# Un'introduzione a

Condor
High Throughput Computing

## II Parte

Condor Project
Computer Sciences Department
University of Wisconsin-Madison
condor-admin@cs.wisc.edu
http://www.cs.wisc.edu/condor

# Il nuovo problema di Frieda: Come possono i miei job avere accesso ai file di dati?

# Condor è flessibile

- file system condivisi: standard input, output, e error sono considerati accessibili (Unix)
- file system non condivisi: bisogna specificare a Condor cosa trasferire e quando farlo (Windows)

# Accesso ai Dati in Condor

- I comandi non sono validi per i job dell'universo standard
- E' preferibile usare il filesystem condiviso se disponibilie
- Se non e' disponibile?

  Condor può trasferire file

  - Restituire i file che sono stati modificati
  - Trasferimento atomico di più file
  - Trasferimento criptato

# Condor File Transfer

Nel file di sottomissione:

```
should_transfer_files = YES
                        NO
                        IF_NEEDED


when_to_transfer_output = ON_EXIT
                          ON_EXIT_OR_EVICT

transfer_input_files = filename1, filename2 . . .
```

# Frieda è felice; Fred non lo è

- Il Job di Fred viene eseguito a lungo
- Prima che il job abbia termine l'host diventa indisponibile
- Il job ritorna in coda e ricomincia dall'inizio

# Perché i job di Fred non terminano…

Preemption: La decisione di Condor di terminare un job in esecuzione

Perchè?

1. La policy dell'host e lo stato dell'host hanno condotto a questa decisione
2. Un altro job o il job di un altro utente con una priorità più alta deve essere eseguito al posto del job di Fred

# Lo standard universe di Condor può salvare Fred!

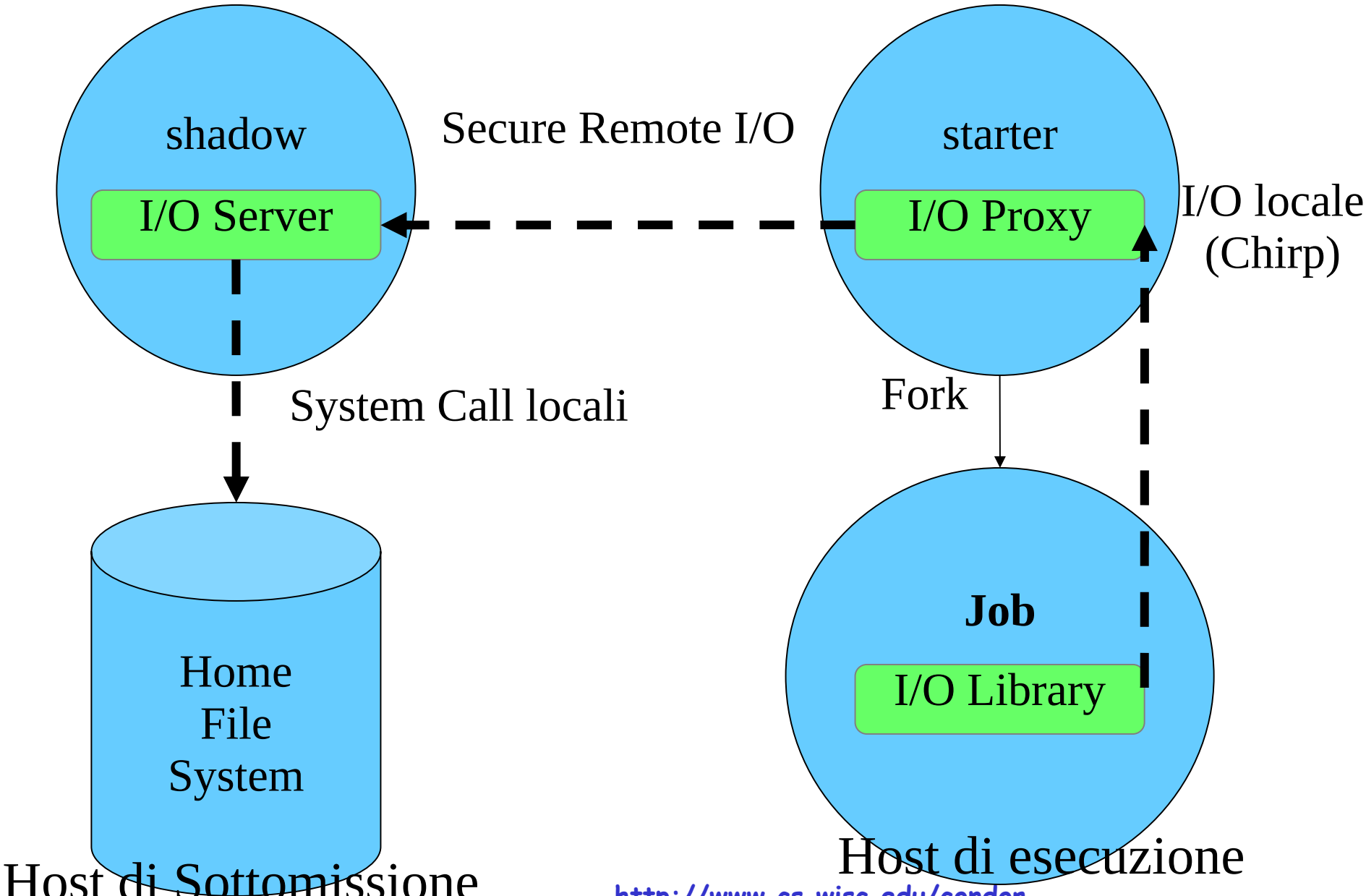
- System call remote (I/O remoto)

  Il job può accedere ai file in lettura e scrittura come se questi fossero file locali

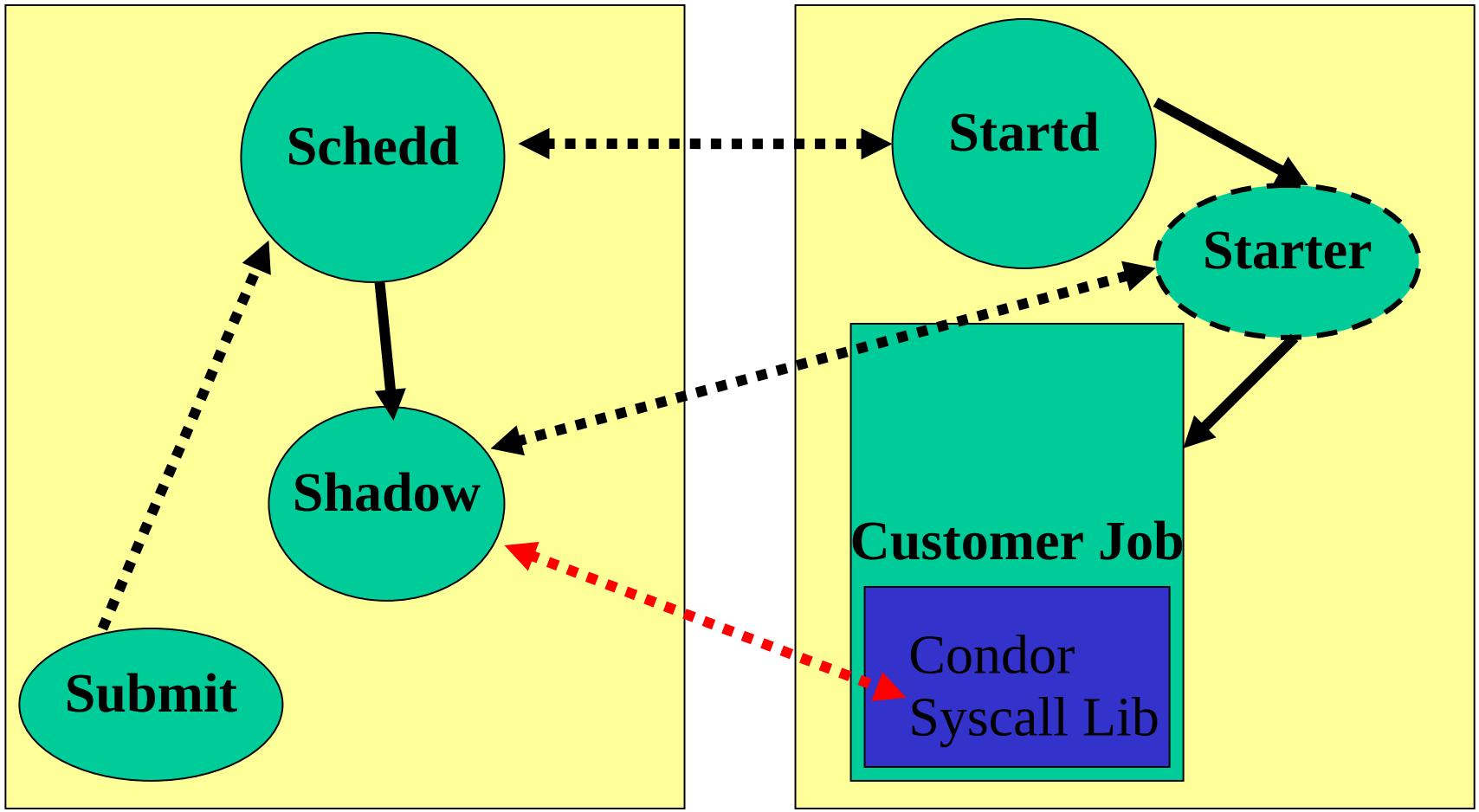
- Supporta il checkpoint e il riavvio trasparente

# System Call remote nello Standard Universe

- Le system call per l'I/O sono intercettate e inviate all'host che ha sottomesso il job
  Esempi: aprire un file, scrivere su di un file
- Consentono la migrazione trasparente
  - Checkpoint sulla risorsa A, restart su B
- Non è necessario cambiare il codice sorgente
- E' indipendente dal linguaggio di programmazione in cui è scritto il programma


shadow
I/O Server
Secure Remote I/O
starter
I/O Proxy
I/O locale
(Chirp)
System Call locali
Fork
Home
File
System
Job
I/O Library
Host di Sottomissione
Host di esecuzione

# Job Startup


Schedd
Startd
Starter
Shadow
Customer Job
Condor
Syscall Lib
Submit

# condor_q -io

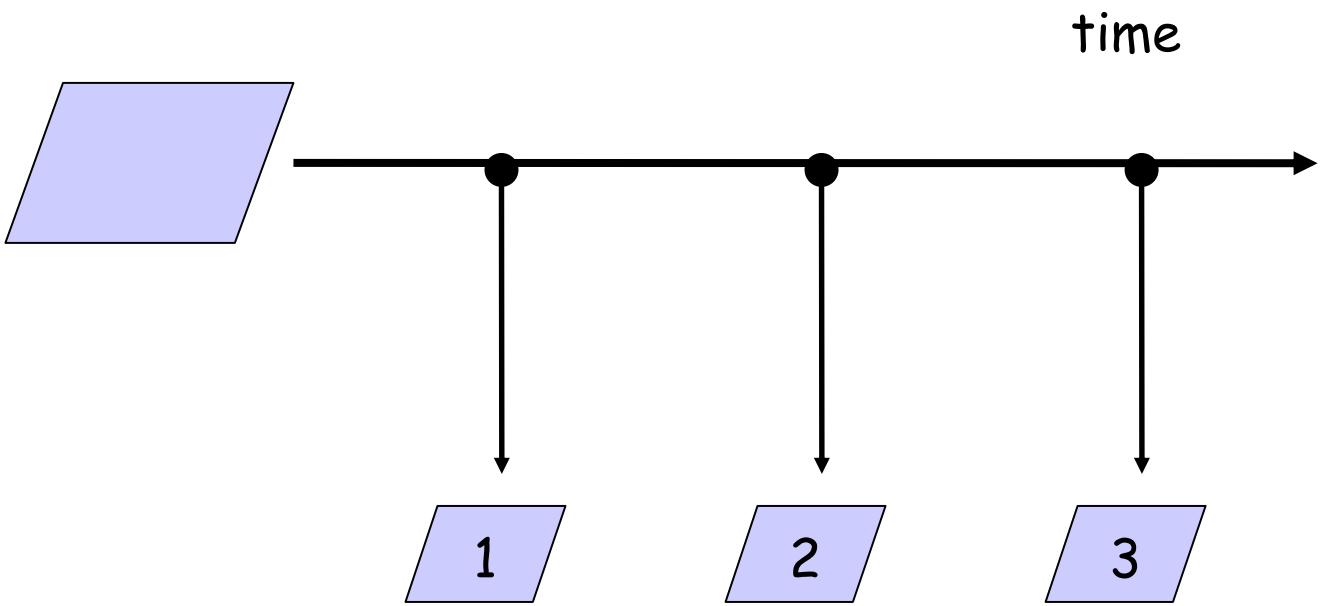
```
c01(69)% condor_q -io


-- Submitter: c01.cs.wisc.edu : <128.105.146.101:2996> : c01.cs.wisc.edu
ID       OWNER        READ    WRITE     SEEK       XPUT  BUFSIZE  BLKSIZE
  72.3   edayton           [ no i/o data collected yet ]
  72.5   edayton    6.8 MB   0.0 B         0 104.0 KB/s 512.0 KB  32.0 KB
  73.0   edayton    6.4 MB   0.0 B         0 140.3 KB/s 512.0 KB  32.0 KB
  73.2   edayton    6.8 MB   0.0 B         0 112.4 KB/s 512.0 KB  32.0 KB
  73.4   edayton    6.8 MB   0.0 B         0 139.3 KB/s 512.0 KB  32.0 KB
  73.5   edayton    6.8 MB   0.0 B         0 139.3 KB/s 512.0 KB  32.0 KB
  73.7   edayton           [ no i/o data collected yet ]

0 jobs; 0 idle, 0 running, 0 held
```

# Checkpoint

checkpoint: lo stato completo del programma viene salvato in un file

- i registri della CPU, l'immagine della memoria, l'I/O

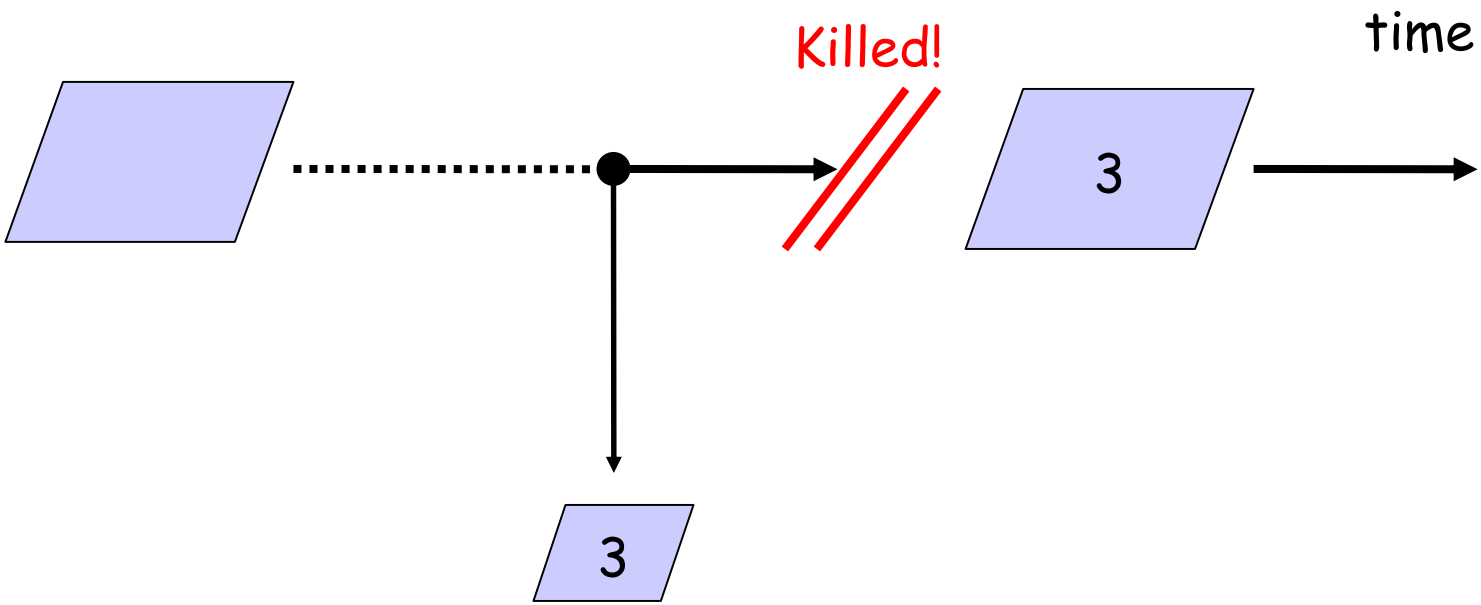
# Checkpoint



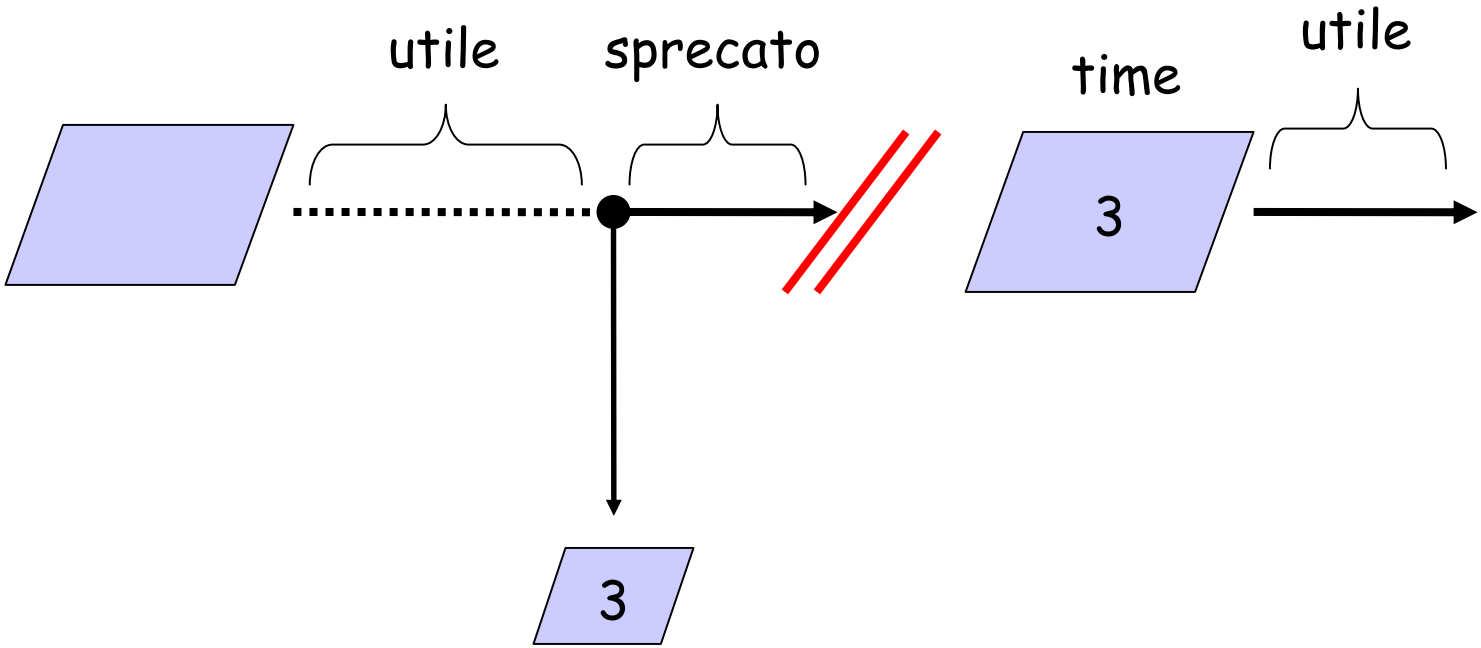

Possono essere effettuati ad intervalli regolari di tempo

http://www.cs.wisc.edu/condor

# Checkpoint



# Checkpoint



# Quando Condor effettua un checkpoint?

- Periodicamente, se richiesto, per attuare la fault tolerance
- Nel caso di preemption, prima dell'esecuzione di un job con priorità più alta, o perché l'host di esecuzione è diventato busy
- Quando l'utente esegue esplicitamente uno dei comandi condor_checkpoint, condor_vacate, condor_off condor_restart

# Quali passi eseguire per effettuare un checkpoint

- Il job deve essere linkato alla libreria dello standard universe di Condor
- Per linkarlo basta anteporre il comando **condor_compile** al comando che viene utilizzato per compilare il job:

```
% condor_compile gcc -o myjob myjob.c
```

- OPPURE -

```
% condor_compile f77 -o myjob filea.f fileb.f
```

- OPPURE -

```
% condor_compile make –f MyMakefile
```

# Limiti dello Standard Universe

- I checkpoint di Condor non vengono effettuati a livello kernel per cui i job dello standard universe non possono:
  - Usare l'istruzione `Fork()`
  - Usare thread del kernel
  - Usare alcuni tipi di IPC, quali pipe e memoria condivisa
  - Utilizzare compilatori non supportati (icc per esempio)
  - Essere eseguiti su macchine Windows
- Deve poter accedere al codice sorgente per effettuare il link
- Molti job per applicazioni del calcolo scientifico rientrano in queste limitazioni

# Non ci accontentiamo mai!

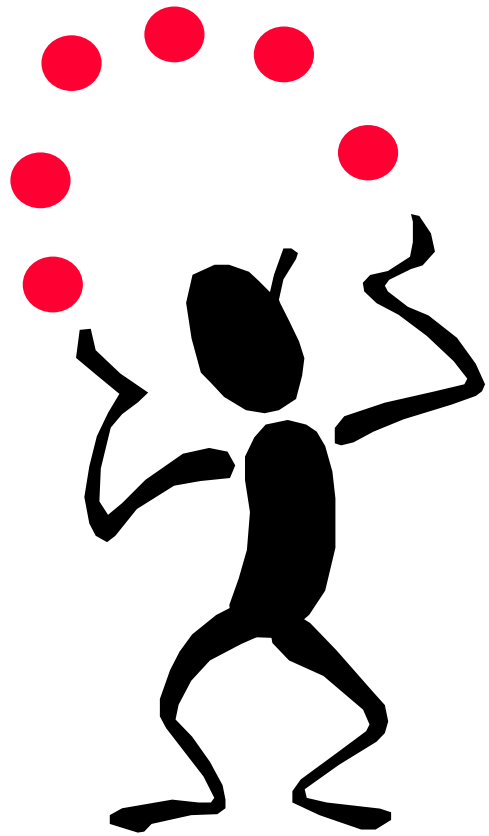
Condor gestisce ed esegue i nostri job, ma

- Abbiamo bisogno di più CPU di quante ne disponiamo
- I Job sono soggetti al preemption troppo spesso

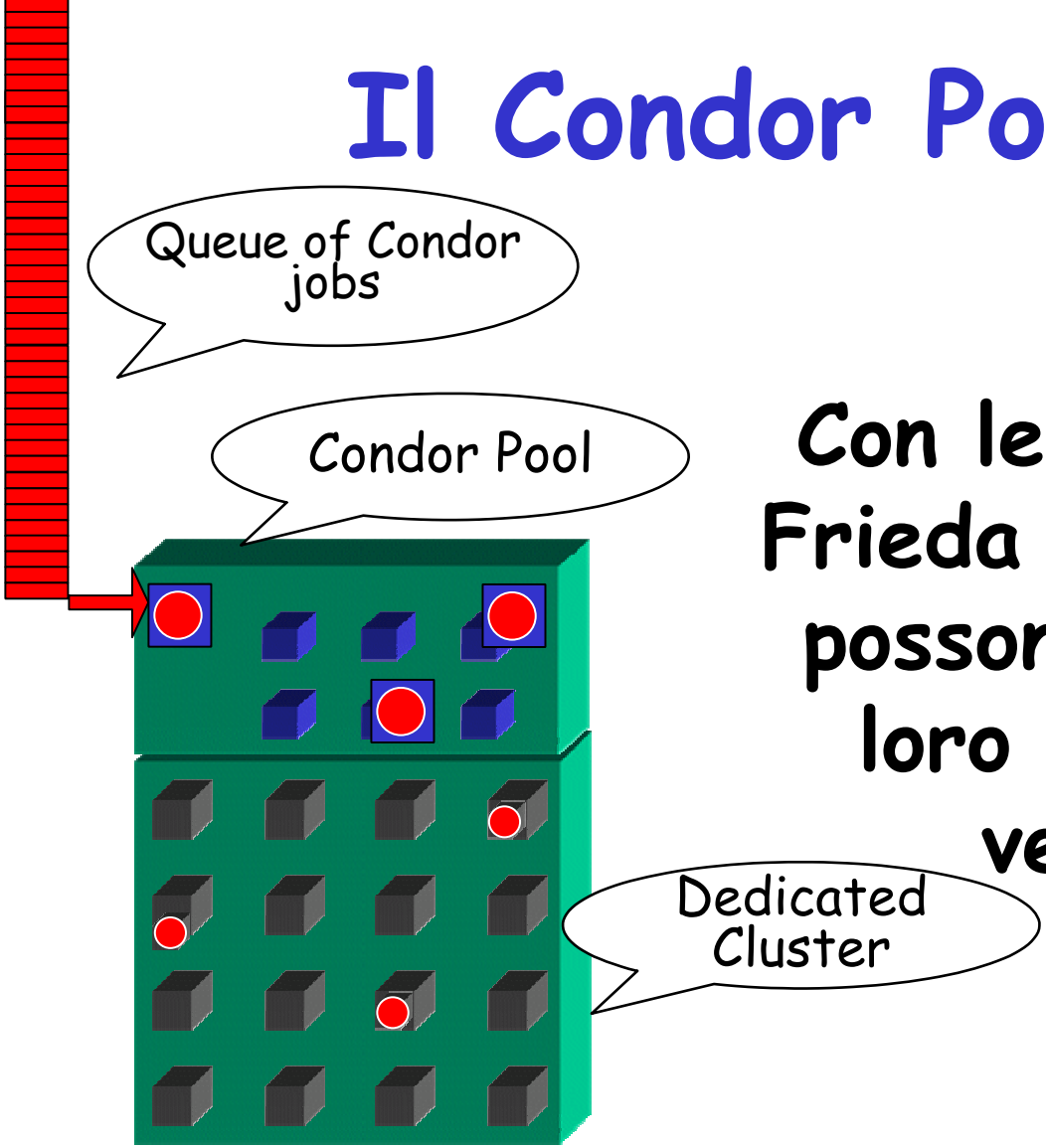
# Evviva! L'organizzazione di Frieda ha comprato un Cluster dedicato!

Allora Frieda:

- installa Condor sui nodi dedicati del Cluster
- aggiunge un central manager dedicato
- configura l'intero pool con questo nuovo host come central manager ...



# Il Condor Pool di Frieda




**Con le nuove risorse, Frieda e i suoi colleghi possono completare i loro job anche più velocemente**

# Ma Frieda ha un nuovo problema:

Alcuni calcolatori del pool non hanno abbastanza memoria o spazio scratch sul disco per i suoi job!!!

# Specifica dei Requisiti

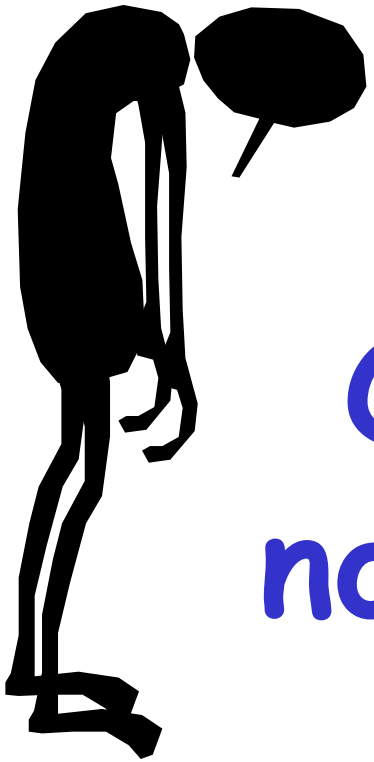
- Un requisito si esplicita attraverso un'espressione (con sintassi simile al C o al Java)
- L'espressione deve essere vera per eseguire un match tra job e calcolatore

```
Universe   = vanilla
Executable = b
Input      = b.input
Log        = b.log
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Queue 8
```

# Specifica del Rank

- Tutti i match che soddisfano i requisiti sono ordinati in base alle preferenze specificate con l'espressione Rank.
- Maggiore è il rank migliore è il match

```
Universe   = vanilla
Executable = b
Log        = b.log
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 8
```



# Ora i job di Frieda non vengono esegiuti…

# Cosa succede?

# Analisi della coda

```
% condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510> :x.cs.wisc.edu
ID  OWNER   SUBMITTED    RUN_TIME ST PRI SIZE CMD
5.0 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.1 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.2 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.3 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.4 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.5 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.6 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
5.7 frieda  4/20 12:23  0+00:00:00 I 0   9.8  b
6.0 frieda  4/20 13:22  0+00:00:00 H 0   9.8  my_job
8 jobs; 8 idle, 0 running, 1 held
```

# Osserviamo i job in stato di hold

```
% condor_q –hold
-- Submiter: x.cs.wisc.edu : <128.105.121.53:510>
   :x.cs.wisc.edu
 ID      OWNER            HELD_SINCE HOLD_REASON
 6.0   frieda             4/20 13:23 Error from starter on
   vm1@skywalker.cs.wisc

9 jobs; 8 idle, 0 running, 1 held
```

Visualizziamo i dettagli di un job

```
% condor_q –l 6.0
```

# Verifichiamo lo stato dei calcolatori

Verifichiamo che ci siamo calcolatori in stato idle con condor_status:

```
% condor_status
Name          OpSys  Arch  State   Activity LoadAv Mem   ActvtyTime
vm1@tonic.c LINUX  INTEL Claimed Busy     0.000   501  0+00:00:20
vm2@tonic.c LINUX  INTEL Claimed Busy     0.000   501  0+00:00:19
vm3@tonic.c LINUX  INTEL Claimed Busy     0.040   501  0+00:00:17
vm4@tonic.c LINUX  INTEL Claimed Busy     0.000   501  0+00:00:05

               Total Owner Claimed Unclaimed Matched Preempting
INTEL/LINUX        4     0       4         0       0          0
      Total        4     0       4         0       0          0
```

# Verifichiamo il file di log del Job

Il file di log del job potrebbe contenere degli indizi:

```
% cat b.log
000 (031.000.000) 04/20 14:47:31 Job submitted from host:
   <128.105.121.53:48740>
...
007 (031.000.000) 04/20 15:02:00 Shadow exception!
        Error from starter on gig06.stat.wisc.edu: Failed
   to open '/scratch.1/frieda/workspace/v67/condor-
   test/test3/run_0/b.input' as standard input: No such
   file or directory (errno 2)
        0   -   Run Bytes Sent By Job
        0   -   Run Bytes Received By Job
...
```

# Quanto tempo occorre? Bisogna avere pazienza…

- Su un pool ha molti job da eseguire può essere necessario attendere un po' di tempo per l'esecuzione dei proprijob
- Bisogna aspettare almeno un ciclo di negoziazione (tipicamente 5 minuti)

# Utilizzo di condor_q per indagare…

```
% condor_q -analyze 29
---
029.000:  Run analysis summary.  Of 1243 machines,
   1243 are rejected by your job's requirements
0 are available to run your job

WARNING:  Be advised:
   No resources matched request's constraints
   Check the Requirements expression below:
Requirements = ((Memory > 8192)) && (Arch == "INTEL") &&
   (OpSys == "LINUX") && (Disk >= DiskUsage) &&
   (TARGET.FileSystemDomain == MY.FileSystemDomain)
```

# Un'indagine migliore ma più lenta

```
% condor_q -better-analyze 29
The Requirements expression for your job is:

( ( target.Memory > 8192 ) ) && ( target.Arch == "INTEL" ) &&
( target.OpSys == "LINUX" ) && ( target.Disk >= DiskUsage ) &&
( TARGET.FileSystemDomain == MY.FileSystemDomain )
Condition                          Machines Matched  Suggestion
---------                          ----------------  ----------
1 ( ( target.Memory > 8192 ) )     0                 MODIFY TO 4000
2 ( TARGET.FileSystemDomain == "cs.wisc.edu" )584
3   ( target.Arch == "INTEL" )     1078
4   ( target.OpSys == "LINUX" )    1100
5   ( target.Disk >= 13 )          1243
```

# Esploriamo il pool:

```
% condor_status –constraint 'Memory > 8192'
```

(no output means no matches)

```
% condor_status -constraint 'Memory > 4096'
Name           OpSys  Arch   State     Activ LoadAv Mem   ActvtyTime
vm1@s0-03.cs. LINUX  X86_64 Unclaimed Idle  0.000  5980  1+05:35:05
vm2@s0-03.cs. LINUX  X86_64 Unclaimed Idle  0.000  5980 13+05:37:03
vm1@s0-04.cs. LINUX  X86_64 Unclaimed Idle  0.000  7988  1+06:00:05
vm2@s0-04.cs. LINUX  X86_64 Unclaimed Idle  0.000  7988 13+06:03:47

                     Total Owner Claimed Unclaimed Matched Preempting
   X86_64/LINUX          4     0       0         4       0          0
          Total          4     0       0         4       0          0
```

# Inserire attributi nelle ClassAd

- E' possibile inserire nel submit description file attributi per il job
- `+Department = biochemistry`

  la ClassAd del job conterrà

  `Department = "biochemistry"`

# Il condor pool di Frieda preferisce eseguire job di biochimica

- Nella configurazione delle macchine del dipartimento di chimica avremo:

```
RANK = (Department == ”biochemisty”)
```

# Quali daemon di Condor sono in esecuzione sulla macchina di Frieda e cosa fanno?

http://www.cs.wisc.edu/condor

# Dietro le quinte

- C'è una discreta quantità di software in esecuzione per far funzionare Condor
- Le varie componenti software hanno la struttura di daemon.
  - I daemon di Condor comunicano tra di loro
  - Ogni daemon ha un proprio compito specifico

# I daemon di Condor

master: Controlla gli altri processi

collector: Memorizza le ClassAds

negotiator: Esegue il matchmaking

schedd: Gestisce le code di job

shadow: Gestisce i job (lato submit)

startd: Gestisce gli host

starter: Gestisce i job (lato esecuzione)

# Ruolo degli host in un Condor pool

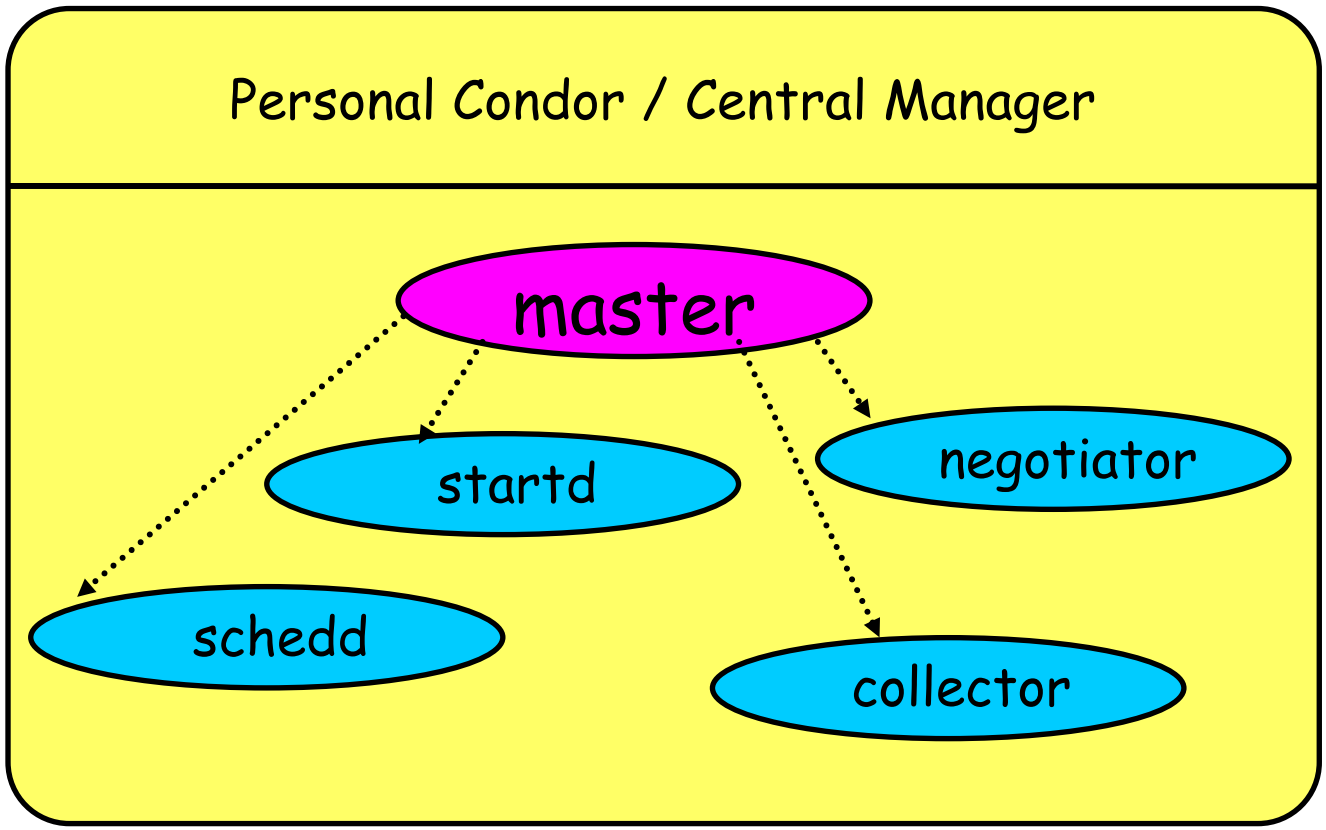
- Central manager: gestore centrale che opera il matchmaking per l'intero pool
- Execute machine: un host che esegue job degli utenti
- Submit machine: un host dal quale l'utente può sottomettere job

# condor_master

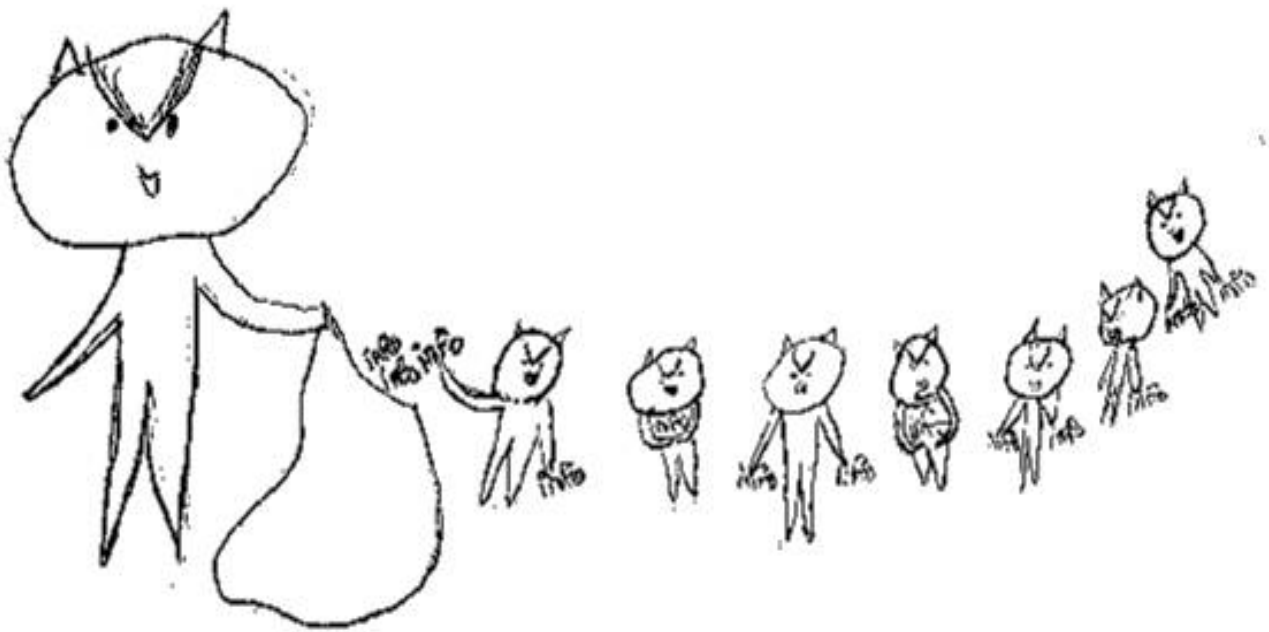
# condor_master

- Avvia gli altri daemon
- Provoca l'interruzione dell'esecuzione di un daemon se si verifica un problema, invia una e-mail all'amministratore e poi riavvia il daemon
- Accetta ed esegue comandi di amministrazione da remoto:
  - **condor_reconfig, condor_restart, condor_off, condor_on, condor_config_val**, etc.

# Condor Daemon Layout: condor_master



# condor_collector



http://www.cs.wisc.edu/condor

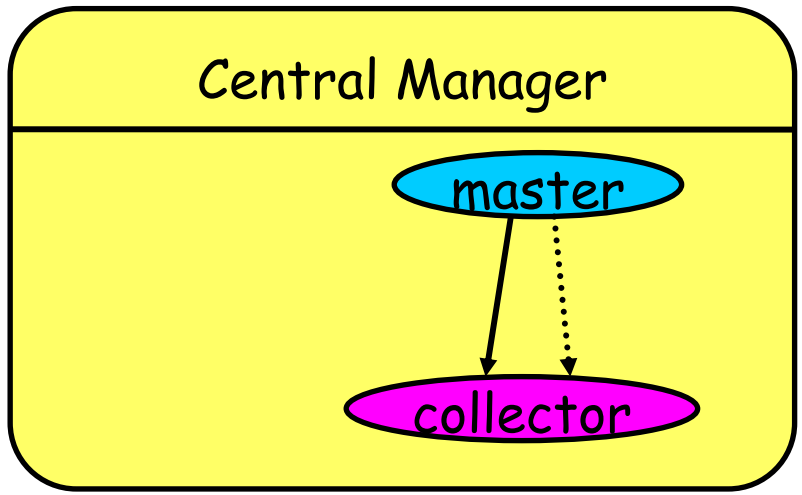
# condor_collector

- E' necessariamente presente in ogni condor pool (almeno uno)
- E' eseguito sul central manager
- Raccoglie le informazioni su tutti gli altri daemon del pool
  - "Directory Service" / Database per il Condor pool
- Riceve periodicamente da ogni daemon una ClassAd
- Risponde alle richieste di informazioni:
  - Richieste dagli altri daemon di Condor
  - Richieste dagli utenti (**condor_status**)

# Condor Pool Layout: condor_collector

= Process Spawned

= ClassAd Communication Pathway



# condor_negotiator

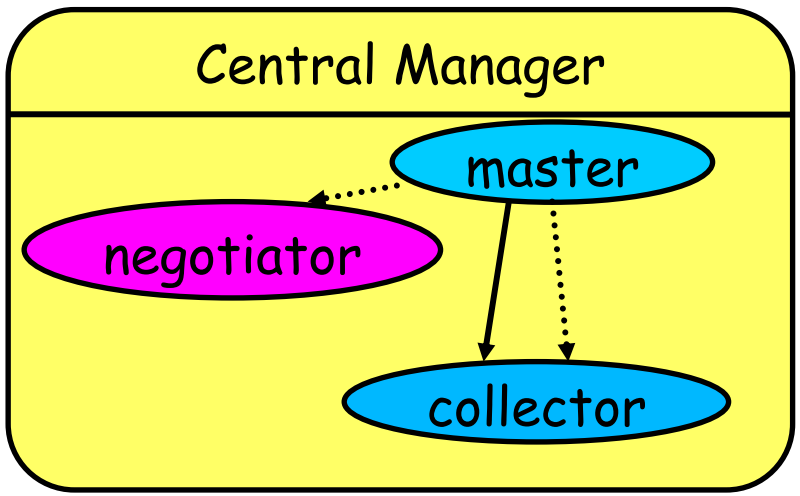
# condor_negotiator

- E' necessariamente presente in ogni condor pool (SOLO uno)
- E' eseguito sul central manager
- Esegue il matchmaking
- In ogni ciclo di negoziazione (tipicamente 5 minuti):
  - Riceve informazioni sui calcolatori disponibili e sui job in attesa di esecuzione dal collector
  - Prova a i match tra macchine e job
  - Job e macchine devono soddisfare i requirement reciprocamente

# Condor Pool Layout: condor_negotiator

= Process Spawned

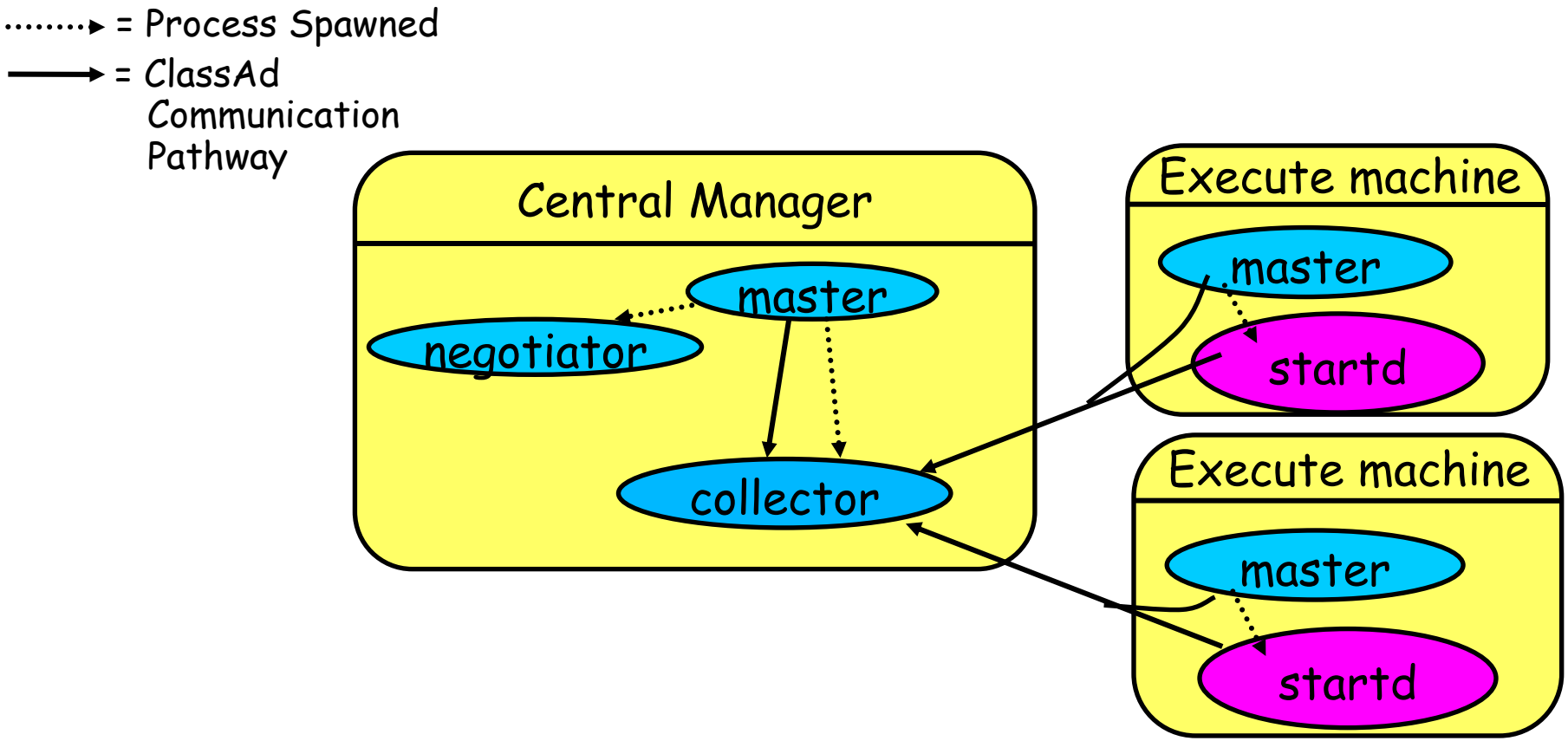
= ClassAd Communication Pathway



# condor_startd

- E' in esecuzione su ogni calcolatore che effettua esecuzioni di job
- Rappresenta per Condor proprio il calcolatore su cui è in esecuzione
- Esegue, sospende e arresta i job dell'utente
- Gestisce il calcolatore secondo le direttive del proprietario (la sua policy)
- Crea un processo condor_starter per ogni job in esecuzione

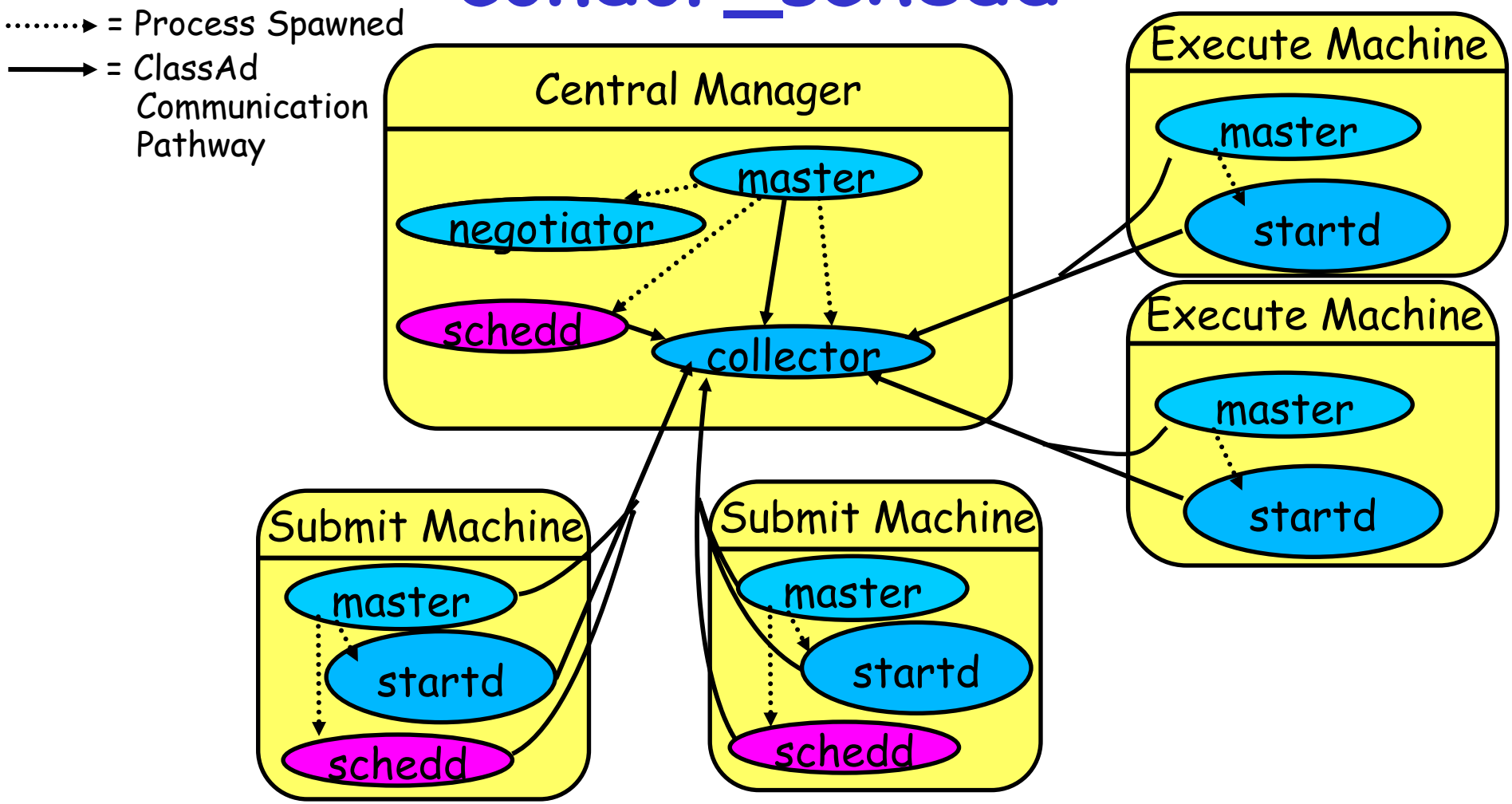
# Condor Pool Layout: condor_startd



# condor_schedd

- E' in esecuzione su ogni un calcolatore del pool
- Conserva la coda dei job
- Contatta il calcolatore disponibile all'esecuzione e gli spedisce i job
- Risponde ai comandi dell'utente che gestiscono i job:
  **condor_submit,condor_rm, condor_q, condor_hold, condor_release, condor_prio**
- Crea il daemon condor_shadow per ogni job in esecuzione

# Condor Pool Layout: condor_schedd



# Comandi utili per l'utente

| | |
|---|---|
| condor_status | Visualizza lo stato del pool |
| condor_q | Visualizza la coda dei Job |
| condor_submit | Sottomette un nuovo Job |
| condor_rm | Cancella un Job dalla coda |
| condor_prio | Priorità dei job dell'utente |
| condor_history | Info complete sul Job |
| condor_checkpoint | Forza un checkpoint |
| condor_compile | Linka la libreria Condor |
| condor_hold | Mette il job in stato di attesa (hold) |

# Comandi per l' amministratore

| | |
|---|---|
| condor_on | Avvia Condor |
| condor_off | Arresta Condor |
| condor_reconfig | Riconfigura "al volo" |
| condor_config_val | Gestisce la configurazione |
| condor_userprio | Gestisce priorità utenti |
| condor_stats | Visualizza dettagliate statistiche d'uso |
| condor_vacate | Libera una macchina dai job in esecuzione |

http://www.cs.wisc.edu/condor

# Come monitorare e registrare cosa sta facendo Condor?

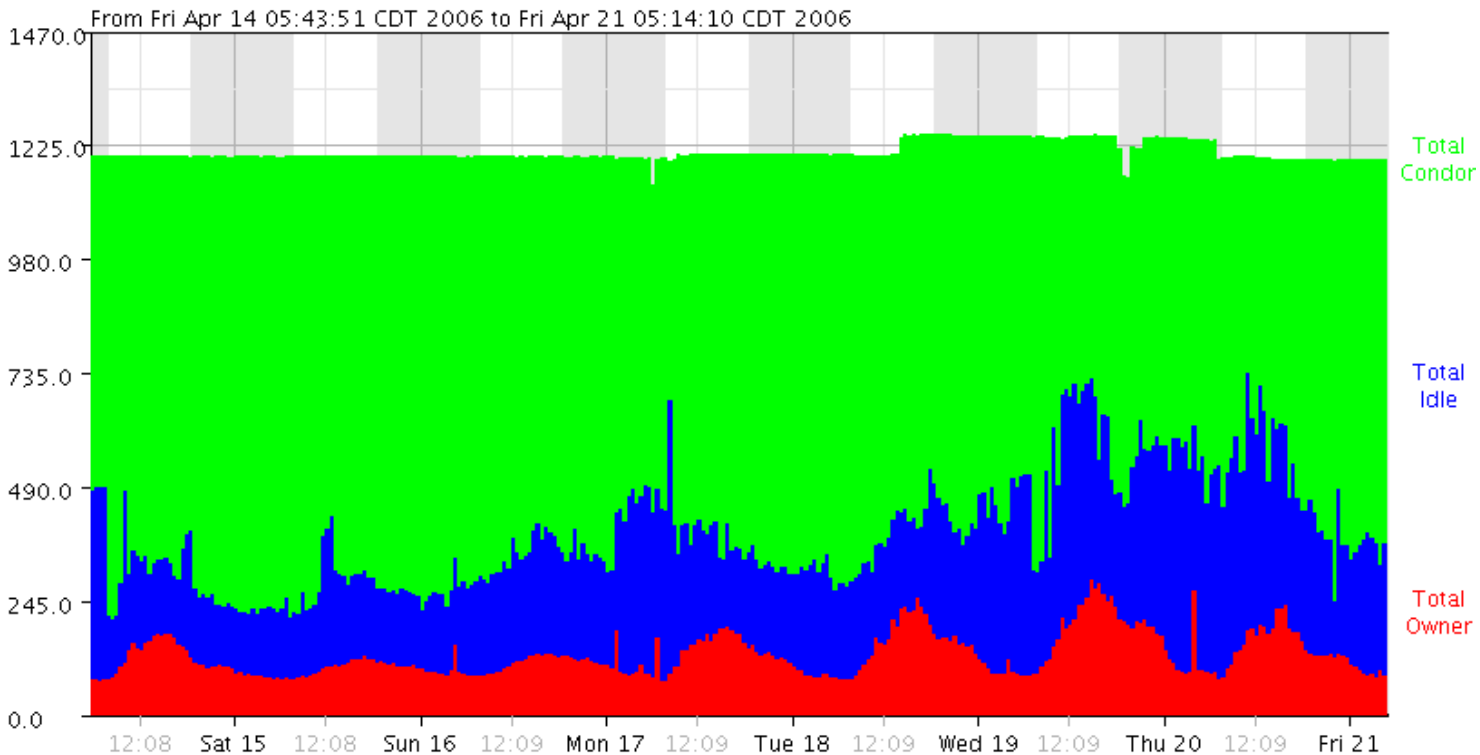
http://www.cs.wisc.edu/condor

# Il CondorView!

- Grafici di utilizzazioni presenti e passate
- I dati provengono direttamente dalle statistiche interne di Condor
- Java applet interattivo
- Visuale semplice e veloce:
  - Quanto è stato usato Condor
  - Quanti cicli idle sono stati utilizzati
  - Chi sta usando i cicli idle
  - Utilizzazione in base alle caratteristiche delle macchine e agli utenti

http://www.cs.wisc.edu/condor

# Grafico di Utilizzo con CondorView




http://www.cs.wisc.edu/condor

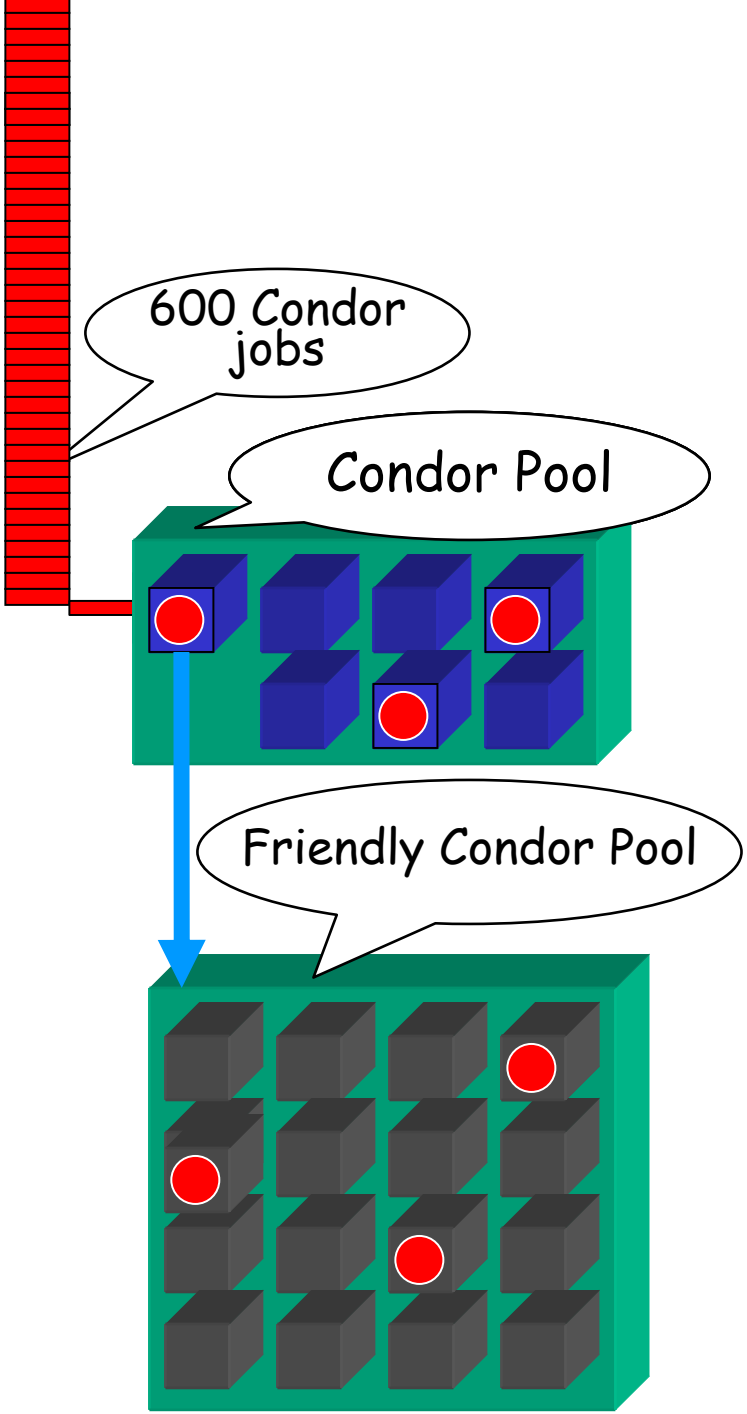
# Numeri di Versione Condor

- I numeri di versione di condor adottano la stessa convenzione del kernel di Linux (pre 2.6)

- Versioni Stabili , principalmente risolvono bug
  - Numerazione: major.minor.release, la maggiore è pari
  - Versione stabile attualmente disponibile: 6.8.5
- Versioni di Sviluppo , con innovazioni ma una maggiore possibilità di trovare bug
  - Numerazione: major.minor.release, la minore è dispari
  - Current developer release: 6.9.3

# Frieda va in Grid!

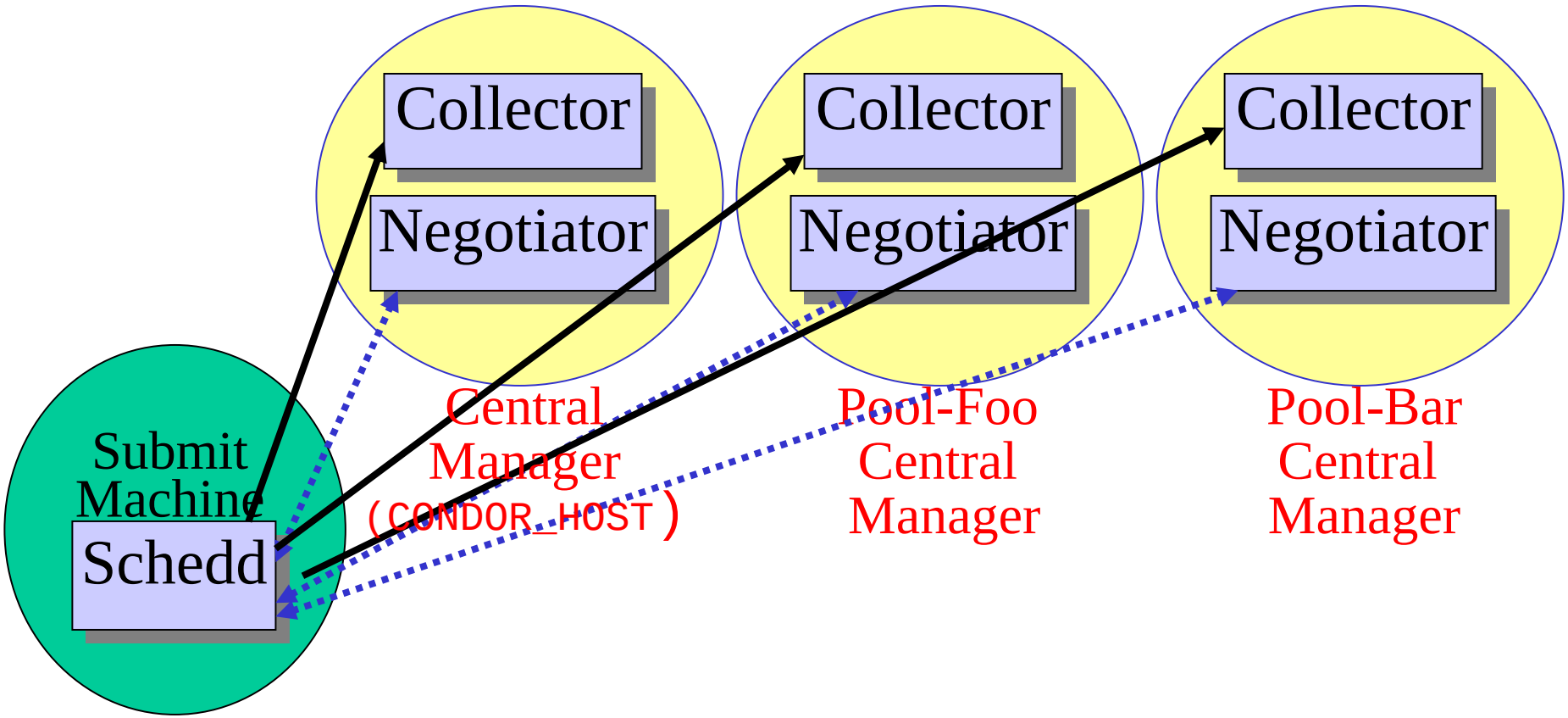
- Prima Frieda ha collaborato con i suoi amici del club di Condor!
- Adesso conosce anche altre persone che possiedono un Condor pool e l'autorizzano a utilizzare le proprie risorse
- Frieda quindi configura il suo condor pool in modo da riempire "flock" gli altri pool


600 Condor jobs
Condor Pool
Friendly Condor Pool

# Come funziona il Flocking

- Aggiunge una linea al condor_config :

```
FLOCK_HOSTS = Pool-Foo, Pool-Bar
```



# Condor Flocking

- I pool remoti sono contattati nell'ordine specificato in FLOCK_HOSTS
- La lista dei pool remoti è una proprietà dello Schedd e non del Central Manager
  - In questo modo utenti differenti possono riempire differenti pool
  - e pool remoti possono consentire l'accesso ad user specifici
- Il sistema di priorità e "flocking-aware"
  - Gli utenti locali di un pool possono avere priorità rispetto ai remoti che fanno "flocking"

# Da Flocking a Condor-G

- Flocking è una tecnologia specifica di "Condor"...
- Frieda ha anche accesso a risorse grid che lei vuole utilizzare
  - Ha un certigicato Globus e vuole accedere alle risorse di un'altra orgnizzazione in remoto
- Ma Frieda desidera continuare a gestire la coda dei propri job con Condor!
- Le basta usare Condor-G e sottomettere i propri job all'universo "Globus"

# Frieda sottomette

# un job a Globus

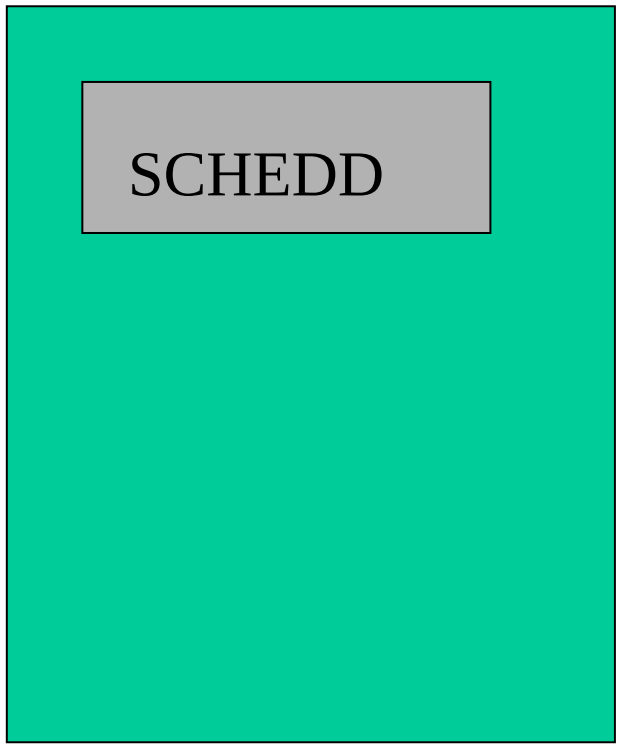
- Nel suo submit description file specifica :
  - Universe = Globus
  - Il gestore di Globus da contattare
  - Opzionalmente: La locazione del suo certificato

```
universe     = globus
globusscheduler = beak.cs.wisc.edu/jobmanager
executable   = progname
queue
```

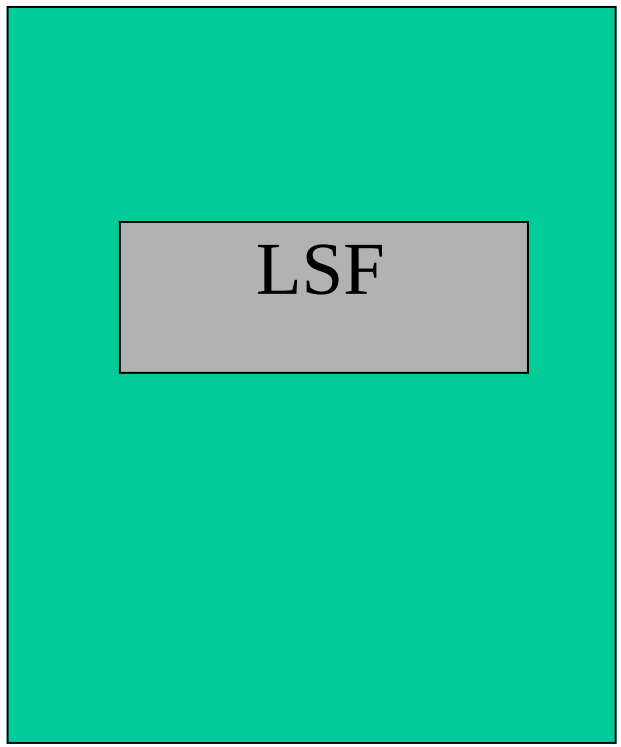
http://www.cs.wisc.edu/condor

# Come funziona

Personal Condor

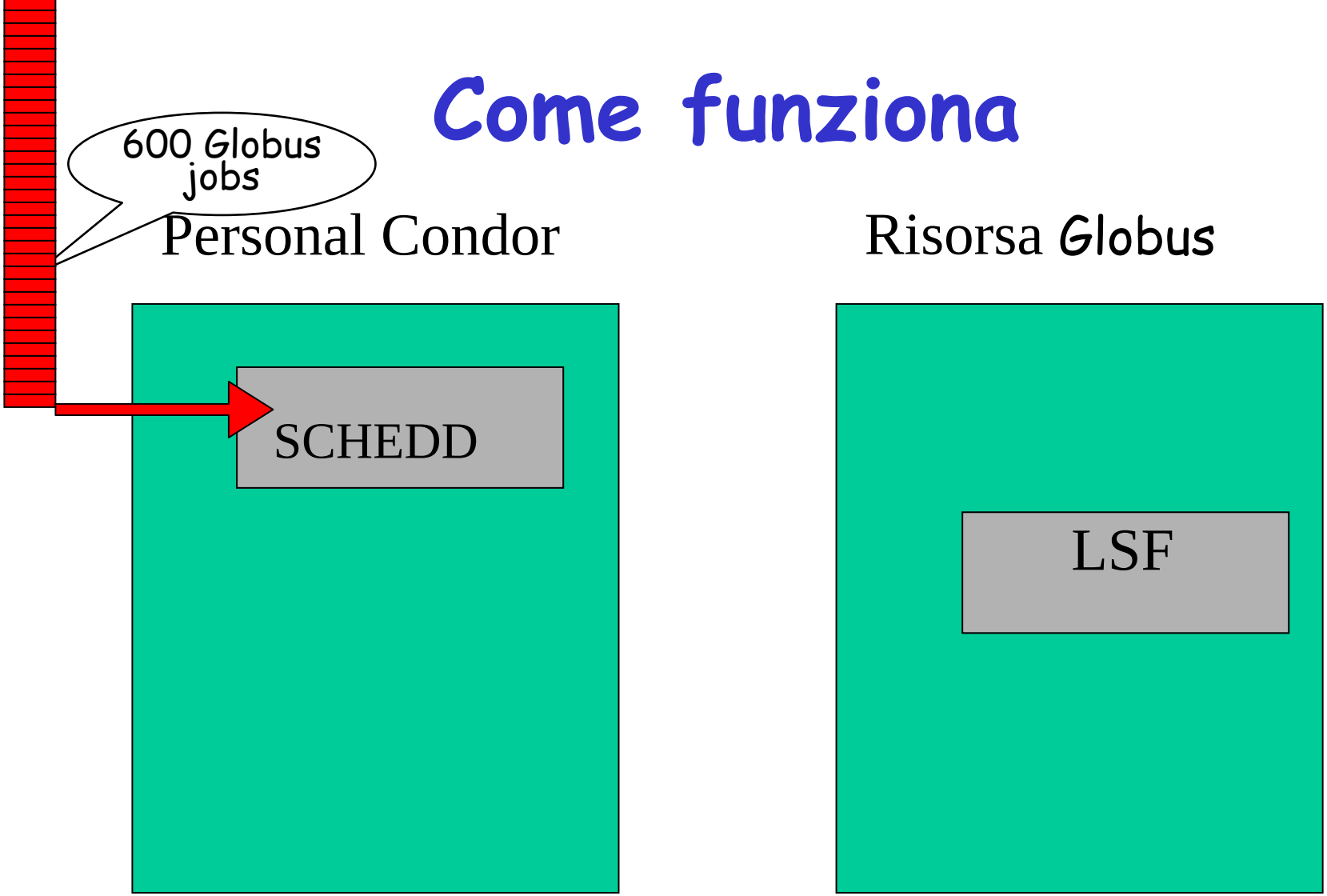



Risorsa Globus

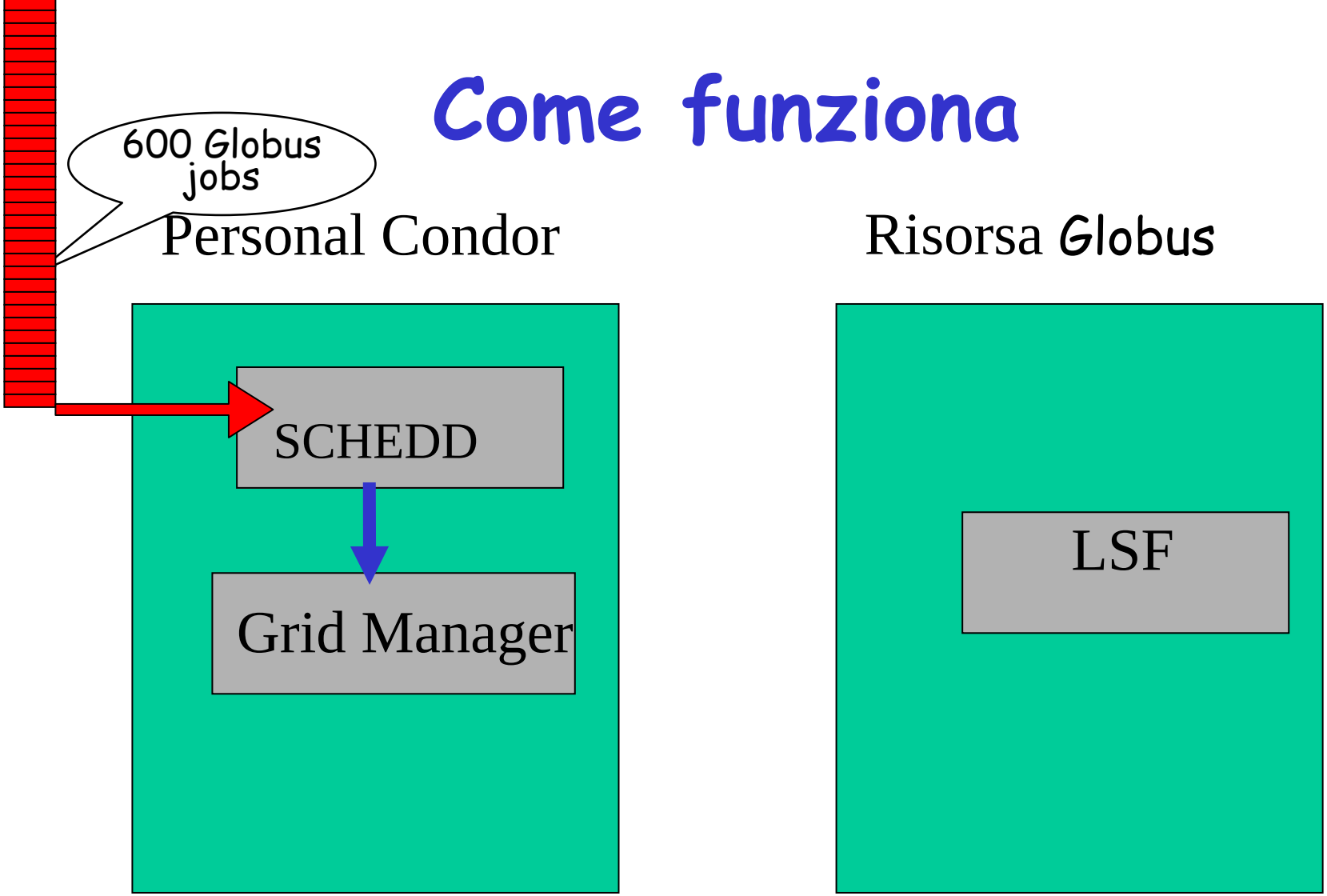



http://www.cs.wisc.edu/condor

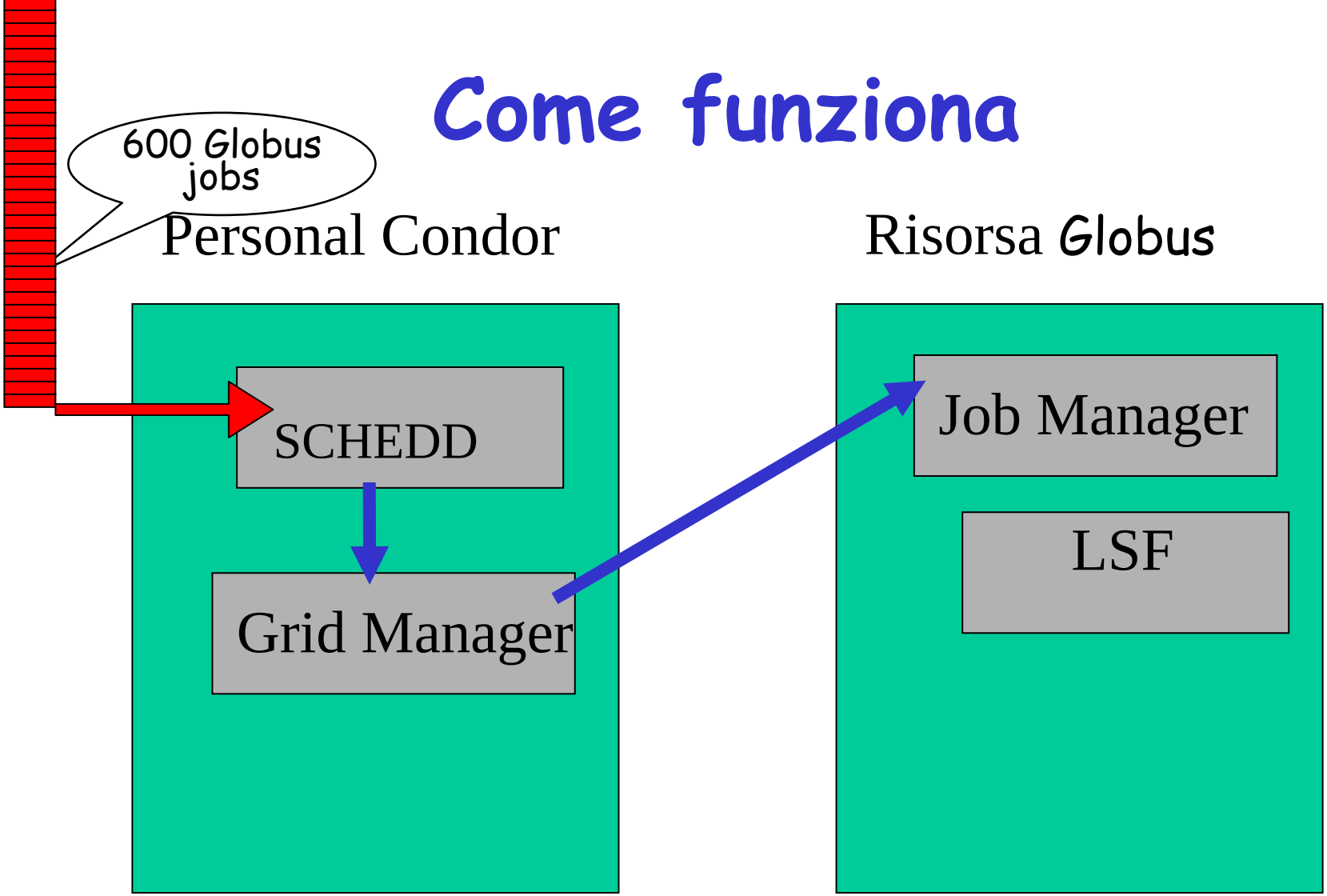
# Come funziona



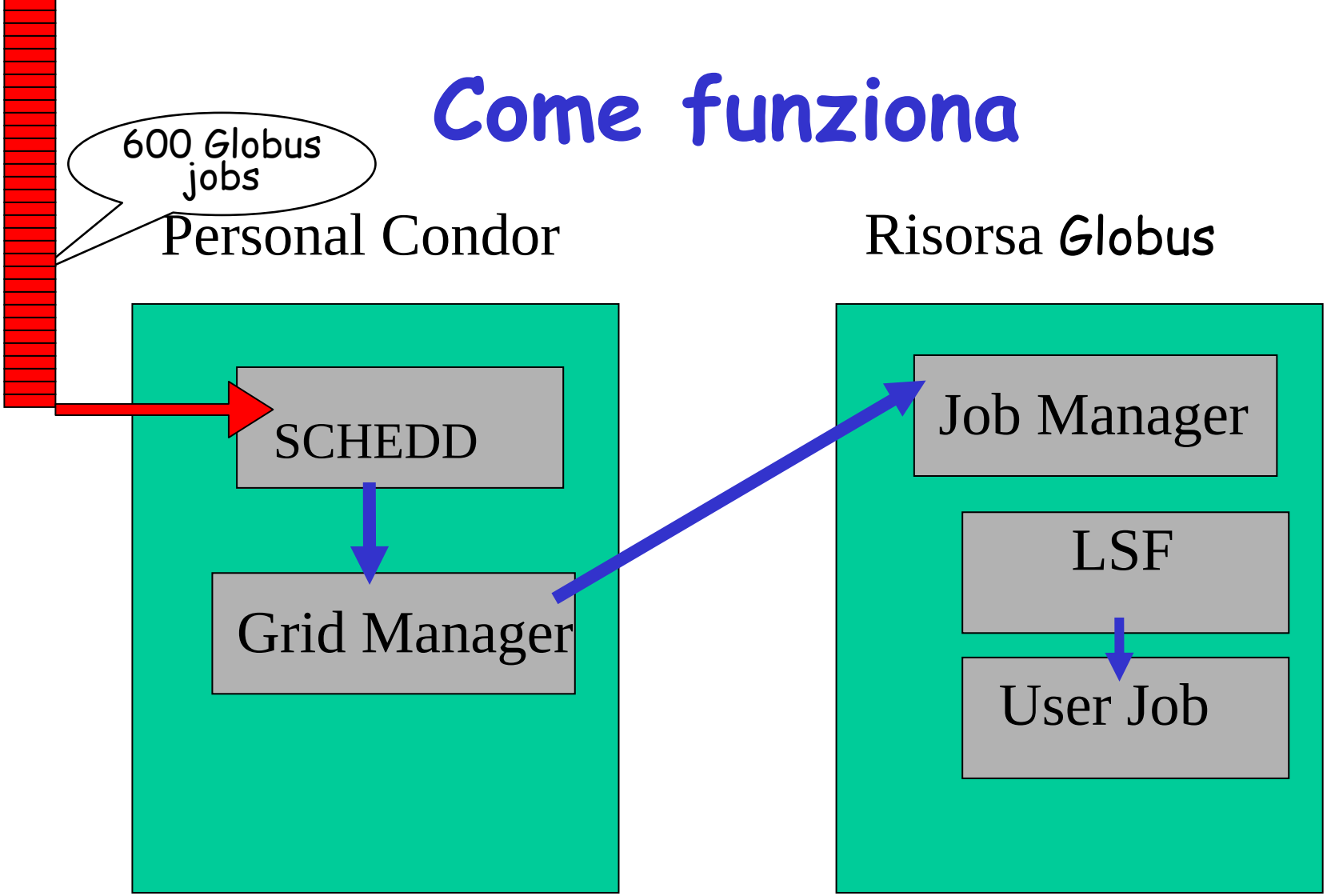
# Come funziona



# Come funziona



# Come funziona



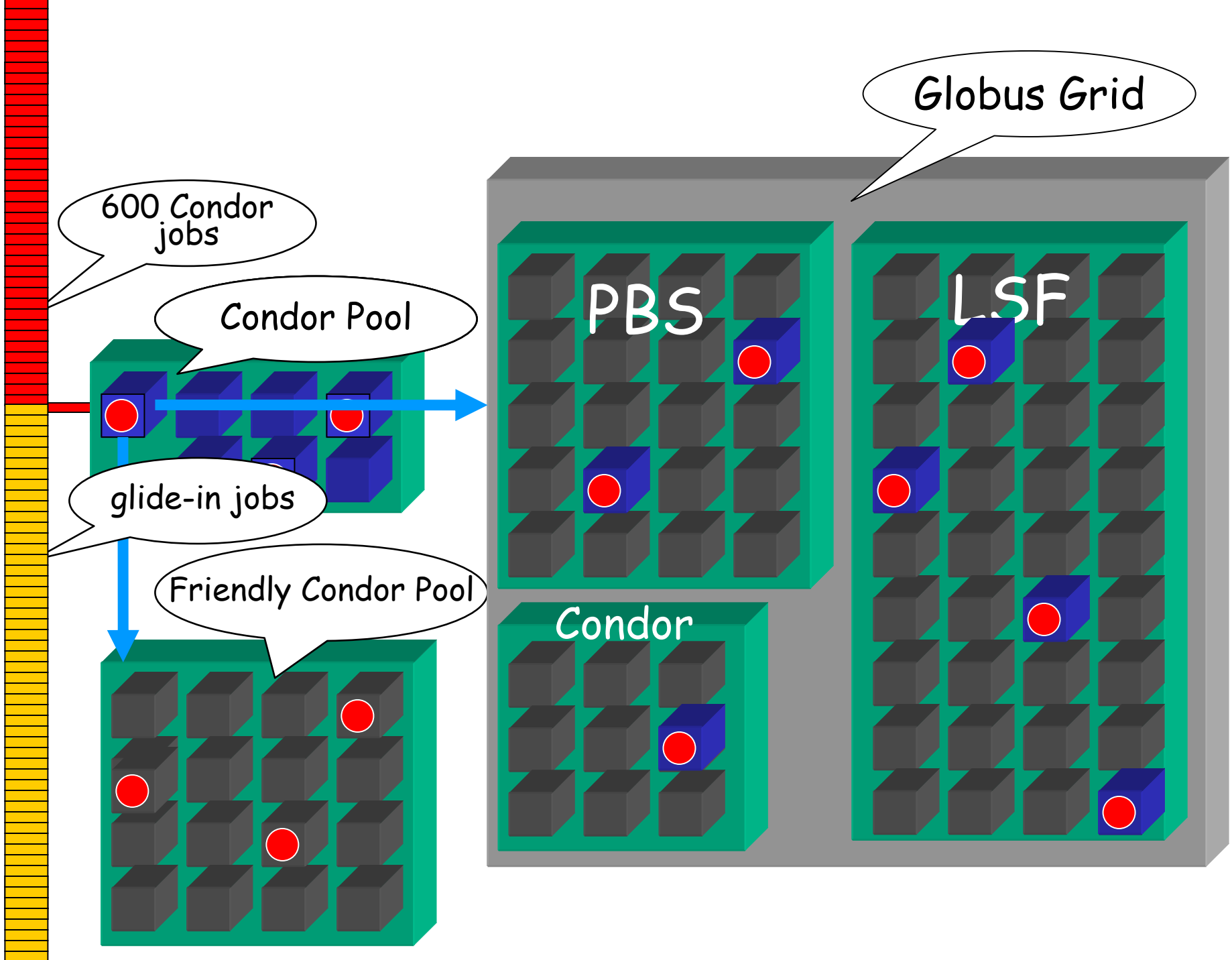
# Ma Frida Vuole di più…

- Frieda vuole eseguire i job dello standard universe sulle risorse gestite da Globus
  - In modo da poter effettuare matchmaking e schedulare i job dinamicamente
  - Per utilizzare le tecnologie di checkpointing e migrazione
  - Per utilizzare le system call remote

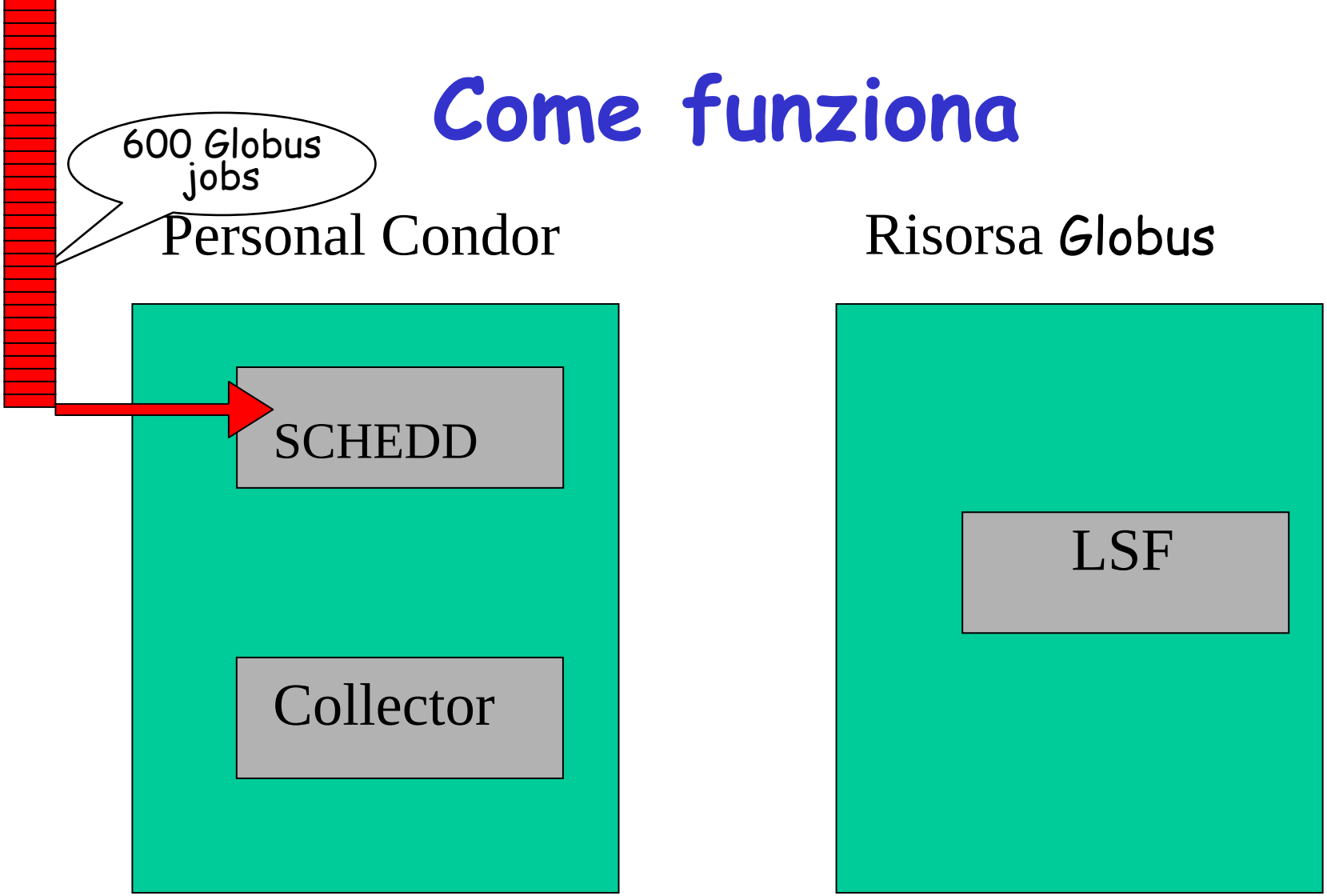
# Soluzione: Condor GlideIn

- Frieda può utilizzare l'universo Globus per eseguire i deamon di Condor sulle risorse Globus
- Quando le risorse eseguono questi job entrano a far parte temporaneamente del pool di Frieda
- Frieda quindi può sottomettere job agli universi Standard, Vanilla, PVM, o MPI che verrano matchati sulle risorse Globus resources

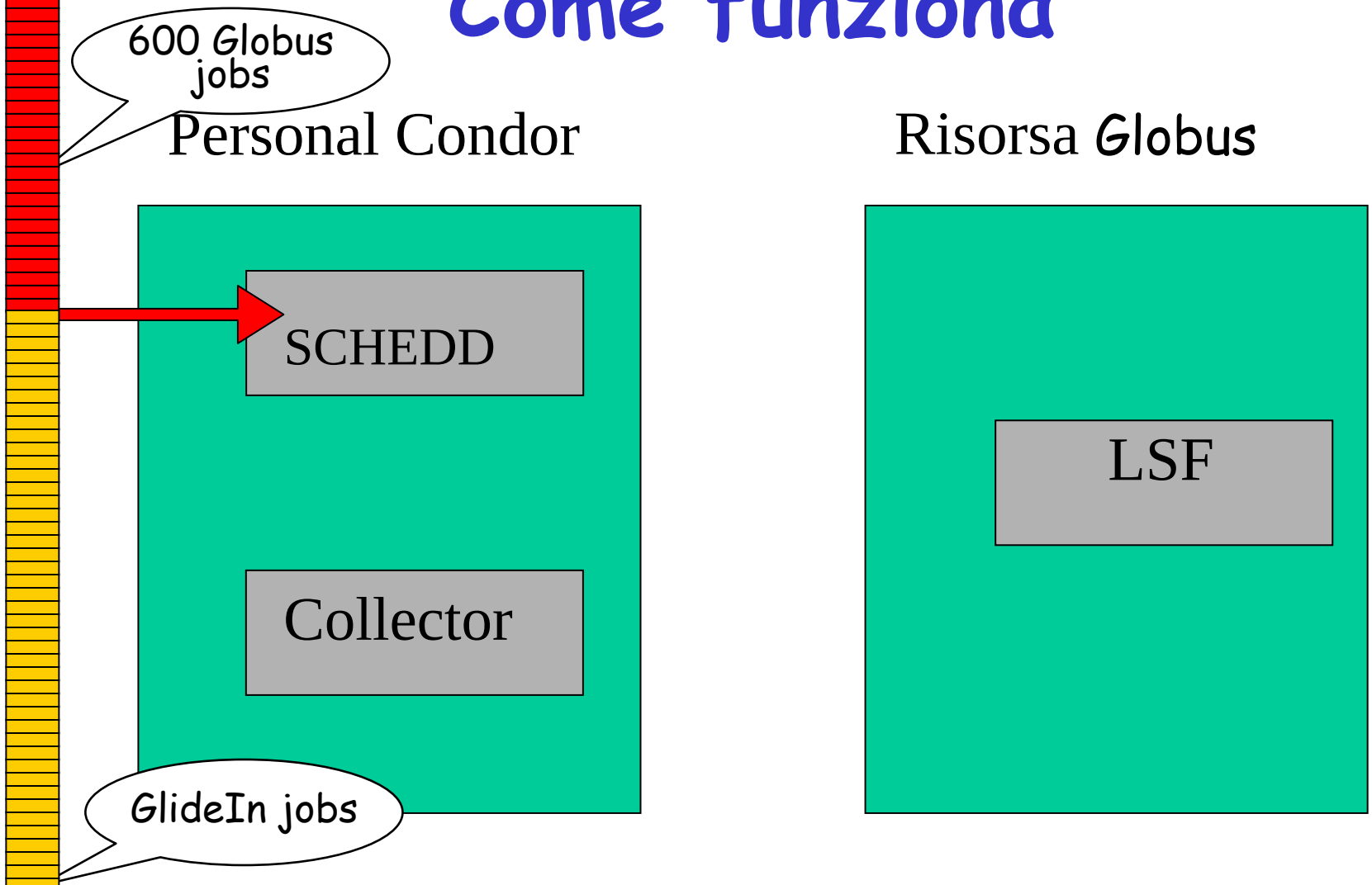
http://www.cs.wisc.edu/condor


Globus Grid
600 Condor jobs
Condor Pool
PBS
LSF
glide-in jobs
Friendly Condor Pool
Condor

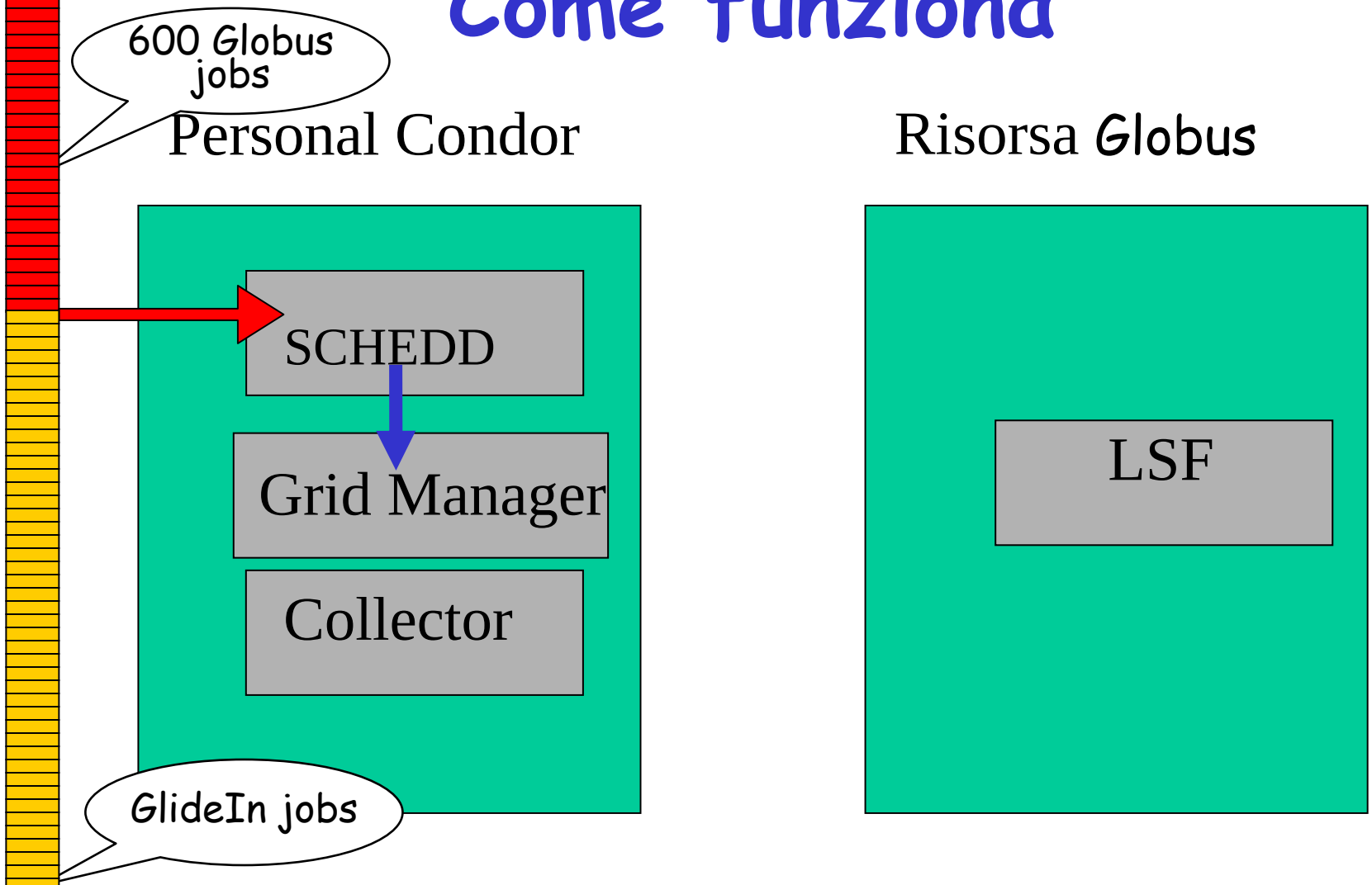
# Come funziona



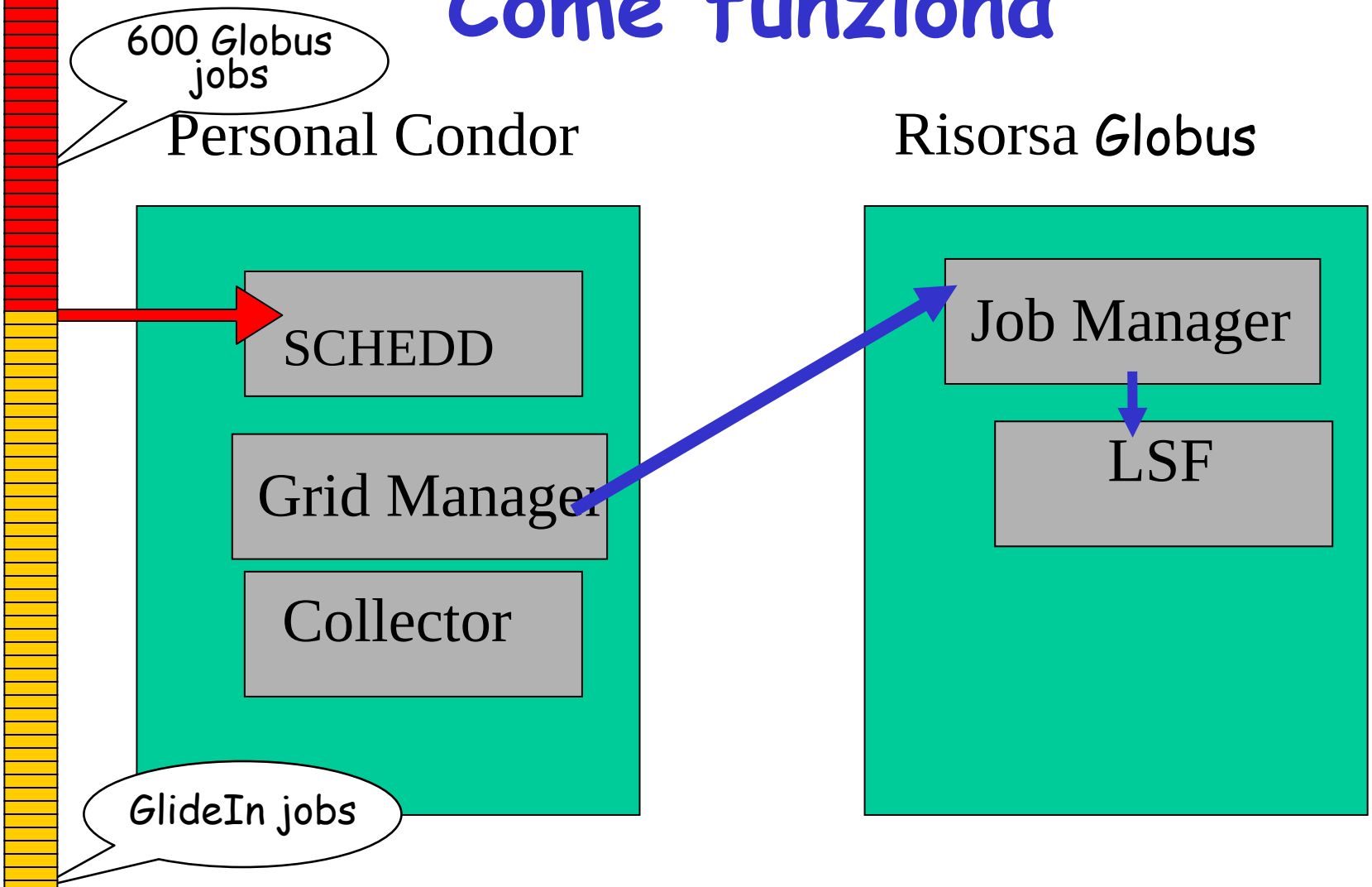
# Come funziona



# Come funziona

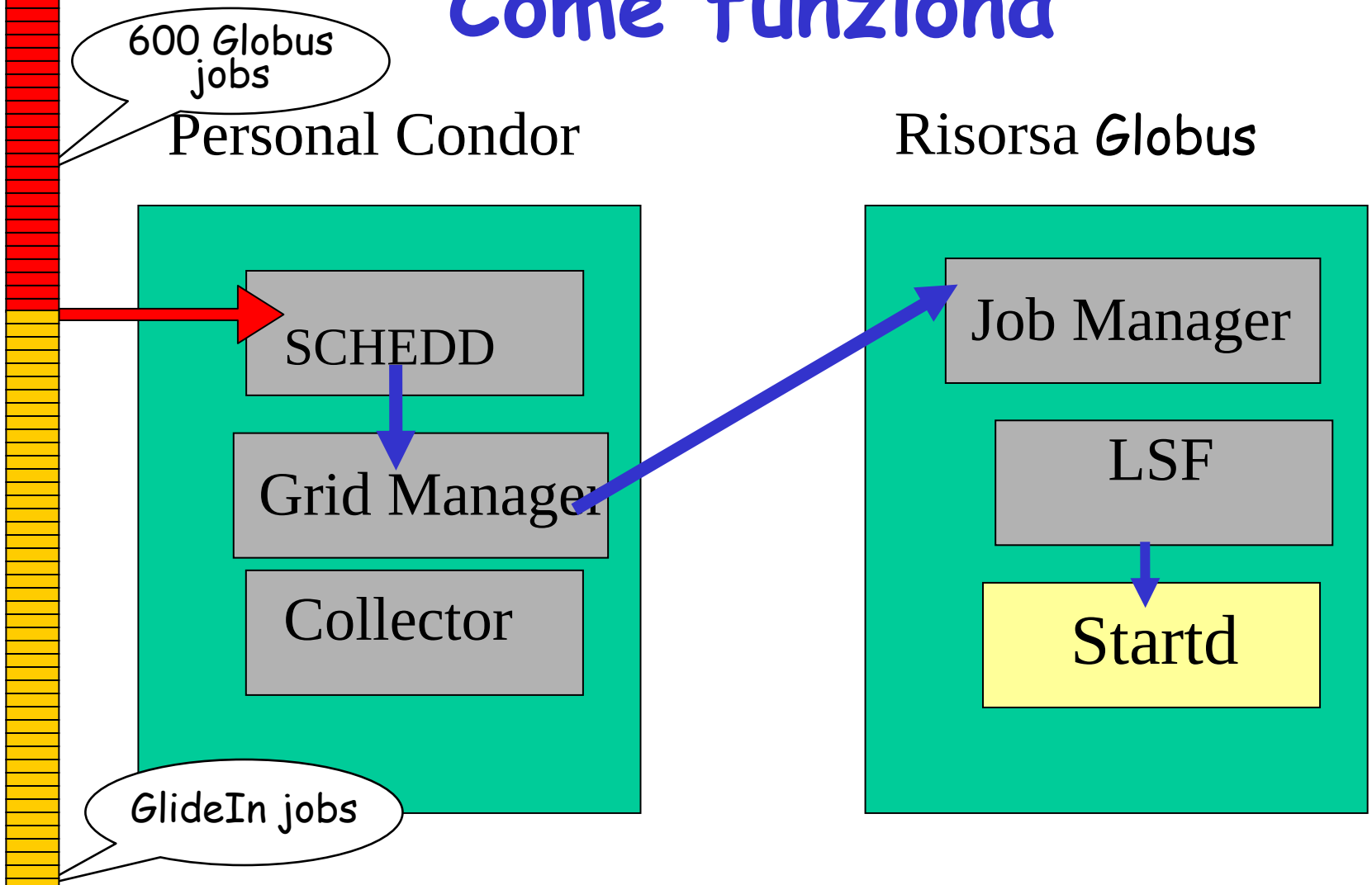


# Come funziona

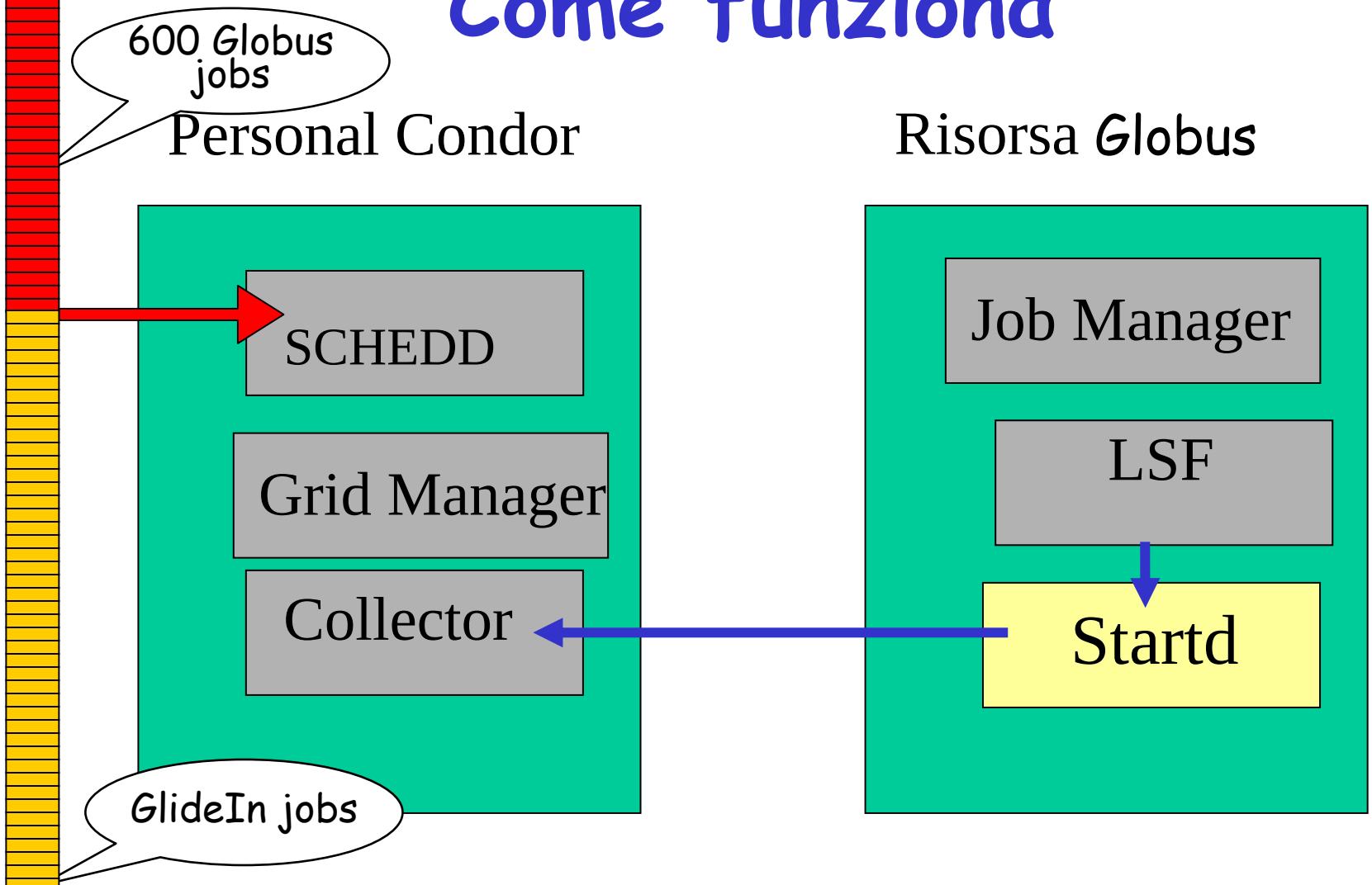



http://www.cs.wisc.edu/condor

# Come funziona

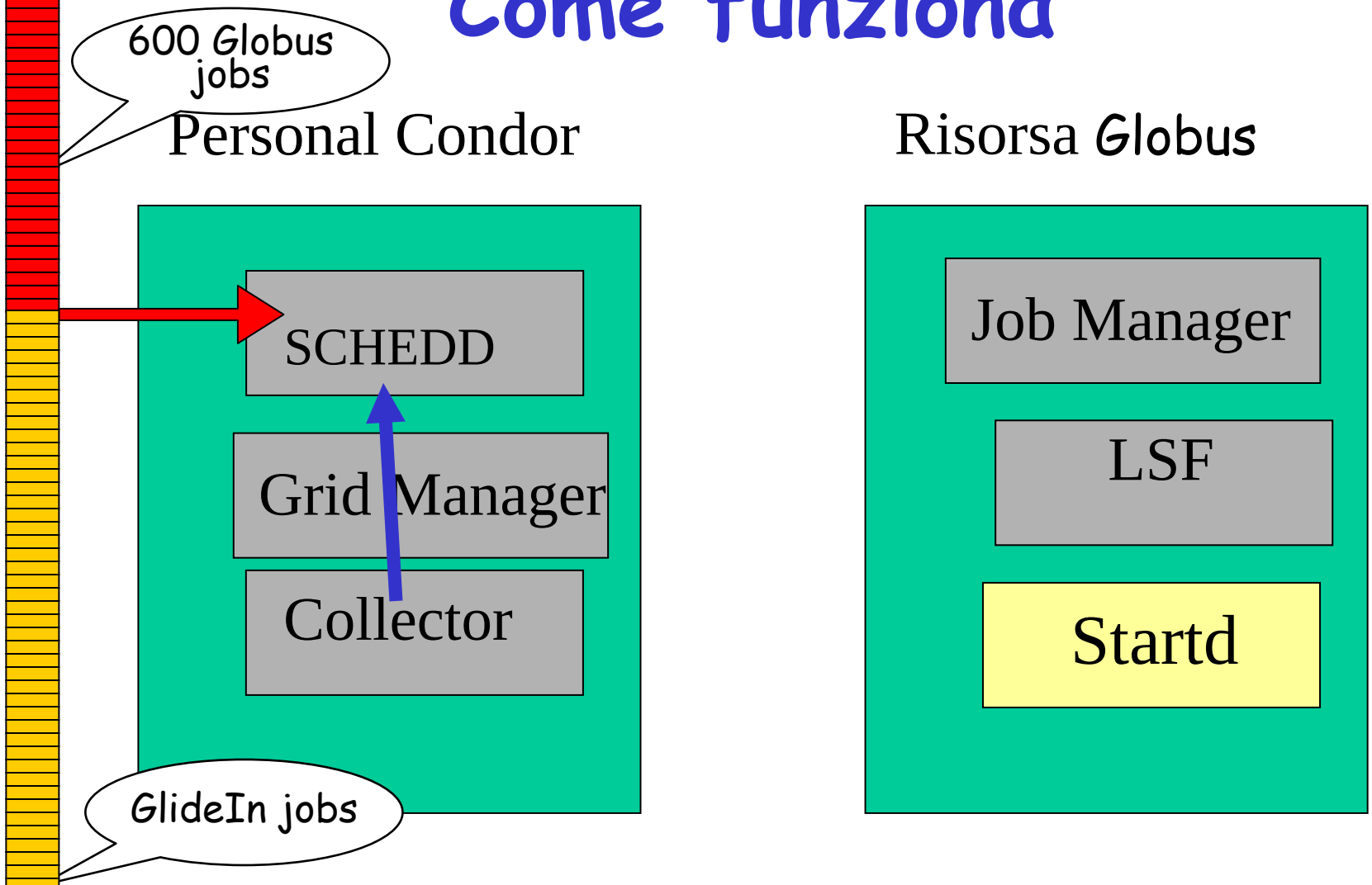


# Come funziona

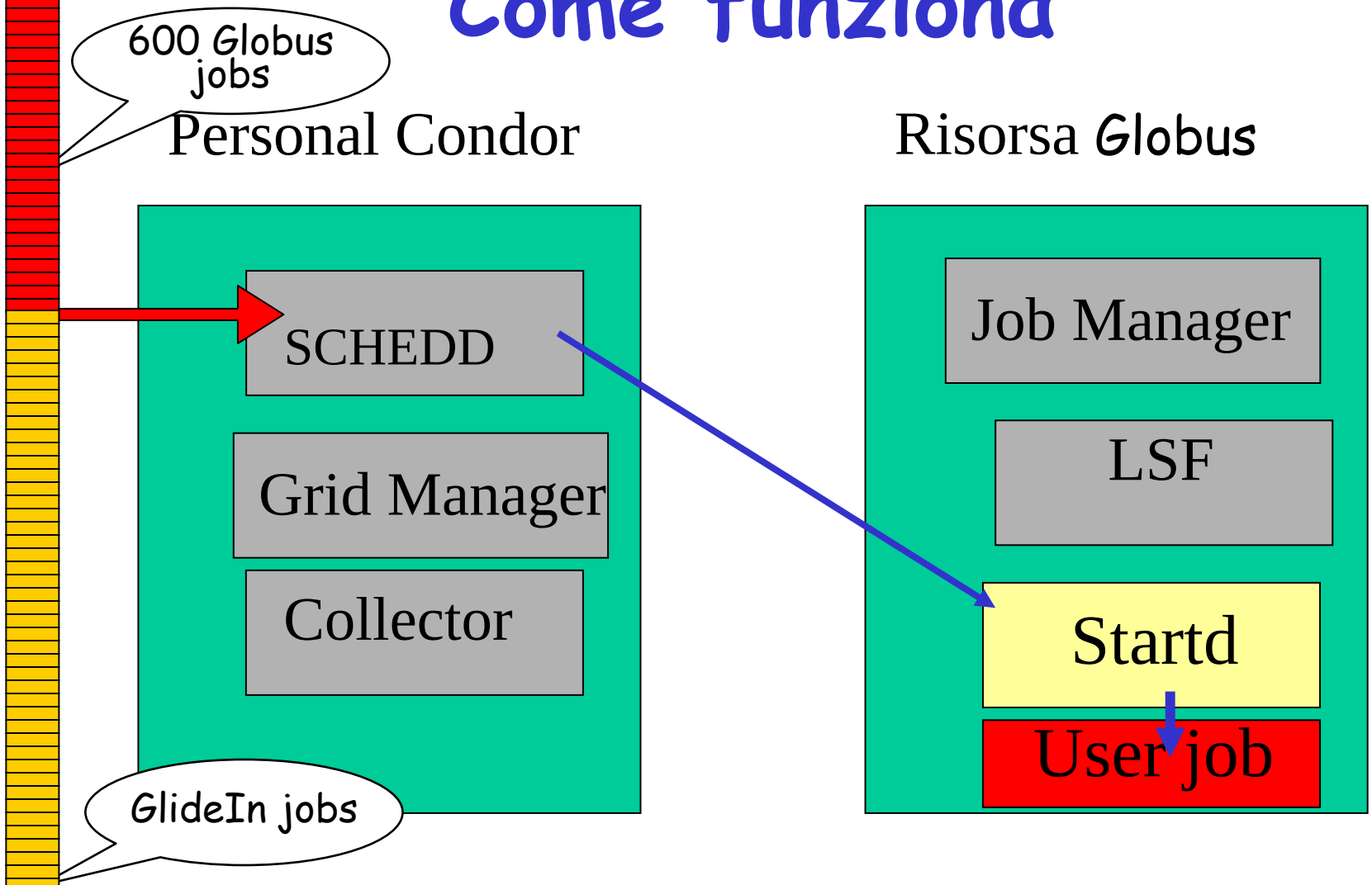



http://www.cs.wisc.edu/condor

# Come funziona



# Come funziona




http://www.cs.wisc.edu/condor

# GlideIn

- Cosa succede quando una risorsa Globus termina il Job Glide-In?
  - La risorsa scompare dal pool e i job di Freida verranno schedulati su un altro calcolatore
  - I Job dello standard universe saranno ripristinati dal loro ultimo checkpoint
- Cosa succede se completo tutti i miei Job prima che un GlideIn job venga eseguito?
  - Se un daemon GlideIn non è matchato con un Job entro 10 minuti dall'esecuzione esso termina liberando la risorsa