Langage de description des jobs (JDL)

Langage de description des jobs (JDL)

Job Description Langage - JDL

- **Un langage extensible de description des tâches ou jobs pour leurs exécutions en grille**
  - ✓ Utilise les attributs pour la description des jobs,
  - ✓ Spécifie les caractéristiques d'un job relatif à une application, programme exécutable, données d'entrés, etc.…
  - ✓ Définie les caractéristiques des ressources préférées et pré-requis,
  - ✓ Basé sur un language de classes CLASSAD "CLASSified ADvertisement" du Condor
- **JDL définie un ensemble d'attributs pour le WMS groupés en 2 catégories:**
  - ✓ Les attributs du job:
    - Executable, Arguments, StdInput/StdOutput/StdError, OutputSandbox …
  - ✓ Les attributs des ressources:
    - MinPhysicalMemory, MinLocalDiskSpace, FreeCPUs, RunningJobs, …

Langage de description des jobs (JDL)

JDL : Syntaxe

- Un fichier jdl consiste a un ensemble de lignes d'entrées qui se terminent par ';' chacune
  - ✓ Ligne d'entrée : <attribut> = <valeur> | < liste de valeurs >;
  - ✓ Attribut : Une chaine qui représente le nom de l'attribut
  - ✓ Valeur : Une chaine qui représente la valeur de l'attribut
    - Chaine : "abc" une double quotte pour la chaine
    - Nombre entier : 1234
    - Nombre réel : 12.52
    - Booléen: "true", "false", expression (voir GLUESchema)
  - ✓ Liste de valeurs : regroupées par <{ }> et séparées par <,>
    Ex : { "abc" , "bcd" , "def" }
- Commentaires sont précédés par (#), pour C/C++ (/* ceci est un commentaire */).

- Attention: JDL est sensible pour les caractères 'espace' ou 'tabulation'
- Pas de caractère espace ou tabulation doivent être à la fin de ligne.

Langage de description des jobs (JDL)

JDL : Les attributs de job – 1/7

- L'attribut Type est une chaine qui indique le type du job à exécuter
  - ✓ Syntaxe Ex. : Type = "Job";
  - ✓ Valeur possibles :
    - Job
    - DAG
    - Collection
- Si la valeur de cet attribut n'est pas spécifiée dans le fichier jdl, WMS la met à "Job".

Langage de description des jobs (JDL)

JDL : Les attributs de job – 2/7

- L'attribut JobType est une chaine ou plusieurs chaines qui représentent le type de job décrit par JDL
  - ✓ Syntaxe :

  Ex: JobType = "Interactive"; ou JobType = {"Checkpointable" ", "MPICH"};
  - ✓ Valeur possibles :
    - "Normal" Job simple
    - "Interactive" Job qui interagit avec l'utilisateur qui l'a soumis
    - "MPICH" Job parallèle MPI
    - "Partitionable" Job composé d'un ensemble d'étapes indépendantes /itérations pour l'exécution parallèle
    - "Checkpointable" Job capable de gardé son état suspendu et complet a partir d'un même point
    - "Parametric" Job avec des attributs paramétriques dans son JDL, pour soumissionner plusieurs instance similaires avec des commandes simples.

Langage de description des jobs (JDL)

JDL : Les attributs de job – 3/7

- L'attribut Executable est une chaine qui représente le nom de la commande ou de l'exécutable
- Un utilisateur peut spécifier un exécutable déjà installé sur le cluster, dans ce cas le chemin absolu doit être indiqué:

  Ex : Executable = "/usr/local/java/j2sdk1.4.0_01/bin/java";
- Une autre possibilité est de fournir un exécutable local, qui et copié de UI aux WN. Dans ce cas, le fichier exécutable doit être spécifié comme exécutable et son chemin doit être listé parmi les valeurs de l'attribut InputSandbox, pour le rendre disponible sur le WN

  Ex :

```
Executable = "cms_sim.exe";
InputSandbox = {"/home/edguser/sim/cms_sim.exe", … };
```

Langage de description des jobs (JDL)

JDL : Les attributs de job – 4/7

- L'attribut Arguments est une chaine qui contient la liste des arguments de la commande du job

  Ex : Un exécutable 'sum' qui calcule la somme de 2 arguments

  $ sum N1 N2 –out result.out

- Il est décrit de la façon suivante :

  Ex :

```
Executable = "sum";
Arguments = "N1 N2 –out result.out";
```

Langage de description des jobs (JDL)

JDL : Les attributs de job – 5/7

- L'attribut StdInput est une chaine qui représente l'entrée standard du job, c'est-à-dire c'est comme si le job s'exécute de la façon suivante:

  $> job < (standard input file)

  Il peut être un chemin absolu du fichier

  StdInput = "/var/tpm/jobInput";

  Ou bien juste le nom du fichier

  StdInput = "myjobInput";

  ➔ le fichier est disponible sur le WN où sera exécuté le job

Langage de description des jobs (JDL)

JDL : Les attributs de job – 6/7

- L'attribut StdOutput est une chaine qui représente le nom de fichier où la sortie standard du JDL sera sauvegardée, l'utilisateur peut spécifier le nom de fichier ou le chemin relatif du fichier

  StdOutput = "myjobOutput";

  StdOutput = "event1/myjobOutput";

- L'attribut StdError est une chaine qui représente le nom de fichier où la sortie d'erreur standard du JDL sera sauvegardée, l'utilisateur peut spécifier le nom de fichier ou le chemin absolu/relatif du fichier

  StdError = "myjobError";

  StdError = "/var/tmp/myjobError";

  StdError = "event1/myjobError";

Langage de description des jobs (JDL)

JDL : Les attributs de job – 7/7

- L'attribut InputSandbox est une chaine ou liste de chaines qui identifie la liste des fichiers du disque local de UI dont a besoin le job pour son exécution.

  InputSandbox = { "/home/user/test.exe", "test.sh};

- L'attribut OutputSandbox est une chaine ou liste de chaines qui identifie la liste des fichiers générés par le job dans le WN, dont il a besoin l'utilisateur sur le UI. Les fichiers sont transférés à la fin du job au WMS et peuvent être téléchargés sur le disque local de UI.

  OutputSandbox = { "myjobOutput", "myjobError", };

- InputSandbox & OutputSandbox ne doivent pas contenir des fichiers avec les mêmes noms (sauf si avec différents chemins), même chose lors du transfert vers UI.

Langage de description des jobs (JDL)

JDL : Les attributs de ressources – 1/4

- Utilisés par l'ordonnanceur (RB) pour définir la ressource utilisée pour l'exécution du job
- Permettent de définir les caractéristiques de calcul requises
  - ✓ Représentent les valeurs des attributs "**Requirements**" et "**Rank**"
  - ✓ Sont définis à l'aide du préfixe "**other.**"
- Définissent les caractéristiques liées aux données
  - ✓ Ce sont : les données entrantes , l'élément de stockage où les données sont prises ou bien mises, les protocoles,…

Langage de description des jobs (JDL)

JDL : Les attributs de ressources; Requirements – 2/4

- Besoins du job vis à vis des ressources de calcul.
- Sont spécifiés à partir des attributs qui sont définis dans le système d'information de la grille (IS).
- S'ils ne sont pas définis dans le jdl, ce sont les valeurs définies par défaut dans l'UI qui sont utilisées
  - ✓ Default: **other.GlueCEStateStatus == "Production"** (Les ressources utilisables devront absolument présenter l'attribut demandé)
- Syntaxe : Requirements = < boolean expression >
  - ✓ C'est un booléen de l'expression ClassAd utilisant comme syntaxe d'opérateurs celle de langage C
  - ✓ Ex : Requirements = other.GlueCEInfoLRMSType == "PBS" **&&** other.GlueCEInfoTotalCPUs > 2; (Les ressources doivent utiliser PBS comme LRMS et les WNs qui ont plus de 2 CPUs)

Langage de description des jobs (JDL)

JDL : Les attributs de ressources; Rank – 3/4

- C'est une préférence, concernant l'ordre de rangement des ressources qui remplissent les « Requirements »
- Le CE qui est sectionné et celui qui a la plus grande valeur de Rank
- Utilisent les attributs GLUE des ressources publiés dans le système d'information IS
- S'ils ne sont pas définis dans le jdl, ce sont les valeurs définies par défaut dans l'UI qui sont utilisées
  - ✓ Default: **other.GlueCEStateFreeCPUs** (le plus grand nombre de CPUs libres)
  - ✓ Default: **other.GlueCEStateEstimatedResponseTime** (le plus petit temps de réponse estimé)
- Syntaxe : Rank = < Nombre réel >
  - ✓ Exprimer en nombre réel
  - ✓ Ex : Rank = other.GlueCEPolicyMaxRunningJobs – other.GlueCEStateRunningJobs;

Langage de description des jobs (JDL)

JDL : Les attributs de ressources; Attributs de données – 4/4

- L'attribut **InputData** (optionnel)
  - ✓ Référant aux données utilisées en entrée par le job, ces données sont publiées dans le Replica Catalog et stockées dans les SEs
  - ✓ LFNs et/ou GUIDs
- L'attribut **DataAccessProtocol** (seulement si InputData est spécifié)
  - ✓ Le protocole ou la liste des protocoles de communication utilisables par l'application pour accéder aux InputData
- L'attribut **OutputData** (optionnel)
  - ✓ Référant aux données de sortie qui seront récupérables
  - ✓ RB l'utilise pour choisir le CE compatible pour le job et avec le SE

Langage de description des jobs (JDL)

JDL : Attribut pour re-soumission de job

- L'attribut RetryCount est une valeur entière qui représente le nombre maximum de re-soumissions de job à l'exécution en cas de résultats avec erreurs dues aux problèmes des composants de la grille.
  Ex : RetryCount = 3;
- Nombre maximum de re-soumissions: min(RetryCount, MaxRetryCount)
  RetryCount: Attribut du jdl
  MaxRetryCount: Attribute dans le fichier de configuration du "RB"
- Pour désactiver ce mécanisme il faut juste mettre RetryCount = 0.

Langage de description des jobs (JDL)

JDL : Commande de soumission de job

- **glite-wms-job-submit** [**–r** <res_id>] [**-c** <config file>] [**-o** <output file>] [**--vo** <VO name>] <job.jdl>
  - ✓ **-r** Le job est directement envoyé par le RB sur le CE identifié par <res_id>
  - ✓ **-c** Utilise le fichier de configuration <config file> afin de surcharger les valeurs par défaut de l'UI
  - ✓ **-o** Renvoi l'identifiant de job dans <output file>
  - ✓ **--vo** L'organisation virtuelle sous laquelle le job doit être exécuté

Langage de description des jobs (JDL)

JDL : Autres commandes relatives au job

- **glite-wms-job-list-match**
  - ✓ Liste les ressources correspondantes à la description du job
  - ✓ Permet de connaître le résultat de l'ordonnancement sans soumettre le job.
- **glite-wms-job-cancel**
  - ✓ Annule le job
- **glite-wms-job-status**
  - ✓ Affiche l'état du job.
- **glite-wms-job-get-output**
  - ✓ Récupère le contenu de OutputSandbox
- **glite-wms-job-logging-info**
  - ✓ Affiche des informations sur les différents états pris par le job tout au long de son existence.
  - ✓ Utilisé essentiellement pour le debuging.

Langage de description des jobs (JDL)

JDL : Soumission de jobs – 1/3

- Trois familles de scénarios sont possibles.
  - ✓ En utilisant le Ressources Broker, c'est à dire en excluant le cas ou l'on soumet le job directement au job manager d'un site.

Scénario 1: Soumission directe

- Le job est directement soumis au CE (spécifié par le paramètre **–r** de la commande ***glite-job-submit***).
- Le RB n'effectue aucune recherche de ressources.
- Peut (et généralement c'est le cas) générer des erreurs si on utilise l'attribut *InputData*.
- L'utilisateur est responsable de la cohérence de son job.

Langage de description des jobs (JDL)

JDL : Soumission de jobs – 2/3

Scénario 2: Soumission de job sans *Requirements* liés aux données

- Aucun CE ni données entrantes (*InputData*) sont précisés
- Le RB utilise l'algorithme de recherche des ressources qui comporte deux phases:
  - Le RB contact le système d'information afin de déterminer quels CEs peuvent satisfaire les demandes.
  - Si plus de deux sont proposés alors on utilise l'attribut *rank* pour faire le choix.

Langage de description des jobs (JDL)

JDL : Soumission de jobs – 3/3

Scénario 3: Le CE n'est pas spécifié et on a des données en entrée

- Le RB contact le service de management des données afin de déterminer quels SEs peuvent satisfaire les besoins (quels SEs possèdent les données requises)
- Le RB cherche le meilleur compromis (best effort) entre :
  - Les CEs où l'utilisateur a le droit de soumettre ces jobs.
  - Les SEs qui ont été déterminé préalablement.
- La stratégie du RB est de soumettre les jobs au plus près des données.
- Les deux phases suivantes sont les mêmes que dans le scénario précédent (uniquement pour les CEs qui satisfont les *Requirements* de données)
  - *Requirements* check
  - *Rank* computation

Langage de description des jobs (JDL)

JDL : Comment faire un jdl ?

- **Version minimale d'un fichier jdl (hello.jdl)**

```
################  Hello Job ###############
Executable = "/bin/echo Hello";
StdOutput = "hello.out";
StdError = "hello.err";
OutputSandbox = {"hello.out","hello.err"};
#####################################
```

- **On spécifie ici :**
  - ✓ Le programme (l'exécutable) et ses arguments (s'ils existent)
  - ✓ On définie les StdOutput et StdError
  - ✓ On dit que faire des outputs (les fichiers qui doivent être transférés de UI à WN et vis versa)

Langage de description des jobs (JDL)

Exercices : écrire un fichier jdl ?

- **ls.jdl : qui permet d'afficher la liste des fichiers de WN en exécutant la commande "ls –al" et en utilisant l'attribut *Argumemts***
  - ✓ **Sans pré requis,**
  - ✓ **Avec 4 tentatives,**
  - ✓ **avec pré requis de ressources :**
    CEId == "ce1.cnrst.magrid.ma:2119/jobmanager-lcgpbs-eumed"
- **simple.jdl: avec un fichier d'entrée " simple.sh " qui est un script qui affiche le nom du WN exécutant la commande "hostname –f " et la date d'exécution "date"**
  - ✓ **Sans pré requis,**
  - ✓ **Avec pré requis de ressources :**
    OpSys == " ScientificSL 4.5 "
  - ✓ **Avec une préférence de :**
    "FreeCPUs"

Langage de description des jobs (JDL)

Solution : ls.jd

- ✓ **Sans pré requis,**

```
Executable = "ls";
Arguments = "-al";
StdOutput = "ls.out";
StdError = "ls.err";
OutputSandbox = {"ls.out", "ls.err"};
```

- ✓ **Avec 4 tentative,**

```
RetryCount = 4;
```

- ✓ **Avec pré requis de ressources :**

```
Requirements = other.CEId == "ce1.cnrst.magrid.ma:2119/jobmanager-lcgpbs-eumed";
```

Langage de description des jobs (JDL)

Solution : simple.jdl

- ✓ **simple.sh:**

```
#!/bin/sh
echo "First JDL simple this is simple.sh running at "
hostname -f
date
```

- ✓ **Sans pré requis,**

```
Executable = “simple.sh";
StdOutput = "simple.out";
StdError = "simple.err";
InputSandbox = {"simple.sh"};
OutputSandbox = {"simple.out", "simple.err"};
```

- ✓ **Avec pré requis de ressources , et préférence :**

```
Requirements = other.SysOp == "SientificSL 4.5";
Rank = other.FreeCPUs;
```

Langage de description des jobs (JDL)

JDL : Message Passing Interface - MPI job – 1/2

- Il y a beaucoup de bibliothèques qui supportent les jobs parallèles, nous décidons de supporter uniquement MPICH
- Le job MPI s'exécute en parallèle et sur différents processeurs
- L'utilisateur doit affecter à l'attribut JobType la valeur MPICH et spécifier l'attribut NodeNumber qui requis le nombre de CPUs
- Quand le job MPI est soumis, UI ajoute:
  - ✓ Dans l'attribut *Requirements* :
    - **Member("MpiCH", other.GlueHostApplicationSoftwareRunTimeEnvironment)** (l'environnement runtime de MPICH doit être installé dans le CE)
    - **other.GlueCEInfoTotalCPUs >= NodeNumber** (le nombre de CPUs doit être au minimum égale à celui requis dans l'attribut *NodeNumber*)
  - ✓ Dans l'attribut *Rank* :
    - **other.GlueCEStateFreeCPUs** (CE avec maximum de CPUs libres)

Langage de description des jobs (JDL)

JDL : Message Passing Interface - MPI job – 2/2

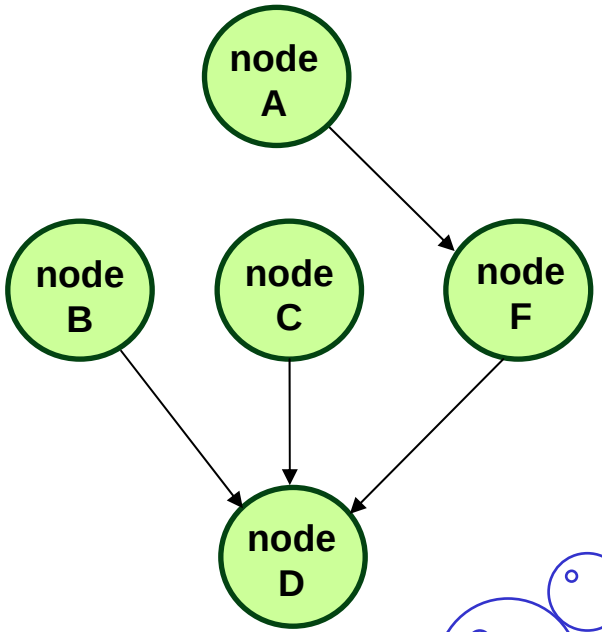
- Les jobs parallèles peuvent être exécutés uniquement dans le même site (cluster ou CE)
- Le fichier source doit être compilé avec les bibliothèques mpicc
- Exemple d'un job MPI :

```
Type = "Job";
JobType = "MPICH";
NodeNumber = 2;
Executable = "cpi";
StdOutput = "cpi.out";
StdError = "cpi.err";
InputSandbox = {"cpi"};
OutputSandbox = {"cpi.err","cpi.out"};
RetryCount = 3;
```

Langage de description des jobs (JDL)

JDL : Directed Acyclic Graph - DAG job – 1/2

- Le job DAG est un ensemble de jobs de tel façon que l'input, output ou l'exécution d'un ou plusieurs jobs peuvent dépendre des autres jobs
- Les dépendances sont représentées à travers le DAG, où les nœuds sont des jobs, et les liaisons représentent les dépendances
- Les sous jobs sont sélectionnés uniquement si le nœud DAG est prêt
- L'utilisateur doit mettre l'attribut Type à **dag**, l'attribut nodes doit contenir la description des nœuds et l'attribut dependencies doit définir les dépendances.



Langage de description des jobs (JDL)

JDL : Directed Acyclic Graph - DAG job – 2/2

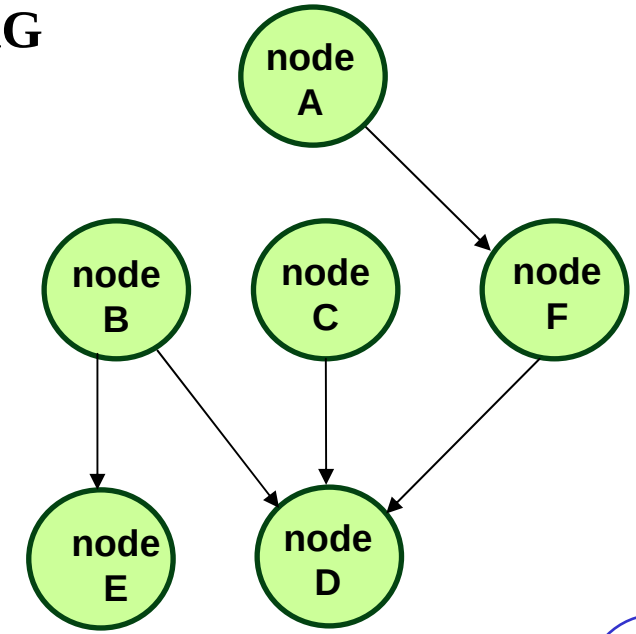
- En utilisant **file** pour définir le fichier jdl ou son chemin relatif, ou bien on peut définir le jdl nœud dans un block de l'attribut **description**
- Pas d'attribut **OutputSandbox**

```
Type = "dag";
nodes = [ nodeA = [
            file = "nodeA.jdl" ;
          ];
          nodeB = [
           file = "nodeB.jdl" ;
          ];
           .....
          nodeF = [
           file = "nodeF.jdl" ;
          ];
         dependencies = {{nodeA, nodeF},{{nodeF, nodeB, nodeC}, nodeD}}; ];
```

Langage de description des jobs (JDL)

Exercice :

- **Monter les dépendances du graph ?**
  - ✓ **Avec deux formes différentes**
- **NodeA : décrire les mêmes attributs correspondant au fichier jdl 'ls.jdl' et les autres nœuds sont des fichiers jdl (NodeX.jdl)**
- **Ecrire le fichier correspondant au DAG**



Langage de description des jobs (JDL)

Solution :

✓ **Dépendances:**

```
1- dependencies = {{nodeA, nodeF},{{nodeF, nodeB, nodeC}, nodeD},{nodeB,{nodeD, nodeE}};
2- dependencies = {{nodeA, nodeF}, {nodeF, nodeD}, {nodeB, nodeD}, {nodeC, nodeD},
{nodeB, nodeE}};
```

✓ **Description des nœuds:**

```
nodeA = [    description =
                              [ Executable = "ls";
                                Arguments = "-al";
                                StdOutput = "ls.out";
                                StdError = "ls.err";
                                OutputSandbox = {"ls.out", "ls.err"};  ];
          ];
nodeB = [
              file = "nodeB.jdl" ;
          ];
.....];
```

Langage de description des jobs (JDL)

JDL : Collection job – 1/2

- Le job collection est un ensemble de jobs indépendants que l'utilisateur veut soumettre et superviser comme une simple requête de soumission.
- La collection des jobs est soumise comme le job DAG sans dépendances

```
[
        Type = "collection";
        VirtualOrganisation = "eumed";
        nodes = {
                [  <job descr 1 >],
                [  <job descr 2 >],
                 ...
               };
]
```

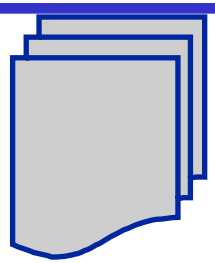
Langage de description des jobs (JDL)

JDL : Collection job – 2/2

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

JDL: Example de soumission de job

Replica Catalogue

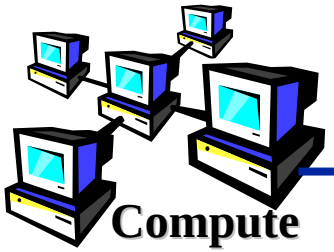


Information Service

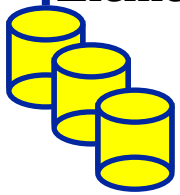
Resource Broker

Logging & Book-keeping

Job Submission Service



Compute Element

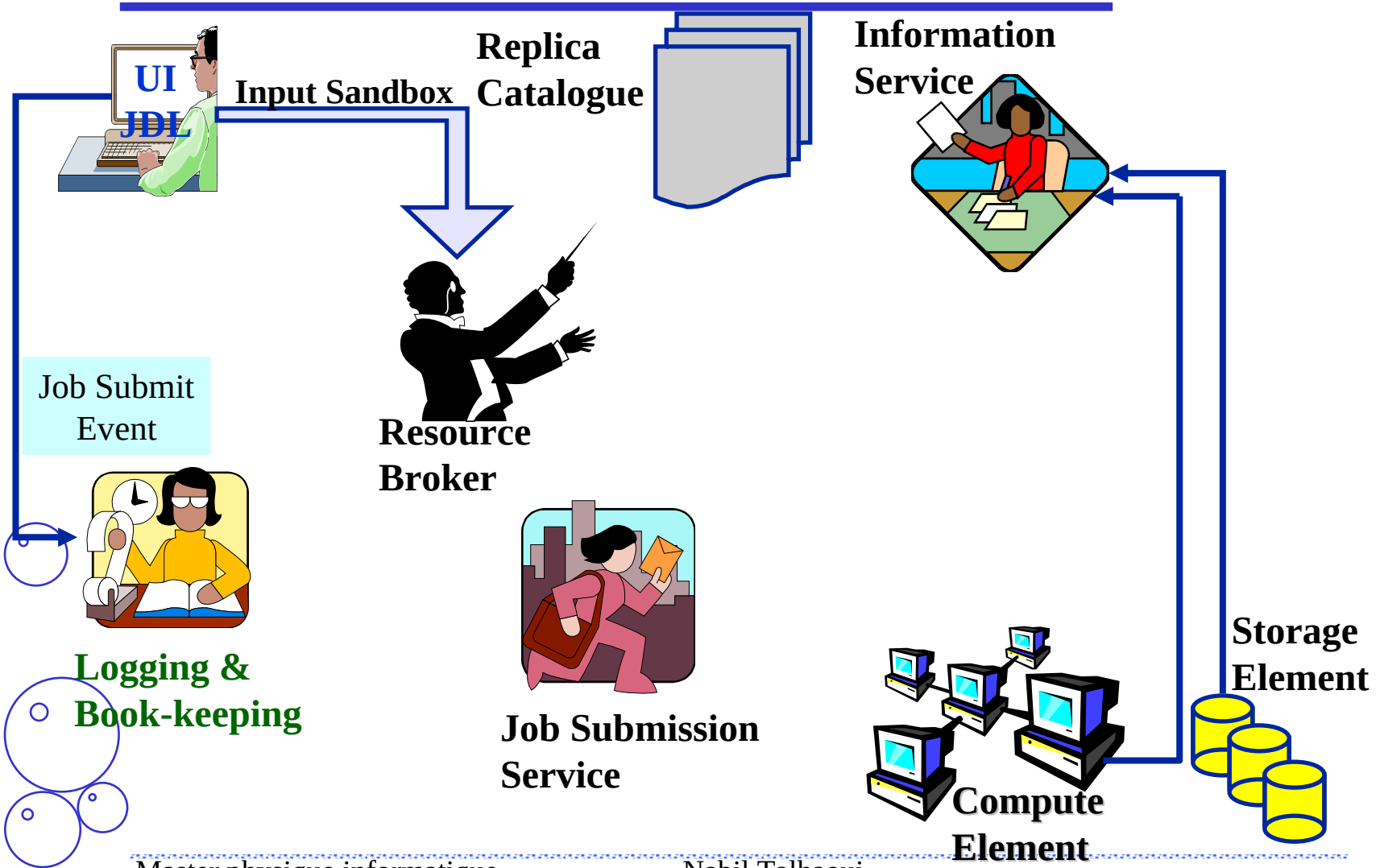


Storage Element

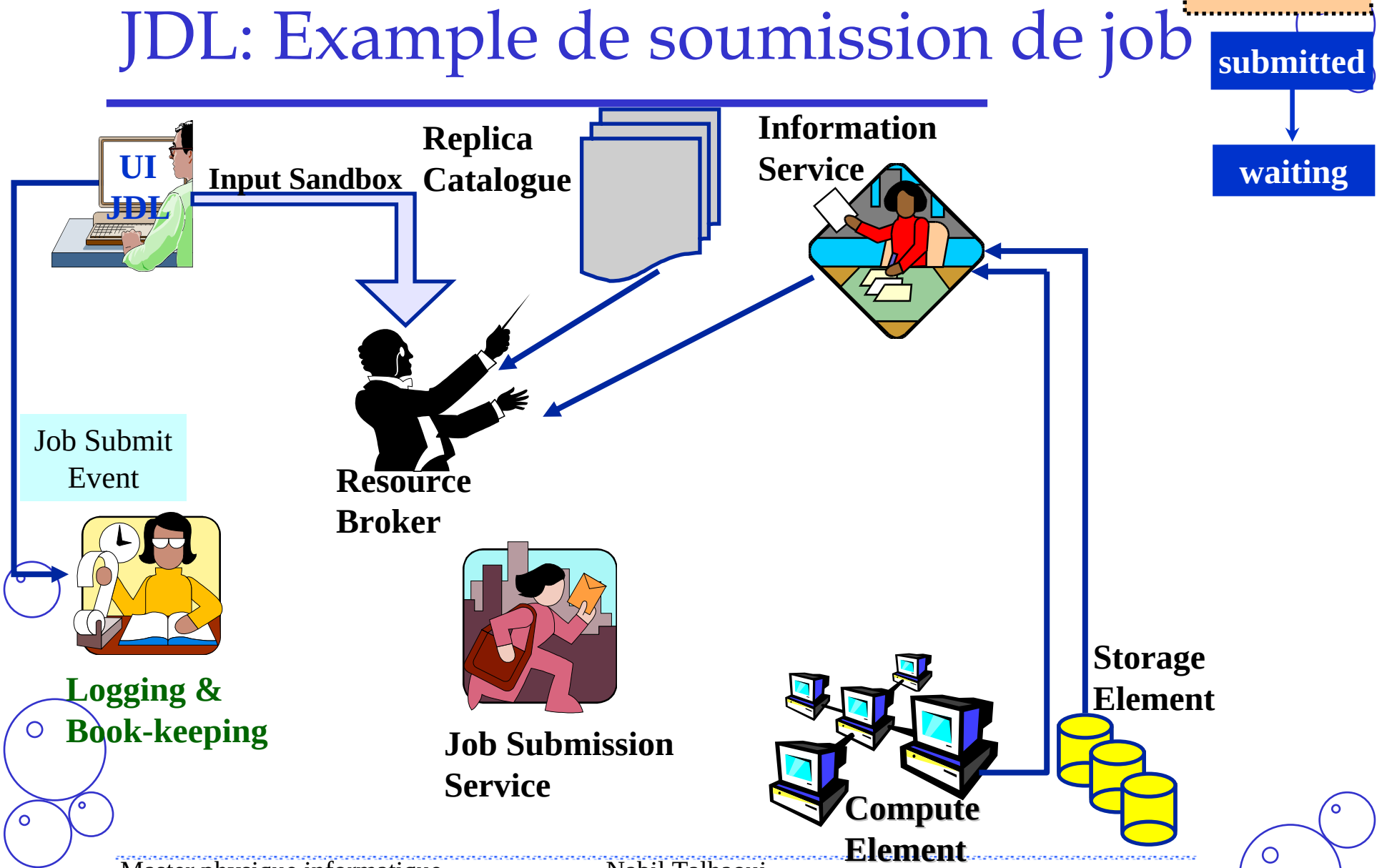
Job Status

submitted

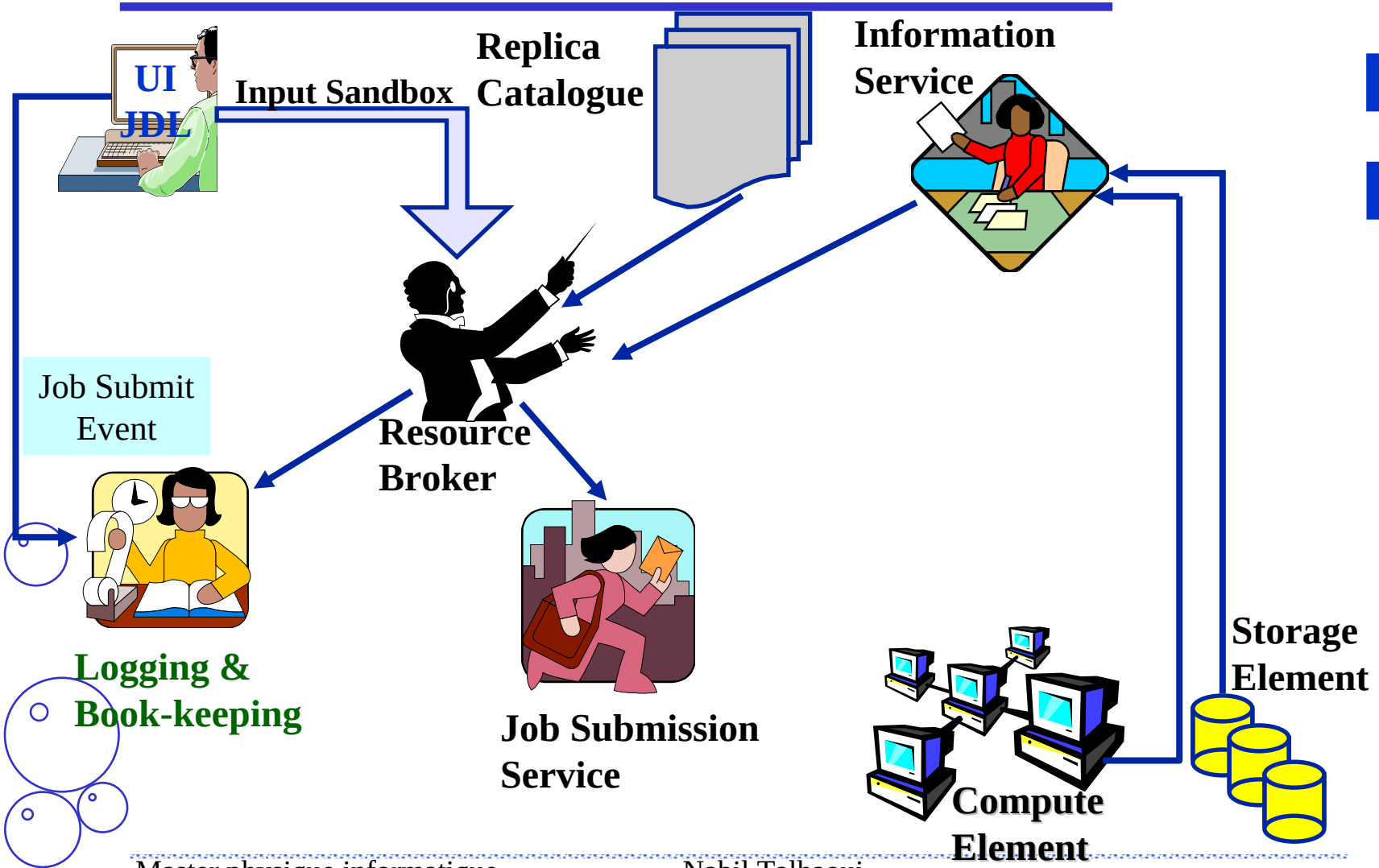
JDL: Example de soumission de job



JDL: Example de soumission de job



JDL: Example de soumission de job



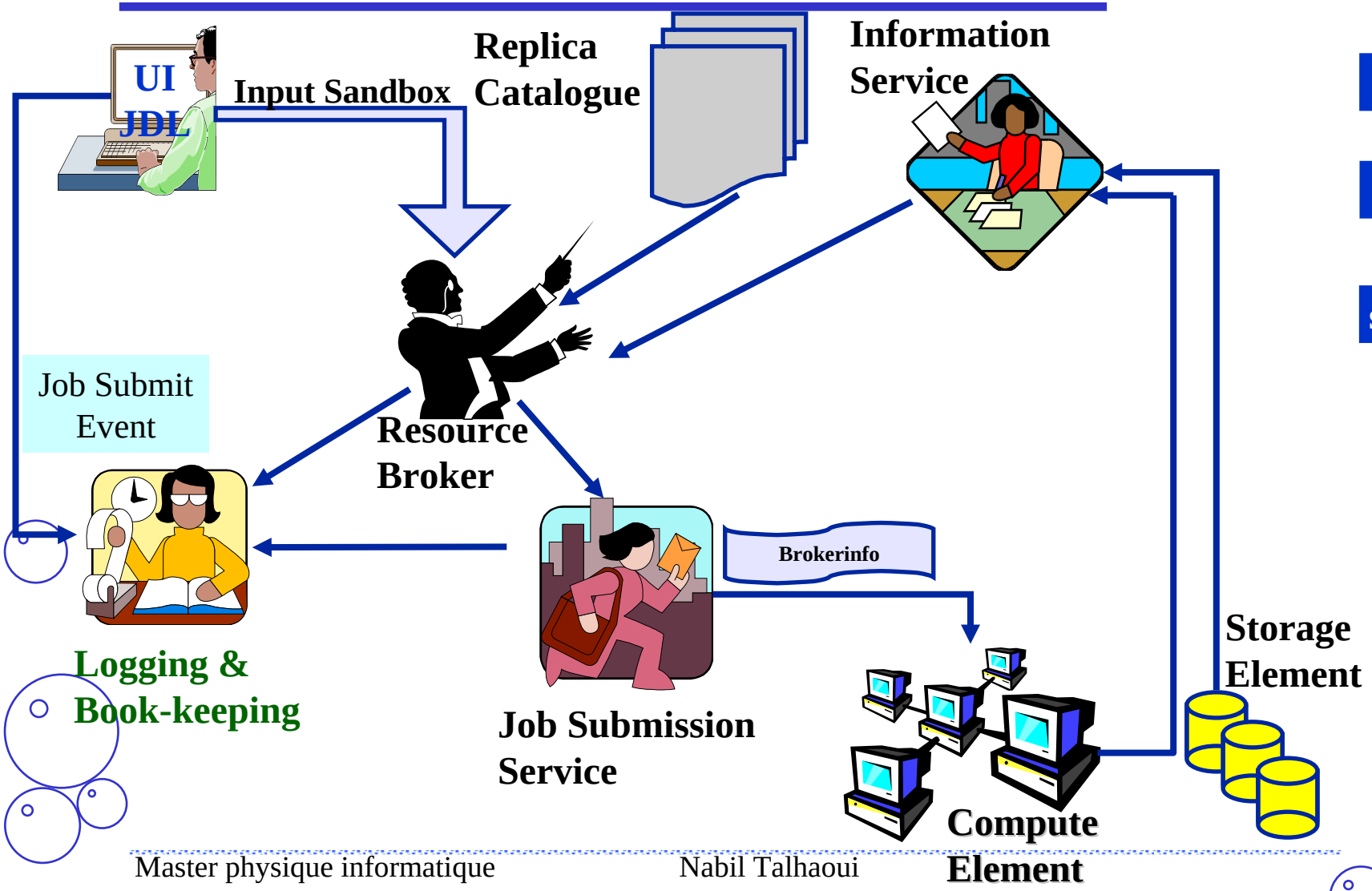
Job Status

submitted

waiting

ready

JDL: Example de soumission de job



Job Status

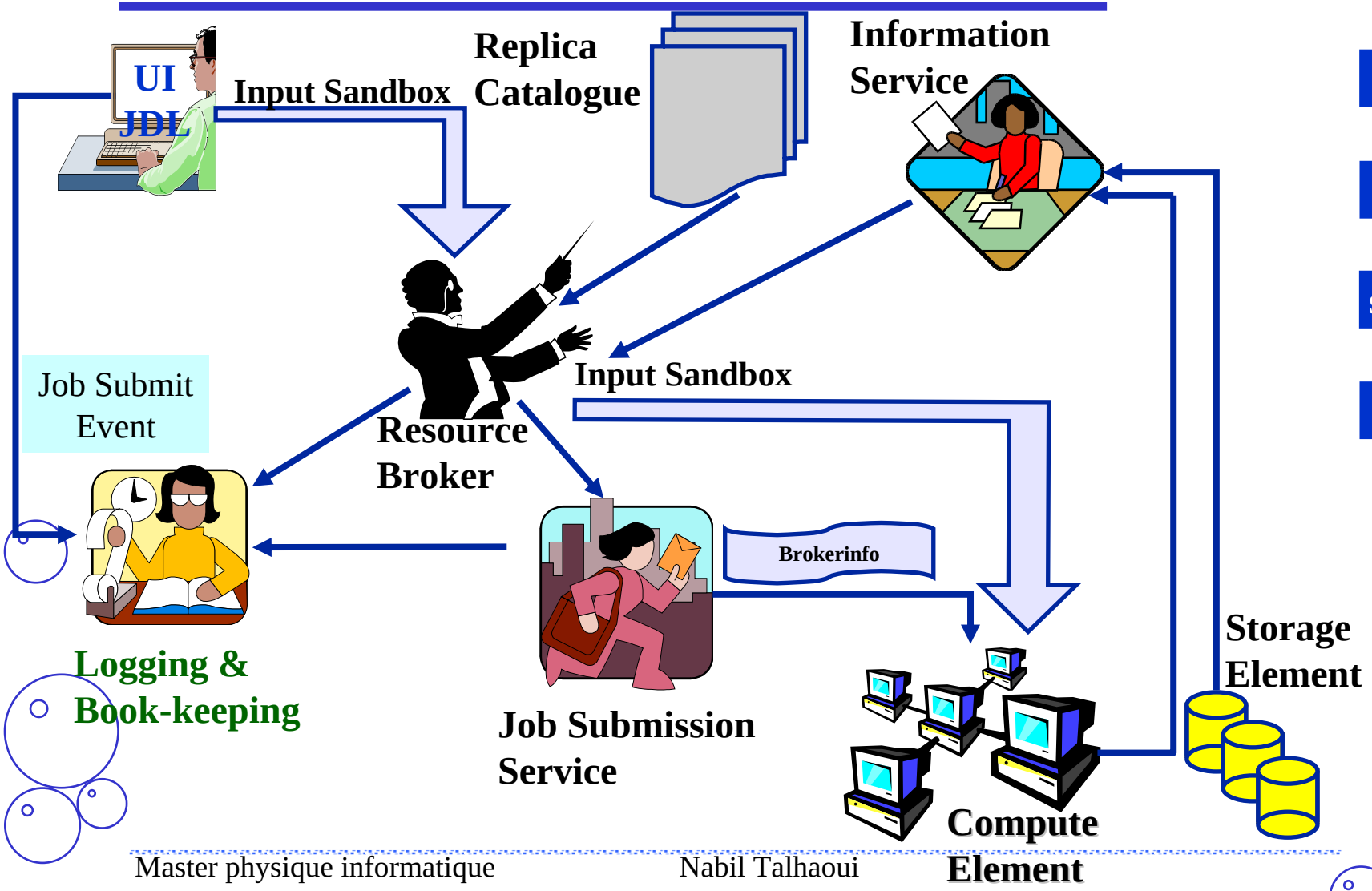
submitted

waiting

ready

scheduled

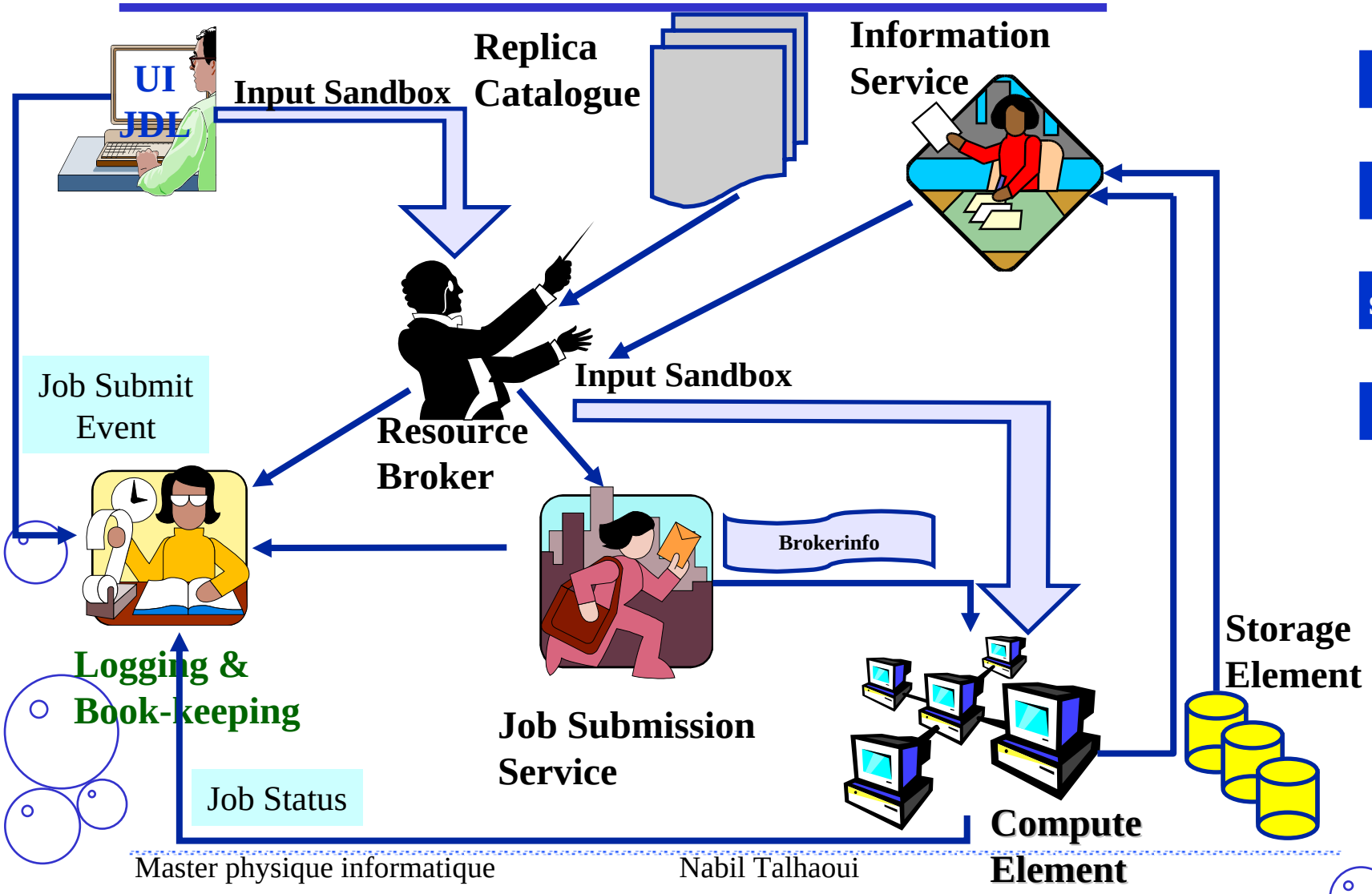
JDL: Example de soumission de job



Job Status

- submitted
- waiting
- ready
- scheduled
- running

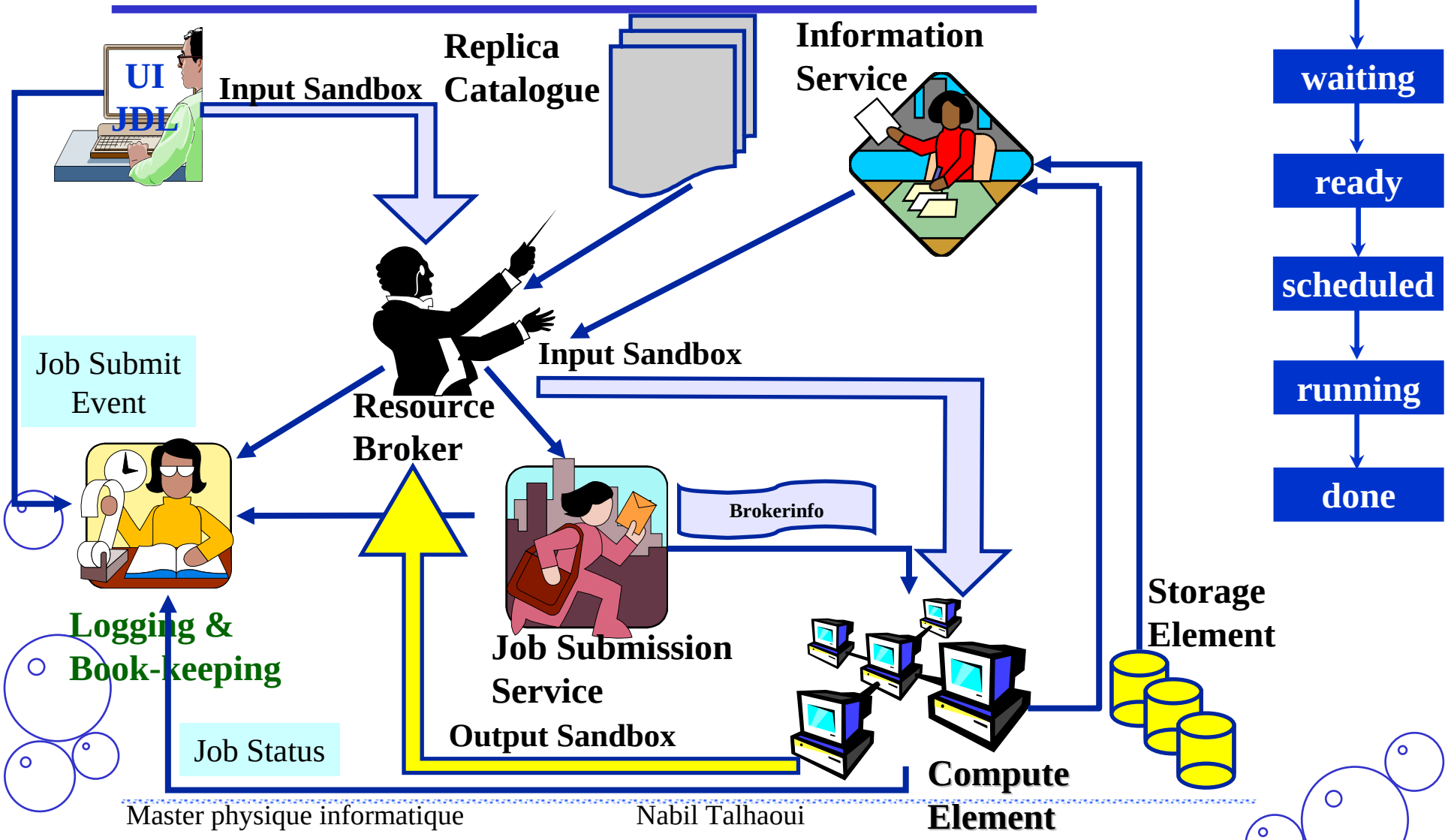
JDL: Example de soumission de job



Job Status

- submitted
- waiting
- ready
- scheduled
- running

JDL: Example de soumission de job



JDL: Example de soumission de job