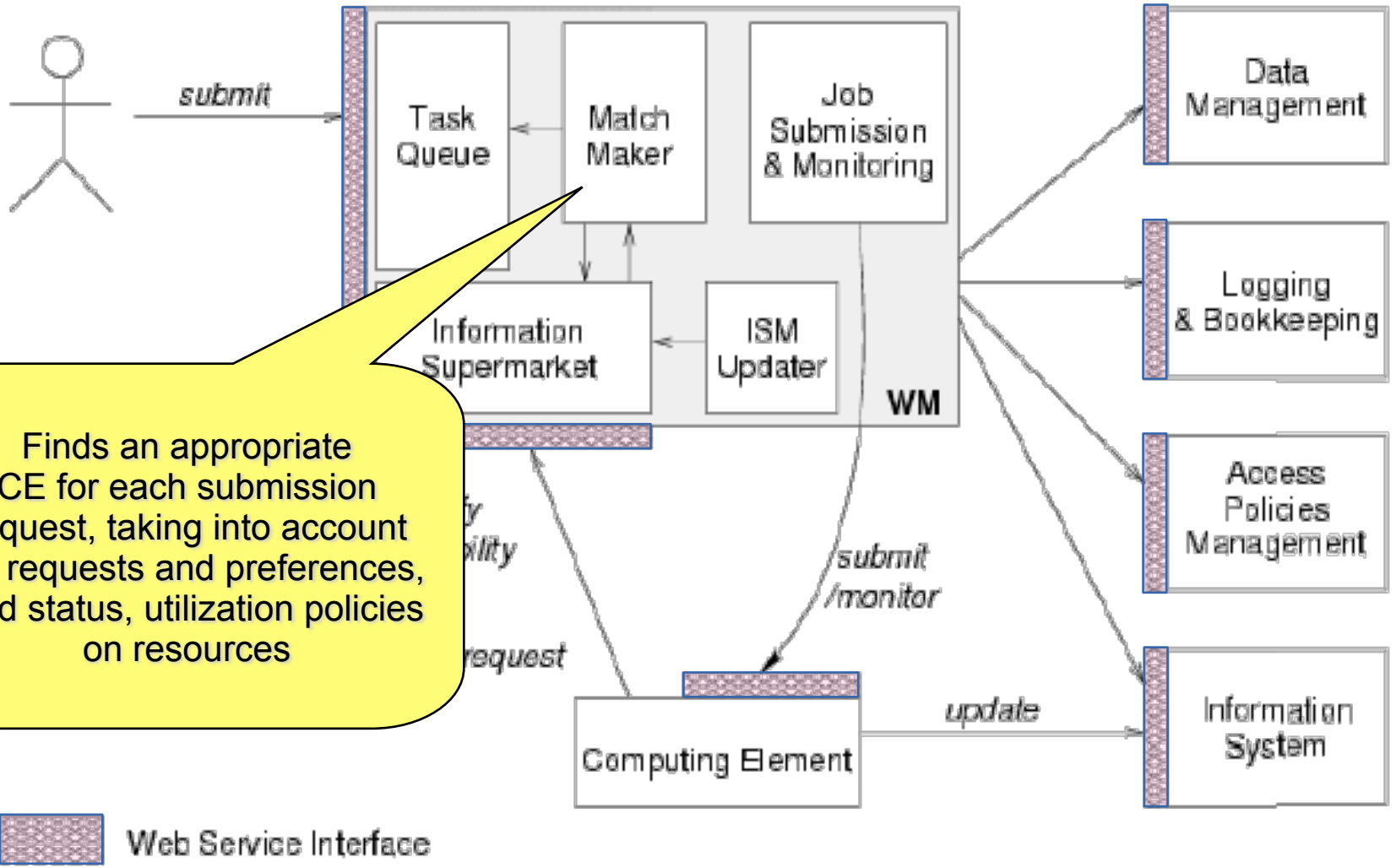


Keeps submission requests

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

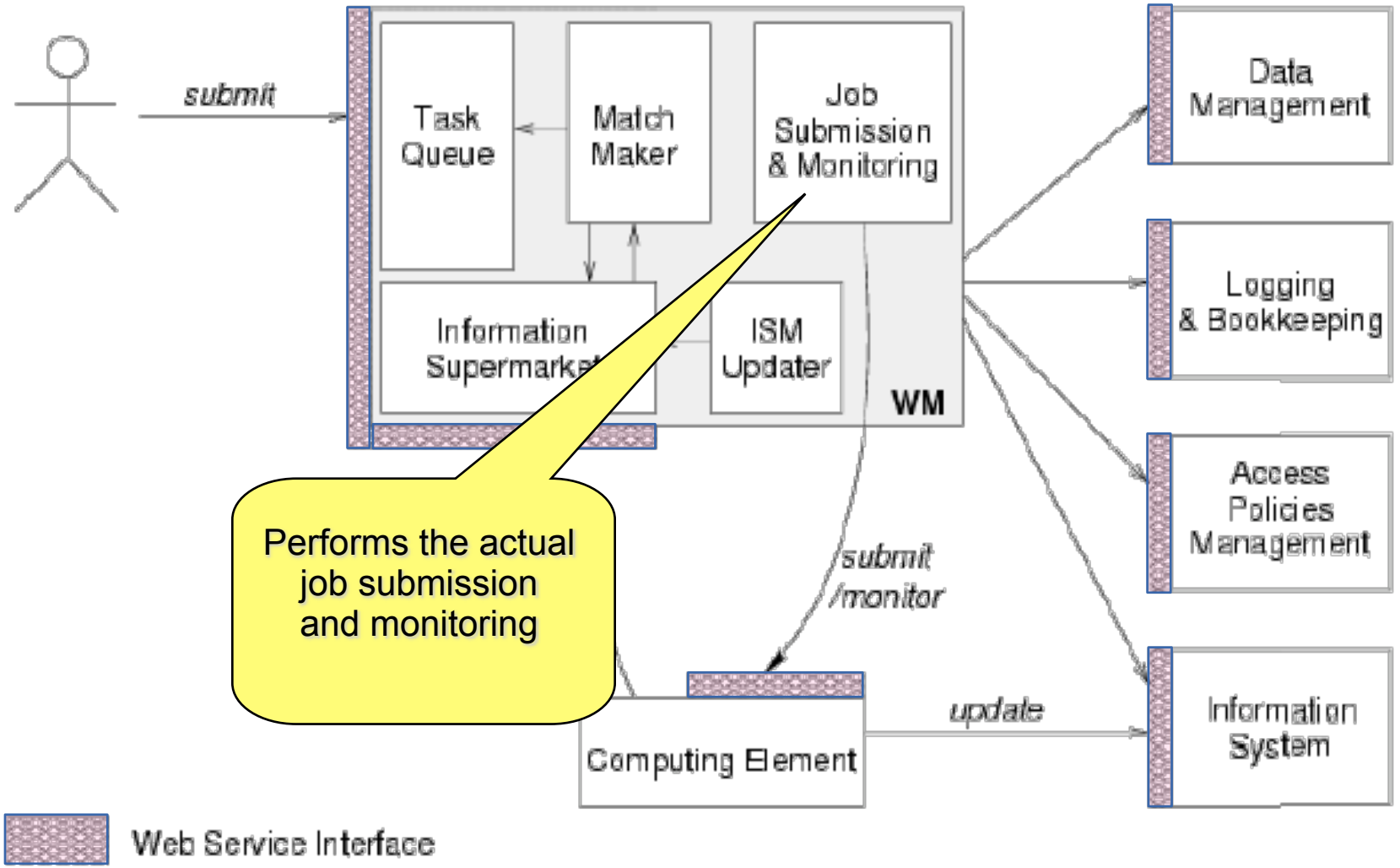
WMS Architecture

State = **Waiting**



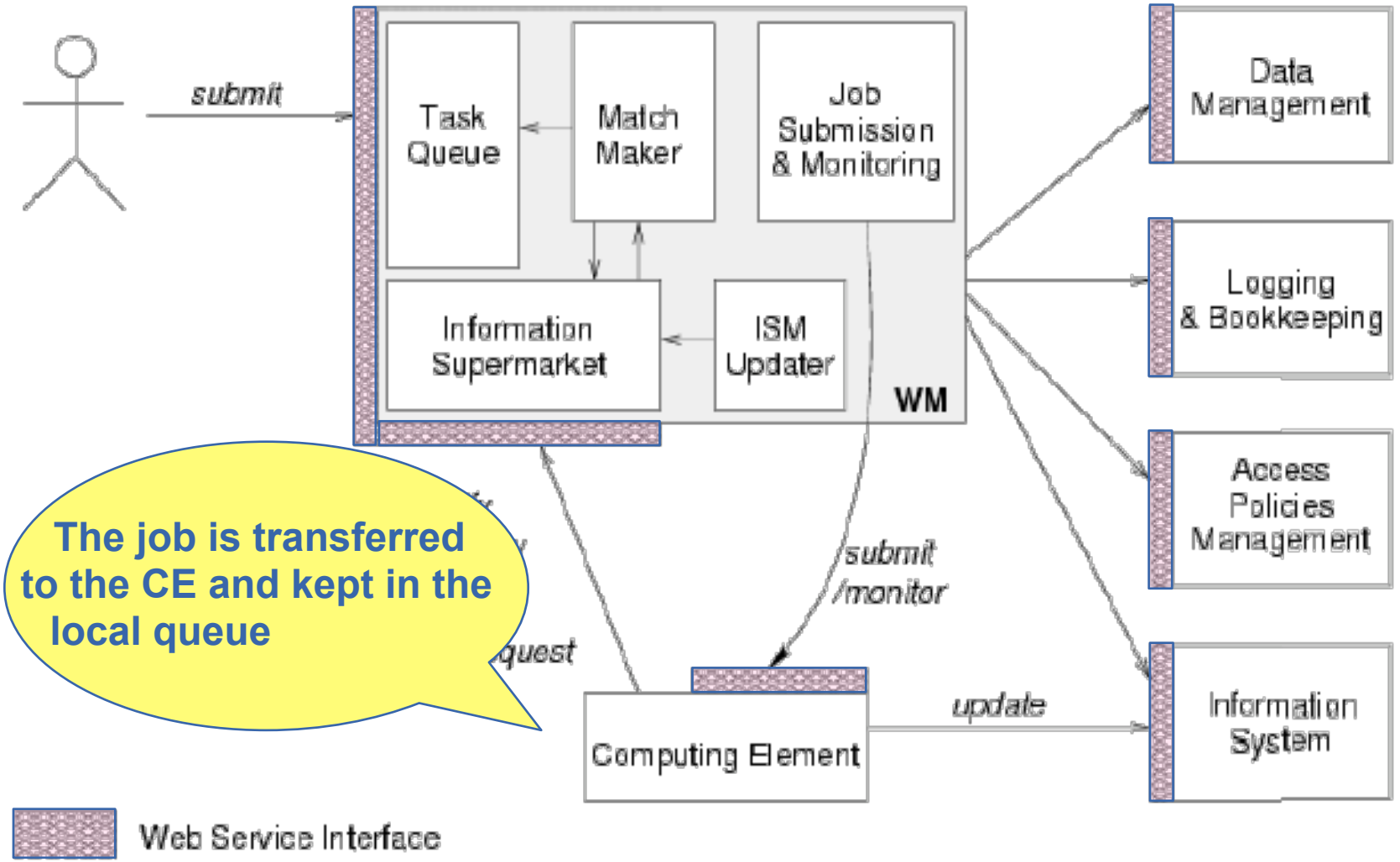
WMS Architecture

State = **Ready**



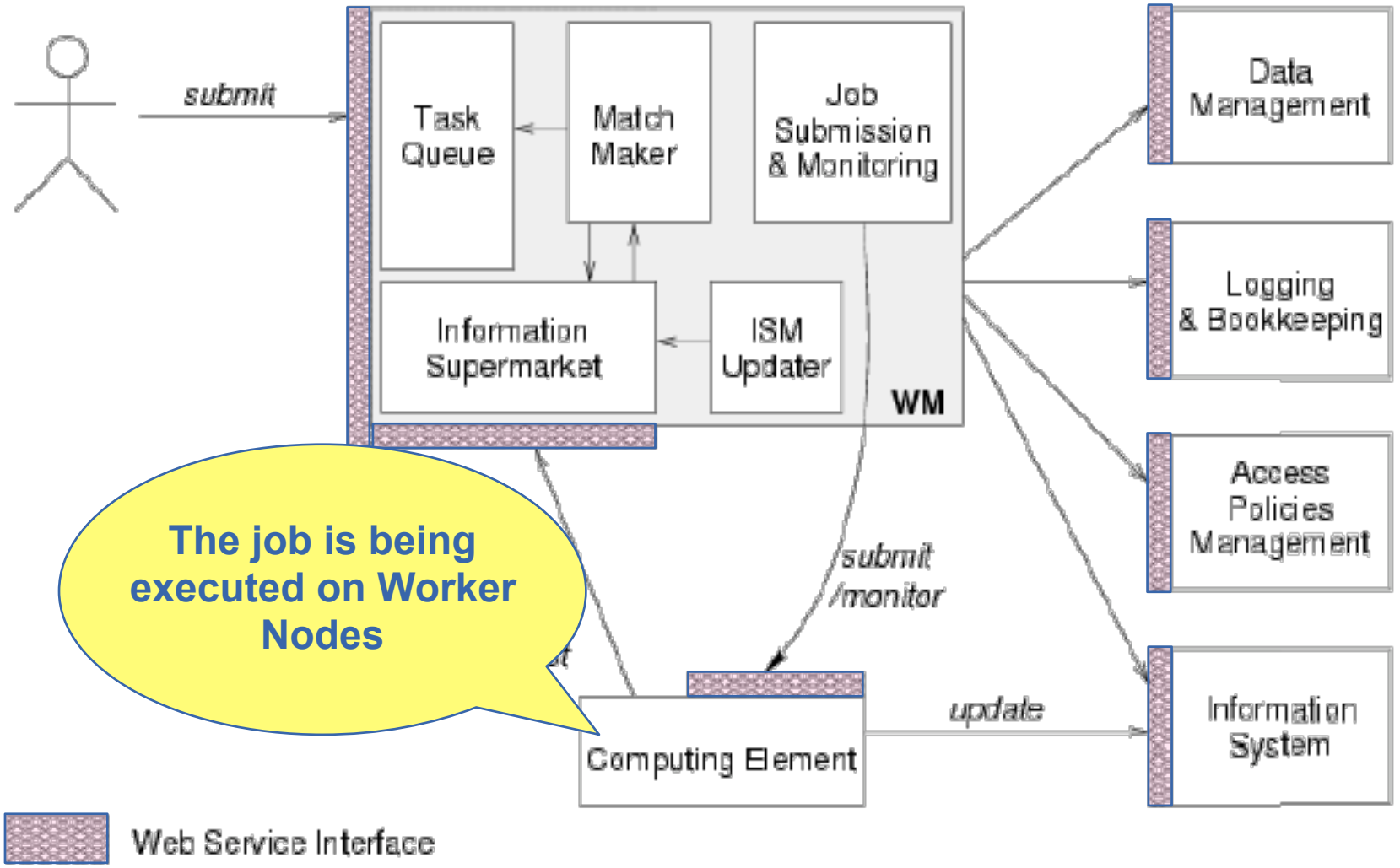
WMS Architecture

State = **Scheduled**



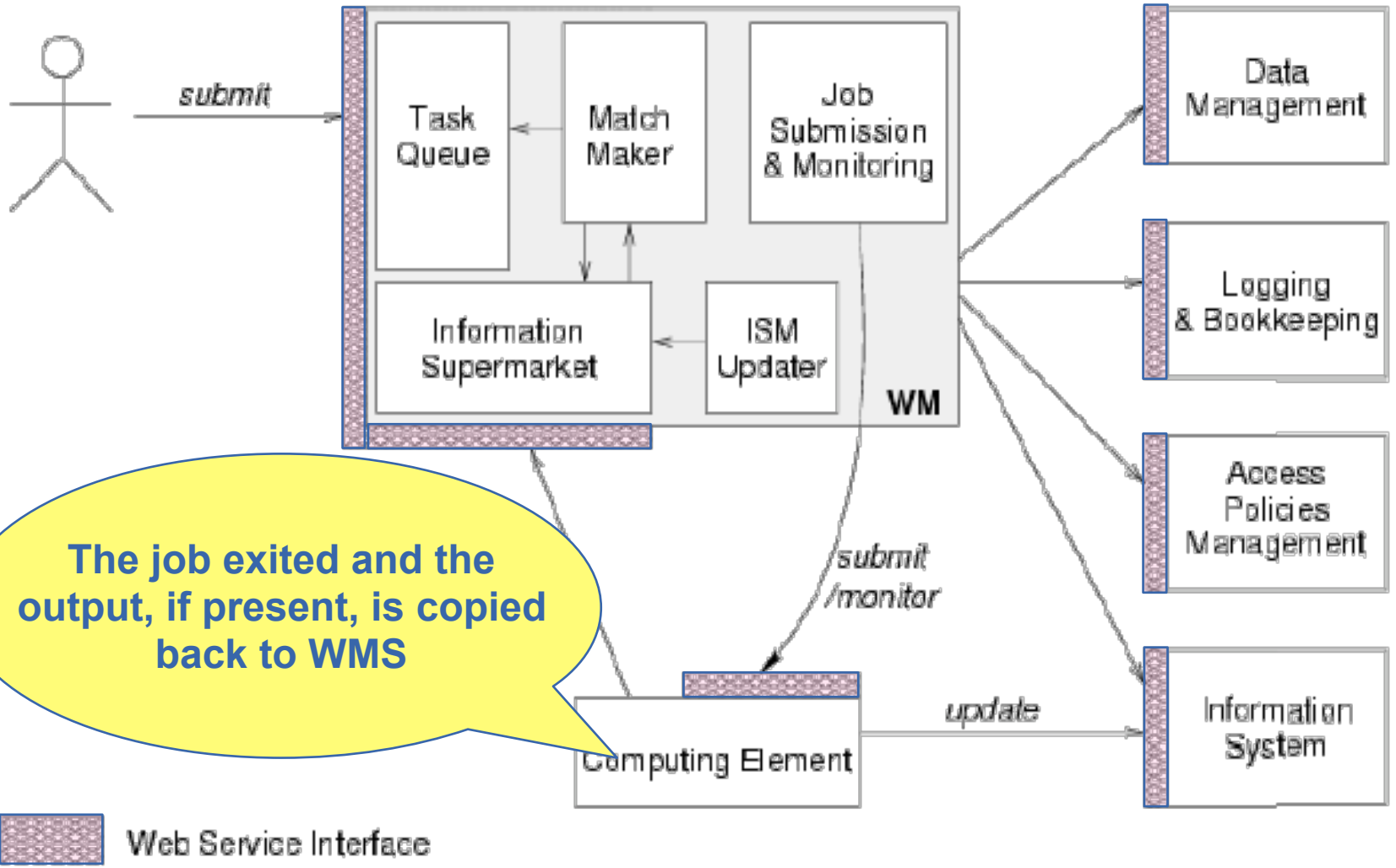
WMS Architecture

State = **Running**

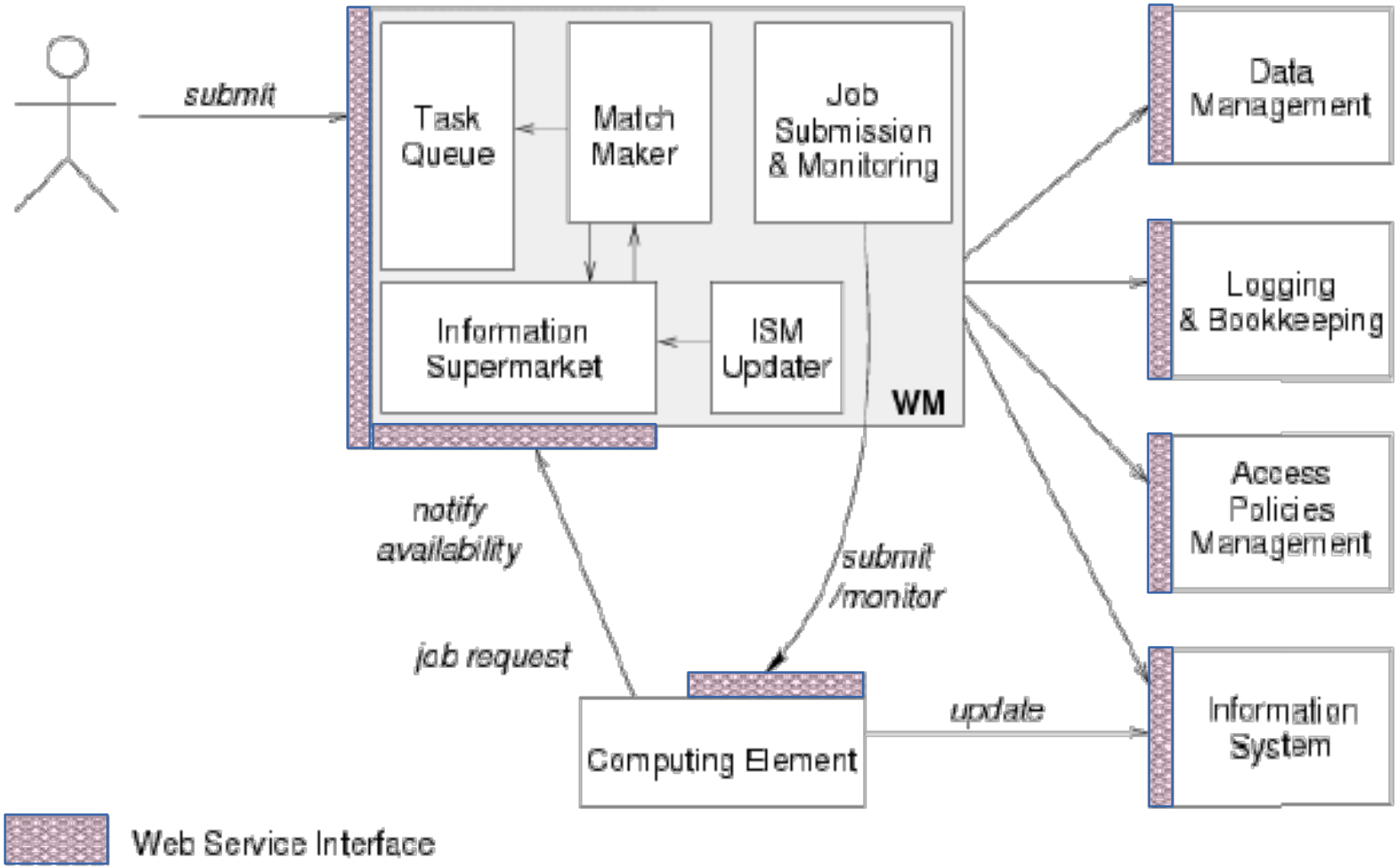


WMS Architecture

State = **Done**

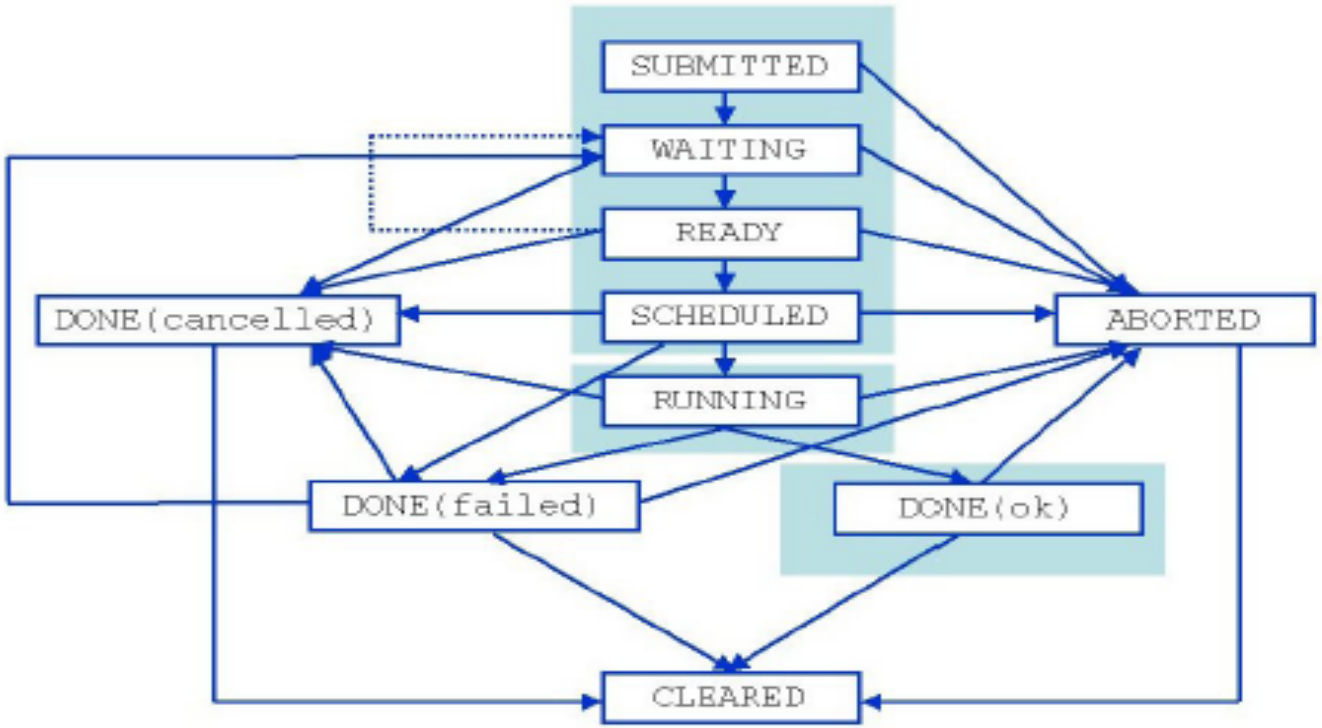


WMS Architecture



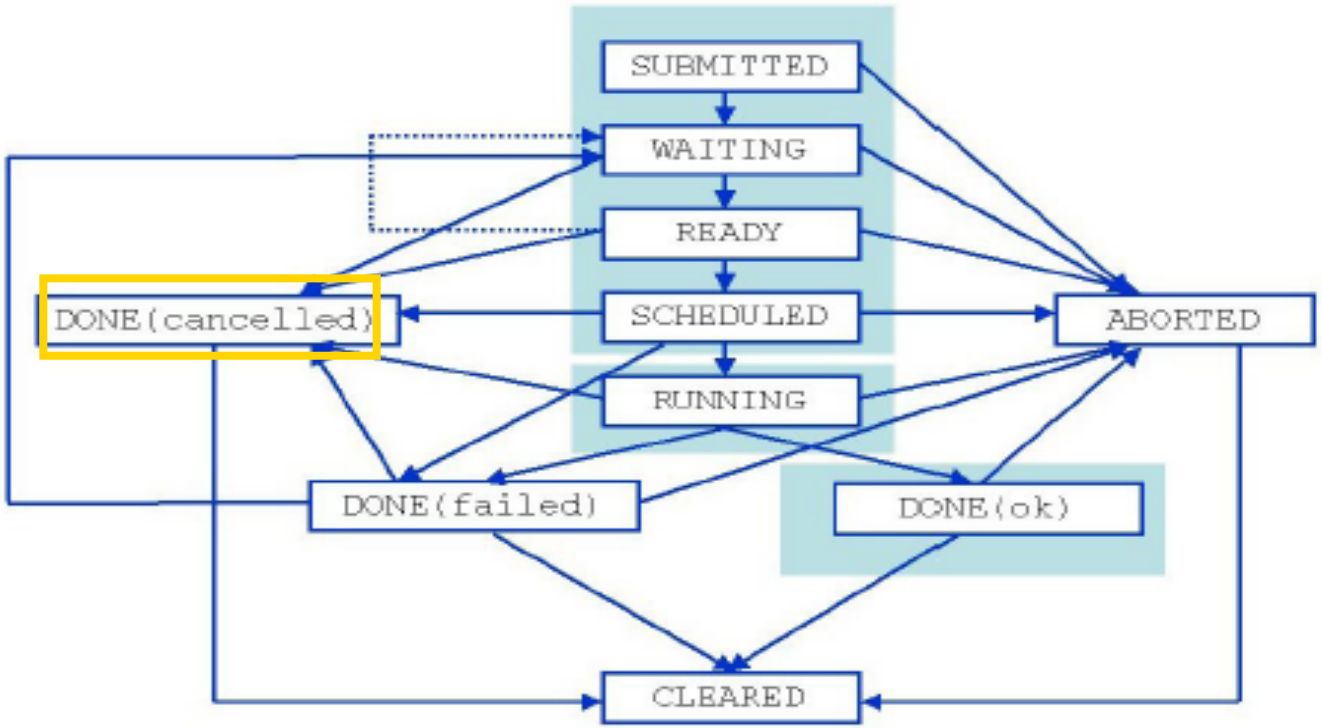
Jobs State Machine : exceptions

Jobs State Machine : exceptions



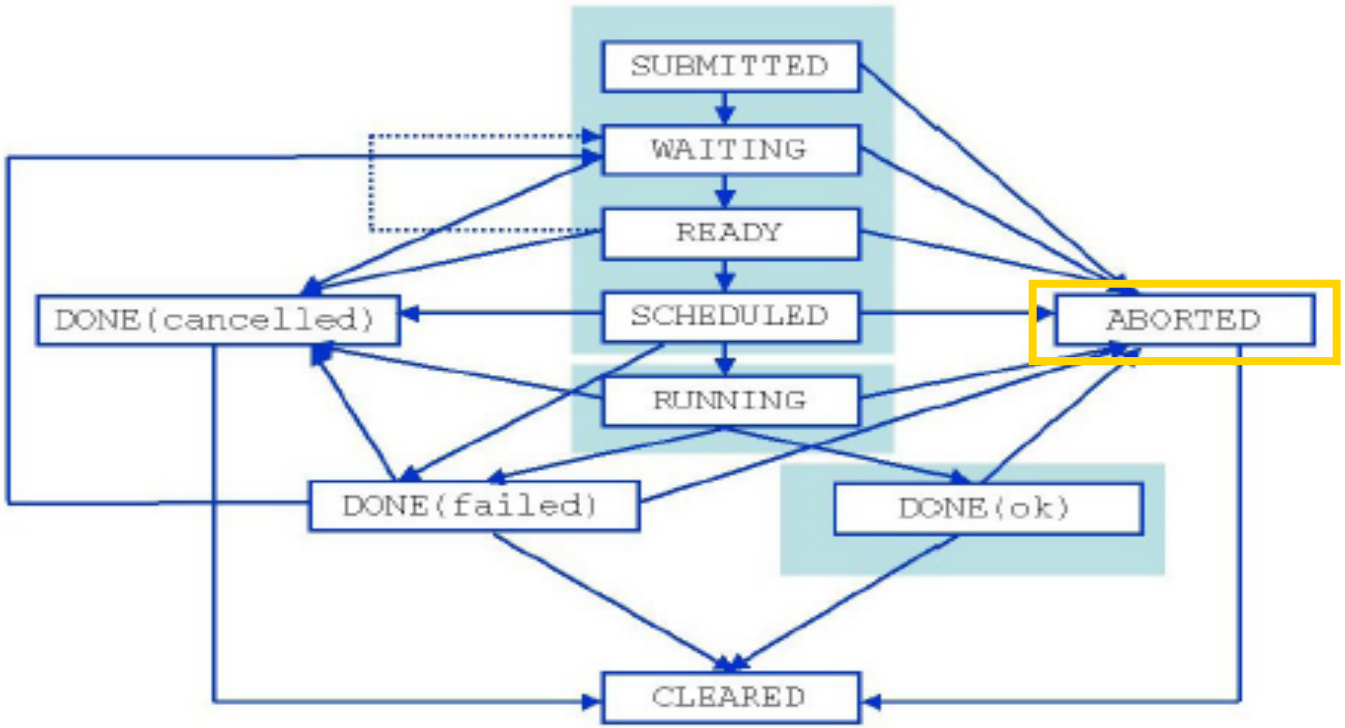
Jobs State Machine : exceptions

Cancelled job has been successfully canceled on user request.

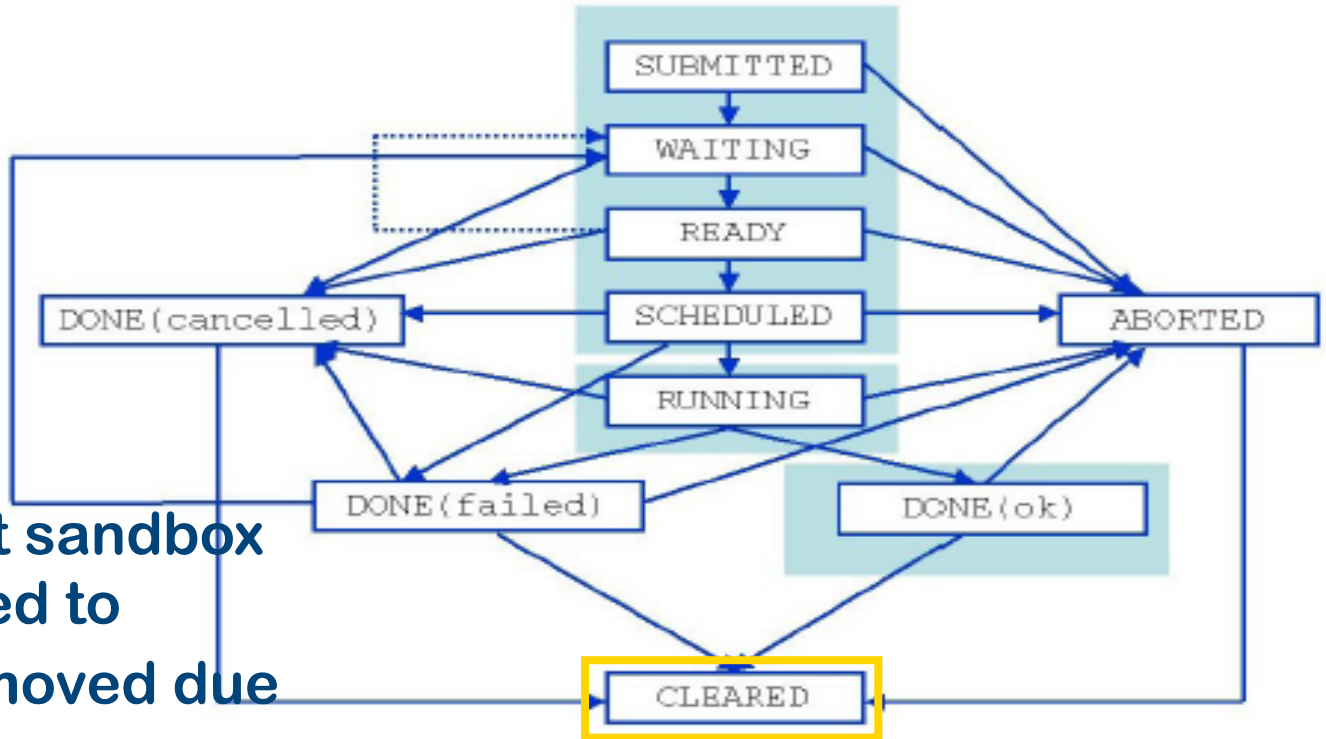


Jobs State Machine : exceptions

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

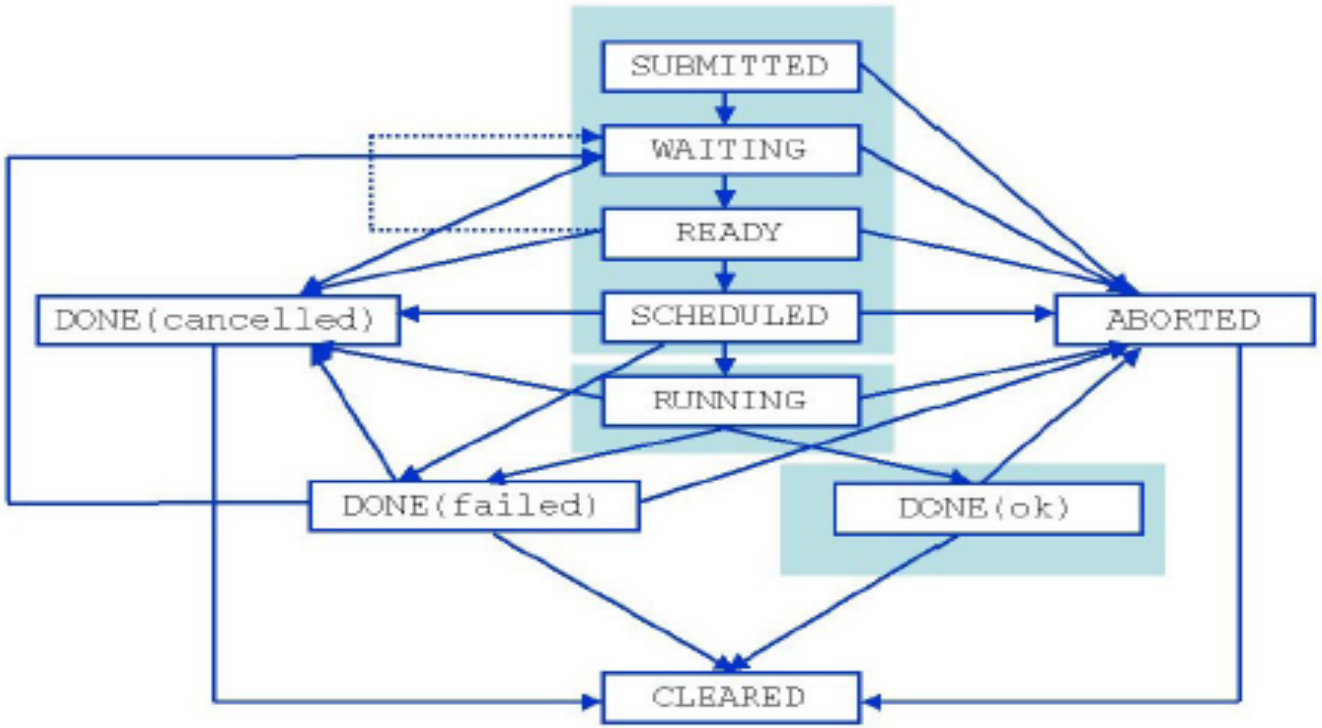


Jobs State Machine : exceptions



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

Jobs State Machine : exceptions



Logging and Bookkeping

- **Every step of the job life cycle is logged on a service called Logging and Bookkeeping**
- **It is useful for users willing to know the status of their execution**
  - when a job is submitted the UI logs it on LB
  - WMS logs each step of scheduling
  - CE logs when it receive a job (scheduled), when it's running and when it's done

Job Description Language

Job Description Language

In gLite, Job Description Language (JDL) is used to describe jobs for execution.

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 attributes 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.

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

Executable indicates which file will be executed remotely

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

Environment allows to specify env. variables which will be set at run time

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

Arguments appends a string (to be used as argument) to Executable

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

StdOutput is the remote file where output will be redirected

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

StdError is the remote file where std error will be redirected

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

InputSandbox defines a set of local files that you want to be staged remotely for execution

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

OutputSandbox defines a set of remote files that you want to get back after execution

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

Requirements allows to specify a set of characteristic (hardware or software that you wish for the resource.

JDL: an example

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./
gint.jar","LD_LIBRARY_PATH=.:
$LD_LIBRARY_PATH","LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it lfn:/grid/
gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox =
{"startGen4.sh","gint.jar","gfal.jar","libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements =
Member("GLITE-3_0_0",other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```

Other relevant JDL attributes

- If your job needs a file stored somewhere, you can specify its LFN :
- The file will not be copied to the resource executing the job but your job scheduled to a CE near the SE holding that file
- That is crucial when dealing with large files

```
DataRequirements = {
                   [
                     InputData = {"lfn:/grid/gilda/emidio/test.txt"};
                     DataCatalogType = "DLI";
                     DataCatalog = "http://lfc-gilda.ct.infn.it:8085";
                   ]
         };
         DataAccessProtocol = {"rfio","gsiftp"};
```

Othere relevant JDL attributes

- **Rank** : allows to override UI's default for fitness function on which resources are classified

```
Rank = ( other.GlueCEStateWaitingJobs == 0 ?
other.GlueCEStateFreeCPUs : -
other.GlueCEStateWaitingJobs);
```

- **RetryCount** : override default for times that a job will be resubmitted after the first failure

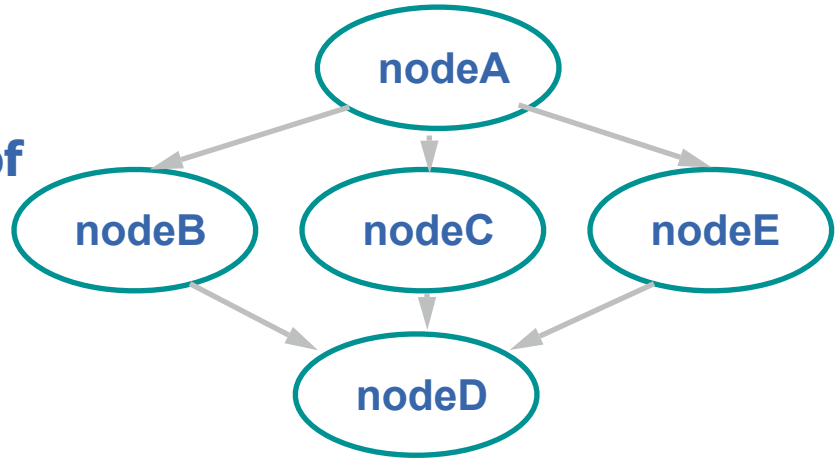
```
RetryCount = 7
```

- **Requirements** : a wide set of attributes, as they are published from the BDII, can be required. Regular expressions can be even set, and/or combined with

```
Requirements =
(RegExp("*.pdc.kth.se",other.GlueCEUniqueID));
```

Workflows of jobs

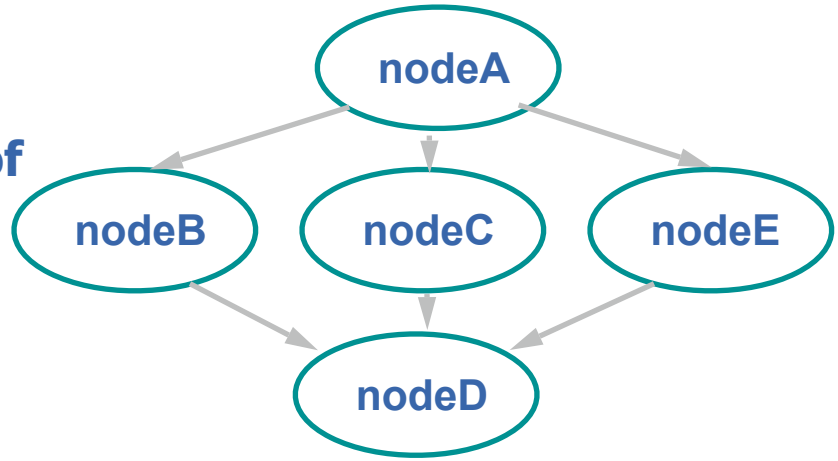
- **With a single request, multiple jobs can be generated and executed**
- **Direct Acyclic Graph (DAG) is a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs**
- **A Collection is a group of jobs with no dependencies**
  - basically a collection of JDL's



- **A Parametric job is a job having one or more attributes in the JDL that vary their values according to parameters**
- **Using compound jobs it is possible to have one shot submission of a (possibly very large, up to thousands) group of jobs**
  - Submission time reduction
    - Single call to WMProxy server
    - Single Authentication and Authorization process
    - Sharing of files between jobs
  - Availability of both a single Job Id to manage the group as a whole and an Id for each single job in the group

Workflows of jobs

- **With a single request, multiple jobs can be generated and executed**
- **Direct Acyclic Graph (DAG) is a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs**
- **A Collection is a group of jobs with no dependencies**
  - basically a collection of JDL's

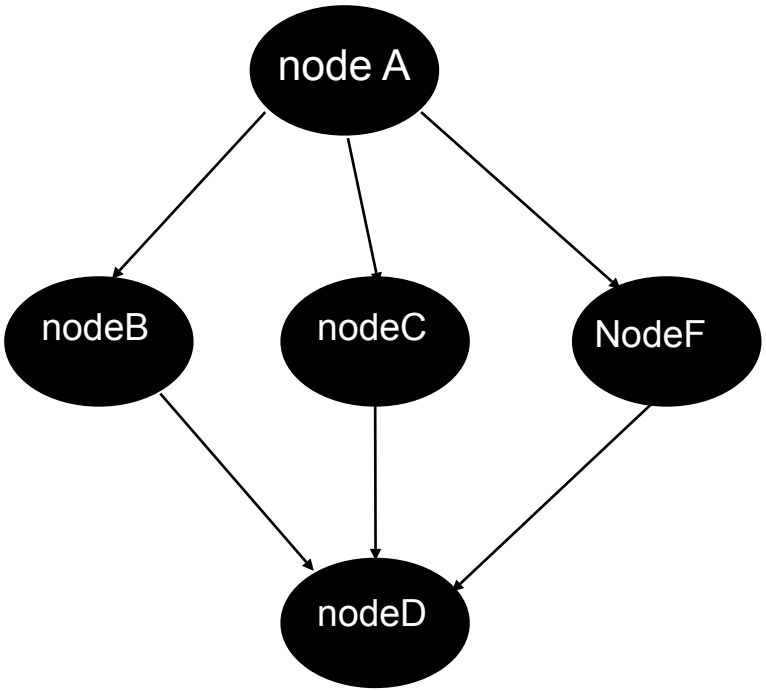


- **A Parametric job is a job having one or more attributes in the JDL that vary their values according to parameters**
- **Using compound jobs it is possible to have one shot submission of a (possibly very large, up to thousands) group of jobs**
  - Submission time reduction
    - Single call to WMProxy server
    - Single Authentication and Authorization process
    - Sharing of files between jobs
  - Availability of both a single Job Id to manage the group as a whole and an Id for each single job in the group

During hands-on limit the size of compound jobs !

DAG job

- A DAG job is a set of jobs where input, output, or execution of one or more jobs can depend on other jobs
- Dependencies are represented through Directed Acyclic Graphs, where the nodes are jobs, and the edges identify the 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

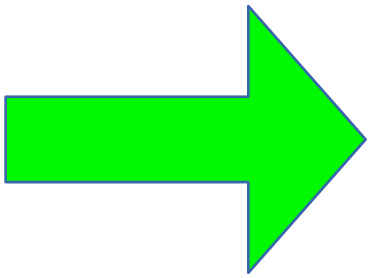
Nodes is the core of DAG attributes :

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

          dependencies = .....

]
```

```
......
Nodefilename1 = [ file = "foo.jdl";]
Nodefilename2 = [ file = "bar.jdl";
    retry = 2;]
]
```



```
Nodefilename1 = [
  description = [ JobType="Normal";
                  Executable="myexe";
                  Arguments = "1 2 4";
                  InputSandbox="input.txt";
                  OutputSandbox="out.txt";
                  .......
    ]
]
```

Attribute: Dependencies

It is a list of list representing dependencies among the DAG nodes

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

          dependencies = .....

]
```

dependencies =
{ nodefilename1,nodefilename2 }

Mandatory : YES !

dependencies = {}

{node1,node2}

{{{node1,node2},node3},node4}

{{node1,node2},node3}

DAG jdl

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

Job Collection

- A job collection is a set of independent jobs that user wants to submit and monitor via a single request
- Jobs of a collection are submitted as DAG nodes without dependencies
- JDL is a list of classad, which describes the subjobs

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

Job collection example

```
[
  type = "collection";
  InputSandbox = {"date.sh"};
  RetryCount = 3;
  nodes = {
    [
      file ="jobs/job1.jdl" ;
    ],
    [
      [
         Executable = "/bin/sh";
         Arguments = "date.sh";
         StdOutput = "date.out";
         StdError =  "date.err";
         OutputSandbox ={"date.out", "date.err"};
       ]
    ],
    [
      file ="jobs/job3.jdl" ;
    ]
  };
]
```

Job collection example

```
[
  type = "collection";
  InputSandbox = {"date.sh"};
  RetryCount = 3;
  nodes = {
    [
      file ="jobs/job1.jdl" ;
    ],
    [
      [
        Executable = "/bin/sh";
        Arguments = "date.sh";
        StdOutput = "date.out";
        StdError =  "date.err";
        OutputSandbox ={"date.out", "date.err"};
      ]
    ],
    [
      file ="jobs/job3.jdl" ;
    ]
  };
]
```

All nodes will share this Input Sandbox

‘Scattered’ Input Sandboxes

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

```
InputSandbox = {
    "gsiftp://neo.datamat.it:2811/var/prg/sim.exe",
     root.nodes.nodeA.description.OutputSandbox[0],
    "file:///home/pacio/myconf“ };
```

- **Only local files (`file://`) are uploaded to the WMS node**
- **File pointed by URIs are directly downloaded on the WN by the JobWrapper just before the job is started**

‘Scattered’ Output Sandboxes

- JDL is enriched with attributes, specifying the destinations for the files listed within 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/";
```

- Files are copied when the job has completed execution by the JobWrapper to the specified destination without transiting on the WMS node

Parametric Job

- A parametric job is a job where one or more of its attributes are parameterized
- Values of attributes vary according to a parameter

```
[
  JobType = "Parametric";
  Executable = "/bin/sh";
  Arguments = "md5.sh input_PARAM_.txt";
  InputSandbox = {"md5.sh", "input_PARAM_.txt"};
  StdOutput = "out_PARAM_.txt";
  StdError = "err_PARAM_.txt";
  Parameters = 4;
  ParameterStart = 1;
  ParameterStep = 1;
  OutputSandbox = {"out_PARAM_.txt", "err_PARAM_.txt"};
]
```

- Job monitoring / managing is always done through an unique jobID, as if the job was single (see submission of collection

Parametric job / 2

- Parameter can be also a list of string
- InputSandbox (if present) has to be coherent with parameters

```
[ui-test] /home/giorgio/param > cat param2.jdl
[
      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"};
]

[ui-test] /home/giorgio/param > ls
inputEARTH.txt  inputMARS.txt  inputMOON.txt param2.jdl
```

The Computing Element

- **The CE is the front-end machine (master node) to a local batch system**
  - supported batch systems are PBS(Torque/MAUI), LSF, Condor
- **WMS "pushes" job execution requests to the CE using `condor-G`**
  - when a CE receives a job, this is moved on a queue
  - Then the job will be executed on the first available among its **Worker Nodes** (where the batch system clients run)
  - when execution is complete, output files are copied to the CE using scp
- **If the job is succesfully executed, output files are copied back to the WMS using `globus-url-copy`**
- **By queries to the LB, users knows when a job is done and they can retrieve the output**

Summary

- **WMS catchs users' request for job executions**
- **Requests are expressed through JDL**
  - JDL allows to specify requirements that selected resources must have
- **The WMS processes request and chooses (matchmaking) a Computing Element for the actual execution**
  - Status of resources is known to WMS with queries to BDII
- **The CE tries to execute the job and copies back output files to WMS**
  - status of execution is logged on LB
- **Users queries LB, discovers their job is done and download output files from WMS**

Practical and references

- **Hands-on**
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/AuthenticationAuthorization
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/SimpleJobSubmission
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/JobDataWMS
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/WmProxyUse
- **If you have time...**
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/MoreOnJDL
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/MPIJobs
- **In depth**
  - https://edms.cern.ch/file/674643/1/WMPROXY-guide.pdf
  - https://edms.cern.ch/file/590869/1/EGEE-JRA1-TEC-590869-JDL-Attributes-v0-9.pdf

Practical information

-