Università di Bari Aldo Moro
CdL Magistrale in Informatica

Corso di Modellazione Sistemi Distribuiti Cooperativi

Seminario su: Modellazione mediante ASM di EMS in un ambiente di Grid Computing

Dott. Luciano Manelli

Grid - OGSA

L'architettura di un sistema GRID è stata definita da I.Foster e C.Kesselman quale “wide-scale distributed computing infrastructure to support large computing resources sharing and cooperating to solve problems in dynamic multi-institutional Virtual Organizations”.

Sono state definite tre principali caratteristiche:

1) a large-scale coordinated management of resources belonging to different administrative domains,

2) standard, open, multi-purpose protocols and

3) good performance parameters.

Grazie all’uso di risorse distribuite eterogenee, i grid systems stanno divenendo una valida alternativa ai tradizionali sistemi distribuiti

Grid - OGSA

The OGSA standard:

- describes requirements (such as interoperability and resource sharing, optimisation, quality of service, job execution, data services, security, scalability and extensibility) ;
- and considers six important independent capabilities needed to support grid systems and applications: Execution Management Services; Data Services; Resource Management Services; Security Services; Self-Management Services; and Information Services.

Grid - OGSA

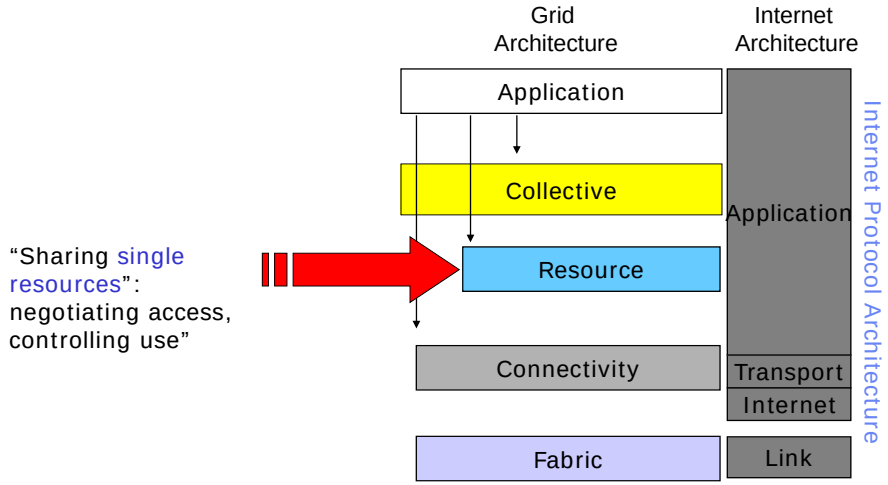
In particular the Execution Management Services (EMS) address the job management and execution capability of a grid system and it is concerned with the research of candidate locations for

- execution,
- preparation for execution,
- initiating and managing the execution of jobs until the end.

Grid - OGSA

These requirements are also partially fulfilled by OGSA specifications in the Basic Execution Service. In particular, any grid middleware offers a job management and execution capability, as it enables users to use distributed resources for computationally intensive applications. In fact, EMS are also implemented in the Globus Toolkit and in gLite, both used in several grid deployments.

Computing Services in the Layered Grid Architecture



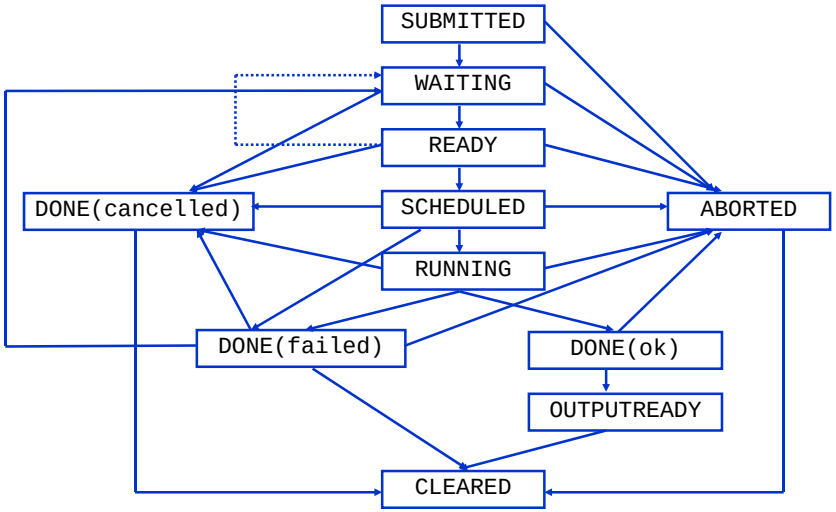
Job types

- Sequential, batch jobs
- Parallel (MPI) jobs
- Checkpointable jobs
- Interactive jobs
- DAG jobs (set of jobs with inter-dependencies modeled with Directed Cycle-Free Graphs)
- Partitionable jobs
  - Jobs to be partitioned within the CE

Job States

- **SUBMITTED**: the user has submitted the job via UI
- **WAITING**. the WMS has received the job
- **READY**: A CE, which matches job requirements, has been selected, and the job is transferred to the JSS
- **SCHEDULED**: the JSS has sent the job to the CE
- **RUNNING**: the job is running on the CE
- **DONE**: this state has different meanings:
  - **DONE (ok)** : the execution has terminated on the CE (WN) with success
  - **DONE (failure)** : the execution has terminated on the CE (WN) with some problems
  - **DONE (cancelled)** : the job has been cancelled with success
- **OUTPUTREADY**: the output sandbox is ready to be retrieved by the user
  - reflects the time difference between end of computation on CE and the moment WMS got necessary notification about job termination.
- **CLEARED**: the user has retrieved all output files successfully, and the job bookkeeping information is purged some time after the job enters in this state.
- **ABORTED**: the job has failed
  - The job may fail for several reasons one of them is external to its execution (no resource found).

State Diagram



Grid - OGSA

Unfortunately, a uniform access to resources is not available across these two different middlewares. That means, for example, that jobs originated on Globus Toolkit cannot be forwarded to gLite, even if they have access authorizations to resources.

Grid - OGSA

So, the use of grid formal models can help the high-level middleware design with the reduction of the risk that a change in the dynamics specifications could have a large impact on the specification of other aspects. Furthermore, the grid model-based approaches allow the specification of dynamic aspects in a more intuitive way, without the necessity of having programming skills.

Grid - OGSA

The system is always initially in a state of inactivity, *i.e.* waiting for a job is submitted. When a client application uses the grid system, a job is submitted. A user can also cancel the submitted job. A job is the smallest unit that the grid system manages. The system controls the availability of needed resources. Available resources are those resources that meet the system necessities (the "matchmaking"). Every different grid middleware controls the resource discovery, allocation and reallocation in different way, that addresses questions of efficiency, stability and scalability and each resource is controlled by its owner host.

Grid - OGSA

If the necessary resources are available, these are allocated and ready to begin the computation. The allocation consists of assigning and queuing the job, eventually scheduled, to their local manager, otherwise, the system returns inactive aborting the execution of the submitted job and returning ready for a new job, or for a different use by the host. If all the sub-processes are correctly performed, the execution of the submitted job can be considered completed without problems. On the other case, the system leaves all the resources and returns inactive. A resource could fail the execution of its job because of any problem. If a resource fails the job, this returns inactive.

Grid - OGSA

A grid system is constituted by a pool of distributed resources and it can be available for a job. It implements the following requirements.

Req.1 A job can be submitted to the grid; if there is no job, the system remains in a inactivity state.

Req.3 The grid middleware checks the matchmaking between resources and job constraints before the execution; if there are problems (*e.g.* lack of memory or of devices, or slow CPU speed, *etc.*) the system reject the job.

Req.4 After accepting the job the system runs it.

Req.5 If there are no failures the job is completed; on the other case the job fails.

Req.6 A user can cancel a job every time.

Req.7 At the end of the computation (job completed or aborted for every reason) every resource is released.

Req.8 If software or hardware errors occur the job is aborted.

Req.9 At the end of every computation the result is communicated to the end user.

Grid - OGSA

It is evident, from analyses requirements, that the system, during execution of a job, passes through various states. Considering the system states emerged in the process and requirements explanation, jobs traverse the following set of states:

IDLE. The system starts in state of inactivity.

READY. After the control of the availability of every resource, the system is enabled to start execution of a job on such a resource, which matches job requirements.

RUNNING. The job is executing on computational resources.

FAILED. The computation can fail due to some error or failure event.

DONE OK. The job is terminated successfully.

CANCELLED,REMOVED. The job has been successfully cancelled on user request.

ABORTED: job processing is aborted by grid middleware due to some error or failure event.

Grid - OGSA

We can note in Figure 1 the graphical representation of the system internal state flow.

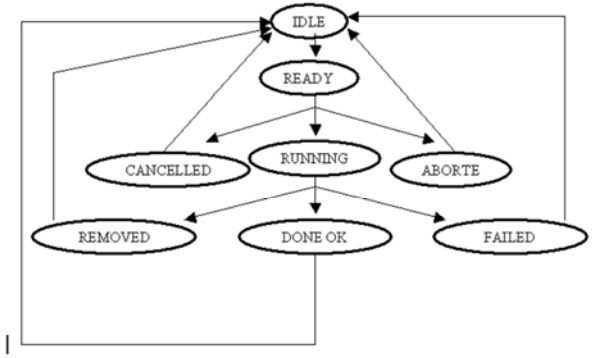


Figure 1: Graphical representation

Grid – OGSA - Rules

From State	RULE	To State
idle	SUBMIT	ready
ready	REJECT	aborted
ready	ACCEPT	running
ready	CANCEL	cancelled
running	COMPLETE	done ok
running	FAILURE	failed
running	REMOVE	removed
removed; failed; done ok; aborted; cancelled	FINISH or ERROR	idle

Table 1: Rules Table

Grid – OGSA – Locations 1

State 1: Idle

active : false

monitoredState : “IDLE”

State 2: Ready

active : true

loadJob(job, system) : the job is loaded

monitoredState : “READY”

Grid – OGSA – Locations 2

State 3: Running

allocateResources(system, job, resources) : the resource are allocated

addLocalQueue (job, resources) : the job is in queue at the resource and the local counter is incremented

run(job) : the job is running

monitoredState : “RUNNING”

Grid – OGSA – Locations 3

State 4: Done ok

minusLocalQueue (job, resources) : the local queue is changed and the local counter is decremented

completeSuccessJob : true

terminedLocalJob : true

showOutputMessageJob : “JOB OK”

monitoredState : “DONE OK”

Grid – OGSA – Locations 4

State 5: Cancelled

completeSuccessJob : false

cancelledJob : true

showOutputMessageJob : “JOB CANCELLED”

monitoredState : “CANCELLED”

Grid – OGSA – Locations 5

State 6: Removed

minusLocalQueue (job, resources) : the local queue is changed and the local counter is decremented

releaseResources(system, job, resources) : resources are released

completeSuccessJob : false

terminedLocalJob : true

removedJob : true

showOutputMessageJob : “JOB REMOVED”

monitoredState : “REMOVED”

Grid – OGSA – Locations 6

State 7: Aborted

completeSuccessJob : false

abortedJob : true

showOutputMessageJob : “ERROR”

monitoredState : “ABORTED”

Grid – OGSA – Locations 7

State 8: Failed

minusLocalQueue (job, resources) : the local queue is changed and the local counter is decremented

completeSuccessJob : false

terminedLocalJob : true

showOutputMessageJob : “JOB FAILED”

monitoredState : “FAILED”

Grid – OGSA – Rules - SUBMIT

if state= IDLE
if submittedJob is true
SUBMIT rule {
monitoredState:= "READY" (5)
active :=true
loadJob(job,system)
state := READY
}

Grid – OGSA – Rules - ACCEPT

if state= READY
if atLeastOneResourceForType(job,resources) is true
ACCEPT rule {
monitoredState:= "RUNNING"

allocateResources(system,job,resources)
addLocalQueue (job,resources)
run(job)
state := RUNNING
}

Grid – OGSA – Rules - REJECT

if atLeastOneResourceForType(job,resources) is false
REJECT rule {
monitoredState:= "ABORTED" (7)
completeSuccessJob := false
abortedJob := true
showOutputMessageJob:="ERROR
state :=ABORTED
}

Grid – OGSA – Rules - COMPLETE

if state= RUNNING
if notFailureAnySubProcs (job,resources) is true
and completeAllSubProcs (job,resources) is true
COMPLETE rule {
monitoredState:= "DONE OK"
completeSuccessJob := true
showOutputMessageJob:="JOB OK"
terminedLocalJob:=true
minusLocalQueue(job,resources)
state := DONE OK
}

Grid – OGSA – Rules - FAILURE

if state= RUNNING
if notFailureAnySubProcs (job,resources) is false
or completeAllSubProcs (job,resources) is false
FAILURE rule {
monitoredState:= "FAILED" (9)
completeSuccessJob := false
showOutputMessageJob:="JOB FAILED"
terminedLocalJob:=true
minusLocalQueue(job,resources)
state := FAILED
}

Grid – OGSA – Rules - CANCEL

if cancelRequest is true
CANCEL rule {
monitoredState:= "CANCELLED" (10)
showOutputMessageJob:="JOB CANCELLED"
completeSuccessJob := false
cancelledJob := true
state := CANCELLED
}

Grid – OGSA – Rules - REMOVE

if abortRunning is true
REMOVE rule {
monitoredState:= "REMOVED" (11)
showOutputMessageJob:="JOB REMOVED"
completeSuccessJob := false
terminedLocalJob:=true
removedJob := true
minusLocalQueue(job,resources)
releaseResources(system, job, resources)
state := REMOVED
}

Grid – OGSA – Rules - FINISH

if termination(job) is true
FINISH rule {
monitoredState:= "IDLE" (12)
active:=false
showOutput(showOutputMessageJob)
state := IDLE
}

Grid – OGSA – Rules - ERROR

if termination(job) is false

ERROR rule{

monitoredState:= "IDLE" (13)

active:=false

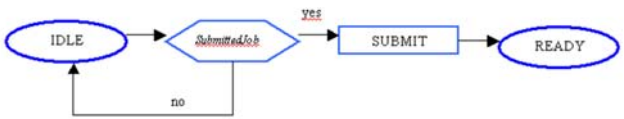
printStackTrace(system, job, resources)

recoverySystem(system, job, resources)

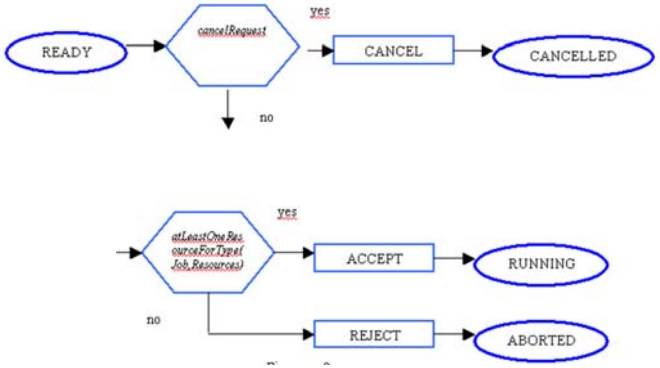
state := IDLE

}

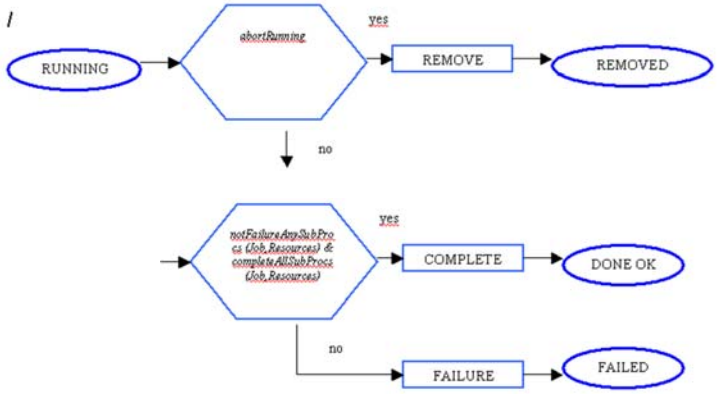
Grid – OGSA – Transition 1



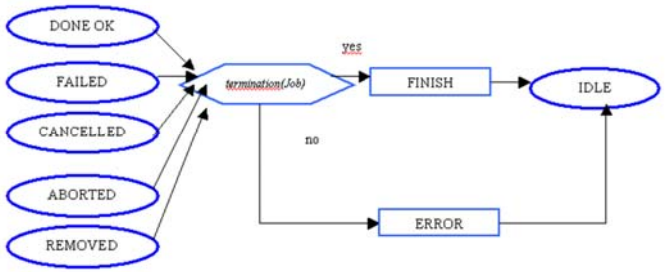
Grid – OGSA – Transition 2



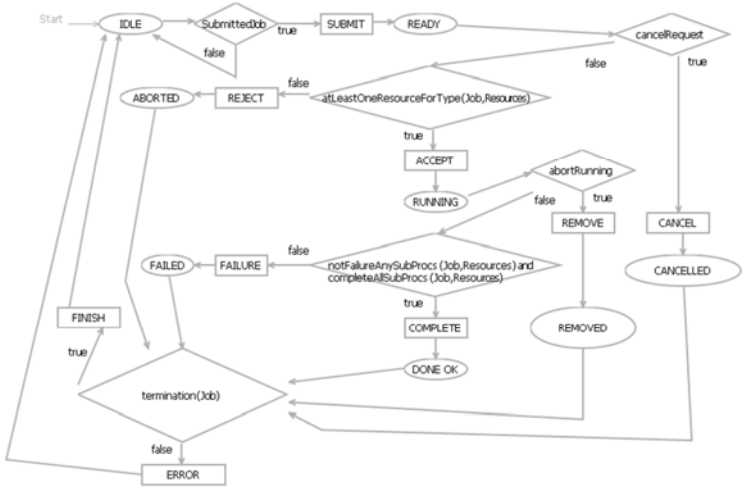
Grid – OGSA – Transition 3



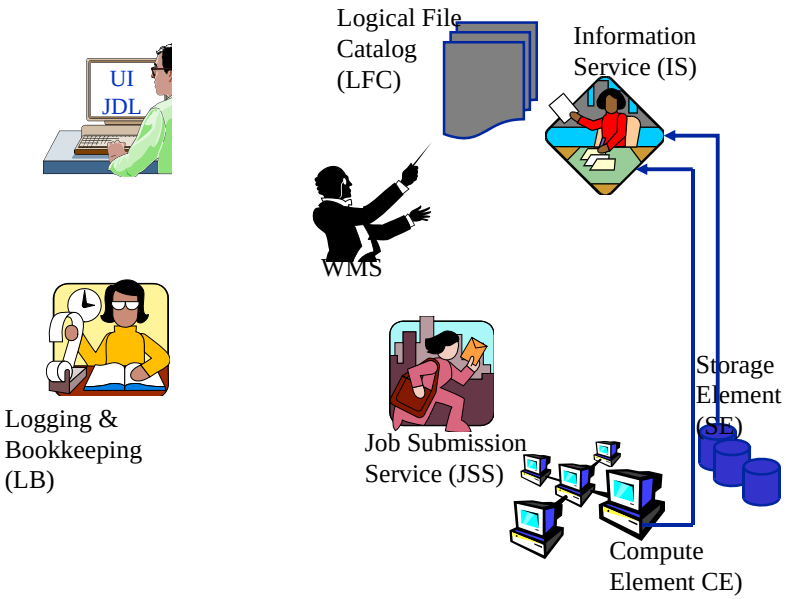
Grid – OGSA – Transition 4



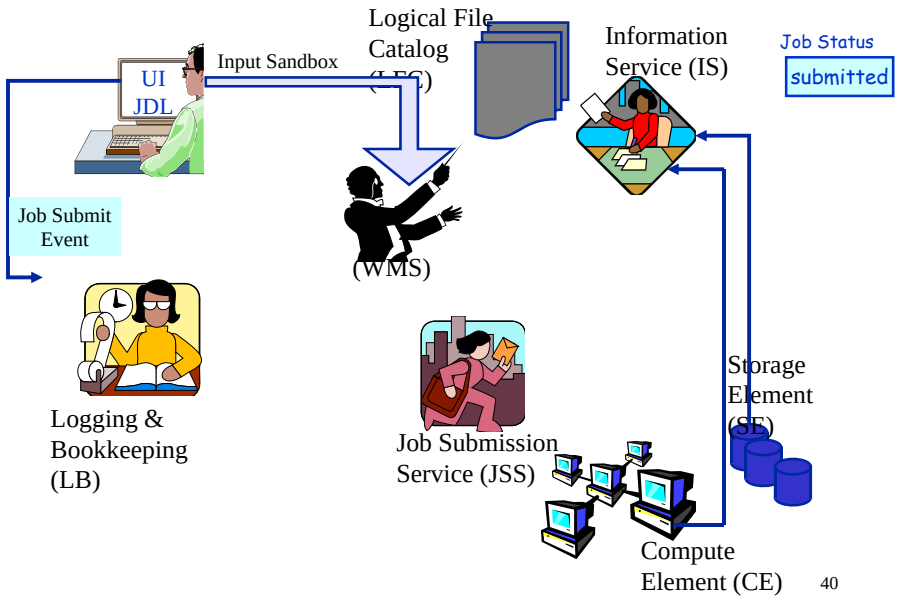
Grid - OGSA



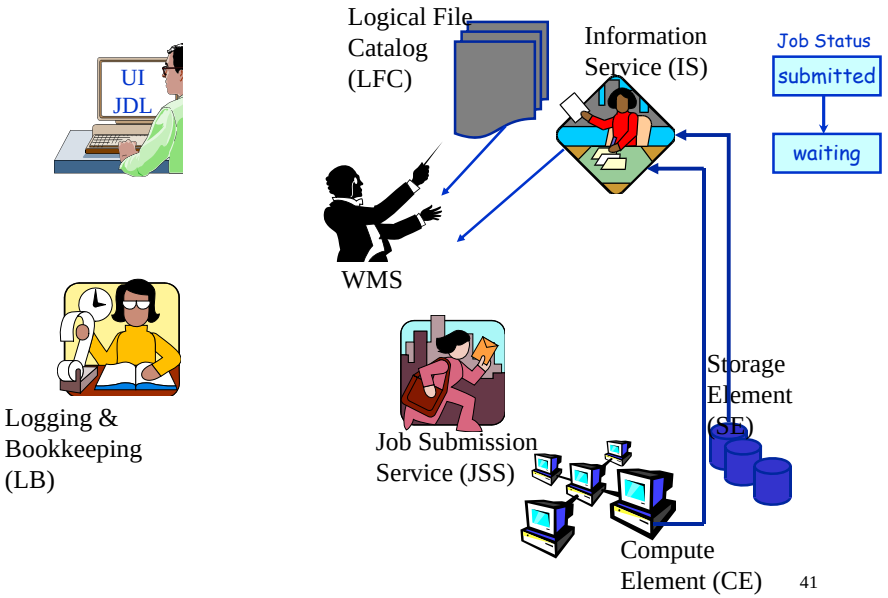
Job Submission Scenario



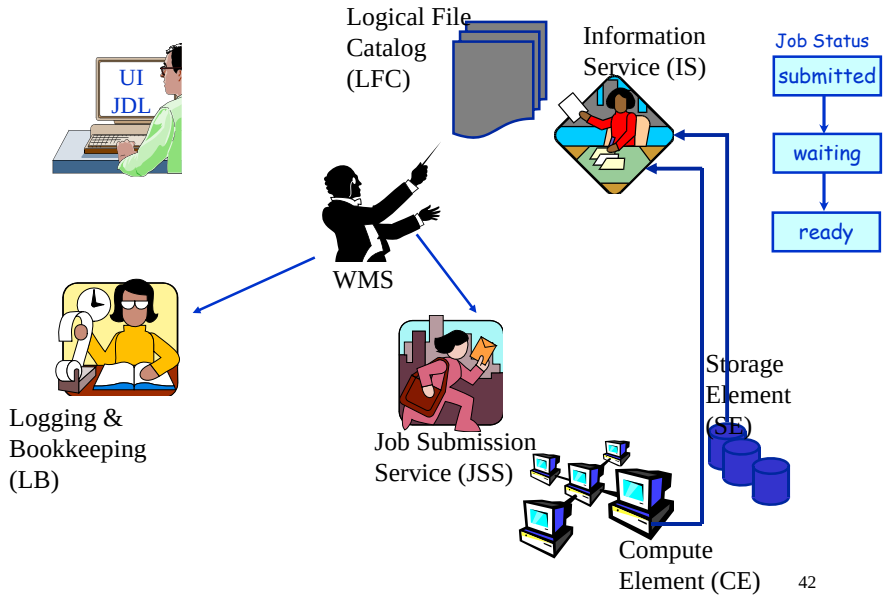
Job Submission Scenario



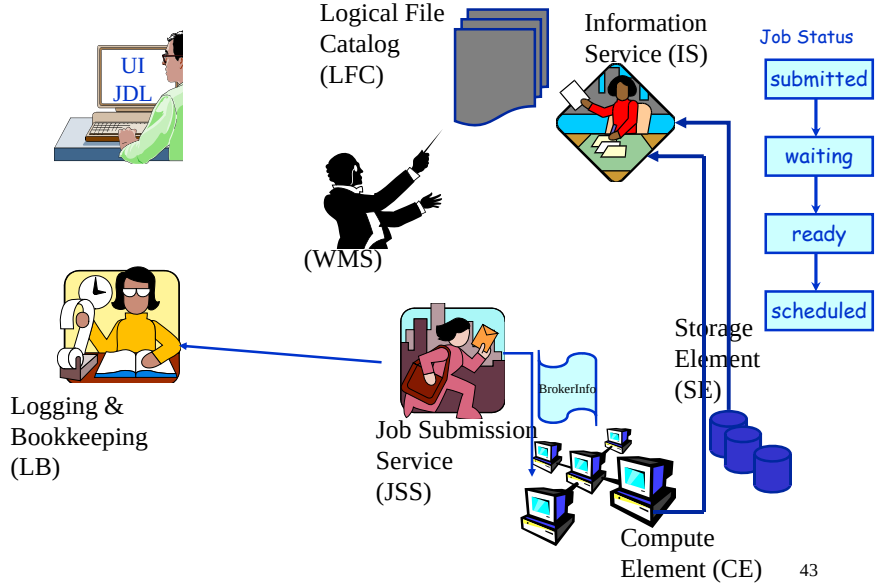
Job Submission Scenario



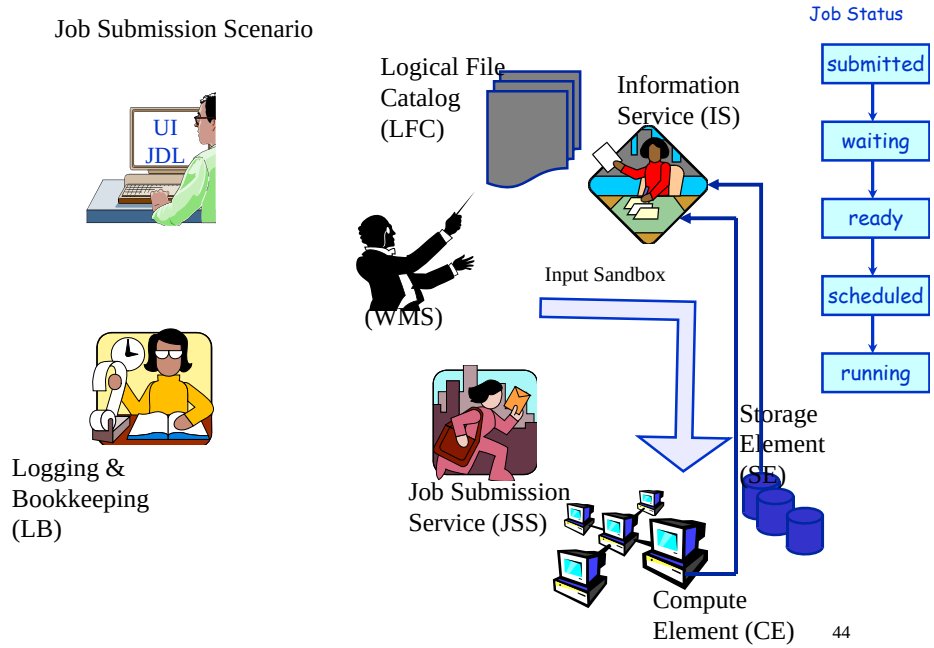
Job Submission Scenario



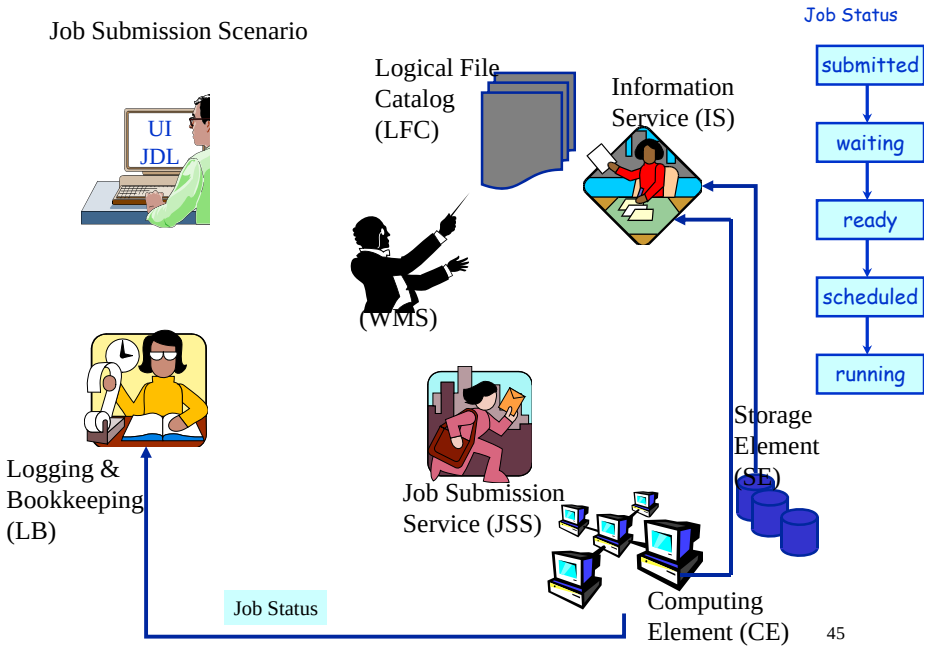
Job Submission Scenario



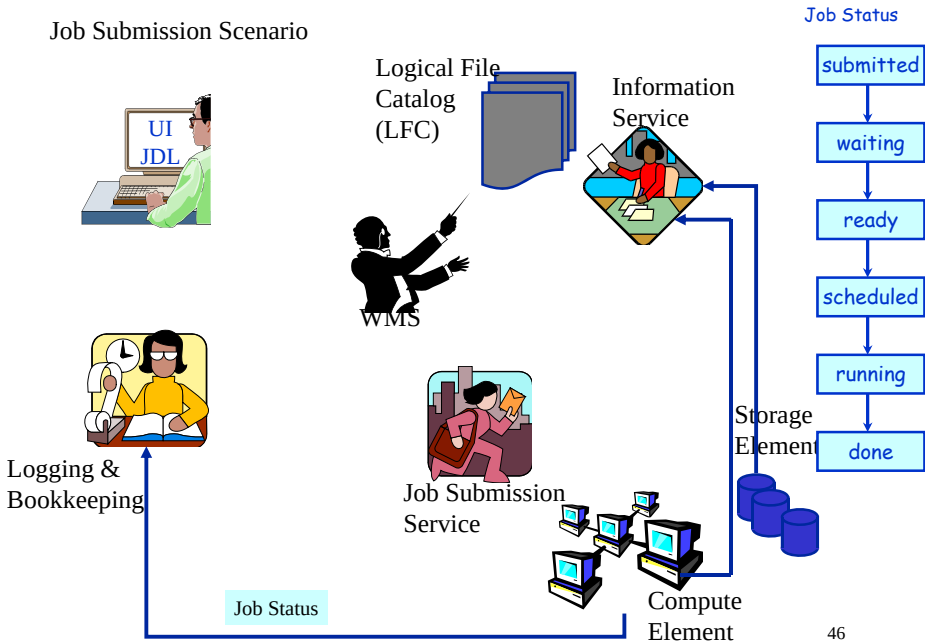
Job Submission Scenario



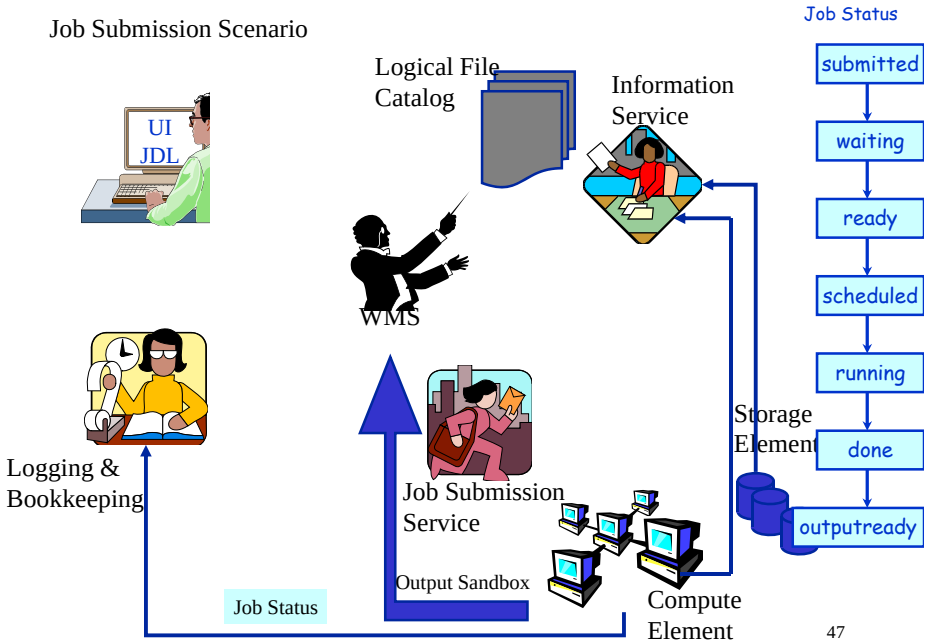
Job Submission Scenario



Job Submission Scenario



Job Submission Scenario



Job Submission Scenario