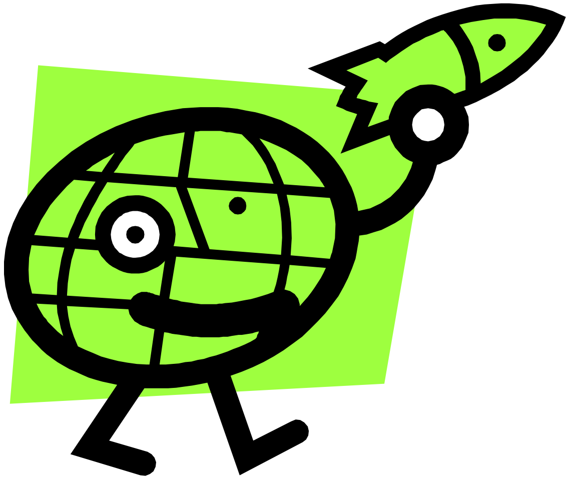
- GILDA WMS - Overview and Praticals

Ruediger Berlich / Forschungszentrum Karlsruhe

slides contributed by INFN Catania, EGEE, FZK

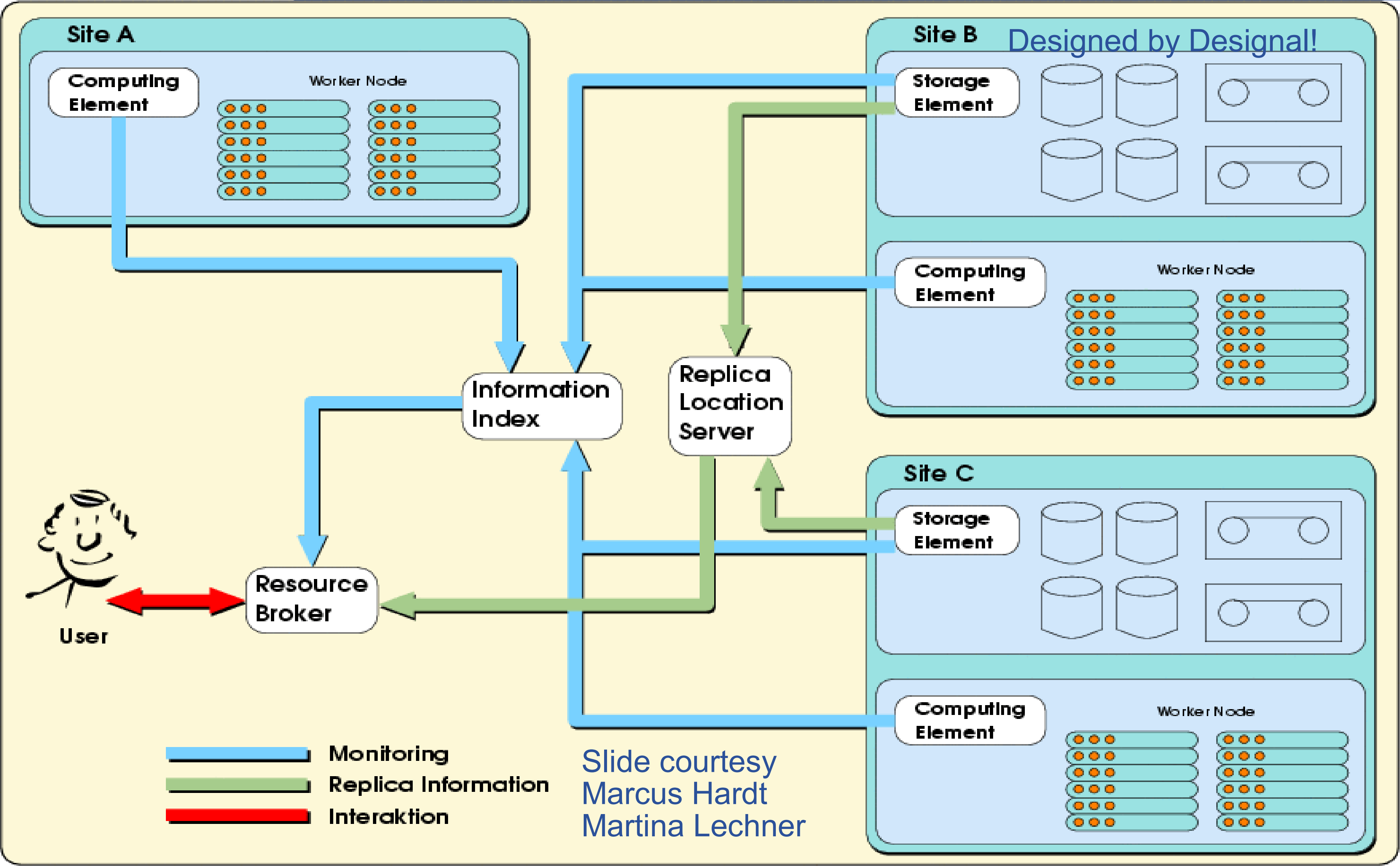
Singapore, 09.02.2006

Information Society



Workload Management System
- First Discussion -

Architecture Overview (1)



Designed by Designal!

Slide courtesy
Marcus Hardt
Martina Lechner

Overview of gLite WMS

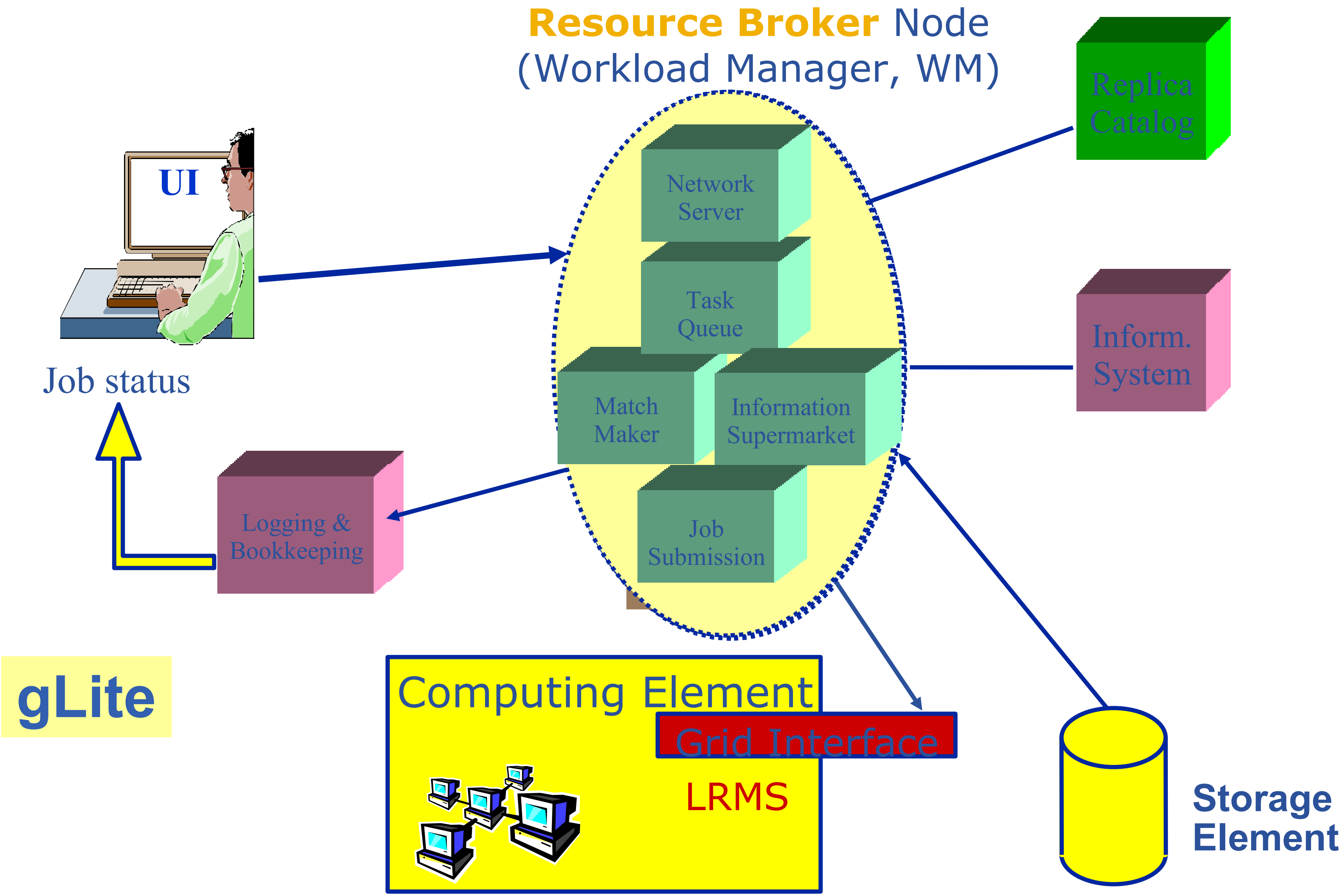
- Job Management Services
  - main services related to job management/execution are
    - computing element
      - *job management (job submission, job control, etc.), but it must also provide*
      - *provision of information about its characteristics and status*
    - workload management
      - *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

Workload Management System

- In the glite middleware a user can submit and cancel jobs, query their status, and retrieve their output. These tasks go under the name of ***Workload Management***.

- There are two different User Interfaces to accomplish these tasks. One is the Command Line Interface and the other is the Graphical User Interface.
- We will now work with the command line interface

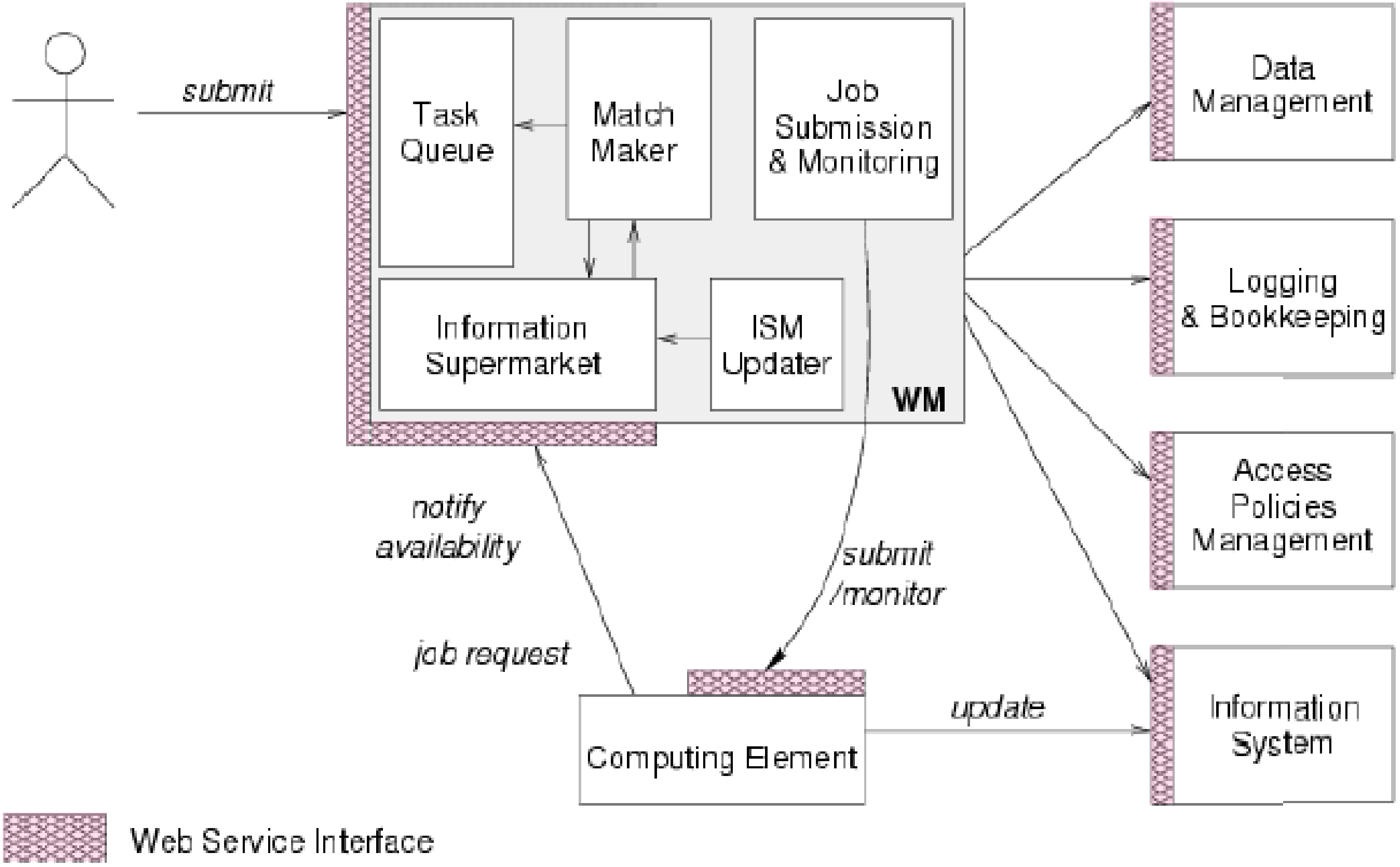
Architecture Overview (2)



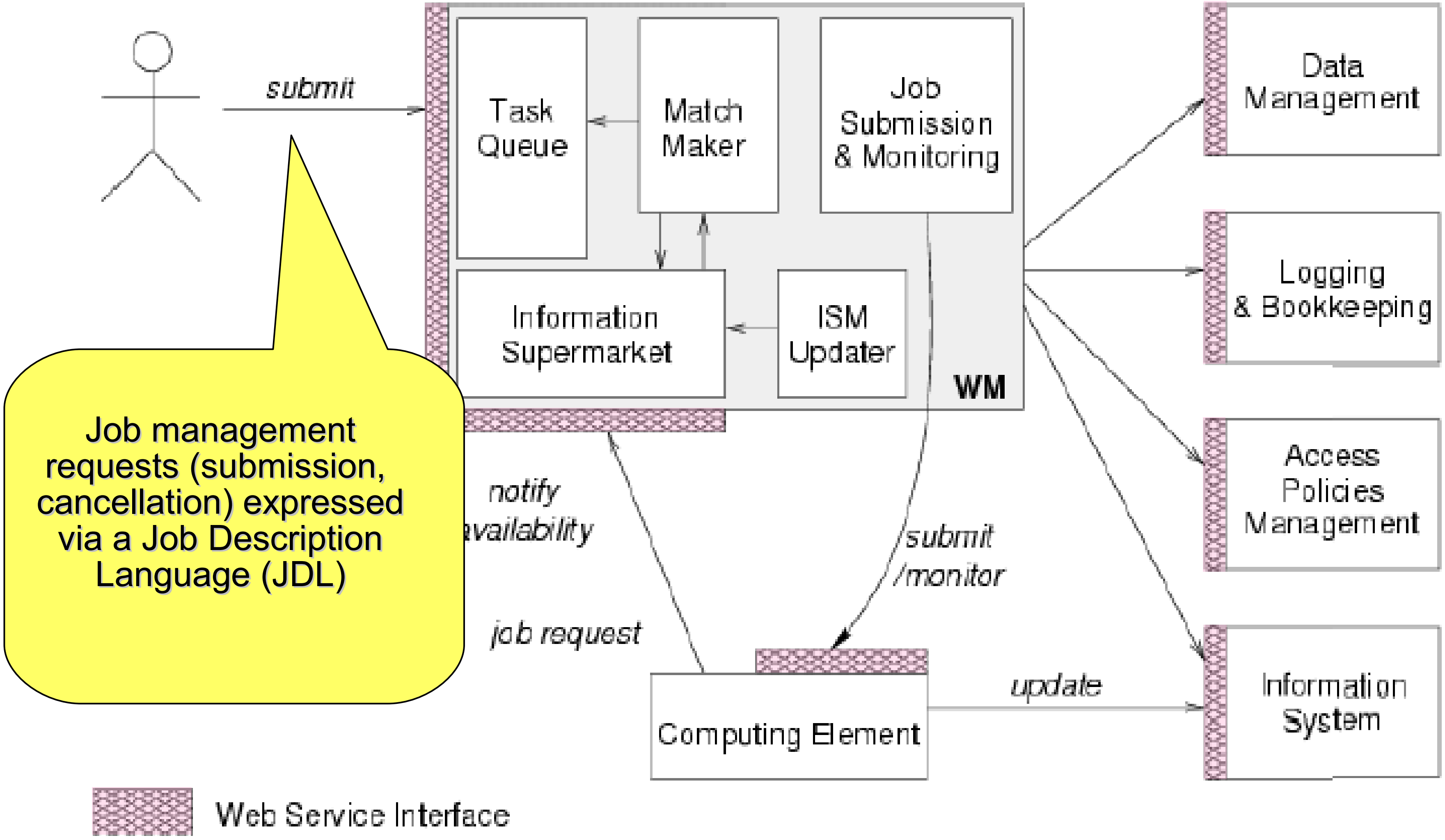
WMS’s Scheduling Policies

- 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

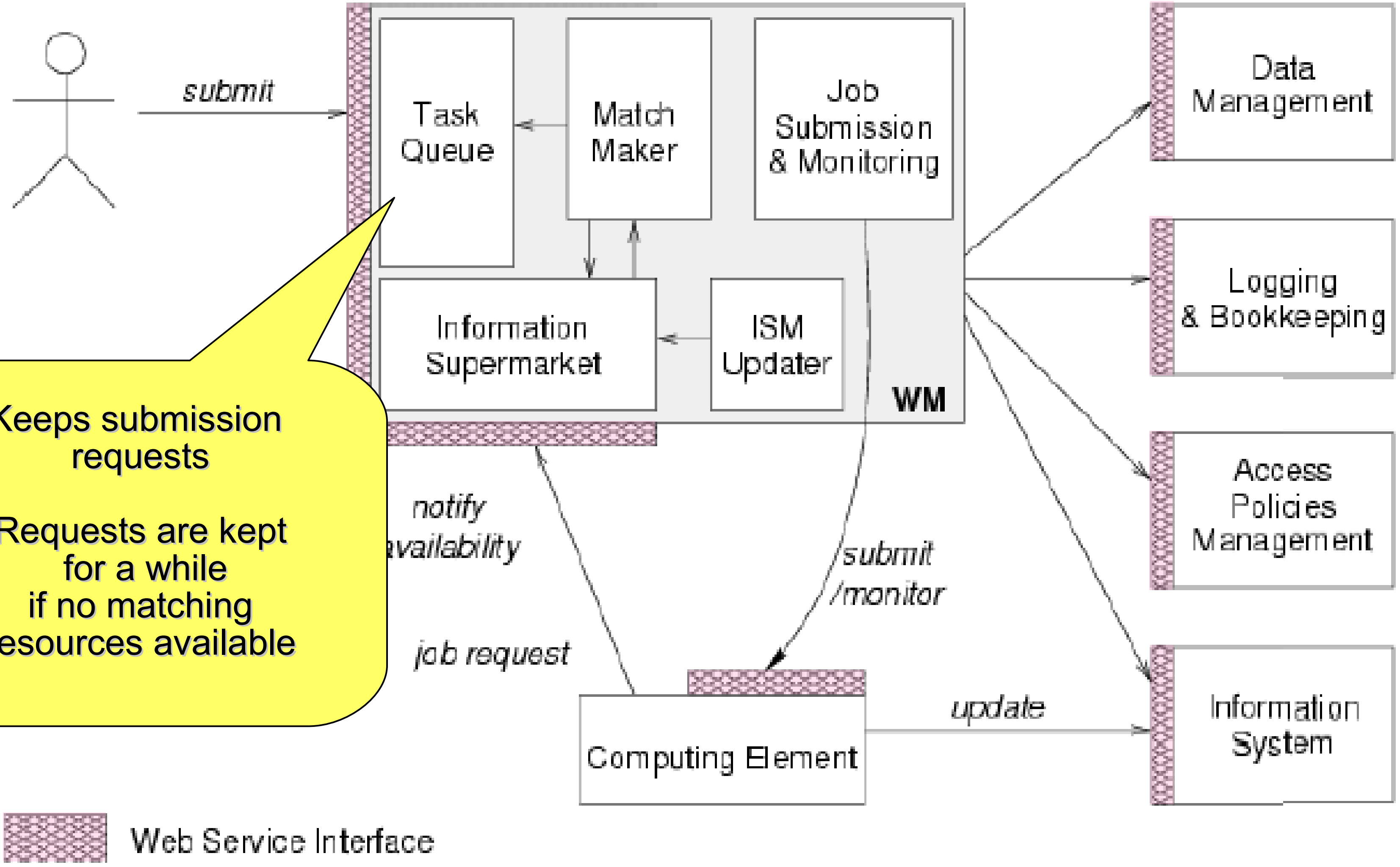


WMS's Architecture



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

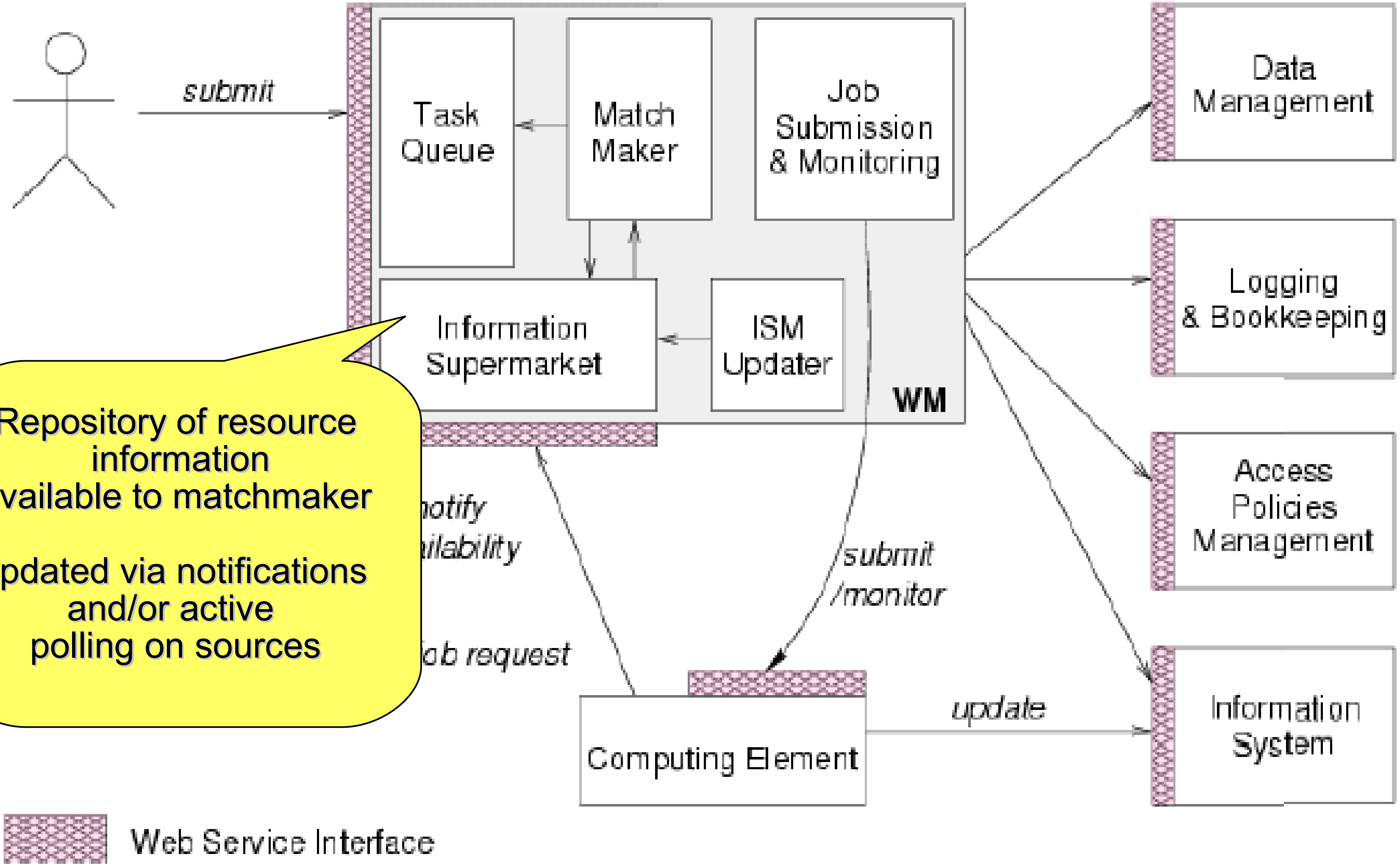
WMS's Architecture



Keeps submission requests

Requests are kept for a while if no matching resources available

WMS's Architecture

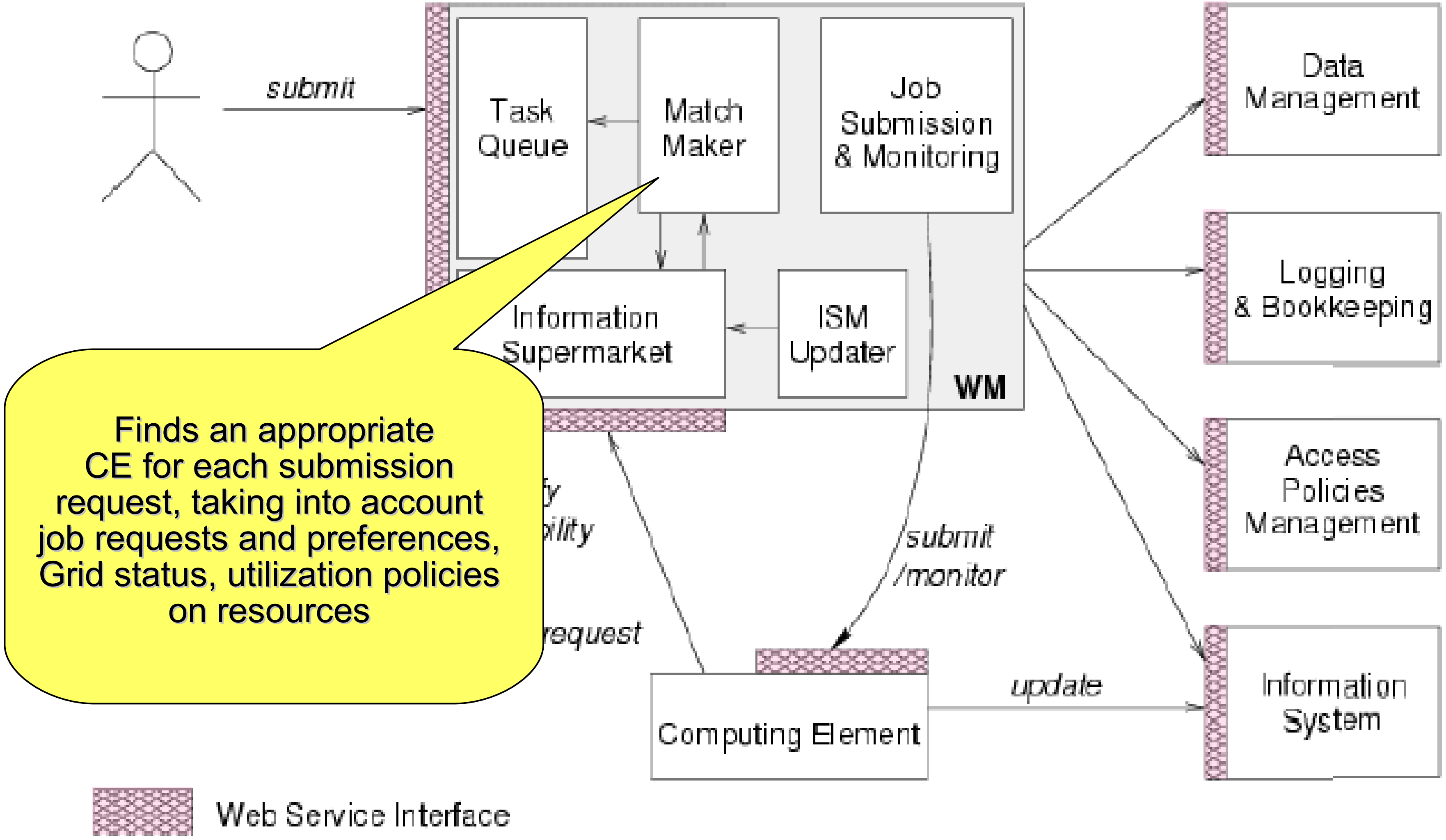


Repository of resource information
available to matchmaker

Updated via notifications and/or active
polling on sources

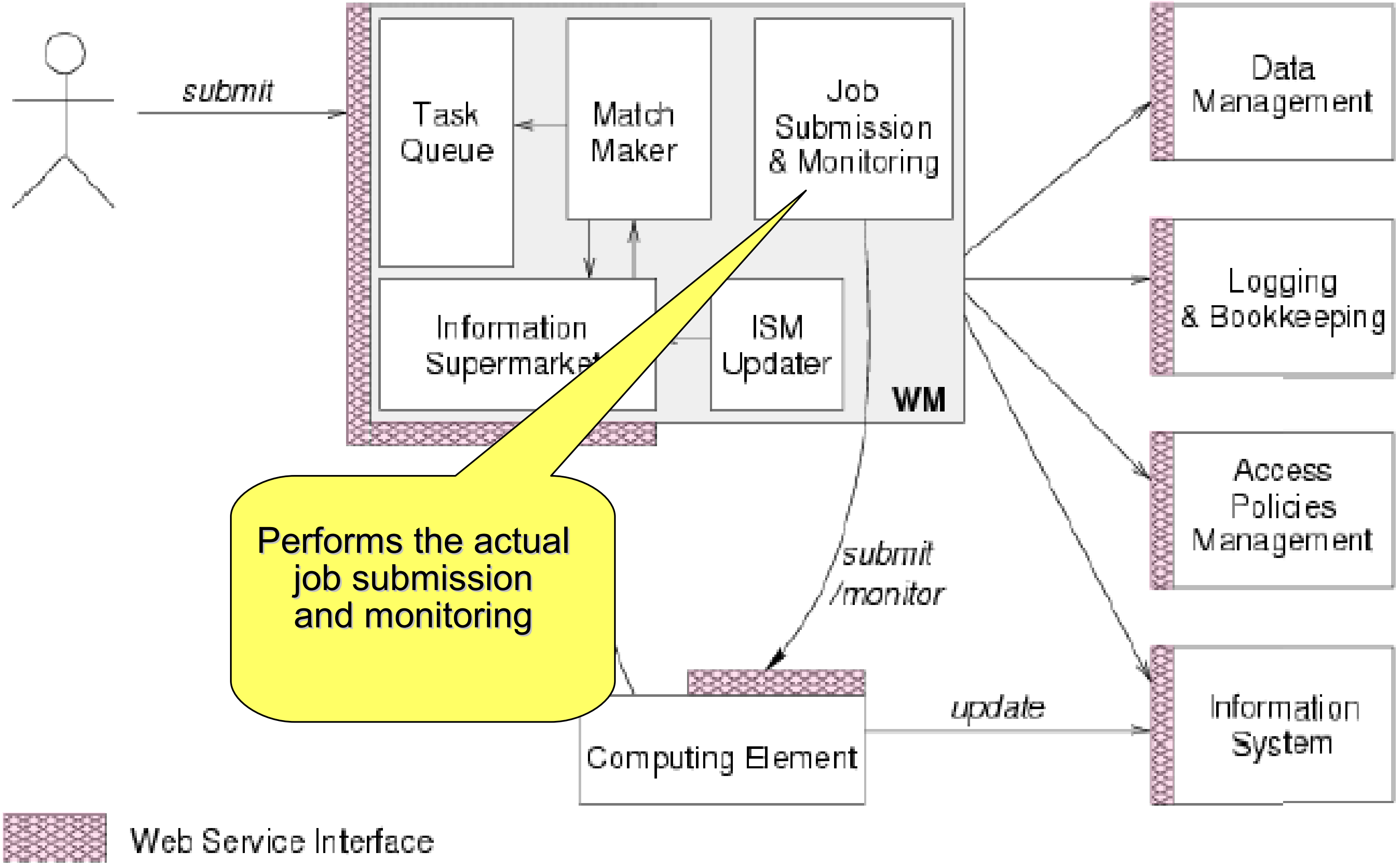
Enabling Grids for E-sciencE

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



Performs the actual job submission and monitoring

The Information Supermarket

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

The Task Queue

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

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

Job 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

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

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;
]
```

Command Line Interface

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 hello.jdl

where <jdl file> is the file containing the job description

```
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
**********************************************************************************
```

Command Line Interface (cont.)

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

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

--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.

Command Line Interface (cont.)

If the submission is successful, the output is similar to:

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.

Command Line Interface (cont.)

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 have the site administrator check the certificate of the server).

(applies to UIPnP)

Command Line Interface (cont.)

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: Ready
Status Reason: unavailable
Destination: lxshare0277.cern.ch:2119/jobmanager-pbs-infinite
reached on: Fri Aug 1 12:21:35 2003
*******************************************************************
```

Command Line Interface (cont.)

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
4 : https://lxshare0234.cern.ch:9000/Tt80pBn17AFPJyUSN9Qb7Q
a : all
q : quit
------------------------------------------------------------------------------------------
Choose one or more edg_jobId(s) in the list - [1-4]all:
```

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

Command Line Interface (cont.)

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
```

Example: All jobs that were submitted before the 17:35 of the current day, and that were not in the Cleared state are retrieved.

```
glite-job-status --all -e Cleared --to 17:00
```

Command Line Interface (cont.)

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

================================================================
```

Command Line Interface (cont.)

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

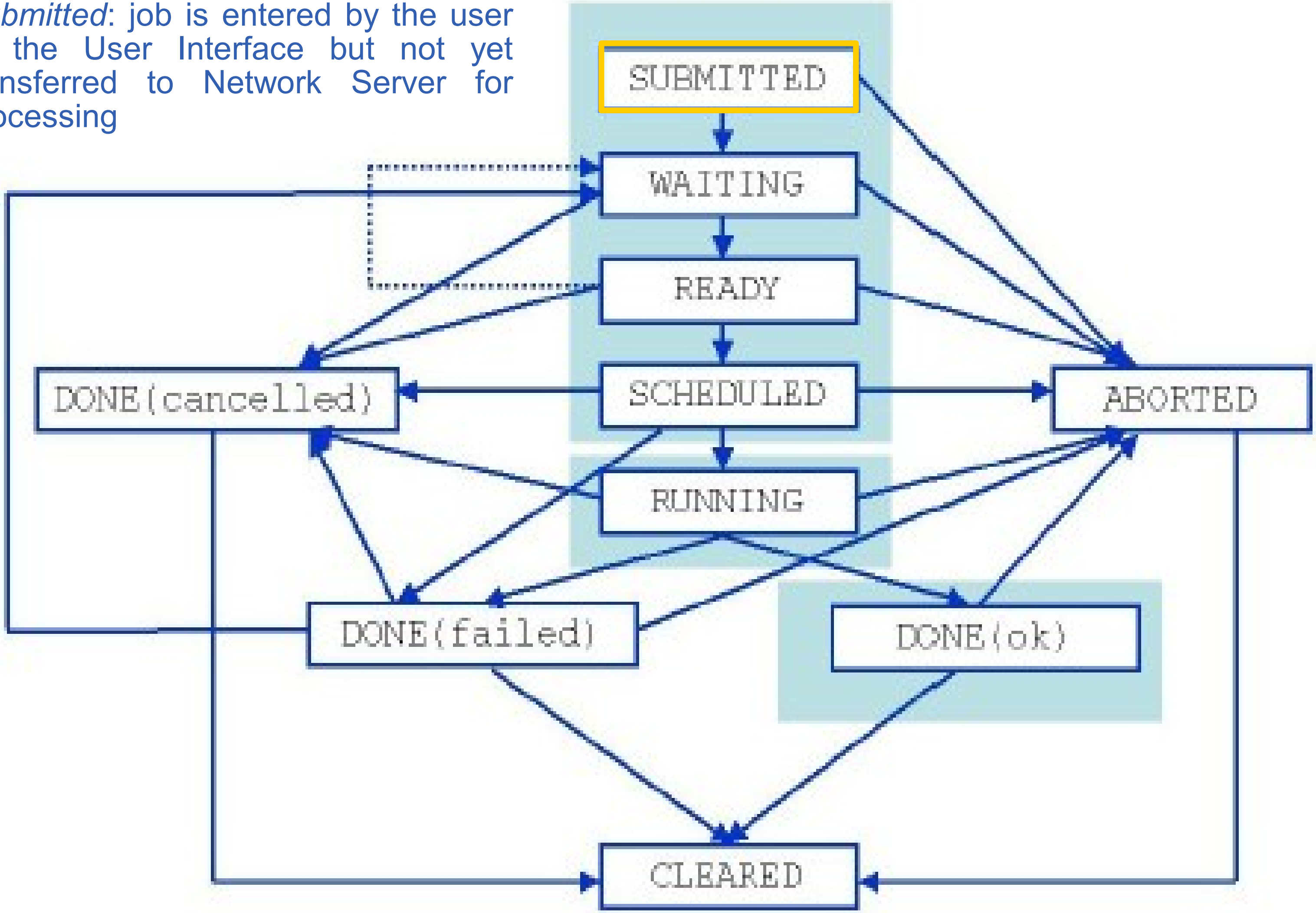
```
glite-job-output https://lxshare0234.cern.ch:9000/snPegp1YMJcnS22yF5pFlg

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/jobOutput/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.

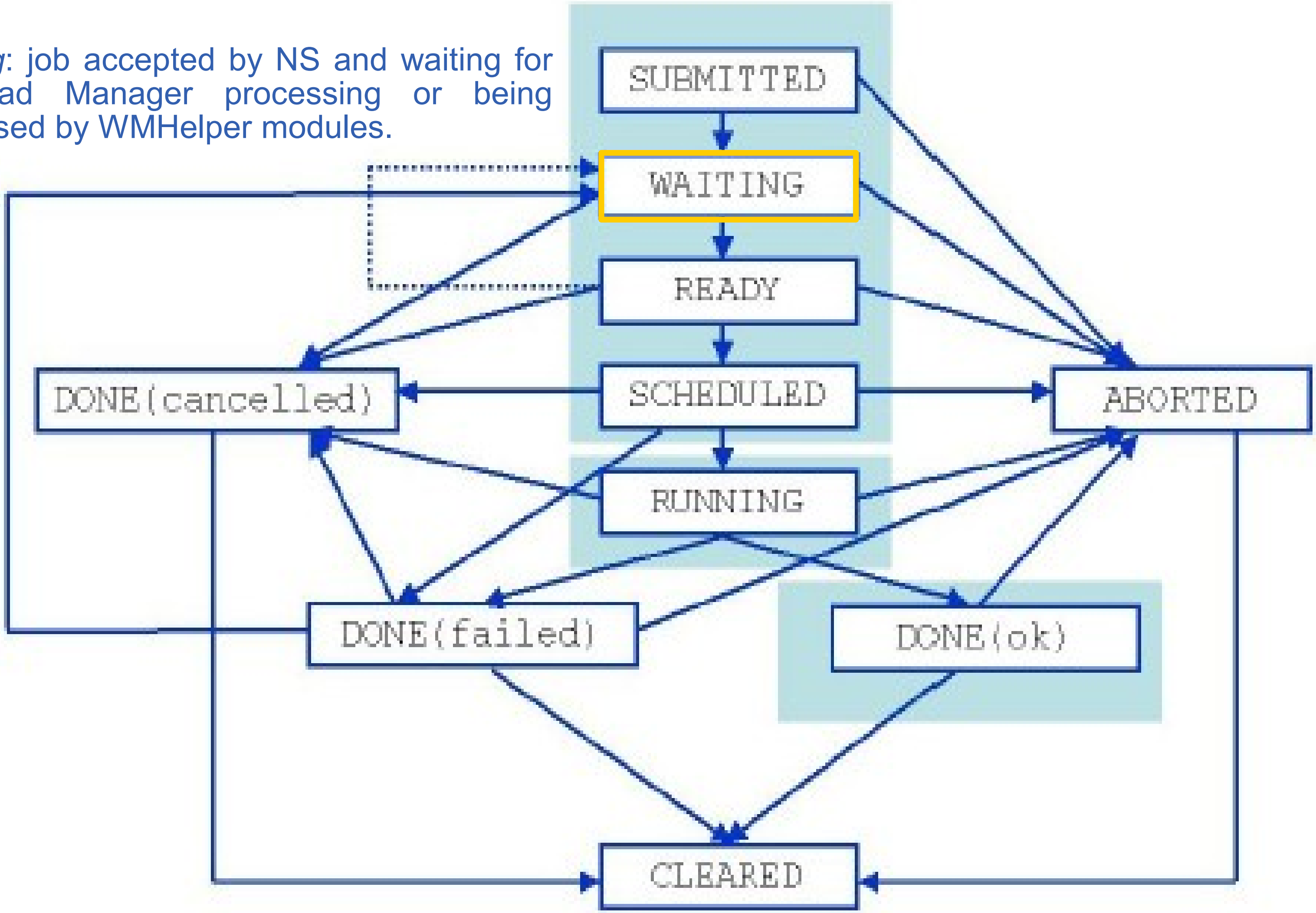
Jobs State Machine (1/9)

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

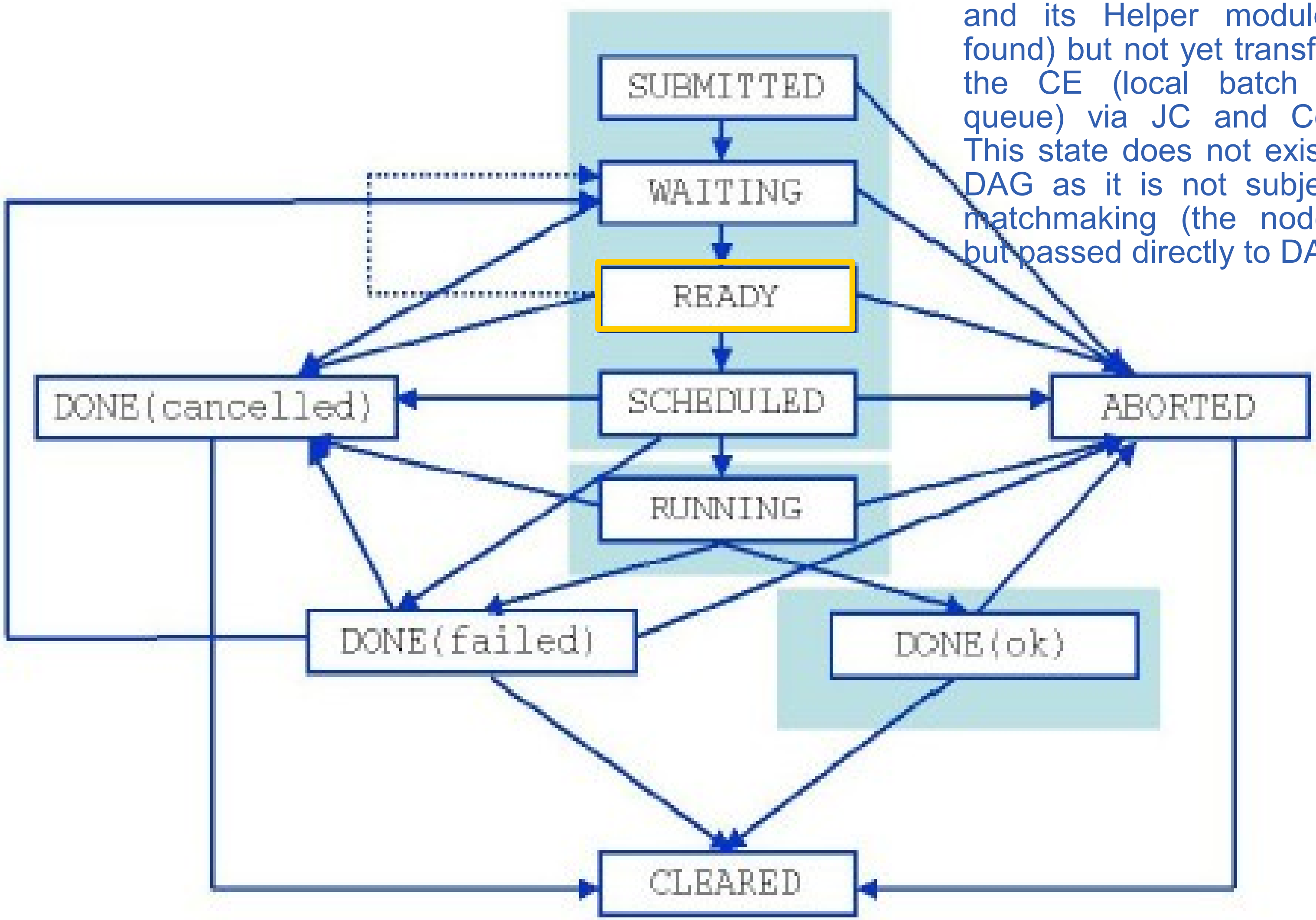


Jobs State Machine (2/9)

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

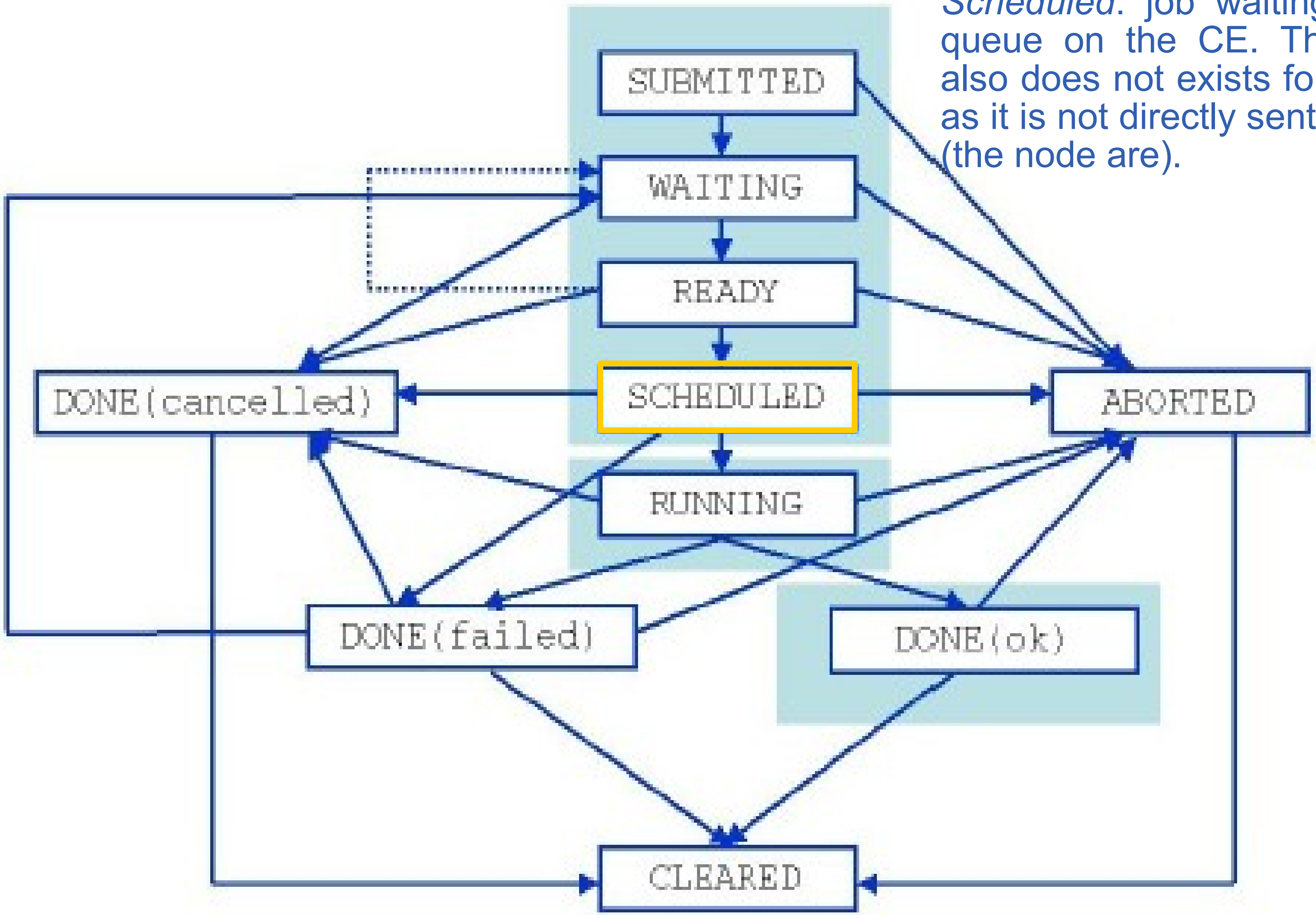


Jobs State Machine (3/9)



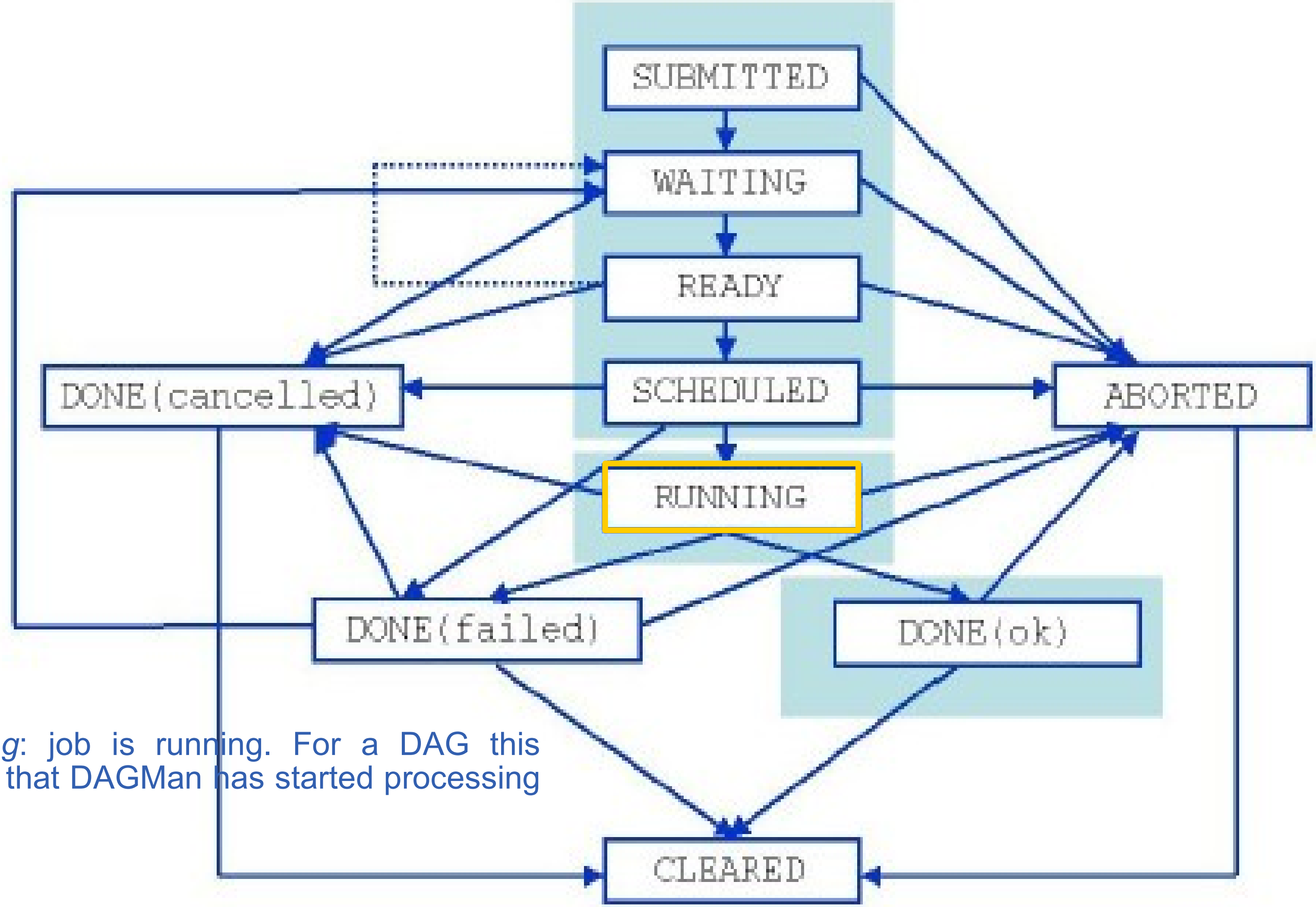
Ready: job processed by WM and its Helper modules (CE found) but not yet transferred to the CE (local batch system queue) via JC and CondorC. This state does not exists for a DAG as it is not subjected to matchmaking (the nodes are) but passed directly to DAGMan.

Jobs State Machine (4/9)



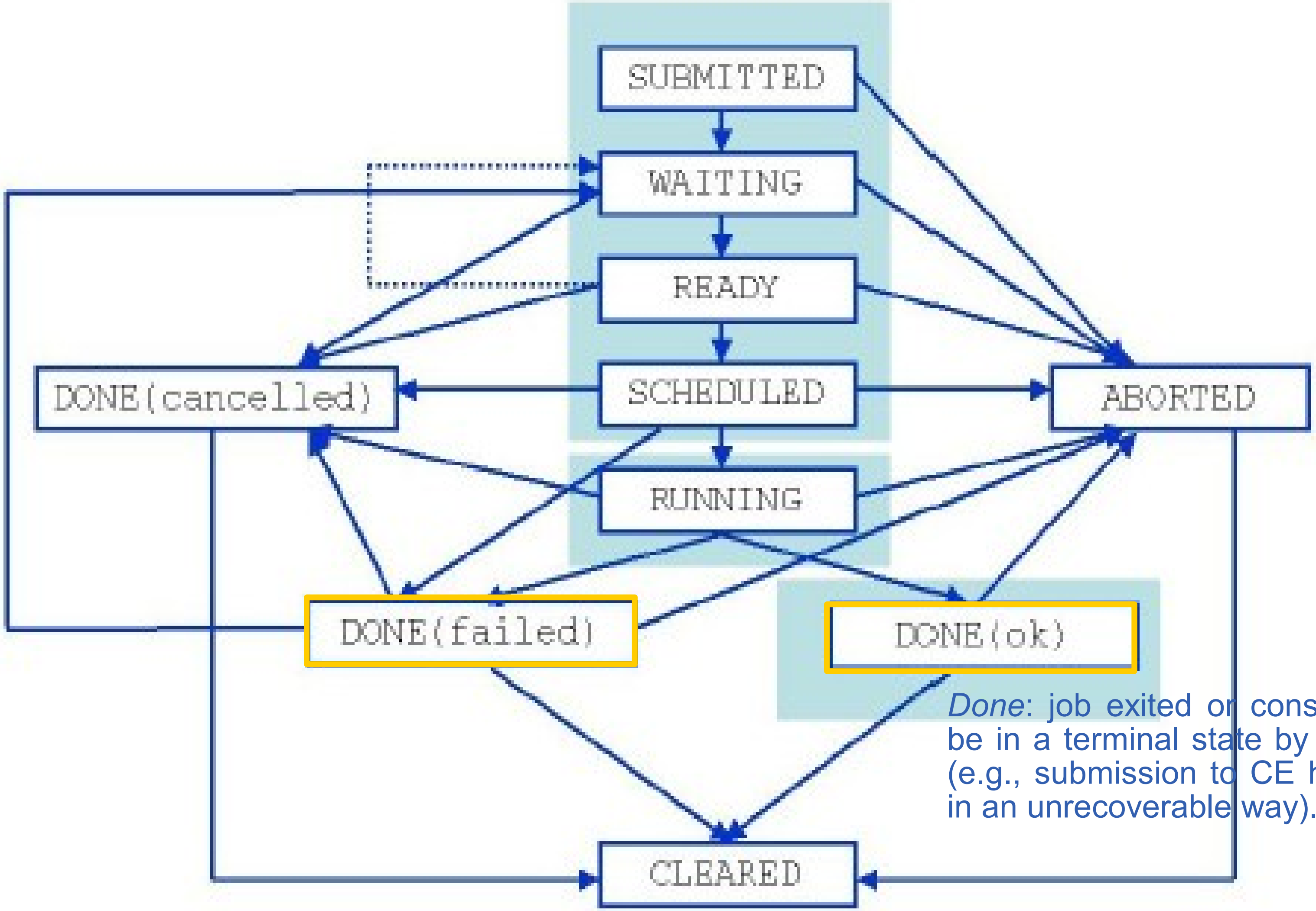
Scheduled: job waiting in the queue on the CE. This state also does not exists for a DAG as it is not directly sent to a CE (the node are).

Jobs State Machine (5/9)



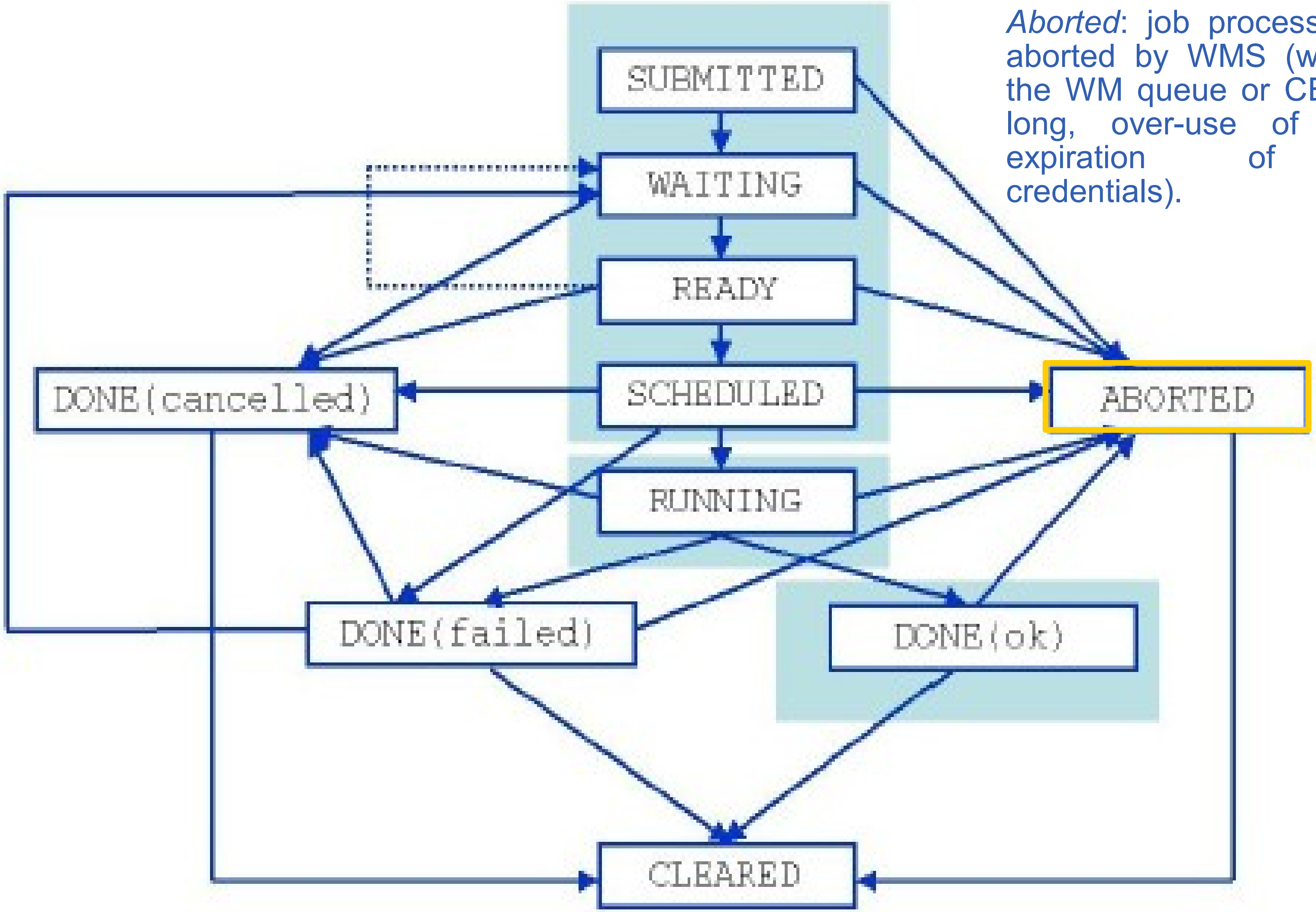
Running: job is running. For a DAG this means that DAGMan has started processing it.

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).

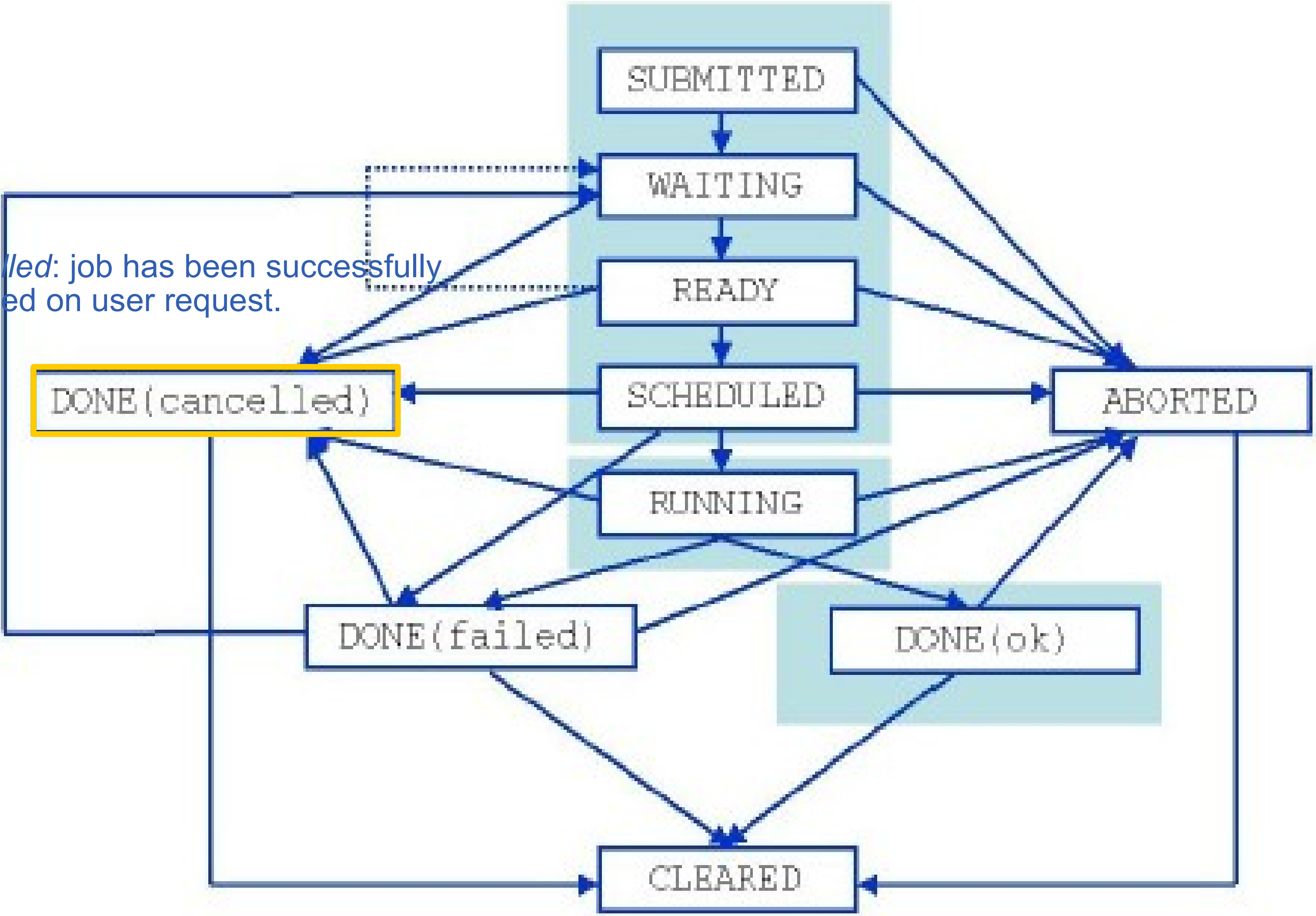
Jobs State Machine (7/9)



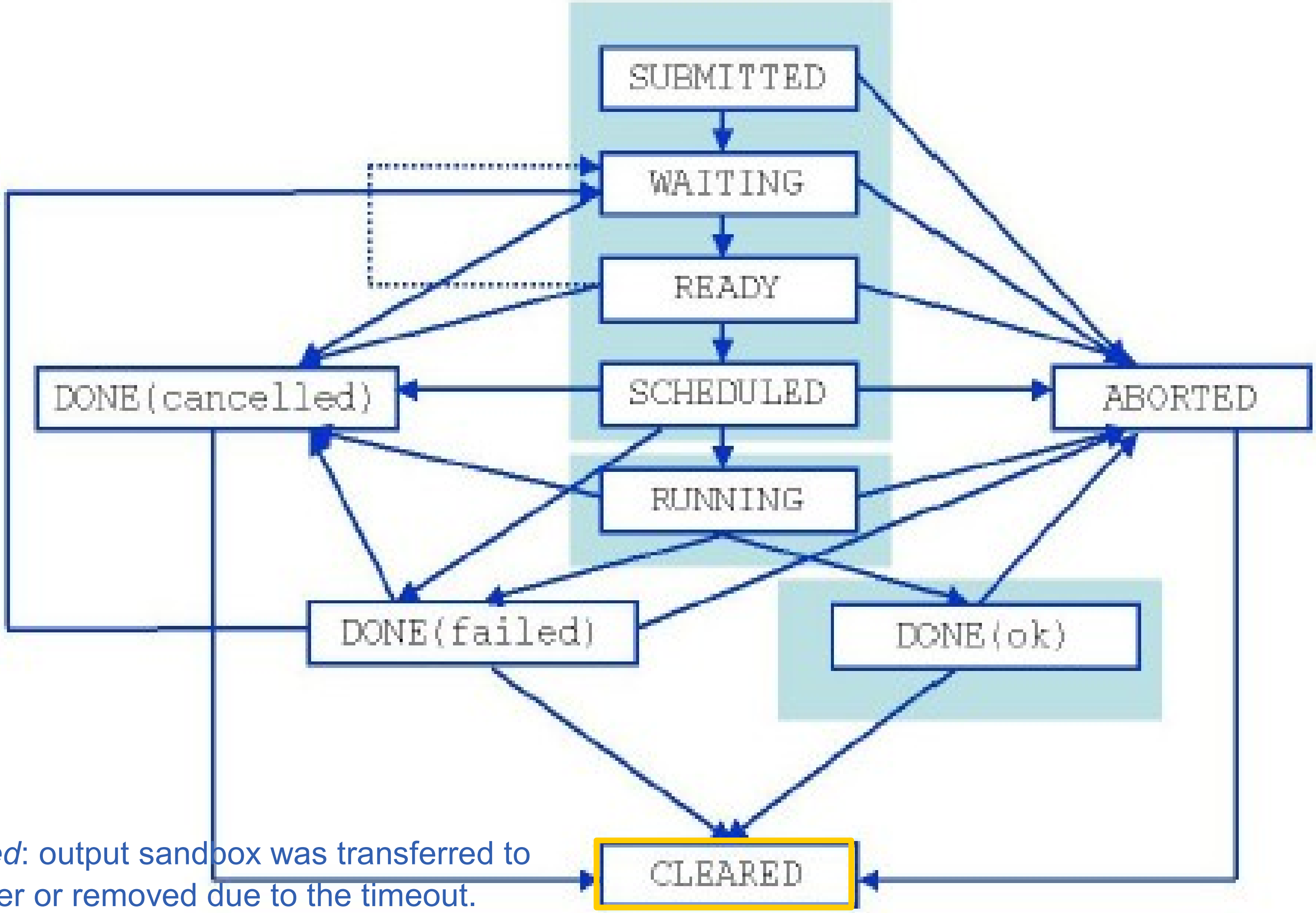
Aborted: job processing was aborted by WMS (waiting in the WM queue or CE for too long, over-use of quotas, expiration of user credentials).

Jobs State Machine (8/9)

Cancelled: job has been successfully canceled on user request.



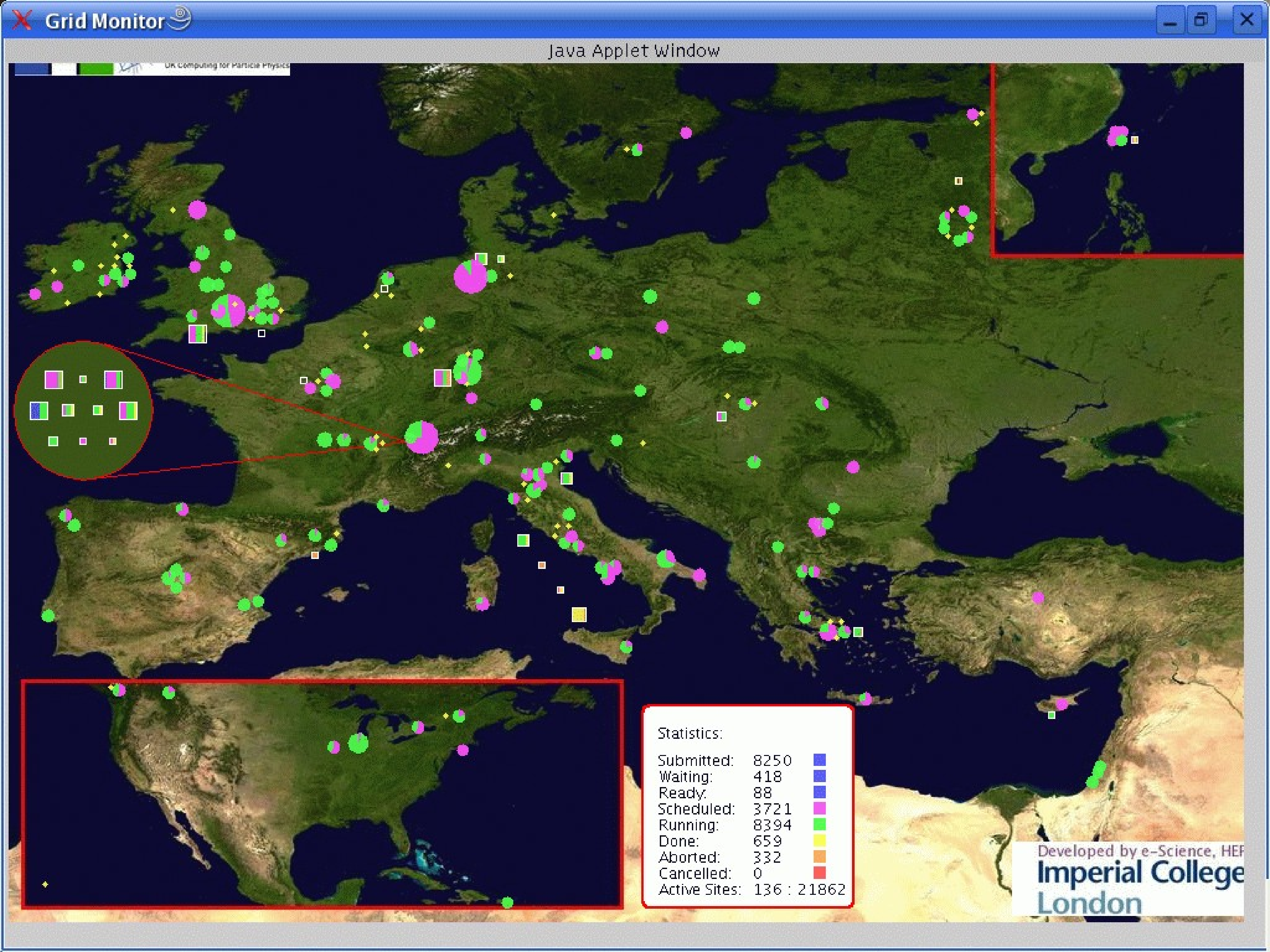
Jobs State Machine (9/9)



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

Grid Monitoring

- LCG-2 Grid Monitor – GridPP / Imperial College London http://gridportal.hep.ph.ic.ac.uk/rtm/
  - Nice graphical and animated view of the Grid
- INFN GridIce : http://gridice2.cnaf.infn.it:50080/gridice/geo/geo.php
  - In-depth information about jobs, VOs,...
  - Graphical overview



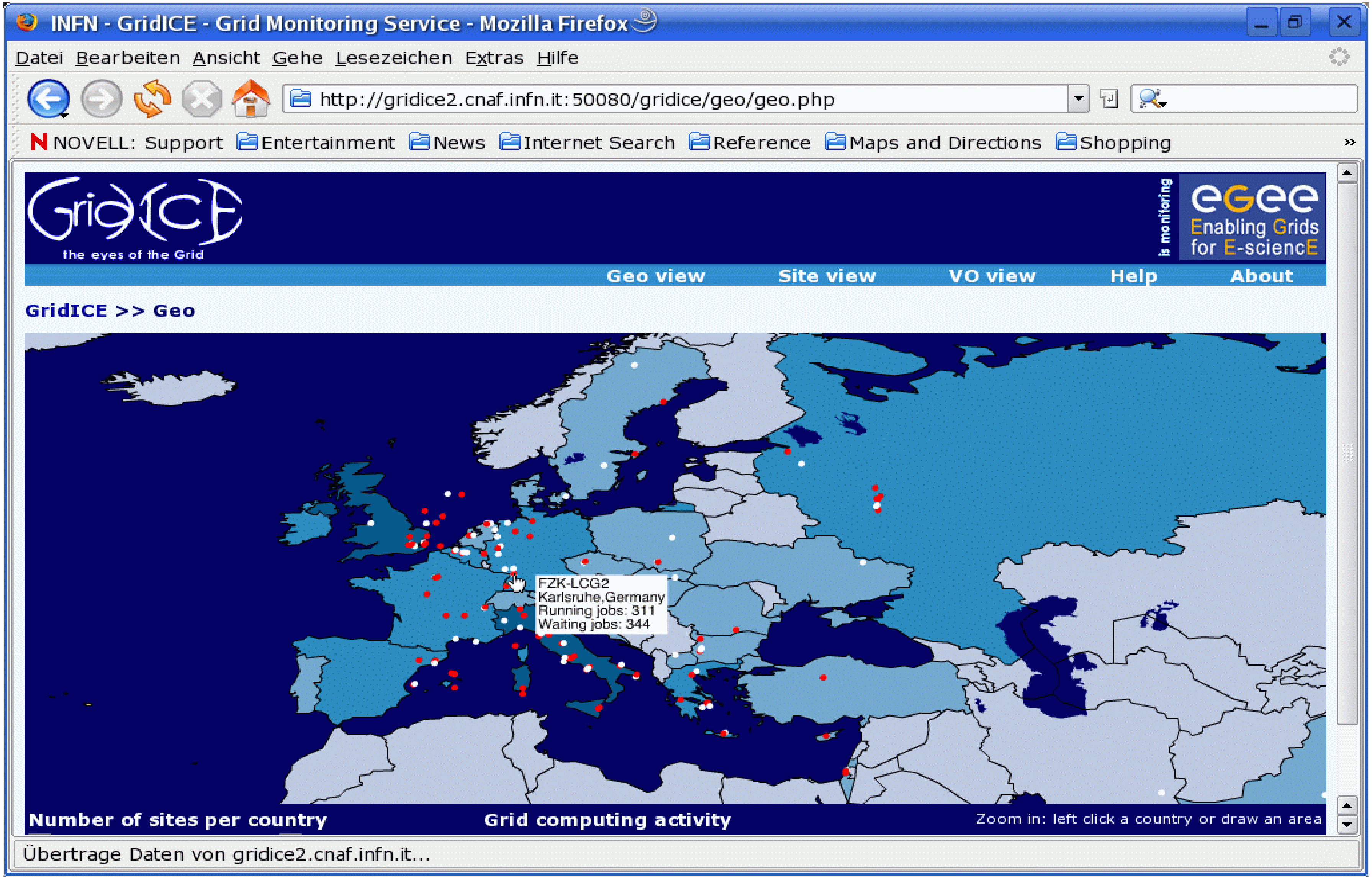
Grid Monitor
Java Applet Window
UK Computing for Particle Physics

```
Statistics:

Submitted:    8250          🟦
Waiting:      418           🟦
Ready:        88            🟦
Scheduled:    3721          🟪
Running:      8394          🟩
Done:         659           🟨
Aborted:      332           🟧
Cancelled:    0             🟥
Active Sites: 136 : 21862
```

Developed by e-Science, HEP
Imperial College
London

GridIce



INFN - GridICE - Grid Monitoring Service - Mozilla Firefox

Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://gridice2.cnaf.infn.it:50080/gridice/geo/geo.php
NOVELL: Support Entertainment News Internet Search Reference Maps and Directions Shopping
GridICE the eyes of the Grid
is monitoring
egee Enabling Grids for E-sciencE
Geo view
Site view
VO view
Help
About

GridICE >> Geo

FZK-LCG2
Karlsruhe,Germany
Running jobs: 311
Waiting jobs: 344

Number of sites per country
Grid computing activity
Zoom in: left click a country or draw an area

Übertrage Daten von gridice2.cnaf.infn.it...

Workload Management System
- Practicals -

User Interface Access

Host: glite-tutor.ct.infn.it

Username: singaporeXX (XX=01…30)

Password: GridSINXX (XX=01…30)

PassPhrase: SINGAPORE

GENIUS: https://glite-tutor.ct.infn.it

hello.jdl

Create JDL file similar to:

```
Type = "Job";
JobType = "Normal";
Executable = "/bin/echo";
StdOutput = "hello.out";
StdError = "hello.err";
OutputSandbox = {"hello.err","hello.out"};
Arguments = "Hello World";
RetryCount = 7;
```

Exercises

- Check where hello.jdl can be executed
- Submit a single “hello world” example
- Submit multiple times, e.g. with

```
for i in `seq 1 5`; do
   glite-job-submit -o myjobs.id hello.jdl;
done
```

- Have a look at myjobs.id
- Follow the states of a job, e.g. with

```
watch --interval=20 glite-job-status <job-id>
```

- Check state of all jobs
- Cancel a job
- Check output

Exercises

- Create a bash script which displays hostname and current date
- Test script locally
- Save the script as yourscript.sh

Simple job submission

cp hello.jdl exercise1.jdl

- Modify exercise1.jdl file
  - **Instead of running hostname command, run a bash script you have just created (yourscript.sh).**
- Submit the job, check its status and when done retrieve the output

JDL – Examples

yourscript.sh

```
#!/bin/bash
/bin/hostname
/bin/date
```

exercise1.jdl

```
Type = "Job";
JobType = "Normal";
Executable = "/bin/bash";
StdOutput = “std.out";
StdError = “std.err";
InputSandbox = {“yourscript.sh"};
OutputSandbox = {“std.err",“std.out"};
Arguments = "yourscript.sh";
```

JDL – Requirements

```
#Insert a requirement to parse only the short queues.
Requirements = (other.GlueCEPolicyMaxWallClockTime <
   720);

#Insert a requirement to parse only the long queues.
Requirements = (other.GlueCEPolicyMaxWallClockTime >
   1440);

#Insert a requirement to parse only the infinite
   queues.
Requirements = (other.GlueCEPolicyMaxWallClockTime >
   2880);

#Insert a requirement to stear the execution on a
   particular CE Queue.
Requirements = other.GlueCEUniqueID ==
   "grid010.ct.infn.it:2119/jobmanager-lcgpbs-long";
```

Exercises

- **Simple job using Requirements**
  - Modify exercise1.jdl file so that user with a even workstation number will submit their job on a “long” queue, and the other to an “infinite” one
  - Verify the list of CE suitable for this job execution
  - Submit the job, check its status and retrieve the output

JDL with Requirements

```
Type = "Job";
JobType = "Normal";
Executable = "/bin/sh";
Arguments = "start_povray_cubo.sh";
StdOutput = "povray_cubo.out";
StdError = "povray_cubo.err";
InputSandbox = {"start_povray_cubo.sh","cubo.pov"};
OutputSandbox =
    {"povray_cubo.out","povray_cubo.err","cubo.png"};

Requirements = Member("POVRAY-3.5",other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

start_povray_cubo.sh

```
#!/bin/bash
mv cubo.pov OBJECT.POV #rename input file
/usr/bin/povray /usr/share/povray-3.5/ini/res800.ini  #run povray
mv OBJECT.png cubo.png #rename output file
```

Exercises

- Modify povray_cubo.jdl, specifying the resource for execution into the jdl file
- Check job status and when done retrieve the output
- Display .png file obtained as output, using ImageMagick (if possible – transfer file on Windows machines or leave out this step)

Exercises

Run an ls command on the selected resource.

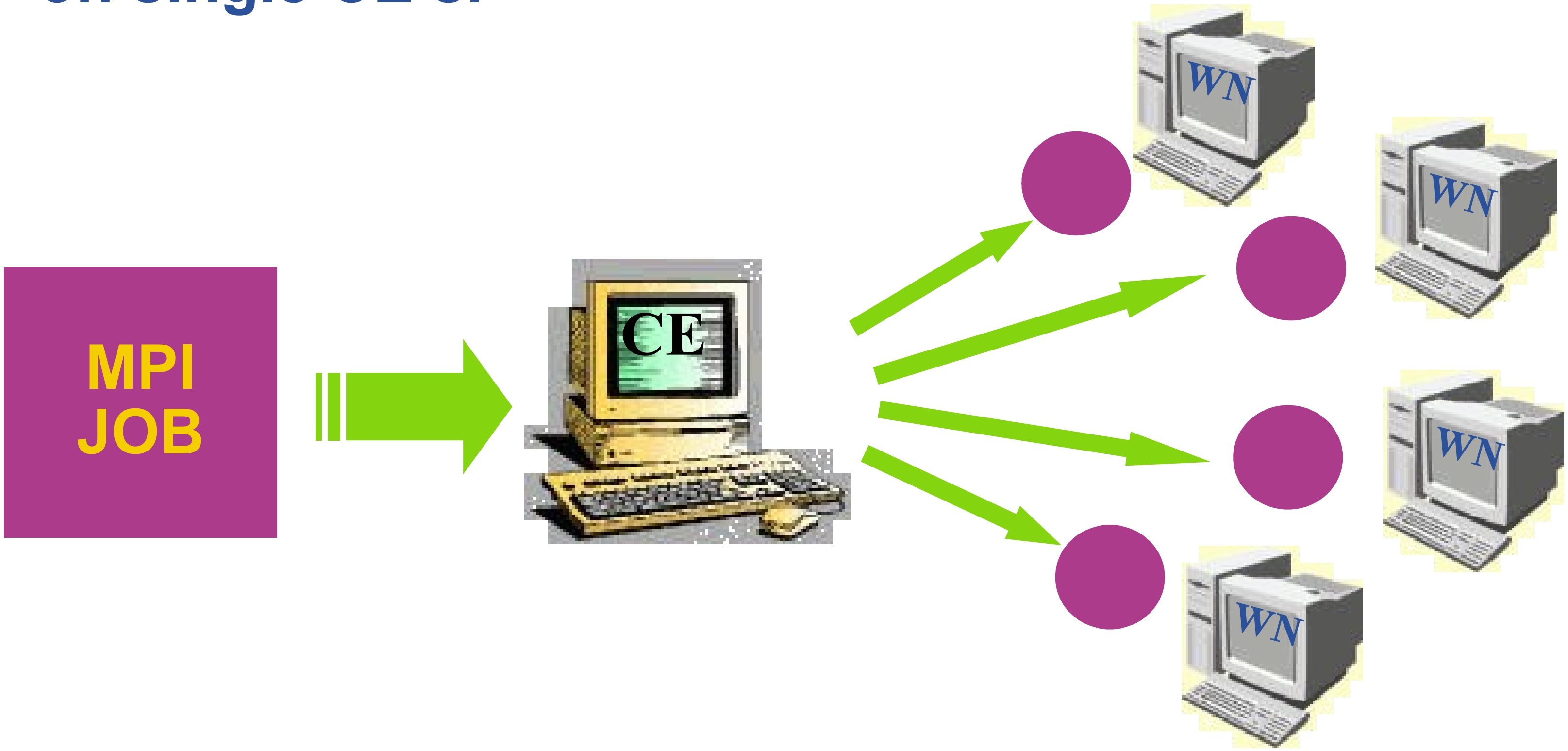
```
[
  Executable = "ls.sh";
  Arguments = "-alt";
  StdError = "stderr.log";
  StdOutput = "stdout.log";
  InputSandbox = "ls.sh";
  OutputSandbox = {"stderr.log", "stdout.log"};
]
```

ls.sh

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

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*

mpi.jdl

```
Type = "Job";
JobType = "MPICH";
Executable = "MPItest.sh";
Arguments = "cpi 2";
NodeNumber = 2;
StdOutput = "test.out";
StdError = "test.err";
InputSandbox = {"MPItest.sh","cpi"};
OutputSandbox = {"test.err","test.out","executable.out"};
Requirements = other.GlueCEInfoLRMSType == "PBS" ||
    other.GlueCEInfoLRMSType == "LSF";
```

MPItest.sh

```
#!/bin/sh
#

# this parameter is the binary to be executed
EXE=$1
# this parameter is the number of CPU's to be reserved for parallel execution
CPU_NEEDED=$2

# prints the name of the master node
echo "Running on: $HOSTNAME"

if [ -f "$PWD/.BrokerInfo" ] ; then
        TEST_LSF=`edg-brokerinfo getCE | cut -d/ -f2 | grep lsf`
else
        TEST_LSF=`ps -ef | grep sbatchd | grep -v grep`
fi
```

MPItest.sh (cont.)

```
if [ "x$TEST_LSF" = "x" ] ; then
        # prints the name of the file containing the nodes allocated for parallel execution
        echo "PBS Nodefile: $PBS_NODEFILE"
        # print the names of the nodes allocated for parallel execution
        cat $PBS_NODEFILE
        HOST_NODEFILE=$PBS_NODEFILE
else
        # print the names of the nodes allocated for parallel execution
        echo "LSF Hosts: $LSB_HOSTS"
        # loops over the nodes allocated for parallel execution
        HOST_NODEFILE=`pwd`/lsf_nodefile.$$

        for host in ${LSB_HOSTS}  do
                host=`host $host | awk '{ print $1 } '`
                echo $host >> ${HOST_NODEFILE}
        done
fi
```

MPItest.sh (cont.)

```
# prints the working directory on the master node
echo "Current dir: $PWD"

for i in `cat $HOST_NODEFILE` ; do
    echo "Mirroring via SSH to $i"
    # creates the working directories on all the nodes allocated for parallel execution
    ssh $i mkdir -p `pwd`
    # copies the needed files on all the nodes allocated for parallel execution
    /usr/bin/scp -rp ./* $i:`pwd`
    # checks that all files are present on all the nodes allocated for parallel execution
    echo `pwd`
    ssh $i ls `pwd`
    # sets the permissions of the files
    ssh $i chmod 755 `pwd`/$EXE
    ssh $i ls -alR `pwd`
done
```

MPItest.sh (cont.)

```
# execute the parallel job with mpirun
echo "*********************************"
echo "Executing $EXE"
chmod 755 $EXE
mpirun -np $CPU_NEEDED -machinefile $HOST_NODEFILE
    `pwd`/$EXE > executable.out
echo "*********************************"
```

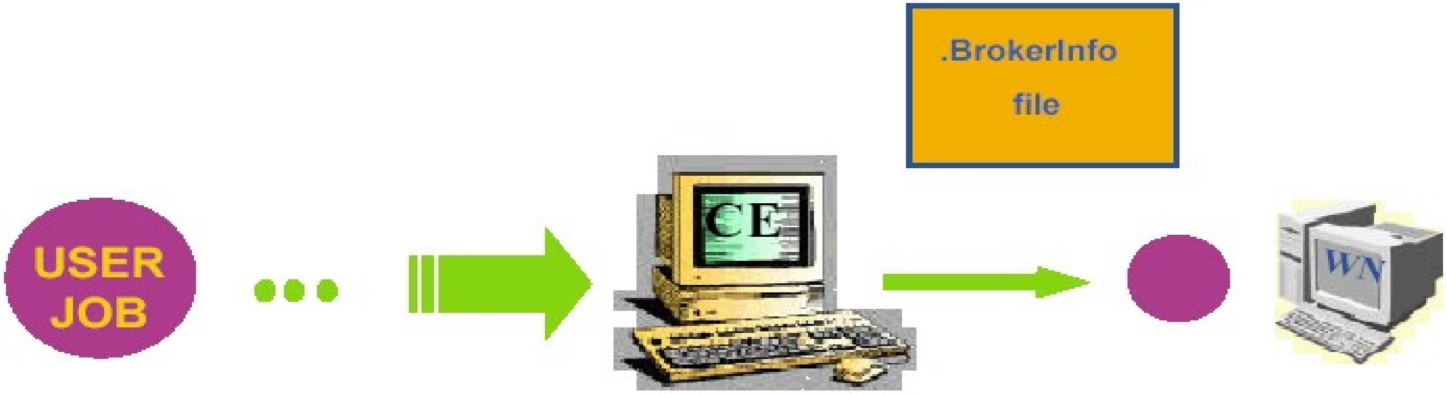
.BrokerInfo file

- It is a mechanism by which a job can access at some information about itself…at execution time!
- The Resource Broker creates and attaches this file to the job when it is ready to be transfered to the resource that best matches the request.
- Two ways for parsing elements from .BrokerInfo file:

1)Directly from the Worker Node at execution time;

2)From User Interface, but only if you have inserted the name of “.BrokerInfo” file in the JDL’s OutputSandbox, and you have just retrieved job output, once that job has been Done;

edg/glite-brokerinfo [options] function param



Example of .BrokerInfo file

```
[       ComputingElement =
          [
             CloseStorageElements =
               {
                 [
                         GlueSAStateAvailableSpace =
     14029724;
                         GlueCESEBindCEAccesspoint
     = "/flatfiles/SE00";
                         mount =
     GlueCESEBindCEAccessPoint;
                         name =
     "grid003.cecalc.ula.ve";
                         freespace =
     GlueSAStateAvailableSpace
                   ]
               };
             name =
     "grid006.cecalc.ula.ve:2119/jobmanager-
     lcgpbs-infinite"
          ];
      InputFNs =
         {
         };
      StorageElements =
         {
         };
      VirtualOrganisation = "gilda"    ]
```

edg-brokerinfo getCE

edg-brokerinfo getDataAccessProtocol

edg-brokerinfo getInputData

edg-brokerinfo getSEs

edg-brokerinfo getCloseSEs

edg-brokerinfo getSEMountPoint <SE>

edg-brokerinfo getSEFreeSpace <SE>

edg-brokerinfo getSEProtocols <SE>

edg-brokerinfo getSEPort <SE> <Protocol>

edg-brokerinfo getVirtualOrganization

edg-brokerinfo getAccessCost

Exercises

Exercise 1

- Create a file called **startScriptBrokerInfo.sh** with this content:

```
#!/bin/sh

MY_NAME=“Your name"
WORKER_NODE_NAME=`hostname`


echo "Hello $MY_NAME, from $WORKER_NODE_NAME"
ls -a
echo "This job is running on this CE: "
/opt/edg/bin/edg-brokerinfo getCE
```

Exercises

Exercise 2

- Create a file called **scriptBrokerInfo.jdl** with this content:

```
[
   Executable = "startScriptBrokerInfo.sh";
   StdOutput = "std.out";
   StdError = "std.err";
   VirtualOrganisation = "gilda";
   InputSandbox = {"startScriptBrokerInfo.sh"};
   OutputSandbox = {"std.out","std.err",".BrokerInfo"};
   RetryCount = 7;
 ]
```

Exercises

1. Replace your name in the file script startScriptBrokerInfo.sh;
2. Submit / Query the status / Retrieve Output the JDL file scriptBrokerInfo.jdl;
3. In JobOutput folder, go into directory of the job that you have just retrieved and inspect the .BrokerInfo file.
4. Take practice with the edg-brokerinfo command and its functions.

Exercises

This exercise allows user to submit a C program.

Modify c_sample.c file as follow:

```
#include <stdio.h>
int main(int argc, char **argv)
{
     printf("\n\n\n");
     printf("Hello !\n");
     printf("Welcome to  EGEE Tutorial, Melbourne\n\n\n");
     exit(0);
}
```

Exercises (cont.)

Compile your script using make.

Submit the c_sample.jdl job to the grid using the *glite-job-submit c_sample.jdl* command.

Inspect the status and retrieve its output when the job is finished.

Exercises

Modify c_sample.c file as follow:

```
#include <stdio.h>
int main(int argc, char **argv)
{
  char *name = argv[1];
  printf("\n\n\n");
  printf("Hello !\n");
  printf("Welcome to EGEE Tutorial\n\n\n");
  exit(0);
}
```

Exercises (cont.)

Compile your script with make.

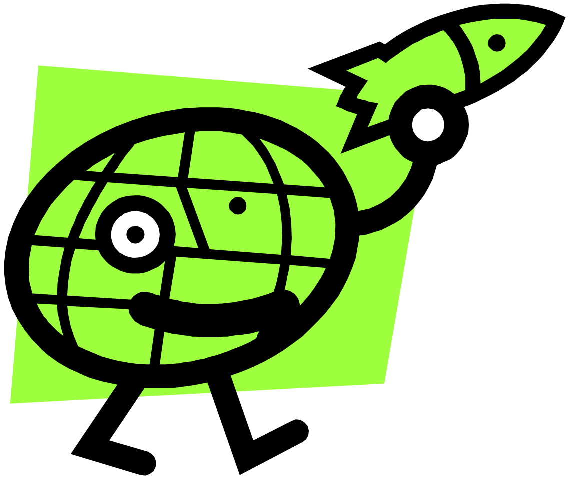
Modify the start_c_sample.sh script as follow:

```
#!/bin/sh
chmod 777 c_sample
./c_sample $1
```

Modify c_sample.jdl's Arguments as follow:

```
Arguments = "start_c_sample.sh <Your Name>";
```

Submit, inspect the status and retrieve its output when the job is finished.



Demonstration

DataGrid Accounting System

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.**

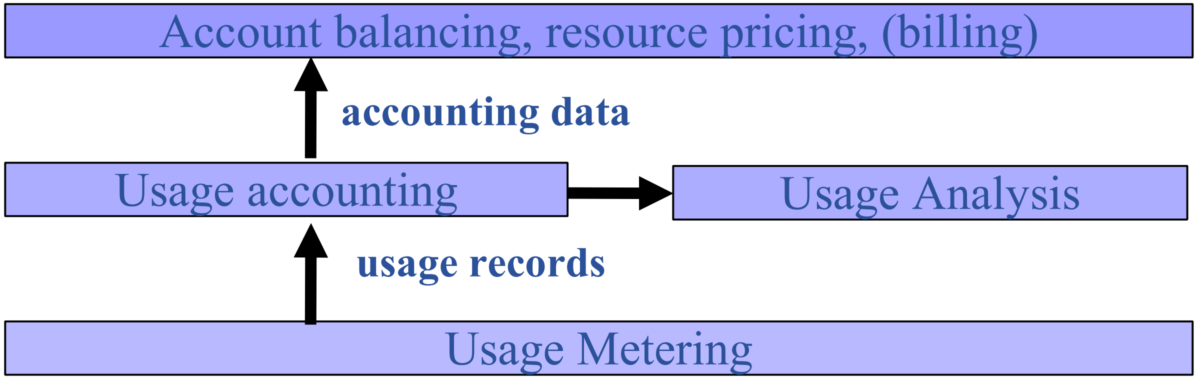
Different types of users are interested in different views of the usage records.

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 independent layers.



Exercise

View user Credits

$ dgas-check-balance

```
User: Giuseppe La Rocca
E-mail: giuseppe.larocca@ct.infn.it
Subject: /C=IT/O=GILDA/OU=Personal Certificate/L=INFN
    Catania/CN=Giuseppe La
    Rocca/Email=giuseppe.larocca@ct.infn.it
Assigned credits (0=infinite): 0
Booked credits: 0
Used credits: 451
Used wall clock time (sec): 1187
Used CPU time (sec): 264
Accounted jobs: 22
```

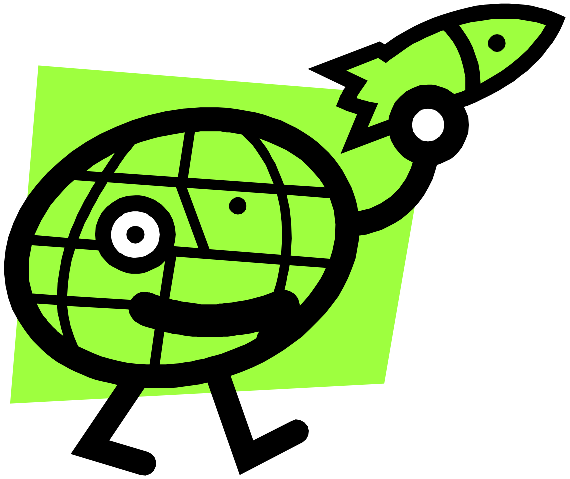
Exercise

View CE Price

Usage: dgas-check-ce-price <CE name>:2119/jobmanager-lcgpbs-<queue>

Example: dgas-check-ce-price grid010.ct.infn.it:2119/jobmanager-lcgpbs-short

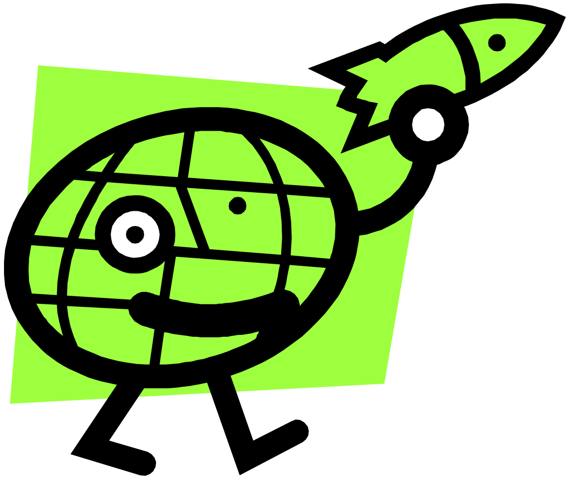
```
Price Authority queried at: Thu Oct 20 18:43:39 CEST 2005
Computing Element: grid010.ct.infn.it:2119/jobmanager-lcgpbs-short
Price (credits for 100 CPU secs):  170
```



Exercise: Monitoring

Grid Monitoring

- Identify the gilda VO in GridIce
- Find the graph describing the jobs submitted during this tutorial
- Reminder: GridIce can be found here: http://gridice2.cnaf.infn.it:50080/gridice/geo/geo.php



Exercise:
The Genius Portal

- Go to https://glite-tutor.ct.infn.it
- Login (you might have to run the myproxy-init command on the command line before that)
- Submit a simple hello world job and look at the output

Questions…