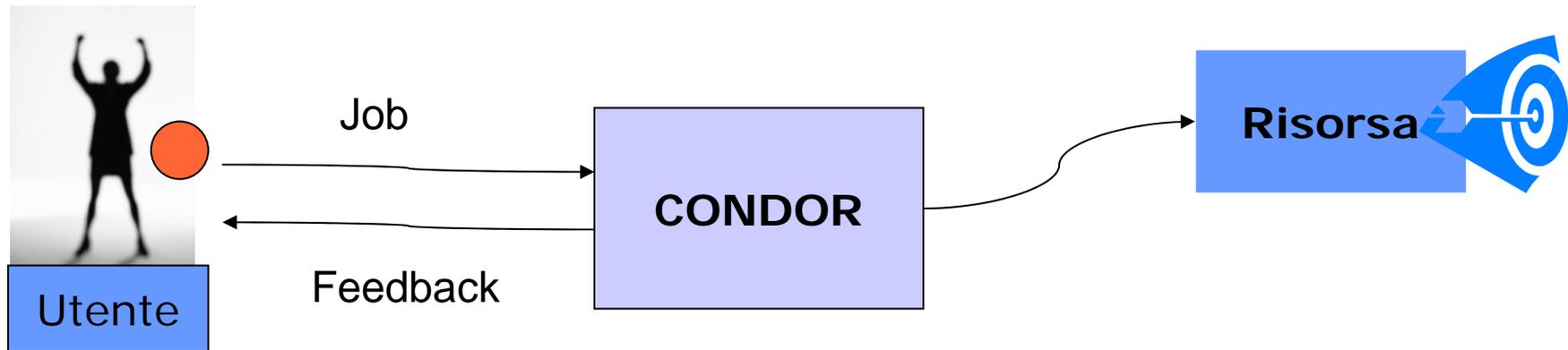


# Condor, Condor-G e la Griglia

# Cosa Fa Condor

---



Meccanismi di job management, politica di scheduling, schema con priorità, monitoring di risorse,.....

# Filosofia di Flessibilità

---

- Lasciare il controllo al proprietario
  - ◆ Politiche di Uso
  - ◆ Decidere quando la risorsa potrà essere usata
  - ◆ Proprietari felici -> più risorse -> maggior throughput
- Lasciar crescere naturalmente le comunità
  - ◆ Cambi di requisiti e relazioni
  - ◆ Contratti non precisi
- Pianificare senza essere prepotenti
  - ◆ Non assumere un funzionamento corretto

# Condor: Un Sistema per High Throughput Computing

---

- Obiettivo
  - ◆ Grandi quantità di potenza di elaborazione fault tolerant
  - ◆ Utilizzazione effettiva di risorse
- Da ottenere tramite “opportunistic computing”
  - ◆ Usa le risorse quando sono disponibili
  - ◆ *ClassAds* – per descrivere risorse e jobs
  - ◆ Job checkpoint e job migration
  - ◆ Remote system calls – preserva l’ambiente di esecuzione locale

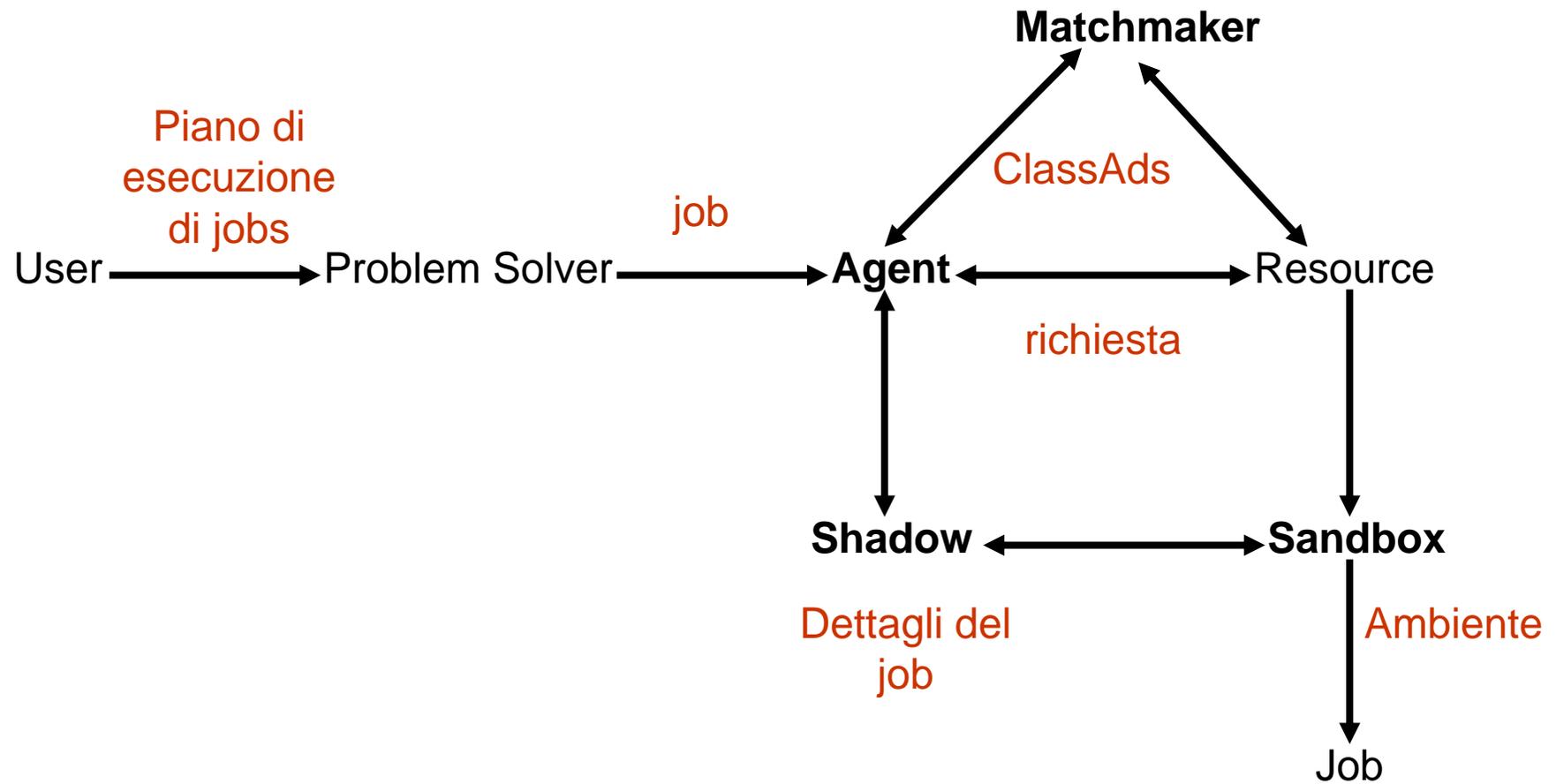
# Condor-G: Agente di gestione Computazionale per Grid Computing

---

- Combinazione di tecnologia Globus e Condor
- Globus
  - ◆ Protocolli per comunicazioni sicure tra domini
  - ◆ Accesso standard a sistemi batch remoti
- Condor
  - ◆ Job submission e allocation
  - ◆ Error recovery
  - ◆ Creazione di un ambiente di esecuzione

# Condor Kernel

---



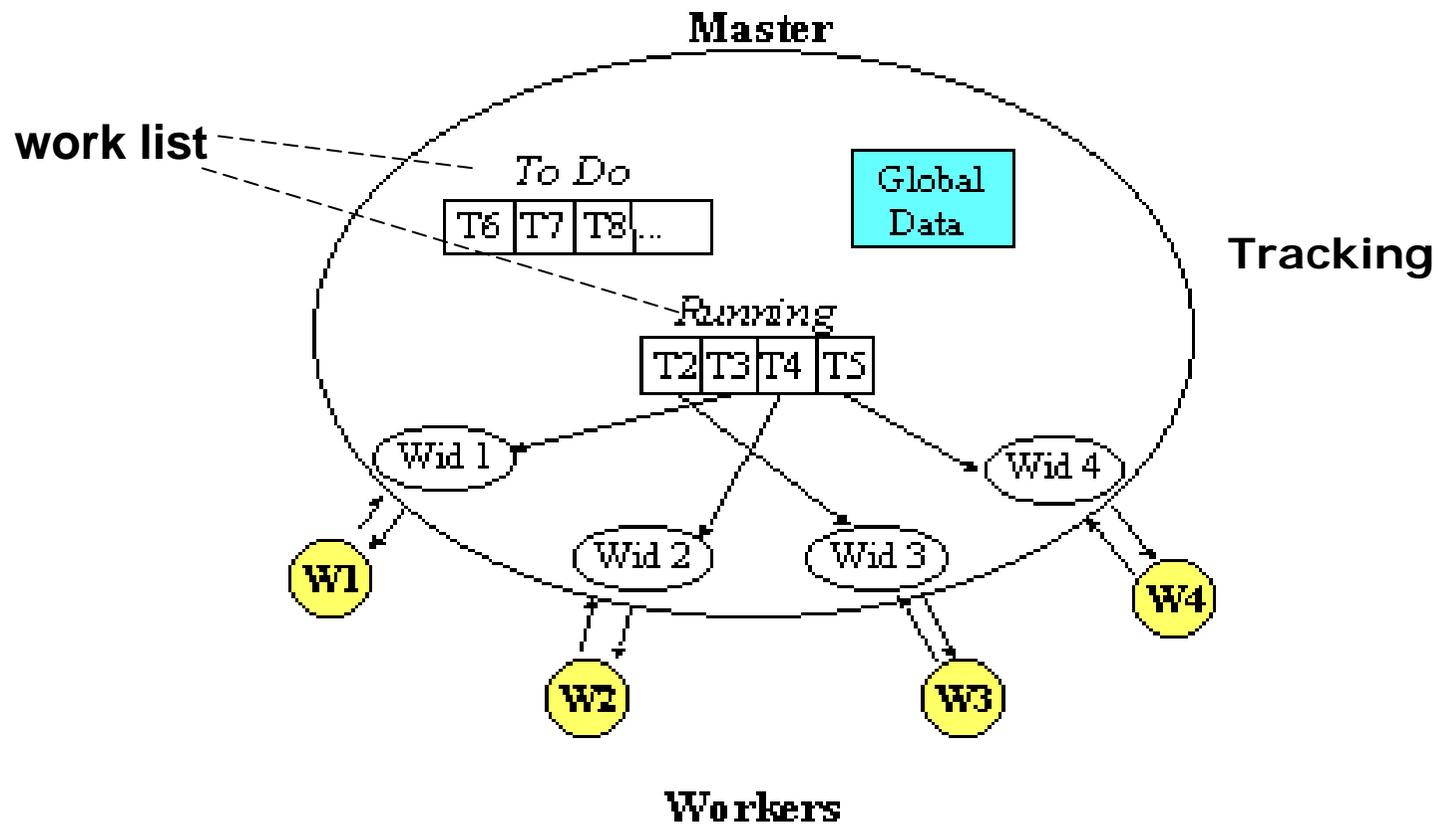
# Problem Solver

---

- Struttura di alto livello costruito 'sopra' un Condor agent
- Si occupa dell'ordinamento dei job e della selezione dei task
- E' esso stesso rappresentato come un job
  - ◆ Un job che sottomette jobs
- Dipende dall'agent per l'esecuzione dei task
- *Master-Worker e DAG Manager*

# Master-Worker

---



T2 in esecuzione su W3 e monitorato da Wid 3

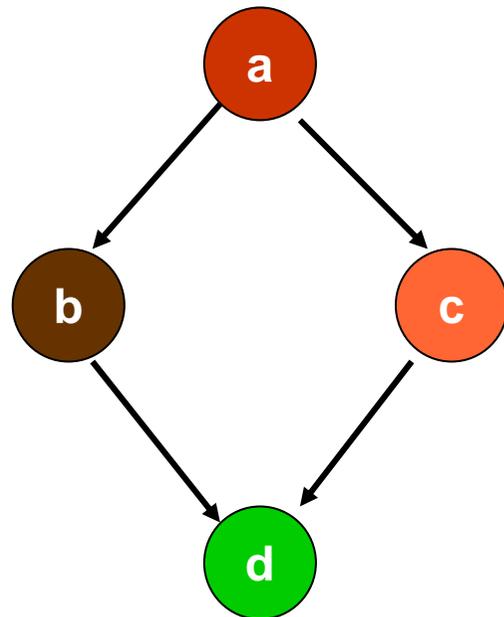
# Directed Acyclic Graph Manager

---

- Esegue più jobs con dipendenze
- Dipendenze dichiarate usando istruzioni *Parent-Child*
- Programmi speciali da eseguire prima o dopo un job sono specificati da comandi *Pre* e *Post*
  - ◆ Per il setup dell'ambiente di esecuzione e l'analisi dei risultati
- L'utente può specificare che un job 'fallito' può essere rieseguito con in comando *Retry*

# Directed Acyclic Graph Manager

---



**Job a**

**Job b**

**Job c**

**Job d**

**Parent a child b c**

**Parent b child d**

**Parent c child d**

**Script Pre c in.pl**

**Retry c 3**

# Split Execution

---

- L'esecuzione di Job richiede
  - ◆ Informazione che specifica il job
  - ◆ Tool come memoria, rete, ecc.

Devono essere nello stesso sito
- Shadow
  - ◆ Ha informazione che specifica il job
    - Eseguibili, argomenti, file di input, ....
- Sandbox
  - ◆ Crea un ambiente per l'esecuzione di un job

# Universi

---

- Sandbox + Shadow
- Universo Standard
  - ◆ Sandbox crea una directory temporanea e la usa per i dettagli sul job in esecuzione
  - ◆ Shadow fornisce accesso remoto a device di memoria dell'utente
  - ◆ Es. Job richiede un file, la richiesta va allo shadow e il file è memorizzato nel sito di esecuzione.
- Universo Java

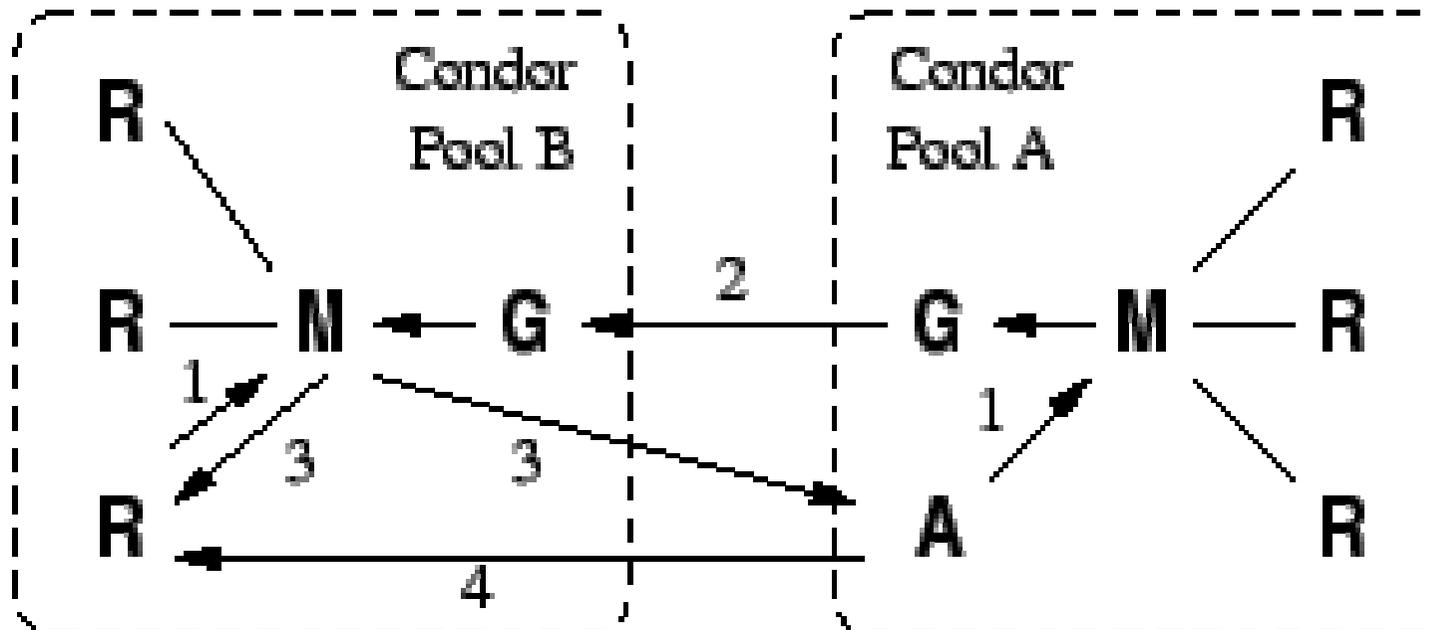
# Politiche

---

- Agente
  - ◆ Quale risorsa è affidabile?
  - ◆ Risorse utili per eseguire jobs
- Risorsa
  - ◆ Di quale utente fidarsi?
- Matchmaker
  - ◆ Politiche di Comunità, controllo di accesso
- Comunità definite dal matchmaker
  - ◆ Agenti possono usare una risorsa solo se condividono un Matchmaker

# Gateway Flocking

---



I Gateway passano informazione sui partecipanti tra pool, MA invia richiesta a MB attraverso un gateway G, MB ritorna un *match*

# Gateway Flocking

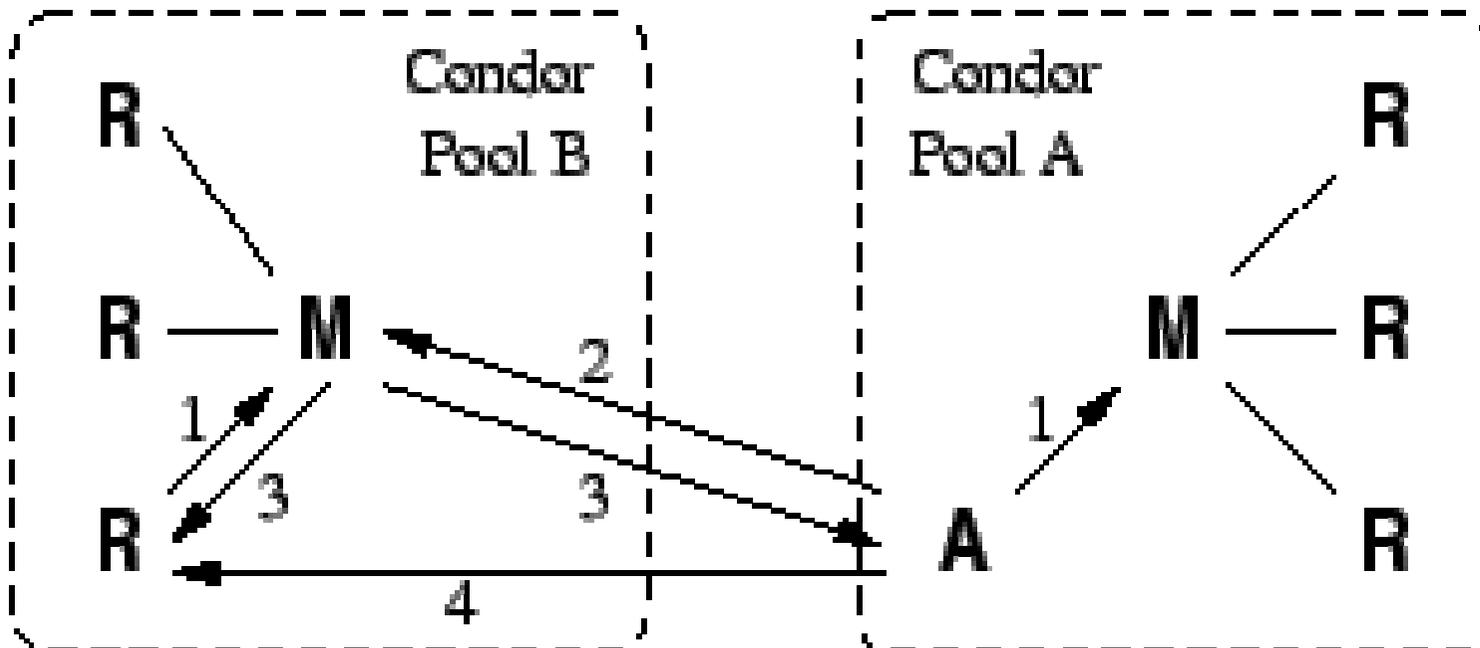
---

- La struttura dei pool è preservata
- Completamente trasparente : nessuna modifica per gli utenti
- Condivisione a livello organizzativo
- Tecnicamente complesso : i gateway partecipano in tutte le interazioni nel kernel Condor

Soluzione: Direct Flocking

# Direct Flocking

---

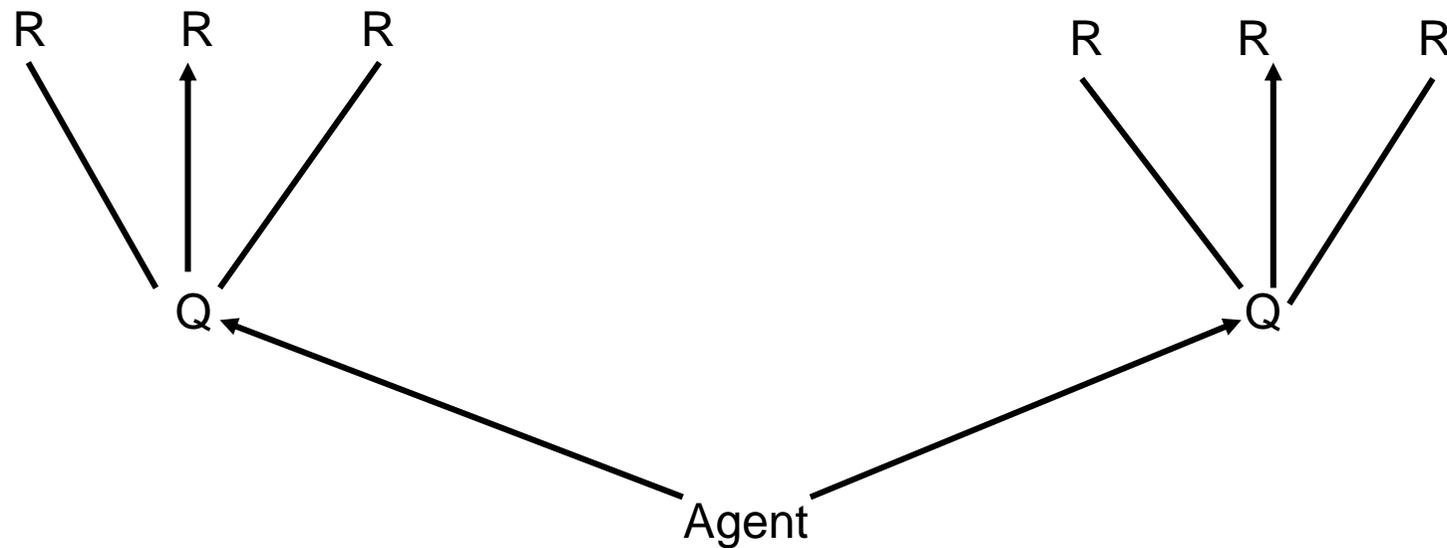


A interagisce anche con il Condor Pool B

# Condor-G

---

- Agenti Condor che comunicano con il GRAM



Jobs vengono accodati e non direttamente assegnati alle risorse

# Direct Flocking vs Condor-G

---

- In Condor un Agente sottomette job ad una risorsa, in Condor-G il job è sottomesso ad una coda.

Agenti in Condor-G possono

- Over subscribe

Sottomettere un job a più code, attendere una risposta e quindi cancellare gli altri job

- Under subscribe

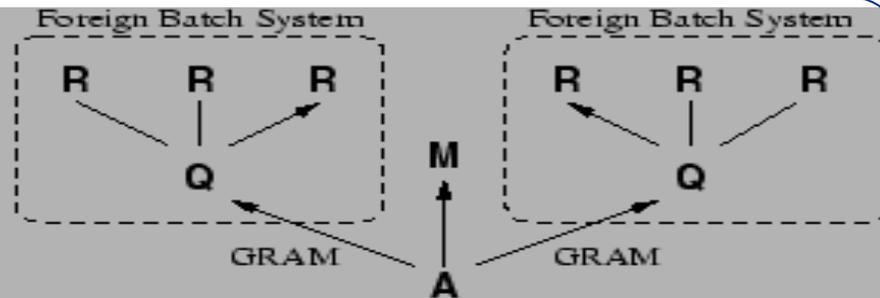
Sottomettere un job a una coda che può essere lunga

- Il GRAM permette l'accesso ad una varietà di sistemi batch, con accodamento ed esecuzione remota

# Condor Pool Personali

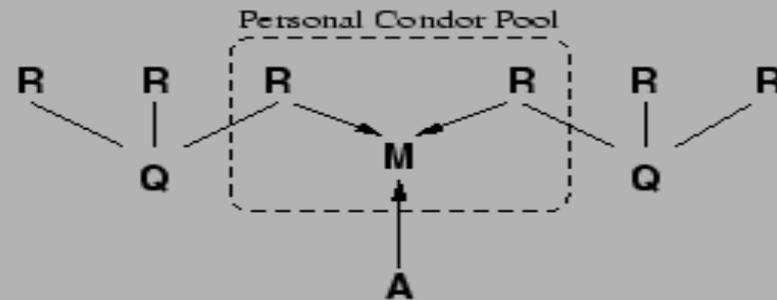
## Step 1:

L'utente sottomette dei daemon Condor come job su sistemi remoti



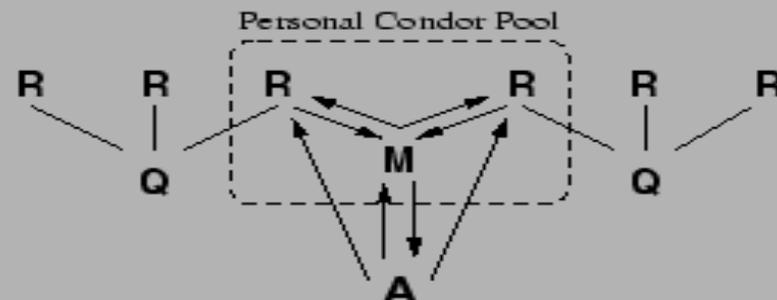
## Step 2:

I daemon Condor forma un Condor pool ad hoc



## Step 3:

L'utente esegue job sul Condor pool così costituito



# Matchmaker: Un Ponte tra Planning e Scheduling

---

- Agenti e risorse pubblicano caratteristiche e requisiti come **ClassAds**
- Sono create coppie che soddisfano i vincoli
- Ambedue le parti sono informate

# ClassAds

---

- Coppie di Attributi nome-valore
- Senza schema specifico
- Logica a tre valori
  - ◆ True, false e undefined
- Requisiti
  - ◆ Vincoli, per un match devono essere *true*
- Rank
  - ◆ Desiderabilità di un match

# ClassAds

---

## Job ClassAd

```
[  
MyType = "Job"  
TargetType = "Machine"  
Requirements =  
((other.Arch=="INTEL"&&  
other.OpSys=="LINUX")  
&& other.Disk > my.DiskUsage)  
Rank = (Memory * 10000) + KFlops  
Cmd = "/home-exe"  
Department = "CompSci"  
Owner = "tannenba"  
DiskUsage = 6000  
]
```

## Machine ClassAd

```
[  
MyType="Machine"  
TargetType="Job"  
Machine="tnt.isi.edu"  
Requirements=  
(Load<3000)  
Rank=dept==self.dept  
Arch="Intel"  
OpSys="Linux"  
Disk=600000  
]
```

# Estensioni al matchmaking

---

- **Gang matching**

- ◆ Co-allocazione di più di una risorsa

- **Collections**

- ◆ Memorie persistenti di ClassAds con tecniche proprie dei database come indexing

- **Set matching**

- ◆ Matching di un alto numero di risorse usando una espressione compatta

- **Indirect references**

- ◆ Per permettere ad un ClassAd di far riferimento ad un altro

# Planning e Scheduling

---

- Planning
  - ◆ Acquisizione di risorse da parte degli utenti
  - ◆ E' relativo a 'cosa' e 'dove'
- Scheduling
  - ◆ Gestione di una risorsa da parte del proprietario
  - ◆ E' relativo a 'chi' e 'quando'
- Feedback tra planning e scheduling

# Planning e Scheduling

---

- Pianificare sulla base di uno schedule
  - ◆ Lo scheduler pubblica informazioni circa orari e priorità
  - ◆ Condor-G pianifica in questo modo
- Scheduling senza una pianificazione
  - ◆ Match dei risultati di una richiesta di risorsa
  - ◆ Un agente crea uno schedule per eseguire i task su quella risorsa

# Condor-G: Agente di gestione Computazionale per Grid Computing

---

- Combinazione di tecnologia Globus e Condor
- Globus
  - ◆ Protocolli per comunicazioni sicure tra domini
  - ◆ Accesso standard a sistemi batch remoti
- Condor
  - ◆ Job submission e allocation
  - ◆ Error recovery
  - ◆ Creazione di un ambiente di esecuzione

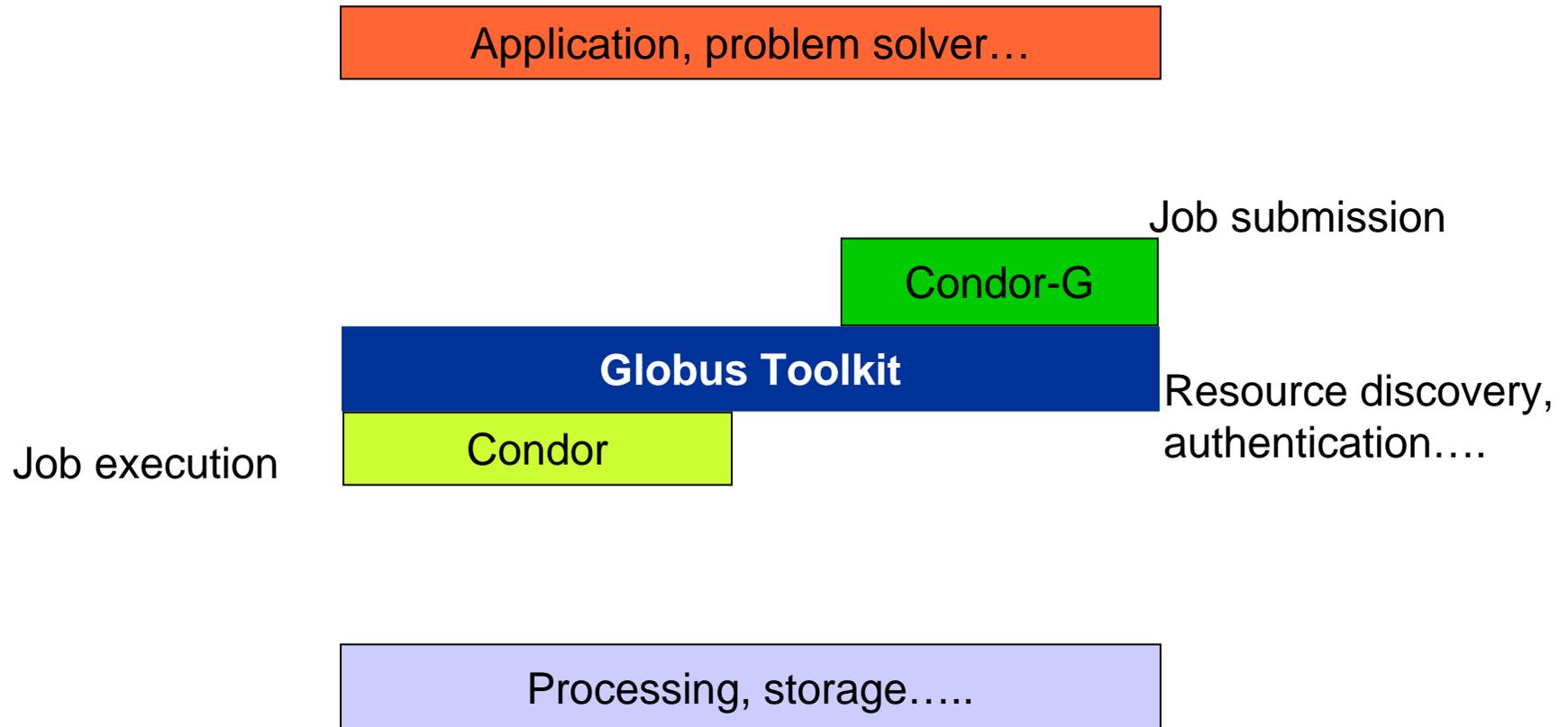
# Condor-G

---

- Sfrutta:
  - ◆ Security, comunicazioni, resource discovery, accesso a risorse in ambienti multi-dominio offerti dal Globus Toolkit
  - ◆ Gestione dell'elaborazione e raccolta di risorse in un singolo dominio amministrativo forniti da Condor
- Condor-G:
  - ◆ Permette agli utenti di integrare risorse appartenenti a più domini come se appartenessero ad un unico dominio personale.

# Tecnologie Condor in Middleware di Grid

---



# Approccio Condor-G

---

## 1. Accesso a Risorse Remote

- ◆ Richiede che le risorse remote usino protocolli standard per la gestione di risorse remote
- ◆ **Uso di protocolli definiti dal Globus Toolkit**

## 2. Gestione dell'elaborazione

- ◆ Introduce agenti per la gestione dei processi utente
- ◆ Responsabile per: resource discovery, job submission, job management, error recovery
- ◆ **Uso del sistema Condor**

## 3. Ambiente di esecuzione remota

- ◆ Uso della tecnologia di sandboxing per creare un ambiente di esecuzione su un nodo remoto
- ◆ **Uso del sistema Condor**

# Protocolli di Grid per l'Accesso a Risorse Remote

---

- GSI (Security)
- GRAM (sottomissione remota di richieste di elaborazione)
  - ◆ Two-phase commit (aggiunto dal Condor team)
    - Fornisce una semantica di esecuzione exactly-once
    - Le richieste di risorse dei client includono numeri di sequenza
    - Il client riceve risposta da una risorsa e invia un messaggio di commit per indicare che l'esecuzione può iniziare
  - ◆ Fault tolerance (aggiunto dal Condor team)
    - Il GRAM memorizza informazione sui job attivi in memoria stabile sul nodo client
    - Accede all'informazione se il GRAM server va in crash e riparte
- MDS-2
- GASS (Global Access to Secondary Storage)
  - ◆ Obsoleto, sostituito da GridFTP

# Gestione dell'Elaborazione: il Condor-G Agent

---

- Supporta l'esecuzione remota
- L'Agente esegue applicazioni su risorse di nodi remoti per conto di utenti
  - ◆ Gestisce lo standard I/O e gli eseguibili dei job usando il GridFTP
  - ◆ Sottomette un job su un nodo remoto usando il GRAM
  - ◆ Monitoring dei job dei fallimenti remoti via GRAM
  - ◆ Autenticazione di tutte le richieste usando il GSI
  - ◆ Riesegue job falliti
  - ◆ Comunica con l'utente in caso di errori
  - ◆ Memorizza lo stato della computazione su memoria stabile per supportare il restart in caso di fallimento di un agent

# Implementazione del Condor-G Agent

---

1. Lo **Scheduler** risponde alla richiesta di un utente
2. Crea un nuovo daemon **GridManager** per eseguire e gestire job
  - Un processo gestisce tutti i job di un singolo utente
  - Termina quando tutti i job sono completati
3. Ogni richiesta di esecuzione di job al **GridManager** crea un daemon Globus **JobManager**
  - Un JobManager per sottomissione di job
  - I JobManager comunicano con il GridManager per trasferire gli eseguibili di job e per l' I/O
4. Il **JobManager** sottomette i job per l'esecuzione allo scheduler locale del sito
5. Aggiornamenti sullo stato dei job sono inviati dal **JobManager** al **GridManager**

# Fallimenti Tollerati da Condor-G

---

## **Tollerante ai seguenti eventi:**

1. Crash del Globus JobManager
2. Crash della macchina che gestisce la risorsa remota (es., il GateKeeper e/o il JobManager)
3. Crash della macchina su cui è in esecuzione il GridManager (o crash del GridManager)
4. Fallimenti nei collegamenti di rete tra le macchine coinvolte

## **Failure detection (rilevazione dei guasti)**

- Rilevati dal GridManager, che periodicamente controlla tutti i JobManagers

# Recupero dai Fallimenti

---

- Se il JobManager è fallito
  - ◆ GridManager tenta di eseguire un nuovo JobManager
- Se non c'è contatto con il nodo remoto, il GridManager attende fino a poter ristabilire il contatto
  - ◆ Quindi tenta di connettersi al JobManager
    - Il JobManager può essere terminato normalmente
  - ◆ Se non riesce a connettersi al JobManager, crea un nuovo JobManager
    - Il JobManager controllerà i job o potrà comunicare al GridManager che i job sono terminati con successo
- Per proteggersi in caso di crash locale, lo stato dei job è memorizzato in modo persistente nella coda dei job dello scheduler Condor-G
  - ◆ GridManager riparte dopo un crash locale
  - ◆ Si riconnette a qualsiasi JobManager in esecuzione al momento del crash

# Condor-G Agent e Gestione delle Credenziali

---

- L'agente Condor-G usa Credenziali Proxy del GSI per autenticare l'utente presso le risorse remote
- Credenziali Short-lived
- Le applicazioni Condor-G con durate lunghe devono gestire la terminazione delle credenziali
- L'agente Condor-G periodicamente analizza le credenziali degli utenti con job in esecuzione
- Gli utenti sono informati della prossima scadenza delle credenziali
- Le credenziali vanno rinnovate su ogni sito coinvolto
- MyProxy: permette agli utenti di memorizzare credenziali a lunga scadenza su un server sicuro
  - ◆ Servizi remoti possono ottenere per conto dell'utente short-lived proxy da MyProxy
  - ◆ Condor-G rinnova le credenziali dal MyProxy server