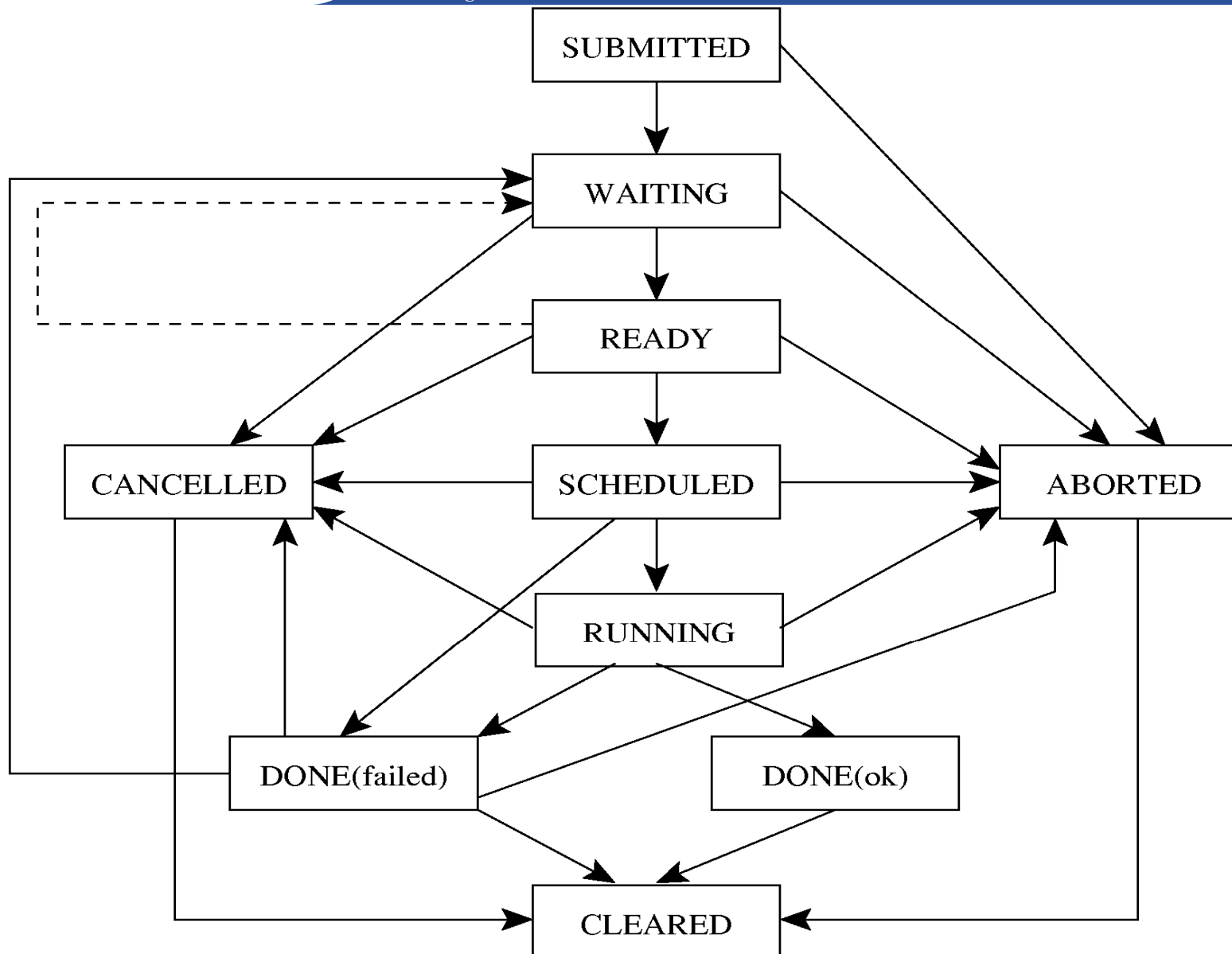


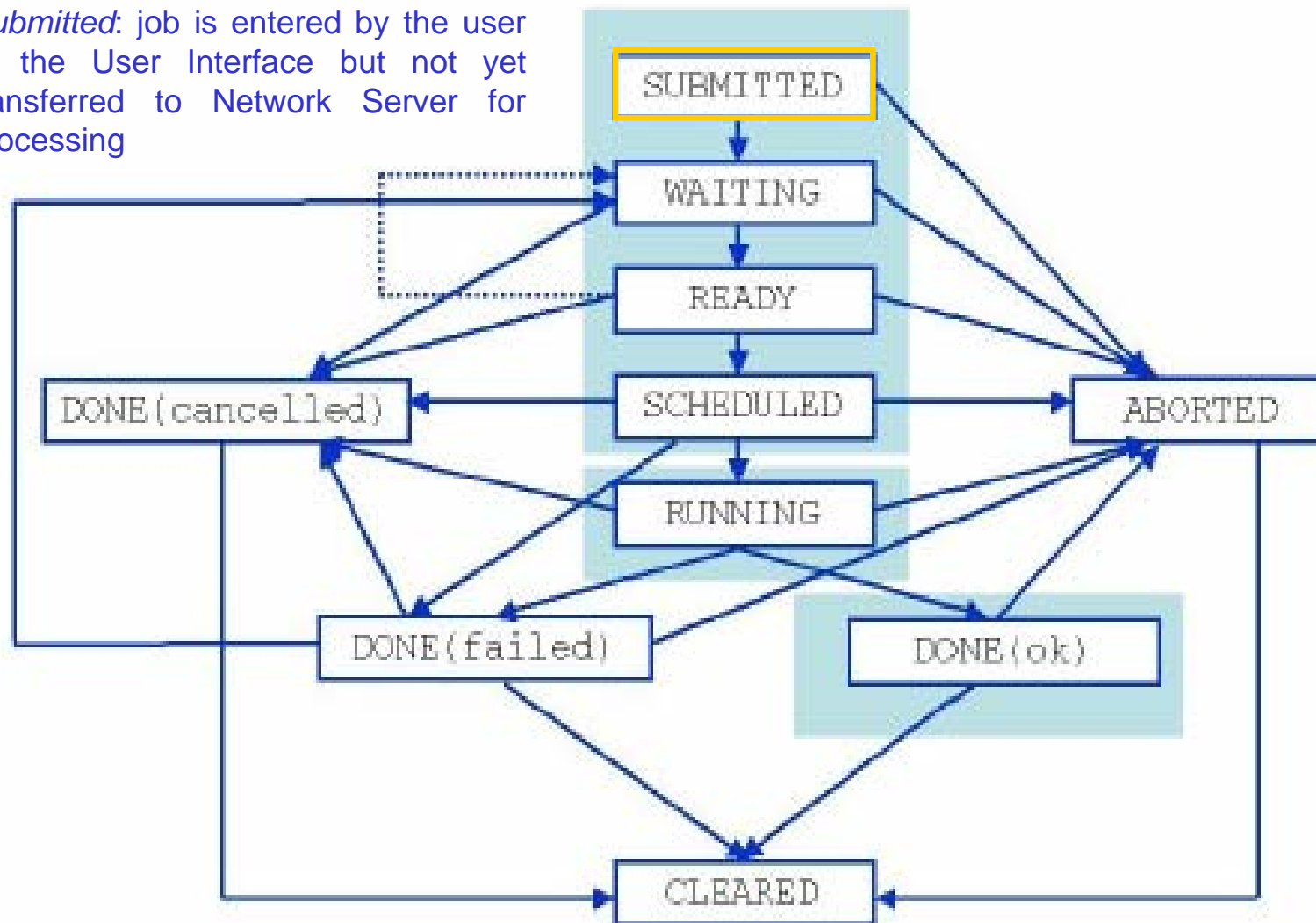
# JDL

[www.eu-egee.org](http://www.eu-egee.org)

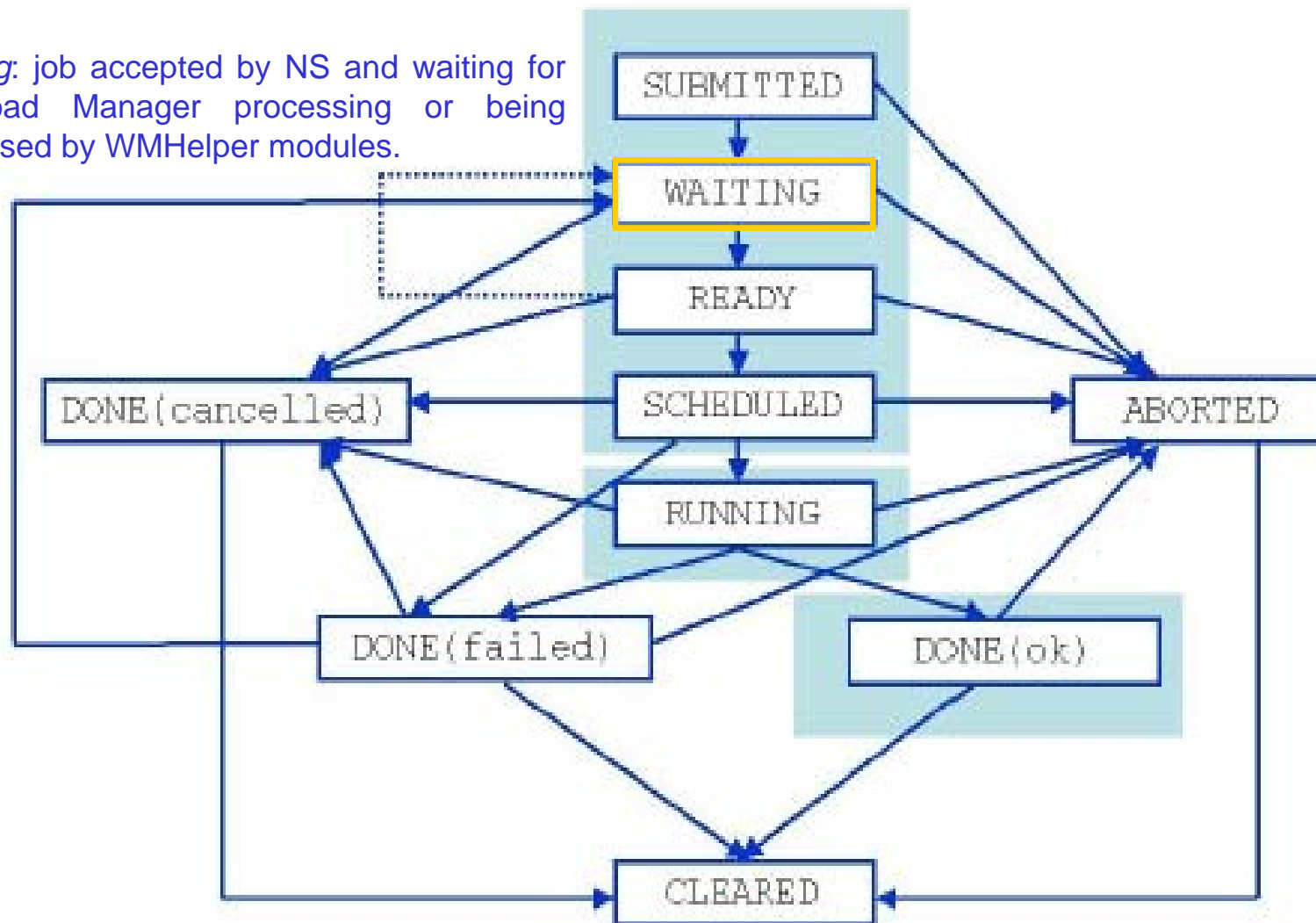
Flavia Donno



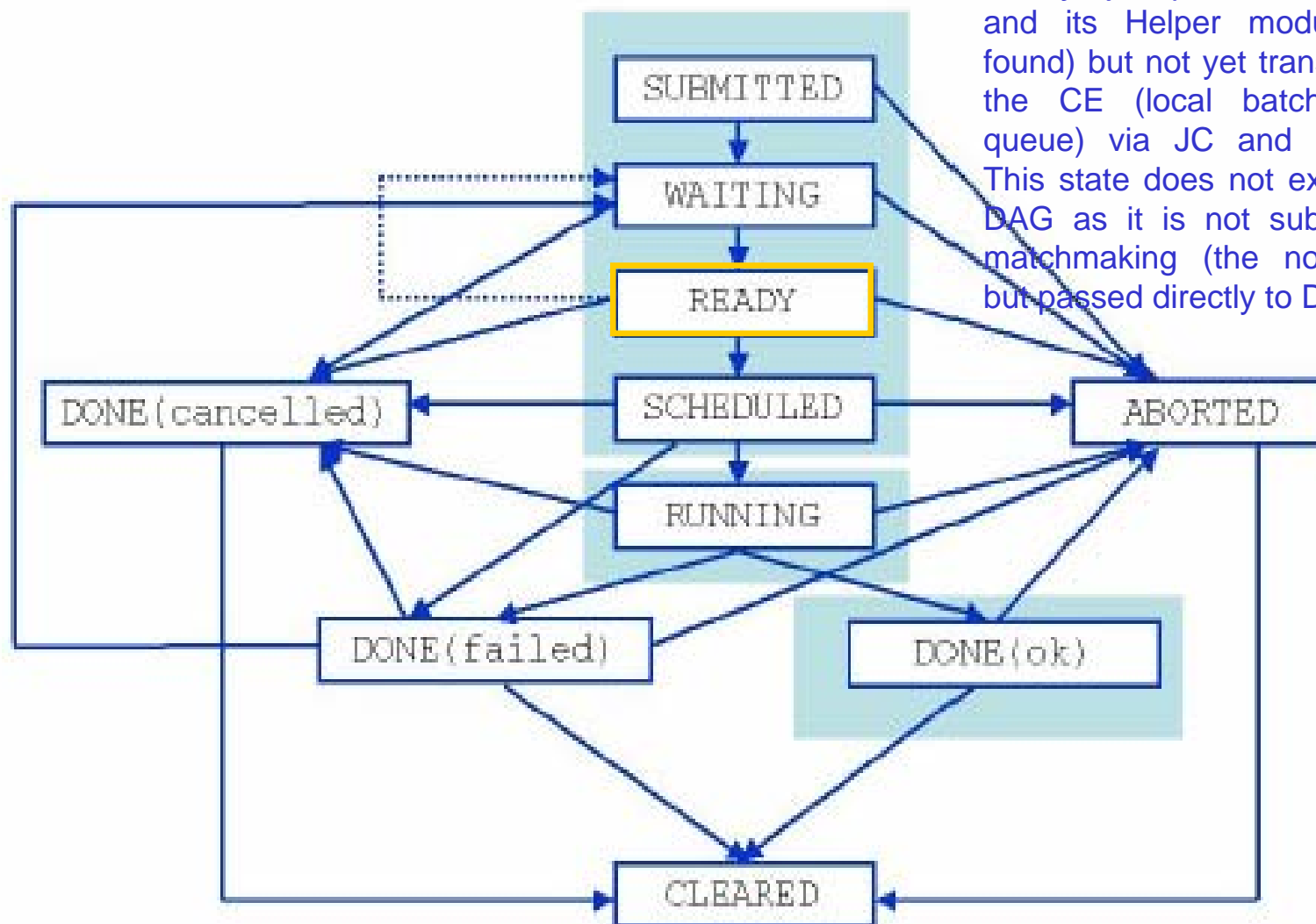
*Submitted:* job is entered by the user to the User Interface but not yet transferred to Network Server for processing

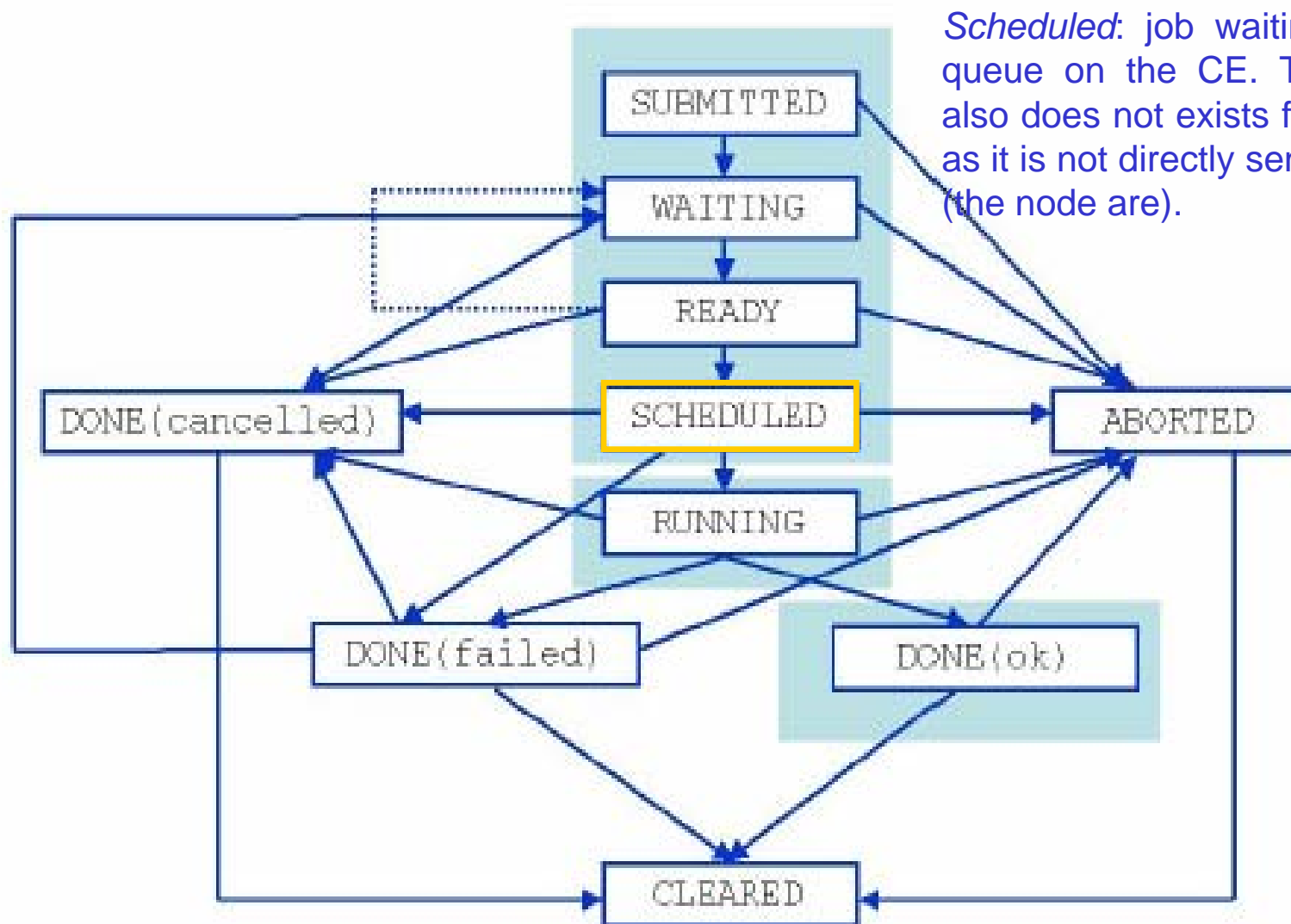


*Waiting:* job accepted by NS and waiting for Workload Manager processing or being processed by WMHelper modules.

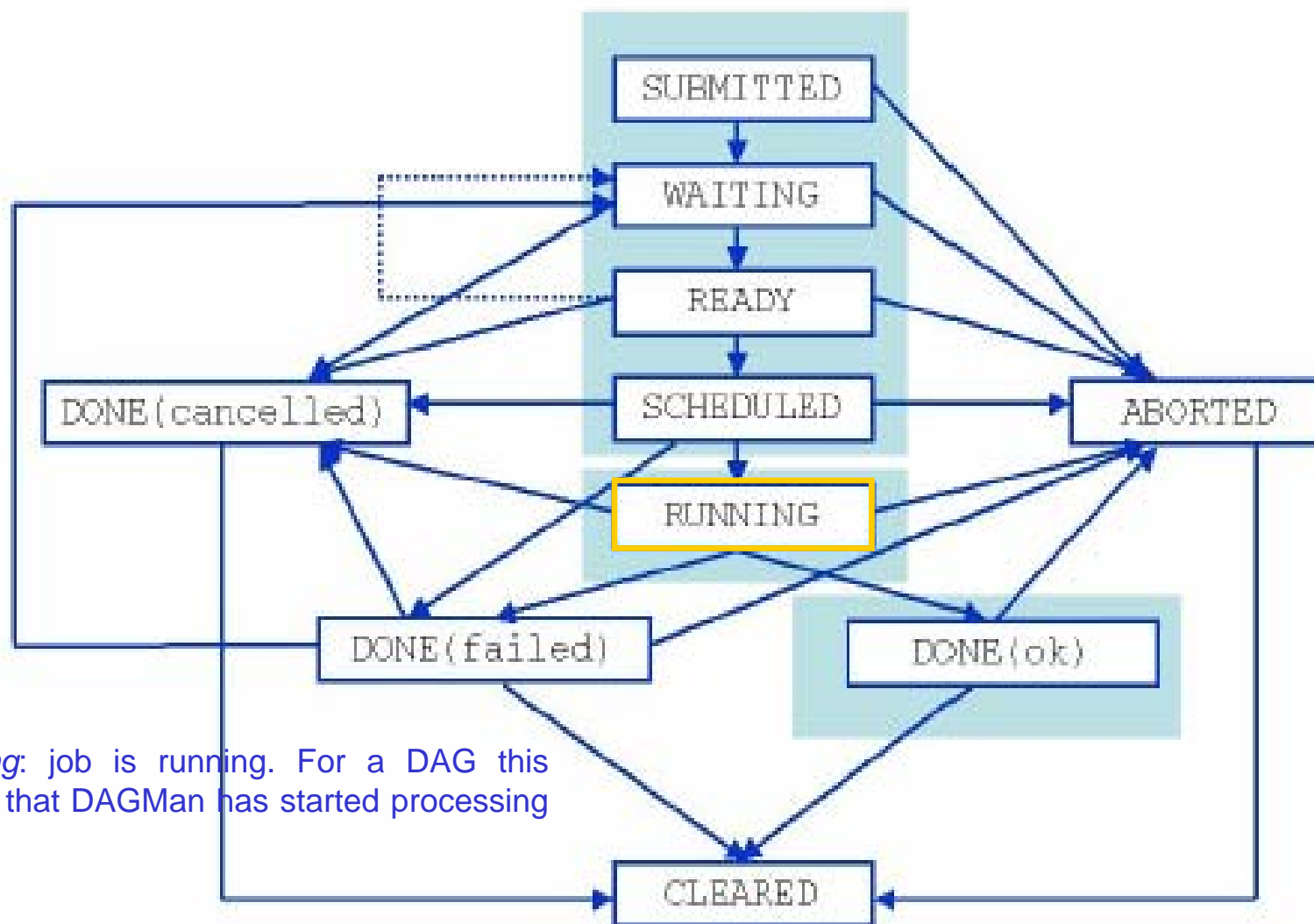


*Ready: job processed by WM and its Helper modules (CE found) but not yet transferred to the CE (local batch system queue) via JC and CondorC. This state does not exist for a DAG as it is not subjected to matchmaking (the nodes are) but passed directly to DAGMan.*

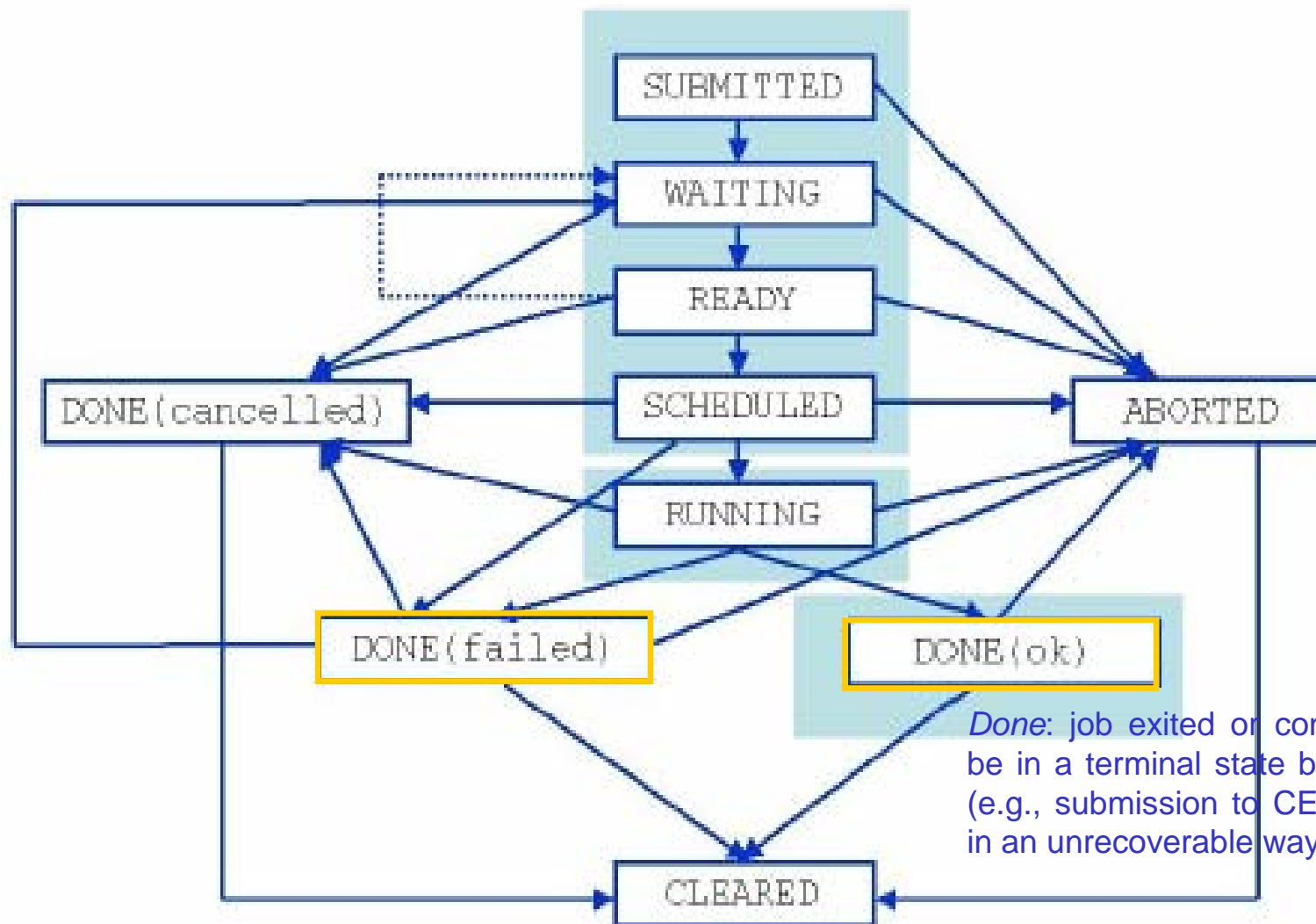




*Scheduled:* job waiting in the queue on the CE. This state also does not exist for a DAG as it is not directly sent to a CE (the node are).

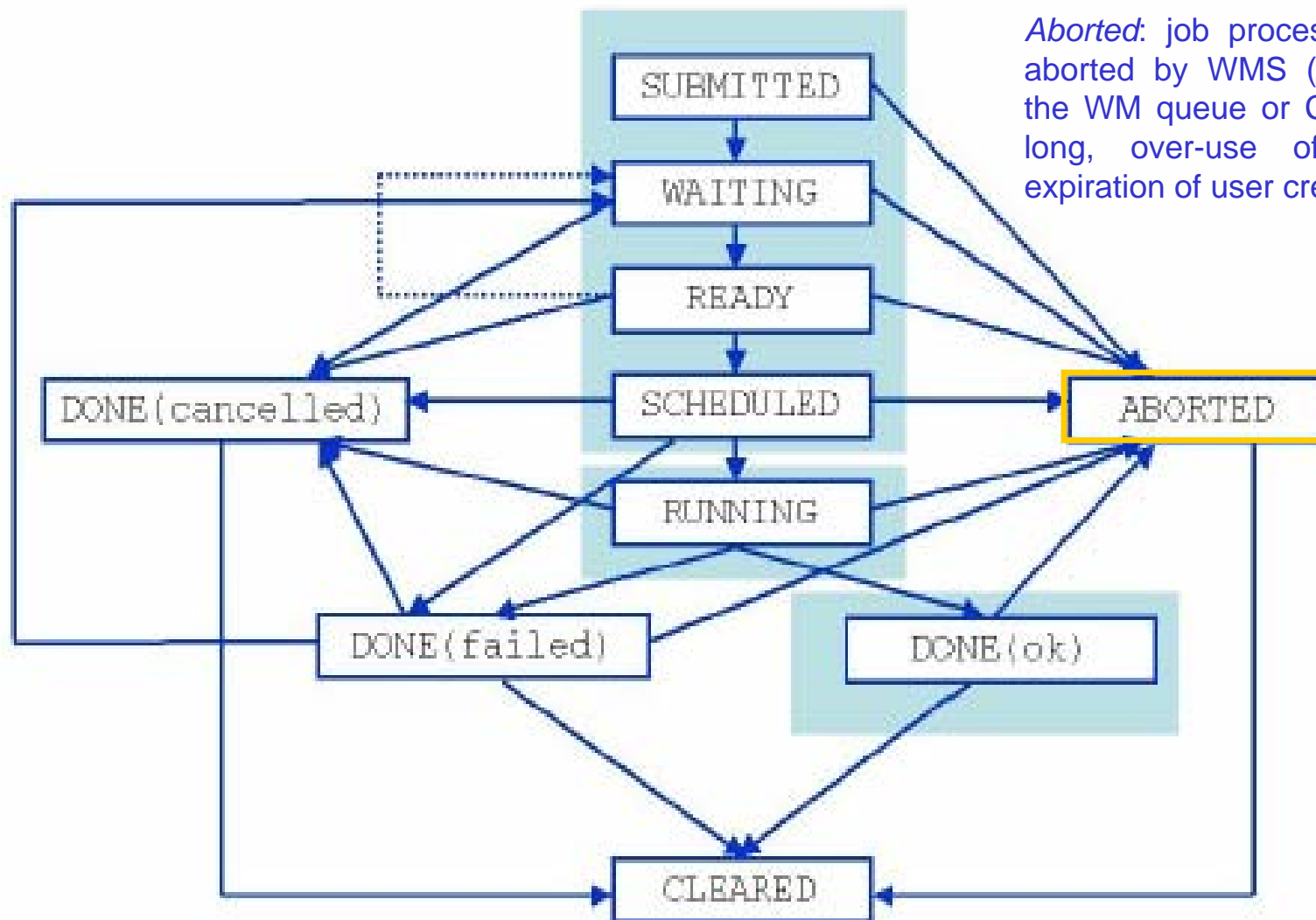


*Running:* job is running. For a DAG this means that DAGMan has started processing it.

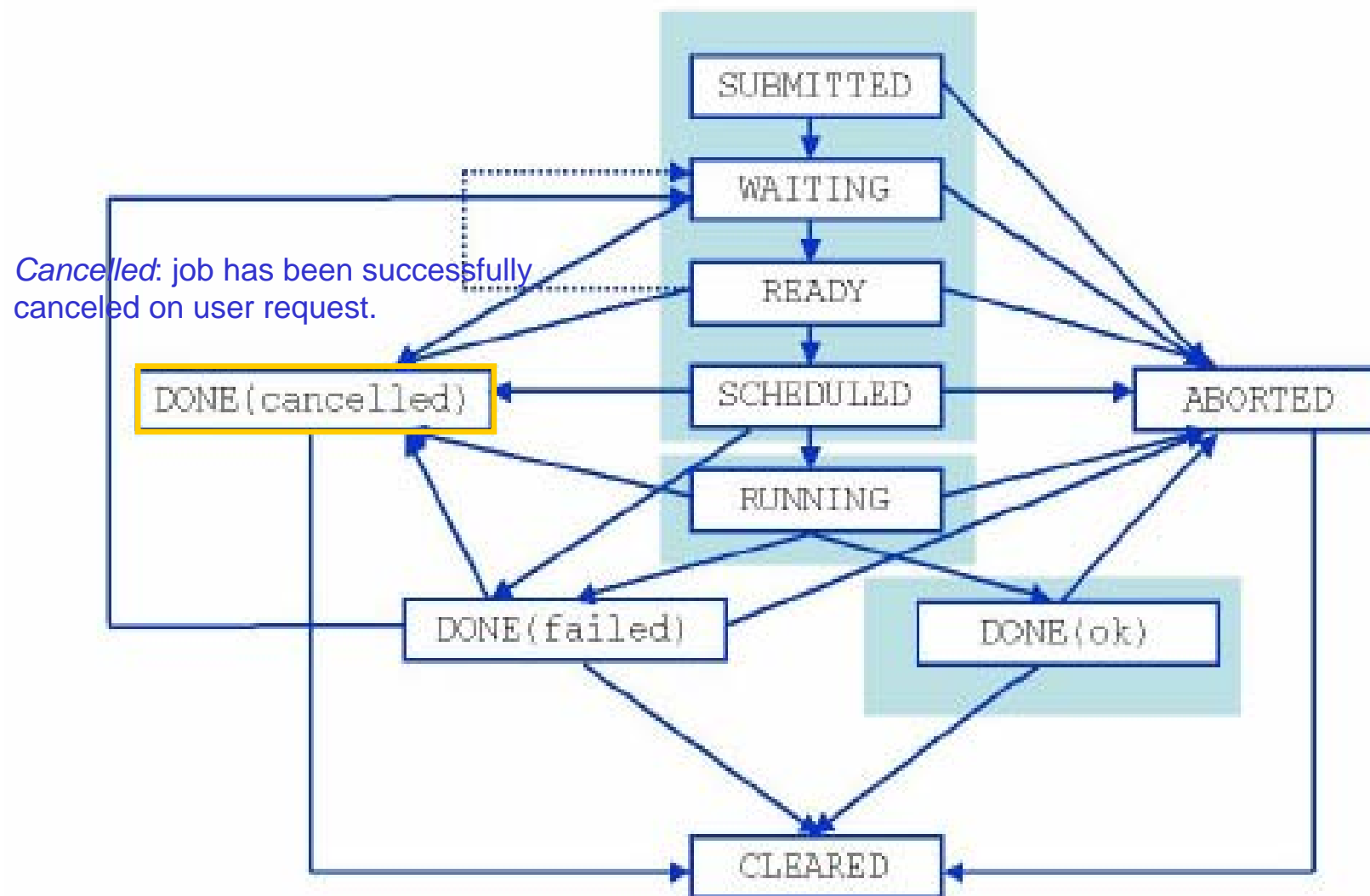


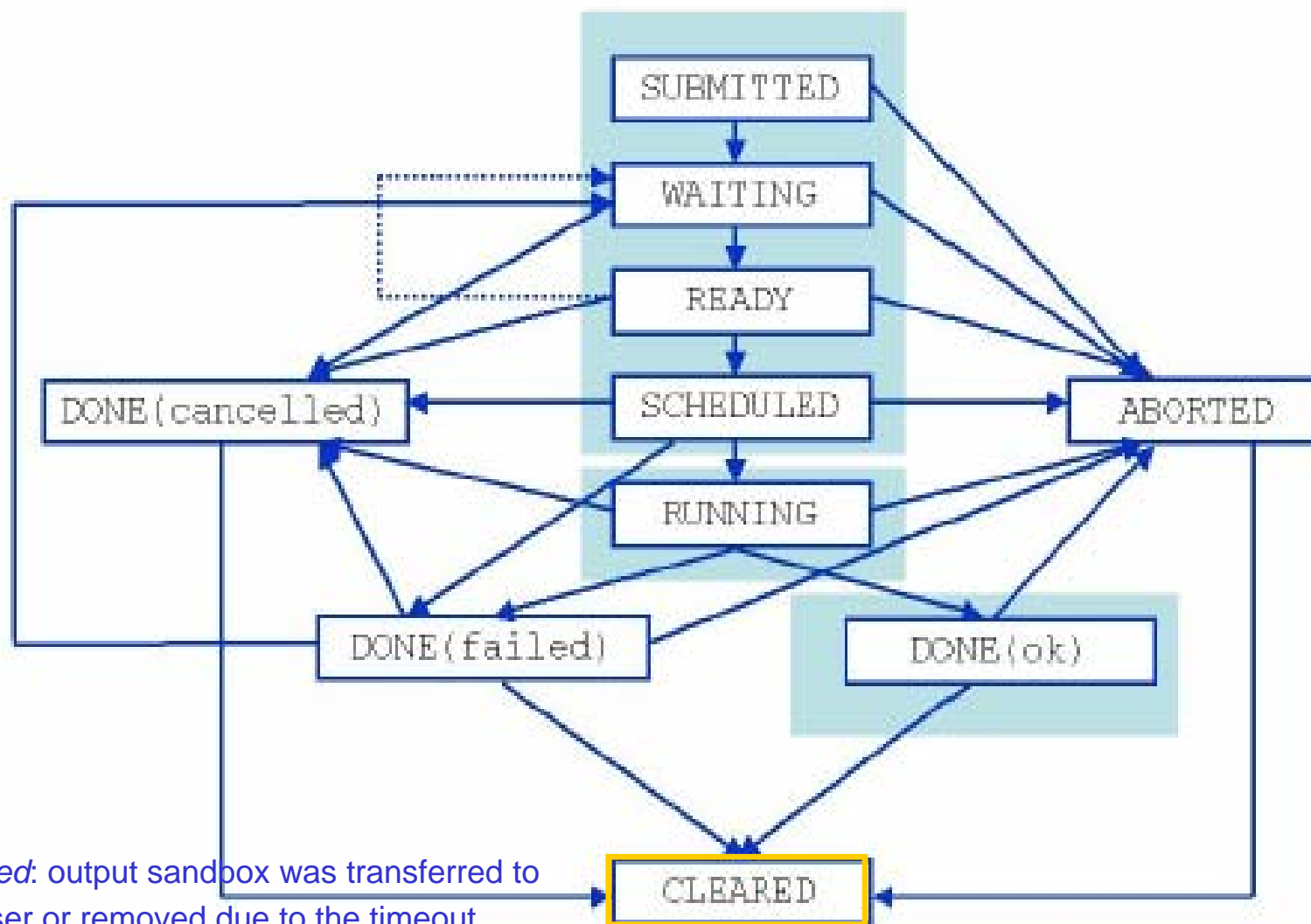
*Done: job exited or considered to be in a terminal state by CondorC (e.g., submission to CE has failed in an unrecoverable way).*





*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).





- **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
- **APIs Overview**

- 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

- **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**

- 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*

- 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



- 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)

- **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

```
[  
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;  
]
```

```
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*

- 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:  $\min(\text{RetryCount}, \text{MaxRetryCount})$ 
  - 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

- `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

- The WMS makes C++ and Java APIs available for UI, LB consumer and client.
- In the following document:

[http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1\\_2.pdf](http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf)

details about the rpms containing the APIs are given.

- Correspondent doxygen documentation can be found in share/doc area. Ex.:

[\\$EDG\\_LOCATION/share/doc/edg-wl-ui-api-cpp-lcg2.1.49/html](#)

- BrokerInfo CLI and APIs are described:

[http://server11.infn.it/workload-grid/docs/edg-brokerinfo-user-guide-v2\\_2.pdf](http://server11.infn.it/workload-grid/docs/edg-brokerinfo-user-guide-v2_2.pdf)

```
% ./workload Hello.jdl lxb0704.cern.ch 7772 lxb0704.cern.ch 9000
```

```
#include <iostream>
#include <string>

#include "edg/workload/logging/client/JobStatus.h"
#include "edg/workload/common/utilities/Exceptions.h"
#include "edg/workload/common/requestad/JobAd.h"
#include "edg/workload/userinterface/client/Job.h"

using namespace std ;
using namespace edg::workload::common::utilities ;
using namespace edg::workload::logging::client ;
/* *****
 * Example based on edg-wl-job-submit.cpp, edg-wl-job-status.cpp
 * for further examples see also:
 *
 * http://isscvcs.cern.ch:8180/cgi-
 * bin/cvsweb.cgi/workload/userinterface/test/?cvsroot=lcgware
 *
 * author: Heinz.Stockinger@cern.ch
 *
 * Example usage on GILDA:
 * ./workload Hello.jdl grid004.ct.infn.it 7772 grid004.ct.infn.it 9000
 */
```



```
int main (int argc, char *argv[])
{
    cout << "Workload Management API Example " << endl;

    try{
        if (argc < 6 || strcmp(argv[1], "--help") == 0) {
            cout << "Usage : " << argv[0]
                << " <JDL file> <ns host> <ns port> <lbHost> <lbPort> [<ce_id>]"
                << endl;
            return -1;
        }

        edg::workload::common::requestad::JobAd jab;

        jab.fromFile ( argv[1] );
        edg::workload::userinterface::Job job(jab);
        job.setLogLevel (6) ;

        cout << "Submit job to " << argv[2] << ":" << argv[3] << endl;
        cout << "LB address: " << argv[4] << ":" << argv[5] << endl;
        cout << "Please wait..." << endl;

        // We now submit the job. If a CE is given (argv[6]), we send it directly
        // to the specified CE
        //
        if (argc == 6)
            job.submit (argv[2], atoi(argv[3]), argv[4], atoi(argv[5]), "");
        else
            job.submit (argv[2], atoi(argv[3]), argv[4], atoi(argv[5]), argv[6]) ;

        cout << "Job Submission OK; JobID= "
            << job.getJobId()->toString() << endl << flush ;
    }
}
```

- The JobAd class provides users with management operations on JDL files
- We instantiate a Job object that corresponds to our JDL file and handles our job

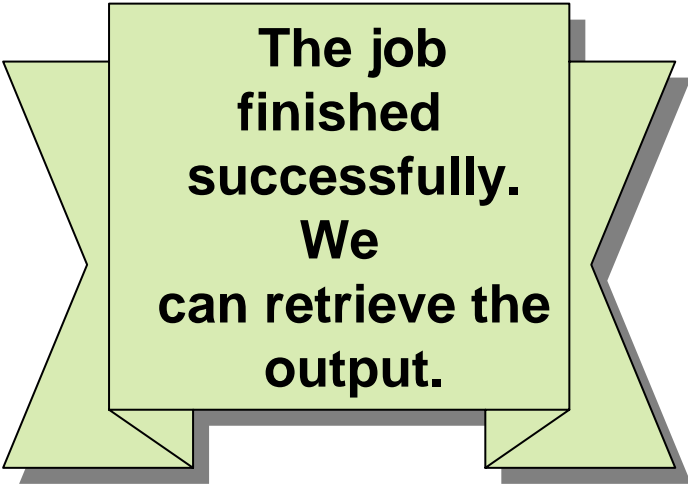
```
// Print some detailed error information in case the job did not
// succeed.
//
if ((status.status == 8) || (status.status == 9)) {
    printStatus(status);
    exit(-1);
}

// Now that the job has successfully finished, we retrieve the output
//
string outputDir = "/tmp";
job.getOutput(outputDir);

cout << "\nThe output has been retrieved and stored in the directory "
      << outputDir << endl;

return 0;

} catch (Exception &exc){
    cerr << "\nWMS Error\n";
    cerr << exc.printStackTrace();
}
return -1;
}
```



**The job  
finished  
successfully.  
We  
can retrieve the  
output.**

```

C          = gcc-3.2.2
GLOBUS_FLAVOR = gcc32

ARES_LIBS = -lares
BOOST_LIBS = -L/opt/boost/gcc-3.2.2/lib/release -lboost_fs \
             -lboost_thread -lboost_regex
CLASSAD_LIBS = -L/opt/classads/gcc-3.2.2/lib -lclassad
EXPAT_LIBS = -lexpat
GLOBUS_THR_LIBS = -L/opt/globus/lib -lglobus_gass_copy_gcc32dbgpthr \
                 -lglobus_ftp_client_gcc32dbgpthr -lglobus_gass_transfer_gcc32dbgpthr \
                 -lglobus_ftp_control_gcc32dbgpthr -lglobus_io_gcc32dbgpthr \
                 -lglobus_gss_assist_gcc32dbgpthr -lglobus_gssapi_gsi_gcc32dbgpthr \
                 -lglobus_gsi_proxy_core_gcc32dbgpthr \
                 -lglobus_gsi_credential_gcc32dbgpthr \
                 -lglobus_gsi_callback_gcc32dbgpthr -lglobus_oldgaa_gcc32dbgpthr \
                 -lglobus_gsi_sysconfig_gcc32dbgpthr \
                 -lglobus_gsi_cert_utils_gcc32dbgpthr \
                 -lglobus_openssl_gcc32dbgpthr -lglobus_proxy_ssl_gcc32dbgpthr \
                 -lglobus_openssl_error_gcc32dbgpthr -lssl_gcc32dbgpthr \
                 -lcrypto_gcc32dbgpthr -lglobus_common_gcc32dbgpthr

GLOBUS_COMMON_THR_LIBS = -L/opt/globus/lib -L/opt/globus/lib \
                        -lglobus_common_gcc32dbgpthr
GLOBUS_SSL_THR_LIBS = -L/opt/globus/lib -L/opt/globus/lib \
                     -lssl_gcc32dbgpthr -lcrypto_gcc32dbgpthr
VOMS_CPP_LIBS = -L/opt/edg/lib -lvomsapi_gcc32dbgpthr

all: workload

```

## Makefile

```

workload: workload.o
    $(CC) -o workload \
    -L$(EDG_LOCATION)/lib -ledg_wl_common_requestad \
    -lpthread \
    -ledg_wl_userinterface_client \
    -ledg_wl_exceptions -ledg_wl_logging \
    -ledg_wl_loggingpp \
    -ledg_wl_globus_ftp_util -ledg_wl_util \
    -ledg_wl_common_requestad \
    -ledg_wl_jobid -ledg_wl_logger -ledg_wl_gsisocket_pp \
    -ledg_wl_checkpointing -ledg_wl_ssl_helpers \
    -ledg_wl_ssl_pthr_helpers \
    $(VOMS_CPP_LIBS) \
    $(CLASSAD_LIBS) $(EXPAT_LIBS) $(ARES_LIBS) \
    $(BOOST_LIBS) \
    $(GLOBUS_THR_LIBS) \
    $(GLOBUS_COMMON_THR_LIBS) \
    $(GLOBUS_SSL_THR_LIBS) \
    workload.o

workload.o: workload.cpp
    $(CC) -I $(EDG_LOCATION)/include \
    -I/opt/classads/gcc-3.2.2/include -c workload.cpp

clean:
    rm -rf workload workload.o

```