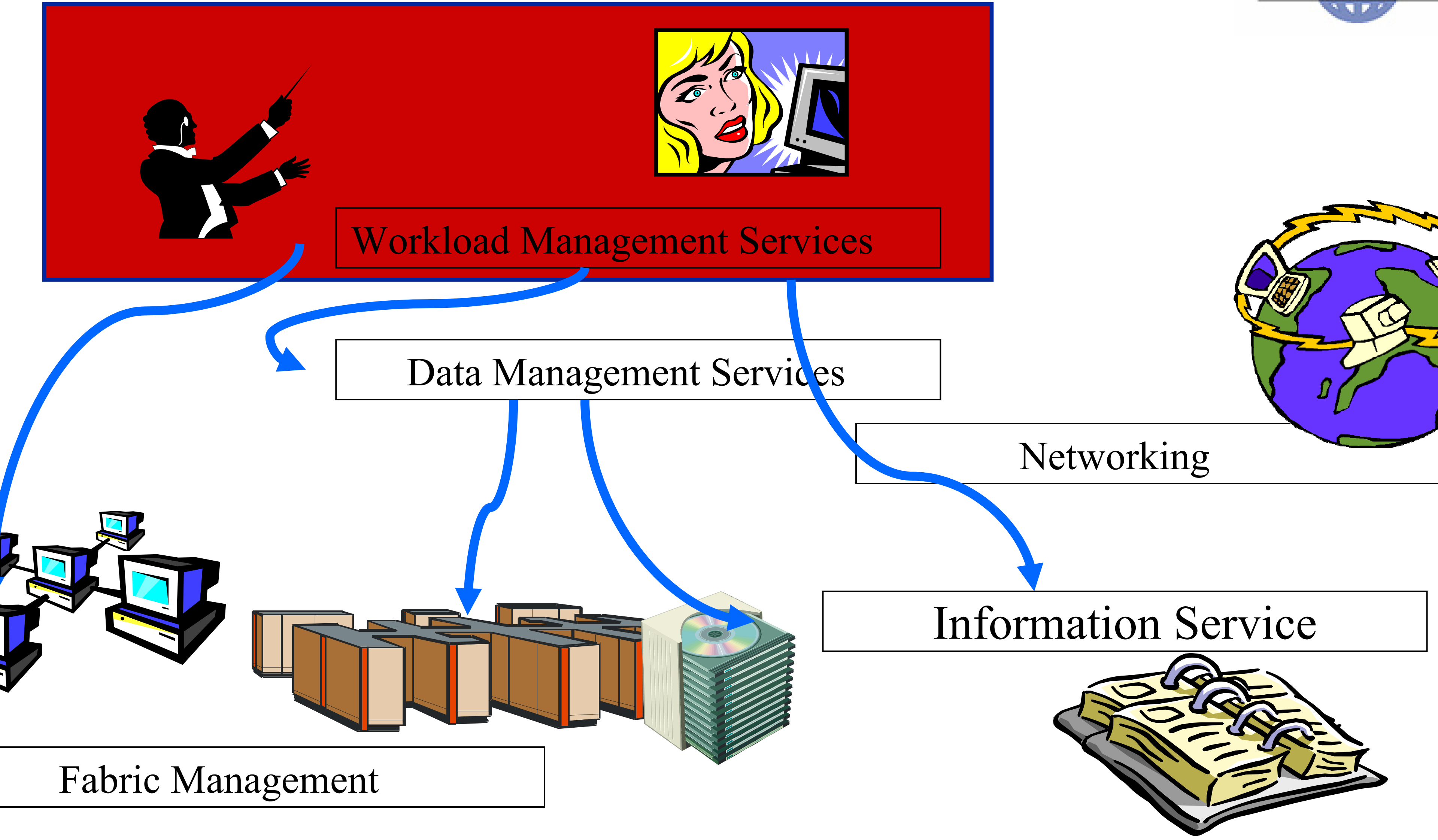
The EDG Workload Management System

EDG Tutorial Overview



Contents

- The EDG Workload Management System
- Job Preparation
  - Job Description Language
- Job submission and job status monitoring
- WMS Matchmaking
- Different job types
  - Normal jobs
  - Interactive jobs
  - Checkpointable jobs
  - Parallel jobs

The EDG WMS

- The user interacts with Grid via a **Workload Management System** (WMS)
- The Goal of WMS is the **distributed scheduling and resource management in a Grid environment**.
- What does it allow Grid users to do?
  - To submit their jobs
  - To execute them on the "best resources"
    - The WMS tries to optimize the usage of resources
  - To get information about their status
  - To retrieve their output

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>;
- So, the JDL allows definition of a set of attribute, the WMS takes into account when making its scheduling decision

Job Description Language (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
      - Have to be prefixed with "other."
    - *Data and Storage resources*
      - Input data to process, SE where to store output data, protocols spoken by application when accessing SEs

JDL: relevant attributes

- **JobType**
  - *Normal* (simple, sequential job), *Interactive*, *MPICH*, *Checkpointable*
  - Or combination of them
- **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 local disk needed by the job for running
  - The listed files will automatically staged to the remote resource
- **OutputSandbox** (optional)
  - List of files, generated by the job, which have to be retrieved

JDL: relevant attributes

- **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

- **InputData**
  - Refers to data used as input by the job: these data are published in the Replica Location Service (RLS) and stored in the SEs)
  - LFNs and/or GUIDs
- **DataAccessProtocol** (mandatory if InputData has been specified)
  - The protocol or the list of protocols which the application is able to speak with for accessing *InputData* on a given SE
- **OutputSE**
  - The Uniform Resource Identifier of the output SE
  - RB uses it to choose a CE that is compatible with the job and is close to SE

Example of JDL File

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

Job Submission

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

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

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

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

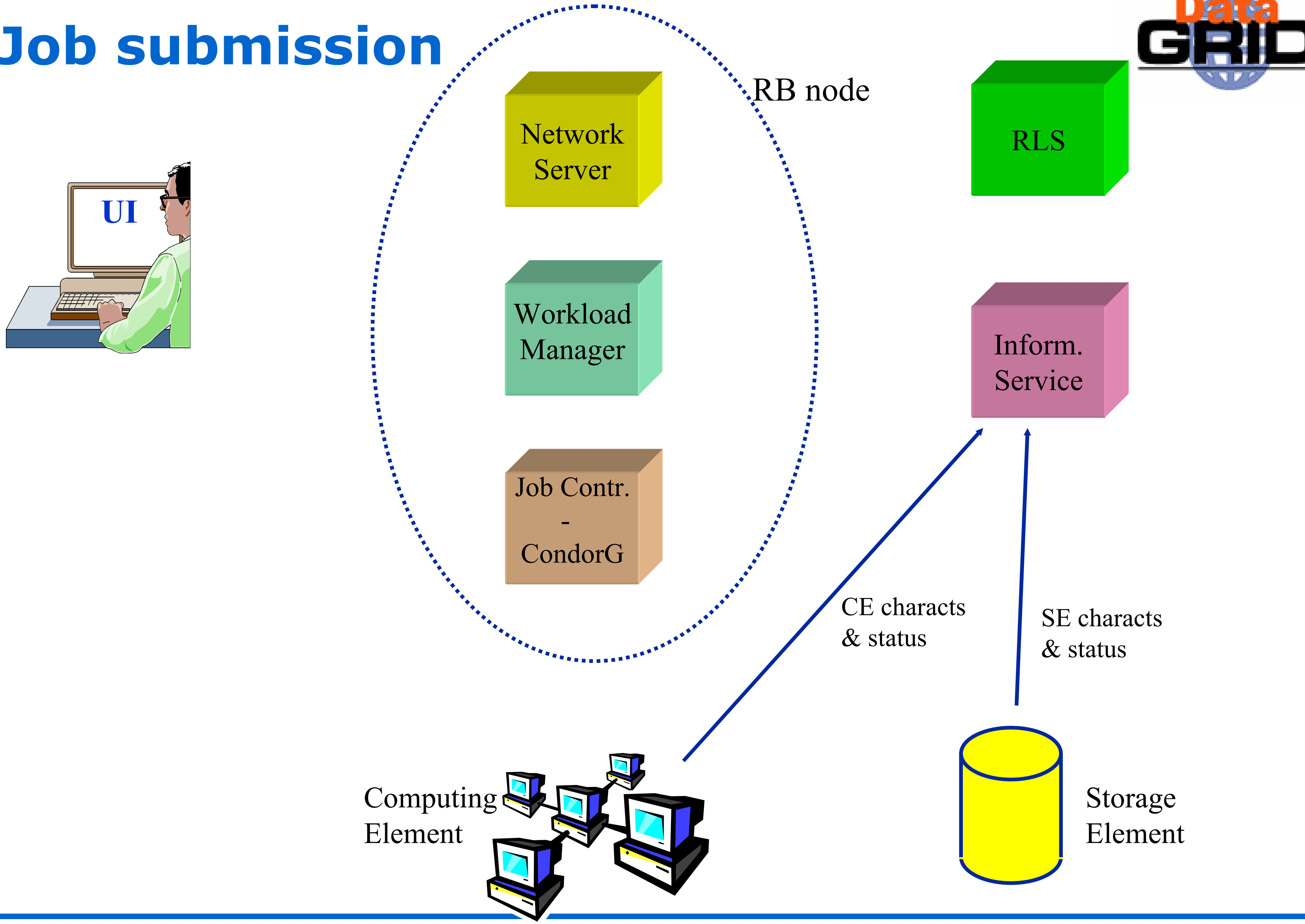
-o the generated edg_jobId is written in the <*output file*>

Useful for other commands, e.g.:

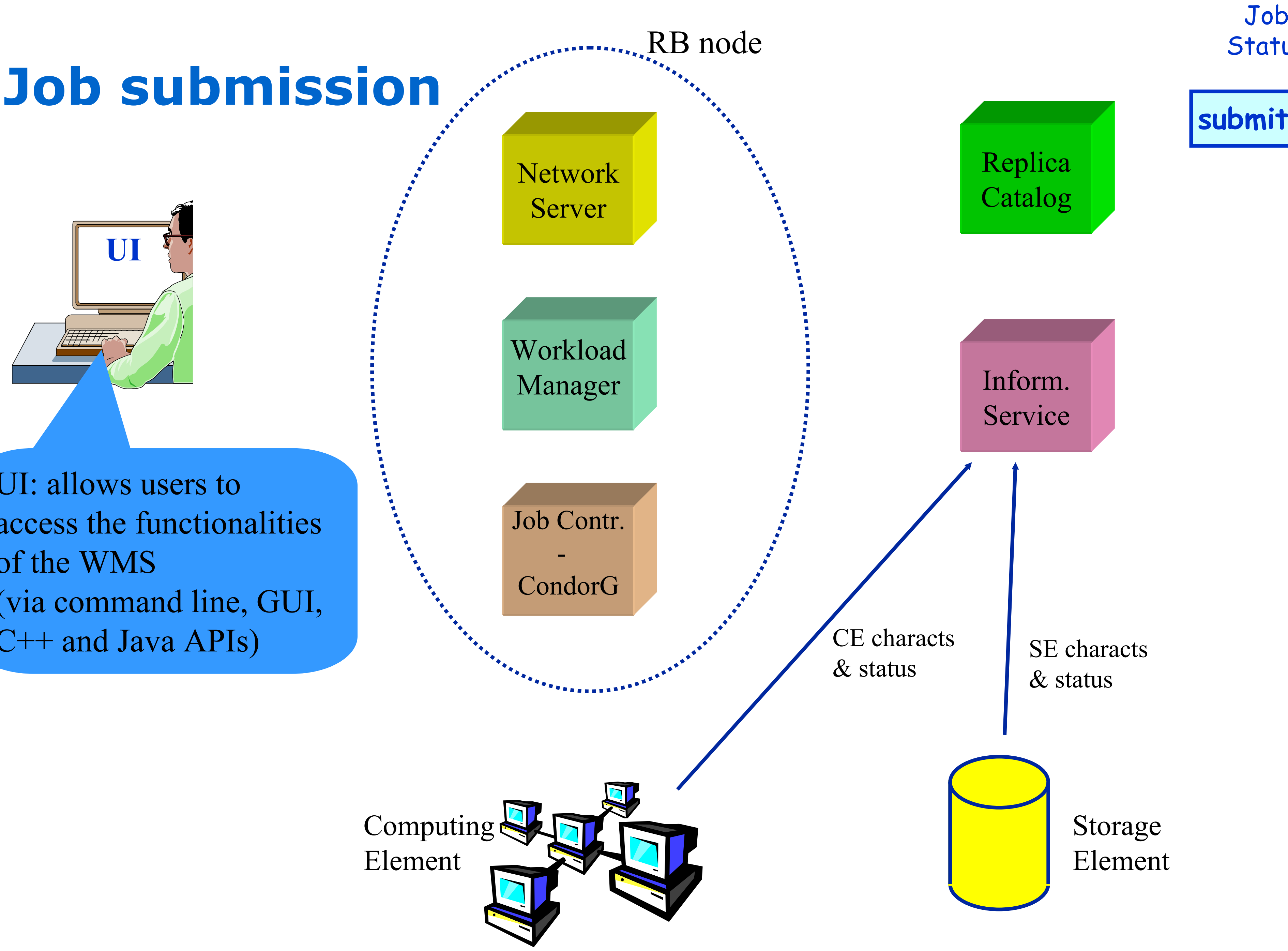
```
edg-job-status -i <input file> (or edg_jobId)
```

-i the status information about edg_jobId contained in the <*input file*> are displayed

Job submission



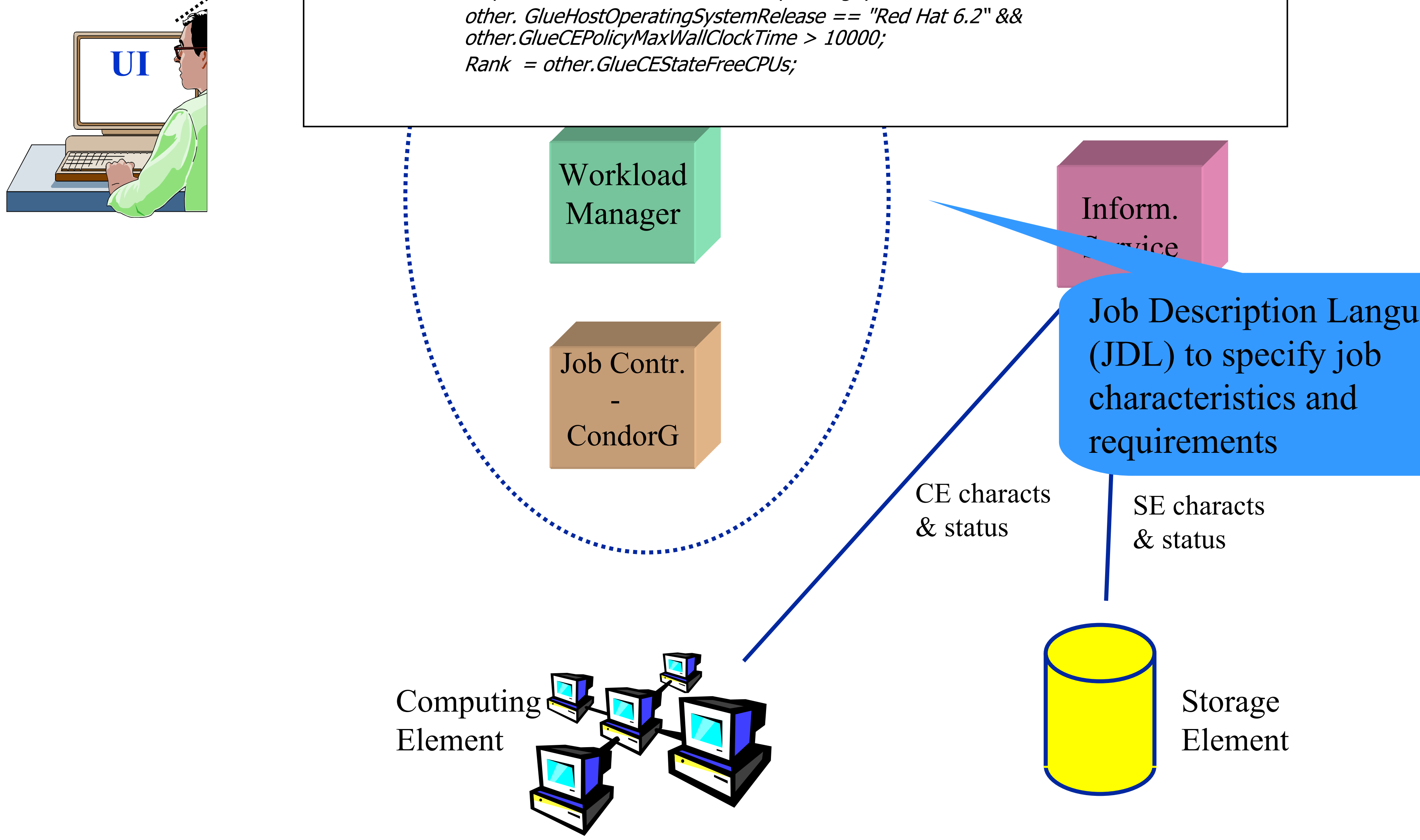
Job submission



UI: allows users to access the functionalities of the WMS (via command line, GUI, C++ and Java APIs)

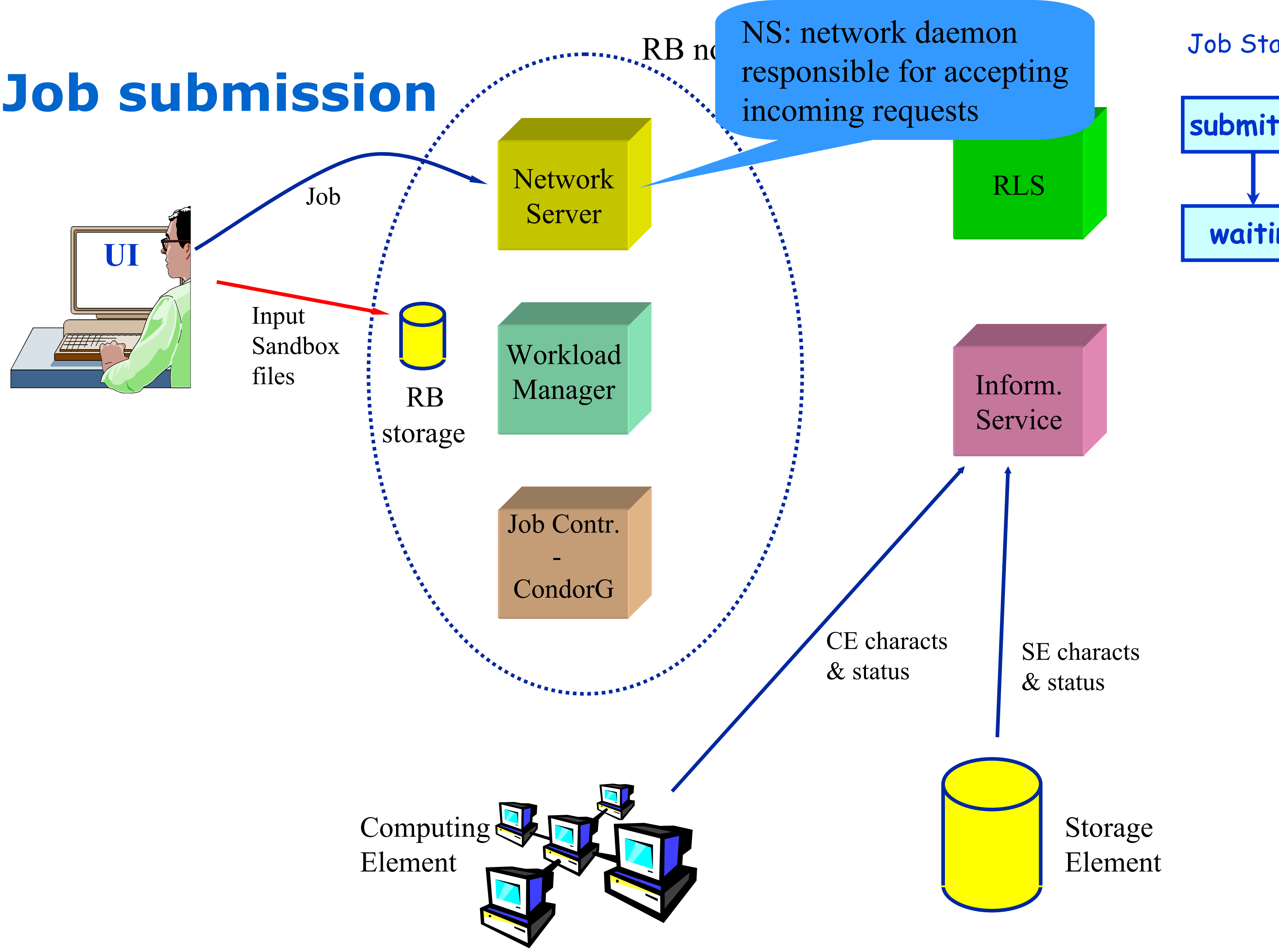
Job subm

```
edg-job-submit myjob.jdl
        Myjob.jdl
                JobType = "Normal";
                Executable     = "$(CMS)/exe/sum.exe";
                InputSandbox = {"/home/user/WP1testC","/home/file*", "/home/user/DATA/*"};
                OutputSandbox = {"sim.err", "test.out", "sim.log"};
                Requirements    = other. GlueHostOperatingSystemName == "linux" &&
                other. GlueHostOperatingSystemRelease == "Red Hat 6.2" &&
                other.GlueCEPolicyMaxWallClockTime > 10000;
                Rank  = other.GlueCEStateFreeCPUs;
```

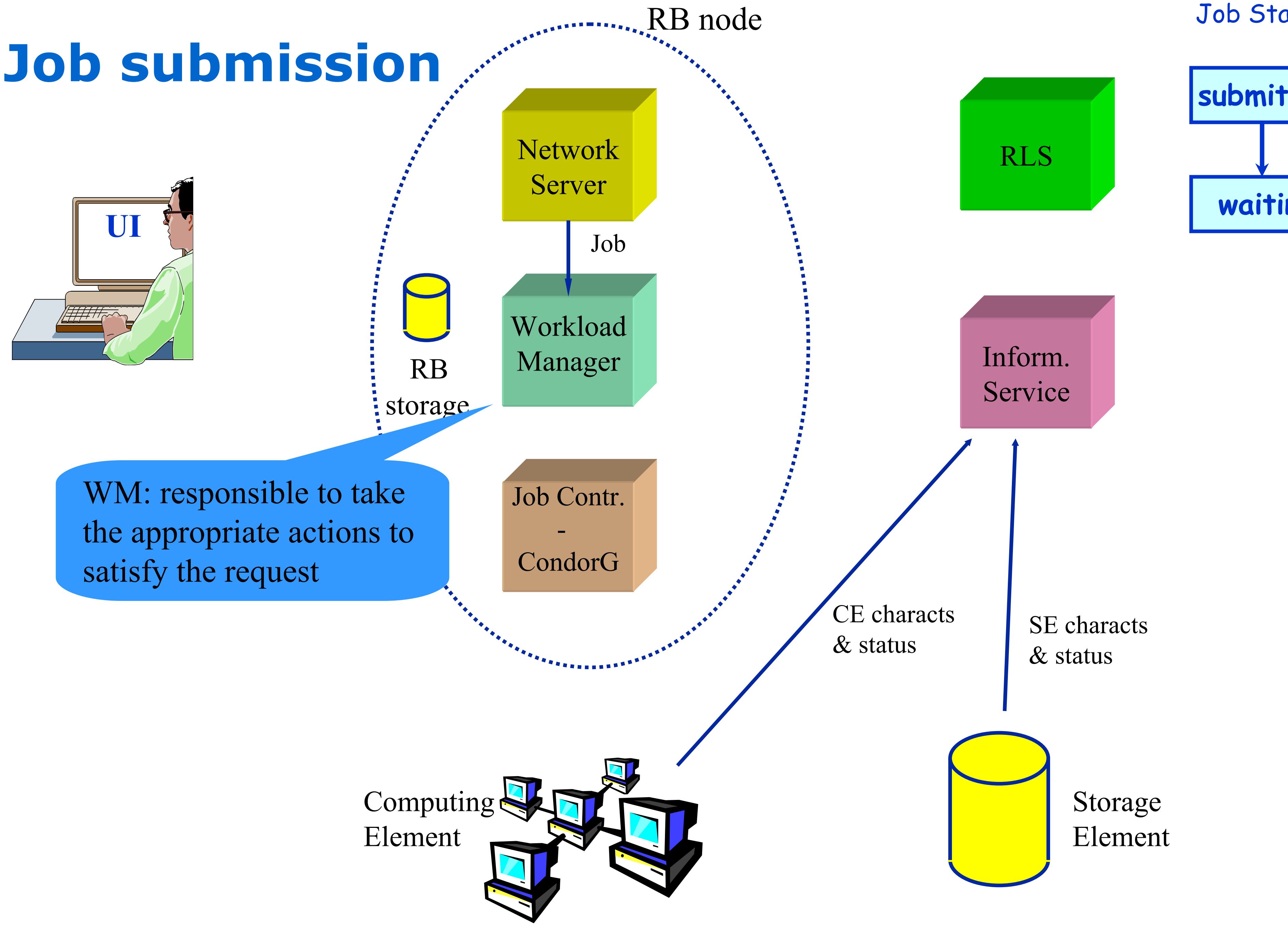


Job Description Langu (JDL) to specify job characteristics and requirements

Job submission

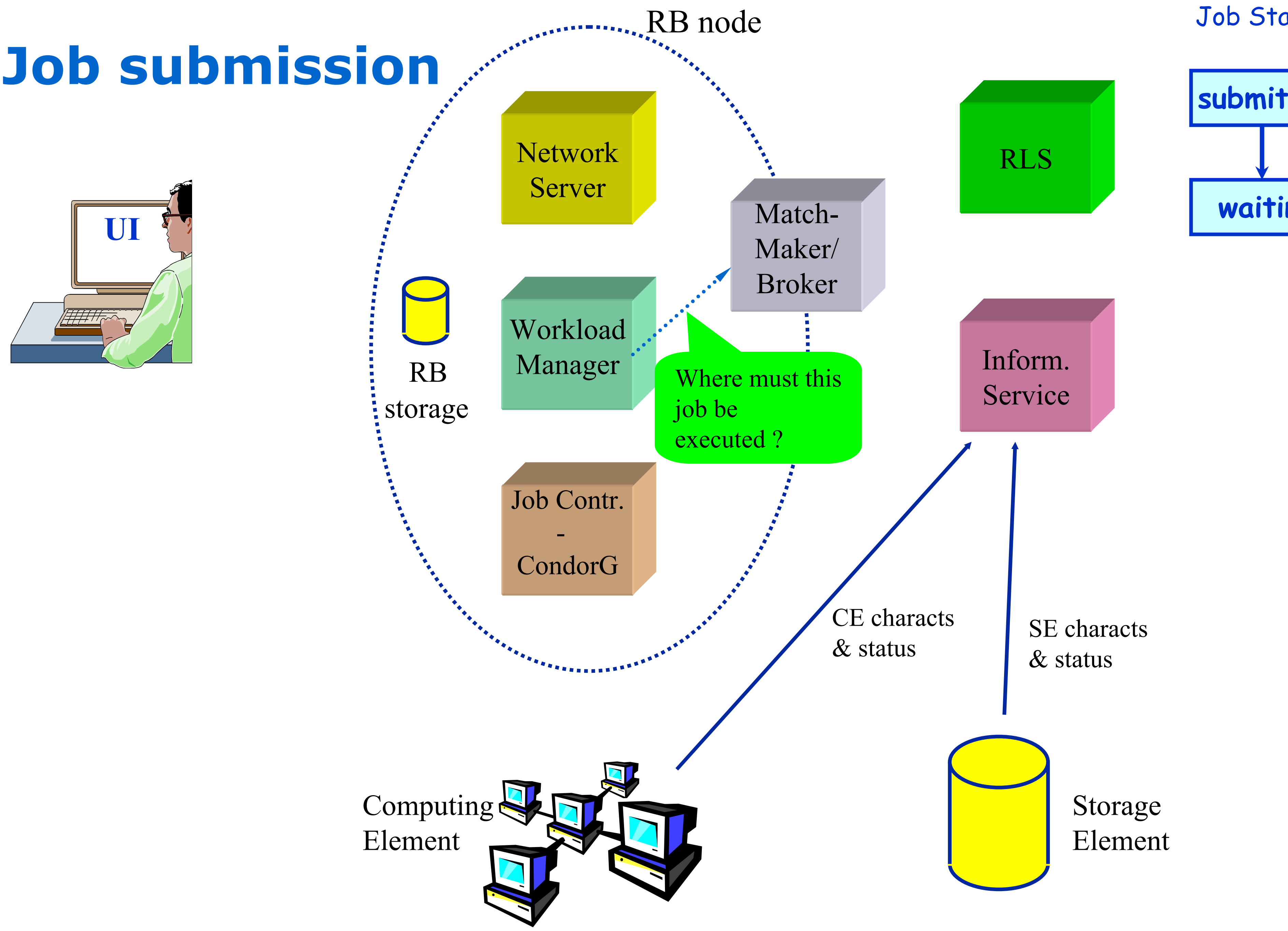


Job submission

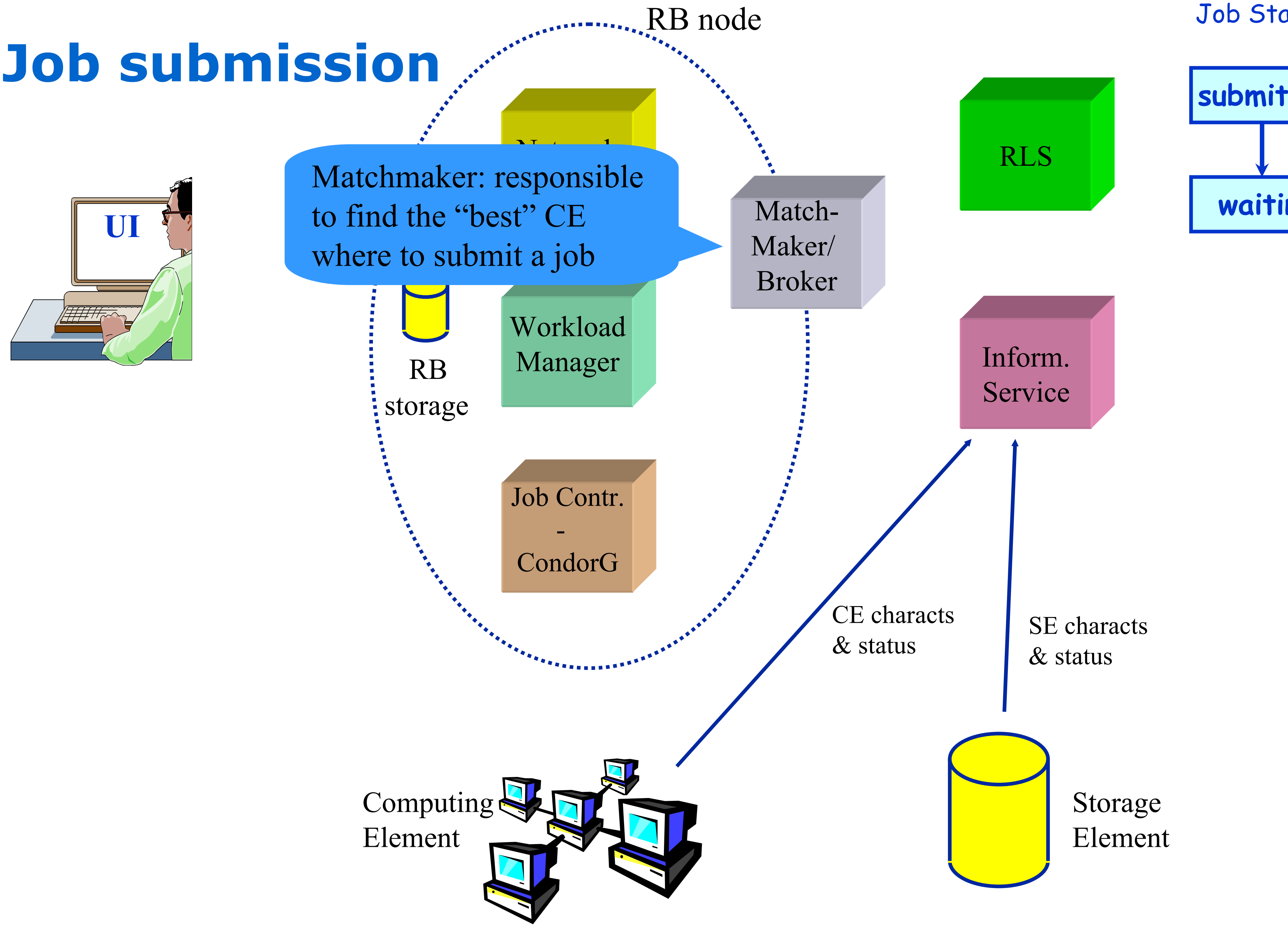


WM: responsible to take the appropriate actions to satisfy the request

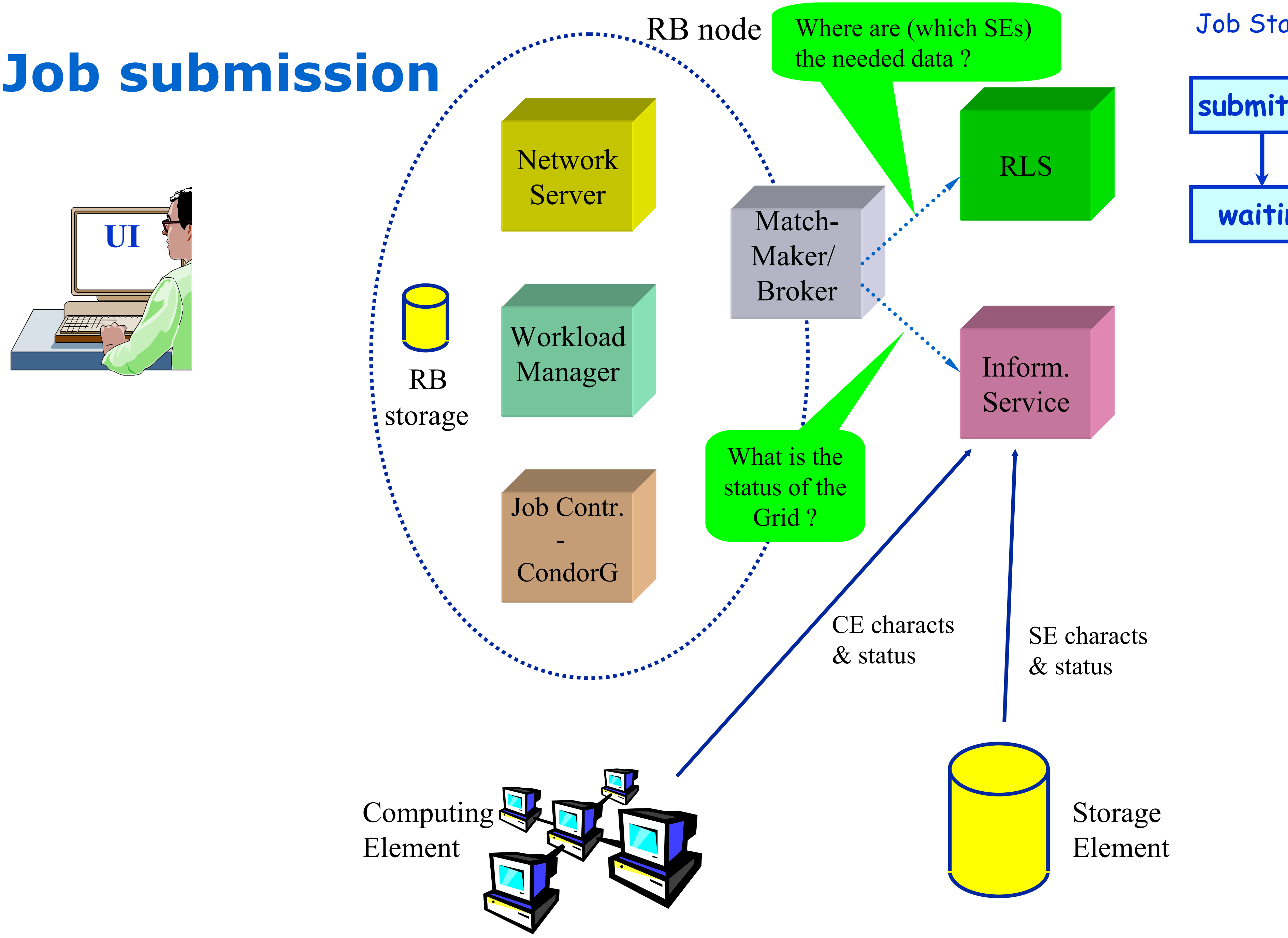
Job submission



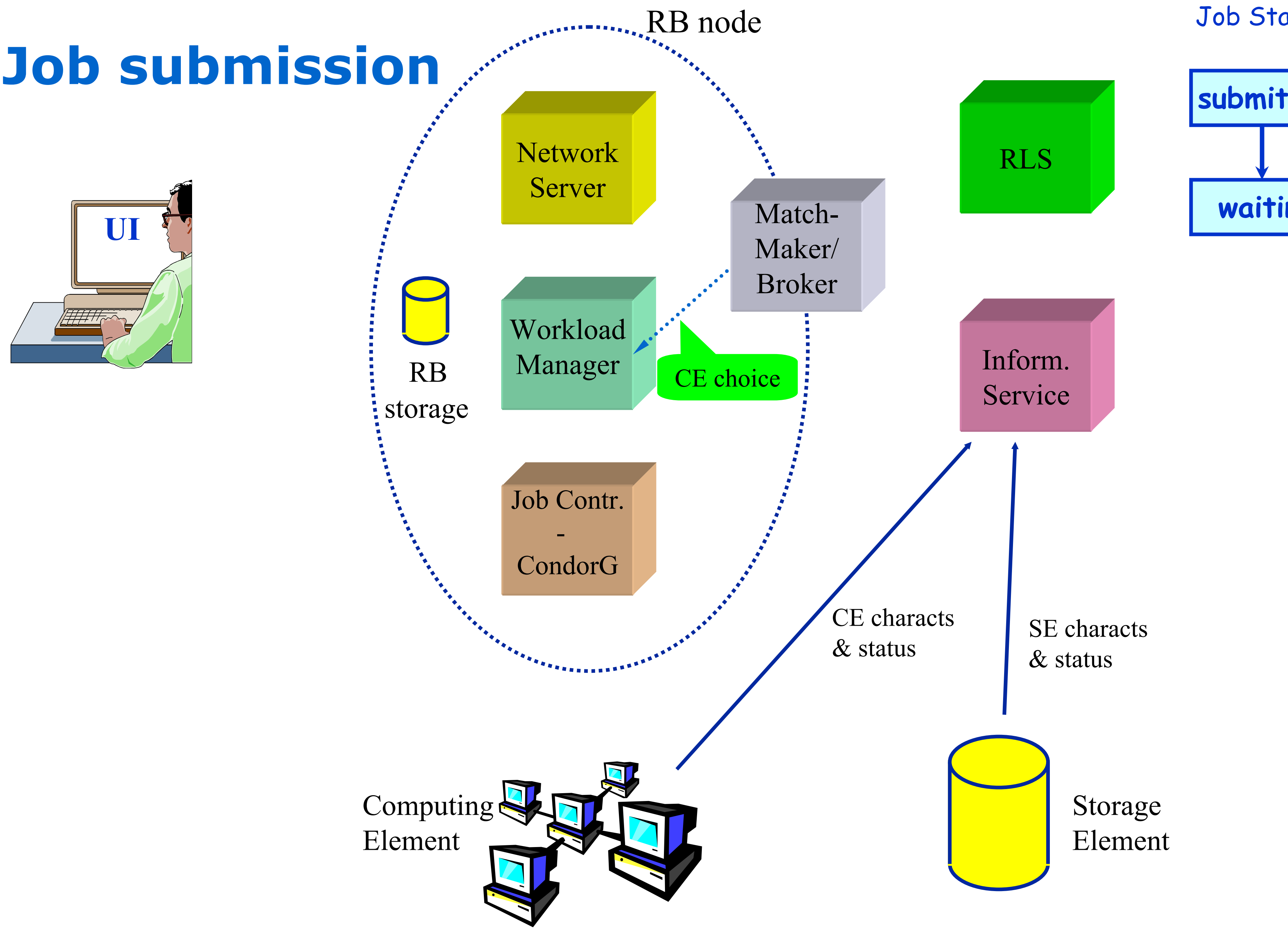
Job submission



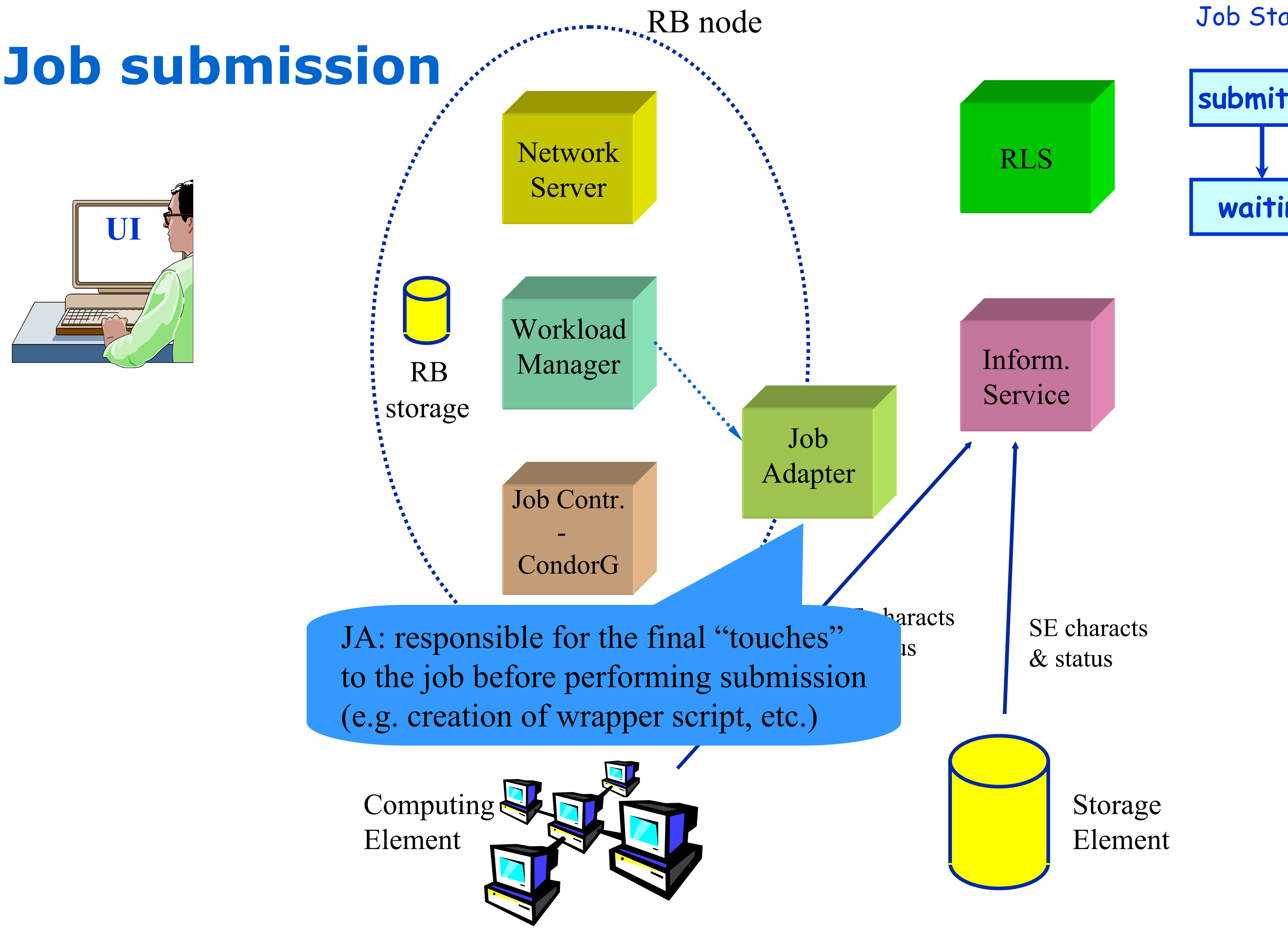
Job submission



Job submission

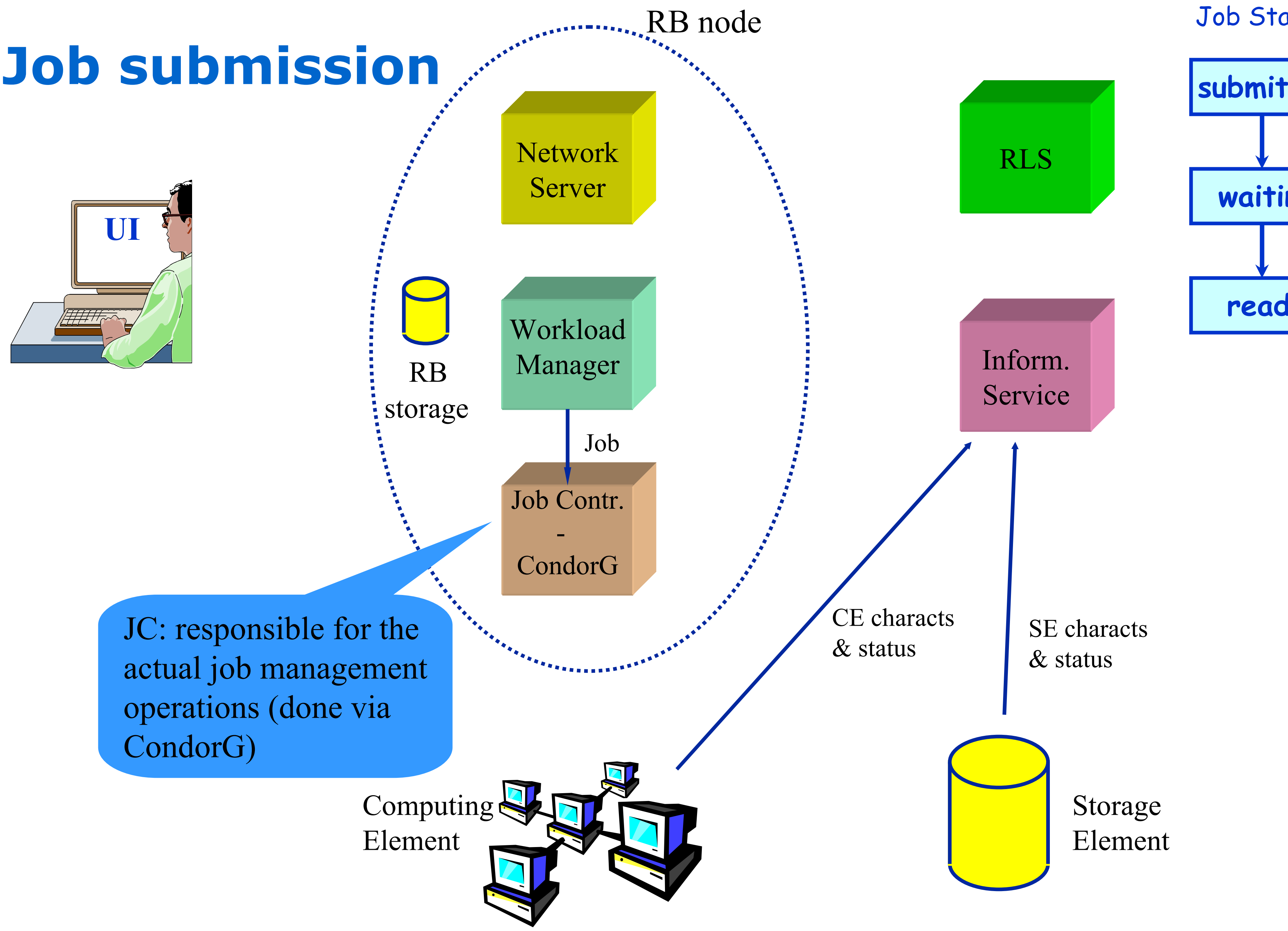


Job submission



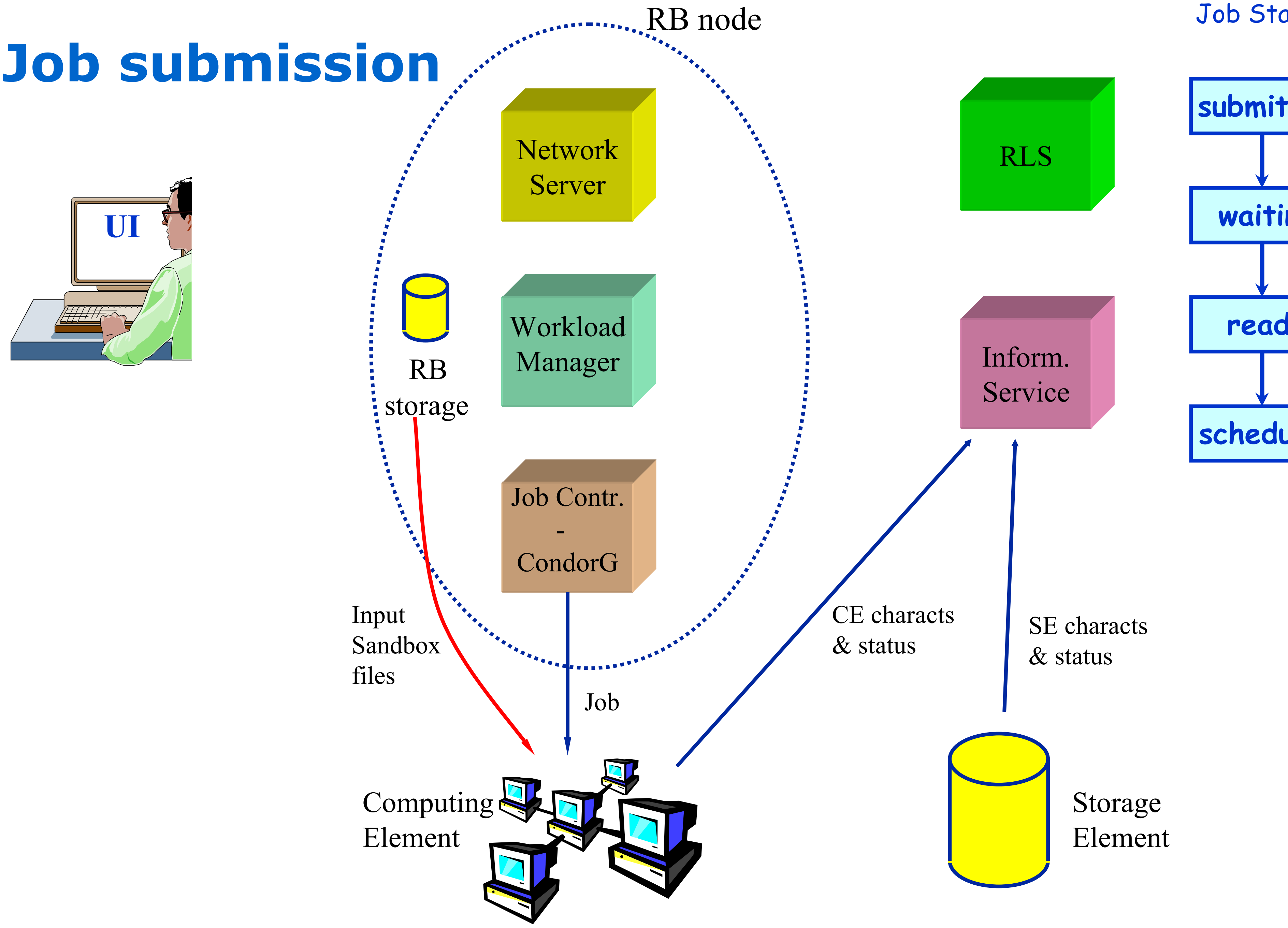
JA: responsible for the final "touches" to the job before performing submission (e.g. creation of wrapper script, etc.)

Job submission

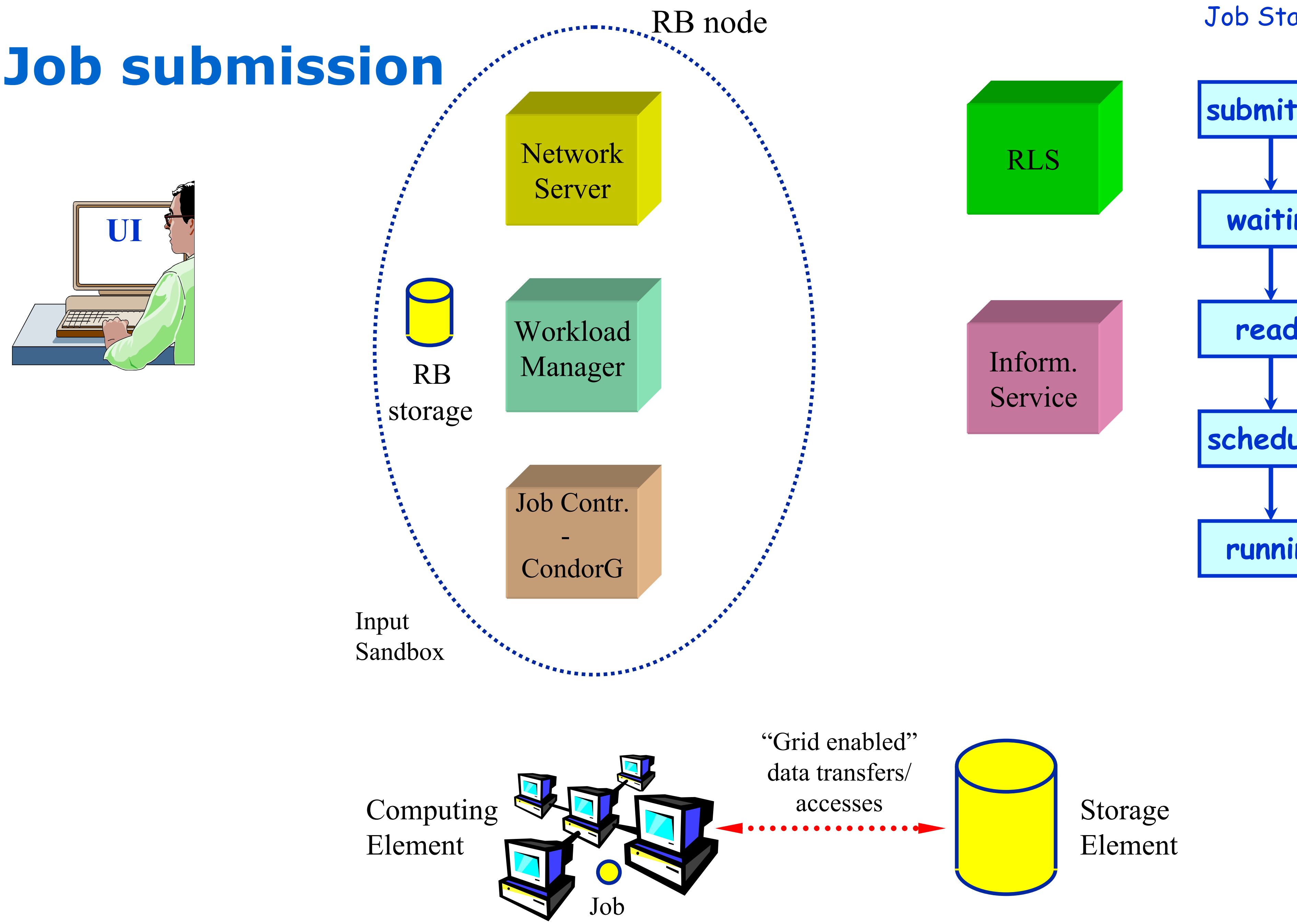


JC: responsible for the actual job management operations (done via CondorG)

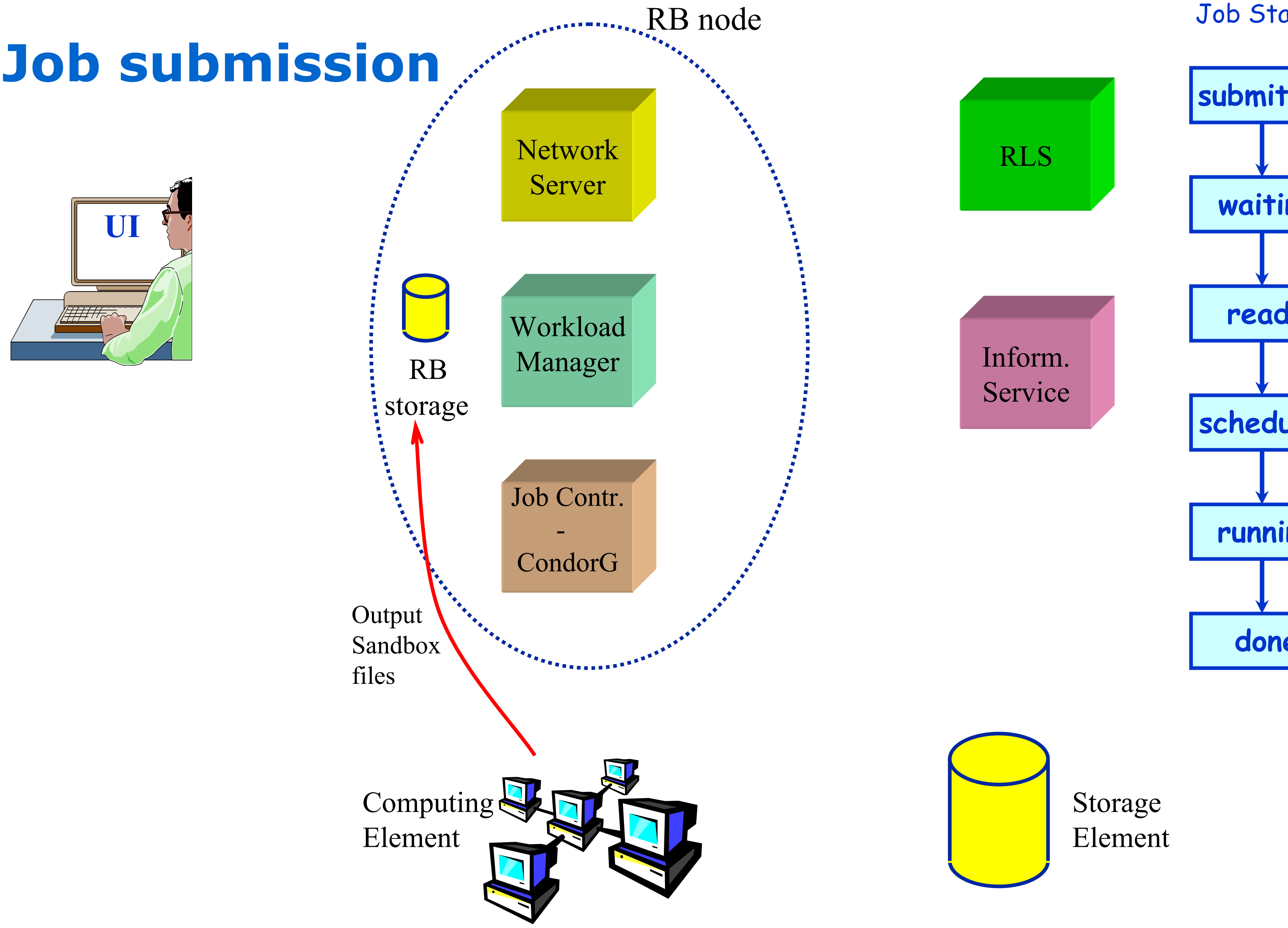
Job submission



Job submission

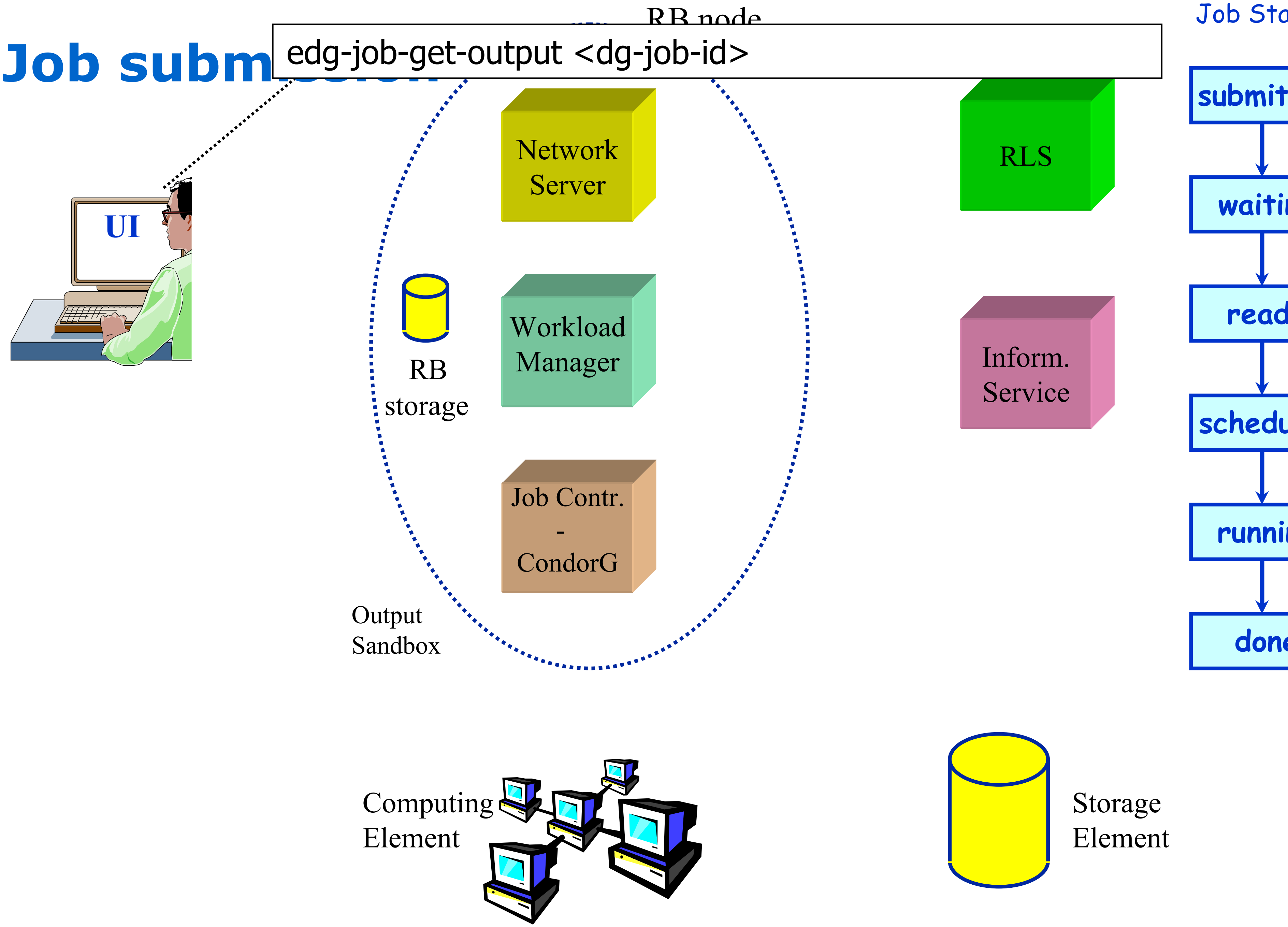


Job submission

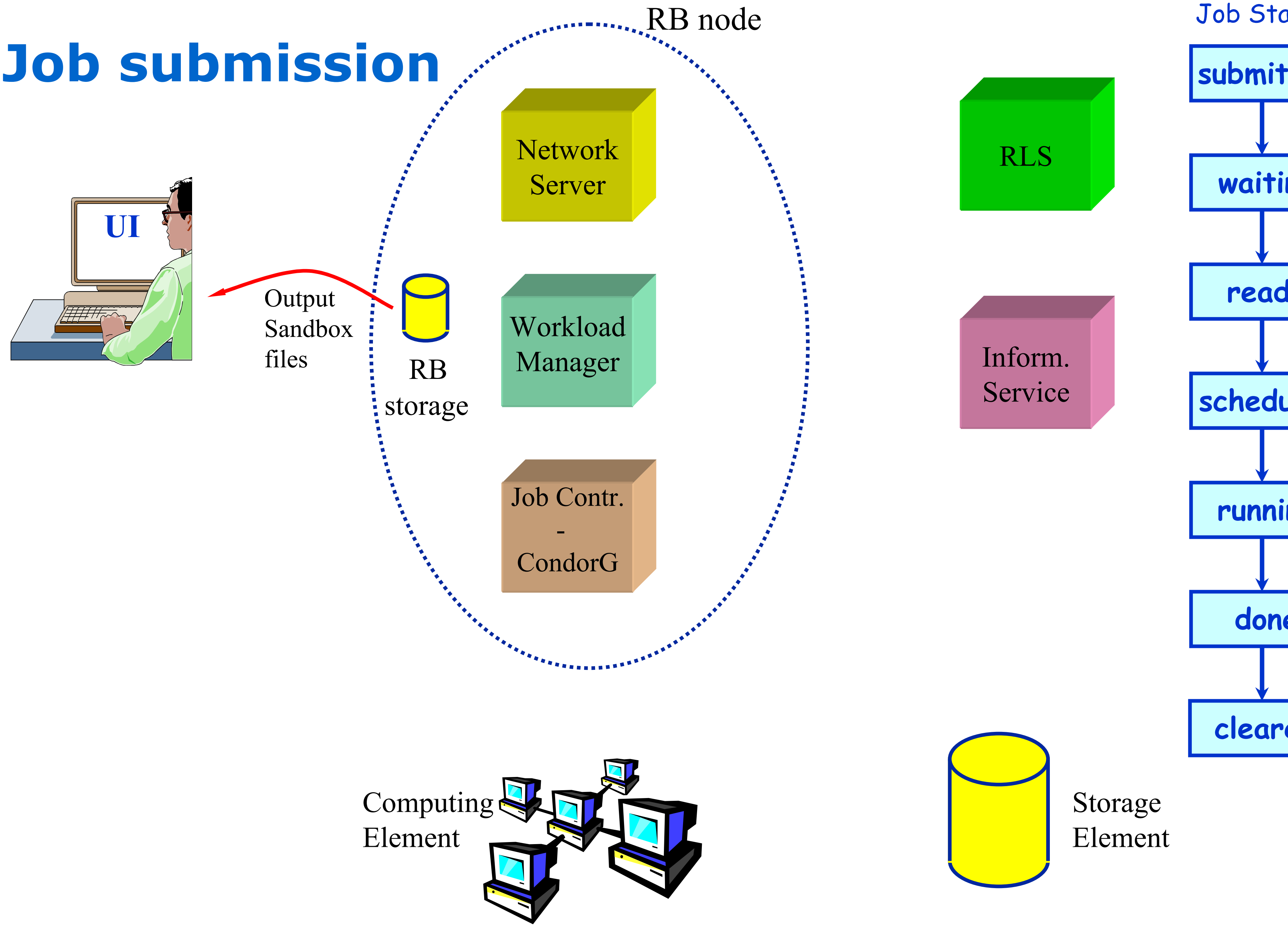


Job subm

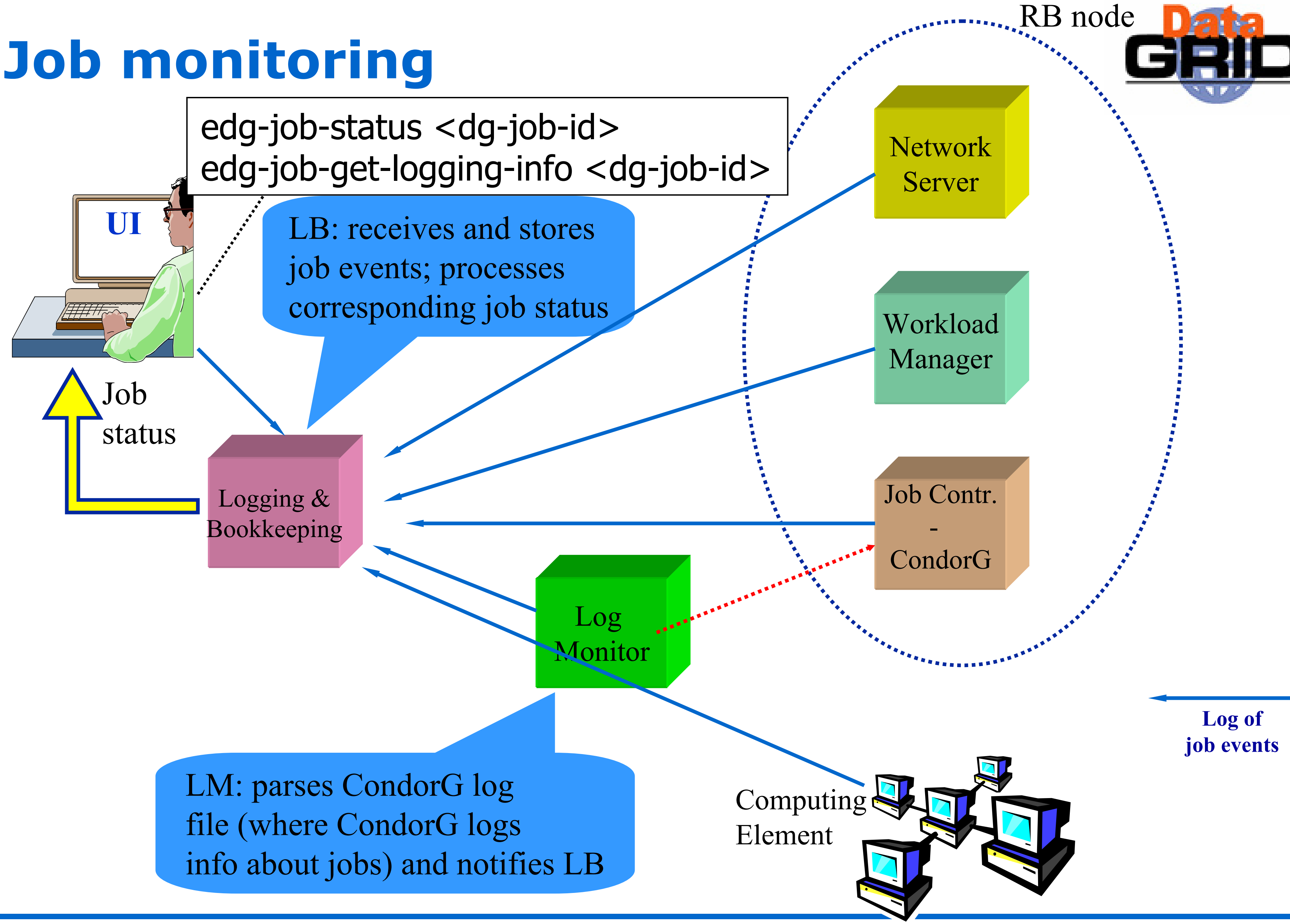
```
edg-job-get-output <dg-job-id>
```



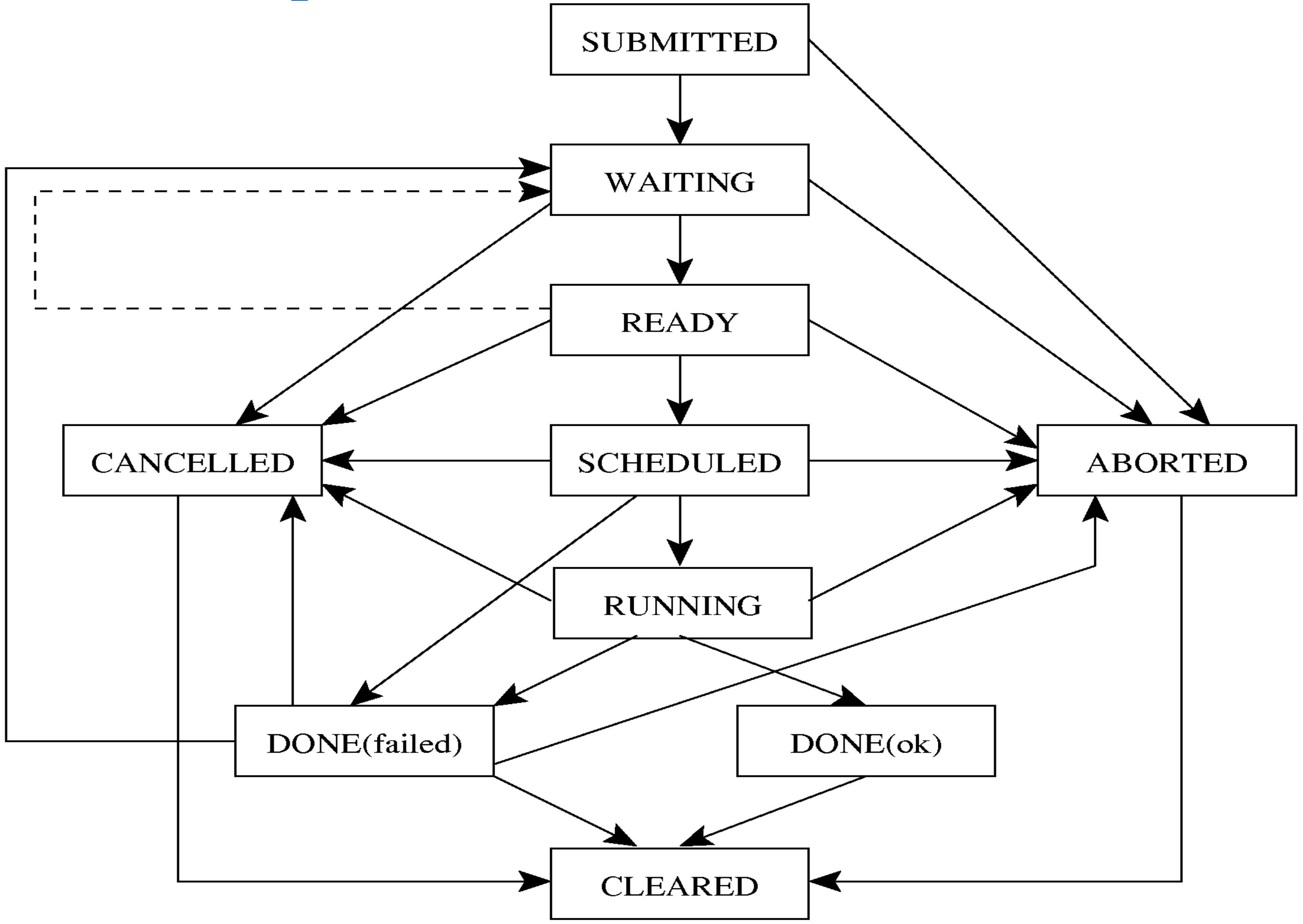
Job submission



Job monitoring


RB node
edg-job-status <dg-job-id>
edg-job-get-logging-info <dg-job-id>
UI
LB: receives and stores job events; processes corresponding job status
Network Server
Workload Manager
Job status
Logging & Bookkeeping
Job Contr. - CondorG
Log Monitor
Log of job events
LM: parses CondorG log file (where CondorG logs info about jobs) and notifies LB
Computing Element

Possible job states



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
- E.g., to disable job resubmission for a particular job: *RetryCount=0;* in the JDL file

Other (most relevant) UI commands

- **`edg-job-list-match`**
  - Lists resources matching a job description
  - Performs the matchmaking without submitting the job
- **`edg-job-cancel`**
  - Cancels a given job
- **`edg-job-status`**
  - Displays the status of the job
- **`edg-job-get-output`**
  - Returns the job-output (the OutputSandbox files) to the user
- **`edg-job-get-logging-info`**
  - Displays logging information about submitted jobs (all the events “pushed” by the various components of the WMS)
  - Very useful for debug purposes

UI configuration files

- Two UI configuration files
  - Common UI conf file
    - *$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf*
    - User can create his own conf file, and refers to it with option --config
  - VO UI conf file
    - *$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf*
    - User can create his own VO conf file, and refers to it with option --vo / --config-vo

Common UI configuration file

- Most relevant attributes
  - Default JDL Requirements
    - *other.GlueCEStateStatus == "Production"*
  - Default JDL Rank
    - *- other.GlueCEStateEstimatedresponseTime*
  - Default VO
  - Default verbosity level for *edg-job-status* and *edg-job-get-logging-info*
  - Default value for *RetryCount*

VO UI configuration file

- Most relevant attributes
  - NS(s)
    - When submitting a job, the first specified RB is tried, if the operation fails the second one is considered, etc.
  - LB server(s)
    - The LB server to be used for a given job to be submitted is chosen in a random way among the listed one
    - When a –all query (e.g. edg-job-status –all) is issued, all these LB servers are queried

WMS Matchmaking

- The RB (Matchmaker) has to find the best suitable computing resource (CE) where the job will be executed
- It interacts with Data Management Service and Information Services
  - They supply RB with all the information required for the resolution of the matches
- The CE chosen by RB has to match the job requirements (e.g. runtime environment, data access requirements, and so on)
- If *FuzzyRank=False* (default):
  - If 2 or more CEs satisfy all the requirements, the one with the best Rank is chosen
  - If there are two or more CEs with the same best rank, the choice is done in a random way among them
- If *FuzzyRank=True* in the JDL:
  - Fuzziness in CE choice: the CE with highest rank has the highest probability to be chosen

WMS matchmaking scenarios

- Possible scenarios for matchmaking:
  1. Direct job submission
     - *edg-job-submit –r <CEId>*
  2. Job submission with only computational requirements
     - Nor InputData nor OutputSE specified in the JDL
  3. Job submission with also data access requirements
     - InputData and/or OutputSE specified in the JDL
  4. Matchmaking with getAccessCost

Direct job submission

edg-job-submit –r CEId

- Job is simply submitted on the given CE
- RB doesn’t perform any matchmaking algorithm
- Information services not queried at all

Job submission with only comput. reqs

- Nor InputData nor OutputSE specified in the JDL

- Matchmaking algorithm:
  - Requirements check
    - RB contacts the IS to check which CEs satisfy all the requirements
    - This includes also authorization check (where is the user allowed to submit jobs ?)
  - Suitable resources directly queried (GRISes queried) to evaluate Rank expression (which usually refers to dynamic values)
  - If more than one CE satisfies the job requirements, the CE with the best rank is chosen by the RB (or has the highest probability to be chosen, if Fuzzyrank enabled)

Job submission with data access reqs

- InputData and/or OutputSE specified in the JDL

- RB strategy: submit jobs close to data
- Matchmaking algorithm:
  - Requirements check as in the previous case
  - CE chosen among the suitable ones (the CEs which passed the requirements check) and where most of the needed files are "close" to it (where most of the needed files are stored on SEs close to the considered CE)

Matchmaking with GetAccessCost

- Can be used when InputData has been specified in the JDL
- Used when Rank = other.DataAccessCost has been specified in the JDL
- Matchmaking algorithm:
  - Requirements check as in the previous case
  - The CE is chosen by the ‘getAccessCost’ method provided by data Management Services among the suitable CEs (the CEs which passed the requirements check), taking into account data location and network information

Example of job submission

- User logs in on the UI
- User issues a *grid-proxy-init* and enters his certificate's password, getting a valid Globus proxy
- User sets up his or her JDL file
- Example of Hello World JDL file :

```
[
 Executable = "/bin/echo";
 Arguments = "Hello World";
 StdOutput = "Messagge.txt";
 StdError = "stderr.log";
 OutputSandbox = {"Message.txt","stderr.log"};
]
```

Example of job submission

- User issues a: *edg-job-submit HelloWorld.jdl*

  and gets back from the system a unique Job Identifier (JobId)

- User issues a: ed*g-job-status JobId*

  to get logging information about the current status of his Job

- When the "Output" status is reached, the user can issue a
  edg-job-get-output JobId

  and the system returns the name of the temporary directory where the job output can be found on the UI machine.

Example of job submission

```
$ edg-job-submit HelloWorld.jdl


*****************************************************************************************

                         JOB SUBMIT OUTCOME

 The job has been successfully submitted to the Network Server.

 Use edg-job-status command to check job current status. Your job identifier
   (edg_jobId) is:


 - https://lxshare0403.cern.ch:9000/KoBA-IgxZyVpLKhANfrhHw

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

JobId

Example of job submission

```
$ edg-job-status https://lxshare0403.cern.ch:9000/KoBA-IgxZyVpLKhANfrhHw


*************************************************************

BOOKKEEPING INFORMATION:



Printing status info for the Job : https://lxshare0403.cern.ch:9000/KoBA-
  IgxZyVpLKhANfrhHw

Current Status:    Done (Success)

Exit code:         0

Status Reason:     Job terminated successfully

Destination:       lxshare0405.cern.ch:2119/jobmanager-pbs-infinite

reached on:        Wed Jun 18 12:06:10 2003

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

Example of job submission

```
$ edg-job-get-output --dir Results https://lxshare0403.cern.ch:9000/KoBA-
  IgxZyVpLKhANfrhHw


************************************************************

                    JOB GET OUTPUT OUTCOME

 Output sandbox files for the job:

 - https://lxshare0403.cern.ch:9000/KoBA-IgxZyVpLKhANfrhHw

 have been successfully retrieved and stored in the directory:

 /shift/lxshare072d/data01/UIhome/sgaravat/Results/KoBA-IgxZyVpLKhANfrhHw

************************************************************


$ more Results/KoBA-IgxZyVpLKhANfrhHw/Message.txt

Hello World

$ more Results/KoBA-IgxZyVpLKhANfrhHw/stderr.log

$
```

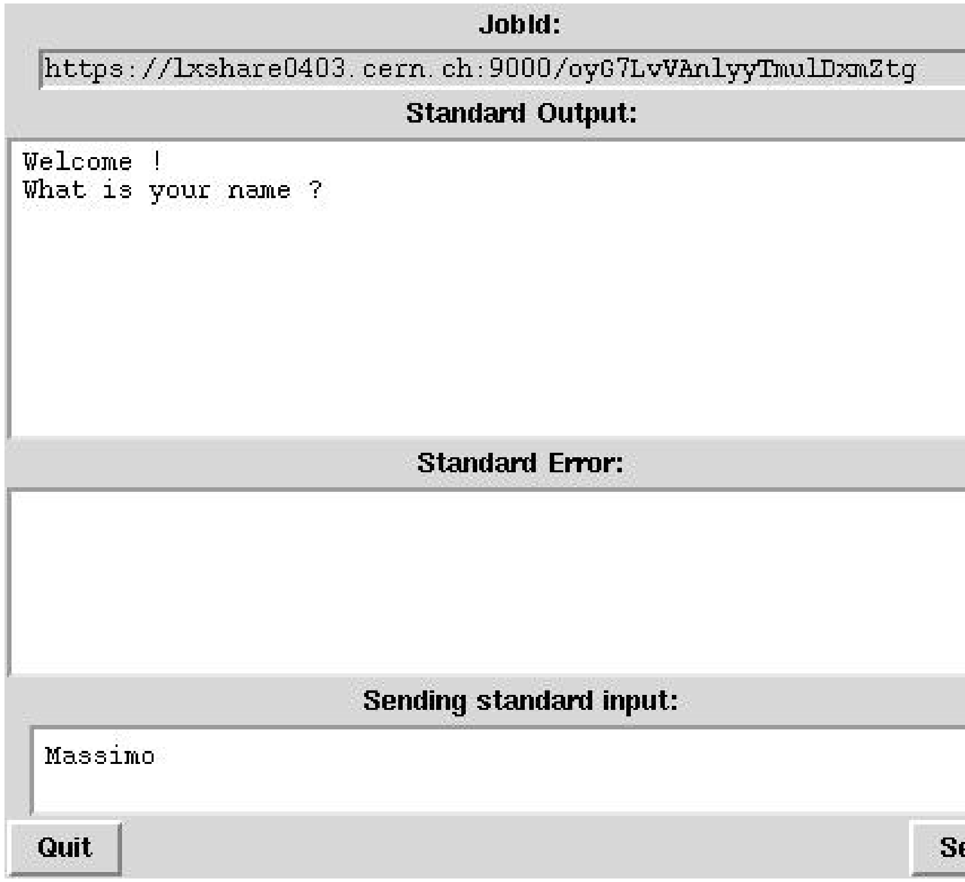
Proxy renewal

- Why?
  - To avoid job failure because it outlived the validity of the initial proxy, avoiding considering long term user proxies
- Solution
  - Short term proxies created as usual in the UI machine
    - *grid-proxy-init –hours <hours>*
  - User registers proxy into a MyProxy server:
    - *myproxy-init –s <server> [-t <cred> -c <proxy>]*
      - *server* is the server address (e.g. lxshare0375.cern.ch)
      - *cred* is the number of hours the proxy should be valid on the server
      - *proxy* is the number of hours renewed proxies should be valid
  - User specifies the MyProxy server in the JDL to enable proxy renewal:
    - MyProxyServer=myproxy.host.name;

  - The Proxy is automatically renewed by WMS without user intervention for all the job life

Interactive jobs

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


JobId:

```
https://lxshare0403.cern.ch:9000/oyG7LvVAnlyyTmulDxmZtg
```

Standard Output:

```
Welcome !
What is your name ?
```

Standard Error:

Sending standard input:

```
Massimo
```

Job checkpointing

- Checkpointing: saving from time to time job state
  - Useful to prevent data loss, due to unexpected failures
  - Approach: provide users with a “trivial” logical job checkpointing service
  - User can save from time to time the state of the job (defined by the application)
  - A job can be restarted from an intermediate (i.e. “previously” saved) job state
- Different than “classical checkpointing (i.e. saving all the information related to a process: process’s data and stack segments, open files, etc.)
  - Very difficult to apply (e.g. problems to save the state of open network connections)
  - Not necessary for many applications
- To submit a checkpointable job
  - Code must be instrumented (see next slides)
  - JobType=Checkpointable to be specified in JDL

Job checkpointing example

```
nt main ()

...
for (int i=event; i < EVMAX; i++)
      {  < process event i>;}
              ...
xit(0); }
```

Example of
Application
(e.g. HEP MonteCarlo
simulation)

Job checkpointing example

```
include "checkpointing.h"

nt main ()
 JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ….
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
…
for (int i=event; i < EVMAX; i++)
      {  < process event i>;
           ...
           state.saveValue("first_event", i+1);
          < save intermediate file on a SE>;
           state.saveValue("filename", PFN of file_on_SE);
            ...
            state.saveValue("var_n", value_n);
           state.saveState(); }
…
exit(0); }
```

User code must be easily instrumented in order to exploit the checkpointing framework …

Job checkpointing example

```
include "checkpointing.h"

nt main ()
 JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ….
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
…
for (int i=event; i < EVMAX; i++)
      {  < process event i>;
            ...
            state.saveValue("first_event", i+1);
           < save intermediate file on a SE>;
            state.saveValue("filename", PFN of file_on_SE);
             ...
             state.saveValue("var_n", value_n);
            state.saveState(); }
…
exit(0); }
```

- User defines what is a state
- Defined as <var, value> pairs
- Must be “enough” to restart a computation from a previously saved state

Job checkpointing example

```
include "checkpointing.h"

nt main ()
 JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ….
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
…
for (int i=event; i < EVMAX; i++)
     {  < process event i>;
          ...
          state.saveValue("first_event", i+1);
         < save intermediate file on a SE>;
          state.saveValue("filename", PFN of file_on_SE);
           ...
           state.saveValue("var_n", value_n);
          state.saveState(); }
…
exit(0); }
```

User can save from time to time the state of the job

Job checkpointing example

```
include "checkpointing.h"

nt main ()
 JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ….
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
…
for (int i=event; i < EVMAX; i++)
      {  < process event i>;
           ...
           state.saveValue("first_event", i+1);
          < save intermediate file on a SE>;
           state.saveValue("filename", PFN of file_on_SE);
            ...
            state.saveValue("var_n", value_n);
           state.saveState(); }
…
exit(0); }
```

Retrieval of the last saved state
The job can restart from that point

Job checkpointing scenarios

- Scenario 1
  - Job submitted to a CE
  - When job runs it saves from time to time its state
  - Job failure, due to a Grid problems (e.g. CE problem)
  - Job resubmitted by the WMS possibly to a different CE
  - Job restarts its computation from the last saved state
    - → No need to restart from the beginning
    - → The computation done till that moment is not lost
- Scenario 2
  - Job failure, but not detected by the Grid middleware
  - User can retrieved a saved state for the job (typically the last one)
    - *edg-job-get-chkpt –o <state><edg-jobid>*
  - User resubmits the job, specifying that the job must start from a specific (the retrieved one) initial state
    - *edg-job-submit –chkpt <state> <JDL file>*

Submission of parallel jobs

- Possibility to submit MPI jobs
- MPICH implementation supported
- Only parallel jobs inside a single CE can be submitted
- Submission of parallel jobs very similar to normal jobs
  - Just needed to specify in the JDL:
    - JobType= "MPICH"
    - NodeNumber = n;
      - The number (*n*) of requested CPUs
- Matchmaking
  - CE chosen by RB has to have MPICH sw installed, and at least *n* total CPUs
  - If there are two or more CEs satisfying all the requirements, the one with the highest number of free CPUs is chosen

Further information

- The EDG User's Guide

  http://marianne.in2p3.fr

- EDG WP1 Web site

  http://www.infn.it/workload-grid

  In particular WMS User & Admin Guide and JDL docs

- ClassAd

  https://www.cs.wisc.edu/condor/classad